Distributed computing systems

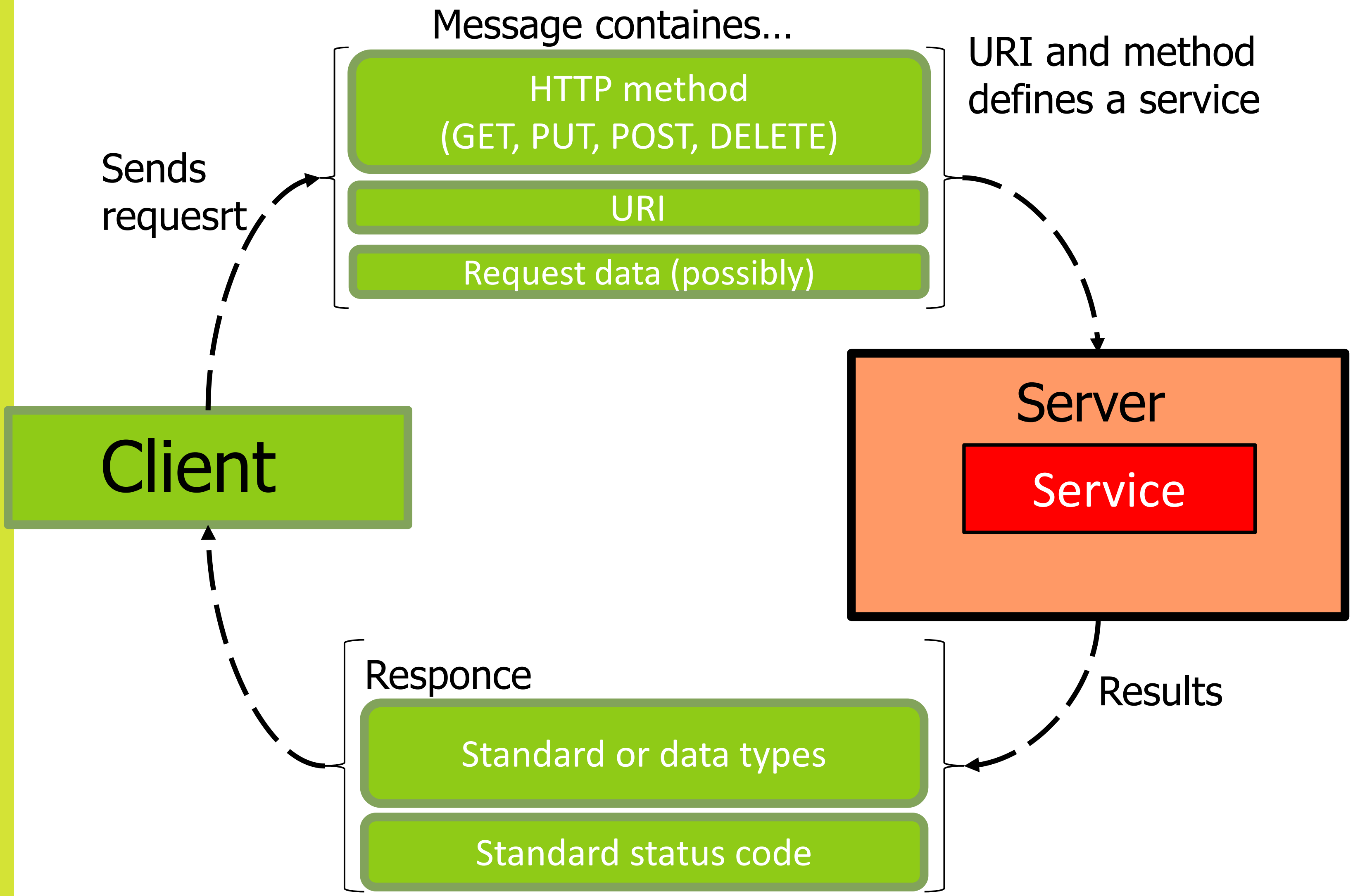
REST

REST Web services

Resource API

- If the foundation of service - is the use and modification of resources located on a remote system, the use of RPC can lead to swelling of the interface (NewPerson, EditPerson, DeletePerson, GetPerson et al.)
- The solution - you must use the standard verbs HTTP (GET, PUT, POST, DELETE) to work with the resources of remote systems.
  Each procedure, the essence of the subject area is assigned to the file URI.
- The client should use the standard HTTP verbs with the corresponding the URI, and the server to perform these commands, and use standard HTTP responses where possible.

