Structured Materialized Views for XML Queries

Andrei Arion[1,2] Véronique Benzaken[2]
Ioana Manolescu[1] Yannis Papakonstantinou[3]

[1]GEMO Project, INRIA Futurs
firstname.lastname@inria.fr

[2]Laboratoire de Recherche en Informatique, University Paris XI
firstname.lastname@lri.fr

[3]University of California San Diego, USA
firstname@cs.ucsd.edu

VLDB 2007, September 25th, 2007

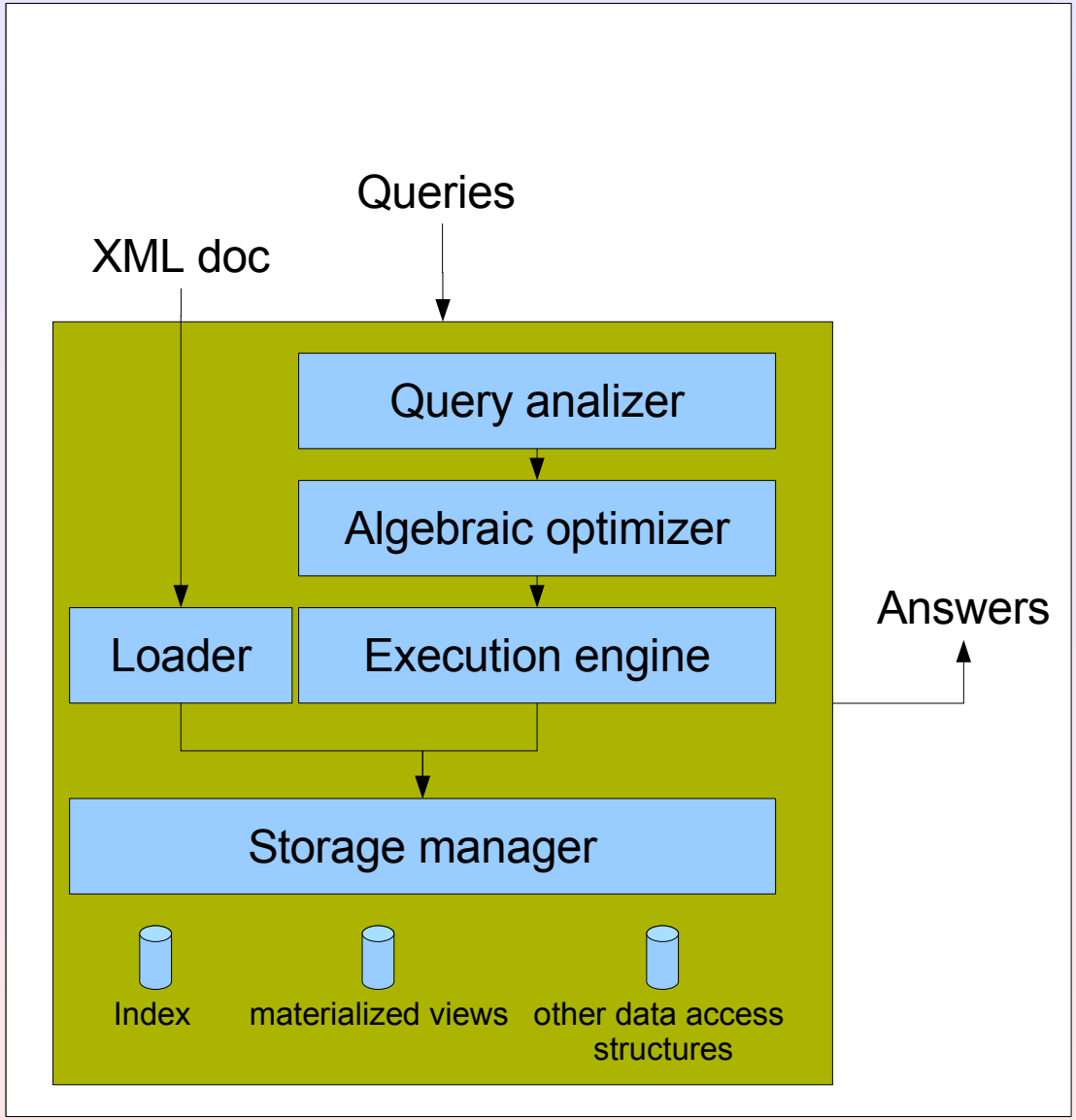
Outline

1. Context: rewriting XQuery using nested views
2. Pattern containment under summary constraints
   - Path summary
   - Summary based containment
3. Summary based query rewriting
4. Experimental results
5. Conclusions

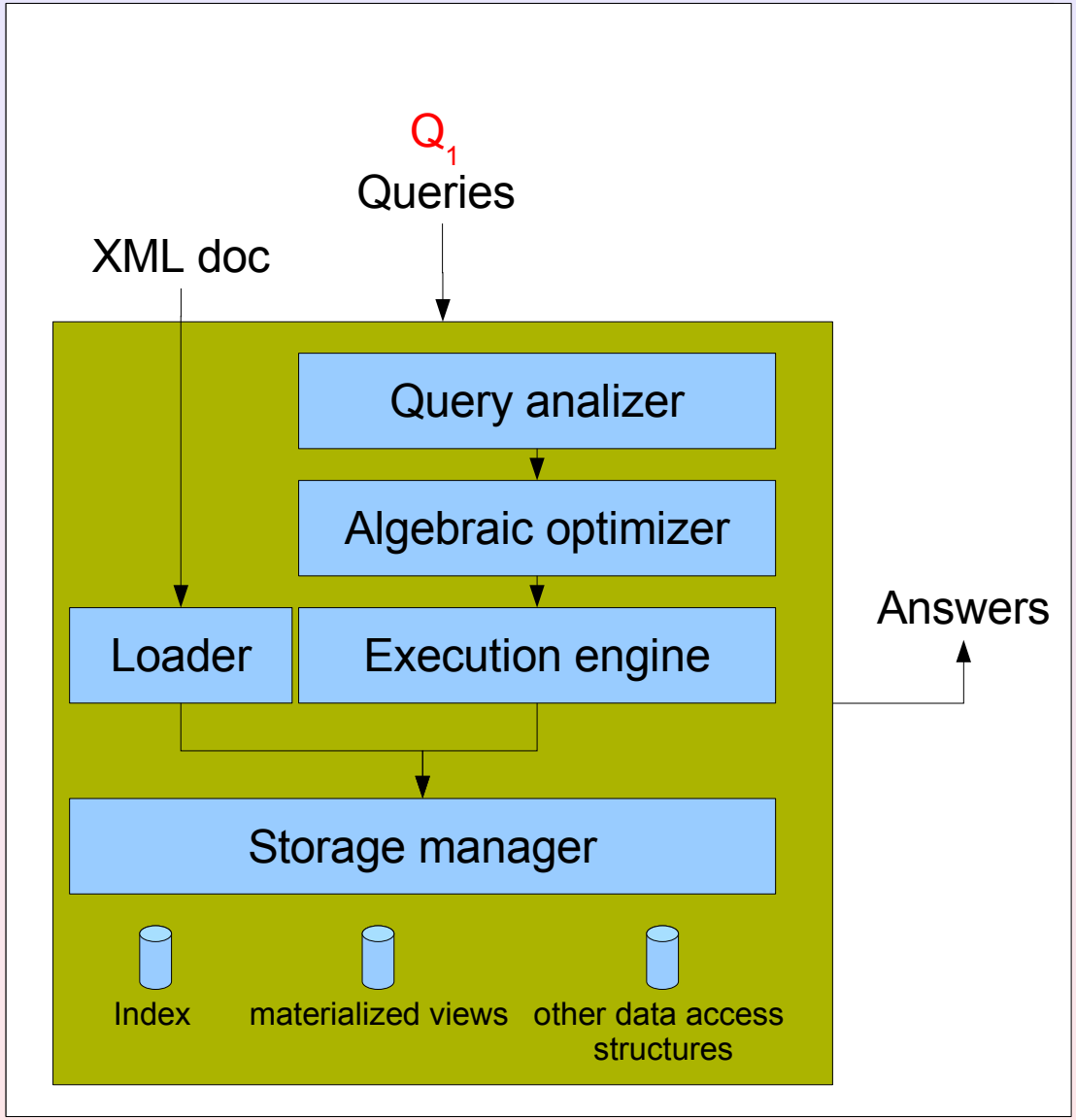
Outline

1. Context: rewriting XQuery using nested views
2. Pattern containment under summary constraints
   - Path summary
   - Summary based containment
3. Summary based query rewriting
4. Experimental results
5. Conclusions

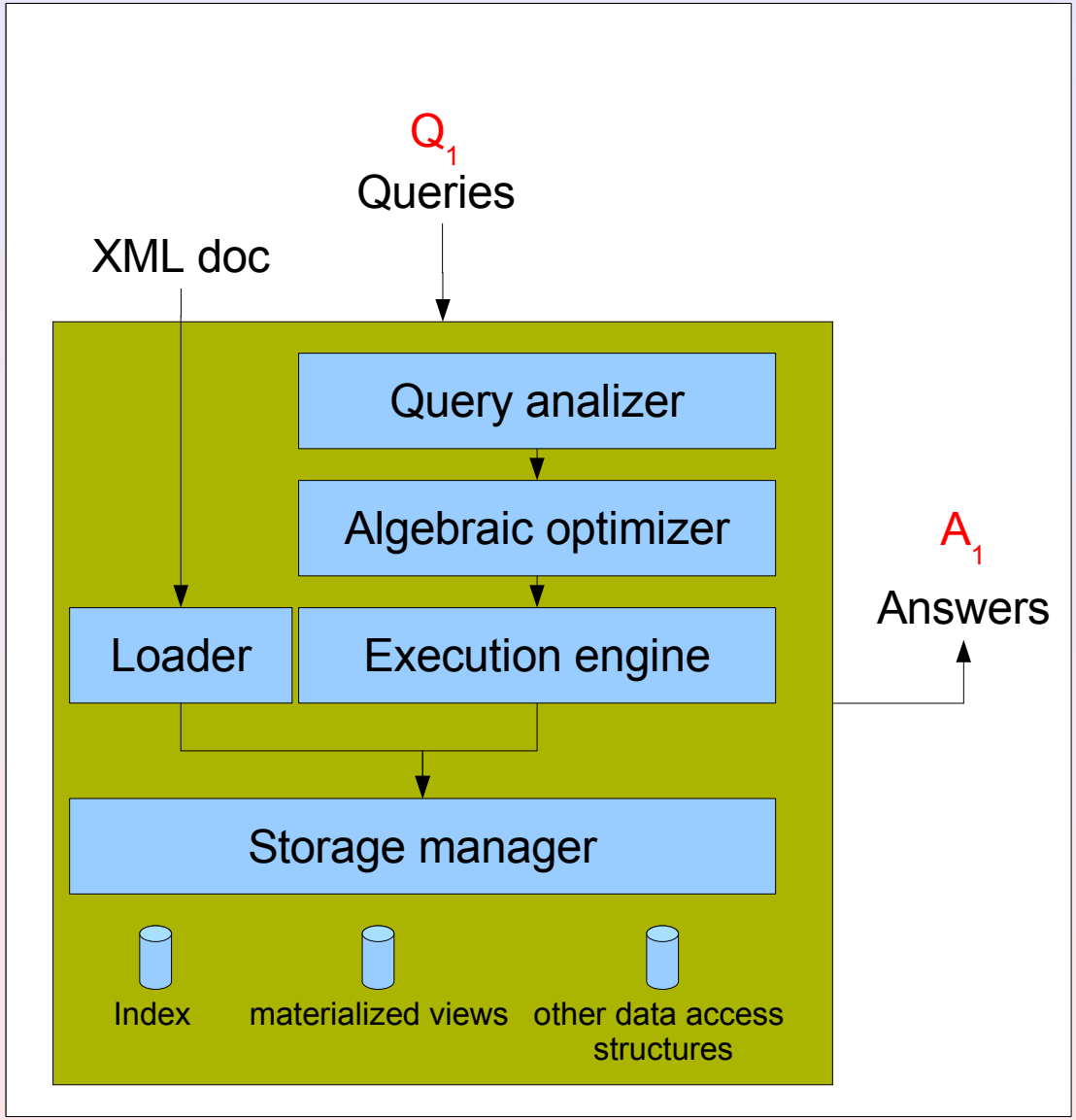
Context: rewriting XQuery using nested views



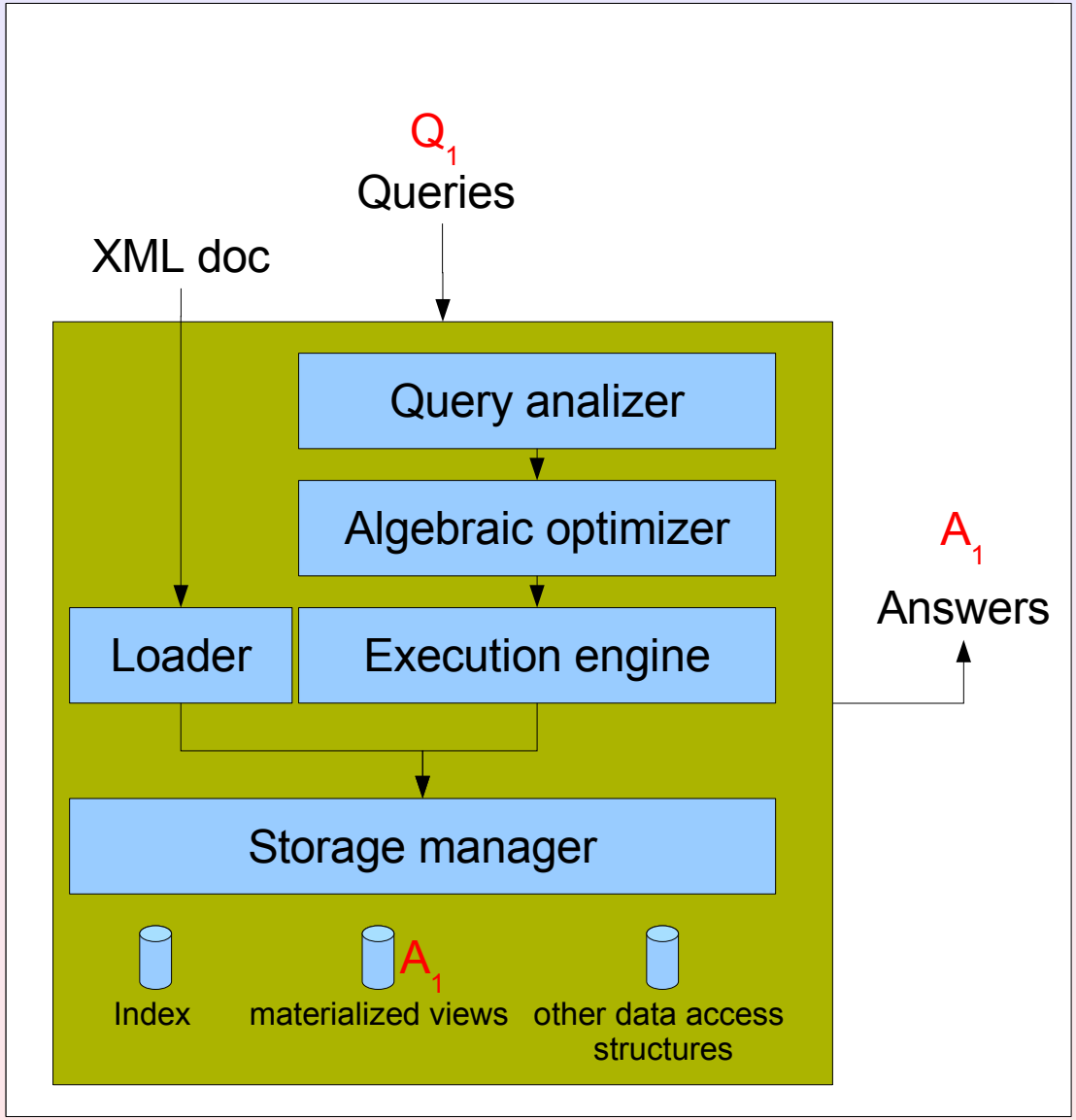
Context: rewriting XQuery using nested views



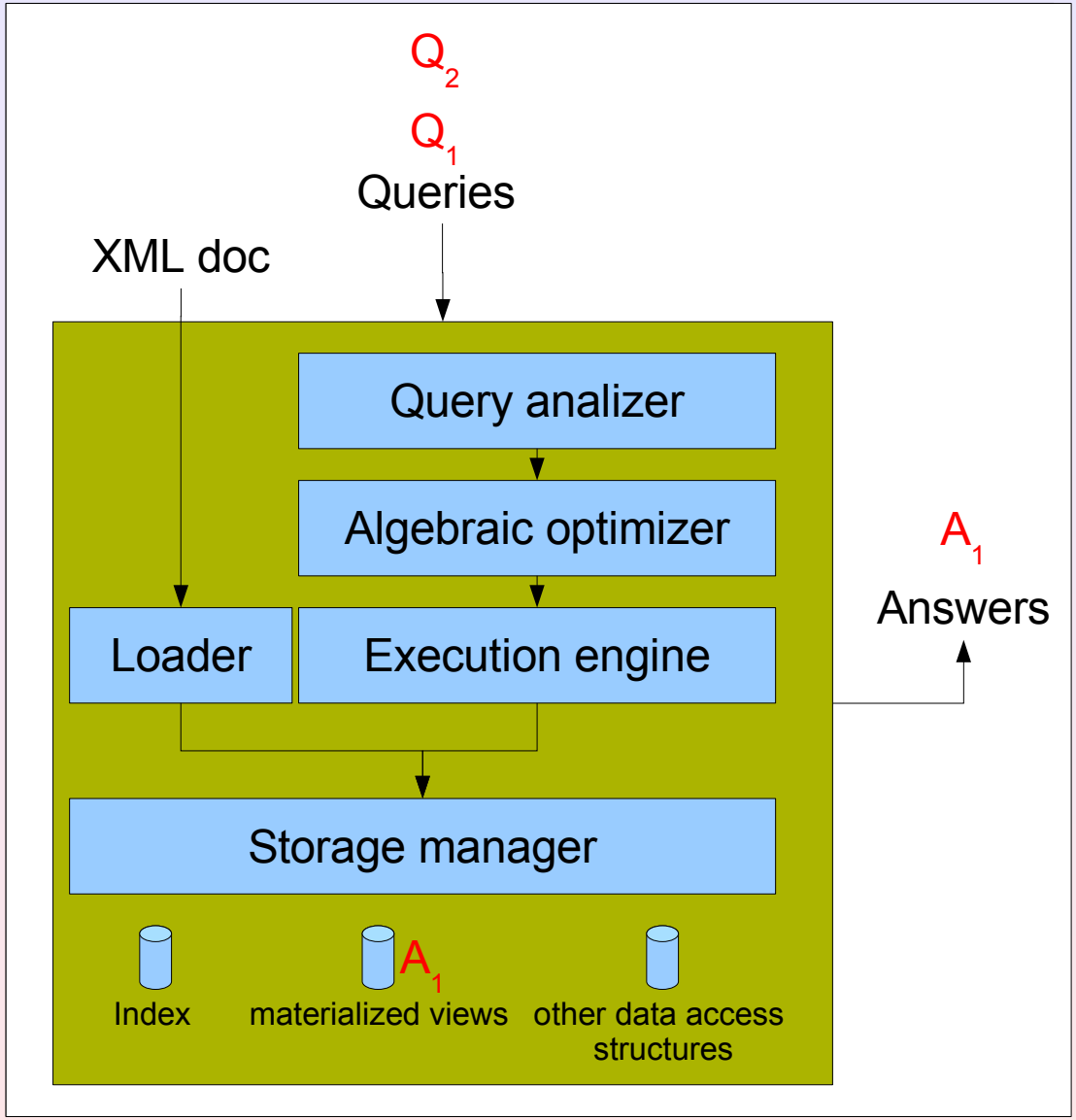
Context: rewriting XQuery using nested views



