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

Studies

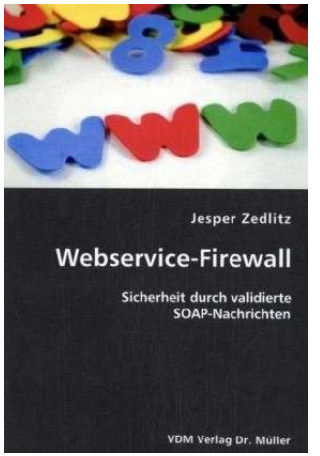
1999-2005 University Kiel

Subject

Computer Science (Dipl.-Inf.)

- Thesis

  Spezifikation und Implementierung eines Application Level Gateways für Webservices



May I introduce myself

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

Photos: V. Diekamp

Expedition in the Mediterranean

June/July 2010

Controlling the "MeBo"

May I introduce myself

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?

What is XML?

- 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

YOUR GOALS

YOUR NON-GOALS

My Lecture Goals

- 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

- 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

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

Lecture Overview

- 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

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

Please …

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

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

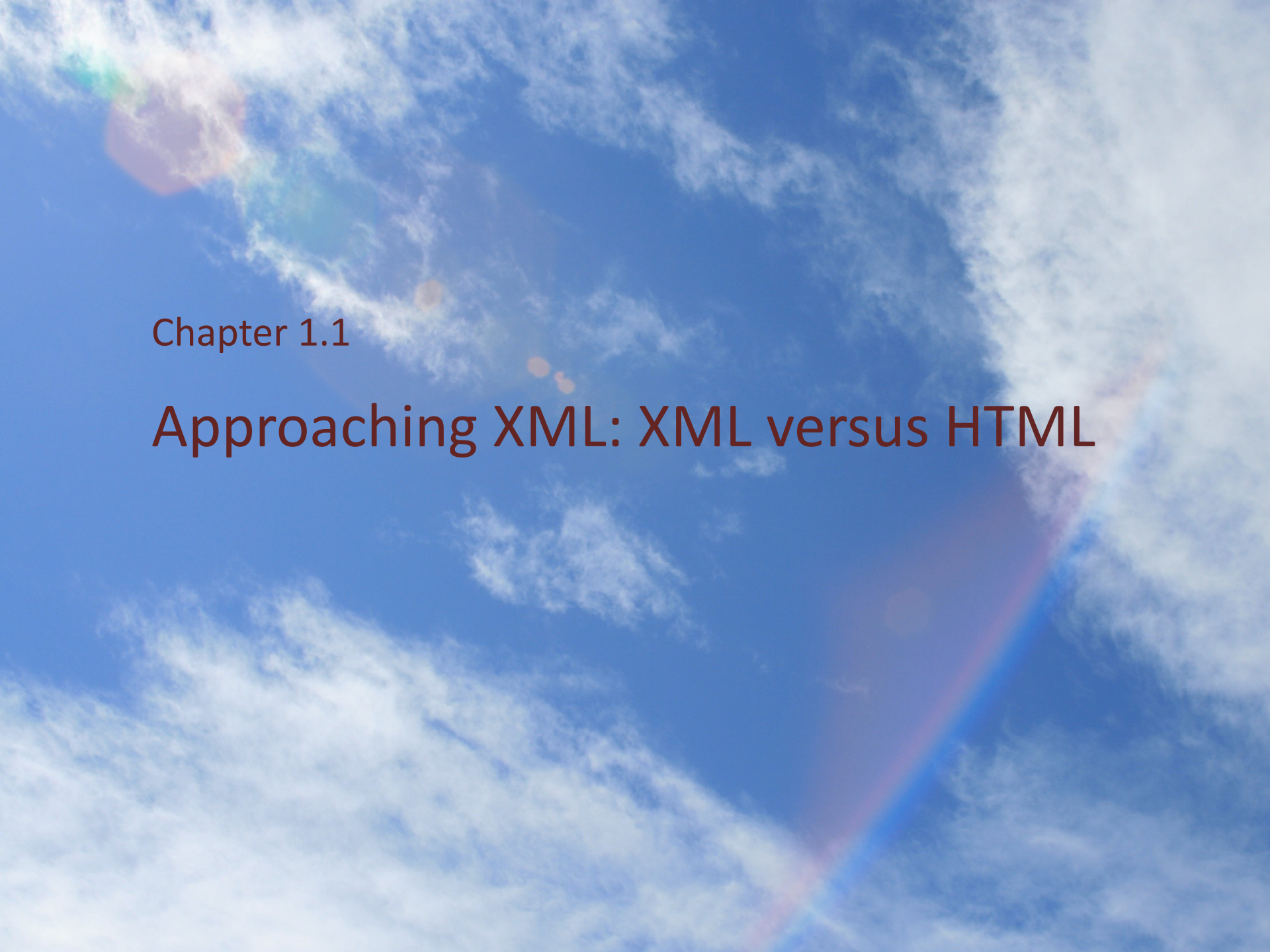
Recommended Reading

- 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

- 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
```

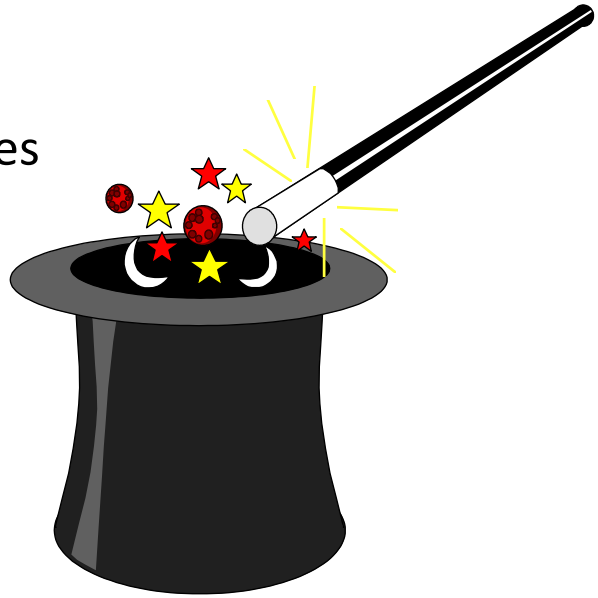
  - Assume an (un)intelligent agent trying to retrieve the names of the authors of the book:
    - Authors' names appear immediately after the title
    - or immediately after the word "by"?
    - Are there two authors?
    - Or just one, called "Holzner, St. and Else, Someone"?

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

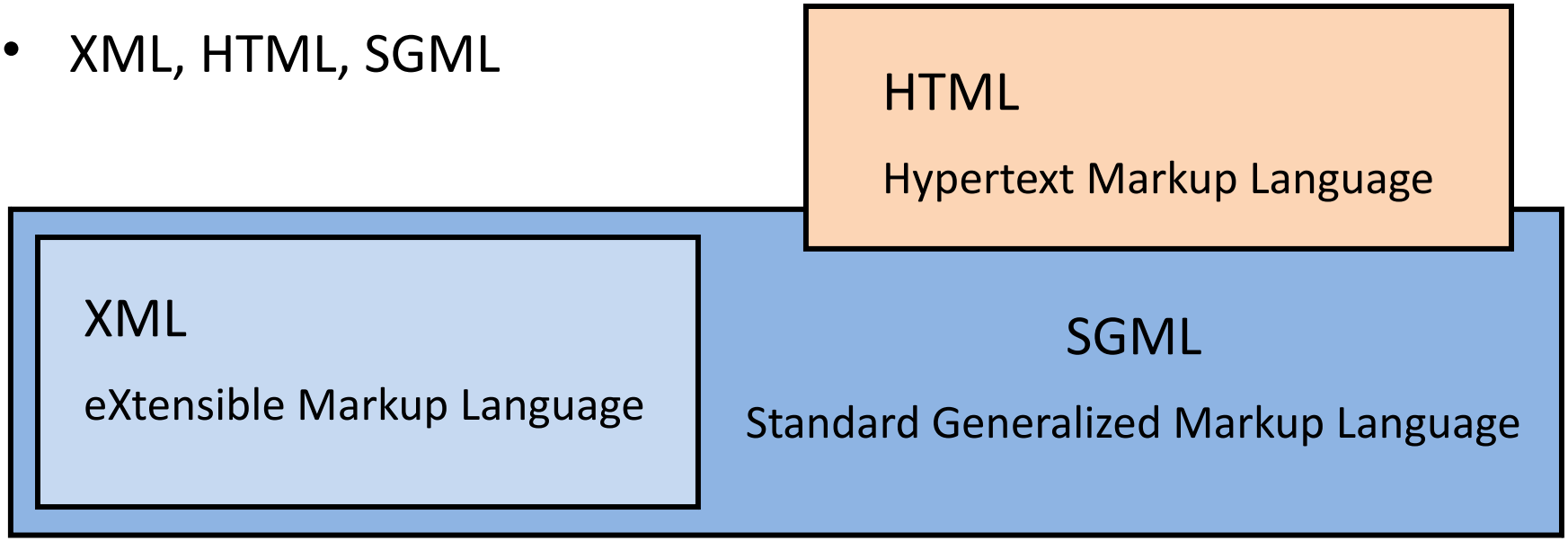
XML versus HTML

- 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



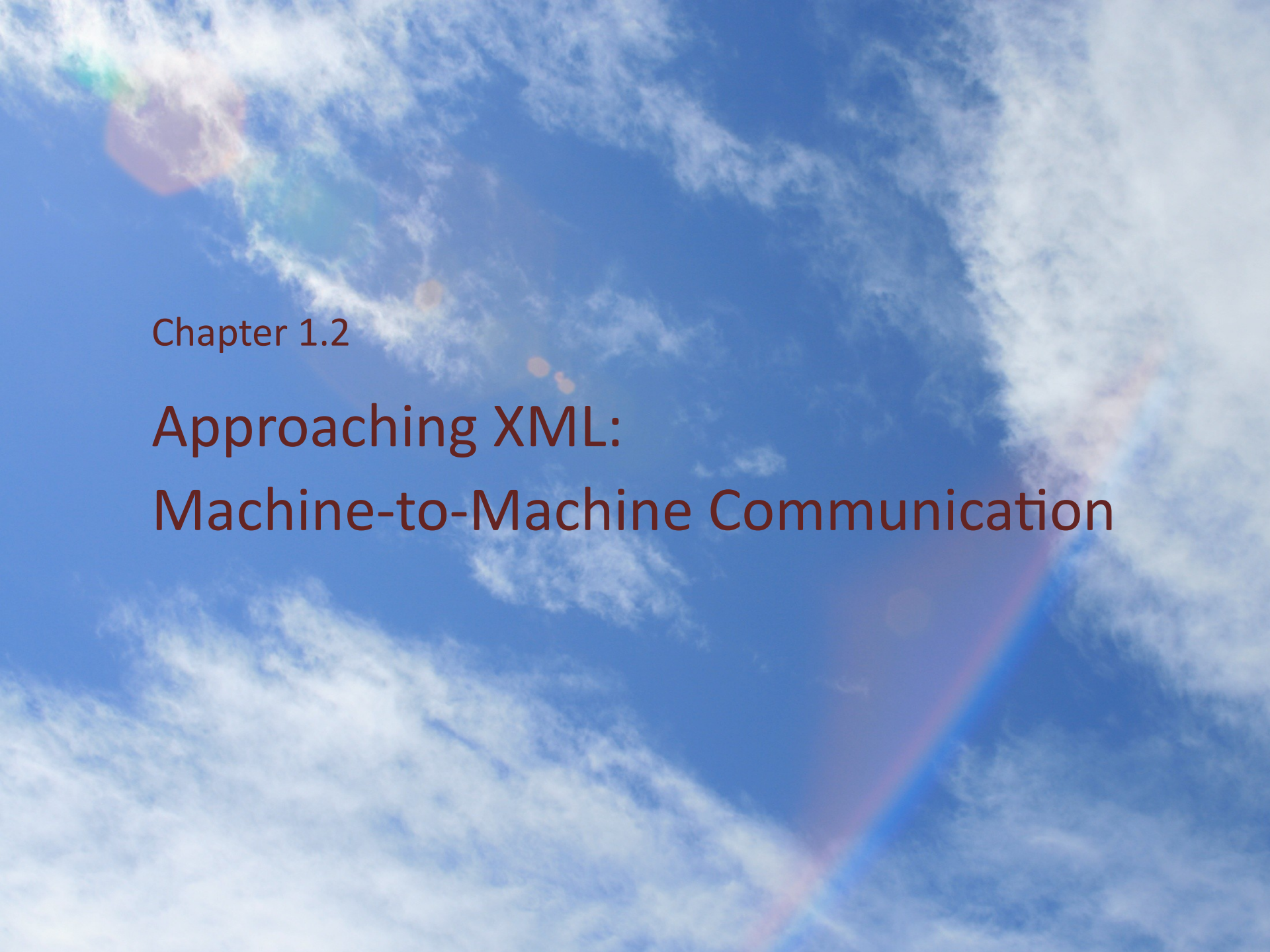
XML versus HTML

- XML, HTML, SGML



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

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

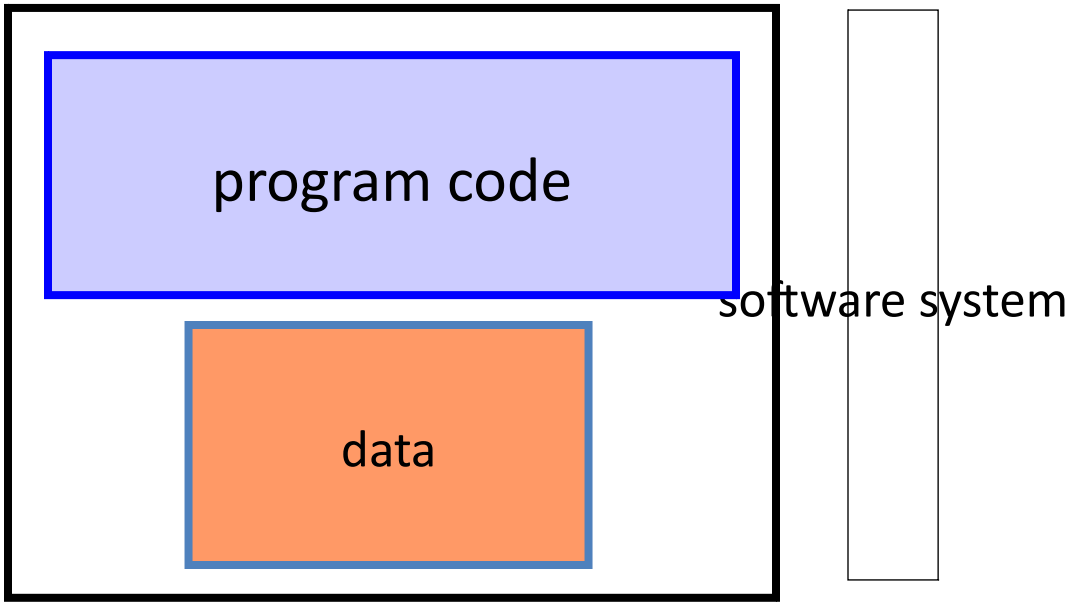


Chapter 1.2

Approaching XML: Machine-to-Machine Communication

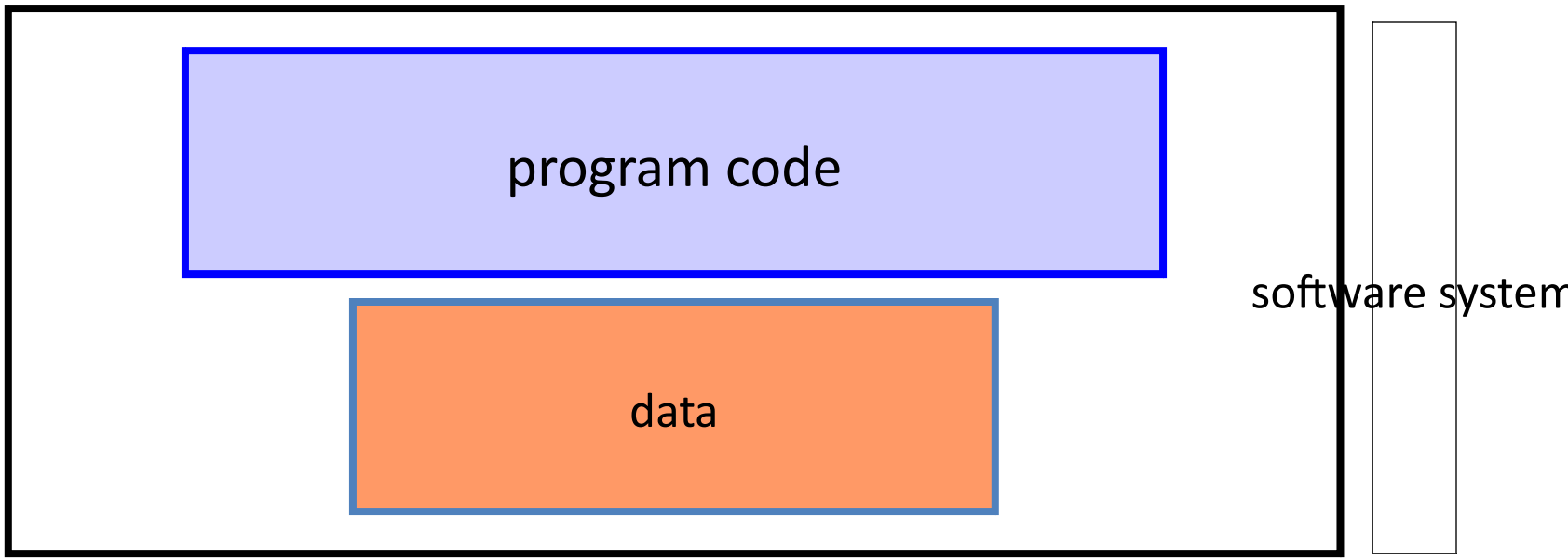
XML in Communications

- In the beginning …



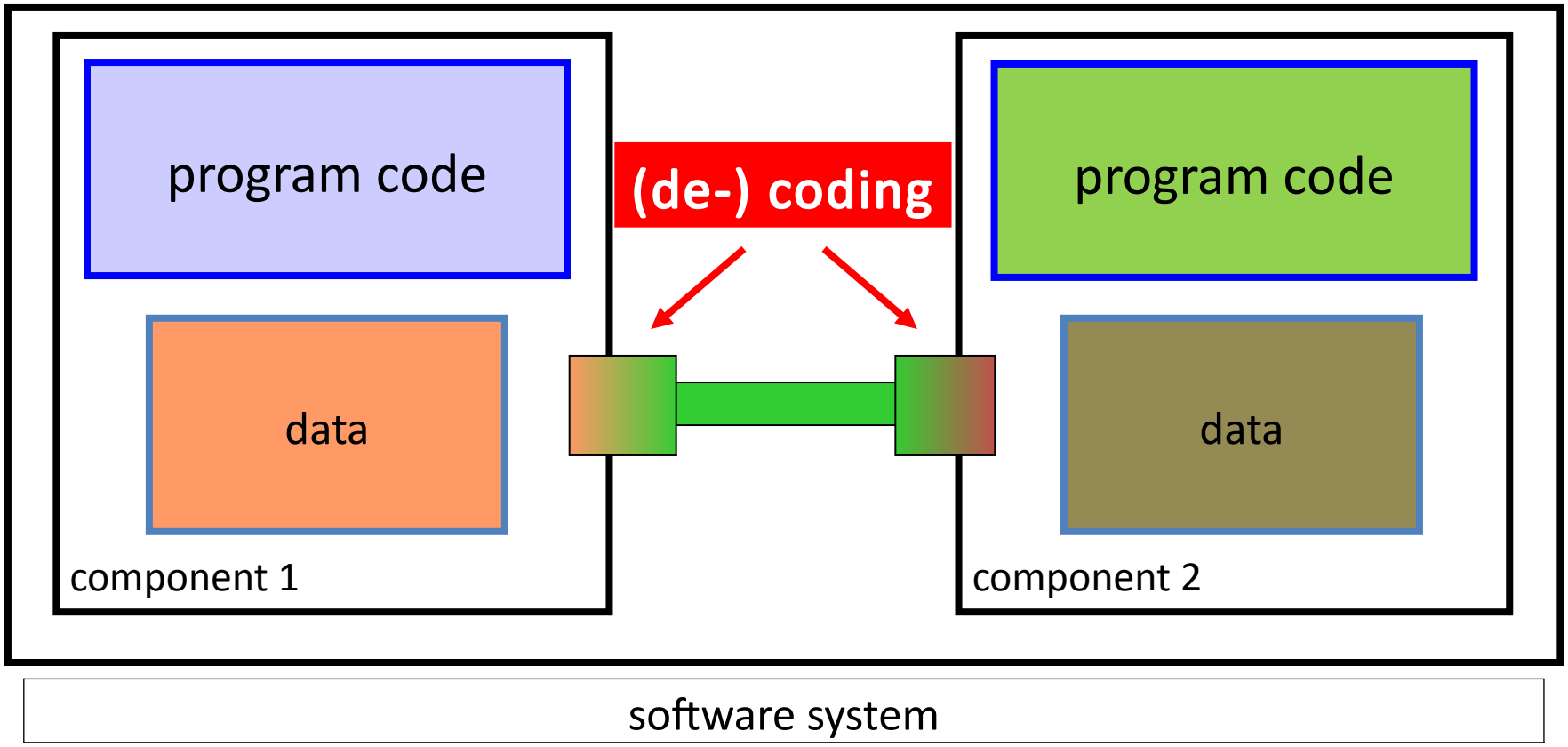
XML in Communications

- Everything became bigger …



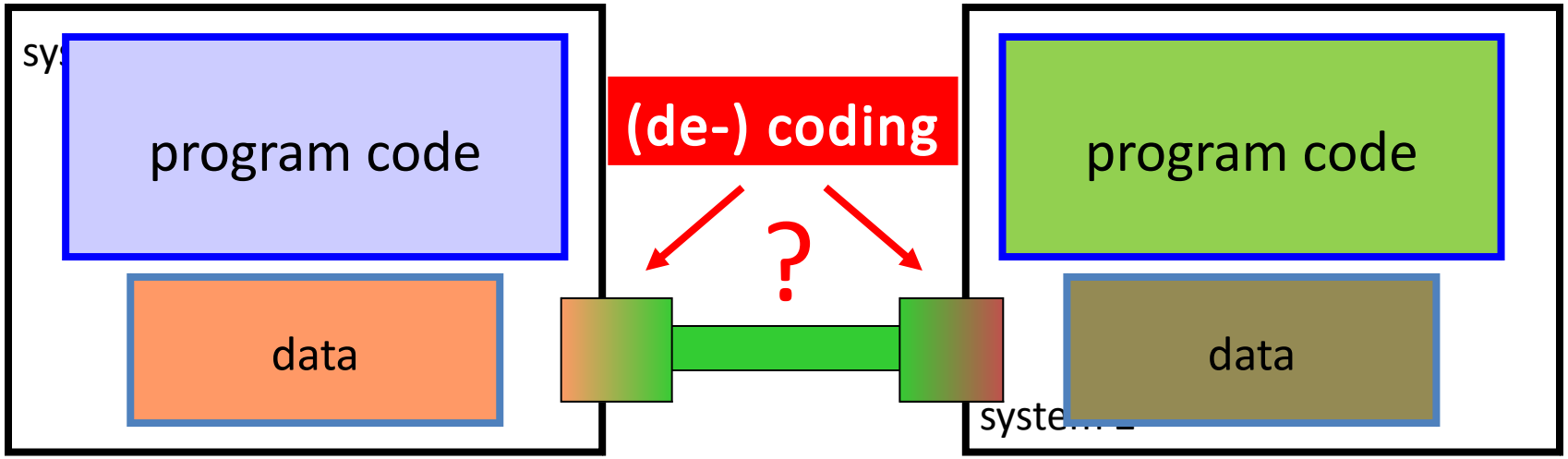
XML in Communications

- … and bigger



XML in Communications

- … and distributed

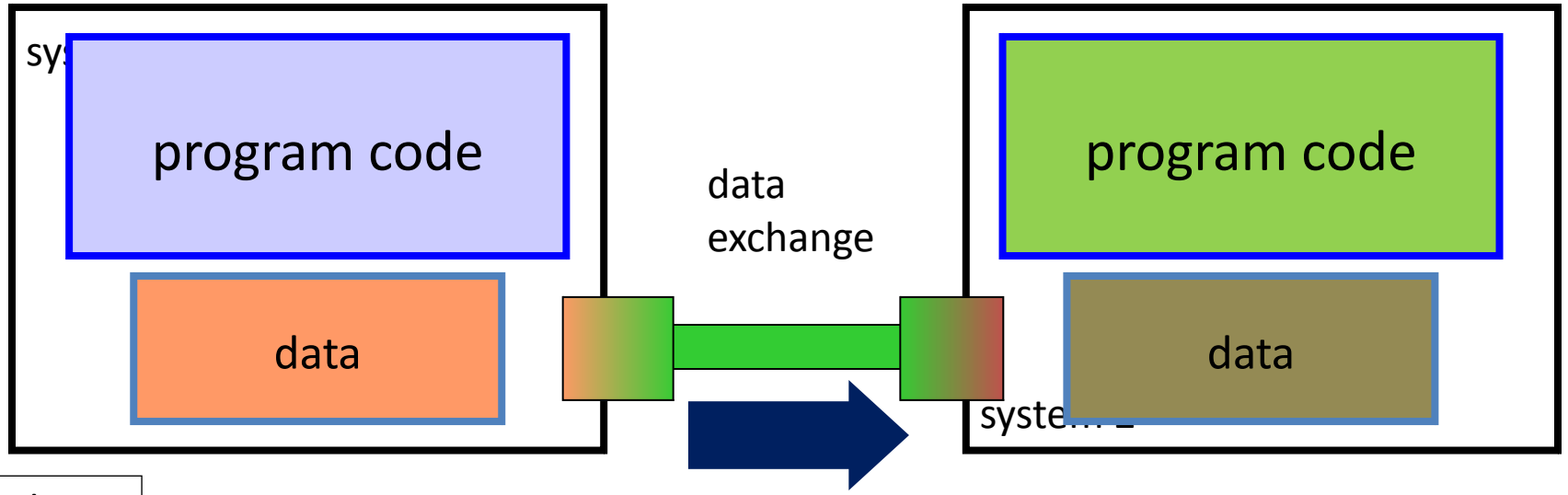


Big questions:

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

XML in Communications

- Sample: record transmission



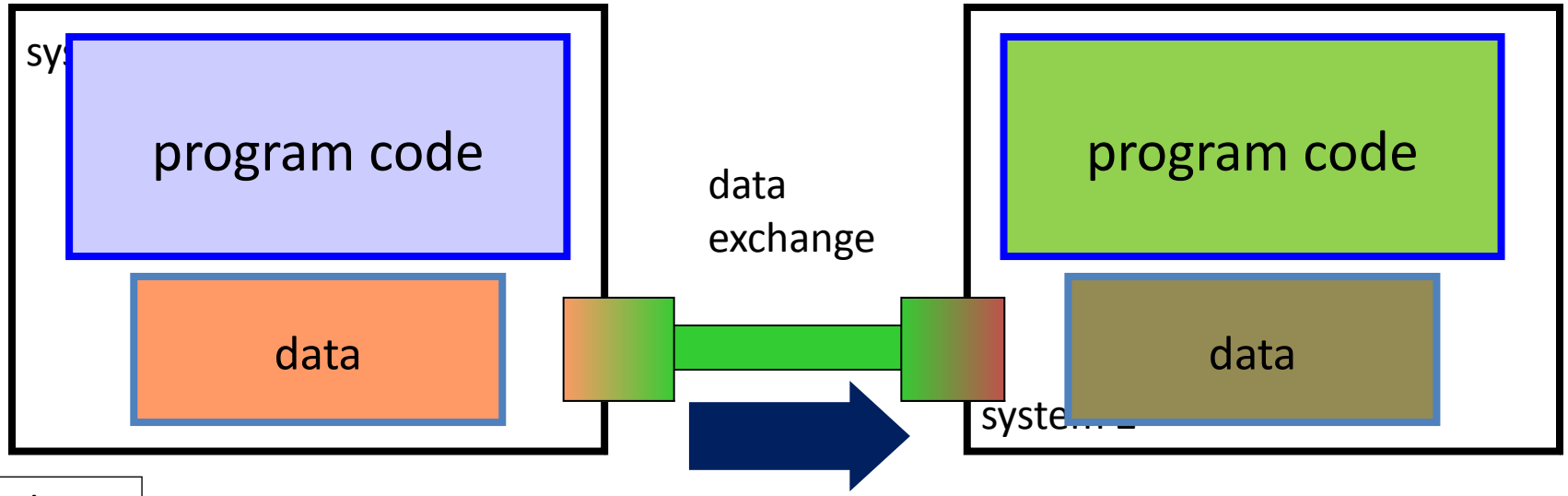
record

Willibald Wusel, 0431/880555, 04321/2233, wiwu@uni-kiel.de, Olshausenstr. 100, 24118 Kiel

first name
surname
phone office
phone private
e-mail
street
number
zipcode
city

XML in Communications

- Record: data + structure



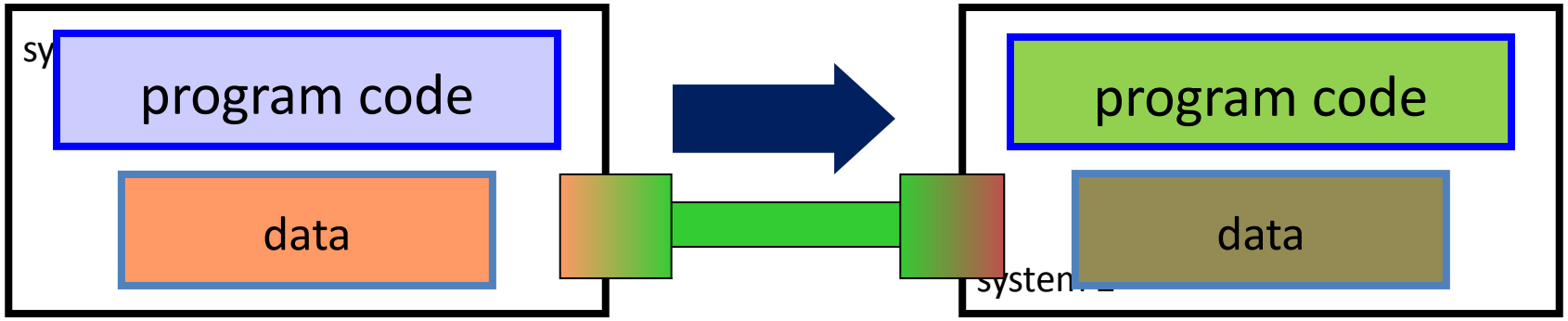
record

Willibald Wusel, 0431/880555, 04321/2233, wiwu@uni-kiel.de, Olshausenstr. 100, 24118 Kiel

first name
surname
phone office
phone private
e-mail
street
number
zipcode
city

XML in Communications

- Record: data + structure

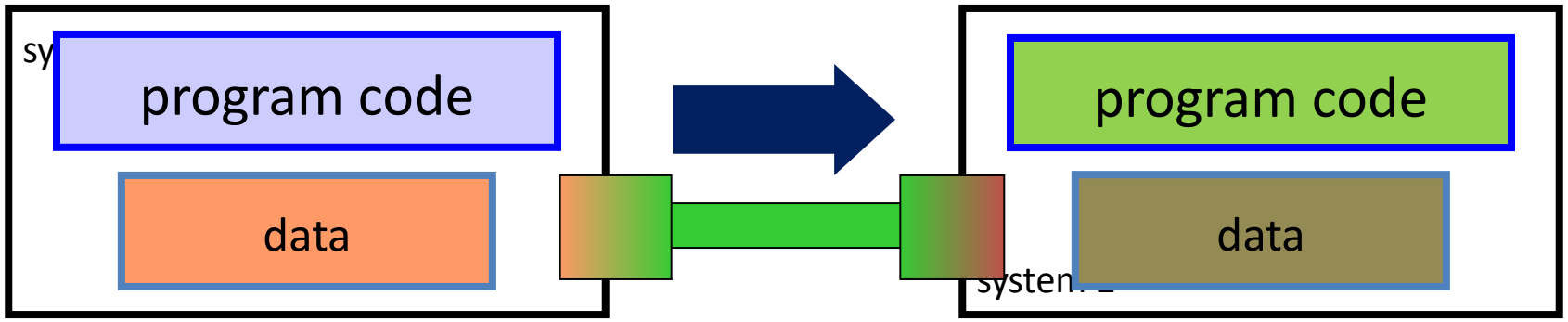


record

