summer school

21st September 2017

EVOLVED BINARY

XML Databases and XQuery

Adam Retter

adam@evolvedbinary.com

@adamretter

http://static.adamretter.org.uk/xmlss17.pdf
http://static.adamretter.org.uk/xmlss17-full.pdf

Who are you?

- Programmer / Consultant
  - XQuery / XSLT
  - Scala / Java
  - Concurrency

- Core contributor to eXist XML Database

- Contributor to Facebook's RocksDB

- Creator of “Granite” polyglot database

- W3C XQuery WG Invited expert

- www.adamretter.org.uk

Today's Plan

summer school

Session 1

Part 1 (45 to 60 minutes)

- Brief Introduction to XQuery
- Understanding the XDM
- Practical XQuery basics

Part 2 (45 to 60 minutes)

- Hands-on: Developing some basic XQuery on eXist-db

Break (30 minutes)

Session 2

Part 1 (45 minutes)

- XML Databases
- Store/Retrieve with an XML Database
- Advanced XQuery

Part 2 (45minutes)

- Hands-on: Developing XQuery for the Web on eXist-db

summer school

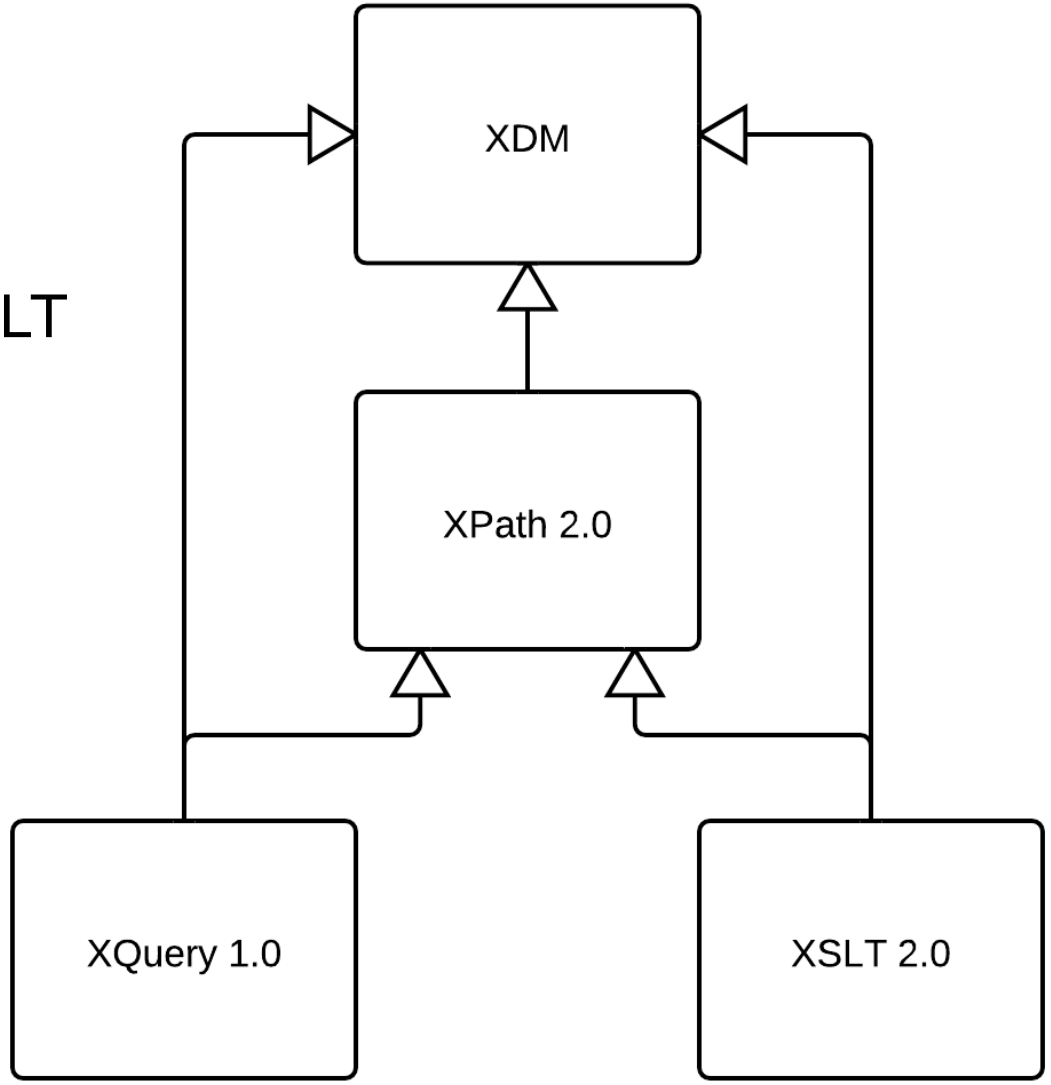
Quick Introduction to XQuery

XQuery is...

