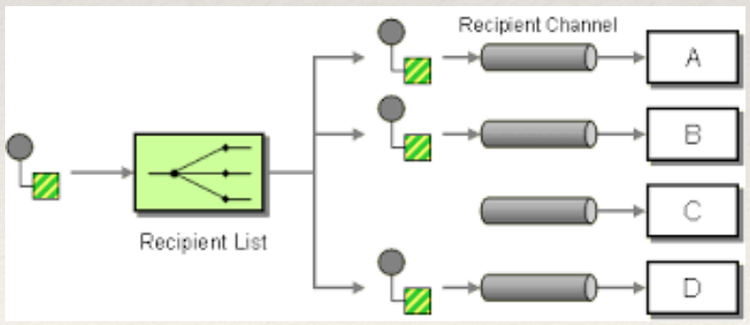
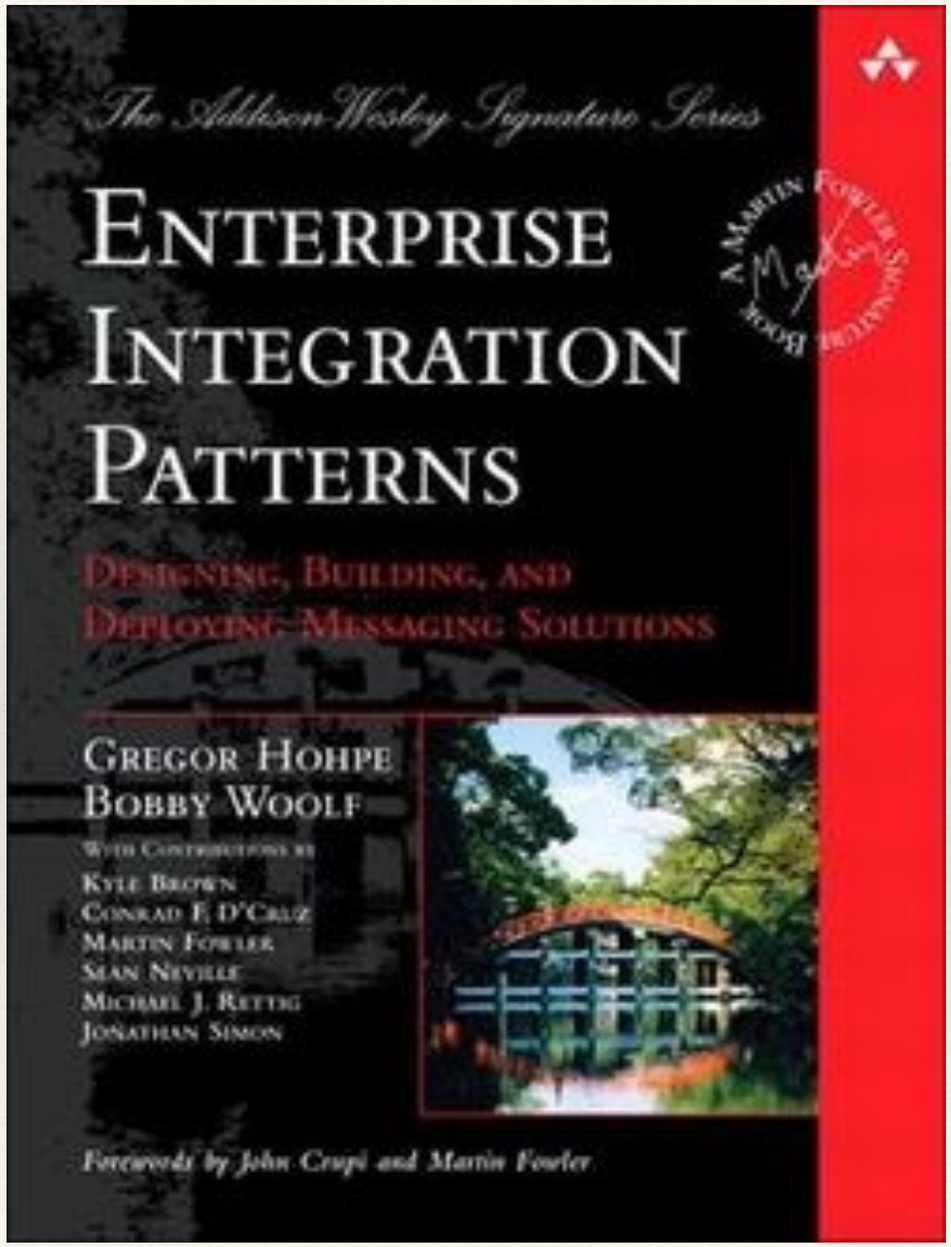


