Materialized View-Based Processing of RDF Queries

Konstantinos Karanasos

joint work with: *François Goasdoué*, *Julien Leblay* and *Ioana Manolescu*

Ecole thématique BDA

May 18, 2010

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

centre de recherche SACLAY - ÎLE-DE-FRANCE

Leo

Why RDF?

- Schema-relaxed graph-based language for representing information/metadata about Web resources
- Used in a wide variety of applications:
  - DBpedia, WordNet RDF/OWL, DBLP
  - UniProt (and other biological data)
  - Search engines
  - Dmoz
  - Social-tagging, folksonomies

RDF Data Management

- The huge amounts of RDF data out there and the challenges raised by querying them…
- … has drawn the attention of the Database community lately
- Existing methods to store and index RDF:
  - Triple table
  - Property tables
  - Vertical Partitioning (in row- and column-stores)
  - Graph
  - Multiple indexing (Hexastore)
  - Native RDF query engines (RDF-3X)
- Generic approaches ignorant of query workload

Materialized Views for RDF

- Of particular importance for the optimization of RDF queries:

  the presence of many joins over a potentially huge table is intrinsic

But:

* Someone has to choose the *appropriate* views!

Problem Definition

- Given an initial query workload over RDF data,
- *choose a set of views to materialize,*
- in order to maximize a *quality function* (i.e., minimize the total cost of evaluating the queries and of maintaining the views).

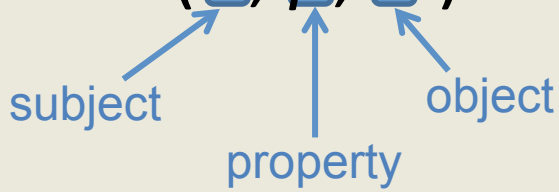
- Variants:
  - Space constraints
  - Query weights

Outline

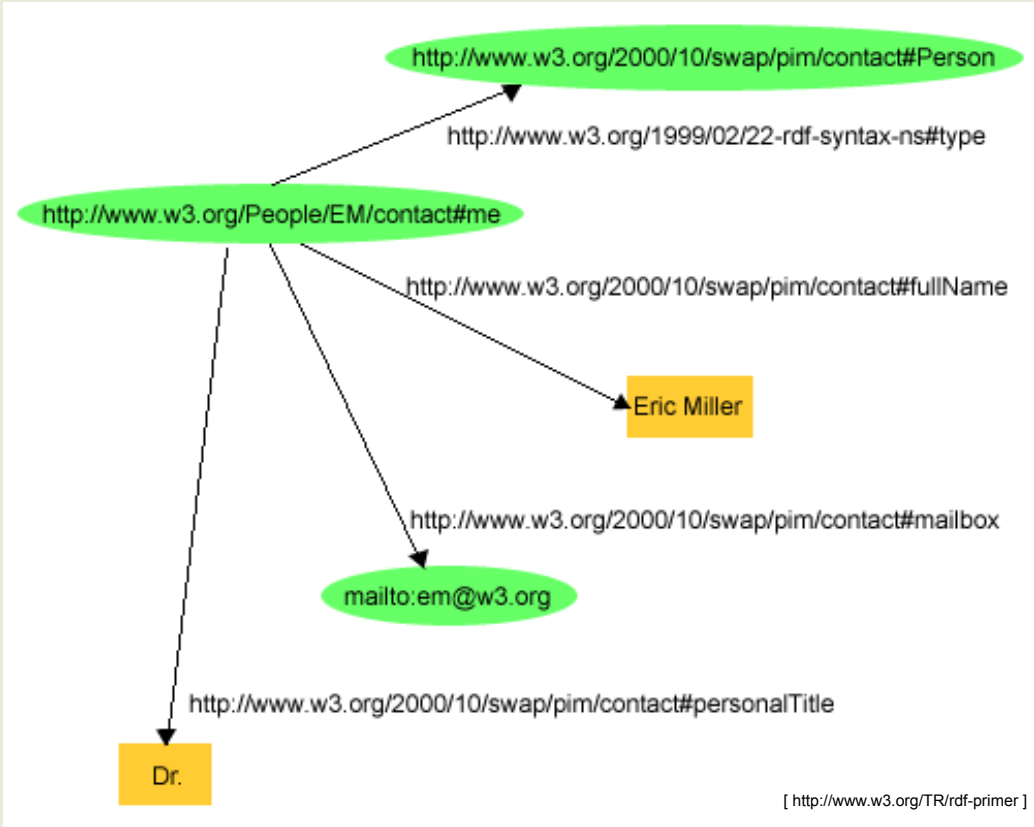
- Introduction
- ➢ Problem Modeling
- Search Space
- Reasoning with RDFS
- Experimental Evaluation
- Future Work & Conclusion

Data Model: RDF

- RDF triples: triple atoms of the form (*s*, *p*, *o*)

