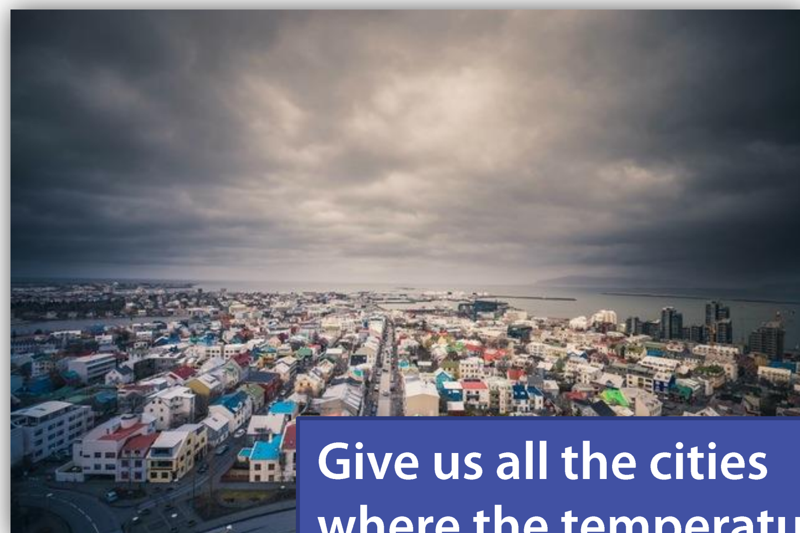
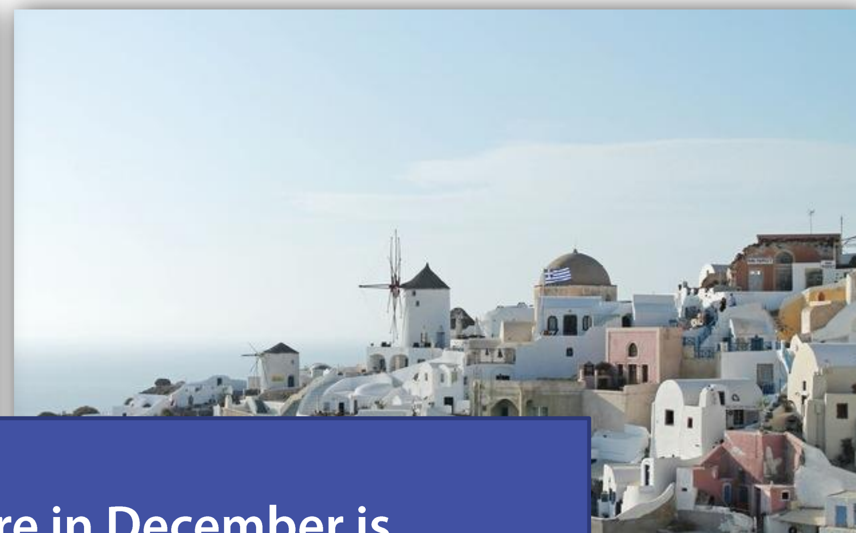
TECHNISCHE
UNIVERSITÄT
WIEN

Complementary Methods for Linked Data Enrichment

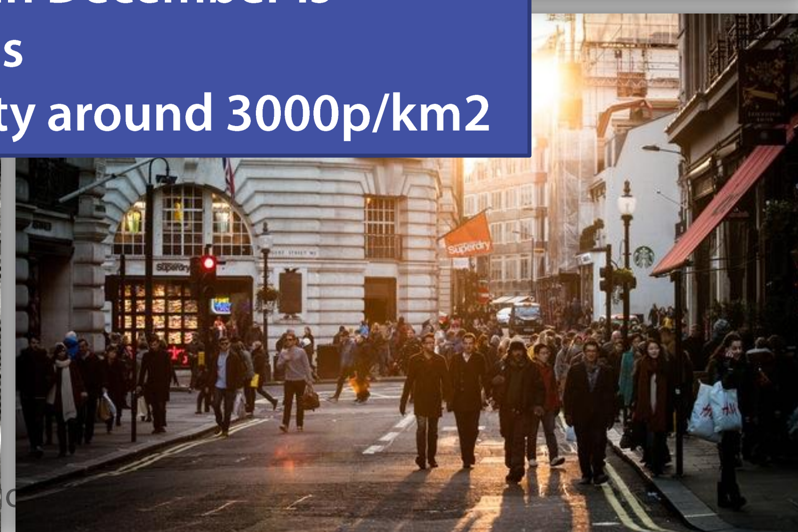
Rigorosum Stefan Bischof – 21.11.2017

Which city is the best?



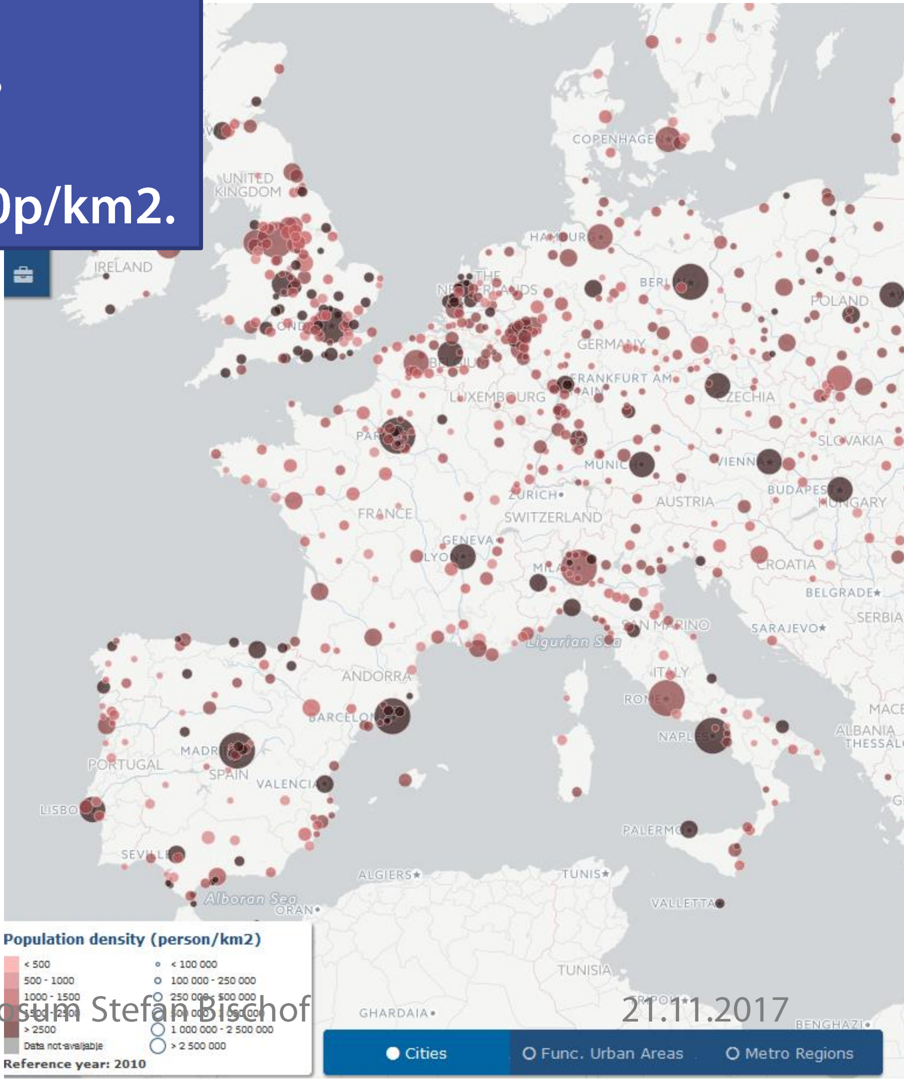


Give us all the cities
where the temperature in December is
above 20 degrees Celsius
with a population density around 3000p/km2



A Data Science approach!

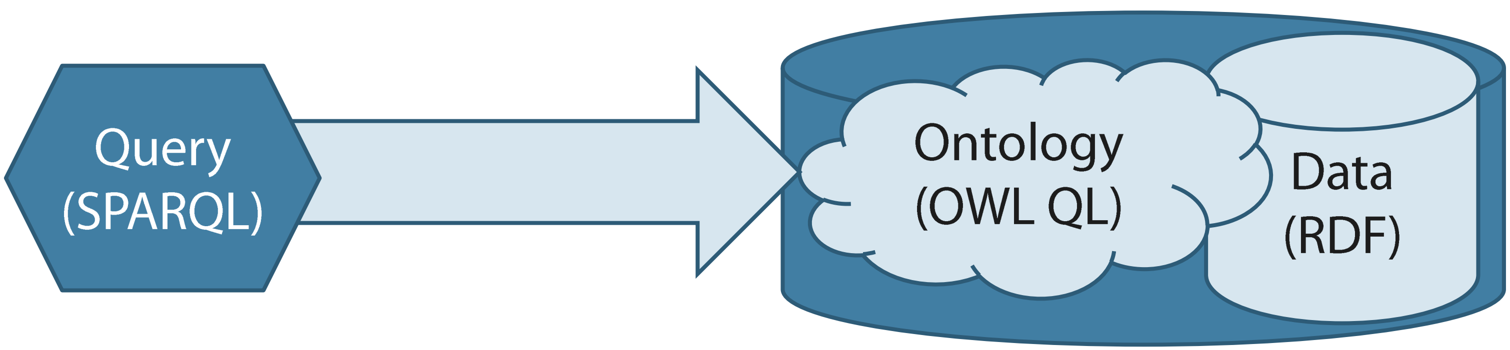
Give us all the cities
where the temperature in December is
above 20 degrees Celsius
with a population density around 3000p/km2.



- Data with global coverage
  - (Linked) Open Data
  - Resource Description Framework
  - Reasoning: Ontologies

Ontological Query Answering for RDF data

- Formulate queries using concepts from ontology (city)
- Standard rewriting approaches: ontology dependent, exponential size
- RDF triple stores or public SPARQL endpoints are different
  - Ontology and data are contained in the same graph
  - Query language is SPARQL instead of conjunctive queries



Give us all the cities where the temperature in December is above 24 degrees Celsius with a population density around 3000p/km2.

We need all the cities

RQ 1: Can we produce and effectively use **rewritings** of SPARQL queries which are **independent of the ontology** and **avoid the exponential blowup** of standard query rewriting techniques?

