

XML in Communication Systems

Dr. Jesper Zedlitz

Research Group for Communication Systems

Dept. of Computer Science, University of Kiel

Kiel, Germany



May I introduce myself

Informatik · CAU Kiel

Studies

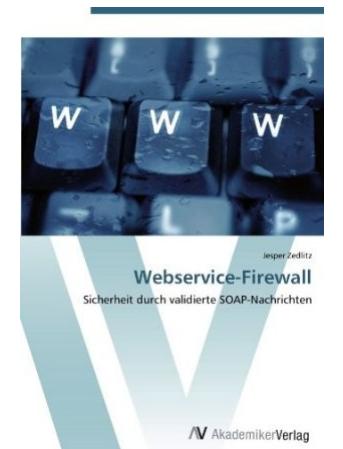
1999-2005 University Kiel

Subject

Computer Science (Dipl.-Inf.)

- Thesis

Spezifikation und Implementierung
eines Application Level Gateways
für Webservices



May I introduce myself

Informatik · CAU Kiel



2005-2007
software development
process automation



2007-2011
Department of Computer Science
RG for Communication Systems

Oceanographic research data
and sensors

IEEE 1451.2 expert group



May I introduce myself

Informatik · CAU Kiel



Photos: V. Diekamp

Expedition in the
Mediterranean
June/July 2010
Controlling the
“MeBo”

May I introduce myself

Informatik · CAU Kiel



2011-2013
Management of scientific
research data

- Dissertation
 - „Konzeptuelle Modellierung mit UML und OWL – Untersuchung der Gemeinsamkeiten und Unterschiede mit Hilfe von Modelltransformationen“

What is XML?

Informatik · CAU Kiel



What is XML?

Informatik · CAU Kiel

- Your answers?
- Mike Wesch's answer
 - Associate Professor of Cultural Anthropology, Kansas State University
- My answers
 - XML is a formal notation.
 - XML is a formal notation for tree-structures, i.e. "documents".
 - XML is a formal notation for message bodies.
 - XML is a formal notation for modeling languages.



Your Lecture Goals

Informatik · CAU Kiel

YOUR GOALS

YOUR NON-GOALS

My Lecture Goals

Informatik · CAU Kiel

- to make you familiar with XML language technology
- to enable you to design XML-based services and protocols
- to enable an informed decision on applications
- to guide you in XML-based modeling
- to provide some working knowledge
- to give some good examples

MY GOALS

- to cover all technical details
- to give a comprehensive tutorial
- to lecture entire specifications
- to learn & practice several new X-languages

MY NON-GOALS

Lecture Overview

Informatik · CAU Kiel

- Introduction
 - Introduction to XML
- Grammars for XML documents
 - XML information set
 - Document Type Definition
 - W3C XML Schema, Part 1
 - W3C XML Schema, Part 2

Lecture Overview

Informatik · CAU Kiel

- XML processing
 - Tree processing
 - Stream processing
- XML navigation, transformation
 - Xpath
 - XML Stylesheet Language Transformations (XSLT)
 - Grammar again: Schematron

Lecture Overview

Informatik · CAU Kiel

- Web Services
 - Introduction
 - SOAP
 - Web Service Description Language (WSDL)
- Web Service security
 - Introduction to Digital Signatures
 - WS security
 - WS security policy
 - DoS attacks against Web Services

Lecture Overview

Informatik · CAU Kiel

- XML-based Modeling
 - Resource Description Framework (RDF)
 - Geography Markup Language (GML)

Please ...

Informatik · CAU Kiel

I appreciate any comments,
questions, remarks.

Please, interact!



Introduction

Introduction to XML

Lecture "XML in Communication Systems"

Chapter 1

Dr.-Ing. Jesper Zedlitz

Research Group for Communication Systems

Dept. of Computer Science

Christian-Albrechts-University in Kiel



Acknowledgements

Informatik · CAU Kiel

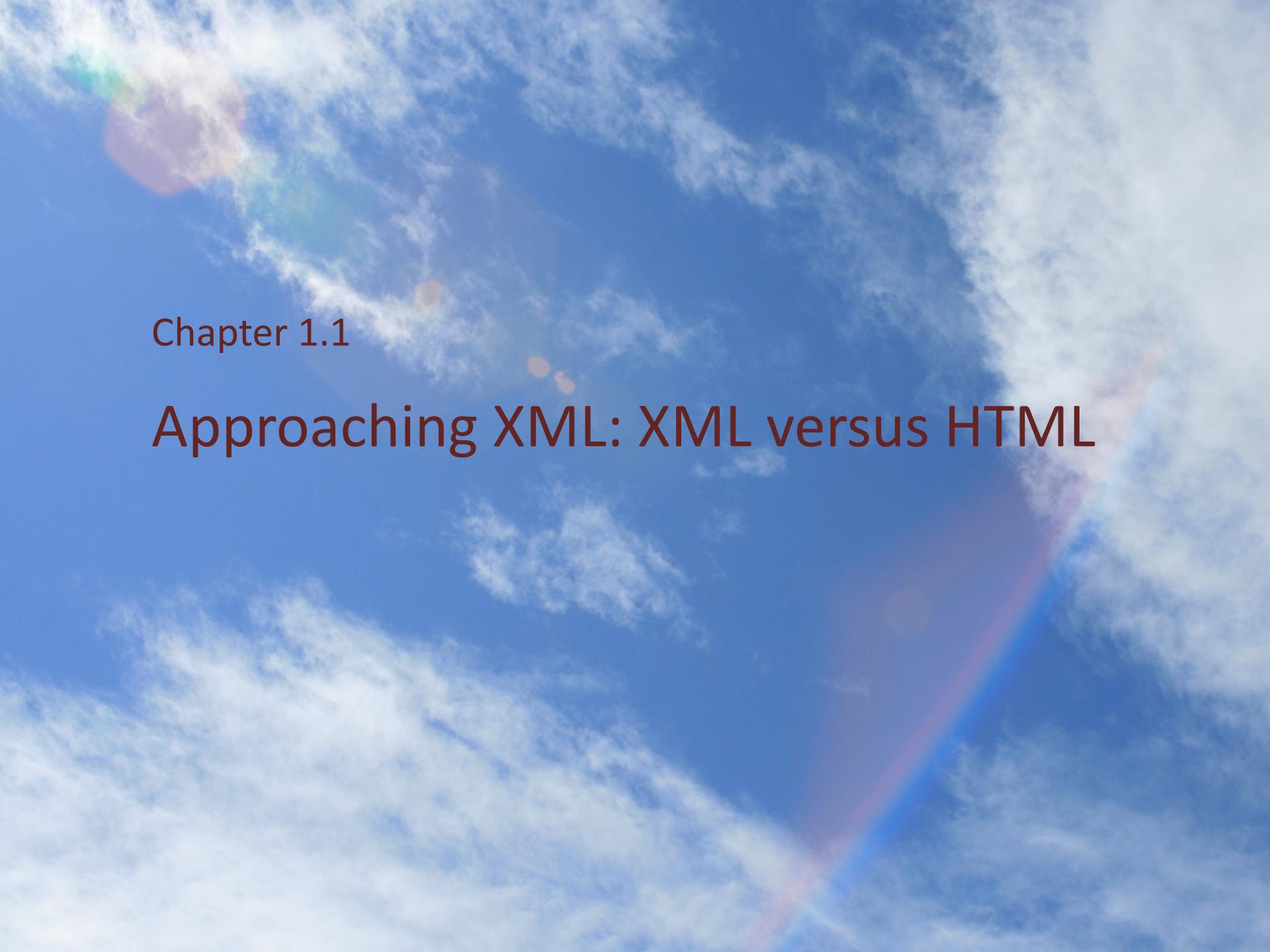
- **Gregor Engels**
University of Paderborn
- **Matthew Langham**
s&n AG Paderborn

Recommended Reading

- T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, J. Cowan (Eds.):
XML 1.1 (Second Edition), W3C Recommendation, 16 August 2006.
<http://www.w3.org/TR/xml11>
- The Unicode Consortium:
The Unicode 5.0 Standard.
can be downloaded chapter-wise from
<http://www.unicode.org/versions/Unicode5.0.0/>

Overview

1. Approaching XML
 1. XML versus HTML
 2. Machine-to-machine communication
2. Well-formed XML documents
3. Excursion: Character sets
4. An initial XML shopping list



Chapter 1.1

Approaching XML: XML versus HTML

XML versus HTML

- An HTML example

```
<h2>Inside XML</h2>
<p>
<i>by
<b>Holzner, St.</b> and
<b>Else, Someone</b>
</i>
<br>
Indianapolis (New Riders) 2001
<br>
ISBN 0-7357-1020-1
```

- and the same example in XML

```
<book>
  <title>Inside XML</title>
  <author>Holzner, St.</author>
  <author>Else, Someone</author>
  <publisher>Indianapolis (New
Riders)
  </publisher>
  <year>2001</year>
  <ISBN>0-7357-1020-1</ISBN>
</book>
```

HTML versus XML

- Similarities
 - Both use **tags**, e.g. <h2> and </year>
 - Tags may be **nested**: tags within tags
 - **Human** users can read and interpret both HTML and XML representations quite easily.

XML versus HTML

- Another example

```
<h2>Relationship  
matter-energy</h2>  
<p>  
<i>E = M × c2</i>
```

- and the same example in XML

```
<equation>      <meaning>Relationship  
                         matter-energy</meaning>  
                         <leftside>E</leftside>  
                         <rightside>M × c2</rightside>  
</equation>
```

- fixed set of tags
- presentation-oriented
- to be rendered via browser

- extensible set of tags
- content-oriented
- transformation before rendering

HTML versus XML

- HTML provides **typography-oriented** structuring
 - Document is cut into pieces alongside different "rules" for **typographical** representation
 - The main use of an HTML document is to **display** information.
- "HTML is XML for typography."
- though "pure HTML" does not actually specify typographical features for different kinds of document pieces

XML in Communications

Informatik · CAU Kiel

- Automated interpretation of HTML?

```
<h2>Inside XML</h2>
<p>
<i>by
<b>Holzner, St.</b> and
<b>Else, Someone</b>
</i>
<br>
Indianapolis (New Riders) 2001
<br>
ISBN 0-7357-1020-1
```

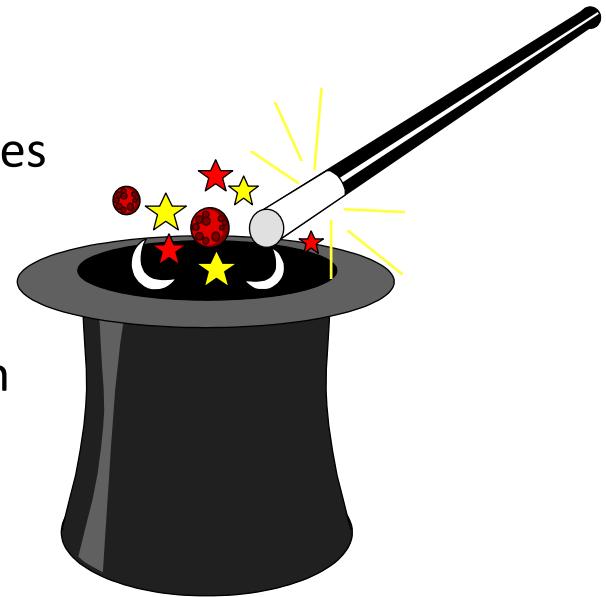
- Assume an (un)intelligent agent trying to retrieve the names of the authors of the book:
 - Authors' names appear immediately after the title
 - or immediately after the word "by"?
 - Are there two authors?
 - Or just one, called "Holzner, St. and Else, Someone"?

HTML versus XML

- XML provides **content-oriented** structuring of information
 - The content of every "piece of information" is described: tags
 - Relations are defined through the nesting structure:
e.g. the **author** element refers to the enclosing **book** element.
 - XML allows the definition of constraints on values
 - e.g. a year must be a number of four digits

XML versus HTML

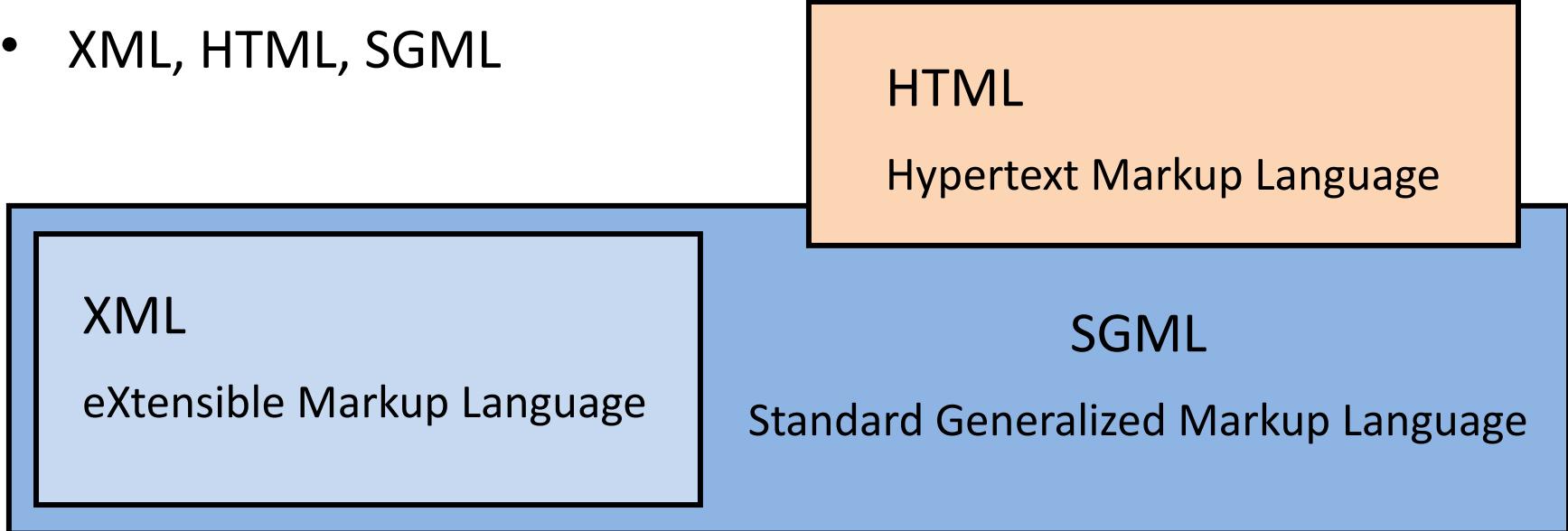
- XML language technology
 - XML = eXtensible Markup Language:
enables definition of domain-specific languages
(vocabulary and grammar)
 - keeps distinction between
internal structure and external representation
 - based on SGML, ISO standard since 1986
(Standard Generalized Markup Language)
 - standardised by W3C
 - XML 1.0 in Feb. 1998
 - XML 1.1 in Aug. 2006



XML versus HTML

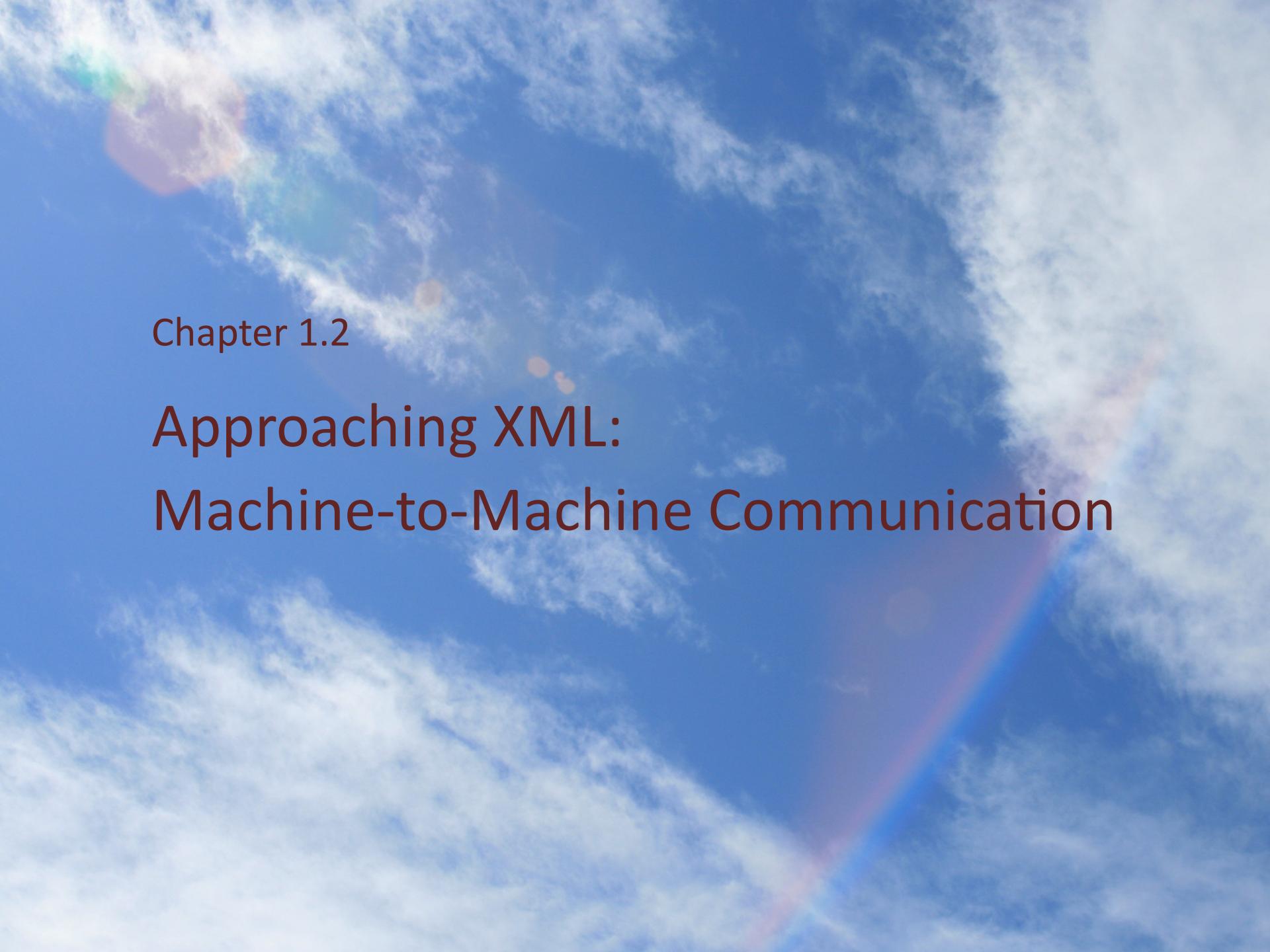
Informatik · CAU Kiel

- XML, HTML, SGML



- XML: a **subset** of SGML
- HTML: an **application** of SGML
for the typography domain

$$\text{HTML4.0} \in \text{XML} \subset \text{SGML}$$

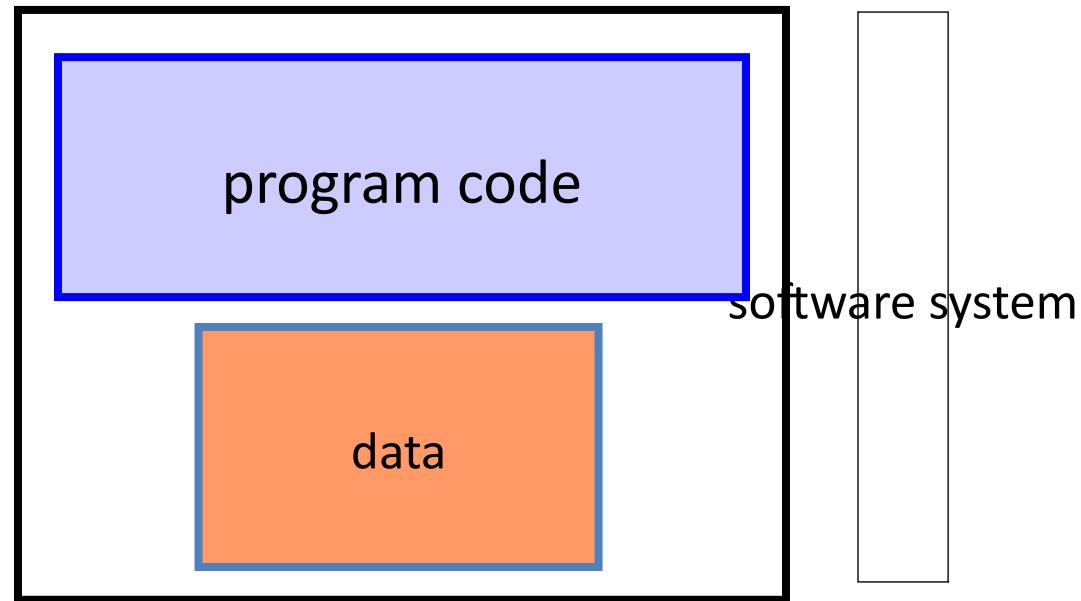


Chapter 1.2

Approaching XML: Machine-to-Machine Communication

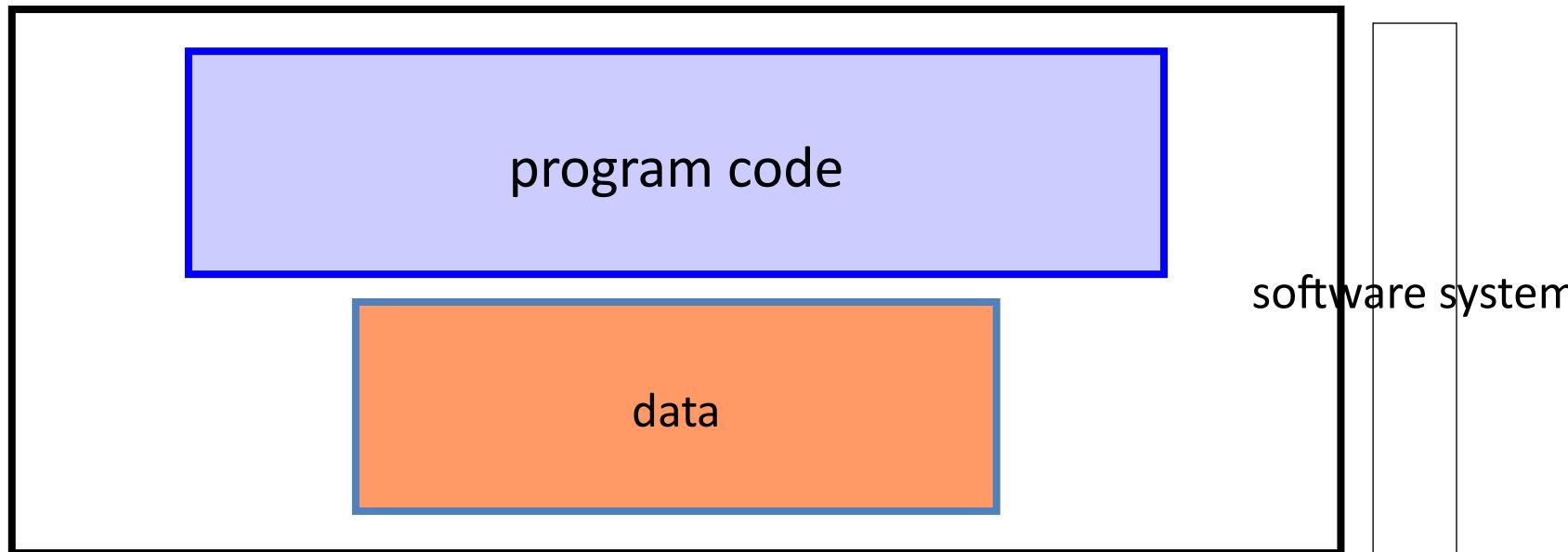
XML in Communications

- In the beginning ...



XML in Communications

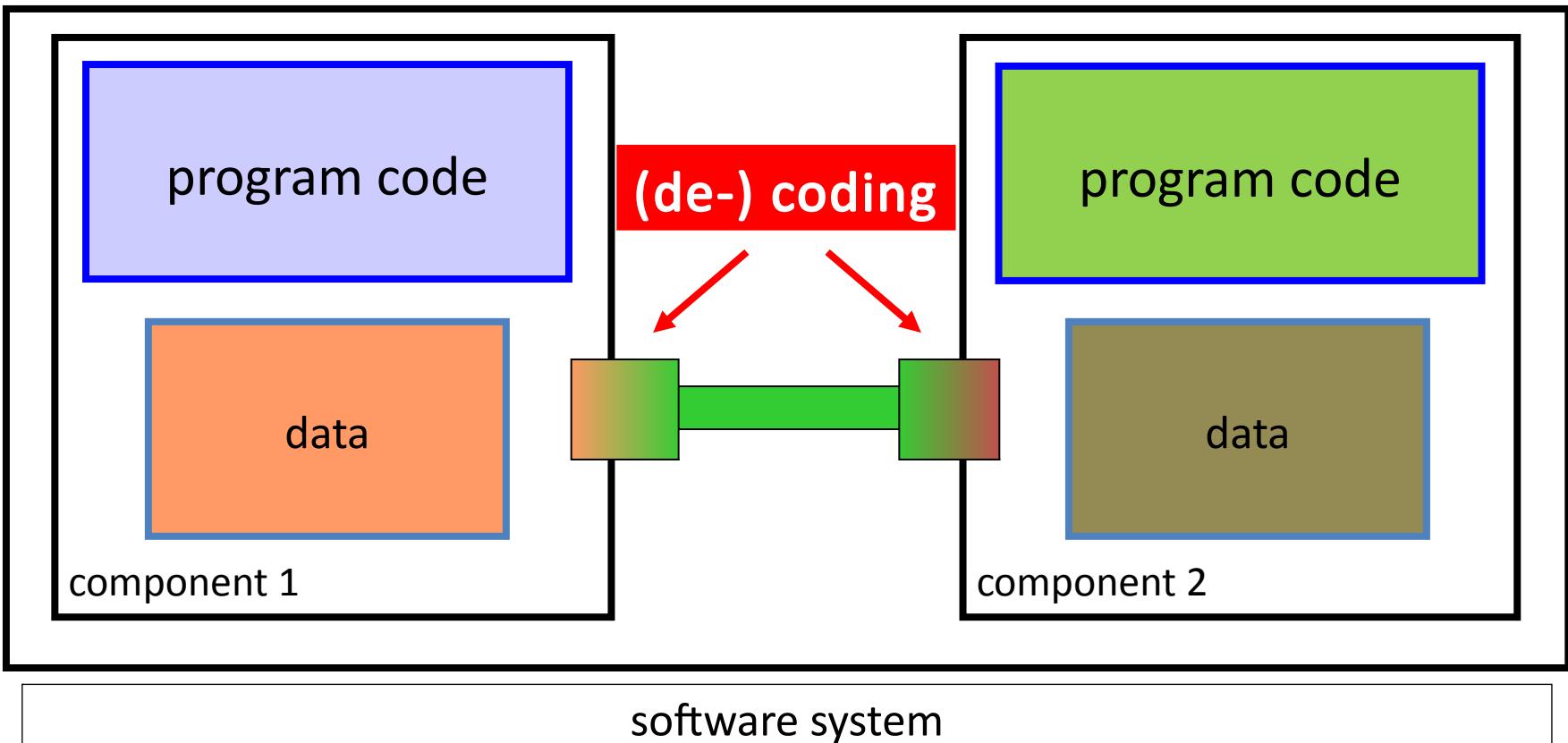
- Everything became bigger ...



XML in Communications

Informatik · CAU Kiel

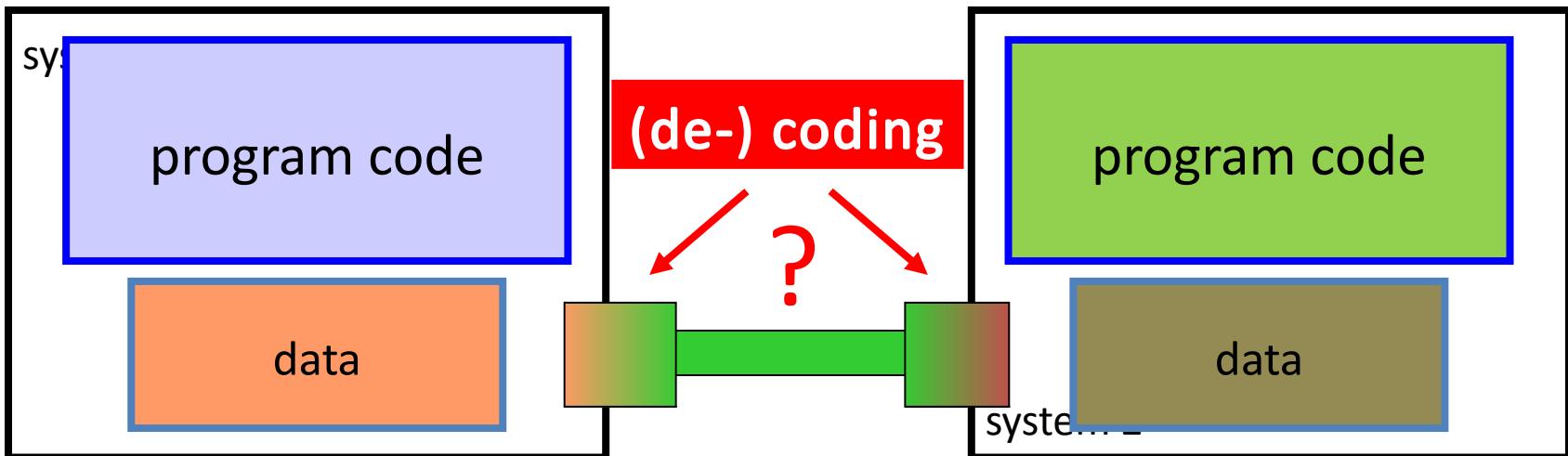
- ... and bigger



XML in Communications

Informatik · CAU Kiel

- ... and distributed



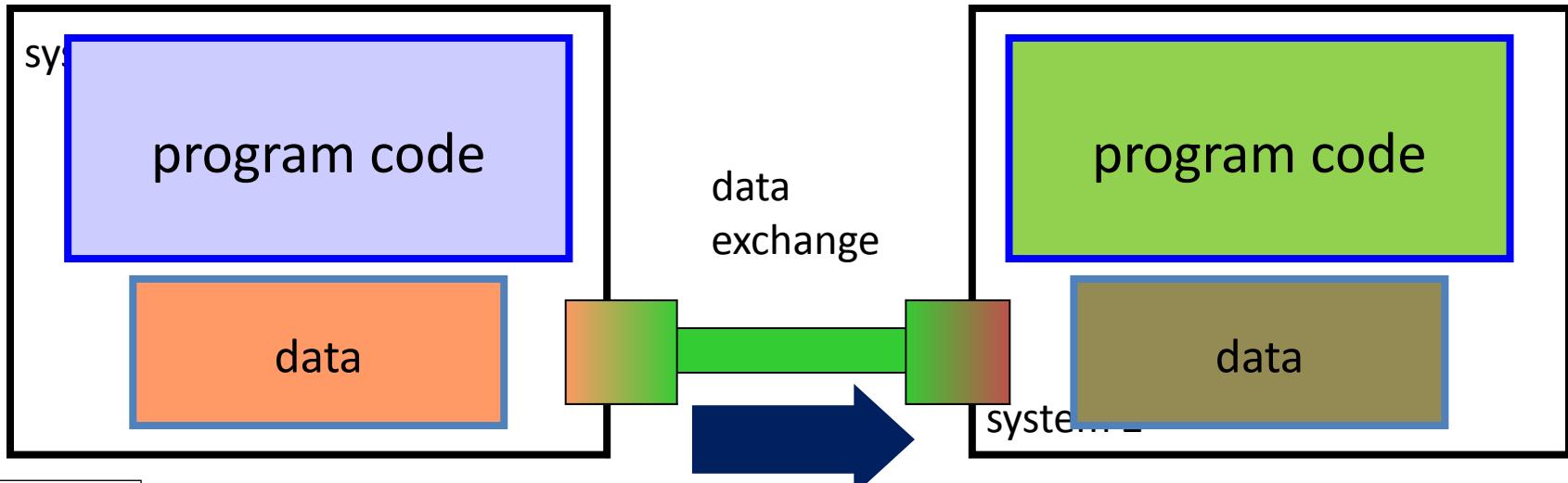
Big questions:

- How to code data?
- How to manage data coding?

XML in Communications

Informatik · CAU Kiel

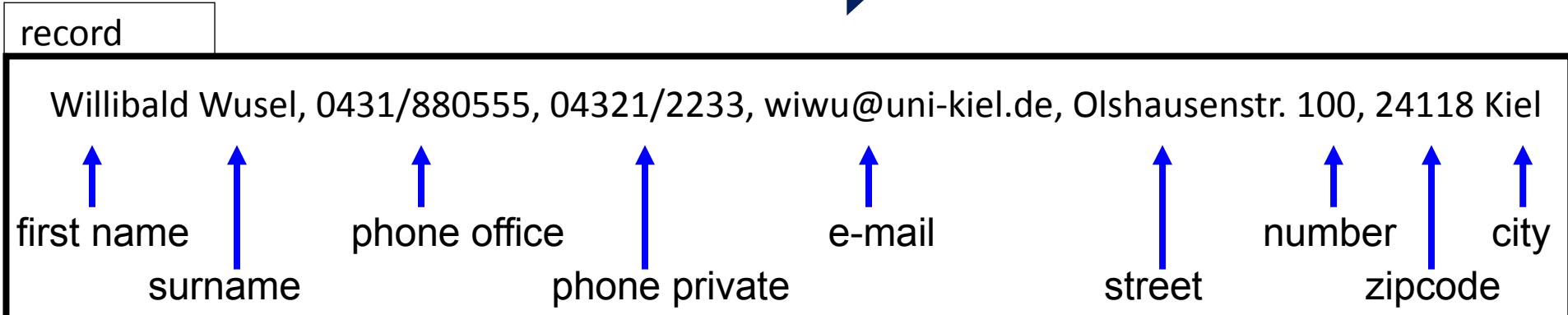
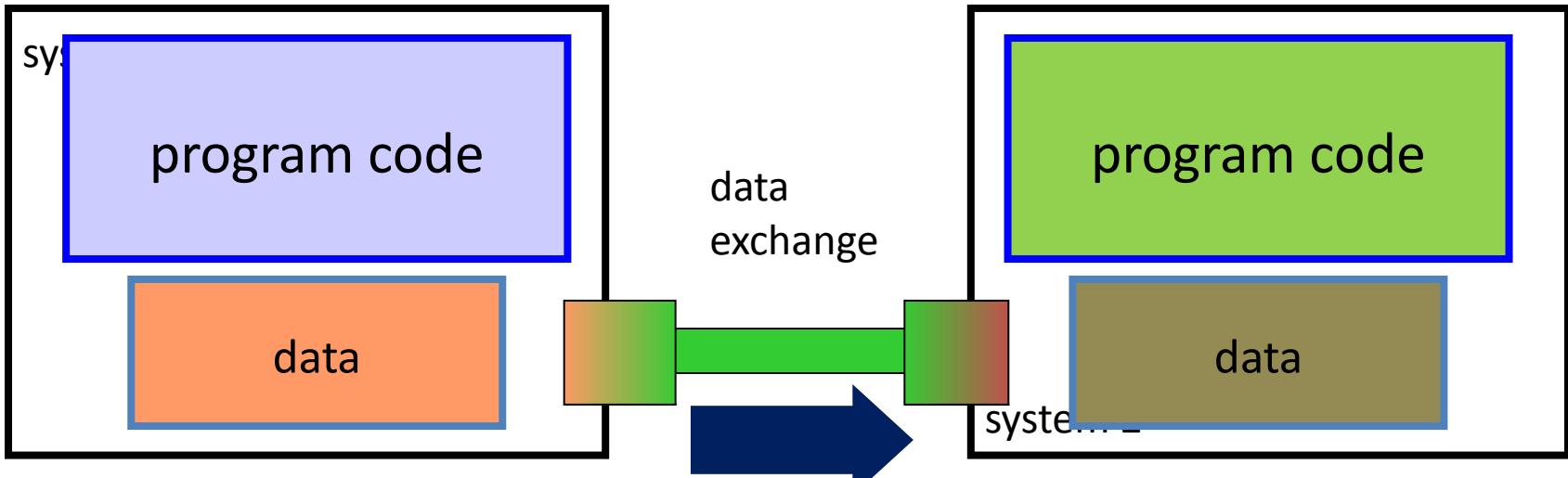
- Sample: record transmission



XML in Communications

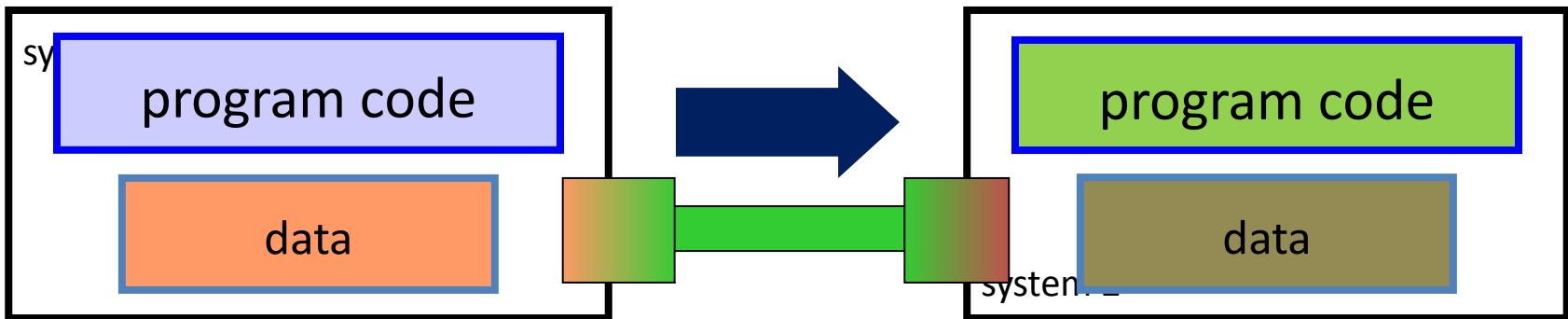
Informatik · CAU Kiel

- Record: data + structure



XML in Communications

- Record: data + structure



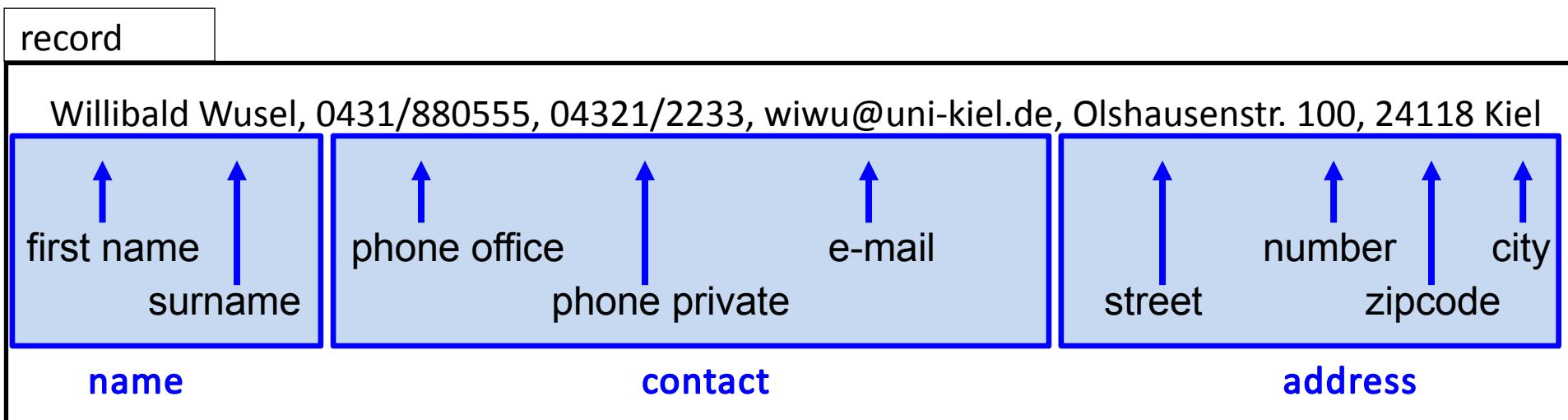
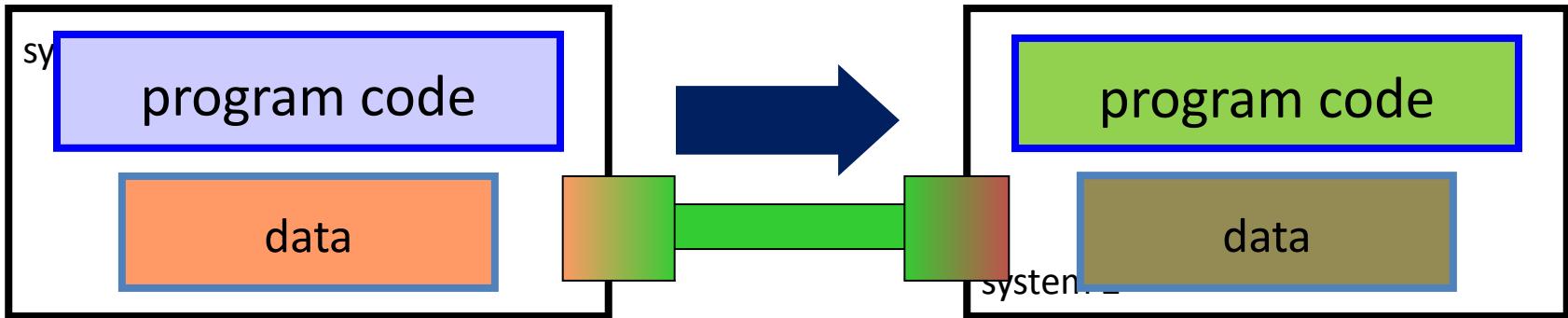
record

<first_name>	Willibald	</first_name>
<surname>	Wusel	</surname>
<phone_office>	0431/880555	</phone_office>
<phone_private>	04321/2233	</phone_private>
<e-mail>	wiwu@uni-kiel.de	</e-mail>
<street>	Olshausenstr.	</street>
<number>	100	</number>
<zipcode>	24118	</zipcode>
<city>	Kiel	</city>

XML in Communications

Informatik · CAU Kiel

- Nested structures



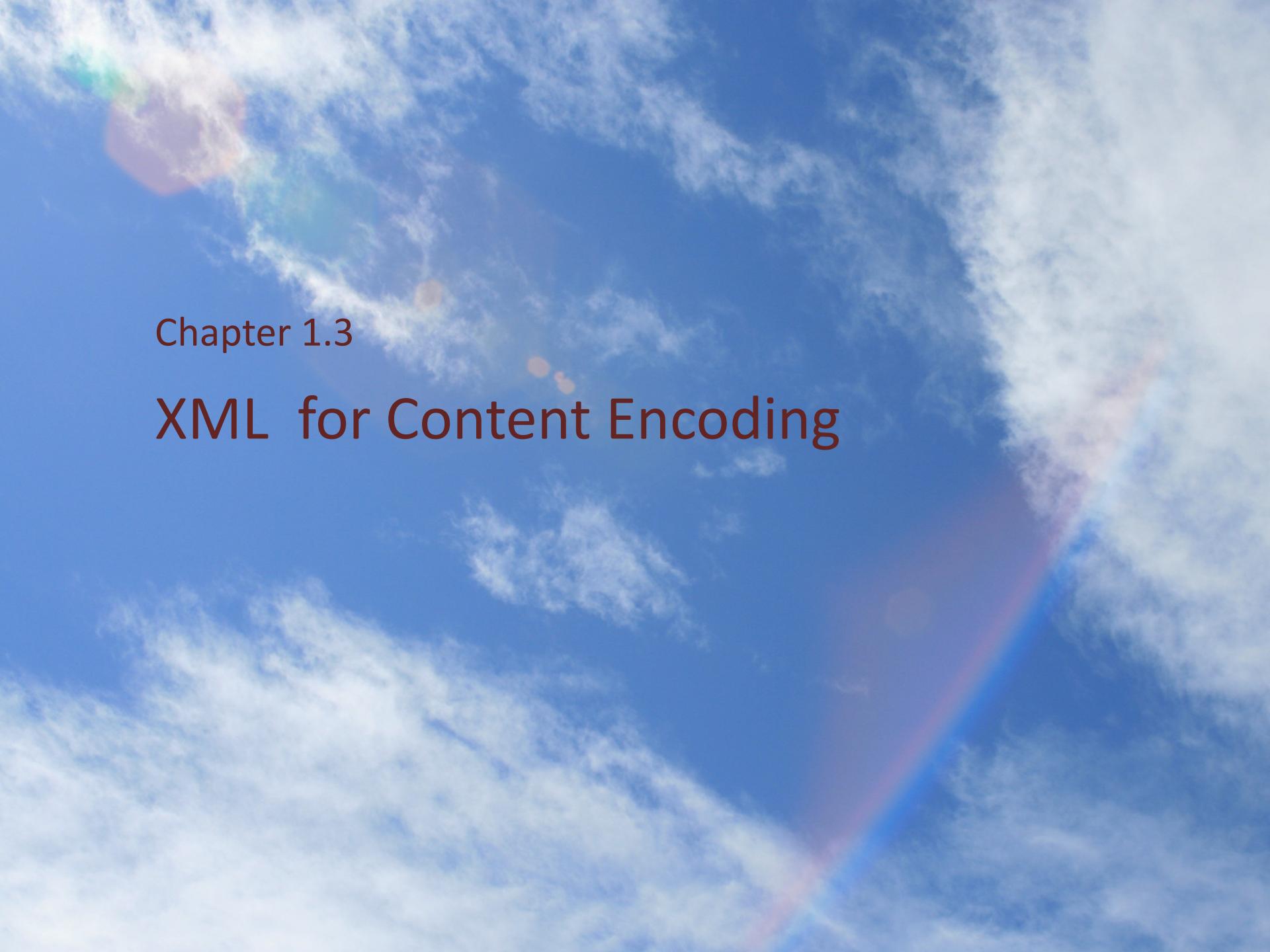
XML in Communications

Informatik · CAU Kiel

- Nested structures

record

```
<Name>
  <first_name> Willibald </first_name>
  <surname> Wusel </surname>
</Name>
<Contact>
  <phone_office> 0431/880555 </phone_office>
  <phone_private> 04321/2233 </phone_private>
  <e-mail> wiwu@uni-kiel.de </e-mail>
</Contact>
<Address>
  <street> Olshausenstr. </street>
  <number> 100 </number>
  <zipcode> 24118 </zipcode>
  <city> Kiel </city>
</Address>
```



Chapter 1.3

XML for Content Encoding

XML for Content Encoding

- Codd's idea of *logical-physical separation*
 - "The most important motivation for the research work ... was the objective of providing a sharp and clear boundary between the **logical and physical aspects** of database management."
 - E.F. Codd, Turing Award Lecture.
CACM 25 (2) (February 1982), 109-117
 - The same *logical* data gets a different *physical* encoding, e.g.
 - converting from an in-memory data structure to something that can be sent elsewhere: "marshalling"
 - converting from an in-memory data structure to something that can be stored on non-volatile memory: "persistance"

XML for Content Encoding

Informatik · CAU Kiel

- Special problem: Metadata for locating documents
 - Text documents: Key word search is great, don't need metadata.
 - But other kinds of content?
 - spreadsheets
 - maps
 - purchase records
 - objects
 - images
 - ...

XML for Content Encoding

- Metadata example: MP3 ID3 format – record at end of file

offset	length	description
0	3	"TAG" identifier string.
3	30	Song title string.
33	30	Artist string.
63	30	Album string.
93	4	Year string.
97	28	Comment string.
125	1	Zero byte separator.
126	1	Track byte.
127	1	Genre byte.

XML for Content Encoding

- Metadata example: JPEG "JFIF" header

- Start of Image (SOI) marker – two bytes (FFD8)
- JFIF marker (FFE0)
- length – two bytes
- ASCII code for zero-terminated "JFIF" string – 4A-46-49-46-00
- version – two bytes: often 01, 02
 - the most significant byte is used for major revisions
 - the least significant byte for minor revisions
- units – one byte: units for the X and Y densities
 - 0 => no units, X and Y specify the pixel aspect ratio
 - 1 => X and Y are dots per inch
 - 2 => X and Y are dots per cm
- X_{density} – two bytes
- Y_{density} – two bytes
- $X_{\text{thumbnail}}$ – one byte: 0 = no thumbnail
- $Y_{\text{thumbnail}}$ – one byte: 0 = no thumbnail
- $(\text{RGB})n$ – $3n$ bytes:
packed (24-bit) RGB values for the thumbnail pixels, $n = X_{\text{thumbnail}} * Y_{\text{thumbnail}}$

XML for Content Encoding

- Desiderata for data interchange
 - Ability to represent many kinds of information represented by different data structures
 - Hardware-independent encoding:
Endian-ness, UTF vs. ASCII vs. EBCDIC
 - Standard tools and interfaces
 - Ability to formally define grammar of expected data with forwards- and backwards-compatibility
- That's XML ...

A Few Common Uses of XML

Informatik · CAU Kiel

- Storage format for DTP tools: "extensible HTML"
 - e.g. MS Office, OpenOffice
 - supplemented with stylesheets (XSL) to define typographic layout
- Exchange format for data
 - marshalling data in Web Services
 - need to agree on terminology → ontology
- Notation for modeling languages
 - kind of Domain-Specific Language (DSL)
 - example: Geography Markup Language (GML)

```

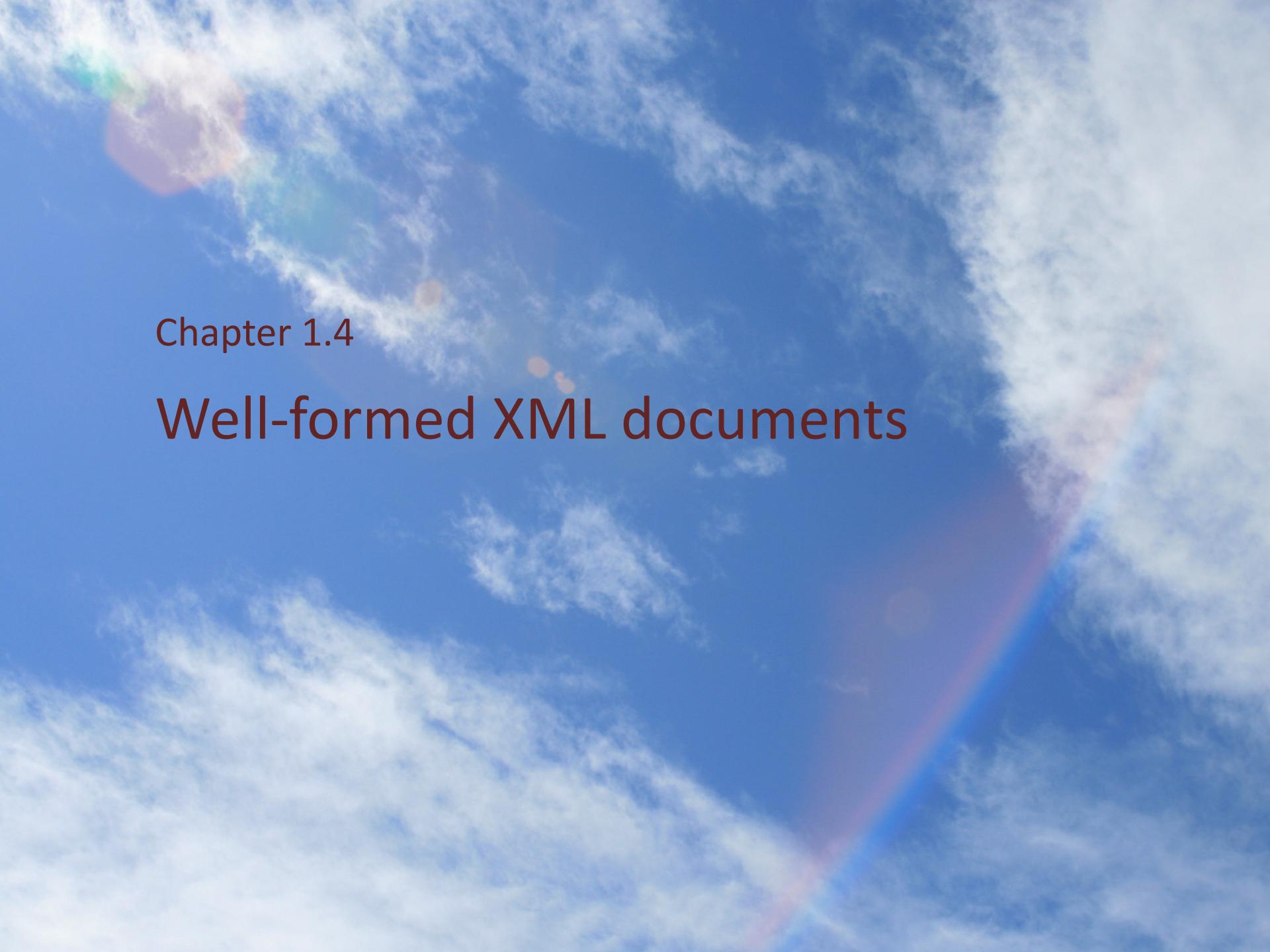
<p:sp>
  <p:nvSpPr>
    <p:cNvPr id="4" name="Foliennummernplatzhalter 3"/>
    <p:cNvSpPr>
      <a:spLocks noGrp="1"/>
    </p:cNvSpPr>
    <p:nvPr>
      <p:ph type="sldNum" sz="quarter" idx="11"/>
    </p:nvPr>
  </p:nvSpPr>
  <p:spPr/>
  <p:txBody>
    <a:bodyPr/>
    <a:lstStyle/>
    <a:p>
      <a:fld id="{25589156-C03E-44C9-A3C4-DD8DDD26385A}" type="slidenum">
        <a:rPr lang="de-DE" smtClean="0"/>
        <a:pPr/>
        <a:t>6</a:t>
      </a:fld>
      <a:endParaRPr lang="de-DE"/>
    </a:p>
  </p:txBody>
</p:sp>
<p:sp>
  <p:nvSpPr>
    <p:cNvPr id="5" name="Inhaltsplatzhalter 4"/>
    <p:cNvSpPr>
      <a:spLocks noGrp="1"/>
    </p:cNvSpPr>
    <p:nvPr>
      <p:ph sz="quarter" idx="12"/>
    </p:nvPr>
  </p:nvSpPr>
  <p:spPr/>
  <p:txBody>
    <a:bodyPr/>
    <a:lstStyle/>
    <a:p>
      <a:r>
        <a:rPr lang="de-DE" smtClean="0"/>
        <a:t>Hello world!</a:t>
      </a:r>
      <a:endParaRPr lang="de-DE"/>
    </a:p>
  </p:txBody>
</p:sp>

```

inside pptx
(snippet)

A Few Common Uses of XML

- Communities and business sectors are defining their specialized vocabularies
 - mathematics (MathML)
 - railway planning and operation (railML)
 - farm operation (agroML)
 - for further: see
http://en.wikipedia.org/wiki/List_of_XML_markup_languages



Chapter 1.4

Well-formed XML documents

Well-formed XML Documents

- A well-formed XML document consists of
 - a **prolog**
 - an **element**
 - an optional **epilog**
 - production:

document ::= (prolog element Misc*)



Well-formed XML Documents

- Prolog

```
prolog ::= XMLDecl Misc* (doctypedecl Misc*)?
```

- The XML declaration consists of

- Version info (required),
- Encoding declaration (optional),
- Standalone declaration (optional)

```
<?xml version="1.0" encoding="UTF-8" ?>
```



Well-formed XML Documents

Informatik · CAU Kiel

- Elements: the "things" the XML document talks about
 - E.g. books, authors, orders, invoices, tracks, signals, ...
- An element consists of:
 - an opening tag
 - the content
 - a closing tag

<lecturer> Jesper Zedlitz </lecturer>



Well-formed XML Documents

- Elements (cont'd.)
 - Tag names are formed following production 5 of the XML spec,
 - and additionally must not begin with the string "xml" in any combination of cases.

for completeness:

[5] **Name** ::= NameStartChar (NameChar)*

NameStartChar ::= ":" | [A-Z] | "_" | [a-z] | [#xC0-#xD6] | [#xD8-#xF6] | [#xF8-#x2FF] | [#x370-#x37D] | [#x37F-#x1FFF] | [#x200C-#x200D] | [#x2070-#x218F] | [#x2C00-#x2FEF] | [#x3001-#xD7FF] | [#xF900-#xFDCF] | [#xFDF0-#xFFFF] | [#x10000-#xEFFFF]

NameChar ::= NameStartChar | "-" | "." | [0-9] | #xB7 | [#x0300-#x036F] | [#x203F-#x2040]

Well-formed XML Documents

- Elements (cont'd.)
 - Element content may be text, or other elements, or a mixture thereof

```
<lecturer>Jesper Zedlitz</lecturer>  
  
<lecturer>  
    <name>Jesper Zedlitz</name>  
</lecturer>  
  
<lecturer>The lecturer of this lecture is  
    <name>Jesper Zedlitz</name></lecturer>
```



- If there is no content, then the element is called empty; it is abbreviated as follows:

`<lecturer/>` stands for `<lecturer></lecturer>`

Well-formed XML Documents

- Elements (cont'd.)
 - To attach **additional** information to an element, attributes can be used.
 - An attribute is a **name-value pair** separated by an equal sign (=).
 - An attribute is placed inside the **opening tag** of an element:

```
<lecturer name="Jesper Zedlitz"  
         phone="+49-431-880-7517" />
```



- An empty element is not necessarily meaningless:
It may have some properties in terms of **attributes**.

Well-formed XML Documents

- Example with/without attributes

```
<order orderNo="23456" customer="John Smith"  
       date="October 15, 2002">  
    <item itemNo="a528" quantity="1"/>  
    <item itemNo="c817" quantity="3"/>  
</order>
```

```
<order>  
    <orderNo>23456</orderNo>  
    <customer>John Smith</customer>  
    <date>October 15, 2002</date>  
    <item>  
        <itemNo>a528</itemNo>  
        <quantity>1</quantity> </item>  
    <item>  
        <itemNo>c817</itemNo>  
        <quantity>3</quantity> </item>  
</order>
```

Well-formed XML Documents

- Attributes vs. elements
 - Attributes are part of markup,
elements are part of the basic document contents.
 - Suggestion: use attributes for identifiers of elements,
and use elements for content.
 - But: When to use elements and when attributes is a matter of taste.
 - Keep in mind:
 - Attributes **must not** be nested!
 - No fixed order of appearance for attributes!

Well-formed XML Documents

Informatik · CAU Kiel

- Further components of XML docs

- Comments

- A piece of text that is to be ignored by the parser



```
<!-- This is a comment -->
```

- Processing Instructions (PIs)

- Define procedural attachments



```
<?stylesheet type="text/css" href="mystyle.css"?>
```

- Both allowed in epilog

- More kinds of components to come

Well-formed XML Documents

- Some syntactic rules for "well-formed" XML documents



One single outermost element.



Each element contains an opening and a corresponding closing tag.



Tags may not overlap.

```
<author> <name> Lee Hong </name> </author>
```



Element content may be characters, or other elements, or nothing.



Elements may have attributes given in the element start tag.



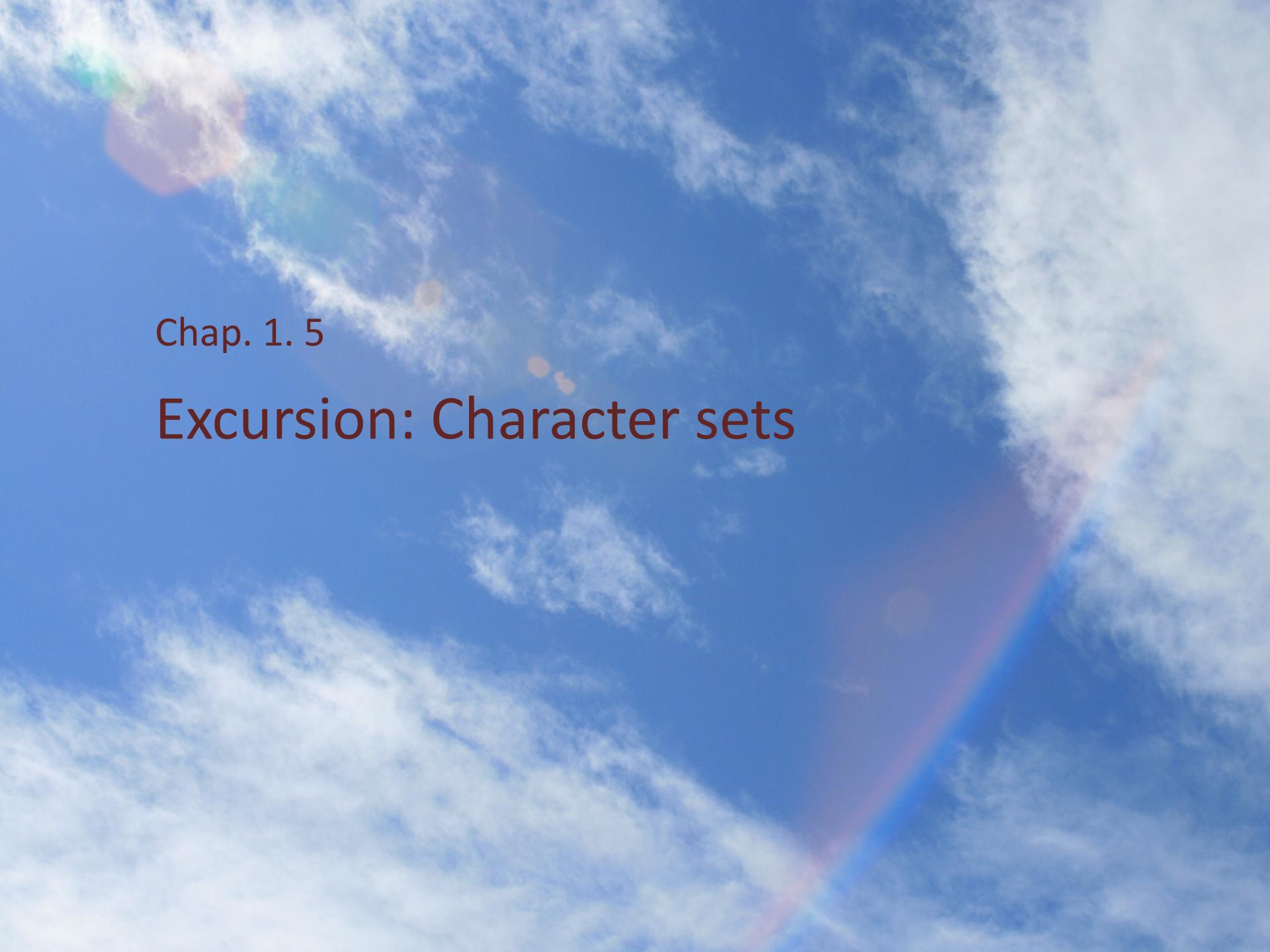
Attribute values must be put in parentheses.



XML is case-sensitive!



...



Chap. 1. 5

Excursion: Character sets

Character Sets

Informatik · CAU Kiel

Computers don't understand text, they only understand numbers. For computers to be able to treat text, there must be a correspondence between numbers and text characters. Such a correspondence is called a coded character set.

Thomas Krichel

Character Sets

- Important character sets
 - American Standard Code for Information Interchange (ASCII, US-ASCII)
 - ISO/IEC 646: International Reference Version (IRV); same as ASCII
 - ISO 8859 series of standards: 8-bit character encodings superseding the ISO 646 IRV and its national variants.
 - ISO 10646 standard: directly related to Unicode, superseding all of the ISO 646 and ISO 8859 sets of national-variant character encodings.

Character Sets

- Terminology
 - **Character repertoire**: a set of distinct characters.
 - **Character code**: a mapping from a character repertoire to a set of non-negative integers, called
 - **code points**
 - **code numbers**
 - **code values**, or
 - **code elements**.
 - **Character encoding**: an algorithm for mapping code points into sequences of octets for storage or transmission.

Character Sets

- Terminology example
 - ISO 10646

Repertoire	code point	16-bit encoding
LATIN SMALL LETTER A ("a")	U+0061 _{hex}	00 61
EXCLAMATION MARK ("!")	U+0021 _{hex}	00 21
LATIN SMALL LETTER A WITH DIAERESIS ("ä")	U+00E4 _{hex}	00 E4
PER MILLE SIGN ("%o")	U+2030 _{hex}	20 30

- U+(four hex numbers) denote code points in ISO 10646.

ASCII and IRV

- American Standard Code for Information Interchange
 - specifies 7-bit coding for space and 94 characters.
 - Character positions 0-31 and 127 are reserved for control codes.
 - Code 32 identifies a space.
 - Sequence:

```
! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @  
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _ `  
a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~
```

ASCII and IRV

- ASCII encoding

	0_	1_	2_	3_	4_	5_	6_	7_
_0	NUL	DLE	<sp>	0	@	P	'	p
_1	SOH	DC1	!	1	A	Q	a	q
_2	STX	DC2	"	2	B	R	b	r
_3	ETX	DC3	#	3	C	S	c	s
_4	EOT	DC4	\$	4	D	T	d	t
_5	ENQ	NAK	%	5	E	U	e	u
_6	ACK	SYN	&	6	F	V	f	v
_7	BEL	ETB	'	7	G	W	g	w
_8	BS	CAN	(8	H	X	h	x
_9	HT	EM)	9	I	Y	i	y
_A	LF	SUB	*	:	J	Z	j	z
_B	VT	ESC	+	;	K	[k	{
_C	FF	FS	,	<	L	\	l	
_D	CR	GS	-	=	M]	m	}
_E	SO	RS	.	>	N	^	n	~
_F	SI	US	/	?	O	-	o	DEL

ISO 8859

- ISO 8859 extends ASCII with characters for western European languages
 - extending USASCII's 7-bit encoding to 8-bit encoding
 - positions 128 to 159 not used
 - default character set of html

- Overview

Part 1	Latin 1 (West European)	
2	Latin 2 (East European)	Slavic, Albanian, Hungarian and a variation of Romanian
3	Latin 3 (South European)	Southern European languages (Maltese) and Esperanto
4	Latin 4 (North European)	Northern European languages
5	Cyrillic	
6	Arabic	
7	Greek	under revision (to include, among others, the euro sign)
8	Hebrew	
9	Latin 5 (Turkish)	covers characters used for Turkish, replacing those in Part 1 for Icelandic
10	Latin 6 (Nordic)	Icelandic, Nordic and Baltic character sets
11	Latin/Thai	is being worked on
12		
13	Latin 7	
14	Latin 8 (Celtic)	
15	Latin 9	also supports the euro sign
16	Latin 10	is being worked on to replace Part 2 for Romania and support some characters with comma below as opposed to with cedilla and also the euro sign

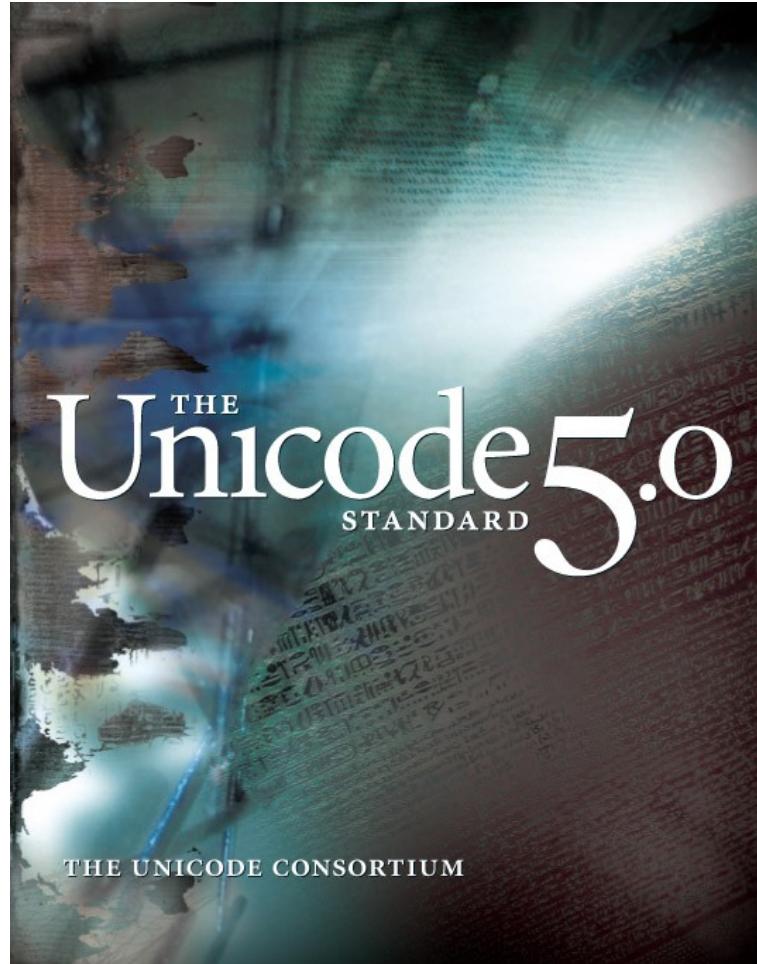
ASCII and IRV

- National variants

Zeichenposition:	23	24	40	5B	5C	5D	5E	60	7B	7C	7D	7E
ISO 646-IRV	#	¤	@	[\]	^	`	{		}	~
Deutschland	#	\$	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß
Schweiz	ù	\$	à	é	ç	ê	î	ô	ä	ö	ü	û
USA (ASCII)	#	\$	@	[\]	^	`	{		}	~
Großbritannien	£	\$	@	[\]	^	`	{		}	~
Frankreich	£	\$	à	°	ç	§	^	`	é	ù	è	"
Kanada	#	\$	à	â	ç	ê	î	ô	é	ù	è	û
Finnland	#	\$	@	Ä	Ö	Å	Ü	é	ä	ö	å	ü
Norwegen	#	\$	@	Æ	Ø	Å	Ü	é	æ	ø	å	~
Schweden	#	\$	É	Ä	Ö	Å	Ü	é	ä	ö	å	ü
Italien	£	\$	§	°	ç	é	^	`	ù	à	ò	ì
Niederlande	£	\$	¾	ÿ	½		^	`	"	f	¼	'
Spanien	£	\$	§	í	Ñ	¿	^	`	°	ñ	ç	~
Portugal	#	\$	@	Ã	ç	Õ	^	`	ã	ç	õ	~

Unicode

Informatik · CAU Kiel



Unicode

- Features
 - The Unicode Standard, Version 5.0 provides codes for 1,114,112 characters from the world's alphabets, ideograph sets, and symbol collections.
 - The Unicode Standard further includes punctuation marks, diacritics, mathematical symbols, technical symbols, arrows, dingbats, etc.
 - ISO/IEC 10646 has been widely adopted in new Internet protocols and implemented in modern operating systems and computer languages.

from:
www.unicode.org/standard/principles.html
and ISO/IEC 10646

What are "ideograph sets"?

(snippet from the Unicode standard)

Ideographic description characters

These are visibly displayed graphic characters, not invisible composition controls.

2FF0		IDEOGRAPHIC DESCRIPTION CHARACTER LEFT TO RIGHT
2FF1		IDEOGRAPHIC DESCRIPTION CHARACTER ABOVE TO BELOW
2FF2		IDEOGRAPHIC DESCRIPTION CHARACTER LEFT TO MIDDLE AND RIGHT
2FF3		IDEOGRAPHIC DESCRIPTION CHARACTER ABOVE TO MIDDLE AND BELOW
2FF4		IDEOGRAPHIC DESCRIPTION CHARACTER FULL SURROUND
2FF5		IDEOGRAPHIC DESCRIPTION CHARACTER SURROUND FROM ABOVE
2FF6		IDEOGRAPHIC DESCRIPTION CHARACTER SURROUND FROM BELOW
2FF7		IDEOGRAPHIC DESCRIPTION CHARACTER SURROUND FROM LEFT
2FF8		IDEOGRAPHIC DESCRIPTION CHARACTER SURROUND FROM UPPER LEFT
2FF9		IDEOGRAPHIC DESCRIPTION CHARACTER SURROUND FROM UPPER RIGHT
2FFA		IDEOGRAPHIC DESCRIPTION CHARACTER SURROUND FROM LOWER LEFT
2FFB		IDEOGRAPHIC DESCRIPTION CHARACTER OVERLAI

Unicode

- Universal Character Set (UCS)
 - ISO 10646 formally defines a 31-bit character set.
 - Characters are represented as 32 bits, i.e. 4 bytes, or 8 hex chars (MSB = 0)
 - Four-dimensional coding space:
 - consisting of 128 groups
 - per group: 256 planes
 - per plane: 256 rows, each having 256 cells.

30	23	15	7	
24	16	8	0	
0	Group	Plane	row	cell

- **Planes of the Universal Character Set (UCS)**
 - **Plane 0:** Basic Multilingual Plane (BMP) contains the common-use characters for all the modern scripts of the world as well as many historical and rare characters.
 - **Plane 1:** Supplementary Multilingual Plane (SMP) is dedicated to the encoding of lesser-used historic scripts, special-purpose invented scripts, and special notational systems.
 - **Plane 2:** Supplementary Ideographic Plane (SIP) is intended as an additional allocation area for CJK characters.
 - **Plane 14:** Supplementary Special-purpose Plane (SSP) is the spillover allocation area for format control characters.
 - **Planes 15 and 16:** Private Use Planes. The two Private Use Planes are allocated, in their entirety, for private use.

