XML Storage

XML and DBMS

- XML documents need to be stored and retrieved in databases
  - Exploit capabilities of RDBMS for efficient XML data processing
    - Scalability, Availability, Performance, ...
  - Need to support mapping schemes for shredding XML into relations
  - Need to propagate constraints from XML to relations
- Data stored in databases need to be published in XML
  - Most (structured) data stored in RDBMS
  - Seamless integration of XML and Relational data
  - Specification schemes for publishing needed
  - Efficient publishing algorithms needed
- When not to use RDBMS
  - Streaming data (RSS, SOAP), Information Retrieval (Google)

Storing XML Data

- Flat streams: store XML data as is in text files
  - fast for storing and retrieving whole documents
  - query support: limited; concurrency control: no
- Native XML Databases: designed specifically for XML
  - XML document stored as is
  - Efficient support for XML queries
  - Many techniques need to be re-developed
- Colonial Strategies: Re-use existing storage systems
  - Leverage mature systems (RDBMS)
  - Simple integration with legacy data
  - Map XML document into underlying structures
    - E.g., shred document into flat tables

Storing XML in a Database

- Build a model
  - Model the data in the XML document, or...
  - Model the XML document itself
- Map the model to the database
- Transfer data according to the model

XML Data - Revisit

- XML adds a new data model to the world
  - In addition to relational, hierarchical, OO, ...
- The “XML” data model is
  - A tree of ordered nodes
  - Nodes have different types (element, attribute, etc.)
  - Some nodes are labeled (Date, Quantity, Price, etc.)
  - Data stored in leaf nodes
- Modeling language is an XML schema language
  - DTD, XML Schemas, etc.

Sample XML Document

```
<Order>
   <Number>1234</Number>
   <Customer>Gallagher Co.</Customer>
   <Date>29.10.00</Date>
   <Item Number="1">
      <Part>A-10</Part>
      <Quantity>12</Quantity>
      <Price>10.95</Price>
   </Item>
   <Item Number="2">
      <Part>B-43</Part>
      <Quantity>600</Quantity>
      <Price>3.99</Price>
   </Item>
</Order>
```

Modeling Data

- Objects in model are specific to XML schema

```
object Order {
 number=1234;
 customer="Gallagher Corp.";
 date=29.10.00;
 items={ptrs to Items};
}
```

```
object Item {
 number=1;
 part="A-10";
 quantity=12;
 price=10.95;
}
```

```
object Item {
 number=2;
 part="B-43";
 quantity=600;
 price=3.99;
}
```

Storing Data

- Database schema specific to XML schema

Orders

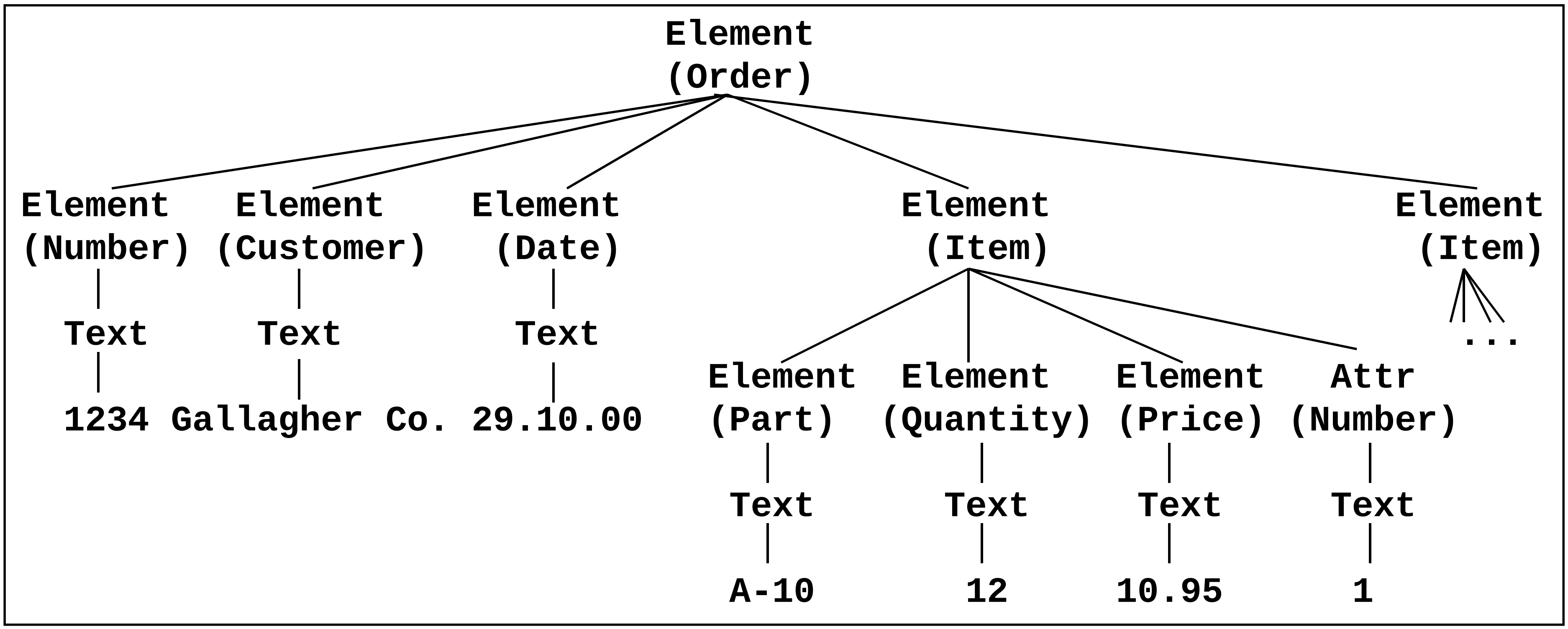
Number	Customer	Date
1234	Gallagher Co.	291000
...	...	...
...	...	...

Items

SONum	Item	Part	Qty	Price
1234	1	A-10	12	10.95
1234	2	B-43	600	3.99
...	...	...	...	...

Modeling Documents

- Objects in model independent of XML schema



Storing Documents

- Database schema independent of XML schema (order columns not shown)

Elements

ID	Name	Parent
1	Order	--
2	Number	1
3	Customer	1
4	Date	1
5	Item	1
6	Item	1
7	Part	5
8	Quantity	5
9	Price	5
10	Part	6
11	Quantity	6
12	Price	6

Attributes

ID	Name	Parent
13	Number	5
14	Number	6

Text

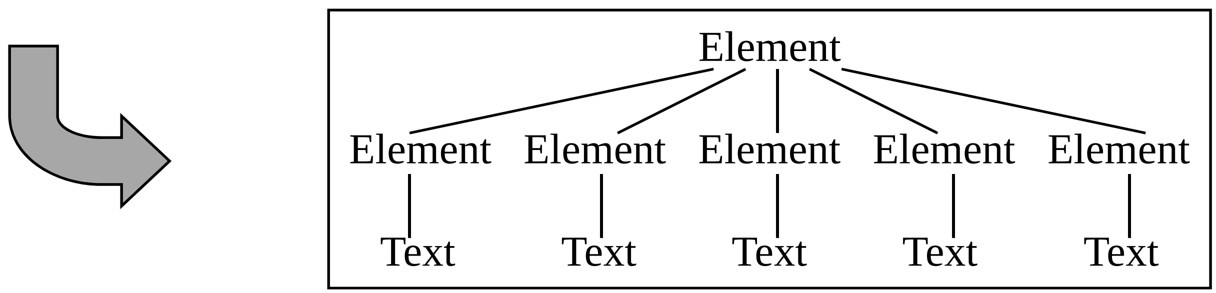
Parent	Value
2	1234
3	Gallagher Co.
4	29.10.00
7	A-10
8	12
9	10.95
10	B-43
11	600
12	3.99
13	1
14	2

Model-based Storage

- Stores documents in “object” form
- Documents parsed when inserted
- For example, store DOM objects in OODBMS
- Underlying storage can be relational, object-oriented, hierarchical, proprietary
- Uses indexes to speed searches

Model-based Storage

```
<Address>
   <Street>123 Main St.</Street>
   <City>Chicago</City>
   <State>IL</State>
   <PostCode>60609</PostCode>
   <Country>USA</Country>
</Address>
```