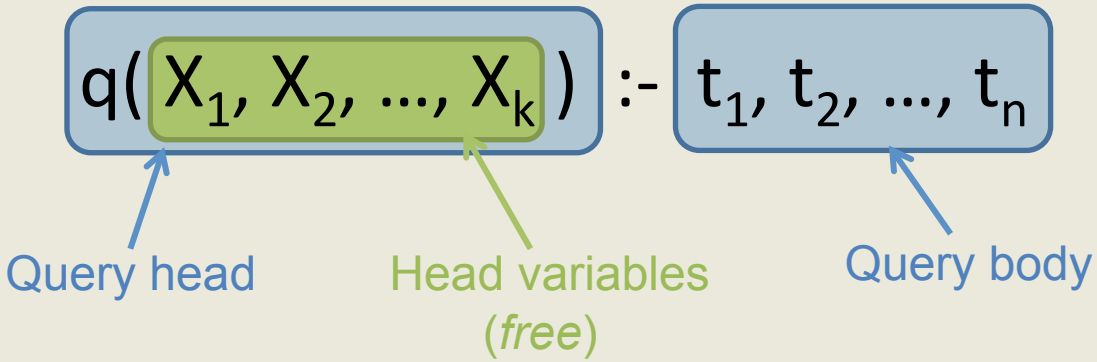


- *s*, *p*: URIs
- *o*: URIs or literals



Query and View Model

- Conjunctive RDF queries:



- In the query body, t_x is an RDF triple pattern of the form *t(s, p, o)*
- *s*, *p* and *o*: variables (free/existential) or constants
- A view is defined exactly as a query
- We consider minimal queries (so will be our views)

Conjunctive RDF Queries: examples

- $q1(X_2, X_5)$:-

  $t(X_1, hasName, X_2), t(X_1, X_3, postImpres),$

  $t(X_1, hasPainted, X_4), t(X_4, hasTitle, X_5),$

  $t(X_4, isExpIn, X_6), t(X_6, isNamed, moma)$

- $q_2(Y_3, Y_5)$:- $t(Y_1, hasCountry, france),$

  $t(Y_1, belongsTo, postImpres), t(Y_1, hasPainted, Y_2),$

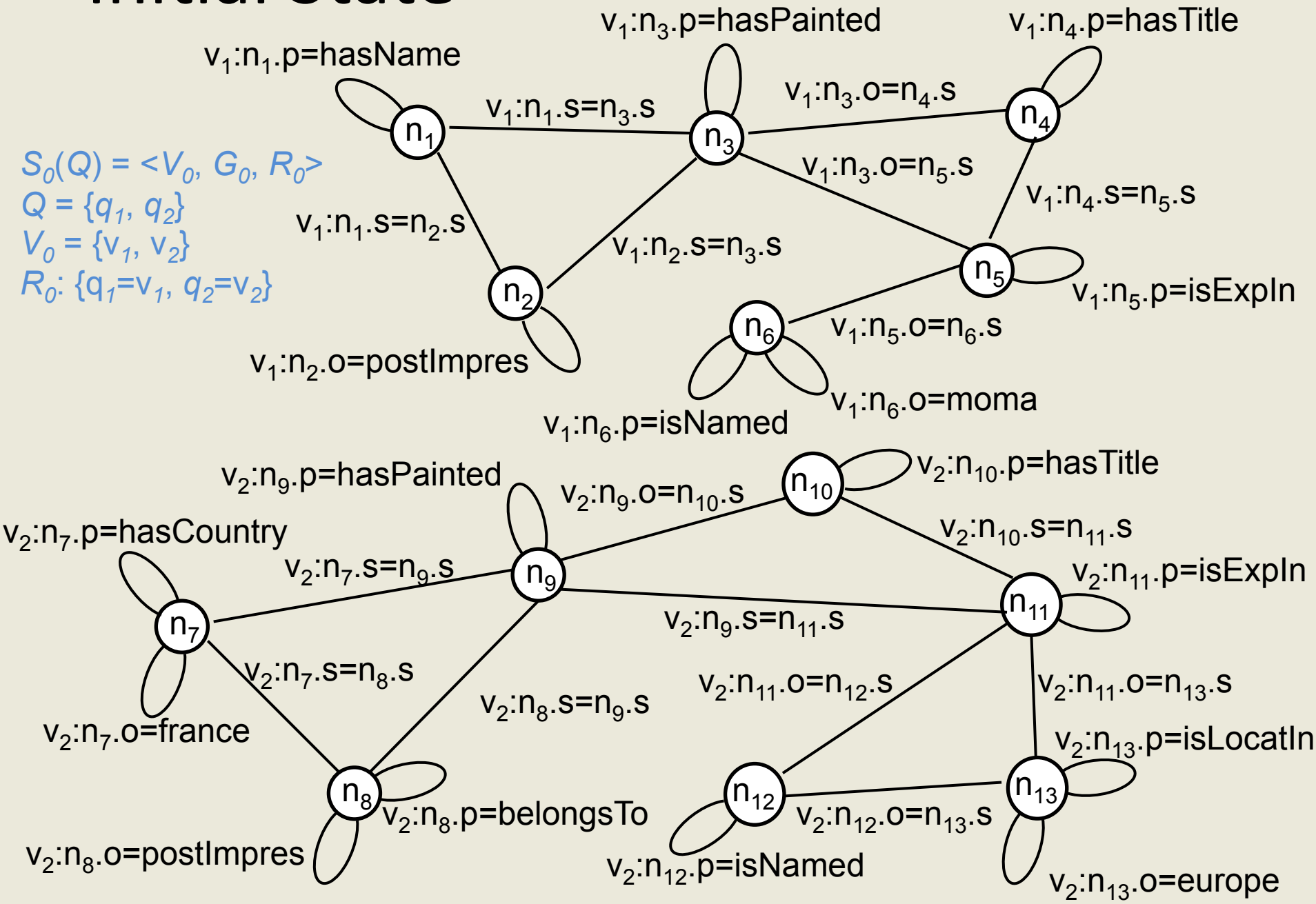
  $t(Y_2, hasTitle, Y_3), t(Y_2, isExpIn, Y_4),$