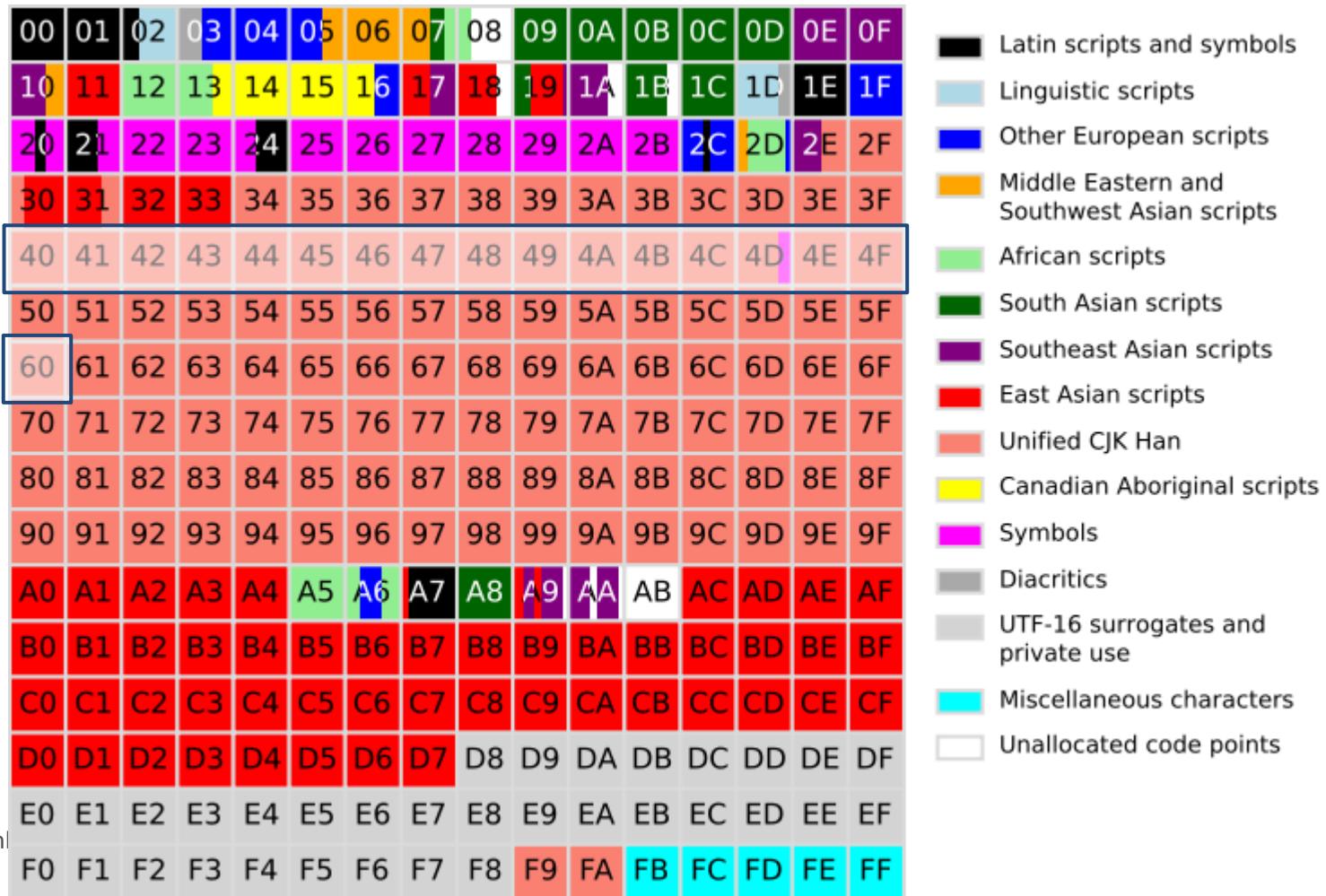
Unicode

- Planes of the Universal Character Set (UCS)
 - Group 0, Plane 0: Basic Multilingual Plane, BMP
 - The 4-octet representation of a character in the BMP is produced by putting two 0 octets before its 2-octet representation.
 - The UCS characters U+0000 to U+007F are identical to those in ASCII.
 - The range U+0000 to U+00FF is identical to ISO 8859-1 (Latin-1).

Unicode

- The Unicode Basic Multilingual Plane (BMP)

row
cell
256 chars
per cell!



Unicode

- Examples

Codepoint	Title	Block
Eintrag in:		
%‰	decode : [U+2030] PER MILLE SIGN	
黍	decode : [U+2FC9] KANGXI RADICAL MILLET	nt
%	decode : [U+0025] PERCENT SIGN	nt
秩	decode : [U+412E]	nt
秆	decode : [U+4130]	nt
𢃥	decode : [U+4136]	nt
穀	decode : [U+413E]	nt
穡	decode : [U+4142]	nt
穧	decode : [U+4147]	nt
穨	decode : [U+4155]	nt
穩	decode : [U+415F]	nt
穪	decode : [U+4163]	nt
穮	decode : [U+416D]	nt
穯	decode : [U+416F]	

Unicode

- Encodings
 - UTF: Unicode Transformation Format
 - UTF-8
 - UCS characters U+0000 to U+007F (ASCII) are encoded as bytes 0x00 to 0x7F.
 - All other UCS characters are encoded as a sequence of bytes, each of which has the most significant bit set.
 - UTF-16
 - Coding of characters in the BMP; encoding equals code point

Unicode

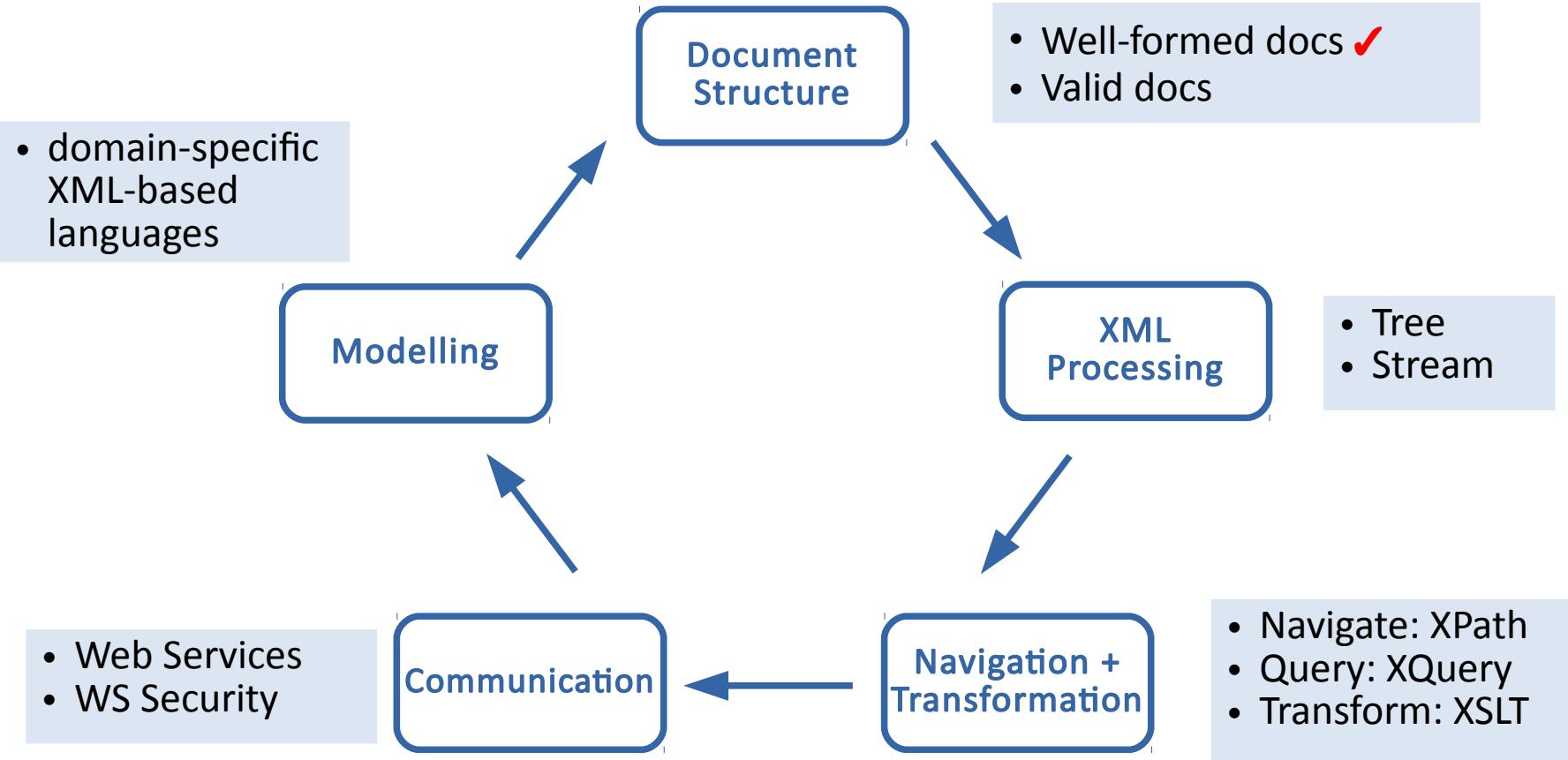
- Encodings
 - Byte order for UTF-16:
 - Unicode files start with the character U+FEFF (ZERO WIDTH NO-BREAK SPACE), also known as the **Byte-Order Mark (BOM)**.
 - Big-endians deliver: FE FF
 - Little-endian deliver: FF FE,
which is not a valid Unicode character
 - BOM for UTF-8: EF BB BF



Chap. 1.6

An Initial »Shopping List«

»Shopping List«



Grammars for XML

XML Information Set

Lecture "XML in Communication Systems"
Chapter 2

Dr.-Ing. Jesper Zedlitz
Research Group for Communication Systems
Dept. of Computer Science
Christian-Albrechts-University in Kiel



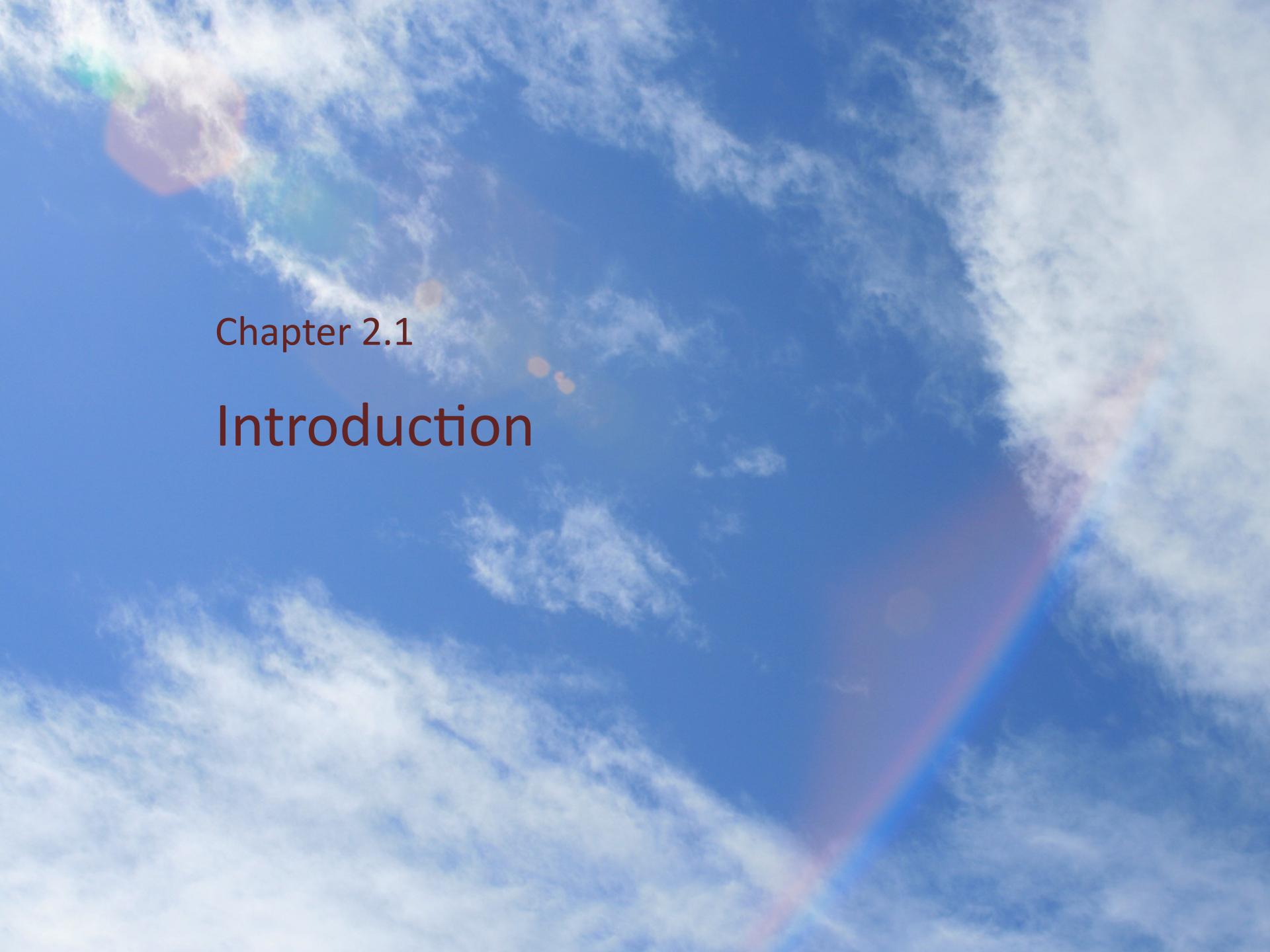
Recommended Reading

- T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, J. Cowan (Eds.):
XML 1.1 (Second Edition), W3C Recommendation, 16 August 2006.
<http://www.w3.org/TR/xml11>
- J. Cowan, R. Tobin (Eds.):
XML Information Set (Second Edition), W3C Recommendation, 4 February 2004.
<http://www.w3.org/TR/xml-infoset>

Overview

Informatik · CAU Kiel

1. Introduction
2. Information items
3. Information item names (Namespaces)
4. Further information item properties



Chapter 2.1

Introduction

Introduction

- What is XML Information Set?
 - A specification of abstract data structures describing the content of well-formed XML documents accessible to applications.
 - A specification describing the "output" of XML processors.
 - A W3C recommendation.

Introduction

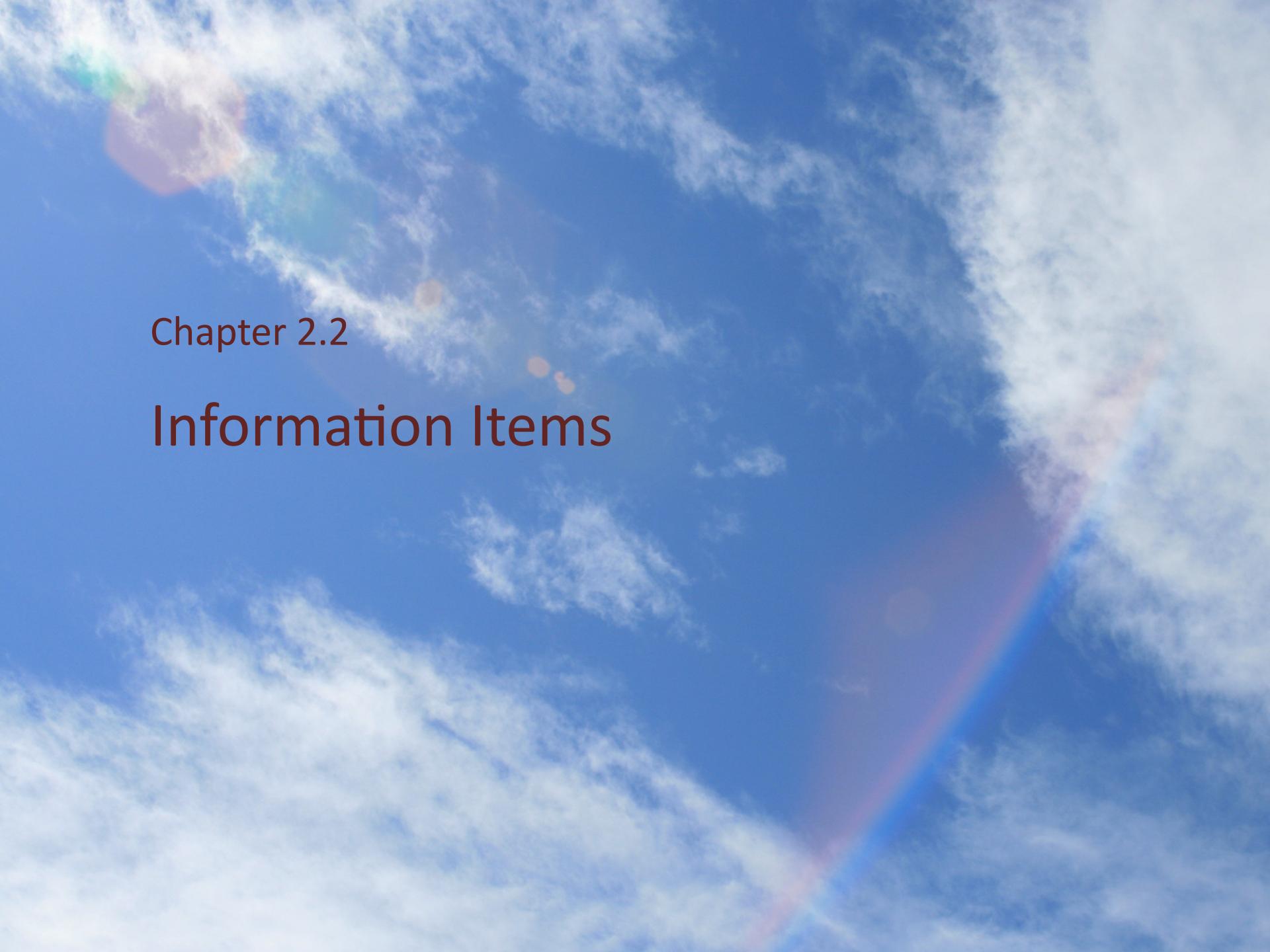
- From the XML specification:
 - "Each XML document has both a **logical** and a **physical** structure.
 - **Logically**, the document is composed of ..., **elements**, **comments**, ... and **processing instructions**, all of which are indicated in the document by explicit markup. (More "logical units" to come later; NL)
 - **Physically**, the document is composed of [storage] units called **entities**. ..."



What is the **logical** structure of XML documents?

Introduction

- What is an XML infoset?
 - "An XML document's information set consists of a number of **information items**; the information set for any well-formed XML document will contain at least a **document information item** and several others. ... each information item has a set of associated named **properties**."

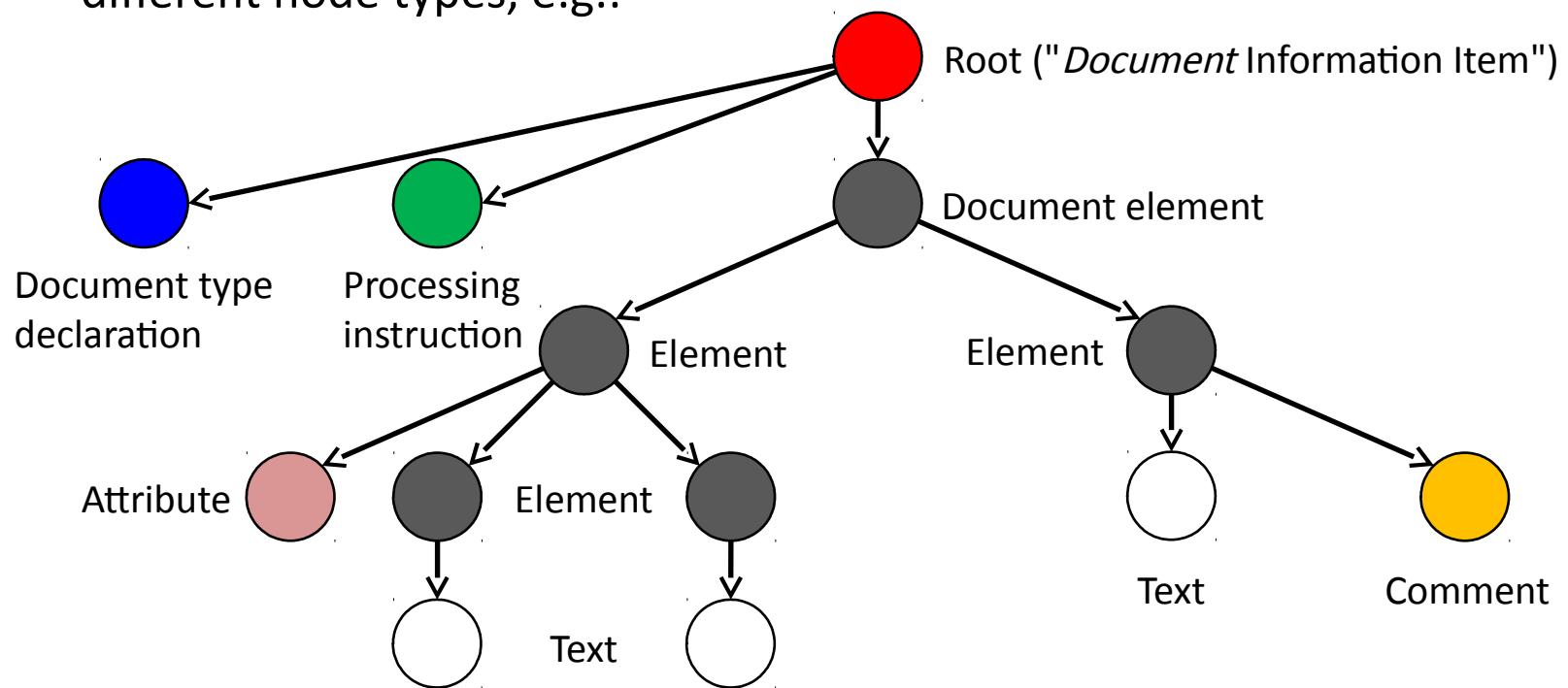


Chapter 2.2

Information Items

Information Items

- XML document
 - attributed, tree-like graph
 - different node types, e.g.:



Information Items

- Types of information items
 - Document Information Item
 - Element Information Items
 - Attribute Information Items
 - Processing Instruction Information Items
 - Unexpanded Entity Reference Information Items
 - Character Information Items
 - Comment Information Items
 - Document Type Declaration Information Item
 - Unparsed Entity Information Items
 - Notation Information Items
 - Namespace Information Items

We skip most of these!

Information Items

- Types of information items



Document Information Item

Element Information Item

- Attribute Information Items
- Processing Instruction Items
- Unexpanded Entity Reference Information Items
- Character Information Items
- Comment Information Items
- Document Type Declaration Information Item
- Unparsed Entity Information Items
- Notation Information Items
- Namespace Information Items

Is the root of the document: It has exactly one child *element information item*: the *document element information item*.
Precisely: The value of the [document element] property of the *document information item* is a single *element information item*. This is called the *document element information item*.

Excursion: Terminology

- Inconsistent terminology between the DOM, XML, and XPath specs
 - "There is confusion between the terms **top level (document) element** (which is an element node) with **root** of the document (which is not)."
 - Clarification:
 - The **root** represents the document itself.
 - The **document element** is the element that has all other elements within the document as descendants.
 - The **root** can have other children besides the document element, e.g. comments and processing instructions.

Information Items

- Types of information items

- Document Information Item



- Element Information Items

- Attribute Information

- Processing Instruction

- Unexpanded Entity Reference Information Items

- Character Information Items

- Comment Information Items

- Document Type Declaration Information Item

- Unparsed Entity Information Items

- Notation Information Items

- Namespace Information Items

There is an *element information item* for each element appearing in the XML document.

Information Items

- Types of information items

- Document Information Item
- Element Information Items



Attribute Information Items

Processing Instruction Information Items

- Unexpanded Entity Reference Information Items
- Character Information Items
- Comment Information Items
- Document Type Declaration Information Items
- Unparsed Entity Information Items
- Notation Information Items
- Namespace Information Items

There is an *attribute information item* for each attribute (specified or defaulted) of each element in the document, including those which are namespace declarations. The latter however appear as members of an element's [namespace attributes] property rather than its [attributes] property.

Information Items

- Types of information items

- Document Information Item
- Element Information Items
- Attribute Information Items
- Processing Instruction Items
- Unexpanded Entity References
- Character Information Items
- Comment Information Items



There is a *character information item* for each data character that appears in the document ... Each character is a logically separate information item, but XML applications are free to chunk characters into larger groups as necessary or desirable. (In this lecture, we prefer a *text information item*.)

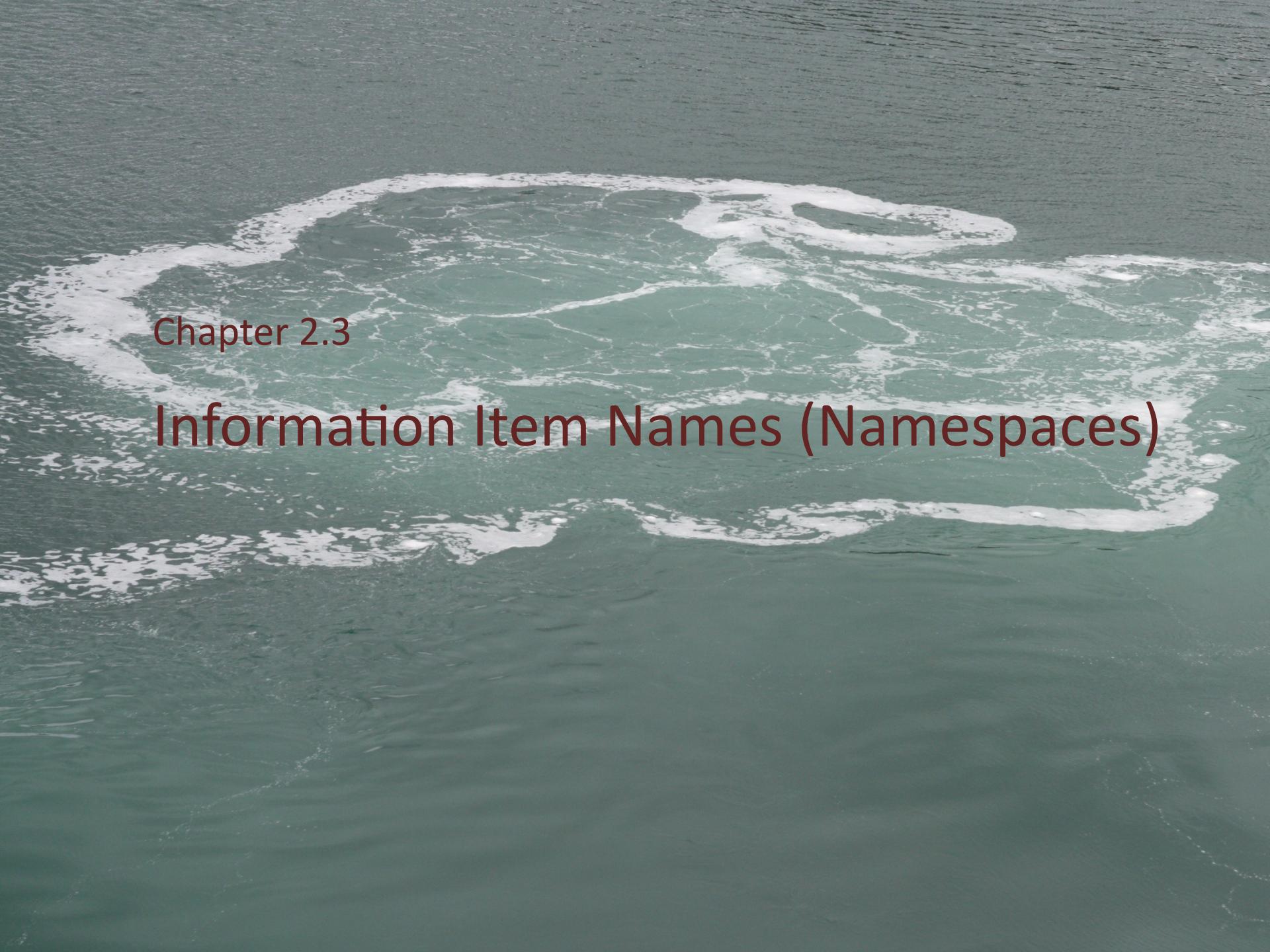
- Document Type Declaration Information Item
- Unparsed Entity Information Items
- Notation Information Items
- Namespace Information Items

Information Items

- Types of information items
 - Document Information Item
 - Element Information Items
 - Attribute Information Items
 - Processing Instruction Information Items
 - Unexpanded Entity Reference Information Items
 - Character Information Items
 - Comment Information Items
 - Document Type Declaration Information Item
 - Unparsed Entity Information Items
 - Notation Information Items
 - Namespace Information Items

Each element in the document has a namespace information item for each namespace that is in scope for that element.



The background image shows an aerial view of a large, circular, white, foamy wake or plume in dark green water, resembling a brain or a map.

Chapter 2.3

Information Item Names (Namespaces)

Information Item Names

- Remember:
 - "An XML document's information set consists of a number of **information items**; the information set for any well-formed XML document will contain at least a document information item and several others. ... each information item has a set of associated named **properties**."
- Three important name-related properties
 - local name
 - expanded name
 - namespace URI

Information Item Names

- Problem
 - How to provide unique names,
when "mixing" XML documents?

```
<?xml version="1.0" encoding="UTF-8"?>
<Book>
    <ISBN>0743204794</ISBN>
    <author>Kevin Davies</author>
    <title>Cracking the Genome</title>
    <price>20.00</price>
</Book>
```

"local
names"

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
    <head>
        <title>My home page</title>
    </head>
    <body>
        <p>My hobby</p>
        <p>My books</p>
    </body>
</html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
  <head>
    <title>My home page</title>
  </head>
  <body>
    <p>My hobby</p>
    <p>My books <img alt="orange arrow pointing to the word books" data-bbox="388 385 485 435">
      <Book>
        <ISBN>0743204794</ISBN>
        <author>Kevin Davies</author>
        <title>Cracking the Genome</title>
        <price>20.00</price>
      </Book>
    </p>
  </body>
</html>
```

Remark: This is called "mixed content"—text content and element content

Both XML validator and application program need context information to distinguish between *HTML page title* and *book title*!

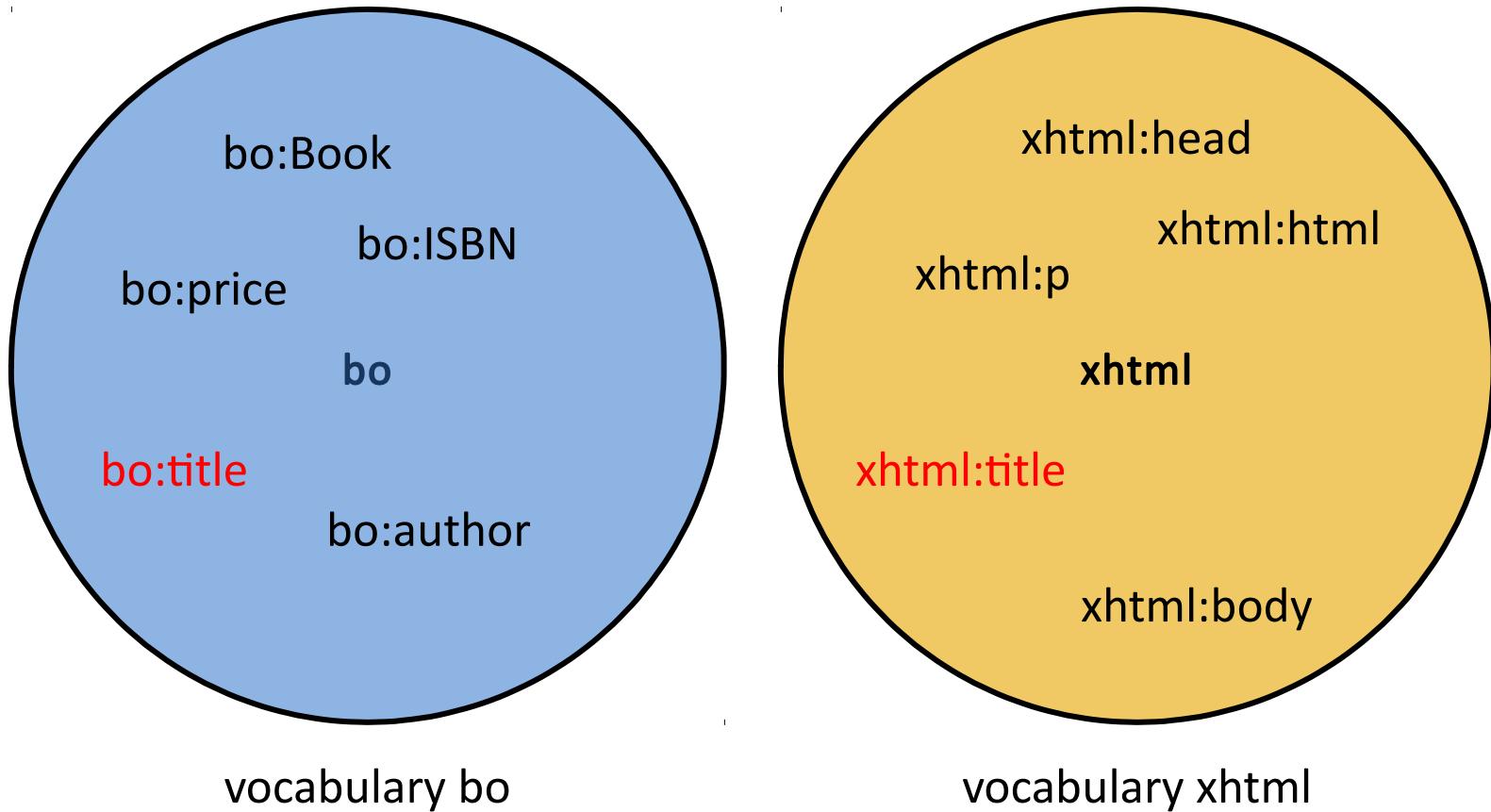
Namespaces

- How the web works
 - Individually created documents
 - Distributed creation of knowledge and lazy integration
 - Problem: Vocabulary collisions!
- Two-step solution
 1. Expand local names by locally unique name prefixes
 2. Bind local prefixes to globally unique URIs

```
<?xml version="1.0" encoding="UTF-8"?>
<xhtml:html>
  <xhtml:head>
    <xhtml:title>My home page</xhtml:title>
  </xhtml:head>
  <xhtml:body>
    <xhtml:p>My hobby</xhtml:p>
    <xhtml:p>My books
      <bo:Book>
        <bo:ISBN>0743204794</bo:ISBN>
        <bo:author>Kevin Davies</bo:author>
        <bo:title>Cracking the Genome</bo:title>
        <bo:price>20.00</bo:price>
      </bo:Book>
    </xhtml:p>
  </xhtml:body>
</xhtml:html>
```

locally unique
"expanded names"

Namespaces



- But who guarantees uniqueness of prefixes?

Namespaces

- Give prefixes only local relevance in an instance document
 - Associate local prefix with global namespace name:
a unique name for a namespace
 - Uniqueness is guaranteed by using a URI (preferably URN)
in domain of the party creating the namespace.
 - URI doesn't have any meaning,
i.e. doesn't have to resolve into anything.



An XML namespace is a collection of names,
identified by a URI reference, which are used in
XML documents as element and attribute names.

```
<?xml version="1.0" encoding="UTF-8"?>
<xhtml:html
    xmlns:xhtml="http://www.w3c.org/1999/xhtml"
    xmlns:bo="http://www.nogood.com/Book">
    <xhtml:head>
        <xhtml:title>My home page</xhtml:title>
    </xhtml:head>
    <xhtml:body>
        <xhtml:p>My hobby</xhtml:p>
        <xhtml:p>My books
            <bo:Book>
                <bo:ISBN>0743204794</bo:ISBN>
                <bo:author>Kevin Davies</bo:author>
                <bo:title>Cracking the Genome</bo:title>
                <bo:price>20.00</bo:price>
            </bo:Book>
        </xhtml:p>
    </xhtml:body>
</xhtml:html>
```

"namespaces"

```
<?xml version="1.0" encoding="UTF-8"?>
<xhtml:html
  xmlns:xhtml="http://www.w3c.org/1999/xhtml">
  <xhtml:head>
    <xhtml:title>My home page</xhtml:title>
  </xhtml:head>
  <xhtml:body>
    <xhtml:p>My hobby</xhtml:p>
    <xhtml:p>My books
      <bo:Book
        xmlns:bo="http://www.nogood.com/Book">
        <bo:ISBN>0743204794</bo:ISBN>
        <bo:author>Kevin Davies</bo:author>
        <bo:title>Cracking the Genome</bo:title>
        <bo:price>20.00</bo:price>
      </bo:Book>
    </xhtml:p>
  </xhtml:body>
</xhtml:html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xhtml:html
  xmlns:xhtml="http://www.w3c.org/1999/xhtml">
  <xhtml:head>
    <xhtml:title>My home page</xhtml:title>
  </xhtml:head>
  <xhtml:body>
    <xhtml:p>My hobby</xhtml:p>
    <xhtml:p>My books
      <Book
        xmlns="http://www.nogood.com/Book">
        <ISBN>0743204794</ISBN>
        <author>Kevin Davies</author>
        <title>Cracking the Genome</title>
        <price>20.00</price>
      </Book>
    </xhtml:p>
  </xhtml:body>
</xhtml:html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<html
  xmlns="http://www.w3c.org/1999/xhtml">
  <head>
    <title>My home page</title>
  </head>
  <body>
    <p>My hobby</p>
    <p>My books
      <Book
        xmlns="http://www.nogood.com/Book">
        <ISBN>0743204794</ISBN>
        <author>Kevin Davies</author>
        <title>Cracking the Genome</title>
        <price>20.00</price>
      </Book>
    </p>
  </body>
</html>
```

Namespaces

- What do namespace URI's point to?
 - The "abstraction" camp
 - A namespace URI is the id for a concept, it shouldn't resolve to anything
 - The "orthodox" camp
 - It should resolve to a schema (xml schema)
 - The "liberal" camp
 - It should resolve to many things



Chapter 2.4

Further Information Item Properties

Information Item Properties

- Properties depend on type of information item
 - e.g. element information item
 - [namespace name], [local name], [prefix]
 - [children] An ordered list of child information items, in document order. This list contains element, processing instruction, unexpanded entity reference, character, and comment information items ...
 - [attributes] An unordered set of attribute information items, one for each of the attributes ...
 - [namespace attributes] An unordered set of attribute information items, one for each of the namespace declarations
 - [base URI] The base URI of the element.
 - [parent] The document or element information item which contains this information item in its [children] property.

Information Item Properties

- Properties depend on type of information item
 - e.g. attribute information item
 - [namespace name], [local name], [prefix]
 - [normalized value] The normalized attribute value
 - [specified] A flag indicating whether this attribute was actually specified in the start-tag of its element, or was defaulted
 - [attribute type] An indication of the type declared for this attribute in the DTD. Legitimate values are ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS, NOTATION, CDATA, and ENUMERATION. ... Applications should treat no value and unknown as equivalent to a value of CDATA. The value of this property is not affected by the validity of the attribute value.
 - [references] For attributes typed IDREF, IDREFS, ENTITY, ENTITIES, or NOTATION
 - [owner element] The element information item which contains this information item in its [attributes] property.

Grammars for XML

Document Type Definitions

Lecture "XML in Communication Systems"
Chapter 3

Dr.-Ing. Jesper Zedlitz
Research Group for Communication Systems
Dept. of Computer Science
Christian-Albrechts-University in Kiel



Recommended Reading

- T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, J. Cowan (Eds.):
XML 1.1 (Second Edition), W3C Recommendation, 16 August 2006.
<http://www.w3.org/TR/xml11>

Document Type Definition

- Snippet from an XML document

```
<Person>
  <Name>
    <first_name>Willibald</first_name>
    <surname>Wusel</surname>
  </Name>
  <Contact>
    <phone_office>0431/880555</phone_office>
    <phone_private>04321/2233</phone_private>
    <e-mail>wiwu@uni-kiel.de</e-mail>
  </Contact>
  <Address>
    <street>Olshausenstr.</street>
    <number>100</number>
    <zipcode>24118</zipcode>
    <city>Kiel</city>
  </Address>
</Person>
```

Document Type Definition

- Related DTD snippet

```
<!ELEMENT Person (Name, Contact, Address)>           document element

<!ELEMENT Name (first_name, surname)>                sequence
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT surname (#PCDATA)>

<!ELEMENT Contact (phone_office | phone_private | e-mail)*> choice
<!ELEMENT phone_office (#PCDATA)>
<!ELEMENT phone_private (#PCDATA)>
<!ELEMENT e-mail (#PCDATA)>

<!ELEMENT Address (street, number, zipcode, city)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT number (#PCDATA)>                         "type":
<!ELEMENT zipcode (#PCDATA)>                        parsed character data
<!ELEMENT city (#PCDATA)>
```

Document Type Definition

- **Attributes**

```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>
<!ATTLIST Book
    Category (autobiography | non-fiction | fiction) #REQUIRED
    InStock (true | false) "false"
    Reviewer CDATA " "
  <!ELEMENT Title (#PCDATA)>
  <!ELEMENT Author (#PCDATA)>
  <!ELEMENT Date (#PCDATA)>
  <!ELEMENT ISBN (#PCDATA)>
  <!ELEMENT Publisher (#PCDATA)>
```

Document Type Definition

- Where to find the DTD
 - DTD may be enclosed in XML docs.
 - XML document may hold reference to ist DTD in its prologue.

```
<?xml version = "1.0" ?>
<!DOCTYPE Staff SYSTEM "http://myserver.com/staff.dtd">

<Staff>
  <Person>
    ...
  </Person>
</Staff>
```

Grammars for XML Documents

XML Schema, Part 1

Lecture "XML in Communication Systems"
Chapter 4

Dr.-Ing. Jesper Zedlitz
Research Group for Communication Systems
Dept. of Computer Science
Christian-Albrechts-University in Kiel



Acknowledgement

Informatik · CAU Kiel

- This chapter is based on:

Roger L. Costello: XML Technologies Course

<http://www.xfront.com/files/tutorials.html>

Copyright (c) 2000. Roger L. Costello. All Rights Reserved.

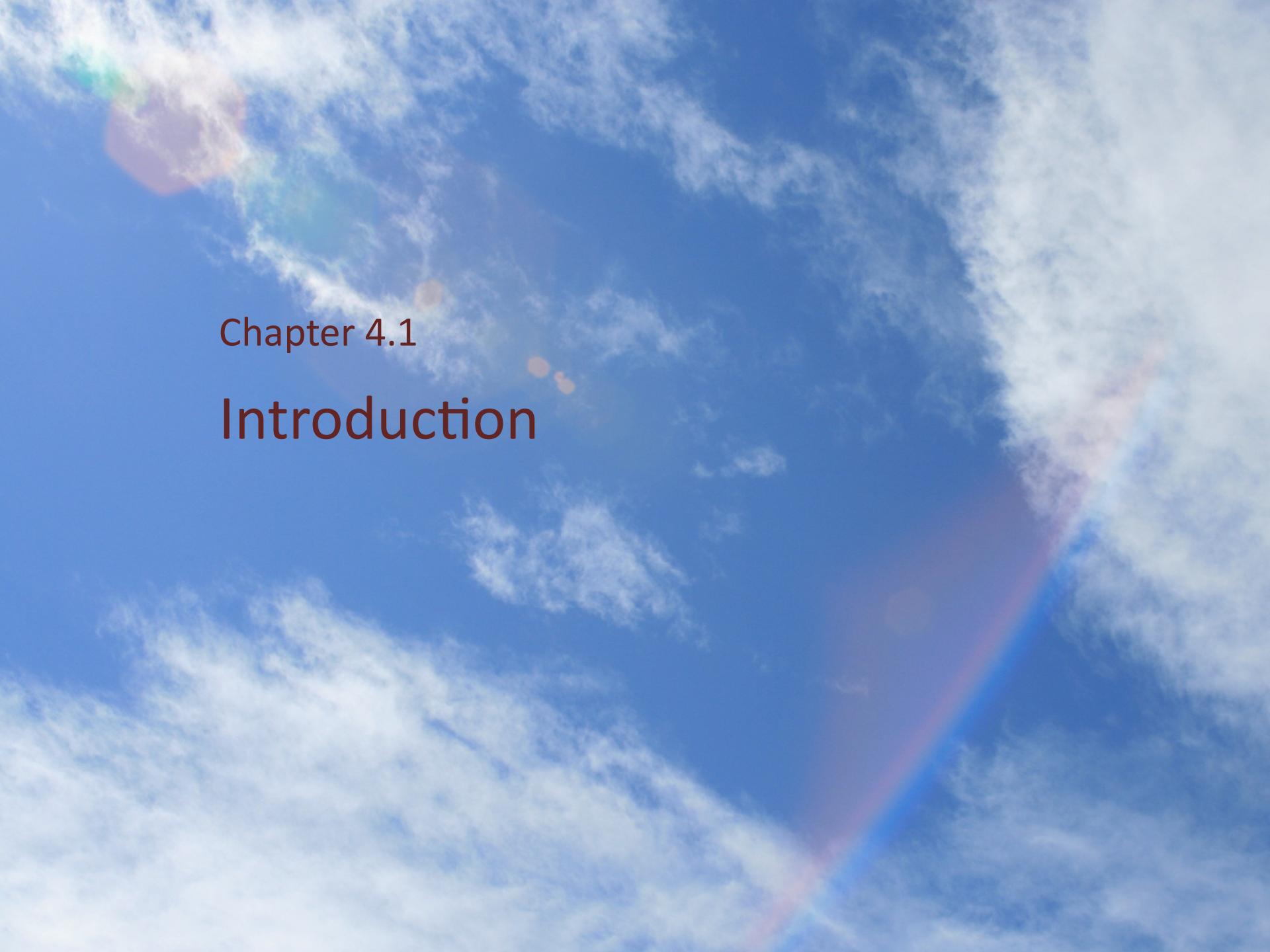
Recommended Reading

- David C. Fallside, Priscilla Walmsley (ed.):
XML Schema Part 0: Primer. Second Edition.
W3C Recommendation 28 October 2004.
<http://www.w3.org/TR/xmlschema-0/>
- Paul V. Biron, Ashok Malhotra (ed.):
XML Schema Part 2: Datatypes. Second Edition.
W3C Recommendation 28 October 2004.
<http://www.w3.org/TR/xmlschema-2/>

Overview

Informatik · CAU Kiel

1. Introduction
2. Getting started
3. Three XML Schema flavours
4. Simple types
5. List and Union datatypes
6. Annotations



Chapter 4.1

Introduction

Motivation for XML Schema

Informatik · CAU Kiel

- People are dissatisfied with DTDs
 - It's a different syntax
 - You write your XML (instance) document using one syntax and the DTD using another syntax → bad, inconsistent
 - Limited datatype capability
 - DTDs supports a single datatype: text
 - Desired: a set of datatypes compatible with those in databases
 - Create your own datatypes: "This is a new type based on the string type and elements of this type must follow this pattern: ddd - dddd, where 'd' represents a digit".

Introduction

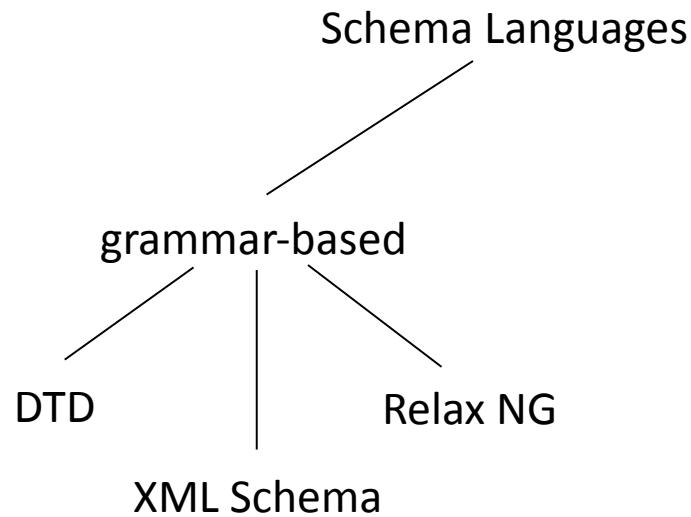
- What is XML Schema?



An XML vocabulary and grammar
for expressing your data's business rules.

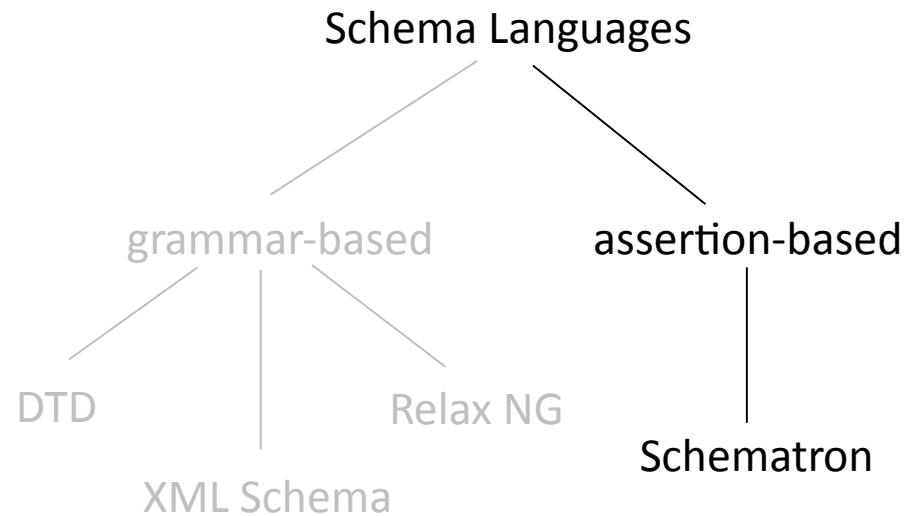
Introduction

- Taxonomy for schema languages
 - A **grammar-based** schema specifies
 - what elements may be used in an XML **instance document**,
 - the **order** of the elements,
 - the number of **occurrences** of each element, and
 - the **datatype** of each element and its **attributes**.



Introduction

- Taxonomy for schema languages
 - An **assertion-based** schema makes assertions about the relationships that must hold between the elements and attributes in an XML instance document.



In this chapter we discuss the grammar-based W3C XML Schema language.

Introduction

- Introductory example
 - sample instance document

```
<location>
    <latitude>32.904237</latitude>
    <longitude>73.620290</longitude>
    <uncertainty units="meters">2</uncertainty>
</location>
```

Introduction

- To be valid, an instance document must meet the following constraints:
 1. The location must be comprised of a latitude, followed by a longitude, followed by an indication of the uncertainty of the lat/lon measurements.
 2. The latitude must be a decimal with a value between -90 to +90.
 3. The longitude must be a decimal with a value between -180 to +180.
 4. For both latitude and longitude the number of digits to the right of the decimal point must be exactly six digits.
 5. The value of uncertainty must be a non-negative integer.
 6. The uncertainty units must be either meters or feet.

Introduction

```
<location>
    <latitude>32.904237</latitude>
    <longitude>73.620290</longitude>
    <uncertainty units="meters">2</uncertainty>
</location>
```

XML instance doc

Declare a location element. Require that its content be latitude, longitude, and uncertainty.
Declare a latitude element. Require that its value be between -90 and +90.
Declare a longitude element. Require that its value be between -180 and +180.
Declare a uncertainty element with a units attribute.
Require that the element's value be between 0 and 10.
Require that the attribute's value be either feet or meters.

XML Schema
validator

Ok!

Informal grammar
→ XML Schema

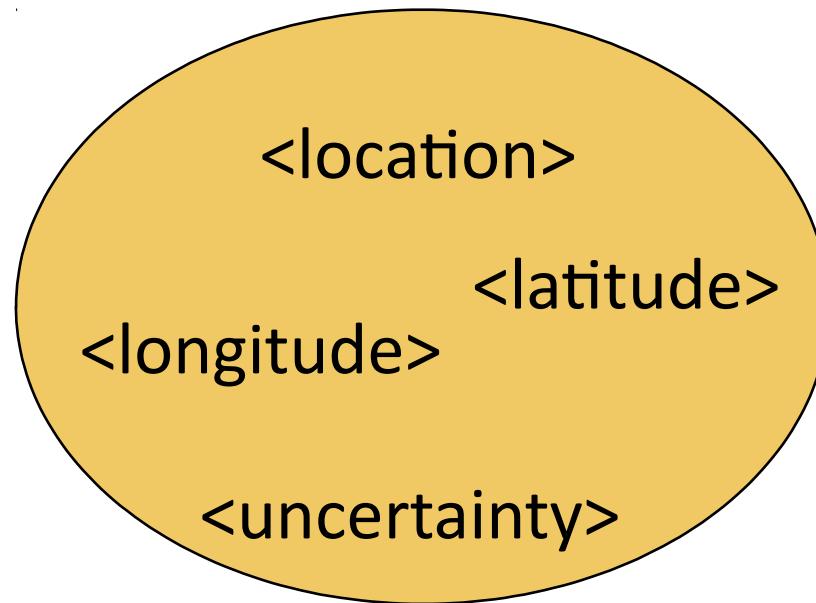
Introduction

- What does an XML Schema accomplish?
 - Answer: It creates an XML **vocabulary**:
 <location>, <latitude>, <longitude>, <uncertainty>
 - It specifies the **contents** of each element, and the **restrictions** on the content.
 - It specifies the **attributes** of each element.
 - An XML Schema specifies that the XML vocabulary that is being created shall be in a "**namespace**".

Introduction

- Namespace

`http://www.example.org/target`



Introduction

- Constraints: in a **document** or in **code**?
 - An XML Schema is an XML document.
The constraints on the data are expressed in a document.
 - All of the constraints could be expressed in a programming language as well in your system's middleware.

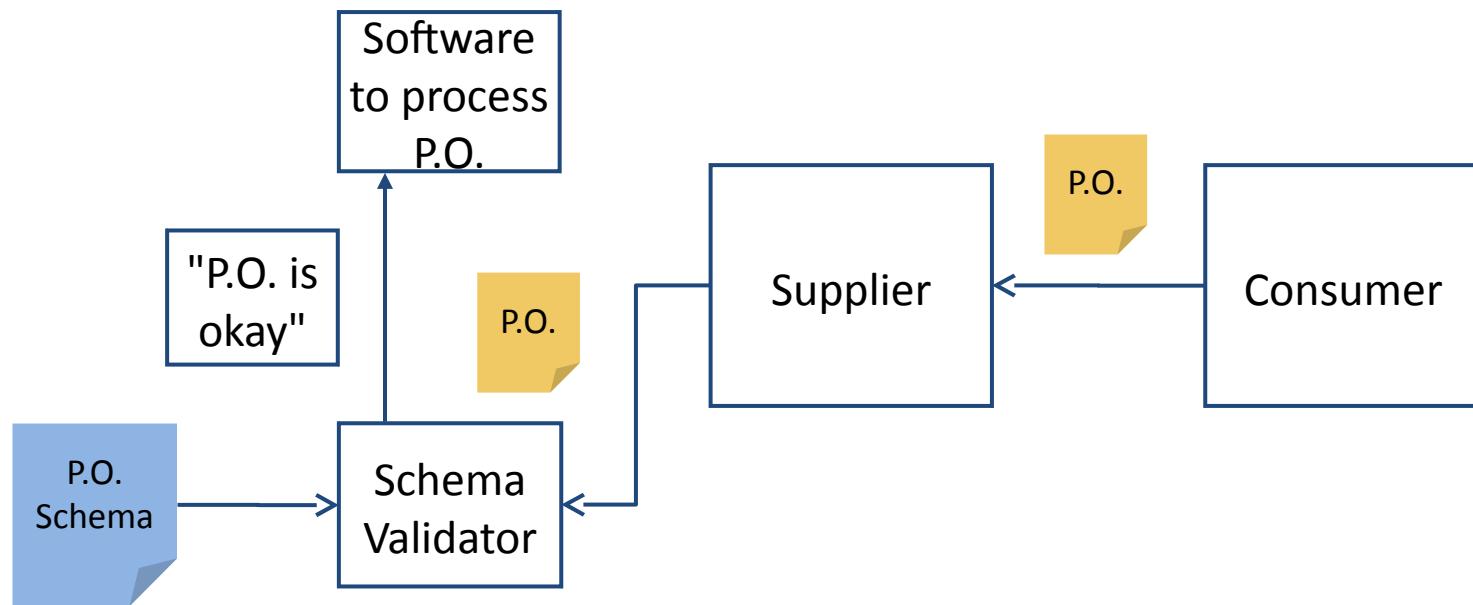


Why in a **document**?

- By expressing the data constraints in a document, the schema itself becomes information!
- The schema can be shipped around, mined, morphed, searched, etc.
- Don't bury your data constraints within code in middleware.
Information is king!

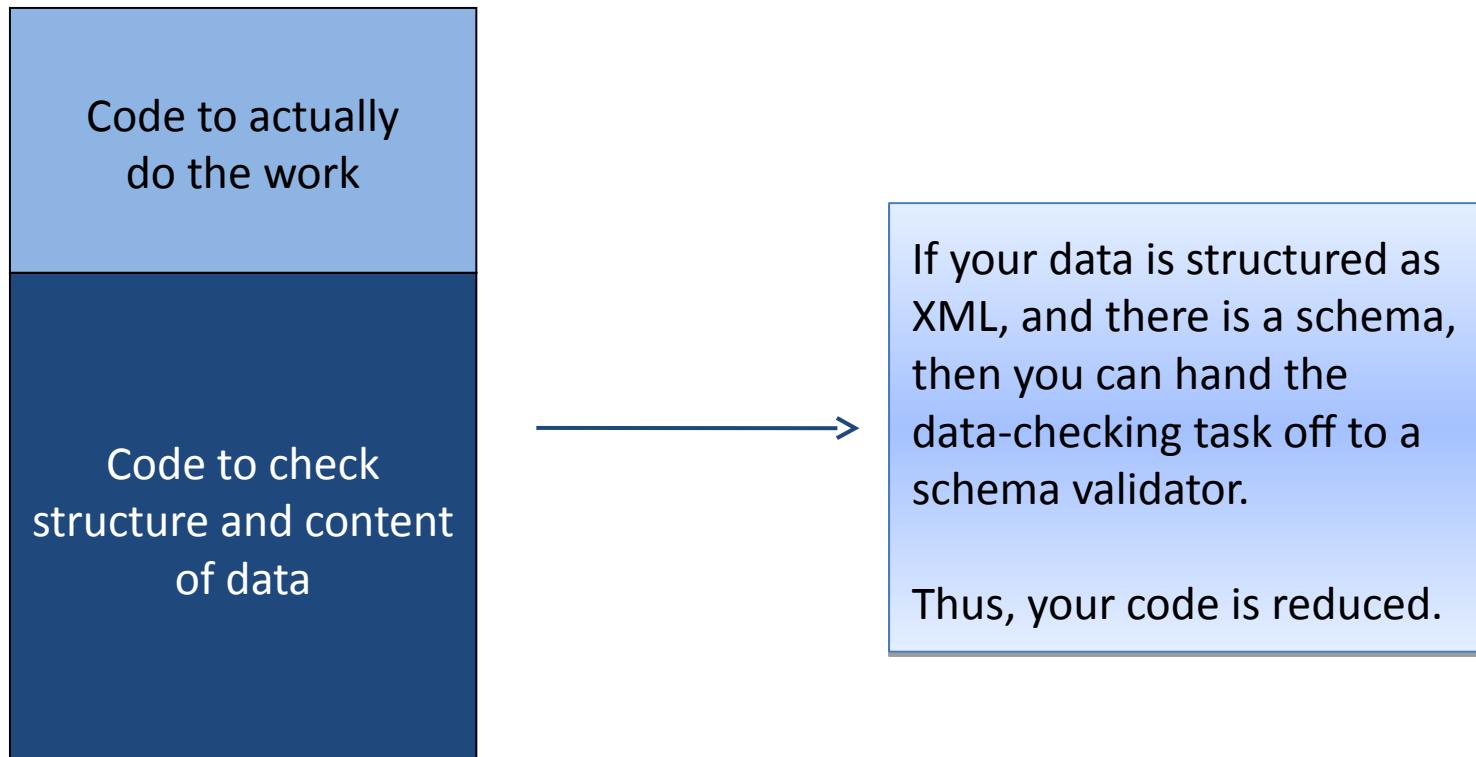
Introduction

- Use of XML Schemas
 - example: purchase order processing



Introduction

- Save \$\$\$ using XML Schemas



Introduction

- What are XML Schemas?
 - Data Model
 - With XML Schemas you specify how your XML data will be organized, and the datatypes of your data. That is, with XML Schemas you model how your data is to be represented in an instance document.
 - A Contract
 - Organizations agree to structure their XML documents in conformance with an XML Schema. Thus, the XML Schema acts as a contract between the organizations.

Introduction

- What are XML Schemas? (cont'd.)
 - A rich source of metadata
 - An XML Schema document contains lots of data about the data in the XML instance documents, such as the datatype of the data, the data's range of values, how the data is related to another piece of data (parent/child, sibling relationship), i.e., XML Schemas contain metadata

Highlights of XML Schema

- XML Schema is a tremendous advancement over DTDs
 - Can express **sets**, i.e., can define the child elements to occur in any order
 - Can specify element content as being **unique** (keys on content) and uniqueness within a region
 - Can define **multiple** elements with the same name but different content
 - **Object-oriented'ish**: can extend or restrict a type: derive new type definitions on the basis of old ones

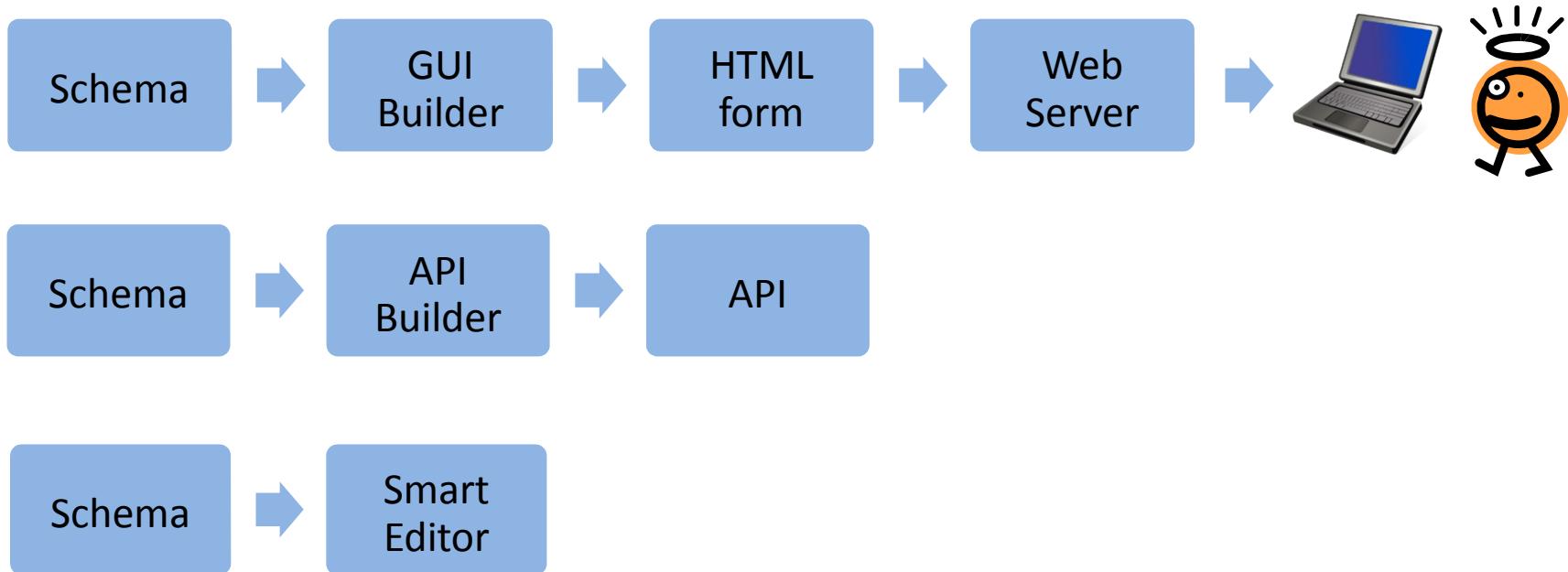
Highlights of XML Schema

Informatik · CAU Kiel

- XML Schema is a tremendous advancement over DTDs
 - Can define elements with **nil** content
 - Can define **substitutable** elements, e.g., the "Book" element is substitutable for the "Publication" element.

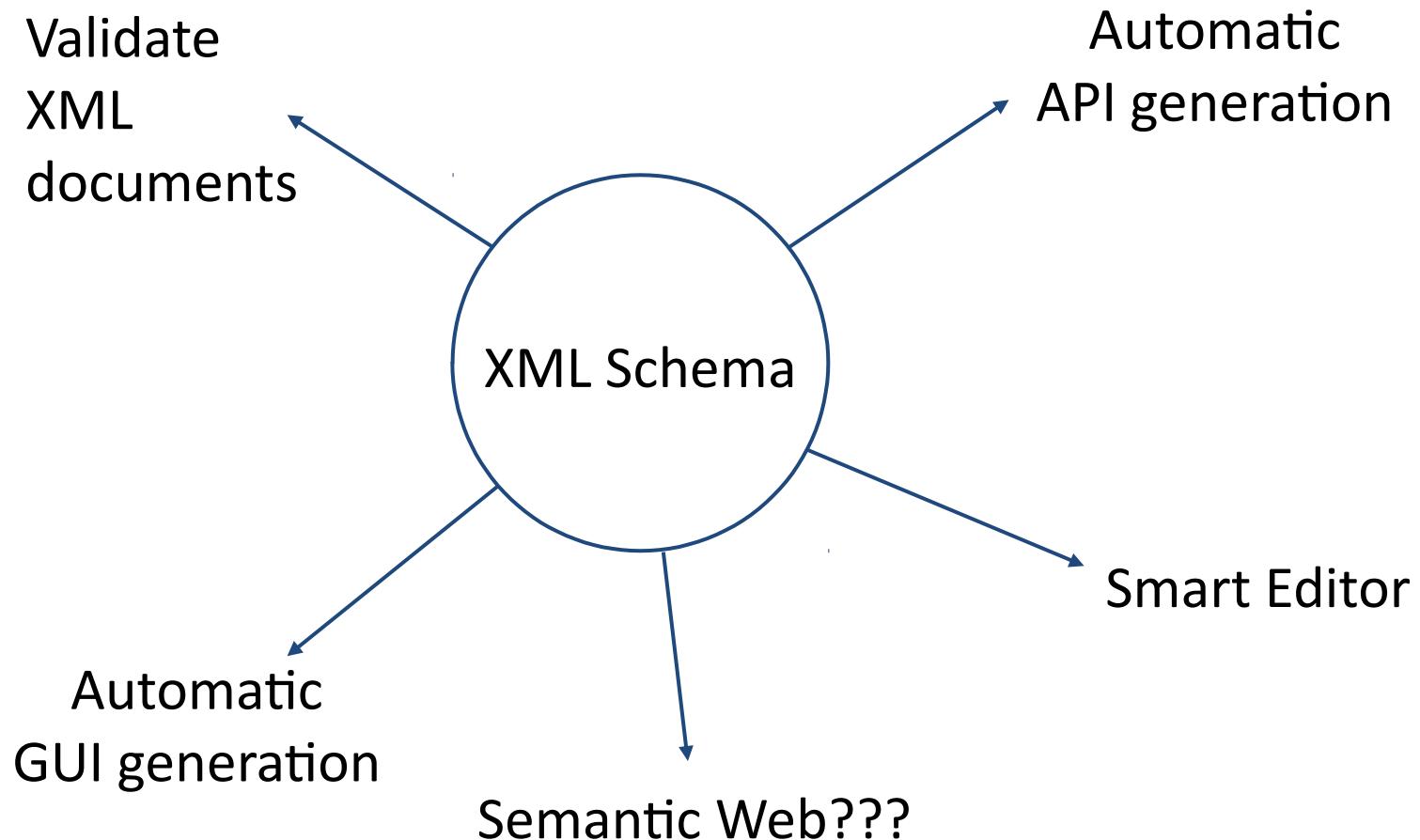
Highlights of XML Schema

- No Limits
 - There are many other uses for XML Schemas. Schemas are a wonderful source of metadata.



Highlights of XML Schema

Informatik · CAU Kiel





Chapter 4.2

Getting started

Let's Get Started!

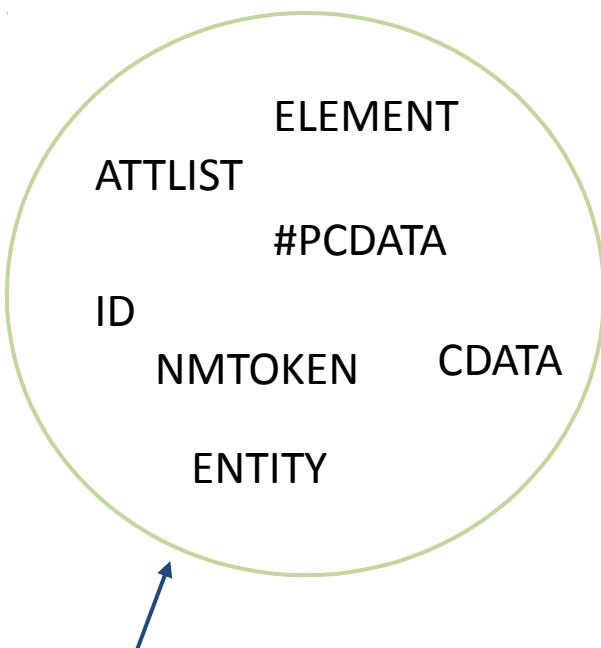
- Convert the BookStore.dtd to the XML Schema syntax

```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```

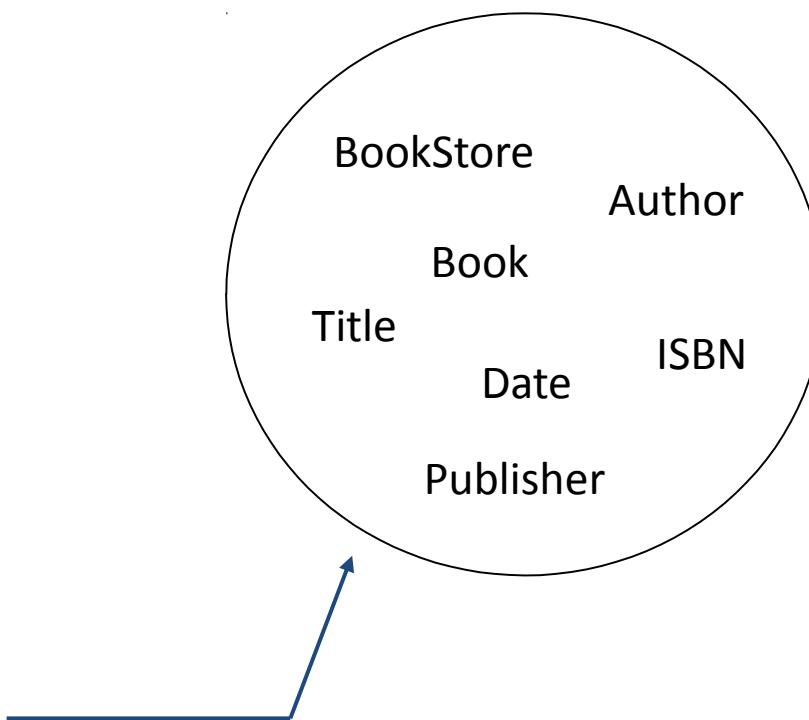
- Straight, one-to-one conversion, i.e.,
Title, Author, Date, ISBN, and Publisher will hold strings
- We will gradually modify the XML Schema to use stronger types

Bookstore Schema

- Namespaces again



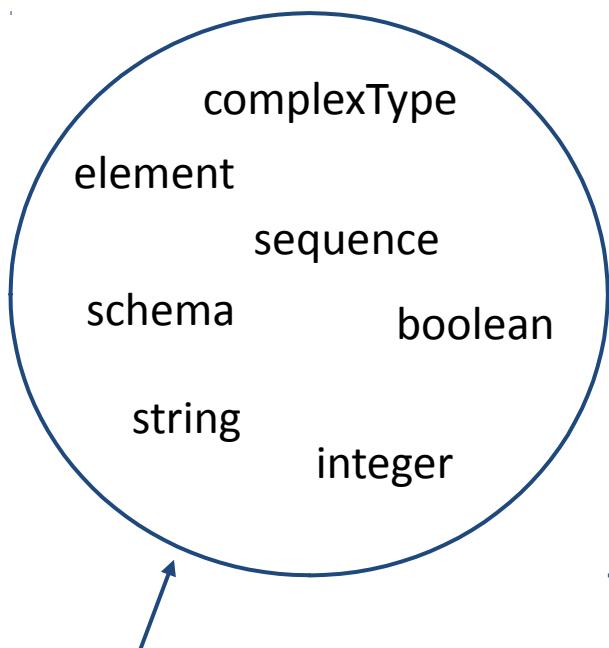
This is the vocabulary that DTDs provide to define your new vocabulary



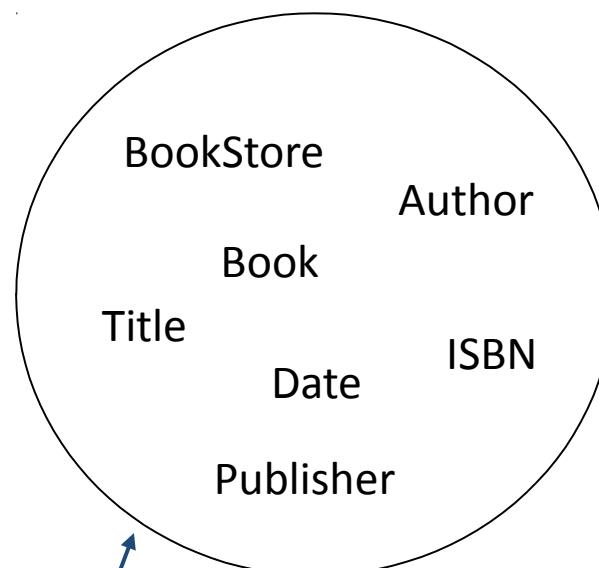
Bookstore Schema

Informatik · CAU Kiel

<http://www.w3.org/2001/XMLSchema>



[http://www.books.org \(*targetNamespace*\)](http://www.books.org)



This is the vocabulary that
XML Schemas provide to define your
new vocabulary

Bookstore Schema

- Notice:
 - XML Schema vocabulary is associated with a namespace.
 - Likewise, the new vocabulary that you define must be associated with a namespace.
 - With DTDs neither set of vocabulary is associated with a namespace [because DTDs pre-dated namespaces].