Model-based Databases

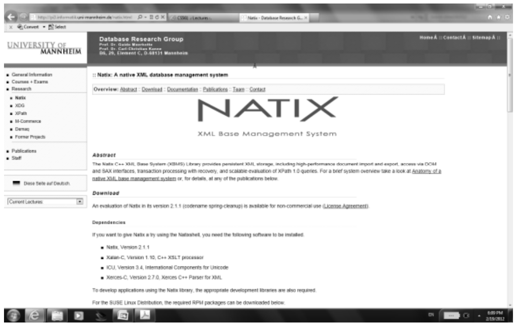
- Proprietary
  - Tamino, Xindice, Neocore, Ipedo, XStream DB, XYZFind, Infonyte, Virtuoso, Coherity, Luci, TeraText, Sekaiju, Cerisent, DOM-Safe, XDBM, ...
- Relational
  - Xfinity, eXist, Sybase, DBDOM
- Object-oriented
  - eXcelon, X-Hive, Ozone/Prowler, 4Suite, Birdstep

Native XML Database

- A native XML database:
  - defines a logical model for XML documents
  - stores and retrieves documents according to this model
  - the model must include elements, attributes, PCDATA (text) and document ordering at a minimum
  - models are generally graph-based, viewing elements, attributes and PCDATA as nodes, with parent/child and sibling relationships
  - has the XML document as its fundamental unit of (logical) storage c.f. rows in relational databases

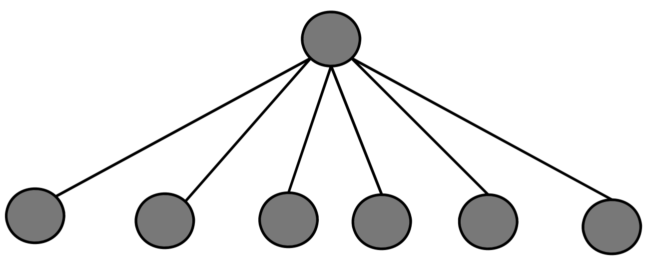
Natix

- Each sub-tree is stored in a record
- Store records in blocks as in any database
- If record grows beyond size of block: split
- Split: establish proxy nodes for subtrees
- Technical details:
  - use B-trees to organize space
  - use special concurrency & recovery techniques
- pi3.informatik.uni-mannheim.de/natix.html

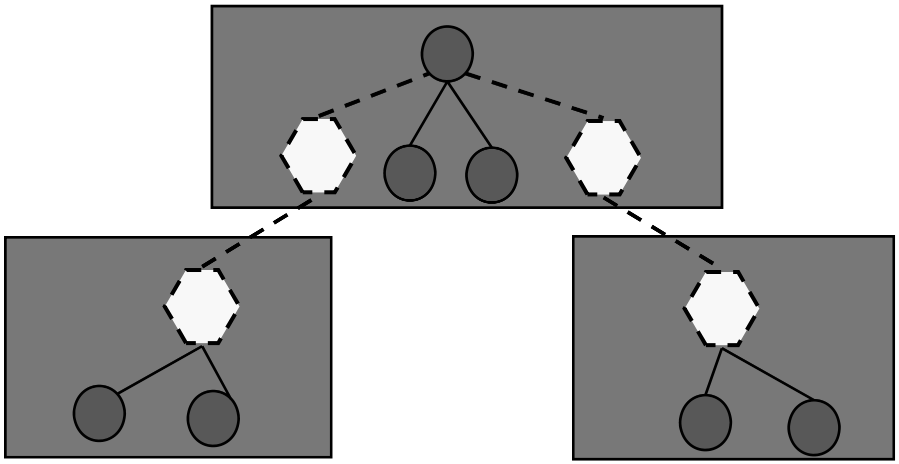


Natix

Logical XML document



Physical representation

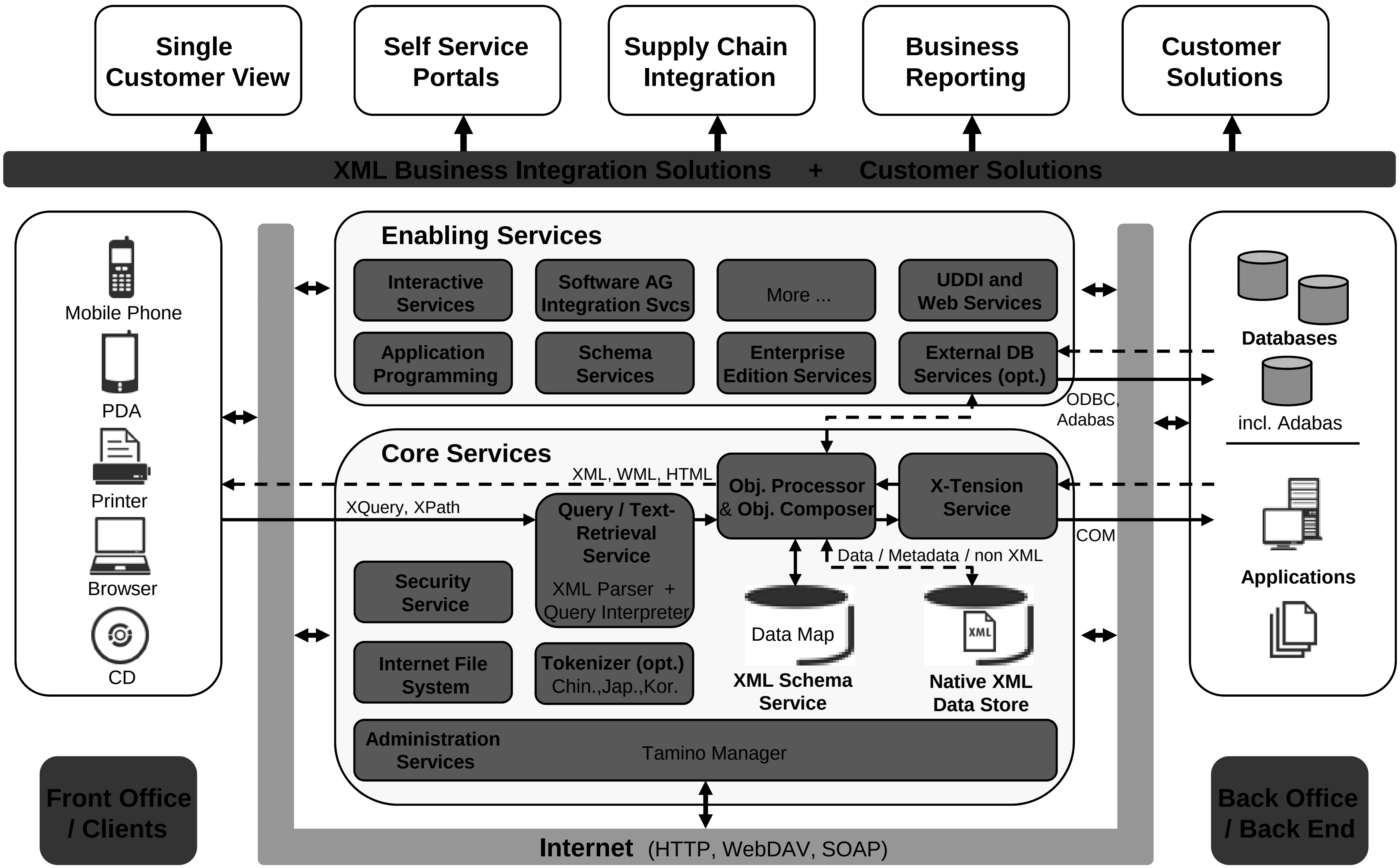


Tamino XML Server

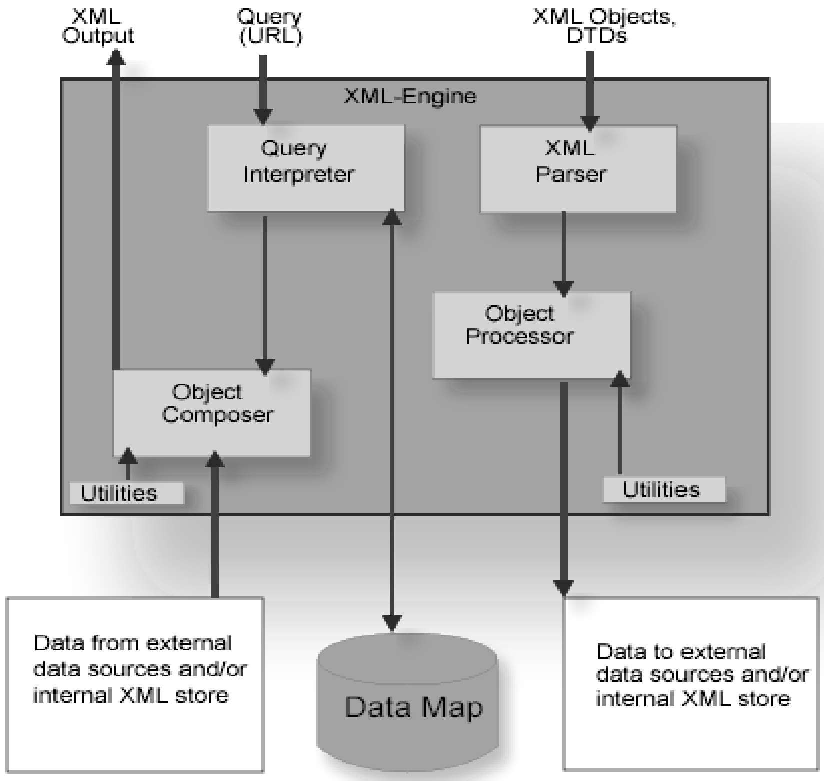
- Supports Internet and W3C standards
- Stores/Retrieves native XML documents and non-XML
- Offers full text search on XML documents
- Accesses Rdbms databases
- Integrates with other business applications
- Supports any programming language
- Works with major Web servers and Applications servers

