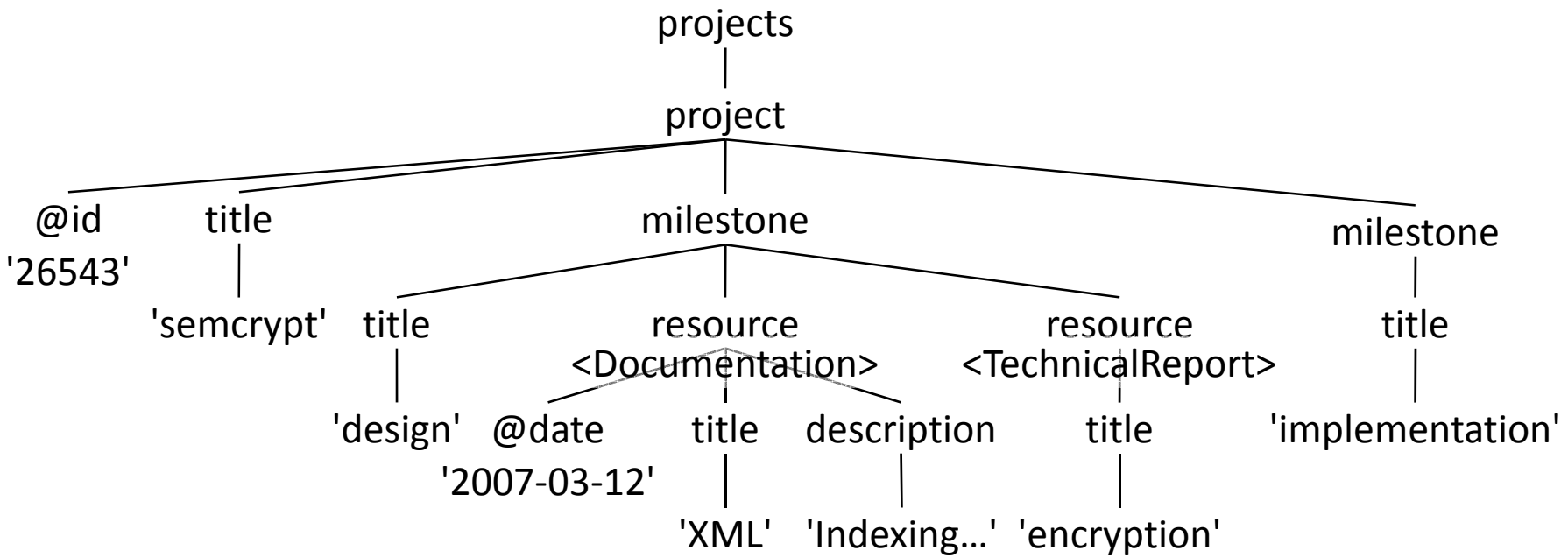
Flexible and Selective Indexing in XML Databases

Katharina Grün

26.09.2008

Motivation

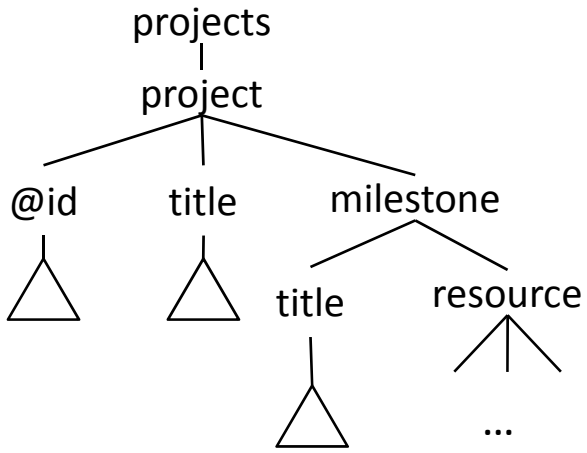
- Widespread use of XML
- XML databases for efficient query and update processing
- Require secondary indices that support arbitrary query workloads
  - Flexibility
    - index content and/or structure
    - support various operations
  - Selectivity
    - index frequently queried document fragments
    - smaller index size, faster index traversal