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.books.org"
    xmlns="http://www.books.org"
    elementFormDefault="qualified">
    BookStore.xsd
    <xsd:element name="BookStore">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Book">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string"/>
    <xsd:element name="Date" type="xsd:string"/>
    <xsd:element name="ISBN" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.books.org"
    xmlns="http://www.books.org"
    elementFormDefault="qualified">
    <xsd:element name="BookStore">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Book">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Title" type="xsd:string"/> <--> <!ELEMENT Title (#PCDATA)>
    <xsd:element name="Author" type="xsd:string"/> <--> <!ELEMENT Author (#PCDATA)>
    <xsd:element name="Date" type="xsd:string"/> <--> <!ELEMENT Date (#PCDATA)>
    <xsd:element name="ISBN" type="xsd:string"/> <--> <!ELEMENT ISBN (#PCDATA)>
    <xsd:element name="Publisher" type="xsd:string"/> <--> <!ELEMENT Publisher (#PCDATA)>
</xsd:schema>
```



```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org"
  elementFormDefault="qualified">
  <element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Title" type="xsd:string"/>
  <xsd:element name="Author" type="xsd:string"/>
  <xsd:element name="Date" type="xsd:string"/>
  <xsd:element name="ISBN" type="xsd:string"/>
  <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

All XML Schemas have "schema" as the document element.

```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.book
    xmlns="http://www.books.org"
    elementFormDefault="qualified">
    <xsd:element name="BookStore">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Book" minOccurs="1" maxOccu
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Book">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string"/>
    <xsd:element name="Date" type="xsd:string"/>
    <xsd:element name="ISBN" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>

```



The elements and datatypes that are used to construct a schema, e.g.

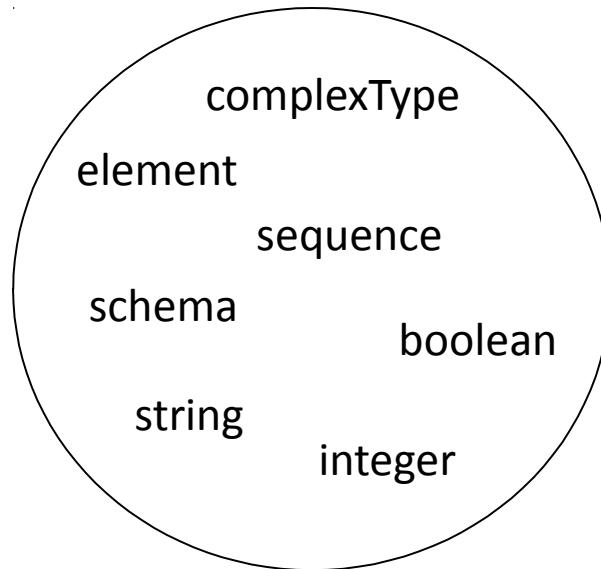
- schema
- element
- complexType
- sequence
- string

come from the <http://.../XMLSchema> namespace.

Bookstore Schema

- XMLSchema Namespace

<http://www.w3.org/2001/XMLSchema>



```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.books.org"
    xmlns="http://www.book
    elementFormDefault="qu d">
    <xsd:element name="BookStor
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Book" minOccurs="1" maxOcu
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Book">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string"/>
    <xsd:element name="Date" type="xsd:string"/>
    <xsd:element name="ISBN" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>

```



Indicates that the elements defined by this schema

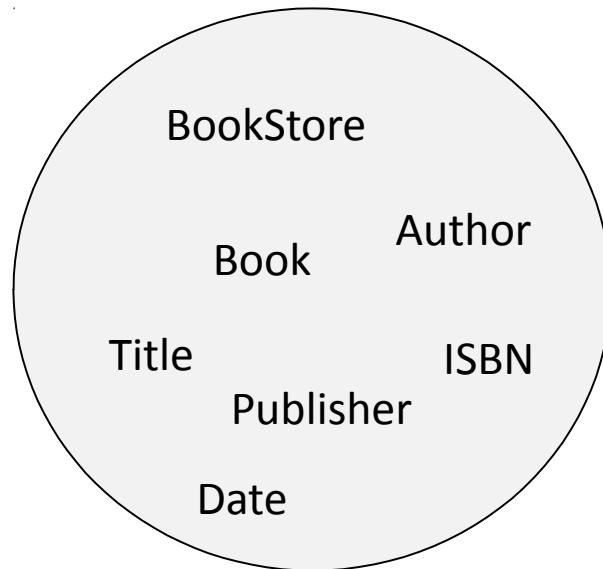
- BookStore
- Book
- Title
- Author
- Date
- ISBN
- Publisher

are to go in the
<http://www.books.org> namespace

Bookstore Schema

- Book Namespace (*targetNamespace*)

<http://www.books.org> (*targetNamespace*)



```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.books.org"
    xmlns="http://www.books.org"
    elementFormDefault="qualified">
    
    <xsd:element name="Bookstore">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Book" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Book">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string"/>
    <xsd:element name="Date" type="xsd:string"/>
    <xsd:element name="ISBN" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

The default namespace is
http://www.books.org
which is the
targetNamespace!

Reference to Book in what
namespace?

Since there is no namespace
qualifier it is referencing the Book
element in the default namespace,
which is the targetNamespace!
Thus, this is a reference to the Book
element declaration in this schema.

```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.books.org"
    xmlns="http://www.books.org"
    elementFormDefault="qualified">
    <xsd:element name="BookSto
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="| min0curs="1" max0curs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Book">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Title" min0curs="1" max0curs="1"/>
                <xsd:element ref="Author" min0curs="1" max0curs="1"/>
                <xsd:element ref="Date" min0curs="1" max0curs="1"/>
                <xsd:element ref="ISBN" min0curs="1" max0curs="1"/>
                <xsd:element ref="Publisher" min0curs="1" max0curs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string"/>
    <xsd:element name="Date" type="xsd:string"/>
    <xsd:element name="ISBN" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>

```



Directive to any instance documents which conform to this schema:
Any elements used by the instance document which were declared in this schema must be namespace qualified.

```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org"
  elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Title" type="xsd:string"/>
  <xsd:element name="Author" type="xsd:string"/>
  <xsd:element name="Date" type="xsd:string"/>
  <xsd:element name="ISBN" type="xsd:string"/>
  <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>

```

Occurrence constraints

The default value for minOccurs is "1"

The default value for maxOccurs is "1"

e.g.

<xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>

equivalent to

<xsd:element ref="Title"/>



Bookstore Schema

- Referencing a schema in an XML instance document

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.books.org/BookStore.xsd">
    <Book>
        <Title>The Catcher in the Rye</Title>
        <Author>J.D. Salinger</Author>
        <Date>1951</Date>
        <ISBN>978-0-316-18663-0</ISBN>
        <Publisher>McMillin Publishing</Publisher>
    </Book>
    ...
</BookStore>
```

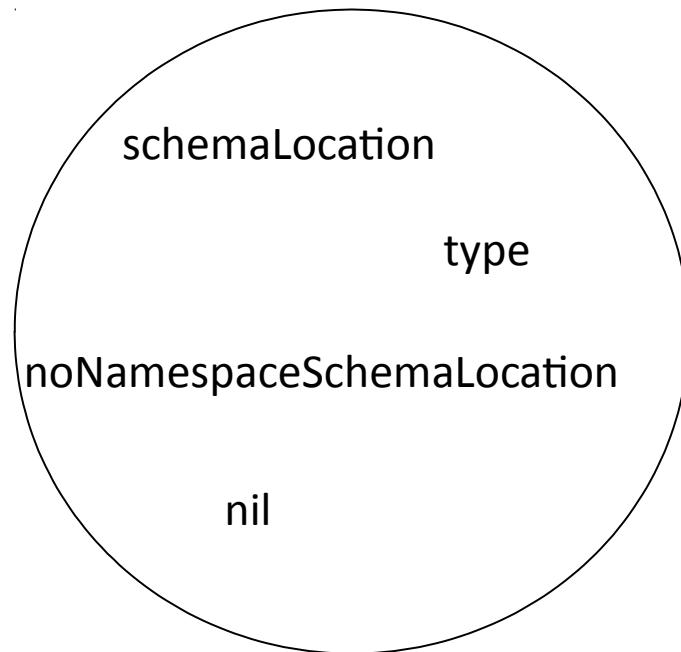
Tell the schema-validator that all of the elements used in this instance document come from the <http://www.books.org> namespace.

Tell the schema-validator that the schemaLocation attribute is in the XMLSchema-instance namespace.

Bookstore Schema

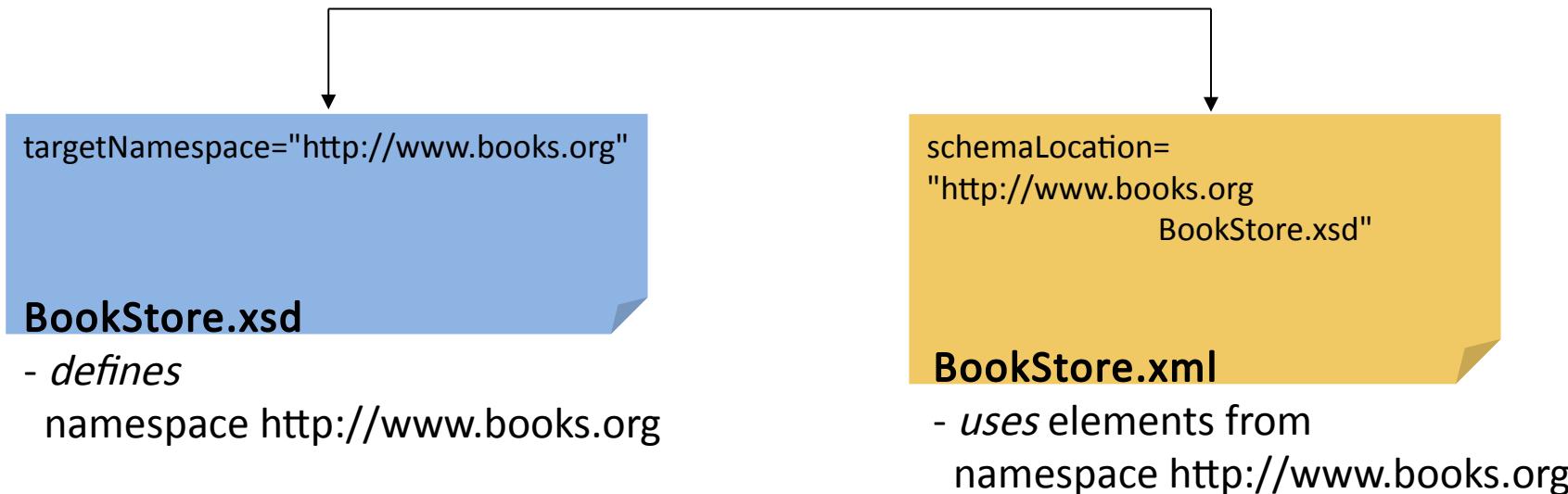
- XMLSchema-instance Namespace

<http://www.w3.org/2001/XMLSchema-instance>



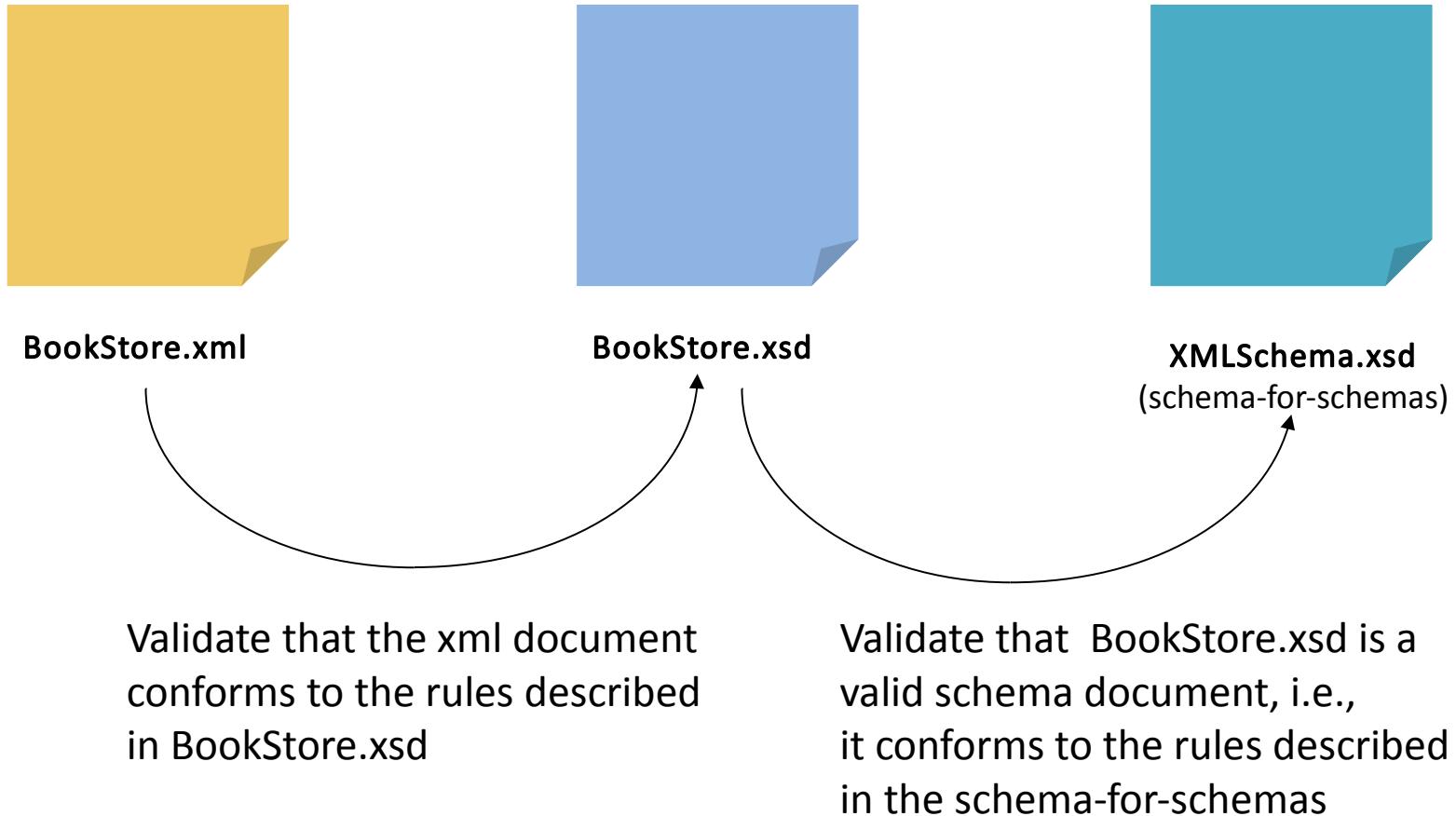
Bookstore Schema

- XML schema and instance document



- A schema defines a new vocabulary.
- Instance documents use that new vocabulary.

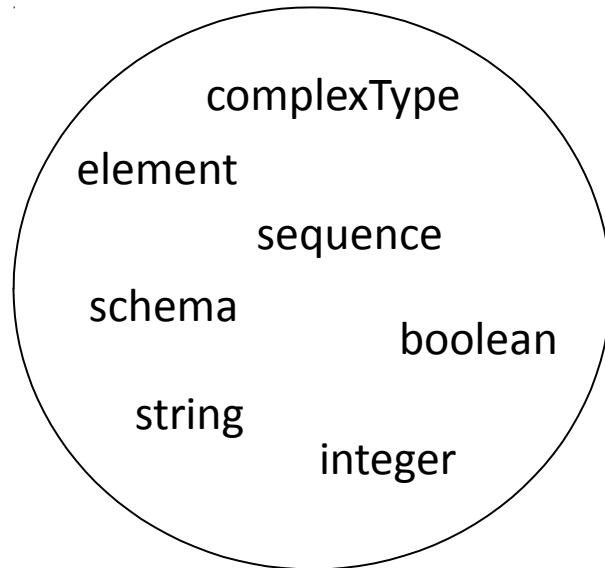
Multiple Levels of Checking



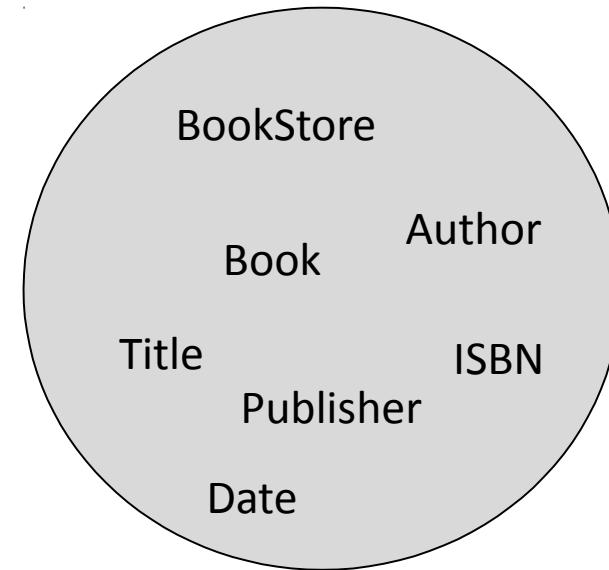
Namespaces Again

- targetNamespace may be the default namespace.

<http://www.w3.org/2001/XMLSchema>



[http://www.books.org \(*targetNamespace*\)](http://www.books.org)



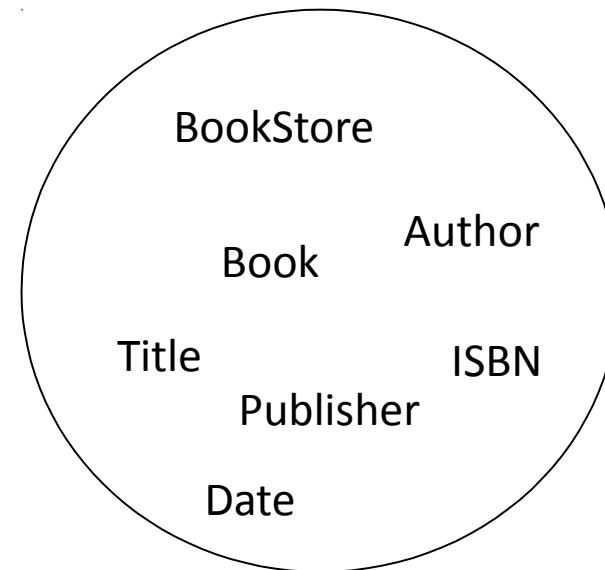
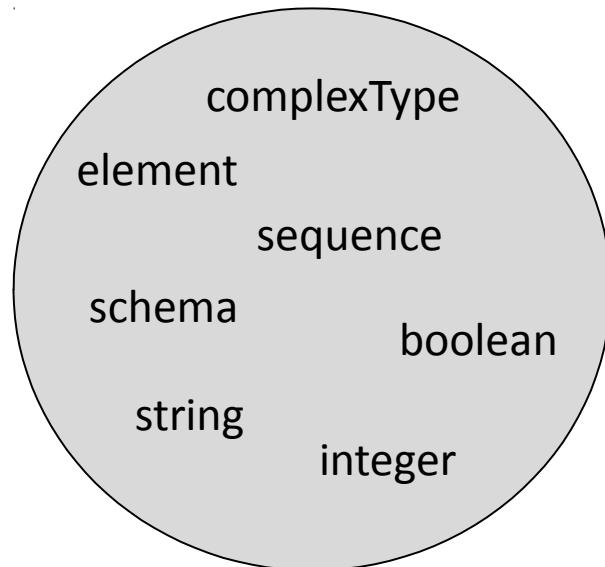
Namespaces Again

Informatik · CAU Kiel

- Alternatively (equivalently), we can design our schema so that XMLSchema is the default namespace.

<http://www.w3.org/2001/XMLSchema>

[http://www.books.org \(*targetNamespace*\)](http://www.books.org)



```

<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
         targetNamespace="http://www.w3.org"
         xmlns:bk="http://www.bookstore.org"
         elementFormDefault="qualified">
    <element name="BookStore">
        <complexType>
            <sequence>
                <element ref="bk:Book" minOccurs="1" maxOccurs="unbounded">
                    <complexType>
                        <sequence>
                            <element ref="bk:Title"/>
                            <element ref="bk:Author"/>
                            <element ref="bk:Date"/>
                            <element ref="bk:ISBN"/>
                            <element ref="bk:Publisher"/>
                        </sequence>
                    </complexType>
                </element>
            </sequence>
        </complexType>
    </element>
    <element name="Book">
        <complexType>
            <sequence>
                <element ref="bk:Title"/>
                <element ref="bk:Author"/>
                <element ref="bk:Date"/>
                <element ref="bk:ISBN"/>
                <element ref="bk:Publisher"/>
            </sequence>
        </complexType>
    </element>
    <element name="Title" type="string"/>
    <element name="Author" type="string"/>
    <element name="Date" type="string"/>
    <element name="ISBN" type="string"/>
    <element name="Publisher" type="string"/>
</schema>

```



Note that http://.../XMLSchema is the default namespace. Consequently, there are no prefixes on

- schema
- element
- complexType
- sequence
- string

```

<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
         targetNamespace="http://www.books.org"
         xmlns:bk="http://www.books.org"
         elementFormDefault="qualified">
    <element name="BookStore">
        <complexType>
            <sequence>
                <element ref="bk:Book" minOccurs="1" maxOccurs="unbounded"/>
            </sequence>
        </complexType>
    </element>
    <element name="Book">
        <complexType>
            <sequence>
                <element ref="bk:Title"/>
                <element ref="bk:Author"/>
                <element ref="bk:Date"/>
                <element ref="bk:ISBN"/>
                <element ref="bk:Publisher"/>
            </sequence>
        </complexType>
    </element>
    <element name="Title" type="string"/>
    <element name="Author" type="string"/>
    <element name="Date" type="string"/>
    <element name="ISBN" type="string"/>
    <element name="Publisher" type="string"/>
</schema>

```



=

Here we are referencing a Book element.
In what namespace is that Book element defined??

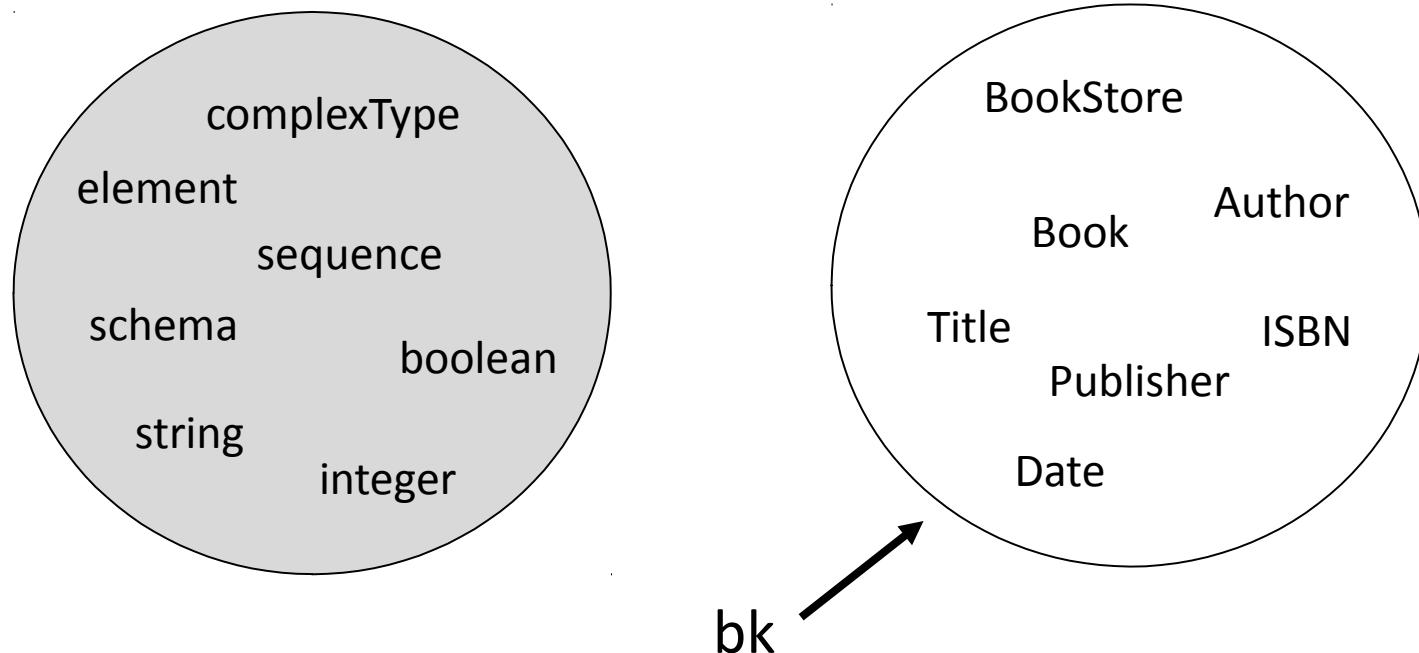
The bk: prefix indicates what namespace this element is in. bk: has been set to be the same as the targetNamespace.

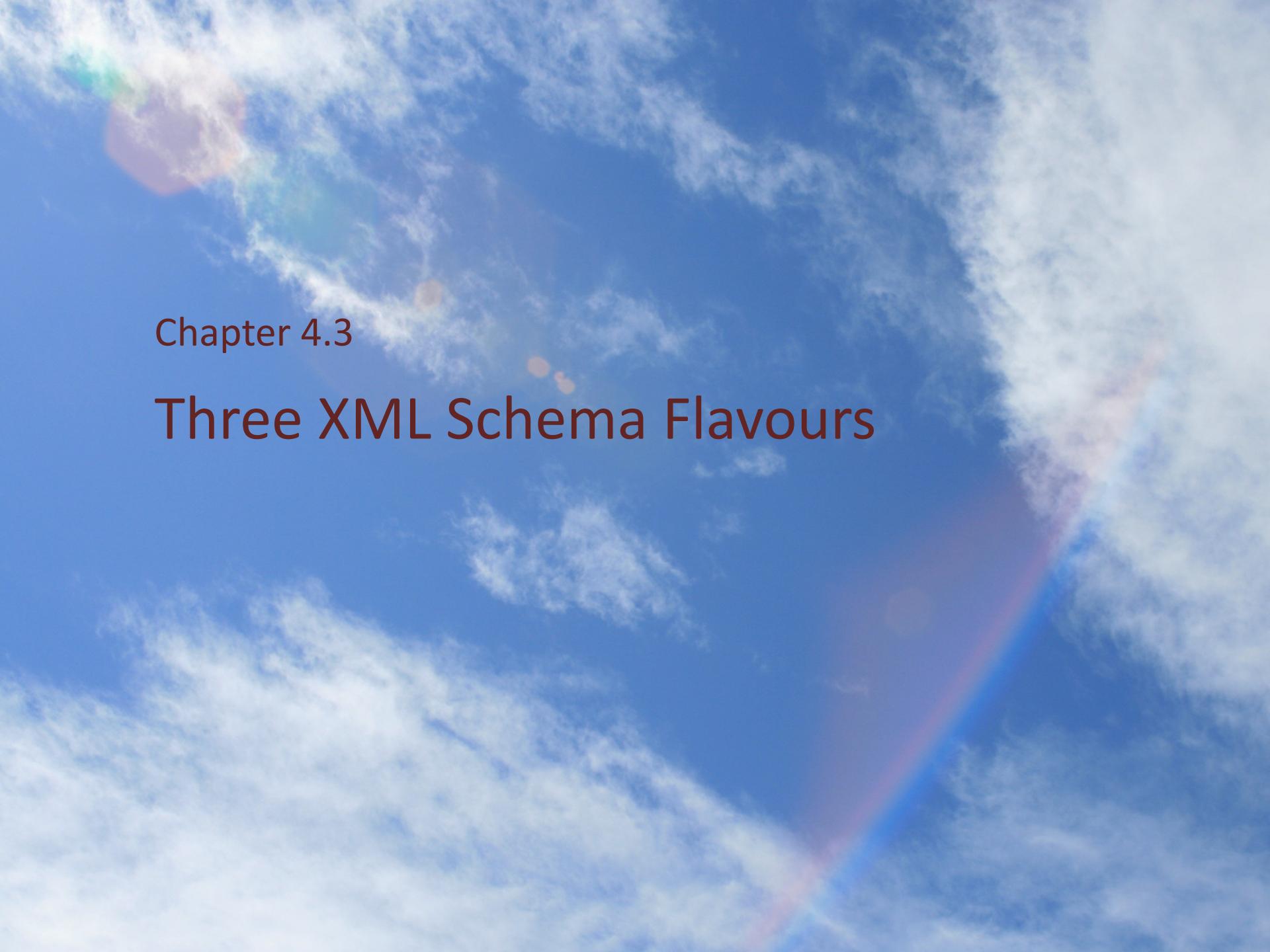
Namespaces Again

Informatik · CAU Kiel

- "bk:" references the targetNamespace
→ bk:Book refers to the Book element in the targetNamespace.

<http://www.w3.org/2001/XMLSchema> [http://www.books.org \(*targetNamespace*\)](http://www.books.org)





Chapter 4.3

Three XML Schema Flavours

Three XML Schema Flavours

- Three different "flavours" for writing an XML schema
 - **embedded types:**
types are defined where they are used in the document hierarchy
 - **named types:**
each element has a name and a named type,
and each named type is defined separately
 - **flat catalogue:**
each element is defined by reference to another element declaration
- Schemas may not be equivalent!

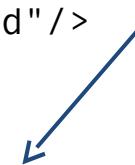
Embedded types

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              targetNamespace="http://www.books.org"
              xmlns="http://www.books.org"
              elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Title" type="xsd:string"/>
              <xsd:element name="Author" type="xsd:string"/>
              <xsd:element name="Date" type="xsd:string"/>
              <xsd:element name="ISBN" type="xsd:string"/>
              <xsd:element name="Publisher" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Types are anonymous.

Named types

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              targetNamespace="http://www.books.org"
              xmlns="http://www.books.org"
              elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" type="BookPublication"
                     maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="BookPublication">
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```



The advantage of splitting out Book's element declarations and wrapping them in a named type is that now this type can be *reused* by other elements.

Three XML Schema Flavours

- Please note that

This

```
<xsd:element name="A" type="foo"/>
<xsd:complexType name="foo">
  <xsd:sequence>
    <xsd:element name="B" .../>
    <xsd:element name="C" .../>
  </xsd:sequence>
</xsd:complexType>
```

is equivalent to:

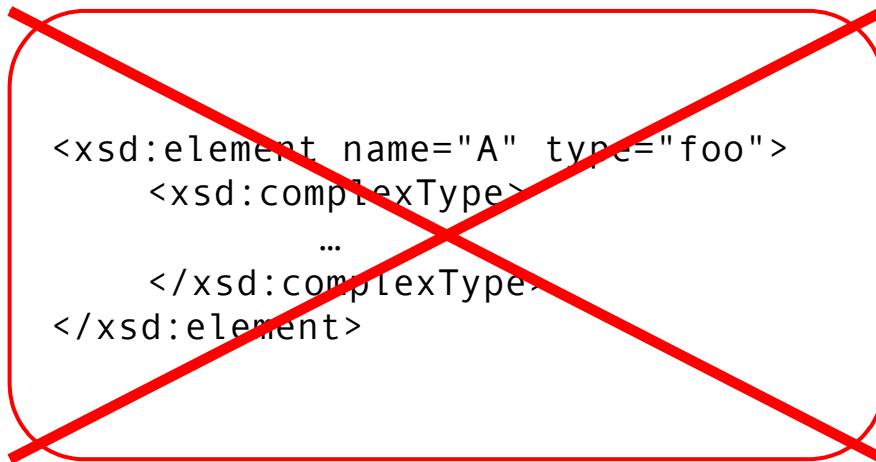
```
<xsd:element name="A">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="B" .../>
      <xsd:element name="C" .../>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Element A *references* the complexType foo.

Element A has the complexType definition *inlined* in the element declaration.

Three XML Schema Flavours

- An element definition has either
 - a `type` attribute or
 - a `complexType` child element, but not both!



Flat catalogue

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.books.org"
    xmlns="http://www.books.org"
    elementFormDefault="qualified">
    <xsd:element name="BookStore">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Book">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string"/>
    <xsd:element name="Date" type="xsd:string"/>
    <xsd:element name="ISBN" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

Embedded types?

```
<schema>
  <element name="weather">
    <complexType>
      <sequence>
        <element name="location">
          <complexType>
            <sequence>
              <element name="city"> <simpleType>
                <restriction base="string">
                  <pattern value="[a-zA-Z]" />
                </restriction>
              </simpleType> </element>
              <element name="country" type="string"/>
            </sequence>
          </complexType>
        </element>
        <element name="temperature" type="integer"/>
        <element name="barometric_pressure" type="integer"/>
        <element name="conditions" type="string"/>
      </sequence>
    </complexType>
  </element>
</schema>
```

Embedded types?

```
<schema>
  <complexType name="weatherType">
    <sequence>
      <element name="location" type="locationType"/>
      <element name="temperature" type="integer"/>
      <element name="barometric_pressure" type="integer"/>
      <element name="conditions" type="string"/>
    </sequence>
  </complexType>
  <complexType name="locationType" >
    <sequence>
      <element name="city" type="cityType"/>
      <element name="country" type="string"/>
    </sequence>
  </complexType>
  <simpleType name="cityType">
    <restriction base="string">
      <pattern value="[a-zA-Z]"/>
    </restriction>
  </simpleType>
  <element name="weather" type="weatherType"/>
</schema>
```

Embedded types?

```
<schema>
  <element name="temperature" type="integer"/>
  <element name="barometric_pressure" type="integer"/>
  <element name="conditions" type="string"/>
  <element name="country" type="string"/>
  <element name="city">
    <simpleType>
      <restriction base="string">
        <pattern value="[a-zA-Z]"/>
      </restriction>
    </simpleType>
  </element>
  <element name="location">
    <complexType> <sequence>
      <element ref="city"/>
      <element ref="country"/> </sequence>
    </complexType>
  </element>
  <element name="weather">
    <complexType> <sequence>
      <element ref="location"/>
      <element ref="temperature"/>
      <element ref="barometric_pressure"/>
      <element ref="conditions"/>
    </sequence>
  </complexType>
  </element>
</schema>
```

Summary of Declaring Elements

1

```
<xsd:element name="name" type="type" minOccurs="int" maxOccurs="int" />
```



A simple type (e.g., `xsd:string`)
or the name of a `complexType`
(e.g., `BookPublication`)

A non-negative
integer

A non-negative
integer or "unbounded"

Note: *minOccurs* and *maxOccurs* can only be
used in nested (local) element declarations.

2

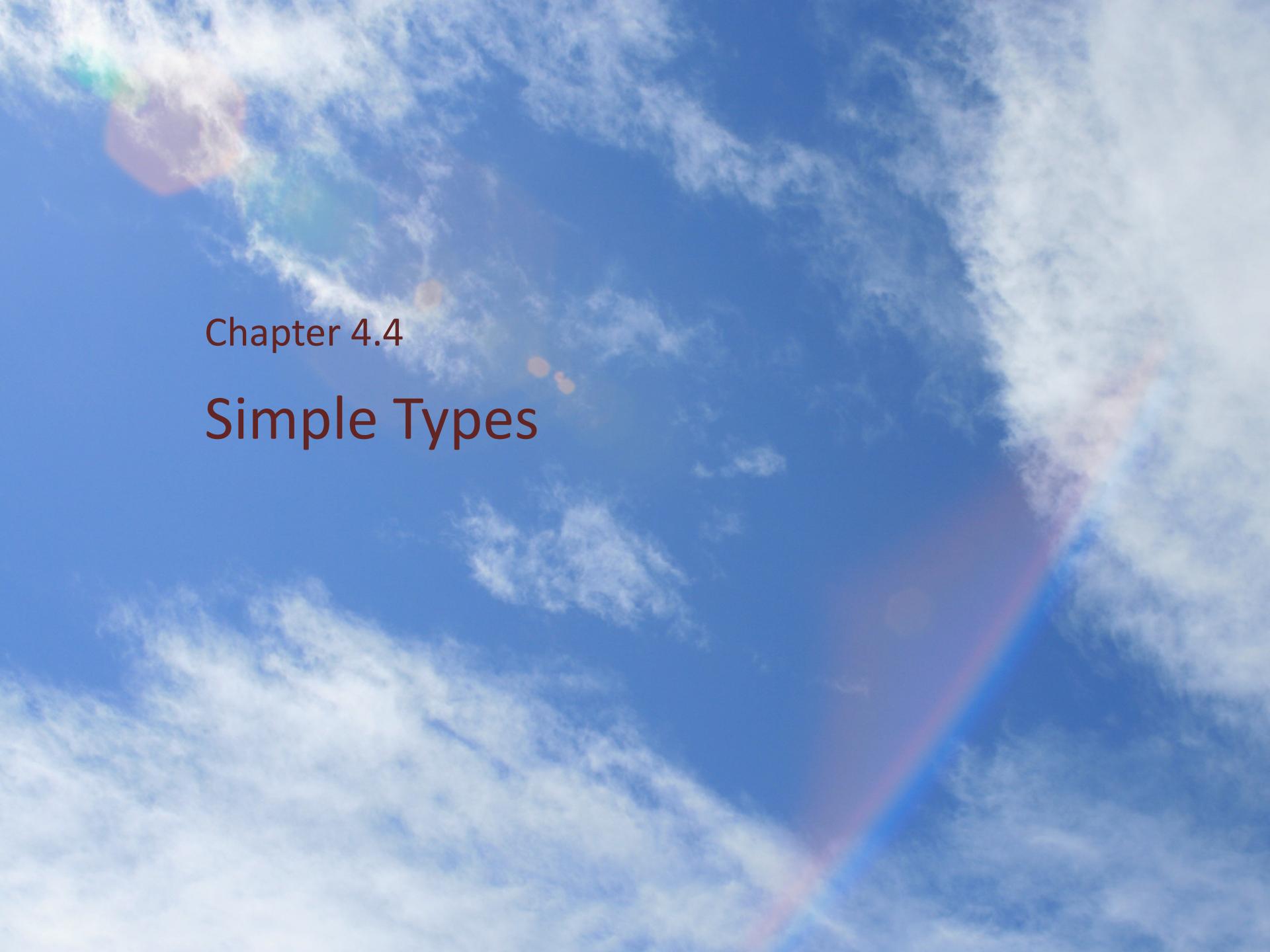
```
<xsd:element name="name" minOccurs="int" maxOccurs="int">  
    <xsd:complexType>
```

...

```
    </xsd:complexType>  
</xsd:element>
```

3

flat catalogue



The background of the slide features a clear blue sky with scattered, wispy white clouds. A full, vibrant rainbow arches across the center of the frame, its colors transitioning from red on the left to violet on the right. The overall effect is bright and airy.

Chapter 4.4

Simple Types

Problem

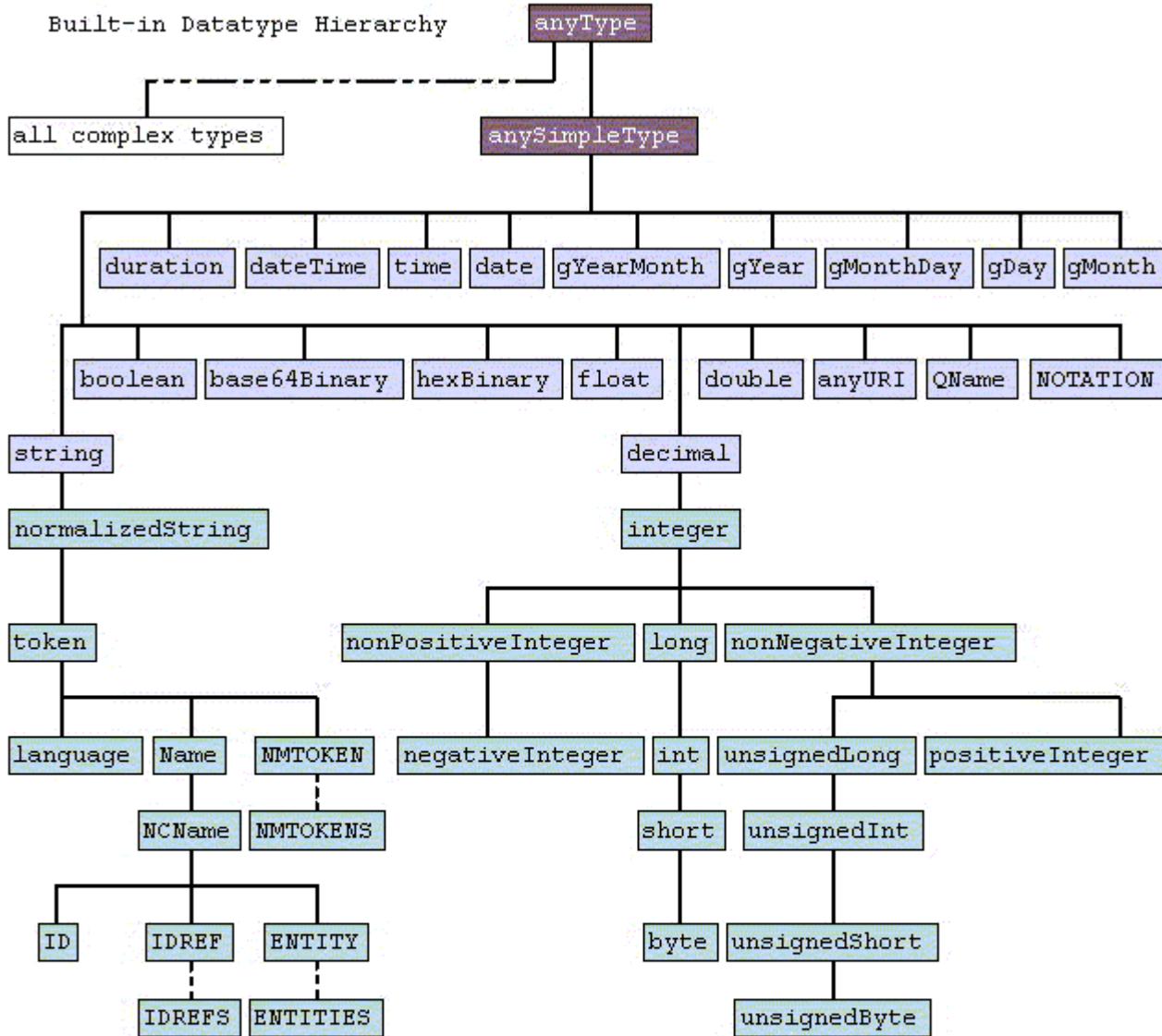
- Declaring the `Date` element of `BookStore.xsd`
 - type `string` is unsatisfactory
 - it allows any string value to be input as the content of the `Date` element, including non-date strings
 - constrain the allowable content that the `Date` element can have, e.g. restrict the content of `Date` to just year values.
- Similar for the `ISBN` element

W3C XML Schema

Hierarchy of built-in types

from:

*XML Schema Part 2:
Datatypes Second Ed.
W3C Recommendation
28 October 2004*



- ur types
- built-in primitive types
- built-in derived types
- complex types
- derived by restriction
- - - - derived by list
- - - - derived by extension or restriction

Datatypes: Definitions

- "A datatype is a 3-tuple, consisting
 - a set of distinct values, called its **value space**,
 - a set of lexical representations, called its **lexical space**, and
 - a set of **facets** that characterize properties of the value space, individual values or lexical items."

Datatypes: Definitions

- Value space
 - A value space is the set of values for a given datatype. Each value in the value space of a datatype is denoted by one or more literals in its lexical space.
 - The value space of a given datatype can be defined:
 - axiomatically from fundamental notions (intensional definition)
 - by enumeration (extensional definition)
 - by restricting the value space of an already defined datatype to a particular subset with a given set of properties
 - by combining the values from one or more already defined value space(s) by a specific construction procedure [list and union]

Datatypes: Definitions

- Lexical space
 - A lexical space is the set of valid literals for a datatype.

For example, "100" and "1.0E2" are two different literals from the lexical space of float which both denote the same value. The type system defined in this specification provides a mechanism for schema designers to control the set of values and the corresponding set of acceptable literals of those values for a datatype.

Datatypes: Definitions

- Primitive datatypes
 - Primitive datatypes are those that are not defined in terms of other datatypes; they exist *ab initio*.
- Derived datatypes
 - Derived datatypes are those that are defined in terms of other datatypes.

For example, `float` is a well-defined mathematical concept that cannot be defined in terms of other datatypes, while a `integer` is a special case of the more general datatype `decimal`.

Datatypes: Definitions

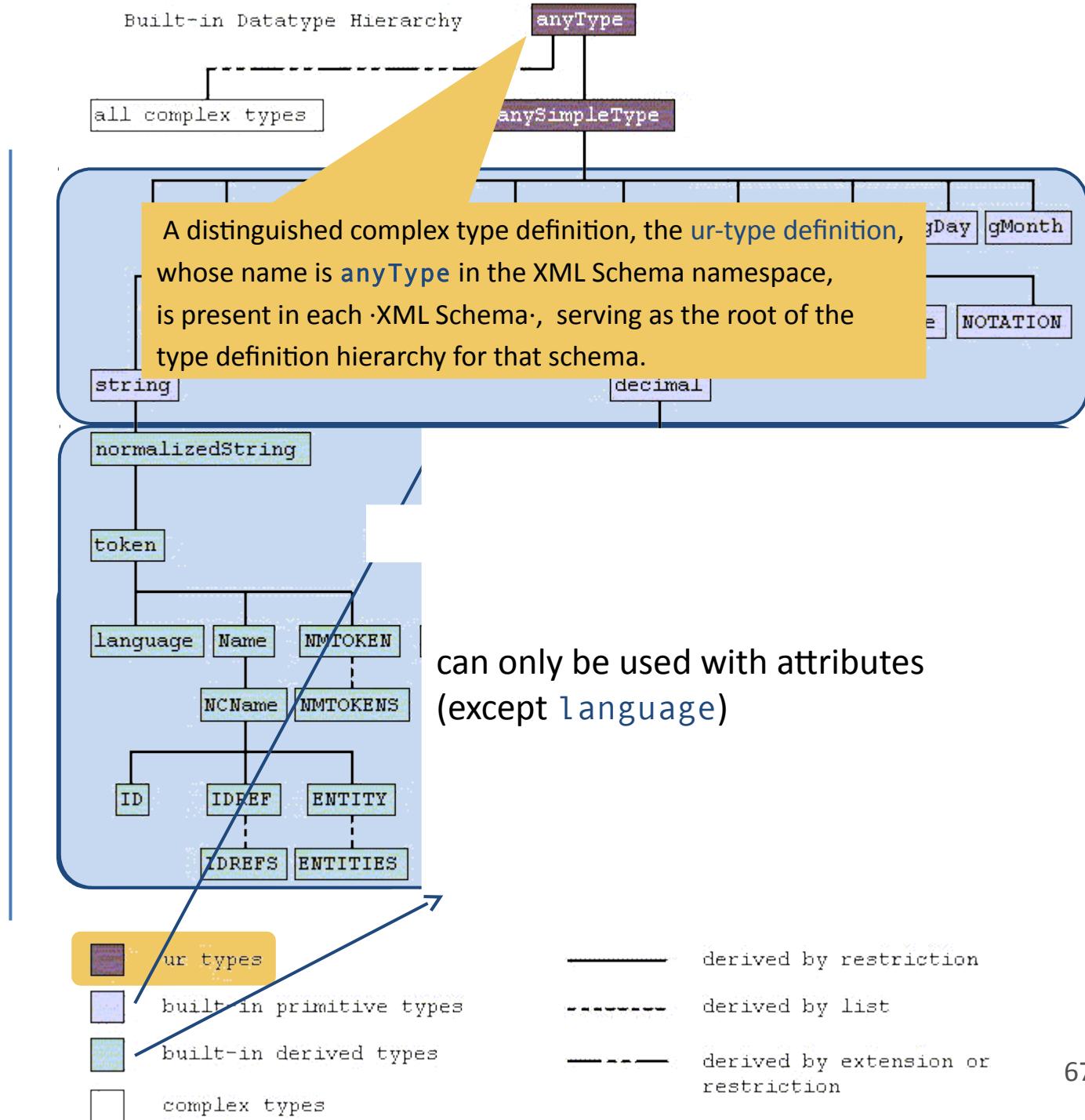
- Facets
 - A facet is a single defining aspect of a value space. Generally speaking, each facet characterizes a value space along independent axes or dimensions.
 - Facets are of two types:
 - fundamental facets
that define the datatype and
 - non-fundamental or constraining facets
that constrain the permitted values of a datatype.

W3C XML Schema

Hierarchy of built-in types

from:

*XML Schema Part 2:
Datatypes Second Ed.
W3C Recommendation
28 October 2004*



Built-in Datatypes

- Primitive datatypes: atomic, built-in

– string	→ "Hello World"
– boolean	→ {true, false, 1, 0}
– decimal	→ 7.08
– float	→ 12.56E3, 12, 12560, 0, -0, INF, -INF, NAN
– double	→ 12.56E3, 12, 12560, 0, -0, INF, -INF, NAN
– duration	→ P1Y2M3DT10H30M12.3S
– dateTime	format: CCYY-MM-DDThh:mm:ss
– time	format: hh:mm:ss.sss
– date	format: CCYY-MM-DD
– gYearMonth	format: CCYY-MM
– gYear	format: CCYY
– gMonthDay	format: --MM-DD

Note: 'T' is the date/time separator
INF = infinity
NAN = Not-A-Number

Built-in Datatypes (cont'd.)

- Primitive datatypes: atomic, built-in
 - gDay → format: ---*DD* (note the 3 dashes)
 - gMonth → format: --*MM*
 - hexBinary → a hex string
 - base64Binary → a base64 string
 - anyURI → <http://www.xfront.com>
 - QName → a namespace qualified name
 - NOTATION → a NOTATION from the XML spec

Built-in Datatypes (cont'd.)

- Derived from built-in types
 - normalizedString → A string without tabs, line feeds, or carriage returns
 - token → String w/o tabs, l/f, leading/trailing/consecutive spaces
 - language → any valid xml:lang value, e.g., EN, FR, ...
 - ID → must be used only with attributes
 - IDREF → must be used only with attributes
 - IDREFS → must be used only with attributes
 - ENTITY → must be used only with attributes
 - ENTITIES → must be used only with attributes
 - NMTOKEN → must be used only with attributes
 - NMTOKENS → must be used only with attributes
 - Name →
 - NCName → **part** (no namespace qualifier)
 - integer → **456**
 - nonPositiveInteger → negative infinity to 0

Built-in Datatypes (cont'd.)

- Derived from built-in types

– negativeInteger	→	negative infinity to -1
– long	→	-9223372036854775808 to 9223372036854775807
– int	→	-2147483648 to 2147483647
– short	→	-32768 to 32767
– byte	→	-127 to 128
– nonNegativeInteger	→	0 to infinity
– unsignedLong	→	0 to 18446744073709551615
– unsignedInt	→	0 to 4294967295
– unsignedShort	→	0 to 65535
– unsignedByte	→	0 to 255
– positiveInteger	→	1 to infinity

The **date** Datatype

- The date datatype
 - a *built-in* datatype
 - used to represent a specific day in notation: CCYY-MM-DD
 - range for CC is: 00-99
 - range for YY is: 00-99
 - range for MM is: 01-12
 - range for DD is:
 - 01-28 if month is 2
 - 01-29 if month is 2 and the gYear is a leap gYear
 - 01-30 if month is 4, 6, 9, or 11
 - 01-31 if month is 1, 3, 5, 7, 8, 10, or 12



Pope Gregory XIII, 1552 – 1614

The **gYear** Datatype

- The gYear datatype
 - a *built-in* datatype
 - Elements declared to be of type gYear must follow this form: CCYY
 - range for CC is: 00-99
 - range for YY is: 00-99

Datatypes for Attributes

- Special datatypes for attributes only
 - ID
 - unique value within the entire document
 - at most one ID attribute per element
 - no default value
 - IDREF
 - its value must be some other element's ID value in the document
 - IDREFS
 - its value is a space-separated set, each element of the set is the ID value of some other element in the document

Datatypes for Attributes

- Special datatypes for attributes only (cont'd.)
 - Example

```
<person id="o555">
    <name> Jane </name>
</person>
<person id="o456">
    <name> Mary </name>
    <children ref="o123 o555"/>
</person>
<person id="o123" mother="o456">
    <name>John</name>
</person>
```

User-defined simpleTypes

- Creating your own simpleTypes
 - A new datatype can be *derived* from an existing datatype, called the "base" type
 - Specify values for one or more of the *facets* for the base type.
 - Example: The string primitive datatype has six optional facets:
 - length
 - minLength
 - maxLength
 - pattern
 - enumeration
 - whitespace (legal values: preserve, replace, collapse)

The ISBN Datatype

- No built-in ISBNType, so ISBNType to be defined in schema
 - 1. restriction of the **string** type,
using **pattern** facet:
 - first pattern: d-ddddd-ddd-d
 - second pattern: d-ddd-ddddd-d
 - third pattern: d-dd-ddddddd-d
 - where 'd' stands for 'digit'
 - 2. use the **simpleType** Schema element to create a new type
that is a refinement of a built-in type

The ISBN Datatype

```
<xsd:simpleType name="ISBNType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{1}-\d{5}-\d{3}-\d{1}"/>
    <xsd:pattern value="\d{1}-\d{3}-\d{5}-\d{1}"/>
    <xsd:pattern value="\d{1}-\d{2}-\d{6}-\d{1}"/>
  </xsd:restriction>
</xsd:simpleType>
```

```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              targetNamespace="http://www.books.org"
              xmlns="http://www.books.org"
              elementFormDefault="qualified">

    <xsd:simpleType name="ISBNTYPE">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="\d{1}-\d{5}-\d{3}-\d{1}"/>
            <xsd:pattern value="\d{1}-\d{3}-\d{5}-\d{1}"/>
            <xsd:pattern value="\d{1}-\d{2}-\d{6}-\d{1}"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:element name="BookStore">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Book" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="Title" type="xsd:string"/>
                            <xsd:element name="Author" type="xsd:string"/>
                            <xsd:element name="Date" type="xsd:gYear"/>
                            <xsd:element name="ISBN" type="ISBNTYPE"/>
                            <xsd:element name="Publisher" type="xsd:string"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>

```

Here we are defining a new data-type, called **ISBNTYPE**.

Declaring **Date** to be of type **gYear**, and **ISBN** to be of type **ISBNTYPE** (defined above)

The ISBN Datatype

- Equivalent Expressions

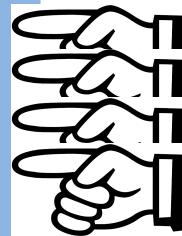
```
<xsd:simpleType name="ISBNType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{1}-\d{5}-\d{3}-\d{1}" />
    <xsd:pattern value="\d{1}-\d{3}-\d{5}-\d{1}" />
    <xsd:pattern value="\d{1}-\d{2}-\d{6}-\d{1}" />
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="ISBNType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{1}-\d{5}-\d{3}-\d{1} |
                           \d{1}-\d{3}-\d{5}-\d{1} |
                           \d{1}-\d{2}-\d{6}-\d{1}" />
  </xsd:restriction>
</xsd:simpleType>
```

User-defined simpleTypes

- Specifying facet values

```
<xsd:simpleType name="TelephoneNumber">
  <xsd:restriction base="xsd:string">
    <xsd:length value="8"/>
    <xsd:pattern value="\d{3}-\d{4}" />
  </xsd:restriction>
</xsd:simpleType>
```



This creates a new
Elements of this type
can hold string
values
follow the pattern:
ddd-dddd, where 'd'
represents a 'digit'

- Obviously, in this example the regular expression makes the length facet redundant.

Pattern Facet

- Regular Expressions
 - Pattern facets defined by regular expressions
 - examples:

Regular expressions

Chapter \d
Chapter \d
a*b
[xyz]b
a?b
a+b
[a-c]x

Facet instances

Chapter 1
Chapter 1
b, ab, aab, aaab, ...
xb, yb, zb
b, ab
ab, aab, aaab, ...
ax, bx, cx

Pattern Facet

- Regular Expressions (cont'd.)

- Regular Expression
 - [a-c]x
 - [\\-ac]x
 - [ac\\-]x
 - [^0-9]x
 - \\Dx
 - Chapter\\s\\d
 - (ho){2} there
 - (ho\\s){2} there
 - .abc
 - (a|b)+x

- Explained
 - x, bx, cx
 - x, ax, cx (Backslash causes metacharacter "-" to be treated as literal char.)
 - ax, cx, -x
 - any non-digit char followed by x
 - any non-digit char followed by x
 - "Chapter" followed by a blank followed by a digit
 - hoho there
 - ho ho there
 - any (*one*) char followed by abc
 - ax, bx, aax, bbx, abx, bax, ...

Pattern Facet

- Regular Expressions (cont'd.)

a{1,3}x

a{2,}x

\w\s\w

[a-zA-Z-[OI]]*

\.

ax, aax, aaax

aax, aaax, aaaax, ...

word **character** (alphanumeric plus dash)
followed by a space followed by a word
character

A string comprised of any lower and upper
case letters, except "O" and "I"

The period ":"

Pattern Facet

- Regular Expressions (concluded)
 - with definitions from Unicode

`\p{L}` A letter, from any language

`\p{Lu}` An uppercase letter, from any language

`\p{Li}` A lowercase letter, from any language

`\p{N}` A number - Roman, fractions, etc

`\p{Nd}` A digit from any language

`\p{P}` A punctuation symbol

`\p{Sc}` A currency sign, from any language

Pattern Facet

- Sample regular expression

```
<xsd:simpleType name="money">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\p{Sc}\p{Nd}+(\.\p{Nd}\p{Nd})?" />
  </xsd:restriction>
</xsd:simpleType>
<xsd:element name="cost" type="money"/>
```

<cost>\$45.99</cost>
<cost>¥300</cost>

"currency sign from any language,
followed by one or more digits
from any language, optionally
followed by a period and two
digits from any language"

Pattern Facet

- Sample regular expression

$$[1-9]?[0-9] | 1[0-9][0-9] | 2[0-4][0-9] | 25[0-5]$$


The diagram illustrates the value ranges for each component of the regular expression. It consists of four horizontal blue brackets with vertical tick marks at their midpoints. Below each bracket is a range of values: the first bracket covers 0 to 99, the second covers 100 to 199, the third covers 200 to 249, and the fourth covers 250 to 255.

0 to 99 100 to 199 200 to 249 250 to 255

- This regular expression restricts a *string* to have values between 0 and 255.
- Such a R.E. might be useful in describing an IPv4 address ...

Pattern Facet

- IP Datatype Definition

```
<xsd:simpleType name="IP">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="(([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}
      ([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])">
    <xsd:annotation>
      <xsd:documentation>
        Datatype for representing IP addresses. Examples,
        129.83.64.255, 64.128.2.71, etc.
        This datatype restricts each field of the IP address
        to have a value between zero and 255, i.e.,
        [0-255].[0-255].[0-255].[0-255]
        Note: In the value attribute (above) the regular expression
        has been split over two lines. This is for readability
        purposes only. In practice the R.E. would all be on one line.
      </xsd:documentation>
    </xsd:annotation>
  </xsd:pattern>
</xsd:restriction>
</xsd:simpleType>
```

User-defined simpleTypes

- Enumeration facet

```
<xsd:simpleType name="shape">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="circle"/>
    <xsd:enumeration value="triangle"/>
    <xsd:enumeration value="square"/>
  </xsd:restriction>
</xsd:simpleType>
```

- This creates a new type called shape.
- An element declared to be of this type must have either the value circle, or triangle, or square.

User-defined simpleTypes

- Facets of the integer datatype
 - The integer datatype has 8 optional facets:
 - totalDigits
 - pattern
 - whitespace
 - enumeration
 - maxInclusive
 - maxExclusive
 - minInclusive
 - minExclusive

User-defined simpleTypes

- min/max facets

```
<xsd:simpleType name="EarthSurfaceElevation">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="-1290"/>
    <xsd:maxInclusive value="29035"/>
  </xsd:restriction>
</xsd:simpleType>
```

- This creates a new datatype called 'EarthSurfaceElevation'.
- Elements declared to be of this type can hold an integer.
- However, the integer is restricted to have a value between -1290 and 29035, inclusive.

User-defined simpleTypes

- Specifying facet values

```
<xsd:simpleType name="name">
  <xsd:restriction base="xsd:source">
    <xsd:facet value="value" />
    <xsd:facet value="value" />
    ...
  </xsd:restriction>
</xsd:simpleType>
```

→ Facets:

- | | |
|---------------|----------------|
| - length | - minInclusive |
| - minlength | - maxInclusive |
| - maxlength | - minExclusive |
| - pattern | - maxExclusive |
| - enumeration | ... |

- Sources:
- string
 - boolean
 - number
 - float
 - double
 - duration
 - dateTime
 - time
 - ...

User-defined simpleTypes

- Multiple facets
 - "and" them together, or
 - "or" them together?

```
<xsd:simpleType name="TelephoneNumber">
  <xsd:restriction base="xsd:string">
    <xsd:length value="8"/>
    <xsd:pattern value="\d{3}-\d{4}"/>
  </xsd:restriction>
</xsd:simpleType>
```

An element declared to be of type TelephoneNumber must be a string of length=8 *and* the string must follow the pattern: 3 digits, dash, 4 digits.

User-defined simpleTypes

- Multiple facets

```
<xsd:simpleType name="TelephoneNumber">
  <xsd:restriction base="xsd:string">
    <xsd:length value="8"/>
    <xsd:pattern value="\d{3}-\d{4}"/>
  </xsd:restriction>
</xsd:simpleType>
```

An element declared to be of type TelephoneNumber must be a string of length=8 *and* the string must follow the pattern: 3 digits, dash, 4 digits.

```
<xsd:simpleType name="shape">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="circle"/>
    <xsd:enumeration value="triangle"/>
    <xsd:enumeration value="square"/>
  </xsd:restriction>
</xsd:simpleType>
```

An element declared to be of type shape must be a string with a value of *either* circle, *or* triangle, *or* square.

- Patterns, enumerations: "or" them together
- All other facets: "and" them together

User-defined simpleTypes

- Creating a simpleType from another simpleType
 - Thus far we have created a simpleType using one of the built-in datatypes as our base type.
 - However, we can create a simpleType that uses another simpleType as the base.

User-defined simpleTypes

- This simpleType uses EarthSurfaceElevation as its base type.

```
<xsd:simpleType name="EarthSurfaceElevation">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="-1290"/>
    <xsd:maxInclusive value="29035"/>
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="BostonAreaSurfaceElevation">
  <xsd:restriction base="EarthSurfaceElevation">
    <xsd:minInclusive value="0"/>
    <xsd:maxInclusive value="120"/>
  </xsd:restriction>
</xsd:simpleType>
```



User-defined simpleTypes

- Fixing a facet value
 - Sometimes it might be required that one (or more) facets have an unchanging value: facet is a constant.

```
<xsd:simpleType name="ClassSize">
  <xsd:restriction base="xsd:nonNegativeInteger">
    <xsd:minInclusive value="10" fixed="true"/>
    <xsd:maxInclusive value="60"/>
  </xsd:restriction>
</xsd:simpleType>
```

simpleTypes which derive from this simpleType may not change this facet.

User-defined simpleTypes

```
<xsd:simpleType name="ClassSize">
  <xsd:restriction base="xsd:nonNegativeInteger">
    <xsd:minInclusive value="10" fixed="true"/>
    <xsd:maxInclusive value="60"/>
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="BostonIEEEClassSize">
  <xsd:restriction base="ClassSize">
    <xsd:minInclusive value="15"/>
    <xsd:maxInclusive value="60"/>
  </xsd:restriction>
</xsd:simpleType>
```

Error! Cannot
change the value
of a fixed facet!

User-defined simpleTypes

- Create an element declaration for an elevation element.
 - Declare the elevation element to be an integer with a range -1290 to 29035
 - Sample instance document:

```
<elevation>5240</elevation>
```

User-defined simpleTypes

- Element Containing a User-Defined Simple Type
 - Here's one way of declaring the elevation element:

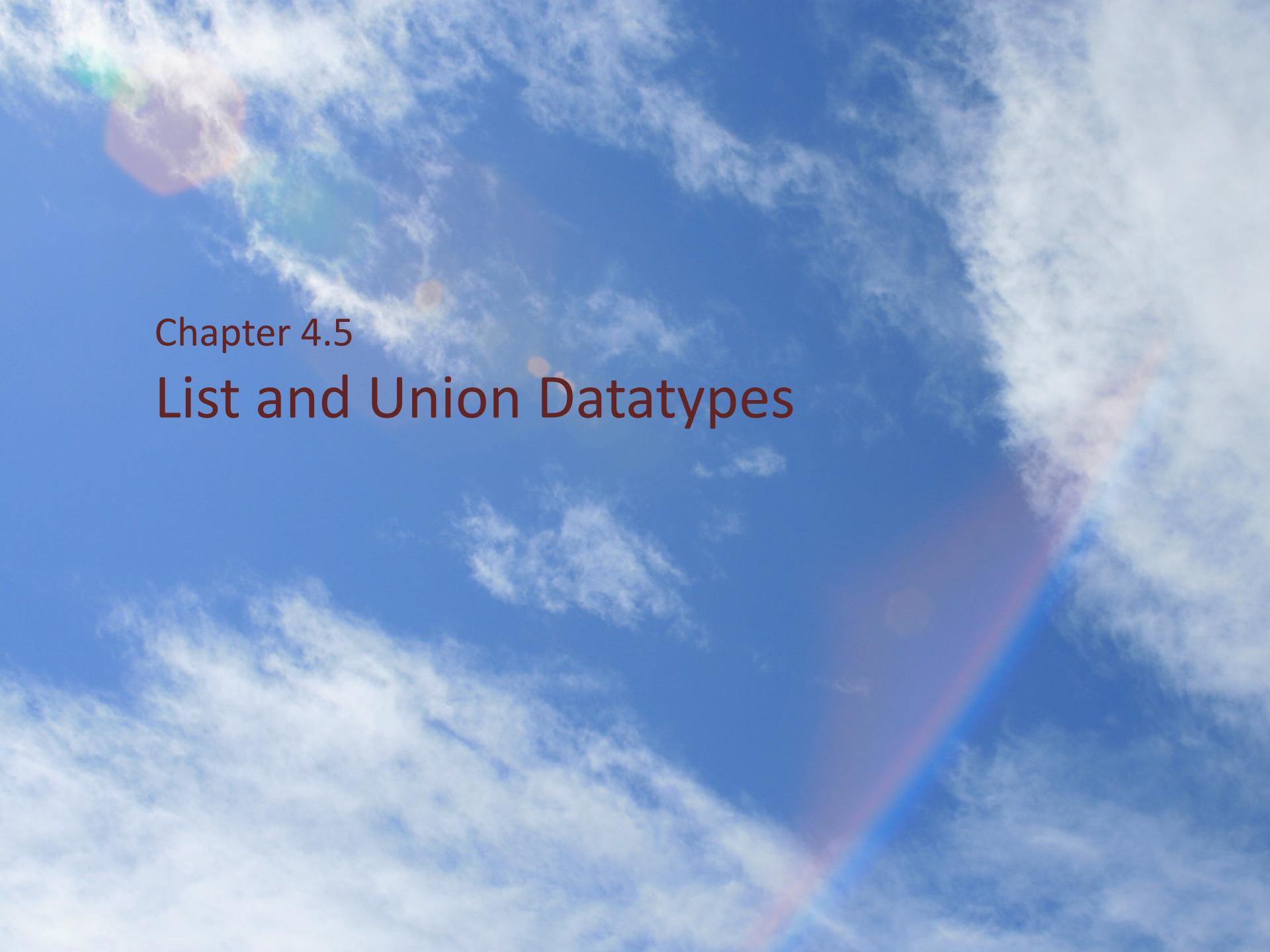
```
<xsd:simpleType name="EarthSurfaceElevation">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="-1290"/>
    <xsd:maxInclusive value="29035"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:element name="elevation" type="EarthSurfaceElevation"/>
```

User-defined simpleTypes

- Element Containing a User-Defined Simple Type
 - Here's an alternative method for declaring elevation:

```
<xsd:element name="elevation">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="-1290"/>
      <xsd:maxInclusive value="29035"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

The simpleType definition is defined inline, it is an *anonymous* simpleType definition.
The disadvantage of this approach is that this simpleType may not be reused by other elements.



Chapter 4.5

List and Union Datatypes

Datatypes: Definitions

- Atomic datatypes
 - Atomic datatypes are those having values which are regarded by this specification as being indivisible.
- List datatypes
 - List datatypes are those having values each of which consists of a finite-length (possibly empty) sequence of values of an atomic datatype.
- Union datatypes
 - Union datatypes are those whose value spaces and lexical spaces are the union of the value spaces and lexical spaces of one or more other datatypes.

List Datatypes

- Creating lists
 - There are times when you will want an element to contain a list of values, e.g., "The contents of the `Numbers` element is a list of numbers".

Example: For a document containing a Lottery drawing we might have

```
<Numbers>12 49 37 99 20 67</Numbers>
```

How do we declare the element `Numbers` ...

- (1) To contain a list of integers, and
- (2) Each integer is restricted to be between 1 and 99, and
- (3) The total number of integers in the list is exactly six.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.lottery.org"
    xmlns="http://www.lottery.org" elementFormDefault="qualified">
    <xsd:simpleType name="LotteryNumbers">
        <xsd:list itemType="xsd:positiveInteger"/>
    </xsd:simpleType>
    <xsd:element name="LotteryDrawings">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Drawing" minOccurs="0" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="Week" type="xsd:string"/>
                            <xsd:element name="Numbers" type="LotteryNumbers"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

```
<?xml version="1.0"?>
<LotteryDrawings xmlns="http://www.lottery.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.lottery.org Lottery.xsd">
    <Drawing>
        <Week>July 1</Week>
        <Numbers>21 3 67 8 90 12</Numbers>
    </Drawing>
    <Drawing>
        <Week>July 8</Week>
        <Numbers>55 31 4 57 98 22</Numbers>
    </Drawing>
    <Drawing>
        <Week>July 15</Week>
        <Numbers>70 77 19 35 44 11</Numbers>
    </Drawing>
</LotteryDrawings>
```

List Datatypes

- List datatypes
 - stronger typing:
 - Restrict the list to length value="6"
 - Restrict the numbers to maxInclusive value="49"

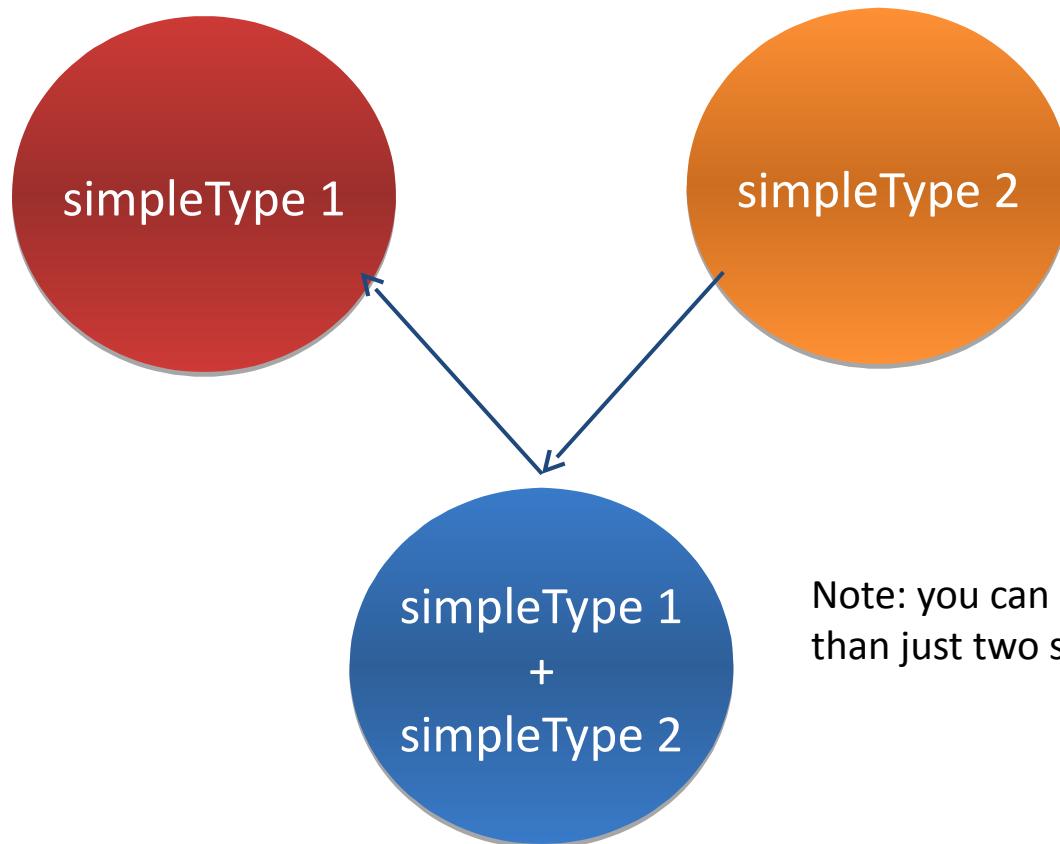
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.lottery.org"
    xmlns="http://www.lottery.org" elementFormDefault="qualified">
    <xsd:simpleType name="OneToFortyNine">
        <xsd:restriction base="xsd:positiveInteger">
            <xsd:maxInclusive value="49"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="NumbersList">
        <xsd:list itemType="OneToFortyNine"/>
    </xsd:simpleType>
    <xsd:simpleType name="LotteryNumbers">
        <xsd:restriction base="NumbersList">
            <xsd:length value="6"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:element name="LotteryDrawings">
        ...
    </xsd:element>
</xsd:schema>
```

List Datatypes

- Notes about the list type
 - You cannot create a list of lists.
 - You cannot create a list of complexTypes.
 - In the instance document, list items are separated by white space chars (blank space, tab, or carriage return)
 - Facets allowed with a list type
 - length: use this to specify the length of the list
 - minLength: use this to specify the minimum length of the list
 - maxLength: use this to specify the maximum length of the list
 - enumeration: use this to specify the values that the list may have
 - pattern: use this to specify the values that the list may have

Union Datatypes

- Creating a simpleType that is a union of types



Note: you can create a union of more than just two simpleTypes

Union Datatypes

- Union type definition

```
<xsd:simpleType name="name">
    <xsd:union memberTypes="space-delimited simpleTypes"/>
</xsd:simpleType>
```

— or

```
<xsd:simpleType name="name">
    <xsd:union>
        <xsd:simpleType>
            ...
        </xsd:simpleType>
        <xsd:simpleType>
            ...
        </xsd:simpleType>
        ...
    </xsd:union>
</xsd:simpleType>
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             targetNamespace="http://www.CostelloReunion.org"
             xmlns="http://www.CostelloReunion.org"
             elementFormDefault="qualified">
  <xsd:simpleType name="Parent">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Mary"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="PatsFamily">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Pat"/>
      <xsd:enumeration value="Patti"/>
      <xsd:enumeration value="Christopher"/>
      <xsd:enumeration value="Elizabeth"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="BarbsFamily">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Barb"/>
      <xsd:enumeration value="Greg"/>
      <xsd:enumeration value="Dan"/>
      <xsd:enumeration value="Kimberly"/>
    </xsd:restriction>
  </xsd:simpleType>
```

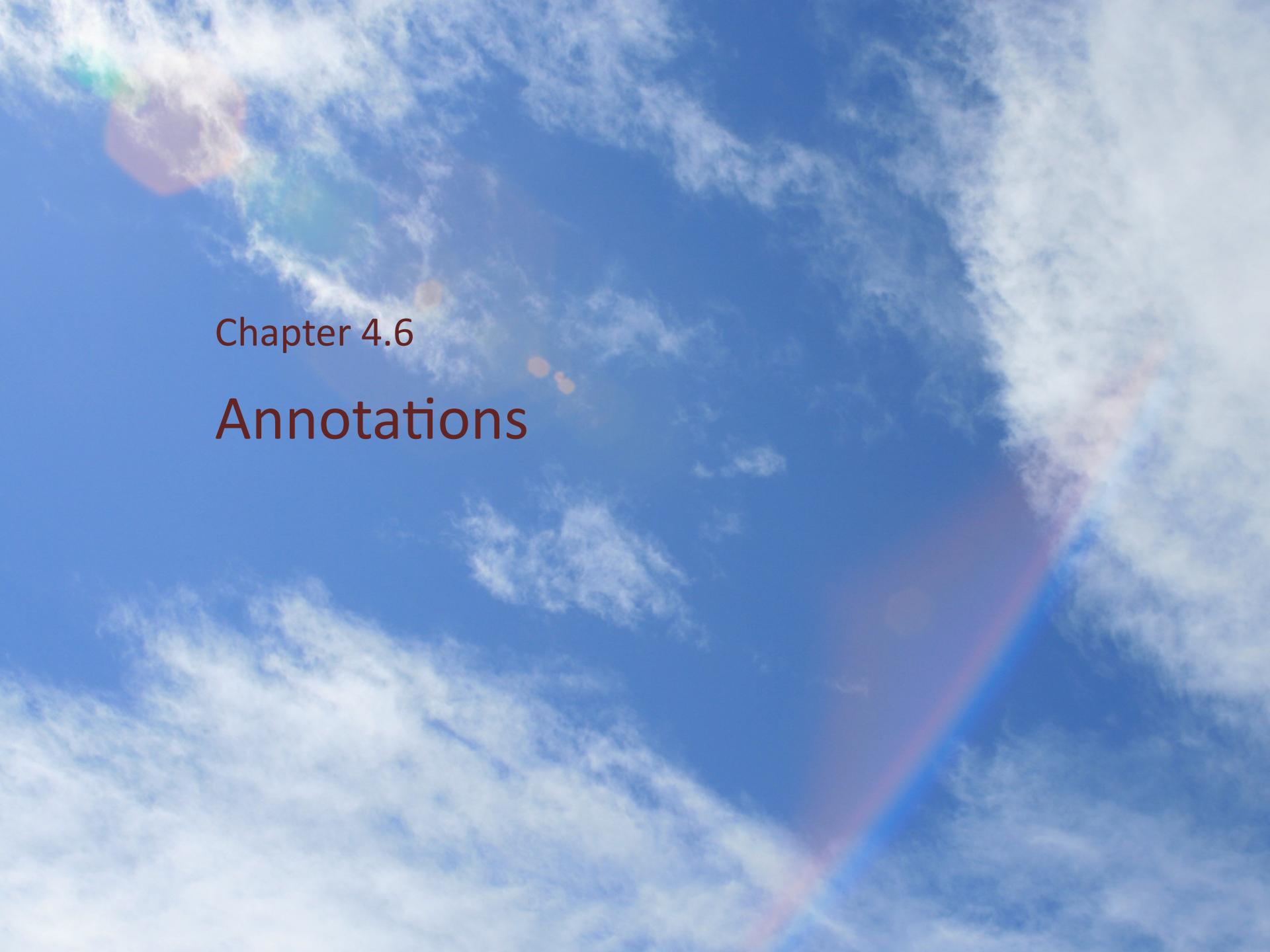
```
<xsd:simpleType name="JudysFamily">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Judy"/>
    <xsd:enumeration value="Peter"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="TomsFamily">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Tom"/>
    <xsd:enumeration value="Cheryl"/>
    <xsd:enumeration value="Marc"/>
    <xsd:enumeration value="Joe"/>
    <xsd:enumeration value="Brian"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="RogersFamily">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Roger"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="JohnsFamily">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="John"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="CostelloFamily">
  <xsd:union memberTypes= "Parent PatsFamily BarbsFamily
                                JudysFamily TomsFamily RogersFamily
                                JohnsFamily"/>
</xsd:simpleType>
```

```
<xsd:element name="Y2KFamilyReunion">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Participants">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Name" type="CostelloFamily"
              minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Union Datatypes

- "maxOccurs" is a union type!
 - The value space for maxOccurs is the union of the value space for nonNegativeInteger with the value space of a simpleType which contains only one enumeration value—"unbounded".