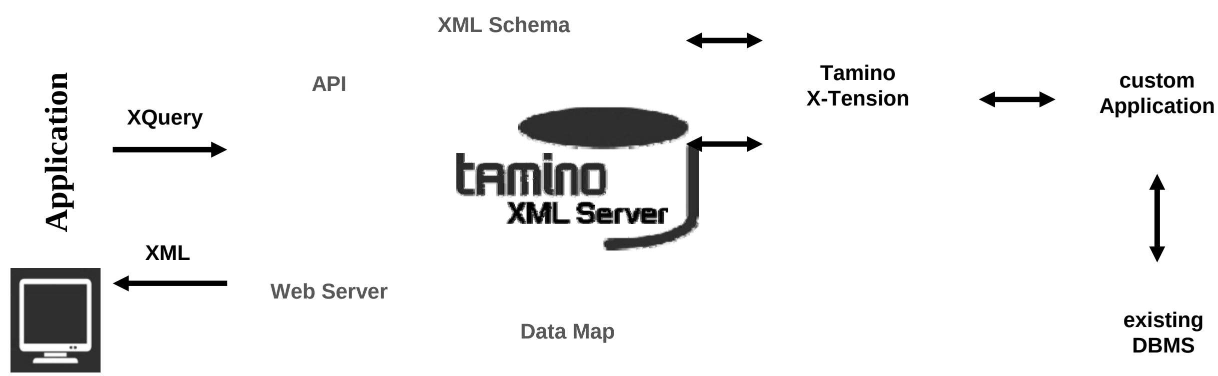
Tamino Features

- Integration of data in existing, external data sources
  - Access to and modification of data from diverse systems (relational DBs, object DBs, Office-Systems...)
- Database Queries with ‘XQuery’
  - XPath-based - regarding document structure and content
- Simple administration
  - Browser-based control via any PC having Internet access
- Simple connection to Internet via standard Web-Server



Single Customer View
Self Service Portals
Supply Chain Integration
Business Reporting
Customer Solutions
XML Business Integration Solutions + Customer Solutions
Mobile Phone
PDA
Printer
Browser
CD
Enabling Services
Interactive Services
Software AG Integration Svcs
More ...
UDDI and Web Services
Application Programming
Schema Services
Enterprise Edition Services
External DB Services (opt.)
Databases
incl. Adabas
ODBC, Adabas
Core Services
XML, WML, HTML
XQuery, XPath
Obj. Processor & Obj. Composer
X-Tension Service
COM
Query / Text-Retrieval Service
XML Parser + Query Interpreter
Data / Metadata / non XML
Security Service
Data Map
XML
XML Schema Service
Native XML Data Store
Applications
Internet File System
Tokenizer (opt.) Chin.,Jap.,Kor.
Administration Services
Tamino Manager
Front Office / Clients
Back Office / Back End
Internet (HTTP, WebDAV, SOAP)


XML Output
Query (URL)
XML Objects, DTDs
XML-Engine
Query Interpreter
XML Parser
Object Processor
Object Composer
Utilities
Utilities
Data from external data sources and/or internal XML store
Data Map
Data to external data sources and/or internal XML store


XML Schema
API
Tamino X-Tension
custom Application
Application
XQuery
XML
Web Server
tamino XML Server
Data Map
existing DBMS

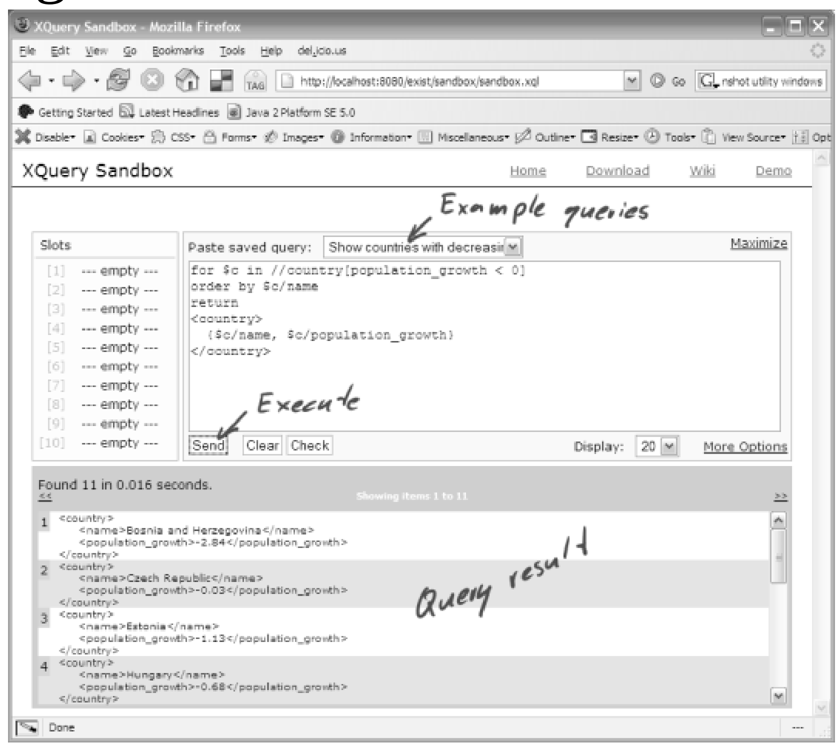
The eXist project

- eXist is the leading open source implementation of XML:DB
- It acts as a repository for indexing and retrieval of XML and RDF documents
- Uses a native java backend data store
  - Which splits out elements, attributes and entities into columns and associates tree path information
  - This allows for very high response times to the search queries
- The eXist project goes beyond simple XPath queries by adding functionality like NEAR based queries and document grouping (collections)

eXist DB

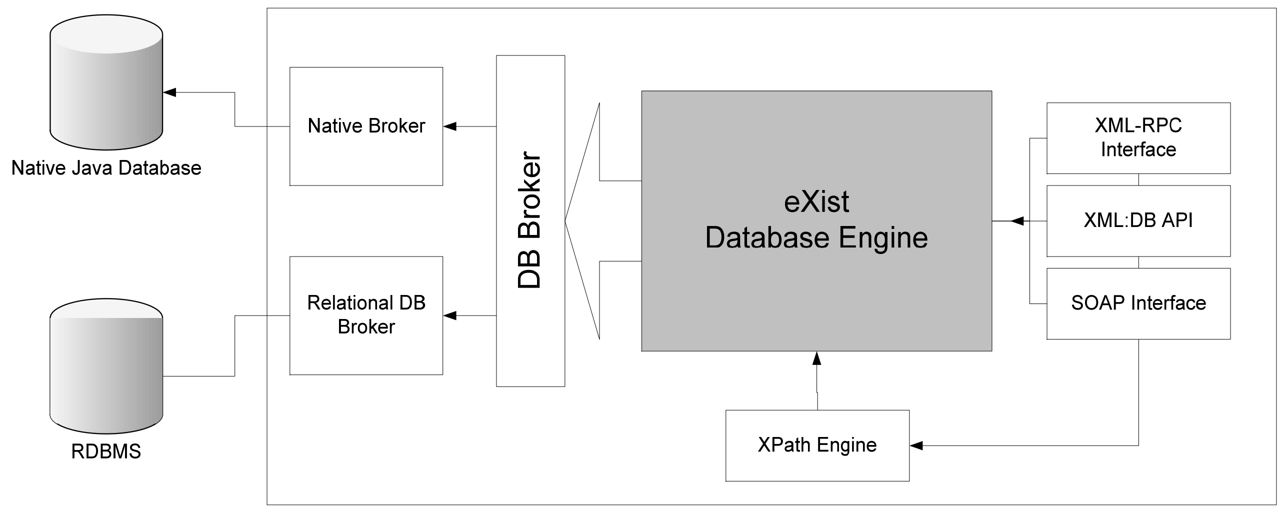
- Schema-less data storage
- Collections
- Index-based query processing
- XPath extension for performing full text search