Schema-Agnostic Rewriting with SPARQL 1.1

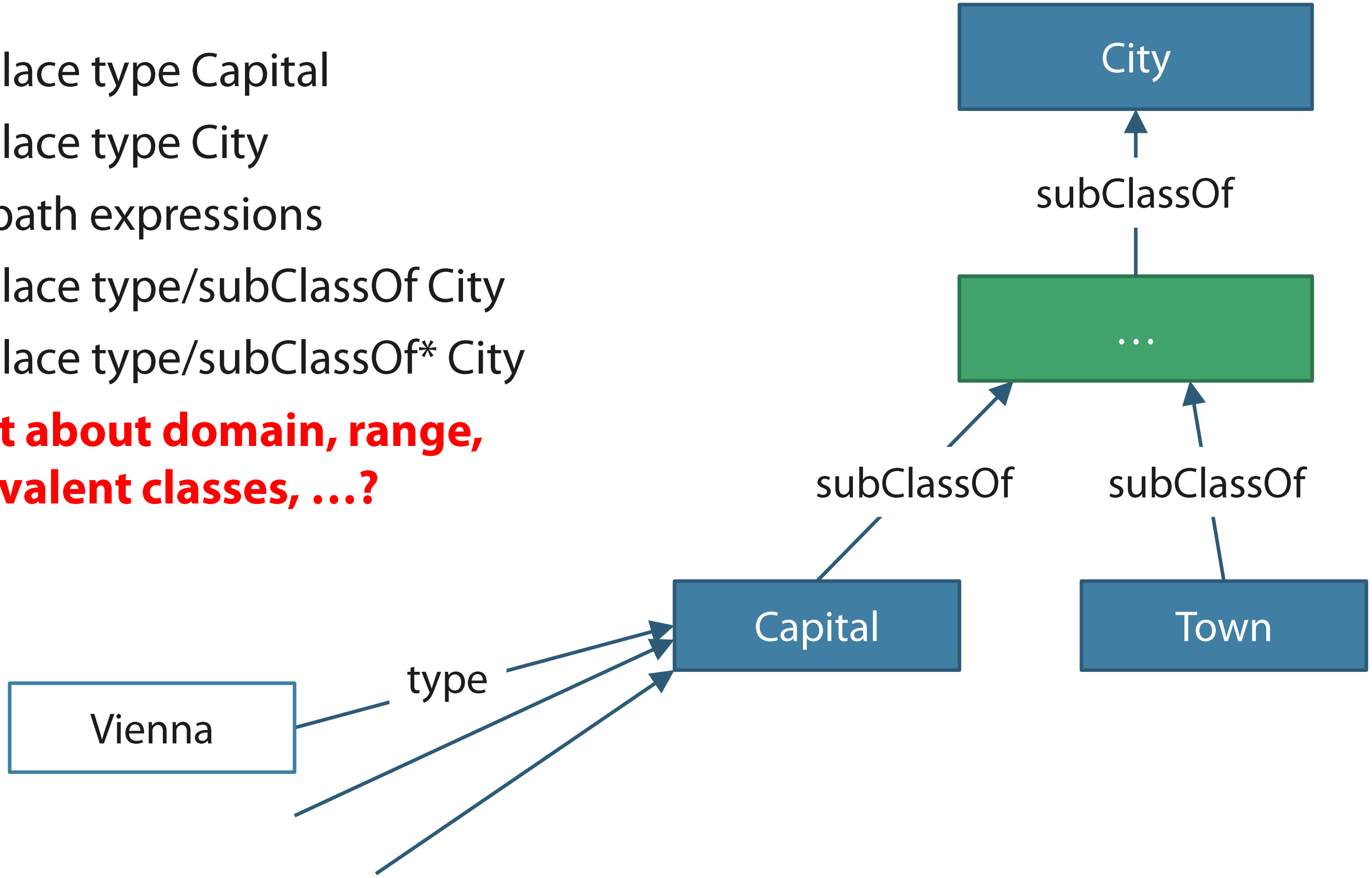
How to find all instances of a class?

- ?place type Capital
- ?place type City

Use path expressions

- ?place type/subClassOf City
- ?place type/subClassOf* City

What about domain, range, equivalent classes, …?



```
{ { x rdf:type/(((((rdfs:subClassOf|owl:equivalentClass)|^owl:equivalentClass)|((owl:intersectionOf/(rdf
:rest)*)/rdf:first))|((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(^owl:
onProperty|rdfs:domain)))|((((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^
owl:equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(owl:
inverseOf|^owl:inverseOf))/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/rdfs:range))* c } UNION
{ x ?P ?Y .
?P ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:inverseOf|^owl:
inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty))*)/(owl:
inverseOf|^owl:inverseOf))))*/(^owl:onProperty|rdfs:domain)/(((((rdfs:subClassOf|owl:
equivalentClass)|^owl:equivalentClass)|((owl:intersectionOf/(rdf:rest)*)/rdf:first))|((owl:
onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:
inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(^owl:onProperty|rdfs:domain)))
|((((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)
|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(owl:inverseOf|^owl:inverseOf))/(((
rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty))*)/rdfs:range))* c }
UNION
{ ?Y ?P x . ?P ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:
inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*/rdfs:range/(((((rdfs:subClassOf|owl:
equivalentClass)|^owl:equivalentClass)|((owl:intersectionOf/(rdf:rest)*)/rdf:first))|((owl:
onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:
inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(^owl:onProperty|rdfs:domain)))|((((
owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((
owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(owl:inverseOf|^owl:inverseOf))/(((
rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty))*)/rdfs:range))* c }
UNION
{ owl:Thing (((((rdfs:subClassOf|owl:equivalentClass)|^owl:equivalentClass)|((owl:intersectionOf/(rdf:
rest)*)/rdf:first))|((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(^owl:
onProperty|rdfs:domain)))|((((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^
owl:equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(owl:
inverseOf|^owl:inverseOf))/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/rdfs:range))* c } UNION
{ owl:topObjectProperty ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(
owl:inverseOf|^owl:inverseOf)))*/((^owl:onProperty|rdfs:domain)|rdfs:range) /(((((rdfs:
subClassOf|owl:equivalentClass)|^owl:equivalentClass)|((owl:intersectionOf/(rdf:rest)*)/rdf:first
))|((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)
|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(^owl:onProperty|rdfs:domain)))|((((
owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((
owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(owl:inverseOf|^owl:inverseOf))/(((
rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty))*)/rdfs:range))* c } }
```

Possible, but complicated . . .

- Rewriting of x type C
- Complete for OWL QL profile
- Constant size

- We can also write queries to answer
  - Is the ontology consistent?
  - Is the class A consistent?
  - Does the ontology entail A subClassOf B ?
  - Does the ontology entail R subPropertyOf S ?
  - Does the ontology entail c R d ?

Can we make this work in pratice?

```
{ { x rdf:type/(((((rdfs:subClassOf|owl:equivalentClass)|^owl:equivalentClass)|((owl:intersectionOf/(rdf
:rest)*)/rdf:first))|((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(^owl:
onProperty|rdfs:domain)))|((((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^
owl:equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(owl:
inverseOf|^owl:inverseOf))/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/rdfs:range))* c } UNION
{ x ?P ?Y .
?P ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:inverseOf|^owl:
inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty))*)/(owl:
inverseOf|^owl:inverseOf))))*/(^owl:onProperty|rdfs:domain)/(((((rdfs:subClassOf|owl:
equivalentClass)|^owl:equivalentClass)|((owl:intersectionOf/(rdf:rest)*)/rdf:first))|((owl:
onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:
inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(^owl:onProperty|rdfs:domain)))
|((((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)
|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(owl:inverseOf|^owl:inverseOf))/(((
rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty))*)/rdfs:range))* c }
UNION
{ ?Y ?P x . ?P ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:
inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*/rdfs:range/(((((rdfs:subClassOf|owl:
equivalentClass)|^owl:equivalentClass)|((owl:intersectionOf/(rdf:rest)*)/rdf:first))|((owl:
onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:
inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(^owl:onProperty|rdfs:domain)))|((((
owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((
owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(owl:inverseOf|^owl:inverseOf))/(((
rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty))*)/rdfs:range))* c }
UNION
{ owl:Thing (((((rdfs:subClassOf|owl:equivalentClass)|^owl:equivalentClass)|((owl:intersectionOf/(rdf:
rest)*)/rdf:first))|((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(^owl:
onProperty|rdfs:domain)))|((((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^
owl:equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(owl:
inverseOf|^owl:inverseOf))/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/rdfs:range))* c } UNION
{ owl:topObjectProperty ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(
owl:inverseOf|^owl:inverseOf)))*/((^owl:onProperty|rdfs:domain)|rdfs:range) /(((((rdfs:
subClassOf|owl:equivalentClass)|^owl:equivalentClass)|((owl:intersectionOf/(rdf:rest)*)/rdf:first
))|((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)
|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(^owl:onProperty|rdfs:domain)))|((((
owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((
owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(owl:inverseOf|^owl:inverseOf))/(((
rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty))*)/rdfs:range))* c } }
```

