BPM in Enterprise Systems Integration

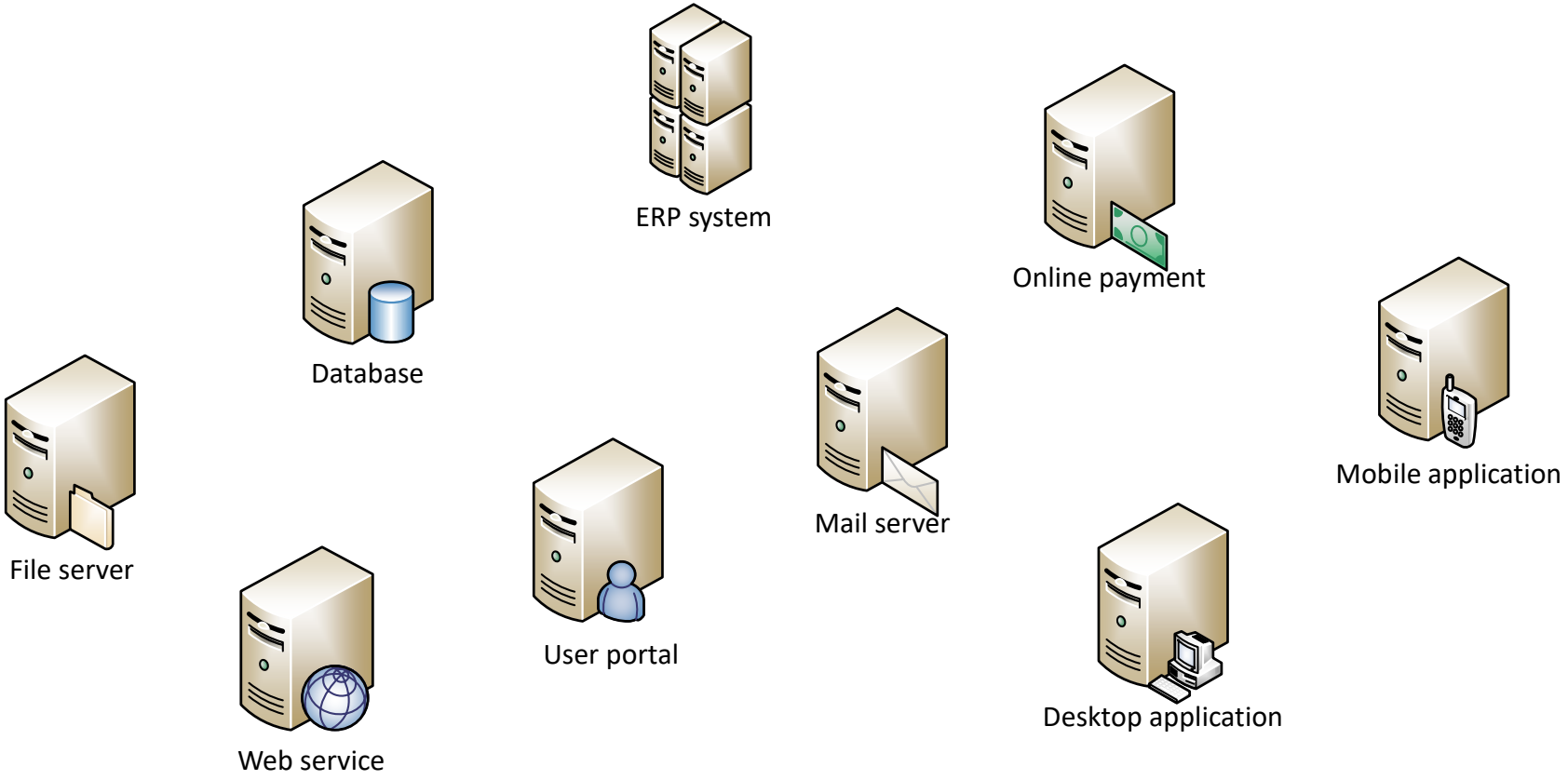
Diogo R. Ferreira

Instituto Superior Técnico (IST)

Universidade de Lisboa

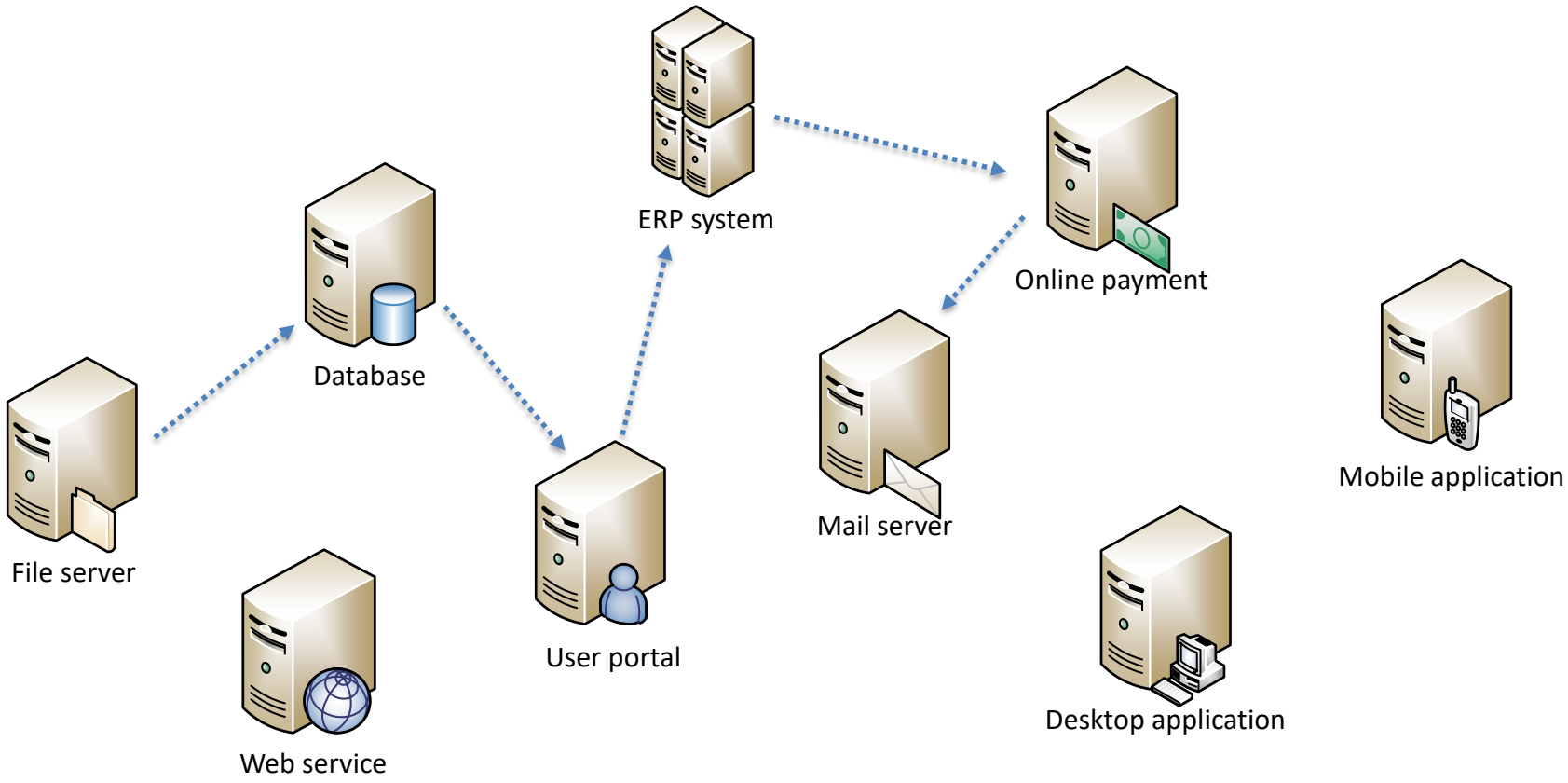
Introduction

- Enterprise Systems Integration
  - many kinds of systems which must be integrated



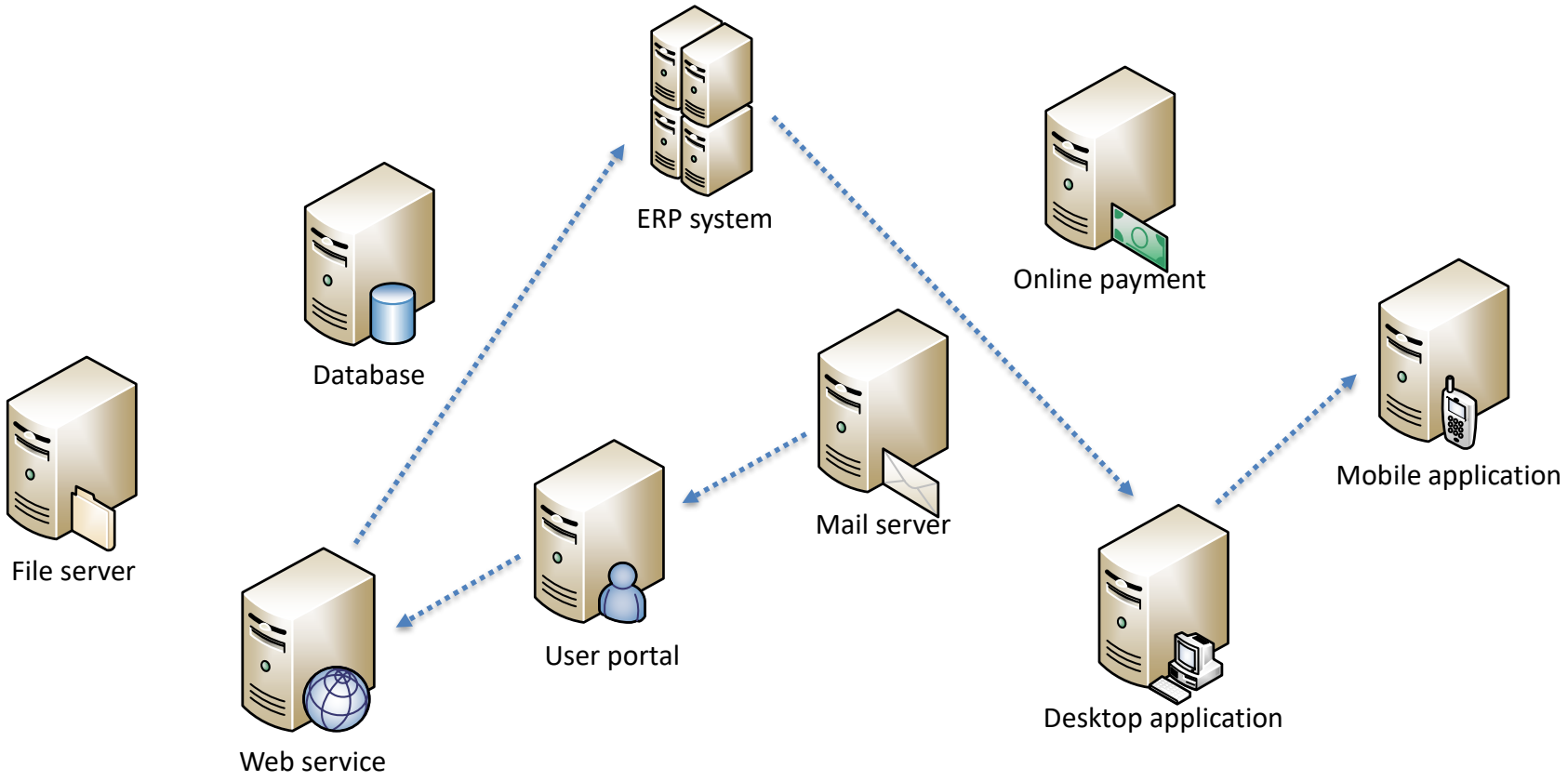
Introduction

- Enterprise Systems Integration
  - integration depends on business processes



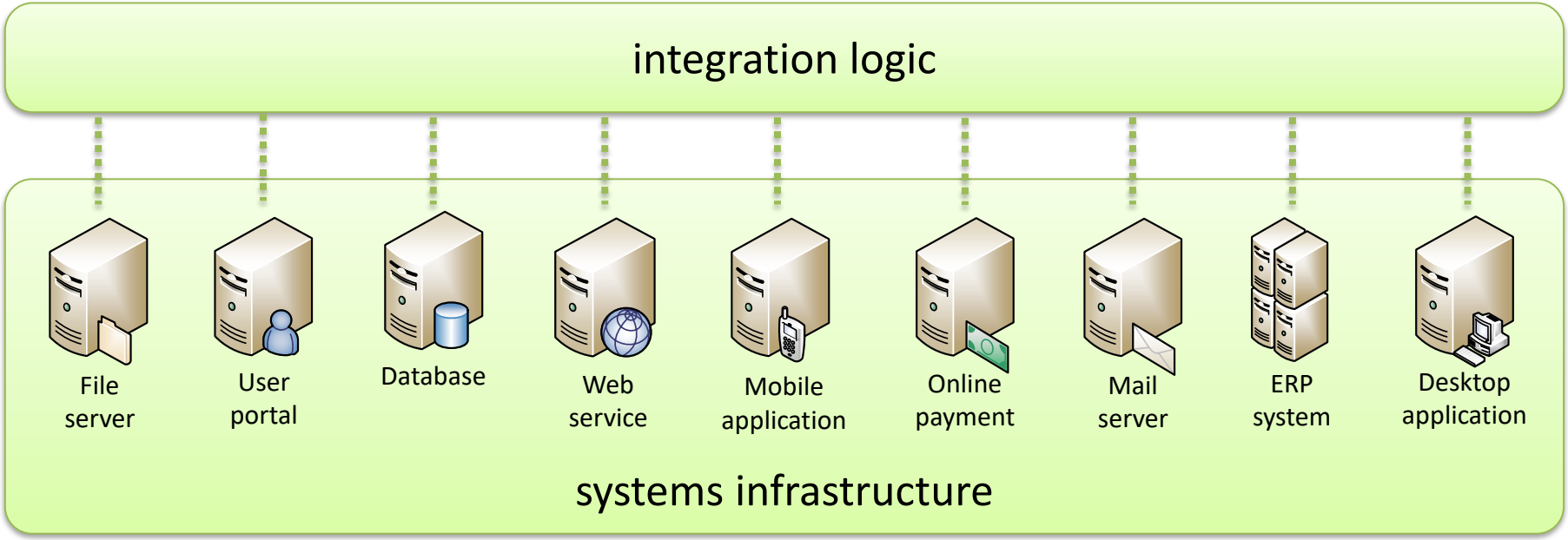
Introduction

- Enterprise Systems Integration
  - integration depends on business processes



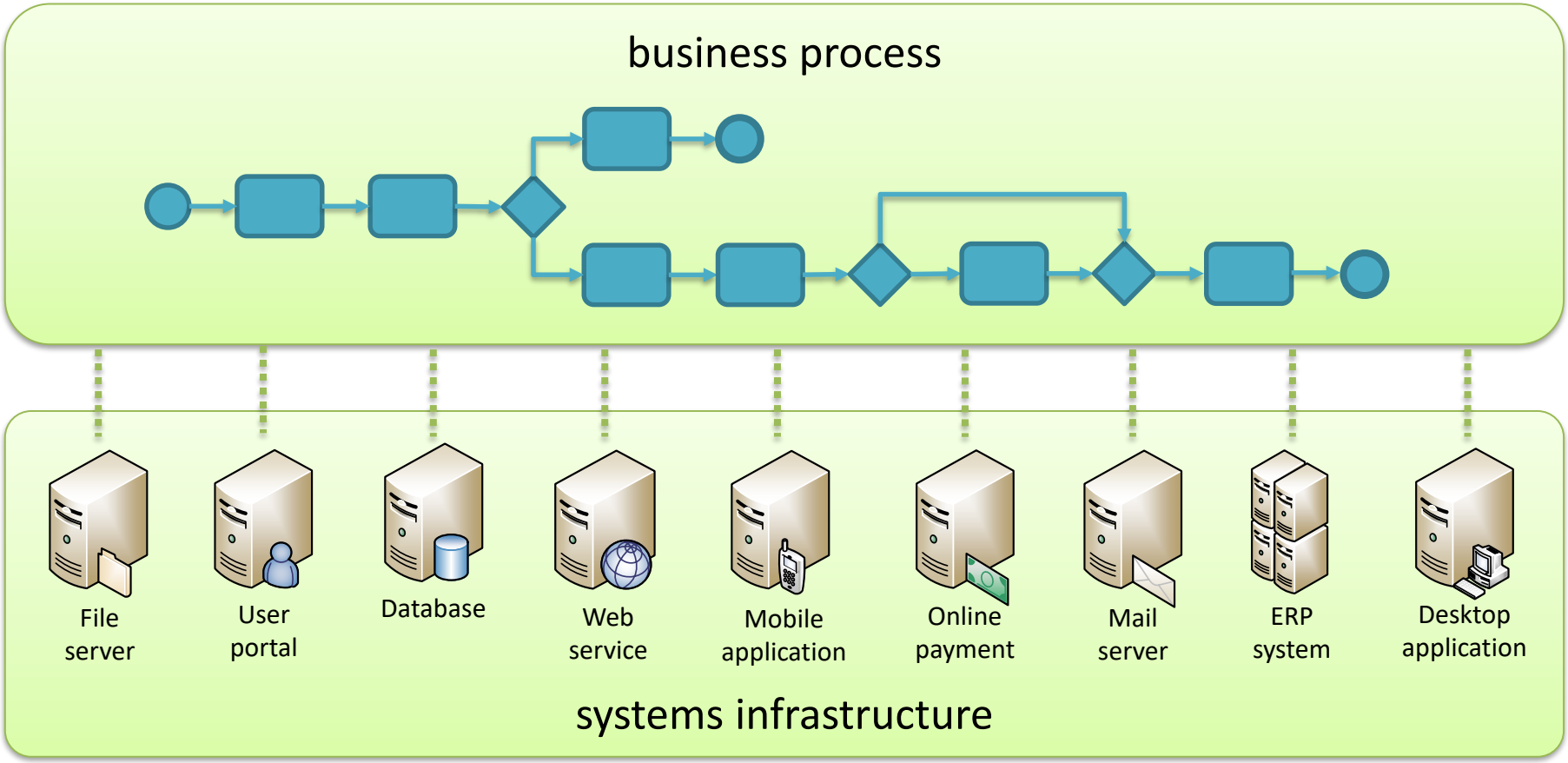
Introduction

- Enterprise Systems Integration
  - easier to integrate if the integration logic can be defined separately from the systems



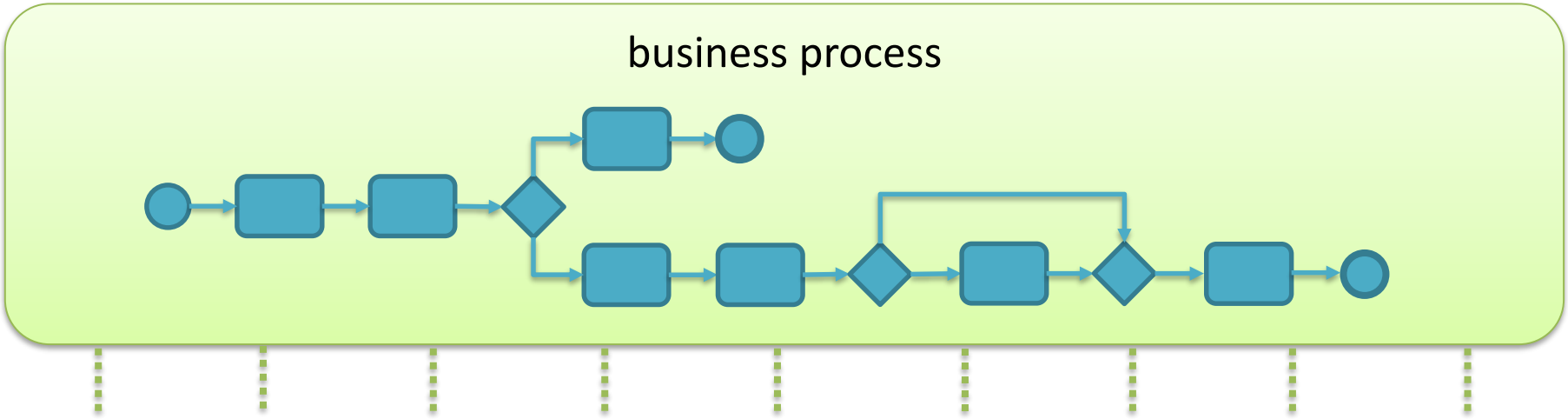
Introduction

- Enterprise Systems Integration
  - the integration logic is the business process



Introduction

- Such business process is an **orchestration**
  - typically, an orchestration:
    - **receives** a message from a system
    - **transforms** the message into another format/structure
    - **sends** the message to another system
  - BPM-like flow constructs are possible



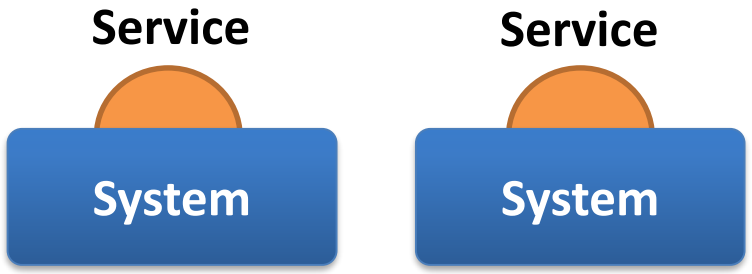
Introduction

- In the past
  - the business process is just a conceptual view
  - implementation is done at the systems level
- In the present
  - the orchestration is the implementation of a process
  - systems infrastructure can be adapted to fit the process

Introduction

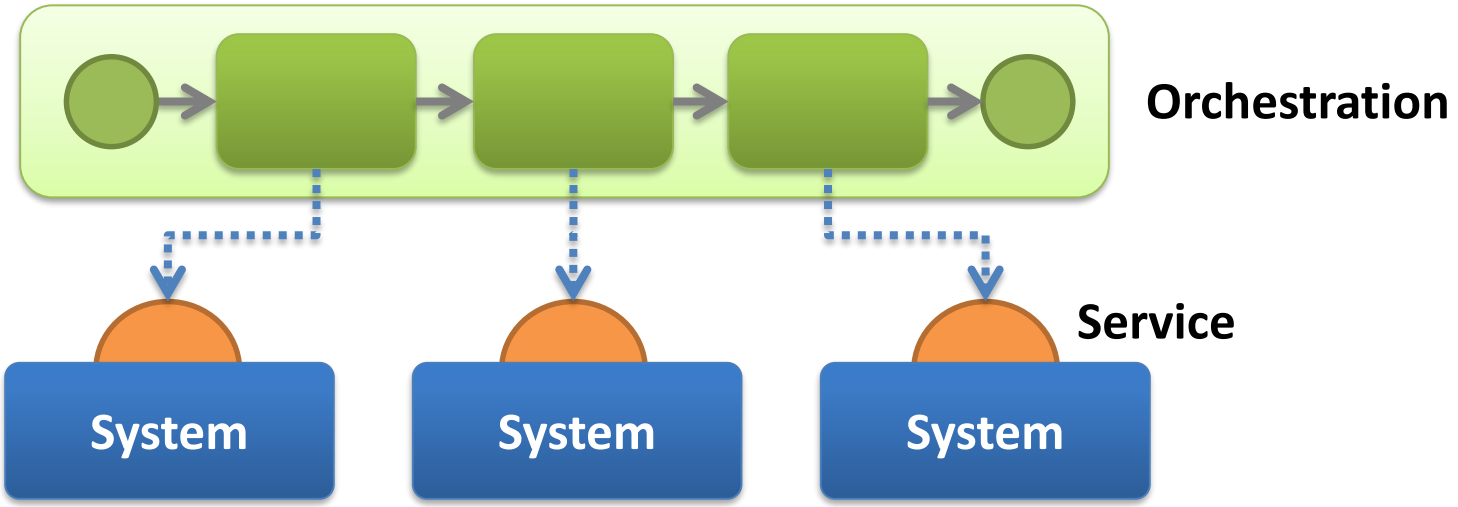
- Key concepts
  - **orchestration**
    - an executable model of a business process

  - **service**
    - an abstraction of some system functionality



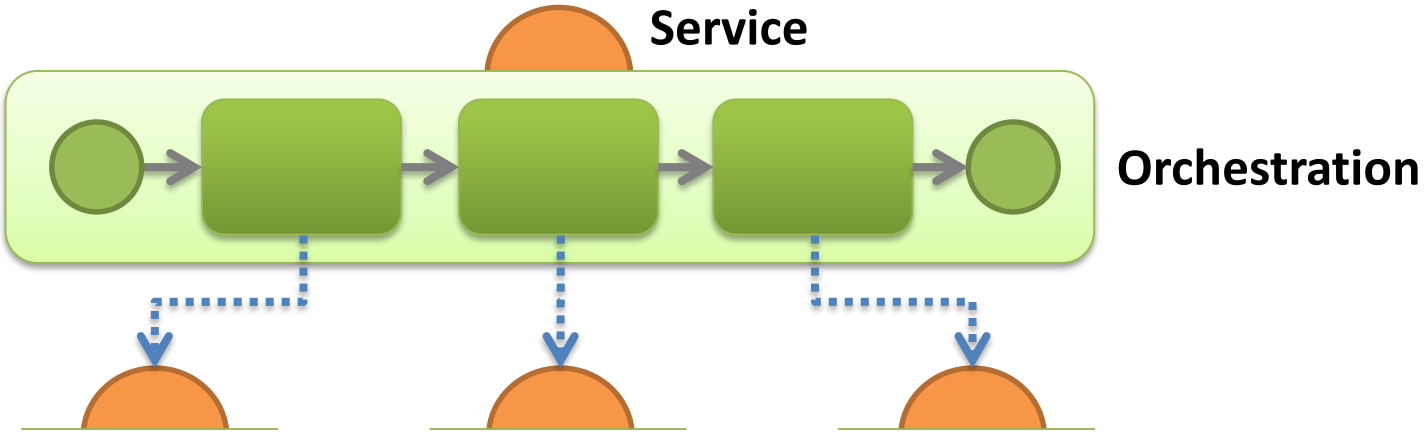
Introduction

- Why services and orchestrations are so important
  - services allow us to create a different landscape over existing systems
  - orchestrations allow us to implement a business process over existing systems/services