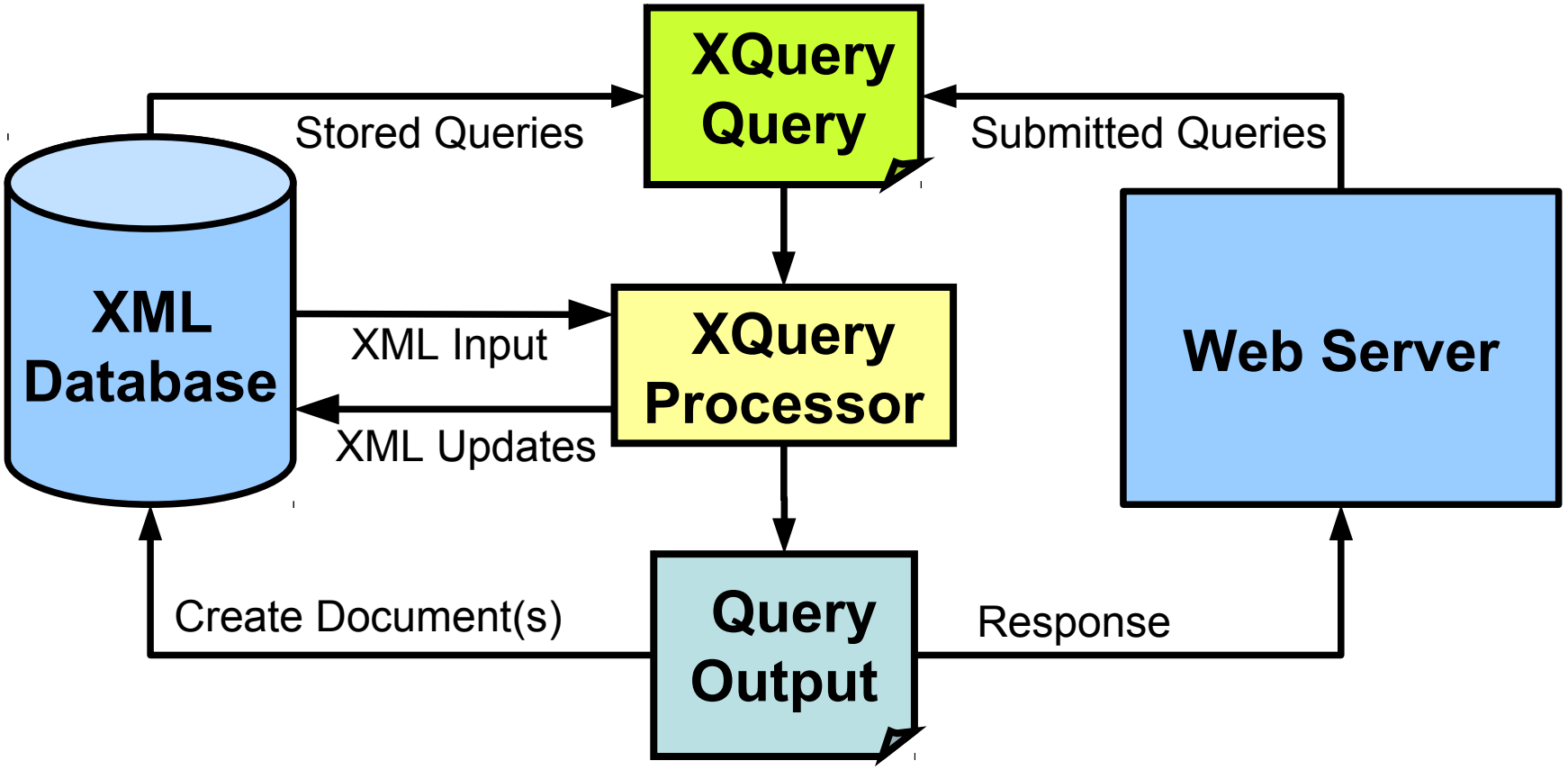
- XML Query Language
  - A W3C Standard
  - Superset of XPath
  - Closely related to XSLT 2.0
  - Is NOT written in XML
- A Query Language!
  - Pull information from one or more XML documents
  - The "*SQL of XML*"
- A Transformation Language
  - Transform data (XML, HTML, JSON, Text, etc.) from one form or structure to another

Where does XQuery fit?

