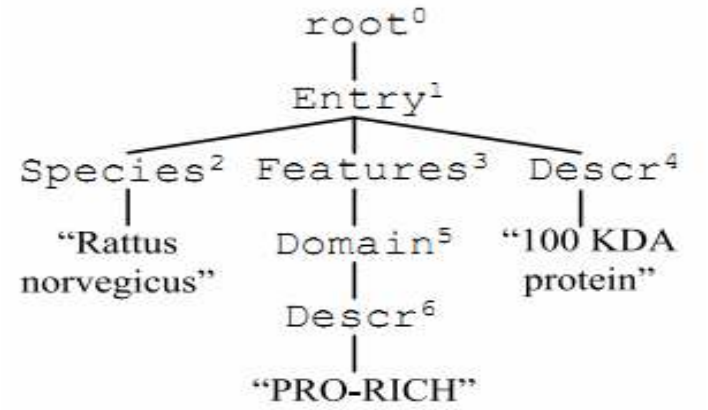
Efficient Storage and Temporal Query Evaluation of Hierarchical Data Archiving Systems

Hui (Wendy) Wang, Ruilin Liu
Stevens Institute of Technology, New Jersey, USA
Dimitri Theodoratos, Xiaoying Wu
New Jersey Institute of Technology, New, Jersey, USA

Scientific Data in XML

- eXtensible Markup Language (XML): A mark-up language
- XML for scientific data
  - Describe data structure and give simple processing instructions (e.g., XSIL and XDMF)
  - Provide common data format that is an open, universal standard (e.g, CML, MathML, GML)

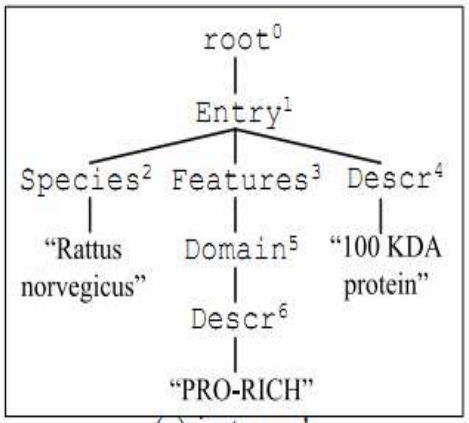


Swiss-Prot Dataset

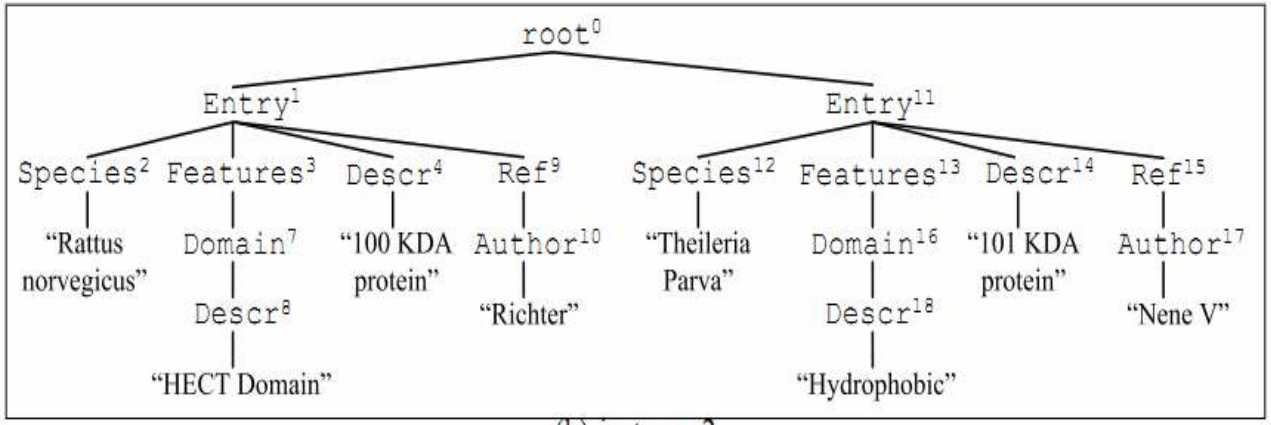
Updates on Scientific Data in XML

- Scientific databases are continuously updated
- Necessity of maintaining all versions in the archive
- **Problem:** as increasing volumes of these data being accumulated, the archives can reach a critical mass.

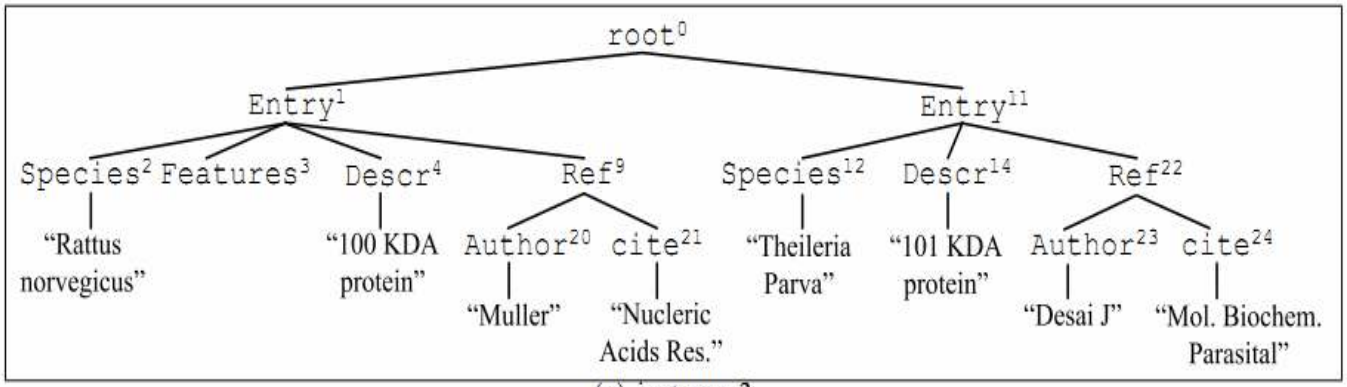
An Example of Multiple XML Dataset Versions



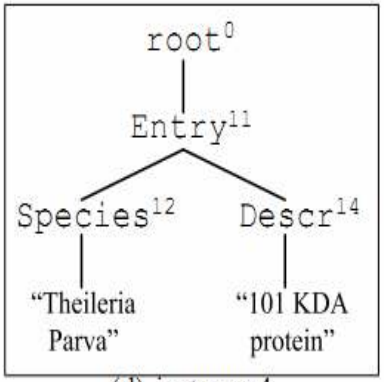
(a) instance 1



(b) instance 2



(c) instance 3



(d) instance 4

Four consecutive instances of an extract of the Swiss-Prot Dataset

Challenges

(1) how to store successive versions of XML databases in an archiving database in a cost-effective way.

(2) how to evaluate queries efficiently on the archiving database.

Our Contributions

- A novel **compact and updateable storage** scheme for XML archiving databases.
- A simple yet expressive **query language** for XML archiving databases.
- **Optimization of query evaluation** over the compact storage.

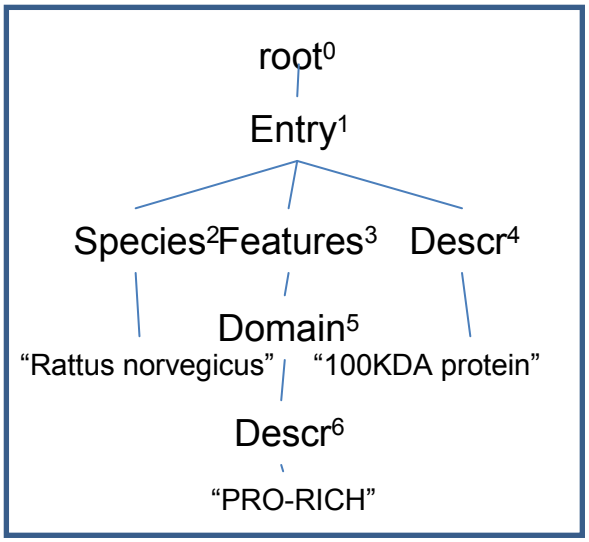
Outline

- Preliminaries
- **Compact storage**
- Temporal query language
- Query optimization
- Experiments
- Conclusion

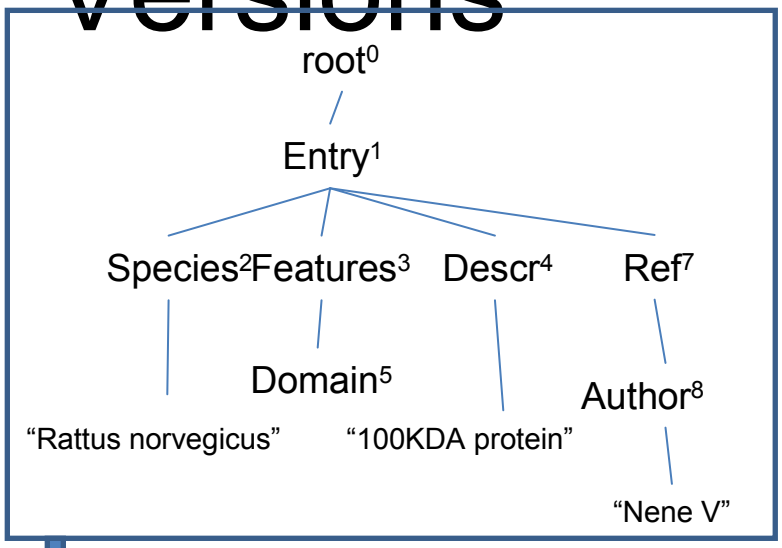
Compact Storage Scheme (I)

- Merge versions
  - Store multiple occurrences of the same node only once in the archiving database *A*.
  - Each node *p* in *A* is associated with a *timestamp set* which contains the timestamps of the instances of *p*.