```
<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

- Nested structures



record

Willibald Wusel, 0431/880555, 04321/2233, wiwu@uni-kiel.de, Olshausenstr. 100, 24118 Kiel

first name
surname
phone office
phone private
e-mail
street
number
zipcode
city

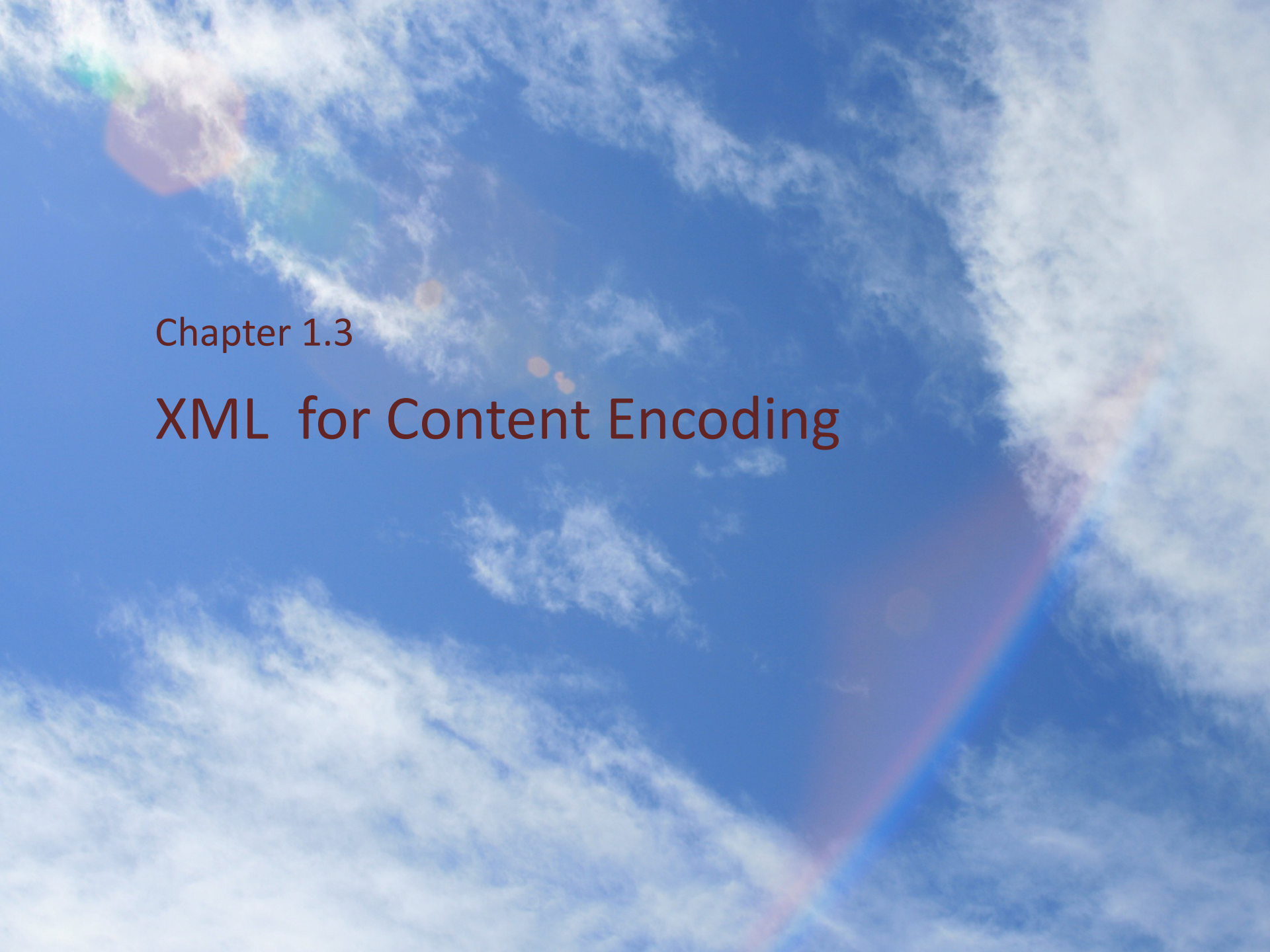
name **contact** **address**

XML in Communications

- 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

- 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
  - $X_{thumbnail}$ – one byte: 0 = no thumbnail
  - $Y_{thumbnail}$ – one byte: 0 = no thumbnail
  - (RGB)n – 3n bytes:
    packed (24-bit) RGB values for the thumbnail pixels, n = $X_{thumbnail}$ * $Y_{thumbnail}$

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

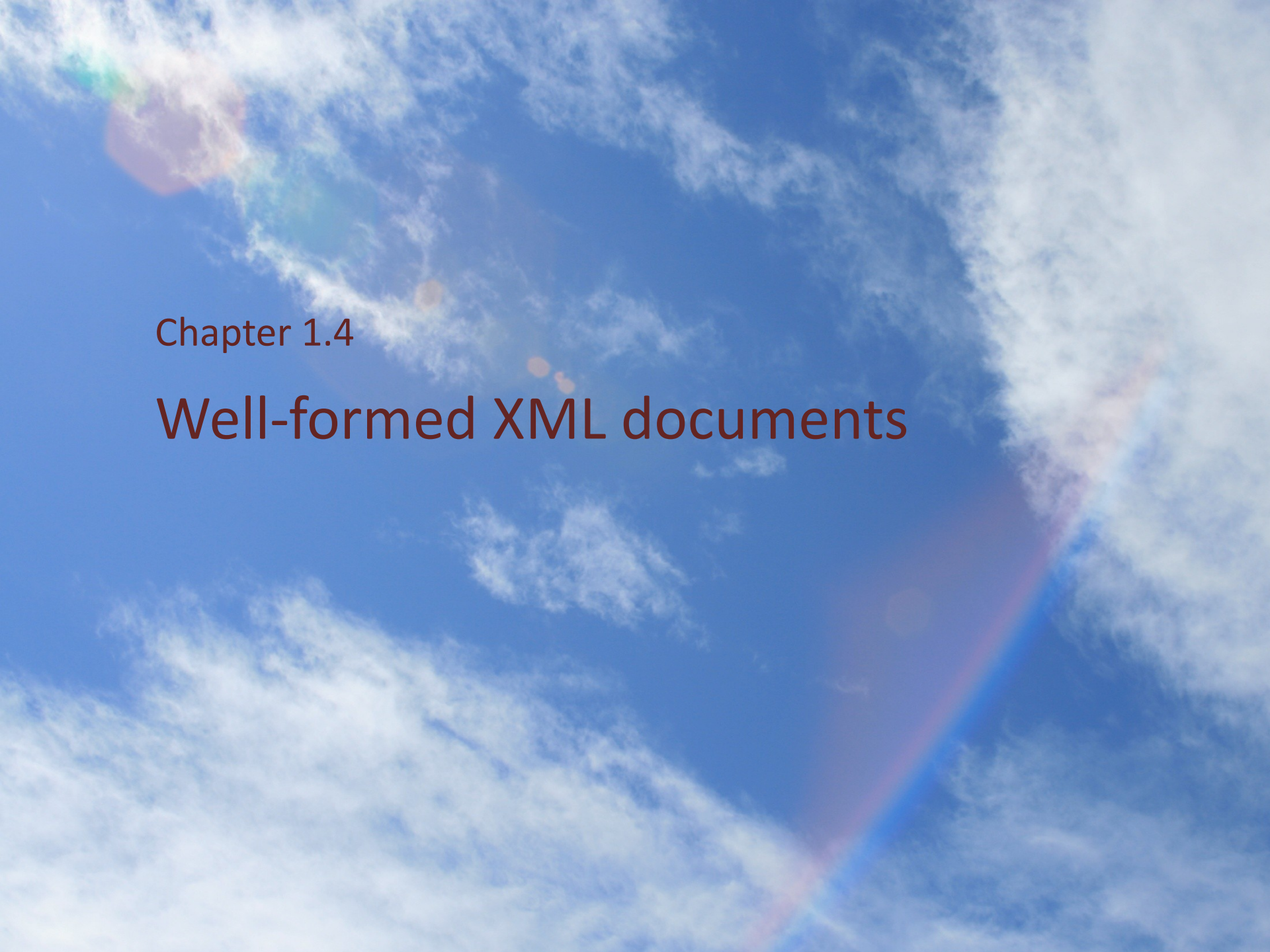
- Storage format for DTP tools: "extensible HTML"
  - e.g. MS Office, OpenOffice
  - supplemented with stylesheets (XSL) to define typographic layout
- Exchange format for data
  - marshalling data in Web Services
  - need to agree on terminology → ontology
- Notation for modeling languages
  - kind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

- 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-#xFFFD] | [#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

- 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

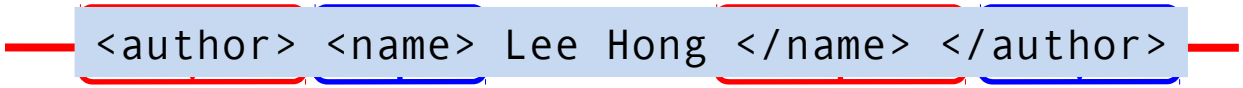
Well-formed XML Documents

- 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.



