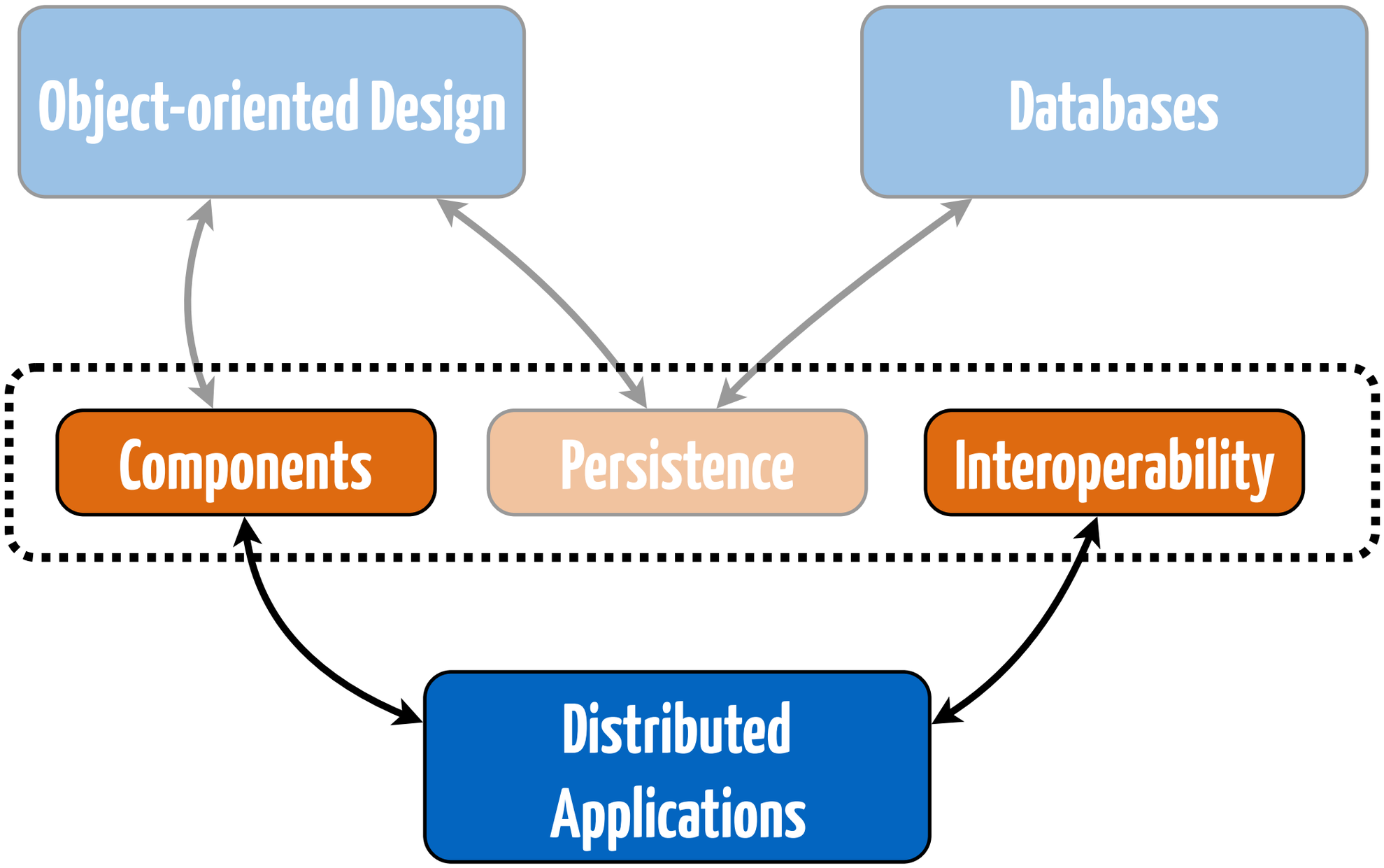
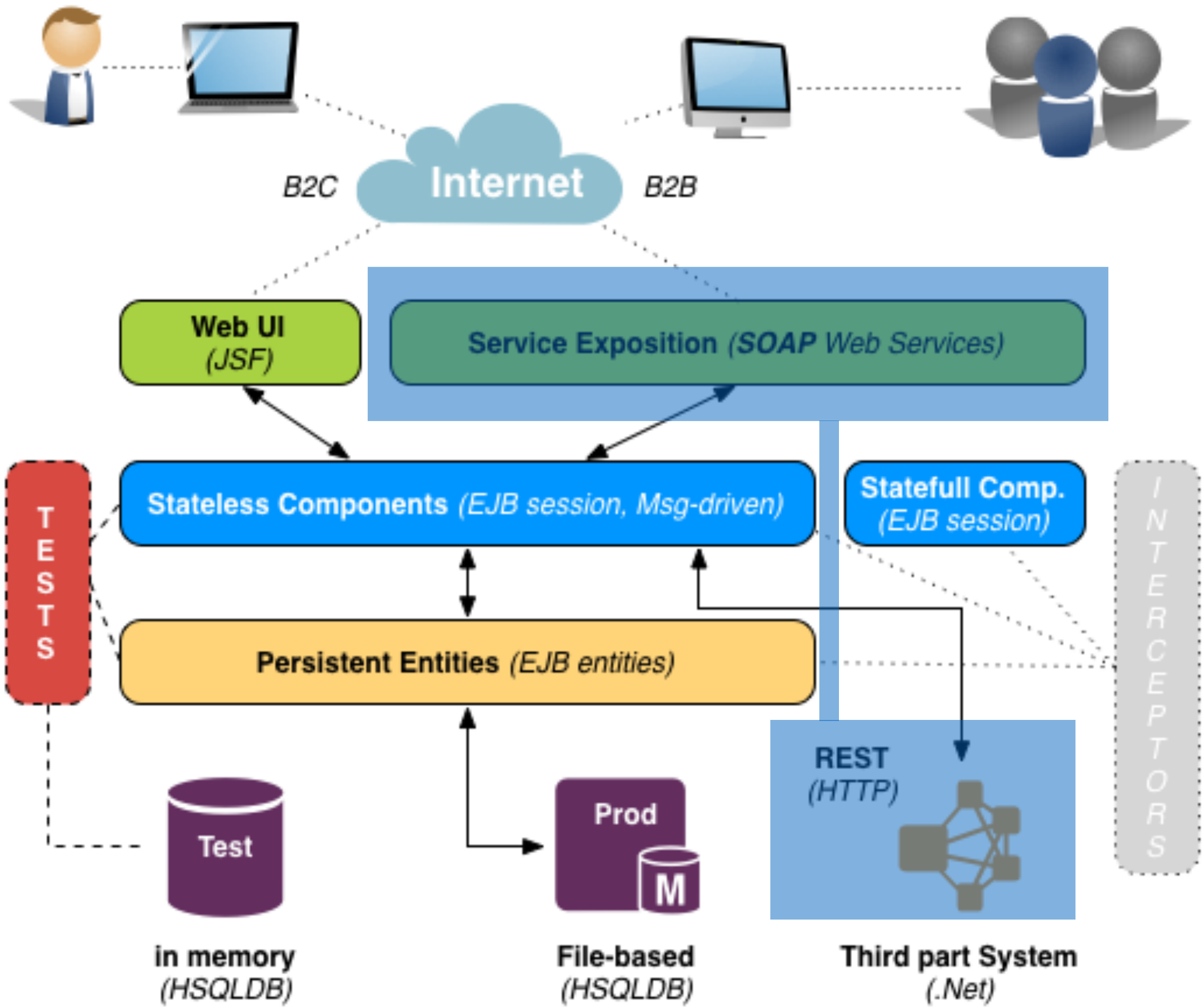
Interoperability with Web Services

Sebastien Mosser

Lecture #3, 16.03.2018

pictures: sxc.hu


Object-oriented Design
Databases
Components
Persistence
Interoperability
Distributed Applications


B2C
Internet
B2B
Web UI
(JSF)
Service Exposition (SOAP Web Services)
TESTS
Stateless Components (EJB session, Msg-driven)
Statefull Comp.
(EJB session)
INTERCEPTORS
Persistent Entities (EJB entities)
REST
(HTTP)
Test
Prod
M
in memory
(HSQLDB)
File-based
(HSQLDB)
Third part System
(.Net)

1 Problem Identification

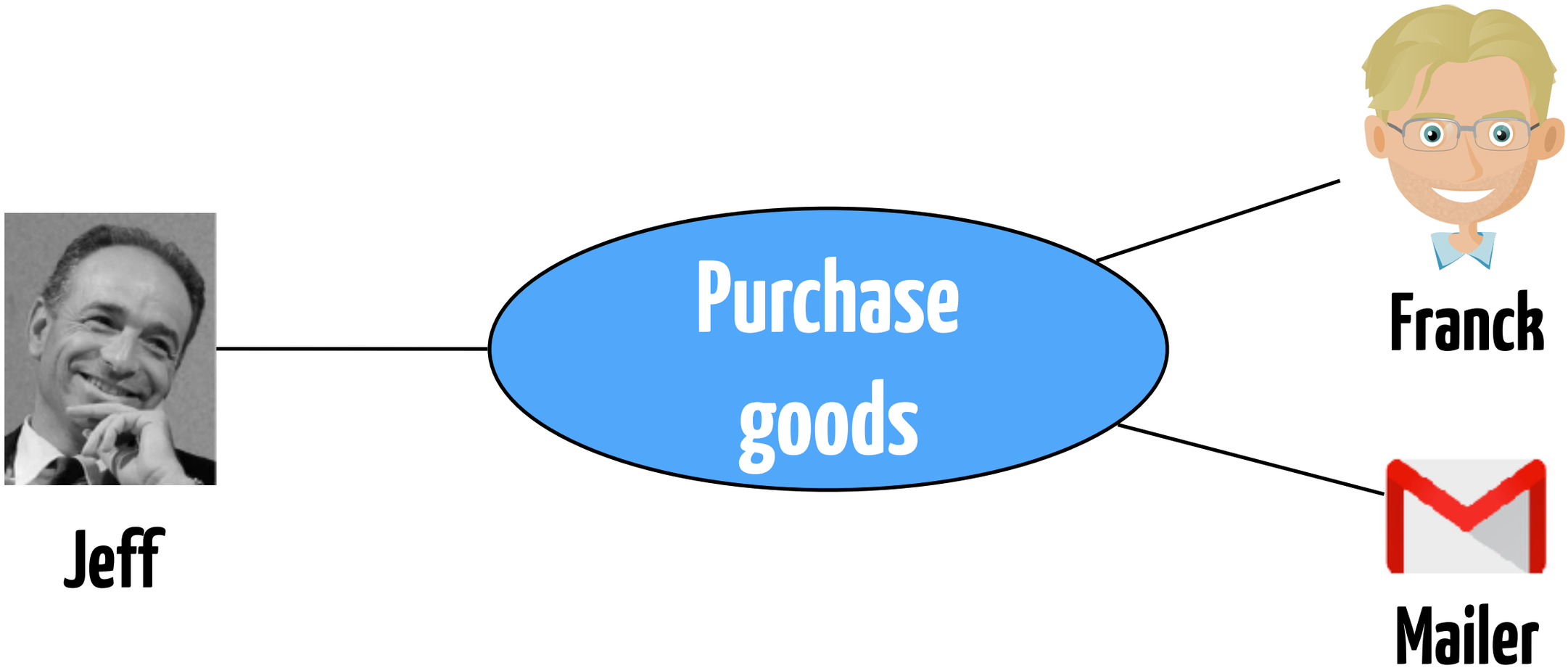
2 Light|No Contract

3 Strong Contract

4 Step Back

Problem Identification

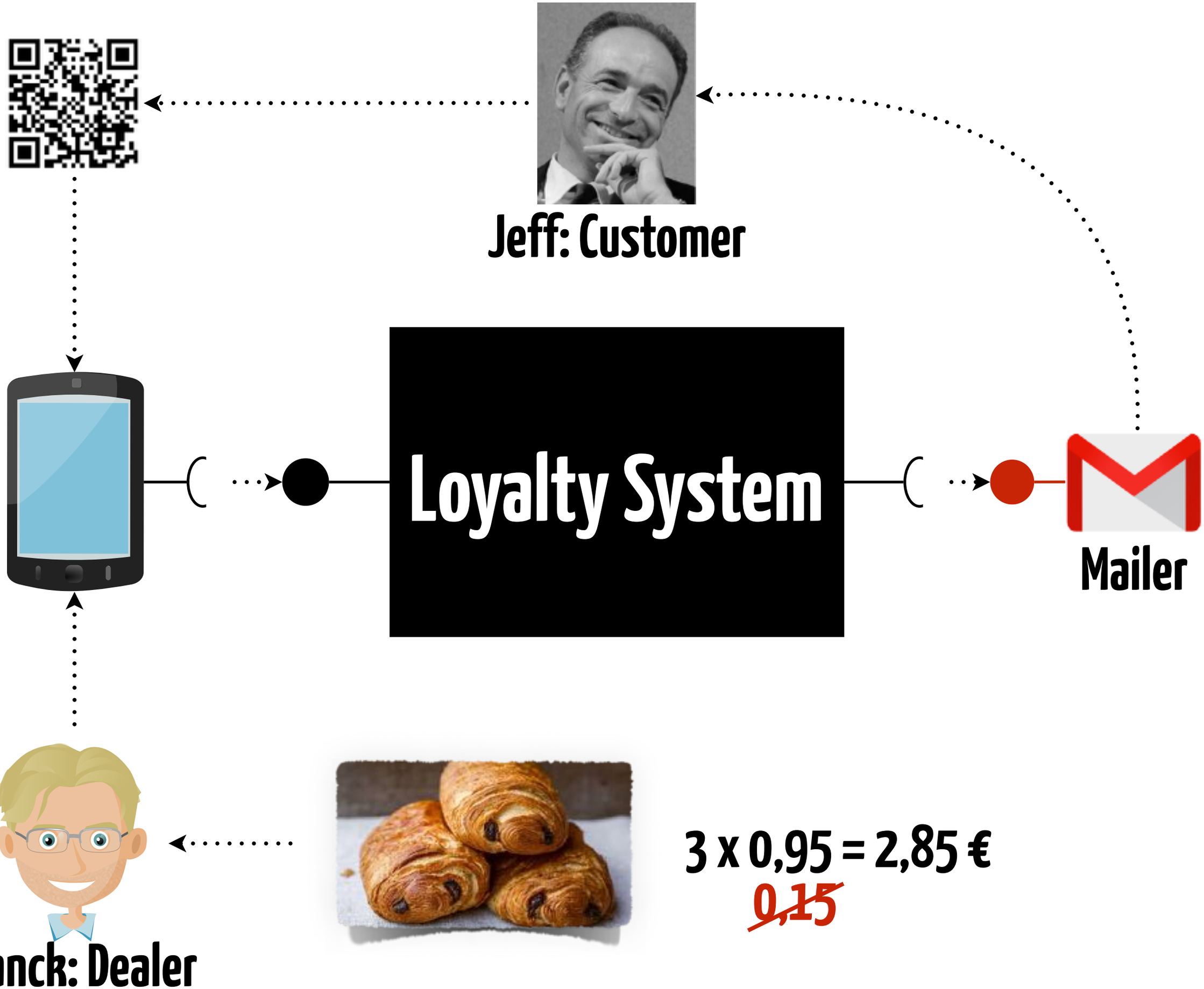
1



As Jeff (Customer),
I want to log a purchase on my card
So that I know my loyalty credit increase

Scenario: Purchase Goods (MVP)

1. Jeff (a Customer) presents a loyalty card and the goods to be purchased;
2. Franck (a Dealer) scans the card, and logs the purchase information;
3. These data are sent to the Loyalty System;
4. The purchase amount is transformed into Loyalty credit points;
5. This amount is added to the balance of the customer (based on the card ID);
6. An email is sent to the user email with the new balance.


Jeff: Customer
Loyalty System
Mailer
Franck: Dealer
3 x 0,95 = 2,85 €
0,15

LogTransaction:

register(???)