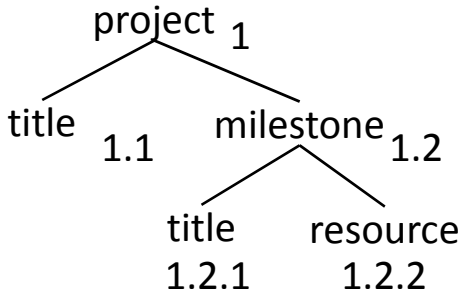


Q1	/projects/project/milestone/title	labelpath
Q2	//title[. = 'XML']	name, exact value
Q3	//resource[@date ≥ '2007-01-01' and @date < '2008-01-01']	name, value range
Q4	//project[@id = '26543']/milestone[title = 'design'] /resource[@date ≥ '2007-01-01']	path, value range
Q5	//element(resource, Report)[description/fn:matches (., '\bdata.*')]	type hierarchy, text

State of the Art - XML index structures

- Path indices (e.g. DataGuide, Index Fabric)
  - index document structure (labelpaths)
  - nested indices on node values
- Element indices (e.g. XISS)
  - use node labels
  - build indices on node names and values
  - process queries via structural join algorithms
- Not flexible and selective
  - index entire documents
  - only support some queries efficiently
  - proprietary processing techniques





milestone	1.2,…
title	1.1, 1.2.1

State of the Art - Index Processing

- Extensible indexing
  - generic index template: GiST
  - index interfaces: Oracle data cartridges, DB2 database extenders,…
- XML databases (e.g. eXist)
  - index nodes by name and/or value (e.g. all date values)
  - no support for navigation (e.g. date and title values)
- KeyX
  - index model based on path expressions
  - index selection, maintenance (poor performance)
- XML Access Modules (XAMs)
  - storage (index) model based on tree patterns
  - index selection

SCIENS

- Structure and Content Indexing with Extensible, Nestable Structures

- Flexibility
  - How can the database represent and compare indexed properties and nodes?
  - Which index structures are necessary?
- Selectivity
  - How can the database represent and process arbitrary indices?
  - How can the database update arbitrary indices when documents change?

