Copyright Notice

Apache Camel
Integration Framework 3.9.0

Prakash Badhe

prakash.badhe@vishwasoft.in

About Me

- ❑Java and Technologies Consultant-Trainer for 20+ years.
- ❑Passion for technologies and frameworks
- ❑Proficient in Java Technology Frameworks and client java script frameworks
- ❑Proficient with service applications and standards
- ❑Working with Spring,Hibernate,GWT,Microservices.
- ❑Integrated service applications with Apache Camel, Spring Framework, Spring Security etc.

About you..

- Job Role
- Skill-Sets
- Objectives
- Experience in Web applications development
- About prior experience with Java, Web services, Messaging etc.

Agenda

- Service applications
- Role of Integration Framework
- Apache Camel introduction
- Messaging with camel
- Camel Integration Components and Routes.
- Enterprise Integration Patterns
- Integration between Service applications
- Apache Camel Routes Testing

Setup

- Windows 10 or Linux Ubuntu or MAC
- JDK 1.8
- Eclipse-Jee-2020
- Adobe PDF Reader
- Apache Tomcat Server
- Apache Active MQ Server
- MySQL database server and client
- Apache CXF Soap Framework
- Open Internet connection

Application and services

- A Calculator and Google web site
- Service is a distributed application
- Service called from clients and other services
- Service has its own protocol and data formats.
- Service is independent of its clients and isolated from clients and other services —loose coupling between services and clients.

More about services

- Any client or service understanding the protocol can call the service
- Service applications are multi-user, multi-clients.
- Services require huge infrastructure.
- Service and clients are developed, tested and deployed independently

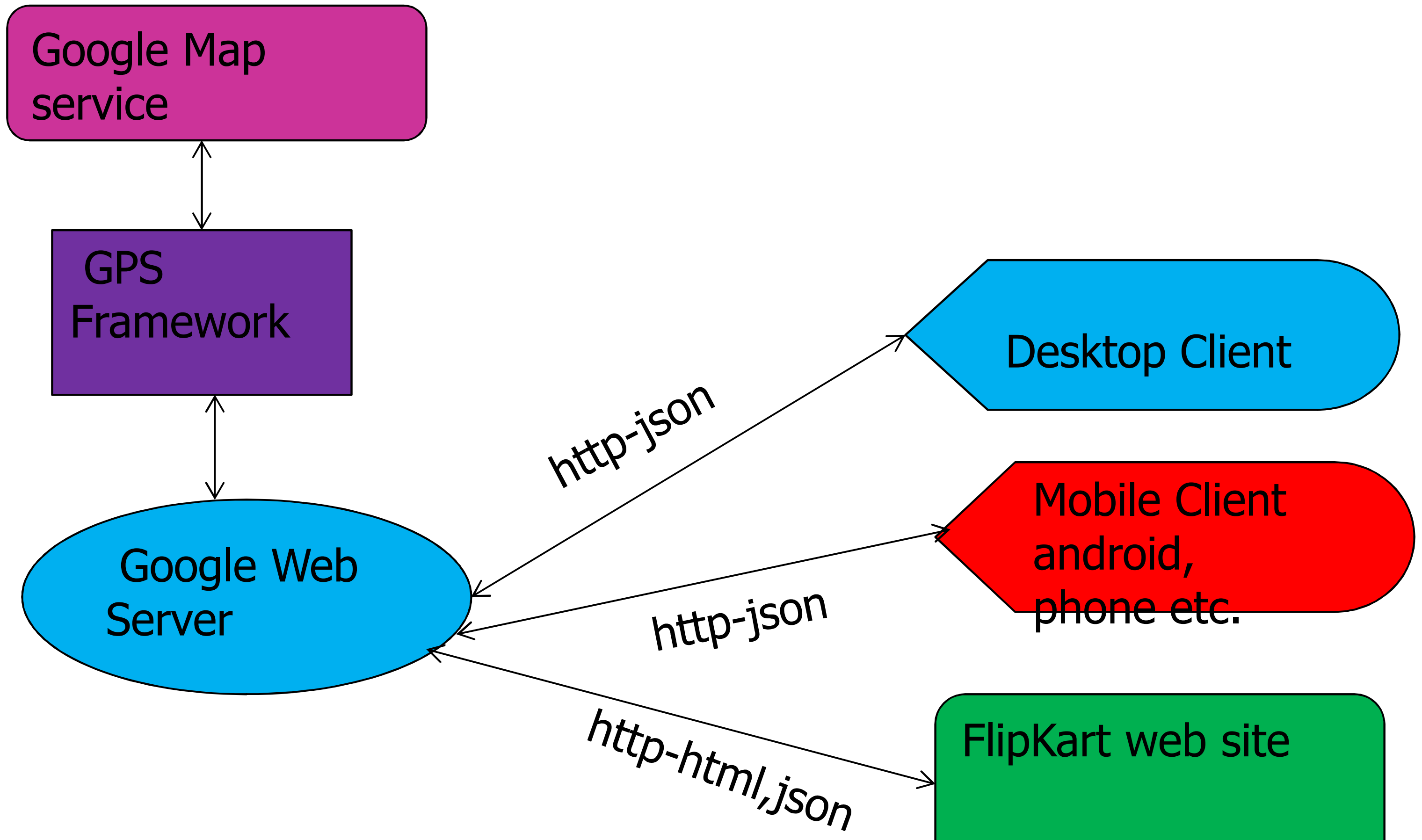
Service Applications

- Distributed applications
- Intercommunication across services
- Different OS platforms and implementations
- Different protocols and data formats
- Different language environments
- Different Hardware platforms
- Different type of devices

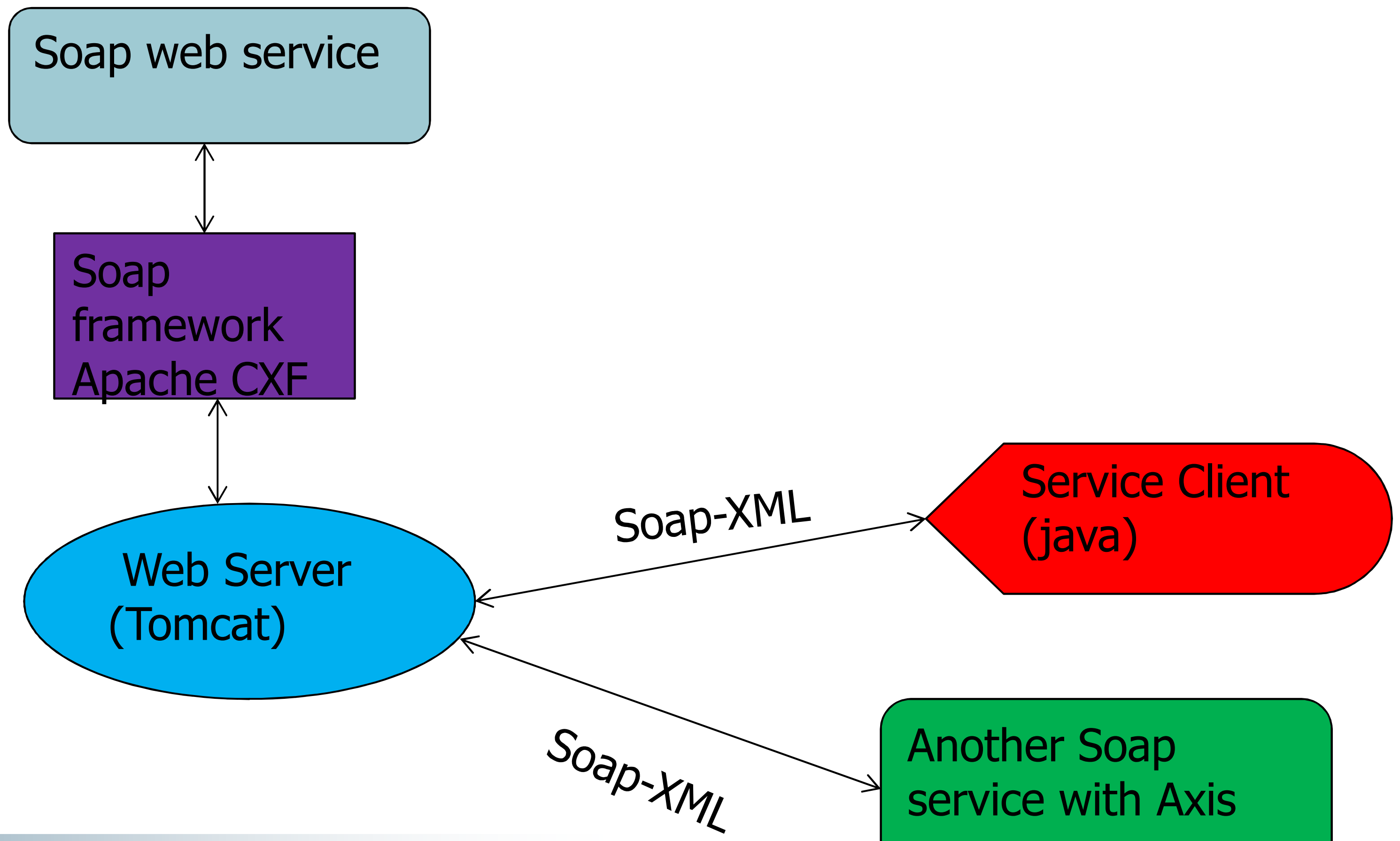
Service Examples

- Web application running on web server.
- Soap web service
- REST service
- MicroService applications
- JPA/Database service
- Timer service
- File monitoring service
- FTP service
- SMTP service