- Its kinda just XPath++
  - If you know XPath...
- Much in common with XSLT
  - XDM and XPath
- XDM / XPath 2.0
  - XQuery 1.0 / XSLT 2.0
- XDM 3.0 / XPath 3.0
  - XQuery 3.0 / XSLT 3.0
- XDM 3.1 / XPath 3.1
  - XQuery 3.1 / …?



XQuery Processing Model (Platform)



Why XQuery?

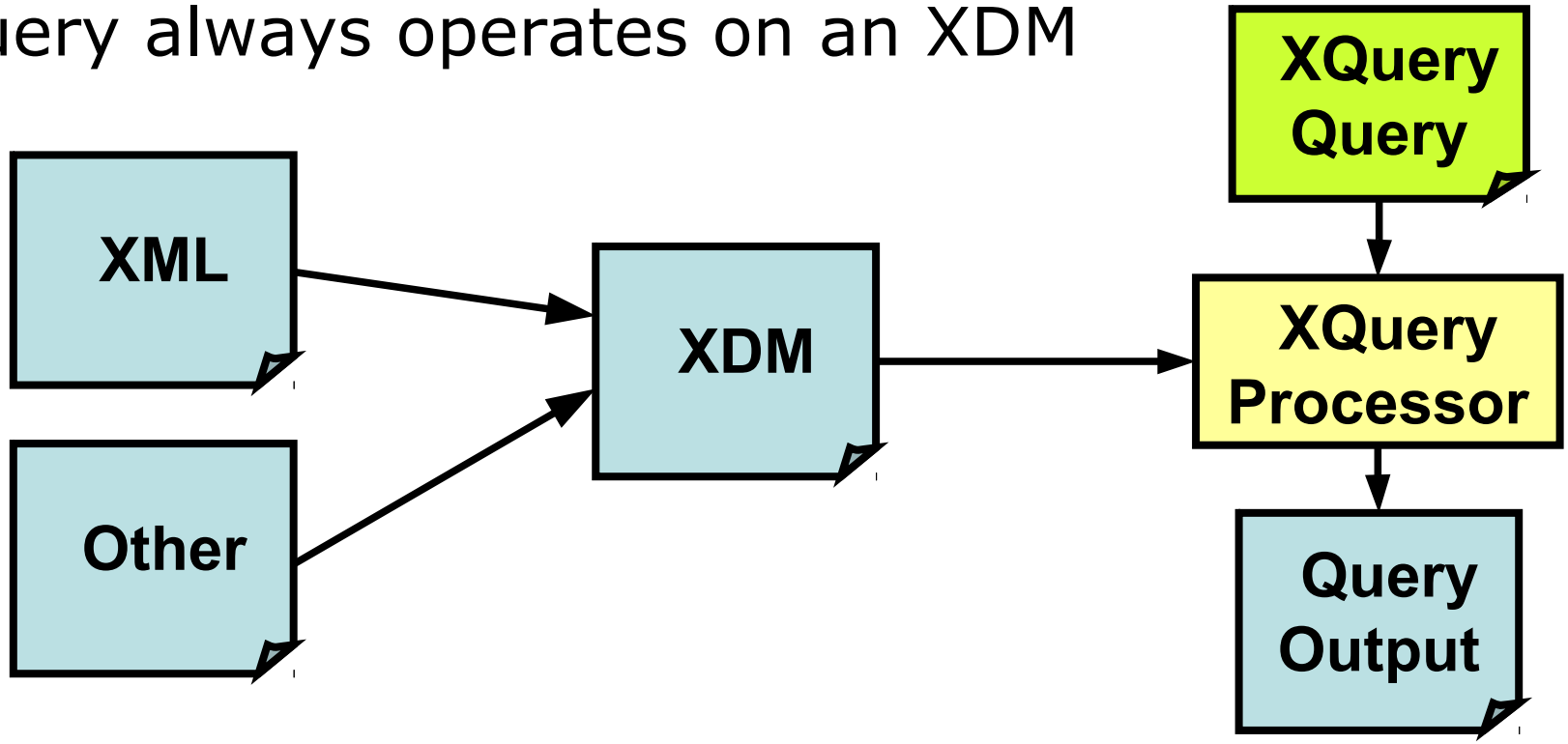
- Why not just use XSLT?
  - Well you could!
- XSLT is best suited to Transformation
  - Typically: Document → XSLT → Document
- XQuery is best suited to query/search
  - Designed to work well over many documents
  - XSLT does not have Update extensions
  - XSLT does not have Full Text extensions

summer school

XDM: XQuery and XPath Data Model

What is XDM?

