TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut Künstliche Intelligenz, Professur Computational Logic

Schema-Agnostic Query Rewriting in SPARQL 1.1

Stefan Bischof, Markus Krötzsch, Axel Polleres and Sebastian Rudolph

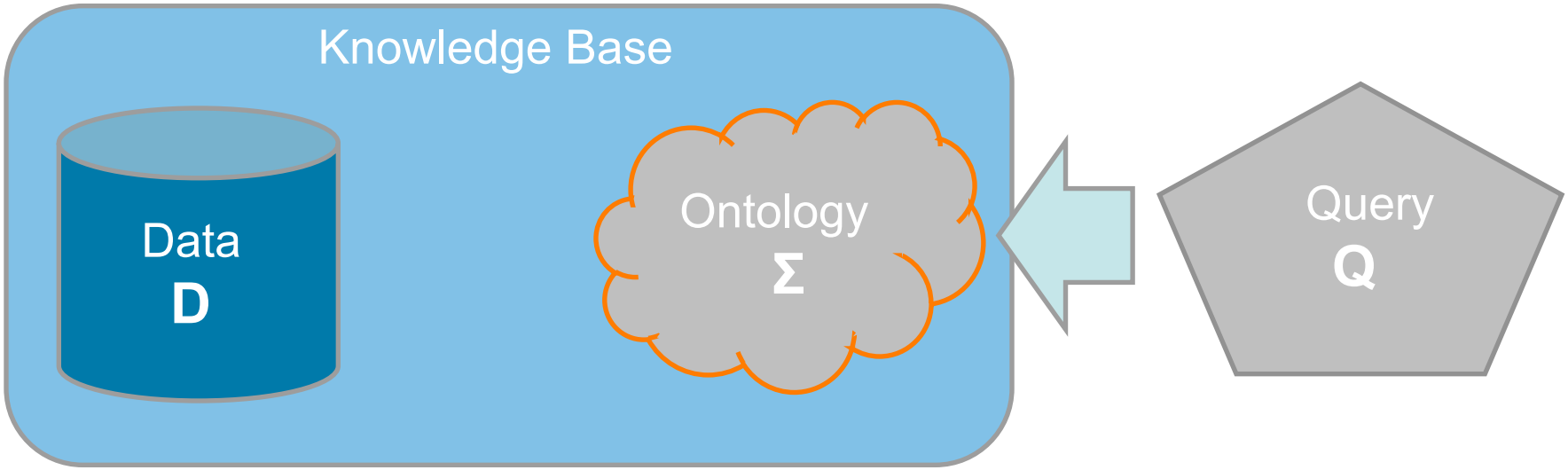
Plain Query Answering