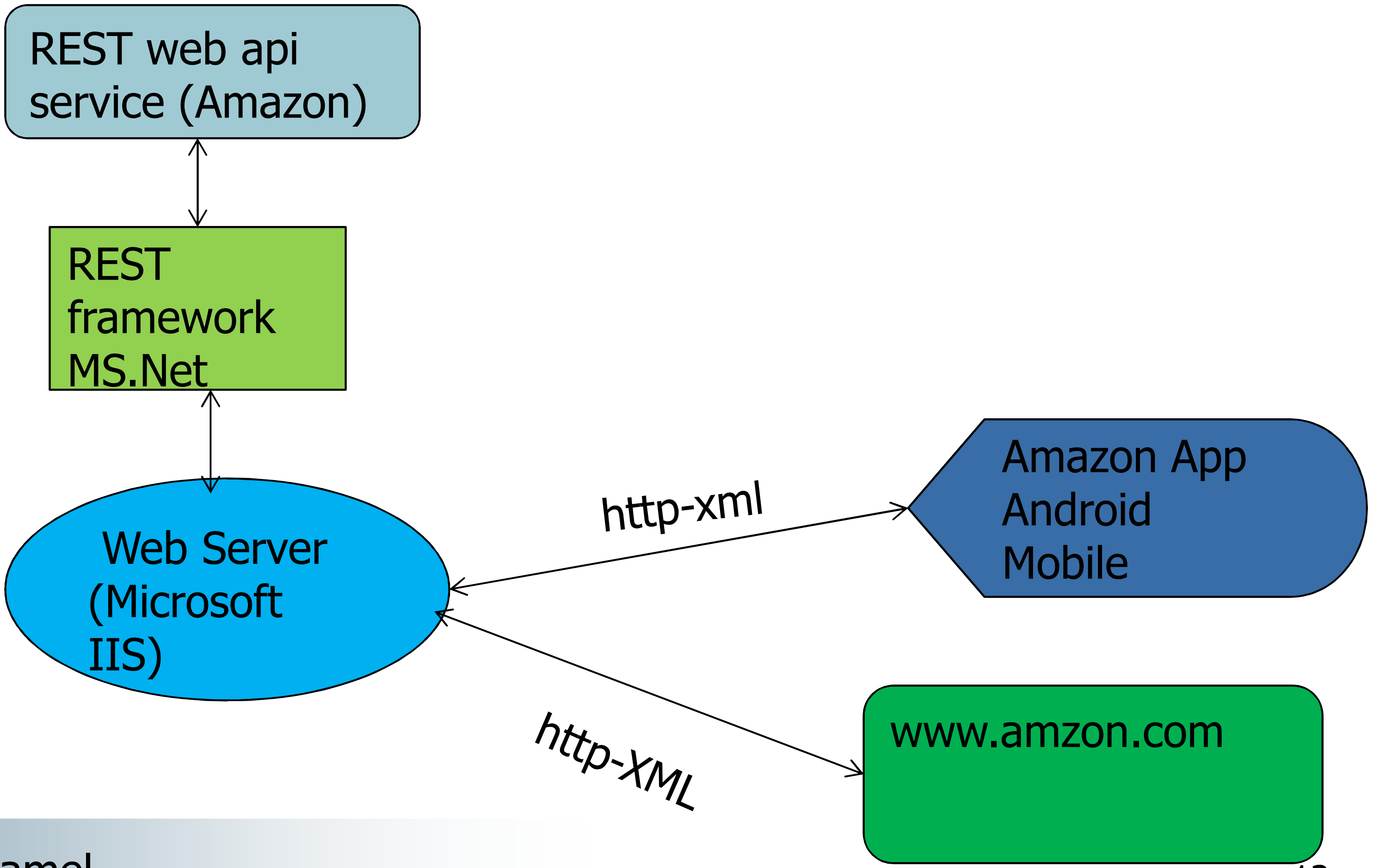
Google Map service and clients



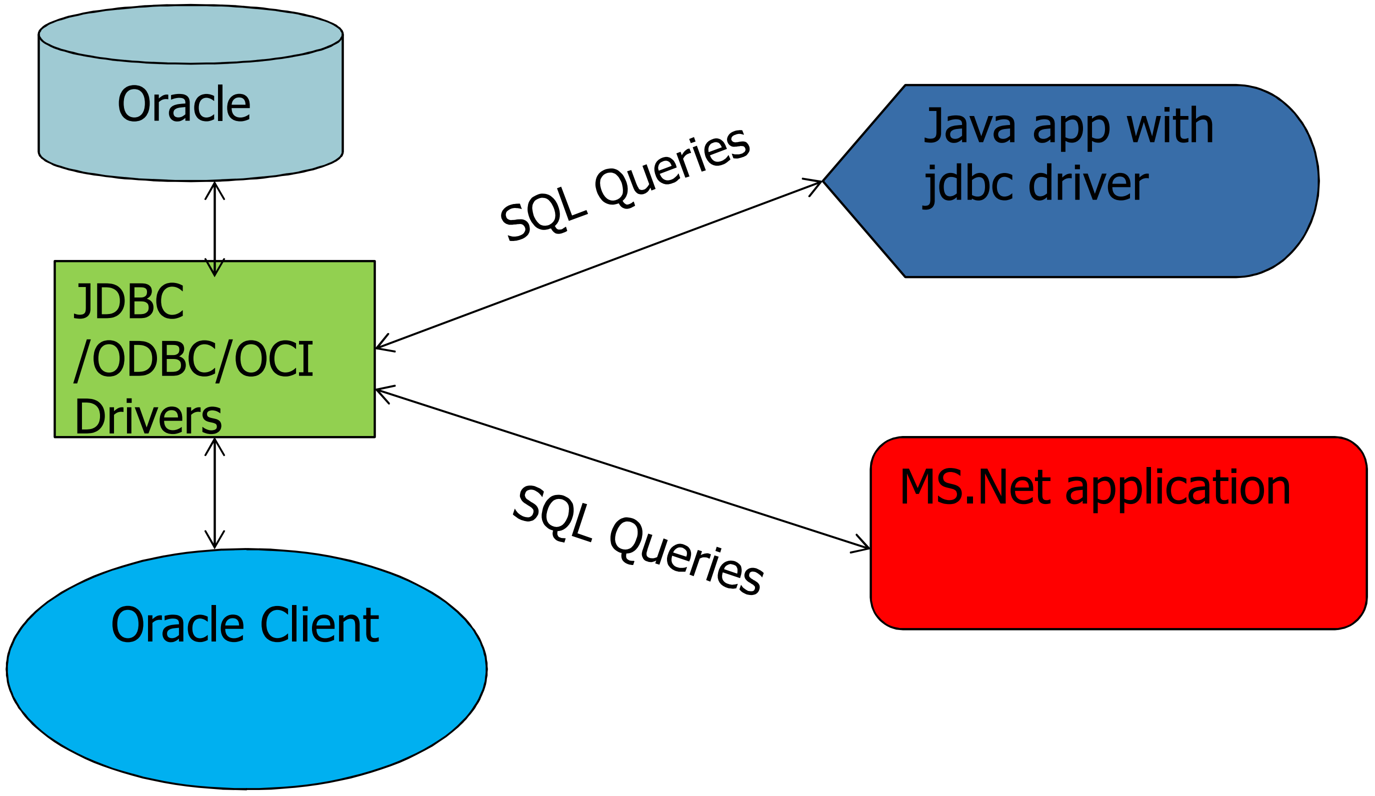
Soap web service



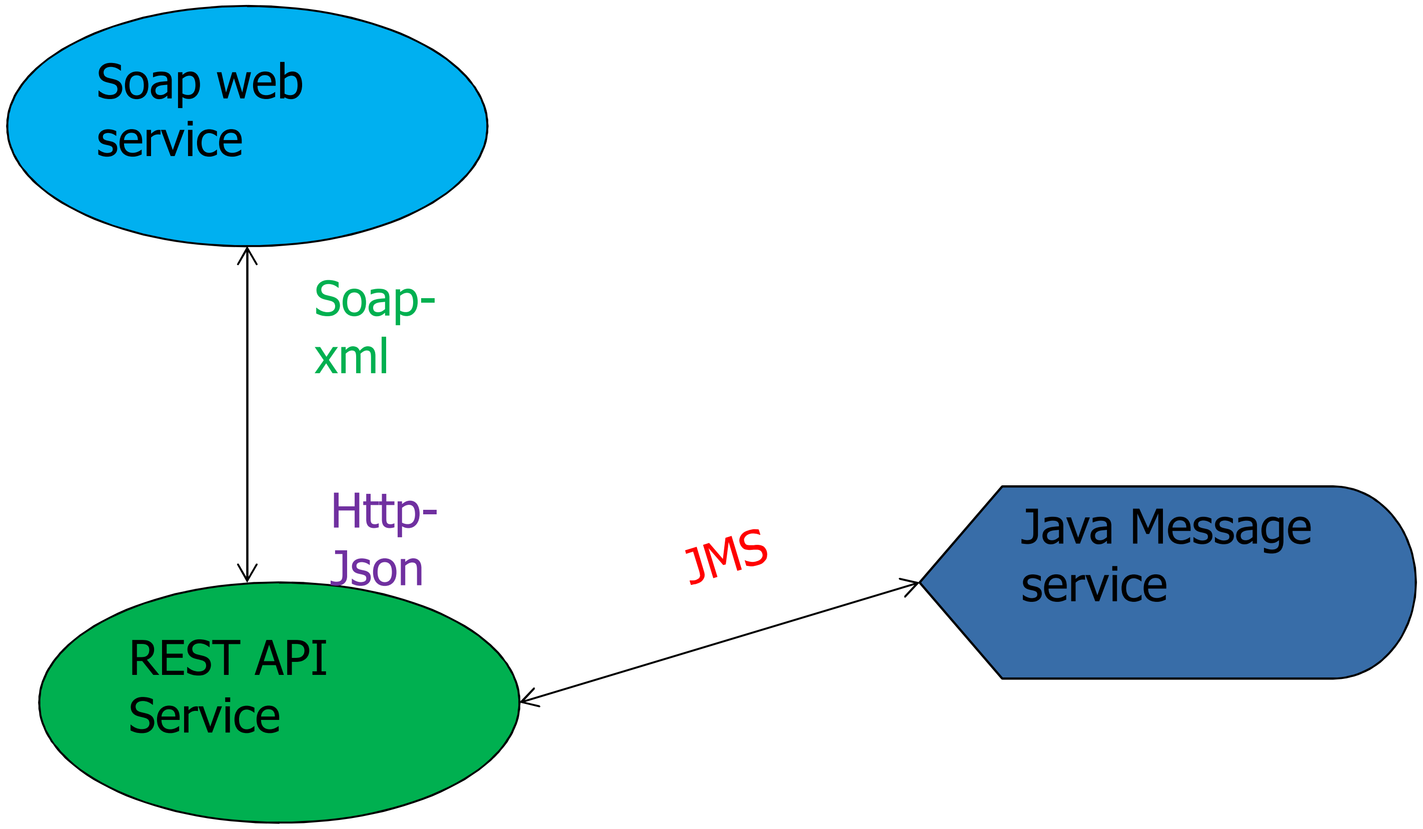
REST API service


REST web api service (Amazon)
REST framework MS.Net
Web Server (Microsoft IIS)
http-xml
Amazon App Android Mobile
http-XML
www.amzon.com

Database service



Service to Service



Service Integration Challenges

- Network communications
- Proprietary (hard coded) communication and data formats
- Data Transfer
- Integration : end to end data management.
- Data across different File systems
- Multi-protocol implementations needed on client side just for communication

Integration Issues

- *Networks are unreliable.*
- *Networks are slow.*
- *Any two applications are different in architecture.*
- *Change is inevitable :* Applications change over time.

Integration approaches

- File Transfer— One application writes a file that another later reads.
- The applications need to agree on the filename and location, the format of the file, the timing of when it will be written and read, and who will manage the file.

Shared DB

- Shared Database — Multiple applications share the same database schema, located in a single physical database.
- There can be duplicate data updated by different services , hence data update has to be controlled.

RPC Invocation

- *Remote Procedure Invocation* — One application exposes some of its functionality so that it can be accessed remotely by other applications as a remote procedure.
- The communication occurs real-time and synchronously

Application Messaging

- *Messaging* — One applications publishes a message to a common message channel. Other applications can read the message from the channel at a later time. The applications must agree on a channel as well as the format of the message. The communication is asynchronous.

Messaging Systems

- *Messaging* is a technology that enables high-speed, asynchronous, program-to-program communication with reliable delivery.
- Programs communicate by sending packets of data called *message*s to each other.
- *Channel*s, also known as queues, are logical pathways that connect the programs and convey messages.
-

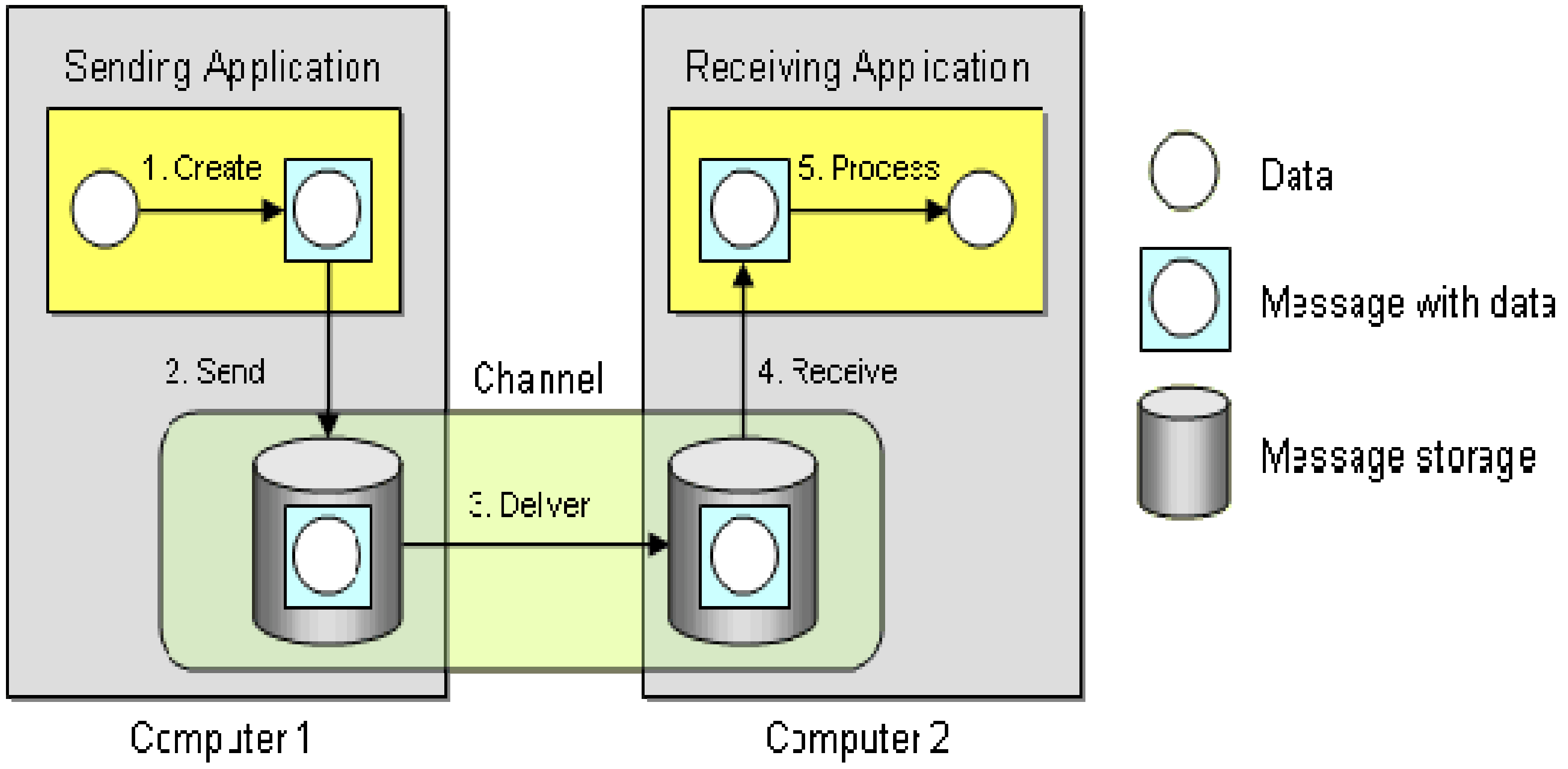
Messaging Server

- Messaging capabilities are typically provided by a separate software system called a *messaging system* or *message-oriented middleware* (MOM).
- A messaging system manages messaging the way a database system manages data persistence.