The Apache Software Foundation
http://www.apache.org/

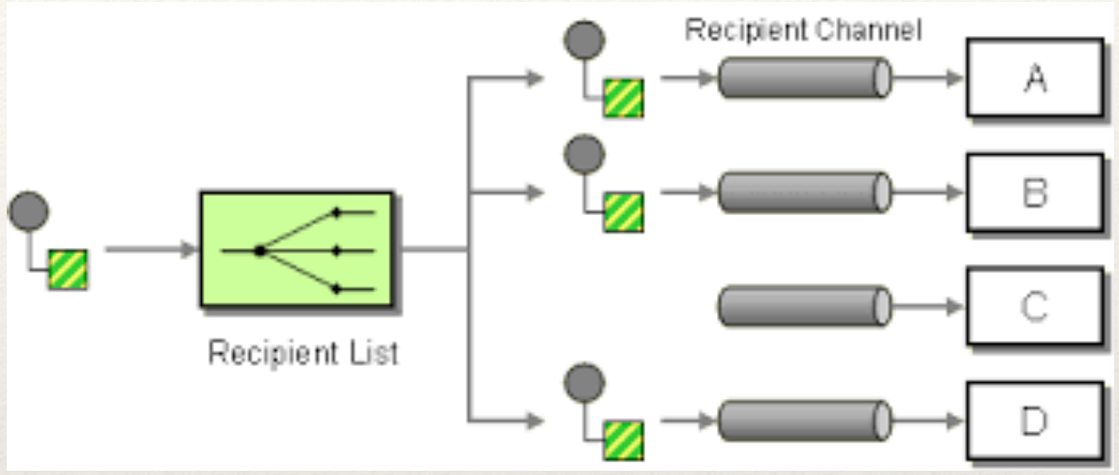
Background

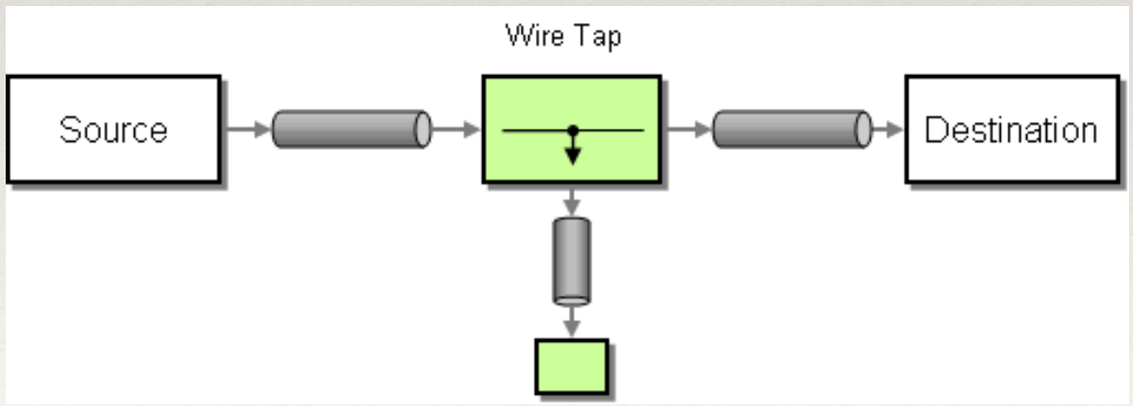
- In 2003 Gregor Hohpe and Bobby Woolf wrote “Enterprise Integration Patterns” (EIP)
- This describes 65 design patterns for the use of enterprise integration and message oriented middleware in the form of a pattern language

