XML Information Retrieval
Tutorial @ SIGIR 2003

Ricardo Baeza-Yates
Universidad de Chile
Chile
rbaeza@dcc.uchile.cl

Norbert Fuhr
University of Duisburg-Essen
Germany
fuhr@uni-duisburg.de

July 28, 2003

Tutorial Structure

1. XML standards
    - plain XML
    - XML namespaces
    - DTDs and XML schema

2. XML Query Languages
    - Requirements
    - Development
    - XPath and XQuery
    - XML databases

3. XML retrieval models and methods

   - Motivation
   - CO search
   - CAS search
   - Other tasks

4. XML retrieval algorithms and data structures

   - Structured text models
   - Example: Proximal Nodes
   - Comparison
   - Indexing and Searching
   - Examples: Toxin, IXPN
   - XML streams

Part I

XML Standards

Content

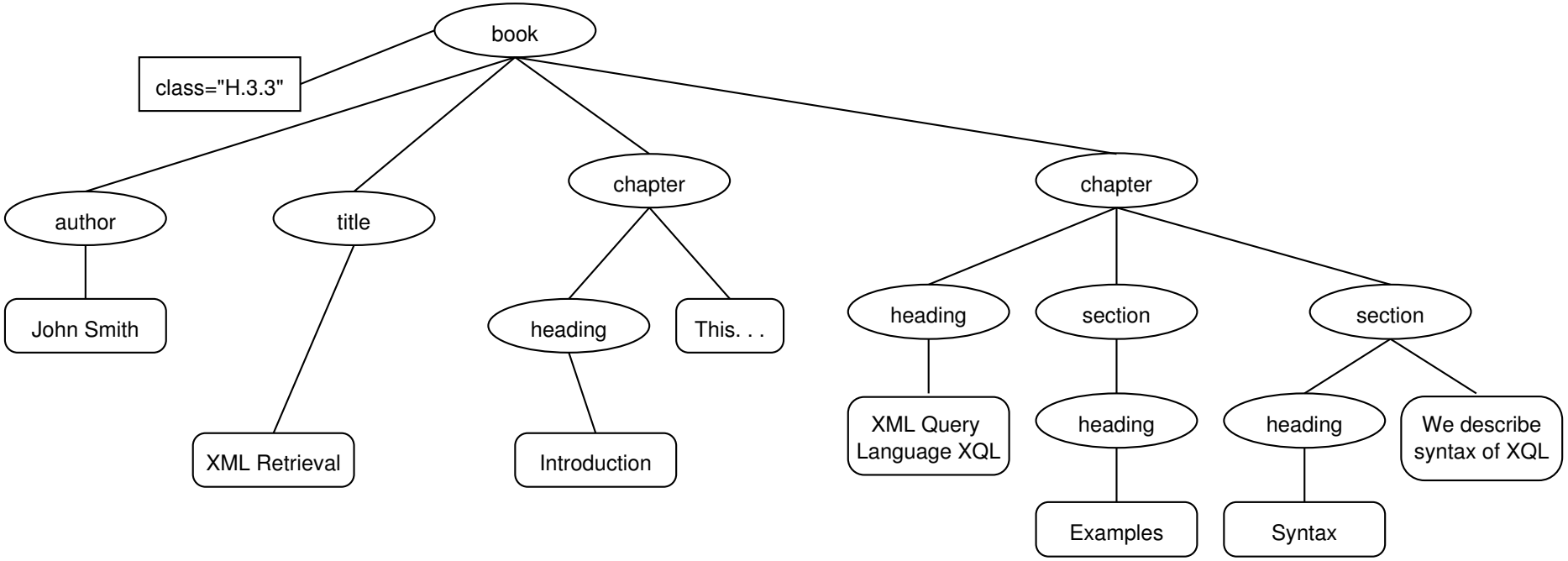
- Introduction
- XML namespaces
- Document Type Definitions (DTDs)
- XML Schema
- Other standards

(For details of the XML standards, see `http://www/w3c.org`)

Introduction: Example XML document

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE book SYSTEM "/services/dtds/book.dtd">
<book class="H.3.3">
  <author>John Smith</author>
  <title>XML Retrieval</title>
  <chapter>
    <heading>Introduction</heading>
    This text explains all about XML and IR.
  </chapter>
  <chapter>
    <heading> XML Query Language XQL </heading>
    <section>
        <heading>Examples</heading>
    </section>
    <section>
        <heading>Syntax</heading>
        Now we describe the XQL syntax.
    </section>
  </chapter>
</book>
```

Tree structure of XML documents



XML properties

- hierarchical structure: nesting of elements
- element: start-tag – content – end tag
  `<tag-name> content </tag-name>`
- tag-name: logical name of element
- content: data or other elements
  (nesting of elements)
  `<author><first>John</first>`
  `<last>Smith</last></author>`
- attributes: assigned to elements
  (specified in start tag)
  pair of (*attribute name*, *attribute value*),
  e.g. `<date format="ISO">2000-05-01</date>`

XML: Basic ideas

- markup of logical structure of documents
  $\rightsquigarrow$ explicit logical structure, can be exploited by appropriate IR methods

- separation of logical structure and layout
  $\rightsquigarrow$ different presentations of one document, depending on output media, user group (language,...)

- support interoperability of Web services and XML-based applications
  $\rightsquigarrow$ standard document format for IR systems

Basic XML standard does not deal with ...

- standardization of element names
  → XML namespaces

- structure of element content
  → XML DTDs

- data types of element content
  → XML schema

I.1 XML namespaces

allow for combination of element names defined independently (in different resources)

```
<?xml version="1.0"?>
 <bk:book xmlns:bk='urn:loc.gov:books'
          xmlns:isbn='urn:ISBN:0-395-36341-6'>
     <bk:title>Cheaper by the Dozen</bk:title>
     <isbn:number>1568491379</isbn:number>
 </bk:book>
```

Example: Dublin Core namespace

```
<oai_dc:dc  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:title>Generic Algebras with Involution of Degree 8m</dc:title>
  <dc:subject>Orthogonal group, Symplectic group,
              invariant field, rational
  </dc:subject>
  <dc:date>2001-02-27</dc:date>
  <dc:format>application/postscript</dc:format>
  <dc:identifier>ftp://ftp.esi.ac.at/pub/esi1001.ps</dc:identifier>
  <dc:source>ESI preprints</dc:source>
  <dc:language>en</dc:language>
</oai_dc:dc>
```

I.2 Document Type Definitions

- well-formed XML: proper nesting of elements
  (e.g. <a> is forbidden

- valid XML: document is well-formed and conforms to *document type definition*

Declaration of DTD

in the document header:

```
<!DOCTYPE name  PUBLIC publicid systemid>

<!DOCTYPE name SYSTEM filename>
```

Example HTML document with public DTD

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>My Home Page</title>
</head>
<body>
<p>Hello! This is my home page.</p>
</body>
</html>
```

Example XML document with system DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE book SYSTEM  "/services/dtds/book.dtd">
<book class="H.3.3">
  <author>John Smith</author>
  <title>XML Retrieval</title>
  <chapter>
    <heading>Introduction</heading>
    This text explains all about XML and IR.
  </chapter>
  <chapter>
    <heading>
      XML Query Language XQL
    </heading>
    <section>
        <heading>Examples</heading>
    </section>
    <section>
        <heading>Syntax</heading>
        Now we describe the XQL syntax.
    </section>
  </chapter>
</book>
```

DTD for example document