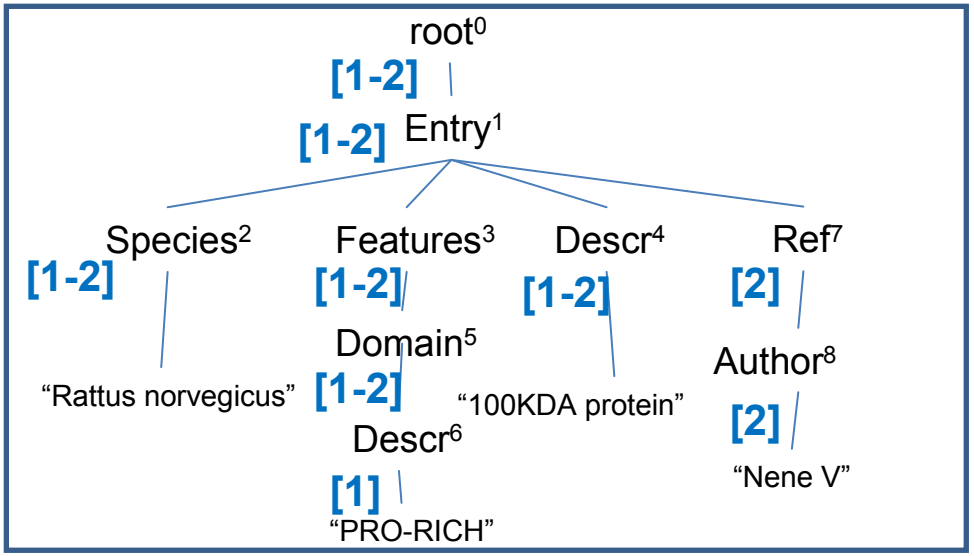
An Example of Merging Versions



Version 1



Version 2



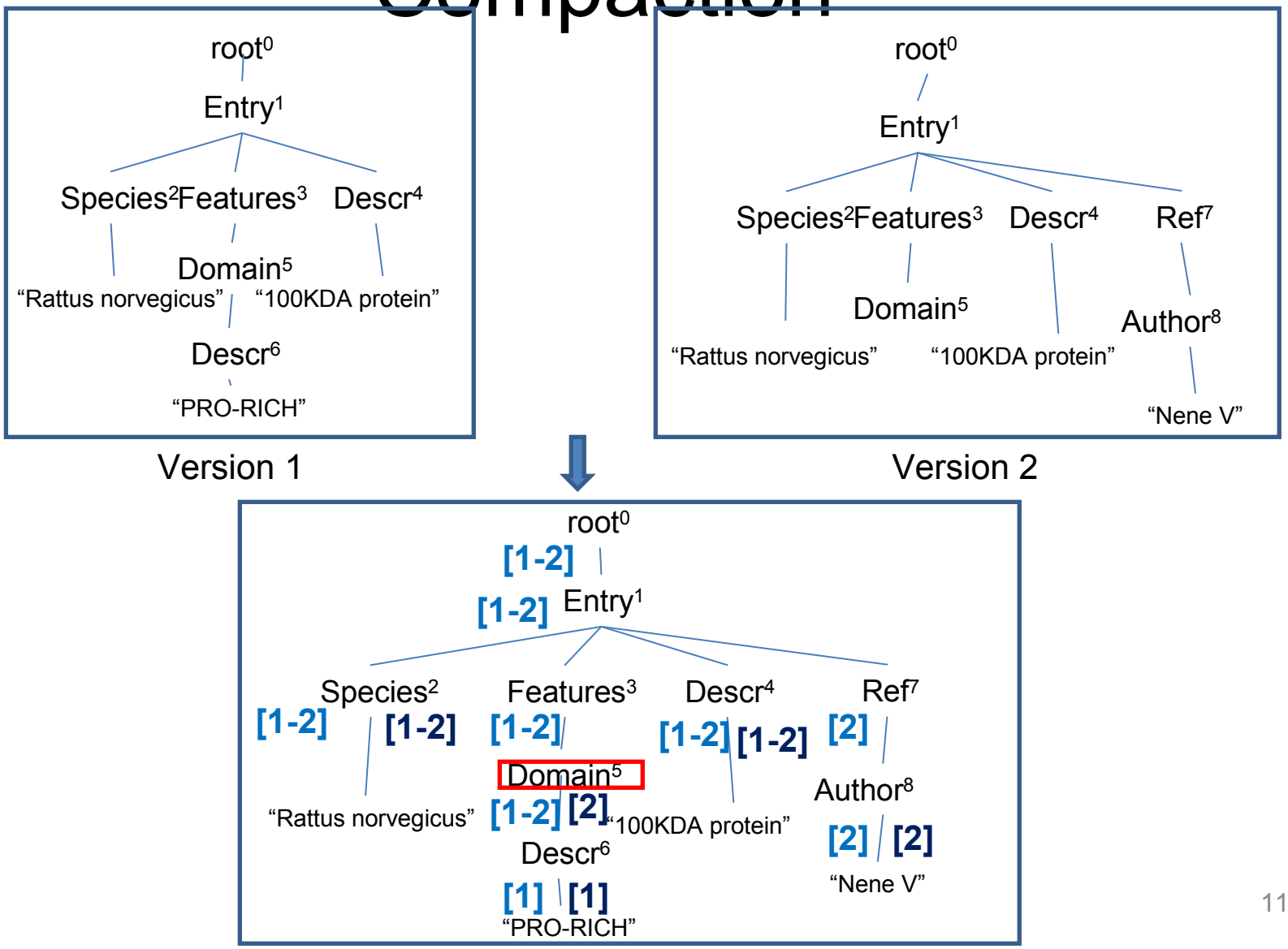
Monotone property on timestamps of the nodes on the same path

Compact Storage Scheme (II)

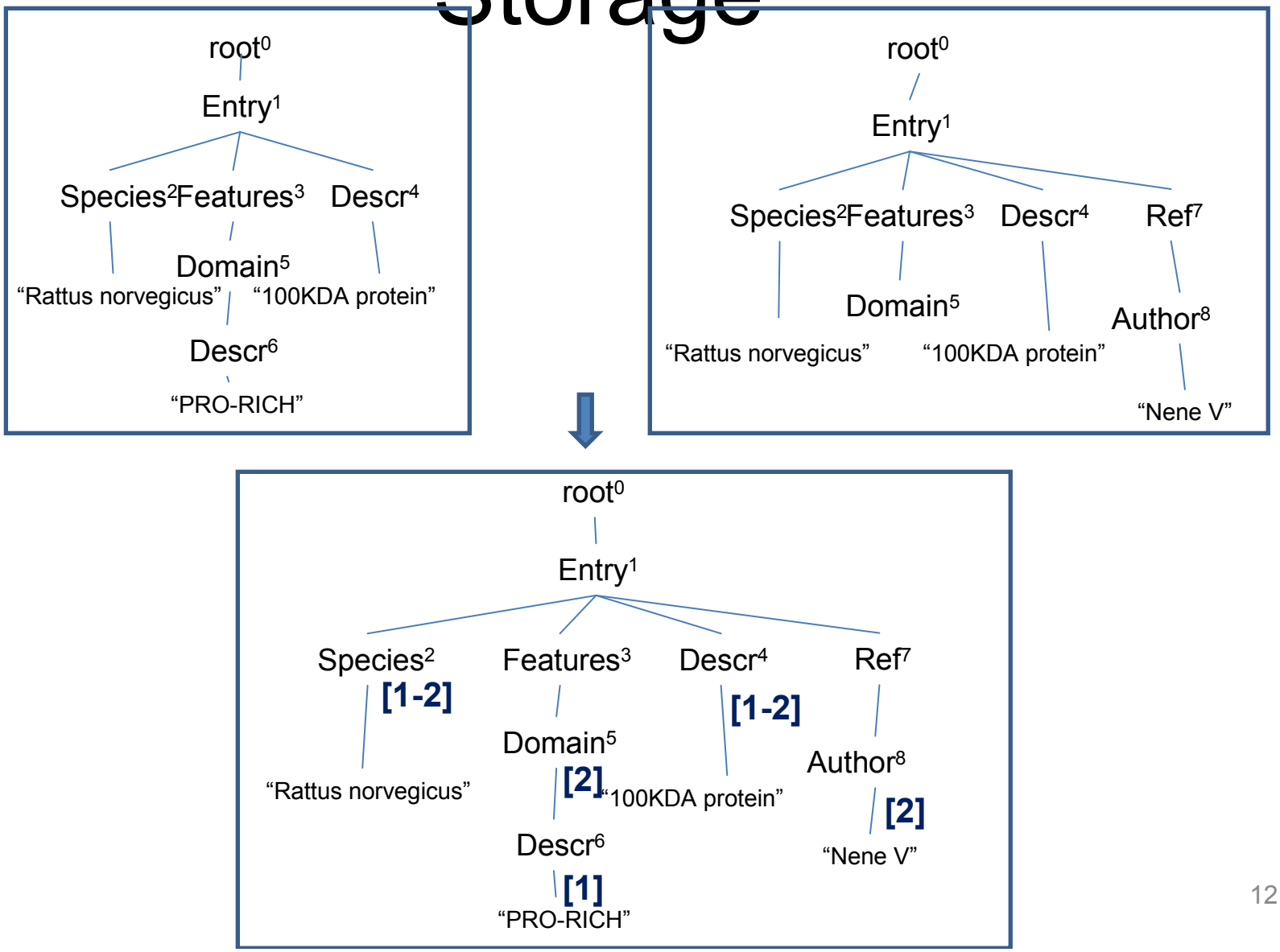
- Timestamp set compaction
  - The timestamp label of the parent node only preserves the timestamps that do not appear on any of its children.