- exist-db.org



eXist Architecture

- **eXist is split into three distinct areas**
  - Brokers, for accessing the data held in either the native Java data store or a relational DB like mySQL or Oracle
  - The engine itself, used to rebuild the documents and query the data store
  - Interfaces to the engine, either XML-RPC, the XML:DB API (Java) or SOAP for use within a Web Services framework



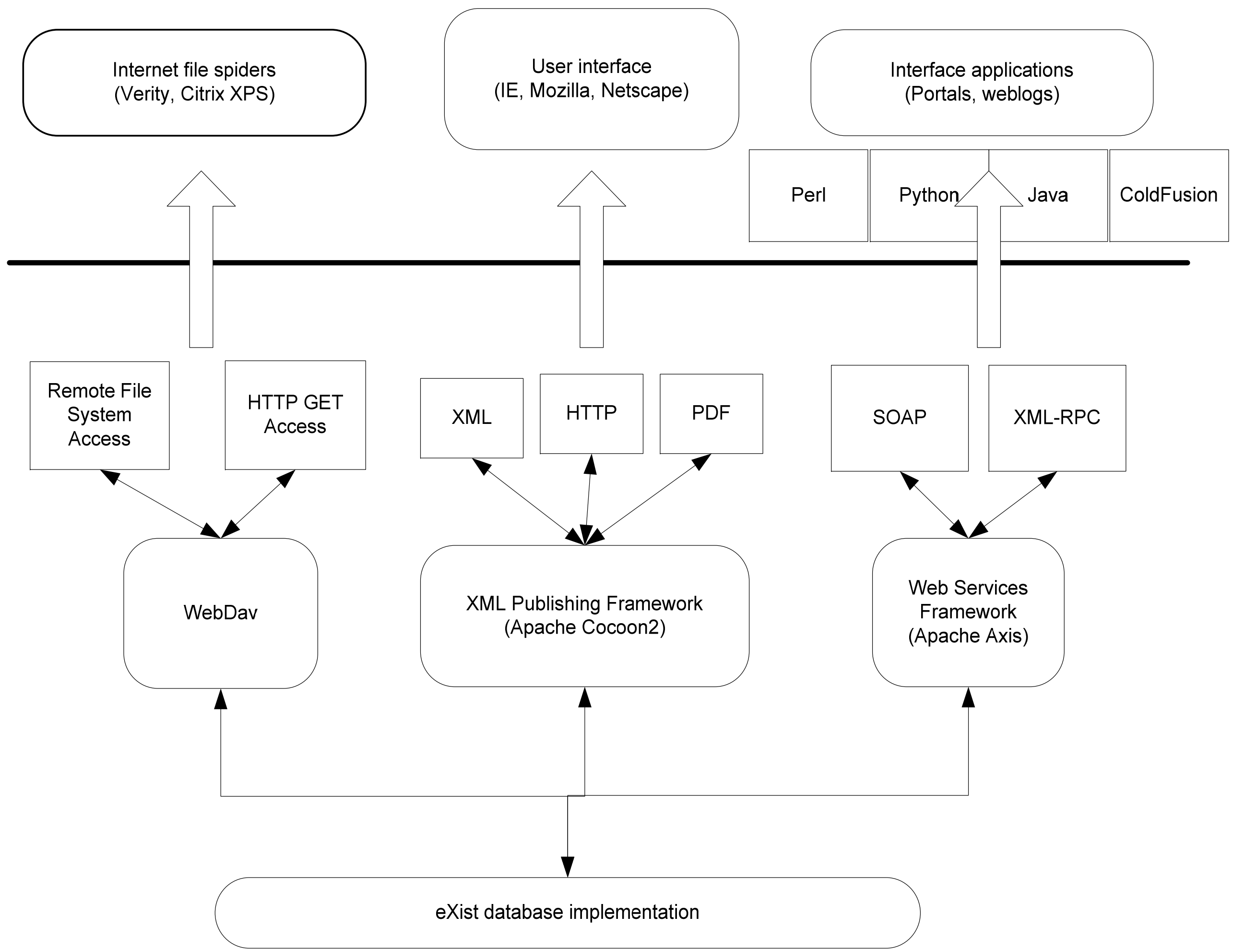
Building products using eXist

- There are three main options for building applications over eXist
  - Using WebDAV (Web-based Distributed Authoring and Versioning) access
  - Java applications, using the XML:DB API
  - Web Service applications (Built using Java, Python, Perl)
    - Web Services can then be used through any SOAP aware programming languages
    - XML-RPC Interfaces, available through most modern programming languages
- A natural fit for XML applications built on eXist is using Apache’s Cocoon as the presentation layer of your application

xml.apache.org

- The Apache XML Project has activities focused on different aspects of XML
  - Xerces - XML parsers in Java, C++ (with Perl and COM bindings)
  - Xalan - XSLT stylesheet processors, in Java and C++
  - Cocoon - XML-based web publishing, in Java
  - FOP - XSL formatting objects, in Java
  - Xang - Rapid development of dynamic server pages, in Java
  - Indice
    - Native XML database

Building products using eXist



Database Features

- **Data Storage**
  - Native XML data store based on B+-trees and paged files. Document nodes are stored in a DOM tree.
- **Collections**
  - Documents are managed in hierarchical collections, similar to storing files in a file system
- **Updates**
  - Document-level and node-level updates.
- **Authorization Mechanism**
  - Unix-like access permissions for users/groups at collection- and document-level.

Database Features

- **Multi-User Access**
  - Concurrent read/write access supported. Database manages concurrency at the level of the basic database operations.
- **Deployment**
  - eXist may be deployed as a stand-alone database server, as an embedded Java library or as part of a web application (running in the servlet engine).
- **Backup/Restore**
  - Backup/restore functionality is provided via Java admin client or Ant scripts. Allows full restore of a database including user/group permissions.
- **XML Standards**
  - XPath, Xquery, XUpdate, XSL/XSLT.
- **Network Protocols:** HTTP/REST, XML-RPC, SOAP, WebDAV