  $t(Y_4, isNamed, Y_5), t(Y_4, isLocatIn, europe)$

Problem Modeling

- We address RDF View Selection as a *state optimization problem* (inspired by [TS97])
- State S_i (Q) = <V_i, G_i, R_i>
  - Q: the initial set of queries
  - V_i: set of views currently proposed for materialization
  - G_i: the corresponding graph of V_i
  - R_i: set of rewriting expressions

Initial State



Outline

- Introduction
- Problem Modeling
- ➢ Search Space
- Reasoning with RDFS
- Experimental Evaluation
- Future Work & Conclusion

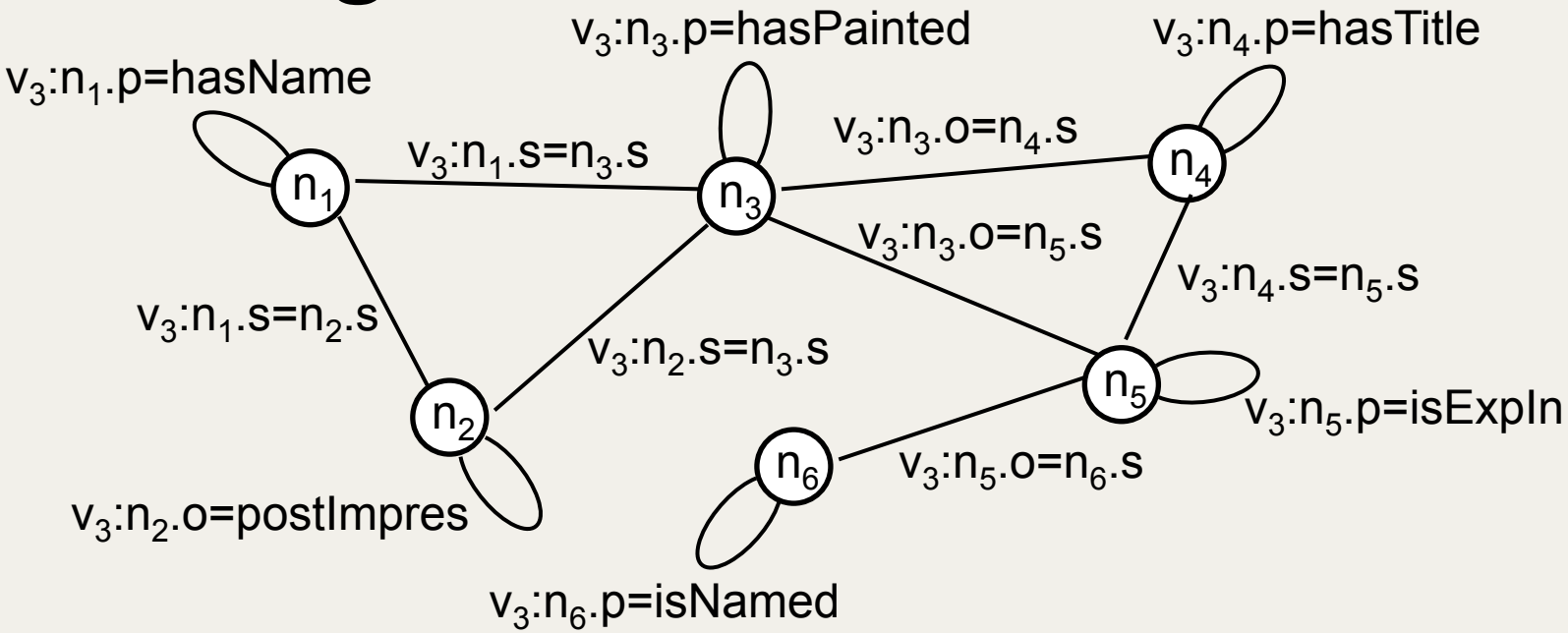
Exploring the Search Space - Transitions

- Starting from S_0, we apply a set of transformation rules and obtain new states…
- Generally:

  $S(Q) = \langle Q, V, G, R \rangle \rightarrow S'(Q) = \langle Q, V', G', R' \rangle$

- Transitions (complete and minimal):
  - Selection Edge Cut
  - Join Edge Cut
  - View Fusion

