XML Technologies

Doc. RNDr. Irena Holubova, Ph.D.

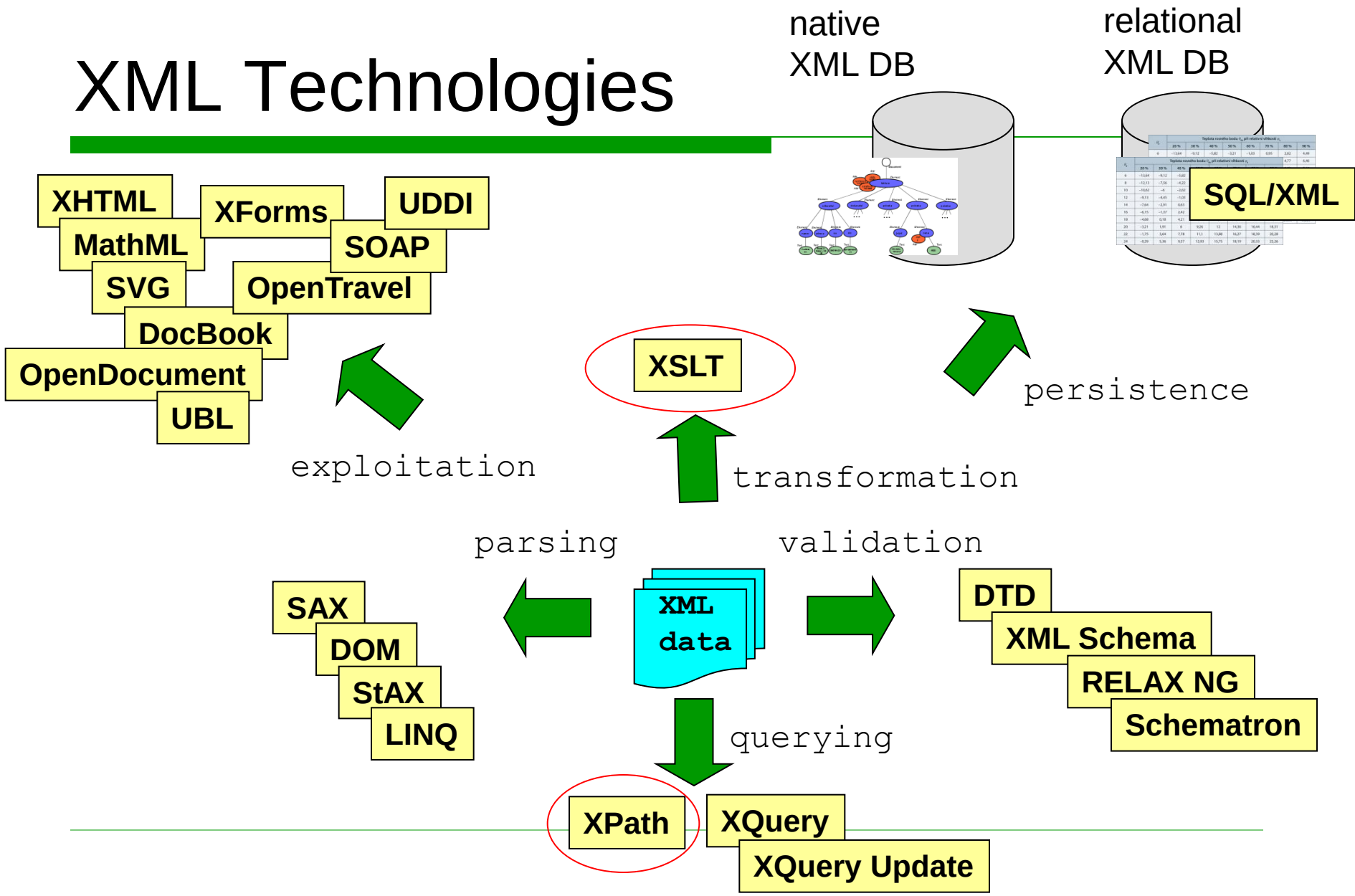
holubova@ksi.mff.cuni.cz

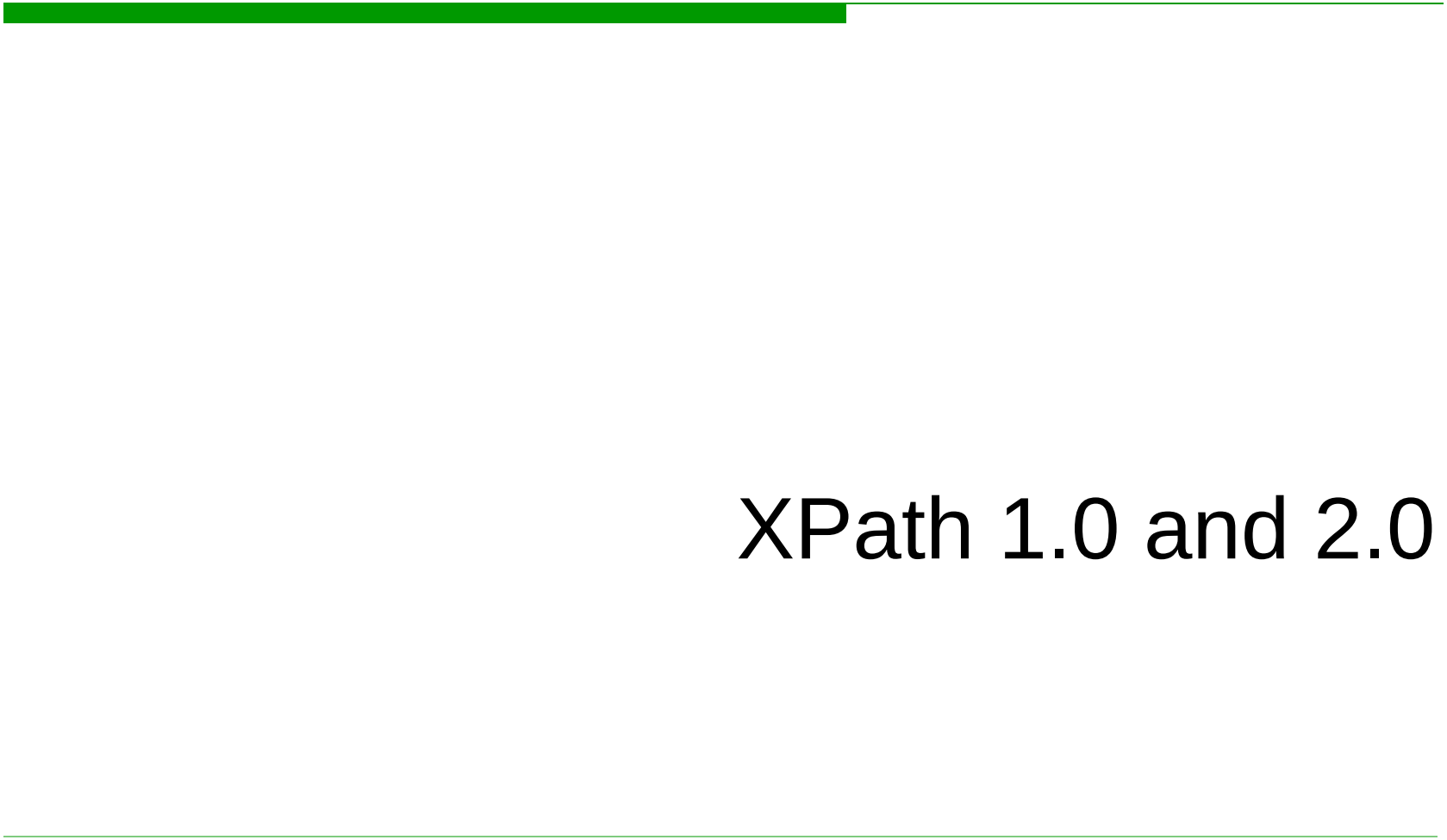
Web pages:
MFF: http://www.ksi.mff.cuni.cz/~holubova/NPRG036/
FEL: http://www.ksi.mff.cuni.cz/~holubova/A7B36XML/

