Cooperative Techniques for SPARQL Query Relaxation in RDF Databases

Géraud FOKOU Stéphane JEAN Allel HADJALI
Mickaël BARON

LIAS/ENSMA and University of Poitiers, France

ESWC 2015, June 2015, Portoroz, Slovenia

Motivation

- A lot of Knowledge Bases (KBs) available on the web
  - Academic projects: YAGO, NELL, DBPedia, ...
  - Commercial projects: Google (Knowledge Vault), ...

- Difficulties to query KBs
  - Incomplete and Heterogeneous Data
  - Large size and complex schema semantics for end-users
  - Correct formulation of queries returning the intended result
  - ...

$\Rightarrow$ failing RDF queries: queries that return an empty set of answers

Motivation

- A lot of Knowledge Bases (KBs) available on the web
  - Academic projects: YAGO, NELL, DBPedia, ...
  - Commercial projects: Google (Knowledge Vault), ...

- Difficulties to query KBs
  - Incomplete and Heterogeneous Data
  - Large size and complex schema semantics for end-users
  - Correct formulation of queries returning the intended result
  - ...

$\Rightarrow$ failing RDF queries: queries that return an empty set of answers

The need for RDF query relaxation techniques

Related Work

- Proposed relaxation operators [Hurtado08, Fokou14, ...]
  - `OPTIONAL` of SPARQL
  - Classes/Properties Hierarchies (Student $\rightarrow$ Person)
  - Replacing constants by variables
  - Removing triple patterns
  - ...

$\Rightarrow$ On which triple patterns?

- Proposed relaxation process [Huang12, ...]
  - Based on similarity measure $Sim(Q, relax(Q))$
  - Execution from the most to the least similar relaxed queries

$\Rightarrow$ blind relaxation of the query (long execution time)

Related Work

- Proposed relaxation operators [Hurtado08, Fokou14, ...]
  - `OPTIONAL` of SPARQL
  - Classes/Properties Hierarchies (Student $\rightarrow$ Person)
  - Replacing constants by variables
  - Removing triple patterns
  - ...

$\Rightarrow$ On which triple patterns?

- Proposed relaxation process [Huang12, ...]
  - Based on similarity measure $Sim(Q, relax(Q))$
  - Execution from the most to the least similar relaxed queries

$\Rightarrow$ blind relaxation of the query (long execution time)

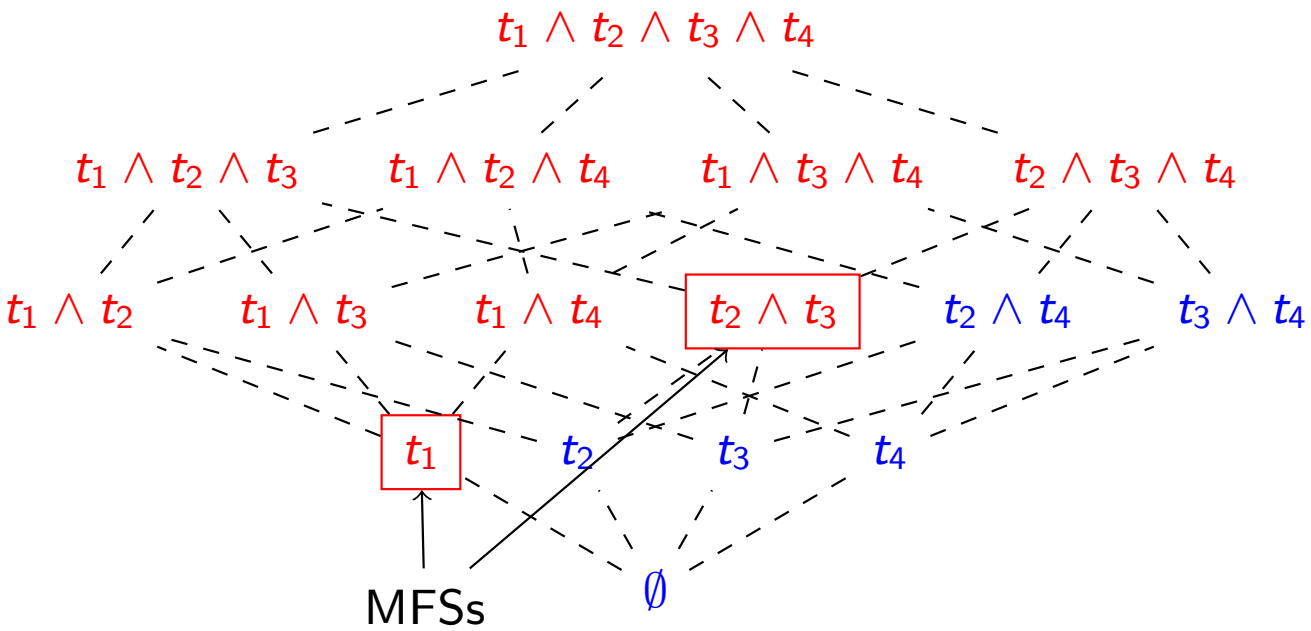
The need to know the causes of failure of an RDF query

Minimal Failing Subquery (MFS)

Definition

Let Q be a conjunction of triple patterns $Q = t_1 \wedge t_2 \wedge t_3 \wedge t_4$
Q* is an MFS of Q iff:

- Q* is a failing subquery of Q
- No subquery of Q* is a failing query



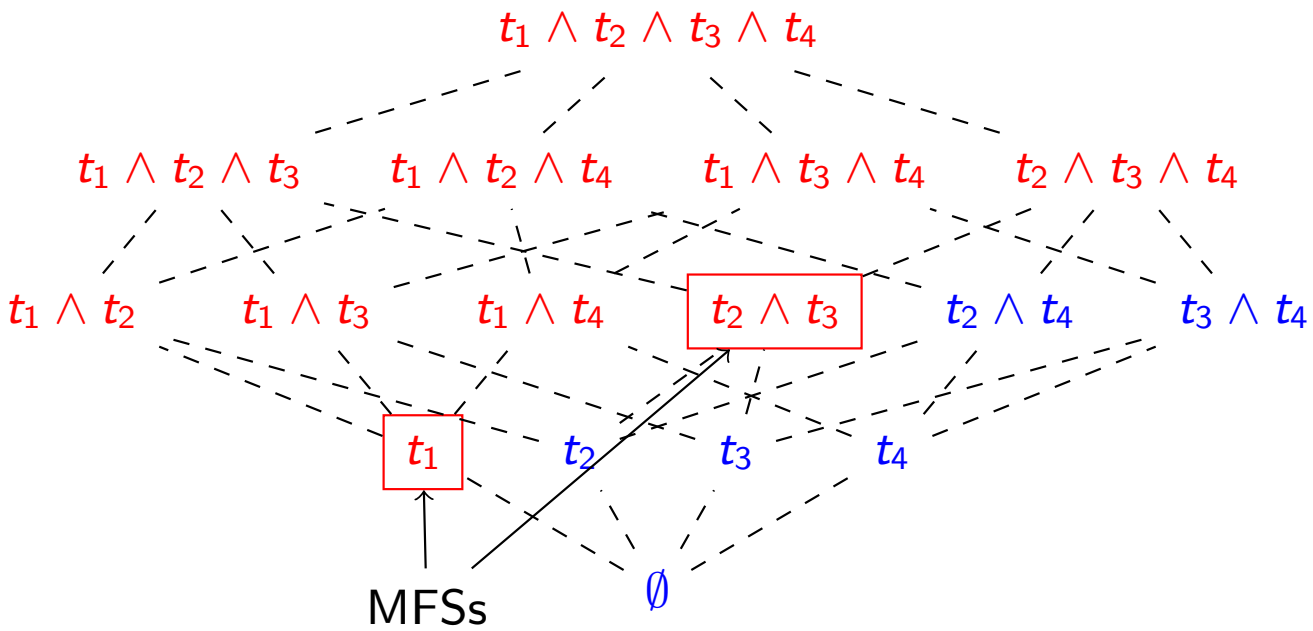
Q : successful query
Q : failing query

Minimal Failing Subquery (MFS)

Definition

Let Q be a conjunction of triple patterns $Q = t_1 \wedge t_2 \wedge t_3 \wedge t_4$
Q* is an MFS of Q iff:

- Q* is a failing subquery of Q
- No subquery of Q* is a failing query



Q : successful query
Q : failing query

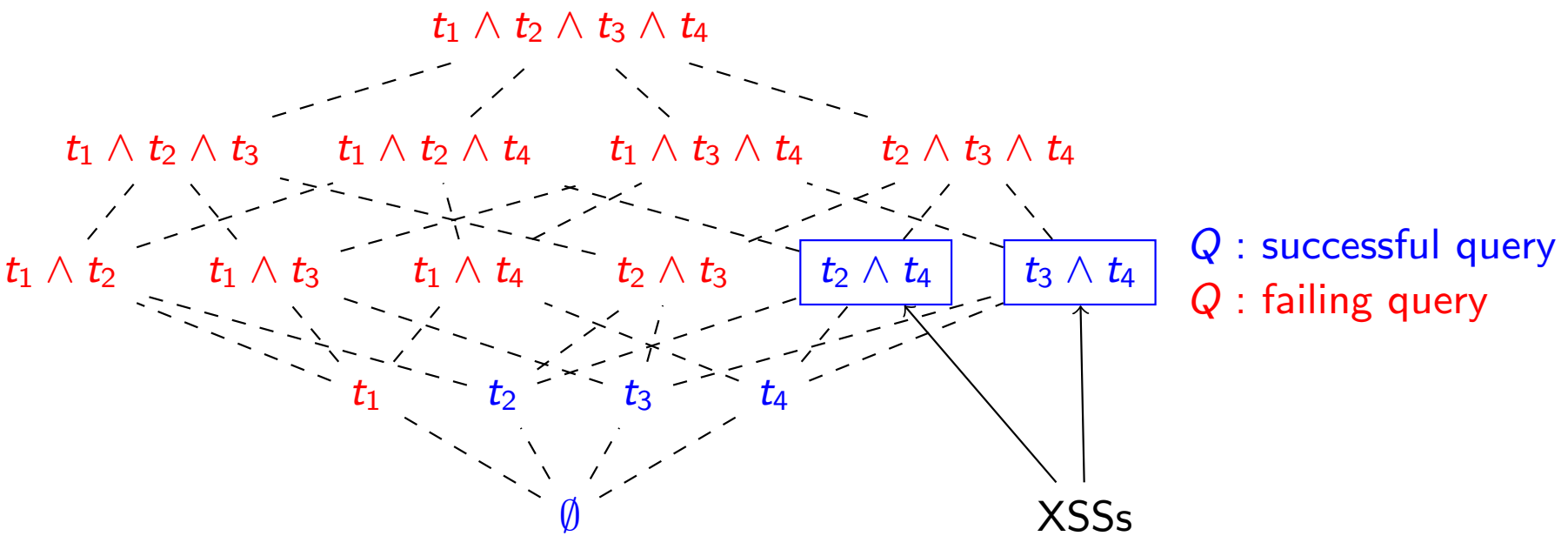
The need to find maximal non failing subquery $\Rightarrow$ notion of XSS

MaXimal Succeeding Subquery (XSS)

Definition

Q* is an XSS of Q iff:

- Q* is a successful subquery of Q
- Q* is not included in any successful subquery of Q

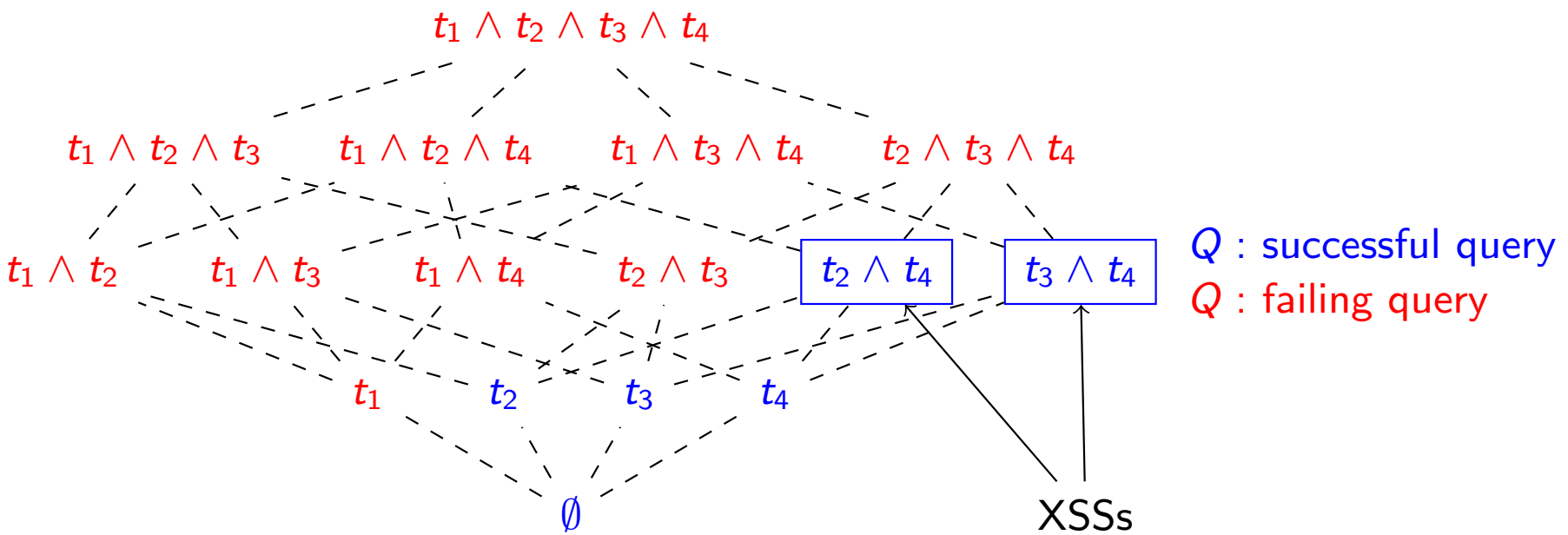


MaXimal Succeeding Subquery (XSS)

Definition

Q* is an XSS of Q iff:

- Q* is a successful subquery of Q
- Q* is not included in any successful subquery of Q



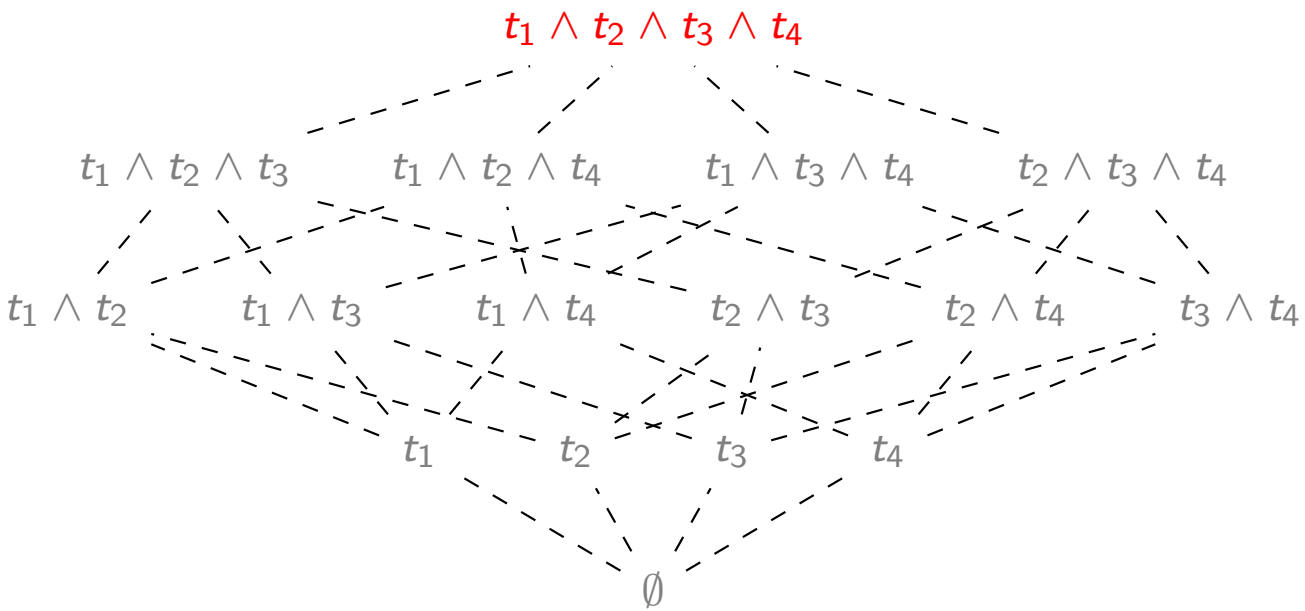
Problem: Computation of MFSs and XSSs of a failing RDF query

Outline

- Problem Statement
- **Lattice-Based Approach (LBA)**
- **Matrice-Based Approach (MBA)**
- **Experimental results**
- **Conclusion and Perspectives**

