COMP207
Database Development

Lecture 20

Beyond Relational Data:
Semistructured Data and XML

Fully Structured Data

- Data in the relation model is **structured**
  - Data **has to fit to schema**
  - Advantage: highly efficient query processing

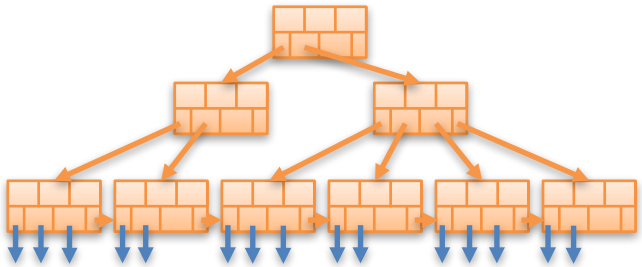
Student(name, number, programme)
Module(code, name)

Module

code	name
COMP207	Database Development
COMP219	Artificial Intelligence
…	…

Student

name	numb	
Anna	20171	
John	20174378	G702
…	…	…

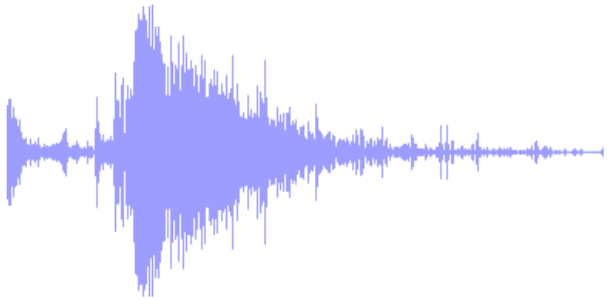


Unstructured Data

- Dropping the schema yields **unstructured data**
  - Completely free on how to organise data
  - No description of structure or data: programs have to know how to read & interpret the data

Anna	20171	
John	20174378	G702
…	…	…

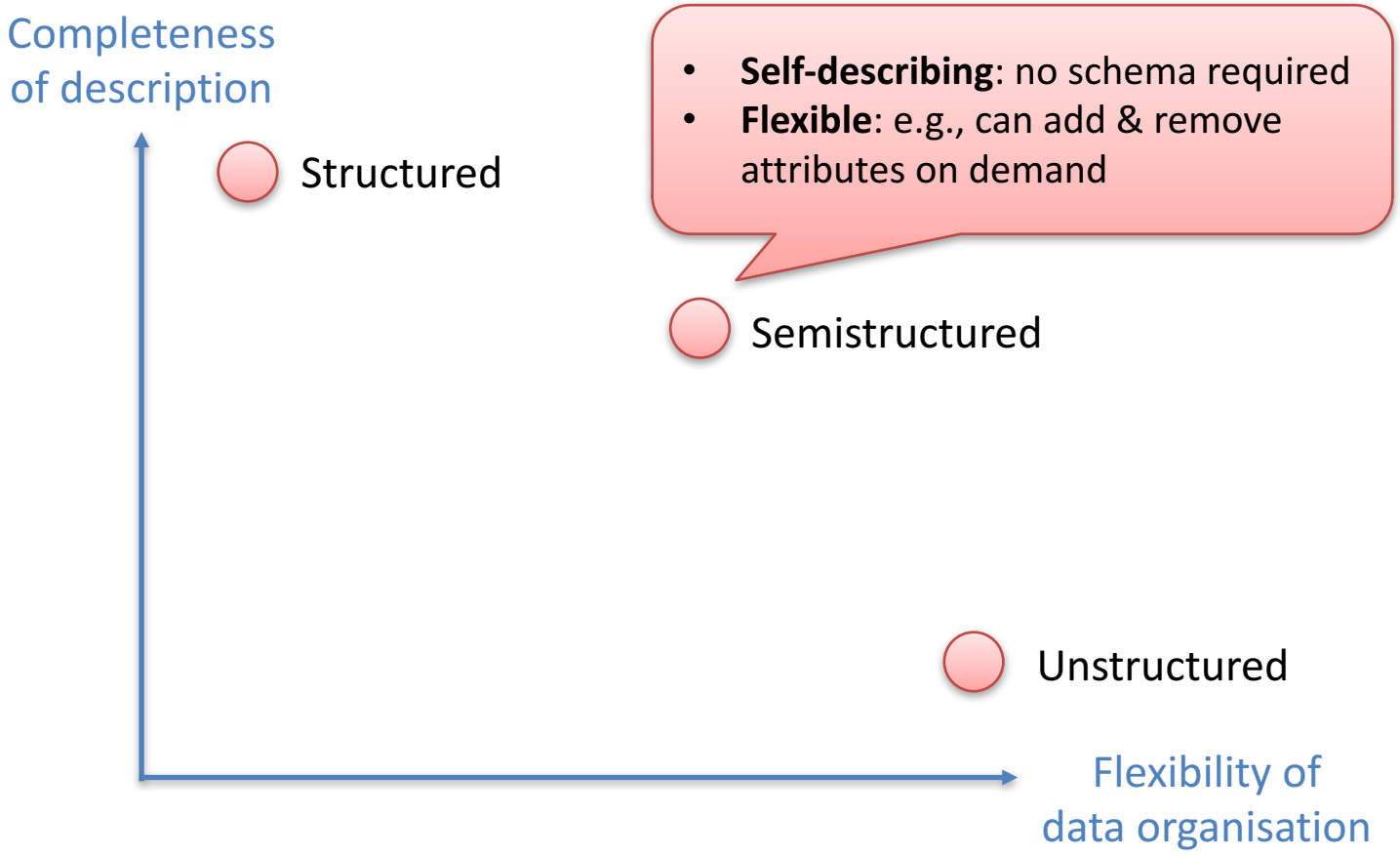
COMP207	Database Development
COMP219	Artificial Intelligence
…	…



