Multidatabase Distributed RDF Graph Management

Peng Peng
Hunan University, China
hnu16pp@hnu.edu.cn

Outline

1. Background
2. Partitioning-agnostic Approaches
3. Federated Systems
4. Conclusions

Part 1

Background

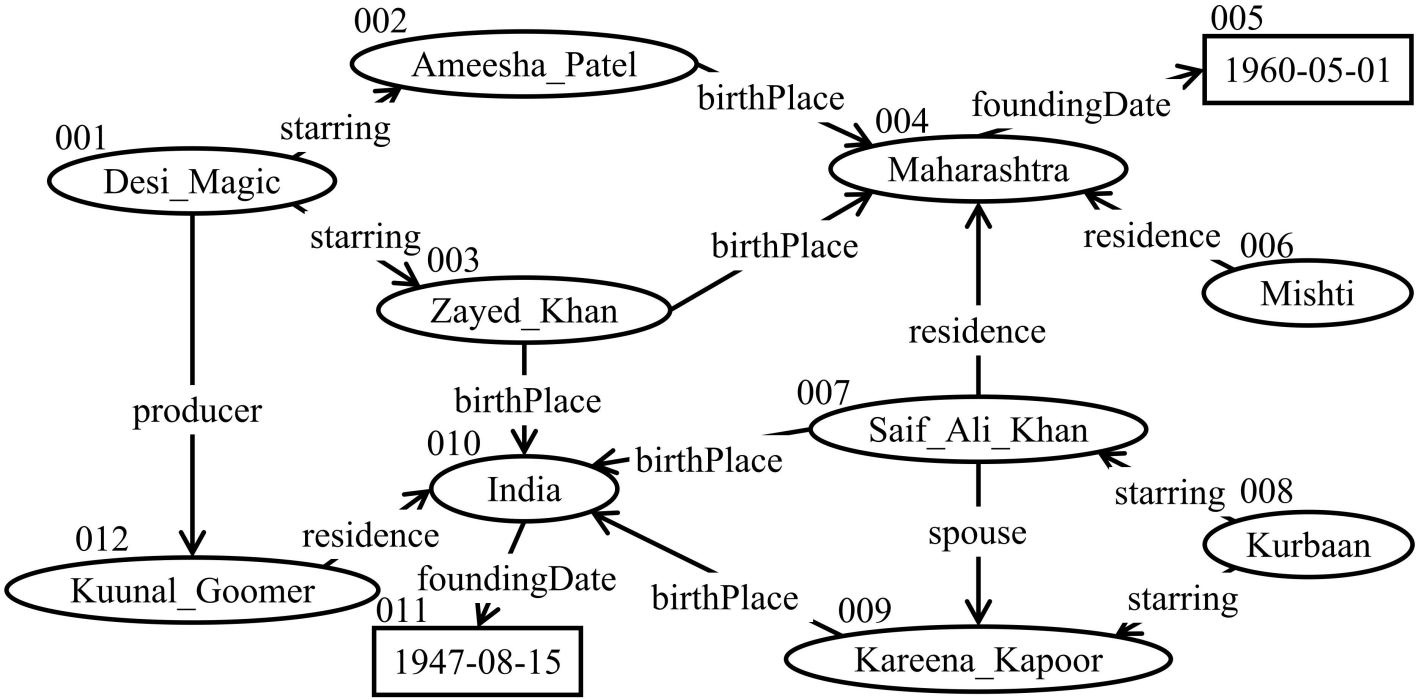
RDF Introduction

Resource Description Framework

- It is a data model proposed widely used for knowledge bases
- Everything is an uniquely named resource
- Properties of resources can be defined
- Relationships with other resources can be defined

RDF Graph

An RDF dataset can be represented as a graph

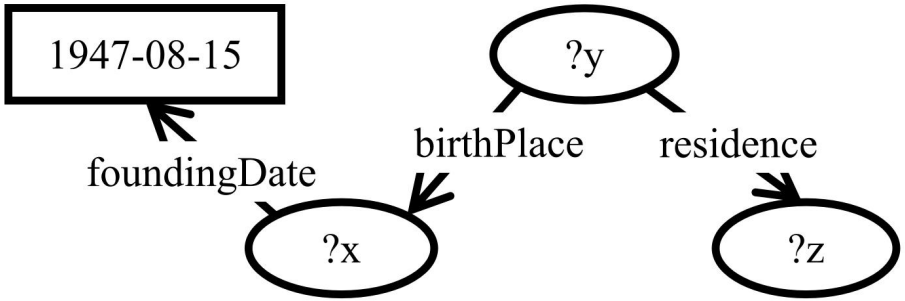


RDF Query Model

Query Model - ***S***PARQL ***P***rotocol ***a***nd ***R***DF ***Q***uery ***L***anguage

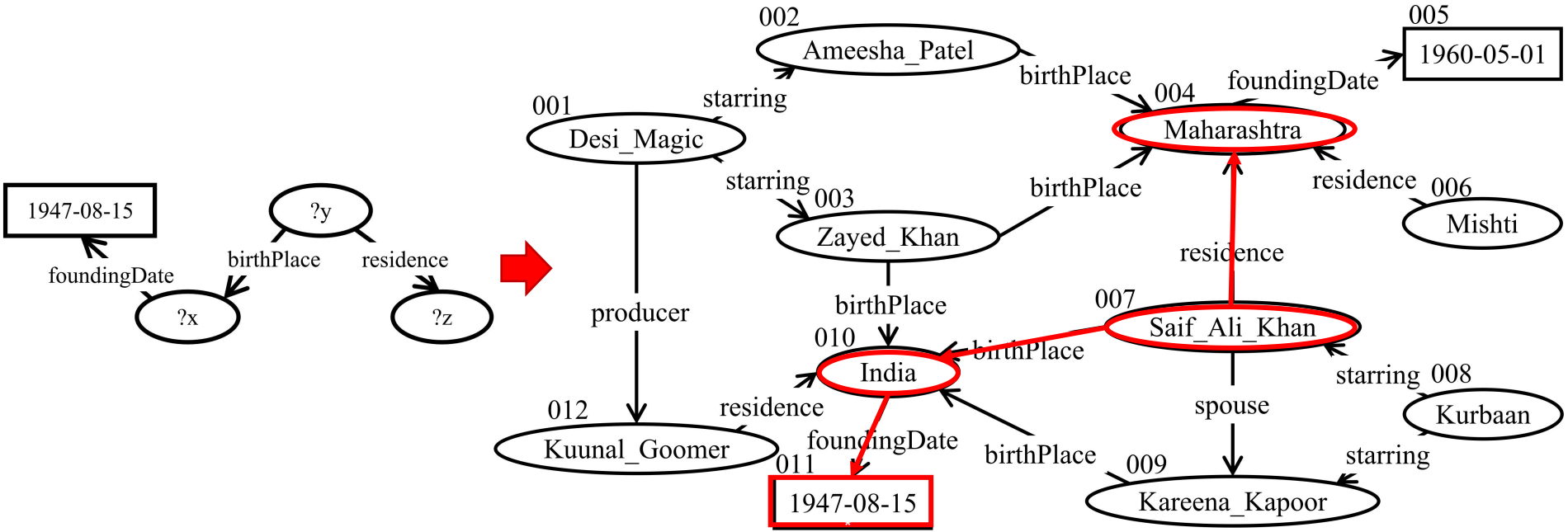
A SPARQL query is a set of triple patterns with variables

```
Select ?x where {
  ?y residence        ?z .
  ?y birthPlace       ?x .
  ?x foundingDate     1947-08-15 . }
```



RDF Query Model

Answering SPARQL query = subgraph matching using homomorphism



Large RDF Graphs

Now, RDF datasets become larger and larger

Max-Planck-Institute

284 million triples

Metaweb Company
acquired by Google in 2010

2.4 billion triples

Leipzig University
University of Mannheim
OpenLink Software

9.5 billion triples

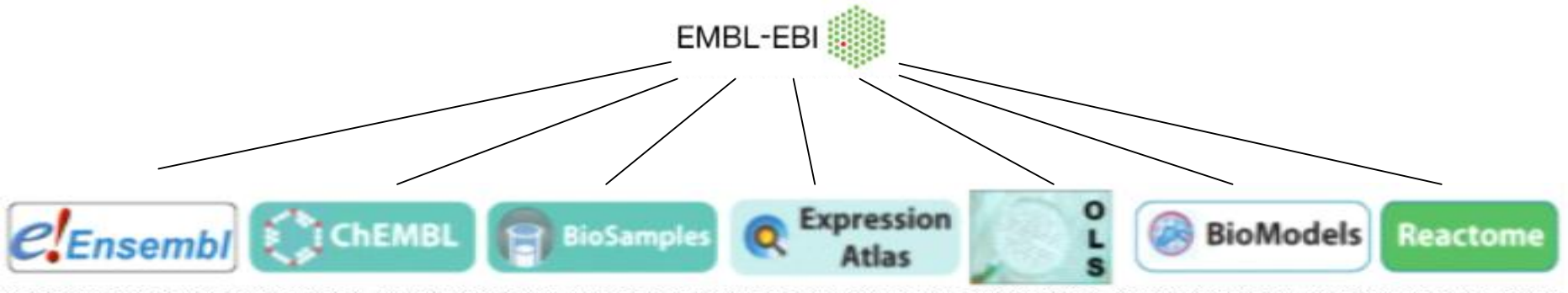
It is important to design a distributed RDF system to manage the large RDF dataset.

Part 2

Partitioning-agnostic Approaches

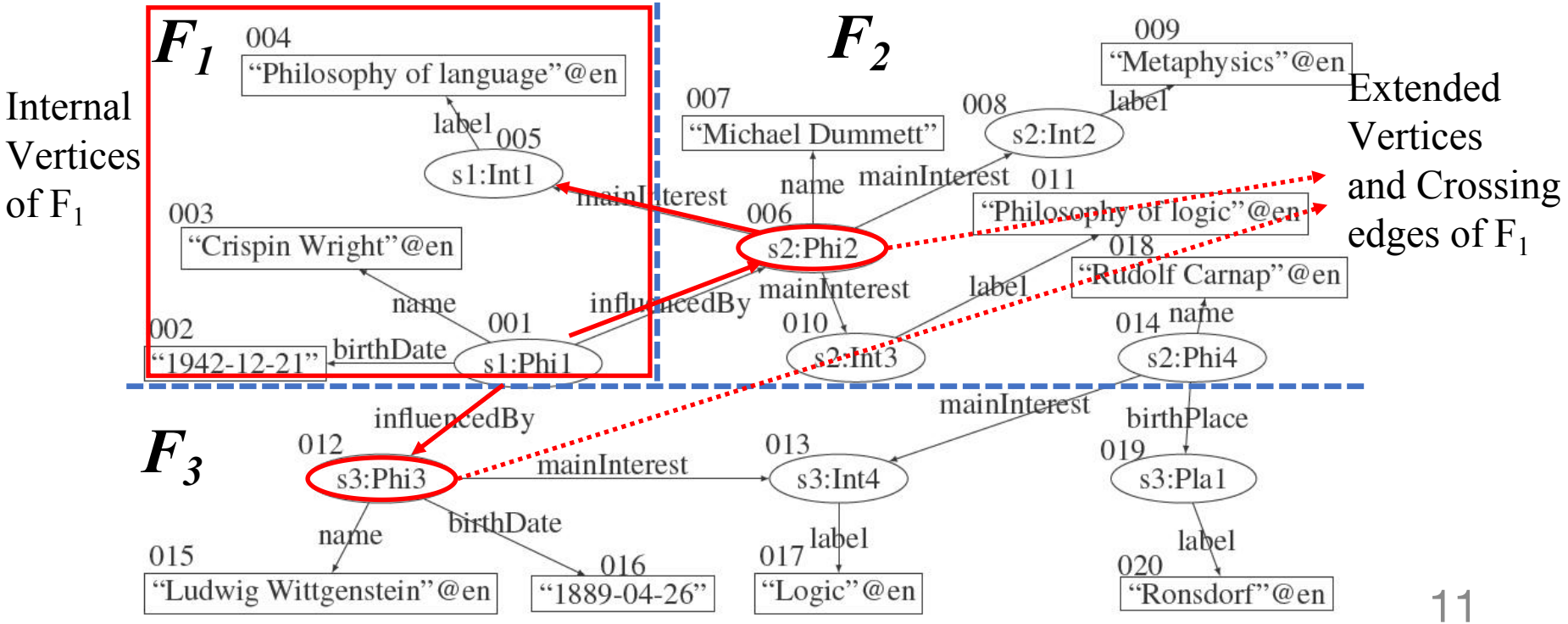
Partitioning-agnostic Application

- In some applications, the RDF graph partitioning strategy is not controlled by the distributed RDF system itself



