

How to secure your Apache Camel deployment

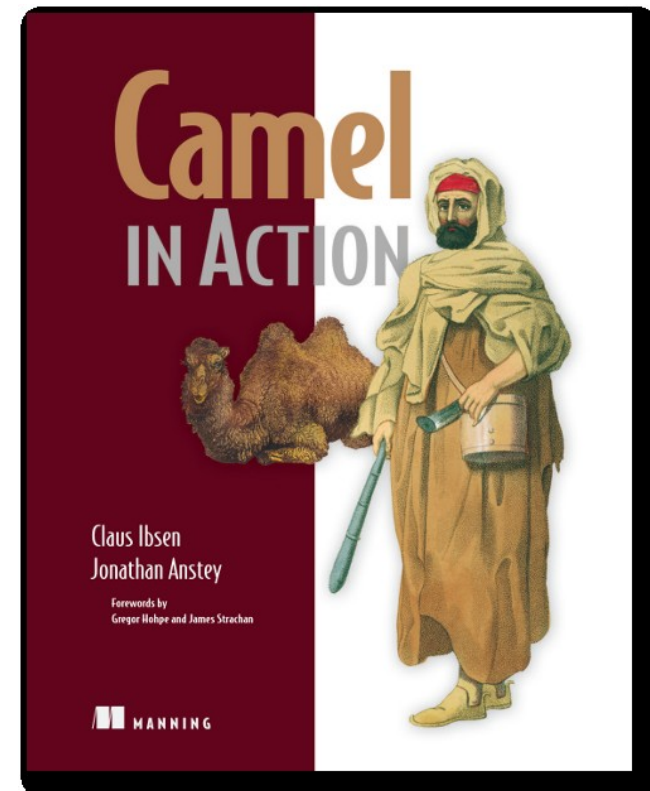
Jonathan Anstey
Principal Engineer
FuseSource



Security B-Sides
2011

Your Presenter is: Jonathan Anstey

- Principal Software Engineer at FuseSource
<http://fusesource.com>
- Apache Camel PMC member, Apache ActiveMQ committer
- Co-author of Camel in Action
- Contact
 - Twitter: @jon_anstey
 - Blog: <http://janstey.blogspot.com>
 - Email: jon@fusesource.com



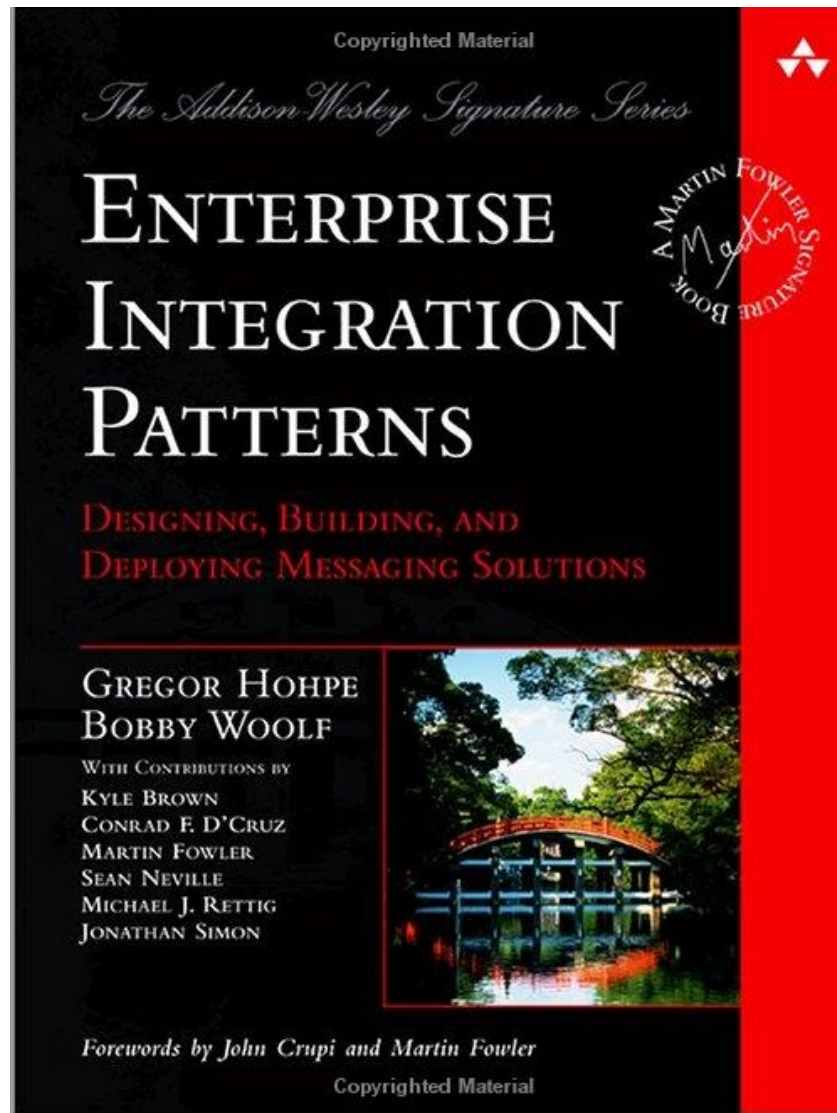
Agenda

1. Intro to Apache Camel
2. Security options in Camel

Why is integration hard?

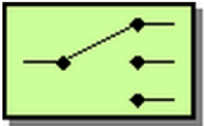

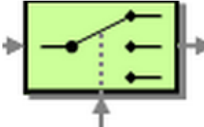
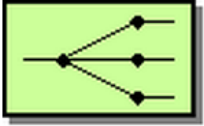
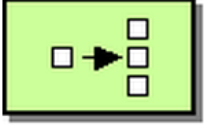

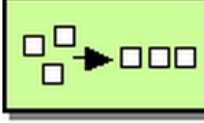
- Multiple **applications** talking over multiple **transports** on multiple **platforms** = PAIN!
- Pain mostly due to
 1. Coming up with good solutions to messaging problems
 2. Coding against specific APIs and transports for each integration point

Enterprise Integration Patterns to the rescue!



Some of the patterns...

