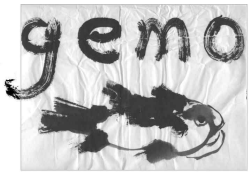
Materialized views for P2P XML warehousing

Ioana Manolescu[1] **Spyros Zoupanos**[1]

[1] GEMO/IASI group, INRIA Saclay – Île-de-France

July 2009
Dataring project meeting, Nantes



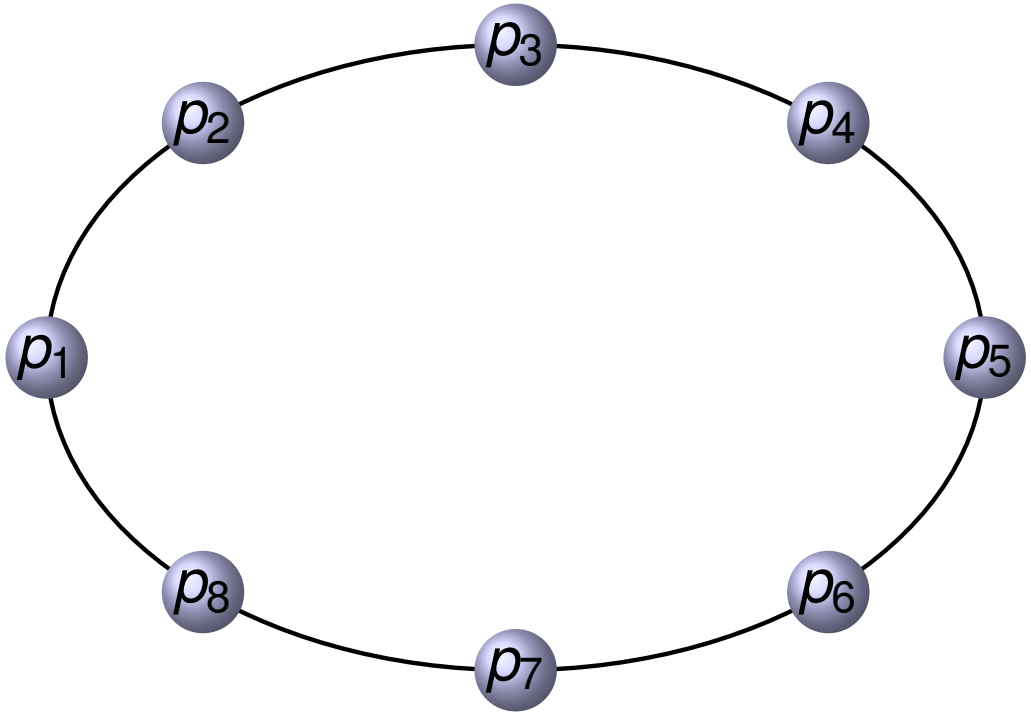
Outline

ViP2P at a glance

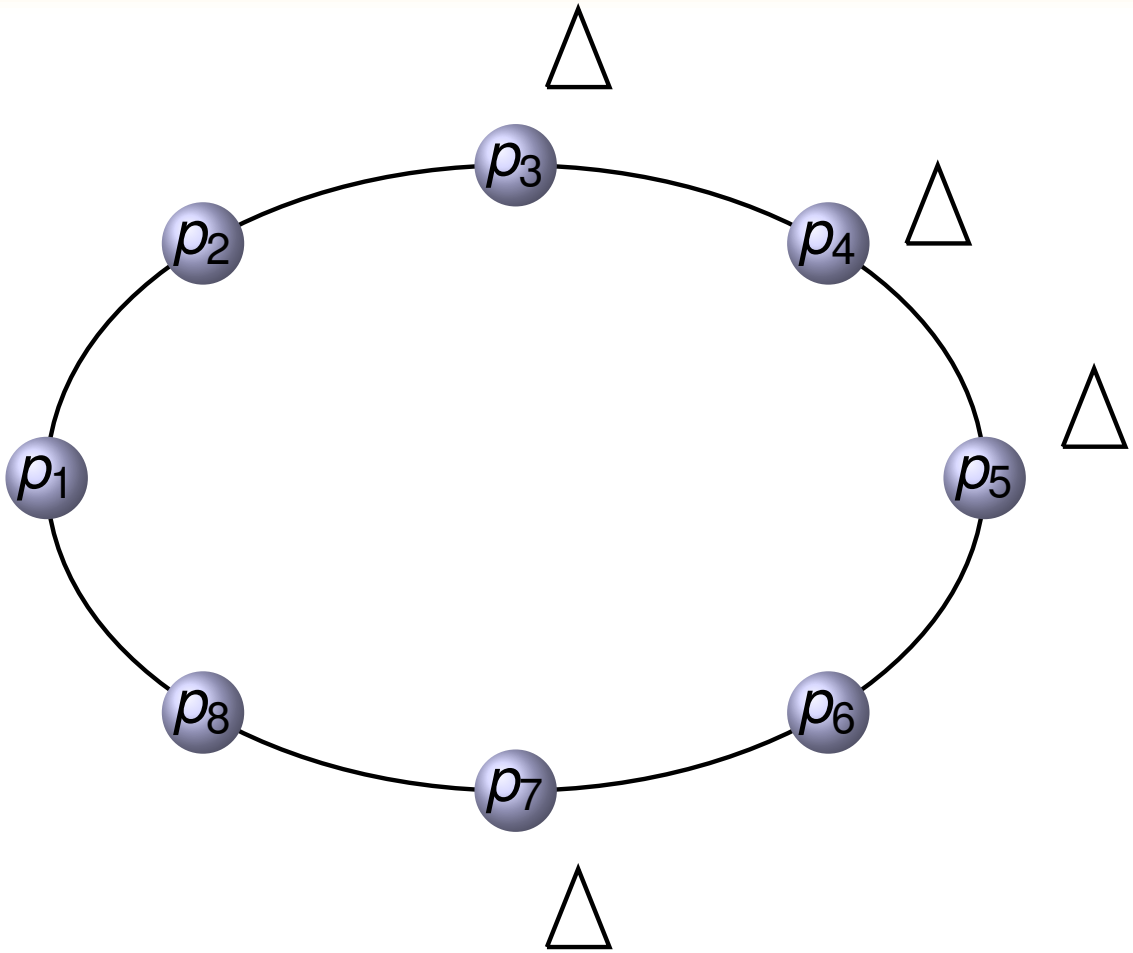
An XML data management system based on a DHT

- Declare tree pattern XML views over the network data
- Fill in the views with XML data
- Answer tree pattern queries using the existing views
  - View definition lookup
  - Query rewriting $\Rightarrow$ logical plan
  - Translation to a (distributed) physical plan
  - Execution of the physical plan

Architecture overview



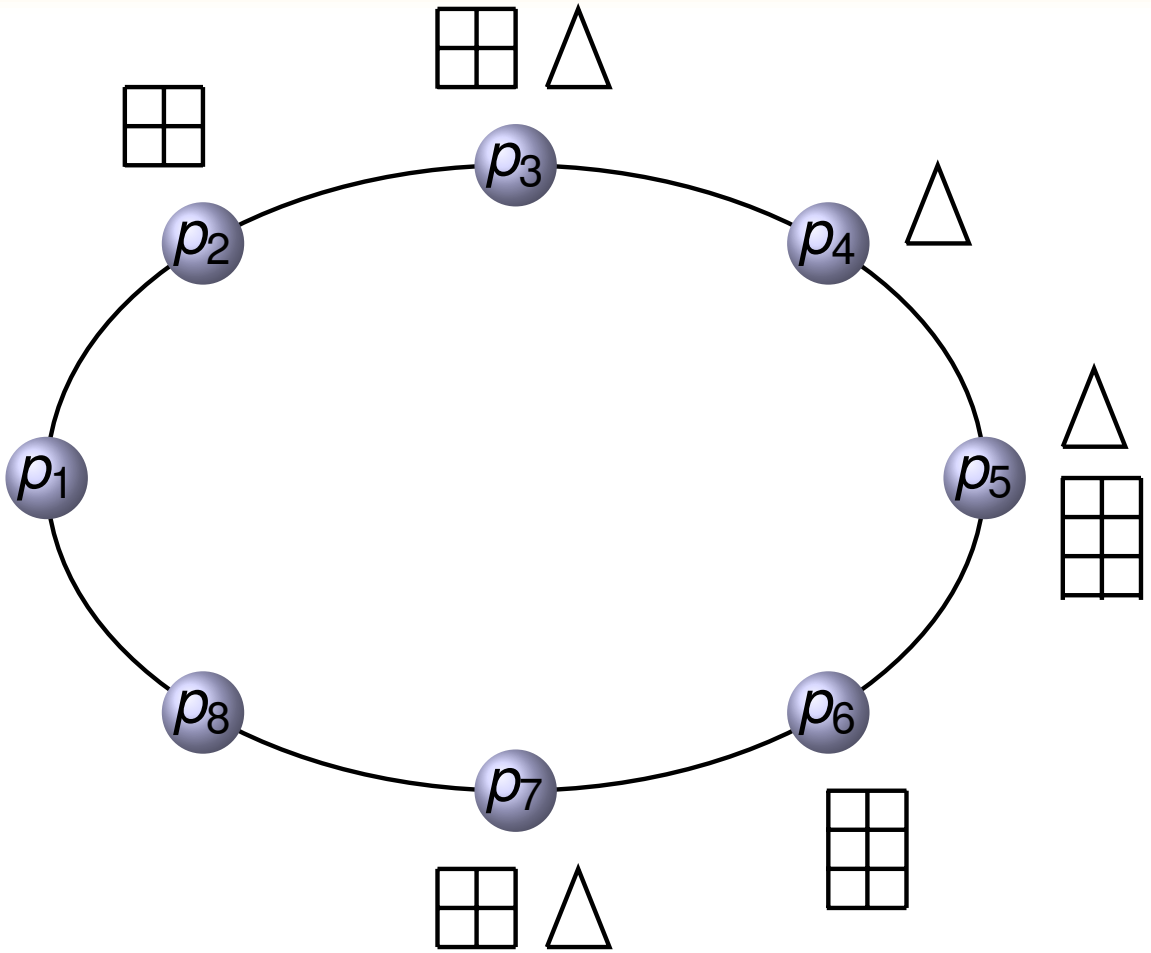
Architecture overview



The peers may store:

- documents

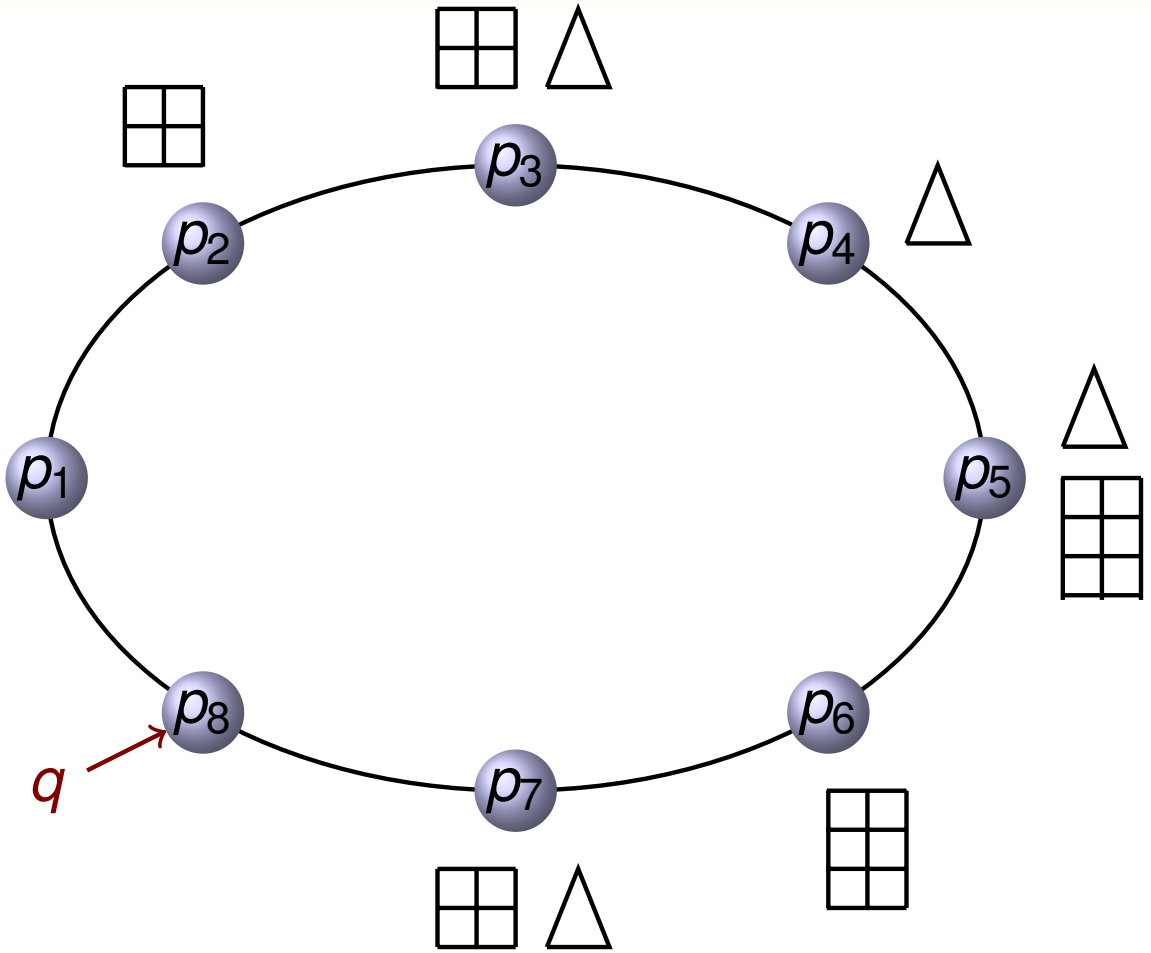
Architecture overview



The peers may store:

