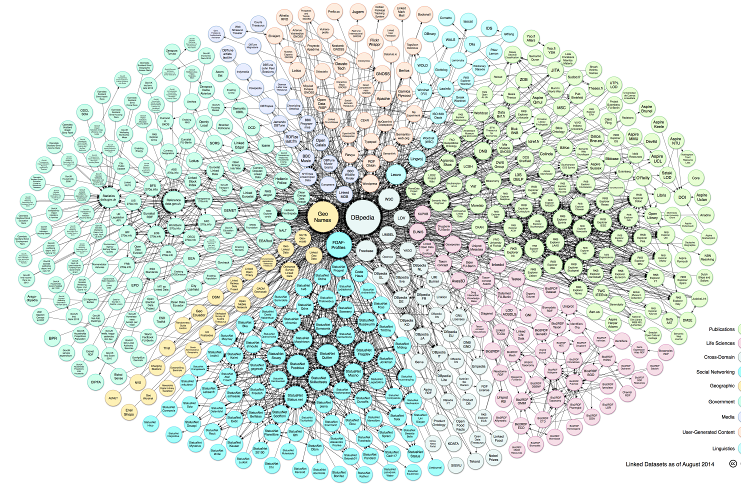
CROISSANT: Centralized Relational Interface for Web-scale SPARQL Endpoints

Takahiro Komamizu, Toshiyuki Amagasa,
Hiroyuki Kitagawa

University of Tsukuba
Japan

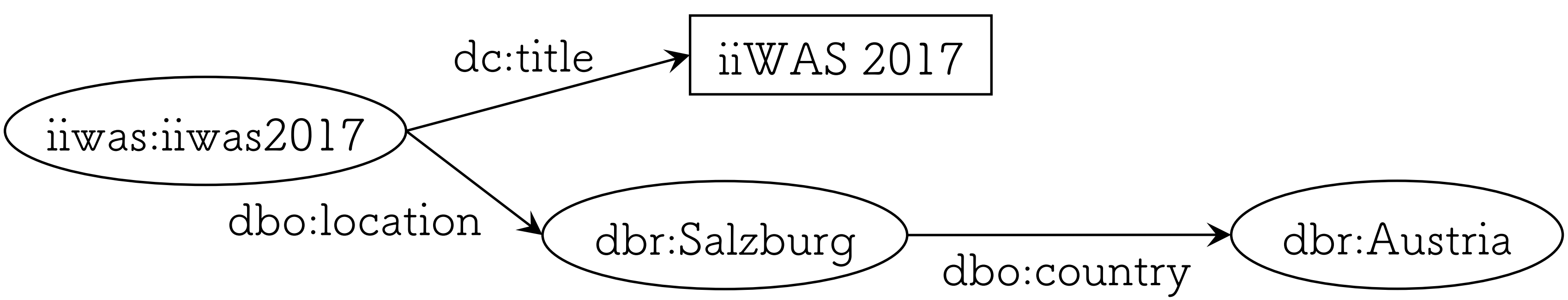
Linked Data (LD)



Linked Open Data cloud diagram (2014-08)

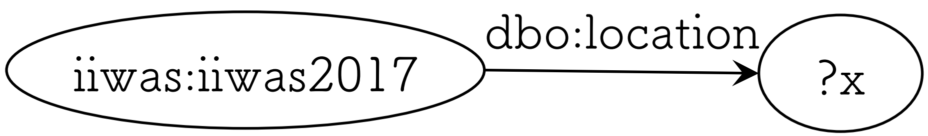
- Open data paradigm
- Linking facts in open data
  - RDF (Resource Description Framework)
  - e.g.,

Subject	Predicate	Object
iiwas:iiwas2017	dc:title	"iiWAS2017"
iiwas:iiwas2017	dbo:location	dbr:Salzburg
dbr:Salzburg	dbo:country	dbr:Austria