Configuration

- Just as an administrator must populate the database with the schema for an application’s data, an administrator must configure the messaging system with the channels that define the paths of communication between the applications

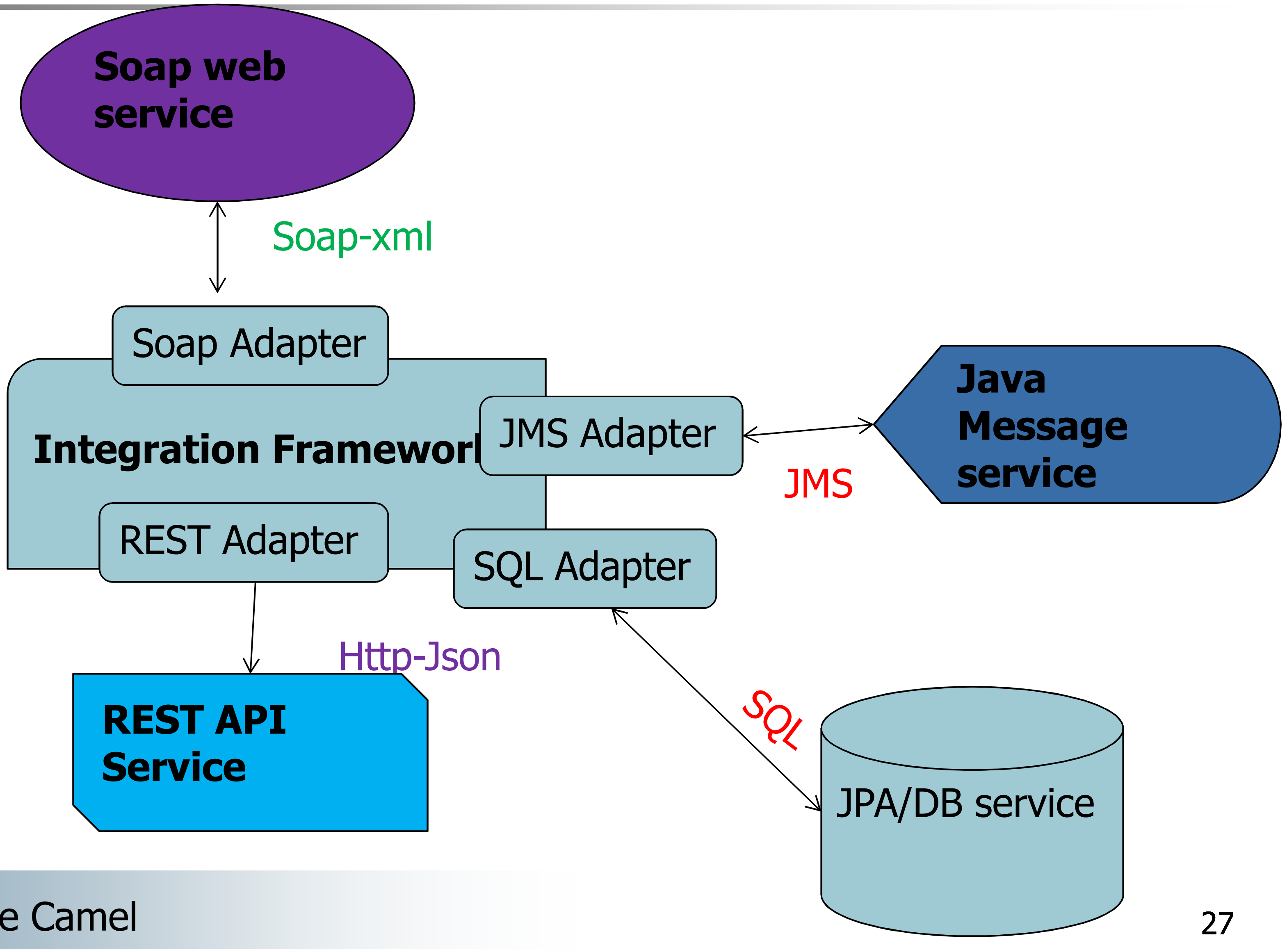
Communication with Messaging



Why Messaging ?

- The messaging is more immediate than *File Transfer*,
- It is better encapsulated than *Shared Database*,
- It is more reliable than *Remote Procedure Invocation*.
- It is asynchronous, non-blocking.

Role of Integration Framework



Integration Frameworks

- Apache camel
- Spring Integration
- Mule
- SAP
- Boomi from Dell
- IBM Integration Tool
- Cloud Integration Tools

Apache Camel Integration

- Apache Camel is an open source integration framework that is designed to make integrating systems easier.
- It increases productivity, simplify development, and provide a solid platform from which the complexities of applications integration are simplified.
- Apache camel integration supports common enterprise integration patterns.

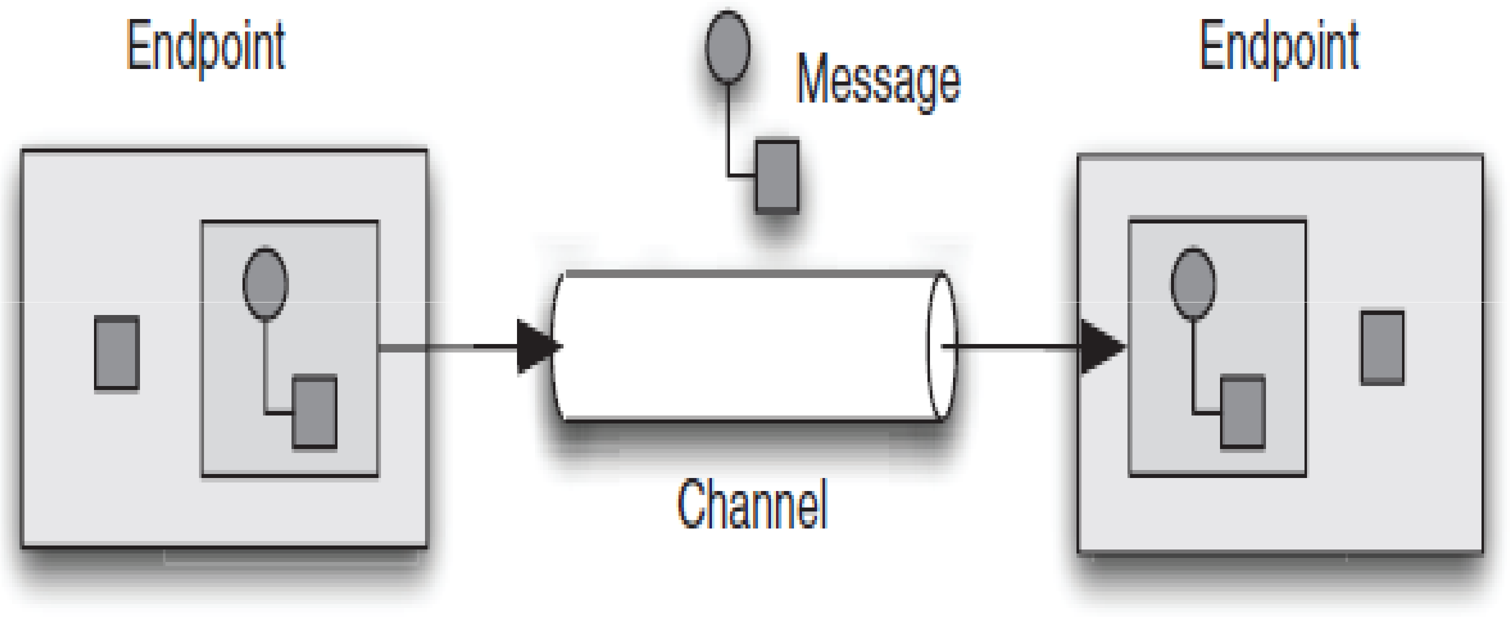
Enterprise Integration patterns

- **https://camel.apache.org/components/latest/eips/enterprise-integration-patterns.html**
- Best practices identified and described by expert developers about the patterns used in the exchange of messages and patterns(guidelines) that provide the communication design between applications.
- Specify what should be done to achieve a particular goal.
- The implementation frameworks follow the implementation guidelines of those patterns.

Messaging patterns

- These patterns in general specify techniques of messaging and integration in the enterprise to achieve the best results.
- The patterns apply to both *intra-application and inter application scenarios.*

Base patterns

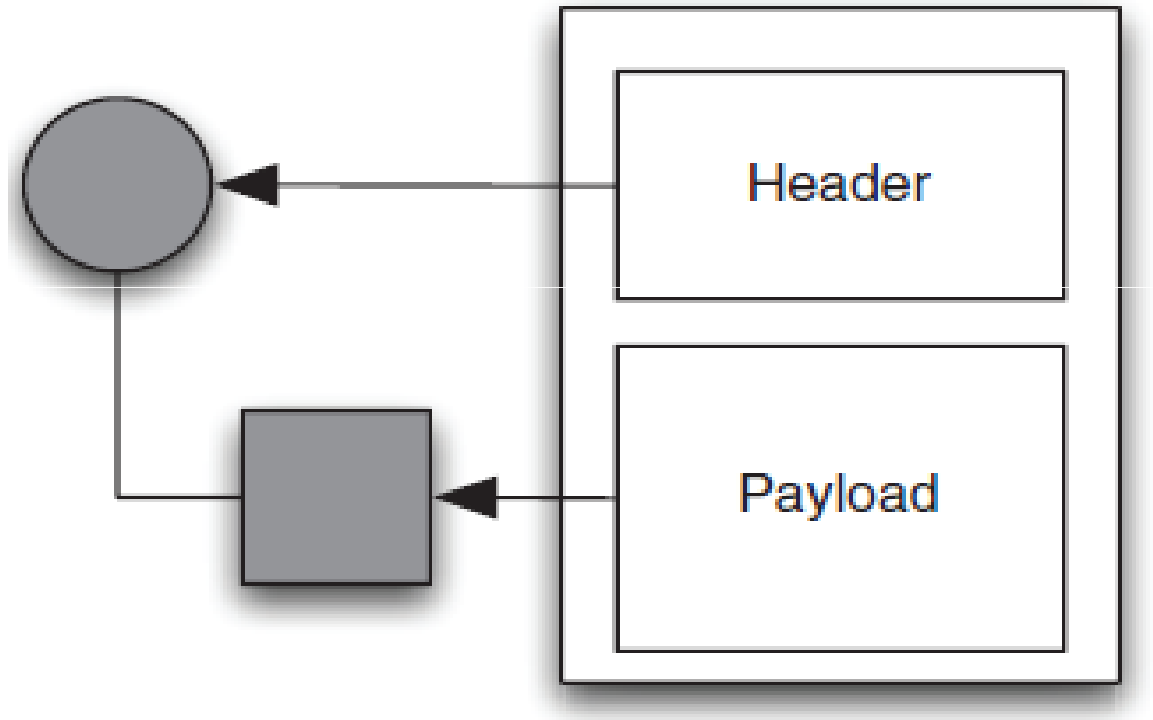


A message is passed through a channel from one endpoint to another endpoint.

Message

- A message is a unit of information that can be passed between different components, called *Message Endpoints. Messages are typically sent after one endpoint is done with a* bit of work, and they trigger another endpoint to do another bit of work.
- Messages can contain information in any format that’s convenient for the sending and receiving endpoints.
- For example, the message’s payload may be XML, a simple string, or a primary key referencing a record in a database

Message structure



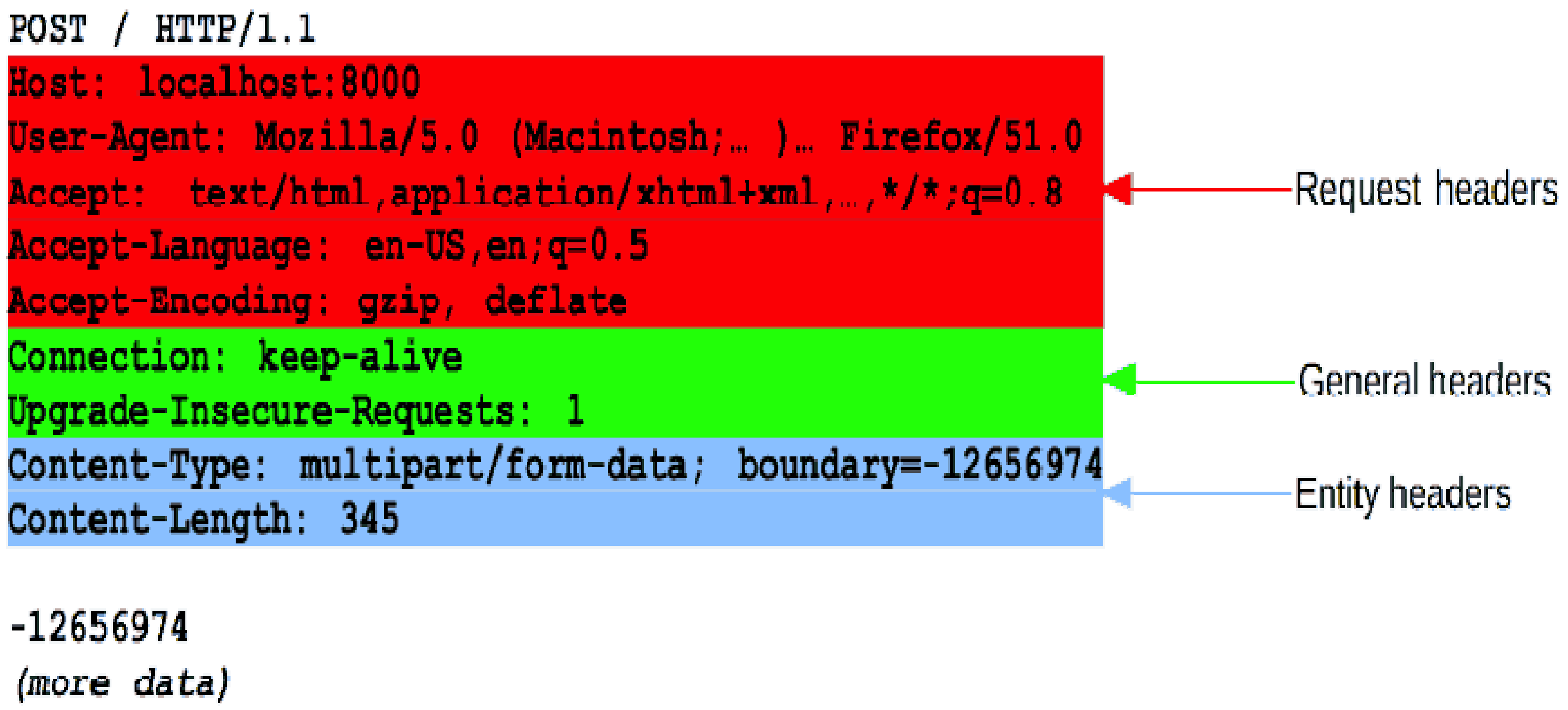
Message Parts

- Messages are the entities used by systems to communicate with each other when using messaging channels. Messages flow in one direction from a sender to a receiver.
- Messages have a body (a payload),
- headers, and optional attachments

Message parts

- Message contains two parts
  - Message header : contains message properties such as id, destination name etc.
  - Message payload : message contents can be any format. The payload contains the actual data to be accessed or processed by the receiver.