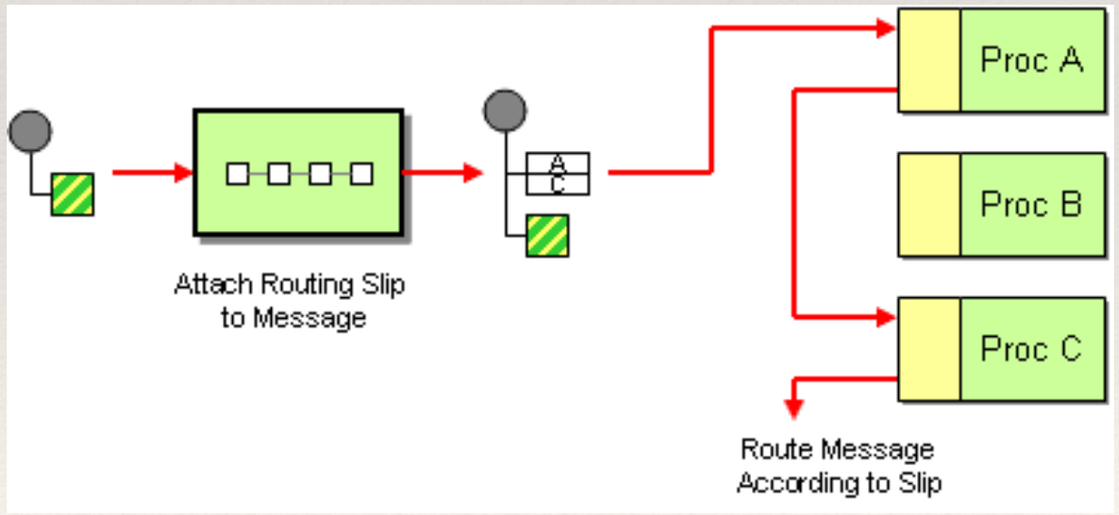


Background

- The patterns are designed to solve common integration problem
- Often represented as "Gregorgrams"







What is Camel

- Camel is an open source integration framework
- It implements many of the patterns described in EIP
- It's main goal is to take care of the integration plumbing, allowing you to focus on solving business problems
- First released in 2007

The Sales Pitch

Camel empowers you to define routing and mediation rules in a variety of domain-specific languages, including a Java-based Fluent API, Spring or Blueprint XML Configuration files, and a Scala DSL. This means you get smart completion of routing rules in your IDE, whether in a Java, Scala or XML editor.

Apache Camel uses URIs to work directly with any kind of Transport or messaging model such as HTTP, ActiveMQ, JMS, JBI, SCA, MINA or CXF, as well as pluggable Components and Data Format options. Apache Camel is a small library with minimal dependencies for easy embedding in any Java application. Apache Camel lets you work with the same API regardless which kind of Transport is used - so learn the API once and you can interact with all the Components provided out-of-box.

Apache Camel provides support for Bean Binding and seamless integration with popular frameworks such as Spring, Blueprint and Guice. Camel also has extensive support for unit testing your routes.

First Example

- Blueprints / Spring XML

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint
    xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
      http://www.osgi.org/xmlns/blueprint/v1.0.0
      http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
      <route>
        <from uri="file:camel//Users/simonvandersluis/camel/FileMover"/>
        <log message="Moving ${file:name} to the output directory"/>
        <to uri="file:camel//Users/simonvandersluis/camel/dump"/>
      </route>
   </camelContext>
</blueprint>
```

- Fluent Java

```
public class FileMover extends RouteBuilder{

  @Override
  public void configure() throws Exception {
    from("file:///Users/simonvandersluis/camel/FileMover")
            .log("Moving ${file:name} to the output directory")
            .to("file:///Users/simonvandersluis/camel/dump");
  }
}
```

First Example

```
from(“ftp://simon@company.com/inbox?password=secret”)
```

Take a message from the URL

```
.log("Moving ${file:name} to the output directory”)
```

Log the header “file:name” to the applications log