Message Routing

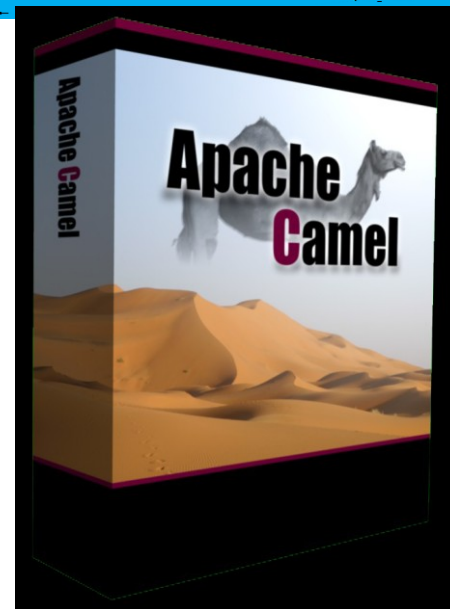
	Content Based Router	How do we handle a situation where the implementation of a single logical function (e.g., inventory check) is spread across multiple physical systems?
	Message Filter	How can a component avoid receiving uninteresting messages?
	Dynamic Router	How can you avoid the dependency of the router on all possible destinations while maintaining its efficiency?
	Recipient List	How do we route a message to a list of (static or dynamically) specified recipients?
	Splitter	How can we process a message if it contains multiple elements, each of which may have to be processed in a different way?
	Aggregator	How do we combine the results of individual, but related messages so that they can be processed as a whole?
	Resequencer	How can we get a stream of related but out-of-sequence messages back into the correct order?



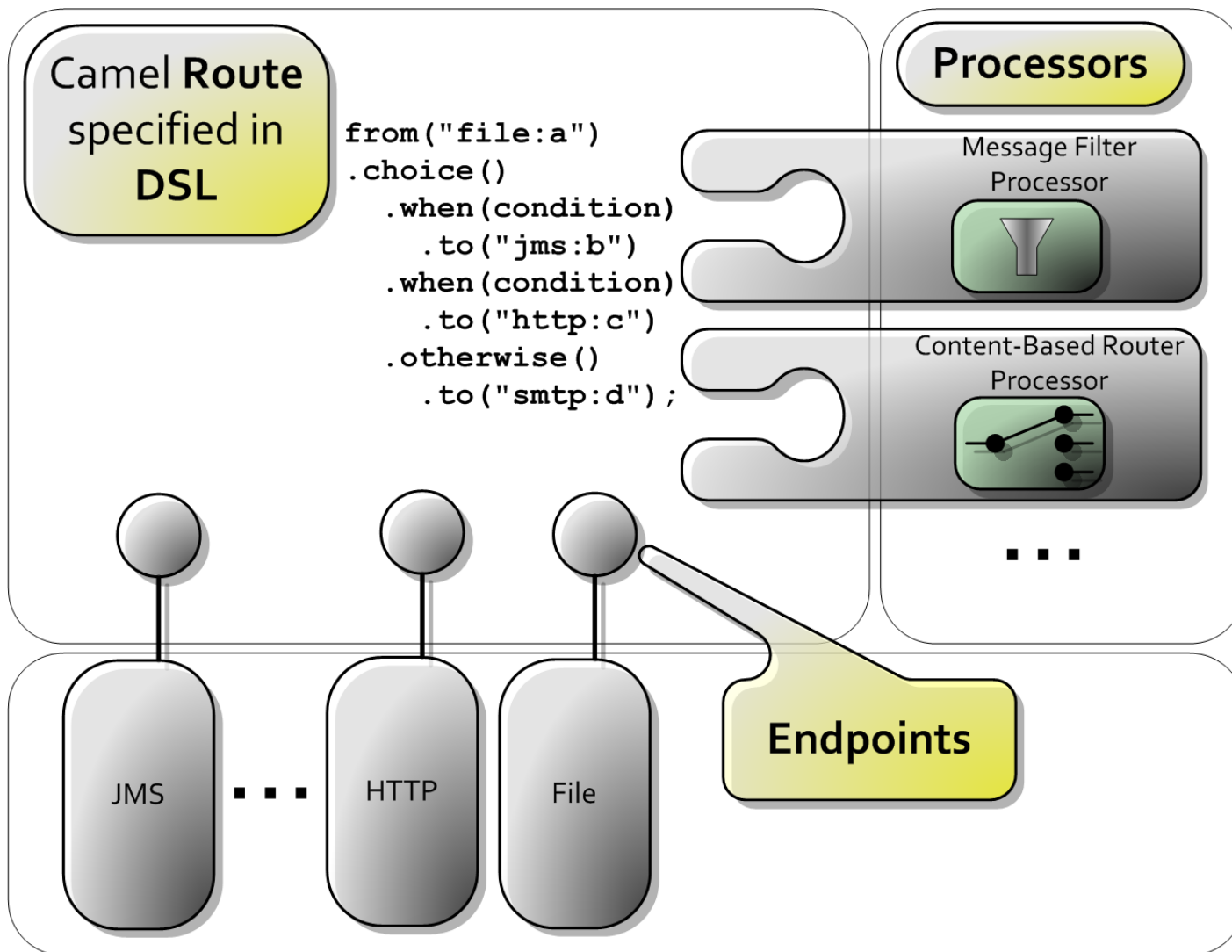
But I still have to code the EIPs and connect to all those transports and APIs...

Introducing Camel

- Apache Camel is an open source integration framework that focuses on making integration easier by having:
 - Implementations of the Enterprise Integration Patterns (EIPs)
 - Connectivity to many transports and APIs
 - Easy to use Domain Specific Language (DSL) to wire EIPs and transports together

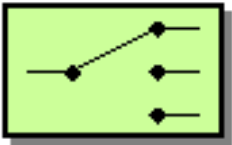
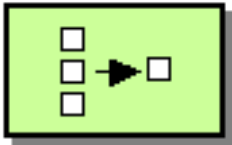
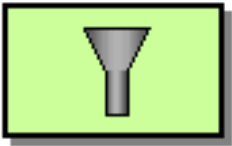
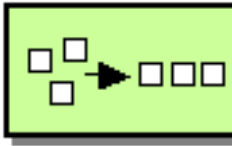
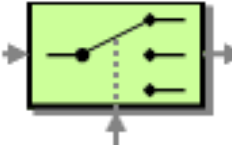
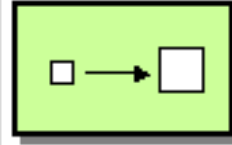
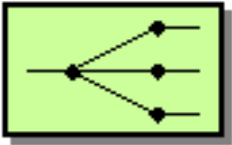
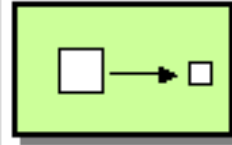
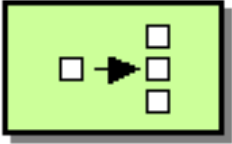
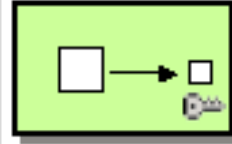