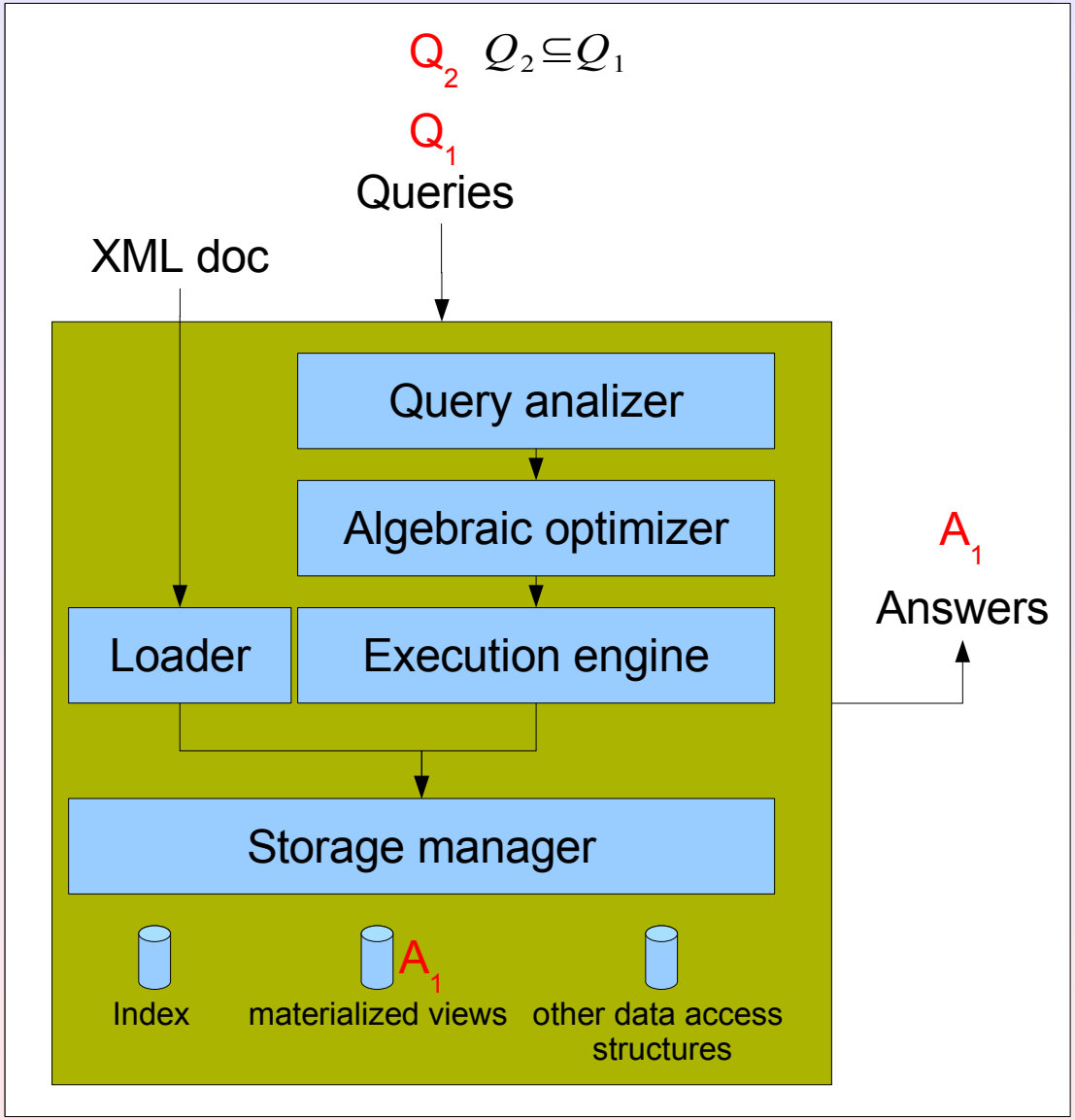
Context: rewriting XQuery using nested views



Context: rewriting XQuery using nested views



Context: rewriting XQuery using nested views



Related works

XPath containment and equivalence

- No constraints:
  - S. Amer-Yahia 2001, Deutsch and Tannen, 2001; Miklau and Suciu, 2002
- DTD constraints:
  - Wood 2003 ,Neven and Schwentick 2003

XPath rewriting

- Balmin et al, 2004, weak path usage
- Wanhong Xu et al 2005
- Lakshmanan et al, 2006, MCR under path summary constraints

XQuery containment and rewriting

- Halevy et al. 2004
- Onose et al. 2006, equivalent rewriting

We address XAM containment and rewriting under path summaries constraints.

Related works

XPath containment and equivalence

- No constraints:
  - S. Amer-Yahia 2001, Deutsch and Tannen, 2001; Miklau and Suciu, 2002
- DTD constraints:
  - Wood 2003 ,Neven and Schwentick 2003

XPath rewriting

- Balmin et al, 2004, weak path usage
- Wanhong Xu et al 2005
- Lakshmanan et al, 2006, MCR under path summary constraints

XQuery containment and rewriting

- Halevy et al. 2004
- Onose et al. 2006, equivalent rewriting

We address XAM containment and rewriting under path summaries constraints.

Materialized views for XML: Query

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- Use an XQuery view that materialize the whole query.
- Problems:
  - The *complexity* of the XQuery language
  - Difficult to *understand and combine* multiple XQuery views

Materialized views for XML: Query

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- Use an XQuery view that materialize the whole query.
- Problems:
  - The *complexity* of the XQuery language
  - Difficult to *understand and combine* multiple XQuery views

Materialized views for XML: Query

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- Use an XQuery view that materialize the whole query.
- Problems:
  - The *complexity* of the XQuery language
  - Difficult to *understand and combine* multiple XQuery views

Materialized views for XML: XPath

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- XPath: V=//item
  - Q=$E_1(V)$
  - Problems: V is big, E_1 is complex
- XPath: V1=//item/name, V2=//item//keyword
  - $Q = E_2(V_1, V_2)$
  - XQuery semantics: output an empty res element even if no *name*, *keyword*!
  - Problem: How to combine V_1 and V_2? Answer: ID!

Materialized views for XML: XPath

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- XPath: V=//item
  - Q=$E_1(V)$
  - Problems: V is big, E_1 is complex
- XPath: V1=//item/name, V2=//item//keyword
  - $Q = E_2(V_1, V_2)$
  - XQuery semantics: output an empty res element even if no *name*, *keyword*!
  - Problem: How to combine V_1 and V_2?

Materialized views for XML: XPath

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- XPath: V=//item
  - Q=$E_1(V)$
  - Problems: V is big, E_1 is complex
- XPath: V_1=//item/name, V_2=//item//keyword
  - $Q = E_2(V_1, V_2)$
  - XQuery semantics: output an empty res element even if no *name*, *keyword*!
  - Problem: How to combine V_1 and V_2?

Materialized views for XML: XPath

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- XPath: V=//item
  - Q=$E_1(V)$
  - Problems: V is big, E_1 is complex
- XPath: V_1=//item/name, V_2=//item//keyword
  - $Q = E_2(V_1, V_2)$
  - XQuery semantics: output an empty res element even if no *name*, *keyword*!
  - Problem: How to combine V_1 and V_2?

Materialized views for XML: XPath

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- XPath: V=//item
  - Q=$E_1(V)$
  - Problems: V is big, E_1 is complex
- XPath: V_1=//item/name, V_2=//item//keyword
  - $Q = E_2(V_1, V_2)$
  - XQuery semantics: output an empty res element even if no *name*, *keyword*!
  - Problem: How to combine V_1 and V_2?

Materialized views for XML: XPath

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- XPath: V=//item
  - Q=$E_1(V)$
  - Problems: V is big, E_1 is complex
- XPath: V_1=//item/name, V_2=//item//keyword
  - $Q = E_2(V_1, V_2)$
  - XQuery semantics: output an empty res element even if no *name*, *keyword*!
  - Problem: How to combine V_1 and V_2?

Materialized views for XML: XPath

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- XPath: V=//item
  - Q=$E_1(V)$
  - Problems: V is big, E_1 is complex
- XPath: V_1=//item/name, V_2=//item//keyword
  - $Q = E_2(V_1, V_2)$
  - XQuery semantics: output an empty res element even if no *name*, *keyword*!
  - Problem: How to combine V_1 and V_2? Add some IDs!

Materialized views for XML: XPath

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- XPath: V=//item
  - Q=$E_1(V)$
  - Problems: V is big, E_1 is complex
- XPath: V_1=//item/name, V_2=//item//keyword
  - $Q = E_2(V_1, V_2)$
  - XQuery semantics: output an empty res element even if no *name*, *keyword*!
  - Problem: How to combine V_1 and V_2? Add some IDs!

Materialized views for XML: XPath

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- XPath: V=//item
  - Q=$E_1(V)$
  - Problems: V is big, E_1 is complex
- XPath: V_1=//item/name, V_2=//item//keyword
  - $Q = E_2(V_1, V_2)$
  - XQuery semantics: output an empty res element even if no *name*, *keyword*!
  - Problem: How to combine V_1 and V_2? Add some IDs!

Materialized views for XML: Enhanced Tree Patterns

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- Store IDs in the view V_1=//item ID, V_2=//name ID, V_3=//keyword ID
  - $Q = V_1 \bowtie\!\!\sqsubset_{ID} V_2 \bowtie\!\!\sqsubset_{ID} V_3$
  - Problem: still store too much!
- ID, Val, Cont in the view definition
  - $Q = V_1 \bowtie\!\!\sqsubset_{ID} V_2 \bowtie\!\!\sqsubset_{ID} V_3$
  - Can I eliminate the joins? Express outer joins algebraically

Materialized views for XML: Enhanced Tree Patterns

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- Store IDs in the view V_1=//item ID, V_2=//name ID, V_3=//keyword ID
  - $Q = V_1 \bowtie\!\sqsubset_{ID} V_2 \bowtie\!\sqsubset_{ID} V_3$
  - Problem: still store too much!
- ID, Val, Cont in the view definition V_1=//item ID, V_2=//name ID Val, V_3=//keyword ID Cont
  - $Q = V_1 \bowtie\!\sqsubset_{ID} V_2 \bowtie\!\sqsubset_{ID} V_3$
  - Can I eliminate the joins? Bypass auto nested edges

Materialized views for XML: Enhanced Tree Patterns

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- Store IDs in the view V_1=//item ID, V_2=//name ID, V_3=//keyword ID
  - $Q = V_1 \bowtie\!\sqsubset_{ID} V_2 \bowtie\!\sqsubset_{ID} V_3$
  - Problem: still store too much!
- ID, Val, Cont in the view definition V_1=//item ID, V_2=//name ID Val, V_3=//keyword ID Cont
  - $Q = V_1 \bowtie\!\sqsubset_{ID} V_2 \bowtie\!\sqsubset_{ID} V_3$
  - Can I eliminate the joins? Express subtree navigation

Materialized views for XML: Enhanced Tree Patterns

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- Store IDs in the view V_1=//item ID, V_2=//name ID, V_3=//keyword ID
  - $Q = V_1 \bowtie\!\sqsubset_{ID} V_2 \bowtie\!\sqsubset_{ID} V_3$
  - Problem: still store too much!
- ID, Val, Cont in the view definition V_1=//item ID, V_2=//name ID Val, V_3=//keyword ID Cont
  - $Q = V_1 \bowtie\!\sqsubset_{ID} V_2 \bowtie\!\sqsubset_{ID} V_3$
  - Can I eliminate the joins? Express outer joins in view!

Materialized views for XML: Enhanced Tree Patterns

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- Store IDs in the view V_1=//item ID, V_2=//name ID, V_3=//keyword ID
  - $Q = V_1 \bowtie\!\sqsubset_{ID} V_2 \bowtie\!\sqsubset_{ID} V_3$
  - Problem: still store too much!
- ID, Val, Cont in the view definition V_1=//item ID, V_2=//name ID Val, V_3=//keyword ID Cont
  - $Q = V_1 \bowtie\!\sqsubset_{ID} V_2 \bowtie\!\sqsubset_{ID} V_3$
  - Can I eliminate the joins? Express outer joins in view!

Materialized views for XML: Enhanced Tree Patterns

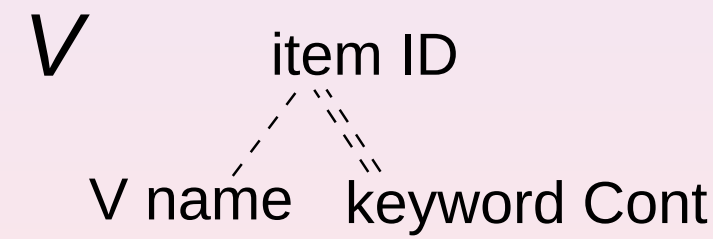
```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- Store IDs in the view V_1=//item ID, V_2=//name ID, V_3=//keyword ID
  - $Q = V_1 \bowtie\!\sqsubset_{ID} V_2 \bowtie\!\sqsubset_{ID} V_3$
  - Problem: still store too much!
- ID, Val, Cont in the view definition V_1=//item ID, V_2=//name ID Val, V_3=//keyword ID Cont
  - $Q = V_1 \bowtie\!\sqsubset_{ID} V_2 \bowtie\!\sqsubset_{ID} V_3$
  - Can I eliminate the joins? Express outer joins in view!

Materialized views for XML: Enhanced Tree Patterns

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- Store ID, Cont, Val + optional edges

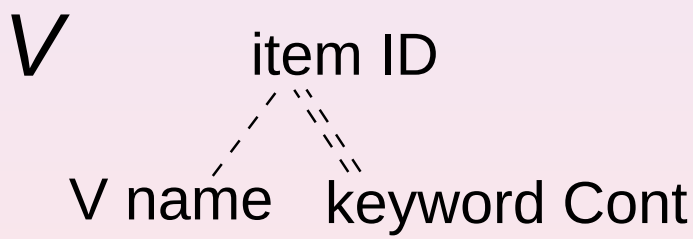


- $Q = GroupBy_{ID}(V)$
- Can I eliminate the need for group by? Annotations

Materialized views for XML: Enhanced Tree Patterns

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- Store ID, Cont, Val + optional edges

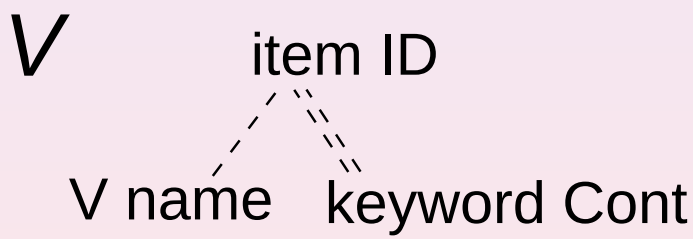


  - $Q = GroupBy_{ID}(V)$
  - Can I eliminate the need for group by? Add nesting!

Materialized views for XML: Enhanced Tree Patterns

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- Store ID, Cont, Val + optional edges

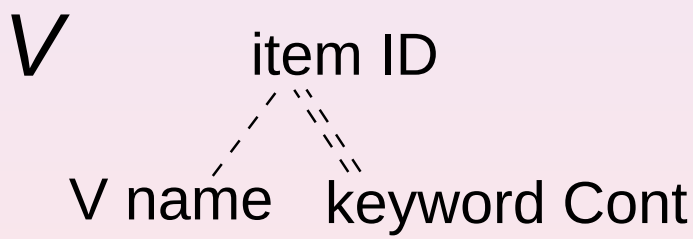


  - $Q = GroupBy_{ID}(V)$
  - Can I eliminate the need for group by? Add nesting!

Materialized views for XML: Enhanced Tree Patterns

```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- Store ID, Cont, Val + optional edges

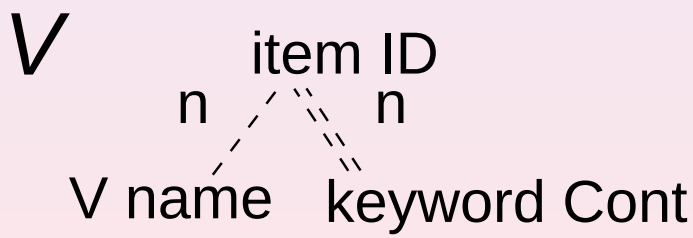


  - $Q = GroupBy_{ID}(V)$
  - Can I eliminate the need for group by? Add nesting!

