XML Document Navigation, Transformation

XPath—Navigating in XML Documents

Lecture "XML in Communication Systems"
Chapter 8

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

Acknowledgement

- Part of the material in this chapter is based on the

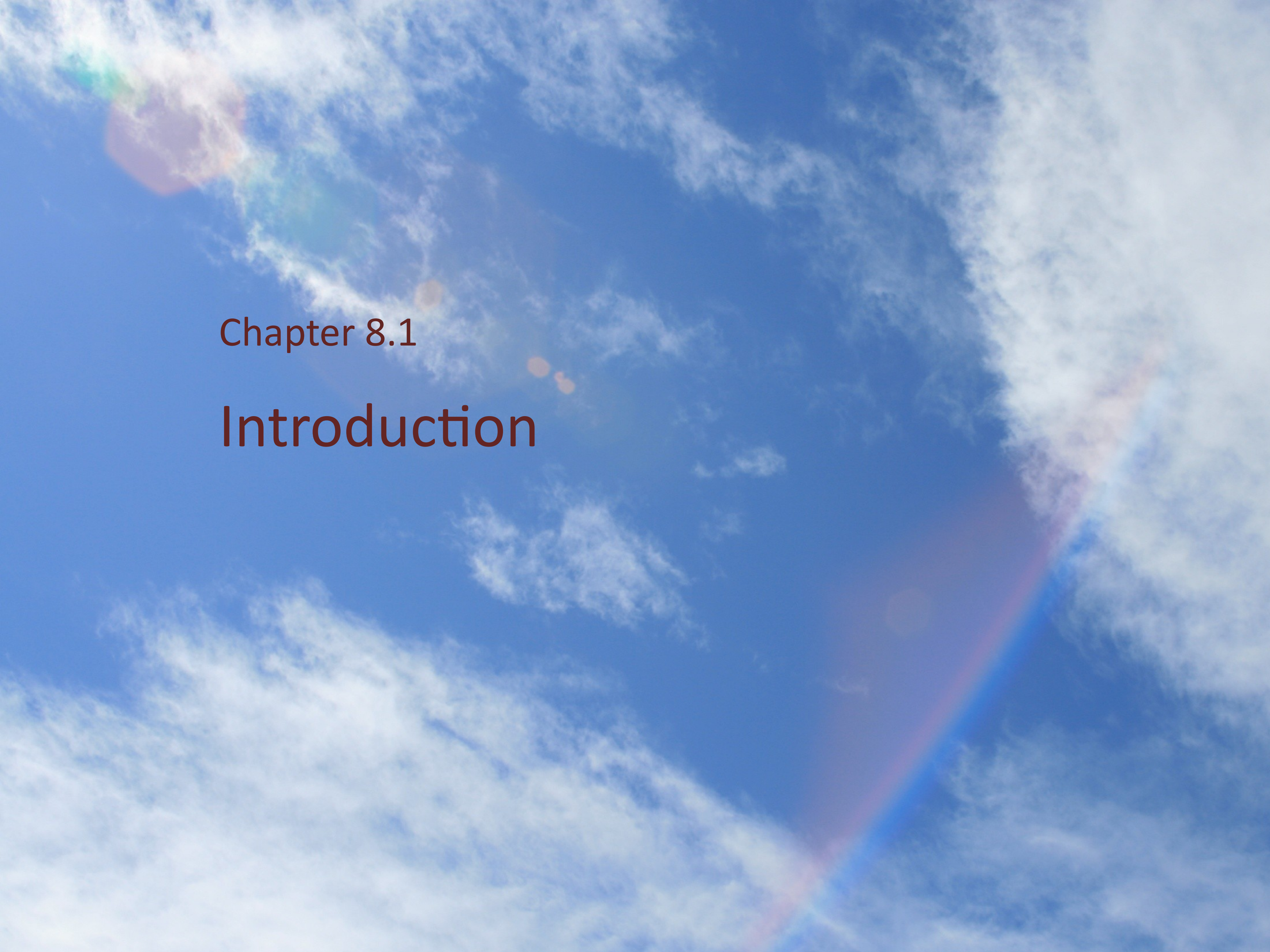
  XML Course of Dr. Torsten Grust,
  Dept. of Computer and Information Science,
  University of Konstanz