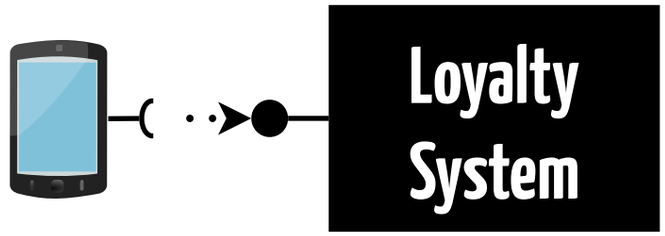
Messaging:

sendMail(data: Message)

Client

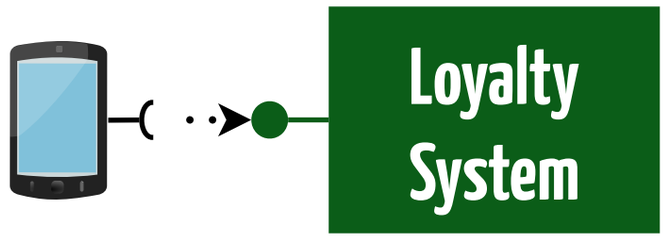
Our System

External partner



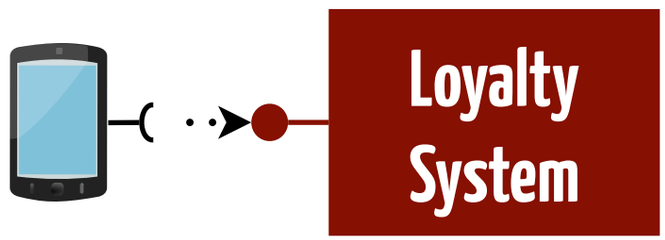
LogTransaction:

register(shop: Shop, card: Image, prod: Product, quantity: Int)



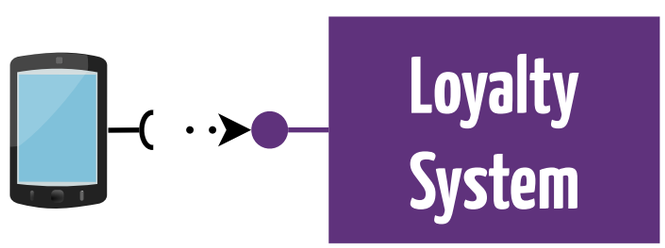
LogTransaction:

register(shop: ID, card:ID, product: Product, value: Float)



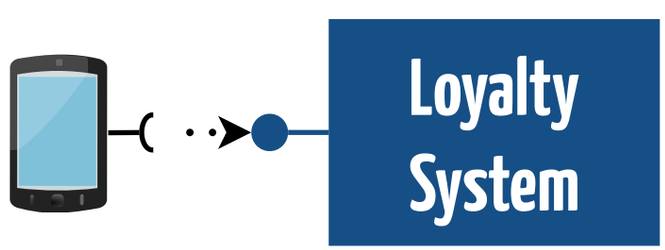
LogTransaction:

register(shop: ID, card:ID, product: ID, value: Float)



LogTransaction:

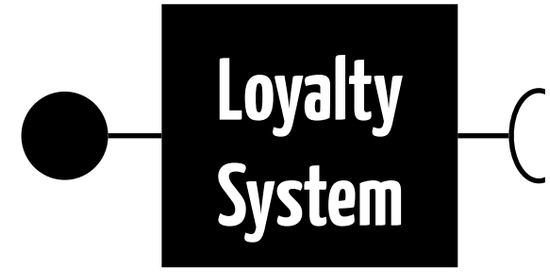
register(shop: ID, card:ID, value: Float)

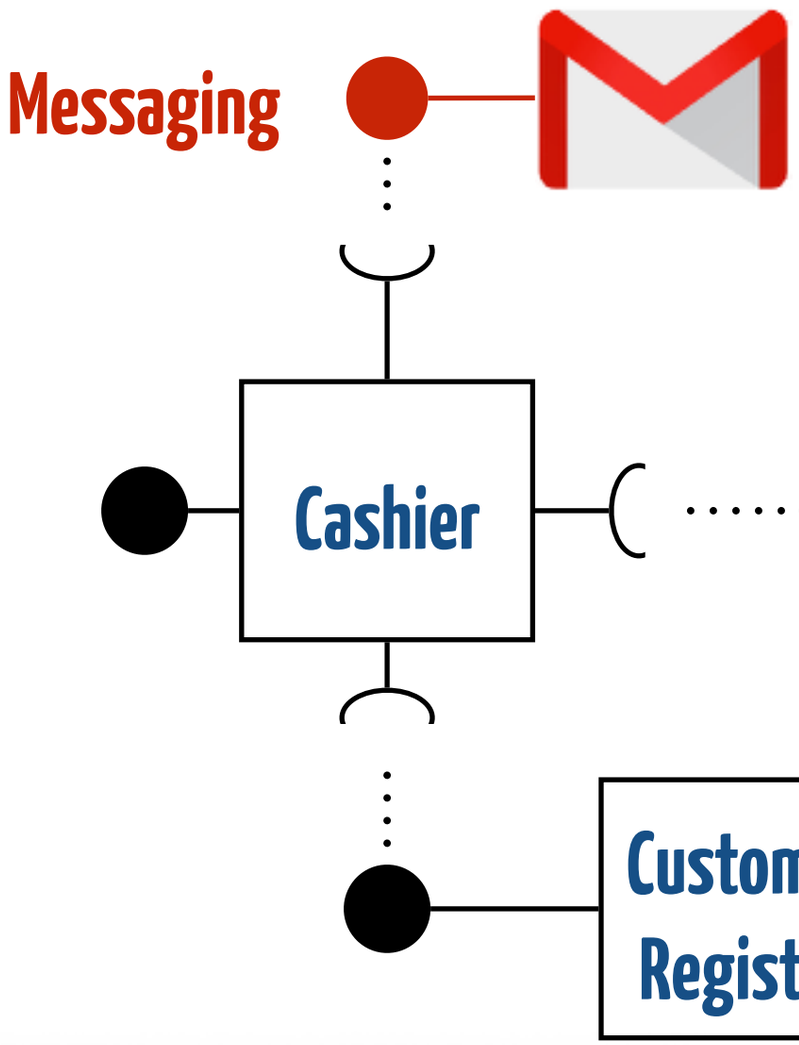


LogTransaction:

register(transaction: Transaction)

Componentizing the system





StockExchange:

exchange(amount: float) → int

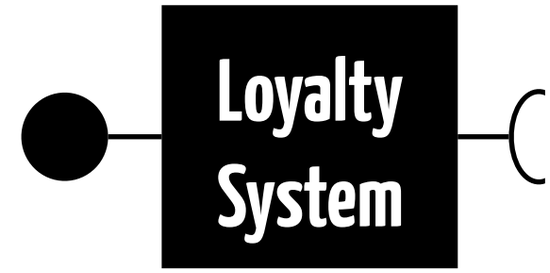
Registry:

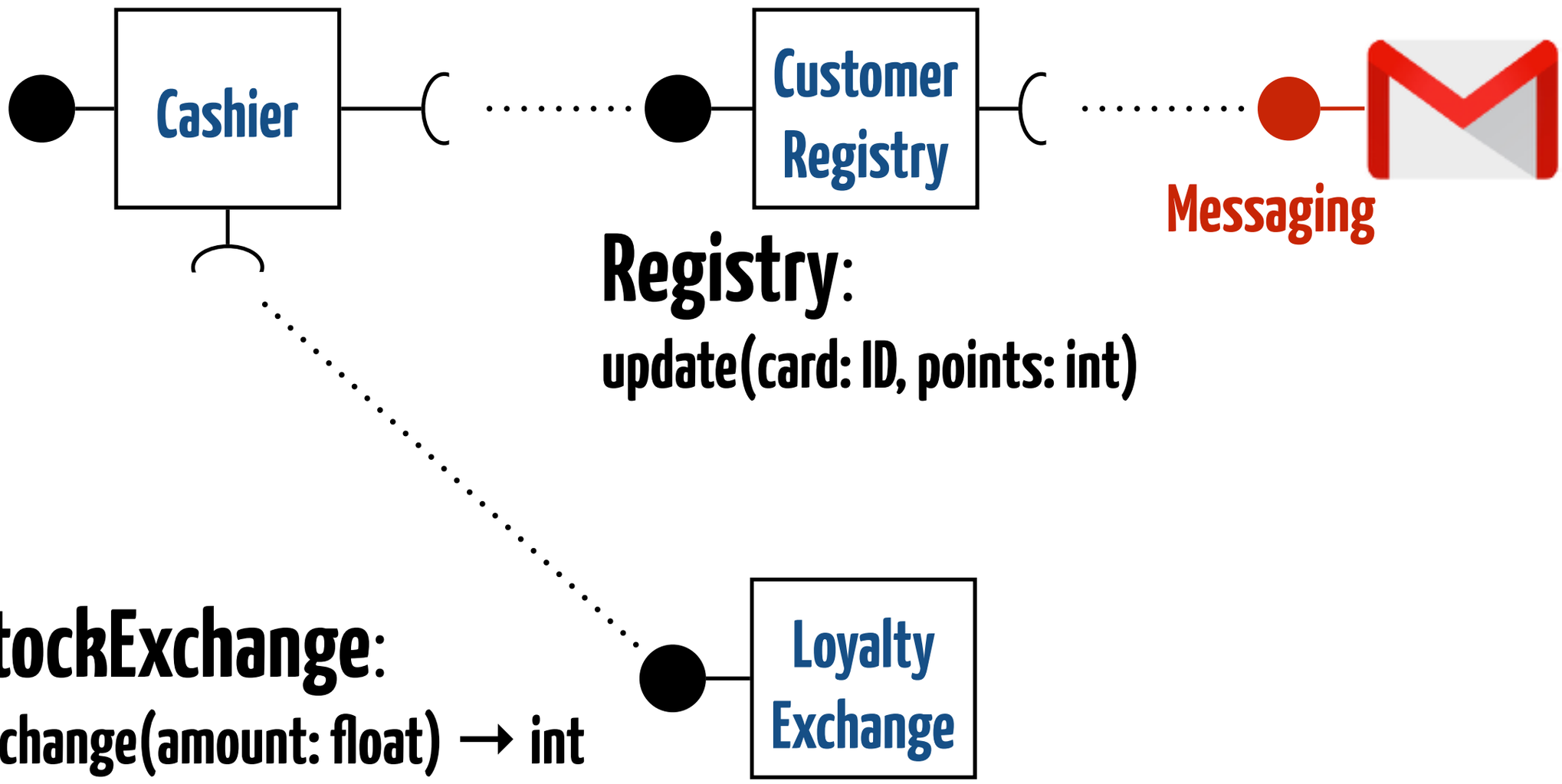
update(card: ID, points: int) → EmailAddress

4. The purchase amount is transformed into Loyalty credit points;
5. This amount is added to the balance of the customer (based on the card ID);
6. An email is sent to the user email with the new balance.

#1

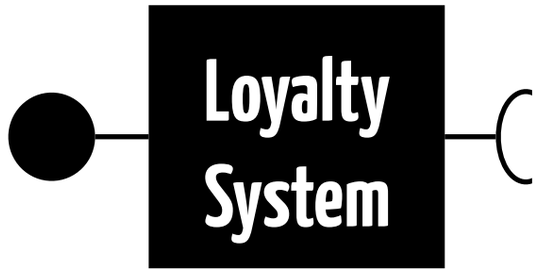
Componentizing the system



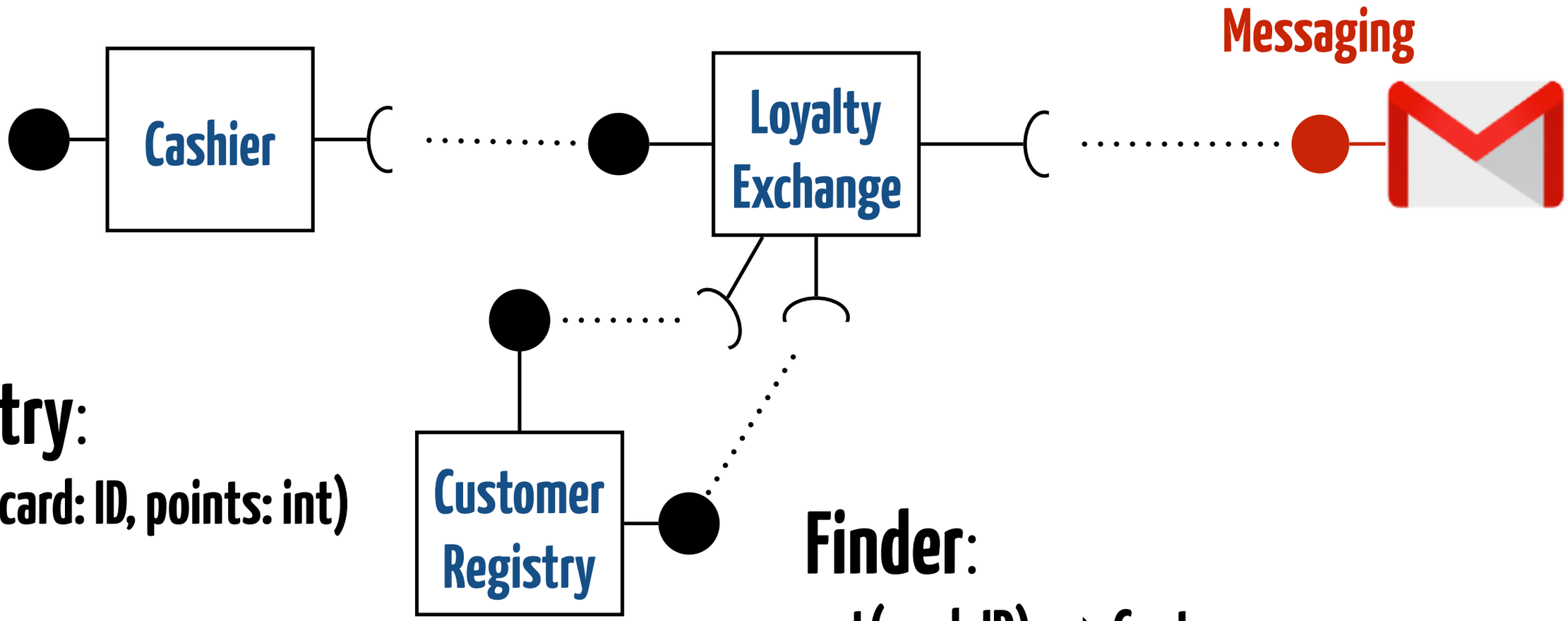


#2

Componentizing the system



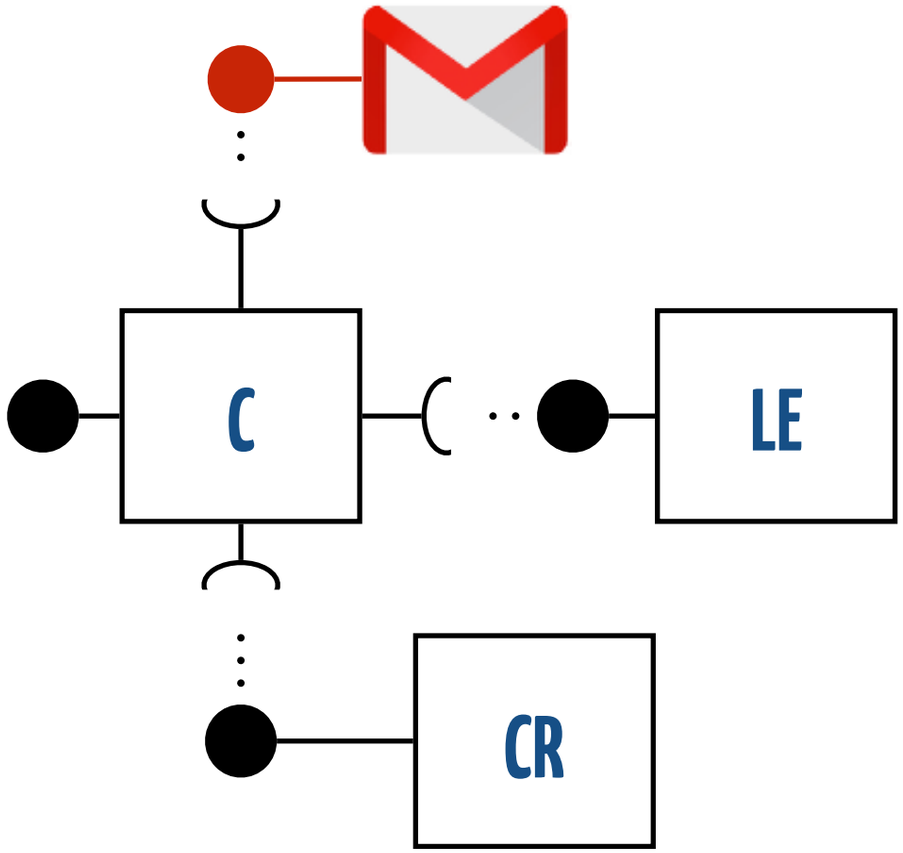
StockExchange:
exchange(amount: float, card: ID)