Search over LD

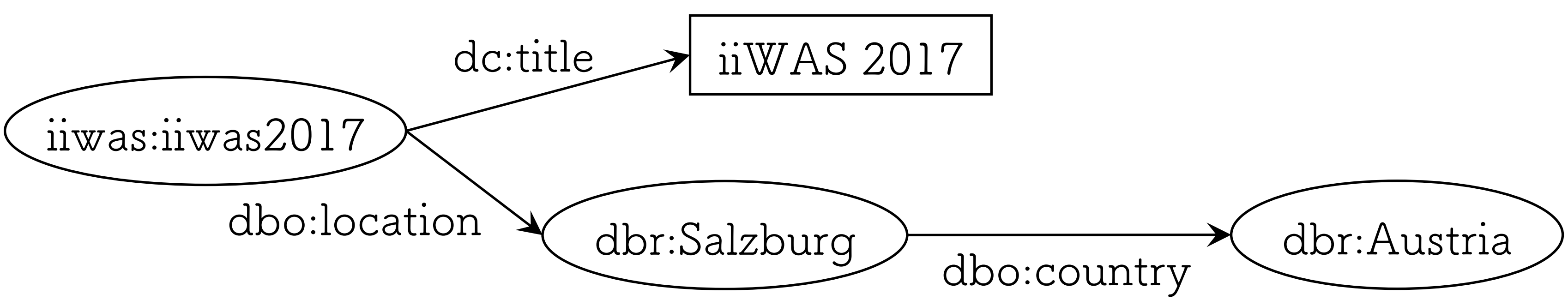
- Finding facts in LD data
- Standardized method: SPARQL query
  - Graph pattern-based requirement representation
  - Bindings to variables in patterns are results.



Graph pattern query

x
dbr:Salzburg

Results



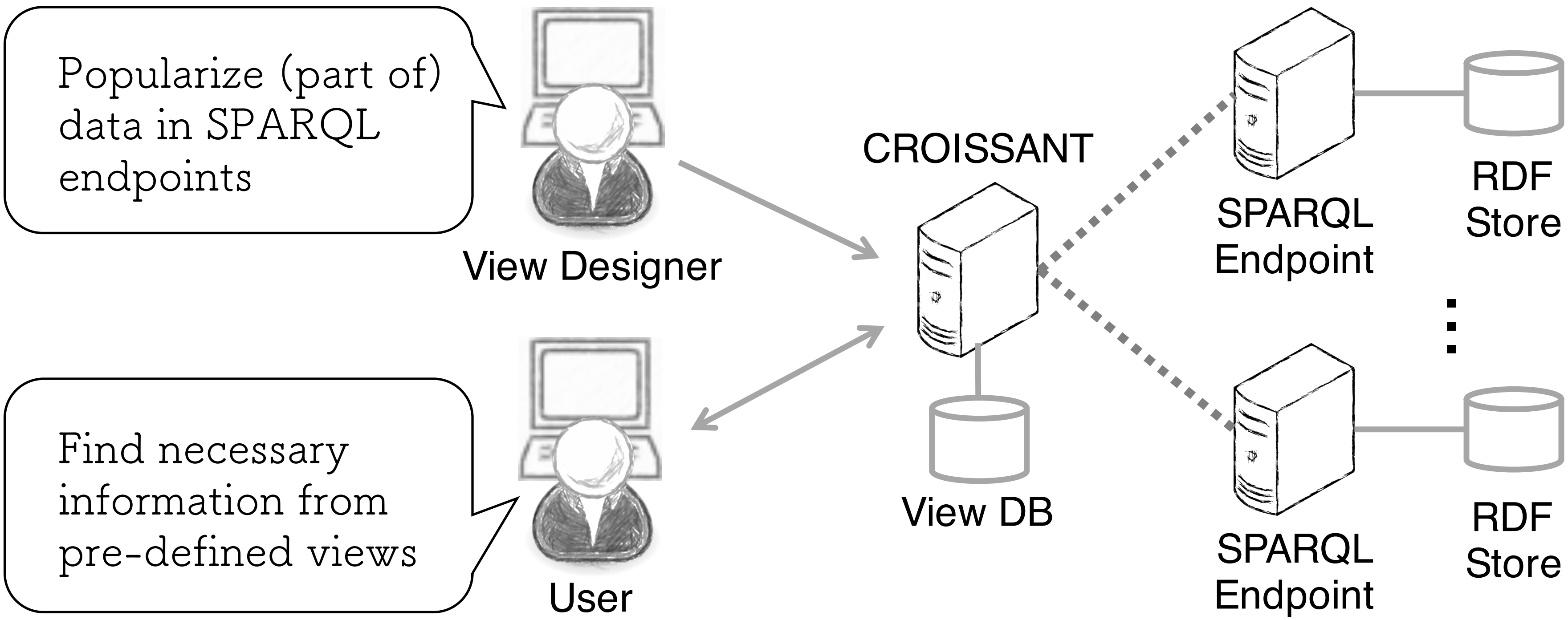
Motivation

- LD is **heterogeneous**, **complicated**, and **distributed**.
  - Large number of facts
  - Large number of classes (types of entities)
  - Complicated multi-graph structure
  - Distributed and self-managed data servers
- Users are demanded to describe search intents by SPARQL.
  - Graph patterns
    ➔ require understanding of the structure of data
  - Multiple queries for distributed data
    ➔ require integration of multiple results