Selection Edge Cut



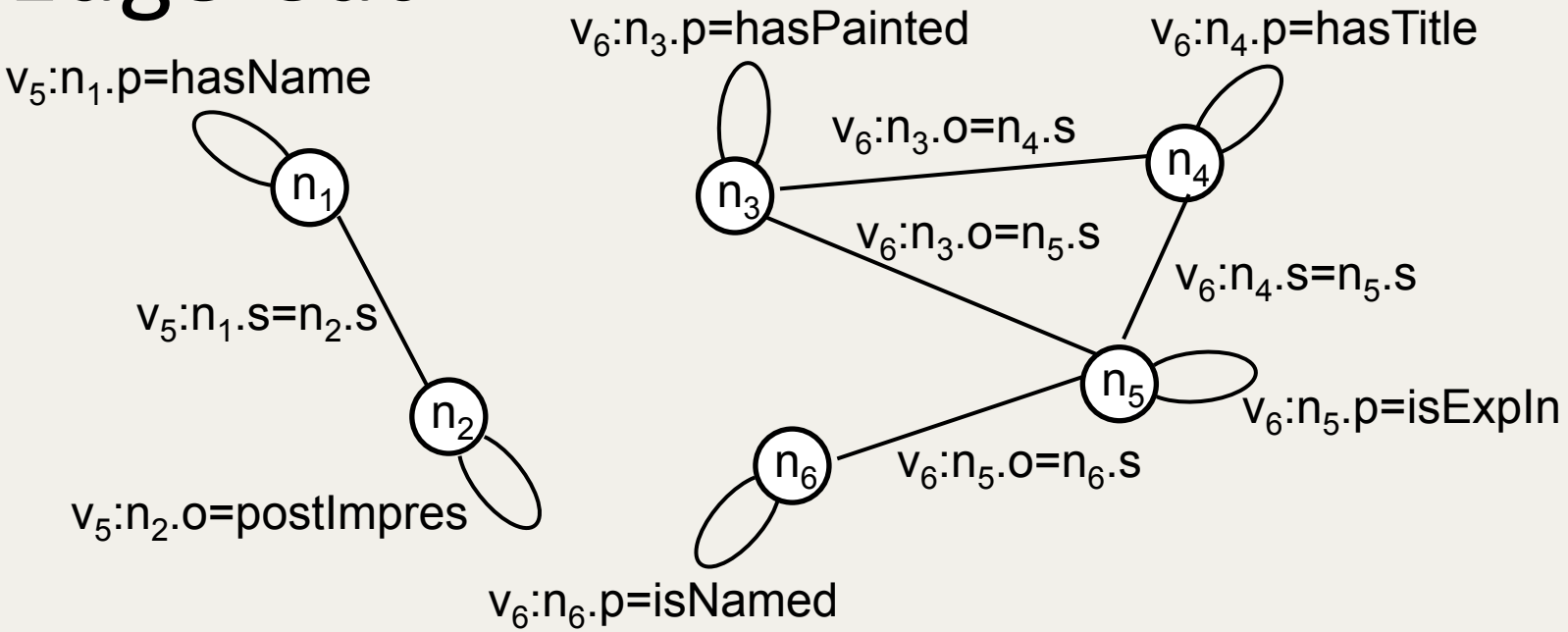
$S_1(Q) = \langle V_1, G_1, R_1 \rangle$

