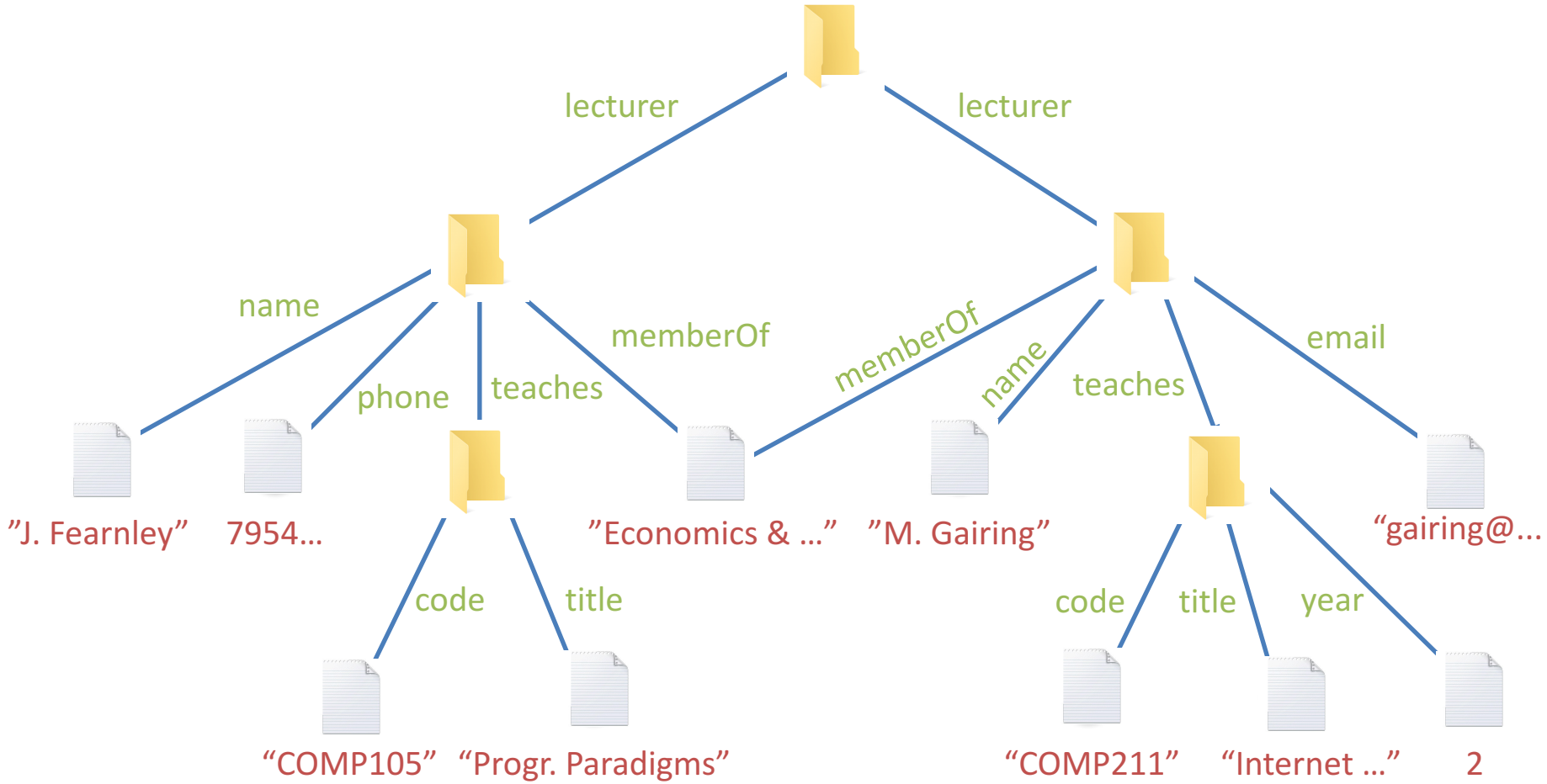
COMP207
Database Development

Lecture 22

Beyond Relational Data:
Querying XML Using Xpath and XQuery

Example

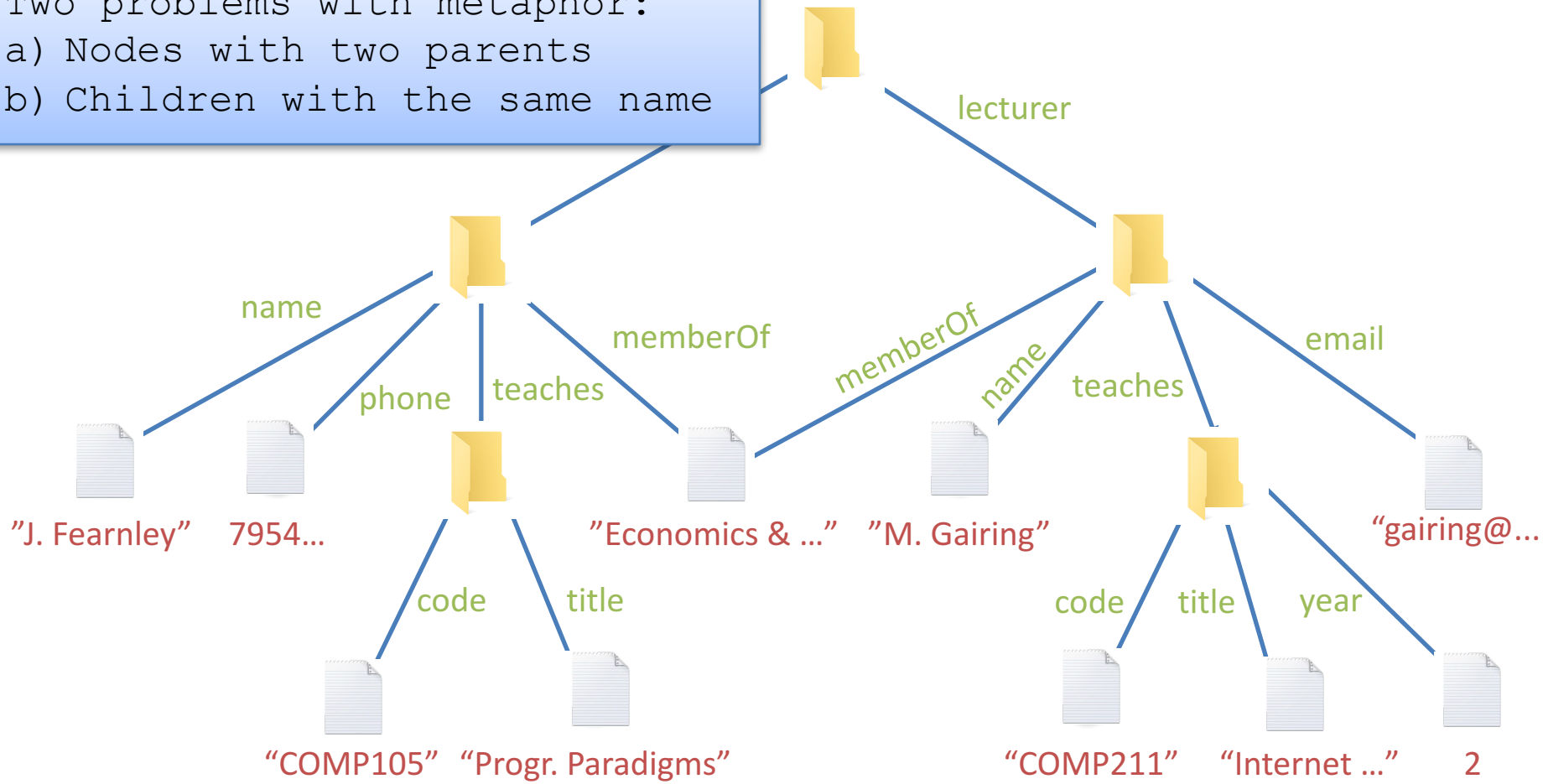
(from last time)



Example

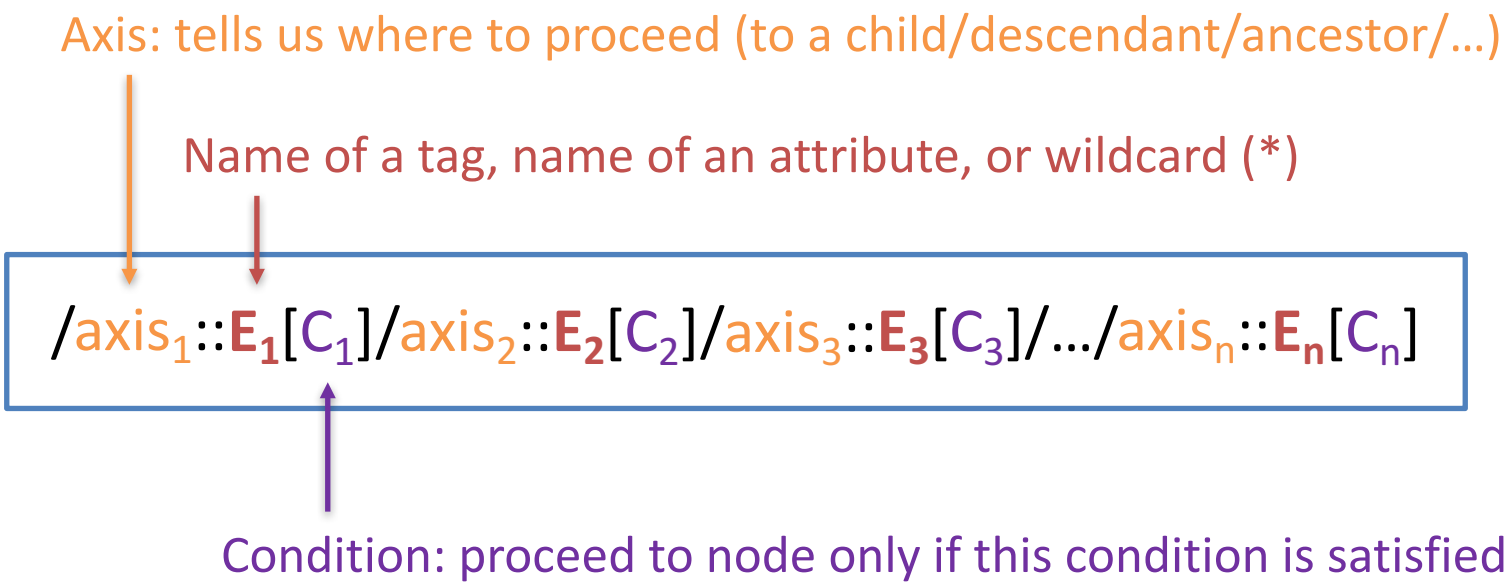
(from last time)

Two problems with metaphor:
a) Nodes with two parents
b) Children with the same name



XPath Expressions (Review)

- Path expression:



- Selects all nodes (or values) that can be reached by following the path

Navigation Axes

- An axis determines the next item on the path:

If $axis_i$ is…	then E_i is the name of…
attribute	an attribute
child	any child
descendant	any proper descendant
descendant-or-self	any descendant
ancestor	any proper ancestor
following-sibling	any sibling to the right
preceding-sibling	any sibling to the left

@ is a shorthand for ”attribute::”

Default, ”child::” can be omitted

Instead of /descendant-or-self:E we write //E

Navigation Axes

- An axis determines the next item on the path:

If $axis_i$ is…	then E_i is the name of…
attribute	an attribute
child	any child
descendant	any proper descendant
descendant-or-self	any descendant
ancestor	any proper ancestor
following-sibling	any sibling to the right
preceding-sibling	any sibling to the left
parent	the parent

