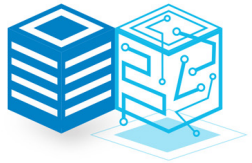


Generating and Linking
RDF data

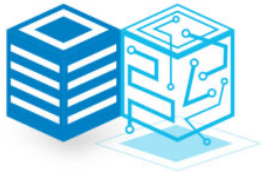
Andrea Cimmino
Ontology Engineering Group
Universidad Politécnica de Madrid, Spain

cimmino@fi.upm.es
@ACimmino

Date 2022/06/09
Place Cercedilla

A bit about me…

A bit about me…

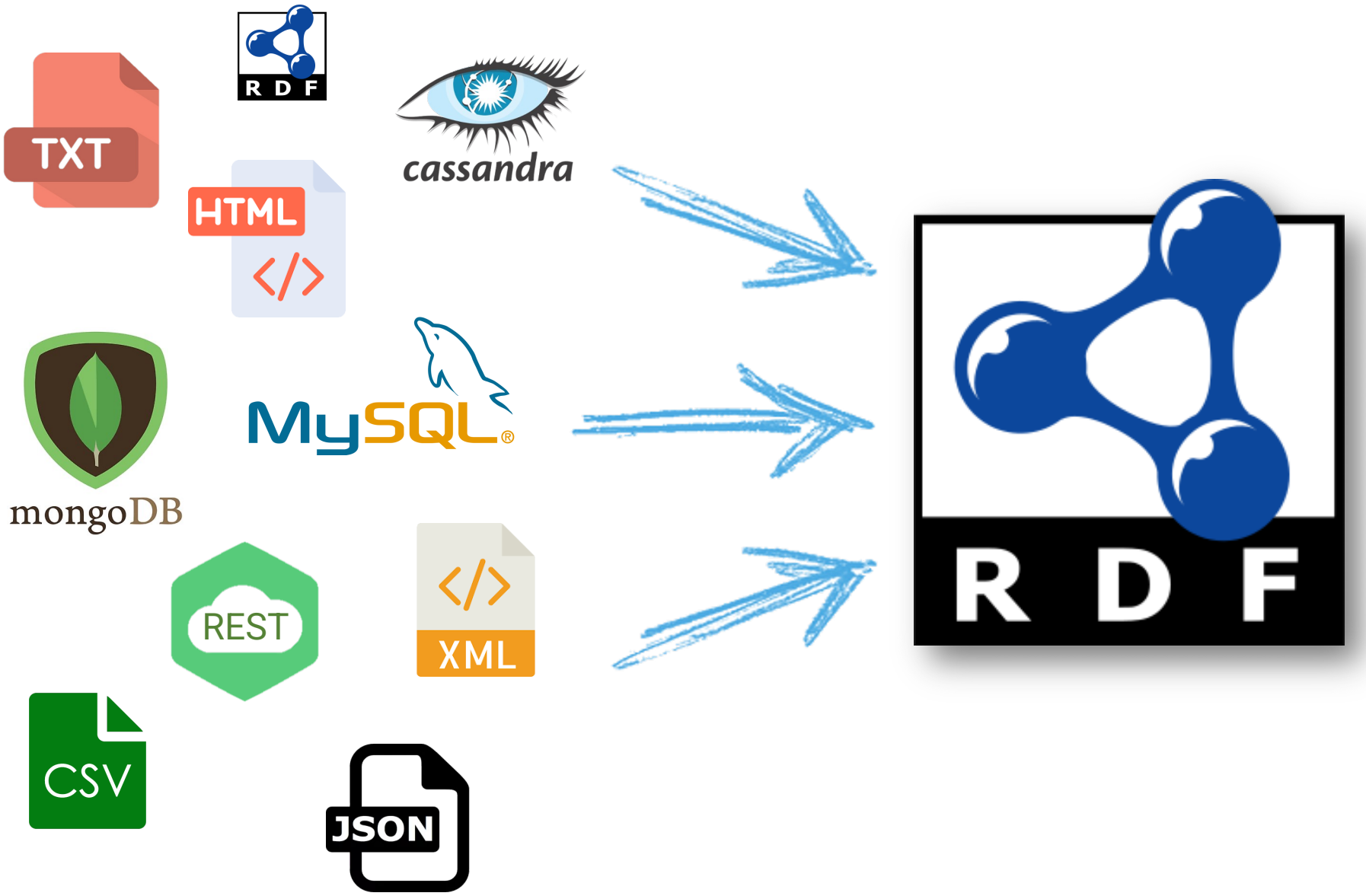


COGITO AURORAL

A bit about me…

https://bit.ly/ldac2022

Problem statement

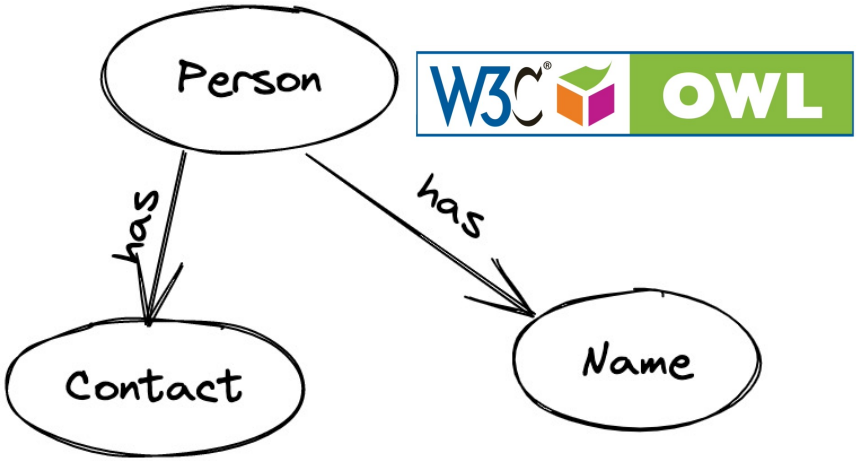


Problem statement

- A vast amount of data already exists
  - Data is publish as isolated islands of information
  - Data follows heterogeneous formats and models.
  - As it is, sometimes is hard to exploit or combine different data sources

- RDF data facilitates exploiting data
  - An “RDF version” of data provides normalised and standard mechanisms for consuming it.
  - RDF data is linkable, and thus, allows to combine data coming from different data sources. Which increases its value.
  - Through the usage of standardised ontologies interoperable data can be achieved

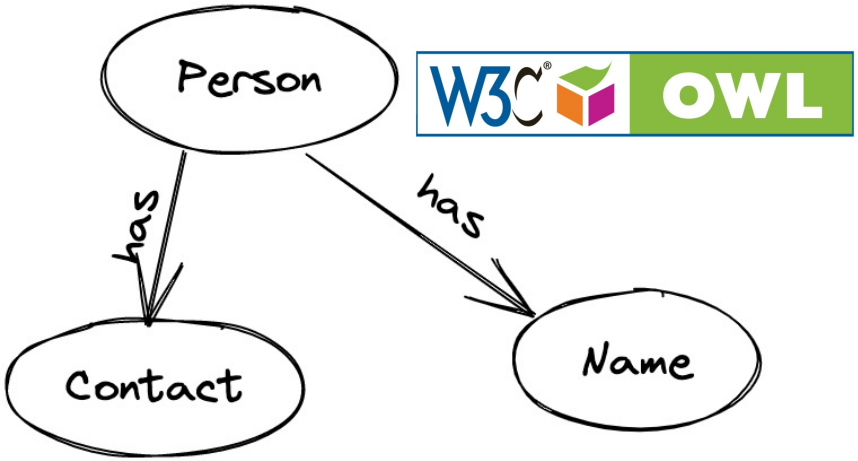
1. How RDF data is



2. Use an ontology

- A mapping language (RML, Helio, SPARQL Generate…)
- A tool for translating data (Google Refine, Karma, …)

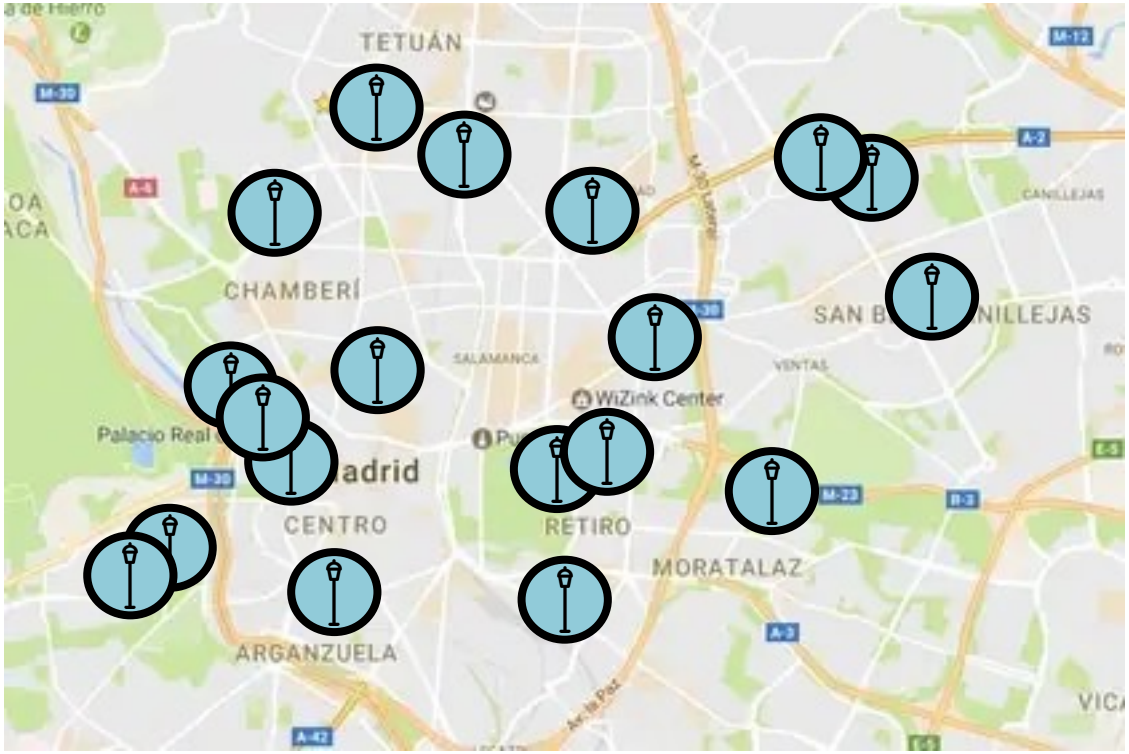
1. How RDF data is



2. Use an ontology

- A mapping language (RML, Helio, SPARQL Generate…)
- A tool for translating data (Google Refine, Karma, …)

- Lampposts in Madrid from its open data portal

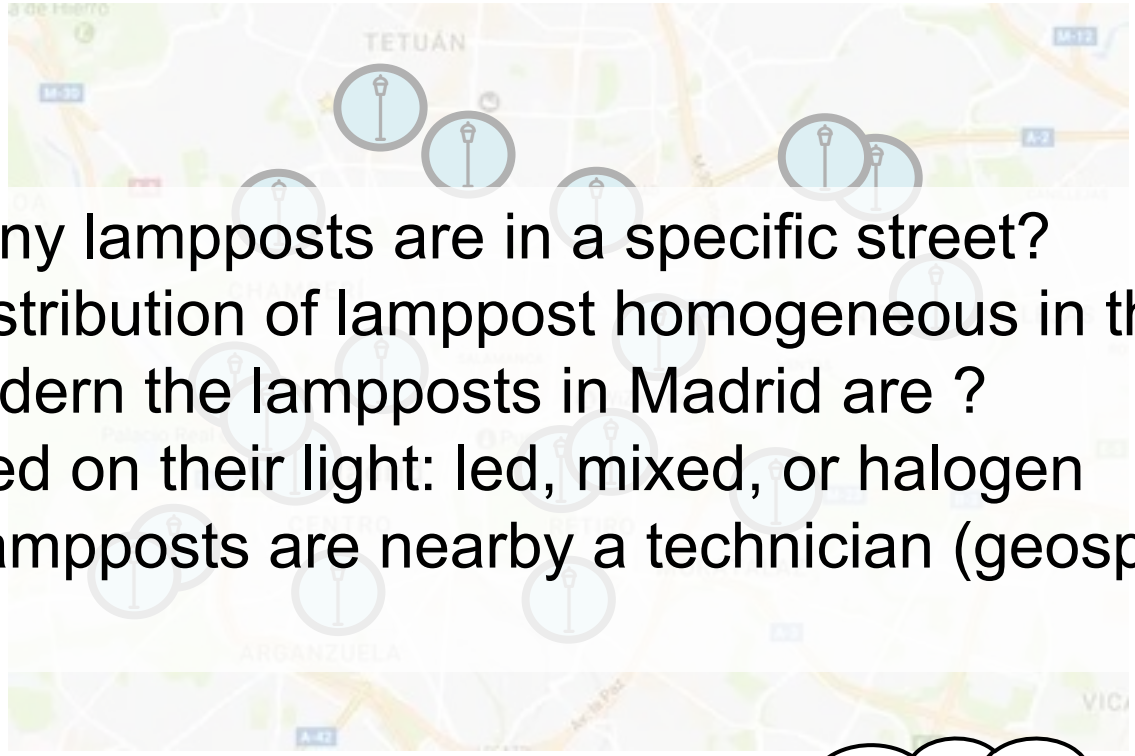


datos abiertos

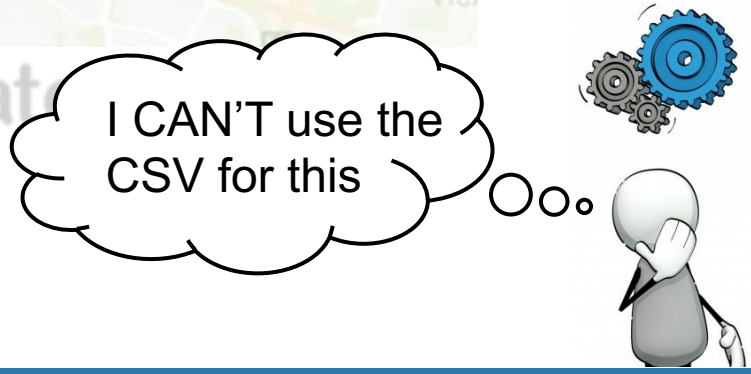
- Lampposts in Madrid from its open data portal

- How many lampposts are in a specific street?
- Is the distribution of lamppost homogeneous in the city?
- How modern the lampposts in Madrid are ?
  - Based on their light: led, mixed, or halogen
- Which lampposts are nearby a technician (geosparql) ?

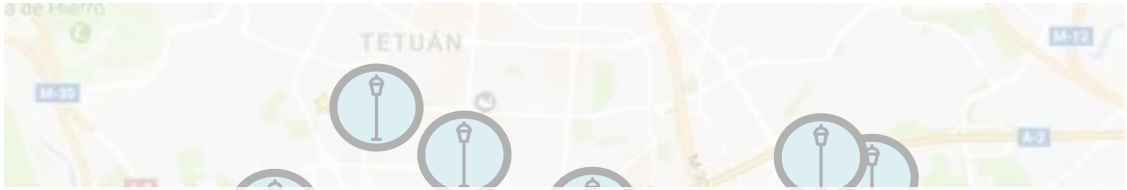
- Lampposts in Madrid from its open data portal



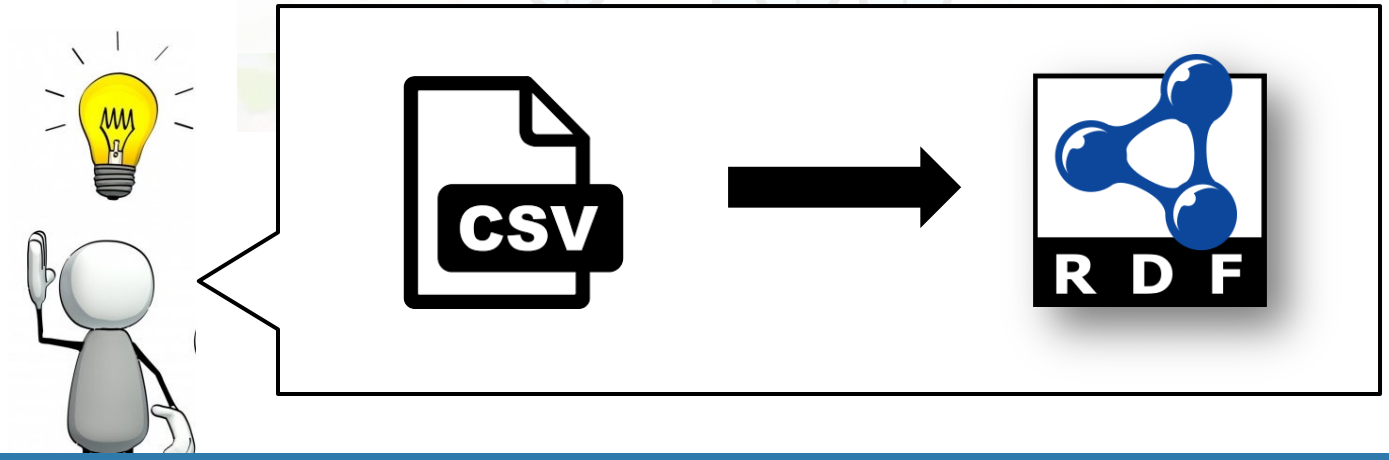
- How many lampposts are in a specific street?
- Is the distribution of lamppost homogeneous in the city?
- How modern the lampposts in Madrid are ?
  - Based on their light: led, mixed, or halogen
- Which lampposts are nearby a technician (geosparql) ?



- Lampposts in Madrid from its open data portal

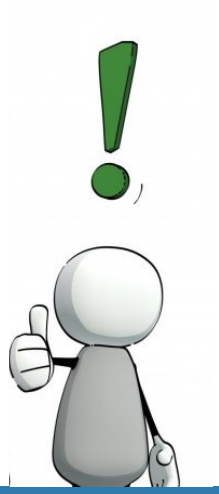


- How many lampposts are in a specific street?
- Is the distribution of lamppost homogeneous in the city?
- How modern the lampposts in Madrid are ?
  - Based on their light: led, mixed, or halogen
- Which lampposts are nearby a technician (geosparql) ?



- Lampposts in Madrid from its open data portal

- How many lampposts are in a specific street?
- Is the distribution of lamppost homogeneous in the city?
- How modern the lampposts in Madrid are ?
  - Based on their light: led, mixed, or halogen
- Which lampposts are nearby a technician (geosparql) ?