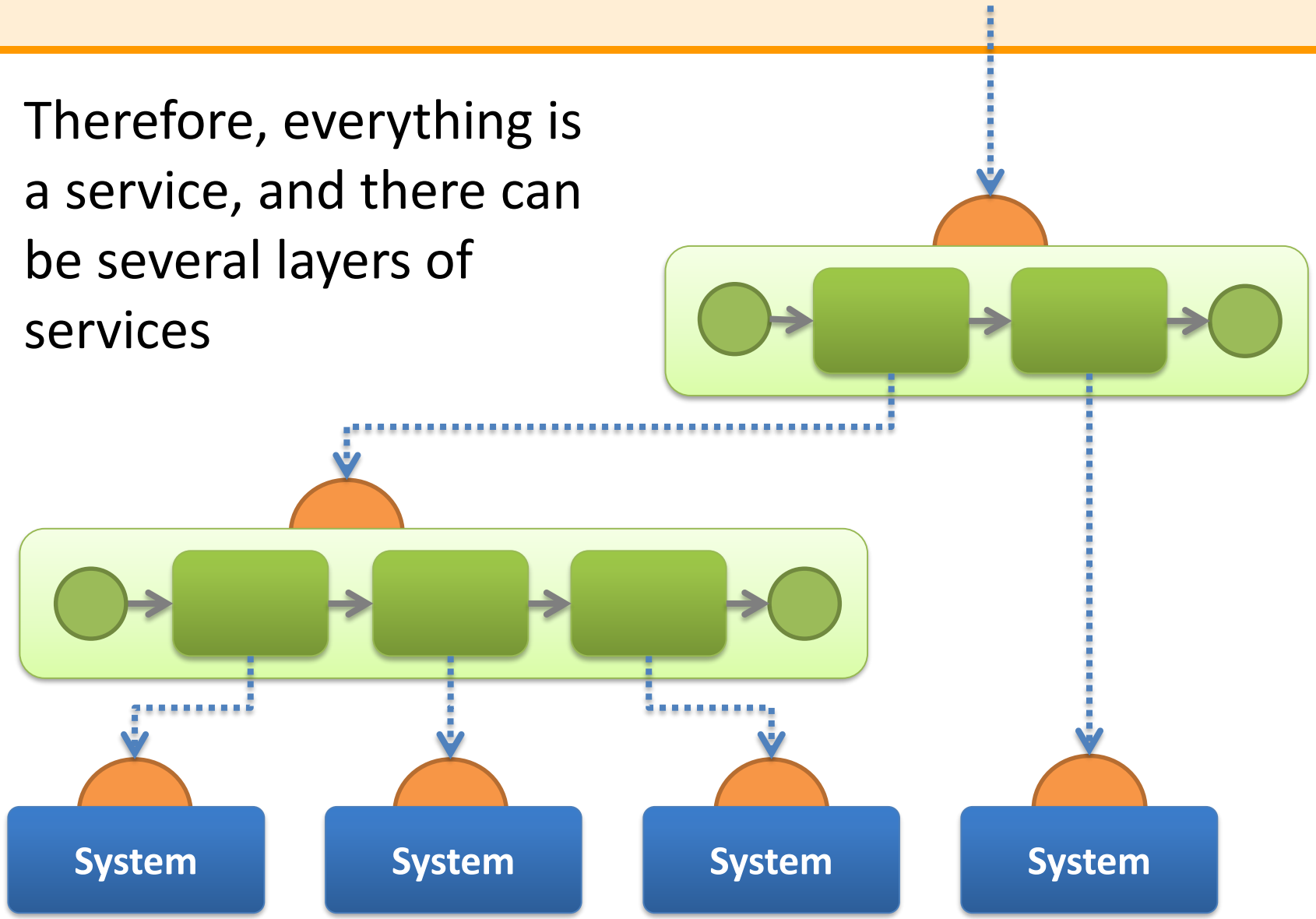
Introduction

- Furthermore
  - the concepts of service and orchestration are interchangeable
    - an orchestration can be exposed as a service
    - a service can be implemented as an orchestration of other services



Introduction

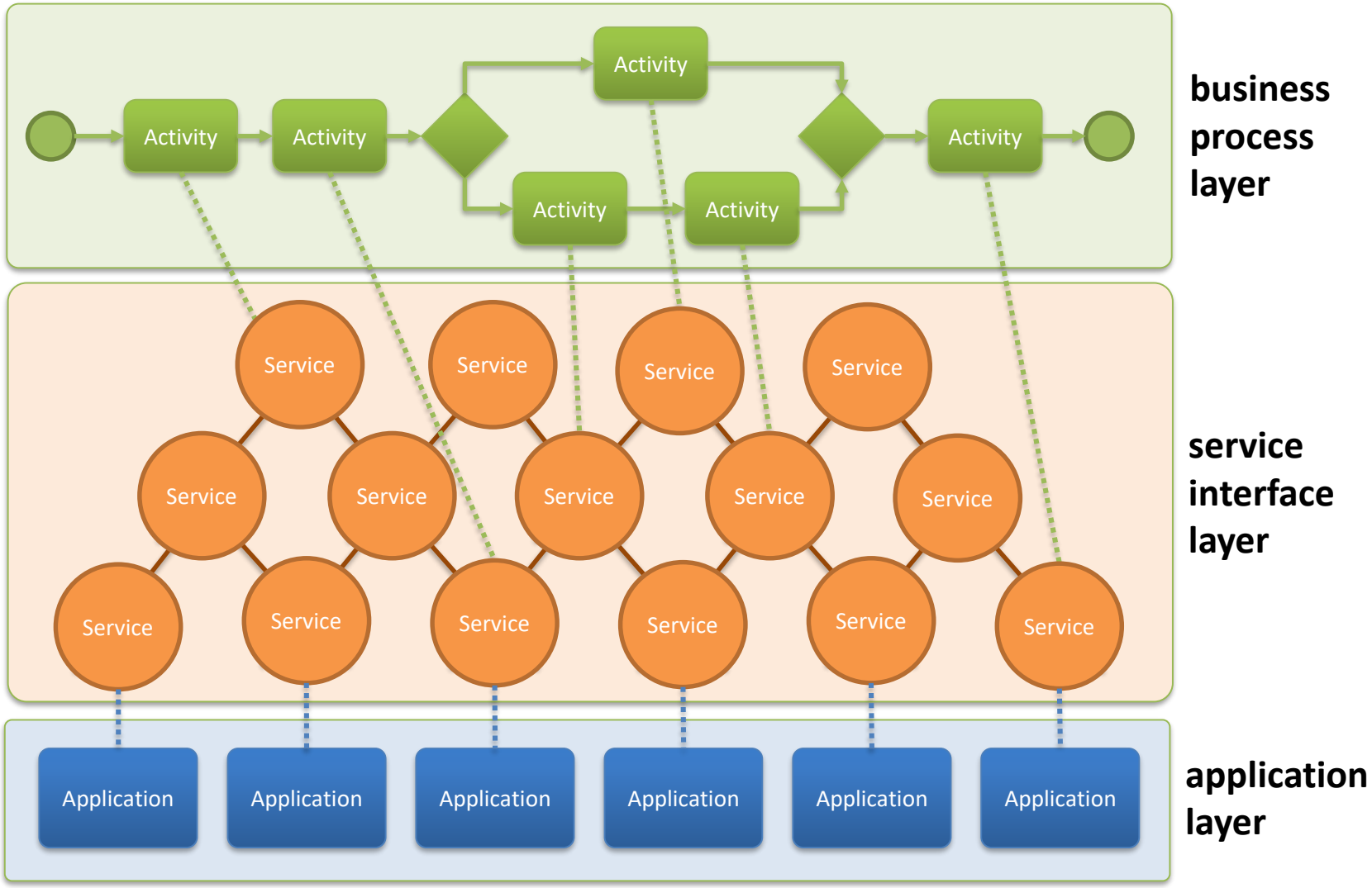
- Therefore, everything is a service, and there can be several layers of services



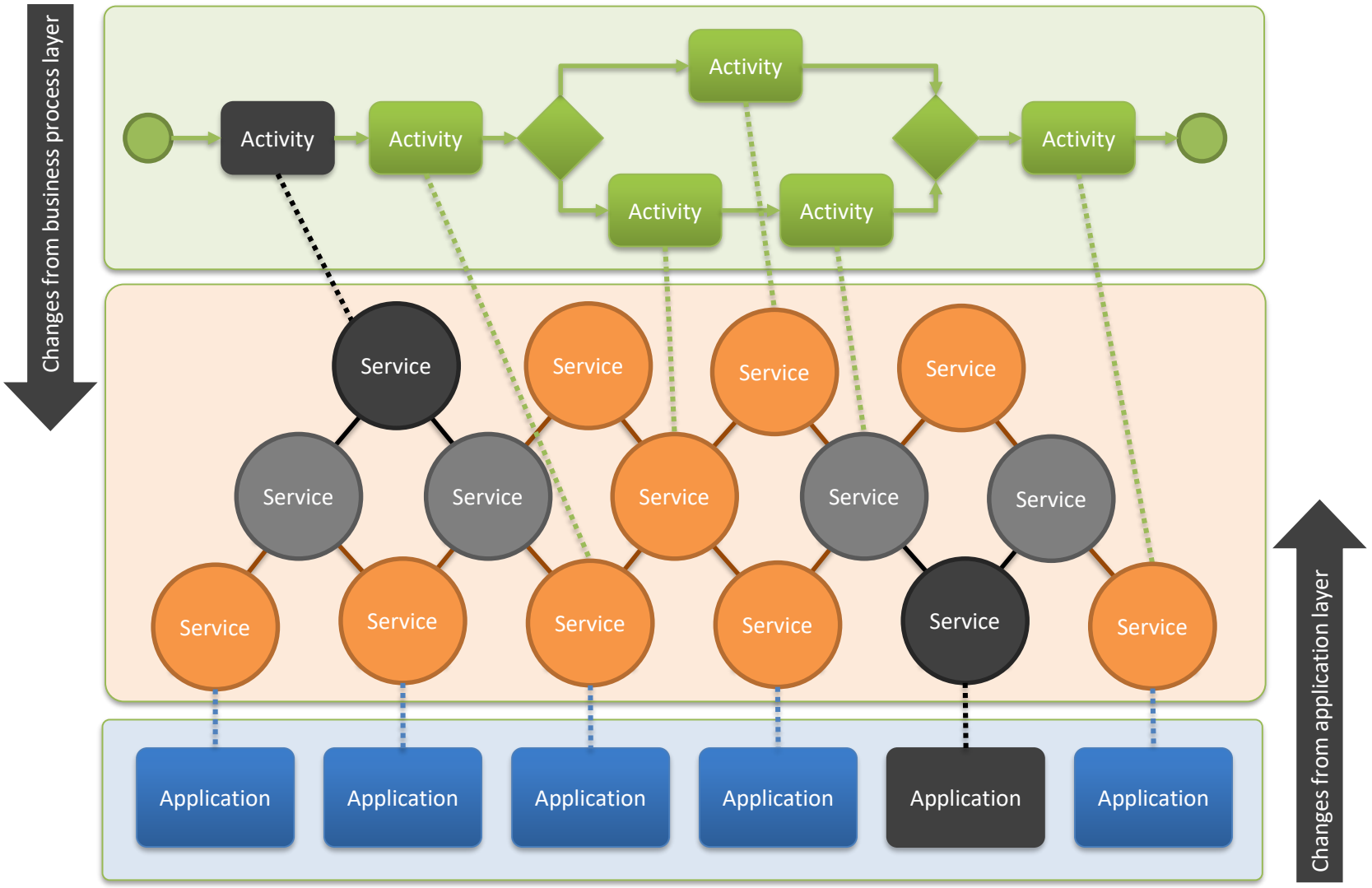
Service-Oriented Architecture

- **A business process** is a top-level service which is implemented as an orchestration of mid-level services which in turn are implemented as orchestrations of low-level services exposed by **systems and applications**.

Service-Oriented Architecture

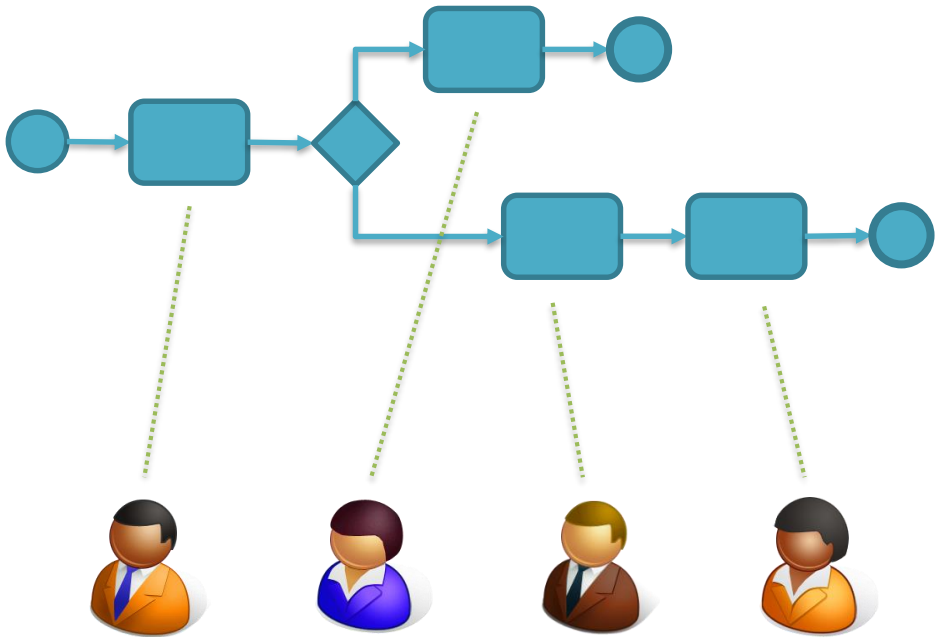


Service-Oriented Architecture



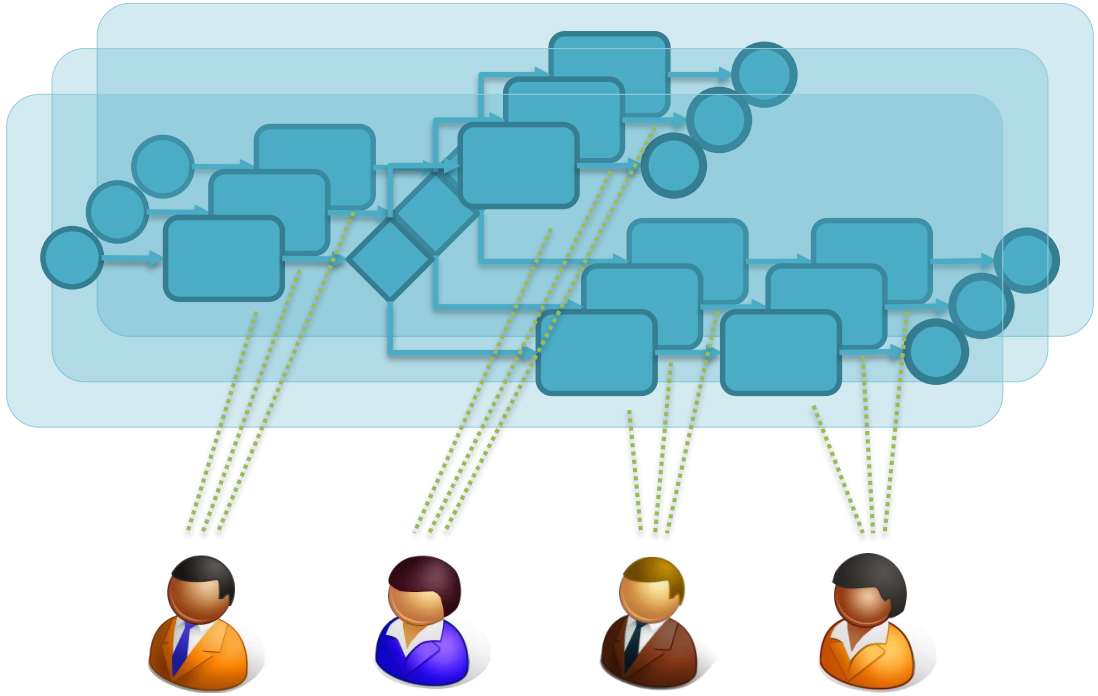
Human workflows

- What happens when processes are performed by people?



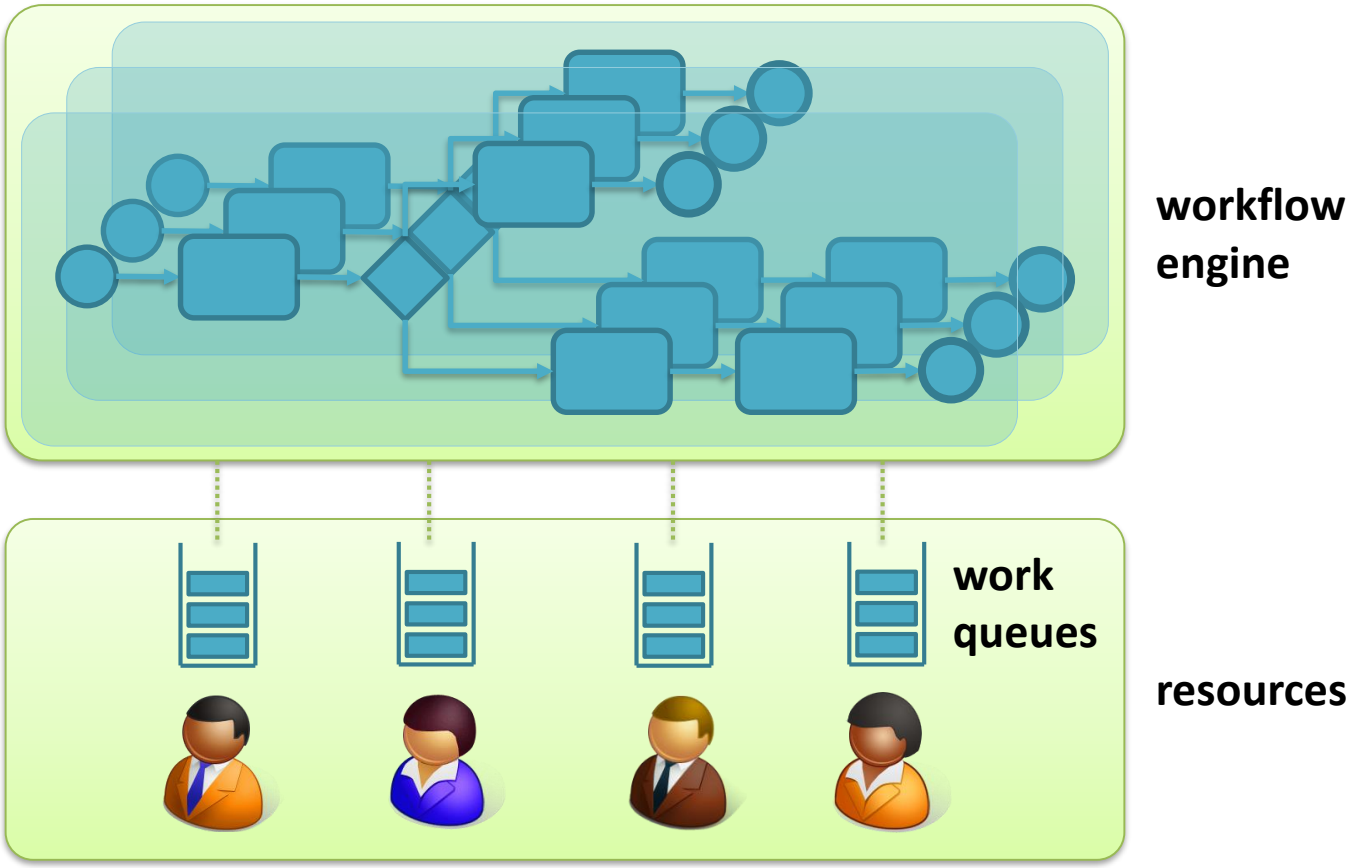
Human workflows

- A process gets instantiated multiple times



Human workflows

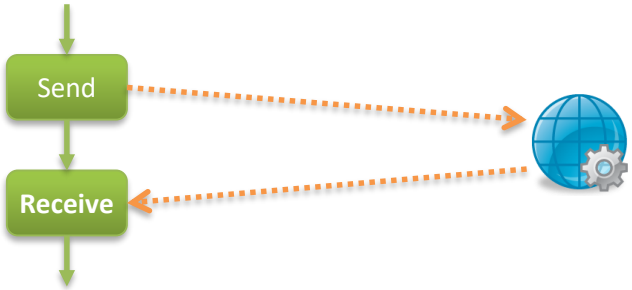
- Tasks lists, to-do lists, or work queues



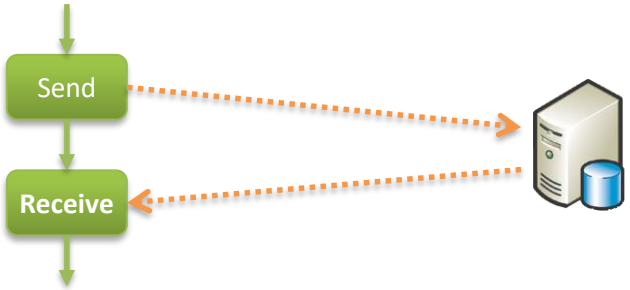
Comparison

Human Workflows	Service Orchestrations
Process model	Orchestration
Process instance	Orchestration instance
Activity • human task	Activity • service/system invocation
Resource • human	Resource • service (e.g. Web service) • database (or other data sources) • etc. (any other application)
Work queue	Message queue
Flow constructs • branching • parallelism • loops	Flow constructs • branching • parallelism • loops
	Advanced mechanisms • exception handling • transactions and compensation

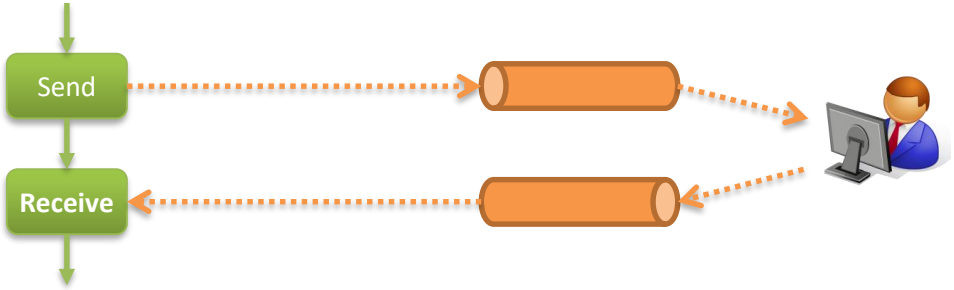
Examples



Web service

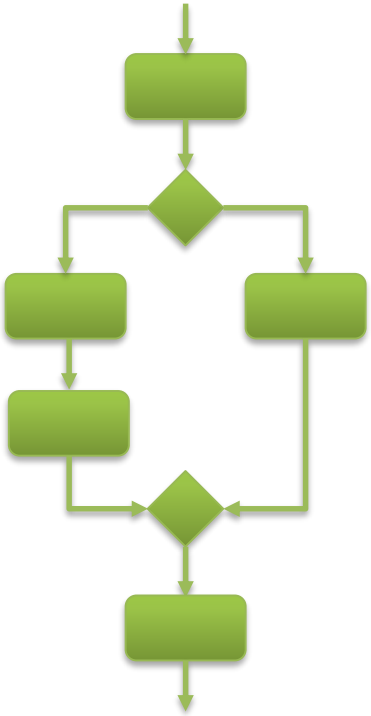


Database

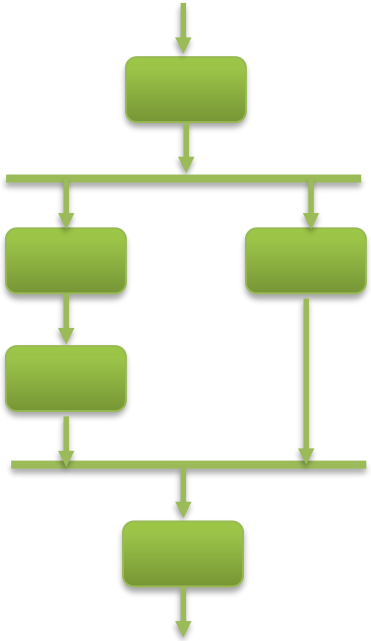


Message queues

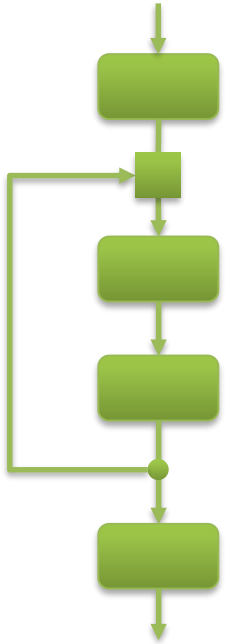
Examples



Branching



Parallelism

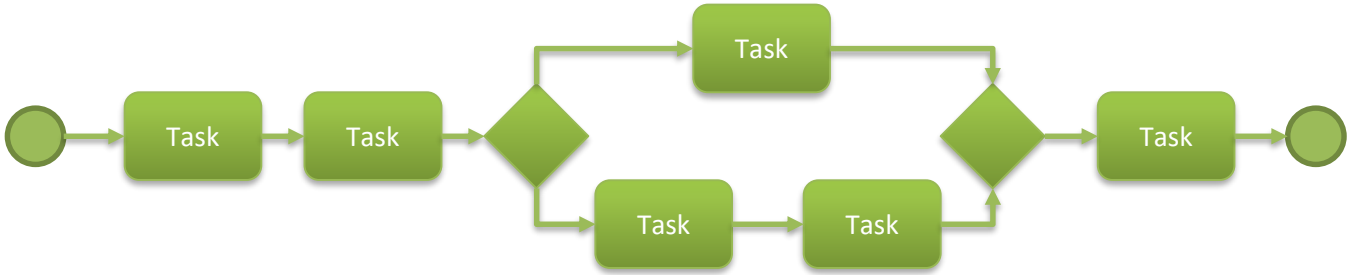


Loop

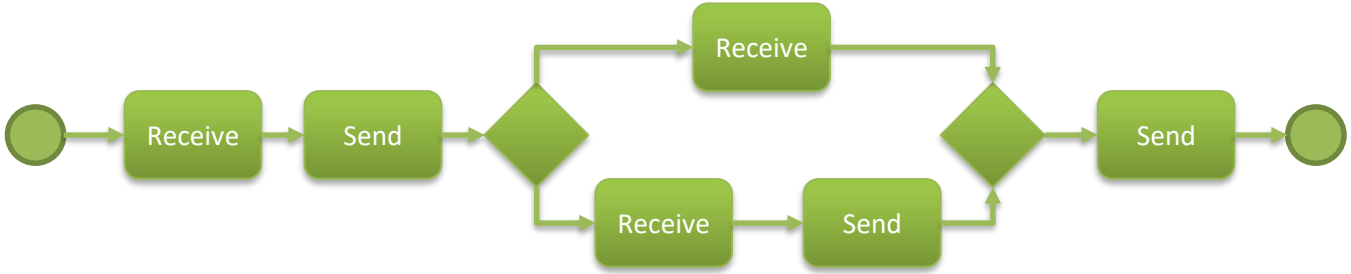
Concepts of orchestration flow

Activities

- In business processes, activities are (mostly) tasks

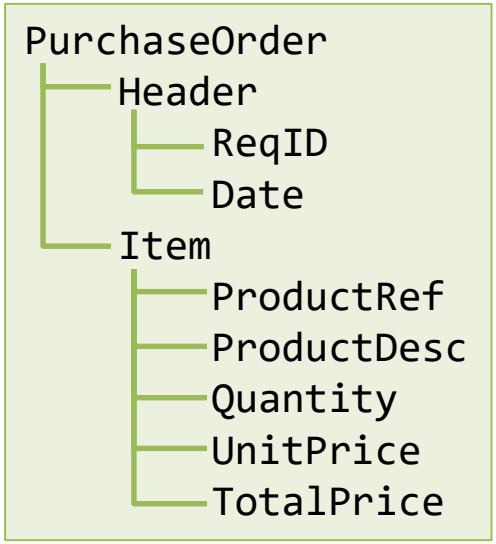


- In orchestrations, activities are (mostly) message exchanges



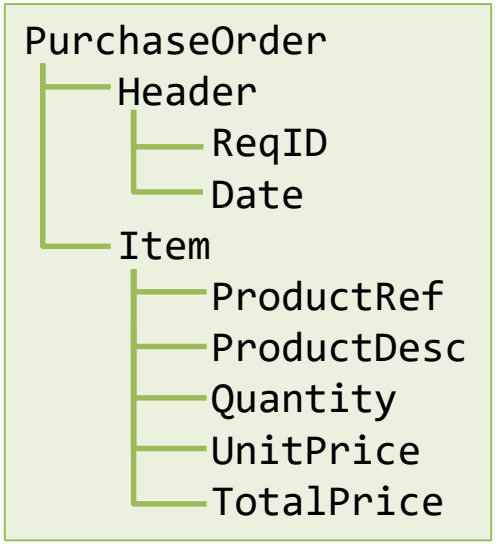
Messages

- What is a message?
  - a message is some structured data
    - it may be a request sent to an external system
    - it may be a response received from an external system
  - usually, it takes the form of an XML document
    - and it may be depicted as a tree structure



