COMP207
Database Development

Lecture 21

Beyond Relational Data:
Querying XML Using XPath

Example

- We will represent the following relational database by an XML document:

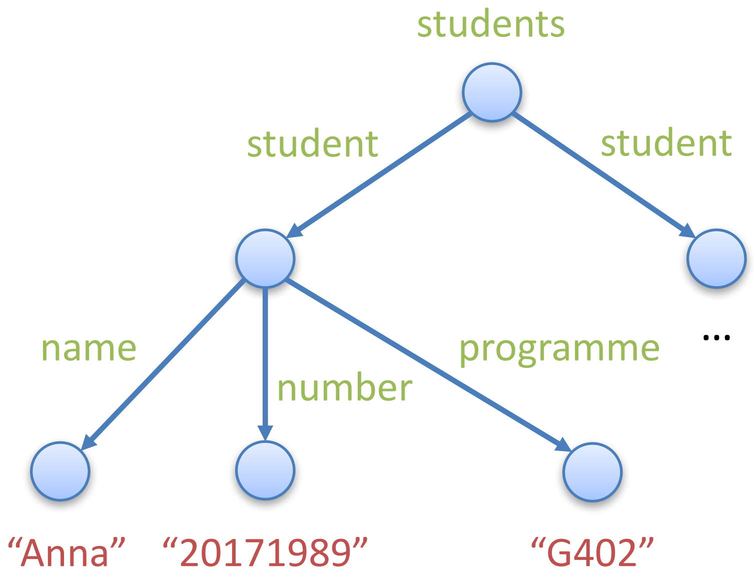
Student

name	number	programme
Anna	20171989	G402
John	20174378	G702
…	…	…

- Bonus: will add a DTD or XML Schema so that the following are in 1-to-1 correspondence
  – XML documents conforming to the DTD or XML Schema
  – Relational databases with the above schema

Possible Solution

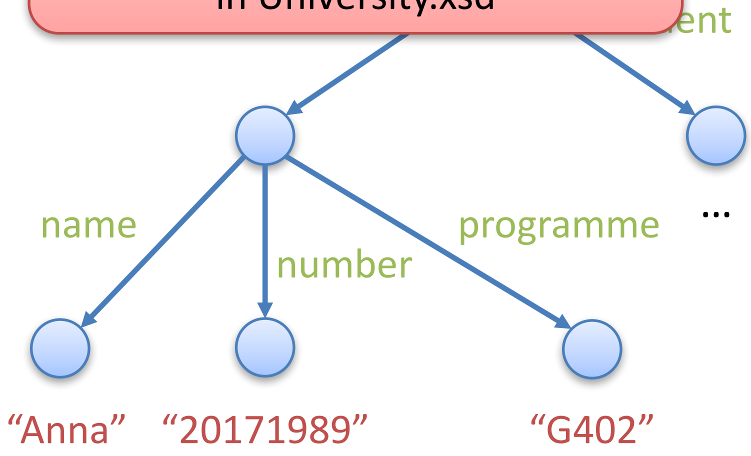
```
<?xml version="1.0" standalone="no">
<!DOCTYPE students [
   <!ELEMENT students (student*)>
   <!ELEMENT student (name, number, programme)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT number (#PCDATA)>
   <!ELEMENT programme (#PCDATA)>
]>
<students>
   <student>
      <name>Anna</name>
      <number>20171989</number>
      <programme>G402</programme>
   </student>
   <student>
      <name>John</name>
      <number>20174378</number>
      <programme>G702</programme>
   </student>
   …
</students>
```



Another possible Solution

```
<?xml version=“1.0” standalone=“no”>
<u:students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation=”University University.xsd”
              xmlns:u=”University">
   <student>
      <name>Anna</name>
      <number>20171989</number>
      <programme>G402</programme>
   </student>
   <student>
      <name>John</name>
      <number>20174378</number>
      <programme>G702</programme>
   </student>
   …
</u:students>
```

Means that the University schema should be applied and it is defined in University.xsd



University.xsd

```
<? xml version="1.0" encoding="utf-8” ?>
<xs:schema xmlns: xs= “http://www.w3.org/2001/XMLSchema”
        targetNamespace=“University”>
<xs:element name = “students”>
<xs:complexType>
    <xs:sequence>
        <xs:element name = “student” type = “studentType”
                minOccurs = “0” maxOccurs = “unbounded”/>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:complexType name = “studentType”>
    <xs:sequence>
        <xs:element name=“name” type = “xs:string”/>
        <xs:element name=“number” type = “xs:string”/>
        <xs:element name=“programme” type = “xs:string”/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

Keys in XML Schema