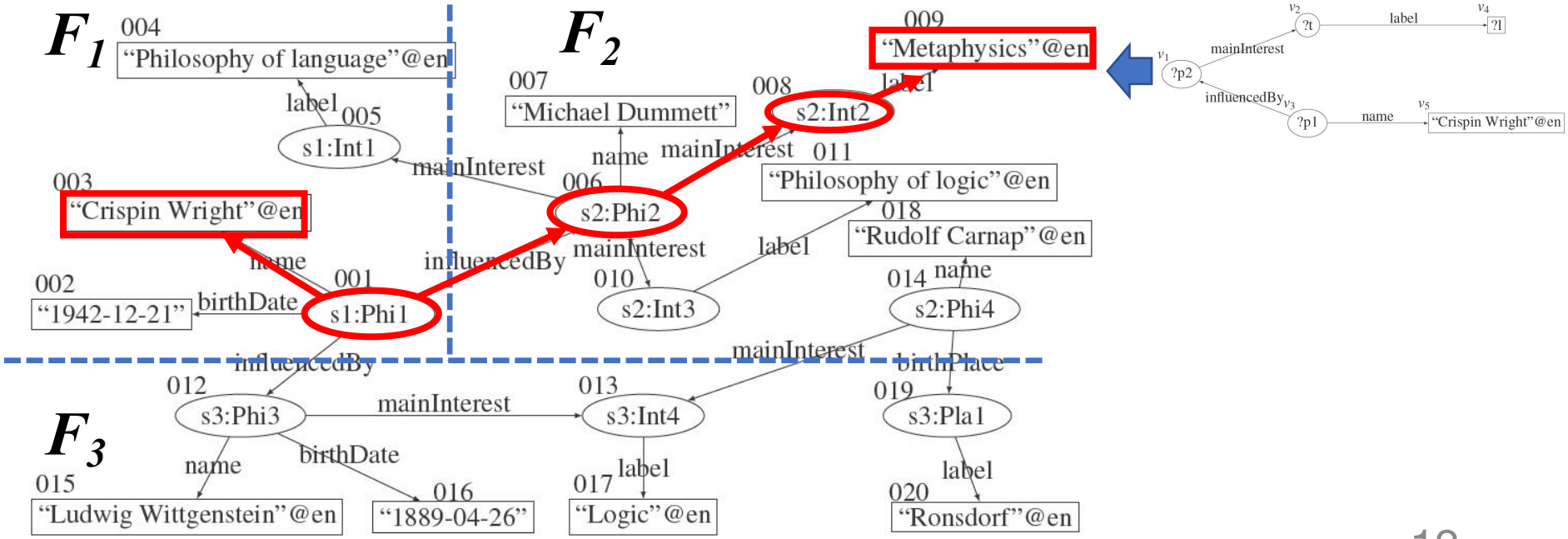
Distributed RDF Graphs

- An RDF graph is vertex-disjoint partitioned through different sites



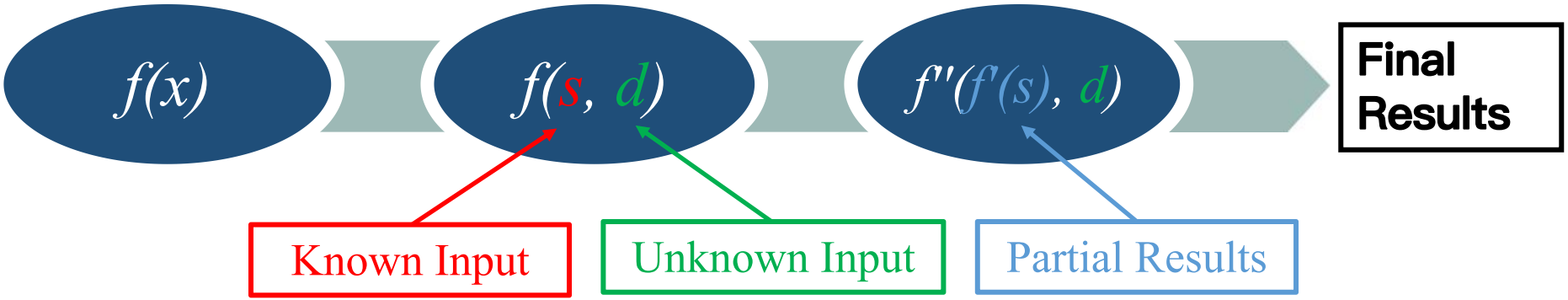
Partial Evaluation

- "Partial evaluation" framework is used for distributed SPARQL query evaluation



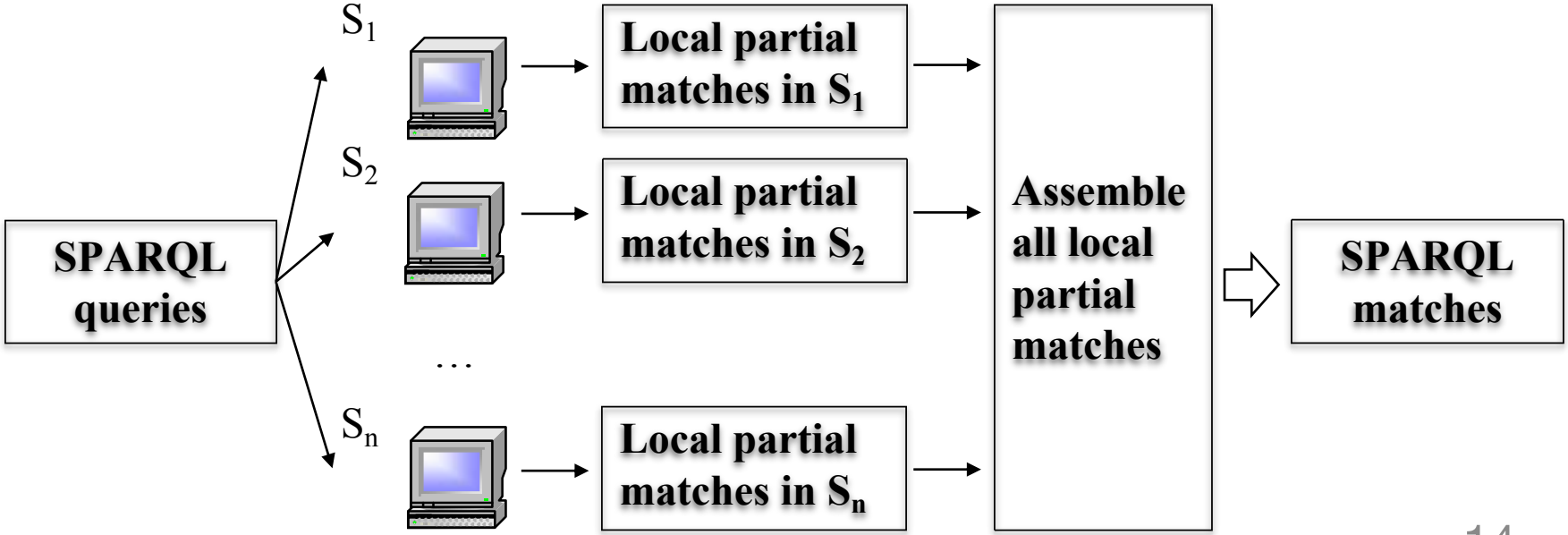
Partial Evaluation

- Partial Evaluation is a framework widely used for evaluating queries on distributed graphs



Partial Evaluation Framework [Peng et al., VLDBJ 2016, ICDE 2019]

- We propose a distributed RDF system, named gStore^D, to evaluate SPARQL queries in a "partial evaluation" framework.

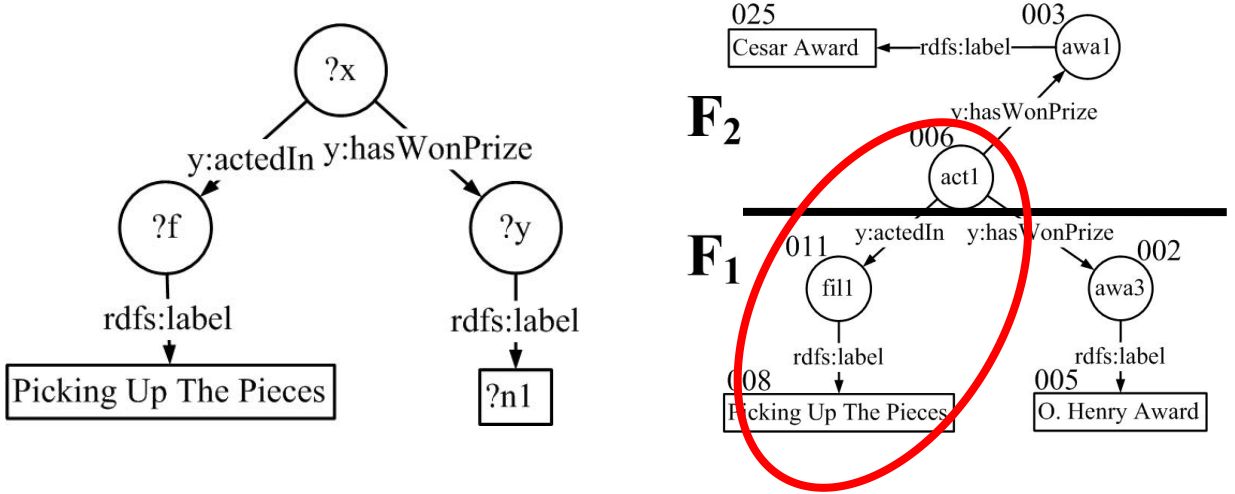


Definition of Local Partial Match

- Given a query Q with n vertices $\{v_1, ..., v_n\}$ and a subgraph PM with m vertices $\{u_1, ..., u_m\}$ ($m \leq n$) in a fragment F_k, PM is a local partial match in fragment F_k if and only if there exists a function $f : \{v_1, ..., v_n\} \rightarrow \{u_1, ..., u_m\} \cup \{NULL\}$, where the following conditions hold:
  1. If $f(v_i)$ is an internal vertex in F_k and v_j is a neighbor, $f(v_j) \neq NULL$;
  2. If $f(v_i)$ and $f(v_j)$ are both internal vertices in PM, then there exists a weakly connected path π between v_i and v_j in Q and each vertex in π maps to an internal vertex of F_k in PM.

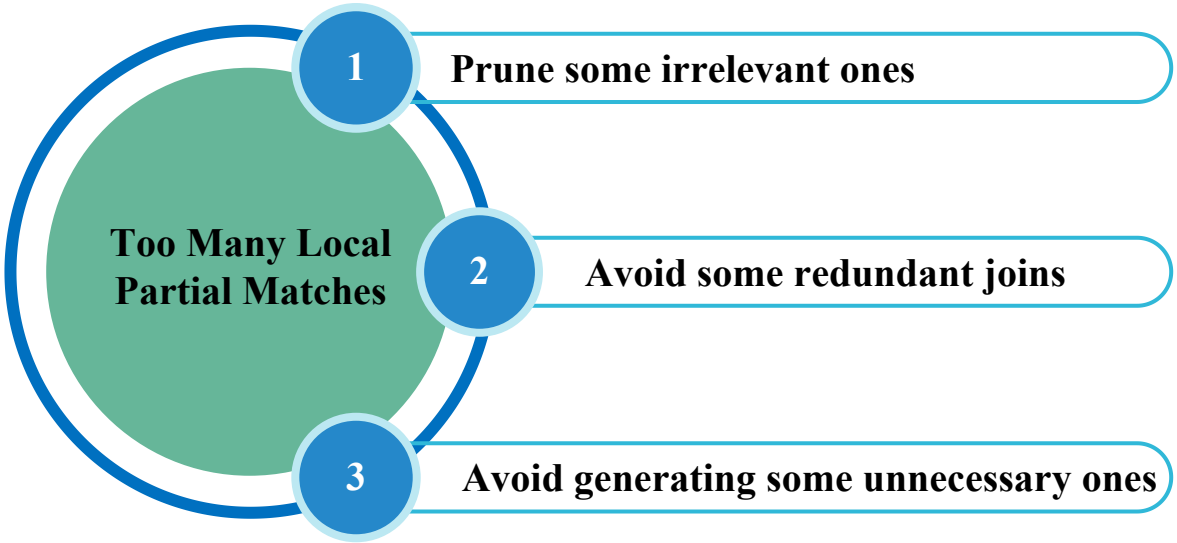
Example Local Partial Matches

- We propose a distributed RDF system, named gStore^D, to evaluate SPARQL queries in a “partial evaluation” framework.



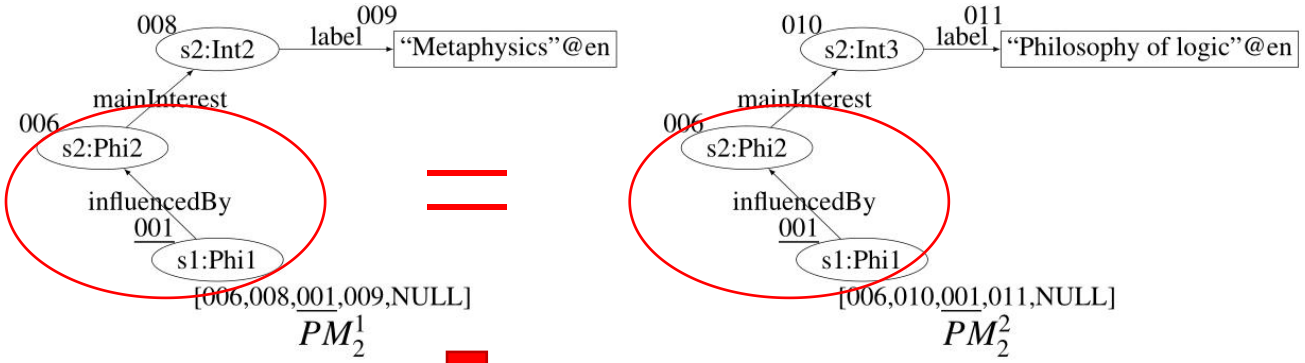
There may be too many local partial matches!!!

Three Optimizations



Properties of Local Partial Matches

- Two local partial matches of the same crossing edges from the same fragment have the same structures.