Materialized views for XML: XML Access Modules

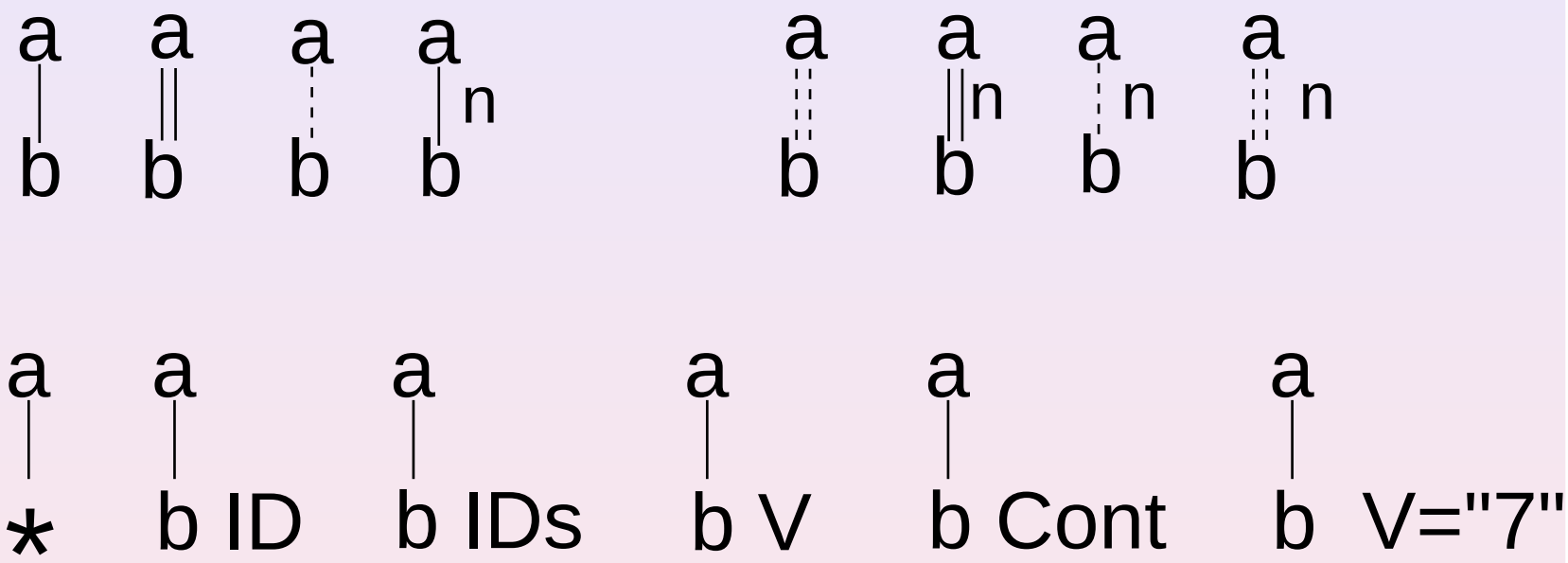
```
for $x in //item return
    <res>
        {$x/name/text()}
        {$x//keyword}
    </res>
```

- XML Access Modules = tree patterns with IDs, Val, Cont + optional and nested edges + value predicates

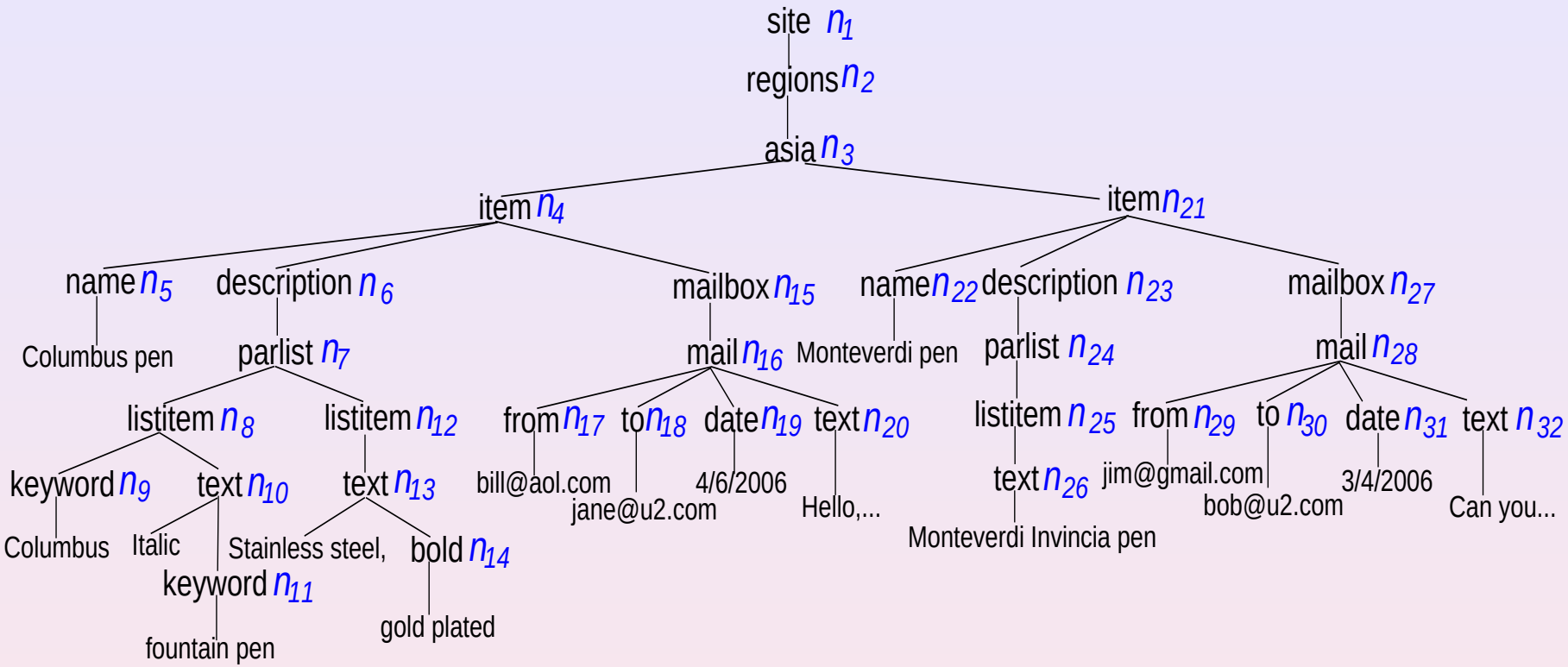


- $Q = V$

XAMs by examples



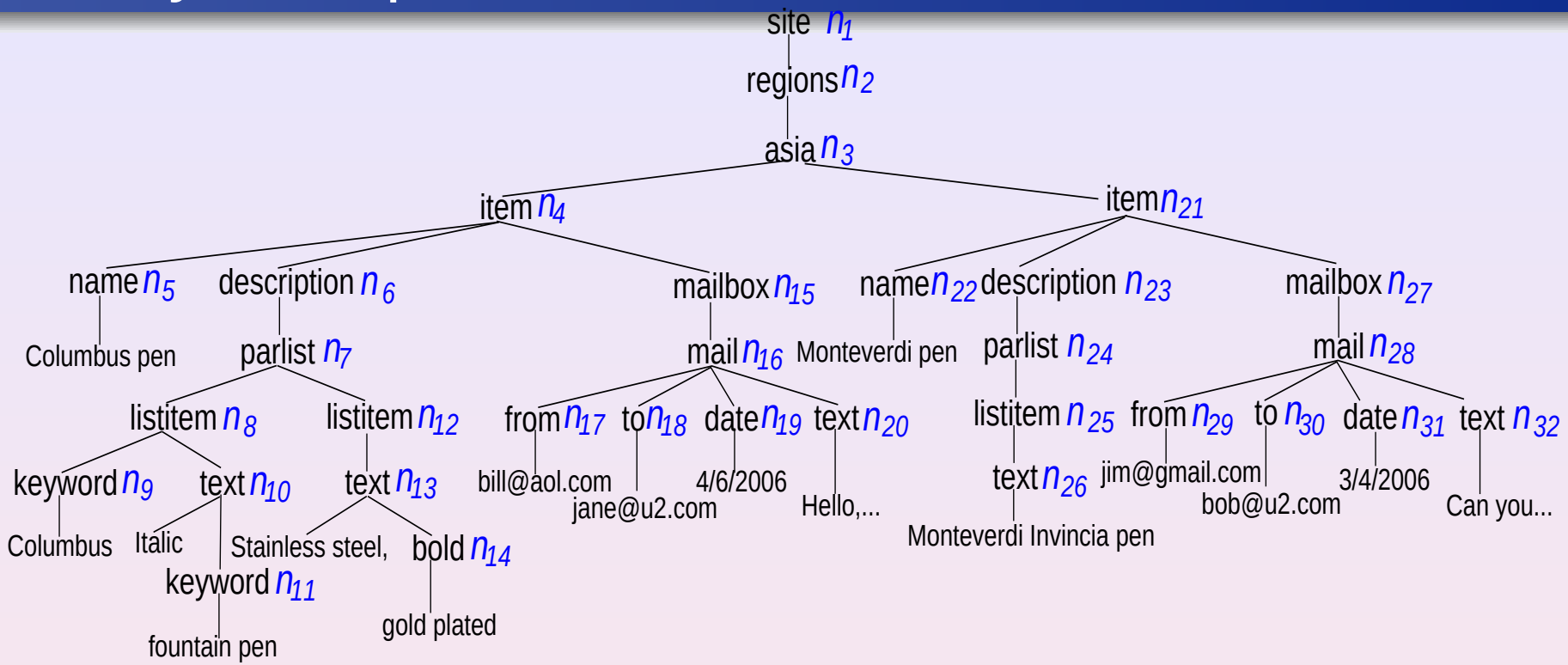
XAMs by examples

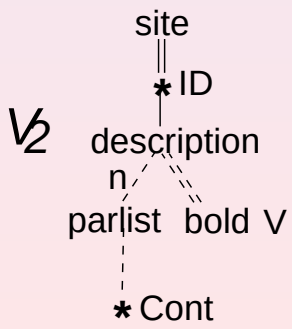


V_1

ID	V
n_5	Columbus pen
n_{22}	Monteverdi pen

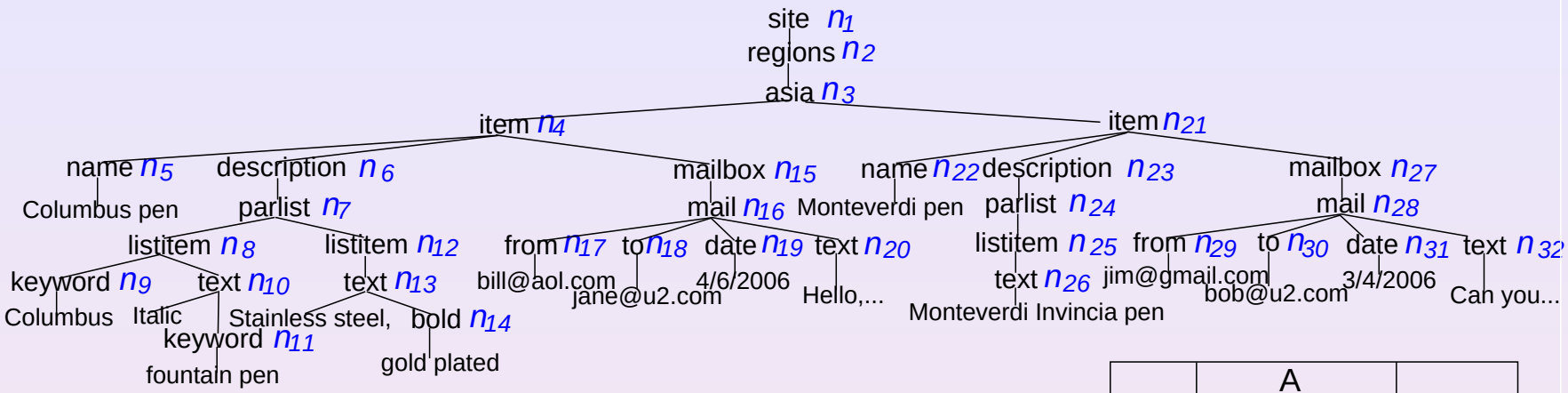
XAMs by examples





ID	A C	V
n_4	<listitem><keyword>Columbus</keyword><text>Italic <keyword>fountain pen</keyword></text> </listitem>	gold plated
	<listitem><text>Stainless steeel, <bold>gold plated</bold></text> </listitem>	
n_{21}	<listitem><text>Monteverdi Invincia pen</text></listitem>	$\perp$

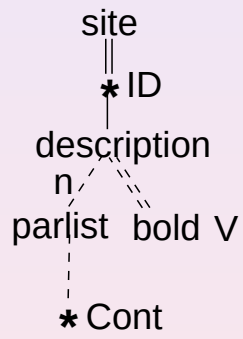
Rewriting XQuery using XAMs



V_1

ID	V
n_5	Columbus pen
n_{22}	Monteverdi pen

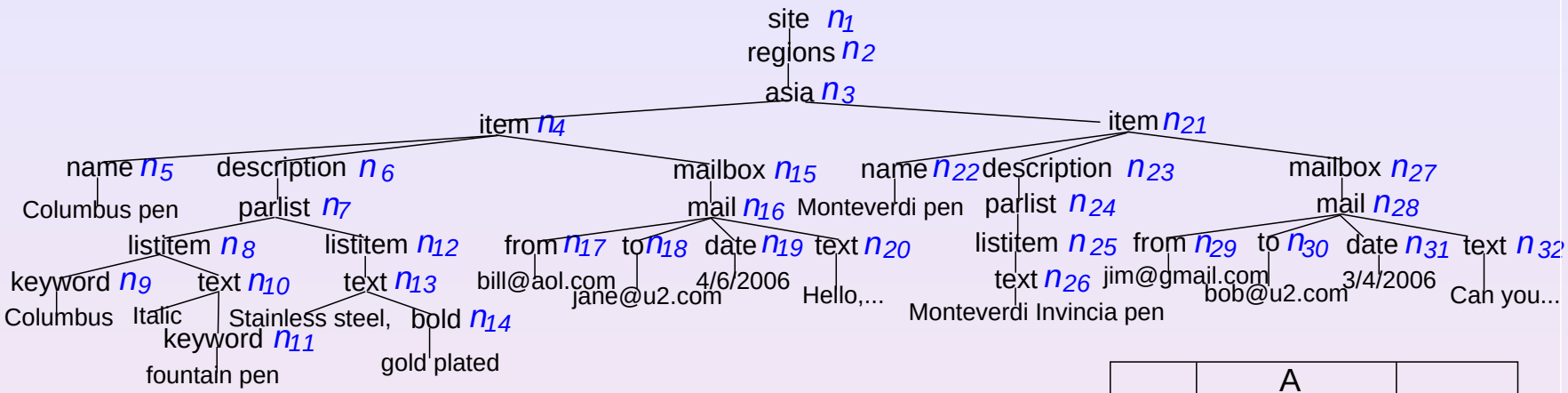
V_2



ID	A C	V
n_4	. . . . . .	gold plated
n_{21}	. . .	$\perp$

```
for $x in //item[//mail] return
    <res> {$x/name/text(),
        for $y in $x//listitem return
        <key> {$y//keyword} </key>} </res>
```

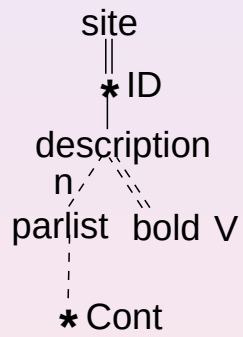
Rewriting XQuery using XAMs



V_1

ID	V
n_5	Columbus pen
n_{22}	Monteverdi pen

V_2



ID	A: C	V
n_4	. . . . . .	gold plated
n_{21}	. . .	$\perp$

```
for $x in //item[//mail] return
    <res> {$x/name/text(),
        for $y in $x//listitem return
        <key> {$y//keyword} </key>} </res>
```