Rest API


Message containes…
HTTP method
(GET, PUT, POST, DELETE)
URI
Request data (possibly)
URI and method
defines a service
Sends
requesrt
Client
Server
Service
Responce
Standard or data types
Standard status code
Results

REST

- **Representational State Transfer (REST)** is an architectural style that abstracts the architectural elements within a distributed hypermedia system.
- Was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation

REST

- Many web services use messages to form their own domain-specific API. These messages incorporate common logical commands. CRUD:
  - Create Read Update Delete
- However, can lead to a proliferation of messages, even in relatively small problem domains .
- REST provide a possibility to manipulate data managed by a remote system, but avoid direct coupling to remote procedures, and minimize the need for domain-specific APIs.
- HTTP makes it relatively easy for clients to reuse logic found in remote procedures while insulating them from underlying technologies. Rather than creating a domain-specific API, one could leverage the standards defined in the HTTP specification.

Daigneau, Robert (2011-10-25). Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. Pearson Education. Kindle Edition.

Everything is a resource

- Assign all procedures, instances of domain data, and files a URI.
- A resource may be a text file, a media file (e.g., images, videos, audio), a specific row in a database table, a collection of related data (e.g., products), a logical transaction, a queue, a downloadable program, a business process (i.e., procedure)— almost anything.
  - `http://music.site/users/max`
  - `http://music.site/albums/8`
- A collection of resources - also a resource
  - `http://music.site/users`

Daigneau, Robert (2011-10-25). Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. Pearson Education. Kindle Edition.

REST FUNDAMENTALS

- **Statelessness**
  - State of the client is stored only on client
  - All information that server needs to process the request should be in the request (self-descriptive messages)
- **Cached architecture**
  - The server response can be cached and reused with no new appeals
- **Client-server separation (loose coupling)**
  - The client knows everything about server interface, but knows nothing about the server implementation.

Resource API in REST

- PUT is used to **create** or **update** resources.
- GET is used to **retrieve** a resource representation.
- DELETE **removes** a resource.
- POST : used to **create a subordinate** of the target resource.

GET	PUT	DELETE	POST
=	=	=	=
READ	UPDATE	DELETE	CREATE

Standardized item interface

`http://example.com/resources/item17`

GET	**Get (Retrieve)** the state of the item
PUT	**Replace** this item with another item. If such item is not exist, than create such item
POST	Usually not used
DELETE	**Delete** an item

Standardized collection interface

`http://example.com/resources`

GET	**List** URIs and another information about items in the collection
PUT	**Replace** this collection by another collection
POST	**Create** a new item in the collection
DELETE	**Delete** a collection

Standard REST actions

Correct REST interface

`POST /albums` **– add new album**

`GET /albums/2` **– get info about album 2**

`PUT /albums/2` **– update album 2**

`DELETE /albums/2` **– delete album 2**

Not Correct REST actions

Not correct REST interface

```
POST /albums/create

GET /albums/show/2

POST /albums/update/2

GET /delete/albums/2

DELETE /albums/3/remove
```

HTTP Server responses

- REST allow to use standardized media types and status codes.
- Server responses are HTTP-codes indicating the status of the operation
  - 200 – OK (“Here is your item”)
  - 201 – Created (“You added an item successfully”)

  - 400 – Bad request (“You provided a bad request”)
  - 403 – Forbidden (“You are not allowed to do this”)
  - 404 – Not found (“There is no such item”)
  - 500 – Server error