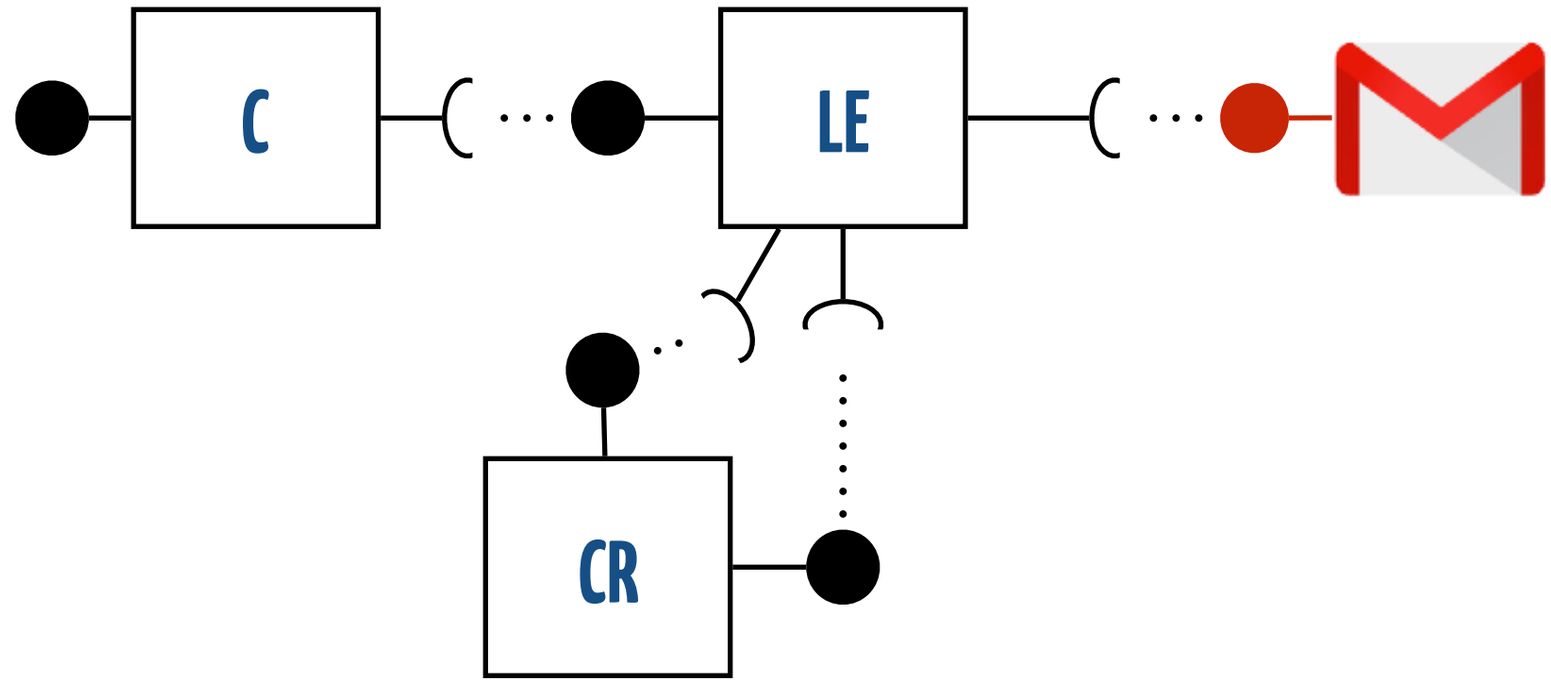


Registry:
update(card: ID, points: int)

