# Research Paper Presentation

Nikolay Vladimirov Nikolov, IMSE, S.N.: 779

# Content

- Introduction
- Key Issues Addressed
- SPARQL-RW: Transparent Query Access over Mapped RDF Datasources
- Rewriting Queries on SPARQL Views
- Conclusion
- Q&A Session

# Introduction

- The Web of Data - interlinked RDF datasets
  - Several datasets describe the same (or overlapped) domain

- Each dataset has its *own schema* (ontology description)
  - How do we access (query) the data from different domains using our *preferred schema*?

# Introduction (continued)

- Data providers use views to expose their data

- Use virtual views and rewrite queries
  - Which views can we use to answer the query?
  - How to combine the views to get all the needed results?
  - How do we achieve all of this in an optimized manner?

# Key Issues

- Transparently access data across different data models

- A ***scalable***, ***natively SPARQL*** algorithm for rewriting queries over virtual views
  - generates ***sound*** and ***complete*** rewritings
  - does not result in ***complex*** or ***empty*** rewritten queries

# SPARQL-RW: Transparent Query Access over Mapped RDF Datasources

K. Makris, N. Bikakis, N. Gioldasis, S. Christodoulakis

# Key Issues

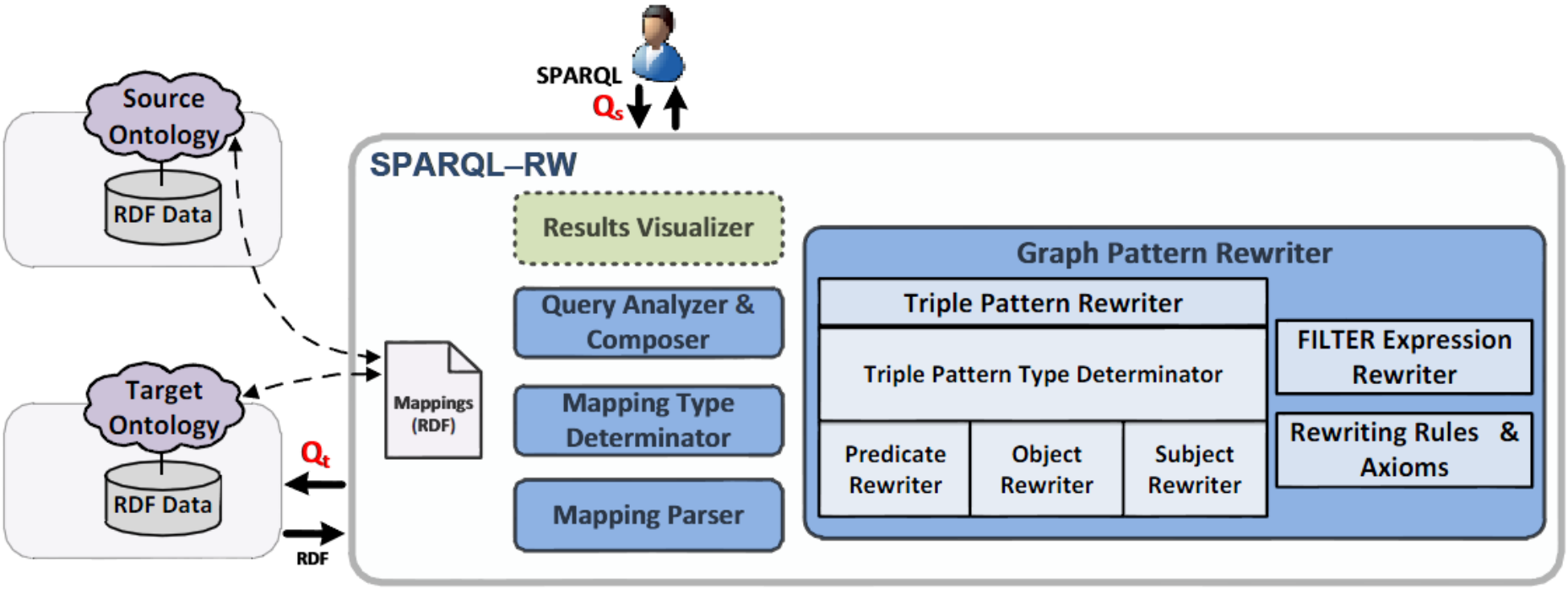
- Transparently access data across different data models

- A ***scalable***, ***natively SPARQL*** algorithm for rewriting queries over virtual views
  - generates ***sound*** and ***complete*** rewritings
  - does not result in ***complex*** or ***empty*** rewritten queries

# Requirements

- Ontology Mapping
  - apply suitable mapping strategy
- Mapping Representation
  - choose appropriate formal model and mapping language
- SPARQL Query Rewriting
  - achieve an equivalent rewriting of the source query

# SPARQL-RW Framework



# Mapping Model

- Highly Expressive Set of Mapping Types
  - expressed using Description Logics (DL)
  - no limitation on language used for mapping representation
- n:m Cardinality of the Mappings
  - equivalence or subsumption relationships