Ontological Query Answering

Views on Ontological Query Answering

TECHNISCHE UNIVERSITÄT DRESDEN

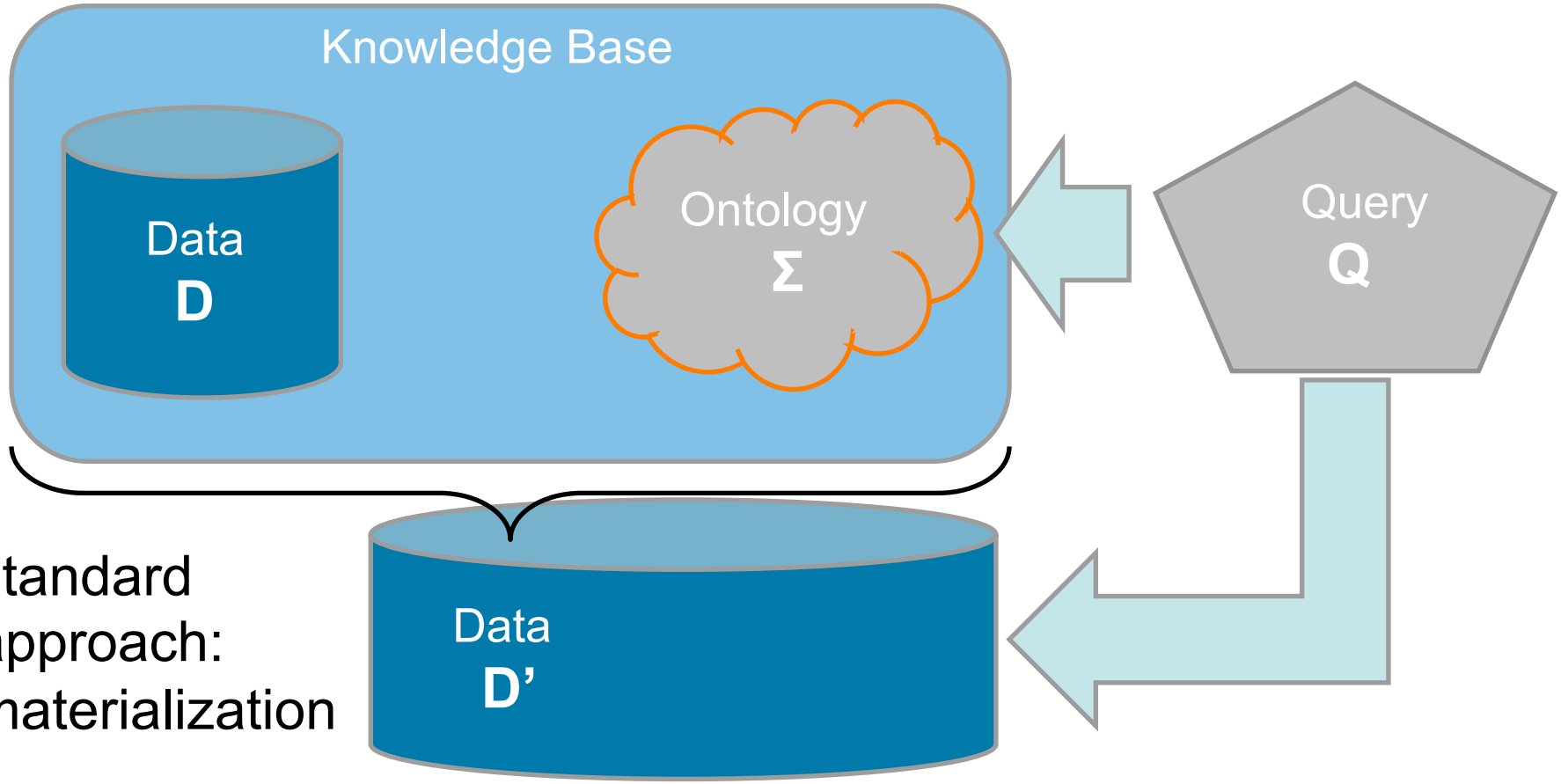
The Knowledge Representation View:



Views on Ontological Query Answering

TECHNISCHE UNIVERSITÄT DRESDEN

The Knowledge Representation View:



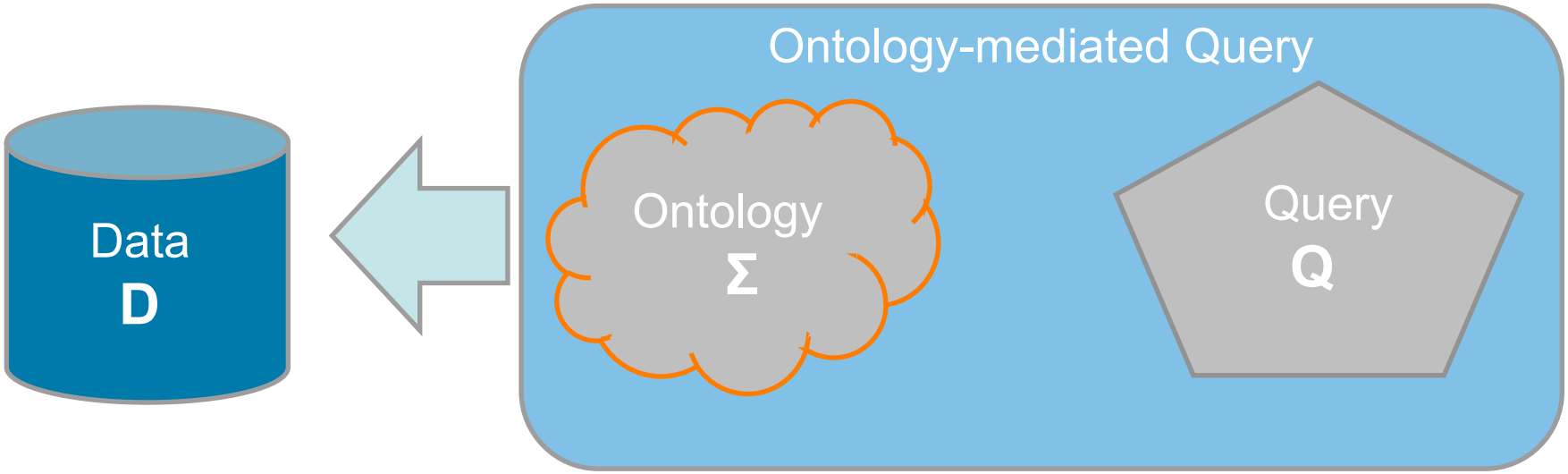
standard approach: materialization

Ontological Query Answering

Views on Ontological Query Answering

TECHNISCHE UNIVERSITÄT DRESDEN

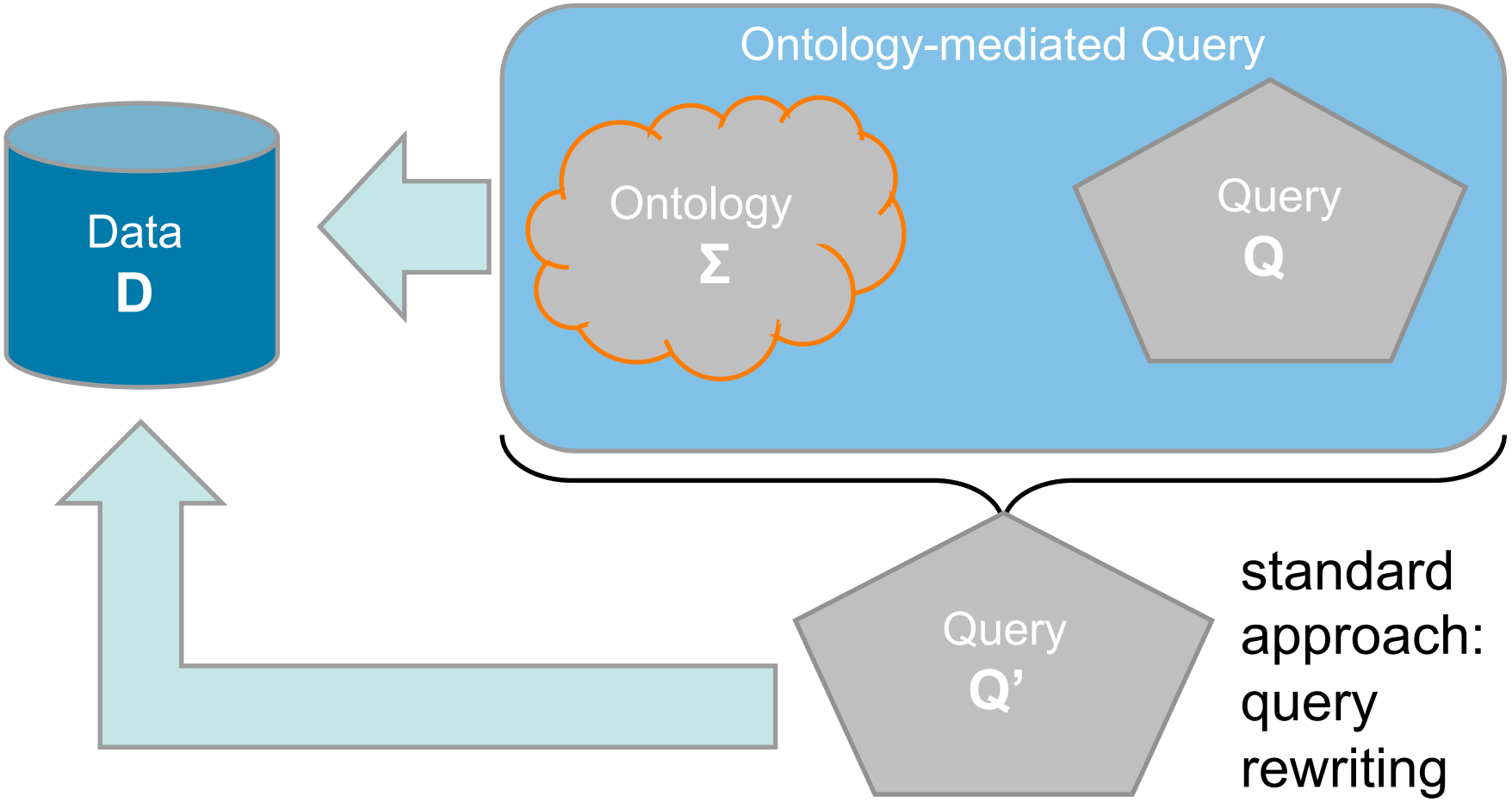
The Database View:



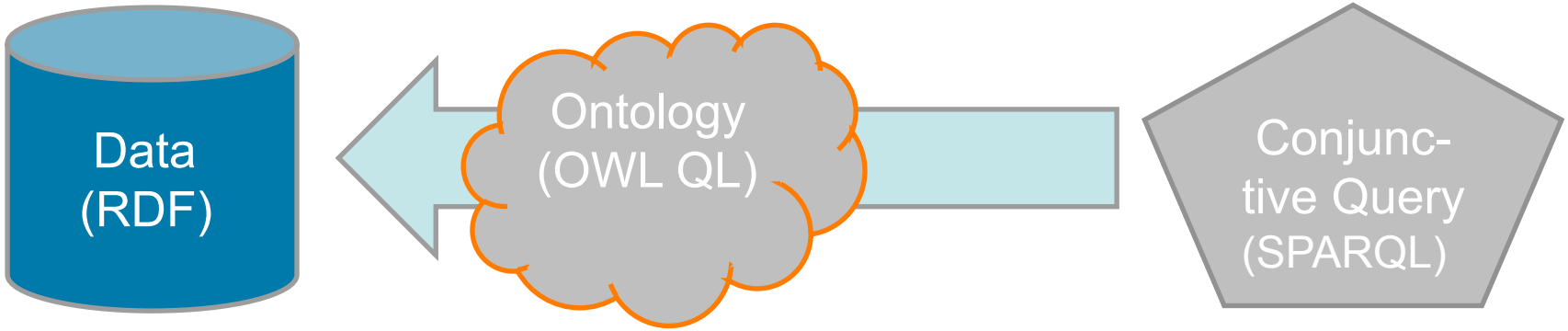
Views on Ontological Query Answering

TECHNISCHE UNIVERSITÄT DRESDEN

The Database View:

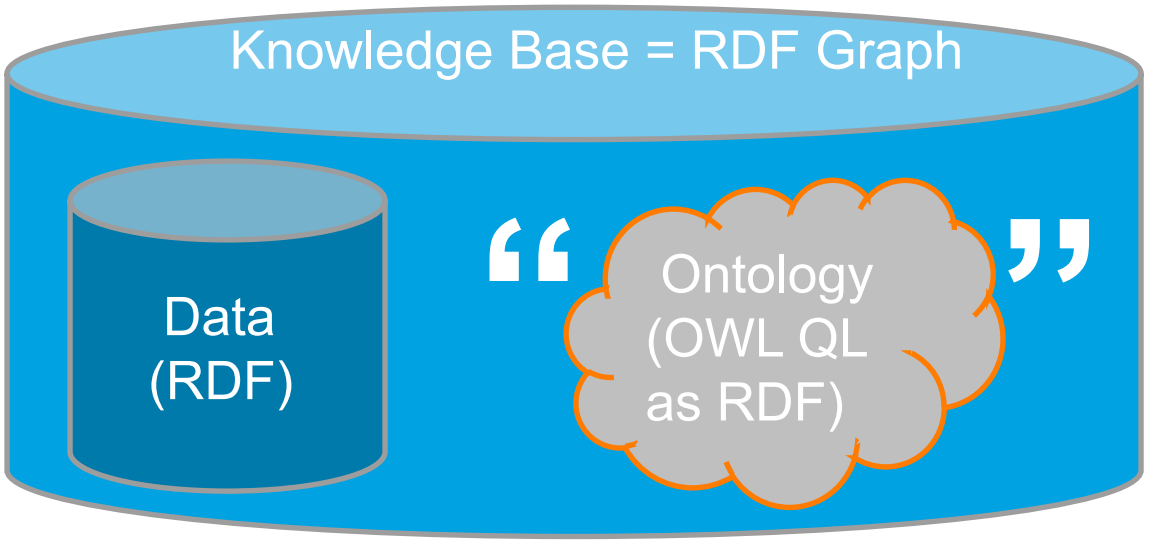


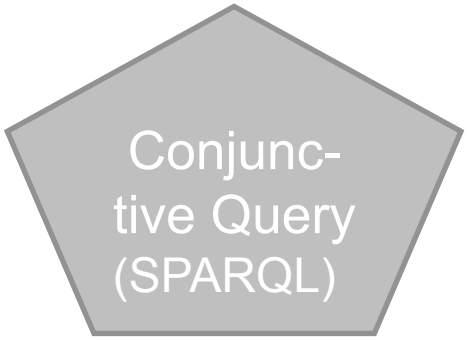
Our Scenario: CQs over OWL QL



- OWL QL based on DL-Lite family
- typically used for data-intensive scenarios
- also covers RDFS in standard use
  (i.e. without schema hijacking)

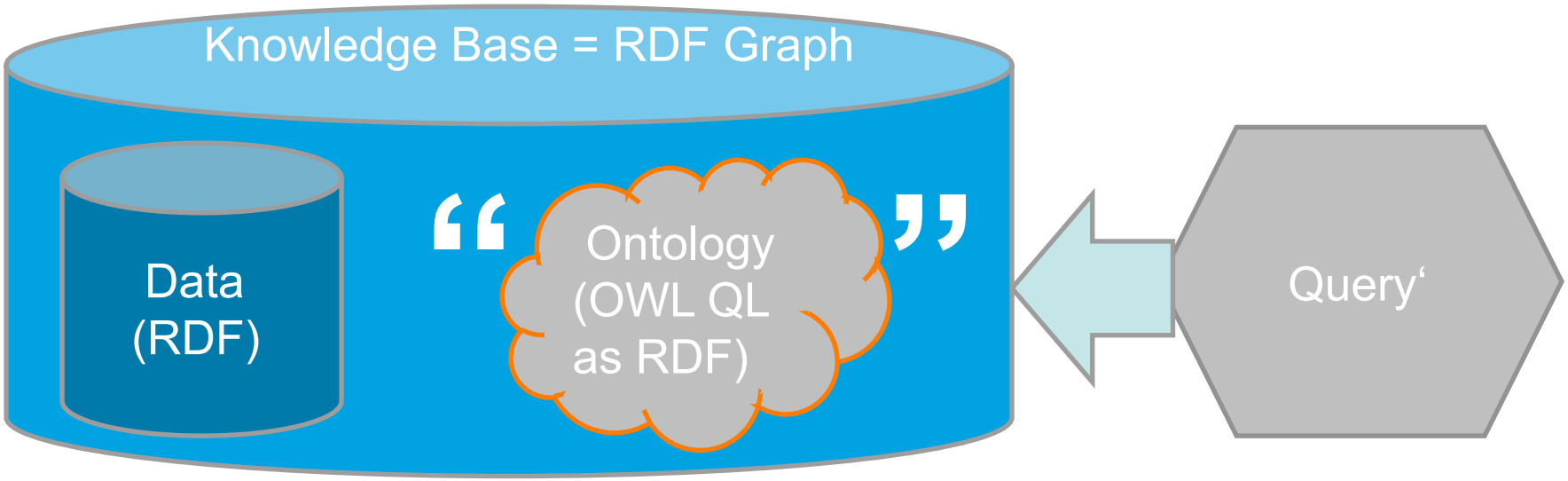
Our Scenario: CQs over OWL QL





- data and schema knowledge all mixed together
- schema knowledge expressed *as data* using special vocabulary: `rdfs:subClassOf`, …
- assumption: RDF Graph not known in advance

Our Scenario: CQs over OWL QL



- data and schema knowledge all mixed together
- schema knowledge expressed *as data* using special vocabulary: `rdfs:subClassOf`, …
- assumption: RDF Graph not known in advance
- Can we still do query rewriting? … Let’s try!

Example

Query: Give me all the individuals known to be persons.

first attempt:

```
SELECT ?p
WHERE ?p rdf:type ex:Person
```

does not work for:

```
ex:shakespeare      rdf:type            ex:Author .
ex:Author           rdfs:subClassOf     ex:Person .
```

Example

Query: Give me all the individuals known to be persons.

second attempt:

```
SELECT ?p
WHERE {?p rdf:type ex:Person} UNION
      {?p rdf:type ?c . ?c rdfs:subClassOf ex:Person}
```

does not work for:

```
ex:shakespeare      rdf:type            ex:Author .
ex:Author           rdfs:subClassOf     ex:Artist .
ex:Artist           rdfs:subClassOf     ex:Person .
```

Example

Query: Give me all the individuals known to be persons.

