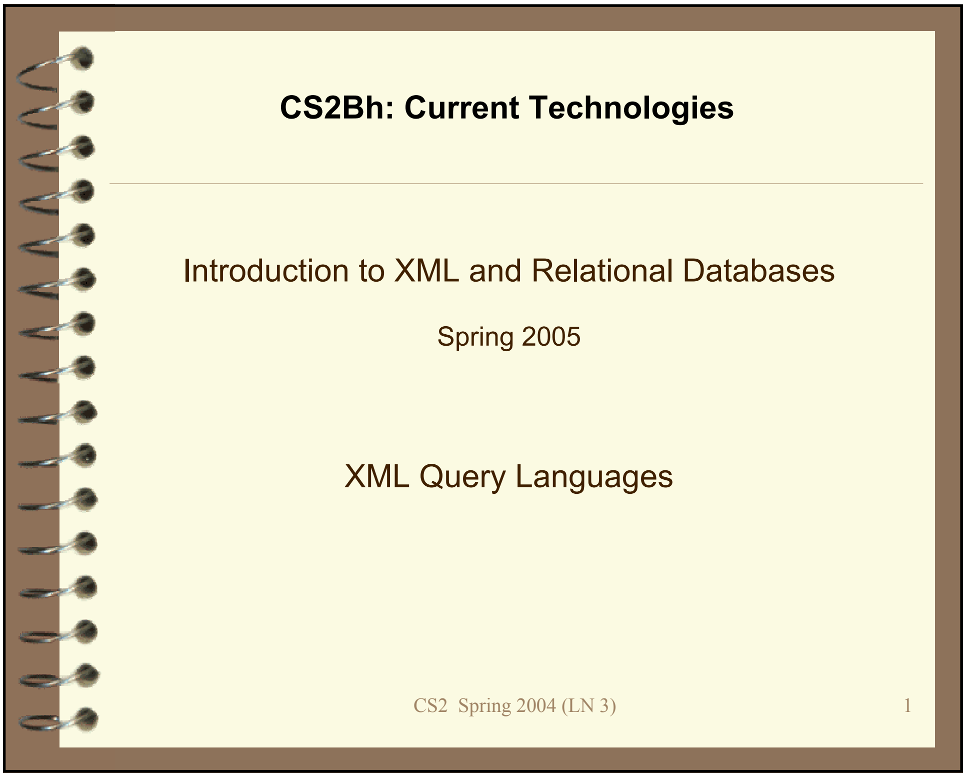
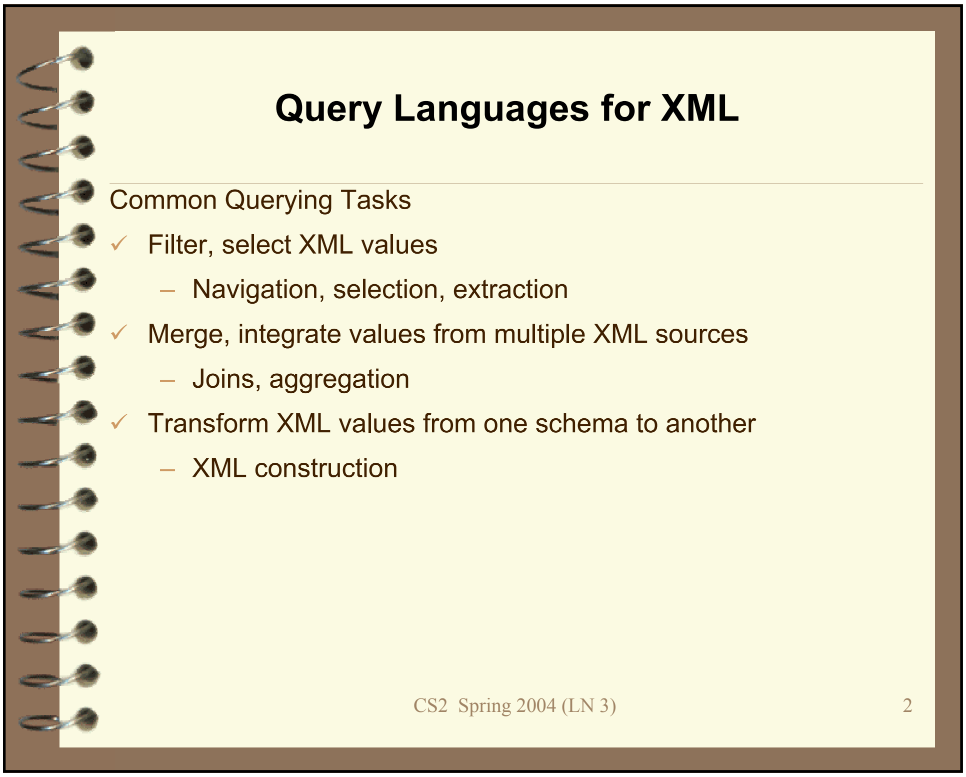


CS2Bh: Current Technologies

Introduction to XML and Relational Databases

Spring 2005

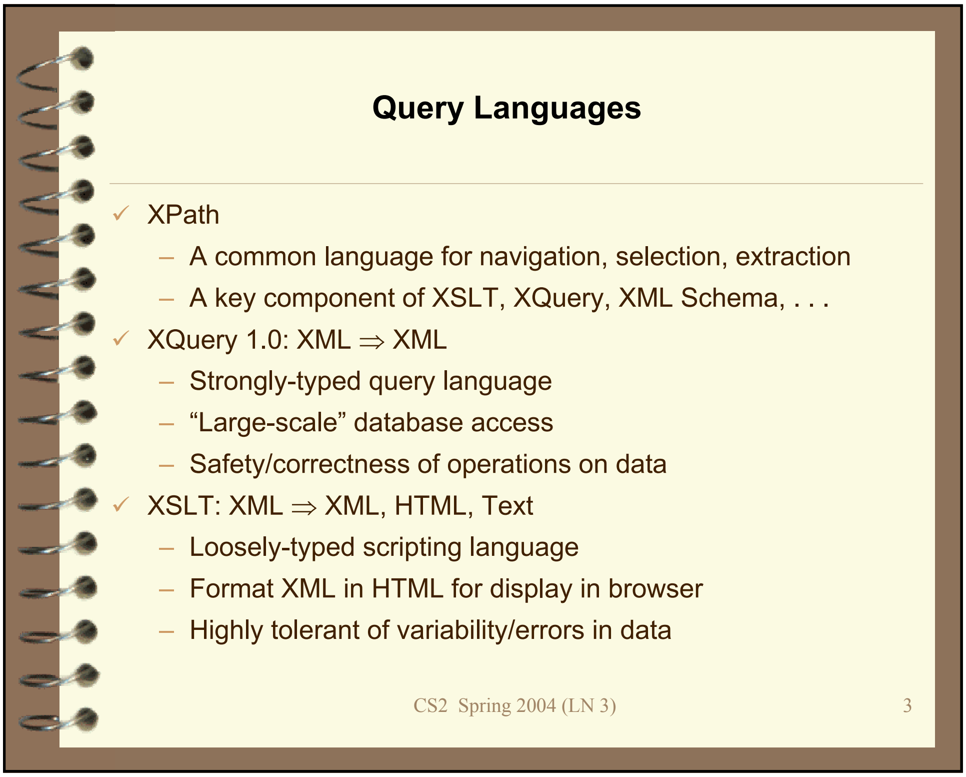
XML Query Languages



Query Languages for XML

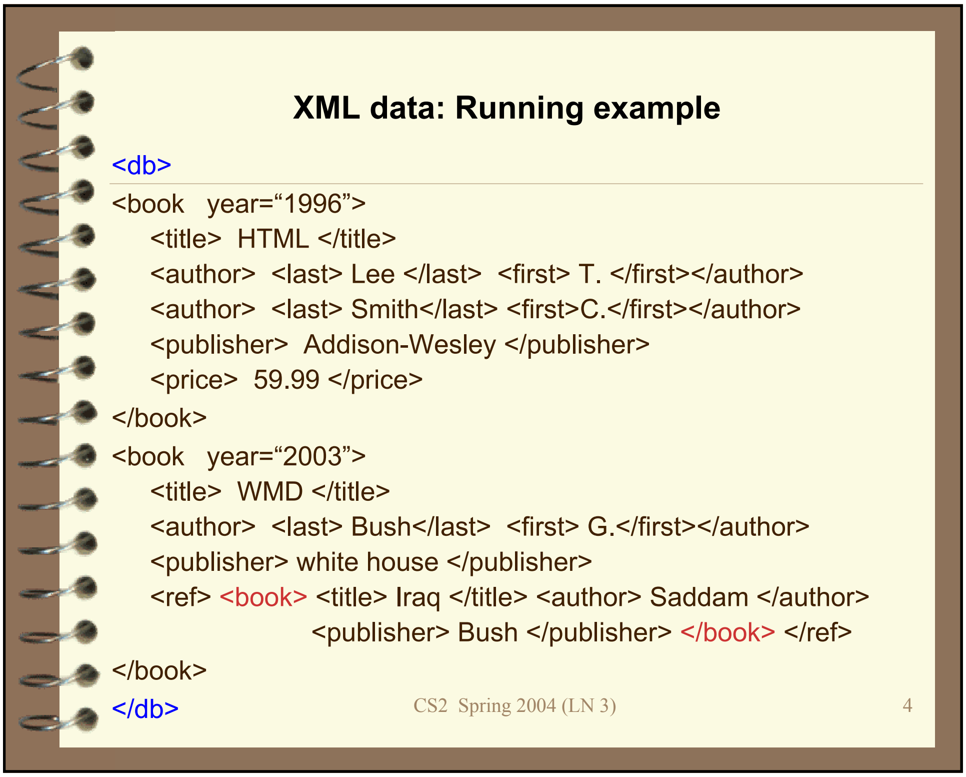
Common Querying Tasks

- ✓ Filter, select XML values
  - Navigation, selection, extraction
- ✓ Merge, integrate values from multiple XML sources
  - Joins, aggregation
- ✓ Transform XML values from one schema to another
  - XML construction



Query Languages

- ✓ XPath
  - A common language for navigation, selection, extraction
  - A key component of XSLT, XQuery, XML Schema, . . .
- ✓ XQuery 1.0: XML ⇒ XML
  - Strongly-typed query language
  - “Large-scale” database access
  - Safety/correctness of operations on data
- ✓ XSLT: XML ⇒ XML, HTML, Text
  - Loosely-typed scripting language
  - Format XML in HTML for display in browser
  - Highly tolerant of variability/errors in data