$$Lp = ts(p) - \cup_{c \in child(p)} ts(c)$$

An Example of Timestamp Set Compaction

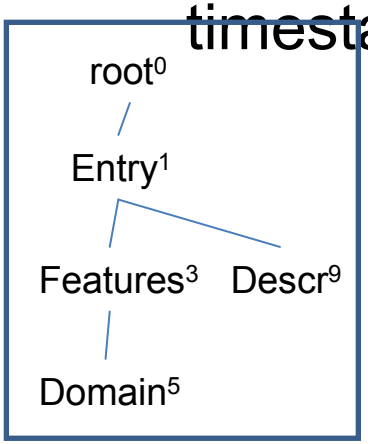


An Example of Compact Storage

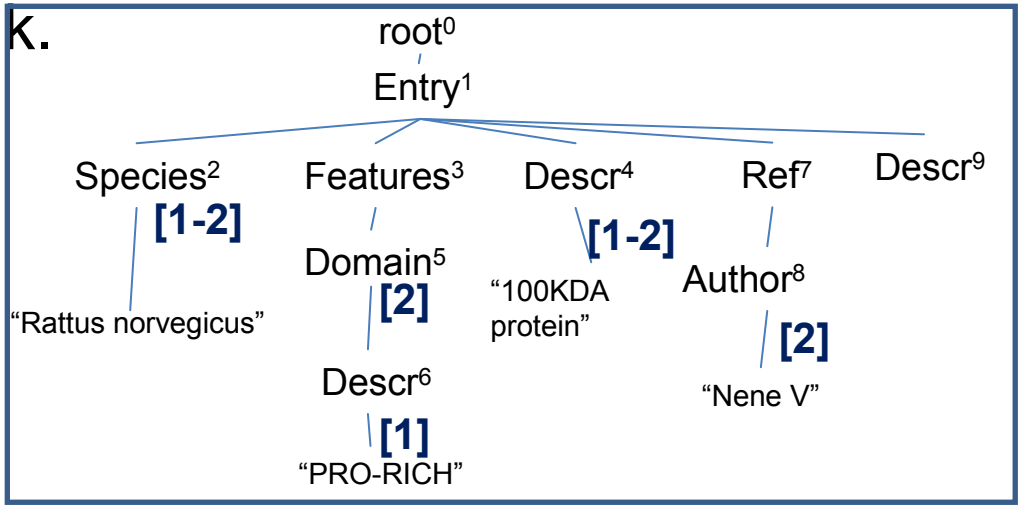


Efficient Updates

- Incremental timestamp label computation
  - When insert a new instance D_k at timestamp k,
    - add the newly inserted trees $T_1, \ldots, T_m$ in D_k to the archive,
    - For **leaf** nodes in D_k , update the timestamp labels of their corresponding archive nodes by adding timestamp k.



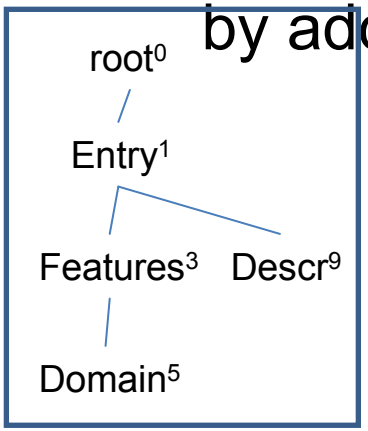
Version 3



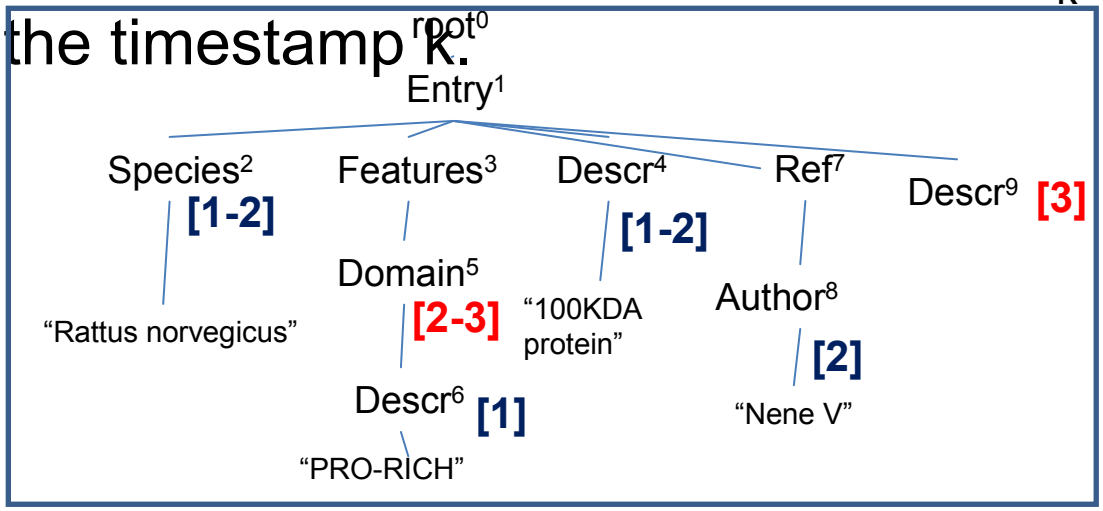
Archiving Dataset before Update

Efficient Updates

- Incremental timestamp label computation
  - When insert a new instance D_k at timestamp k,
    - add the newly inserted trees $T_1, \ldots, T_m$ in D_k to the archive,
    - update the archive nodes of the **leaf** nodes in D_k by adding the timestamp k.



Version 3

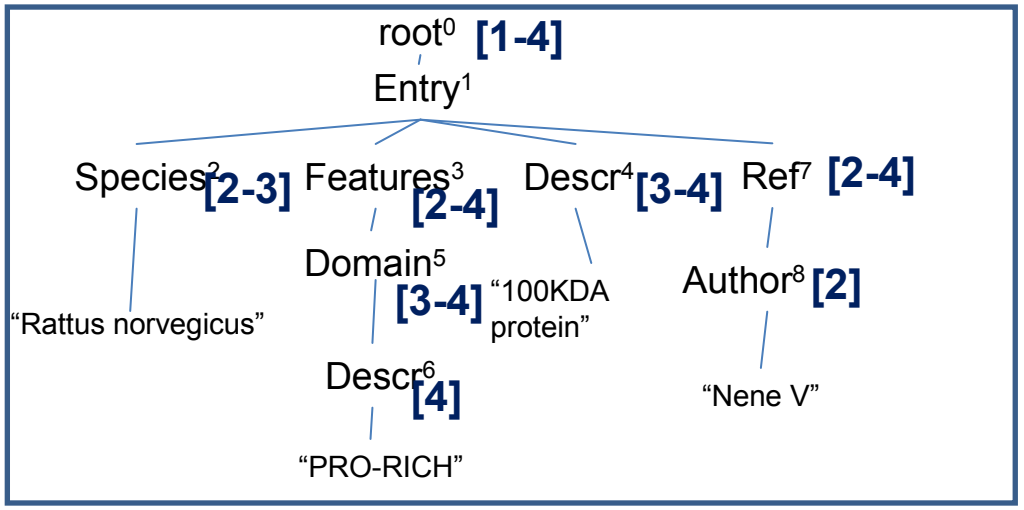


Archiving Dataset after Update

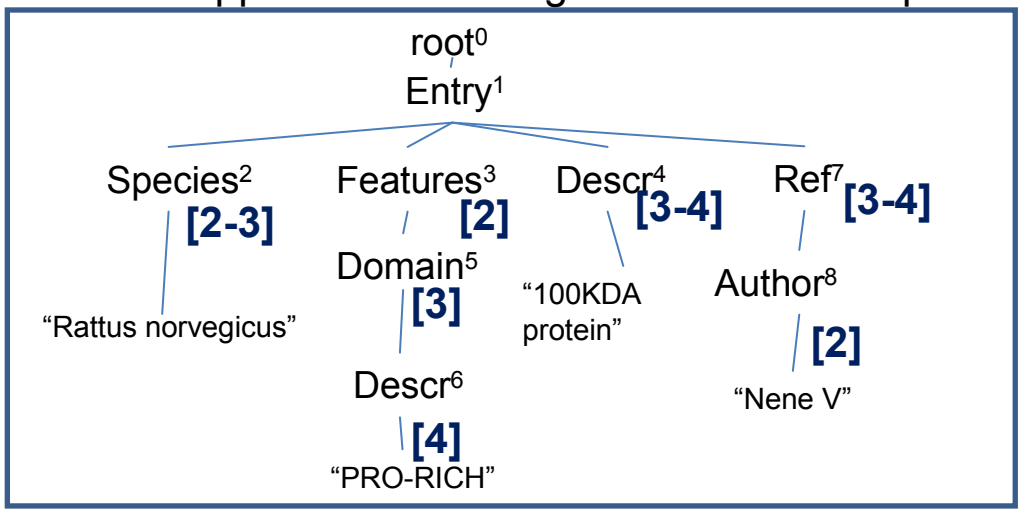
Comparison with Related Work

- Compact storage [1] (Top-down approach)
  - Remove timestamps at children if they are the same as those of the parent node
- Compared with our solution (bottom-up approach) w.r.t. # of updated timestamp labels when inserting a new instance D_k into the archiving database *A*
  - TD: # of nodes in D_k whose corresponding nodes in A have timestamp labels + new nodes in D_k
  - BU: # of leaf nodes in D_k