Our Objective and Approach

- Objective
  - Queriable interface for popular query languages for distributed RDF data
- Approach: CROISSANT
  - Centralized interface
    ➔ unified access interface for distributed data
  - Relational view-based interface
    ➔ reducing effort for understanding data
  - Query rewriting from SQL to SPARQL
  - Query optimization

CROISSANT: an overview



<u>Users' point-of-view,</u>
<u>SPARQL endpoints are invisible.</u>

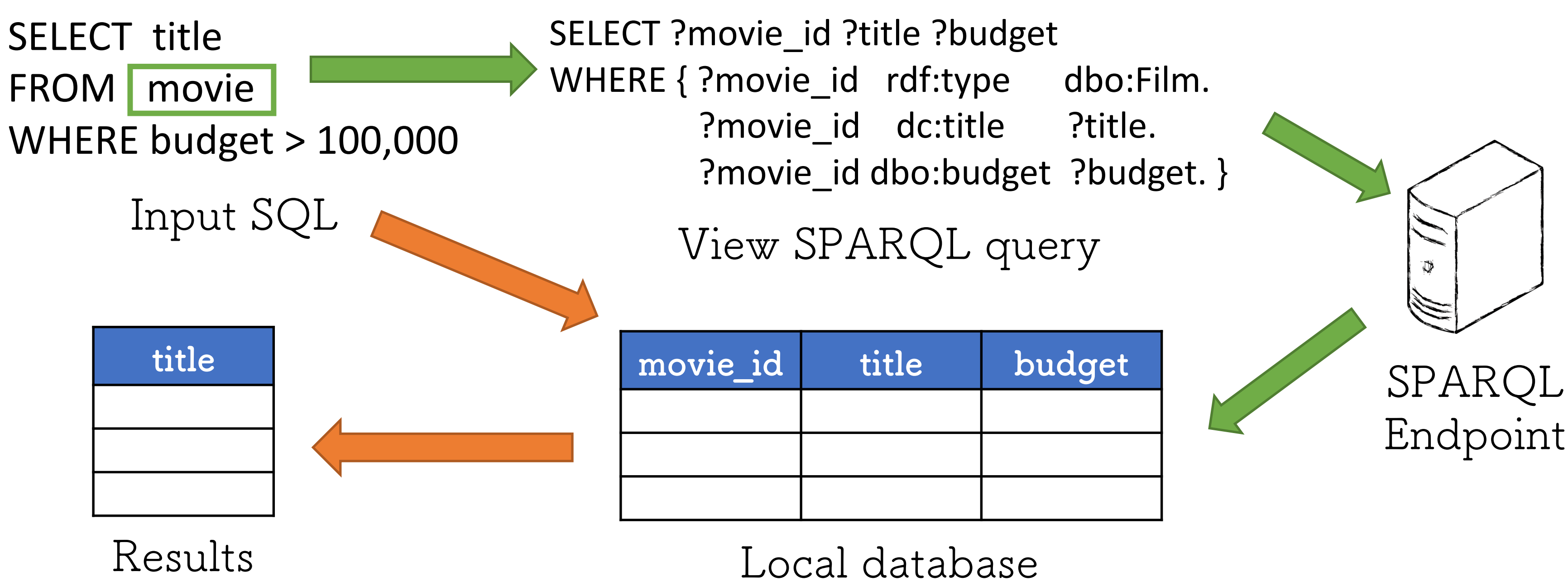
CROISSANT: view definition

$$\langle name, schema, endpoint\_url, SPARQL\_query \rangle$$

- *name*: view name
  - e.g., movie
- *schema*: relational schema of the view
  - e.g., (movie_id, title, budget)
- *endpoint_url*: location of SPARQL endpoint
  - e.g., http://dbpedia.org/sparql
- *SPARQL_query*: SELECT query of data in the view
  - e.g.,
    ```
    SELECT  ?movie_id ?title ?budget
    WHERE { ?movie_id   rdf:type     dbo:Film.
            ?movie_id   dc:title     ?title.
            ?movie_id   dbo:budget ?budget. }
    ```