Local Partial Match Equivalence Class (LEC)

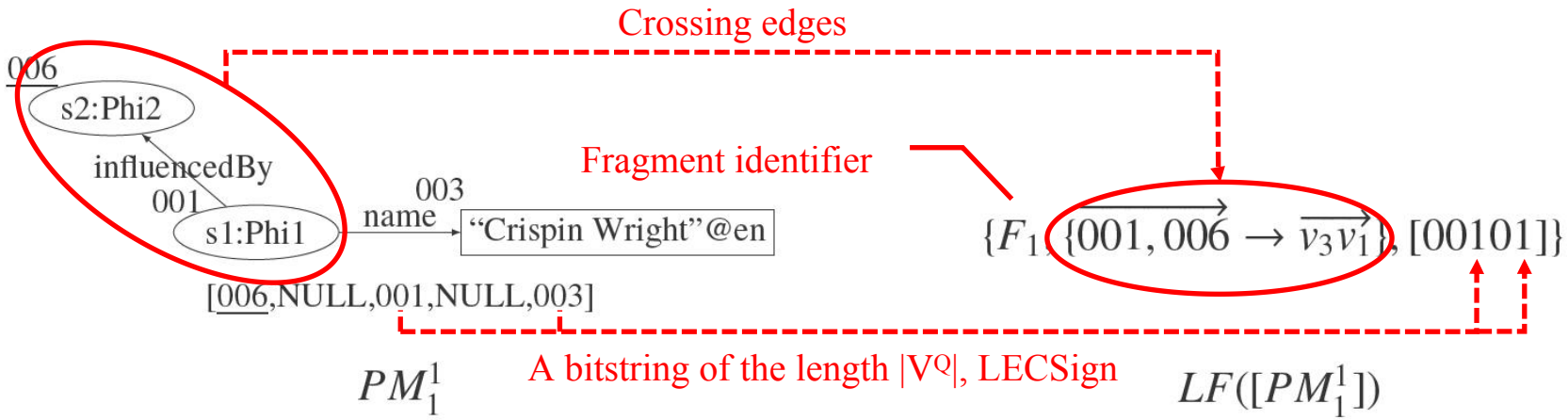
- All local partial matches of the same crossing edges from the same fragment are put into the same class

$$PM_2^1 \sim PM_2^2$$

$$[PM_2^1] = \{PM_2^i \mid PM_2^i \sim PM_2^1\}$$
$$= \{PM_2^1, PM_2^2\}$$
$$= [PM_2^2]$$

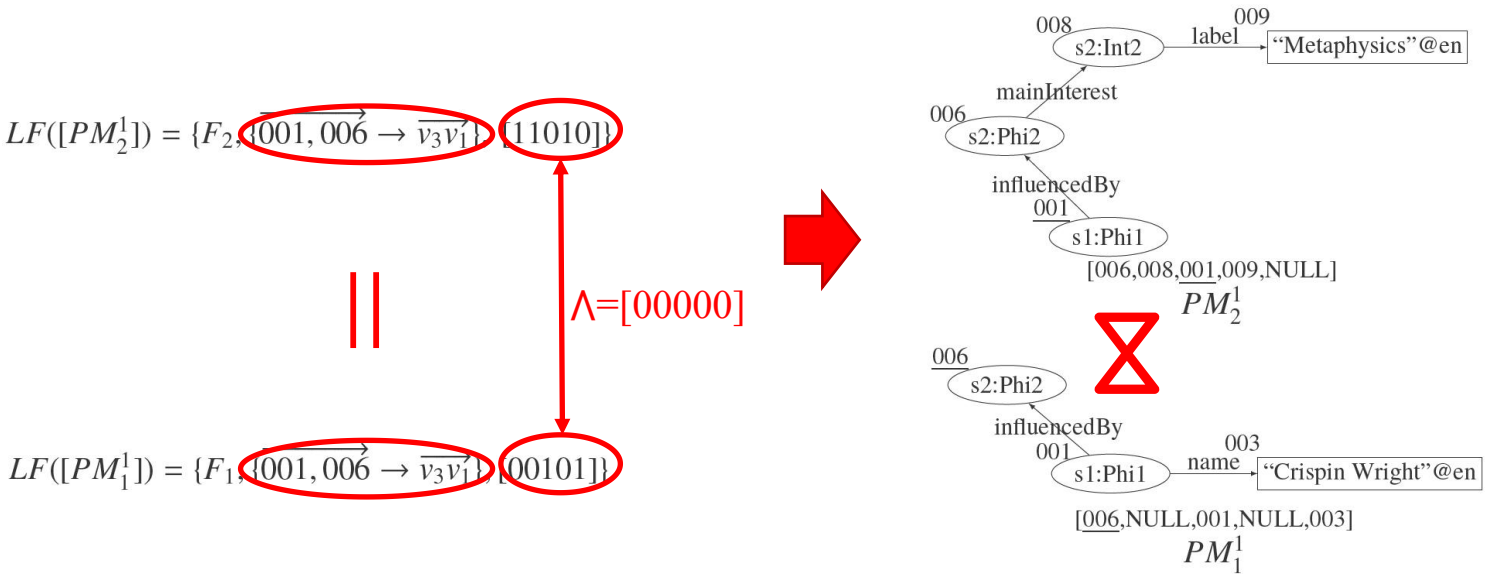
LEC Feature

- The structural information of a LEC is maintained in a compact data structure called *LEC feature*



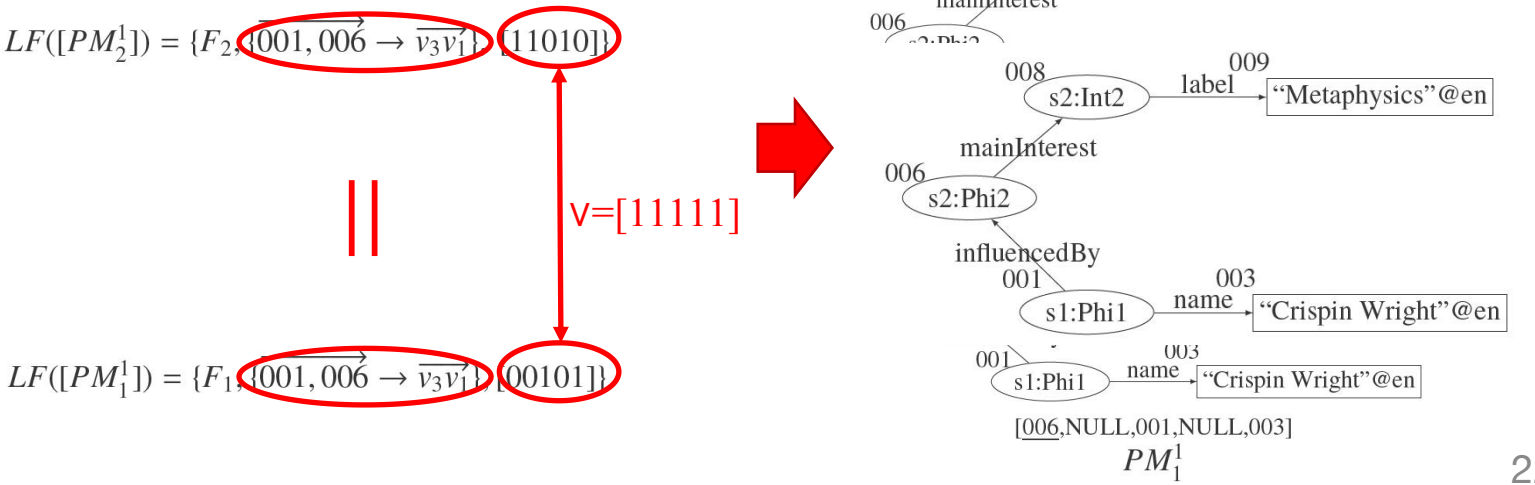
Joinable

- Two LEC features can join if they share the same crossing edges and all bits in the AND result of their LECSigns are 0

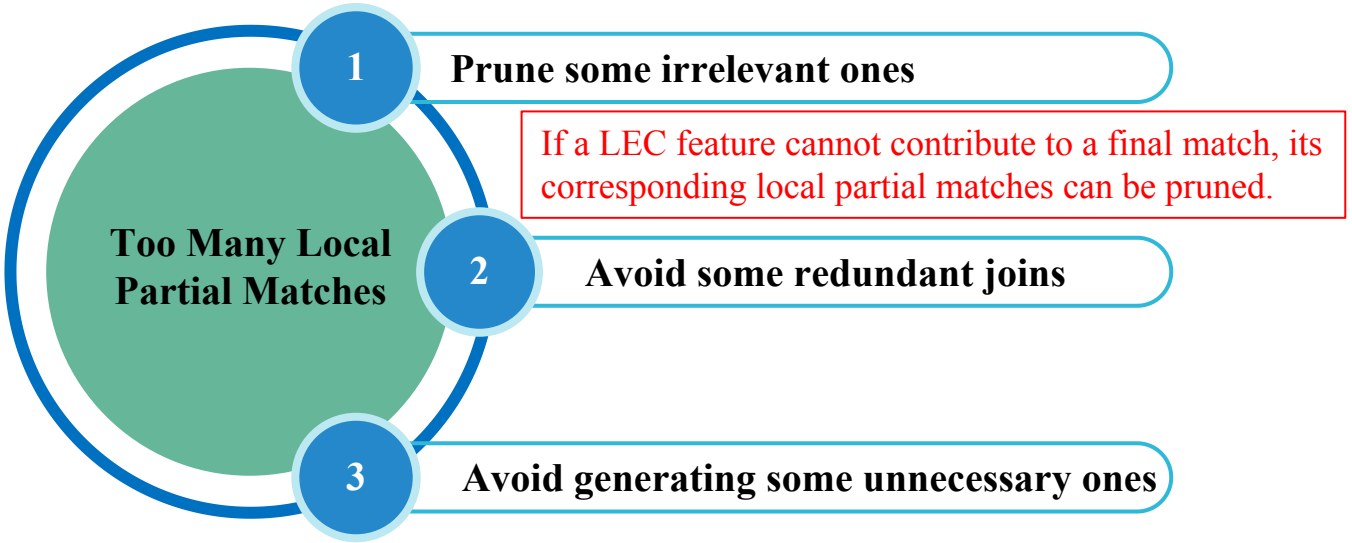


Joinable

- A local partial match can contribute to a final match if its LEC feature can join with some LEC features and all bits in the OR result of their LECSigns are 1



Optimizations based on LEC Features



The optimization is partition bounded in both response time and data shipment

LEC Feature-based Local Partial Match Group

- If the LECSigns of two local partial matches are the same, they cannot join together

Group 2

$$LF([PM_1^1]) = \{F_1, \{\overrightarrow{001, 006} \rightarrow \overrightarrow{v_3v_1}\}, [00101]\}$$
$$LF([PM_1^2]) = \{F_1, \{\overrightarrow{001, 012} \rightarrow \overrightarrow{v_3v_1}\}, [00101]\}$$

$$LF([PM_1^3]) = \{F_1, \{\overrightarrow{006, 005} \rightarrow \overrightarrow{v_1v_2}\}, [01010]\}$$

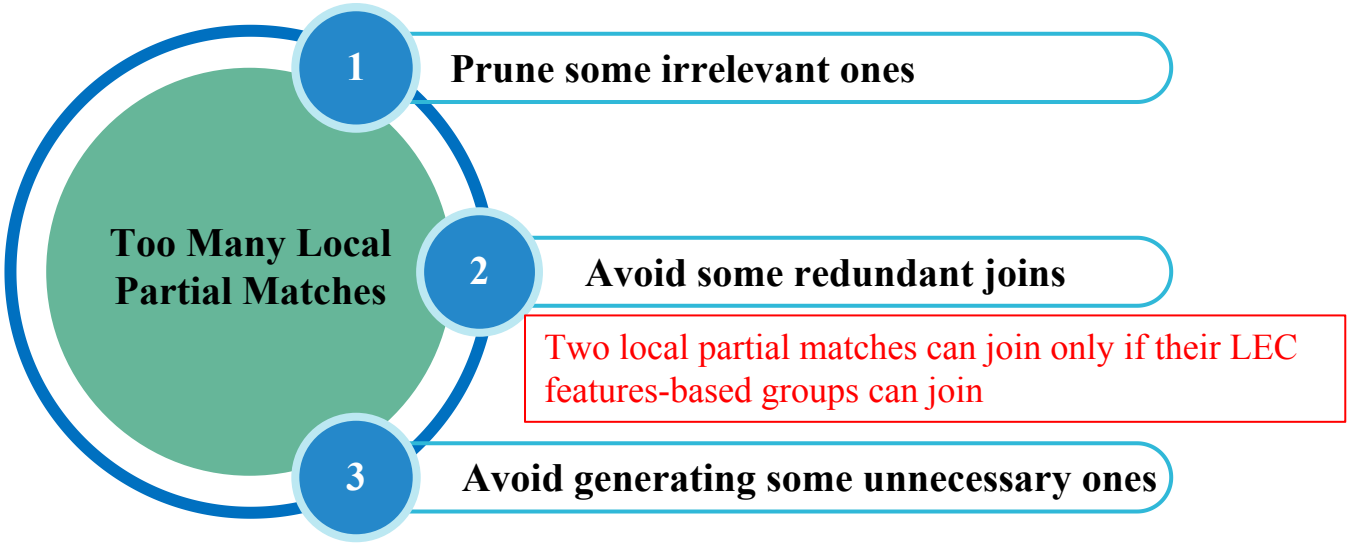
Group 1

Group 3

$$LF([PM_2^1]) = \{F_2, \{\overrightarrow{001, 006} \rightarrow \overrightarrow{v_3v_1}\}, [11010]\}$$
$$LF([PM_3^1]) = \{F_3, \{\overrightarrow{001, 012} \rightarrow \overrightarrow{v_3v_1}\}, [11010]\}$$

$$LF([PM_3^2]) = \{F_3, \{\overrightarrow{014, 013} \rightarrow \overrightarrow{v_1v_2}\}, [01010]\}$$