Camel's Main Concepts



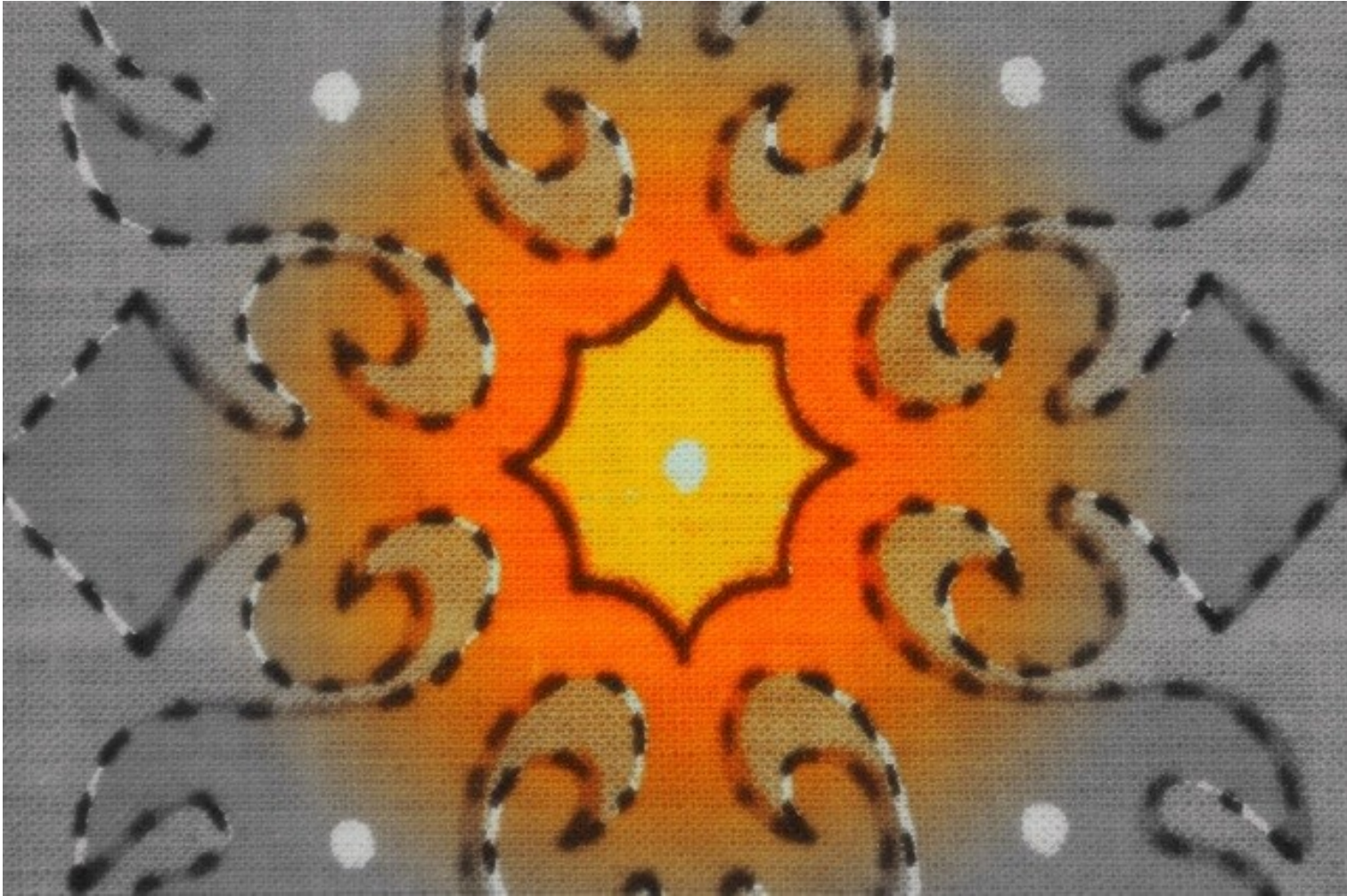
Camel has implementations of the EIPs

- 50+ EIP patterns

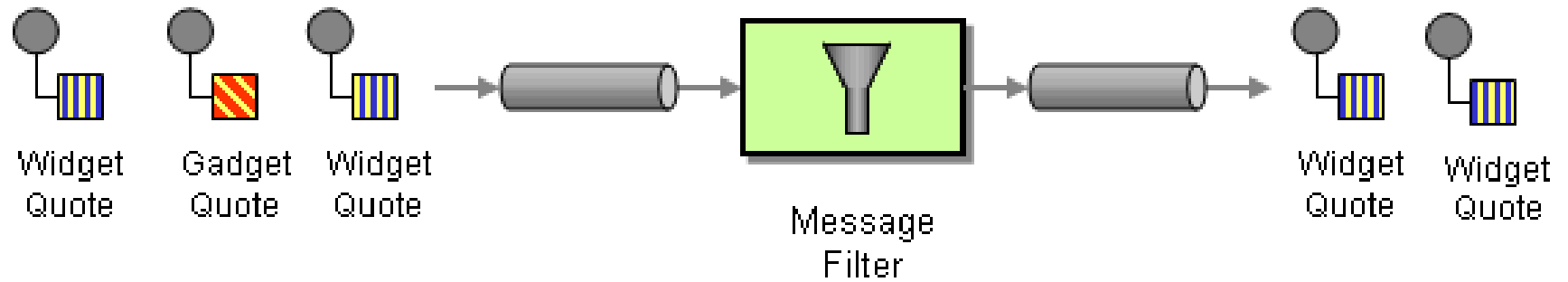
	Content Based Router		Aggregator
	Message Filter		Resequencer
	Dynamic Router		Content Enricher
	Recipient List		Content Filter
	Splitter		Claim Check