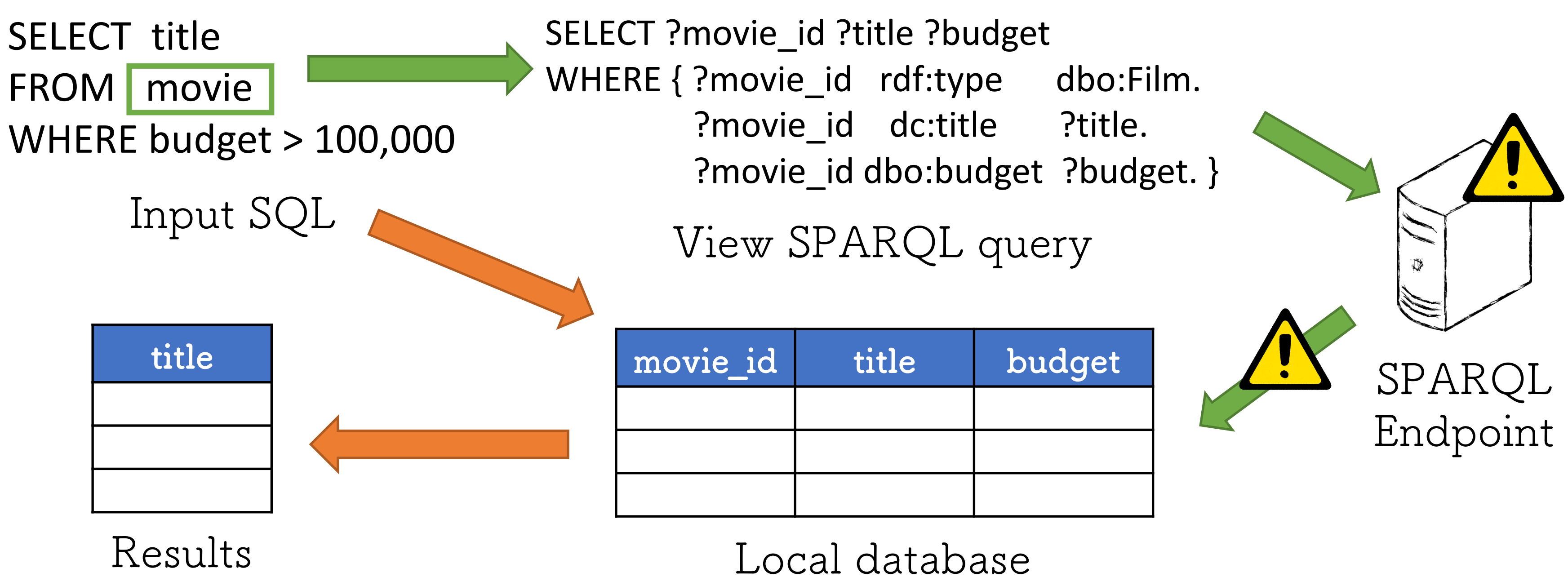
CROISSANT: query execution

- Naive execution
  1. Execute SPARQL query of corresponding views and store the results into local database.
  2. Perform SQL query over the local database.



Performance issues

- Execution cost of SPARQL queries
  - Immature performance of SPARQL endpoints
- Transportation cost from SPARQL endpoints



Query Optimization

- Basic idea:
  reduce #results from SPARQL endpoints
- Strategies
  - View materialization
    - Everything is transferred to local database in advance.
    - Suffer from update issues.
  - Projection push-down
  - Selection push-down
  - View query merge

Projection Push-down

- Push projection conditions into view SPARQL queries.

```
SELECT  title
FROM    movie
WHERE budget > 100,000
```

Input SQL

```
SELECT ?movie_id ?title ?budget
WHERE { ?movie_id   rdf:type      dbo:Film.
        ?movie_id   dc:title      ?title.
        ?movie_id dbo:budget  ?budget. }
```