Optimizations based on LEC Feature-based Group



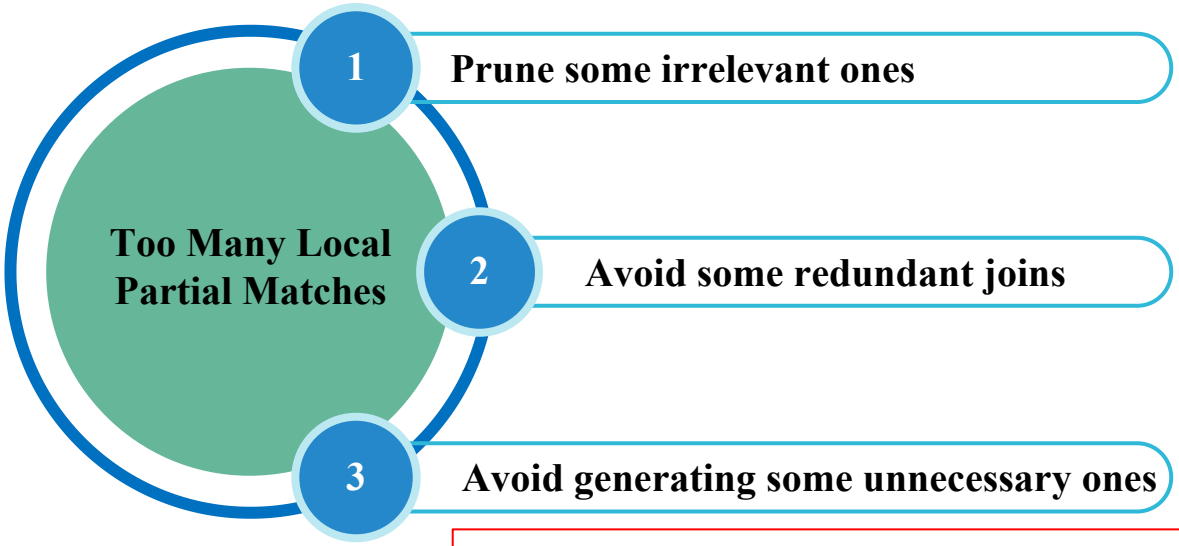
Assembling Internal Candidates

A vertex occurring in a final match must be an internal vertex of a fragment

We can prune a candidate of a variable that is only an extended vertex of fragments

We can avoid generating the local partial matches from the above candidates

Optimizations based on Internal Candidates

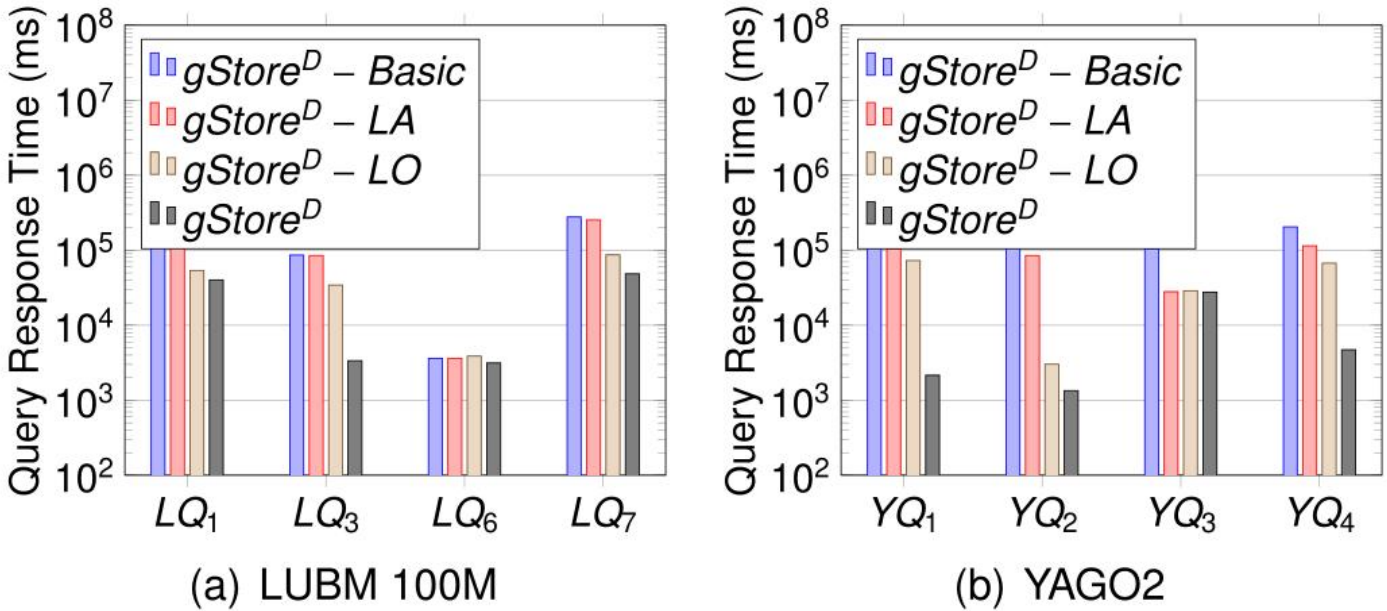


We use the internal candidates of all variables in each site to filter out some false positives.

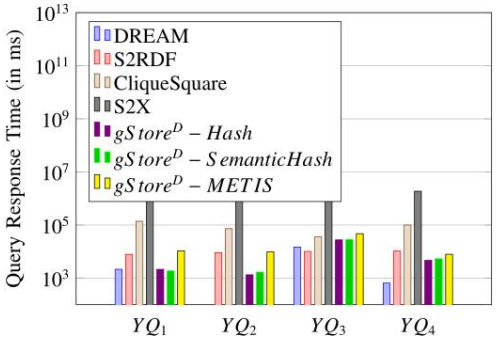
Experiments

- **Datasets:** YAGO2 (about 284 millions triples), BTC (about 1 billion triples) and LUBM (100 millions to 1 billion triples)
- **Competitor:** DREAM, S2X, S2RDF and CliqueSquare
- **Environment:** 12machines running Linux in a LAN

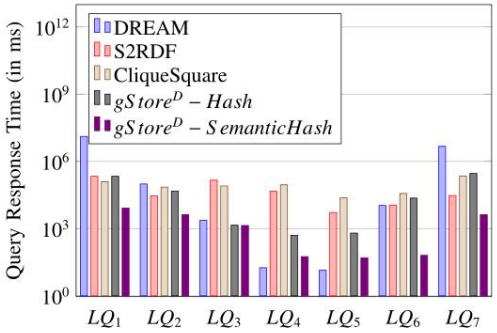
Evaluation of Different Optimizations



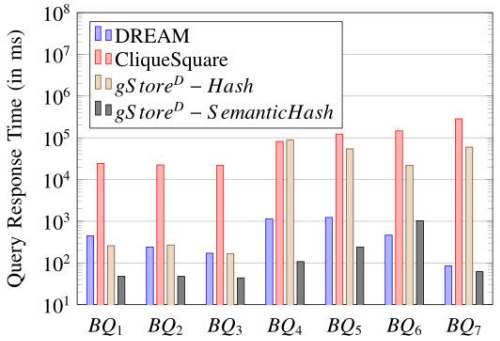
Online Performance Comparison



(a) YAGO2



(b) LUBM 1B



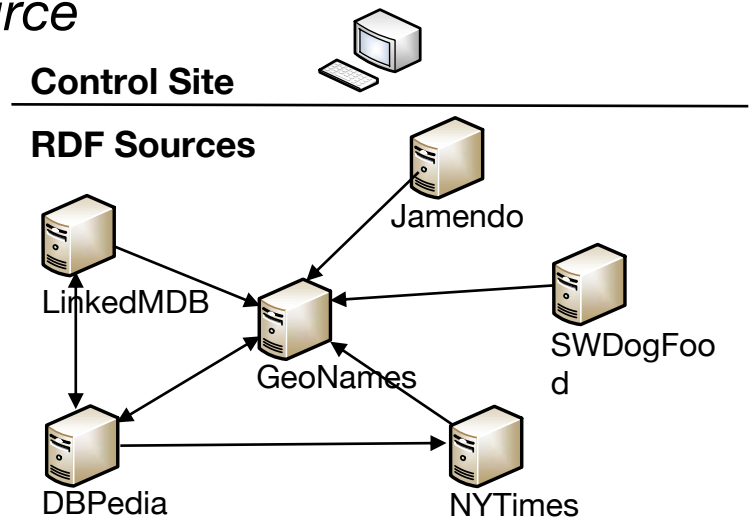
(c) BTC

Part 3

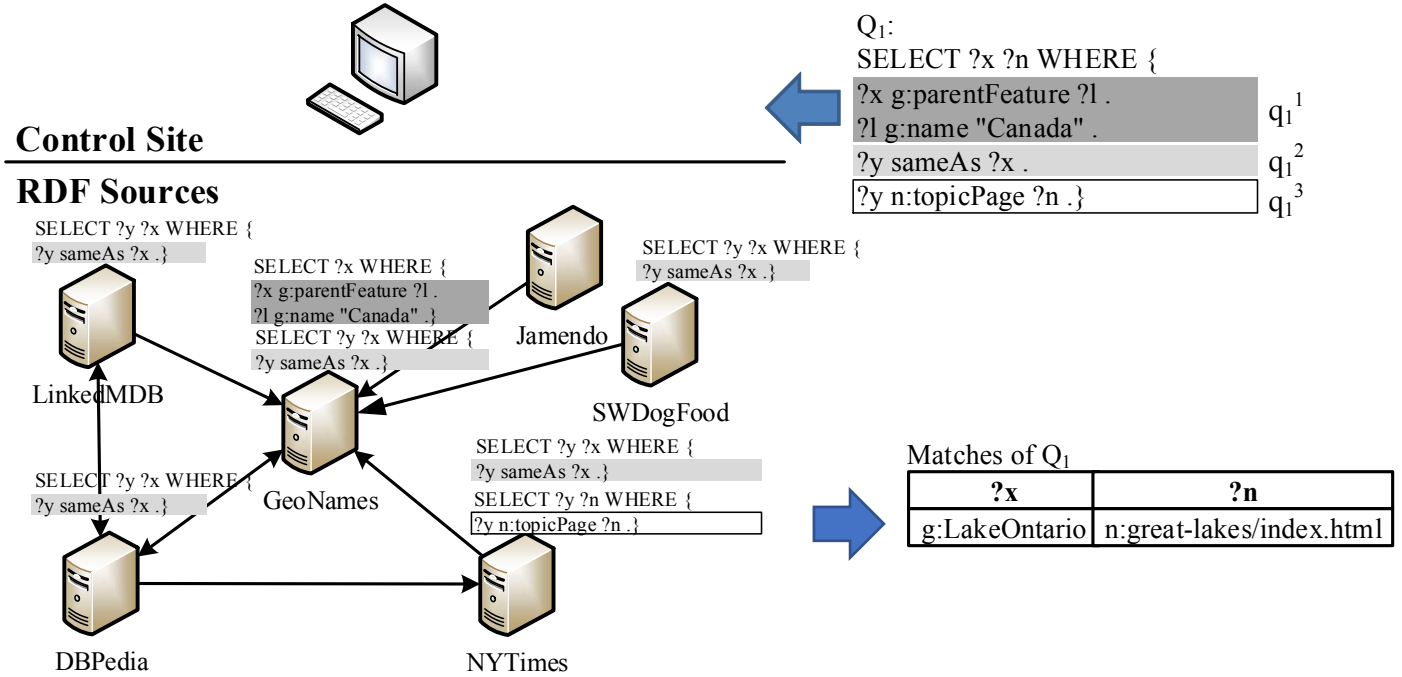
Federated Systems

Federated RDF Systems

- A federated RDF system consists of multiple "autonomous" sites which store their own RDF data and provide SPARQL query interfaces. Here, an autonomous site with a SPARQL interface is called an *RDF source*