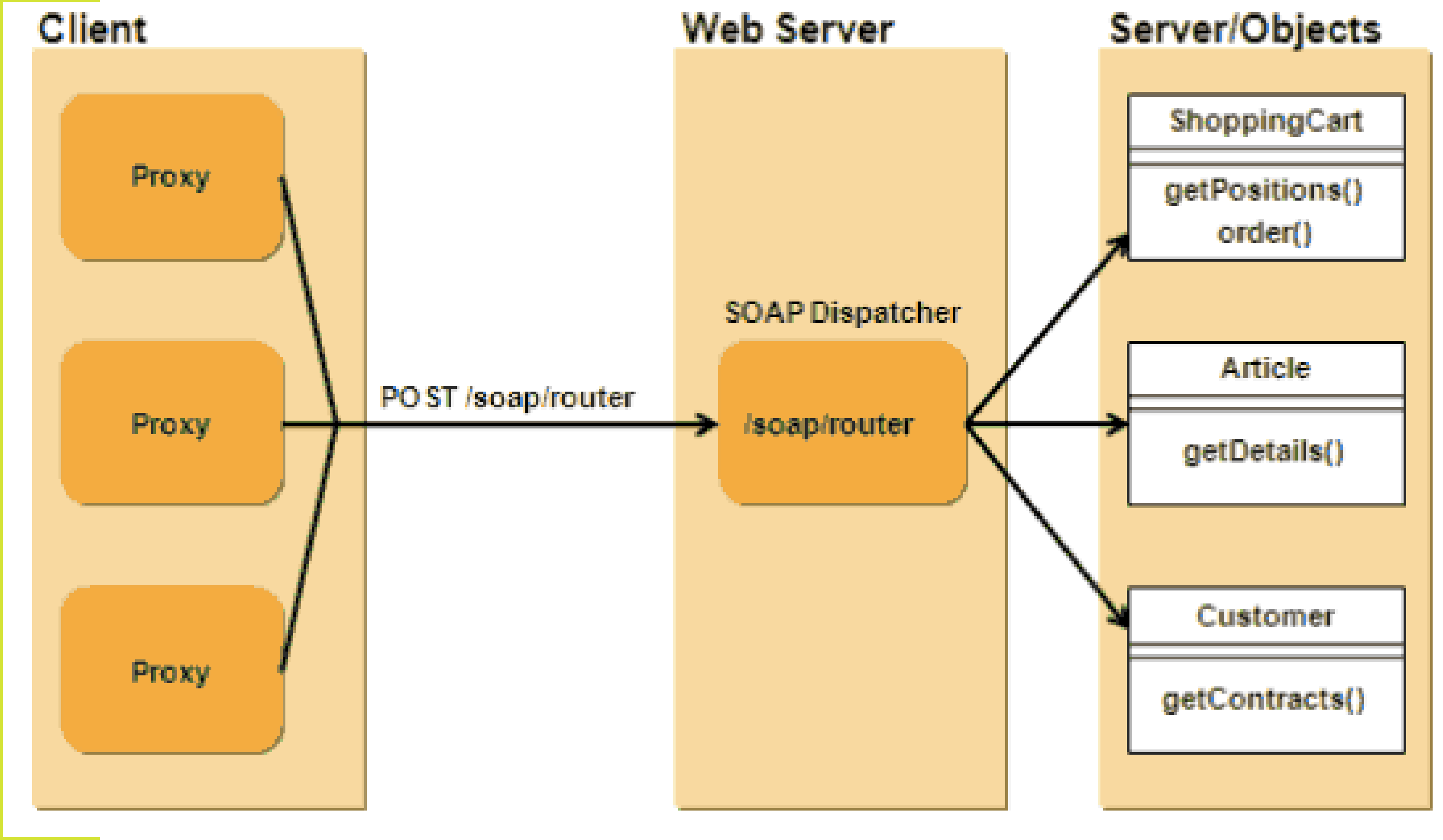
REST vs SOAP

REST	SOAP Web Services
Architectural style	A family of standard protocols
XML, JSON, HTML, JPG, MP3 …	XML.
HTTP – is a basis of all	HTTP – a transport layer
Resource – is a key concept	**Operation** – is a key concept