@ is a shorthand for ”attribute::”

Default, ”child::” can be omitted

Instead of /descendant-or-self:E we write //E

.. is a shorthand for ”parent::”

Navigation Axes

- An axis determines the next item on the path:

If $axis_i$ is…	then E_i is the name of…
attribute	an attribute
child	any child
descendant	any proper descendant
descendant-or-self	any descendant
ancestor	any proper ancestor
following-sibling	any sibling to the right
preceding-sibling	any sibling to the left
parent	the parent

@ is a shorthand for ”attribute::”

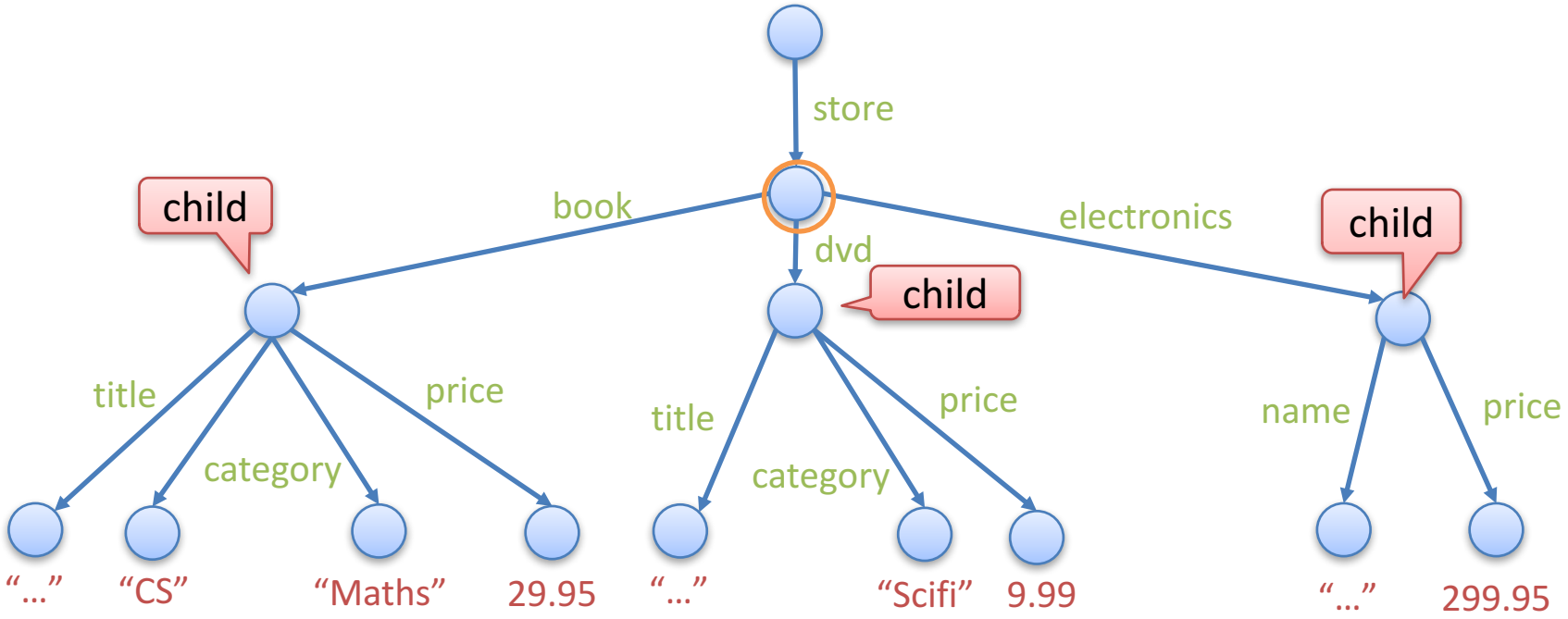
Default, ”child::” can be omitted

Instead of /descendant-or-self:E we write //E

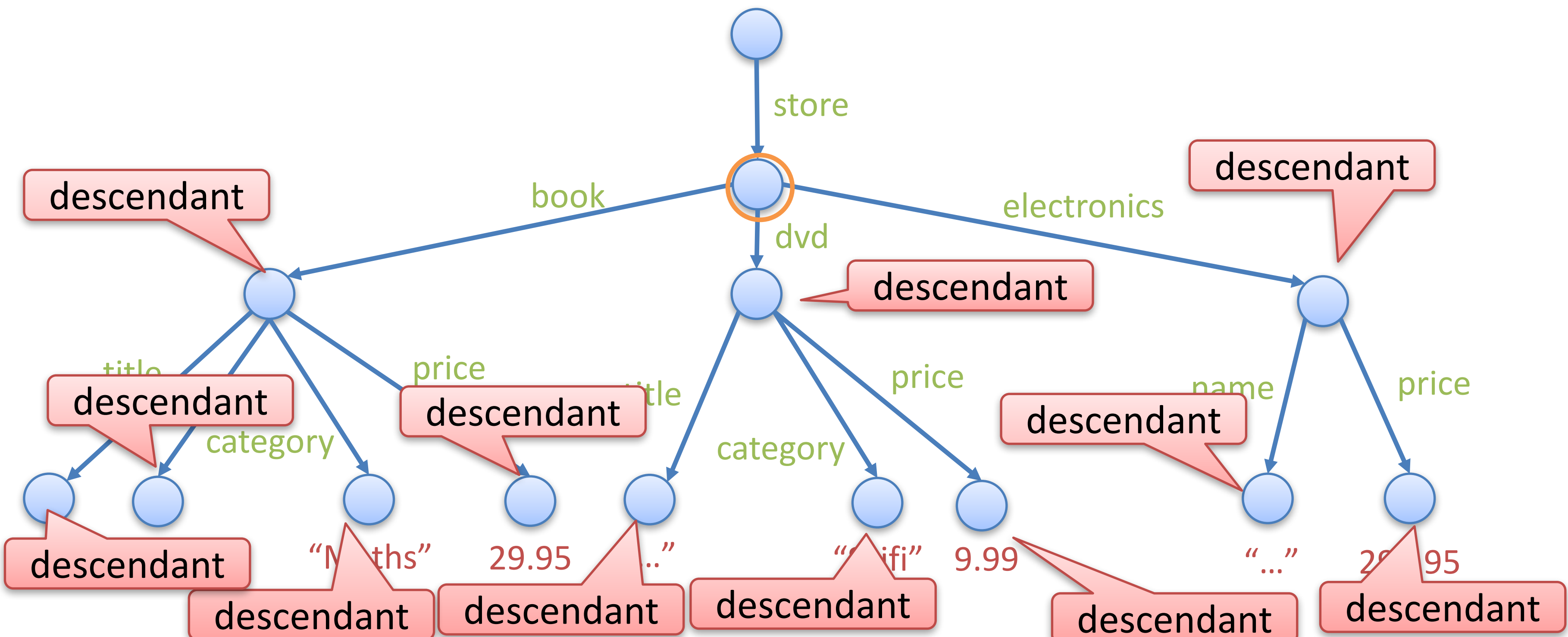
.. is a shorthand for ”parent::”