Outline

- Introduction to XML format, overview of XML technologies
- DTD
- XML data models
- Interfaces for XML data
- XPath
- XSLT
- XQuery, XQuery Update
- XML schema languages
- SQL/XML
- An overview of standard XML formats
- XML data persistence

XML Technologies




XPath 1.0 and 2.0

XPath 1.0 – Brief Tutorial

- Path consists of steps

  /step1/step2/step3/…
  step1/step2/step3/…

- Step:

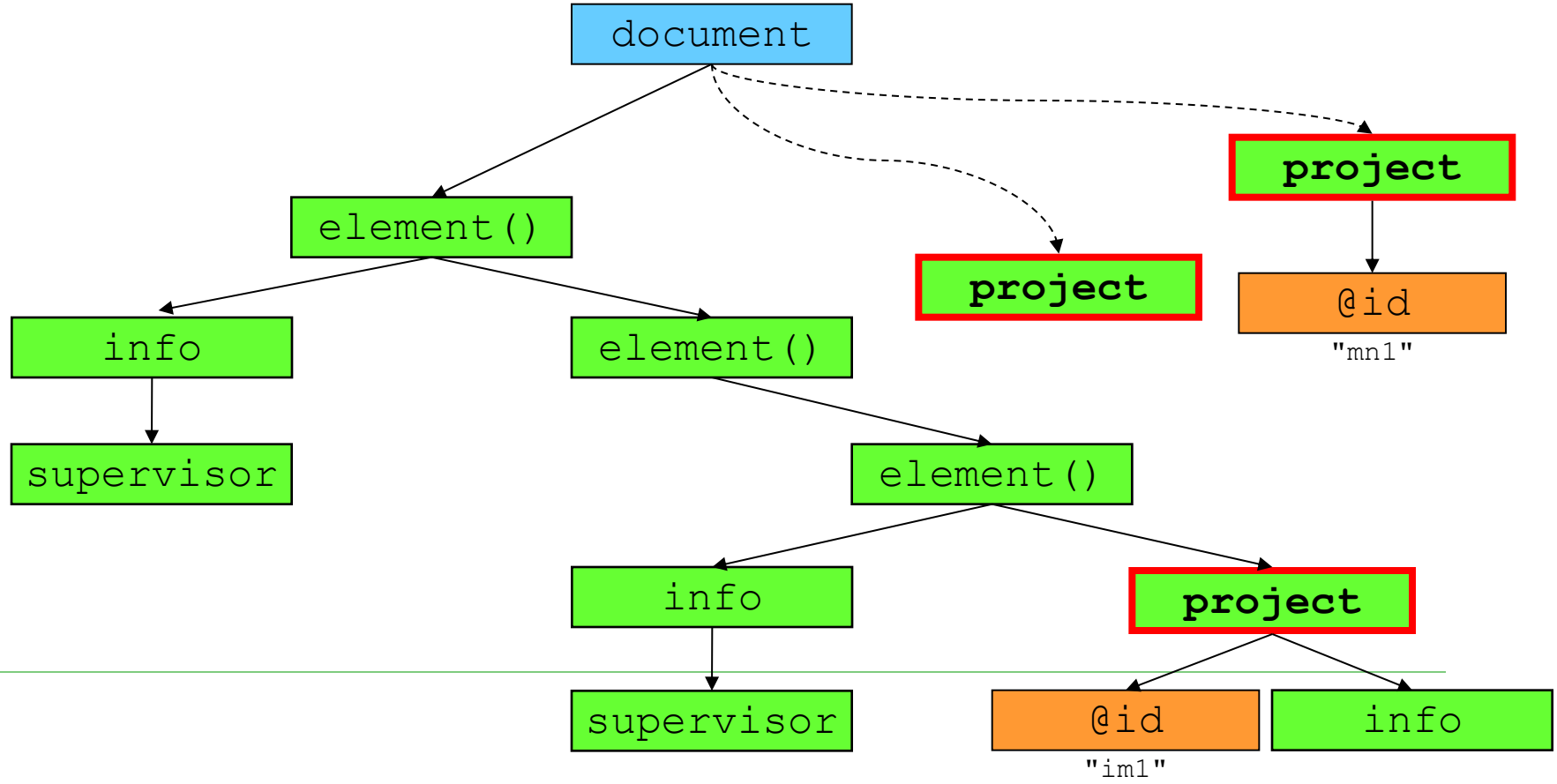
  axis::node-test predicate1 ... predicateN