```
<!ELEMENT book (author, title, chapter+)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT chapter (heading,#PCDATA?,section*)>
<!ELEMENT section (heading,#PCDATA?)>
<!ELEMENT heading (#PCDATA)>
<!ATTLIST book
    class  CDATA #REQUIRED
    crdate CDATA #IMPLIED
    type   (monograph|collection|proceedings) "monograph">
```

DTD Specification

- element definitions
- definition of element attributes
- entity definitions (macros)

DTDs from an IR point of view

- restrict logical structure of documents
  $\rightsquigarrow$ IR methods can be tailored for document type

- element types have a well-defined meaning
  $\rightsquigarrow$ specialized search methods for content of specific elements possible
  e.g. person name, date, classification code

I.3 XML Schema

types of XML applications:

1. structured documents
   text documents with markup of logical structure
   $\rightsquigarrow$ document-centric

2. formatted data
   (e.g. spreadsheets, business forms, databases, . . .)
   XML as exchange format
   $\rightsquigarrow$ data-centric

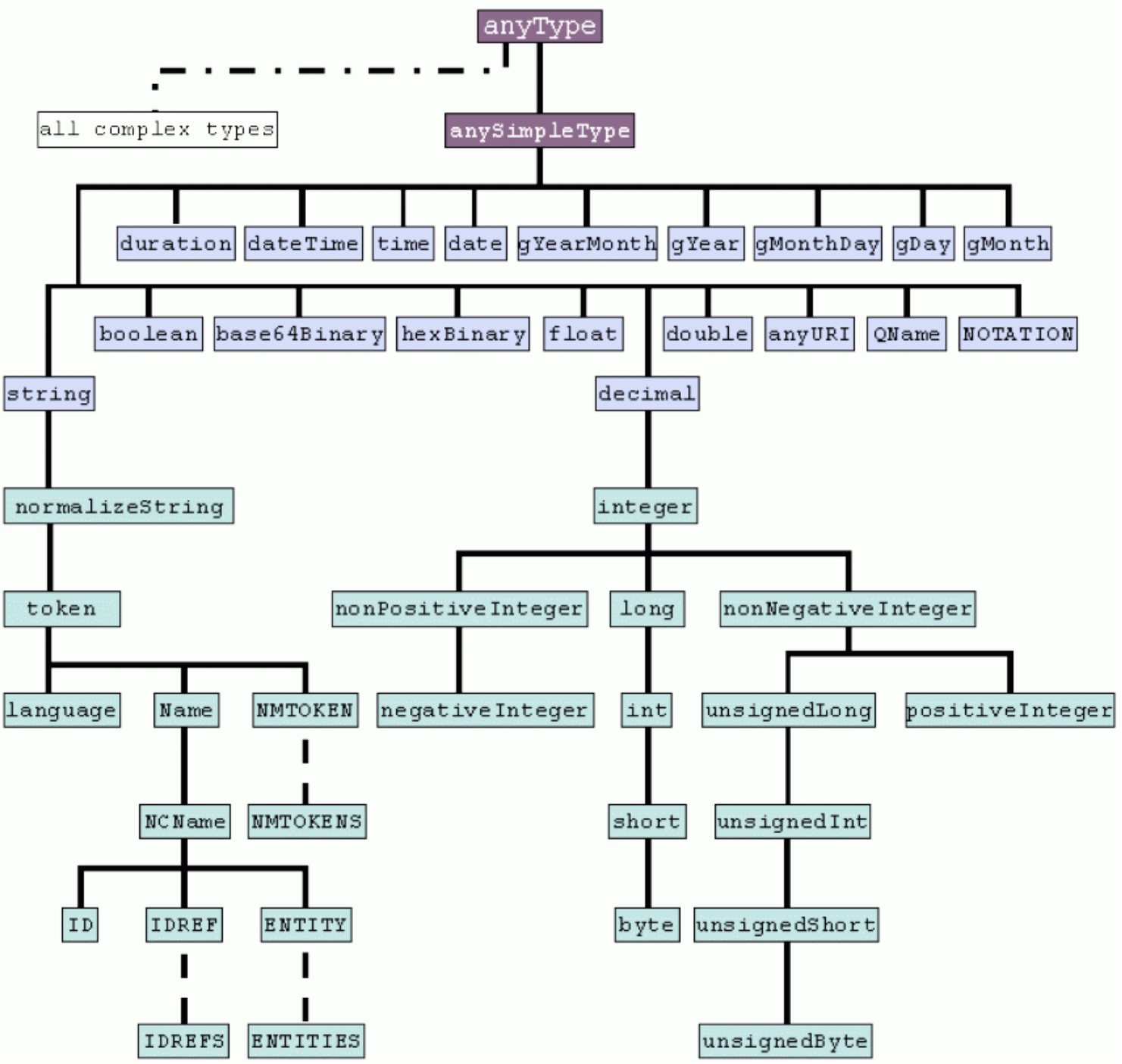
XML Schema

resolves weaknesses of DTDs wrt. formatted data:

- support for data types
  (DTDs: only child elements, PCDATA, mixed content and EMPTY)
- specification of structured data
  (e.g. arrays with lower/upper bounds)
- reuse of data-types
  (explicit support for elements only, but not for attributes)
- extensible type system
  (user-defined types / subtypes)
- support for namespaces
- support for reference mechanism
  (ID, IDREF(S)) supports local references only)
- DTD syntax in XML

Classification of XSD types

- atomic vs. aggregated:
  - atomic: atomic values, content not further interpreted
  - aggregated: lists, union (of different types)
- primitive vs. derived:
  - primitive: independent of other types
  - derived: subtype of other type
- predefined vs. user-defined:
  - XML Schema Part 2 defines 44 types
  - users may define additional types by aggregation or as subtypes of existing types


anyType
all complex types
anySimpleType
duration
dateTime
time
date
gYearMonth
gYear
gMonthDay
gDay
gMonth
boolean
base64Binary
hexBinary
float
double
anyURI
QName
NOTATION
string
decimal
normalizeString
integer
token
nonPositiveInteger
long
nonNegativeInteger
language
Name
NMTOKEN
negativeInteger
int
unsignedLong
positiveInteger
NCName
NMTOKENS
short
unsignedInt
ID
IDREF
ENTITY
byte
unsignedShort
IDREFS
ENTITIES
unsignedByte

Subtyping

- restriction
  - value restriction
    (string length, range, string pattern)
  - cardinality restriction
    (min,max bounds) of arrays

- extension
  (adding elements to a type, like type attributes in object-oriented programming)

- redefinition
  (redefine types of a given schema definition)

XML Schema from an IR point of view

Data types for IR applications:

- language
- thesaurus term / classification code
- geographic location
- . . .

→ most IR data types can not be defined at the syntactic level

→ XML schema can be used for IR data type definition, but type checking is not possible

Other XML Standards

- Style: XSL
- Transformations: XSLT (can be seen as a query language)
- Linking: XLink and XPointer
- Documents: DOM (Document Object Model), MathML, SVG (Scalable Vector Graphics), etc.

Part II

XML Query Languages

Content

The meeting (or clashing) place of databases and IR:

- Requirements
- Query languages history
- XPath locator language
- XQuery query language
- XML Databases

II.1 Requirements (1)

- From Semistructured Data
  - Selection: pattern + filter + constructor
  - Filtering
  - Reduction: pruned elements
  - Restructuring: grouping, sorting, etc.
  - Combine data: joins and semi-joins
  - Vague queries
  - Navigation
  - Aggregation
  - Existential and universal quantifiers
  - Data types
  - Insert, delete, and update operations

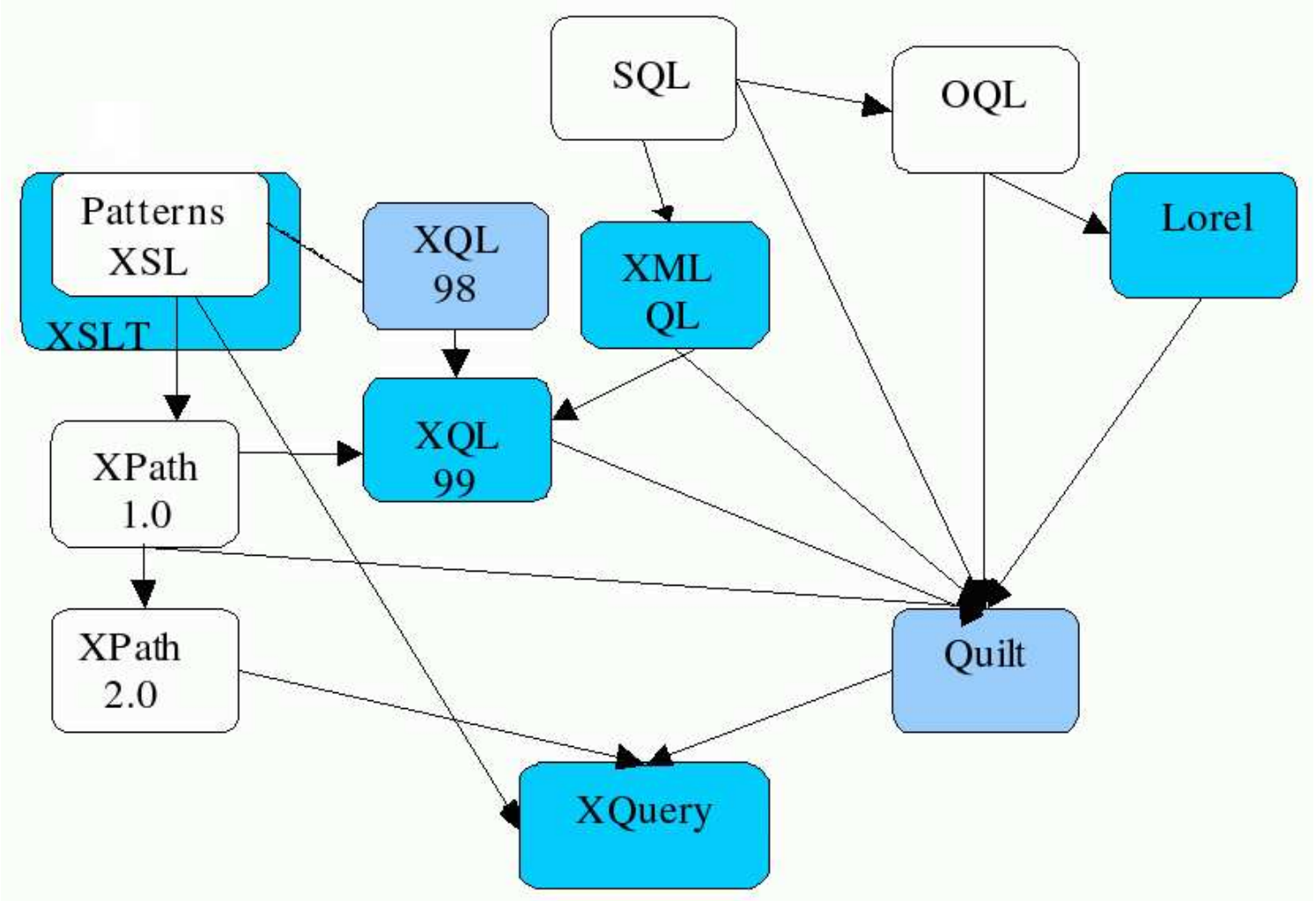
Requirements (2)

- From Information Retrieval
  - Keyword queries: Boolean, context, similarity, etc.
  - Pattern matching
  - Structural queries: inclusion, distance relations, etc.
  - Weighting query terms
  - Ranking

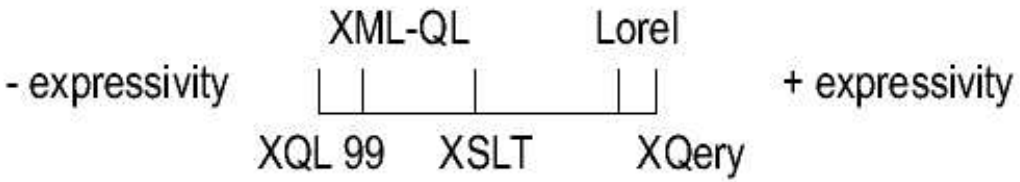
- Others
  - Use of metadata
  - DTD or Xschema awareness
  - Support for XLink and XPointer
  - Set operations on results

II.2 Query Languages History

From [Delgado and Baeza-Yates(2002)]:



Comparison: Basic Model (1)



	Lorel	XSLT	XML-QL	XQL 99	XQuery
Main functions	Queries of semi-structured data	Transformation of documents	Data queries, transformations, integration of XML data from different sources	Queries within a document and queries on collections of documents	Queries on heterogeneous data sources
Data model	Graph / Tree	Tree (such as XPath 1.0)	Graph	Tree (DOM of XML)	Ordered sequence of nodes (such as XPath 2.0)
Input source & format	XML Documents	XML Document/s + StyleSheet	XML Documents from different sources	XML Document/s	XML Document, XML Fragments, Collections of XML documents
Output information	XML Document (Ordered list of identifiers of the resulting elements)	XML Document (Transformed XML tree), Collections of XML documents *(xsl:document)*	XML Document (XML Fragments)	XML Document (XML Fragments, List of resulting elements)	XML Document, XML Fragment, Collections of XML documents

Comparison: Semistructured Data (2)

		Lorel	XSLT	XML-QL	XQL 99	XQuery
Selection Operation	**Pattern/ Filter/ Constructor**	**select** *constructor* **from** *pattern* **where** *filter*	**<xsl:for-each select=** pattern **>** **<xsl:if match=**filter**>** **<copy-of />** **</xsl:if>** **</xsl:for-each>**	**WHERE** *pattern* **IN** *source*, *filter* **CONSTRUCT** *constructor*	pattern [filter]	**FOR** *patterns* **LET** *bindings* **WHERE** *filter* **RETURN** *constructor*
	Relational Operators	>, >=, <, <=, =, <>, ==	>, >=, <, <=, =, !=	>, >=, <, <=, =, !=	>, >=, <, <=, =, !=	>, >=, <, <=, =, != For nodes: ==, !==
	Boolean Operators	*and, or, not*	*and, or*	No	*and, or*	*AND, OR*
	Nesting queries	Yes	Yes	Yes	Yes	Yes
	Creation of new elements	Yes	Yes	Yes	No	Yes
Filtering of elements preserving hierarchy		No	Yes (using templates)	No	Yes	Yes *(filter)*
Reduction		No	Yes	No	Yes	No
Restructuring operations	**Grouping of results**	Yes *(group by)*	No	No	Only by structure, not by value	Yes
	Skolem Functions	Yes	No	Yes	No	Yes
	Sorting of results	Yes *(order by)*	Partial (xsl:sort[a])	Yes *(ORDER-BY)*	No	Yes *(SORTBY)*
Inter-document links (join), Intra-documents links (semi-join)		Join,Semi-join	Semi-join	Join, semi-join	Semi-join, join	Join, semi-join

Comparison: Semistructured Data (3)

		Lorel	XSLT	XML-QL	XQL 99	XQuery
Use of tag variables		Yes	Yes	Yes	No	Yes
Path expressions		Regular expression operators: *, |, +, ? Qualifiers: >, @	XPath Expressions	Regular expression operators *, |, +, .	Wild card: * Path Operators: /, //	XPath Expressions
Dereferencing of IDREF(S) attributes		Yes (As a subelement using the point notation)	Yes *(id())*	Yes (By means of a join)	Yes *(id())*	Yes (Dereference Operator =>)
Set Functions		*min, max, count, sum, avg*	*sum, count*	*min, max, count, sum, avg*	*sum, count*	*min, max, count, sum, avg*
Quantifiers	**Existential**	Yes *(exists)*	Yes (implicit)	Yes (implicit)	Yes (implicit)	Yes *(SOME)*
	Universal	Yes *(for all)*	No	No	Yes *(all)*	Yes *(EVERY)*
Handling of datatypes (XML Schema)		Partial	No (under study)	No	No	Yes
Insertion, delete and update		Yes	Yes	No	No	No

Comparison: IR & others (4)

		Lorel	XSLT	XML-QL	XQL 99	XQuery
Keywords	**A word inside free text**	By means of path expressions	By means of path expressions	By means of path expressions	By means of path expressions	By means of path expressions
	Similarity	No	No	No	No	No
	Context	No	No	No	No	No
	Boolean Operators	Yes	Yes	No	Yes	Yes
Pattern matching		operators: *like, grep, soundex*	String operators and functions	*Like* operator	String operators and functions	String operators and functions
Structural Queries	**Structural Inclusion**	By means of path expressions	By means of path expressions	By means of path expressions	By means of path expressions	By means of path expressions
	Positional Inclusion	Yes	Yes	Yes	Yes	Yes
	Structural proximity	No	No	No	Yes (immediately precedes ";")	Context node
	Structural Order	By means of comparison of positional indexes	Yes (*preceding, preceding-siblings, following, following-siblings*)	By means of comparison of positional indexes	Yes (*before, after*)	Yes (*BEFORE, AFTER*)
Assignation of weighting to the terms of the query		No	No	No	No	No
RDF support		No	No	No	No	No
XLink and Xpointer support		No	No	No	Partial	No (In study)
Operations over sets		Intersection, union, difference	Union, difference	Intersection, union	Intersection, union	Intersection, union, difference

II.3 XPath locator language

restricted XML query language
retrieves complete elements (subtrees) of XML documents

used in

XSLT (Extensible Style Sheet Language Transformations)
for specifying argument of a transformation

XPointer (XML Pointer)
for defining sources / targets of links

XQuery (XML Query Language)
for selecting elements that are arguments of further operations
(value joins, restructuring, aggregation)

Path Expressions

- search for single elements:
  `heading`
- parent-child:
  `chapter/heading`
- ancestor-descendant:
  `chapter//heading`
- document root:
  `/book/*`
- filter wrt. structure:
  `//chapter[heading]`
- filter wrt content:
  `/document[@class="H.3.3" and author="John Smith"]`

Axes

generalization of locator operators

child:: children of the context node

descendant:: descendants of the context node

parent:: parent of the context node

ancestor:: ancestors of the context node

following-sibling:: all the following siblings of the context node

preceding-sibling:: all the preceding siblings of the context node

following:: all nodes in the same document as the context node that are after the context node in document order

preceding:: all nodes in the same document as the context node that are before the context node in document order,

attribute:: attributes of the context node

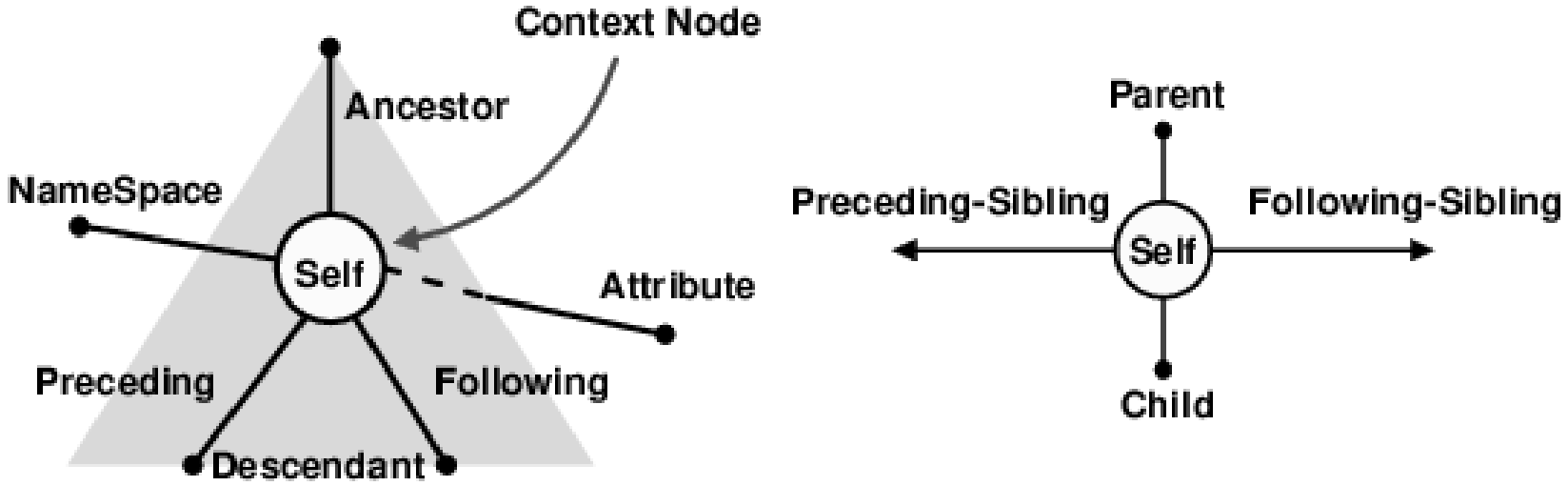
namespace:: namespace nodes of the context node

self:: just the context node itself

descendant-or-self:: context node and the descendants of the context node

ancestor-or-self:: context node and the ancestors of the context node

Axes



Model: Ordered set of nodes with attributes

XPath axes examples

- `child::para` *para element children of the context node*
- `child::*` *element children of the context node*
- `child::text()` *text node children of the context node*
- `child::node()` *children of the context node, whatever their node type*
- `attribute::name` *name attribute of the context node*
- `attribute::*` *the attributes of the context node*

- `descendant::para` *para element descendants of the context node*
- `ancestor::div` *div ancestors of the context node*
- `ancestor-or-self::div` *div ancestors of the context node and, if the context node is a div element, the context node as well*
- `descendant-or-self::para` *para element descendants of the context node and, if the context node is a para element, the context node as well*
- `self::para` *context node if it is a para element, and otherwise selects nothing*
- `child::chapter/descendant::para` *para element descendants of the chapter element children of the context node*

- `child::*/child::para` *para grandchildren of the context node*

- */ document root (which is always the parent of the document element)*

- `/descendant::para` *para elements in the same document as the context node*

- `/descendant::olist/child::item` *item elements that have an olist parent and that are in the same document as the context node*

II.4 XQuery

Last draft: syntax (May 2003), requirements (June 2003)

Weak data model:

- Ordered, labelled forest, with node identity and data types
- Static semantics: type inference, structural hierarchy
- Dynamic semantics: value inference
- Same data model as XPath 2.0
- Pure functional language with impure syntax
  - A query is an expression
  - Expressions can be nested
  - Basic structure:

```
FOR PathExpression
WHERE AdditionalSelectionCriteria
RETURN ResultConstruction
```

Advantages

- Expressive power
- Easy to learn
- Easy to implement (?)
- Optimizable in many environments
- Related to concepts people already know
- Several current implementations
- The accepted W3C XML Query Language

Expressions

- Element constructors
- Path expressions
- Restructuring
  - FLWOR expressions
  - Conditional expressions
  - Quantified expressions
- Operators and functions
- List constructors
- Expressions that test or modify data-types

Element Constructors

Element constructors look like the XML they construct

```
<book year="1994">
    <title>TCP/IP Illustrated</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <publisher>Addison-Wesley</publisher>
    <price> 65.95</price>
 </book>
```

Element Constructors: Examples

Generate an <emp> element that has an "empid" attribute and nested <name> and <job> elements.

```
<emp empid = "12345">
   <name>John Smith</name>
   <job>Anthropologist</job>
</emp>
```

Generate an <emp> element that has an "empid" attribute. The value of the attribute and the content of the element are specified by variables that are bound in other parts of the query.

```
<emp empid = {$id}>
   {$name}
   {$job}
</emp>
```

Path Expressions

```
<-- XQuery uses the abbreviated syntax
     of XPath for path expressions      -->

document("bib.xml")

/bib/book/author

/bib/book//*

//author[last="Stevens" and first="W."]

document("bib.xml")//author
```

Path Expressions: Extensions

```
<-- precedes, follows -->
//book[ author[last="Stevens"] precedes author[last="Abiteboul"] ]

<--  Namespaces -->
 namespace rev = "www.reviews.com"
 //rev:rating

<-- Dereference -->
//publisher/web/@href->html
```

Path Expressions: Examples

In the second chapter of the document named "zoo.xml", find the figure(s) with caption "Tree Frogs".

```
document("zoo.xml")//chapter[2]//figure[caption = "Tree Frogs"]
```

Find all the figures in chapters 2 through 5 of the document named "zoo.xml."

```
document("zoo.xml")//chapter[2 TO 5]//figure
```

Find captions of figures that are referenced by `<figref>` elements in the chapter of "zoo.xml" with title "Frogs".

```
document("zoo.xml")//chapter[title = "Frogs"]
//figref/@refid=>fig/caption
```

XQuery Examples (1)

List the names of the second-level managers of all employees whose rating is "Poor".

```
//emp[rating = "Poor"]/@mgr=>emp/@mgr=>emp/name
```

Find all captions of figures and tables in the chapter of "zoo.xml" with title "Monkeys".

```
document("zoo.xml")//chapter[title = "Monkeys"]
//(figure | table)/caption
```

From a document that contains employees and their monthly salaries, extract the annual salary of the employee named "Fred".

```
//emp[name="Fred"]/salary * 12
```

FLWOR Expressions

FOR - LET - WHERE - ORDER BY - RETURN

Similar to SQL's SELECT - FROM - WHERE

```
for $book in document("bib.xml")//book
where $book/publisher = "Addison-Wesley"
return
  <book>
        {
      $book/title,
          $book/author
        }
  </book>
```

FOR vs. LET

FOR iterates on a sequence, binds a variable to each node

LET binds a variable to a sequence as a whole

```
for $book in document("bib.xml")//book
let  $a := $book/author
where contains($book/publisher, "Addison-Wesley")
return
  <book>
        {
      $book/title,
          <count> Number of authors: { count($a) } </count>
        }
  </book>
```

Conditional Expressions

```
IF expr THEN expr ELSE expr
FOR $h IN //holding
RETURN
   <holding>
           { $h/title,
            IF ($h/@type = "Journal")
               THEN $h/editor
               ELSE $h/author
           }
   </holding>
```

Sorted Expressions:

```
expr SORTBY (expr ASCENDING , ... )

FOR $b IN //book
RETURN
    $b SORTBY(title, author[1]/name)
```

Inner and Outer Joins

```
FOR $book IN document("www.bib.com/bib.xml")//book,
       $quote IN document("www.bookstore.com/quotes.xml")//listing
WHERE $book/isbn = $quote/isbn
RETURN
        <book>
                { $book/title }
                { $quote/price }
        </book>
SORTBY (title)

FOR $book IN document("bib.xml")//book
RETURN
        <book>
          { $book/title }
          {
             FOR $review IN document("reviews.xml")//review
             WHERE $book/isbn = $review/isbn
             RETURN $review/rating
           }
        </book>
SORTBY (title)
```

Quantifiers

EVERY var IN expr SATISFIES expr

SOME var IN expr SATISFIES expr

```
FOR $b IN //book
WHERE SOME $p IN $b//para SATISFIES
     contains($p, "sailing")
     AND contains($p, "windsurfing")
RETURN $b/title
```

```
FOR $b IN //book
WHERE EVERY $p IN $b//para SATISFIES
     contains($p, "sailing")
     AND contains($p, "windsurfing")
RETURN $b/title
```

FLWOR: Data for Examples

```
<book>
 <title> XML: An Introduction</title>
 <author>Smith </author> <author>Miller</author>
 <publisher>Morgan Kaufmann</publisher>
 <year>1998</year>
 <price>50</price>
</book>
<book>
 <title>XSLT Course</title>
 <author> Jones</author>
 <publisher>Addison Wesley</publisher>
 <year>2000</year>
 <price>40</price>
</book>
```

XQuery Examples (2)

List the titles of books published by Morgan Kaufmann in 1998.

```
FOR $b IN document("bib.xml")//book
WHERE $b/publisher = "Morgan Kaufmann"
AND $b/year = "1998"
RETURN $b/title
```

List each publisher and the average price of its books.

```
FOR $p IN distinct(document("bib.xml")//publisher)
LET $a := avg(document("bib.xml")//book[publisher = $p]/price)
RETURN
   <publisher>
      <name> {$p/text()} </name>
      <avgprice> {$a} </avgprice>
   </publisher>
```

XQuery Examples (3)

List the publishers who have published more than 100 books.

```
<big_publishers>
   {
    FOR $p IN distinct(document("bib.xml")//publisher)
    LET $b := document("bib.xml")//book[publisher = $p]
    WHERE count($b) > 100
    RETURN $p
   }
</big_publishers>
```

XQuery Examples (4)

Invert the structure of the input document so that, instead of each book element containing a sequence of authors, each distinct author element contains a sequence of book-titles.

```
<author_list>
   {
    FOR $a IN distinct(document("bib.xml")//author)
    RETURN
      <author>
         <name> {$a/text()} </name>
         {
          FOR $b IN document("bib.xml")//book[author = $a]
          RETURN $b/title
         }
      </author>
   }
</author_list>
```

XQuery Examples (5)

For each book whose price is greater than the average price, return the title of the book and the amount by which the book's price exceeds the average price.