SPARQL

Data linking: increasing value of semantic data

Lampposts in Madrid

datos abiertos

+

Weather API

=

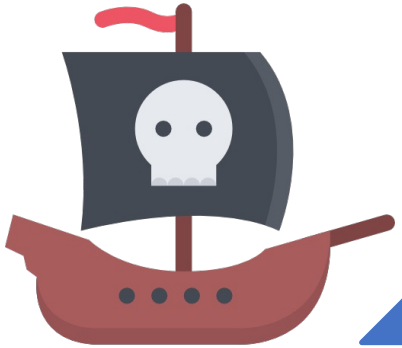
Smart Decisions

In summary

Data with potential value

Heterogeneous data

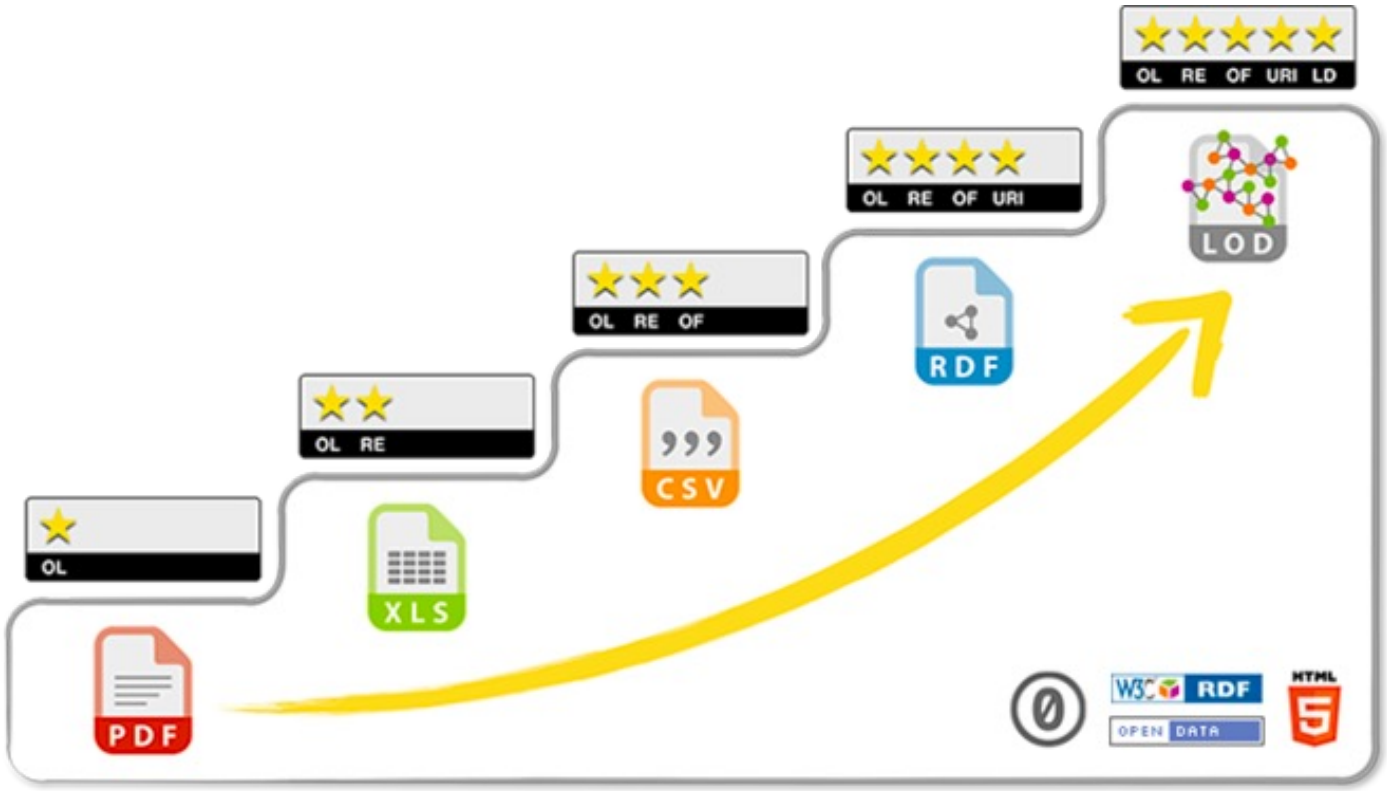
Data with value



RDF data

Highly valuable data

Linked RDF data



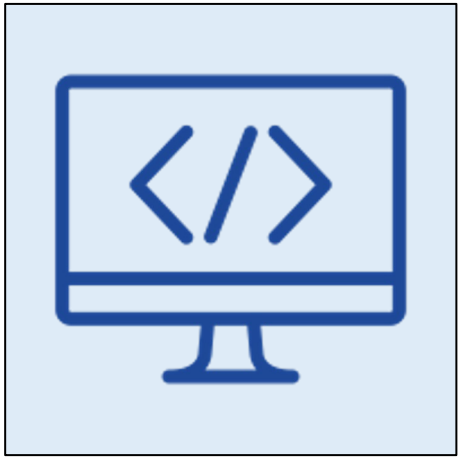
5th start open data

Index

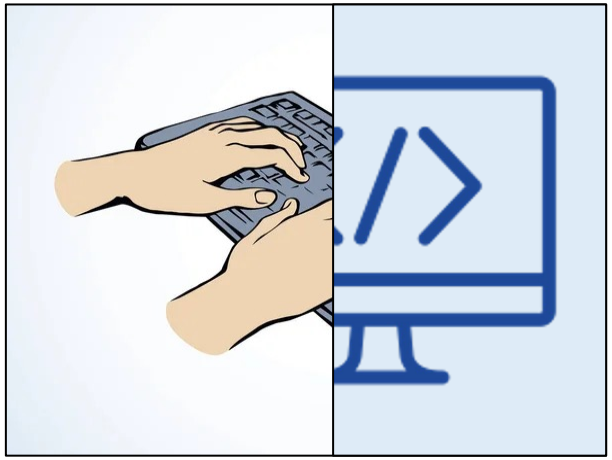
- **Generating and linking RDF data**
- Using declarative mappings: RML mappings
- Using procedural mappings: SIoT mappings

Generating the RDF data

Ad-hoc code



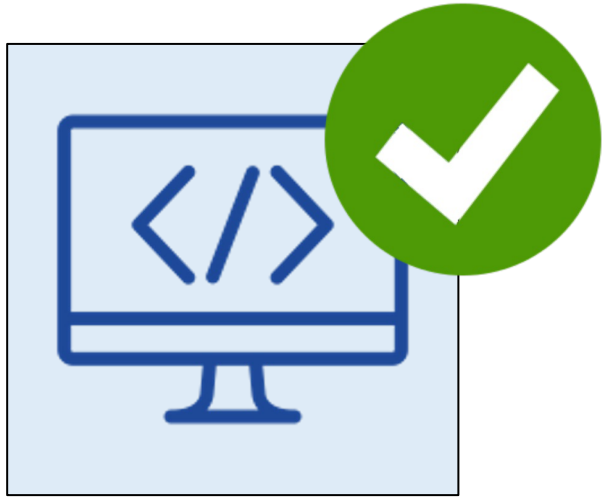
Generic tools



Ad-hoc code for cleaning data + Generic tools

Generating the RDF data

Ad-hoc code

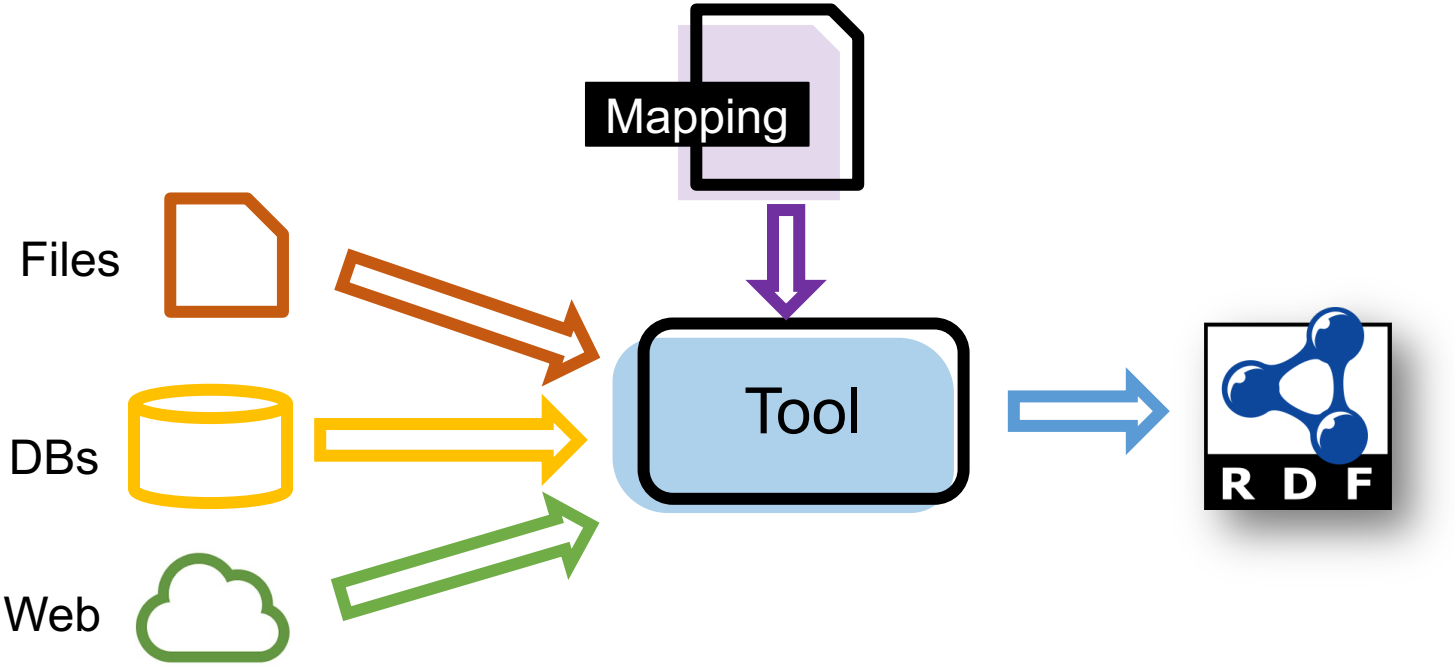


Generic tools

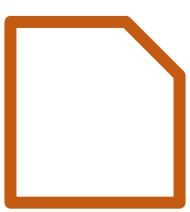
Ad-hoc code for cleaning data + Generic tools

Generic tools for generating RDF

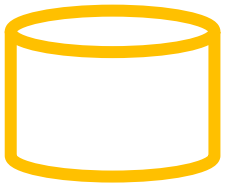
- The Tool is feed with a Mapping.
- The Tool executes the mapping, and as a result, outputs an RDF version of the heterogeneous data.
- Some Mappings can deal with multiple data sources and formats.



Eagle's eye view of Mappings



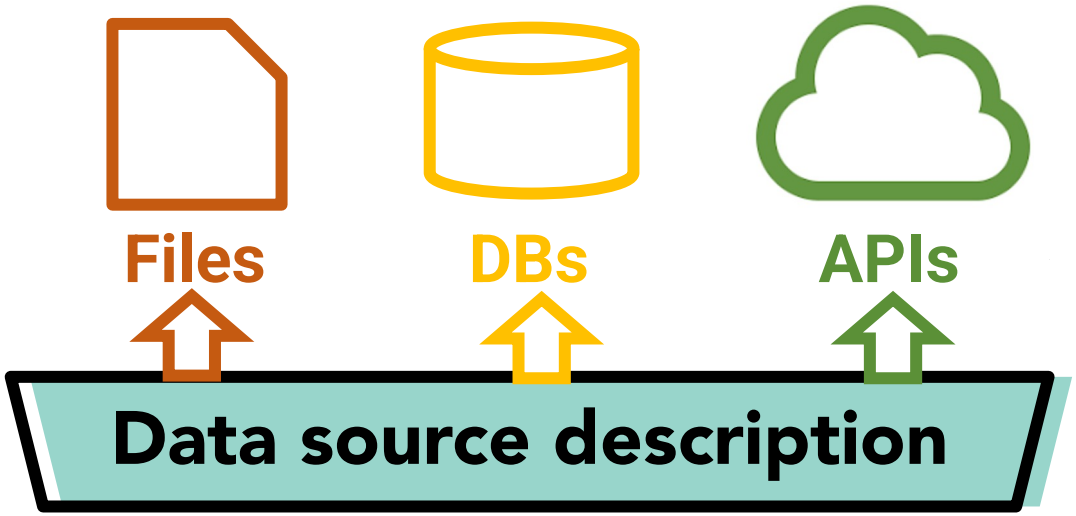
Files

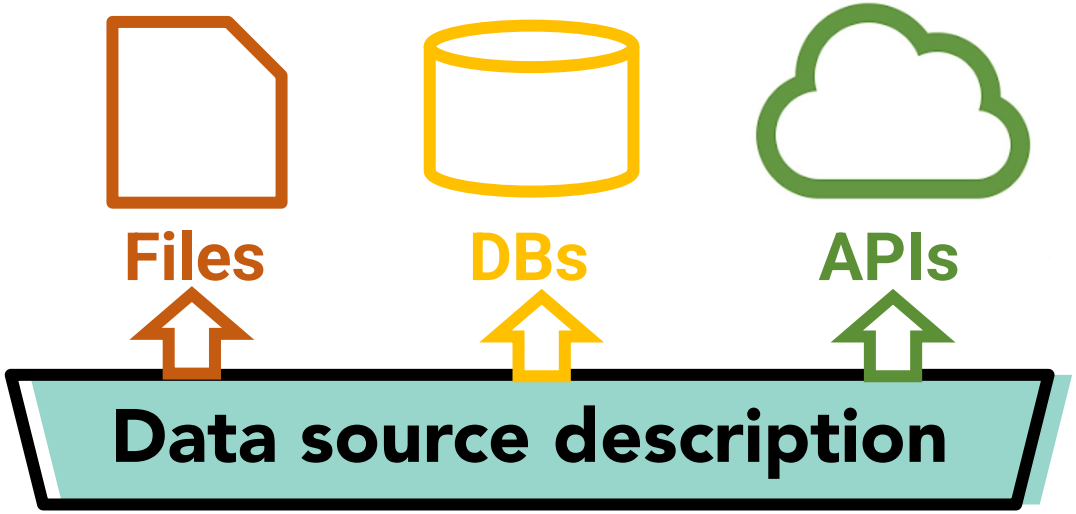


DBs

APIs

Eagle's eye view of Mappings

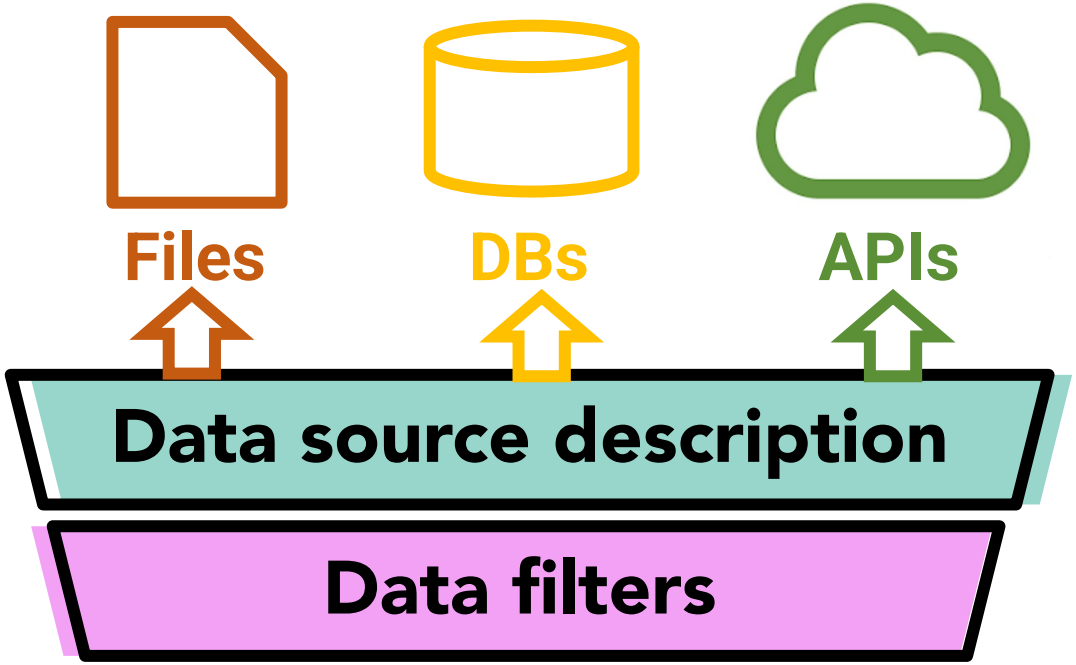




Mapping content

- Protocols for fetching data
  - File, HTTP, …
- Format of data
- Other:
  - Security/credentials
  - Headers for HTTP
  - ….

Eagle's eye view of Mappings

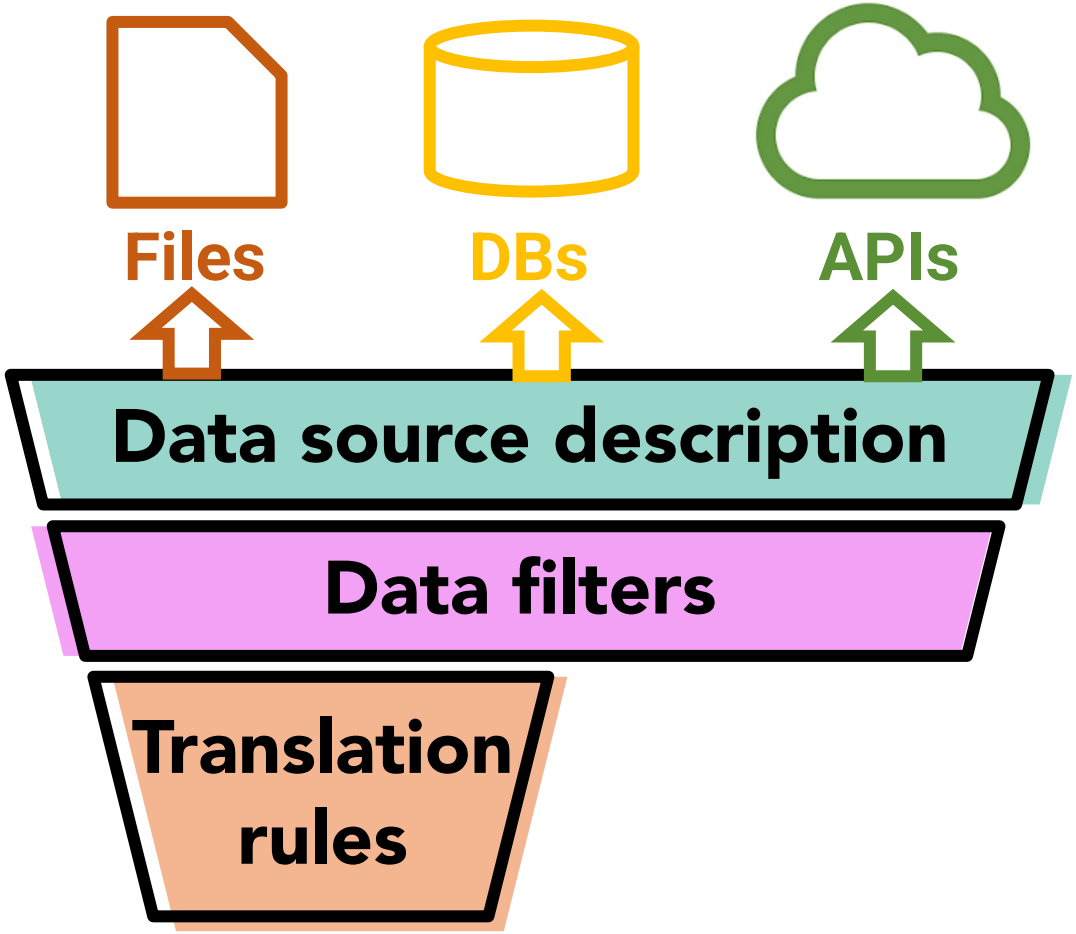


Mapping content

- Protocols for fetching data
  - File, HTTP, …
- Format of data
- Other:
  - Security/credentials
  - Headers for HTTP
  - ….