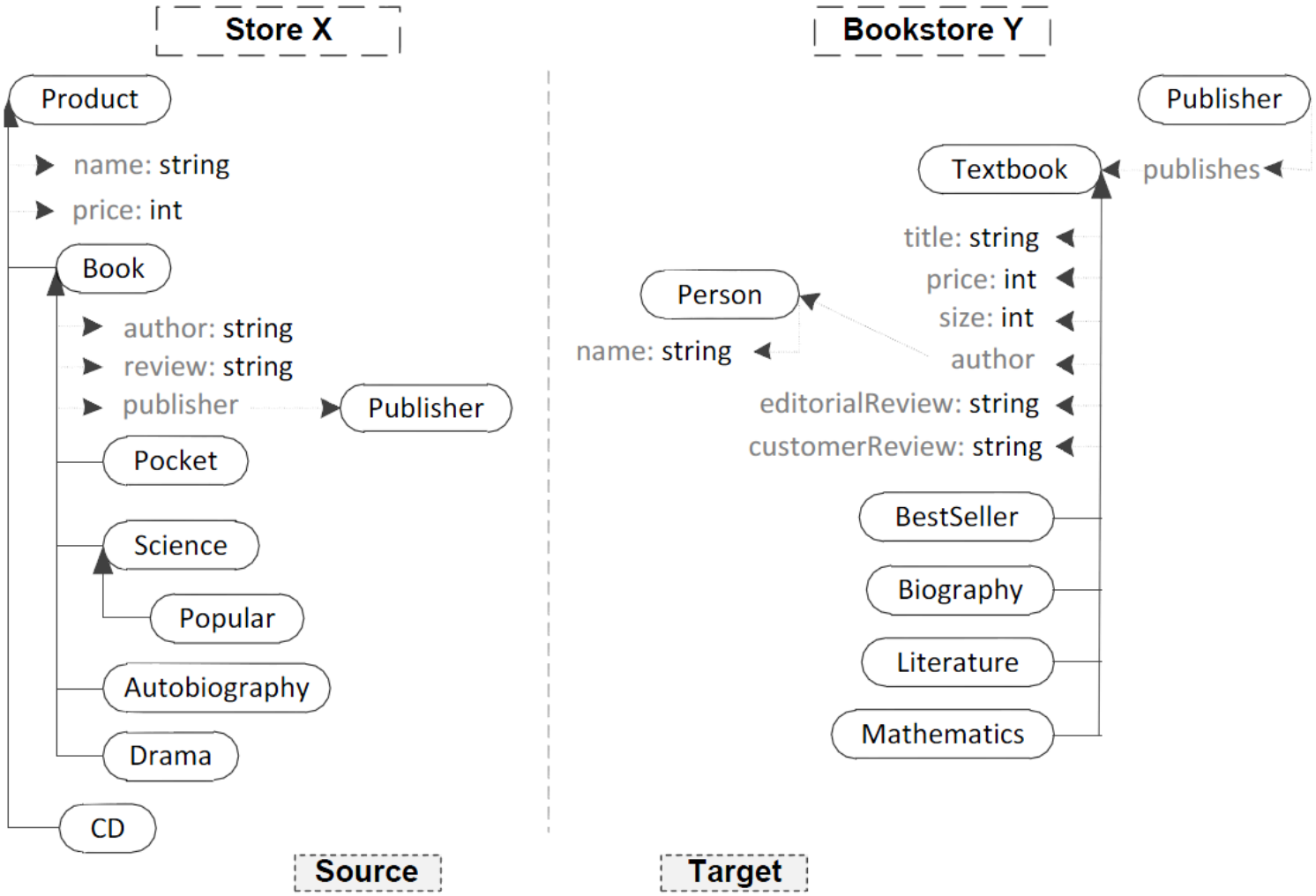
# Mapping Model (continued)

- Class Expressions
  - union and intersection operations
  - restriction using property expressions
- Object Property Expressions
  - union, intersection, composition and inverse operations

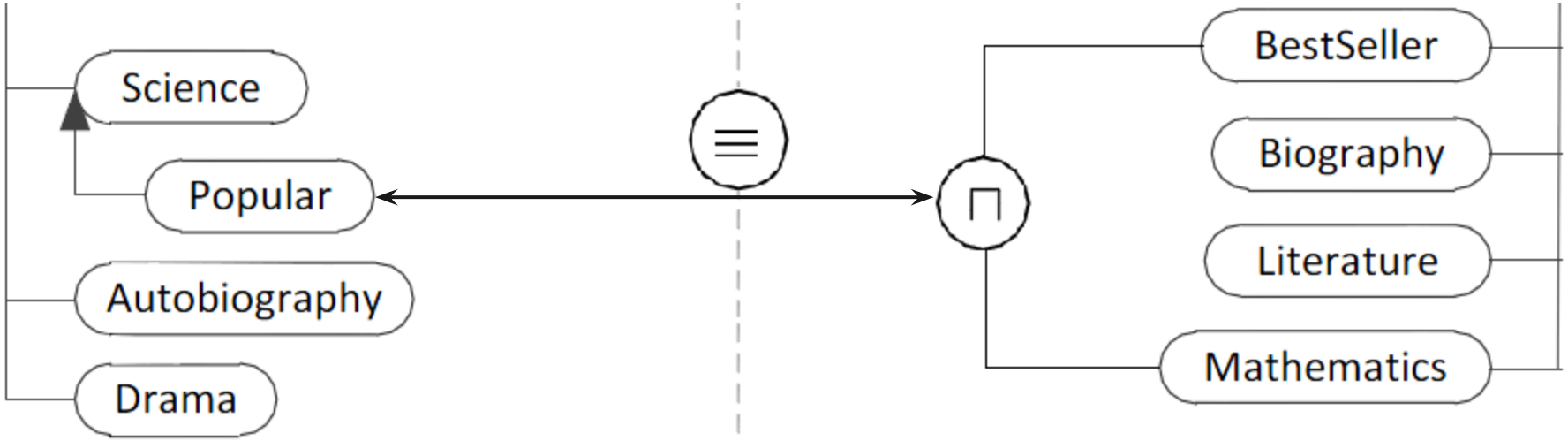
# Mapping Model (continued)

- Datatype Property Expressions
  - union, intersection and composition operations
- Individual Expressions
  - direct mapping between individuals

# Example 1

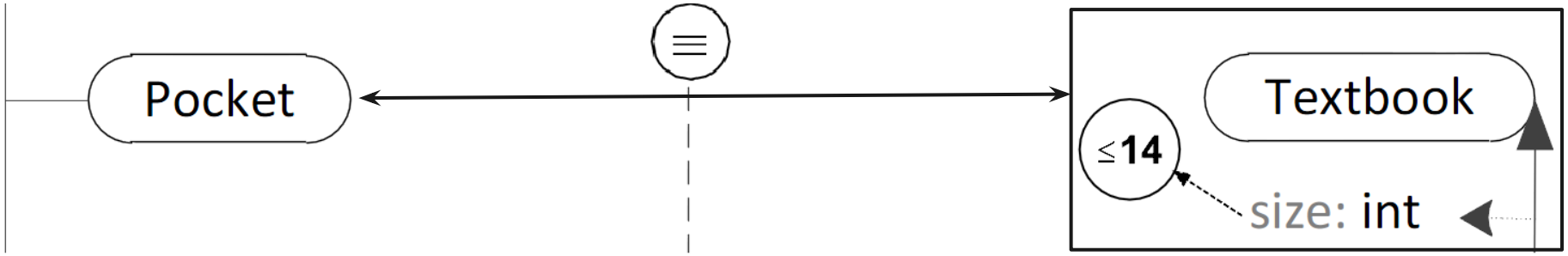

Store X
Product
name: string
price: int
Book
author: string
review: string
publisher
Publisher
Pocket
Science
Popular
Autobiography
Drama
CD
Source
Bookstore Y
Publisher
Textbook
publishes
title: string
price: int
size: int
author
Person
name: string
editorialReview: string
customerReview: string
BestSeller
Biography
Literature
Mathematics
Target

