Chapter 4

XML Query Languages

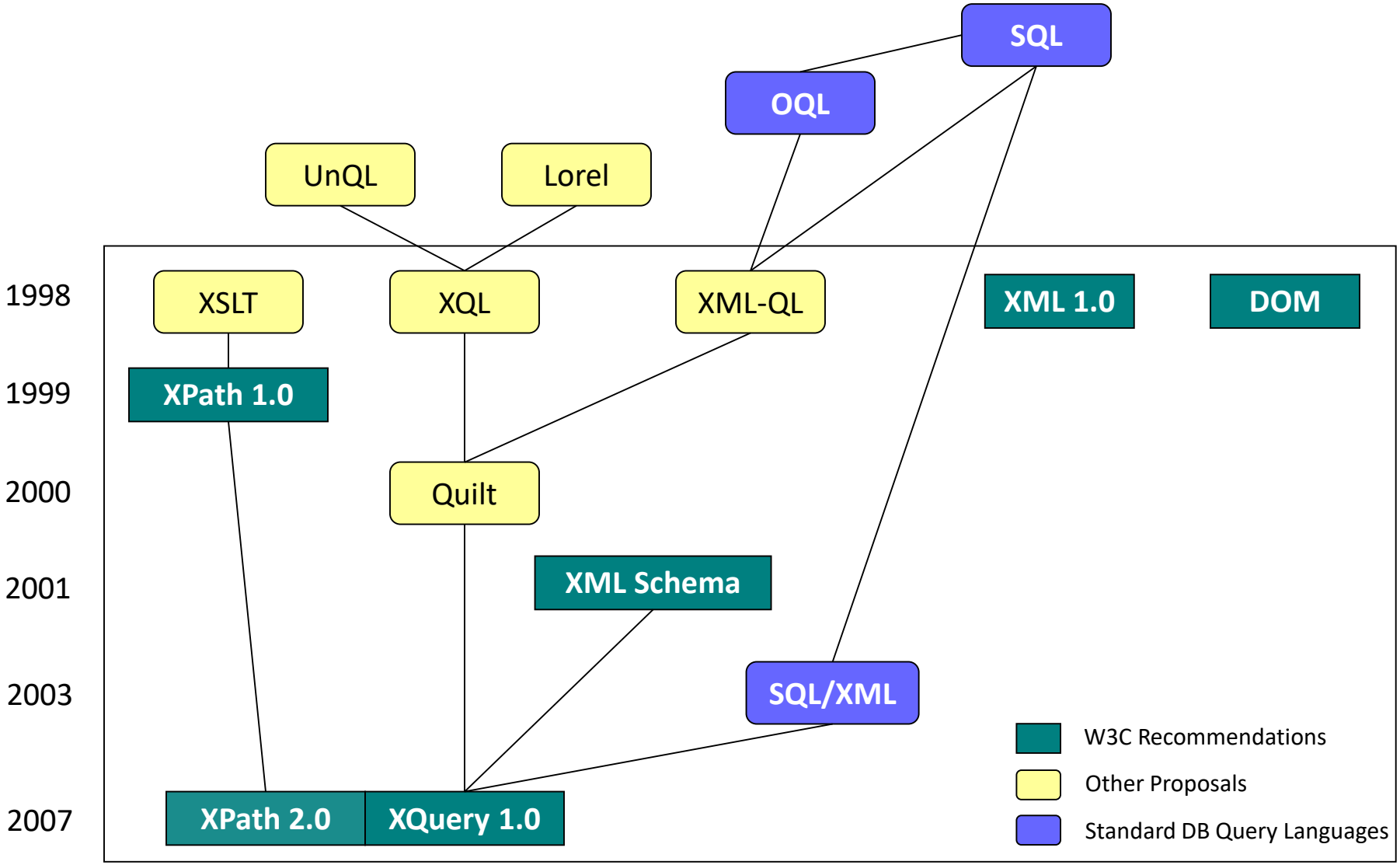
Foundations

XML Path Language (XPath) 2.0

XQuery 1.0: An XML Query Language

XIRQL

History


SQL
OQL
UnQL
Lorel
1998
XSLT
XQL
XML-QL
XML 1.0
DOM
1999
XPath 1.0
2000
Quilt
2001
XML Schema
2003
SQL/XML
2007
XPath 2.0
XQuery 1.0
W3C Recommendations
Other Proposals
Standard DB Query Languages

Basic Query Language Requirements

- **Ad-hoc:** Formulate queries without writing complete programs
- **Declarative:** Describe what is searched, not how the search should be computed
- **Generic:** Query language is built upon a few generic operations
- **Set-Oriented:** Operations work on set of objects
- **Adequate:** All constructs of the data model are exploited
- **Orthogonal:** All operations can be combined
- **Closed:** Query results can used as input for other queries
- **Complete:** All stored informations can be retrieved
- **Optimizable:** Queries can be optimized using equivalence rules
- **Efficient:** Operations can be implemented efficiently
- **Safe:** Queries always terminate and deliver a finite result
- **Formal Semantics:** All operations are formally defined

Relational Model and Algebra

- **Information represented by relations (tables) ≈ sets of tuples (rows)**
- **All attributes of tuples are atomic**
- **Algebra operations:**
  - **Selection:** select tuples (rows)
  - **Projection:** select attributes (columns)
  - **Set operations:** relation union, except (difference), intersect
  - **Join:** combine tuples / relations
  - **Rename:** rename attributes (columns)
- **All operations can be combined**
- **Relational algebra provides fundament for query optimization**

- **Hint: Combined algebra expressions are read from right to left!**

Relational Algebra Operations

R1

A1	A2
1	'Jim'
2	'Dad'
3	'Joe'

R2

A1	A2
2	'Dad'
3	'Bob'

R3

A1	A3
2	'F'
3	'T'
4	'F'

Join R1 ⋈ R3

A1	A2	A3
2	'Dad'	'F'
3	'Joe'	'T'

Selection σ[A1<3](R1)

A1	A2
1	'Jim'
2	'Dad'

Projection π[A1](R1)

A1
1
2
3

Rename β[A1,X1](R1)

X1	A2
1	'Jim'
2	'Dad'
3	'Joe'

Union R1 ∪ R2

A1	A2
1	'Jim'
2	'Dad'
3	'Joe'
3	'Bob'

Except R1 \ R2

A1	A2
1	'Jim'
3	'Joe'

Intersect R1 ∩ R2

A1	A2
2	'Dad'

Nested Relations

- **NF2 (Non First Normal Form) model supports atomic and relation-valued attributes**
- **Minimal extension of relational algebra includes operations for relation-valued attributes**
  - Access to inner structures of nested relations via recursive nesting of selections and projections within selection predicates and projection lists
  - Selection predicates can include relational conditions
    - Set comparisons ($\subset$, $\subseteq$, $=$, $\neq$, $\supseteq$, $\supset$)
    - Set inclusion ($\in$, $\notin$)
  - Nesting / Unnesting

Nesting

R	A1	A2	A3
	2	'Kim'	'T'
	2	'Dad'	'F'
	3	'Joe'	'T'

ν[(A2, A3); A23] (R) $\Longrightarrow$

$\Longleftarrow$ μ[A23] (R')

Unnesting

R'	A1	{A23}	
		A2	A3
	2	'Kim'	'T'
		'Dad'	'F'
	3	'Joe'	'T'

Nesting Selections and Projections

$\pi[\pi[A2](A23)](R)$

{R'}
{A23}
A2
'Kim'
'Dad'
'Joe'

$\pi[\sigma[A3='T'](A23)](R)$

{R'}	
{A23}	
A2	A3
'Kim'	'T'
'Joe'	'T'

R	A1	{A23}	
		A2	A3
	2	'Kim'	'T'
		'Dad'	'F'
	3	'Joe'	'T'

$\sigma[\emptyset \subset \pi[A2](A23)](R)$

{R'}		
A1	{A23}	
	A2	A3
2	'Kim'	'T'
	'Dad'	'F'
3	'Joe'	'T'

$\sigma[\emptyset \subset \sigma[A3='F'](A23)](R)$

{R'}		
A1	{A23}	
	A2	A3
2	'Kim'	'T'
	'Dad'	'F'

Extended Relational Models (1)

- **Support for further attribute types: tuple type, collection types, reference type**
- **Operations on tuples**
  - Tuple field access
  - Navigation: Access fields of nested tuples
- **Operations on collections**
  - Element containment and subset associations
  - Unnesting

R

A1	<A2>	
	A21	{A22}
1	1970	{41, 16}
2	1972	{12, 1, 78}
3	1969	{13, 11, 69}

σ[A2.A21>1971](R)

A1	<A2>	
	A21	{A22}
2	1972	{12, 1, 78}

σ[12∈A2.A22](R)

A1	<A2>	
	A21	{A22}
2	1972	{12, 1, 78}

Extended Relational Models (2)

- **Operations on references**
  - Dereferencing: Access the referenced object
  - Navigation: Access attributes of referenced objects using path expressions

R

TID	Name	Boss
@911	'Jim'	@655
@655	'Joe'	@324
@876	'Bob'	@655
@324	'Kim'	@324

π[Name, DEREF(Boss)](R)

Name	<DEREF(Boss)>	
	Name	Boss
'Jim'	'Joe'	@324
'Joe'	'Kim'	@324
'Bob'	'Joe'	@324
'Kim'	'Kim'	@324

π[Name, Boss→Name](R)

Name	Boss→Name
'Jim'	'Joe'
'Joe'	'Kim'
'Bob'	'Joe'
'Kim'	'Kim'

  - Hint: The arrow symbol is a shortcut for a dereferencing (**DEREF**) followed by a tuple field access (**.**). Sometimes, the dot symbol is used instead of the arrow symbol. This however might cause confusion since the dereferencing is then done implicitly.

