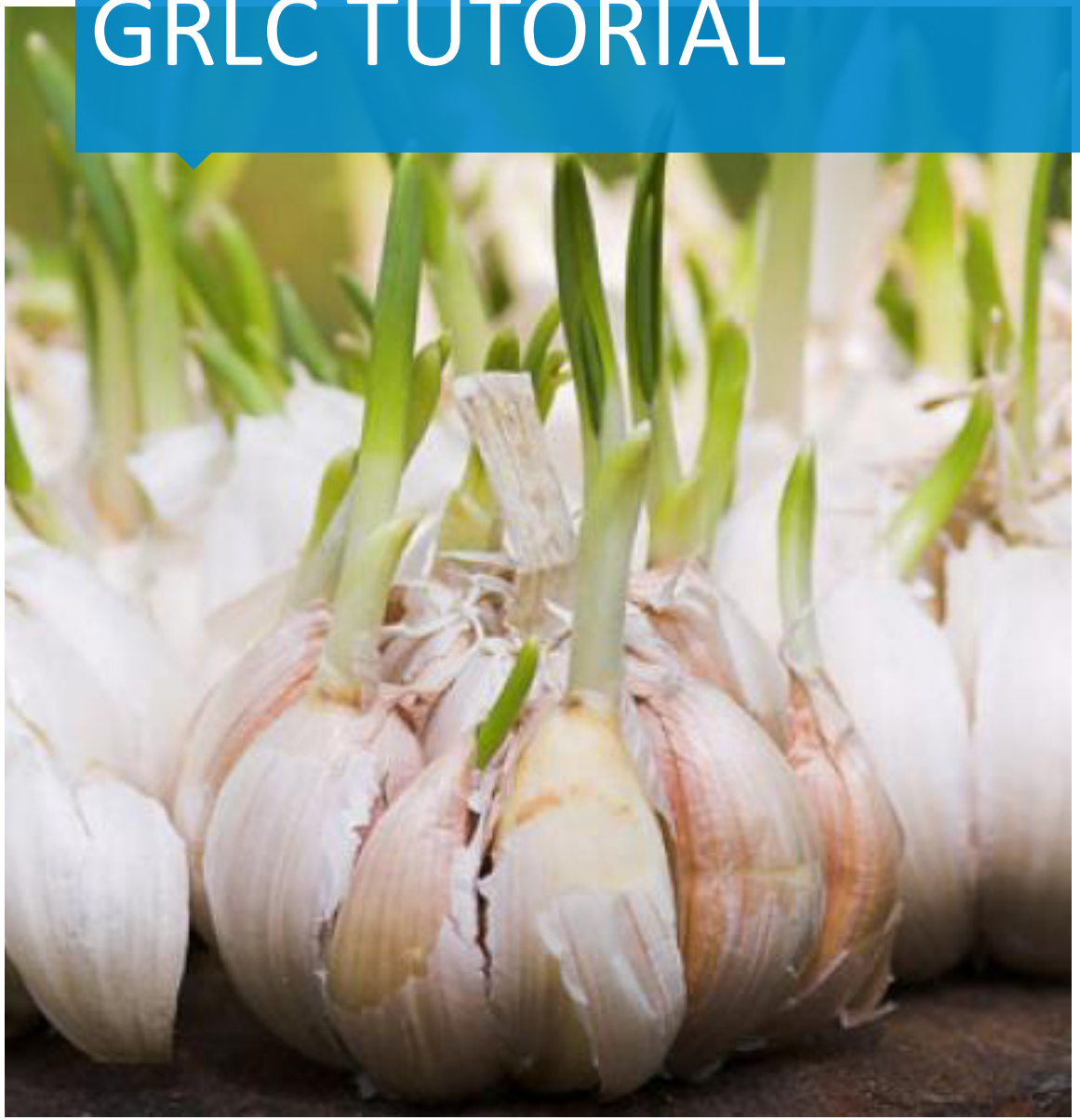