# Example 1 (continued)




$\mu_1$: src : Popular ≡ trg : BestSeller ⊓ trg : Mathematics

# Example 1 (continued)




$\mu_2$: src : Pocket ≡ trg : Textbook ⊓ ∃trg : size.$\leq_{14}$

# Example 1 (continued)

$\mu_3$: src : name $\sqsupseteq$ trg : title

$\mu_4$: src : publisher $\equiv$ trg : publishes$^-$

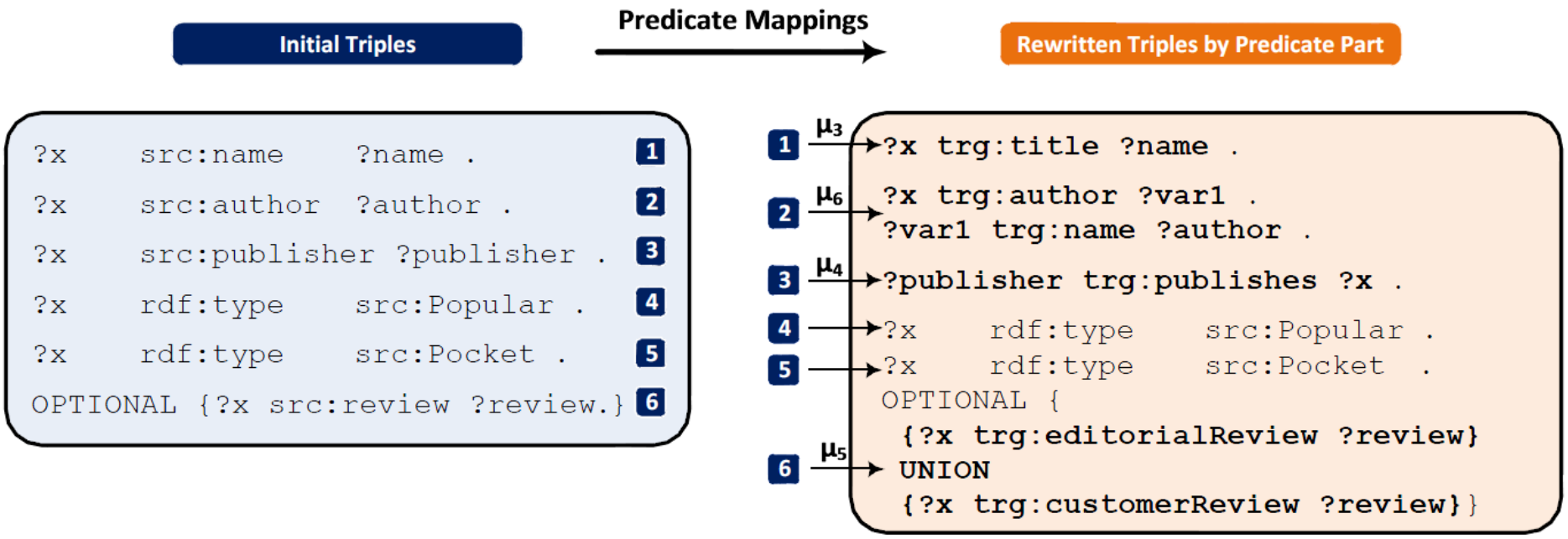
$\mu_5$: src : review $\equiv$ trg : editorialReview $\sqcup$ trg : customerReview

$\mu_6$: src : author $\equiv$ trg : author $\circ$ trg : name

# Query Rewriting

- Three Step Process
  - performed by *Subject*, *Predicate* and *Object Rewriter*
- Independent of Query Type
- Independent of Solutions Sequence Modifiers
- Independent of SPARQL Algebra Operators

# Example 2



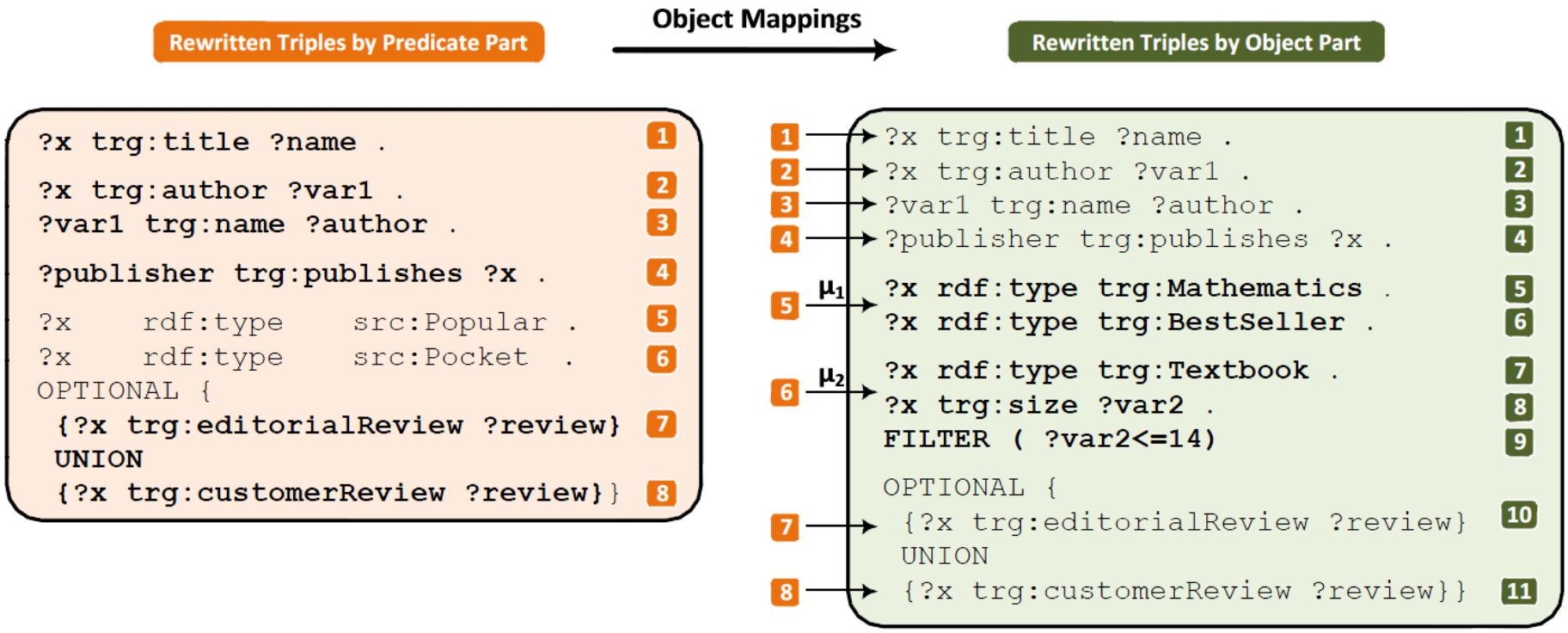

$\mu_3$: src : name $\sqsupseteq$ trg : title

$\mu_6$: src : author $\equiv$ trg : author $\circ$ trg : name

$\mu_4$: src : publisher $\equiv$ trg : publishes $^-$

$\mu_5$: src : review $\equiv$ trg : editorialReview $\sqcup$ trg : customerReview