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nos exercitat

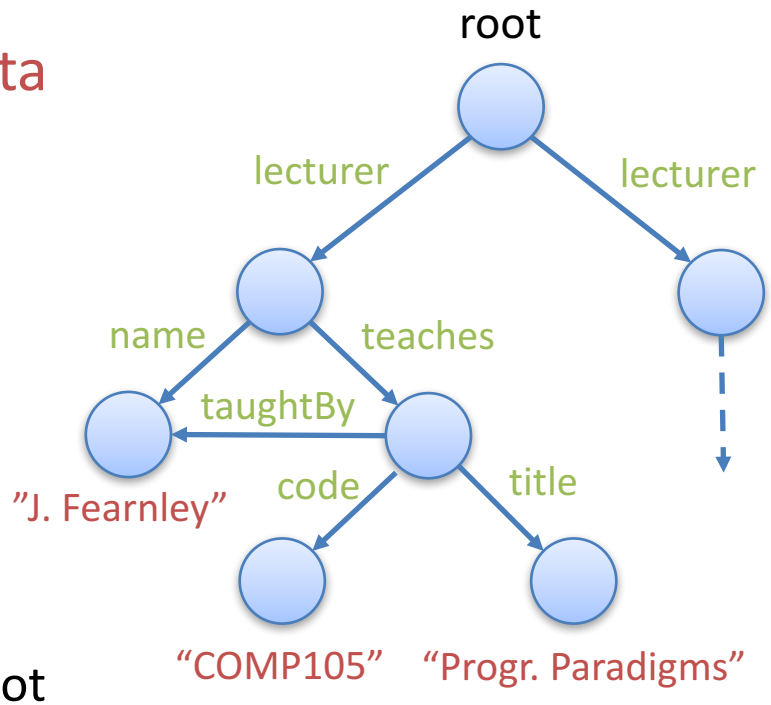
Semistructured Data

- Combines good aspects of both worlds

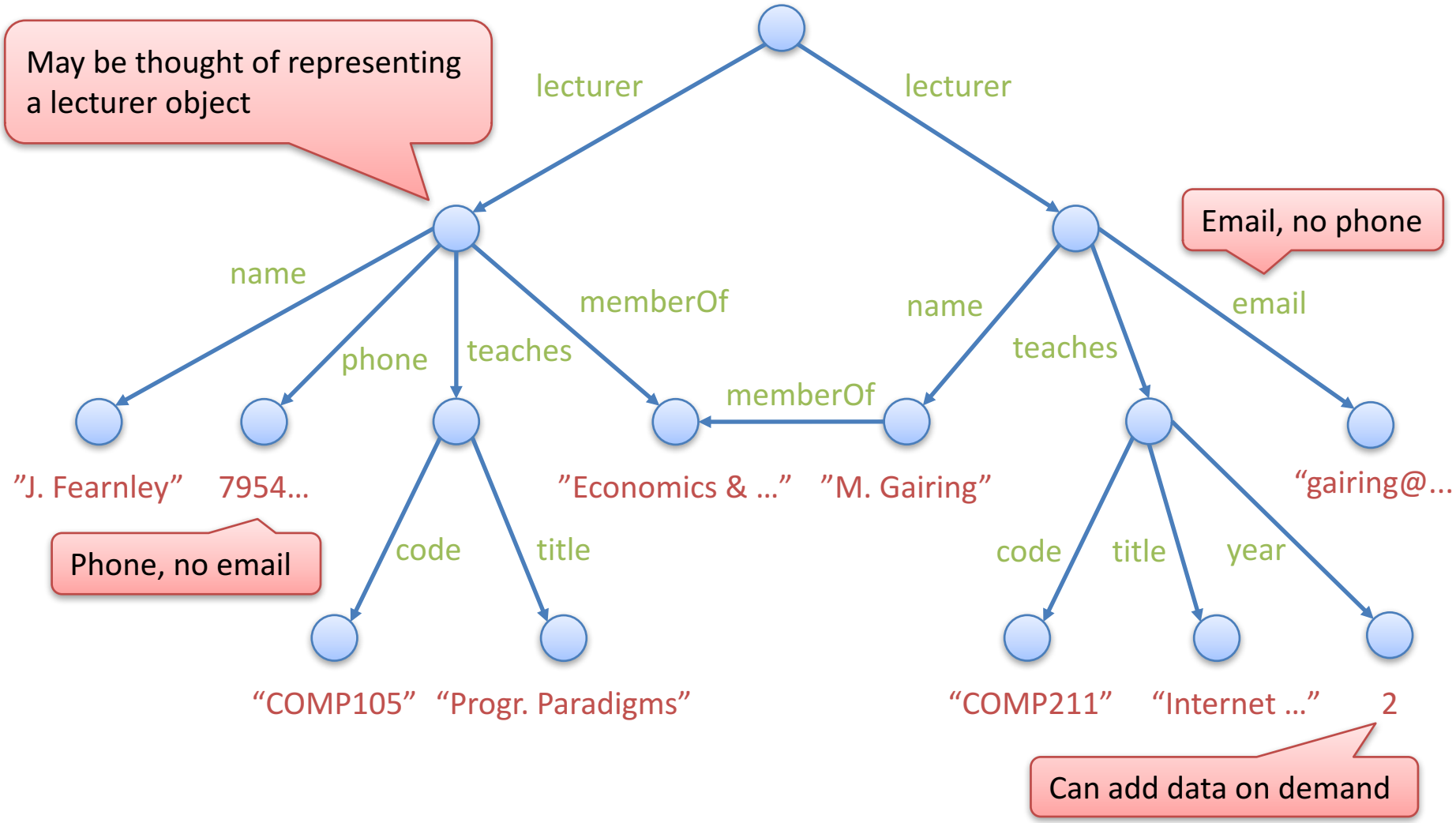


Semistructured Data Model

- Semistructured databases may be thought of as collections of nodes linked by edges
- **Leaf nodes** have associated data
  - Strings, integers, etc.
- **Inner nodes** have edges going to other nodes
  - Each edge has a label
- **Root**:
  - No incoming edges
  - Each node reachable from the root
- Typically a tree or “tree-like”