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              targetNamespace="http://www.maxOccurs.org"
              xmlns="http://www.maxOccurs.org"
              elementFormDefault="qualified">
  <xsd:simpleType name="unbounded_type">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="unbounded"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="maxOccurs_type">
    <xsd:union memberTypes="unbounded_type xsd:nonNegativeInteger"/>
  </xsd:simpleType>
  <xsd:element name="schema">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="element">
          <xsd:complexType>
            <xsd:attribute name="maxOccurs" type="maxOccurs_type" default="1"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```



Chapter 4.6

Annotations

Annotating Schemas

- Three schema elements for schema annotation
 - `<annotation>` element:
 - used for documenting the schema, both for humans and for machines.
 - `<documentation>`
 - used for providing a comment to humans
 - `<appinfo>`
 - used for providing a comment to machines
 - content is any well-formed XML
 - Note that annotations have no effect on schema validation

Annotating Schemas

- Example

```
<xsd:annotation>
  <xsd:documentation>
    The following constraint is not expressible with XML Schema:
    The value of element A should be greater than the value of element B.
    So, we need to use a separate tool (e.g., Schematron) to check
    his constraint. We will express this constraint in the
    appinfo section (below).
  </xsd:documentation>
  <xsd:appinfo>
    <assert test="A &gt; B">A should be greater than B</assert>
  </xsd:appinfo>
</xsd:annotation>
```

Annotating Schemas

- Where can you put annotations?
 - Annotations may occur before and after any global component.
 - Annotations may occur only at the beginning of non-global components.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              targetNamespace="http://www.books.org"
              xmlns="http://www.books.org"
              elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Title" type="xsd:string"/>
              <xsd:element name="Author" type="xsd:string"/>
              <xsd:element name="Date" type="xsd:string"/>
              <xsd:element name="ISBN" type="xsd:string"/>
              <xsd:element name="Publisher" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Can put
annotations
only at
these
locations

Inline the annotation within the Date element declaration.

Annotating Schemas

- Two optional attributes for the <documentation> element

```
<xsd:documentation source="http://www.xfront.com/BookReview.txt"
                   xml:lang="FR"/>
```

- source
this attribute contains a URL to a file
which contains supplemental information
- xml:lang
this attribute specifies the language
that the documentation was written in

Annotating Schemas

- One optional attribute for the `<appinfo>` element

```
<xsd:appinfo source="http://www.xfront.com/Assertions.xml"/>
```

- `source`
this attribute contains a URL to a file
which contains supplemental information

Grammars for XML Documents

XML Schema, Part 2

Lecture "XML in Communication Systems"
Chapter 5

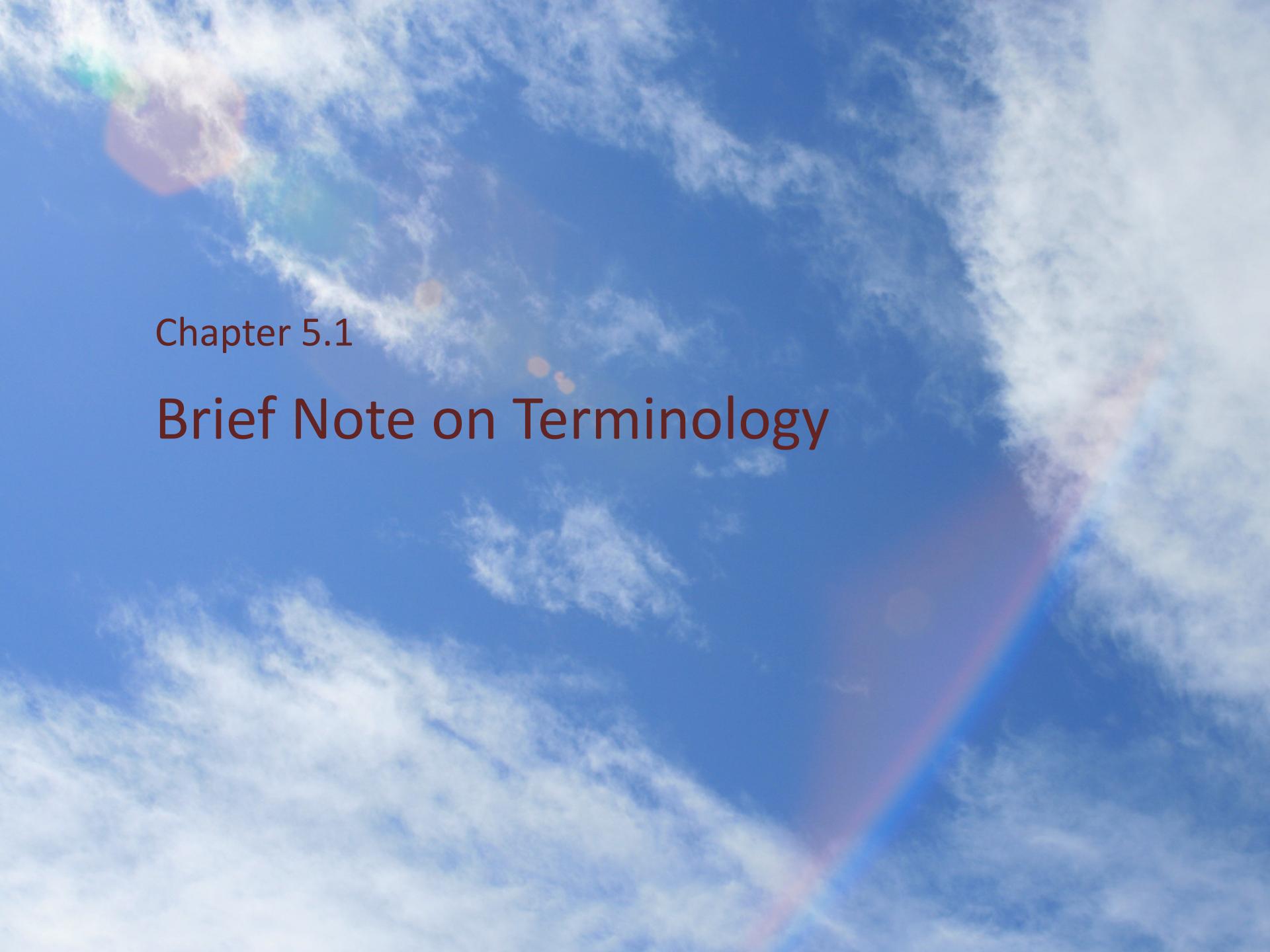
Dr.-Ing. Jesper Zedlitz
Research Group for Communication Systems
Dept. of Computer Science
Christian-Albrechts-University in Kiel



Overview

Informatik · CAU Kiel

1. Brief Note on Terminology
2. Attributes
3. Content Models
4. Element Declaration
5. Uniqueness and Keys
6. Substitution
7. Abstraction



Chapter 5.1

Brief Note on Terminology

Brief Note on Terminology

Informatik · CAU Kiel

- Declaration vs Definition

- In a schema:

- You *declare* elements and attributes. Schema components that are *declared* are those that have a representation in an XML instance document.
 - You *define* components that are used just within the schema document(s). Schema components that are *defined* are those that have no representation in an XML instance document.

Declarations:	Definitions:
<ul style="list-style-type: none">- element declarations- attribute declarations	<ul style="list-style-type: none">- type (simple, complex) definitions- attribute group definitions- model group definitions

Brief Note on Terminology

- Global versus Local
 - Global element declarations, global type definitions:
 - These are element declarations/type definitions that are immediate children of <schema>
 - Local element declarations, local type definitions:
 - These are element declarations/type definitions that are nested within other elements/types.
 - Only global elements/types can be referenced (i.e., reused). Local elements/types are effectively invisible to the rest of the schema (and to other schemas).

```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.books.org" xmlns="http://www.books.org"
    elementFormDefault="qualified">
    <xsd:complexType name="Publication"><!-- Global type definition -->
        <xsd:sequence>
            <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
            <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
            <xsd:element name="Date" type="xsd:gYear"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="BookPublication"><!-- Global type definition -->
        <xsd:complexContent>
            <xsd:extension base="Publication" >
                <xsd:sequence>
                    <xsd:element name="ISBN" type="xsd:string"/>
                    <xsd:element name="Publisher" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="BookStore"><!-- Global element declaration -->
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Book" type="BookPublication" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>

```

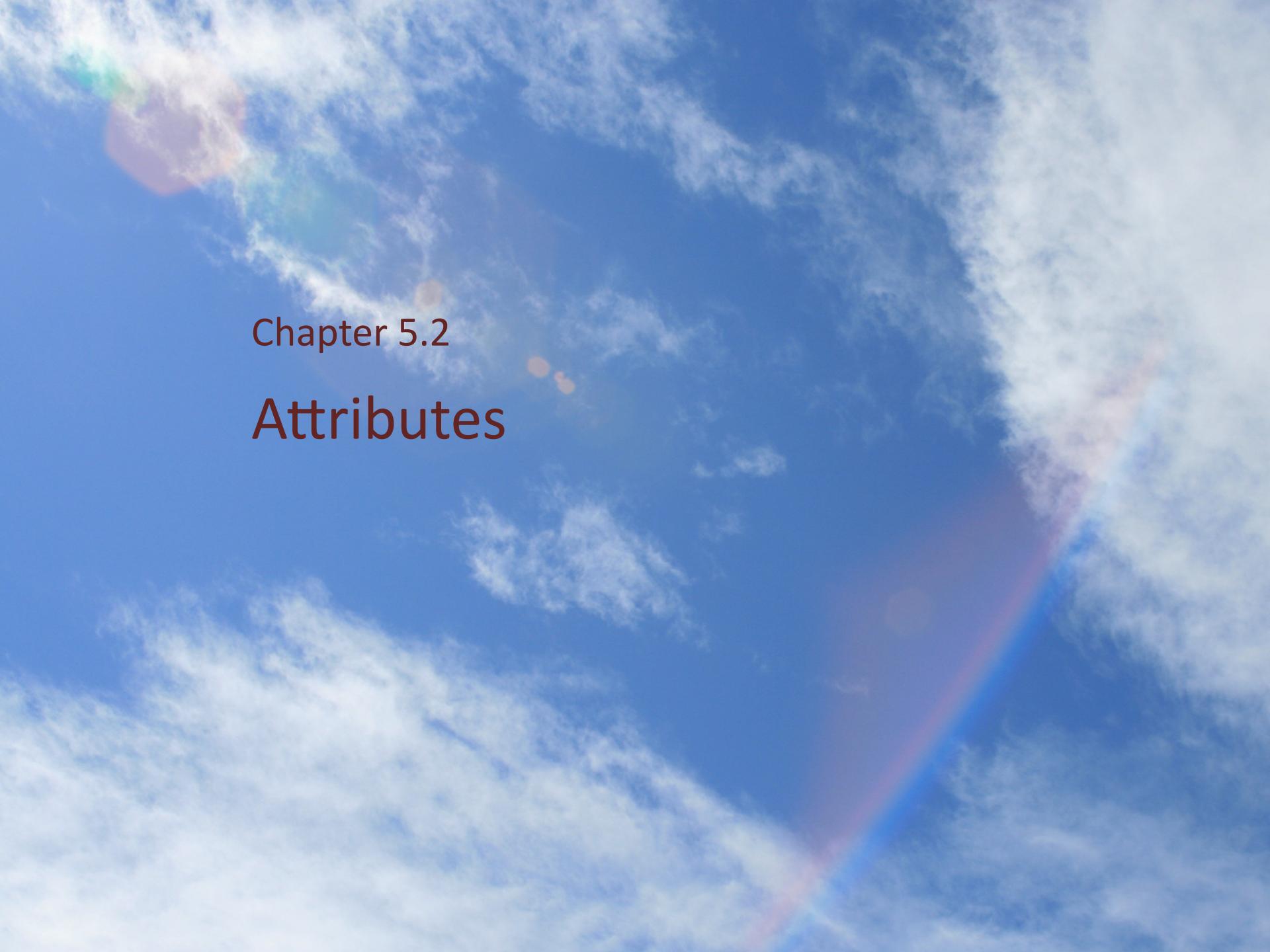
Local type definition

Local element declarations

Global type definition

Global type definition

Global element declaration



Chapter 5.2

Attributes

Declaring Attributes

Informatik · CAU Kiel

- Basics

1

```
<xsd:attribute name/ref="name" type="simple-type" use="how-its-used" default/fixed="value"/>
```

xsd:string
xsd:integer
xsd:boolean
...
(Attributes cannot have child elements.)

required
optional
prohibited

The `use` attribute value must be `optional` if you use `default` or `fixed`.

2

attribute value has embedded derived type

More attributes:
next slide!

```
<xsd:attribute name="name" use="how-its-used" default/fixed="value">  
  <xsd:simpleType>  
    <xsd:restriction base="simple-type">  
      ...
```

Attribute Declaration Attributes

Informatik · CAU Kiel

name	declares the name of the attribute in instance documents.
ref	refers a global attribute declaration.
type	declares the datatype of the attribute in instance documents.
default	gives the attribute a default value.
fixed	gives the attribute a fixed (constant) value.
use	specifies whether the attribute is optional , required , or prohibited . If a default or fixed value is specified, then use must have the value optional . The default value of use is optional .
id	associates a unique identifier to the attribute. This is purely internal to the schema. The type of this value must be xsd:ID .
form	overrides the attributeFormDefault schema attribute. form attribute values are either "qualified" or "unqualified".

Declaring Attributes

- Embedded derived attribute type

```
<xsd:attribute name="Category" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="autobiography"/>
      <xsd:enumeration value="non-fiction"/>
      <xsd:enumeration value="fiction"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
```

- Instance documents are required to have the **Category** attribute (as indicated by `use="required"`).
- The value of **Category** must be either `autobiography`, `non-fiction`, or `fiction` (as specified by the enumeration facets).

Declaring Attributes

- No datatype specified
 - You can declare an attribute without specifying a datatype, e.g.:
`<attribute name="shape"/>`
 - This attribute is unconstrained with respect to the type of value it can have.
 - Creating unconstrained attributes is a very useful technique:
 - Consider creating a `complexType` containing unconstrained attributes, and then create other `complexTypes` which derive by restriction and which constrain the `shape` attribute to a specific datatype. That's nice!

Attributes

- `use="prohibited"` example

```
<xsd:complexType name="shape">
  <xsd:attribute name="length" type="xsd:nonNegativeInteger"/>
  <xsd:attribute name="height" type="xsd:nonNegativeInteger"/>
  <xsd:attribute name="width" type="xsd:nonNegativeInteger"/>
  <xsd:attribute name="radius" type="xsd:nonNegativeInteger"/>
  <xsd:attribute name="diameter" type="xsd:nonNegativeInteger"/>
</xsd:complexType>
```

"master"

```
<xsd:complexType name="box">
  <xsd:complexContent>
    <xsd:restriction base="shape">
      <xsd:attribute name="length" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="height" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="width" type="xsd:nonNegativeInteger"/>
      <xsd:attribute name="radius" type="xsd:nonNegativeInteger" use="prohibited"/>
      <xsd:attribute name="diameter" type="xsd:nonNegativeInteger" use="prohibited"/>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

"subtype"

Local and Global Attributes

- Attributes may be declared locally or globally
 - **local** attribute declarations:
 - Inline an attribute declaration within a `complexType`.
 - **global** attribute declarations
 - attribute declarations are immediate children of `<schema>`

Local and Global Attributes

```
<xsd:schema ... >
  <xsd:element name="Book">
    <xsd:complexType>
      <xsd:sequence>
        ...
        </xsd:sequence>
        <xsd:attribute ref="Category" use="required"/>
        ...
      </xsd:complexType>
    </xsd:element>

<xsd:attribute name="Category"><!--
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="autobiography"/>
      <xsd:enumeration value="fiction"/>
      <xsd:enumeration value="non-fiction"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
```

Local attribute declaration.

Notice the "ref"

Global attribute declaration.

Must NOT have a "use":
"use" only makes sense
in the context of an
element

Declaring Attributes Locally

```
<xsd:element name="Book" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="Category" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="autobiography"/>
          <xsd:enumeration value="non-fiction"/>
          <xsd:enumeration value="fiction"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="InStock" type="xsd:boolean" default="false"/>
    <xsd:attribute name="Reviewer" type="xsd:string" default=" "/>
  </xsd:complexType>
</xsd:element>
```

Local (inlined) attributes

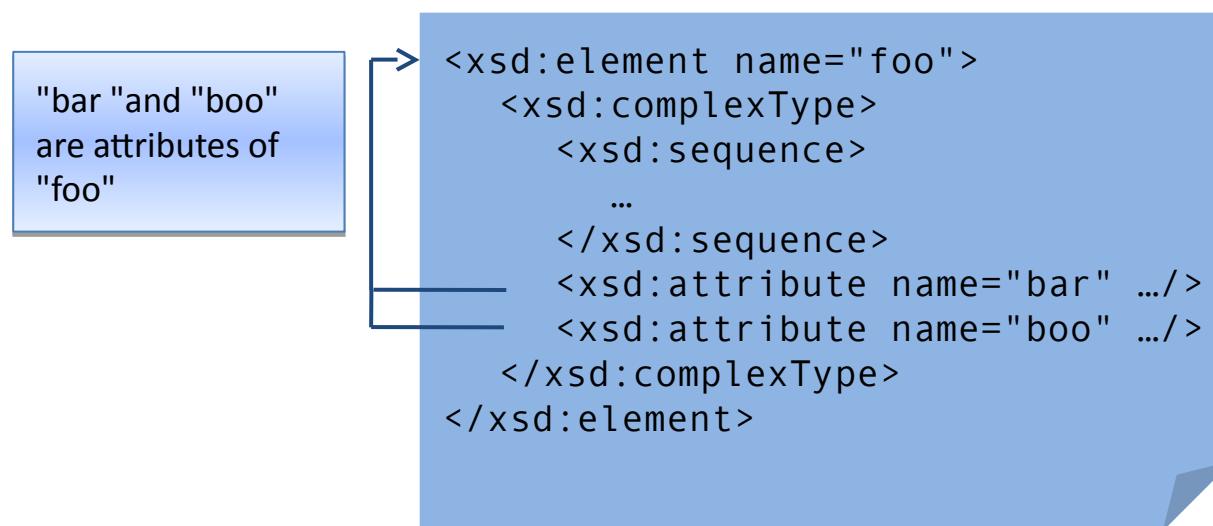
Declaring Attributes Locally

```
<xsd:element name="Book">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
      <xsd:element name="Date" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="Category" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="autobiography"/>
          <xsd:enumeration value="non-fiction"/>
          <xsd:enumeration value="fiction"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="InStock" type="xsd:boolean" default="false"/>
    <xsd:attribute name="Reviewer" type="xsd:string" default=" "/>
  </xsd:complexType>
</xsd:element>
```

These attributes apply to the element they are nested within (Book)
That is, Book has three attributes – Category, InStock, and Reviewer.

Declaring Attributes Locally

- Notes about inlining
 - The attribute declarations always come last, after the element declarations.
 - The attributes are always with respect to the element that they are defined (nested) within.



Attributes for simpleType Elements

- Intermediate summary: Three ways to declare elements



```
<xsd:element name="name" type="type" minOccurs="int" maxOccurs="int" />
```



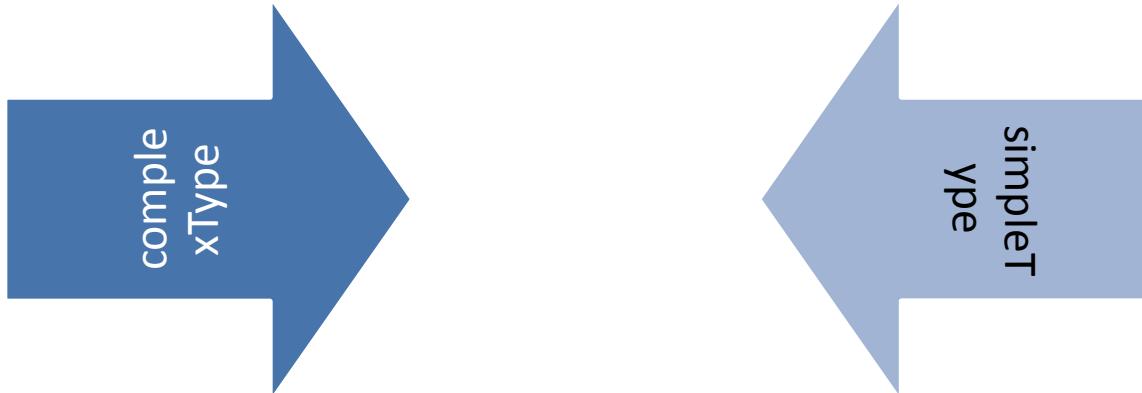
```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
  <xsd:simpleType>
    <xsd:restriction base="type">
      ...
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```



```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
  <xsd:complexType>
    ...
  </xsd:complexType>
</xsd:element>
```

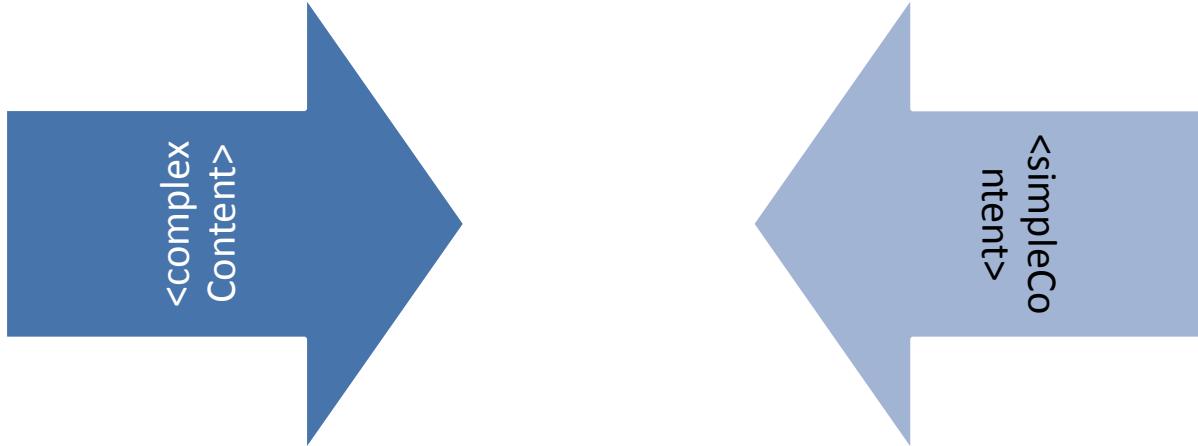
Attributes for simpleType Elements

Informatik · CAU Kiel



- allows elements in its content
 - and
 - may carry attributes
- does **not** allow elements in its content
 - and
 - may **not** carry attributes

Attributes for simpleType Elements



- to extend or restrict a complexType

- to extend or restrict a simpleType

Attributes for simpleType Elements

- Element with simple content and attributes
 - Example. Consider this:

```
<elevation units="feet">5440</elevation>
```

- The elevation element has these two constraints:
 - it has a simple (integer) content
 - it has an attribute called units

Attributes for simpleType Elements

Informatik · CAU Kiel

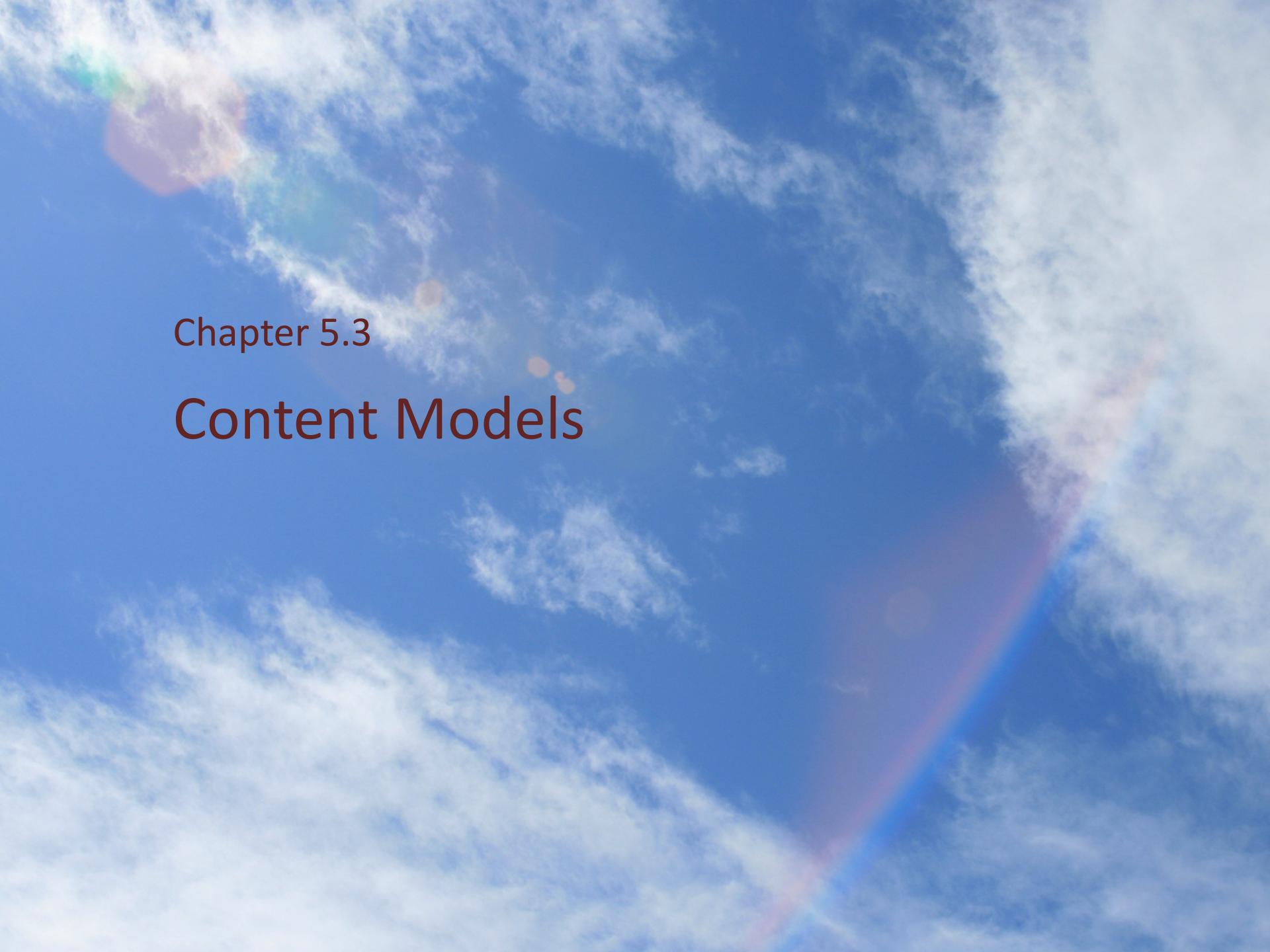
```
<xsd:element name="elevation">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:integer">
        <xsd:attribute name="units" type="xsd:string"
use="required"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

- elevation contains an attribute: `<xsd:complexType>`
- elevation has no child elements: `<xsd:simpleContent>`
- extend the simpleContent (an integer) with an attribute:
`<xsd:extension>`

```
<xsd:simpleType name="elevationRange">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="0"/>
    <xsd:maxInclusive value="12000"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="unitsType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="feet"/>
    <xsd:enumeration value="meters"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name="elevation">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="elevationRange">
        <xsd:attribute name="units" type="unitsType"
use="required"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

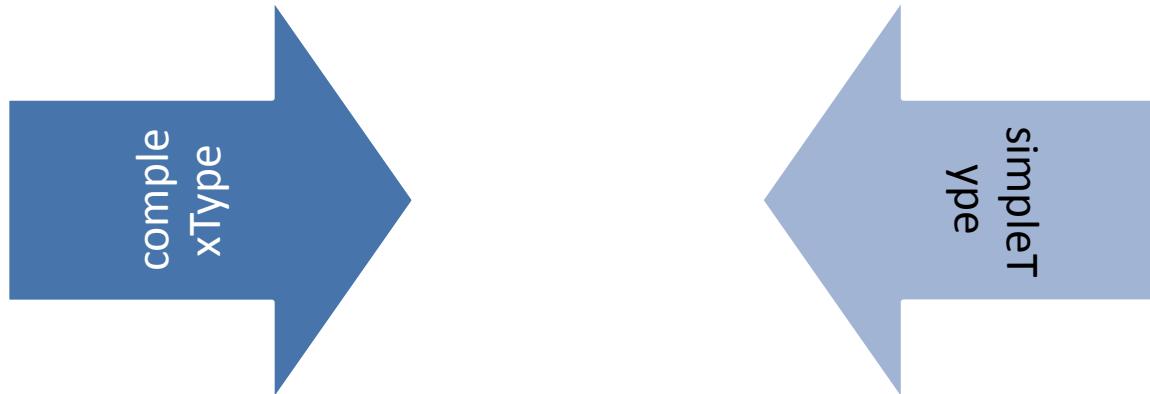


The background of the slide features a clear blue sky with scattered, wispy white clouds. A full, vibrant rainbow arches from the top left towards the bottom right, its colors transitioning smoothly from red at the top to violet at the bottom. The overall effect is bright and airy.

Chapter 5.3

Content Models

Element Declaration



- allows elements in its content
 - and
 - may carry attributes
- does **not** allow elements in its content
 - and
 - may **not** carry attributes

Content Models

- Three kinds of content models
 - sequential (**sequence**):
Particles in specified order.
 - disjunctive (**choice**):
Exactly one of the specified particles.
 - conjunctive (**all**):
all and only exactly zero or one of each particle.
The particles can occur in any order.
To reduce implementation complexity,
only local and top-level element declarations are allowed,
with $\{\text{min occurs}\}=0$ or 1 , $\{\text{max occurs}\}=1$.

Content Models

- Terminology
 - We call **sequence**, **choice**, **all** *compositors*.
 - Compositors specify the sequence of *element information item children content* in a **model group**.
 - Model groups occur inside the following Schema elements
 - **complexType**
 - **group**

Content Models

- **group element**
 - The group element enables you to group together element declarations.
 - Note: the group element is just for grouping together element declarations, no attribute declarations allowed!

```
<xsd:element name="Book" >
  <xsd:complexType>
    <xsd:sequence>
      <xsd:group ref="PublicationElements"/>
      <xsd:element name="ISBN" type="string"/>
      <xsd:element name="Reviewer" type="string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="CD">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:group ref="PublicationElements"/>
      <xsd:element name="RecordingStudio" type="string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:group name="PublicationElements">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string"
maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:string"/>
  </xsd:sequence>
</xsd:group>
```

Content Models

- Group definitions must be global

```
<xsd:element name="Book">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:group name="PublicationElements">
        <xsd:sequence>
          <xsd:element name="Title" type="xsd:string" minOccurs="0"/>
          <xsd:element name="Author" type="xsd:string" minOccurs="0"
                        maxOccurs="unbounded"/>
          <xsd:element name="Date" type="xsd:string"/>
        </xsd:sequence>
      </xsd:group>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
  ...
</xsd:complexType>
</xsd:element>
```

Cannot inline the group definition.

Instead, you must use a *ref* here and define the group globally.

Content Models

- Expressing alternates

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              targetNamespace="http://www.travel.org"
              xmlns="http://www.travel.org"
              elementFormDefault="qualified">
  <xsd:element name="transportation">
    <xsd:complexType>
      <xsd:choice>
        <xsd:element name="train" type="xsd:string"/>
        <xsd:element name="plane" type="xsd:string"/>
        <xsd:element name="automobile" type="xsd:string"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Note: the choice is an exclusive-or, that is, transportation can contain only **one** element—either train, or plane, or automobile.

Content Models

- Expressing repeatable choice

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              targetNamespace="http://www.binary.org"
              xmlns="http://www.binary.org"
              elementFormDefault="qualified">
  <xsd:element name="binary-string">
    <xsd:complexType>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="zero" type="xsd:unsignedByte" fixed="0"/>
        <xsd:element name="one" type="xsd:unsignedByte" fixed="1"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Content Models

- Using <sequence> and <choice>

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              targetNamespace="http://www.life.org"
              xmlns="http://www.life.org" elementFormDefault="qualified">
  <xsd:element name="life">
    <xsd:complexType>
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="work" type="xsd:string"/>
          <xsd:element name="eat" type="xsd:string"/>
        </xsd:sequence>
        <xsd:choice>
          <xsd:element name="work" type="xsd:string"/>
          <xsd:element name="play" type="xsd:string"/> </xsd:choice>
          <xsd:element name="sleep" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
```

Content Models

- Expressing any order

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              targetNamespace="http://www.books.org"
              xmlns="http://www.books.org"
              elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType> <xsd:sequence>
      <xsd:element name="Book" maxOccurs="5">
        <xsd:complexType> <xsd:all>
          <xsd:element name="Title" type="xsd:string"/>
          <xsd:element name="Author" type="xsd:string"/>
          <xsd:element name="Date" type="xsd:string"/>
          <xsd:element name="ISBN" type="xsd:string"/>
          <xsd:element name="Publisher" type="xsd:string"/>
        </xsd:all> </xsd:complexType>
      </xsd:element>
    </xsd:sequence> </xsd:complexType>
  </xsd:element>
```

<all> means that Book must contain all five child elements, but they may occur in any order.

Content Models

- Constraints on using <all>
 - Elements declared within <all> must have a maxOccurs value of "1" (minOccurs can be either "0" or "1")
 - If a complexType uses <all> and it extends another type, then that parent type must have empty content.
 - The <all> element cannot be nested within either <sequence>, <choice>, or another <all>
 - The contents of <all> must be just elements. It cannot contain <sequence> or <choice>

Content Models

- Empty element

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              targetNamespace="http://www.photography.org"
              xmlns="http://www.photography.org"
              elementFormDefault="qualified">
  <xsd:element name="gallery">
    <xsd:complexType> <xsd:sequence>
      <xsd:element name="image" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="href" type="xsd:anyURI"
                         use="required"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence> </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

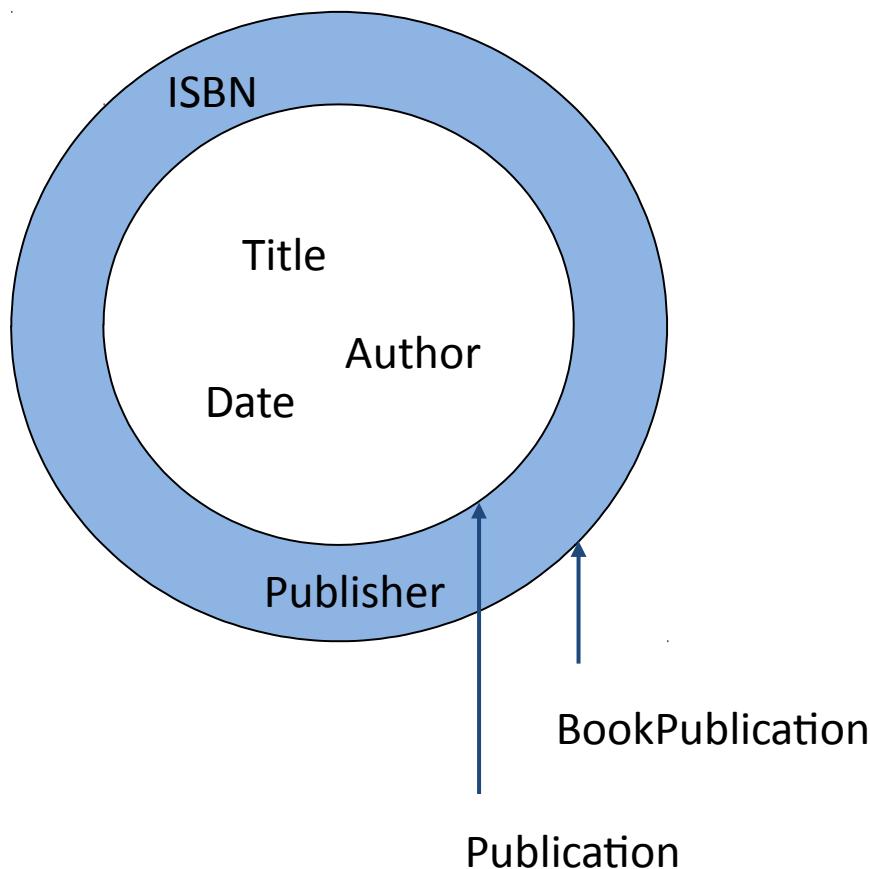
Content Models

- Subclassing complexType definitions
 - derive by extension:
extend the parent complexType with more elements.
 - derive by restriction:
create a type which is a subset of the base type.
 - redefine a base type to have a
restricted range of values
 - redefine a base type to have a
more restricted number of occurrences

Content Models

Informatik · CAU Kiel

- Derive by extension



```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              targetNamespace="http://www.books.org"
              xmlns="http://www.books.org"
              elementFormDefault="qualified">
    > <xsd:complexType name="Publication">
        <xsd:sequence>
            <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
            <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
            <xsd:element name="Date" type="xsd:gYear"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="BookPublication">
        <xsd:complexContent>
            <xsd:extension base="Publication" >
                <xsd:sequence>
                    <xsd:element name="ISBN" type="xsd:string"/>
                    <xsd:element name="Publisher" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="BookStore">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Book" type="BookPublication"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>

```

Note that BookPublication extends the Publication type, i.e., we are doing Derive by Extension

Elements declared to be of type BookPublication will have 5 child elements - Title, Author, Date, ISBN, and Publisher. Note that the elements in the derived type are appended to the elements in the base type.

Content Models

- Derive by restriction

```
<xsd:complexType name="Publication">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:gYear"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SingleAuthorPublication">
  <xsd:complexContent>
    <xsd:restriction base="Publication">
      <xsd:sequence>
        <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="Author" type="xsd:string"/>
        <xsd:element name="Date" type="xsd:gYear"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

There must be exactly one Author element.

In the restriction type you must repeat all the declarations from the base type (exception: see next slide).

Content Models

- Deleting an element in the base type

```
<xsd:complexType name="Publication">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Author" type="xsd:string" minOccurs="0"/>
    <xsd:element name="Date" type="xsd:gYear"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ZeroAuthorPublication">
  <xsd:complexContent>
    <xsd:restriction base="Publication">
      <xsd:sequence>
        <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="Date" type="xsd:gYear"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

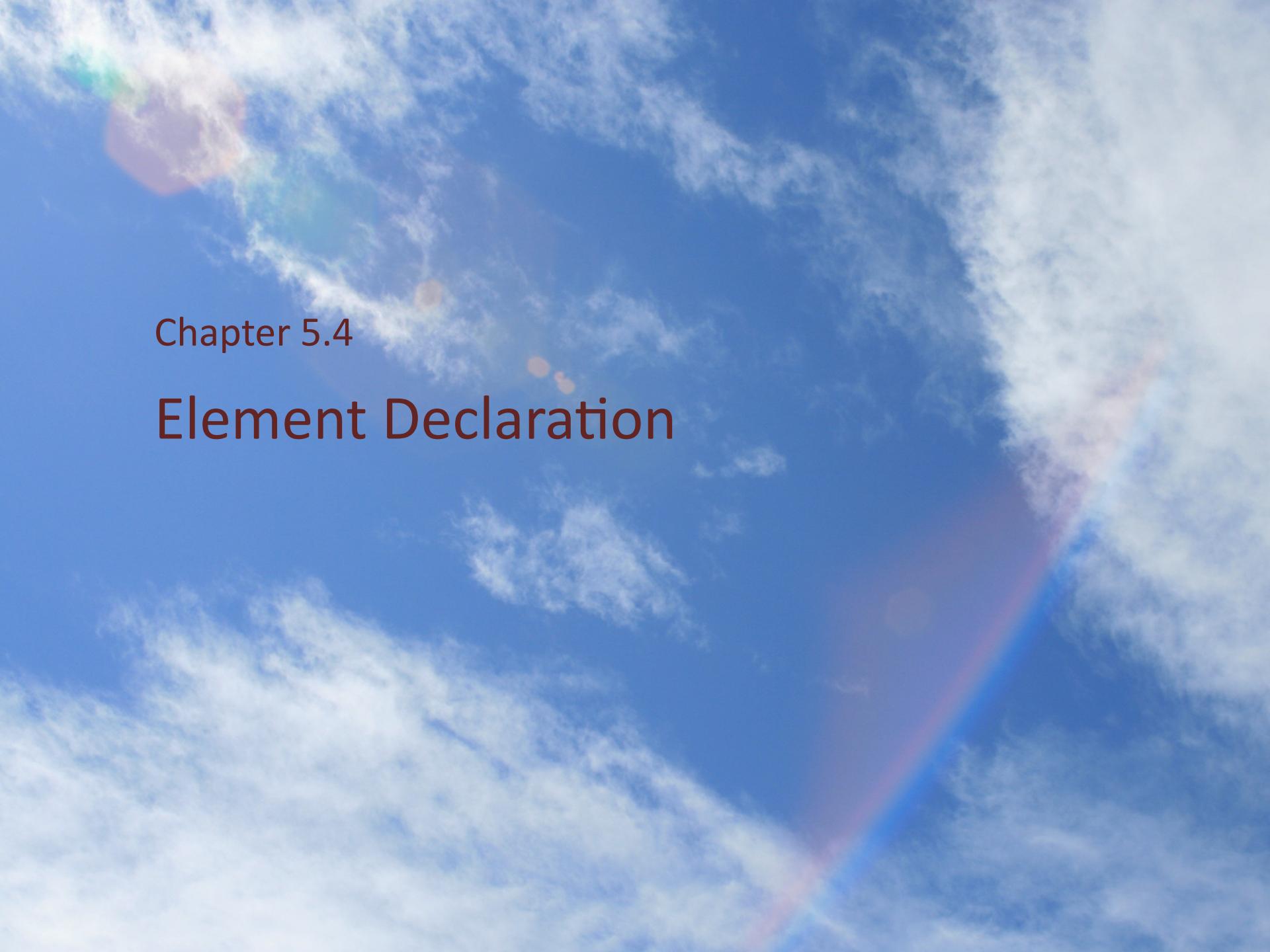
If the base type has an element with `minOccurs="0"`, and the subtype wishes to not have that element, then it can simply leave it out.

In this subtype we have eliminated the Author element, i.e., the subtype is just comprised of an unbounded number of Title elements followed by a single Date element.

Content Models

- Prohibiting Derivations
 - Sometimes we may want to create a type and
 - disallow all derivations of it, or
 - disallow extension derivations, or
 - disallow restriction derivations.

```
<xsd:complexType name="Publication" final="#all" ...>  
  
<xsd:complexType name="Publication" final="extension" ...>  
  
<xsd:complexType name="Publication" final="restriction" ...>
```



Chapter 5.4

Element Declaration

Element Declaration

1. Element with simple content

- Declaring an element using a **built-in** type:

```
<xsd:element name="numStudents" type="xsd:positiveInteger"/>
```

- Declaring an element using a **user-defined** simpleType:

```
<xsd:simpleType name="shapes">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="triangle"/>
    <xsd:enumeration value="rectangle"/>
    <xsd:enumeration value="square"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:element name="geometry" type="shapes"/>
```

Element Declaration

2. Element contains child elements

- Declaring the child elements **inline**

```
<xsd:element name="Person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="FirstName" type="xsd:string"/>
      <xsd:element name="Surname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Element Declaration

2. Element contains child elements

- Create a **named complexType**

```
<xsd:complexType name="PersonType">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="FirstName" type="xsd:string"/>
    <xsd:element name="Surname" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="Person" type="PersonType"/>
```

Element Declaration

3. Extension/restriction

- complexType that is an extension of another complexType

```
<xsd:complexType name="Publication">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:gYear"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="BookPublication">
  <xsd:complexContent>
    <xsd:extension base="Publication" >
      <xsd:sequence>
        <xsd:element name="ISBN" type="xsd:string"/>
        <xsd:element name="Publisher" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="Book" type="BookPublication"/>
```

Element Declaration

3. Extension/restriction

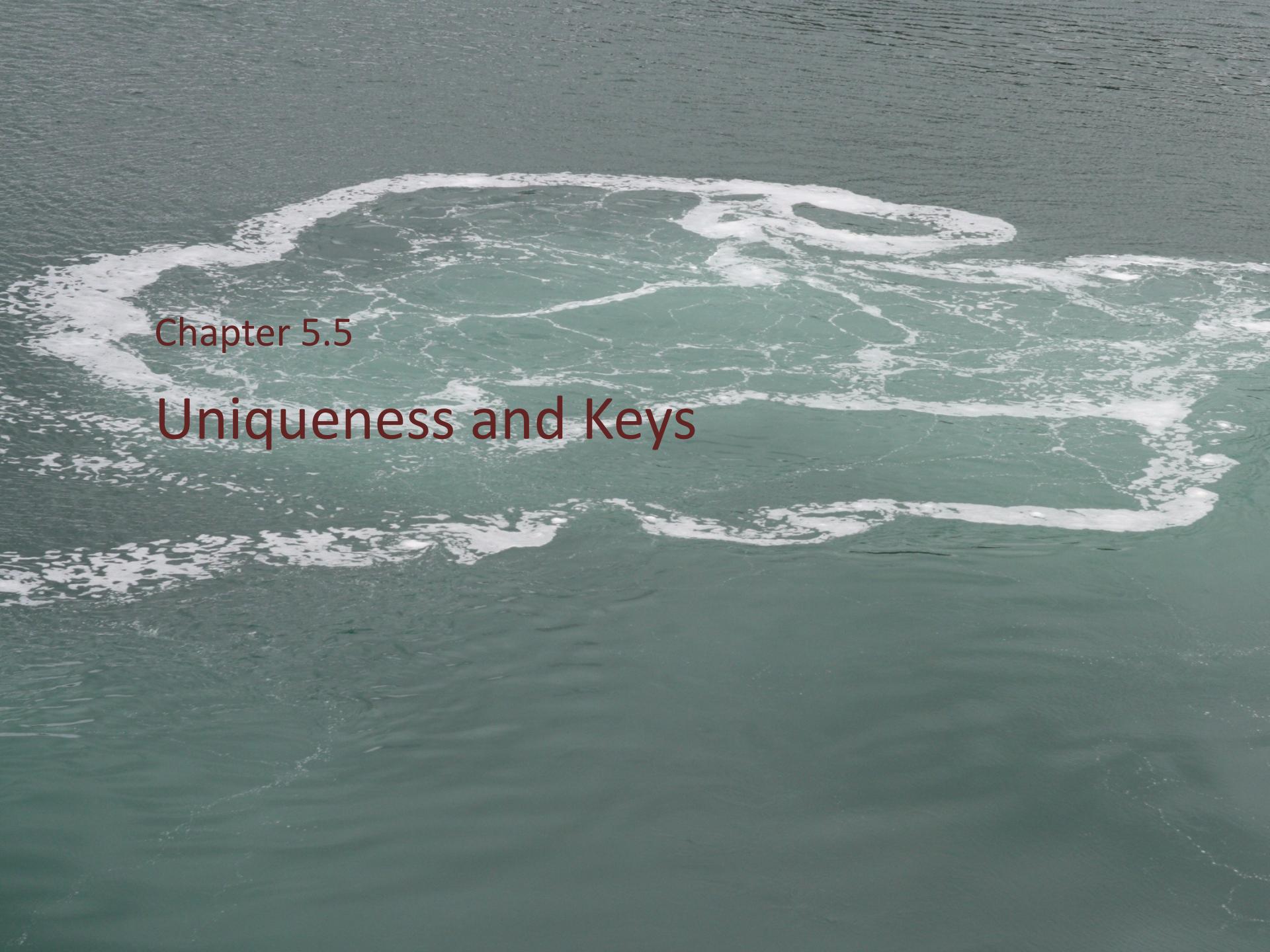
- complexType that is a restriction of another complexType

```
<xsd:complexType name="Publication">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:gYear"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name= "SingleAuthorPublication">
  <xsd:complexContent>
    <xsd:restriction base="Publication">
      <xsd:sequence>
        <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="Author" type="xsd:string"/>
        <xsd:element name="Date" type="xsd:gYear"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="Catalogue" type="SingleAuthorPublication"/>
```

Element Declaration

4. Element contains simple content and attributes

```
<xsd:element name="apple">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="variety"
                      type="xsd:string" use="required"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```



Chapter 5.5

Uniqueness and Keys

Uniqueness & Keys

- XML Schema has enhanced uniqueness capabilities:
 - enables you to define element content to be **unique**.
 - enables you to define non-ID attributes to be unique.
 - enables you to define a combination of element content and attributes to be unique.
 - enables you to distinguish between unique versus **key**.
 - enables you to declare the range of the document over which something is unique.

Key

- Key
 - an element or attribute (or combination thereof) which is defined to be a key must:
 - always be present (minOccurs must be greater than zero)
 - be non-nillable (i.e., nillable="false")
 - have unique content

Key

- Example
 - Within <BookStore> each <Book> must have an <ISBN> and it must be unique.
 - Key is called PK.
 - Select each <Book>, and within each <Book> the ISBN element is a key.

```
<xsd:element name="BookStore">
  ...
  <xsd:key name="PK">
    <xsd:selector xpath="bk:Book"/>
    <xsd:field xpath="bk:ISBN"/>
  </xsd:key>
</xsd:element>
```

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.books.org"
    xmlns="http://www.books.org"
    xmlns:bk="http://www.books.org"
    elementFormDefault="qualified">
    <xsd:element name="BookStore">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Book" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="Title" type="xsd:string"/>
                            <xsd:element name="Author" type="xsd:string"/>
                            <xsd:element name="Date" type="xsd:string"/>
                            <xsd:element name="ISBN" type="xsd:string"/>
                            <xsd:element name="Publisher" type="xsd:string"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:key name="PK">
            <xsd:selector xpath="bk:Book"/>
            <xsd:field xpath="bk:ISBN"/>
        </xsd:key>
    </xsd:element>
</xsd:schema>
```

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation=
                "http://www.books.org
                 BookStore.xsd">

    <Book>
        <Title>My Life and Times</Title>
        <Author>Paul McCartney</Author>
        <Date>1998</Date>
        <ISBN>1-56592-235-2</ISBN> ←
        <Publisher>McMillin Publishing</Publisher>
    </Book>
    <Book>
        <Title>Illusions The Adventures of a Reluctant Messiah</Title>
        <Author>Richard Bach</Author>
        <Date>1977</Date>
        <ISBN>0-440-34319-4</ISBN> ←
        <Publisher>Dell Publishing Co.</Publisher>
    </Book>
    <Book>
        <Title>The First and Last Freedom</Title>
        <Author>J. Krishnamurti</Author>
        <Date>1954</Date>
        <ISBN>0-06-064831-7</ISBN> ←
        <Publisher>Harper & Row</Publisher>
    </Book>
</BookStore>
```

A schema-validator will verify that each Book has an ISBN element and that the values are all unique.

- Properties
 - It must be nested within an <element>
 - It must come at the end of <element>
(after the content model, and attribute declarations)
 - Use the <selector> element as a child of <key> to select a set of elements for which the key applies.
 - Use the <field> element as a child of <key> to identify the element or attribute that is to be the key
 - There can be multiple <field> elements. See next example.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.CostelloReunion.org"
    xmlns="http://www.CostelloReunion.org"
    xmlns:reunion="http://www.CostelloReunion.org"
    elementFormDefault="qualified">
    <xsd:element name="Y2KFamilyReunion">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Participants" >
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="Name" minOccurs="0" maxOccurs="unbounded">
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element name="First" type="xsd:string"/>
                                        <xsd:element name="Last" type="xsd:string"/>
                                    </xsd:sequence>
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:key name="PK">
        <xsd:selector xpath="reunion:Participants/reunion:Name"/>
        <xsd:field xpath="reunion:First"/>
        <xsd:field xpath="reunion:Last"/>
    </xsd:key>
</xsd:element>
</xsd:schema>
```

unique

- The <unique> element is used exactly like the <key> element is used. It has a <selector> and one or more <field> elements, just like <key> has.
- The only difference is that the schema validator will simply validate that, *whenever present*, the values are unique.

```

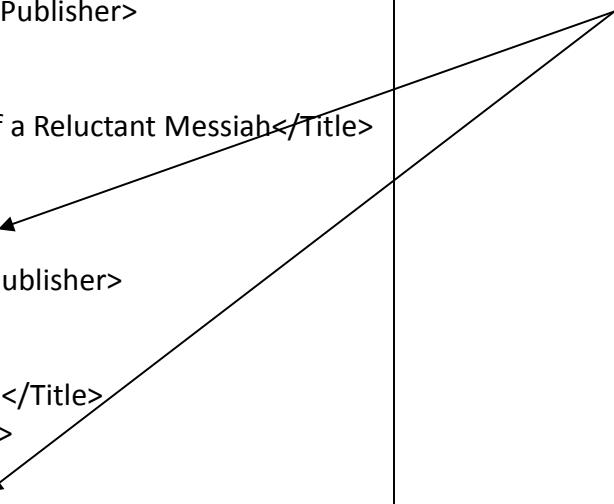
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.books.org"
    xmlns="http://www.books.org"
    xmlns:bk="http://www.books.org"
    elementFormDefault="qualified">
    <xsd:element name="BookStore">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Book" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="Title" type="xsd:string"/>
                            <xsd:element name="Author" type="xsd:string"/>
                            <xsd:element name="Date" type="xsd:string"/>
                            <xsd:element name="ISBN" type="xsd:string" minOccurs="0"/>
                            <xsd:element name="Publisher" type="xsd:string"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>

```

Note: ISBN
is optional

Require
every ISBN
be unique.

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org/namespaces/BookStore"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation=
                "http://www.books.org/namespaces/BookStore
                 BookStore24.xsd">
    <Book>
        <Title>My Life and Times</Title>
        <Author>Paul McCartney</Author>
        <Date>1998</Date>
        <Publisher>McMillin Publishing</Publisher>
    </Book>
    <Book>
        <Title>Illusions The Adventures of a Reluctant Messiah</Title>
        <Author>Richard Bach</Author>
        <Date>1977</Date>
        <ISBN>0-440-34319-4</ISBN>
        <Publisher>Dell Publishing Co.</Publisher>
    </Book>
    <Book>
        <Title>The First and Last Freedom</Title>
        <Author>J. Krishnamurti</Author>
        <Date>1954</Date>
        <ISBN>0-06-064831-7</ISBN>
        <Publisher>Harper & Row</Publisher>
    </Book>
</BookStore>
```



A schema-validator will verify that each Book which has an ISBN element has a unique value (note that the first Book does not have an ISBN. That's perfectly valid!)

Referencing a key

- Recall that by declaring an element of type IDREF then that element must reference an ID attribute, and an XML Parser will verify that the IDREF value corresponds to a legitimate ID value.
- Similarly, you can define a keyref which asserts, "the value of this element must match the value of an element referred to by this".

```
<?xml version="1.0"?>
<Library xmlns="http://www.library.org"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation=
              "http://www.library.org
              AuthorSigningAtLibrary.xsd">

    <Books>
        <Book>
            <Title>Illusions The Adventures of a Reluctant Messiah</Title>
            <Author>Richard Bach</Author>
            <Date>1977</Date>
            > <ISBN>0-440-34319-4</ISBN>
            <Publisher>Dell Publishing Co.</Publisher>
        </Book>
        ...
    </Books>
    <GuestAuthors>
        <Author>
            <Name>Richard Bach</Name>
            <BookForSigning>
                <Title>Illusions The Adventures of a Reluctant Messiah</Title>
                > <ISBN>0-440-34319-4</ISBN>
            </BookForSigning>
        </Author>
    </GuestAuthors>
</Library>
```

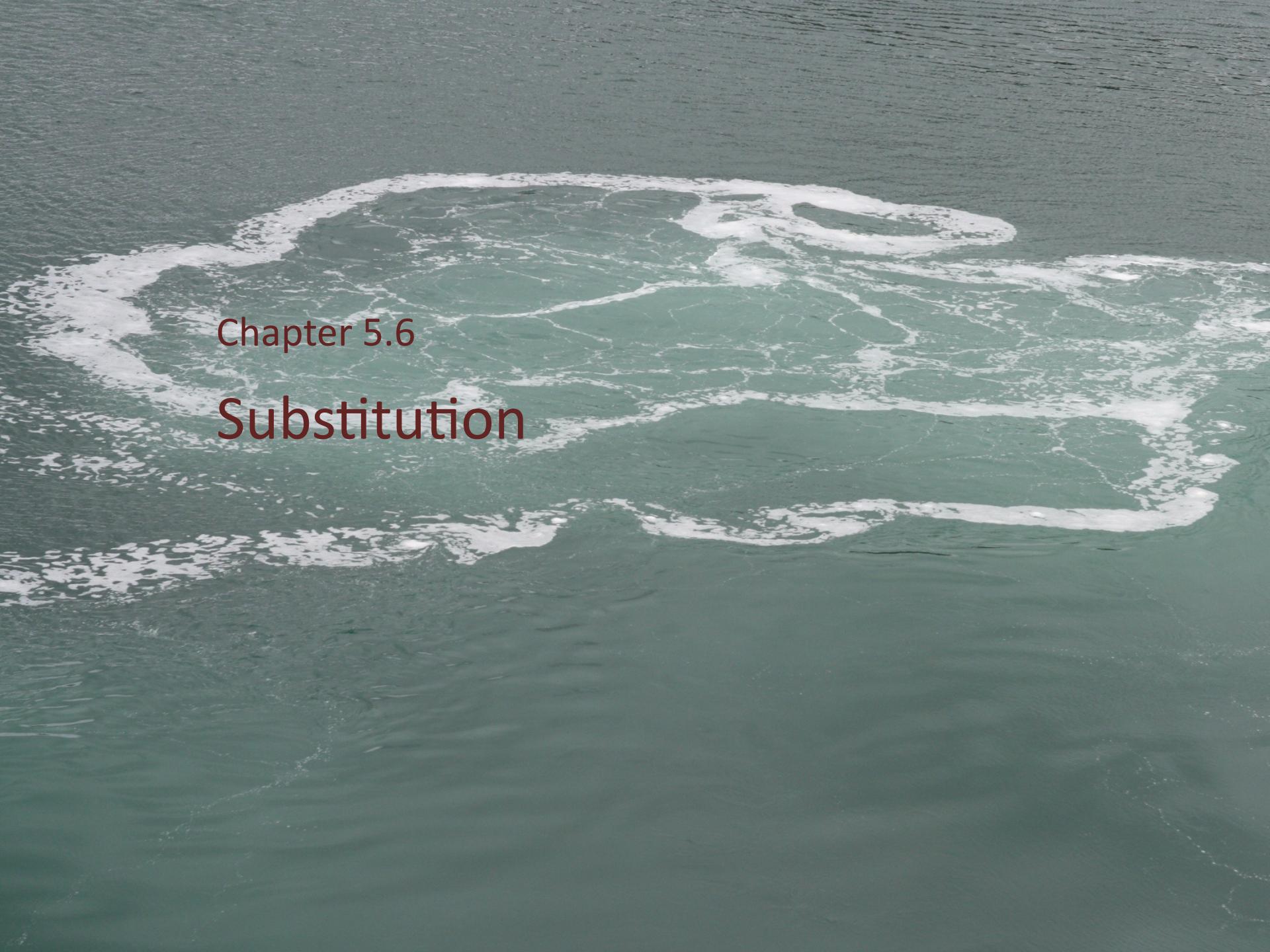
A key element

Suppose each Book must have an ISBN and it must be unique.

We would like to ensure that the ISBN for the GuestAuthor matches one of the ISBNs in the BookStore.

A keyref element

```
<xsd:element name="Library">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Books">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="Book" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element ref="GuestAuthors"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:key name="PK">
    <xsd:selector xpath="bk:Books/bk:Book"/>
    <xsd:field xpath="bk:ISBN"/>
  </xsd:key>
  <xsd:keyref name="isbnRef" refer="PK">
    <xsd:selector xpath="bk:GuestAuthors/bk:Author/bk:BookForSigning"/>
    <xsd:field xpath="bk:ISBN"/>
  </xsd:keyref>
</xsd:element>
```



Chapter 5.6

Substitution

Substitution

Informatik · CAU Kiel

Element Substitution

- Declare in a Schema
 - an element called "subway",
 - an element called "T",
 - and then state
that "T" may be substituted for "subway".
 - Instance documents can then use either `<subway>` or `<T>`,
depending on their preference.



Element Substitution

- **substitutionGroup**

subway is the *head* element

T is substitutable for subway

```
<xsd:element name="subway" type="xsd:string" />
<xsd:element name="T" type="xsd:string"
substitutionGroup="subway" />
```

- Anywhere a head element can be used in an instance document, any member of the substitutionGroup can be substituted!

Element Substitution

```
<xsd:element name="subway" type="xsd:string"/>
<xsd:element name="T" type="xsd:string"
              substitutionGroup="subway" />
<xsd:element name="transportation">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="subway"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Schema



```
<transportation>
  <subway>Red Line</subway>
</transportation>
```

Instance doc



```
<transportation>
  <T>Red Line</T>
</transportation>
```

Alternative
instance doc



Element Substitution

```
<transportation>
  <subway>Red Line</subway>
</transportation>
```



English instance doc

```
<transporte>
  <metro>Linea Roja</metro>
</transporte>
```



Spanish instance doc

```
<xsd:element name="subway" type="xsd:string"/>
<xsd:element name="transportation" type="transport"/>
<xsd:complexType name="transport">
  <xsd:sequence>
    <xsd:element ref="subway"/>
  </xsd:sequence>
</xsd:complexType>
```

```
<xsd:element name="transporte" substitutionGroup="transportation"/>
<xsd:element name="metro" substitutionGroup="subway"/>
```

Element Substitution

- `substitutionGroup` element types

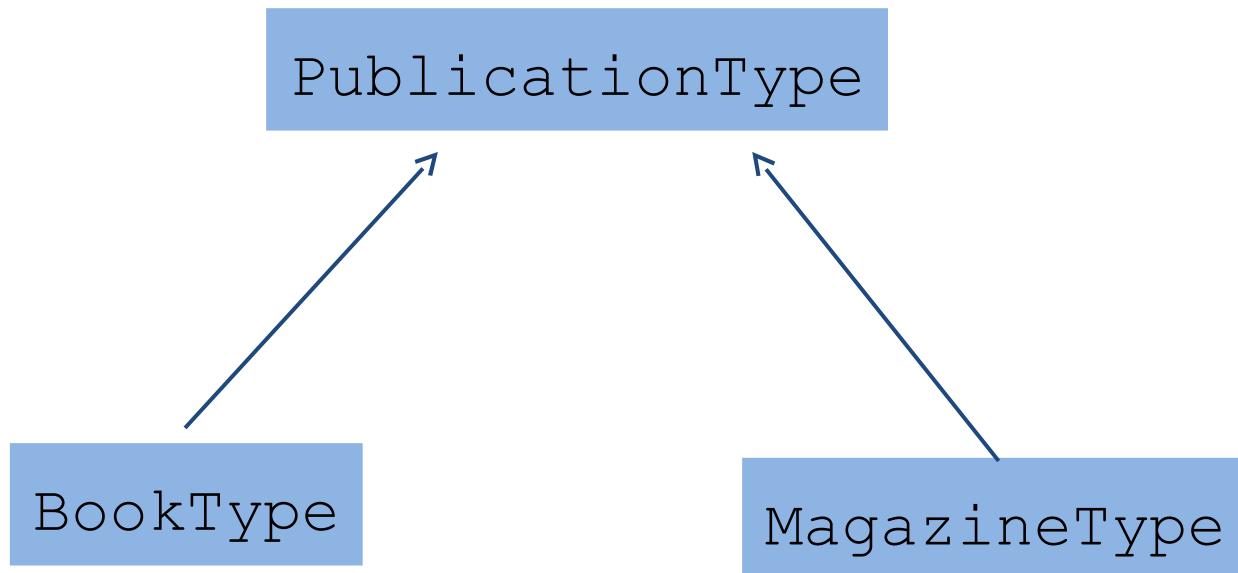
```
<xsd:element name="A" type="xxx"/>
<xsd:element name="B" type="yyy" substitutionGroup="A" />
```



This type must be the same as "xxx"
or it must be derived from "xxx".

Element Substitution

- Element Substitution with Derived Types
 - BookType and MagazineType are derived from PublicationType



```
<xsd:complexType name="PublicationType">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:gYear"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="BookType">
  <xsd:complexContent>
    <xsd:extension base="PublicationType" >
      <xsd:sequence>
        <xsd:element name="ISBN" type="xsd:string"/>
        <xsd:element name="Publisher" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="MagazineType">
  <xsd:complexContent>
    <xsd:restriction base="PublicationType">
      <xsd:sequence>
        <xsd:element name="Title" type="xsd:string"/>
        <xsd:element name="Date" type="xsd:gYear"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

Element Substitution

- Element Substitution with Derived Types

```
<xsd:element name="Publication" type="PublicationType"/>
<xsd:element name="Book"           type="BookType"
              substitutionGroup="Publication"/>
<xsd:element name="Magazine"      type="MagazineType"
              substitutionGroup="Publication"/>
<xsd:element name="BookStore">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Publication" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Element Substitution

```
<?xml version="1.0"?>
<BookStore ...>
  <Book>
    <Title>Illusions: The Adventures of a Reluctant Messiah</Title>
    <Author>Richard Bach</Author>
    <Date>1977</Date>
    <ISBN>0-440-34319-4</ISBN>
    <Publisher>Dell Publishing Co.</Publisher>
  </Book>
  <Magazine>
    <Title>Natural Health</Title>
    <Date>1999</Date>
  </Magazine>
  <Book>
    <Title>The First and Last Freedom</Title>
    <Author>J. Krishnamurti</Author>
    <Date>1954</Date>
    <ISBN>0-06-064831-7</ISBN>
    <Publisher>Harper & Row</Publisher>
  </Book>
</BookStore>
```

<BookStore> can contain
any element in the
substitutionGroup with
Publication.

Element Substitution

- Blocking element substitution
 - An element may wish to block other elements from substituting with it. This is achieved by adding a block attribute.

```
<xsd:element name="..." type="..." block="substitution"/>
```

Element Substitution

```
<xsd:element name="subway" type="xsd:string" block="substitution"/>
<xsd:element name="T" substitutionGroup="subway"/>
<xsd:element name="transportation">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="subway"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<transportation>
  <subway>Red Line</subway>
</transportation>
```

```
<transportation>
  <T>Red Line</T>
</transportation>
```



Note: there is no error in declaring T to be substitutable with subway. The error occurs only when you try to do substitution in the instance.

Element Substitution

- Rules
 - The elements that are declared to be in the substitution group must be declared as **global elements**.
 - The type of every element in the substitutionGroup must be the **same as, or derived from**, the type of the head element.
 - If the type of a substitutionGroup element is the same as the head element then you can **omit** it.
 - **Transitive**: If element A can substitute for element B, and element B can substitute for element C, then element A can substitute for element C: If $A \rightarrow B \rightarrow C$ then $A \rightarrow C$
 - **Non-symmetric**: If element A can substitute for element B, it is not the case that element B can substitute for element A.

Type Substitution

- A base type can be substituted by any derived type.
 - Example:
 - Suppose that BookType is derived from PublicationType.
 - We declare an element, Publication, to be of type PublicationType.
 - In the instance document Publication's content can be either of PublicationType or BookType.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              targetNamespace="http://www.books.org"
              xmlns="http://www.books.org"
              elementFormDefault="unqualified">
    <xsd:complexType name="PublicationType">
        <xsd:sequence>
            <xsd:element name="Title" type="xsd:string"/>
            <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
            <xsd:element name="Date" type="xsd:year"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="BookType">
        <xsd:complexContent>
            <xsd:extension base="PublicationType">
                <xsd:sequence>
                    <xsd:element name="ISBN" type="xsd:string"/>
                    <xsd:element name="Publisher" type="xsd:string"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="BookStore">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Publication" maxOccurs="unbounded" type="PublicationType" />
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

PublicationType
is the base type

BookType extends
PublicationType

Publication is of type
PublicationType
(the base type)

```
<?xml version="1.0"?>
<bk:BookStore      xmlns:bk="http://www.books.org"
                      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xsi:schemaLocation="http://www.books.org BookStore.xsd">
    <Publication>
        <Title>Staying Young Forever</Title>
        <Author>Karin Granstrom Jordan, M.D.</Author>
        <Date>1999</Date>
    </Publication>
    <Publication xsi:type="bk:BookType">
        <Title>Illusions The Adventures of a Reluctant Messiah</Title>
        <Author>Richard Bach</Author>
        <Date>1977</Date>
        <ISBN>0-440-34319-4</ISBN>
        <Publisher>Dell Publishing Co.</Publisher>
    </Publication>
    <Publication xsi:type="bk:BookType">
        <Title>The First and Last Freedom</Title>
        <Author>J. Krishnamurti</Author>
        <Date>1954</Date>
        <ISBN>0-06-064831-7</ISBN>
        <Publisher>Harper & Row</Publisher>
    </Publication>
</bk:BookStore>
```

Type Substitution

- The *type* attribute
 - The Publication element is declared to be of type PublicationType.
 - BookType is derived from PublicationType.
 - As the content is not the source type, but rather a derived type, we specify the derived type that is being used: attribute *type*
 - The attribute *type* comes from the XML Schema Instance (xsi) namespace.



Chapter 5.7

Abstraction

Abstract Elements

- You can declare an element to be abstract
 - An abstract element is a template/placeholder element.
 - If an element is declared abstract then in an XML instance document that element may not appear.
 - However, elements that are substitutionGroup'ed to the abstract type may appear in its place.

Abstraction

- Abstract elements

```
<xsd:element name="publication" abstract="true"  
    type="pubType"/>
```

```
<xsd:element name="book" substitutionGroup="publicati  
    type="pubType"/>
```

```
<xsd:element name="magazine" substitutionGroup="publicati  
    type="pubType"/>
```

implies: "publication" not allowed in instance doc

- Same mechanisms applicable to

"book" and "magazine" may substitute for "publication"

```

<xsd:complexType name="PublicationType">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:gYear"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="BookType">
  <xsd:complexContent>
    <xsd:extension base="PublicationType" >
      <xsd:sequence>
        <xsd:element name="ISBN" type="xsd:string"/>
        <xsd:element name="Publisher" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

```

<xsd:complexType name="MagazineType">
  <xsd:complexContent>
    <xsd:restriction base="PublicationType">
      <xsd:sequence>
        <xsd:element name="Title" type="xsd:string"/>
        <xsd:element name="Author" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="Date" type="xsd:gYear"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

```

```

<xsd:element name="Publication" abstract="true" type="PublicationType"/>
<xsd:element name="Book" substitutionGroup="Publication" type="BookType"/>
<xsd:element name="Magazine" substitutionGroup="Publication" type="MagazineType"/>

```

```

<xsd:element name="BookStore">
  <xsd:complexType> <xsd:sequence>
    <xsd:element ref="Publication" maxOccurs="unbounded"/>
  </xsd:sequence> </xsd:complexType>
</xsd:element>

```

The Publication element is abstract, only substitutionGroup'ed elements can appear in instance doc.