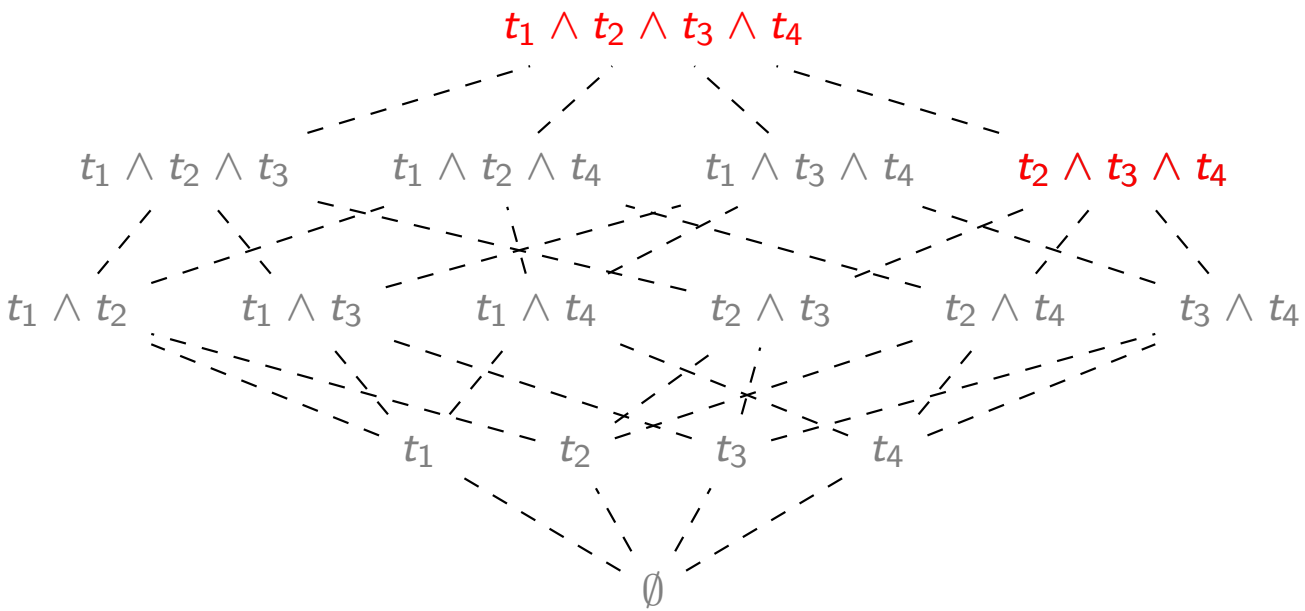
LBA Approach: 1) Finding an MFS

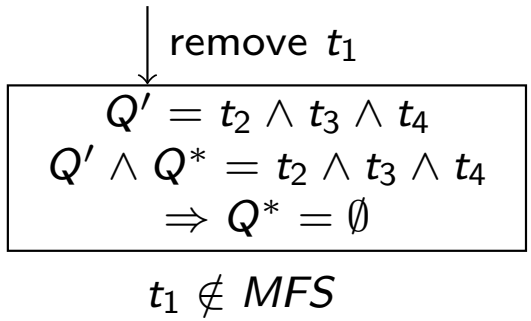
- Inspired by the work of Godfrey in RDBMSs [Godfrey97]
- A 3 steps procedure
- **First step:** $Q^* \leftarrow FindAnMFS(Q)$
  - remove iteratively each triple pattern t_i of Q (query Q')
  - if $Q' \wedge Q^*$ is successful, t_i is an element of Q^* (Proposition)
  - otherwise Q^* is a subquery of $Q' \wedge Q^*$



LBA Approach: 1) Finding an MFS

- Inspired by the work of Godfrey in RDBMSs [Godfrey97]
- A 3 steps procedure
- **First step:** $Q^* \leftarrow FindAnMFS(Q)$
  - remove iteratively each triple pattern t_i of Q (query Q')
  - if $Q' \wedge Q^*$ is successful, t_i is an element of Q^* (Proposition)
  - otherwise Q^* is a subquery of $Q' \wedge Q^*$

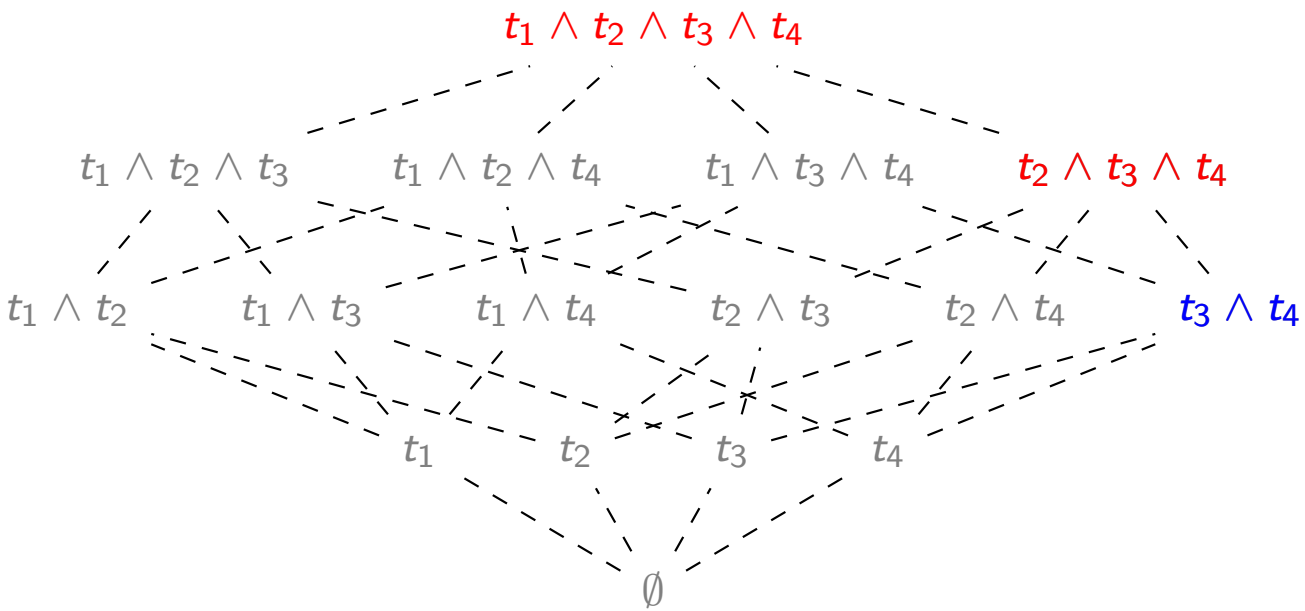


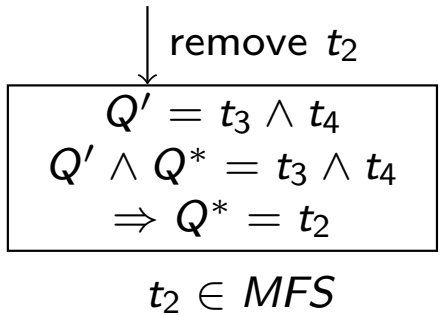


$t_1 \notin MFS$

LBA Approach: 1) Finding an MFS

- Inspired by the work of Godfrey in RDBMSs [Godfrey97]
- A 3 steps procedure
- **First step:** $Q^* \leftarrow FindAnMFS(Q)$
  - remove iteratively each triple pattern t_i of Q (query Q')
  - if $Q' \wedge Q^*$ is successful, t_i is an element of Q^* (Proposition)
  - otherwise Q^* is a subquery of $Q' \wedge Q^*$