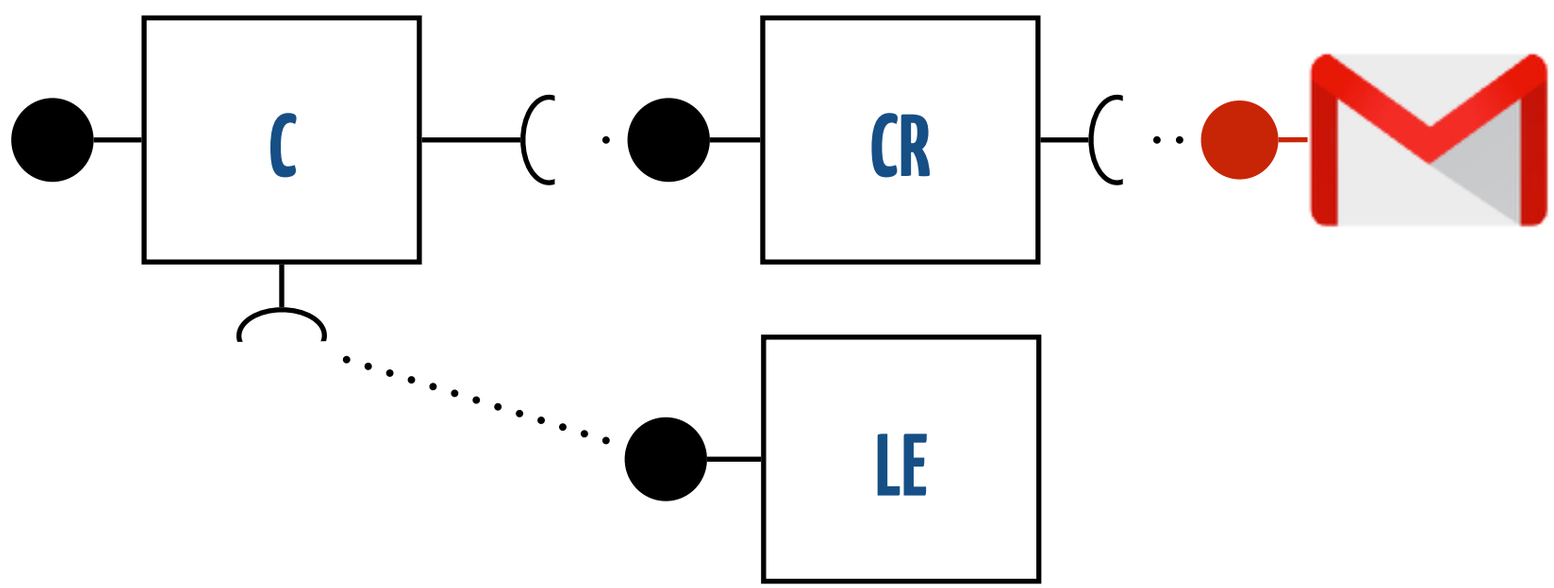
Finder:
get(card: ID) → Customer

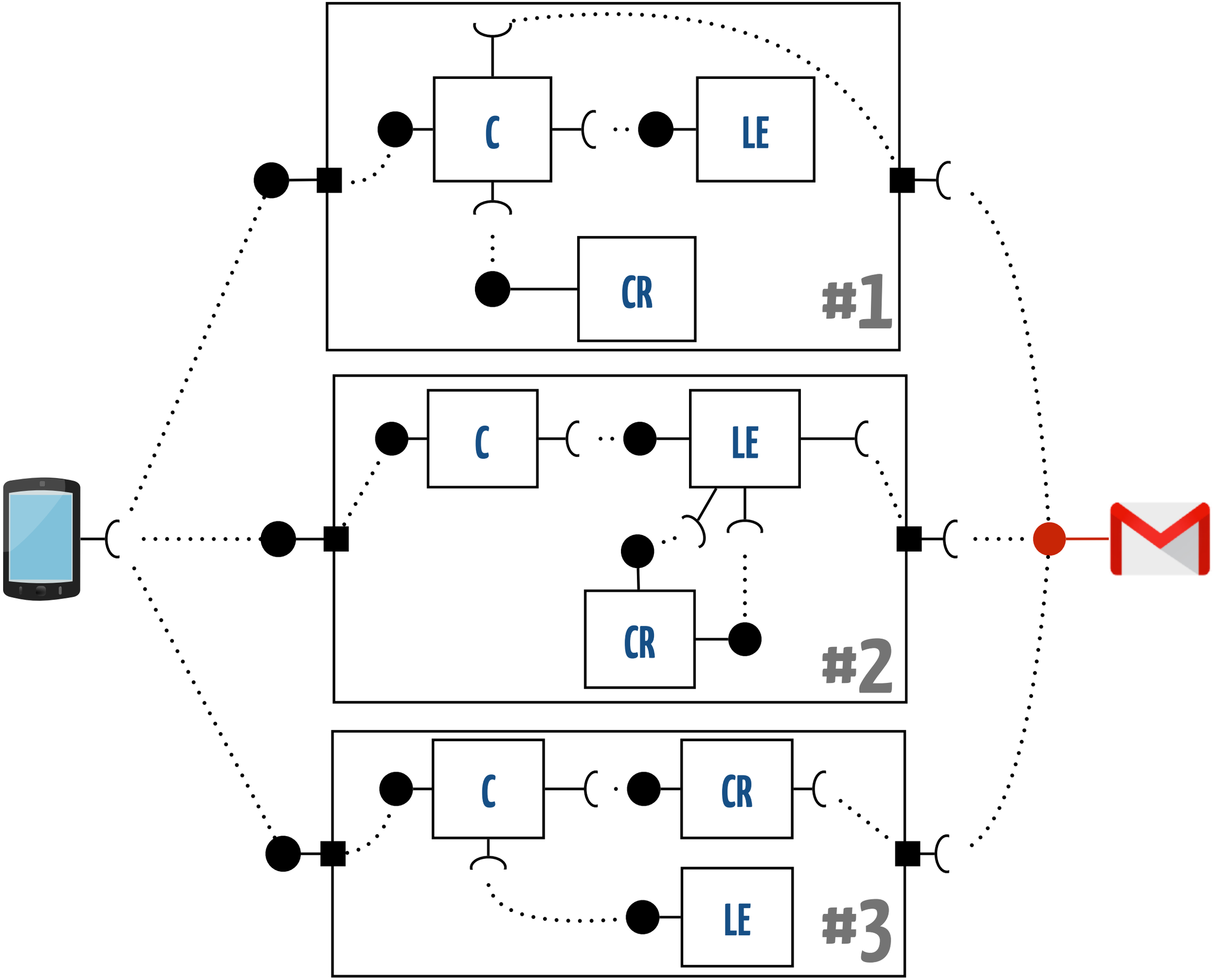
#3

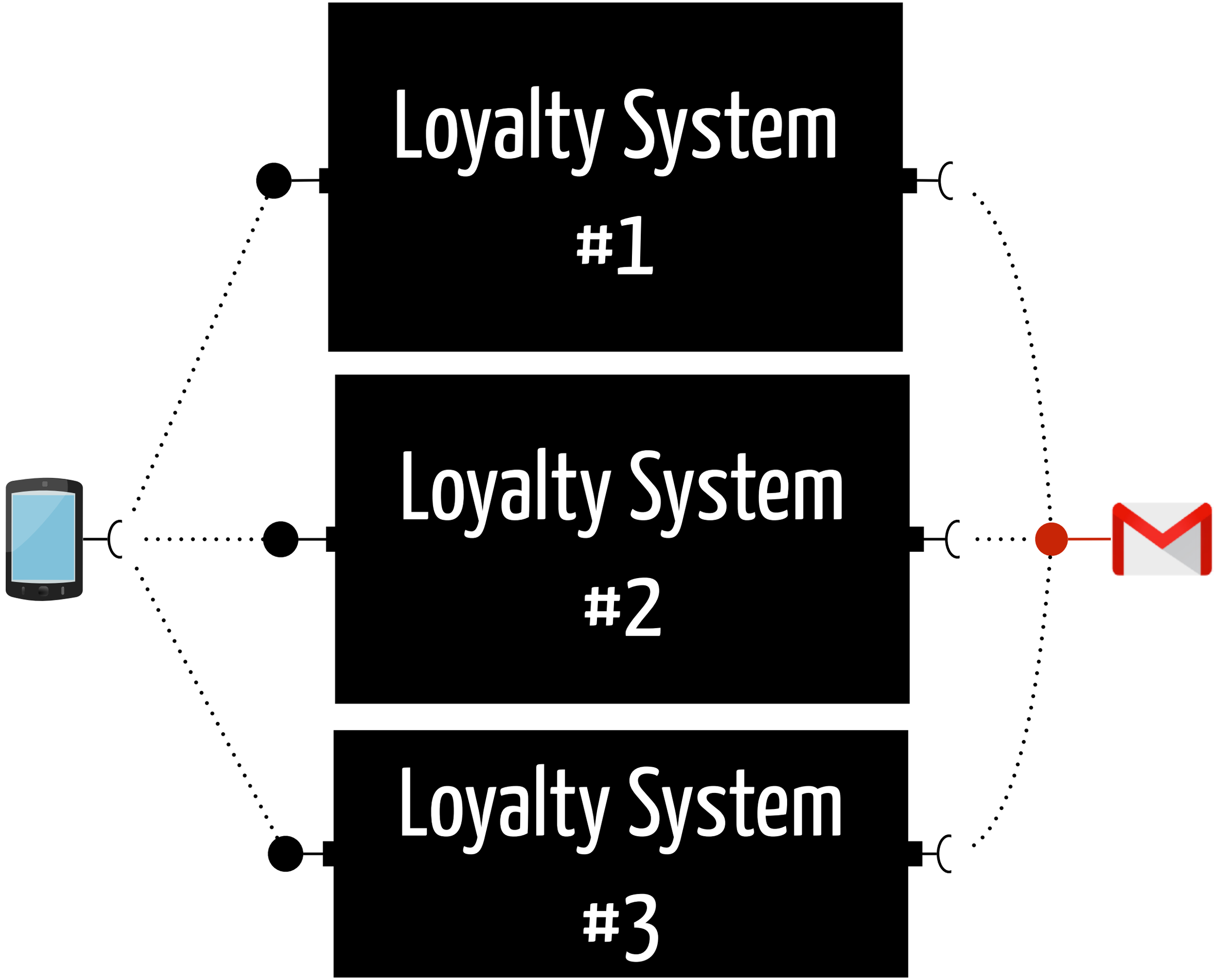

C
LE
CR


C
LE
CR

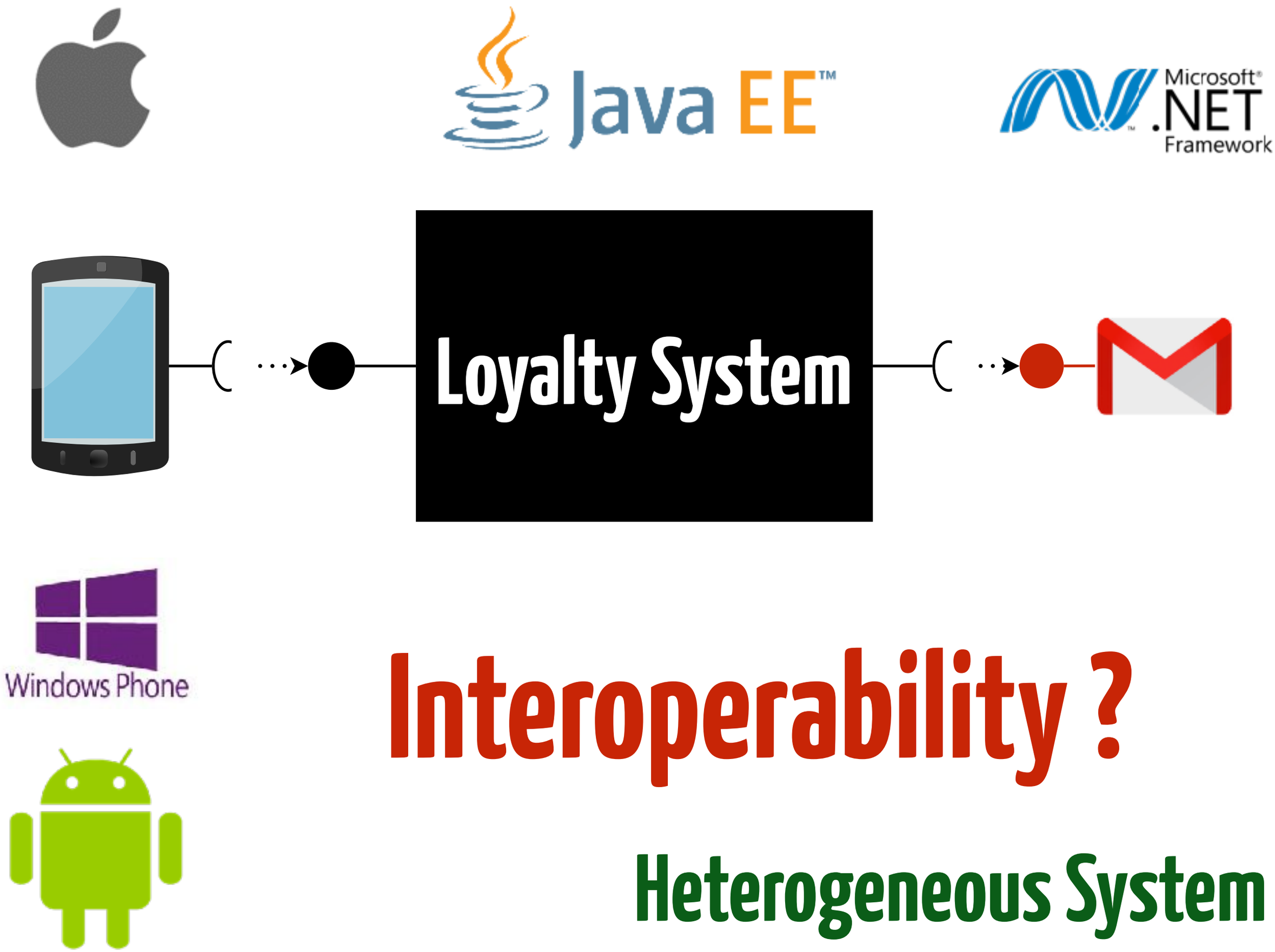

Deal With It


C
CR
LE


C
LE
CR
#1
C
LE
CR
#2
C
CR
LE
#3


Loyalty System #1
Loyalty System #2
Loyalty System #3

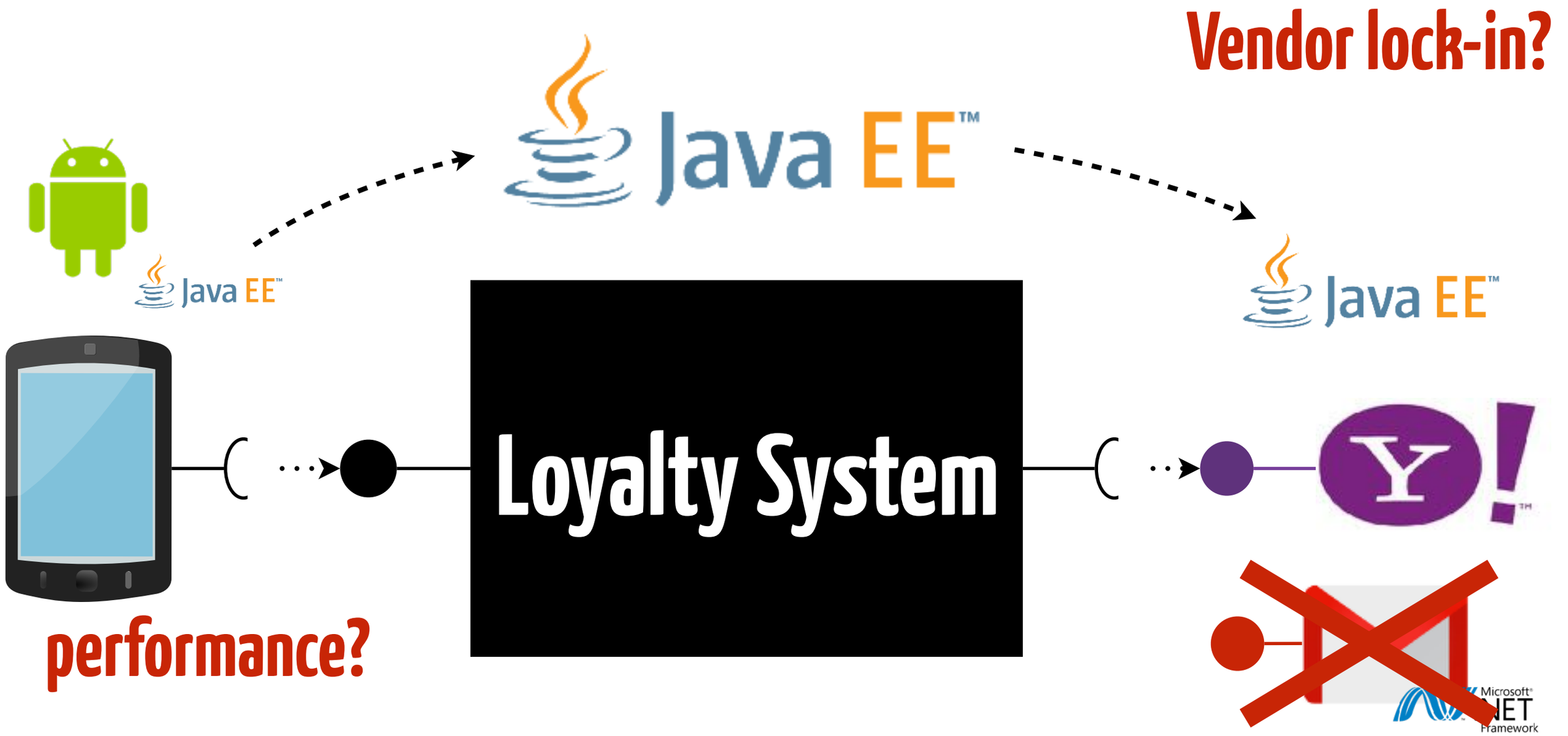
Public APIs support flexibility



Interoperability ?

Heterogeneous System

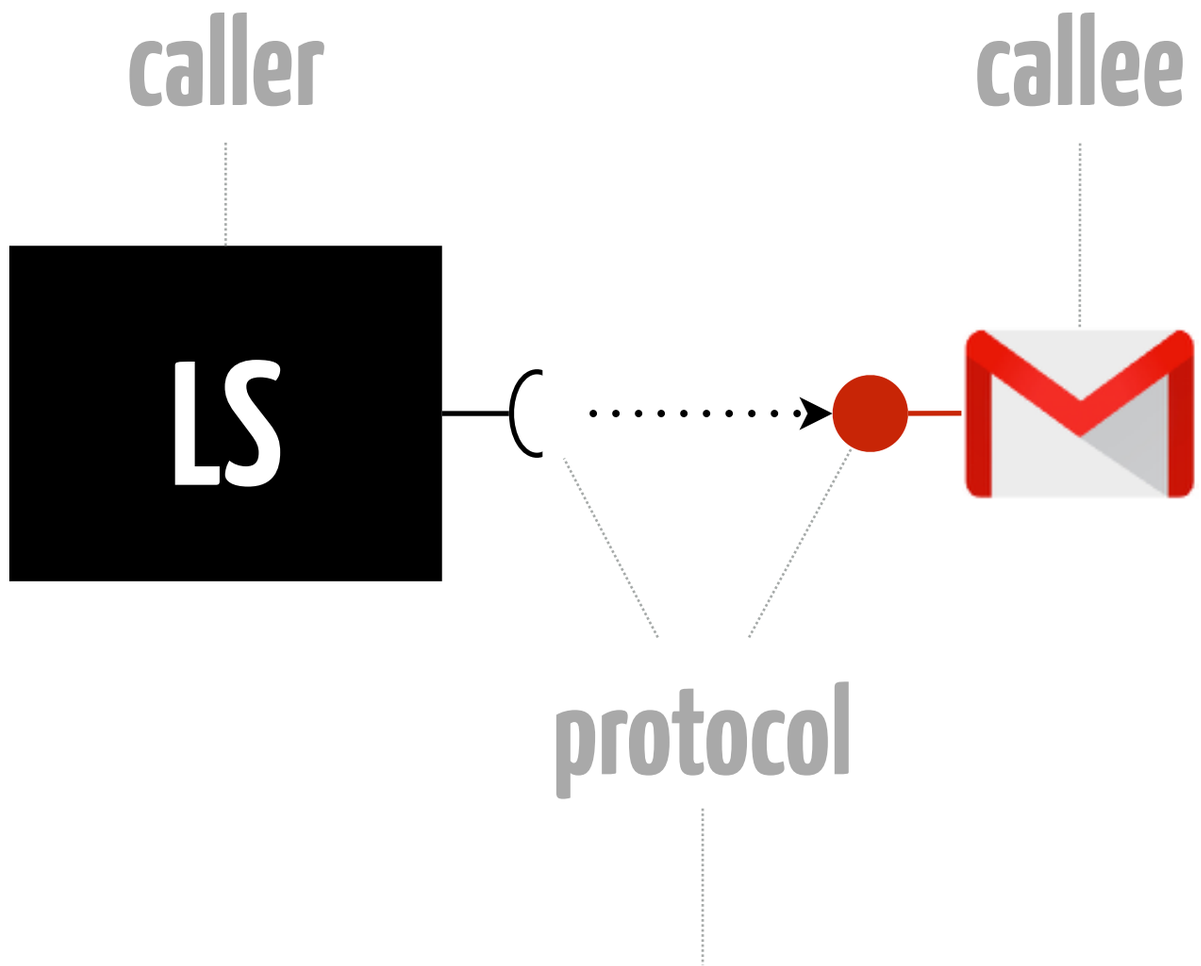
Using J2E dependency injection



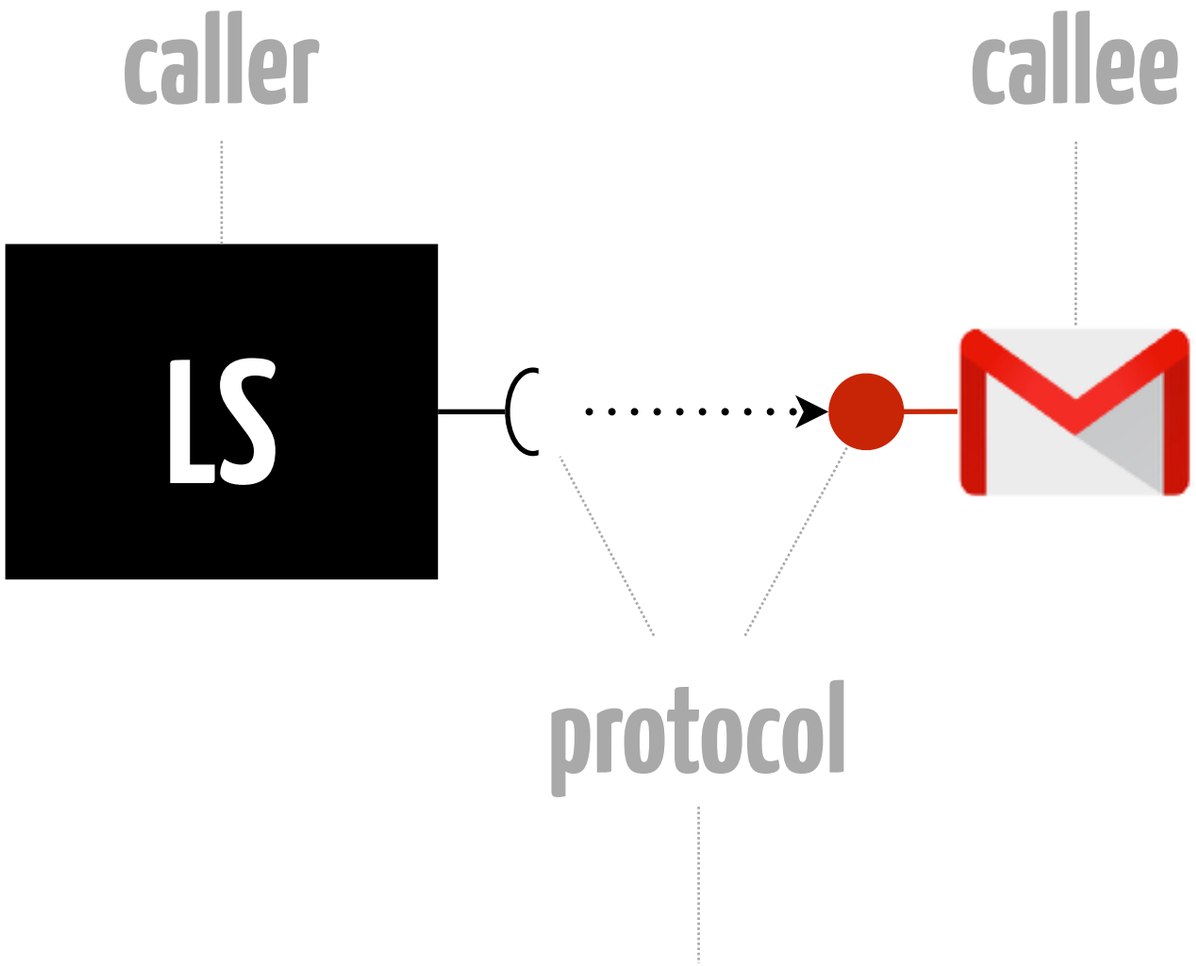
technology driving business?

Homogeneous System

Abstracting from Implementation



- Endpoint: Where, How ?
- Operations: Why ?
- Business Object: What ?



- Endpoint: Where, How ?
- Operations: Why ?
- Business Object: What ?

Messaging:

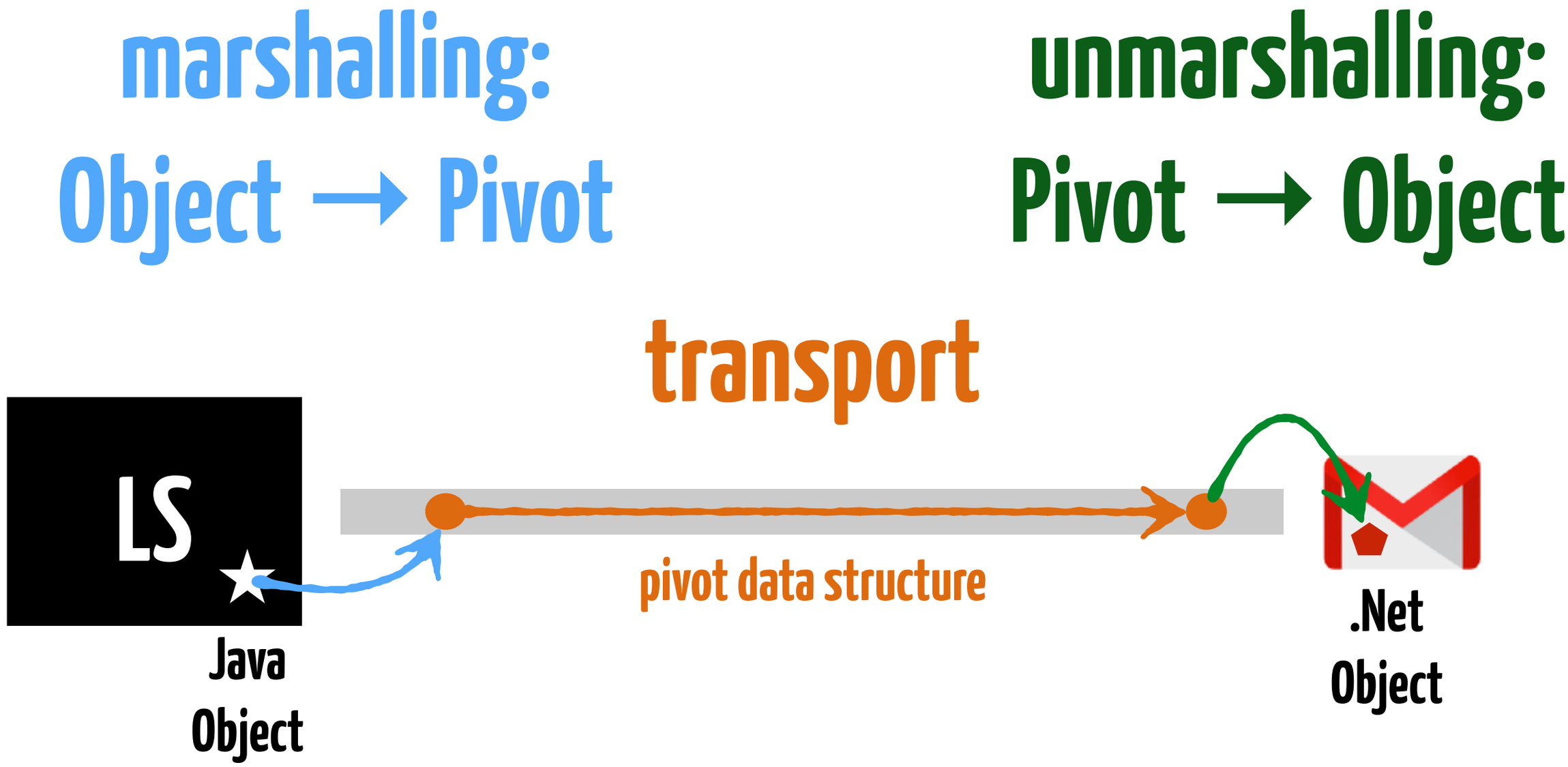
sendMail(data: Message)