- Filtering criteria
  - Xpath, JSON Path, …
- Some iterator

Eagle's eye view of Mappings



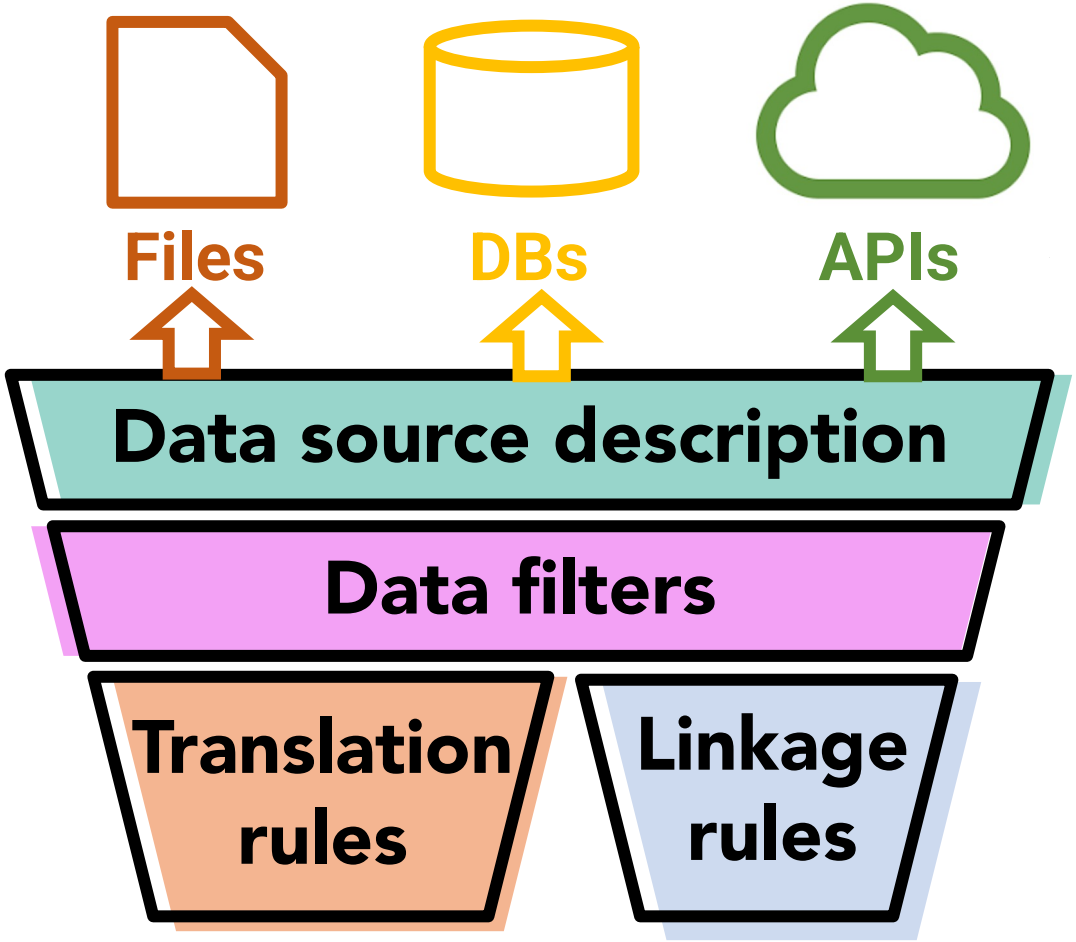
Mapping content

- Protocols for fetching data
  - File, HTTP, …
- Format of data
- Other:
  - Security/credentials
  - Headers for HTTP
  - ….

- Filtering criteria
  - Xpath, JSON Path, …
- Some iterator

- Combination of filters, functions, and static values

Eagle's eye view of Mappings



Mapping content

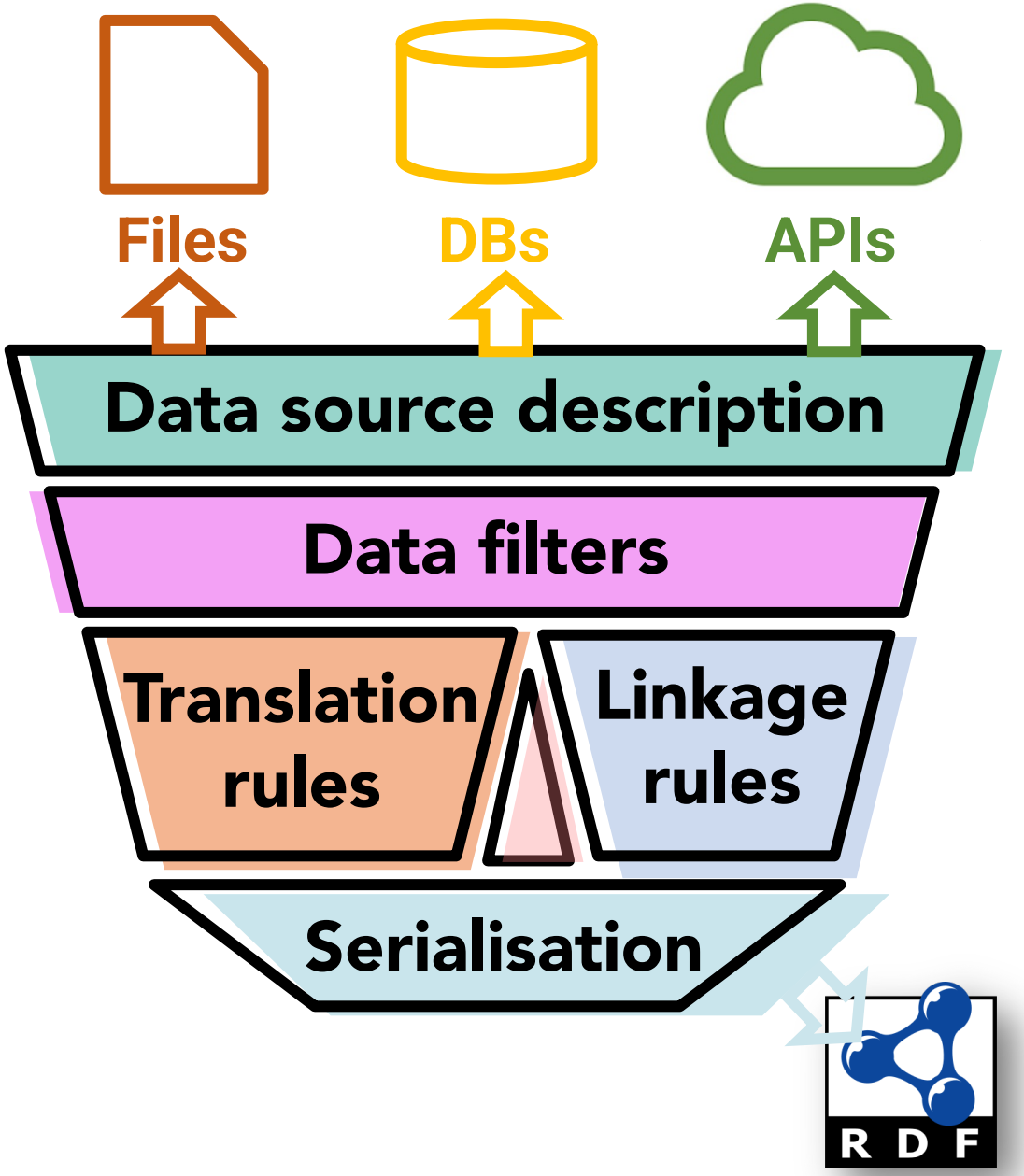
- Protocols for fetching data
  - File, HTTP, …
- Format of data
- Other:
  - Security/credentials
  - Headers for HTTP
  - ….

- Filtering criteria
  - Xpath, JSON Path, …
- Some iterator

- Combination of filters, functions, and static values

- Conditions for linking triples translated from different data sources

Eagle's eye view of Mappings



Mapping content

- Protocols for fetching data
  - File, HTTP, …
- Format of data
- Other:
  - Security/credentials
  - Headers for HTTP
  - ….

- Filtering criteria
  - Xpath, JSON Path, …
- Some iterator

- Combination of filters, functions, and static values

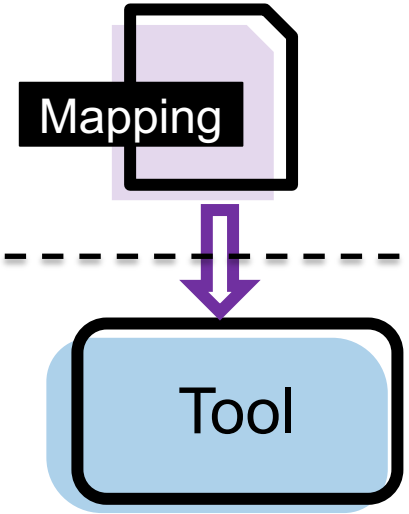
- Conditions for linking triples translated from different data sources

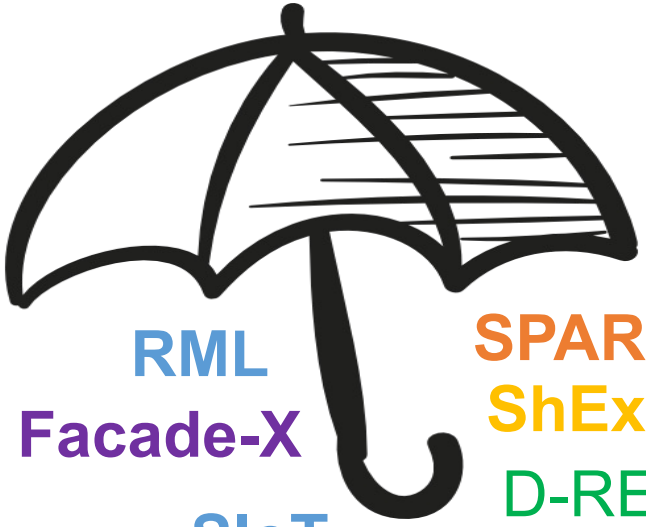
- Data output format (RDF)
  - Turtle, N3, JSONLD1.1, …

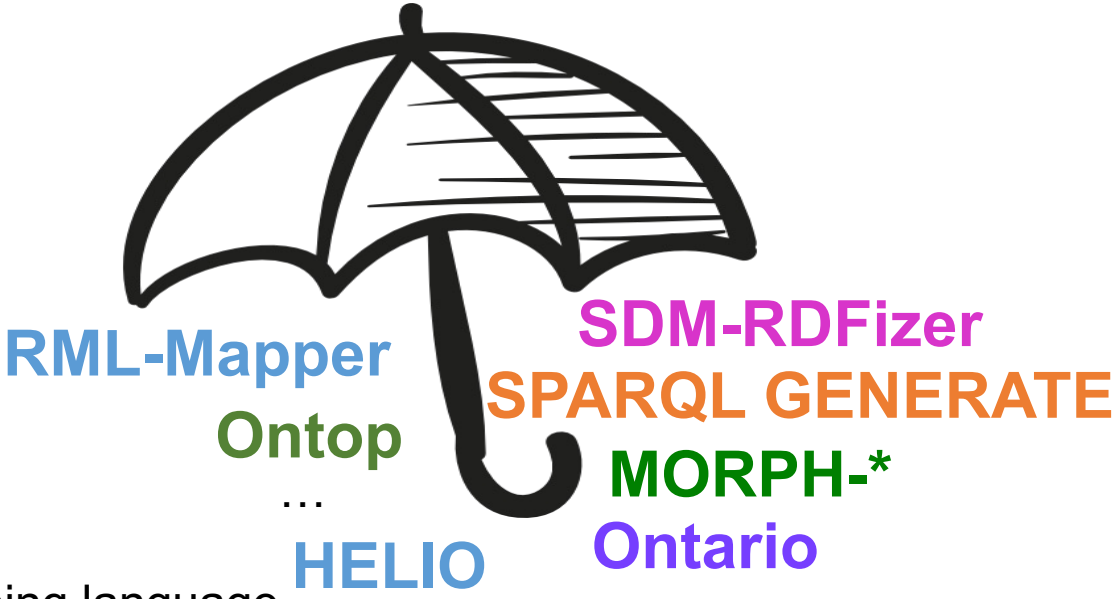
Mappings

- The number of mapping languages is large
- Some languages are especially endowed for some data sources or formats
- Not all mapping languages support all the same features
  - Some only support one kind of data source, or format
  - Some do not have functions (cleaning, linking, …)
  - Some are declarative and others procedural
  - Some mappings support conditional generation of triples
  - ...
- The mapping language chosen for generating and linking RDF data may impose some restrictions on the process
  - For instance, if the mapping does not support cleaning functions the data will need to be pre-processed ad-hoc.

A general perspective

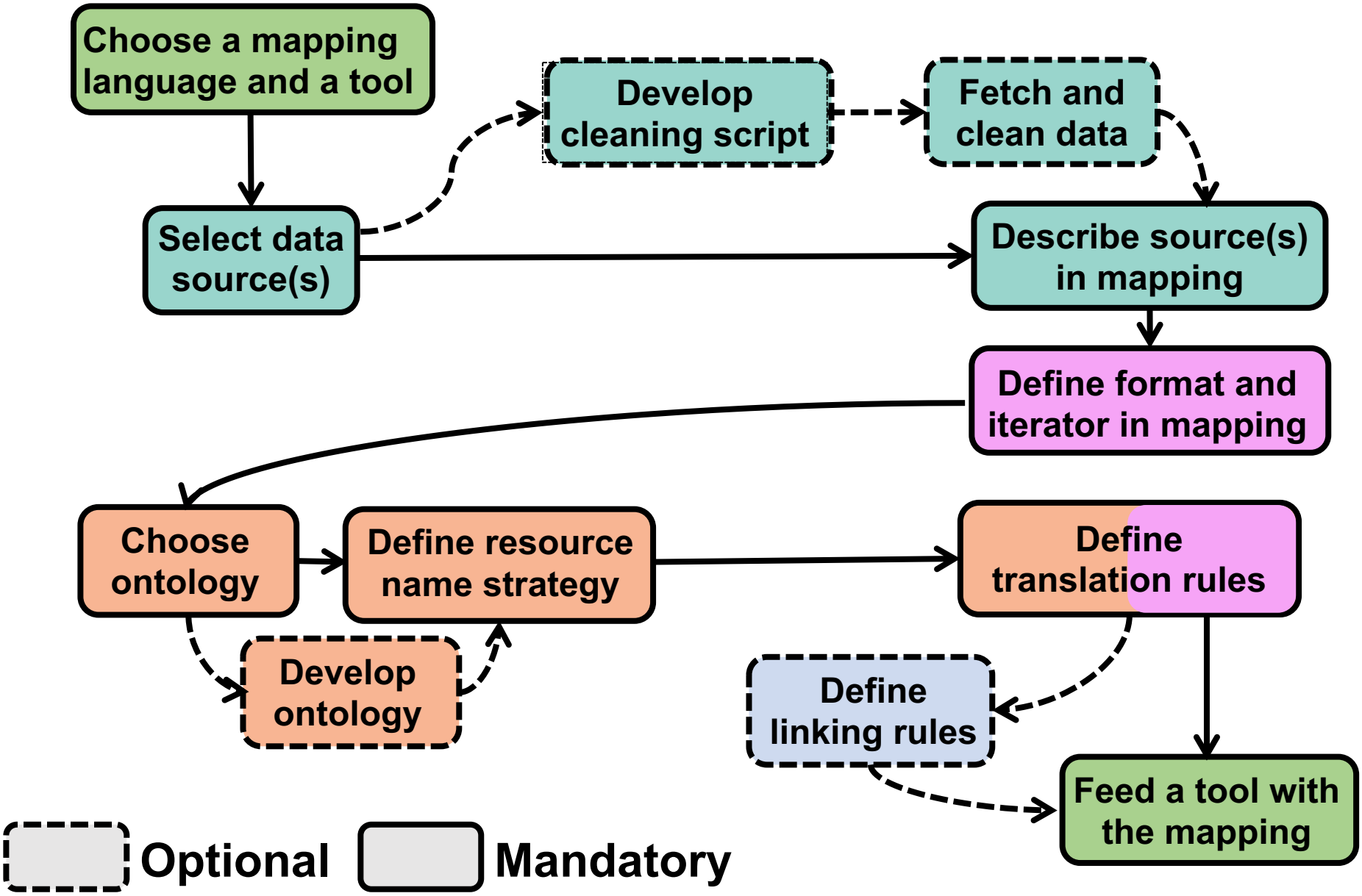






*Tools usually process only a mapping language

Methodology for generating and linking RDF

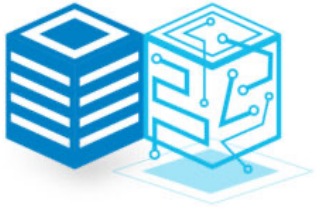


- Generating and linking RDF data
- **Using declarative mappings: RML mappings**
- Using procedural mappings: SIoT mappings

- Several tools support RML mappings
- Documentation: https://rml.io/docs/
- RML specification: https://rml.io/specs/rml/

- A set of tools for generating RDF from heterogeneous data sources
- Designed to be highly extensible using plugins
  - Supports multiple mappings (RML, SIoT, …)

https://github.com/helio-ecosystem



COGITO

AURORAL

DELTA

Sample 1 scenario

people.csv

ID	Name	Phone	City
1	Lily D.	585-610	Oneonta
…	…	…	…

location.csv

CName	latitude	longitude
Oneonta	44.9625	-74.4168
…	…	…

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

<http://sample.com/resource/P1>
        a foaf:Person ;
        foaf:name "Lily D." ;
        foaf:phone "585-610-2352" ;
        rdfs:seeAlso <https://dbpedia.org/page/Oneonta> ;
        foaf:based_near <https://sample.com/resource/Oneonta> .