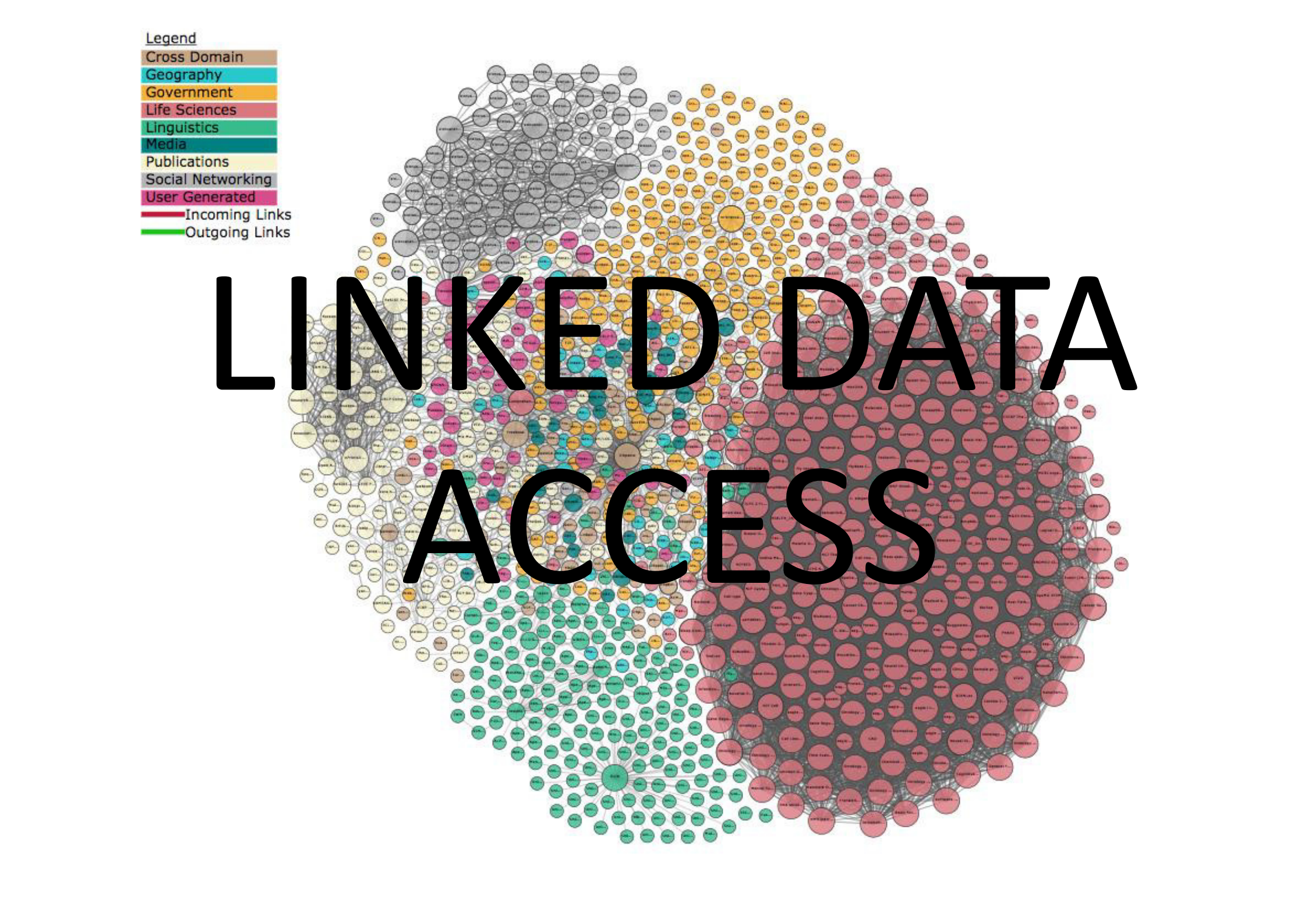
GRLC TUTORIAL