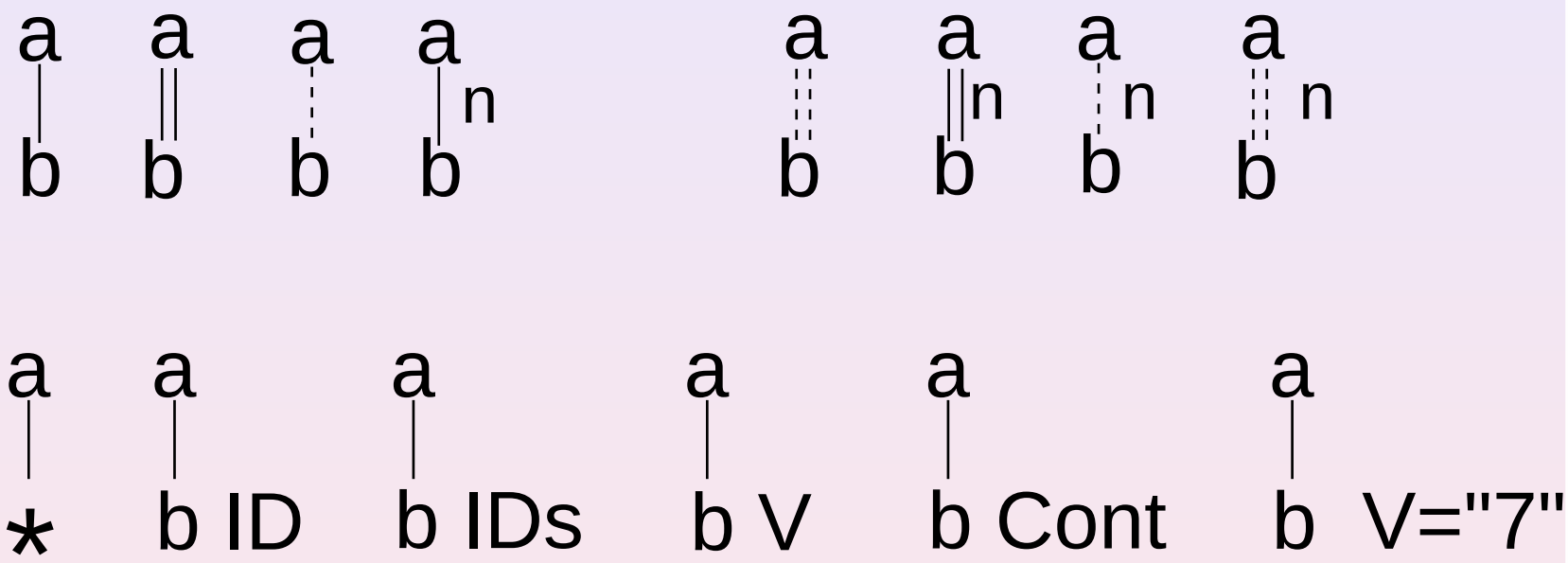
If document structure is known

$V_2 \bowtie V_1$

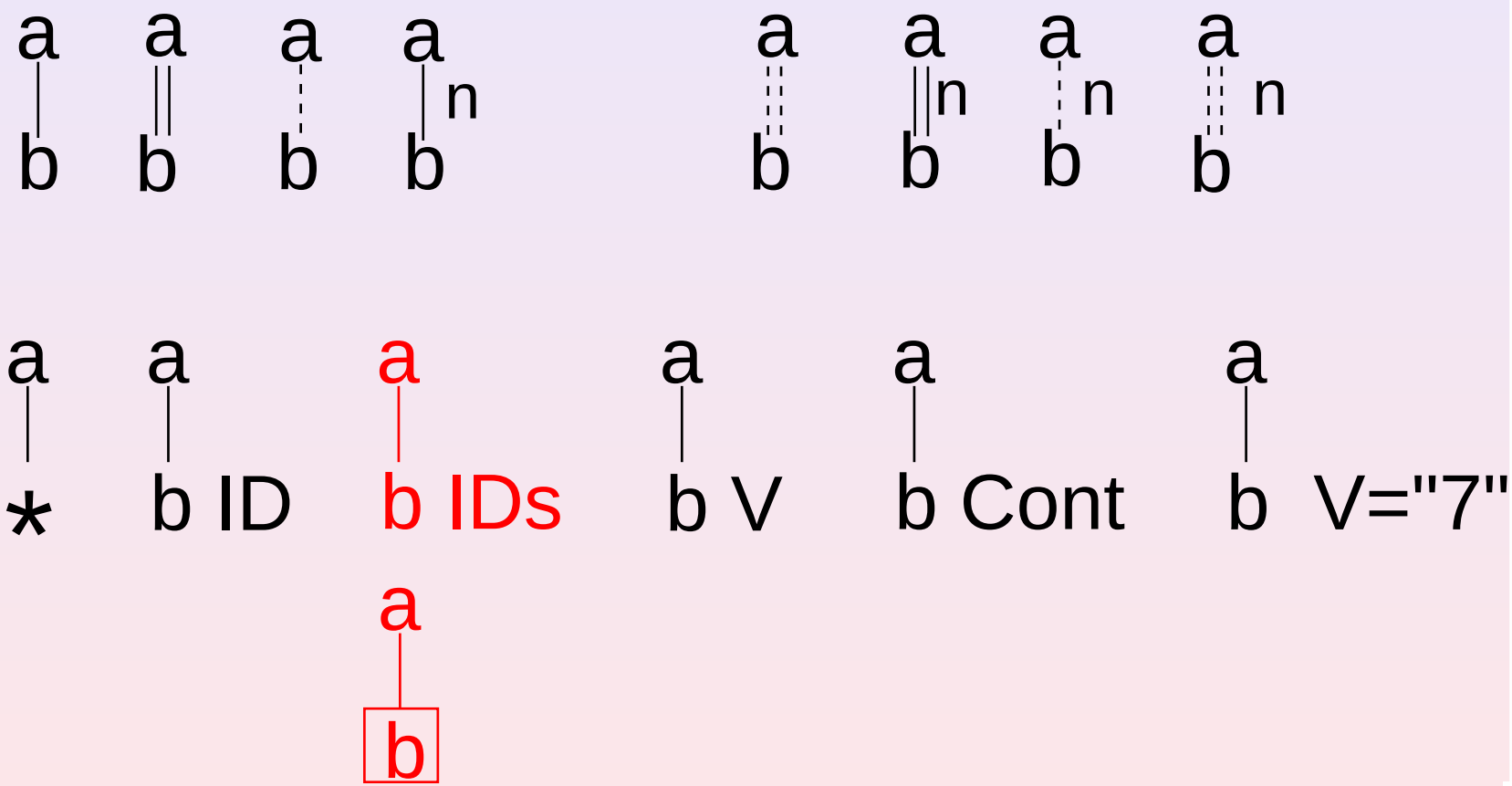
item ANCESTOR−OF name

ID	A: C	V	ID	V

XAMs by example



XAMs by example



Outline

1 Context: rewriting XQuery using nested views

2 Pattern containment under summary constraints
- Path summary
- Summary based containment

3 Summary based query rewriting

4 Experimental results

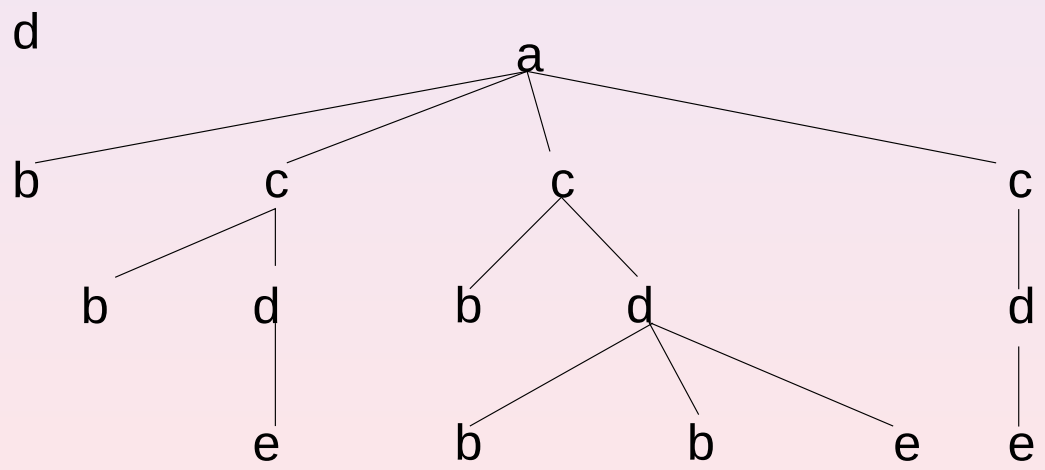
5 Conclusions

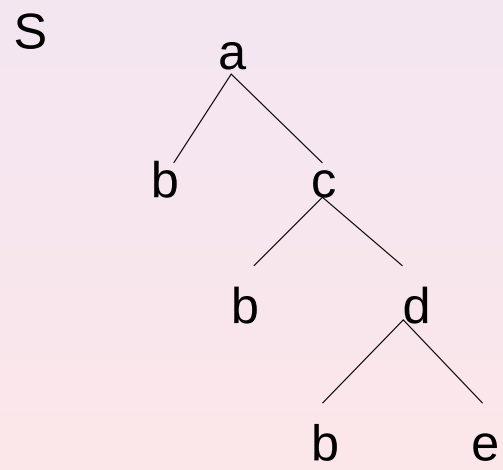
Path summary

Simple summary of a document d:

- root, label and parent-preserving mapping $\phi : d \rightarrow S$
- the children of a summary node have distinct labels

A document d' conforms to a path summary S ($S \models d'$) iff $\exists \phi' : d' \rightarrow S$



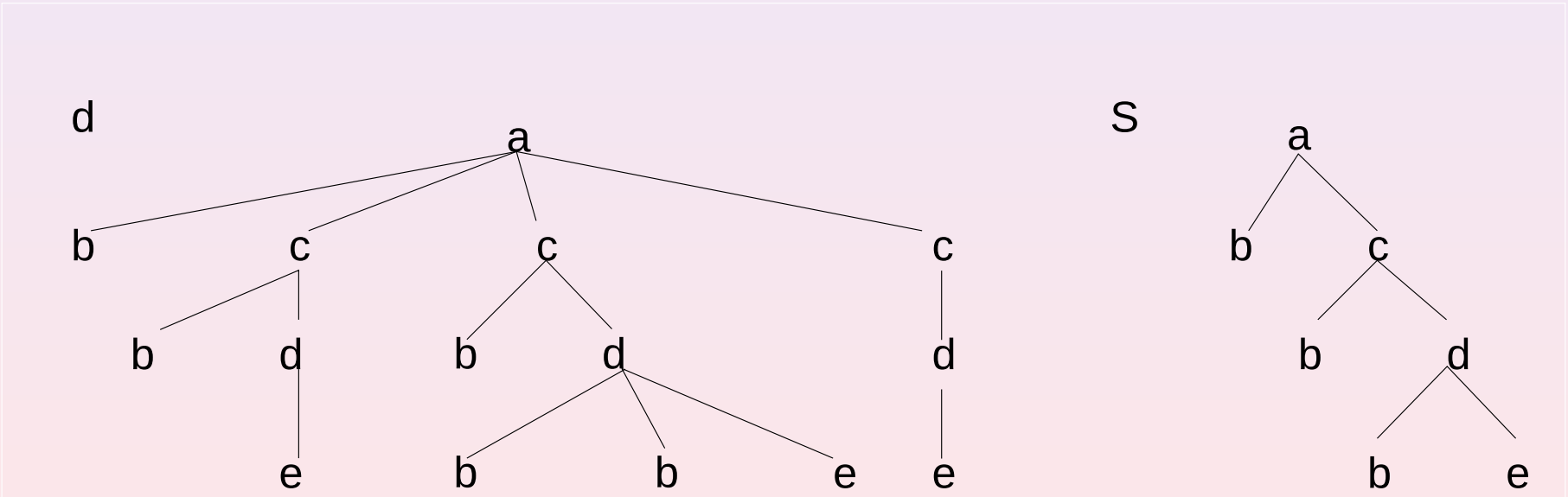


Path summary

Simple summary of a document d:

- root, label and parent-preserving mapping $\phi : d \rightarrow S$
- the children of a summary node have distinct labels

A document d' conforms to a path summary S ($S \models d'$) iff $\exists \phi' : d' \rightarrow S$

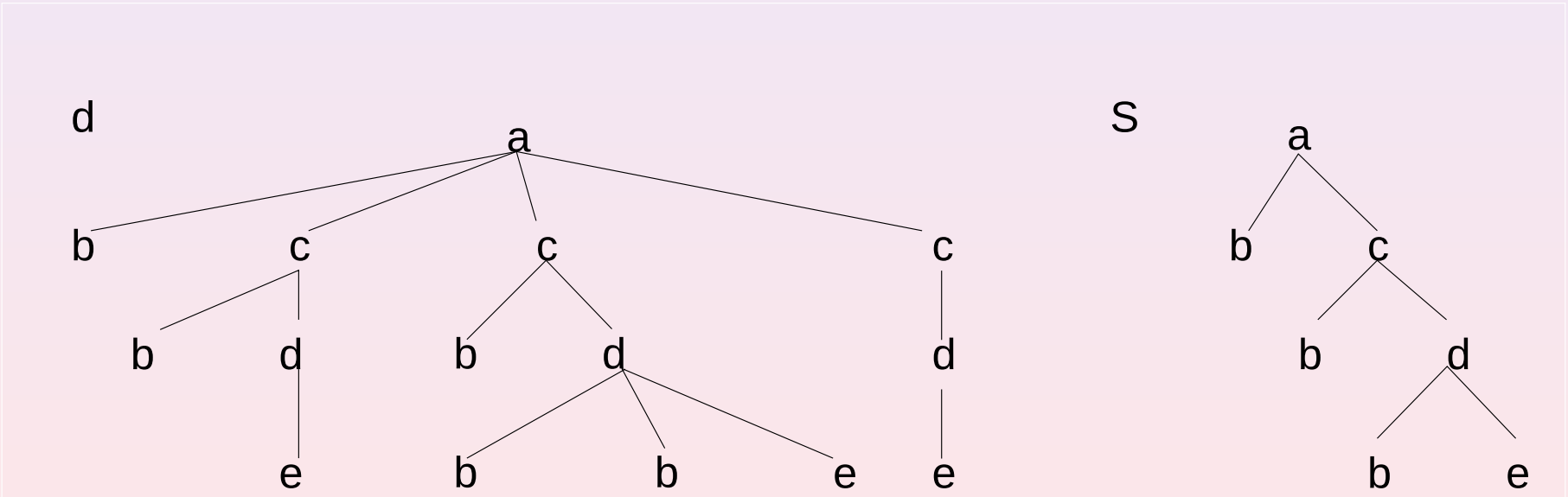


Path summary

Simple summary of a document d:

- root, label and parent-preserving mapping $\phi : d \rightarrow S$
- the children of a summary node have distinct labels

A document d' conforms to a path summary S ($S \models d'$) iff $\exists \phi' : d' \rightarrow S$

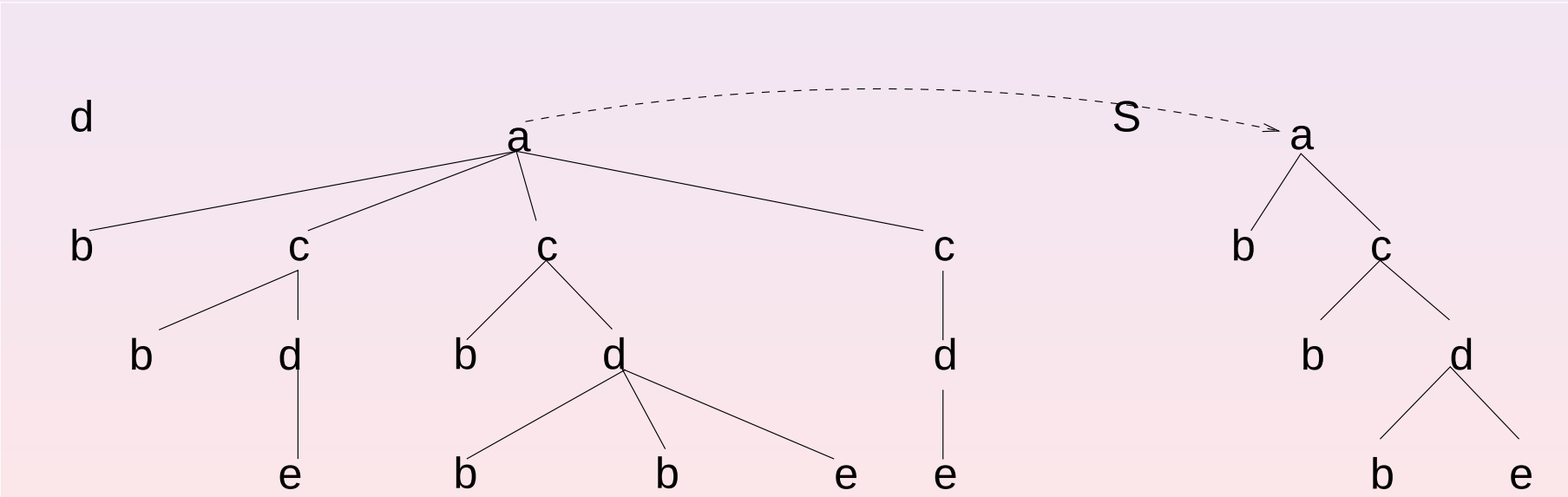


Path summary

Simple summary of a document d:

- root, label and parent-preserving mapping $\phi : d \rightarrow S$
- the children of a summary node have distinct labels

A document d' conforms to a path summary S ($S \models d'$) iff $\exists \phi' : d' \rightarrow S$

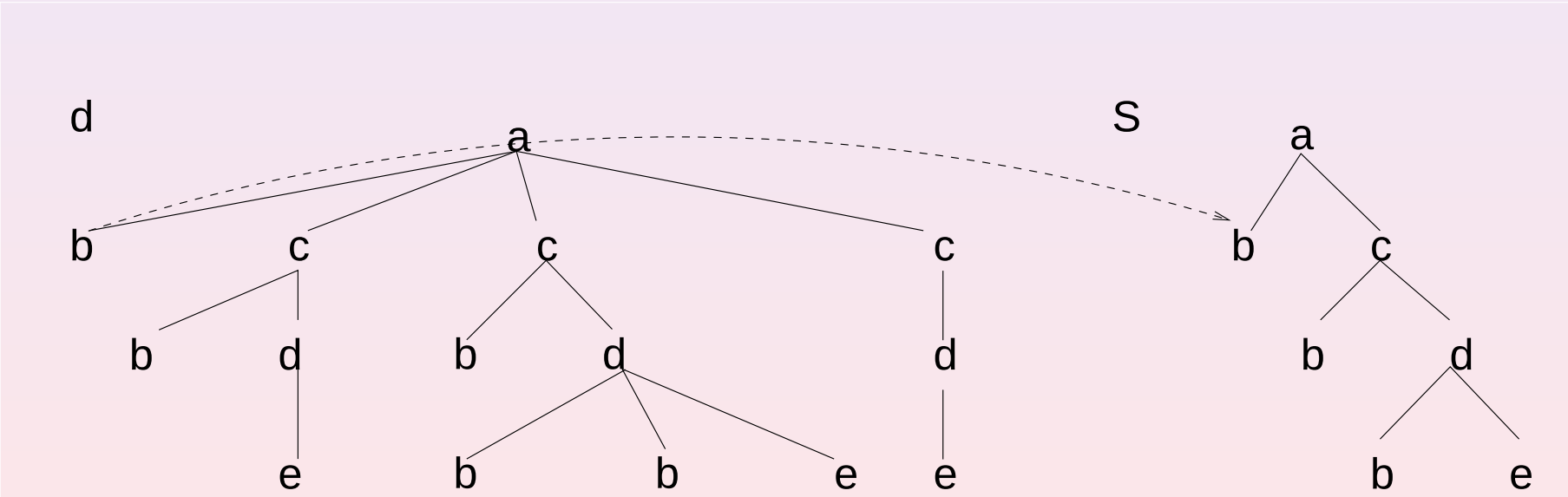


Path summary

Simple summary of a document d:

- root, label and parent-preserving mapping $\phi : d \rightarrow S$
- the children of a summary node have distinct labels

A document d' conforms to a path summary S ($S \models d'$) iff $\exists \phi' : d' \rightarrow S$

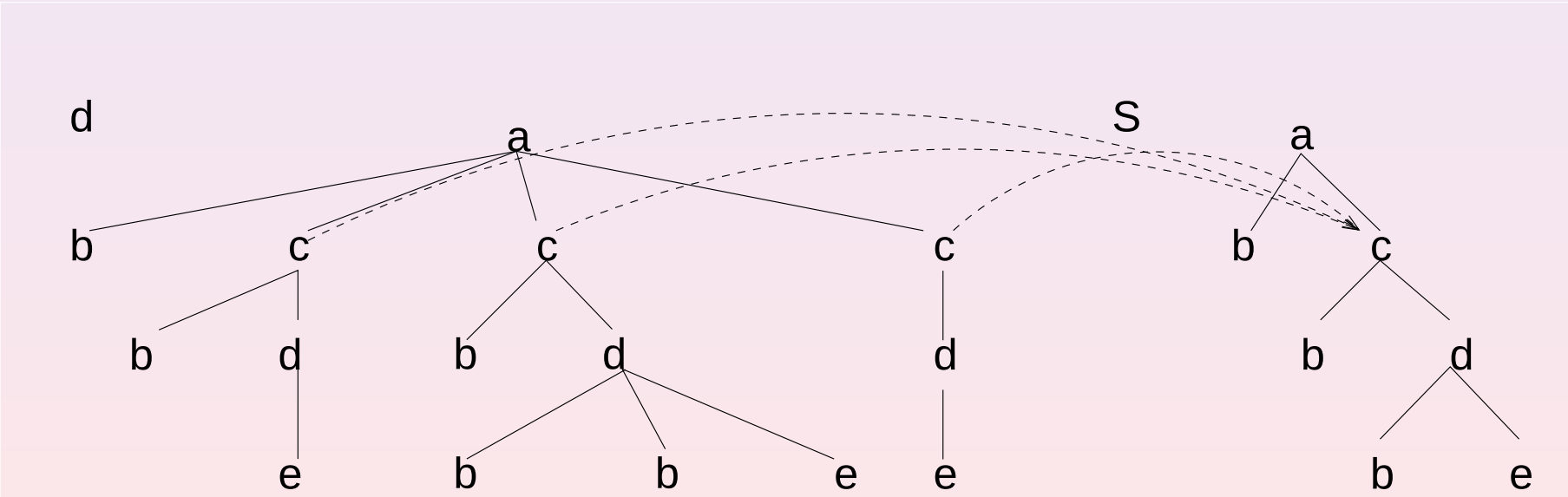


Path summary

Simple summary of a document d:

- root, label and parent-preserving mapping $\phi : d \rightarrow S$
- the children of a summary node have distinct labels

A document d' conforms to a path summary S ($S \models d'$) iff $\exists \phi' : d' \rightarrow S$

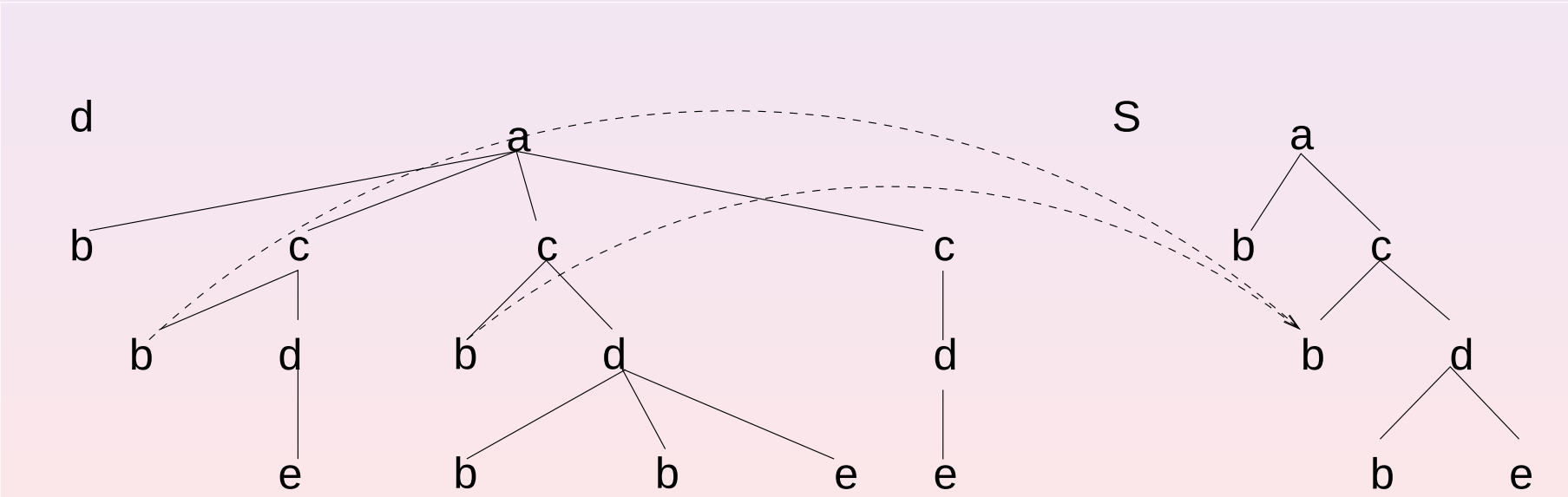


Path summary

Simple summary of a document d:

- root, label and parent-preserving mapping $\phi : d \rightarrow S$
- the children of a summary node have distinct labels

A document d' conforms to a path summary S ($S \models d'$) iff $\exists \phi' : d' \rightarrow S$

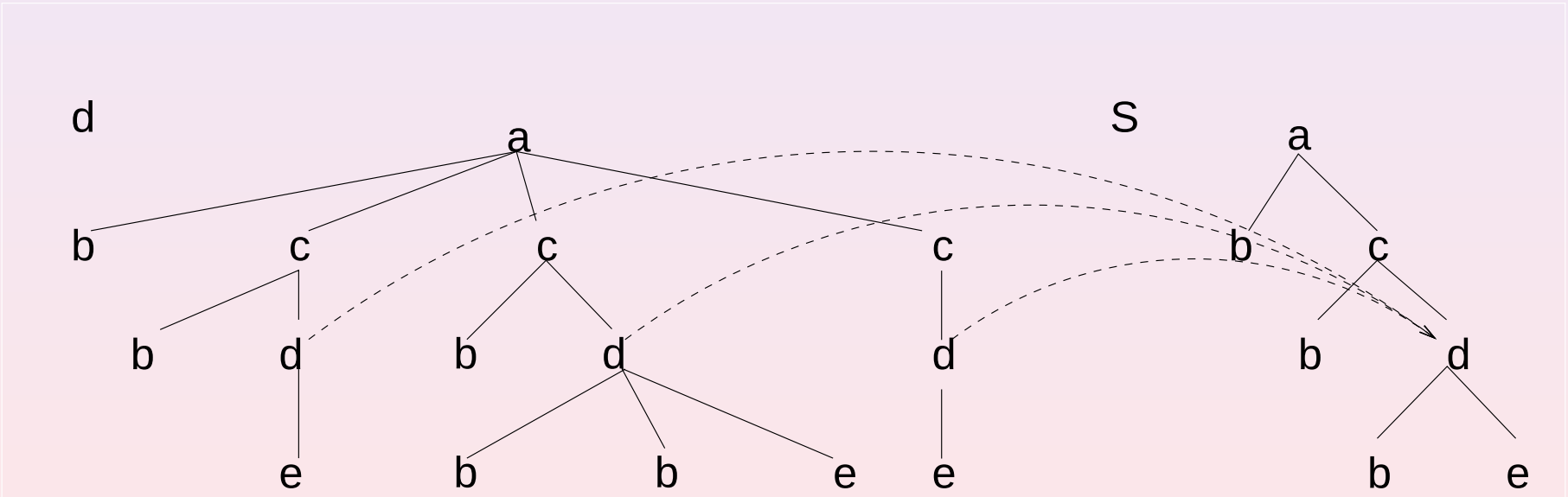


Path summary

Simple summary of a document d:

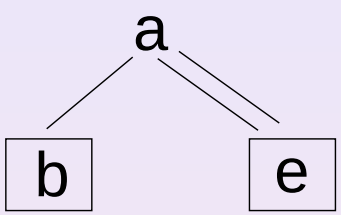
- root, label and parent-preserving mapping $\phi : d \rightarrow S$
- the children of a summary node have distinct labels

A document d' conforms to a path summary S ($S \models d'$) iff $\exists \phi' : d' \rightarrow S$

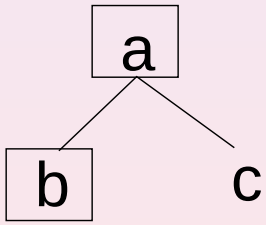


Path summaries and XAMs in query rewriting

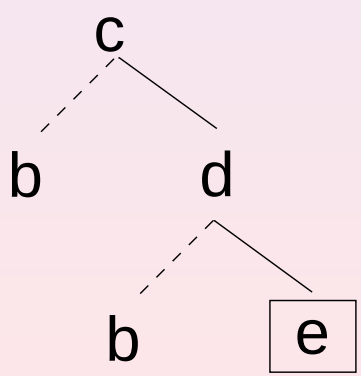
Q



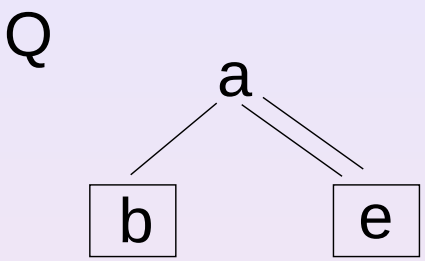
V1

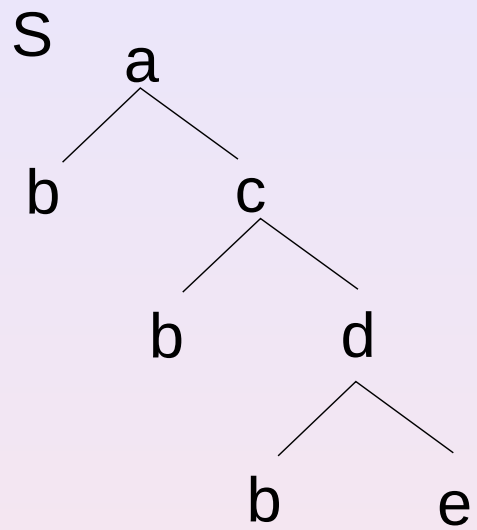


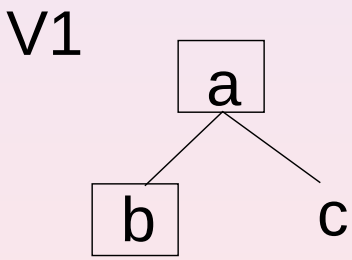
V2

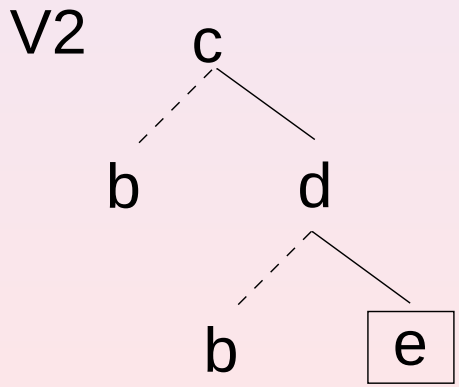


Path summaries and XAMs in query rewriting

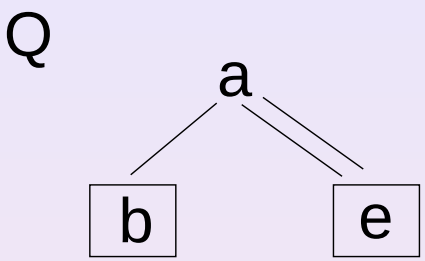


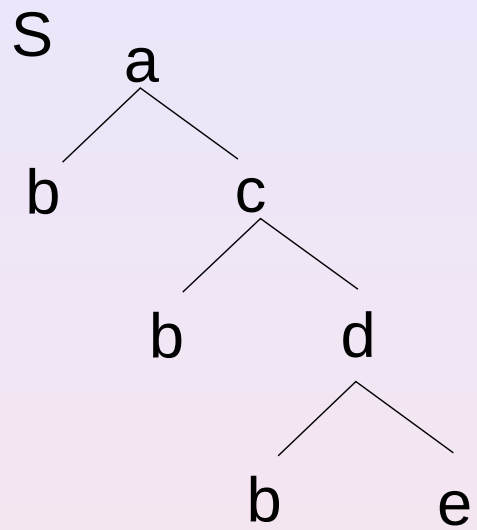


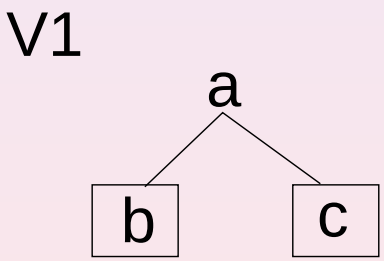


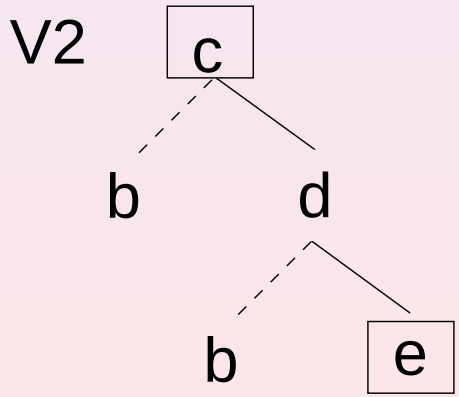


Path summaries and XAMs in query rewriting









Summary based containment

Definition

$p \subseteq_S p'$ *iff* for any t such that $S \models t$, $p(t) \subseteq p'(t)$.

Problem: infinitely many trees t

Solution:

- consider trees from a finite set canonical model of S
- if no canonical tree is a counter-example then $p \subseteq_S p'$

Summary based containment

Definition

$p \subseteq_S p'$ *iff* for any t such that $S \models t$, $p(t) \subseteq p'(t)$.

Problem: infinitely many trees t

Solution:

- consider trees from a finite set canonical model of S
- if no canonical tree is a counter-example then $p \subseteq_S p'$

Summary based containment

Definition

$p \subseteq_S p'$ *iff* for any t such that $S \models t$, $p(t) \subseteq p'(t)$.

Problem: infinitely many trees t

Solution:

- consider trees from a finite set canonical model of S
- if no canonical tree is a counter-example then $p \subseteq_S p'$

Summary based containment

Definition

p is S-contained in p' *iff* for any t such that $S \models t$, $p(t) \subseteq p'(t)$.

Proposition: $p \subseteq_S p'$ iff

$\forall\, t_p \in mod_S(p)$ the return tuple of $p(t_p) \in p'(t_p)$.

Outline

1. Context: rewriting XQuery using nested views
2. Pattern containment under summary constraints
   - Path summary
   - Summary based containment
3. Summary based query rewriting
4. Experimental results
5. Conclusions

Query rewriting: problem statement

Input: Path summary S, XQuery Q, set of XAMs $X_1, X_2, \ldots, X_n$

Output: all minimal algebraic expressions $e(X_1, X_2, \ldots, X_n)$ (up to algebraic equivalence) s.t. $\forall\, d$ conforming to S

$$Q(d) = e(X_1, X_2, ..., X_n)(d)$$

Algebra: σ , Π, $\bowtie_{ID}$, $\bowtie\!\!\sqsubset_{ID}$, $\bowtie_{\prec}$, $\bowtie\!\!\sqsubset_{\prec}$ (variants: ancestor-descendent, nested joins), *Nest*, *Unnest*, *Nav* and $\cup$

Query rewriting: problem statement

Input: Path summary S, XQuery Q, set of XAMs $X_1, X_2, \ldots, X_n$
Output: all minimal algebraic expressions $e(X_1, X_2, \ldots, X_n)$ (up to algebraic equivalence) s.t. $\forall\, d$ conforming to S

$$Q(d) = e(X_1, X_2, ..., X_n)(d)$$

Algebra: σ , Π, $\bowtie_{ID}$, $\bowtie\!\!\sqsubset_{ID}$, $\bowtie_{\prec}$, $\bowtie\!\!\sqsubset_{\prec}$ (variants: ancestor-descendent, nested joins), *Nest*, *Unnest*, *Nav* and $\cup$

Query rewriting: problem statement

Input: Path summary S, XQuery Q, set of XAMs $X_1, X_2, \ldots, X_n$
Output: all minimal algebraic expressions $e(X_1, X_2, \ldots, X_n)$ (up to algebraic equivalence) s.t. $\forall\, d$ conforming to S

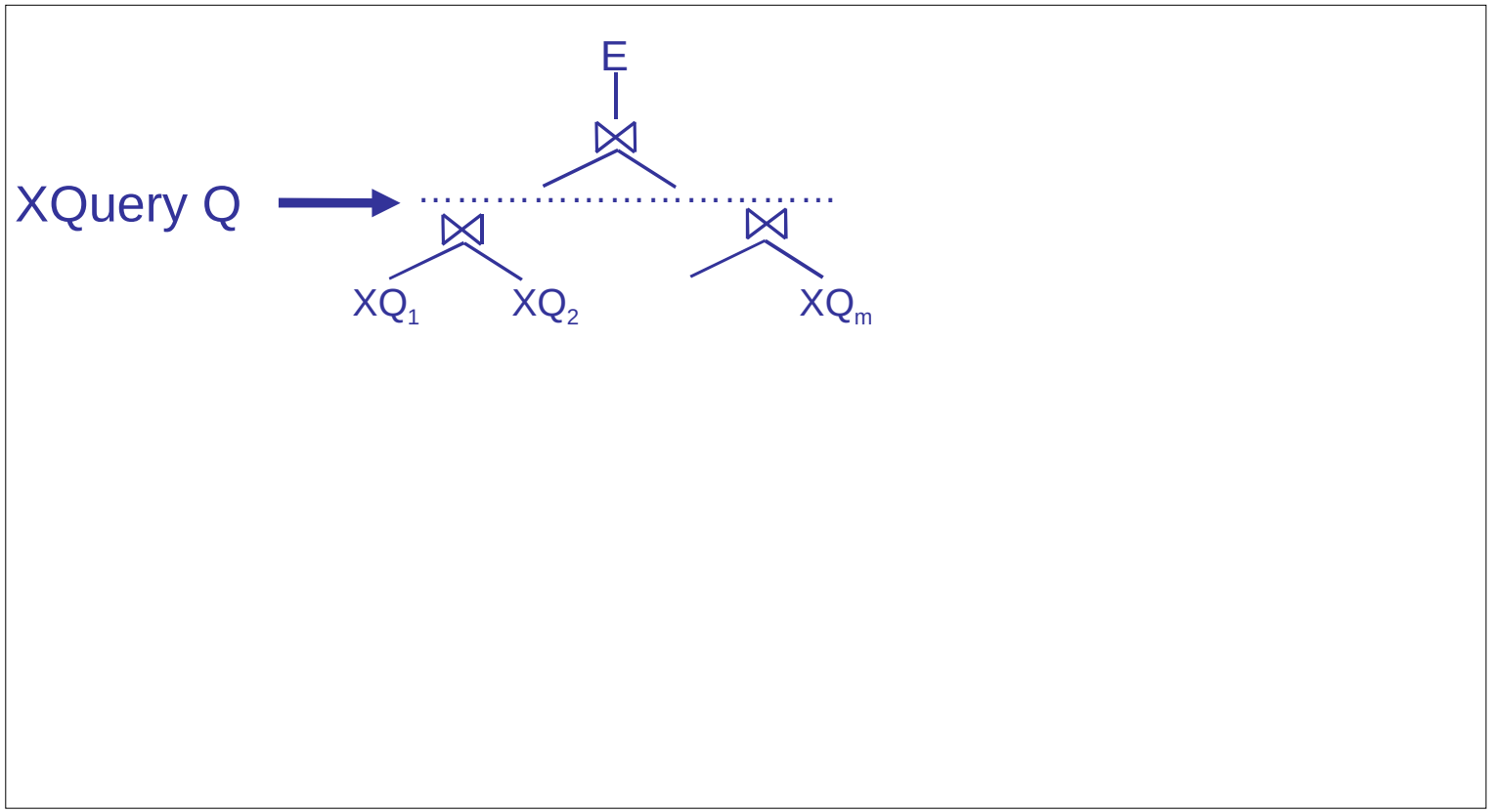
$$Q(d) = e(X_1, X_2, ..., X_n)(d)$$

Algebra: σ , Π, $\bowtie_{ID}$, $\bowtie\!\!\sqsubset_{ID}$, $\bowtie_{\prec}$, $\bowtie\!\!\sqsubset_{\prec}$ (variants: ancestor-descendent, nested joins), *Nest*, *Unnest*, *Nav* and $\cup$