<https://sample.com/resource/Oneonta>
        foaf:name "Oneonta" ;
        geo:lat "44.9625" ;
        geo:long "-74.4168" .
```

Sample 1 scenario

people.csv

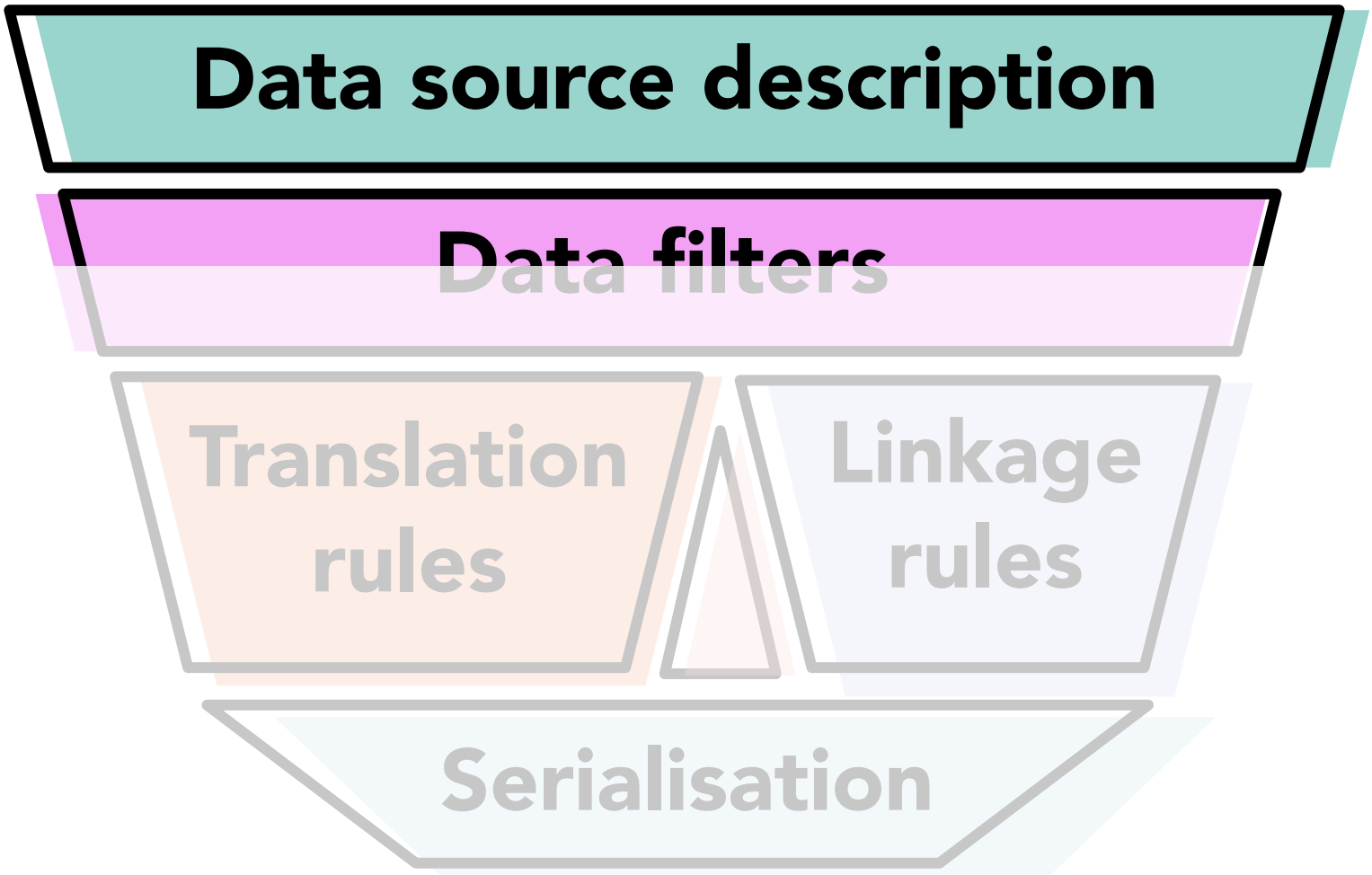
ID	Name	Phone	City
1	Lily D.	585-610	Oneonta
…	…	…	…

location.csv

CName	latitude	longitude
Oneonta	44.9625	-74.4168
…	…	…

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

<http://sample.com/resource/P1>
        a foaf:Person ;
        foaf:name "Lily D." ;
        foaf:phone "585-610-2352" ;
        rdfs:seeAlso <https://dbpedia.org/page/Oneonta> ;
        foaf:based_near <https://sample.com/resource/Oneonta> .

<https://sample.com/resource/Oneonta>
        foaf:name "Oneonta" ;
        geo:lat "44.9625" ;
        geo:long "-74.4168" .
```

RML: Data source description + filters



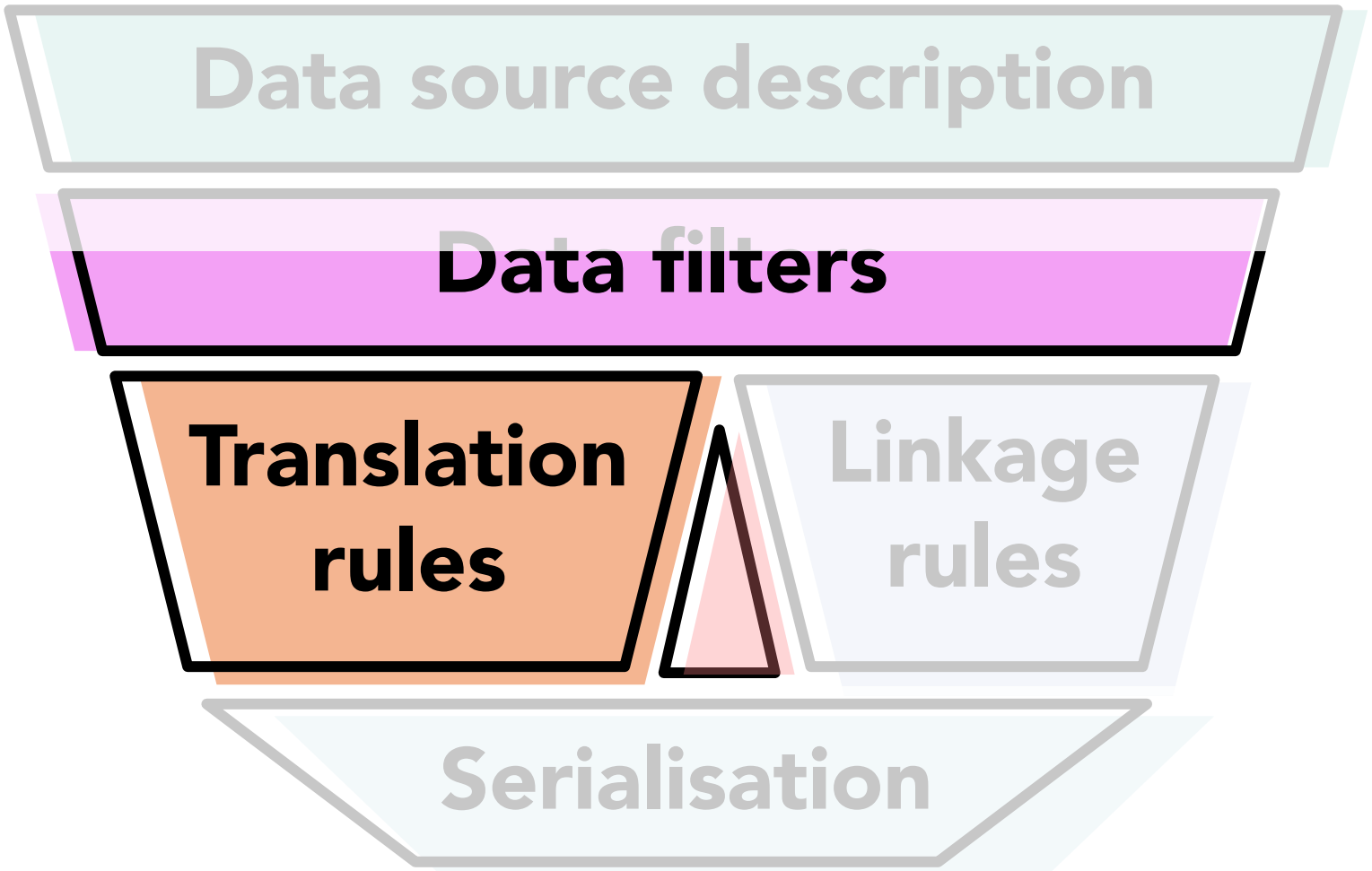
RML: Data source description + filters

```
<#TriplesMap1>
    rml:logicalSource [
        rml:source "people.csv";
        rml:referenceFormulation ql:CSV
    ];
    ...
```

people.csv

ID	Name	Phone	City
1	Lily D.	585-610-2352	Oneonta
…	…	…	…

RML: filters with translation rules



RML: filters with translation rules (subject)

```
<#TriplesMap1>
    rml:logicalSource [
        rml:source "people.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://sample.com/resource/P{ID}";
        rr:termType rr:IRI; rr:class foaf:Person
    ];
    …
```

people.csv

ID	Name	Phone	City
1	Lily D.	585-610-2352	Oneonta
…	…	…	…

RML: filters with translation rules (data properties)

```
<#TriplesMap1>
    rml:logicalSource [
        rml:source "people.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://sample.com/resource/P{ID}";
        rr:termType rr:IRI; rr:class foaf:Person
    ];
    rr:predicateObjectMap [
        rr:predicate  foaf:name;
        rr:objectMap [
            rml:reference "Name"; rr:termType rr:Literal ;
            rr:datatype xsd:string
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate  foaf:phone;
        rr:objectMap [
            rml:reference "Phone";
            rr:termType rr:Literal ;
        ]
    ];
```

people.csv

ID	Name	Phone	City
1	Lily D.	585-610-2352	Oneonta
…	…	…	…

RML: filters with translation rules (object properties)

```
<#TriplesMap1>
     rml:logicalSource [ rml:source "people.csv"; rml:referenceFormulation ql:CSV ];
     rr:subjectMap [
         rr:template "http://sample.com/resource/P{ID}";
         rr:termType rr:IRI; rr:class foaf:Person
     ];
     rr:predicateObjectMap [
          rr:predicate  foaf:name;
          rr:objectMap [
               rml:reference "Name"; rr:termType rr:Literal; rr:datatype xsd:string
          ]
     ];
     rr:predicateObjectMap [
          rr:predicate  foaf:phone;
          rr:objectMap [ rml:reference ”Phone"; rr:termType rr:Literal ; ]
      ];
     rr:predicateObjectMap [
          rr:predicate rdfs:seeAlso;
          rr:objectMap [
              rr:template "https://dbpedia.org/page/{City}";
          ]
      ];
```

people.csv

ID	Name	Phone	City
1	Lily D.	585-610-2352	Oneonta
…	…	…	…

Mapping for locations

- How is the mapping for locations?

RML: mapping for locations

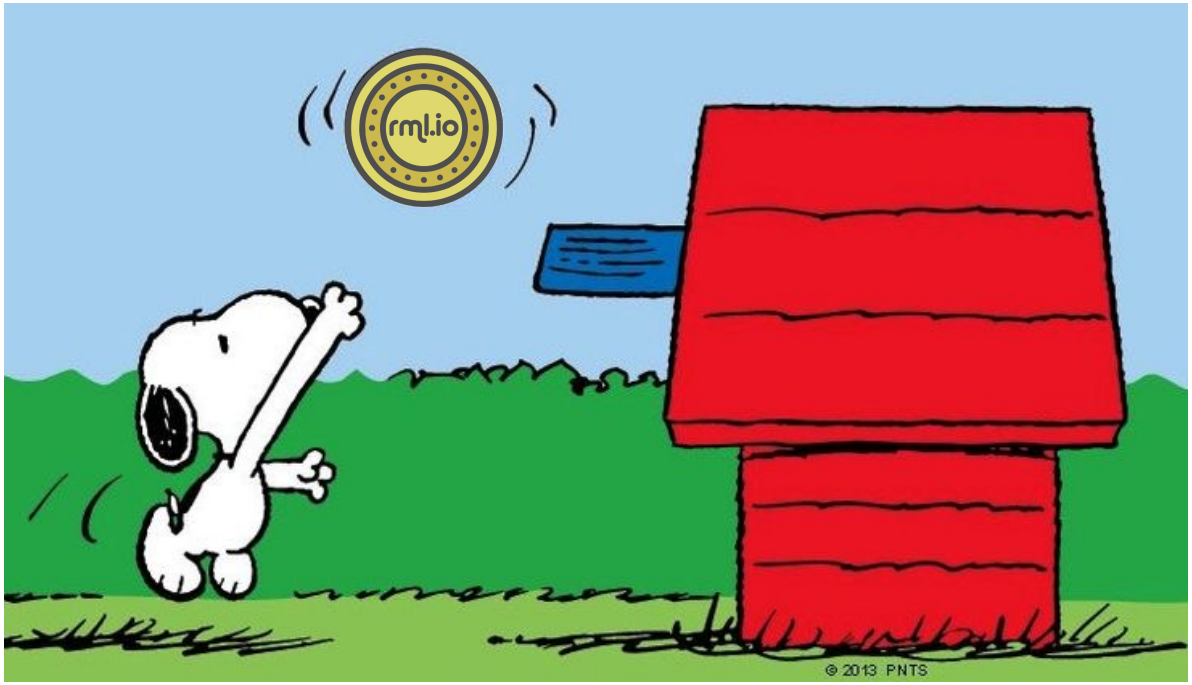
```
<#TriplesMap2>
    rml:logicalSource [
      rml:source "locations.csv"; rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://sample.com/resource/{CName}";
        rr:termType rr:IRI;
    ];
    rr:predicateObjectMap [
        rr:predicate  foaf:name;
        rr:objectMap [ rml:reference "CName"; rr:termType rr:Literal;]
    ];
    rr:predicateObjectMap [
        rr:predicate  geo:lat;
        rr:objectMap [
            rml:reference "latitude"; rr:termType rr:Literal ;
            rr:datatype xsd:double ]
    ];
    rr:predicateObjectMap [
        rr:predicate  geo:long;
        rr:objectMap [
            rml:reference "longitude"; rr:termType rr:Literal ;
            rr:datatype xsd:double ]
    ].
```

RML: linking

```
<#TriplesMap1>
    rml:logicalSource [ ... ];
    rr:subjectMap [ ... ];          # Subject definition
    rr:predicateObjectMap [ ... ]; # foaf:name definition
    rr:predicateObjectMap [ ... ]; # foaf:phone definition
    rr:predicateObjectMap [ ... ]; # rdfs:seeAlso definition
    rr:predicateObjectMap [
        rr:predicate foaf:based_near;
        rr:objectMap [
          rr:parentTriplesMap <#TriplesMap2>;
          rr:joinCondition [
            rr:child "City";  rr:parent "CName";   ];
        ]
      ] .
<#TriplesMap2>
    rml:logicalSource [ ... ];
    rr:subjectMap [ ... ];          # Subject definition
    rr:predicateObjectMap [ ... ]; #foaf:name definition
   rr:predicateObjectMap [ ... ]; #geo:lat definition
    rr:predicateObjectMap [ ... ]; #geo:long definition
```

Using Helio for running the mapping

- **Sample2 – Working with JSONs**
- Sample3 – Complex JSON structures I
- Sample4 – Complex JSON structures II



Sample 2 scenario

people.csv

ID	Name	Phone	City
1	Lily D.	585-610	Oneonta
…	…	…	…

locations.json

```
[
 {
  "name": "Oneonta",
  "surface": "90.16",
  "population": "1812",
  "latitude": "44.9625428",
  "longitude": "-74.4168577"
 }, …
]
```

```
<http://sample.com/resource/P1>
        a foaf:Person ;
        foaf:name "Lily D." ;
        foaf:phone "585-610-2352" ;
        rdfs:seeAlso <https://dbpedia.org/page/Oneonta> ;
        foaf:based_near <https://sample.com/resource/Oneonta> .

<https://sample.com/resource/Oneonta>
        foaf:name "Oneonta" ;
        geo:lat "44.9625"
        geo:long "-74.4168" ;
        onto:population 1.812E3;
        onto:surface 90.16 .
```

Sample 2 mapping

```
<#TriplesMap2>
  rml:logicalSource [
      rml:source "./data/sample2/locations.json";
      rml:referenceFormulation ql:JSONPath;
      rml:iterator "$.*"
    ];
  rr:subjectMap [
      rr:template "http://sample.com/resource/{name}";
      rr:termType rr:IRI;
  ];
  rr:predicateObjectMap [
      rr:predicate  foaf:name;
      rr:objectMap [
        rml:reference "name";
        rr:termType rr:Literal;
      ]
    ];
   rr:predicateObjectMap [
    rr:predicate  onto:surface;
    rr:objectMap [
      rml:reference "surface";
      rr:termType rr:Literal ;
      rr:datatype xsd:double
    ]
  ];
```

Sample 2 mapping

```
<#TriplesMap2>
  rml:logicalSource [
      rml:source "./data/sample2/locations.json";
      rml:referenceFormulation ql:JSONPath;
      rml:iterator "$.*"
    ];
  rr:subjectMap [
      rr:template "http://sample.com/r
      rr:termType rr:IRI;
  ];
  rr:predicateObjectMap [
      rr:predicate  foaf:name;
      rr:objectMap [
        rml:reference "name";
        rr:termType rr:Literal;
      ]
    ];
   rr:predicateObjectMap [
    rr:predicate  onto:surface;
    rr:objectMap [
      rml:reference "surface";
      rr:termType rr:Literal ;
      rr:datatype xsd:double
    ]
  ];
```

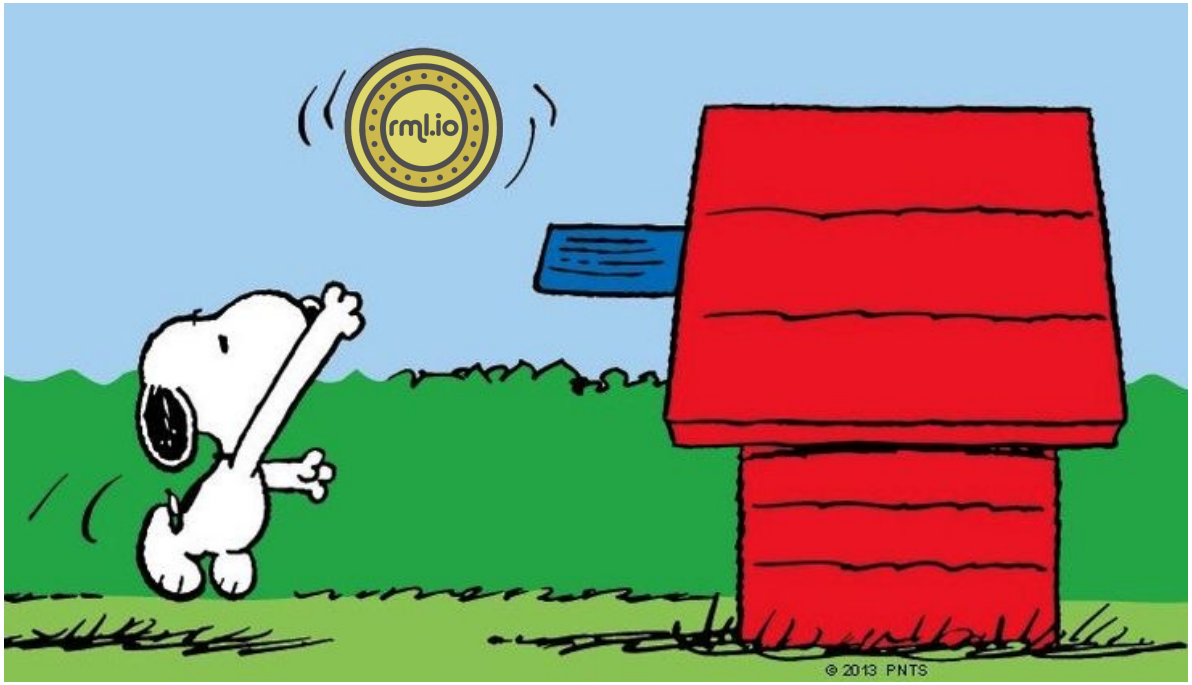
```
rr:predicateObjectMap [
    rr:predicate  onto:population;
    rr:objectMap [
      rml:reference "population";
      rr:termType rr:Literal ;
      rr:datatype xsd:double
    ]
  ];
  rr:predicateObjectMap [
    rr:predicate  geo:lat;
    rr:objectMap [
      rml:reference "latitude";
      rr:termType rr:Literal ;
      rr:datatype xsd:double
    ]
  ];
  rr:predicateObjectMap [
      rr:predicate  geo:long;
      rr:objectMap [
      rml:reference "longitude";
         rr:termType rr:Literal ;
         rr:datatype xsd:double ]
    ].
```

- What should be changed in the <#TriplesMap1> for linking the locations from the JSON?

