Module: XML et les bases de données

Les SGBD natifs XML (XDB)

Houda Chabbi Drissi

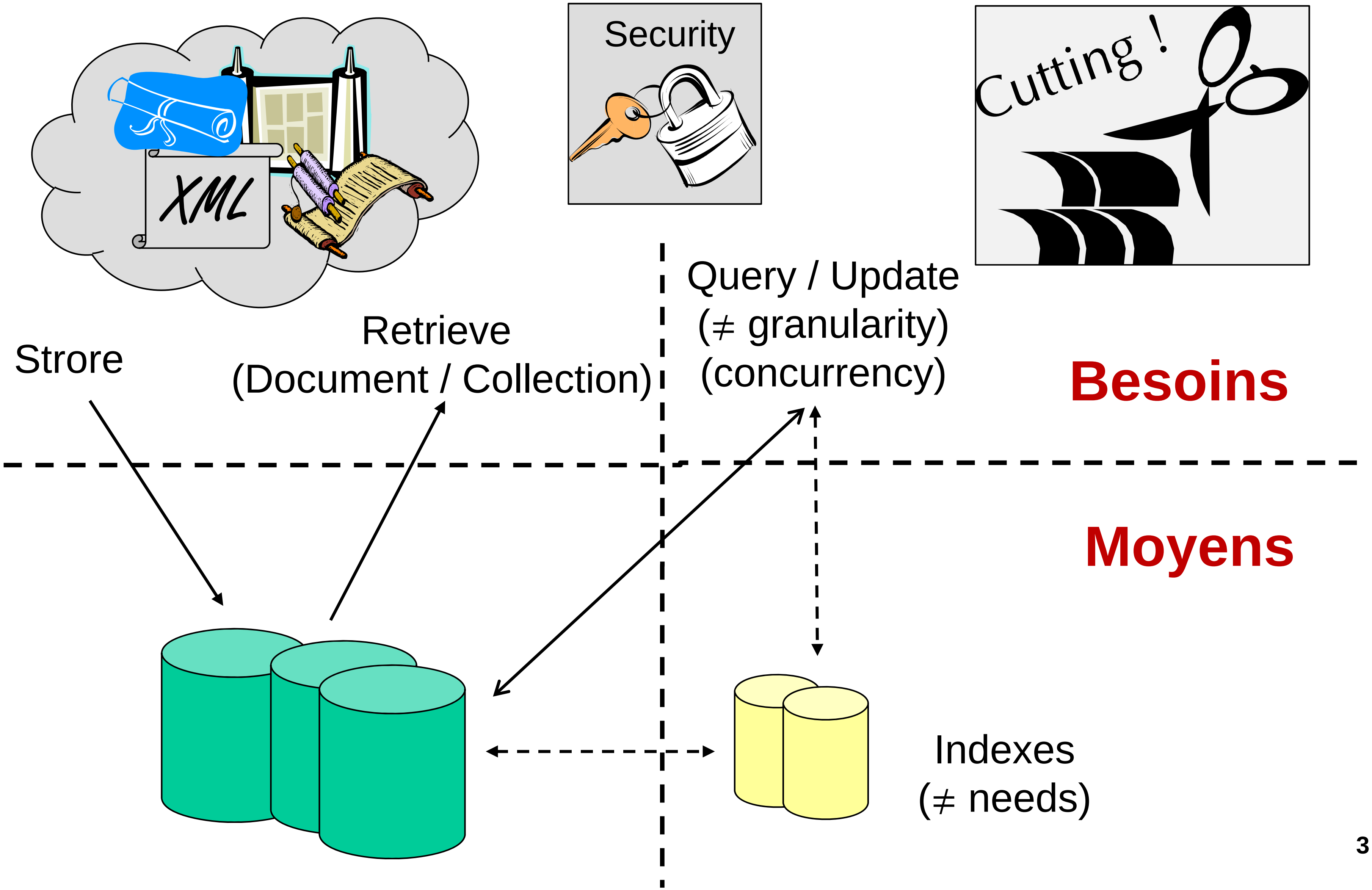
houda.chabbi@hefr.ch

Source: *georges.gardarin.free.fr/Cours_Total/X5-BD_XML.ppt*

Plan

- ▶Motivation – Needs – Means.
- NXD Concepts
- Indexing approaches
- Conclusion

Motivation



Security

Strore

Retrieve
(Document / Collection)

Query / Update
(≠ granularity)
(concurrency)

Besoins

Moyens

Indexes
(≠ needs)

Users' database needs

- Store/Retrieve - global/parts of XML
- Query/Update - global/parts of XML
- Independency: Logical model versus Physical model
- Optimization
- ACID Transactions
- Recovery
- Security

Means

- Choose logical/physical models for storage/retrieval’s mechanism.
- Query/Update languages.
- Optimization: indexing, caching.
- Concurrency handling : ACID.
- Recovery approaches.

Databases’ solutions

Plan

- Motivation – Needs – Means.
- ▶ NXD Concepts
- Indexing approaches
- Conclusion

Native XML Data Stores

- Provides a logical model for storing and retrieving XML documents efficiently
- XML interface for accessing XML data (e.g. XPath, XQuery)
- Advantages
  - Scalability
  - Data-access speed
  - Reliability

Native XML Database

- Definition (XML:DB Initiative – www.xmldb-org/sourceforge.net)
  - XML document is the fundamental unit of logical storage
  - Uses a logical model for the XML document itself
  - Requires a particular underlying physical storage model

Abstract View of Components

- Native XML Interface
  - Most popular choices are XQuery and Xpath
- Storage Manager
  - Performs any necessary transformations to store/retrieve XML data