Pushed-down View SPARQL query

Selection Push-down

- Push selection conditions into view SPARQL queries.
- Pushing rules

Comparator	Variable type	FILTER expression
θ	Numeric	`FILTER (`*variable θ value*`)`
$=$	Textual	`FILTER (str(`*variable*`) = `*value*`)`
like	Textual	`FILTER regex(`*variable*`, `*value*`)`

```
SELECT  title
FROM    movie
WHERE budget > 100,000
```

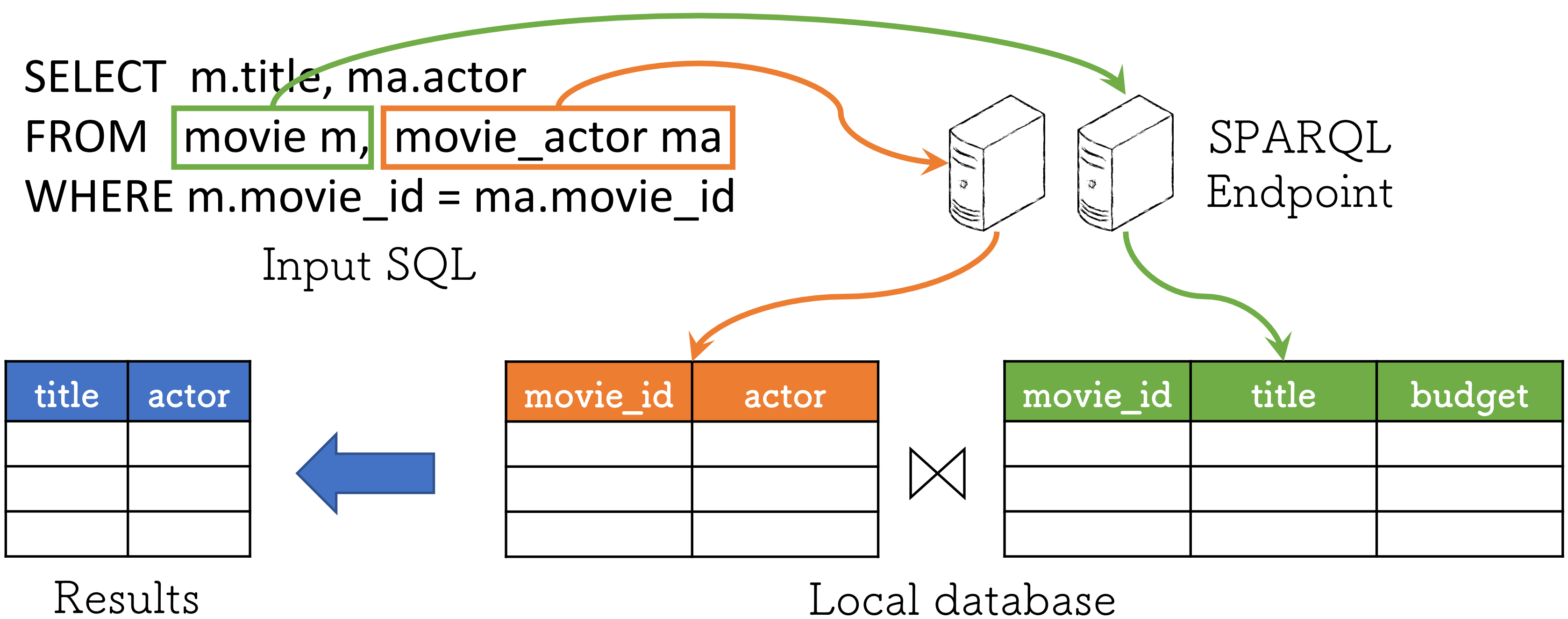
Input SQL

```
SELECT ?movie_id ?title ?budget
WHERE { ?movie_id   rdf:type      dbo:Film.
        ?movie_id   dc:title      ?title.
        ?movie_id dbo:budget  ?budget.
        FILTER (?budget > 100,000) }
```