- XQuery always operates on an XDM



- XDM is the Data Model for XPath and XQuery
- Understanding basics of XDM is key!

XDM Basics

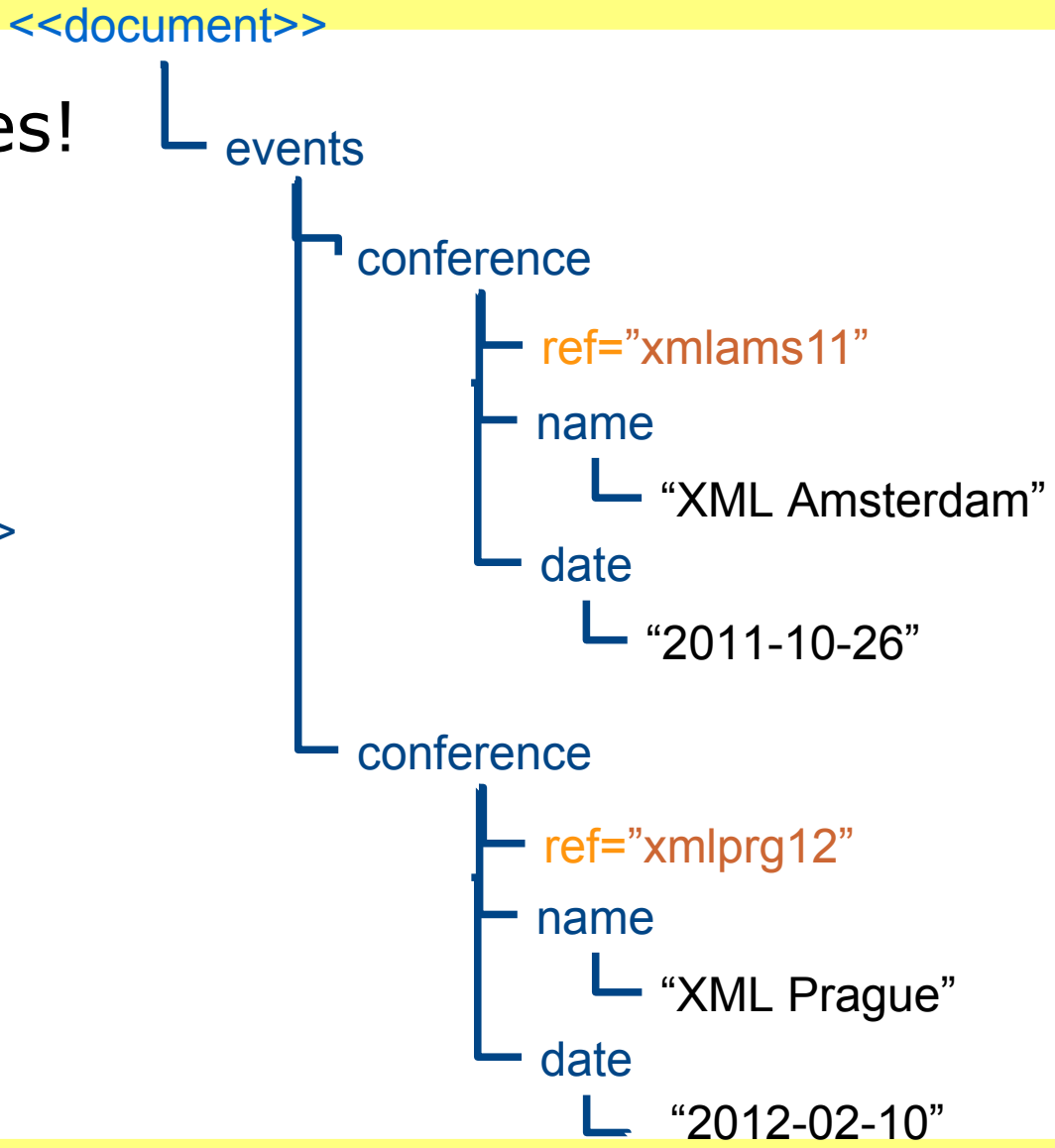
- An XDM consists of Items and Sequences
  - Builds on XML Infoset and XML Schema
- Items are of two main types:
  - Node or Atomic Value (3.0 adds Function Item Type, (3.1 adds Maps and Arrays))
- Nodes
  - XML Documents are made of these!
  - Different types of nodes:
    - document, element, attribute, text, comment, processing-element
  - Have a Unique Identity!

```
<root>
  <hello>world</hello>
  <hello>world</hello>
  ...
```