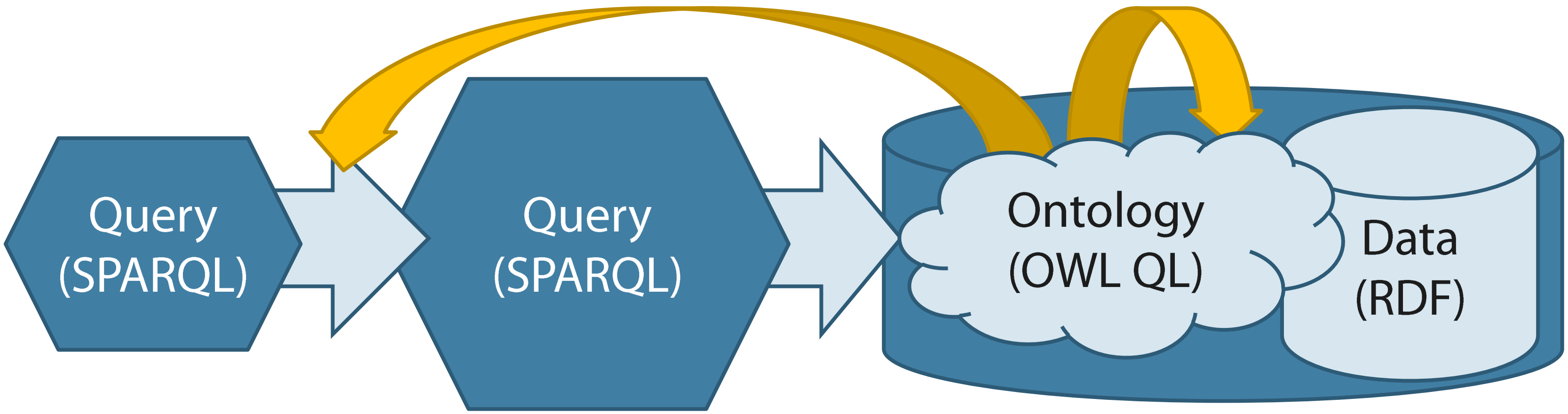
Algebraic Optimization

```
{ { ?V
(((((rdfs:subClassOf|owl:equivalentClass)|^owl:equivalentClass)|((owl:intersectionOf/(rdf:rest)*)/
rdf:first))|((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(^owl:
onProperty|rdfs:domain)))|((((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^
owl:equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(owl:
inverseOf|^owl:inverseOf))/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/rdfs:range))* c } .
{ { x rdf:type ?V} UNION
{ x ?P _:b0 .?P ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:
inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*/(^owl:onProperty|rdfs:domain) ?V }
UNION
{ _:b1 ?P x . ?P ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:
inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*/rdfs:range ?V } UNION
{ BIND(owl:Thing AS ?V) } UNION
{ owl:topObjectProperty ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty)|(owl:inverseOf|^owl:inverseOf)))*/((^owl:onProperty|rdfs:domain)|rdfs
:range) ?V } } }
```

- Evaluate common sub-paths only **once**
- Also: Apply standard query optimization

What about using some information from the ontology?

Optimization: Use some info from ontology




```
{ { ?V
(((((rdfs:subClassOf|owl:equivalentClass)|^owl:equivalentClass)|((owl:intersectionOf/(rdf:rest)*)/
rdf:first))|((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(^owl:
onProperty|rdfs:domain)))|((((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^
owl:equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(owl:
inverseOf|^owl:inverseOf))/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/rdfs:range))* c } .
{ { x rdf:type ?V} UNION
{ x ?P _:b0 .?P ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:
inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*/(^owl:onProperty|rdfs:domain) ?V }
UNION
{ _:b1 ?P x . ?P ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:
inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*/rdfs:range ?V } UNION
{ BIND(owl:Thing AS ?V) } UNION
{ owl:topObjectProperty ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty)|(owl:inverseOf|^owl:inverseOf)))*/((^owl:onProperty|rdfs:domain)|rdfs
:range) ?V } } }
```

Consider some information from the ontology

- Remove unused properties from the paths (OI)
- Materialize the paths as predicates to the triple stores (OM)

Optimization: Use some info from ontology
Remove irrelevant properties from the path

- An IRI not occuring in the RDF graph can be removed from the path
- Example for a graph with no equivalentClass:
  - x type/subClassOf*/equivalentClass* C -> x type/subClassOf* C

Rewriting rules

$$\boxed{\bot^* \rightarrow \epsilon}$$
$$\epsilon^* \rightarrow \epsilon$$
$$\hat{}\bot \rightarrow \bot$$
$$\hat{}\epsilon \rightarrow \epsilon$$
$$p_1 \mid \bot \mid p_2 \rightarrow p_1 \mid p_2$$
$$p_1 \mid \epsilon \mid p_2 \mid \epsilon \mid p_3 \rightarrow p_1 \mid \epsilon \mid p_2 \mid p_3$$
$$p_1 \;/\; \bot \;/\; p_2 \rightarrow \bot$$
$$\boxed{p_1 \;/\; \epsilon \;/\; p_2 \rightarrow p_1 \;/\; p_2}$$

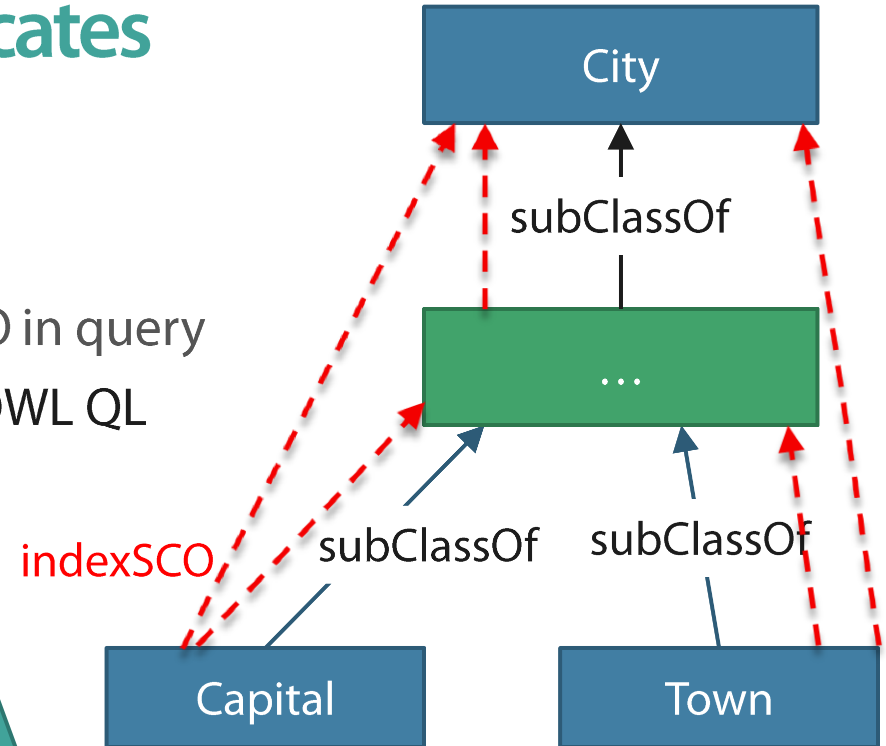
```
{ { ?V
(((((rdfs:subClassOf|owl:equivalentClass)|^owl:equivalentClass)|((owl:intersectionOf/(rdf:rest)*)/
rdf:first))|((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(^owl:
onProperty|rdfs:domain)))|((((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^
owl:equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(owl:
inverseOf|^owl:inverseOf))/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/rdfs:range))* c } .
{ { x rdf:type ?V} UNION
{ x ?P _:b0 .?P ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:
inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*/(^owl:onProperty|rdfs:domain) ?V }
UNION
{ _:b1 ?P x . ?P ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:
inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*/rdfs:range ?V } UNION
{ BIND(owl:Thing AS ?V) } UNION
{ owl:topObjectProperty ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty)|(owl:inverseOf|^owl:inverseOf)))*/((^owl:onProperty|rdfs:domain)|rdfs
:range) ?V } } }
```

```
{ { ?V
((((rdfs:subClassOf|owl:equivalentClass)|^owl:equivalentClass)|((owl:intersectionOf/(rdf
:rest)*)/rdf:first))|((owl:onProperty/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl
:equivalentProperty))*)/(^owl:onProperty|rdfs:domain)))* c }
{ { x rdf:type ?V} UNION
{ x ?P _:b0 . ?P (((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty))*/(^
owl:onProperty|rdfs:domain) ?V } UNION
{ _:b1 ?P x . ?P (((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty))*/
rdfs:range ?V } UNION
{ BIND(owl:Thing AS ?V)} UNION
{ owl:topObjectProperty (((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*/((^owl:onProperty|rdfs:domain)|rdfs:range) ?V } } }
```

Optimization: Use some info from ontology

Materialize paths as predicates

- Example:
  - Add indexSCO for subClassOf+
  - Replace subClassOf+ by indexSCO in query
- Only 6 distinct paths necessary for OWL QL
  - Makes approach feasible

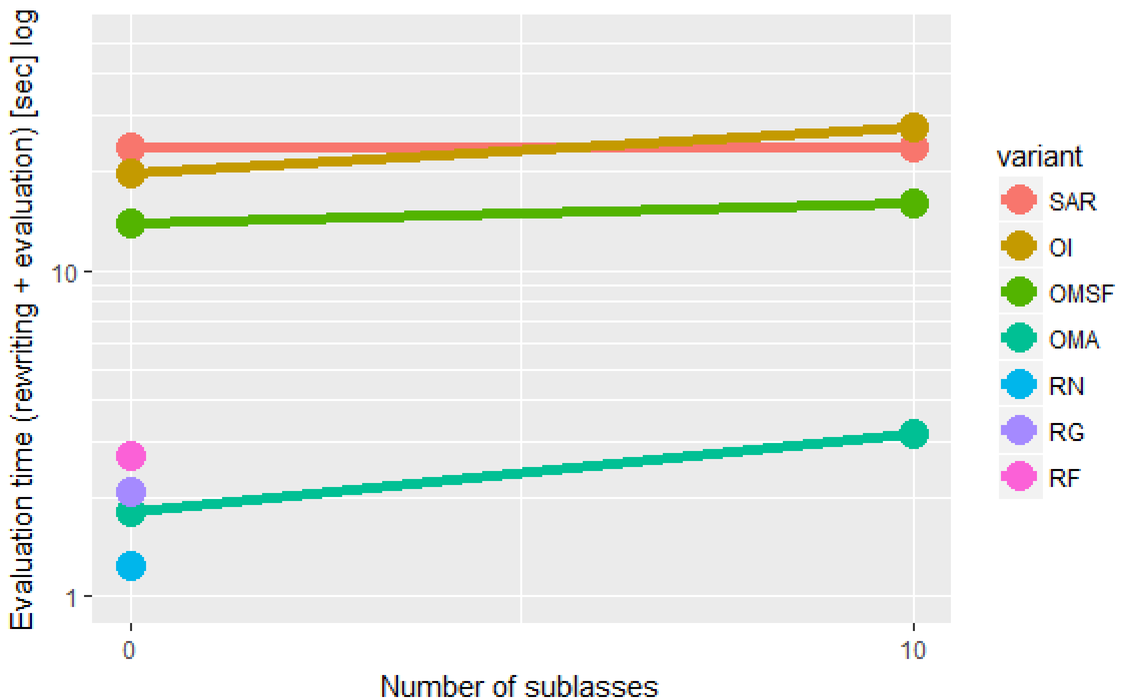



```
{ { ?V
(((((rdfs:subClassOf|owl:equivalentClass)|^owl:equivalentClass)|((owl:intersectionOf/(rdf:rest)*)/
rdf:first))|((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(^owl:
onProperty|rdfs:domain)))|((((owl:onProperty/((((rdfs:subPropertyOf|owl:equivalentProperty)|^
owl:equivalentProperty)|(((owl:inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:
equivalentProperty)|^owl:equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*)/(owl:
inverseOf|^owl:inverseOf))/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/rdfs:range))* c } .
{ { x rdf:type ?V} UNION
{ x ?P _:b0 .?P ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:
inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*/(^owl:onProperty|rdfs:domain) ?V }
UNION
{ _:b1 ?P x . ?P ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:equivalentProperty)|(((owl:
inverseOf|^owl:inverseOf)/(((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty))*)/(owl:inverseOf|^owl:inverseOf))))*/rdfs:range ?V } UNION
{ BIND(owl:Thing AS ?V) } UNION
{ owl:topObjectProperty ((((rdfs:subPropertyOf|owl:equivalentProperty)|^owl:
equivalentProperty)|(owl:inverseOf|^owl:inverseOf)))*/((^owl:onProperty|rdfs:domain)|rdfs
:range) ?V } } }
```