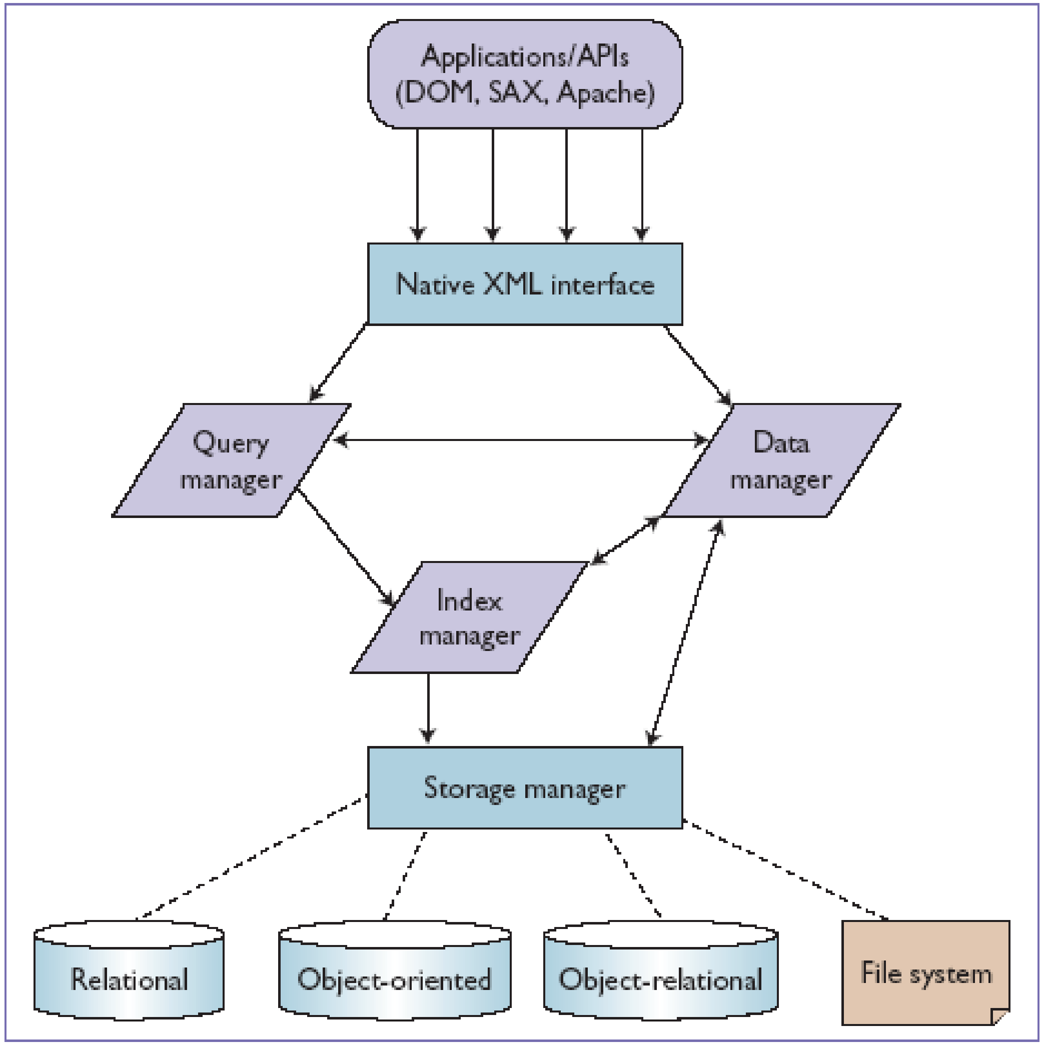
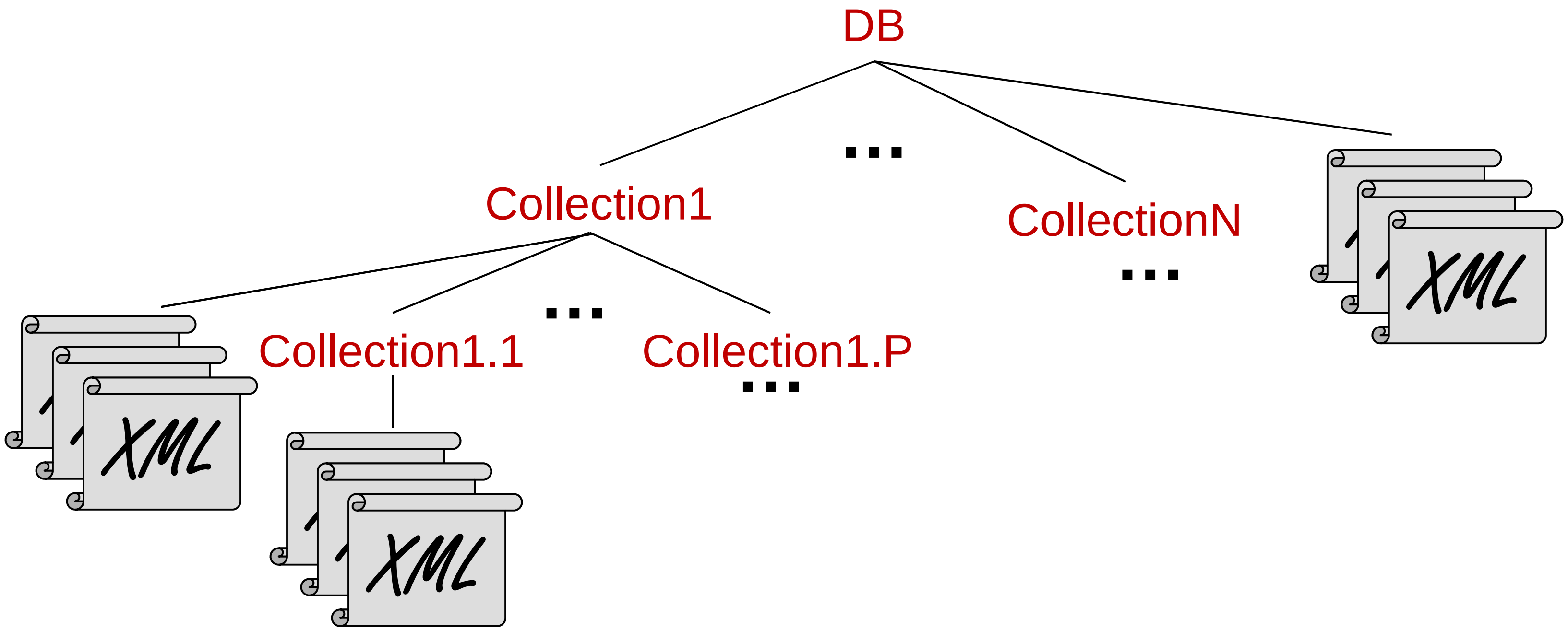


Figure 1. Abstract view of a native XML data management framework. The store includes two key components: a native XML interface and a storage manager.

Vocabulary: XML Collections

- XML Collection is a collection of XML documents
- “A row is to a table as an XML document is to an XML collection.”



Collections

NXDs manage collections of documents, allowing to query and manipulate those documents as a set.

→ similar to the relational concept of a table.

NXDs diverge from the table concept in that not all native XML databases require a schema to be associated with a collection: any XML document can be store in the collection, regardless of schema.

→ can construct queries across all documents in the collection.

Schema-independent NXDs

☺ Having schema-independent document collections gives the database a lot of flexibility and makes application development easier.

☹Unfortunately, the risk of low data integrity is high.

→If strong schema structure is needed, then:

- either make sure you use a NXD that supports schemas
- or find another way to store your XML data.

Queries / Updates: standard languages

- Xquery is the NXD query langage

- Xupdate is the NXD update langage

- Xquery Full Text fusion between IR and databases concepts.

Plan

- Motivation – Needs – Means.
- NXD Concepts
- ▶ Indexing approaches
- Conclusion

Indexing characteristics

- Index Type:
  - ▶ Structure + implementation example
  - value,
  - full-text.
- Index Scope:
  - document, collection.
- Index Target:
  - document, node.
- Index Control:
  - automatic, voluntary, required.

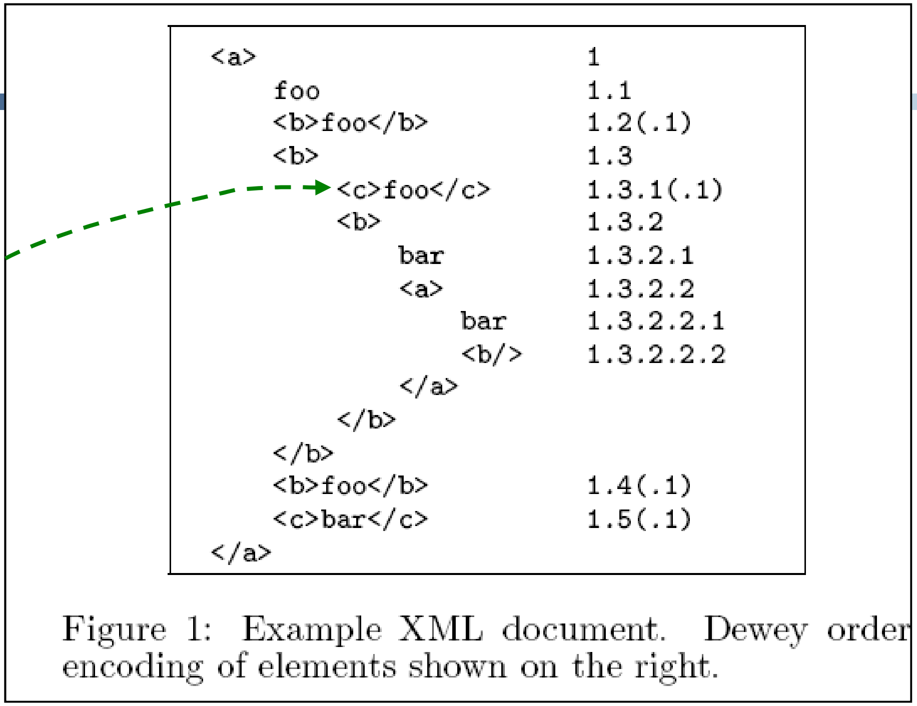
Index type: structural indexes or path indexes

Structural indexes materialize results of path expressions. Useful for navigational portions of queries.

Used to answer to PATH, such as "/a/b/c".

Implementation: A(k) indexes

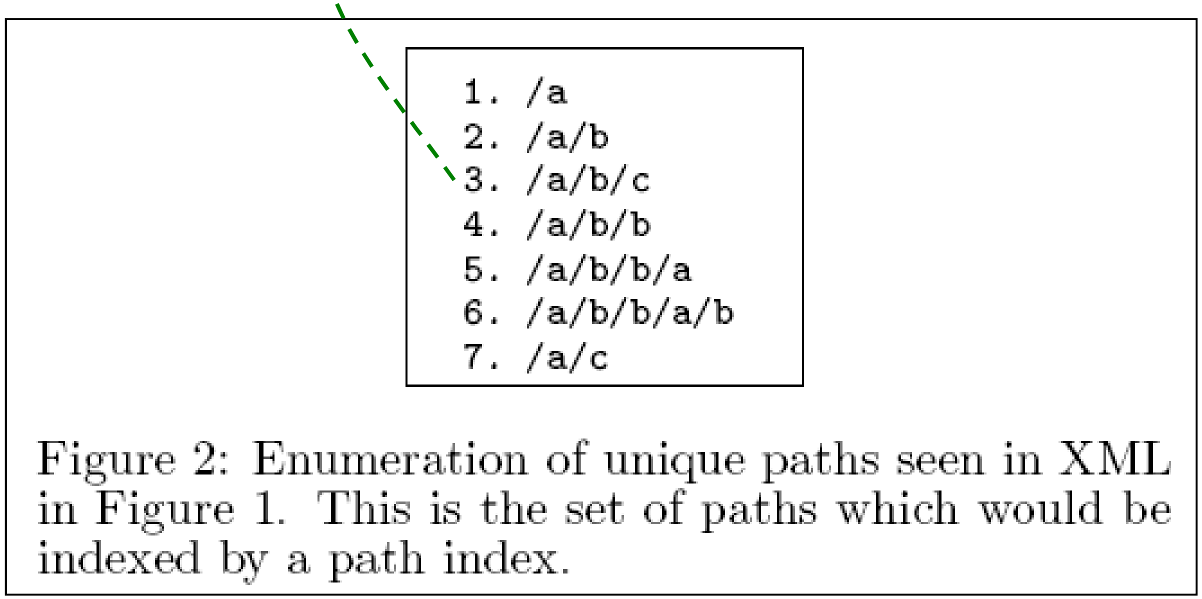
* http://www.idi.ntnu.no/~nilsgri/docs/Gri08PathIndex.pdf



```
<a>                            1
    foo                        1.1
    <b>foo</b>                 1.2(.1)
    <b>                        1.3
        <c>foo</c>             1.3.1(.1)
        <b>                    1.3.2
            bar                1.3.2.1
            <a>                1.3.2.2
                bar            1.3.2.2.1
                <b/>           1.3.2.2.2
            </a>
        </b>
    </b>
    <b>foo</b>                 1.4(.1)
    <c>bar</c>                 1.5(.1)
</a>
```