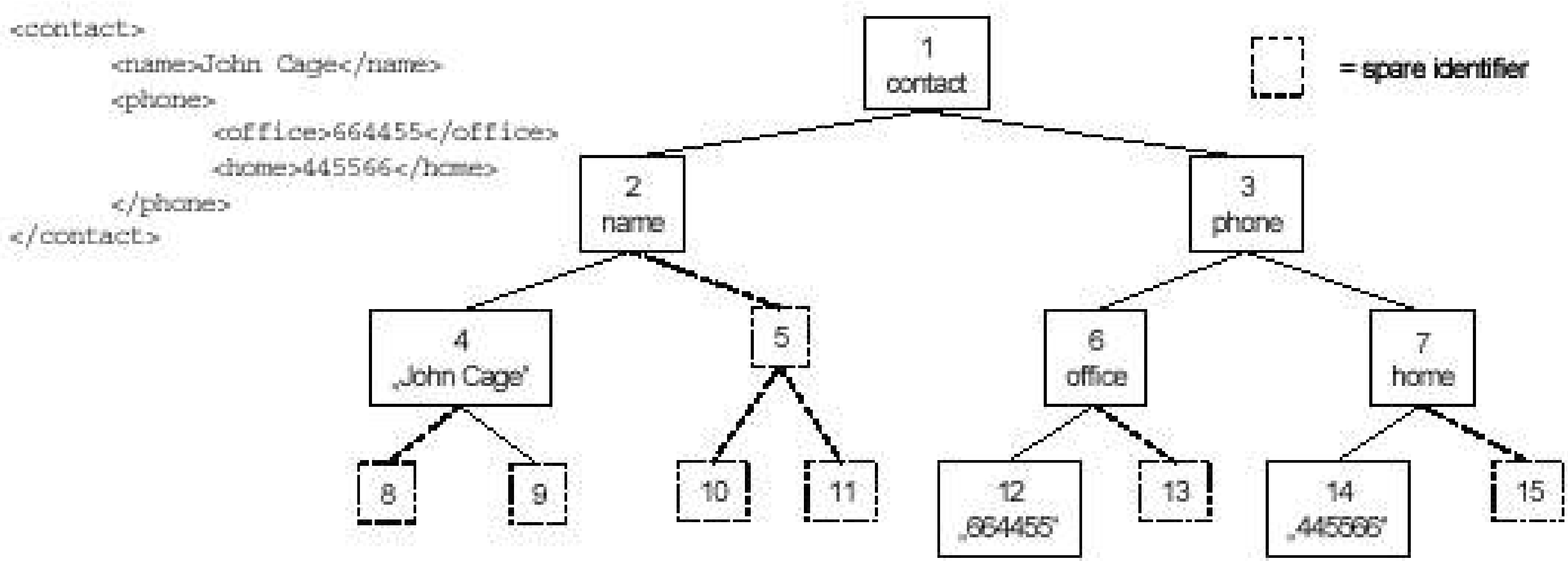
Indexing

Virtual nodes

$$Parent(i) = \left\lfloor \frac{(i-2)}{k} + 1 \right\rfloor$$

Complete K-ary tree

Fig. 1. Unique identifiers assigned by the level-order numbering scheme

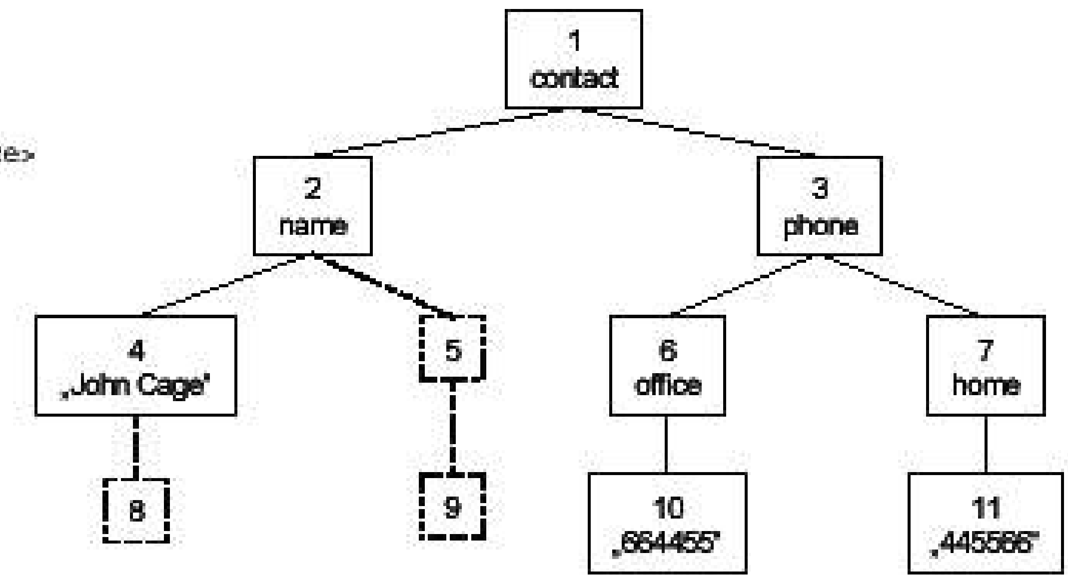


Indexing

- Too high numbers => document size limitation => drop completeness
  - For 2 nodes x and y of a tree, size(x) = size(y) if level(x) = level(y), where size(n) is the number of children of a node n and level(m) is the length of the path from the root node of the tree to m

Fig. 2. Unique node identifiers assigned by the alternating level-order numbering scheme

```
<contact>
        <name>John Cage</name>
        <phone>
                <office>664455</office>
                <home>445566</home>
        </phone>
</contact>
```

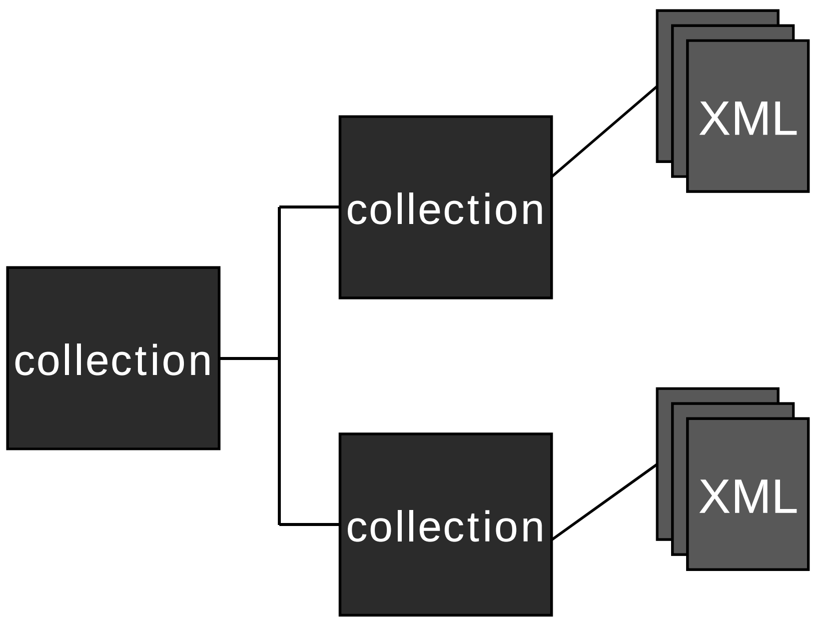


Storage Implementation

- Storage backend
  - dom.dbx collects DOM nodes in a paged file and associates node identifiers to the actual nodes
  - collections.dbx manages the collection hierarchy
  - element.dbx indexes elements and attributes
  - words.dbx keeps track of word occurrences and is used by the full text search extensions

collections.dbx

- Manages the collection hierarchy and maps collection names to collection objects.
- An important point to note is that the indexes for elements, attributes and keywords are organized by collection and not by document

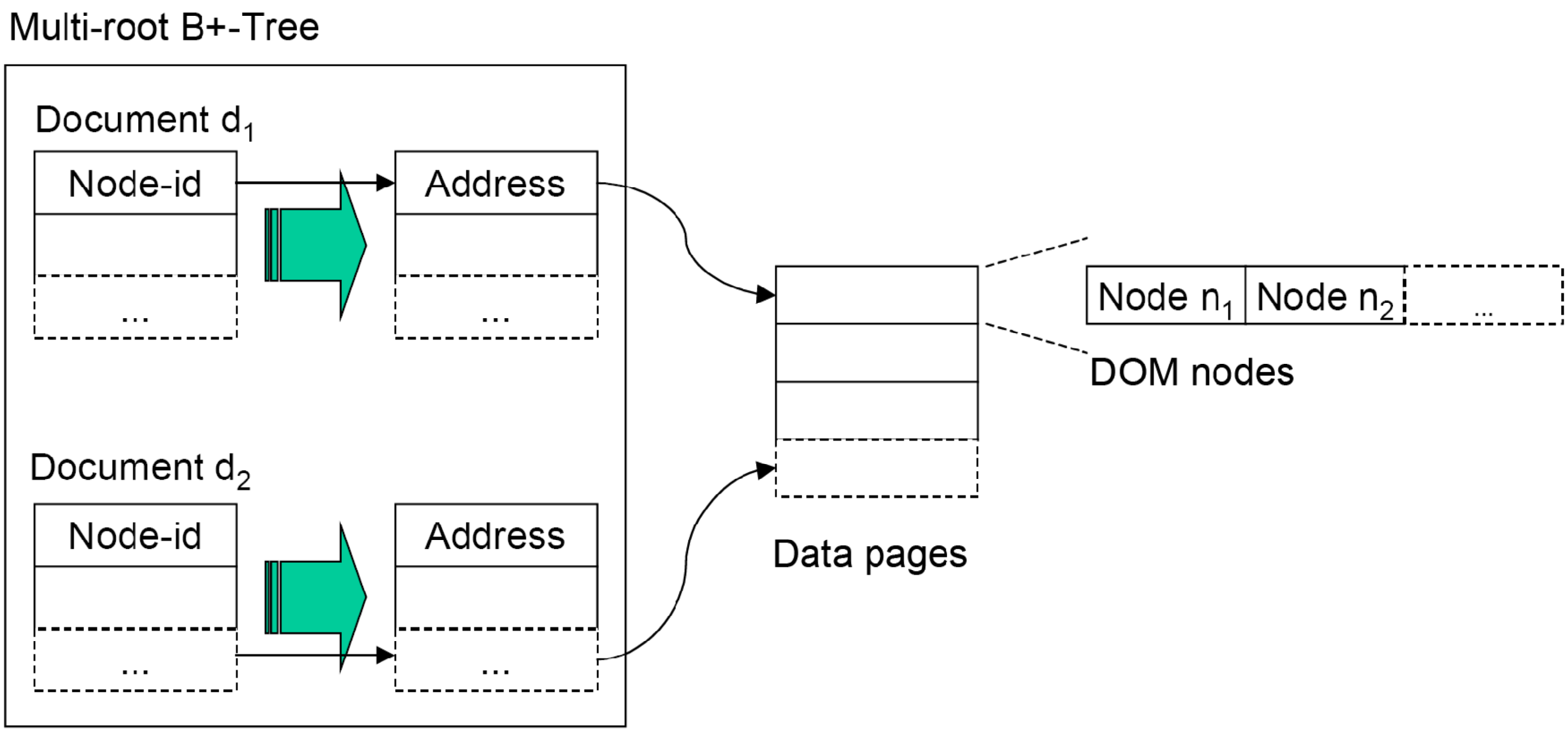


dom.dbx

- The XML data store (dom.dbx) represents the central component of eXist's native storage architecture.