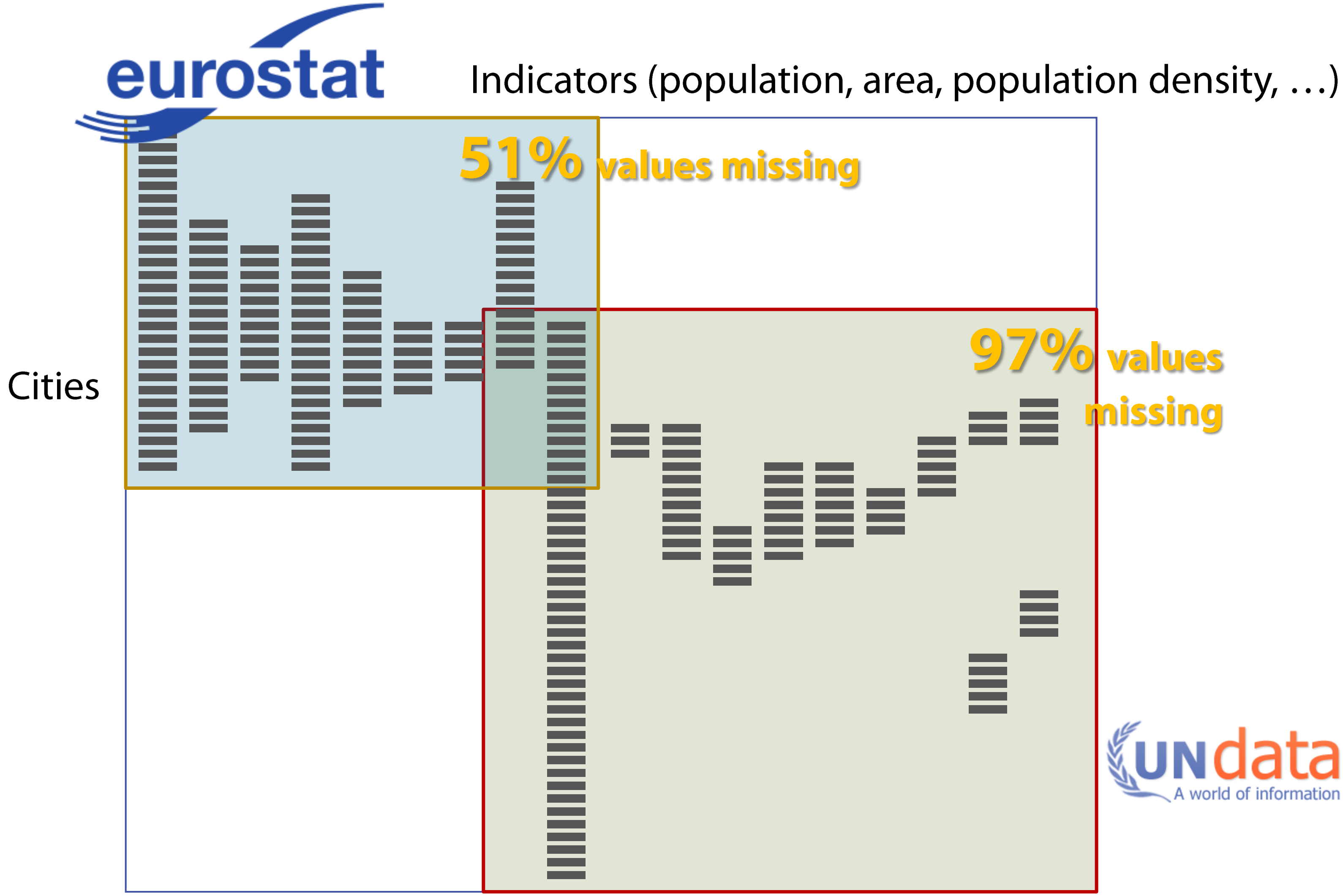
```
{ ?V c:sc? c}.
{{x rdf:type ?V } UNION
{x ?P _:b0 . ?P c:dom ?V } UNION
{ _:b1 ?P x . ?P c:rng ?V }
} UNION {c rdf:type c:UnivClass }}
```

Schema-Agnostic Query Rewriting Evaluation

- Avoids worst-case exponential blowup of other rewriting approaches
- Rewriting times negligible
- Number of materialized triples similar to number of ontology triples
- Example EUGEN query 2
  - Number of subclasses configurable



However numeric Open data is still too sparse, ontological reasoning is not enough



How about missing numerical data

- Can we infer population density from given data?
  - computations not supported by Semantic Web reasoners
- How to formalize relationships between numeric attributes for automatic transitive computation?
- Use Equations!
  - Population density
  - Unit conversion: area (km^2)

Give us all the cities where the temperature in December is above 24 degrees Celsius with a population density around 3000p/km2.

We need all the cities

RQ 1: Can we produce and effectively use **rewritings** of SPARQL queries which are **independent of the ontology** and **avoid the exponential blowup** of standard query rewriting techniques?

Schema-Agnostic Rewriting with SPARQL 1.1

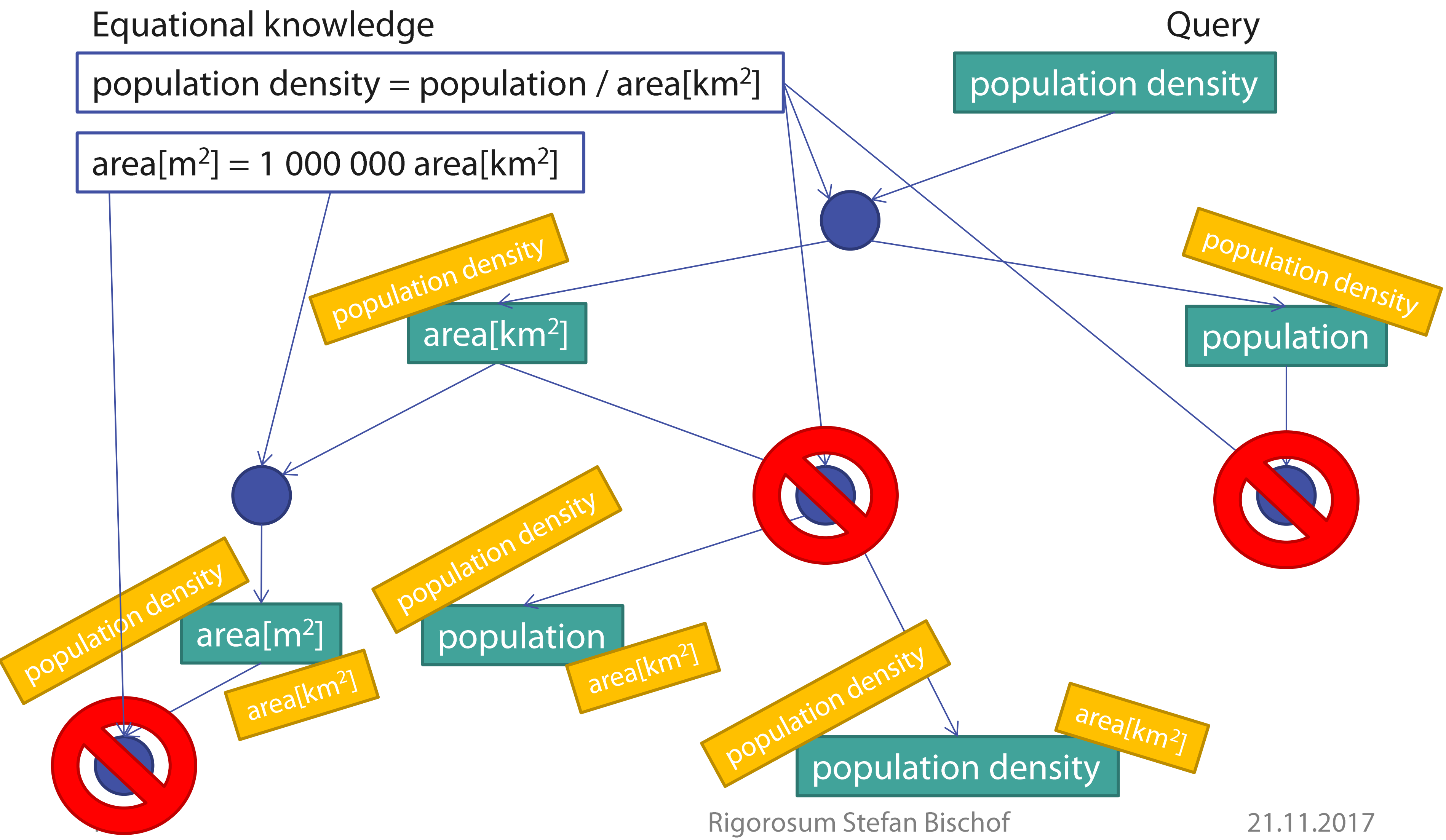
Missing numeric data?

RQ 2: Can we express and effectively use **equational knowledge** about numerical values of instances along with RDFS and OWL to **derive new values**?

RDF Attribute Equations

Use equations to expand indicators

… but only if no adornment occurs in equation .