An Example of TD V.S. BU

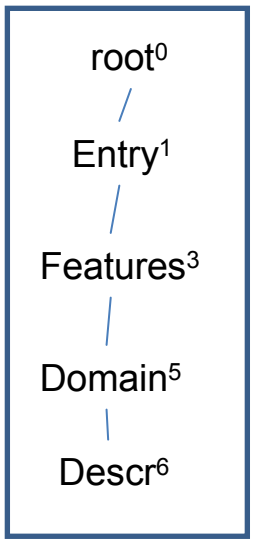


TD approach: Archiving Dataset before Update

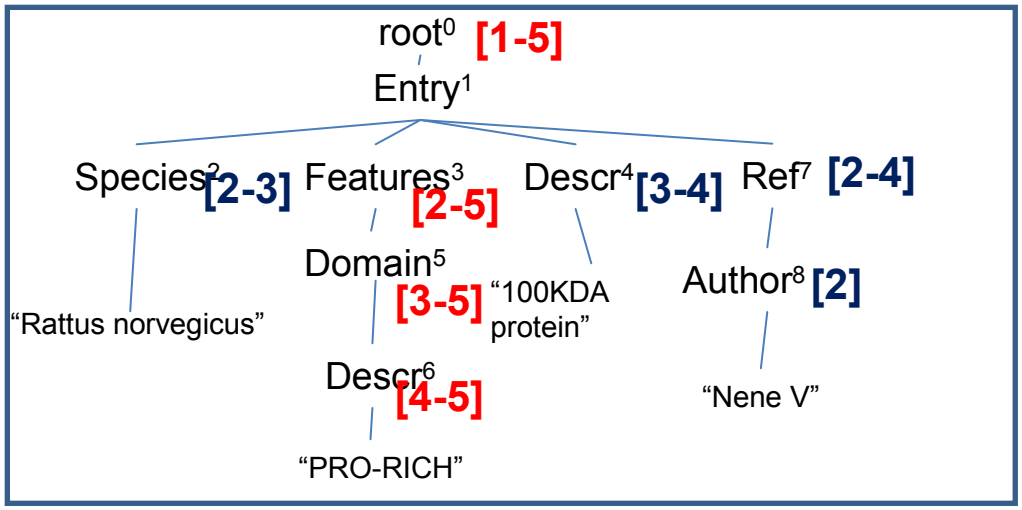


BU approach: Archiving Dataset before Update

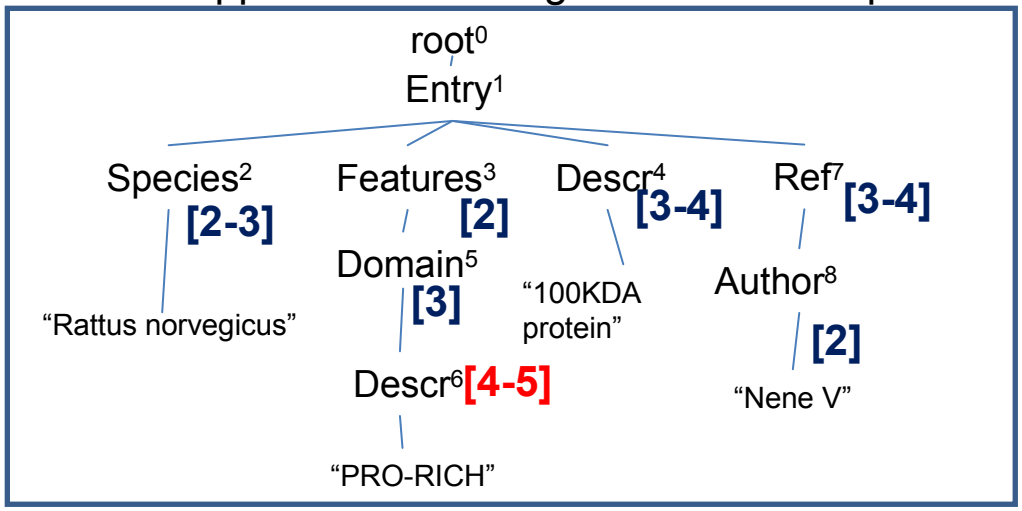
An Example of TD V.S. BU



Version 5



TD approach: Archiving Dataset after Update



BU approach: Archiving Dataset after Update

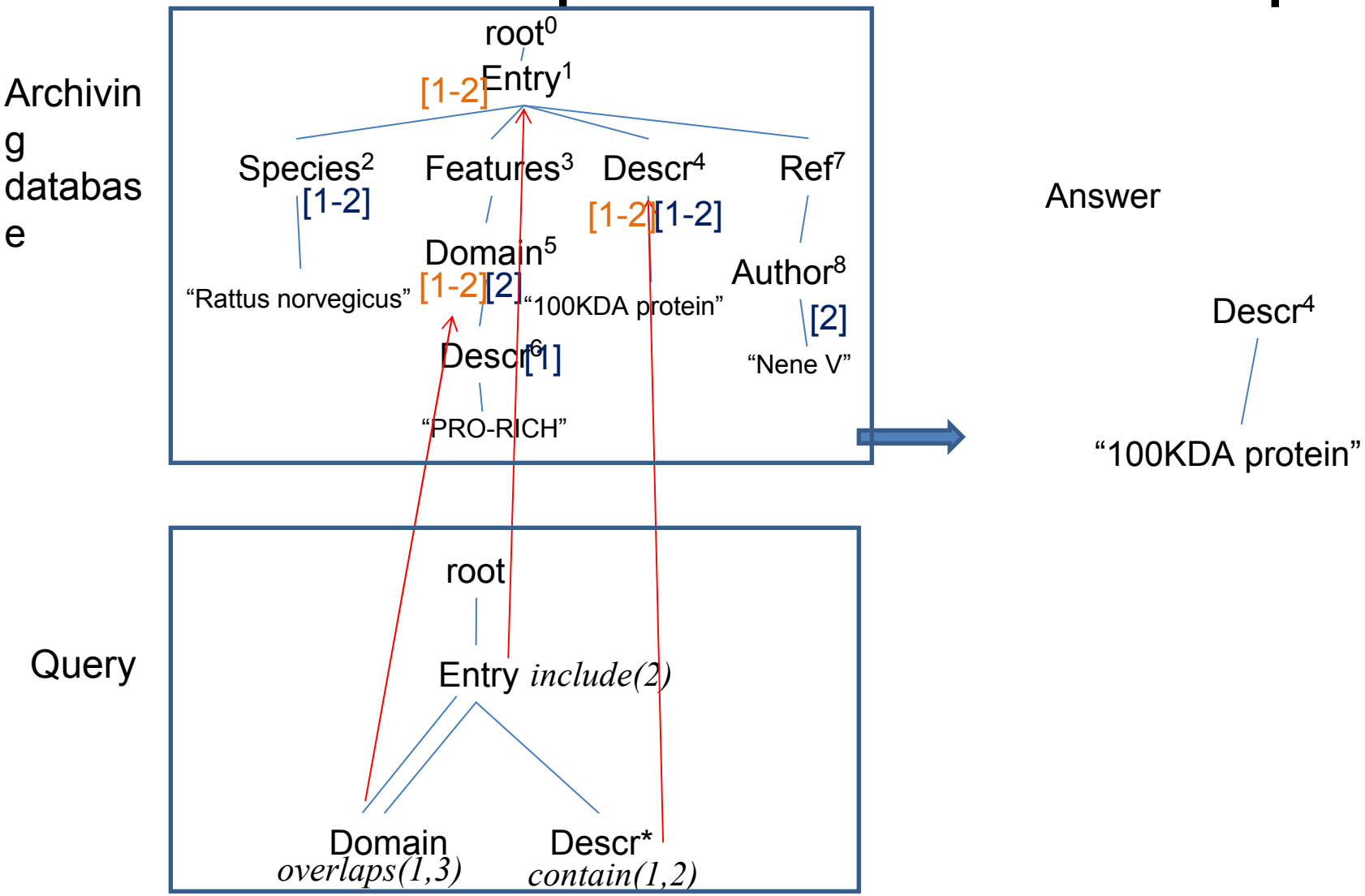
Outline

- Compact storage
- **Temporal query language**
- Query optimization
- Experiments
- Conclusion