Figure 1: Example XML document. Dewey order encoding of elements shown on the right.

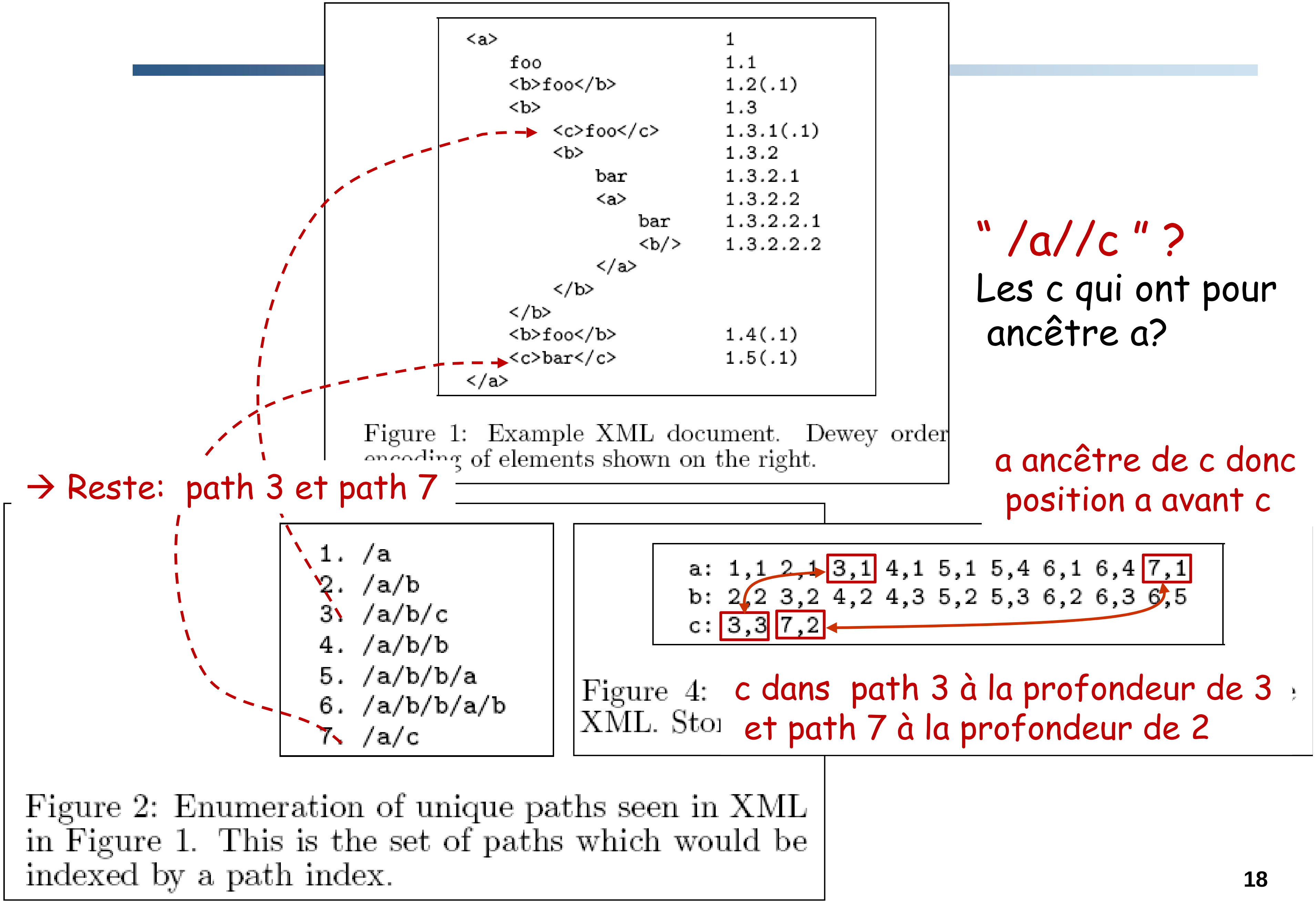
"/a/b/c"



```
1. /a
2. /a/b
3. /a/b/c
4. /a/b/b
5. /a/b/b/a
6. /a/b/b/a/b
7. /a/c
```

Figure 2: Enumeration of unique paths seen in XML in Figure 1. This is the set of paths which would be indexed by a path index.

* http://www.idi.ntnu.no/~nilsgri/docs/Gri08PathIndex.pdf



```
<a>                    1
    foo                1.1
    <b>foo</b>         1.2(.1)
    <b>                1.3
        <c>foo</c>     1.3.1(.1)
        <b>            1.3.2
            bar        1.3.2.1
            <a>        1.3.2.2
                bar    1.3.2.2.1
                <b/>   1.3.2.2.2
            </a>
        </b>
    </b>
    <b>foo</b>         1.4(.1)
    <c>bar</c>         1.5(.1)
</a>
```

Figure 1: Example XML document. Dewey order encoding of elements shown on the right.

→ Reste: path 3 et path 7

```
1. /a
2. /a/b
3. /a/b/c
4. /a/b/b
5. /a/b/b/a
6. /a/b/b/a/b
7. /a/c
```

Figure 2: Enumeration of unique paths seen in XML in Figure 1. This is the set of paths which would be indexed by a path index.

```
a: 1,1 2,1 3,1 4,1 5,1 5,4 6,1 6,4 7,1
b: 2,2 3,2 4,2 4,3 5,2 5,3 6,2 6,3 6,5
c: 3,3 7,2
```

Figure 4:
XML. Sto

c dans path 3 à la profondeur de 3
et path 7 à la profondeur de 2

" /a//c " ?

Les c qui ont pour ancêtre a?

a ancêtre de c donc position a avant c

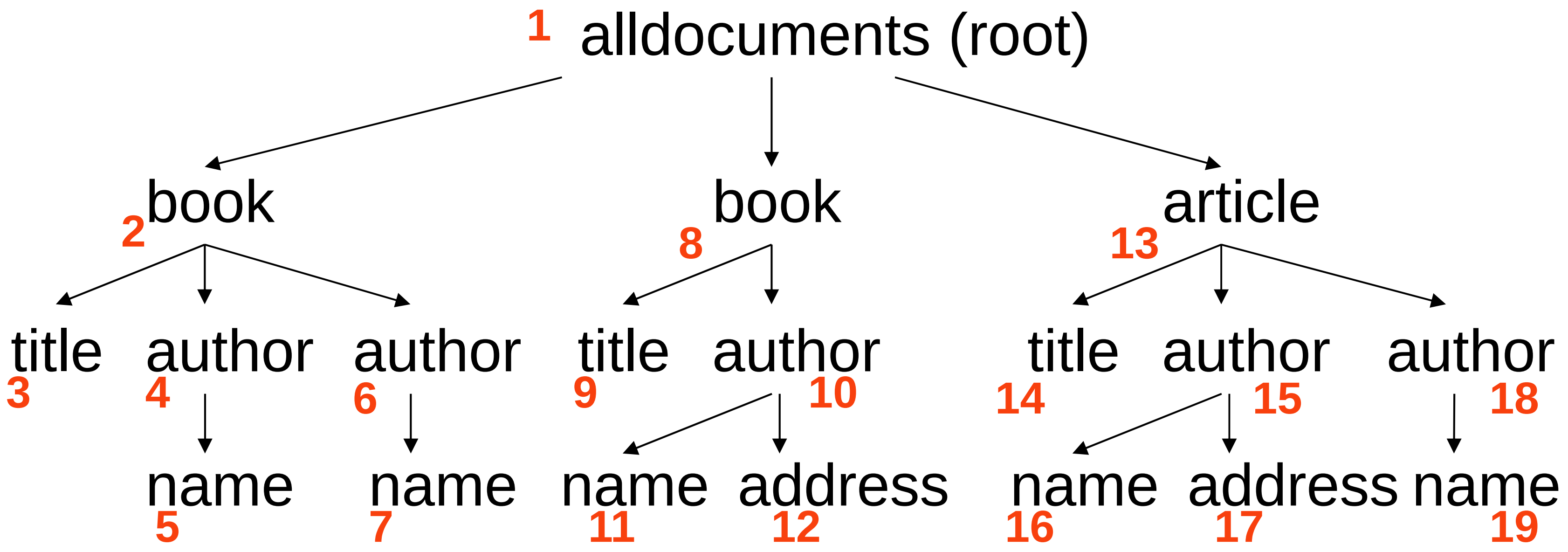
Bisimilarity

- Two nodes u, v are called bisimilar if:
  - They have the same label.
  - If u’ is the parent of u, then there is a parent v’ of v, such that u’,v’ are also bisimilar, and vice versa.