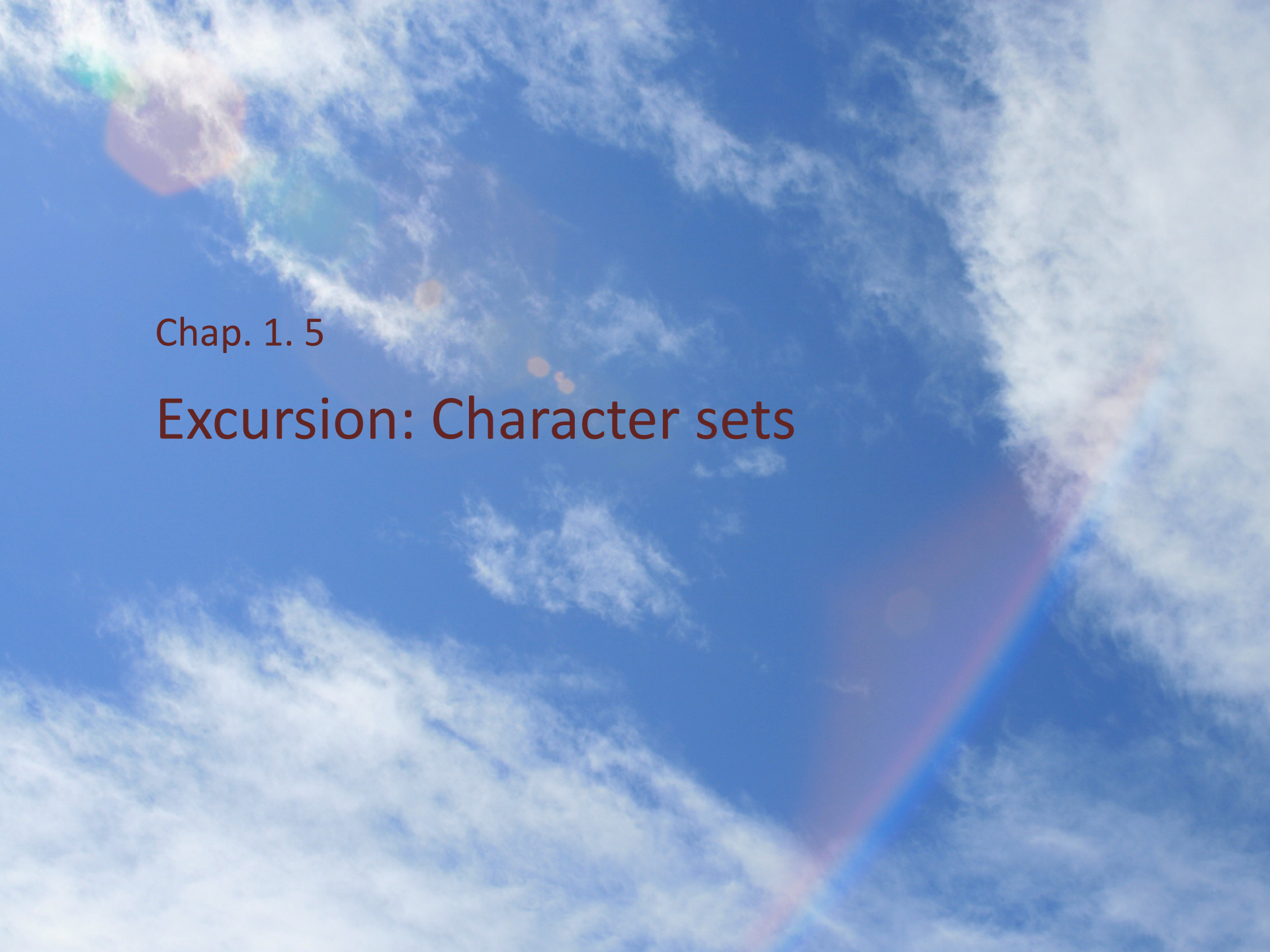
Element content may be characters, or other elements, or nothing.

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

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 ("‰")	$U+2030_{hex}$	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 USACII's 7-bit encoding to 8-bit encoding
  - positions 128 to 159 not used
  - default character set of html

ISO 8859

- Overview

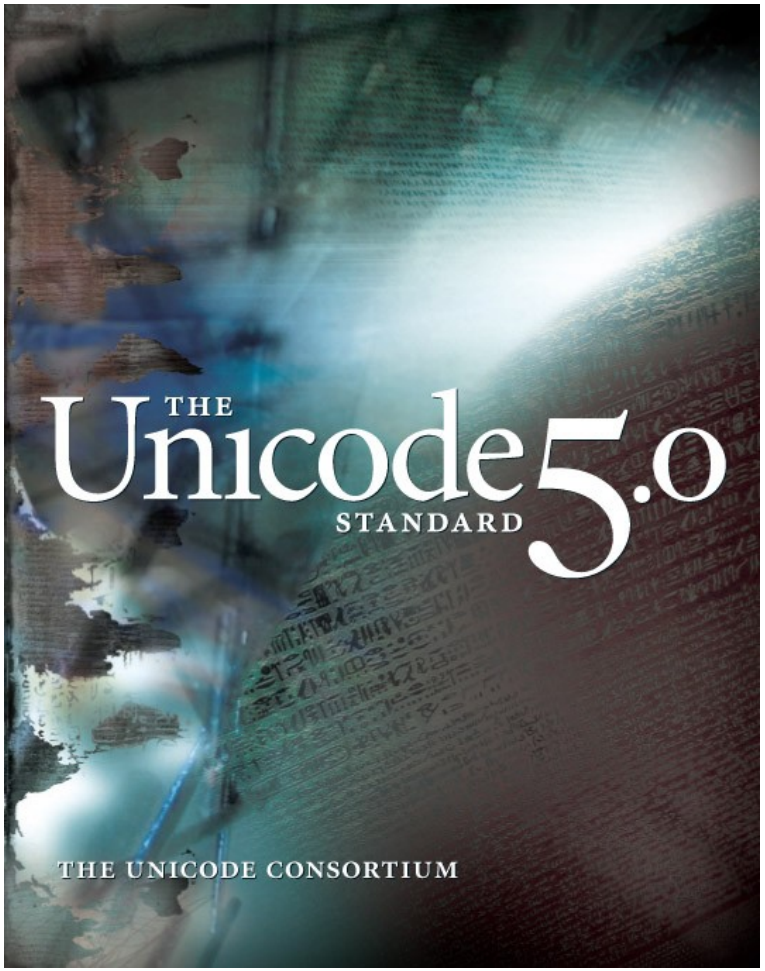
Part 1	Latin 1 (West European)	
2	Latin 2 (East European)	Slavic, Albanian, Hungarian and a variation of Romanian
3	Latin 3 (South European)	Southern European languages (Maltese) and Esperanto
4	Latin 4 (North European)	Northern European languages
5	Cyrillic	
6	Arabic	
7	Greek	under revision (to include, 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	^	`	æ	ø	å	~
Schweden	#	$	É	Ä	Ö	Å	Ü	é	ä	ö	å	ü
Italien	£	$	§	°	ç	é	^	ù	à	ò	ù	ì
Niederlande	£	$	¾	ÿ	½	|	^	`	¨	ƒ	¼	´
Spanien	£	$	§	¡	Ñ	¿	^	`	°	ñ	ç	~
Portugal	#	$	@	Ã	Ç	Õ	^	`	ã	ç	õ	~

Unicode



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)

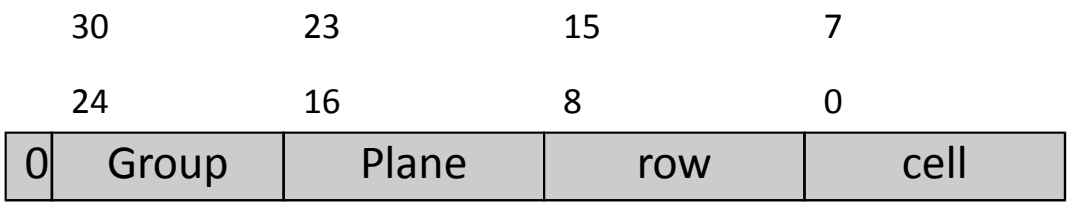
Ideographic description characters

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 OVERLAID

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.



Unicode

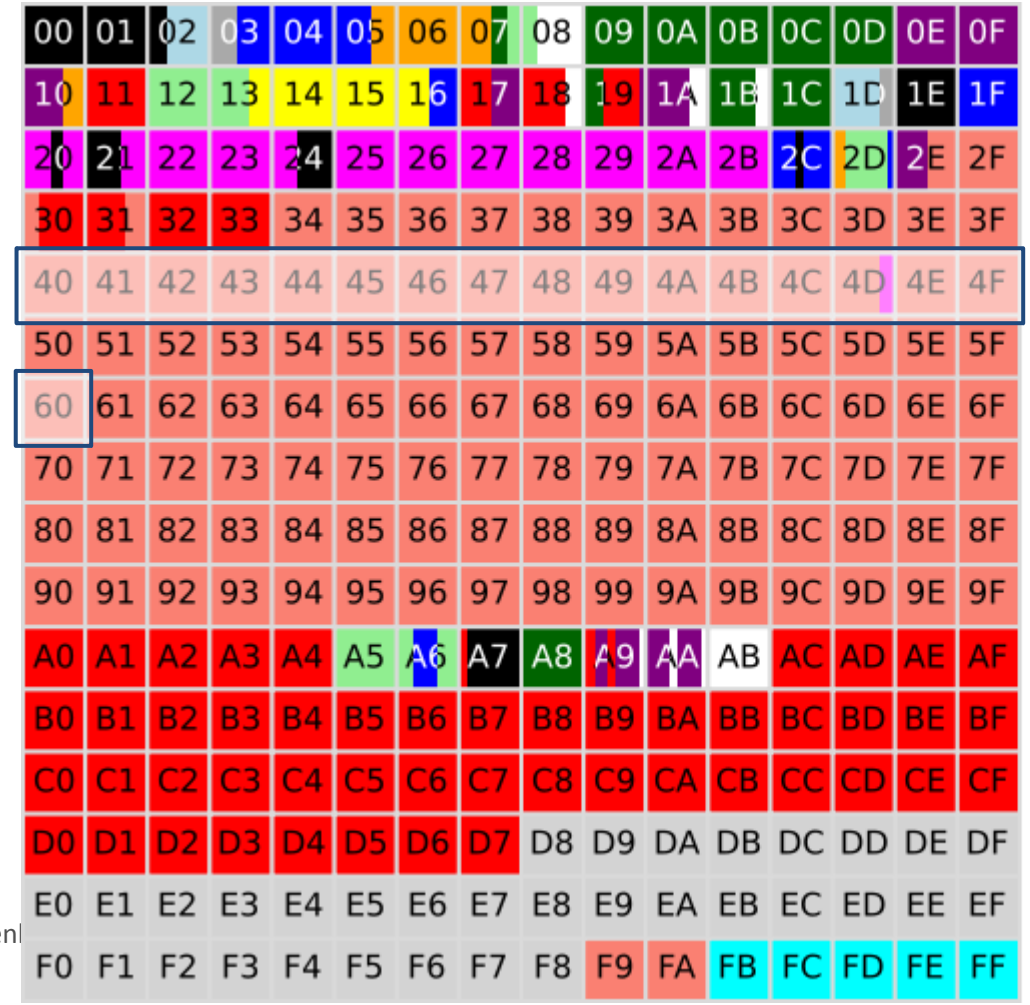
- Planes of the Universal Character Set (UCS)
  - Plane 0: Basic Multilingual Plane (BMP) contains the common-use characters for all the modern scripts of the world as well as many historical and rare characters.
  - Plane 1: Supplementary Multilingual Plane (SMP) is dedicated to the encoding of lesser-used 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)

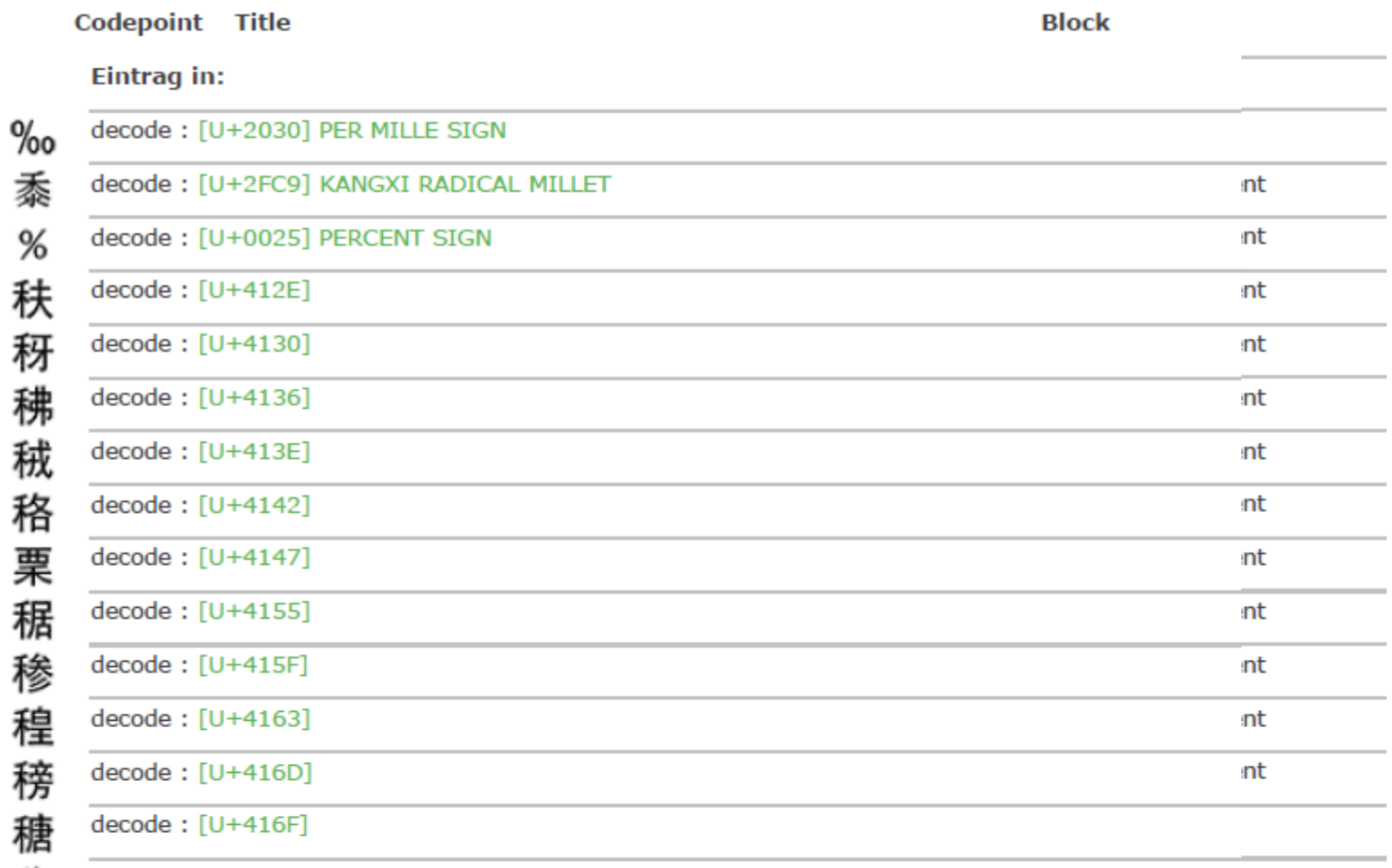


256 chars per cell!

http://de.wikipedia.org/wiki/Datei:Roadmap_to_Unicode_BMP.svg

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
秤	decode : [U+4130]	nt
稃	decode : [U+4136]	nt
稧	decode : [U+413E]	nt
稫	decode : [U+4142]	nt
稯	decode : [U+4147]	nt
稰	decode : [U+4155]	nt
稳	decode : [U+415F]	nt
稹	decode : [U+4163]	nt
稶	decode : [U+416D]	nt
穑	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

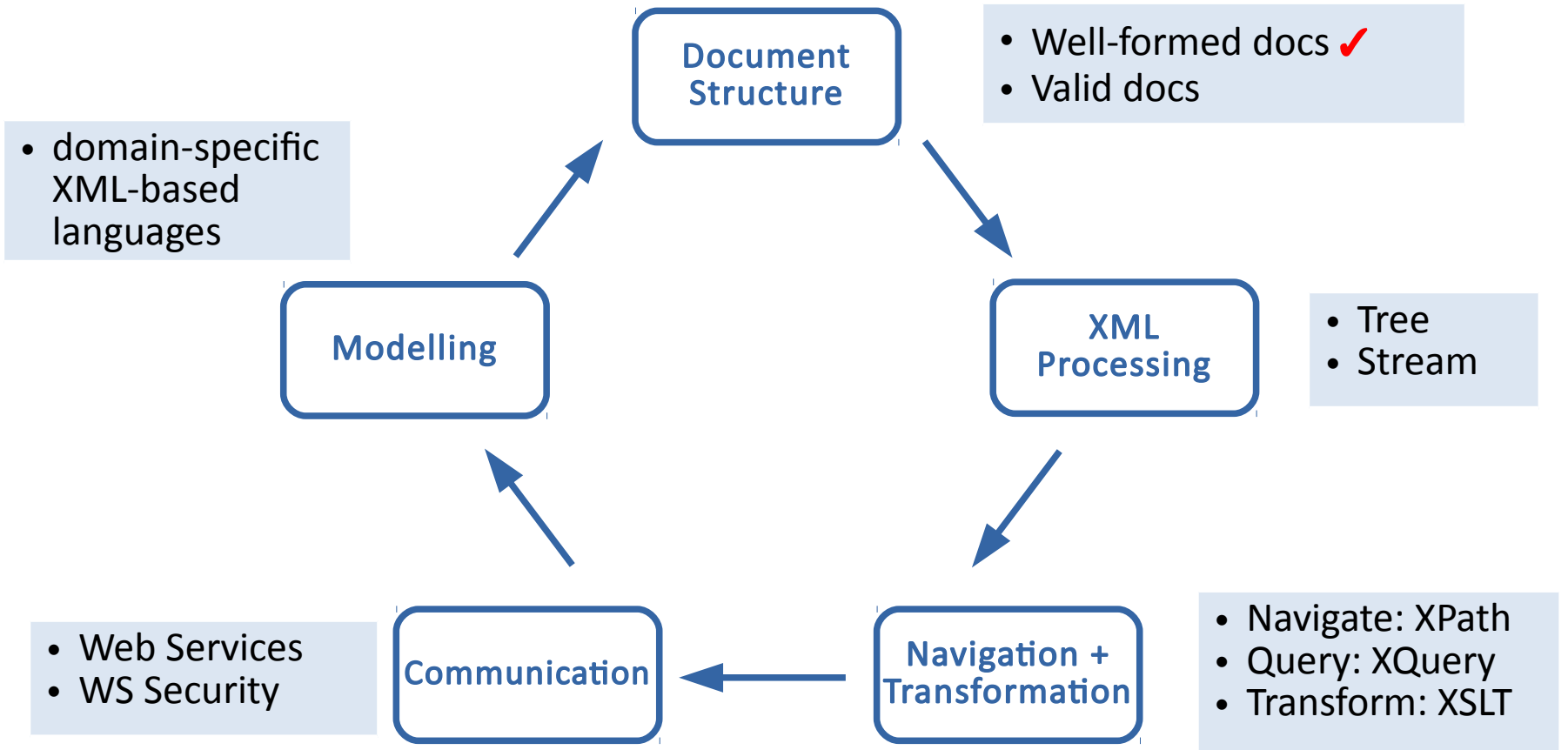
Unicode

- 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

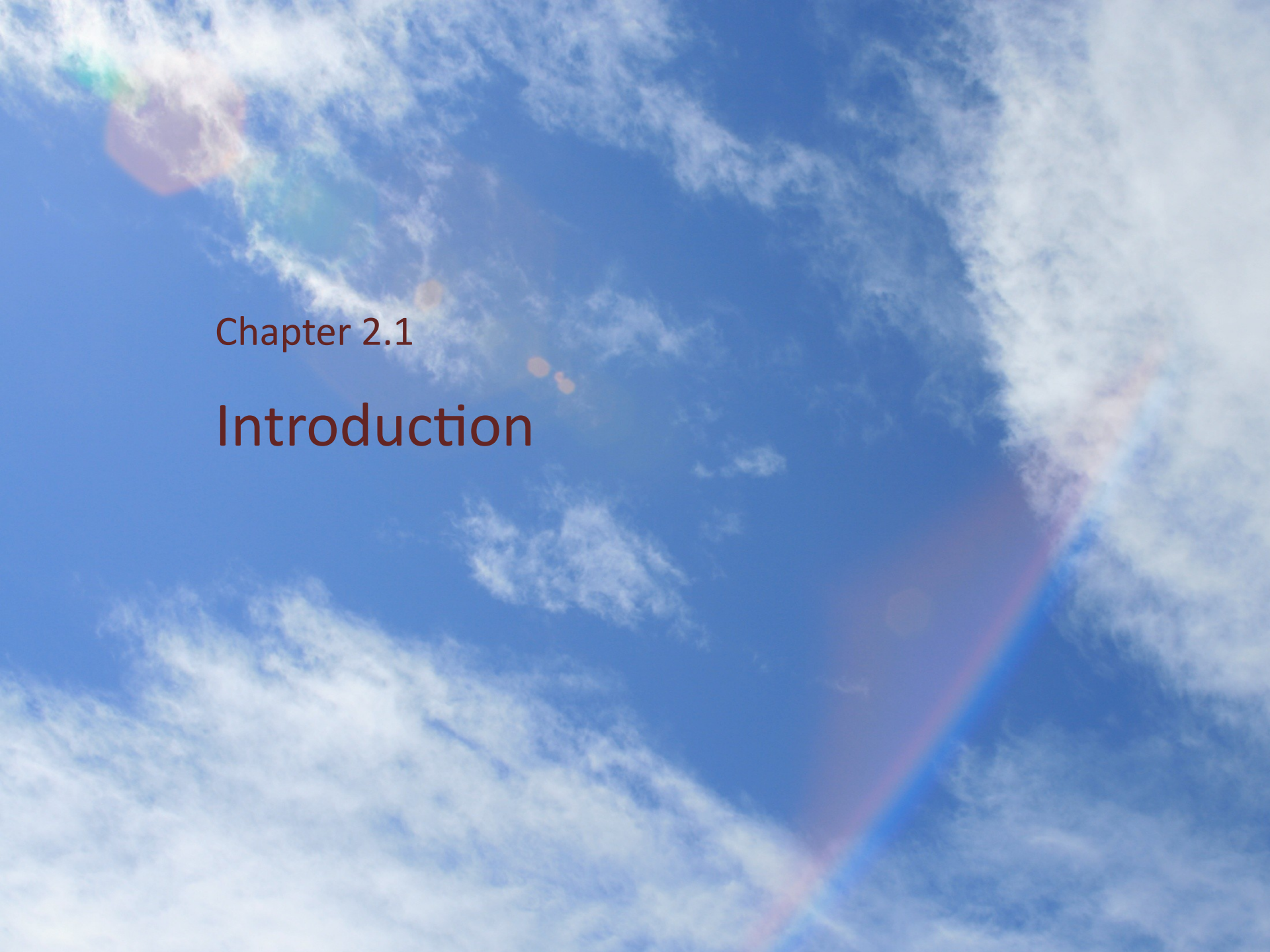
Recommended Reading

- 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

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.