Albert Meroño-Peñuela

CLARIAH WP4 workshop
23/2/2018

VU VRIJE UNIVERSITEIT AMSTERDAM
LOOKING FURTHER


Legend
Cross Domain
Geography
Government
Life Sciences
Linguistics
Media
Publications
Social Networking
User Generated
Incoming Links
Outgoing Links
LINKED DATA
ACCESS

```
from SPARQLWrapper.SPARQLExceptions import QueryBadFormed, EndPointNotFound, EndPointInternalError
import urllib2
from httplib import BadStatusLine
import json
from simplejson import JSONDecodeError
import socket
import time
from timeout import timeout, TimeoutError
from xml.parsers.expat import ExpatError
from pymongo import Connection

connection = Connection('localhost', 27017)
db = connection.lsddimensions

db.dimensions.drop()
db.dsds.drop()

query = """
    PREFIX sdmx: <http://purl.org/linked-data/sdmx#>
    PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
    PREFIX qb: <http://purl.org/linked-data/cube#>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    SELECT DISTINCT ?dimensionu ?dimension ?codeu ?code
    WHERE {
    ?dimensionu a qb:DimensionProperty ;
    rdfs:label ?dimension .
    OPTIONAL {?dimensionu qb:codeList ?codelist .
    ?codelist skos:hasTopConcept ?codeu .
    ?codeu skos:prefLabel ?code . }
    } GROUP BY ?dimensionu ?dimension ?codeu ?code ORDER BY ?dimension
"""
```

LINKED DATA ACCESS

Linked Data is great for information integration on the Web, **but**:

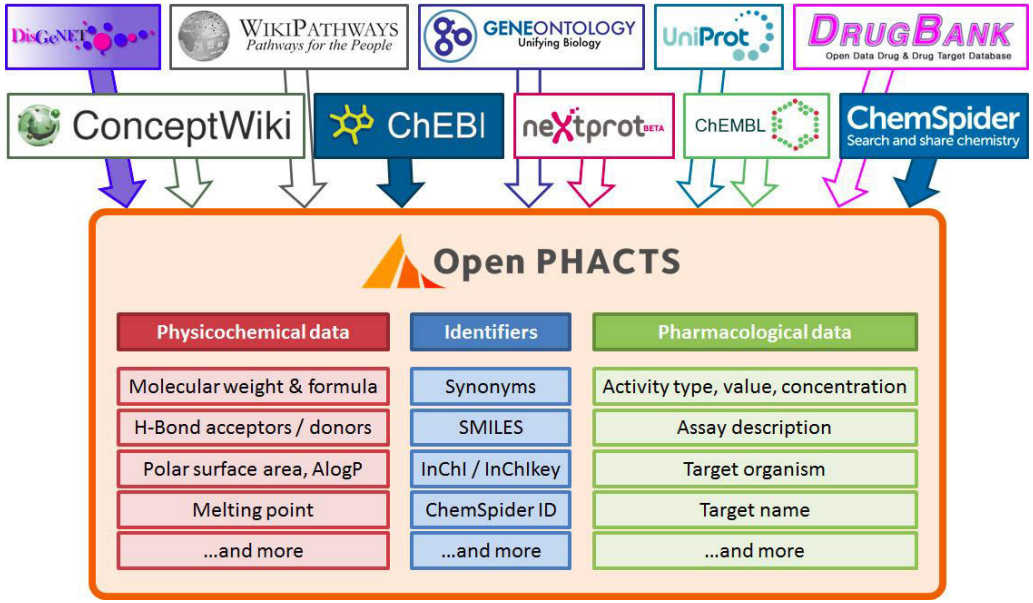
- Heterogeneity of **access methods**: SPARQL, #LD, dumps, HTML/RDFa, LDA
- Hard technological **requirements**: RDF, SPARQL
- **Coupling** with Linked Data specific libararies
- Web developers want APIs and JSON
- Queries are **second-class** Web citizens (i.e. volatile)
  - > Lost after execution (reusability?)
  - > Multiple out-of-sync instances if shared among applications (reliability?)

*How to make semantic queries **repeatable** for Linked Data **consumers** automatically?*