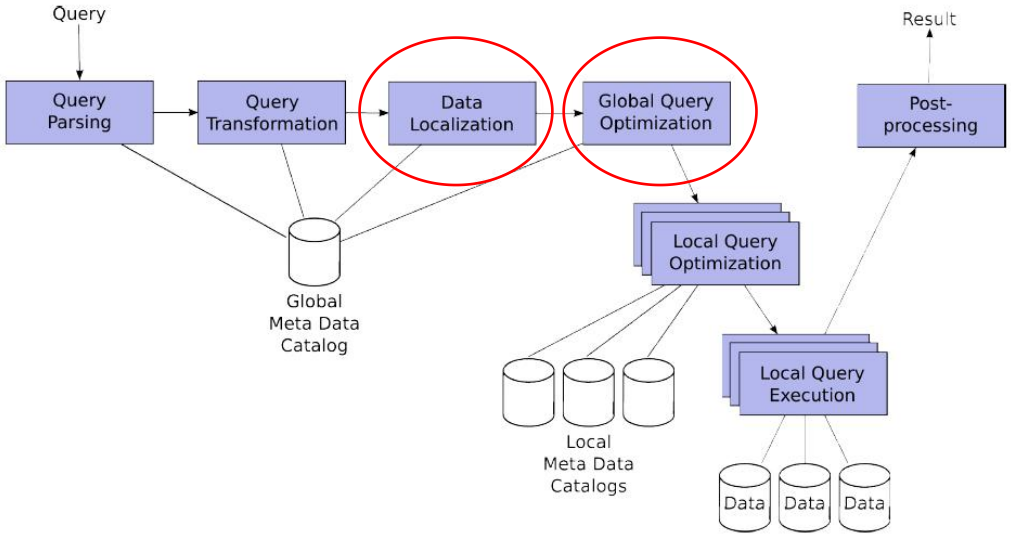


Federated SPARQL Query Processing



Existing Approaches

- Most of them focus on query decomposition and source selection



Federated RDF Systems

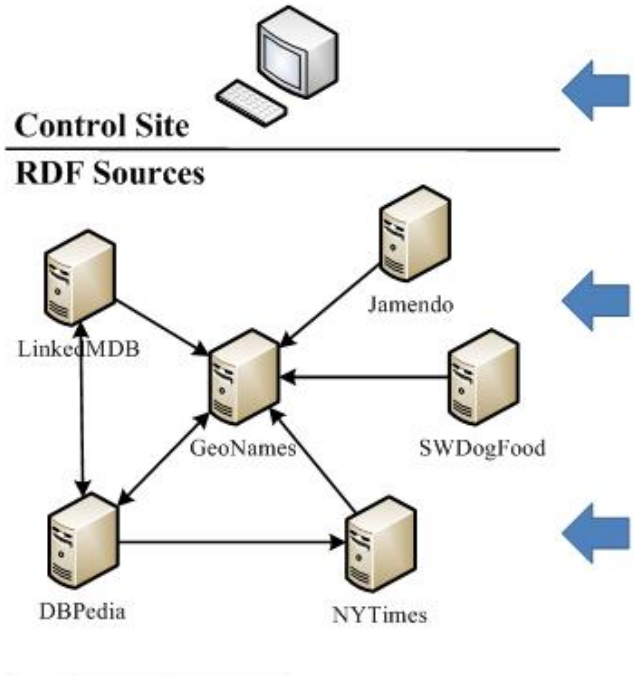
Metadata-based approaches

- Use the metadata to determine which sources are relevant
  - DARQ [Quilitz & Leser, ESWC 2008]
  - QTree [Harth et al., WWW 2010; Prasser et al., EDBT 2012]
  - . . .

ASK query-based approach

- Asking whether or not a triple pattern has an answer at a source
  - FedX [Schwarte et al., ISWC 2011]

Multi-Query Optimization [Peng et al., DASFAA 2018, TKDE 2021]



Q_1:
SELECT ?x ?n WHERE {
?x g:parentFeature ?l .
?l g:name "Canada" .
?y sameAs ?x .
?y n:topicPage ?n .}

Q_2:
SELECT ?c ?p WHERE {
?c g:featureCode g:School.
?c g:parentCountry ?l .
?l g:name "Canada" .
?y sameAs ?c .
?p d:almaMater ?y . }

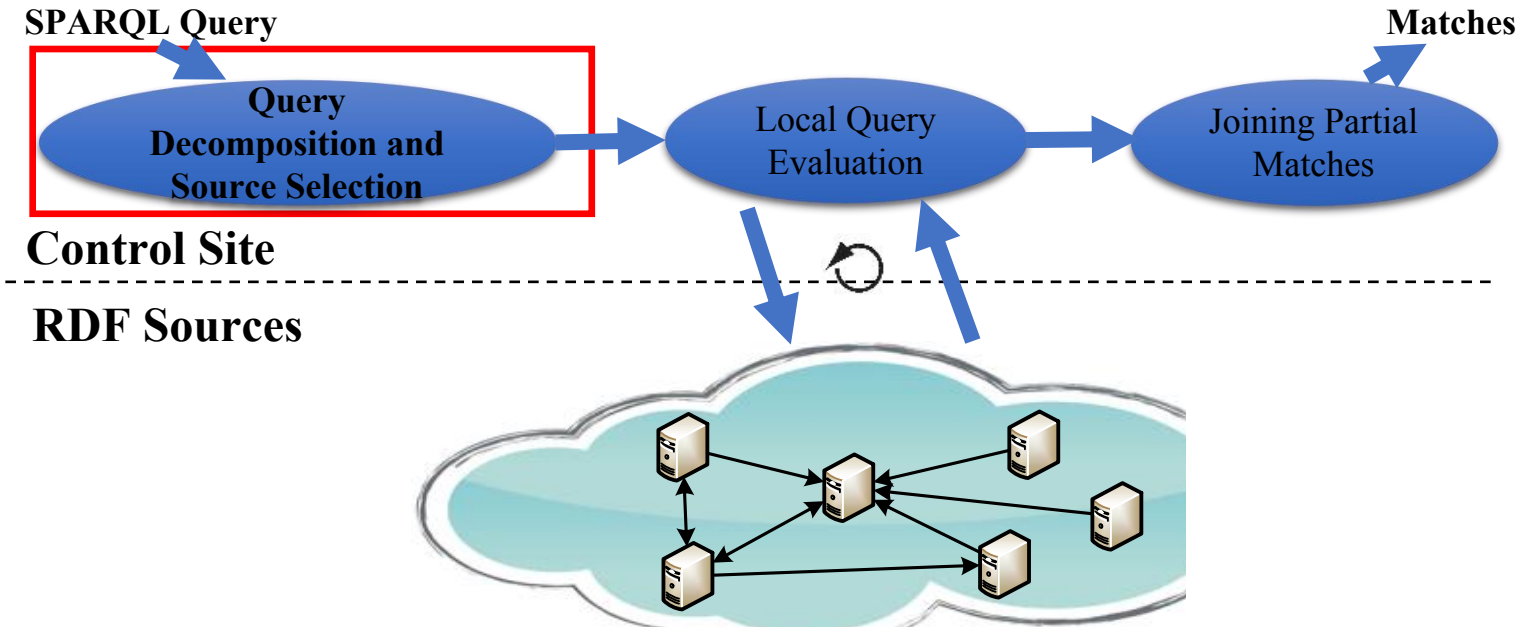
Q_3:
SELECT ?c ?m WHERE {
?c g:featureCode g:PopulatedPlace.
?c g:parentCountry ?l .
?l g:name "Canada" .
?y sameAs ?c .
?m s:hasLocation ?y .
?m type s:Workshop . }

The shaded triple patterns correspond to the common subgraph during each iteration.

Main Ideas

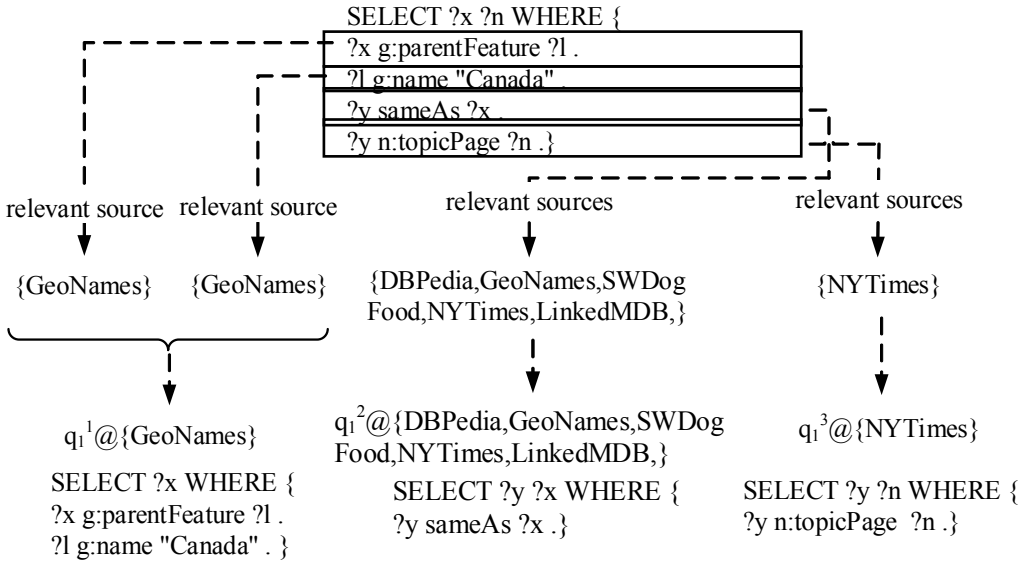
- A query rewriting-based approach for query evaluation on RDF sources while considering the cost of both query evaluation and data shipment
- An effective method to use the interconnection topology between RDF sources to filter out irrelevant sources
- An efficient method to share the common computation of intermediate results joining.

Framework

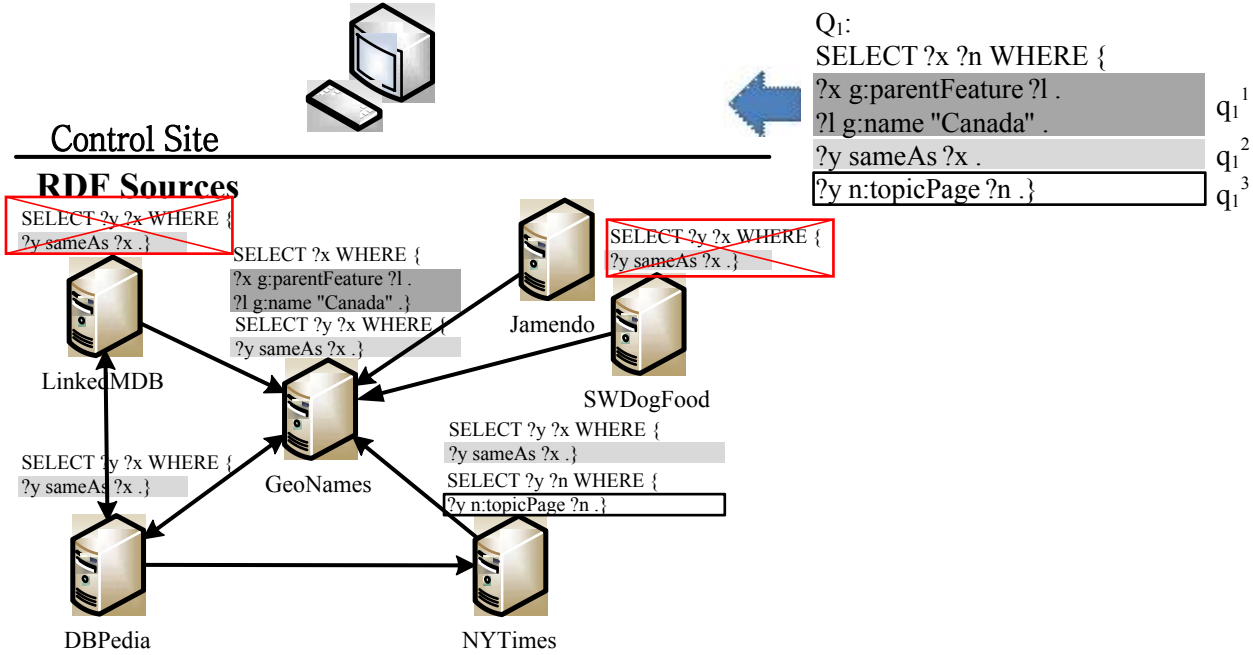

SPARQL Query
Matches
Query Decomposition and Source Selection
Local Query Evaluation
Joining Partial Matches
Control Site
RDF Sources

Basic Query Decomposition and Source Selection

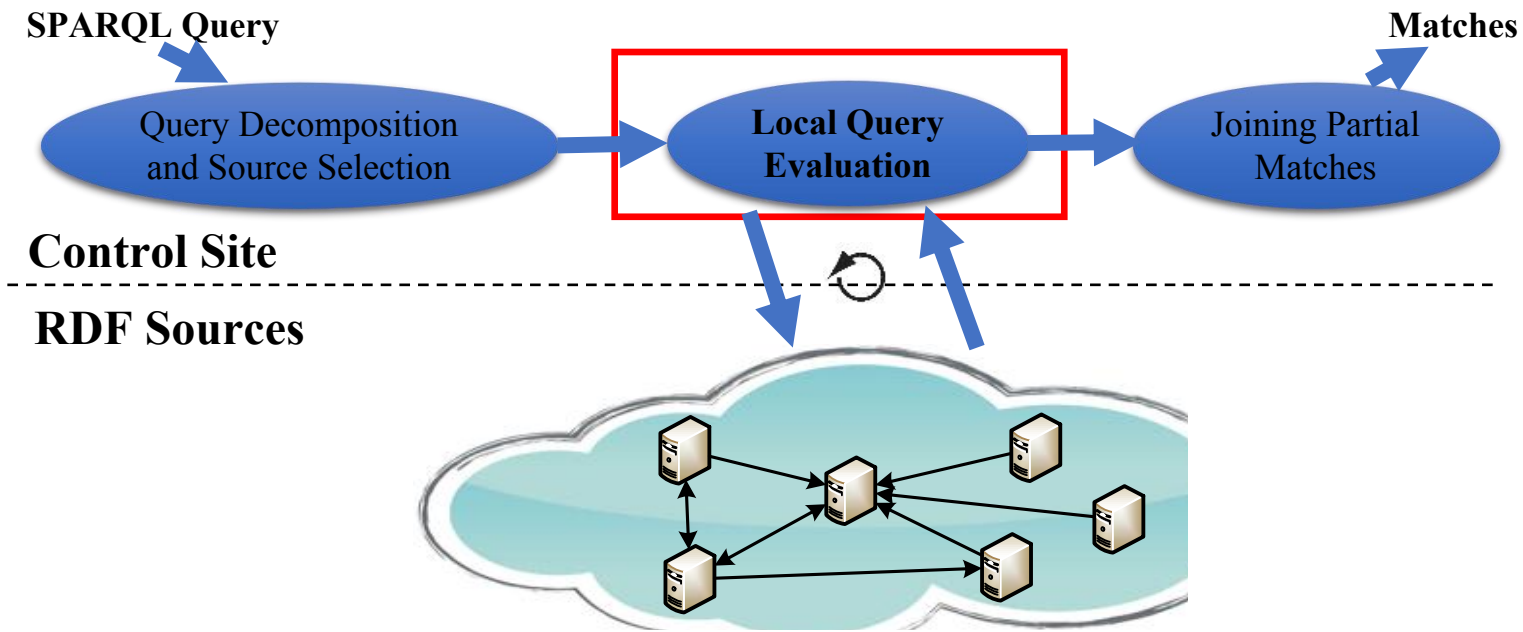
- Most existing solutions decompose the input query based on the triple patterns