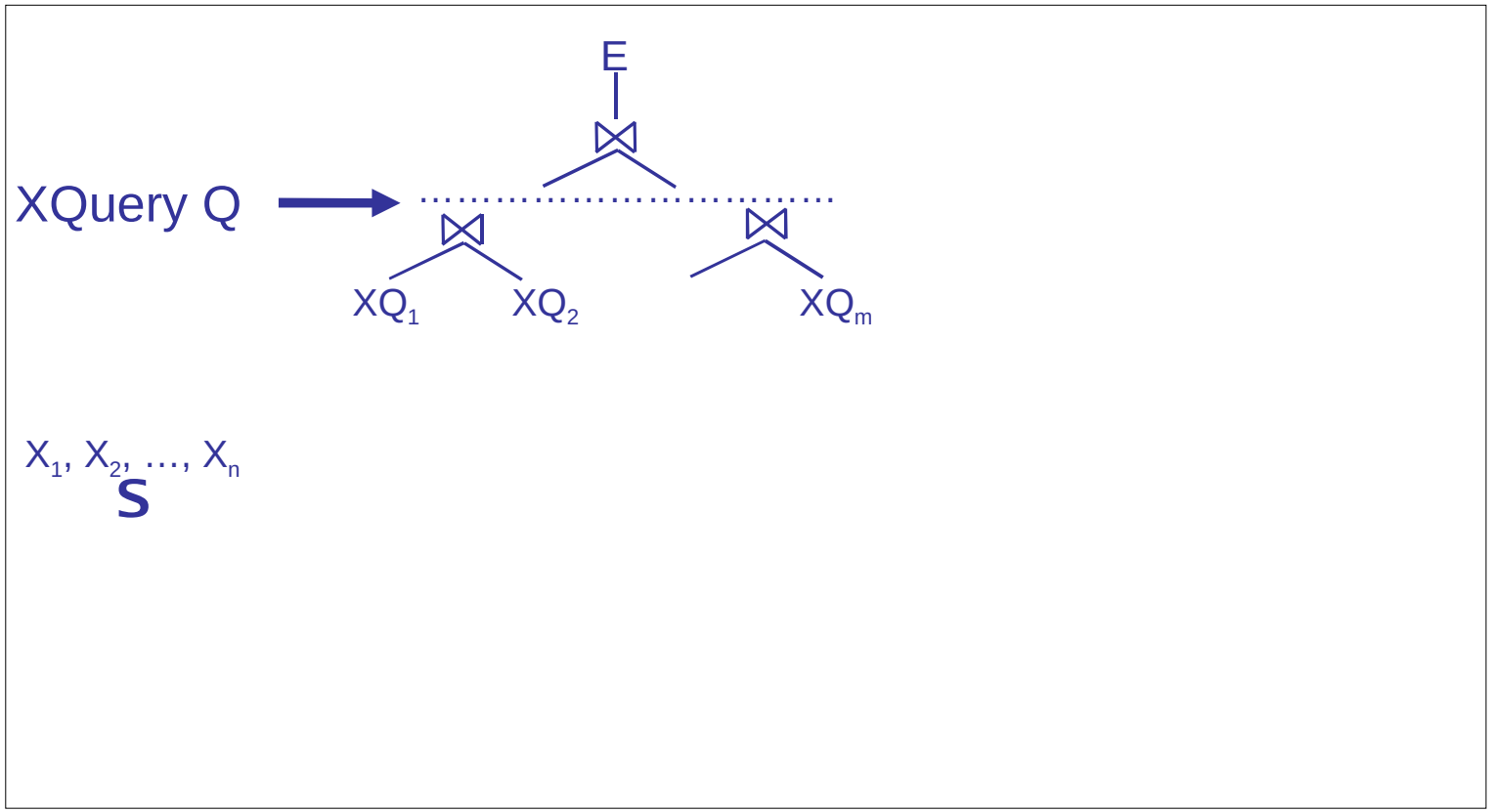
Query rewriting algorithm

XQuery Q

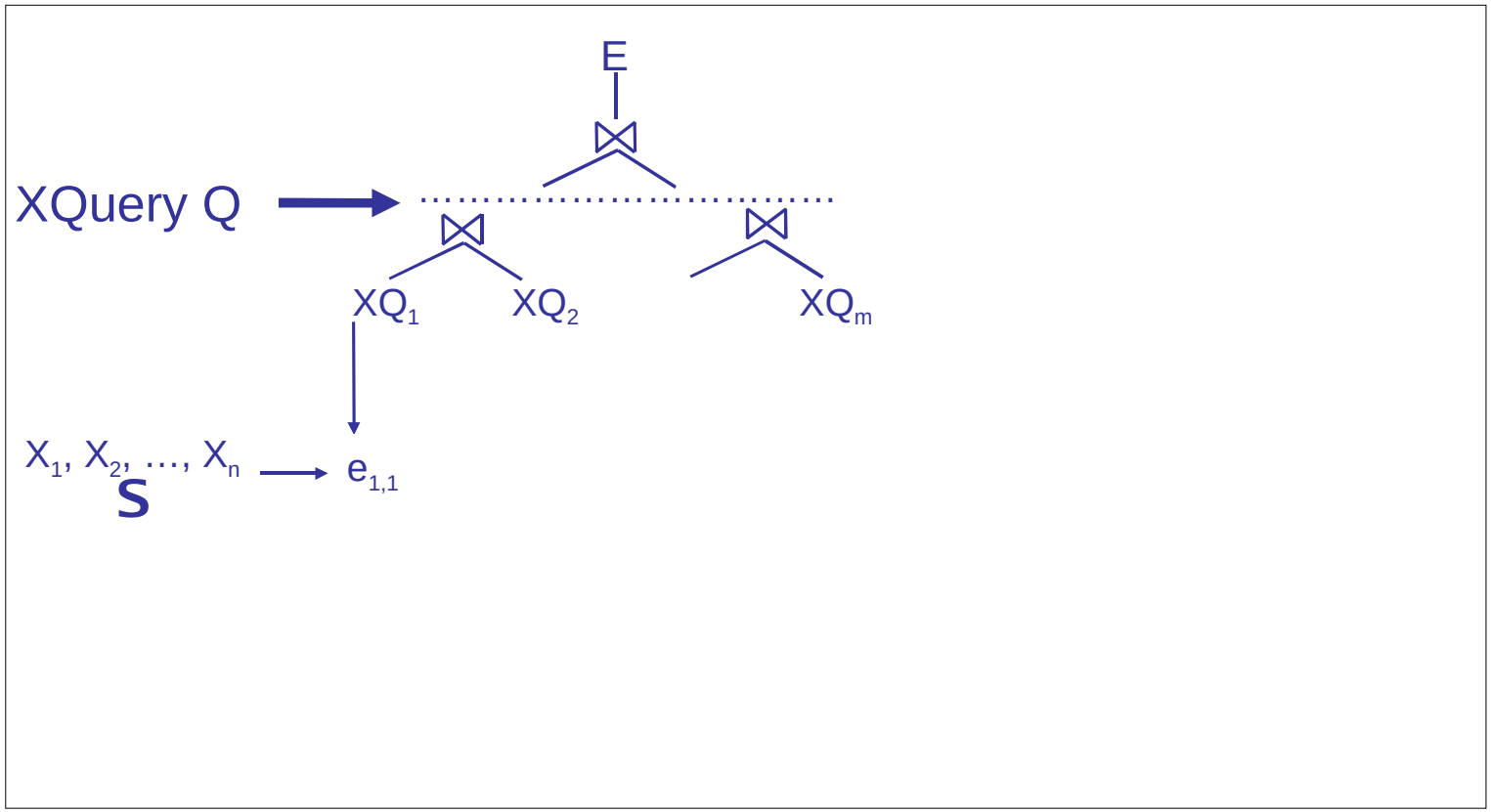
Query rewriting algorithm



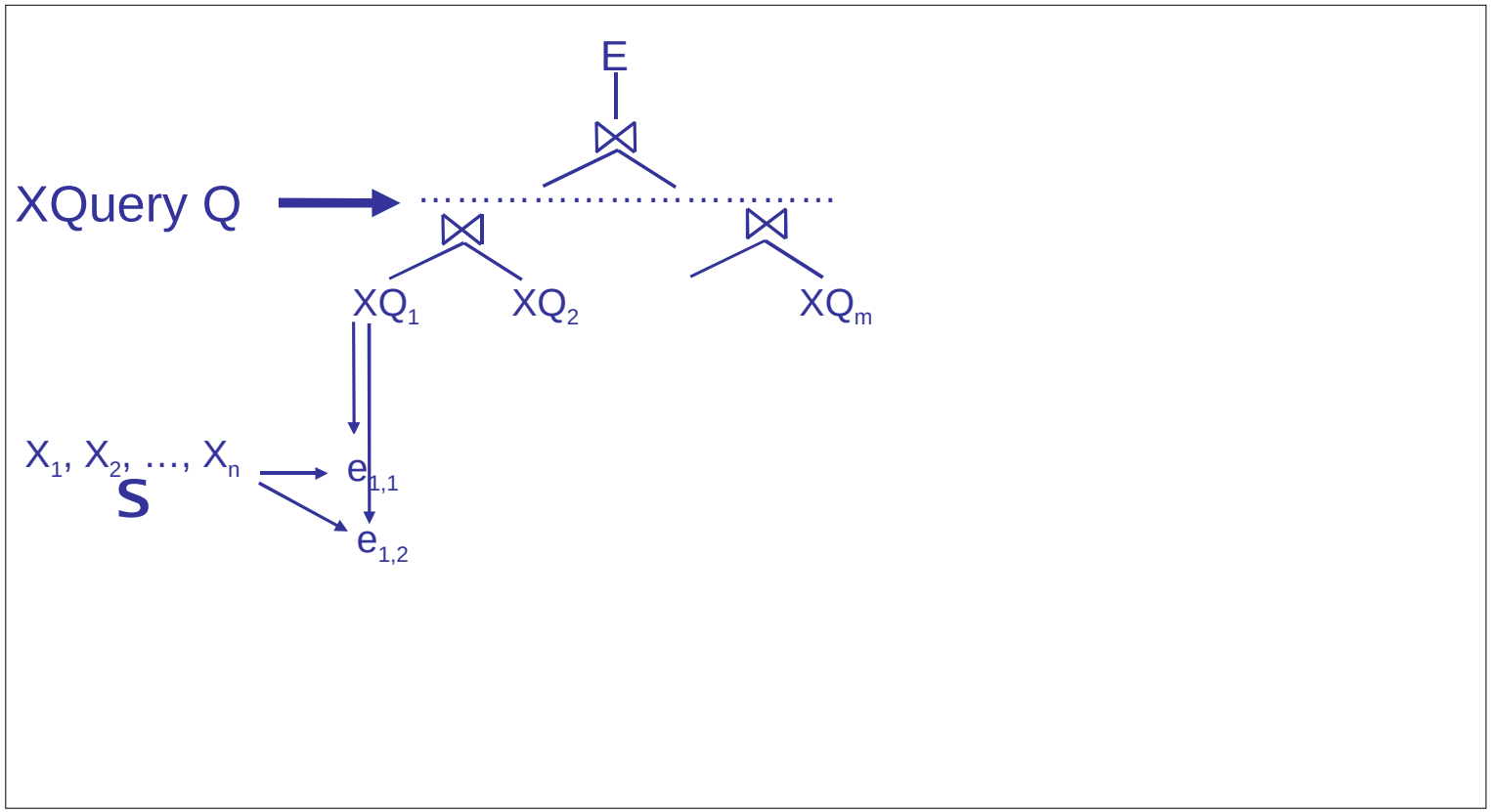
Query rewriting algorithm



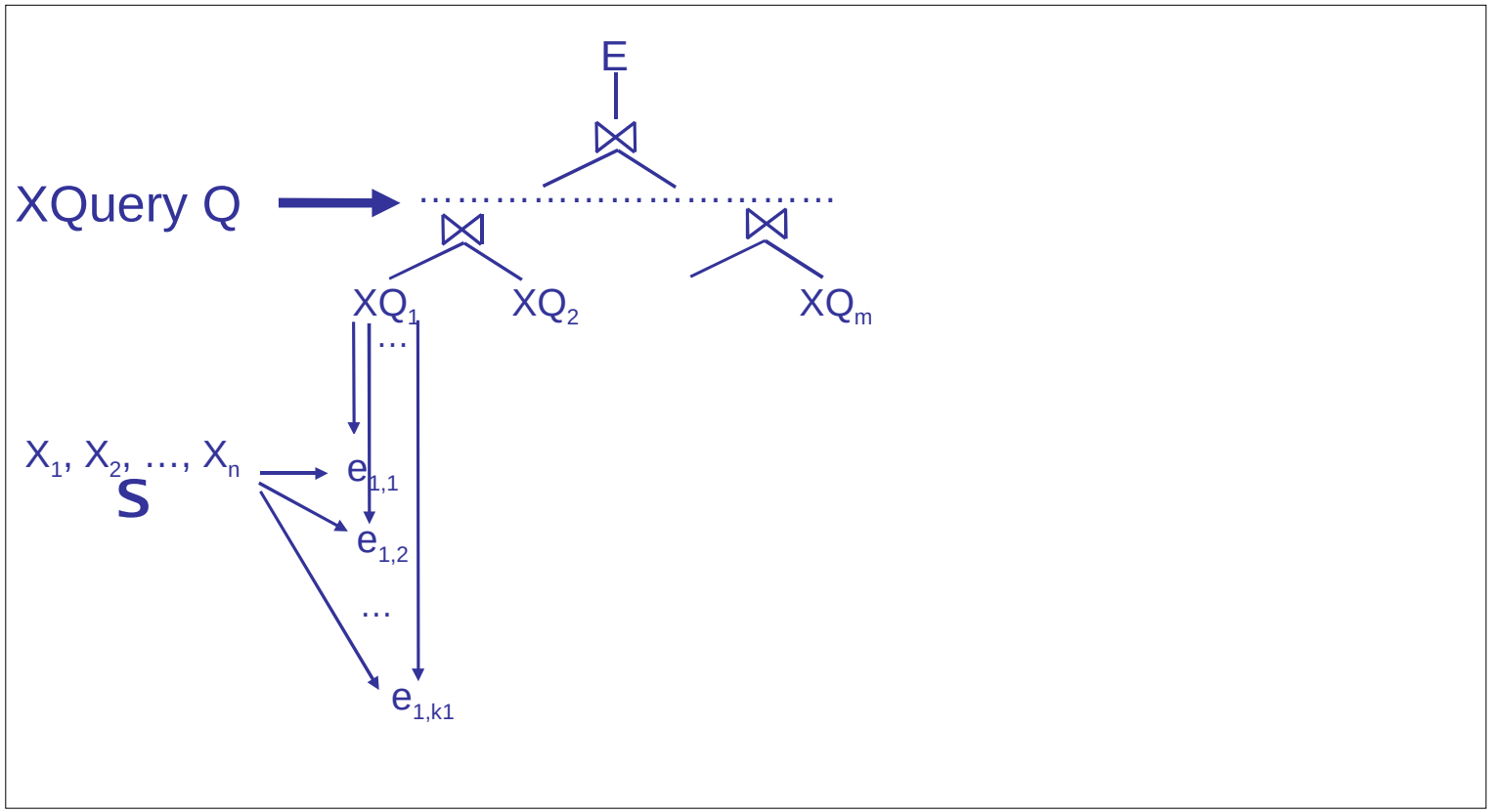
Query rewriting algorithm



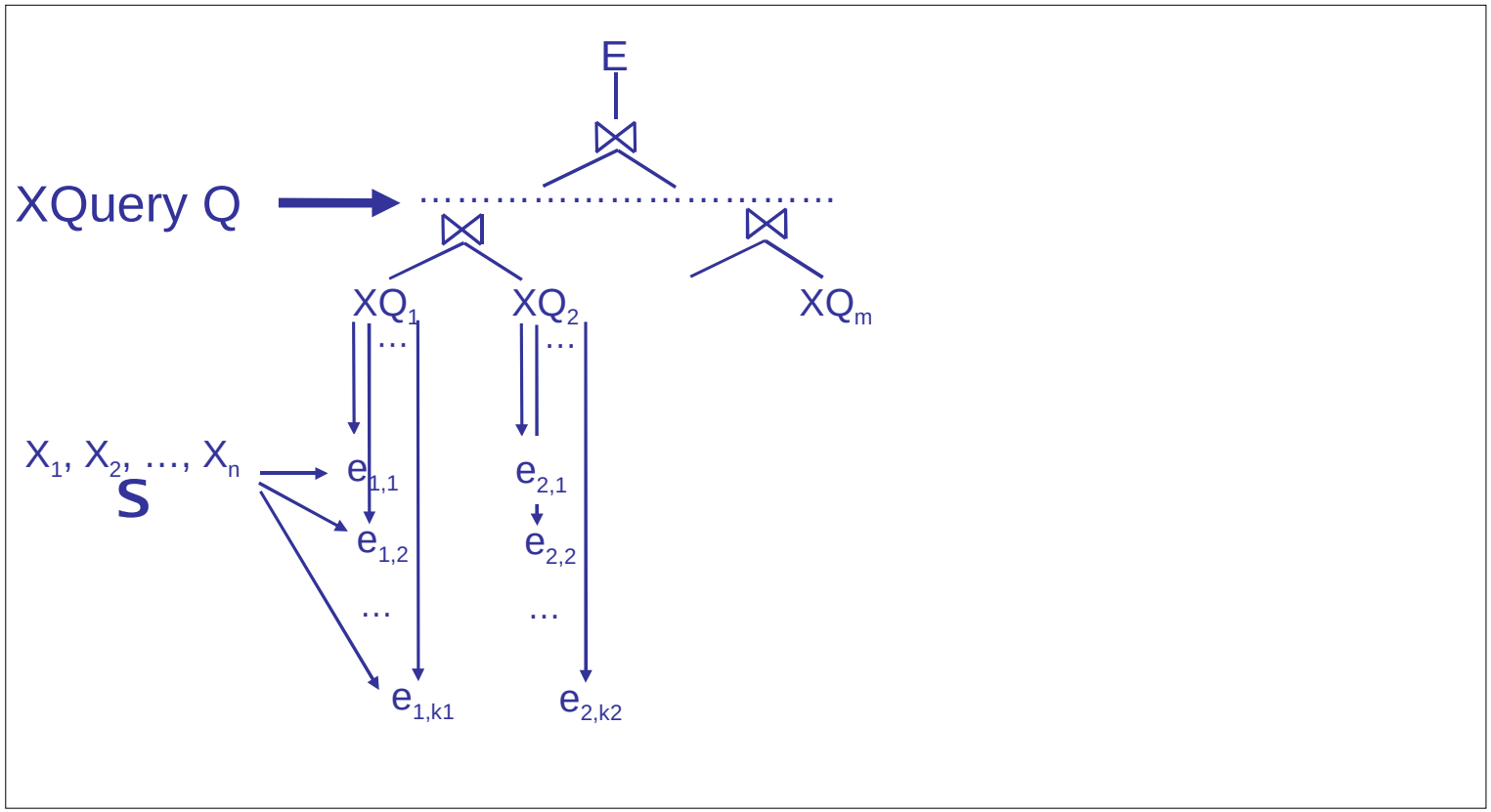
Query rewriting algorithm



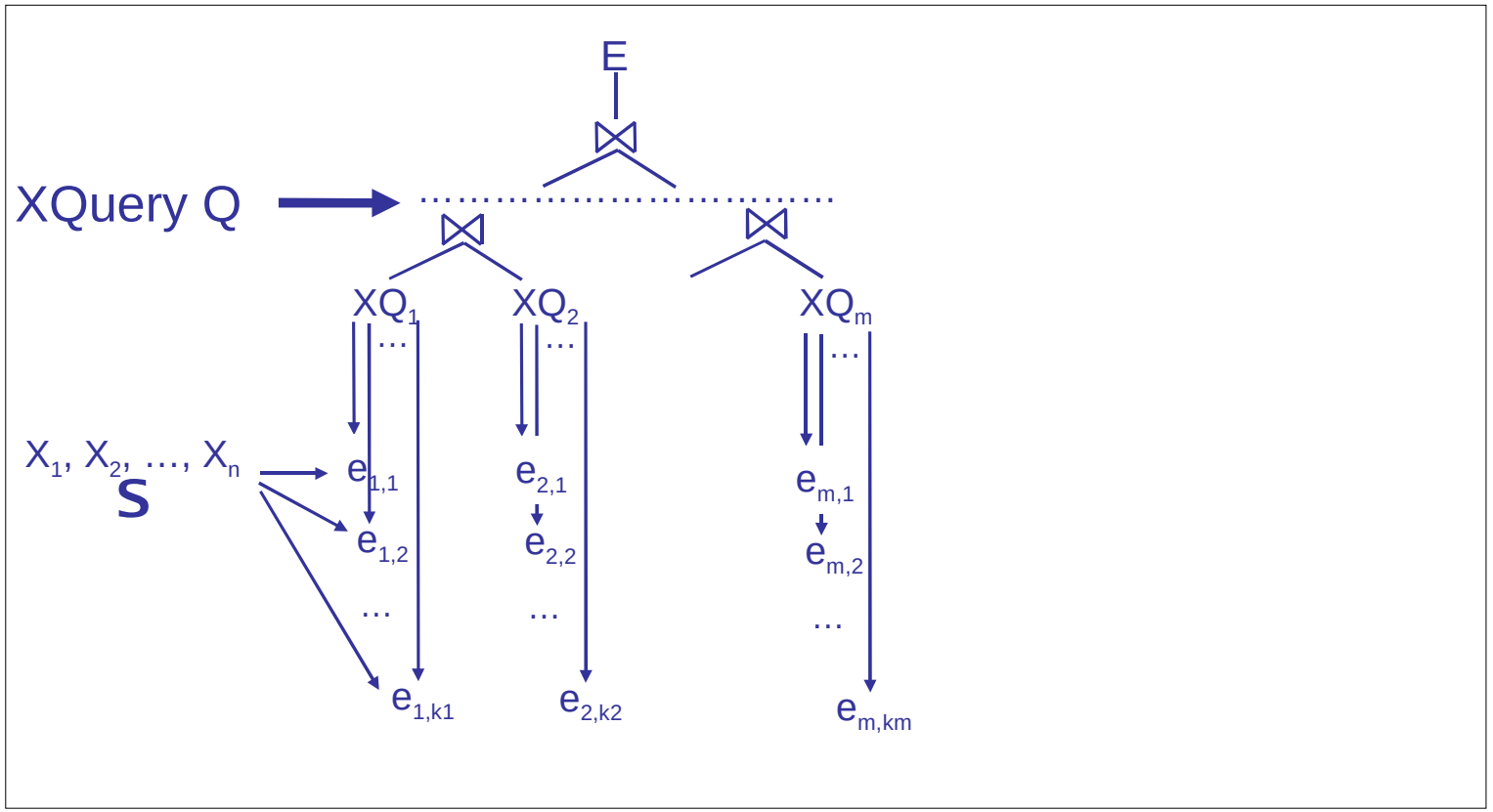
Query rewriting algorithm



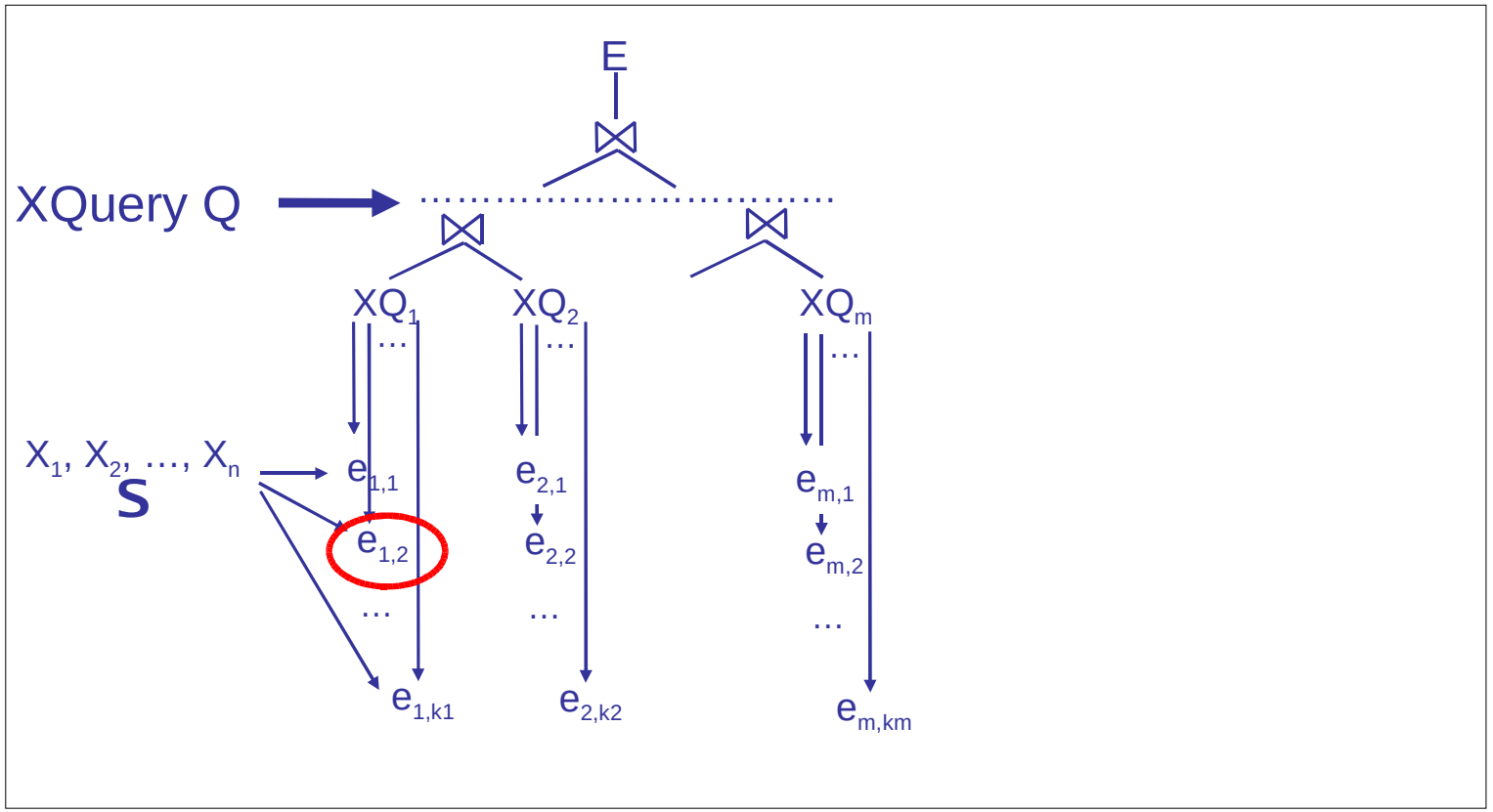
Query rewriting algorithm



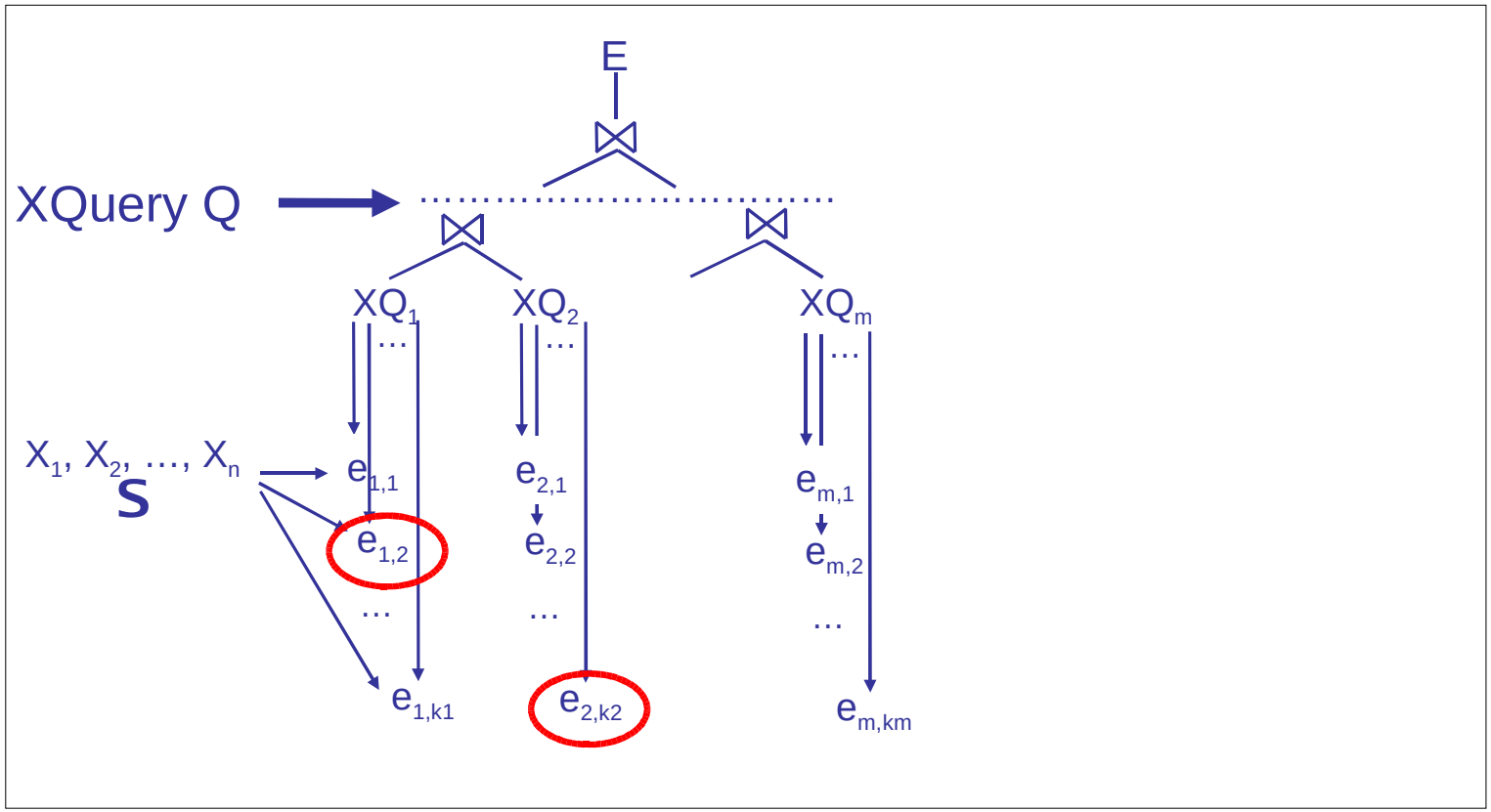
Query rewriting algorithm



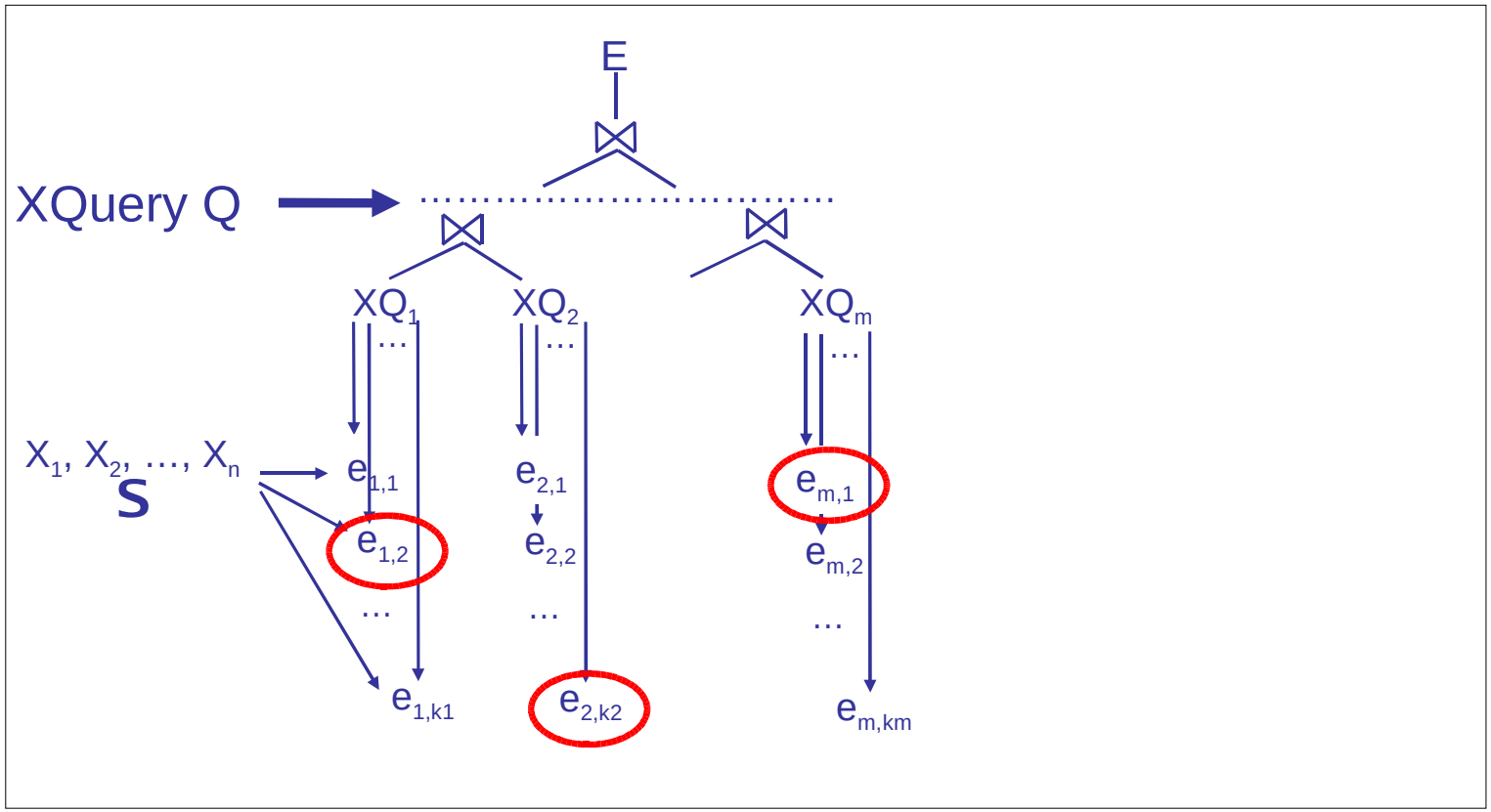
Query rewriting algorithm



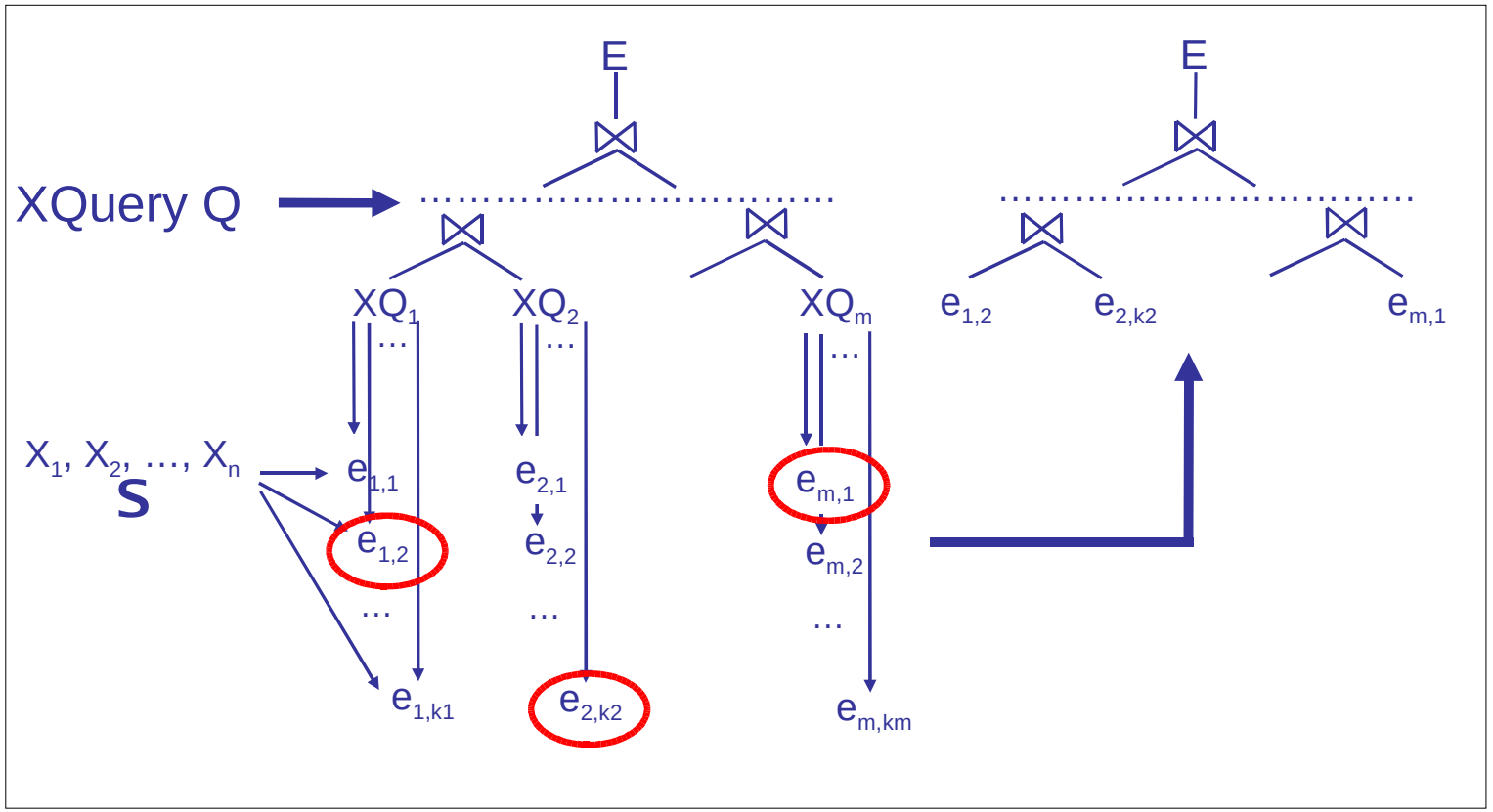