VU

LINKED DATA APIS

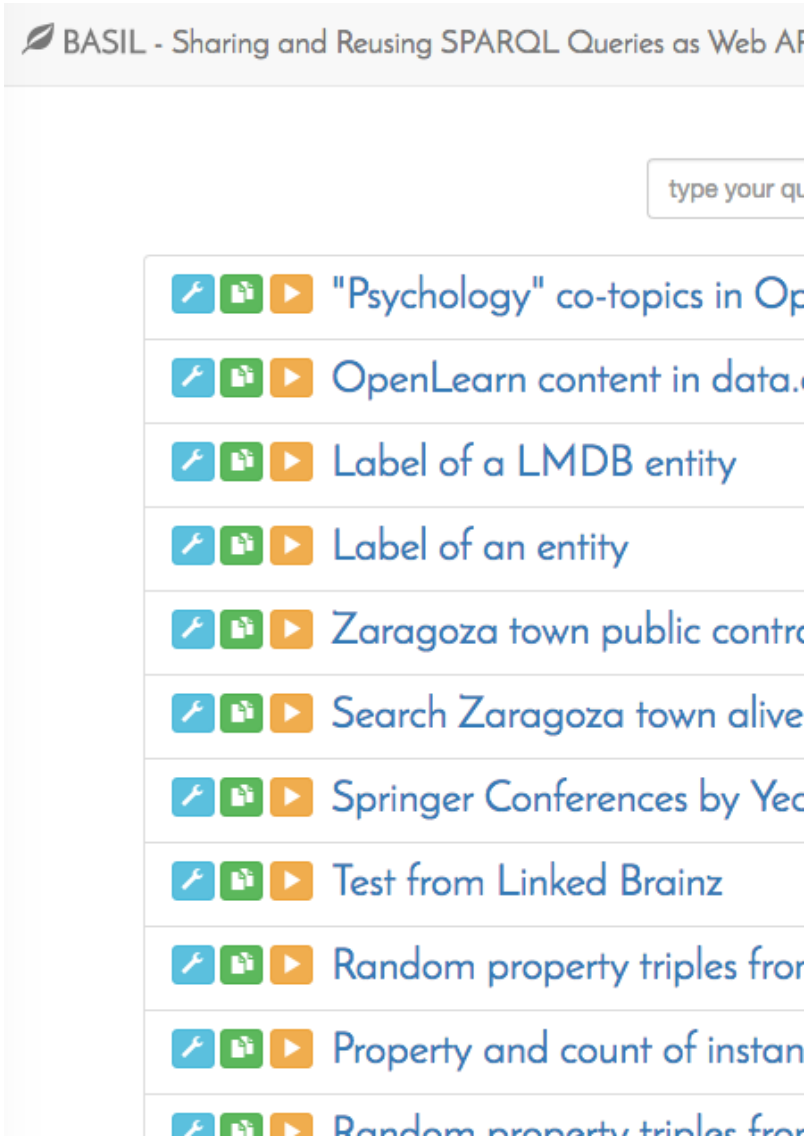
- OpenPHACTS: RESTful entry point to Linked Data hubs for Web applications
- **Query = Service = URI**



However:

- The API (e.g. Swagger spec, code itself) still needs to be coded and maintained
- Exclusion of SPARQL <- query reuse?