Recommended Reading

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

Overview

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



Chapter 8.1

Introduction

Introduction

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

Introduction

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

from the W3C TR on XPath 2.0

Chapter 8.2

XML Document Model

Information Items

- Node types

  r = root node

  C = set of comment nodes

  P = set of processing-instruction (PI) nodes

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

  A = set of attribute nodes

  N = set of namespace nodes

  T = set of text nodes

Information Items

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

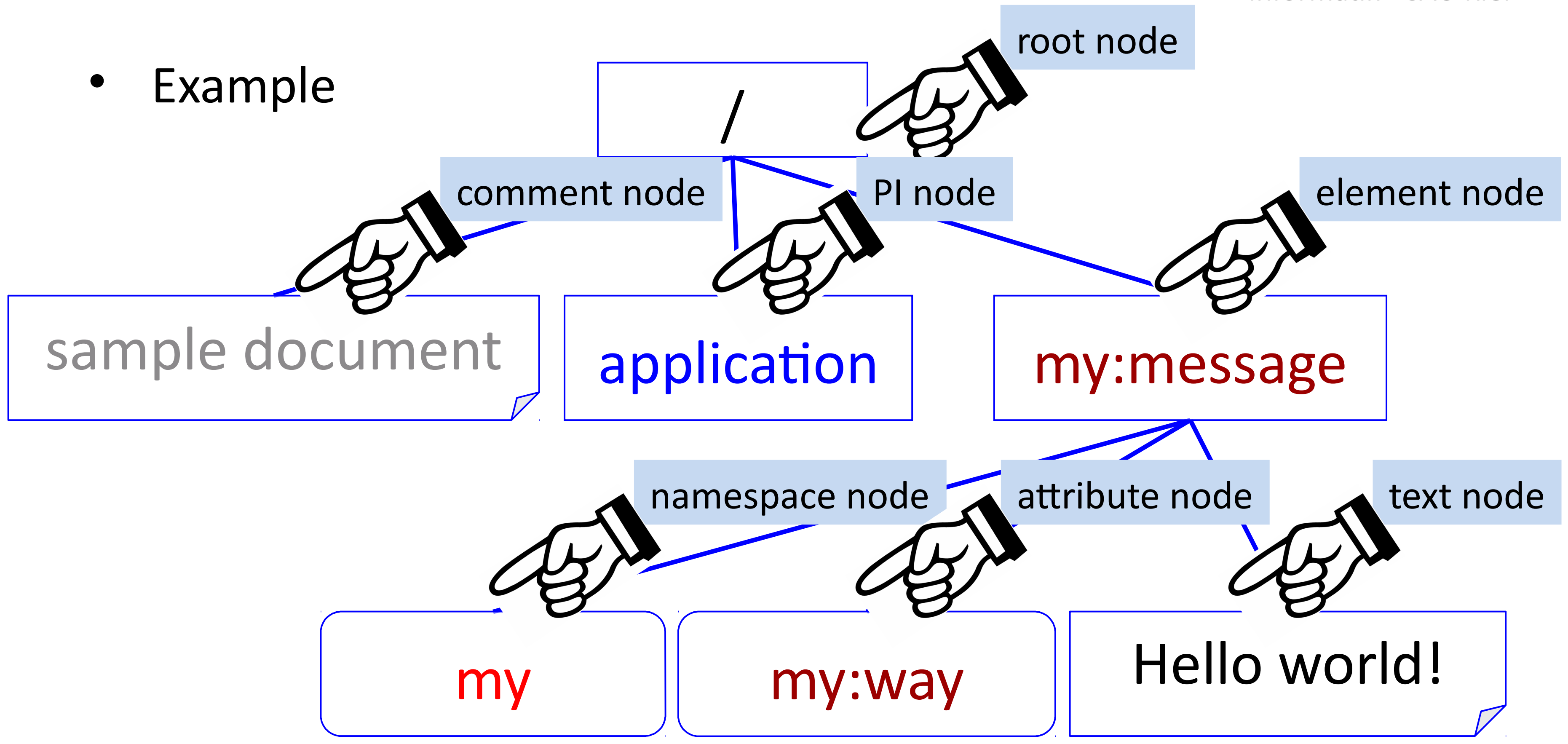
Information Items

- Example

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

Information Items

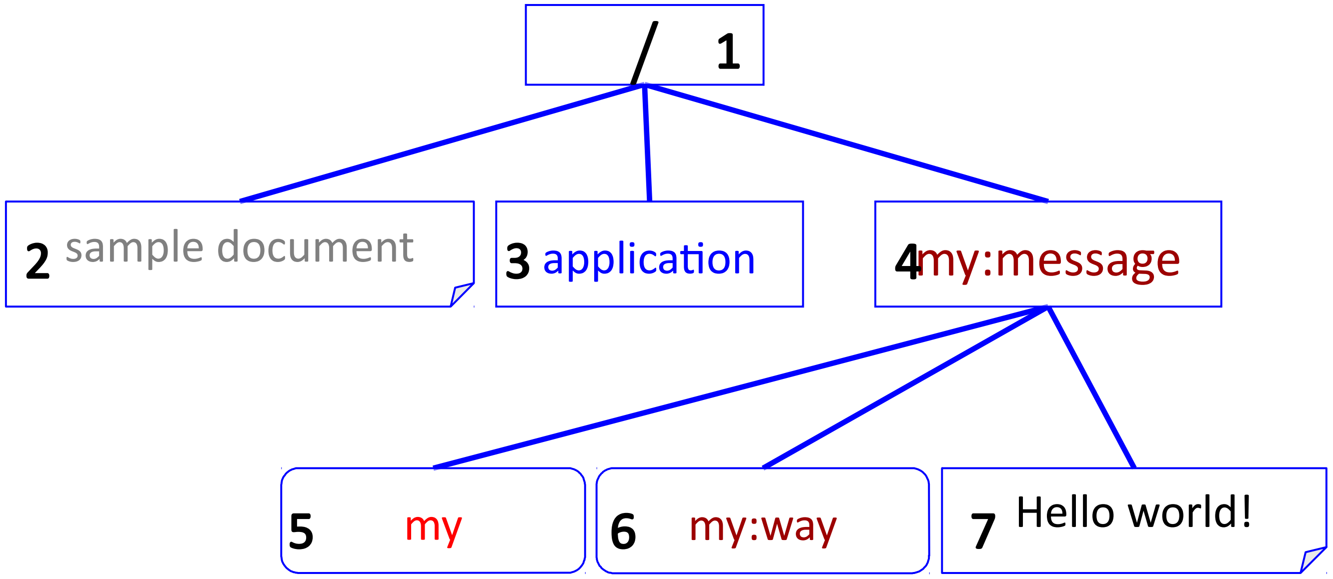
- Example



Information Items

- Element order

  o: {r}∪C∪P∪E∪A∪N∪T → { 1, 2, 3, 4, ... }



☞ Nodes are ordered according to *sequence of appearance* in XML document

Information Items

- Sample properties

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

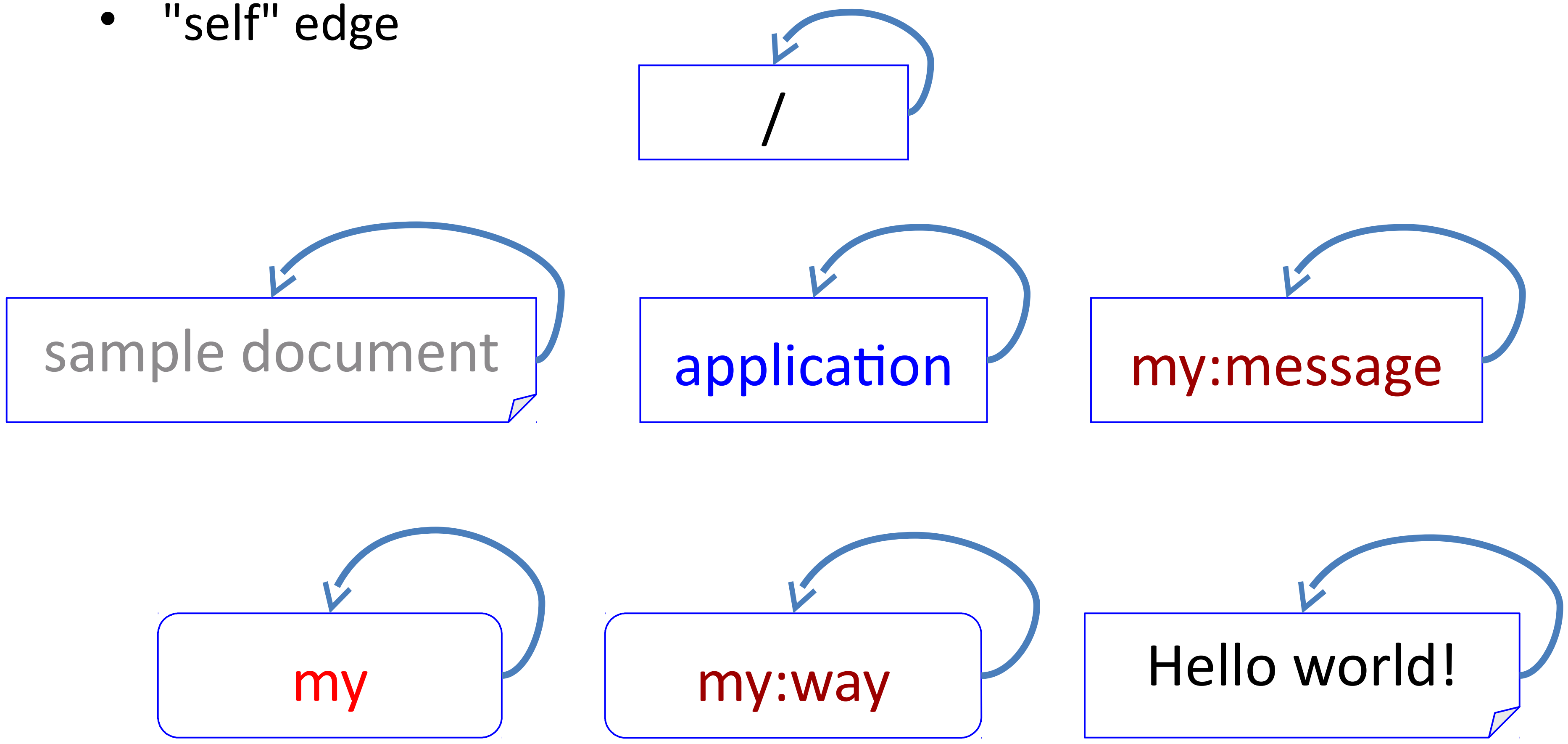
Information Items

- From example document

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

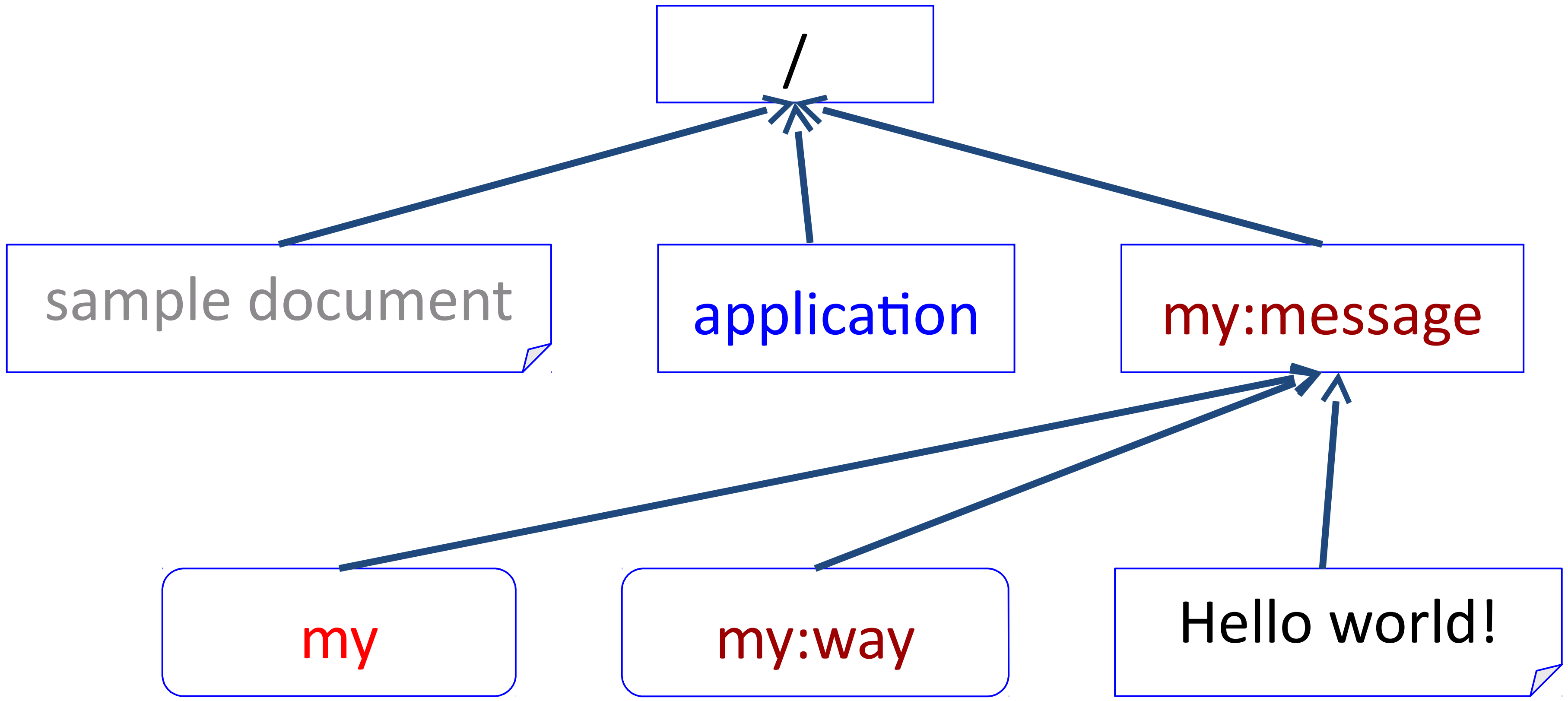
Edge Types

- "self" edge



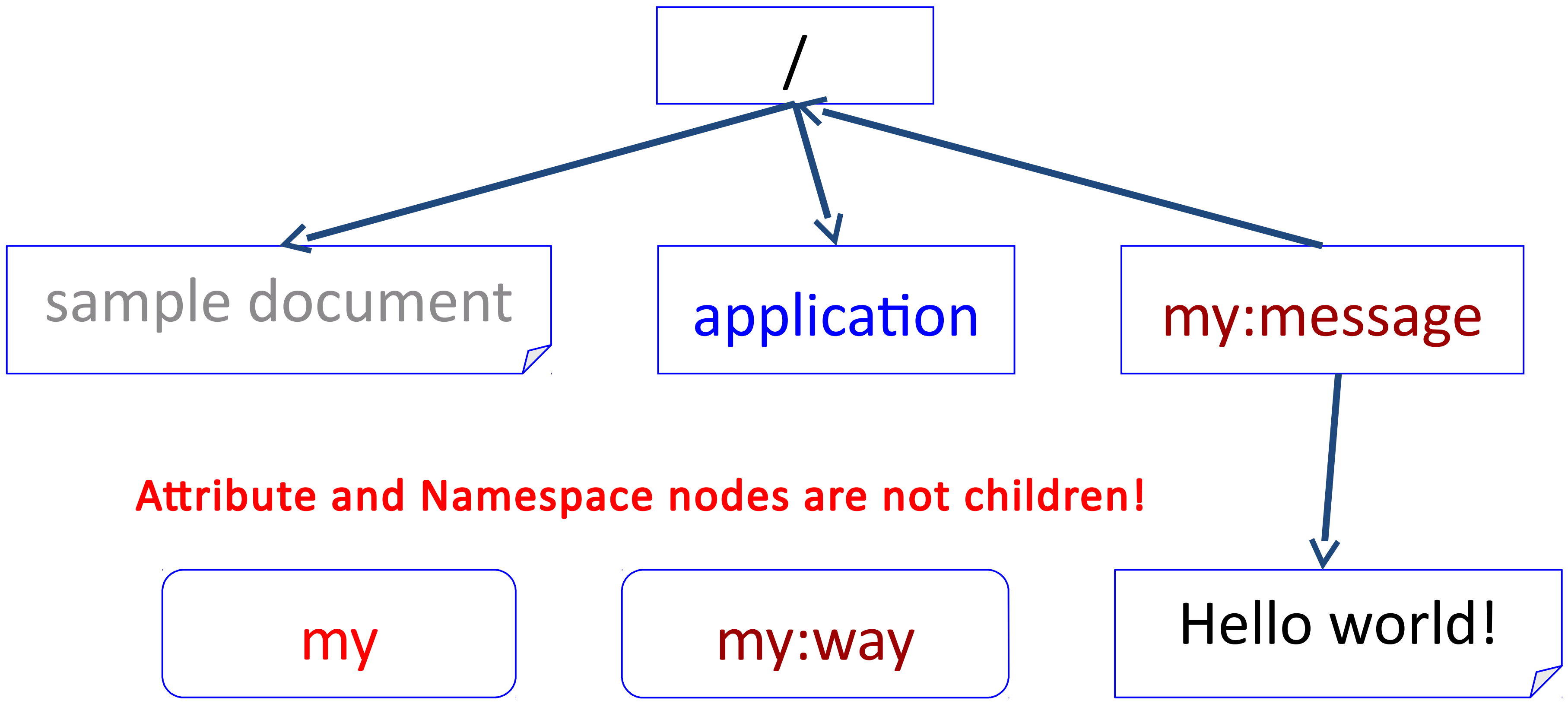
Edge Types

- "parent" edge



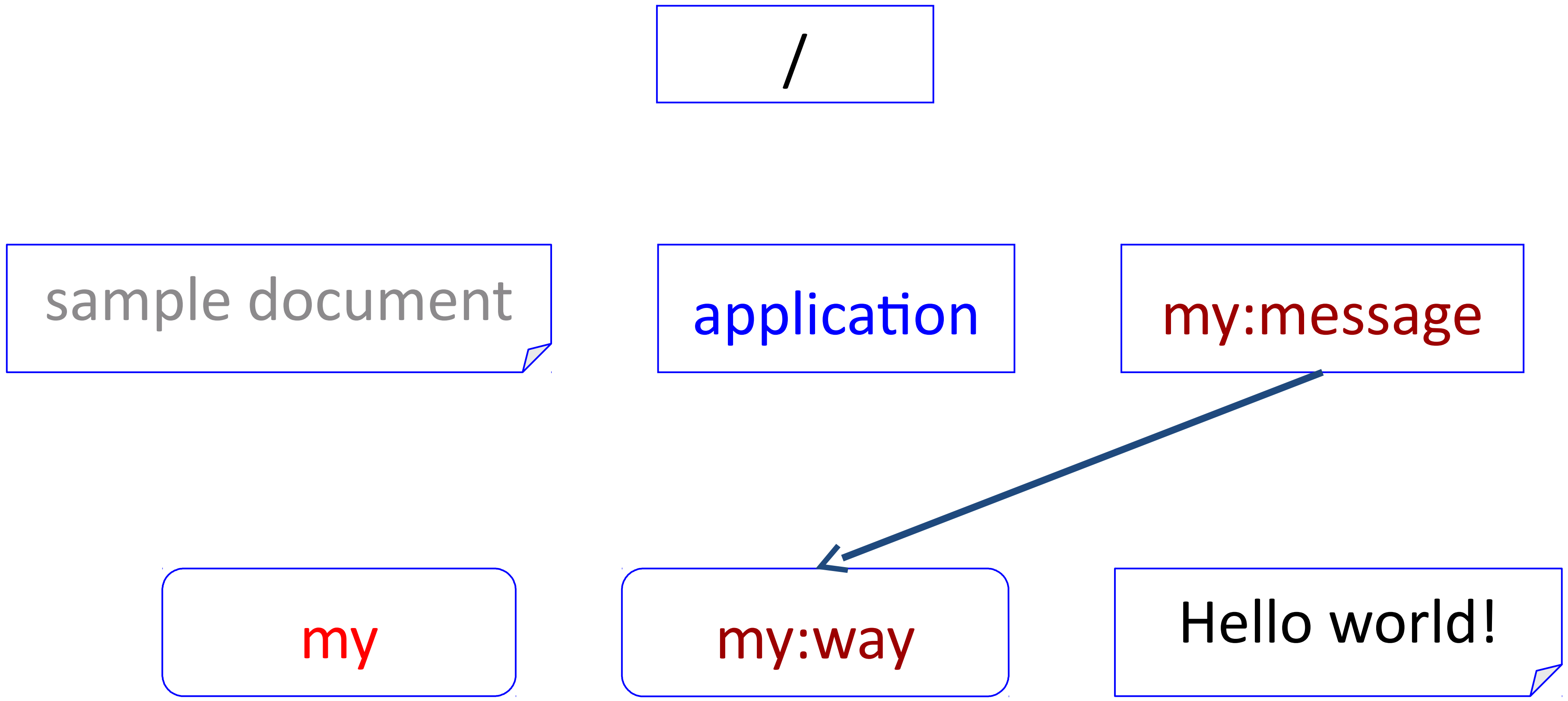
Edge Types

- "child" edge