What can be done so far?

- **Selection on sets of tuples/objects**
- **Projection on certain attributes**
- **Join sets of tuples/objects**
- **Classic set operations on tuples/objects**
- **Navigation in nested structures: path expressions on embedded tuples/objects**
- **Navigation in network structures: path expressions using references**
- **...**

What is missing for XML Query Languages?

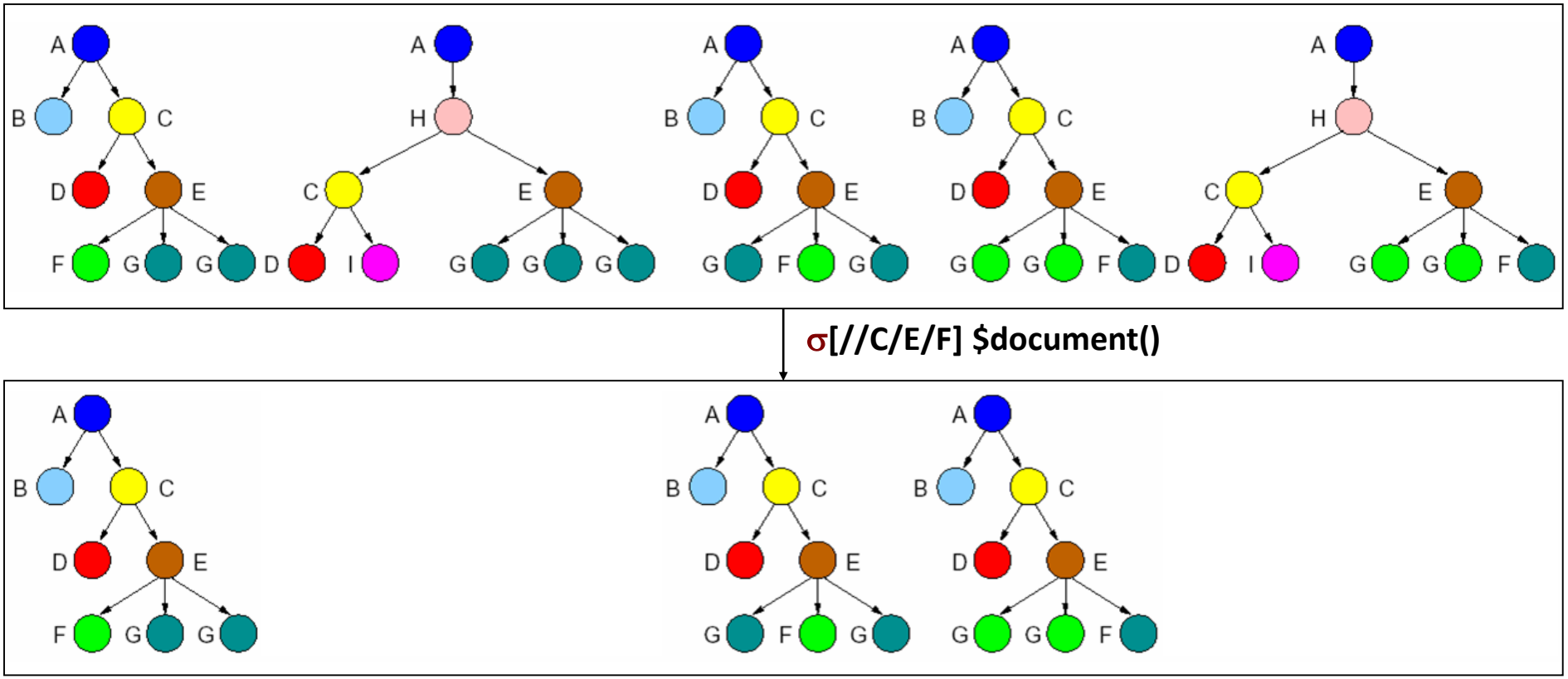
Additional Requirements for XML Query Languages

- **Schema Awareness**
  - Queries must be possible whether or not a schema is available; Support “wildcards“ in path expressions
- **Flexible Types**
  - Processing elements of different types
- **Seamless XML Embedding**
  - Queries embedded with XML and XML embedded in Queries
- **Ordering**
  - Element order must be retained

Basic Operations of XML Query Languages

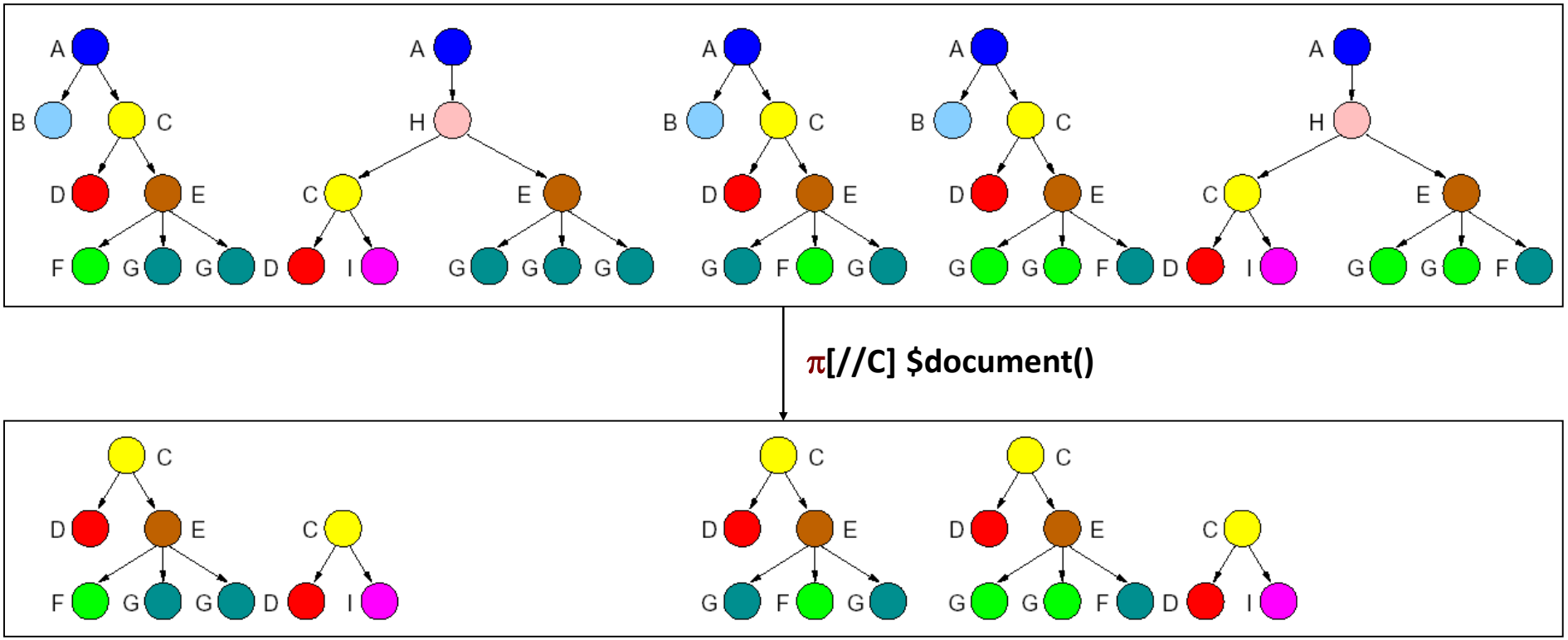
- **Selection**
  - select documents or elements based on content and structure as well
- **Extraction and Reduction**
  - extract and delete sub elements
- **Combination and Restructuring**
  - compose two or more elements combined to a new element, create new element sequences