```
<result>
   {
    LET $a := avg(document("bib.xml")//book/price)
    FOR $b IN document("bib.xml")//book
    WHERE $b/price > $a
    RETURN
       <expensive_book>
          {$b/title}
          <price_difference>
             {$b/price - $a}
          </price_difference>
       </expensive_book>
   }
</result>
```

XQuery Examples (6)

Construct a new element having the same name as the element bound to $e. Transform all the attributes of $e into subelements, and all the subelements of $e into attributes.

```
<{name($e)}>
   {
    FOR $c IN $e/*
    RETURN attribute(name($c), string($c))
   }
   {
    FOR $a IN $e/@*
    RETURN
       <{name($a)}>
          {string($a)}
       </>
   }
</>
```

Conditions on Text

```
Equality:
       //section[title="Procedure"]
Full-text:
      //section[contains(title, "Procedure")]
```

More Full-text support in the future

Last published requirements: June 2003

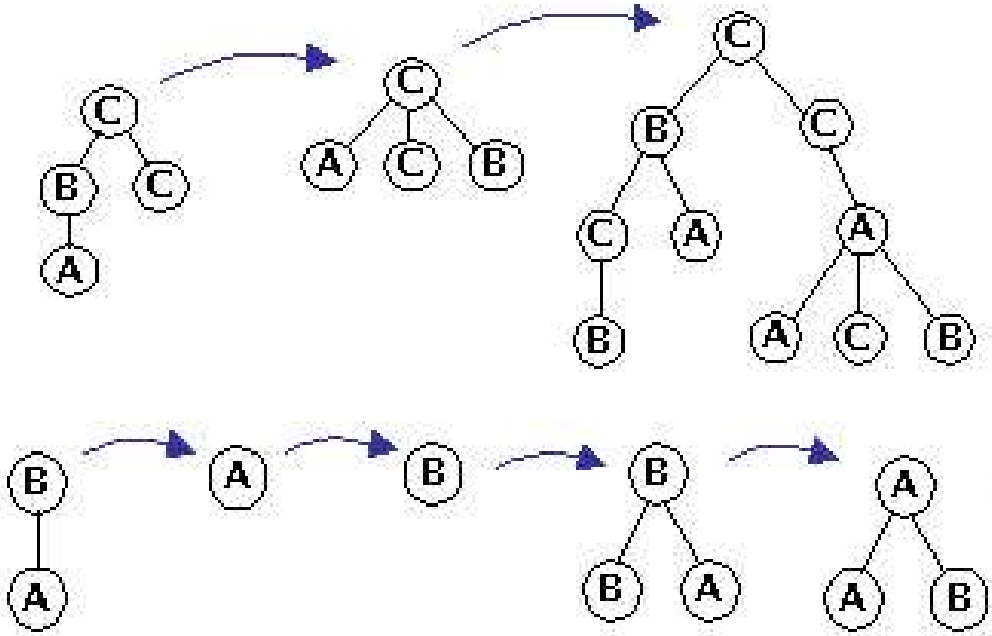
Full-text Requirements

- Full-Text predicates and SCORE functions independently
- Full-Text predicates use a language subset of SCORE functions
- Allow the user to return and sort-by SCORE (float in 0-1)
- SCORE must not require explicit global corpus statistics
- SCORE algorithm should be provided and can be disabled
- Minimal operations:
  - single-word and phrase search with stopwords
  - suffixes, prefix, infix
  - proximity searching (with order)
  - Boolean operations
  - Word normalization, diacritics
  - Ranking, relevance
- Search over everything, including attributes
- Proximity across markup elements
- Extensible

Filters

Filter(expression)

Result is an "ordered forest" that preserves sequence and hierarchy



Functions

- Built-in functions: max(), min(), sum(), count(), avg() distinct(), empty(), contains()
- User-defined functions
- Defined in XQuery syntax
- May be recursive
- May be typed
- Extensibility mechanisms planned

Example:

```
define function depth(element $e) returns integer
{
   <-- An empty element has depth 1
        Otherwise, add 1 to max depth of children -->
   if (empty($e/*))
       then 1
       else max(depth($e/*)) + 1
}

depth(document("partlist.xml"))
```

Possible Future Extensions

Insert-replace-delete:

```
INSERT
UPDATE
FOR $e IN /emp
INSERT <n_skills>{ count($e/skill) }</n_skills>
BEFORE $e/skill[1]

REPLACE
UPDATE
FOR $e IN /emp
WHERE $e/empno = "12345"
REPLACE $e/job
WITH <job> Broom Tester </job>