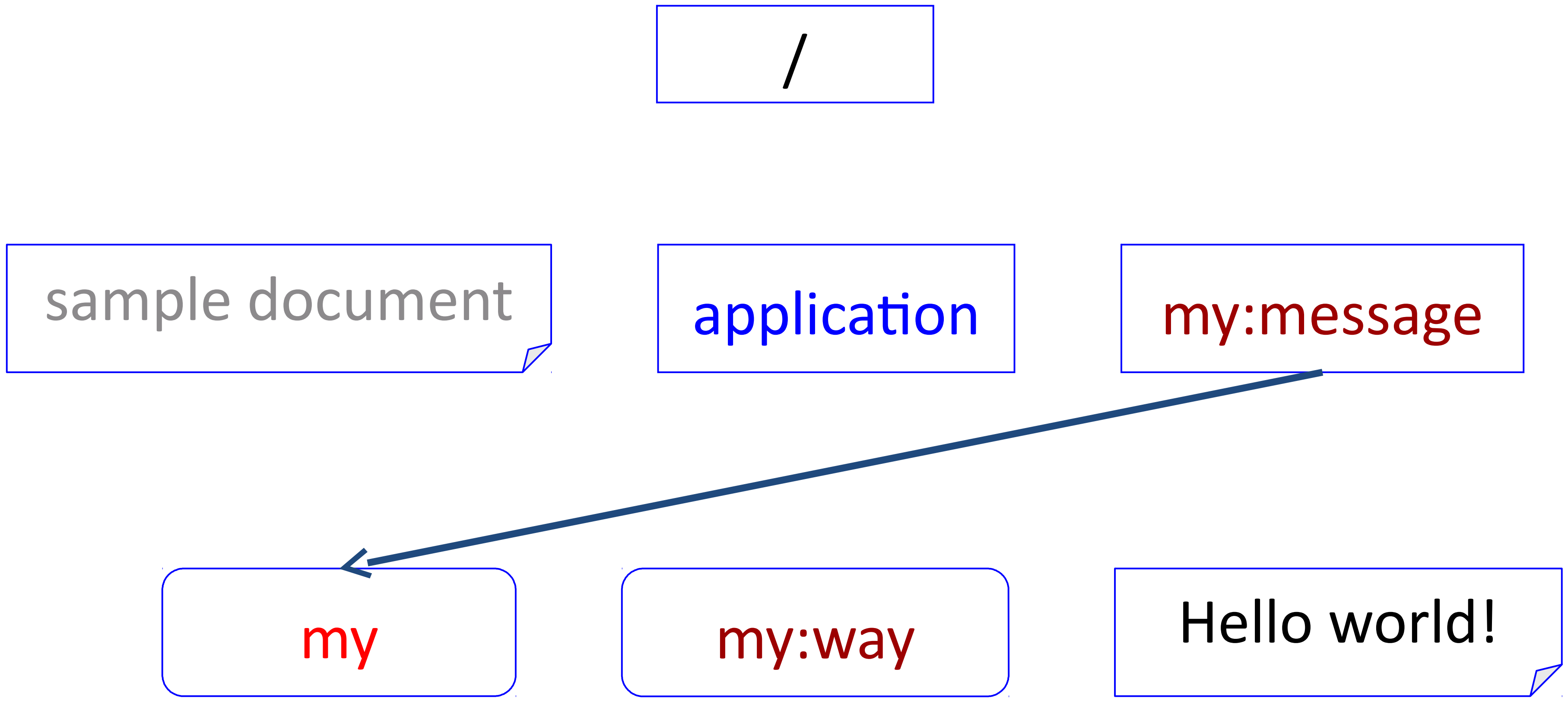
Edge Types

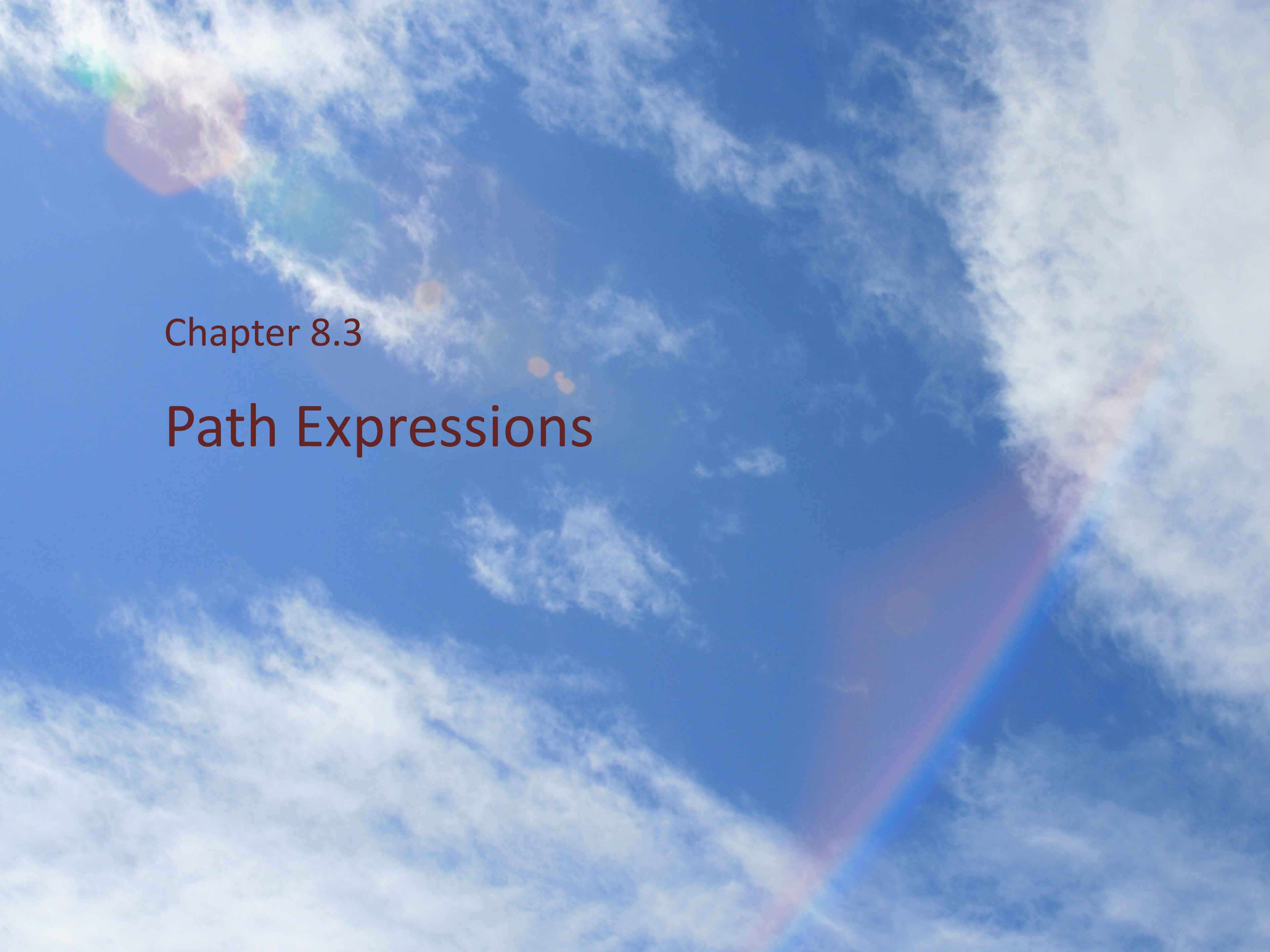
- "attribute" edge



Edge Types

- "namespace" edge





Chapter 8.3

Path Expressions

Path Expressions

- The path expression (or path) is XPath's core construct.
  - A path features one or more steps s_i (evaluated left to right), syntactically separated by /:

    $s_0/s_1/ \ldots /s_n$
  - Each step acts like an operator of type *(Node)* $\longrightarrow$ *(Node)*
  - Given the context node *c* and step *s*, the sequence of nodes reached by this step is computed.
  - Result is a duplicate-free node sequence in doc order.

Path Expressions

- The locstep (c; s) primitive function
  - Each step s is built from three components

    a::v [predicate]
  - The axis a determines, based on the location of context node c in the document tree, a sequence of reachable nodes:

    axis : context node $c \longrightarrow$ node set
  - XPath divides axes into two classes:
    - forward axes ($\longrightarrow$) and
    - reverse axes ($\longleftarrow$)
  - Reverse axes return their result in reverse document order.

Path Expressions

- Axes

self	$\rightarrow$	*c*
child	$\rightarrow$	child nodes of *c*
descendant	$\rightarrow$	closure of child
descendant-or-self	$\rightarrow$	like descendant, plus *c*
parent	$\leftarrow$	parent node of *c*
ancestor	$\leftarrow$	closure of parent
ancestor-or-self	$\leftarrow$	like ancestor, plus *c*
following	$\rightarrow$	nodes following *c* in doc order, but not descendants
