Applied Databases

Lecture 1
Introduction, Basics of XML

Sebastian Maneth

Univeristy of Edinburgh - January 11th, 2016

Applied Databases

→ Apply database technology (e.g. MySQL) in varying contexts

→ with other technology:

- XML
- Lucene (full-text search)
- RDF

Applied Databases

→ Apply database technology (e.g. MySQL) in varying contexts

→ with other technology:

- XML
- Lucene (full-text search)
- RDF

---

also “big data” issues: → similarity search
→ mining / analytics

Course Organization

Lectures Monday 14:10-15:00
Lecture Theatre 3, Appleton Tower

Thursday 14:10-15:00
Lecture G.03, 50 George Square

Lecturer Sebastian Maneth (smaneth@inf.ed.ac.uk)
TA Geng Lyu

Assessment Exam (70%)

Assignment 1 (15%)
due 12th February, 4:00pm

Assignment 2 (15%)
due 11th March, 4:00pm

Course Format

21 Lectures all material covered in the lectures is examinable

Assignments Lectures 1-8 cover material relevant to the Assignments

Assignments

→ taken, with consent and warm thanks,
from UCLA lecture “CS144: Web Applications”

Assignments 1 & 2

- Programming assignments, in Java & SQL

- Pair programming:
you are allowed to program in pairs of two persons

Rules:
→ either alone or with partner
→ may change partner for 2nd assignment
→ submit **one** solution
→ same mark for both in the team

Assignments

Pair programming

→ together design database schema

→ individually write load functions for different tables

Ideally together find abstractions that
make the code *small*, *elegant*, and *readable*

Assignment 1

1) design a relational schema for EBAY data

2) convert EBAY data from XML into relational tables (csv files)

3) import csv files into a MySQL database

4) execute some SQL queries over the database

Assignment 1

1) design a relational schema for EBAY data

2) convert EBAY data from XML into relational tables (csv files)

3) import csv files into a MySQL database

4) execute some SQL queries over the database

Requires

- XML parsing (DTDs, DOM, SAX)

- basic DB knowledge (schema design, basic SQL queries)

Assignment 1

1) design a relational schema for EBAY data

2) convert EBAY data from XML into relational tables (csv files)

3) import csv files into a MySQL database

4) execute some SQL queries over the database

Requires

Lectures 1 & 2

- XML parsing (DTDs, DOM, SAX)

- basic DB knowledge (schema design, basic SQL queries)

Lectures 3 & 4

Assignment 2

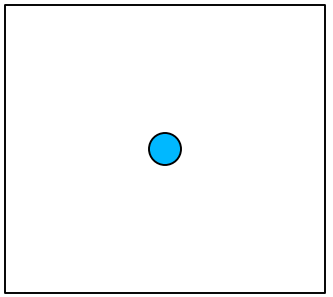
1) create a Lucene full-text Index (from Java)

2) implement a basic keyword search function

3) build a spatial index in MySQL

4) implement spatial search

5) create web interface for keyword & spatial search and for display of results



Assignment 2

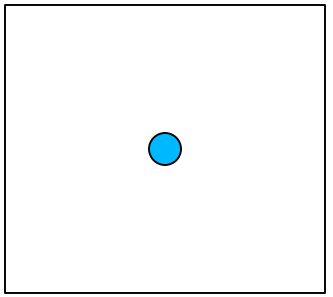
1) create a Lucene full-text Index (from Java)

2) implement a basic keyword search function

3) build a spatial index in MySQL

4) implement spatial search

5) create web interface for keyword & spatial search
   and for display of results



Requires

- spatial search

Lecture 6

- basic knowledge of Lucene / text-indexing

Lecture 7

Assignment 2

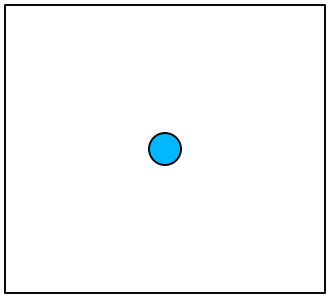
1) create a Lucene full-text Index (from Java)

2) implement a basic keyword search function

3) build a spatial index in MySQL

4) implement spatial search

5) create web interface for keyword & spatial search and for display of results



Assignments 1 & 2

→ hands-on experience to implement a web store such as EBAY or similar!!

Applied Databases

Main Topics

- → Heterogeneous Data (XML, Text, RDF / Graph) **Lectures 1,2,7-13**
- → Recap RDBMS, SQL, Schema Design, Indexes **Lectures 3-6**
- → Similarity Search **Lectures 14-18**
- → Data Analytics **Lectures 19-21**