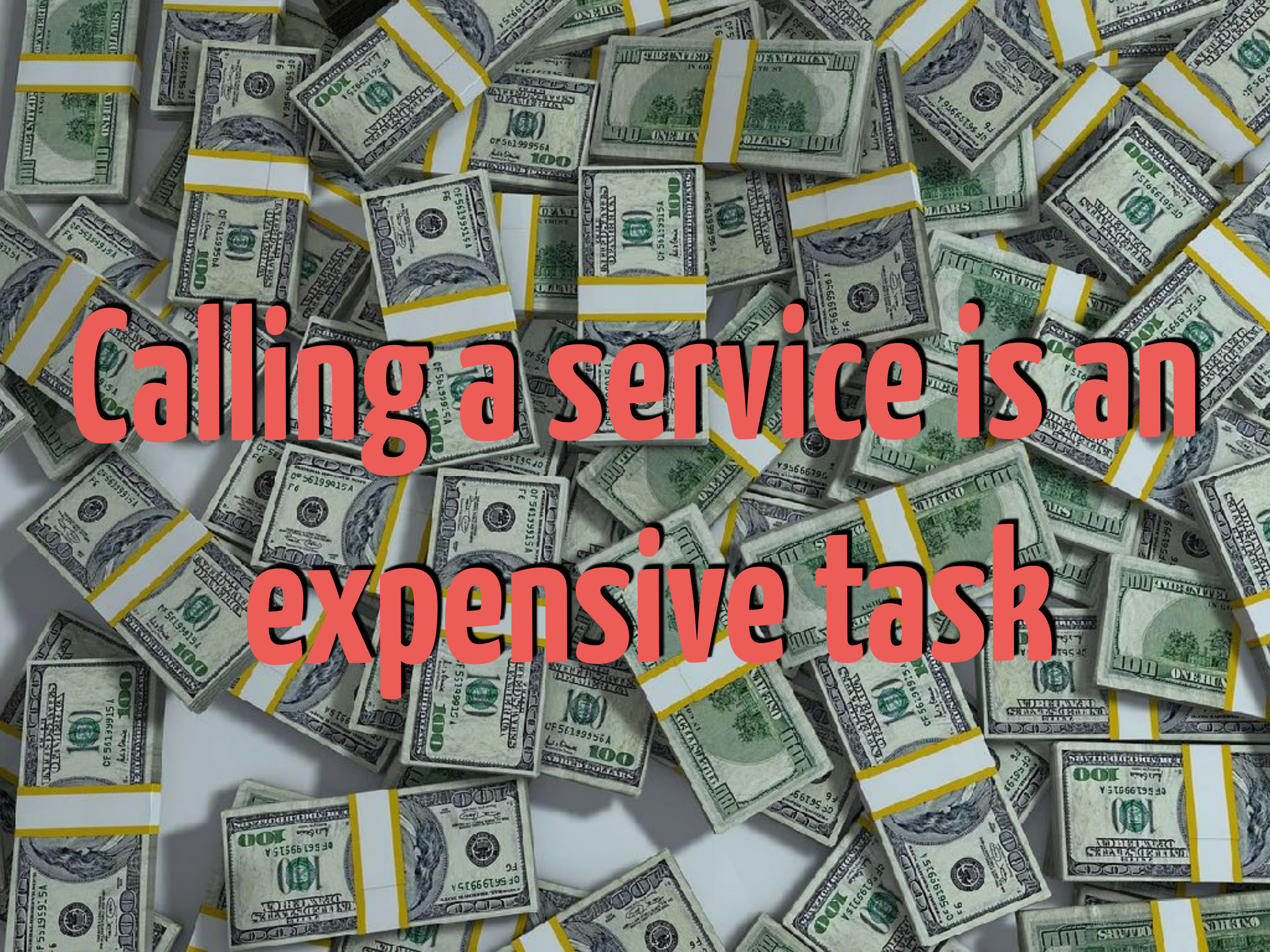
Defined in the interface

Endpoint

- **Where**:
  - IP Address
  - hostname (resolved to IP)
- **How**:
  - Communication protocol (e.g., HTTP)
  - Data Encoding (e.g., XML, JSON)

Platform Independent


marshalling:
Object → Pivot
unmarshalling:
Pivot → Object
transport
LS
Java
Object
pivot data structure
.Net
Object


Calling a service is an
expensive task

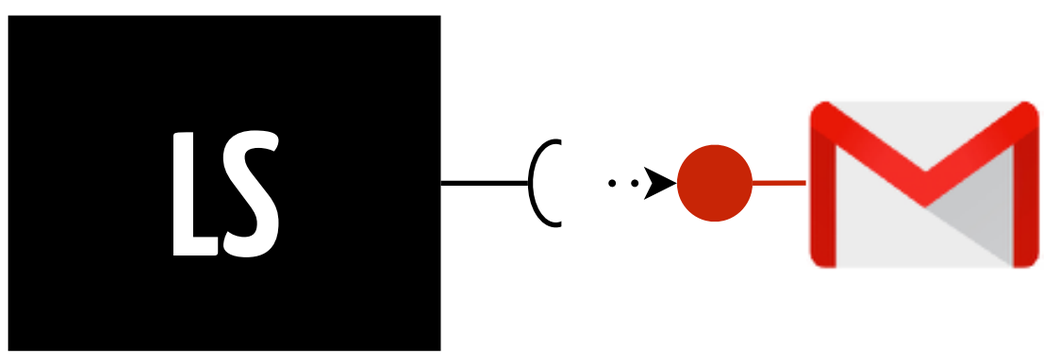
Public APIs support interoperability

The Cookie Factory open source example

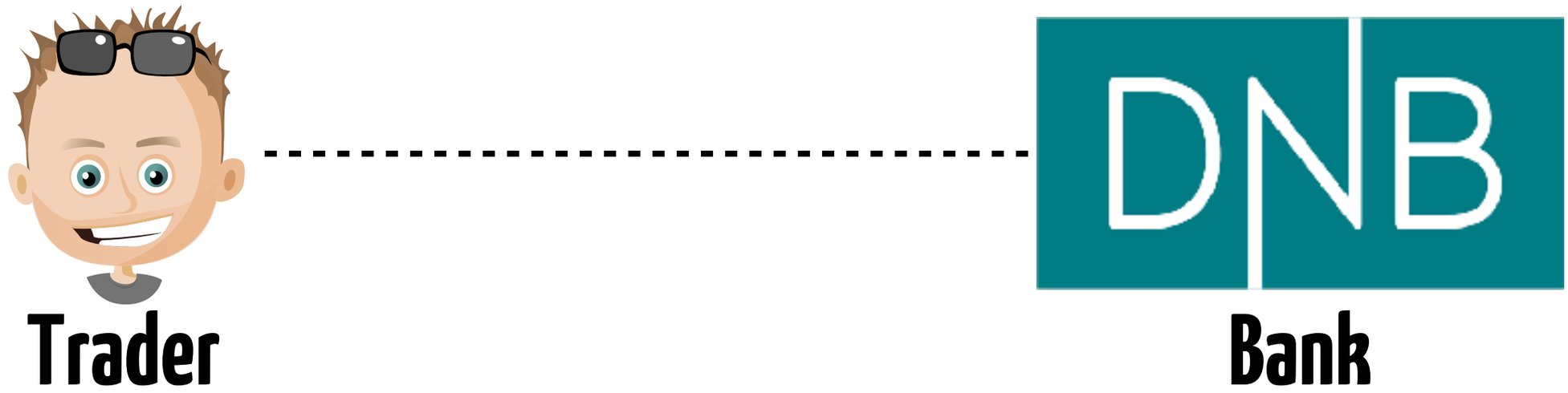
https://github.com/polytechnice-si/4A_ISA_TheCookieFactory/blob/develop/chapters/Exposing_SOAP.md

https://github.com/polytechnice-si/4A_ISA_TheCookieFactory/blob/develop/chapters/Consuming_REST.md

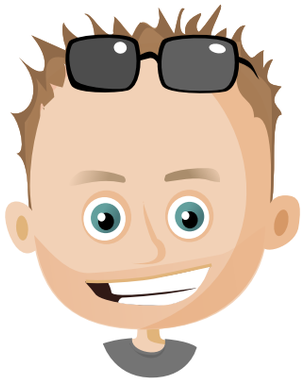
Light|No Contract



2



- Operations :
  - Send a payment request to be processed by the bank
  - Describe a given payment (status, ...)
  - List all received payments from the calling trader
- Protocol:
  - Plain HTTP (GET, POST, ...)
  - Data encoding using JSON



Trader

Bank

- Send a payment request to be processed by the bank
  - POST /mailbox { “card”: “123456780”, “amount”: 2.85 } ↦ 42
- Describe a given payment (status, ...)
  - GET /payments/42 ↦ {“card”: “...”, “amount”: 2.85, “status”: “OK” , ...}
- List all received payments from the calling trader
  - GET /payments ↦ [“42”, “24”, ...]

Exercice

- A Customer owns Orders
- An Order contains Items
- An Item binds a CookieType to a given Quantity (as an integer)

Which Resource interface ?

CREATE / READ / UPDATE / DELETE

- Customers:
  - POST /customers
  - GET /customers/{id}
    - GET /customers/{id}/orders
  - PUT /customers/{id}
  - DELETE /customers/{id}
- Orders:
  - POST /orders
  - GET /orders/{id}
  - PUT /orders/{id}
  - DELETE /orders/{id}
- Items:
  - ...

Where is my business ?

CRUD services leads and promoted to the database as a service kind of thinking (e.g. ADO.NET data services) which as I explained in another post last yearis a bad idea since:

1. It circumvents the whole idea about "Services" - there's no business logic.
2. It is exposing internal database structure or data rather than a thought-out contract.
3. It encourages bypassing real services and going straight to their data.
4. It creates a blob service (the data source).
5. It encourages minuscule demi-services (the multiple "interfaces" of said blob) that disregard few of the fallacies of distributed computing.
6. It is just client-server in sheep's clothing.

http://www.drdobbs.com/architecture-and-design/crud-is-bad-for-rest/228700407

Seriously ...

REST ≠ CRUD

Being “RESTful” is **way much more** than sending GET & POST to URLs

When you talk about REST, be sure of what you are talking about

(seriously. http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api)

Describing the Business Objects


```
[DataContract(Namespace = "http://partner/external/payment/data/",
              Name = "PaymentRequest")]
public class PaymentRequest
{
  [DataMember]
  public string CreditCard { get; set; }

  [DataMember]
  public double Amount { get; set; }
}
```

No methods. Structure only.

Describing the Interface


```
[ServiceContract]
public interface IPaymentService
{

  [OperationContract]
  [WebInvoke( Method = "POST", UriTemplate = "mailbox",
              RequestFormat = WebMessageFormat.Json,
              ResponseFormat = WebMessageFormat.Json)]
  int ReceiveRequest(PaymentRequest request);

  [OperationContract]
  [WebInvoke( Method = "GET", UriTemplate = "payments/{identifier}",
              ResponseFormat = WebMessageFormat.Json)]
  Payment FindPaymentById(int identifier);

  [OperationContract]
  [WebInvoke( Method = "GET", UriTemplate = "payments",
              ResponseFormat = WebMessageFormat.Json)]
  List<int> GetAllPaymentIds();

}
```

Implementing the service

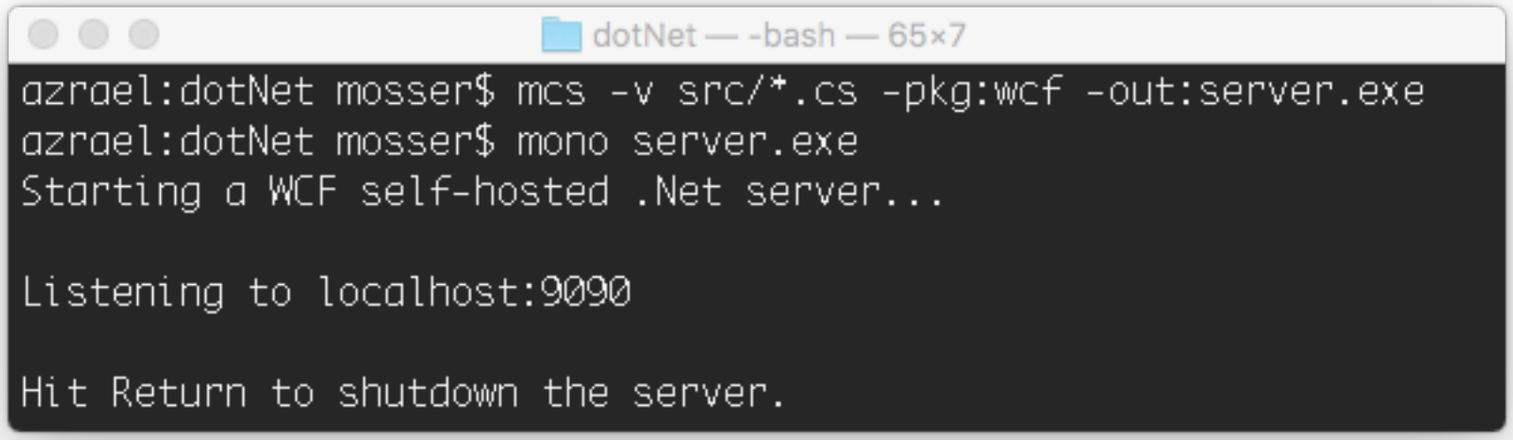

```
public class PaymentService : IPaymentService
{

  public int ReceiveRequest(PaymentRequest request)
  {
    Console.WriteLine("ReceiveRequest: " + request);
    var payment = BuildPayment(request);
    accounts.Add(counter, payment);
    return counter;
  }

  // ...
}
```

Mocked Implementation

Starting a self-hosted server




```
public void start()
{
  Console.WriteLine("Starting a WCF self-hosted .Net server... ");
  string url = "http://" + "localhost" + ":" + Port;

  WebHttpBinding b = new WebHttpBinding();
  Host = new WebServiceHost(typeof(PaymentService), new Uri (url));

  // Adding the service to the host
  Host.AddServiceEndpoint(typeof(IPaymentService), b, "");

  // Staring the Host server
  Host.Open();
  Console.WriteLine("\nListening to " + "localhost" + ":" + Port + "\n");

  if ( Standalone ) { lockServer(); } else { interactive(); }

}
```

Plain HTTP communication

```
azrael:~ mosser$ REQUEST='{ "CreditCard": "1234-896983", "Amount": 12.09 }'
azrael:~ mosser$ BASE_URL="http://localhost:9090"
azrael:~ mosser$ HEADERS='Content-Type: application/json'
azrael:~ mosser$ curl -i -w "\n" -H "$HEADERS" \
                     -X POST -d “$REQUEST" \
                     $BASE_URL/mailbox
HTTP/1.1 200
Content-Type: application/json; charset=utf-8
Server: Mono-HTTPAPI/1.0
Date: Thu, 25 Feb 2016 09:24:41 GMT
Content-Length: 1
Keep-Alive: timeout=15,max=100

1
azrael:~ mosser$
```