⇒ two nodes are bisimilar *if for every finite path starting* at one node there is a path with identical sequence of labels starting from the other.

Définition récursive. En fait, les deux nœuds sont atteignables par un même chemin depuis la racine.

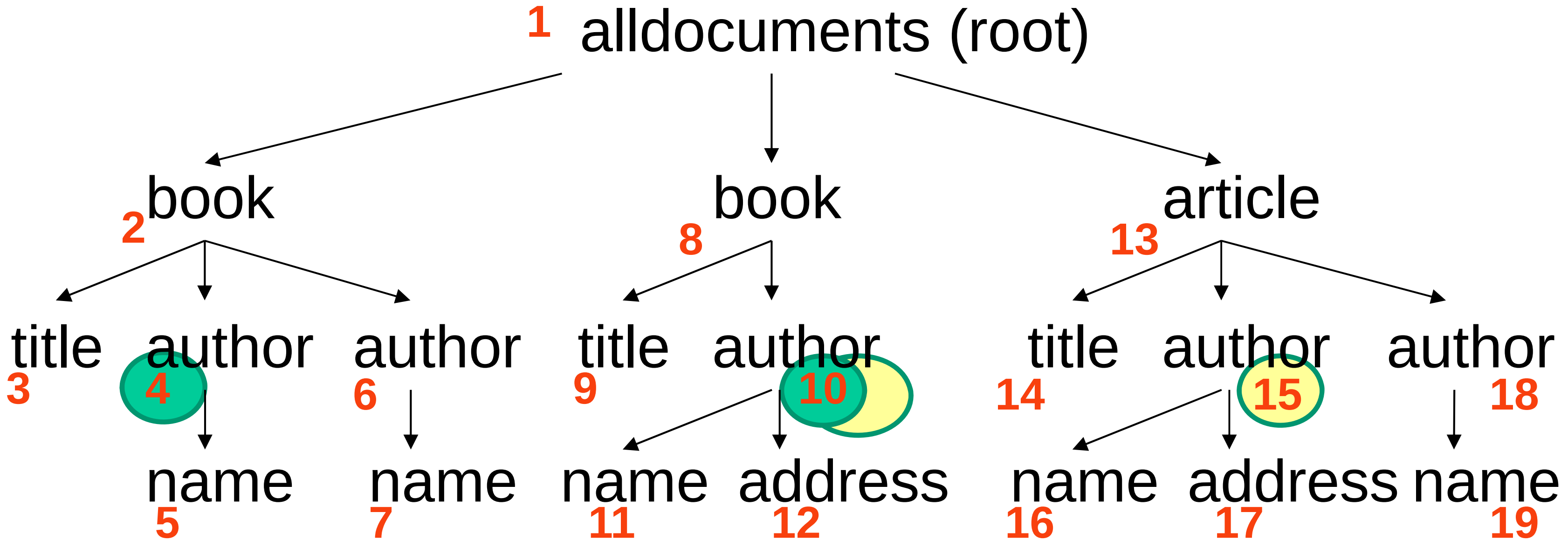
XML example: data graph



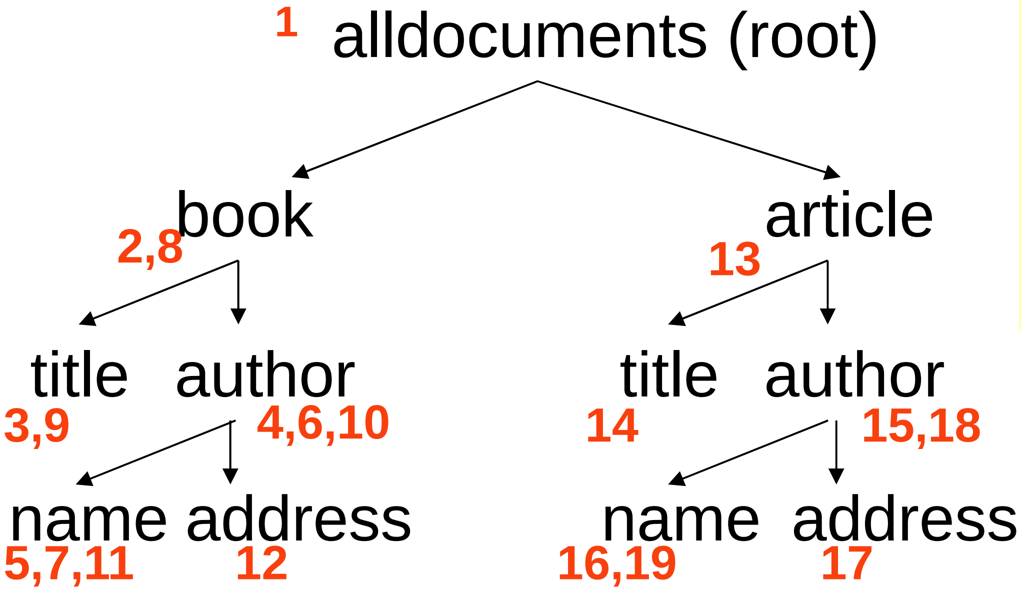
Source: http://i.cs.hku.hk/~nikos/courses/CSIS7101/intro_slides.pdf

Bisimilarity examples

- Nodes 4 and 10 are bisimilar
- Nodes 10 and 15 are not bisimilar



1-index graph



/alldocuments/book/title
/alldocuments/book/author
/alldocuments/book/author/name
/alldocuments/book/author/address
/alldocuments/article/title
/alldocuments/articleauthor
/alldocuments/article/author/name
/alldocuments/article/author/address

- Index summarizes path information
- Each entry: list of pointers to data nodes

Path indexes for XML data

- The 1-index maintains information about all paths in the original graph.
- This makes **the index very large** (with size comparable to the data size) ⇒ quite expensive to evaluate queries using this index.
- To address this problem an A(k)-index is proposed which indexes exactly only paths up to length k.

K-bismilarity

- Two nodes u,v are 0-bisimilar, if they have the same label.

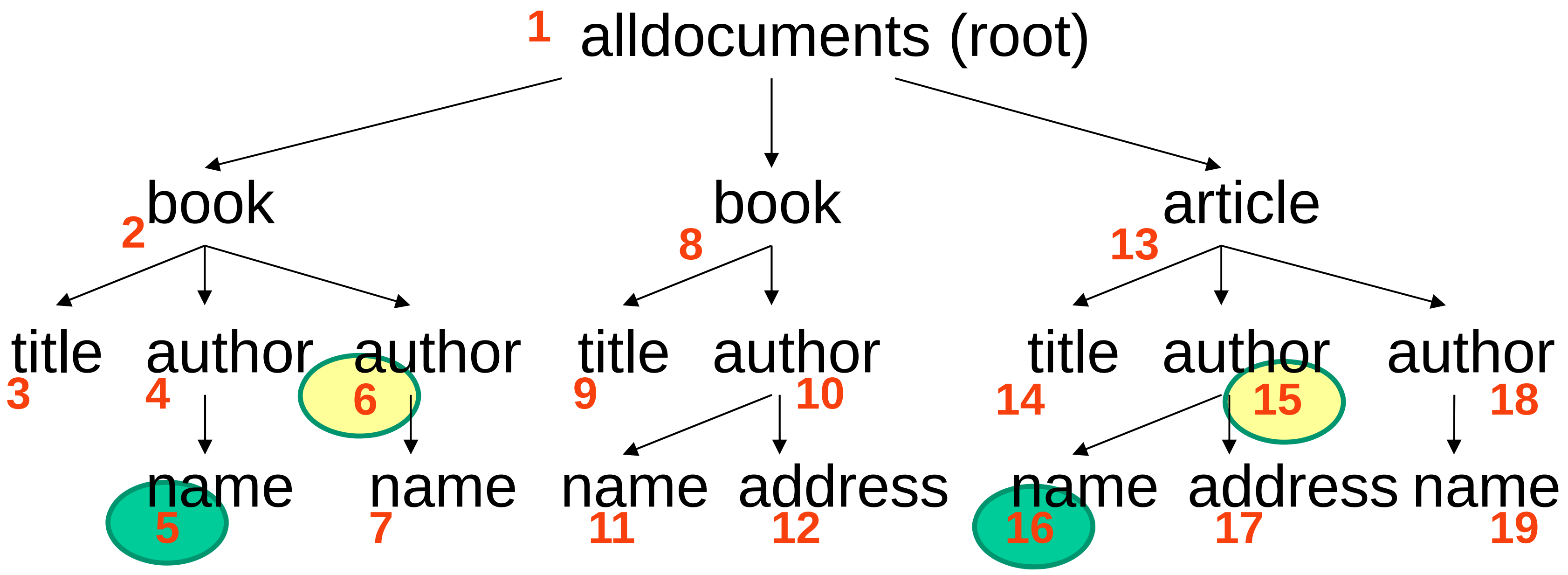
- Two nodes u,v are k-bisimilar,
  - if they have the same label
  - and for every parent u' of u, there is a parent v' of v, such that u' and v' are (k-1)-bisimilar, and vice versa.