Source Topology-based Query Decomposition and Source Selection



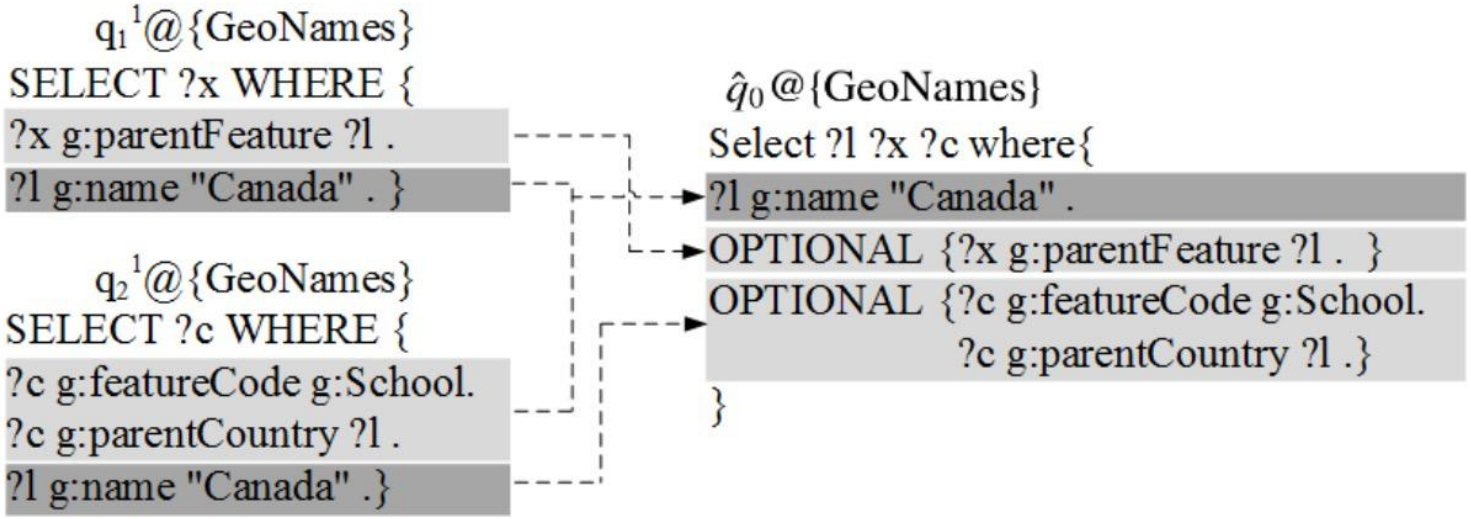
Framework


SPARQL Query
Matches
Query Decomposition and Source Selection
Local Query Evaluation
Joining Partial Matches
Control Site
RDF Sources

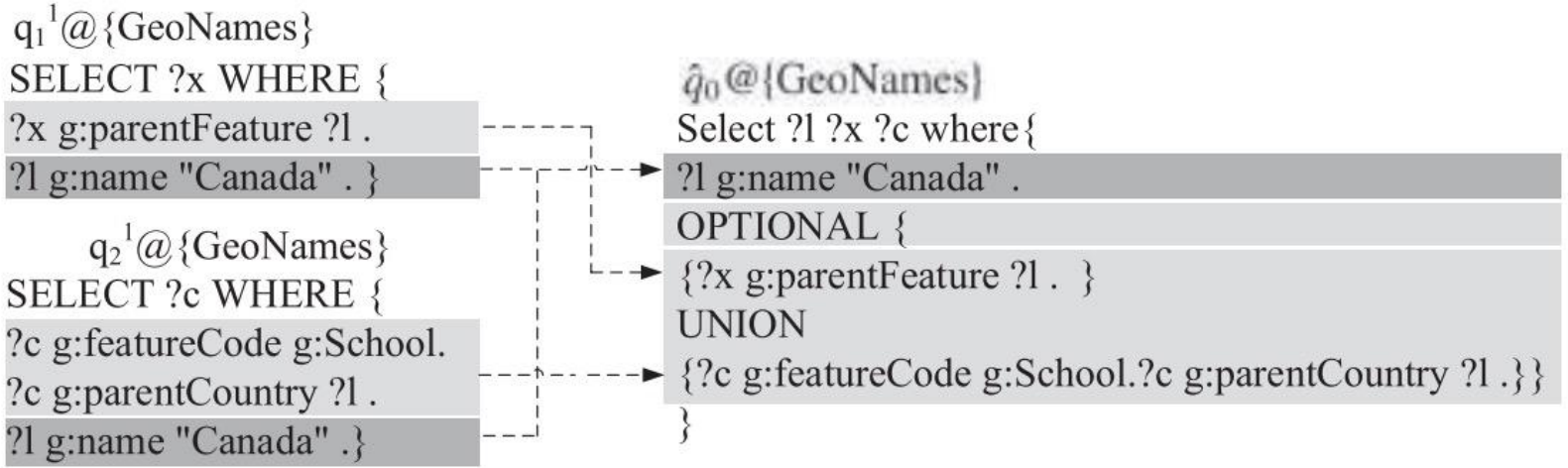
Local Query Evaluation

- When multiple SPARQL queries are posed simultaneously, there is room for sharing computation when executing these queries
- We proposes a **cost-driven query rewriting scheme** to rewrite them (at the control site) into fewer SPARQL queries, which can reduce the number of remote requests and improve the performances

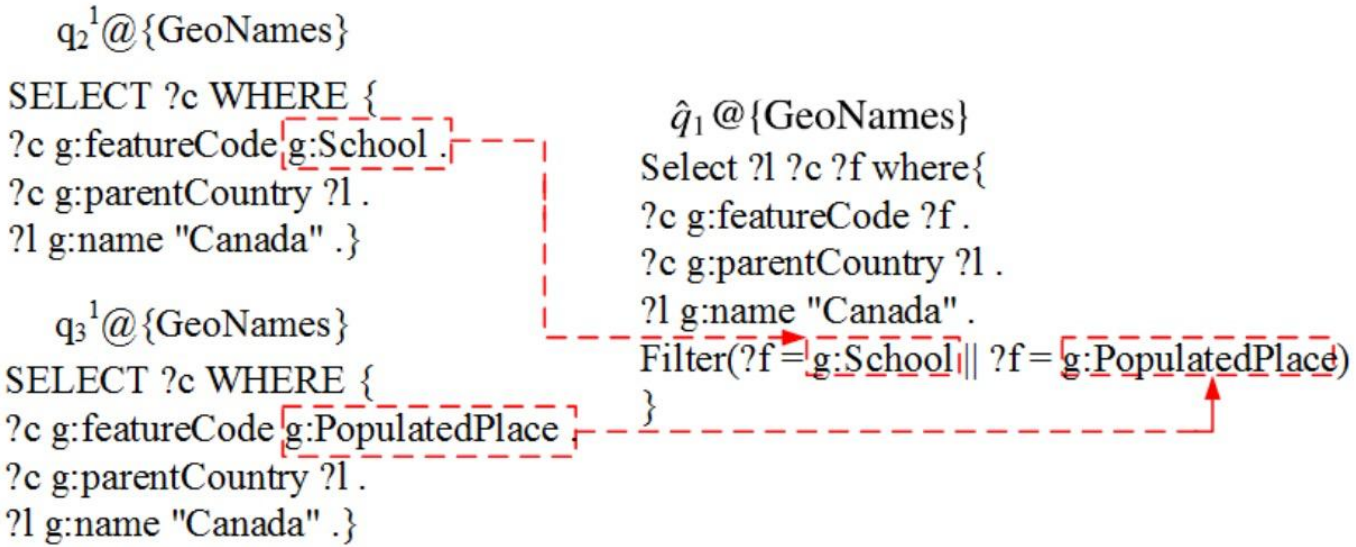
OPTIONAL-Based Rewriting



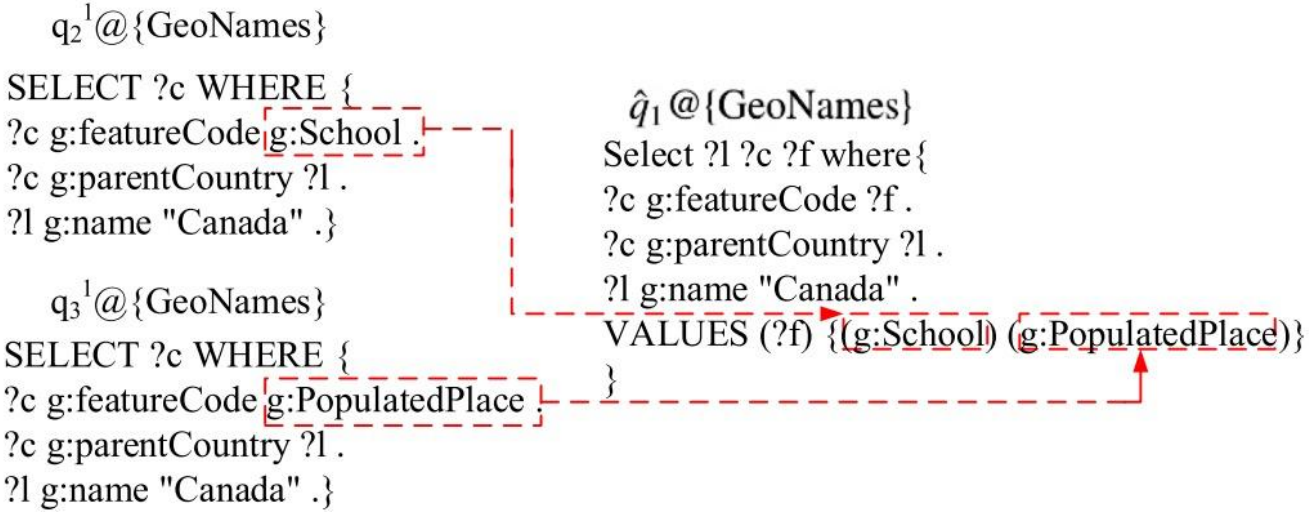
OPTIONAL-UNION-Based Rewriting



FILTER-based Rewriting



VALUES-based Rewriting

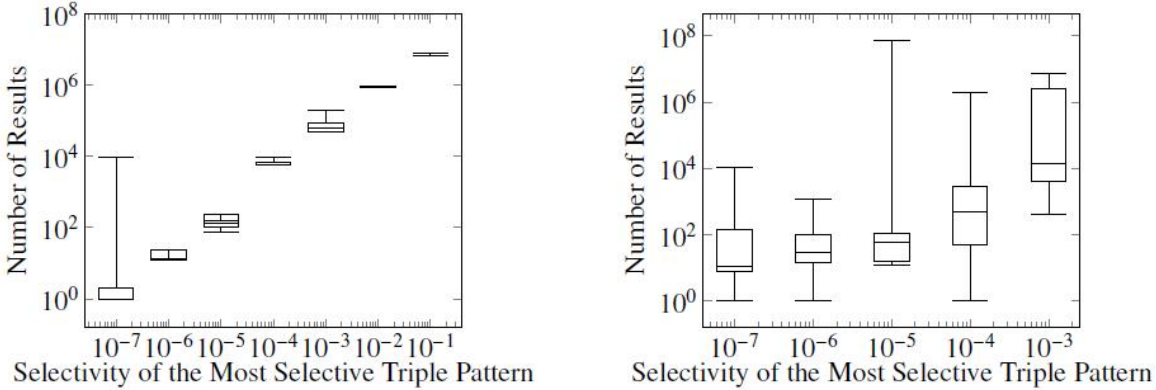


Cost Model for BGPs

- The cardinality of evaluating a basic graph pattern Q is as follows.

$$card(Q) = min_{e \in E(Q)}\{sel(e)\}$$

where sel(e) is the selectivity of triple pattern e in Q.