BASIL

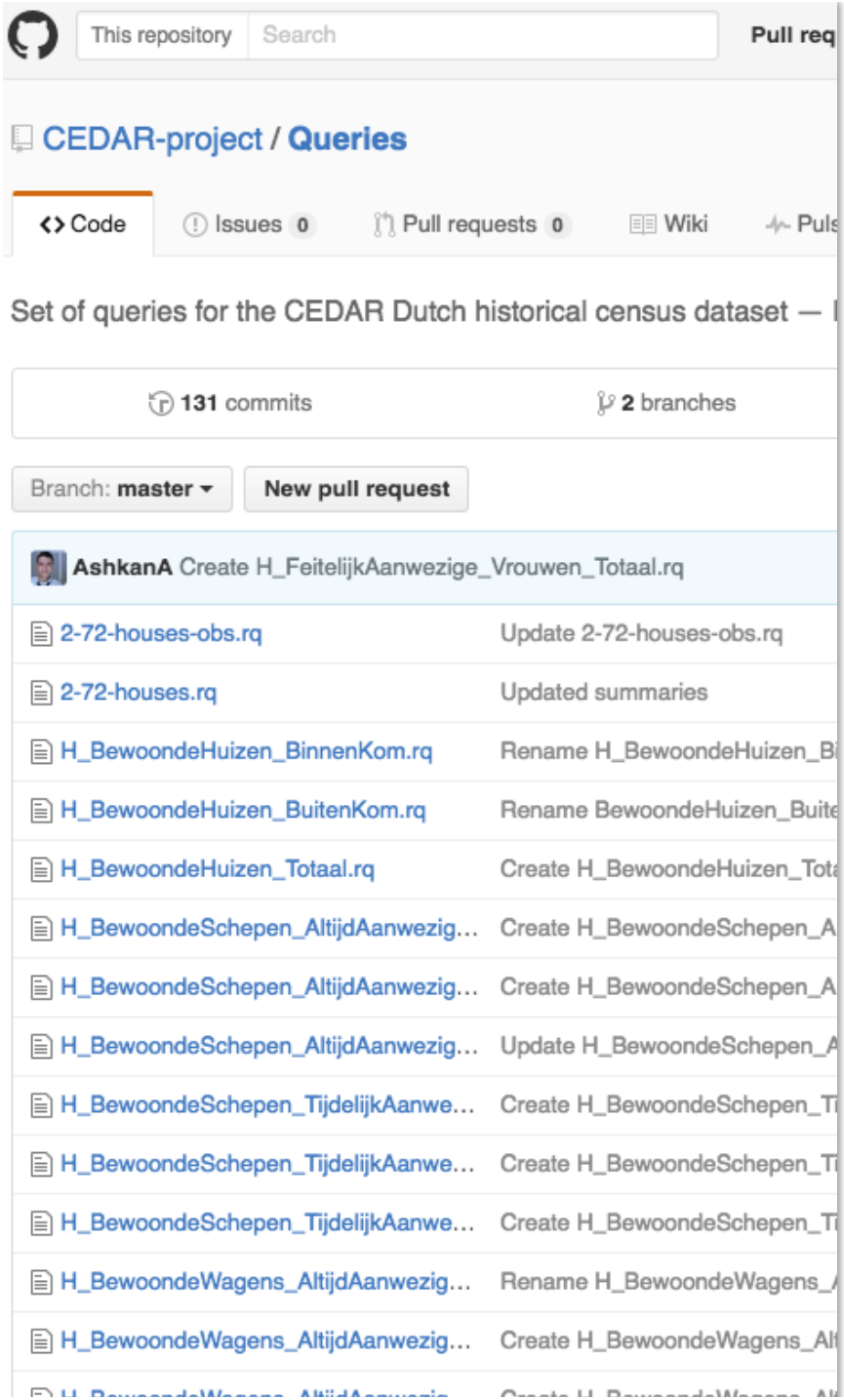


- Automatically builds Swagger specs and API code
- Takes SPARQL queries as input (1 API operation = 1 SPARQL query)
  - > API call functionality limited to SPARQL expressivity
- Makes SPARQL queries uniquely referenceable by using their equivalent LDA operation
  - > Stores SPARQL internally
  - > But we already have uniquely referenceable SPARQL…

VU

Writing SPARQL by trial and error = **versioning** support

Variety of access interfaces needed



GITHUB AS A HUB OF SPARQL QUERIES

- One .rq file for SPARQL query
- Good support of query curation processes
  - > Versioning
  - > Branching
  - > Clone-pull-push
- Web-friendly features!
  - > One URI per query
  - > Uniquely identifiable
  - > De-referenceable (*raw.githubusercontent.com*)

VU

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbr: <http://dbpedia.org/resource/>