Properties

- If a and b are bisimilar
  - set of incoming paths into them is same (from root)
- *If* a and b are 0-bisimilar
  - They have the same label
- If a and b are k-bisimilar
  - set of incoming paths of length <= k are same

k-bisimilarity Example

- Nodes 5 and 16 are 1-bisimilar
- Nodes 6 and 15 are not 1-bisimilar



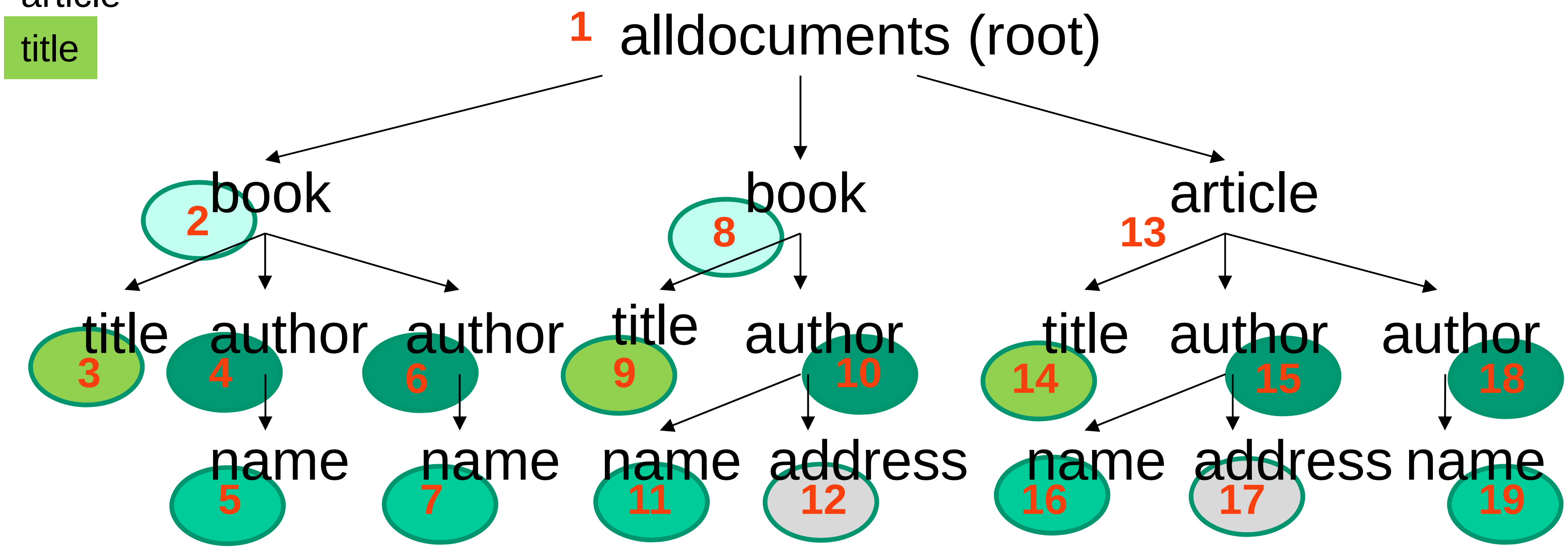
Bisimilarity and the A(k)-index

- The *A(k)-index* stores exactly all paths of length k,or else: all *k-bisimilar nodes* in the data graph are stored in the same node in the index graph.

- This means that all incoming paths up to length *k* are encoded in the index.