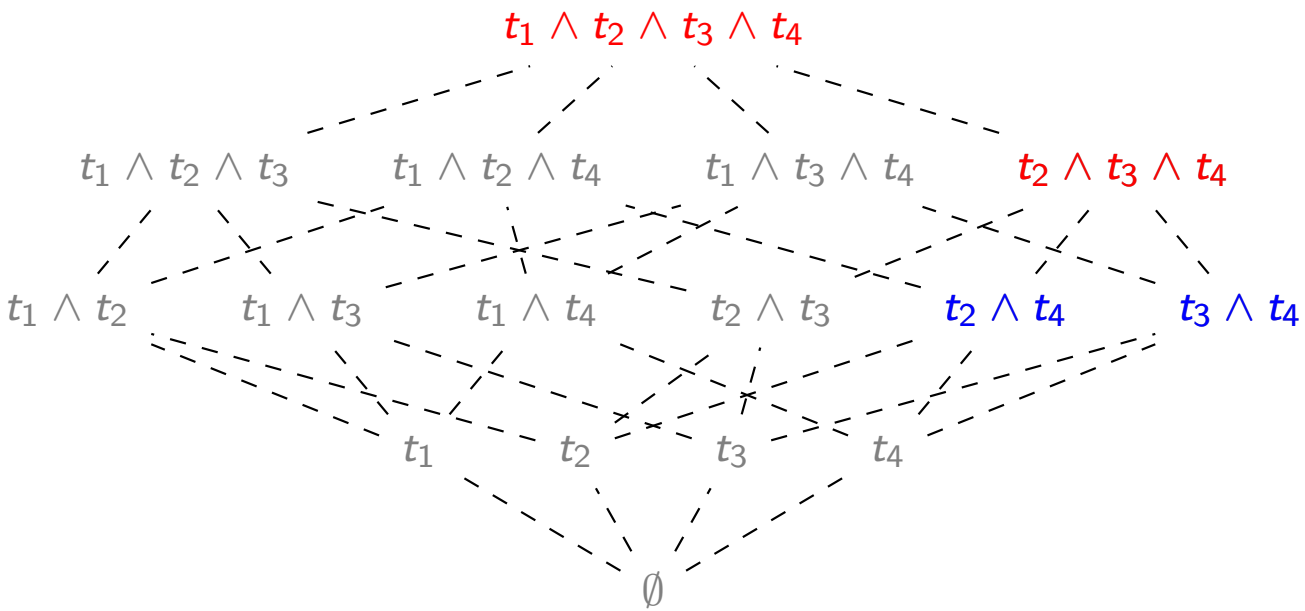


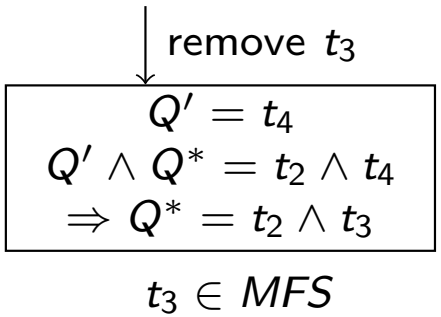


$t_2 \in MFS$

LBA Approach: 1) Finding an MFS

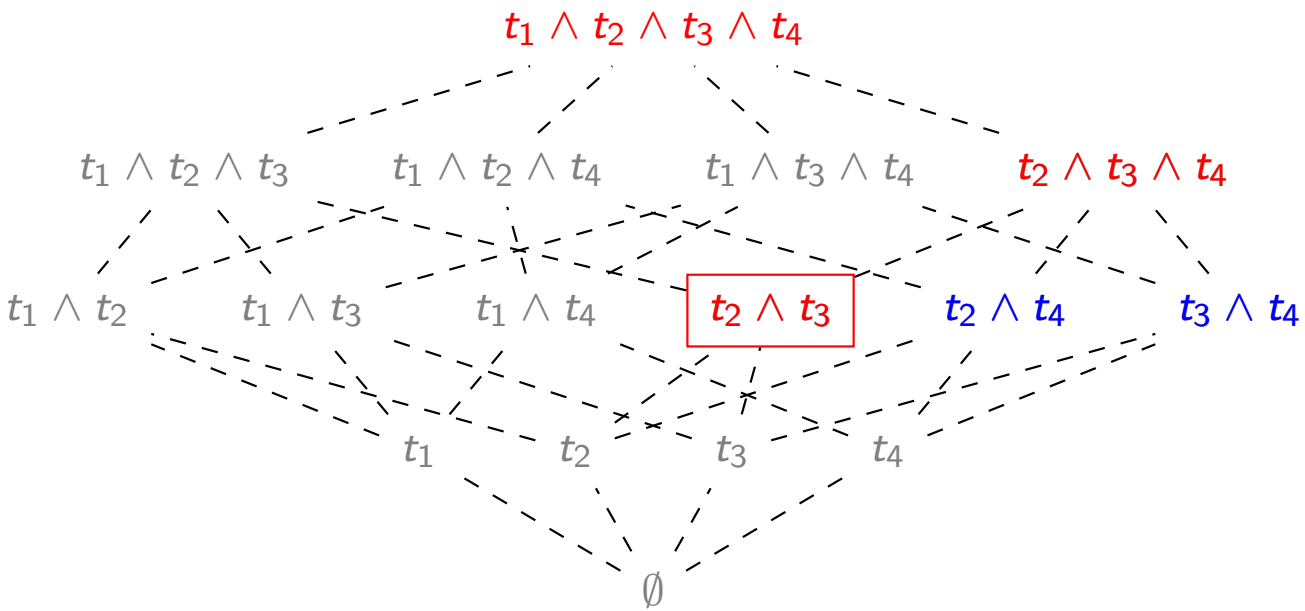
- Inspired by the work of Godfrey in RDBMSs [Godfrey97]
- A 3 steps procedure
- **First step:** $Q^* \leftarrow FindAnMFS(Q)$
  - remove iteratively each triple pattern t_i of Q (query Q')
  - if $Q' \wedge Q^*$ is successful, t_i is an element of Q^* (Proposition)
  - otherwise Q^* is a subquery of $Q' \wedge Q^*$

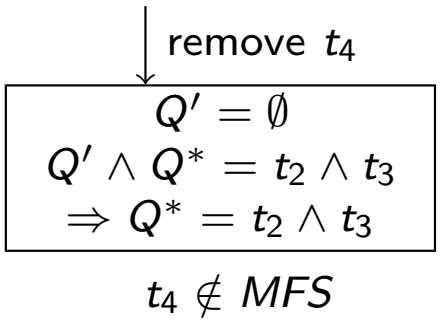




LBA Approach: 1) Finding an MFS

- Inspired by the work of Godfrey in RDBMSs [Godfrey97]
- A 3 steps procedure
- **First step:** $Q^* \leftarrow FindAnMFS(Q)$
  - remove iteratively each triple pattern t_i of Q (query Q')
  - if $Q' \wedge Q^*$ is successful, t_i is an element of Q^* (Proposition)
  - otherwise Q^* is a subquery of $Q' \wedge Q^*$

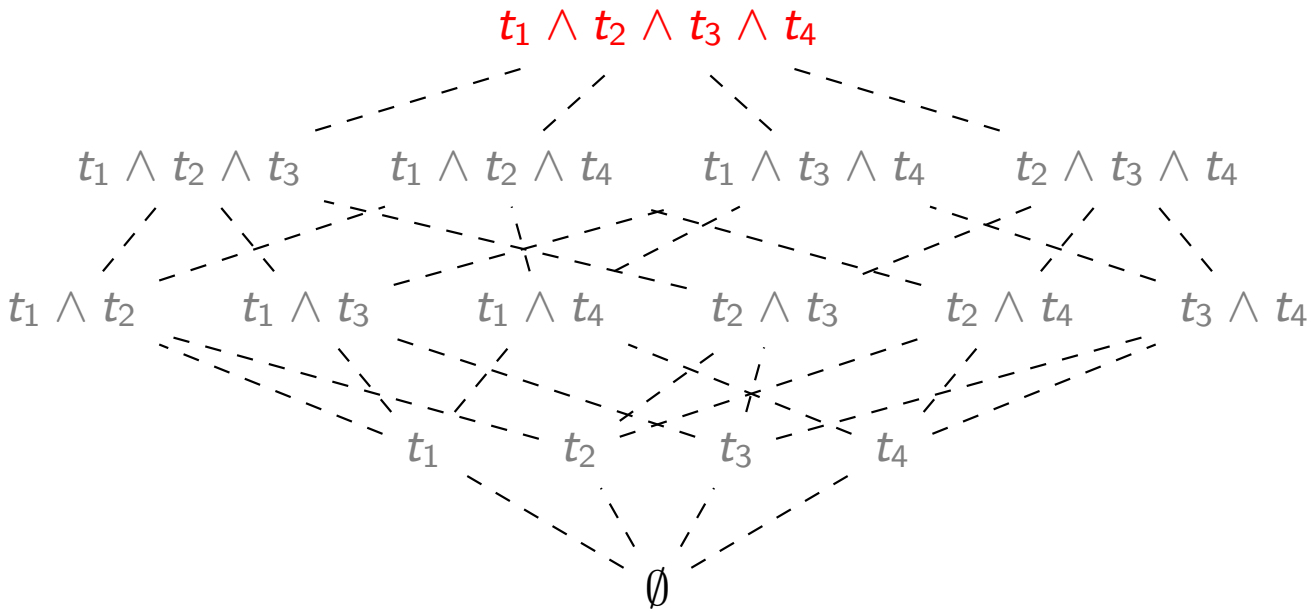




LBA Approach: 2) Computing Potential XSSs

Second step:

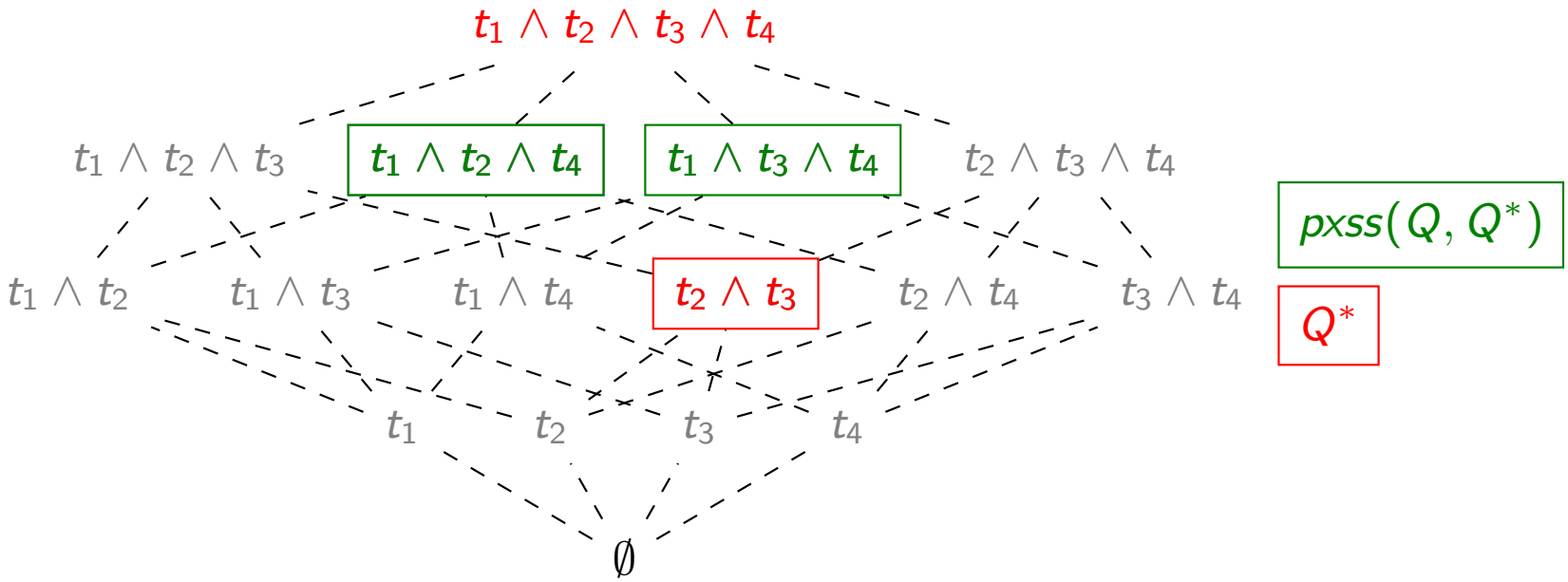
- All queries that include Q^* are failing
  $\Rightarrow$ they can be neither MFS nor XSS
- Continue with the largest subqueries of Q that do not include Q^*. If they are successful, they are XSSs.
  $\Rightarrow$ potential XSSs (PXSS): $pxss(Q, Q^*) = \{Q - t_i \mid t_i \in Q^*\}$
- $Q^* = t_2 \wedge t_3 \Rightarrow pxss(Q, Q^*) = \{t_1 \wedge t_3 \wedge t_4, t_1 \wedge t_2 \wedge t_4\}$