(a) DBPedia (b) WatDiv 10M

Fig. 6. Relationship Between the Cardinality and the Most Selective Triple Pattern

Cost Model for Query Rewriting

- Hence, given a SPARQL Q, its cost $cost_{LE}(Q)$ for local evaluation is defined as follows

$$card(Q) = \begin{cases} \min_{e \in E(Q)}\{sel(e)\} & if\ Q\ is\ a\ BGP; \\ \min\{card(Q_1), card(Q_2)\} & if\ Q = Q_1\ AND\ Q_2; \\ card(Q_1) + card(Q_2) & if\ Q = Q_1\ UNION\ Q_2; \\ card(Q_1) + \Delta_1 & if\ Q = Q_1\ OPT\ Q_2; \\ card(Q_1) + \Delta_2 & if\ Q = Q_1\ FILTER\ F; \\ \min\{card(Q), |D|\} & if\ Q = VALUES\ \overrightarrow{W}\ D; \end{cases}$$

where T_{CPU} is the CPU unit time to construct a result, and $\triangle_3$ and $\triangle_4$ are empirically trivial values

Cost Model for Data Shipment

- Based on the above, the cost of data shipment for a given query is as follows

$$cost_{DS}(\hat{q}) = card(\hat{q}) \times T_{MSG} = \min_{e \in p}\{sel(e)\} \times T_{MSG}$$

where T_{MSG} is the unit time to transmit a data unit.

Cost Model for Query Rewriting

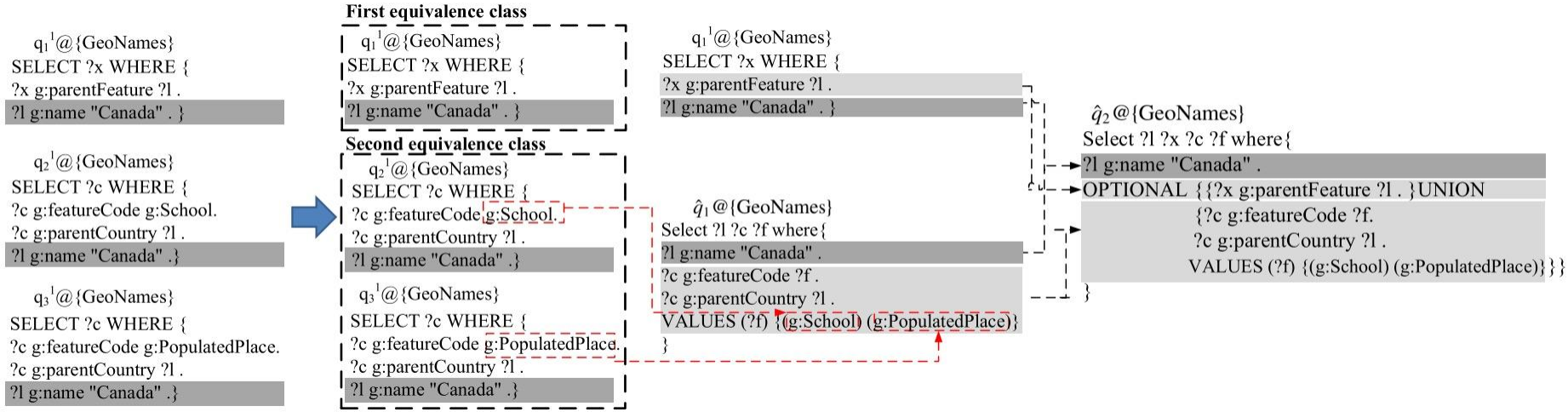
- Given a set of subqueries on a RDF source, if p is their common subgraph among these queries, we rewrite them into a SPARQL query. The cost of the rewriting is the cost of the rewritten query is as follows.

$$
\begin{aligned}
cost(\hat{q}) &= cost_{LE}(\hat{q}) + cost_{DS}(\hat{q}) \\
&= \min_{e \in p}\{sel(e)\} \times (T_{CPU} + T_{MSG}) = cost(p).
\end{aligned}
$$

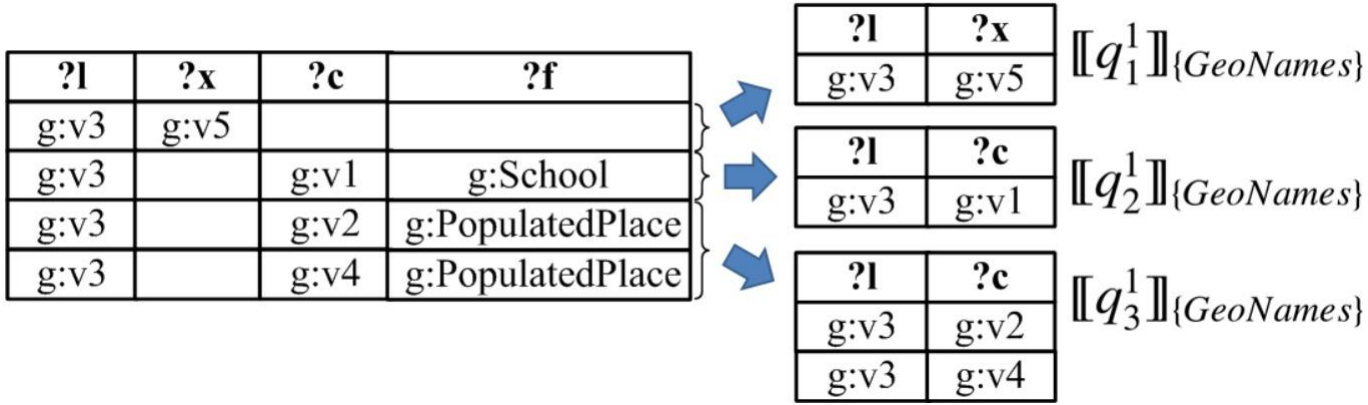
Query Rewriting

- Given a set of *n* subqueries, the objective is to find the set of *m* rewritten queries (m $\leq$ *n*) with the smallest rewriting cost.
- We can prove that finding the optimal set of rewritten queries is NP-complete, and propose a greedy heuristic algorithm to find an approximately optimal solution

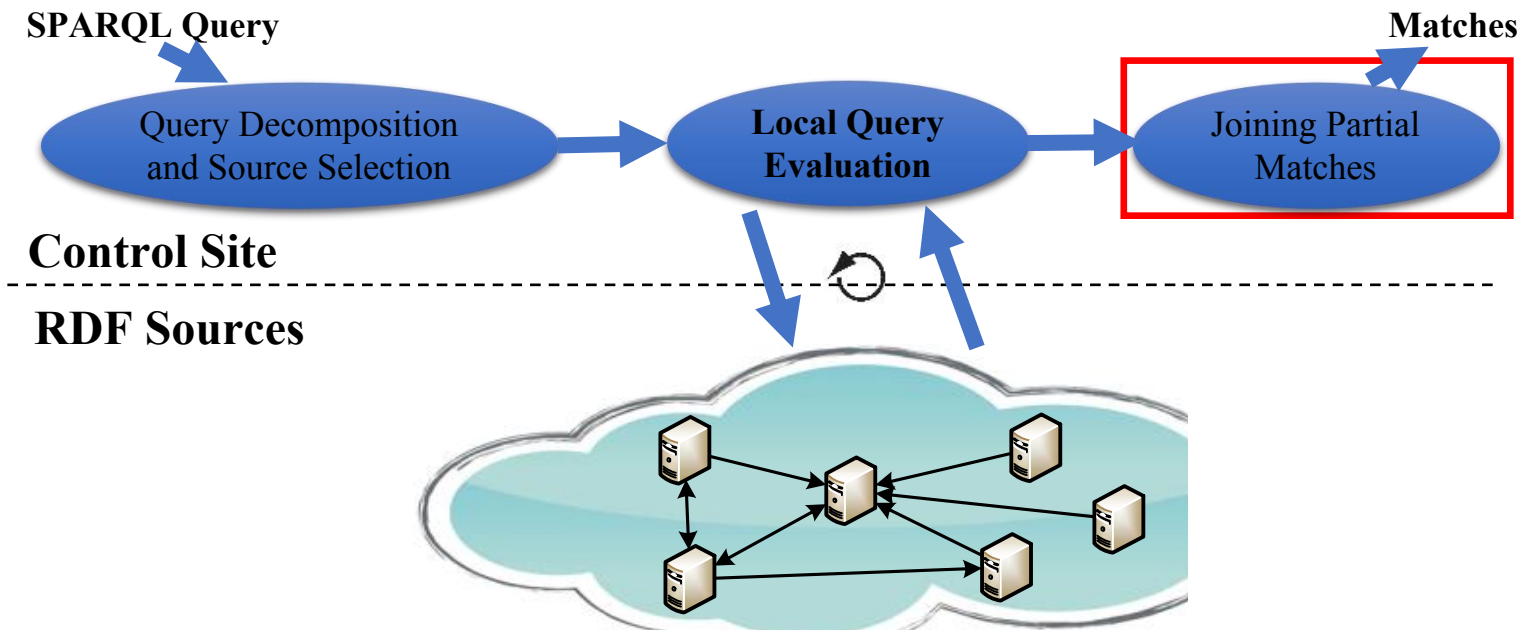
Query Rewriting Example



Postprocessing

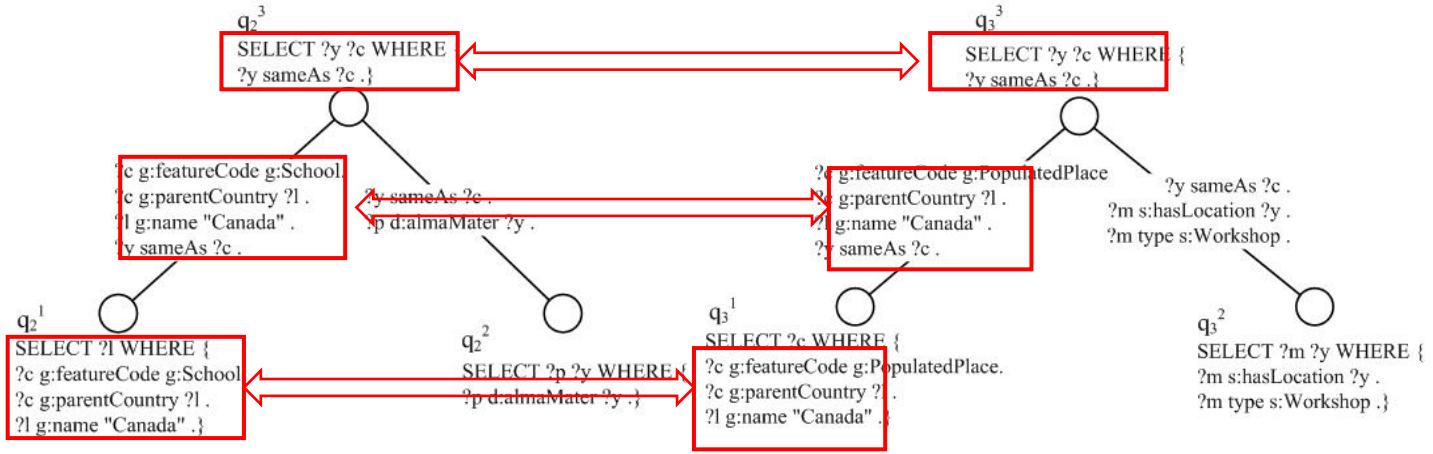


Framework


SPARQL Query
Matches
Query Decomposition and Source Selection
Local Query Evaluation
Joining Partial Matches
Control Site
RDF Sources

Joining Partial Matches

- There may exist some **common computation** in joining partial matches of different queries



Joining Partial Matches

- Formally,

$$[\![q_2^1]\!] \bowtie [\![q_2^3]\!]$$

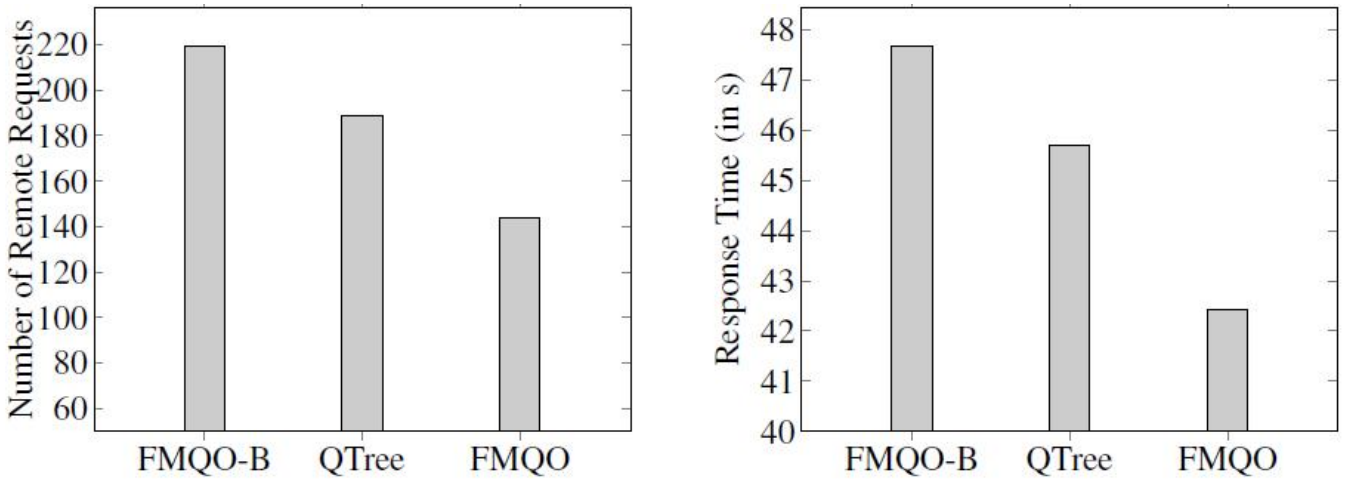
$$[\![q_3^1]\!] \bowtie [\![q_3^3]\!]$$

$$\rightarrow ([\![q_2^1]\!] \cup [\![q_3^1]\!]) \bowtie ([\![q_2^3]\!] \cup [\![q_3^3]\!])$$

Experiments

- **Datasets:** WatDiv, FedBench and LargeRDFBench
- **Competitor:** FedX, SPLENDID and HiBISCuS
- **Environment:** a cluster of machines running Linux, each of which installs Sesame 2.7 to build up an RDF source.