Lecture 1

Basics of XML

Outline

1. Motivations for XML
2. Well-formed XML
3. Parsing / DTD Validation: Introduction

XML

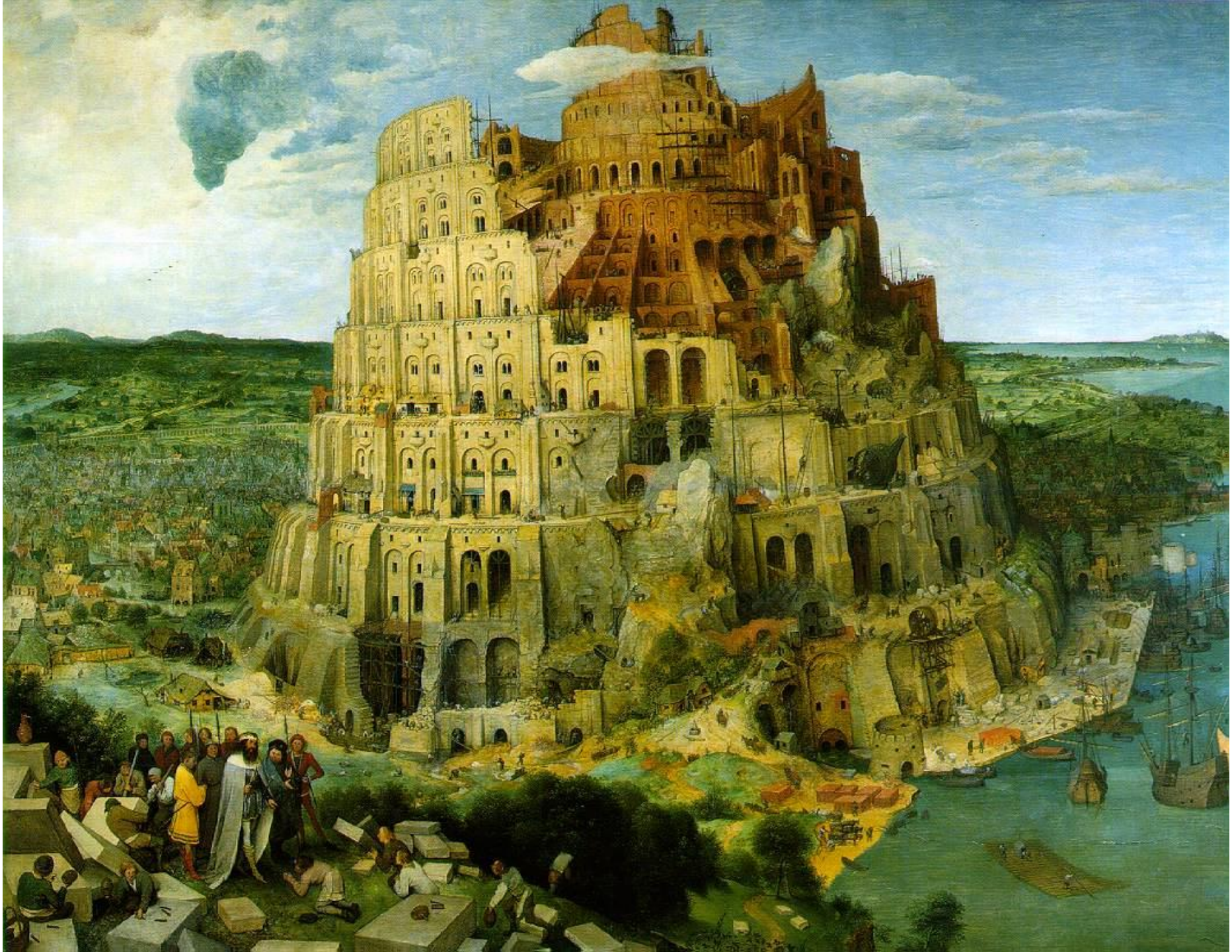
→ Similar to HTML (Berners-Lee, CERN → W3C)
use your own tags.

→ XML is the de-facto standard for data exchange on the web

1. XML

Motivation

to have one language to speak about data



1. XML Motivation

→ XML is a **Data Exchange Format**

1974	SGML Standardized Generalized Markup Language (Charles Goldfarb at IBM Research)
1989	HTML (Tim Berners-Lee at CERN/Geneva)
1994	Berners-Lee founds Web Consortium (W3C)
1996	**XML** (W3C draft, v1.0 in 1998)