SELECT * WHERE {
  GRAPH <urn:uuid:286ba3b4-2556-4321-9d9f-3da38426dae8> {
    ?s ?p ?o
  }
}
LIMIT 10
```

CLARIAH Common Lab Research Infrastructure for the Arts and Humanities

VU VRIJE UNIVERSITEIT AMSTERDAM

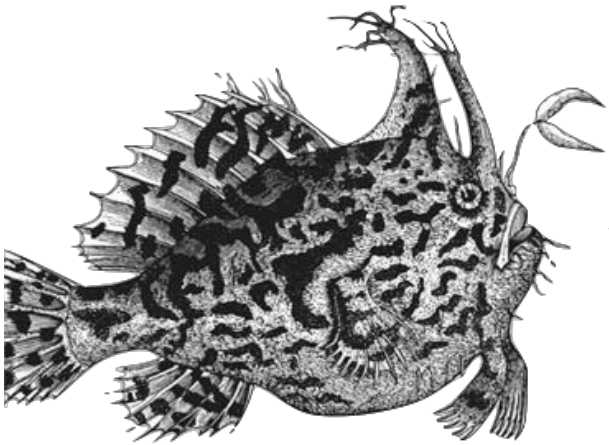
grlc

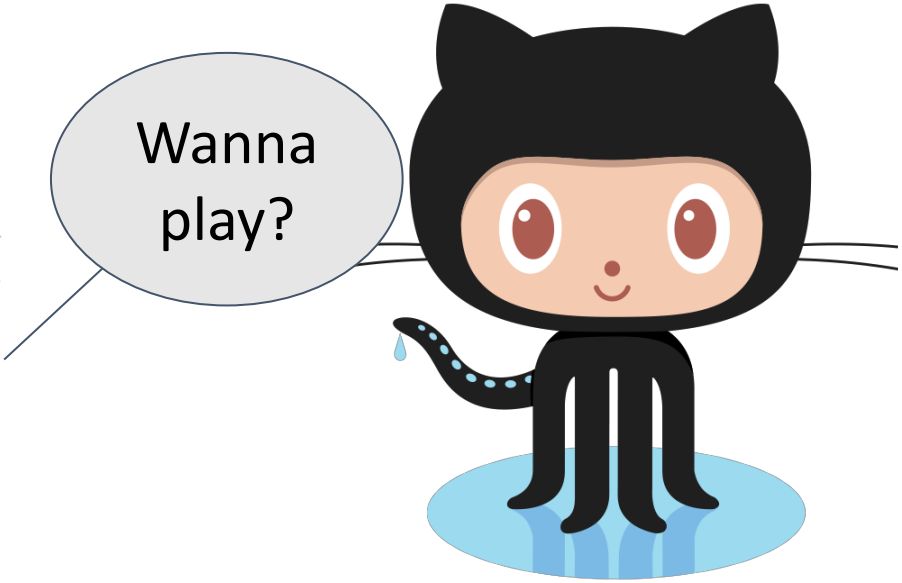
Transparent SPARQL and Linked Data API Curation via GitHub

RESTful API

http://grlc.io

https://github.com/CLARIAH/grlc





README.md

grlc, the git repository linked data API constructor, autor
repositories.

Author: Albert Meroño
Copyright: Albert Meroño, VU University Amsterdam
License: MIT License (see license.txt)

Install and run