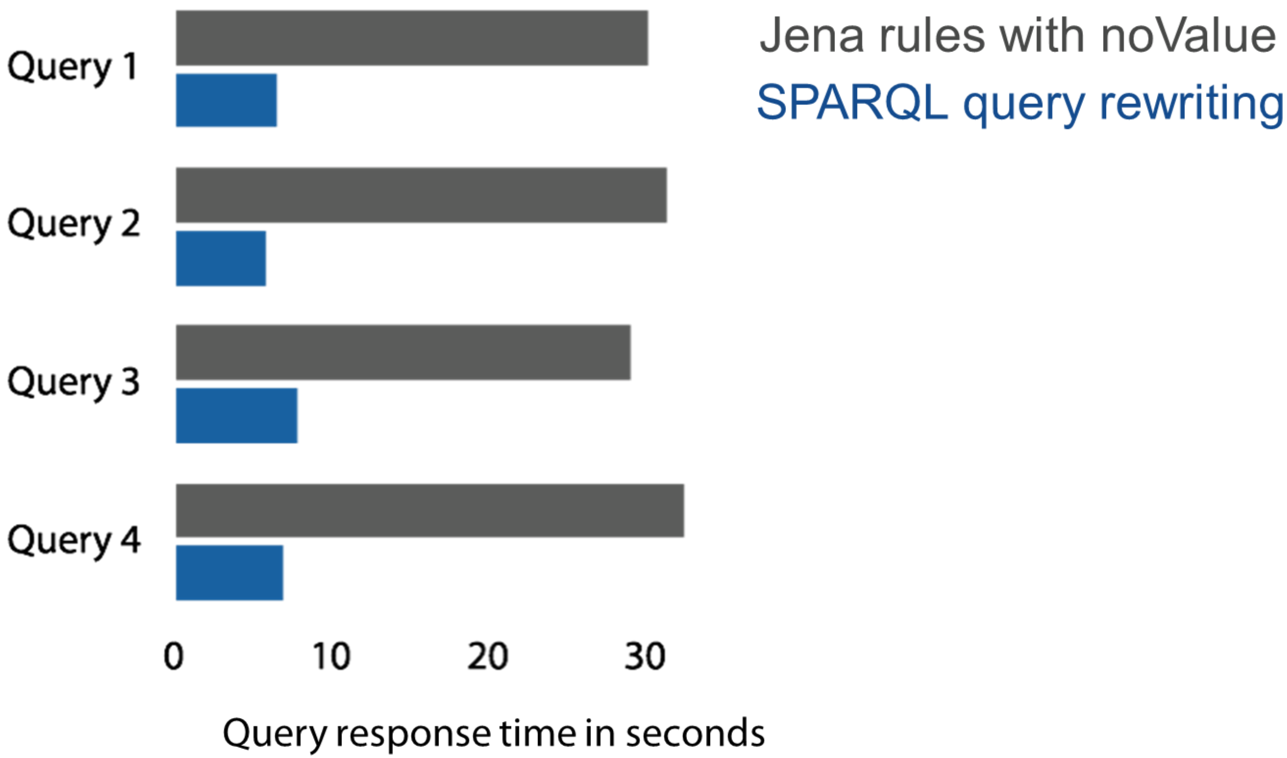


Comparison with declarative rule language

- System: Apache Jena
  - Triple store with SPARQL API
  - Rules (forward and backward)
- Backward-chaining did not terminate (missing termination condition)
- Naïve forward-chaining did not terminate (condition, rounding errors)
- Forward-chaining on acyclic (data-coherent) instance data did not terminate (rounding errors)
- Forward-chaining rules with negation-as-failure did terminate

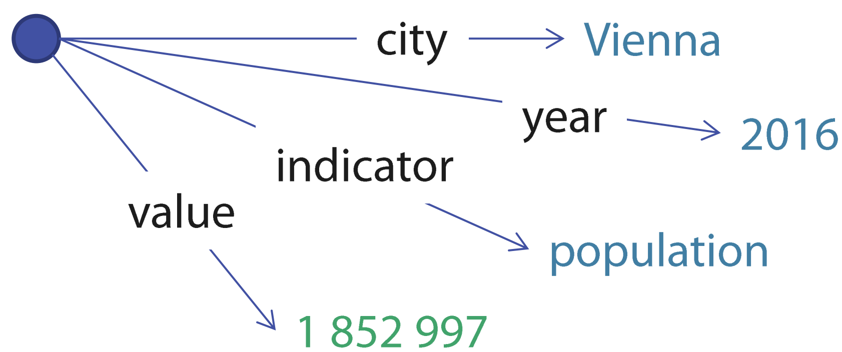
- Our SPARQL query rewriter

Comparison with declarative rule language



RDF Attribute Equations are not enough

- Data from some sources like Eurostat come as multidimensional data:
  - Temporal (December)
  - Unit of measurement (degrees Celsius)
  - Aggregation (mean, min, max, …)



Give us all the cities where the temperature in December is above 24 degrees Celsius with a population density around 3000p/km2.

We need all the cities

RQ 1: Can we produce and effectively use **rewritings** of SPARQL queries which are **independent of the ontology** and **avoid the exponential blowup** of standard query rewriting techniques?

Schema-Agnostic Rewriting with SPARQL 1.1

Missing numeric data?

RQ 2: Can we express and effectively use **equational knowledge** about numerical values of instances along with RDFS and OWL to **derive new values**?

RDF Attribute Equations

Equations for multidimensional data?

QB Equations

Multidimensional Data in RDF

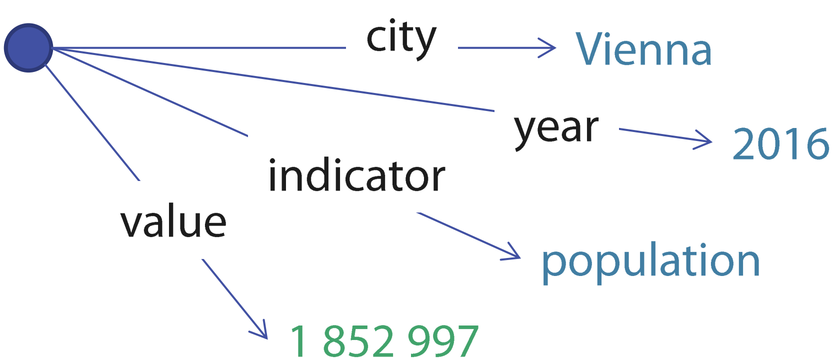
- W3C Recommendation: Data Cube Vocabulary (QB)

Dataset Schema
<City, Year, Indicator, Value>

Dimensions | Measure

- Data: Observations

<Vienna, 2016, population, 1 852 997>

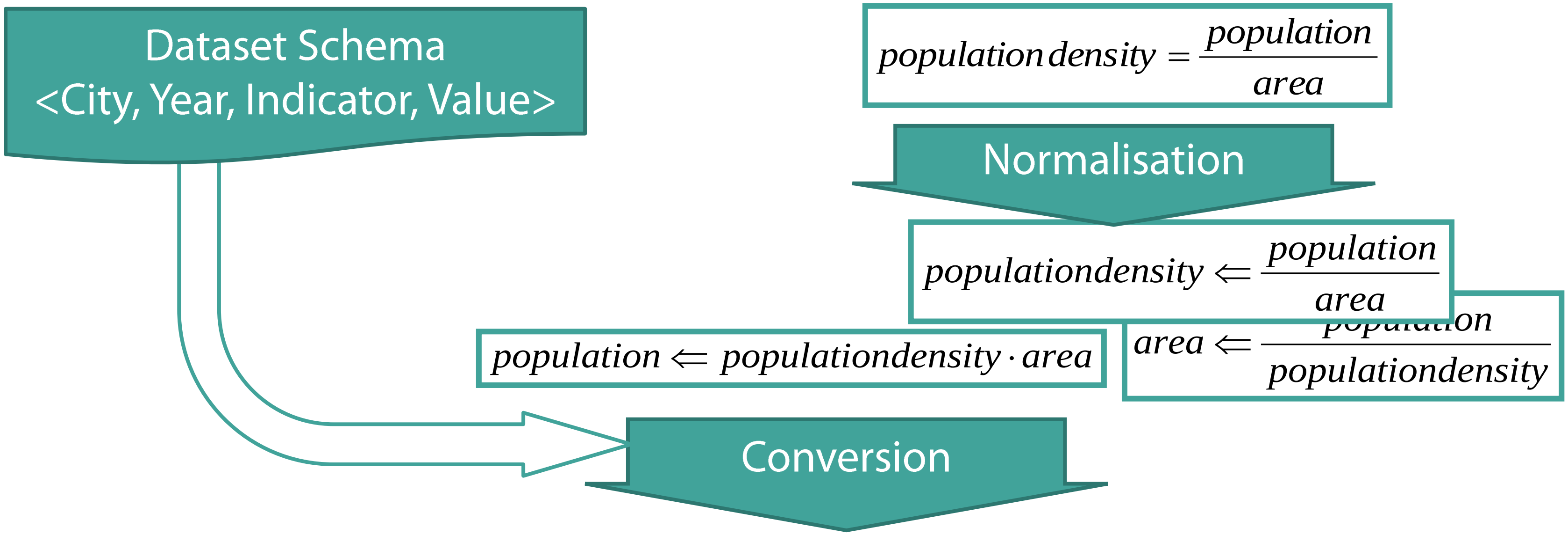


- How can we **efficiently** express:
  - Population density can be computed:

$$population\,density = \frac{population}{area}$$

  - Regardless of other dimensions, e.g., city, year, …

QB Equations Rule-Based Semantics



?newID<?City, ?Year, **population_density**, ?Value, ±?Error> provenance(…) ⟵ — **Output observation**