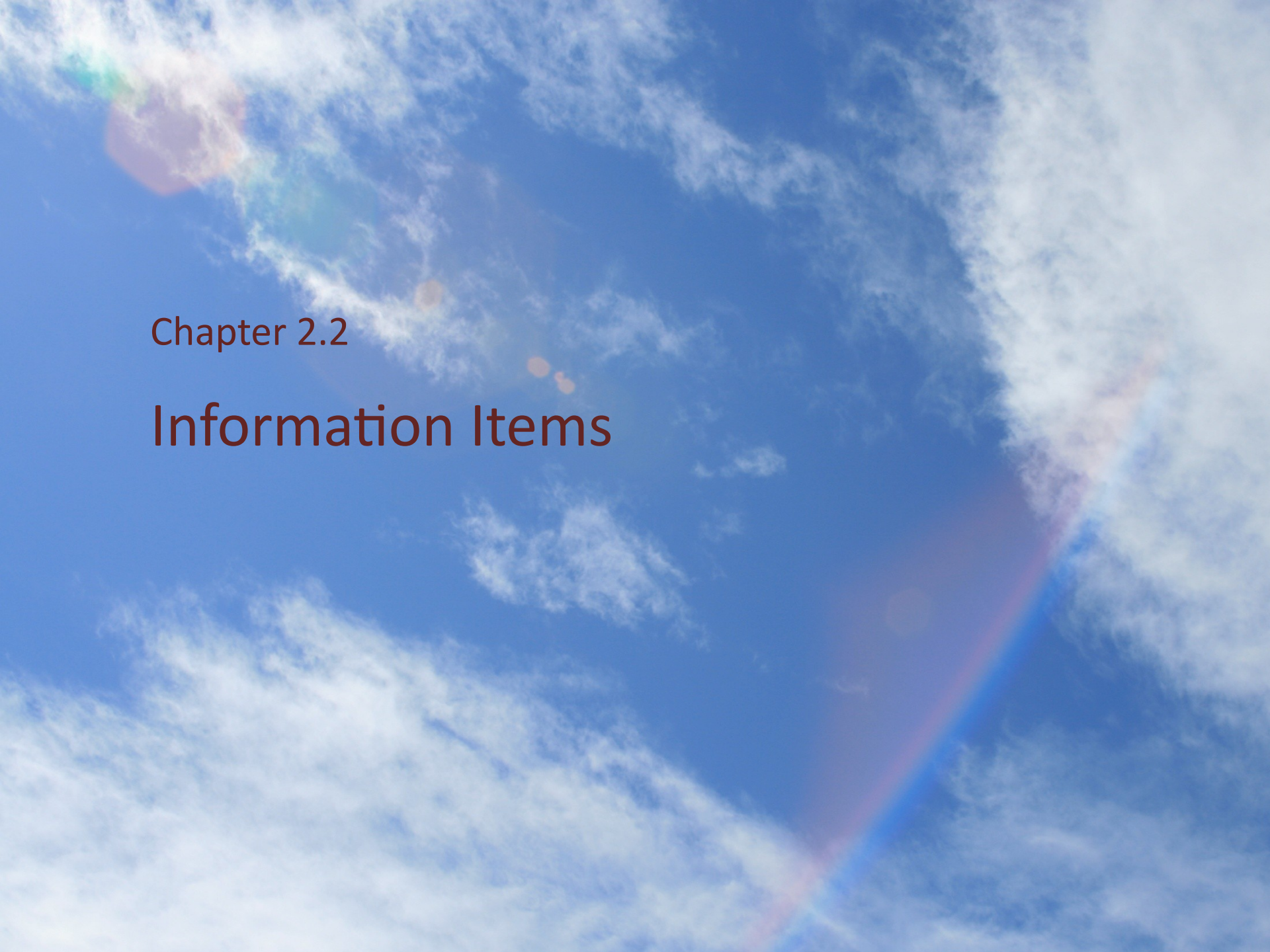
Introduction

- 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."

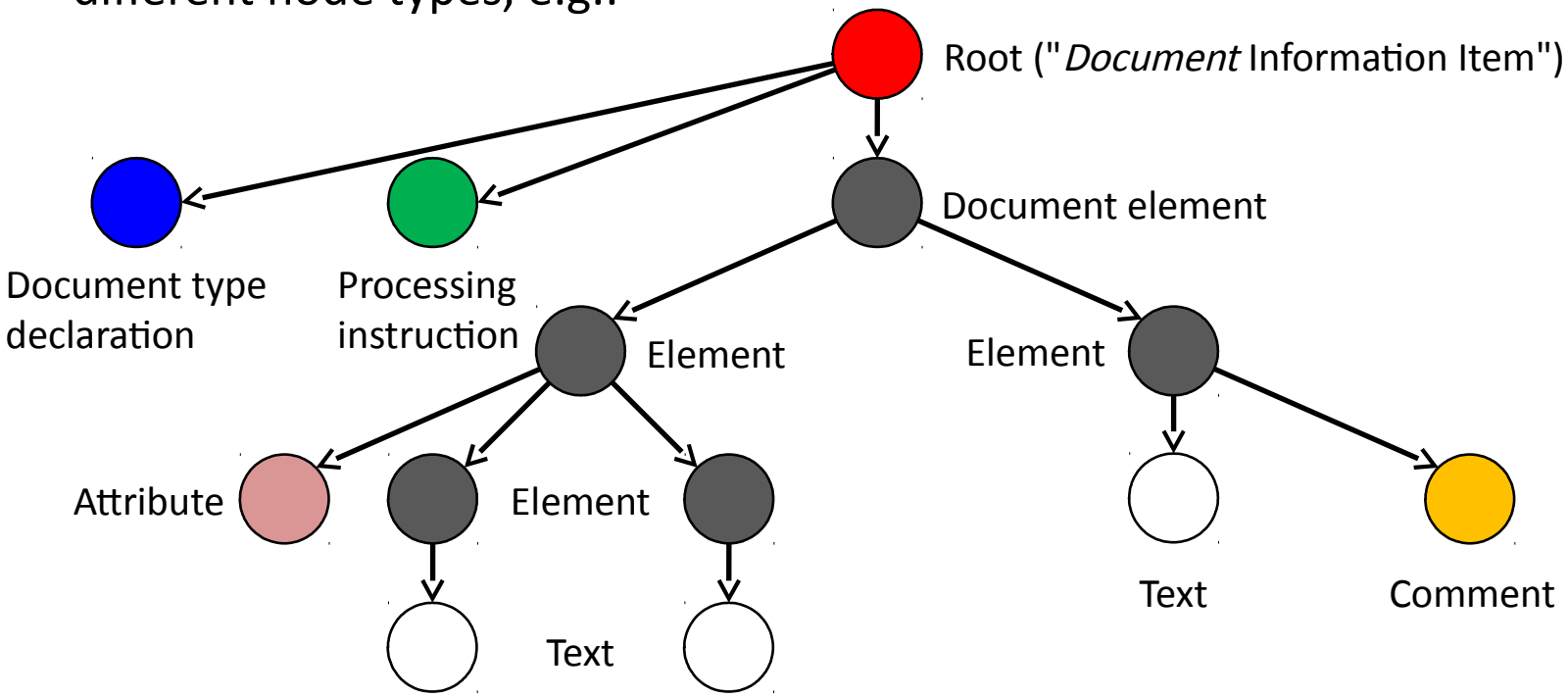


Chapter 2.2

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

- Attribute Information
- Processing Instruction
- Unexpanded Entity
- Character Information
- 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 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

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

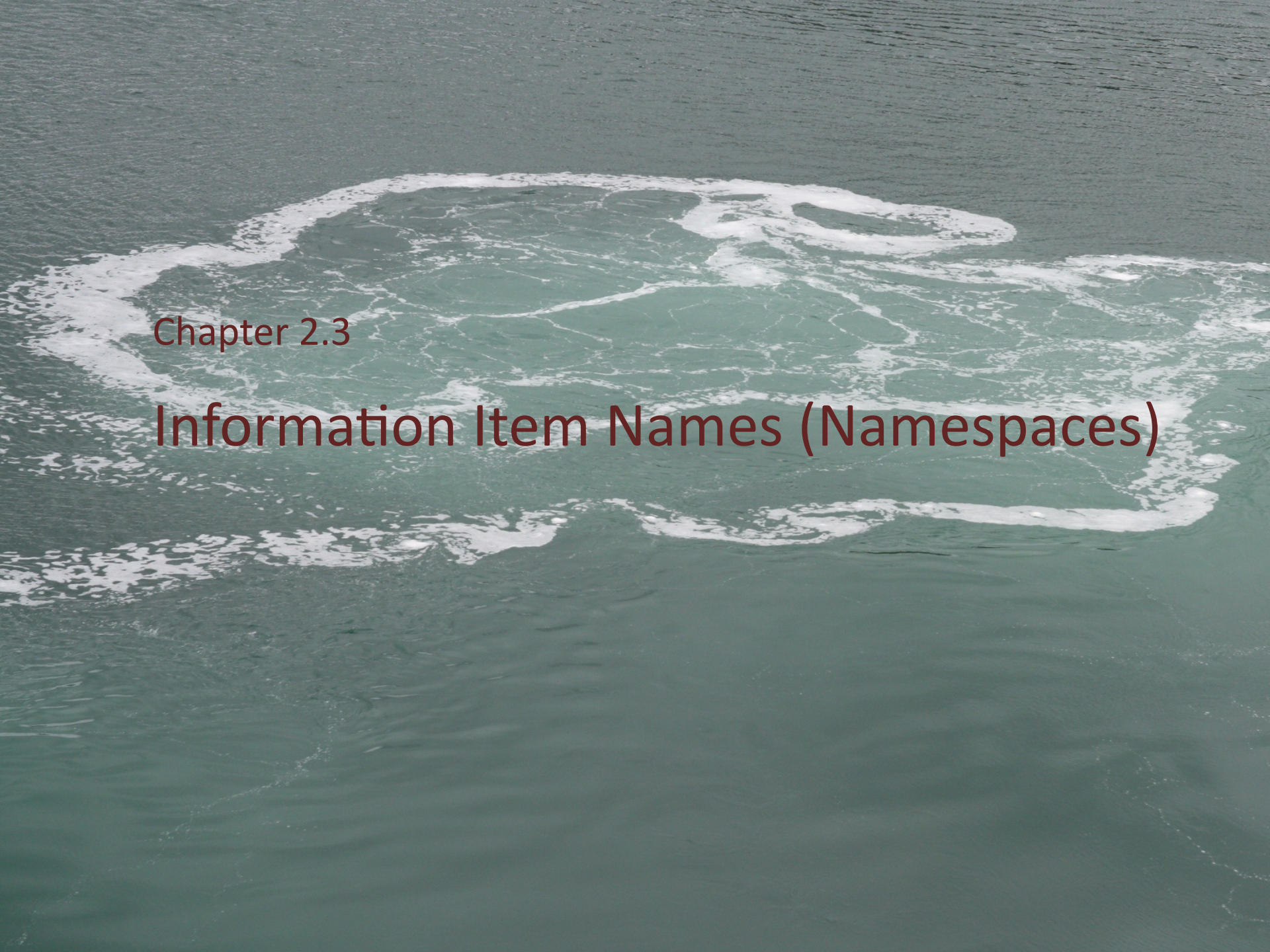
- Types of information items
  - Document Information Item
  - Element Information Items
  - Attribute Information Ite
  - Processing Instruction In
  - Unexpanded Entity Refer
  - Character Information Ite
  - Comment Information It
  - Document Type Declaration Information Item
  - Unparsed Entity Information Items
  - Notation Information Items
  - Namespace 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*.)

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 Informa
  - Notation Information Ite
  - Namespace Information Items

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



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" encoing="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
         <Book>
            <ISBN>0743204794</ISBN>
            <author>Kevin Davies</author>
            <title>Cracking the Genome</title>
            <price>20.00</price>
         </Book>
      </p>
   </body>
</html>
```

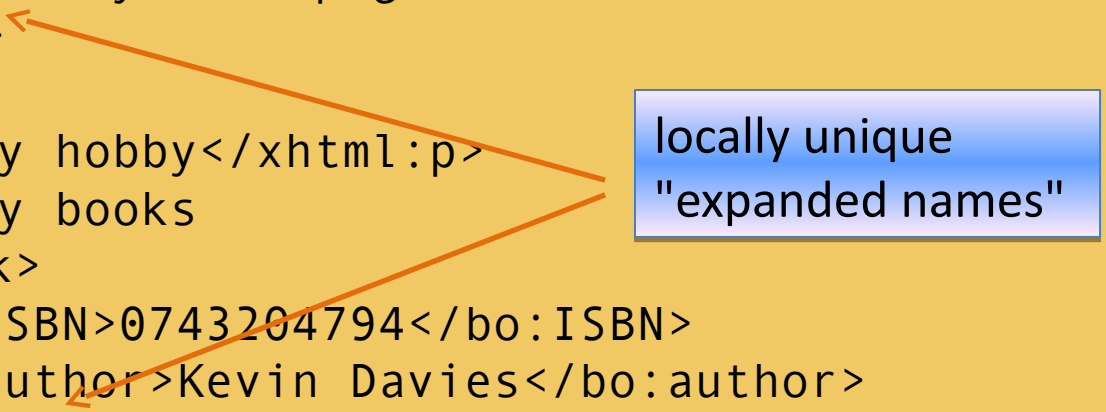
Remark: This is called "mixed content"—text content and element content

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

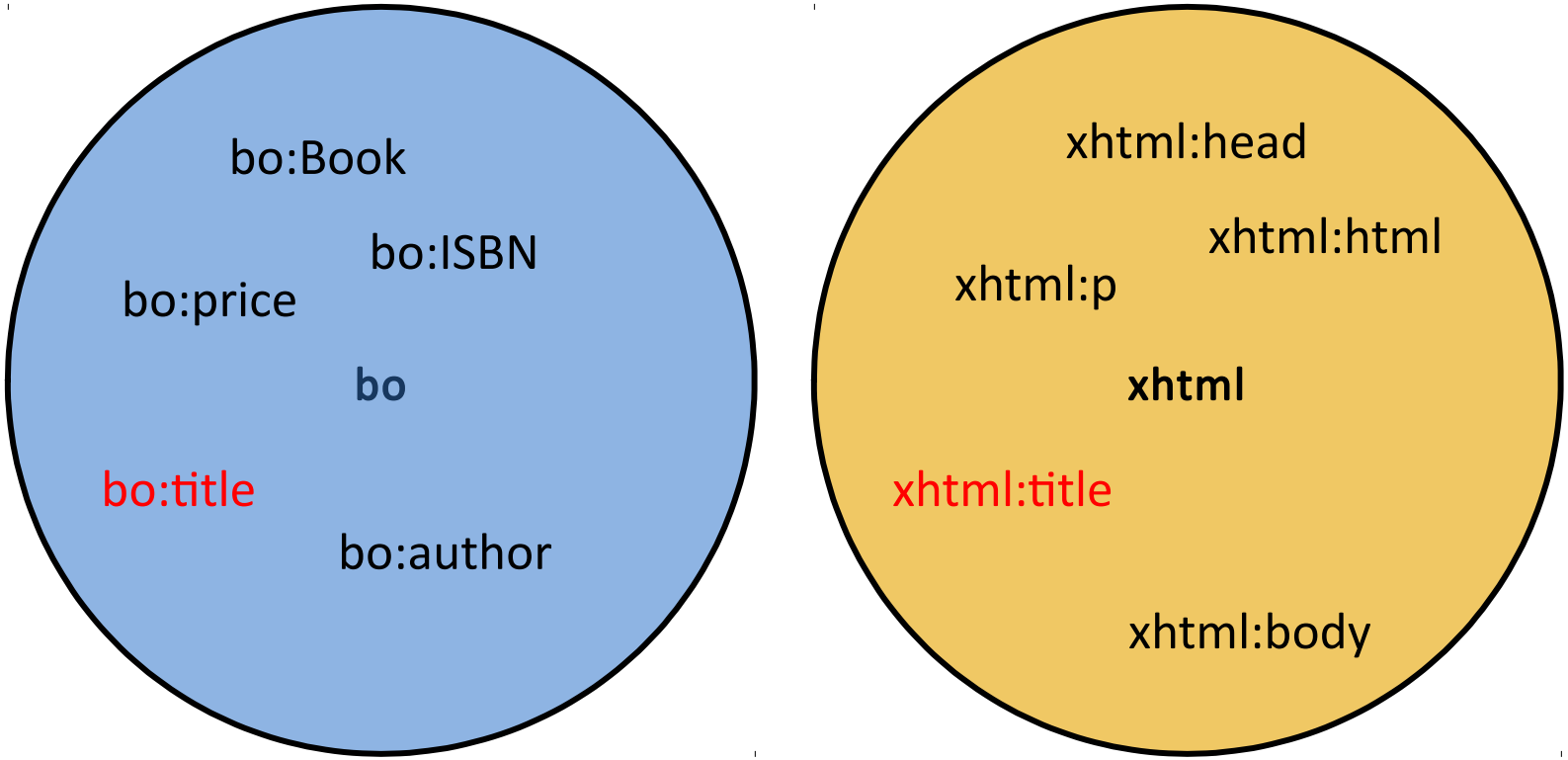
Namespaces

- 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>
```



Namespaces



vocabulary bo

vocabulary xhtml

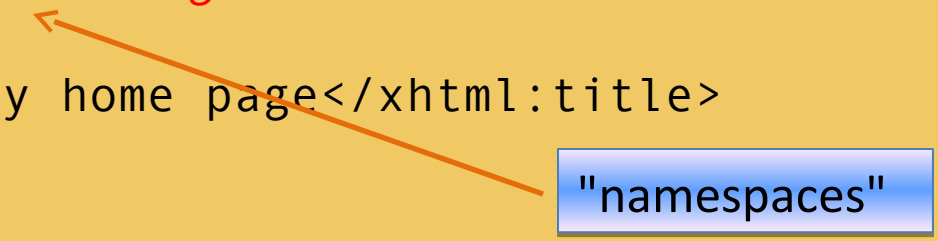
- But who guarantees uniqueness of prefixes?