UPDATE
FOR $e IN /emp/[job = "Programmer"],
       $s IN $e/skill
WHERE $s/rating < 4
       OR $s/cert_date < date("1995-01-01")
DELETE $s
```

Further support for full-text retrieval

XQuery Implementations

- Software AG's QuiP
- Microsoft demo
- Lucent Galax
- GMD-IPSI
- X-Hive
- XML Global
- SourceForge XQuench
- Fatdog
- Qexo (GNU Kawa) - compiles to Java byte code
- Openlink, CL-XML (Common Lisp), Kweelt, ...
- Soda3, DB4XML and about ten more

II.5 XML Databases

- XML as exchange protocol in heterogeneous systems
- Moving and sharing data
- Data-centric vs. document-centric
- Publish XML: old data
- Store XML: native format or not
- Querying: just in XML!
- Hard, but solvable problems

Product Categories

- Middleware: data-centric
- XML-Enabled databases: mostly data-centric
- Native XML Databases: data and document-centric
  Examples: eXist, Lucid, Ozone, Tamino, Virtuoso, XIndice, XYZFind, ...
- XML Servers or Query Engines: both
- Content Management Systems: document-centric
- XML Data-binding: data-centric
- Benchmarks: XOO7, XMach-1, XMark, Michigan, XSLT Processor
- XML Generators: ToXgene, Niagara, IBM
- XML Validators, parsers, authoring, etc.

Part III

XML retrieval models and methods

XML retrieval models and methods

- Motivation
- Retrieval Models
- Tasks:
  1. Content-only search
  2. Content-and-structure search
  3. Document clustering

Motivation

- Structure improves precision
- Exploit visual memory

This paragraph is in italics, and it is interesting....

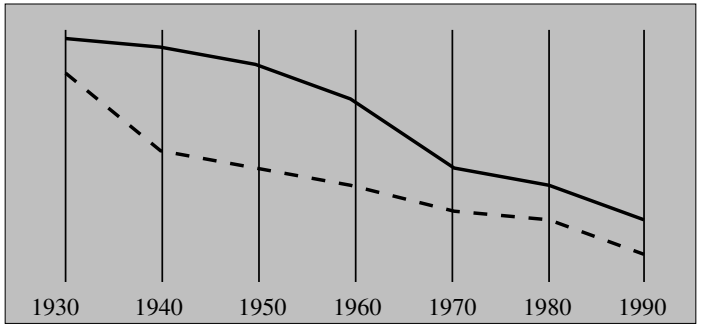


Fig 2.3: Estimated amounts of resources in the earth, per decade.

Retrieval Models

- Relational Model: DB2XML, XML-QL, TSIMMIS, LOREL, ...
- Object-oriented Model: SOX, StruQL, ...
- Extended Vector Model
- Weighted Boolean Model: XQL, ...
- Probabilistic Model: XIRQL, ...

INEX: Initiative for the Evaluation of XML Retrieval

[Gövert and Kazai(2003)]

- Documents: 12 107 articles in XML format, from 18 journals of the IEEE Computer Society, 494 MB
- Queries: 30 Content-only, 30 Content-and-structure
- Relevance Assessments: per retrieved element, by participating groups
- Participants: 36 active groups in 2002

`http://www.is.informatik.uni-duisburg.de/projects/inex/index.html`

III.1 Content-only search

- task
- augmentation
- language models
- Bayesian networks

III.1.1 CO search task

- document as hierarchical structure of nested elements
- type of elements is not considered
- query refers to content only
- query syntax as in standard text retrieval
- task: find smallest subtree (element) satisfying the query

A CO topic from the INEX test collection

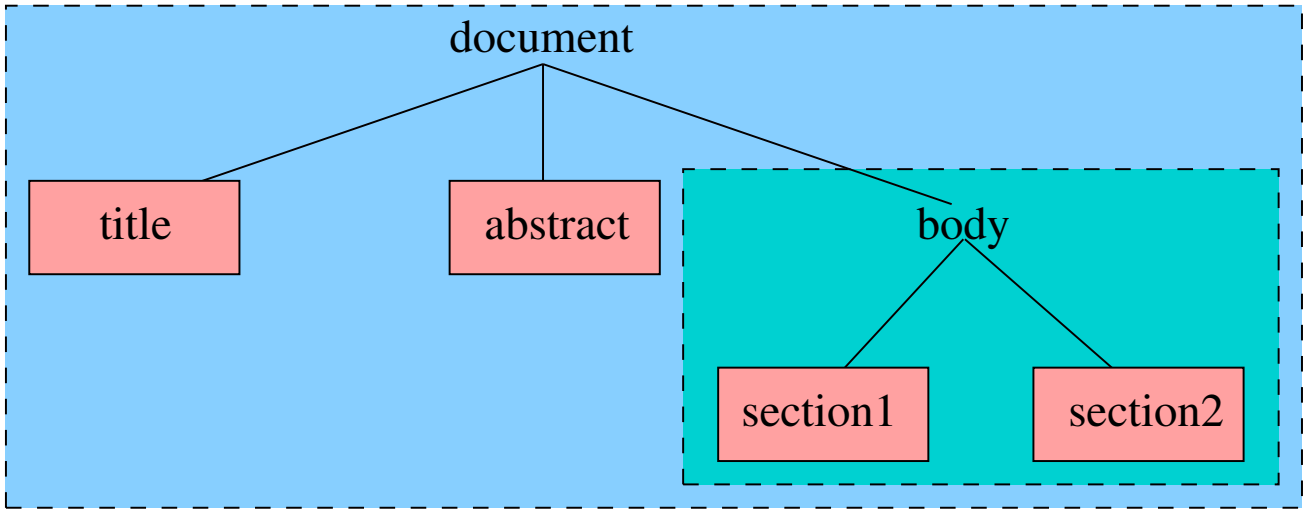
```
<INEX-Topic topic-id="45" query-type="CO" ct-no="056">
<Title> <cw>augmented reality and medicine</cw> </Title>
<Description>
How virtual (or augmented) reality can contribute to improve the medical and
surgical practice.
</Description>
<Narrative>
In order to be considered relevant, a document/component must include
considerations about applications of computer graphics and especially augmented
(or virtual) reality to medicine (including surgery).
</Narrative>
<Keywords>
augmented virtual reality medicine surgery improve computer assisted aided
image
</Keywords>
</INEX-Topic>
```

CO approaches: 1. Disjoint nodes

1. split document text into disjoint nodes

2. index nodes separately

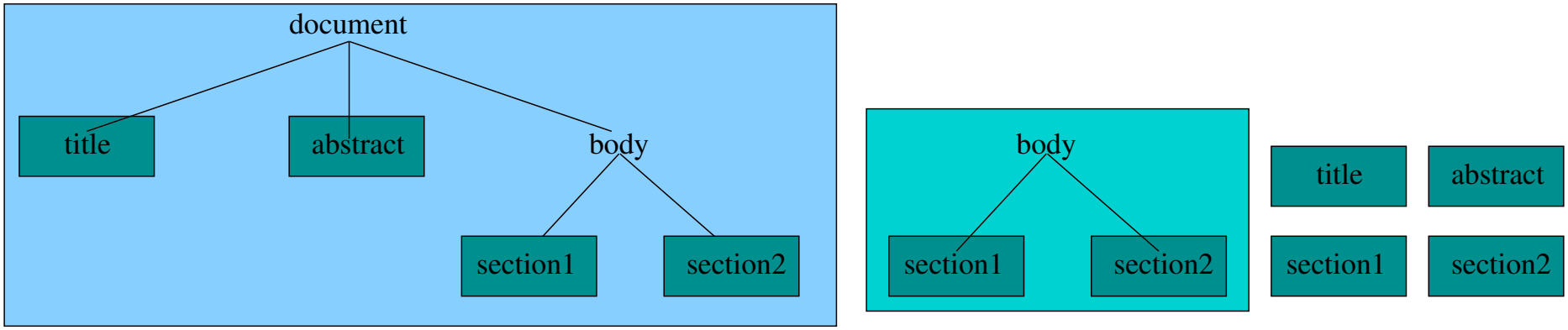
3. aggregate indexing weights for higher-level elements (subtrees)



approaches: augmentation, language models

CO approaches: 2. Indexing subtrees