HTTP Request Message



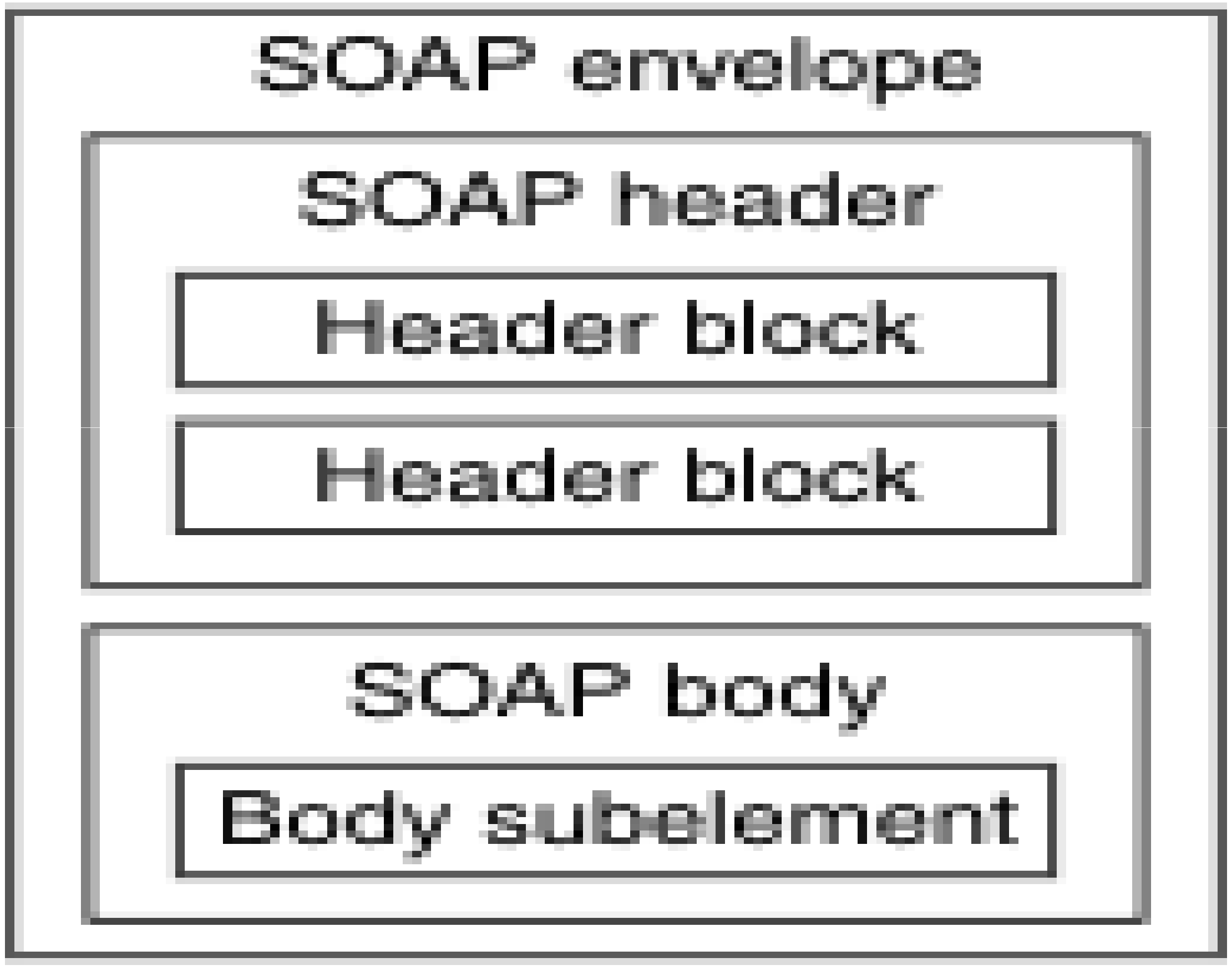
```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;… )… Firefox/51.0
Accept: text/html,application/xhtml+xml,…,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
```

Http Response Message Headers

- User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
- Host: www.server.com
- Accept-Language: en, mi
- Date: Mon, 27 Jul 2009 12:28:53 GMT Server: Apache
- Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
- ETag: "34aa387-d-1568eb00"
- Accept-Ranges: bytes Content-Length: 51 Vary:
- Accept-Encoding Content-Type: text/plain

SOAP Message



SOAP XML Message

- <?xml version = "1.0"?>
- <SOAP-ENV:Envelope xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope" SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">
- <SOAP-ENV:Header>
- ... ...
- </SOAP-ENV:Header>
- <SOAP-ENV:Body> ... ...
- <SOAP-ENV:Fault> ... ... </SOAP-ENV:Fault>
- ...
- </SOAP-ENV:Body>
- -------
- </SOAP_ENV:Envelope>

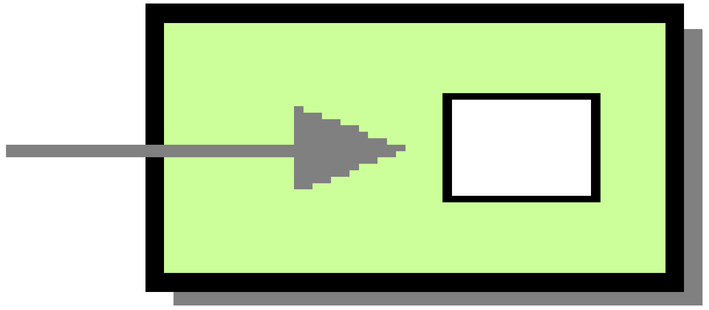
Message behavior

- Messages can have different functions.
- A Command Message tells the receiver to do some action.
- A an Event Message notifies the receiver that something has happened,
- A Document Message transfers some data from the sender to the receiver.

Message standardization

- Message must be understood by all the concerned parties.
- The message is a representation of the contract between the sender and receiver.
- In some applications it might be fine to send a reference to an object over the channel, but in others it might be necessary to use a more interoperable representation like an identifier or a serialized version of the original data.
- In some cases message might need transformation

Messaging Endpoint

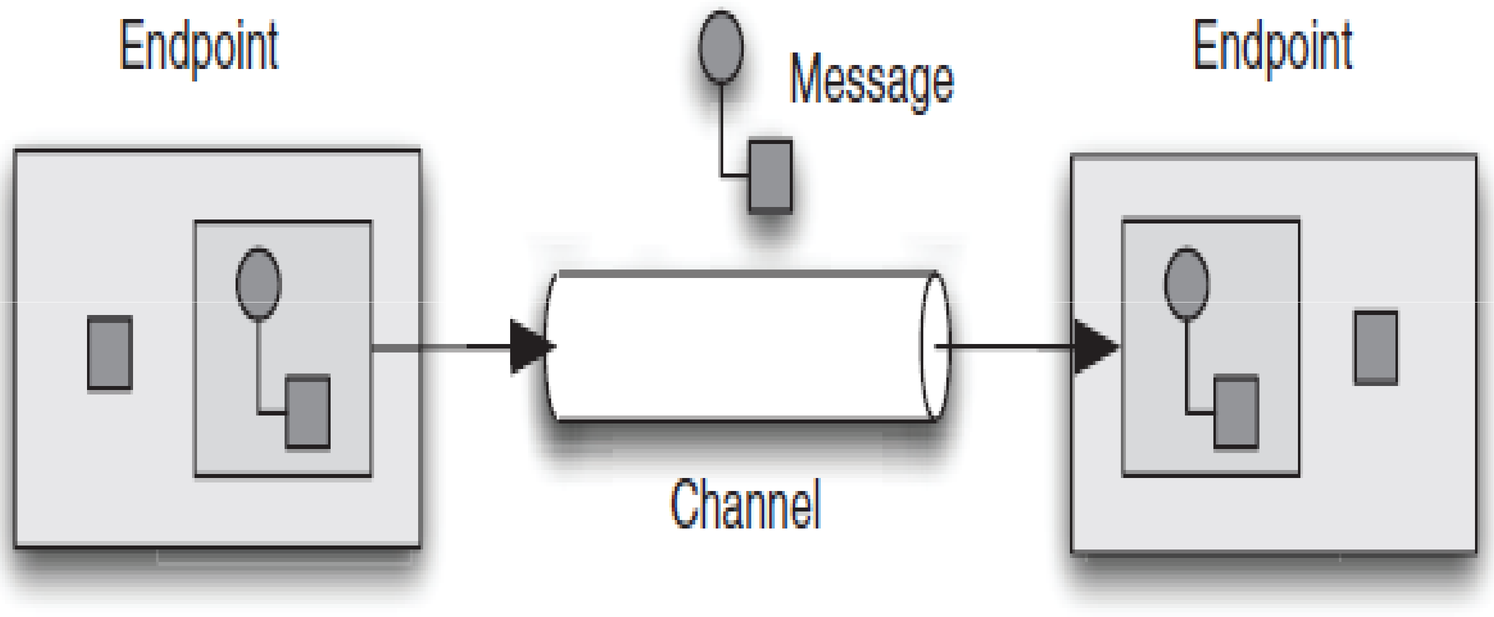


How does an application connect to a messaging channel to send and receive messages?

Messaging End-Point

- **Endpoints**
- Most applications do not have any built-in capability to interface with a messaging system.
- Rather, they must contain a layer of code that knows both how the application works and how the messaging system works, bridging the two so that they work together.
- This bridge code is a set of coordinated *Message Endpoints* that enable the application to send and receive messages

Message travel to End Points



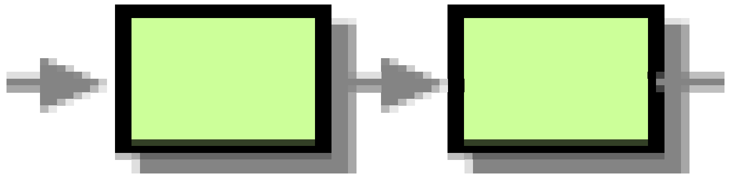
Message endpoints

- Message endpoints are the components that actually do something with the message.
- This can be as simple as routing to another channel or as complicated as splitting the message into multiple parts or aggregating the parts back together.

Message transport

- **Messages :** A *Message* is an atomic packet of data that can be transmitted on a channel
- **Channels** —Messaging applications transmit data through a *Message Channel* a virtual pipe that connects a sender to a receiver.

Message pipe and Filter



How can we perform complex processing on a message while maintaining independence and flexibility?

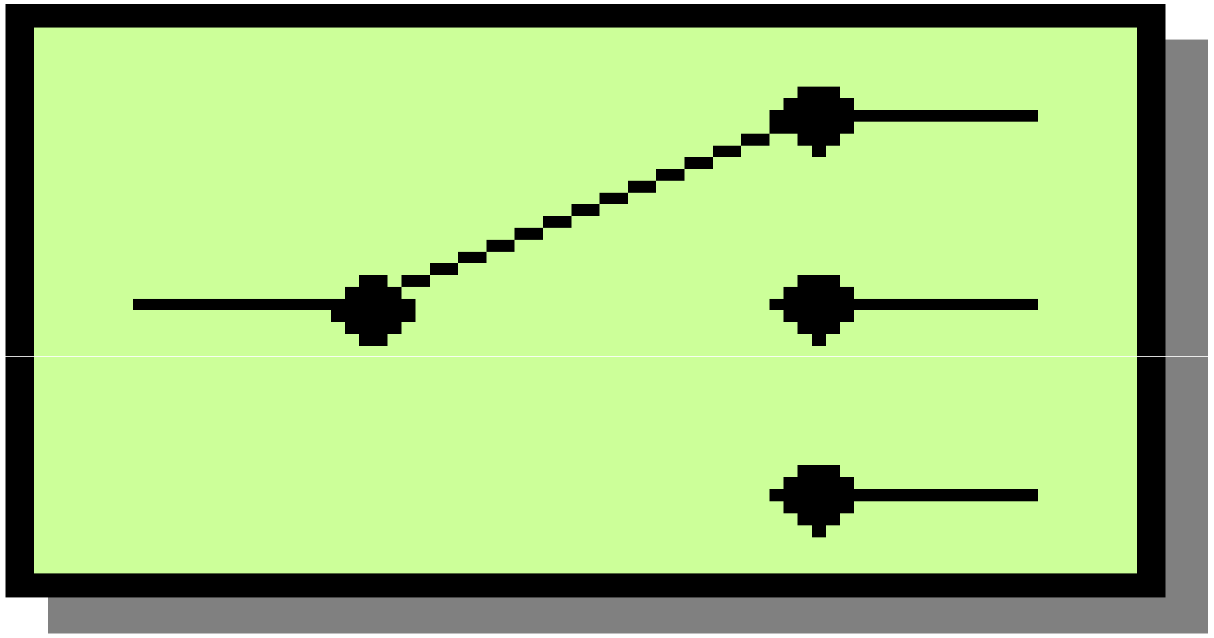
Messaging Pipes

- **Pipes and Filters** — The messaging system delivers a message directly from the sender's computer to the receiver's computer.
- However, certain actions often need to be performed on the message after it is sent by its original sender but before it is received by its final receiver.
- For example, the message may have to be validated or transformed because the receiver expects a message format different from the sender's.