Example


May be thought of representing a lecturer object
lecturer
lecturer
name
phone
teaches
memberOf
memberOf
name
teaches
email
Email, no phone
"J. Fearnley"
7954…
"Economics & …"
"M. Gairing"
"gairing@...
Phone, no email
code
title
code
title
year
"COMP105"
"Progr. Paradigms"
"COMP211"
"Internet …"
2
Can add data on demand

Applications

- Popular form for **storing & sharing data on the web**
  - Text-based (typically)
  - Self-describing (no what is the meaning of this? big-endian vs little-endian, etc.)
  - Easy to process and manipulate, even for humans
  - Very flexible
- **Storage of documents** (word processor documents, spreadsheets, vector graphics, etc.)
- Some database systems popular in data analytics & big data use semistructured data

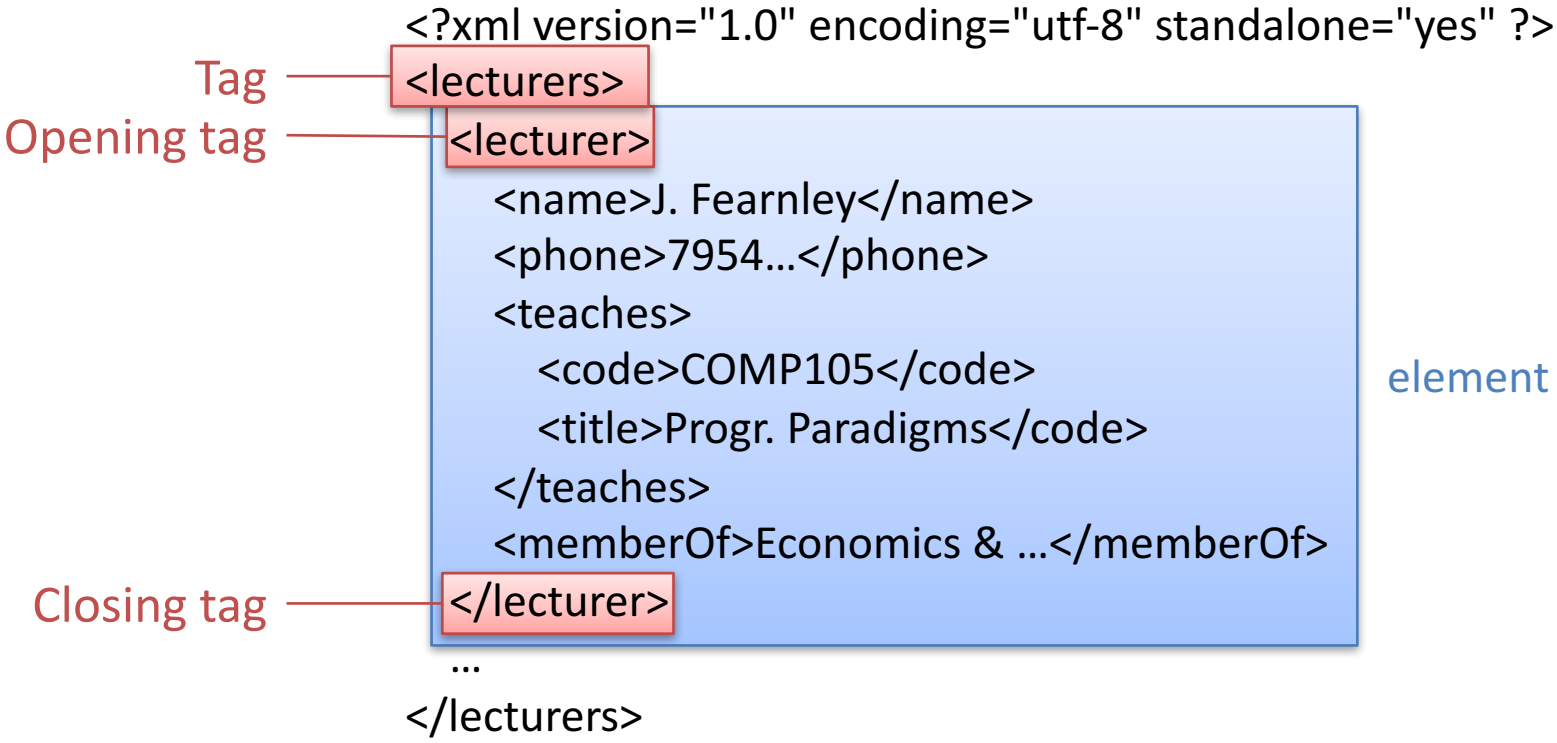
Many Forms

- **XML** ("Extensible Markup Language")  today
  - Tree-structured data
- **JSON** ("JavaScript Object Notation")
  - Similar to XML
  - Heavily used in conjunction with JavaScript
- **Graphs**
  - One of the most general forms of semistructured data
- Simple **key-value** relationships
  - Correspond to very simple XML/JSON documents

XML (“Extensible Markup Language”)