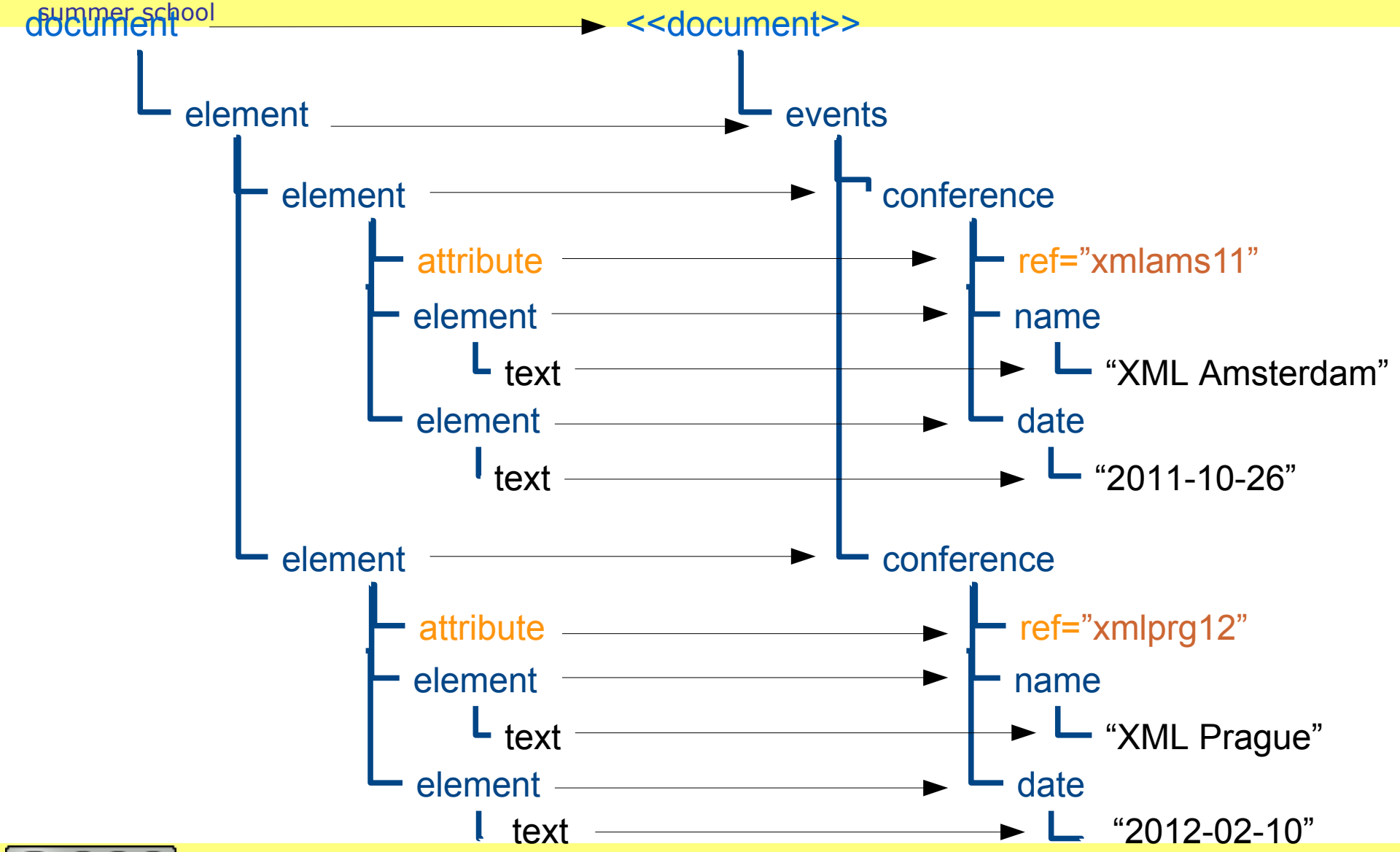
XDM - Node Trees

XML: Its a Tree of Nodes!

```
<events>
   <conference ref="xmlams11">
      <name>XML Amsterdam</name>
      <date>2011-10-26</date>
   </conference>
   <conference ref="xmlprg12">
      <name>XML Prague</name>
      <date>2012-02-10</date>
   </conference>
</events>
```