Query rewriting algorithm



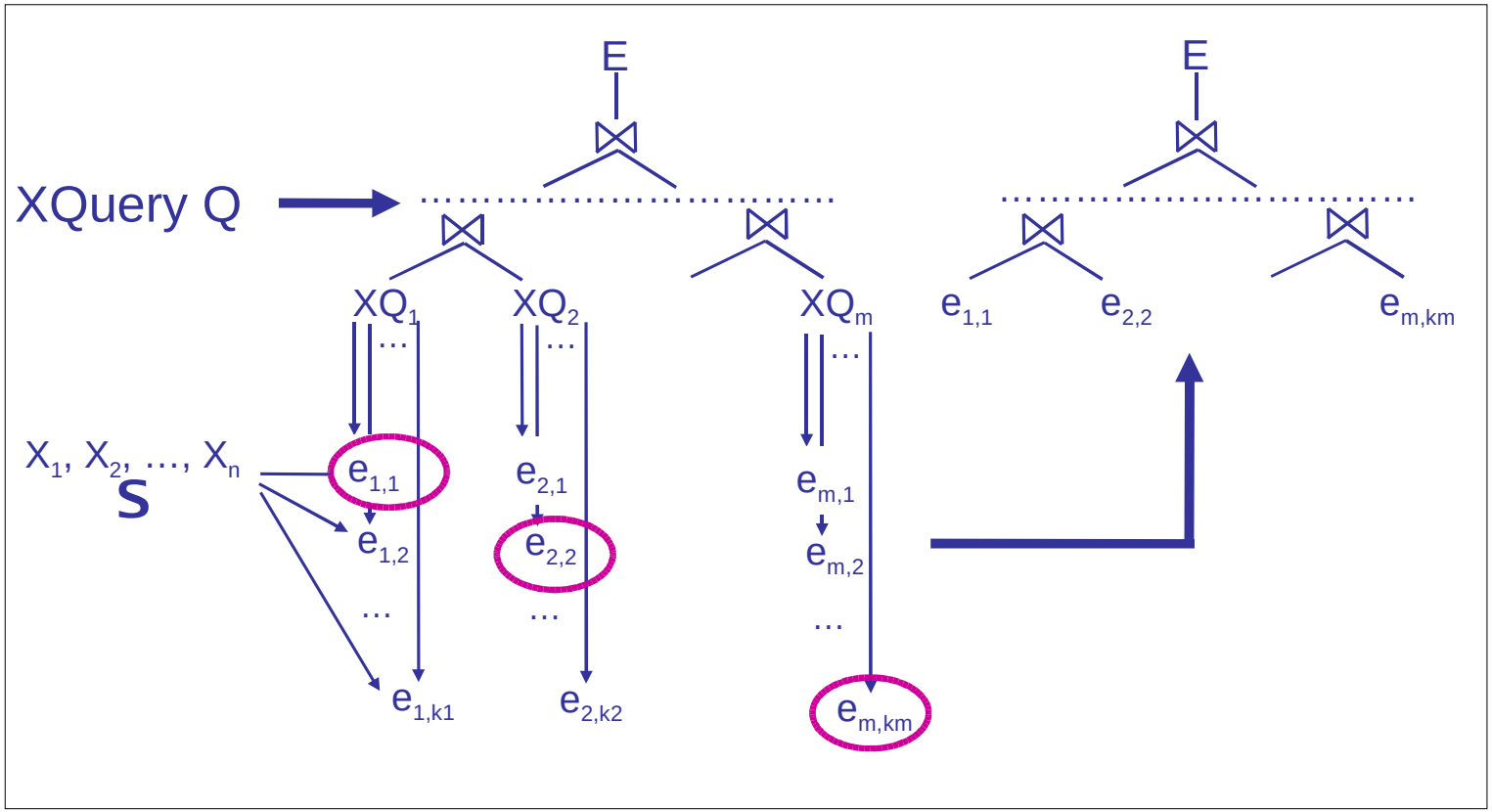
Query rewriting algorithm



Query rewriting algorithm



Query rewriting algorithm



Zoom on XAM rewriting

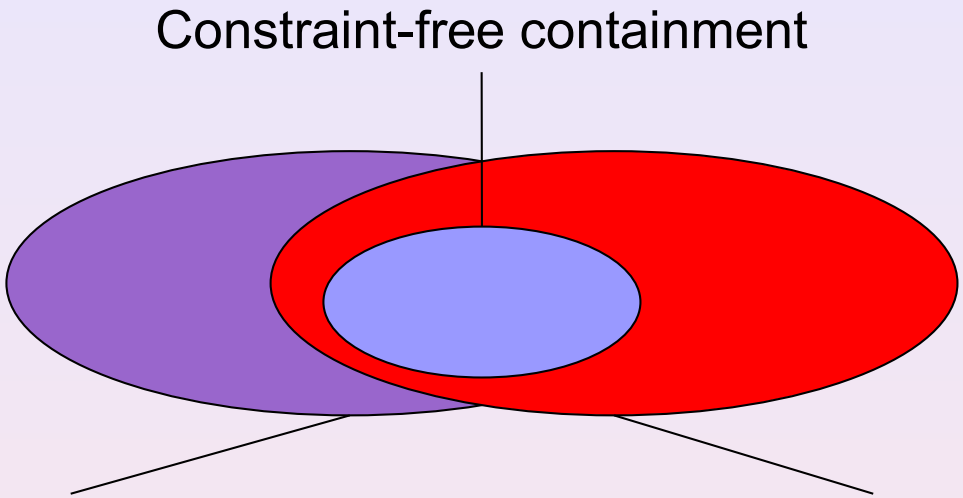
Rewriting one query XAM

- In the presence of a summary bucket algorithms would be incomplete
- Inflationary algorithm:
  - Builds (structural) joins
  - a plan is a full rewriting when its equivalent XAM is $\equiv_S$ to the query XAM.
  - XAM containment algorithm under S constraints
- When to stop: no new plans or plans outgrow a certain size
- Agressive summary-based pruning of plans to keep search space manageable