- All document nodes are stored according to the W3C's DOM (Document Object Model).
- The data store is backed by a multi-root B+-Tree in the same file.
- Associate the unique node identifiers of top-level elements in a given document to the node’s storage address in the data pages.

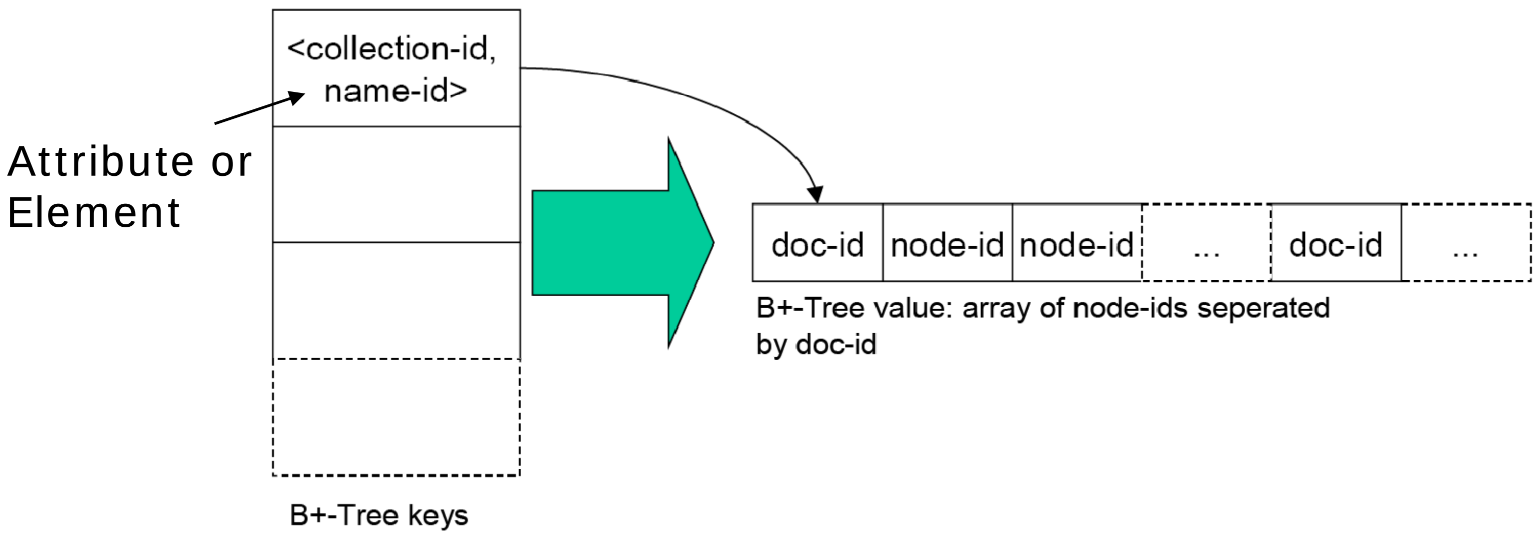
dom.dbx



- However, the query engine will process most types of XPath without accessing dom.dbx.

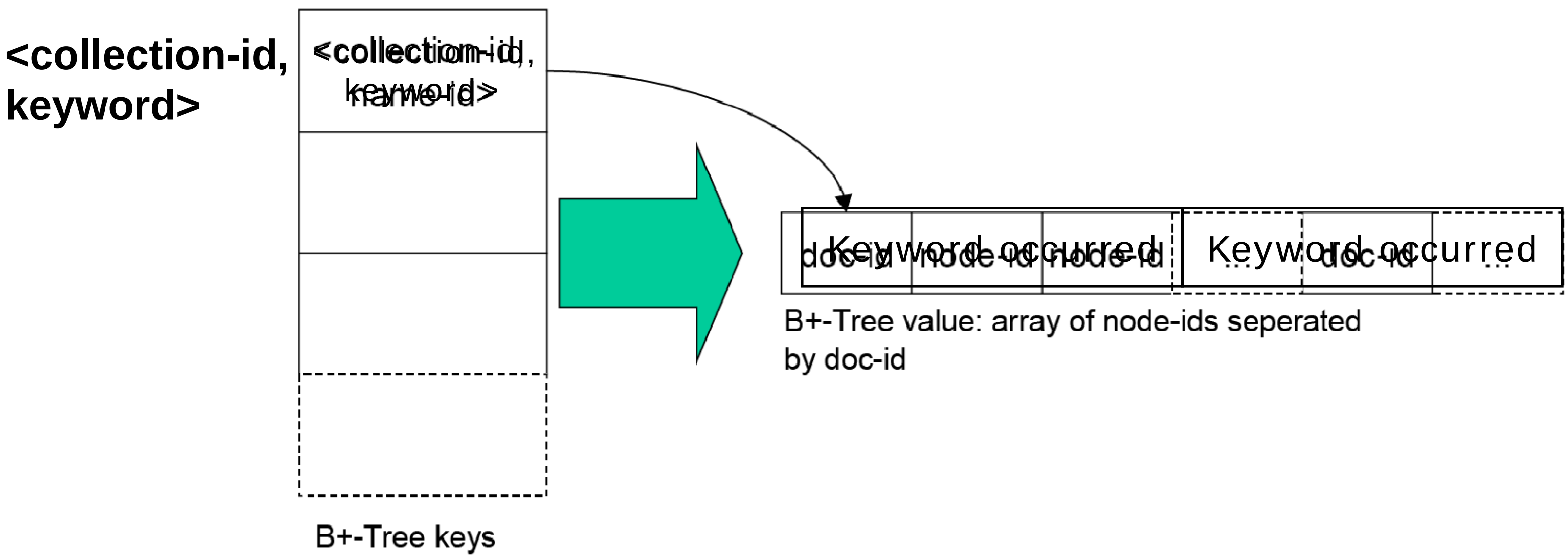
elements.dbx

- Element and Attribute names are mapped to unique node identifiers in file elements.dbx.
- Each entry in the B$^+$-tree index consists of a key - <*collection-id, name-id*>
- And each key correspond an array containing an list of <*document-id, node-id*>



words.dbx

- words.dbx corresponds to an *inverted index* as found with the set of documents in which it has been found and the exact position where it occurred
- Using<*collection-id, keyword*> pair for key, like element.dbx
- Each entry in the value list points to a text or attribute node where the keyword occurred.