preceding	$\leftarrow$	nodes preceding *c* in doc order, but not ancestors
following-sibling	$\rightarrow$	like following, if same parent as *c*
preceding-sibling	$\leftarrow$	like preceding, if same parent as *c*
attribute	$\rightarrow$	attributes of *c*
namespace	$\rightarrow$	namespace nodes of *c* (not discussed here)

Path Expressions

- Axes

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

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

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

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

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

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

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

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

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

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

following-sibling::(c) = sibling(c) $\cap$ following::(c)

preceding-sibling::(c) = sibling(c) $\cap$ preceding::(c)

with $\to^{+}$ denoting transitive closure, $\to^{*}$ reflexive-transitive closure

Path Expressions

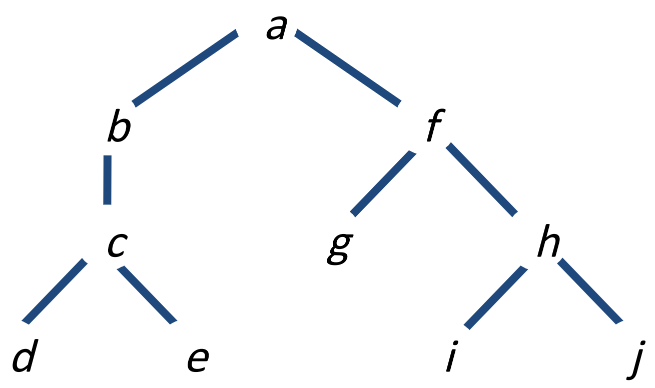
- The locstep (*c*; *s*) primitive function
  - The node test v filters this sequence to contain only specific types of nodes (e.g., only specifically named element nodes, only text/comment/PI nodes, etc.)
  - Predicates: (optional) expressions to further refine the set of nodes selected by the location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