We need to incorporate arbitrarily long subclass paths.

Use SPARQL 1.1!

```
SELECT ?p
WHERE {?p rdf:type ?c . ?c rdfs:subClassOf* ex:Person}
```

Is this it?

Example

Query: Give me all the individuals known to be persons.

We need to incorporate arbitrarily long subclass paths.

Use SPARQL 1.1!

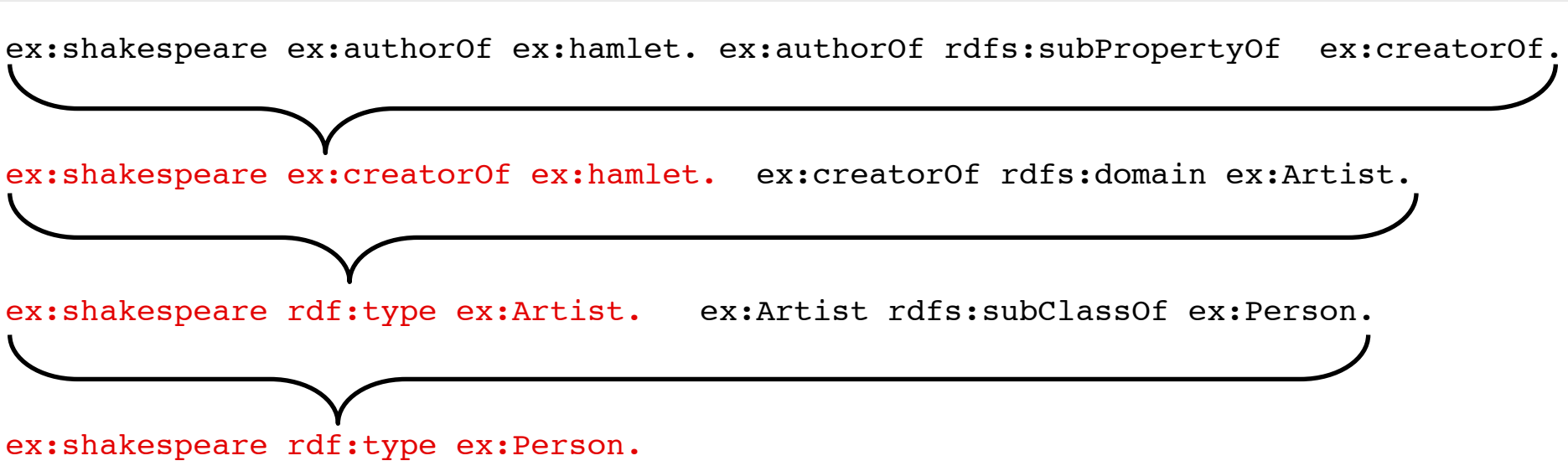
```
SELECT ?p
WHERE {?p rdf:type ?c . ?c rdfs:subClassOf* ex:Person}
```

Is this it? No!

Example

```
ex:shakespeare    ex:authorOf           ex:hamlet .
rdf:authorOf      rdfs:subPropertyOf    ex:creatorOf .
ex:creatorOf      rdfs:domain           ex:Artist .
ex:Artist         rdfs:subClassOf       ex:Person .
```

Example



Important Observations:

- proof tree is linear
- leaf triples in the proof tree form sort of a chain

→ this holds in general

Results (1/2) Rewriting for Single Triples

TECHNISCHE UNIVERSITÄT DRESDEN

Theorem:

For (almost) all OWL QL ontologies, each of the following reasoning tasks can be expressed in a single SPARQL 1.1 query:

- Is the ontology consistent?
- Is the class `A` consistent?
- Does the ontology entail `A rdfs:subClassOf B` ?
- Does the ontology entail `R rdfs:subPropertyOf S` ?
- Does the ontology entail `c rdf:type A` ?
- Does the ontology entail `c R d` ?

Directly extends to schema queries: `A`,`B`,`R`,`S` can be variables!