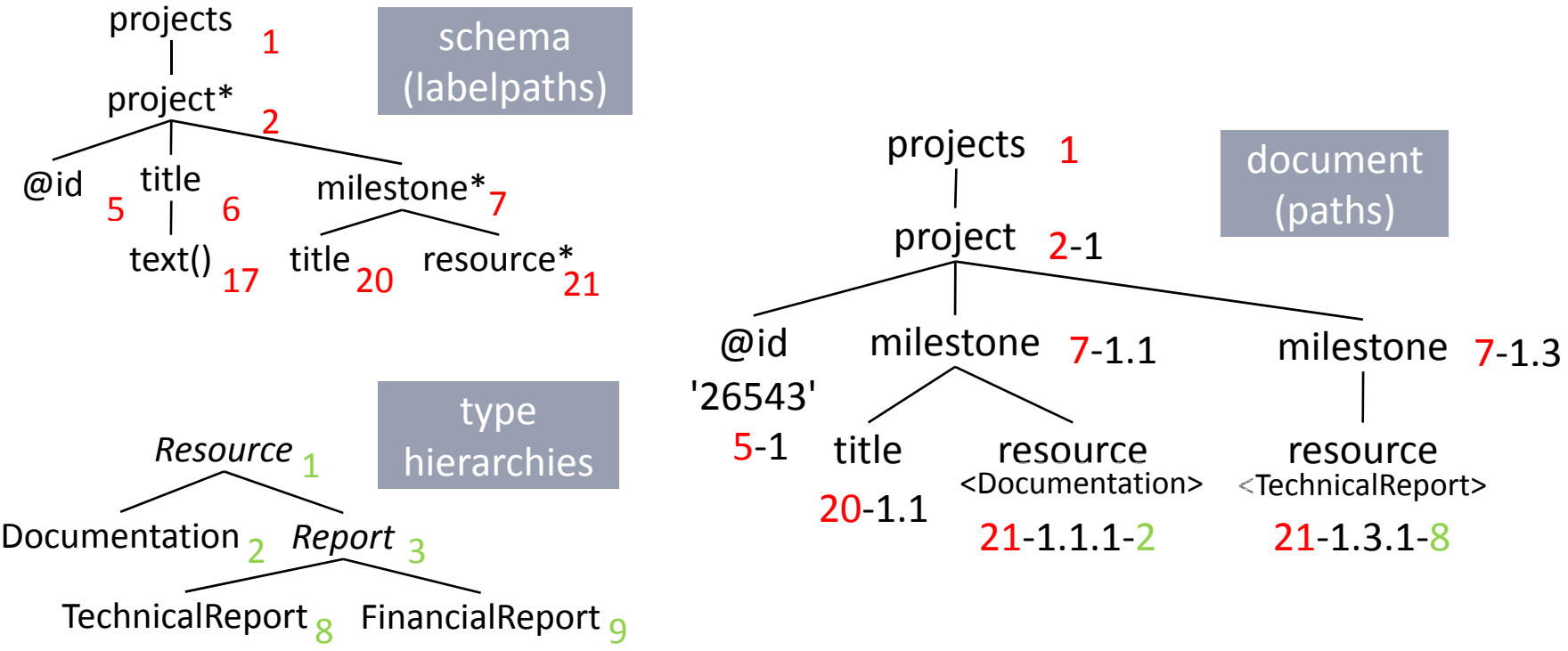
labeling scheme

extensible, nestable index structures

index model and framework

maintenance algorithm

Labeling Scheme



labelpath → schema label
path → instance label
type → type label

schema label, instance label, type label → static k-ary labeling
schema label, instance label, type label → dynamic Ordpath prefix labeling

static k-ary labeling, dynamic Ordpath prefix labeling → document order, structural relationships

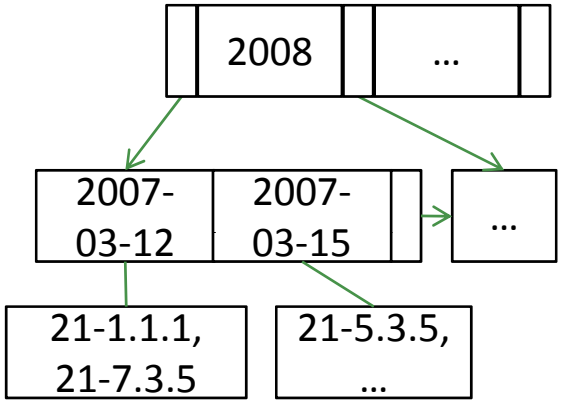
Index Structures

- Index
  - consists of a set of index entries
  - index entry maps index keys to nodes
    - key is value, name, path, labelpath or type
    - node returned is represented by label
    - node returned may differ from node indexed

key	return
2007-03-12	21-1.1.1
2007-03-15	21-5.3.5

(date) (resource)

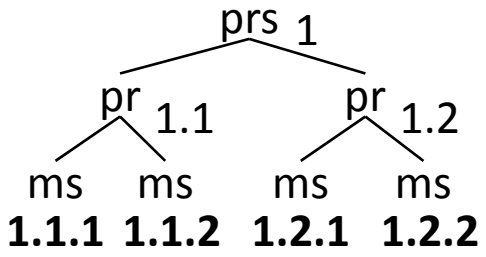
- Index structures
  - hash table, prefix B+-tree and KDB-tree
  - extensible by binding comparison operators (=, <) to indexed property
  - nestable