```
<? xml version="1.0" encoding="utf-8” ?>
<xs:schema xmlns: xs= “http://www.w3.org/2001/XMLSchema”
targetNamespace=“University”>
…
<xs:element name = “lecturers”>
<xs:complexType>
     <xs:sequence>
          <xs:element name = “lecturer” type = “lecturerType”
                    minOccurs = “0” maxOccurs = “unbounded”/>
    </xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

If we want name to be a key for lecturer, we add

Keys in XML Schema

```
<? xml version="1.0" encoding="utf-8” ?>
<xs:schema xmlns: xs= “http://www.w3.org/2001/XMLSchema”
targetNamespace=“University”>
…
<xs:element name = “lecturers”>
<xs:complexType>
       <xs:sequence>
              <xs:element name = “lecturer” type = “lecturerType”
                            minOccurs = “0” maxOccurs = “unbounded”/>
      </xs:sequence>
</xs:complexType>
<xs:key name=”lecKey”>
      <xs:selector xpath=”lecturer”/>
      <xs:field xpath=”@name”/>
</xs:key>
</xs:element>
</xs:schema>
```

If we want name to be a key for lecturer, we add

@ because it is an attribute

Using keys in XML Schema

```
<? xml version="1.0" encoding="utf-8” ?>
<xs:schema xmlns: xs= “http://www.w3.org/2001/XMLSchema”
targetNamespace=“University”
elementFormDefault="qualified">
…
<xs:element name = “students”>
<xs:complexType>
     <xs:sequence>
          <xs:element name = “student” type = “studentType”
                    minOccurs = “0” maxOccurs = “unbounded”/>
    </xs:sequence>
</xs:complexType>
<xs:keyref name=”lecKeyRef” refers = “lecKey”>
    <xs:selector xpath=”student/academicAdvisor”/>
    <xs:field xpath=”@name”/>
</xs:key>
</xs:element>
</xs:schema>
```

Students have elements called academic advisors that has the name of an lecturer

Query Languages for XML

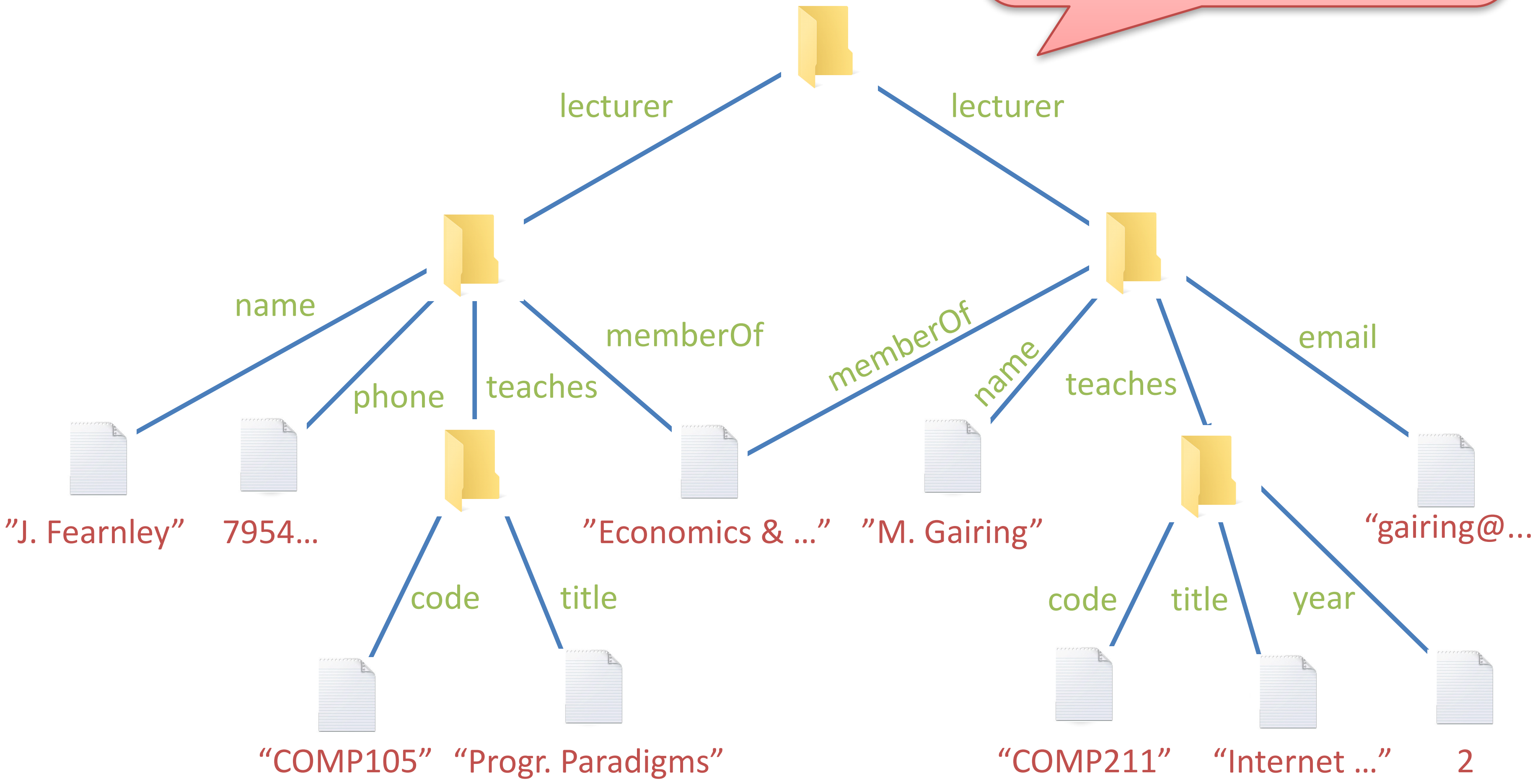
- Several defined by the W3C
- XPath ← today
  - Selects nodes (elements) from an XML document
  - Basis for other W3C standards related to XML
  - Latest version: 3.1 (March 2017)
- XQuery
  - Builds on XPath
  - Allows for more complex SQL-like queries
  - Latest version: 3.1 (March 2017)
- Related: XSLT