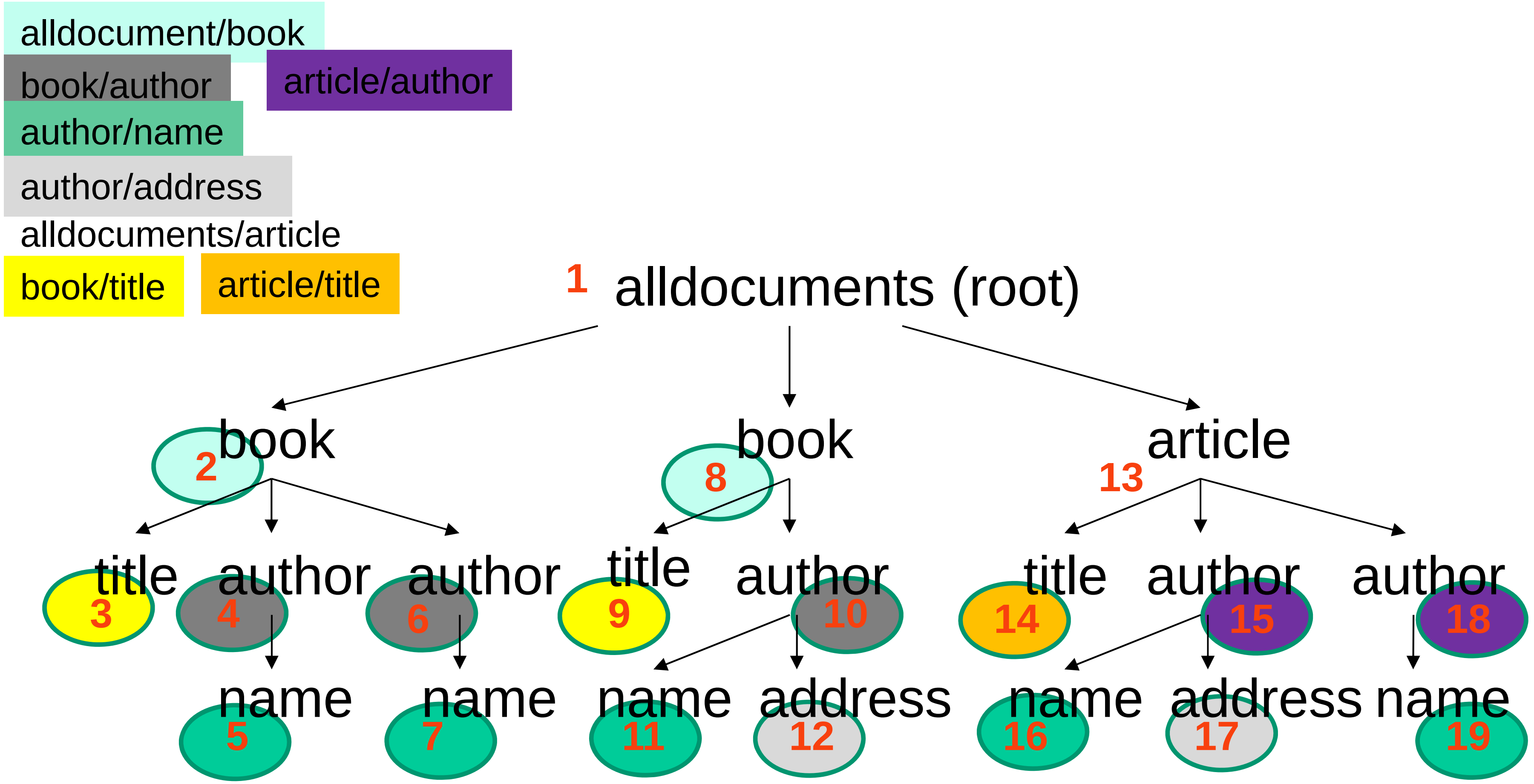
XML example: data graph + 0-bisimilar

book
author
name
address
article
title



Source: http://i.cs.hku.hk/~nikos/courses/CSIS7101/intro_slides.pdf

XML example: data graph + 1-bisimilar



alldocument/book

book/author

article/author

author/name

author/address

alldocuments/article

book/title

article/title

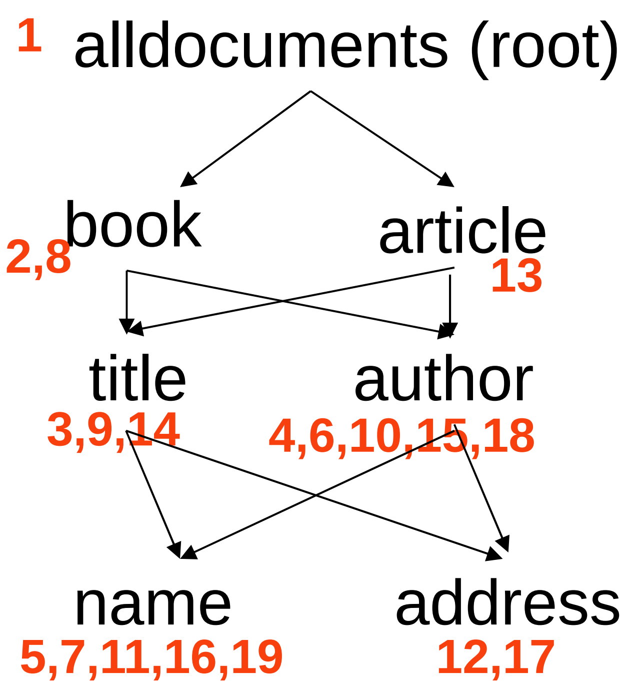
Source: http://i.cs.hku.hk/~nikos/courses/CSIS7101/intro_slides.pdf

A(k)-index examples

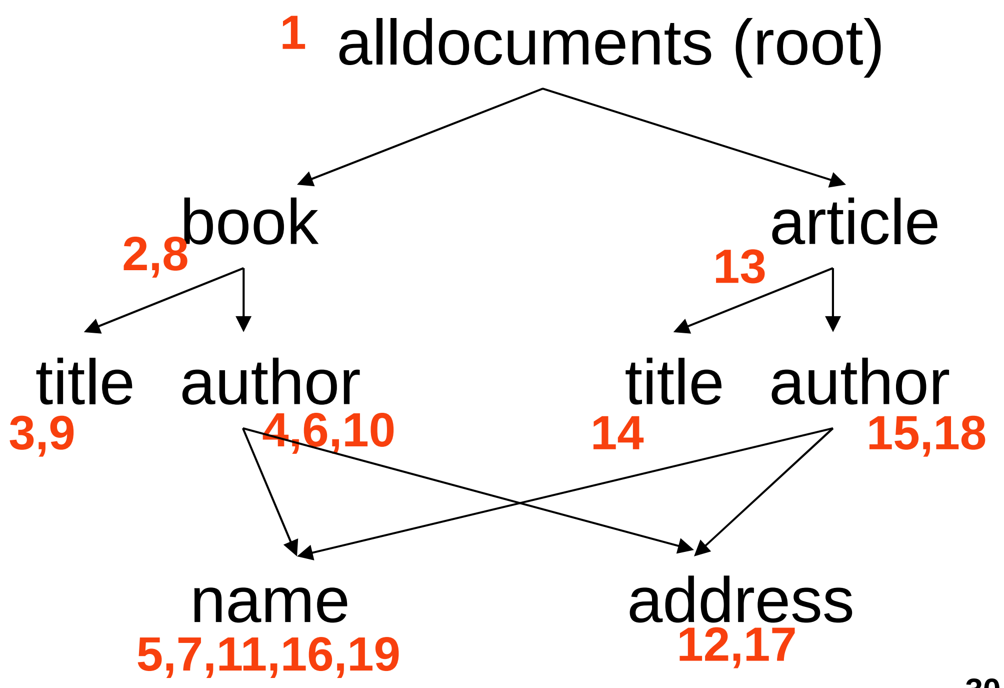
book
author
name
address
article
title

alldocument/book
book/author
article/author
author/name
author/address
alldocuments/article
book/title
article/title

A(0)-index


1 alldocuments (root)
2,8 book
article 13
title 3,9,14
author 4,6,10,15,18
name 5,7,11,16,19
address 12,17

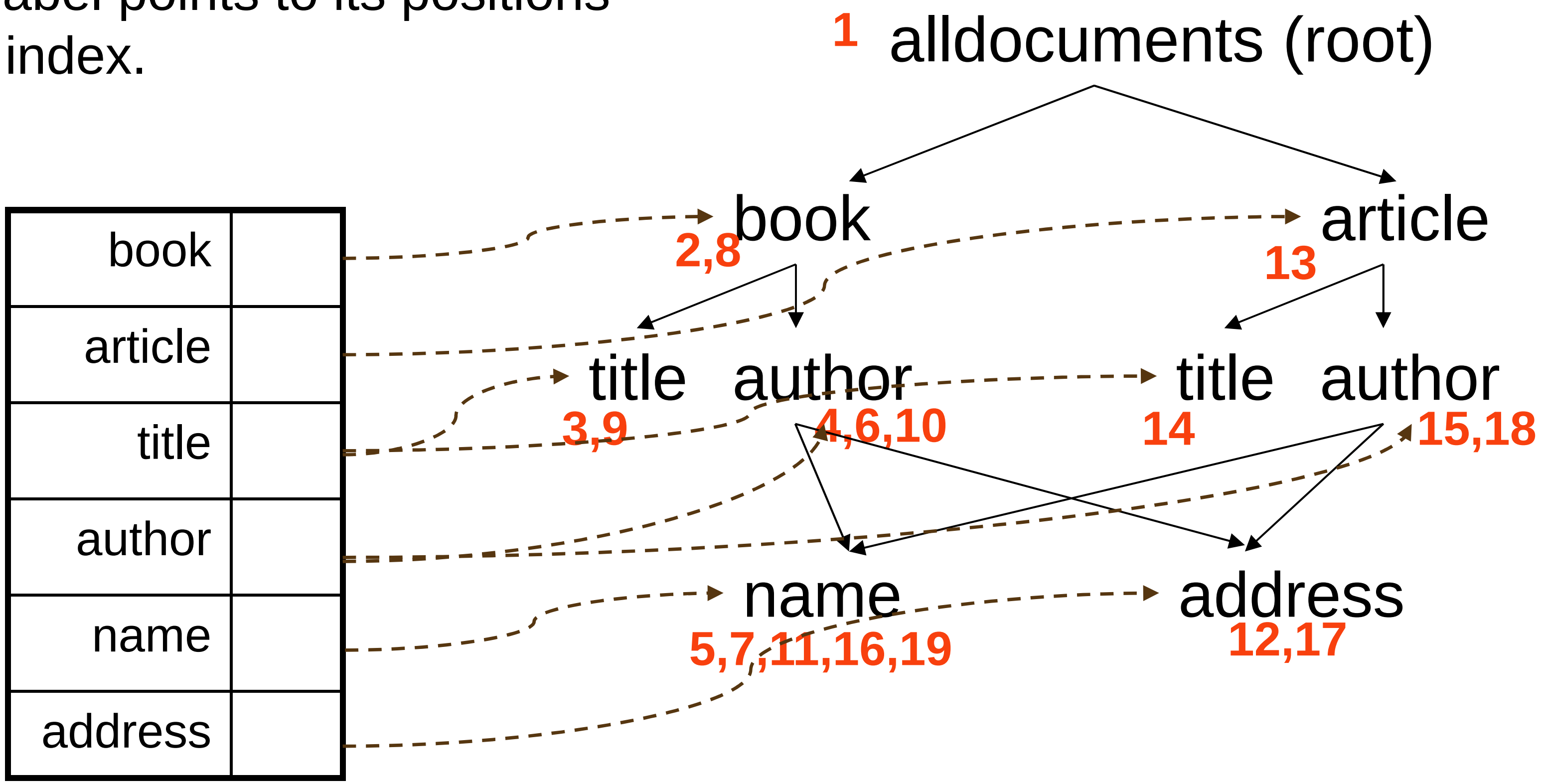
A(1)-index


1 alldocuments (root)
2,8 book
13 article
title 3,9
author 4,6,10
title 14
author 15,18
name 5,7,11,16,19
address 12,17

Using the A(k) index to search

A *Label Map* is constructed together with the index, where each label points to its positions in the index.