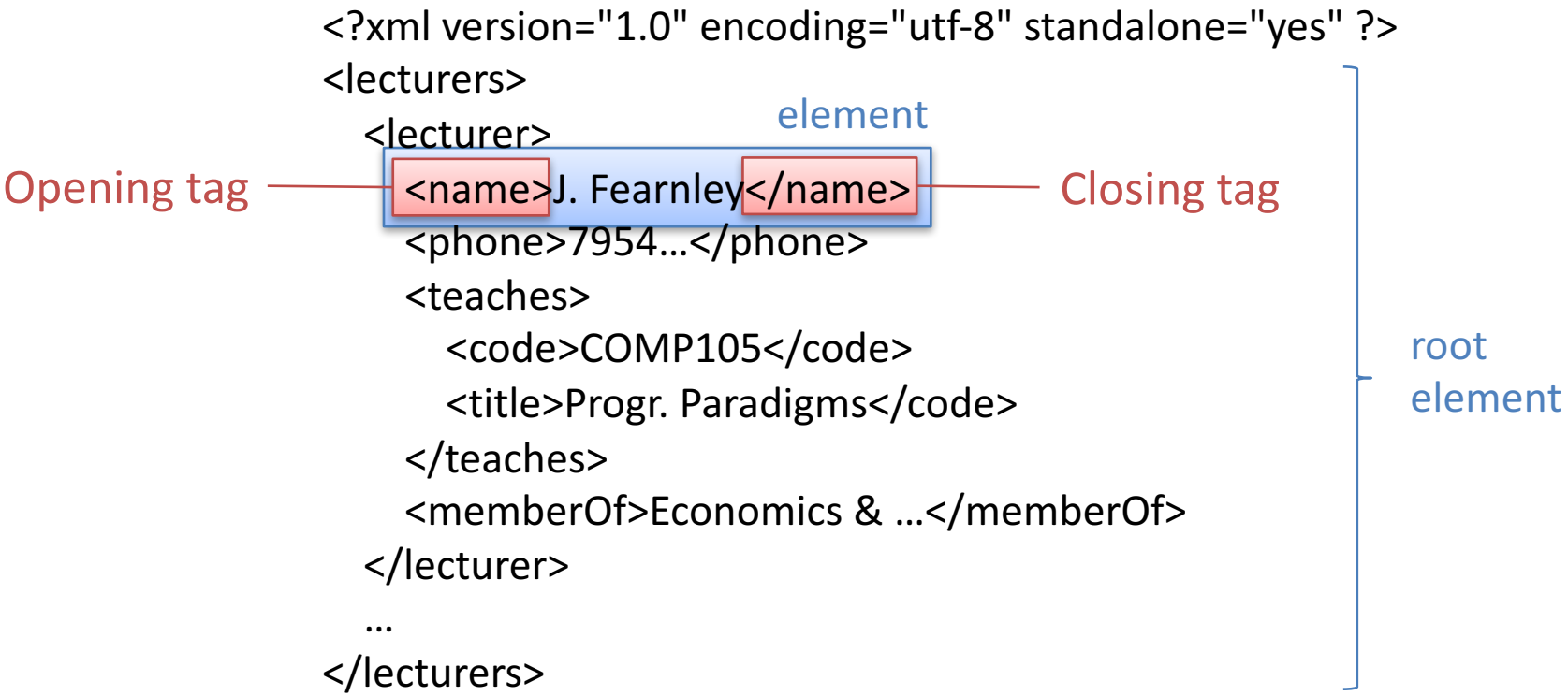
XML

- Notation for marking documents (similar to HTML)
- An XML document representing the example graph:



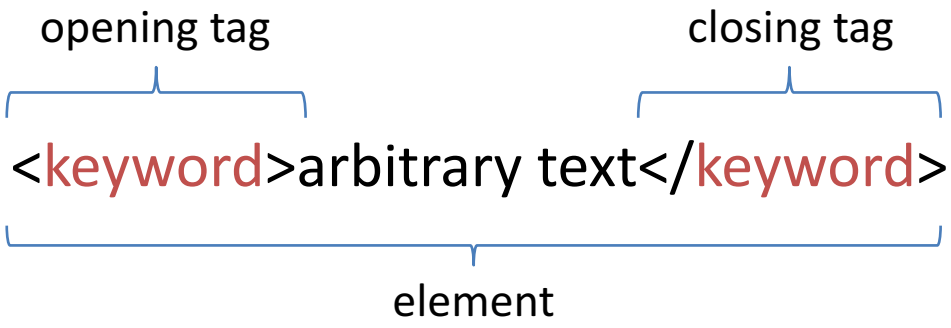
XML

- Notation for marking documents (similar to HTML)
- An XML document representing the example graph:



Elements & Tags

- An element has the following form:



- Elements may be nested
  - Nestings must be proper
  - The **root element** is the element not contained in any other element
  - A well-formed XML document must have **exactly one root element**