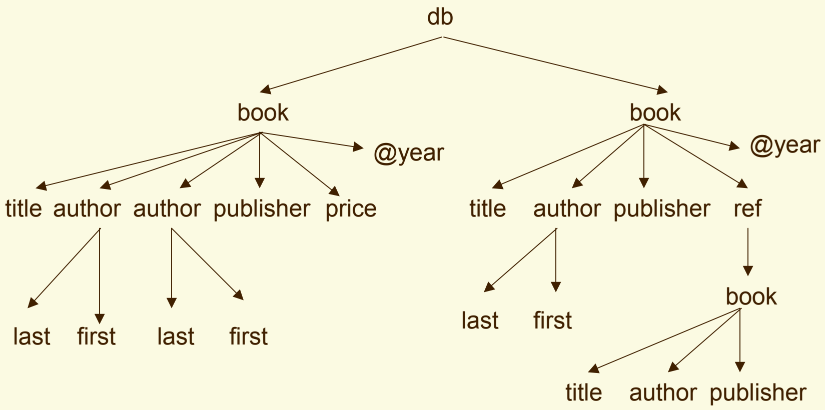


XML data: Running example

```
<db>
<book   year=“1996”>
    <title>  HTML </title>
    <author>  <last> Lee </last>  <first> T. </first></author>
    <author>  <last> Smith</last> <first>C.</first></author>
    <publisher>  Addison-Wesley </publisher>
    <price>  59.99 </price>
</book>
<book   year=“2003”>
    <title>  WMD </title>
    <author>  <last> Bush</last>  <first> G.</first></author>
    <publisher> white house </publisher>
    <ref> <book> <title> Iraq </title> <author> Saddam </author>
                    <publisher> Bush </publisher> </book> </ref>
</book>
</db>
```

Data model

Node-labeled, ordered tree



db

book book

@year @year

title author author publisher price

title author publisher ref

last first last first

last first

book

title author publisher

XPath

W3C standard: www.w3.org/TR/xpath

- ✓ Navigating an XML tree and finding parts of the tree (node selection and value extraction)

  Given an XML tree T and a context node n, an XPath query Q returns

  - the set of nodes reachable via Q from the node n in T – if Q is a unary query
  - truth value indicating whether Q is true at n in T – if Q is a Boolean query (filter).
- ✓ Implementations: XALAN, SAXON, Berkeley DB XML – freeware, which you can play with
- ✓ A major element of XSLT, XQuery and XML Schema
- ✓ XPath 2.0 (Turing-Complete)

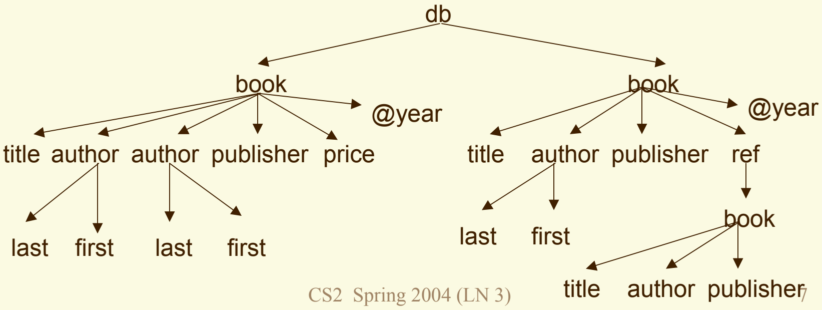
XPath constructs

XPath query Q:

- Tree traversal: downward, upward, sideways
- Relational/Boolean expressions: qualifiers (predicates)
- Functions: aggregation (e.g., count), string functions

//author[last="Bush"]

//book[author/last="Bush"]/title | //book[author/last="Blair"]/title



Downward traversal

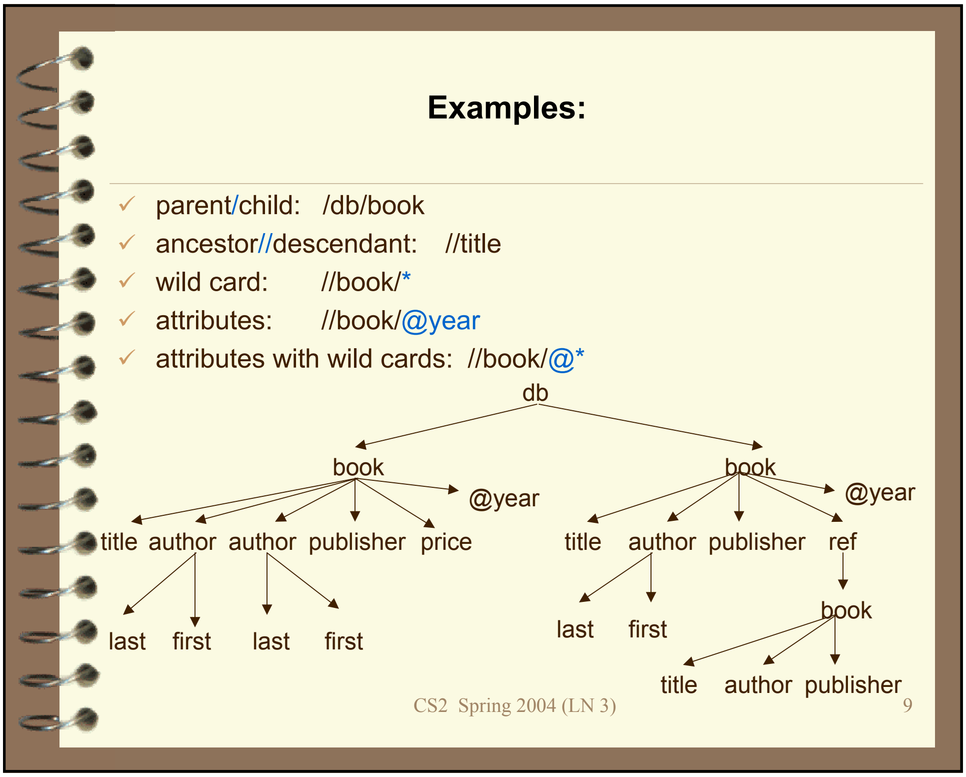
Syntax:

Q ::= . | A | @a | Q/Q | Q//Q | /Q | Q[q]

q ::= Q | Q op c | q and q | q or q | not(q)

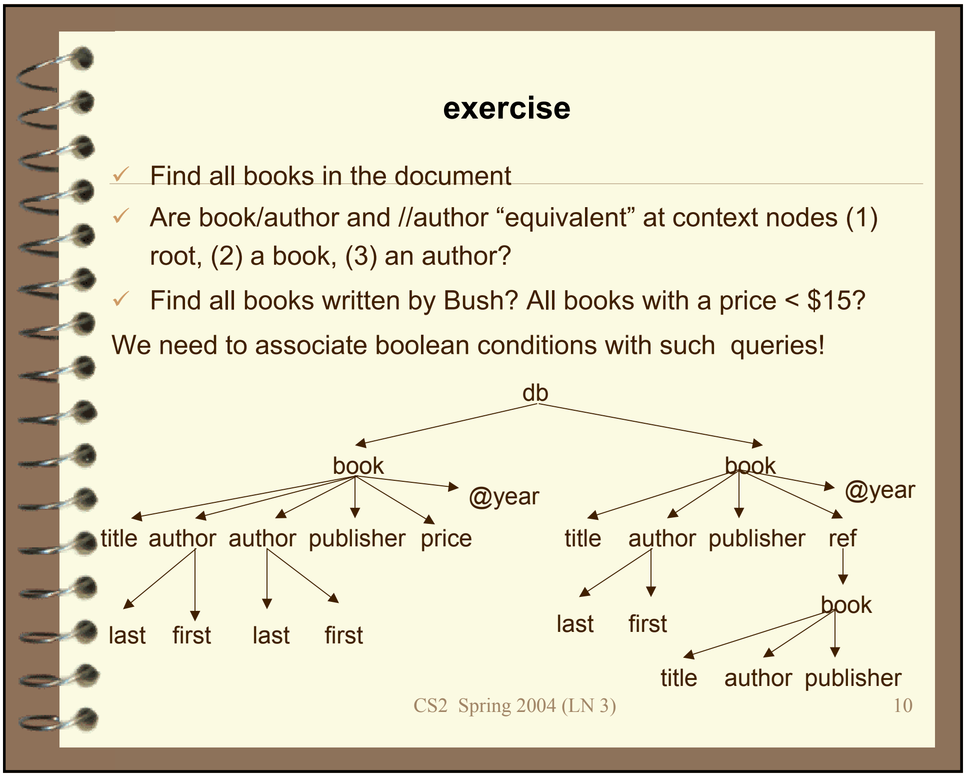
- ✓ .: self, the current node
- ✓ A: either a tag (label) or *: wildcard that matches any label
- ✓ @a: attribute
- ✓ /: concatenation (child)
- ✓ //: descendants or self, "recursion"
- ✓ [q]: qualifier (filter, predicate)
  - op: =, !=, <=, <, >, >=, >
  - c: constant
  - and, or, not(): conjunction, disjunction, negation

Existential semantics: /db/book[author/last="Bush"]



Examples:

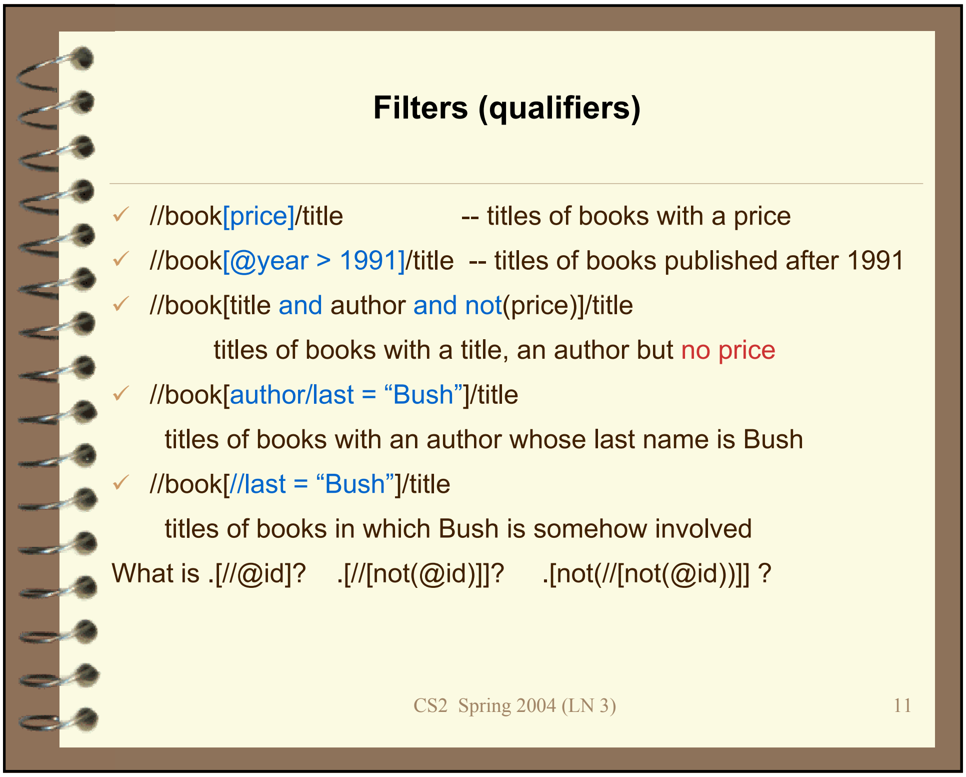
- ✓ parent/child: /db/book
- ✓ ancestor//descendant: //title
- ✓ wild card: //book/*
- ✓ attributes: //book/@year
- ✓ attributes with wild cards: //book/@*



exercise

- ✓ Find all books in the document
- ✓ Are book/author and //author “equivalent” at context nodes (1) root, (2) a book, (3) an author?
- ✓ Find all books written by Bush? All books with a price < $15?

We need to associate boolean conditions with such queries!



Filters (qualifiers)

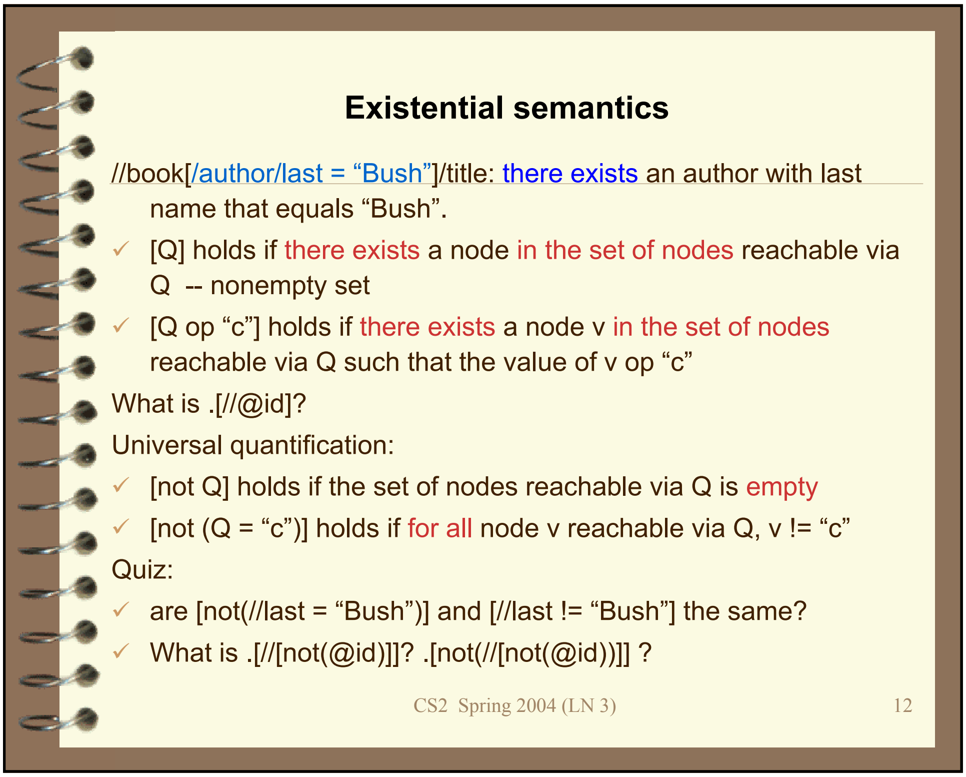
- //book[price]/title -- titles of books with a price
- //book[@year > 1991]/title -- titles of books published after 1991
- //book[title and author and not(price)]/title

  titles of books with a title, an author but no price
- //book[author/last = “Bush”]/title

  titles of books with an author whose last name is Bush
- //book[//last = “Bush”]/title

  titles of books in which Bush is somehow involved

What is .[//@id]? .[//[not(@id)]]? .[not(//[not(@id))]] ?



Existential semantics

//book[/author/last = “Bush”]/title: there exists an author with last name that equals “Bush”.

- [Q] holds if there exists a node in the set of nodes reachable via Q -- nonempty set
- [Q op “c”] holds if there exists a node v in the set of nodes reachable via Q such that the value of v op “c”

What is .[//@id]?

Universal quantification:

- [not Q] holds if the set of nodes reachable via Q is empty
- [not (Q = “c”)] holds if for all node v reachable via Q, v != “c”

Quiz:

- are [not(//last = “Bush”)] and [//last != “Bush”] the same?
- What is .[//[not(@id)]]? .[not(//[not(@id))]] ?

Other useful XPath constructs

- ✓ union: (//book/editor | //book/author) -- top-level or in qualifiers

XPath does not allow Q/(Q1 | Q2)

- ✓ text(): all the text children of the current node, e.g., //text()
- ✓ node(): all the children of the context node, including text and attribute nodes, e.g., //node()
- ✓ position:
  - *[2]: the second child of the current node
  - author[2]: the second author of the current node
  - *[last()]: the last child of the current node

Upward traversal

Syntax:

Q ::= . . . | ../Q | ancestor ::Q | ancestor-or-self::Q

- ✓ ../: parent
- ✓ ancestor, ancestor-or-self: recursion

Example:

- ✓ //author[../title = "WMD"]/last

  find the last names of authors of books with the title "WMD"
- ✓ ancestor :: book[author/last="Bush"]

  find book ancestors with a descendant that has a "last" (name) child that equals "Bush"

Are the following equivalent to each other at a book context node?

../book/author, ./author

Sideways

Syntax:

Q ::= . . . | following-sibling ::Q | preceding-sibling::Q

- following-sibling: the next sibling
- preceding-sibling: the previous sibling
- position function: e.g., //author[position( ) < 2]

Example:

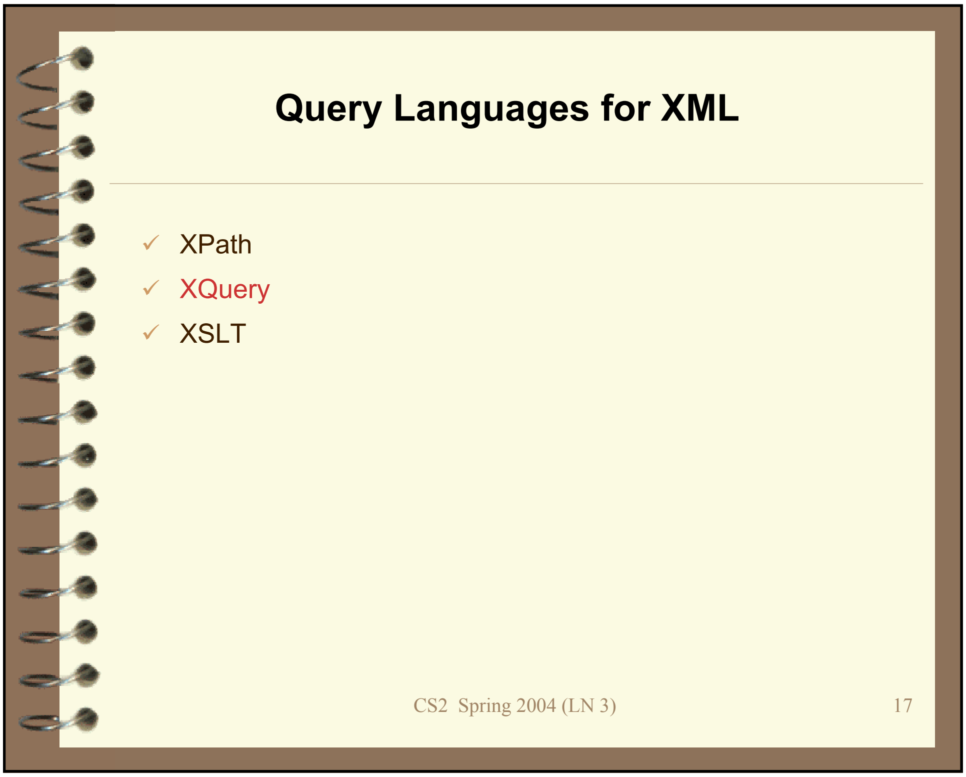
- following-sibling :: book [author/last="Bush"]

  find the next book written by Bush
- preceding-sibling :: book[author/last="Bush"]

  find the previous book written by Bush

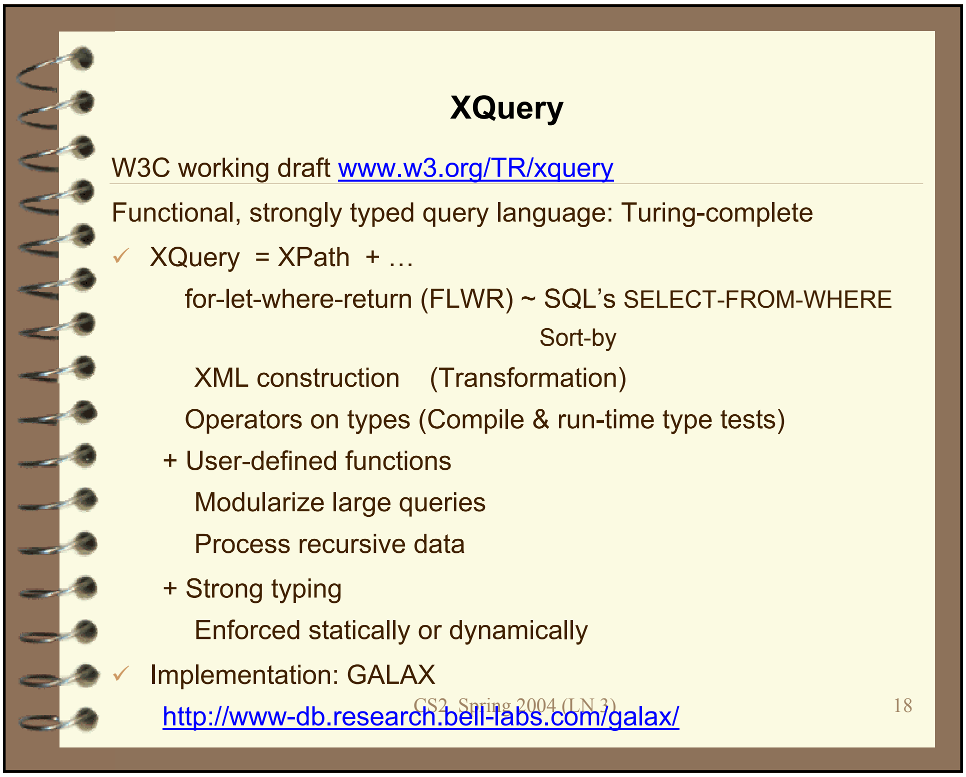
Why isn't XPath a proper database query language?