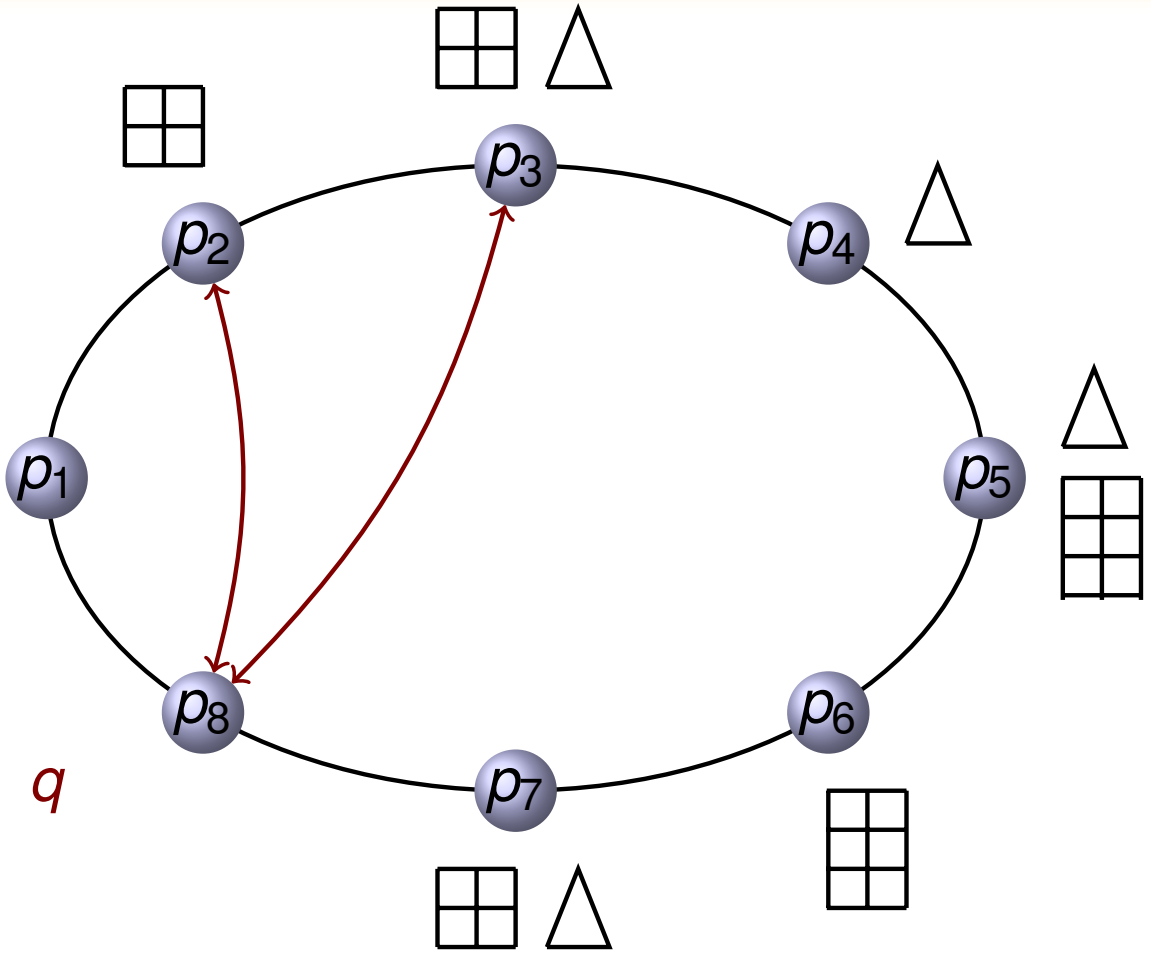
- documents
- views

Architecture overview



When q arrives:

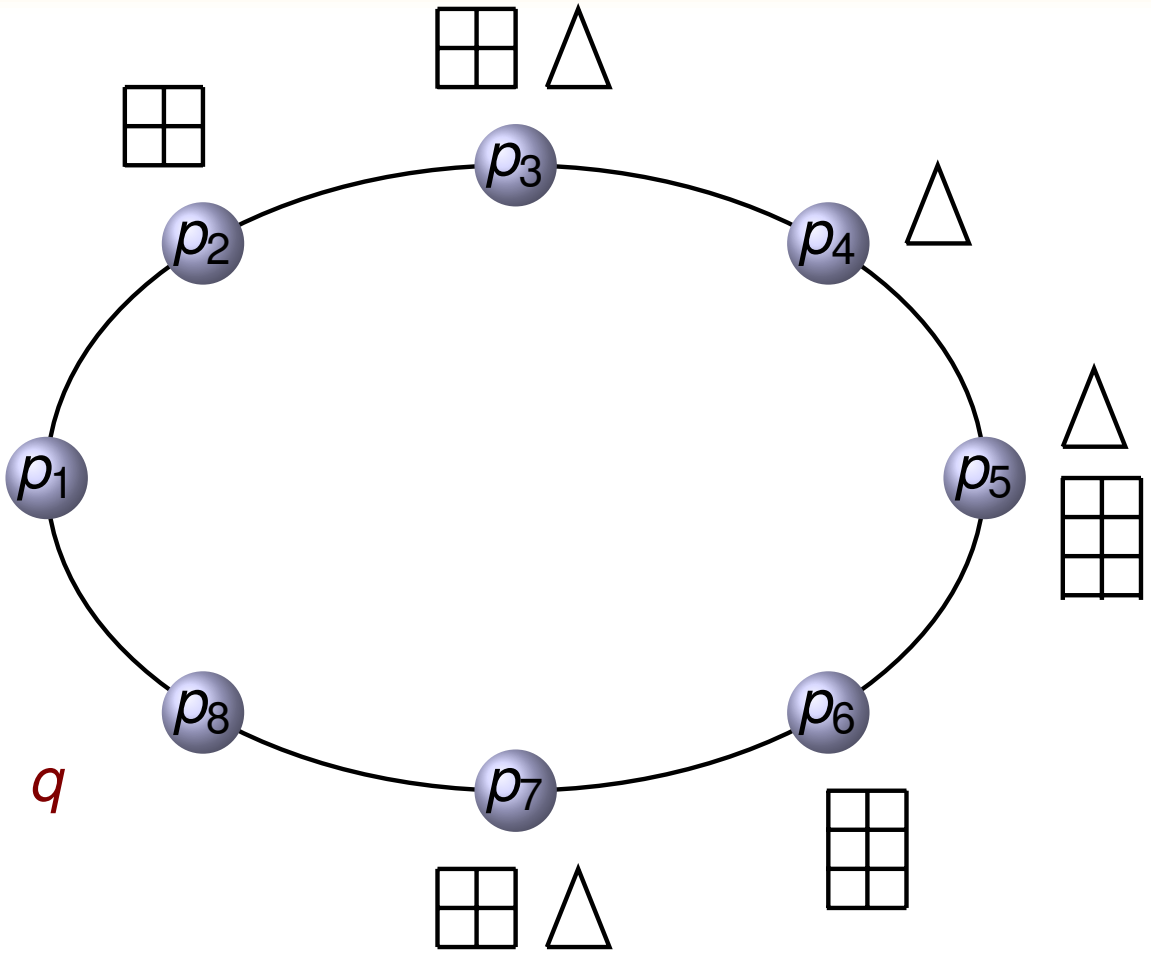
Architecture overview



When q arrives:

- view definition lookup

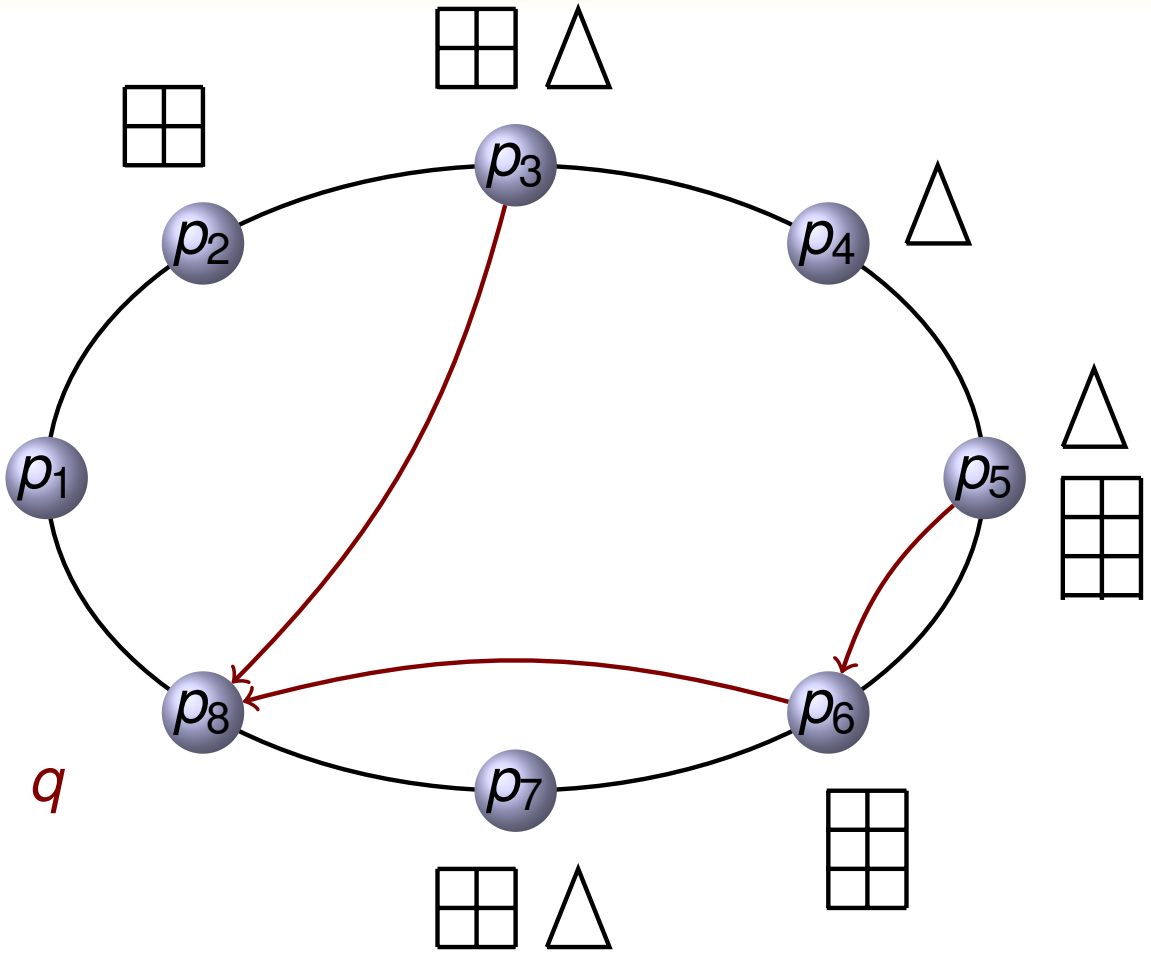
Architecture overview



When q arrives:

- view definition lookup
- rewriting

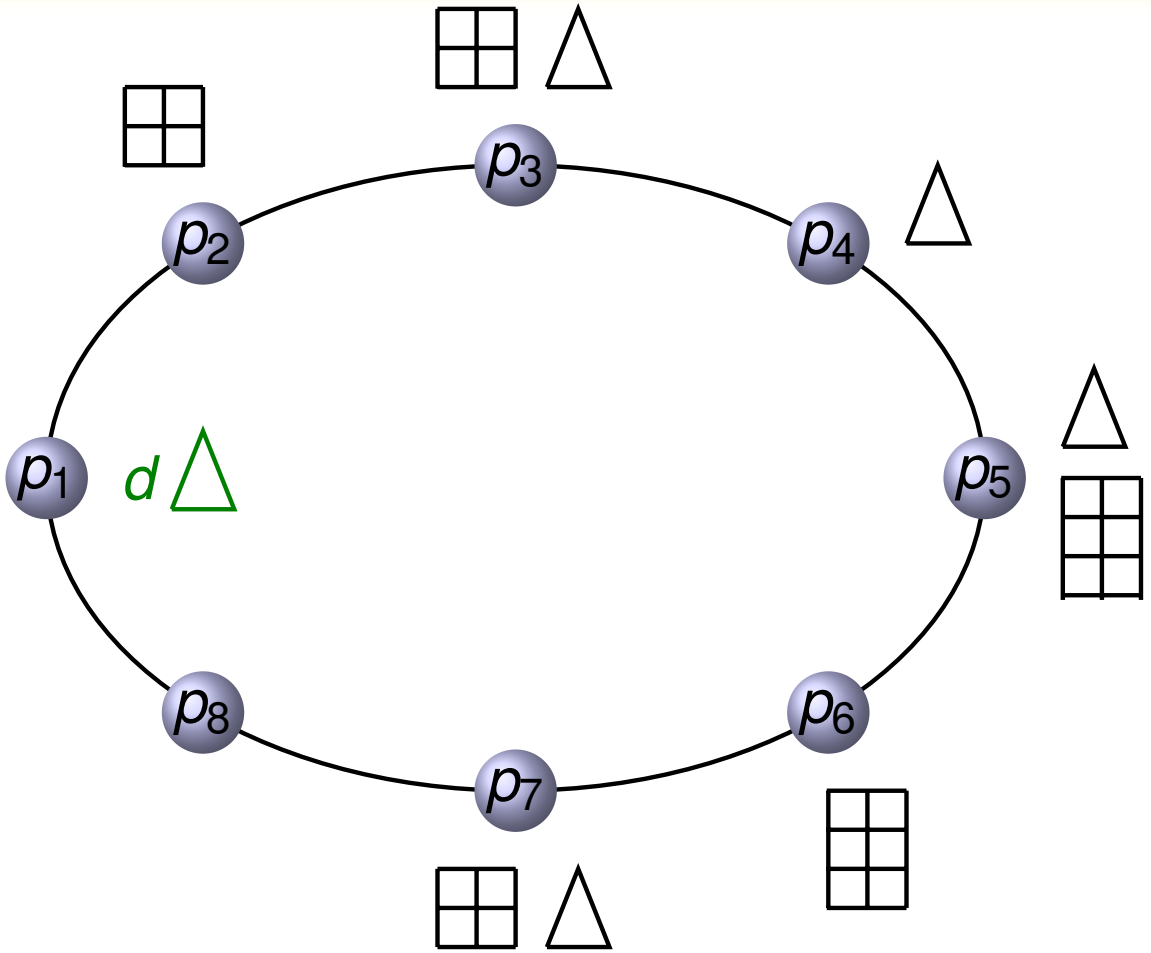
Architecture overview



When q arrives:

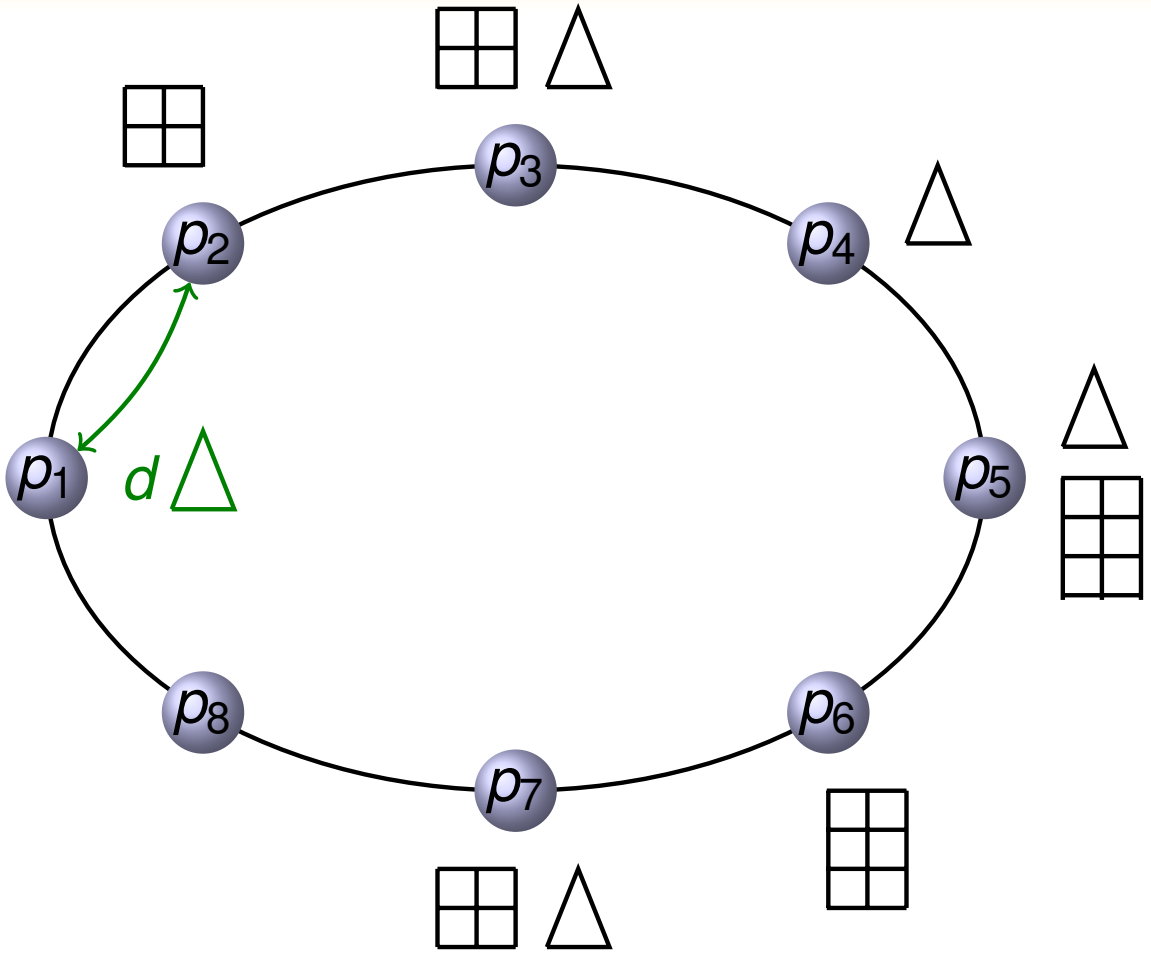
- view definition lookup
- rewriting
- execution of physical plan

Architecture overview



When *d* arrives:

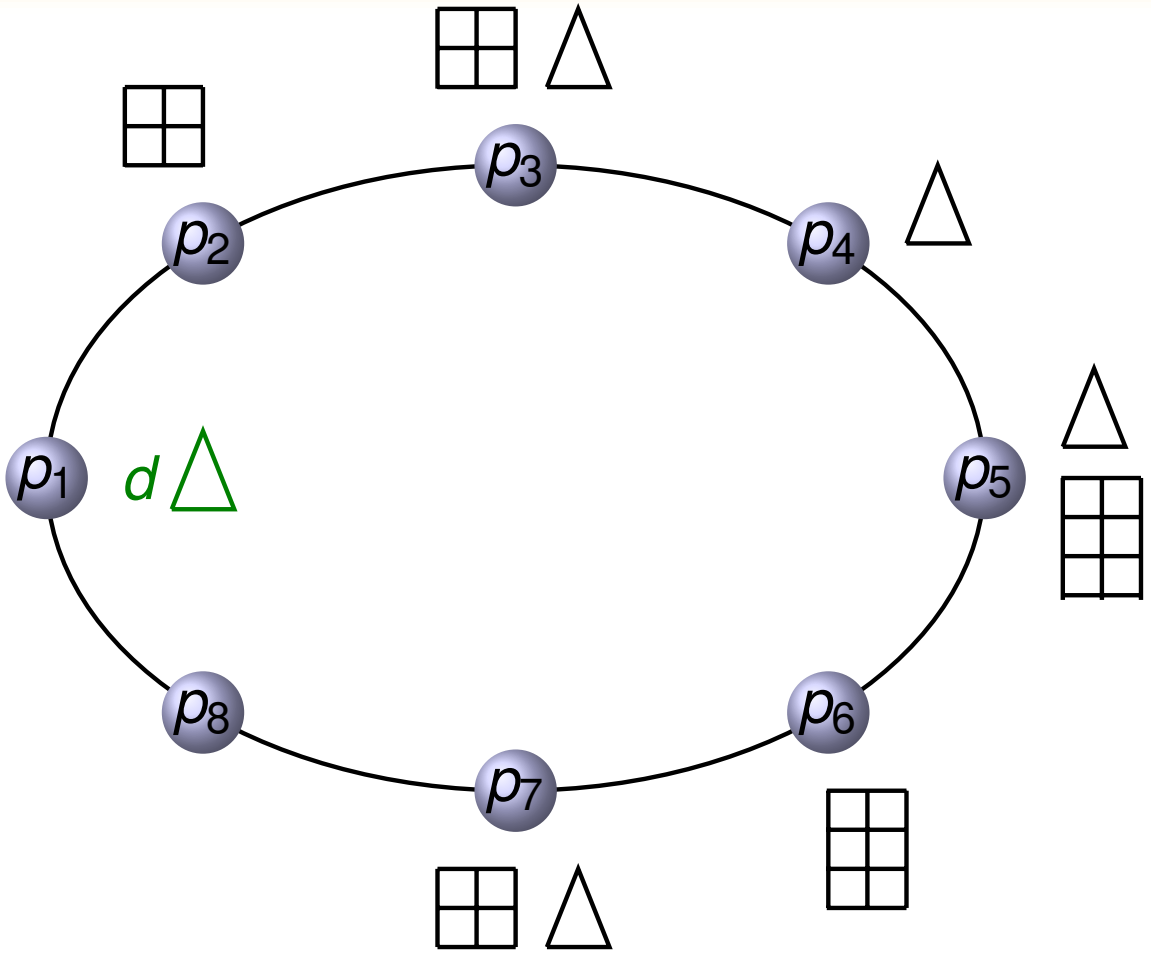
Architecture overview



When d arrives:

- search view definitions for which $v_i(d) \neq \emptyset$

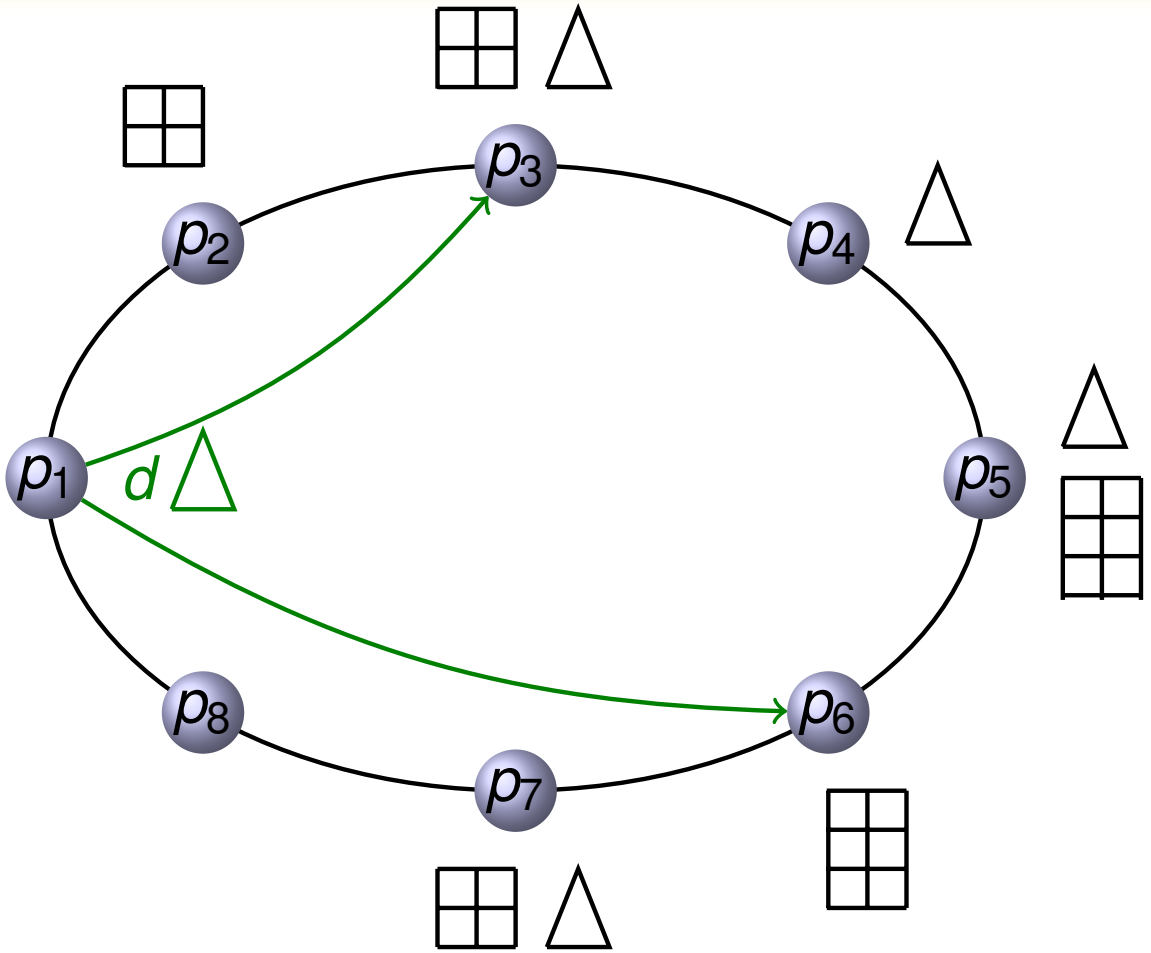
Architecture overview



When d arrives:

- search view definitions for which $v_i(d) \neq \emptyset$
- compute $v_i(d)$

Architecture overview



When d arrives:

- search view definitions for which $v_i(d) \neq \emptyset$
- compute $v_i(d)$
- send results