$Q = \{q_1, q_2\}$

$V_1 = \{v_2, v_3\}$

$R_1 = \{q_1 = \sigma_{n6.o=moma}(v_3), q_2 = v_2\}$

Join Edge Cut



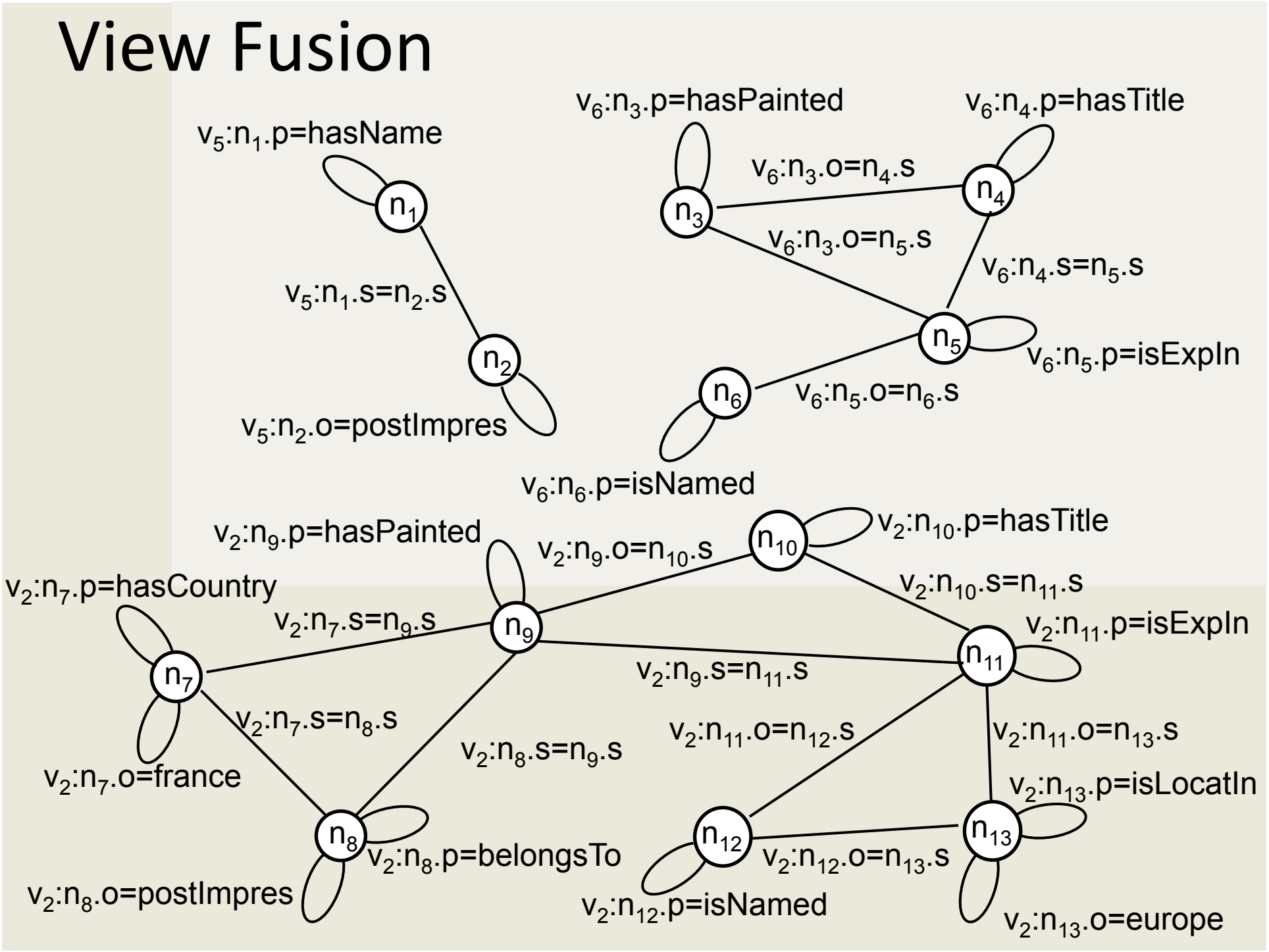
$S_2(Q) = <V_2, G_2, R_2>$

$Q = \{q_1, q_2\}$

$V_2 = \{v_2, v_5, v_6\}$

$R_2 = \{q_1=\sigma_{n2.s=n3.s}(v_5 \times v_6), q_2=v_2\}$

View Fusion


v5:n1.p=hasName
n1
v5:n1.s=n2.s
n2
v5:n2.o=postImpres
v6:n3.p=hasPainted
v6:n4.p=hasTitle
v6:n3.o=n4.s
n3
n4
v6:n3.o=n5.s
v6:n4.s=n5.s
n5
v6:n5.p=isExpIn
n6
v6:n5.o=n6.s
v6:n6.p=isNamed
v2:n9.p=hasPainted
v2:n9.o=n10.s
n10
v2:n10.p=hasTitle
v2:n10.s=n11.s
v2:n7.p=hasCountry
v2:n7.s=n9.s
n9
v2:n11.p=isExpIn
n7
n11
v2:n9.s=n11.s
v2:n7.s=n8.s
v2:n11.o=n12.s
v2:n11.o=n13.s
v2:n8.s=n9.s
v2:n7.o=france
v2:n13.p=isLocatIn
n8
n12
n13
v2:n8.p=belongsTo
v2:n12.o=n13.s
v2:n8.o=postImpres
v2:n12.p=isNamed
v2:n13.o=europe

Search Strategies

- Naïve exhaustive strategy
- Stratified strategies (every path is SC* JC* VF*):
  - BFS-like (perform all SC, then JC, then VF) – memory shortcomings
  - DFS ("get rid of" each state as soon as possible) – less memory demand, more probable to find a good solution quickly
- Quality Estimators:
  - average constant ratio
  - join count in rewritings
  - view popularity
  - selectivity estimation

Search Strategies - Heuristics

- Aggressive view fusion
- Stop (pruning) conditions
  - if a view is the triple table
  - if a view contains no constants
  - if we have exceeded a given number of views
- "Pull & push selections":
  - initially remove a number of *non-promising* constants
  - try to *put them back* at the end

Outline

- Introduction
- Problem Modeling
- Search Space
- ➢ Reasoning with RDFS
- Experimental Evaluation
- Future Work & Conclusion

RDF Schemas

- Define semantic relations between classes and properties (*subClassOf*, *domain*, *range*, *subPropertyOf*)
- Two approaches:
  - Compile the knowledge of the schema into the data (add all implicit triples) – no modifications needed
  - Compile the knowledge of the schema into the queries (rewriting based) – slightly modify Q, S_0 and R_i