- It does not return XML: it returns a set of nodes – unary queries -- and can't organize the result in any structures beyond a set of nodes, e.g., it is not capable of finding, e.g., (title, author) pairs of books
- It can't do complex queries involving joins: it does not have an explicit notion of variables; it can't "remember" the nodes it visited



Query Languages for XML

- ✓ XPath
- ✓ XQuery
- ✓ XSLT

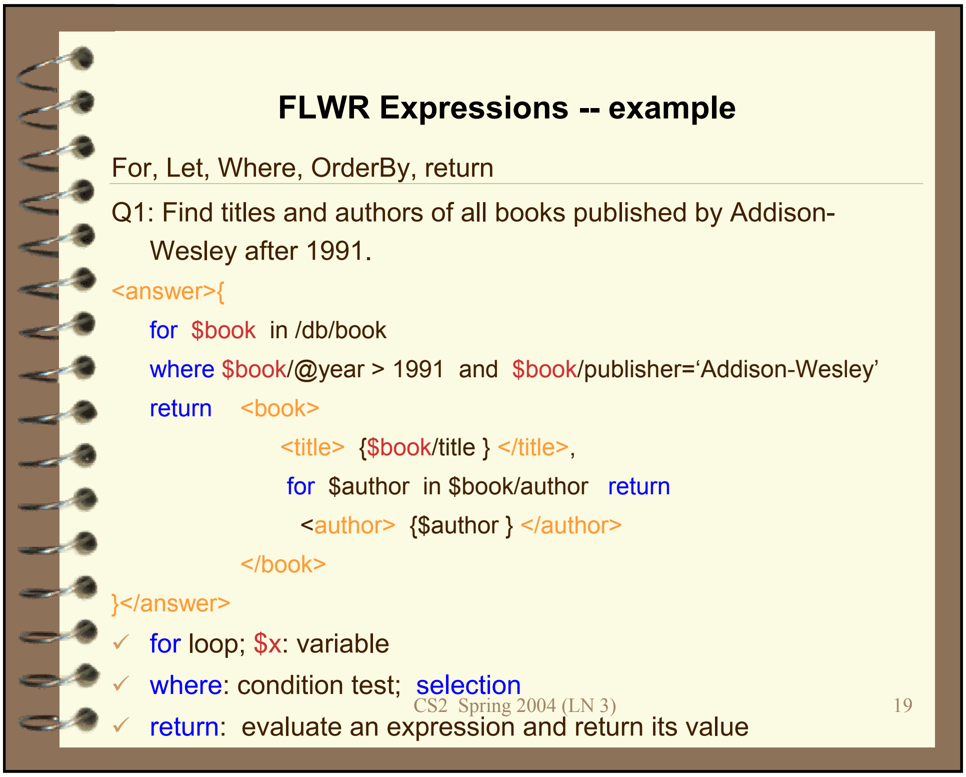


XQuery

W3C working draft www.w3.org/TR/xquery

Functional, strongly typed query language: Turing-complete

- ✓ XQuery = XPath + …
  - for-let-where-return (FLWR) ~ SQL's SELECT-FROM-WHERE
  - Sort-by
  - XML construction (Transformation)
  - Operators on types (Compile & run-time type tests)
  - + User-defined functions
    - Modularize large queries
    - Process recursive data
  - + Strong typing
    - Enforced statically or dynamically
- ✓ Implementation: GALAX
  - http://www-db.research.bell-labs.com/galax/



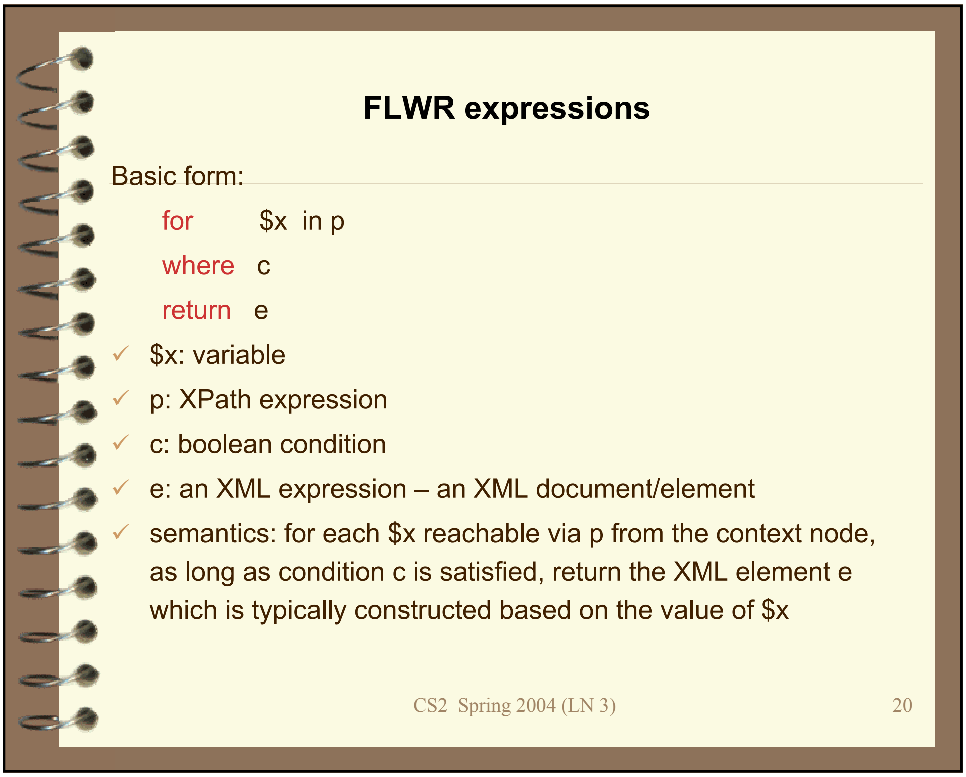
FLWR Expressions -- example

For, Let, Where, OrderBy, return

Q1: Find titles and authors of all books published by Addison-Wesley after 1991.

```
<answer>{
    for  $book  in /db/book
    where $book/@year > 1991  and  $book/publisher='Addison-Wesley'
    return    <book>
                  <title>  {$book/title } </title>,
                  for  $author  in $book/author   return
                    <author>  {$author } </author>
              </book>
}</answer>
```

- ✓ for loop; $x: variable
- ✓ where: condition test; selection
- ✓ return: evaluate an expression and return its value



FLWR expressions

Basic form:

```
for       $x  in p
where   c
return   e
```

- ✓ $x: variable
- ✓ p: XPath expression
- ✓ c: boolean condition
- ✓ e: an XML expression – an XML document/element
- ✓ semantics: for each $x reachable via p from the context node, as long as condition c is satisfied, return the XML element e which is typically constructed based on the value of $x

join

Find books that cost more at Amazon than at BN

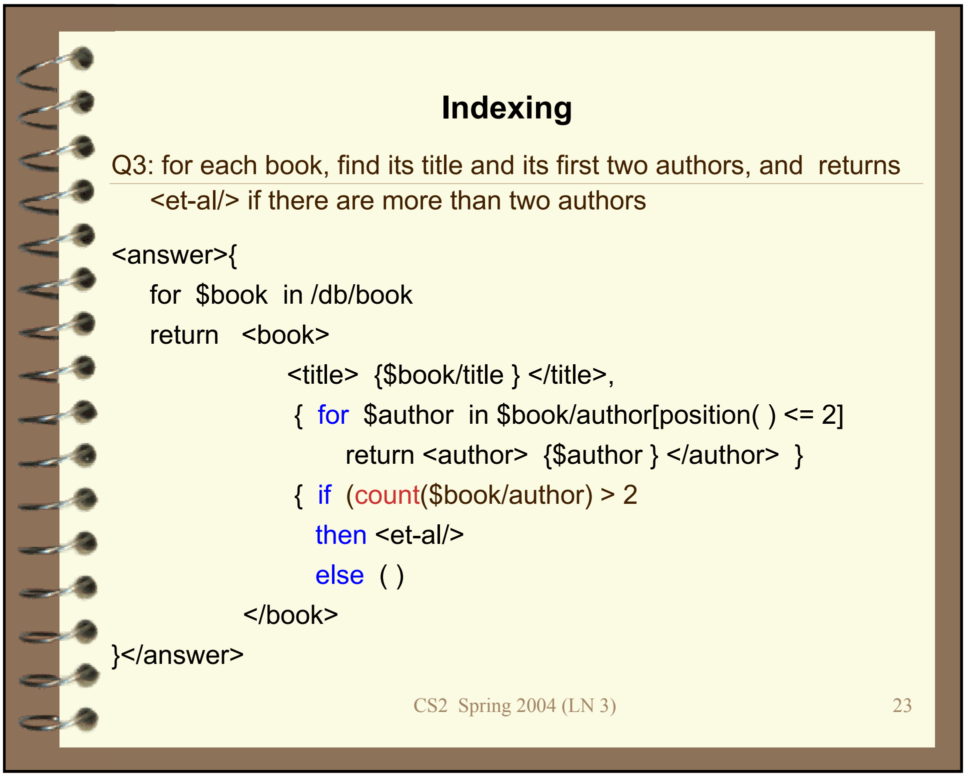
```
<answer>{
    let  $amazon := doc(“http://www.amozon.com/books.xml”),
         $bn  :=  doc(“http://www.BN.com/books.xml”)
    for    $a  in $amozon/books/book,
           $b  in $bn/books/book
    where    $a/isbn  = $b/isbn    and     $a/price > $b/price
    return   <book> {$a/title, $a/price, $b/price }  <book>
}</answer>
```