  - Axis
    - Denotes the "direction" of the step
  - Node test
    - Denotes the type/name of nodes selected by the axis
  - Predicate
    - Logical condition further specifying requirements on the selected data

- Abbreviations simplify the expressions

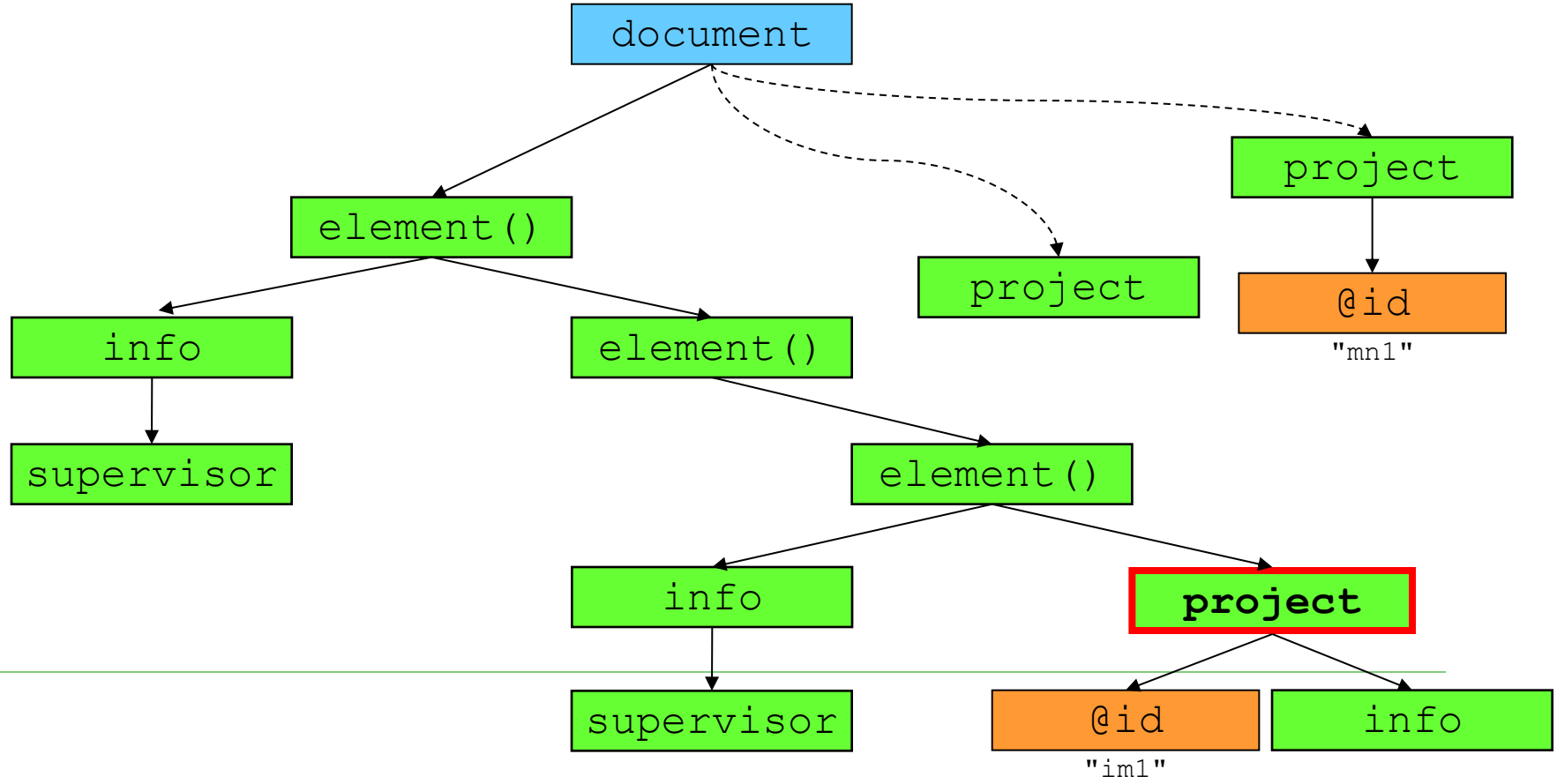
XPath 1.0 – Brief Tutorial

```
(//project[@id="im1"]/ancestor-or-self::*/info/supervisor)[last()]
```