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

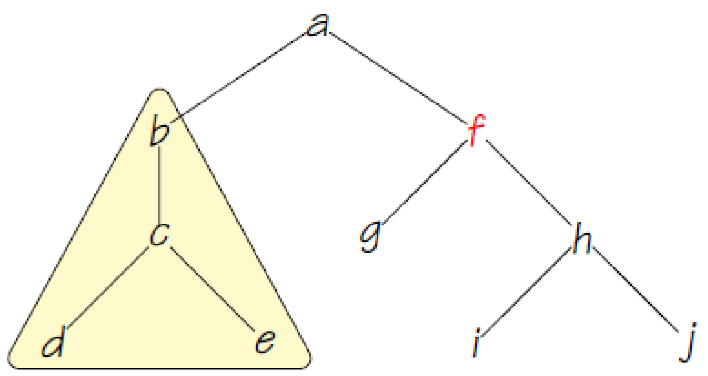
Path Expressions

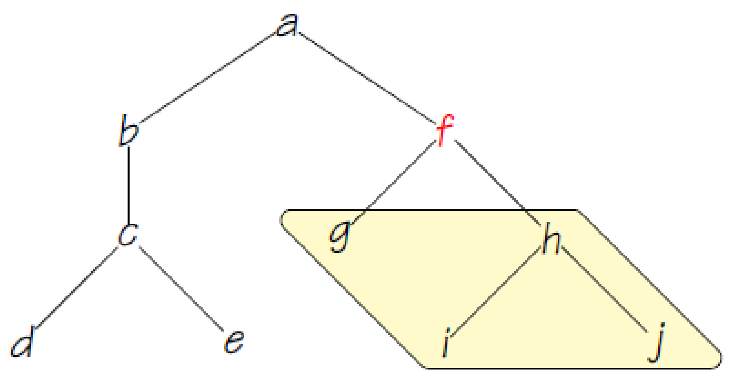
- Example

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



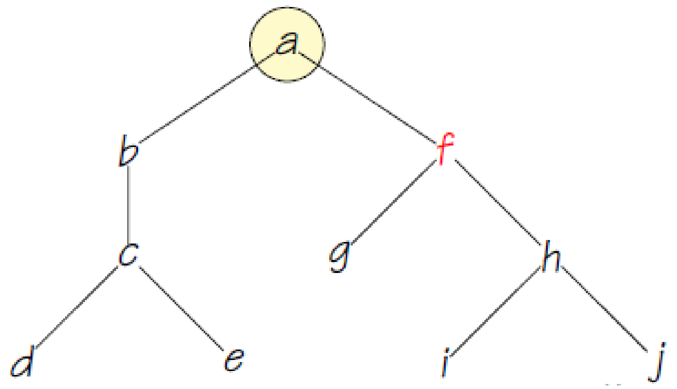
Path Expressions





①

,



ƒ

Examples

context node *f* and

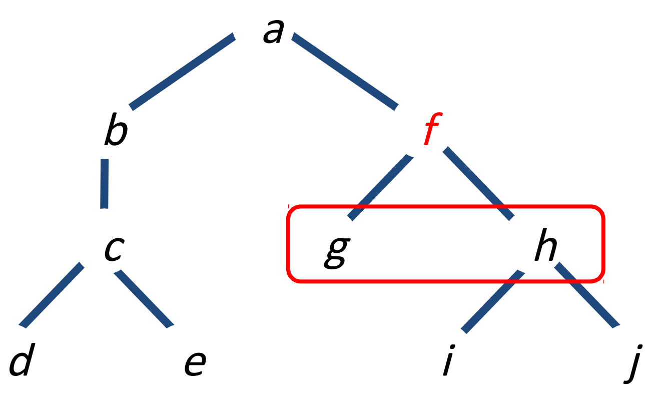
v =node(): don't care node test

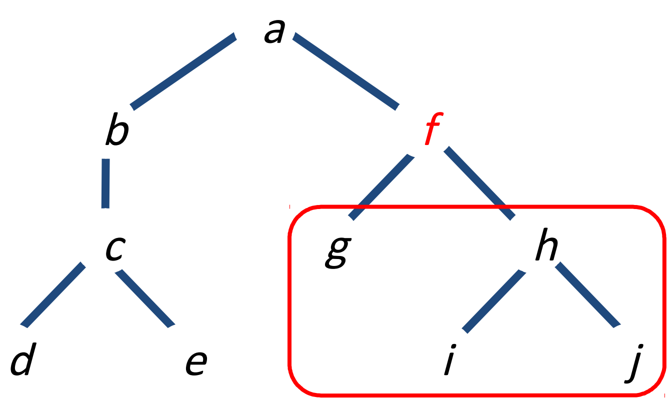
Path Expressions

- Just do it!

locstep (*f*; child::node())

locstep (*f*; descendant::node())

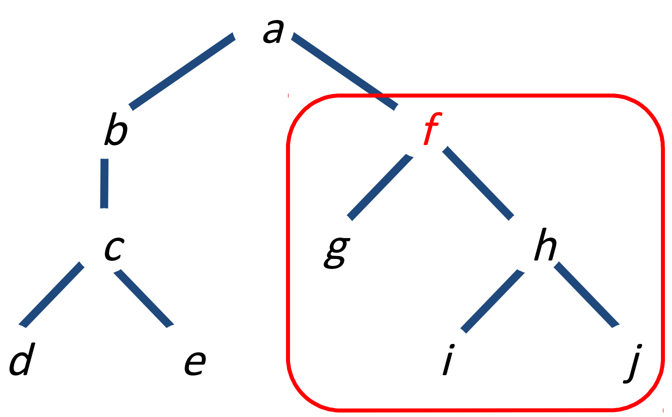




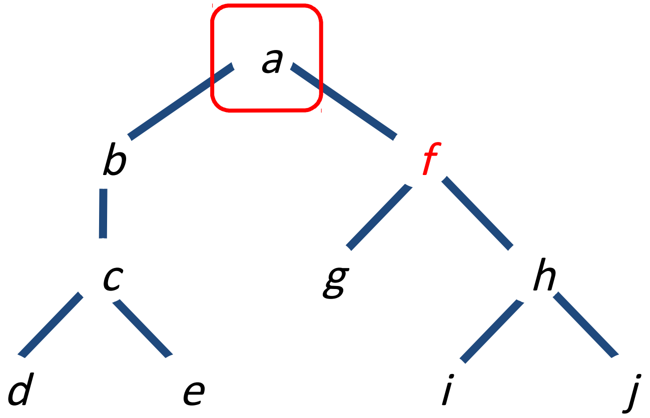
Path Expressions

- Just do it!

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



locstep (*f*, parent::node())

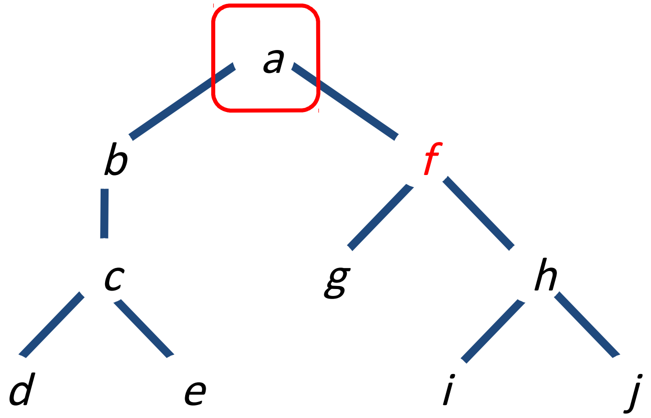


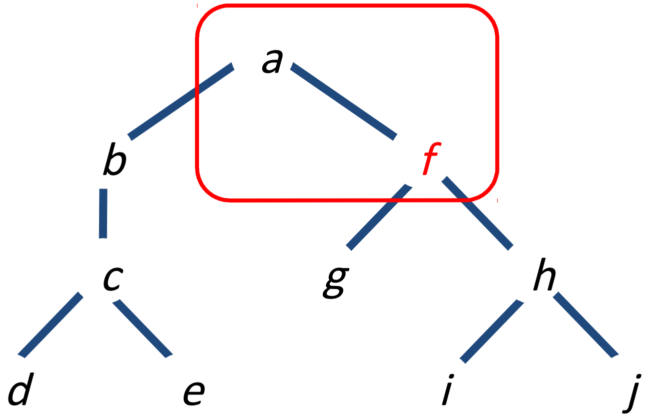
Path Expressions

- Just do it!

locstep (*f*; ancestor::node())

locstep (*f*; ancestor-or-self::node())

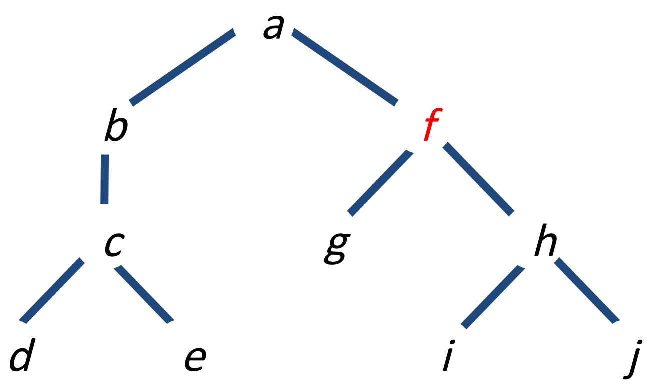




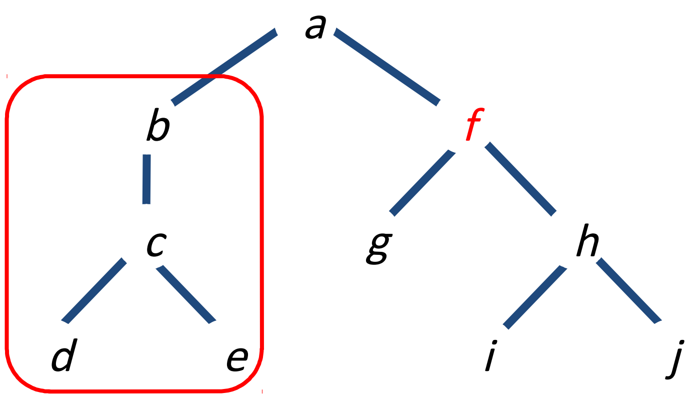
Path Expressions

- Just do it!

locstep (*f*; following::node())



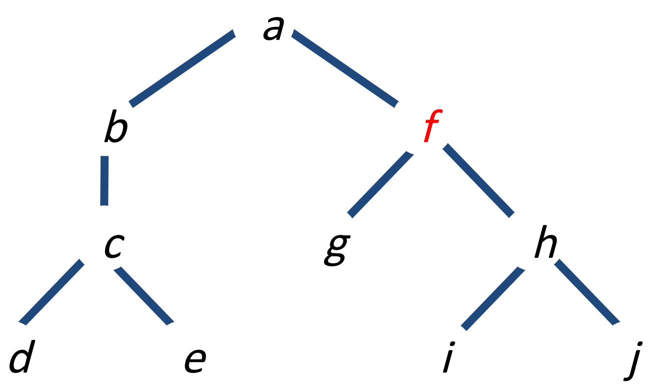
locstep (*f*; preceding::node())



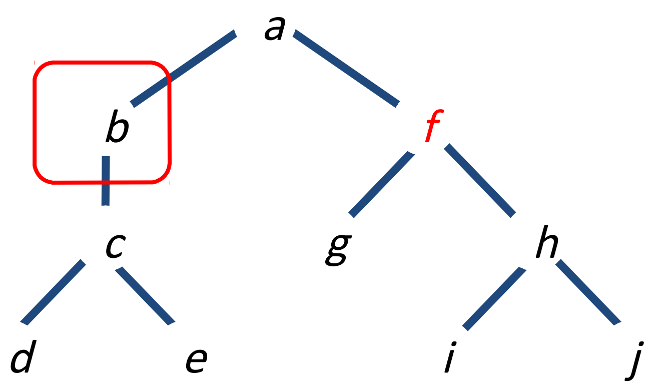
Path Expressions

- Just do it!