- ✓ let clause
- ✓ join: of two documents
- ✓ Equality test: existential semantics like in XPath

Conditional expression

Q2: Find all book titles, and prices where available

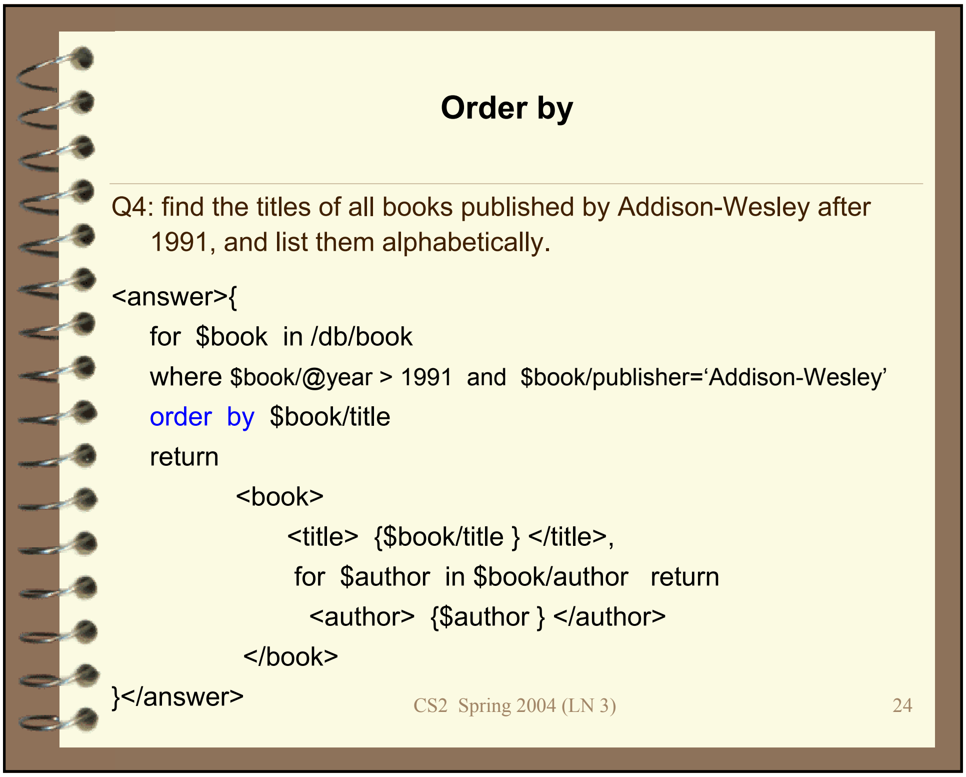
```
<answer>{
    for  $book  in /db/book
    return   <book>
                  <title>  {$book/title } </title>,
                  { if   $book[price]
                    then  <price> {$book/price } </price>
                    else   ( ) }
             </book>
}</answer>
```



Indexing

Q3: for each book, find its title and its first two authors, and returns <et-al/> if there are more than two authors

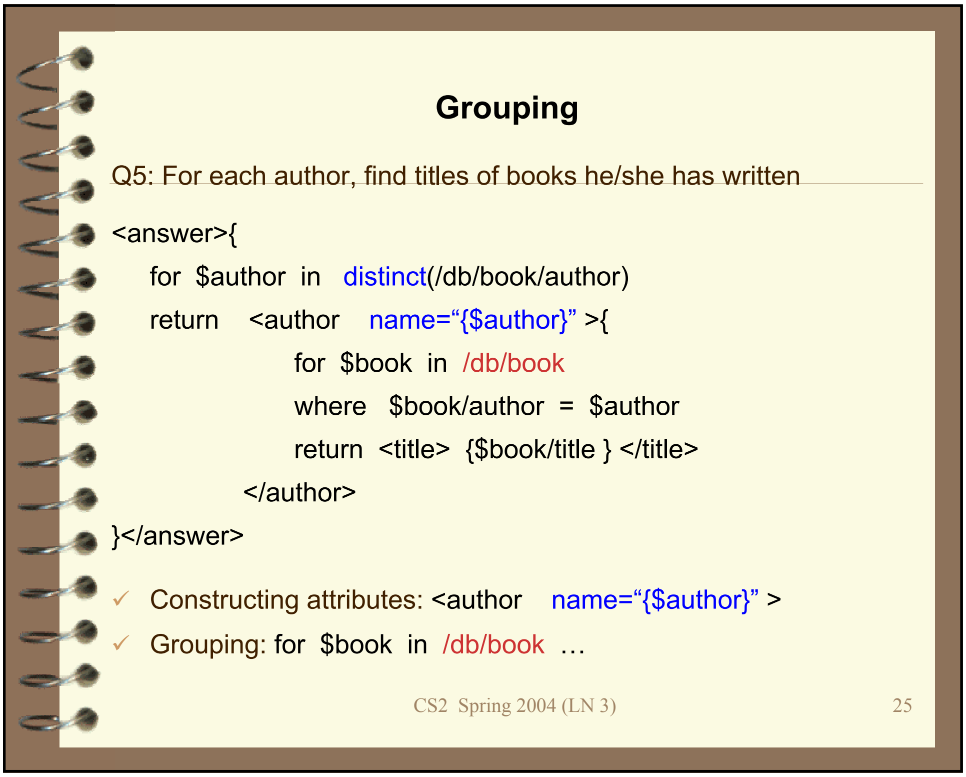
```
<answer>{
    for  $book  in /db/book
    return   <book>
                   <title>  {$book/title } </title>,
                    {  for  $author  in $book/author[position( ) <= 2]
                         return <author>  {$author } </author>  }
                    {  if  (count($book/author) > 2
                       then <et-al/>
                       else  ( )
               </book>
}</answer>
```



Order by

Q4: find the titles of all books published by Addison-Wesley after 1991, and list them alphabetically.

```
<answer>{
    for  $book  in /db/book
    where $book/@year > 1991  and  $book/publisher='Addison-Wesley'
    order  by  $book/title
    return
             <book>
                   <title>  {$book/title } </title>,
                    for  $author  in $book/author   return
                      <author>  {$author } </author>
              </book>
}</answer>
```

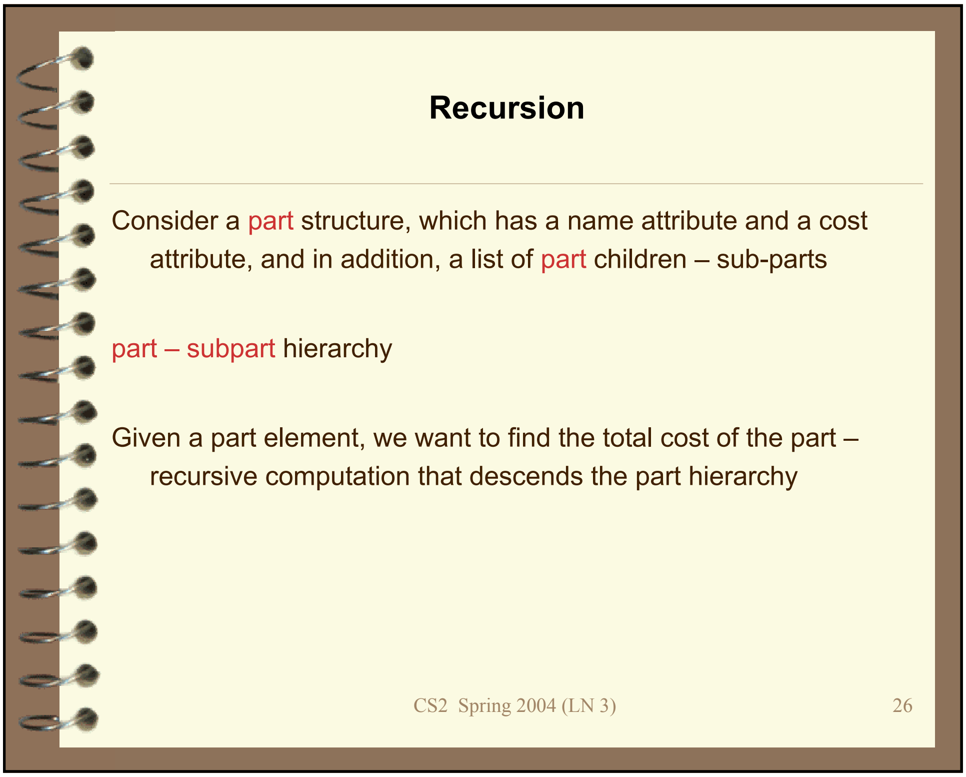


Grouping

Q5: For each author, find titles of books he/she has written

```
<answer>{
    for  $author  in   distinct(/db/book/author)
    return    <author    name="{$author}" >{
                        for  $book  in  /db/book
                        where   $book/author  =  $author
                        return  <title>  {$book/title } </title>
                   </author>
}</answer>
```

- ✓ Constructing attributes: <author   name="{$author}" >
- ✓ Grouping: for $book in /db/book …