Temporal Query Language

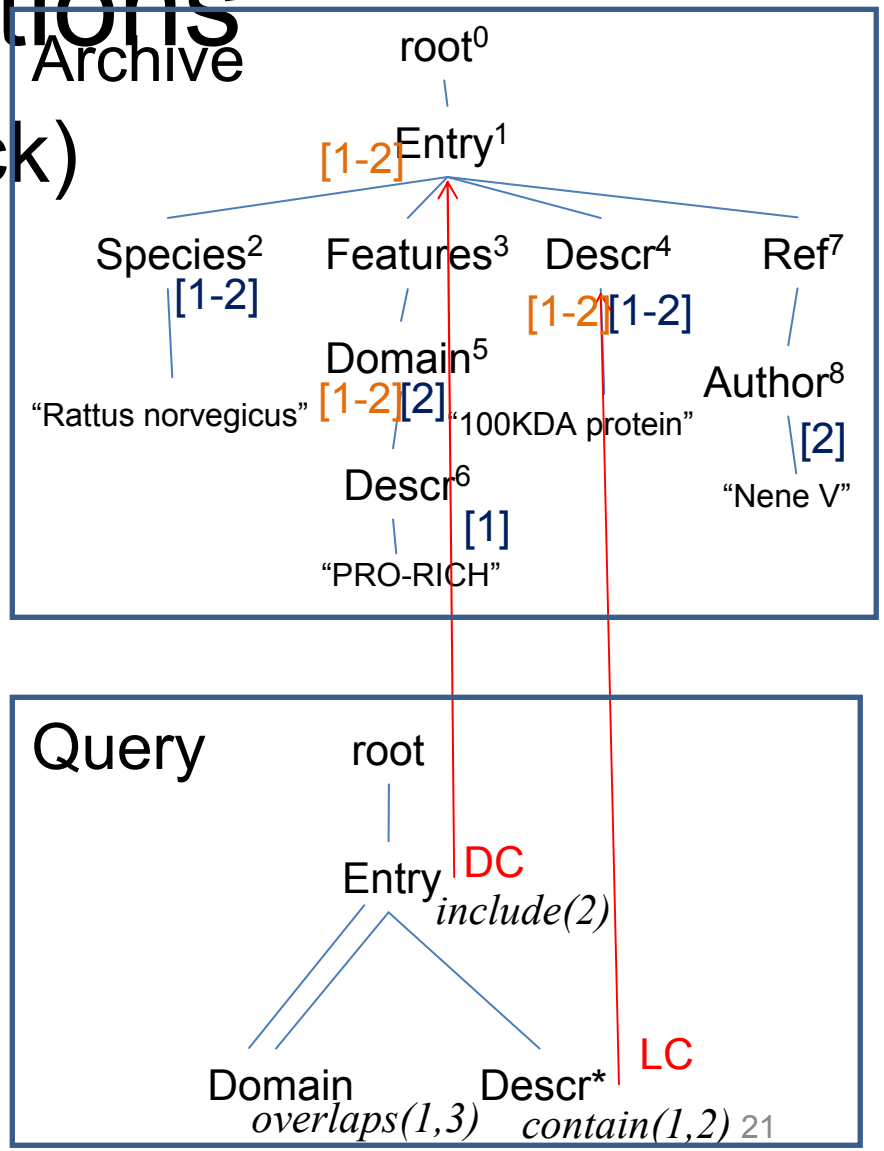
- Types:
  - Snapshot
  - history trace
- Temporal constraints:
  - $includes(t)$, $overlaps(t_a, t_b)$, $before(t)/after(t)$, $contains(t_a, t_b)/is\_contained(t_a, t_b)$, $meets(t_a, t_b)$
- Temporal queries: XML structural queries + temporal constraints on query nodes

Evaluation of Temporal Constraints over Compressed Timestamps



Temporal Evaluation Annotations

- *DC* (Descendant Check)
- *LC* (Local Check)
- *NC* (No Check)

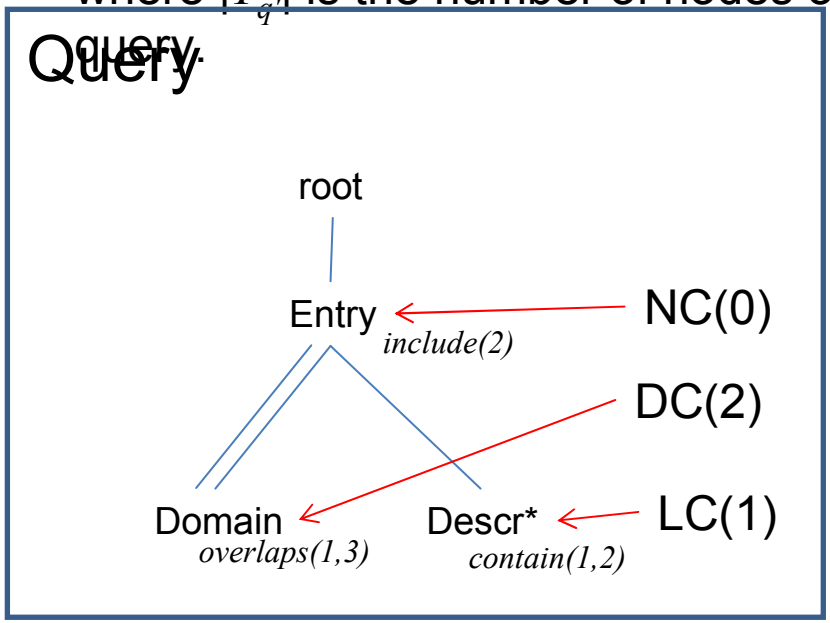


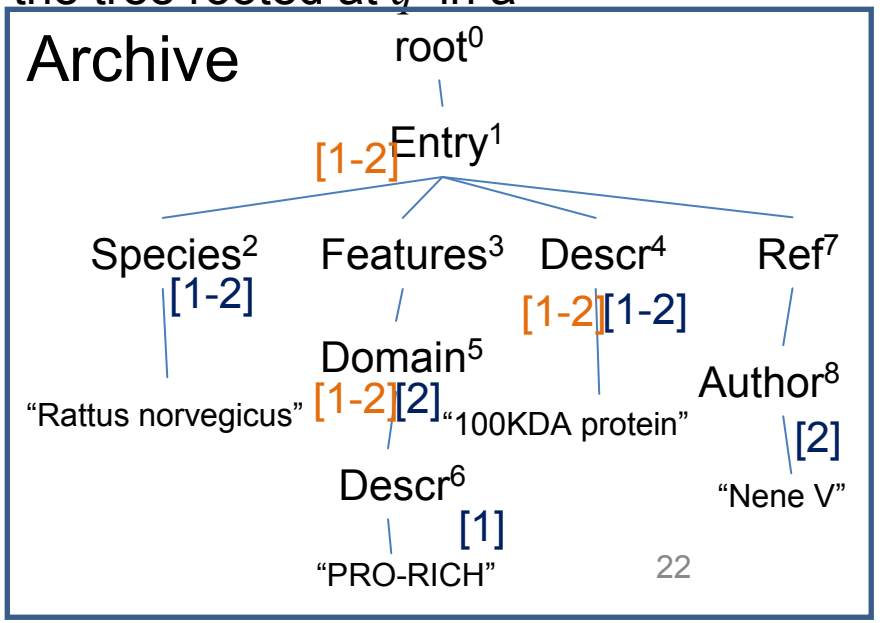
Cost Model

Cost model of temporal evaluation annotations

$$C_\ell(q) = \begin{cases} \max_{q'} |T_{q'}| & \ell = \mathrm{DC}; \\ 1 & \ell = \mathrm{LC}; \\ 0 & \ell = \mathrm{NC}. \end{cases}$$

where $|T_{q'}|$ is the number of nodes of the tree rooted at q' in a query.





Outline

- Preliminaries
- Compact storage
- Temporal query language
- **Query optimization**
- Experiments
- Conclusion

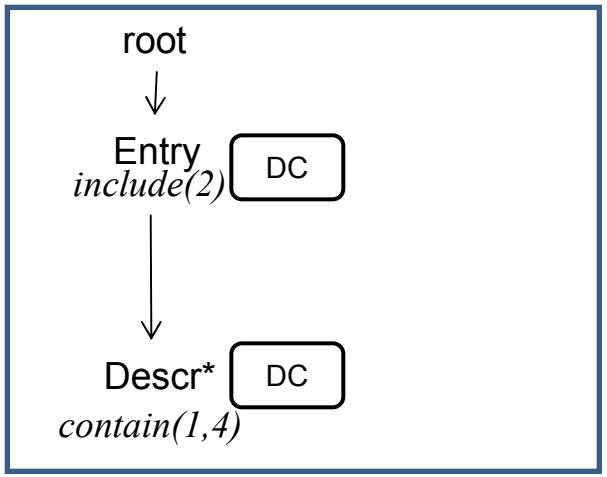
Optimization Problem

- DC is expensive (recursive check)
- DC can be replaced by LC/NC in some cases

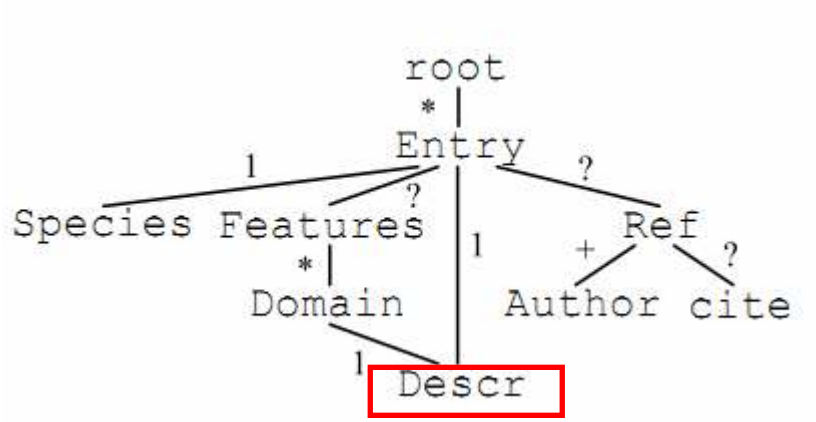
Goal:

- Replace as many DCs as possible with LCs/NCs

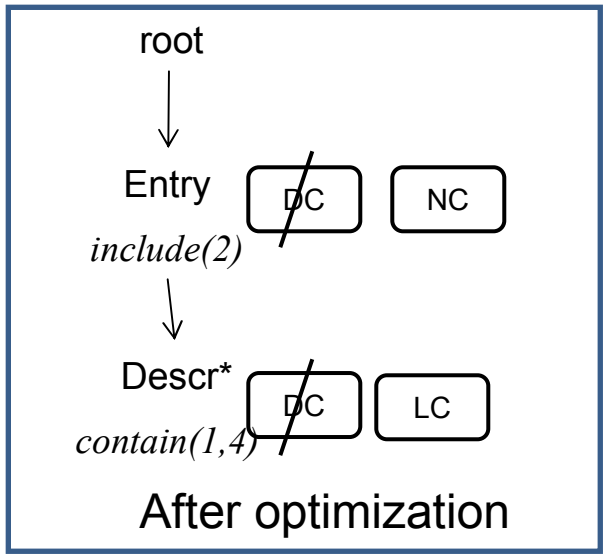
An Example of Optimization



Query Q



Database Schema



After optimization

Inference Rules

- We use inference rules to find redundant temporal annotations
- *Inference rules:* $P_1, \ldots, P_k \rightarrow R$
  - if the premises $P_1, \ldots, P_k$ are true, then the conclusion R is also true.
- Types of inference rules
  - Without database schema
  - With presence of database schema

Inference Rules: No Database Schema

- AD(ancestor-descendant) Rule:

$$Q \models p//q \rightarrow q \sqsupset_t p$$

- TR(transitivity) Rule:

$$p \sqsupset_t \mathrm{q} \text{ and } q \sqsupset_t r \rightarrow p \sqsupset_t r$$

Inference Rules: with Database Schema

- SP(SinglePath) Rule:

$$Q \models p//q, SinglePath(p\Delta, q\Delta) \rightarrow p =_t q$$

- DC(descendant) Rule:

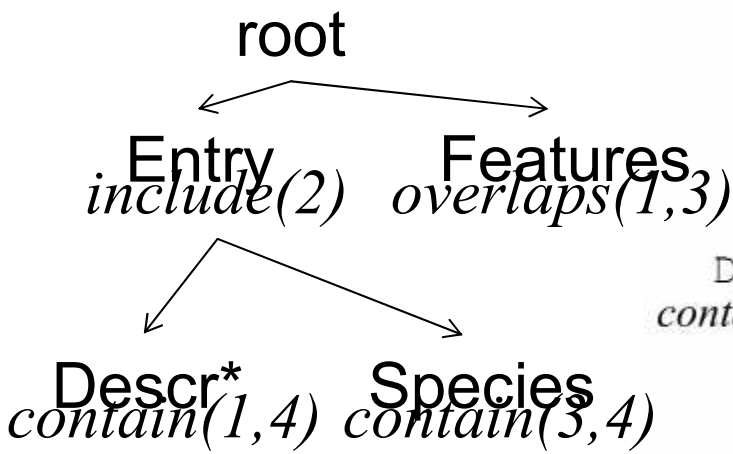
$$Q \models p//q, Q \models p//r, SinglePath(p\Delta, r\Delta) \rightarrow q \ \square_t \ r$$

- DE(derived) Rule:

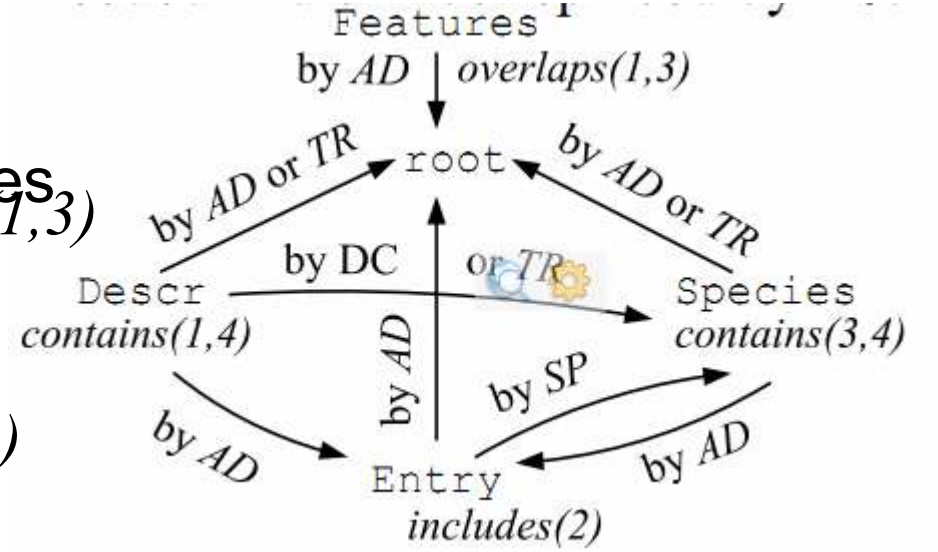
$$Q \models p//q, Q \models p//r, SinglePath(p\Delta, r\Delta), SinglePath(q\Delta, r\Delta) \rightarrow r \ \square_t \ q$$

Temporal Constraint Graph

Query



Temporal constraint graph



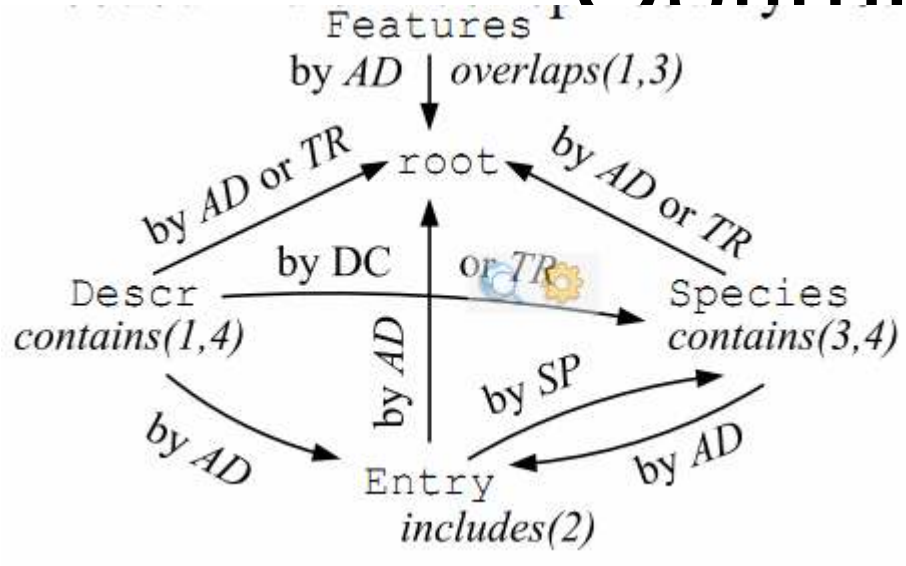
An edge from p to q indicates an inferred $p \square_t q$ relationship

Temporal Constraint Consumption

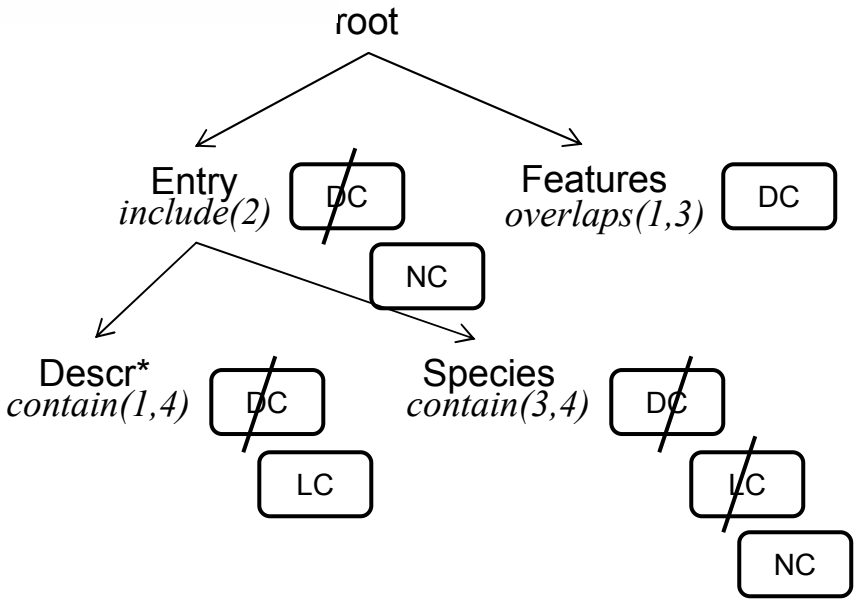
- We check temporal constraint consumption on the temporal constraint graph