Selection

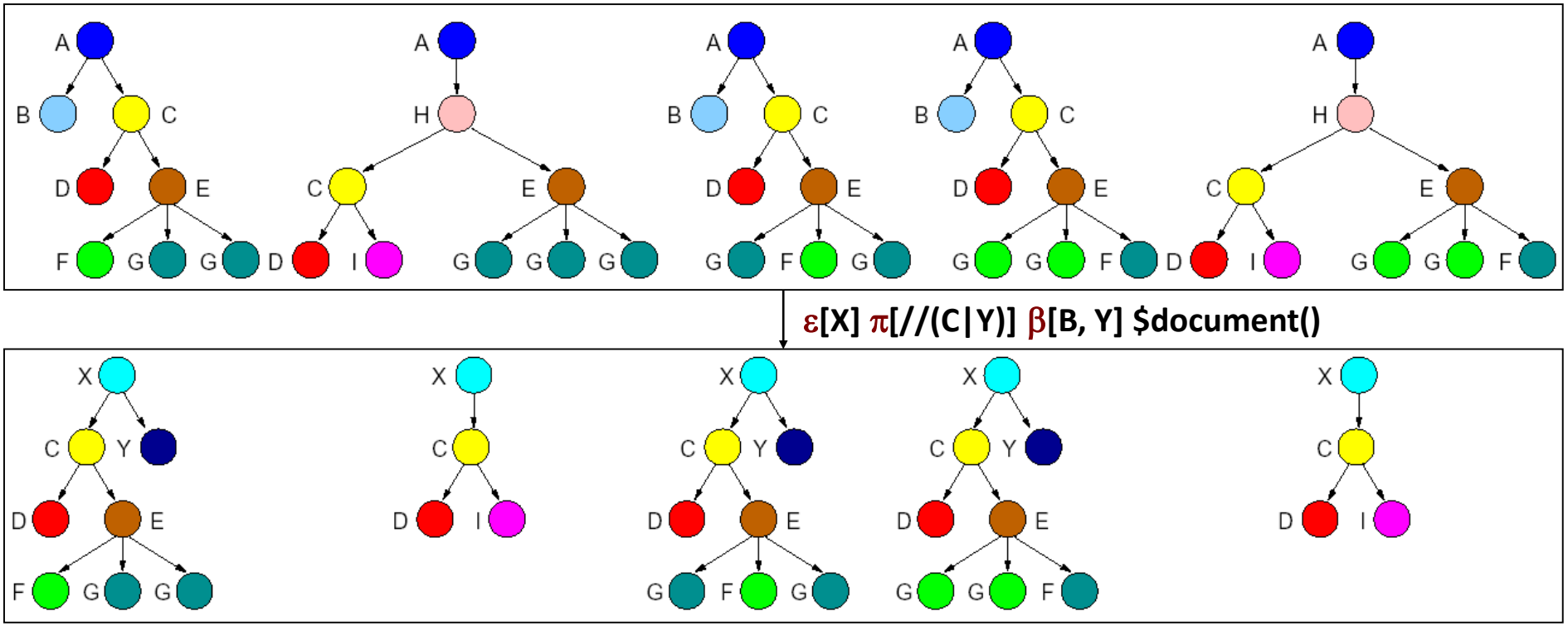


selection predicate based on the document structure

Extraction and Reduction



Combination and Restructuring



Element constructor ε[X] creates a new element

«B elements are renamed to Y, then C and Y elements are placed as sub elements of a new element X»

XML Path Language (XPath) 2.0

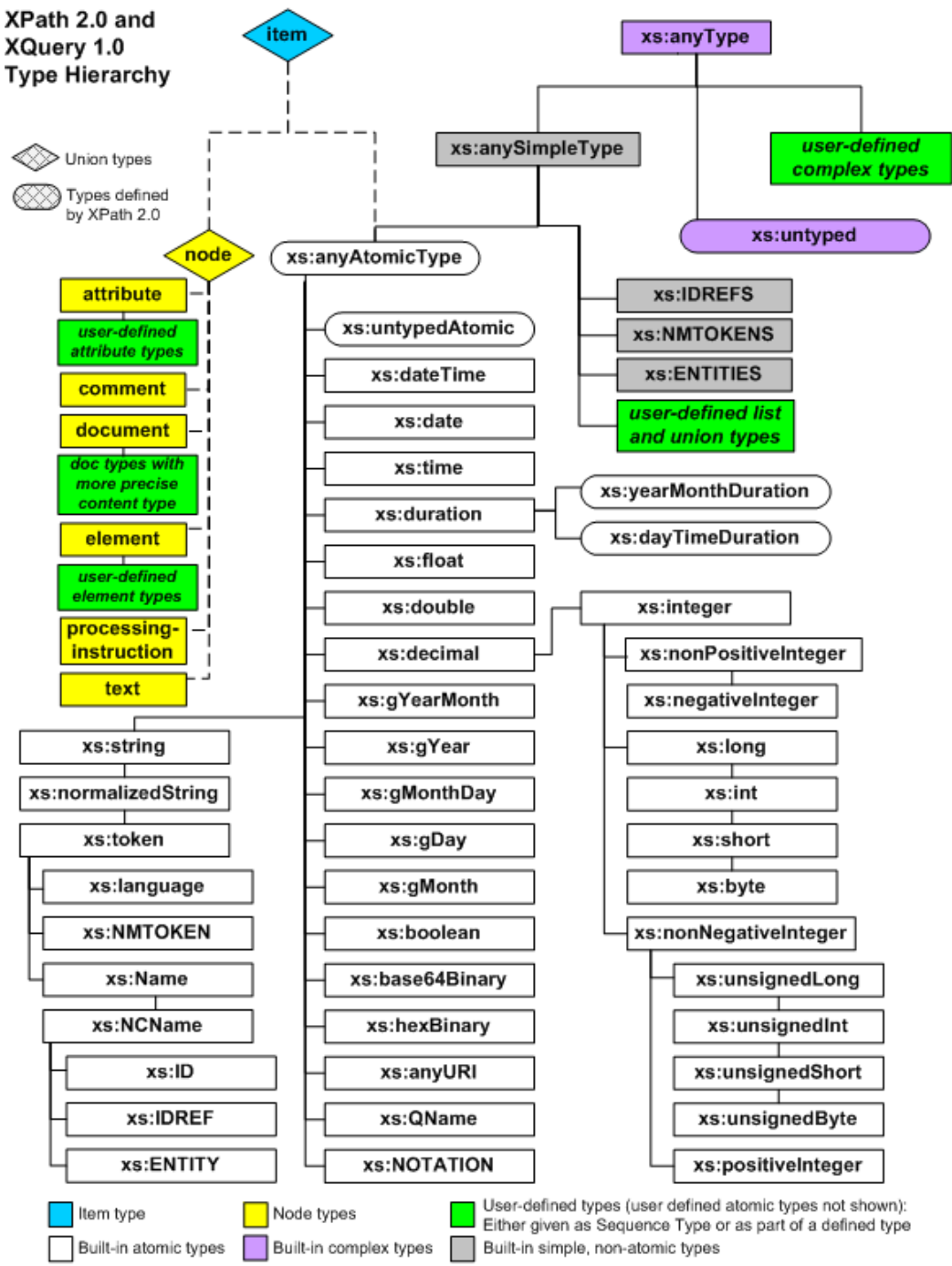
- **W3C Recommendation 23 January 2007**
- **XPath defines pattern, functions, and expressions to select XML elements and attributes**
  - Addressing node sets
  - Formulating conditions on these node sets
- **Basic construct: XPath expressions**
  - Expressions are of type boolean, number, string, or node-set (unordered collection of nodes)
  - Path expressions
  - Logical and mathematical operators
  - Function calls
- **XPath is part of several standards**
  - XSL, XLink/XPointer, XQuery

Data Model of XPath/XQuery

- **Bases on XML Information Set**
  - XML 1.0, Namespaces, XML Schema
- **Data types**
  - Simple and complex types known from XML Schema
  - XML 1.0 Characters
  - XPath node types (document, element, attribute, text, namespace, comment, processing instruction)
- **With and without schema**
- **XPath/XQuery expressions return a sequence of items**
  - Ordered collection of null or more *items*
  - An *item* is a *atomic value or a node*
  - Atomic value (of a XML Schema type): `string` | `boolean` | `decimal` | `ID` | `IDREF` …
  - Node: `document` | `element` | `attribute` | `text` | `namespaces` | `comment` | `PI`
  - Values and nodes can be typed or untyped

Built-in Data Types

Predefined Namespaces - Prefixes	
xml:	http://www.w3.org/XML/1998/namespace
xs:	http://www.w3.org/2001/XMLSchema
xsi:	http://www.w3.org/2001/XMLSchema-instance
fn:	http://www.w3.org/2003/11/xpath-functions
xdt:	http://www.w3.org/2003/11/xpath-datatypes
local:	http://www.w3.org/2005/xquery-local-functions