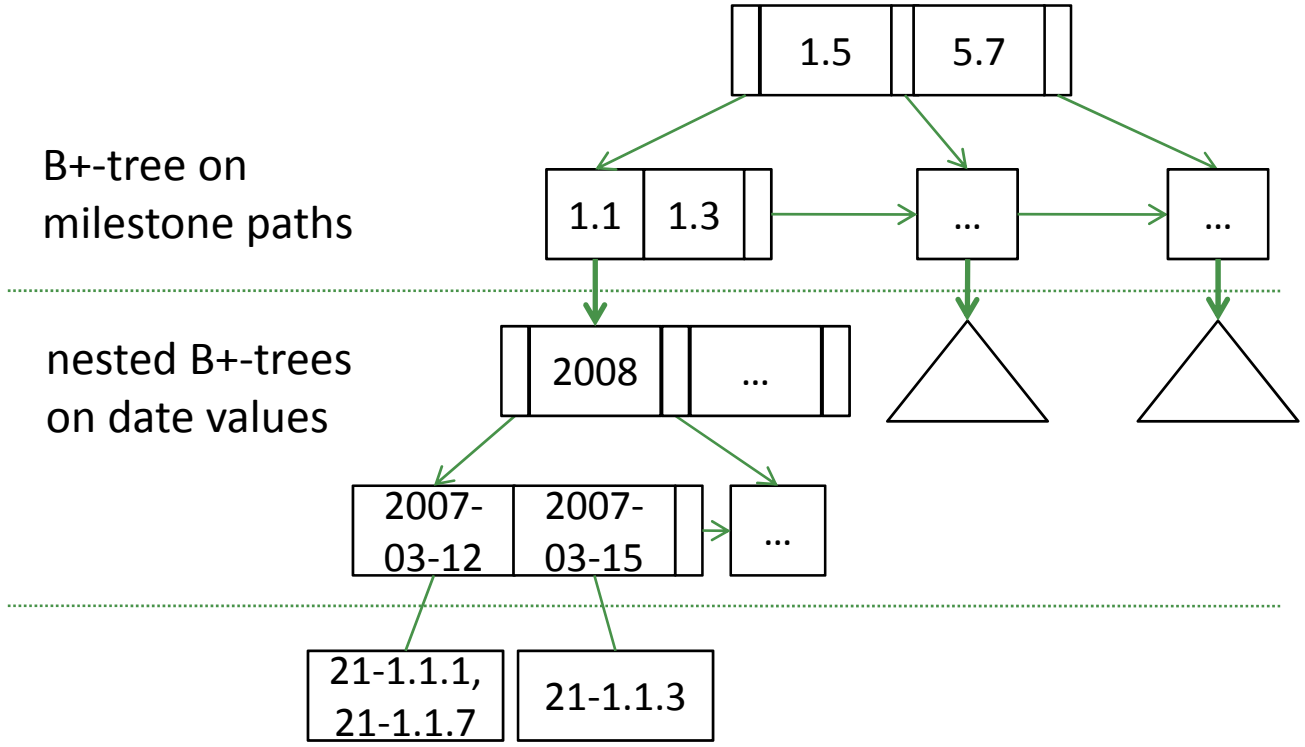
Index Structures - Extensibility

	operator		example	hash table	prefix B+-tree	KDB-tree
content	exact	=	k = 2003	✓	✓	✓
	range	=, <[1]	2003 ≤ k ≤ 2005 k ≤ '2007-01-01'		✓	✓
	word[2]	=	k = 'XML'	✓	✓	✓
	word prefix[2]	=, <	data ≤ k ≤ data∞		✓	✓
structure	exact[3]	=	k = 'resource' k = 63 k = 1.1.1	✓	✓	✓
	hierarchy [4]	=[5], <[6]	1.1 ≤ k ≤ 1.1		✓	✓

[1] numeric / lexicographic order
[2] index each word of node value
[3] name, [4] labelpath, path, type
[5] is-a
[6] document order



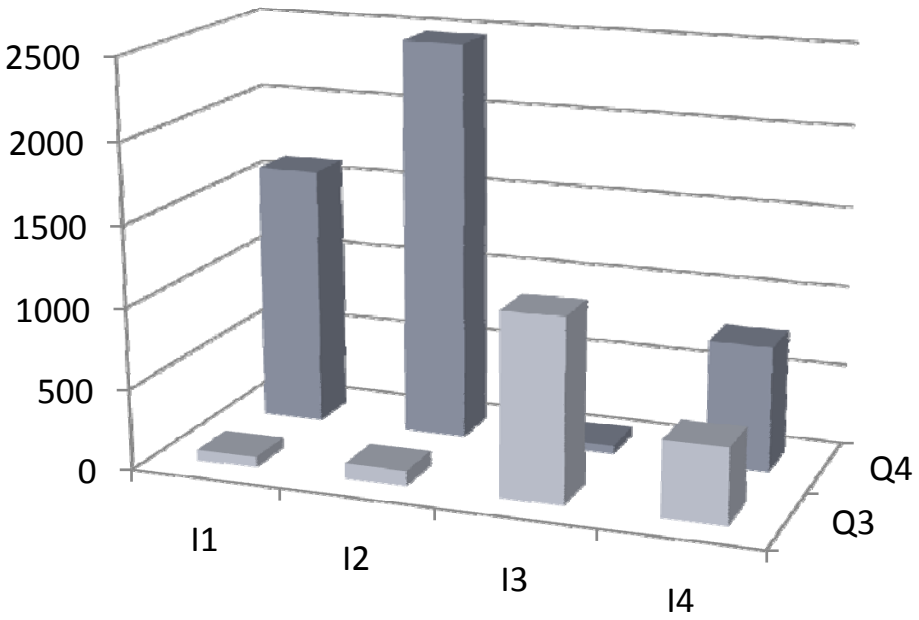
Index Structures – Example



Q3	//resource[@date ≥ '2007-01-01' and @date < '2008-01-01']
Q4	//project[@id = '26543']/milestone[title = 'design']/resource[@date ≥ '2007-01-01']