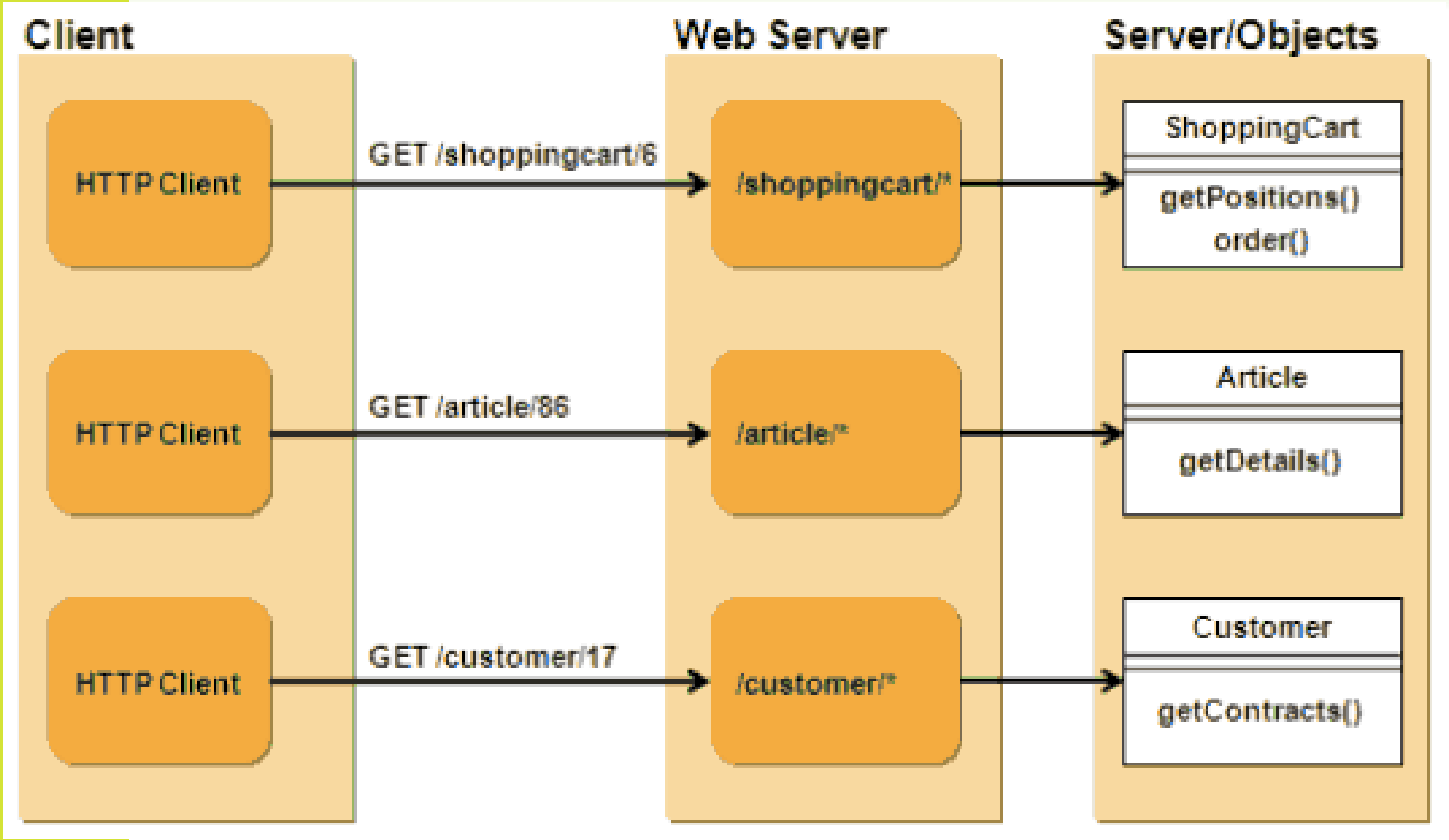
REST vs SOAP

- SOAP-services have a description (WSDL), which allows to generate a client
- SOAP does not allow caching of queries
- SOAP only works with POST-requests
- Application
  - SOAP - business applications, distributed system infrastructure
  - REST - the external interface of the system

SOAP REQUESTS



REST REQUESTS



REST API Examples

Twitter REST API v1.1

`GET statuses/retweets/:id`

Returns up to 100 retweets of an «id» tweet

`GET statuses/show/:id`

Returns a single tweet «id»

`GET statuses/destroy/:id`

Delete an «id» tweet

`GET statuses/update`

Update the status of the user (create a new tweet)