# Example 2 (continued)




$\mu_1$: src : Popular $\equiv$ trg : BestSeller $\sqcap$ trg : Mathematics

$\mu_2$: src : Pocket $\equiv$ trg : Textbook $\sqcap \exists$trg : size.$\leq_{14}$

# Relevant Work in the Problem Area

- Correndo et. al proposes a similar framework
  - restrictive with regard to supported query types
  - mapping definition is harder to achieve
- Akahani et. al proposes a theoretical approach
  - no specific context (SPARQL)
  - no specific rewriting algorithms

# Rewriting Queries on SPARQL Views

W. Le, S. Duan, A. Kementsietsidis, F. Li, M. Wang

# Key Issues

- Transparently access data across different data models

- A ***scalable***, ***natively SPARQL*** algorithm for rewriting queries over virtual views
  - generates ***sound*** and ***complete*** rewritings
  - does not result in ***complex*** or ***empty*** rewritten queries

# SQR (SPARQL Query Rewriting) Algorithm

- Output: **Sound** and **complete** rewriting of a query
- Input: Views, Query
- Two-phase algorithm
  - First phase - determine *sets of candidate views* ($CandV_i$) that can be used to rewrite the query
  - Second phase - *construct rewriting* as a union of conjunctive queries (**from the Cartesian product of $CandV_i$**)

# Side Effects of SQR

- Exponential blowup of the rewriting
- Contain queries which could not be evaluated:
  - groups of them can be combined into compact queries
  - some queries return empty results

# Optimizing Individual Rewritings

- Algorithm designed to merge rewritings to remove redundancy
  - *one view* could end up with *several different variable mappings* for different predicates in query
- Input: Two copies of a view with the variable mappings
- Output: A copy of the input view with a merging variable mapping

# Pruning Rewritings with Empty Results

- Based on the following observation:
  - if a rewriting contains a join between two predicates, the result of the rewriting is empty when the *intersection of the value sets* of each variable is empty

Example:

(?v1 p1 ?v2) (?v3 p2 ?v3)

1. Join on ?v2 and ?v3
2. Denote A(?x) be the *value set* of ?x, then

if $A(?v2) \cap A(?v3) = \emptyset$ , then the result of the join is empty

# Pruning Rewritings with Empty Results

- However, using value set is **too expensive** to be practical!!
- Solution: use the KMV(k-minimal values) synopsis
  - estimate the size of the intersection
  - if it is *above a threshold* with certain *probability* with an *error margin*, the predicates are joinable
  - otherwise - issue an *ASK query* on the rewriting (just to make sure!)

# Putting It All Together

The simple way to do it:

1. Construct rewritings using SQR algorithm
2. Apply merging
3. Prune out empty rewritings

...but we can do better!

# Optimizing Generation of Rewritings (OSQR)

- Based on the following observations:
  - different rewritings may *share similar sub-queries* (partial rewritings)
  - we can determine if one such sub-query returns empty results before generating a lot of rewritings
- We are able to remove **entire branches of rewritings** early
  - we do not have to make additional steps for pruning

# Optimized SPARQL Query Rewriting (OSQR)

- Output: Again a **sound** and **complete** rewriting of a query
- Input: Views, Query
- Two-phase algorithm
  - First phase - determine *sets of candidate views* ($CandV_i$) that can be used to rewrite the query
  - Second phase - *construct rewriting* as a union of conjunctive queries, **while in the process merging and pruning rewritings**

# Experimental Setup 1

- *Algorithms and optimization* components written in C++
- *RDF store* - 4store
- Apply SQL translation to the experimental data (MySQL database)
  - use view predicate tables
  - rewriting done by query expansion
- Use *benchmark* LUBM
- 64-bit Linux machine with 2GHz Intel Xeon CPU and 4GB memory