- What should be changed in the <#TriplesMap1> for linking the locations from the JSON?

```
<#TriplesMap1>
    rml:logicalSource [...];
    rr:subjectMap [...];
    rr:predicateObjectMap [...];
    rr:predicateObjectMap [...];
    rr:predicateObjectMap [...];
    rr:predicateObjectMap [
    rr:predicate foaf:based_near;
    rr:objectMap [
      rr:parentTriplesMap <#TriplesMap2>;
      rr:joinCondition [
        rr:child "City";
        rr:parent "name";
      ];
    ]
  ] .
```

- Sample2 – Working with JSONs
- **Sample3 – Complex JSON structures I**
- Sample4 – Complex JSON structures II



Sample 3 scenario

people.csv

ID	Name	Phone	City
1	Lily D.	585-610	Oneonta
…	…	…	…

locations.json

```
[
  {
    "name": "Oneonta",
    "features": {
      "surface": {
        "value": 90.16,
        "units": "Km2"
      },
      "population": {
        "amount": "1,812",
        "date": 2018
      }
    },
    "location": {
      "latitude": "44.9625428",
      "longitude": "-74.4168577"
    }
  }
]
```

```
<http://sample.com/resource/P1>
        a foaf:Person ;
        foaf:name "Lily D." ;
        foaf:phone "585-610-2352" ;
        rdfs:seeAlso <https://dbpedia.org/p
        foaf:based_near <https://sample.co

<https://sample.com/resource/Oneonta>
        foaf:name "Oneonta" ;
        geo:lat "44.9625"
        geo:long "-74.4168" ;
        onto:population 1.812E3;
        onto:surface 90.16 .
```

Sample 3 mapping

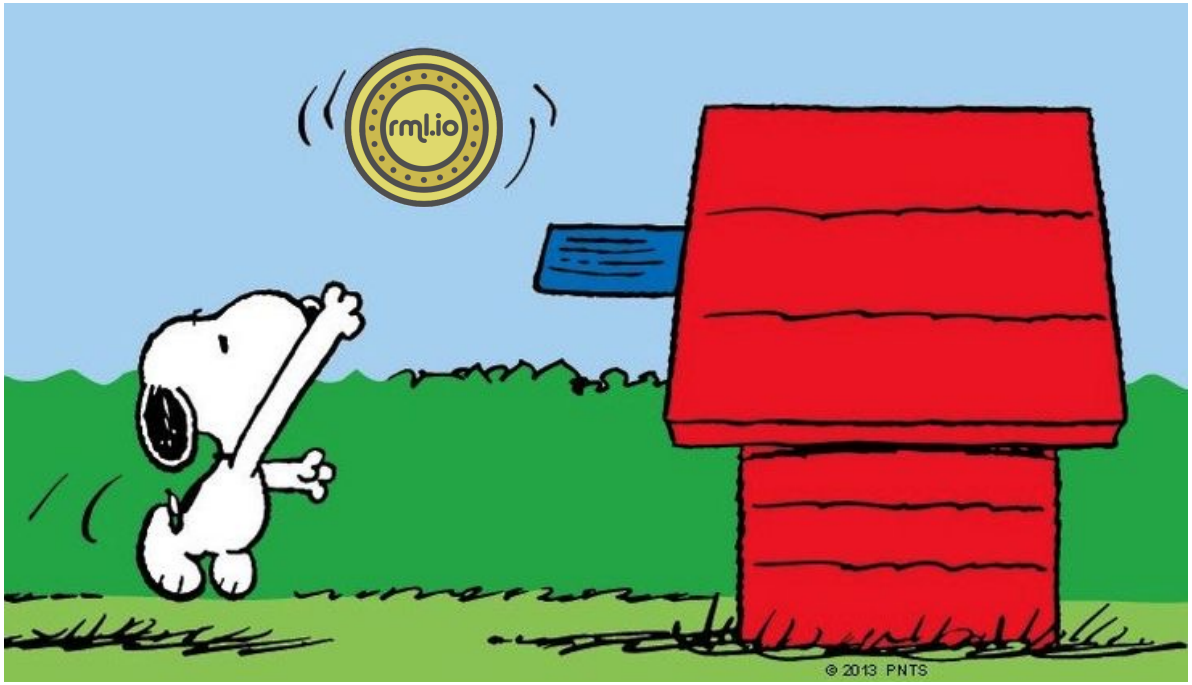
```
<#TriplesMap2>
  rml:logicalSource [
      rml:source "./data/sample3/locations.json";
      rml:referenceFormulation ql:JSONPath;
      rml:iterator "$.*"
    ];
  rr:subjectMap [
      rr:template "http://sample.com/resource/{name}";
      rr:termType rr:IRI;
  ];
  rr:predicateObjectMap [
      rr:predicate  foaf:name;
      rr:objectMap [
        rml:reference "name";
        rr:termType rr:Literal;
      ]
    ];
   rr:predicateObjectMap [
    rr:predicate  onto:surface;
    rr:objectMap [
      rml:reference "features.surface.value";
      rr:termType rr:Literal ;
      rr:datatype xsd:double
    ]
  ];
```

Sample 3 mapping

```
<#TriplesMap2>
  rml:logicalSource [
      rml:source "./data/sample3/locati
      rml:referenceFormulation ql:JSONP
      rml:iterator "$.*"
    ];
  rr:subjectMap [
      rr:template "http://sample.com/re
      rr:termType rr:IRI;
  ];
  rr:predicateObjectMap [
      rr:predicate  foaf:name;
      rr:objectMap [
        rml:reference "name";
        rr:termType rr:Literal;
      ]
    ];
   rr:predicateObjectMap [
    rr:predicate  onto:surface;
    rr:objectMap [
      rml:reference "features.surface.v
      rr:termType rr:Literal ;
      rr:datatype xsd:double
    ]
  ];
```

```
 rr:predicateObjectMap [
    rr:predicate  onto:population;
    rr:objectMap [
      rml:reference
"features.population.amount";
      rr:termType rr:Literal ;
      rr:datatype xsd:double
    ]
  ];
  rr:predicateObjectMap [
    rr:predicate  geo:lat;
    rr:objectMap [
      rml:reference "location.latitude";
      rr:termType rr:Literal ;
      rr:datatype xsd:double
    ]
  ];
  rr:predicateObjectMap [
      rr:predicate  geo:long;
      rr:objectMap [
      rml:reference "location.longitude";
rr:termType rr:Literal ;
      rr:datatype xsd:double ]
    ].
```

- Sample2 – Working with JSONs
- Sample3 – Complex JSON structures I
- **Sample4 – Complex JSON structures II**



locations.json

```
[
  {
    "name": "Oneonta",
    "features": {
      "surface": {
        "value": 90.16,
        "units": "Km2"
      },
      "population": {
        "amount": "1,812",
        "date": 2018
      }
    },
    "location": {
      "latitude": "44.9625428",
      "longitude": "-74.4168577"
    }
  },
  …
```

```
  …,
  {
    "name": "Gainesvl",
    "features": {
      "surface": {
        "value": 167.2,
        "units": "Km2"
      },
      "population": [{
        "amount": "133,611",
        "date": 2020
      }]
    },
    "location": {
      "latitude": "29.6862705",
      "longitude": "-82.319746"
    }
  }
  …
]
```

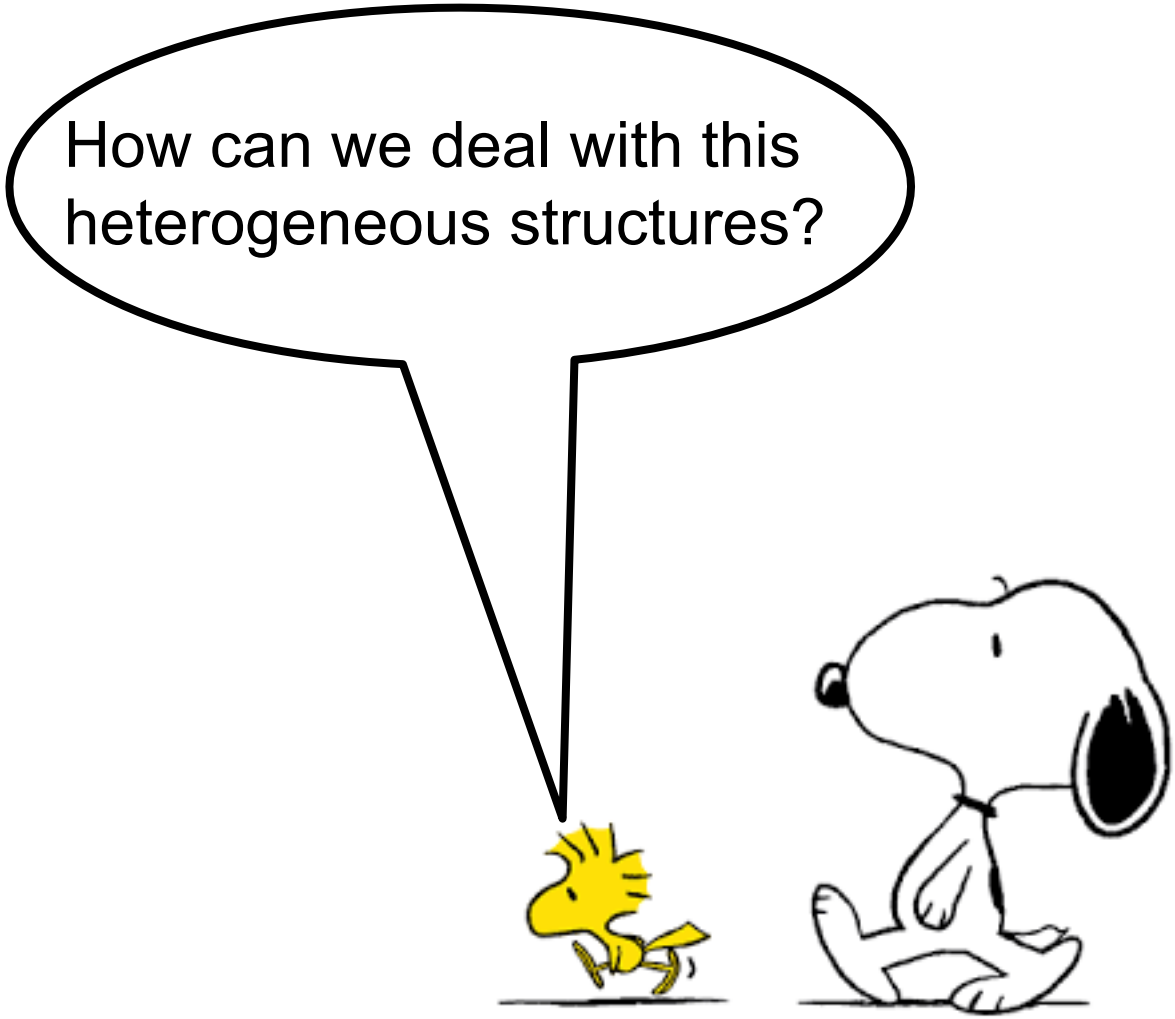
locations.json

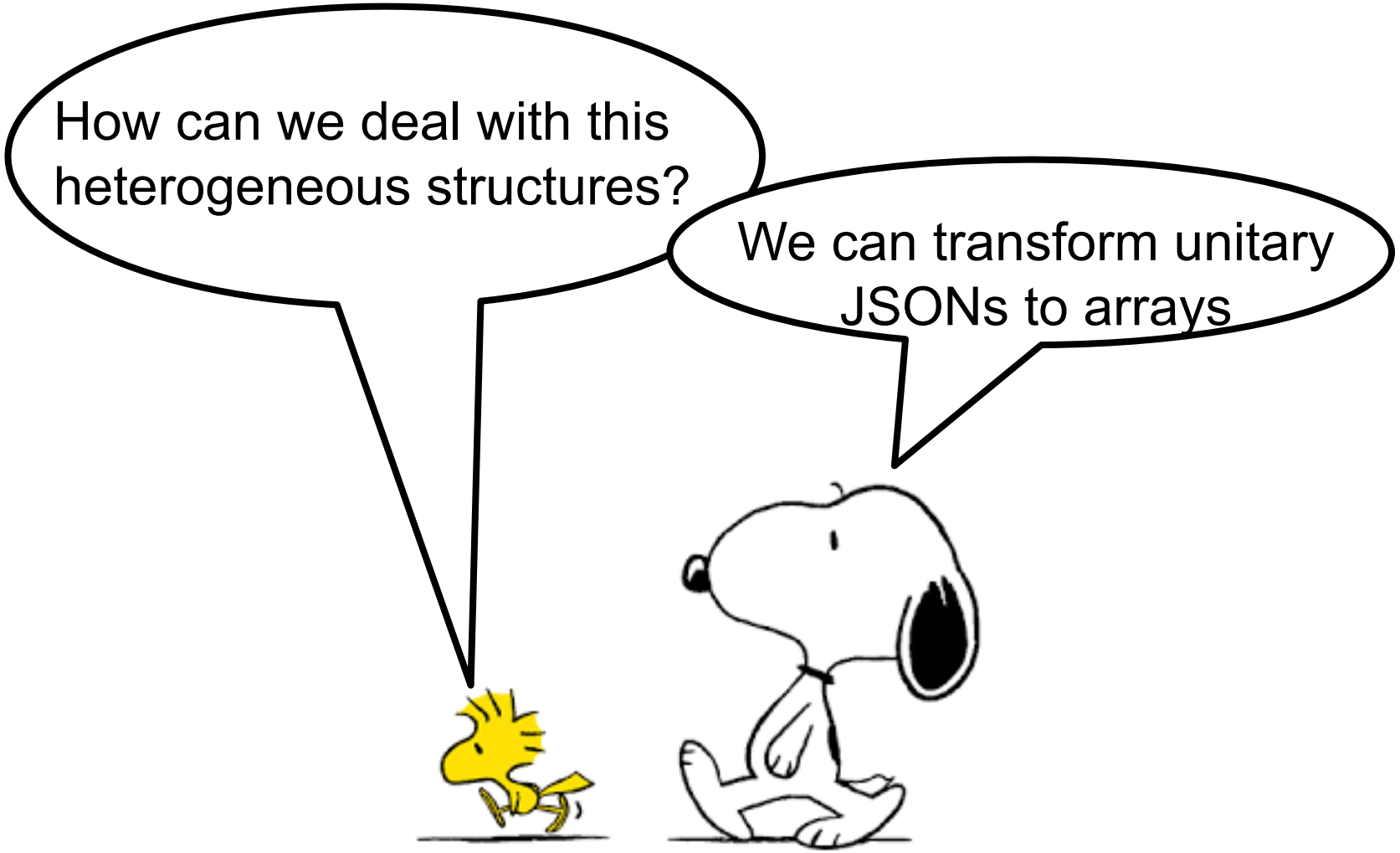
```
[
  {
    "name": "Oneonta",
    "features": {
      "surface": {
        "value": 90.16,
        "units": "Km2"
      },
      "population": {
        "amount": "1,812",
        "date": 2018
      }
    },
    "location": {
      "latitude": "44.9625428",
      "longitude": "-74.4168577"
    }
  },
  …
```

```
  …,
  {
    "name": "Gainesvl",
    "features": {
      "surface": {
        "value": 167.2,
        "units": "Km2"
      },
      "population": [{
        "amount": "133,611",
        "date": 2020
      }]
    },
    "location": {
      "latitude": "29.6862705",
      "longitude": "-82.319746"
    }
  }
  …
]
```

Sample 4 scenario

```
<http://sample.com/resource/P1>
        a foaf:Person ;
        foaf:name "Lily D." ;
        foaf:phone "585-610-2352" ;
        rdfs:seeAlso <https://dbpedia.org/page/Oneonta> ;
        foaf:based_near <https://sample.com/resource/Oneonta> .

<https://sample.com/resource/Oneonta>
        foaf:name "Oneonta" ;
        geo:lat "44.9625"
        geo:long "-74.4168" ;
        onto:population 1.812E3;
        onto:surface 90.16 .
```


How can we deal with this heterogeneous structures?


How can we deal with this heterogeneous structures?
We can transform unitary JSONs to arrays

locations.json

```
[
  {
    "name": "Oneonta",
    "features": {
      "surface": {
        "value": 90.16,
        "units": "Km2"
      },
      "population": [{
        "amount": "1,812",
        "date": 2018
      }]
    },
    "location": {
      "latitude": "44.9625428",
      "longitude": "-74.4168577"
    }
  },
  …
```

```
  …,
  {
    "name": "Gainesvl",
    "features": {
      "surface": {
        "value": 167.2,
        "units": "Km2"
      },
      "population": [{
        "amount": "133,611",
        "date": 2020
      }]
    },
    "location": {
      "latitude": "29.6862705",
      "longitude": "-82.319746"
    }
  }
  …
]
```

Mapping: unitary JSONs to arrays

```
<#TriplesMap3>
  rml:logicalSource [
      rml:source "./data/sample3/locations.json";
      rml:referenceFormulation ql:JSONPath;
      rml:iterator "$.*.population.[*]"
    ];
  rr:subjectMap [
      rr:template "http://sample.com/resource/pop/{name}/{amount}";
      rr:termType rr:IRI;
  ];
  rr:predicateObjectMap [
      rr:predicate  onto:population;
      rr:objectMap [
        rml:reference "amount";
        rr:termType rr:Literal;
      ]
  ].
```

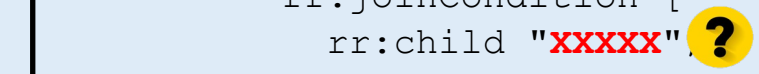
Mapping: unitary JSONs to arrays

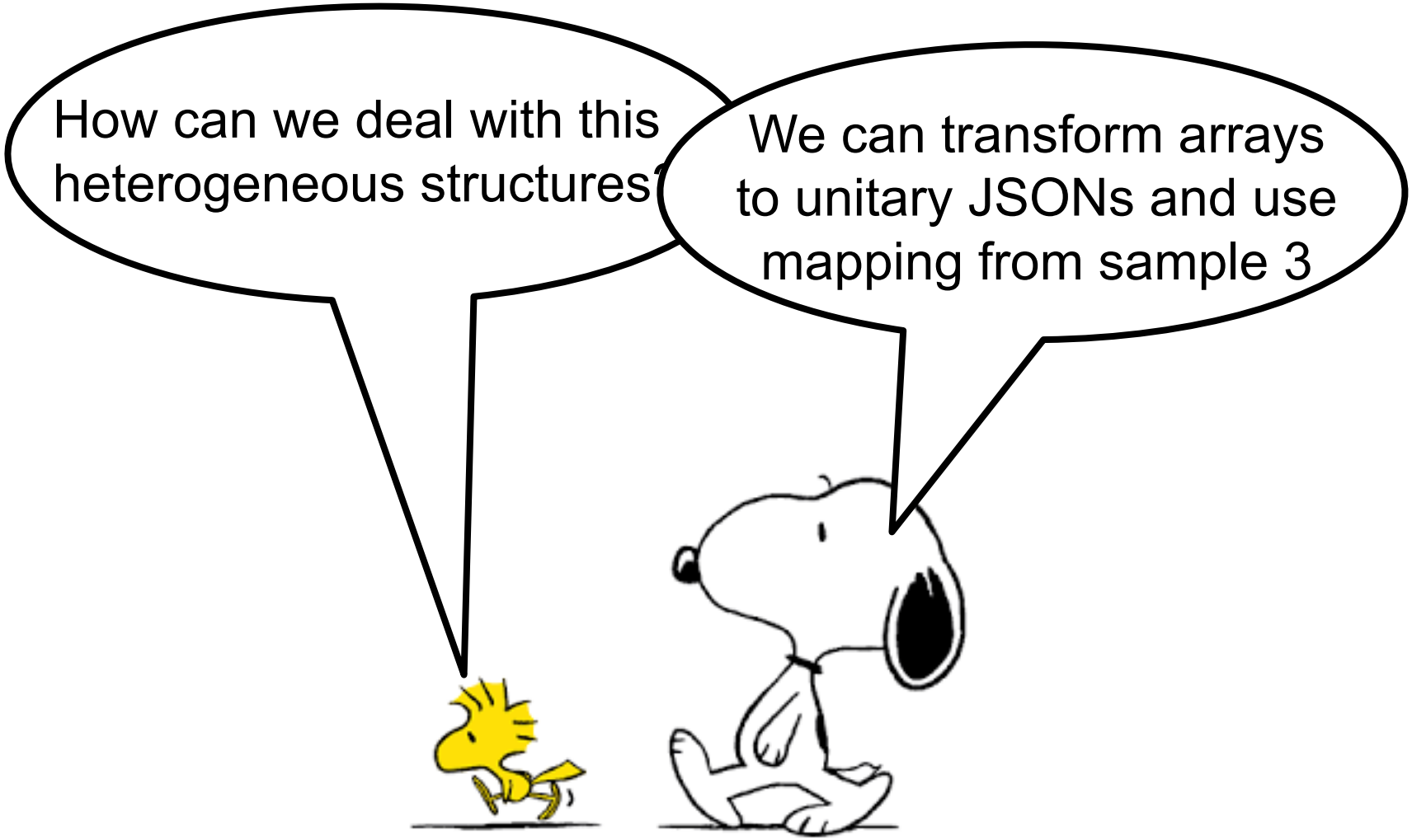
```
<http://sample.com/resource/P1>
        a foaf:Person ;
        foaf:name "Lily D." ;
        foaf:phone "585-610-2352" ;
        rdfs:seeAlso <https://dbpedia.org/page/Oneonta> ;
        foaf:based_near <https://sample.com/resource/Oneonta> .

<https://sample.com/resource/Oneonta>
        foaf:name "Oneonta" ;
        geo:lat "44.9625"
        geo:long "-74.4168" ;
        ? [ onto:population 1.812E3 ; ] ;
        onto:surface 90.16 .
```