`http://www.w3.org/TR/REC-xml/`

XML = data

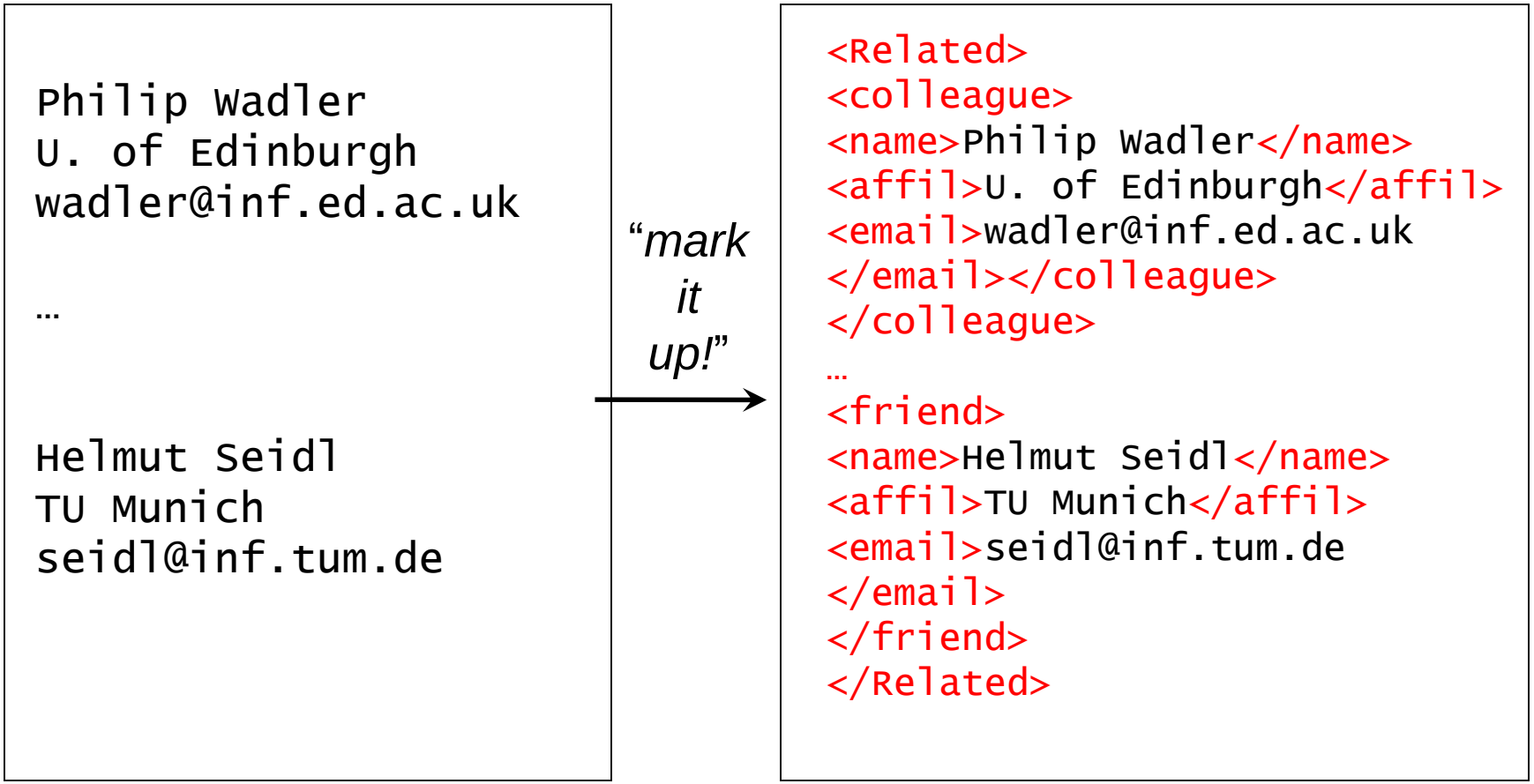
```
Philip Wadler
U. of Edinburgh
wadler@inf.ed.ac.uk

…


Helmut Seidl
TU Munich
seidl@inf.tum.de
```