1.1

Index Structures - Performance

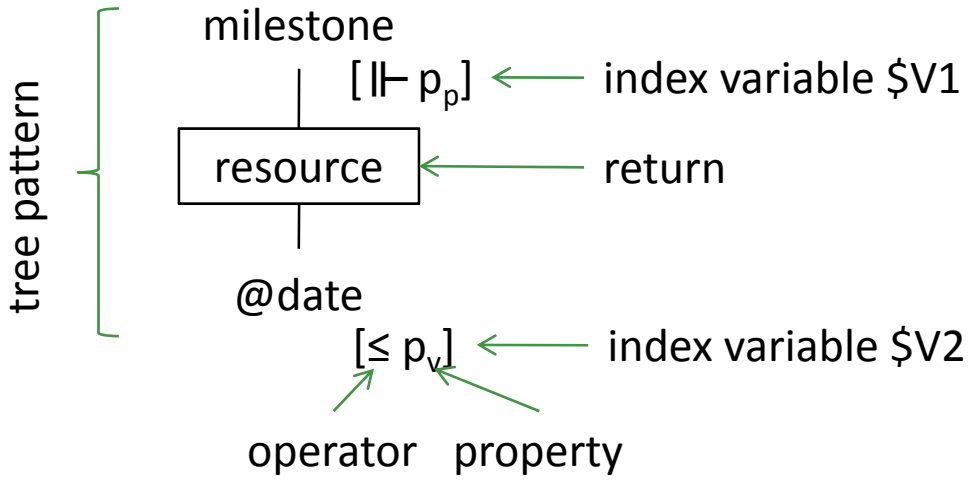


- I1: B+-tree on date values
- I2: B+-tree on date values with nested B+-tree on milestone paths
- I3: B+-tree on milestone paths with nested B+-tree on date values
- I4: KDB-tree on date values and milestone paths

Q3	//resource[@date ≥ '2007-01-01' and @date < '2008-01-01']
Q4	//project[@id = '26543']/milestone[title = 'design']/resource[@date ≥ '2007-01-01']

10MB document, 63 matching resources (same result size), index access time in ms

Index Model



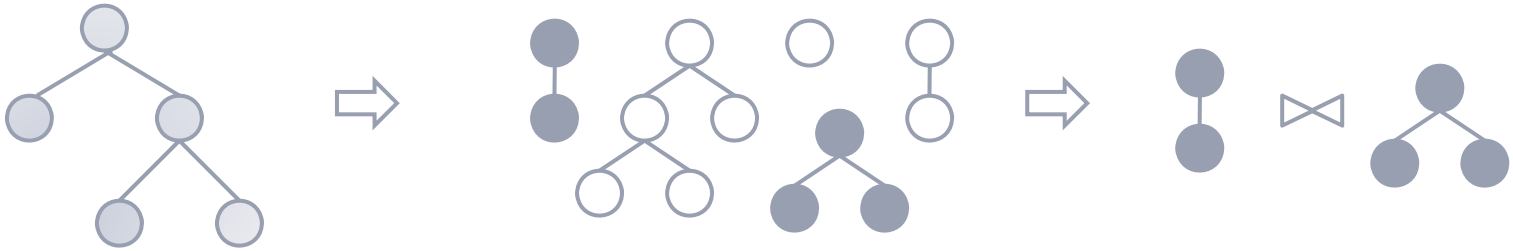
config :
$V1, $V2: KDB-tree or
$V1 : B+-tree, $V2 : B+-tree

search:
$V1 : (= 1.1)
$V2 : (2007-01-01 ≤ ≤ ∞)

- Index pattern: unordered tree pattern defining
  - indexed properties (keys) and supported operations via index variables
  - nodes to be returned
- Index configuration
  - maps index variables to index structures
- Search configuration
  - defines search key and search operator for each index variable

Index Framework

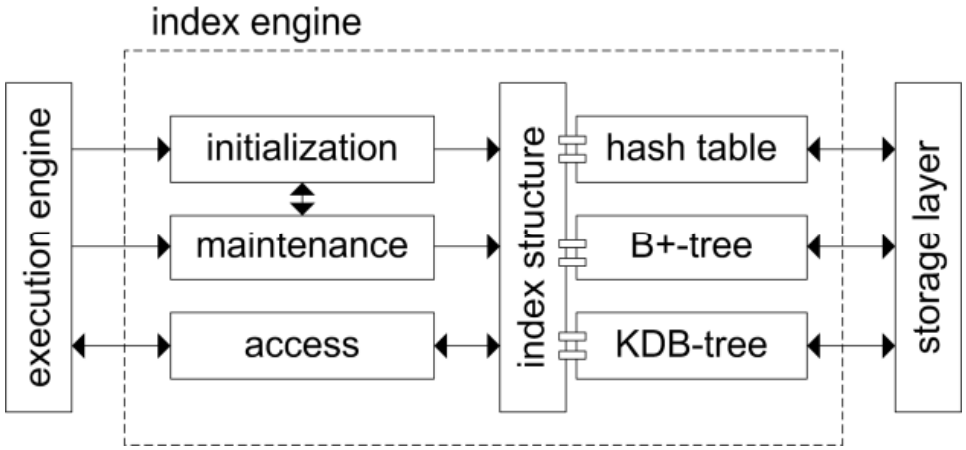
- extend query optimizer
  - index selection based on XAM approach