- Elements may be empty, in which case we combine the opening and the closing tag: <keyword />
- Elements are case sensitive

Attributes

- Elements can have attributes (name-value pairs), which are put into the element's opening tag

```
<lecturer name="J. Fearnley">
    <module code="COMP105" title="Programming Paradigms" />
</lecturer>
```

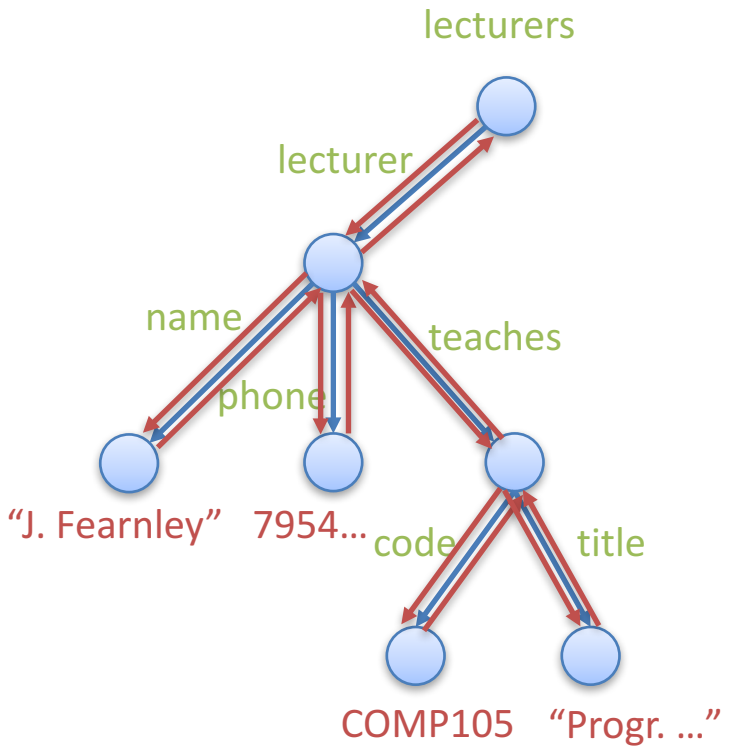
attribute 1 attribute 2

- Only one attribute of a given name for each element
- Useful for modelling "side links"
- **Question:** Attribute or a sub-element?
  - Staff ID of a lecturer
  - Email address