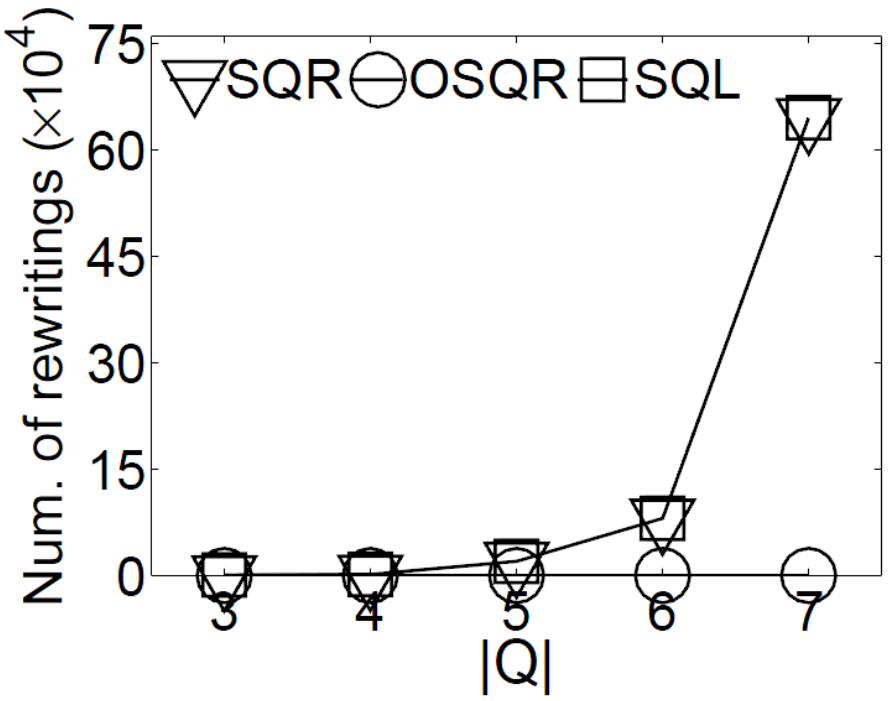
# Key Performance Metrics

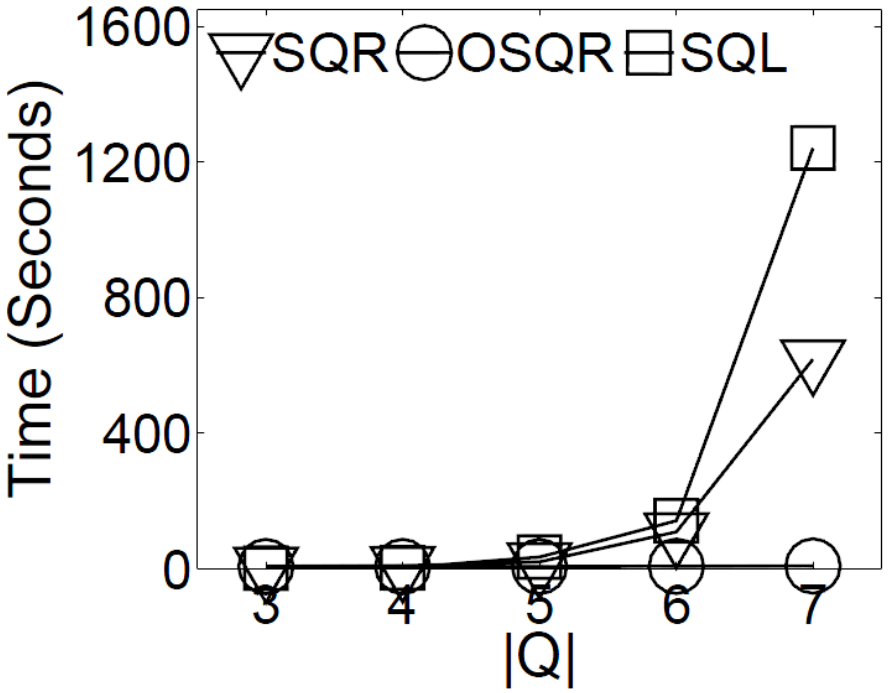
- **Number of rewritings** generated by query rewriting
- **End-to-end evaluation time**
  - including query rewriting and execution

# Experiment 1: Native SPARQL Rewriting vs. SQL Expansion

- Goal: Prove that SQL expansion is *not a viable option*
- Experimental Data:
  - 56 different views
  - each view exposes different aspects of student data
  - each view created using one of seven view templates
  - each consecutive view contains data for fewer students
- Query:
  - complexity of query increased at each iteration (by adding more predicates)

# Results



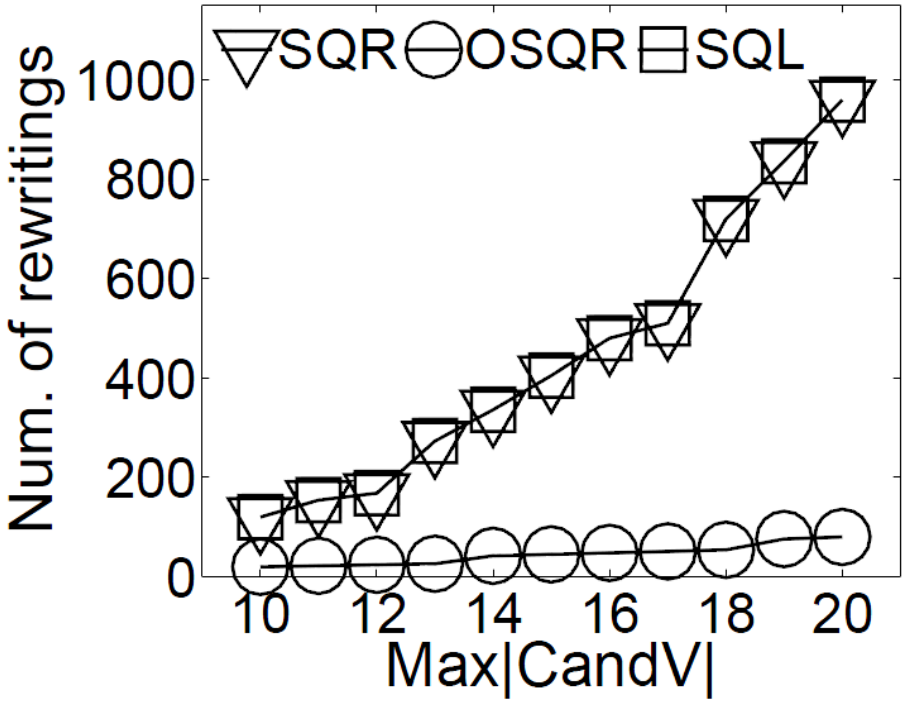



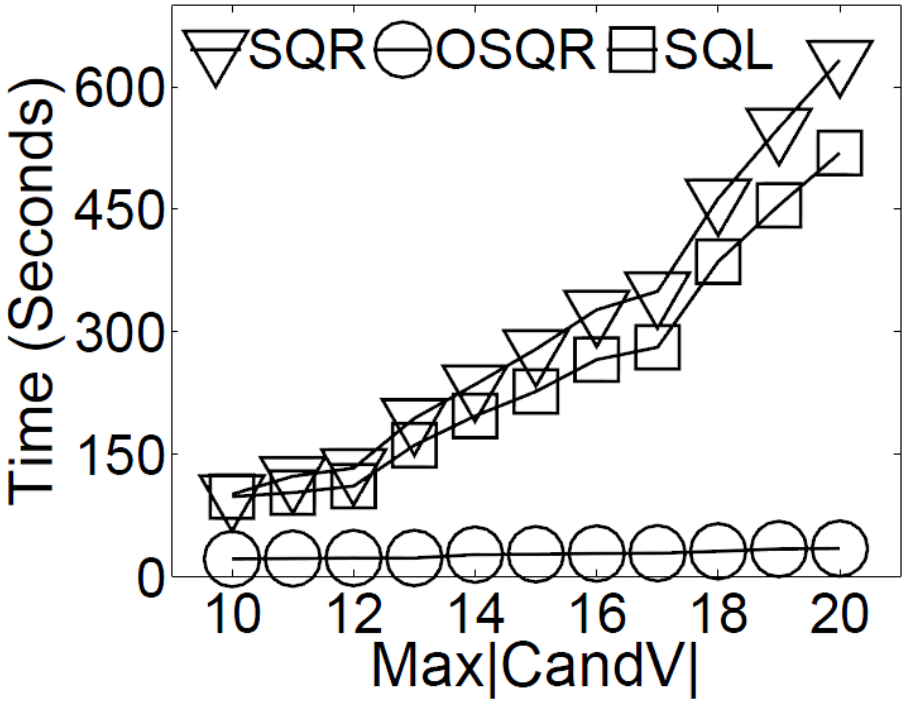

(a) Rewritten queries over query size (b) Eval. time over query size

# Experiment 2: Native SPARQL Rewriting vs. SQL Expansion

- Goal: Show that the previous results hold for different queries and views
- Experimental data:
  - views generated by using one of five templates
  - add consecutive views so that number of candidate views for the largest candidate view set increases linearly
  - student data chosen so that only few rewritings have non-empty results
- Query:
  - query has three predicates for each iteration