Messaging Pipes

- The *Pipes and Filters* architecture describes how multiple processing steps can be chained together using channels.

Message Router



How can we decouple individual processing steps so that messages can be passed to different endpoints depending on a set of conditions?

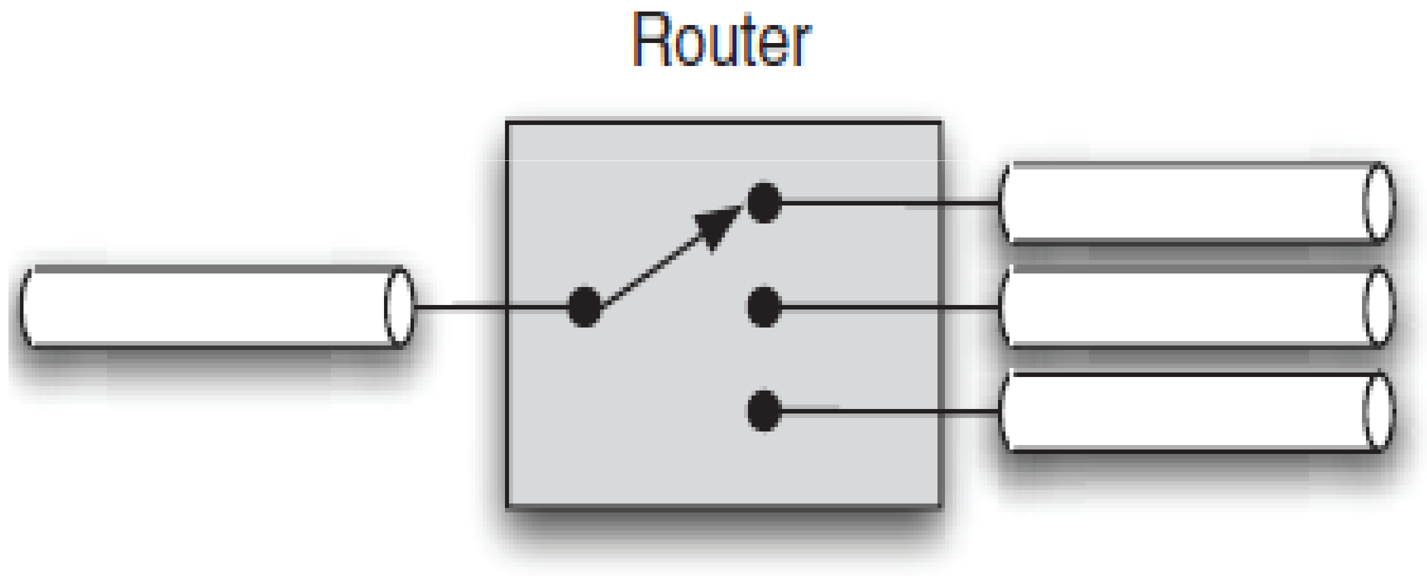
Message Routing

- **Routing**
- In a large enterprise with numerous applications and channels to connect them, a message may have to go through several channels to reach its final destination.
- The route a message must follow may be so complex that the original sender does not know what channel will get the message to the final receiver.

Routing

- The original sender sends the message to a *Message Router* ,an application component that takes the place of a filter in the *Pipes and Filters* architecture.
- The router then determines how to navigate the channel topology and directs the message to the final receiver, or at least to the next router.

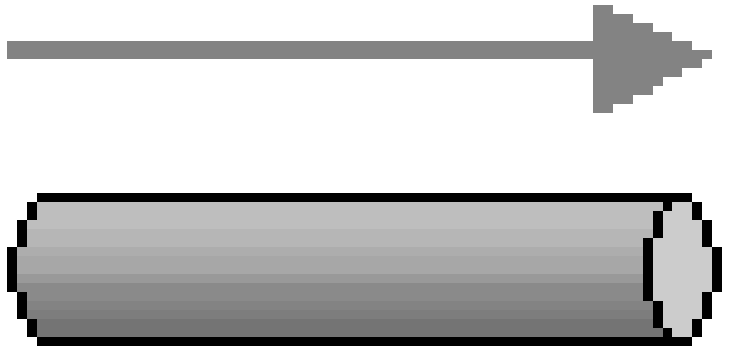
Router Implementation



Message **Transformation**

- Various applications may not agree on the format for the same conceptual data; the sender formats the message one way, but the receiver expects it to be formatted another way.
- To reconcile this, the message must go through an intermediate filter, a *Message Translator*, which converts the message from one format to another.

Message Channel

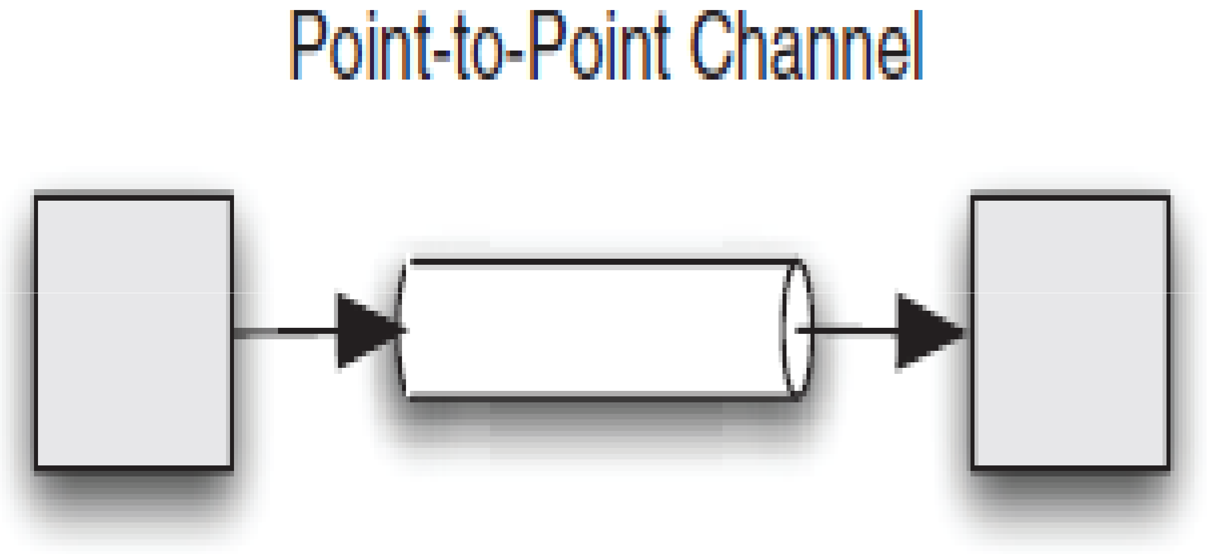


How does one application communicate with another using messaging?

Message Channel categories

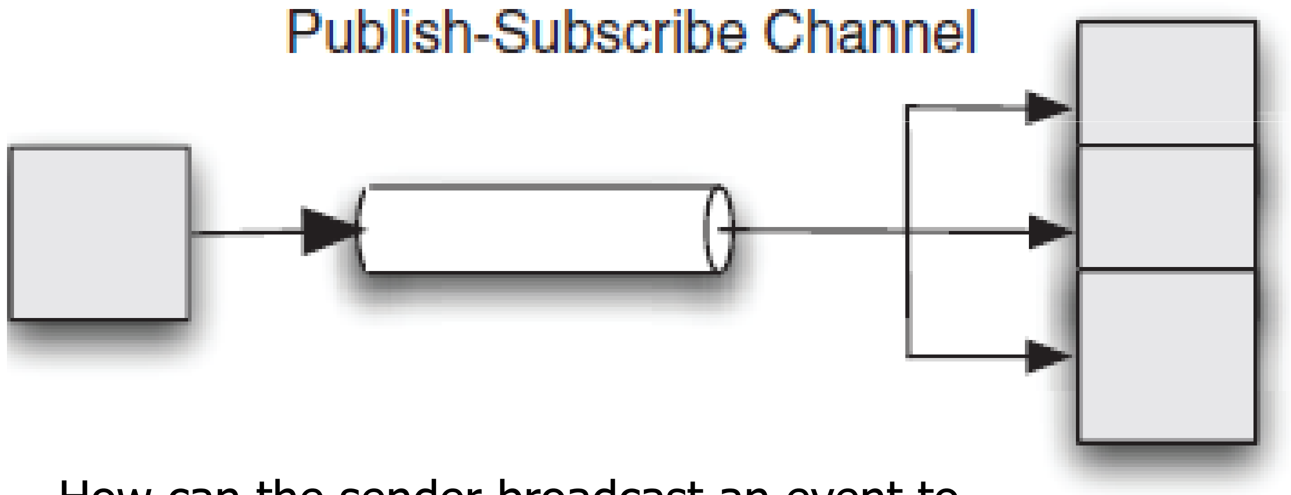
- Channels can be categorized based on two dimensions:
  - The type of handoff
  - The type of delivery.
- The handoff can be either synchronous or asynchronous.
- The delivery can be either point-to-point or publish-subscribe delivery

Point to Point channel



Each single message that's sent by a producer is received by exactly one consumer.

Publish-Subscribe channel

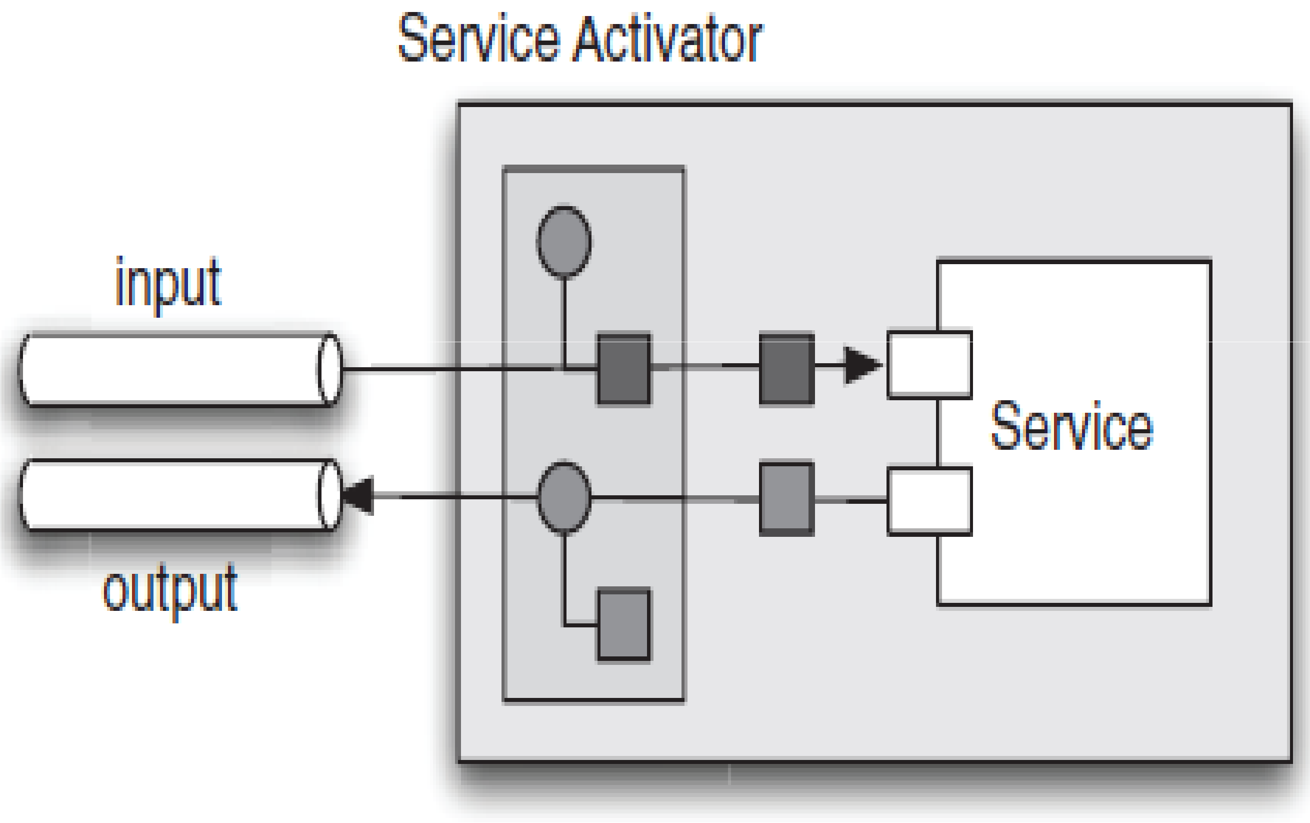


How can the sender broadcast an event to all interested receivers?

Publish-Subscribe Drawback

- The drawback of publish-subscribe messaging is that the sender isn’t informed about message delivery or failure to the same extent as in point-to-point configurations.
- Publish-subscribe scenarios often require failure-handling techniques.