Messages

- Message ***schema*** vs. message ***instance***




```
<PurchaseOrder>
    <Header>
        <ReqID>R1</ReqID>
        <Date>2014-07-08</Date>
    </Header>
    <Item>
        <ProductRef>537</ProductRef>
        <ProductDesc>Printer cartridge</ProductDesc>
        <Quantity>2</Quantity>
        <UnitPrice>24.99</UnitPrice>
        <TotalPrice>49.98</TotalPrice>
    </Item>
</PurchaseOrder>
```

  - the schema is usually defined in XSD (XML Schema Definition)

Messages

- Some platforms use XML namespaces

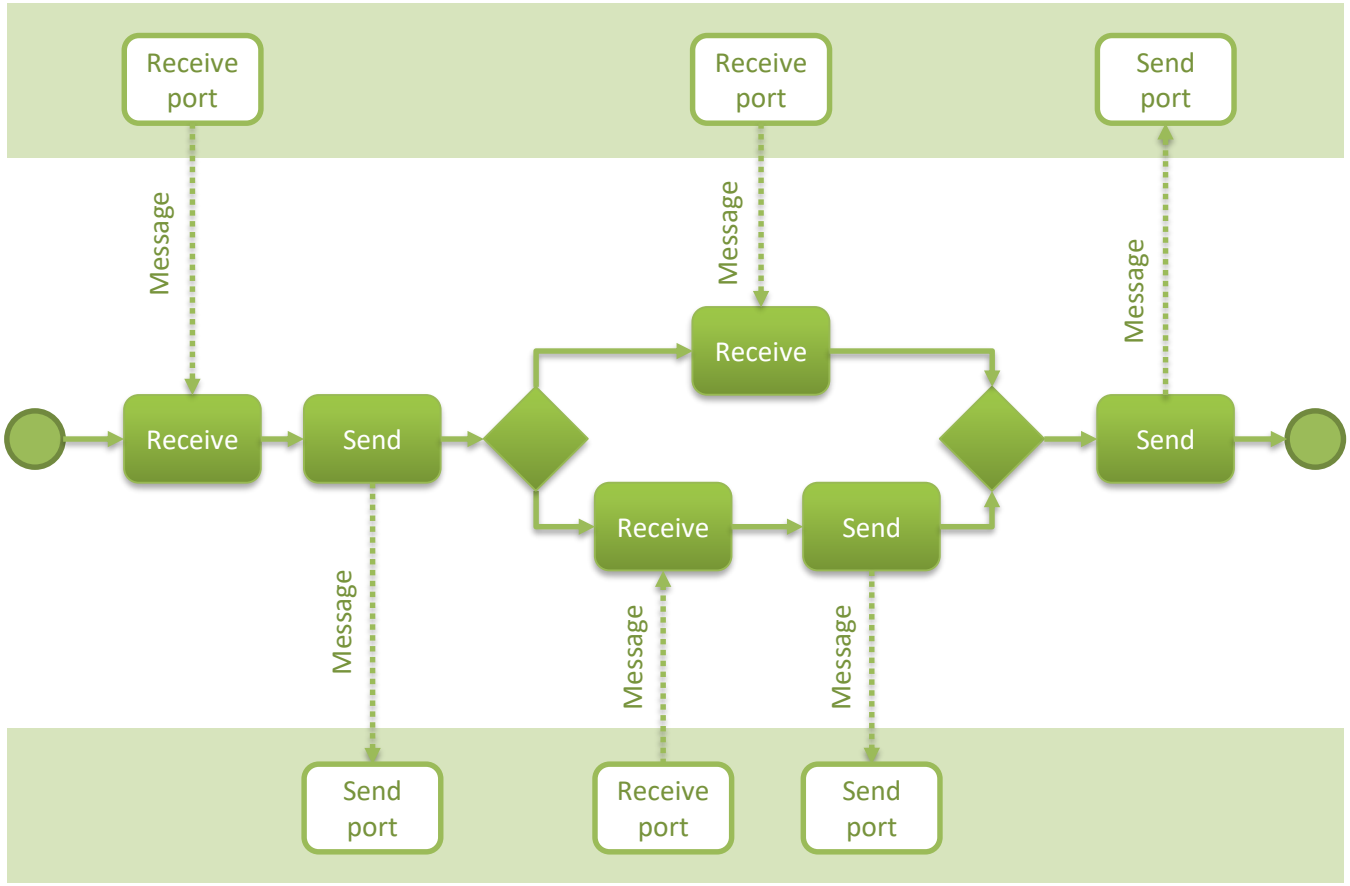
```
<?xml version="1.0" encoding="utf-8"?>
<ns0:PurchaseOrder xmlns:ns0="http://OfficeSupplies.PurchaseOrder">
    <Header>
        <ReqID>R1</ReqID>
        <Date>2014-07-08</Date>
    </Header>
    <Item>
        <ProductRef>537</ProductRef>
        <ProductDesc>Printer cartridge</ProductDesc>
        <Quantity>2</Quantity>
        <UnitPrice>24.99</UnitPrice>
        <TotalPrice>49.98</TotalPrice>
    </Item>
</ns0:PurchaseOrder>
```

  - the type of message is then

```
http://OfficeSupplies.PurchaseOrder#PurchaseOrder
```

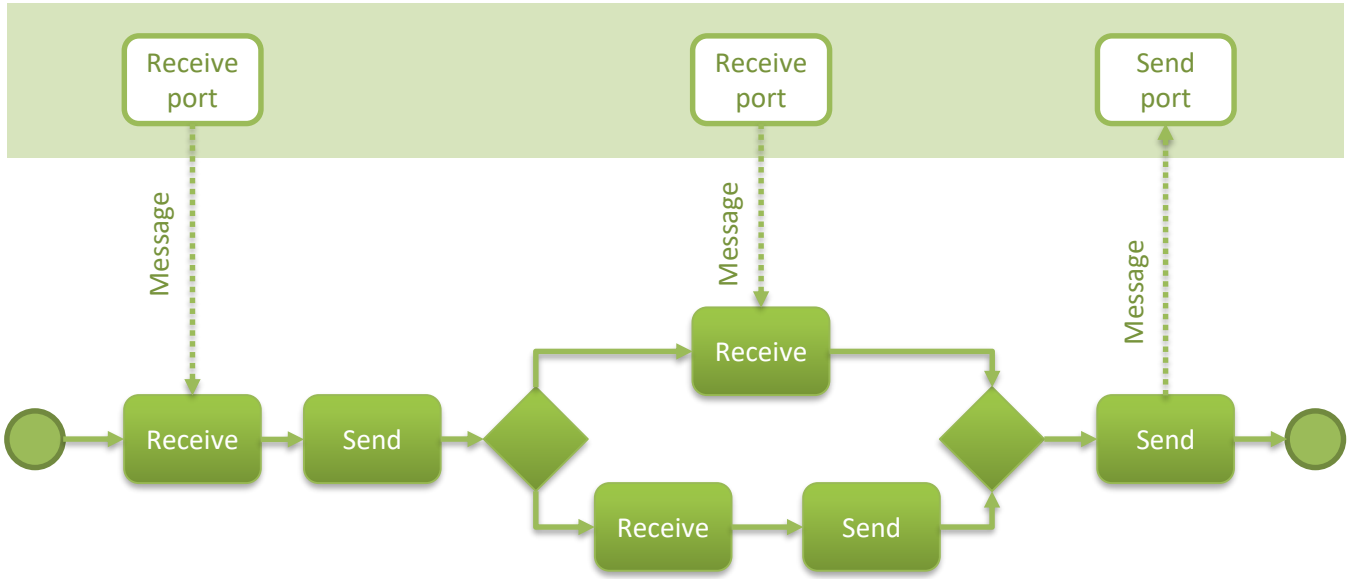
Ports

- Messages are sent or received through ports



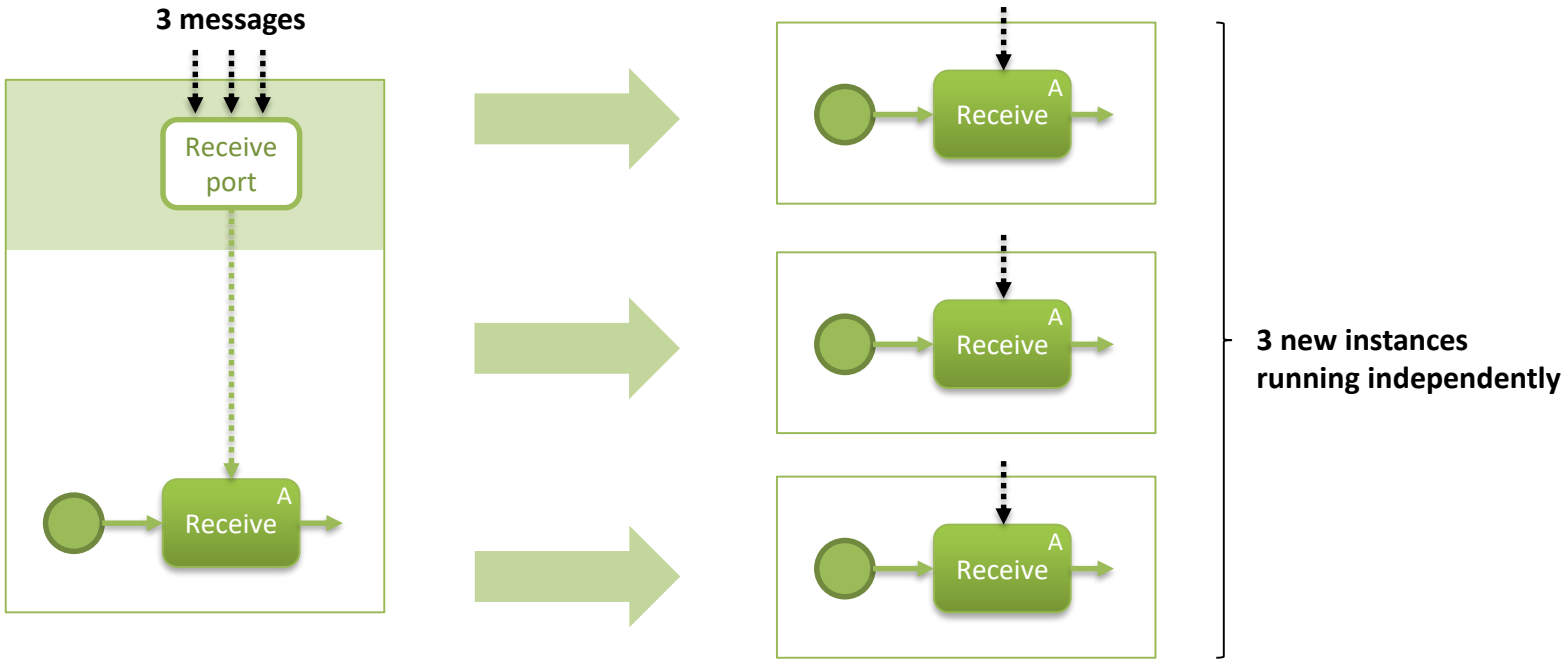
Ports

- A port is a connection to an external system
  - i.e. a Web service, a database, a message queue, etc.
  - communication is configured in the port itself, and not in the orchestration



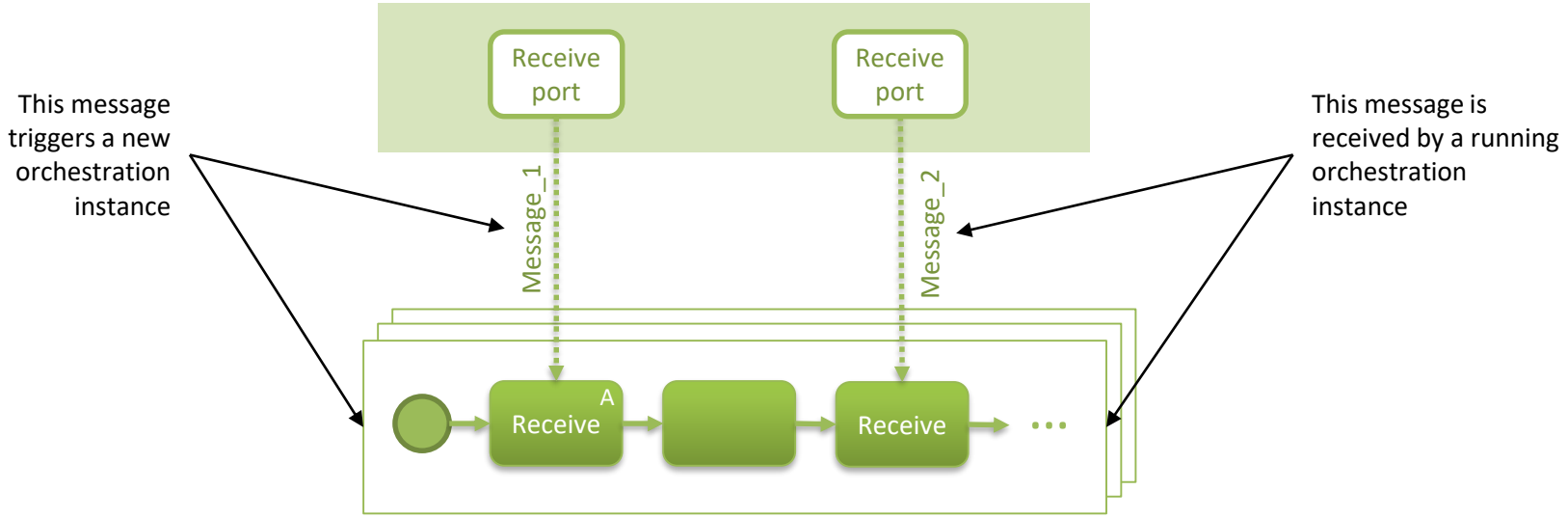
Activating receive

- An orchestration begins with the arrival of a message
  - the first receive creates new orchestration instances and is called the *activating receive*



Receive

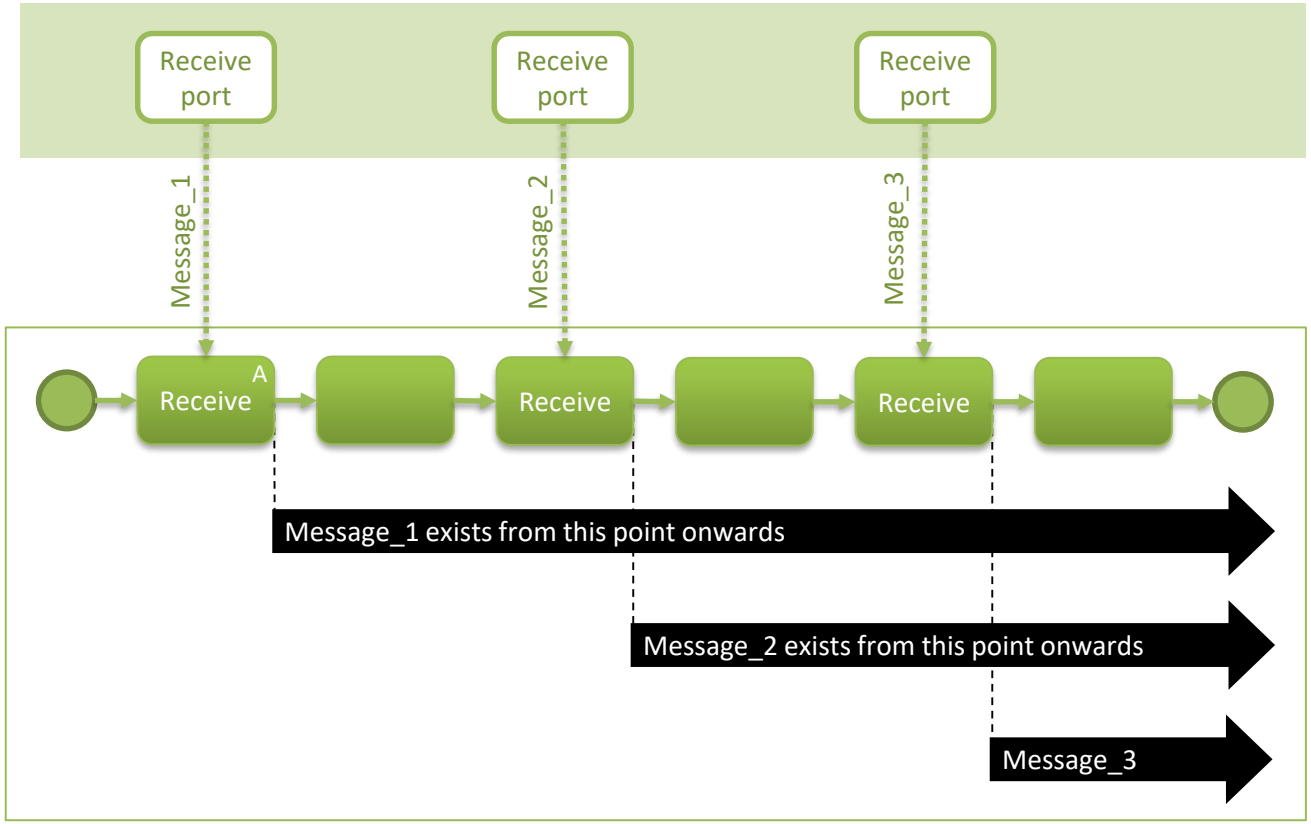
- A normal receive executes in an existing instance



  - but how do we know which instance should receive Message_2 ?
    - more on this later…

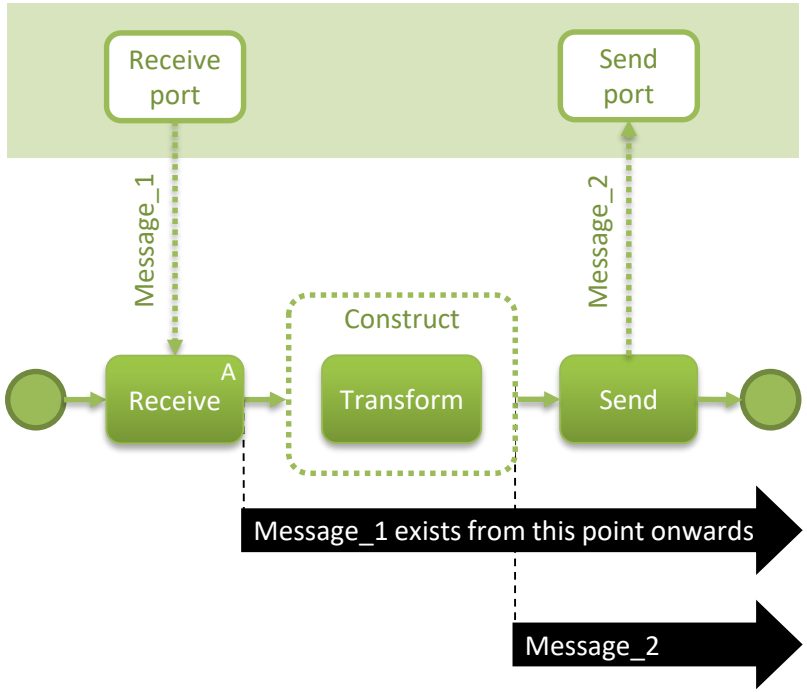
Messages

- Once a message is received, it cannot be changed
  - but the message lives through the entire orchestration



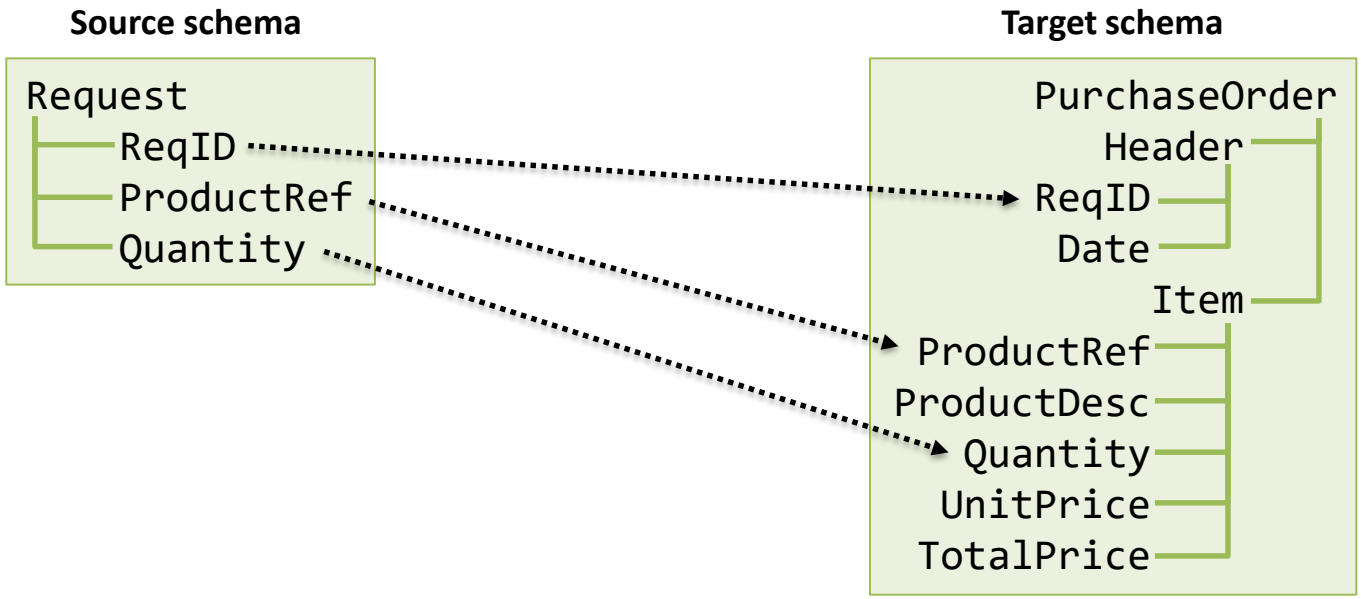
Message construction

- It is possible to create new messages through ***transformation*** of existing ones



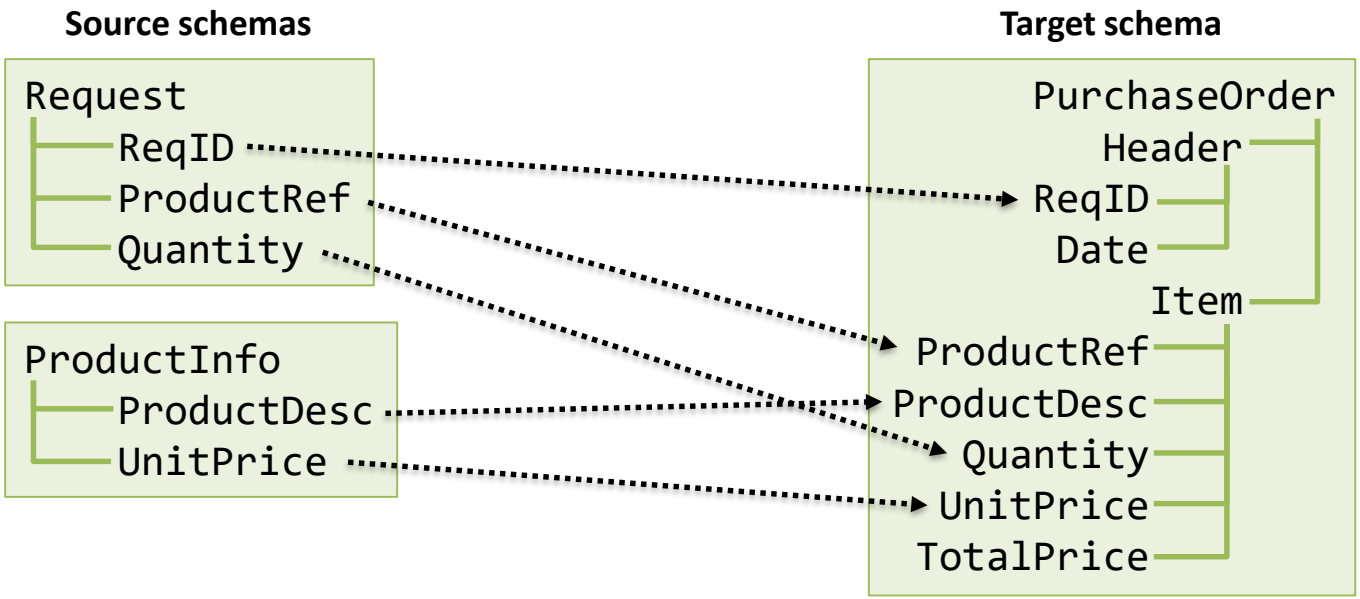
Message transformation

- How does transformation work?
  - messages can be transformed using ***transformation maps***
  - for XML messages, the map can be specified using XSLT



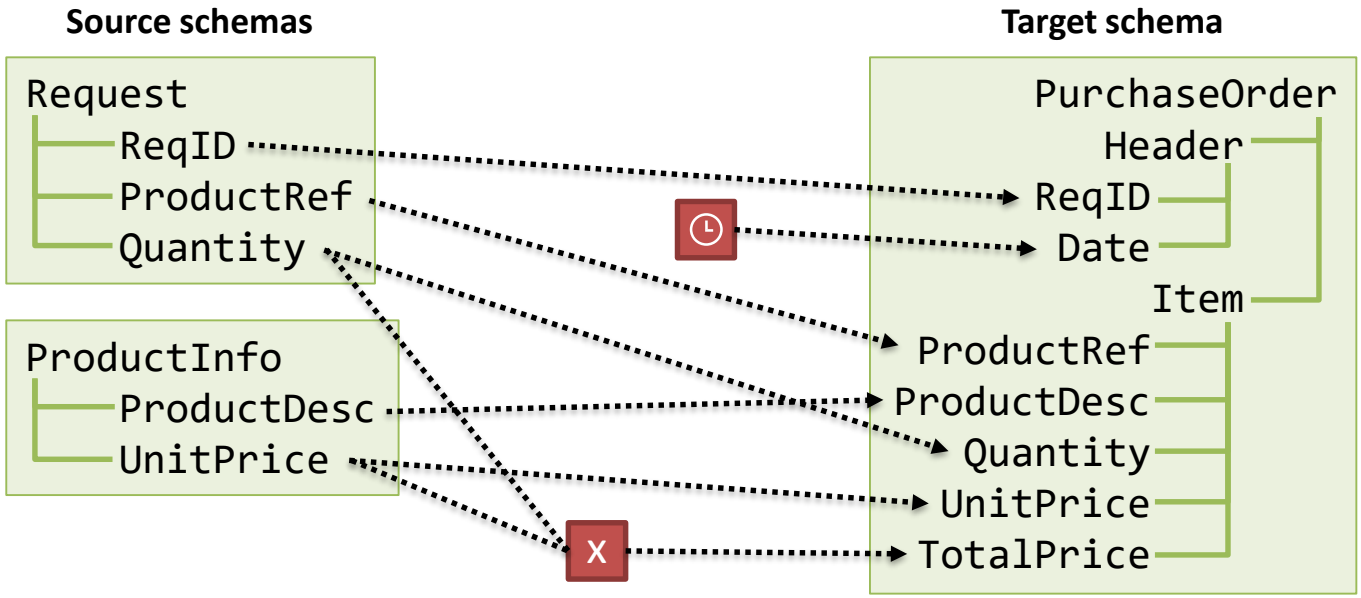
Message transformation

- A map may have several source schemas
  - grabbing data from several source messages