Google Translate REST API

IN:

```
GET https://www.googleapis.com/language/translate/v2?key=INSERT-YOUR-KEY&source=en&target=de&q=Hello%20world
```

OUT :

```
200 OK
{
        "data": {
                "translations": [
                {
                        "translatedText": "Hallo Welt"
                }
            ]
        }
}
```

PayPal REST API

IN: `https://api.paypal.com/v1/payments/payment`

```
curl -v https://api.sandbox.paypal. com/v1/payments/payment \ -H "Content-
Type:application/json" \ -H "Authorization:Bearer EMxItHE7Zl4cMdkvMg-
f7c63GQgYZU8FjyPWKQlpsqQP" \ -d '{ "intent":"sale", "payer":{
"payment_method":"credit_card", "funding_instruments":[{ "credit_card":{
"number":"4417119669820331", "type":"visa", "expire_month":11,
"expire_year":2018, "cvv2":"874", "first_name":"Joe", "last_name":"Shopper",
"billing_address":{"line1":"52 N Main ST", "city":"Johnstown", "country_code":"US",
"postal_code":"43210", "state":"OH" } } } ] },

"transactions":[ { "amount":{ "total":"7.47", "currency":"USD", "details":{
"subtotal":"7.41", "tax":"0.03", "shipping":"0.03" } }, "description":"This is the
payment transaction description.” } ] }'
```

PayPal REST API

OUT:

200 OK

{ "id": "PAY-17S8410768582940NKEE66EQ", "create_time": "2013-01-31T04:12:02Z", "update_time": "2013-01-31T04:12:04Z", "state": "approved", "intent": "sale", "payer": {

…

Developing your own RESTful-service

Server development

- Ruby on Rails
  - Has a reference implementation of the resource model
  - Easy to learn and understand
  - A lot of magic included
- Java – JAX-RS
  - The most popular Web service platform
  - The most popular language
- Python – Django
  - We will try this during our lab.

An example of Java service

```
@Path("/stores")
public class StoreService {

@GET
@Produces("application/xml")
public JAXBElement <Stores> getStoresAsXML()    {
   Stores stores = Stores.getStores();
   return new JAXBElement <Stores>
      ( new Qname("Stores"), Stores.class, stores);
}

@Path("/{id}")
@GET
@Produces("application/xml")
public Store getStoreAsXML(@ PathParam("id") String id) {
   // implementation here
}
```

Daigneau, Robert. Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. Pearson Education. Kindle Edition.

An example of Java service

```
@POST
@Consumes("application/xml")
@Produces("application/xml")
public Store createStore(JAXBElement <Store>
store)    {
   // implementation here
}

@Path("/{id}")
@PUT
@Produces("application/xml")
public Store updateStore(@PathParam("id") String
id)   {
   // implementation here
} }
```

Daigneau, Robert. Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. Pearson Education. Kindle Edition.

An example of Java Client

You can use **Jersey** framework as the reference implementation for REST support in Java. Jersey contains basically a REST server and a REST client. it provides a library to communicate with the server producing REST services.

```
public class Test {
public static void main(String[] args) throws ClientProtocolException,
IOException {
        Client client = Client.create();
        WebResource r = client.resource("http://localhost:8080/xyz");
        MultivaluedMap<String, String> params = new MultivaluedMapImpl();
        params.add("foo", "x");
        params.add("bar", "y");
        // getting XML data: http://localhost:8080/xyz/abc?foo=x&bar=y
        System.out.println(r.path("abc").
queryParams(params).accept(MediaType.APPLICATION_XML).get(String.class));
        // getting JSON data: http://localhost:8080/xyz/abc?foo=x&bar=y
        System.out.println(r.path("abc").
queryParams(params).accept(MediaType.APPLICATION_JSON).get(String.class));
         }
}
```