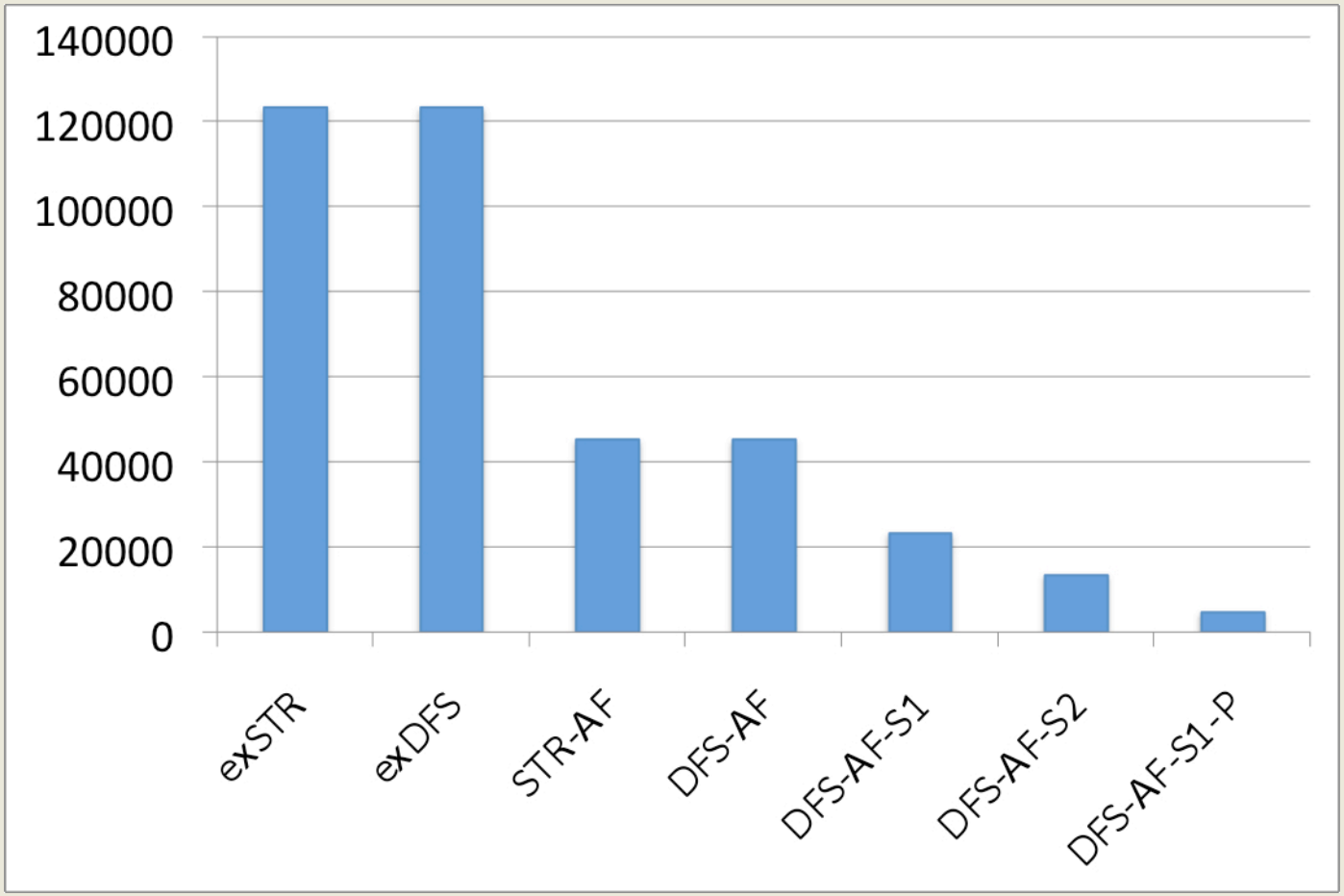
Outline

- Introduction
- Problem Modeling
- Search Space
- Reasoning with RDFS
- ➢ Experimental Evaluation
- Future Work & Conclusion