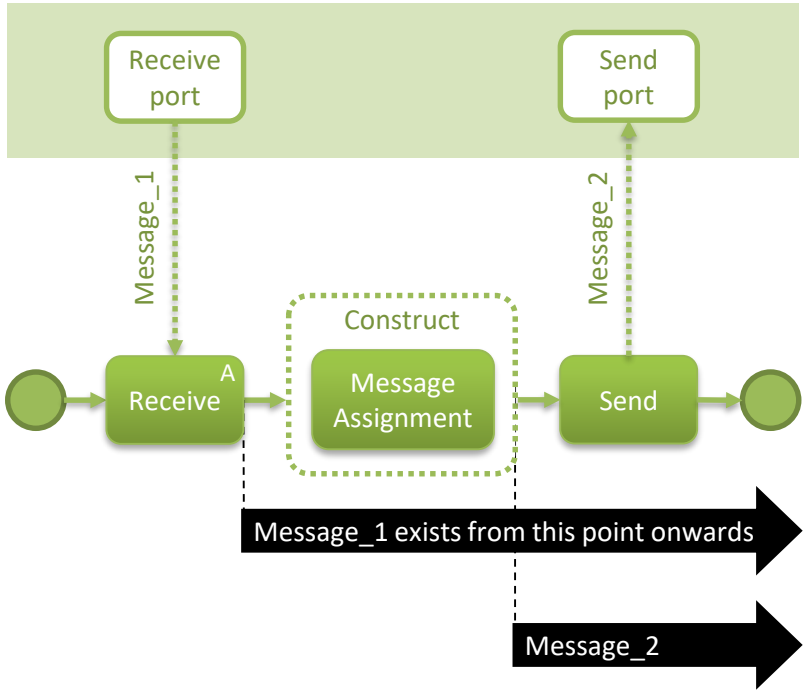
Message transformation

- It is possible to use special functions too
  - in some platforms, these are called ***functoids***
  - can be specified as XSLT functions and operators



Message assignment

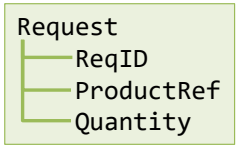
- Alternatively, it may be possible to manipulate messages directly through code
  - in this case we use a **message assignment**



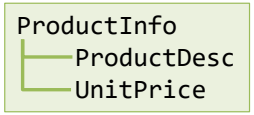
Message assignment

- A message assignment contains code (e.g. C#)
  - the message elements are accessible as properties

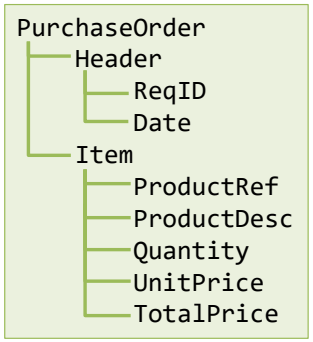
Message_1



Message_2



Message_3



Message Assignment

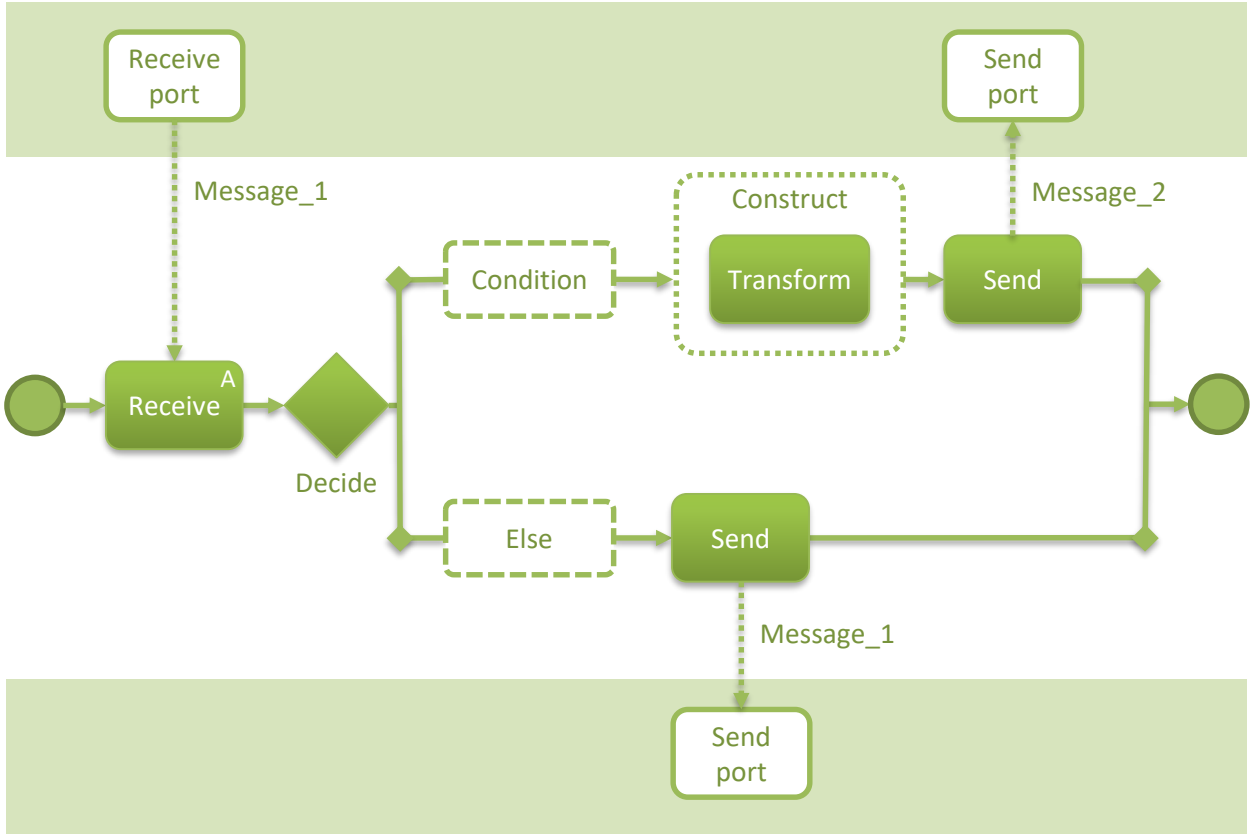
```
Message_3.Header.ReqID = Message_1.ReqID;

Message_3.Header.Date = System.DateTime.Now.ToString("yyyy-MM-dd");

Message_3.Item.ProductRef = Message_1.ProductRef;

Message_3.Item.ProductDesc = Message_2.ProductDesc;

Message_3.Item.Quantity = Message_1.Quantity;

Message_3.Item.UnitPrice = Message_2.UnitPrice;

Message_3.Item.TotalPrice = Message_1.Quantity * Message_2.UnitPrice;
```

Controlling the flow

- We have seen
  - how to receive messages
  - how to send messages
  - how to construct new messages
- Now we will see how to control the flow
  - with branching decisions
  - with parallelism
  - with loops (not addressed)

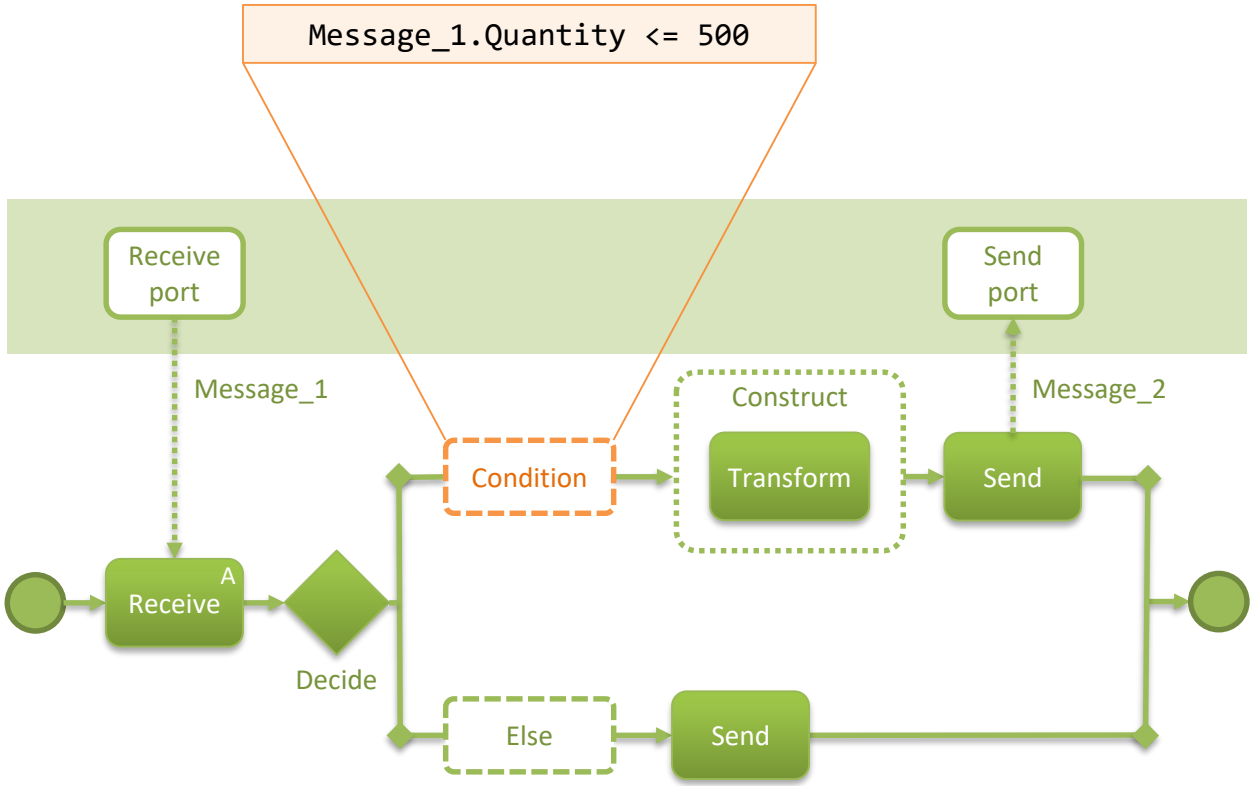
Decisions

- Choosing between alternative paths



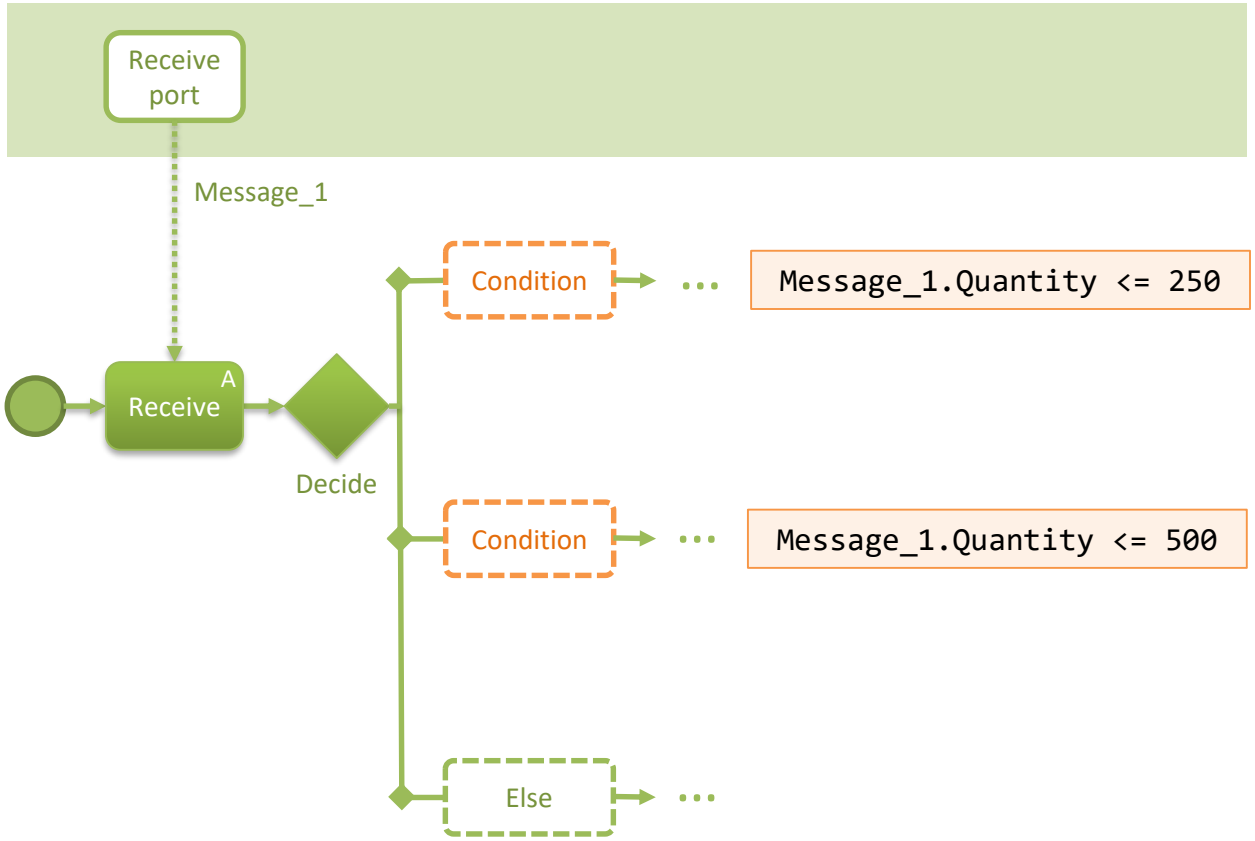
Decisions

- The condition is usually based on message properties



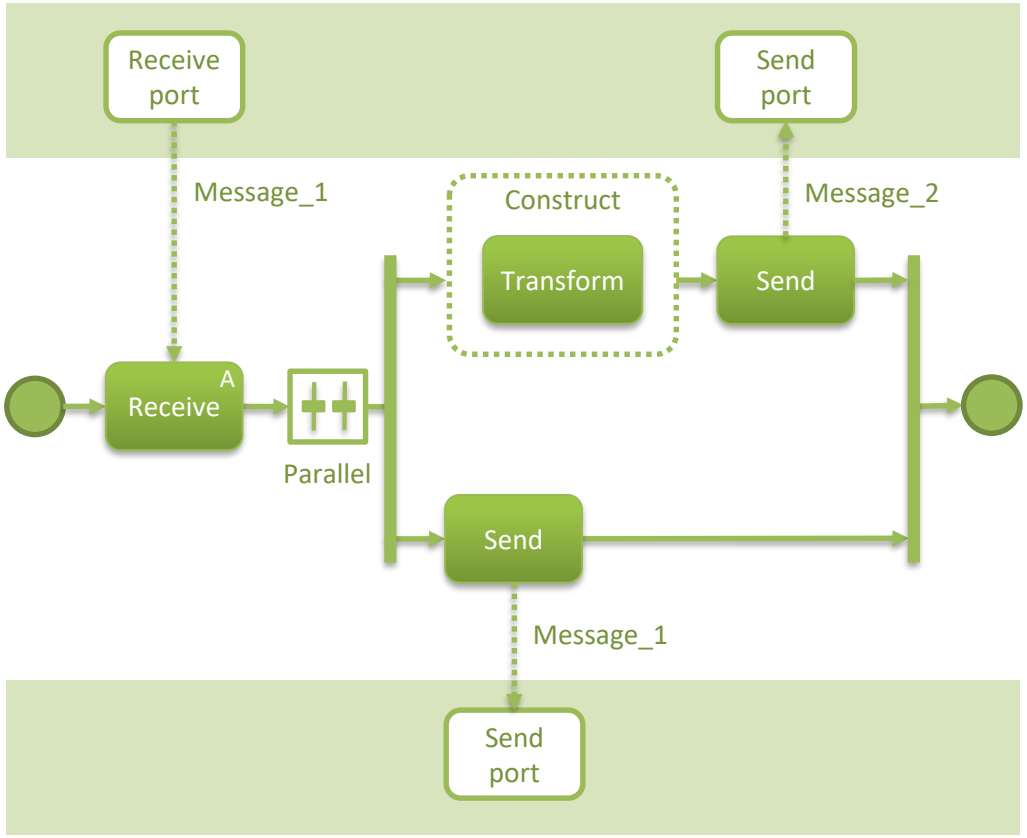
Decisions

- What if there are multiple conditions?



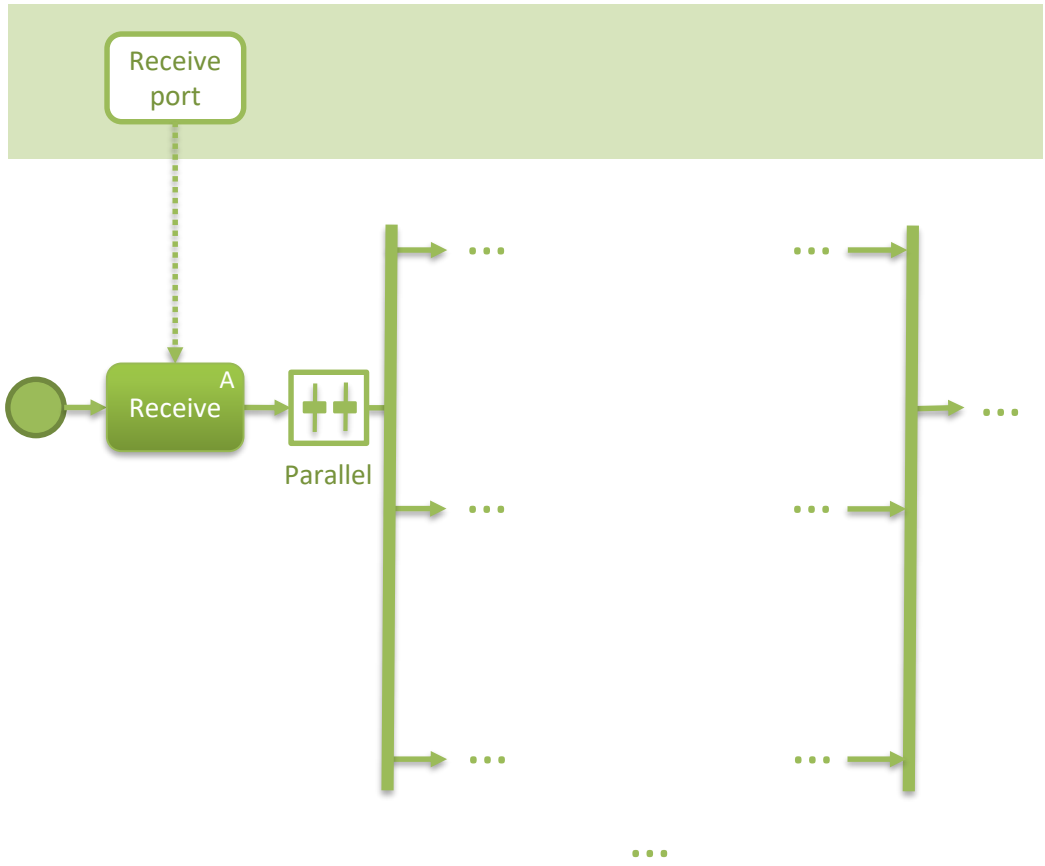
Parallelism

- Multiple paths running concurrently



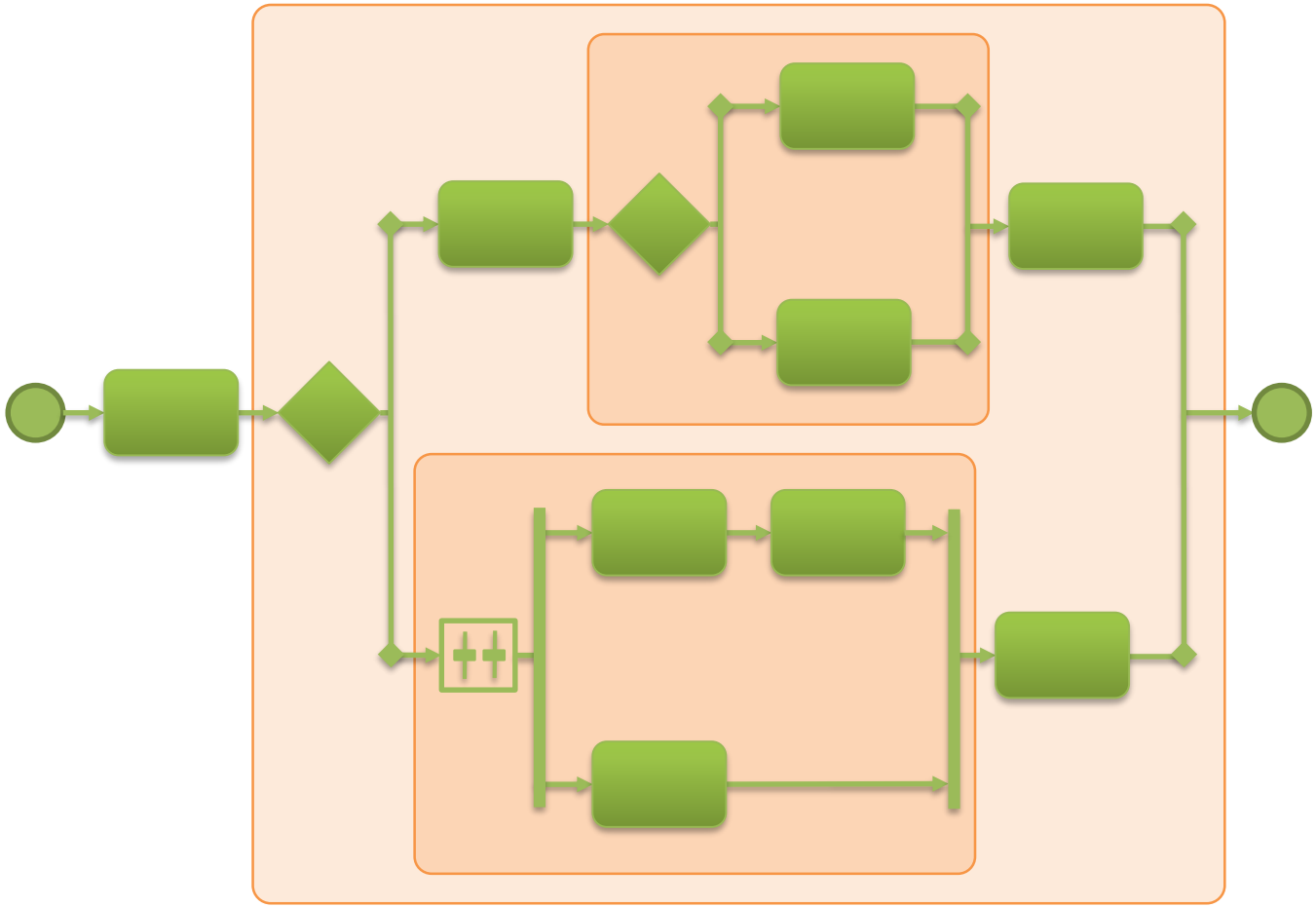
Parallelism

- Multiple paths running concurrently



Block structure

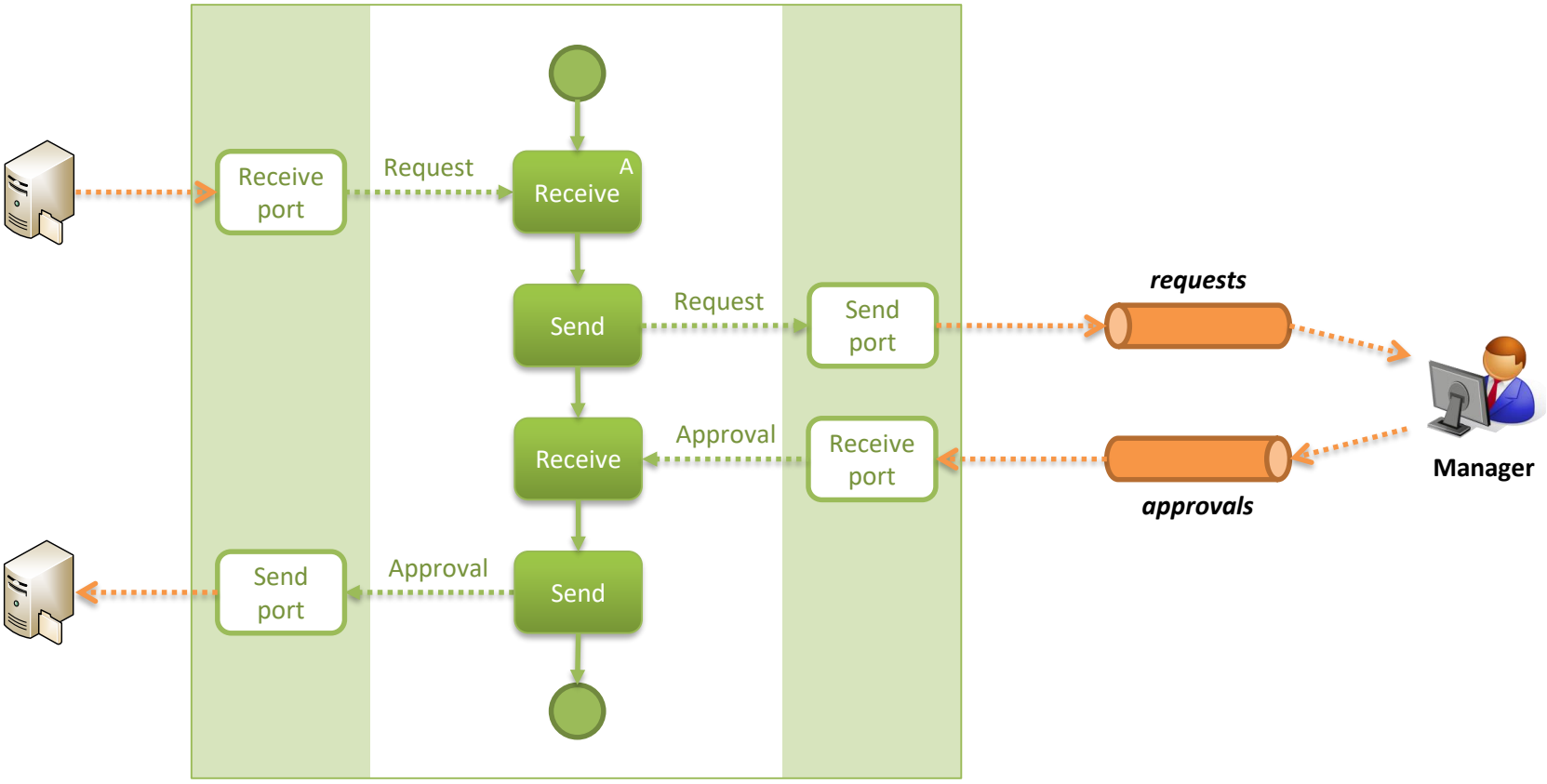
- Orchestrations have a nested block structure