XPath

Example

(from last time)

There are 2(!) problems with this metaphor.
Which problems?

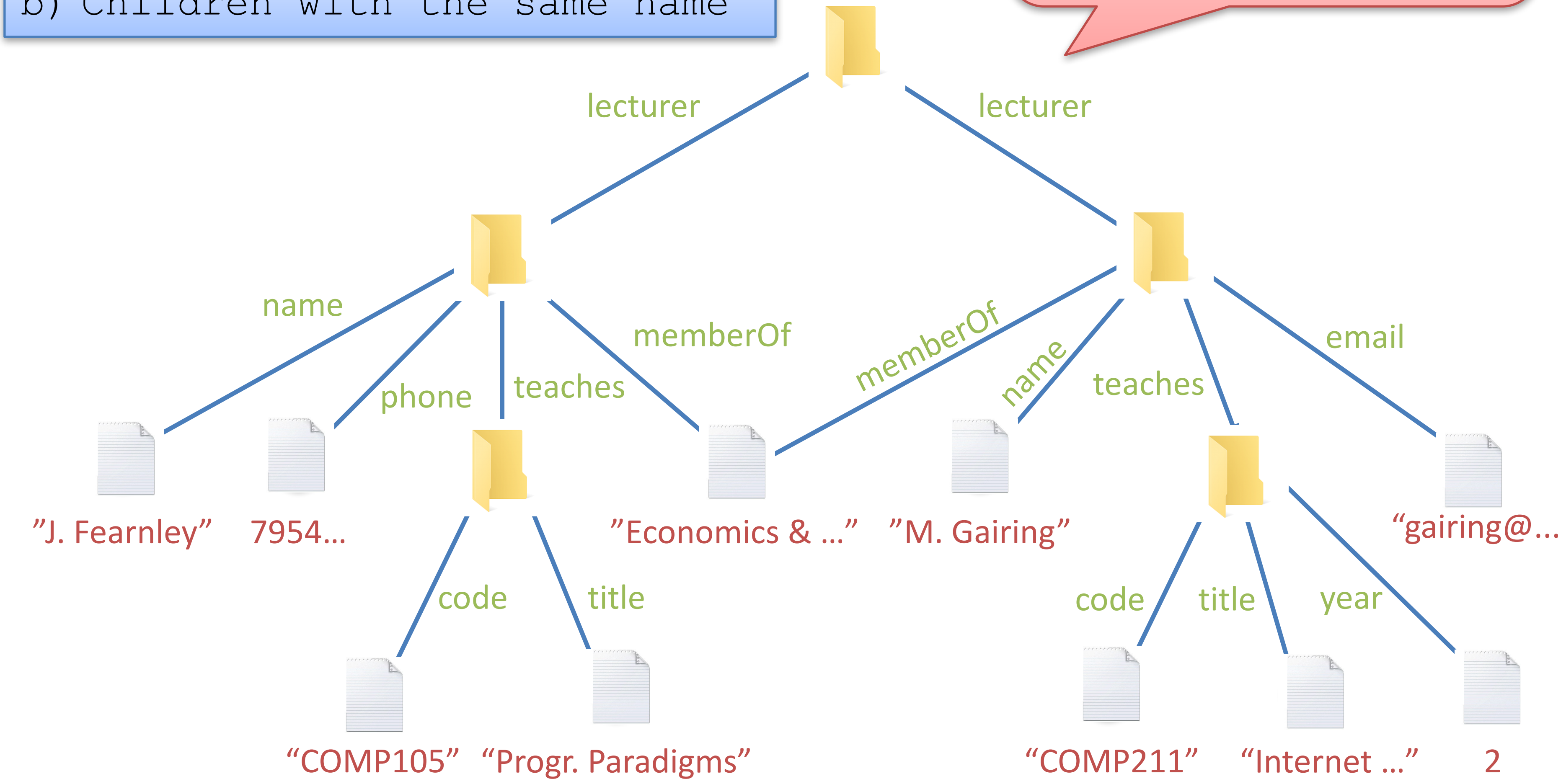


Example

ast time)