The Book and Magazine elements are substitutionGroup'ed to the Publication element.

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.books.org BookStore.xsd">
    <Magazine>
        <Title>Natural Health</Title>
        <Date>1999</Date>
    </Magazine>
    <Book>
        <Title>Illusions The Adventures of a Reluctant Messiah</Title>
        <Author>Richard Bach</Author>
        <Date>1977</Date>
        <ISBN>0-440-34319-4</ISBN>
        <Publisher>Dell Publishing Co.</Publisher>
    </Book>
    <Book>
        <Title>The First and Last Freedom</Title>
        <Author>J. Krishnamurti</Author>
        <Date>1954</Date>
        <ISBN>0-06-064831-7</ISBN>
        <Publisher>Harper & Row</Publisher>
    </Book>
</BookStore>
```

Abstract Types

- You can declare a complexType to be abstract
 - An abstract complexType is a template/placeholder type.
 - If an element is declared to be a type that is abstract then in an XML instance document the content model of that element may not be that of the abstract type.
 - However, complexType's that are derived from the abstract type may substitute for the abstract type.

```

<xsd:complexType name="PublicationType" abstract="true"> ←
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Date" type="xsd:year"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="BookType"> ←
  <xsd:complexContent>
    <xsd:extension base="PublicationType">
      <xsd:sequence>
        <xsd:element name="ISBN" type="xsd:string"/>
        <xsd:element name="Publisher" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="SingleAuthorPublication">
  <xsd:complexContent>
    <xsd:restriction base="PublicationType">
      <xsd:sequence>
        <xsd:element name="Title" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="Author" type="xsd:string"/>
        <xsd:element name="Date" type="xsd:year"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="BookStore">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Book" maxOccurs="unbounded" type="PublicationType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

Note that PublicationType is declared abstract.

Book derives from PublicationType. By default abstract="false". Thus, this type can substitute for the PublicationType.

Abstract Types

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.books.org BookStore.xsd">
    <Book xsi:type="BookType">
        <Title>My Life and Times</Title>
        <Author>Paul McCartney</Author>
        <Date>1998</Date>
        <ISBN>94303-12021-43892</ISBN>
        <Publisher>McMillin Publishing</Publisher>
    </Book>
    <Book xsi:type="SingleAuthorPublication">
        <Title>FooManchu</Title>
        <Author>Don Keyote</Author>
        <Date>1951</Date>
    </Book>
</BookStore>
```

- The content model of each `<Book>` element must be from a type that derives from `PublicationType`.

Abstraction

- If you declare an element to be abstract
 - Use element substitution for the abstract element
(as provided by substitutionGroup)
- If you declare a complexType to be abstract
 - Use type substitution for the abstract type
(as provided by type derivation)

XML Processing

Tree Processing (DOM)

Lecture "XML in Communication Systems"
Chapter 6

Dr.-Ing. Jesper Zedlitz
Research Group for Communication Systems
Dept. of Computer Science
Christian-Albrechts-University in Kiel



Recommended Reading

Informatik · CAU Kiel

- Lauren Wood, Arnaud Le Hors, Vidur Apparao et al.:
Document Object Model (DOM) Level 1 Specification (Second Edition)
Version 1.0, W3C Working Draft 29 September, 2000
<http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/>

Overview

Informatik · CAU Kiel

1. Introduction
2. Document Object Model (DOM)

The background of the image is a vast, dark blue-grey sky filled with scattered, wispy clouds. The horizon line is visible at the bottom, showing a mix of dark blues and hints of orange and yellow from the setting or rising sun. The overall atmosphere is serene and slightly somber.

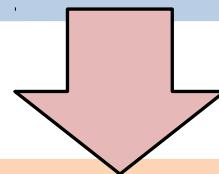
Chapter 6.1

Introduction

Introduction

Informatik · CAU Kiel

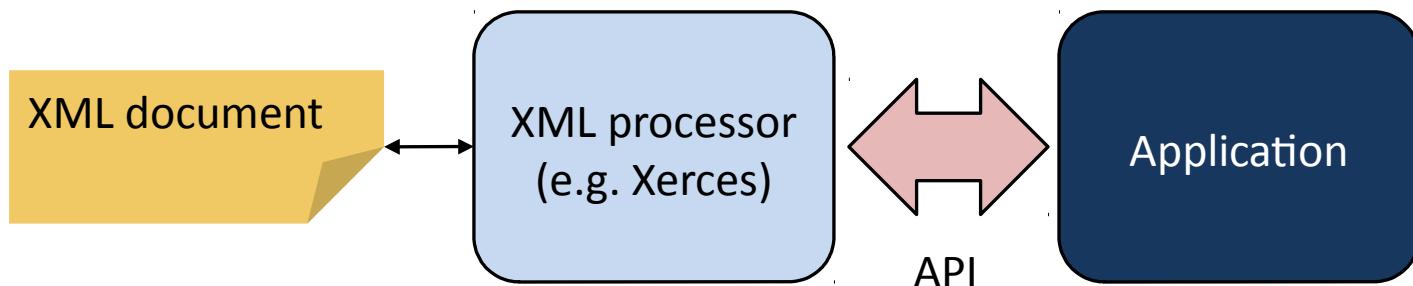
How to give program code
access to
XML documents?



Application Programming
Interfaces (APIs)

Introduction

- XML processing

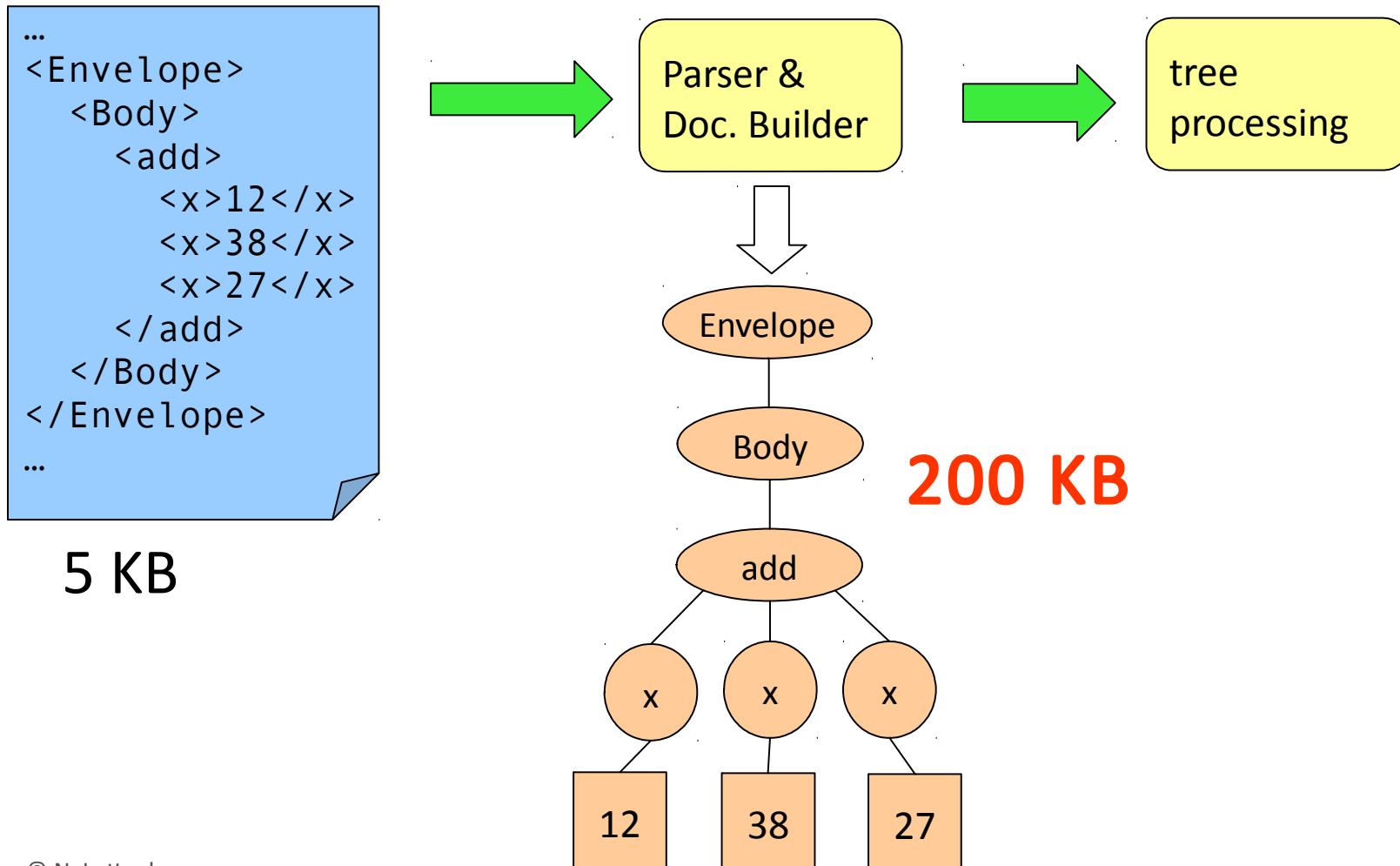


Introduction

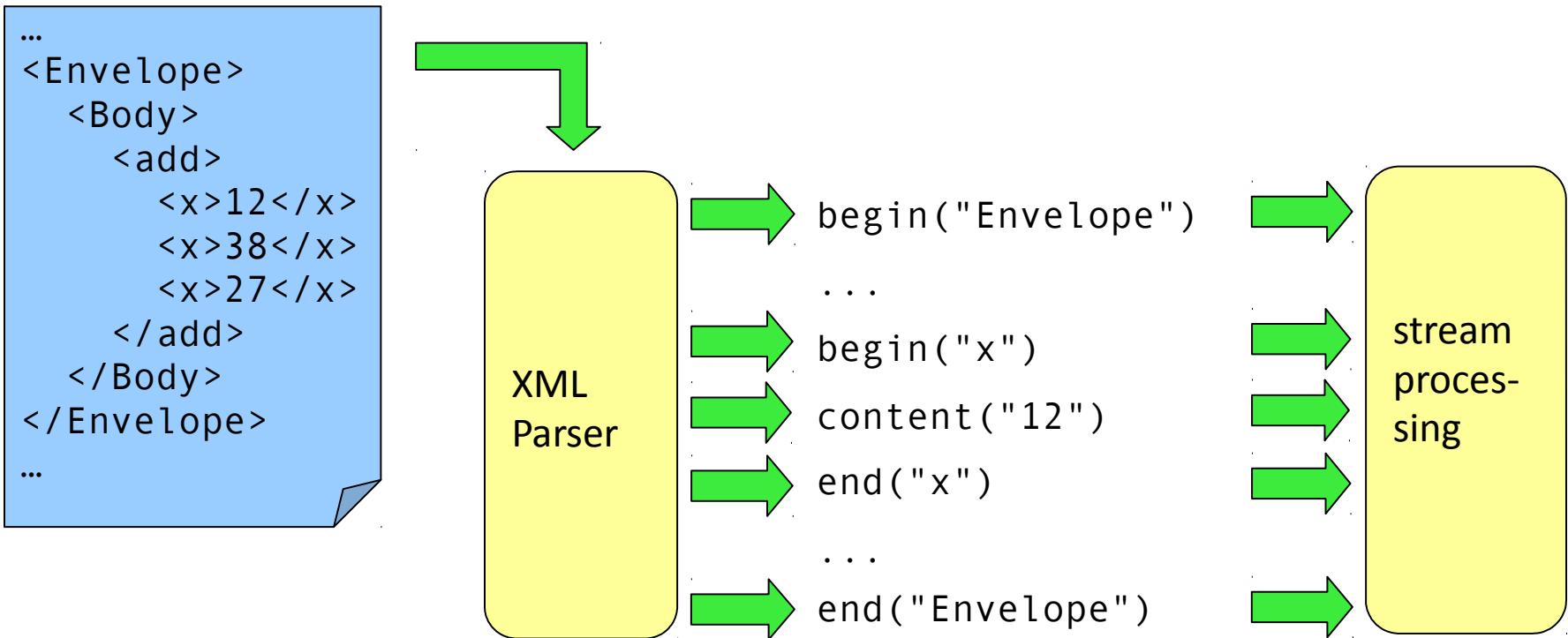
- XML processors provide the structure and contents of XML documents to applications through APIs
 - Tree-based APIs
 - provide full parse tree to application
 - e.g., DOM W3C Recommendation
(Document Object Model)
 - Stream-based APIs
 - notify application through parsing events
 - e.g., the SAX callback interfaces
(Simple API for XML)

Tree-based APIs

Informatik · CAU Kiel

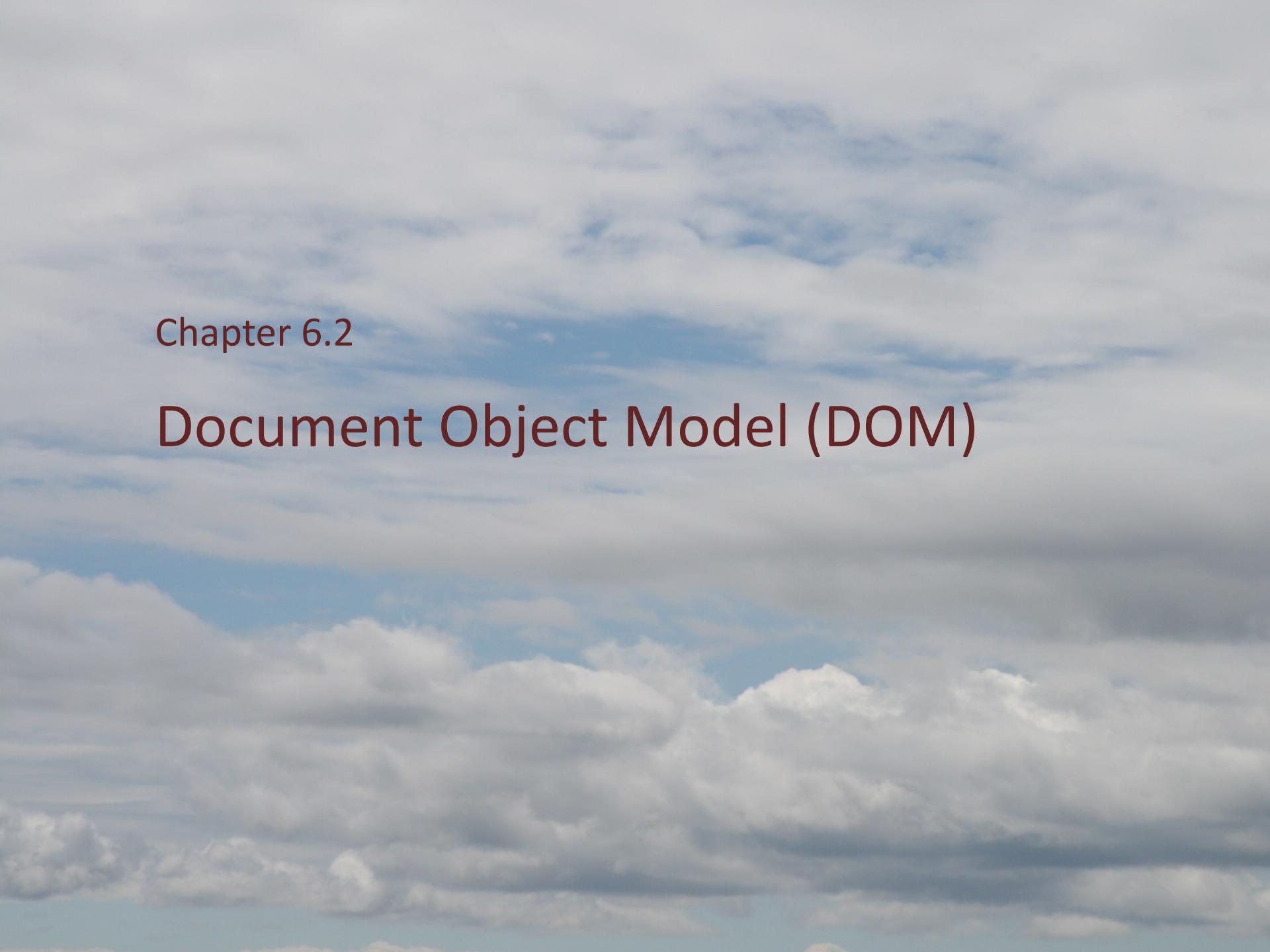


Stream-based APIs



Tree vs. Stream

- Pro and cons
 - tree processing
 - convenient tree navigation
 - straightforward impl. of modification operations
 - but
 - consumes 10 to 100 times more memory than XML text
 - tree building must be completed before application processing starts
 - stream processing
 - low memory consumption
 - XML processing starts with first parsing event
 - but
 - no internal document representation for navigation and processing
 - low overhead for validation of invalid documents



Chapter 6.2

Document Object Model (DOM)

Document Object Model (DOM)

- DOM—What is it?
 - With the DOM (Document Object Model), the W3C has defined a language- and platform-neutral view of XML documents much like the XML Information Set.
 - DOM APIs exist for a wide variety of—predominantly object-oriented—programming languages (Java, C++, C, Perl, Python, ...).

Document Object Model (DOM)

- DOM—What is it?
 - DOM allows *programs and scripts*
 - to build documents,
 - to navigate their structure,
 - to add, modify or delete elements and content
 - for implementations of querying, filtering, transformation, rendering etc. applications
 - One could read DOM as "Directly Obtainable in Memory"

Document Object Model (DOM)

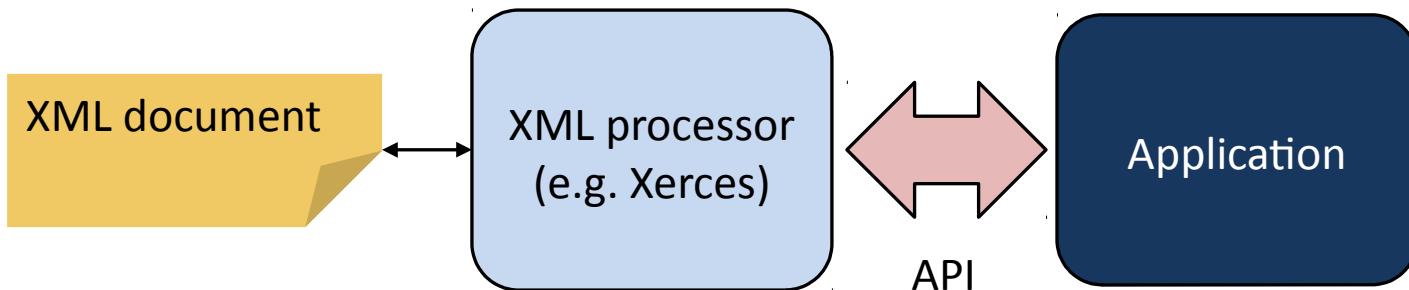
- From the Mozilla developer site

"The Document Object Model (DOM) is a **programming interface** for HTML and XML documents. It provides a structured representation of the document and it defines a way that the **structure can be accessed from programs** so that they can change the document structure, style and content. The DOM provides a representation of the document as a **structured group of nodes and objects** that have properties and methods. Essentially, it connects web pages to scripts or programming languages."

Document Object Model (DOM)

Informatik · CAU Kiel

- Two major DOM concepts



1. An XML Processor offering a DOM interface parses the XML input document, and constructs the complete XML document tree in memory.
2. The XML application then issues DOM library calls to explore and manipulate the XML document, or generate new XML documents.

Document Object Model (DOM)

- Example: JavaScript
 - All of the properties, methods, and events available for manipulating and creating web pages are organized into objects, e.g.,
 - the document object represents the document itself,
 - the table object implements the special HTMLTableElement DOM interface for accessing HTML tables, and so forth

Document Object Model (DOM)

- Example: JavaScript

```
<html>
  <head>
    <script>
      // run this function when the document is loaded
      window.onload = function() {
        // create a couple of elements
        // in an otherwise empty HTML page
        heading = document.createElement("h1");
        heading_text = document.createTextNode("Big Head!");
        heading.appendChild(heading_text);
        document.body.appendChild(heading);
      }
    </script>
  </head>
  <body></body>
</html>
```

Document Object Model (DOM)

- DOM structure model based on OO concepts
 - *methods* (to access or change object's state)
 - *interfaces* (declaration of a set of methods)
 - *objects* (encapsulation of data and methods)
- Roughly similar to the XPath/XSLT data model
(to be discussed later)
 - DOM "product" \approx a parse tree
- Language-independence
 - DOM interfaces defined in Interface Definition Language (IDL) from Corba Specification (OMG)

Document Object Model (DOM)

- W3C DOM Specification
 - second in the "XML family" of recommendations
 - Level 1, W3C Rec, Sept. 2000 (2nd ed.)
 - Level 2, W3C Rec, Nov. 2000
 - Level 3, W3C Rec, April 2004

Document Object Model (DOM)

Informatik · CAU Kiel

- DOM Level 1
 - Basic representation and manipulation of document structure and content
 - DOM Core Interfaces
 - Fundamental interfaces: basic interfaces to structured documents
 - Extended interfaces (XML-specific): CDATASection, DocumentType, Notation, Entity, EntityReference, ProcessingInstruction
 - DOM HTML Interfaces
 - more convenient access to HTML documents
 - No access to contents of DTD

Document Object Model (DOM)

- DOM Level 2 adds
 - support for namespaces
 - accessing elements by ID attribute values
 - optional features
 - interfaces to document views and style sheets
 - an event model (for user actions on elements)
 - methods for traversing the document tree and manipulating regions of document (e.g., selected by the user or an editor)

Document Object Model (DOM)

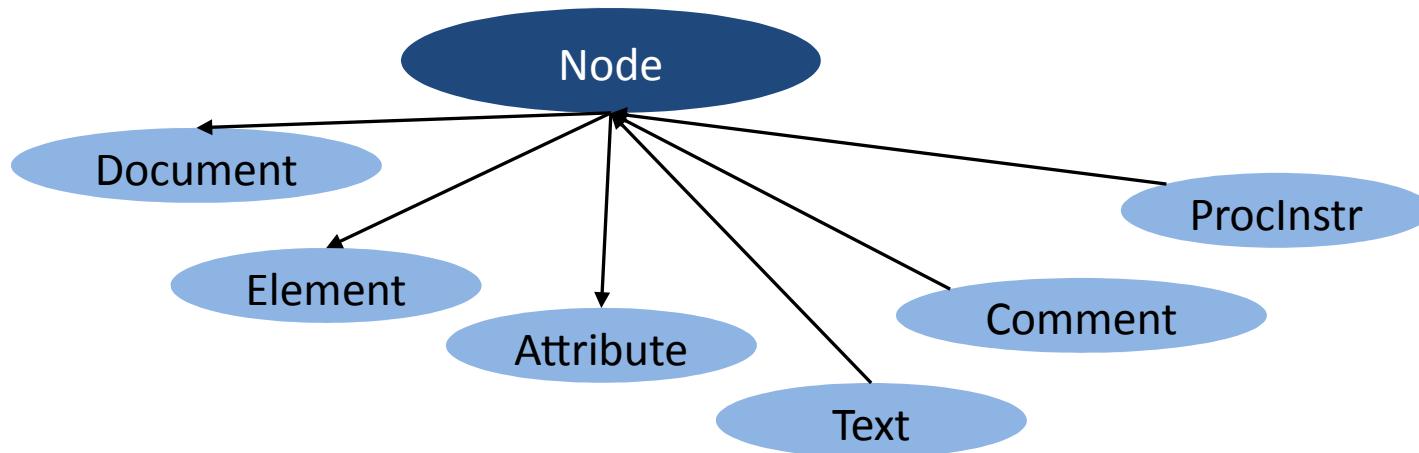
Informatik · CAU Kiel

- DOM Level 3 adds
 - Loading and writing of docs

Document Object Model (DOM)

Informatik · CAU Kiel

- Excerpt from the inheritance hierarchy of node objects



- NodeList**
 - interface to handle ordered lists of Nodes, such as the elements returned by the `Element.getElementsByTagName` method
- NamedNodeMap**
 - interface to handle unordered sets of nodes referenced by their name attribute, such as the attributes of an Element.

Document Object Model (DOM)

- DOM establishes two basic types of operations
 1. Navigation:
the ability to traverse the node hierarchy, and
 2. Reference:
the ability to access a collection of nodes by name.

Document Object Model (DOM)

- Navigation
 - The structure of the document determines the inheritance of element attributes. Thus, it is important to be able to navigate among the node objects representing parent and child elements.
 - Given a node, you can find out where it is located in the document structure model and you can refer to the parent, child as well as siblings of this node.
 - This might be done using the **NodeList** object, which represents an ordered collection of nodes.

Document Object Model (DOM)

- Reference
 - Example: a gallery page is filled with individual images. A unique name or ID can be assigned to each image using the NAME or ID attribute.
 - It is possible to create an index of image names or ID's by iterating over a list of nodes.
 - To reference an image the **NamedNodeMap** object can be used, which represents an (unordered) collection of nodes that can be accessed by name.

Document Object Model (DOM)

- Node methods

- getNodeType
- hasAttributes
- getParentNode
- hasChildNodes
- getLastChild
- insertBefore(newChild, refChild)
- replaceChild(newChild, oldChild)
- getPreviousSibling
- getOwnerDocument
- getNodeValue
- getAttributes

- getChildNodes
- getFirstChild

Document Object Model (DOM)

- Document methods

`getDocumentElement`

`createAttribute(name)`

`createElement(tagName)`

`createTextNode(data)`

`getDocType()`

`getElementById(IdVal)`

Document Object Model (DOM)

- Element methods

`getTagName`

`getAttributeNode(name)`

`setAttributeNode(attr)`

`removeAttribute(name)`

`getElementsByTagName(name)`

`hasAttribute(name)`

Document Object Model (DOM)

- Object creation in DOM
 - Each DOM object X lives in the context of a Document:
`X.getOwnerDocument()`
 - Objects implementing interface X are created by factory methods
`D.createX(...)`, where D is a Document object.
 - Examples:

`createElement("A")`, `createAttribute("href")`,
`createTextNode("Hello!")`
 - Creation and persistent saving of Documents left to be specified by implementations.

DOM: Implementations

- Java-based parsers
 - e.g. IBM XML4J, Apache Xerces, Apache Crimson
- MS IE5 browser
 - COM programming interfaces for C/C++ and MS Visual Basic,
 - ActiveX object programming interfaces for script languages
- XML::DOM (Perl implementation of DOM Level 1)

Document Object Model (DOM)

- Concluding remarks
 - Inherent memory hunger of the DOM may lead
 - to heavy swapping activity or even "out-of-memory" failures.
 - To remedy:
 - Try to preprocess the input XML document to reduce its overall size.
 - Use an XPath/XSLT processor to preselect interesting document regions,
 - Or: Use a completely different approach to XML processing (→ SAX).

XML Processing

Stream Processing (SAX)

Lecture "XML in Communication Systems"
Chapter 7

Dr.-Ing. Jesper Zedlitz
Research Group for Communication Systems
Dept. of Computer Science
Christian-Albrechts-University in Kiel



Recommended Reading

Informatik · CAU Kiel

- <http://www.saxproject.org/>

Parsers

- What is a parser?
 - A program that analyses the grammatical structure of an input, with respect to a given formal grammar
 - The parser determines how a sentence can be constructed from the grammar of the language by describing the atomic elements of the input and the relationship among them.

XML Parsing Standards

- Two parsing methods for accessing XML
 - DOM (Document Object Model)
 - convert XML into a tree of objects
 - "random access" method
 - can update XML document (insert/delete/update nodes)
 - SAX (Simple API for XML)
 - event-driven parsing
 - "serial access" method
 - read-only API

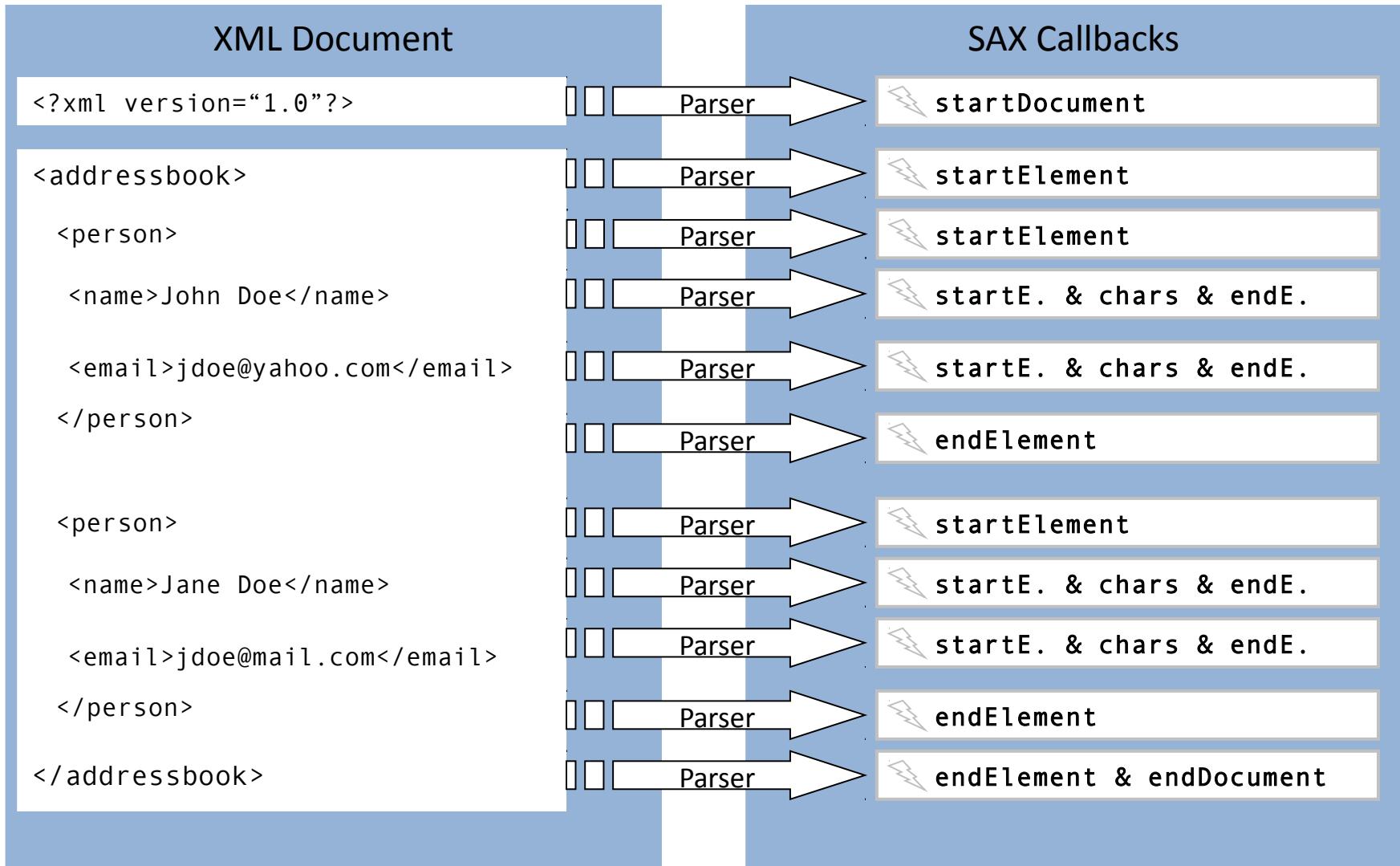
SAX vs. DOM Comparison

- SAX
 - Java-specific
 - interprets XML as a stream of events
 - You supply event-handling callbacks.
 - SAX parser invokes your event-handlers as it parses.
 - doesn't build data model in memory
 - serial access: very fast, lightweight
 - good choice when
 - no data model is needed, or
 - natural structure for data model is list, matrix, etc.
- DOM
 - W3C standard for representing structured documents
 - platform and language neutral

SAX Parser

- SAX = Simple API for XML
 - XML is read sequentially
 - When a parsing event happens, the parser invokes the corresponding method of the corresponding handler
 - The handlers are programmer's implementation of standard Java interfaces and classes
 - Similar to an I/O-Stream, goes in one direction

How Does SAX Work?



How Does SAX Work?

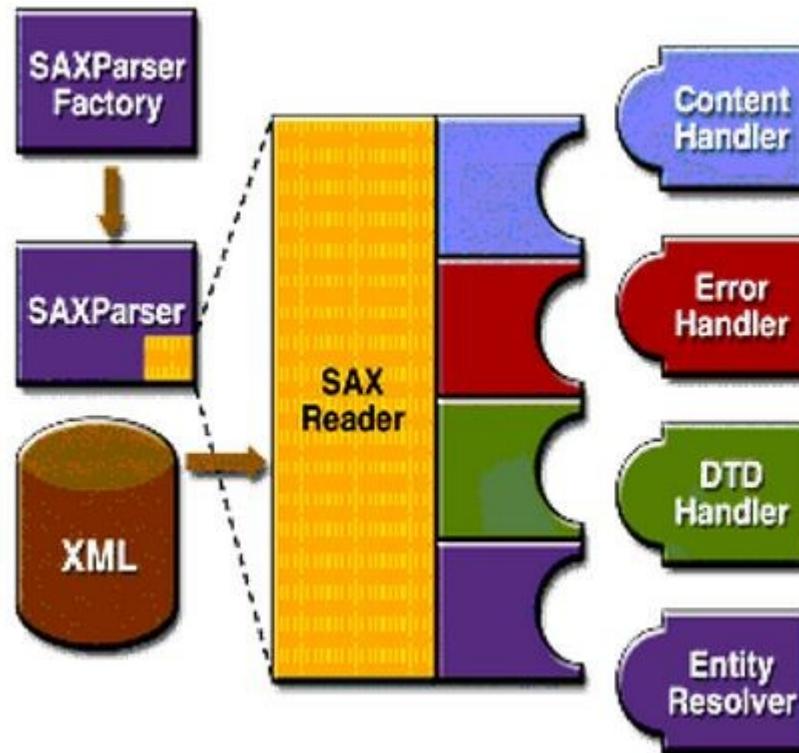
Informatik · CAU Kiel

```
<?xml version="1.0" encoding="UTF-8"?>  
  
<dots>  
    this is before the first dot  
    and it continues on multiple lines  
    <dot x="9" y="81" />  
    <dot x="11" y="121" />  
    <flip>  
        flip is on  
        <dot x="196" y="14" />  
        <dot x="169" y="13" />  
    </flip>  
    flip is off  
    <dot x="12" y="144" />  
    <extra>stuff</extra>  
    <!-- a final comment -->  
</dots>
```

```
startDocument  
startElement: dots (0 attributes)  
characters: this is before the first dot  
            and it continues on multiple lines  
startElement: dot (2 attributes)  
endElement: dot  
startElement: dot (2 attributes)  
endElement: dot  
startElement: flip (0 attributes)  
characters: flip is on  
startElement: dot (2 attributes)  
endElement: dot  
startElement: dot (2 attributes)  
endElement: dot  
endElement: flip  
characters: flip is off  
startElement: dot (2 attributes)  
endElement: dot  
startElement: extra (0 attributes)  
characters: stuff  
endElement: extra  
endElement: dots  
endDocument
```

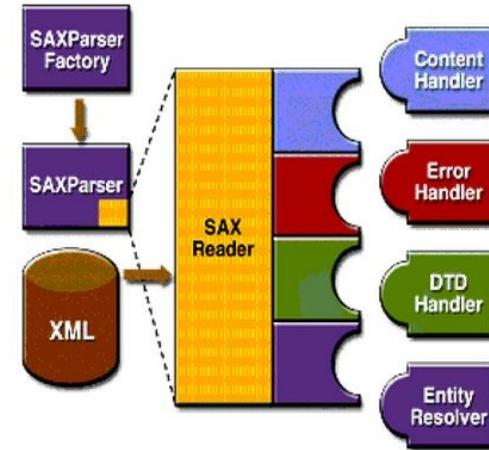
Finished parsing input. Got the following dots:
[(9, 81), (11, 121), (14, 196), (13, 169), (12, 144)]

SAX Structure



SAX Structure

- SAX structure
 - SAXParserFactory
A SAXParserFactory object creates an instance of the parser determined by the system property, `javax.xml.parsers.SAXParserFactory`.
 - SAXParser
The SAXParser interface defines several kinds of parse() methods. In general, you pass an XML data source and a DefaultHandler object to the parser, which processes the XML and invokes the appropriate methods in the handler object.

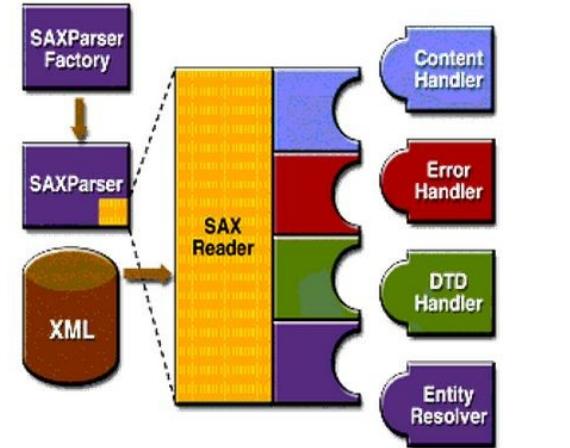


SAX Structure

- SAX structure
 - SAXReader

The SAXParser wraps a SAXReader. Typically, you don't care about that, but every once in a while you need to get hold of it using SAXParser's `getXMLReader()` so that you can configure it. It is the SAXReader that carries on the conversation with the SAX event handlers you define.
 - DefaultHandler

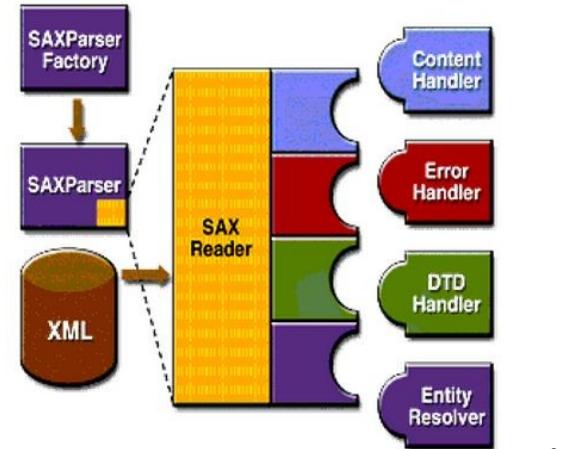
Not shown in the diagram, a DefaultHandler implements the ContentHandler, ErrorHandler, DTDHandler, and EntityResolver interfaces (with null methods), so you can override only the ones you're interested in.



SAX Structure

- SAX structure
 - ContentHandler

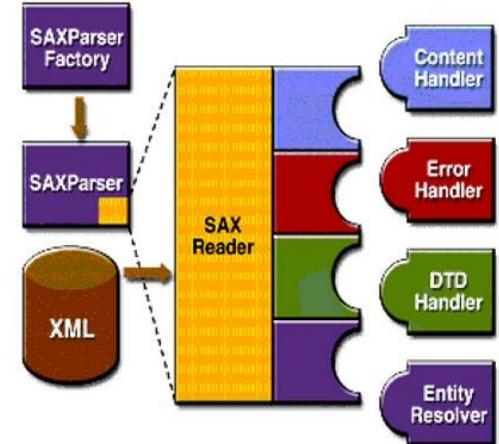
Methods such as startDocument, endDocument, startElement, and endElement are invoked when an XML tag is recognized. This interface also defines the methods characters and processingInstruction, which are invoked when the parser encounters the text in an XML element or an inline processing instruction, respectively.



SAX Structure

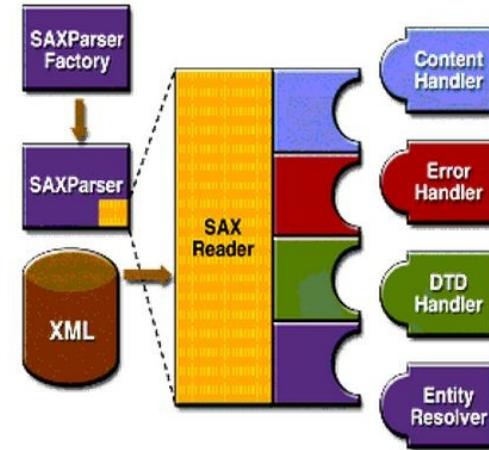
- SAX structure
 - ErrorHandler

Methods `error`, `fatalError`, and `warning` are invoked in response to various parsing errors. The default error handler throws an exception for fatal errors and ignores other errors (including validation errors). That's one reason you need to know something about the SAX parser, even if you are using the DOM. Sometimes, the application may be able to recover from a validation error. Other times, it may need to generate an exception. To ensure the correct handling, you'll need to supply your own error handler to the parser.



SAX Structure

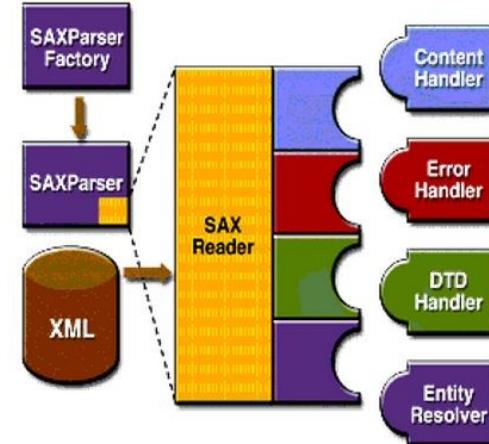
- SAX structure
 - DTDHandler
Defines methods you will generally never be called upon to use. Used when processing a DTD to recognize and act on declarations for an unparsed entity.



SAX Structure

- SAX structure
 - EntityResolver

The resolveEntity method is invoked when the parser must identify data identified by a URI. In most cases, a URI is simply a URL, which specifies the location of a document, but in some cases the document may be identified by a URN—a public identifier, or name, that is unique in the web space. The public identifier may be specified in addition to the URL. The EntityResolver can then use the public identifier instead of the URL to find the document—for example, to access a local copy of the document if one exists.



SAX Programming

- In some cases, programming with SAX is difficult
 - How can we find, using a SAX parser, elements e1 with ancestor e2?
 - How can we find, using a SAX parser, elements e1 that have a descendant element e2?
 - How can we find the element e1 referenced by the IDREF attribute of e2?
 - SAX parsers do not provide access to elements other than the one currently visited in the serial (DFS) traversal of the document.
 - They do not read backwards
 - They do not enable access to elements by ID or name

XML Document Navigation, Transformation

XPath—Navigating in XML Documents

Lecture "XML in Communication Systems"

Chapter 8

Dr.-Ing. Jesper Zedlitz

Research Group for Communication Systems

Dept. of Computer Science

Christian-Albrechts-University in Kiel



Acknowledgement

Informatik · CAU Kiel

- Part of the material in this chapter is based on the
XML Course of Dr. Torsten Grust,
Dept. of Computer and Information Science,
University of Konstanz

Recommended Reading

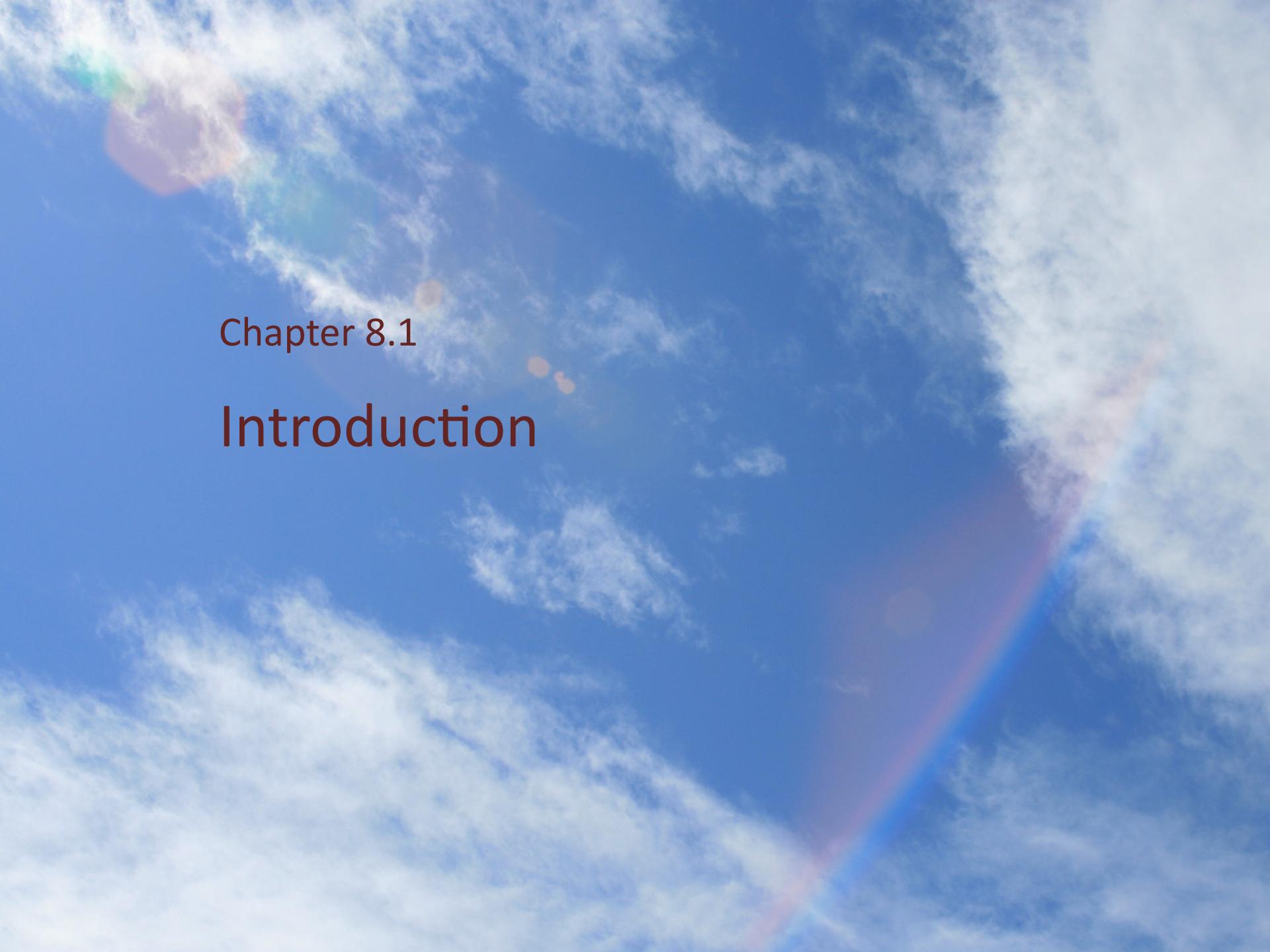
Informatik · CAU Kiel

- Anders Berglund et al. (Ed.):
XML Path Language (XPath) 2.0
W3C Recommendation 23 January 2007
<http://www.w3.org/TR/xpath20/>

Overview

Informatik · CAU Kiel

1. Introduction
2. XML document model
3. Path expressions
4. Abbreviated syntax



Chapter 8.1

Introduction

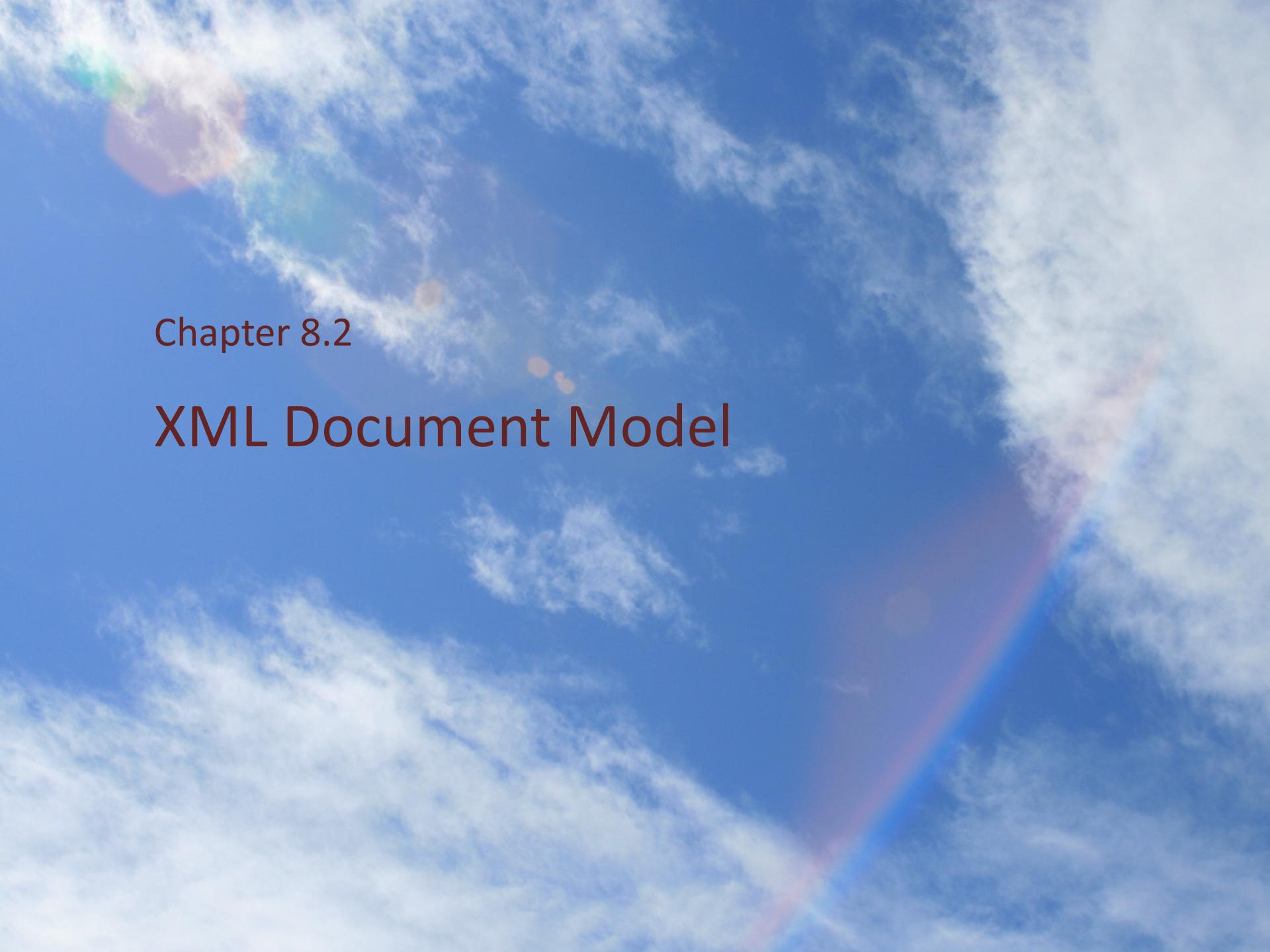
Introduction

- What is XPath?
 - A declarative, expression-based language to locate and test document nodes.
 - Addressing document nodes—a core task in the XML world.
 - XPath occurs as an embedded sub-language in
 - [XSLT](#): extract and transform XML document [fragments] into XML, XHTML, PDF, ...
 - [XQuery](#): compute with XML document nodes and contents, compute new docs, ...
 - [XPointer](#): representation of the address of one or more doc nodes in a given XML document

Introduction

"XPath operates on the abstract, logical structure of an XML document, rather than its surface syntax. XPath uses a compact, non-XML syntax to facilitate use of XPath within URIs and XML attribute values."

from the W3C TR on XPath 2.0



Chapter 8.2

XML Document Model

Information Items

- Node types

r = root node

C = set of comment nodes

P = set of processing-instruction (PI) nodes

E = set of element nodes (among them the document element)

A = set of attribute nodes

N = set of namespace nodes

T = set of text nodes

Information Items

- Basic grammar
 - The root node has no parent.
 - The root node has exactly one element node child; it is called the **document element**.
 - The node sets {r}, C, P, E, A, N, T are pairwise disjunct.
 - Only the root node and element nodes have children.
 - Only element nodes "have" attribute and/or namespace nodes.

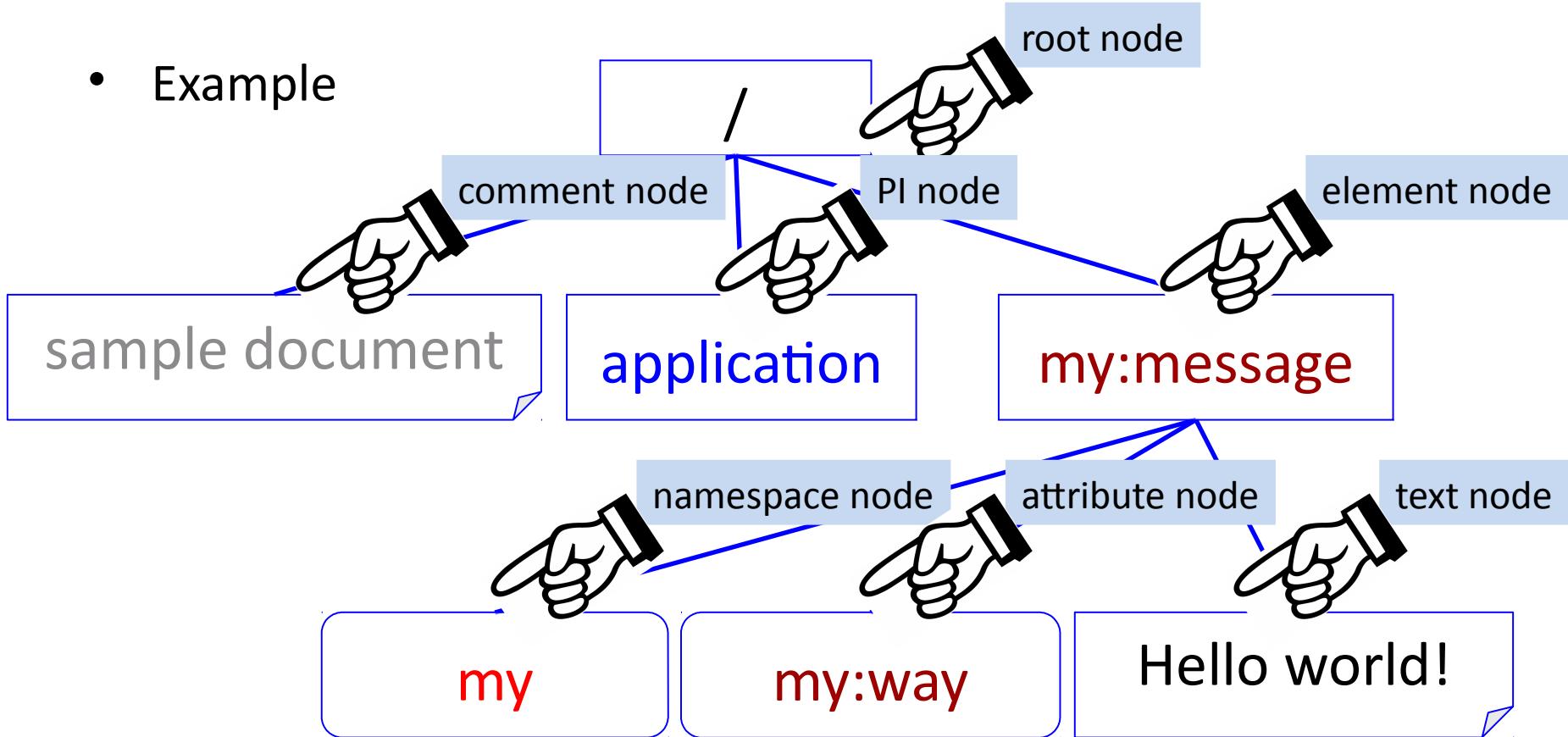
Information Items

- Example