Correlations

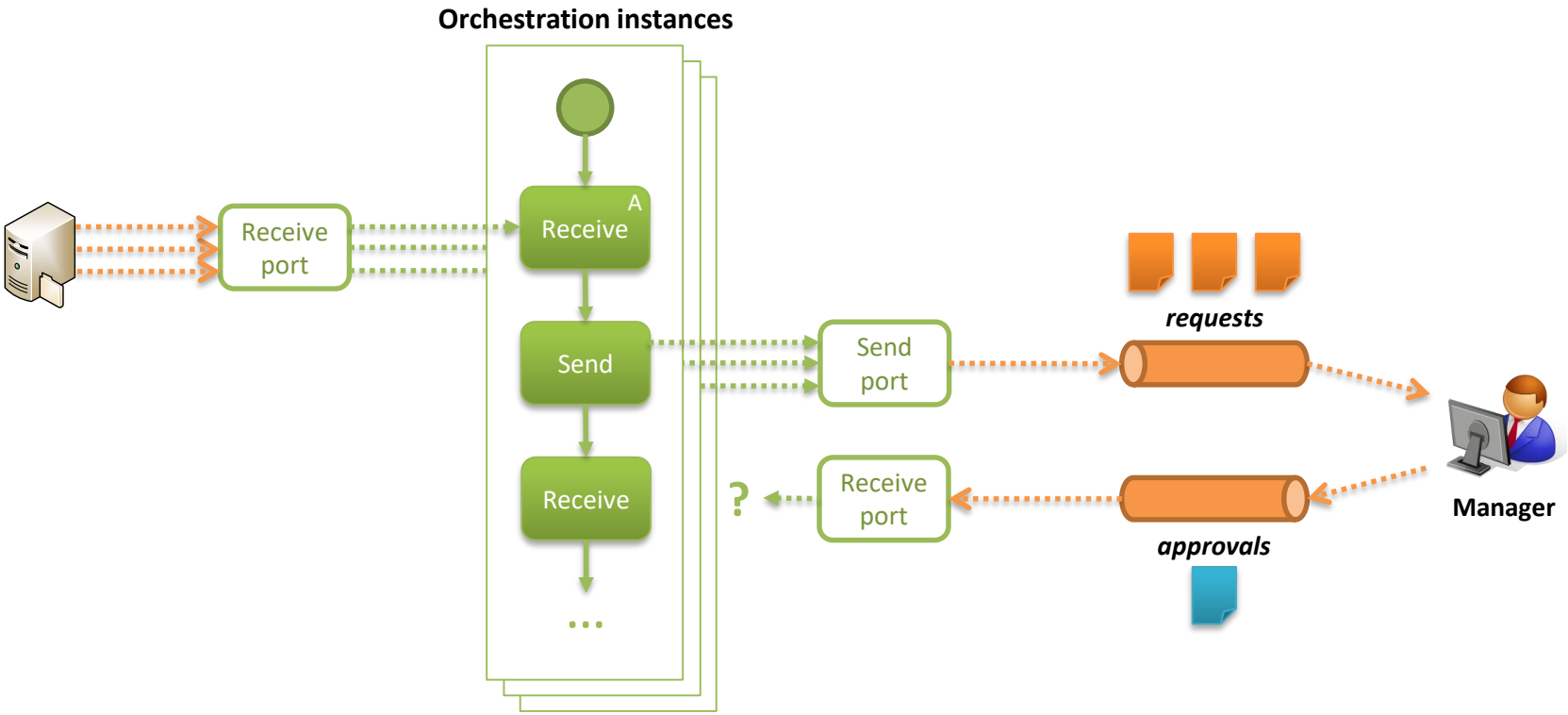
The problem

- An approval process



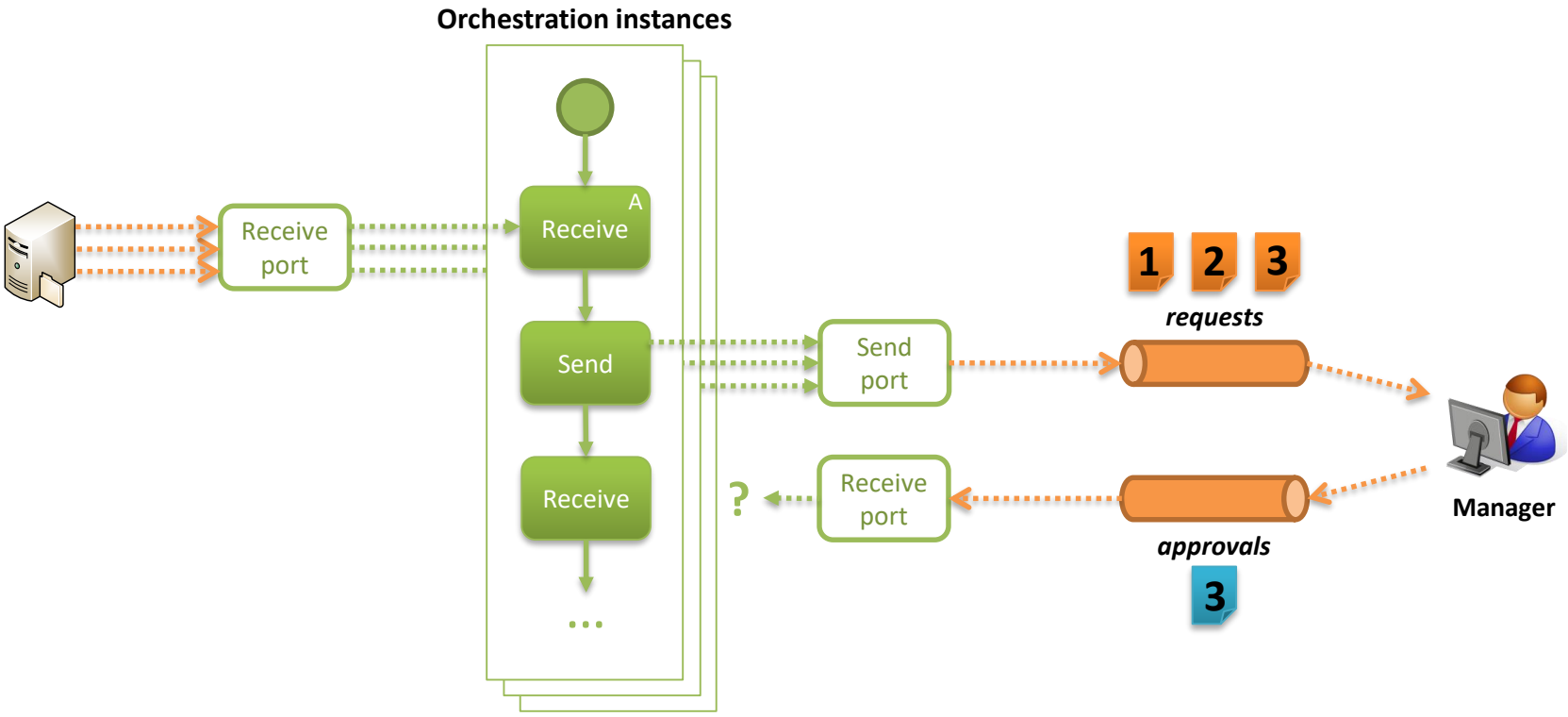
The problem

- The process is instantiated multiple times



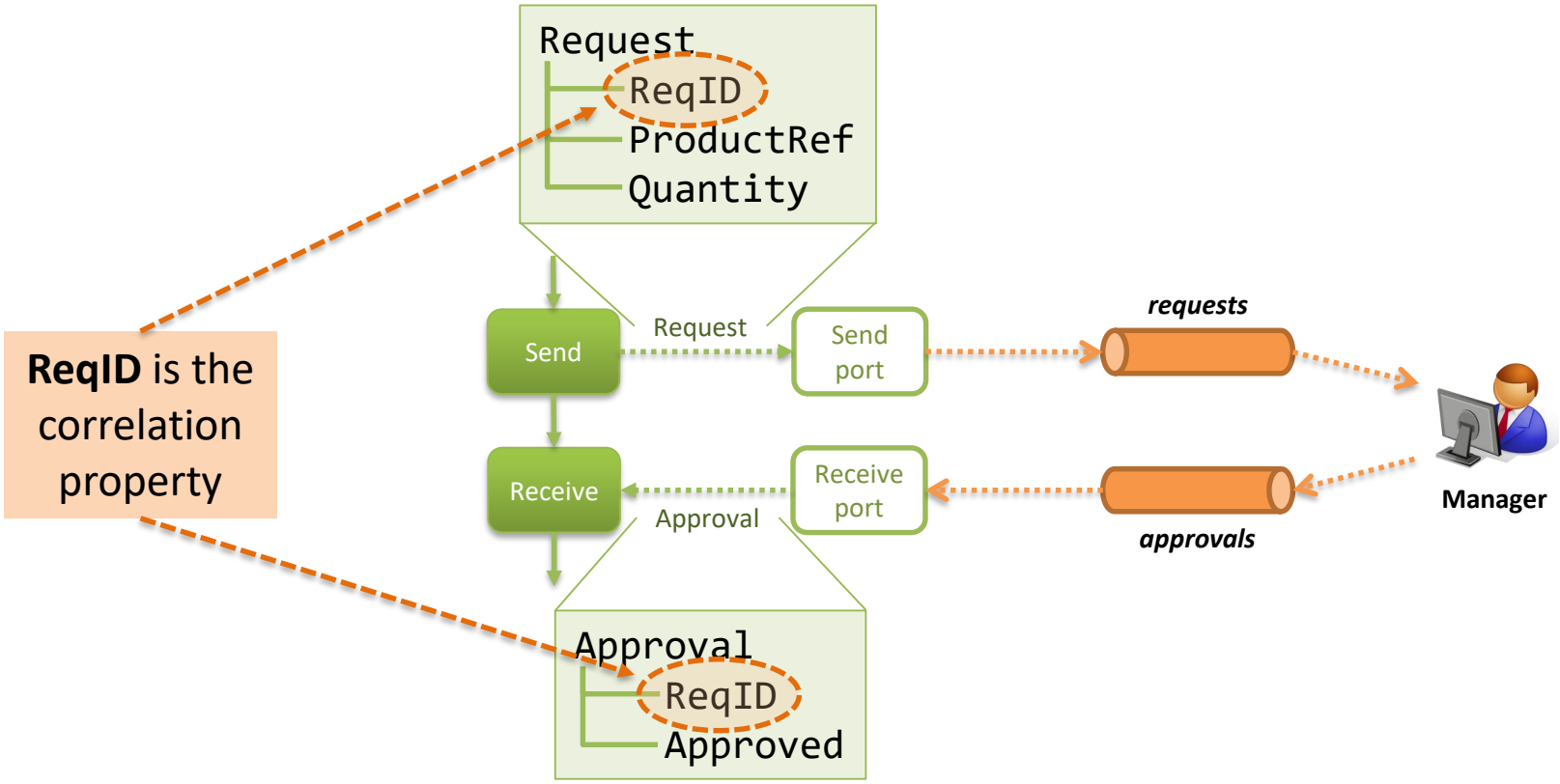
The solution

- The solution is to have a ***correlation id***
  - a unique request number in every request



Correlation properties

- The correlation is based on a common message property

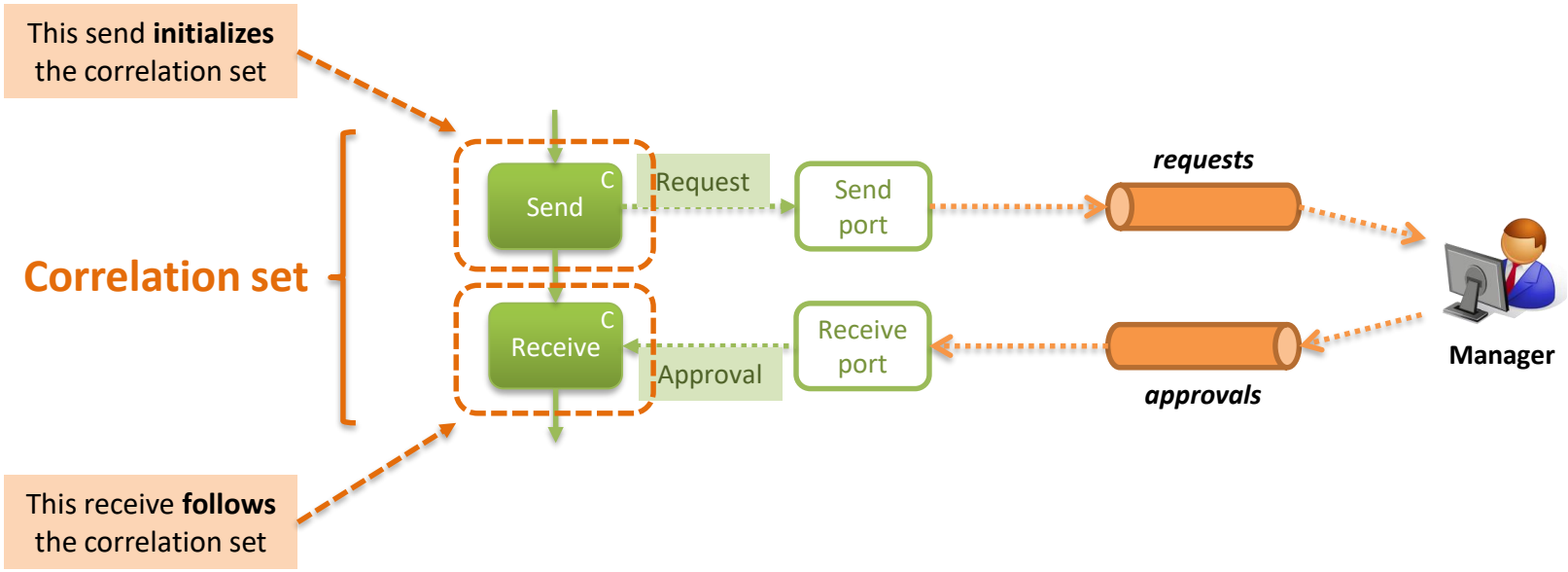


Correlation type vs. correlation set

- Some definitions
  - **Correlation type** is the set of message properties (one or more) that are used as correlation id
  - **Correlation set** is the set of message exchanges (send or receive) included in the same correlation

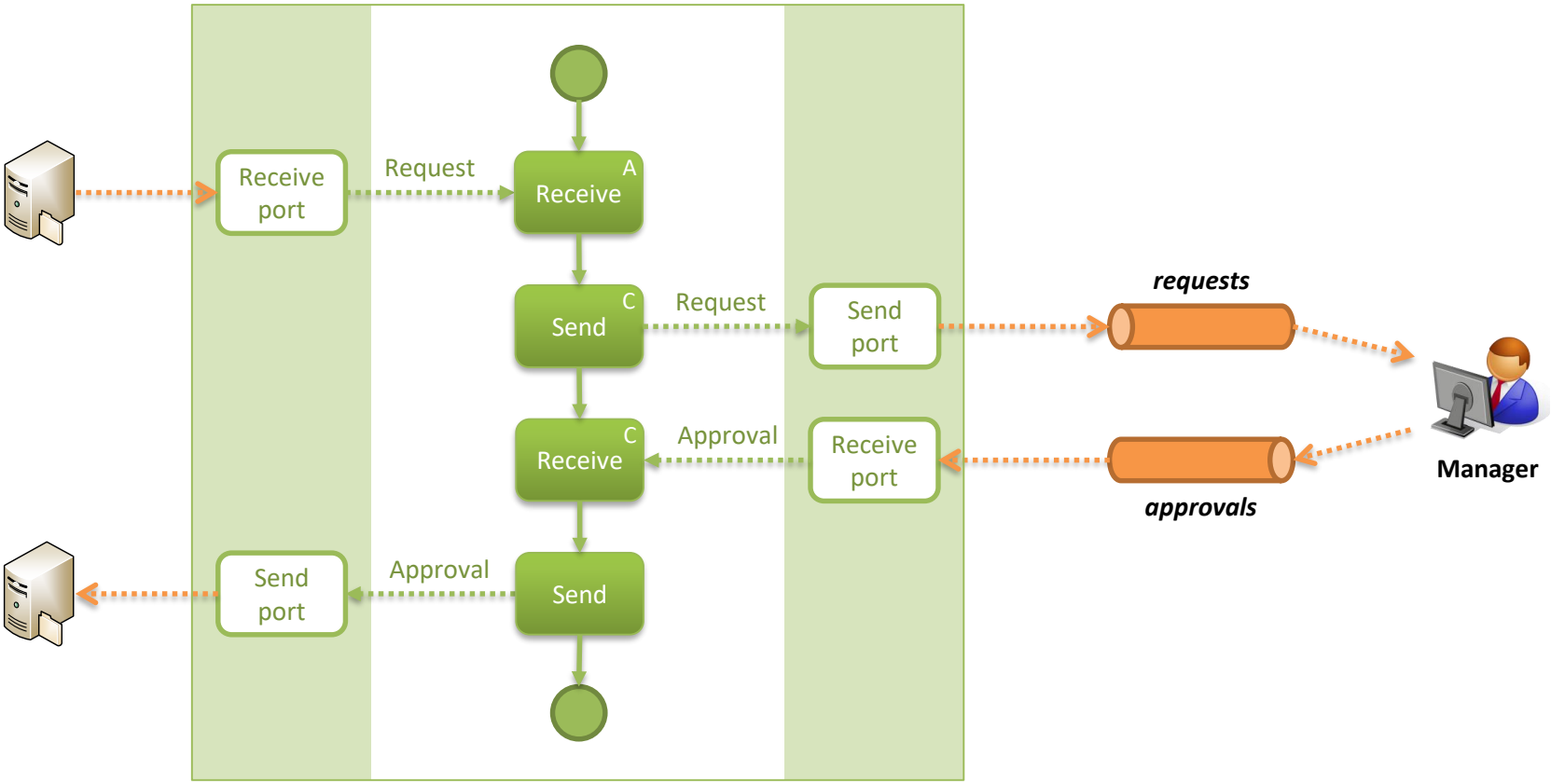
Correlation set

- The correlation set
  - is **initialized** in one exchange
  - is **followed** by one or more exchanges



Correlation

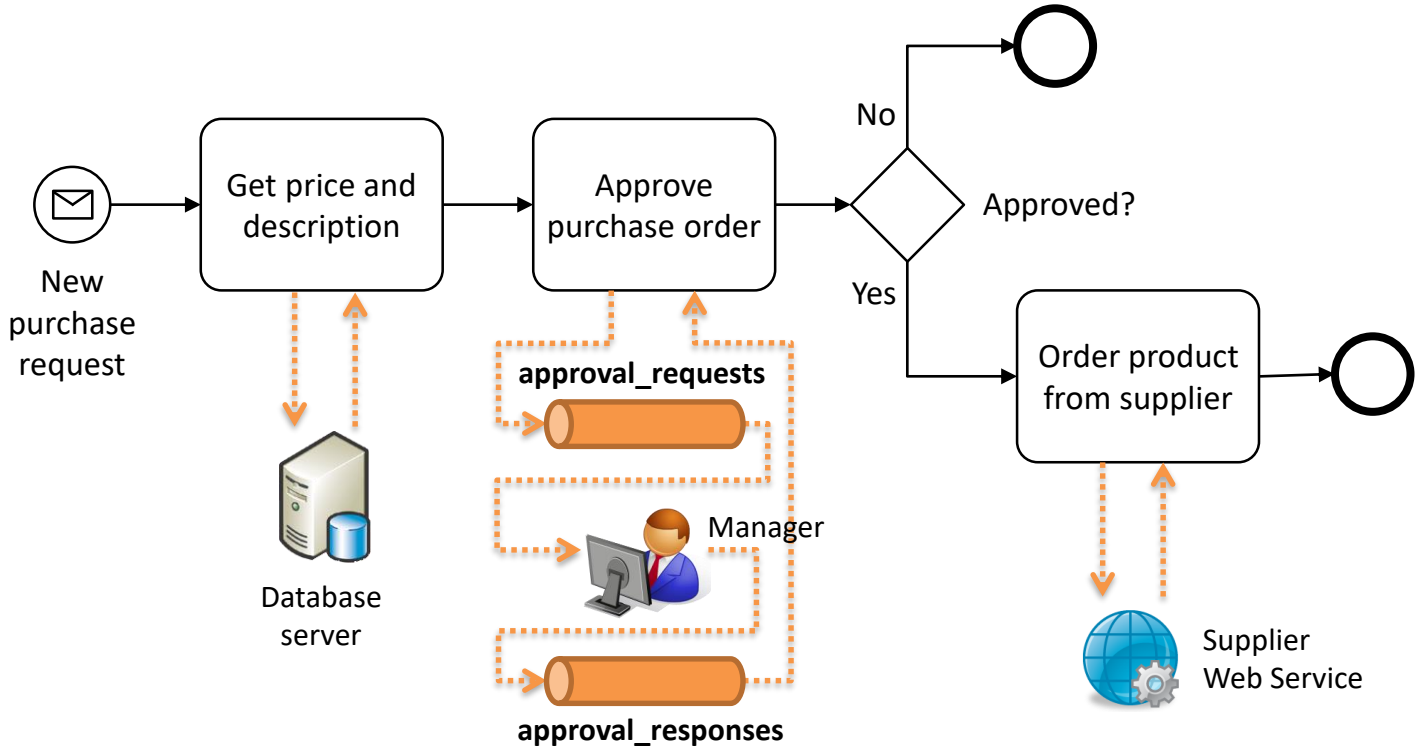
- Adding the correlation to the orchestration



A simple business process

A simple business process

- A purchase process for office supplies

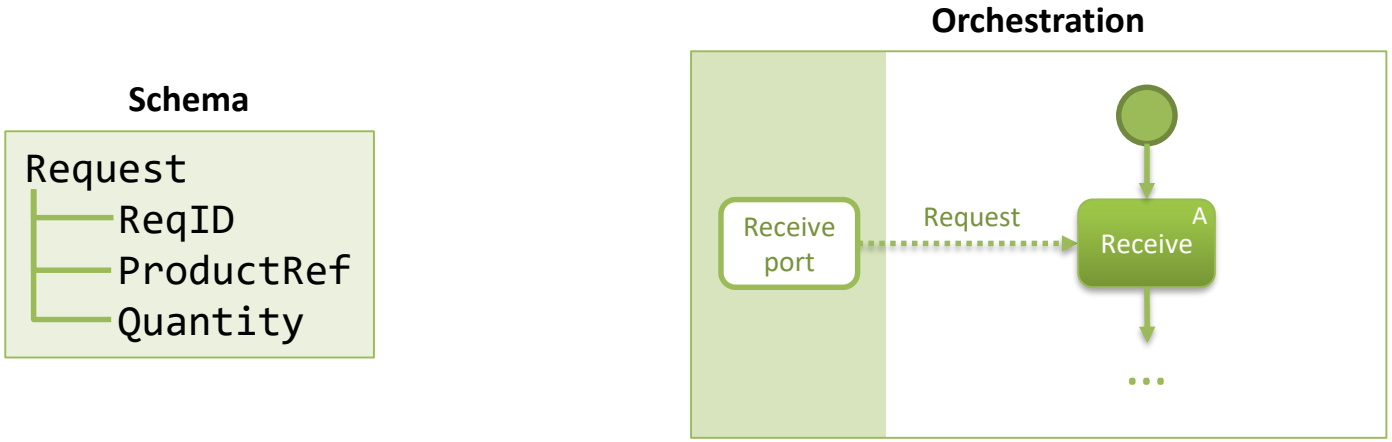


A simple business process

- What our orchestration must do
  - receive a request
  - query a database
  - interact with message queues
  - invoke a Web service

Receiving the request

- The orchestration is instantiated every time a new request is received
  - define the request schema
  - use an activating receive



Querying the database

- A database for office supplies
  - to make things simpler, we will use a single table

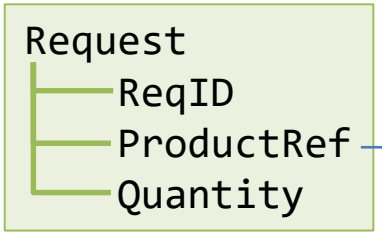
Products

ProductRef	ProductDesc	UnitPrice
537	Printer cartridge	24.99
538	Paper A4 80g	3.99
539	Binder A4 25mm	5.49
540	Office Chair	149.99

Querying the database

- Our request contains a ProductRef

Message schema



Products

ProductRef	ProductDesc	UnitPrice
537	Printer cartridge	24.99
538	Paper A4 80g	3.99
539	Binder A4 25mm	5.49
540	Office Chair	149.99

Message instance (example)

```
<ns0:Request xmlns:ns0="http://OfficeSupplies.Request">
  <ReqID>R1</ReqID>
  <ProductRef>537</ProductRef>
  <Quantity>2</Quantity>
</ns0:Request>
```

Querying the database

- We have to query the database for the given ProductRef

Products

ProductRef	ProductDesc	UnitPrice
537	Printer cartridge	24.99
538	Paper A4 80g	3.99
539	Binder A4 25mm	5.49
540	Office Chair	149.99

Query

```
SELECT ProductDesc, UnitPrice
FROM Products
WHERE ProductRef = 537;
```

Result

ProductDesc	UnitPrice
Printer cartridge	24.99

Querying the database

- The query must work for *any* given ProductRef
  - we turn it into a ***stored procedure***

Stored procedure

```
CREATE PROCEDURE GetProductInfo(@ProductRef INT) AS
SELECT ProductDesc, UnitPrice
FROM Products
WHERE ProductRef = @ProductRef;
```

Sample run

```
EXEC GetProductInfo 537;
```

Result

ProductDesc	UnitPrice
Printer cartridge	24.99

Querying the database

- Since we are working with XML messages, we would like to have the output in XML

Stored procedure

```
CREATE PROCEDURE GetProductInfo(@ProductRef INT) AS
SELECT ProductDesc, UnitPrice
FROM Products
WHERE ProductRef = @ProductRef
FOR XML AUTO;
```

Sample run

```
EXEC GetProductInfo 537;
```

Result

```
<Products ProductDesc="Printer cartridge" UnitPrice="24.99" />
```

Querying the database

- We can even get the schema definition

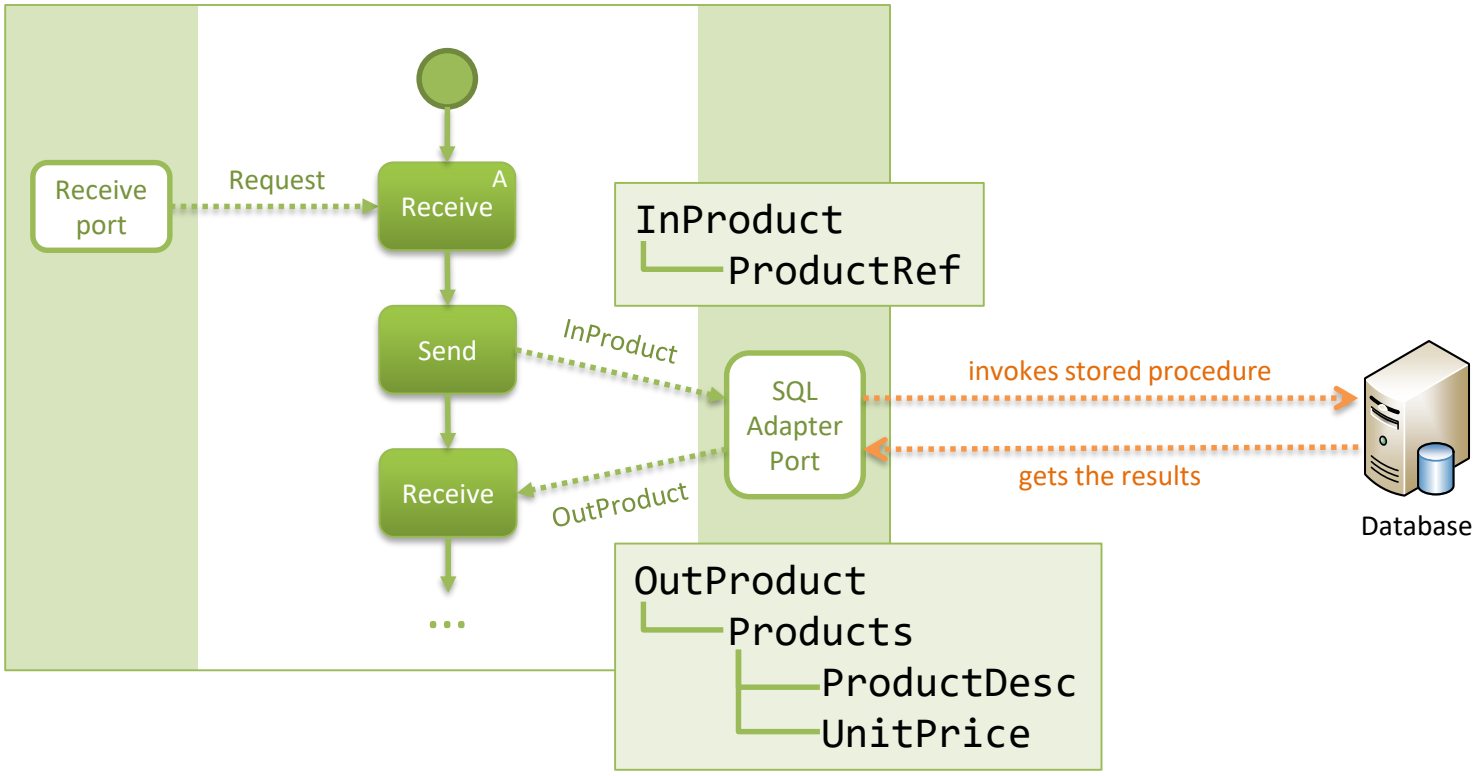
```
CREATE PROCEDURE GetProductInfo(@ProductRef INT) AS
SELECT ProductDesc, UnitPrice
FROM Products
WHERE ProductRef = @ProductRef
FOR XML AUTO, XMLDATA;
```

```
EXEC GetProductInfo 537;
```

```
<Schema name="Schema1" xmlns="urn:schemas-microsoft-com:xml-data"
                       xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="Products" content="empty" model="closed">
    <AttributeType name="ProductDesc" dt:type="string" />
    <AttributeType name="UnitPrice" dt:type="number" />
    <attribute type="ProductDesc" />
    <attribute type="UnitPrice" />
  </ElementType>
</Schema>
<Products xmlns="x-schema:#Schema1" ProductDesc="Printer cartridge" UnitPrice="24.99" />
```