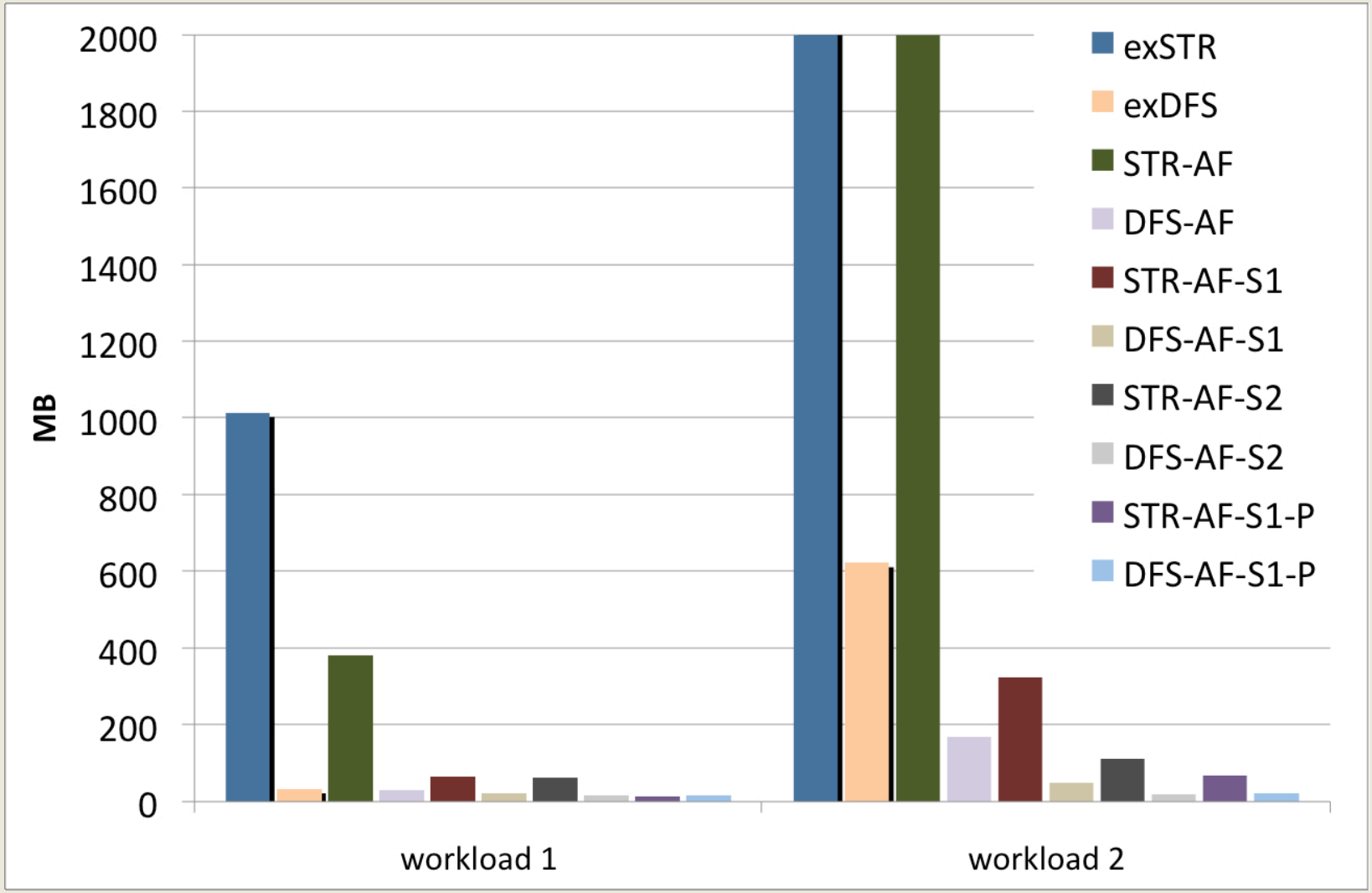
Some details…

- Fully implemented in Java
- PostgreSQL as an RDBMS
- Datasets used for experiments
  - Barton dataset (35 million triples after cleaning)
  - Yago (40 million triples)

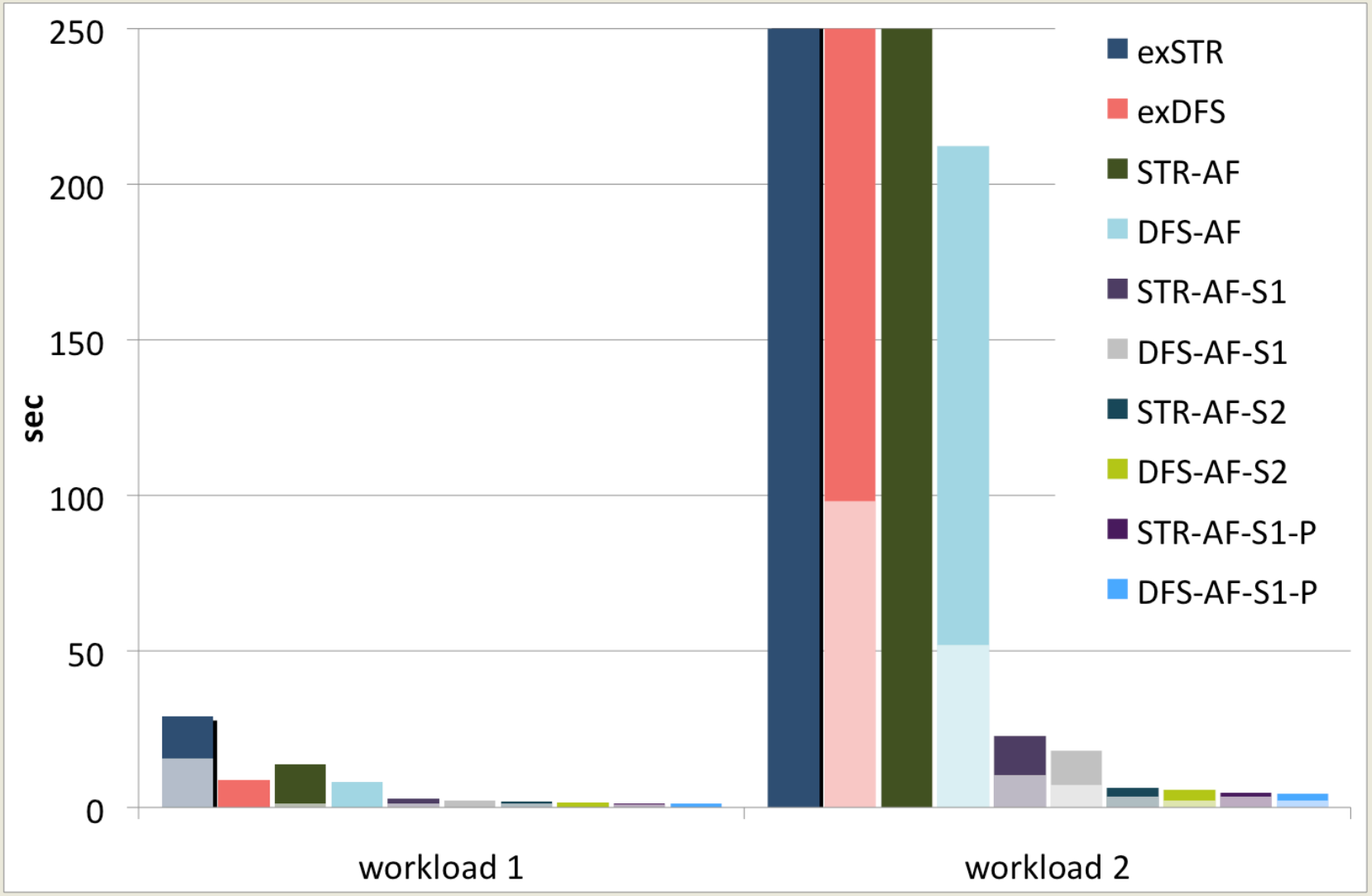
Strategies Evaluation (1)