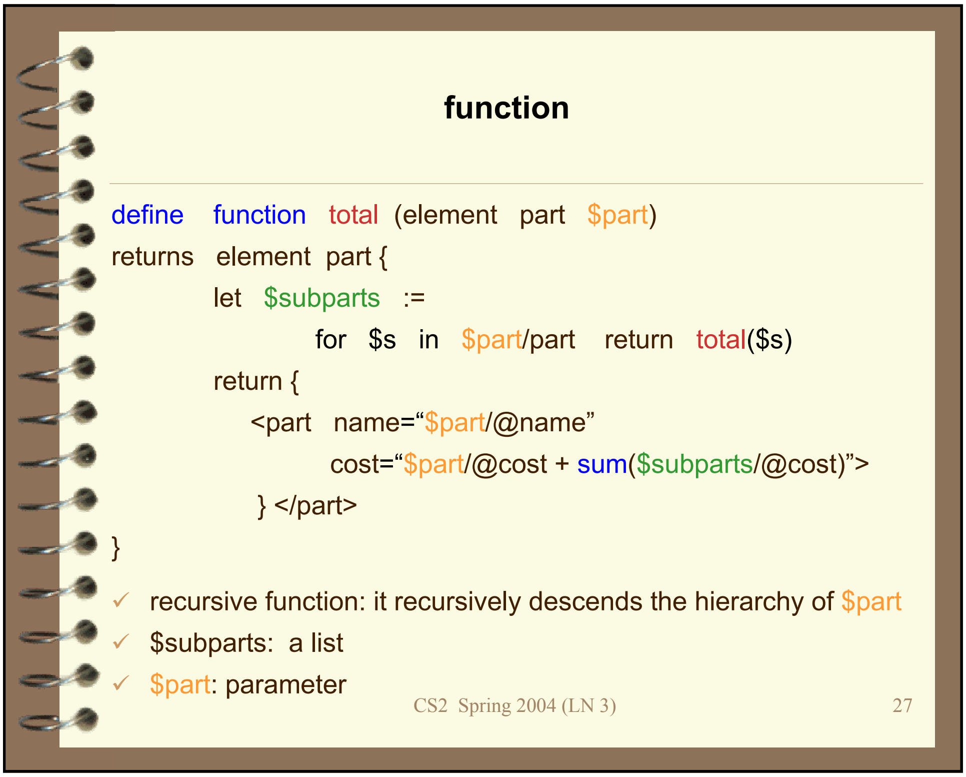


Recursion

Consider a part structure, which has a name attribute and a cost attribute, and in addition, a list of part children – sub-parts

part – subpart hierarchy

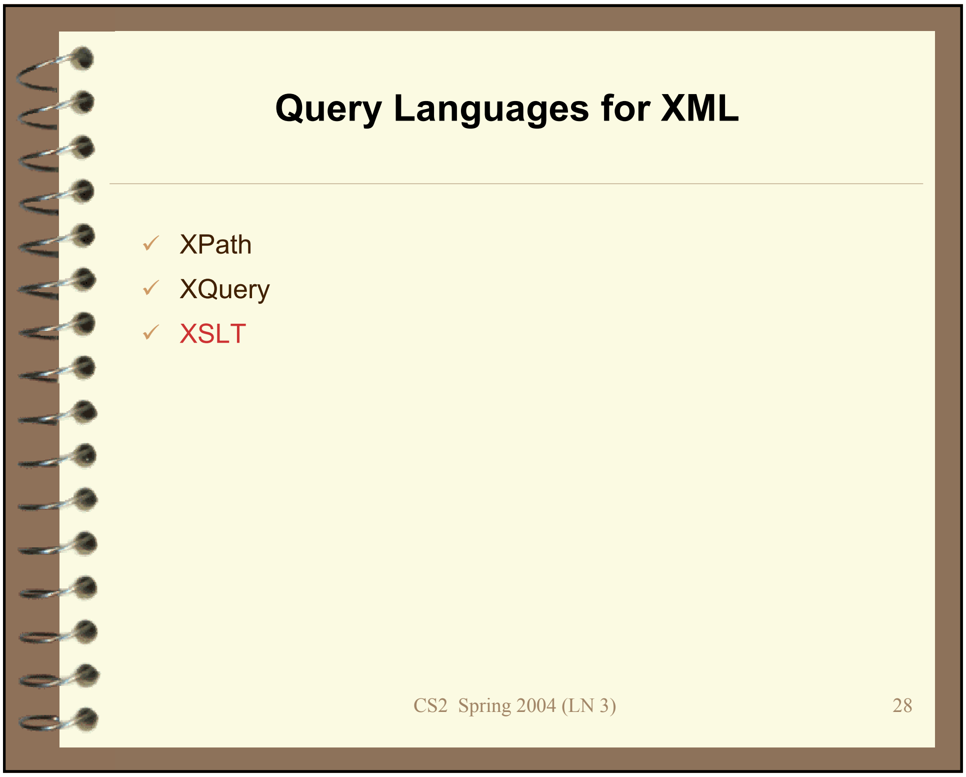
Given a part element, we want to find the total cost of the part – recursive computation that descends the part hierarchy



function

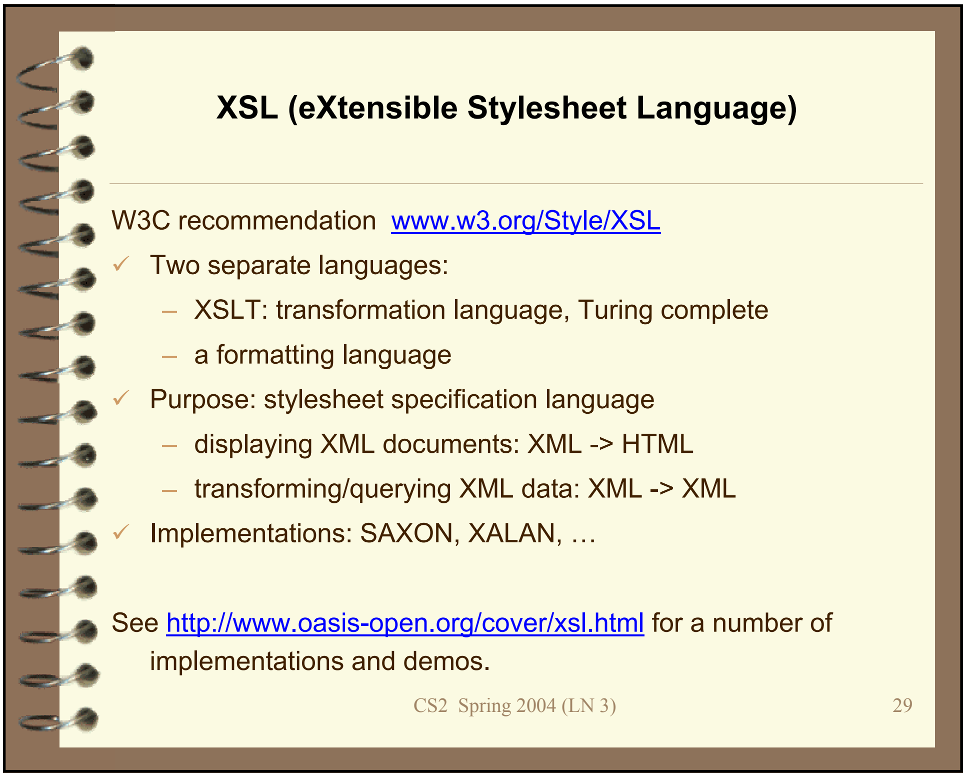
```
define   function   total  (element   part   $part)
returns   element  part {
            let   $subparts   :=
                        for   $s   in   $part/part    return   total($s)
            return {
                <part   name="$part/@name"
                         cost="$part/@cost + sum($subparts/@cost)">
                 } </part>
}
```

- ✓ recursive function: it recursively descends the hierarchy of $part
- ✓ $subparts: a list
- ✓ $part: parameter



Query Languages for XML

- ✓ XPath
- ✓ XQuery
- ✓ XSLT

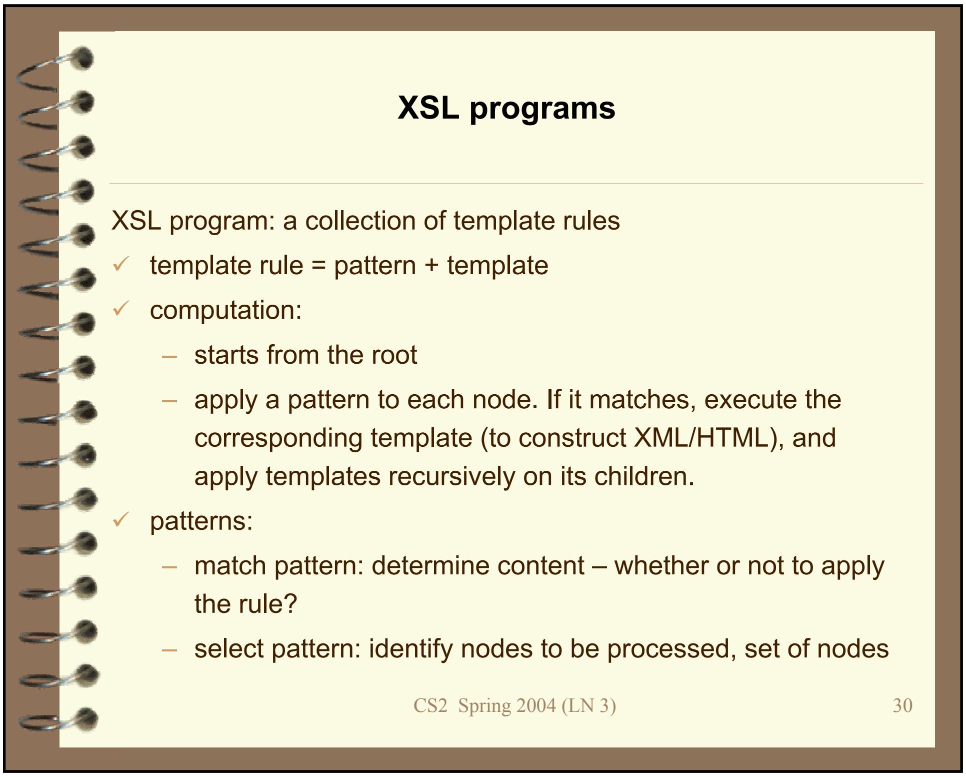


XSL (eXtensible Stylesheet Language)

W3C recommendation www.w3.org/Style/XSL

- ✓ Two separate languages:
  - XSLT: transformation language, Turing complete
  - a formatting language
- ✓ Purpose: stylesheet specification language
  - displaying XML documents: XML -> HTML
  - transforming/querying XML data: XML -> XML
- ✓ Implementations: SAXON, XALAN, …

See http://www.oasis-open.org/cover/xsl.html for a number of implementations and demos.