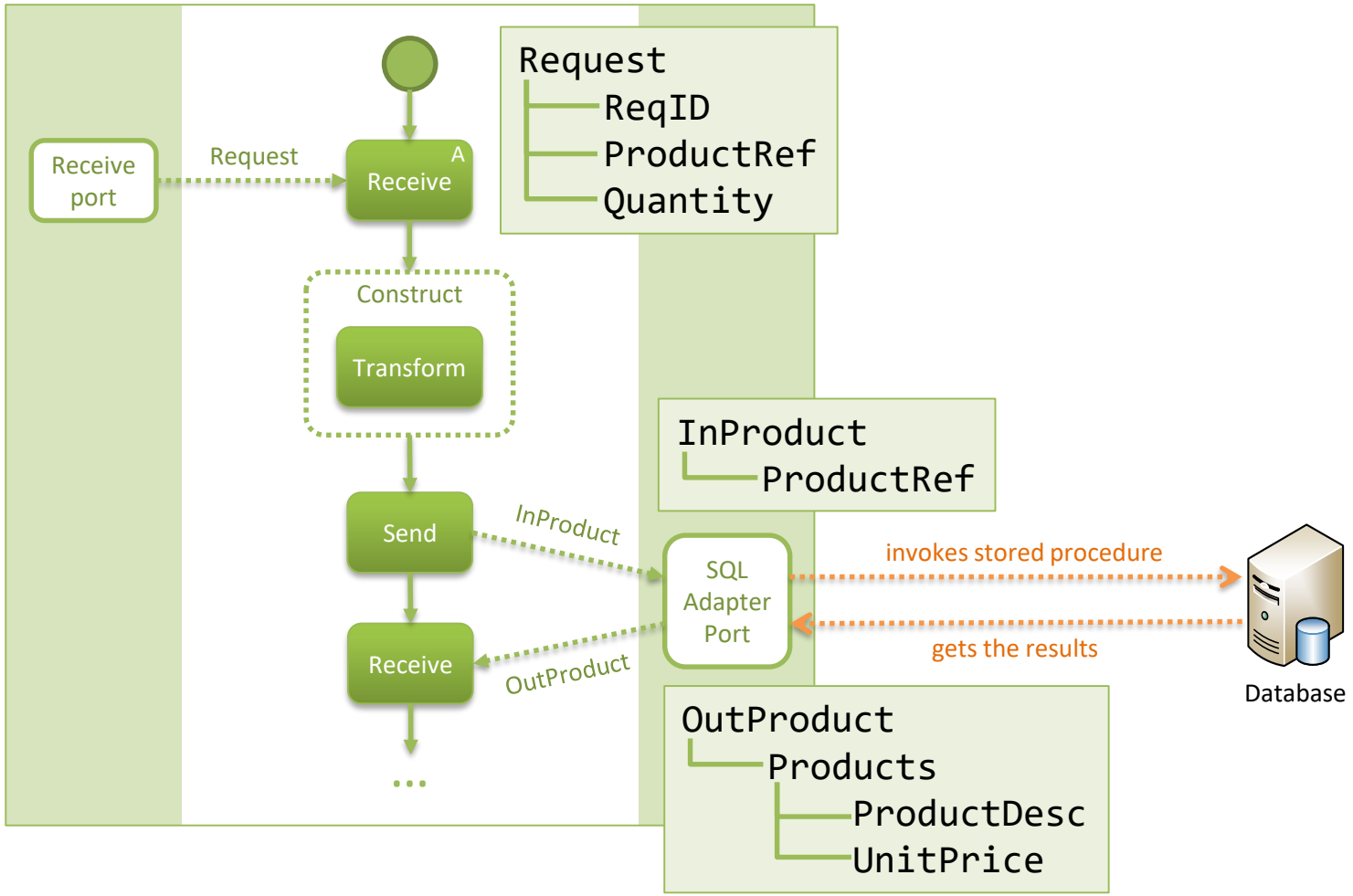
Querying the database

- Orchestration sends input parameters to the stored procedure, and receives the results



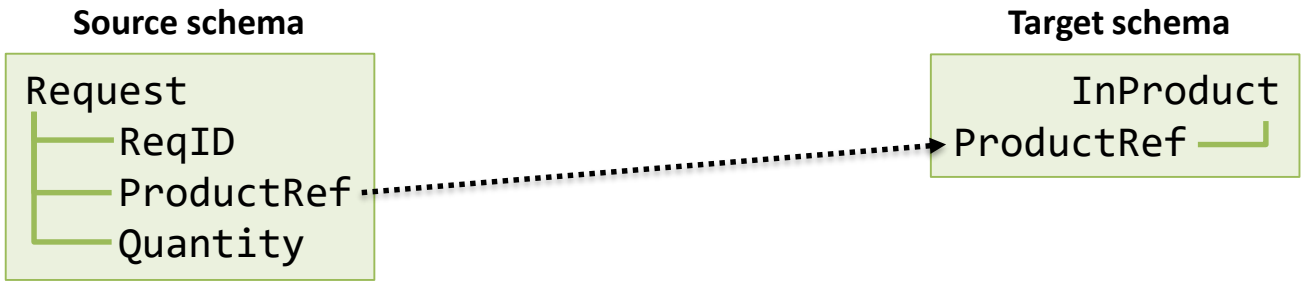
Querying the database

- We need to construct the InProduct message



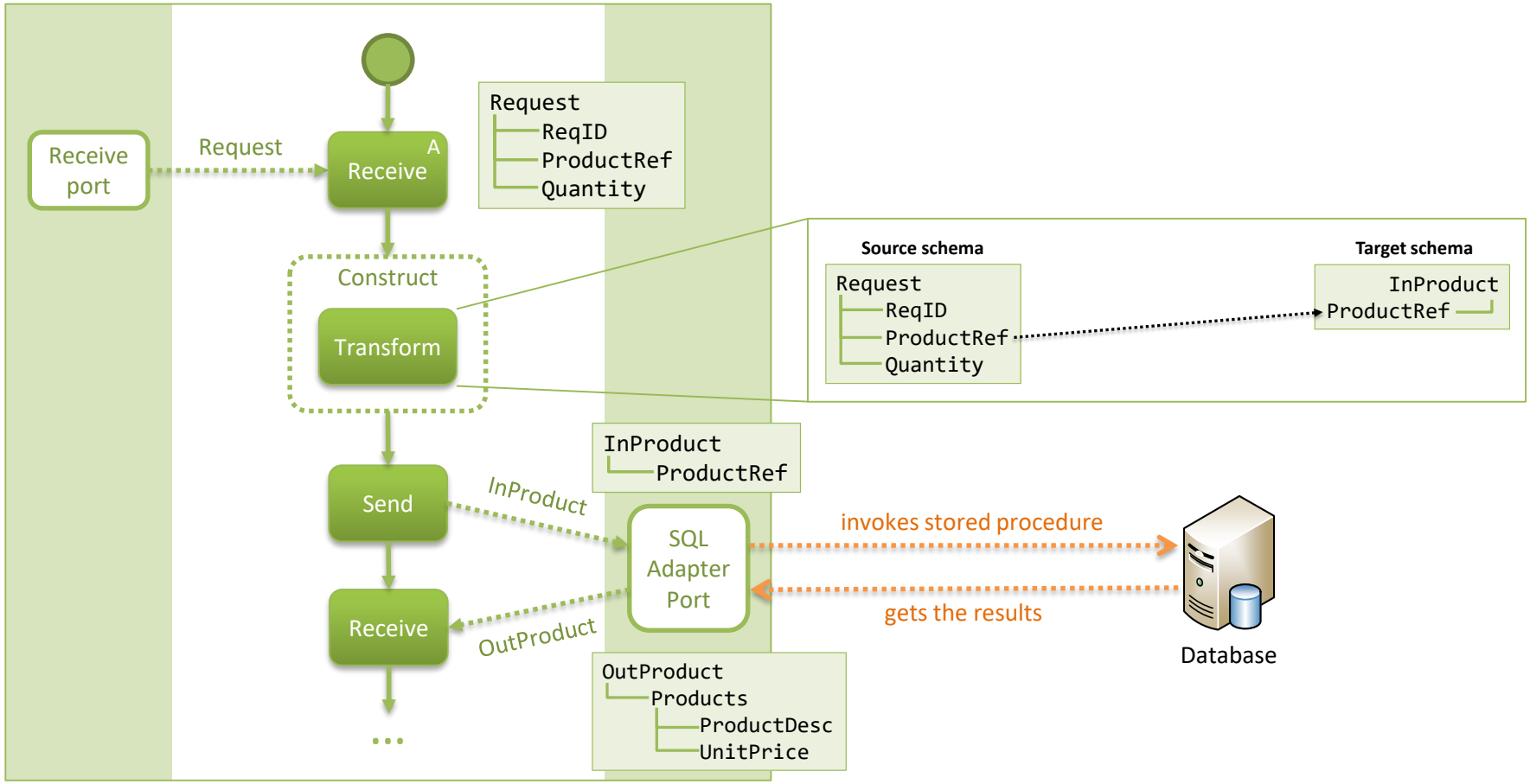
Querying the database

- For this purpose, we use a transformation map



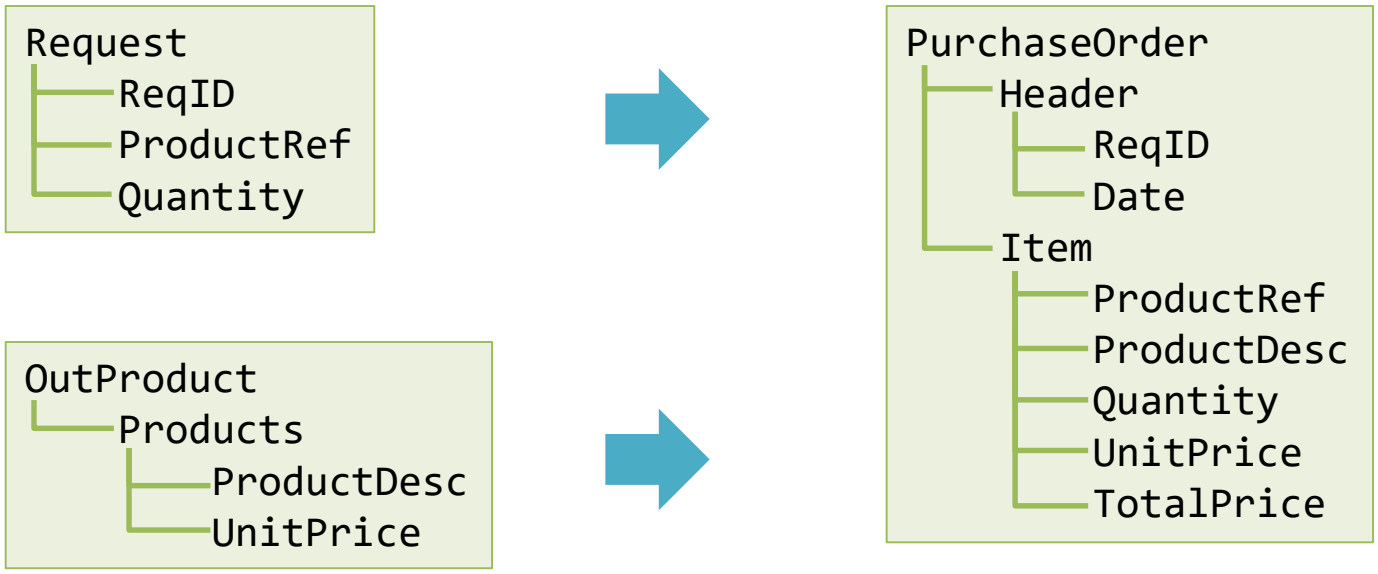
Querying the database

- Our orchestration looks like this



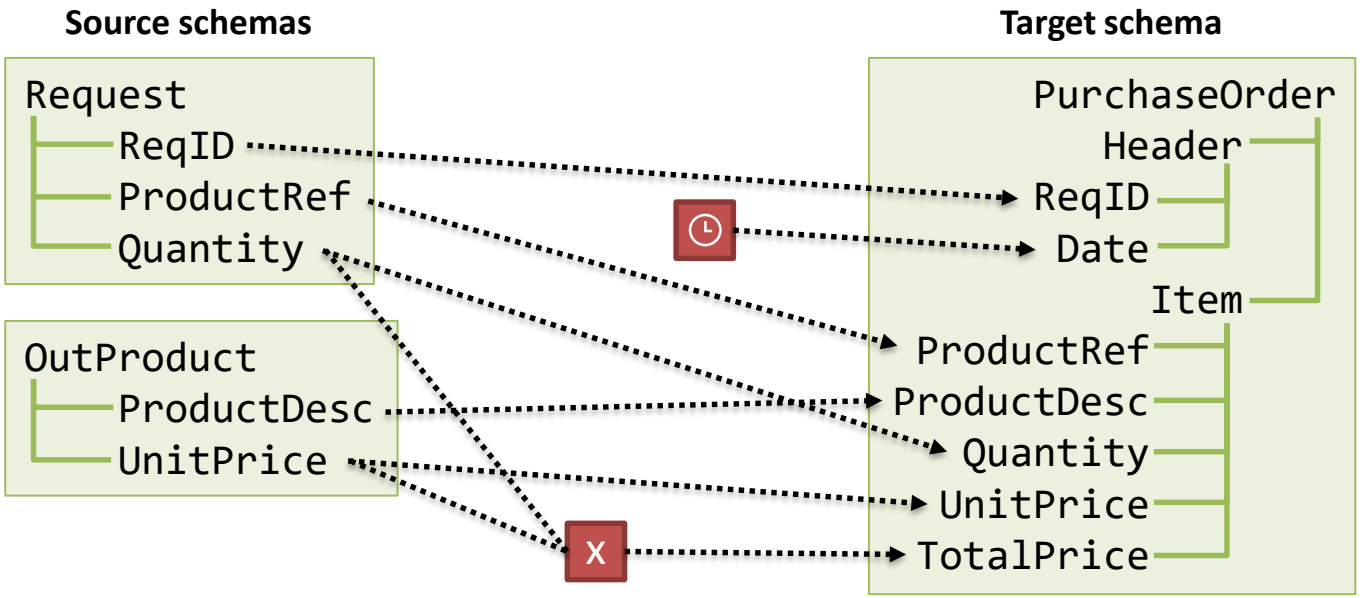
Approving the purchase

- With the original request and the response from the database, we construct a purchase order for approval



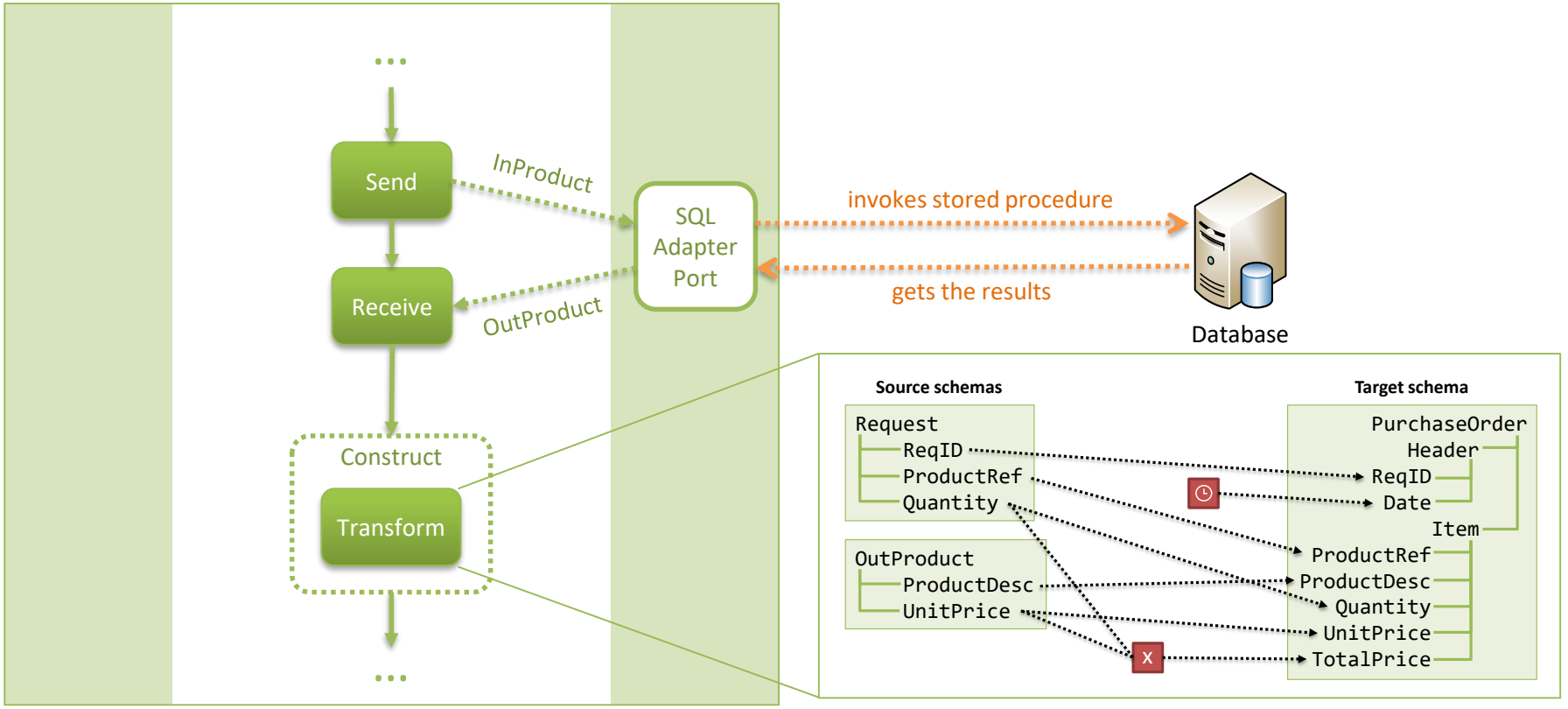
Approving the purchase

- Again, we use a transformation map



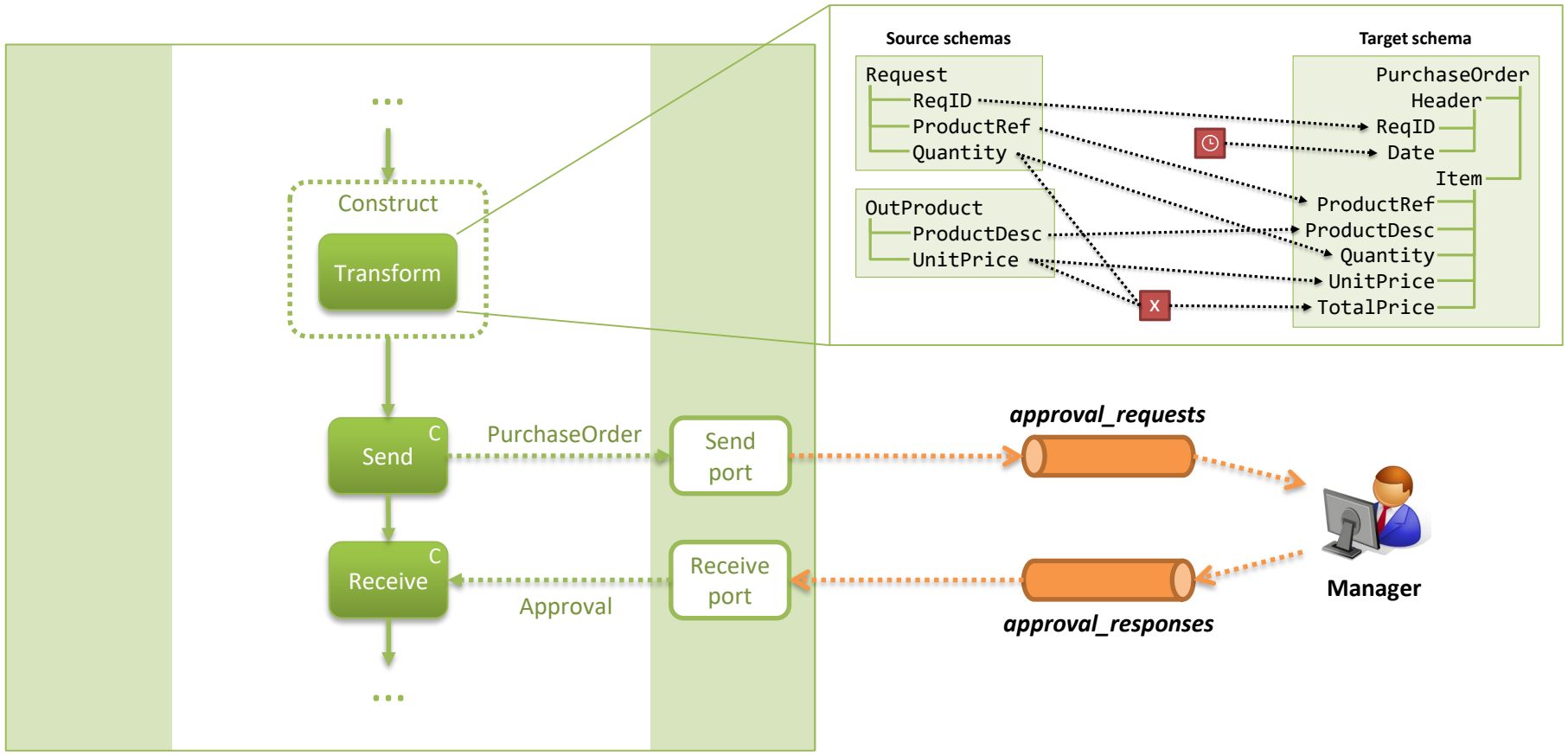
Approving the purchase

- Our orchestration now looks like this



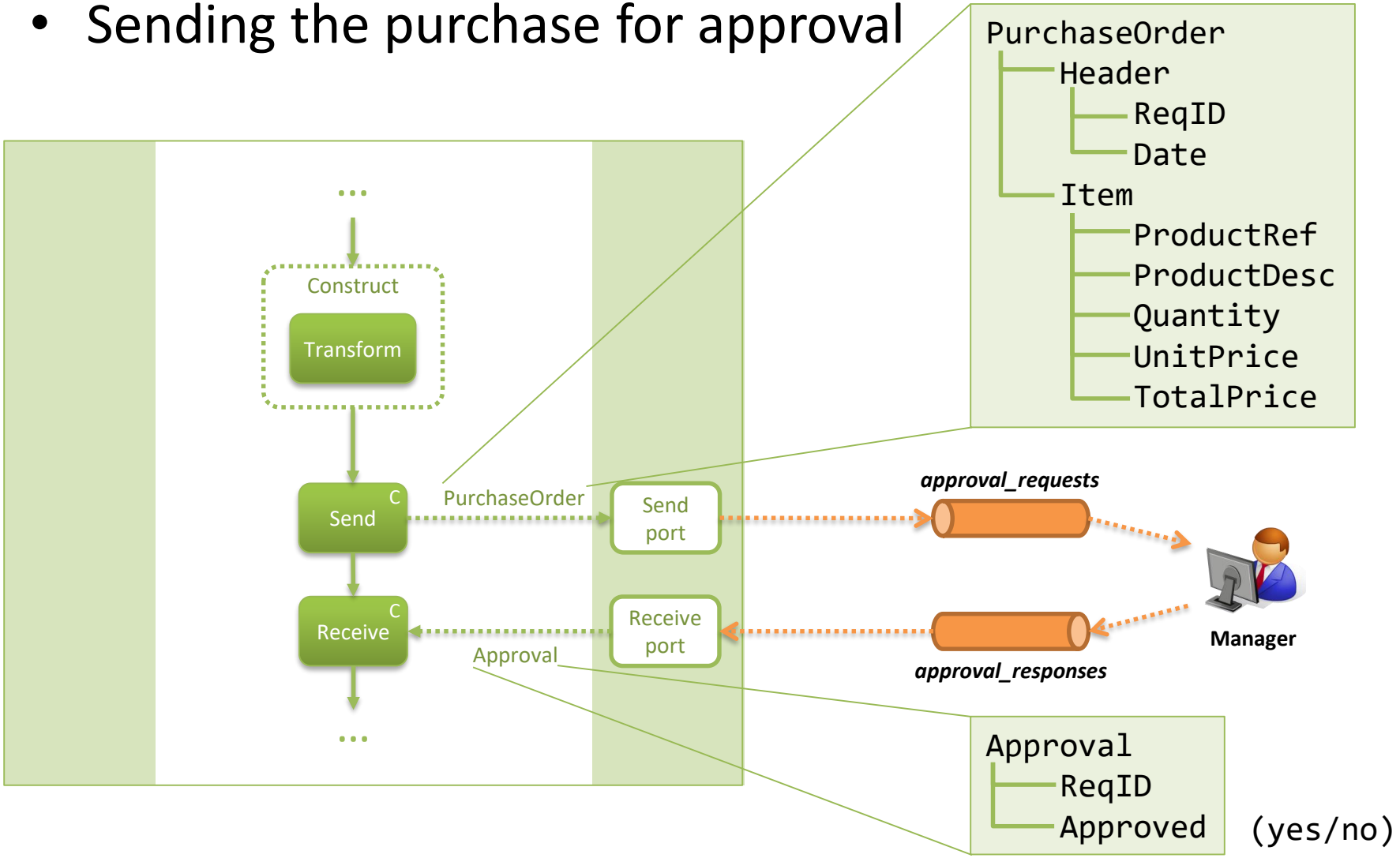
Approving the purchase

- Sending the purchase for approval



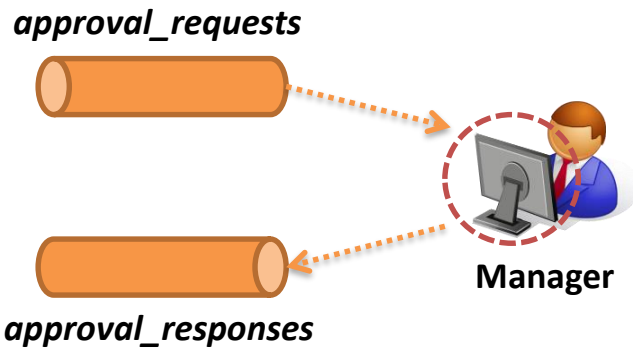
Approving the purchase

- Sending the purchase for approval



Approving the purchase

- The manager has an application to approve the purchase order
  - receives a message (PurchaseOrder) from the ***approval_requests*** queue
  - shows the purchase order and asks whether it should be approved or not
  - sends a message (Approval) to the ***approval_responses*** queue