locstep (*f*; following-sibling::node())



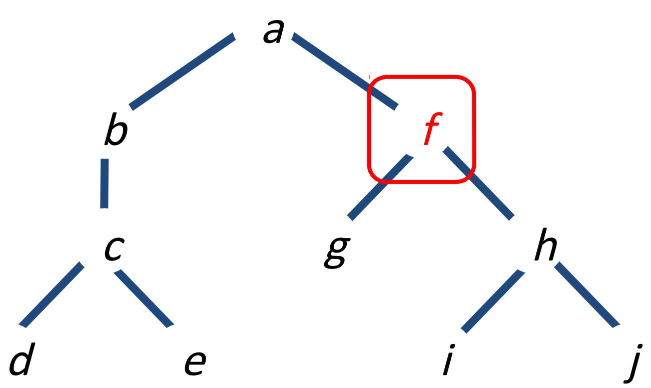
locstep (*f*; preceding-sibling::node())



Path Expressions

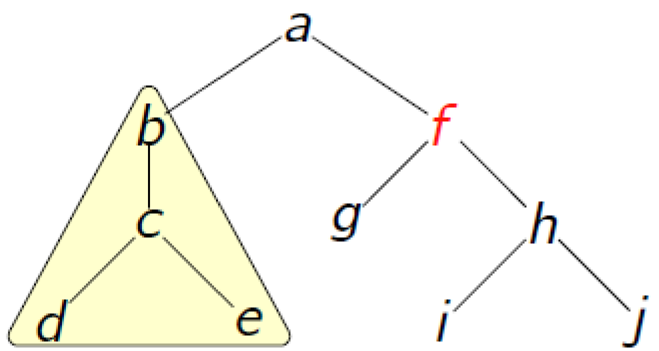
- Just do it!

locstep (*f*; self::node())

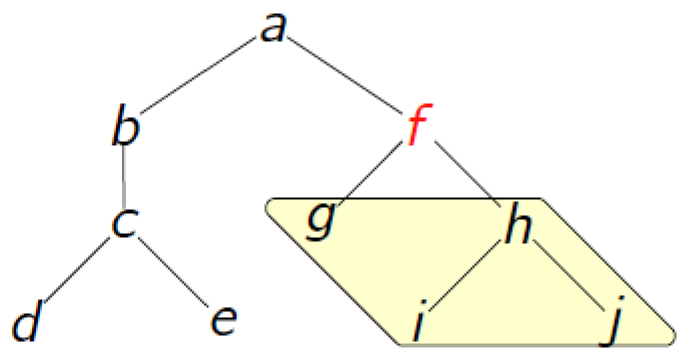


Path Expression

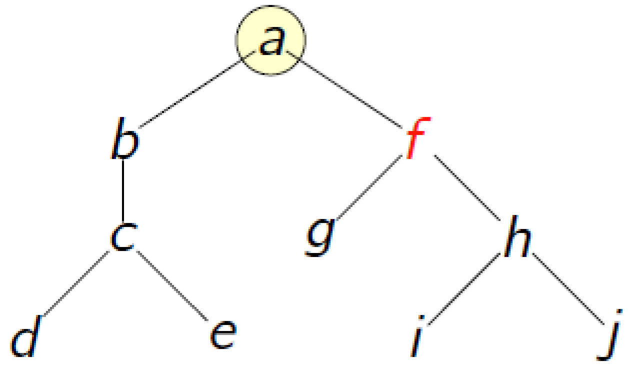
- Axes
  - The `ancestor`, `descendant`, `following`, `preceding` and `self` axes partition a document (ignoring attribute and namespace nodes): they do not overlap and together they contain all the nodes in the document.



f/`preceding::*`



f/`descendant::*`



f/`ancestor::*`

f/`descendant::*` $\cup$ f/`ancestor::*` $\cup$

f/`preceding::*` $\cup$ f/`following::*` $\cup$ $\{f\}$

$=$

$\{a \ldots j\}$

Path Expressions

- Absolute/relative location paths

LocationPath ::= RelativeLocationPath | AbsoluteLocationPath

AbsoluteLocationPath ::= '/' RelativeLocationPath? |
AbbreviatedAbsoluteLocationPath

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

Step ::= AxisSpecifier NodeTest Predicate*

Path Expressions

- Node tests
  - principal node type (PNT)