XSL programs

XSL program: a collection of template rules

- ✓ template rule = pattern + template
- ✓ computation:
  - starts from the root
  - apply a pattern to each node. If it matches, execute the corresponding template (to construct XML/HTML), and apply templates recursively on its children.
- ✓ patterns:
  - match pattern: determine content – whether or not to apply the rule?
  - select pattern: identify nodes to be processed, set of nodes

An example XSLT program

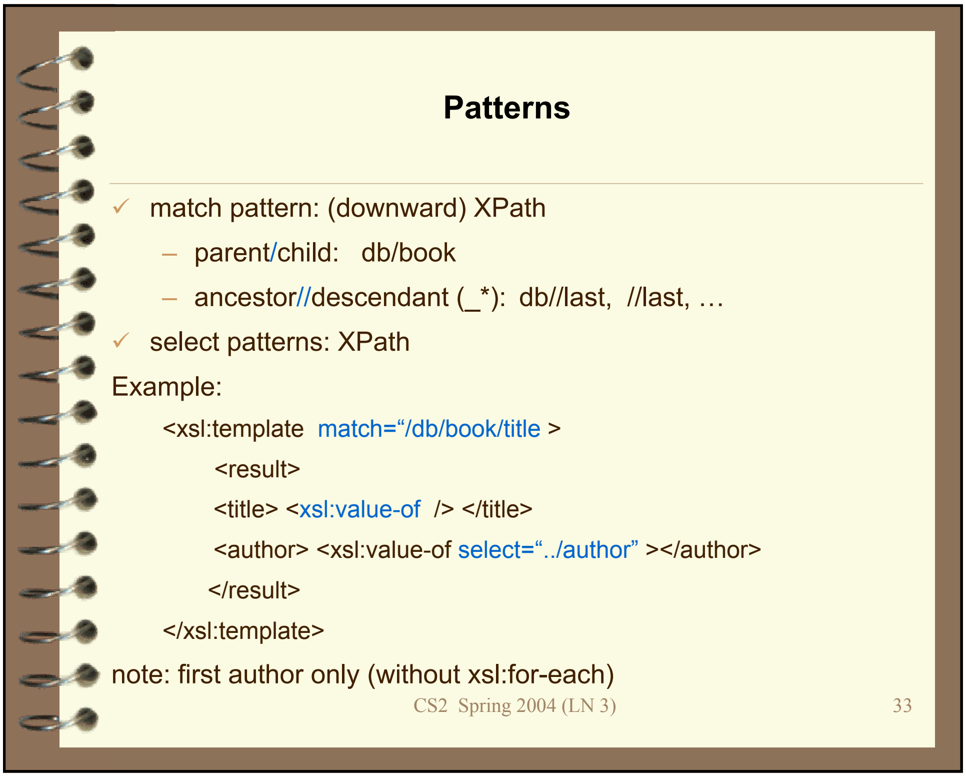
Q1: Find titles and authors of all books published by Addison-Wesley after 1991.

```
<xsl:template    match=“/db/book[@year > 1991 and
                          publisher=‘Addison-Wesley’]” >
   <result>
        <title> <xsl:value-of   select=“title” /> </title>
        <xsl:for-each select=“author” />
           <author><xsl:value-of /> </author>
        </xsl:for-each>
   </result>
</xsl:template>
```

Basic XSLT constructs

- ✓ a collection of templates: <xsl:template>
- ✓ match pattern: match=“/db/book[@year > 1991 and
  publisher=‘Addison-Wesley’]”
- ✓ select pattern: select=“title”, xsl:for-each select=“author”
- ✓ value-of: string
- ✓ constructing XML data:

```
   <result>
        <title> <xsl:value-of   select=“title” /> </title>
        ...
   </result>
```



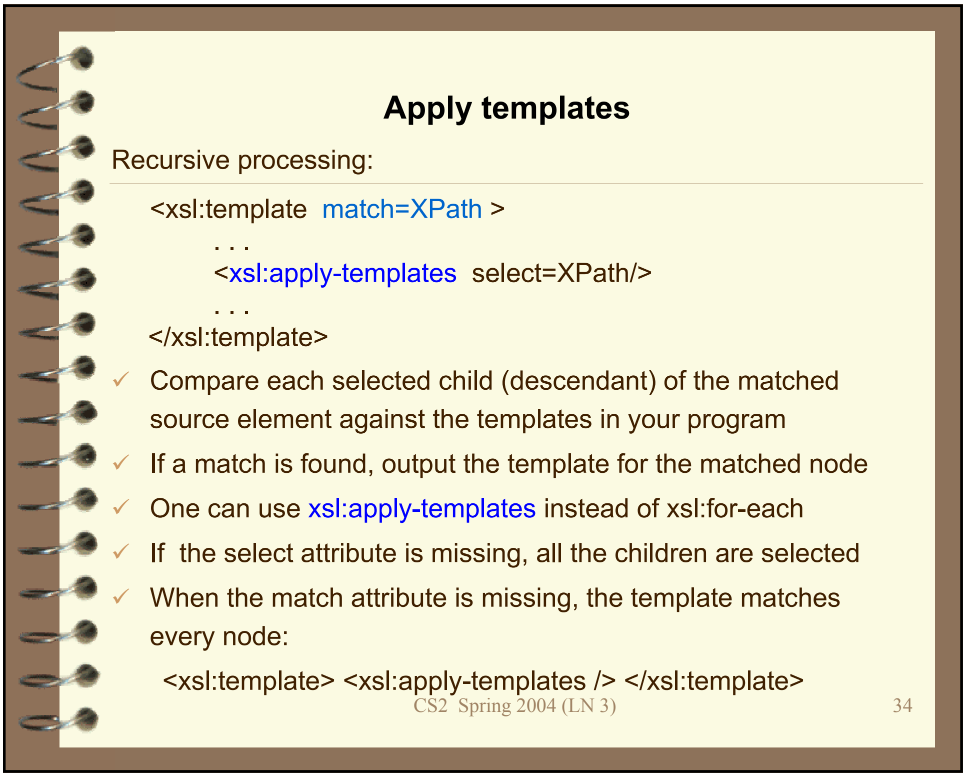
Patterns

- ✓ match pattern: (downward) XPath
  - – parent/child: db/book
  - – ancestor//descendant (_*): db//last, //last, …
- ✓ select patterns: XPath

Example:

```
    <xsl:template  match="/db/book/title >
          <result>
          <title> <xsl:value-of  /> </title>
          <author> <xsl:value-of select="../author" ></author>
         </result>
    </xsl:template>
```

note: first author only (without xsl:for-each)



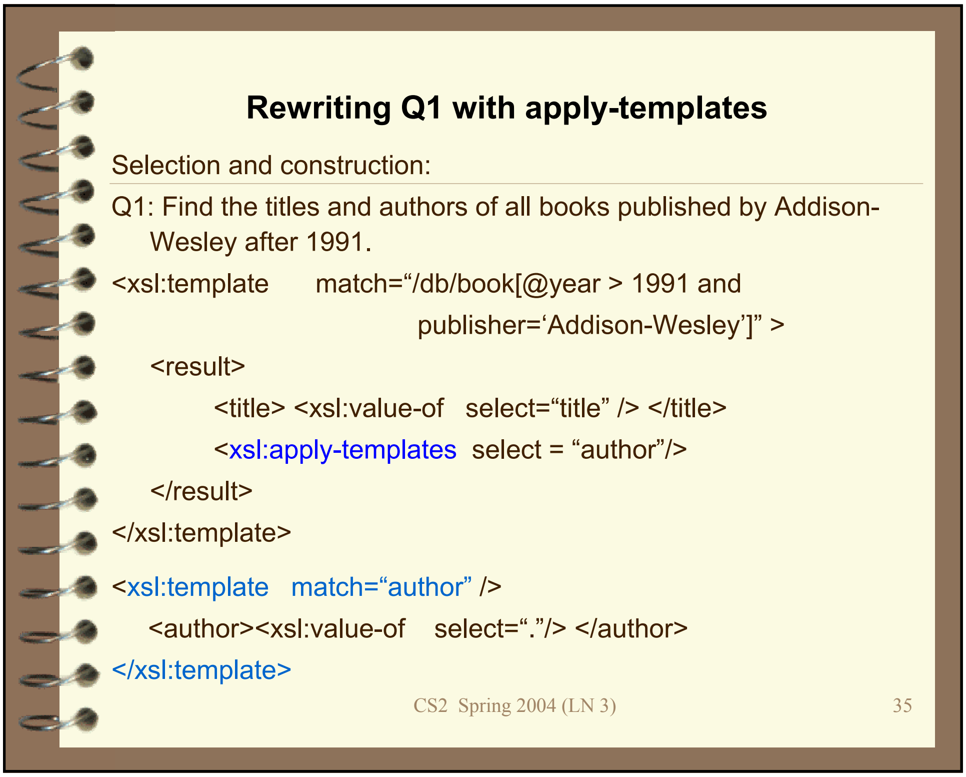
Apply templates

Recursive processing:

```
  <xsl:template  match=XPath >
          . . .
          <xsl:apply-templates  select=XPath/>
          . . .
  </xsl:template>
```

- ✓ Compare each selected child (descendant) of the matched source element against the templates in your program
- ✓ If a match is found, output the template for the matched node
- ✓ One can use xsl:apply-templates instead of xsl:for-each
- ✓ If the select attribute is missing, all the children are selected
- ✓ When the match attribute is missing, the template matches every node:

```
<xsl:template> <xsl:apply-templates /> </xsl:template>
```