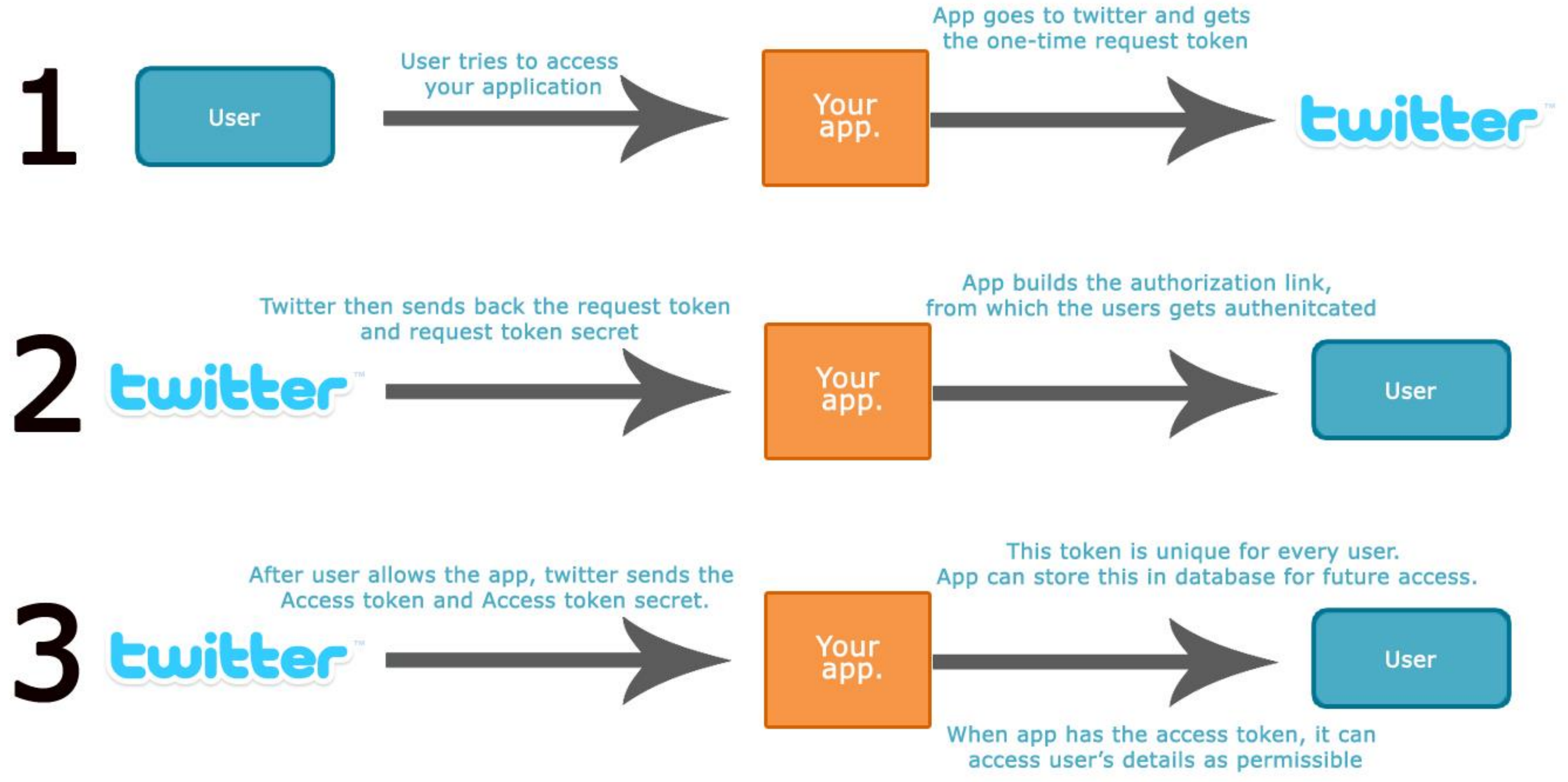
REST Security

- REST-service is usually publicly available
  - Protection is a must!
  - To authenticate using a unique token of the user
  - We can use HTTPS to provide security

Token

- The client receives all its data (and token) during login
- The token uniquely identifies the user
- The token is applied to each authorized message (as a parameter or in the HTTP-header Authorization)

OAuth


1
User
User tries to access your application
Your app.
App goes to twitter and gets the one-time request token
twitter
2
twitter
Twitter then sends back the request token and request token secret
Your app.
App builds the authorization link, from which the users gets authenitcated
User
3
twitter
After user allows the app, twitter sends the Access token and Access token secret.
Your app.
This token is unique for every user. App can store this in database for future access.
User
When app has the access token, it can access user's details as permissible