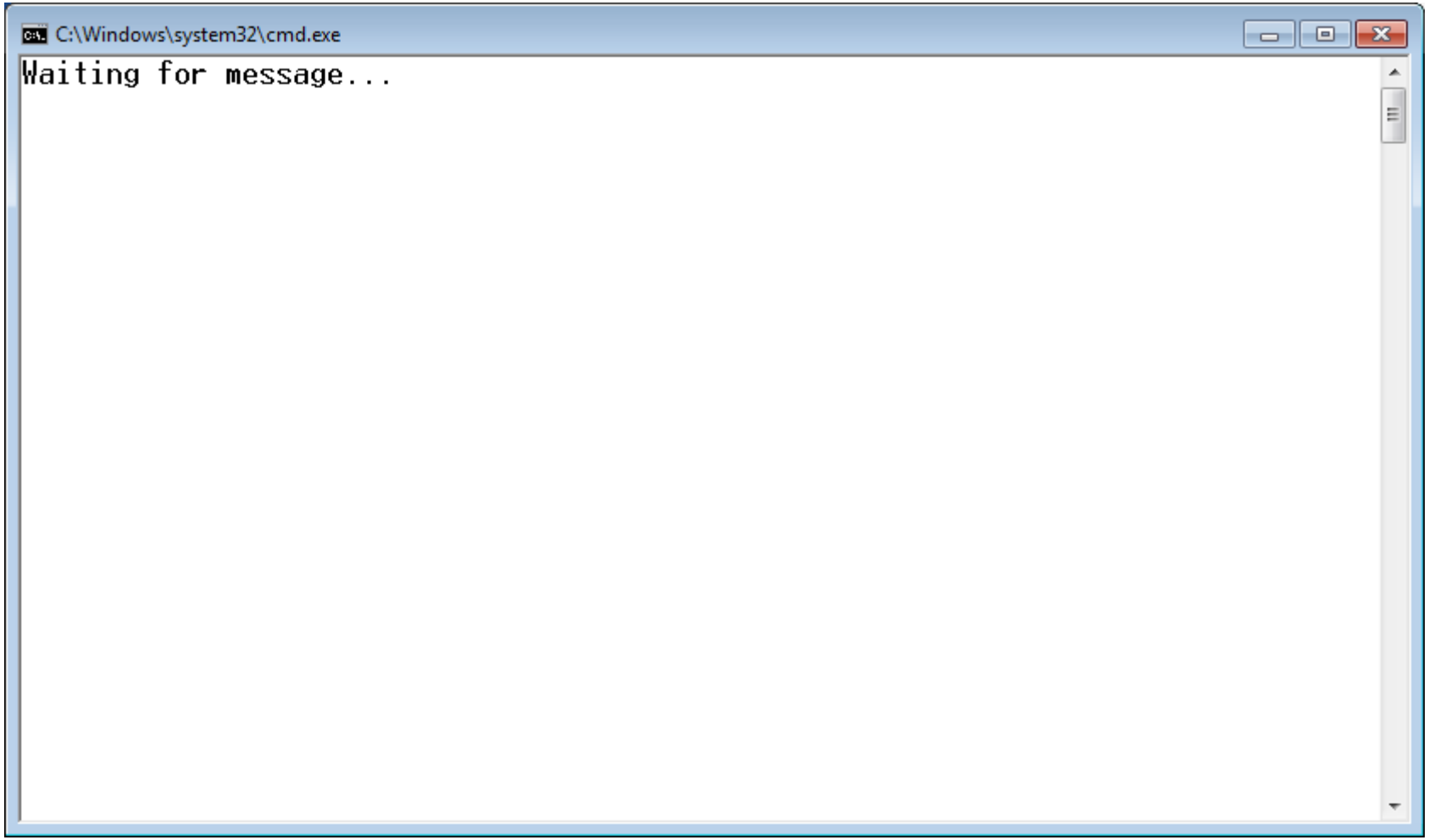
Approving the purchase

- Using C# and MSMQ

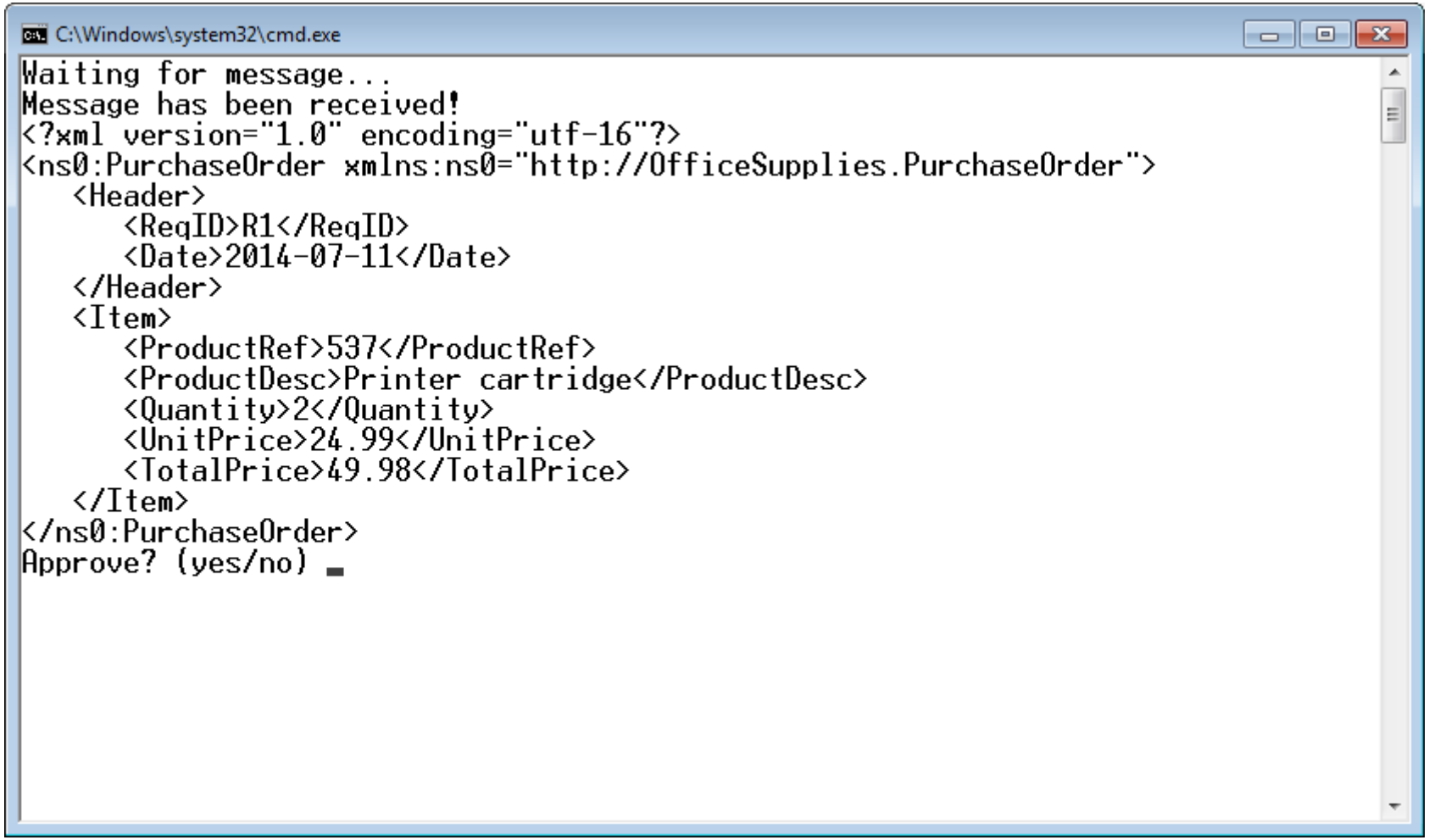
```
using System;
using System.IO;
using System.Xml;
using System.Messaging;

namespace ConsoleApproval
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Waiting for message...");

            string queueName = @".\private$\approval_requests";
            MessageQueue mq = new MessageQueue(queueName);

            Message msg = mq.Receive();
            Console.WriteLine("Message has been received!");

            StreamReader reader = new StreamReader(msg.BodyStream);
            string request = reader.ReadToEnd();
            Console.WriteLine(request);
```

```
            XmlDocument doc = new XmlDocument();
            doc.LoadXml(request);
            string ReqID = doc.GetElementsByTagName("ReqID")[0].InnerText;

            string Approved = "";
            while ((Approved != "yes") && (Approved != "no"))
            {
                Console.Write("Approve? (yes/no) ");
                Approved = Console.ReadLine().ToLower();
            }

            string response = "<ns0:Approval xmlns:ns0=\"http://OfficeSupplies.Approval\">";
            response += "<ReqID>" + ReqID + "</ReqID>";
            response += "<Approved>" + Approved + "</Approved>";
            response += "</ns0:Approval>";
            Console.WriteLine(response);

            queueName = @".\private$\approval_responses";
            mq = new MessageQueue(queueName);

            msg = new Message();

            StreamWriter writer = new StreamWriter(msg.BodyStream);
            writer.Write(response);
            writer.Flush();

            mq.Send(msg);

            Console.WriteLine("Message has been sent!");
        }
    }
}
```

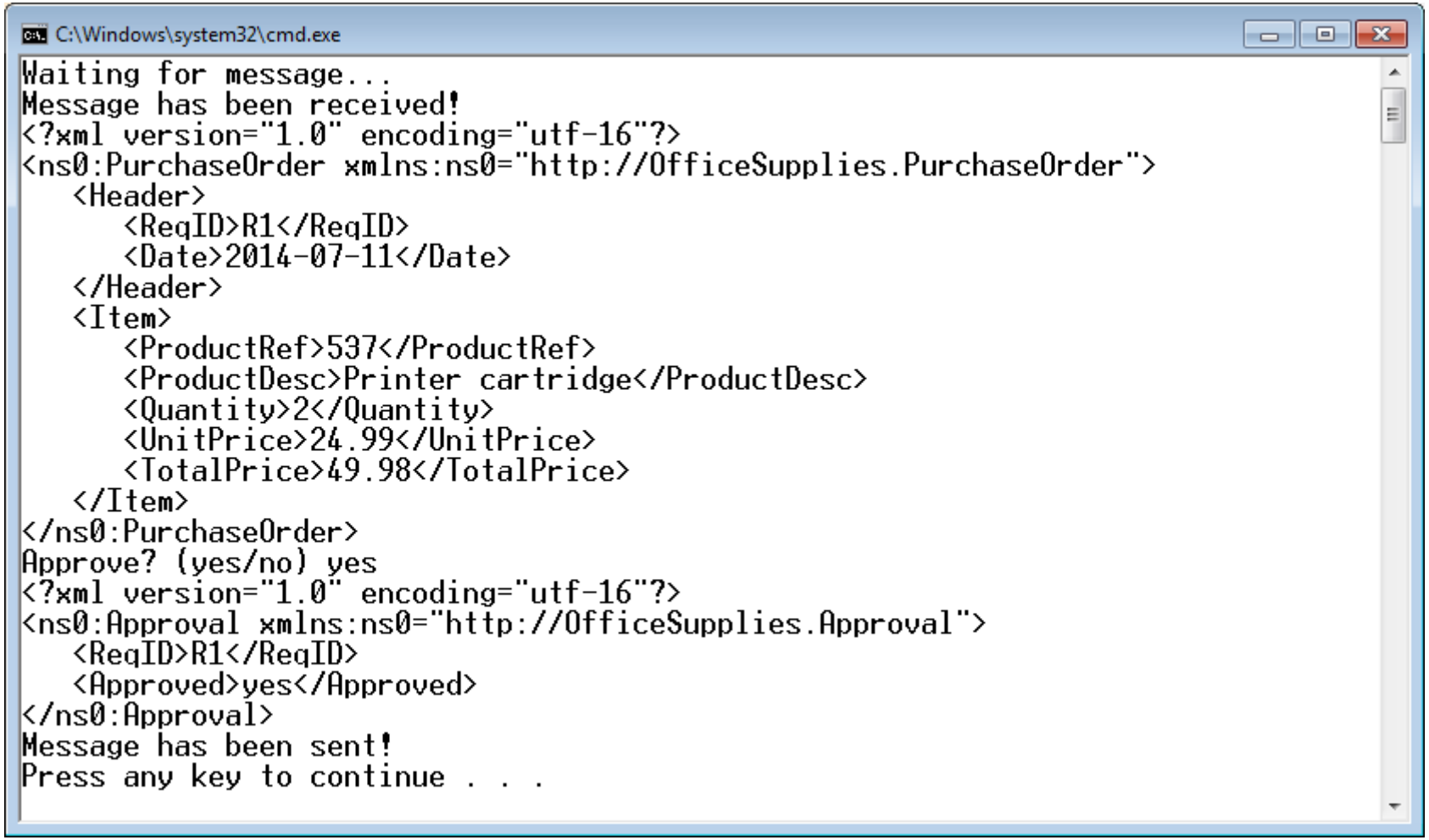
Approving the purchase



Approving the purchase

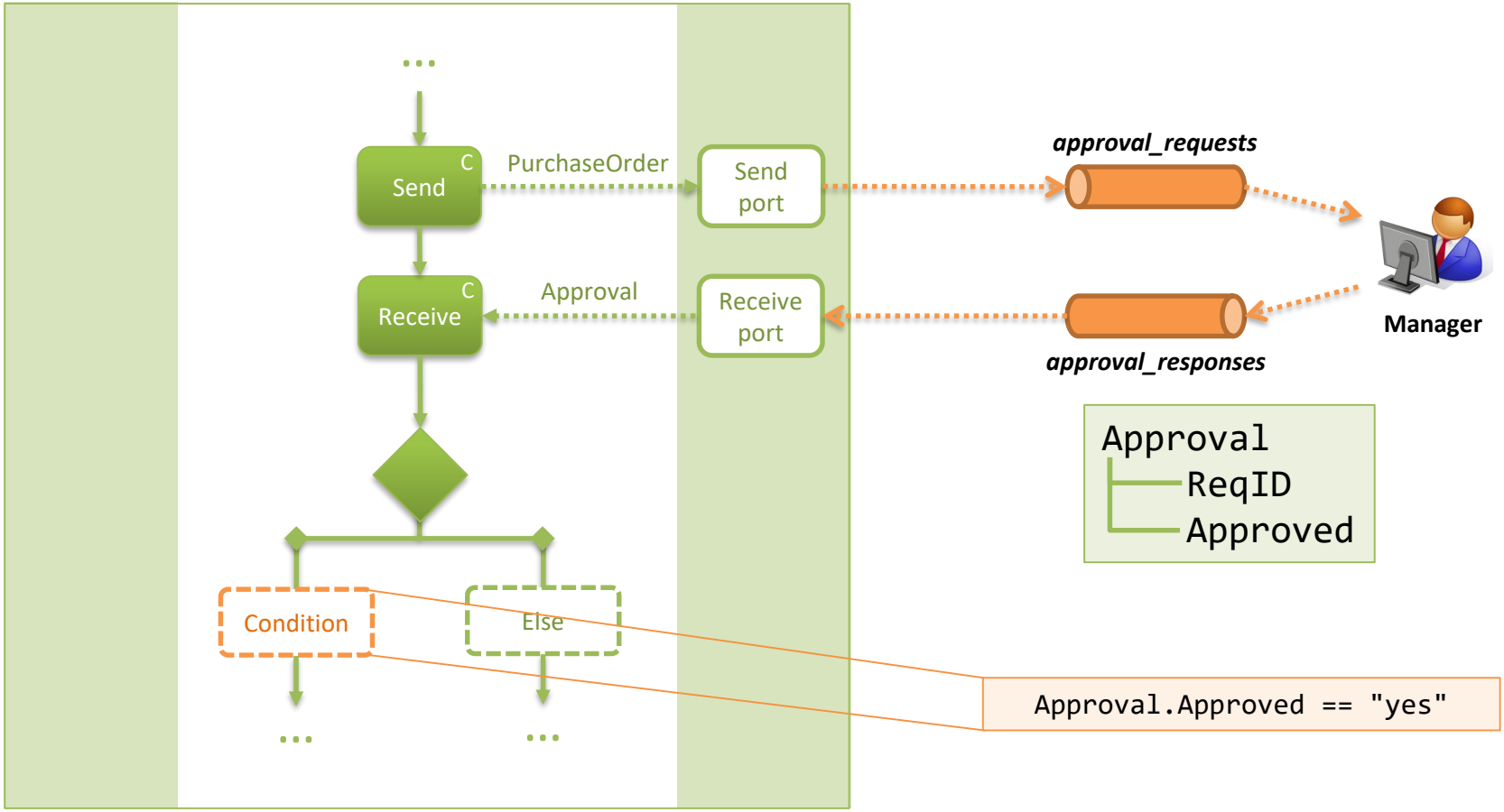


Approving the purchase

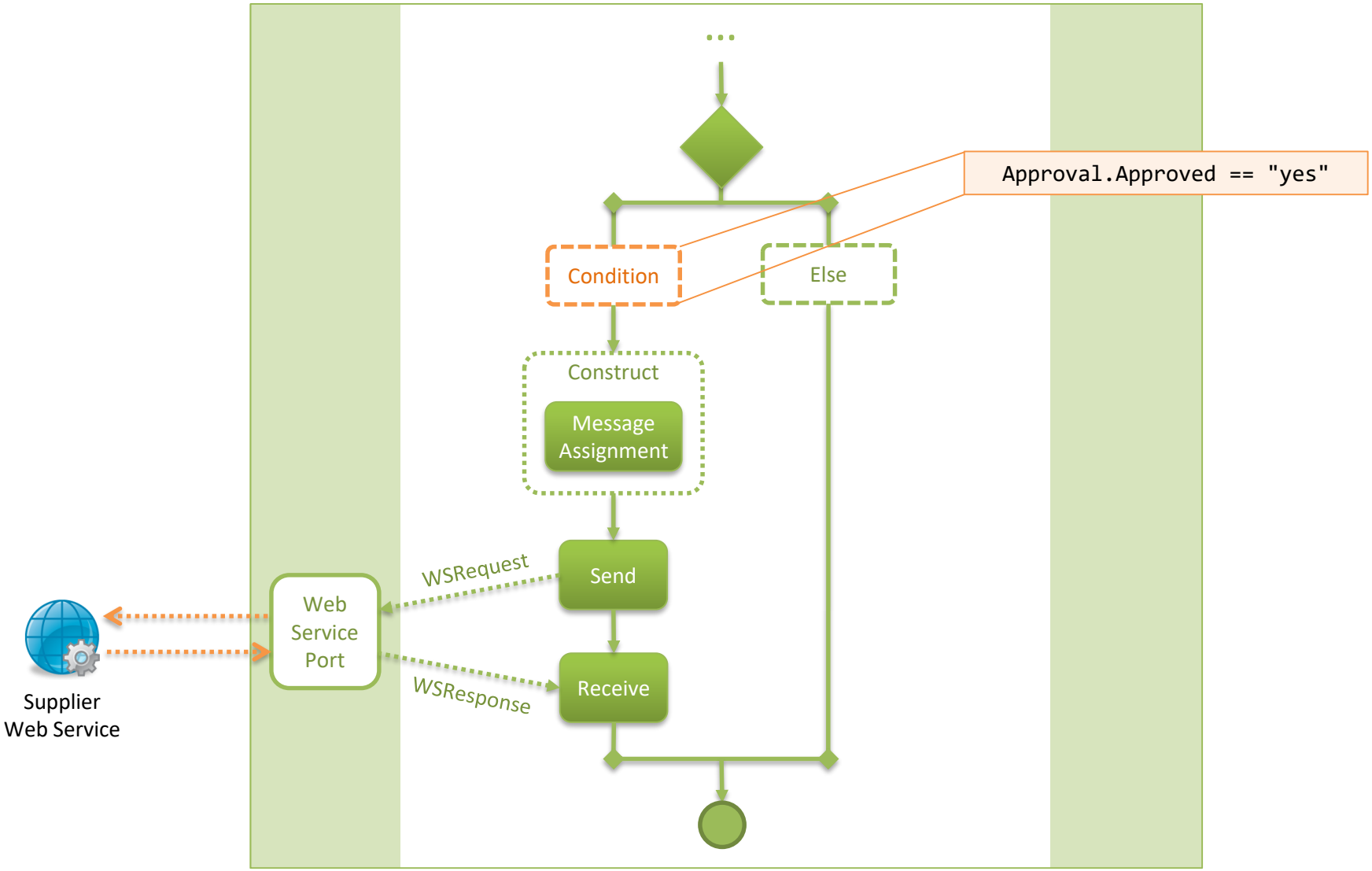


Approving the purchase

- Checking if the purchase is approved

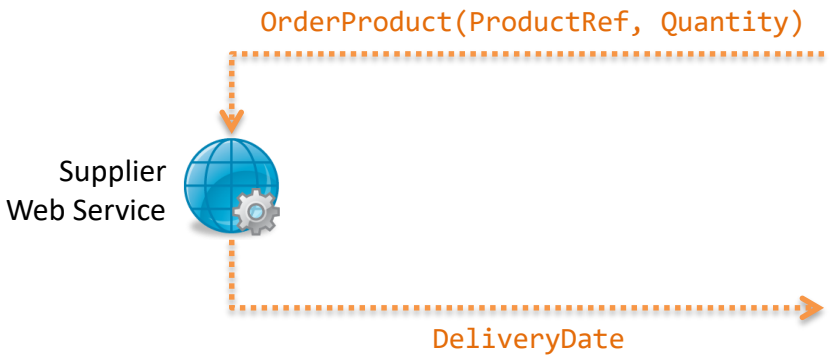


Invoking the Web Service



Invoking the Web Service

- The Supplier Web Service
  - has a single method **`OrderProduct()`**




```
string OrderProduct(int ProductRef, int Quantity)
{
    ...

    return DeliveryDate;
}
```

Invoking the Web Service

- The Supplier Web Service
  - a simple implementation in ASP.NET and C#

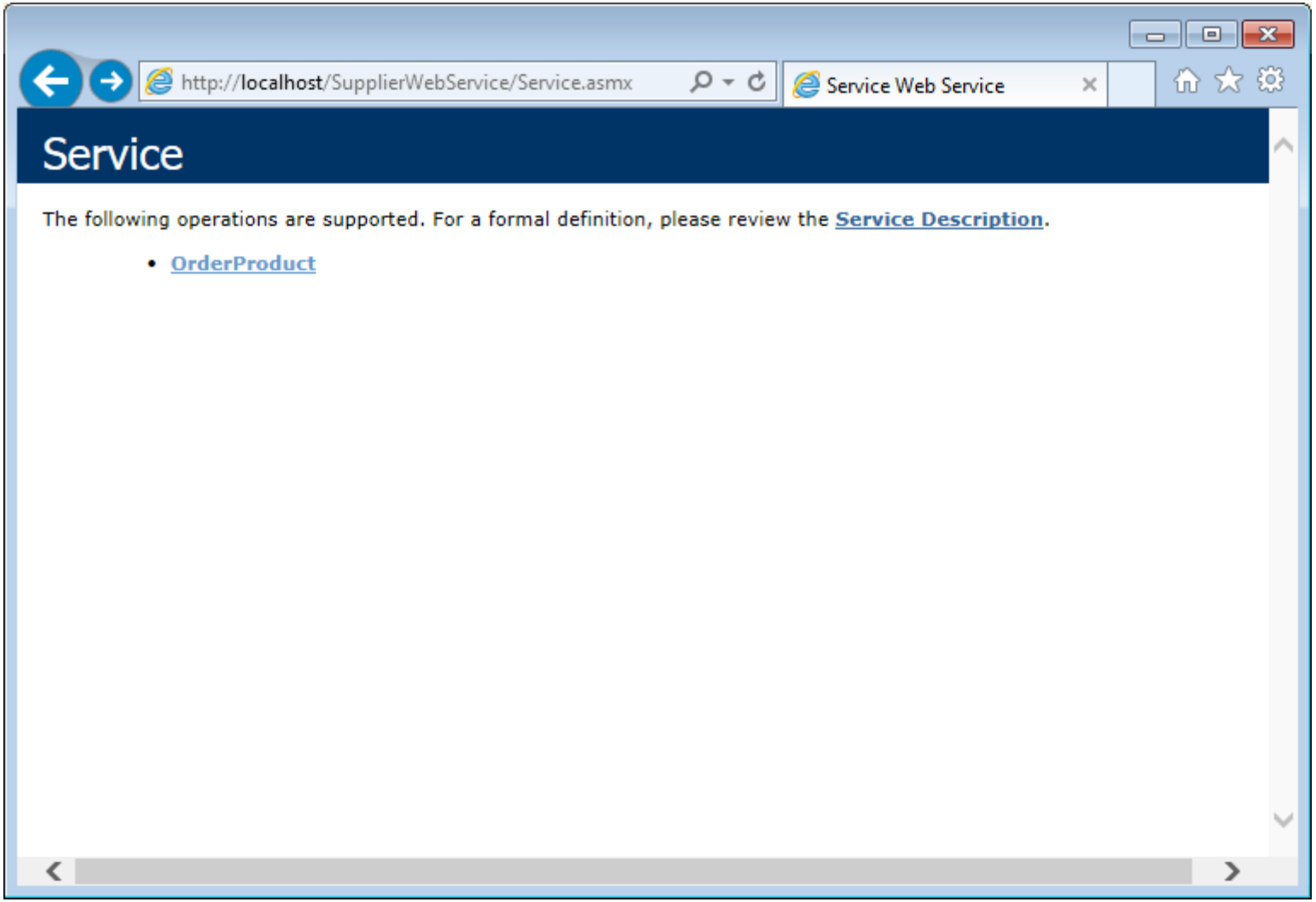
Service.asmx

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/Service.cs" Class="Service" %>
```

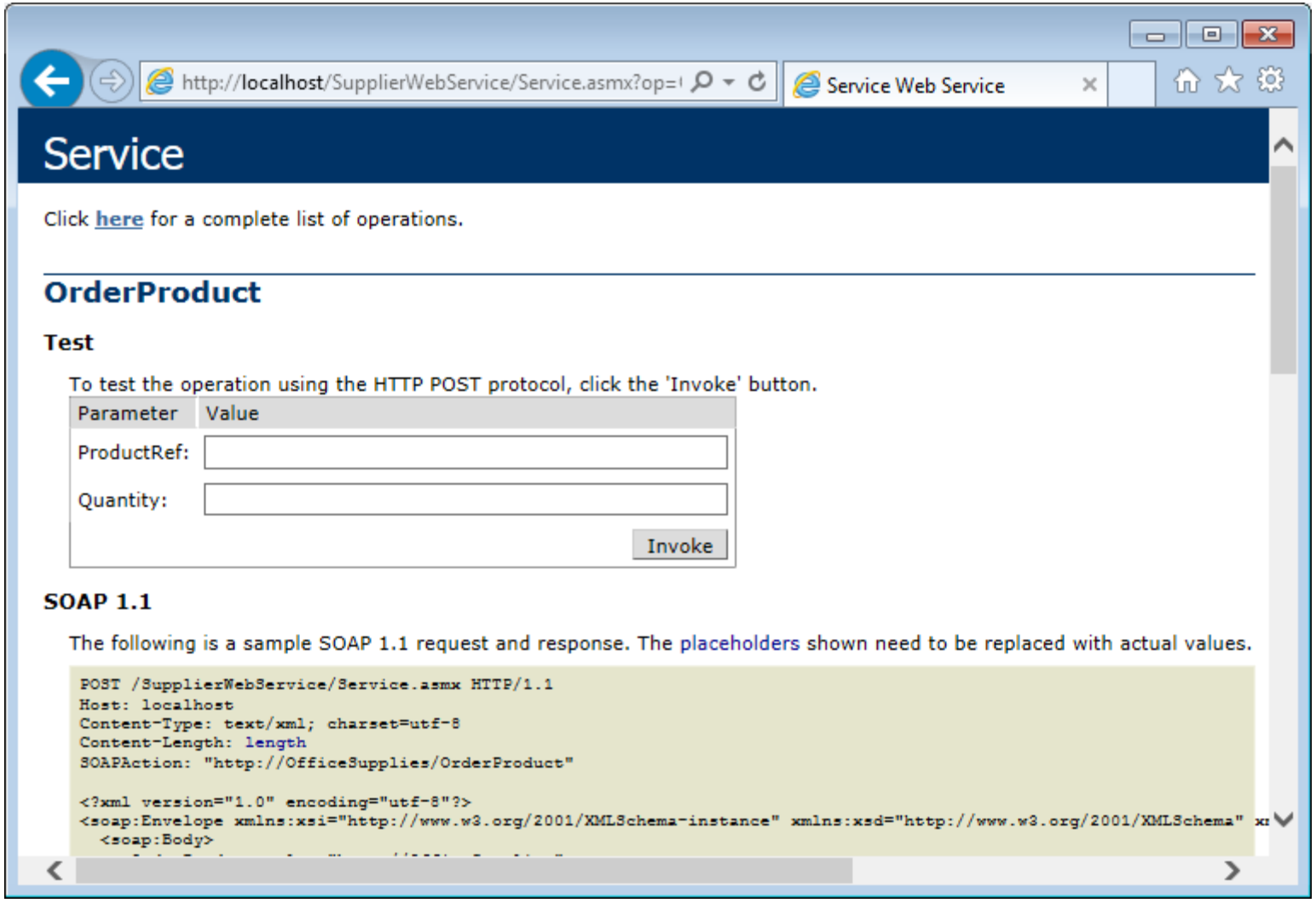
Service.cs