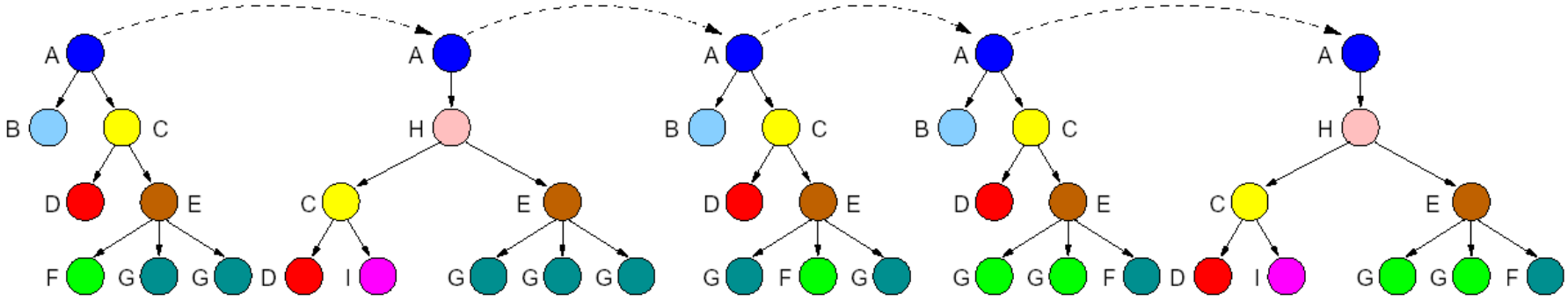
* Figure taken from

XQuery 1.0 and XPath 2.0 Data Model (XDM)
W3C Recommendation 23 January 2007

http://www.w3.org/TR/xpath-datamodel/

Main Data Type: Sequence

- **A sequence of one item equals the same item**
  - (1) ≡ 1
- **Sequences are implicitly unnested**
  - (1, (2, 3)) ≡ (1, 2, 3)
- **Sequences can be heterogeneous**
  - (<a/>, 3)
- **Sequences can contain duplicates**
  - (2, 2, 2)



Path Expressions

- **Address nodes of an XML tree**
- **Designed to be embedded in a host language**
- **Have the form:** `/Step/Step/…/Step`
  - can consist of several expressions (steps) that are connected via the slash (/) symbol
- **Stepwise processing from left to right**

```
//book[title='XML and Databases']/author
```

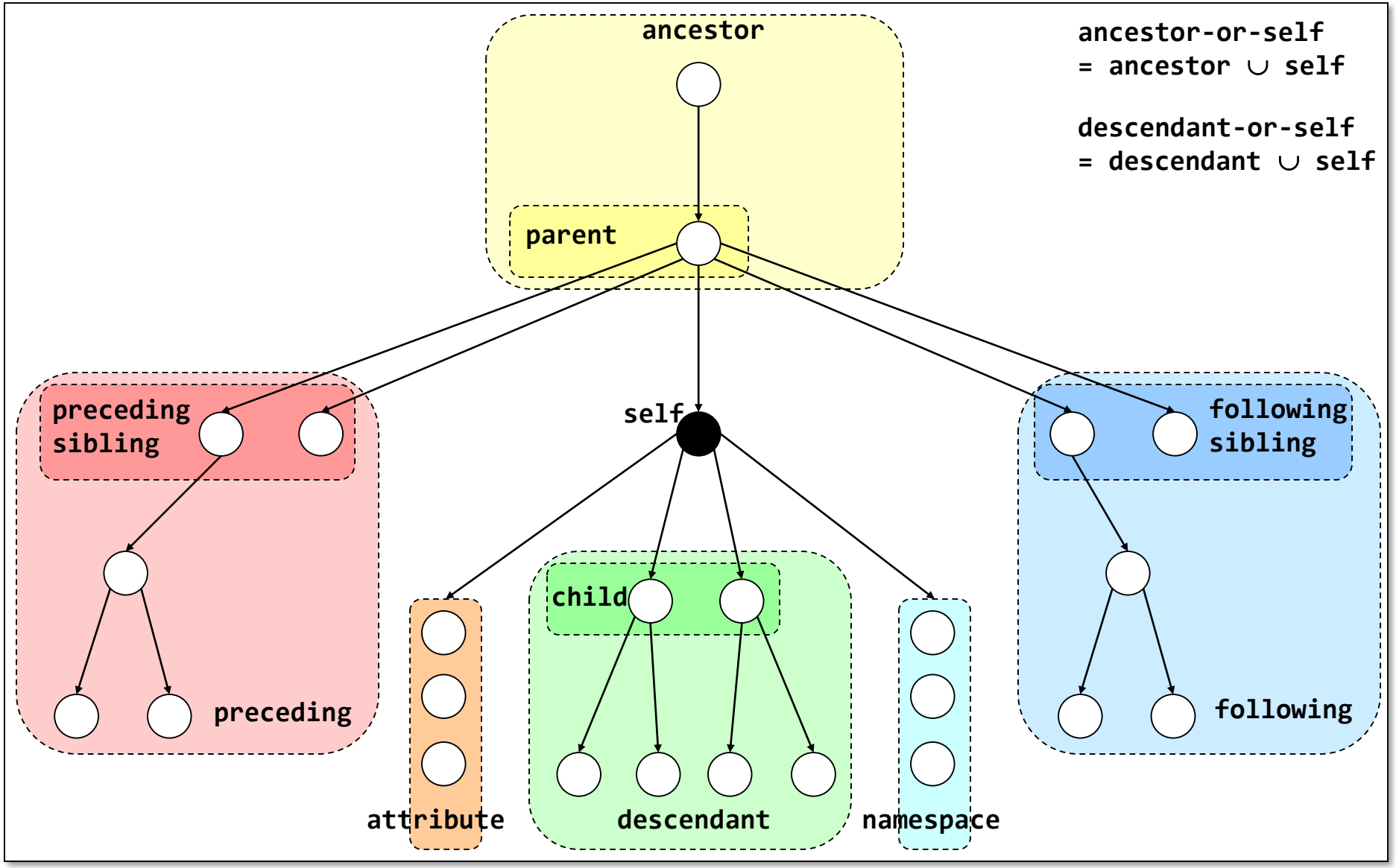
yields all author elements of book elements whose title element has the content 'XML and Databases'

- **Absolute versus relative path expressions**
- **Note: Path expressions define extractions and also selections by using filter predicates**

Steps

- **Input and output of a step is a sequence**
- **Step expression:** `NavigationAxis::NodeTest[Predicate]`
  - **NavigationAxis** defines relationship between context node and nodes to be selected
  - **NodeTest** includes type and name of nodes to be selected
  - **Predicate** provides filter for nodes to be selected
- **Steps processed within a context**
- **Processing context includes**
  - context node (`self`)
  - context position and size
  - set of namespace declarations

Navigation Axis



Node Test

- **Restricts types and names of the nodes to be selected**

Node Test	Restriction to
`*`	no restriction ("wildcard")
`name`	all sub elements with the given name
`document-node()`	the document node
`node()`	all sub element nodes
`text()`	all text element nodes
`processing-instruction()`	all processing instruction nodes
`comment()`	all comment nodes

Predicate

- **Filter expression on a node sequence**
- **Combined predicates**
  - Example: `//book[2][author/last-name='Melville']`
  - Filter evaluation from left to right
  - NOT commutative: a[b][2] != a[2][b]
- **Conjunctive predicates**
  - Example: `//book[author/last-name='Melville' and price<15]`
- **If a predicate does not yield a Boolean value, it is implicitly converted to a Boolean value**
  - Numeric values converted to position predicates
    - Example: `//book[2]` ≡ `//book[position()=2]`
  - In all other cases the result of the conversion is false

Syntax: Shortcuts

Shortcut	Full Expression	yields
`tagname`	`child::tagname`	all child nodes referring to tagname elements
`.`	`self::node()`	the current context node
`..`	`parent::node()`	the parent node
`*`	`descendant-or-self::`	all descendant nodes
`@name`	`attribute::name`	attribute 'name' of the current node
`/`		the root node
`//`		all descendant nodes of the root
`[expr]`		elements of node sequence that satisfy the expression
`[n]`		n-th element of a node sequence

XPath Functions

Function	yields
number **last()**	position of the last element
number **position()**	context position
number **sum(*node-set*)**	sum of the node values
number **count(*node-set*)**	count of the nodes
string **name(*node-set?*)**	name of the node set
node-set **id(*object*)**	nodes with this id
boolean **contains(*string*, *string*)**	true if second argument is contained in the first argument
boolean **not(*boolean*)**	negation
…	…