There are 2(!) problems with this metaphor.
Which problems?

```
a) Nodes with two parents
b) Children with the same name
```



Nodes with 2 parents

- For Windows

/J for folders and /H for files

```
mklink /J C:\LinkToFolder C:\Users\Name\OriginalFolder
```

- Linux/macOS

must add –s for folders

```
ln source.file link.file
```

hardlinked directory loop otherwise

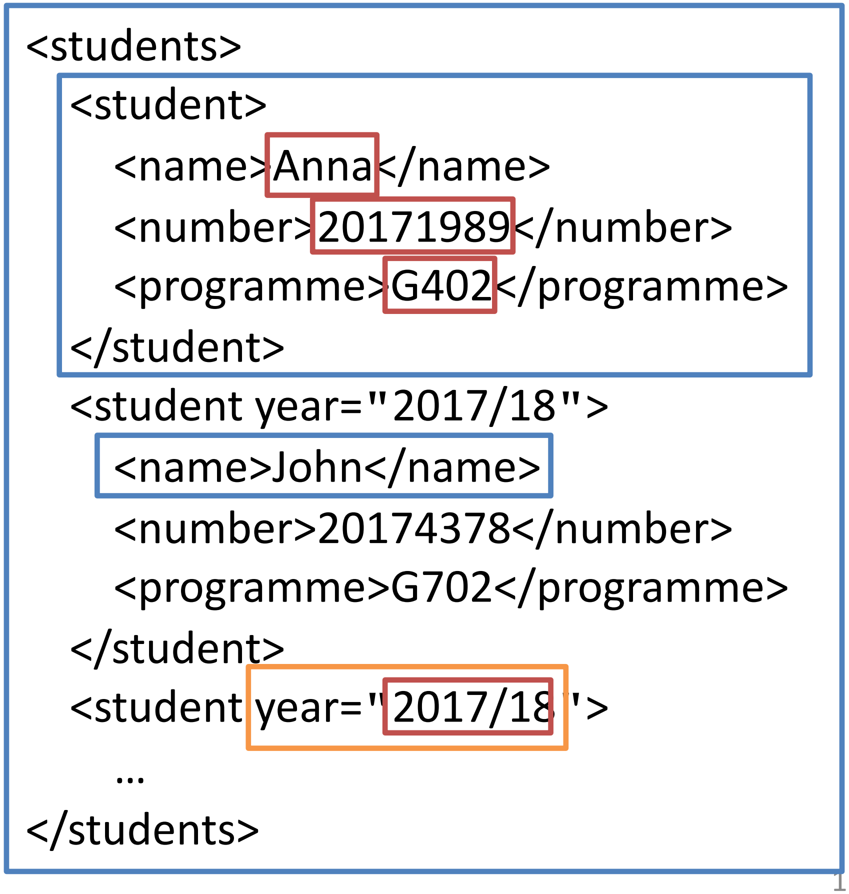
Nodes with same name

- Two simple ideas for solution:
  - Return all
  - Return *i*’th item (e.g. first or last or random or …)

- XPath does both and more
  - In essence: the main problem handled today!