```
.to(“file:///Users/simonvandersluis/camel/dump“)
```

Put the message body to the URL

Key Concepts

- Camel Context
- **Routes**
- Exchange
- **Processors**
- **Messages**
  - **Body**
  - **Headers**
  - Attachments

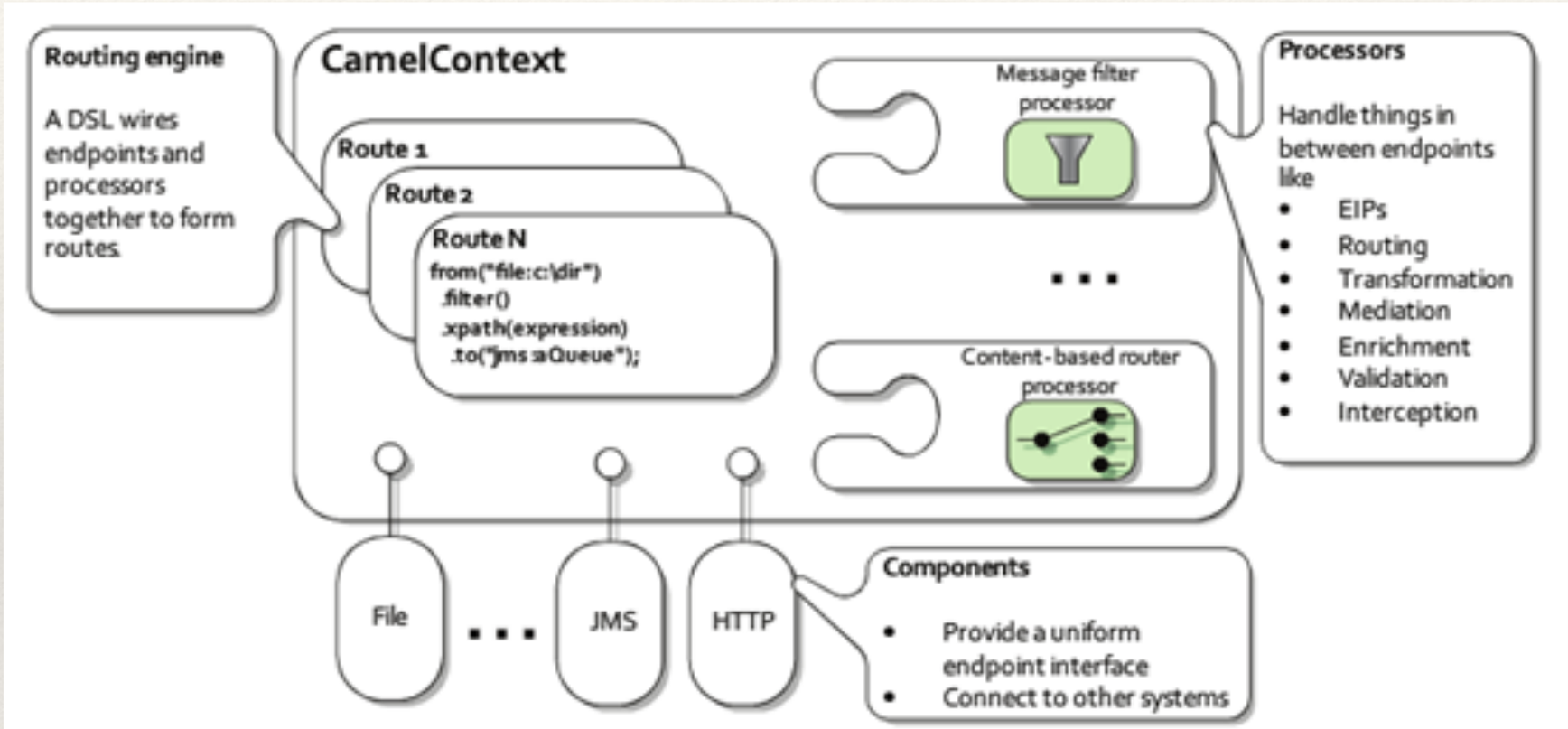


Figure 1.6 **At a high level, Camel is composed of processors, components, and routes. All of these are contained within the `CamelContext`.**

Camel Context

- A CamelContext represents a single Camel routing rule base. You use it in a similar way as you would a Spring ApplicationContext
- In short you need to instantiate one to make Camel work
- Easiest way is using Spring (but there are others)

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
http://camel.apache.org/schema/spring
http://camel.apache.org/schema/spring/camel-spring.xsd">

  <camelContext xmlns="http://camel.apache.org/schema/spring" >
    <package>nz.co.edmi.servicemixdemo</package>
  </camelContext>
</beans>
```