for attribute axes:	PNT = A
for namespace axes:	PNT = N
otherwise:	PNT = E

Path Expressions

- Node tests

  Node_test: node set M $\rightarrow$ node set

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

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

  * true for any node of the principal node type

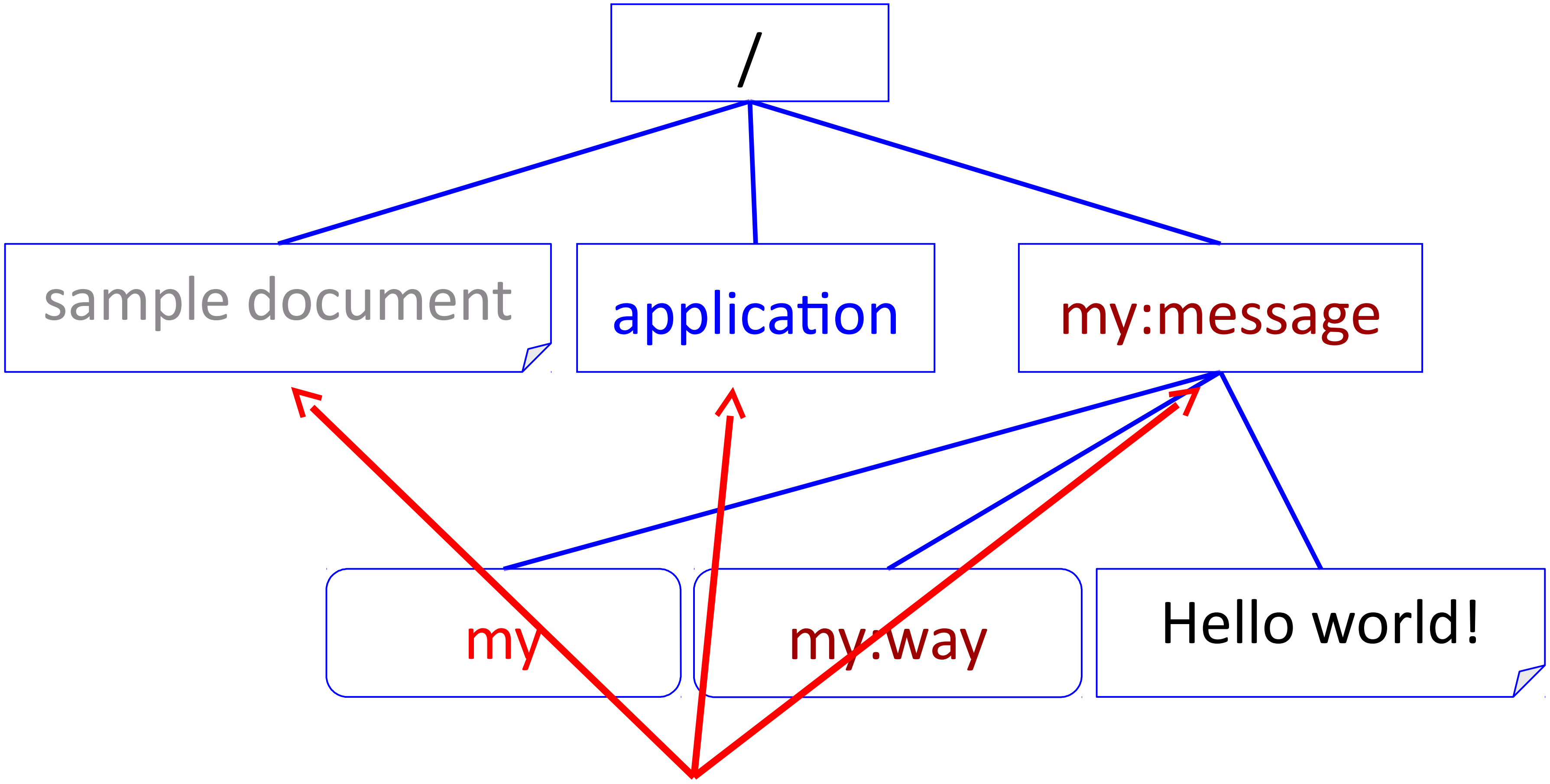
  text() true for any Text node

  comment() true for any Comment node

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

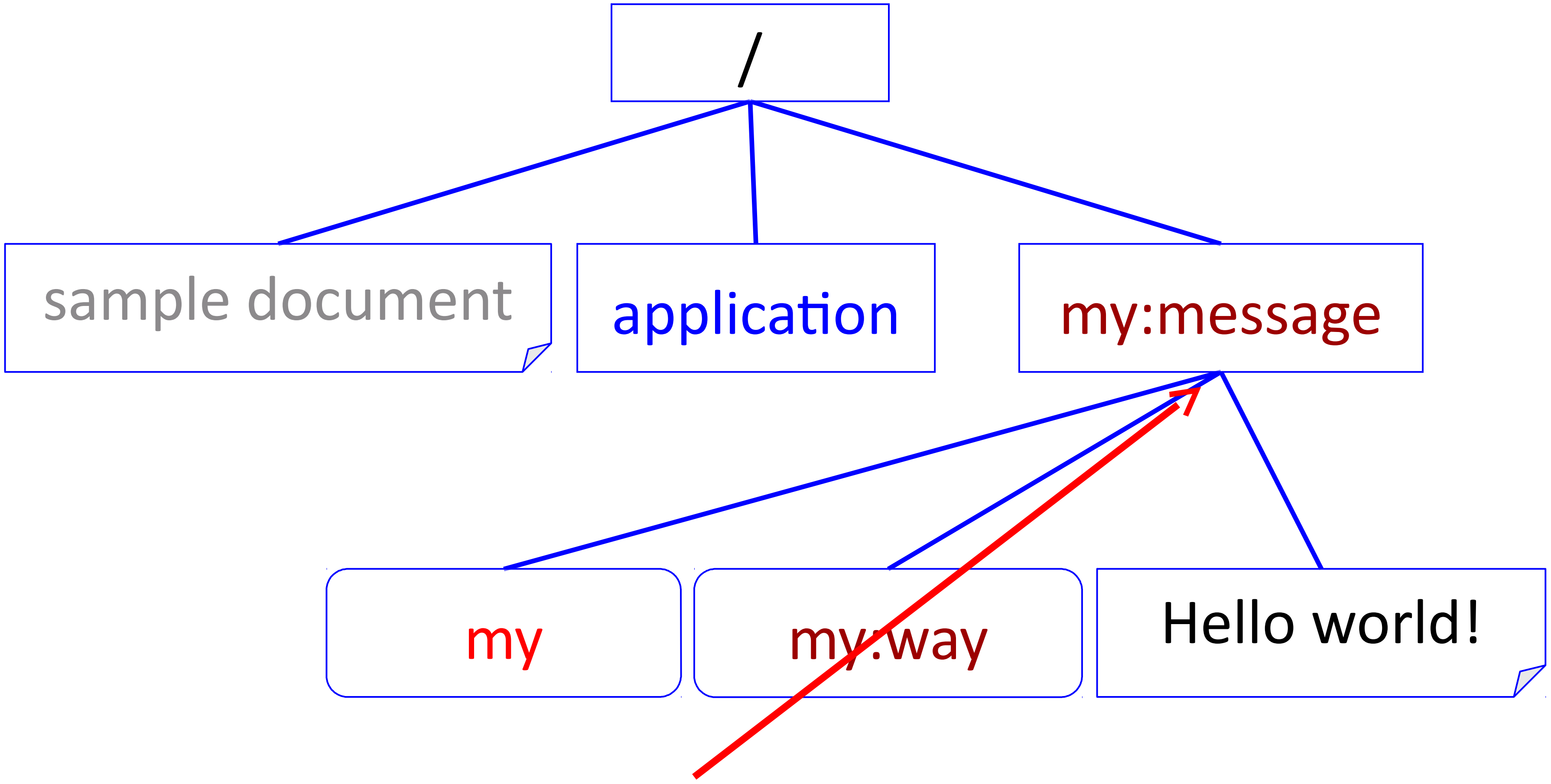
  "A node test that is a QName is true if and only if the type of the node is the principal node type and has an expanded-name equal to the expanded-name specified by the QName. For example, child::para selects the para element children of the context node; if the context node has no para children, it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

Path Expressions



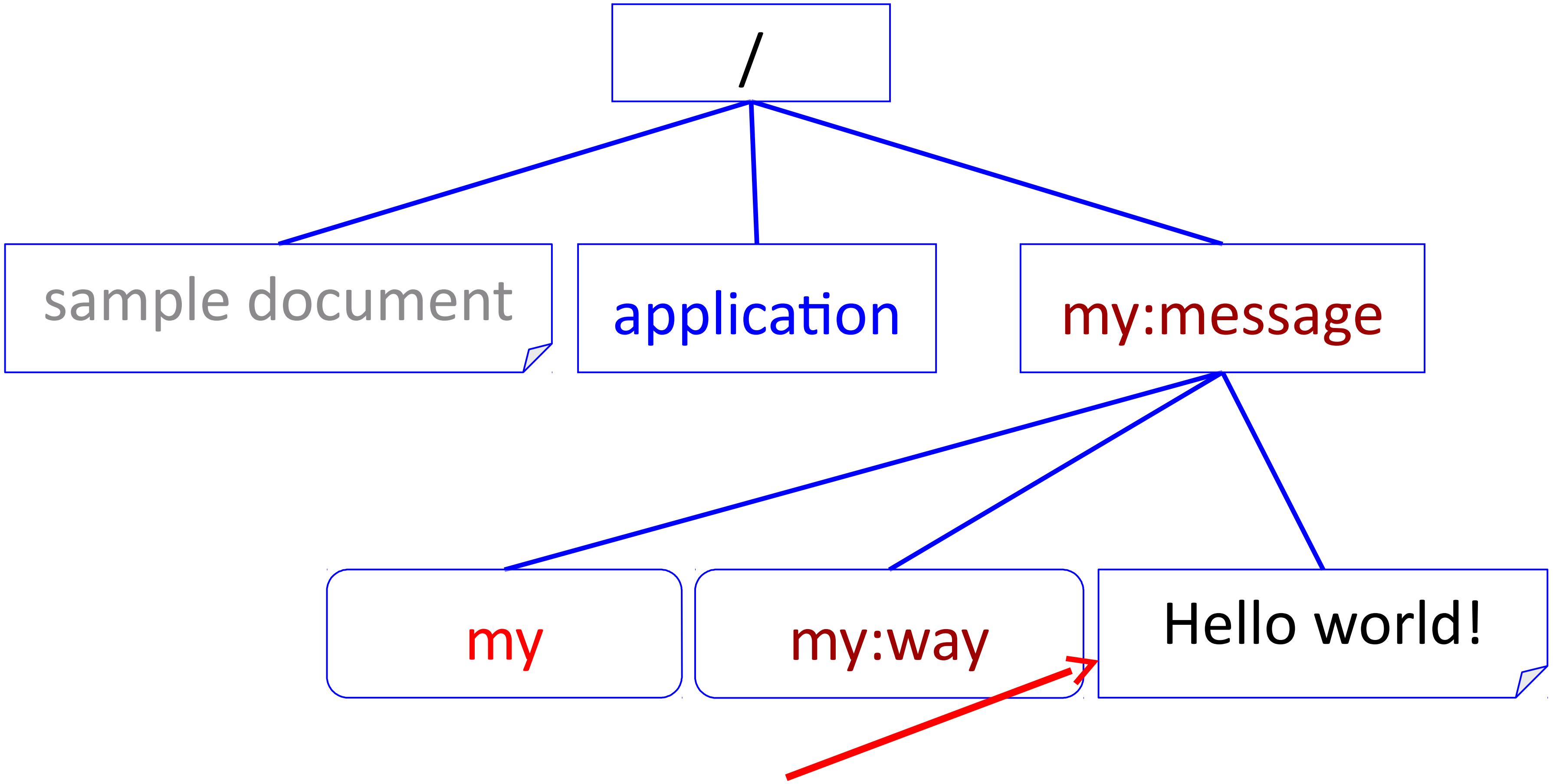
/child::node()