- Red axes: Reverse axes (i.e. nodes are ordered in reverse of document order)

Definition


store
child
book
dvd
child
electronics
child
title
category
price
title
category
price
name
price
"..."
"CS"
"Maths"
29.95
"..."
"Scifi"
9.99
"..."
299.95

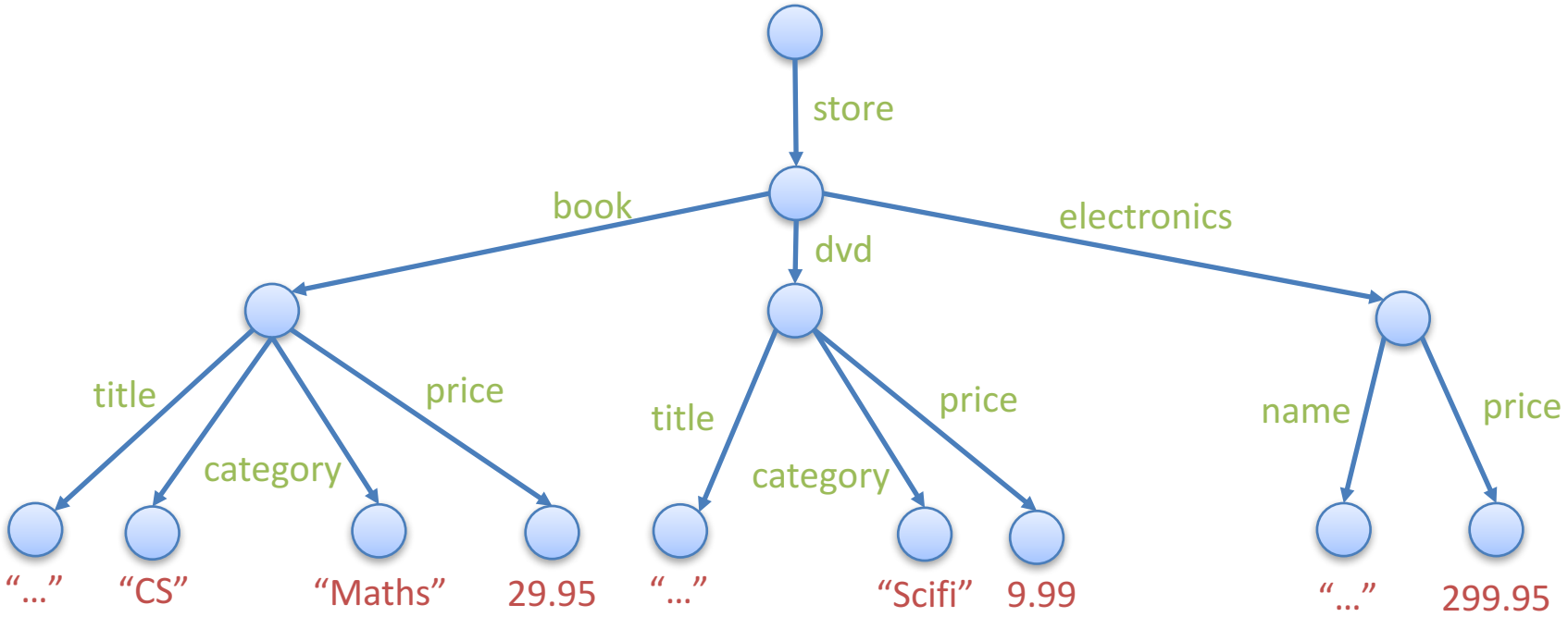
Definition



- Child is 1 outgoing arrow away
- Descendant is any number of outgoing arrows away (but at least 1)
- x is the parent of y ⇔ y is the child of x
- x is the ancestor of y ⇔ y is the descendant of x