Text file

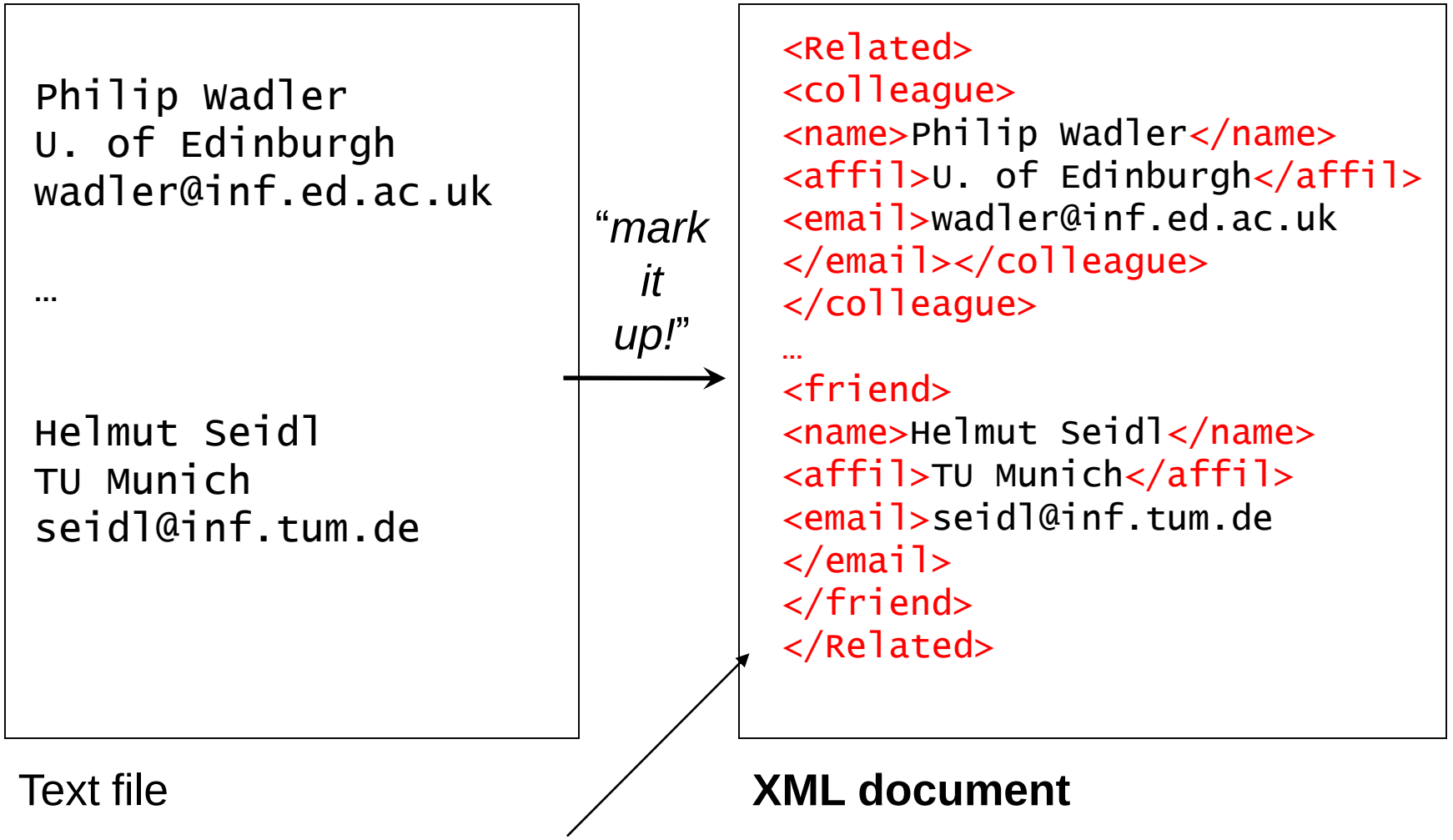
XML = data **+ structure**



Text file

XML document

XML = data **+ structure**



XML Documents

→ Ordinary text files (UTF-8, UTF-16, UCS-4 …)

→ Originates from typesetting/DocProcessing community

→ Idea of labeled brackets (“mark up”) for structure is not new!
(already used by Chomsky in the 1960’s)

→ Brackets describe a tree structure

→ Allows applications from different vendors to exchange data!

➔ **standardized, extremely widely accepted!**

XML Documents

→ Ordinary text files (UTF-8, UTF-16, UCS-4 …)

→ Originates from typesetting/DocProcessing community

→ Idea of labeled brackets (“mark up”) for structure is not new!
(already used by Chomsky in the 1960’s)

→ Brackets describe a tree structure

→ Allows applications from different vendors to exchange data!

➔ **standardized, extremely widely accepted!**

Social Implications!
All sciences (biology, geography, meteorology, astrology…)
have own XML “dialects” to exchange *their* data optimally

XML Documents

→ Ordinary text files (UTF-8, UTF-16, UCS-4 …)

→ Originates from typesetting/DocProcessing community

→ Idea of labeled brackets (“mark up”) for structure is not new!
(already used by Chomsky in the 1960’s)

→ Brackets describe a tree structure

→ Allows applications from different vendors to exchange data!

➔ **standardized, extremely widely accepted!**

Problem highly verbose, lots of repetitive markup

XML Documents

→ Ordinary text files (UTF-8, UTF-16, UCS-4 …)

→ Originates from typesetting/DocProcessing community

→ Idea of labeled brackets (“mark up”) for structure is not new!
(already used by Chomsky in the 1960’s)

→ Brackets describe a tree structure

→ Allows applications from different vendors to exchange data!

➔ **standardized, extremely widely accepted!**

Contra.. highly verbose, lots of repetitive markup

Pro.. we have a standard! A STANDARD!

→ ☺ *You never need to write a parser again! Use XML!* ☺

XML: Validation & Parsing

… instead of writing a validator, you simply fix your own "XML dialect",

by describing all "admissible templates" (+ maybe even the specific

data types that may appear inside).

You do this, using an *XML Type definition language* such as DTD, XML Schema, or Relax NG.

→ *type definition languages* must be SIMPLE, because you want the parsers to be efficient!

They are similar to EBNF. → context-free grammar with reg. expr's in the right-hand sides. ☺

XML Documents

Example **DTD** (Document Type Description)

```
Related    → (colleague | friend | family)*
colleague  → (name,affil*,email*)
friend     → (name,affil*,email*)
family     → (name,affil*,email*)
name       → (#PCDATA)
…
```