- **Further functions:**
  - number, floor, ceiling, round
  - string, concat, starts-with, ends-with, substring-before, substring-after, substring, string-length, normalize-space, translate
  - base-uri, document-uri, namespace-uri, node-name
- **Document access with fn:doc(*uri*)**

XPath-Query Examples

bs.xml

```
<?xml version="1.0"
  encoding="UTF-8"?>
<bookstore>
 <book genre="autobiography">
  <title>The Autobiography of
    Benjamin Franklin
  </title>
  <author>
   <first-name>Benjamin</first-name>
   <last-name>Franklin</last-name>
  </author>
  <price currency="USD">8.99</price>
 </book>
 <book genre="novel">
  <title>The Confidence Man</title>
  <author>
   <first-name>Herman</first-name>
   <last-name>Melville</last-name>
  </author>
  <price currency="USD">11.99</price>
 </book>
 <book genre="philosophy">
  <title>The Gorgias</title>
  <author>
   <name>Plato</name>
  </author>
  <price currency="USD">9.99</price>
 </book>
</bookstore>
```

fn:doc("bs.xml")/bookstore/book/@genre

```
genre="autobiography«
genre="novel"
genre="philosophy"
```

fn:doc("bs.xml")/bookstore/book[author/name='Plato']

```
<book genre="philosophy">
 <title>The Gorgias</title>
 <author><name>Plato</name></author>
 <price currency="USD">9.99</price>
</book>
```

fn:doc("bs.xml")//author[first-name='Herman']/last-name

```
<last-name>Melville</last-name>
```

fn:doc("bs.xml")//book[author/last-name='Franklin']/price

```
<price currency="USD">8.99</price>
```

fn:doc("bs.xml")//book[contains(title, 'an')]/title

```
<title>The Autobiography of Benjamin Franklin</title>
<title>The Confidence Man</title>
```

fn:doc("bs.xml")//book[.//name='Plato' and price < 20]/title

```
<title>The Gorgias</title>
```

XPath-Query Examples (2)

bs.xml

```
<?xml version="1.0"
  encoding="UTF-8"?>
<bookstore>
 <book genre="autobiography">
  <title>The Autobiography of
    Benjamin Franklin
  </title>
  <author>
   <first-name>Benjamin</first-name>
   <last-name>Franklin</last-name>
  </author>
  <price currency="USD">8.99</price>
 </book>
 <book genre="novel">
  <title>The Confidence Man</title>
  <author>
   <first-name>Herman</first-name>
   <last-name>Melville</last-name>
  </author>
  <price currency="USD">11.99</price>
 </book>
 <book genre="philosophy">
  <title>The Gorgias</title>
  <author>
   <name>Plato</name>
  </author>
  <price currency="USD">9.99</price>
 </book>
</bookstore>
```

fn:doc("bs.xml")//book/title[1]

```
<title>The Autobiography of Benjamin Franklin</title>
<title>The Confidence Man</title>
<title>The Gorgias</title>
```

(fn:doc("bs.xml")//book/title)[1]

```
<title>The Autobiography of Benjamin Franklin</title>
```

(fn:doc("bs.xml")//book/title[1])[2]

```
<title>The Confidence Man</title>
```

fn:doc("bs.xml")//book/title[1][2]

fn:doc("bs.xml")//book[price>9]/title

```
<title>The Confidence Man</title>
<title>The Gorgias</title>
```

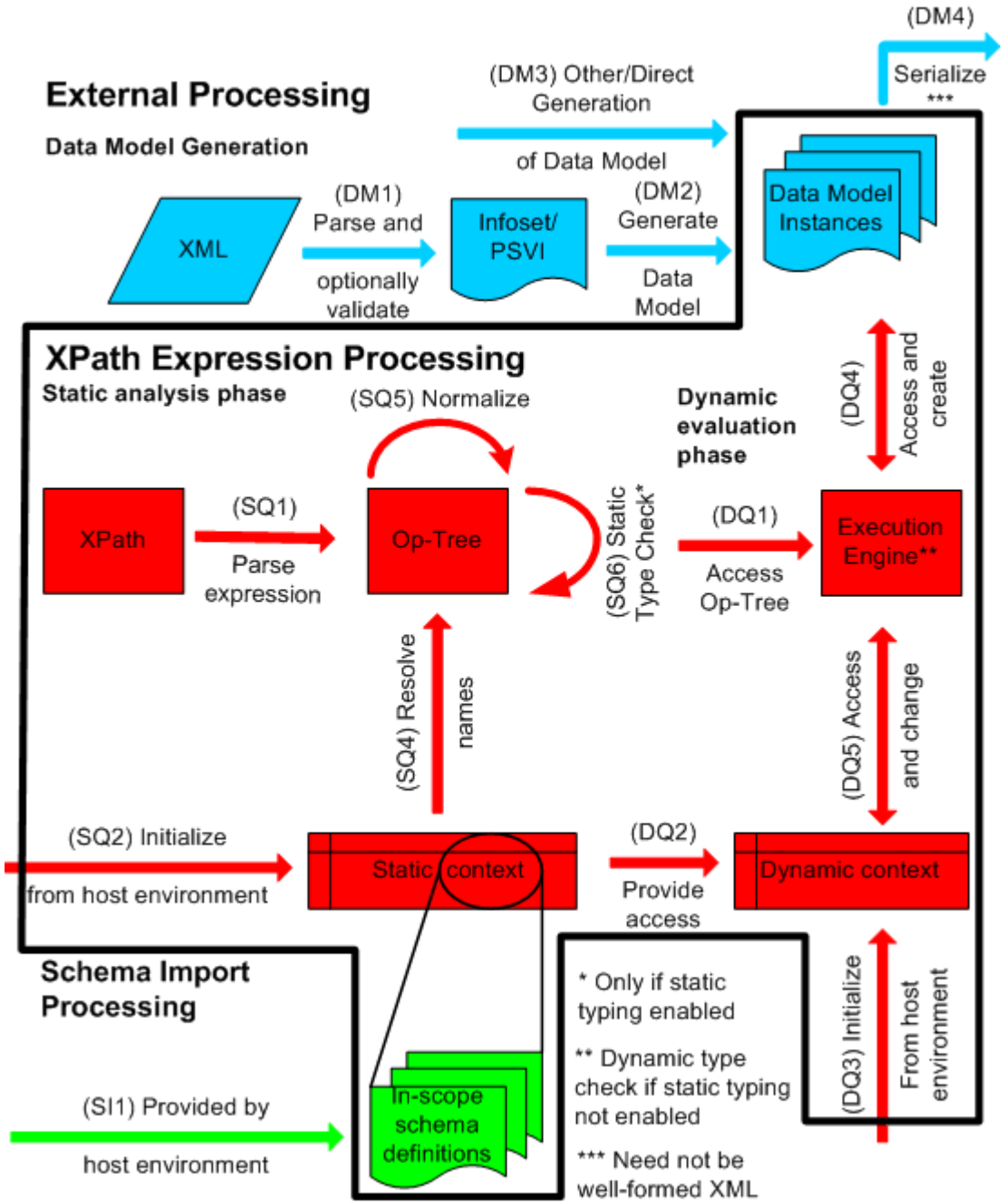
fn:doc("bs.xml")//book[price>9][2]/title

```
<title>The Gorgias</title>
```

fn:doc("bs.xml")//book[2][price>9]/title

```
<title>The Confidence Man</title>
```

XPath Processing Model



* figure taken from

XML Path Language (XPath) 2.0
W3C Recommendation 23 January 2007

http://www.w3.org/TR/xpath20/

Conclusions: XPath

- **Tree-based data model**
- **Queries formulated as path expressions**
- **Well-defined semantics**
- **XPath supports**
  - Extraction and reduction (described by steps of path expressions)
  - Selection (described by filter predicates in steps)
  - Aggregate functions (count, sum)
  - Navigation functions
  - Wildcards
  - Order preservation
- **No support for combination and restructuring!**

XQuery 1.0: An XML Query Language

- **W3C Recommendation 23 January 2007**
- **Based on the XPath/XQuery data model**
- **Strongly typed based on XML Schema**
- **Similar to SQL/OQL**
- **Functional language but also includes imperative constructs**
- **Supports composite expressions and orthogonal usage of different expression types**

→ **XQuery is more than a declarative query language**

→ **Programming language for arbitrary XML transformations**

XQuery Basics

- **Embedding XML in XQuery expressions and vice versa**
- **Element constructors and computed XML elements**
- **Path expressions (XPath 2.0) for selection of node sequences**
- **Data type specific operators**
- **FLWOR expressions allow queries similar to SFW clauses in SQL**
  - **for/let**: ordered list of tuples of bound variables
  - **where**: restricted list of tuples of bound variables
  - **order**: sorted list of tuples of bound variables
  - **return**: result construction which is an instance of the XQuery data model
- **Conditional statements**
- **Quantified expressions using the ALL and SOME quantifiers**
- **Data type testing and conversion**
- **Function calls**