General Idea

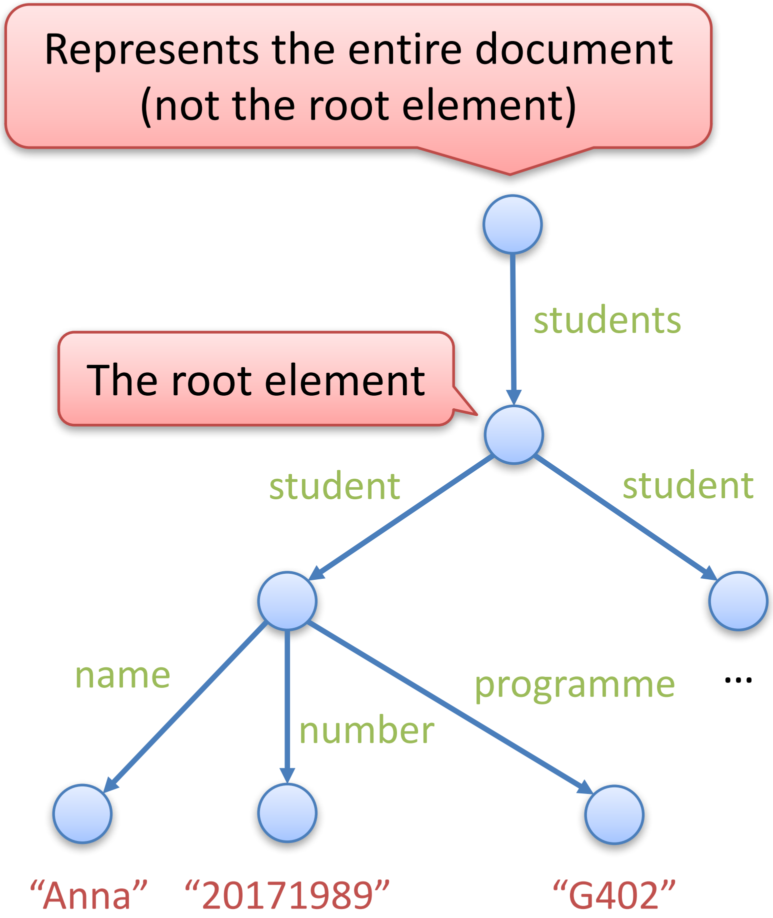
- XPath allows us to write queries that return a set of values or nodes from an XML document
- Values
  - strings, integers, reals, etc.
- Nodes
  - Document “node”:
    - Represents the entire document
    - Not the root element
  - **Element node**: any element
  - **Attributes**: found inside opening tags of elements



```
<students>
  <student>
    <name>Anna</name>
    <number>20171989</number>
    <programme>G402</programme>
  </student>
  <student year="2017/18">
    <name>John</name>
    <number>20174378</number>
    <programme>G702</programme>
  </student>
  <student year="2017/18">
    …
</students>
```

Streamlined Representation of XML

```
<students>
  <student>
    <name>Anna</name>
    <number>20171989</number>
    <programme>G402</programme>
  </student>
  <student>
    <name>John</name>
    <number>20174378</number>
    <programme>G702</programme>
  </student>
  …
</students>
```


Represents the entire document (not the root element)

The root element

Path Expressions

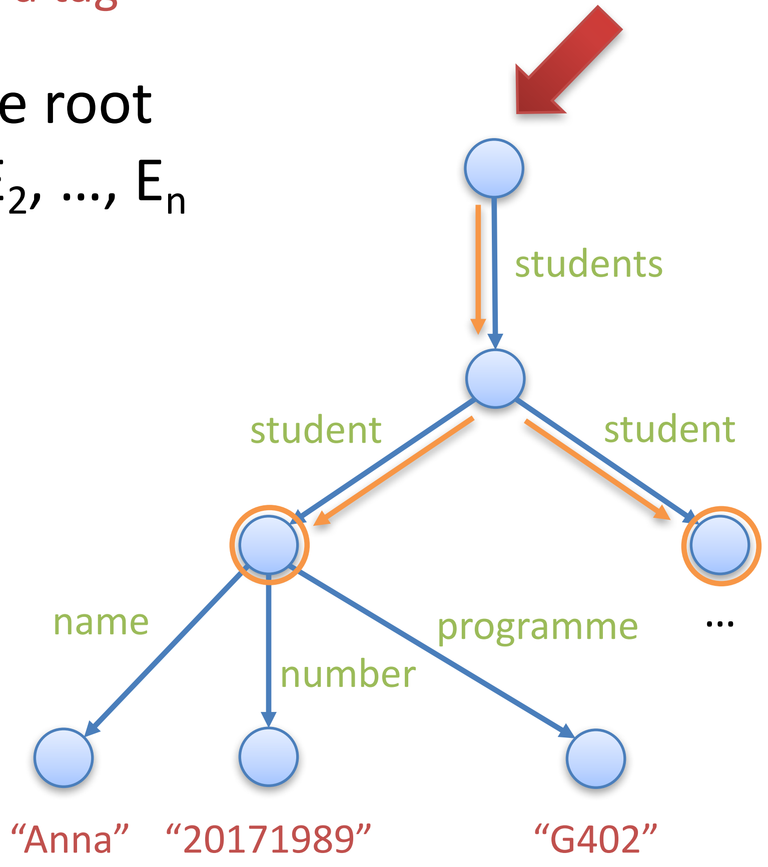
- Format: $/E_1/E_2/E_3/\ldots/E_n$

Sometimes just called an “XPath”