Consuming from Java

```java
public class BankAPI {

  private String url;

  public BankAPI(String host, String port) {
    this.url = "http://" + host + ":" + port;
  }

  public BankAPI() { this("localhost", "9090"); }

  public boolean performPayment(Customer customer, double value) throws ExternalPartnerException {
    // Building payment request
    JSONObject request = new JSONObject().put("CreditCard", customer.getCreditCard())
                                         .put("Amount", value);
    // Sending a Payment request to the mailbox
    Integer id;
    try {
      String str = WebClient.create(url).path("/mailbox")
                            .accept(MediaType.APPLICATION_JSON_TYPE)
                            .header("Content-Type", MediaType.APPLICATION_JSON)
                            .post(request.toString(), String.class);
      id = Integer.parseInt(str);
    } catch (Exception e) {
      throw new ExternalPartnerException(url+"/mailbox", e);
    }

    // Retrieving the payment status
    JSONObject payment;
    try {
      String response = WebClient.create(url).path("/payments/" + id).get(String.class);
      payment = new JSONObject(response);
    } catch (Exception e) {
      throw new ExternalPartnerException(url + "payments/" + id, e);
    }
    // Assessing the payment status
    return (payment.getInt("Status") == 0);
  }
}
```

The All Together

```
WebClient.create(url).path("/mailbox")
   .accept(MediaType.APPLICATION_JSON_TYPE)
   .header("Content-Type", MediaType.APPLICATION_JSON)
   .post(request.toString(), String.class);
```

transport

marshalling

```
JSONObject request =
  new JSONObject()
    .put("CreditCard",customer.getCreditCard())
    .put("Amount", value);
```

unmarshalling

```
[WebInvoke( Method = "POST", UriTemplate = "mailbox",
            RequestFormat = WebMessageFormat.Json,
            ResponseFormat = WebMessageFormat.Json)]
```

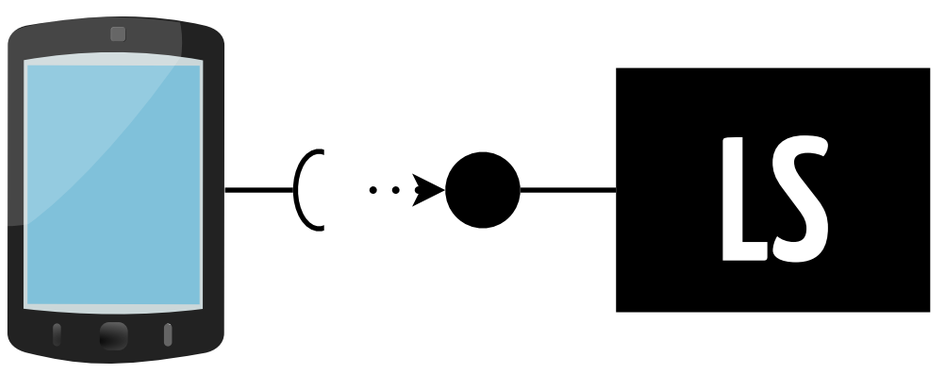
```
WebClient.create(url).path("/mailbox")
  .accept(MediaType.APPLICATION_JSON_TYPE)
  .header("Content-Type", MediaType.APPLICATION_JSON)
  .post(request.toString(), String.class);
```

¿¿ Consistency ??

```
JSONObject request =
  new JSONObject()
    .put("CreditCard",customer.getCreditCard())
    .put("Amount", value);
```

```
[WebInvoke( Method = "POST", UriTemplate = "mailbox",
            RequestFormat = WebMessageFormat.Json,
            ResponseFormat = WebMessageFormat.Json)]
```

Strong Contract



3

Keystone

Contracts are reified into shared artefacts, and used by tools instead of humans

Nothing new...

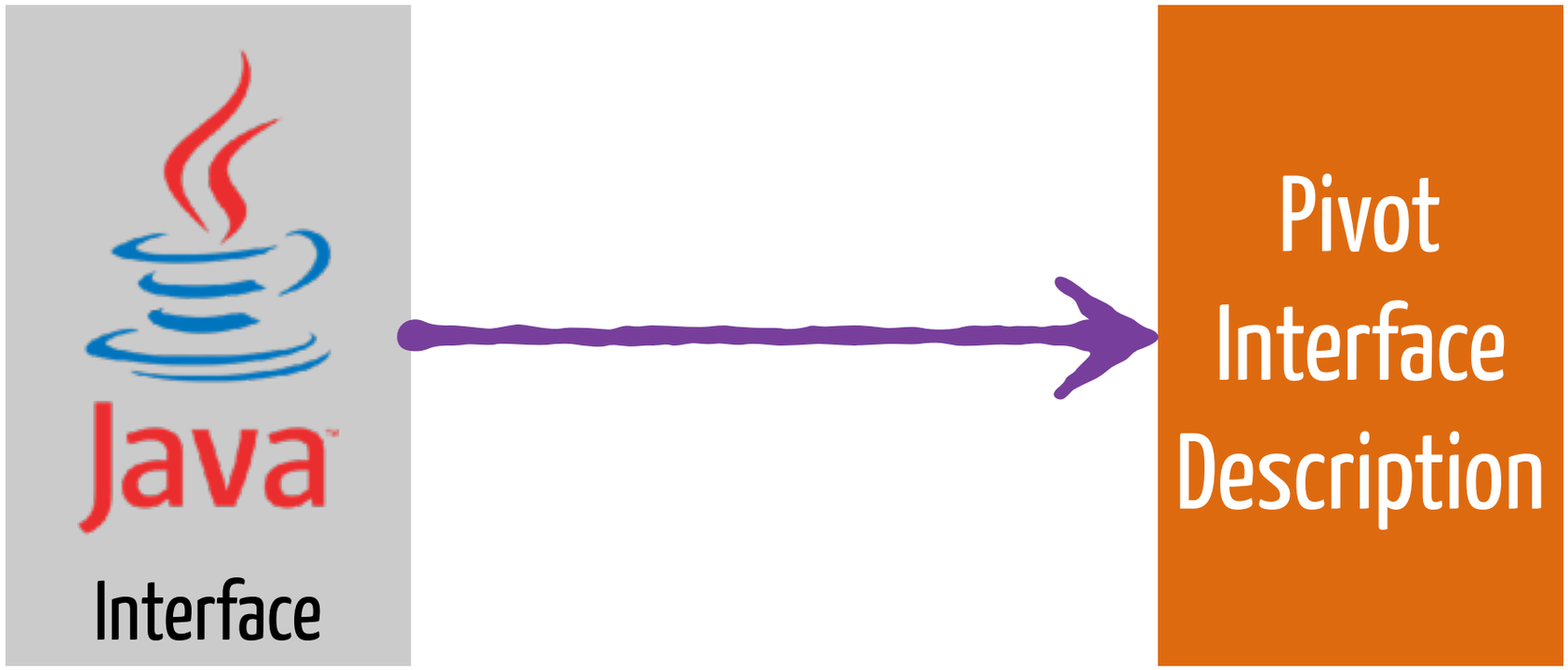
```java
@WebService
public interface CartWebService {

  @WebMethod
  void addItemToCustomerCart(
        @WebParam(name = "customer_name") String customerName,
        @WebParam(name = "item") Item it
      ) throws UnknownCustomerException;

  @WebMethod
  void removeItemToCustomerCart(
        @WebParam(name = "customer_name") String customerName,
        @WebParam(name = "item") Item it
      ) throws UnknownCustomerException;

  @WebMethod
  @WebResult(name = "cart_contents")
  Set<Item> getCustomerCartContents(
            @WebParam(name = "customer_name") String customerName
          ) throws UnknownCustomerException;

  @WebMethod
  @WebResult(name = "order_id")
  String validate(@WebParam(name = "customer_name") String customerName)
        throws PaymentException, UnknownCustomerException,
               EmptyCartException;
}
```



Compilation process: Interface → Pivot

SOAP standard (SoC course)

```
@WebMethod
void addItemToCustomerCart(
        @WebParam(name = "customer_name") String customerName,
        @WebParam(name = "item") Item it
    ) throws UnknownCustomerException;
```

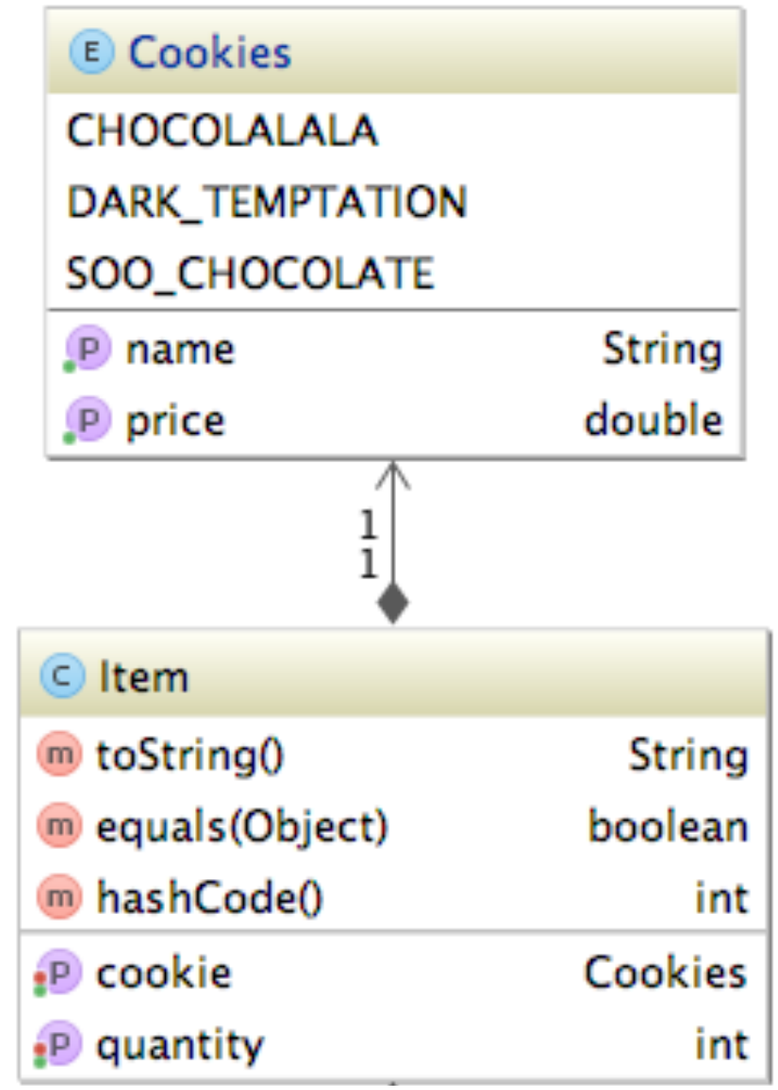
Java2WSDL

```
<wsdl:portType name="CartWebService">
  <wsdl:operation name="addItemToCustomerCart">
    <wsdl:input message="ns1:addItemToCustomerCart"
                name="“addItemToCustomerCart" />
    <wsdl:output message="“ns1:addItemToCustomerCartResponse"
                 name="“addItemToCustomerCartResponse"/>
    <wsdl:fault message="ns1:UnknownCustomerException"
                name="“UnknownCustomerException" />
  </wsdl:operation>
  ...
</wsdl:portType>
```

Web Service Description Language

XSD for data structures

```
<xs:complexType name="item">
  <xs:sequence>
    <xs:element minOccurs="0"
                name="cookie"
                type="tns:cookies"/>
    <xs:element name="quantity"
                type="xs:int"/>
  </xs:sequence>
</xs:complexType>
```




```
<xs:simpleType name="cookies">
  <xs:restriction base="xs:string">
    <xs:enumeration value="CHOCOLALALA"/>
    <xs:enumeration value="DARK_TEMPTATION"/>
    <xs:enumeration value="SOO_CHOCOLATE"/>
  </xs:restriction>
</xs:simpleType>
```

e.g., Business objects, Messages

Standard ⇒ No freedom

Standard ⇒ Automation

Why should we **write piece of codes** instead of **being lazy** and write **pieces of code that will write pieces of code** on our behalf

?

- Jean-Marc Jézéquel

Automated Generation & Exposition

localhost:8080/tcf-backend/w

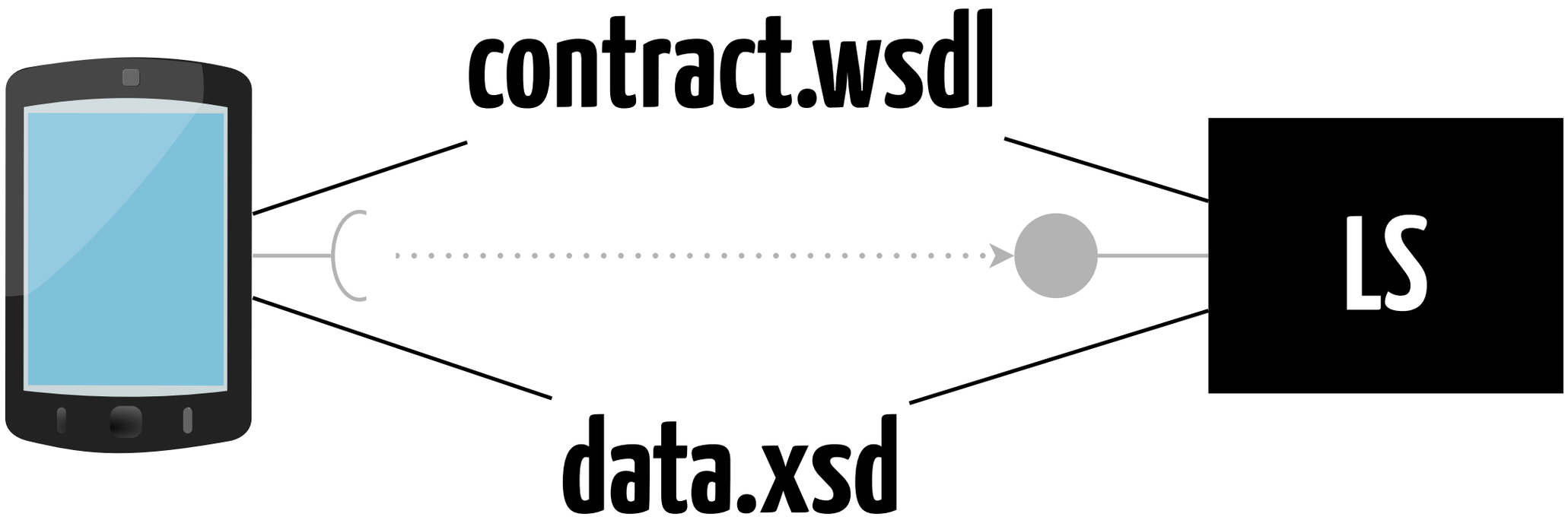
Sébastien

localhost:8080/tcf-backend/webservices/CartWS?wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.