Document Order

- Elements in an XML document are **ordered as they occur in the document**
  - Part of the data representation!

```
<?xml … ?>
<lecturers>
   <lecturer>
      <name>J. Fearnley</name>
      <phone>7954…</phone>
      <teaches>
         <code>COMP105</code>
         <title>Progr. Paradigms</code>
      </teaches>
</lecturer>
…
```

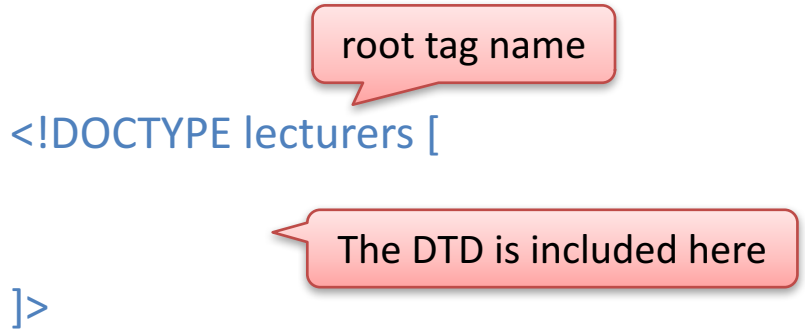


Other Features

- Optional **document type definition** at start of XML document.
- **Entity references**: serve various purposes, such as shortcuts to often repeated text or to distinguish reserved characters from content.
- **Comments**: enclosed in <!– and --> tags.
- **CDATA sections**: instructs XML processor to ignore markup characters and pass enclosed text directly to application.
- **Processing instructions**: can also be used to provide information to application.

Document Type Definitions (DTDs)

- Information about the structure of an XML document
  - Elements that may occur in the document,
  - Subelements that an element may have,
  - Attributes that an element may have, etc.
- Included at the beginning the XML document (between <?xml … **standalone="no"**> and root element)



- Good tradeoff between a full schema and flexibility

Document Type Definitions (DTDs)

root tag name

There must be 1 or more lecturer elements below the root element

```
<!ELEMENT lecturers (lecturer+)>
<!ELEMENT lecturer (name, phone?, email?, teaches*)
<!ELEMENT teaches (code, title)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT code (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

- Phone and email optional
- Zero or more occurrences of teaches

Special symbol for text data

DTDs – Element Type Declarations

- Identify the rules for elements that can occur in the XML document. Options for repetition are:
  - * indicates zero or more occurrences for an element;
  - + indicates one or more occurrences for an element;
  - ? indicates either zero occurrences or exactly one occurrence for an element.
- Name with no qualifying punctuation must occur exactly once.

DTDs – Attribute List Declarations

- Identify which elements may have attributes, what attributes they may have, what values attributes may hold, plus optional defaults. Some types:
- CDATA: character data, containing any text.
- ID: used to identify individual elements in document (ID is an element name).
- IDREF/IDREFS: must correspond to value of ID attribute(s) for some element in document.
- List of names: values that attribute can hold (enumerated type).

DTDs – Element Identity, IDs, IDREFs

- ID allows unique key to be associated with an element.
- IDREF allows an element to refer to another element with the designated key, and attribute type IDREFS allows an element to refer to multiple elements.
- To loosely model relationship Department has Lecturers:
  - <!ATTLIST lecturer staffNo ID #REQUIRED>
  - <!ATTLIST department staff IDREFS #IMPLIED>

DTDs – Document Validity

- Two levels of document processing: well-formed and valid.
- Non-validating processor ensures an XML document is well-formed before passing information on to application.
- XML document that conforms to structural and notational rules of XML is considered well-formed; e.g.:
  - document must start with <?xml version “1.0” …>
  - all elements must be within one root element
  - elements must be nested in a tree structure without any overlap

DTDs – Document Validity

- Validating processor will not only check that an XML document is well-formed but that it also conforms to a DTD, in which case XML document is considered valid.

XML Schema

- If you need/want more precision on the schema than offered by DTD
- Positive: (Beyond DTD) XML Schema can easily do:
  - Defined in XML
  - Give bounds on number of occurrences
  - Can do unordered sets
  - More complex simple data types
    - E.g. a number > 1915
- Negative:
  - The syntax is heavy

Example of XML Schema

- Want: XML Schema version of