For the moment: name of a tag

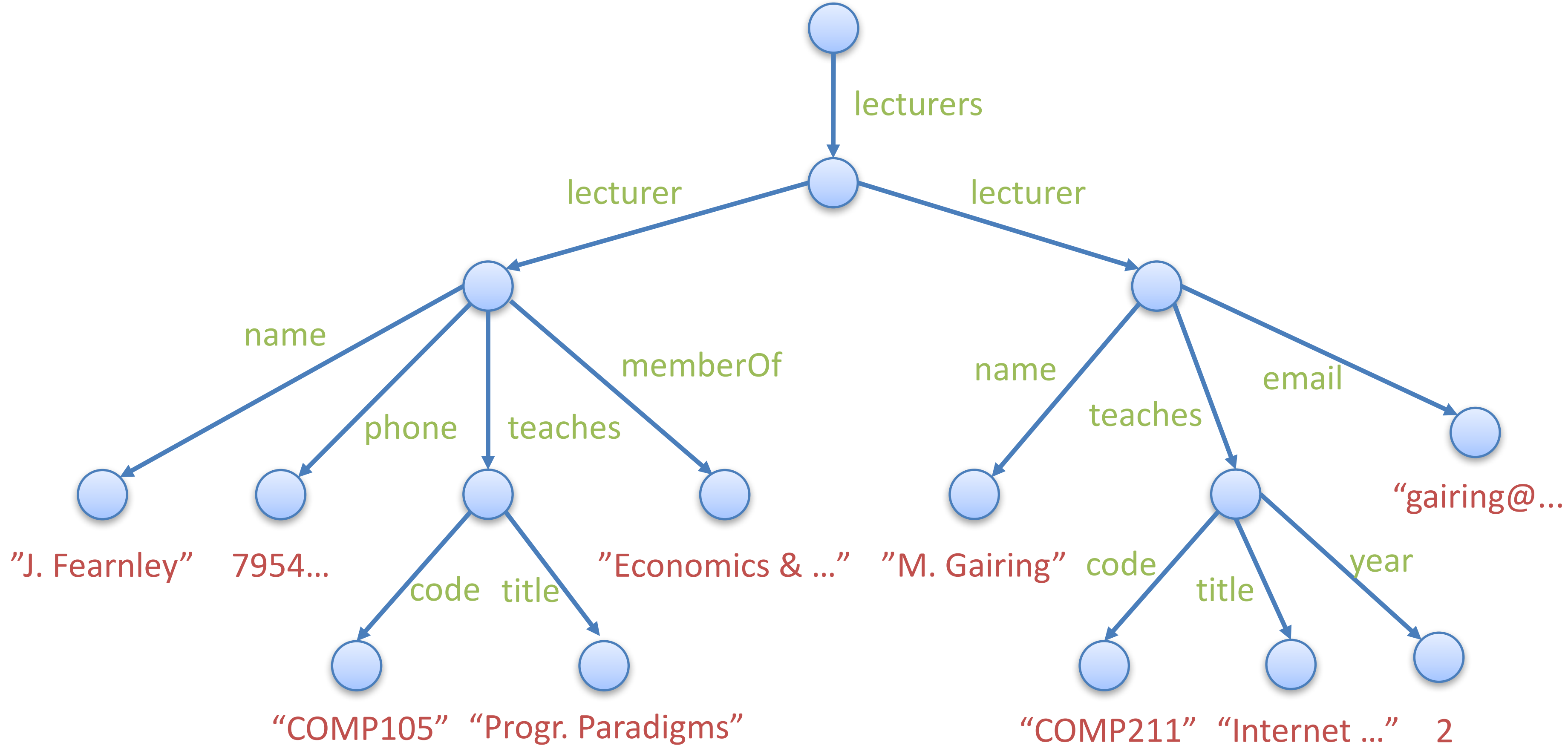
- Selects all nodes reachable from the root by following the edges labeled E_1, E_2, …, E_n
- Examples:
  - /students: selects the root element
  - /students/student: selects all student elements
- The result is returned in *document order*

Start here



Exercise (2 min)

- What is the result of /lecturers/lecturer/name?
- What is the result of /lecturers/lecturer/teaches/year?