Element names and their content

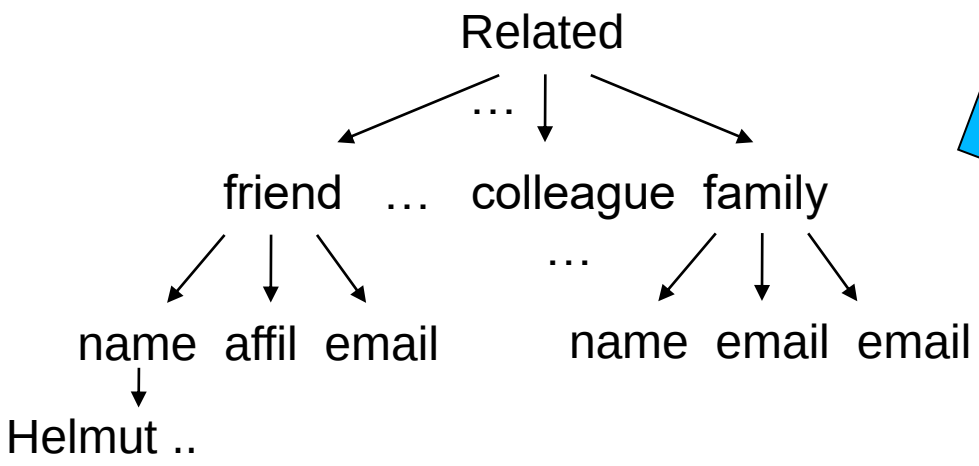
XML Documents

Example DTD (Document Type Description)

```
Related    → (colleague | friend | family)*
colleague  → (name,affil*,email*)
friend     → (name,affil*,email*)
family     → (name,affil*,email*)
name       → (#PCDATA)
…
```

Element names and their content

ordered, unranked tree

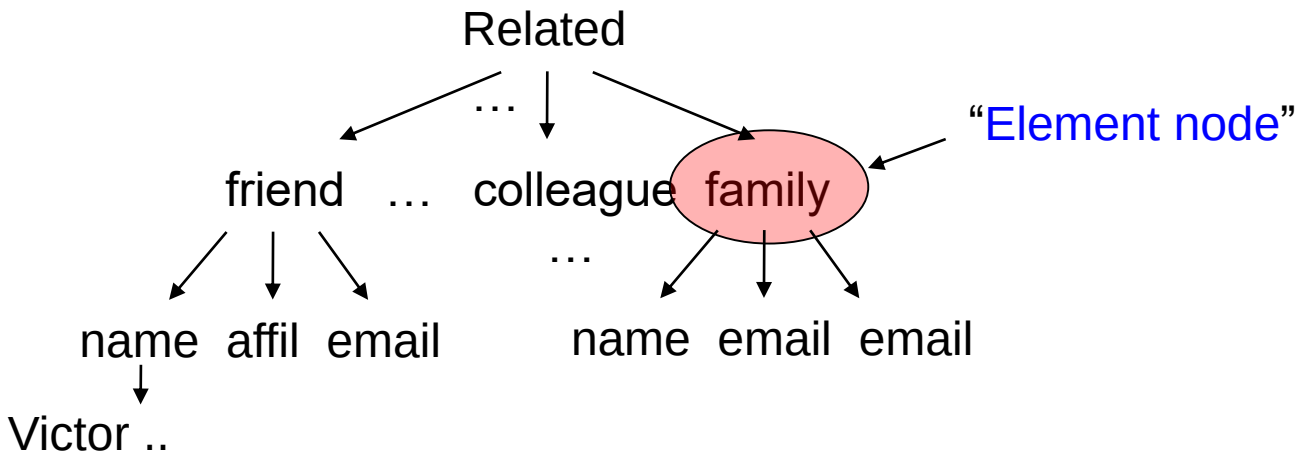


XML Documents

Example DTD

```
Related     → (colleague | friend | family)*
colleague   → (name,affil*,email*)
friend      → (name,affil*,email*)
family      → (name,affil*,email*)
name        → (#PCDATA)
…
```

Element names and their content

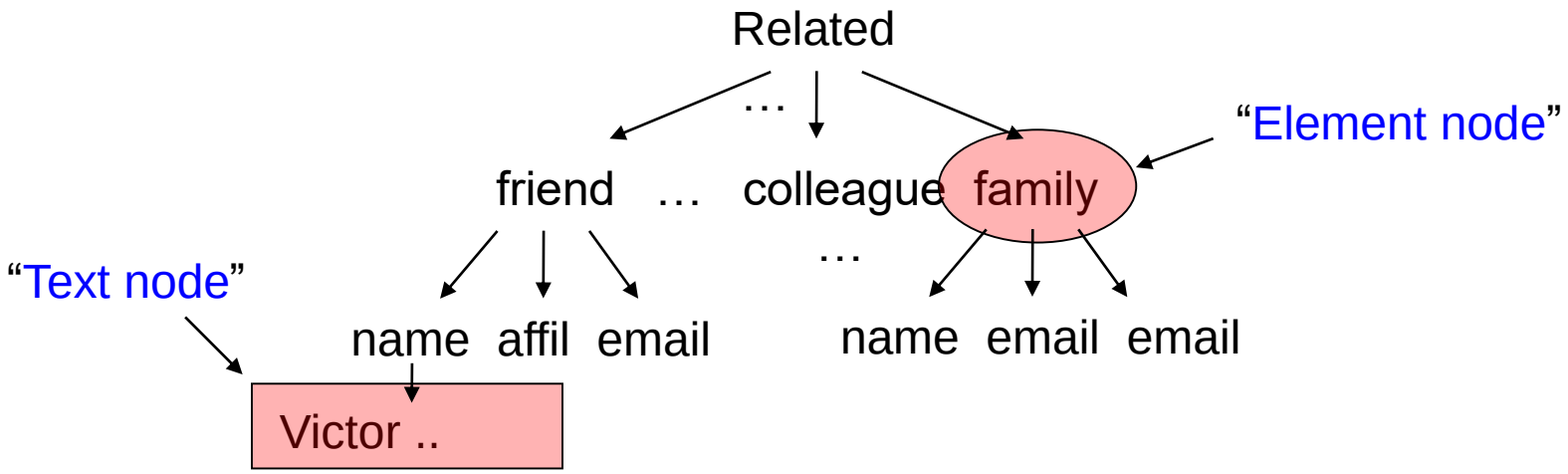


XML Documents

Example DTD

```
Related     → (colleague | friend | family)*
colleague   → (name,affil*,email*)
friend      → (name,affil*,email*)
family      → (name,affil*,email*)
name        → (#PCDATA)
…
```