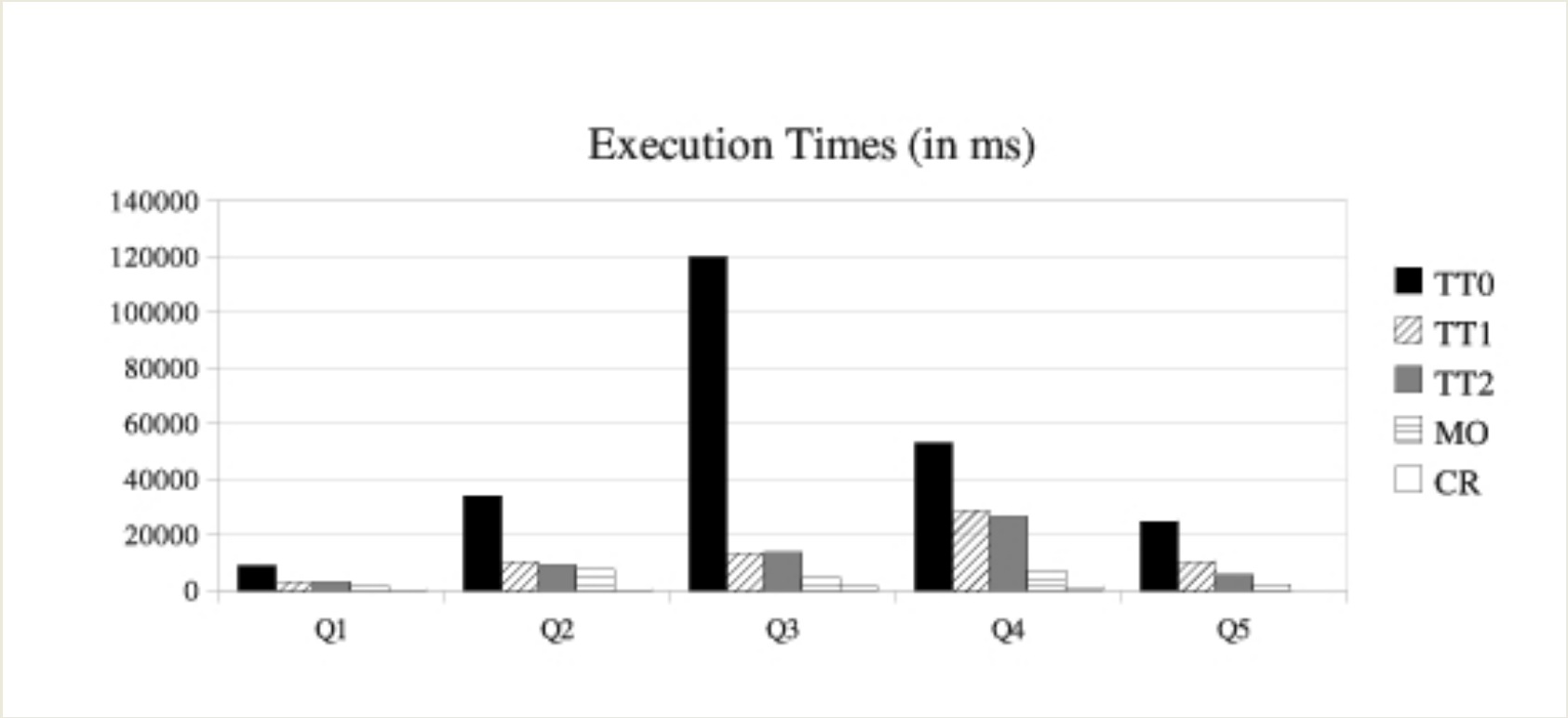
Strategies Evaluation (2)



Strategies Evaluation (3)



Query Execution Times



Outline

- Introduction
- Problem Modeling
- Search Space
- Reasoning with RDFS
- Experimental Evaluation
- ➢ Future Work & Conclusion

What's left to be done?

- RDF View Selection:
  - Further heuristics
  - More sophisticated usage of selectivity estimation
  - Experiment on even bigger datasets and heavy query workloads
  - Switch to a distributed (P2P) setting
  - Adapt the views as new queries arrive

Concluding…

- View Selection problem for RDF queries and data as a state optimization problem (transformation rules)
- Strategies and heuristics to efficiently navigate through the search space
- Support for reasoning on the data via RDFS
- Experimental evaluation

Thank you!