<http://camel.apache.org/enterprise-integration-patterns.html>

Let's take a look at a pattern



Message Filter



Message Filter – Java DSL

```
import org.apache.camel.builder.RouteBuilder;

public class FilterRoute extends RouteBuilder {

    public void configure() throws Exception {
        from("activemq:quote")
            .filter().xpath("/quote/product = 'widget'")
            .to("wmq:quote");
    }
}
```

Message Filter – Spring XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>
  <camelContext ...>
    <route>
      <from uri="activemq:quote"/>
      <filter>
        <xpath>/quote/product = 'widget'</xpath>
        <to uri="wmq:quote"/>
      </filter>
    </route>
  </camelContext>
</beans>
```

URIs select and configure the component

- “activemq:quote” will select the ActiveMQ component and use a queue named “quote”

There are many components

- 80+ Components

activemq	crypto	flatpack	irc	ldap
activemq-journal	cxfrs	freemarker	javaspace	mail/imap/pop3
amqp	cxfrs	ftp/https/sftp	jbi	mina
atom	dataset	gae	jcr	mock
bean	direct	hdfs	jdbc	msv
bean validation	esper	hibernate	jetty	nagios
browse	event	hl7	jms	netty
cache	exec	http	jpa	nmr
cometd	file	ibatis	jt/400	printer

<http://camel.apache.org/components.html>

There are many components

- 80+ Components

properties	scalate	stream	xslt	
quartz	seda	string-template	ejb	
quickfix	servlet	test		
ref	smooks	timer		
restlet	smpp	validation		
rmi	snmp	velocity		
rnc	spring-integration	vm		
rng	spring-security	xmpp		
rss	sql	xquery		

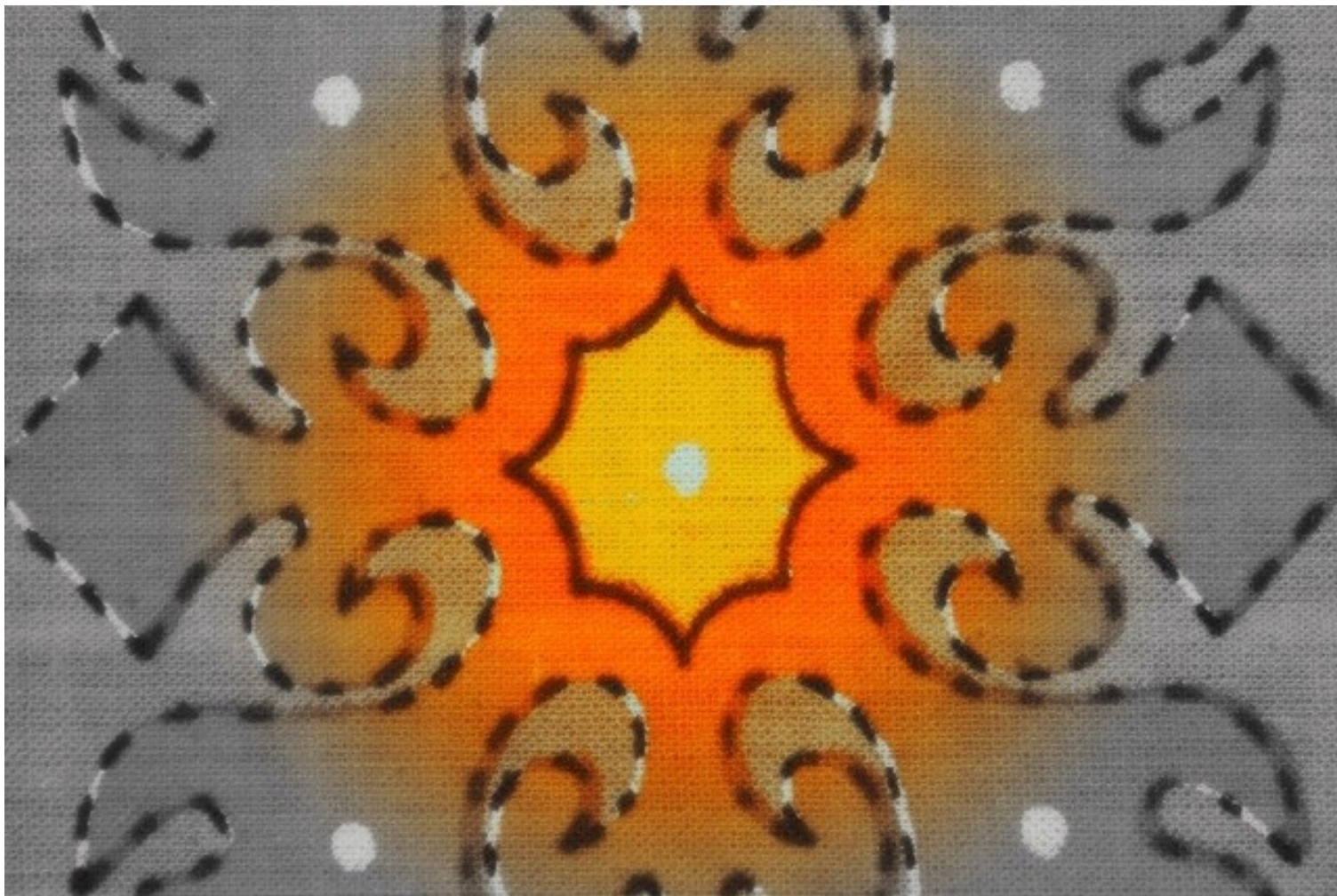
<http://camel.apache.org/components.html>

Predicates & Expressions

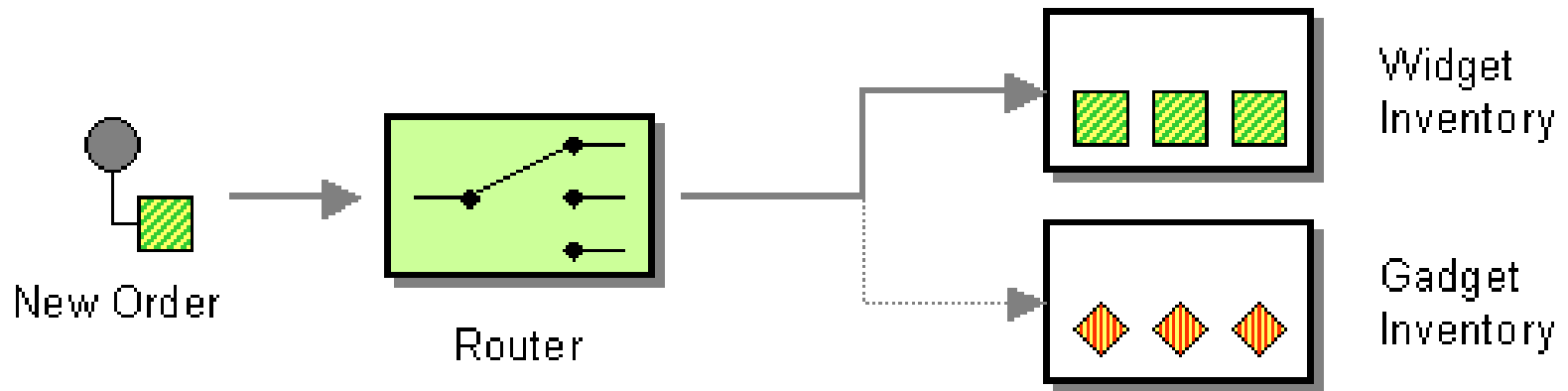
BeanShell	PHP
EL	Python
Groovy	Ruby
JavaScript	Simple
JSR 223	SQL
OGNL	XPath
MVEL	XQuery

<http://camel.apache.org/languages.html>

More patterns!

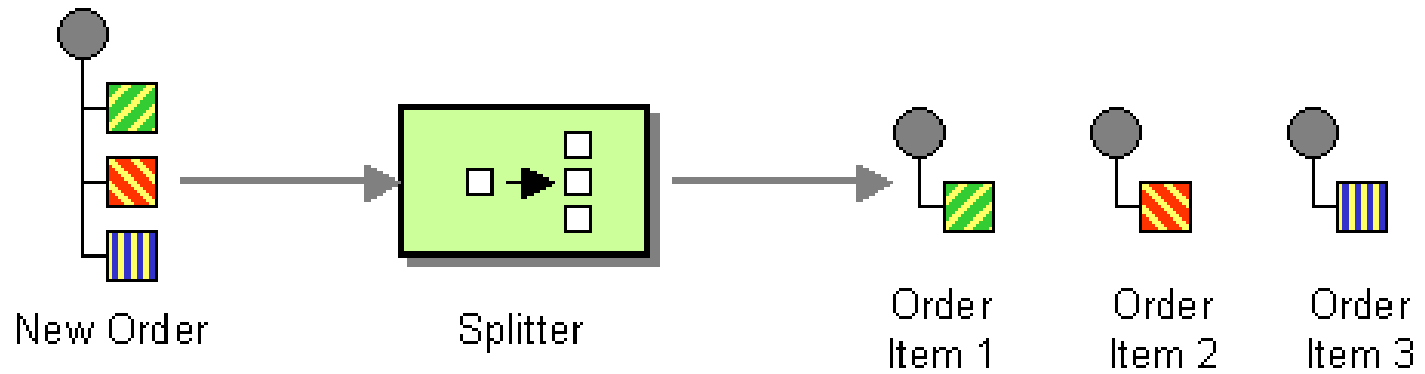


Content-Based Router



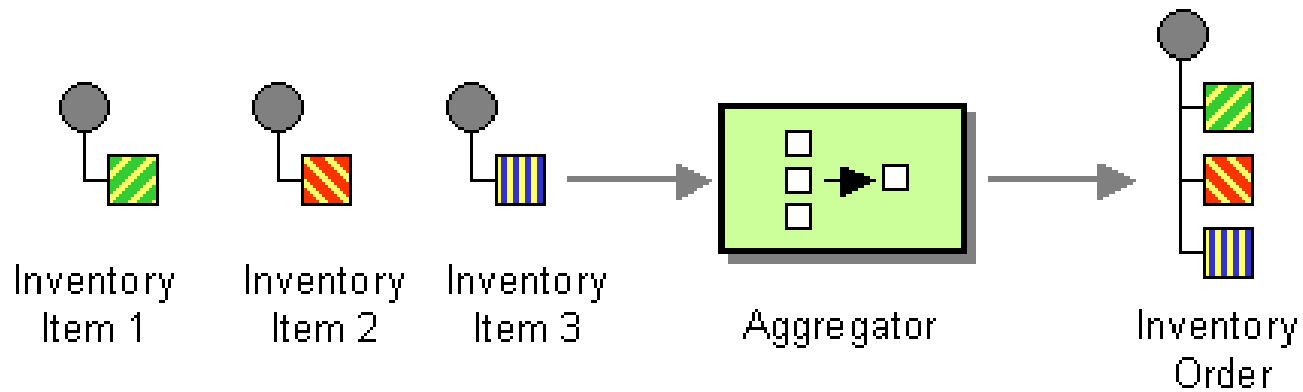
```
from("activemq:NewOrders")  
  .choice()  
    .when().xpath("/order/product = 'widget'")  
      .to("activemq:Orders.Widgets")  
    .otherwise()  
      .to("activemq:Orders.Gadgets");
```

Splitter