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>
```



```
<?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

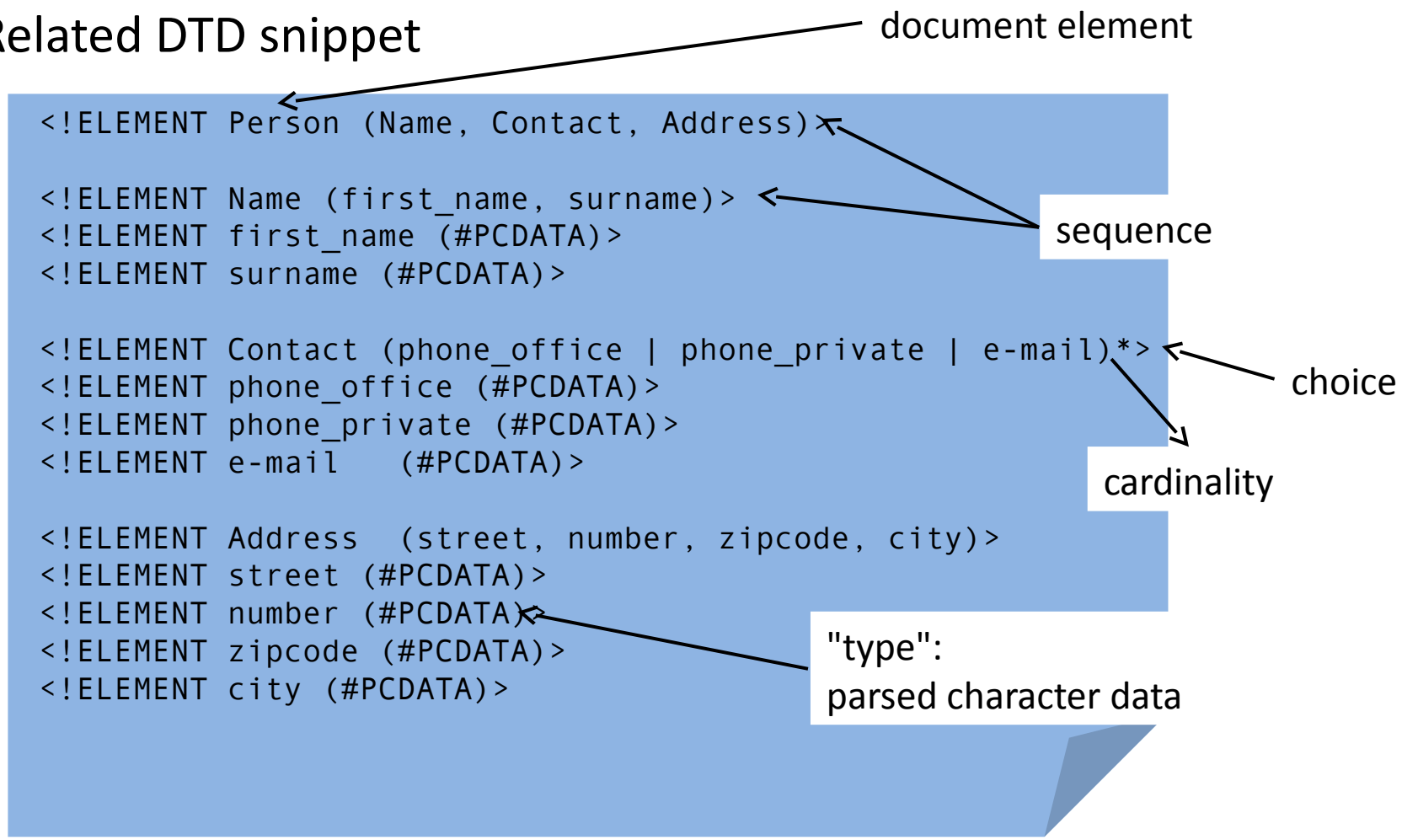
Document Type Definition

- 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



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>
      ...
```

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

- This chapter is based on:

  Roger L. Costello: XML Technologies Course

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

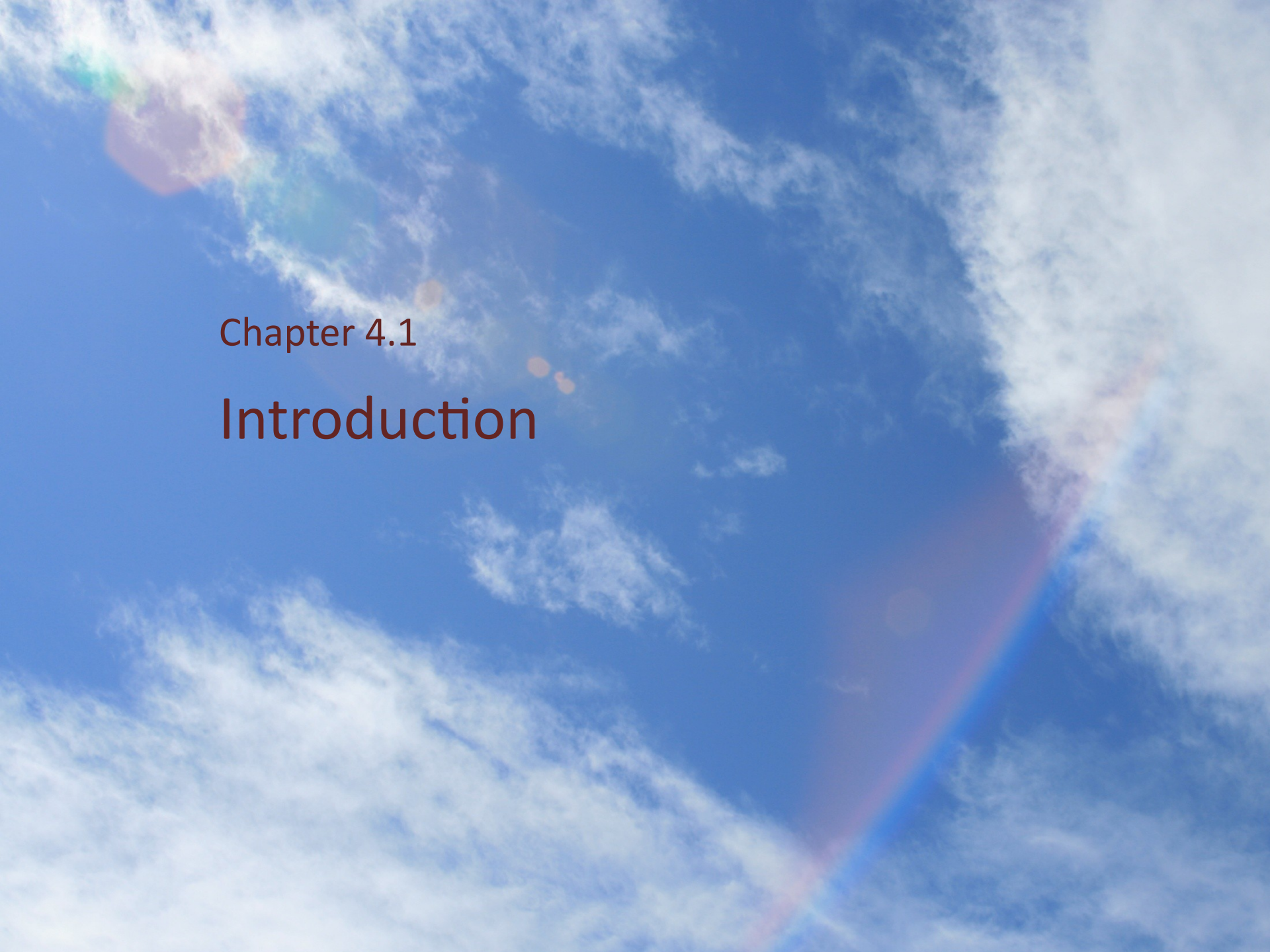
Recommended Reading

- 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

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

- 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".

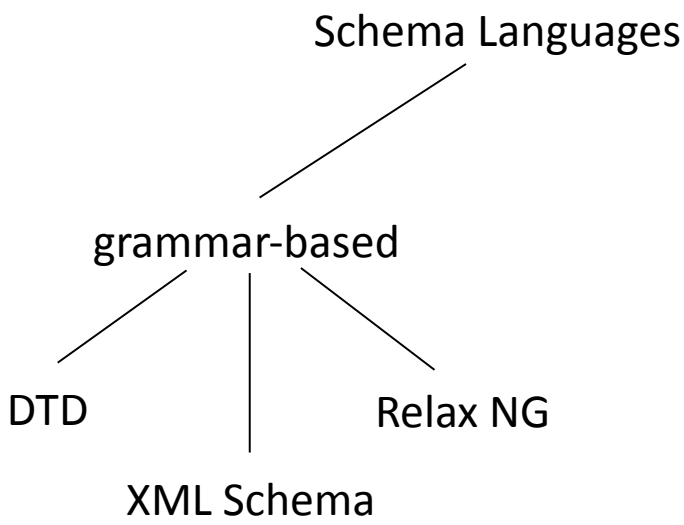
Introduction

- What is XML Schema?

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

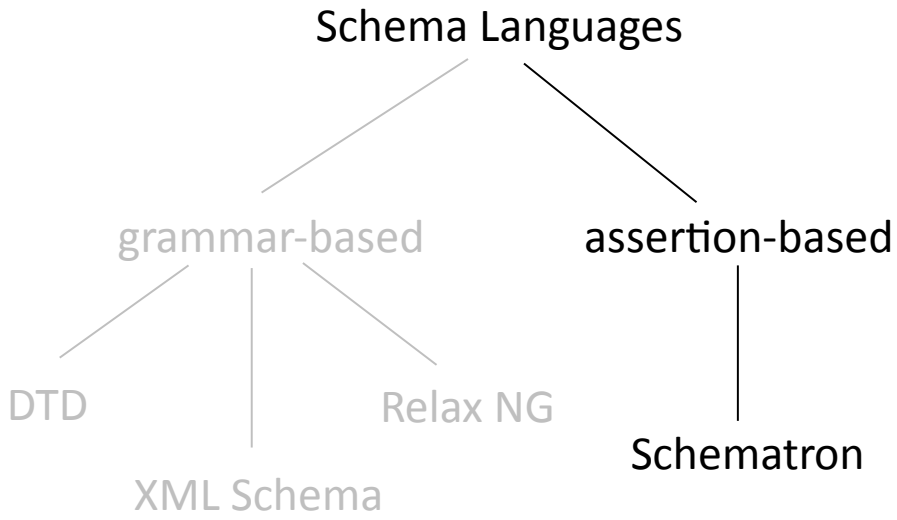
Introduction

- 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

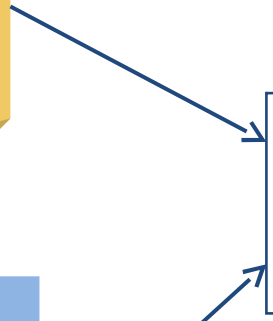
- To be valid, an instance document must meet the following constraints:

  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>
```

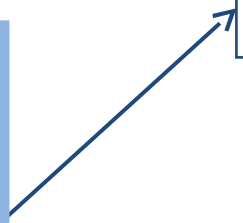
XML instance doc



XML Schema validator

Ok!

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.



Informal grammar
→ XML Schema

Introduction

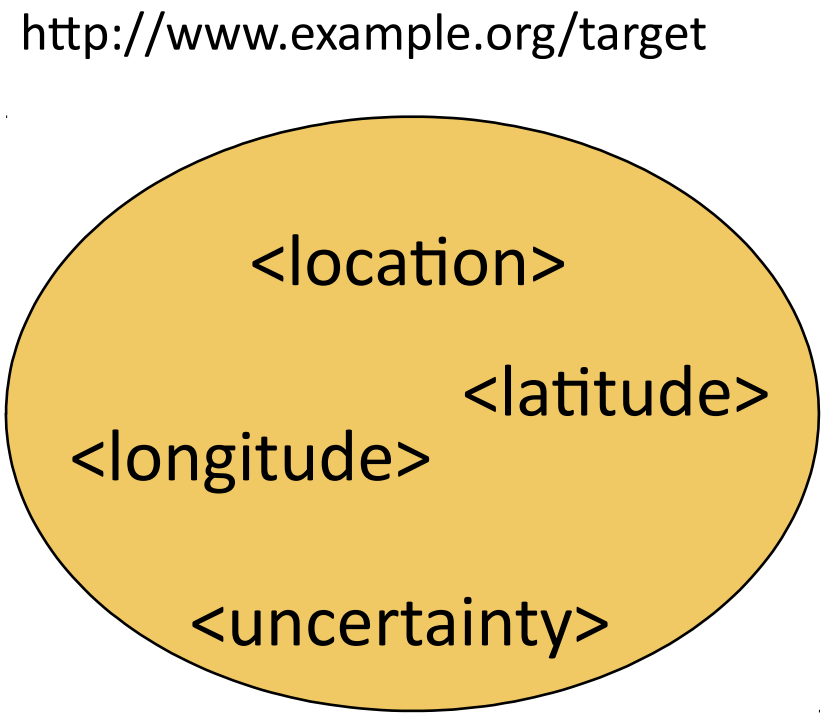
- What does an XML Schema accomplish?
  - Answer: It creates an XML vocabulary:

<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



Introduction

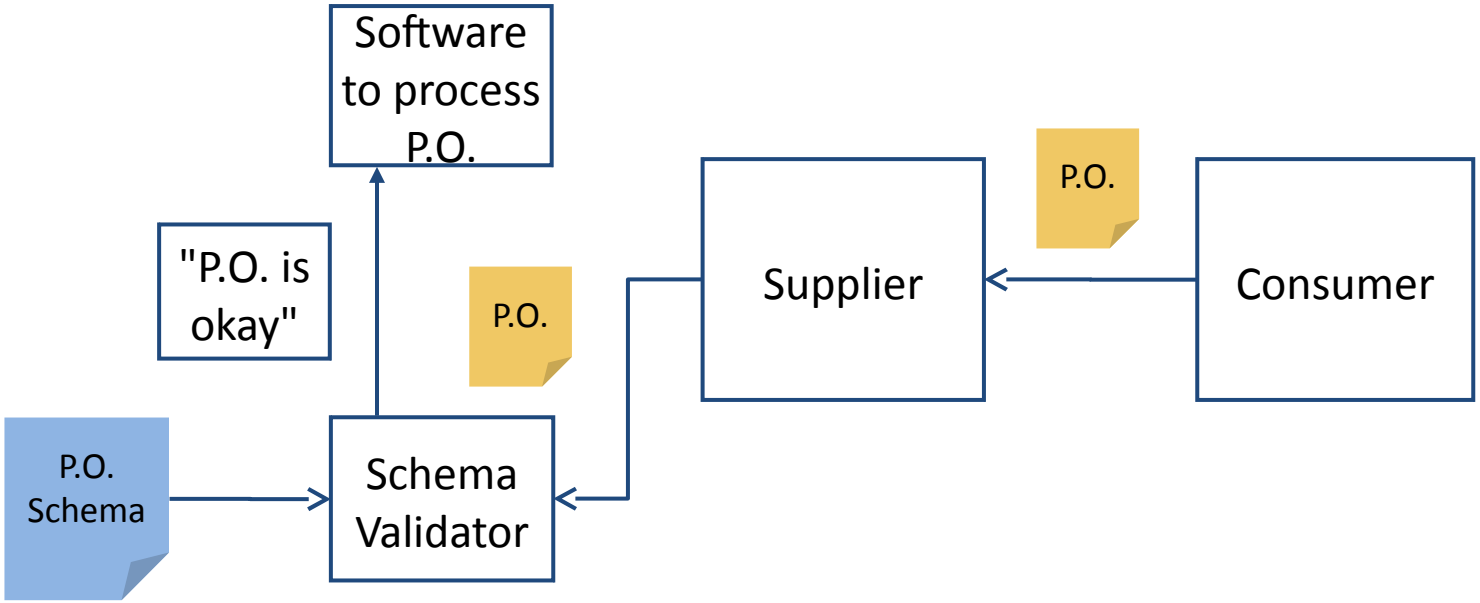
- Constraints: in a document or in code?
  - An XML Schema is an XML document.
    The constraints on the data are expressed in a document.
  - All of the constraints could be expressed in a programming language as well in your system's middleware.

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!

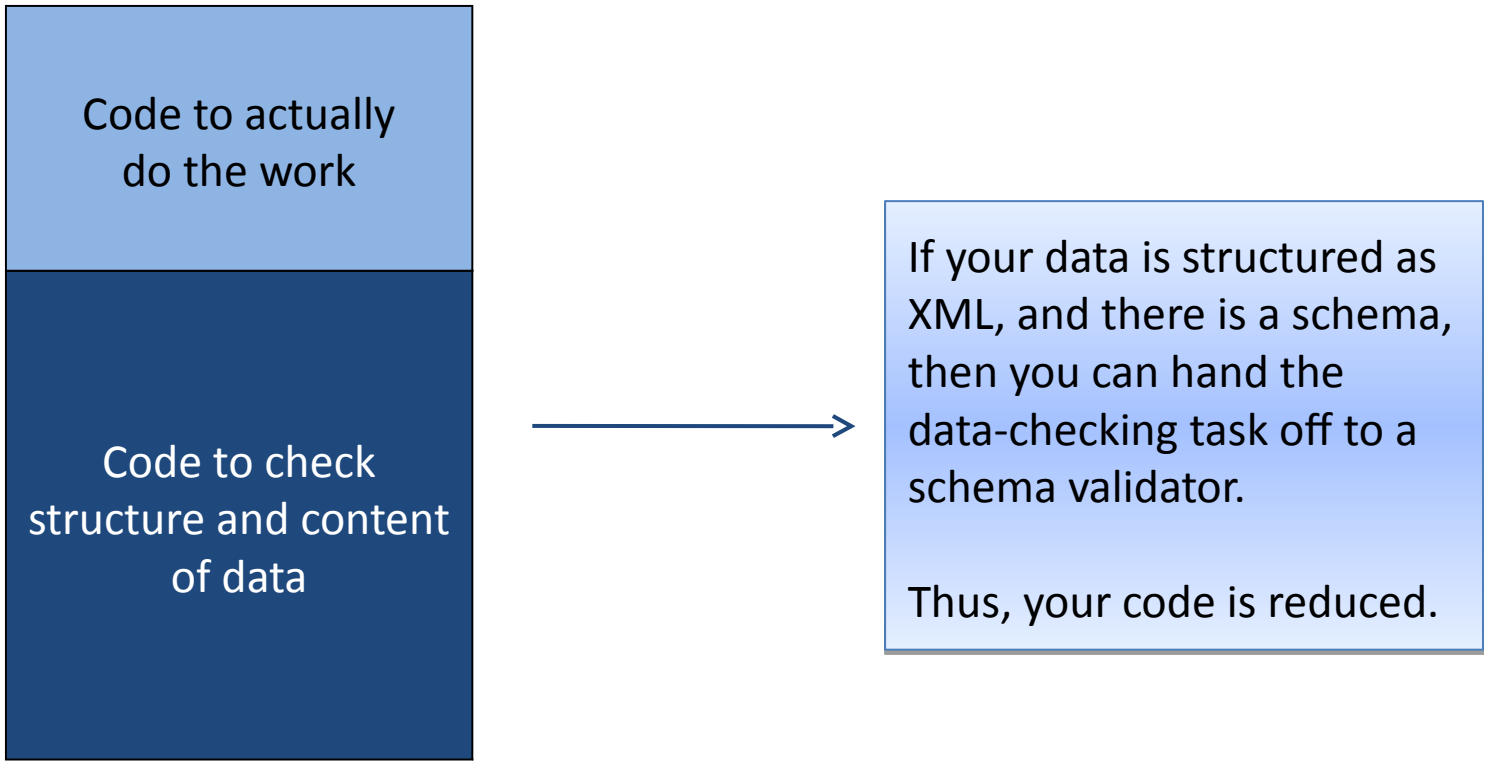
Introduction

- 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

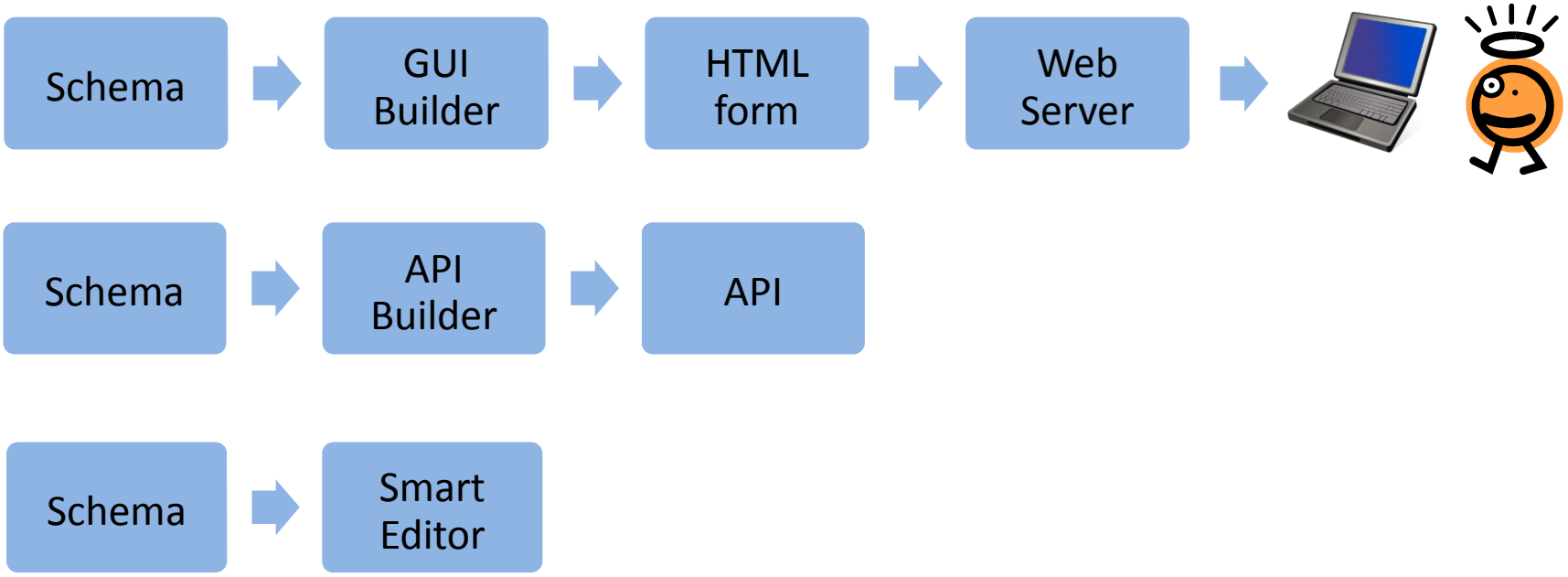
- XML Schema is a tremendous advancement over DTDs
  - Can express sets, i.e., can define the child elements to occur in any order
  - Can specify element content as being unique (keys on content) and uniqueness within a region
  - Can define multiple elements with the same name but different content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