Consuming temporal constraint on p	Consumed temporal constraint on q
$p \square_t q$	
$include(t)$	$includes(t)$ $overlaps(t_1, t_2), t \square [t_1, t_2]$
$contains(t_1, t_2)$	$includes(t_3), t_3 \square [t_1, t_2]$ $contains(t_3, t_4), t_3 \geq t_1, t_4 \leq t_2$ $overlaps(t_3, t_4), t_1 < t_3 < t_2$ or $t_1 < t_4 < t_2$
$q \square_t p$	
$is\_contained(t_1, t_2)$	$is\_contained(t_3, t_4), t_3 \leq t_1, t_4 \geq t_2$
$before(t)$	$before(t_1), t_1 \geq t$
$after(t)$	$after(t_1), t_1 \leq t$
$q =_t p$	
$meets(t_1, t_2)$	$meets(t_1, t_2)$

An Example of Query Optimization



Consuming temporal constraint on *p* *(Descr)*	Consumed temporal constraint on *q* *(Entry)*
Descr $\square_t$ *Entry*	
contains(*1*, *4*)	*includes*(*2*)



Outline

- Preliminaries
- Compact storage
- Temporal query language
- Query optimization
- **Experiments**
- Conclusion

Experiment Setup

- Hardware
  - Intel Core 2 CPU 2.40 GHz processor, 4.00 GB of RAM
- Software
  - OS: Windows 7
  - The algorithms were implemented in Java
  - JDOM engine: parse the XML databases
  - Wutka DTD parser: parse the XML DTD
  - Oracle Berkeley DB XML engine: query evaluation

Datasets

- Synthetic dataset
  - using the IBM XML generator on the DTD of the XMark benchmark
- Real dataset
  - Treebank dataset

Dataset	Size	# of elements	Max. depth	Avg. depth
Treebank	22.3MB	491108	36	8.5
Xmark	14.6MB	160929	21	20.068

Versions

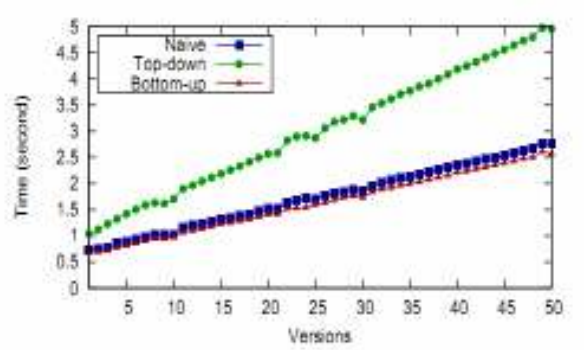
- For Xmark and Treebank datasets, we created 50 and 20 consecutive database instances respectively.
- Each instance was generated from the previous one by first deleting and then inserting (sub)trees.
- The (inserted and deleted) trees take 10% of the nodes of the database instance.

Experiment

- Three storage approaches:
  - The naive approach (NA): keeps the timestamp sets as un-compacted
  - The top-down (TD) approach ([1]): eliminates the timestamps of the ***children*** nodes that are identical to the parent.
  - Our bottom-up (BU) approach: eliminates the timestamps from the ***parent*** nodes that are repeated on children.

Experiment: Archiving Overhead

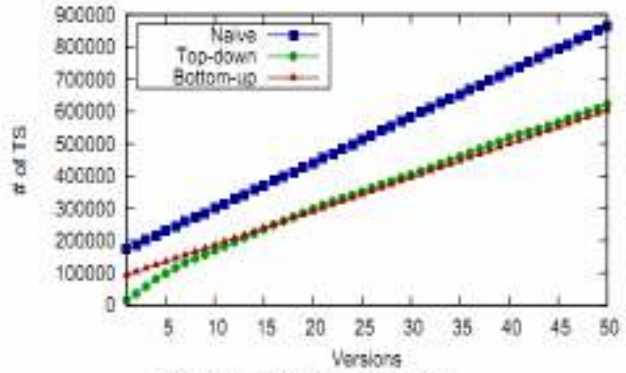
Archiving Time



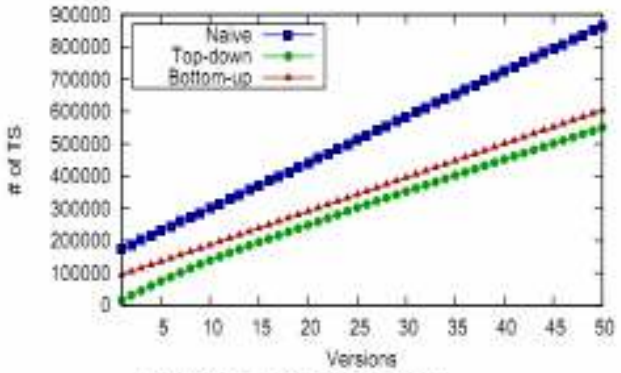
Compaction ratio

Dataset	Top-down	Bottom-up
XMark (shallow&fat)	2.15%	2.72%
XMark (deep&thin)	3.09%	2.75%

Total number of timestamps (space overhead)

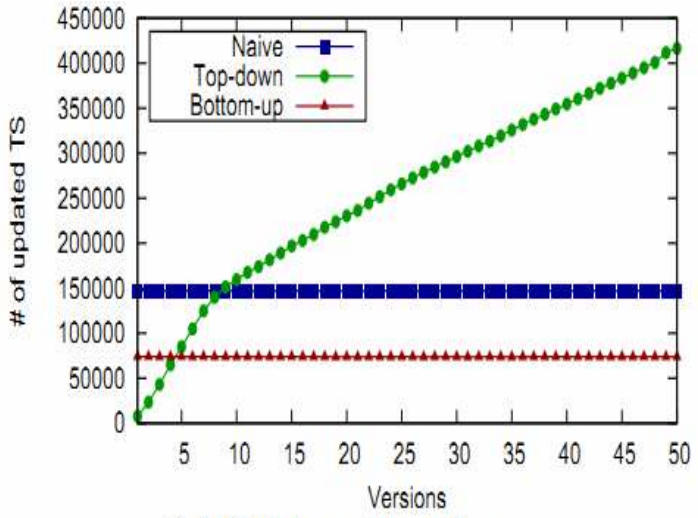


(a) DT tree insertion

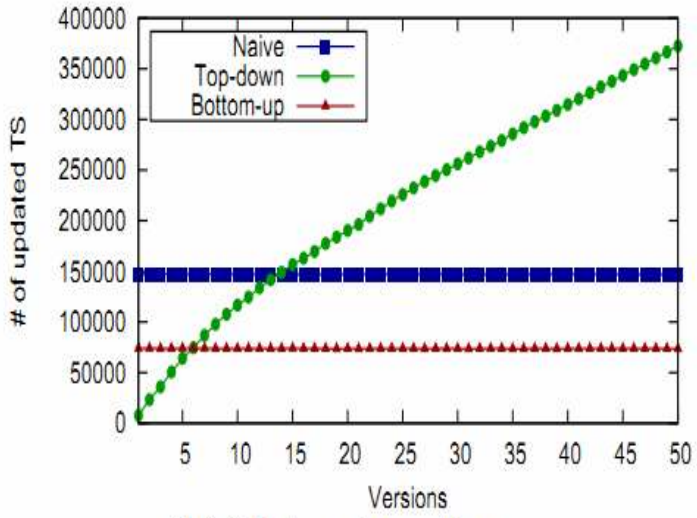


(b) SF tree insertion

Experiment: Update Cost



(a) DT tree insertion



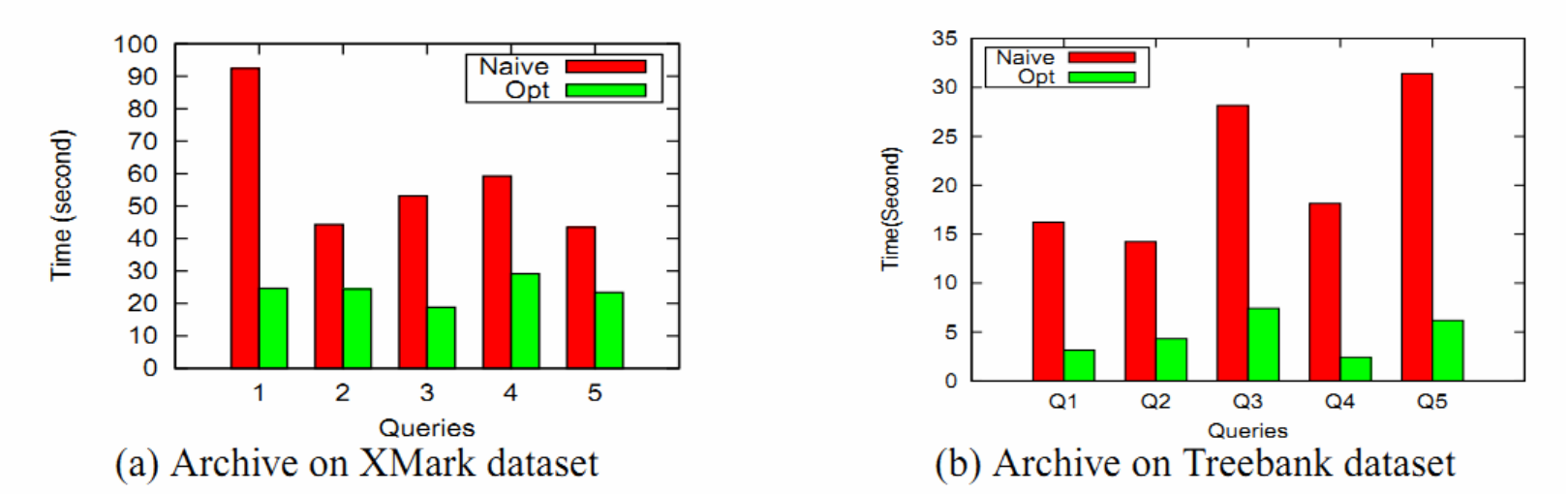
(b) SF tree insertion

Summary of archiving overhead

- Both TD and BU can reduce the number of timestamps in the archive.
- The difference between TD and BU regarding the number of timetstamps is not significant.
- BU always has much better update cost than TD.