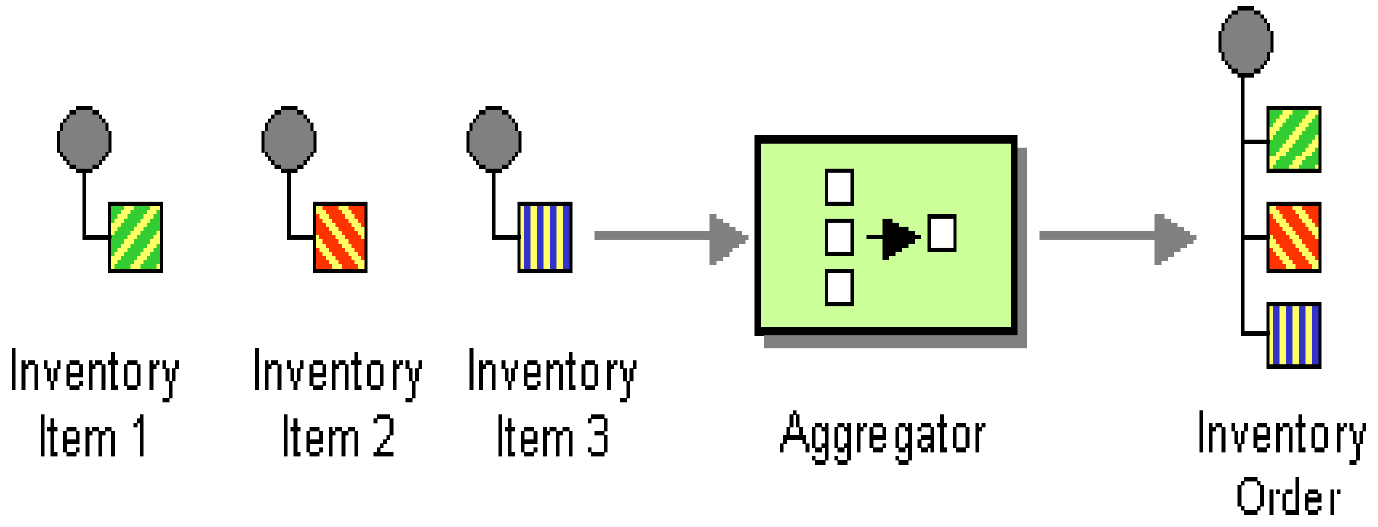
Service Activator pattern



Service Activator pattern

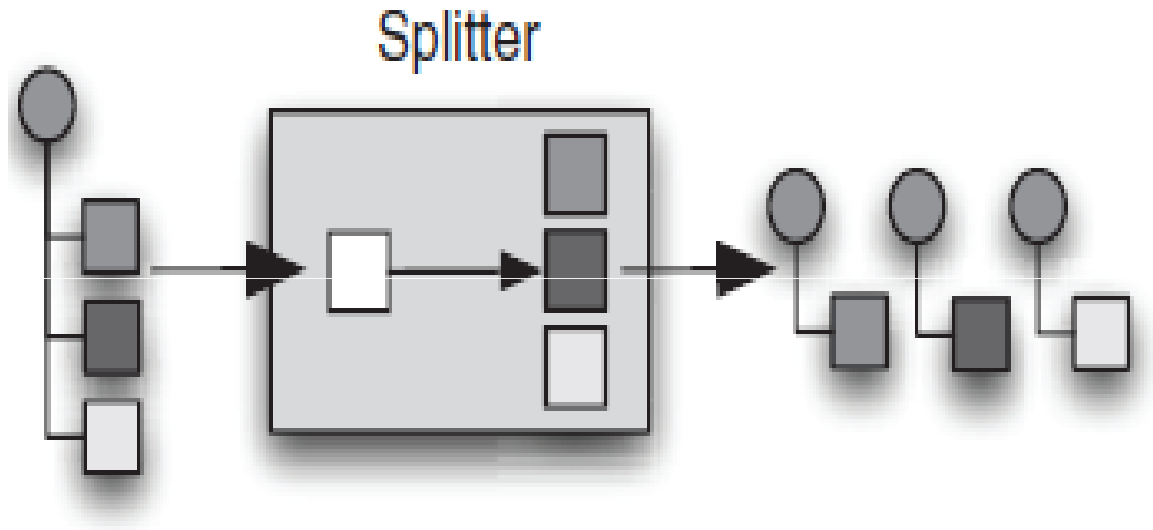
- A *Service Activator is a component that invokes a service based on an* incoming message and sends an outbound message based on the return value of this service invocation.
- In Spring Integration, the definition is constrained to local method calls, so you can think of a Service Activator as a method-invoking outbound gateway. The method that’s being invoked is defined on an object that’s referenced within the same Spring application context.

Aggregator pattern



Determine the messages which should be aggregated together. To aggregate all messages into a single message, use a constant expression.
An AggregationStrategy is used to combine all the message exchanges for a single correlation key into a single message exchange.

Splitter Pattern



How can we process a message if it contains multiple elements, each of which may have to be processed in a different way?

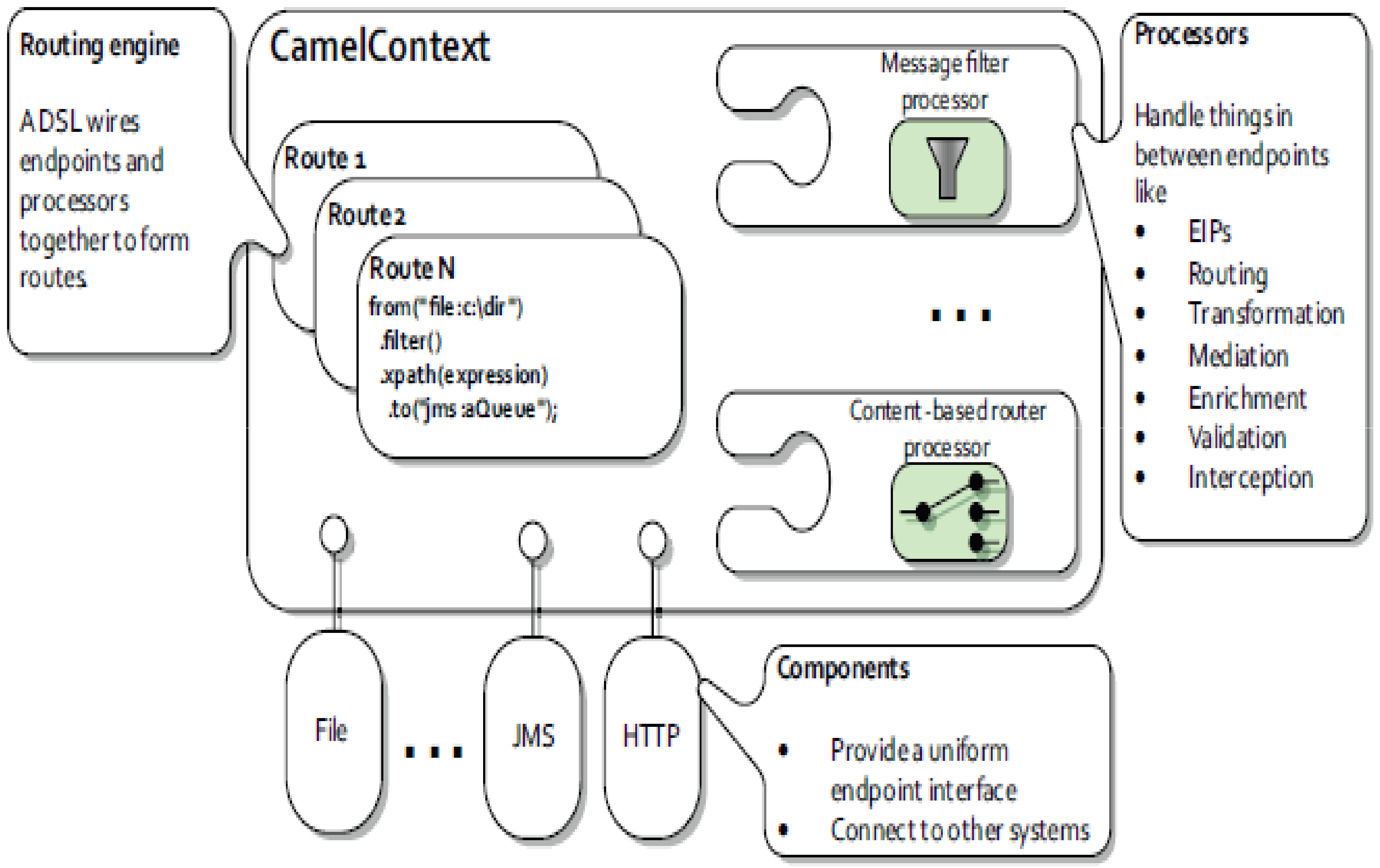
ENTERPRISE INTEGRATION PATTERNS (EIPS)

- Many problems and their solutions are quite similar.
- These are catalogued and known as *Enterprise Integration Patterns.*
- *These pattern also specify best practices for messaging and integration.*

Camel Integration Components

- Camel offers higher-level abstractions that allow to interact with various systems using the same API regardless of the protocol or data type the systems you are using.
- Components in Camel provide specific implementations of the API that target different protocols and data types.
- Camel comes with support for over 80
- protocols and data types.