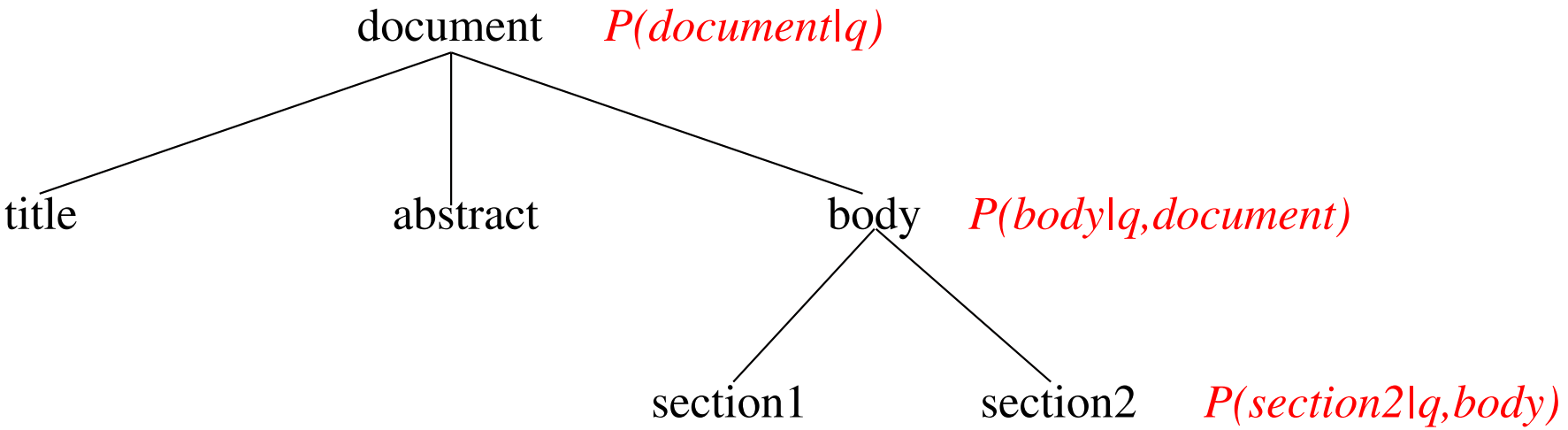
Index all subtrees (elements) of the document



- a) statically (at indexing time)
- b) dynamically (at retrieval time)

CO approaches: 3. Context-dependent retrieval

retrieval weight (RSV) of node depends on RSVs of nodes in the context



approach: Bayesian networks

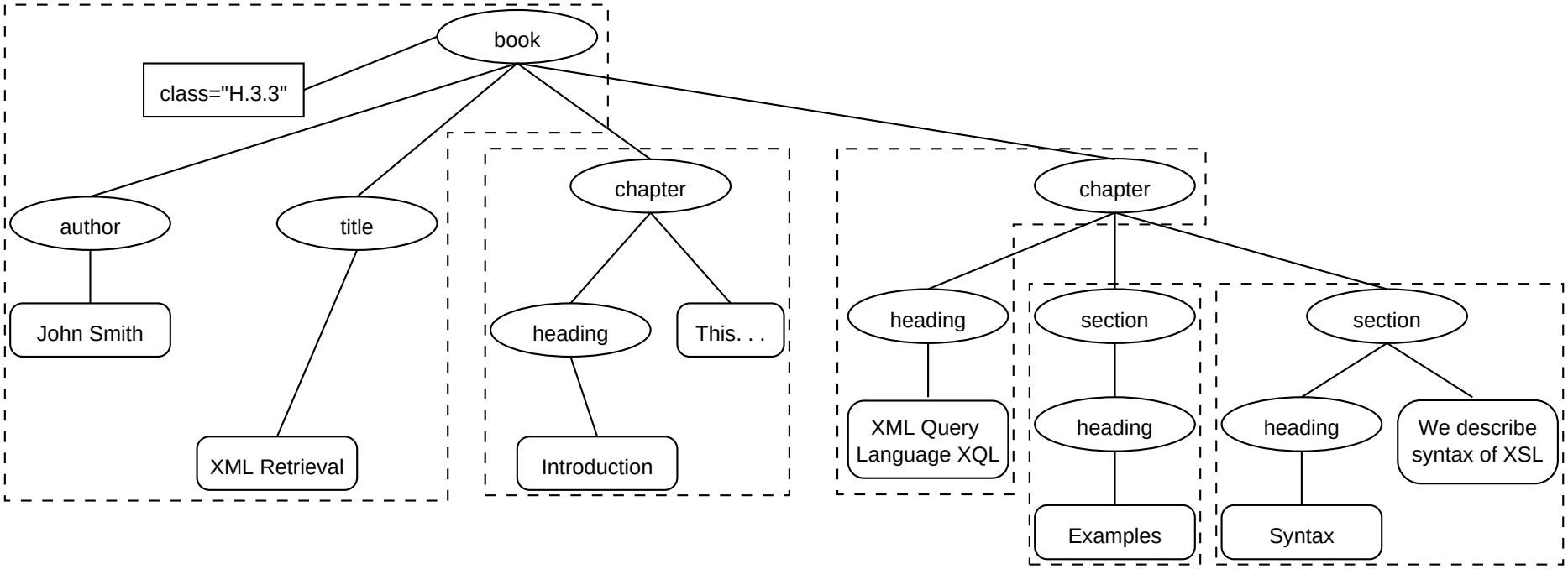
III.1.2 Augmentation

[Fuhr and Großjohann(2001), Gövert et al.(2003)Gövert, Fuhr, Abolhassani

- Choose index nodes as roots of possible answers
  (not all XML elements are meaningful as result units)

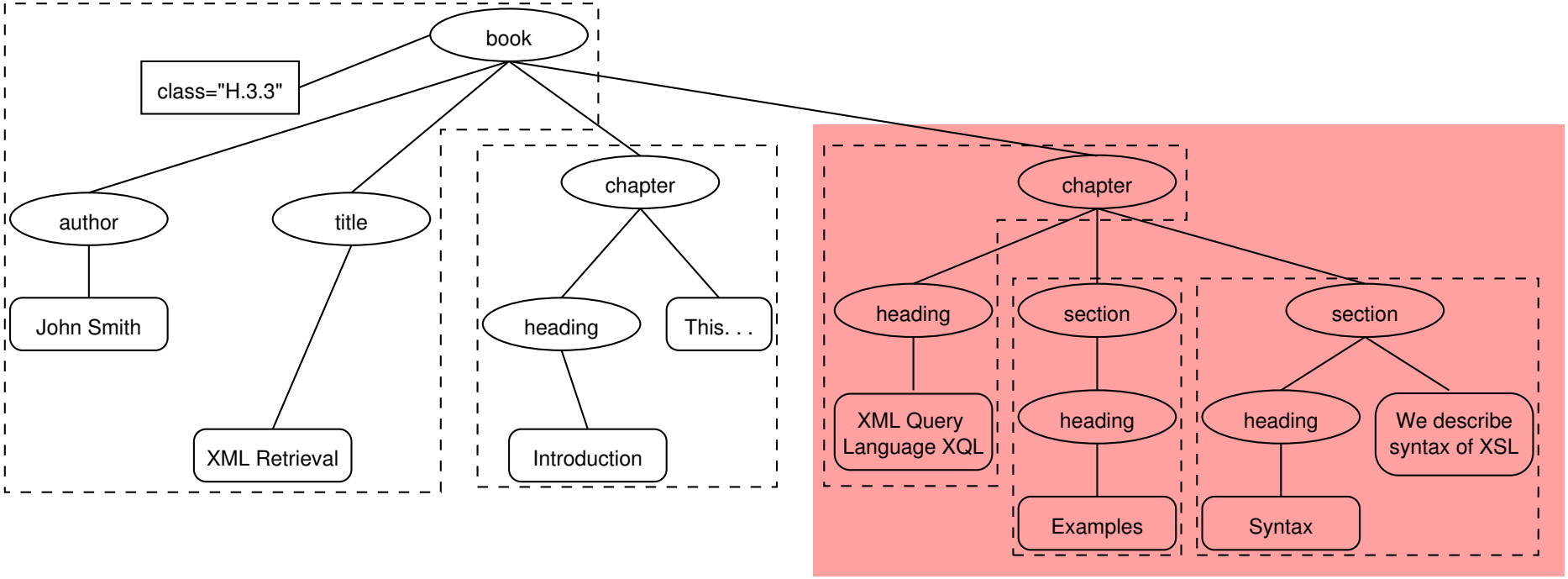
- Augmentation weights for computing trade-off between indexing weights and specificity of answers

Index nodes

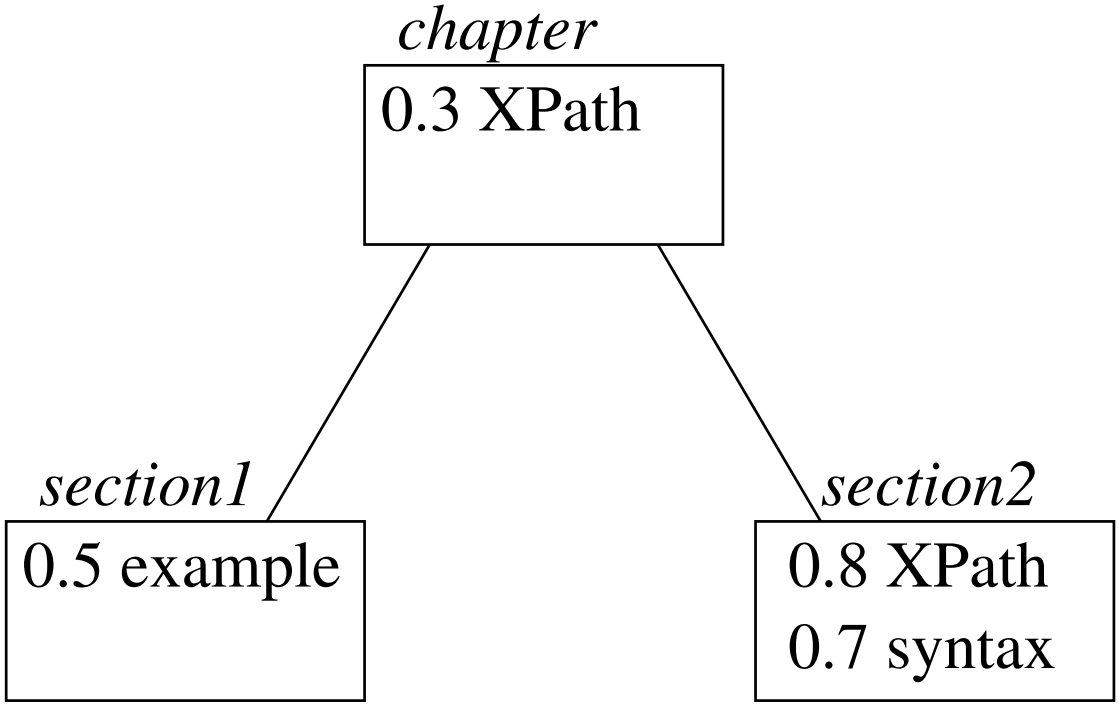


Indexing of text chunks with standard indexing functions (e.g. $tf \cdot idf$)

Index nodes(2)

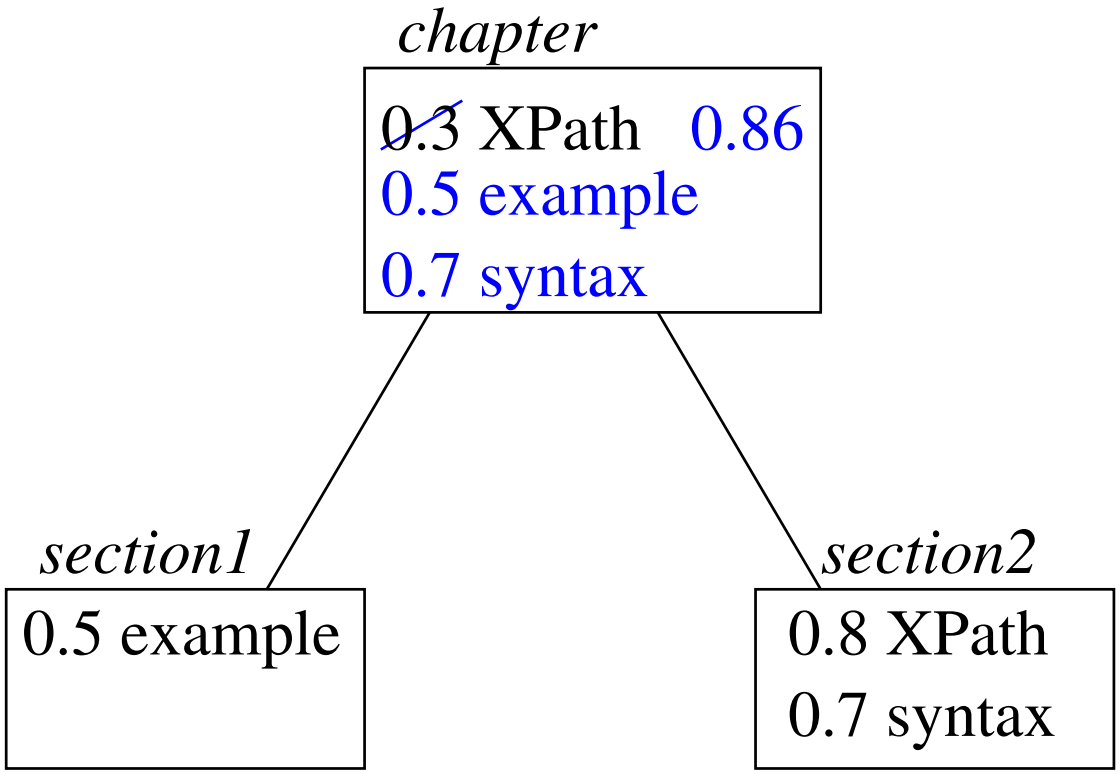


Index nodes(3)



Q1: 'syntax $\wedge$ example'

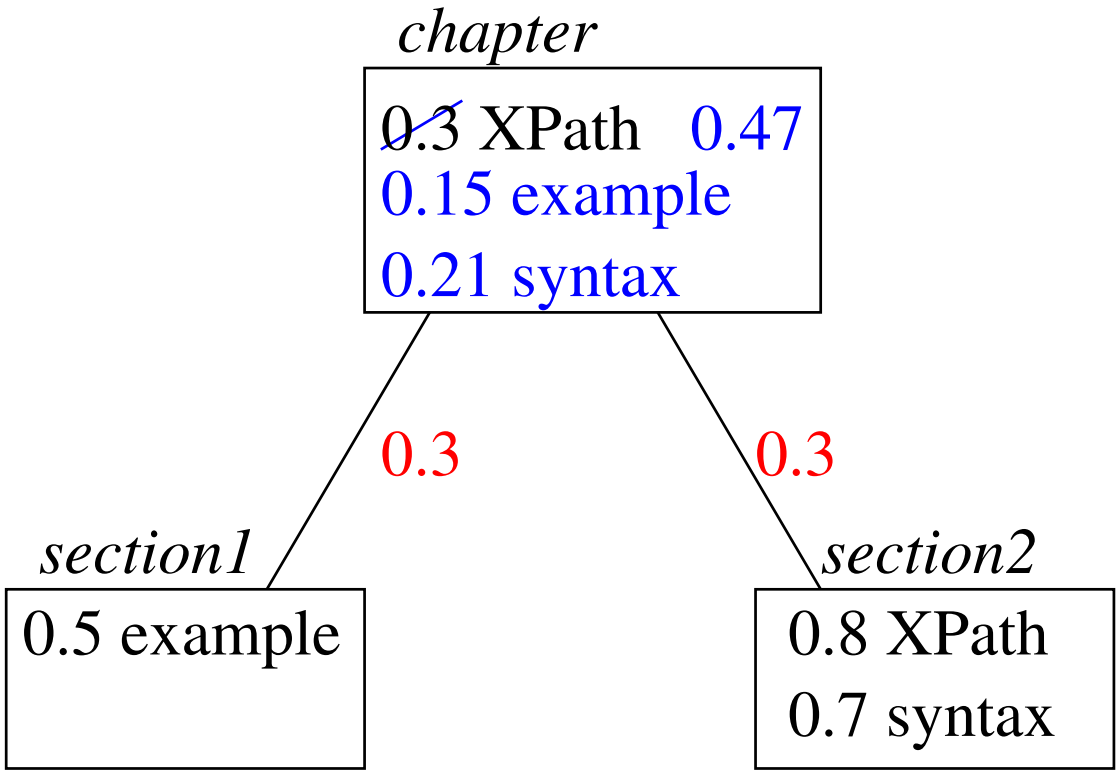
Augmentation by disjunction



Q1: 'syntax $\wedge$ example'

Q2: 'XPath'

Augmentation with propagation weights



Q1: 'syntax $\wedge$ example'

Q2: 'XPath'

III.1.3 Language models

[Ogilvie and Callan(2003)]

- $Q = \{q_1, \ldots, q_n\}$: query as set of words
- θ_D document D's language model

$$P(Q|\theta_D) = \prod_{w \in \{q_1, \ldots, q_n\}} P(w|\theta_D)$$

Estimation of $P(w|\theta_D)$

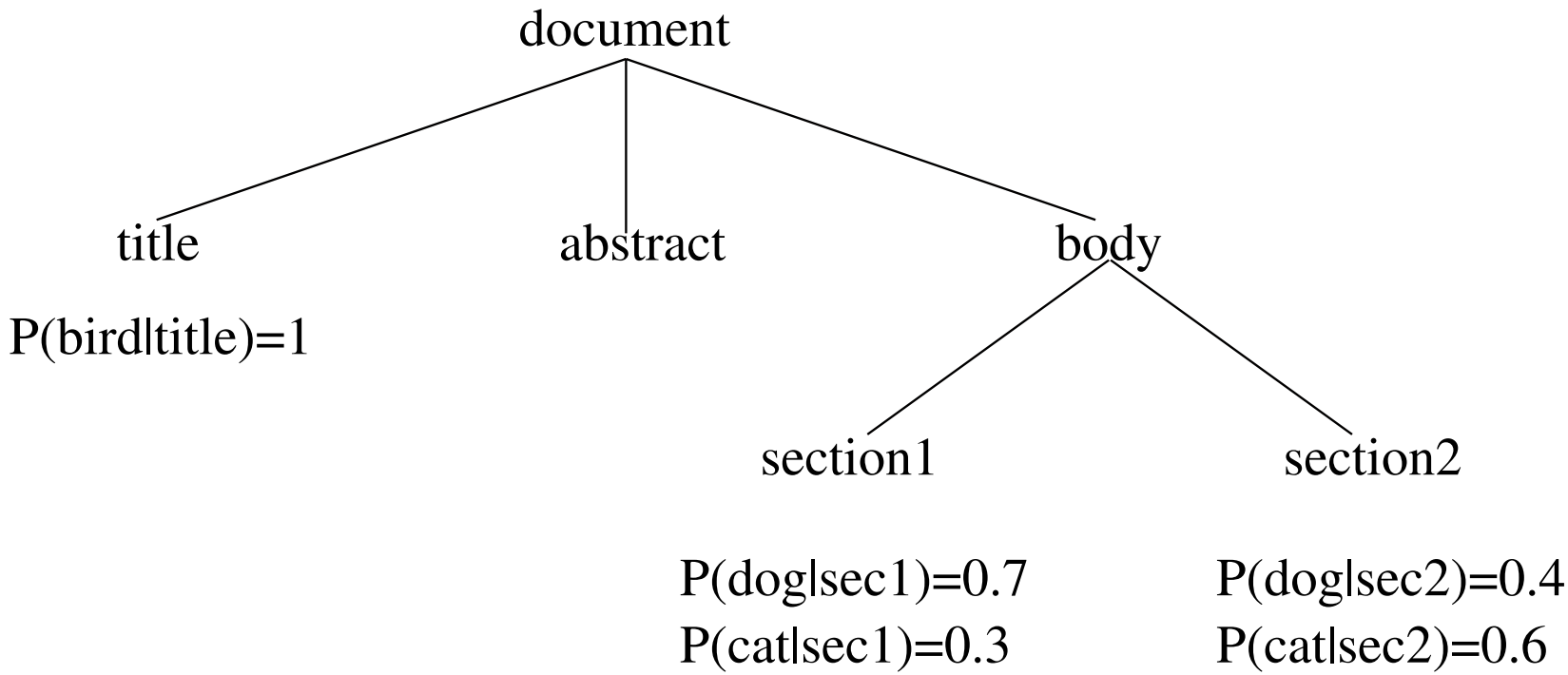
(general idea)

- tf within-document frequency of word w
- $|D|$ length of document D
- N collection size
- n # documents containing w

sparse data $\rightsquigarrow$ $P(w|\theta_D)$ as mixture of document-specific and collection-specific maximum likelihood estimates:

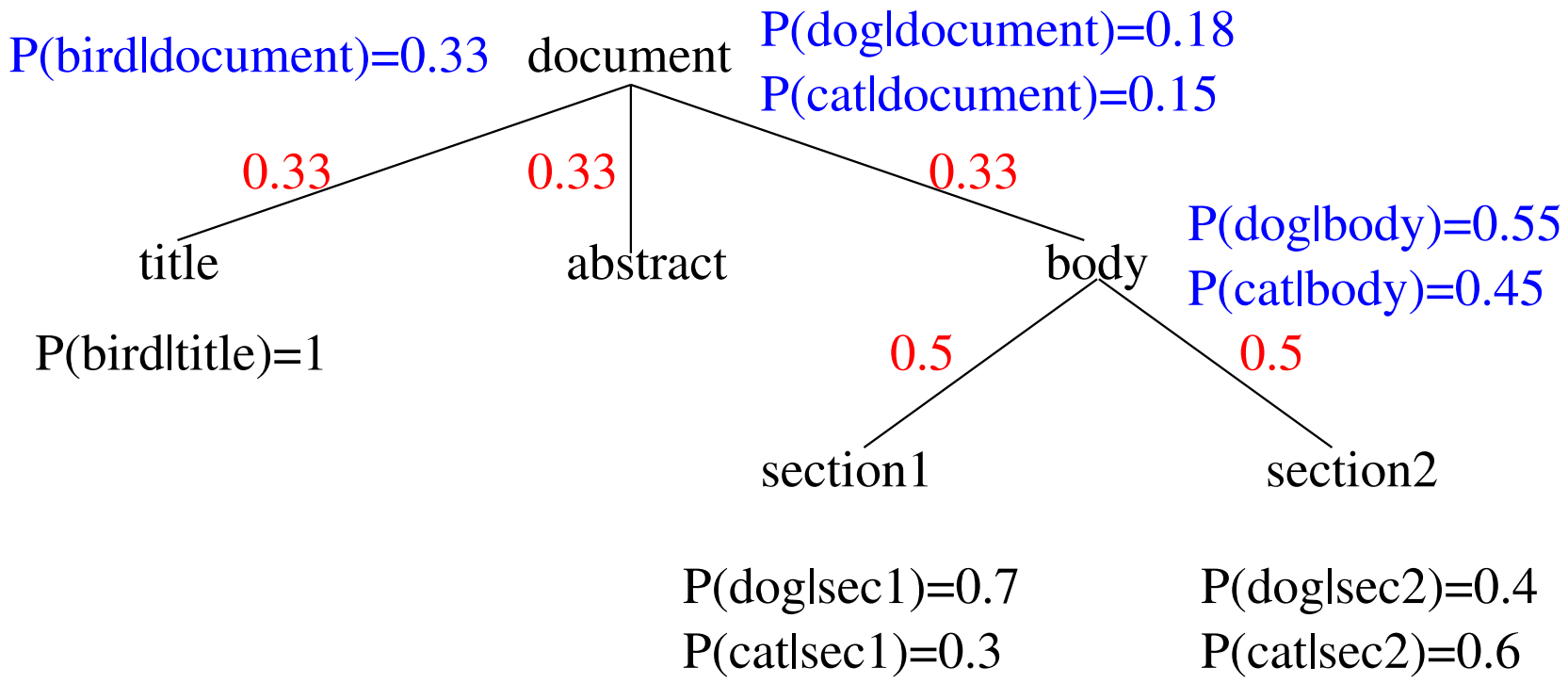
$$P(w|\theta_D) \approx \alpha \frac{tf}{|D|} + (1-\alpha)\frac{n}{N}$$

Element-specific language models $P(w|\theta_e)$

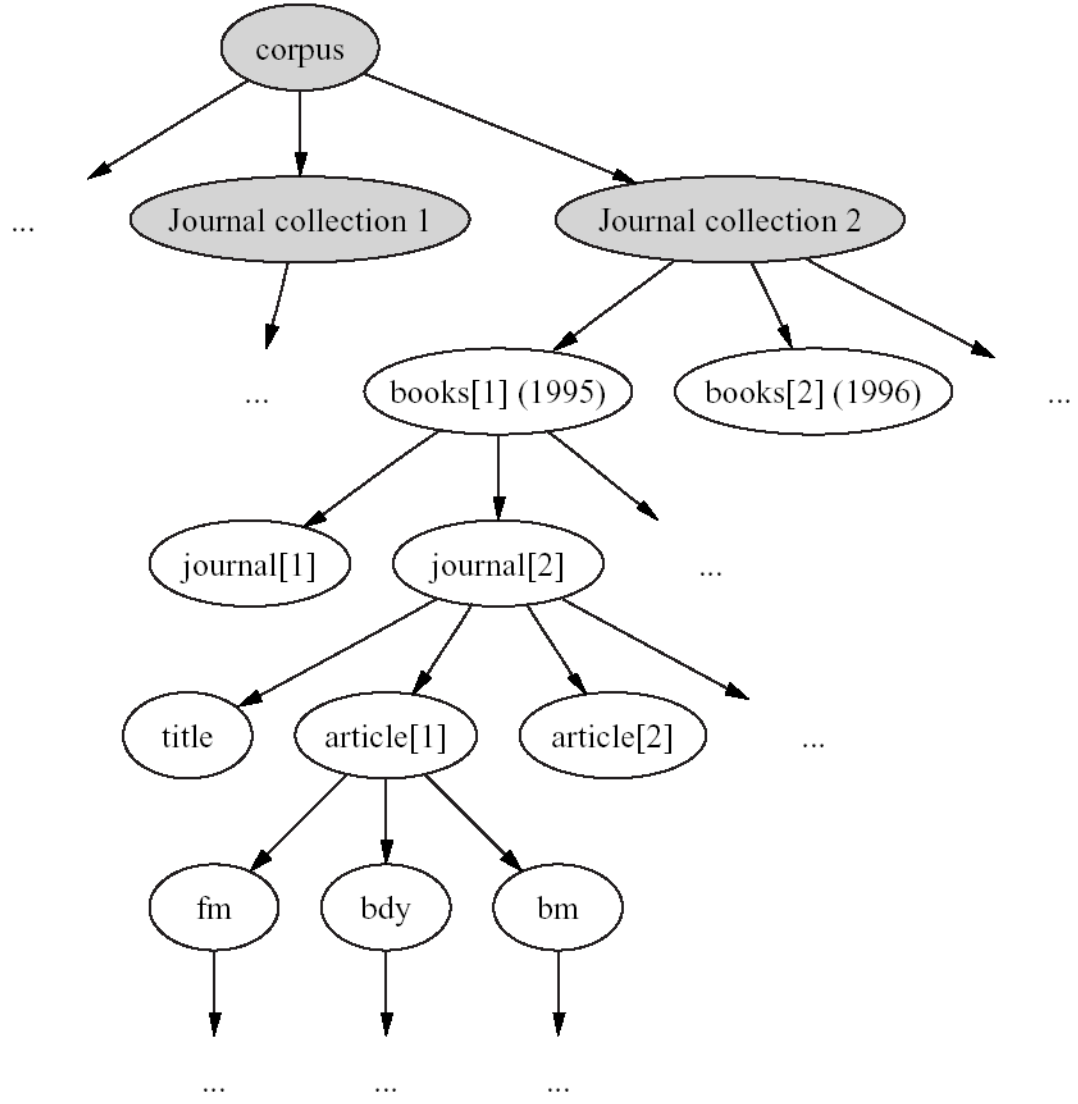


Higher level nodes: mixture of language models

$$P(w|\theta) = \sum \lambda_i P(w|\theta_i)$$

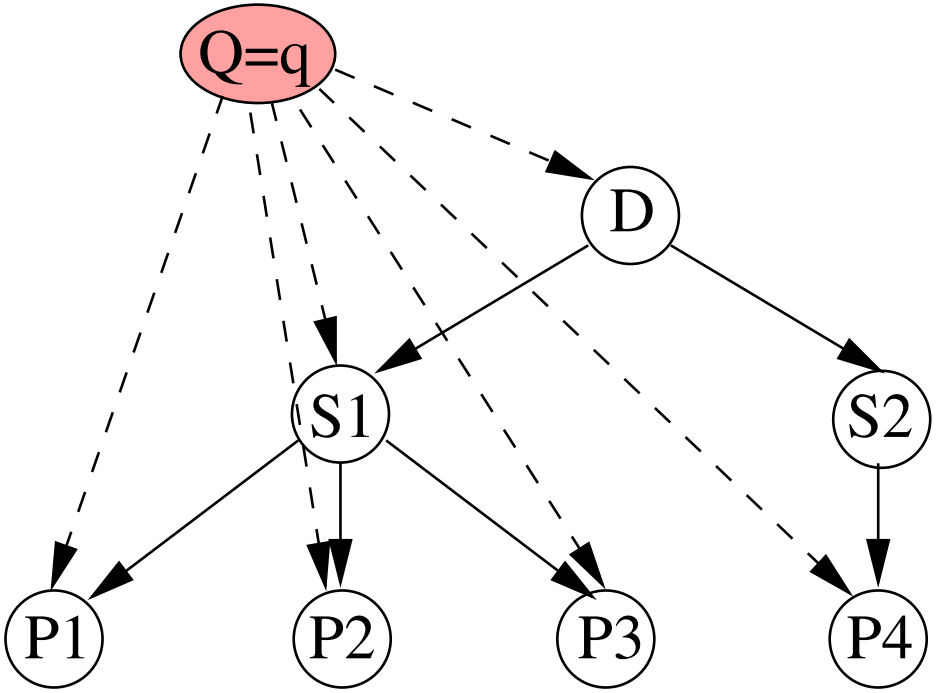


III.1.4 Bayesian Networks



[Piwowarski et al.(2003)Piwowarsl

Bayesian Networks (2)



Q: query = source of evidence

$\mathcal{R} = \{\text{relevant, irrelevant}\}$

estimate relevance probabilities top-down:

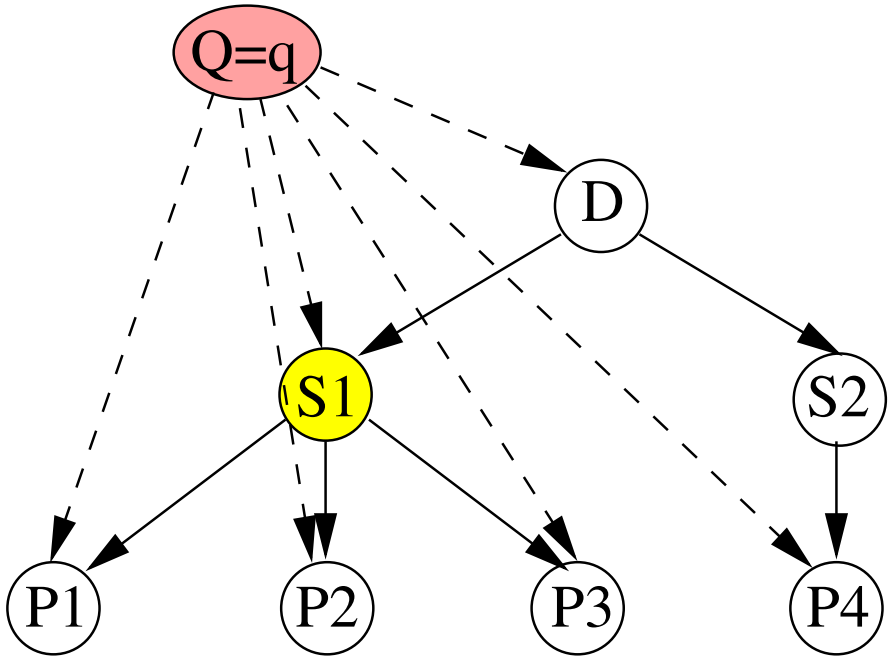
$P(D|Q)$

$P(S|D,Q)$

$P(P|S,Q)$

for $D, S, P \in \mathcal{R}$

Bayesian Networks (3)



$$P(S1|q) = P(S1|D,q)P(D|q) + P(S1|\neg D,q)P(\neg D|q)$$

1. estimate conditional probabilities considering query, element text and weight of parent element:
   $P(D|q) = 0.6$
   $P(S1|d,q) = (0.8, 0.4)$
   $P(P1|s1,q) = (0.3, 0.2)$
2. Bayesian inference for RSV computation
   $P(D|q) = 0.6$
   $P(S1|q) = 0.8 \cdot 0.6 + 0.4 \cdot 0.4 = 0.64$
   $P(P1|q) = 0.3 \cdot 0.64 + 0.2 \cdot 0.36 = 0.264$

III.1.5 Summary: CO search

- most approaches focus on hierarchical structure only
- element types not considered yet
  e.g. title more important than paragraph
- indexing and retrieval functions are extensions of methods for atomic documents

III.2 Content-and-structure search

- Task definition
- Query languages with uncertainty and vagueness:
  XIRQL, JuruXML, ELIXIR
- Type-specific language models
- Dynamic $tf \cdot idf$

III.2.1 Task definition

Content-and-structure (CAS) queries:

queries that contain explicit references to the XML structure, by restricting

- the context of interest:
  `<te>`: target element

- the context of certain search concepts:
  (`<cw>`, `<ce>`) pairs
  (content word, content element)

(INEX 2003: variant of XPath expressions as CAS queries)

Example INEX CAS topic