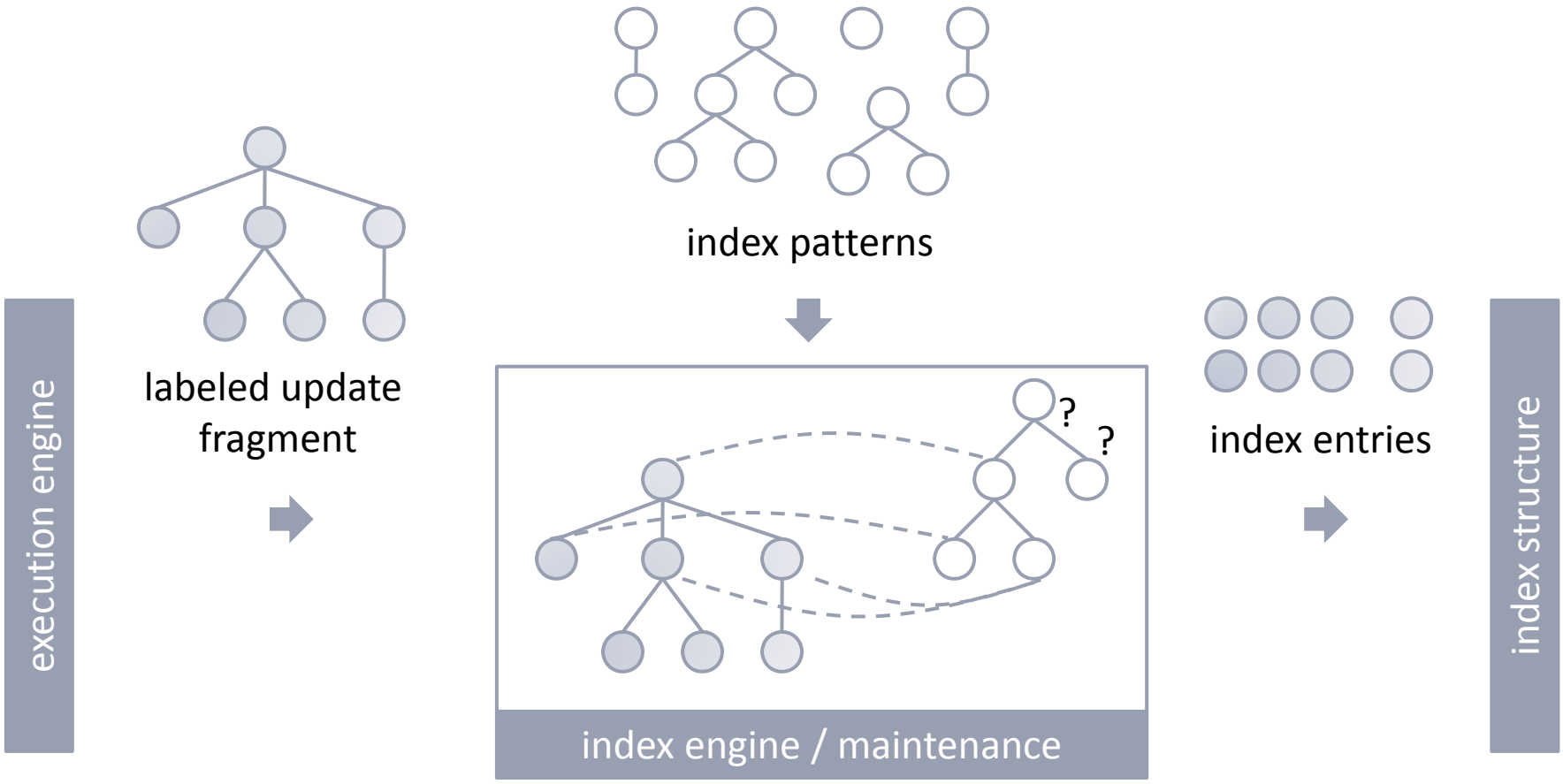
query | index patterns / XAMs | execution plan