Element names and their content



XML Documents

Example DTD

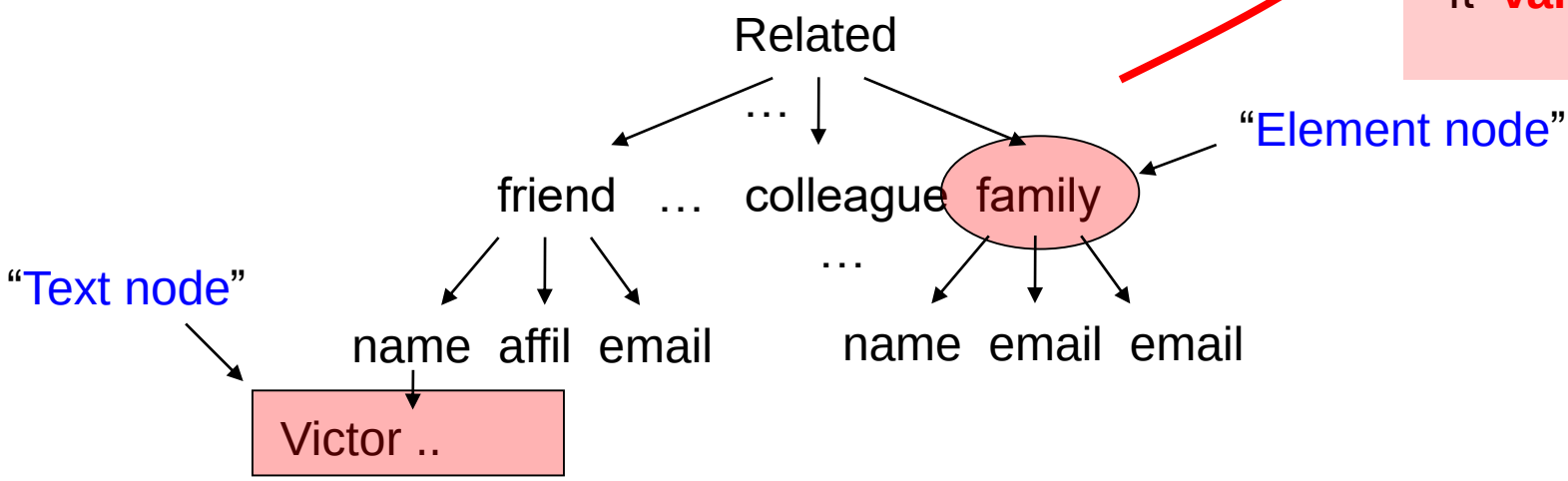
```
Related     → (colleague | friend | family)*
colleague   → (name,affil*,email*)
friend      → (name,affil*,email*)
family      → (name,affil*,email*)
name        → (#PCDATA)
…
```

Element names and their content

Terminology

document is **valid** wrt the DTD

"It **validates**"



XML Documents

What else: (besides *element* and *text* nodes)

→ attributes
→ processing instructions
→ comments
→ namespaces
→ entity references (two kinds)

XML Documents

What else: (besides *element* and *text* nodes)

→ attributes
→ processing instructions
→ comments
→ namespaces
→ entity references (two kinds)

<family rel="brother",age="25">
<name>
…
</family>

XML Documents

<?php sql (“SELECT * FROM …”) …?>

See 2.6 Processing Instructions

What else:

→ attributes
→ processing instructions
→ comments
→ namespaces
→ entity references (two kinds)

<family rel=“brother”,age=“25”>
<name>
…
</family>

XML Documents

<?php sql (“SELECT * FROM …”) …?>

See 2.6 Processing Instructions

What else:

→ attributes
→ processing instructions
→ comments `<!-- some comment -->`
→ namespaces
→ entity references (two kinds)

<family rel=“brother”,age=“25”>
<name>
…
</family>

XML Documents

<?php sql (“SELECT * FROM …”) …?>

See 2.6 Processing Instructions

What else:

→ attributes
→ processing instructions
→ comments `<!-- some comment -->`
→ namespaces
→ entity references (two kinds)

<family rel=“brother”,age=“25”>
<name>
…
</family>

<!-- the 'price' element's namespace is http://ecommerce.org/schema -->
<edi:price xmlns:edi='http://ecommerce.org/schema' units='Euro'>32.18</edi:price>

XML Documents

<?php sql (“SELECT * FROM …”) …?>

See 2.6 Processing Instructions

What else:

→ attributes
→ processing instructions
→ comments `<!-- some comment -->`
→ namespaces
→ entity *references* (two kinds) → *character reference*

Type <key>less-than</key>
(<) to save options.