Highlights of XML Schema

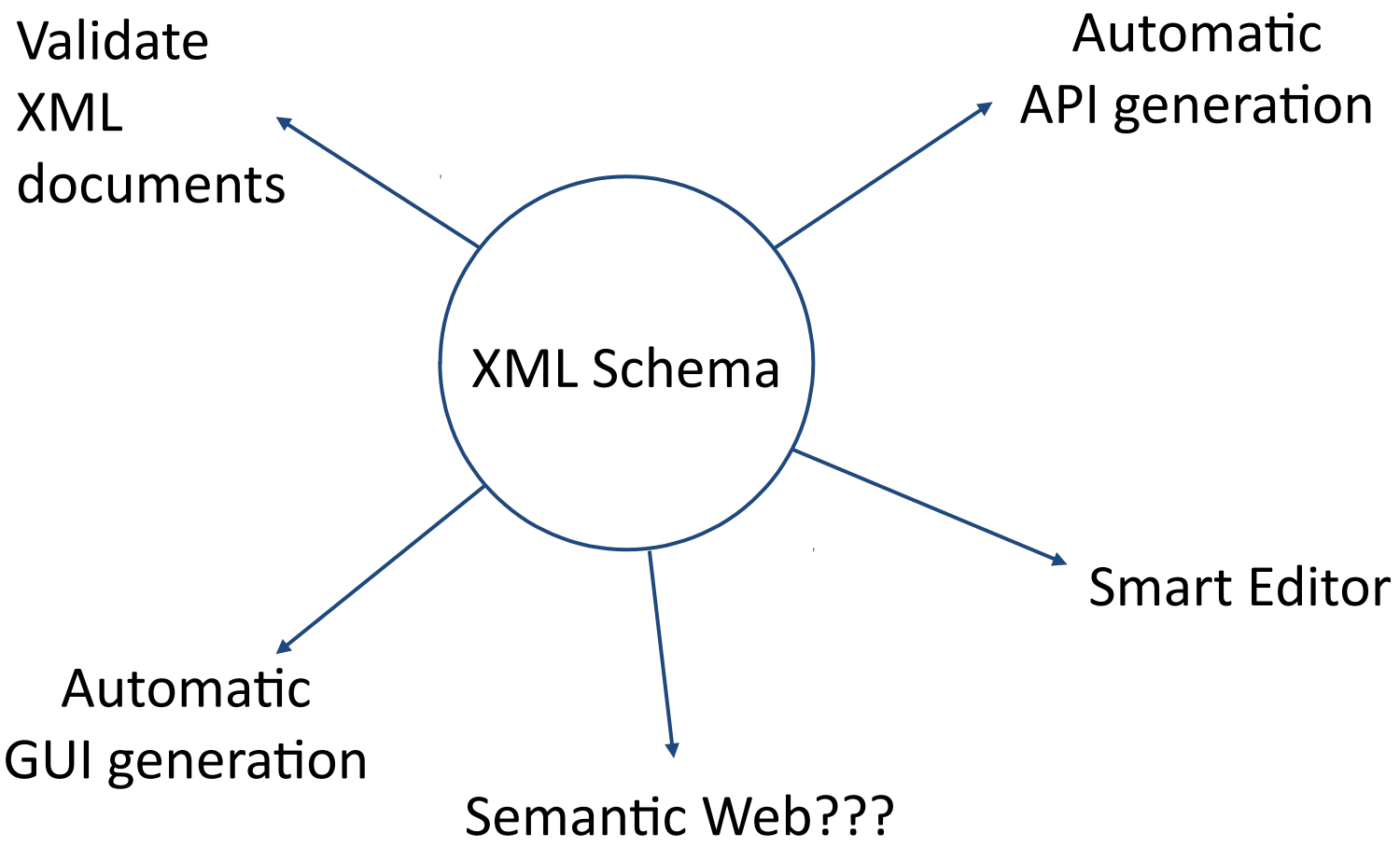
- XML Schema is a tremendous advancement over DTDs
  - Can define elements with nil content
  - Can define substitutable elements, e.g., the "Book" element is substitutable for the "Publication" element.

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



Chapter 4.2

Getting started

Let's Get Started!

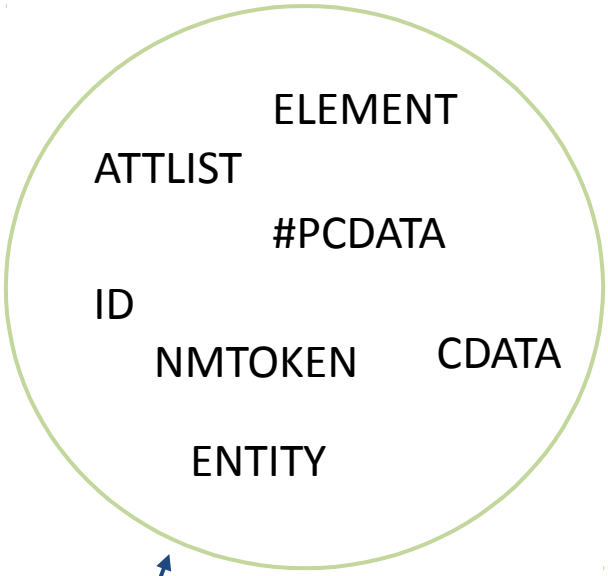
- Convert the `BookStore.dtd` to the XML Schema syntax

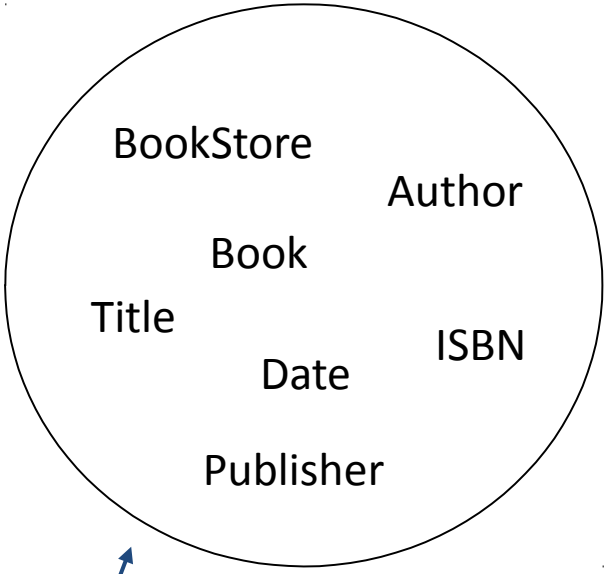
```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```

  - 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

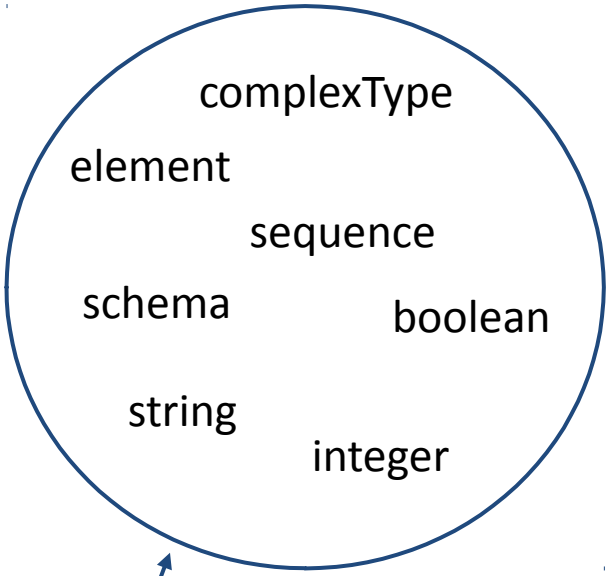




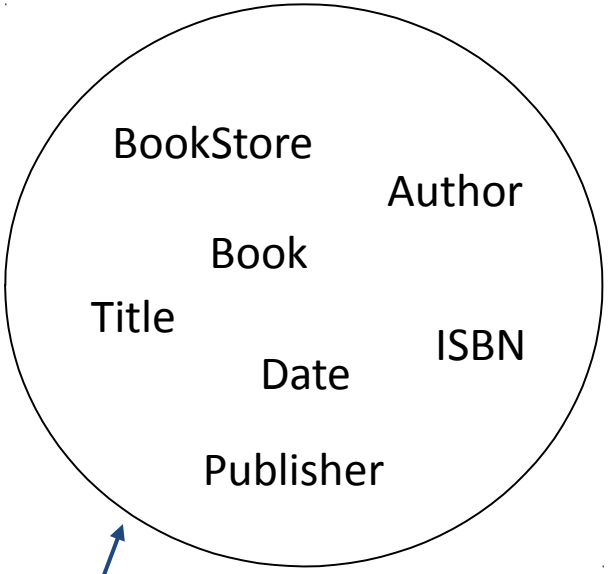
This is the vocabulary that DTDs provide to define your new vocabulary

Bookstore Schema

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



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



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].

BookStore.xsd

```
<?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>
```

```
<?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>
```

<!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)>

```
<?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>
```

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>
```

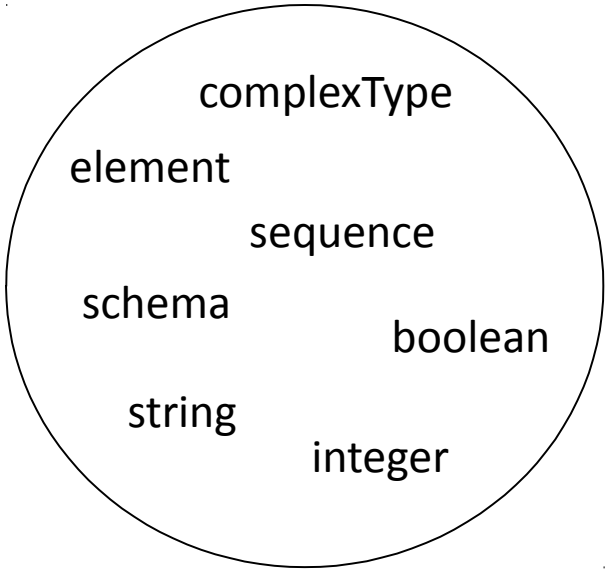
The elements and datatypes that are used to construct a schema, e.g.
- schema
- element
- complexType
- sequence
- string

come from the http://…/XMLSchema namespace.

Bookstore Schema

- XMLSchema Namespace

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



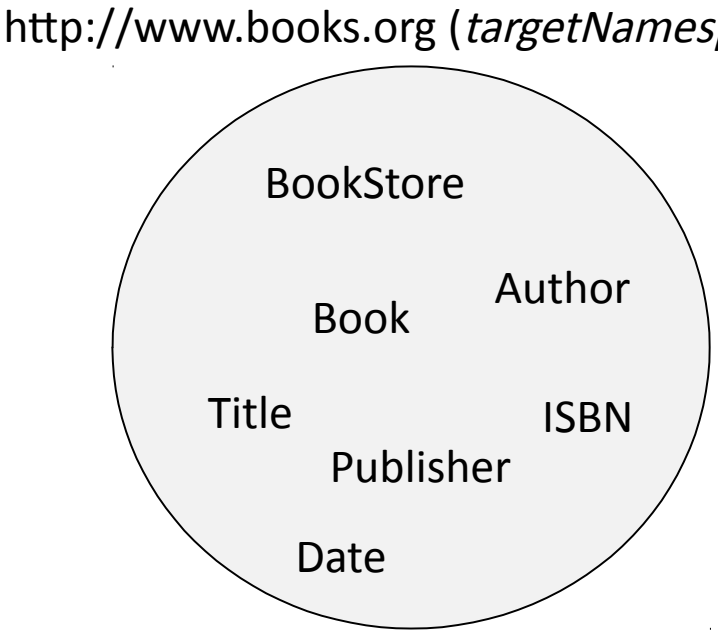
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema
        targetNamespace="http://www.books.org"
        xmlns="http://www.book
        elementFormDefault="qu                d">
   <xsd:element name="BookStor
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Bo          inOccurs="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>
```

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)



```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.books.org"
        xmlns="http://www.books.org"
        elementFormDe[illegible]="qualified">
   <xsd:element name=[illegible]tore">
      <xsd:complexTyp[illegible]
         <xsd:sequence[illegible]
            <xsd:eleme[illegible]="Book" minOccurs="1" maxOccurs="unbounded"/>
         </xsd:sequenc[illegible]
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Book">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Title" minOccurs="1" maxOc[illegible]
            <xsd:element ref="Author" minOccurs="1" maxO[illegible]
            <xsd:element ref="Date" minOccurs="1" maxOcc[illegible]
            <xsd:element ref="ISBN" minOccurs="1" maxOcc[illegible]
            <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/XMLSchem
        targetNamespace="http://www.books.org"
        xmlns="http://www.books.org"
        elementFormDefault="qualified">
   <xsd:element name="BookSto
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="                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>
```

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.or
        targetNamespace="http://www.boo
        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 i

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.book
    <Book>
       <Title>
       <Author
       <Date>J
       <ISBN>9
       <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.

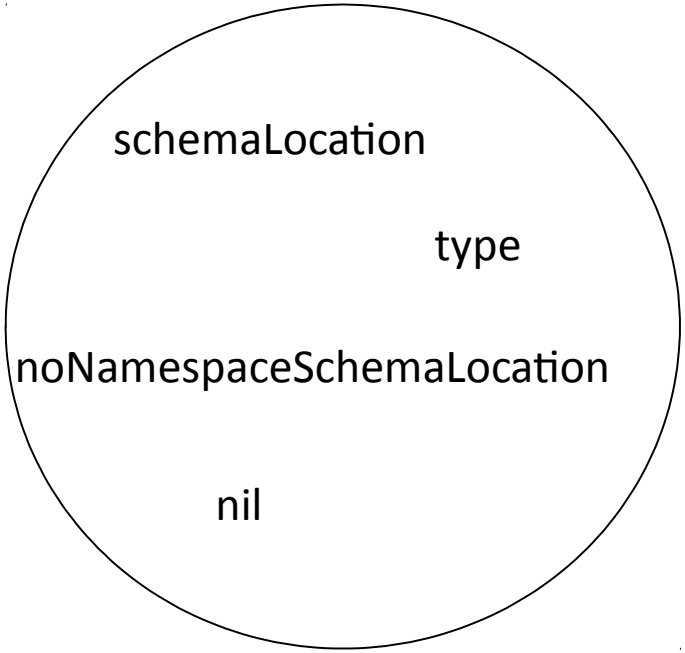
Tell the schema-validator that the schemaLocation attribute is in the XMLSchema-instance namespace.

Tell the schema-validator that the http://www.books.org namespace is defined by BookStore.xsd; schemaLocation attribute has a pair of values.

Bookstore Schema

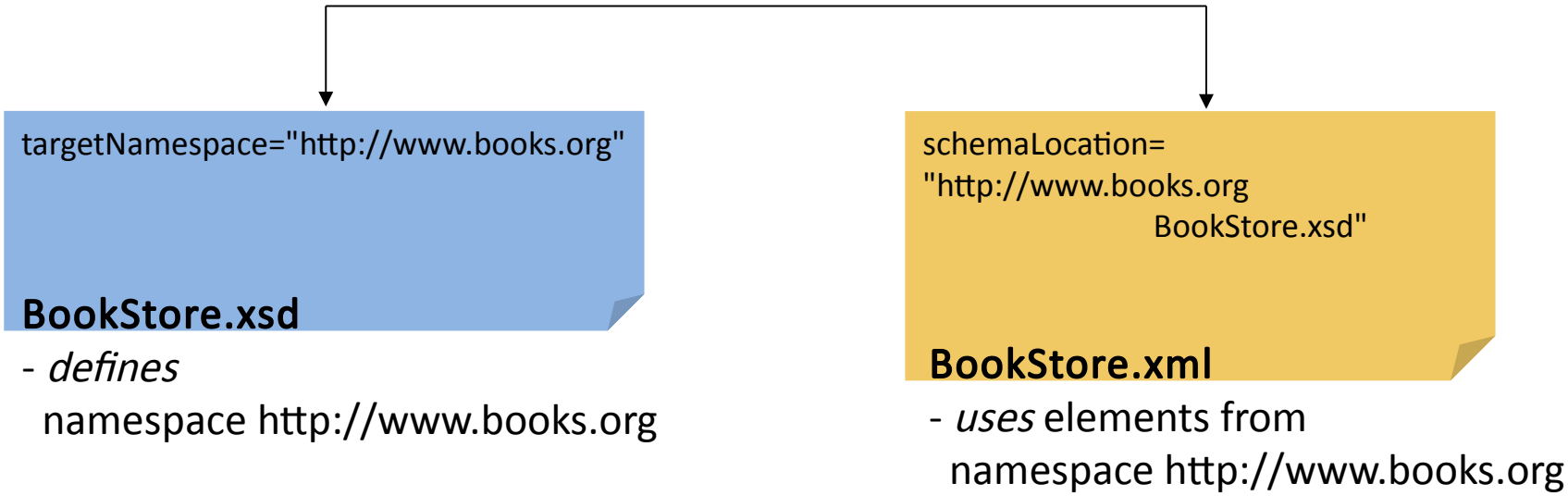
- XMLSchema-instance Namespace

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



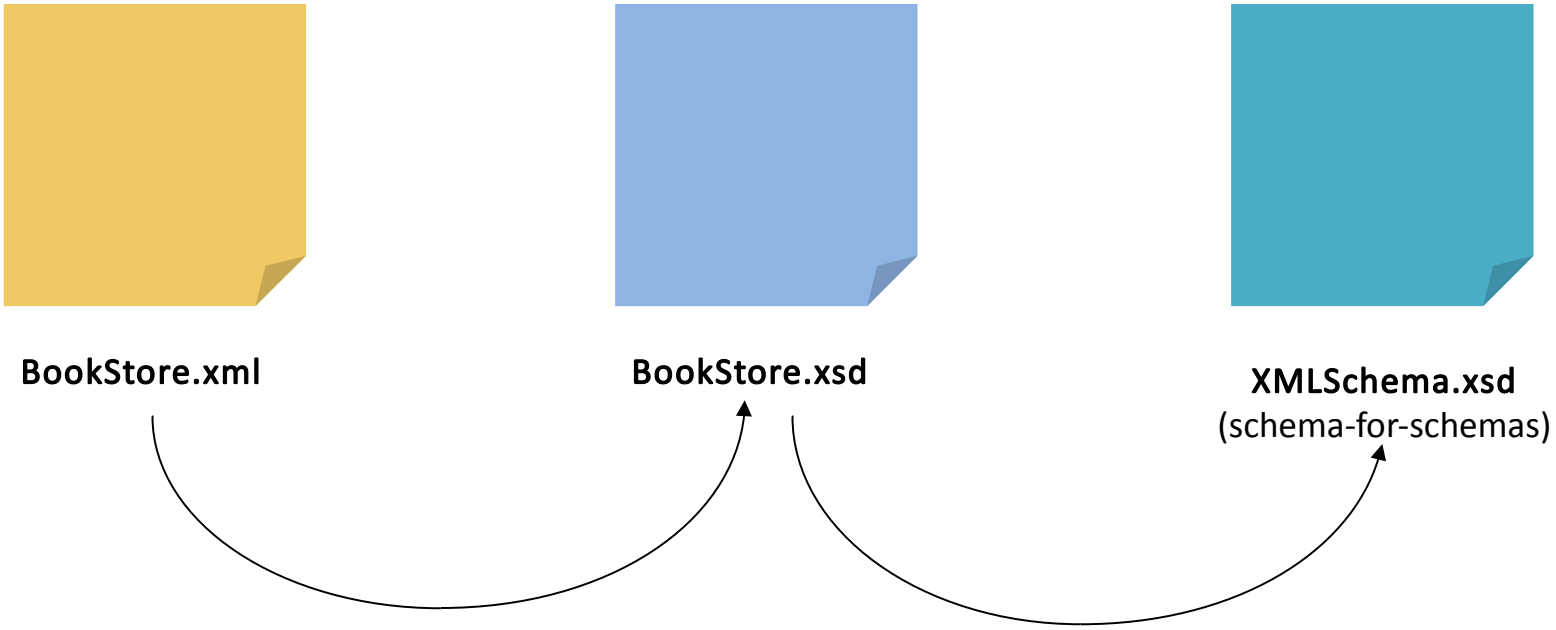
Bookstore Schema

- XML schema and instance document



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

Multiple Levels of Checking



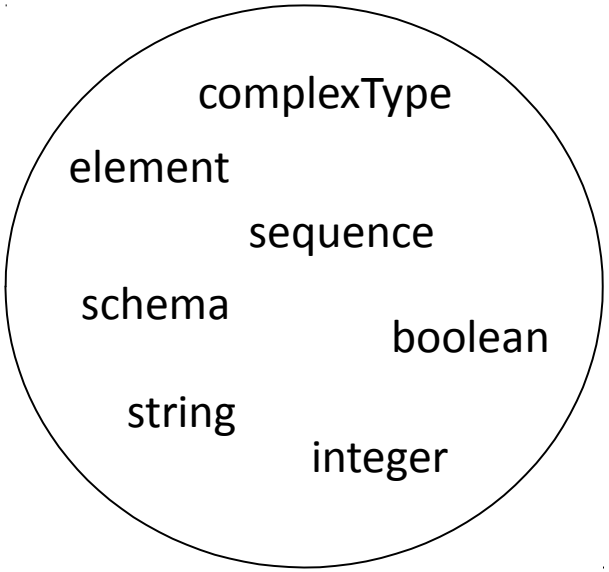
Validate that the xml document conforms to the rules described in BookStore.xsd

Validate that BookStore.xsd is a valid schema document, i.e., it conforms to the rules described in the schema-for-schemas

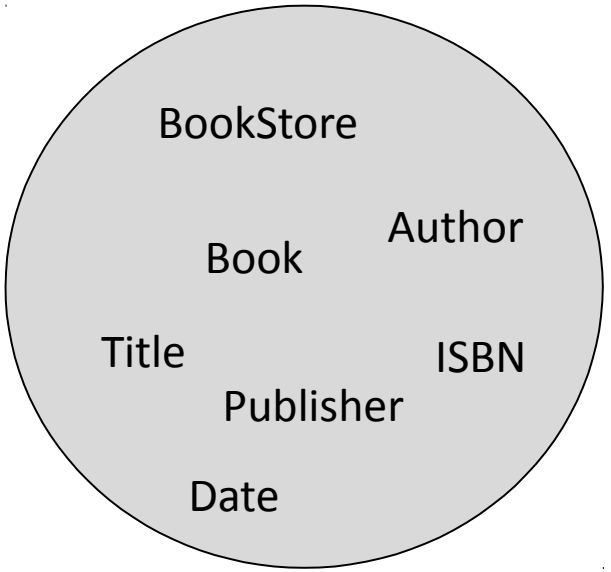
Namespaces Again

- targetNamespace may be the default namespace.

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



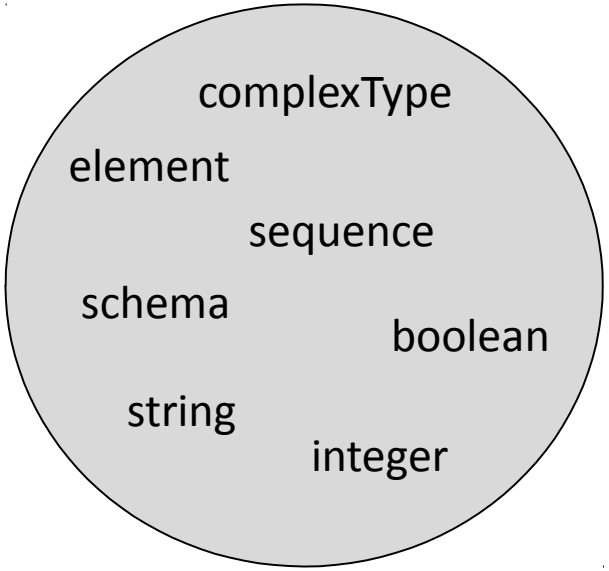
http://www.books.org (*targetNamespace*)



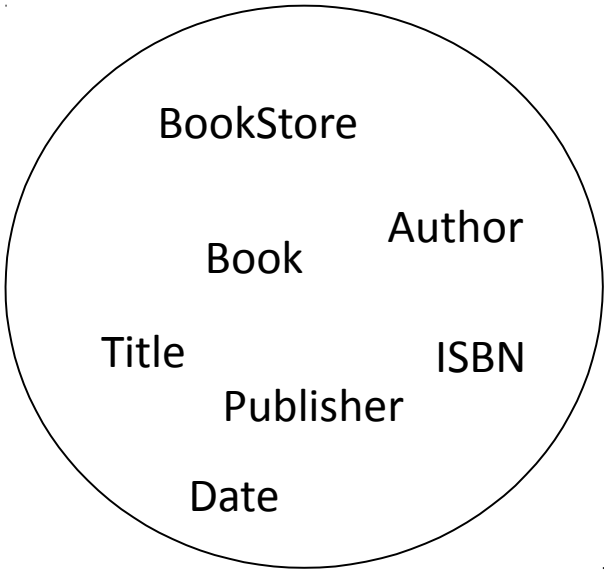
Namespaces Again

- 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*)