XDM – Node Trees



XDM – Atomic Values

- Atomic Values
  - i.e. A literal value, e.g. "*hello*"
  - These are NOT Nodes!
  - Many different types of Atomic Value:
    - See: XML Schema Part 2: Datatypes
      - xs:string e.g. "I am a String"
      - xs:int e.g. 1234
      - xs:date e.g. xs:date("2004-03-01")

- Useful Links:
  - http://www.w3.org/TR/xpath-datamodel/#types-hierarchy
  - http://www.w3.org/TR/xmlschema-2/#built-in-datatypes

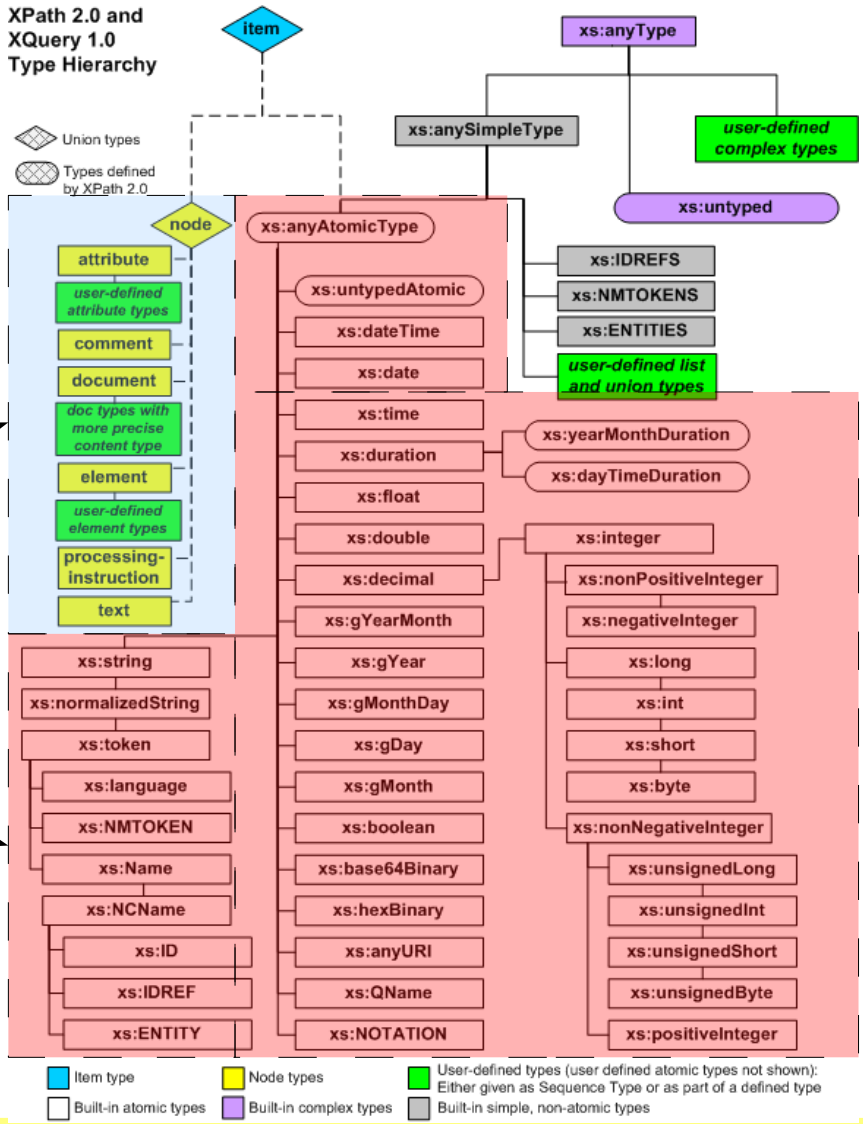
XDM – Type Hierarchy

Simple!