- Using spring to load a camel context means you can take advantage of spring dependency injection for your camel routes

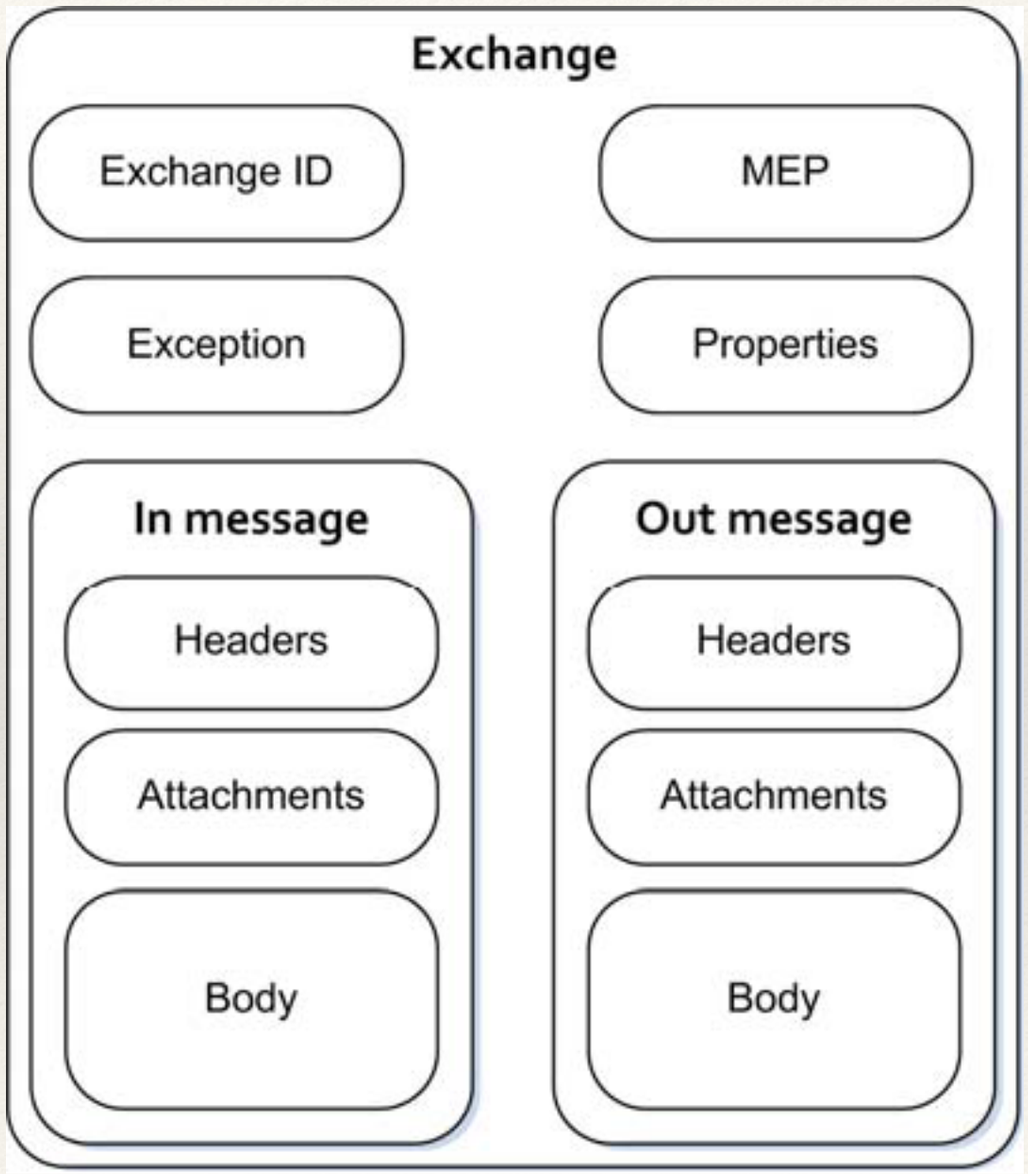
Routes

- The core abstraction in Camel
- Think of these as a chain of processors
- Provide decoupling of clients from servers, producers from consumers
- Defined in a RouteBuilder
- Camel is designed so that routes can flow when read (helped by the fluent api), e.g.

```
from("file:///Users/simonvandersluis/camel/FileMover")
            .log("Moving ${file:name} to the output directory")
            .to("file:///Users/simonvandersluis/camel/dump");
}
```