```
<INEX-Topic topic-id="09" query-type="CAS" ct-no="048">
<Title>
<te>article</te>
<cw>non-monotonic reasoning</cw> <ce>bdy/sec</ce>
<cw>1999 2000</cw> <ce>hdr//yr</ce>
<cw>-calendar</cw> <ce>tig/atl</ce>
<cw>belief revision</cw>
</Title>
<Narrative>
Retrieve all articles from the years 1999-2000 that deal with works on non-
monotonic reasoning.  Do not retrieve CfPs/calendar entries
</Narrative>
<Keywords> non-monotonic reasoning belief revision </Keywords>
</INEX-Topic>
```

CAS approaches

- vague (probabilistic/fuzzy) XPath:
  - *ELIXIR*
  - *XIRQL*
  - *JuruXML*
- element-specific indexing
  - vector space: *dynamic* $tf \cdot idf$
  - language models: *CMU language model*

III.2.2 ELIXIR

[Chinenyanga and Kushmerik(2001)]

- indexing weights + vague predicates
- vague joins
- XPath + result construction

ELIXIR: example queries

comparison with constant value

```
CONSTRUCT <item>$b</>
WHERE <items.(book|cd)>$t</> in "db.xml",
      $t ~ "Ukrainian cookery".
```

vague join

```
CONSTRUCT <item>$b</>
WHERE <items.book>$b</> in "db.xml",
      <items.cd>$c</> in "db.xml",
      $b ~ $c.
```

ELIXIR: retrieval function

- Q=(Q_1, Q_2): query as pair of condition sets
- Q_1 set of vague comparisons with constant of the form $\$x_j \sim c$
- Q_2 set of vague join conditions of the form $\$x_j \sim \$x_k$
- v_j current value in document D bound to variable $\$x_j$

$$\varrho(Q, D) = \prod_{\$x_j \sim c \in Q_1} sim(v_j, c) \cdot \prod_{\$x_j \sim \$x_k \in Q_2} sim(v_j, v_k)$$

III.2.3 XIRQL

[Fuhr and Großjohann(2001)]

extension of XPath by

- weighting and ranking
- data types with vague predicates
- structural vagueness

XIRQL: weighting and ranking

- indexing weights / vague predicates

- probabilistic interpretation of boolean connectors
  e.g. $P(a \wedge b) = P(a) \cdot P(b)$

- weighting of query terms
  e.g. $P(\textit{wsum((0.6,a), (0.4,b))} = 0.6 \cdot P(a) + 0.4 \cdot P(b)$

Example query

```
<te>tig</te>
<cw>Survey on Software Engineering</cw>
<cw>programming languages</cw><ce>sec</ce>
<cw>
  software engineering survey,
  programming survey,
  programming tutorial,
  software engineering tutorial
</cw>
<ce>tig</ce>
```

XIRQL expression

```
/article[wsum(1.0,.//* $stem$ "survey",
              2.0,.//* $stem$ "software",
              1.0,.//* $stem$ "engineering") $and$
          .//sec//(.* $stem$ "programming" $and$
                   .* $stem$ "languages")]
 //tig[  (.//* $stem$ "engineering" $and$
              .//* $stem$ "software" $and$
              .//* $stem$ "survey") $or$
              (.//* $stem$ "programming" $and$
              .//* $stem$ "survey")   $or$
              (.//* $stem$ "programming" $and$
              .//* $stem$ "tutorial")   $or$
              (.//* $stem$ "engineering" $and$
              .//* $stem$ "software" $and$
              .//* $stem$ "tutorial")  ]
```

XIRQL: Data types with vague predicates

- XML markup allows for detailed markup of text elements
- Exploit markup for more precise searches
- Consider also vagueness and imprecision of IR

vague predicate:

- compares two values of a specific data-type
- returns (probabilistic) matching value

XIRQL: Data types with vague predicates

"Search for an artist named Ulbrich, living in Frankfurt, Germany about 100 years ago"
$\rightsquigarrow$ *Ernst Olbrich, Darmstadt, 1899"*

P(Olbrich $\approx_p$ Ulbrich)=0.8	(phonetic similarity)
P(1899 $\approx_n$ 1903)=0.9	(numeric similarity)
P(Darmstadt $\approx_g$ Frankfurt)=0.7	(geographic distance)

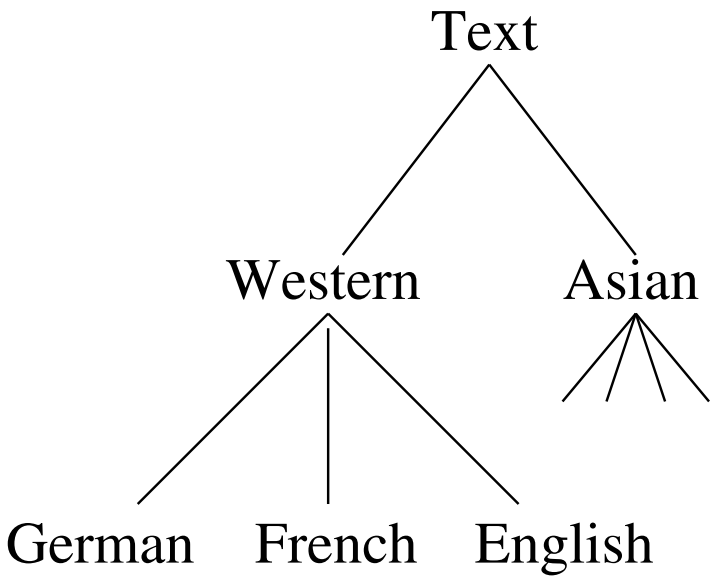
(Extensible) data types for document-centric view:

- person names
- dates
- geographic locations
- classifications
- application-specific data (e.g. chemical formulas)
- images
- ...

XIRQL: Extensible type hierarchy

Extensible type hierarchy with vague predicates for each data type

- Text: substring-match
- Western language: single word search, truncation, word distance
- English text: stemming, noun phrases



Data types of XML elements defined in (extended) XML schema

III.2.4 JuruXML

[Mass et al.(2003)Mass, Mandelbrod, Amitay, Y., and Soffer]

element-specific indexing + vector space model:

1. transform query into set of (term,path)-conditions

2. vague matching of path conditions

3. modified cosine similarity as retrieval function

JuruXML: 1. Transform query

INEX query:

```
<Title>
<te>article/bm/bib/bibl/bb</te>
<cw> hypercube, mesh, torus,
nonnumerical, database </cw>
<ce>article/bm/bib/bibl/bb</ce>
<cw>1996 or 1997</cw>
<ce>article/fm/hdr/hdr2/pdt</ce>
</Title>
<Keywords>
hypercube mesh torus
non-numerical database
</Keywords>
```

(term,path)-pairs

hypercube, `article/bm/bib/bibl/bb`
mesh, `article/bm/bib/bibl/bb`
torus, `article/bm/bib/bibl/bb`
nonnumerical, `article/bm/bib/bibl/bb`
database, `article/bm/bib/bibl/bb`
1996, `article/fm/hdr/hdr2/pdt`
1997, `article/fm/hdr/hdr2/pdt`
hypercube
mesh
torus
non-numerical
database

JuruXML: 2. Vague matching of path conditions

- c_i^q path condition for term t_i

- c_i^d actual path of term t_i in the document

similarity function:

$$cr(c_i^Q, c_i^D) = \begin{cases} \frac{1+|c_i^Q|}{1+|c_i^D|} & \text{if } c_i^Q \text{ subsequence of } c_i^D \\ 0 & \text{otherwise} \end{cases}$$

Example:

$cr($`article/bibl`, `article/bm/bib/bibl/bb`$) = 3/6 = 0.5$

JuruXML: 3. Retrieval function

Standard vector space model: cosine similarity

- $w^Q(t_i)$: query term weight of term t_i

- $w^D(t_i)$: indexing weight of term t_i in the document

$$\varrho(Q, D) = \frac{1}{|Q||D|} \sum_{t_i \in Q \cup D} w^Q(t_i) \cdot w^D(t_i)$$

JuruXML: modified cosine similarity

- $w^Q(t_i, c_i^Q)$: query term weight of pair (t_i, c_i^Q)

- $w^D(t_i, c_i^D)$: indexing weight of pair (t_i, c_i^D) in the document

$$\varrho(Q, D) = \frac{1}{|Q||D|} \sum_{(t_i, c_i^Q) \in Q} \sum_{(t_i, c_{i_j}^D) \in D} w^Q(t_i, c_i^Q) \cdot w^D(t_i, c_{i_j}^D) \cdot cr(c_i^Q, c_{i_j}^D)$$

JuruXML: Alternative approach

Merging contexts

- vague matching of path conditions as before

- separate term and path weights in the retrieval function, weighting by length of query path only:

$$w(c_i^Q) = 1 + |c_i^Q|$$

retrieval function:

$$\varrho(Q, D) = \frac{1}{|Q||D|} \sum_{(t_i, c_i^Q) \in Q} w^Q(t_i) \cdot w^D(t_i) \cdot w(c_i^Q)$$

Experimental results:

merging contexts better than (term,path)-pairs

III.2.5 Language models for CAS search

[Ogilvie and Callan(2003)]

- $Q = \{(t_1, c_1^Q), \ldots, (t_n, c_n^Q)\}$: query as set of (term,path) conditions
- θ_D document D's language model
- θ_e element e's language model

Element- vs. document specific language models:

P('transaction'|`Journal_title`)=0.3
P('transaction'|`body`)=0.001
P('transaction'|`document`)=0.3

instead of document-specific probabilities $P(t_i|\theta_D)$,
use element-specific probabilities $P(t_i|\theta_e)$
(consider only term occurrences satisfying the path conditions c_i^Q)

III.2.6 Dynamic $tf \cdot idf$

[Grabs and Schek(2003)]
(similar to element-specific language models)

- N_C # elements with path c in collection
- $ef(t_i, c_j)$ # elements with path c_j containing term t_i

inverse element frequency:

$$ief(t_i, c_i) = \frac{\sum_{c \in c_i} N_c}{\sum_{c \in c_i} ef(t_i, c)}$$

Retrieval function

- $Q = \{(t_1, c_1^Q), \ldots, (t_n, c_n^Q)\}$: query
- c_i^Q set of paths (conditions)
- $tf(t_i, e)$ term frequency of term t_i in element e

retrieval function:

$$\varrho(Q, e) = \sum_{(t_i, c_i^Q) \in Q} ief(t_i, c_i) \cdot \frac{tf(t_i, e)}{1 + tf(t_i, e)}$$

III.2.7 Summary: CAS queries

- features of appropriate query language yet to be defined
- strict vs. vague interpretation of structural conditions
- INEX 2002: best results for vague interpretation of all conditions (JuruXML, 2nd approach)

III.3 Clustering of XML documents

[Doucet and Ahonen-Myka(2003)]

document similarity:
based on particular document representation
N-dimensional vector, N = # document features
feature sets:

- terms only
- tags only
- terms+tags
- (term,tag)-pairs

feature weighting in the document vector: e.g. $tf \cdot idf$ weights

similarity measure: vector similarity, e.g. cosine measure:

$$\cos(\vec{d_1}, \vec{d_2}) = \frac{\vec{d_1} \cdot \vec{d_2}}{|\vec{d_1}||\vec{d_2}|}$$

Clustering methods

n: # documents, k: # clusters

- hierarchical clustering: $O(n^2)$
  single link, complete link, average link

- partitional clustering: $O(k \cdot n)$
  k-means

k-means clustering algorithm

1. Initialization
   - k documents are chosen as initial centroids
   - assign each document to the closest centroid
2. Iterate
   - compute the centroid of each cluster
   - assign each document to the closest centroid
3. Stop condition
   - as soon as the centroids are stable

Measuring clustering quality

external quality: comparison of clusters with external classification

- entropy distribution of classes within clusters
- purity largest class in a cluster / cluster size

INEX results

(external classification by journal)

Table 1: Results of k-way clustering for k=5

Features	Text	Tags	Text + Tags
Entropy	0.711	0.836	0.812
Purity	0.301	0.211	0.216
Clustering Time	150s.	4s.	160s.

Table 2: Results of k-way clustering for k=15

Features	Text	Tags	Text + Tags
Entropy	0.633	0.798	0.678
Purity	0.379	0.228	0.372
Clustering Time	754s.	11s.	837s.

Table 3: Results of k-way clustering for k=20

Features	Text	Tags	Text + Tags
Entropy	0.598	0.775	0.677
Purity	0.413	0.237	0.332
Clustering Time	1101s.	15s.	1191s.

Table 4: Results of k-way clustering for k=35

Features	Text	Tags	Text + Tags
Entropy	0.568	0.758	0.612
Purity	0.454	0.254	0.385
Clustering Time	2016s.	25s.	2215s.

- text terms alone give best results
- text+tags: problem with weighting of tags vs. terms
- no results for (term,tag)-pairs

III.4 Summary: Models and methods

- CO search
  - well-defined task
  - various approaches using extensions of classical models
- CAS search
  - syntax and semantics of appropriate query language yet to be defined
  - approaches should support vague interpretation of all types of conditions
- Other tasks
  - all classical IR tasks also relevant for XML documents
  - methods should consider (vague interpretations of) nested structure and element typing

Part IV

XML Retrieval Algorithms and Data Structures

Content

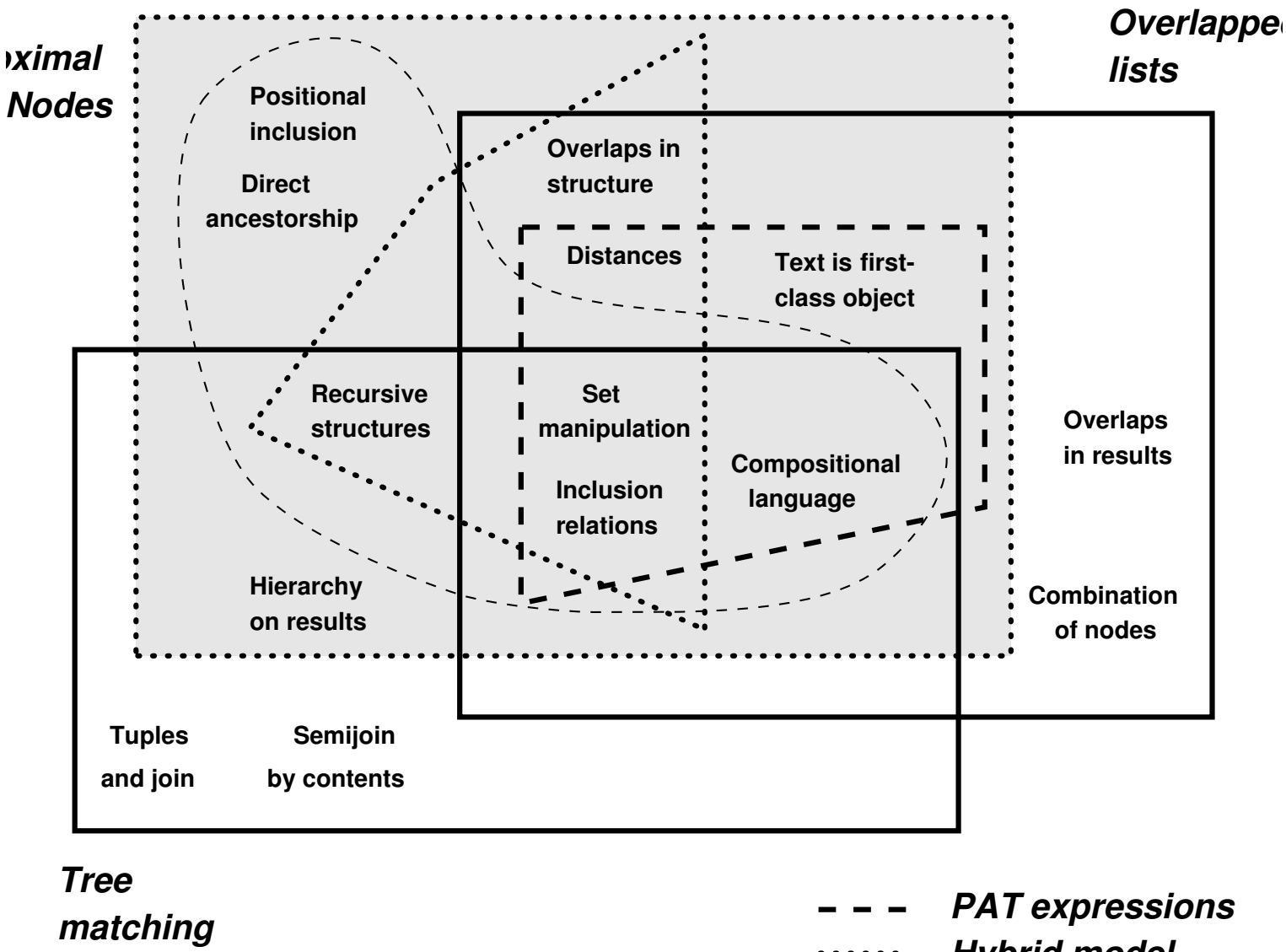
- Structured text models
- Example: Proximal Nodes
- Indexing and Searching
- Examples: Toxin, IXPN
- XML streams

IV.1 Structured Text Models

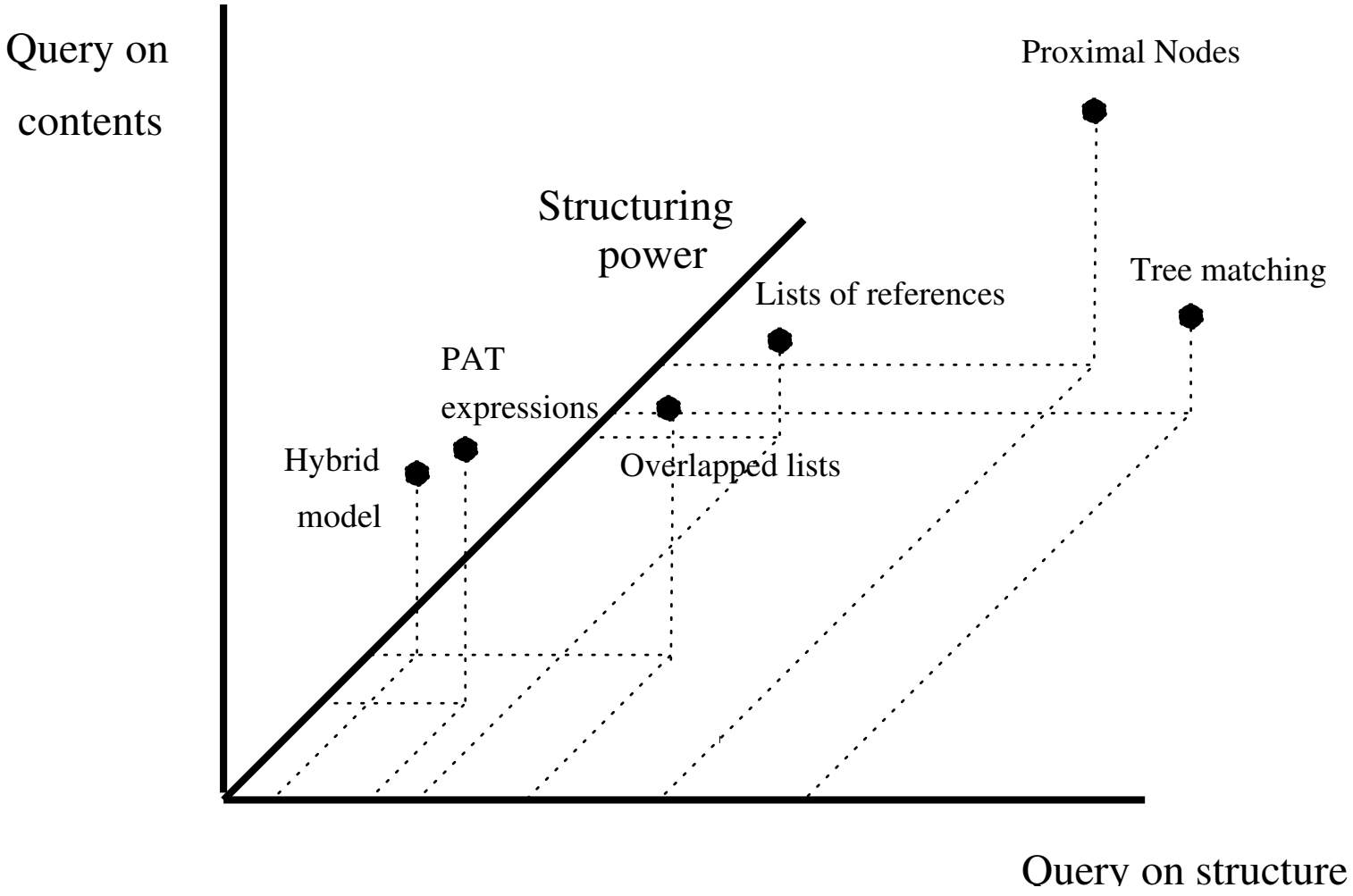
- Motivation
  - Pre-XML!
  - Important as XQuery lacks a formal data model
  - Basis for fast implementations
  - Trade-off: expressiveness vs. speed
- Models in order of expressiveness
  - The hybrid model (R. Baeza-Yates, 1993)
  - PAT expressions (G. Gonnet et al., 1989)
  - Overlapped lists (C. Clarke et al., 1995)
  - Reference lists (I. MacLeod, 1991)
  - Proximal Nodes (G. Navarro and R. Baeza-Yates, 1995)
  - Tree matching (P. Kilpeläinen and H. Mannila, 1992)
  - *p-strings* (G. Gonnet and F. Tompa, 1987)
- Survey in SIGMOD Record [Baeza-Yates and Navarro(1996)]

Model	Structuring mechanism	Contents query language	Structural query language
Hybrid Model [BY94]	IR-like documents + fields + text. Fields can nest and overlap, but this cannot be later queried. It is a flat model.	Query = matches + documents. Almost all the language is oriented to matches, which are seen as their start point. Expresses distances. Has separate set manipulation tools for matches and documents.	Only to restrict match points to be in a given field or to select fields including match points (selected fields are then seen as match points). Very simple in general.
PAT Expressions [ST92]	Dynamic definition of regions, by pattern matching. Each region is a flat list of disjoint segments.	Powerful matching language. Handles points and regions. Has set manipulation operations. Expresses distances.	Simple, since structures are flat. Can express inclusion, set manipulation and some very specialized operations.
Overlapped Lists [CCB95]	A set of regions, each one a flat list of possibly overlapping segments.	Not specified. Words and regions are seen in a uniform way, by an inverted list metaphor.	Results can overlap, but not nest. Can express inclusion, union and combinations.
Lists of References [Mac91]	A single hierarchy with attributes in nodes and hypertext links.	Text queries can only be used to restrict other queries.	Results are flat and from the same constructor. Can express inclusions, complex context conditions and set manipulation.
Proximal Nodes [NBY95b]	A set of disjoint strict trees (views). Views can overlap. Nodes cannot be dissociated from segments.	Text is a special view. Text queries are leaves of query syntax trees. Text content is accessed only in matching subqueries, thereafter it is seen just as segments. There are powerful distance operators, and special set operators for text.	Can express inclusion, positions, direct and transitive relations, and set manipulation. Can express complex context conditions if they involve proximal nodes.
Tree Matching [KM93]	A single tree, with strict hierarchy. No more restrictions.	Not specified, orthogonal to the model. It can only be used to restrict sets of nodes of the tree (leaves of patterns). Weak link between content and structure.	Powerful tree pattern matching language. Can distinguish order but not positions nor direct relationships. Can express equality between different parts of a structure, by using logical variables. Set manipulation features via logical connectives.