```
from("file:orders")  
    .split(body().tokenize("\n"))  
    .to("activemq:Order.Items");
```

Aggregator




```
from("activemq:Inventory.Items")  
  .aggregate(xpath("/order/@id"))  
  .to("activemq:Inventory.Order");
```

Message payload handling

- Camel supports any type of payload
- **TypeConverters** automatically convert common Java types
 - DOM to String, File to InputStream, Stream to byte[], etc
- **Dataformats** are used to explicitly marshall and unmarshall to specific data formats

Data formats

bindy	protobuf
castor	serialization
csv	soap
crypto	tidy markup
flatpack	xml beans
gzip	xml security
hl7	xstream
jaxb	zip
json	dozer

<http://camel.apache.org/data-format.html>

Data format example

```
from("activemq:QueueWithJavaObjects")  
  .marshal().jaxb()  
  .to("wmq:QueueWithXmlMessages");
```

Running Camel



Running Camel

- Java Application

```
CamelContext context = new DefaultCamelContext();  
context.addRoutes(new MyRouteBuilder());  
context.start();
```

- Spring Application

```
<camelContext>  
  <package>com.acme.myroutebuilders</package>  
</camelContext>
```

Security options in Camel

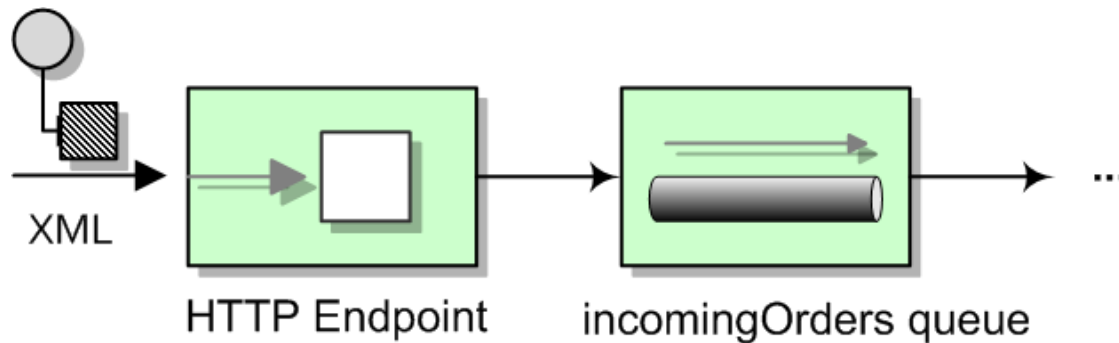


Security options in Camel

- **Endpoint Security**
 - Security options handled by transport or API
- **Payload Security**
 - Encryption/decryption or payload using data formats
- **Route Security**
 - Authentication and authorization within a route's flow
- **Configuration Security**
 - Encrypting/decrypting configuration files

Simple example

- JAX-WS web service accepts order and dumps it to a JMS queue for processing



Simple example

- Camel route that consumes messages from the HTTP endpoint

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="cxf:bean:orderEndpoint" />
    <to uri="jms:incomingOrders" />
    <transform>
      <constant>OK</constant>
    </transform>
  </route>
</camelContext>
```

Send back "OK" response

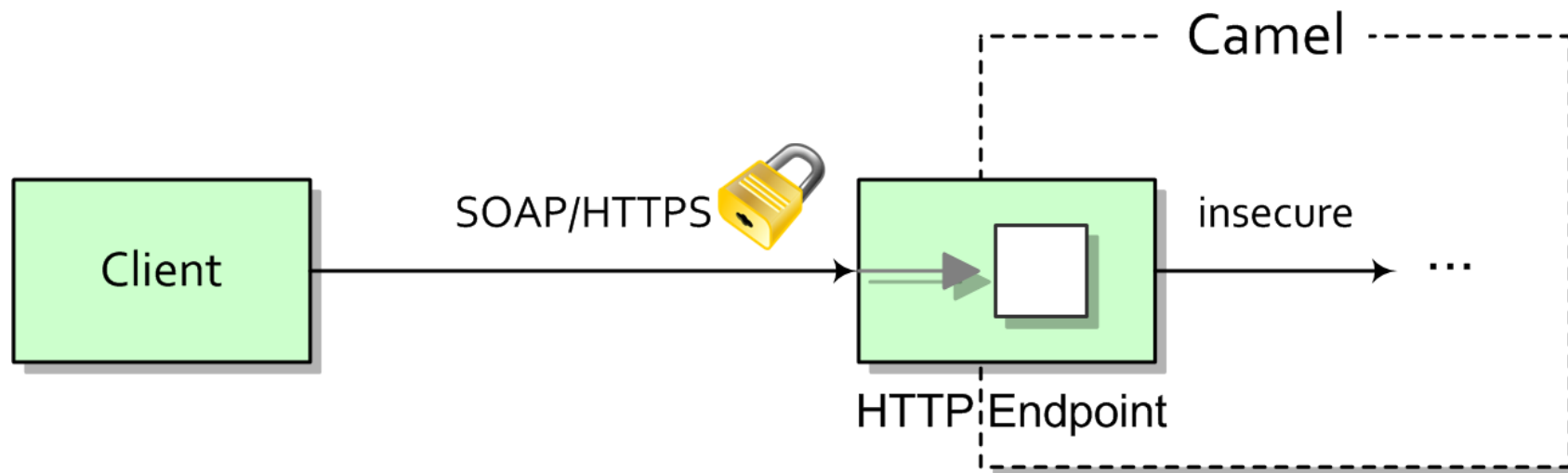
- Implemented using camel-cxf component

```
<cxf:cxfEndpoint id="orderEndpoint"
  address="http://localhost:9000/order/"
  serviceClass="camelinaction.order.OrderEndpoint"
  wsdlURL="wsdl/order.wsdl"/>
```

Endpoint Security...

Endpoint Security

- Useful for publishing secure services for external consumption or integrating with secure services
- Securing the pipe rather than anything within the Camel route
- Varies based on transport type



Setting up SSL/TLS on JAX-WS Web Service

- First switch to use HTTPS in address...

```
<cxf:cxfEndpoint id="orderEndpoint"  
  address="https://localhost:9000/order/"  
  serviceClass="camel.inaction.order.OrderEndpoint"  
  wsdlURL="wsdl/order.wsdl" />
```

Setting up SSL/TLS on JAX-WS Web Service

- Next configure the transport to use mutually authenticated SSL... whew!