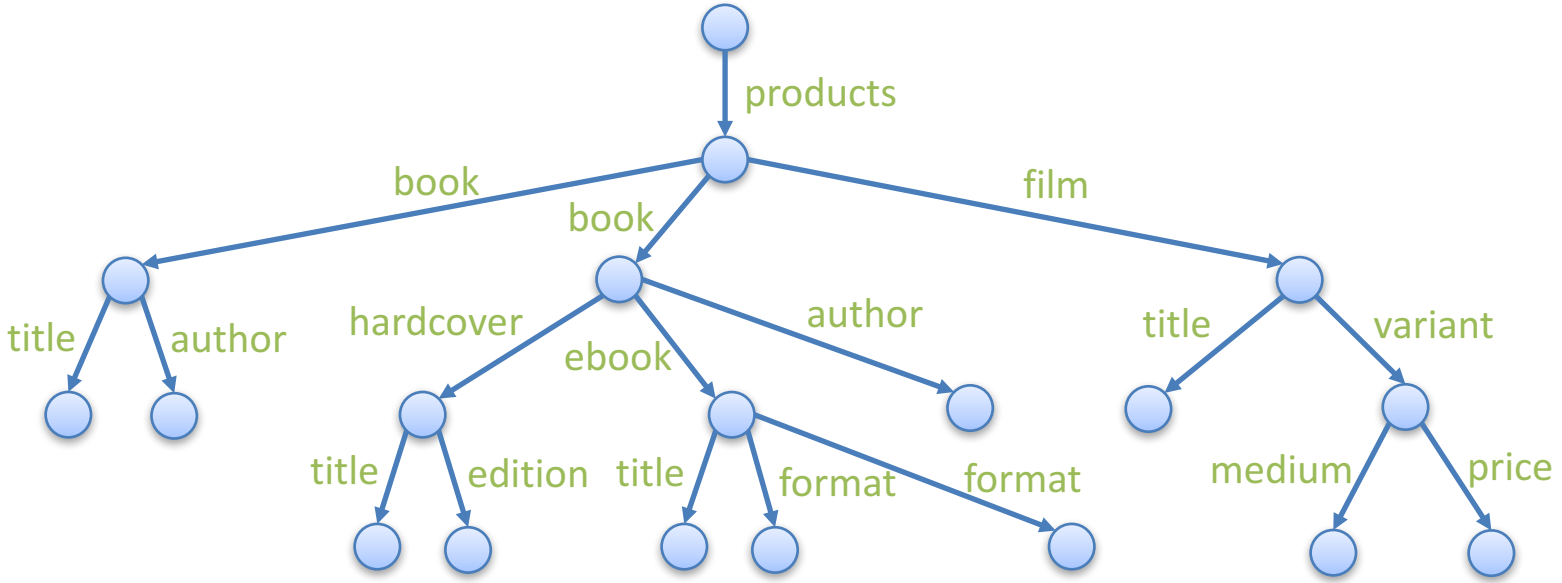
Example

(from previous lecture)



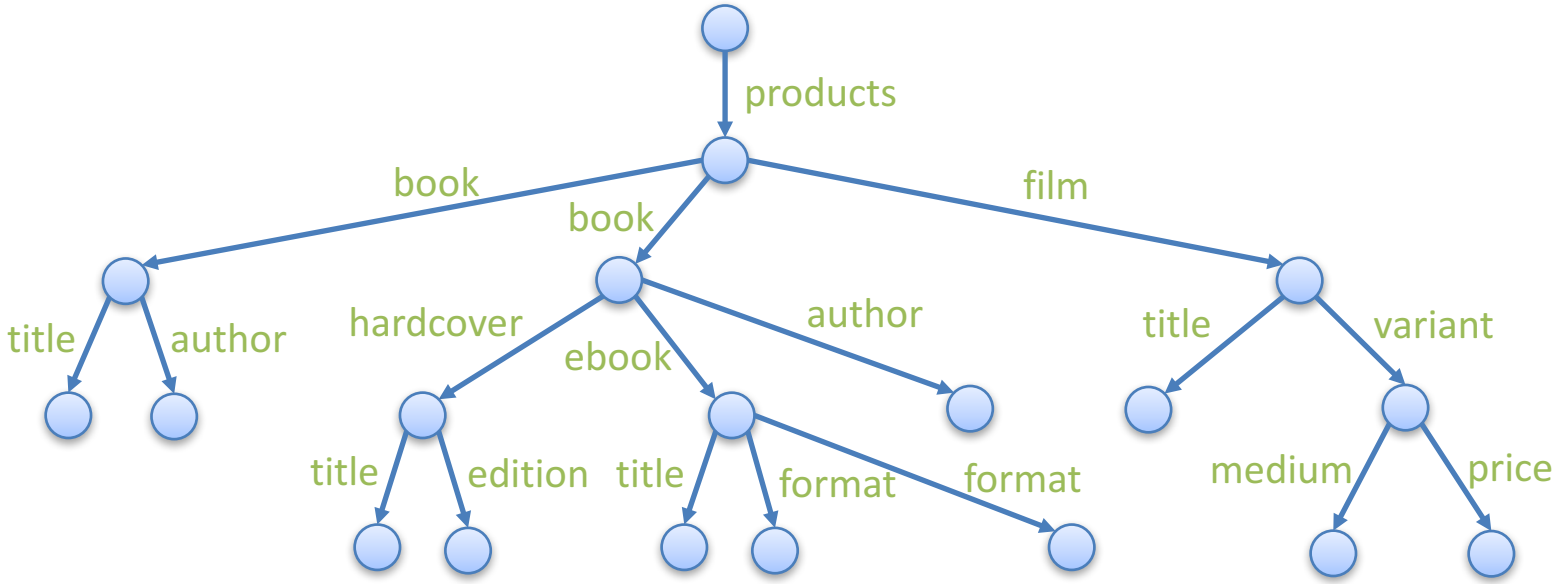
- //book[category="CS"]/title
  All titles of books in category "CS"
- //*[(category="CS" or category="Scifi") and price <= 30]
  All products in category "CS" or "Scifi", with a price of at most £30

Exercise (5 min)



- Write XPath expressions that:
  - return the titles of all books
  - return the authors of all books who have an ebook with format “epub”
  - return the authors of all “epub” ebooks with title “Databases”