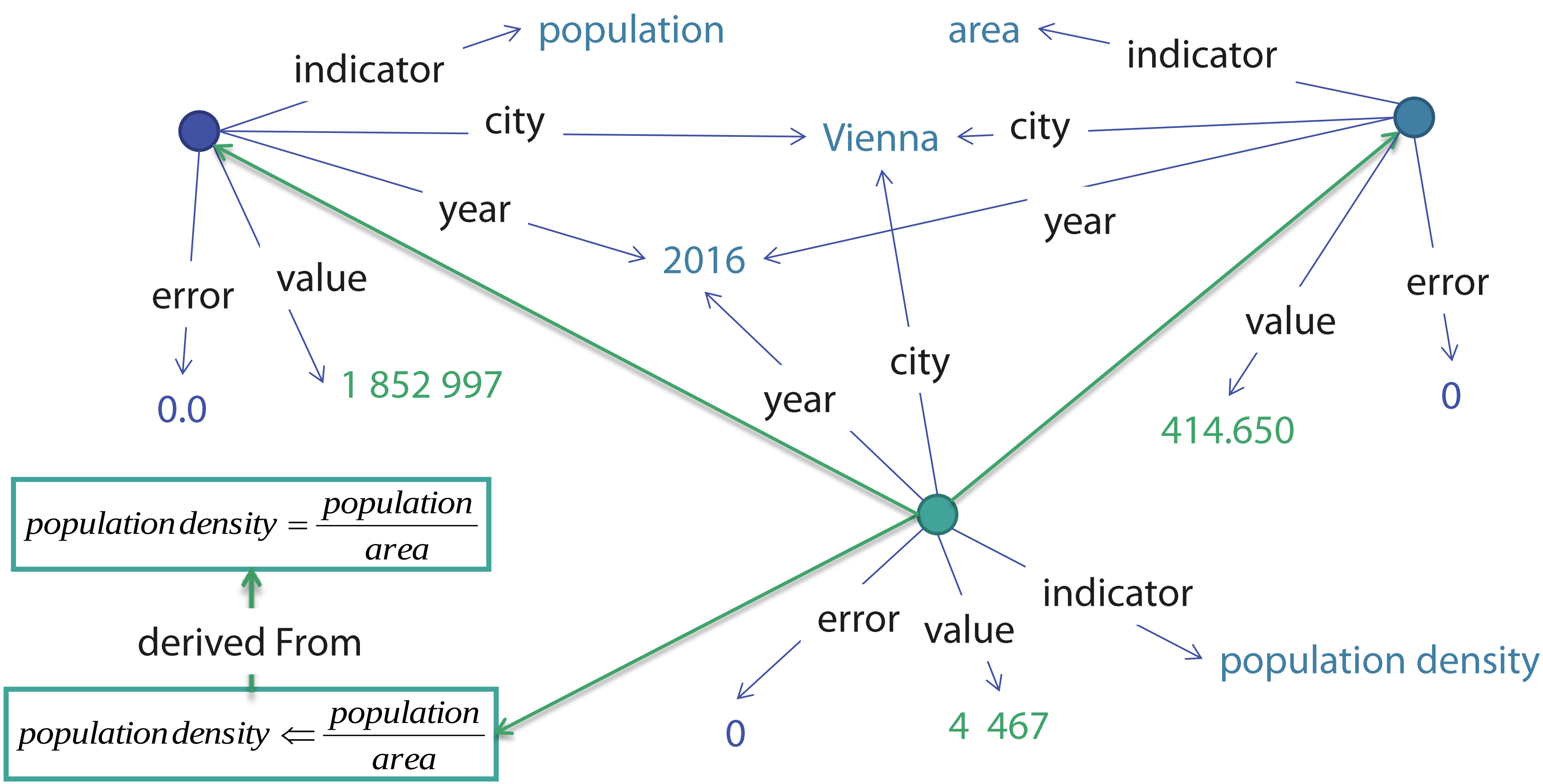
?IDP<?City, ?Year, **population**, ?P, ±?E_P>,
?IDW<?City, ?Year, **area**, ?A, ±?E_A>, — **Input observations**

?Value := ?P/?A, — **Value computation**

?Error := propagated_error(?P,?E_P,?A,E_A), — **Error propagation**

FILTER(no better observation exists),
FILTER(none of the input observations have been computed based on this equation) — **Termination**

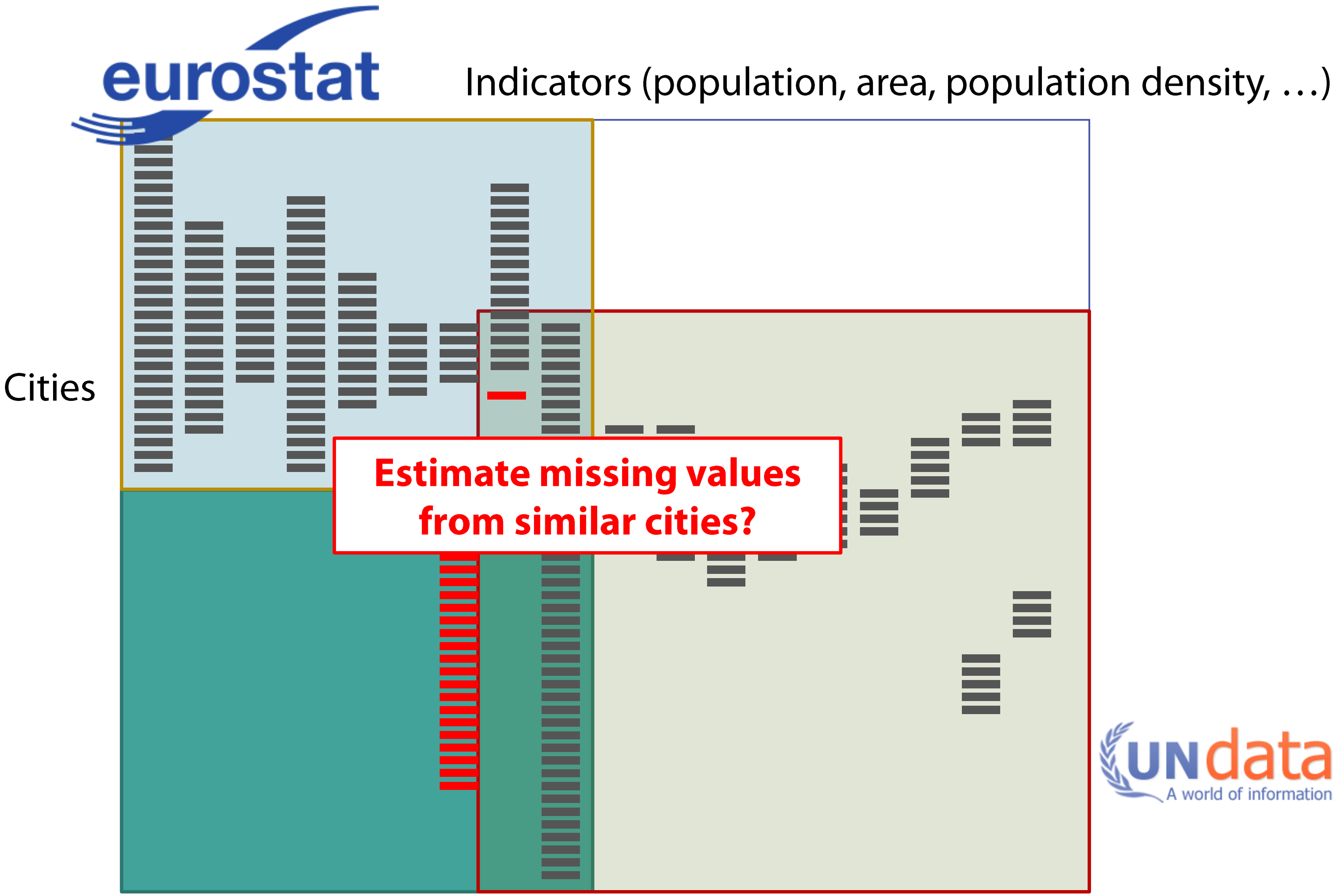
Example: compute population density



Evaluation: QB Equations derived from Eurostat

- QB Equations generated from 61 Eurostat indicator definitions
  - Normalised to 267 QB rules
  - 147 QB rules applicable
  - Implemented as SPARQL CONSTRUCT queries + data loading
- Evaluation of QB rules recomputes the Eurostat indicator values
- Found inconsistencies in integrated data and indicator definitions
- QB Equations could compute 10k new values for the indicator **women per 100 men**, mainly for UN data cities

Integrated Open Data is Still Very Sparse



Give us all the cities where the temperature in December is above 24 degrees Celsius with a population density around 3000p/km2.

We need all the cities

RQ 1: Can we produce and effectively use **rewritings** of SPARQL queries which are **independent of the ontology** and **avoid the exponential blowup** of standard query rewriting techniques?

Schema-Agnostic Rewriting with SPARQL 1.1

Missing numeric data?

RQ 2: Can we express and effectively use **equational knowledge** about numerical values of instances along with RDFS and OWL to **derive new values**?

RDF Attribute Equations

Equations for multidimensional data?

QB Equations

Can statistical methods help?

Adopt machine learning methods

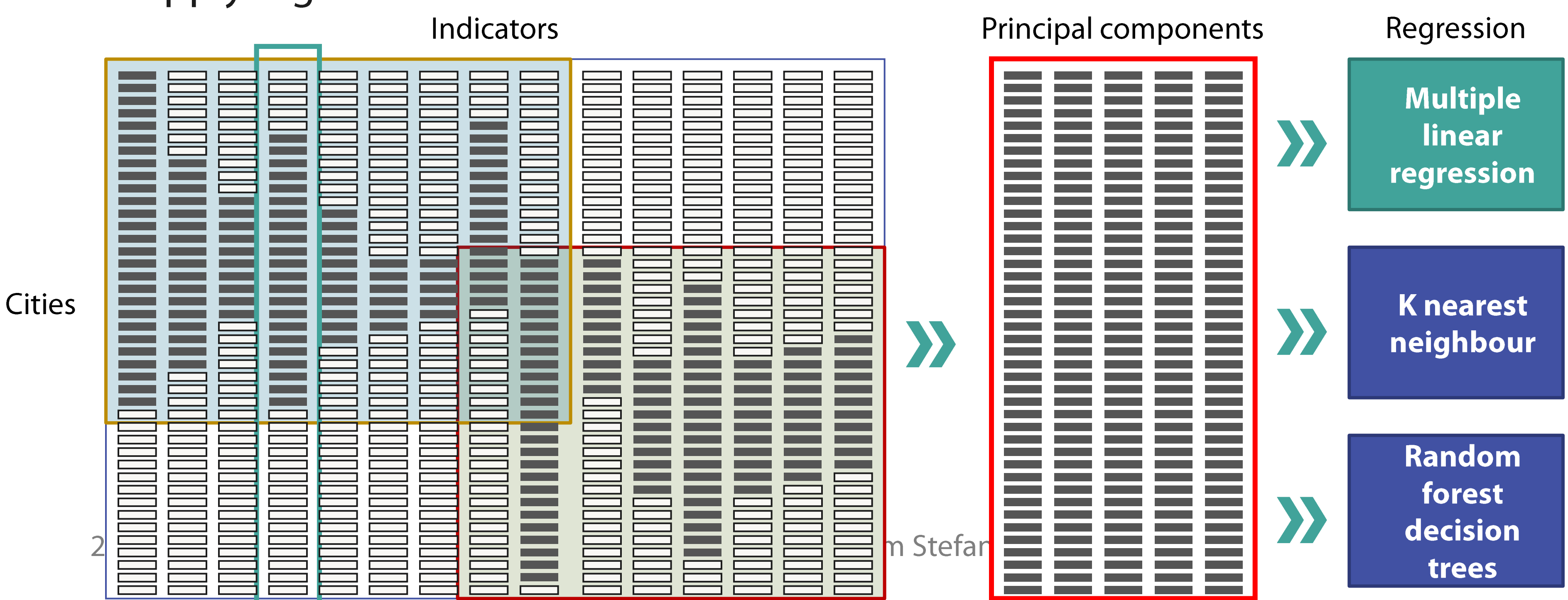
Statistical Regression Analysis on incomplete data