Path Expressions



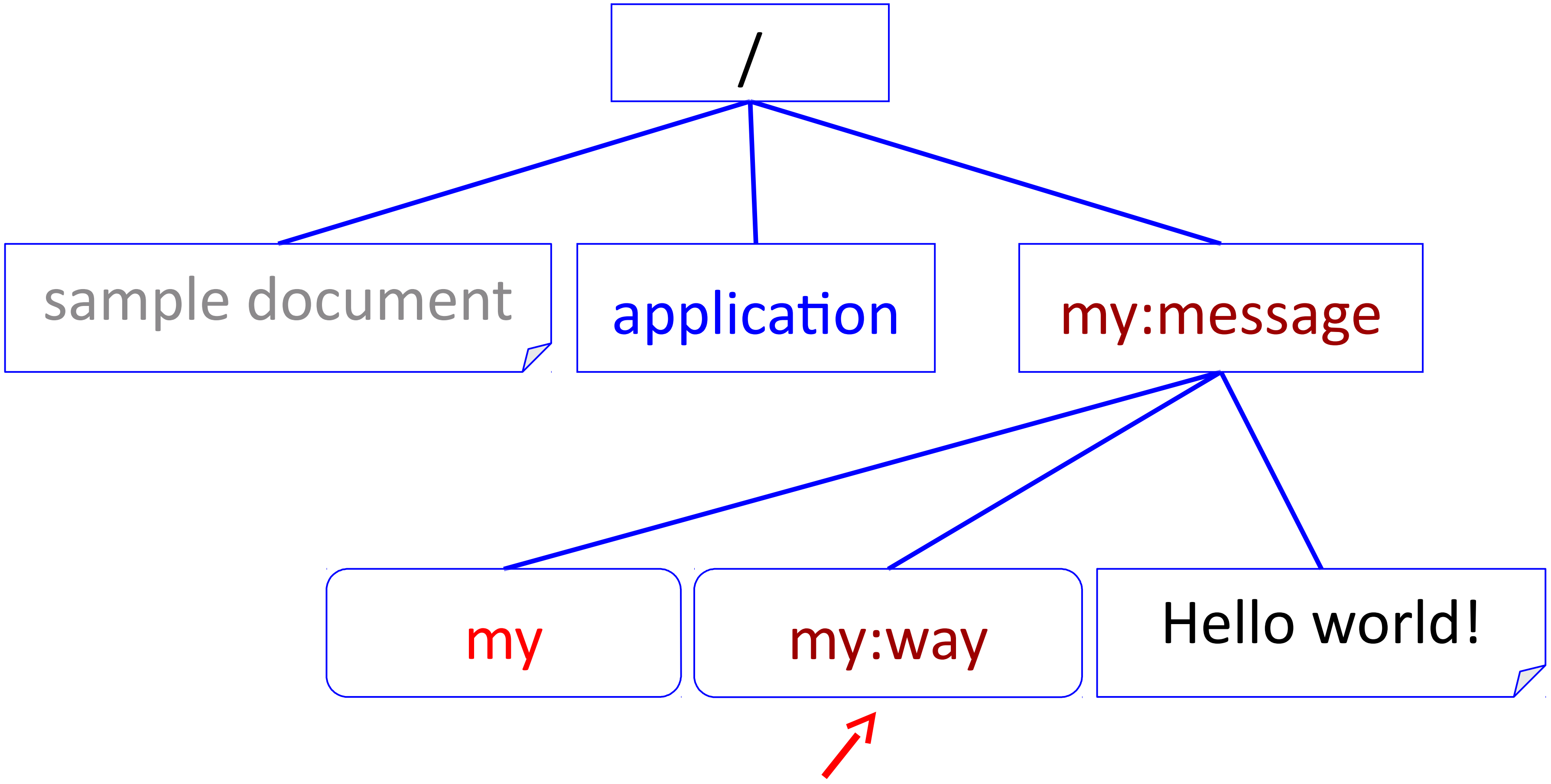
/child::*

Path Expressions



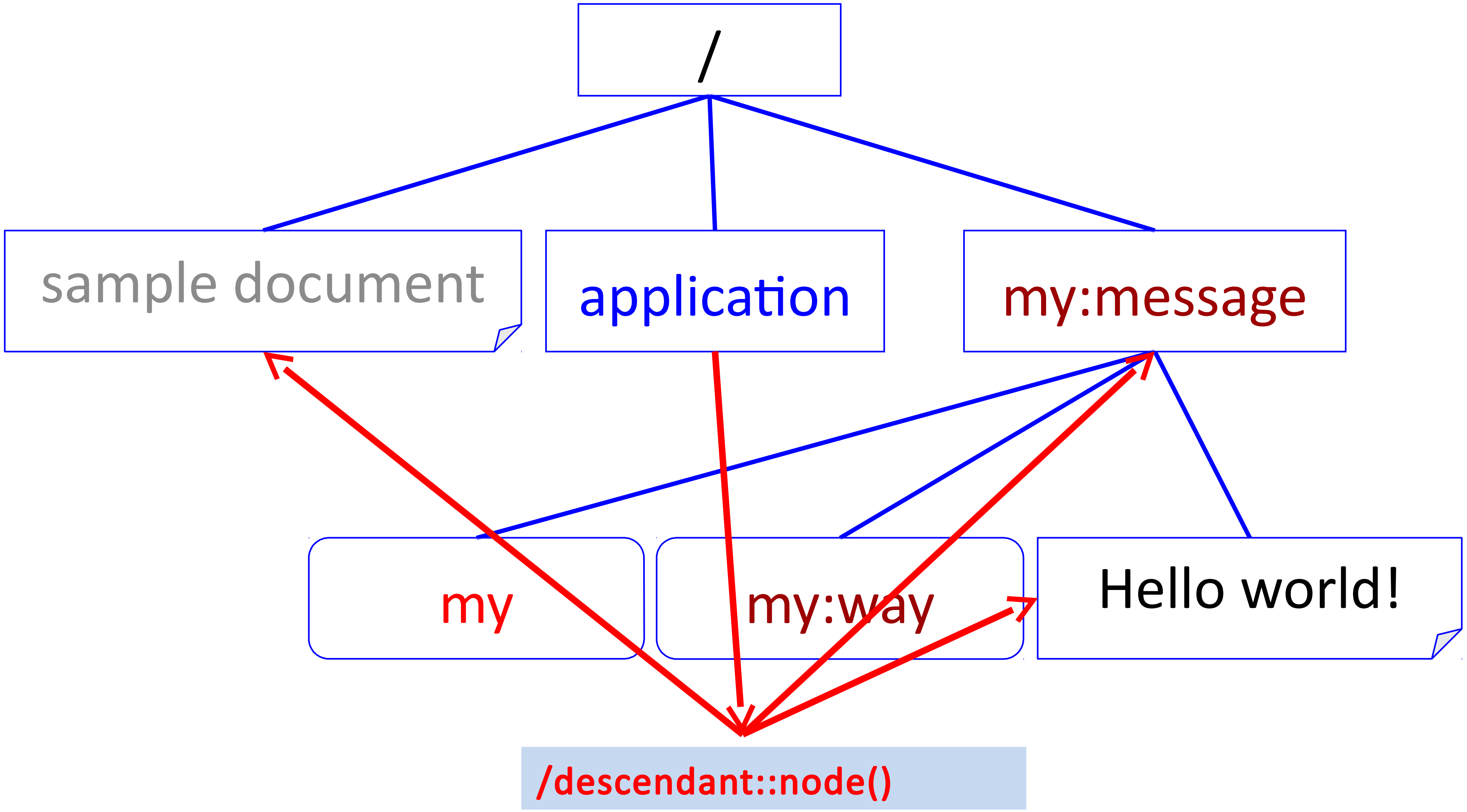
/child::*/child::text()

Path Expressions



/child::my:message/attribute::my:way

Path Expressions



Path Expressions

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

Path Expressions

- Examples for XPath predicate functions

```
string concat(string, string, string*)
boolean contains(string, string)
string substring-before(string, string)
number string-length(string?)
string normalize-space(string?)
```

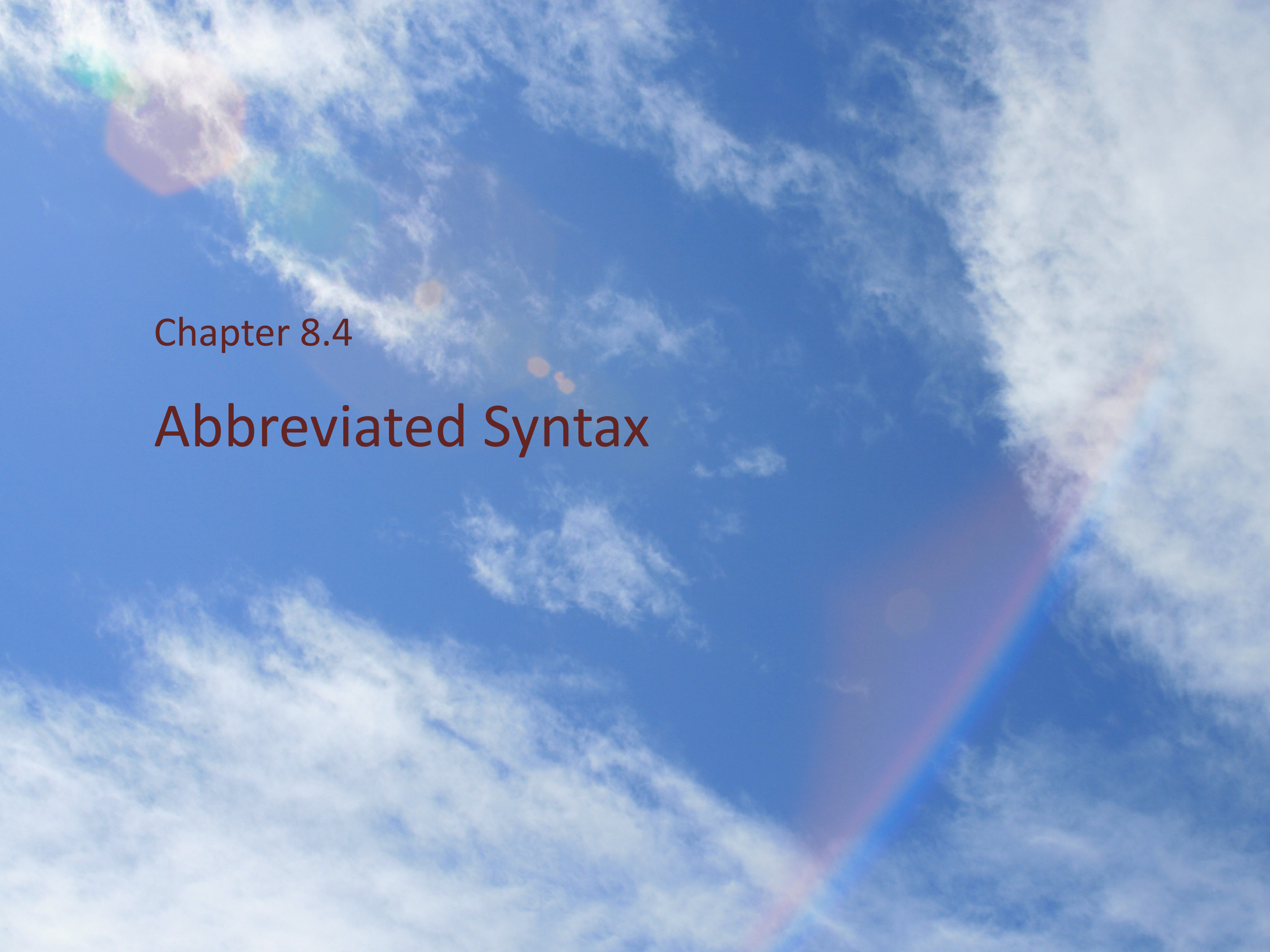
Path Expressions

- Predicate examples
  - /descendant::text()[contains(string(self::node()), "Hello")]

    selects all text nodes containing "Hello"
  - /descendant::node()[position()=5]

    selects the 5th node of the document
  - /descendant::node()[false()]

    empty set



Chapter 8.4

Abbreviated Syntax

Abbreviated Syntax

- Main rules

child::   can be omitted from a location step

.   is short for self::node()

..   is short for parent::node()

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

attribute::   can be abbreviated to @

Abbreviated Syntax

- Examples

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

  `*` selects all element children of the context node

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

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

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

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

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

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

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

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

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

Abbreviated Syntax

- Examples (cont'd.)

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

  `..` selects the parent of the context node

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

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

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

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

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

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

Abbreviated Syntax

- Examples (cont'd.)

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

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