Mapping: unitary JSONs to arrays

```
<#TriplesMap3>
  rml:logicalSource [
      rml:source "./data/sample3/locations.json";
      rml:referenceFormulation ql:JSONPath;
      rml:iterator "$.*.population.[*]"
    ];
  rr:subjectMap [
      rr:template "http://sample.com/resource/pop/{name}/{amount}";
      rr:termType rr:IRI;
  ];
  rr:predicateObjectMap [
      rr:predicate  onto:population;
      rr:objectMap [
        rml:reference "amount";
        rr:termType rr:Literal;
      ]
  ].
```

```
<#TriplesMap2>
    ...
    rr:predicateObjectMap [
        rr:predicate onto:hasPopulation;
        rr:objectMap [
          rr:parentTriplesMap <#TriplesMap3>;
          rr:joinCondition [
            rr:child "XXXXX";
            rr:parent "amount";
          ];
        ]
      ] .
```



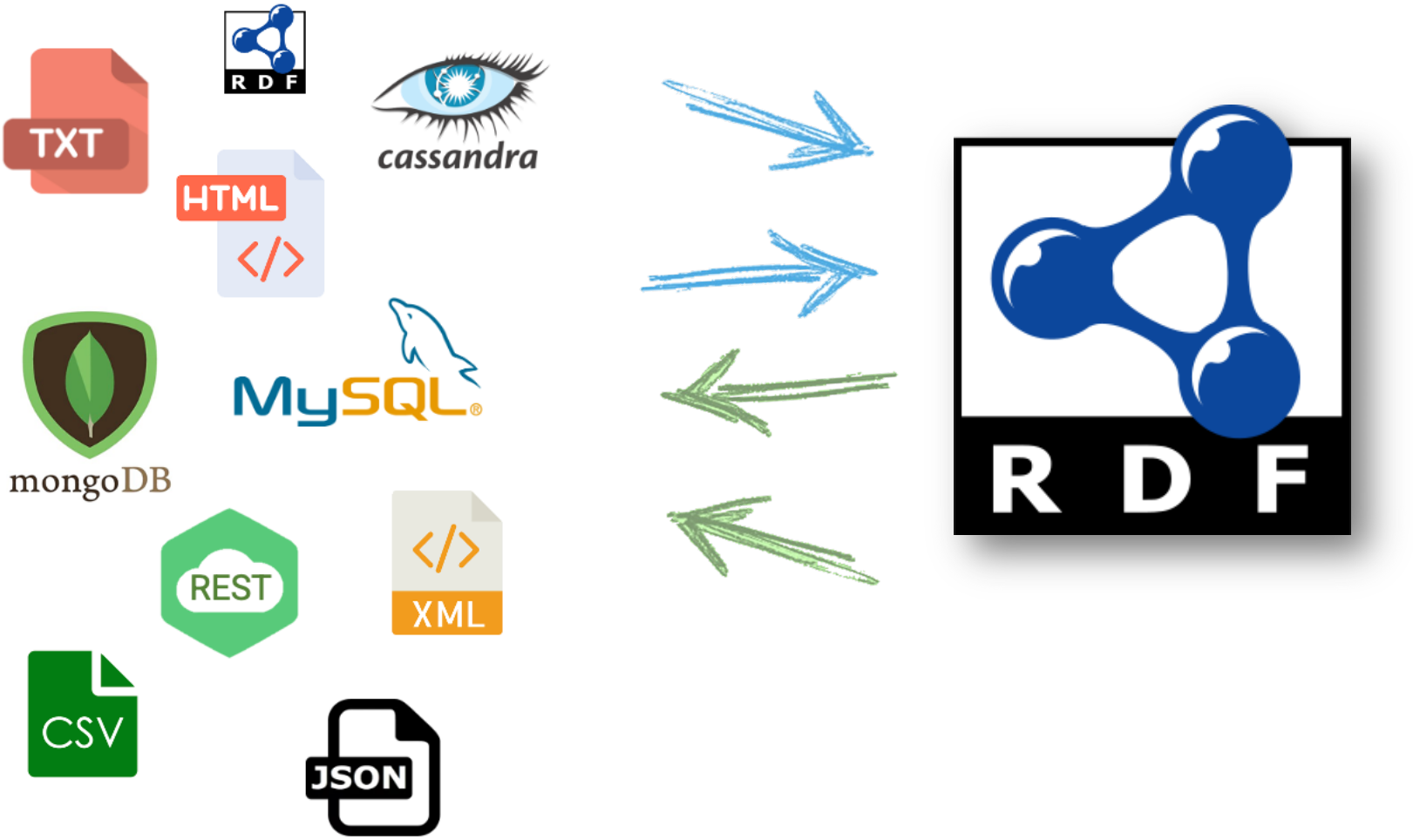

How can we deal with this heterogeneous structures?
We can transform arrays to unitary JSONs and use mapping from sample 3

A final though…

Index

- Generating and linking RDF data
- Using declarative mappings: RML mappings
- **Using procedural mappings: SloT mappings**

- **SloT is a procedural mapping language**
  - SloT is based on Freemarker, an HTML templating language
  - https://freemarker.apache.org/
  - *“Kind of a programming language”*


RDF
TXT
cassandra
HTML
MySQL
mongoDB
REST
XML
CSV
JSON
RDF

Sample 5 scenario

people.csv

ID	Name	Phone	City
1	Lily D.	585-610	Oneonta
…	…	…	…

location.csv

CName	latitude	longitude
Oneonta	44.9625	-74.4168
…	…	…

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

<http://sample.com/resource/P1>
        a foaf:Person ;
        foaf:name "Lily D." ;
        foaf:phone "585-610-2352" ;
        rdfs:seeAlso <https://dbpedia.org/page/Oneonta> ;
        foaf:based_near <https://sample.com/resource/Oneonta> .

<https://sample.com/resource/Oneonta>
        foaf:name "Oneonta" ;
        geo:lat "44.9625" ;
        geo:long "-74.4168" .
```

SloT: Data source description + filters (sample5)

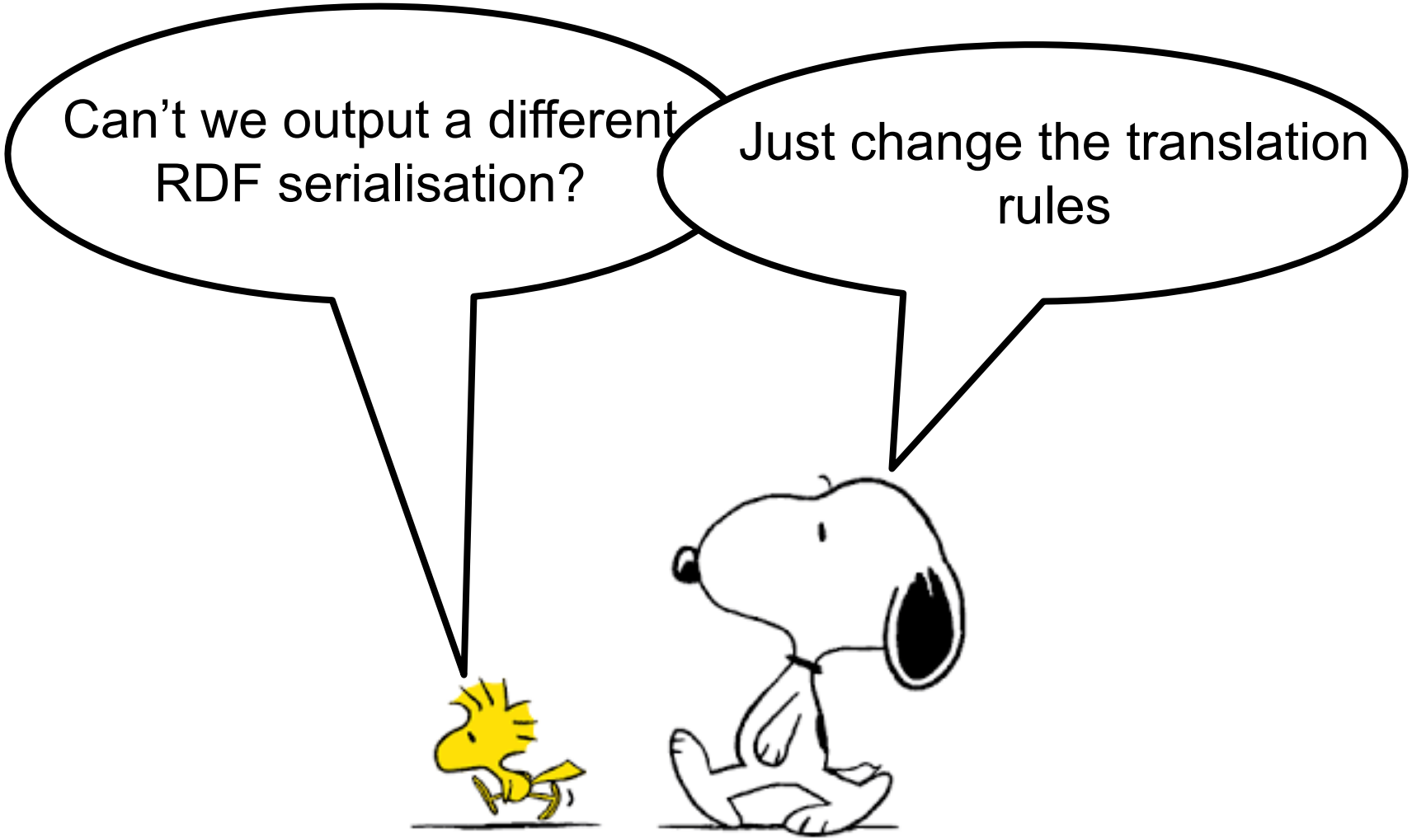
```
<#assign people=providers(type="FileProvider",
file="./data/sample5/people.csv")>
```

SloT: filters with translation rules (sample5)

```
<#assign people=providers(type="FileProvider",
file="./data/sample5/people.csv")>

   @prefix foaf: <http://xmlns.com/foaf/0.1/> .
   @prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
   @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

<#list people?split("\n") as person>
  <#if !person?is_first>
    <#assign cols=person?split(",")>
    <#if cols?size == 4>
    <http://sample.com/resource/P[=cols[0]]> a foaf:Person ;
        foaf:name [=cols[1]] ;
        foaf:phone "[=cols[2]]" ;
        rdfs:seeAlso <https://dbpedia.org/page/[=cols[3]]> .
    </#if>
  </#if>
</#list>
```

SloT: filters with translation rules (sample5)

```
<#assign people=providers(type="FileProvider",
file="./data/sample1/people.csv")>

@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

<#list people?split("\n") as person>
 <#if !person?is_first>
   <#assign cols=person?split(",")>
   <#if cols?size == 4>
    <#if cols[0]??>
      <http://sample.com/resource/P[=cols[0]]> a foaf:Person ;
          foaf:name [=cols[1]] ;
     <#if cols[2]??>  foaf:phone "[=cols[2]]" ; <#else> </#if>
          rdfs:seeAlso <https://dbpedia.org/page/[=cols[3]]> .
    </#if>
   </#if>
  </#if>
</#list>
```


Can’t we output a different RDF serialisation?
Just change the translation rules

Sample 5 scenario

```
[ {
    "@context": {
      "foaf": "http://xmlns.com/foaf/0.1/",
      "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
      "name": {
        "@id": "foaf:name",
        "@type": "xsd:string"
      },
      "phone": {
        "@id": "foaf:phone",
        "@type": "xsd:string"
      },
      "alt": {
        "@id": "rdfs:seeAlso",
        "@type": "@id"
      }
    },
    "@id": "http://sample.com/resource/P1",
    "@type": "foaf:Person",
    "name": "Lily Dickens",
    "phone": "585-610-2352",
    "alt": "https://dbpedia.org/page/Oneonta"
  }
  …]
```

SloT: filters with translation rules (sample5)

```
<#assign people=providers(type="FileProvider",
file="./data/sample5/people.csv")>
[ <#list people?trim?split("\n") as person>
  <#if !person?is_first>
  <#assign cols=person?split(",")>
  <#if cols?size == 4>
    {
      "@context" : { ... },
      "@id" : "http://sample.com/resource/P[=cols[0]]",
      "@type" : "foaf:Person",
      "name" : [=cols[1]],
      "phone" : "[=cols[2]]",
      "alt" : "https://dbpedia.org/page/[=cols[3]?trim]"
    }
    <#if !person?is_last>
        ,
    </#if>
    </#if>
   </#if>
</#list> ]
```

A final thought….

We can use these mappings for generating HTML+JSON-LD1.1 or HTML+RDFa

We can translate heterogeneous data into RDF with different representations, but also vicevera

Mapping for locations

- How is the mapping for locations?

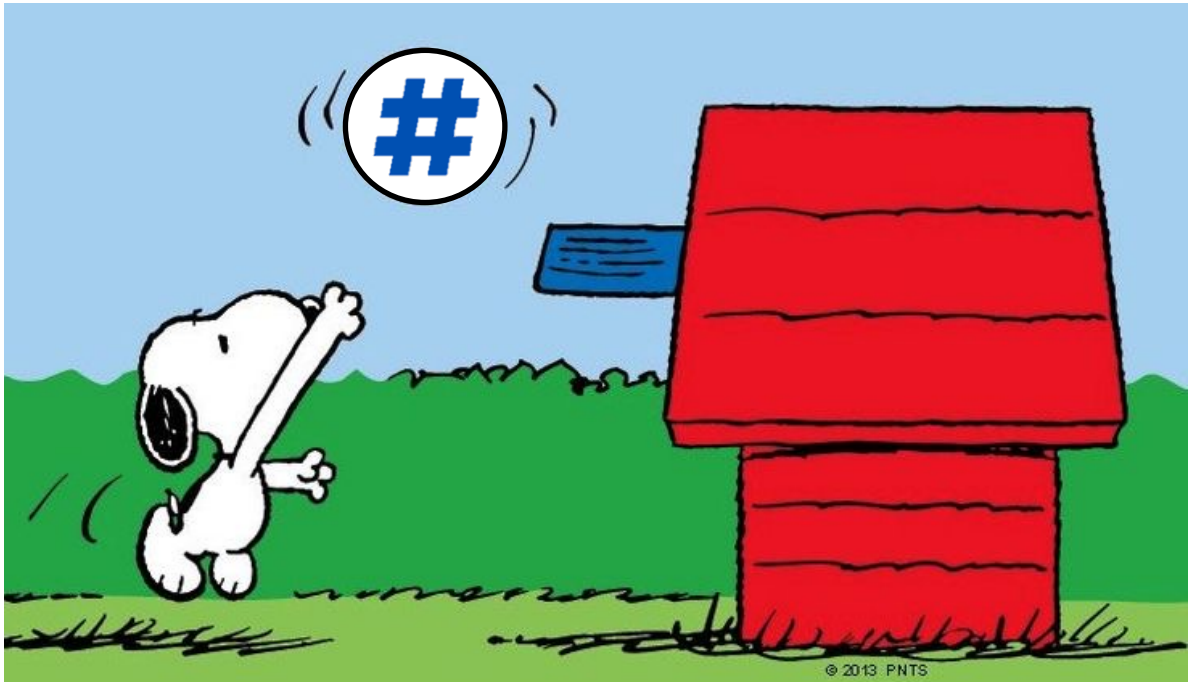
SloT: mapping for locations (sample5)

```
...

<#assign locations=providers(type="FileProvider",
file="./data/sample5/locations.csv")>

<#list locations?split("\n") as location>
    <#if !location?is_first>
        <#assign cols=location?split(",")>
        <#if cols?size == 3>

         <http://sample.com/resource/[=cols[0]]> foaf:name
"[=cols[0]]";
                geo:lat [=cols[1]?trim] ;
                geo:long [=cols[2]?trim] .
        </#if>
    </#if>
</#list>
```

SloT: linking (sample5)

```
...

<#list locations?split("\n") as location>
    <#if !location?is_first>
      <#assign locs=location?split(",")>
      <#if locs?size == 3>
        <#list people?split("\n") as person>
          <#if !person?is_first>
          <#assign pers=person?split(",")>
            <#if pers?size == 4>
              <#if pers[3] == locs[0]>
              <http://sample.com/resource/P[=pers[0]]>
foaf:based_near <http://sample.com/resource/[=locs[0]]> .
              </#if>
            </#if>
          </#if>
        </#list>
      </#if>
    </#if>
</#list>
```

- **Sample6 – Working with JSONs**
- Sample7 – Complex JSON structures II
- Sample8 – Linking with external sources (DBPedia)
- Sample9 – Actions
- Sample10 – Large data



Sample 6 scenario

people.csv

ID	Name	Phone	City
1	Lily D.	585-610	Oneonta
…	…	…	…

locations.json

```
[
 {
  "name": "Oneonta",
  "surface": "90.16",
  "population": "1812",
  "latitude": "44.9625428",
  "longitude": "-74.4168577"
 }, …
]
```

```
<http://sample.com/resource/P1>
        a foaf:Person ;
        foaf:name "Lily D." ;
        foaf:phone "585-610-2352" ;
        rdfs:seeAlso <https://dbpedia.org/page/Oneonta> ;
        foaf:based_near <https://sample.com/resource/Oneonta> .