Regression analysis to predict missing values, also needs complete data

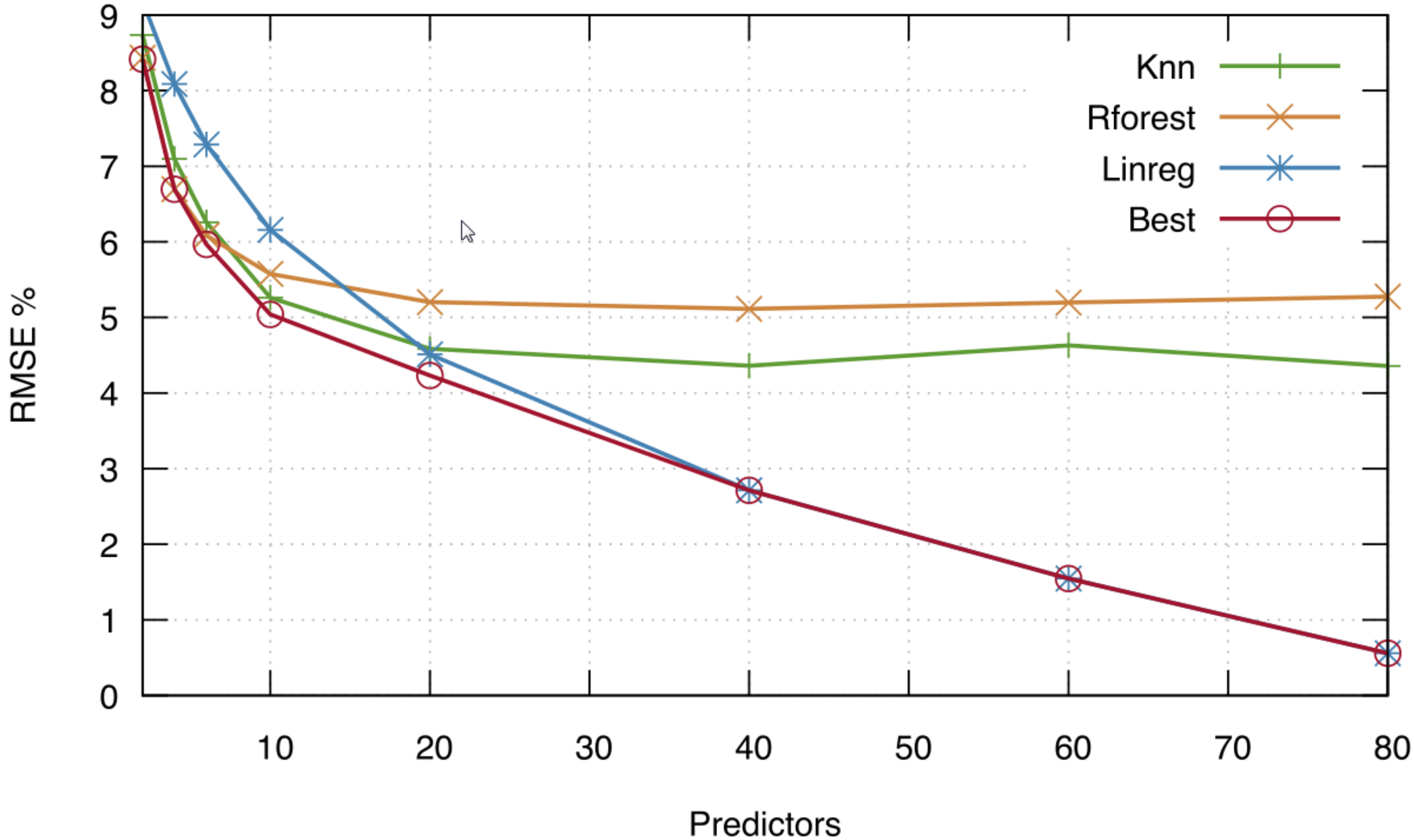
Use regularised iterative PCA to obtain complete matrix

For each **target indicator**

- Apply regularised iterative PCA to obtain principal components
- Apply regression → evaluate → select the best method



How Many Principal Components are Needed?



Evaluation: PCA Regression

- Data from Eurostat and UN Data
  - 1961 cities in total
  - 212 indicators with enough data (875 in total)
  - Years 2004-20017 with varying completeness
  - 693k observations available for training the regression models
- 609k new observations estimated by PCA regression
  - Relative error (normalised root-mean square error) < 0.55%

Give us all the cities where the temperature in December is above 24 degrees Celsius with a population density around 3000p/km2.

We need all the cities

RQ 1: Can we produce and effectively use **rewritings** of SPARQL queries which are **independent of the ontology** and **avoid the exponential blowup** of standard query rewriting techniques?

Schema-Agnostic Rewriting with SPARQL 1.1

Missing numeric data?

RQ 2: Can we express and effectively use **equational knowledge** about numerical values of instances along with RDFS and OWL to **derive new values**?

RDF Attribute Equations

Equations for multidimensional data?

QB Equations

Can statistical methods help?

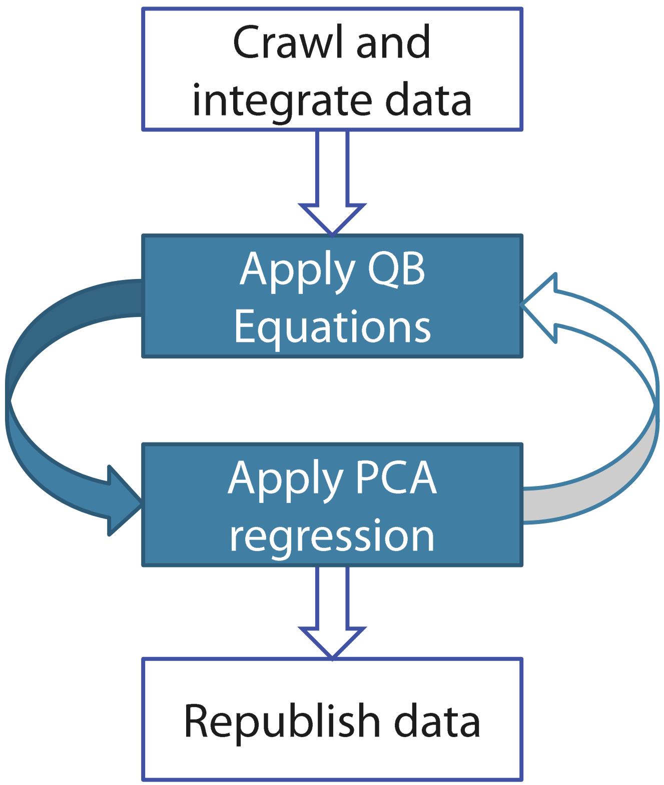
Adopt machine learning methods

Can we combine these two?

RQ 3: Can we **combine statistical inference** with OWL and **equational knowledge** to improve missing value imputation?