Tree pattern dialect P

a_{cont}

Tree pattern dialect P

a_{cont} $a_{id,cont}$

Tree pattern dialect P

a_{cont} $a_{id,cont}$ $a_{id,val}$

Tree pattern dialect P

a_{cont} $a_{id,cont}$ $a_{id,val}$

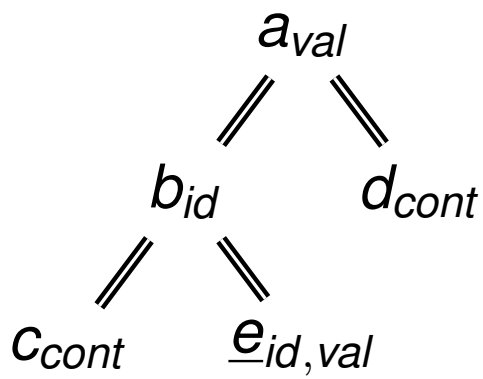
a_{id}

|

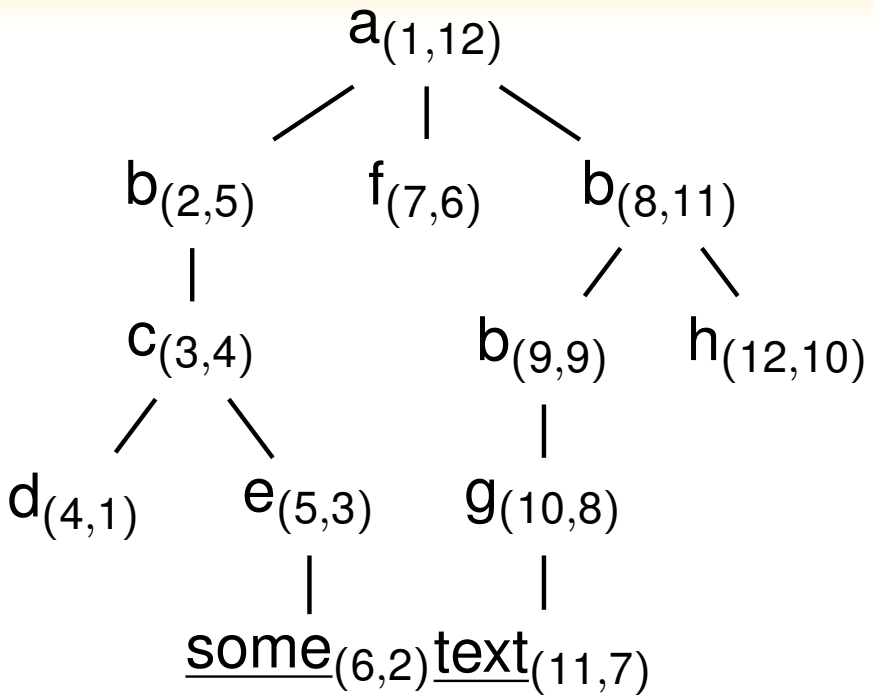
b_{val}

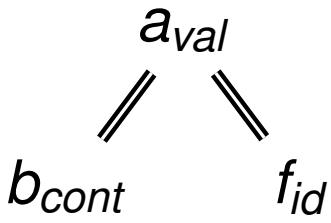
Tree pattern dialect P

a_{cont} $a_{id,cont}$ $a_{id,val}$



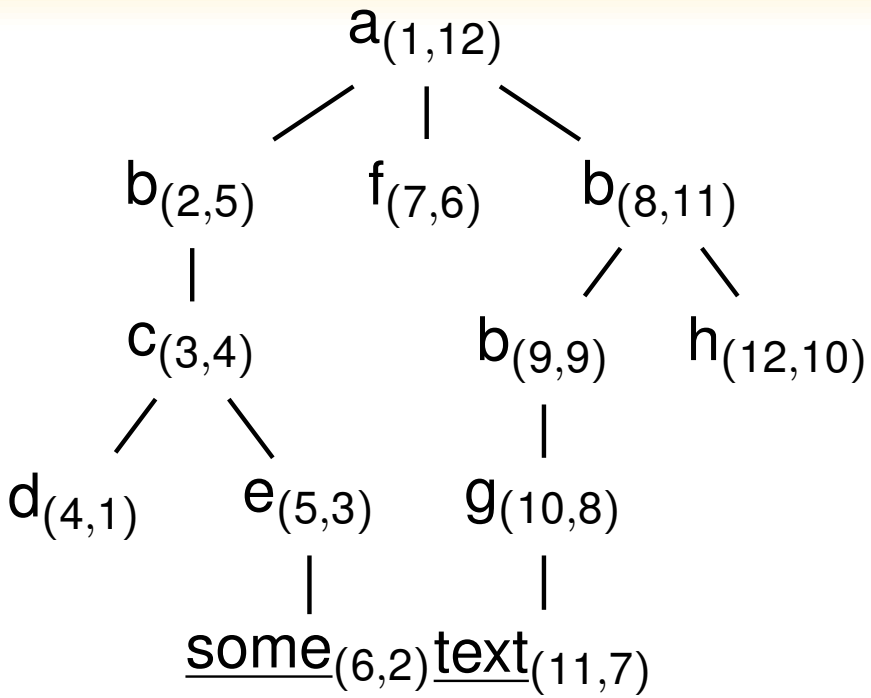
Pattern semantics and equivalence

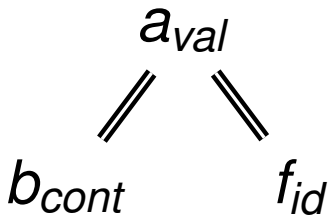




a_{val}	b_{cont}	f_{id}

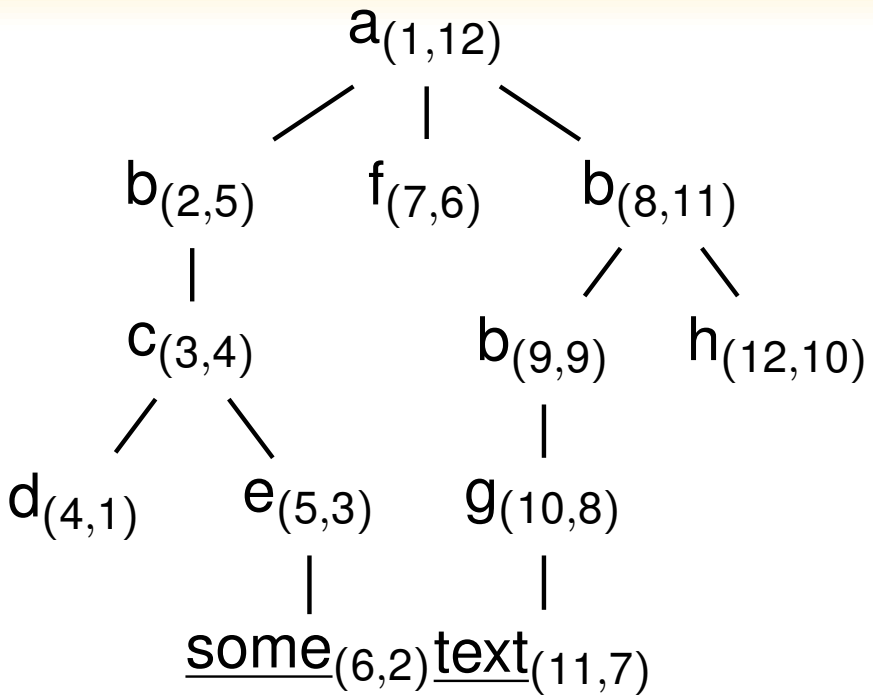
Pattern semantics and equivalence

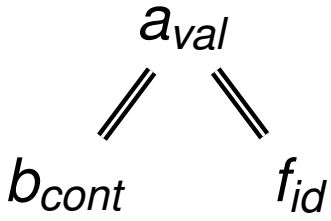




a_{val}	b_{cont}	f_{id}
some text	<c><d/><e>some</e></c>	(7,6)

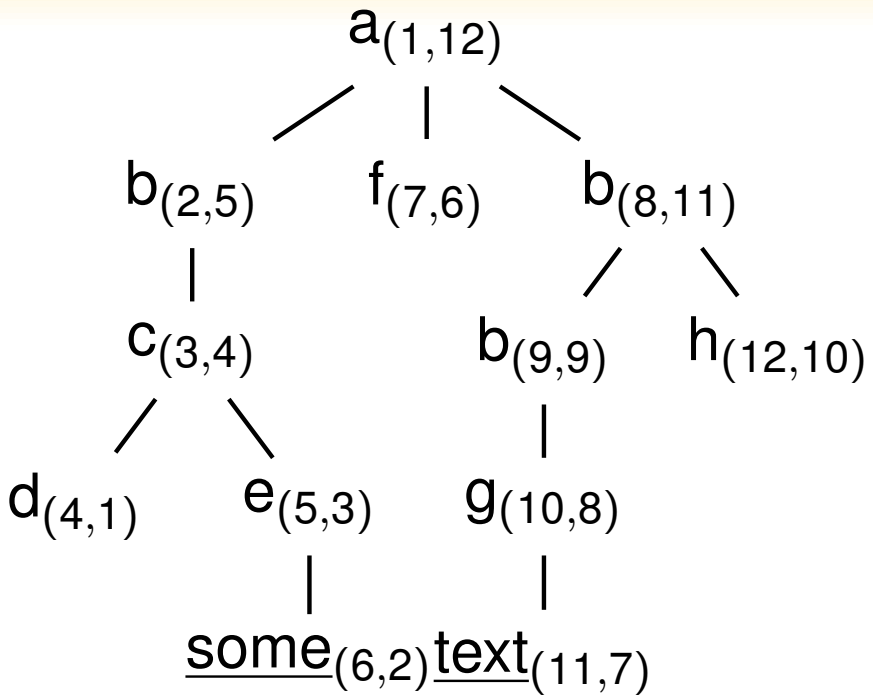
Pattern semantics and equivalence

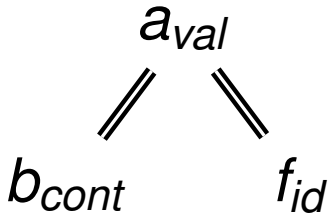




a_{val}	b_{cont}	f_{id}
some text	<c><d/><e>some</e></c>	(7,6)
some text	<g>text</g>	(7,6)

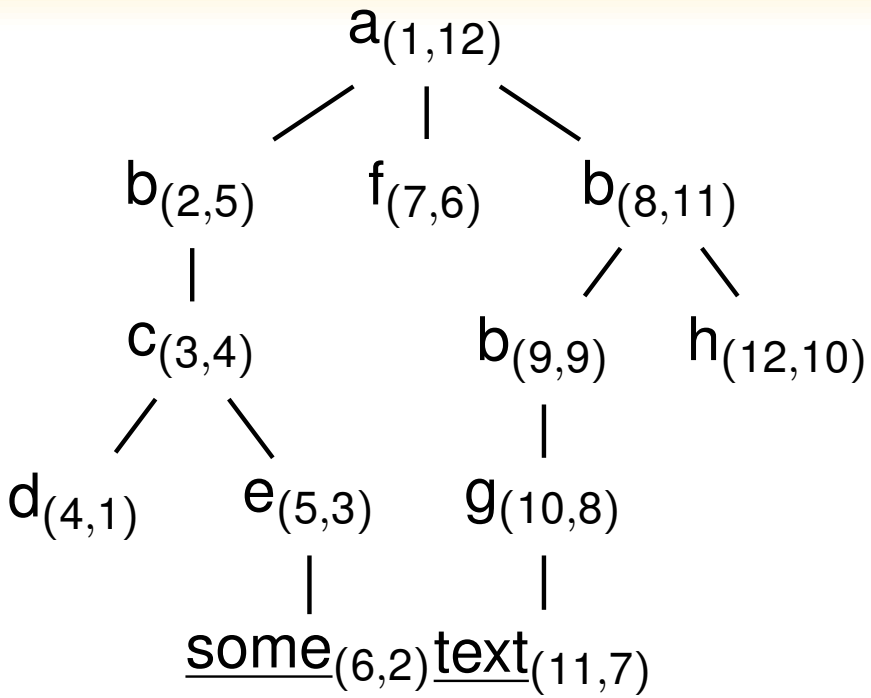
Pattern semantics and equivalence

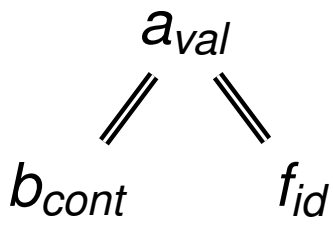




a_{val}	b_{cont}	f_{id}
some text	<c><d/><e>some</e></c>	(7,6)
some text	<g>text</g>	(7,6)
some text	<g>text</g>	(7,6)

Pattern semantics and equivalence





- Document d, pattern $p \Rightarrow p(d)$
- $p_1 \equiv p_2$ *iff* $\forall\, d$, $p_1(d) = p_2(d)$

a_{val}	b_{cont}	f_{id}
some text	<c><d/><e>some</e></c>	(7,6)
some text	<g>text</g>	(7,6)
some text	<g>text</g>	(7,6)

Algebraic rewriting & operators

Let $q \in \mathcal{P}$ be a query and $\mathcal{V} = \{v_1, v_2, \ldots, v_k\}$ a set of views.
A **rewriting** of q using $\mathcal{V}$ is an algebraic expression

$$e(v_1, v_2, \ldots, v_k)$$

such that $e(\mathcal{D}) = q(\mathcal{D})$ for any document set $\mathcal{D}$

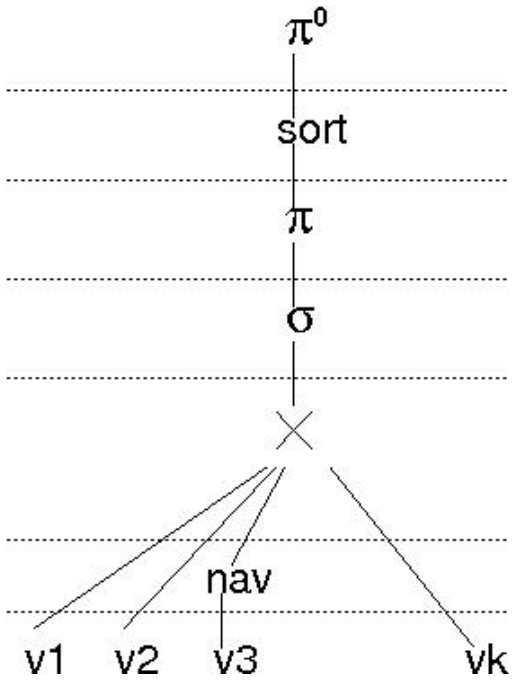
Algebra operators

$scan(v)$	$\times$
$\pi_{cols}(op)$	$\pi^o(op)$
$sort_{cols}(op)$	$\sigma_{cond}(op)$

$nav_{i,np}(op)$ evaluates np over the *cont* attribute $op.i$

Canonical rewritings

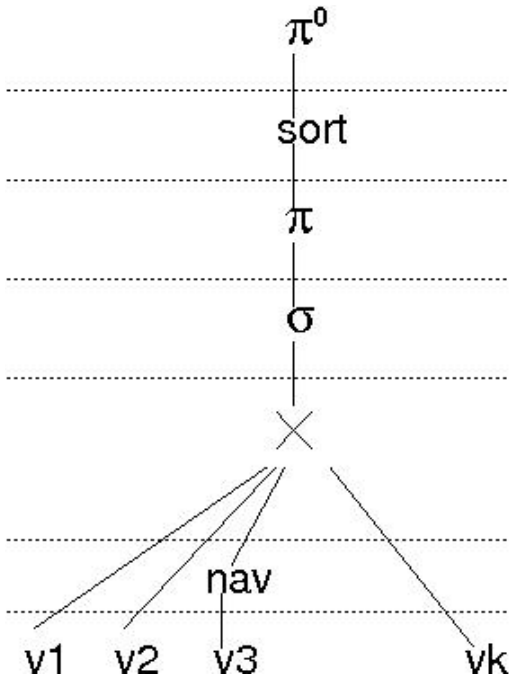
Canonical expression:



We are interested in canonical rewritings only

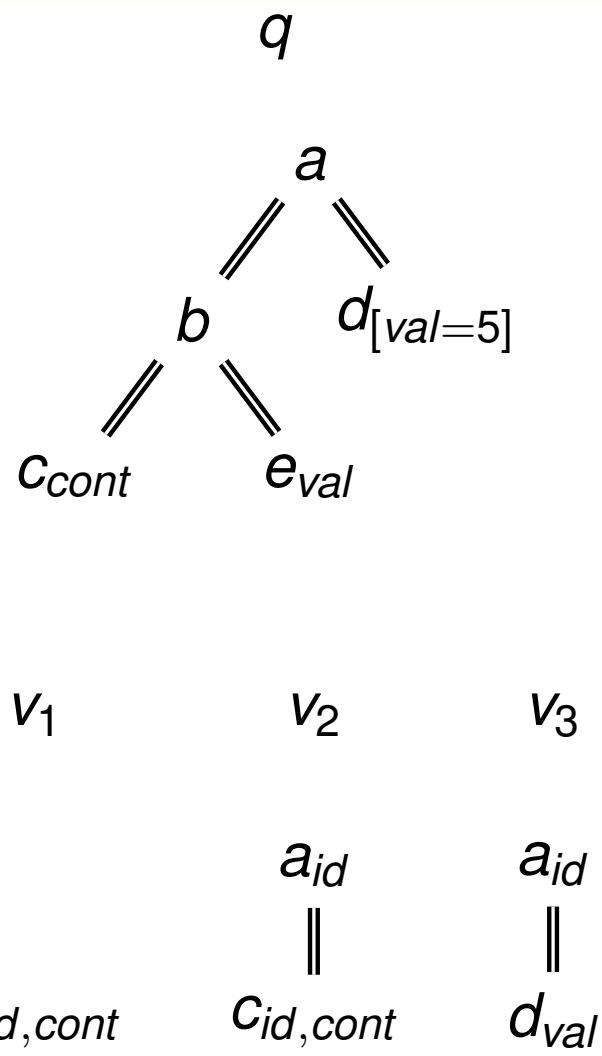
Minimal canonical rewritings

Canonical expression:

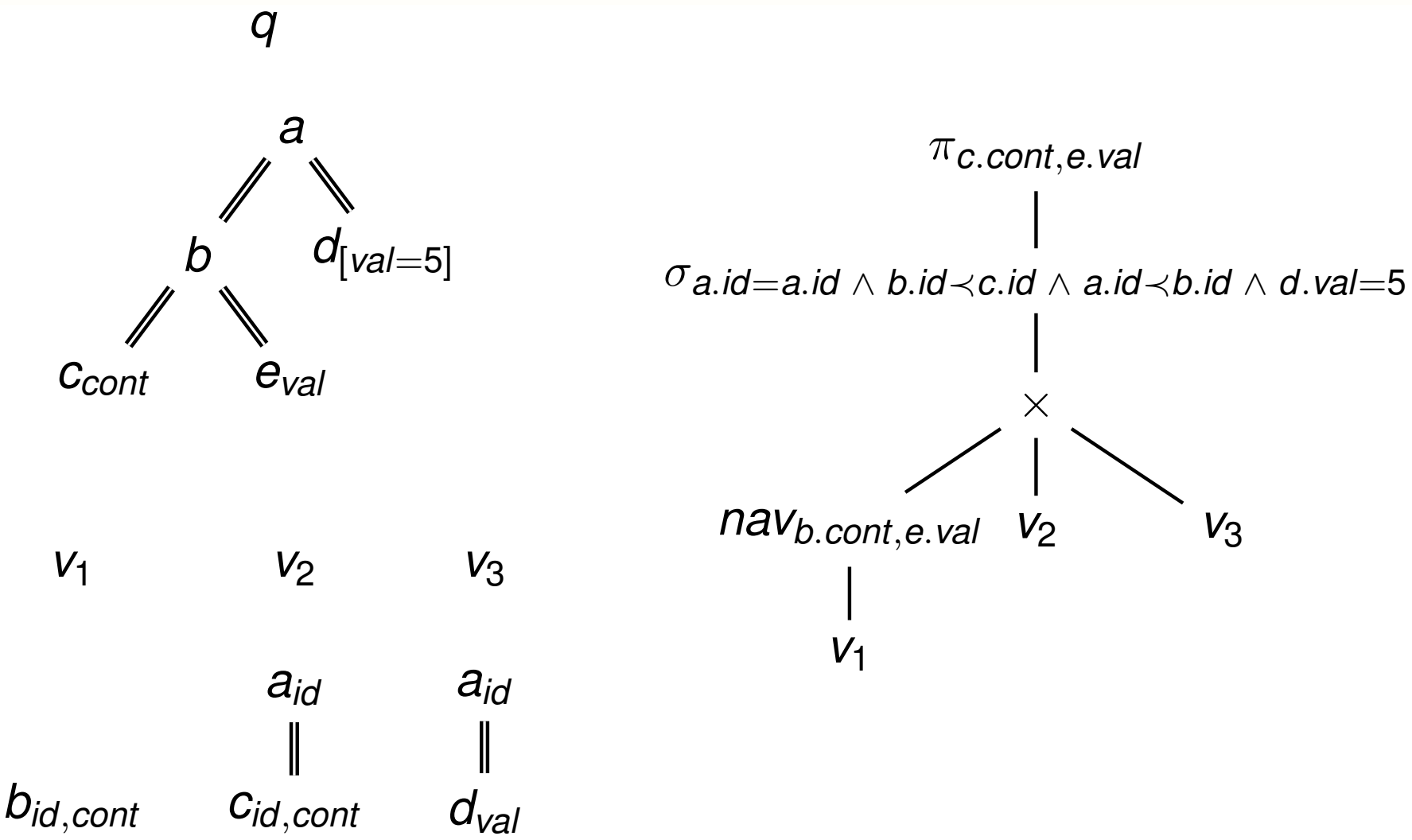


We are interested in minimal canonical rewritings only

Rewriting example



Rewriting example



Finding query rewritings

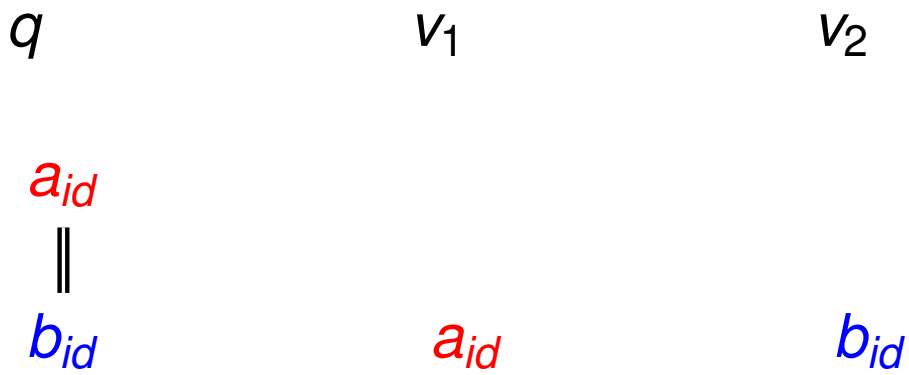
Idea:

Compute covers of the query nodes with the view nodes.

Finding query rewritings

Idea:

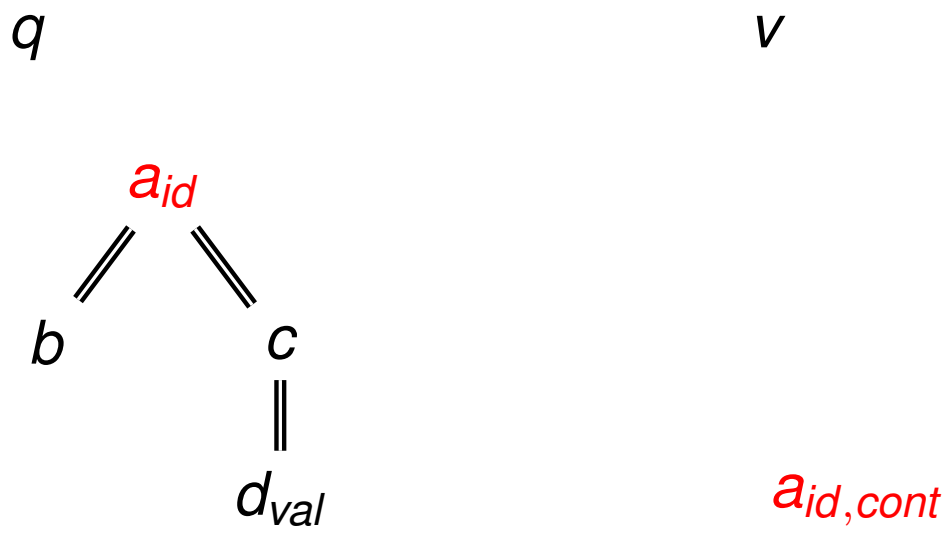
Compute covers of the query nodes with the view nodes.



Finding query rewritings

Idea:

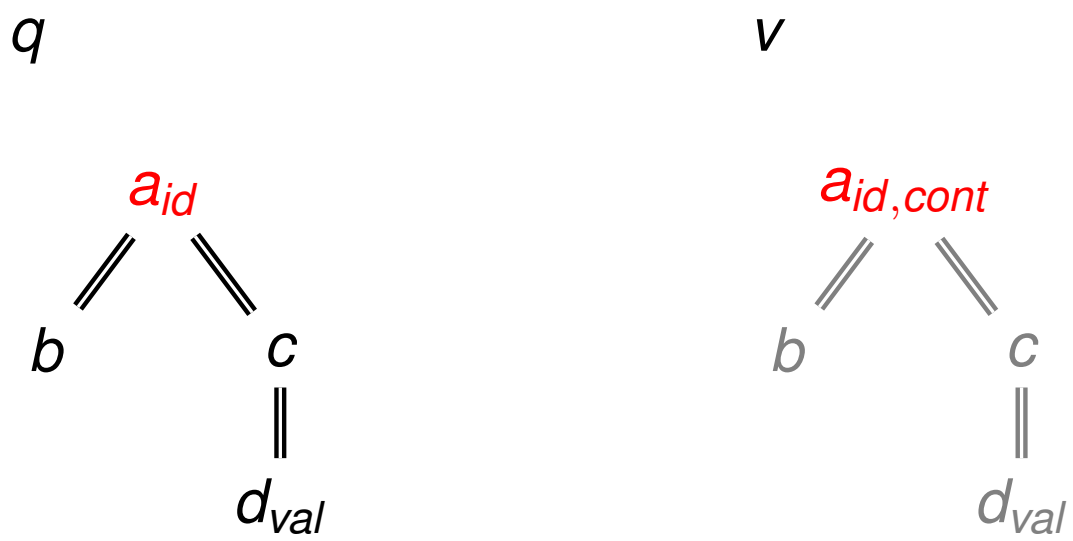
Compute covers of the query nodes with the view nodes.



Finding query rewritings

Idea:

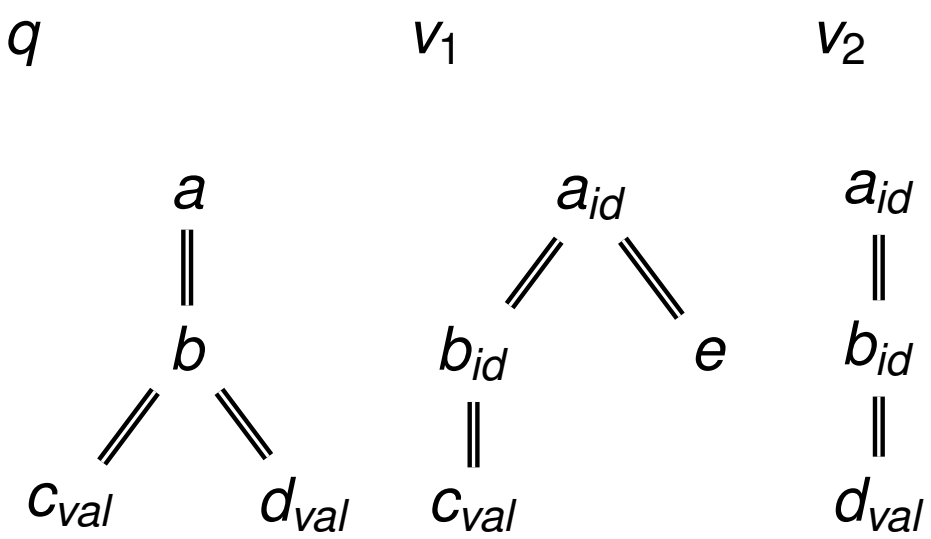
Compute covers of the query nodes with the view nodes.



Finding query rewritings

Idea:

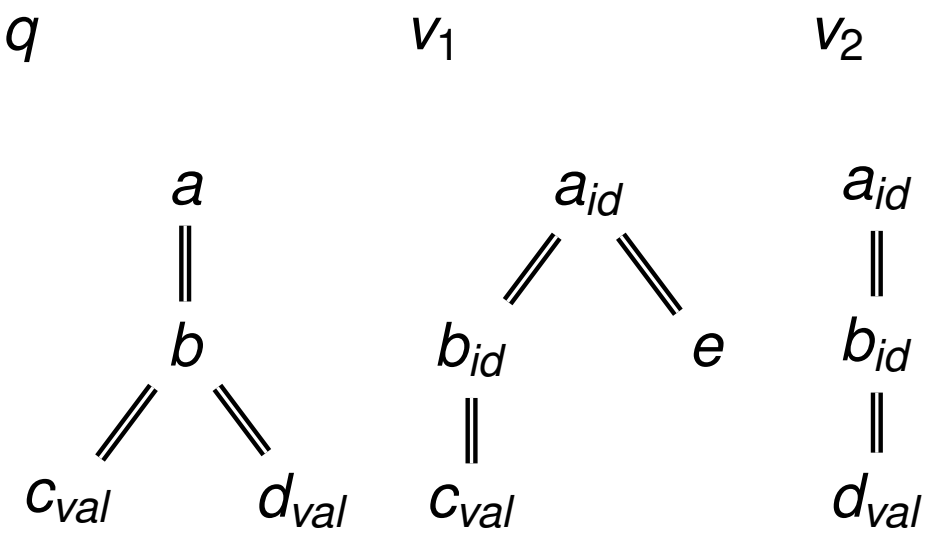
Compute covers of the query nodes with the view nodes.



Finding query rewritings

Idea:

Compute covers of the query nodes with the view nodes.

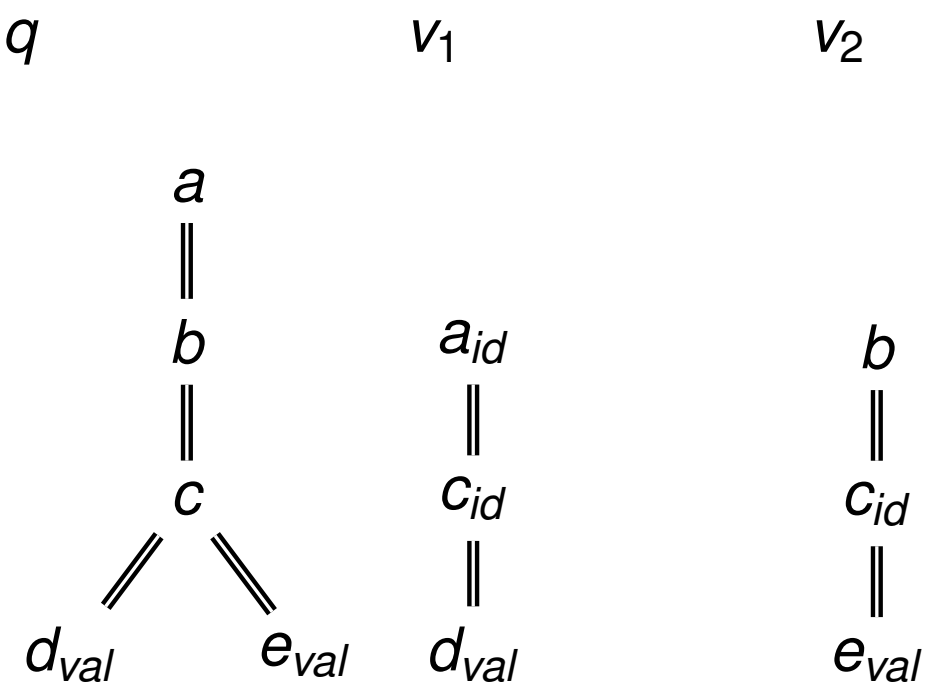


No rewriting

Finding query rewritings

Idea:

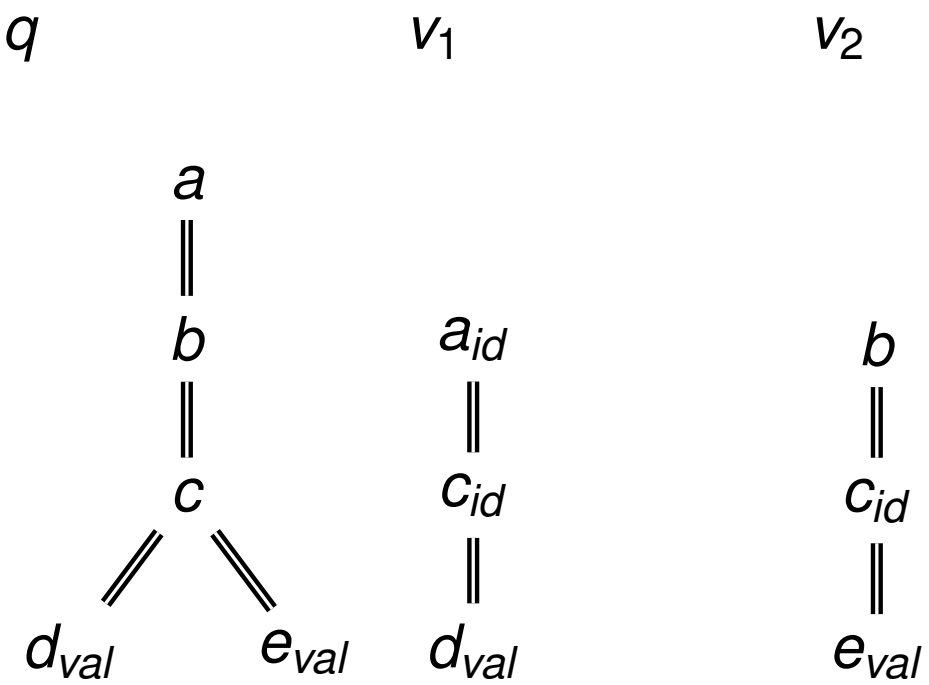
Compute covers of the query nodes with the view nodes.



Finding query rewritings

Idea:

Compute covers of the query nodes with the view nodes.