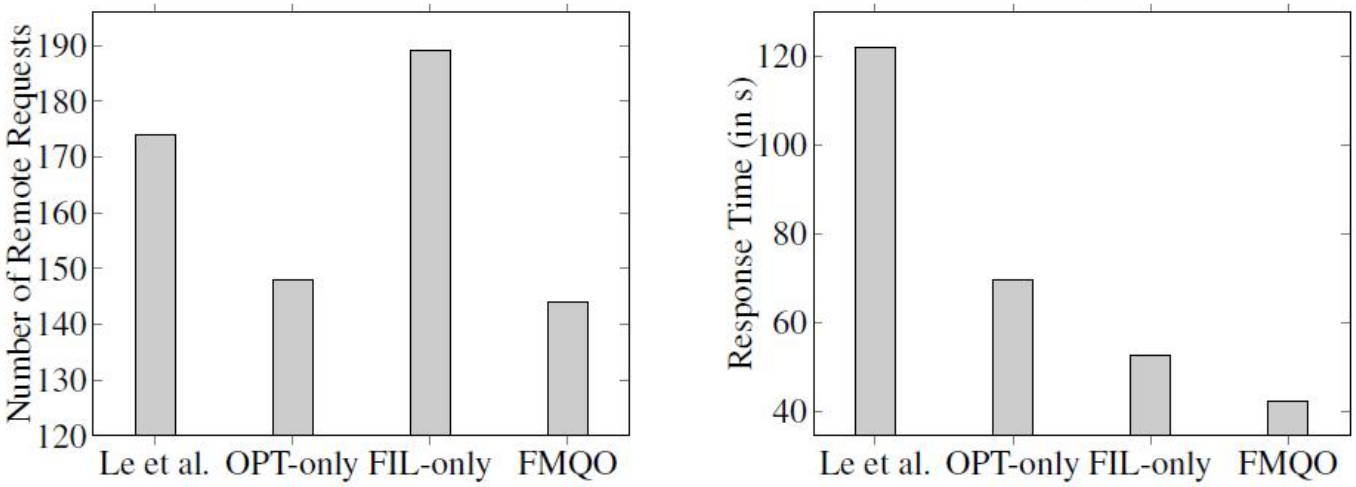
Effect of the Query Decomposition and Source Selection Technique



(a) Number of Remote Requests (b) Response Time

Fig. 8. Evaluating Source Topology-based Source Selection Technique

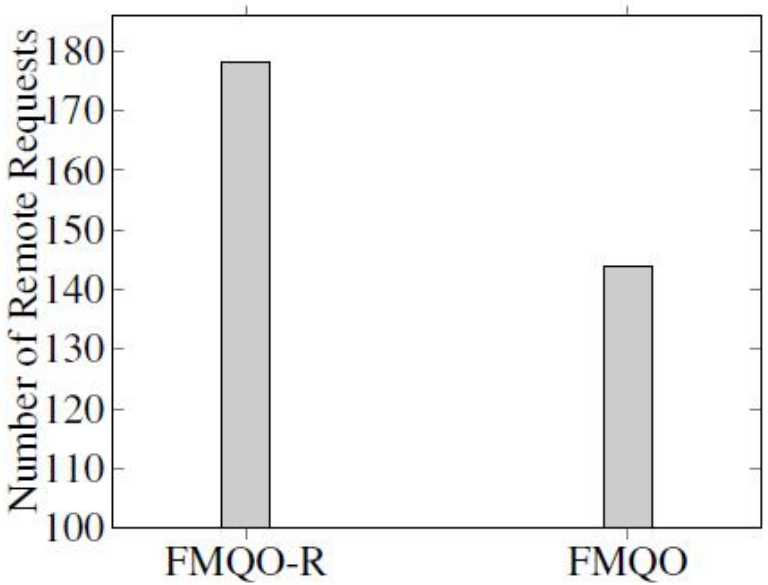
Effect of the Rewriting Strategies



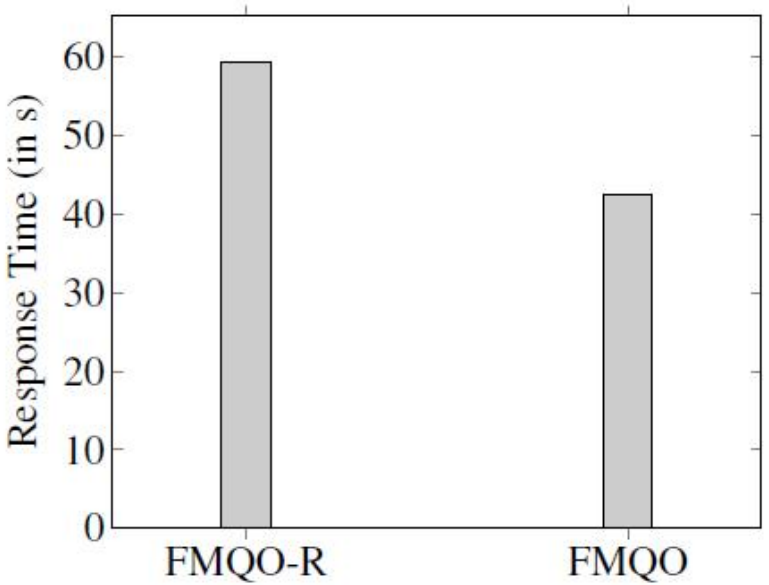
(a) Number of Remote Requests (b) Response Time

Fig. 9. Evaluating Different Rewriting Strategies

Evaluation of the Cost Model



(a) Number of Remote Requests



(b) Response Time

Effect of Optimization Techniques for Joins

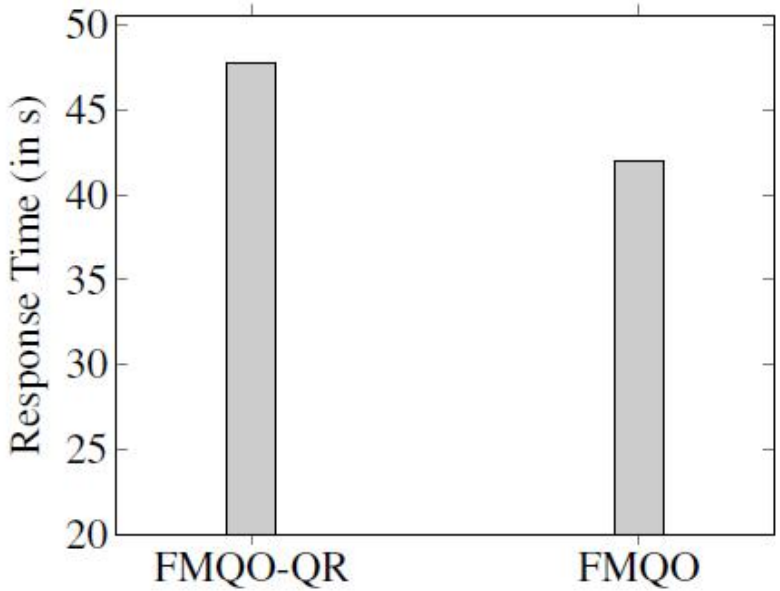
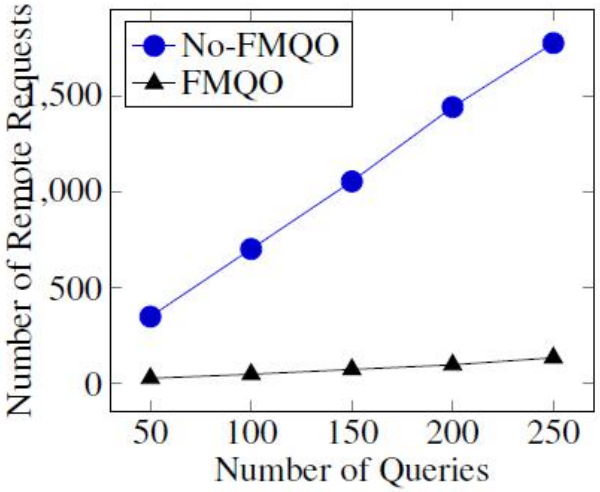
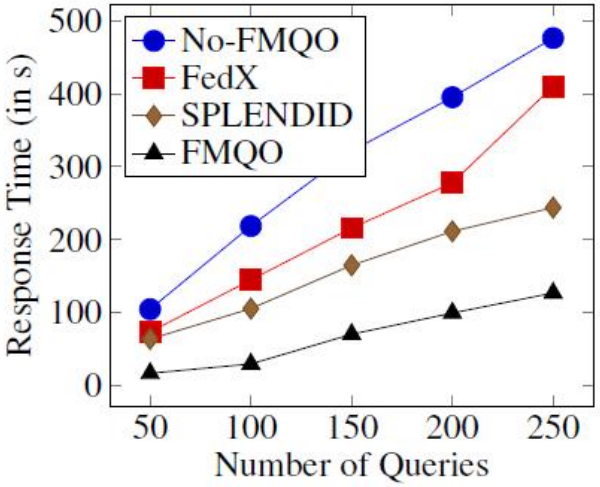


Fig. 11. Effect of Optimization Techniques for Joins

FedBench (Cross Domain)

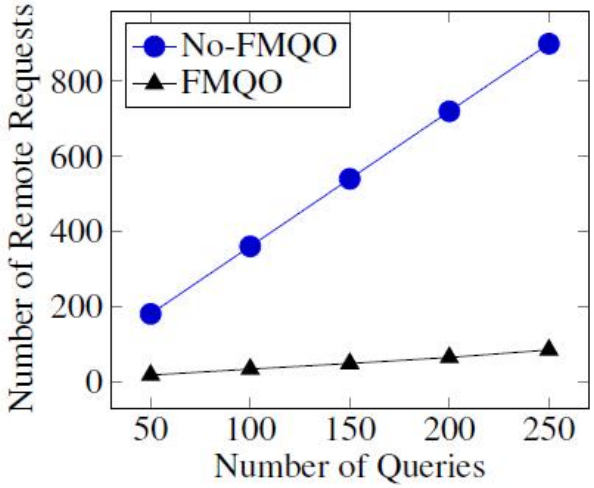


(a) Number of Remote Requests

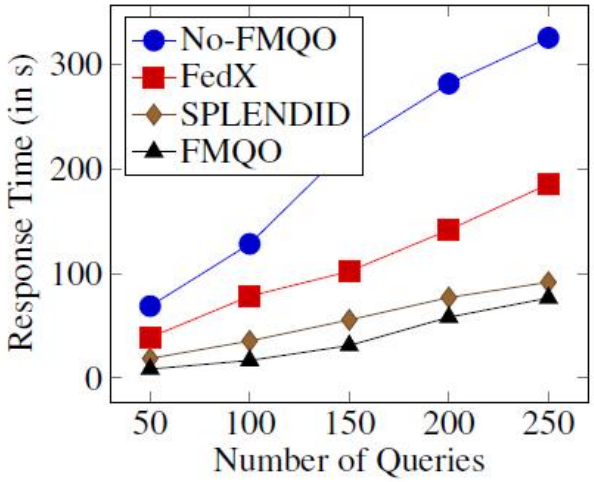


(b) Response Time

FedBench (Life Science)



(a) Number of Remote Requests



(b) Response Time

Part 4

Conclusions

Conclusions

- We introduce **partitioning-agnostic** and **federated** RDF systems and discuss how to optimize query processing

Future Work

- There are many distributed database managment system architectures which are further used for RDF graphs
  1. Parallel Computing
  2. Edge Computing

  .....

THANK YOU

References

- Peng Peng, Lei Zou, Runyu Guan. Accelerating Partial Evaluation in Distributed SPARQL Query Evaluation. ICDE 2019: 112-123
- Peng Peng, Lei Zou, M. Tamer Özsu, Lei Chen, Dongyan Zhao. Processing SPARQL queries over distributed RDF graphs. VLDB J. 25(2): 243-268 (2016)
- Bastian Quilitz, Ulf Leser. Querying Distributed RDF Data Sources with SPARQL. ESWC 2008: 524-538
- Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, Jürgen Umbrich. Data summaries for on-demand queries over linked data. WWW 2010: 411-420
- Fabian Prasser, Alfons Kemper, Klaus A. Kuhn. Efficient distributed query processing for autonomous RDF databases. EDBT 2012: 372-383
- Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, Michael Schmidt. FedX: Optimization Techniques for Federated Query Processing on Linked Data. ISWC (1) 2011: 601-616
- Peng Peng, Qi Ge, Lei Zou, M. Tamer Özsu, Zhiwei Xu, Dongyan Zhao. Optimizing Multi-Query Evaluation in Federated RDF Systems. IEEE Trans. Knowl. Data Eng. 33(4): 1692-1707 (2021)
- Peng Peng, Lei Zou, M. Tamer Özsu, Dongyan Zhao. Multi-query Optimization in Federated RDF Systems. DASFAA (1) 2018: 745-765