<family rel=“brother”,age=“25”>
<name>
…
</family>

<!-- the 'price' element's namespace is http://ecommerce.org/schema -->
<edi:price xmlns:edi='http://ecommerce.org/schema' units='Euro'>32.18</edi:price>

XML Documents

<?php sql (“SELECT * FROM …”) …?>

See 2.6 Processing Instructions

What else:

→ attributes
→ processing instructions
→ comments `<!-- some comment -->`
→ namespaces
→ entity *references* (two kinds)

character reference
Type <key>less-than</key>
(<) to save options.

<family rel=“brother”,age=“25”>
<name>
…
</family>

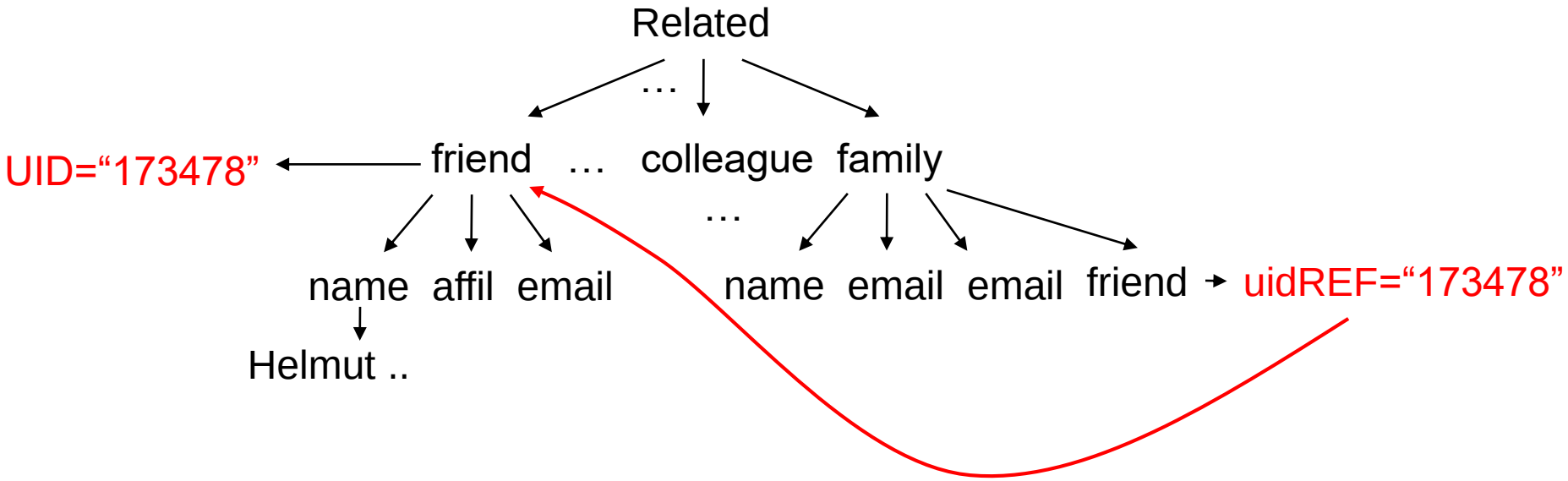
This document was prepared on &docdate; and

<!-- the 'price' element's namespace is http://ecommerce.org/schema -->
<edi:price xmlns:edi='http://ecommerce.org/schema' units='Euro'>32.18</edi:price>

XML: not tree but DAG (Directed Acyclic Graph)

→ attributes of type ID: must be unique, i.e., no duplicate values

→ may be referenced via attributes of type IDREF



XML, typical usage scenario

Document structure
Def. of price, gst, …

DTD, XML Schema

Presentation
Format info

XML Stylesheet

```
<Product>
  <product_id> m101 </product_id>
  <name> Sony walkman </name>
  <currency> AUD </currency>
  <price> 200.00 </price>
  <gst> 10% </gst>
</Product>
…
```

X M L

XML Stylesheet

XML Stylesheet

One data source

→ several
(dynamic gen.) **views**

XML: has it succeded?

Yes and No:

- has become *very* popular and adopted

- technically it is still (!) challenging:

  (*) standard too complex
  (*) causes, e.g., slowness of XML parsers
  (a “threat to databases”)

→ JSON

- invented in 2001 by Douglas Crockford
- took off since 2005/2006

JavaScript Object Notation

XML Grammar - EBNF-style

```
[1]    document    ::=   prolog element Misc*
[2]        Char    ::=   a Unicode character
[3]           S    ::=   (‘ ’ | ‘\t’ | ‘\n’ | ‘\r’)+
[4]    NameChar    ::=   (Letter | Digit | ‘.’ | ‘-’ | ‘:’
[5]        Name    ::=   (Letter | '_' | ':') (NameChar)*

[22]     prolog    ::=   XMLDecl? Misc* (doctypedecl Misc*)?
[23]    XMLDecl    ::=   '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>‘
[24]VersionInfo    ::=   S'version'Eq("'"VersionNum"'"|'"'VersionNum'"')
[25]         Eq    ::=   S? '=' S?
[26]VersionNum     ::=   '1.0‘

[39]    element    ::=   EmptyElemTag
                       | STag content Etag
[40]       STag    ::=   '<' Name (S Attribute)* S? '>'
[41]  Attribute    ::=   Name Eq AttValue
[42]       ETag    ::=   '</' Name S? '>‘
[43]    content    ::=   (element | Reference | CharData?)*
[44]EmptyElemTag   ::=   '<' Name (S Attribute)* S? '/>‘

[67] Reference     ::=   EntityRef | CharRef
[68] EntityRef     ::=   '&' Name ';‘
[84] Letter        ::=   [a-zA-Z]
[88] Digit         ::=   [0-9]
```