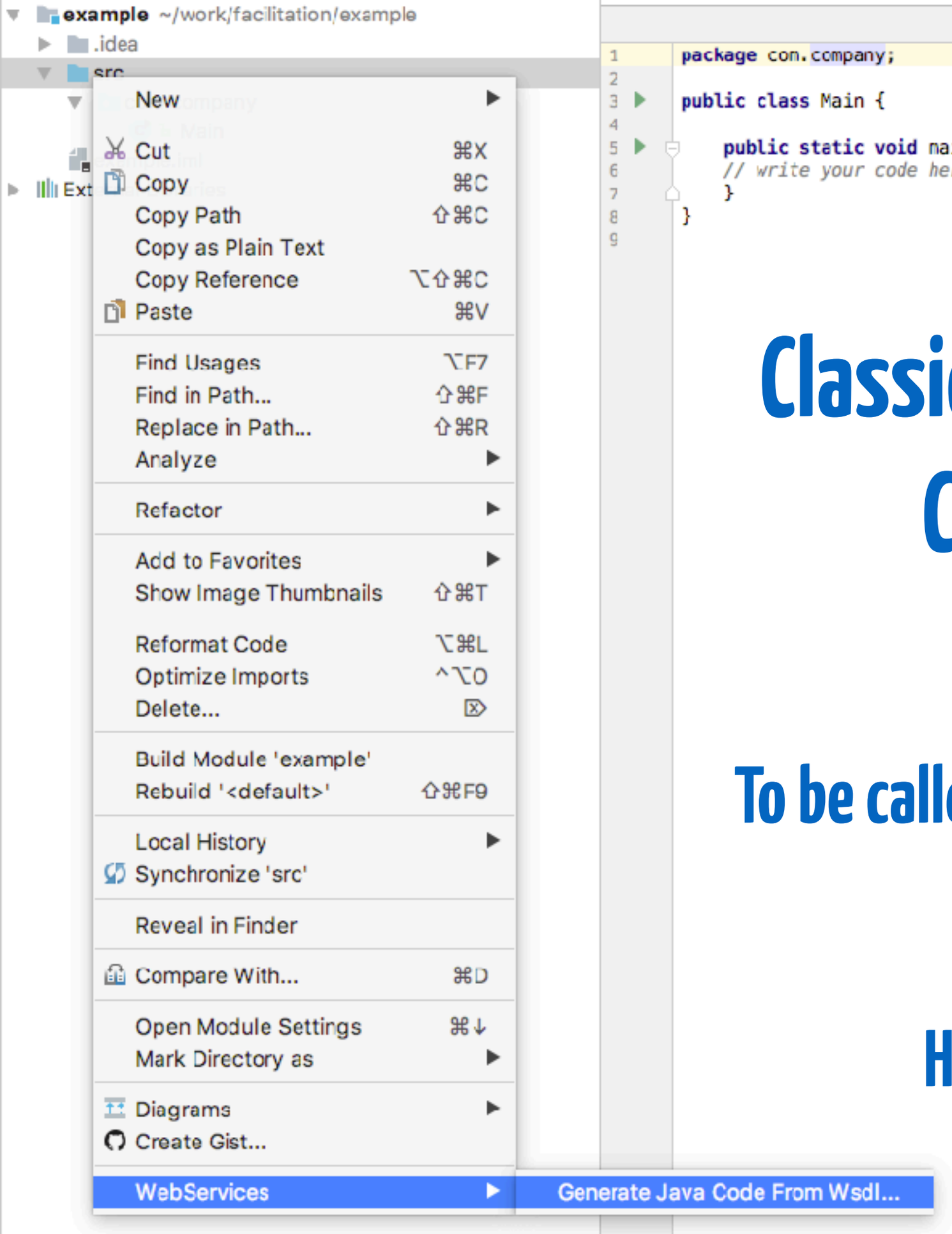
```
▼<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:tns="http://www.polytech.unice.fr/si/4a/isa/tcf/cart"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:ns2="http://schemas.xmlsoap.org/soap/http"
 xmlns:ns1="http://webservice.tcf.isa.polytech.unice.fr/"
 name="CartWebServiceImplService"
 targetNamespace="http://www.polytech.unice.fr/si/4a/isa/tcf/cart">
  <wsdl:import location="http://localhost:8080/tcf-backend/webservices/CartWS?
  wsdl=CartWebService.wsdl"
  namespace="http://webservice.tcf.isa.polytech.unice.fr/"></wsdl:import>
 ▼<wsdl:binding name="CartWebServiceImplServiceSoapBinding"
  type="ns1:CartWebService">
    <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
   ▼<wsdl:operation name="getCustomerCartContents">
      <soap:operation soapAction="" style="document"/>
    ▼<wsdl:input name="getCustomerCartContents">
       <soap:body use="literal"/>
```

Java2WSDL



WSDL2Xxx

- Data structures using the Xxx language
- (Un)Marshalling using the Xxx language



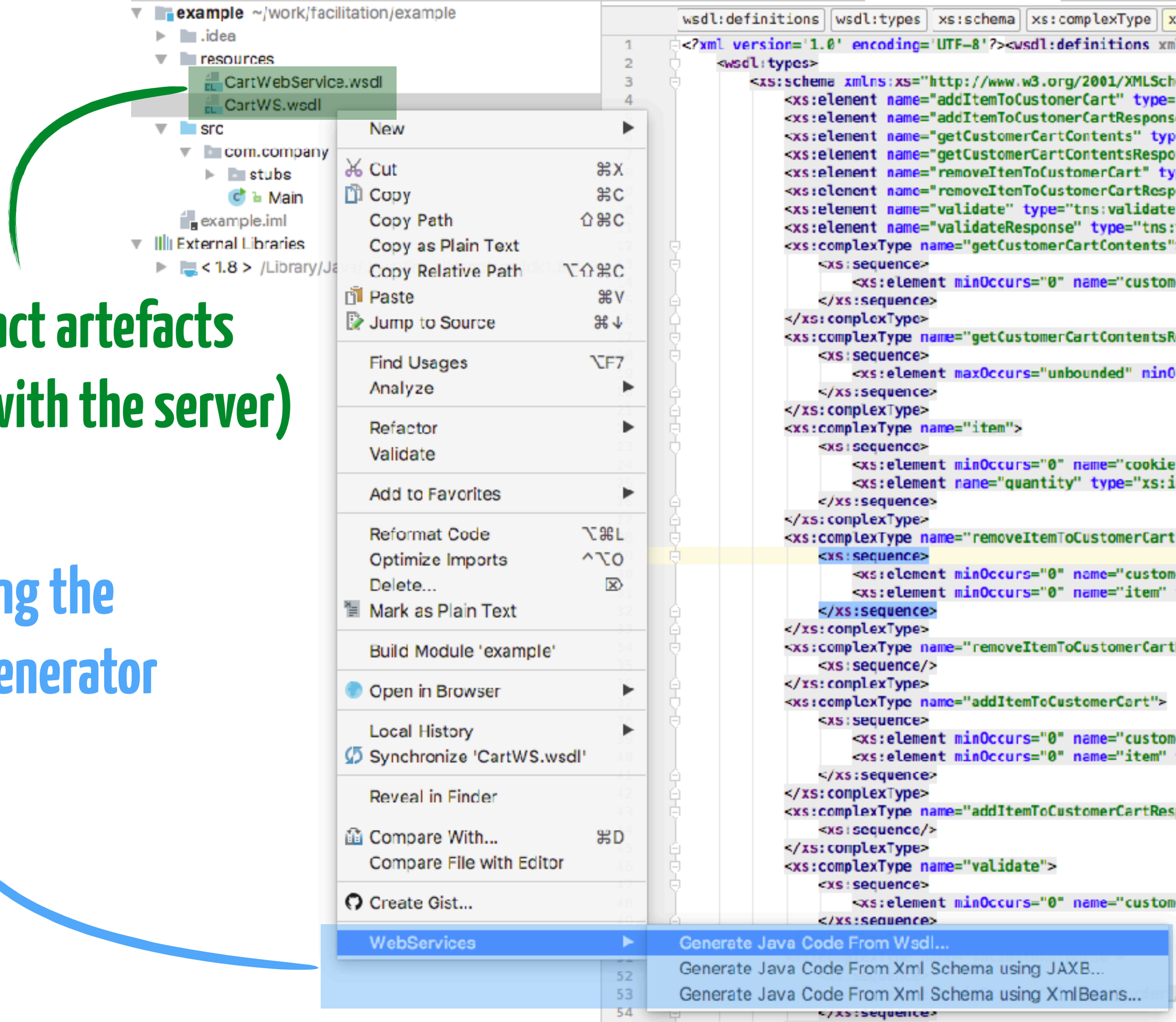
Classical (as in Standard) Code generator

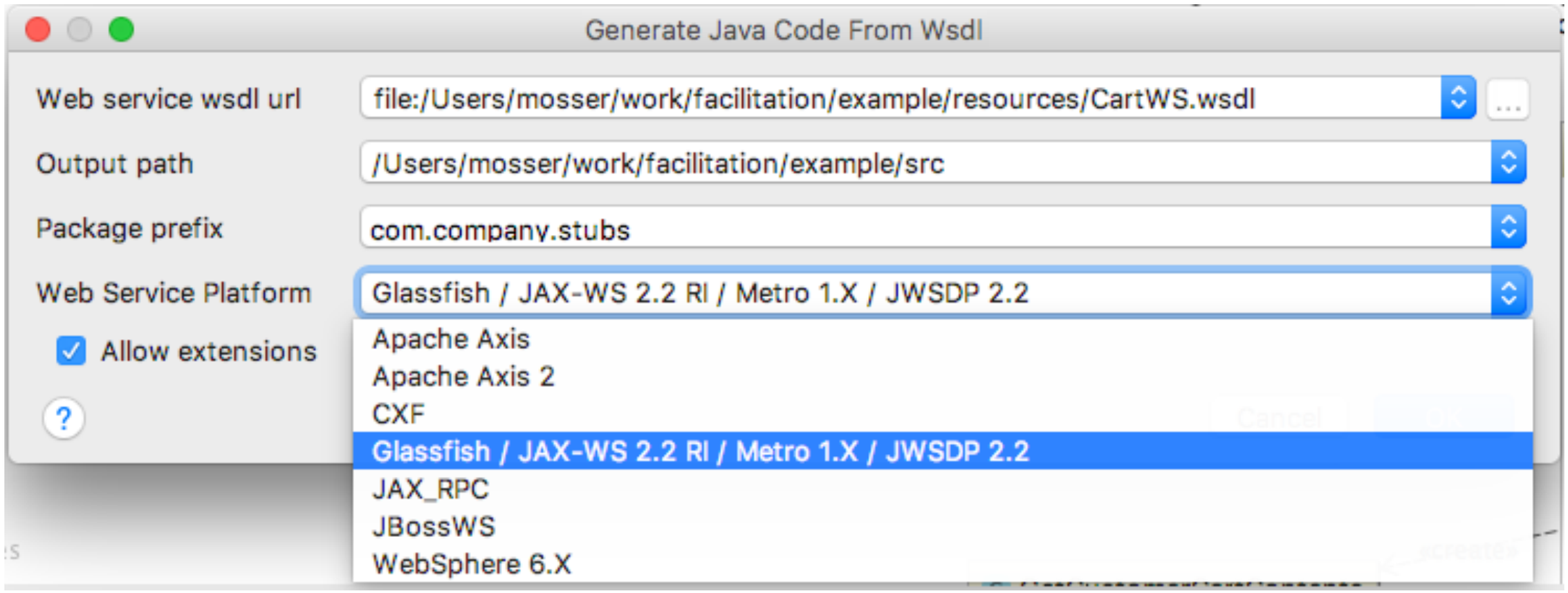
To be called on client side (obviously)

Here IntelliJ Ultimate

Contract artefacts (shared with the server)

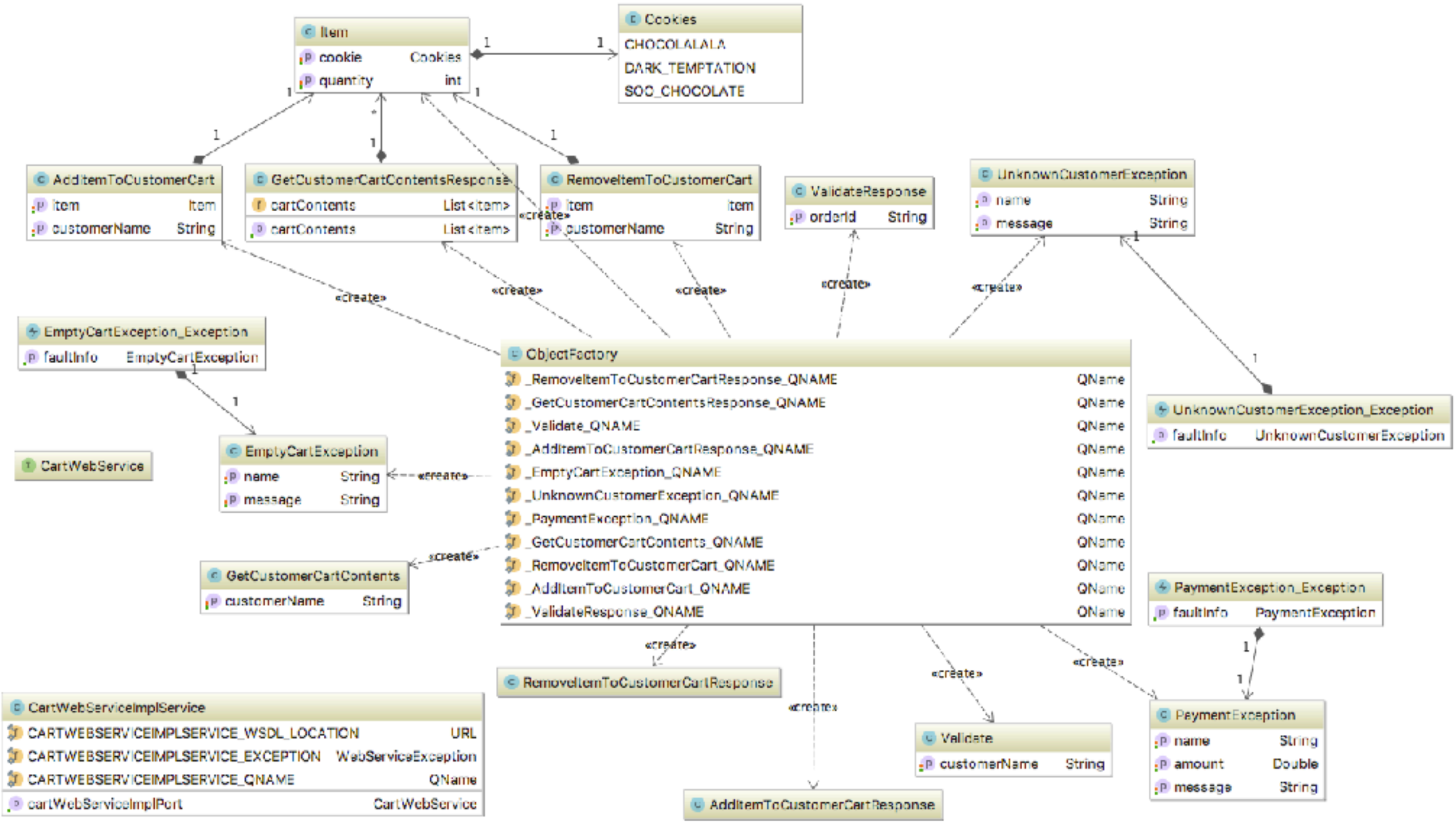
Calling the code generator





Standard ⇏ single implementation

Generated Code!



Consuming a service == \o/ sending messages to objects \o/

```
public static void main(String[] args) throws Exception {
  System.out.println("#### Instantiating the WS Proxy");
  CartWebServiceImplService factory = new CartWebServiceImplService();
  CartWebService ws = factory.getCartWebServiceImplPort();

  List<Item> cart = ws.getCustomerCartContents("john");
  System.out.println("Cart is empty: " + cart.isEmpty());

  Item i = new Item();
  i.setCookie(Cookies.CHOCOLALALA); i.setQuantity(3);
  ws.addItemToCustomerCart("john", i);
  i.setCookie(Cookies.DARK_TEMPTATION); i.setQuantity(2);
  ws.addItemToCustomerCart("john", i);
  i.setCookie(Cookies.CHOCOLALALA); i.setQuantity(4);
  ws.addItemToCustomerCart("john", i);

  cart = ws.getCustomerCartContents("john");
  System.out.println("John's cart: " +cart);
}
```