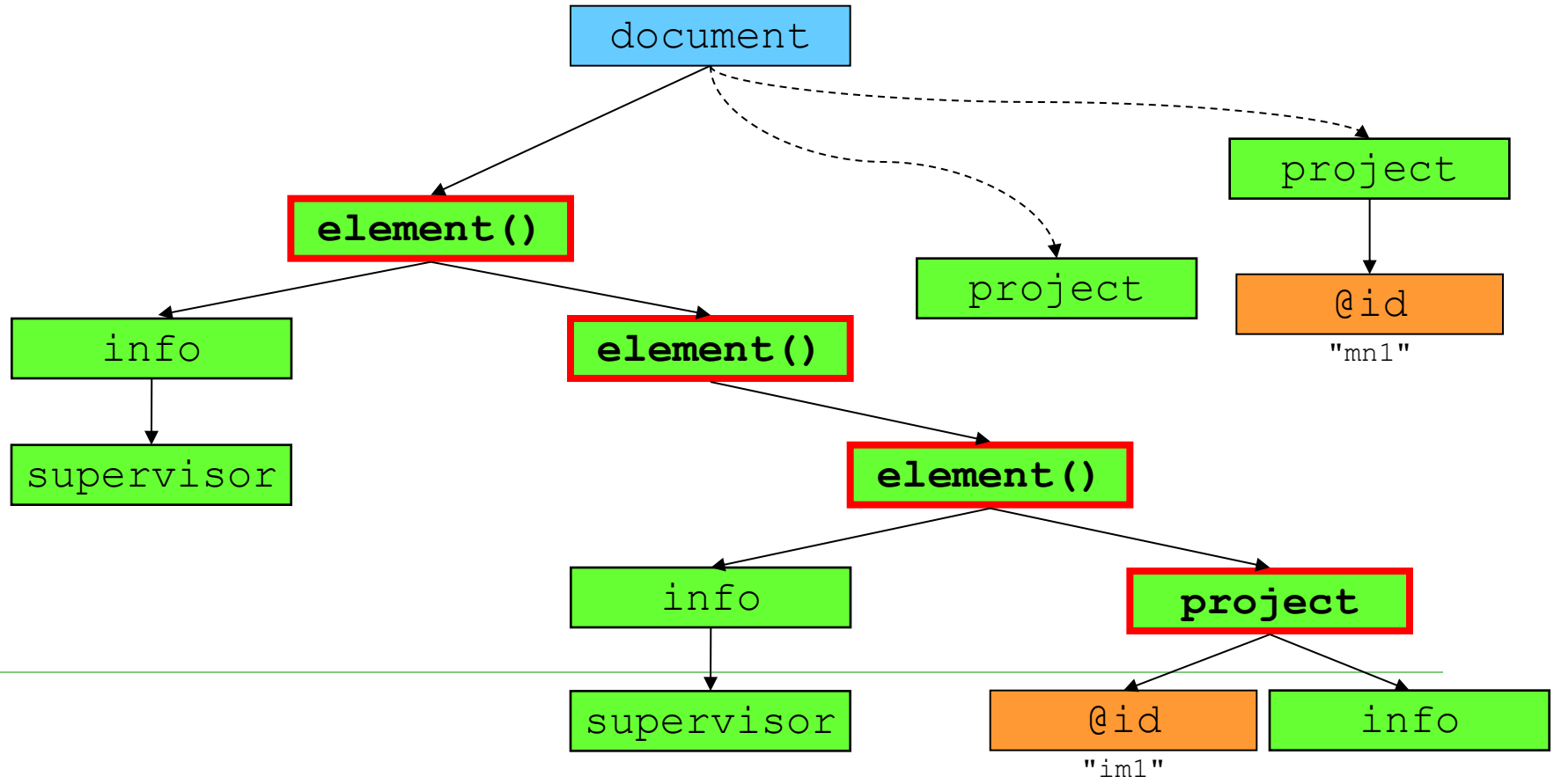
XPath 1.0 – Brief Tutorial

```
(//project[@id="im1"]/ancestor-or-self::*/info/supervisor)[last()]
```



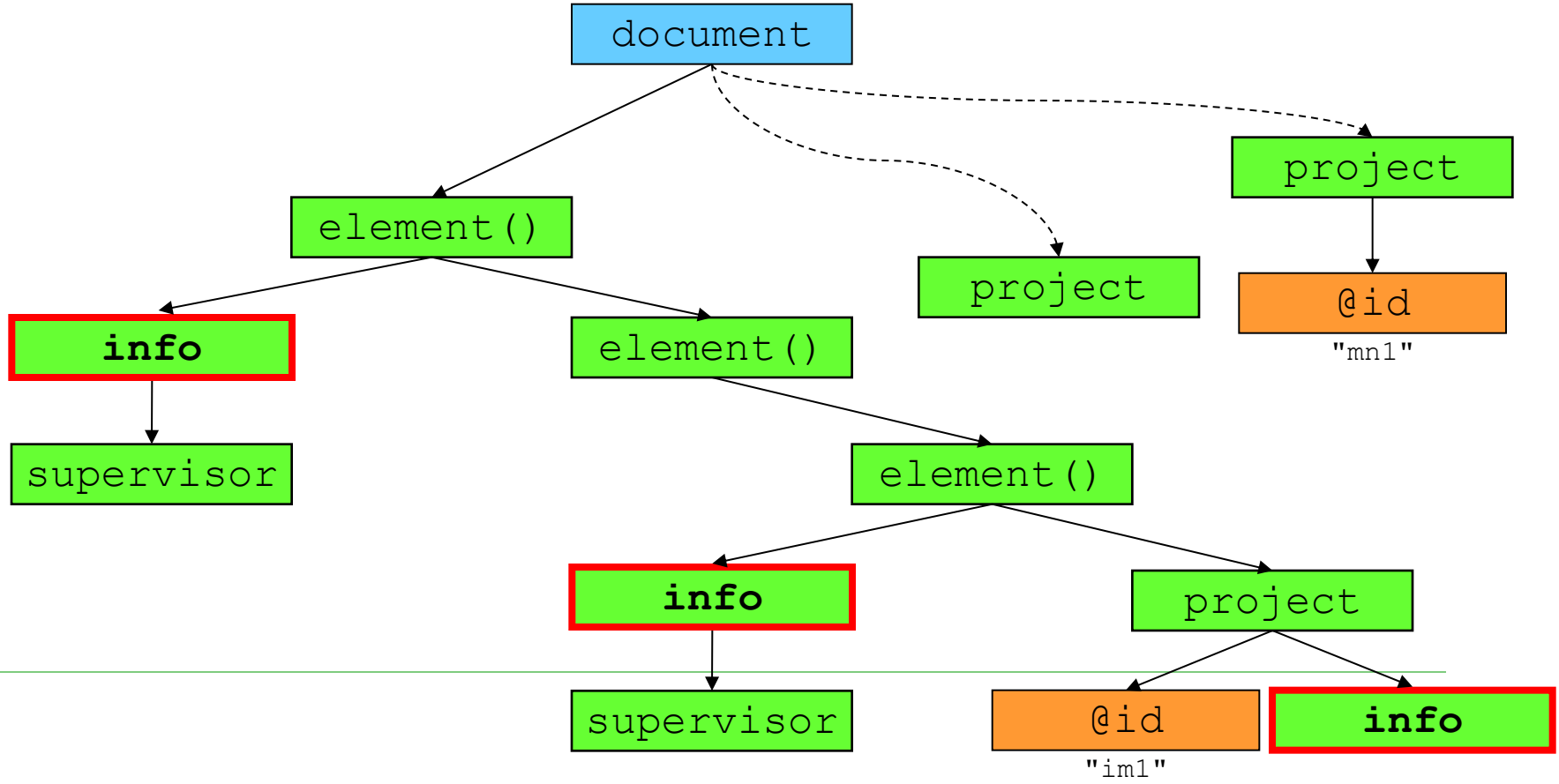
XPath 1.0 – Brief Tutorial

```
(//project[@id="im1"]/ancestor-or-self::*/info/supervisor)[last()]
```