2. Well-Formed XML

From the W3C XML recommendation

`http://www.w3.org/TR/REC-xml/`

"A textual object is **well-formed XML** if,

(1) taken as a whole, it **matches the production labeled** *document*
(2) it meets all the **well-formedness constraints** given in this specification .."

---

document = start symbol of a context-free grammar ("XML grammar")

→ (1) contains the *contex-free properties* of well-formed XML
→ (2) contains the *context-dependent properties* of well-formed XML

There are 10 WFCs (well-formedness constraints).
E.g.: **Element Type Match** "The Name in an element's end tag must match the element name in the start tag."

→ Why is this *not* context-free?

XML vs JSON

```
<Related>
<colleague>
<name>Philip Wadler</name>
<affil>U. of Edinburgh</affil>
<email>
wadler@inf.ed.ac.uk
</email></colleague>
</colleague>
…
<friend>
<name>Helmut Seidl</name>
<affil>TU Munich</affil>
<email>seidl@inf.tum.de
</email>
</friend>
</Related>
```

```
Related = {
“colleague”:{
“name”:”Philip Wadler”,
“affil”:”U. of Edinburgh”,
“email”:wadler@inf.ed.ac.uk
}
…
“friend”: {
“name”:”Helmut Seidl”,
“affil”:”TU Munich”,
“email”:”seidl@inf.tum.de”}
```

XML vs JSON

- 7 node types
- DTDs are built in

Very rich schema languages, e.g.,

- XML Schema

(e.g., XHTML schema: >2000 lines)

6 data types:

- numbers
- strings
- booleans (`true` / `false`)
- array
- object (set of `name:value` pairs)
- empty value (`null`)

XML Parsing: A Threat to Database Performance

Matthias Nicola
IBM Silicon Valley Lab
555 Bailey Avenue
San Jose, CA 95123, USA
mnicola@us.ibm.com

Jasmi John
IBM Toronto Lab
8200 Warden Ave
Markham, ON L6G 1C7, Canada
jasmij@ca.ibm.com

in: Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, November 2-8, 2003

ABSTRACT

XML parsing is generally known to have poor performance characteristics relative to transactional database processing. Yet, its potentially fatal impact on overall database performance is being underestimated. We report real-word database applications where XML parsing performance is a l[illegible] deployment. There is a considera[illegible] cations which are prone to fail a[illegible] XML parsing. We analyze XM[illegible] tify the extra overhead of DTD [illegible] son with relational database pe[illegible] response times and transaction rates over XML data can not be achieved without major improvements in XML parsing technology. Thus, we identify research topics which are most promising for XML parser performance in database systems.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems–*transaction processing.*

General Terms: Algorithms, Measurement, Performance, Design.

Keywords: XML, Parser, Database, Performance, SAX, DOM, Validation.

1. INTRODUCTION

tially because processing of XML requires *parsing* of XML documents which is very CPU intensive.

The performance of many XML operations is often determined by the performance of the XML parser. Examples are converting XML into a relational format, evaluating XPath expressions, or [illegible]

2. XML PARSING IN DATABASES

There are two models of XML parsing, DOM and SAX. DOM parsers construct the "Document Object Model" in main memory which requires a considerable amount of CPU time and memory (2 to 5 times the size of the XML document, hence unsuitable for large documents). Lazy DOM parsers materialize only those parts of the document tree which are actually accessed, but if most the document is accessed lazy DOM is slower than regular DOM. SAX parsers report parsing events (e.g. start and end of elements) to the application through callbacks. They deliver an event stream which the application processes in event handlers. The memory consumption does not grow with the size of the document. In general, applications requiring random access to the document nodes use a DOM parser while for serial access a SAX parser is better.

How expensive is XML Validation?

→ DTD is part of XML

→ DTDs may contain (deterministic) regular expressions

→ How expensive is it to match a text of size n
against a regular expression of size m?

→ DTDs allow recursive definitions

→ DTDs can specify ID and IDREF attributes
(ID: check uniqueness, IDREF: check existence)

How expensive is XML Validation/Parsing?

→ DTD is part of XML

→ DTDs may contain (deterministic) regular expressions

→ How expensive is it to match a text of size n
against a regular expression of size m?

→ DTDs allow recursive definitions

→ DTDs allow ID and IDREF attributes
(ID: check uniqueness, IDREF: check existence)

Compare this to parsing complexity of

→ JSON
→ csv files (csv = “comma-separated values”) [IBM Fortran, 1967]

END
Lecture 1