Possible Solutions



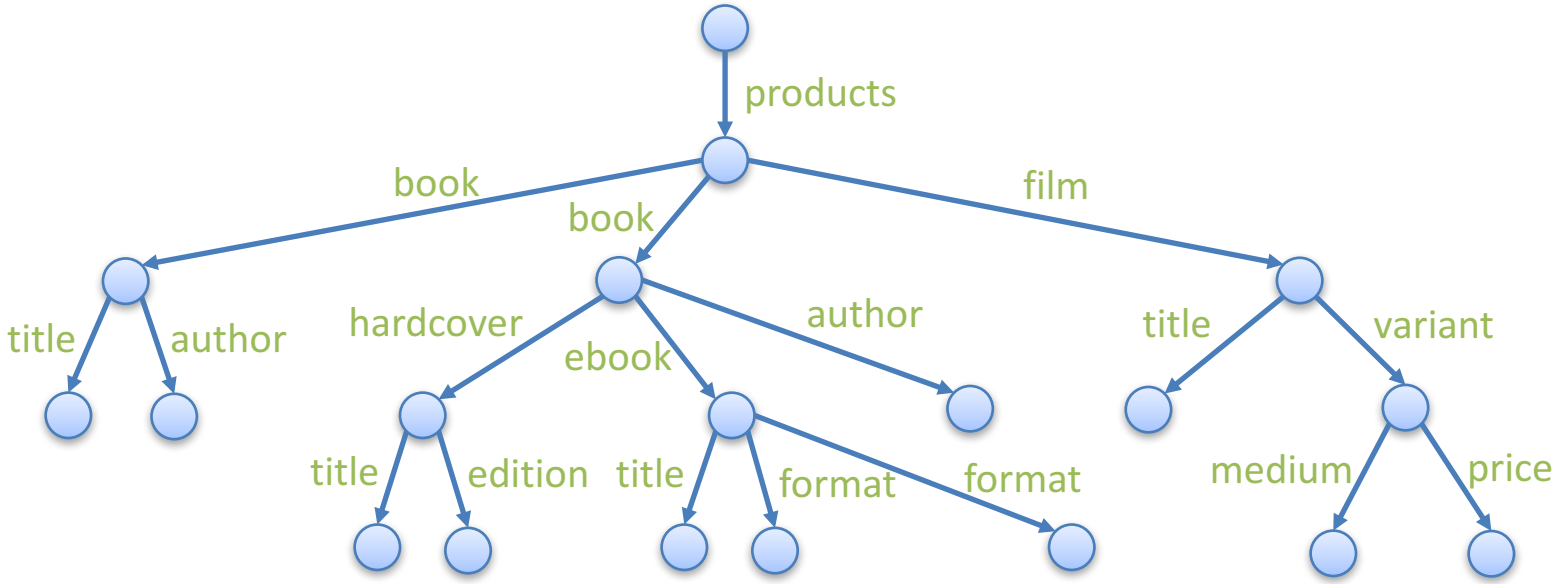
- Return the titles of all books:

  /products/book//title

- Return the authors of all books who have an ebook with format “epub”:

  /products/book[ebook/format="epub"]/author

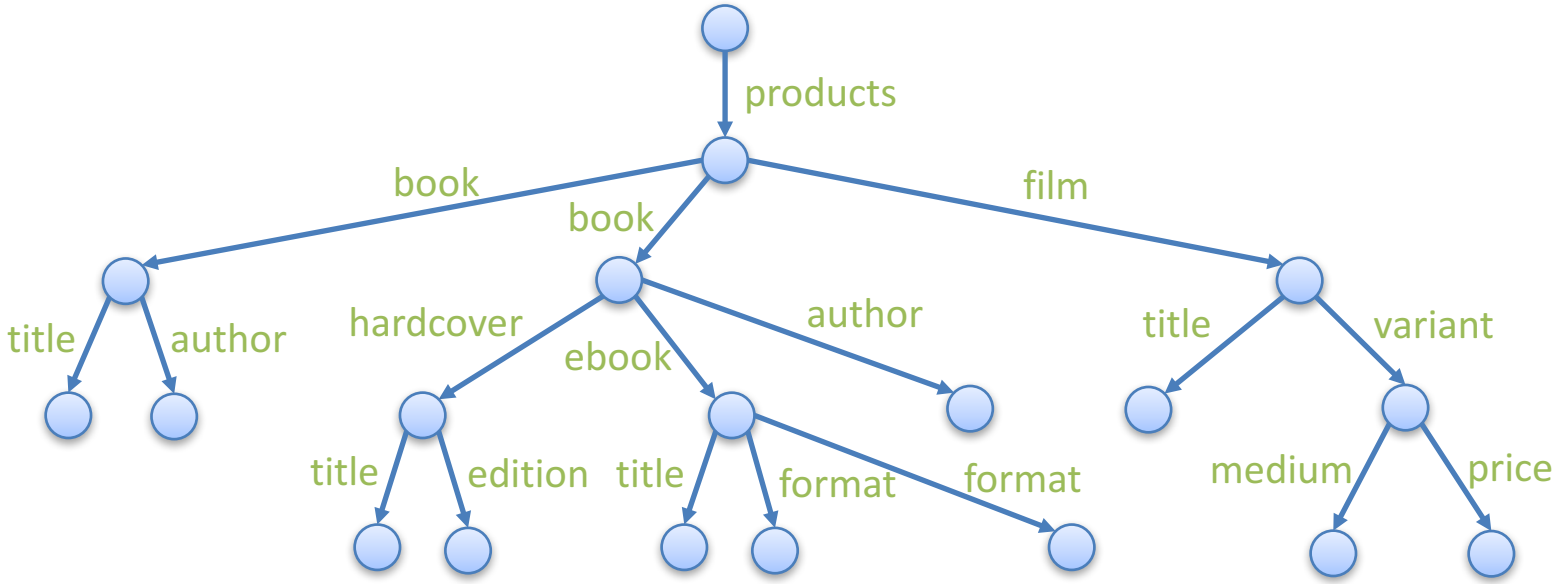
Possible Solutions 2



- Return the authors of all “epub” ebooks with title “Databases”

```
/products
    /book[ebook/format = "epub" and ebook/title="Databases"]
        /author
```