XQuery Prolog

- **Every query in XQuery consists of an expression and an optional prolog which defines the context for the expression evaluation**
- **Prolog can contain different types of declarations:**
  - XQuery version

```
xquery version "1.0" encoding "utf-8";
```

  - Global and external variables

```
define variable $x external;
define variable $copyright as xs:string := "Copyright 2003-2007";
```

  - Document order

```
declare ordering ordered;
declare ordering unordered;
```

  - Functions

```
declare function local:depth($e as node()) as xs:integer {
  if (fn:empty($e/*)) then 1 else fn:max(for $c in $e/*
  return local:depth($c)) + 1
};
```

  - Namespaces
  - Import of schemata and function libraries
  - …

XQuery Expressions

Constructors	`pcdata(expr), processing-instruction(expr, expr), comment(expr), etc.`
Navigation methods	`children(expr), parent(expr), attributes(expr), name(expr), etc.`
Arithemetic functions	`+ | - | * | mod | div`
Comparison functions	`= | != | < | <= | > | >=`
Aggregate functions	`agg(expr)` *with* `agg ∈ {`**count, min, max, sum, avg**`}`
Set functions	**union** | **except** | **intersect**
Iterator	**for** *variable* **in** *expr* **return** *expr*
Conditions	**if** (expr) **then** *expr* **else** *expr*
Local variable binding	**let** *variable* **:=** *expr*
Sorting	*expr* **order by** (*expr*)
Document access	`fn:doc(`*uri*`)` or `fn:collection(`*uri*`)`

XQuery FLWOR Expressions

```
flwor-expr ::= (for-expr | let-expr)+
               (where expr)?
               (order by expr)?
               return expr

for-expr ::=    (for $var in expr (, $var in expr)*)+

let-expr ::=    (let $var := expr (, $var := expr)*)+
```

```
for $v1 in e1, $v2 in e2, …, $vn in en
where SelectionPredicate
order by OrderExpression
return ProjectionList
```

≡

```
SELECT ProjectionList
FROM e1 $v1, e2 $v2, …, en $vn
WHERE SelectionPredicate
ORDER BY OrderExpression
```

```
for $v1 in e1 for $v2 in $v1
where SelectionPredicate
order by OrderExpression
return ProjectionList
```

≡

```
SELECT ProjectionList
FROM e1 $v1, UNNEST($e2) $v2
WHERE SelectionPredicate
ORDER BY OrderExpression
```

XQuery Variables

- **Binding in for and let expressions**
- **Type derived from the binding**
- **Values fixed with binding**
- **Binding visible only within the current and all included query expressions**
- **Binding released with finishing the expression evaluation**
- **In case of several bindings, the last one is visible**

Atomization

- **The fn:data function accepts a sequence of items and returns their typed values**
  - For atomic values: return the value itself
  - For nodes: extract the typed value of the node
- **Calling fn:data is often unnecessary because the typed value of a node is automatically extracted (atomized) for many XQuery/XPath expressions, including comparisons, arithmetic operations, function calls**

```
<result>
<f1>{fn:data(fn:doc("bookstore.xml")//book)}</f1>
<f2>{fn:data(fn:doc("bookstore.xml")//@genre)}</f2>
<f3>{fn:data(fn:doc("bookstore.xml")//book[1]/title)}</f3>
<f4>{fn:data(fn:doc("bookstore.xml")//book[1]/title/text())}</f4>
</result>

yields

<result>
  <f1>The Autobiography of Benjamin FranklinBenjaminFranklin8.99 The
Confidence ManHermanMelville11.99 The GorgiasPlatoPlatoPlato9.99</f1>
  <f2>autobiography novel philosophy</f2>
  <f3>The Autobiography of Benjamin Franklin</f3>
  <f4>The Autobiography of Benjamin Franklin</f4>
</result>
```

Example Data and Schema

```
type Bib = element bib (Book*)
type Book = element book
    (attribute year (xs:integer) &
     attribute isbn (xs:string),
     element title (xs:string),
     (element author(xs:string))+)

let $bib0 :=
<bib>
 <book year="1999" isbn="1-55860-622-X">
  <title>Data on the Web</title>
  <author>Abiteboul</author>
  <author>Buneman</author>
  <author>Suciu</author>
 </book>
 <book year="2001" isbn="1-XXXXX-YYY-Z">
  <title>XML Query</title>
  <author>Fernandez</author>
  <author>Suciu</author>
 </book>
</bib> as Bib
return $bib0
```

```
let $book0 :=
<book year="1999" isbn="1-55860-622-X">
 <title>Data on the Web</title>
 <author>Abiteboul</author>
 <author>Buneman</author>
 <author>Suciu</author>
</book> as Book
return $book0
```

Element and Attribute Constructor

- **Element construction using the XML notation**

```
<lecture>XML and Databases</lecture>
```

- **XQuery expressions are wrapped by curly brackets { }**
- **The curly brackets { and } are masked by doubling**

```
<references lecture="XML and Databases">
{
  for $x in fn:$bib0//book
  return <book title={$x/title/text()}>{string($x/@ISBN)}</book>
}
</references>

yields

<references lecture="XML and Databases" >
 <book title="Data on the Web">1-55860-622-X</book>
 <book title="XML Query">1-XXXXX-YYY-Z</book>
</references>
```

Extraction and Reduction

```
(: projection on elements:)

$bib0/bib/book/author


yields

<author>Abiteboul</author>
<author>Buneman</author>
<author>Suciu</author>
<author>Fernandez</author>
<author>Suciu</author>
```

```
(: projection on element content :)

for $a in $bib0/bib/book/author
return <a>{fn:data($a)}</a>

yields

<a>Abiteboul</a>
<a>Buneman</a>
<a>Suciu</a>
<a>Fernandez</a>
<a>Suciu</a>
```

```
(: projection on attribute :)

<y>{$book0/book/@year}</y>

yields

<y year="1999"/>
```

```
(: projection on attribute values :)

<y>{fn:data($book0/book/@year)}</y>

yields

<y>1999</y>
```

Iteration

```
(: iteration over elements :)

for $b in $bib0/bib/book
return <book>{$b/author, $b/title}</book>

yields

<book>
 <author>Abiteboul</author>
 <author>Buneman</author>
 <author>Suciu</author>
 <title>Data on the Web</title>
</book>
<book>
 <author>Fernandez</author>
 <author>Suciu</author>
 <title>XML Query</title>
</book>
```

Element construction using pure XML or composite expressions (as here!)

```
for $b in $bib0/bib/book
return $b/author

is equivalent to

$bib0/bib/book/author
```

Selection

```
(: selection of elements :)

for $b in $bib0/bib/book
where $b/@year <= 2000
return $b

yields

<book year="1999" isbn="1-55860-622-X">
 <title>Data on the Web</title>
 <author>Abiteboul</author>
 <author>Buneman</author>
 <author>Suciu</author>
</book>
```

Predicate can be a complex one consisting of several parts

```
for $b in $bib0/bib/book
where true
return $b/author

is equivalent to

$bib0/bib/book/author
```

Quantification

```
(: using existence quantifier :)

for $b in $bib0/bib/book
where some $a in $b/author
      satisfies $a = "Buneman"
return $b

yields

<book year="1999" isbn="1-55860-622-X">
 <title>Data on the Web</title>
 <author>Abiteboul</author>
 <author>Buneman</author>
 <author>Suciu</author>
</book>
```

```
(: using all quantifier :)

for $b in $bib0/bib/book
where every $a in $b/author
      satisfies $a = "Buneman"
return $b

yields

()
```

Combination and Restructuring

```
type Reviews = element reviews
   ((element book
      (element title (xs:string),
       element review
    (xs:string)))*)

let $review0 :=
<reviews>
 <book>
  <title>Data on the Web</title>
  <review>A darn fine book.</review>
  </book>
  <book>
   <title>XML Query</title>
   <review>This is great!</review>
  </book>
</reviews>
as Reviews
return $review0
```

```
for $b in $bib0/bib/book,
    $r in $review0/reviews/book
where $b/title = $r/title
return <book>
        {$b/title, $b/author, $r/review}
       </book>
```

Join condition

yields

```
<book>
 <title>Data on the Web</title>
 <author>Abiteboul</author>
 <author>Buneman</author>
 <author>Suciu</author>
 <review>A darn fine book.</review>
</book>
<book>
 <title>XML Query</title>
 <author>Fernandez</author>
 <author>Suciu</author>
 <review>This is great!</review>
</book>
```