XPath Extension

- document()
  - Selection of a document, a set of documents, or all
- collection()
  - Specification of a collection of documents to be included in query evaluation
  - collection('/db/vincent')//scence[speech[speaker="David"]]/title

XPath Extension

- Querying text
  - XPath only have a few functions in text searching
  - contains(), near(), &=, |=, match-all()
- //chapter[contains(., ‘XML’) and contains(., ‘database)]
  - Find a chapter that its content contains the words ‘XML’ and ‘database’
- //section[near(., ‘XML database’, 50)]
  - Find sections containing both keywords in the correct order and with less than 50 words between them
- //scene [speech [&= ‘witch’ and line &= ‘fenny snake’]]
- //speech [match-all [line, ‘li[fv]e[s]’)]

Query Execution

- Basic approach
  - Top-down/bottom-up traversal for XPath expression
  - Very inefficient
    - /book//section[contains (title, ‘XML’)]
    - Follow every child path beginning at BOOK to check for potential SECTION descendants
- Indexing structure
  - Efficient processing of regular path expressions on large, unconstrained document collections

Query Processing

- Decompose a path into a chain of basic steps
  - **/PLAY//SPEECH [SPEAKER=`HAMLET`]**

Fig. 5. Decomposition of Path Expression

- Load the root element (PLAY) for all documents in the input document set
- The set of SPEECH elements is retrieved for the input documents via an index lookup from file element.dbx
- Use an ancestor-descendant path join algorithm to join the two sets
- Evaluate the predicate

Advantages of eXist

- Advantages of eXist as a Native XML DB
  - eXist Provides a scalable, reliable XML database implementation royalty free
  - By including multiple different interfaces eXist makes application integration simple
  - With the addition of other open source platforms (like Cocoon for presentation and Axis for Web Services) eXist can be extended easily to fit many application needs
  - eXist is being used currently in many live implementations

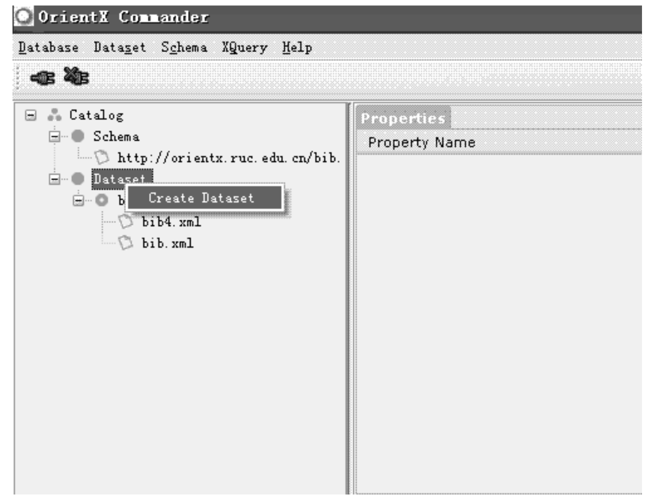
Drawbacks

- Drawbacks of eXist as a Native XML DB
  - With open source, documentation is scarce, you will have to rely on mailing lists for the more difficult problems
  - Warranty is not included from source, although it can be given by a third party
  - Although stable, eXist is still considered beta software
  - Some issues still surround parts of the XPath implementation

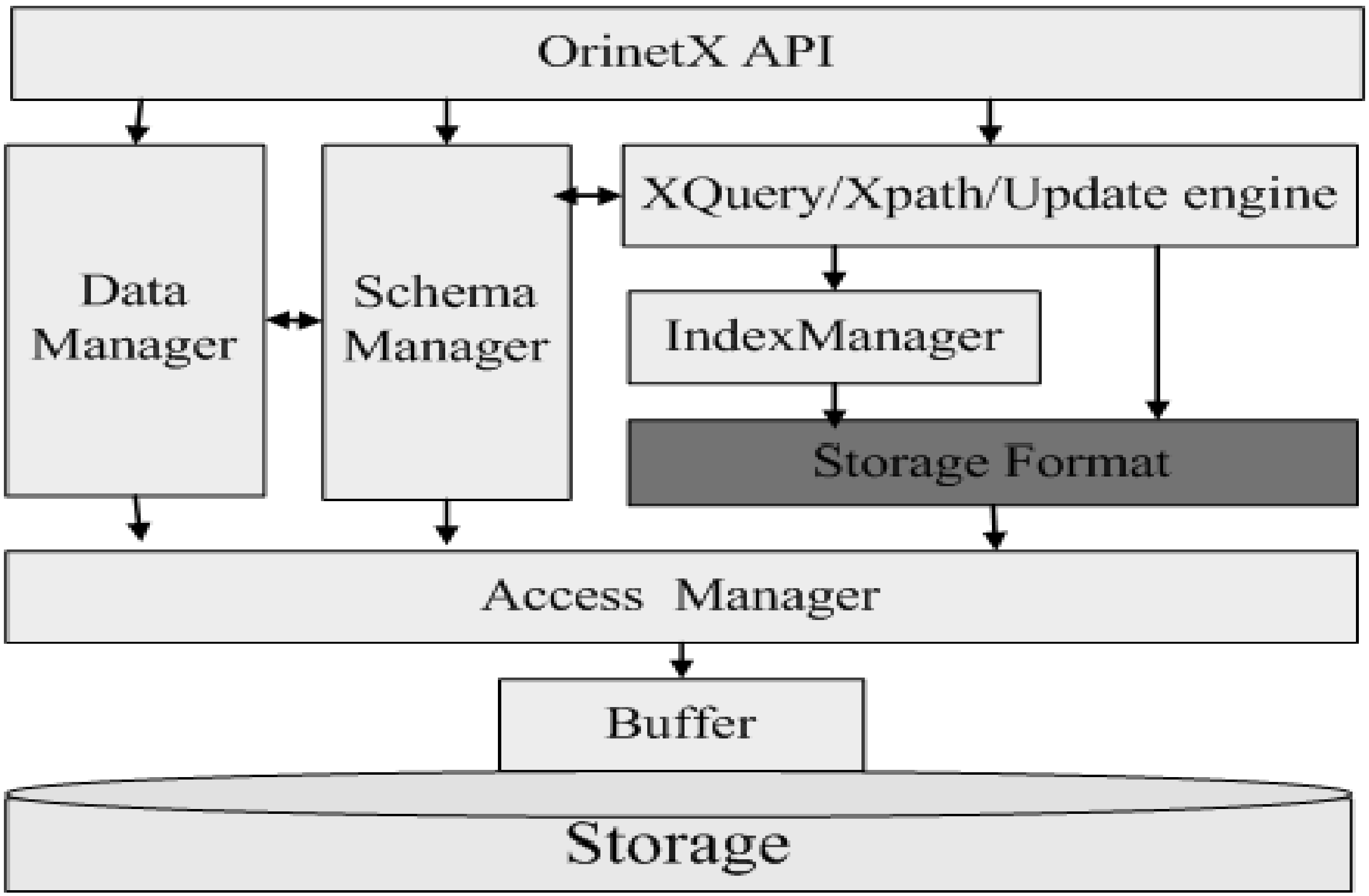
OrientX

- A Schema-based Native XML Database
  - http://idke.ruc.edu.cn/OrientX
- OrientX means:

  Original **R**UC **I**DK**E** **N**a**t**ive **X**ML Database
  - RUC: Renmin University of China
  - IDKE: Institute of Data and Knowledge Engineering
  - Native XML DataBase: Exposing a logical model of storing and retrieving XML documents (vs non-Native XML Database: for example, based on relation database)
- Latest version 3.5 (2009)

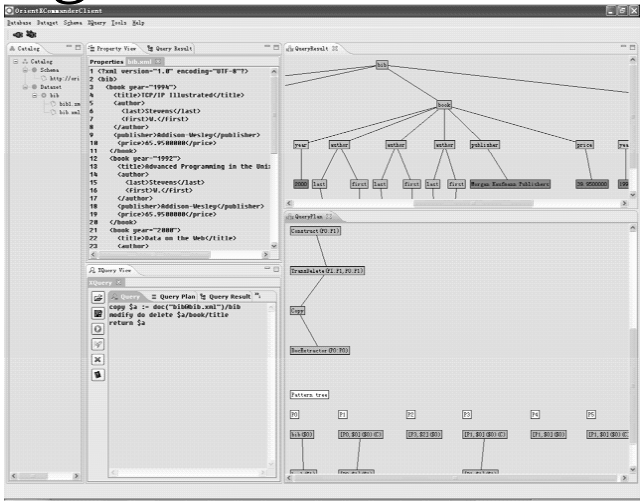


Architecture



Features

- Full support to XML Schema
- Supporting XQuery1.0 and XPath2.0 Data Model
- Various native storage techniques
- Path index and value index
- Multi-Query Processing strategies based on native storage.