- extend execution engine
  - index access
  - index maintenance

➢ process arbitrary indices

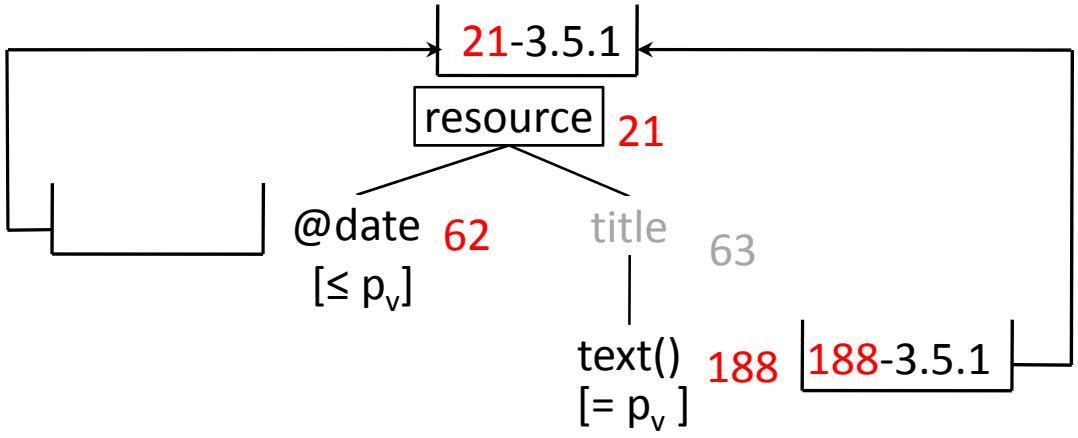


Index Maintenance

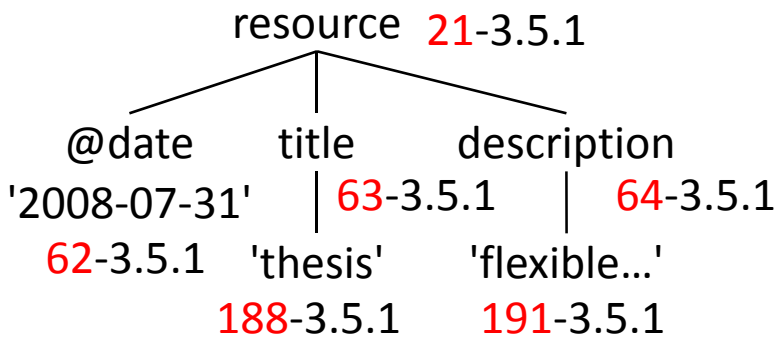


Index Maintenance - Example

index pattern



update fragments

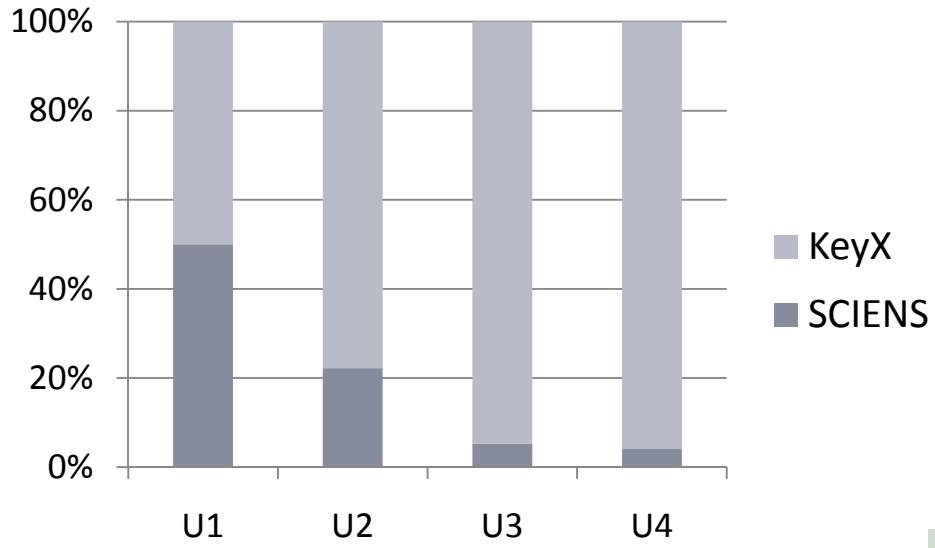


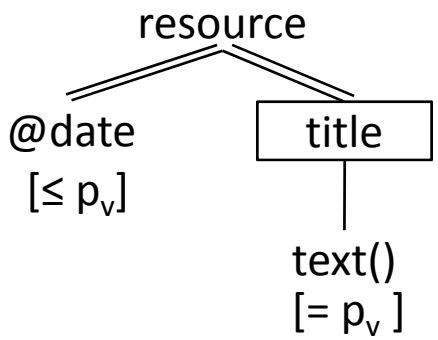
@date
'2008-08-03'
62-3.5.1

2008-07-31	thesis	21-3.5.1

~~2008-07-31~~ 2008-08-03	thesis	21-3.5.1

Index Maintenance - Performance



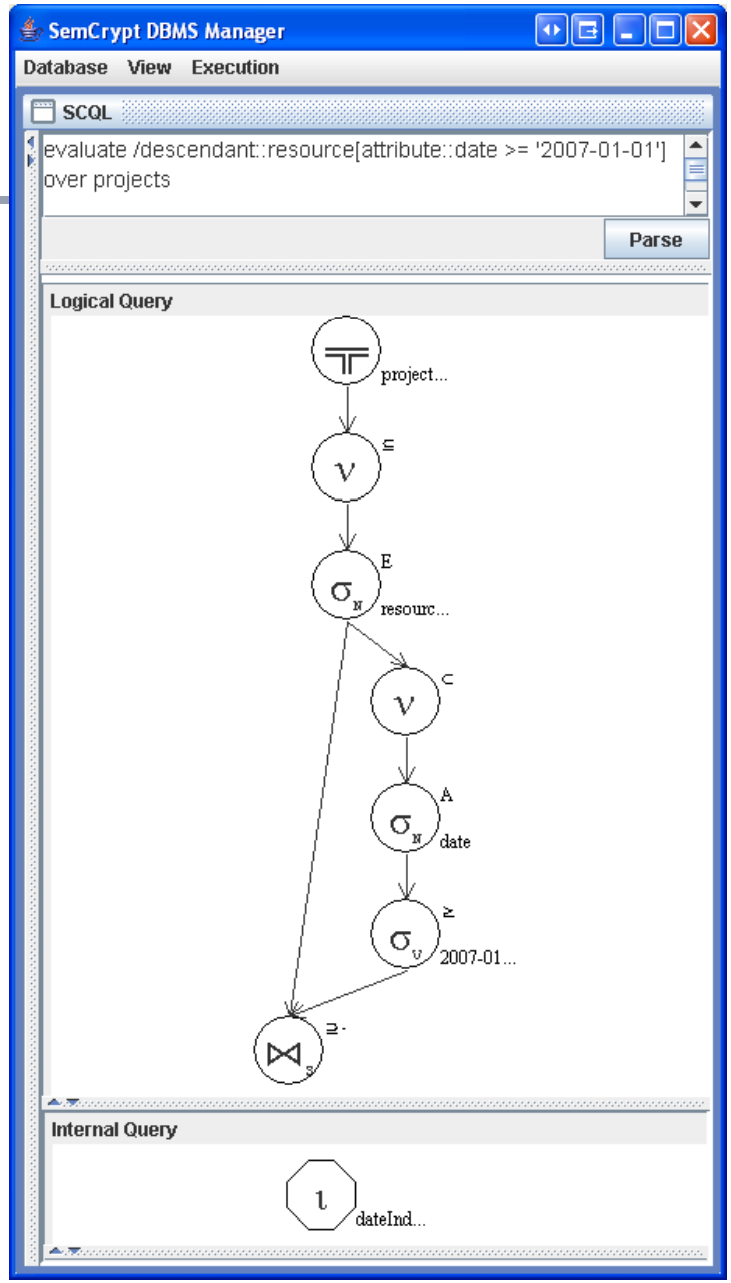


- match entire update fragment instead of single nodes
- only query nodes that are not contained in update fragment
- improvement to KeyX

U1	1 date
U2	1 resource
U3	1 milestone with 100 resources
U4	1 project with 10 milestones and 100 resources each

XML Database SemCrypt

- FIT-IT project at DKE
- only stores and processes encrypted data at the server
- for outsourcing sensitive XML documents
- uses labeling and indices to efficiently search in encrypted data
- integrates all concepts of this thesis



Conclusion

- Provide arbitrary indices on document content and structure
  - Labeling scheme
  - Extensible, nestable structures
  - Index framework based on index model
  - Maintenance algorithm
- Represent and process those indices that best match query workload