<https://sample.com/resource/Oneonta>
        foaf:name "Oneonta" ;
        geo:lat "44.9625"
        geo:long "-74.4168" ;
        onto:population 1.812E3;
        onto:surface 90.16 .
```

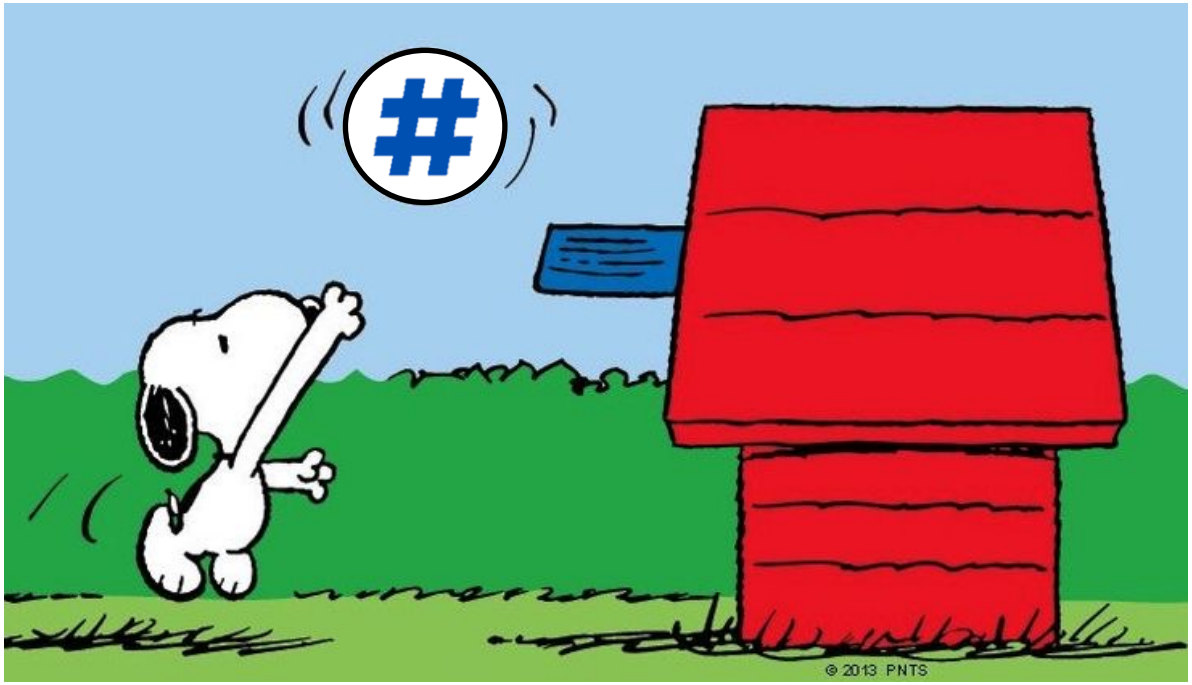
Sample 6 mapping

```
<#assign locations=providers(type="FileProvider",
file="./data/sample6/locations.json")>
<#assign jpath=handlers("JsonHandler")>
```

Sample 6 mapping

```
<#assign locations=providers(type="FileProvider",
file="./data/sample6/locations.json")>
<#assign jpath=handlers("JsonHandler")>

        @prefix foaf: <http://xmlns.com/foaf/0.1/> .
        @prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
        @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

<#list jpath.filter("$.[*]", locations) as location>
    <#assign name = jpath.filter("$.name", location)>
    <#assign lat = jpath.filter("$.latitude", location)>
    <#assign long = jpath.filter("$.longitude", location)>

    <#if name??>
        <http://sample.com/resource/[=name]> foaf:name "[=name]";
              geo:lat [=lat] ;
              geo:long [=long] .
    </#if>
</#list>
```

- Sample6 – Working with JSONs
- **Sample7 – Complex JSON structures II**
- Sample8 – Linking with external sources (DBPedia)
- Sample9 – Actions
- Sample10 – Large data



locations.json

```
[
  {
    "name": "Oneonta",
    "features": {
      "surface": {
        "value": 90.16,
        "units": "Km2"
      },
      "population": {
        "amount": "1,812",
        "date": 2018
      }
    },
    "location": {
      "latitude": "44.9625428",
      "longitude": "-74.4168577"
    }
  },
  …
```

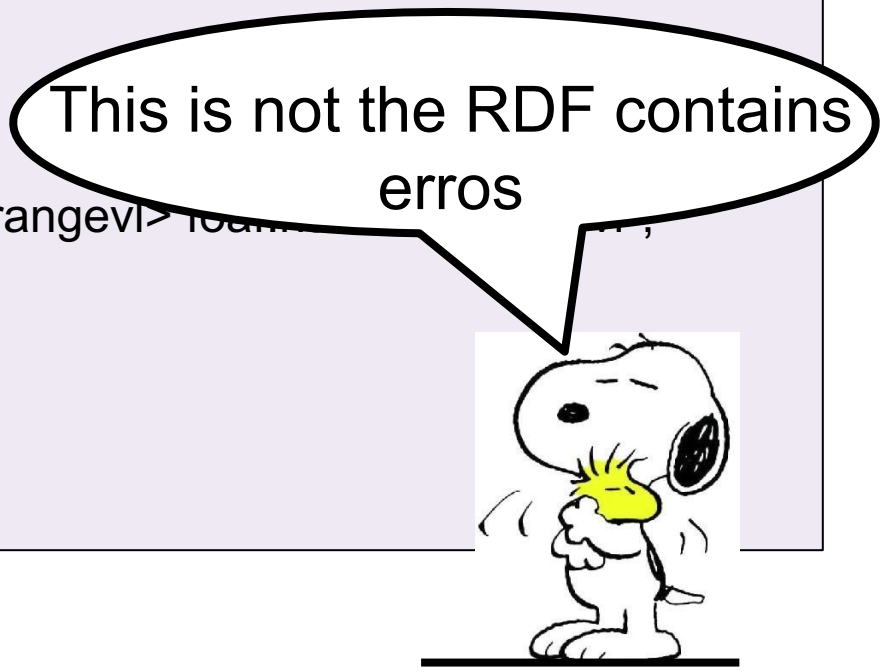
```
  …,
  {
    "name": "Gainesvl",
    "features": {
      "surface": {
        "value": 167.2,
        "units": "Km2"
      },
      "population": [{
        "amount": "133,611",
        "date": 2020
      }]
    },
    "location": {
      "latitude": "29.6862705",
      "longitude": "-82.319746"
    }
  }
  …
]
```

Sample 7 mapping

```
<#assign locations=providers(type="FileProvider",
file="./data/sample7/locations.json")>
<#assign jpath=handlers("JsonHandler")>

<#list jpath.filter("$.[*]", locations) as location>
    <#assign name = jpath.filter("$.name", location)>
    <#assign lat = jpath.filter("$.location.latitude", location)>
    <#assign long = jpath.filter("$.location.longitude", location)>
    <#assign surface = jpath.filter("$.features.surface.value", location)>
    <#assign population = jpath.filter("$.features.population", location)>

    <#if name??>
        <http://sample.com/resource/[=name]> foaf:name "[=name]";
            geo:lat [=lat] ;
            geo:long [=long] ;
            onto:surface [=surface] ;
            <#if population?is_sequence>
              <#list population as pop>
                <#assign value = jpath.filter("$.amount", pop)>
                onto:population [=value] .
              </#list>
            <#else>
            <#assign value = jpath.filter("$.amount", population)>
              onto:population [=value].
            </#if>
    </#if>
</#list>
```

Sample 7 output

```
<http://sample.com/resource/Oneonta> foaf:name "Oneonta";
        geo:lat 44.9625428 ;
        geo:long -74.4168577 ;
        onto:surface 90.16 ;
          onto:population 1,812 ;

      <http://sample.com/resource/Gainesvl> foaf:name "Gainesvl";
        geo:lat 29.6862705 ;
        geo:long -82.319746 ;
        onto:surface 167.2 ;
          onto:population 133,611 .

      <http://sample.com/resource/Orangevl> foaf:name "Orangevl";
        geo:lat 41.0816094 ;
        geo:long -76.4098259 ;
        onto:surface 33.75 ;
          onto:population 1,242 .
....
```



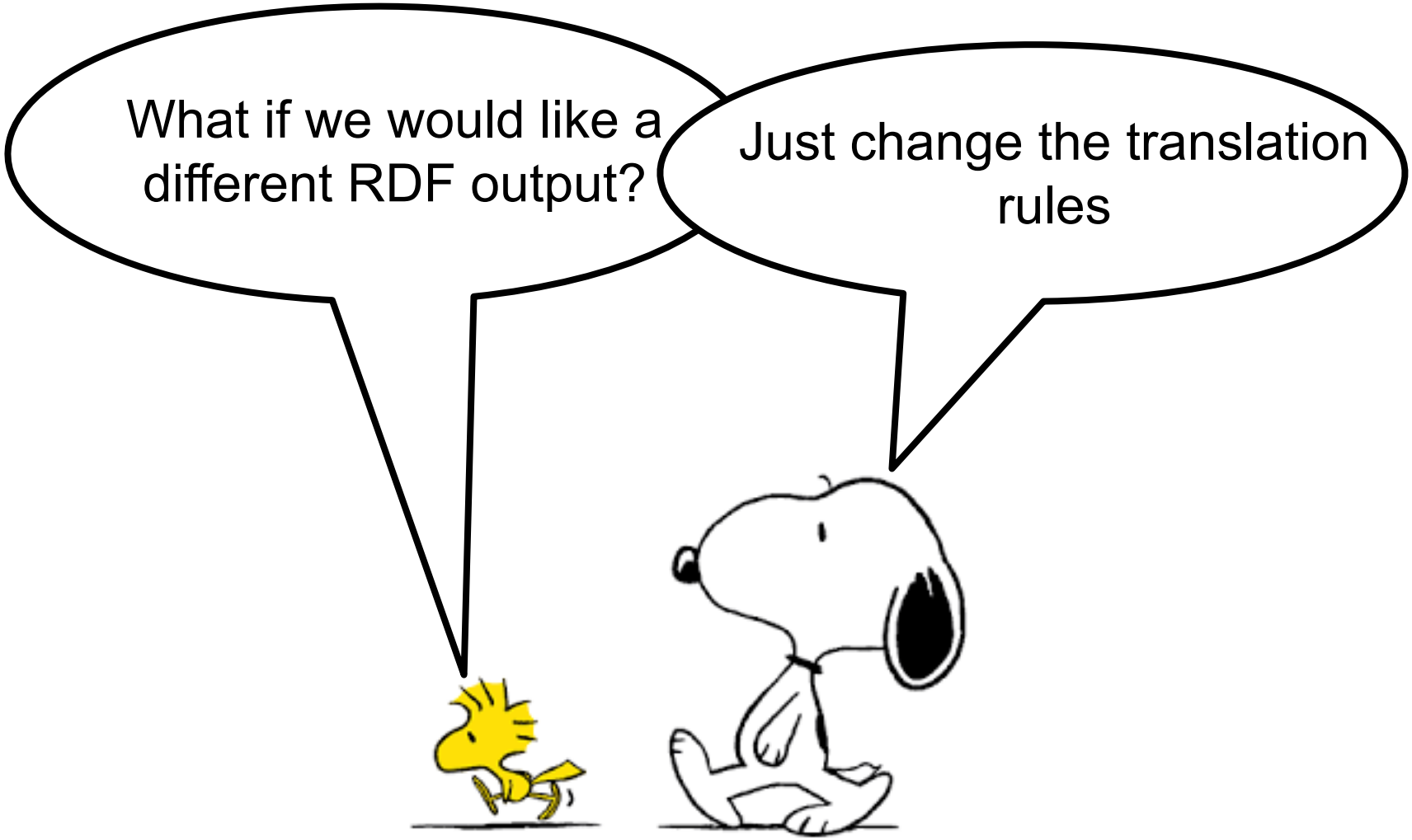
Sample 7 mapping

```
<#assign locations=providers(type="FileProvider",
file="./data/sample7/locations.json")>
<#assign jpath=handlers("JsonHandler")>

<#list jpath.filter("$.[*]", locations) as location>
    <#assign name = jpath.filter("$.name", location)>
    <#assign lat = jpath.filter("$.location.latitude", location)>
    <#assign long = jpath.filter("$.location.longitude", location)>
    <#assign surface = jpath.filter("$.features.surface.value", location)>
    <#assign population = jpath.filter("$.features.population", location)>

    <#if name??>
        <http://sample.com/resource/[=name]> foaf:name "[=name]";
            geo:lat [=lat] ;
            geo:long [=long] ;
            onto:surface [=surface] ;
            <#if population?is_sequence>
              <#list population as pop>
                <#assign value = jpath.filter("$.amount", pop)>
                onto:population [=value?replace(',', '', 'i')] .
              </#list>
            <#else>
            <#assign value = jpath.filter("$.amount", population)>
              onto:population [=value?replace(',', '', 'i')] .
            </#if>
    </#if>
</#list>
```

Sample 7 output

```
<http://sample.com/resource/Oneonta> foaf:name "Oneonta";
        geo:lat 44.9625428 ;
        geo:long -74.4168577 ;
        onto:surface 90.16 ;
          onto:population 1812 .

    <http://sample.com/resource/Gainesvl> foaf:name "Gainesvl";
        geo:lat 29.6862705 ;
        geo:long -82.319746 ;
        onto:surface 167.2 ;
          onto:population 133611  .

    <http://sample.com/resource/Orangevl> foaf:name "Orangevl";
        geo:lat 41.0816094 ;
        geo:long -76.4098259 ;
        onto:surface 33.75 ;
          onto:population 1242  .
....
```

This is not the RDF we wanted


What if we would like a different RDF output?
Just change the translation rules

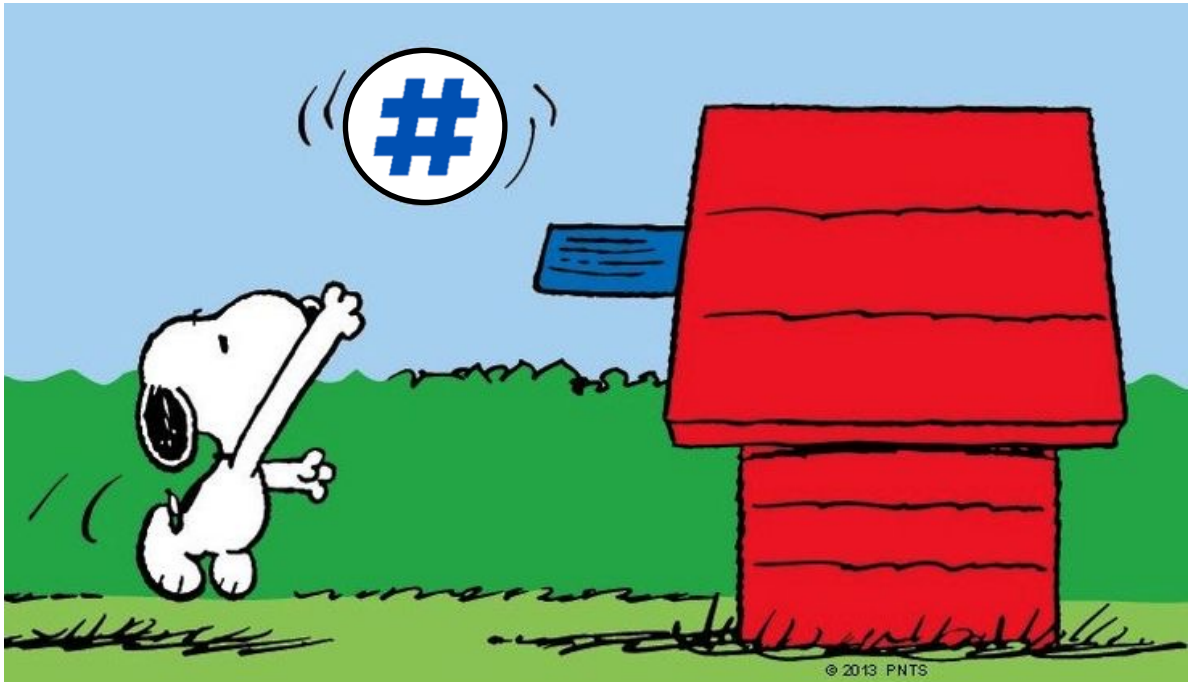
Sample 7 output

```
<http://sample.com/resource/Oneonta> foaf:name "Oneonta";
        geo:lat 44.9625428 ;
        geo:long -74.4168577 ;
        onto:hasFeatures [
              onto:surface [ onto:value 90.16; onto:units “Km2”] ;
              onto:population  [ onto:value 1812; onto:date 2018] ;
        ] .
```

Sample 7 mapping

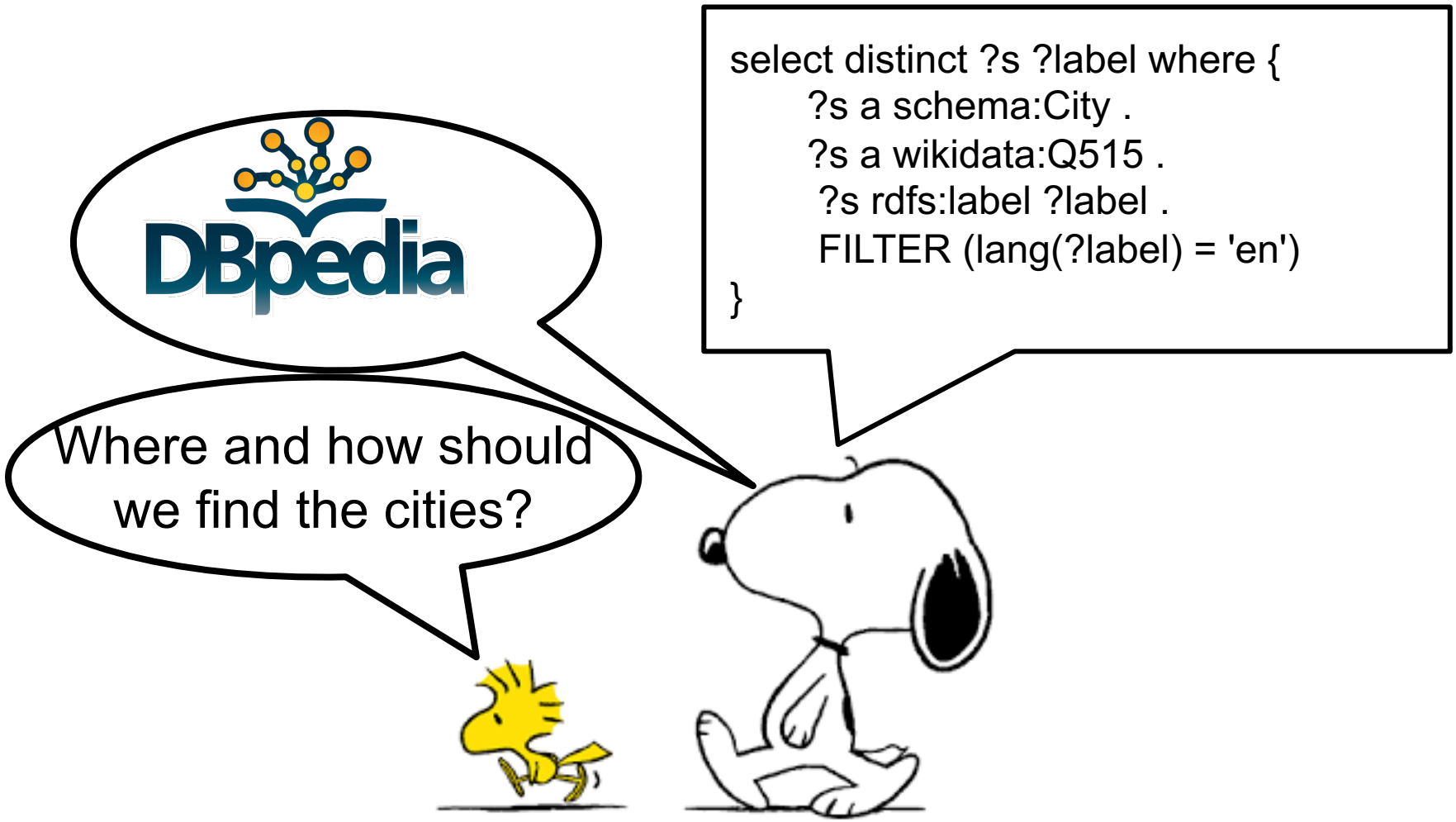
```
<#list jpath.filter("$.[*]", locations) as location>
  <#assign name = jpath.filter("$.name", location)>
  <#assign lat = jpath.filter("$.location.latitude", location)>
  <#assign long = jpath.filter("$.location.longitude", location)>
  <#assign surface = jpath.filter("$.features.surface.value", location)>
  <#assign units = jpath.filter("$.features.surface.units", location)>
  <#assign population = jpath.filter("$.features.population", location)>