A(1)-index



1 alldocuments (root)

book 2,8

article 13

title 3,9

author 4,6,10

title 14

author 15,18

name 5,7,11,16,19

address 12,17

book

article

title

author

name

address

Source: http://i.cs.hku.hk/~nikos/courses/CSIS7101/intro_slides.pdf

Evaluation of path queries

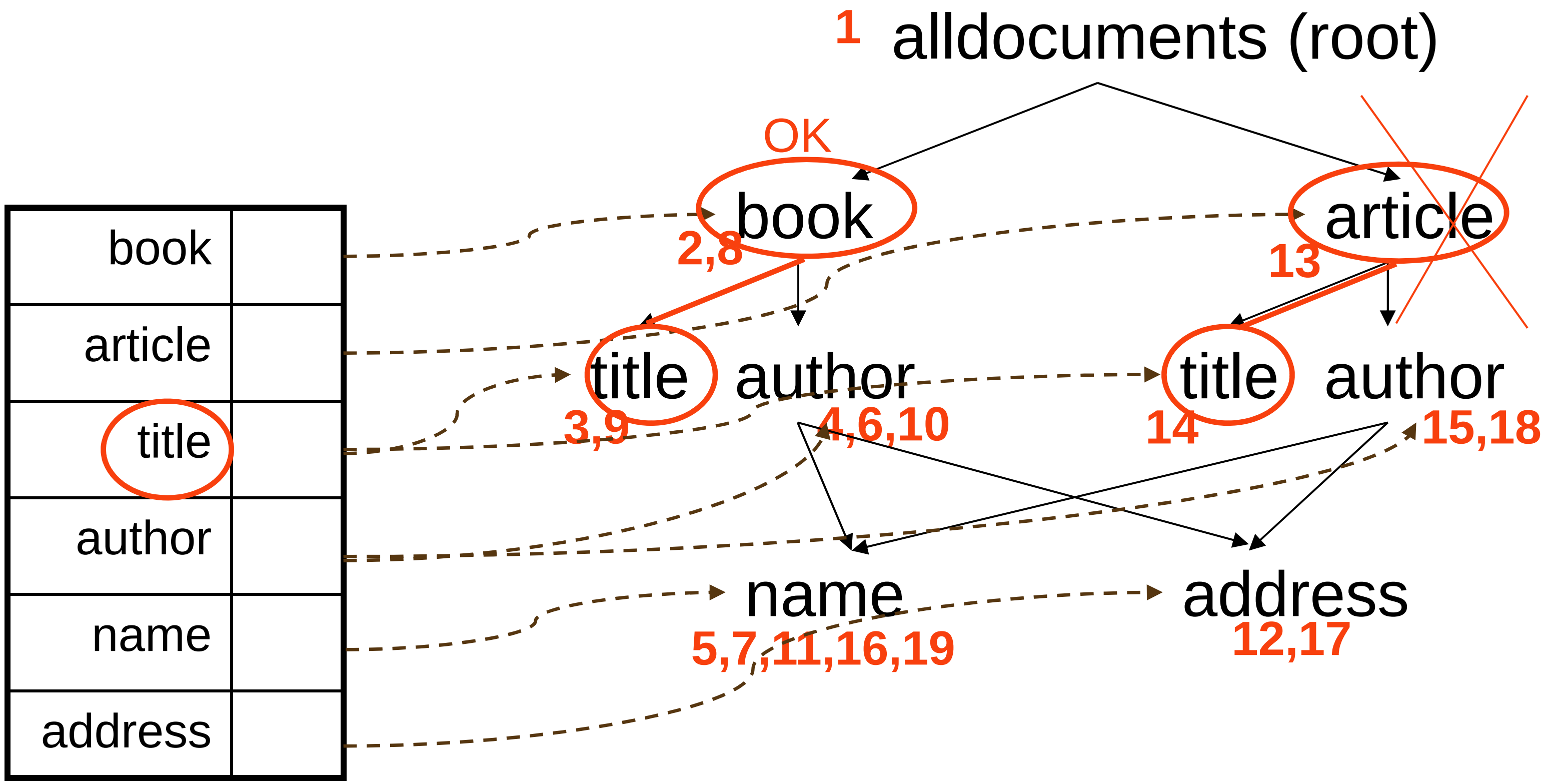
Assume that a path query *q* of length $\leq k$ is applied. The *A(k) index* can answer the query as follows:

1. First the last label of *q* is found and the label map is used to *find its positions in the index*.
2. Then the index *is traversed backwards* to complete the answer.

Evaluation of path queries (example)

- Query: book/title

A(1)-index



book	
article	
title	
author	
name	
address	

Problems of path indexes

- They are appropriate only for simple path queries up to a certain length.
- Therefore if a query has branches or regular path expressions the index cannot provide exact answers, but the actual data have to be accessed.
- Also these indexes have high storage and update cost.

Indexing characteristics

- Index Type:
  - Structure + implementation example
  - ▶ value,
  - full-text.
- Index Scope:
  - document, collection.
- Index Target:
  - document, node.
- Index Control:
  - automatic, voluntary, required.

Index type: value indexes

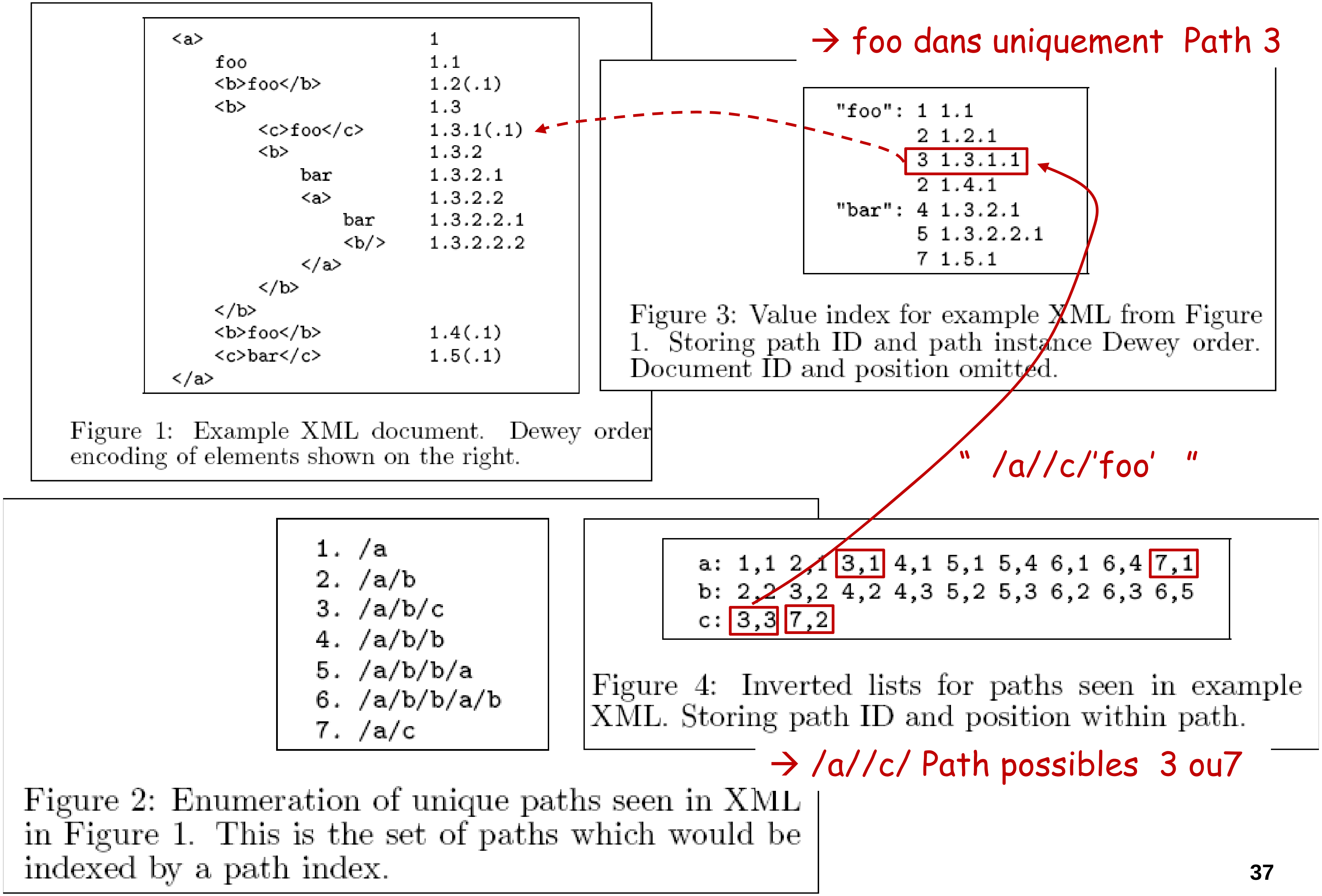
Value indexes: may be typed ⇒ Useful for comparisons (=, <, etc.).