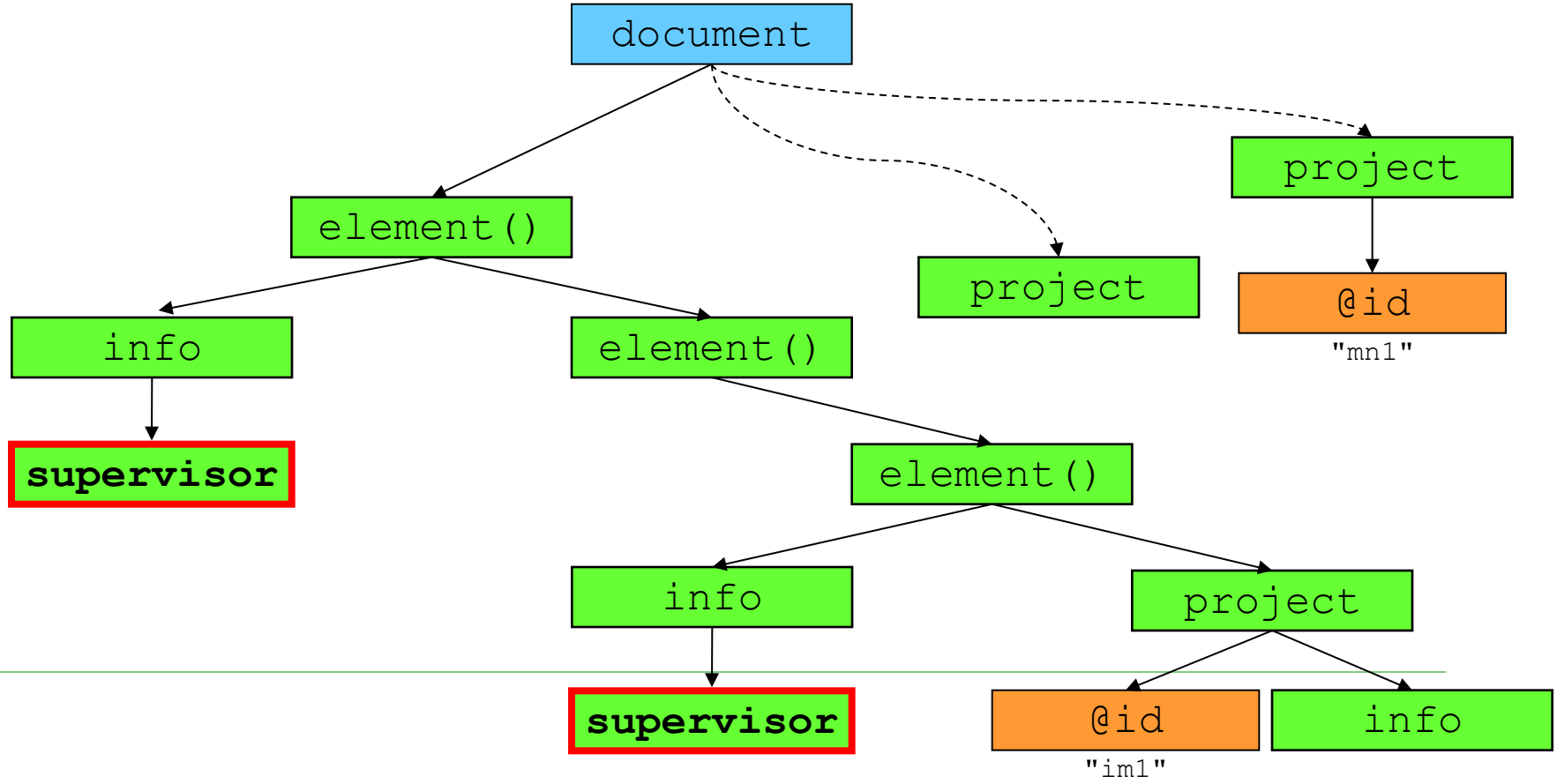
XPath 1.0 – Brief Tutorial

```
(//project[@id="im1"]/ancestor-or-self::*/info/supervisor)[last()]
```



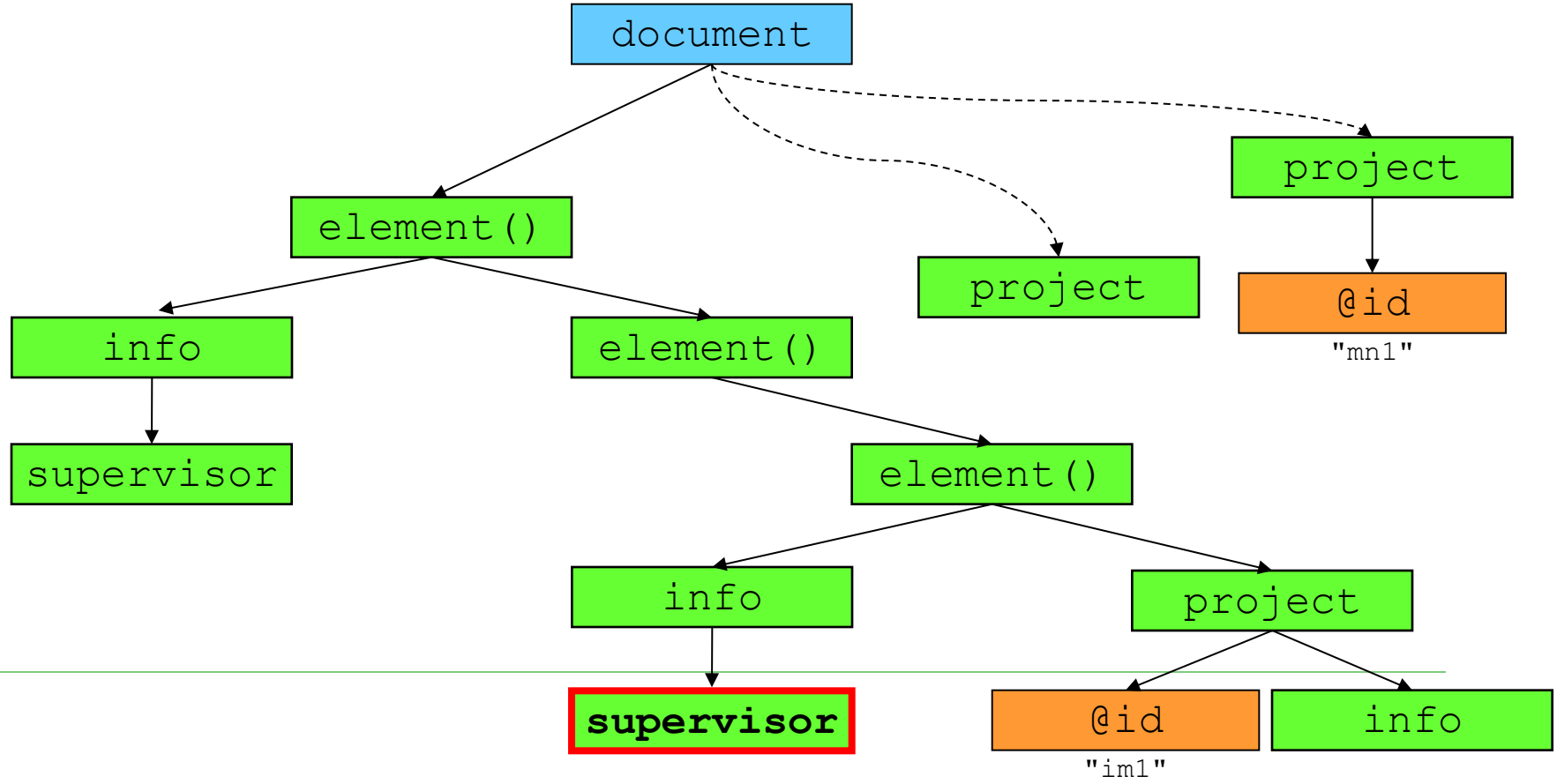
XPath 1.0 – Brief Tutorial

```
(//project[@id="im1"]/ancestor-or-self::*/info/supervisor)[last()]
```