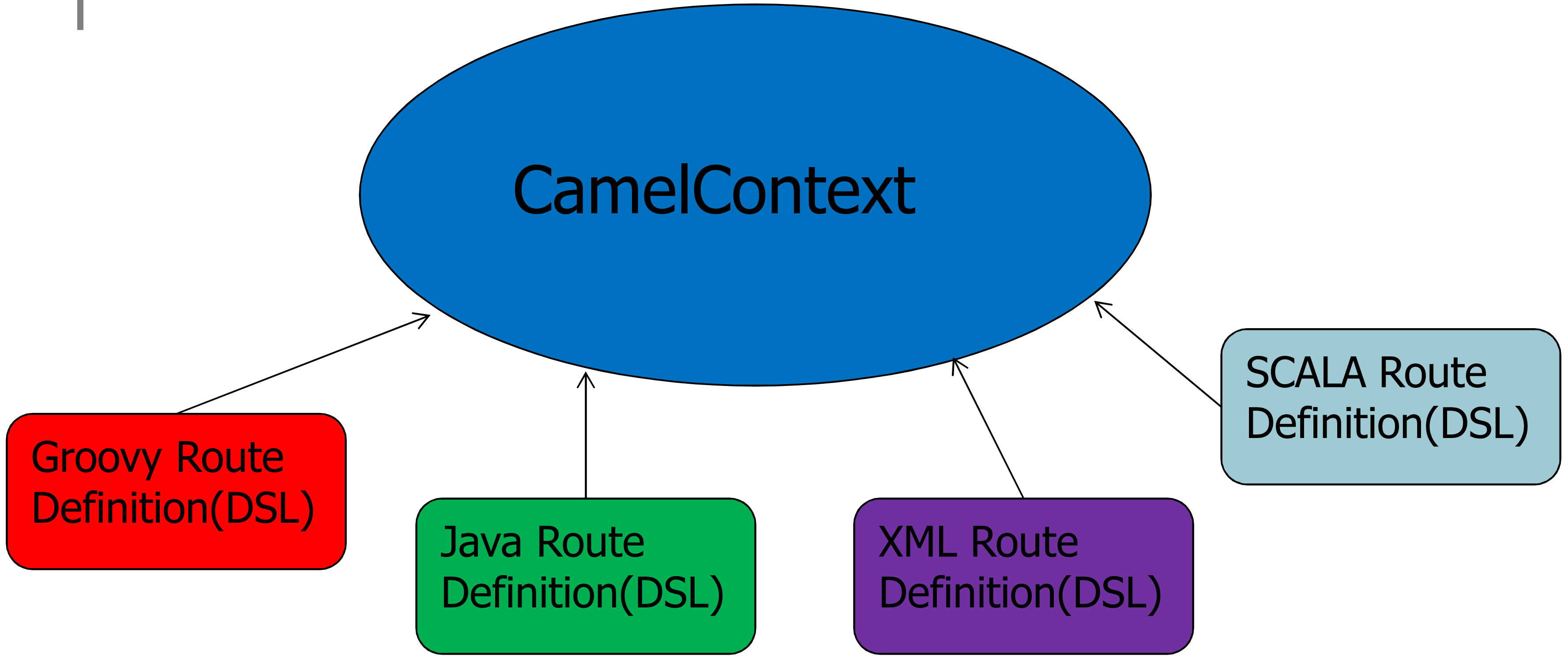
Camel Architecture



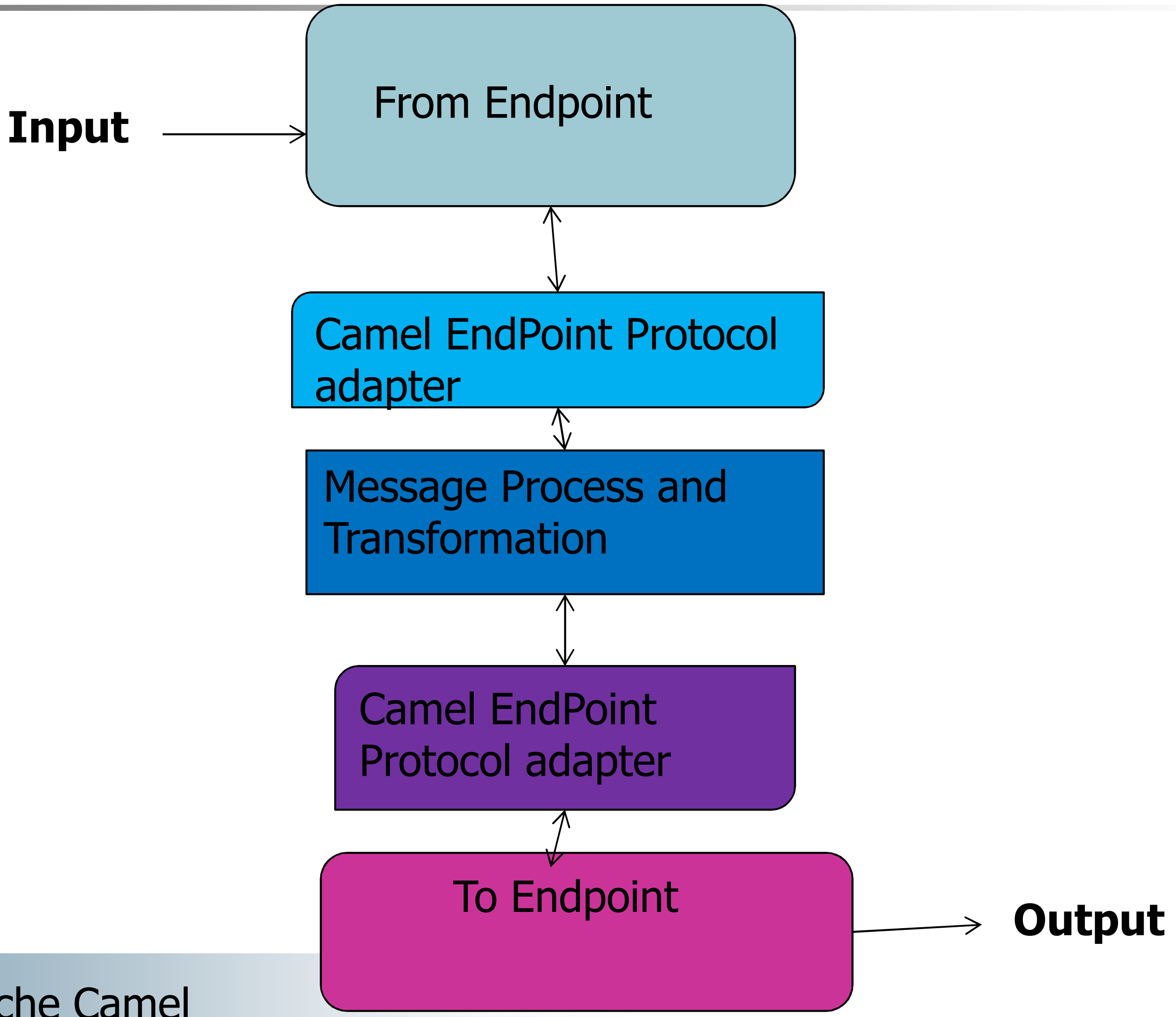
ROUTING AND MEDIATION ENGINE

- The core feature of Camel is its routing and mediation engine.
- A routing engine will selectively move a message around, based on the route’s configuration.
- In Camel’s case, routes are configured with a combination of enterprise integration patterns and domain-specific language.

Camel Message Routing Execution


CamelContext
Groovy Route Definition(DSL)
Java Route Definition(DSL)
XML Route Definition(DSL)
SCALA Route Definition(DSL)

Route Definition with DSL


Input
From Endpoint
Camel EndPoint Protocol adapter
Message Process and Transformation
Camel EndPoint Protocol adapter
To Endpoint
Output

DOMAIN-SPECIFIC LANGUAGE (DSL)

- Camel’s domain-specific language (DSL) is a major contribution to the integration space.
- Some other frameworks use XML to describe routing rules, but unlike Camel their DSLs are based on custom languages.
- Camel is unique because it offers multiple DSLs in regular programming
- languages such as Java, Scala, Groovy, and it also allows routing rules to be specified in XML.

DSL advantage

- The purpose of the DSL is to allow the developer to focus on the integration problem rather than on the tool—the programming language.
- Although Camel is written mostly in Java, it does support mixing multiple programming platforms.

DSL Flavors

- Java DSL
- from("file:data/inbox").to("jms:queue:order");
- ■ Spring DSL
- <route>
- <from uri="file:data/inbox"/>
- <to uri="jms:queue:order"/>
- </route>
- ■ Scala DSL
- from "file:data/inbox" -> "jms:queue:order"
- These examples are real code, and they show how easily

PAYLOAD-AGNOSTIC ROUTER

- Camel can route any kind of payload—you aren’t restricted to carrying XML payloads.
- This freedom means that you don’t have to transform your payload into a canonical format to facilitate routing.
-

MODULAR AND PLUGGABLE ARCHITECTURE

- Camel has a modular architecture, which allows any component to be loaded into Camel, regardless of whether the component ships with Camel, is from a third party, or is your own custom creation.

POJO MODEL

- Beans (or POJOs) are considered first-class citizens in Camel.
- Camel strives to let you use beans anywhere and anytime in your integration projects.
- This means that in many places you can extend Camel’s built-in functionality with your own custom code.

EASY CONFIGURATION

- The *convention over configuration paradigm is followed whenever possible, which minimizes*
- configuration requirements.
- In order to configure endpoints directly in routes,Camel uses an easy and intuitive URI configuration.

Configuration made easier

- For example, you could configure a file consumer to scan recursively in a subfolder and include only a .txt file, as follows:
- from("file:data/inbox?recursive=true&include=*.txt")...

AUTOMATIC TYPE CONVERTERS

- Camel has a built-in type-converter mechanism that ships with more than 150 converters.
- You no longer need to configure type-converter rules to go from byte arrays to strings, for example. And if you find a need to convert to types that Camel doesn’t support, you can create your own type converter.

Test support

- Camel provides a Test Kit that makes it easier to test your own Camel applications.

Camel message model

- In Camel, there are two abstractions for modelling messages
- ■ org.apache.camel.Message—The fundamental entity containing the data
- being carried and routed in Camel
- ■ org.apache.camel.Exchange—The Camel abstraction for an exchange of messages.
- This exchange of messages has an "in" message and as a reply, an "out" message

CAMELCONTEXT

- Camel-Context is a container of sorts,
- Think of it as Camel’s runtime system, which keeps all the pieces together.
- There are a lot of services for the Camel-Context to keep track of.

Camel-Context services

- Components
- Endpoints
- Routes
- Routes
- Type converters
- Data formats
- Registry
- Languages

ROUTES

- Routes are obviously a core abstraction for Camel.
- The simplest way to define a route is as a chain of processors. There are many reasons for using routers in messaging applications.
- Routes decoupe clients from servers, and producers from consumers.

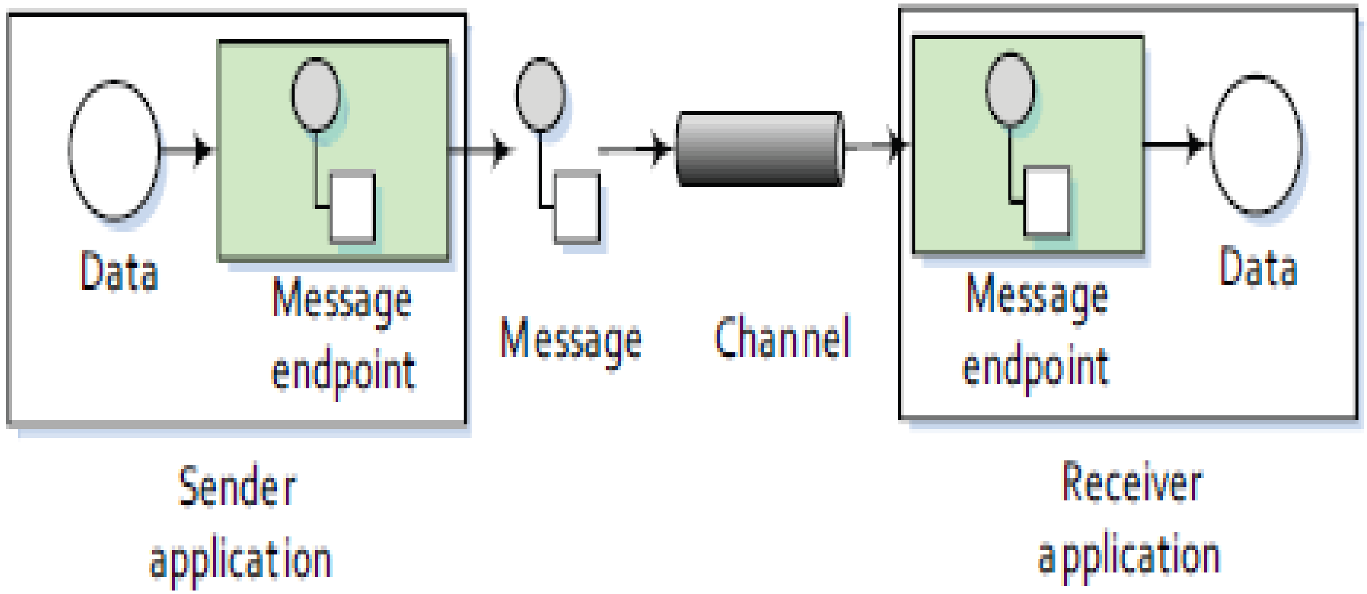
PROCESSOR

- The processor represents a node capable of using, creating, or modifying an incoming exchange.
- During routing, exchanges flow from one processor to another

Camel Message ENDPOINT

- An endpoint is the Camel abstraction that models the end of a channel through which a system can send or receive messages.

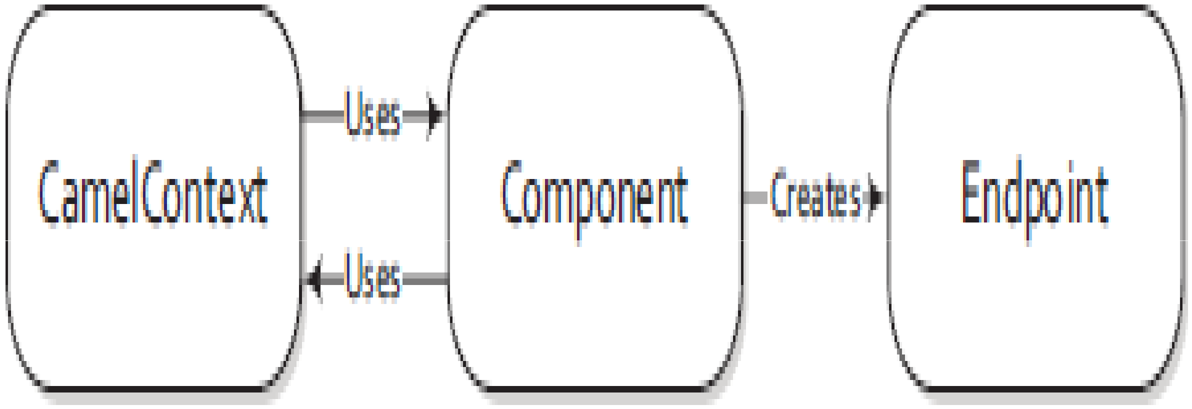
Camel Messaging revisited



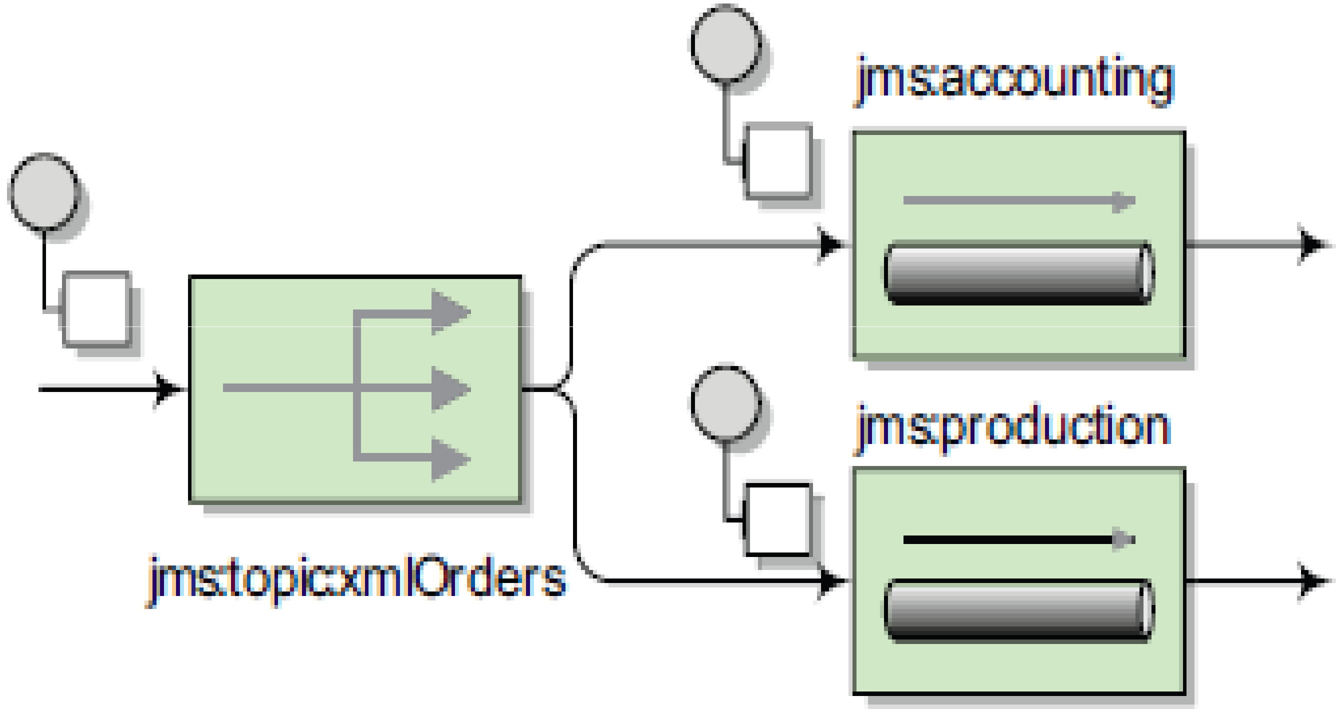
Components as adapters

- File and FTP : File I/O
- JMS :Asynchronous messaging
- CXF : Using web services
- MINA: Networking
- JDBC and JPA: Database support
- Direct, SEDA and VM :In-memory Messaging support
- Timer and Quartz : Task automation