LBA Approach: 2) Computing Potential XSSs

Second step:

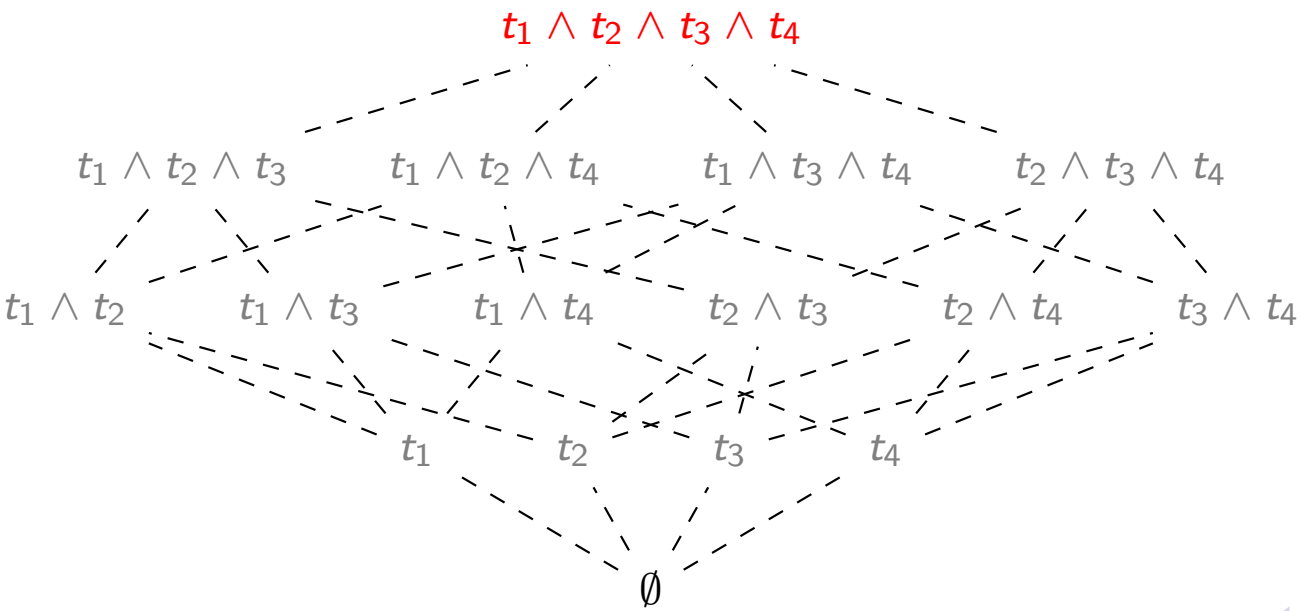
- All queries that include Q^* are failing
  $\Rightarrow$ they can be neither MFS nor XSS
- Continue with the largest subqueries of Q that do not include Q^*. If they are successful, they are XSSs.
  $\Rightarrow$ potential XSSs (PXSS): $pxss(Q, Q^*) = \{Q - t_i \mid t_i \in Q^*\}$
- $Q^* = t_2 \wedge t_3 \Rightarrow pxss(Q, Q^*) = \{t_1 \wedge t_3 \wedge t_4, t_1 \wedge t_2 \wedge t_4\}$



LBA Approach: 3) Finding all MFSs and XSSs

Third step:

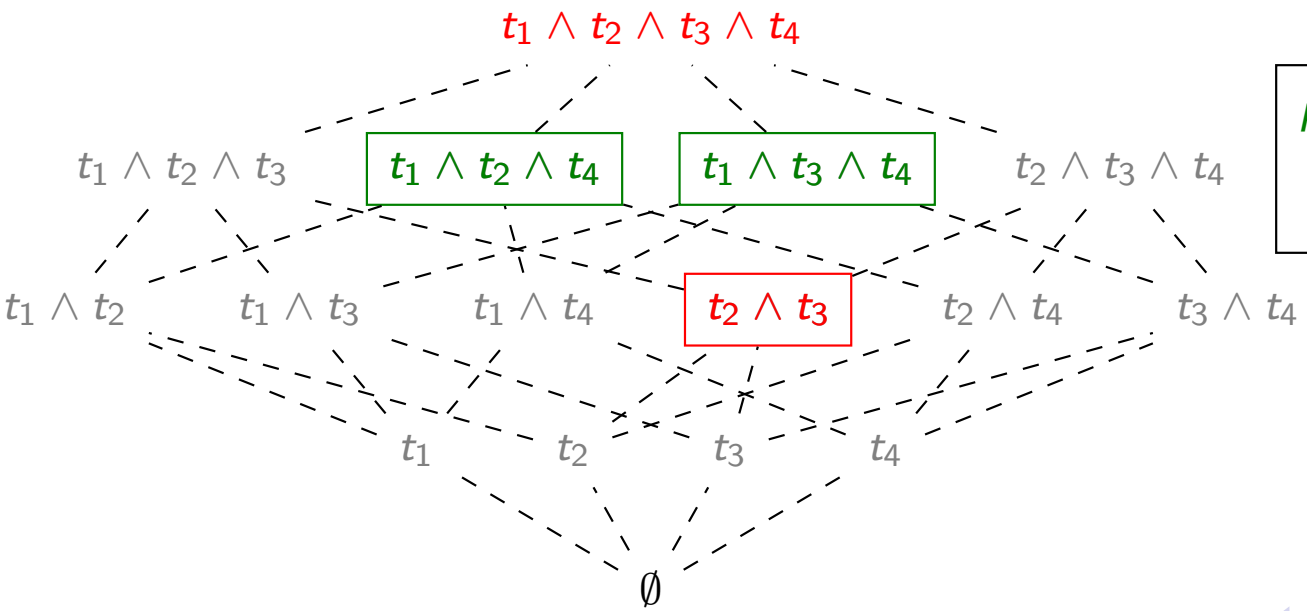
- Naïve approach
  - Execute each PXSS PQ
  - If PQ is successful, this is an XSS
  - Otherwise apply recursively the LBA algorithm on PQ
- Problem: the same MFS can be found several times
- Solution: incremental computation of the PXSSs that do not include the identified MFSs



LBA Approach: 3) Finding all MFSs and XSSs

Third step:

- Naïve approach
  - Execute each PXSS PQ
  - If PQ is successful, this is an XSS
  - Otherwise apply recursively the LBA algorithm on PQ
- Problem: the same MFS can be found several times
- Solution: incremental computation of the PXSSs that do not include the identified MFSs



$t_1 \wedge t_2 \wedge t_3 \wedge t_4$

$Q^* = t_2 \wedge t_3$

$pxss = \{t_1 \wedge t_3 \wedge t_4, t_1 \wedge t_2 \wedge t_4\}$

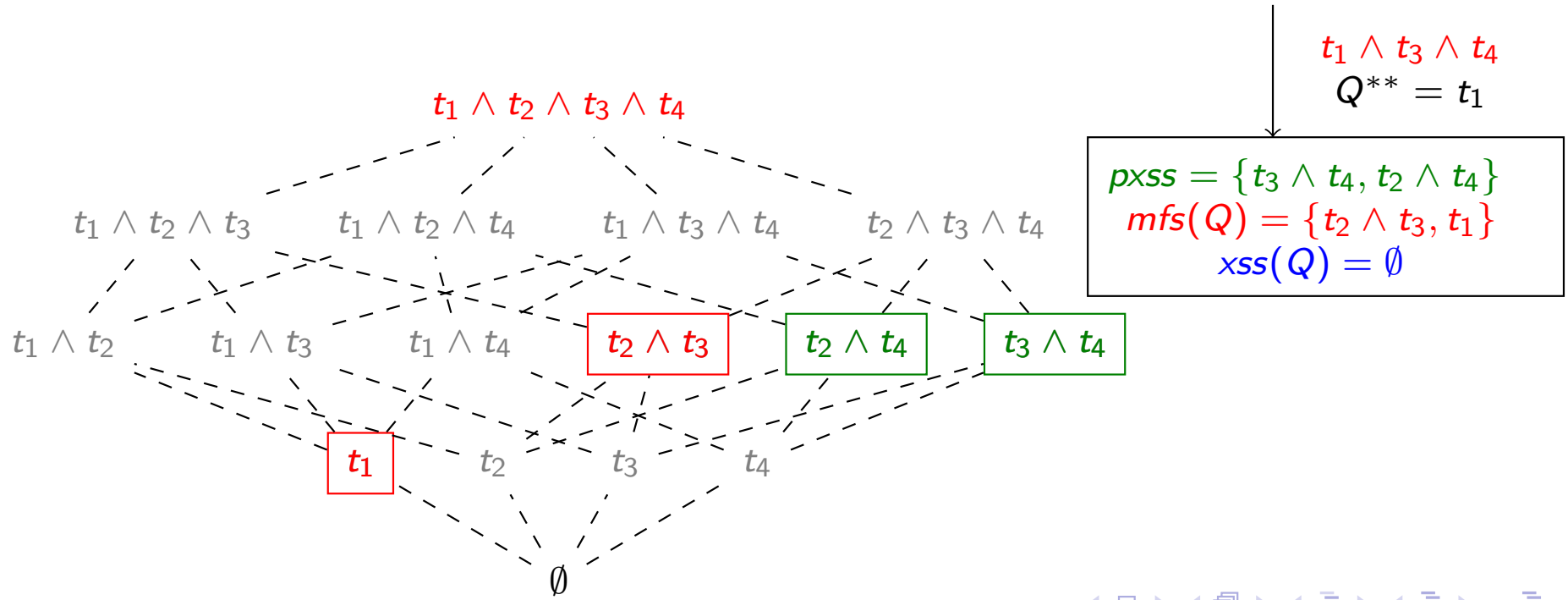
$mfs(Q) = \{t_2 \wedge t_3\}$

$xss(Q) = \emptyset$

LBA Approach: 3) Finding all MFSs and XSSs

Third step:

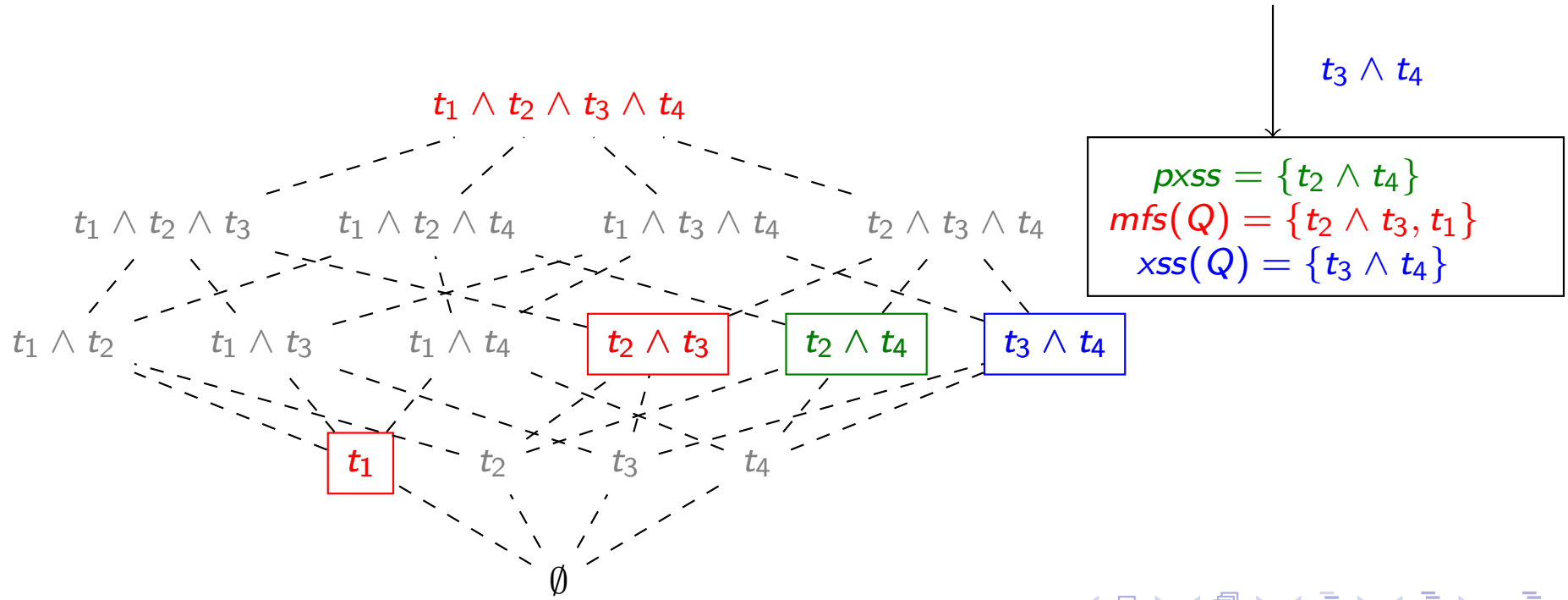
- Naïve approach
  - Execute each PXSS PQ
  - If PQ is successful, this is an XSS
  - Otherwise apply recursively the LBA algorithm on PQ
- Problem: the same MFS can be found several times
- Solution: incremental computation of the PXSSs that do not include the identified MFSs



LBA Approach: 3) Finding all MFSs and XSSs

Third step:

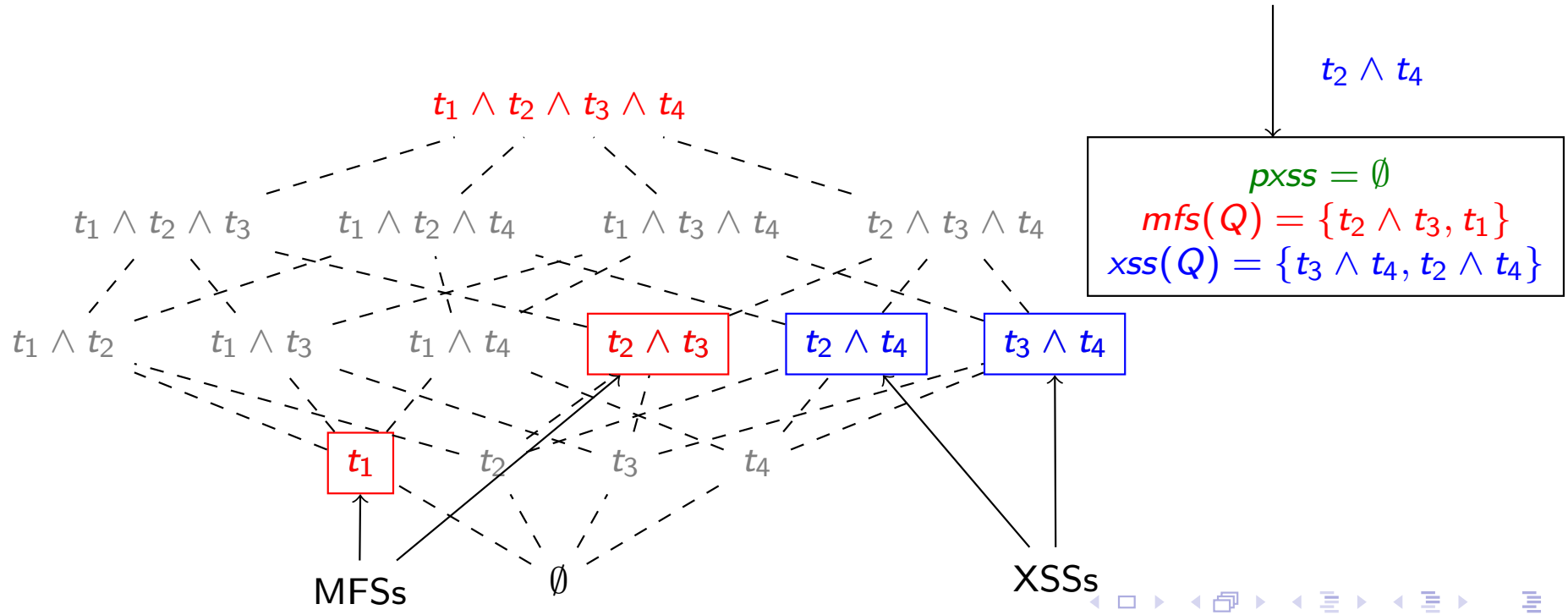
- Naïve approach
  - Execute each PXSS PQ
  - If PQ is successful, this is an XSS
  - Otherwise apply recursively the LBA algorithm on PQ
- Problem: the same MFS can be found several times
- Solution: incremental computation of the PXSSs that do not include the identified MFSs



LBA Approach: 3) Finding all MFSs and XSSs

Third step:

- Naïve approach
  - Execute each PXSS PQ
  - If PQ is successful, this is an XSS
  - Otherwise apply recursively the LBA algorithm on PQ
- Problem: the same MFS can be found several times
- Solution: incremental computation of the PXSSs that do not include the identified MFSs



Outline

- Problem Statement
- Lattice-Based Approach (LBA)
- Matrice-Based Approach (MBA)
- Experimental results
- Conclusion and Perspectives

MBA Approach: Overview

- Inspired by the work of Jannach in Recommender Systems
- Based on the computation of a Matrix
  - one row by potential solution μ
    (a mapping satisfying at least one triple pattern)
  - one column by triple pattern
  - $M[\mu][t_i] = 1 \Leftrightarrow \mu$ satisfies t_i (else 0)

t		
s	**p**	**o**
Smith	type	Professor
Smith	research	SW
Smith	teacherOf	DB
Jane	type	Lecturer
Jane	research	DB
Jane	teacherOf	DB

RDF triples

```
SELECT ?p ?a WHERE {
?p age ?a                (t1)
?p type Lecturer         (t2)
?p research SW           (t3)
?p teacherOf DB }        (t4)
```