Sorting

```
for $b in $review0//book
order by $b/title ascending
return $b

yields

<book>
 <title>Data on the Web</title>
 <review>A darn fine book.</review>
</book>
<book>
 <title>XML Query</title>
 <review>This is great!</review>
</book>
```

alternative: descending

Grouping and Aggregate Functions

```
for    $a in distinct-values($bib0//author)
let    $b := $bib0//book[author=$a]
return <group>
         {$a}
         {<count>{count($b)}</count>}
       </group>

yields

<group>
 <author>Abiteboul</author><count>1</count>
</group>
<group>
 <author>Buneman</author><count>1</count>
</group>
<group>
 <author>Suciu</author><count>2</count>
</group>
<group>
 <author>Fernandez</author><count>1</count>
</group>
```

Group by composite for/let expressions

Aggregate functions can be used also in for clauses

Parent Operator

```
for $b in $bib0/bib/book
where $b/@year = 2001
return $b/..

yields

<bib>
 <book year="1999" isbn="1-55860-622-X">
  <title>Data on the Web</title>
  <author>Abiteboul</author>
  <author>Buneman</author>
  <author>Suciu</author>
 </book>
 <book year="2001" isbn="1-XXXXX-YYY-Z">
  <title>XML Query</title>
  <author>Fernandez</author>
  <author>Suciu</author>
 </book>
</bib>
```

Parent

Type Conversion: Treat and Cast

```
for $p in $book0/book/
return $p treat as Book

yields

<book year="1999" isbn="1-55860-622-X">
 <title>Data on the Web</title>
 <author>Abiteboul</author>
 <author>Buneman</author>
 <author>Suciu</author>
</book>
```

Declared node type=Book

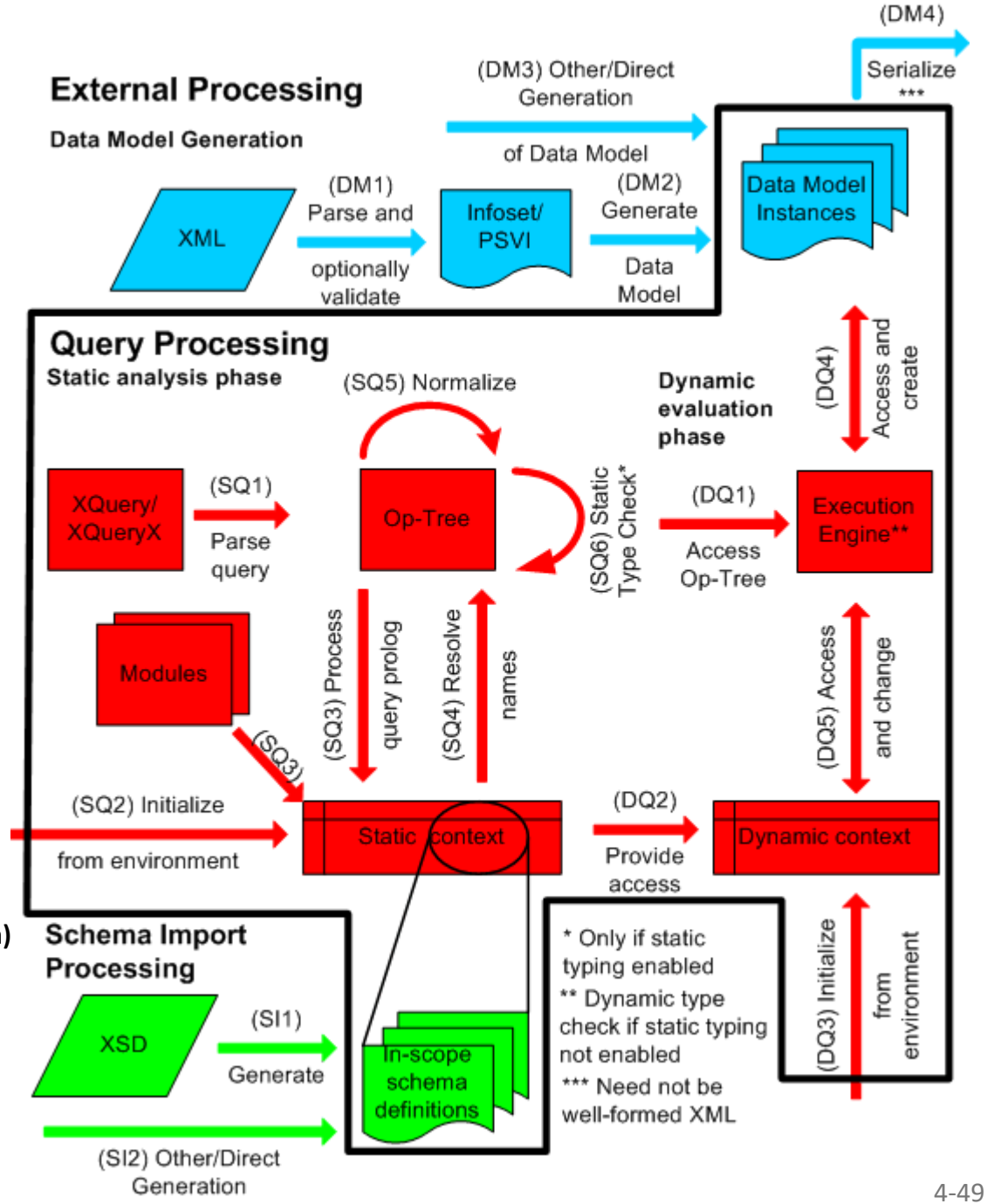
Type cast
- treat → static
- cast → dynamic

Semantics differs from TREAT and CAST in SQL:1999

```
type Book0 = element book
   (attribute year (xs:integer) &
    attribute isbn (xs:string),
    element title (xs:string),
    (element author (xs:string))*
)

for $p in $book0/book/
return $p cast as Book0

yields

<book year"1999" isbn="1-55860-622-X">
 <title>Data on the Web</title>
 <author>Abiteboul</author>
 <author>Buneman</author>
 <author>Suciu</author>
</book>
```

Most special node type=Book0

XQuery Processing Model



* figure taken from

XQuery 1.0: An XML Query Language (Second Edition)
W3C Recommendation 14 December 2010

http://www.w3.org/TR/xquery/

Comparison XQuery and SQL

XQuery	SQL
`for $k in /bookstore/book` `return $k`	`SELECT *` `FROM bookstore`
`for $k in //book` `return $k`	`SELECT *` `FROM bookstore`
`for $k in //book/title` `return $k`	`SELECT title` `FROM bookstore`
`for $k in //book` `return $k/title`	`SELECT title` `FROM bookstore`
`for $k in //book/author` `return $k/last-name`	`SELECT author.last-name` `FROM bookstore`

`for $k in /bookstore/book`
`where $k/title='XML and Databases'`
`order by $k/author/last-name`
`return $k/author`
	`SELECT author`
	`FROM bookstore`
	`WHERE title='XML and Databases'`
	`ORDER BY author.last-name`

`for $k in /bookstore/book`
`where count($k/author) > 2`
`return $k/title`
	`SELECT title`
	`FROM bookstore`
	`GROUP BY title`
	`HAVING COUNT(author) > 2`