```
git clone --recursive https://github.com/CLARIAH/grl
cd grlc
virtualenv .
source bin/activate
pip install -r requirements.txt
python grlc.py
```

Direct your browser to http://localhost:8088.

Alternatively, you can use the provided Gunicorn configu

Usage

grlc assumes a GitHub repository (support for general gi

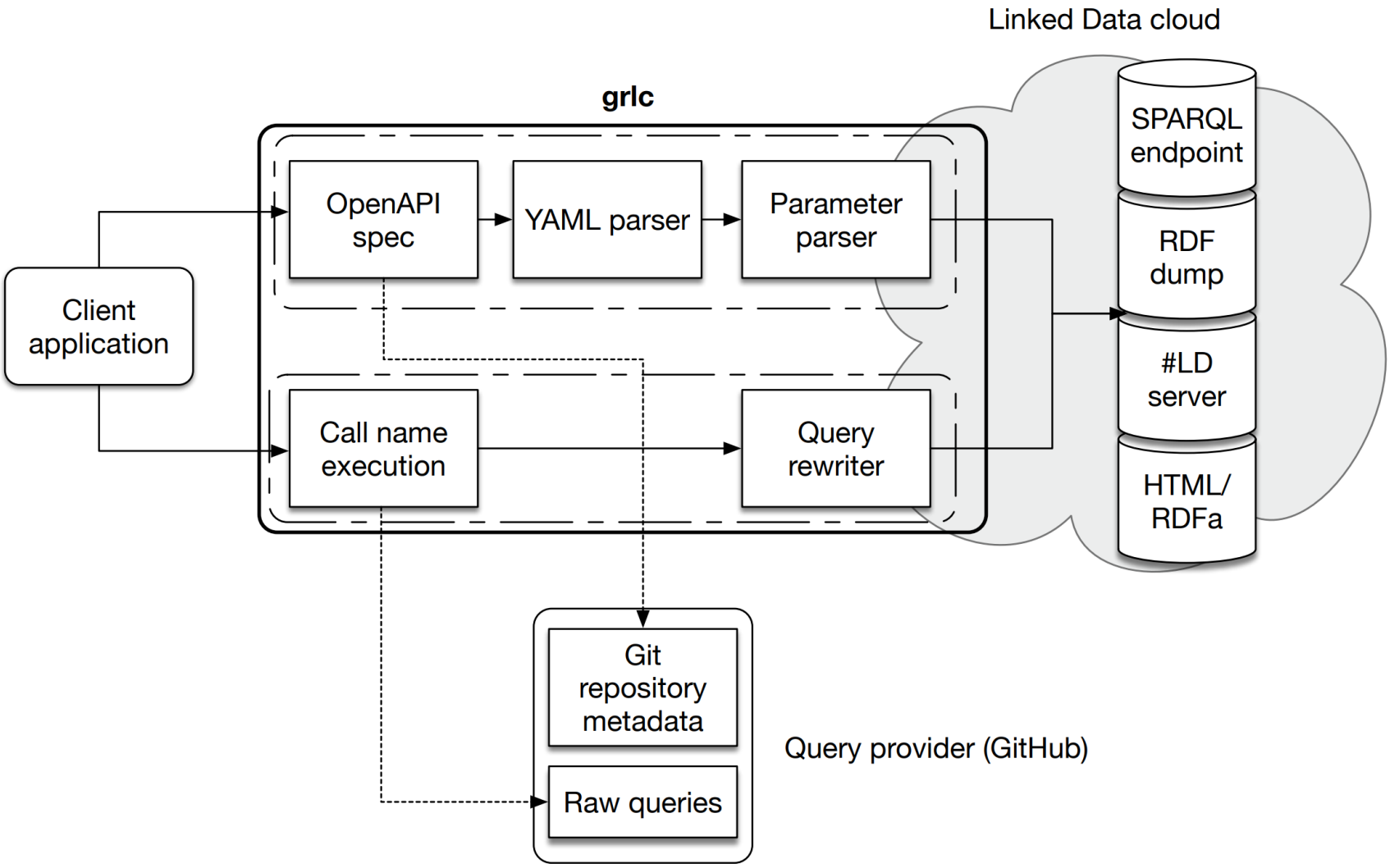
- Same basic principle of BASIL: 1 SPARQL query = 1 API operation
- Automatically builds Swagger spec and UI from SPARQL
- Answers API calls

But:

- External query management
- Organization of SPARQL queries in the GitHub repo matches organization of the API
- Thin layer – nothing stored server-side
- Maps
  - > GitHub API
  - > Swagger spec

VU

GRLC'S ARCHITECTURE



SPARQL DECORATOR SYNTAX

```
#+ summary: A brief summary of what the query does
#+ endpoint: http://example.org/sparql
#+ tags:
#+      - UseCase1
#+      - Awesomeness
```

```
PREFIX qb: <http://purl.org/linked-data/cube#>
PREFIX cedar: <http://lod.cedar-project.nl/vocab/cedar#>
PREFIX sdmx-dimension: <http://purl.org/linked-data/sdmx/2009/dimension#>
PREFIX sdmx-code: <http://purl.org/linked-data/sdmx/2009/code#>

SELECT ?municipality (SUM(?pop) AS ?tot)
FROM <urn:graph:cedar-mini:release>
WHERE { ?obs a qb:Observation .
        ?obs cedar:population ?pop .
```


VU

THE GRLC SERVICE

- Assuming your repo is at https://github.com/:owner/:repo
  - http://grlc.io/api/:owner/:repo/spec returns the JSON swagger spec
  - http://grlc.io/api/:owner/:repo/ returns the swagger UI
  - http://grlc.io/api/:owner/:repo/:operation?p_1=v_1...p_n=v_n calls operation with specifiec parameter values
    - Uses BASIL's SPARQL variable name convention for query parameters
- Sends requests to
  - https://api.github.com/repos/:owner/:repo to look for SPARQL queries and their decorators
  - https://raw.githubusercontent.com/:owner/:repo/master/file.rq to dereference queries, get the SPARQL, and parse it
  - Supports **versioning** through http://grlc.io/api/:owner/:repo/commit/:sha

VU

HANDS-ON

Go to https://github.com/

- If you have a GitHub account, log in
- If you don’t, create one and log in

HANDS-ON

Create a new repository (by clicking on the + sign in the upper right corner), and name it **grlc-tutorial** .

Check “Initialize this repository with a README” and click on “Create repository”

HANDS-ON

Open a new tab

Go to https://tinyurl.com/wikidata-sparql and choose a query you find interesting (e.g. cats)

Copy the query into the clipboard (Ctrl-C)

HANDS-ON

Back to the GitHub tab

Click on “Create new file”

Name it **cats.rq** or **cats.sparql** (yes, the extension matters!)

Paste the contents of the clipboard (Ctrl-V)

Check for errors!

Don’t click on Commit yet

VU

HANDS-ON

Need to add **prefixes** (keep names short, but need reference) and **metadata** (will sortour API operations neatly)

Typical prefixes (wd, wdt, wikibase, etc.): go to http://prefix.cc/ and use the lookup to add on top of your query e.g.

PREFIX wd: <http://www.wikidata.org/entity/>

CHECK ALL PREFIXES IN YOUR QUERY

VU

HANDS-ON

Now to the **metadata**

You **MUST** add

#+ endpoint: https://query.wikidata.org/sparql

(or any other endpoint name)

Optionally