```
<?xml version="1.0" ?>
<!-- sample document -->
<?application instruction="Read it!" ?>
<my:message xmlns:my="urn:comsys.uni-kiel.de:nl"
              my:way="Example">
    Hello world!
</my:message>
```

Information Items

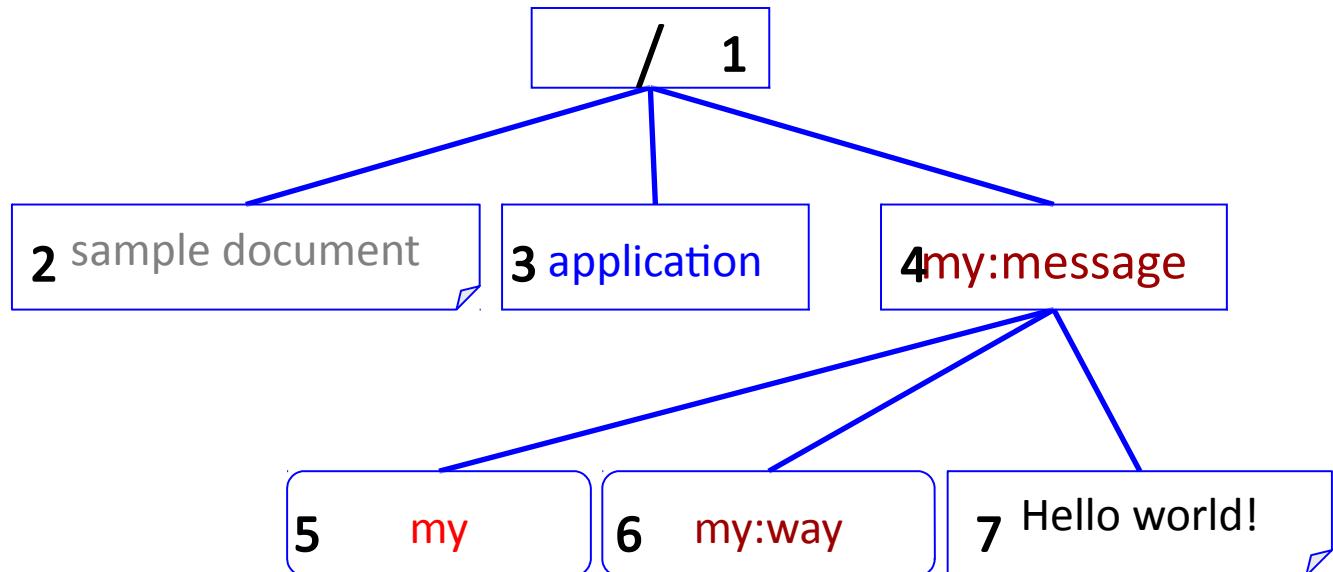
- Example



Information Items

- Element order

o: $\{r\} \cup C \cup P \cup E \cup A \cup N \cup T \rightarrow \{1, 2, 3, 4, \dots\}$



Nodes are ordered according to
sequence of appearance in XML document

Information Items

- Sample properties

Node type	string-value	expanded name	local name	namespace URI
root	concatenation of the string-values of all text node descendants of the root node in doc order.	–	–	–
element	concatenation of the string-values of all text node descendants of the element node in doc order.	prefix:tag	tag	namespace URI
attribute	normalized string-value of the attribute	prefix:attr._name	attr._name	namespace URI
text	the character data of the text node	–	–	–
comment	content of the comment not including the opening <!-- or the closing -->	–	–	–
processing instruction	that part of the processing instruction that follows the target and any whitespace	target	–	–
namespace	namespace URI that is being bound to the namespace prefix	prefix	–	–

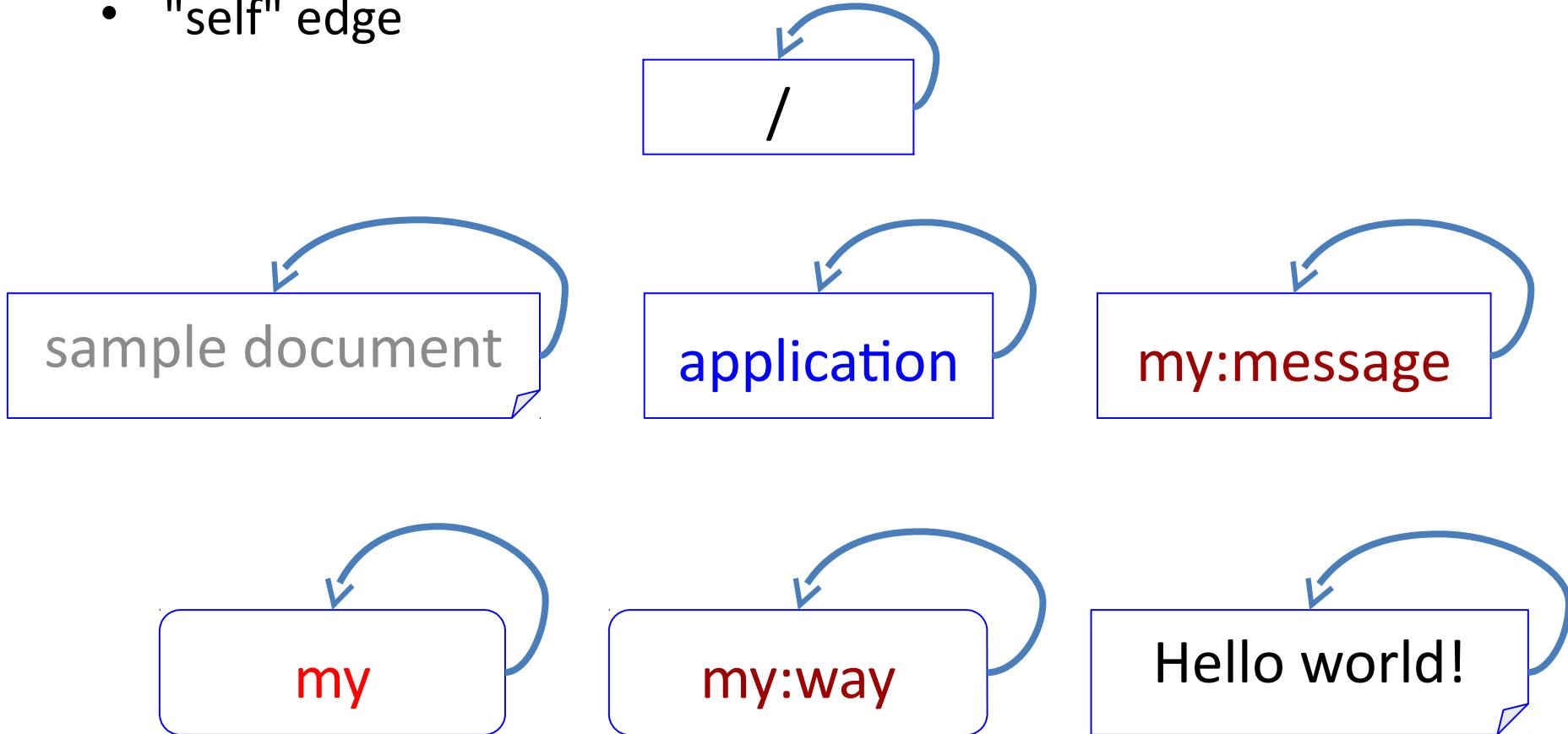
Information Items

- From example document

Node type	string-value	expanded name	local name	namespace URI
root	Hello world!	—	—	—
element	Hello world!	my:message	message	urn:comsys.uni-kiel.de:nl
attribute	Example	my:way	way	urn:comsys.uni-kiel.de:nl
text	Hello world!	—	—	—
comment	sample document	—	—	—
processing instruction	Instruction="Read it!"	application	—	—
namespace	urn:comsys.uni-kiel.de:nl	my	—	—

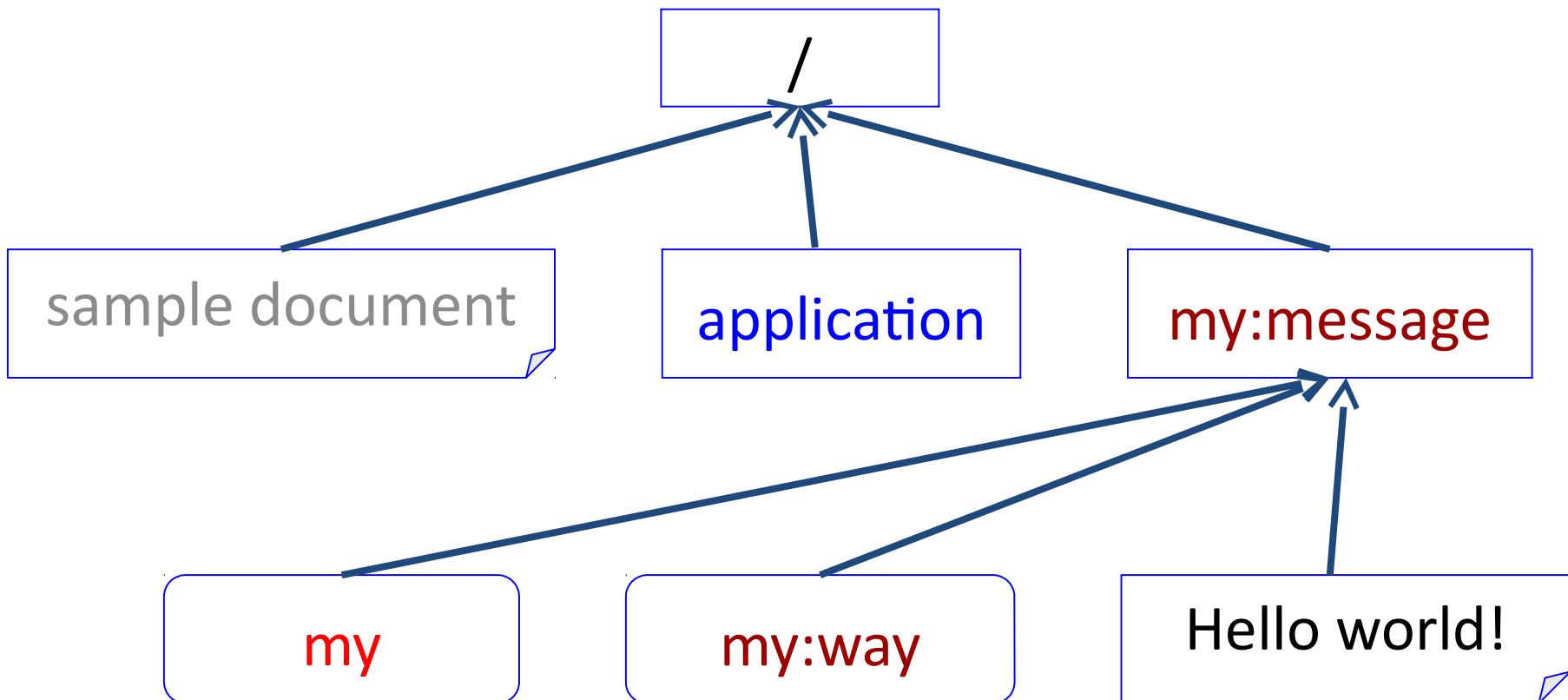
Edge Types

- "self" edge



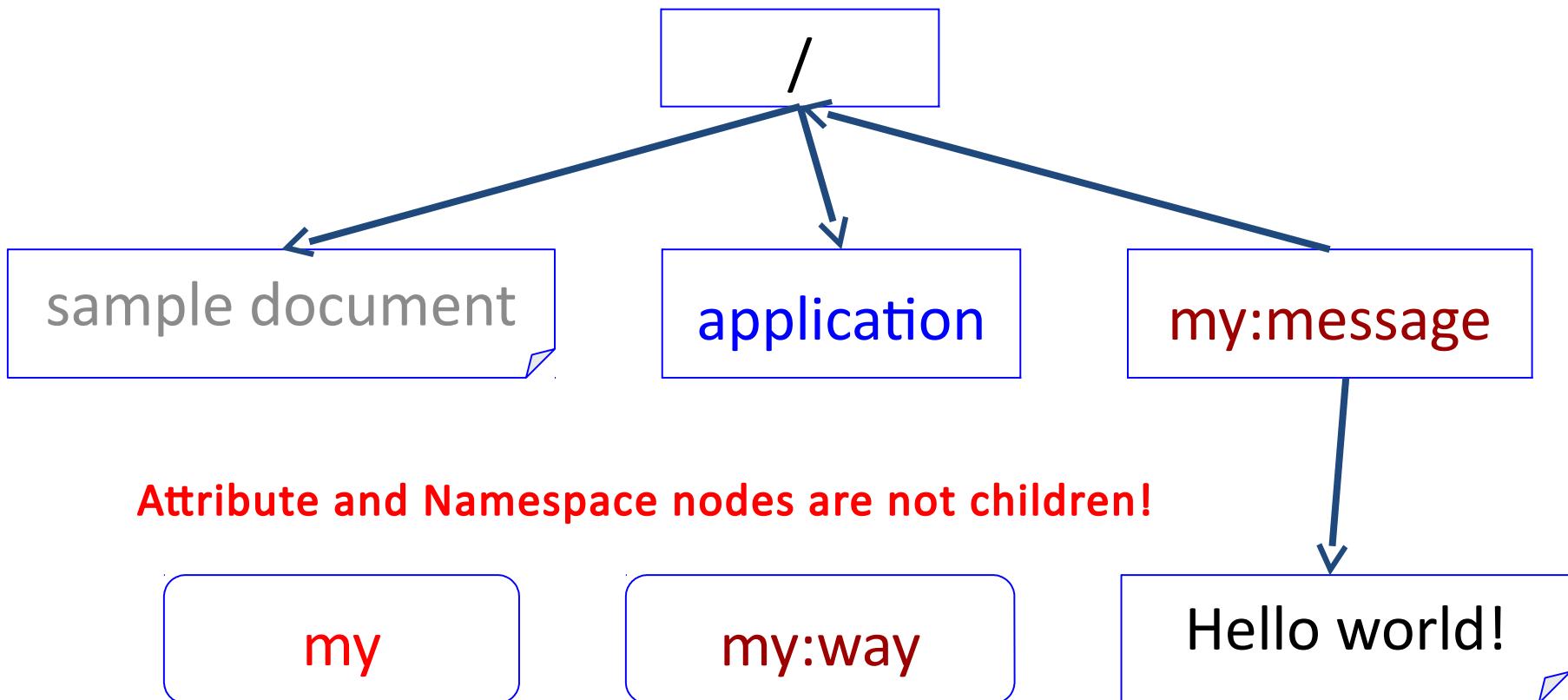
Edge Types

- "parent" edge



Edge Types

- "child" edge



Edge Types

- "attribute" edge

/

sample document

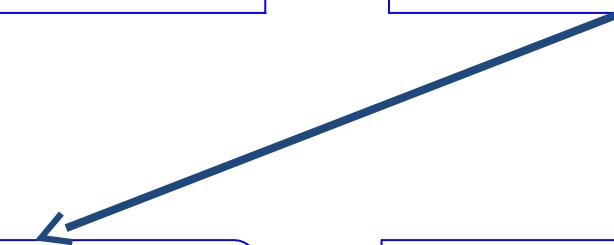
application

my:message

my

my:way

Hello world!



Edge Types

- "namespace" edge

/

sample document

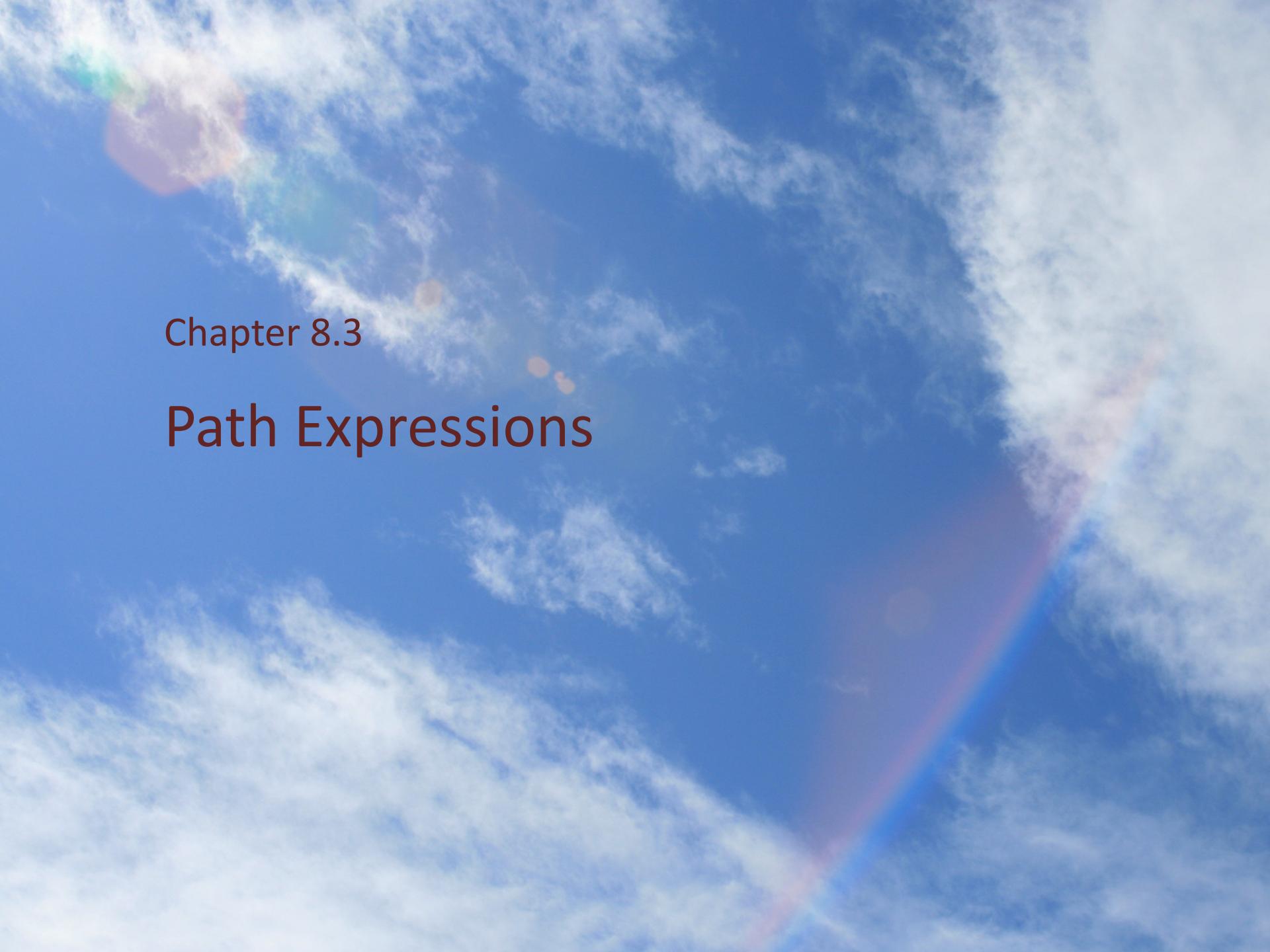
application

my:message

my

my:way

Hello world!



Chapter 8.3

Path Expressions

Path Expressions

- The path expression (or path) is XPath's core construct.

- A path features one or more steps s_i (evaluated left to right), syntactically separated by /:

$$s_0 / s_1 / \dots / s_n$$

- Each step acts like an operator of type $(\text{Node}) \rightarrow (\text{Node})$
 - Given the **context node** c and **step** s , the sequence of nodes reached by this step is computed.
 - Result is a duplicate-free node sequence in doc order.

Path Expressions

- The locstep ($c; s$) primitive function
 - Each step s is built from three components
$$a::\nu \text{ [predicate]}$$
 - The **axis** a determines, based on the location of context node c in the document tree, a sequence of reachable nodes:
$$\text{axis} : \text{context node } c \rightarrow \text{node set}$$
 - XPath divides axes into two classes:
 - forward axes (\rightarrow) and
 - reverse axes (\leftarrow)
 - Reverse axes return their result in reverse document order.

Path Expressions

- Axes

self → c

child → child nodes of c

descendant → closure of child

descendant-or-self → like descendant, plus c

parent ← parent node of c

ancestor← closure of parent

ancestor-or-self ← like ancestor, plus c

following → nodes following c in doc order, but not descendants

preceding ← nodes preceding c in doc order, but not ancestors

following-sibling → like following, if same parent as c

preceding-sibling ← like preceding, if same parent as c

attribute → attributes of c

namespace → namespace nodes of c (not discussed here)

Path Expressions

- Axes

$$\text{self::}(c) = \{ c \}$$

$$\text{child::}(c) = \{ v \mid c \rightarrow v \}$$

$$\text{descendant::}(c) = \{ v \mid c \rightarrow^+ v \}$$

$$\text{descendant-or-self::}(c) = \{ c \} \cup \{ v \mid c \rightarrow^+ v \} = \{ v \mid c \rightarrow^* v \}$$

$$\text{parent::}(c) = \{ v \mid v \rightarrow c \}$$

$$\text{ancestor::}(c) = \{ v \mid v \rightarrow^+ c \}$$

$$\text{ancestor-or-self::}(c) = \{ c \} \cup \{ v \mid v \rightarrow^+ c \} = \{ v \mid v \rightarrow^* c \}$$

$$\text{following::}(c) = \{ v \mid o(v) > o(c) \} \setminus (\text{descendant::}(c) \cup A \cup N)$$

$$\text{preceding::}(c) = \{ v \mid o(v) < o(c) \} \setminus (\text{ancestor::}(c) \cup A \cup N)$$

$$\text{ sibling}(c) = \{ v \mid \exists p: (p \rightarrow c \wedge p \rightarrow v) \}$$

$$\text{following-sibling::}(c) = \text{ sibling}(c) \cap \text{ following::}(c)$$

$$\text{preceding-sibling::}(c) = \text{ sibling}(c) \cap \text{ preceding::}(c)$$

with \rightarrow^+ denoting transitive closure, \rightarrow^* reflexive-transitive closure

Path Expressions

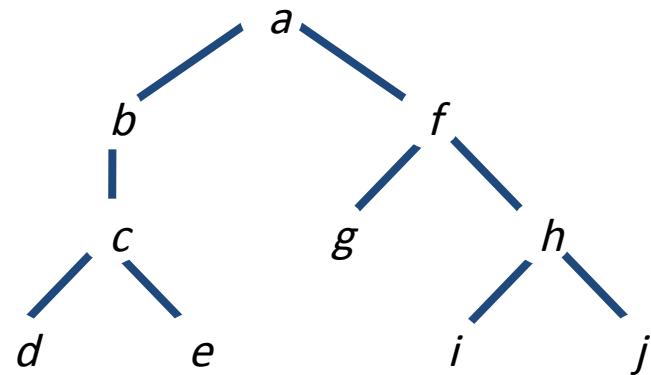
- The locstep ($c; s$) primitive function
 - The **node test** v filters this sequence to contain only specific types of nodes (e.g., only specifically named element nodes, only text/comment/PI nodes, etc.)
 - **Predicates:** (optional) expressions to further refine the set of nodes selected by the location step.
 - Typical XPath queries thus look like

$a_0::v_0/a_1::v_1/a_2::v_2/\dots a_n::v_n$

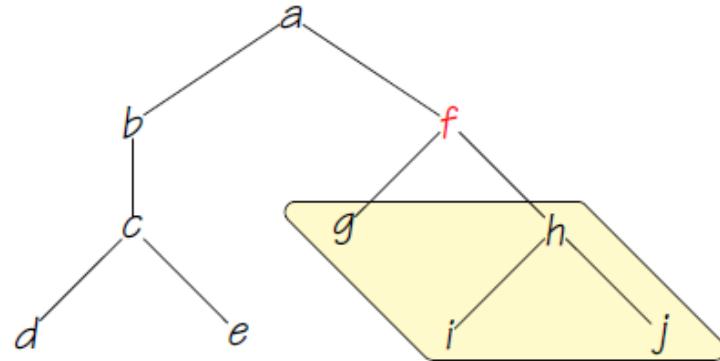
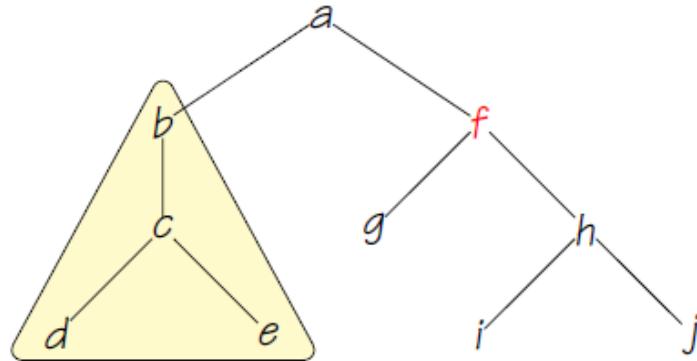
Path Expressions

- Example

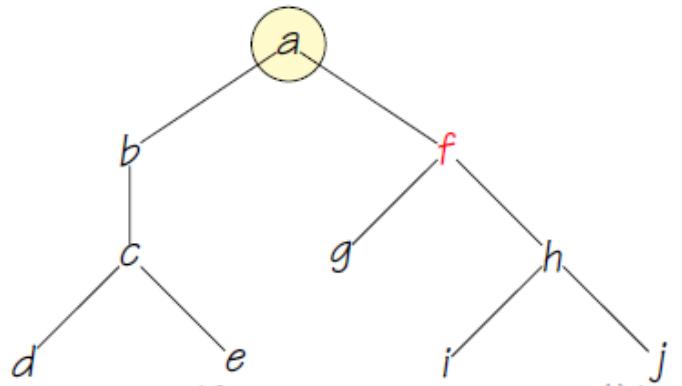
```
<a>
  <b>
    <c>
      <d/>
      <e/>
    </c>
  </b>
  <f>
    <g/>
    <h>
      <i/>
      <j/>
    </h>
  </f>
</a>
```



Path Expressions



①



f

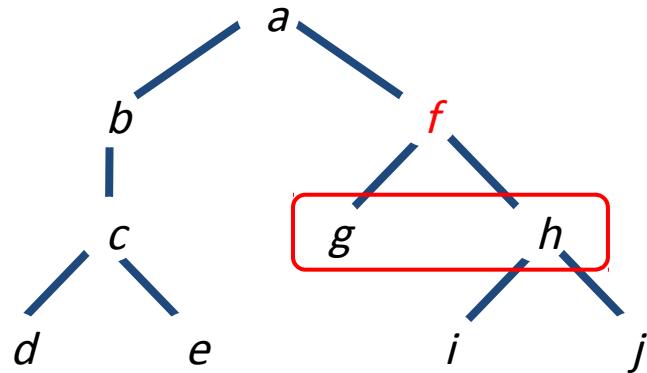
Examples

context node *f* and
v=*node()*: don't care node test

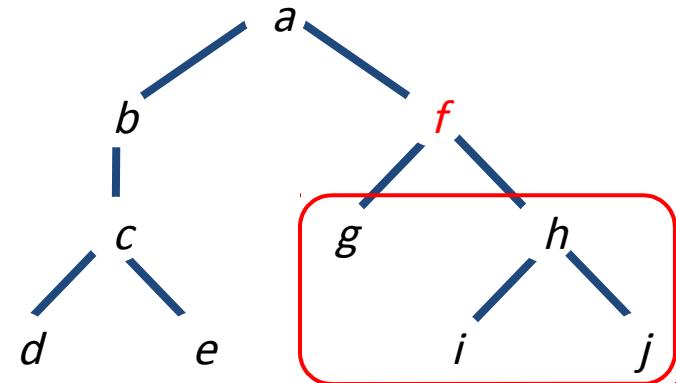
Path Expressions

- Just do it!

locstep (f, child::node())



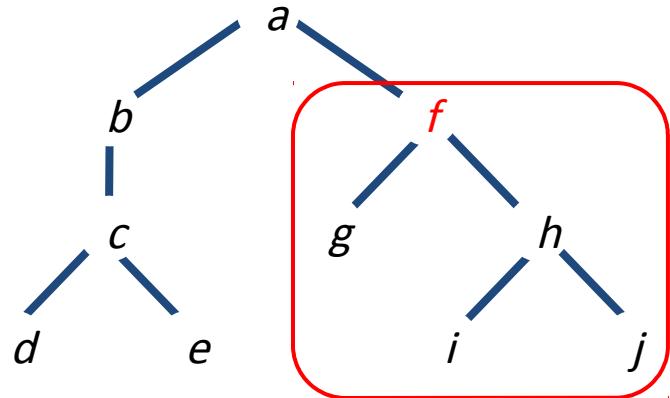
locstep (f, descendant::node())



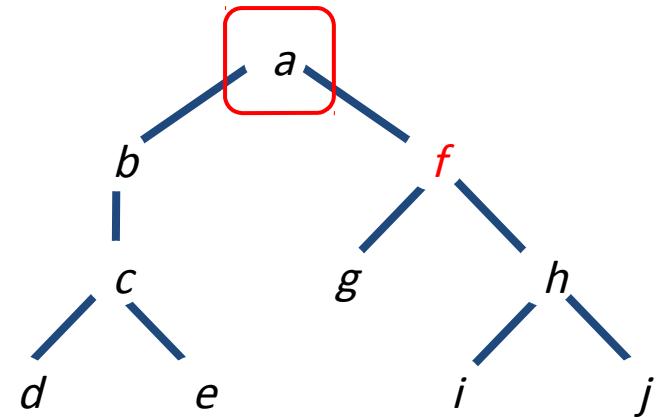
Path Expressions

- Just do it!

locstep (f, descendant-or-self::node())



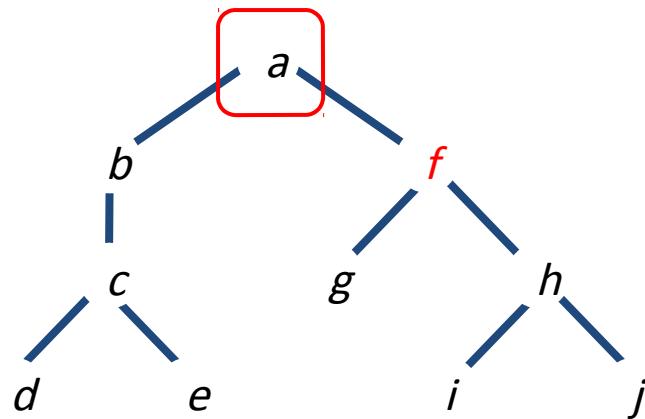
locstep (f, parent::node())



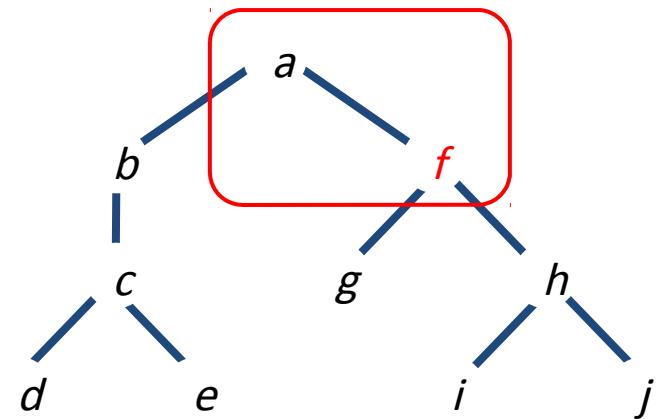
Path Expressions

- Just do it!

locstep (f, ancestor::node())



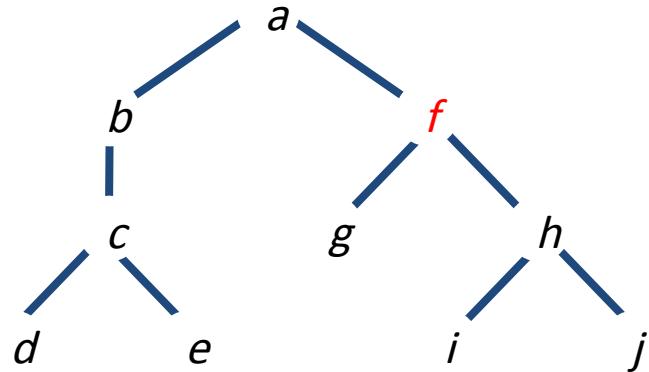
locstep (f, ancestor-or-self::node())



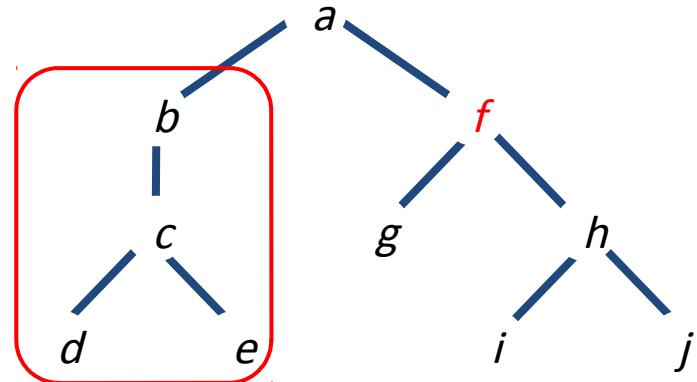
Path Expressions

- Just do it!

locstep (f, following::node())



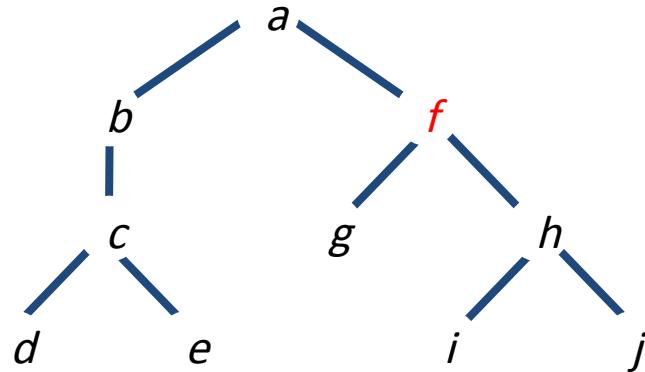
locstep (f, preceding::node())



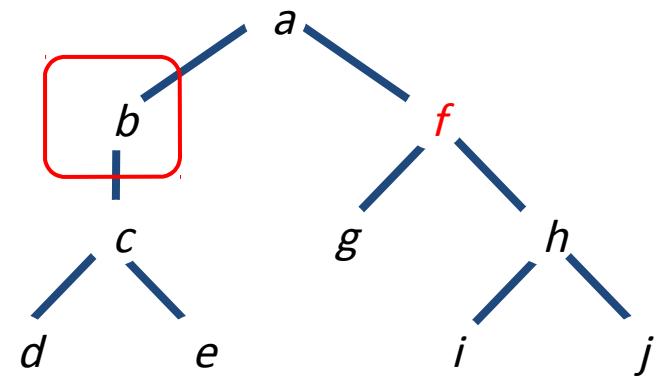
Path Expressions

- Just do it!

locstep (f, following-sibling::node())



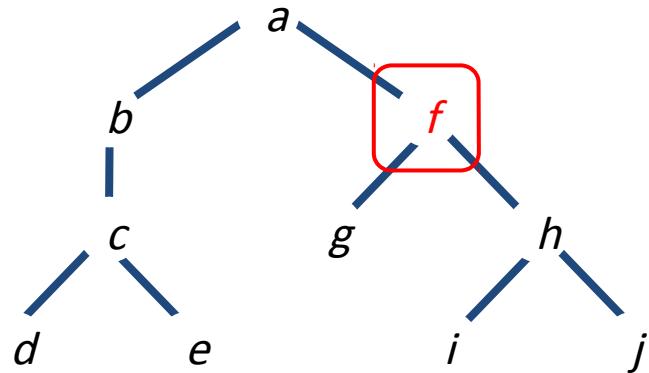
locstep (f, preceding-sibling::node())



Path Expressions

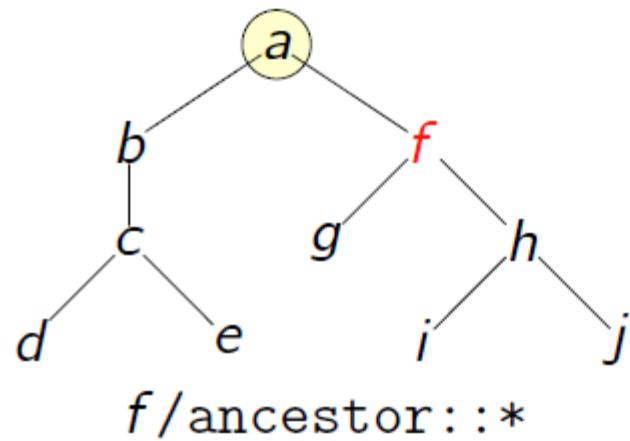
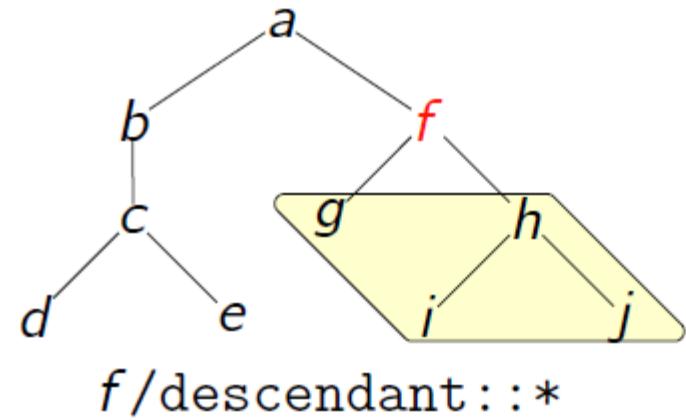
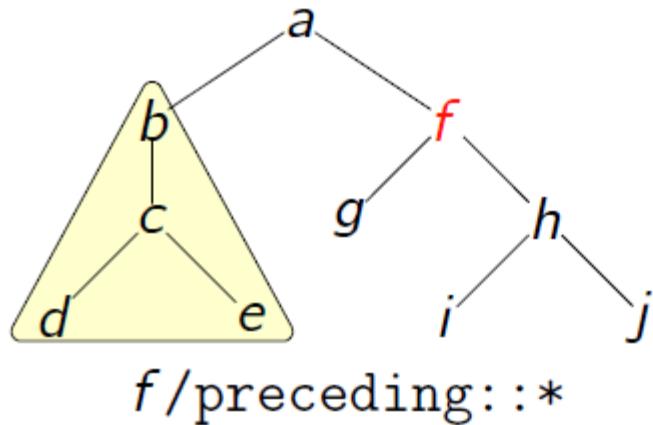
- Just do it!

locstep (f; self::node())



Path Expression

- Axes
 - The **ancestor**, **descendant**, **following**, **preceding** and **self** axes partition a document (ignoring attribute and namespace nodes): they do not overlap and together they contain all the nodes in the document.



$$\begin{aligned}
 & f/\text{descendant}::^* \cup f/\text{ancestor}::^* \cup \\
 & f/\text{preceding}::^* \cup f/\text{following}::^* \cup \{f\} \\
 & = \\
 & \{a \dots j\}
 \end{aligned}$$

Path Expressions

- Absolute/relative location paths

LocationPath ::= RelativeLocationPath | AbsoluteLocationPath

AbsoluteLocationPath ::= '/' RelativeLocationPath? |
AbbreviatedAbsoluteLocationPath

RelativeLocationPath ::= Step | RelativeLocationPath '/' Step |
AbbreviatedRelativeLocationPath

Step ::= AxisSpecifier NodeTest Predicate*

Path Expressions

- Node tests
 - principal node type (PNT)
 - for attribute axes: $PNT = A$
 - for namespace axes: $PNT = N$
 - otherwise: $PNT = E$

Path Expressions

- Node tests

Node_test: node set $M \rightarrow \text{node set}$

`node()` true for any node whatsoever (don't care)

`tag_name t` true for elements/attributes named *t*

`*` true for any node of the principal node type

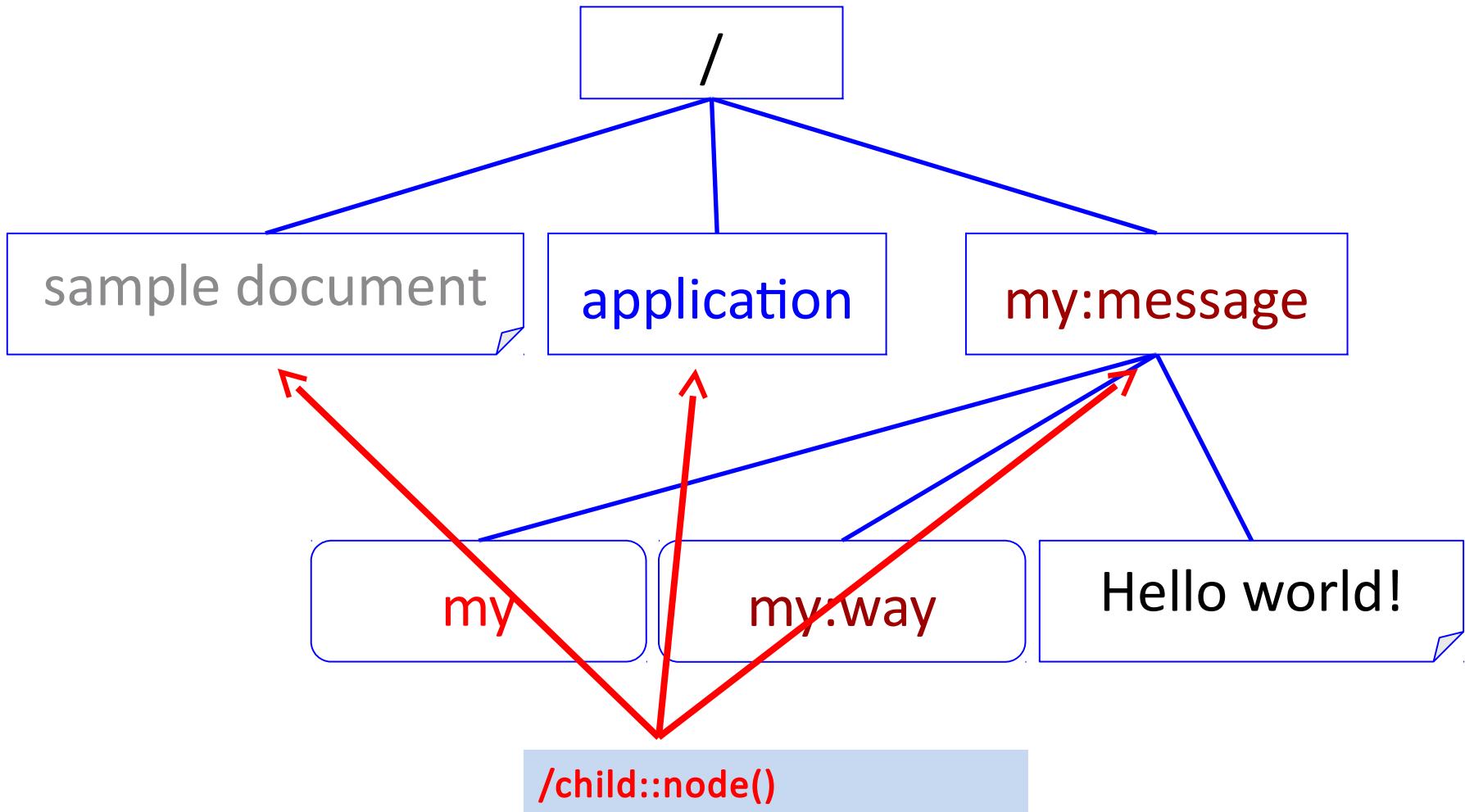
`text()` true for any Text node

`comment()` true for any Comment node

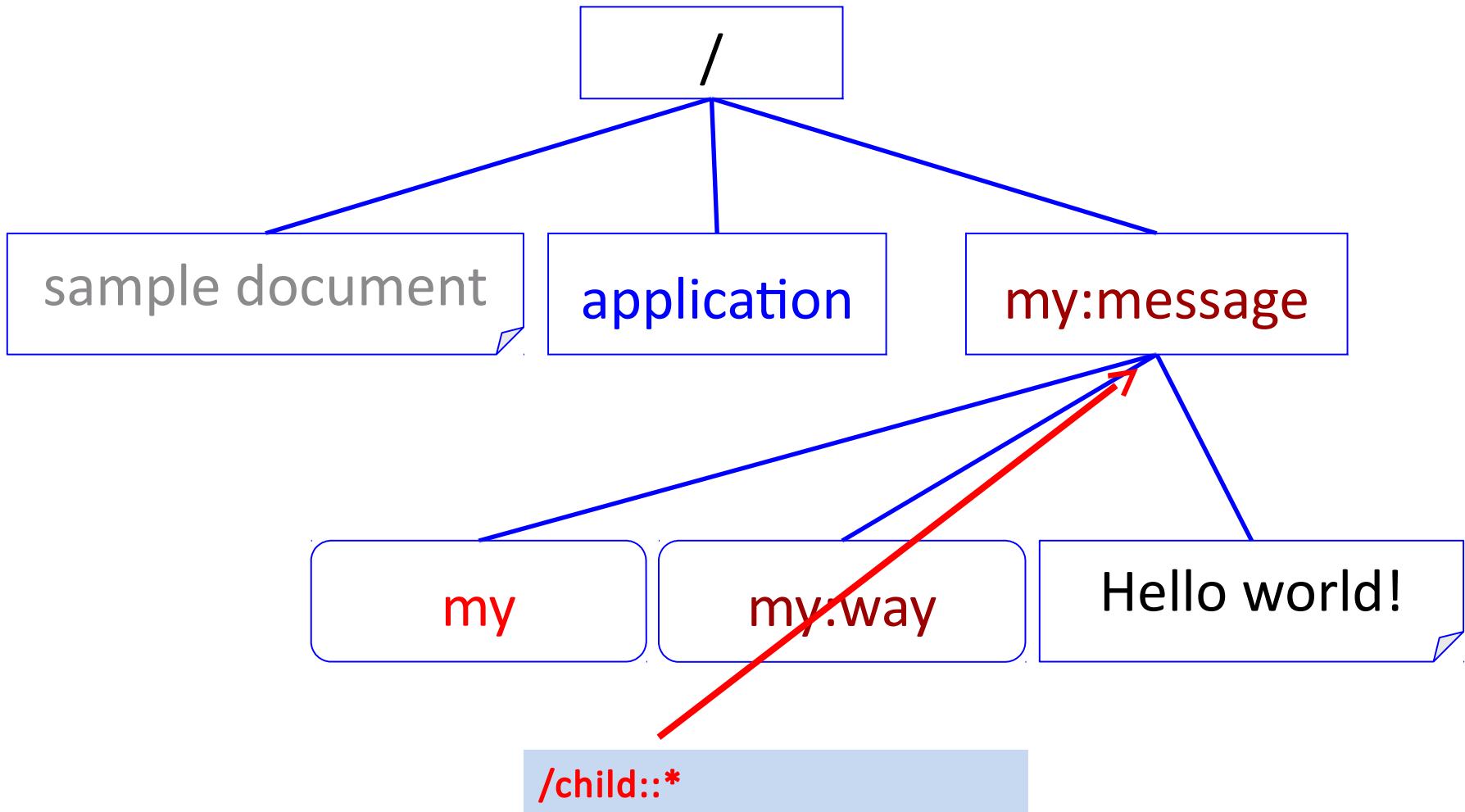
`processing-instruction(t)` true for any PI node of the form `<?t ... ?>`

"A node test that is a QName is true if and only if the type of the node is the principal node type and has an expanded-name equal to the expanded-name specified by the QName. For example, `child::para` selects the para element children of the context node; if the context node has no para children, it will select an empty set of nodes."

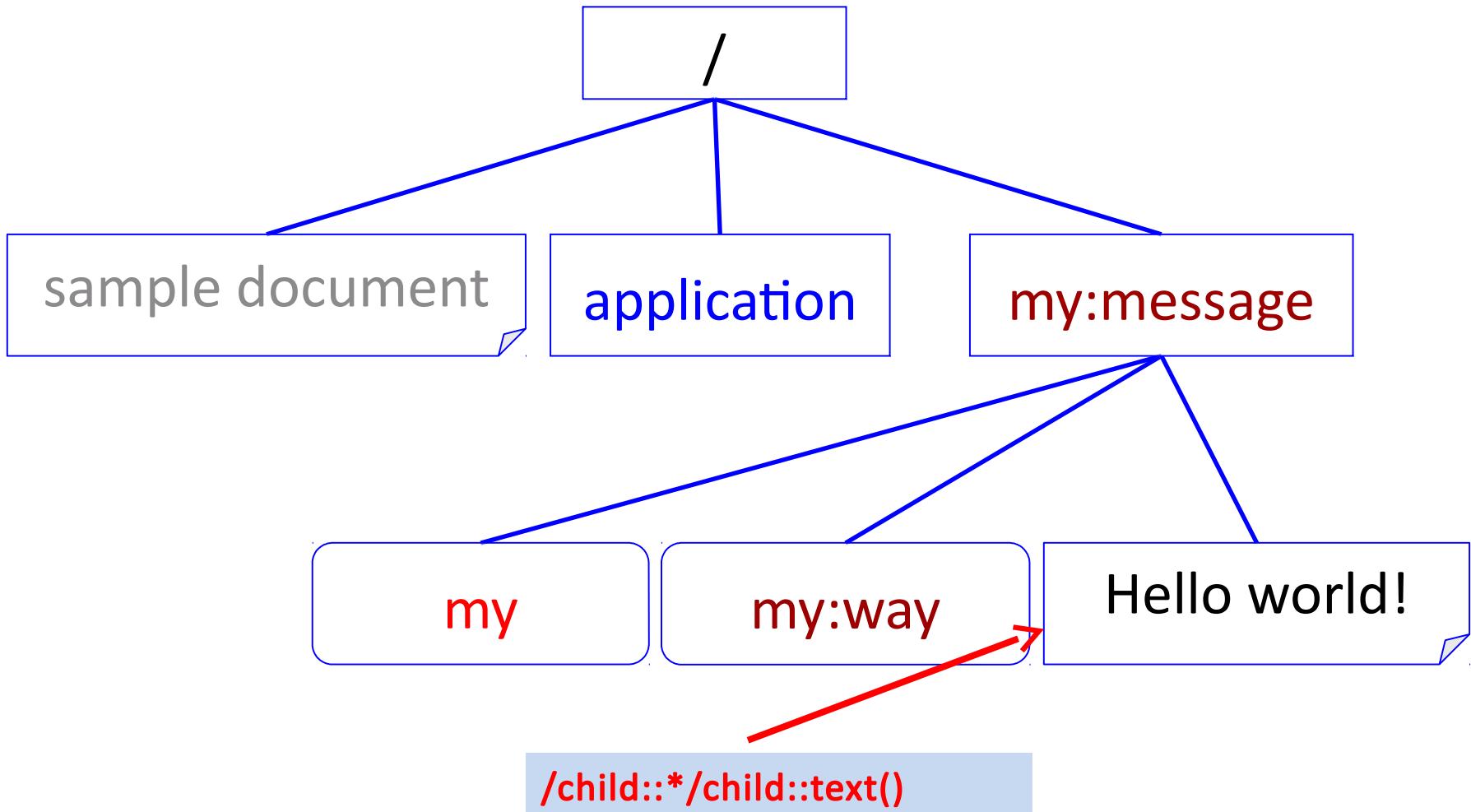
Path Expressions



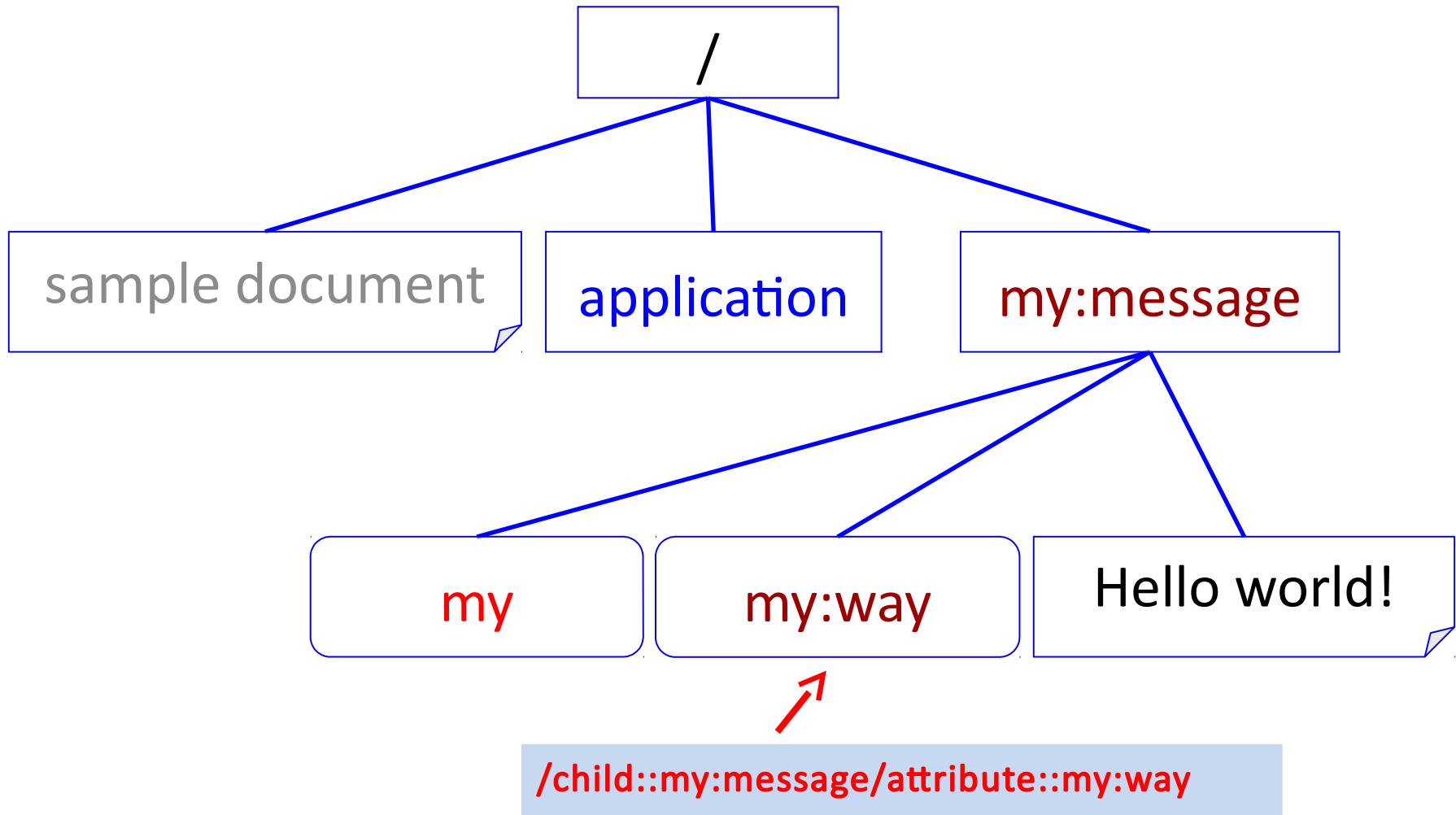
Path Expressions



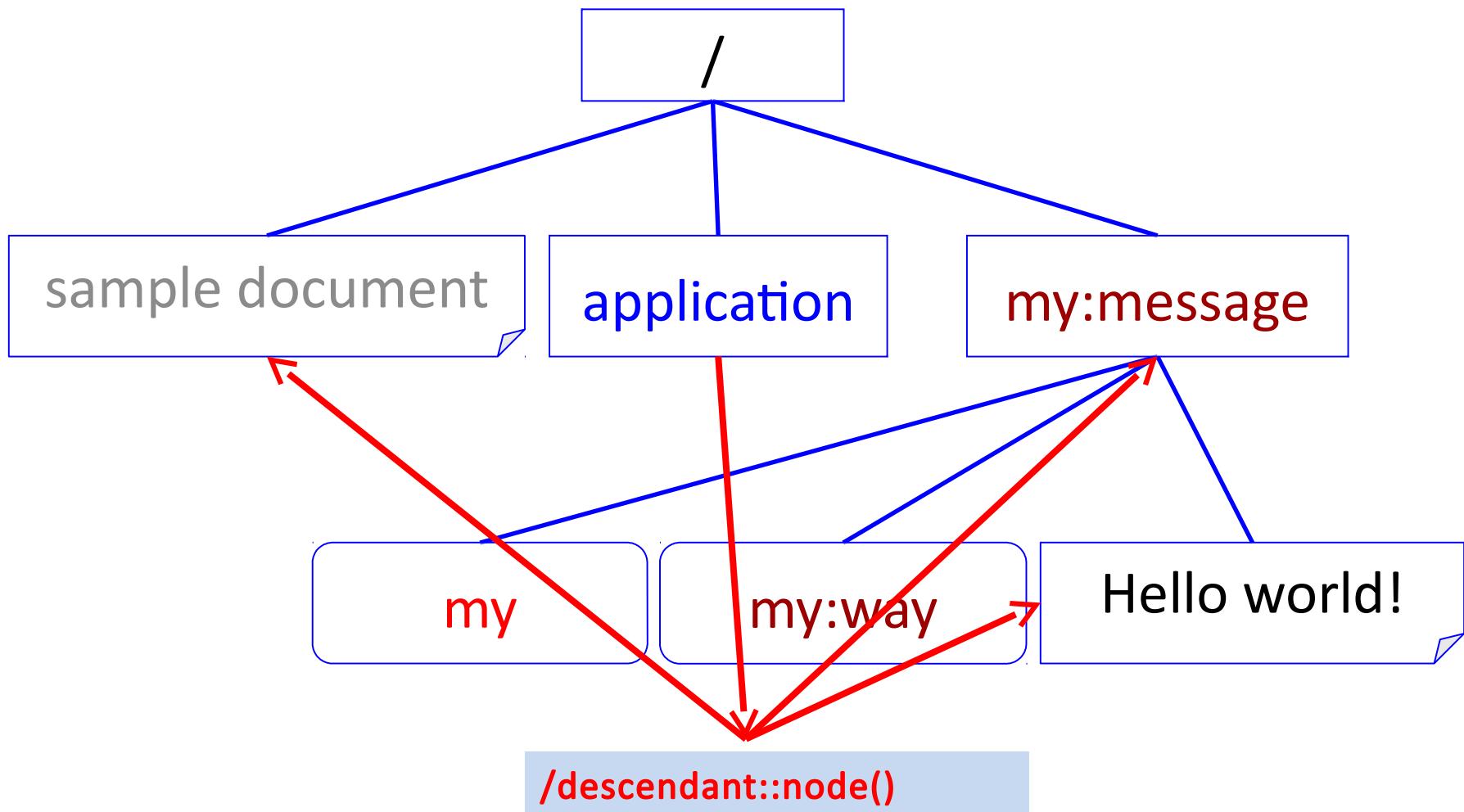
Path Expressions



Path Expressions



Path Expressions



Path Expressions

- Predicates
 - further test to retain a node or eliminate it from a node set:
predicate: node set \rightarrow node set
 - predicates are well-formed expressions consisting of
 - boolean operators
 - comparison operators
 - functions
 - node sets, numbers, strings
 - Predicates have high precedence (priority):
In the path expression $/s_0/s_1[q]$, predicate q is applied to the result of step s_1 , not to the whole path expression: $/s_0/s_1[q] \neq (/s_0/s_1)[q]$

Path Expressions

- Examples for XPath predicate functions

string concat(string, string, string)*

boolean contains(string, string)

string substring-before(string, string)

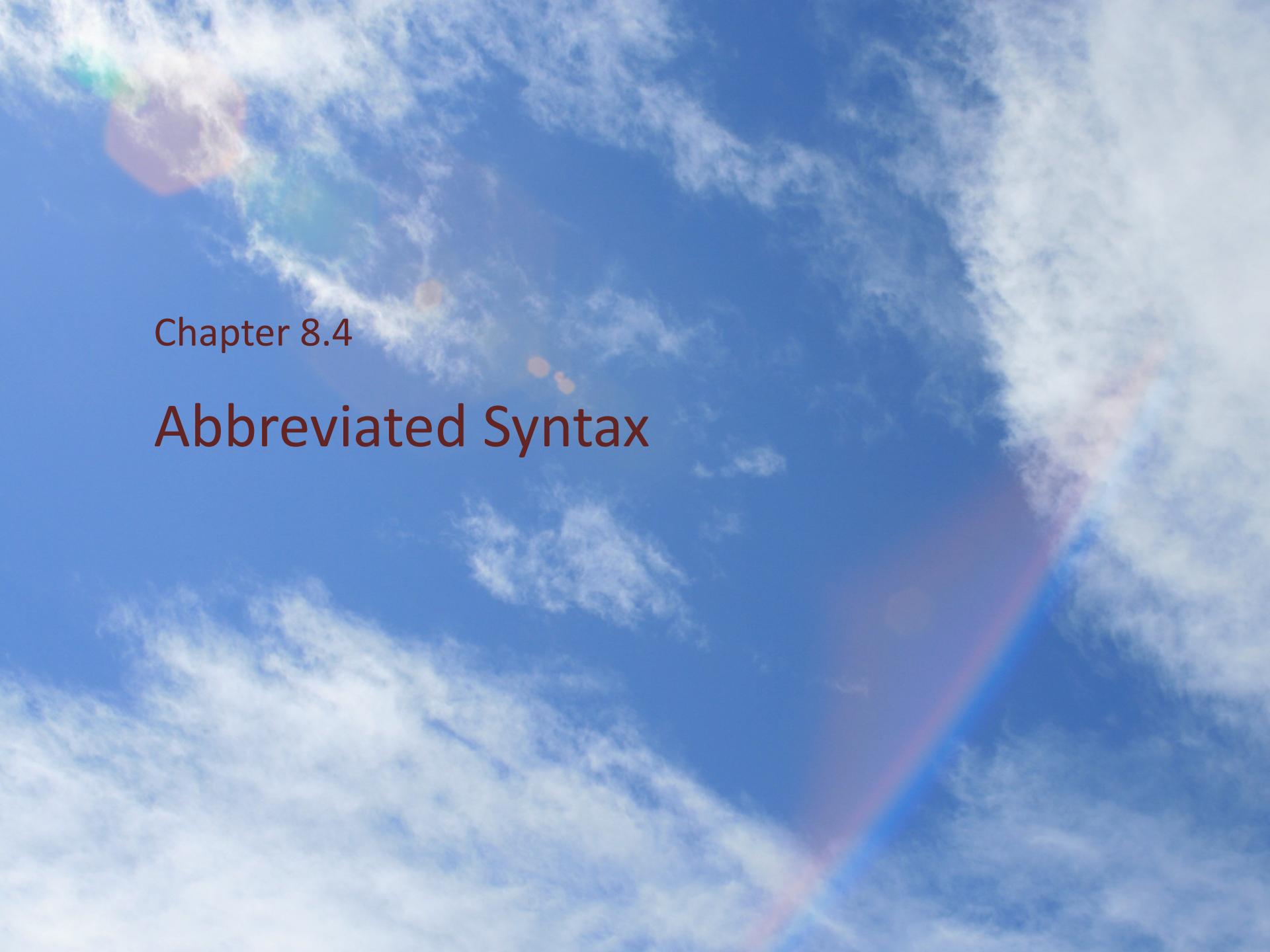
number string-length(string?)

string normalize-space(string?)

Path Expressions

- Predicate examples

- `/descendant::text()[contains(string(self::node()), "Hello")]`
selects all text nodes containing "Hello"
- `/descendant::node()[position()=5]`
selects the 5th node of the document
- `/descendant::node()[false()]`
empty set



Chapter 8.4

Abbreviated Syntax

Abbreviated Syntax

- Main rules

child:: can be omitted from a location step

- . is short for self::node()

- .. is short for parent::node()

// is short for /descendant-or-self::node()//

attribute:: can be abbreviated to @

Abbreviated Syntax

- Examples

`para` selects the `para` element children of the context node

* selects all element children of the context node

`text()` selects all text node children of the context node

`@name` selects the `name` attribute of the context node

`@*` selects all the attributes of the context node

`para[1]` selects the first `para` child of the context node

`para[last()]` selects the last `para` child of the context node

`*/para` selects all `para` grandchildren of the context node

`/doc/chapter[5]/section[2]` selects the second `section` of the fifth `chapter` of the `doc`

`chapter//para` selects the `para` element descendants of the `chapter` element children of the context node

`//para` selects all the `para` descendants of the document root and thus selects all `para` elements in the same document as the context node

Abbreviated Syntax

- Examples (cont'd.)

`.//para` selects the `para` element descendants of the context node

`..` selects the parent of the context node

`../@lang` selects the `lang` attribute of the parent of the context node

`para[@type="warning"]` selects all `para` children of the context node that have a `type attribute` with value `warning`

`para[@type="warning"] [5]` selects the fifth `para` child of the context node that has a `type attribute` with value `warning`

`para[5] [@type="warning"]` selects the fifth `para` child of the context node if that child has a `type attribute` with value `warning`

`chapter[title="Introduction"]` selects the `chapter` children of the context node that have one or more `title` children whose typed value is equal to the string `Introduction`

`chapter[title]` selects the `chapter` children of the context node that have one or more `title` children

Abbreviated Syntax

- Examples (cont'd.)

`employee[@secretary and @assistant]` selects all the `employee` children of the context node that have both a `secretary` attribute and an `assistant` attribute

`book/ (chapter|appendix)/section` selects every `section` element that has a parent that is either a `chapter` or an `appendix` element, that in turn is a child of a `book` element that is a child of the context node.

XML Navigation, Transformation

XSLT—Transforming XML Documents

Lecture "XML in Communications"

Chapter 9

Dr.-Ing. Jesper Zedlitz

Research Group for Communication Systems

Dept. of Computer Science

Christian-Albrechts-University in Kiel



Recommended Reading

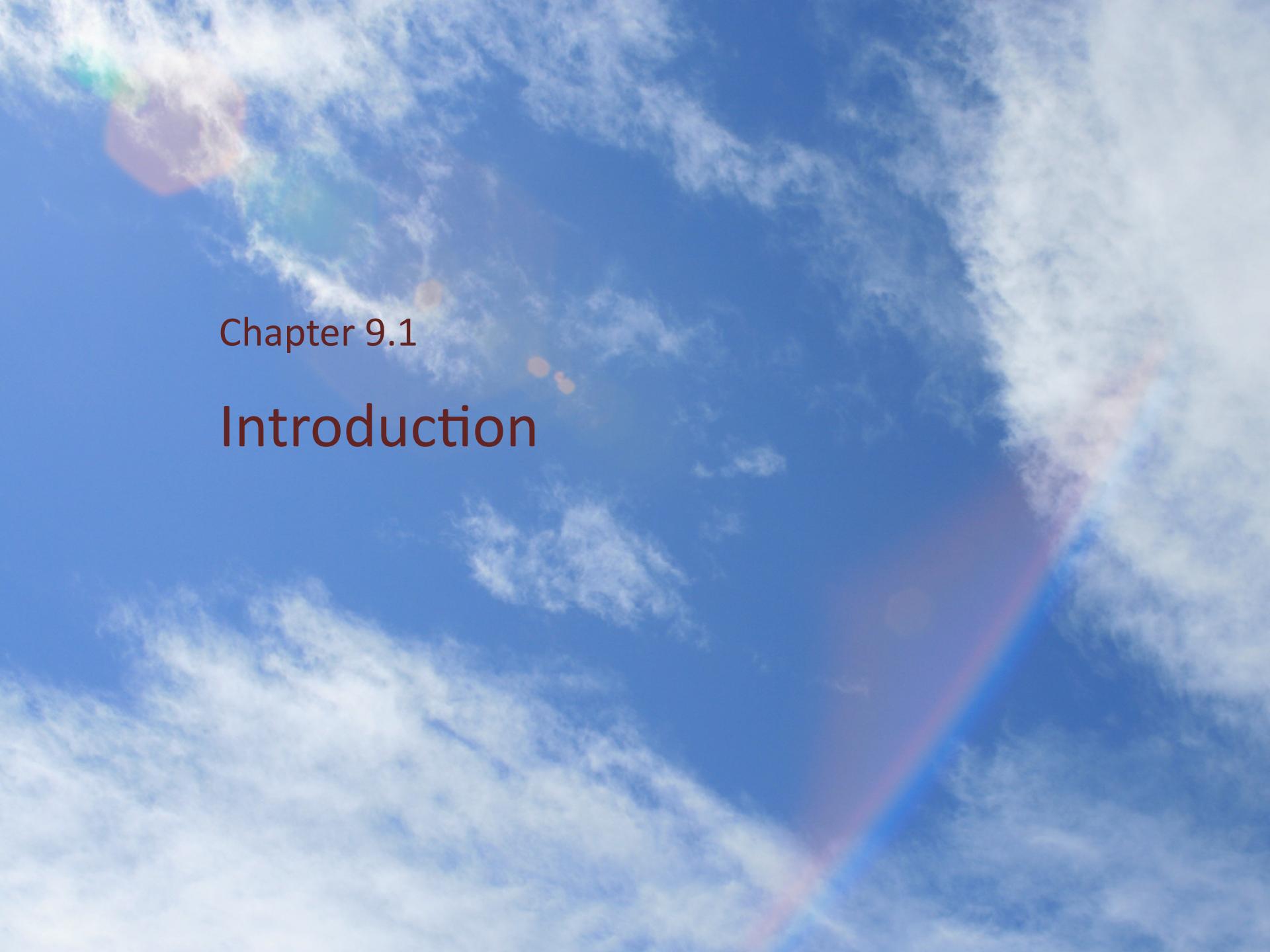
Informatik · CAU Kiel

- M. Kay (Editor): XSL Transformations (XSLT) Version 2.0.
W3C Recommendation 23 January 2007.
<http://www.w3.org/TR/xslt20/>

Overview

Informatik · CAU Kiel

1. Introduction
2. Models for Tree Editing
3. Stylesheet Structure
4. XSLT Programming Examples
5. XSLT-based Schematron Implementation

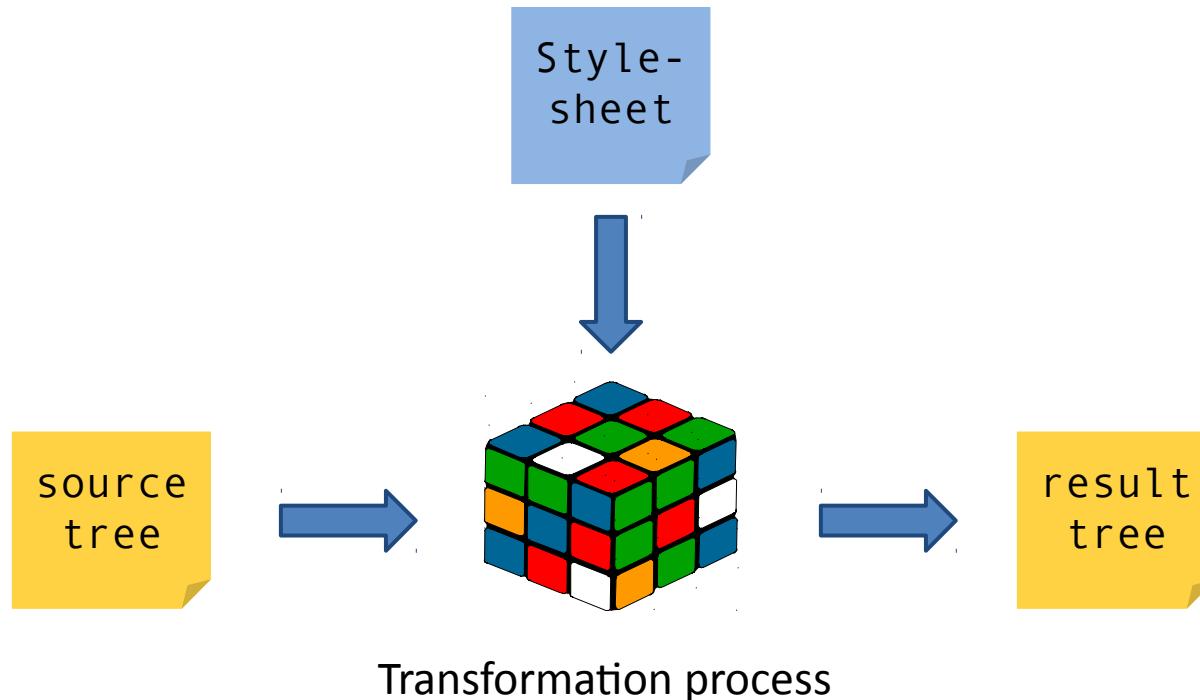


Chapter 9.1

Introduction

Introduction

- XSLT: *eXtensible Stylesheet Language for Transformations*
 - Rule-based (declarative) language for transformations
 - Transformation of an XML *source tree* into an (arbitrary) *result tree*



Introduction

- Where does XSLT fit?
 - Dependency path: XML → XPath → XSLT
- Status
 - Full W3C Recommendation, in wide use
 - Version 2.0 available since 2007:
<http://www.w3.org/TR/xslt20/>

Introduction

- Transformation?
 - generate (new) constant content,
 - suppress content,
 - move subtrees (e.g., swap day/month in a date),
 - copy subtrees (e.g., copy section titles into tables of contents),
 - sort content,
 - general transformations that compute new from given content.

Introduction

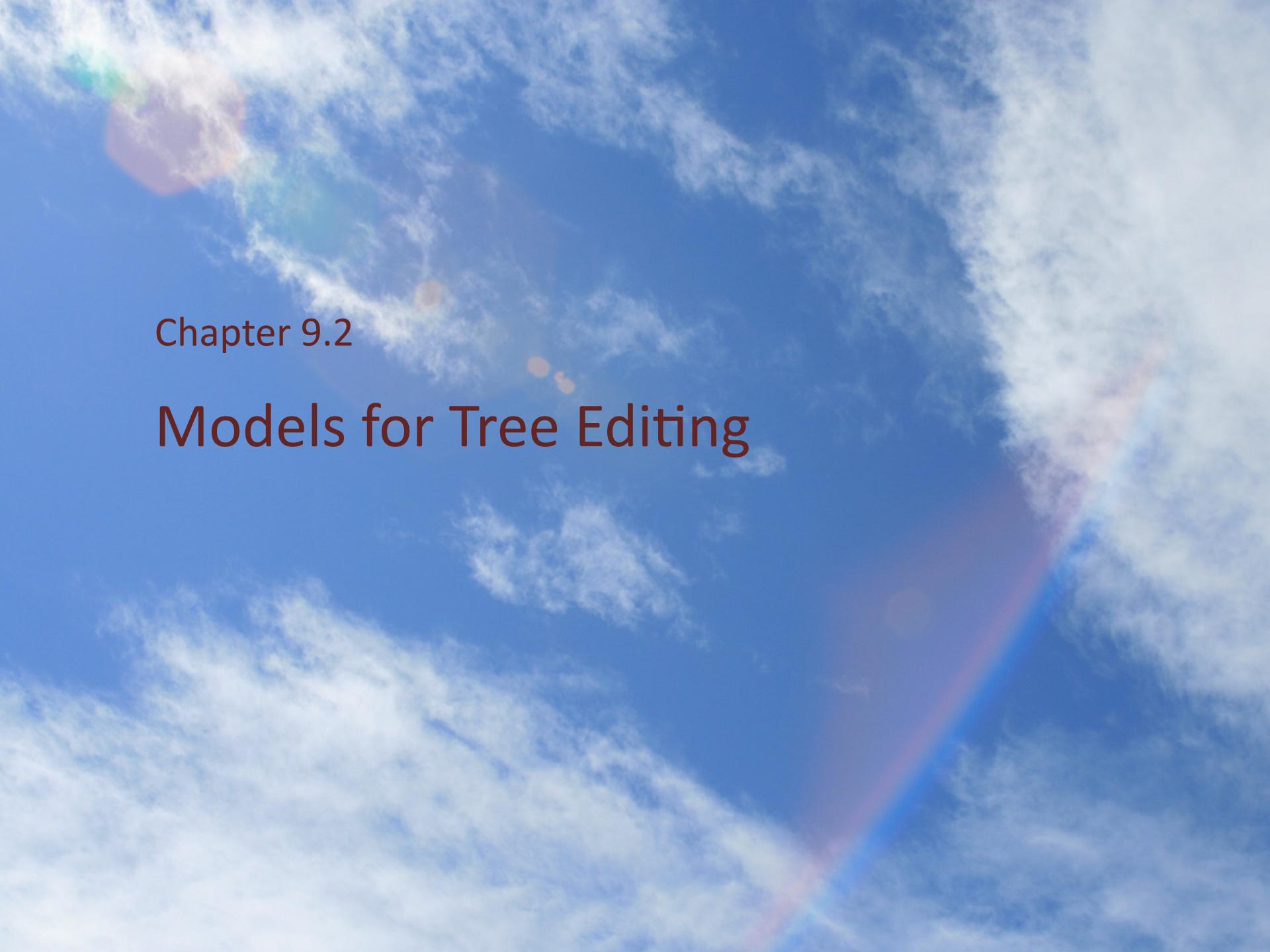
- Where to transform documents?
 - in the server:
A server program, such as a Java servlet, can use a stylesheet to transform a document automatically and serve it to the client.
 - in the client:
A client program, such as a browser, can perform the transformation and render the transformed document to the user.
 - with a separate program:
Several standalone programs (XSLT processors), e.g. Saxon, may perform XSLT transformations.

Introduction

- Stylesheet location
 - Specify a stylesheet in the document preamble using a processing instruction

```
<?xml version="1.0" ?>
...
<?xml-stylesheet type="text/xsl"
                  href="http://xyz.edu/Report/report.xsl" ?>

<Report Date="2008-11-11">
  ...
</Report>
```



Chapter 9.2

Models for Tree Editing

Models for Tree Editing

- Different kinds of approaches to tree editing
 - Functional
 - Rewrite rule-based
 - Template-based
 - Imperative

Models for Tree Editing

- Functional tree rewriting
 - Recursive processing
 - Invoke start function at the root, construct a new tree
 - Can think of this as "node functions"
 - Result is "compositional" — substitution is generally nested
 - Side effects often avoided: caching values, clarity

Models for Tree Editing

- Rule-based (rewriting systems)
 - A transformation is defined by a list of pattern/result pairs
 - Each is a piece of a tree with "holes" (variables)
 - A match leads to replacement of the matched tree nodes by a result tree
 - Variables shared between pattern and result allow preservation and re-arrangement of arbitrary data
 - Powerful, incremental definitions
 - Non-deterministic processing

Models for Tree Editing

- Template based processing
 - Starting point is a pattern document.
 - This model is very familiar from many web-based systems.
 - It contains literal results interleaved with queries and sometimes imperative code.
 - Well-suited to repetitive or rigid structures.
 - Often requires extensions to deal with recursion and looping.

Models for Tree Editing

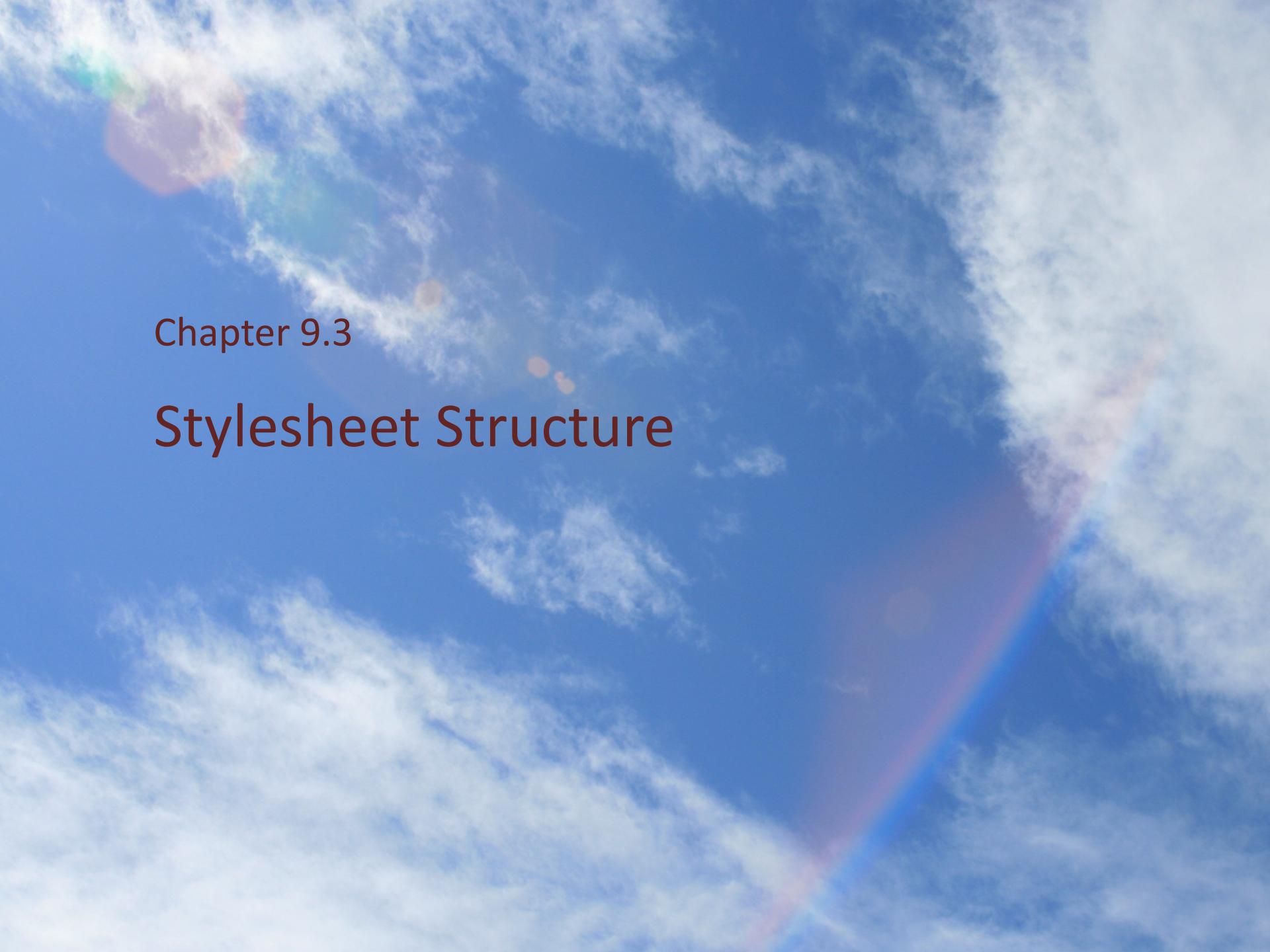
- Imperative
 - Parser calls imperative code, which uses
 - stacks
 - global variables
 - explicit output commands
 - Results are obtained by side effects.
 - Reasoning about the program may be hard, but creating it often starts out easily.
 - This approach makes it easy to create non-XML, or ill-formed XML documents.

Models for Tree Editing

- What's the biggest drawback to tree editing?
 - Buffering!
 - You need a copy of the tree to edit:
a document entirely in-memory!
 - Doing this from secondary storage is fairly subtle,
and has its own performance penalties.

Models for Tree Editing

- What side is XSLT on?
 - Rule-based substitution
 - Results are like template languages
 - XPath addressing also looks like queries
in traditional template languages
 - Limited non-determinism



Chapter 9.3

Stylesheet Structure

Stylesheet Structure

- An XSLT stylesheet is an XML document
 - document element: `xsl:stylesheet` or `xsl:transform`
 - version attribute values: `1.0` or `2.0`
 - namespace URI: `http://www.w3.org/1999/XSL/Transform`
 - MIME media type: `text/xml` or `application/xml`

Stylesheet Structure

- Basic structure
 - Bunch of templates of the form:

```
<xsl:template match="pattern">  
    ... rules ...  
</xsl:template>
```

- **match** attribute specifies elements in *source tree*
- **rules** specify contribution to *result tree*
- Rules are "instantiated" per matching node.

Stylesheet Structure

- Closer look to "templates"
 - The `match` attribute
 - derived from (abbreviated) XPath
 - selects the node set in the source tree the template rules are applied to
 - Rules generate
 - Literal output
 - Results of XSLT functions
 - Results of further template applications
 - Results of queries on the document

Stylesheet Structure

- A trivial stylesheet

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        x
    </xsl:template>
</xsl:stylesheet>
```

- replaces the source tree by "x"

Stylesheet Structure

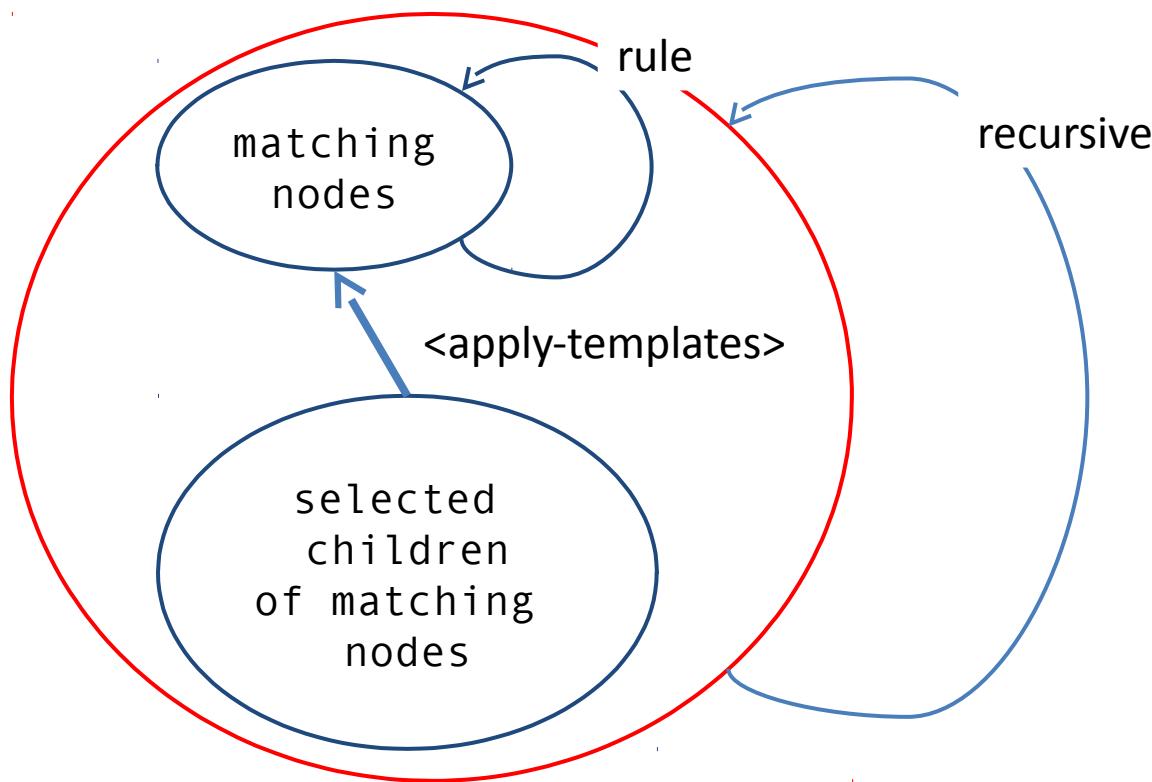
- Result tree creation
 - Literals – any element not in xsl namespace
 - <xsl:text> send content directly to output (retain whitespace)
 - <xsl:value-of> expression processing
 - <xsl:copy> copy current node into result tree
 - <xsl:element> instantiate an element
 - <xsl:attribute> instantiate an attribute
 - ...

Stylesheet Structure

- Result tree creation (cont'd.)
 - Further template applications
 - controlled by the `<xsl:apply-templates>` element:
apply matching templates to the current element's child nodes.
 - `<xsl:apply-templates>` element may have a
`select` attribute: reduce set of nodes
 - If no template available after matching and selection:
add string value of remaining nodes to result tree

Stylesheet Structure

- Result tree creation (cont'd.)



Stylesheet Structure

- Conflict Resolution
 - If several templates match: use the best matching template, i.e. the template with the smallest (by inclusion) set of matching nodes.
 - If several of those:
 - Imported templates have lower priority.
 - Take **priority** attribute value into account if available.
 - If several patterns with equal priority → **error**.
 - If no template matches, use the matching **default** template—later.