# Results



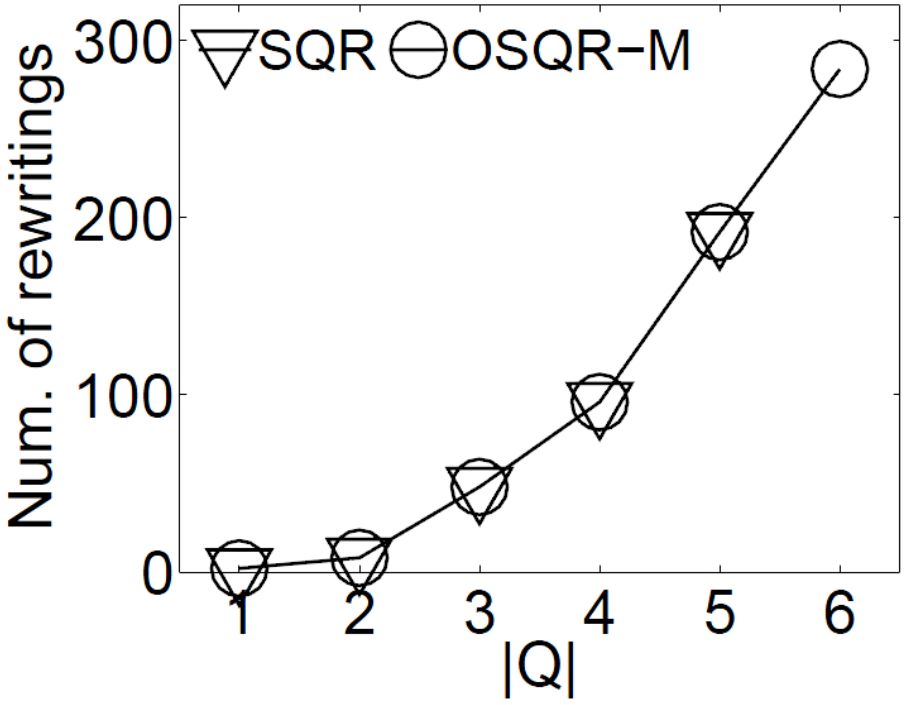



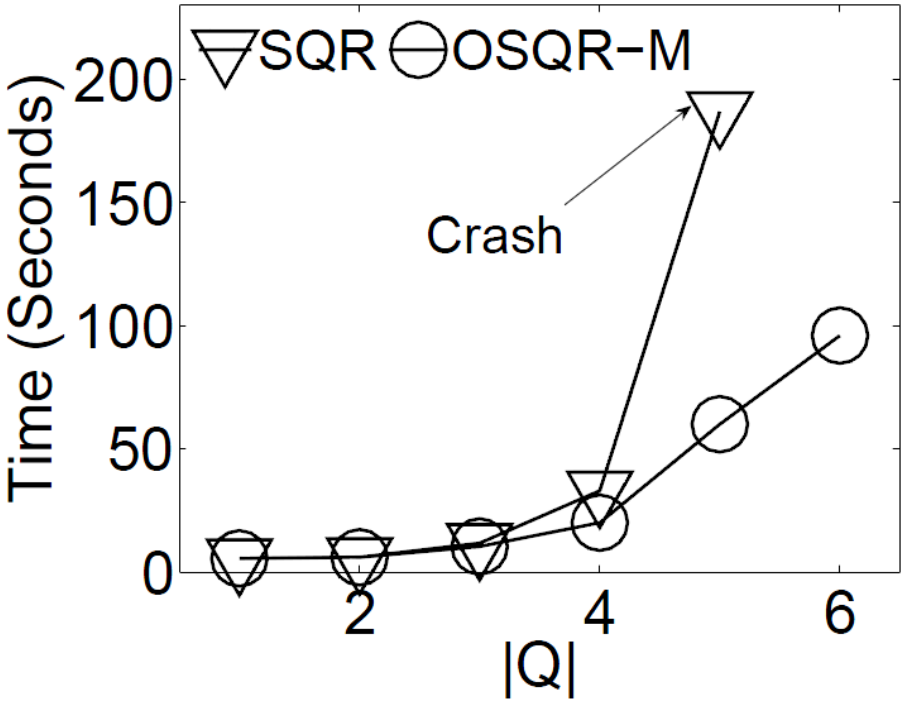

(a) Rewritten queries over max CandV (b) Eval. time over max CandV

# Experiment 3: Optimizing Individual Rewritings

- Goal: Show the effect of the merging views optimization
  - switch off all optimizations except merging views
- Experimental Data:
  - use 6 views, each defining different student data
- Query:
  - use 6 different queries increasingly bringing together the data

# Results




(a) Rewritten queries over query size (b) Eval. time over query size

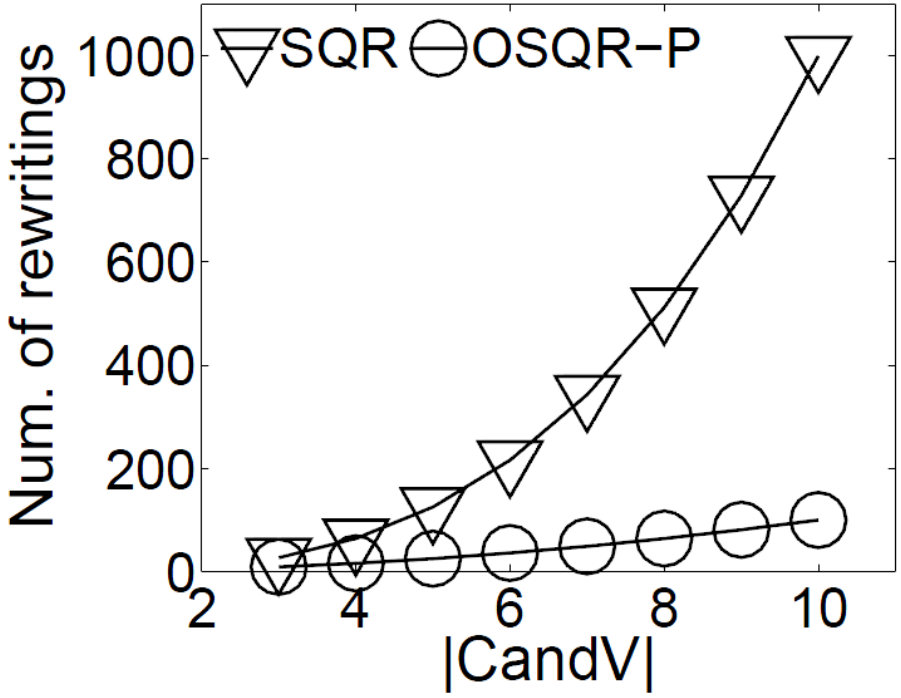
# Experiment 4: Pruning Empty Rewritings