Pushed-down View SPARQL query

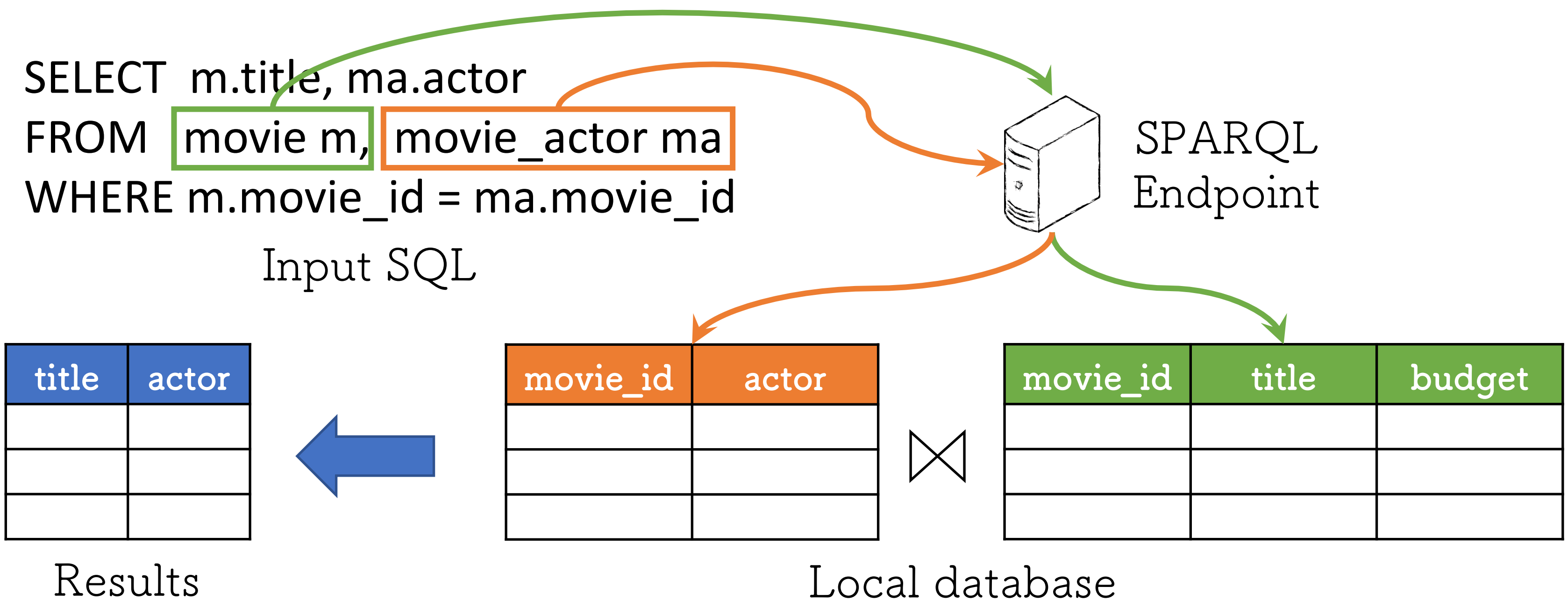
Naive Join Query Processing

- Steps
  1. Materialize joining views
  2. Perform relational join



View Query Merge – motivation

- Naive join is not efficient if two views are of same SPARQL endpoints.
  - Two separate SPARQL queries are performed.
  - Two sets of larger results are transferred.



View Query Merge – approach

- Idea:
  Combine two view SPARQL query on the same SPARQL endpoint into a single query.
- Approach:
  1. Combine projection variables into one set.
  2. Combine graph patterns into one pattern set.
  3. Put FILTER clause for join conditions.
     - If a condition is equality of same attribute names, corresponding FILTER clause is eliminated.

View Query Merge – example

Input SQL

```
SELECT  m.title, ma.actor
FROM   movie m,  movie_actor ma
WHERE m.movie_id = ma.movie_id
```

```
SELECT ?movie_id ?title ?budget
WHERE { ?movie_id   rdf:type     dbo:Film.
        ?movie_id   dc:title     ?title.
        ?movie_id dbo:budget  ?budget. }
```

Movie view

```
SELECT ?movie_id ?actor
 WHERE { ?movie_id   rdf:type      dbo:Film.
         ?movie_id  dbo:starring  ?actor. }
```

Movie-actor view

```
SELECT ?movie_id  ?title  ?budget  ?actor
WHERE { ?movie_id   rdf:type      dbo:Film.
        ?movie_id   dc:title      ?title.
        ?movie_id dbo:budget  ?budget.
        ?movie_id   rdf:type      dbo:Film.
        ?movie_id  dbo:starring ?actor.
        FILTER (movie_id = movie_id). }
```