The query Q

?p	?a	t_1	t_2	t_3	t_4
Smith	null	0	0	1	1
Jane	null	0	1	0	1

The relaxed matrix of Q

$xss(Q) = \{t_2 \wedge t_4, t_3 \wedge t_4\}$
$mfs(Q) = \{t_1, t_2 \wedge t_3\}$

The MFSs and XSSs of Q

MBA Approach: Relaxed Matrix Computation

- Execution of each triple pattern t_i
- Computation of $t_1 \overset{*}{\bowtie} t_2 \ldots \overset{*}{\bowtie} t_n$
  - where $t_1 \overset{*}{\bowtie} t_2 = t_1 \cup t_2 \cup t_1 \bowtie t_2$
  - the mappings that satisfies t_1 or t_2 or a combination of t_1, t_2
- Problem: subqueries of Q can lead to Cartesian Product
- $\Rightarrow$ Matrix of a huge size (exceeds the size of main memory)
- Not the case of **Star-shaped queries**
  (a shared join variable in the subject position)

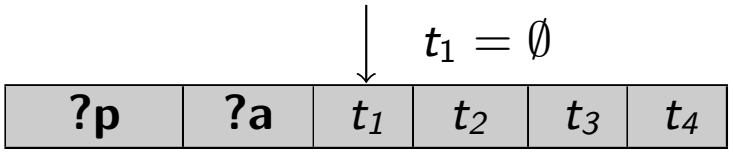
```
SELECT ?p ?a WHERE {
?p age ?a                (t1)
?p type Lecturer         (t2)
?p research SW           (t3)
?p teacherOf DB }        (t4)
```

MBA Approach: Optimized Relaxed Matrix Computation

- Algorithm NQ
  - execute each triple pattern t_i
  - for each value v of the join variable
  - if v is already in the matrix, complete the row with a 1 for t_i
  - else add a new row in the matrix with only a 1 for t_i

MBA Approach: Optimized Relaxed Matrix Computation

- Algorithm NQ
  - execute each triple pattern t_i
  - for each value v of the join variable
  - if v is already in the matrix, complete the row with a 1 for t_i
  - else add a new row in the matrix with only a 1 for t_i



MBA Approach: Optimized Relaxed Matrix Computation

- Algorithm NQ
  - execute each triple pattern t_i
  - for each value v of the join variable
  - if v is already in the matrix, complete the row with a 1 for t_i
  - else add a new row in the matrix with only a 1 for t_i

$t_1 = \emptyset$

?p	?a	t_1	t_2	t_3	t_4

$t_2 = Jane$

?p	?a	t_1	t_2	t_3	t_4
Jane	null	0	1	0	0

MBA Approach: Optimized Relaxed Matrix Computation

- Algorithm NQ
  - execute each triple pattern t_i
  - for each value v of the join variable
  - if v is already in the matrix, complete the row with a 1 for t_i
  - else add a new row in the matrix with only a 1 for t_i

$t_1 = \emptyset$

?p	?a	t_1	t_2	t_3	t_4

$t_2 = Jane$

?p	?a	t_1	t_2	t_3	t_4
Jane	null	0	1	0	0

$t_3 = Smith$

?p	?a	t_1	t_2	t_3	t_4
Jane	null	0	1	0	0
Smith	null	0	0	1	0

MBA Approach: Optimized Relaxed Matrix Computation

- Algorithm NQ
  - execute each triple pattern t_i
  - for each value v of the join variable
  - if v is already in the matrix, complete the row with a 1 for t_i
  - else add a new row in the matrix with only a 1 for t_i

$t_1 = \emptyset$

?p	?a	t_1	t_2	t_3	t_4

$t_2 = Jane$

?p	?a	t_1	t_2	t_3	t_4
Jane	null	0	1	0	0

$t_3 = Smith$

?p	?a	t_1	t_2	t_3	t_4
Jane	null	0	1	0	0
Smith	null	0	0	1	0

$t_4 = Smith, Jane$

?p	?a	t_1	t_2	t_3	t_4
Jane	null	0	1	0	1
Smith	null	0	0	1	1

MBA Approach: Optimized Relaxed Matrix Computation

- Algorithm 1Q
  - if the RDF database is implemented as a triple table: T(s, p, o)
  - a single SQL query can be used to compute the matrix
  - roughly the translation of $t_1 \ltimes t_2 \ldots \ltimes t_n$

```
select coalesce(t1.s , t2.s, t3.s, t4.s),
       case when t1.s is null then 0 else 1 end as t1,
       case when t2.s is null then 0 else 1 end as t2,
       case when t3.s is null then 0 else 1 end as t3,
       case when t4.s is null then 0 else 1 end as t4
from t1 full outer join t2 on t1.s = t2.s
    full outer join t3 on coalesce(t1.s , t2.s) = t3.s
    full outer join t4 on coalesce(t1.s , t2.s, t3.s) = t4.s
```

MBA Approach: MFSs and XSSs computation

- XSSs computation
  1) Compute the skyline of the relaxed matrix
     Many solutions: block nested loop, sort filter skyline, ...
  2) Retrieve the queries that correspond to skyline rows
     They are the XSSs

?p	?a	t_1	t_2	t_3	t_4
Smith	null	0	0	1	1
Jane	null	0	1	0	1

$$\Rightarrow xss(Q) = \{t_2 \wedge t_4, t_3 \wedge t_4\}$$

- MFSs computation
  - The matrix provides a simple way to know if a query is failing
  - $t_2 \wedge t_3$ is failing $\Leftrightarrow col(t_2)$ AND $col(t_3) = \binom{0}{0}$
  - The matrix can be used as an index for optimizing LBA
    LBA without any database query

Outline

- Problem Statement
- Lattice-Based Approach (LBA)
- Matrice-Based Approach (MBA)
- **Experimental results**
- **Conclusion and Perspectives**

Experimental Setup

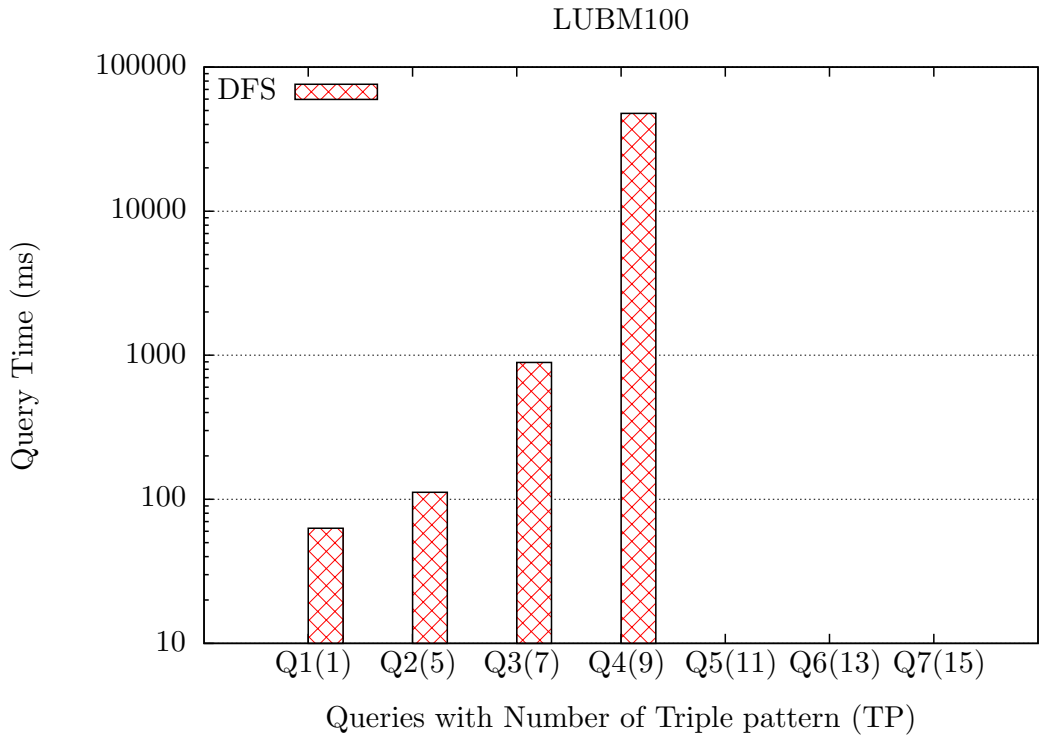
- Java implementation of LBA and MBA on top of Jena TDB
  `http://www.lias-lab.fr/forge/projects/qars`
- Dataset: LUBM20 ($\approx$ 300 MB) and LUBM100 ($\approx$ 1 GB)
- Queries: 7 failing queries ranging 1 to 15 triple patterns (TP)
- Hardware: Intel Core i7 CPU and 8GB RAM
- Implementation of the relaxed matrix as a set of Bitmap
- Relaxed matrix size and computation time:

	LUBM20				LUBM100		
	Q3	Q5	Q7		Q3	Q5	Q7
Computation time with NQ (in sec)	8.6	8.6	8.6		42.6	43.4	44.6
Computation time with 1Q (in sec)	6.1	6.3	6.8		30.4	34.6	38.5
Size (in KB)	293	400	335		1385	1912	1590
Number of rows (in K)	430	430	430		2149	2149	2149

The algorithm 1Q is 25% faster than NQ
The size of the matrix is small thanks to bitmap compression