Example

SPARQL 1.1 query retrieving all inconsistent classes:

```
x (sCO | eqC | ^eqC | intListMember | owl:someValuesFrom |
    (owl:onProperty / (Inv | SpoEqp)* / (^owl:onProperty | rdfs:domain | rdfs:range))* ?C . {
  {?C subClassOf owl:Nothing} UNION
  {?C subClassOf ?D1 {{?C subClassOf ?D2} UNION univClass[?D2]} {
     {?D1 disjointClasses ?D2} UNION
     {?V rdf:type owl:AllDisjointClasses . twoMembers[?V, ?D1, ?D2]}
  }} UNION
  {?C (owl:onProperty / (Inv | SpoEqp)*) ?P . {
     {?P subPropertyOf owl:bottomObjectProperty} UNION
     {?P subPropertyOf ?Q1 {{?P subPropertyOf ?Q2} UNION univProperty[?Q2]} {
        {?Q1 (owl:propertyDisjointWith | ^owl:propertyDisjointWith) ?Q2} UNION
        {?V rdf:type owl:AllDisjointProperties . twoMembers[?V, ?Q1, ?Q2]}
     }
  }}
}
```

Result (2/2) Rewriting for CQs

Theorem: For (almost) all OWL QL ontologies, for all conjunctive queries Q, there is a schema-agnostic SPARQL 1.1 rewriting of linear size.

→ More complicated due to non-distinguished variables

→ Use more SPARQL features: VALUES and FILTER (some guessing involved)

→ Resulting query still of the same complexity as before

First Implementation and Testing

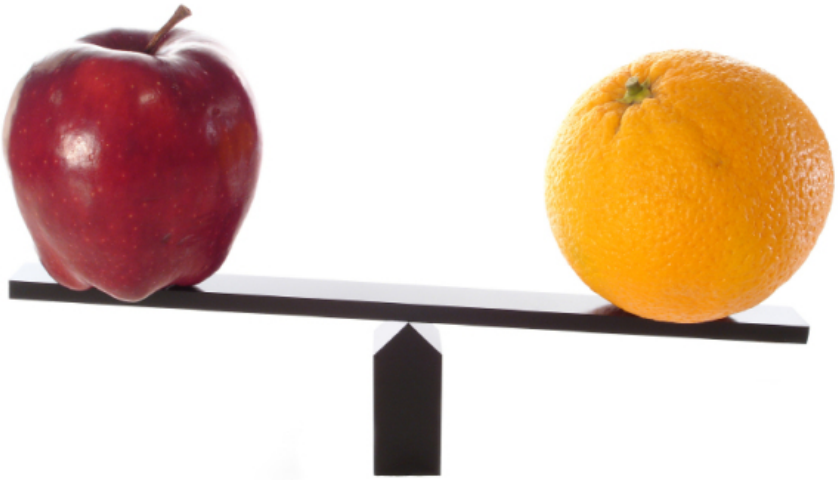
Prototype path rewriter available for download

- Simple web interface and commandline application
- http://stefanbischof.at/publications/iswc14/

First Implementation and Testing

Preliminary evaluation with LUBM and Jena ARQ.

Compared to REQUIEM using QL fragment of the LUBM ontology



- It works!
- Rewriting is always faster than REQUIEM
- Size of rewritings usually smaller than REQUIEM
- Query evaluation is usually slower than REQUIEM

Limits of the Approach

Limit 1: Reasoning complexity

A query language with data complexity C cannot express a reasoning task of combined complexity C' > C.

What does this mean for us?

SPARQL 1.1 is in NLogSpace (data complexity).

Reasoning is PTime-hard for:

OWL EL, OWL RL, RDFS in nonstandard use

→ No schema-agnostic rewriting possible for these cases unless NLogSpace = PTime.

Limits of the Approach

Limit 2: Query expressivity

Query languages usually do not solve all problems of their complexity class – some things might be impossible.

What does this mean for us?

SPARQL 1.1 cannot support owl:SymmetricProperty axioms, although it can support inverse properties.

We require OWL 2 QL with owl:SymmetricProperty paraphrased.

Take-Home Message

Every RDF database featuring SPARQL 1.1 queries can be used as an OWL QL reasoner, with full support for conjunctive queries including schema variables.

Future Work

- Evaluation
- Optimization
  - order of triple patterns impact performance
  - Exploit algebraic equivalences
- Materialize some re-occurring query fragments
  - For example subClassOf “macro”

- What can we do for more expressive ontologies?
  - Combined approaches (touch the data just a little bit)
  - More powerful queries (such as variants of Datalog)

TECHNISCHE UNIVERSITÄT DRESDEN

Thank You!