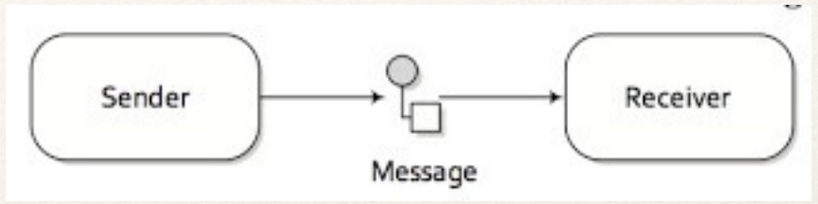
Exchange

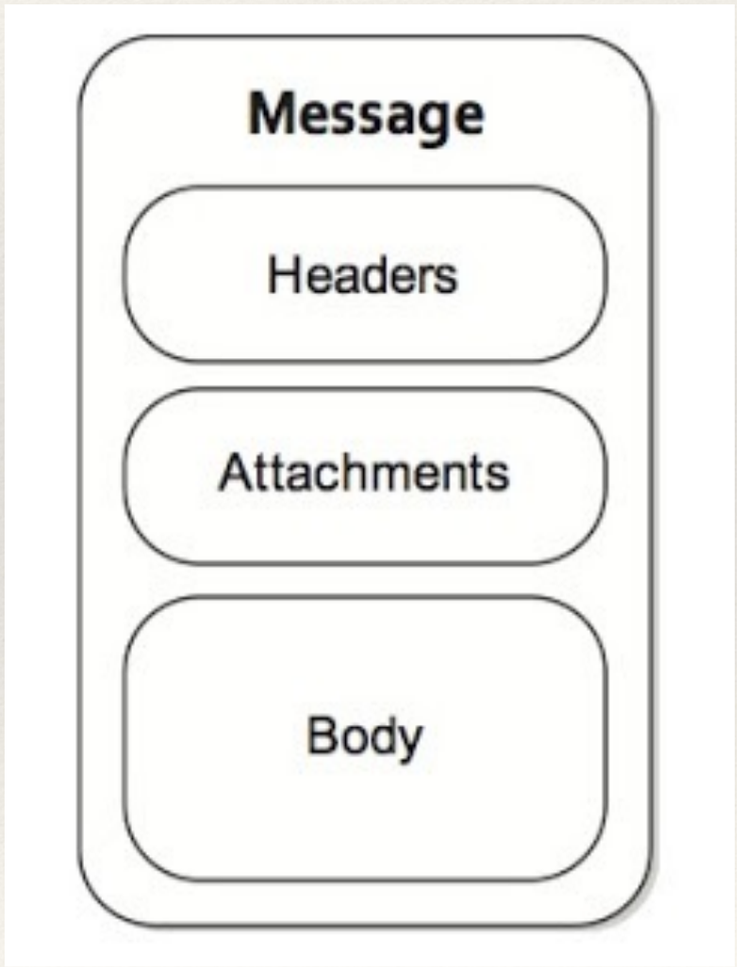
- An Exchange in Camel is the message container during routing
- Can be In-Only, or In-Out
- This will make more sense when we talk about Processors



Messages

- Messages are the entities used by subsystems to communicate with each other.
- They Flow in one direction. Sender to Receiver
- They have a body (payload), headers and optional attachments





Processors

- Used to manipulate messages in a route
- This is where you implement your business specific logic and processing
- Camel has many pre-built processors for you
- Although there is a processor interface you can implement, there are alternative ways to implement processors that have less coupling to camel classes
- code => FileReverser