Experiment: Query Optimization

Temporal Constraint Evaluation Optimization



(a) Archive on XMark dataset

(b) Archive on Treebank dataset

Our optimization can bring significant performance improvement with negligible overhead

Outline

- Preliminaries
- Compact storage
- Temporal query language
- Query optimization
- Experiments
- **Conclusion**

Conclusion

- We proposed an efficient XML archiving database system that consists of
  - A novel compact and updateable storage scheme
  - A simple yet expressive query language for XML archiving databases
  - Optimization of query evaluation over the compact storage

Future Work

- Consider additional temporal constraints
- Consider unrestricted database schemas that may contain cycles
- Design efficient optimization algorithms that can work in this broader framework

Thank you!

Reference

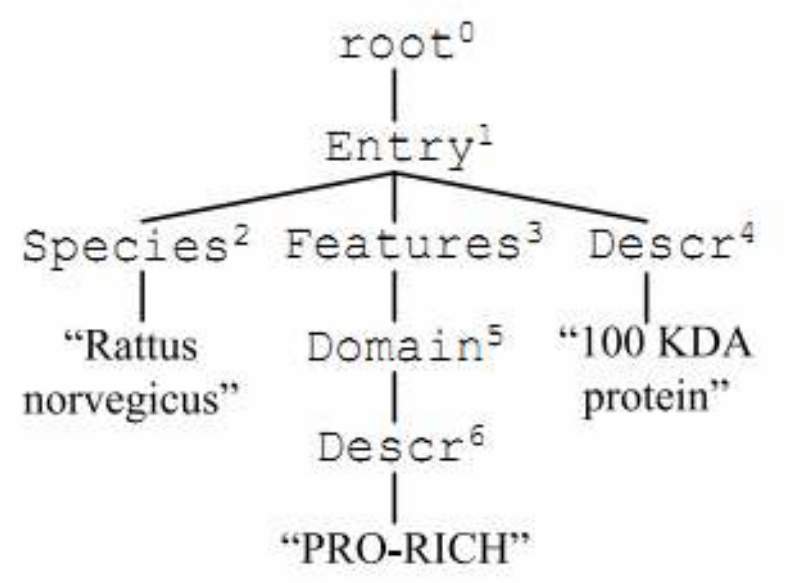
- [1] P. Buneman, S. Khanna, K. Tajima, and W.-C. Tan. Archiving scientific data. In ACM Transactions on Database Systems, 2004.
- [2] A. P. Chapman, H. Jagadish, and P. Ramanan. Efficient provenance storage. In SIGMOD, 2008.
- [3] S. Chawathe and H. Garcia-molina. Meaningful change detection in structured data. In SIGMOD, 1997.
- [4] P. T. Jayant and J. R. Haritsa. Xgrind: A query-friendly xml compressor. In ICDE, 2002.
- [5] H. Liefke and D. Suciu. XMill: an efficient compressor for XML data. In SIGMOD, 1999.
- [6] H. M ¨uller, P. Buneman, and I. Koltsidas. Xarch: Archiving scientific and reference data. In SIGMOD, 2008.
- [7] F. Rizzolo and A. A. Vaisman. Temporal xml: modeling, indexing, and query processing. The VLDB Journal, 17:1179–1212, August 2008.
- [8] F. Wang and C. Zaniolo. Temporal queries in XML document archives and web warehouses. In TIME-ICTL, 2003.
- [9] F. Wang and C. Zaniolo. Temporal queries and version management in XML-based docu-ment archives. Data Knowl. Eng., 65:304–324, May 2008.

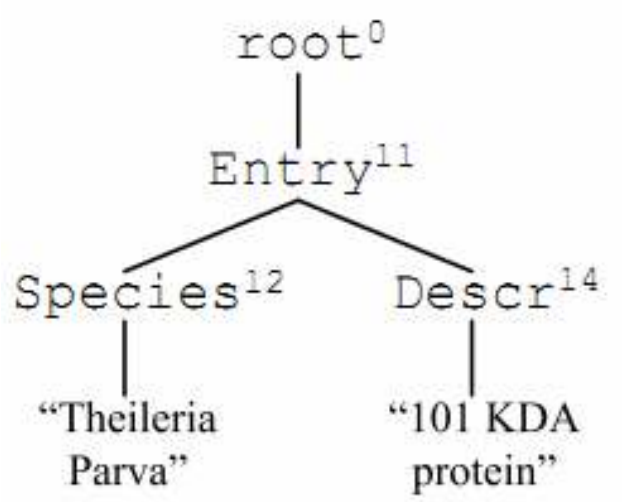
Outline

- **Preliminaries**
- Compact storage
- Temporal query language
- Query optimization
- Experiments
- Conclusion

XML database

- XML database: tree-structured, ID-based.



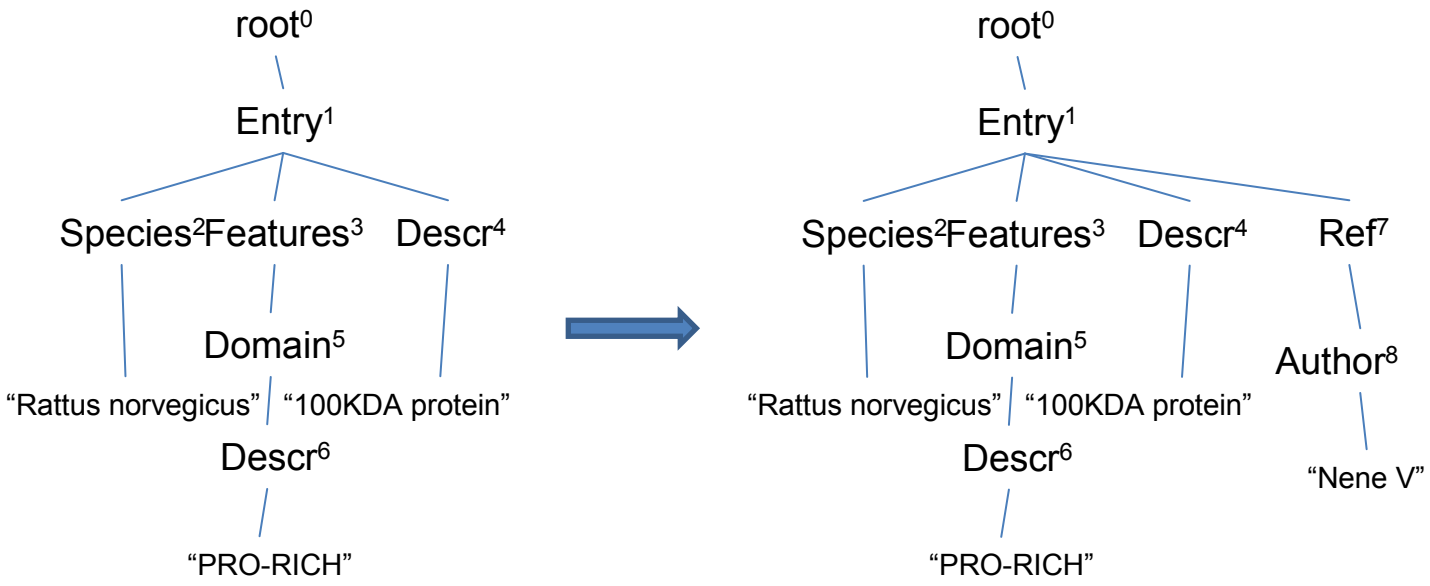


Archiving XML database

- XML instances: XML database at certain time point
  - Each instance is associate with a timestamp (version number)
- Archiving Database: multiple instances are merged into one database

Updating XML database

- Update operations:
  - ***insertion***
  - ***deletion***
- A sequence of update operations can be modeled as a set of deletions followed by a set of insertions.



A Piece of Related Work

P. Buneman, et al. *Archiving scientific data.* TODS, 2004.

- Solution:
  - Merge instances
  - Remove timestamps if same as parent node (Top-down)
- Weakness:
  - Inefficient updates
  - Inefficient query

Witness Graph

- Definition

  Given a query Q , a witness graph for Q is a graph WQ such that

  a)the nodes of W_Q correspond to the nodes of Q

  b)there is an edge from node p to node q in W_Q , iff p is a witness node of q in Q.

- Example
  - Construct the witness graph from the temporal constraint graph after temporal annotation consumption (next page)