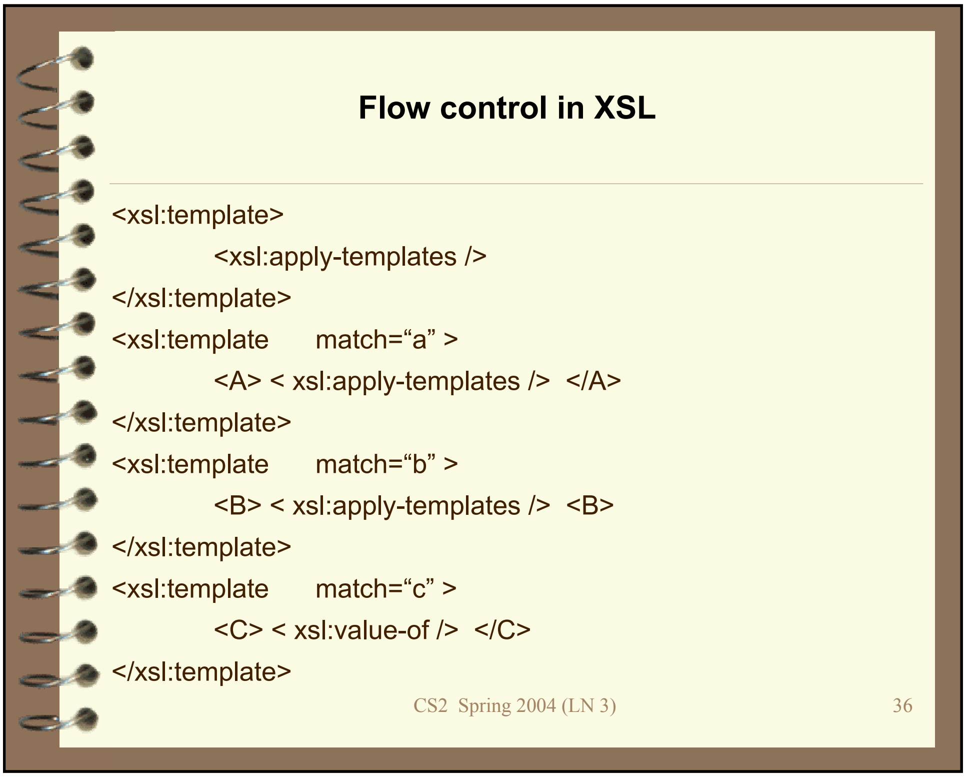


Rewriting Q1 with apply-templates

Selection and construction:

Q1: Find the titles and authors of all books published by Addison-Wesley after 1991.

```
<xsl:template      match=“/db/book[@year > 1991 and
                              publisher=‘Addison-Wesley’]” >
    <result>
          <title> <xsl:value-of   select=“title” /> </title>
          <xsl:apply-templates  select = “author”/>
    </result>
</xsl:template>
<xsl:template   match=“author” />
    <author><xsl:value-of    select=“.”/> </author>
</xsl:template>
```



Flow control in XSL

```
<xsl:template>
          <xsl:apply-templates />
</xsl:template>
<xsl:template      match=“a” >
          <A> < xsl:apply-templates />  </A>
</xsl:template>
<xsl:template      match=“b” >
          <B> < xsl:apply-templates />  <B>
</xsl:template>
<xsl:template      match=“c” >
          <C> < xsl:value-of />  </C>
</xsl:template>
```

transformation

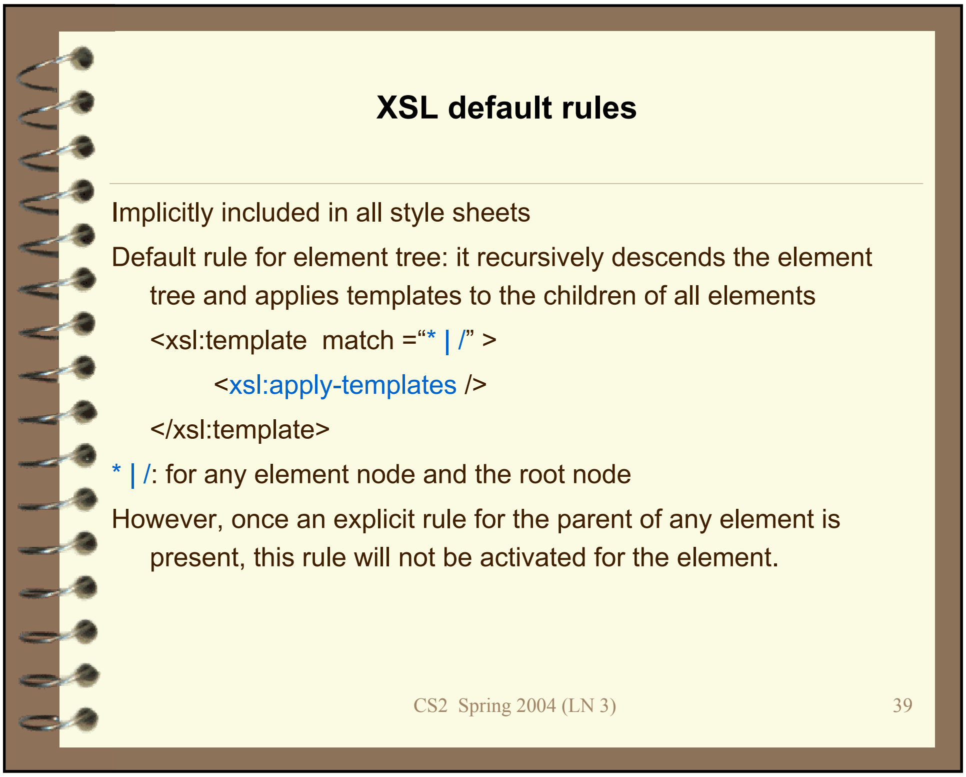
```
<a> <e> <b> <c> 1 </c>
                        <c> 2  </c>
                  </b>
          </e>
         <c>  4 </c>
</a>
⇒
<A> <B> <C> 1 </C>
                  <C> 2 </C>
         </B>
         <C> 4 </C>
</A>
```

Divergence

XSL program may not terminate.

Add the following to the previous program:

```
<xsl:template      match="e" >
          < xsl:apply-templates   select="/" />
</xsl:template>
```



XSL default rules

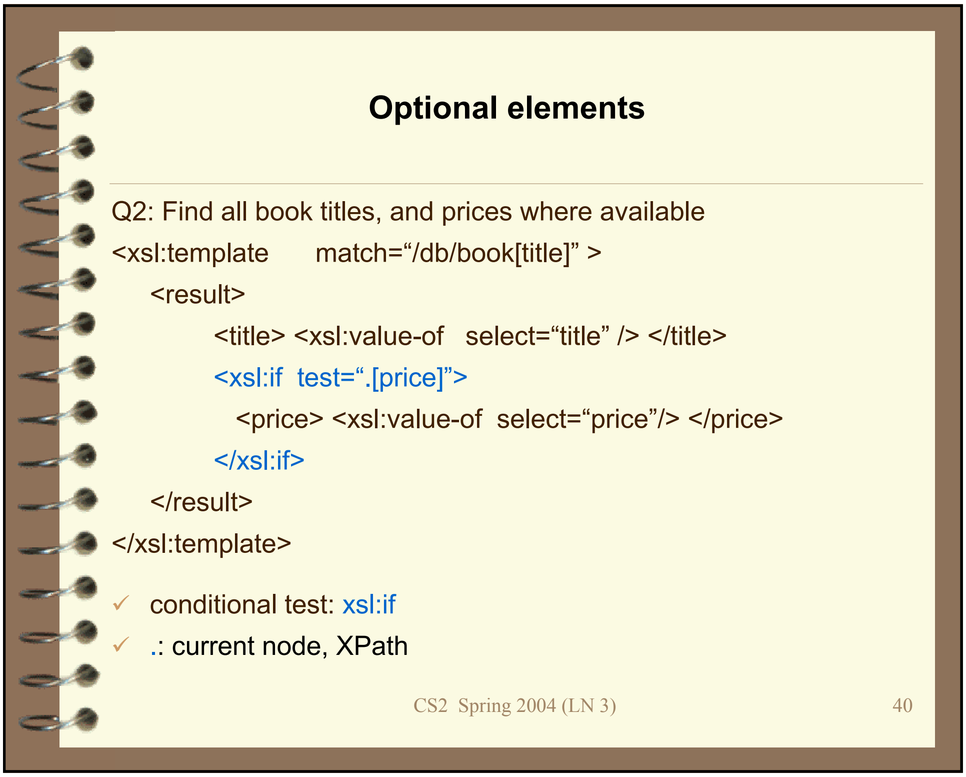
Implicitly included in all style sheets

Default rule for element tree: it recursively descends the element tree and applies templates to the children of all elements

```
<xsl:template  match =“* | /” >
        <xsl:apply-templates />
</xsl:template>
```

* | /: for any element node and the root node

However, once an explicit rule for the parent of any element is present, this rule will not be activated for the element.



Optional elements

Q2: Find all book titles, and prices where available

```
<xsl:template      match=“/db/book[title]” >
     <result>
             <title> <xsl:value-of   select=“title” /> </title>
             <xsl:if  test=“.[price]”>
                <price> <xsl:value-of  select=“price”/> </price>
             </xsl:if>
     </result>
</xsl:template>
```

- ✓ conditional test: xsl:if
- ✓ .: current node, XPath

indexing

Q3: for each book, find its title and its first two authors, and returns <et-al/> if there are more than two authors

```
<xsl:template      match="/db/book" >
    <result>
          <title> <xsl:value-of   select="title" /> </title>
          <xsl:apply-templates  select="author" />
     </result>
</xsl:template>
<xsl:template      match="author[position( ) < 2]" >
    <author> <xsl:value-of  /> </author>
</xsl:template>
<xsl:template      match="author[position( ) = 2]" >       <et-al />
</xsl:template>
```

sorting

Q4: find the titles of all books published by Addison-Wesley after 1991, and list them alphabetically.

```
<xsl:template      match="/db/book[@year > 1991 and
                            publisher='Addison-Wesley']" >
          <title> <xsl:value-of   select="title" /> </title>
          <xsl:apply-templates>
                    <xsl:sort   select="title" />
          </xsl:apply-templates>
</xsl:template>
```

- ✓ Key: title
- ✓ xsl:sort: used together with xsl:for-each or xsl:apply-templates

XML to HTML: display

Q5: generate a HTML document consisting of the titles and authors of all books.

```
<xsl:template   match="/">
    <html>
        <head> <title> Books  </title>   </head>
        <body> <ul> <xsl:apply-templates select="/db/book "></ul></body>
    </html>
</xsl:template>
<xsl:template    match="book">
    <li>  <b> <xsl:value-of   select="title" />,  </b>
        <xsl:for-each select="author" /> <em><xsl:value-of /> </em>
        </xsl:for-each> <br>
    </li>
</xsl:template>
```

Summary and Review

Query languages for XML

- ✓ XPath: navigating an XML tree
- ✓ XSLT: XML transformations – can be used as a query language
- ✓ XQuery: XML query language

Very powerful (as opposed to relational algebra); however, query processing/optimization is hard – open issue!

Homework: study tutorials

- ✓ XPath: http://www.zvon.org/xxl/XPathTutorial/General/examples.html
- ✓ XSLT: http://www.zvon.org/xxl/XSLTutorial/Output/index.html
- ✓ XQuery: http://www.w3.org/TR/xquery-use-cases/