Experiments: XSS and MFS Computation Time

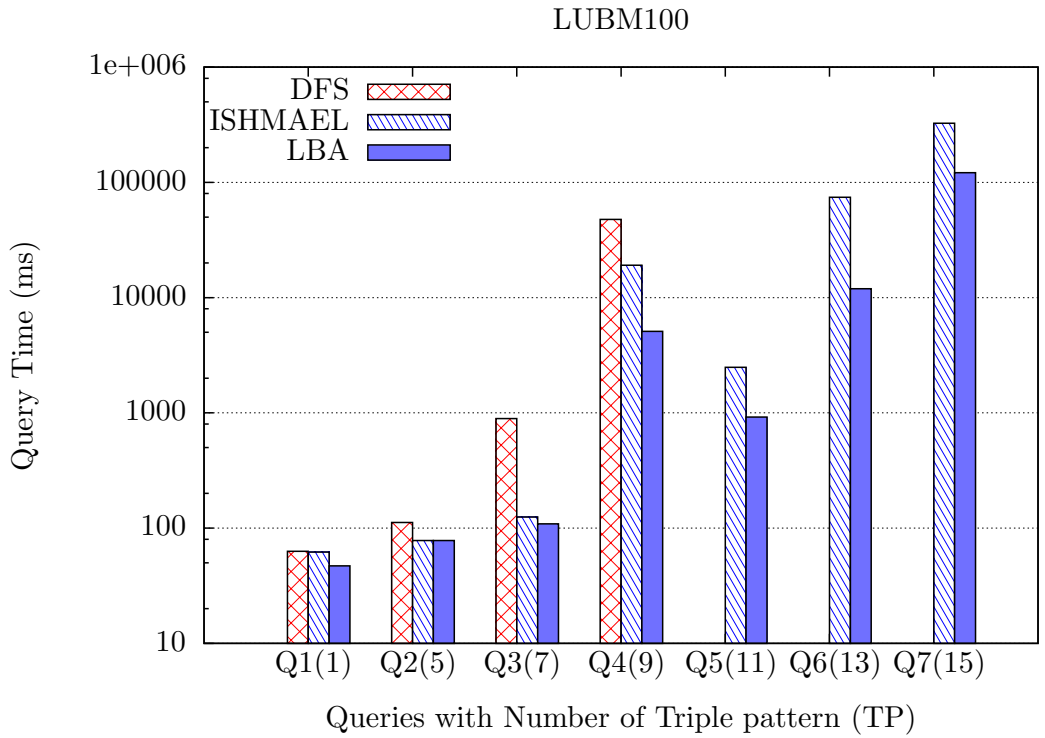
- DFS: Depth-First Search algorithm (baseline method)



DFS does not scale for medium size queries ($\approx$ 9TP)

Experiments: XSS and MFS Computation Time

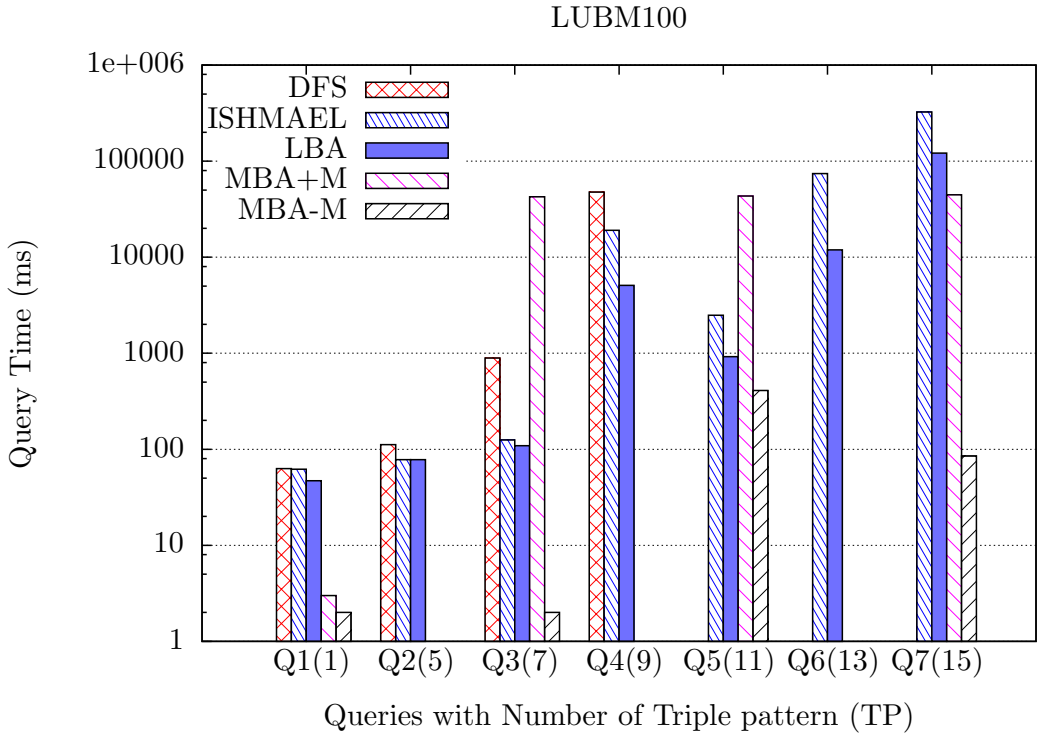
- DFS: Depth-First Search algorithm (baseline method)
- ISHMAEL: adaptation of Godfrey'algorithm, LBA



LBA/ISHMAEL scale till queries $\approx$ 11 TP

Experiments: XSS and MFS Computation Time

- DFS: Depth-First Search algorithm (baseline method)
- ISHMAEL: adaptation of Godfrey'algorithm, LBA
- MBA+M: computation of the matrix + LBA algorithm
- MBA-M: same as MBA+M but without computing the matrix

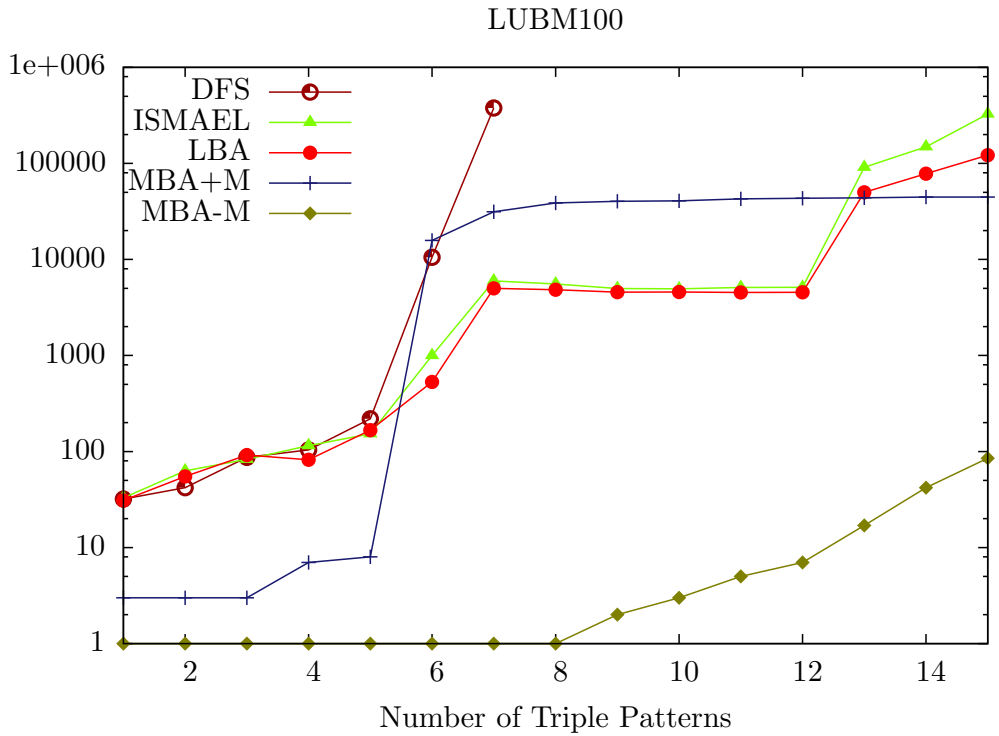


MBA is only useful for Star-shaped queries

MBA+M only useful for large queries, MBA-M scales well

Experiments: Performance as the number of TP scales

- The number of TP plays a important role in the performance
- Experiment: decompose Q7 in 15 subqueries (1 to 15TP)
- Execute each such subquery



⇒ Confirm our previous results

Outline

- Problem Statement
- Lattice-Based Approach (LBA)
- Matrice-Based Approach (MBA)
- Experimental results
- **Conclusion and Perspectives**

Conclusion and Perspectives

- Study of 2 cooperative techniques for RDF query relaxation
  - MFS: a cause of query failure
  - XSS: a relaxed query with a maximum of TP
- Contribution: 2 solutions to find MFSs/XSSs of RDF queries
  - LBA is smart exploration of the subquery lattice
  - MBA is based on a matrix used as a bitmap index
  - LBA >> baseline algorithms for queries with more than 5 TPs
  - MBA is only interesting for large star-shaped queries
  - if the matrix has been precomputed, MBA runs in milliseconds even for large queries (>15TP)
- Perspectives:
  - Optimization of LBA using heuristics (obvious MFS)
  - Optimization of MBA for other kinds of RDF queries
  - Definition of query relaxation strategies based on MFSs/XSSs