Stylesheet Structure

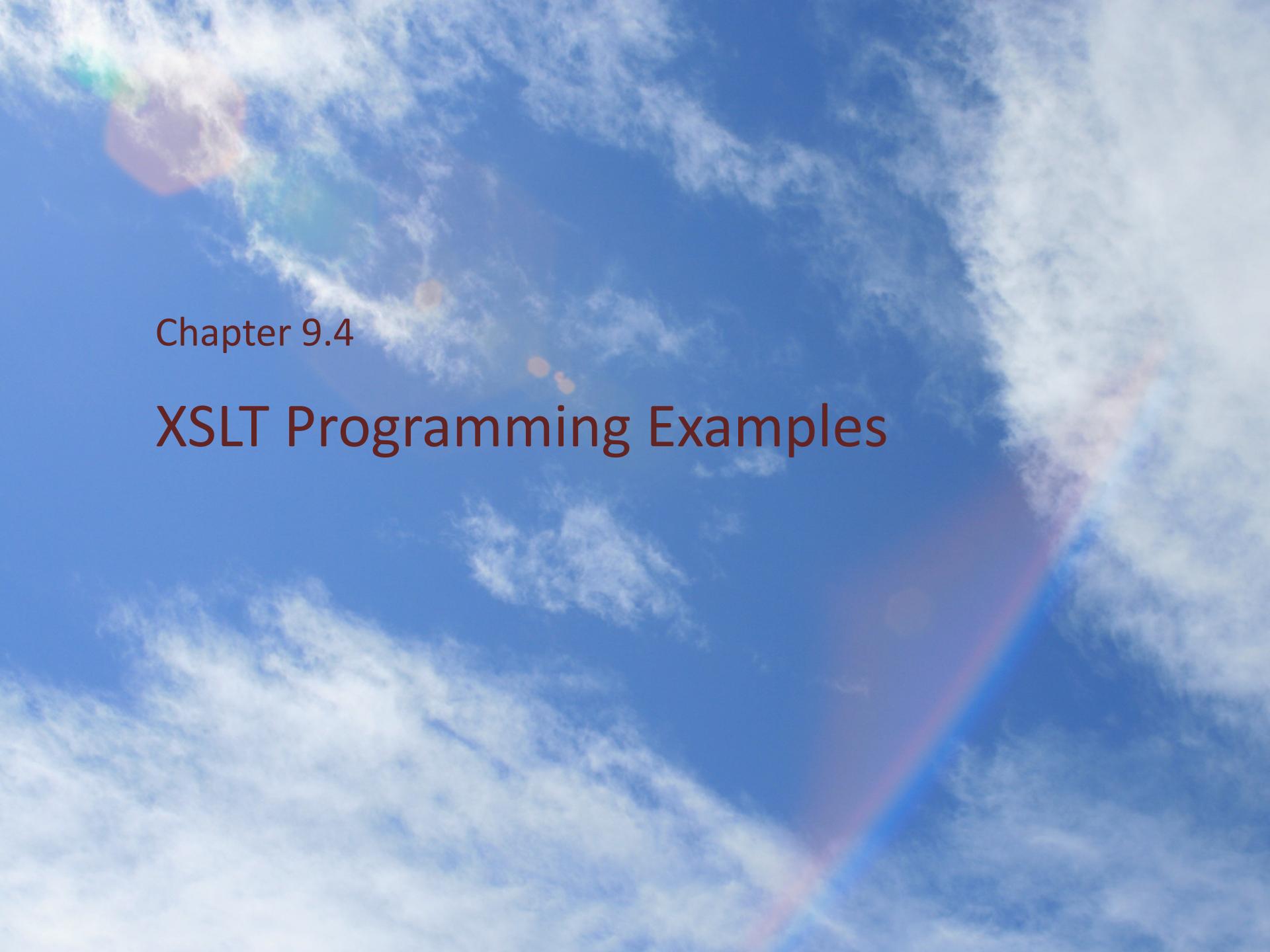
- XSLT processing uses built-in templates:

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

<xsl:template match="text()|@*">
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="processing-instruction()|comment()"/>
```

- "The stylesheet author can override a built-in template rule by including an explicit template rule."



Chapter 9.4

XSLT Programming Examples

XSLT Programming Examples

- Stylesheet elements

xsl:import	href = uri-reference	import of further stylesheets; existing definitions and rules are overwritten
xsl:include	href = uri-reference	same as import, but without overwriting
xsl:strip-space	elements = tokens	controls removal of text nodes with white space only
xsl:preserve-space	elements = tokens	
xsl:output	method = "xml" "html" "text" ...	output syntax
xsl:key	name = ...	declaration of cross references by patterns
xsl:decimal-format		definition of formats for decimals
xsl:namespace-alias		definition of a namespace alias
xsl:attribute-set		definition of attribute sets
xsl:variable		declaration of variables and parameters
xsl:param		
xsl:template		

XSLT Programming Examples

- Stylesheet elements

<xsl:apply-templates select = node-set-expression .../>	In the absence of a select attribute, the xsl:apply-templates instruction processes all of the children of the current node, including text nodes. A select attribute can be used to process nodes selected by an expression instead of processing all children.
<xsl:for-each select = node-set-expression/>	iteration
<xsl:value-of select = string-expression .../>	inserts string values into result tree
<xsl:copy-of select = expression />	copies node set
<xsl:sort select = string-expression .../>	sorting a node set
<xsl:element name = { qname } namespace = { uri-reference } use-attribute-sets = qnames> <!-- Content: template --> </xsl:element>	generates elements in result tree

XSLT Programming Examples

- Sample XSLT document

```
<xsl:template match="/">
  <html><body><h1>
    <xsl:value-of select="message"/>
  </h1></body></html>
</xsl:template>
```

- The `<xsl:template match="/">` chooses the root
- The `<html><body><h1>` is written to the output file
- The contents of the `message` element is written to the output file
- The `</h1></body></html>` is written to the output file
- The result is: `<html><body><h1>Howdy!</h1></body></html>`

XSLT Programming Examples

- How XSLT works
 - The XML text document is read in and stored as a tree of nodes
 - The `<xsl:template match="/">` template selects the entire tree
 - The rules within the template are applied to the matching nodes, thus changing the structure of the tree
 - Unmatched parts of the XML tree are not changed
 - After the template is applied, the tree is written out as a text document

XSLT Programming Examples

- Some functions

`xsl:value-of`

- `<xsl:value-of select="XPath expression" />`
selects the contents of an element and adds it to the output stream
- The `select` attribute is required.

XSLT Programming Examples

- Some functions

- xsl:for-each

- loop statement
 - The syntax is

```
<xsl:for-each select="XPath expression">  
    Text to insert and rules to apply  
</xsl:for-each>
```

- Example: Make an unordered list of book titles

```
<ul>  
    <xsl:for-each select="//book">  
        <li><xsl:value-of select="title"/></li>  
    </xsl:for-each>  
</ul>
```

XSLT Programming Examples

- Sample XML file

```
<?xml version="1.0" ?>

<bookstore>

  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>

  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J.K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>

</bookstore>
```

XSLT Programming Examples

- More precise

```
<xsl:for-each select="//book">
  <li>
    <xsl:value-of
      select="title[../author='J.K. Rowling']"/>
  </li>
</xsl:for-each>
```

- This will output `` and `` for every book, so we will get empty bullets for authors other than J.K. Rowling
- There is no obvious way to solve this with just `xsl:value-of`

XSLT Programming Examples

- Conditional transformation

```
<xsl:for-each select="//book">
  <xsl:if test="author='J.K. Rowling'">
    <li>
      <xsl:value-of select="title"/>
    </li>
  </xsl:if>
</xsl:for-each>
```

- `xsl:if` allows us to include content
if a given condition (in the `test` attribute) is true
- This does work correctly!

XSLT Programming Examples

- Choose
 - The `xsl:choose` ... `xsl:when` ... `xsl:otherwise` construct is XSLT's equivalent of Java's `switch` ... `case` ... `default` statement

```
<xsl:choose>
    <xsl:when test="some condition">
        ... some code ...
    </xsl:when>
    <xsl:otherwise>
        ... some code ...
    </xsl:otherwise>
</xsl:choose>
```

XSLT Programming Examples

- Sorting elements

- You can place an `xsl:sort` inside an `xsl:for-each`
- The attribute of the sort tells what field to sort on
- Example

```
<ul>
  <xsl:for-each select="//book">
    <xsl:sort select="author"/>
    <li>
      <xsl:value-of select="title"/> by
      <xsl:value-of select="author">
    </li>
  </xsl:for-each>
</ul>
```

- This example creates a list of titles *and* authors, sorted by author

XSLT Programming Examples

- Creating tags from XML data

- Suppose the XML contains

```
<name>jze's Home Page</name>
<url>http://www.uni-kiel.de/~jze</url>
```

- and we want to turn this into

```
<a href="http://www.uni-kiel.de/~jze">
jze's Home Page</a>
```

XSLT Programming Examples

- Creating tags, solution 1
 - `<xsl:attribute name="...">` adds the named attribute to the enclosing tag
 - The *value* of the attribute is the content of this tag

```
<a>
  <xsl:attribute name="href">
    <xsl:value-of select="url"/>
  </xsl:attribute>
  <xsl:value-of select="name"/>
</a>
```

- Result:
`
jze's Home Page`

XSLT Programming Examples

- Creating tags, solution 2
 - An attribute value template (AVT) consists of braces { } inside the attribute value
 - The content of the braces is replaced by its value

```
<a href="{url}">  
    <xsl:value-of select="name"/>  
</a>
```

- Result:

```
<a href="http://www.uni-kiel.de/~nl">  
jze's Home Page</a>
```

XSLT Programming Examples

- Calling named templates
 - You can name a template, then call it, similar to the way you would call a method in Java
 - The named template:

```
<xsl:template name="myTemplateName">
    ...body of template...
</xsl:template>
```
 - A call to the template:

```
<xsl:call-template name="myTemplateName">
    ...parameters...
</xsl:call-template>
```

XML Navigation, Transformation Schematron

Lecture "XML in Communications"
Chapter 10

Dr.-Ing. Jesper Zedlitz
Research Group for Communication Systems
Dept. of Computer Science
Christian-Albrechts-University in Kiel



Acknowledgement

Informatik · CAU Kiel

- This chapter is based on:

Roger L. Costello: XML Technologies Course

<http://www.xfront.com/files/tutorials.html>

Copyright (c) 2000. Roger L. Costello. All Rights Reserved.

Recommended Reading

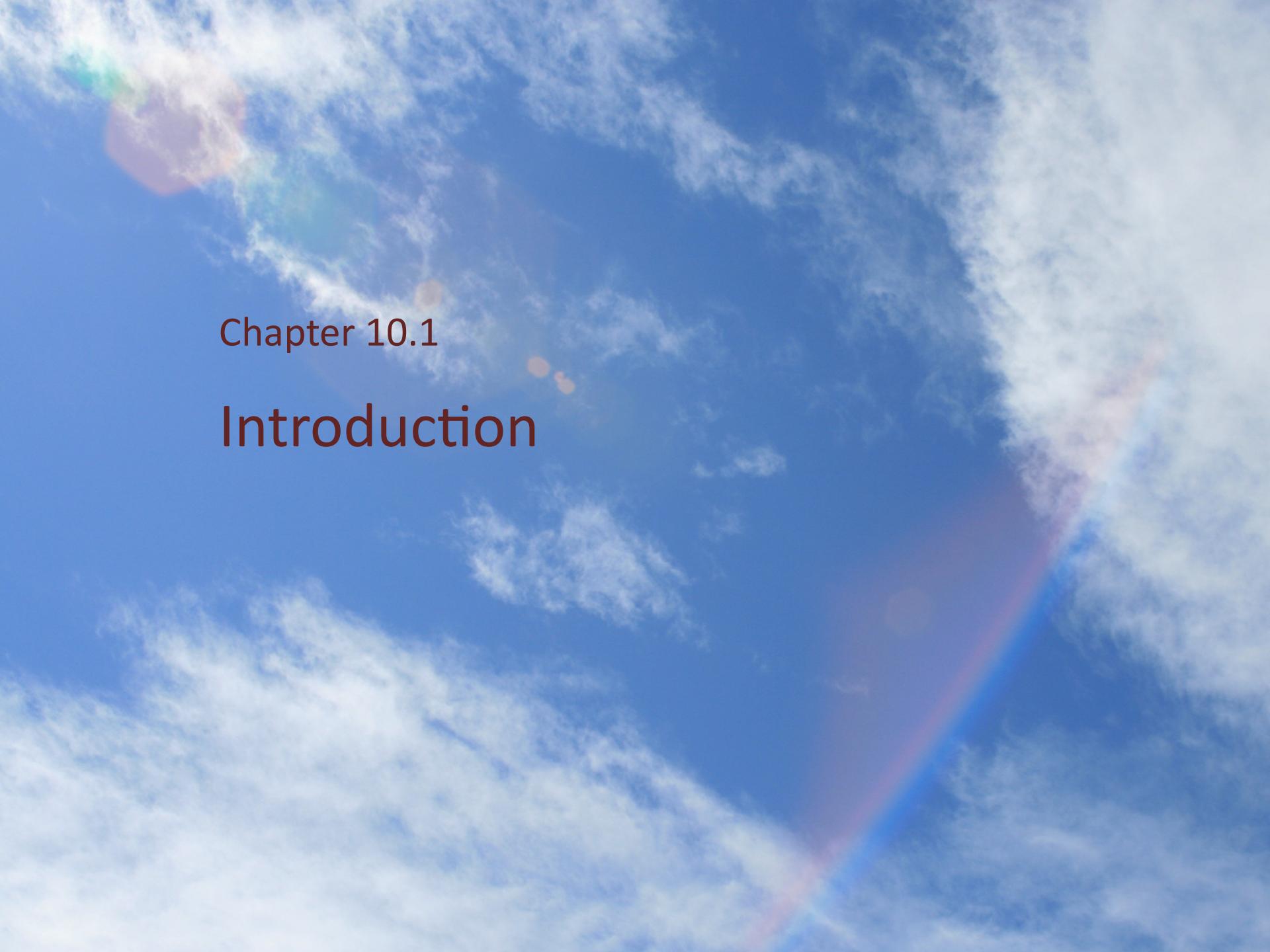
Informatik · CAU Kiel

- ISO/IEC-Standard 19757-3:2006
<http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html>
- <http://www.schematron.com/>

Overview

Informatik · CAU Kiel

1. Introduction
2. Basic Features
3. Schematron Assertions
4. Inside Schematron

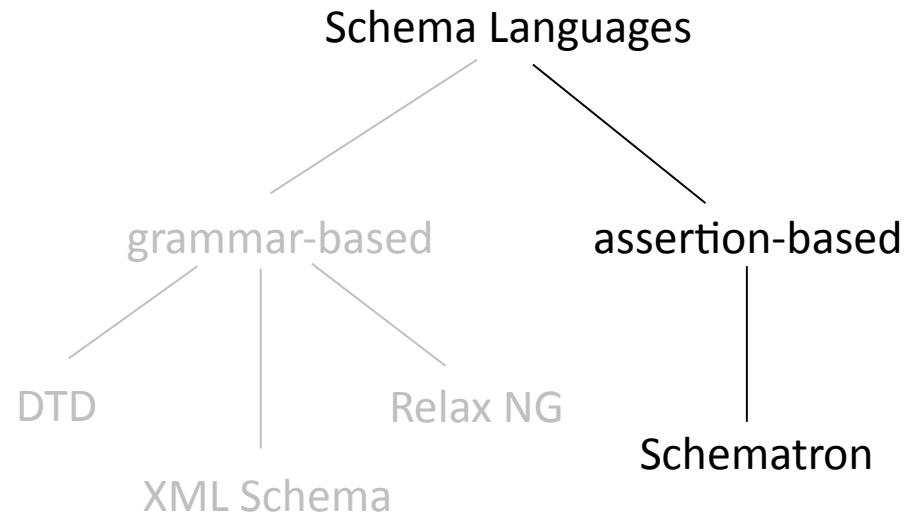


Chapter 10.1

Introduction

Introduction

- Schematron is assertion-based rather than grammar-based
 - grammar-based approaches take a closed approach: everything not explicitly allowed is treated as invalid
 - assertion-based approaches take an open approach: everything not explicitly disallowed is treated as valid



Introduction

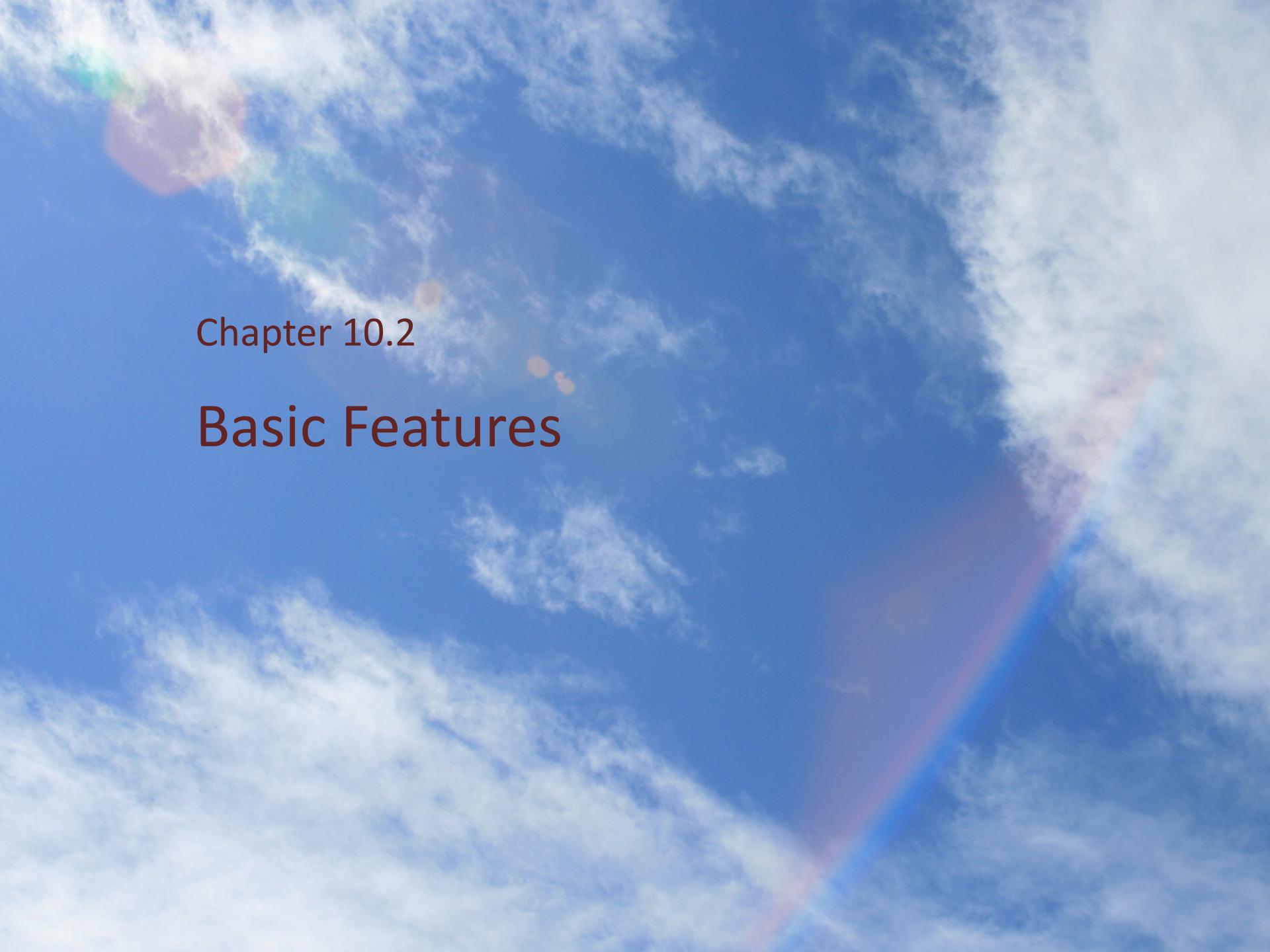
- Schematron is part of ISO/IEC's 19757 effort:
Document Schema Definition Languages (DSDL)
 - "The main objective of DSDL is to bring together different validation-related tasks and expressions to form a single extensible framework that allows technologies to work in series or in parallel to produce a single or a set of validation results." (<http://dsdl.org/>)
 - Schematron is undergoing standardization as one part of this:
"Rule-based validation – Schematron" – part 3
 - Namespace is
<http://purl.oclc.org/dsdl/schematron>

Introduction

- Schematron not be used as the only validation method
 - use grammar-based method for **structure** and **value** constraints
 - use Schematron for constraints that can't be described in grammar-based methods, such as
 - constraints between multiple elements/attributes
 - validation across documents (using document function)
- **co-constraints**

Introduction

- Co-constraint: constraints that exist between data
 - element-to-element co-constraints
 - element-to-attribute co-constraints
 - attribute-to-attribute co-constraints
 - includes element/attribute existence checking
 - includes formula checking:
validity checking may require performing an algorithm on the data,
e.g. "% values in column 5 of table 'results' add up to 100%"
 - includes co-constraint checking "within" single XML document,
or "across" multiple XML documents
(intra- and inter-document co-constraints)



Chapter 10.2

Basic Features

Basic Features

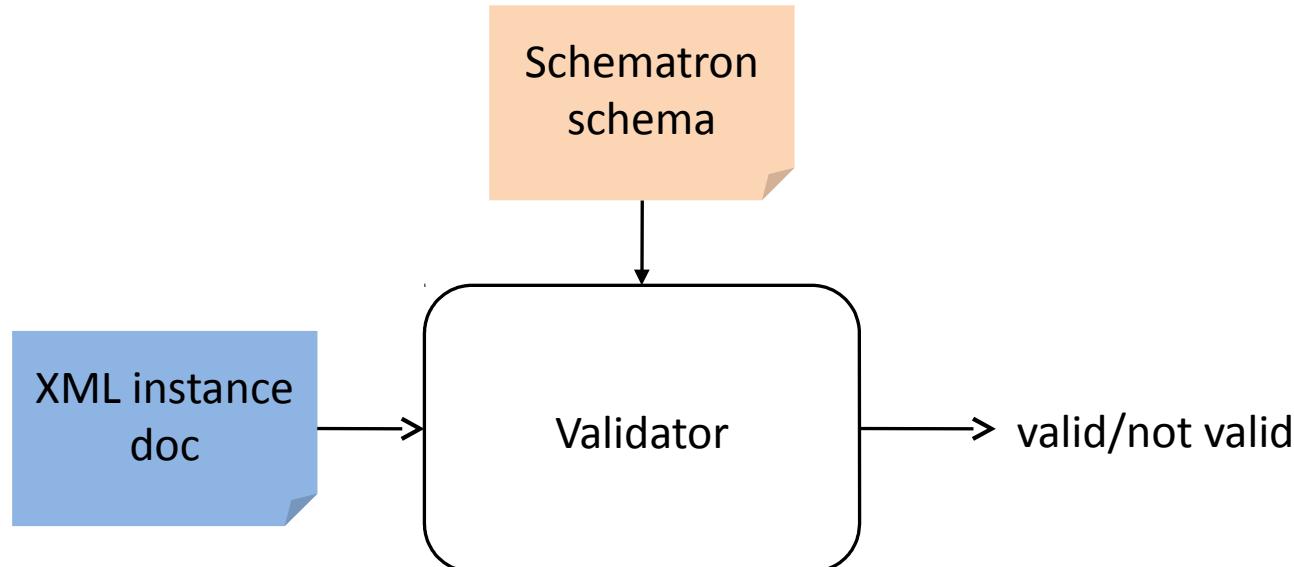
- Schematron core language elements
 - Assertions
 - conditions to be tested such as existence and values of elements/attributes
 - assert and report elements
 - Rules
 - groups of assertions
 - selects the set of context nodes under which they are evaluated

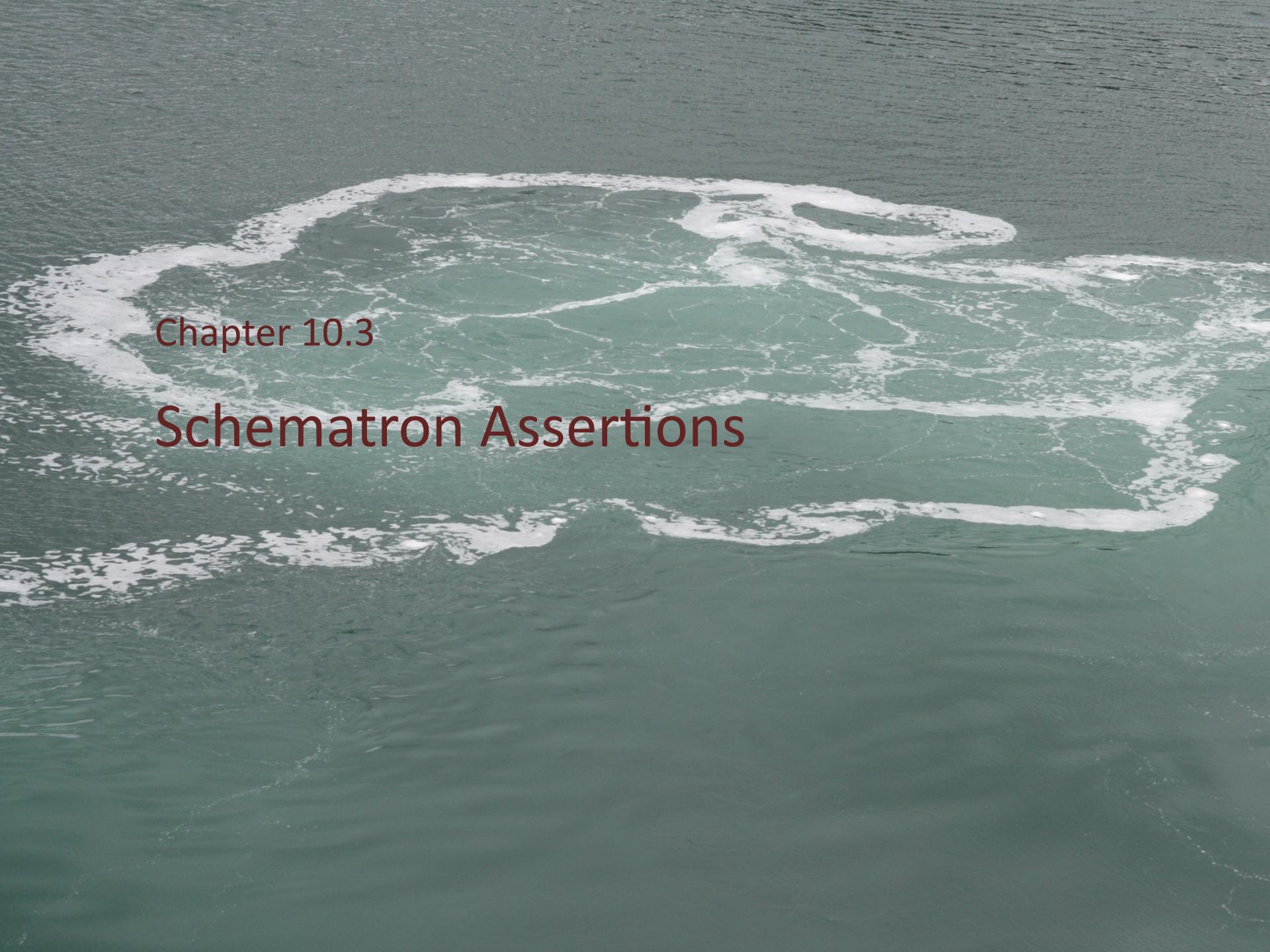
Basic Features

- Schematron language elements
 - Patterns
 - groups of rules with an id (used by phases, see below)
 - Phases
 - groups of patterns (specified by their id)
that allow evaluating only the rules in those patterns

Basic Features

- Schematron validation





Chapter 10.3

Schematron Assertions

Schematron Assertions

- Assertions use XPath expressions
 - can validate anything that can be expressed as a boolean XPath expression
- Assertion location
 - can be in a separate file, typically with a ".sch" extension
 - can be embedded within other schema files
- Validator
 - XSLT-based validators can easily be implemented
 - can also be implemented without using XSLT for better performance

Schematron Assertions

- Formal stuff
 - A Schematron schema is an XML document.
 - The Schematron elements are in this namespace:
<http://purl.oclc.org/dsdl/schematron>

```
<?xml version="1.0"?>
<sch:schema
  xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  ...
</sch:schema>
```

- By convention, the file name of a Schematron schema has the suffix
".sch"

Schematron Assertions

- Patterns
 - A Schematron schema is comprised of one or more pattern elements (with optional id):

```
<?xml version="1.0"?>
<sch:schema xmlns:sch=" http://purl.oclc.org/dsdl/schematron">
  <sch:pattern id="..."> ... </sch:pattern>
  <sch:pattern id="..."> ... </sch:pattern>
  <sch:pattern id="..."> ... </sch:pattern>
</sch:schema>
```

- The purpose of each pattern is to specify a relationship that must exist among information in the XML instance document.

Schematron Assertions

- Rules
 - A pattern element is comprised of one or more rule elements:

```
<sch:pattern id=" ... ">
    <sch:rule context=" ... "> ... </sch:rule>
    <sch:rule context=" ... "> ... </sch:rule>
    <sch:rule context=" ... "> ... </sch:rule>
</sch:pattern>
```

- The purpose of each rule is to specify a context node in the XML instance document, and assert a relationship relative to that context node.

Schematron Assertions

- **Assertions**
 - A rule element is comprised of one or more assert elements:

```
<sch:rule context=" ... ">
  <sch:assert test="boolean-XPath">
    ... message ...
  </sch:assert>
  <sch:assert test="boolean-XPath">
    ... message ...
  </sch:assert>
</sch:rule>
```

- The purpose of an assert element is to state a relationship that **must exist** in the XML instance document.

Schematron Assertions

- **Assertions**

```
<sch:assert test="boolean-XPath">  
  ... message ...  
</sch:assert>
```

- The value of the `test` attribute is an XPath expression.
- A Schematron validator checks the XML instance document against the XPath expressions of every `<assert>` element within a rule.
- The contents of the `<assert>` element is the text of message displayed to the user in case the assertion **does not** hold.

Schematron Assertions

- Reports

```
<sch:report test="boolean-XPath">  
  ... message ...  
</sch:assert>
```

- The value of the test attribute is an XPath expression.
- A Schematron validator checks the XML instance document against the XPath expressions of every `<report>` element within a rule.
- The contents of the `<report>` element is the text of message displayed to the user in case the assertion **does** hold.

Schematron Assertions

- Example 1

```
<?xml version="1.0"?>
<Document classification="secret">
  <Para classification="unclassified">
    Schematron is a schema language for XML.
  </Para>
</Document>
```

- Assume the following Security Classification Policy:
 - No `<Para>` element may have a classification value higher than the `<Document>`'s classification value.
 - Possible classification values, from highest to lowest:
top-secret — secret — confidential — unclassified

Schematron Assertions

- Example 1: first step

```
<sch:rule context="Para[@classification='top-secret']">
  <sch:assert test="/Document/@classification='top-secret'">
    If there is a Para labeled "top-secret",
    then the Document must be labeled top-secret.
  </sch:assert>
</sch:rule>
```

- Read as: Within the context of a `<Para>` element whose classification value is 'top-secret', I assert that the classification value of the `<Document>` element must be 'top-secret'.

Schematron Assertions

- Example 1: second step

```
<sch:rule context="Para[@classification='secret']">
  <sch:assert test=" (/Document/@classification='top-secret') or
                    (/Document/@classification='secret')">
    If there is a Para labeled "secret", then the Document
    must be labeled either secret or top-secret.
  </sch:assert>
</sch:rule>
```

- Read as: Within the context of a `<Para>` element whose classification value is 'secret', I assert that the classification value of the `<Document>` element must be either 'secret' or 'top-secret'.

Schematron Assertions

- Example 1: third step

```
<sch:rule context="Para[@classification='confidential']">
    <sch:assert test=" (/Document/@classification='top-secret') or
                      (/Document/@classification='secret') or
                      (/Document/@classification='confidential')">
        If there is a Para labeled "confidential" then the Document
        must be labeled either confidential, secret or top-secret.
    </sch:assert>
</sch:rule>
```

- Read as: Within the context of a `<Para>` element whose classification value is 'confidential', I assert that the classification value of the `<Document>` element must be either 'top-secret', 'secret' or 'confidential'.

Schematron Assertions

- Example 1: final solution

```
<sch:pattern id="SecurityClassificationPolicy">
  <sch:p>A Para's classification value cannot be more sensitive
        than the Document's classification value.</sch:p>
  <sch:rule context="Para[@classification='top-secret']"> ...
</sch:rule>
  <sch:rule context="Para[@classification='secret']"> ...
</sch:rule>
  <sch:rule context="Para[@classification='confidential']"> ...
</sch:rule>
</sch:pattern>
```

- Note that a pattern can be annotated using the `<sch:p>` element.
- Note that a pattern element wraps the rule elements, and the pattern element is given an identifier.

Schematron Assertions

- Example 2

```
<?xml version="1.0"?>
<election>
  <candidates>
    <candidate name="John Doe" percentage="51"/>
    <candidate name="Joe Blogs" percentage="49"/>
  </candidates>
</election>
```

Schematron Assertions

- Example 2: validation schema

```
<?xml version="1.0" encoding="UTF-8"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:pattern>
    <sch:rule context="candidates">
      <sch:assert test="sum(candidate/@percentage)='100'">
        The sum of the percentage attributes for the
        candidate elements must be 100.
      </sch:assert>
      <sch:report test="count(candidate) < 2">
        There must be at least two candidate elements inside
        the candidates element.
      </sch:report>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

Schematron Assertions

- Example 3

```
<?xml version="1.0" encoding="UTF-8"?>
<?oxygen SCHSchema="duplicateRole.sch"?>
<movies>
  <movie>
    <title>Elf</title>
    <actor name="Will Ferrell" role="Buddy"/>
    <actor name="James Caan" role="Walter"/>
    <actor name="Bob Newhart" role="Papa Elf"/>
    <actor name="Edward Asner" role="Santa"/>
    <actor name="Mary Steenburgen" role="Claus"/>
    <actor name="Zooey Deschanel" role="Jovie"/>
    <actor name="Mark Volkmann" role="Buddy"/>
    <actor name="Edward Asner" role="Mr. Grant"/>
  </movie>
</movies>
```

Schematron Assertions

- Example 3: validation schema

```
<?xml version="1.0" encoding="UTF-8"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:pattern>
    <sch:rule context="actor">
      <sch:report test="@role=preceding-sibling::actor/@role">
        Role assigned twice or more!
      </sch:report>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

Schematron Assertions

- Example 3: validation schema improved

```
<?xml version="1.0" encoding="UTF-8"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:pattern id="single">
    <sch:rule context="actor">
      <sch:report test="@role=preceding-sibling::actor/@role"
                  diagnostics="duplicateActorRole"/>
    </sch:rule>
  </sch:pattern>
  <sch:diagnostics>
    <sch:diagnostic id="duplicateActorRole">
      More than one actor plays the role
      <sch:value-of select="@role"/>. A duplicate is named
      <sch:value-of select="@name"/>.
    </sch:diagnostic>
  </sch:diagnostics>
</sch:schema>
```

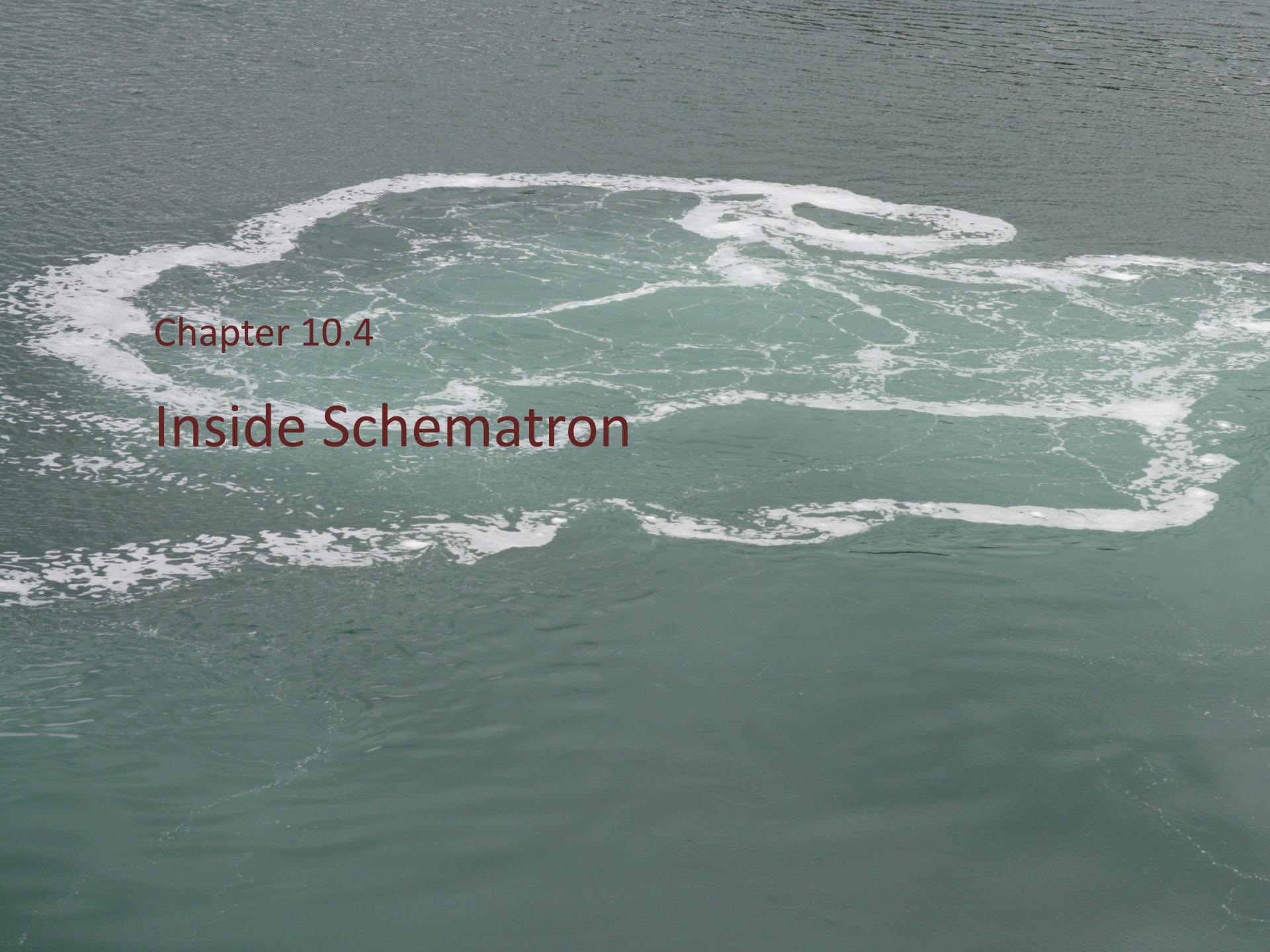
Schematron Assertions

- Phases
 - Optional, named groups of patterns
 - Can evaluate only the rules of specific patterns instead of evaluating all rules in all patterns
 - by specifying a phase id
 - Options for specifying the phase to evaluate include
 - command-line option
 - selection in a GUI
 - parameter in API call

Schematron Assertions

- Phases

```
<phase id="phase-id">
  <active pattern="pattern-id"/>
  ... more active elements go here ...
</phase>
```



Chapter 10.4

Inside Schematron

Inside Schematron

- Processing order
 - Only the order of rule elements is significant.
 - For each context node, only the first matching rule within a pattern is used.
 - The order of other things is implementation dependent
 - order in which context nodes are validated
 - order in which phases are evaluated
 - order in which patterns within a phase are evaluated
 - order in which asserts and reports within a rule are tested

Inside Schematron

Informatik · CAU Kiel

- Processing

- For each context node in the document being validated

- For each phase being evaluated

- For each pattern in the phase

if phases aren't used then
all patterns are evaluated

- Find the first rule that matches the context node

- For each assert and report in the rule

- Perform the test

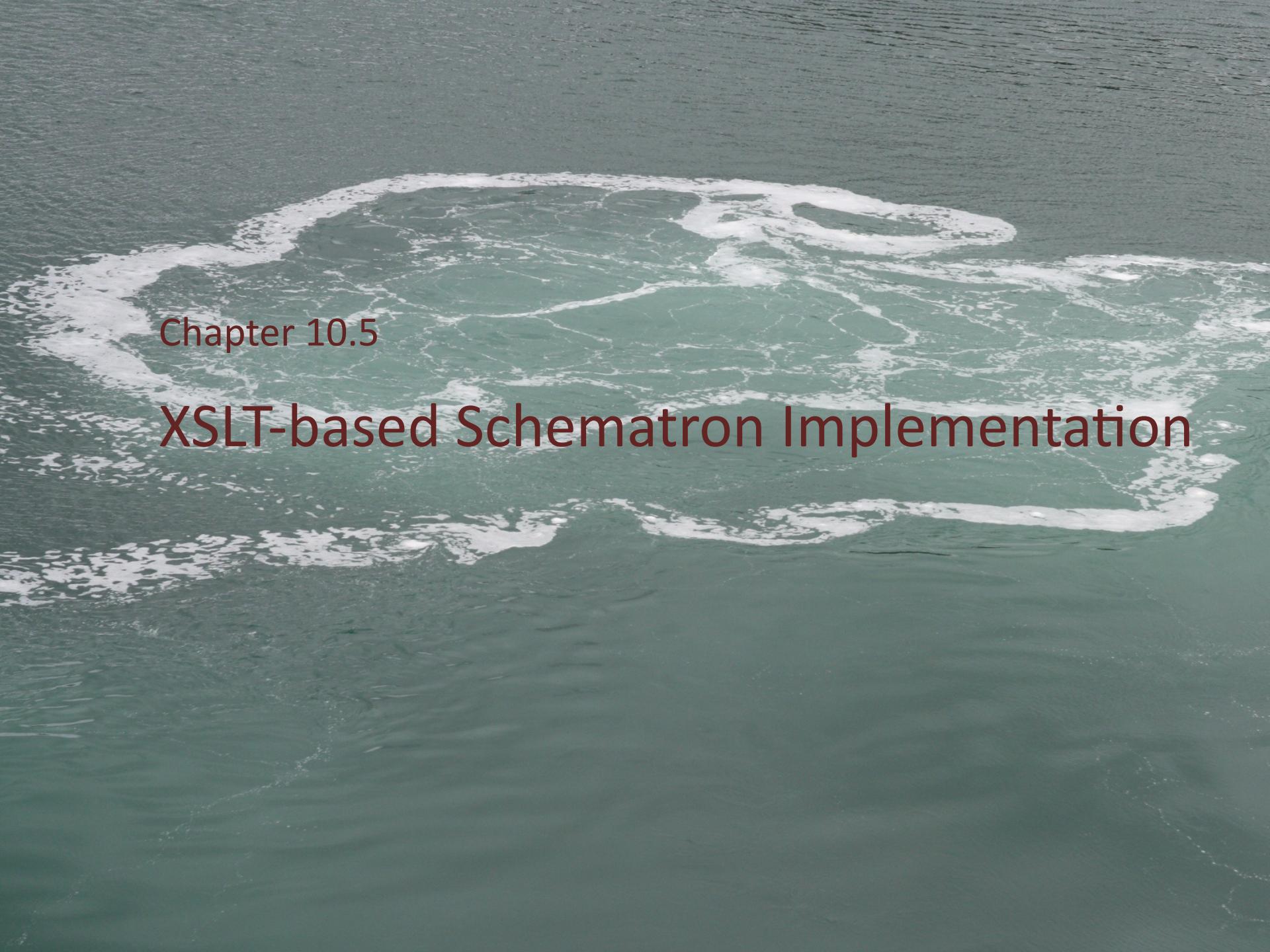
- If an assert test fails or a report test passes

- Output the message inside it

- If diagnostics are enabled and there are associated diagnostics

- Output the diagnostic messages

diagnostics provide information beyond
messages in assert and report elements
such as actual/expected values and
hints to repair the document

The background of the slide is a photograph of a large, circular oil spill or chemical spill on the surface of the ocean. The spill is white and contrasts sharply with the dark, textured green of the surrounding water. The spill has irregular edges and some internal patterns, suggesting it has been drifting for some time.

Chapter 10.5

XSLT-based Schematron Implementation

XSLT-based Schematron Implementation

Informatik · CAU Kiel

- Slides after:

Bob DuCharme:

Schematron 1.5: Looking Under the Hood.

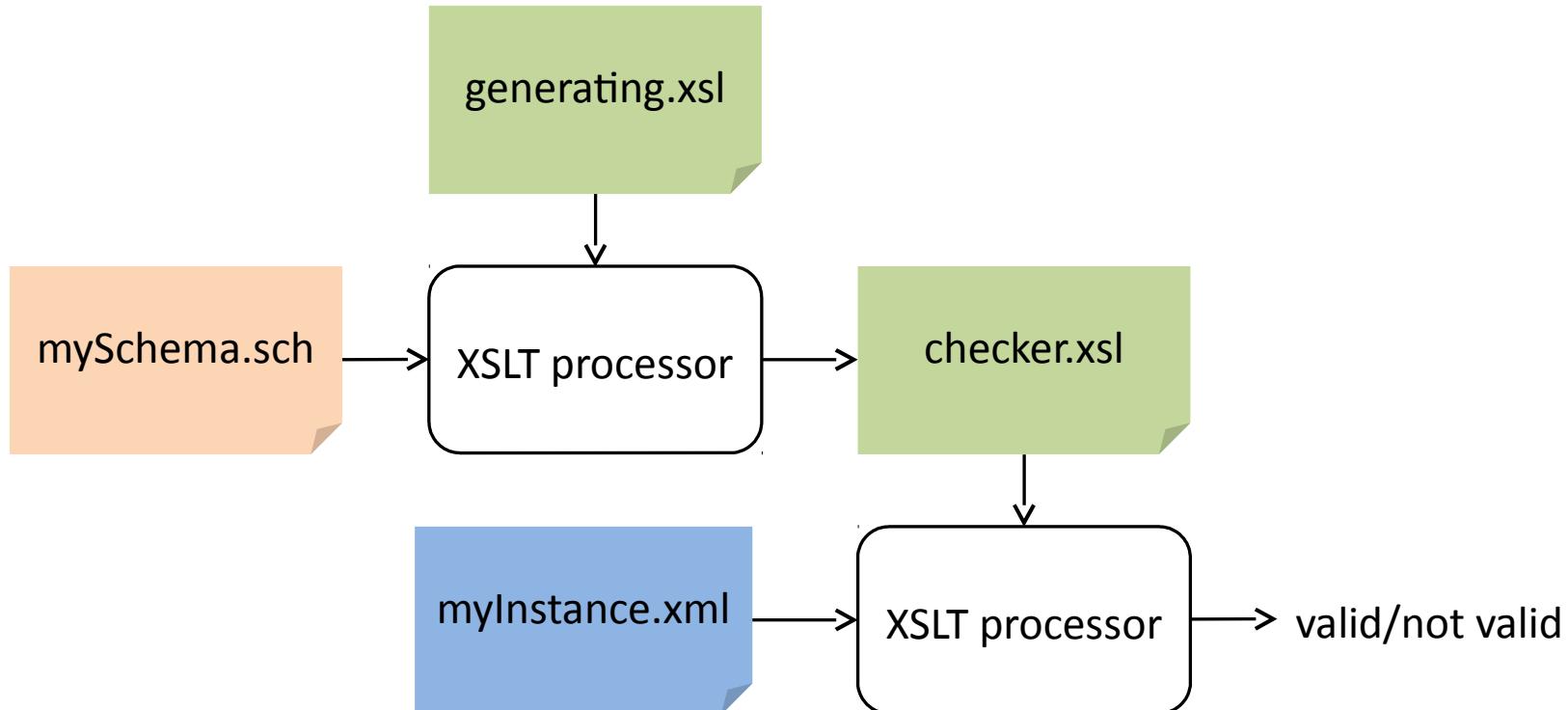
October 06, 2004

<http://www.xml.com/pub/a/2004/10/05/tr.html>

XSLT-based Schematron Implementation

Informatik · CAU Kiel

- XSLT-based implementation



XSLT-based Schematron Implementation

Informatik · CAU Kiel

- Problem
 - "If a stylesheet must add `xsl:template` elements to the result tree, you need a way to distinguish the `xsl:template` elements that tell the generating stylesheet what to do from the `xsl:template` elements that the generating stylesheet adds to the generated stylesheet in the result tree."
 - Concept:
 - declare a dummy namespace
 - include an `xsl:namespace-alias` element in the generating stylesheet

XSLT-based Schematron Implementation

Informatik · CAU Kiel

- What is namespace aliasing?
 - The `<xsl:namespace-alias>` element is used to replace a namespace in the style sheet to a different namespace in the output.
 - It must be a child node of `<xsl:stylesheet>` or `<xsl:transform>`.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:wxsl="http://www.w3schools.com/w3style.xsl">

  <xsl:namespace-alias stylesheet-prefix="wxsl" result-prefix="xsl"/>

  <xsl:template match="/">
    <wxsl:stylesheet>
      <xsl:apply-templates/>
    </wxsl:stylesheet>
  </xsl:template>

</xsl:stylesheet>
```

XSLT-based Schematron Implementation

- Core element

```
<xsl:template match="sch:assert | assert">
  <xsl:if test="not(@test)">
    <xsl:message>Markup Error: no test attribute in <assert></xsl:message>
  </xsl:if>
  <axsl:choose>
    <axsl:when test="{@test}" />
    <axsl:otherwise>
      <xsl:call-template name="process-assert">
        <xsl:with-param name="role" select="@role"/>
        <xsl:with-param name="id" select="@id"/>
        <xsl:with-param name="test" select="normalize-space(@test) " />
        <xsl:with-param name="icon" select="@icon"/>
        <xsl:with-param name="subject" select="@subject"/>
        <xsl:with-param name="diagnostics" select="@diagnostics"/>
      </xsl:call-template>
    </axsl:otherwise>
  </axsl:choose>
</xsl:template>
```

Look for assert elements
in the source tree

Do nothing.

axsl:choose element goes into the
result tree as xsl:choose element.

Specific template to handle
assert elements.

Web Services Introduction

Lecture "XML in Communication Systems"
Chapter 11

Dr.-Ing. Jesper Zedlitz
Research Group for Communication Systems
Dept. of Computer Science
Christian-Albrechts-University in Kiel



Recommended Reading

Informatik · CAU Kiel

- Booth, D. et al.:
Web Services Architecture
<http://www.w3.org/TR/ws-arch/>
- Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.:
Web Services Description Language (WSDL) 1.1
<http://www.w3.org/TR/wsdl>
- Booth, D., Liu, C.K.:
Web Services Description Language (WSDL) Version 2.0 Part 0: Primer
<http://www.w3.org/TR/wsdl20-primer>
- Ballinger, K., Ehnebuske, D., Gudgin, M., Nottingham, M., Yendluri, P.:
Web Services Interoperability Organization Basic Profile Version 1.0
<http://www.ws-i.org/Profiles/BasicProfile-1.0.html>

Introduction

- Terminology
 - Intranet:
 - operated by a single organisation,
 - applies Internet technologies (TCP/IP, SMTP, http, ...),
 - possibly consists of several interrelated local area networks,
 - provides information services for business processes

Introduction

- Terminology (cont'd.)
 - Extranet:
 - enables sharing part of an organization's business processes with suppliers, vendors, partners, customers, etc.
 - extends company's intranet to users outside the company,
 - provides services offered by one company to a group of other companies.
 - "loose coupling"

Introduction

- Distributed Object Infrastructures
 - Distributed object infrastructure
= component technology + wire protocol
 - Current distributed object infrastructures:
COM/DCOM, Java RMI/JRMP, CORBA/GIOP
 - Disadvantages of proprietary distributed object infrastructures
 - Lack of interoperability: Vendor- and/or platform specific
 - Administrative costs:
Custom runtime environment, configuring firewalls

Introduction

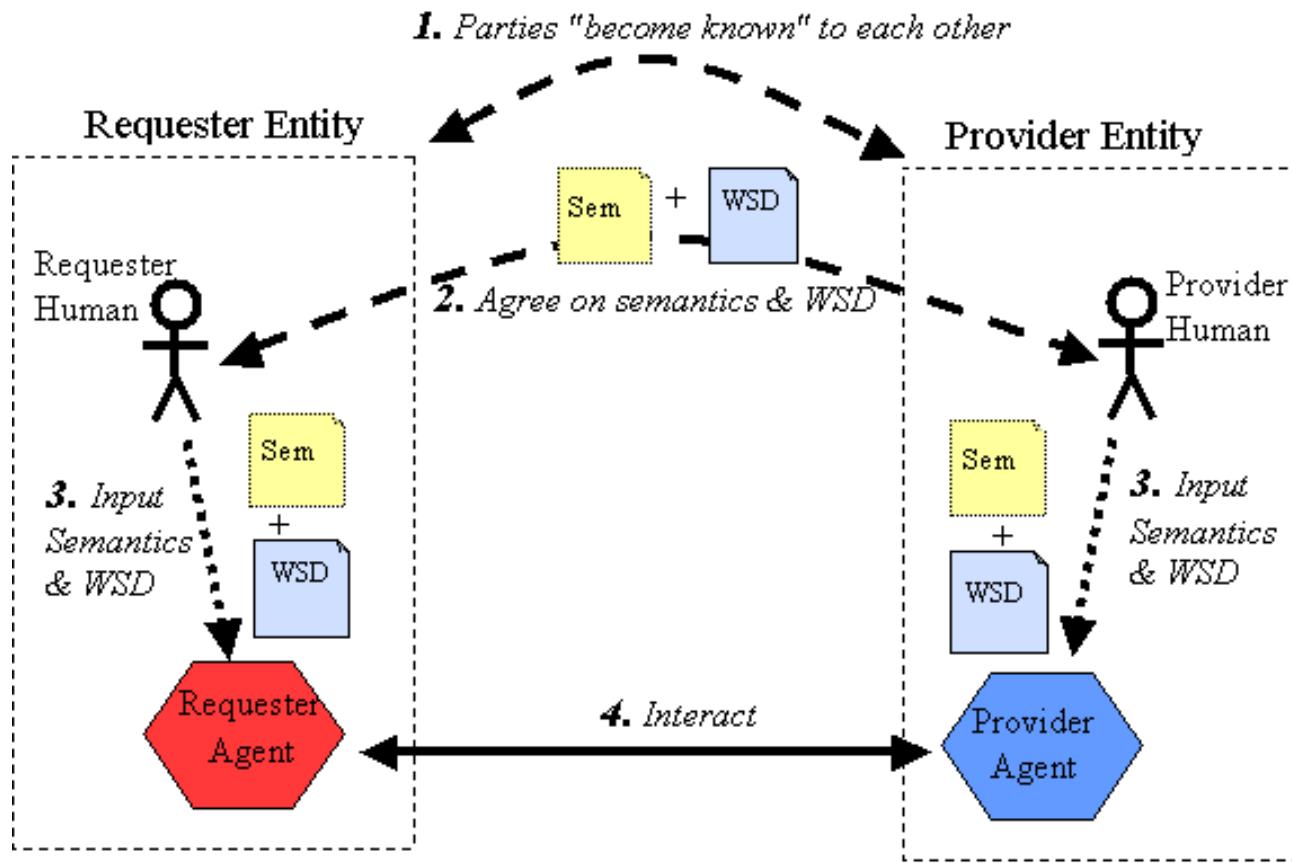
- Web Service definition

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL).

Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Introduction

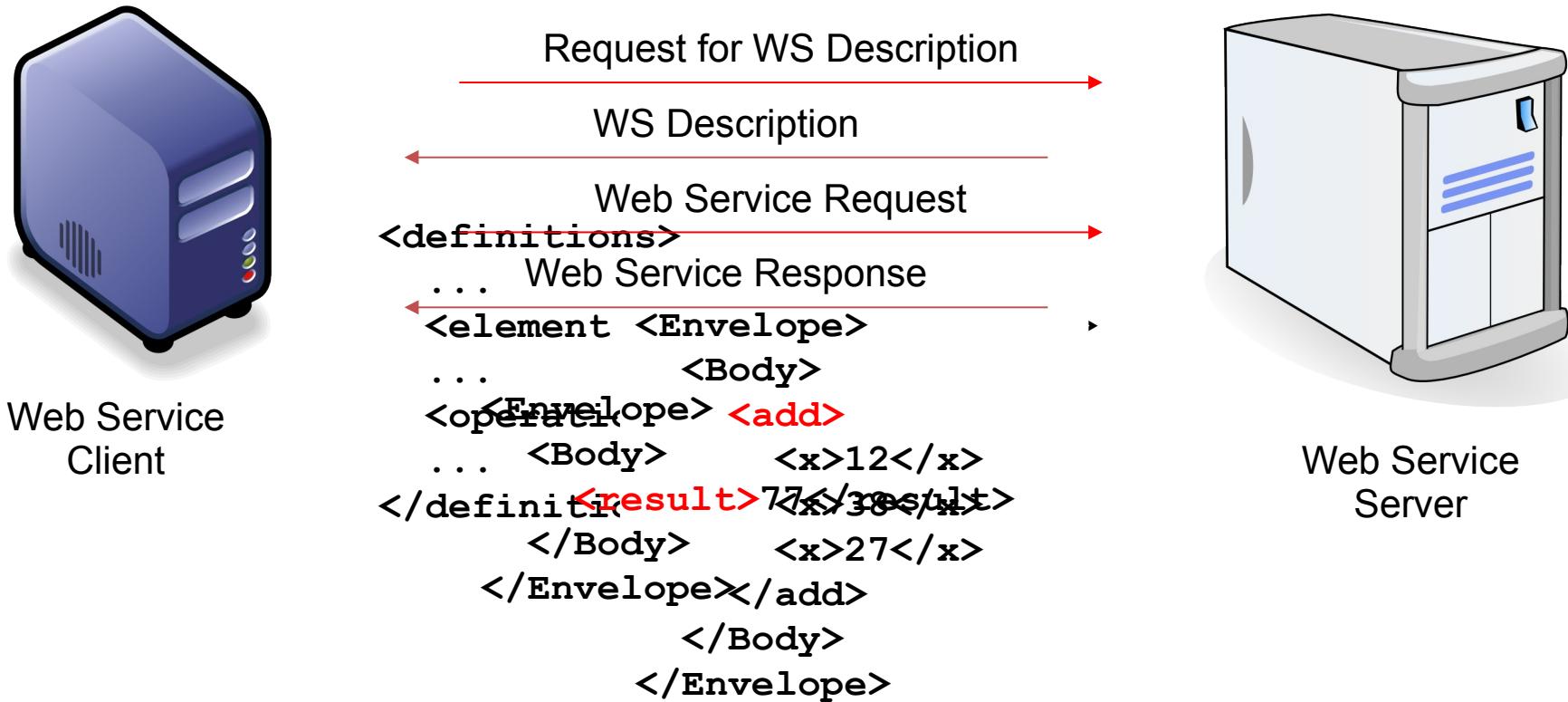
- "Engaging a Web Service"



from: Web Services Architecture
W3C Working Group Note 11 February 2004

Introduction

- WS basic operation



Web Service
Client

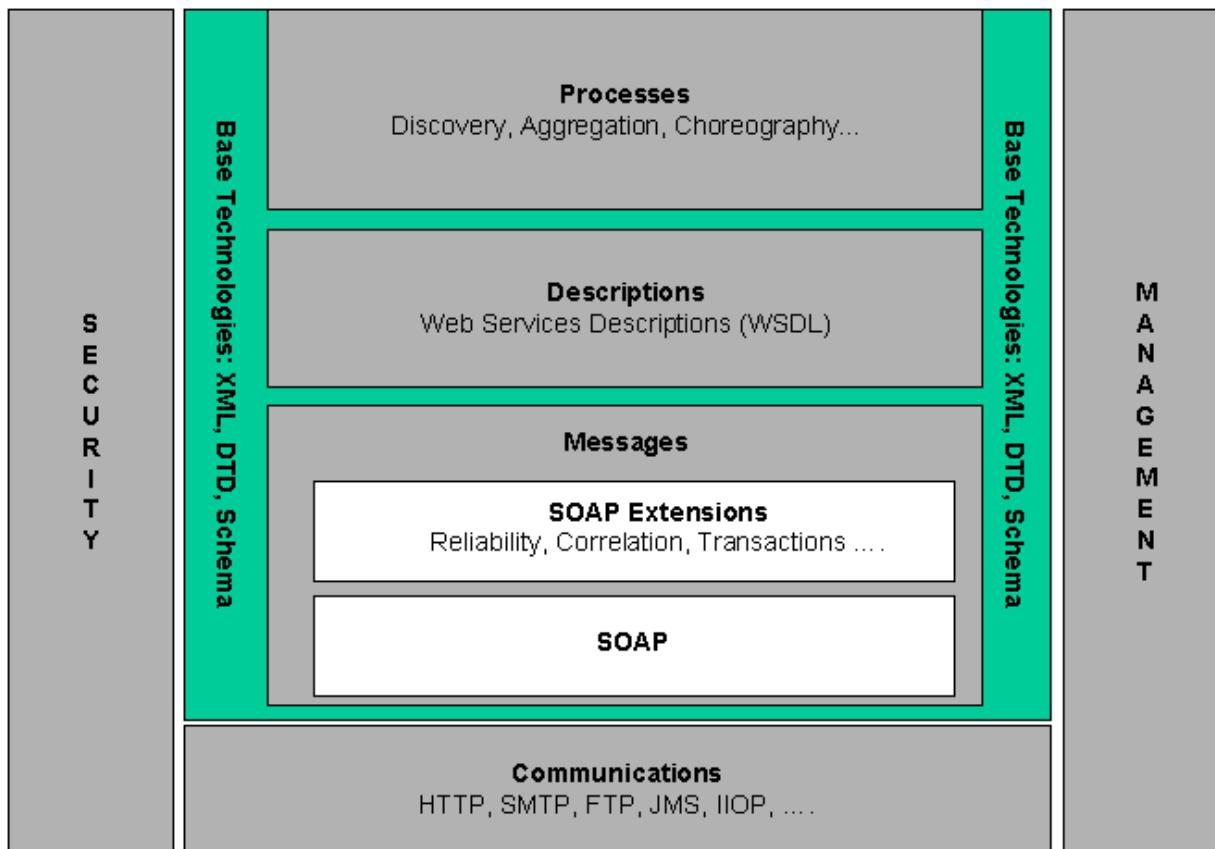
Web Service
Server

Introduction

- Web Services: Core element of Service-Oriented Architecture (SOA)
 - Middleware for distributed systems:
Independent of OS or programming language
 - XML-encoded messages in
XML-based "framing protocol": SOAP
 - Formal interface description:
Web Service Description Language (WSDL)

Introduction

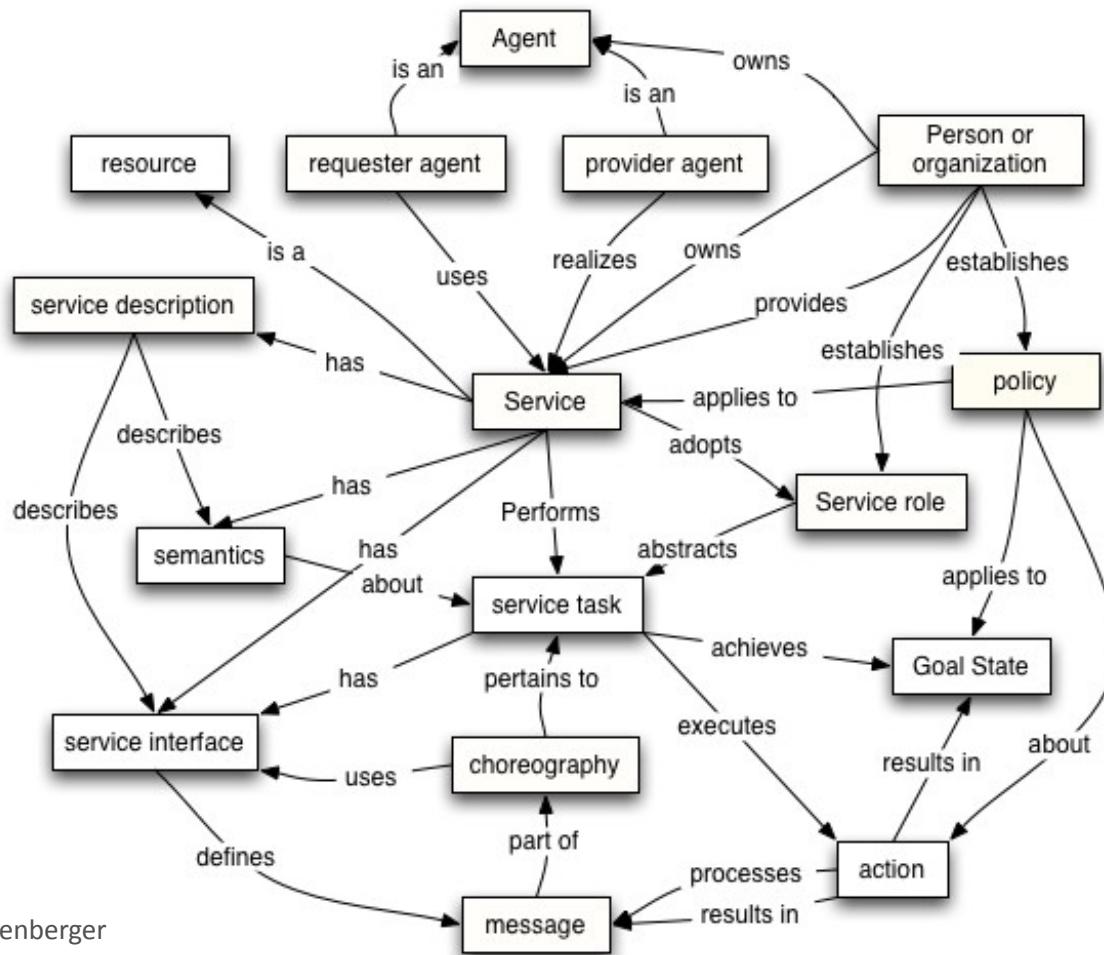
- "Big picture"



from: Web Services Architecture
W3C Working Group Note 11 February 2004

Introduction

- Service-oriented model



from: Web Services Architecture
W3C Working Group Note 11 February 2004

Web Services

SOAP

Lecture "XML in Communication Systems"
Chapter 12

Prof. Dr.-Ing. Norbert Luttenberger
Research Group for Communication Systems
Dept. of Computer Science
Christian-Albrechts-University in Kiel



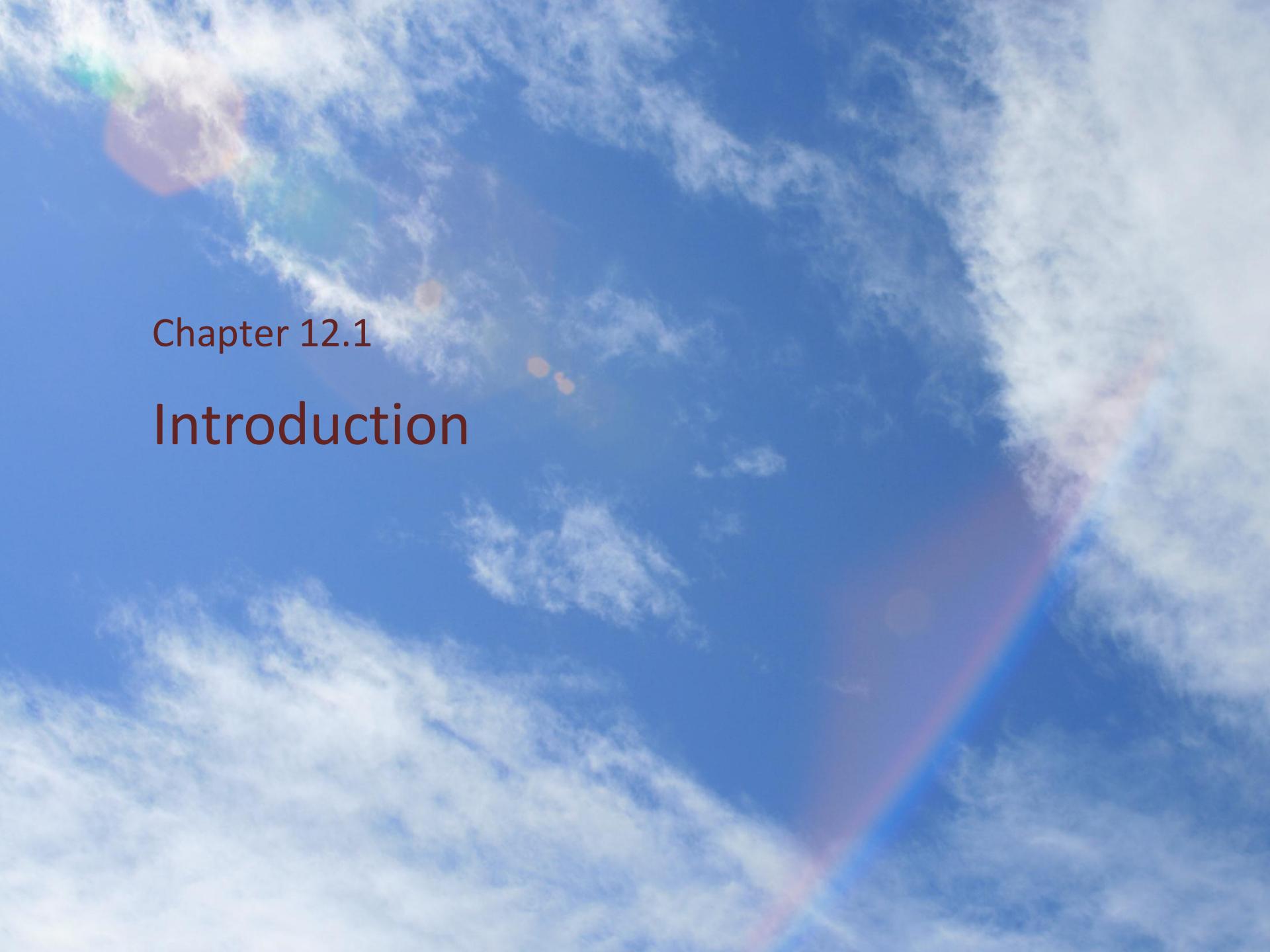
Recommended Reading

- Booth, D. et al.:
Web Services Architecture
<http://www.w3.org/TR/ws-arch/>
- Nilo Mitra, Yves Lafon (eds.):
SOAP Version 1.2 Part 0: Primer (Second Edition)
W3C Recommendation 27 April 2007
<http://www.w3.org/TR/soap12-part0/>
- Ballinger, K., Ehnebuske, D., Gudgin, M., Nottingham, M., Yendluri, P.:
Web Services Interoperability Organization Basic Profile Version 1.0
<http://www.ws-i.org/Profiles/BasicProfile-1.0.html>
- Papazoglou, M.:
Web Services: Principles and Technology.
Pearson Education Ltd. 2008

Overview

Informatik · CAU Kiel

1. Introduction
2. SOAP Processing
3. SOAP Transport Binding
4. SOAP Message Format
5. Remote Procedure Calls



Chapter 12.1

Introduction

Introduction

- Former name: *Simple Object Access Protocol*
 - protocol for exchange of structured and typed data
 - between peers in a distributed environment
 - features:
 - XML message coding
 - stateless request/response communication
 - SOAP endpoints identified by URL
 - SOAP binding: definitions for transport of SOAP messages over different Internet protocols

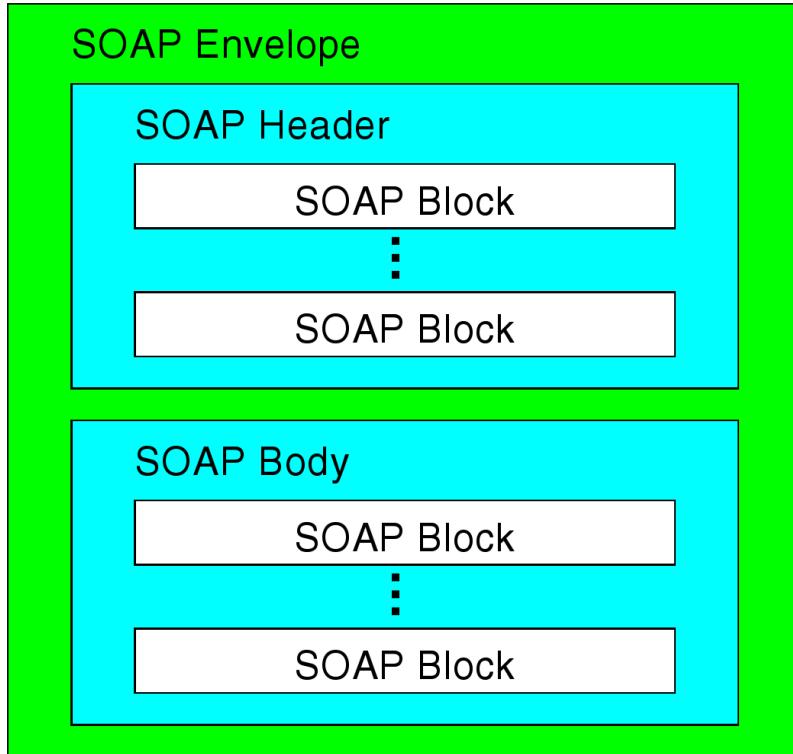
Introduction

- Former name: *Simple Object Access Protocol* (cont'd.)
 - No explicit programming model, unlike DCOM and CORBA:
no special components or tools needed to make an implementation
 - Can be implemented in any language and OS (Java, Perl, C++, VB, Windows, UNIX)
 - SOAP defines two types of messages:
 - Requests
 - Responses

Introduction

- Former name: *Simple Object Access Protocol* (cont'd.)
 - versions: SOAP 1.1 (2000) and SOAP 1.2 (2003)
 - namespaces for SOAP 1.1
 - <http://schemas.xmlsoap.org/soap/envelope/>
 - <http://schemas.xmlsoap.org/soap/encoding/>
 - namespaces for SOAP 1.2
 - <http://www.w3.org/2003/05/soap-envelope>
 - <http://www.w3.org/2003/05/soap-encoding>
 - <http://www.w3.org/2003/05/soap-rpc>
 - ...