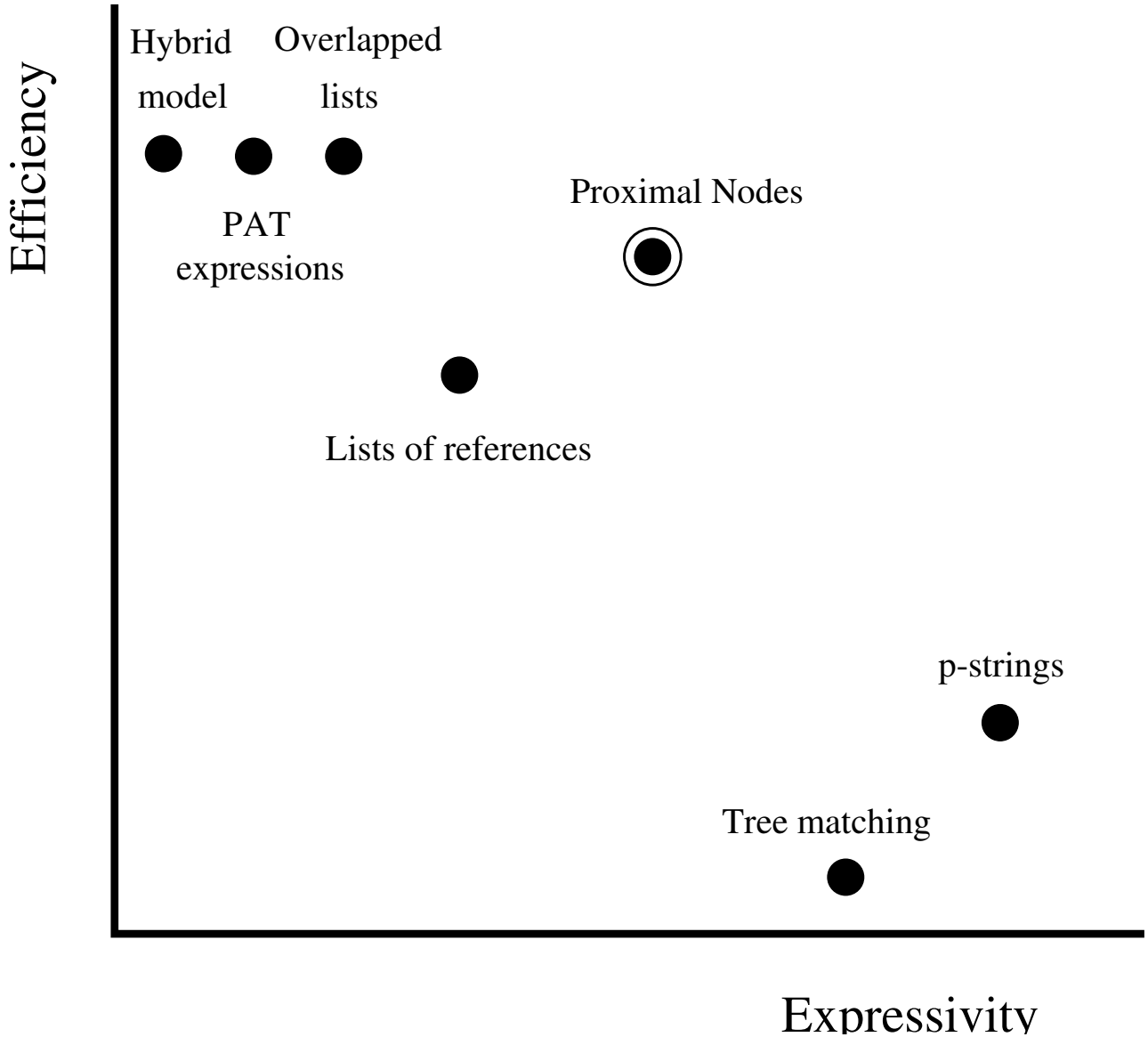
Expressiveness (1)



Expressiveness (2)

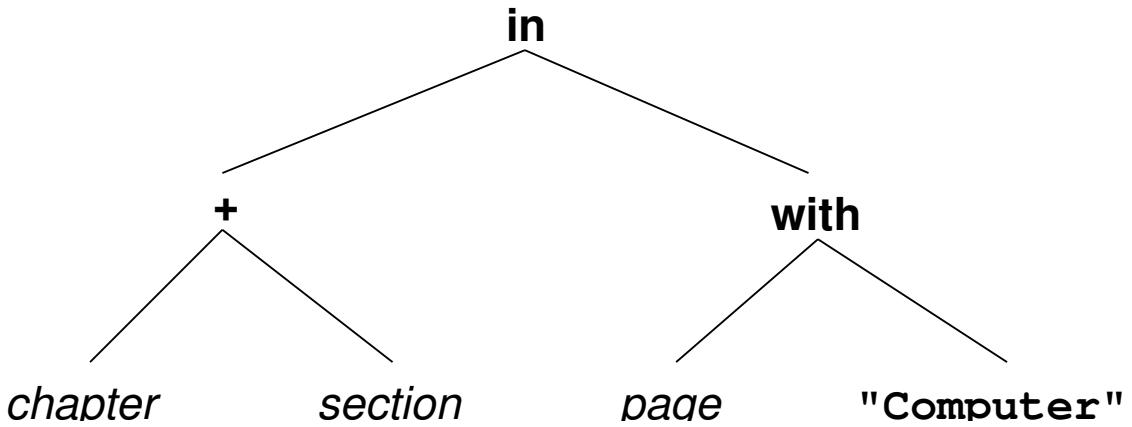


Final Comparison



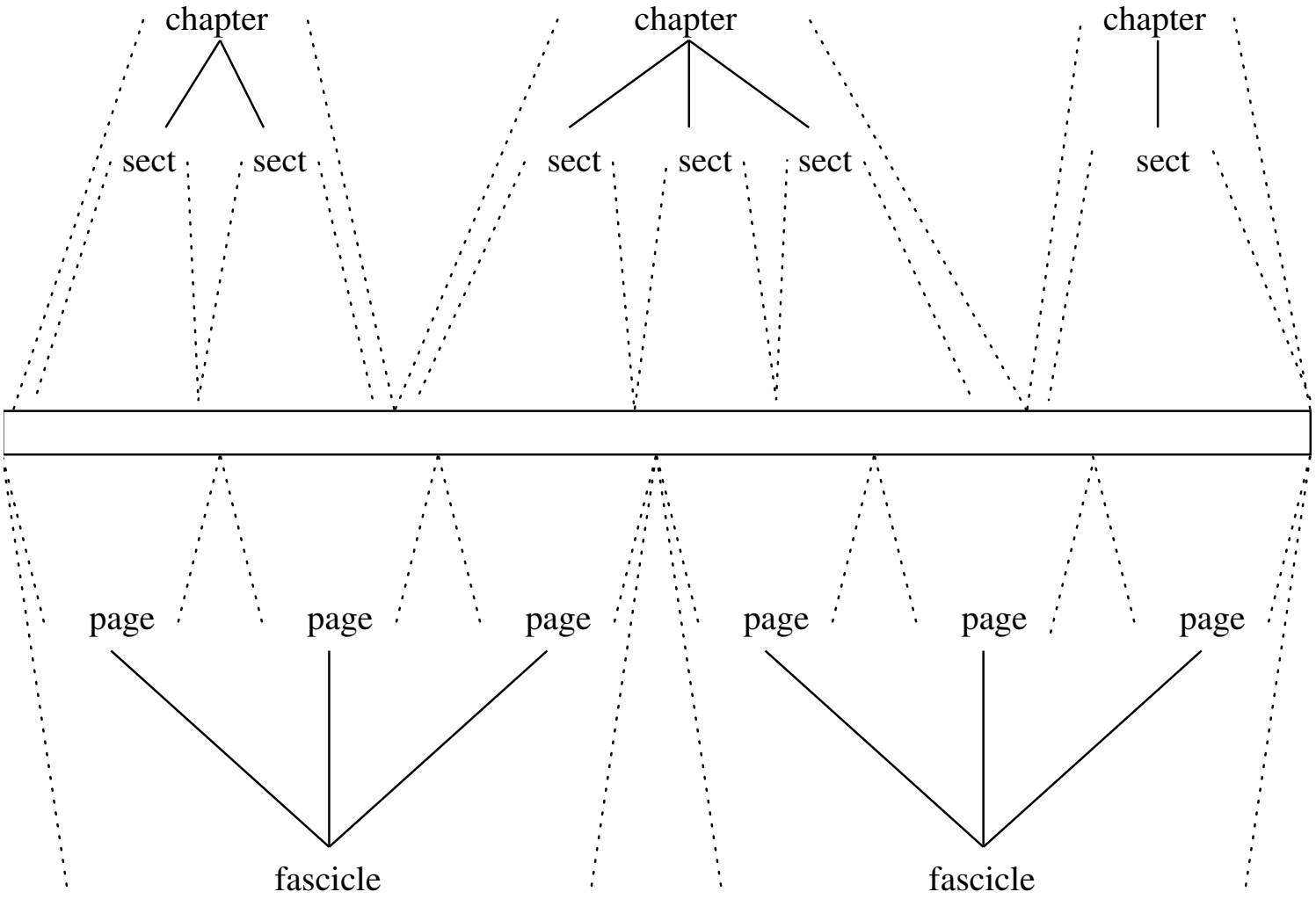
IV.2 Example: Proximal Nodes

- Hierarchical structure
- Set-oriented language
- Avoid traversing the whole database
- Bottom-up strategy
- Solve leaves with indexes
- Operators work with near-by nodes
- Operators cannot use the text contents
- Most XPath and XQuery expressions can be solved using this model [Baeza-Yates and Navarro(2002)]



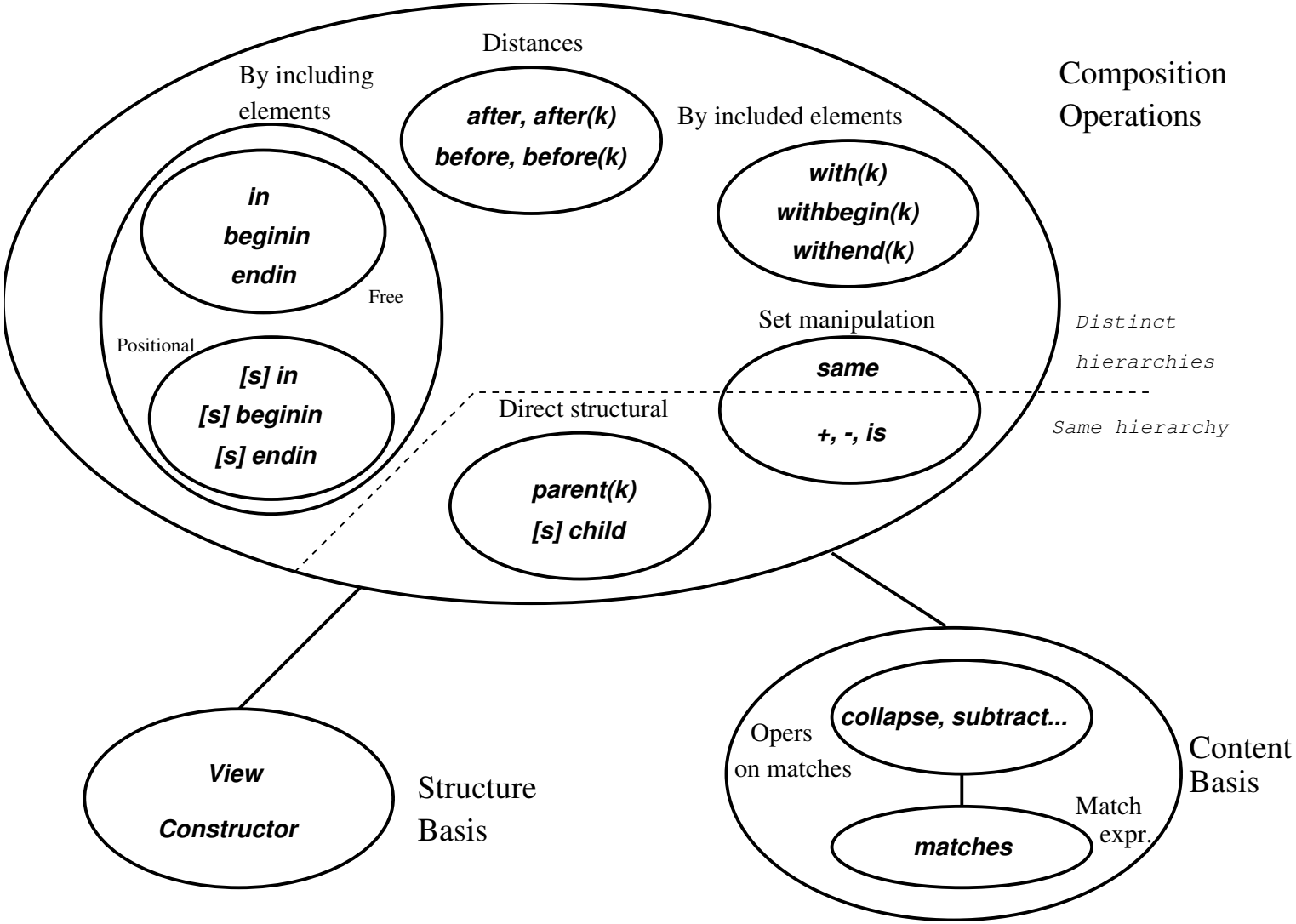
Data Model

- Text = sequence of symbols (filtered)
- Structure = set of independent and disjoint hierarchies, "views"
- Node = Constructor + Segment
- Segment of node $\supseteq$ segment of children
- Text view, to modelize pattern-matching queries
- Query result = subset of some view


chapter
chapter
chapter
sect
sect
sect
sect
sect
sect
page
page
page
page
page
page
fascicle
fascicle

Query Language

Set-oriented and only efficiently implementable operations

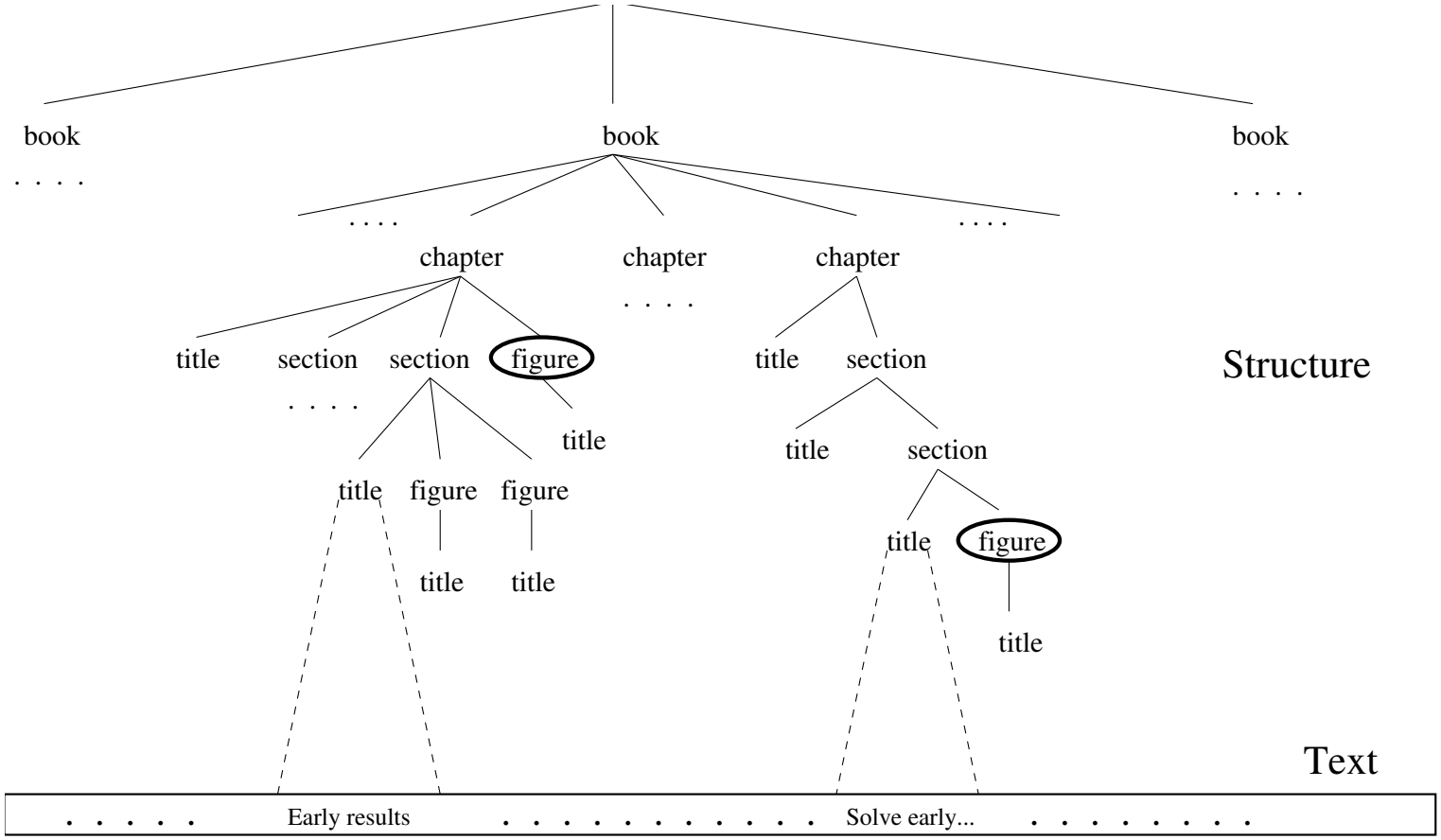


Examples (1)

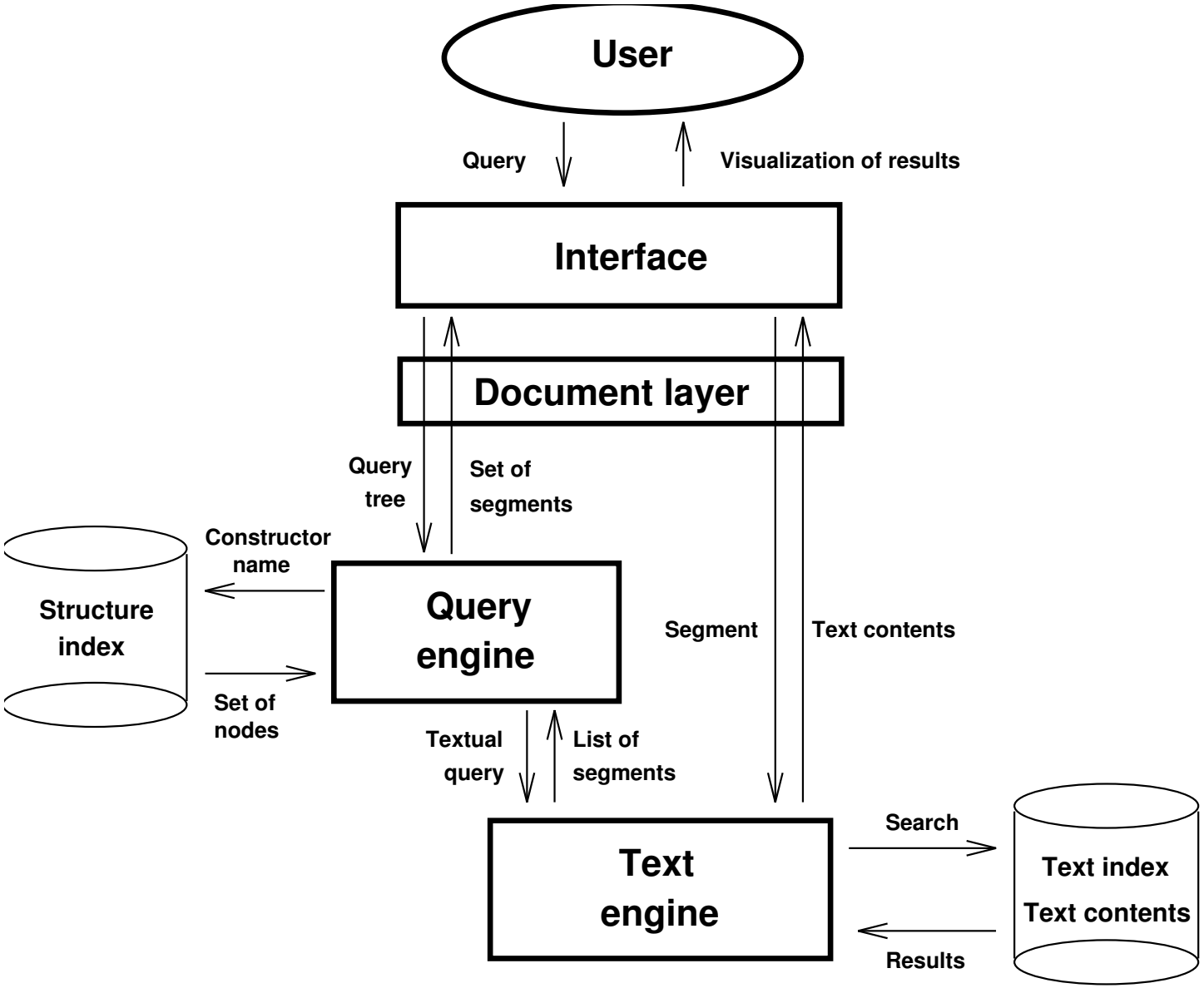
- `italics` **before**`(100) (figure` **with** `"earth") (page)`
- `chapter` **parent** `(title` **same** `"Architecture")`
- `paragraph` **before** `(paragraph` **with** `("Computer"` **before**`(10) "Science" (paragraph))) (page)`
- `[3] column` **in** `([2] row` **in** `(table` **with** `(title` **same** `"Results")))`
- `(citations` **in** `([2..4] chapter` **in** `book))` **with** `"Knu*"`
- `(section` **with** `formula) – (section` **in** `appendix)`

Examples (2)

`[last] figure` **in** `(chapter` **with** `(section` **with** `(title` **with** `"early")))`



Software architecture



Efficiency: Worst case

- Full and lazy approaches
- Most operators are linear in the worst case

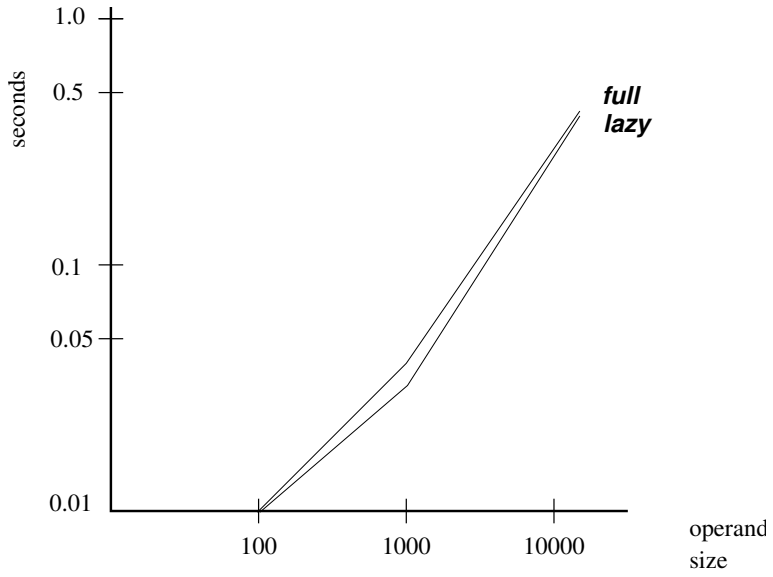
Operation	Full	Lazy
$+,-$	n	$n \min(d,h)$
is/same	n	n
in	$\min(n, d^2h)$	$\min(n, d^2h)$
beginin/endin	$\min(n, d^2h)$	$\min(n+dh, d^2h)$
$[s]$***in**	$n \min(d,h)$	$n \min(d,h)$
with*(k)	n	$n \min(n, k+dh)$
$[s]$**child**	n	n
parent(k)	n	$n \min(d,h)$
after/before	$n \min(n,dh)$	$n \min(n,dh)$

n = number of nodes, d = arity of trees, h = height of trees

Efficiency: Practice

- All operators are linear in practical situations
- Full evaluation is strongly linear (low variance)
- Lazy expands 40-100% of the operands
- Lazy takes 25-90% of the time of full evaluation (high variance)

A typical example:



IV.3 Indexing

From [Luk et al.(2002)Luk, Leong, Dillon, Chan, Croft, and Allan]:

- Flat File: add information, SQL accelerators, ...
- Semi-structured:
  - Field based: no overlapping, Hybrid model, ...
  - Segment based: Overlapped lists, List of references, p-strings
  - Tree based: Proximal Nodes, XRS, ...
- Structured:
  IR/DB, Path-based, Position-based, Multidimensional
- Indexes:
  - Structure + Value index:
    Toxin, Dataguides, T-indexes, Index Fabric, etc.
  - Full-text index:
    Proximal Nodes, Region Algebra, String Indexing, IXPN, ...

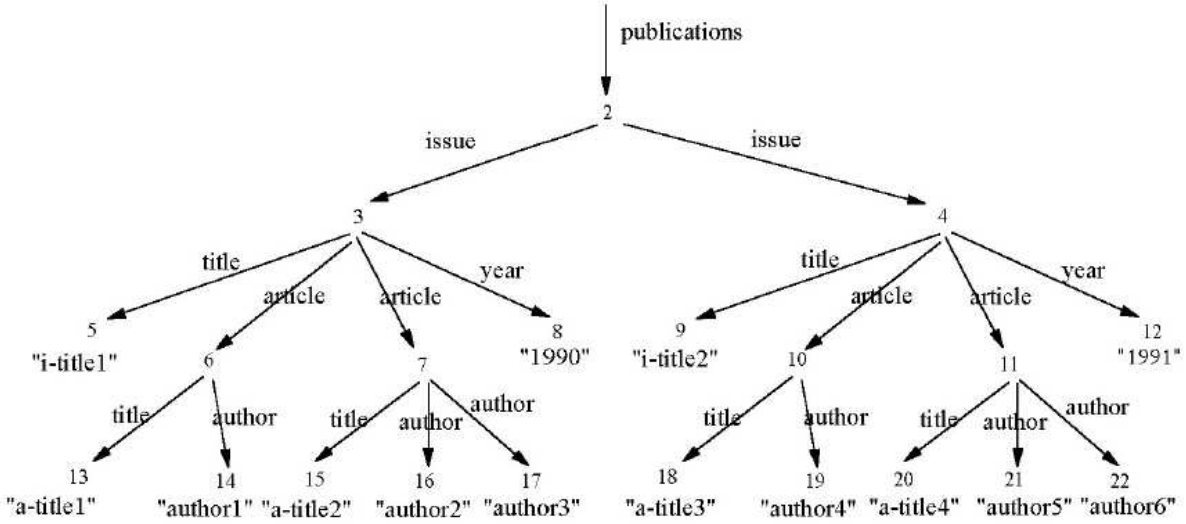
Searching

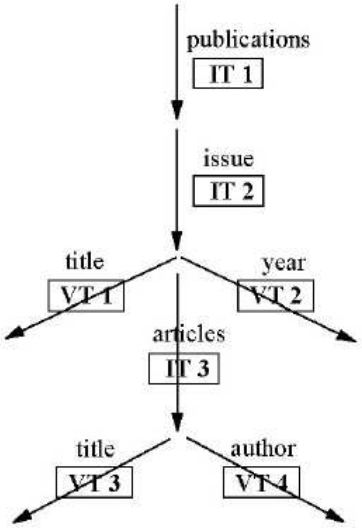
- Full text: sgrep, ...
- XML Assisted search: XML as a protocol
- Multi-stage search: XYZFind, ...‘
- Search results:
  - ranking
  - unit to be retrieved (complete, fragment, mixed)
  - layout
- Performance

IV.4 Examples: Toxin

From [Rizzolo and Mendelzon(2001)]:

- ToX query language (subset of XPath)
- memory/relational XML index
- backward/forward navigation
- Path index: path summaries
- Value indexes: support predicates
- Full-text: map string to XPath expression





VT 4

Node	Value
6	"author1"
7	"author2"
7	"author3"
10	"author4"
11	"author5"
11	"author6"

VT 1

Node	Value
3	"i-title1"
4	"i-title2"

VT 3

Node	Value
6	"a-title1"
7	"a-title2"
10	"a-title3"
11	"a-title4"

VT2

Node	Value
3	"1990"
4	"1991"

IT 3

Parent	Child
3	6
3	7
4	10
4	11

IT 1

Parent	Child
1	2

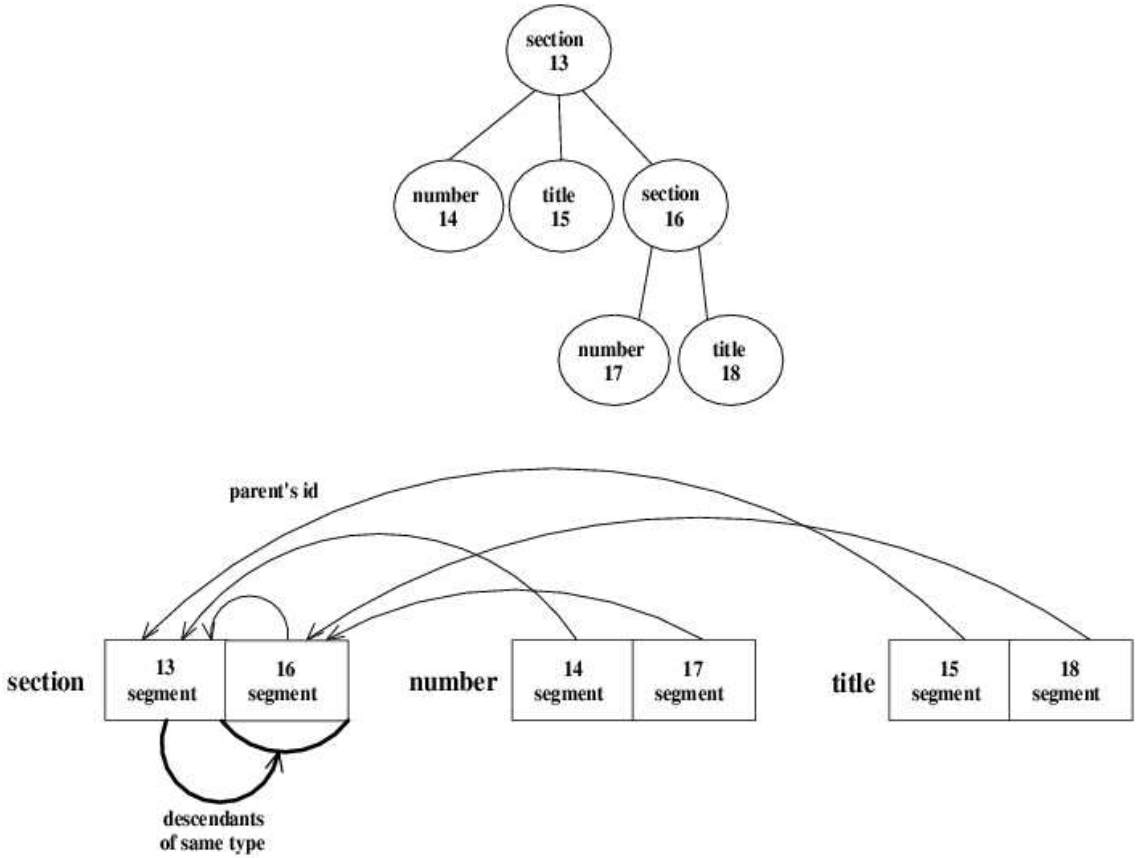
IT 2

Parent	Child
2	3
2	4

Examples: IXPN

From [Navarro and Ortega(2003)]:

- A fast implementation of XPath
- Maps XPath expression into Proximal Nodes
- Formal translation of Axes
- Node + Text index
- Lazy evaluation



Query	IXPN	Xind	eXist	Grep	Saxon	MS	ToXin
`/tstmt/bookcoll/book/chapter`	**1.8**	20.5	8.8	3.4	4.0	3.3	2.5
`/tstmt/coverpg/coverpg[title]`	**0.5**	2.8	2.2	0.7	3.3	1.3	—
`/tstmt//chapter`	**1.8**	58.9	8.8	3.8	4.1	3.2	2.5
`/tstmt[//chapter]`	**0.9**	22.7	8.8	3.7	4.0	4.2	—
`v[.=~"love"]`	**0.4**	9.9	9.8	0.7	3.4	1.8	3.7
`/tstmt/coverpg/title /following-sibling::subtitle`	**0.5**	2.6	9.8	0.7	3.3	1.3	—

IV.5 XML Streams

- Input: ten thousand XPath queries
- On-line: stream of XML data (say 10Mb/s)
- Problem: find which XML documents satisfy which queries
- Solutions for Linear Expressions: NFAs, DFAs, Lazy DFAs
- Solutions for Branching Expressions: Pushdown Automata
- Open problems:
  - constant time processing of branching expressions
  - sublinear sequential time per XML document
  - Optimization of lazy automata

IV.6 Open Problems

- Heterogeneous Data
- Ranking
- Evaluation
- New Retrieval Models
- More on indexing and searching
- Document Management
- Efficiency
- Most XQuery can be solved with XSLT: are they almost the same?

V. References

R. Baeza-Yates, D. Carmel, Y. Maarek, and A. Soffer, editors. *Special issue on XML*, volume 53(6), 2002a.

R. Baeza-Yates and G. Navarro. XQL and proximal nodes. *JASIST*, 53(6):504–514, 2002.

R.A. Baeza-Yates and G. Navarro. Integrating contents and structure in text retrieval. *ACM SIGMOD Record*, 25(1):67–79, March 1996.

Ricardo Baeza-Yates, Norbert Fuhr, and Yoelle S. Maarek, editors. *Proceedings of the SIGIR 2002 Workshop on XML and Information Retrieval*, 2002b.

D. Carmel, Y. Maarek, and A. Soffer, editors. *Proceedings of the SIGIR 2000 Workshop on XML and Information Retrieval*, 2000. `http://www.haifa.il.ibm.com/sigir00-xml/index.html`.

T.T. Chinenyanga and N. Kushmerik. Expressive retrieval from XML documents. In [Croft et al.(2001)Croft, Harper, Kraft, and Zobel], pages 163–171.

W.B. Croft, D. Harper, D.H. Kraft, and J. Zobel, editors. *Proceedings of the 24th Annual International Conference on Research and development in Information Retrieval*, New York, 2001. ACM.

Adelaida Delgado and Ricardo Baeza-Yates. An analysis of query languages for xml'. *Upgrade*, 3(3):12–25, 2002.

A. Doucet and H. Ahonen-Myka. Naïve clustering of a large XML document collection. In [Fuhr et al.(2003)Fuhr, Gövert, Kazai, and Lalmas], pages 81–87. `http://www.dcs.qmul.ac.uk/~gabs/papers/INEX2002_proceedings.pdf`.

N. Fuhr and K. Großjohann. XIRQL: A query language for information retrieval in XML documents. In [Croft et al.(2001)Croft, Harper, Kraft, and Zobel], pages 172–180.

Norbert Fuhr, Norbert Gövert, Gabriella Kazai, and Mounia Lalmas, editors. *First INitiative for the Evaluation of XML Retrieval (INEX) workshop. Dagstuhl, Germany, December 8–11, 2002*, ERCIM Proceedings, Sophia Antipolis, France, March 2003. `http://www.dcs.qmul.ac.uk/~gabs/papers/INEX2002_proceedings.pdf`.

Norbert Gövert, Norbert Fuhr, Mohammad Abolhassani, and Kai Großjohann. Content-oriented XML retrieval with HyREX. In [Fuhr et al.(2003)Fuhr, Gövert, Kazai, and Lalmas], pages 26–32. `http://www.dcs.qmul.ac.uk/~gabs/papers/INEX2002_proceedings.pdf`.

Norbert Gövert and Gabriella Kazai. Overview of the INitiative for the Evaluation of XML retrieval (INEX) 2002. In [Fuhr et al.(2003)Fuhr, Gövert, Kazai, and Lalmas], pages 1–17. `http://www.dcs.qmul.ac.uk/~gabs/papers/INEX2002_proceedings.pdf`.

T. Grabs and H.-J. Schek. Flexible information retrieval from XML with PowerDB-XML. In [Fuhr et al.(2003)Fuhr, Gövert, Kazai, and Lalmas], pages 141–148. http://www.dcs.qmul.ac.uk/~gabs/papers/INEX2002_proceedings.pdf.

R. Luk, H.V. Leong, T. Dillon, A. Chan, B. Croft, and J. Allan. A survey in indexing and searching xml documents. *JASIST*, 53(6):415–437, 2002.

Y. Mass, M. Mandelbrod, E. Amitay, Maarek Y., and A. Soffer. JuruXML - an XML retrieval system at INEX 02. In [Fuhr et al.(2003)Fuhr, Gövert, Kazai, and Lalmas], pages 73–90. http://www.dcs.qmul.ac.uk/~gabs/papers/INEX2002_proceedings.pdf.

Gonzalo Navarro and Manuel Ortega. IXPN: An index-based xpath implementation. *Submitted*, 2003.

P. Ogilvie and J. Callan. Language models and structure document retrieval. In [Fuhr et al.(2003)Fuhr, Gövert, Kazai, and Lalmas], pages 33–40. http://www.dcs.qmul.ac.uk/~gabs/papers/INEX2002_proceedings.pdf.

B. Piwowarski, G.-E. Faure, and P. Gallinari. Bayesian networks and INEX. In [Fuhr et al.(2003)Fuhr, Gövert, Kazai, and Lalmas], pages 149–154. http://www.dcs.qmul.ac.uk/~gabs/papers/INEX2002_proceedings.pdf.

Flavio Rizzolo and Alberto Mendelzon. Indexing XML data with ToXin. In *Fourth International Workshop on the Web and Databases*, 2001.