Exchange In-Only vs In-Out

- The Exchange passed to a processor has an in-message and an out-message
- The out-message start out as null
- If you set the out-message you have an in-out exchange BUT
  - The headers from the in-message are now lost to the next processor in the chain
  - If you don't to loose (or manually copy) the headers, it's common practice to re-use the in-message

Recap + Teaser

- Camel run in a CamelContext, easily loaded by Spring
- Routes **consume** messages **from** an end point, **process** them and **produce** output that can be sent **to** other consumers
- End points are defined by URLs
- We've only seen files being used so far, but camel supports
  - Making / receiving HTTP / REST calls
  - Message queue, eg ActiveMQ
  - Social media eg supports the Facebook API
  - SMTP, pop3, gmail
  - SNMP
  - Solr/Lucene
  - SQL / NoSQL
  - Templating e.g. Velocity
  - lots more: http://camel.apache.org/components.html

Handlers

- These are the alternative to processors
- Implemented as POJOs
- No coupling to Camel classes
- If your bean has multiple methods, you can specify the method that the camel route should use
- Very much suited to unit testing
- code => WordReverser

Data Type Transformation

- Did you notice in the previous examples there was no file handling? Only Strings even though the input and output of the route was a file
- Camel has an elaborate type conversion mechanism that automatically converts between well know types, e.g.

```
String content = exchange.getIn().getBody(String.class);
```

- Type converters are stored in a registry, and detected at startup using classpath scanning
- To avoid excessive/slow scanning the packages to scan are declared in the file META-INF/services/org/apache/camel/TypeConverter
- You can write your own
- code => URLConverter, URLFetcher

Data Transformation

- Camel come with many built in data transformation mechanisms, e.g. csv parsing, conversion from object to json
- It also supports transformation with templates, e.g. Apache Velocity and FreeMarker
- code => DataTransformer

Error Handling

- Camel considers 2 types of errors
  - Recoverable, e.g. network outage causes an IOException. Trying again when the network is fixed and it will work
  - Non-Recoverable, e.g. SQLException due to a bug in an SQL query. No matter how many times to retry the error will still occur
- If unspecified any Throwable is considered to be a recoverable error
- To make an error non-recoverable you need to set the fault flag to true

```
try {
  …
} catch (Throwable ex) {
  exchange.setException(ex);
  Message msg = exchange.getOut();
  msg.setFault(true)
}
```

- To do this you need access to the exchange, which couples your handlers to camel, which isn;t really desirable (but OK if you've decided to implement Processor)

Error Handlers

- Camel also provides error handlers
  - DefaultErrorHandler - This is automatically and retries the route (like a mad thing)
  - DeadLetterChannel - Implements the Dead Letter Channel EIP, it's a great place to send non-recoverable errors
  - TransactionErrorHandler - An error handler that is aware of transactions
  - NoErrorHandler - Use this to disable error handling
  - LoggingErrorHandler - Simply logs the error
  - code => ErrorHandling

Expression Language

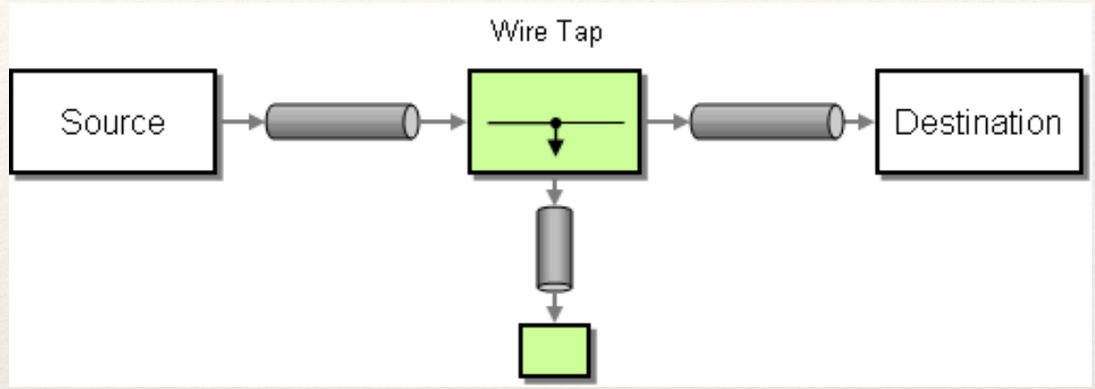
- Camel has a build in expression language that lets you make decisions in routes, or rename the output of a route. It's called "simple".
- The expression handler can evaluate and compare:
  - built in variables e.g. file:name
  - Message headers
  - The Message Body, and JavaBean paths
  - It can even perform transformations on the body!!!
- I can perform many functions:
  - Comparisons: <, >, ==, etc
  - String functions: contains, in, etc, and regex
  - Date formatting/parsing
- Expressions can be combined using AND, OR
- code => ExpressionLanguage