Introduction



elements

- Envelope
- Header
- Body
- Fault

attributes

- actor (SOAP 1.1) / role (SOAP 1.2)
- mustUnderstand
- encodingStyle
- relay (SOAP 1.2)

Introduction

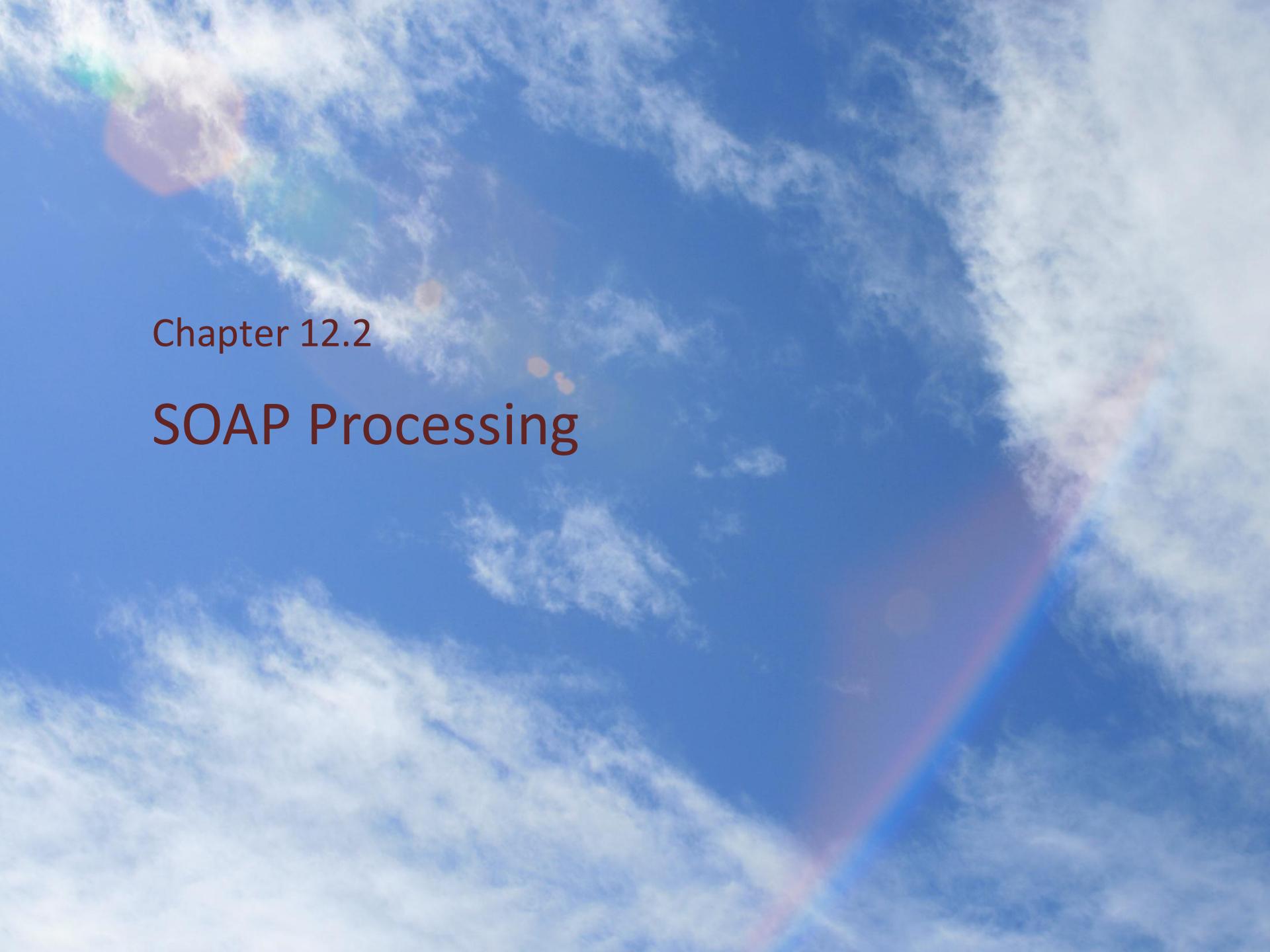
- Sample SOAP message (from the SOAP 1.2 standard)

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation
      xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
      <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
    </m:reservation>
    <n:passenger
      xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <n:name>Åke Jógvind</n:name>
    </n:passenger>
  </env:Header>
```

Introduction

- Sample SOAP message (from the SOAP 1.2 standard)

```
<env:Body>
  <p:itinerary
    xmlns:p="http://travelcompany.example.org/reservation/travel">
    <p:departure>
      <p:departing>New York</p:departing>
      <p:arriving>Los Angeles</p:arriving>
      <p:departureDate>2001-12-14</p:departureDate>
      <p:departureTime>late afternoon</p:departureTime>
      <p:seatPreference>aisle</p:seatPreference>
    </p:departure>
    <p:return> ... </p:return>
  </p:itinerary>
  <q:lodging
    xmlns:q="http://travelcompany.example.org/reservation/hotels">
    <q:preference>none</q:preference>
  </q:lodging>
  </env:Body>
</env:Envelope>
```

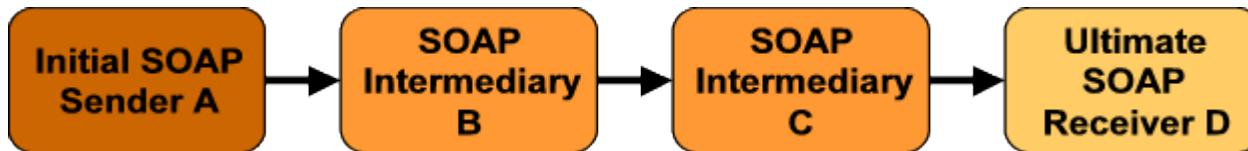


Chapter 12.2

SOAP Processing

SOAP Processing

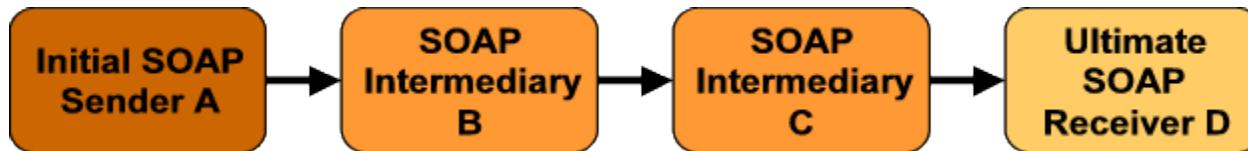
- SOAP Nodes
 - are sources and sinks of SOAP (request/response) messages, SOAP faults
 - different types of SOAP nodes
 - Initial SOAP sender
 - Ultimate SOAP Receiver
 - SOAP Intermediary



Henrik Frystyk Nielsen: SOAP Message Path Modeling.
W3C Workshop on Web Services, 11-12 April 2001, San Jose, CA - USA

SOAP Processing

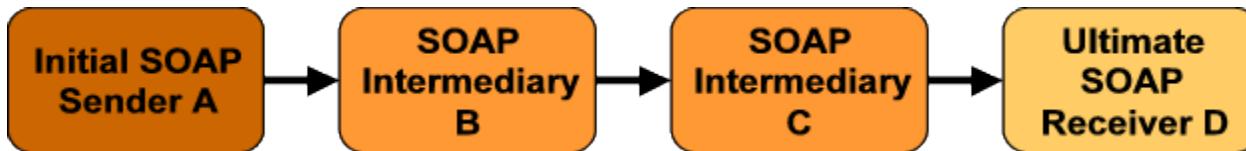
- Message path



- "A SOAP message generated by A can indicate which part of a message is for B, C and D. However, it cannot indicate that the intermediaries are to be organized into a message path ... "
- "The SOAP targeting model is ... a decentralized message-processing concept rather than a mechanism for transferring SOAP messages across the Web."

SOAP Processing

- SOAP Intermediaries



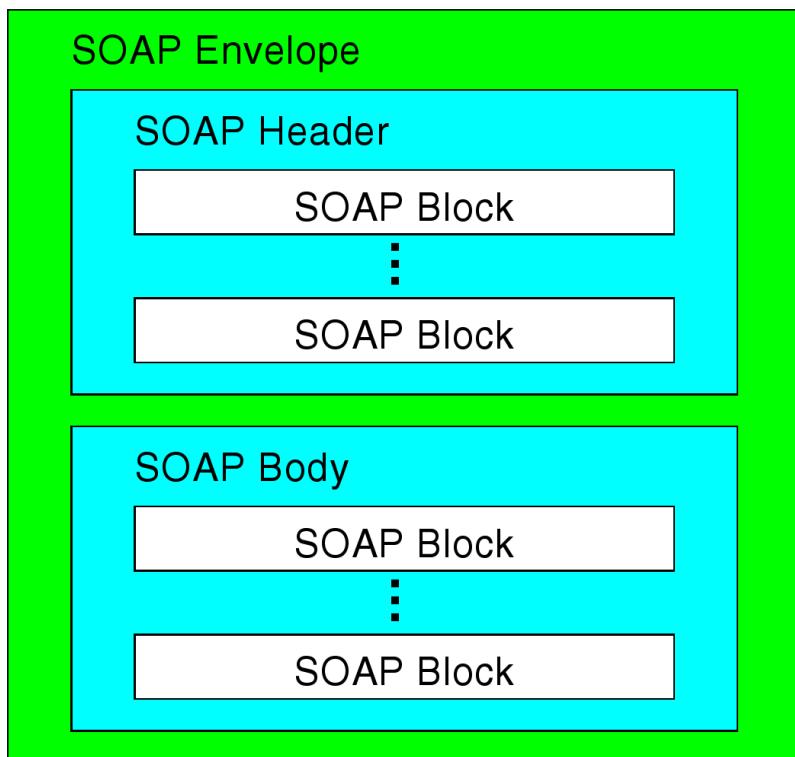
- processing intermediaries deal with the message using the same rules as any other SOAP receiver and/or sender
- protocol intermediaries don't take any part in the SOAP message path other than acting as a relay at the underlying protocol level.
- assignment of roles to SOAP nodes

SOAP Processing

- An aside: RFC-2616 (HTTP 1.1, 1999) on other kinds of intermediaries
 - A **proxy** is a forwarding agent, receiving requests for a URI in its absolute form, **rewriting** all or part of the message, and forwarding the reformatted request toward the server identified by the URI.
 - A **gateway** is a receiving agent, acting as a layer above some other server(s) and, if necessary, **translating** the requests to the underlying server's protocol.
 - A **tunnel** acts as a relay point between two connections **without changing the messages**; tunnels are used when the communication needs to pass through an intermediary (such as a firewall) even when the intermediary cannot understand the contents of the messages.

SOAP Processing

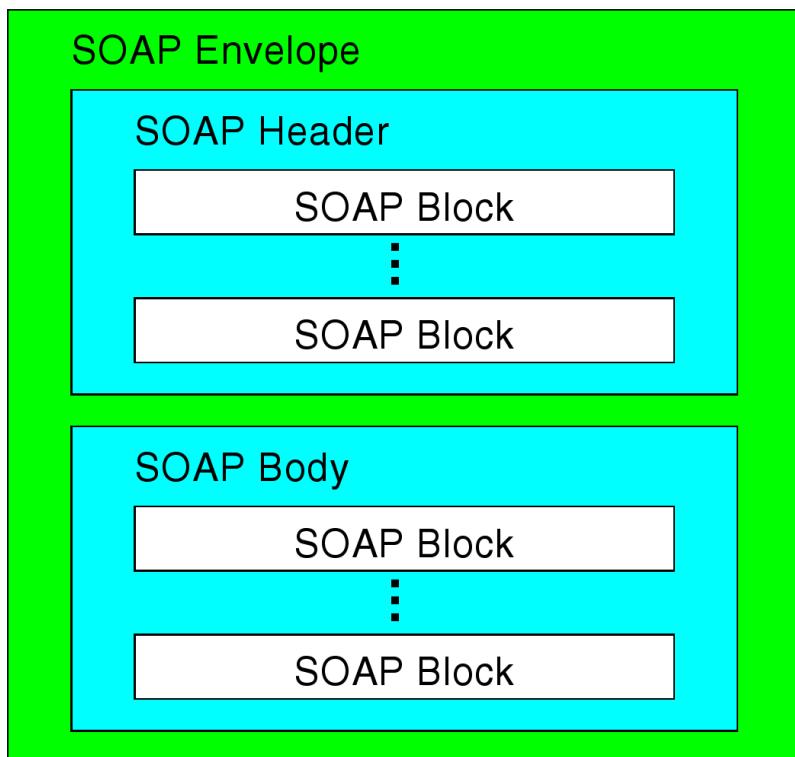
- Header block processing



- determined by the "role" a SOAP node adopts
- SOAP Header blocks may carry a related
 - `env:actor` (SOAP 1.1) resp.
 - `env:role` (SOAP 1.2) attribute (optional)

SOAP Processing

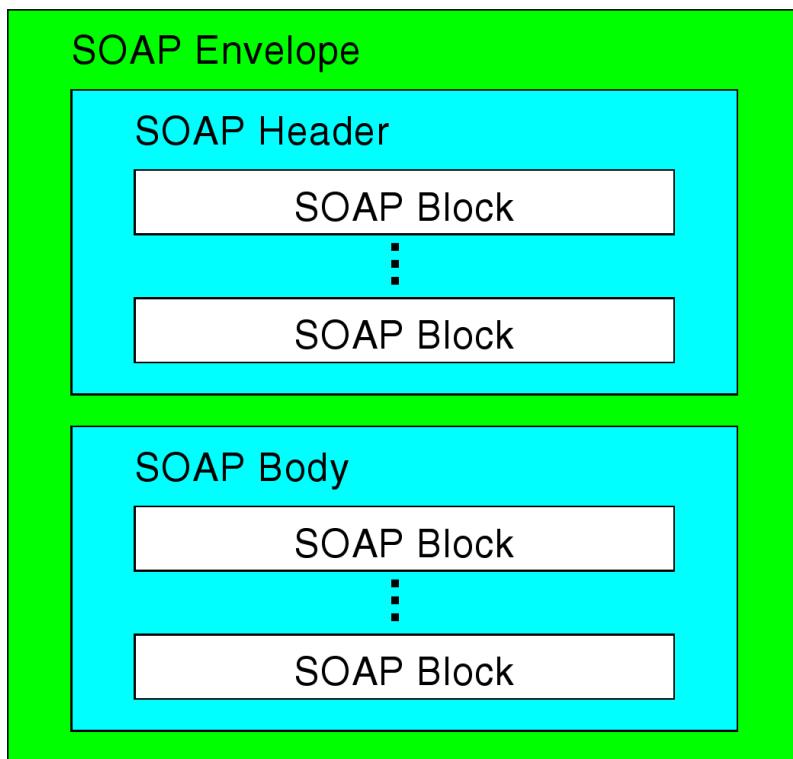
- Header block processing (cont'd.)



- predefined roles (SOAP 1.2):
 - `next`
for both all SOAP
Intermediaries and the
Ultimate SOAP Receiver
 - `none`
no node in this message path
 - `ultimateReceiver`
for the Ultimate SOAP
Receiver

SOAP Processing

- Header block processing (cont'd.)

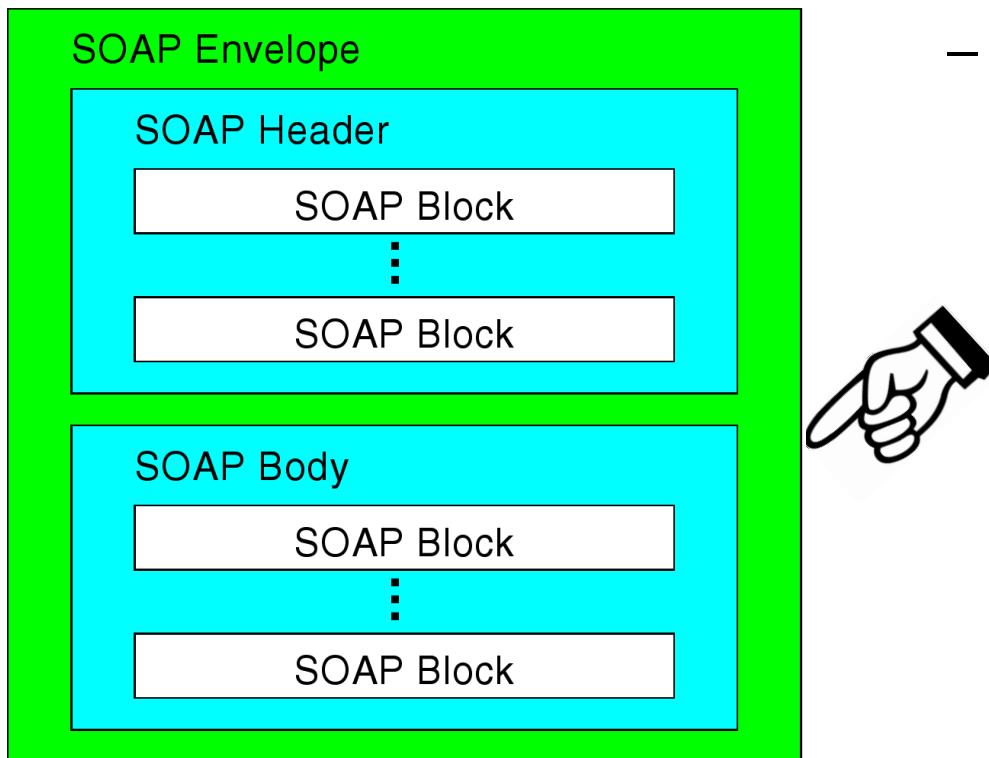


- Optional attribute
env:mustUnderstand
→ block must be processed!
 - SOAP fault has to be sent, if block cannot be processed.

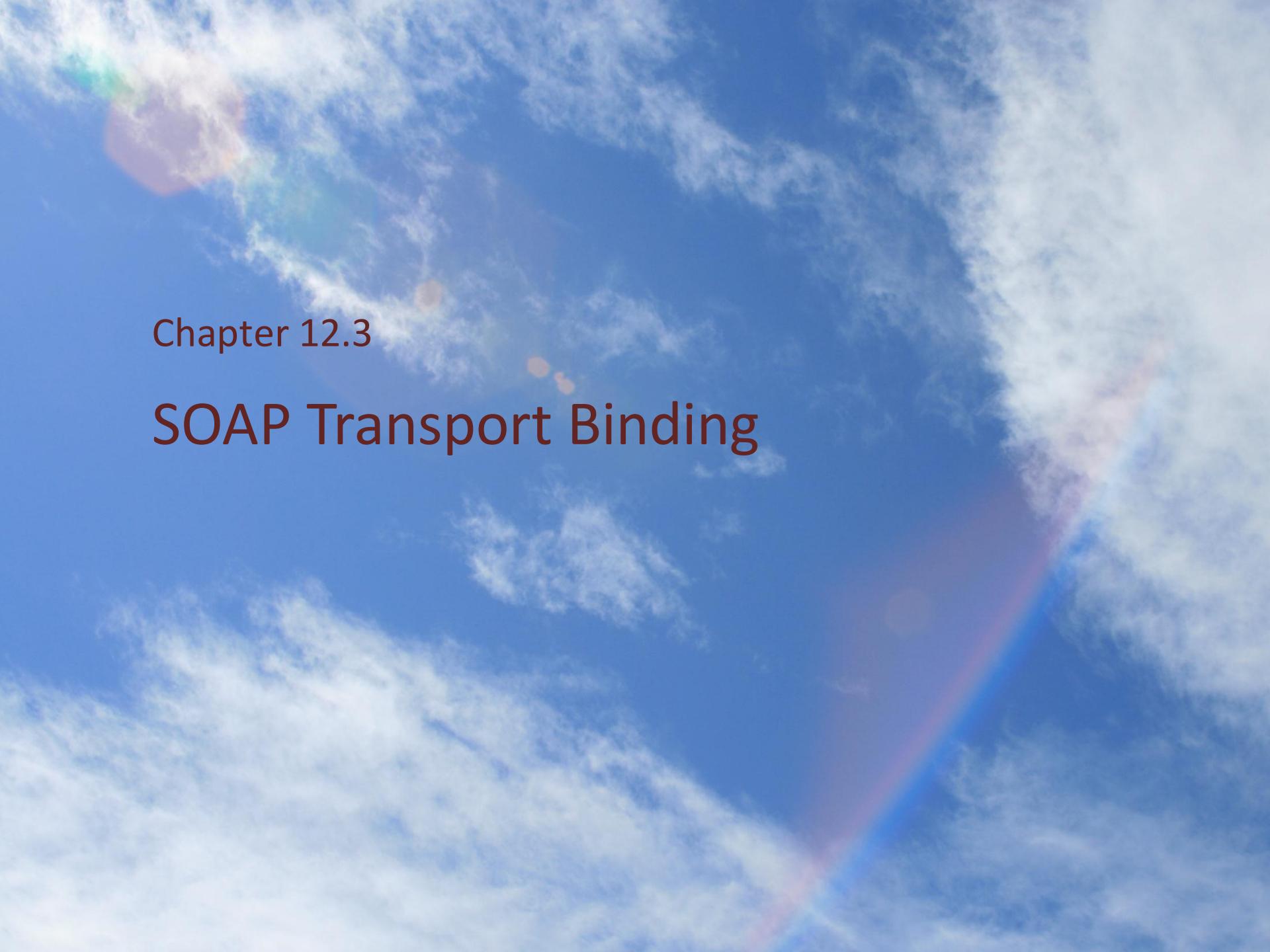
SOAP Processing

Informatik · CAU Kiel

- Body block processing



- Ultimate SOAP Receiver must process SOAP Body.



Chapter 12.3

SOAP Transport Binding

SOAP Transport over HTTP

Informatik · CAU Kiel

- SOAP HTTP Binding
 - SOAP in Request-Response mode
 - HTTP Request carries SOAP Request and
 - HTTP Response carries SOAP Response.
 - HTTP features
 - method: POST
 - Content-Type: text/xml (SOAP 1.1) or
 application/soap+xml (SOAP 1.2)
 - SOAP 1.1: additional HTTP header line SOAPAction or
SOAP 1.2: action parameter in the Content-Type (RFC-3902)

SOAP Transport over HTTP

Informatik · CAU Kiel

```
POST /Service/Stockquotes HTTP/1.0
Content-Type: text/xml
SOAPAction: "urn:getQuote"

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Body>
    <ns:getQuote xmlns:ns="urn:example-delayed-quotes">
      <ns:symbol>NYSE:IBM</ns:symbol>
    </ns:getQuote>
  </env:Body>
</env:Envelope>
```

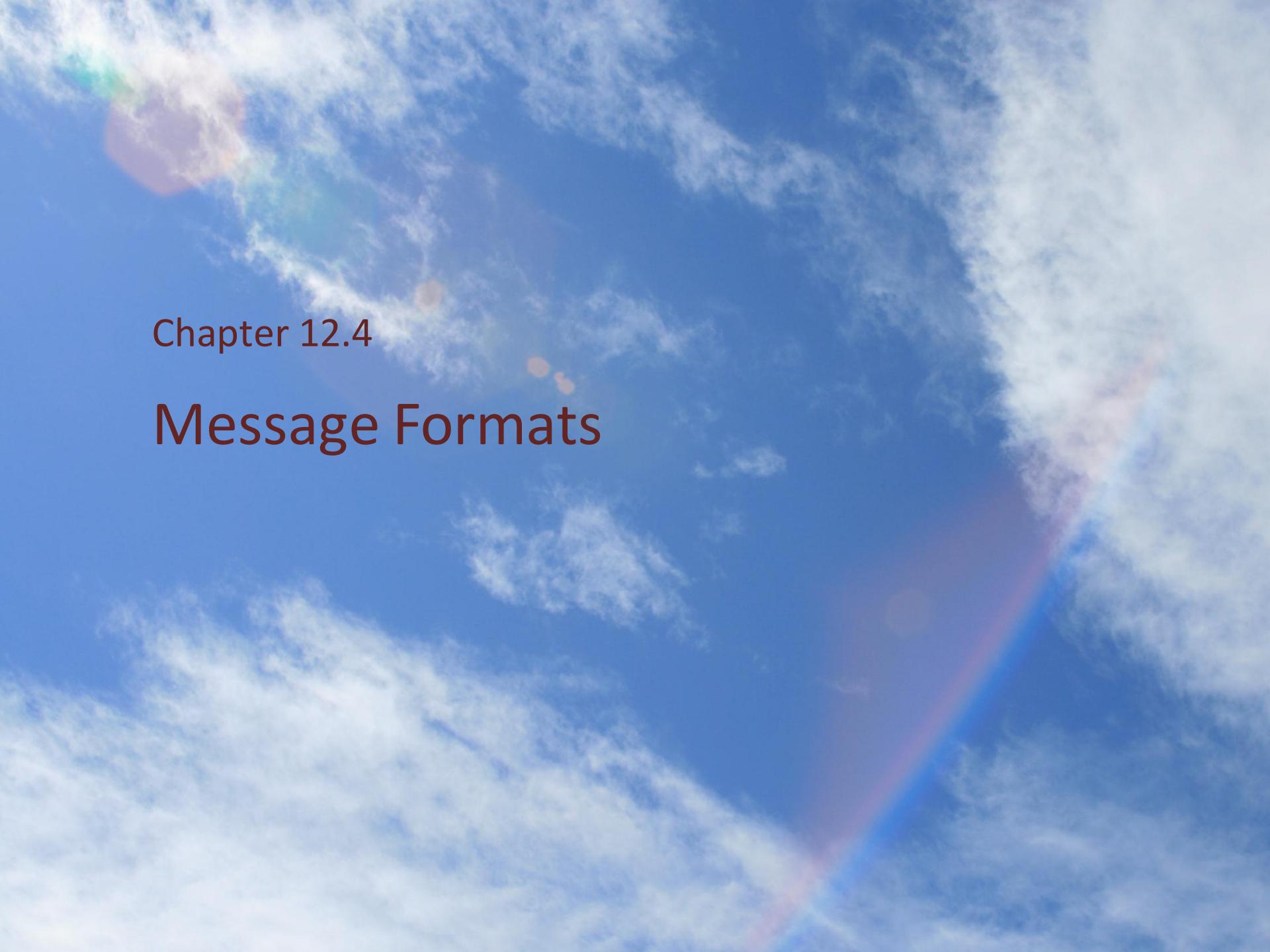
SOAP Transport over HTTP

Informatik · CAU Kiel

```
HTTP/1.0 200 OK
```

```
Content-Type: text/xml
```

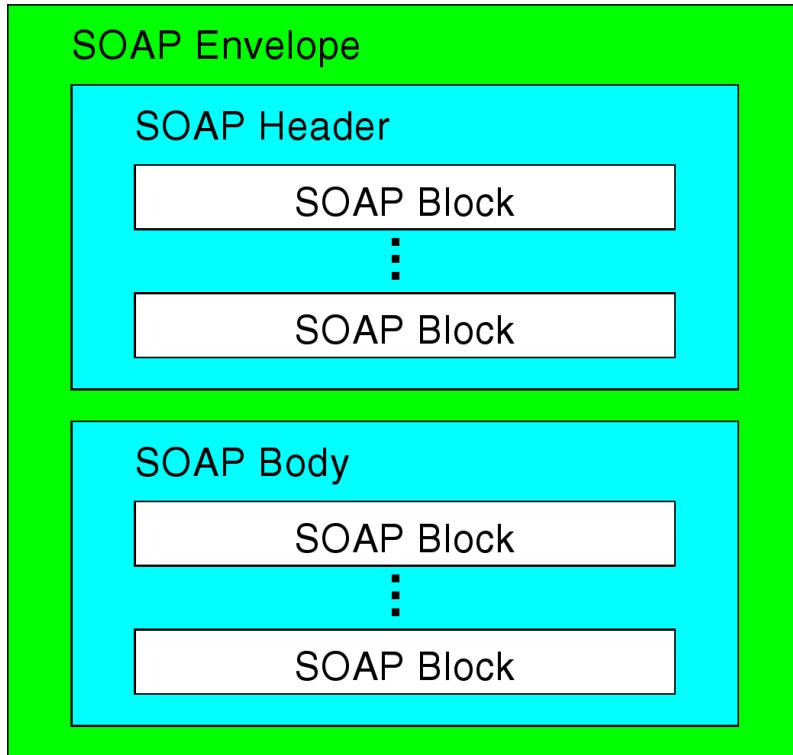
```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Body>
    <ns:getQuoteResponse xmlns:ns="urn:example-delayed-quotes">
      <ns:result>91.35</ns:result>
    </ns:getQuoteResponse>
  </env:Body>
</env:Envelope>
```



Chapter 12.4

Message Formats

Message Formats



elements

- Envelope
- Header
- Body
- Fault

attributes

- actor (SOAP 1.1) / role (SOAP 1.2)
- mustUnderstand
- encodingStyle
- relay (SOAP 1.2)

Message Formats

- SOAP header
 - is an optional component
 - has information about how the message is to be processed
 - can contain extensions to the message like transaction ids
 - can also contain security information
- SOAP body
 - contains the message referred to as "payload"
 - can contain the encodingStyle attribute
 - can also contain a <Fault> element

SOAP Message Format

- **SOAP message**

A SOAP message is the basic unit of communication between peer SOAP nodes.

- **SOAP envelope**

The outermost syntactic construct or structure of a SOAP message defined by SOAP within which all other syntactic elements of the message are enclosed.

SOAP Message Format

- **SOAP header**

A collection of zero or more SOAP blocks which may be targeted at any SOAP receiver within the SOAP message path.

- **SOAP body**

A collection of zero or more SOAP blocks targeted at the ultimate SOAP receiver within the SOAP message path.

- **SOAP fault**

A special SOAP block which contains fault information generated by a SOAP node.

SOAP Message Format

- **SOAP block**

A syntactic construct or structure used to delimit data that logically constitutes a single computational unit as seen by a SOAP node. The type of a SOAP block is identified by the fully qualified name of the outer element for the block, which consists of the namespace URI and the local name. A block encapsulated within the SOAP header is called a header block and a block encapsulated within a SOAP body is called a body block.

SOAP Message Format

- **Formal definitions**

The **document** element information item has:

- A local name of **Envelope**
- Zero or more namespace qualified attribute information items
- One or two element information item children in order as follows:
 - An optional **Header** element information item
 - A mandatory **Body** element information item.

SOAP Message Format

- Formal definitions (cont'd.)

The **Header** element information item has:

- A local name of Header
- Zero or more namespace qualified attribute information item children
- Zero or more namespace qualified element information item children
- Each SOAP header block element information item
 - MUST be namespace qualified
 - MAY have an encodingStyle attribute information item
 - MAY have an actor / role attribute information item
 - MAY have a mustUnderstand attribute information item

SOAP Message Format

- Formal definitions (cont'd.)

The **Body** element information item has:

- A local name of Body
- Zero or more namespace qualified element information item children
- Each SOAP header block element information item
 - MUST be namespace qualified;
 - MAY have an `encodingStyle` attribute information item

The namespaces for all Envelope elements are:

- `http://schemas.xmlsoap.org/soap/envelope/` (SOAP 1.1)
- `http://www.w3.org/2003/05/soap-envelope` (SOAP 1.2)

SOAP Message Format

- Sample SOAP fault (SOAP 1.1)

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Body>
    <env:Fault>
      <faultcode>env:Server</faultcode>
      <faultstring>Server Error</faultstring>
      <detail>My application didn't work</detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

SOAP Message Format

- Sample SOAP fault (SOAP 1.2)

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
      </env:Code>
      <env:Reason>
        <env:Text>Sender Timeout</env:Text>
      </env:Reason>
      <env:Detail>Have waited 5 minutes</env:Detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

SOAP Message Format

- Fault-Codes
 - *VersionMismatch*:
invalid SOAP Envelope
 - *MustUnderstand*:
A mustUnderstand header block could not be processed
 - *DataEncodingUnknown* (nur SOAP 1.2):
message encoding not supported
 - *Client* (SOAP 1.1) / *Sender* (SOAP 1.2):
message invalid
 - *Server* (SOAP 1.1) / *Receiver* (SOAP 1.2):
server-side error

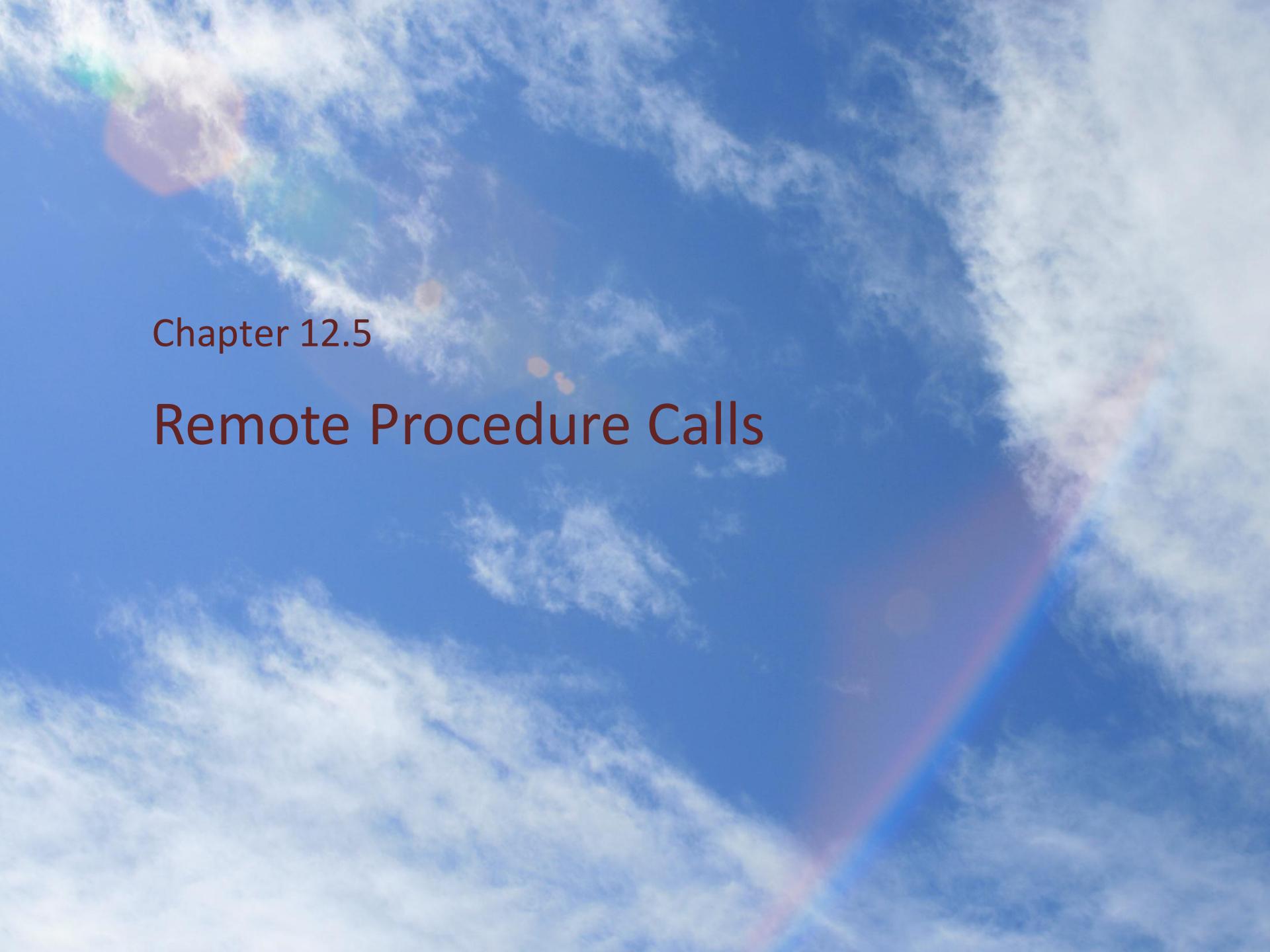
SOAP Message Format

- Message Encoding
 - given in the encodingStyle attribute (optional)
 - Mostly: literal encoding
 - XML serialization
 - recommended by WS-I

SOAP Message Format

- Another example

```
<env:Envelope xmlns:env="http://www.w3.org/2001/09/soap-envelope">
  <env:Header>
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Pick up Mary at school at 2pm</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```



Chapter 12.5

Remote Procedure Calls

Remote Procedure Calls

- Messages are not a "natural" programming model
 - require that programmer worry about message formats
 - must be packed and unpacked
 - have to be decoded by client and server
 - often asynchronous
 - may require special error handling functions

Remote Procedure Calls

- Procedure call: a more "natural" way to communicate
 - every language supports it
 - semantics are well defined and understood
 - natural for programmers to use
- Basic idea:
 - Let's just define a server as a module.
 - Let the server export a set of procedures that can be called by clients.
 - Clients just do procedure calls.

Remote Procedure Calls

- Issues
 - How do we make **RPCs** invisible to the programmer?
 - What are the semantics of parameter passing?
 - How is binding done (locating the server)?
 - How do we support heterogeneity (OS, language)?

Remote Procedure Calls

- The basic model for Remote Procedure Call (RPC)
 - described by Birrell and Nelson from Xerox PARC in 1984:
A. D. Birrell and B. J. Nelson: *Implementing remote procedure calls.*
ACM Transactions on Computer Systems, 2(1):39–59, February 1984.
 - goals: make RPC look as much like local PC as possible
 - used OS/language support

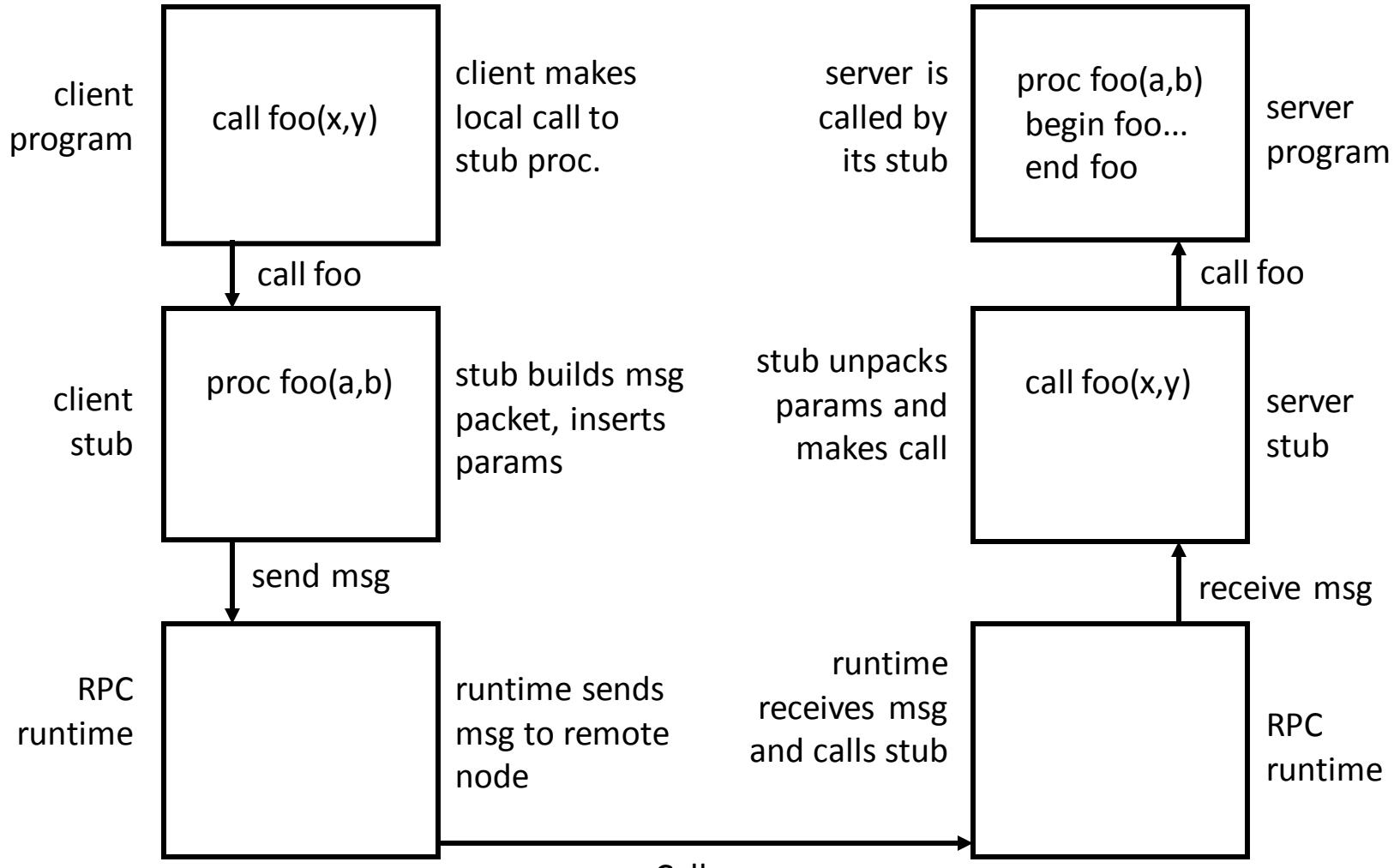
Remote Procedure Calls

- Implementation: 3 components on each side
 - a user program (client or server)
 - a set of **stub** procedures
 - client-side stub appears to the client code as if it were a callable procedure
 - server-side stub appears to the server code as a calling client
 - stubs send messages to each other to make the RPC happen
 - RPC runtime support

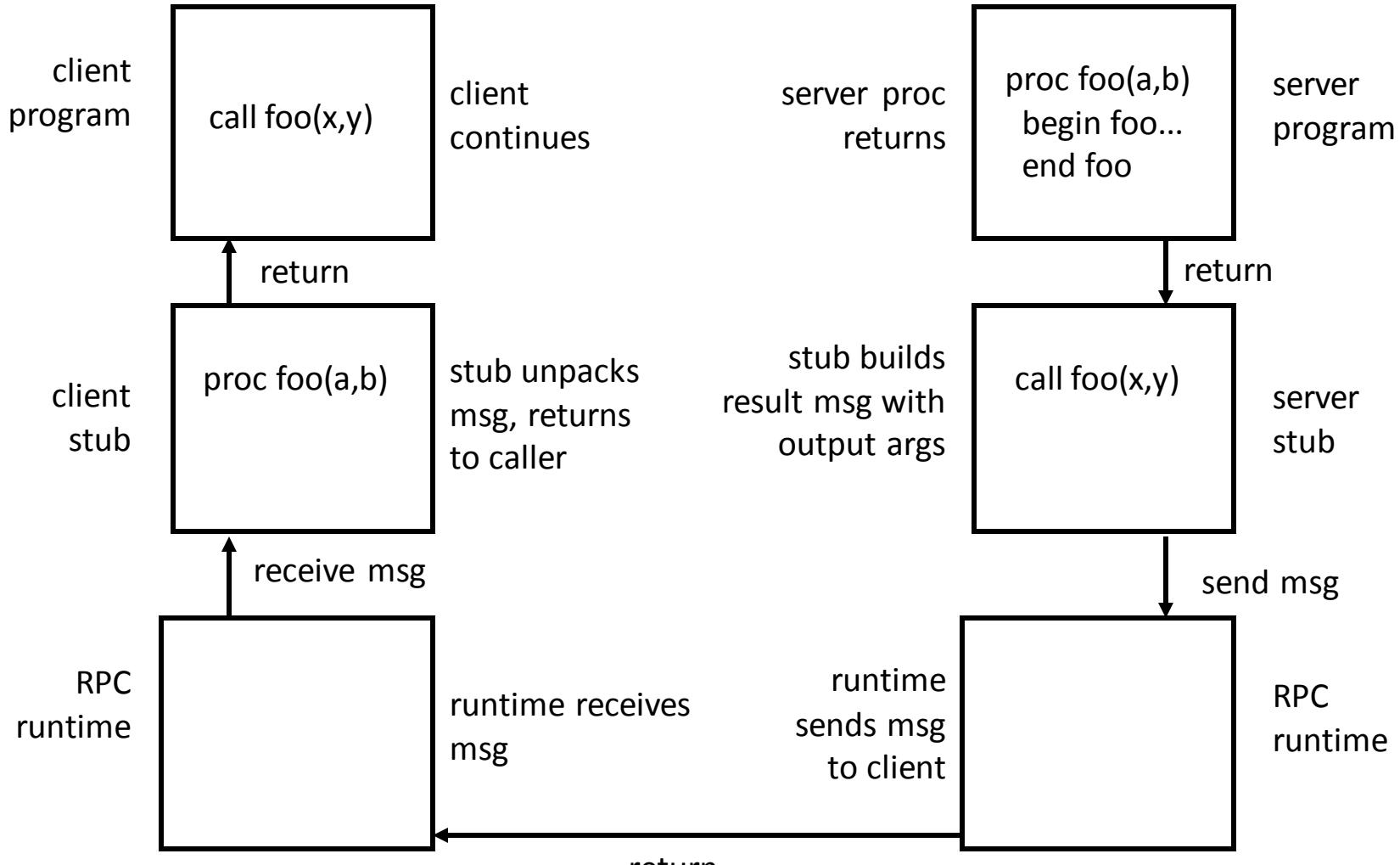
Remote Procedure Calls

- Building a server
 - Server program defines the server's interface using an *interface definition language* (IDL)
 - IDL specifies the names, parameters, and types for all client-callable server procedures
 - A *stub compiler* reads the IDL and produces two stub procedures for each server procedure: a client-side stub and a server-side stub
 - The server code is linked with the server-side stubs; the client code is linked with the client-side stubs.

Remote Procedure Calls



Remote Procedure Calls



Remote Procedure Calls

- RPC binding
 - process of connecting the client and server
 - server, when it starts up, *exports* its interface, identifying itself to a network name server and telling the local runtime its dispatcher address
 - The client, before issuing any calls, *imports* the server, which causes the RPC runtime to lookup the server through the name service and contact the requested server to setup a connection.
 - The *import* and *export* are explicit calls in the code.

Remote Procedure Calls

- RPC marshalling
 - packing of procedure parameters into a message packet
 - The RPC stubs call type-specific procedures to marshall and unmarshall all of the parameters to the call.
 - On the client side, the client stub marshalls the parameters into the call packet; on the server side the server stub unmarshalls the parameters in order to call the server's procedure.
 - On the return, the server stub marshalls return parameters into the return packet; the client stub unmarshalls return parameters and returns to the client.

Remote Procedure Calls

Informatik · CAU Kiel

- SOAP-based RPC
 - parameter passing by value only
 - binding via UDDI, WSDL
 - data model based on XML Schema

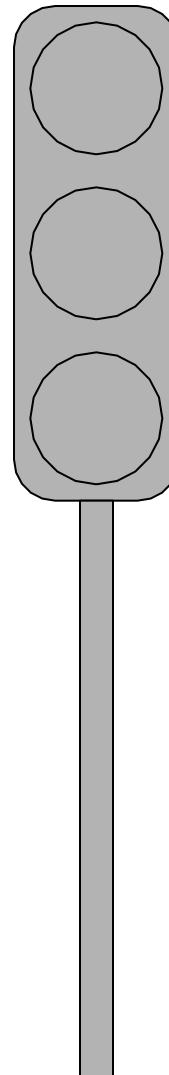
Remote Procedure Calls

- SOAP-based RPC
 - namespace of the target object
 - method name
 - parameters

```
<env:Body>
  <m:GetLastTradePrice
    xmlns:m="http://stocks.com/StockQuotes">
    <tickerSymbol>SUNW</tickerSymbol>
  </m:GetLastTradePrice>
</env:Body>
```

Feedback for this Chapter

- Well understood –
easy material
- Mostly understood –
material is ok
- Hardly understood –
difficult material



Web Services

Web Service Description Language

Lecture "XML in Communication Systems"

Chapter 13

Prof. Dr.-Ing. Norbert Luttenberger

Research Group for Communication Systems

Dept. of Computer Science

Christian-Albrechts-University in Kiel



Recommended Reading

- Booth, D. et al.:
Web Services Architecture
<http://www.w3.org/TR/ws-arch/>
- Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.:
Web Services Description Language (WSDL) 1.1
<http://www.w3.org/TR/wsdl>
- Booth, D., Liu, C.K.:
Web Services Description Language (WSDL) Version 2.0 Part 0: Primer
<http://www.w3.org/TR/wsdl20-primer>
- Ballinger, K., Ehnebuske, D., Gudgin, M., Nottingham, M., Yendluri, P.:
Web Services Interoperability Organization Basic Profile Version 1.0
<http://www.ws-i.org/Profiles/BasicProfile-1.0.html>

Overview

Informatik · CAU Kiel

1. Web Service Description Language (WSDL)



Chapter 13.1

Web Service Description Language

Web Service Description

- Service description

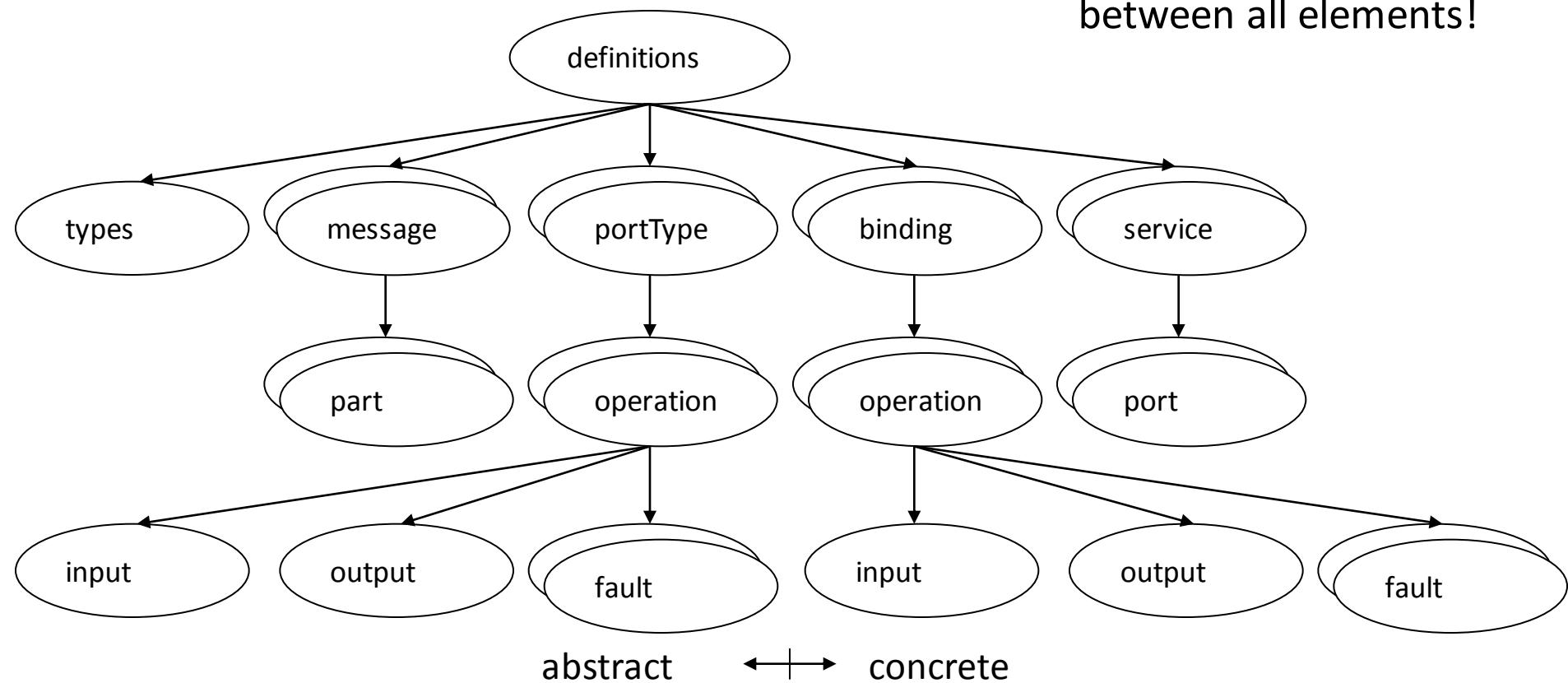
The mechanics of the message exchange are documented in a Web service description (WSD). The WSD is a machine-processable specification of the Web service's interface, written in WSDL. It defines the message formats, datatypes, transport protocols, and transport serialization formats that should be used between the requester agent and the provider agent. It also specifies one or more network locations at which a provider agent can be invoked, and may provide some information about the message exchange pattern that is expected. In essence, the service description represents an agreement governing the mechanics of interacting with that service.

Web Service Description

Informatik · CAU Kiel

- WSDL overview

Cross-references
between all elements!



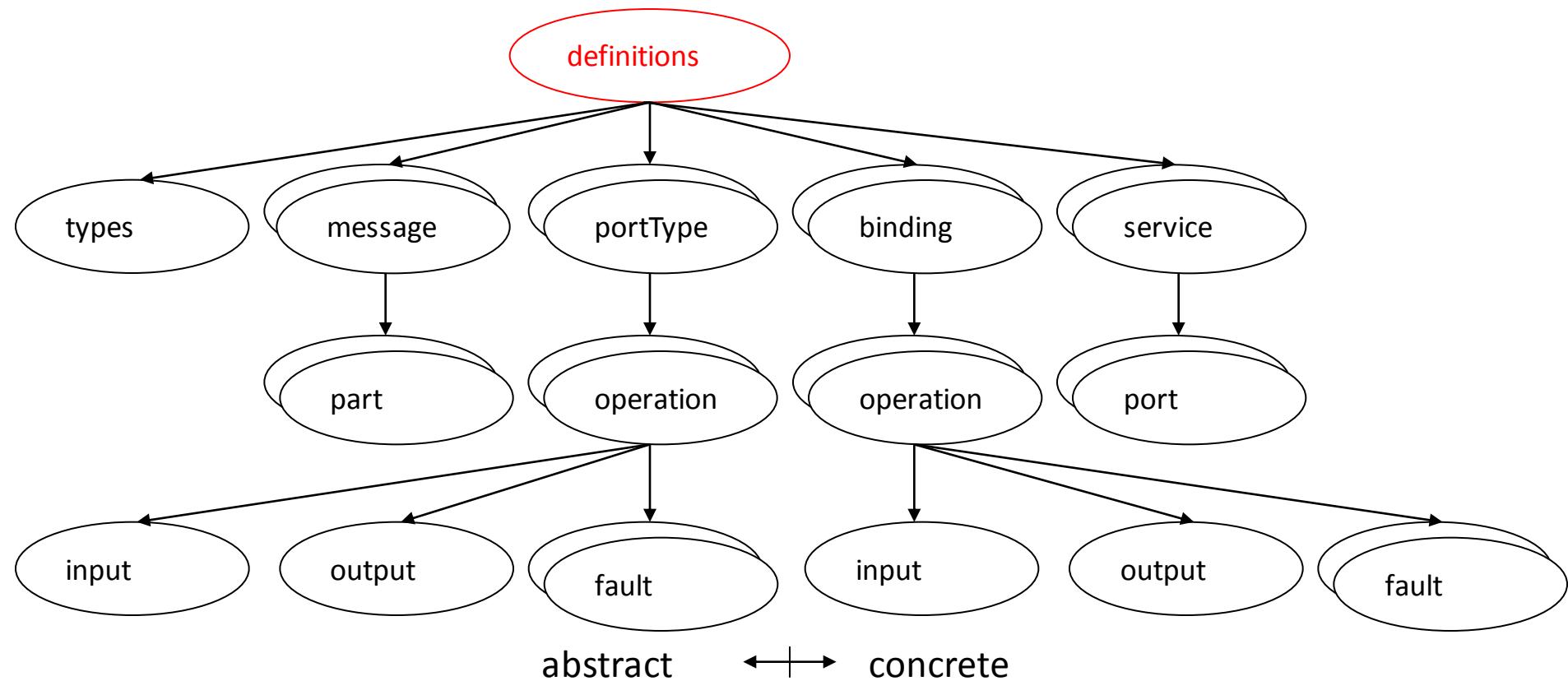
Web Service Description

- "A WSDL document is simply a set of definitions."
 - **types** provides data type definitions used to describe the messages exchanged.
 - **message** represents an abstract definition of the data being transmitted. A message consists of logical parts each of which is associated with a definition within some type system.
 - **portType** is a set of abstract operations. Each operation refers to an input message and output messages.
 - **binding** specifies concrete protocol and data format specifications for the operations and messages defined by a particular portType.
 - **service** is used to aggregate a set of related ports.

Web Service Description

Informatik · CAU Kiel

- WSDL overview



Web Service Description

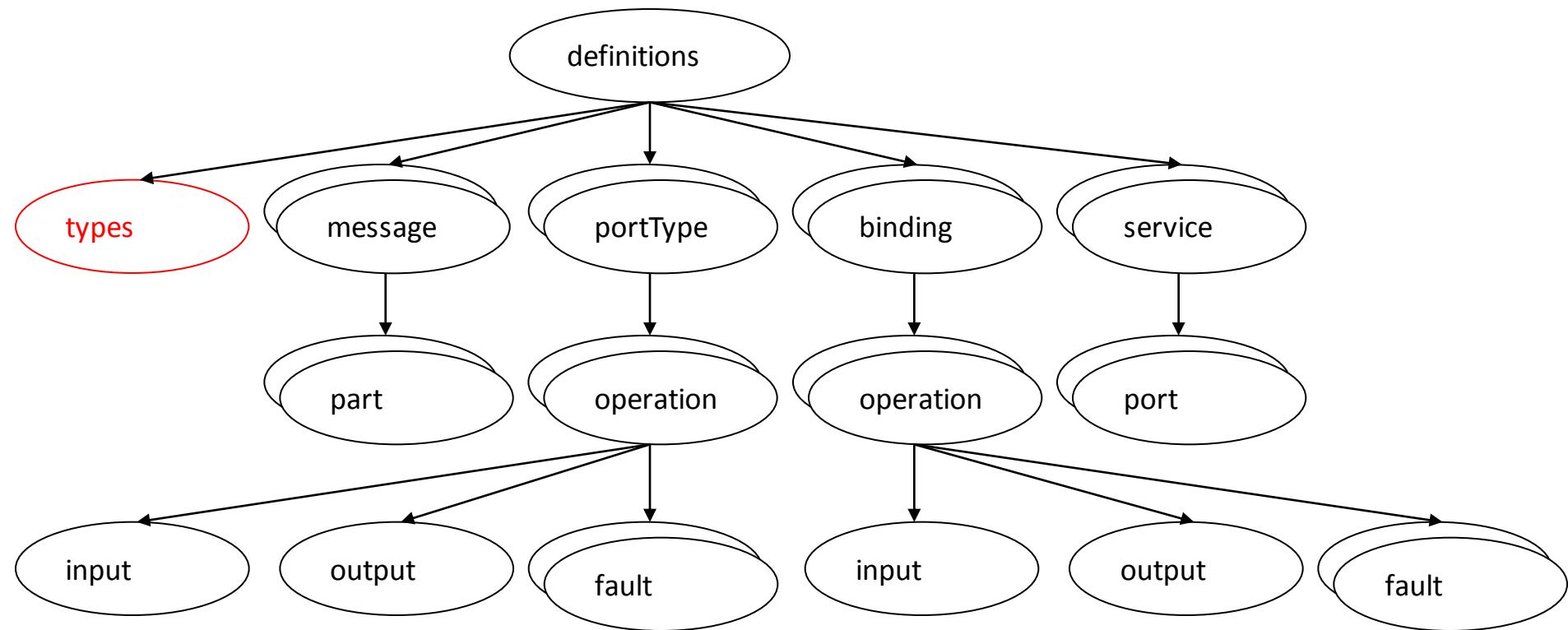
- Sample definitions element

```
<?xml version="1.0" encoding="UTF-8"?>

<wsdl:definitions
    targetNamespace="http://example.org/math/types/"
    xmlns:tns="http://example.org/math/types/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Web Service Description

- WSDL types element



Web Service Description

- WSDL types element
 - definition of data types for messages
 - WS-I: data type definition language is W3C XML Schema
 - WS-I: types must "literally" be transformed into wireline encoding (`encoding="literal"`)

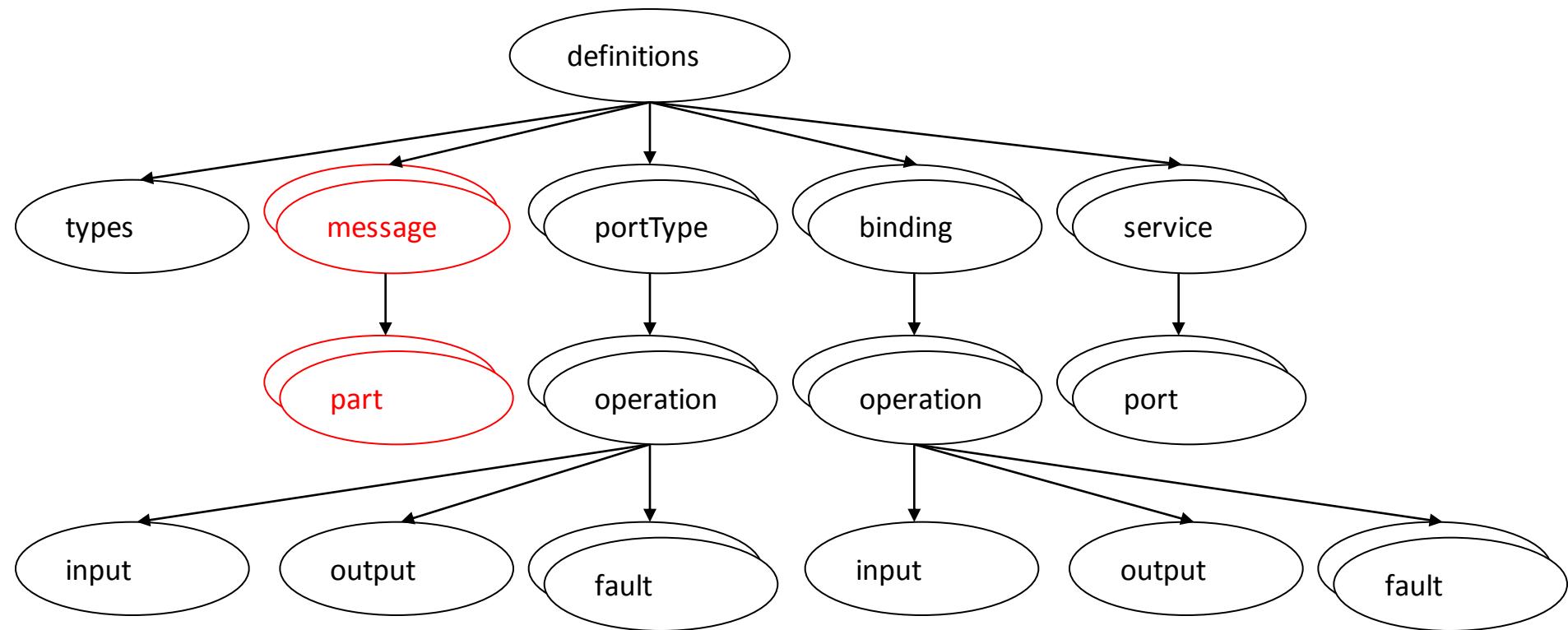
Web Service Description

- Sample types element

```
<wsdl:types>
  <xsd:schema
    targetNamespace="http://example.org/math/types/" >
    <xsd:complexType name="MathInput"> <xsd:sequence>
      <xsd:element name="x" type="xsd:double"/>
      <xsd:element name="y" type="xsd:double"/>
    </xsd:sequence> </xsd:complexType>
    <xsd:complexType name="MathOutput"> <xsd:sequence>
      <xsd:element name="result" type="xsd:double"/>
    </xsd:sequence> </xsd:complexType>
    <xsd:element name="AddRequest" type="tns:MathInput"/>
    <xsd:element name="AddResult" type="tns:MathOutput"/>
  </xsd:schema>
</wsdl:types>
```

Web Service Description

- WSDL message and part elements



Web Service Description

- Message format depends on SOAP binding style :-\
 - document style
 - paradigm: exchange of self-contained documents
 - possibly one-way operation
 - WS description to specify document schema
 - remote procedure call (RPC) style
 - paradigm: WS client passes parameters to procedure, receives result (request/response operation)
 - The body contains an XML representation of a method call, the message parts represent the parameters to the method.
 - WS message must include identification for "method"

Web Service Description

- WSDL message elements
 - composition of messages from parts
 - two mutually exclusive kinds of references to the types element:

Reference to element
→ "document style"

```
<wsdl:message name="AddRequestMessage">
  <wsdl:part name="RequestBody"
    element="tns:AddRequest"/>
</wsdl:message>

<wsdl:message name="AddResultMessage">
  <wsdl:part name="ResultBody"
    element="tns:AddResult"/>
</wsdl:message>
```

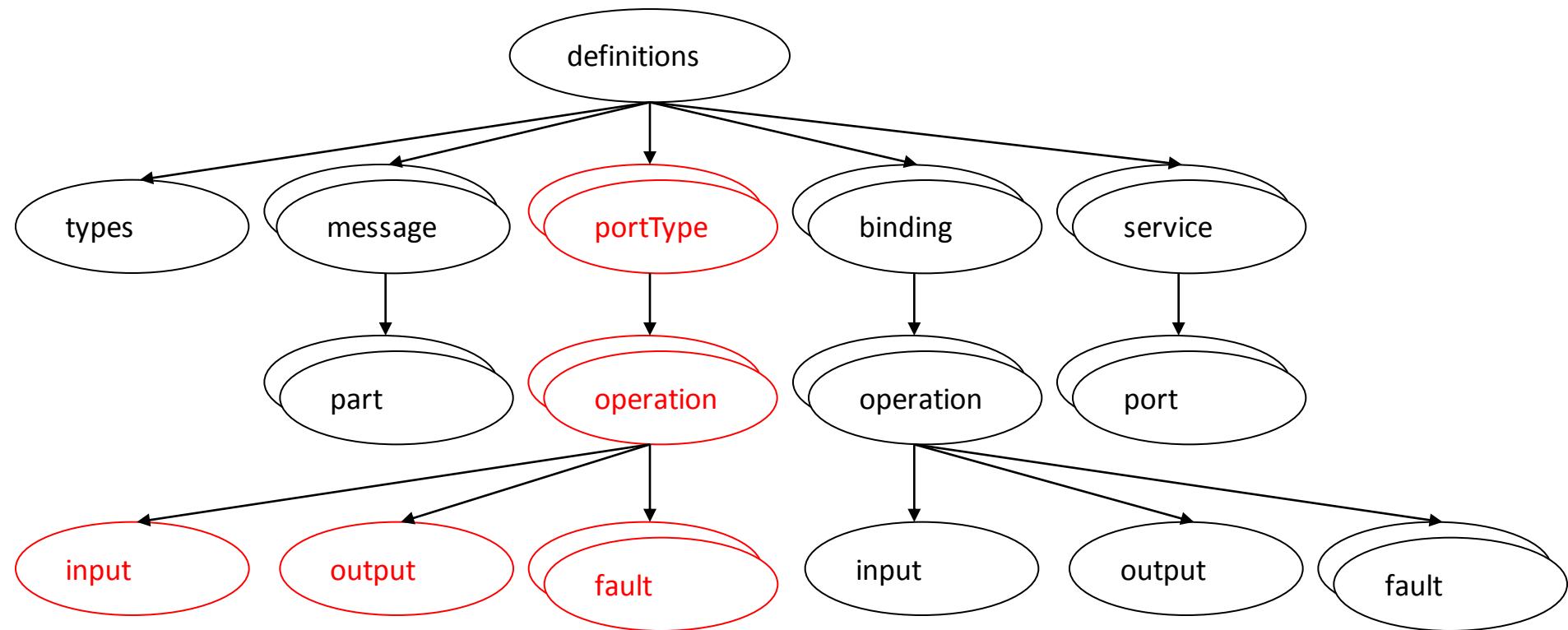
Reference to type
→ "RPC style"

```
<wsdl:message name="AddRequestMessage">
  <wsdl:part name="addInPara"
    type="tns:MathInput"/>
</wsdl:message>

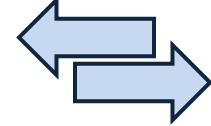
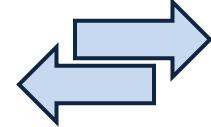
<wsdl:message name="AddResultMessage">
  <wsdl:part name="addOutPara"
    type="tns:MathOutput"/>
</wsdl:message>
```

Web Service Description

- WSDL portType element



Web Service Description

- WSDL portType elements
 - "A portType is a named set of **abstract operations** and the abstract messages involved."
 - Four kinds of operations:
 - **One-way**: The endpoint receives a message. 
 - **Request-response**: The endpoint receives a message, and sends a correlated message. 
 - **Solicit-response**: The endpoint sends a message, and receives a correlated message. 
 - **Notification**: The endpoint sends a message. 

Web Service Description

- WSDL portType elements

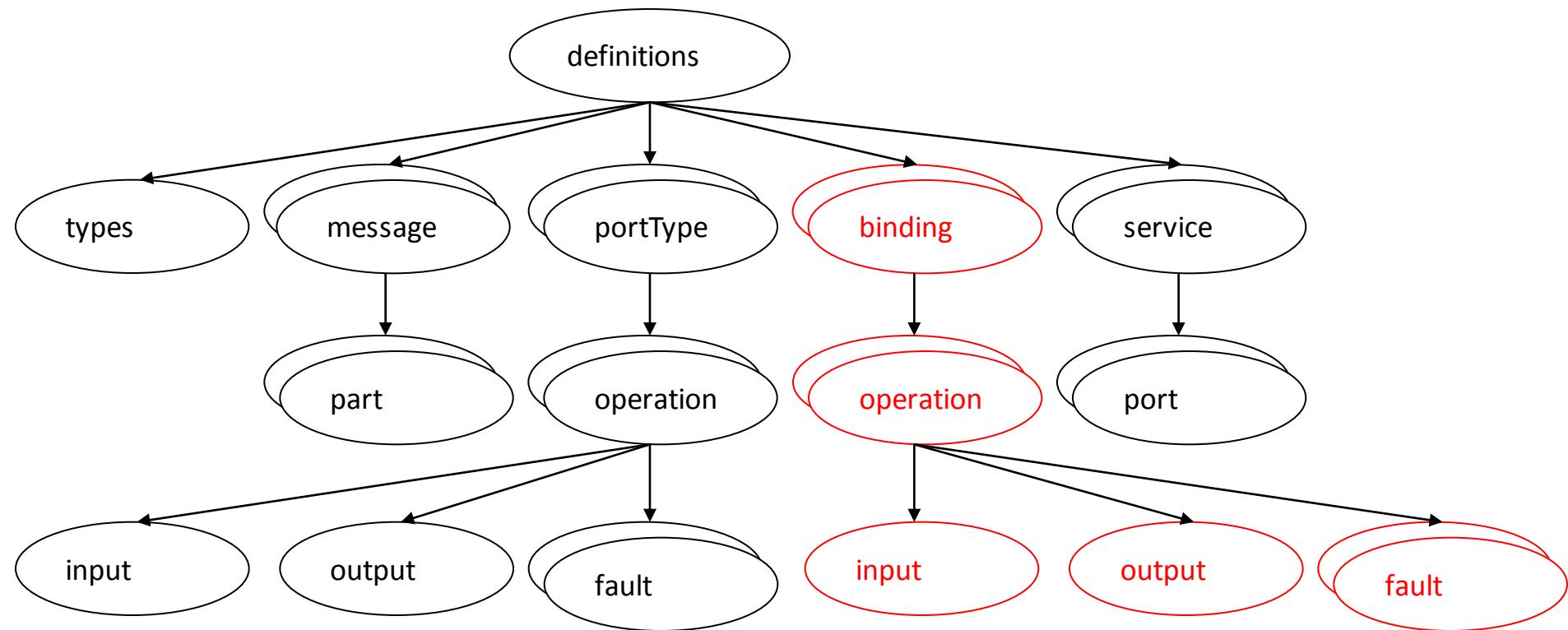
```
<wsdl:portType name="mathPortType">
    <wsdl:operation name="Addition">
        <wsdl:input message="tns:AddRequestMessage"/>
        <wsdl:output message="tns:AddResultMessage"/>
    </wsdl:operation>
</wsdl:portType>
```

Web Service Description

- WSDL portType elements
 - WS-I: only request-response and one-way operations
 - WS-I: only literal encoding, i.e. abstract = concrete operations

Web Service Description

- WSDL binding elements



Web Service Description

- WSDL binding elements
 - A binding defines message format and protocol details for operations and messages defined by a particular portType.
 - WSDL binding element required per portType
 - defined bindings:
 - SOAP
 - HTTP
 - MIME
 - There may be any number of bindings for a given portType.
 - discussed here: SOAP-HTTP binding only

Web Service Description

- Example: SOAP binding (WSDL extensibility elements)

```
<wsdl:binding name="SOAPBinding" type="tns:mathPortType">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="Addition">
        <soap:operation soapAction="Add two numbers!"/>
        <wsdl:input> <soap:body use="literal"/> </wsdl:input>
        <wsdl:output> <soap:body use="literal"/> </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

- `style="rpc | [document]"` identifies message format
- `transport="http://..."` identifies transport protocol (http-POST)
(other value e.g.: `transport="http://schemas.xmlsoap.org/soap/smtp"`)

Web Service Description

- Example: SOAP binding (WSDL extensibility elements)

```
<wsdl:binding name="SOAPBinding" type="tns:mathPortType">
    <soap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="Addition">
        <soap:operation soapAction="Add two numbers!"/>
        <wsdl:input> <soap:body use="literal"/> </wsdl:input>
        <wsdl:output> <soap:body use="literal"/> </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

- `style="rpc | [document]"` identifies message format
- `transport="http://..."` identifies transport protocol (http-POST)
(other value e.g.: `transport="http://schemas.xmlsoap.org/soap/smtp"`)

Web Service Description

- Resulting messages

- document style message:

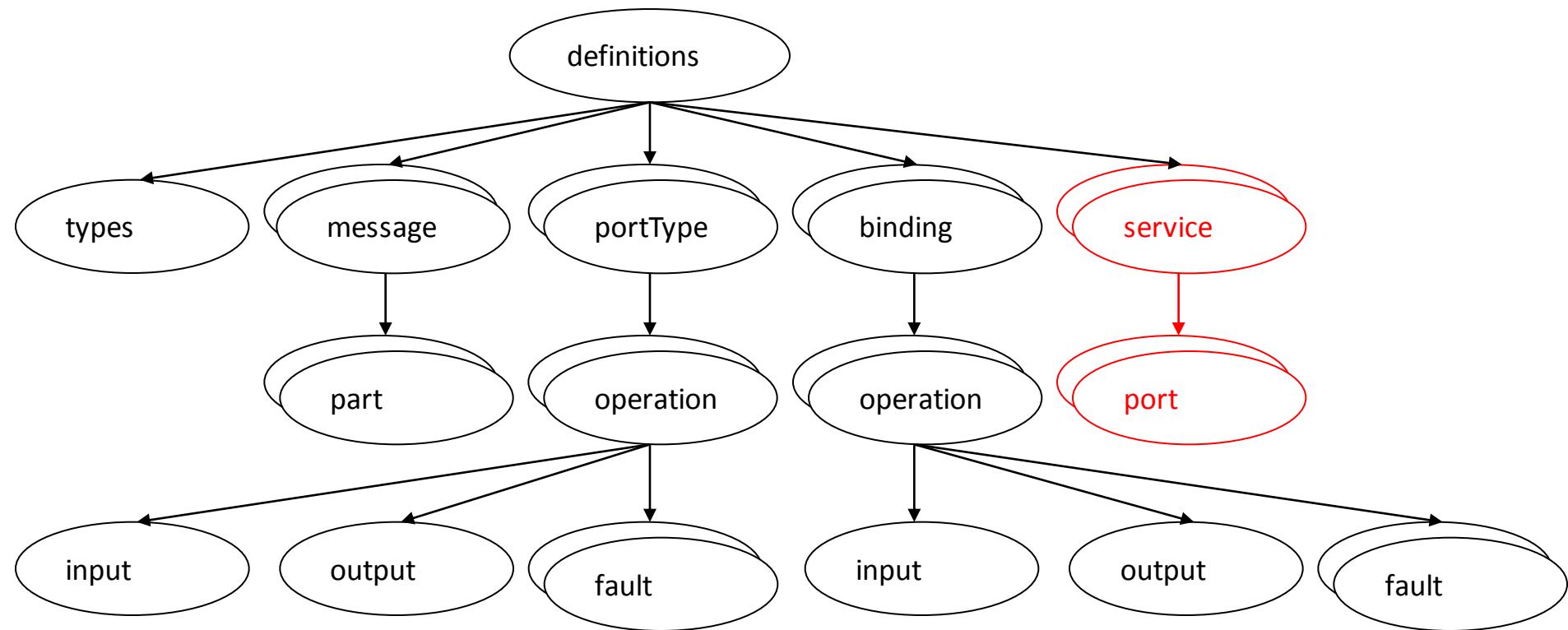
```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="...">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <AddRequest xmlns="...">
      <x>3.14</x> <y>3.14</y>
    </AddRequest>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- rpc style message:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="...">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <oxy:Addition>
      <addInPara>
        <x>3.14</x> <y>3.14</y>
      </addInPara>
    </oxy:Addition>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Web Service Description

- WSDL operation elements



Web Service Description

- WSDL service elements
 - A port is defined as individual endpoint by specifying a single address for a binding.

```
<service name="MathService">
  <port name="MathEndpoint" binding="mt:SOAPBinding">
    <soap:address location="http://localhost/math/math"/>
  </port>
</service>
```

Feedback for this Chapter

Informatik · CAU Kiel

- Well understood –
easy material
- Mostly understood –
material is ok
- Hardly understood –
difficult material