```
<!ELEMENT lecturer (name, phone?, email?, teaches*)
<!ELEMENT teaches (code, title)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT code (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

XML Schema version

Means that anything prefixed with xs: must be an object as defined by 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>
```

Means we are defining a University schema

Content of …

```
<xs:complexType name = “theachesType”>
    <xs:all>
        <xs:element name=“code” type = “xs:string”/>
        <xs:element name=“title” type = “xs:string”/>
    </xs:all>
</xs:complexType>

<xs:complexType name = “lecturerType”>
    <xs:sequence>
        <xs:element name=“name” type = “xs:string
        <xs:element name=“phone” type = “phoneType”
                minOccurs = “0” maxOccurs = “1”/>
        <xs:element name=“email” type = “xs:string”
                minOccurs = “0” maxOccurs = “1”/>
        <xs:element name=“teaches” type = “theachesType”
                minOccurs = “10” maxOccurs = “20”/>
    </xs:sequence>
</xs:complexType>
```

If we want code and title to come in any order, we simply change to

If we want it to be a valid phone number, we change to

and add

If we want between 10 and 20 teaches, we simply change to

Definition of phoneType

```
<xs:simpleType name="phoneType">
      <xs:restriction base="xs:string">
            <xs:pattern value=”07[0-9]{3} [0-9]{6}"/>
     </xs:restriction>
</xs:simpleType>
```

Attributes in XML Schema

If we want name to be an attribute instead, we change to

```
<xs:complexType name = “lecturerType”>
      <xs:sequence>
            <xs:element name=“name” type = “xs:string”/>
            <xs:element name=“phone” type = “xs:string”
                        minOccurs = “0” maxOccurs = “1”/>
            <xs:element name=“email” type = “xs:string”
                        minOccurs = “0” maxOccurs = “1”/>
            <xs:element name=“teaches” type = “theachesType”
                        minOccurs = “0” maxOccurs = “unbounded”/>
    </xs:sequence>
</xs:complexType>
```

Attributes in XML Schema

If we want name to be an attribute instead, we change to

```
<xs:complexType name = “lecturerType”>
      <xs:sequence>
            <xs:element name=“phone” type = “xs:string”
                        minOccurs = “0” maxOccurs = “1”/>
            <xs:element name=“email” type = “xs:string”
                        minOccurs = “0” maxOccurs = “1”/>
            <xs:element name=“teaches” type = “theachesType”
                        minOccurs = “0” maxOccurs = “unbounded”/>
    </xs:sequence>
    <xs:attribute name=“name” type = “xs:string” use=“required”/>
</xs:complexType>
```

Attributes must come after sequence and are optional by default

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

Applying XML Schema

```
<? xml version="1.0" encoding="utf-8” ?>
<u:lecturers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=”University University.xsd”
      xmlns:u=”University">
      <lecturer name=“Rasmus”>
            <phone>07365 583329</phone>
      </lecturer>
</u:lecturers>
```

Means that the University schema is defined in University.xsd

Summary

- Semistructured data allows it to represent data in a flexible way (without imposing a rigid structure on the organisation of the data)
- XML is a language for representing semistructured data
- XML documents essentially correspond to the trees we've seen at the beginning of this lecture
- Remains: How to process XML documents?