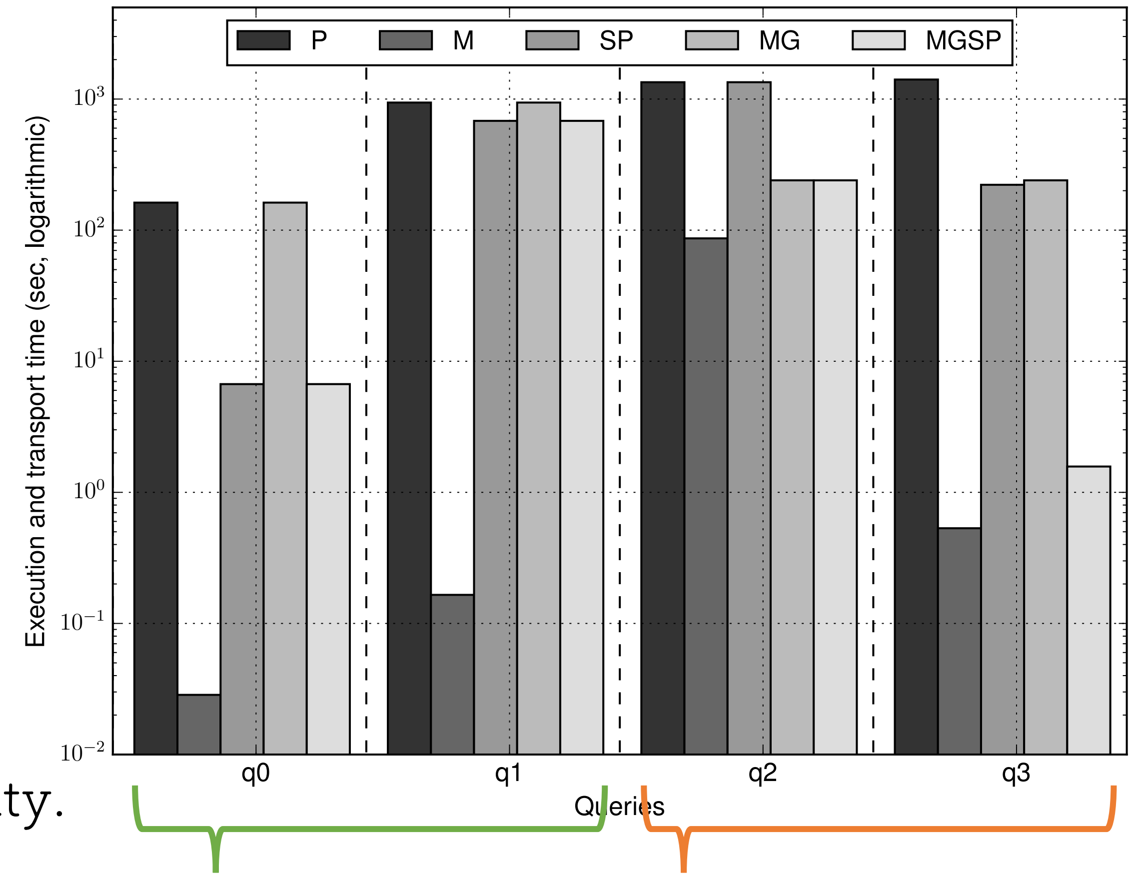
Merged SPARQL query

Experimental Evaluation

- Objective: Check efficiency of CROISSANT
  - Efficiency: query execution time
- Dataset: DBpedia (http://dbpedia.org/sparql)
- Views: movie, actor, movie_actor
- Queries (on the same SPARQL endpoint)
  - Two selection queries of different selectivity
    - Observe effect of selectivity
  - Two join queries w/ and w/o selection condition
    - Observe effect of view query merge
    - Observe overall performance of CROISSANT

Experimental Results

Bars
P: Pure execution
M: Materialization
SP: Selection push-down
MG: View query marge
MGSP: SP + MP



Insights

- Materialization is obviously the best.
- Selection push-down works when low selectivity.
- View query merge is **powerful** for join queries.

selection queries
left: low selectivity
right: high selectivity

join queries
left: w/o selection cond.
right: w/ selection cond.

Conclusion & Future Work

- CROISSANT
  - Centralized interface
  - Relational view-based interface
  - Query rewriting from SQL to SPARQL
  - Query optimization
    - Experimental evaluation introduces optimization techniques work.
- Future work
  - Update issue for taking full advantage of materialization
  - Unified optimization with materialization