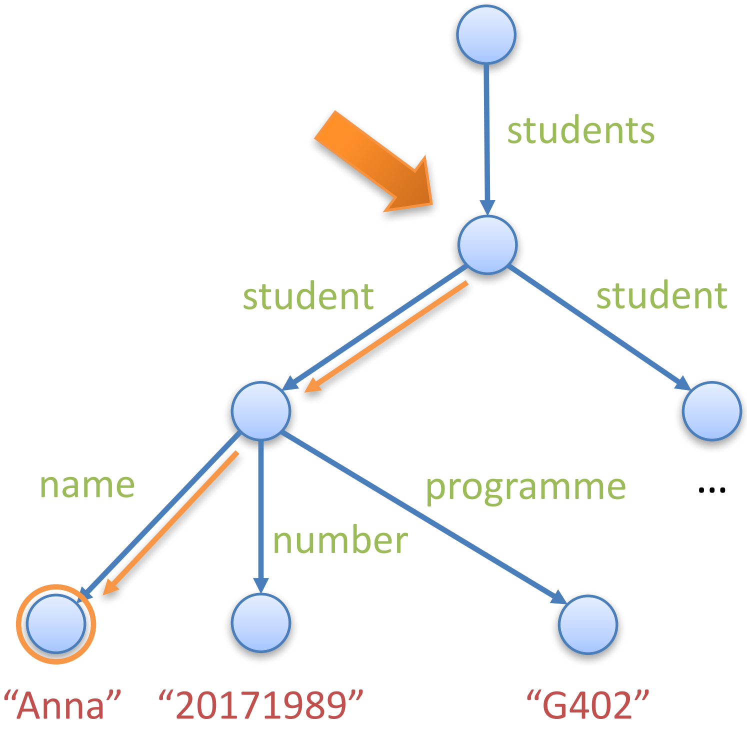


Relative Path Expressions

- Format: $E_1/E_2/E_3/\ldots/E_n$

Do not start with /

- Don't evaluate at the root, but relative to another node
- Examples:
  - student/name: all name elements of student elements below a given node
- Again, the result is returned in *document order*



Attributes

- Path expressions can be extended so that we can return attribute values

Attribute name

- Idea: replace the last tag name by @A

- Example:
  - /students/student/@name returns ”Anna”, “Ben”, “Cloe”

Does not work in Zorba – must add /data()

  - /students/student/module/@code returns “COMP207”, “COMP219”
  - student/@name

- Again: document order

```
ame="Anna" id="123">
e code="COMP207">

    </module>
  </student>
  <student name="Ben" id="456">
    <module code="COMP219">
      ...
    </module>
  </student>
  <student name="Chloe" id="789">
  </student>
</students>
```

Wildcards

- A wildcard (*) can be used to stand for any tag name or attribute name
- Example:
  - /students/student/*
    returns all elements directly below student elements
  - /students/student/module/@*
    returns all attributes of modules

Does not work in Zorba – must add /data()

```
<students>
   <student name="Anna" id="123">
      <programme code="G402"/>
      <module code="COMP207">
         ...
      </module>
   </student>
   <student name="Ben" id="456">
      <programme code="G702" />
      <module code="COMP219">
         ...
      </module>
      <email>ben@liv.ac.uk</email>
   </student>
   <student name="Chloe" id="789">
   </student>
</students>
```

Navigation Axes

- More general form of a path expression:

Attributes only at the end

$$/axis_1::E_1/axis_2::E_2/axis_3::E_3/\ldots/axis_n::E_n$$

Name of a tag, name of an attribute, or *

An axis

- 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
preceeding-sibling	any sibling to the left

@ is a shorthand for ”attribute::”

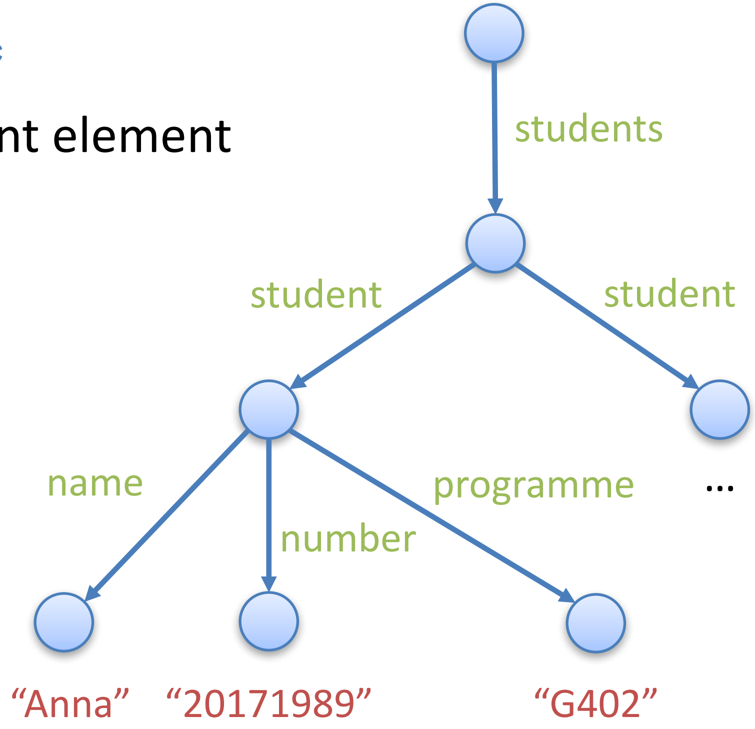
Default, ”child::” can be omitted

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

Examples

In fact, *child::* can be omitted, because it is the default

- /child::students/child::student/child::name
  represents the path /students/student/name
- /students//* or /students/descendant-or-self::*
  selects all but the document node (root of the tree)
- /descendant::name/next-sibling::*
  selects the number element of the student element
- //email
  selects all email address elements
- //module/@*
  selects all attributes of modules