Possible Solutions 2



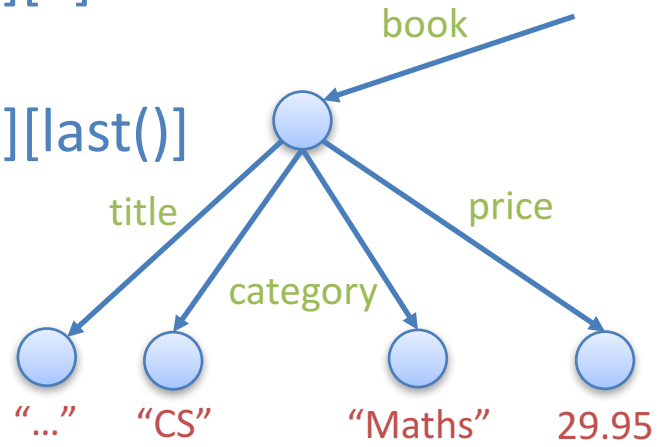
- Return the authors of all “epub” ebooks with title “Databases”

```
/products
    /book/ebook[format = "epub" and title="Databases"]/..
        /author
```

More Conditions

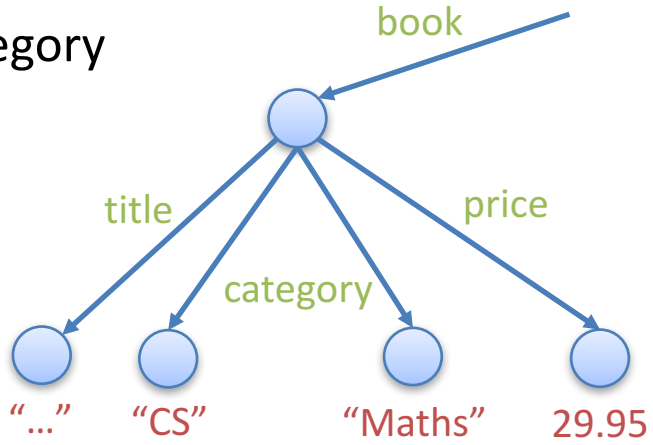
Integer

- **…E[*i*]**: true if current node is the *i*-th E-child of its parent
  - //*[category="CS" or category="Scifi"][1]
    Returns the first item in “CS” or ”Scifi”
  - //*[category="CS" or category="Scifi"][last()]
    Returns the last item in “CS” or “Scifi”
  - //book[category[1]="CS"]/title
    Returns the title of each book
    whose first category is “CS”
  - //book[1]/following-sibling::*[1]
    finds the item after the first book in document order
  - //*[.="Maths"]/preceding-sibling::*[1]
    finds the element preceding an element containing “Maths”

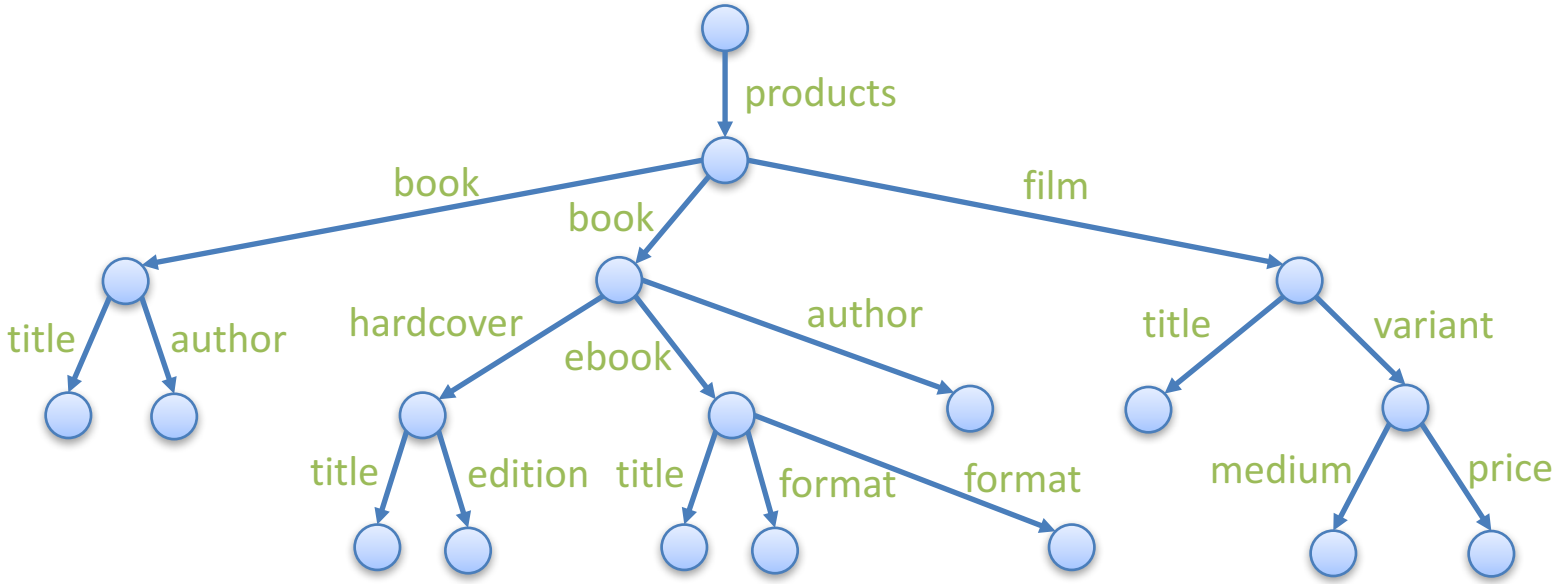


Boolean Conversions

- **…E[xpath]**: true iff the xpath gives a non-empty node-set
  - //books[category]/title
    Returns the prices of all books that has a category
  - //book[@price]
    returns a book if it has a price attribute
- **…E[string]**: true iff string is not ””