```
using System;
using System.Web.Services;

[WebService(Namespace = "http://OfficeSupplies")]
public class Service : WebService
{
    [WebMethod]
    public string OrderProduct(int ProductRef, int Quantity)
    {
        string DeliveryDate = DateTime.Now.AddDays(2).ToString("yyyy-MM-dd");

        return DeliveryDate;
    }
}
```

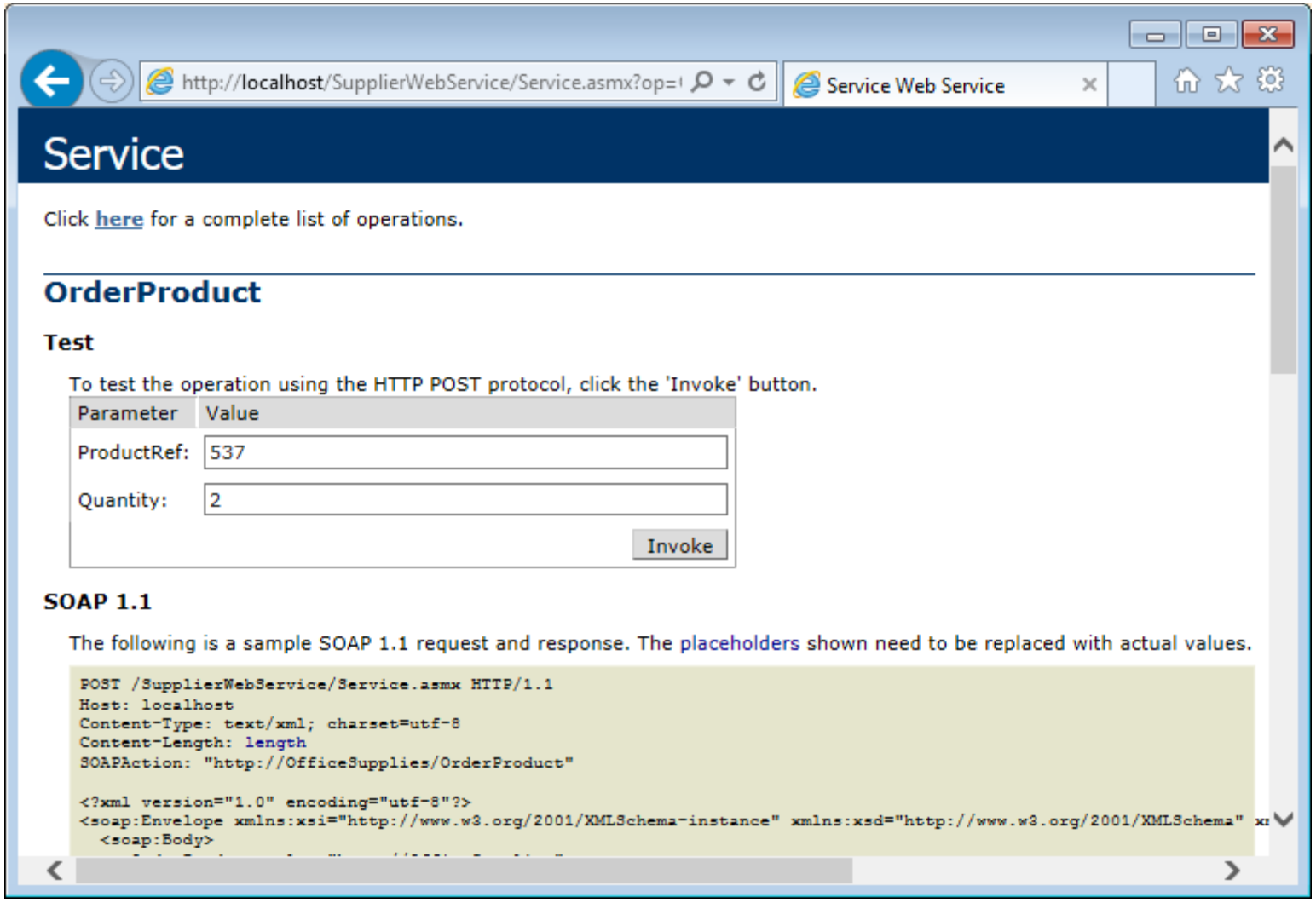
Invoking the Web Service



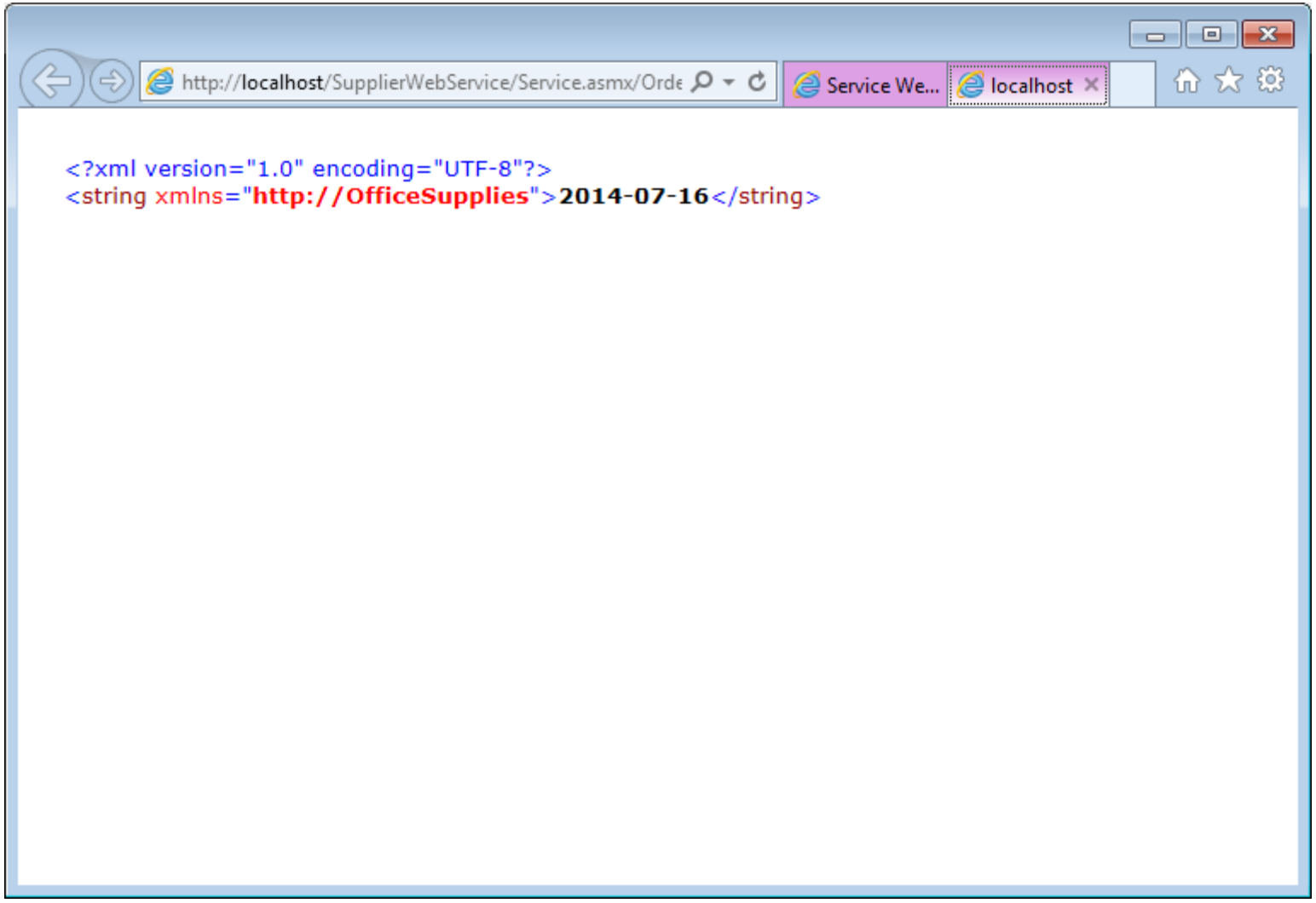
Invoking the Web Service



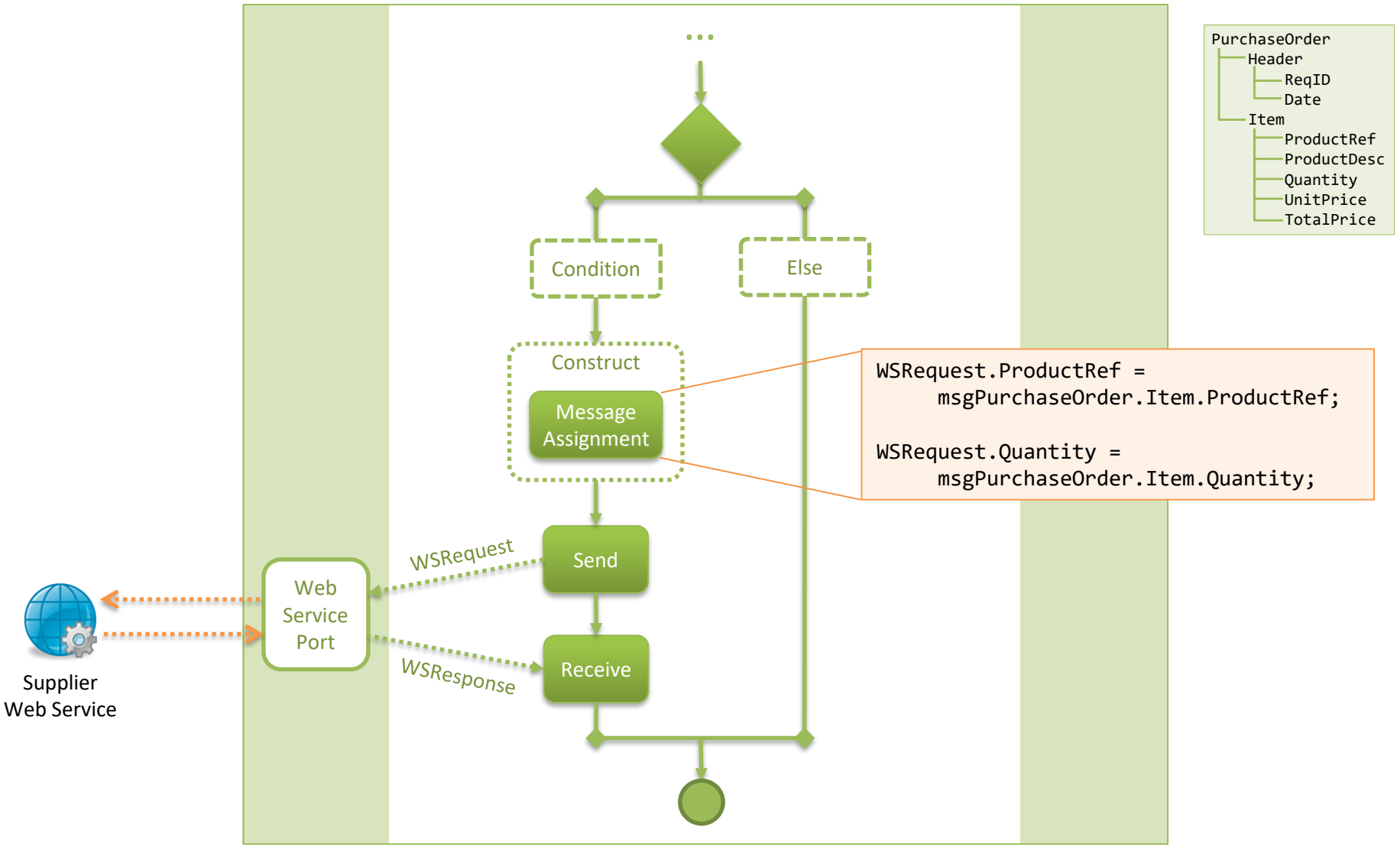
Invoking the Web Service



Invoking the Web Service



Invoking the Web Service

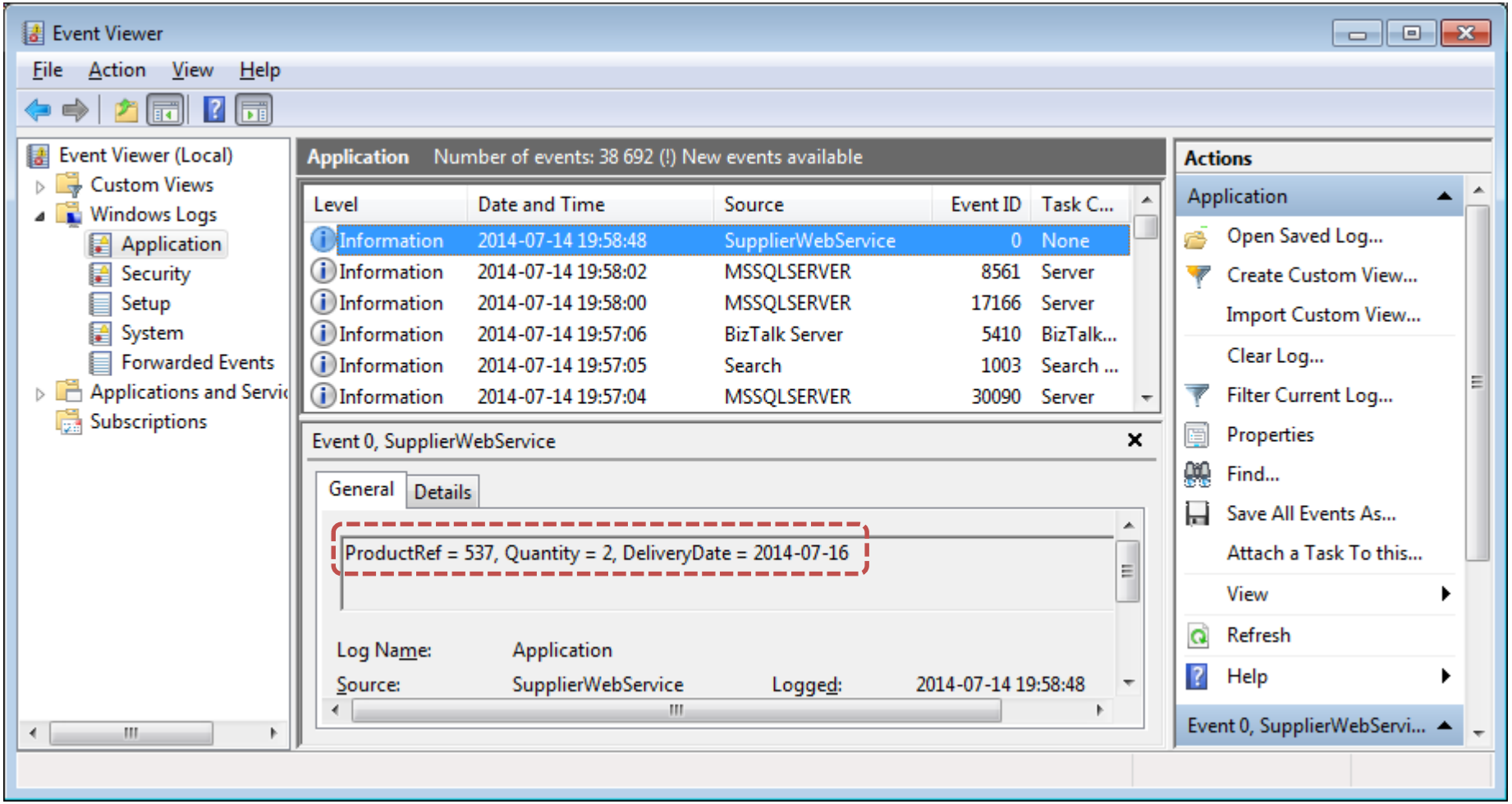


Invoking the Web Service

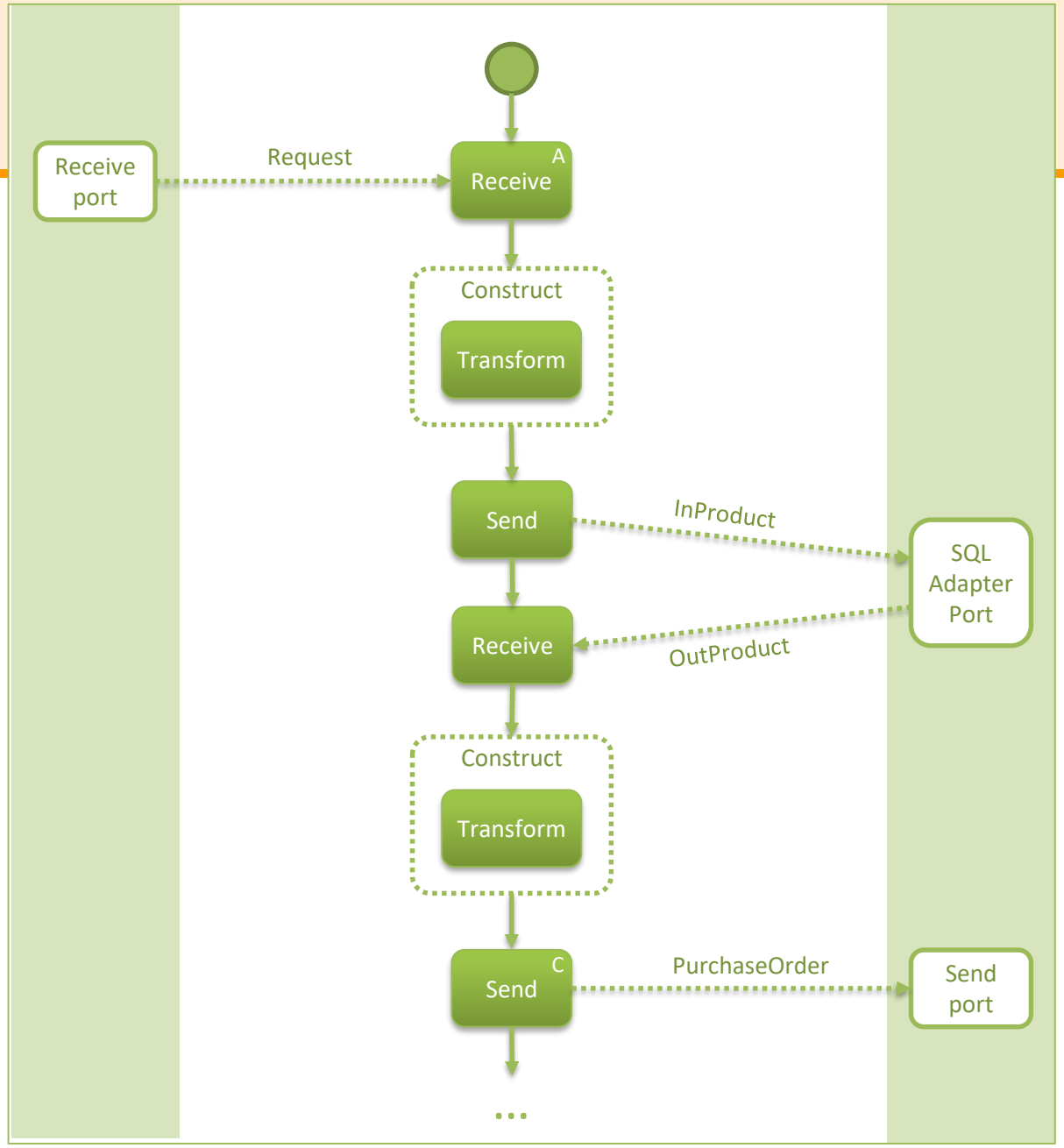
- How do we know if the orchestration called the WS?
  - insert some "debugging" code

```
using System;
using System.Web.Services;

[WebService(Namespace = "http://OfficeSupplies")]
public class Service : WebService
{
    [WebMethod]
    public string OrderProduct(int ProductRef, int Quantity)
    {
        string DeliveryDate = DateTime.Now.AddDays(2).ToString("yyyy-MM-dd");
        string entry = String.Format("ProductRef = {0}, Quantity = {1},
DeliveryDate = {2}", ProductRef, Quantity, DeliveryDate);

        System.Diagnostics.EventLog.WriteEntry("SupplierWebService", entry);
        return DeliveryDate;
    }
}
```

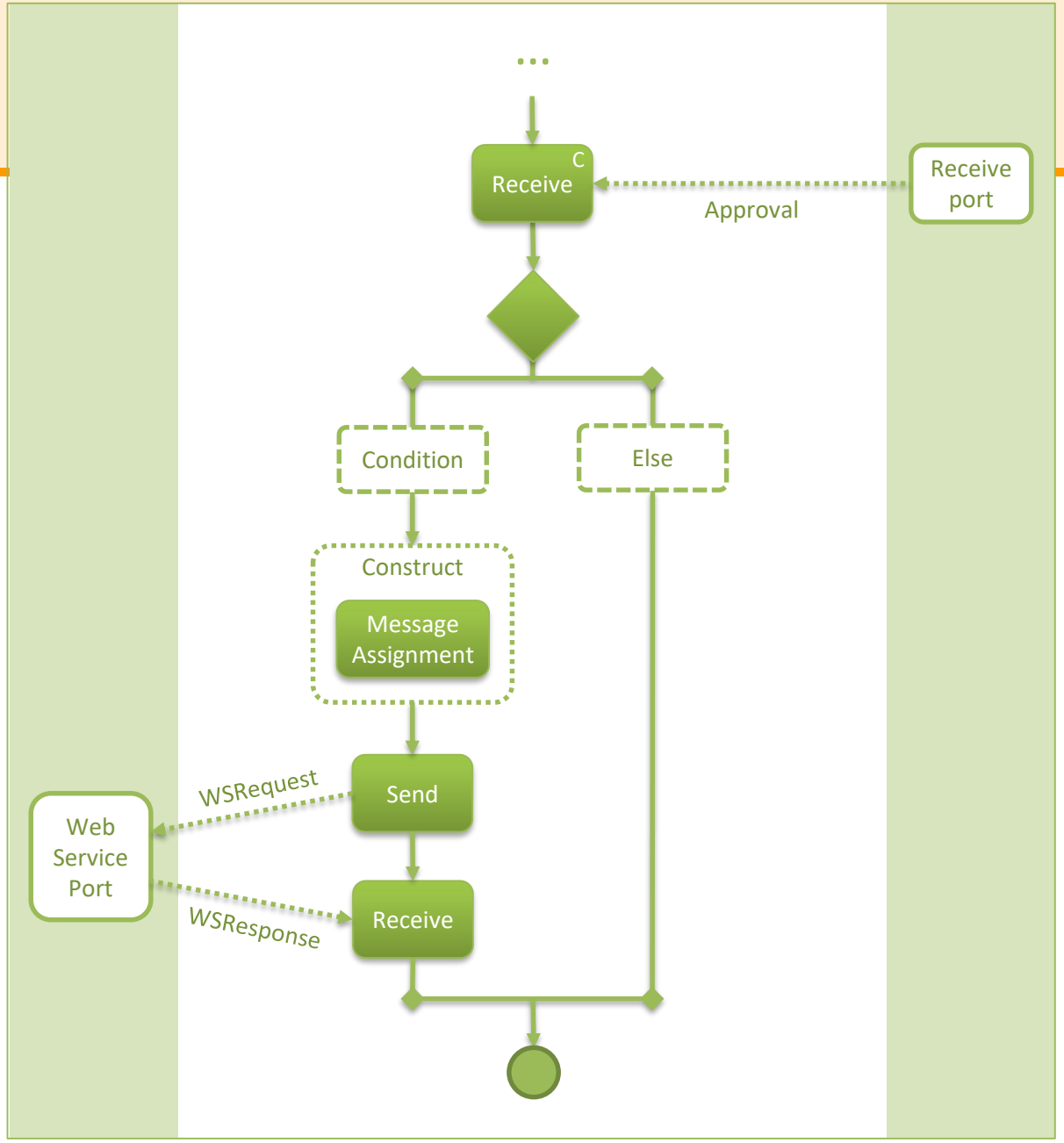
Invoking the Web Service



Overview


Receive port
Request
A
Receive
Construct
Transform
Send
InProduct
SQL Adapter Port
Receive
OutProduct
Construct
Transform
C
Send
PurchaseOrder
Send port
...

Overview


C
Receive
Receive port
Approval
Condition
Else
Construct
Message Assignment
WSRequest
Send
Web Service Port
WSResponse
Receive

Tool support

- There are several tools available
  - Apache ODE
  - Microsoft BizTalk Server
  - JBoss Enterprise SOA Platform
  - OpenESB / Glassfish
  - Oracle Fusion Middleware
  - IBM WebSphere
  - TIBCO BusinessWorks
  - Software AG webMethods
  - etc.

Conclusion

- Current tools for Enterprise Systems Integration draw heavily from BPM and BPM systems
  - similar concepts, similar constructs, similar execution
  - orchestrations can be seen as "low-level" processes

- The concepts of services and SOA are a powerful mechanism to raise the level of abstraction
  - low-level services and low-level orchestrations vs. high-level services and high-level orchestrations

Conclusion

- BPM concepts together with services and orchestrations provide a **systematic approach** to implement business processes on top of enterprise systems