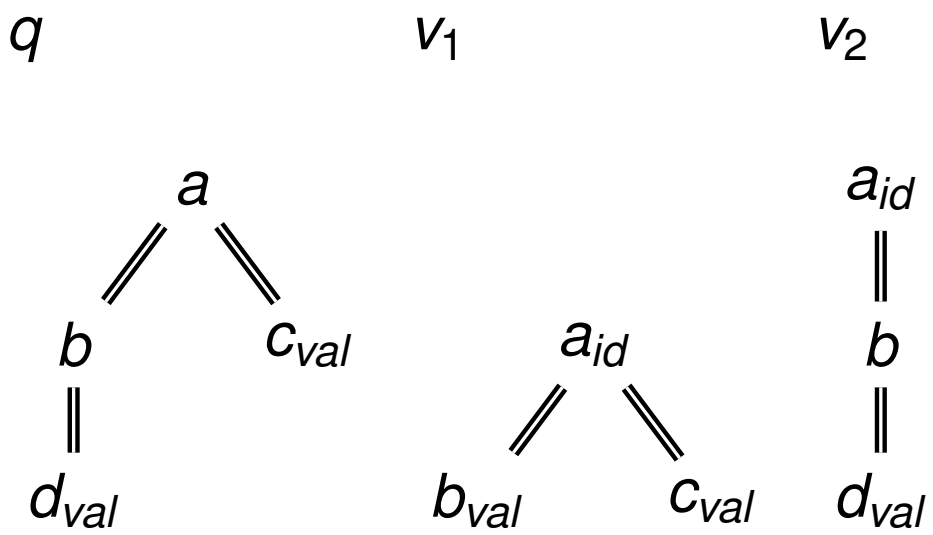


No rewriting

Finding query rewritings

Idea:

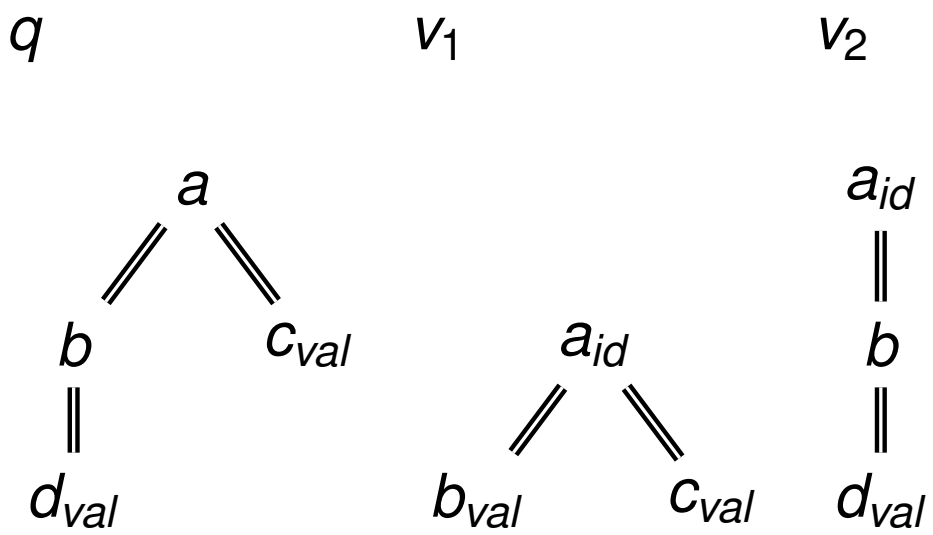
Compute covers of the query nodes with the view nodes.



Finding query rewritings

Idea:

Compute covers of the query nodes with the view nodes.



No rewriting

Rewriting algorithms

SE (Subset Enumeration)

- For each new sebset, check if a rewriting can be found
- Test minimality at the end

ISE (Increasing Subset Enumeration)

- Like SE but enumerates sets from the smallest to the largest
- Finds minimal rewritings first

SE and ISE

- Repeat work
- Try all possible subsets

Rewriting algorithms

Bottom-up algorithms: use smaller partial rewritings to build bigger ones

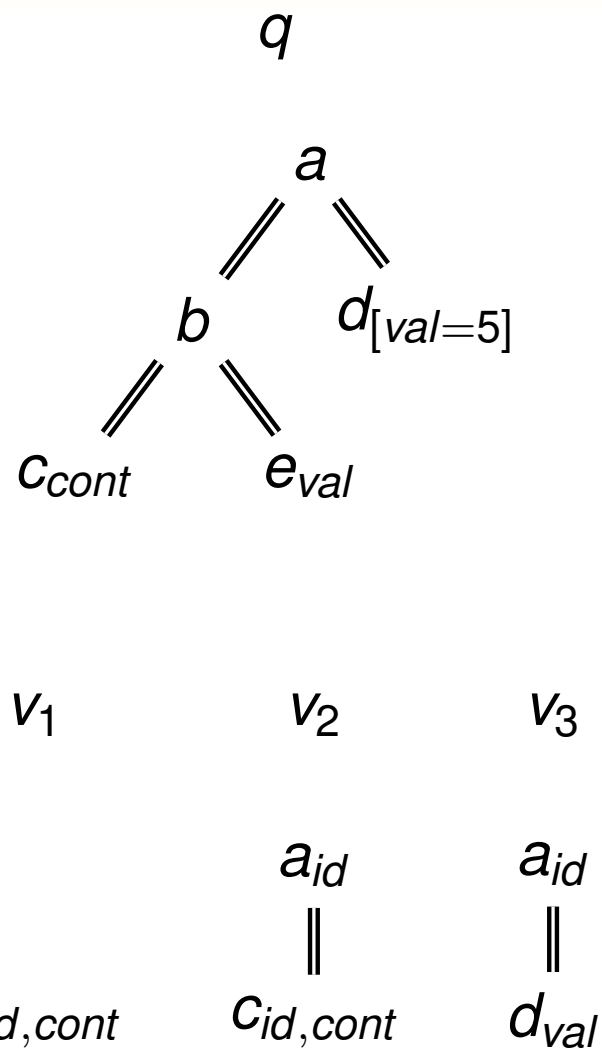
DPR (Dynamic Programming Rewriting)

- Dynamic programming style

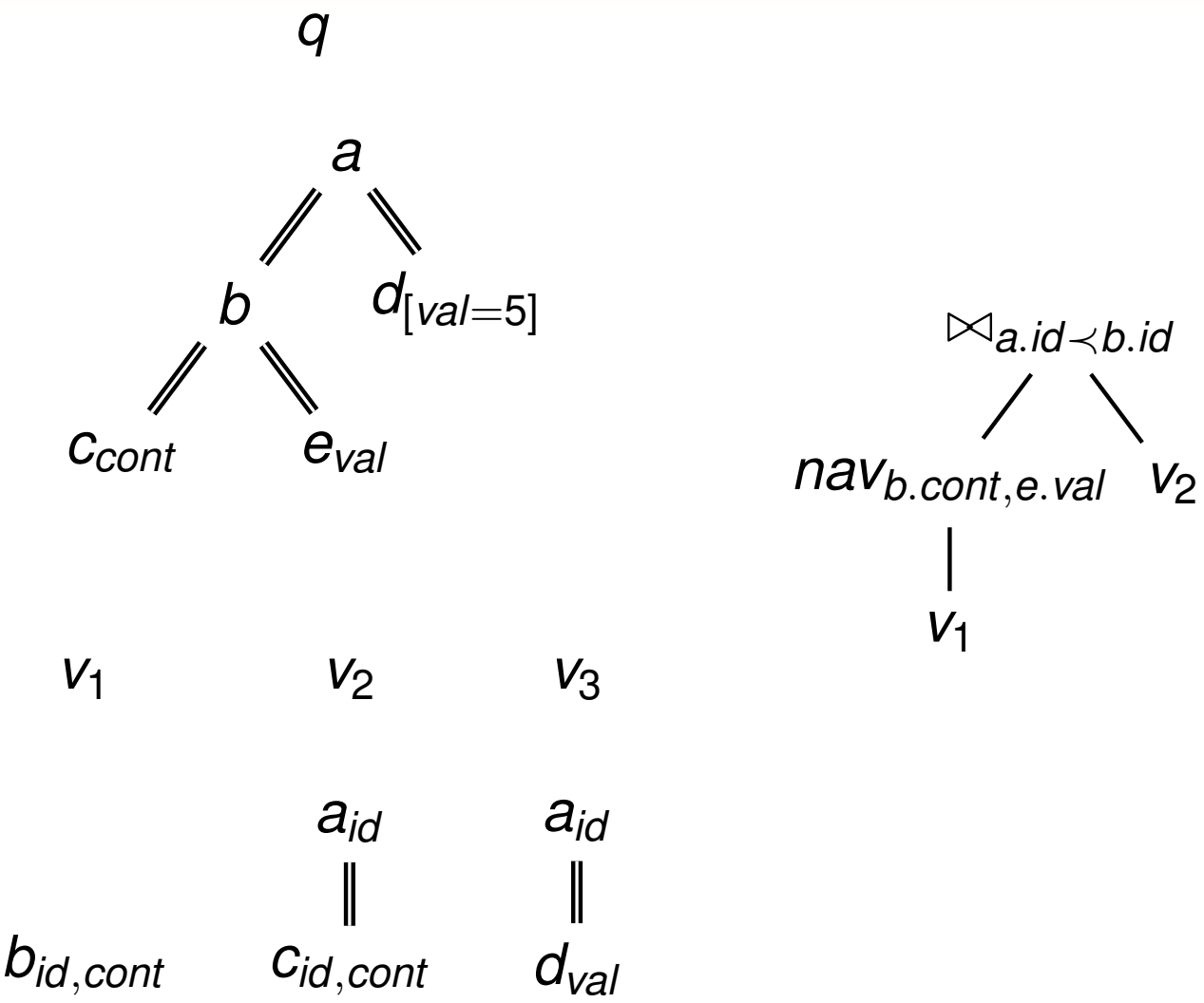
DFR (Depth First Rewriting)