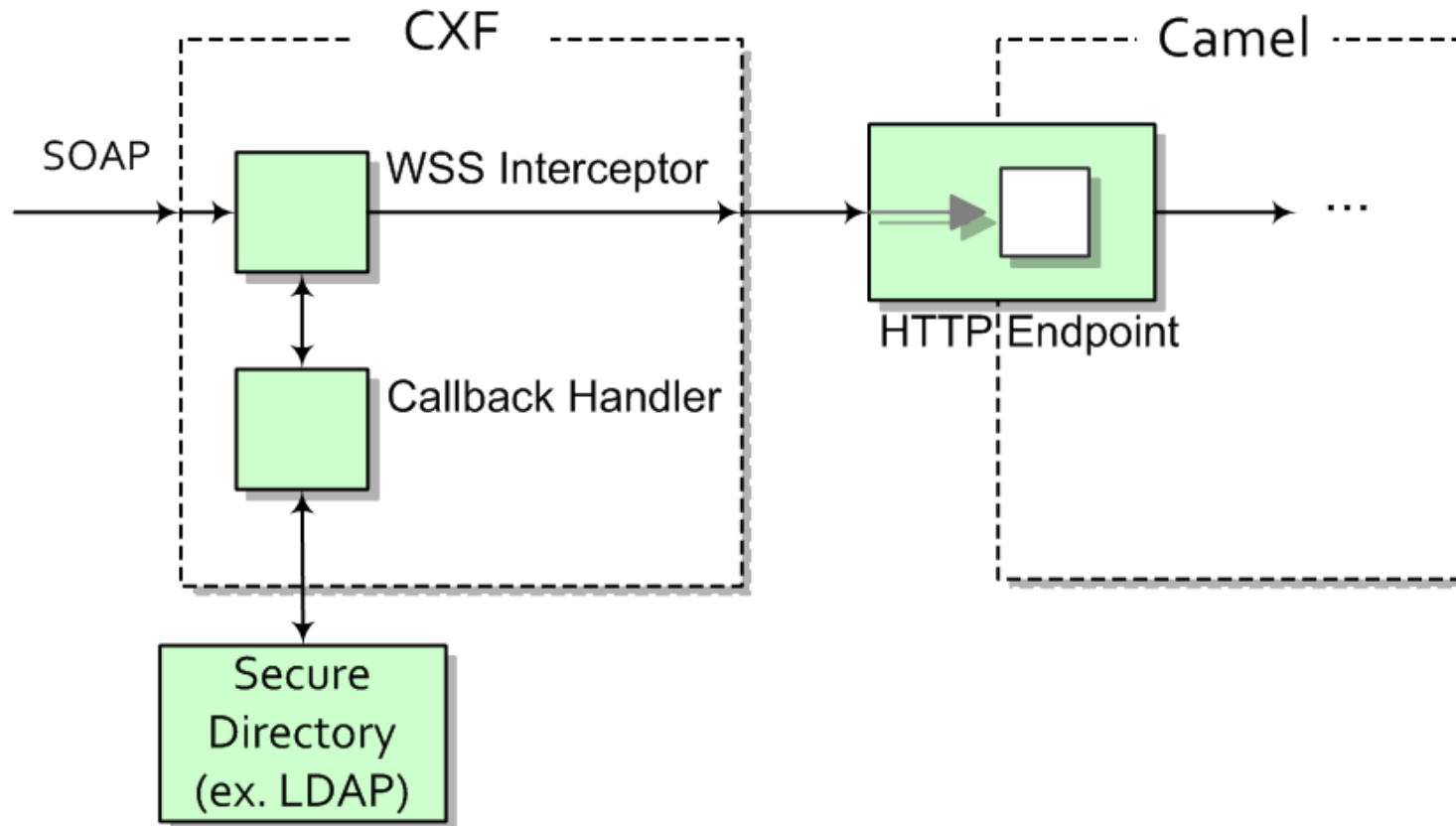
Same port as web service

```
<httpj:engine-factory bus="cxf">
  <httpj:engine port="9000">
    <httpj:tlsServerParameters>
      <sec:keyManagers keyPassword="password">
        <sec:keyStore type="JKS" password="password"
          file="certs/keystore.jks"/>
      </sec:keyManagers>
      <sec:trustManagers>
        <sec:keyStore type="JKS" password="password"
          file="certs/truststore.jks"/>
      </sec:trustManagers>
      <sec:clientAuthentication want="true" required="true" />
    </httpj:tlsServerParameters>
  </httpj:engine>
</httpj:engine-factory>
```

Generated by Java keytool; Server and client need these

Adding Credentials to a JAX-WS Web Service

- Using WS-Security to check credentials
- Interceptor inserted before endpoint to check credentials



Adding Credentials to a JAX-WS Web Service


- Adding WS-Security interceptor to endpoint

```
<cxf:cxfEndpoint id="orderEndpoint"
  address="https://localhost:9000/order/"
  serviceClass="camelinaction.order.OrderEndpoint"
  wsdlURL="wsdl/order.wsdl">
  <cxf:inInterceptors>
    <bean class="org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor">
      <constructor-arg>
        <map>
          <entry key="action" value="UsernameToken Timestamp" />
          <entry key="passwordType" value="PasswordText" />
          <entry key="passwordCallbackClass"
            value="com.mycompany.UTPasswordCallback" />
        </map>
      </constructor-arg>
    </bean>
  </cxf:inInterceptors>
</cxf:cxfEndpoint>
```

Interceptor



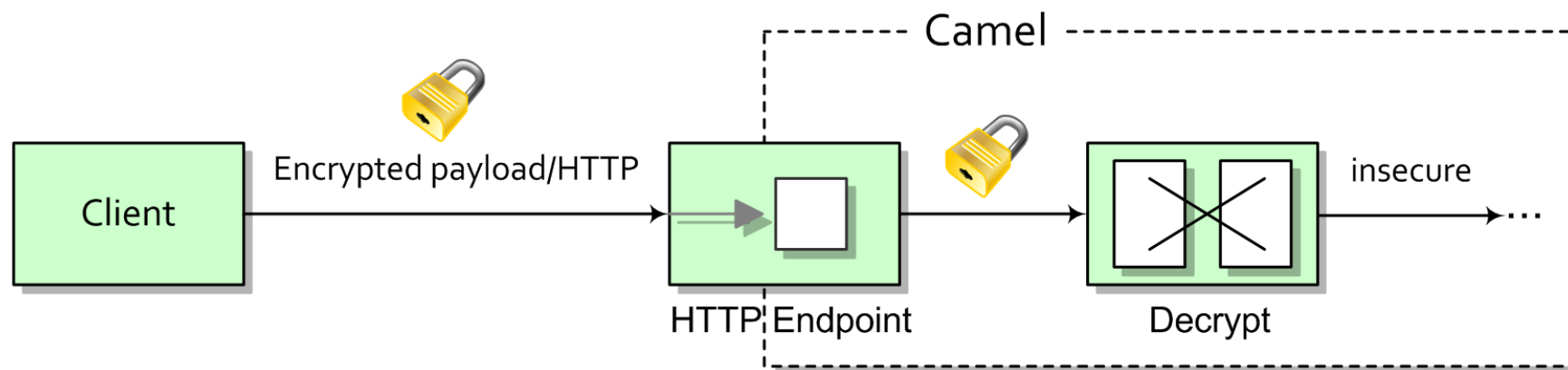
*Callback class
checks password*



Payload Security...