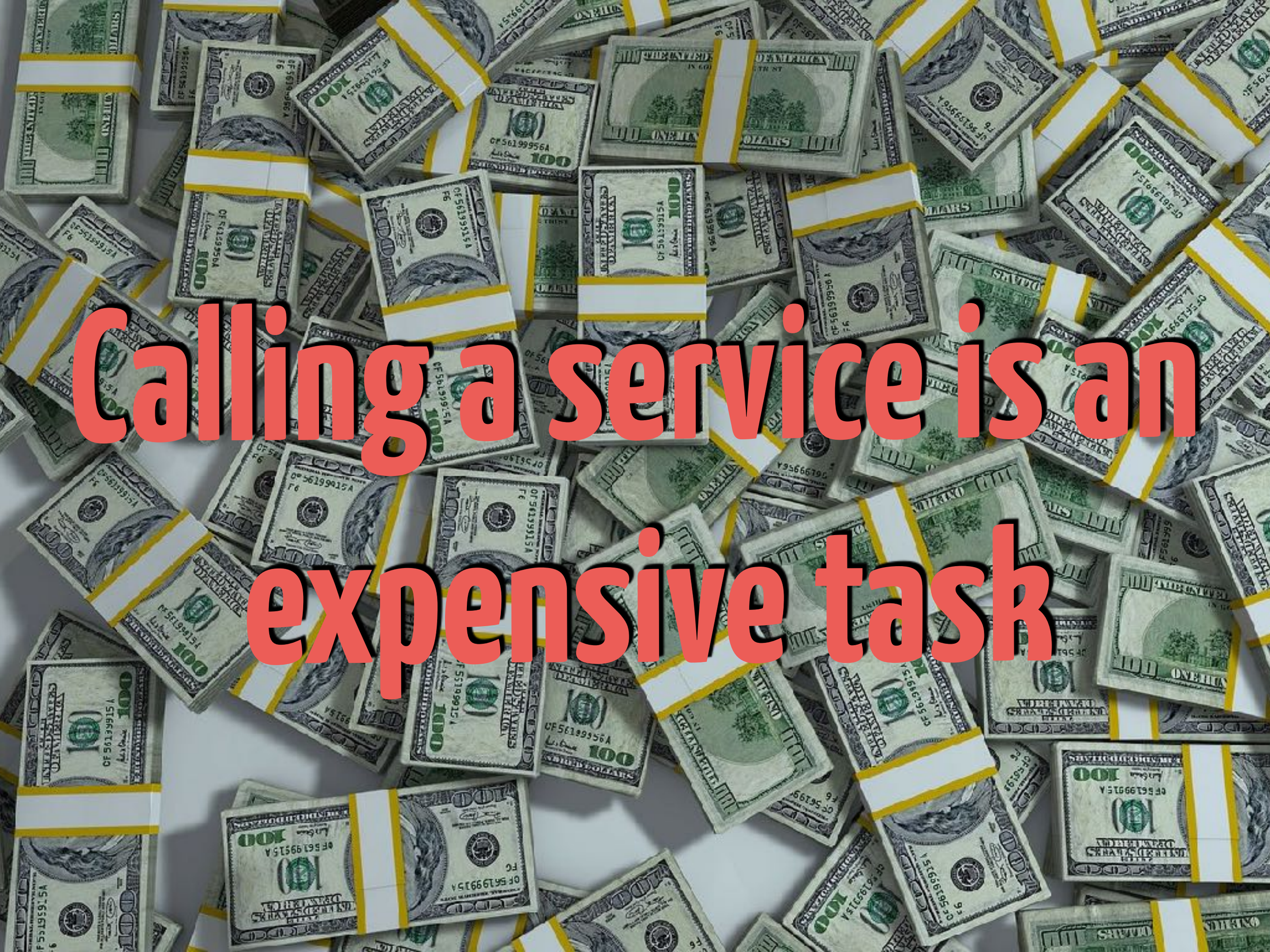
/!\ Cost /!\

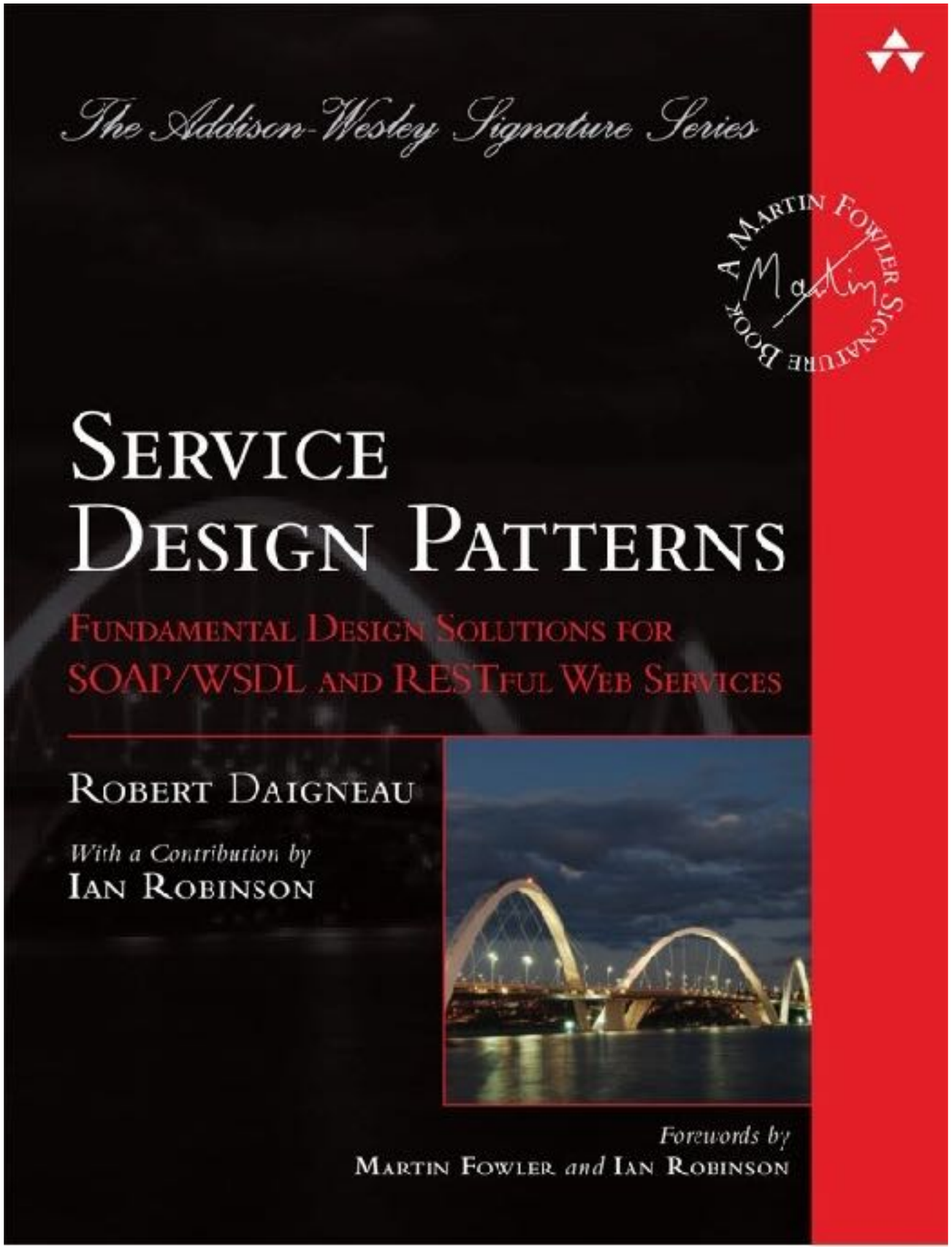
Step Back

4

Public APIs support flexibility

Public APIs support interoperability


Calling a service is an
expensive task



Bibliography

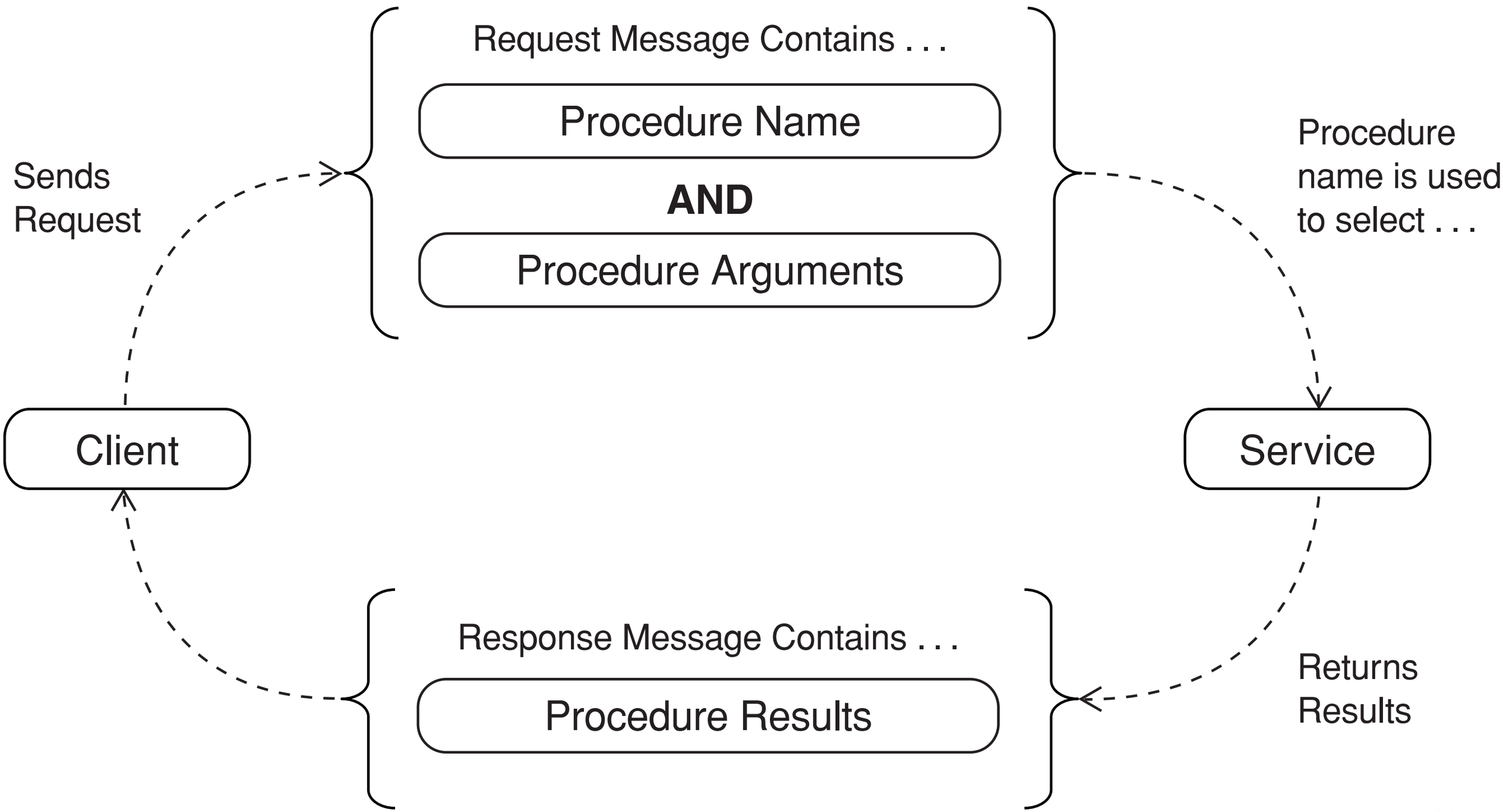
We only talked about contracts

One can also talk about “style”

Exposing Resources (Nouns) ?

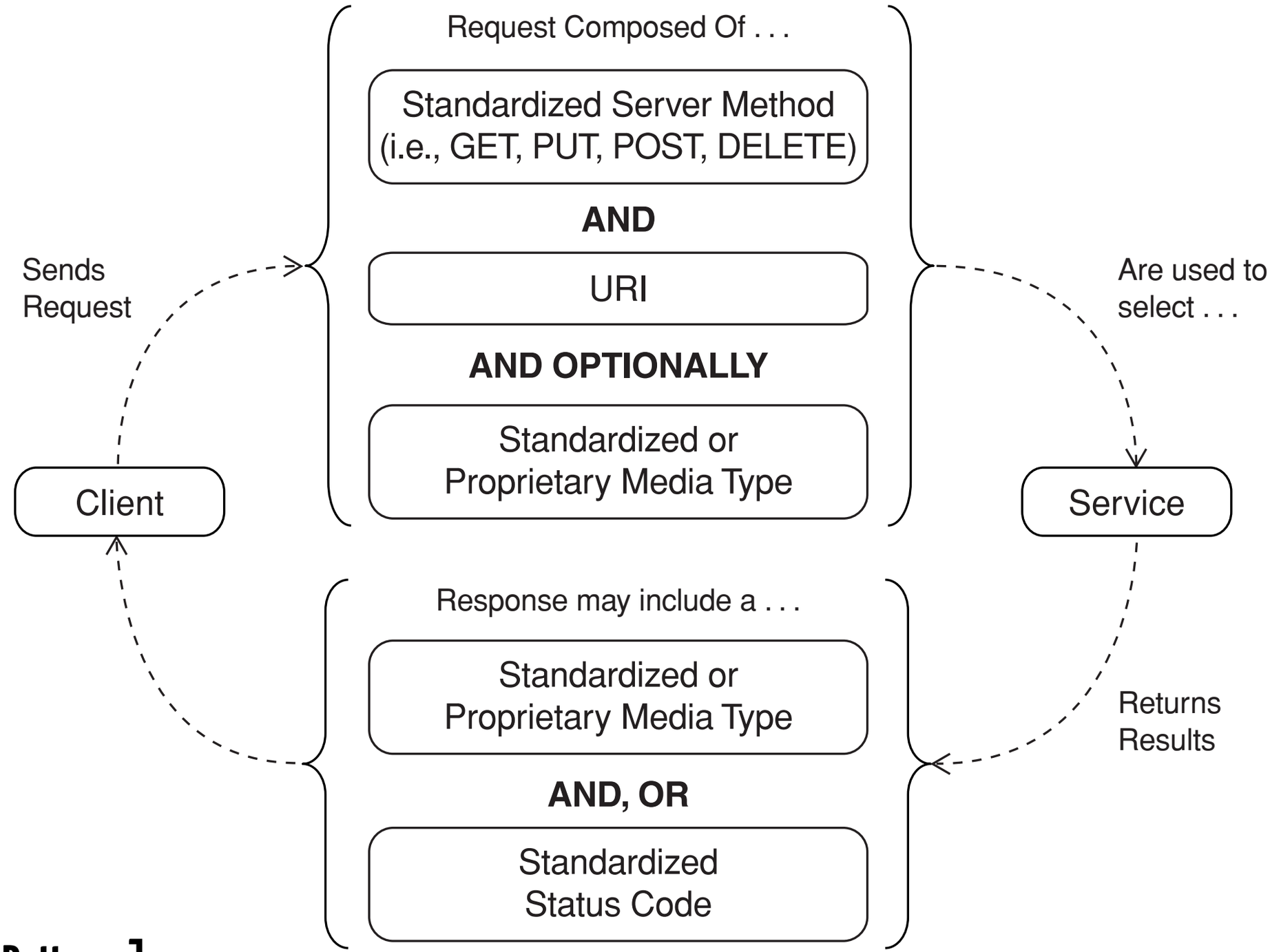
Exposing Operations (Verbs) ?

RPC Interaction Protocol

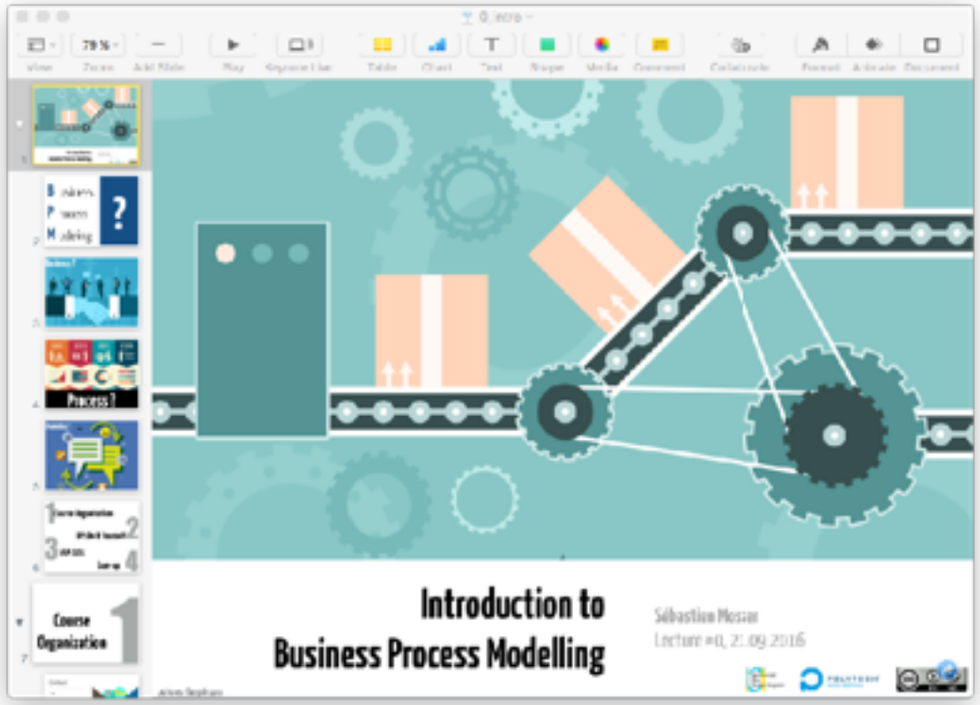


[Service Design Patterns]

Resource Interaction Protocol



Blatant advertisement



Micro-services
& Software Architecture

5th year courses