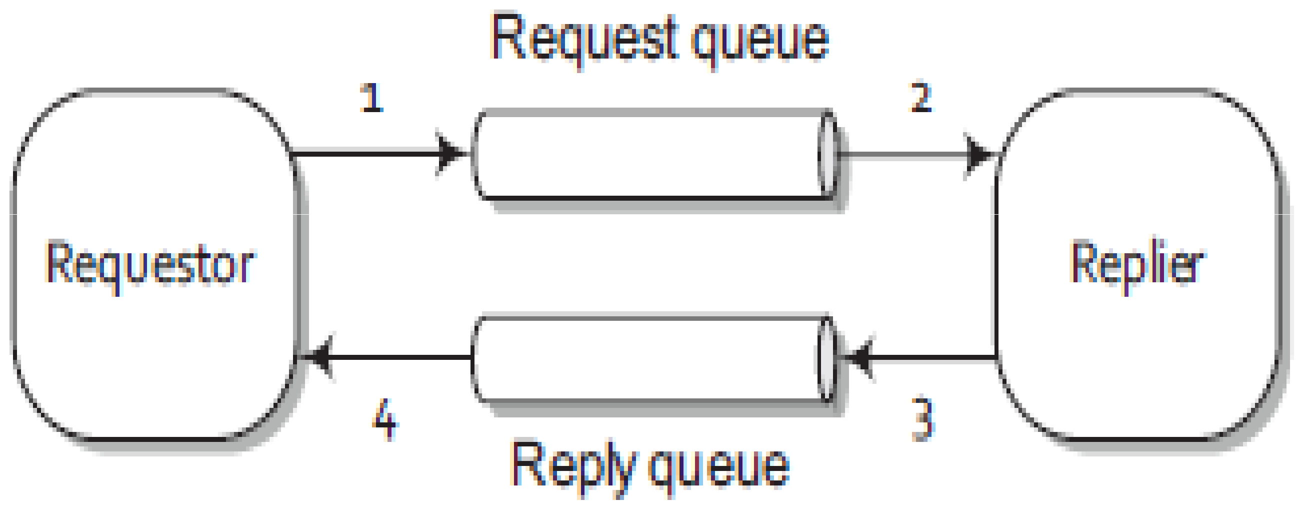
Component =channel +adapter



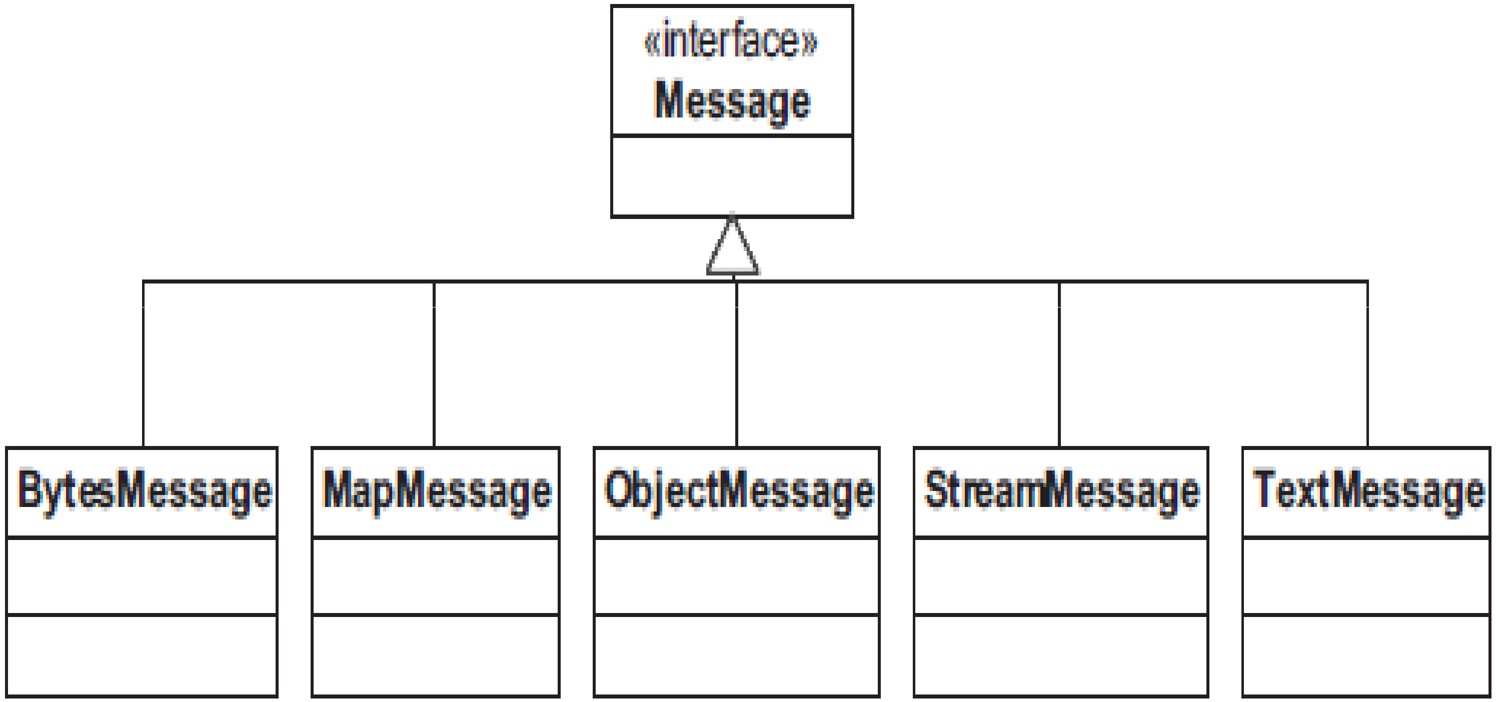
Messaging with JMS



Request-Reply Messaging



Message types



Camel Maven Archetypes

- Predefined maven configuration (as template for maven project)for dependencies of camel maven modules.
- camel-archetype-java
- camel-archetype-main
- camel-archetype-spring
- camel-archetype-spring-boot

Camel Components

- camel-file : Read and write files.
- camel-timer: Generate messages in specified intervals using java.util.Timer.
- camel-stream : Read and write to console.
- camel-log:Logs messages to the underlying logging mechanism.
- camel-mock: test routes and mediation rules using mocks.

Camel Components

- camel-bean :Invoke methods of Java beans stored in Camel registry.
- camel-direct: call another endpoint from the same Camel Context synchronously. This is logical component used just as input and output endpoint.
- camel-seda:Asynchronously calls another endpoint from any Camel Context in the same JVM. This is logical component used just as input and output endpoint

Camel http component

- The HTTP component provides HTTP based endpoints for calling external HTTP resources (as a client to call external servers using HTTP).
- Used to define routes from rest api and web applications in general.

Camel with CXF Soap services

- The CXF component provides integration with Apache CXF for connecting to JAX-WS services hosted in CXF.
- cxf:bean:cxfEndpoint[?options]
- cxf:*//someAddress[?options]*

Execute Camel Routes

- Create new class with public static void main method.
- Create CamelContext object in public static void main method of class.
- Add Route definition.
- Start the CamelContext
- Wait for sometime or infinitely.
- CamelContext stop.
- Shutdown the CamelContext.
- More control of CamelContext activities.

Execute with Camel Main

- Create the object of camel Main.
- Call main.run(args) to run the CamelContext indefinitely.
- The main.run(args) is blocking method, it will never run any code after this.
- Use separate thread to signal the main object for stopping and shutdown.

Execute the Route with XML DSL

- Define the route definition in XML with spring bean dependencies with context and routes and any beans.
- Load the context, in main method with Spring context with xml,it automatically executes the routes in xml.

Execute the xml route with Spring main

- The spring Main class checks for xml definitions in resources/META-INF/spring folder and loads and executes automatically when you run camel:run maven goal with camel-maven-plugin in maven project.
- The spring Main is from camel-spring-main jar module.

The camel-maven-plug-in

- Automatically executes the route definitions from xml files in resources/META-INF/spring/*.xml.
- Can be configured to execute multiple xml route definitions from diff path locations.
- There should be only one camel context defined across the xml files and others may be bean definitions.

Loading java DSL in xml

- The route execution main class can be configured with camel-maven-plugin as mainClass.
- The java routebuilder bean can be specified in configuration.
- The java routebuilder class package can be specified in configuration.
- The java routebuilder class can be scanned from packages specified in configuration.

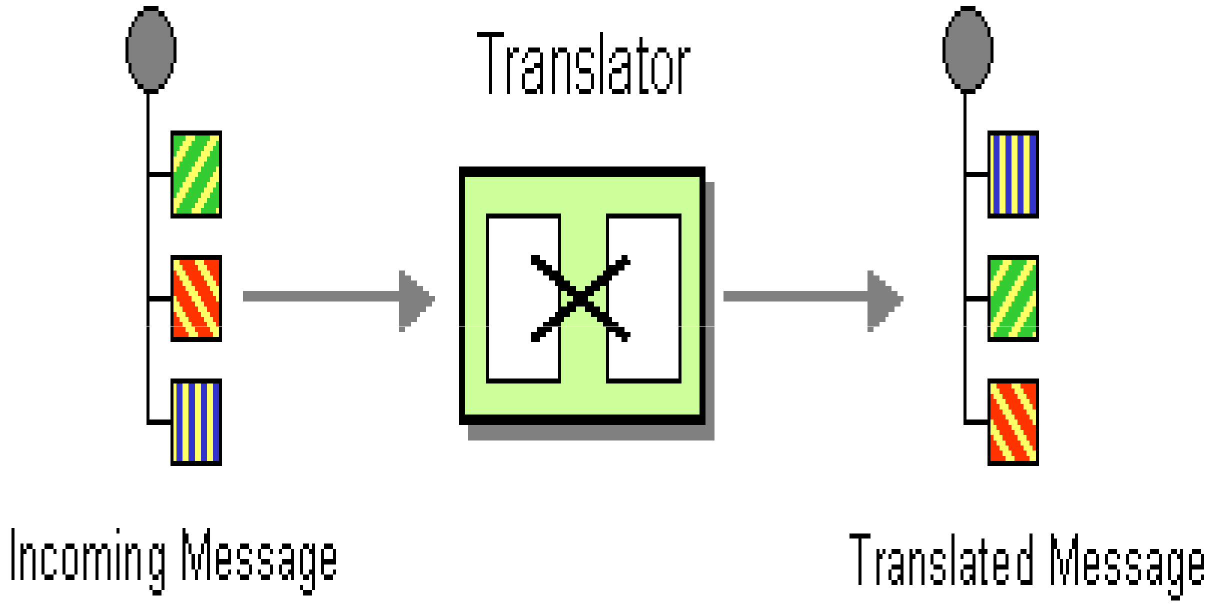
Execute XML route DSL from web application

- With the Camel servlet configured with url mapping in web.xml will load the xml routes files configured as context files and execute on demand on url invocation.
- With the Spring Context listener configured in web.xml will load the xml routes and execute automatically on startup.

Camel in SpringBoot application

- The spring Boot component provides auto-configuration for Apache Camel.
- The auto-configuration of the Camel context auto-detects Camel routes available in the Spring context and executes automatically on startup.
- It also registers the key Camel utilities (like producer template, consumer template and the type converter) as beans and executes the camel context.

Message Translator



Camel with Message Translator

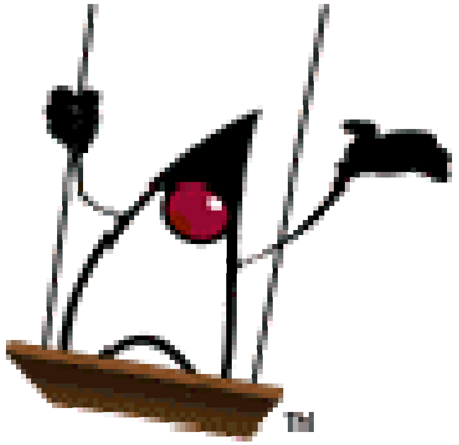
- Camel supports the Message Translator EIP pattern by using a Processor in the routing logic.
- By using a bean to perform the transformation, or by using transform() in the DSL.
- Also supported a Data Format to marshal and un-marshal messages in different encodings.

Java DSL for Message

- from("direct:start").
- process(new Processor() {
-   public void process(Exchange exchange) {
- Message in = exchange.getIn(); in.setBody(in.getBody(String.class) + " World!"); } }).
- to("mock:result");
- from("direct:start").transform(body().append(" World!")).to("mock:result");

XML DSL for Message

- <camelContext xmlns="http://camel.apache.org/schema/spring">
- <route>
- <from uri="direct:start"/>
- <transform> <simple>${in.body} extra data!</simple>
- </transform>
- <to uri="mock:end"/>
- </route>
- </camelContext>



Thank You!