```
#+ tags:
#+   - mytag1
#+   - mytag2
```


VU

HANDS-ON

Optionally

#+ pagination: 50
#+ summary: Gets names of important cats
#+ method: GET
(could be: POST, PUT, etc.)

HANDS-ON

Click on “**Commit new file**” (green button at the bottom)

Time to get those APIs working!

Go to
http://grlc.io/api/<github_username>/grlc-tutorial

And wait for the magic to happen

VU

HANDS-ON

You should see

HANDS-ON

If you don’t (or see missing queries):

- Check the query syntax
- Check all prefixes have been added
- Check #+ in all your metadata lines

After your fixes:

- Commit your changes
- Copy the commit hash
- Reload the API by going to **http://grlc.io/api/<github_username>/grlc-tutorial/commit/<hash>**

VU

HANDS-ON

If all your queries go to the same endpoint, you can omit the #+ endpoint entry and

Add an “**endpoint.txt**” file in your repo

Edit it and add the endpoint name

HANDS-ON

Exercise 1.

Repeat for a number of Wikidata queries (about 10). Create an API by collecting and organizing a set of queries that make sense together. Try to play with parameters. Try different response formats (CSV, JSON,e tc.). Make sure all of them work.

HANDS-ON

Exercise 2.

Let's make an API on top of your data!

1. Go to https://druid.datalegend.net
2. Click on your Dataset > Services
3. Select "SPARQL" as service type, "**Create service**"
4. Click on "Show" > "Visit SPARQL endpoint"
5. Copy the endpoint address

HANDS-ON

Try to fire the query

```
SELECT * WHERE {
  ?sub ?pred ?obj .
}
```

On that endpoint address via your API

VU

HANDS-ON

More queries, columns of your dataset

```
SELECT * WHERE {
  ?sub <http://www.w3.org/ns/csvw#name> ?obj .
}
```

HANDS-ON

More queries, ordered countries **(Notice your column URI might be different!)**

```
SELECT ?country WHERE {
  ?sub <https://iisg.amsterdam/resource/Country>
?country .
} ORDER BY ?country
```

```
SELECT ?country WHERE {
  ?sub <https://iisg.amsterdam/resource/Country>
?country .
} ORDER BY DESC(?country)
```


VU

HANDS-ON

More queries, top country by GDP

```
SELECT ?country ?gdp WHERE {
  ?sub <https://iisg.amsterdam/resource/Country>
?country ;
   <https://iisg.amsterdam/resource/Int> ?gdp .
} ORDER BY DESC(?gdp) LIMIT 1
```

HANDS-ON

More queries, GDP average

```
SELECT (AVG(?obj) AS ?avg) WHERE {
  ?sub <https://iisg.amsterdam/resource/Int> ?obj .
}
```

(Notice your column URI might be different!)

CONCLUSIONS

The spectrum of Linked Data clients: SPARQL intensive applications *vs* RESTful API applications

grlc uses **decoupling of SPARQL** from all client applications (including LDA) as a powerful practice

- Separates **query curation workflows** from everything else
- Allows at the same time
  - Web-friendly SPARQL queries
  - Web-friendly RESTful APIs
- (Moderate) costs mainly due to HTTP requests and query payload

- SPARQL projections
- Reusability of query catalogs

VU

THANK YOU!

@ALBERTMERONYO

DATALEGEND.NET
CLARIAH.NL