Combination: QB Equations + machine learning

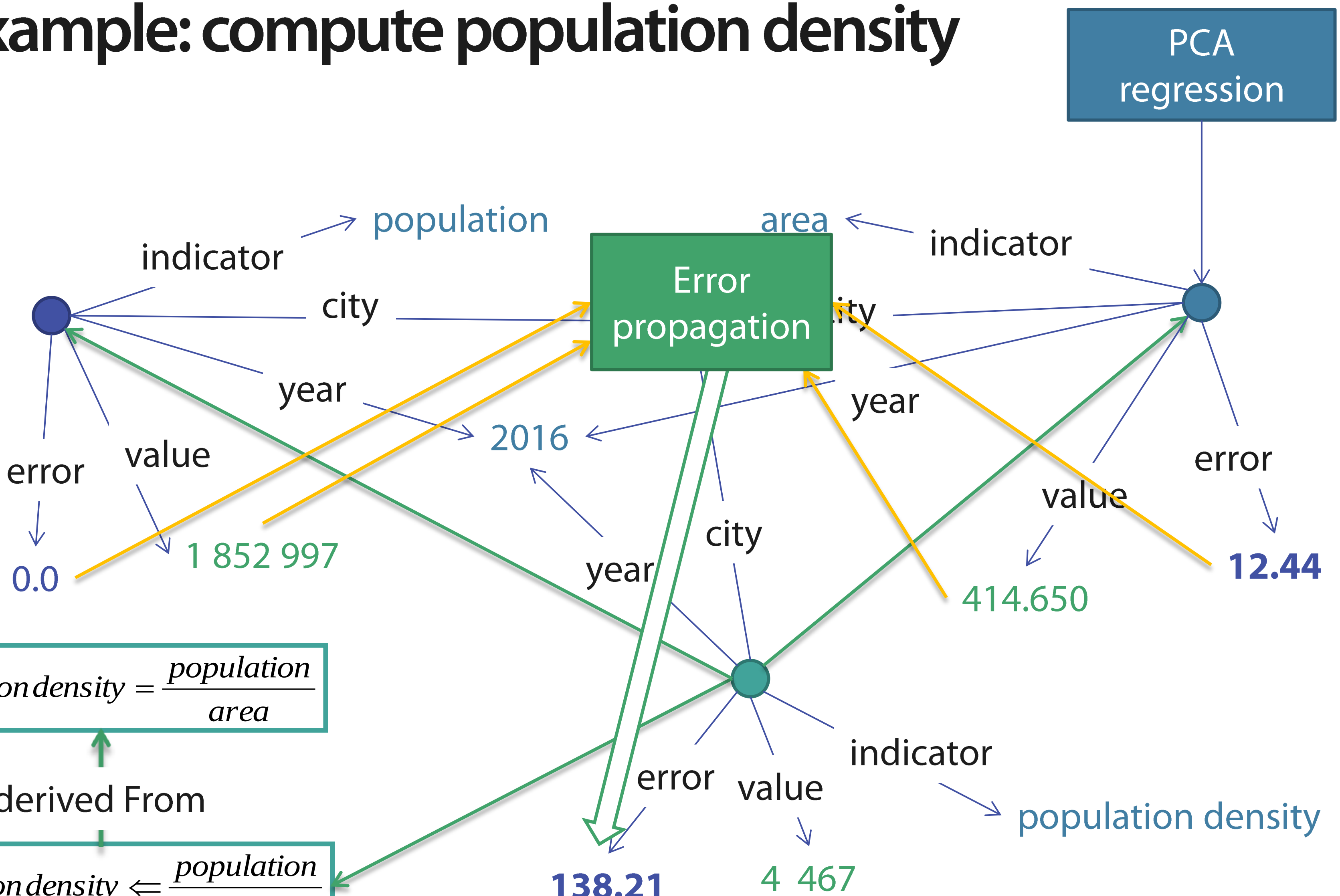
Iterative Enrichment of Numerical Data



- Use complementary methods for numerical data enrichment
  - Statistics
  - Equations
- After each iteration: decide which values are better than earlier values?
  - Use error estimate from statistical methods

How can we get error estimates for the QB Equations?

Example: compute population density

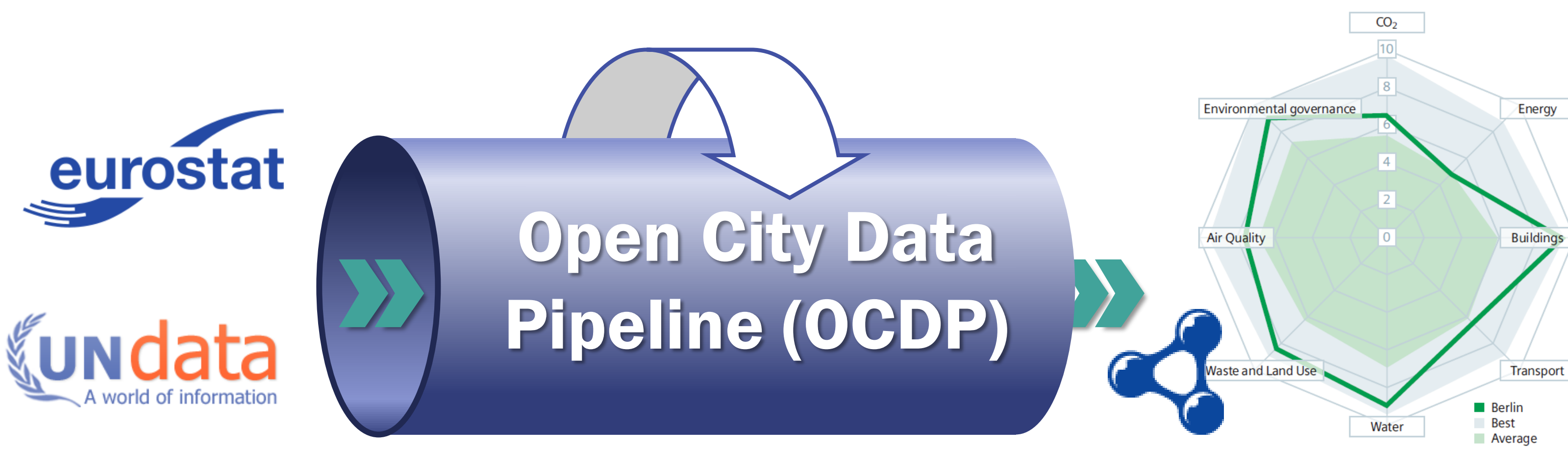


$$population\,density = \frac{population}{area}$$

$$population\,density \Leftarrow \frac{population}{area}$$

We built a system: Open City Data Pipeline

- Exploit available open data on cities to compute comparable indicators
- Crawled and integrated Eurostat and UN Data for statistical data of cities
- Enrich integrated data by equational knowledge and statistical methods
- Republish integrated and enriched data as Linked Data

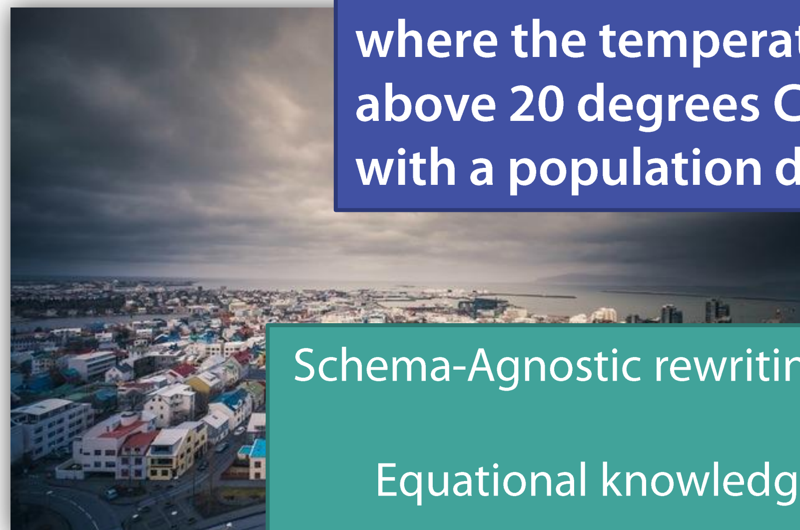


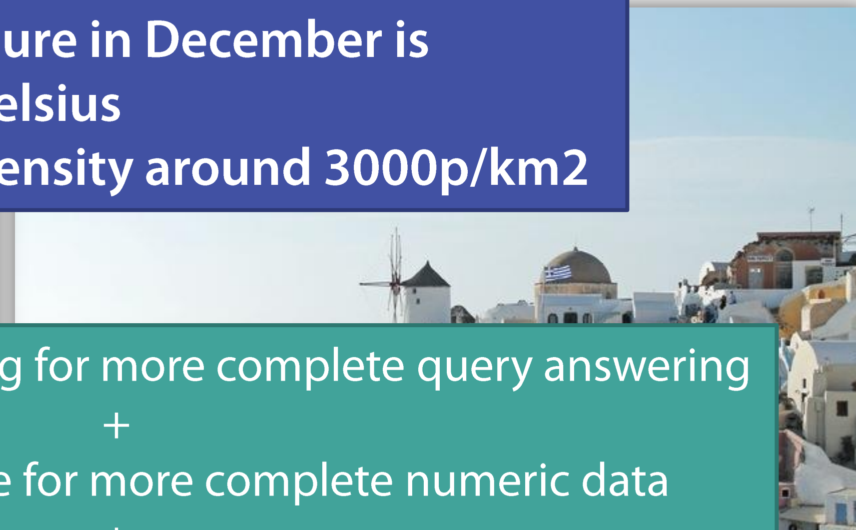
Evaluation Combination Equations + PCA Regression

- Statistics one iteration (PCA regression + QB Equations)
  - 991k observations from crawled data
  - 522k new or better observations from PCA regression
  - 230k better observations from QB Equations
  - 232k new observations from QB Equations
- Evaluation of the decision task, use error to decide which value to pick
  - 91% average precision (are picked values really better?)
  - Favor precision over accuracy
  - QB Equations are sensitive to correct error estimates
- Same or better values for 80 of 82 indicators

Which city is the best?

**Give us all the cities
where the temperature in December is
above 20 degrees Celsius
with a population density around 3000p/km2**





Schema-Agnostic rewriting for more complete query answering
+
Equational knowledge for more complete numeric data
+
Statistical methods for more complete numeric data

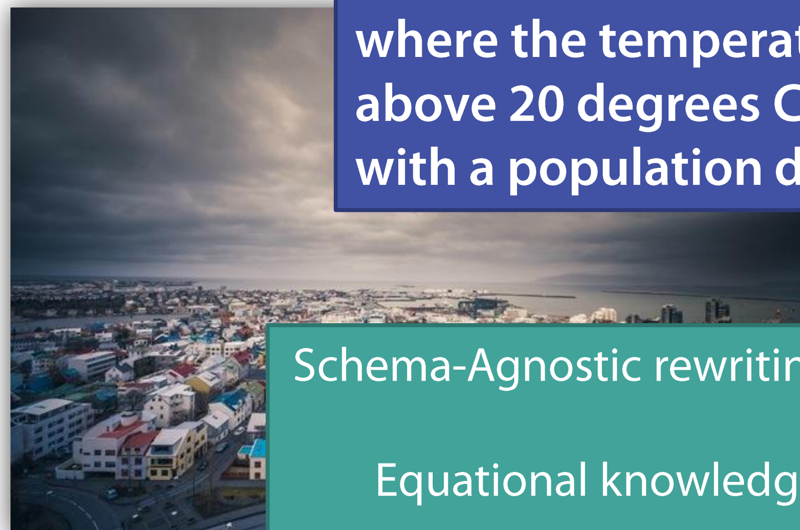
Combination of equational knowledge with Schema-Agnostic Rewriting

- Combination Attribute Equations with Schema-Agnostic Rewriting
  - Combination possible but maybe not feasible in practice
  - Integrated combination: how to encode the termination condition?
- Combination QB Equations with Schema-Agnostic Rewriting
  - Combination possible (forward chaining + backward chaining)
  - Integrated combination: SPARQL property paths might not be expressive enough to properly handle the n-ary relations of the QB Vocabulary

Open Challenges and Opportunities

- More expressive path languages for Schema-Agnostic rewriting for shorter rewritings
- Extend RDF Attribute Equations to OWL QL
- Slow QB Equation evaluation (many joins needed) needs more efficient data structure
- Investigate other approximate/statistical methods and get more data for missing value imputation

Which city is the best?




Give us all the cities
where the temperature in December is
above 20 degrees Celsius
with a population density around 3000p/km2

Schema-Agnostic rewriting for more complete query answering
+
Equational knowledge for more complete numeric data
+
Statistical methods for more complete numeric data