XPath 1.0 – Brief Tutorial

```
(//project[@id="im1"]/ancestor-or-self::*/info/supervisor)[last()]
```



XPath 2.0

- Adds a huge number of new functions
  - see http://www.w3.org/TR/xpath-functions/
  - Prefixed with fn:
    - Namespace http://www.w3.org/2005/xpath-functions
- Works with ordered collections
  - Adds new constructs
    - Iterations of sequences (for loop), merging of sequences (union, intersect, except), conditions (if-then-else), quantifiers (some/every)
- Relation to XML Schema
  - Nodes are assigned with data types in the sense of XML Schema language
- Backward compatibility with XPath 1.0
  - Expressions from 1.0 return the same value
  - Few exceptions

http://www.w3.org/TR/xpath20/

XPath Data Types

- XPath 1.0
  - node-set, Boolean, number, string
- XPath 2.0
  - sequence, XML Schema data types

XPath 2.0 – Data Model

- sequence is an ordered collection of items
  - The result of an XPath 2.0 expression is a sequence
- item is either an atomic value or a node
  - atomic value = a value of any simple data type of XML Schema
  - node = an instance of any type of node
    - attribute, element, text, …

XPath 2.0 – Nodes

- Node has
  - Identity
  - Data type
    - XML Schema simple/complex data type
  - Typed value
    - Value according to a data type
      - Returned by fn:data()
  - String value
    - Type value converted to string (xs:string)
      - Returned by fn:string()

XPath 2.0 – Sequence

- Constructor ()
  - (1, 2, 3, 4)
- Constructor to
  - (1, 5 to 8) = (1, 5, 6, 7, 8)
- Constructor can contain XPath expressions
  - (//book, //cd)
- Constructor can be used as a step in an XPath expression
  - (1 to 100)[. mod 5 = 0]
  - //item/(price,value)
  - orders[fn:position() = (5 to 9)]

XPath 2.0 – Sequence

- Everything is a sequence
  - 1 = (1)
- Sequences are shallow = do not contain subsequences
  - (1, (2, 3), 4) = (1, 2, 3, 4)
- Sequences can contain duplicities
  - (1, 2, 1 to 2) = (1, 2, 1, 2)
- Sequences can contain atomic values and nodes together
  - (1, 2, //book)

XPath 2.0 – Iteration

- Construct for for iteration of sequences within expressions

```
for $i in (1,2,3)
return $i
```

- (1,2,3)

```
for $i in (10,20),
    $j in (1,2)
return ($i+$j)
```

- (11,12,21,22)

variables

```
for $varname1 in expression1,
    ...,
    $varnameN in expressionN
return expression
```

XPath 2.0 – Iteration

```
fn:sum( for
                $item in //item
        return
                $item/amount * $item/price )
```

XPath 2.0 – Quantifiers

```
some/every $variable in expression satisfies test_expression
```

- If the quantifier is some (every), the expression is true if at least one (every) evaluation of the test expression has the value true; otherwise the expression is false

```
every $part in /parts/part satisfies $part/@discounted
```

```
some $x in (1, 2, 3), $y in (2, 3, 4)
     satisfies $x + $y = 4
```

XPath 2.0 – Merging

- Union of sequences
  - union or | (| is already in XPath 1.0)
- Intersection of sequences
  - intersect
- Exception of sequences
  - except

```
expression1 union expression2

expression1 intersect expression2

expression1 except expression2
```

- Only for sequences of nodes
  - If the sequence includes an item which is not - error
- All operators eliminate duplicities
  - Two nodes are duplicate if they have the same identity

XPath 2.0 – Merging

```
for
        $item in /order//item
return
        $item/* except $item/price
```

XPath 2.0 – Merging

```
//item[color="blue"]
intersect
//order[ordernumber>1000]//*

<order number="O233" ordernumber="2911">
  <customer>
   <name>Martin Necasky</name>
   <email>martinnec@gmail.com</email>
  </customer>
  <items>
   <item code="V289348">
    <name>Name of item 289348</name>
    <color>blue</color>
    <count>1</count><price-item>1234</price-item>
   </item>
   ...
  </items>
</order>
```

XPath 2.0 – Comparison

- We already know operators for sets
  - =, !=, …
- New type: comparison of nodes
  - expression1 **is** expression2
    - true, if both the operands evaluate to the same node
  - expression1 **<<** expression2, resp. expression1 **>>** expression2
    - true, if the node on the left precedes/succeeds the node on the right (in the document order)
  - If any of the operands is converted to an empty sequence, the result is an empty sequence
  - If any of the operands is converted to a sequence longer than 1, error