```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://w      ks.org"
                xmlns:bk="http://www.book
                elementFormDefault="quali
   <element name="BookStore">
      <complexType>
         <sequence>
            <element ref="bk:Book" minOccurs="1" maxOccurs="ur
         </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

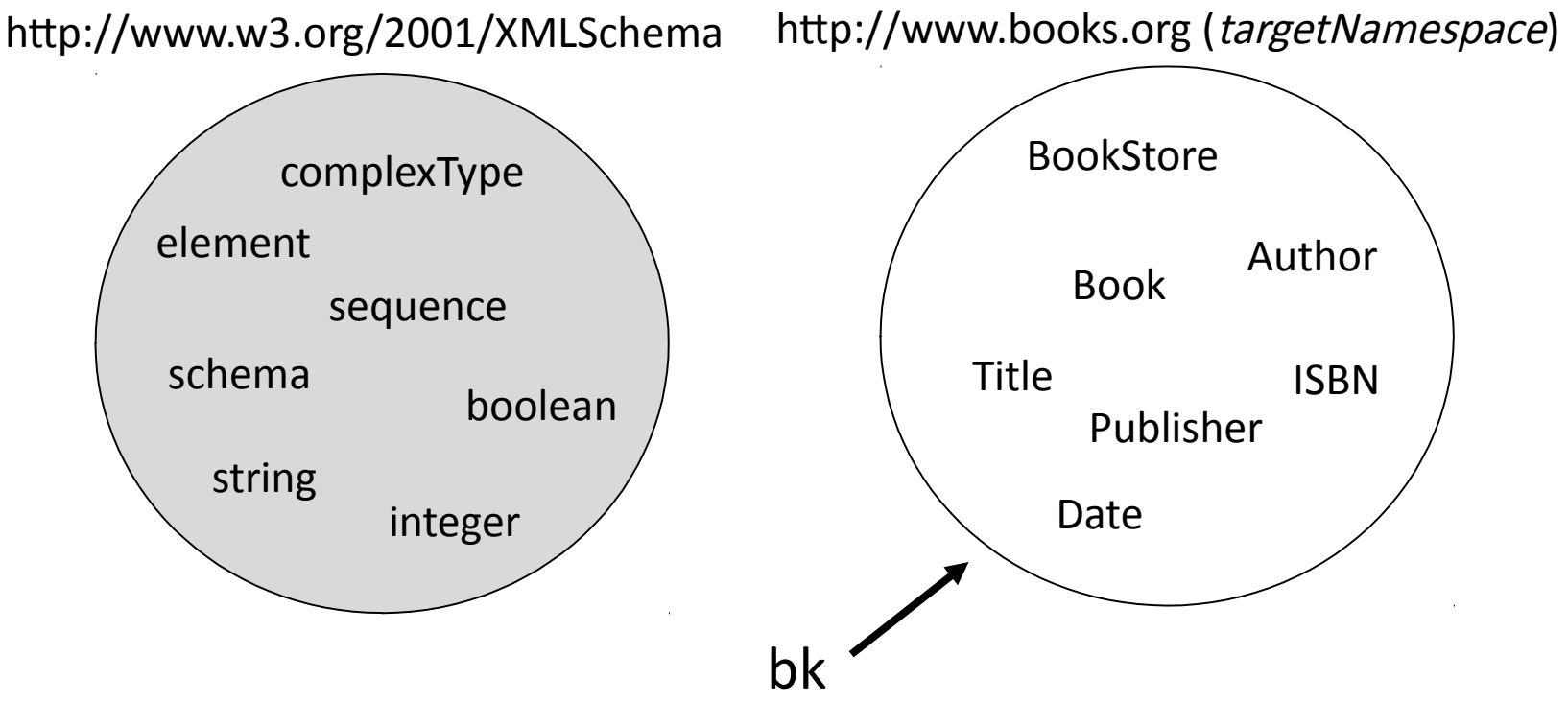
```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://www.books.org"
                xmlns:bk="http://www.books.org"
                elementFormDefault="q      ed">
   <element name="BookStore">
      <complexType>
         <sequence>
            <element ref="bk:Book" min      ="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>
```

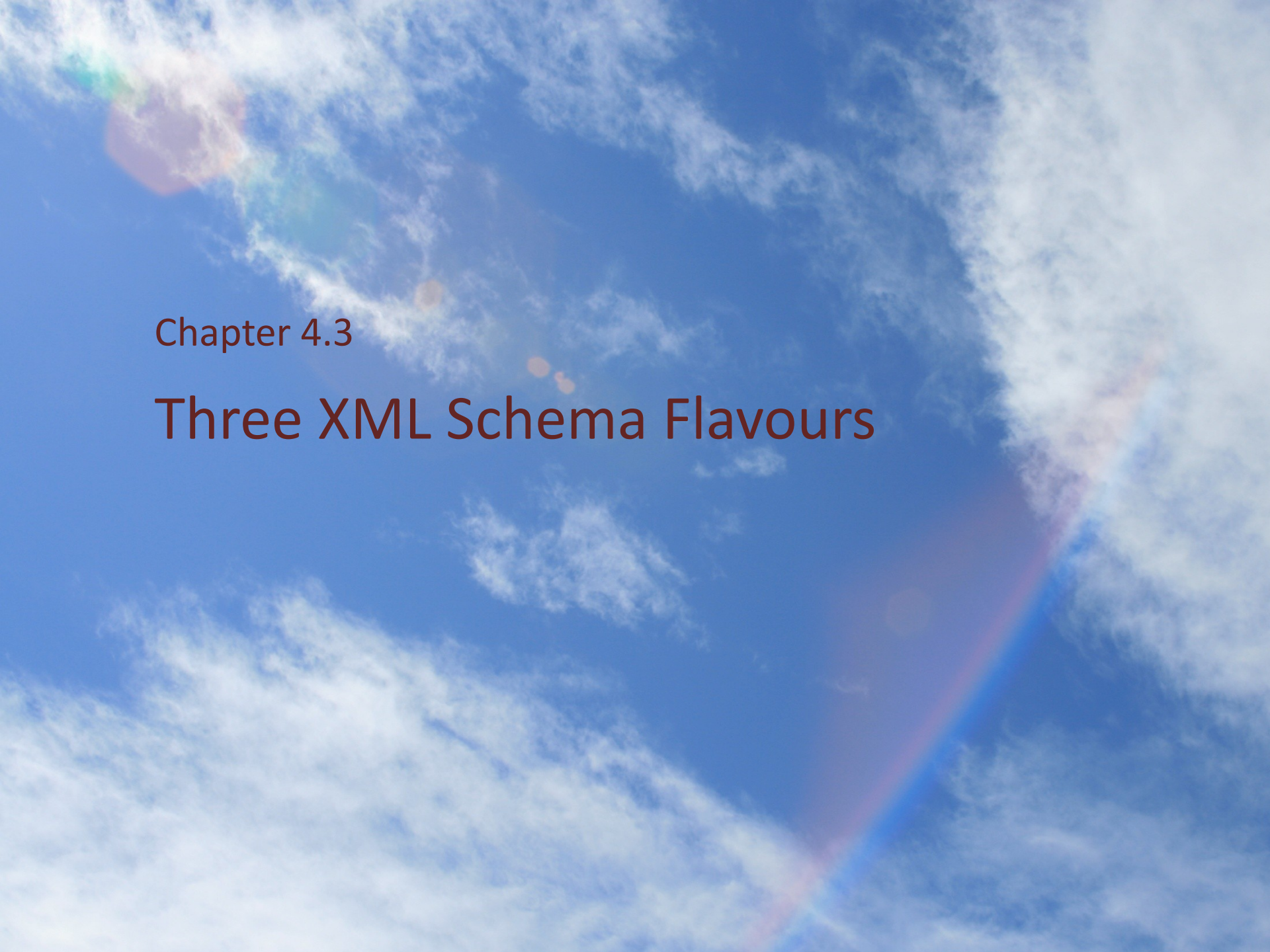
Here we are referencing a Book element. In what namespace is that Book element defined??

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

Namespaces Again

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





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!

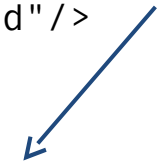
Embedded types

Types are anonymous.

```
<?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>
```

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>
```

Element A *references* the complexType foo.

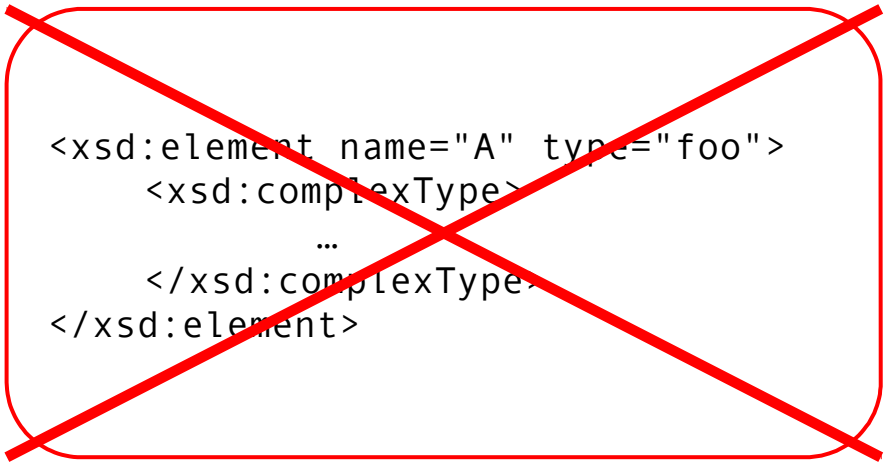
is equivalent to:

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

Element A 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>
```

```
<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>
```

Embedded types?

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"

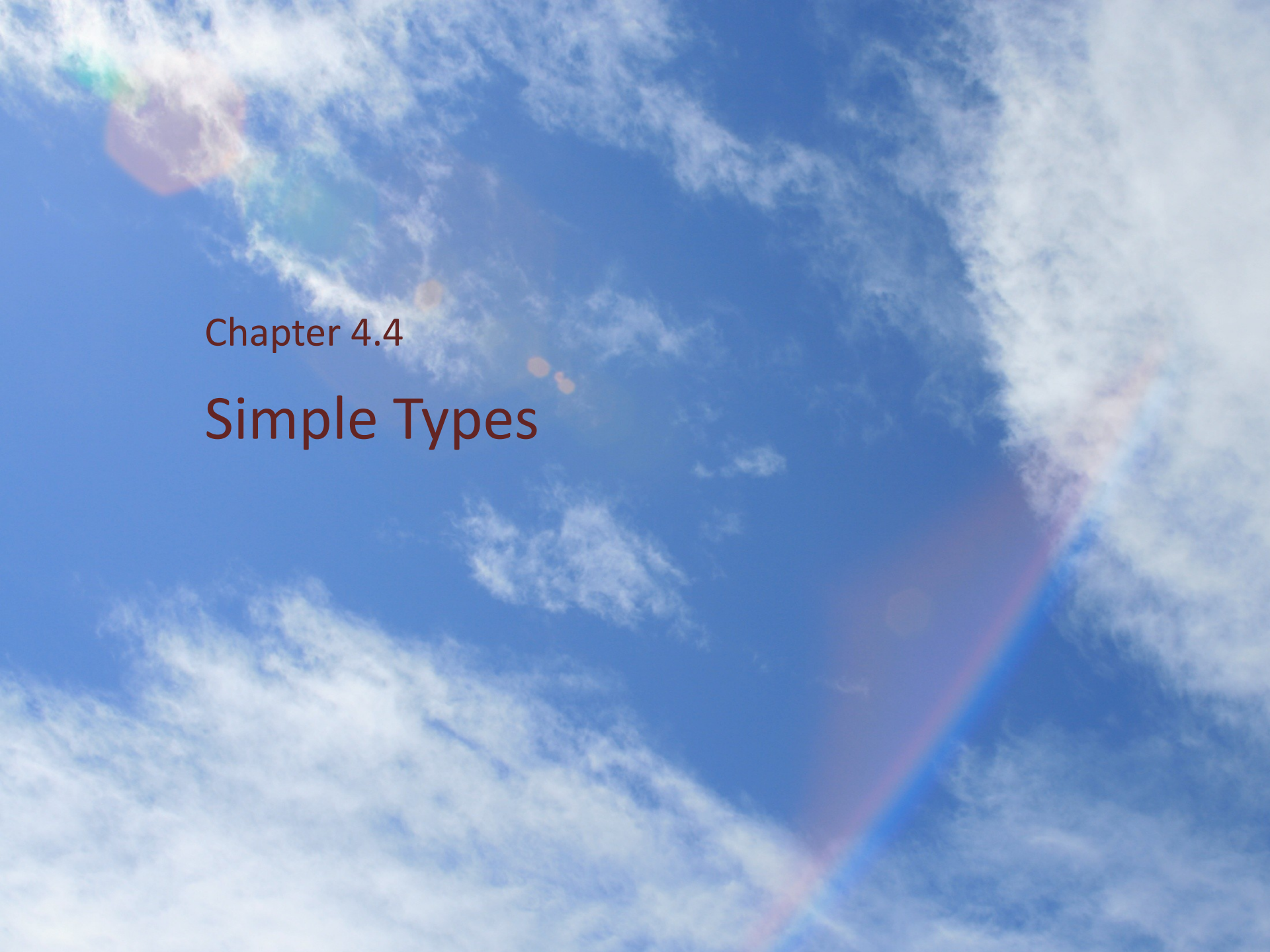
Note: *minOccurs* and *maxOccurs* can only be used in nested (local) element declarations.