Full results

- XAM language
  - Value and structure predicates
  - Nesting
  - Optional edges
  - Structural and non-structural IDs
- Enhanced summaries: integrity constraints (required children)

Summary vs. DTD constraints



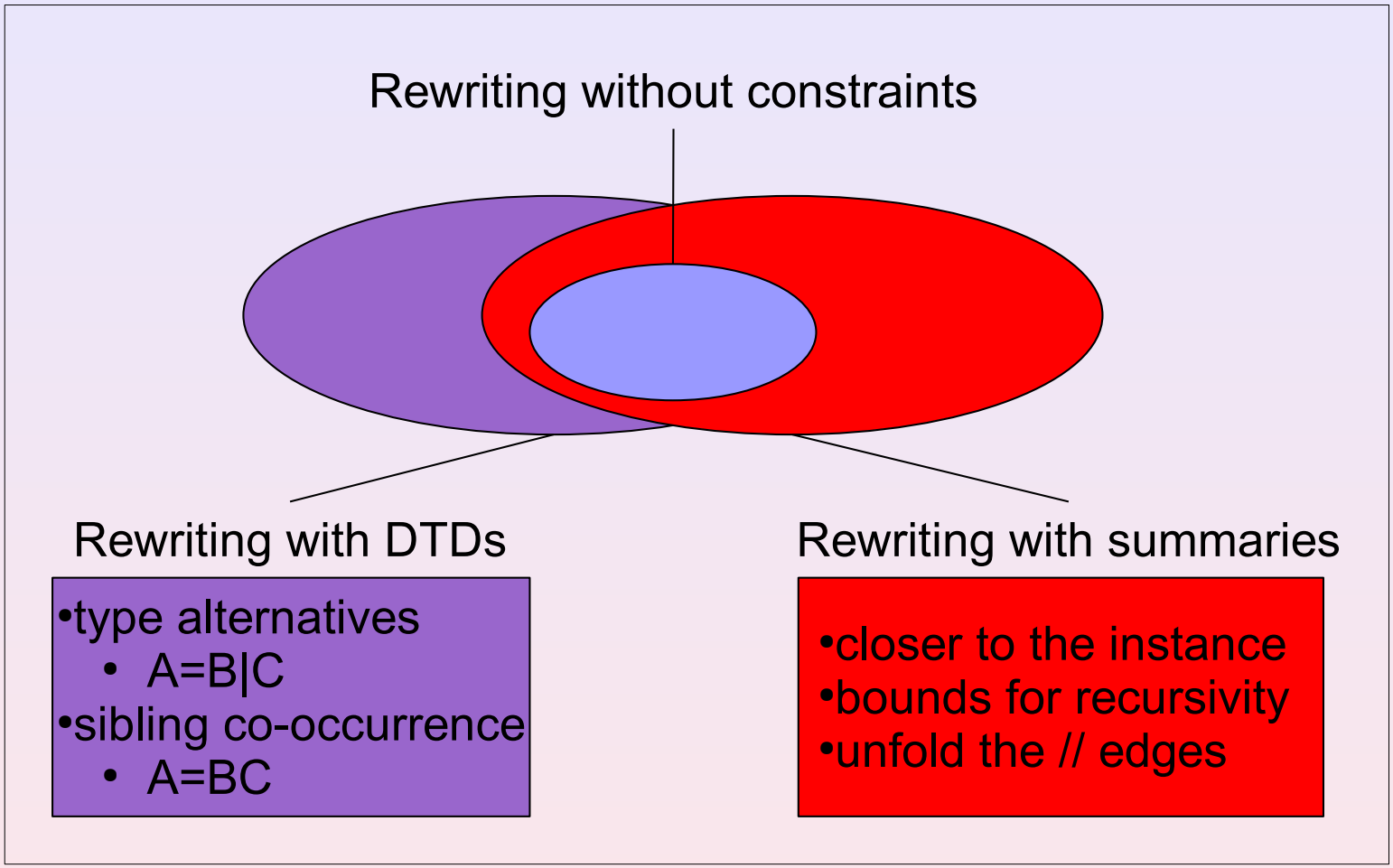
DTD-based containment

- type alternatives
  - A=B|C
- sibling co-occurrence
  - A=BC

Summary based containment

- closer to the instance
- bounds for recursivity
- unfold the // edges

Summary vs. DTD constraints



Example: Is there a rewriting //a=//b/a + //c/a ?

Outline

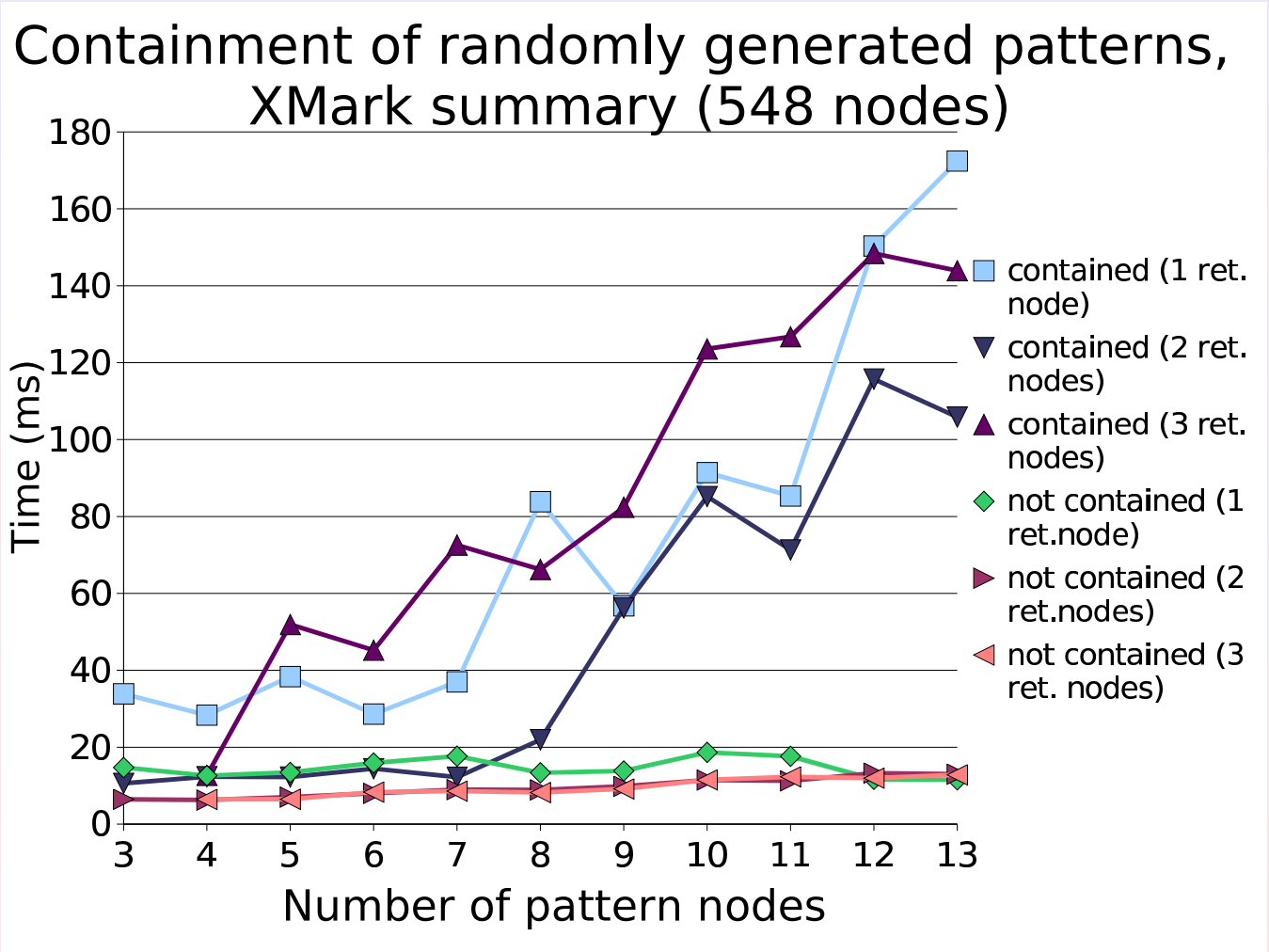
1. Context: rewriting XQuery using nested views
2. Pattern containment under summary constraints
   - Path summary
   - Summary based containment
3. Summary based query rewriting
4. Experimental results
5. Conclusions

Experimental setting

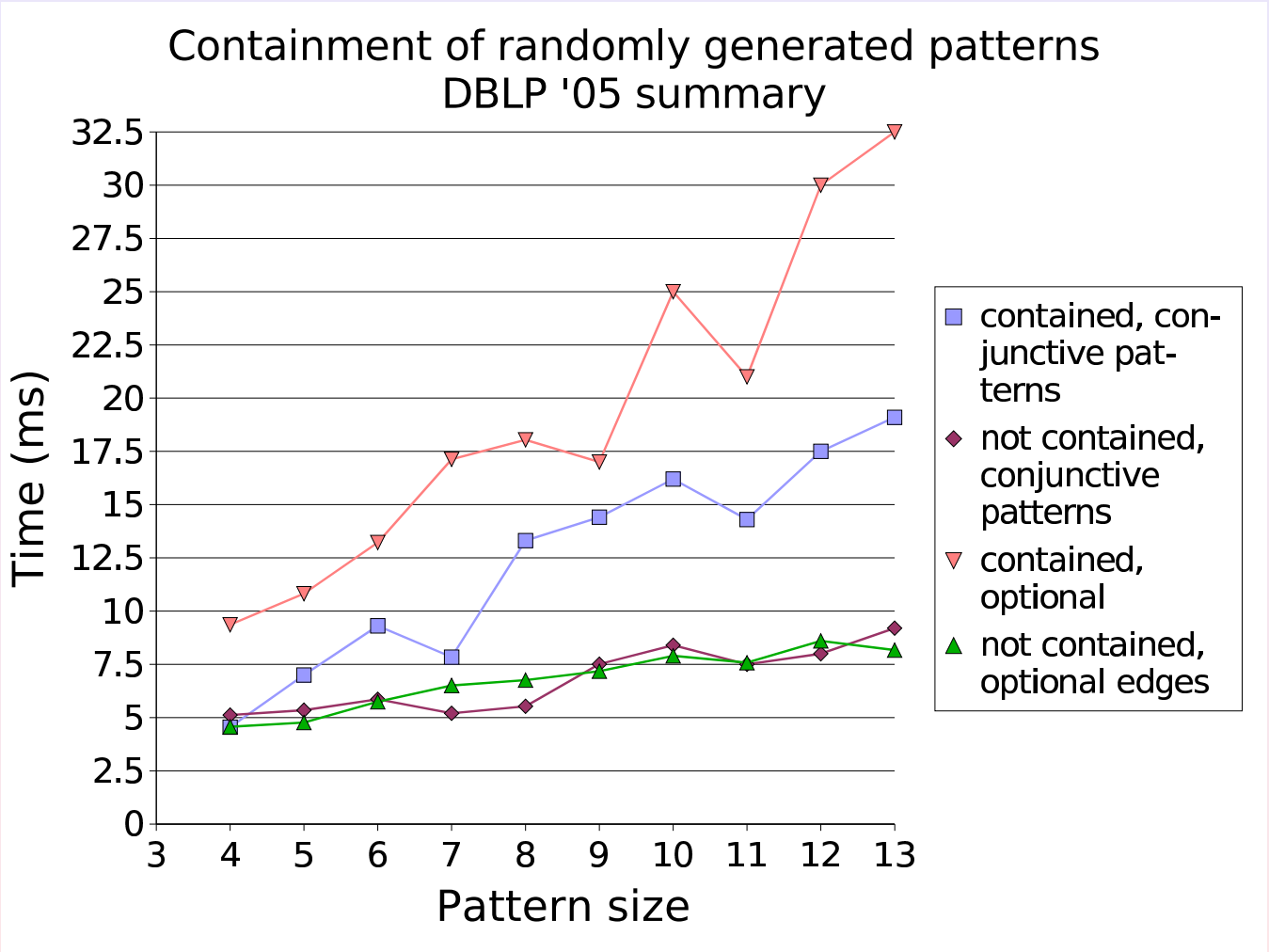
ULoad prototype, Java 1.5, 1GHz CPU, 1GB RAM

- Documents
  - DBLP 2005 (280MB)
  - XMark (233MB)
- Queries
  - XMark query patterns
  - Randomly generated patterns based on DBLP and XMark path summaries
- Views
  - 1 view per each XMark tag (IDs V)
  - randomly generated patterns based on XMark and DBLP

Pattern containment results

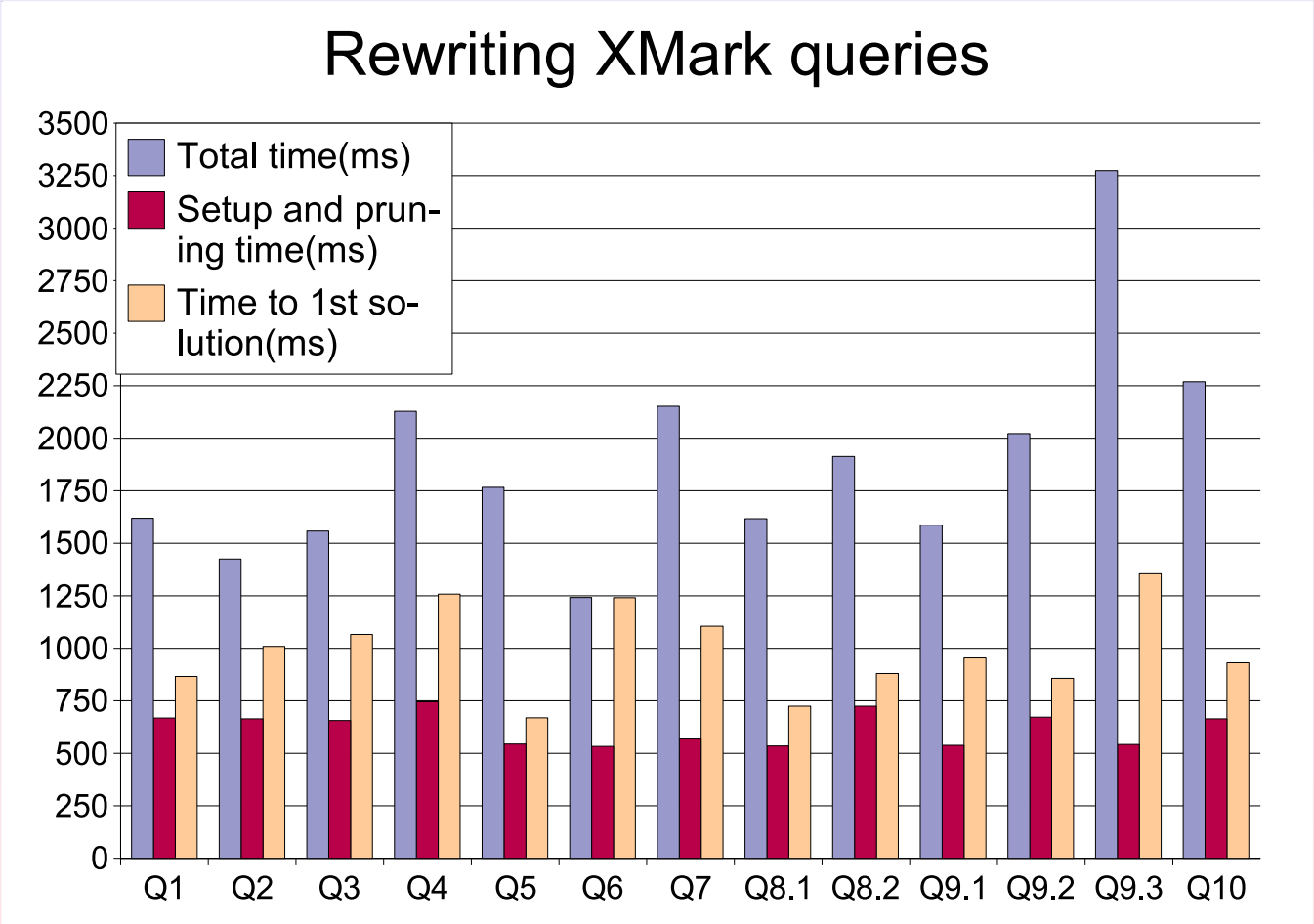


Pattern containment results



Query rewriting

- 183 views: 100 generated + 83 tag partition



Outline

1. Context: rewriting XQuery using nested views
2. Pattern containment under summary constraints
   - Path summary
   - Summary based containment
3. Summary based query rewriting
4. Experimental results
5. Conclusions

Conclusions

Contributions

- XML query pattern containment and equivalent rewriting based on summary constraints
- exploiting detailed informations about view contents and IDs expressed by XAMs
- query rewriting using XAMs is feasible

Future work

- Cost-based materialized view selection
- View maintenance in the presence of updates

Questions?

More information

- ULoad, XAM home: http://gemo.futurs.inria.fr/projects/XAM/
- A.Arion, V.Benzaken, I.Manolescu: ”XML Access Modules: Towards Physical Independence in XML Databases”, [XIMEP2005]
- A.Arion, V.Benzaken, I.Manolescu, R.Vijay: ”ULoad: Finding the Best Storage for your XML Management Application”, [VLDBDemo2005]
- A.Arion, V.Benzaken, I.Manolescu, Y.Papakonstantinou and R.Vijay: ”Algebra-Based Tree Pattern Extraction from XQuery”, [FQAS2006]
- I.Manolescu, V.Benzaken, A.Arion and Y.Papakonstantinou: ”Structured Materialized views for XML Queries”, INRIA report, http://hal.inria.fr, 2006