  <#if name??>
    <http://sample.com/resource/[=name]> foaf:name "[=name]";
        geo:lat [=lat] ;
        geo:long [=long] ;
        onto:hasFeatures [
        onto:surface [ onto:value [=surface]; onto:units "[=units]" ] ;

        <#if population?is_sequence>
          <#list population as pop>
            <#assign value = jpath.filter("$.amount", pop)?replace(',', '', 'i')>
            <#assign date = jpath.filter("$.date", pop)>
            onto:population  [ onto:value [=value]; onto:date [=date] ] ;
          </#list>
        <#else>
          <#assign value = jpath.filter("$.amount", population)?replace(',', '', 'i')>
          <#assign date = jpath.filter("$.date", population)>
          onto:population  [ onto:value [=value]; onto:date [=date] ] ;
        </#if>
       ] .
    </#if>
</#list>
```

- Sample6 – Working with JSONs
- Sample7 – Complex JSON structures II
- **Sample8 – Linking with external sources (DBPedia)**
- Sample9 – Actions
- Sample10 – Large data



Let's find first the cities

- https://dbpedia.org/sparql



Sample 8 mapping

```
<#assign cities=providers(type="URLProvider", url="https://dbpedia.org/sparql?default-
graph-
uri=http%3A%2F%2Fdbpedia.org&query=select+distinct+%3Fs+%3Flabel+where+%7B+%0D%0A+++++
++%3Fs+a+schema%3ACity+.%0D%0A+++++++%3Fs+a+wikidata%3AQ515+.%0D%0A+++++++++%3Fs+rdfs%3
Alabel+%3Flabel+.%0D%0A++++++++FILTER+%28lang%28%3Flabel%29+%3D+%27en%27%29%0D%0A%7D+&
format=application%2Fsparql-
results%2Bjson&timeout=30000&signal_void=on&signal_unconnected=on")>

<#assign locations=providers(type="FileProvider",
file="./data/sample8/locations.json")>
<#assign jpath=handlers("JsonHandler")>
```

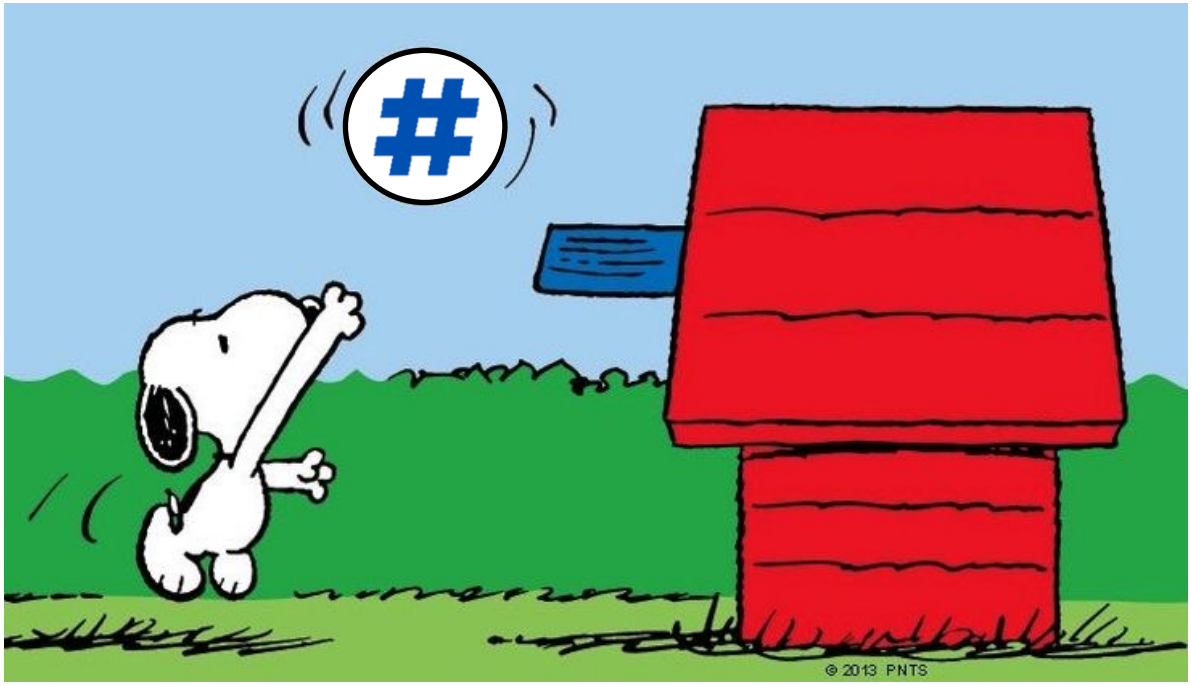
Sample 8 mapping

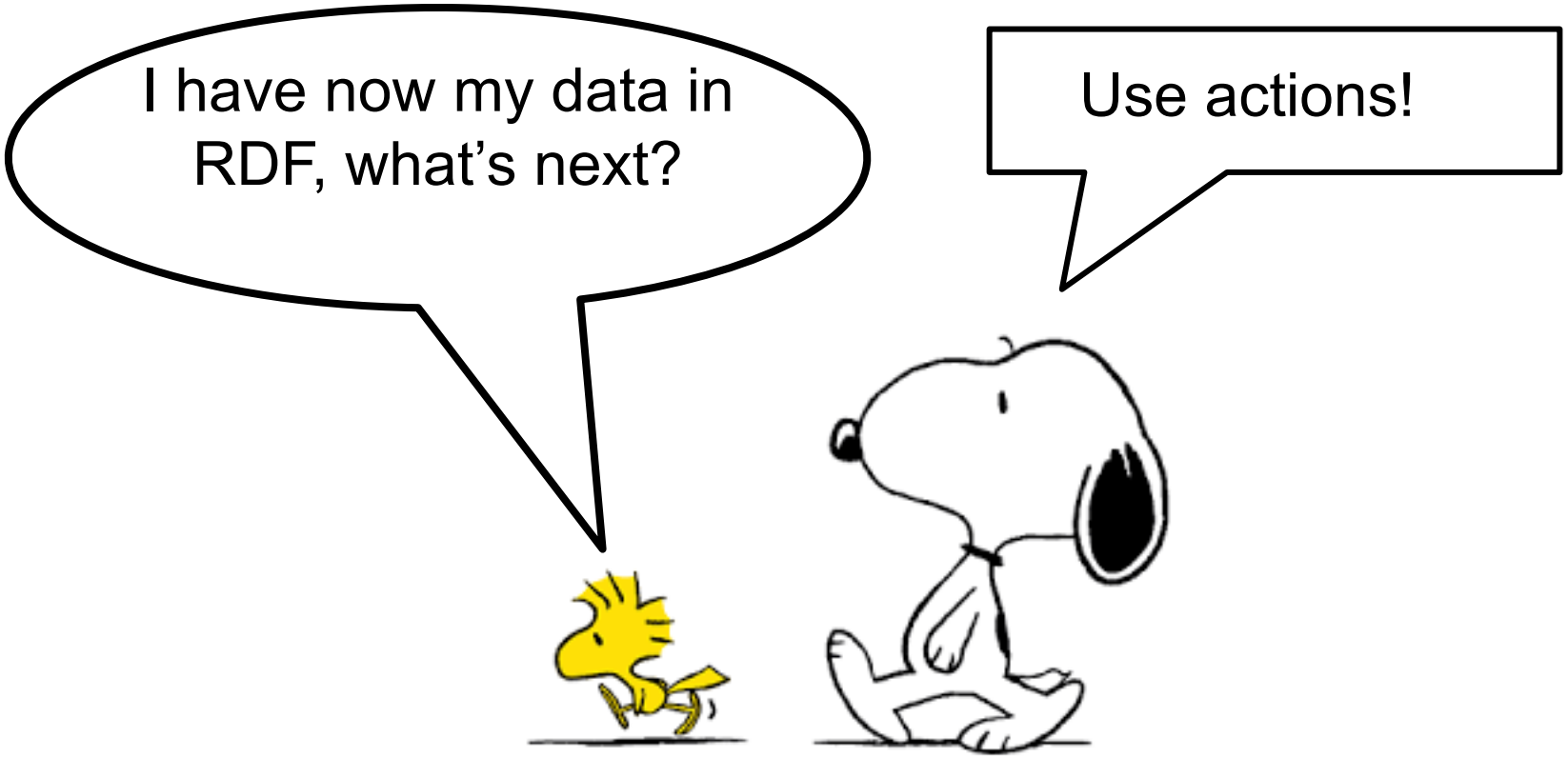
```
<#assign cities=providers(type="URLProvider", url="https://dbpedia.org/sparql?default-
graph-
uri=http%3A%2F%2Fdbpedia.org&query=select+distinct+%3Fs+%3Flabel+where+%7B+%0D%0A+++++
++%3Fs+a+schema%3ACity+.%0D%0A+++++++%3Fs+a+wikidata%3AQ515+.%0D%0A++++++++%3Fs+rdfs%3
Alabel+%3Flabel+.%0D%0A++++++++FILTER+%28lang%28%3Flabel%29+%3D+%27en%27%29%0D%0A%7D+&
format=application%2Fsparql-
results%2Bjson&timeout=30000&signal_void=on&signal_unconnected=on")>

<#assign locations=providers(type="FileProvider",
file="./data/sample8/locations.json")>
<#assign jpath=handlers("JsonHandler")>

<#list jpath.filter("$.[*]", locations) as location>
  <#assign name = jpath.filter("$.name", location)>
  <#list jpath.filter("$.results.bindings.[*]", cities) as city>
    <#assign uriDBpedia = jpath.filter("$.s.value", city)>
    <#assign nameDBpedia = jpath.filter("$.label.value", city)?replace(", .*",
"","r")>
    <#if name?contains(nameDBpedia)>
      <http://sample.com/resource/[=name]> owl:sameAs <[=uriDBpedia]> .
    </#if>
  </#list>
</#list>
```

Sample 8 result

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix onto: <http://fake-onto.com/ontology/def#>.



<http://sample.com/resource/Oneonta> foaf:name "Oneonta";
      geo:lat 44.9625428 ;
      geo:long -74.4168577 ;
      onto:surface 90.16 ;
      onto:population 1,812 .

 <http://sample.com/resource/Oneonta> owl:sameAs
<http://dbpedia.org/resource/Oneonta,_New_York> .
```

- Sample6 – Working with JSONs
- Sample7 – Complex JSON structures II
- Sample8 – Linking with external sources (DBPedia)
- **Sample9 – Actions**
- Sample10 – Large data

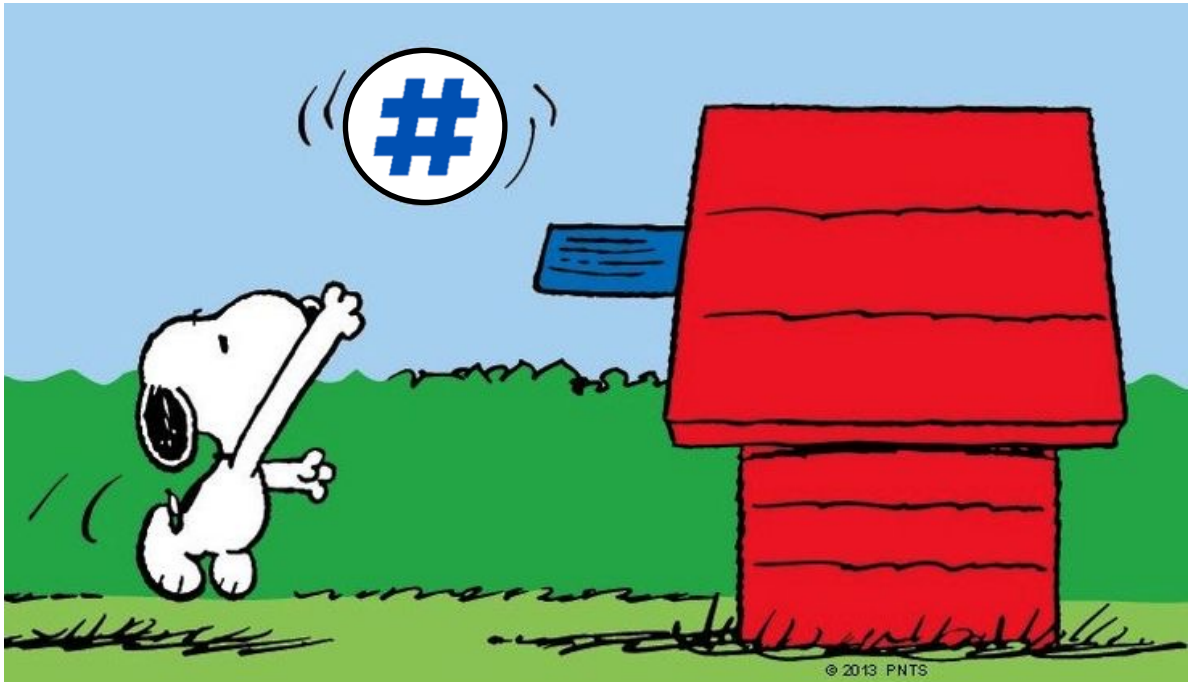



I have now my data in RDF, what's next?
Use actions!

Sample 9 mapping

```
<#assign sample1=providers(type="URLProvider",
url="http://localhost:4567/api/sample1/data")>


<#assign configuration= "{ \"url\": \"https://r56r4.mocklab.io/sparql\",\"method\":
\"POST\" }">


<@action type="HttpRequest" data=sample1 conf=configuration?eval; result>
  [=result]
</@action>
```

- Sample6 – Working with JSONs
- Sample7 – Complex JSON structures II
- Sample8 – Linking with external sources (DBPedia)
- Sample9 – Actions
- **Sample10 – Large data**



Sample 10

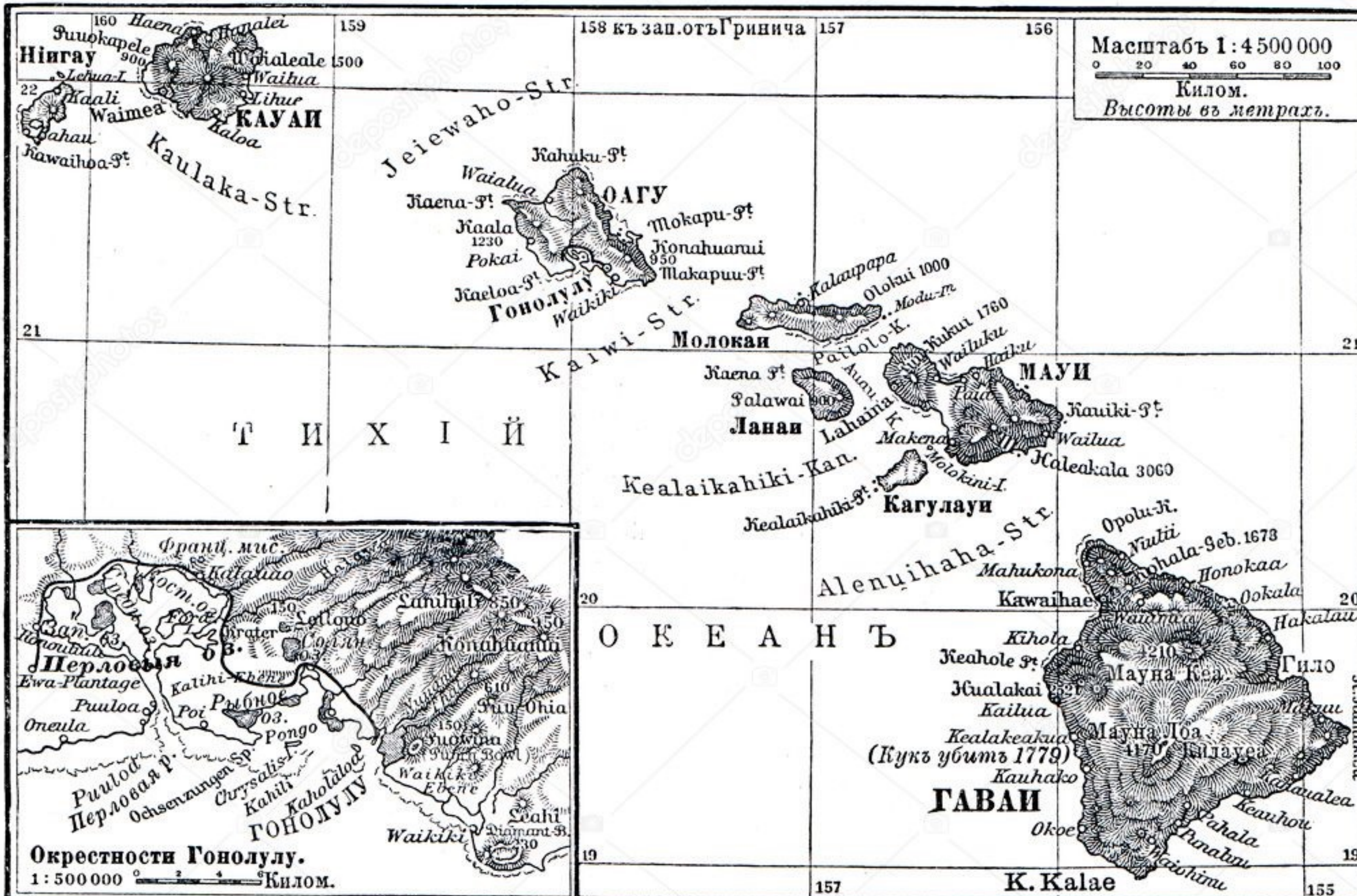
- Let's generate ~500.000 RDF triples from 225.000 CSV lines
- How much time would it take?

Conclusions

- The generation of RDF from other sources is the key-enabler for a richer exploitation of such data
- There are mapping languages for performing these translations
- The mapping languages feed tools that are able to process them

- There are no better or worst mappings, just mappings designed for different things


Q
A



158 къ зап. отъ Гринича
Масштабъ 1 : 4 500 000
Килом.
Высоты въ метрахъ.
Нiигау
КАУАИ
Waimea
Kaulaka-Str.
Jeiewaho-Str.
ОАГУ
Гонолулу
Kaiwi-Str.
Молокаи
Ланаи
МАУИ
Кагулауи
Kealaikahiki-Kan.
Alenuihaha-Str.
Haleakala 3060
ТИХIЙ
ОКЕАНЪ
ГАВАИ
Мауна Кеа
Мауна Лоа
Гило
(Кукъ убитъ 1779)
K. Kalae
Окрестности Гонолулу.
1 : 500 000 Килом.
Перловая р.
ГОНОЛУЛУ
Рыбное оз.