Different Storage Granularities

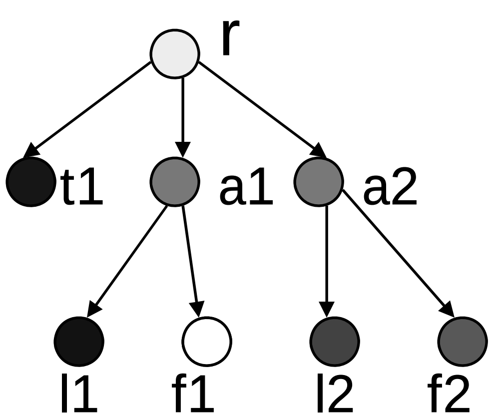
- Document:
  - do not decompose the document, build index on it to direct the structure.
  - Query complexity and efficiency are restricted by the power of index.
- Sub tree:
  - decompose the document into sub trees according to storage space partition.
  - Persistent the structure in the tree.
  - save space
- Node:
  - decompose the document into nodes sequence , each node corresponding to a type (element, attribute, …).
  - May use too many links to persistent relation between nodes

Storage Techniques in OrientX

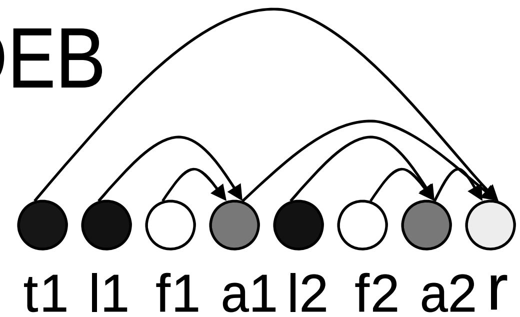
	DEB	**DSB**	
	CEB	**CSB**	

Implemented techniques are marked in red

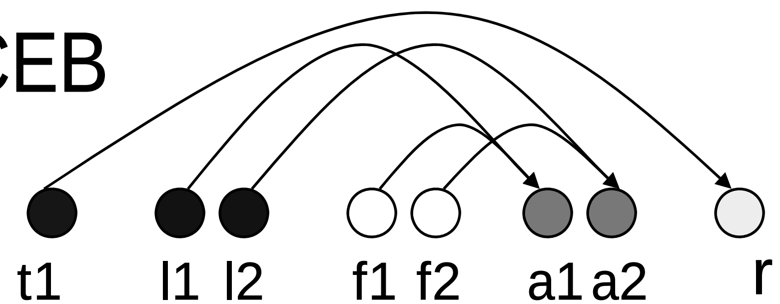
Example-- Element based



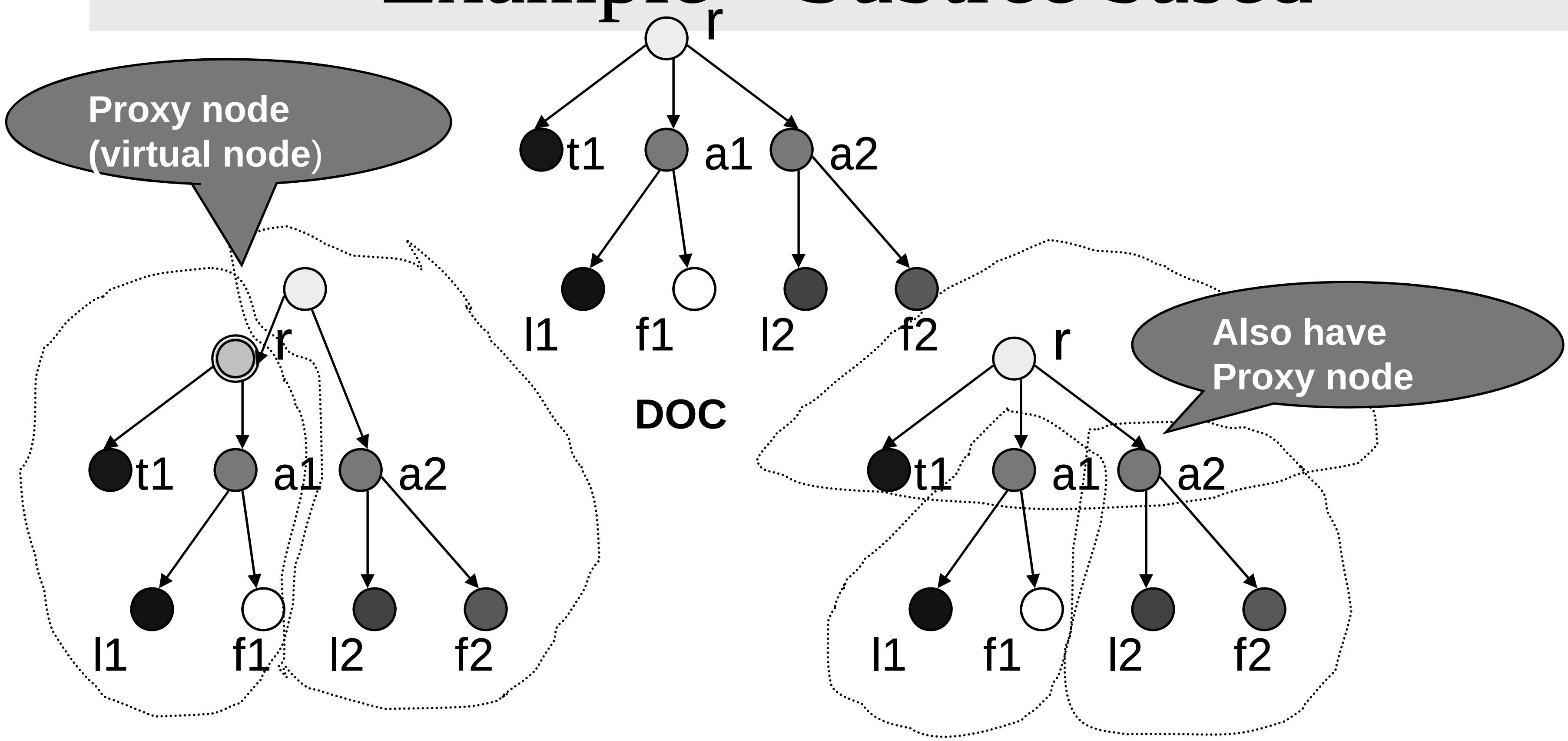
- DEB



- CEB



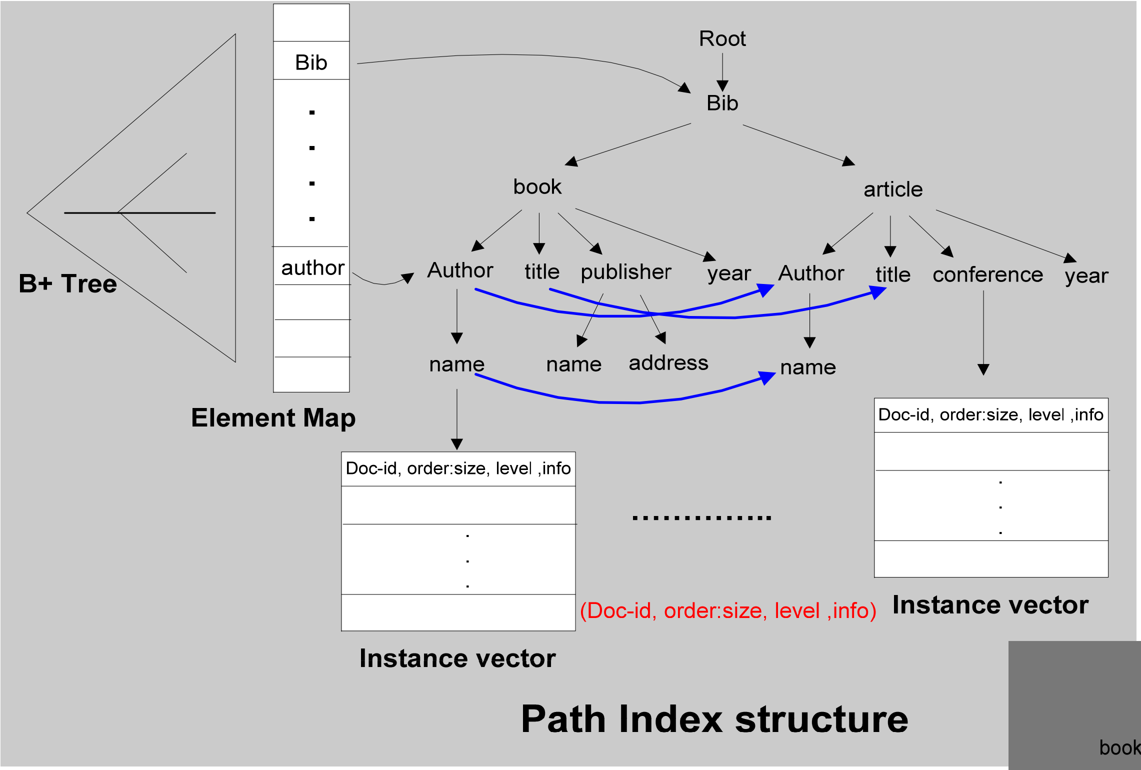
Example-- Subtree based



DSB(Depth-first sub-tree based)

CSB (clustered sub-tree based)

Index Architecture



Bib//author

Or //author