XPath 2.0 – Comparison

- New type: comparison of values
  - lt, gt, le, ge, eq, ne meaning "less than", "greater than", "less or equal", "greater or equal", "equal", "non equal"
  - If any of the operands is converted to an empty sequence, the result is an empty sequence
  - If any of the operands is converted to a sequence longer than 1, error

XPath 2.0 – Comparison

```
//order[
        customer/name = "Martin Necasky"
          and
        . << (
            //order[
                    customer/name = "Martin Necasky"
                      and
                    fn:sum(
                            for $p in .//item
                            return $p/amount * $p/price
                            ) > 100000
                    ]
            )[1]
      ]
```

XPath 2.0 – Conditions

```
if (expression1)
  then (expression2)
  else (expression3)
```

```
for $product in /catalogue//product
return
 if ($product/discount = "yes")
  then $product/discount-price
  else $product/full-price
```

XSLT:
Advanced Constructs

XSLT 1.0 – Sorting

- Element xsl:sort
  - Within xsl:apply-templates or xsl:for-each
    - Influences the order of further processing
  - Attribute select
    - According to what we sort
  - Attribute order
    - ascending / descending
      - Default: ascending

XSLT 1.0 – Sorting

```
<xsl:for-each select="//item">
 <xsl:sort select="./name" />
 ...
</xsl:for-each>
```

```
<xsl:for-each select="book">
 <xsl:sort select="author/surname"/>
 <xsl:sort select="author/firstname"/>
 <p>
   <xsl:value-of select="author/surname"/>
   <xsl:text> - </xsl:text>
   <xsl:value-of select="title"/>
 </p>
</xsl:for-each>
```

XSLT 1.0 – Keys

- Element xsl:key
  - Attribute name
    - Name of key
  - Attribute match
    - XPath expression identifying elements for which we define the key
  - Attribute use
    - XPath expression identifying parts of the key
- Function key(key-name, key-value)
  - Finds the node with key having key-name and value key-value

XSLT 1.0 – Keys

```
<xsl:key name="product-key"
         match="product"
         use="./product-code" />

<xsl:for-each select="//item">
 <xsl:variable name="prod"
               select="key('product-key', ./@code)" />
 <xsl:value-of select="$prod/name" />
 <xsl:value-of select="$prod/vendor" />
</xsl:for-each>
```

XSLT 1.0 – Modes

- Processing of the same nodes in different ways = modes
- Attribute mode of element xsl:template and xsl:apply-template
  - Only for unnamed templates

XSLT 1.0 – Modes

```
<xsl:template match="/">
 <xsl:apply-templates mode="overview" />
 <xsl:apply-templates mode="full-list" />
</xsl:template>

<xsl:template match="item" mode="overview">
 ...
</xsl:template>

<xsl:template match="item" mode="full-list">
 ...
</xsl:template>
```

XSLT 1.0 – Combinations of Scripts

- Referencing to another XSLT script
  - Element xsl:include
    - Attribute href refers to an included script
    - The templates are "included" (copied) to the current script
  - Element xsl:import
    - Attribute href refers to an imported script
    - In addition, the rules from the current script have higher priority than the imported ones
    - xsl:apply-imports – we want to use the imported templates (with the lower priority)

XSLT 1.0 – Combinations of Scripts

```
<!-- stylesheet A -->
<xsl:stylesheet ...>

 <xsl:import  href="C.xsl" />
 <xsl:include href="B.xsl" />

</xsl:stylesheet>
```

XSLT 1.0 – Copies of Nodes

- Element xsl:copy-of
  - Attribute select refers to the data we want to copy
  - Creates a copy of the node including all child nodes
- Element xsl:copy
  - Creates a copy of the current node, but not its attributes or child nodes

XSLT 1.0 – Copies of Nodes

```
<xsl:template match="/">
  <xsl:copy-of select="."/>
</xsl:template>
```

```
<xsl:template match="/|@*|*|text()">
  <xsl:copy>
    <xsl:apply-templates select="@*|*|text()"/>
  </xsl:copy>
</xsl:template>
```

- Both create a copy of the input document, but in a different way

XSLT 2.0

- Uses XPath 2.0
  - XSLT 1.0 uses XPath 1.0
- Adds new constructs (elements)
  - The output (input) can be into (from) multiple documents
  - User-defined functions
    - Can be called from XPath expressions
  - Element xsl:for-each-group for grouping of nodes
- …and many other extensions
  - see http://www.w3.org/TR/xslt20/

XSLT 2.0 – Output and Input

- Element xsl:result-document
  - Attribute href
    - URL of output document
  - Attribute format
    - Format of the output document
    - Reference to an xsl:output element
- Element xsl:output
  - New attribute name
    - To enable referencing

XSLT 2.0 – Output and Input

```
<xsl:output name="orders-report-format" method="xhtml" .../>
<xsl:output name="order-format"         method="xml" ... />

<xsl:template match="/">
 <xsl:result-document href="orders-report.html"
                      format="orders-report-format">
  <html>
   <body><xsl:apply-templates /></body>
  </html>
 </xsl:result-document>

 <xsl:for-each select="document('orders.xml')//order">
  <xsl:result-document href="order{./@number}.html"
                       format="order-format">
   <xsl:apply-templates select="." />
  </xsl:result-document>
 </xsl:for-each>
</xsl:template>
```

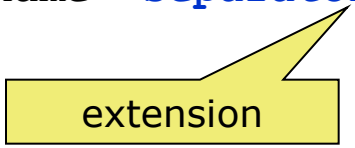
other input

XSLT 2.0 – Grouping of Nodes

- Grouping of nodes according to specific conditions
- Element xsl:for-each-group
  - Attribute select
    - Like for xsl:for-each
  - Attribute group-by
    - XPath expression specifying values according to which we group
  - … and other attributes for other types of grouping
  - Function current-group() returns items in the current group

XSLT 2.0 – Grouping of Nodes

```
<xsl:template match="/">
 <xsl:for-each-group select="document('products.xml')//product"
                     group-by="./category">
  <h1><xsl:value-of select="./category" /></h1>
  <p>
   <xsl:value-of select="current-group()/name" separator=", " />
  </p>
 </xsl:for-each-group>
</xsl:template>
```