- Greedy based on the biggest query coverage

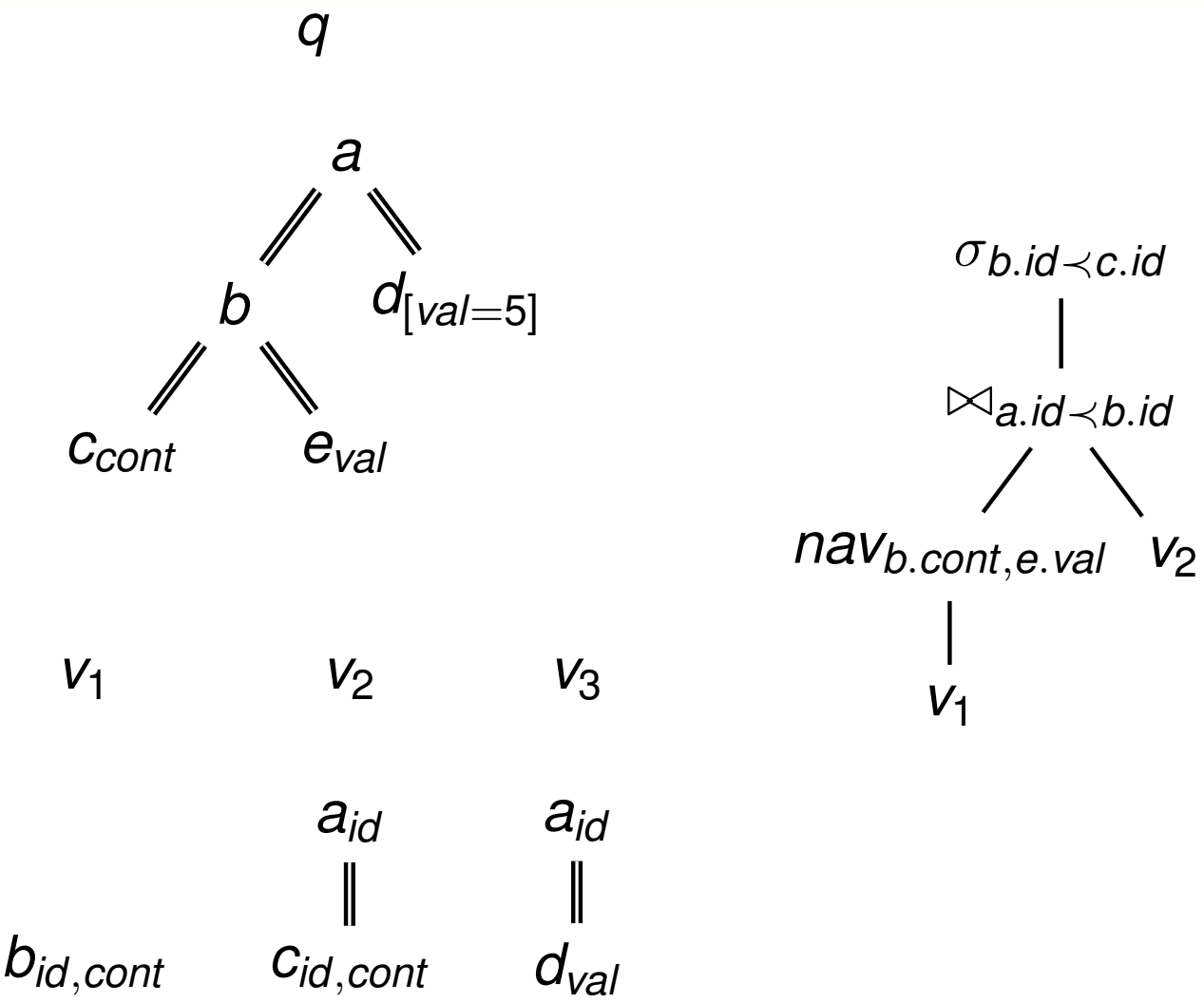
Bottom-up rewriting



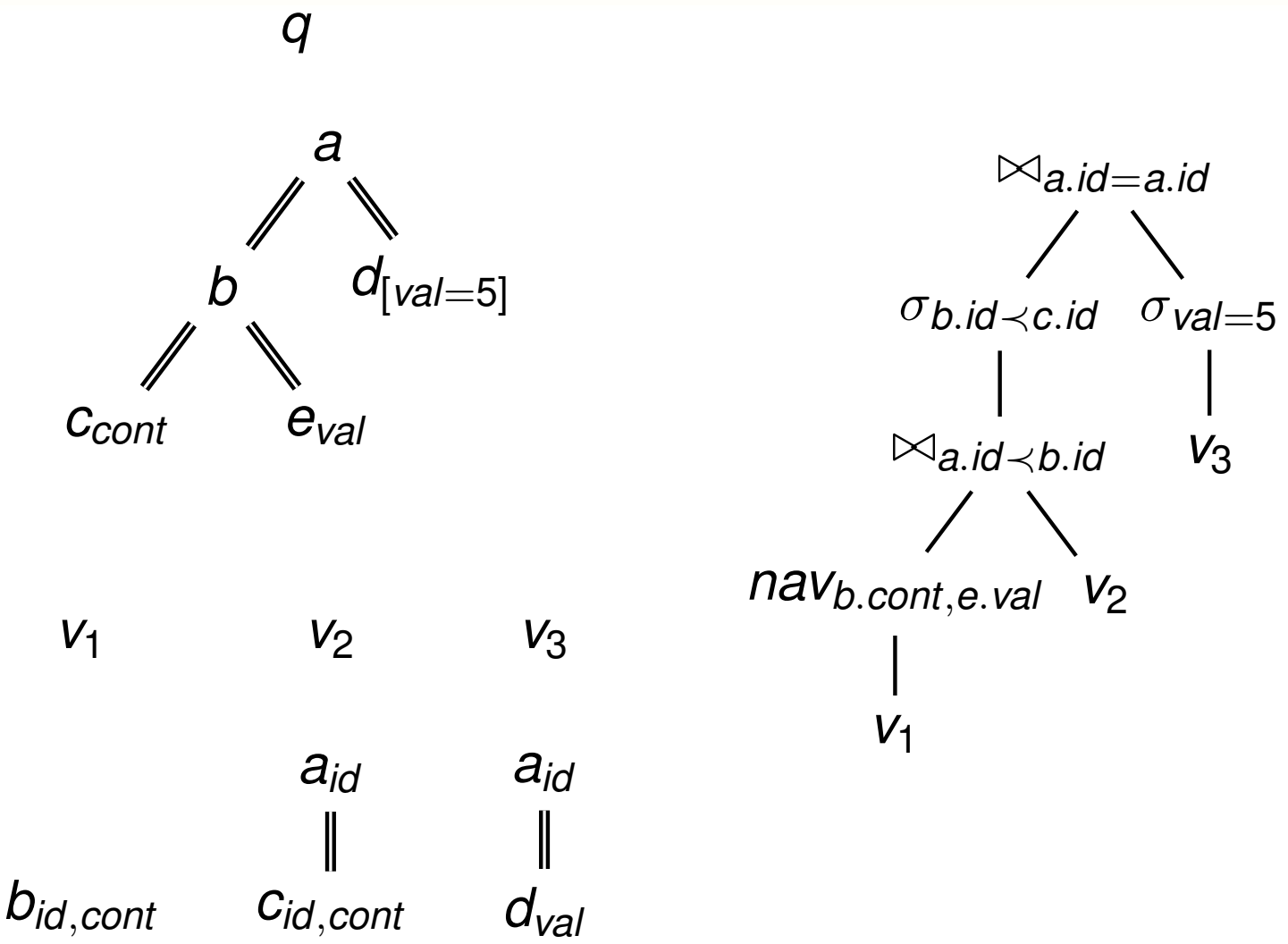
Bottom-up rewriting



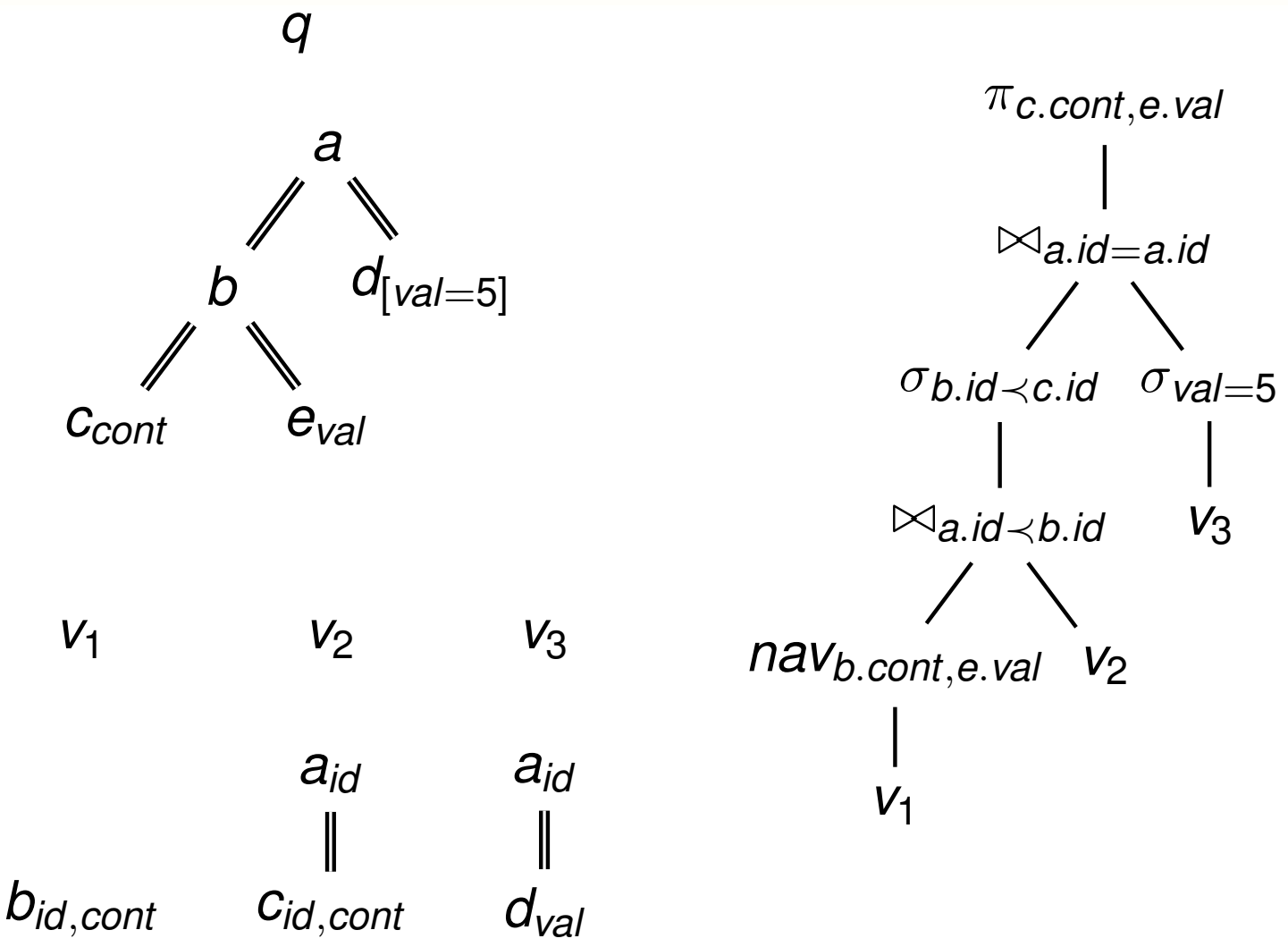
Bottom-up rewriting



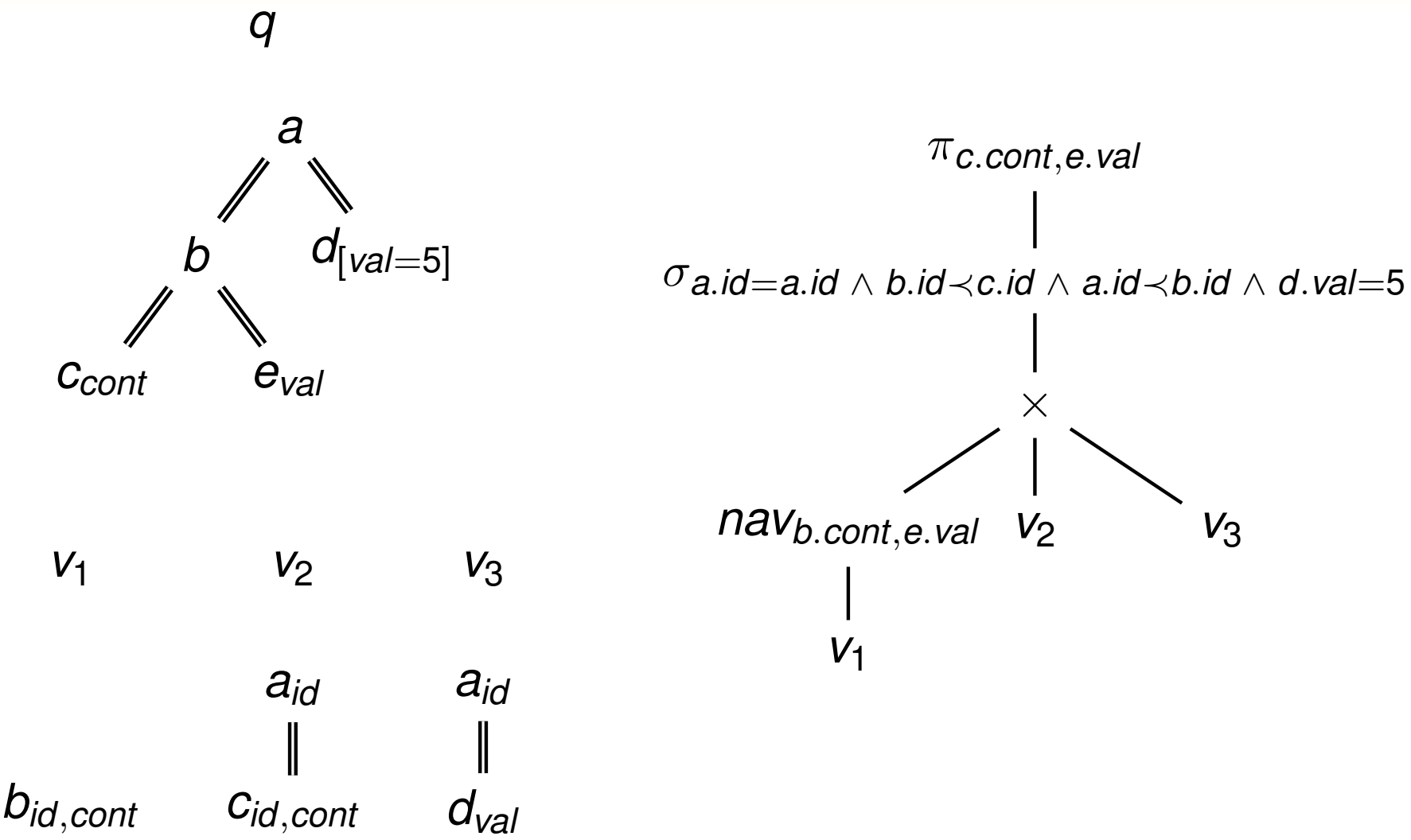
Bottom-up rewriting



Bottom-up rewriting



Bottom-up rewriting



Rewriting algorithms trade-offs

SE, ISE, DPR and DFR are correct and complete.
They produce all minimal canonical rewritings of q given $\mathcal{V}$.

- Which rewritings are ”good”?
  - The one which leads to the best physical plan
  - We learn this too late!
- Heuristic: a good rewriting uses the **smallest number** of views.
  - DFR typically finds fast a solution which is reasonably good
  - ISE, DPR will need more time but return better quality results. They produce rewritings towards the end of the search

View materialization

- Peer p has a view v, peer p_d publishes a document d
- p indexes v on the DHT by the labels of the view
- p_d looks up the labels and keywords of d
  - View set S_a
  - All views v such that $v(d) \neq \emptyset$ are is S_a
- p_d evaluates $v(d)$ for each $v \in S_a$

View indexing and lookup for query rewriting

Query q asked at peer $p \Rightarrow p$ needs to find useful views

4 different strategies

- **Label indexing** (LI):
  - Index v by each v node label
  - Look up by all node labels of q
- **Return label indexing** (RLI):
  - Index v by the labels of all v nodes which project some attributes
  - Same as for LI: use the labels of all q nodes as lookup keys

Indexing and lookup view definitions

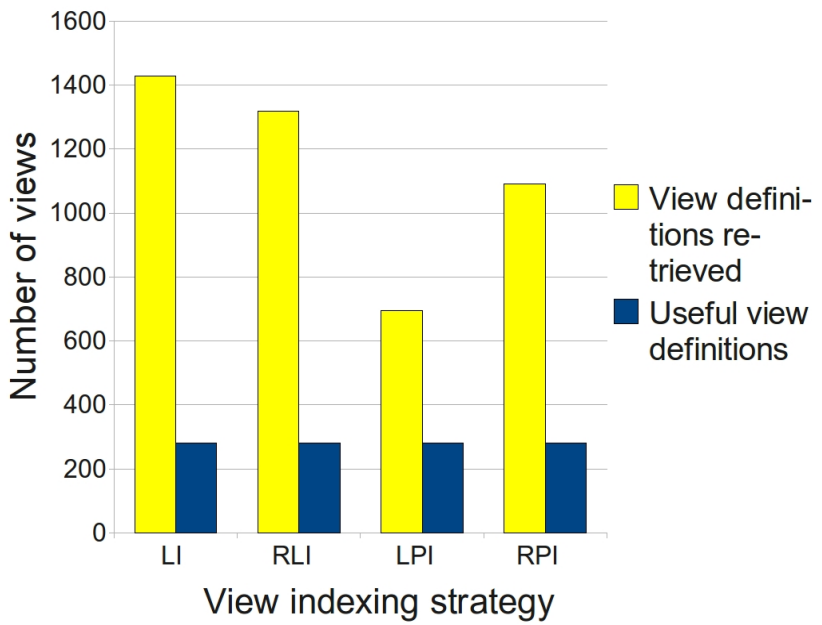
- **Leaf path indexing** (LPI):
  - Let $LP(v)$ be the set of all the distinct root-to-leaf label paths of v. Index v using each element of $LP(v)$ as key
  - Look up details in the paper
- **Return Path Indexing** (RPI):
  - Let $RP(v)$ be the set of all rooted paths in v which end in a node that returns some attribute. Index v using each element of $LP(v)$ as key
  - Look up is the same as for LPI

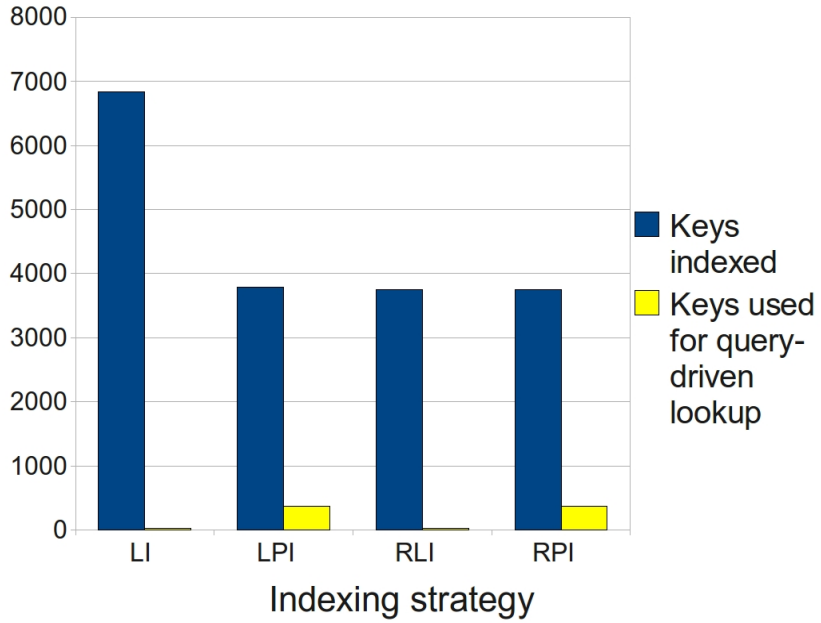
ViP2P platform

- Fully implemented using Java 6
- Used Berkeley DB (version 3.3.75) to store view data
- Used FreePastry (version 2.1) as our DHT network
- Experiments carried on *Grid5000* using **250 machines**
- **1000 ViP2P peers** were deployed