Payload Security

- Useful for communicating over insecure endpoints
- camel-crypto provides dataformat for encrypting and decrypting payloads
- Can use any JCE algorithm



Payload Security

- Encrypt before sending to an endpoint using "marshal"

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <dataFormats>
    <crypto id="des" algorithm="DES" keyRef="desKey" />
  </dataFormats>

  <route>
    <marshal ref="des" />
    <to uri="cxf:bean:orderEndpoint" />
  </route>
```

- Decrypt on the other side using "unmarshal"

```
<route>
  <from uri="cxf:bean:orderEndpoint" />
  <unmarshal ref="des" />
  <to uri="jms:incomingOrders" />
  <transform>
    <constant>OK</constant>
  </transform>
</route>
```

Route Security...

Route Security

- Provides authorization within Camel routes
- Can handle auth errors with Camel's error handlers rather than coding your own
- Handle auth in one place rather than for every transport
- Implemented by either Spring Security or Apache Shiro

Route Security

- Using the camel-spring-security component you first have to set up Spring Security
- Users can come from lots of sources: LDAP, JAAS, JDBC, etc
 - Here we just hardcode one user in the "customer" role

```
<spring-security:authentication-manager alias="authenticationManager">  
  <spring-security:authentication-provider user-service-ref="userDetailsService"/>  
</spring-security:authentication-manager>
```

```
<spring-security:user-service id="userDetailsService">  
  <spring-security:user name="bsides" password="stjohns" authorities="ROLE_CUST"/>  
</spring-security:user-service>
```

```
<bean id="accessDecisionManager" class="org.springframework.security.access.vote.AffirmativeBased">  
  <property name="allowIfAllAbstainDecisions" value="true"/>  
  <property name="decisionVoters">  
    <list>  
      <bean class="org.springframework.security.access.vote.RoleVoter"/>  
    </list>  
  </property>  
</bean>
```

Route Security

- Next step is to create a policy in Camel to authorize users in the "customer" role

```
<authorizationPolicy id="customer" access="ROLE_CUST"  
    authenticationManager="authenticationManager"  
    accessDecisionManager="accessDecisionManager"  
    xmlns="http://camel.apache.org/schema/spring-security"/>
```

```
<camelContext xmlns="http://camel.apache.org/schema/spring">  
  <route>  
    <from uri="cxf:bean:orderEndpoint" />  
    <policy ref="customer">  
      <to uri="jms:incomingOrders" />  
      <transform>  
        <constant>OK</constant>  
      </transform>  
    </policy>  
  </route>
```

Create a protected block of route using an auth policy

Route Security

- If a client was not authorized to send an order (not in the "customer" role) an error message will be sent back

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <onException>
    <exception>org.apache.camel.CamelAuthorizationException</exception>
    <handled>
      <constant>true</constant>
    </handled>
    <transform>
      <simple>Access denied by security policy ${exception.policyId}</simple>
    </transform>
  </onException>
</camelContext>
```


Configuration Security...

Configuration Security

- Say we instead sent new orders to an FTP server, which requires a password in the URI

```
<route>
  <from uri="cxf:bean:orderEndpoint" />
  <to uri="ftp://myuser@remotehost:21/neworders?password={{ftp.password}}"/>
  <transform>
    <constant>OK</constant>
  </transform>
</route>
```

- Properties file has one plaintext entry
 - ftp.password=mysecretpassword

Configuration Security

- Use camel-jasypt command line tool to encrypt properties
 - ftp.password=ENC(bsW9uV37gQ0QHfu7KO03Ww==)
- Jasypt properties parser can understand this encrypted file

```
<bean id="jasypt" class="org.apache.camel.component.jasypt.JasyptPropertiesParser">  
  <property name="password" value="sysenv:CAMEL_ENCRYPTION_PASSWORD" />  
</bean>
```

```
<camelContext xmlns="http://camel.apache.org/schema/spring">  
  <propertyPlaceholder id="properties" propertiesParserRef="jasypt"  
    location="classpath:myproperties.properties" />
```

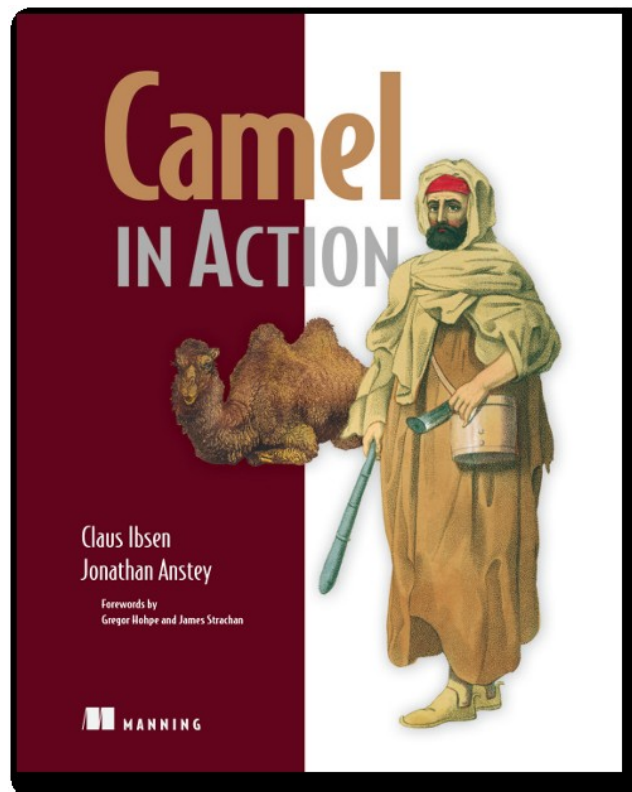
```
<route>  
  <from uri="cxf:bean:orderEndpoint?synchronous=true" />  
  <to uri="ftp://myuser@remotehost:21/neworders?password={{ftp.password}}" />  
  <transform>  
    <constant>OK</constant>  
  </transform>  
</route>
```

Useful links

- Apache Camel web site – <http://camel.apache.org>
- Camel security docs - <http://camel.apache.org/security.html>
- Camel in Action book – <http://manning.com/camelinaction>

Offer from Manning Publications!

- Order any book from <http://manning.com> and use the **bsides40java** code for 40% off



Any Questions?