XSLT 2.0 – User-defined Functions

- Element xsl:function
  - Attribute name
    - Name of function
  - Attribute as
    - Return value of function
  - Subelement xsl:param
    - Parameter of function
- Similar mechanism as named templates
- But we can use the functions in XPath expressions

XSLT 2.0 – User-defined Functions

```
<xsl:function name="mf:value-added-price" as="xs:anyAtomicType">
 <xsl:param name="price" as="xs:anyAtomicType"/>
 <xsl:value-of select="$price * 1.19" />
</xsl:function>

<xsl:template match="/">
 <html>
  <body>
   <xsl:value-of select="mf:value-added-price
     (sum(for $p in //item return $p/price * $p/amount))" />
  </body>
 </html>
</xsl:template>
```

XPath 2.0

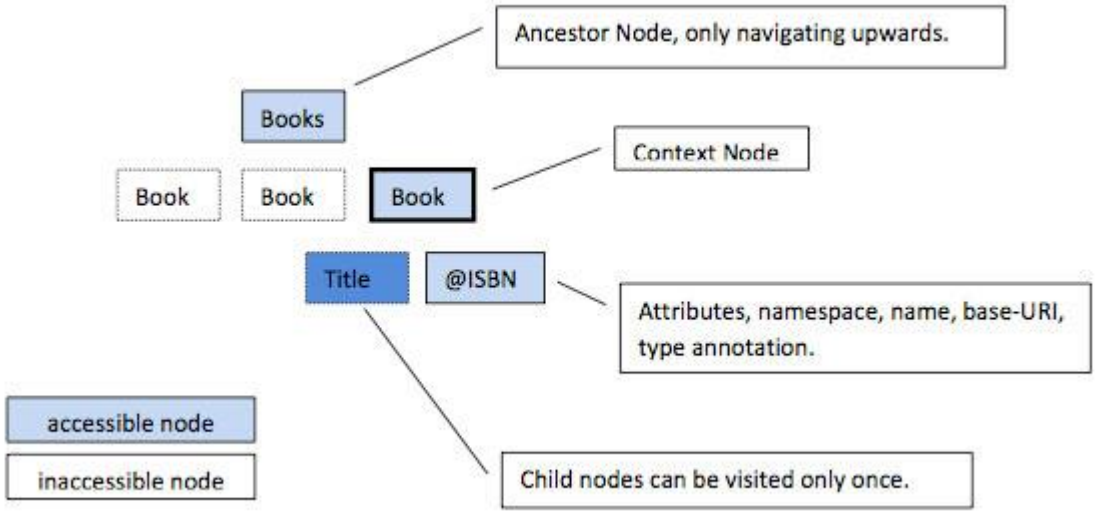
XSLT 3.0

- Currently just W3C candidate recommendation
  - To be used in conjunction with XPath 3.0
- Main extensions:
  - Streaming mode of transformations
    - Neither the source document nor the result document is ever held in memory in its entirety
    - Motivation: we do not want to load the entire document in memory
  - Higher order functions
  - Extended text processing
  - Improves modularity of large stylesheets
  - …

http://www.w3.org/TR/xslt-30/

XSLT 3.0 and Streaming

- Restrictions to be aware of:
  - We have access only to the current element attributes and namespace declaration
  - Sibling nodes and ancestor siblings are not reachable
  - We can visit child nodes only once



A processor that claims conformance with the streaming option offers a guarantee that an algorithm will be adopted allowing documents to be processed that are orders-of-magnitude larger than the physical memory available.

XSLT 3.0 and Streaming

```
<?xml version="1.0"?>
<xsl:stylesheet version="3.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

 <xsl:template match="/">
  <xsl:stream href="books.xml">
   <xsl:iterate select="/books/book">
    <xsl:result-document
         href="{concat('book', position(),'.xml')}">
      <xsl:copy-of select="."/>
    </xsl:result-document>
    <xsl:next-iteration/>
   </xsl:iterate>
  </xsl:stream>
 </xsl:template>
</xsl:stylesheet>
```

We explicitly indicate to stream the execution of its instruction body

XSLT 3.0 and Higher-Order Functions

- Higher order functions = functions that either take functions as parameters or return a function
- XSLT 3.0 introduces the ability to define anonymous functions
  - Enables meta-programming using lambda expressions
- Example:
  - (x, y) $\rightarrow$ x*x + y*y … lambda expression that calculates the square of two numbers and sums them
  - x $\rightarrow$ (y $\rightarrow$ x*x + y*y) … equivalent expression that accepts a single input, and as output returns another function, that in turn accepts a single input

XSLT 3.0 and Higher-Order Functions

```
<?xml version='1.0'?>
<xsl:stylesheet
     version="3.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xsl:template match="/">
    <xsl:variable name="f1" select="
      function($x as xs:integer) as (function(xs:integer) as
xs:integer){
        function ($y as xs:integer) as xs:integer{
          $x * $x + $y * $y
          }
      } "/>
    <xsl:value-of select="$f1(2)(3)"/>
  </xsl:template>
</xsl:stylesheet>
```

Variable f1 is assigned to an anonymous function that takes an integer and returns a function that takes an integer and returns an integer

XSLT 3.0 and Higher-Order Functions

- Support for common lambda patterns (operators)
  - map – applies the given function to every item from the given sequence, returning the concatenation of the resulting sequences
  - filter – returns items from the given sequence for which the supplied function returns true
  - fold-left – processes the supplied sequence from left to right, applying the supplied function repeatedly to each item, together with an accumulated result value
  - fold-right – respectively
  - map-pairs – applies the given function to successive pairs of items taken one from sequence 1 and one from sequence 2, returning the concatenation of the resulting sequences

http://www.w3.org/TR/xpath-functions-30/

XSLT 3.0 and Higher-Order Functions

```
<?xml version="1.0"?>
<xsl:stylesheet  version="3.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:variable name="list" select="(10,-20,30,-40)"/>

  <xsl:template match="/">
    <xsl:variable name="f1" select="
      function($accumulator as item()*, $nextItem as item()) as item()*
        {
        if ($nextItem &gt; 0) then
           $accumulator + $nextItem
        else
           $accumulator
        }"/>
    <xsl:value-of select="fold-left($f1, 0, $list)"/>
  </xsl:template>
</xsl:stylesheet>
```

Folding that sums only positive numbers from a list