Summary: XQuery

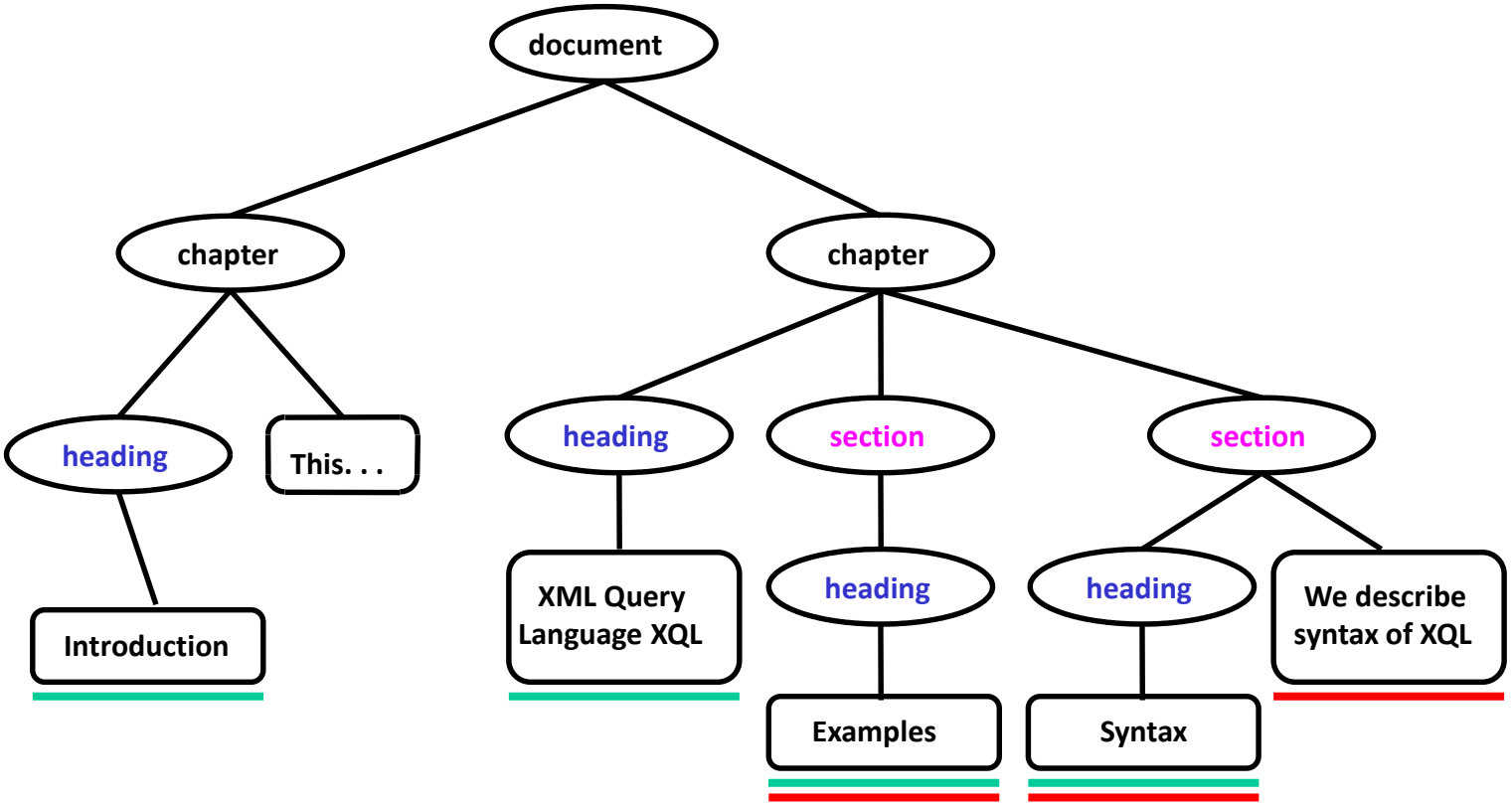
- **Standard for XML Query Languages**
- **Based on a tree model that supports all XML node types**
- **Well-defined semantics**
- **Strongly typed**
- **Support all requirements of XML Query Languages**
  - Selection
  - Extraction and reduction
  - Combination and restructuring
  - Preservation document order
- **Provides SQL goodies**
  - Grouping and aggregate functions
  - Sorting
  - Dynamic and static typing

Extending Query Languages with IR Functionality

- **XPath/XQuery focus on largely structured XML documents**
  - Precise predicates (exact match)
  - Well-defined result sets
  - Operations: selection, extraction, restructuring, aggregation
  - → Data-oriented view
- **Document-oriented View**
  - XML used as format for representing the logical structure of (text) documents
  - XPath/XQuery support only simple boolean retrieval but NOT IR on XML documents
    - No search for single word occurrences and substring matches
    - No weighting of descriptors
    - No relevance-oriented ranking of result sets
- **IR extensions needed**
  - Weighting and ranking
  - Relevance-oriented search
  - Data types with vague predicates
  - Structural relativism

Weighting and Ranking

- **Classic IR considers only entire documents**
- **XML Retrieval in contrast can restrict conditions to specific parts of the documents**
  - `/document[.//heading ∋ "XML" ∨ .//section//* ∋ "XML"]`
- **Problem: Weighting terms of different types**



Relevance-Oriented Search

- **Content-only Queries**
  - Expressions do not refer to the document structure
  - Example: "Search for XML Query Languages"
  - Retrieval strategy: return most specific sub tree that matches the given query best, i.e., has the highest retrieval status value
- **Content-and-Structure Queries**
  - Expressions formulate restrictions on the document structure
  - Example: "Search for all abstract or conclusion elements dealing with XML Query Languages"
  - Retrieval strategy : return the structure elements with the highest retrieval status value which satisfy the conditions on the structure

Data Types with Vague Predicates

- **Example**
  - Query: Search for informations about the work of an artist called Ulbrich who was active around 1900 in the Rhein/Main area
  - Actual target: Ernst Olbrich, Darmstadt, 1899
- **Extended data types for document-oriented view**
  - Person names
  - Dates
  - Geographic nomenclature
  - Images, audio, video, ...
- **Idea: Exploit XML markups for formulating more precise search queries while considering uncertainty and vagueness**

Structural Relativism

- **XPath only supports precise conditions in path expressions**
  - Example: `/store/auction/name[last-name="Schek"]`
  - Example with wildcards: `//name[last-name="Schek"]`
  - Query writing requires good knowledge about the structure of the given documents
  - In big document collections, it is a unrealistic that a user has this knowledge
- **Structural relativism extends relevance-oriented search to paths and path expressions**
  - No distinction between elements and attributes
  - Search in all elements of a given data type, e.g. Date
  - Search for elements that contain a given keyword in their path
  - Search for elements that are on a path with the highest relevance w.r.t. a given query text

Query Language XIRQL

- **Extension of XPath expressions**
  - Probabilistic retrieval based on weighted query conditions
    - `//*[0.7 . $c-word$ "retrieval" + 0.3 . $c-word$ "XML"]`
  - Relevance-oriented search: IR search restricted to selected parts of XML documents
    - `//section[... $c-phrase$ "XML retrieval"]`
  - Data types with vague predicates instead of "=" or "<"
    - Keyword search: `//title $c-word$ "autobiography"`
    - Phonetic match: `//author $soundslike$ "franklin"`
  - Structural relativism do not distinguish between elements and attributes
    - `//#author $soundslike$ "franklin"`

- **XIRQL provides a set of operators and allows to define own new operators and data types**

Operator	Semantics
nodeset **$c-word$** string	Weighted search for word occurrences
nodeset **$c-phrase$** string	Weighted search for phrase occurrences
nodeset **$soundslike$** string	Weighted phonetic search
#name	No distinction between attributes and elements

XIRQL: Example Document

```
<bib>
 <book year="1999" isbn="1-55860-622-X">
  <title>Data on the Web</title>
  <author>Abiteboul</author>
  <author>Buneman</author>
  <author>Suciu</author>
  <abstract> The Web is causing a revolution in how we
    present, retrieve, and process information. ...</abstract>
 </book>
 <book year="2001" isbn="1-XXXXX-YYY-Z">
  <title>XML Query</title>
  <author>Fernandez</author>
  <author>Suciu</author>
  <summary>...</summary>
 </book>
</bib>
```

XIRQL: Example Query (1)

```
Weighted Keyword Search

//book[abstract $c-word$ "Web"]/title

yields

<title rsv="0.75">Data on the Web</title>

<title rsv="0.1">XML Query</title>
```

Weighting the results

XIRQL: Example Query (2)

Individual Weighting of Query Conditions

```
//book[ 0.7 ./abstract $c-word$ "Web" +
       0.3 ./author $soundslike$ "Sutschu"]
```

Different weights for query condition parts

yields

```
<book year="1999" isbn="1-55860-622-X" rsv="0.6">
 <title>Data on the Web</title>
 <author>Abiteboul</author>
 <author>Buneman</author>
 <author>Suciu</author>
 <abstract>...</abstract>
</book>
<book year="2001" isbn="1-XXXXX-YYY-Z" rsv="0.2">
 <title>XML Query</title>
 <author>Fernandez</author>
 <author>Suciu</author>
 <summary>...</summary>
</book>
```

XIRQL: Example Query (3)

Retrieval of XML trees

```
//*[ ... $c-word$ "Web"]
```

yields

```
<title rsv="0.8">Data on the Web</title>
<book year="1999" isbn="1-55860-622-X" rsv="0.6">
    <title>Data on the Web</title>
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
    <abstract> … </abstract>
</book>
<summary rsv="0.2"> … </summary>
```

Sub tree operator

Different types in the result – the most specific result with the highest relevance should be top in the ranking

- **Relevance ranking shall**
  - consider the structure of XML documents
  - deliver best matching documents as top of the ranking

Comparison of XML Query Languages

General Requirements	XPath	XQuery	XIRQL
Schema Awareness	+	+	+
Flexible Types	-	+	?
Embedding	-	+	-
Order Preservation	+	+	+
Weighted Queries	-	-	+

Operation	XPath	XQuery	XIRQL
Selection	+	+	+
Extraction and Reduction	+	+	+
Combination and Restructuring	-	+	+