EIPs in Camel

- Apache Camel was written as an implementation of the book enterprise integration patterns, so make it easy to use them
- Wire Tap
- Dynamic Router
- Recipient List
- Splitter
- Aggregator
- Resequencer
- … (the list goes on)

EIP Wire Tap

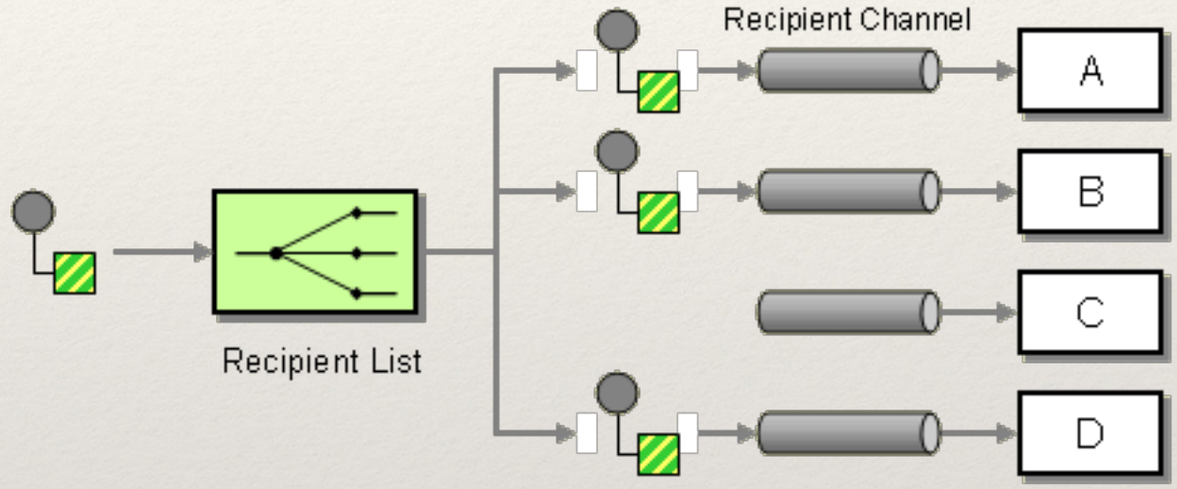
- Allows you to route a message to a separate location, while it is being forwarded to the ultimate destination



- code => WireTap

EIP Recipient List

- Allows you to route a message to a number of dynamically specified recipients
- The list of recipients can come from:
  - Message headers
  - A handler bean/processor
- code => RecipientList



Testing

- Apache Camel provide the camel-test jar, which contains a light-weight way to
  - create a camel context,
    - send messages to the end point
    - mock end points
- code => TestableRoute

Best Practice

- Use the Java DSL, the other lag, and rumour has it they are a bit buggier
- Make user of handlers, marshallers, and type converters
- Decouple parts of a route that depend on external services into a separate small route this will:
  - simplify error handling
  - reduce the amount of rework camel has to do on retry
  - not block input
- This is very easy with Active-MQ
- code => BestPratice

Things Not Covered

- XML support - Camel has impressive xml support, allowing path annotations to get values into a Handler argument
- Transactions - it is possible to make routes transactional, useful when using and sql endpoint or an active-mq endpoint
- Probably lots of other things I haven't got to yet
- Camel Properties - Similar to Springs Property Configurer
- ThreadPools
- Scale Out (think Active MQ)

More Info

- Web - http://camel.apache.org
- Books - Camel in Action by Claus Ibsen and Jonathan Anstey. http://www.amazon.com/Camel-Action-Claus-Ibsen/dp/1935182366