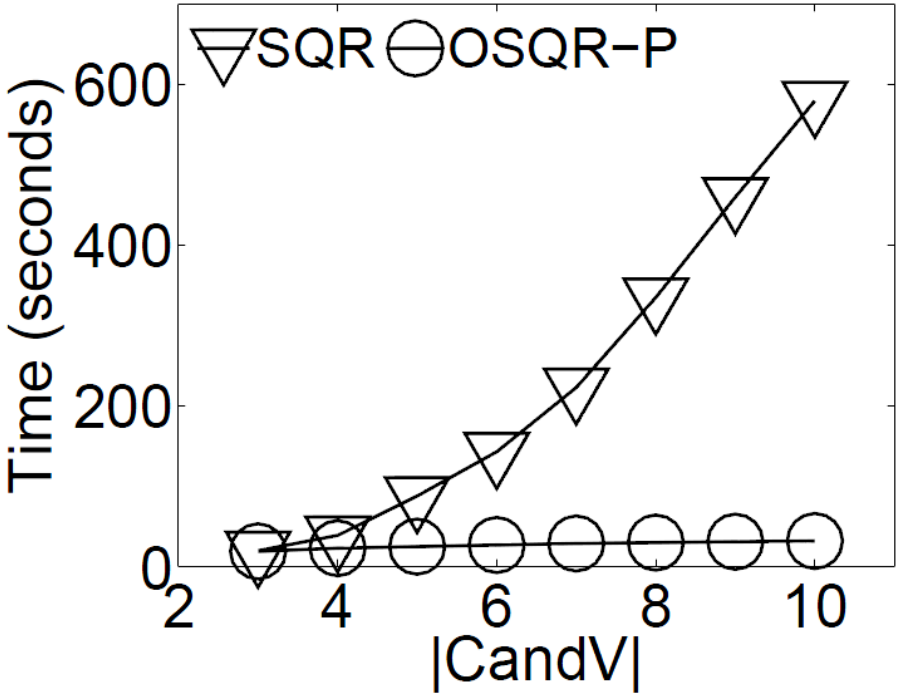
- Goal: Show the effect of pruning empty rewritings
  - switch off all optimizations except pruning
- Experimental Data:
  - 10 views created from one template but contain data for students in different departments (views are non-overlapping)
  - The result description of each view is the same as the graph pattern in the query
- Query:
  - Contains 3 predicates

# Experiment 4: Pruning Empty Rewritings (continued)

- Experiment done in 8 iterations (*i*)
- We have *i+2* views at iteration *i*
- SQR always generates $(i+2)^3$ rewritings at step *i*

# Results




(a) Rewritten queries over max CandV (b) Eval. time over max CandV

# Experimental Setup 2

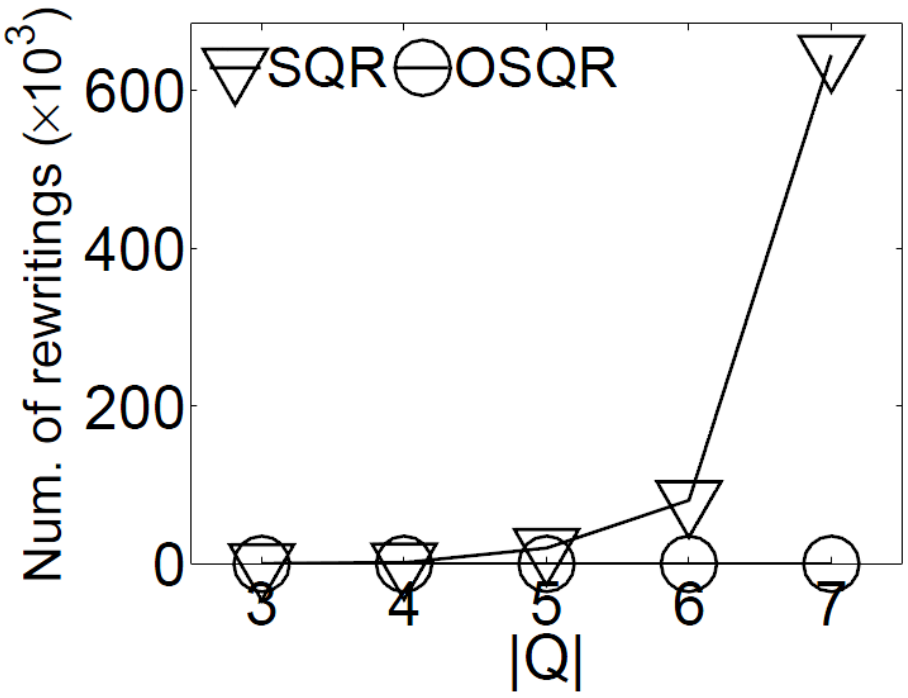
- Demonstrates flexibility of the algorithms
- Change the RDF store **- Jena TDB**
- The rest of the setup settings are the same as Experimental Setup 1