Possible Solutions 2



- Return the authors of all “epub” ebooks with title “Databases”

```
/products
    /book[ebook[format = "epub" and title="Databases"]]
        /author
```

“Real” XPath

- We’ve only covered the basics of XPath
- XPath has many more features
  - Datatypes
  - Text nodes (to extract the text enclosed in leaf elements)
  - Other node tests
  - Built-in functions to perform arithmetic, operations on strings, etc.
  - …
- More information:
  - https://www.w3.org/TR/xpath-31/

The Role of XPath

- Not used as a query language itself
  - Used for other languages
  - Examples: XQuery, XMLSchema, …
- Purpose: **select items** in an XML document
- Compare with attributes in relational databases, which select items in relations

```
SELECT S.name, E.module_code
FROM Students S, EnrolledIn E
WHERE S.sid = E.sid;
```

/students/student/name

/enrolledIn/module/code

XQuery

XQuery

- Extension of XPath by SQL-like features
  - Every XPath expression is an XQuery expression
- More general XQueries: FLWR expressions

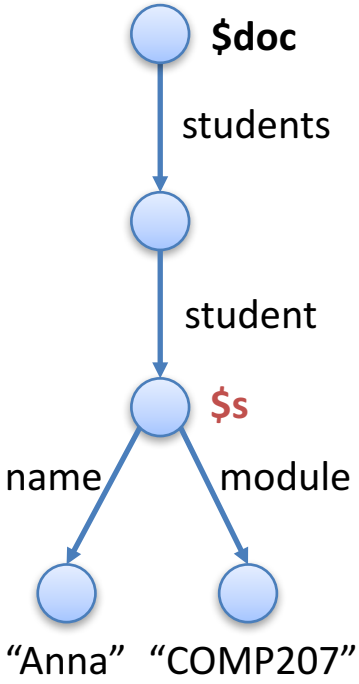
Let clause — let **$doc** := doc("students.xml")

For clause — for $s in **$doc**/students/student

Where clause — where $s/module = "COMP207"

Return clause — return $s/name

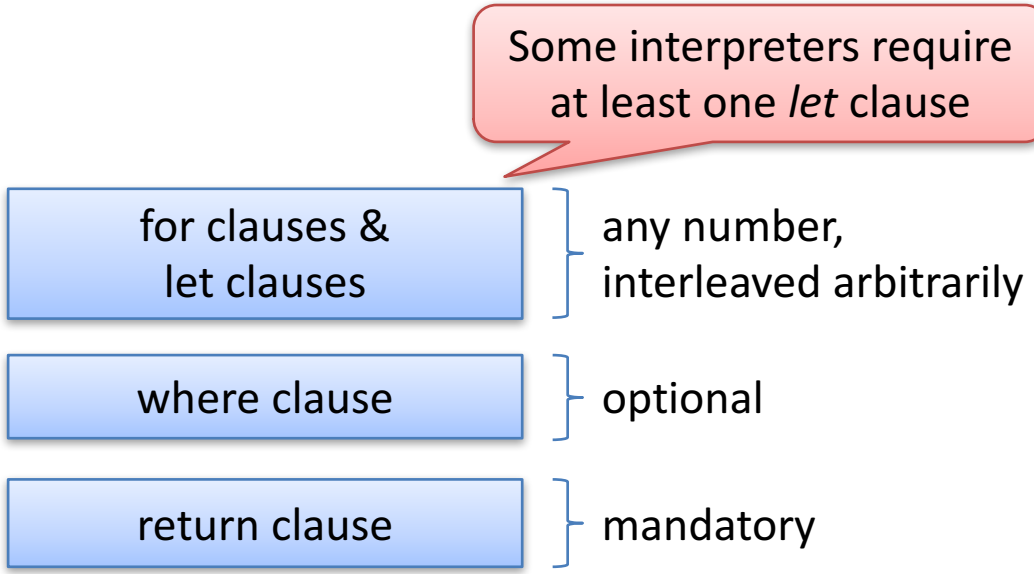
Case sensitive!



- Return lists of values/nodes … document-order

FLWR Expressions

- Structure:



- Extreme case:

```
return <greeting>Hello World!</greeting>
```

```
let $hello := <greeting>Hello World!</greeting>
return $hello
```

Let Clauses

let **variable** := **XQuery expression**

- Assigns the result of **XQuery expression** to **variable**
- Examples:
  - let $doc := doc("students.xml")
  - let $student_names := $doc/students/student/name
- Variable names:
  - Start with $ (e.g., $doc, $x, $student)

doc("students.xml") is an XPath that returns the document node of “students.xml”

For Clauses

for **variable** in **XQuery expression**

- Execution:
  - Consider each item in the result of **XQuery expression** in turn (same order as in the result)
  - For each item, assign it to **variable** & execute whatever follows the *for* clause
- Examples:
  - for $s in $doc/students/student
  - for $name in $doc//student[module="COMP207"]/name

Where Clauses

where **condition 1**, **condition 2**, …

- Execution:
  - Evaluate all the conditions
  - If the conditions are true, then execute the return clause
- Conditions
  - e.g. comparison between XPath and constant
- Example:
  - where $s/module = “COMP207”
  - where $s/year > 2

Existential semantics again!

Summary

- A number of languages have been proposed and defined for processing XML
- XPath: allows us to select items
- XQuery: extends XPath by SQL-like features
  - FLWR expressions (today)
  - Many more features
- Try it out: various XQuery processors available
  - Online, as command line tools, or as libraries for various programming languages
  - E.g., Zorba (http://www.zorba.io)