Conditions

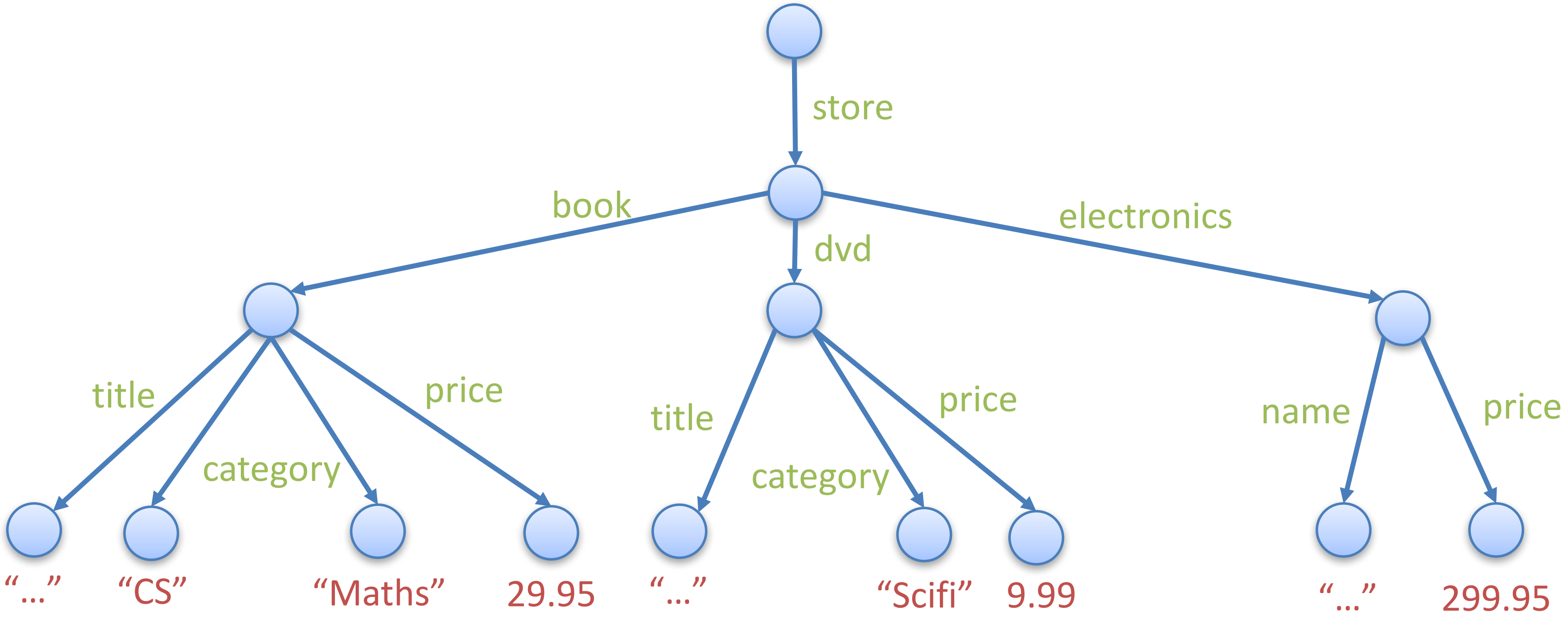
- Most general form of a path expression:

$$\boxed{/\mathrm{axis}_1{::}E_1[C_1]/\mathrm{axis}_2{::}E_2[C_2]/\mathrm{axis}_3{::}E_3[C_3]/\ldots/\mathrm{axis}_n{::}E_n[C_n]}$$

A condition (in principle, anything that can be true or false)

- Idea: if the condition is true, follow the path further
- Basic form of conditions:
  - Comparisons of two values with =, <, >, <=, >=, !=
    - A value can be a relative path expression or any constant
    - “Existential semantics”
  - Combinations of such comparisons using ’and’, ‘or’

Example



- //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

Summary

- A number of languages have been proposed and defined for processing XML
  - XPath, XQuery, …
  - XPath: foundation for several other languages
- XPath
  - Central concept: (X)Path expressions
  - Path expressions select nodes from XML documents
  - Different flavours:
    - Plain: just follow a sequence of tag names (perhaps followed by an attribute name)
    - With directions: go to child, go to any descendant, go to parent, …
    - With conditions
- Next lecture: XQuery