Track all values for specific elements or attributes: (value, label)

- index atomic values; e.g., data(//emp/salary)
- use B+ trees

A value index on the element “color” would have an index entry for every separate instance of color and would be useful for a query such as: //color[.='green'].

http://www.idi.ntnu.no/~nilsgri/docs/Gri08PathIndex.pdf



```
<a>                         1
    foo                     1.1
    <b>foo</b>              1.2(.1)
    <b>                     1.3
        <c>foo</c>          1.3.1(.1)
        <b>                 1.3.2
            bar             1.3.2.1
            <a>             1.3.2.2
                bar         1.3.2.2.1
                <b/>        1.3.2.2.2
            </a>
        </b>
    </b>
    <b>foo</b>              1.4(.1)
    <c>bar</c>              1.5(.1)
</a>
```

Figure 1: Example XML document. Dewey order encoding of elements shown on the right.

→ foo dans uniquement Path 3

```
"foo": 1 1.1
       2 1.2.1
       3 1.3.1.1
       2 1.4.1
"bar": 4 1.3.2.1
       5 1.3.2.2.1
       7 1.5.1
```

Figure 3: Value index for example XML from Figure 1. Storing path ID and path instance Dewey order. Document ID and position omitted.

" /a//c/'foo' "

```
1. /a
2. /a/b
3. /a/b/c
4. /a/b/b
5. /a/b/b/a
6. /a/b/b/a/b
7. /a/c
```

Figure 2: Enumeration of unique paths seen in XML in Figure 1. This is the set of paths which would be indexed by a path index.

```
a: 1,1 2,1 3,1 4,1 5,1 5,4 6,1 6,4 7,1
b: 2,2 3,2 4,2 4,3 5,2 5,3 6,2 6,3 6,5
c: 3,3 7,2
```

Figure 4: Inverted lists for paths seen in example XML. Storing path ID and position within path.

→ /a//c/ Path possibles 3 ou7

Indexing characteristics

- Index Type:
  - Structure + implementation example
  - value,
  - ▶ full-text.
- Index Scope:
  - document, collection.
- Index Target:
  - document, node.
- Index Control:
  - automatic, voluntary, required.

Index type: Full-text indexing

⇒ Related to IR domain

Full Text indexes:

- Keyword search, inverted files
- IR world, text extenders

Some XML database products implement what they call full-text indexing, which minimally is a word index over a document.

Exemple

Doc 1:

```
<invoice>
  <buyer>
    <name>ABC Corp</name>
    <address>1 Industrial
   Way</address>
  </buyer>
  <seller>
    <name>Acme Inc</name>
    <address>2 Acme Rd.</address>
  </seller>
  <item count=3>saw</item>
  <item count=2>drill</item>
</invoice>
```

Doc 2:

```
<invoice>
  <buyer>
    <name>Oracle Inc</name>
    <phone>555-1212</phone>
  </buyer>
  <seller>
    <name>IBM Corp</name>
  </seller>
  <item>
    <count>4</count>
    <name>nail</name>
  </item>
</invoice>
```

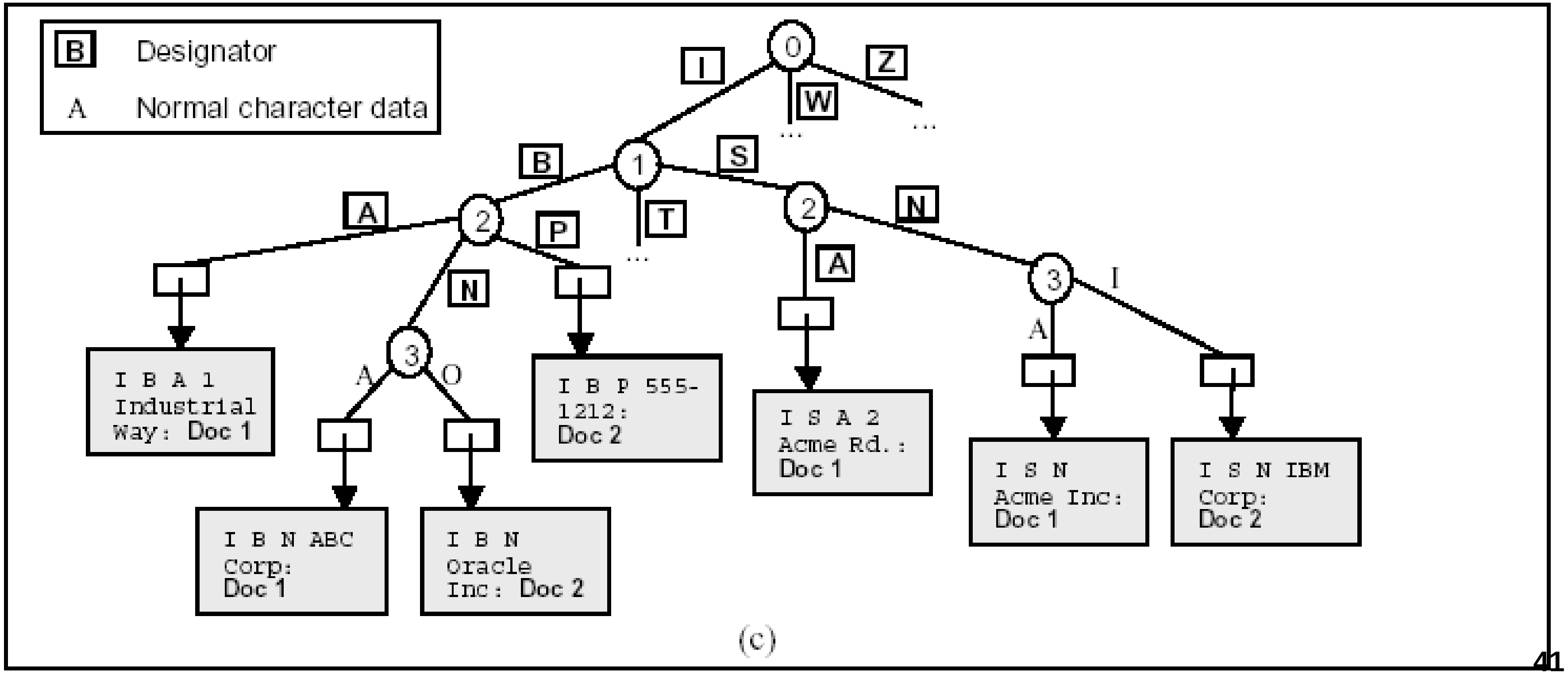
Patricia Trie

(a)

<invoice>	= I
<buyer>	= B
<name>	= N
<address>	= A
<seller>	= S
<item>	= T
<phone>	= P
<count>	= C
count (attribute)	= C'

(b)

Document 1	Document 2
I B N ABC Corp	I B N Oracle Inc
I B A 1 Industrial Way	I B P 555-1212
I S N Acme Inc	I S N IBM Corp
I S A 2 Acme Rd.	I T C 4
I T drill	I T N nail
I T C' 2	
I T saw	
I T C' 3	



(c)

Indexing characteristics

- Index Type:
  - structure, value, full-text.
- ► Index Scope:
  - document, collection.
- Index Target:
  - document, node.
- Index Control:
  - automatic, voluntary, required.

Index scope

Most native XML databases store documents in a collection. The scope of a given index could be:

- collection-wide
- or it could be restricted to a single document.

A native XML database system can choose the index scope it implements. Queries against a collection can return documents or sets of nodes within documents. In order to support efficient restriction of a query to a manageable set of documents, the system must support indexes at the collection scope.

Indexing characteristics

- Index Type:
  - structure, value, full-text.
- Index Scope:
  - document, collection.
- ► Index Target:
  - document, node.
- Index Control:
  - automatic, voluntary, required.

Index target

Related to scope is the target or the object referenced by an index entry. It can be:

- a document
- or an object within a document.

An index is capable of pointing down to the addressable unit in the system, but such granularity is not always necessary and can be expensive.

Although this is possible, it is the case that most database systems with fine-grained document storage reference directly to nodes in indexes rather than to the containing documents.

Indexing characteristics

- Index Type:
  - structure, value, full-text.
- Index Scope:
  - document, collection.
- Index Target:
  - document, node.
- ▶ Index Control:
  - automatic, voluntary, required.

Index control

- Voluntary indexes are specified explicitly by an interface to the system. These indexes allow for some experimentation to find the minimal useful set of indexes.

- Some systems have automatic indexes, where a well-defined set of indexes always is created, except for those that are disabled explicitly, by way of configuration or interface. The system also may have required indexes, which cannot be disabled because they are necessary for proper functioning of the system.

Plan

- Motivation – Needs – Means.
- NXD Concepts
- Indexing approaches
- ▶Conclusion

Conclusion

- Logical model: collection-documents
- Physical model: depends on the products
- Query/Update: Xquery (Xpath)
- Optimization: Indexes and execution plan of queries
- ACID Transactions: like for traditional DBMS (BaseX)
- Security: users accounts
- Recovery: like for traditional DBMS