View look up performance

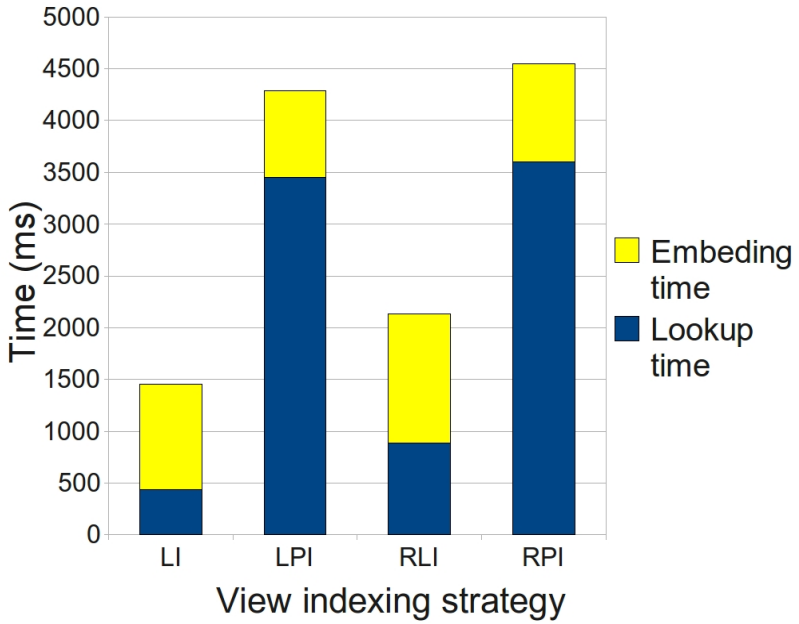
We used 1440 views related to but different from query *q*





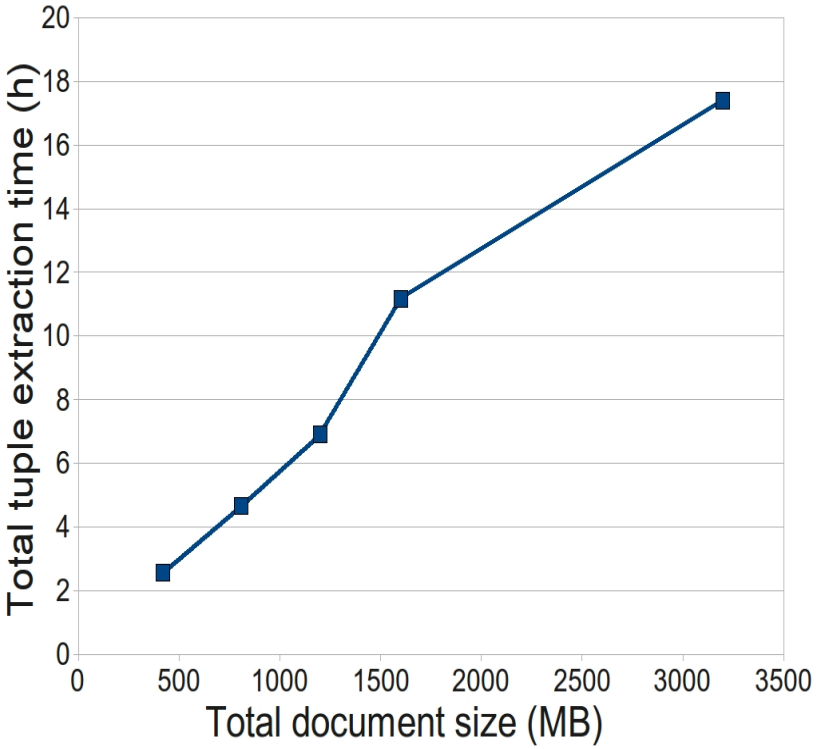
View look up performance

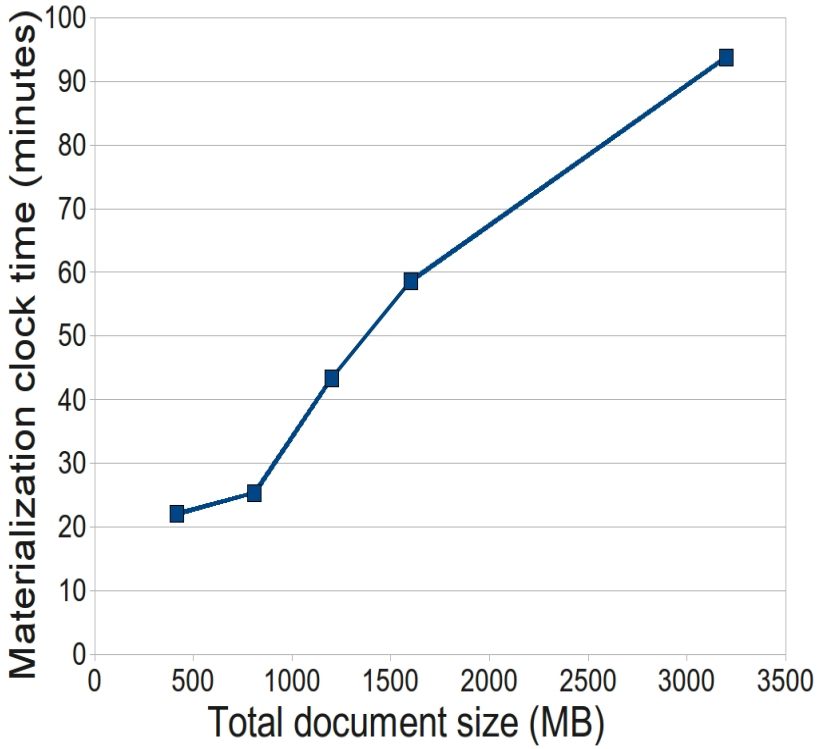
We used 1440 views related to but different from query *q*



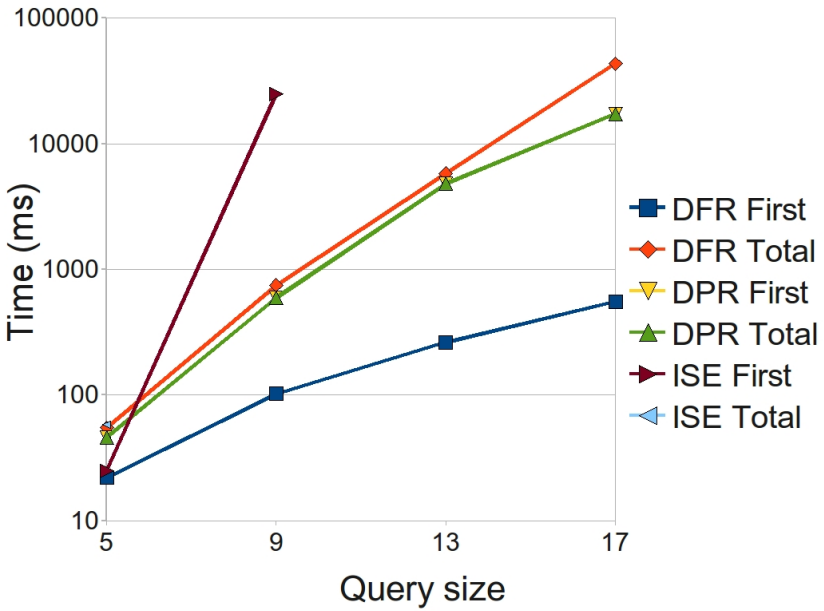
View building

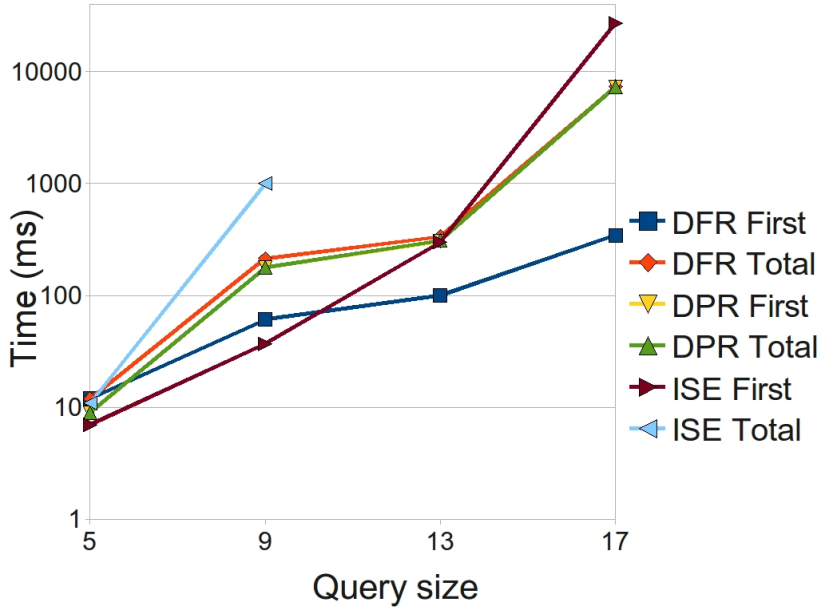
For the experiments we indexed 2000 XMark documents and 500 views (70 related to the documents)



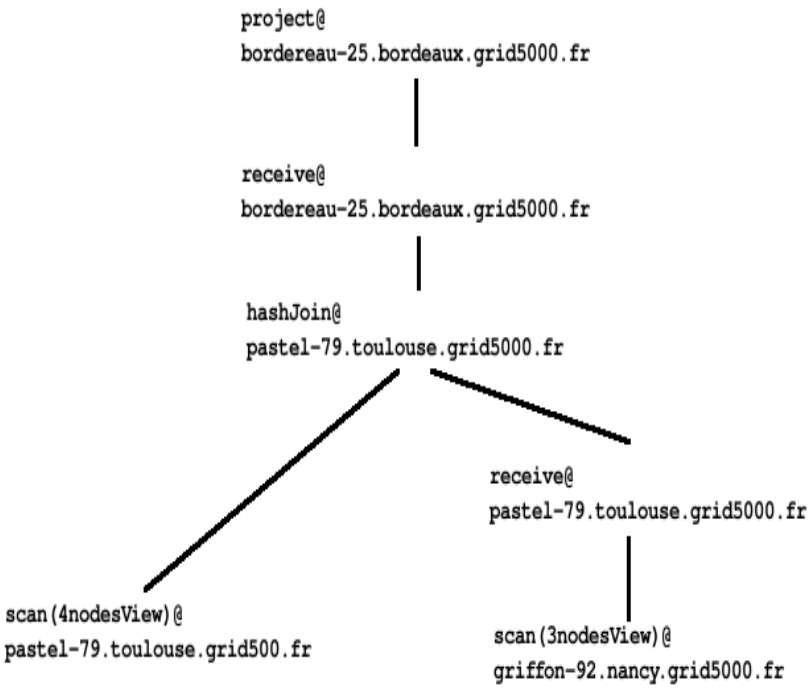


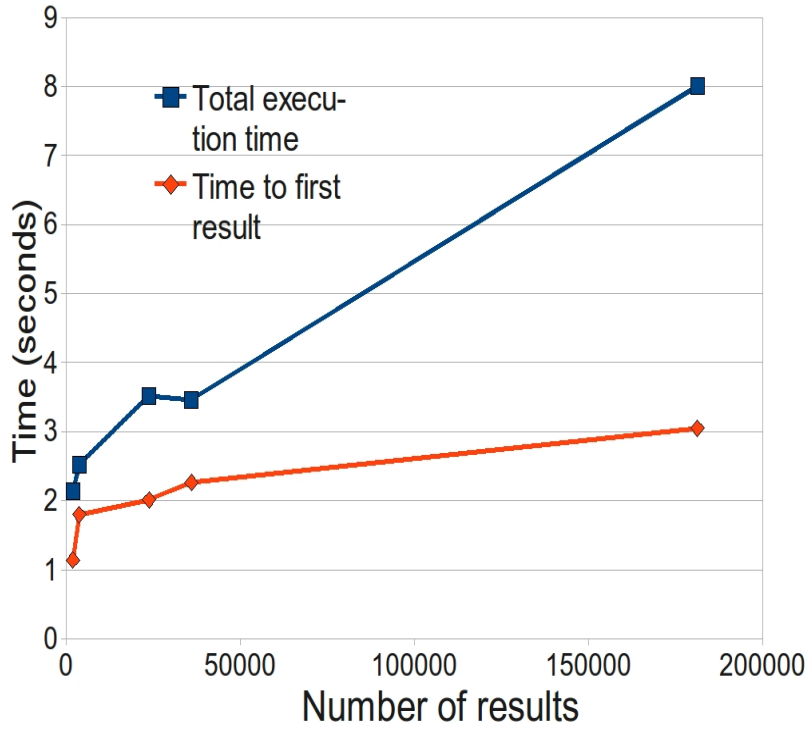
Performance of rewriting algorithms





Query execution





Related work

Indexing documents in the DHT [GWJD03, BC06, SHA05, AMP+08]

XPath query rewriting [BOB+04, XO05, CDO08, TYÖ+08]

- XPath: wildcard *, union
- Rewritings: intersection, navigations, joins

Rewriting with structural constrains [ABMP07]

- Centralized setting
- Dataguide [GW97] constraints

Summing up

- ViP2P: data access support structures for DHT based XML data management
- All the presented algorithms have been fully implemented in a functional Java based platform
- Presented at DataX 2009 (no proceedings)
- Extended version submitted for publication
- Visit us at vip2p.saclay.inria.fr!

Thank you!

[ABMP07] **Andrei Arion, Véronique Benzaken, Ioana Manolescu, and Yannis Papakonstantinou.**
Structured materialized views for XML queries.
In *VLDB*, pages 87–98, 2007.

[AMP+08] **S. Abiteboul, I. Manolescu, N. Polyzotis, N. Preda, and C. Sun.**
XML processing in DHT networks.
In *ICDE*, 2008.

[BC06] **Angela Bonifati and Alfredo Cuzzocrea.**
Storing and retrieving XPath fragments in structured P2P networks.
Data Knowl. Eng., 59(2), 2006.

[BOB+04] **A. Balmin, F. Ozcan, K. Beyer, R. Cochrane, and H. Pirahesh.**
A framework for using materialized XPath views in XML query processing.
In *VLDB*, 2004.

[CDO08] **Bogdan Cautis, Alin Deutsch, and Nicola Onose.**
Xpath rewriting using multiple views: Achieving completeness and efficiency.
In *WebDB*, 2008.

[GW97] **Roy Goldman and Jennifer Widom.**
Dataguides: Enabling query formulation and optimization in semistructured databases.
In *VLDB*, 1997.

[GWJD03] **L. Galanis, Y. Wang, S.R. Jeffery, and D.J. DeWitt.**
Locating data sources in large distributed systems.
In *VLDB*, 2003.

[SHA05] **Gleb Skobeltsyn, Manfred Hauswirth, and Karl Aberer.**
Efficient processing of XPath queries with structured overlay networks.

In *OTM Conferences (2)*, 2005.

[TYÖ+08] **Nan Tang, Jeffrey Xu Yu, M. Tamer Özsu, Byron Choi, and Kam-Fai Wong.**
Multiple materialized view selection for xpath query rewriting.
In *ICDE*, pages 873–882, 2008.

[XO05] **W. Xu and M. Ozsoyoglu.**
Rewriting XPath queries using materialized views.
In *VLDB*, 2005.