2

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

3

flat catalogue



Chapter 4.4

Simple Types

Problem

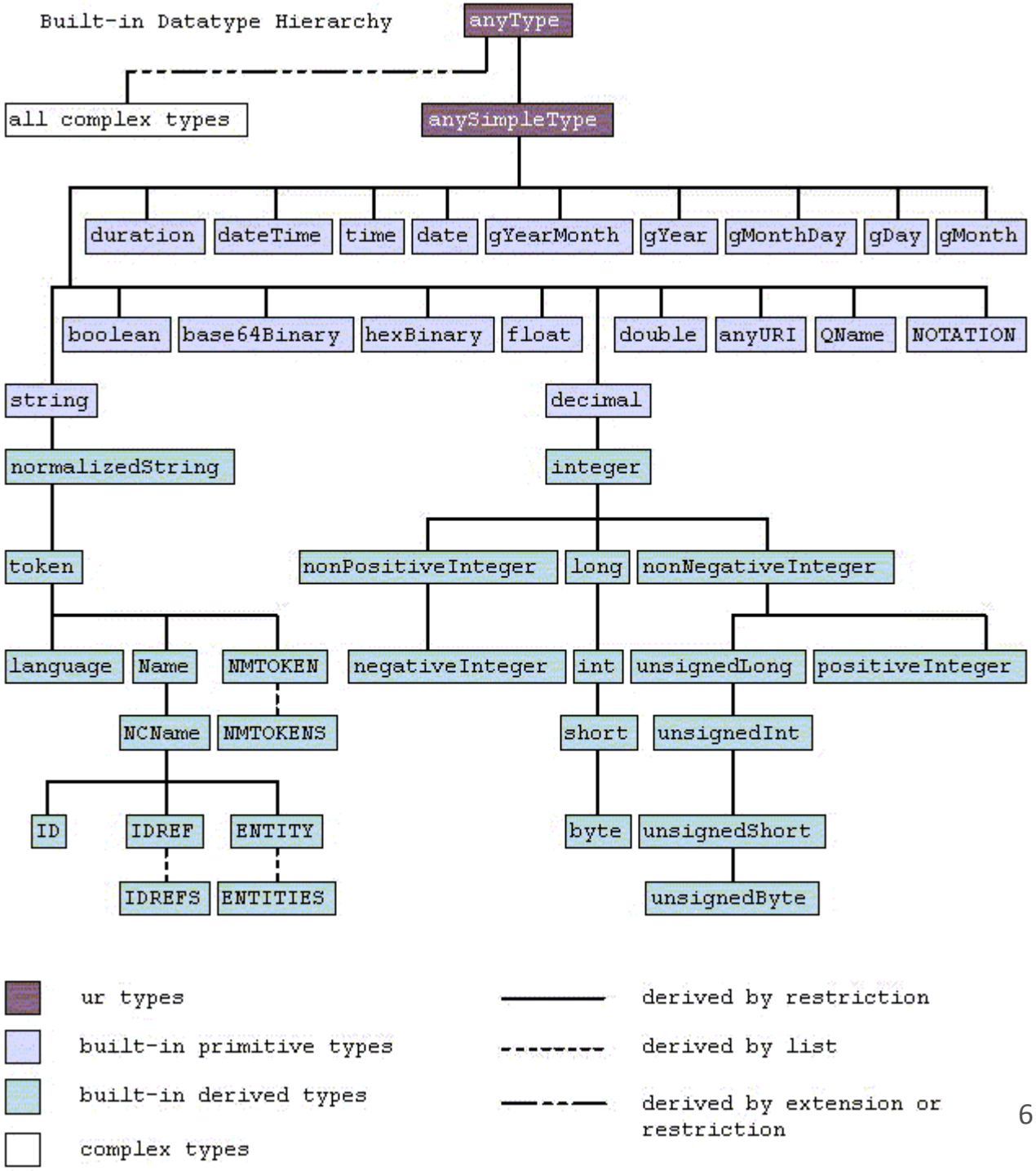
- Declaring the `Date` element of `BookStore.xsd`
  - type `string` is unsatisfactory
    - it allows any string value to be input as the content of the `Date` element, including non-date strings
  - constrain the allowable content that the `Date` element can have, e.g. restrict the content of `Date`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



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

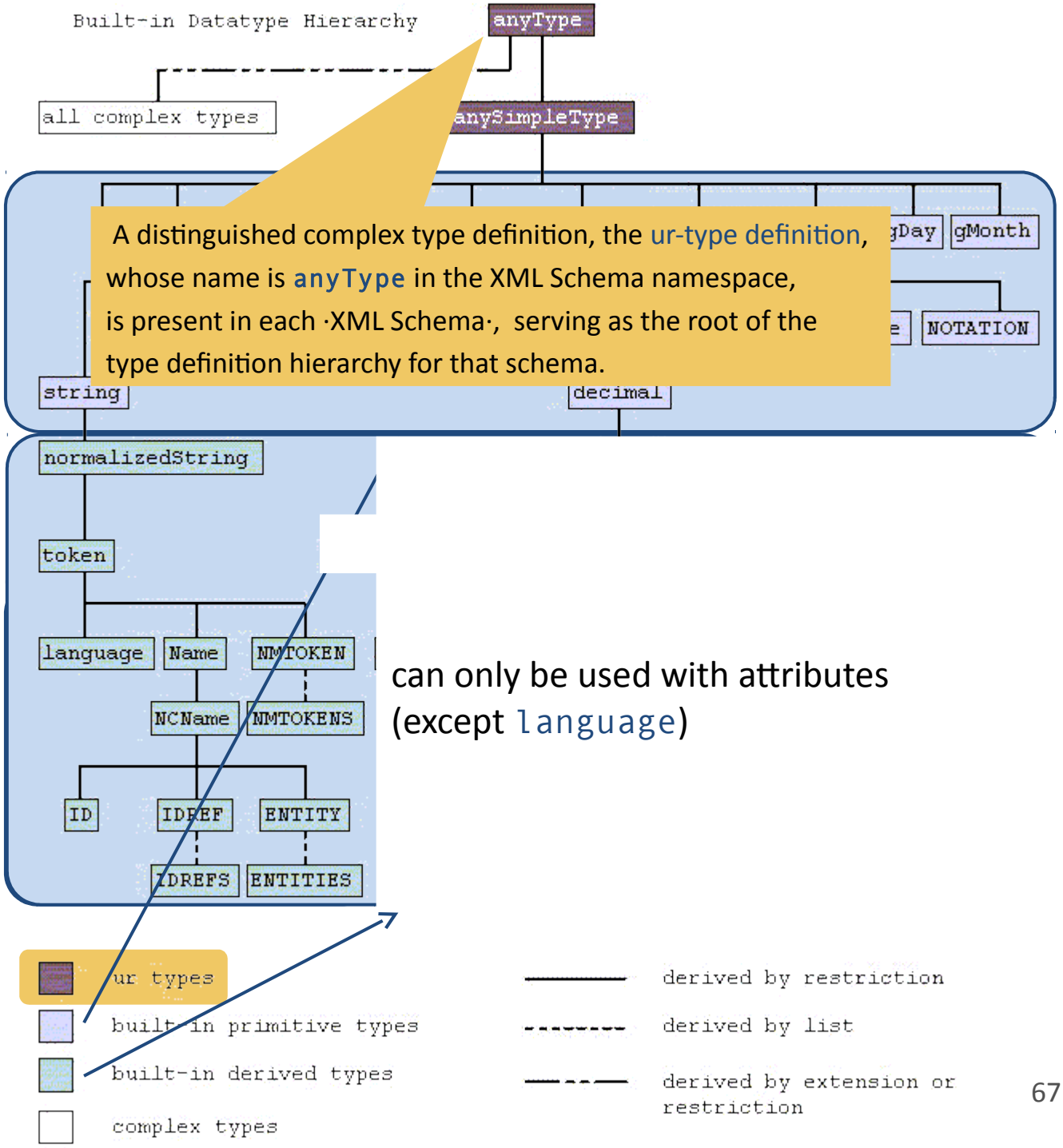
- Facets
  - A facet is a single defining aspect of a value space. Generally speaking, each facet characterizes a value space along independent axes or dimensions.
  - Facets are of two types:
    - fundamental facets
      that define the datatype and
    - non-fundamental or constraining facets
  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



can only be used with attributes (except `language`)

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-dddddd-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

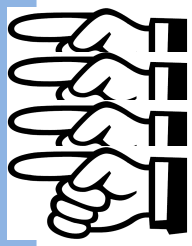
```
<xsd:simpleType name="ISBNType">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{1}-\d{5}-\d{3}-\d{1}"/>
      <xsd:pattern value="\d{1}-\d{3}-\d{5}-\d{1}"/>
      <xsd:pattern value="\d{1}-\d{2}-\d{6}-\d{1}"/>
   </xsd:restriction>
</xsd:simpleType>
```

```
<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	Facet instances
Chapter \d	Chapter 1
Chapter \d	Chapter 1
a*b	b, ab, aab, aaab, …
[xyz]b	xb, yb, zb
a?b	b, ab
a+b	ab, aab, aaab, …
[a-c]x	ax, bx, cx

Pattern Facet

- Regular Expressions (cont'd.)

Regular Expression	Explained
[a-c]x	x, bx, cx
[\-ac]x	-x, ax, cx (Backslash causes metacharacter "-" to be treated as literal char.)
[ac\-]x	ax, cx, -x
[^0-9]x	any non-digit char followed by x
\Dx	any non-digit char followed by x
Chapter\s\d	"Chapter" followed by a blank followed by a digit
(ho){2} there	hoho there
(ho\s){2} there	ho ho there
.abc	any (*one*) char followed by abc
(a|b)+x	ax, bx, aax, bbx, abx, bax,...

Pattern Facet

- Regular Expressions (cont'd.)

a{1,3}x	ax, aax, aaax
a{2,}x	aax, aaax, aaaax, …
\w\s\w	word **character** (alphanumeric plus dash) followed by a space followed by a word character
[a-zA-Z-[Ol]]*	A string comprised of any lower and upper case letters, except "O" and "l"
\.	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{Ll}	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

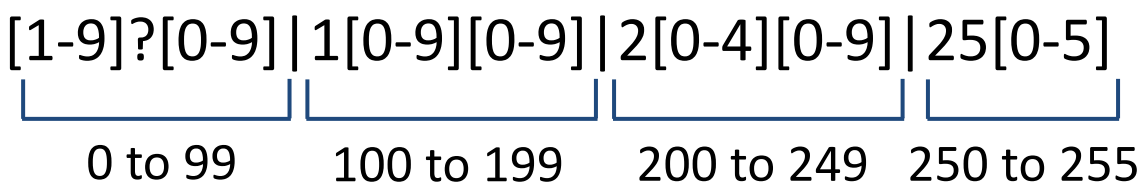
```
<xsd:simpleType name="money">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="\p{Sc}\p{Nd}+(\.\p{Nd}\p{Nd})?"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:element name="cost" type="money"/>
```

```
<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



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

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:

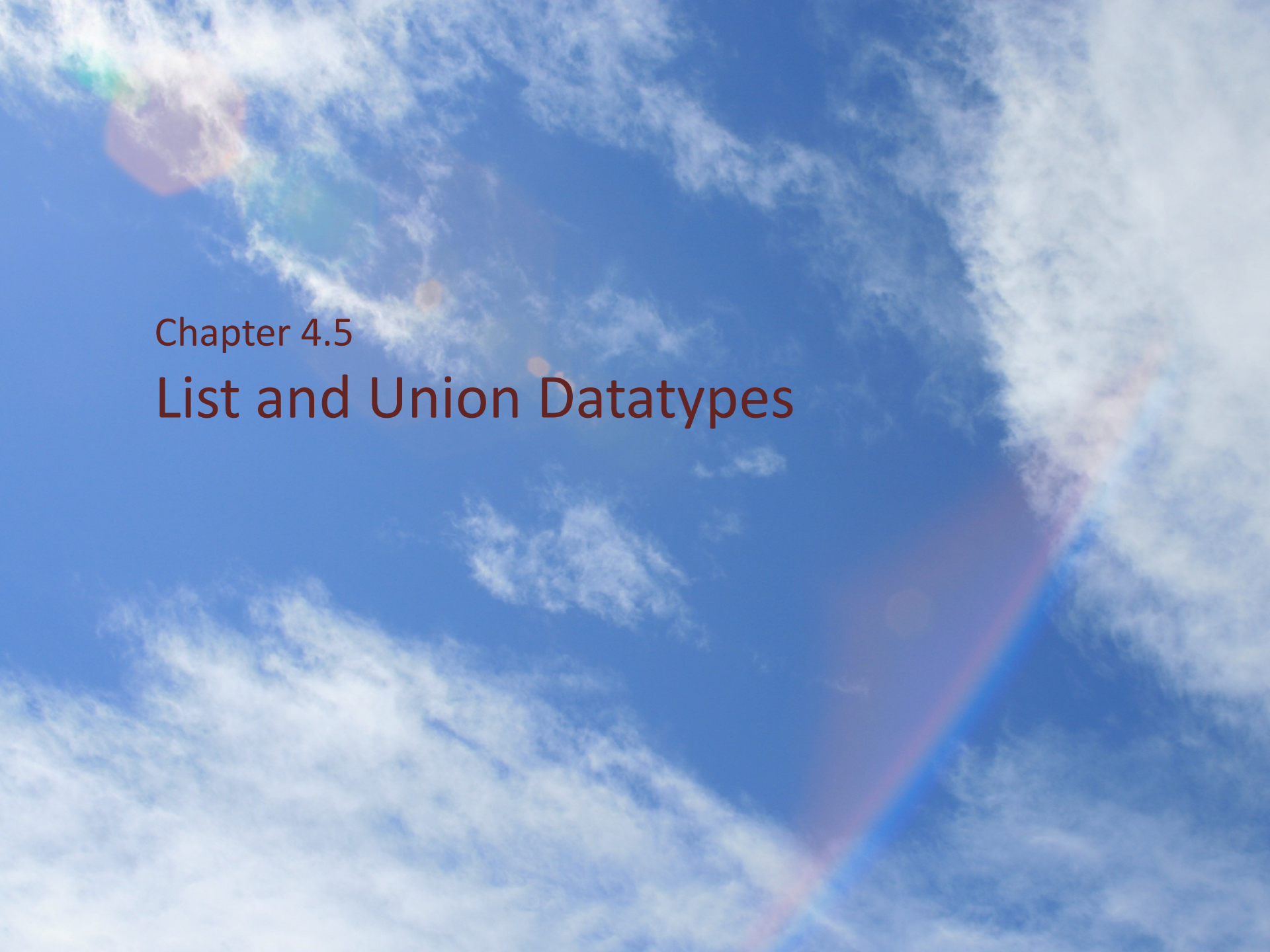
```
<xsd:simpleType name="EarthSurfaceElevation">
   <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="-1290"/>
      <xsd:maxInclusive value="29035"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:element name="elevation" type="EarthSurfaceElevation"/>
```

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"

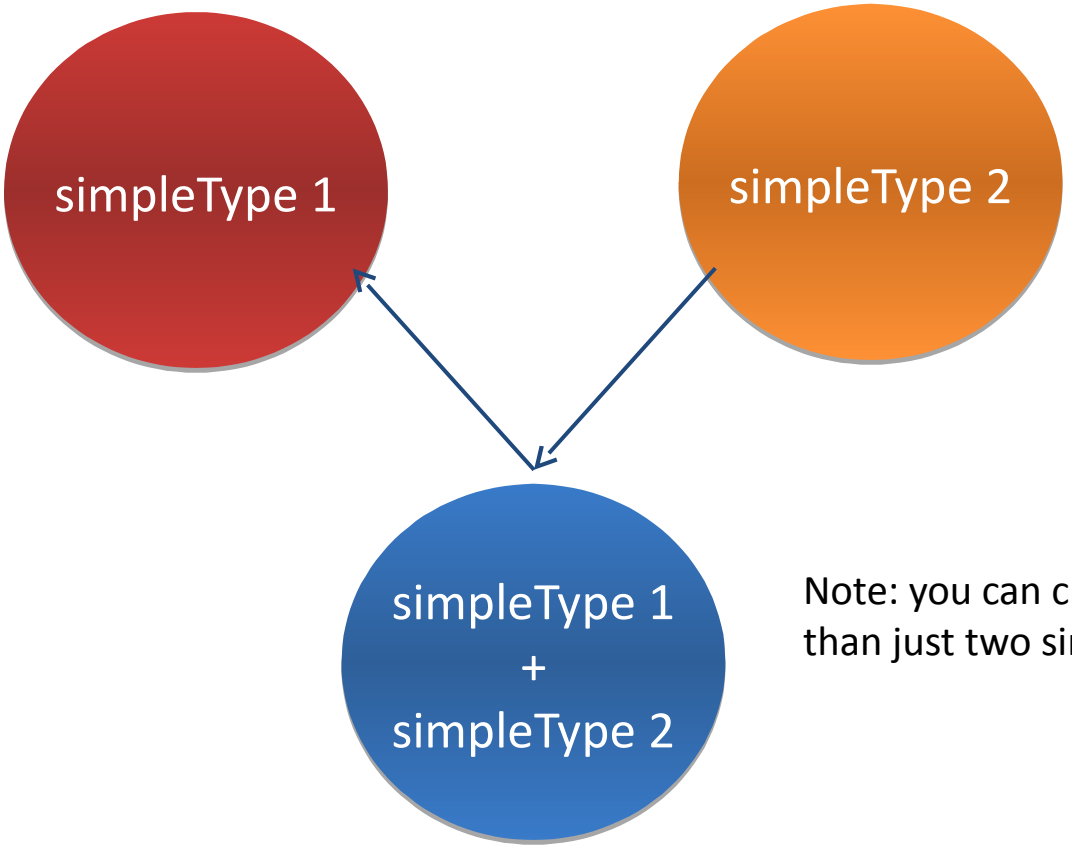
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     targetNamespace="http://www.lottery.org"
     xmlns="http://www.lottery.org" elementFormDefault="qualified">
   <xsd:simpleType name="OneToFortyNine">
     <xsd:restriction base="xsd:positiveInteger">
       <xsd:maxInclusive 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: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>
```

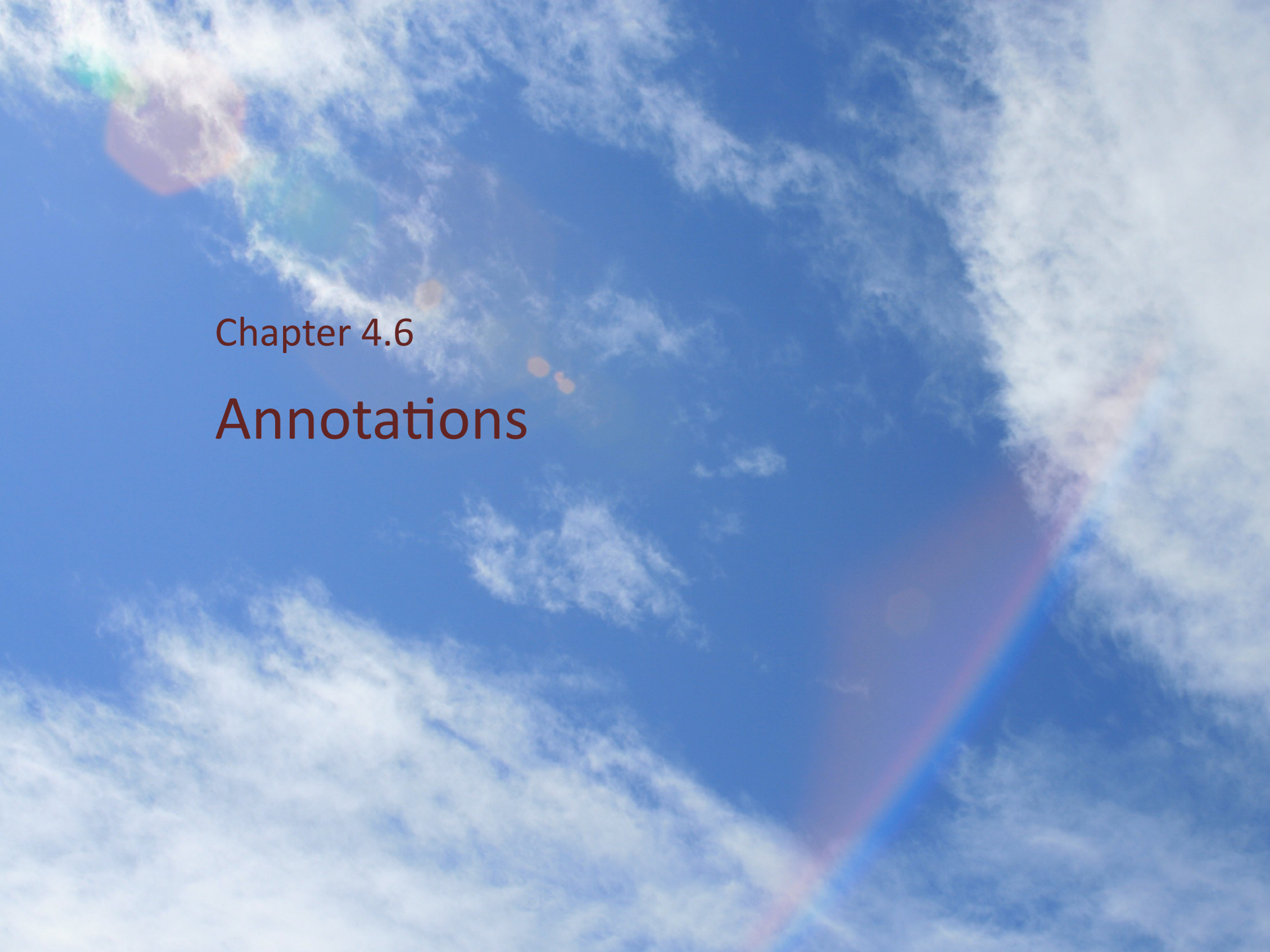
```
<xsd:simpleType name="JudysFamily">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Judy"/>
      <xsd:enumeration value="Peter"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="TomsFamily">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Tom"/>
      <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

```
<?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="Date" type="xsd:string">
                        <xsd:annotation>
                           <xsd:documentation>This is how to annotate the
Date                                          element!</xsd:documentation>
                        </xsd:annotation>
                     </xsd:element>
                     <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>
```

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