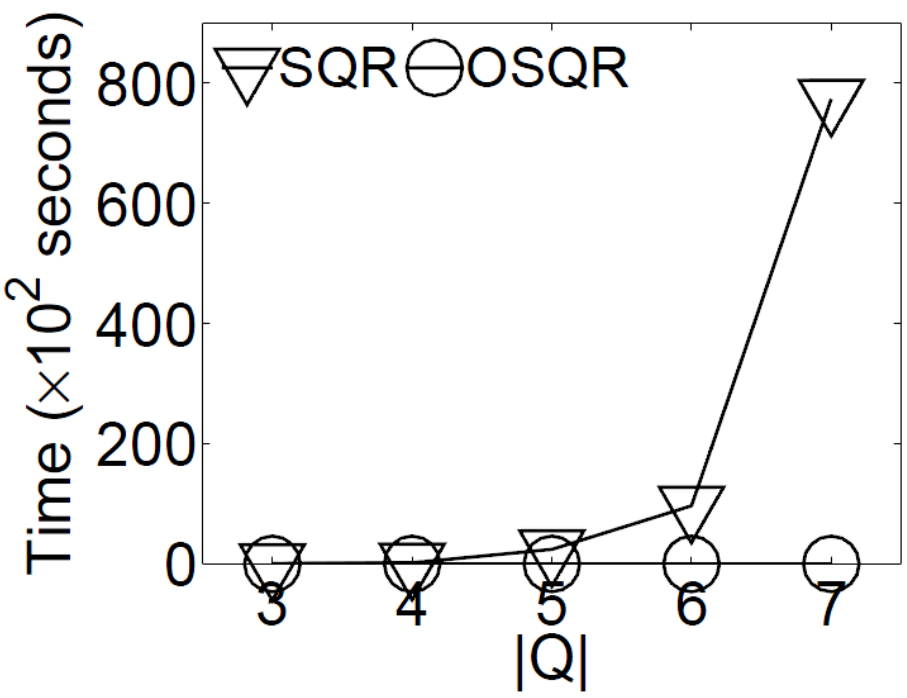
# Experiment 5: SPARQL Rewriting vs. SQR in Jena TDB

- Goal: Demonstrate flexibility of the algorithms

- Exactly the same as Experiment 1
  - need to demonstrate the store-independent property of OSQR

# Results



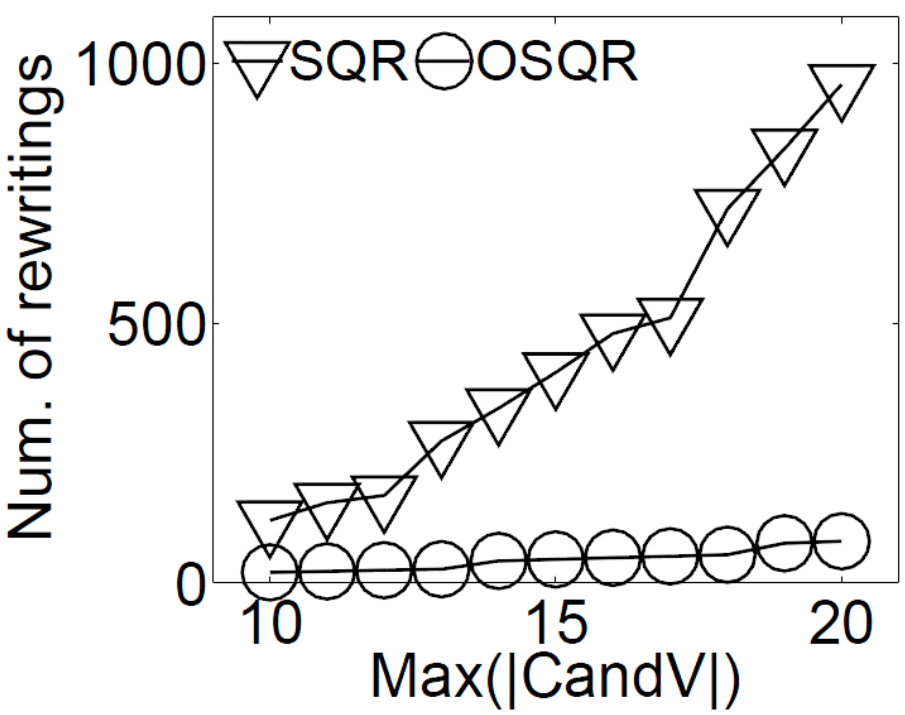

(a) Rewritten queries over query size (b) Eval. time over query size

# Experiment 6: SPARQL Rewriting vs. SQR in Jena TDB

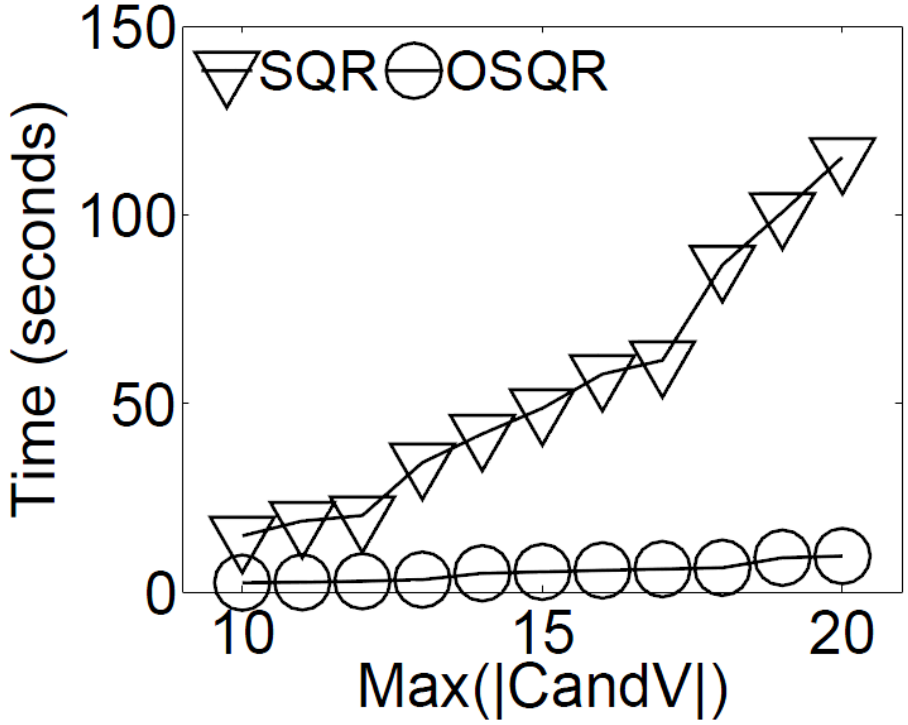
- Goal: Demonstrate flexibility of the algorithms

- Exactly the same as Experiment 2
  - need to demonstrate the store-independent property of OSQR

# Results




(a) Rewritten queries over max CandV




(b) Eval. time over max CandV

# Relevant Work in the Problem Area

- Extensive studies of query rewriting over views in relational and XML cases
- This is the first **native** SPARQL query rewriting algorithm

# Conclusion

Using the approaches and algorithms proposed in the presented articles we can achieve:

- Transparent query processing over heterogeneous data sets with different data models using **SPARQL-RW**
- Efficient native SPARQL query rewriting over virtual views of RDF data using the **OSQR** algorithm

# Thank You for Your Attention!

Any Questions?