- Probably only use a few of the Atomic Value Types

Nodes

Atomic Values



Modified from:
W3C XQuery 1.0 and XPath 2.0 Data Model (XDM) (Second Edition)

XDM – Nodes Quiz

- Quiz on XDM Nodes

```
<document lang="en_GB">
   <fragment1>Hello there <gn>James</gn> <fn>Smith</fn>, </fragment1>
   <fragment2>how are you today?</fragment2>
</document>
```

1) What kind of node is '*fragment2*'?

2) What are the names of the attributes?

3) How many text nodes are in the document?

4) How many nodes are in the document?

XDM - Sequences

- Sequences
  - An Ordered List
  - Sequence Constructor starts with '*(*' and ends with '*)*'
  - Consist of Zero or More Items

    ("hello", "world")
  - Can be mix of Nodes and Atomic Values

    ("hello", <gn>james</gn>, <fn>smith</fn>)
  - No Nested Sequences!

    ("a", "b", ("c", "d")) *becomes:* ("a", "b", "c", "d")

XDM - Sequences

- Sequences
  - An Item == Sequence containing just that Item

    ("hello") *is the same as:* "hello"
  - A Sequence with Zero Items, is an Empty Sequence

    () *is the Empty Sequence*
  - Can be the parameter to a function, a computed result, or the result of an expression e.g.

    "Find me all the names?"

    ```
    //name
    ```

    - Returns the *Sequence* of two Elements:

      ```
      (<name>adam</name>, <name>bob</name>)
      ```

Comparison Operators

- XQuery has two types of Comparison Operators

	Atomic Values	Sequences
Equal to	eq	=
Not equal to	ne	!=
Greater than	gt	>
Greater than or equal to	ge	>=
Less than	lt	<
Less than or equal to	le	<=

```
("james", "simon", "mark", "bob") = "mark"
```

- is ?

```
("james", "simon", "mark", "bob") = ("mark", "james")
```

- is ?

```
("james", "simon", "mark", "bob") = ("mark", "cliff")
```

- ...and this??

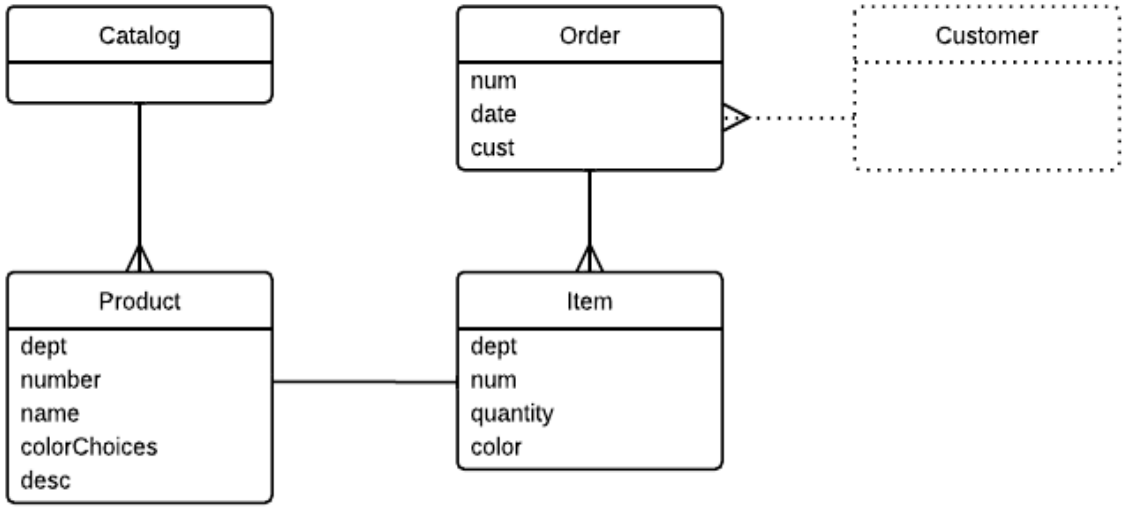
summer school

Practical XQuery

Example Data

- Initially to illustrate some simple XQuery we will use some very simple data centric XML

- Two documents:
  - Catalog (of Products)
  - Orders (of Products)



Example Data

catalog.xml

```
<catalog>
   <product dept="WMN" commenced=”2009-02-04”>
      <number>557</number>
      <name language="en">Linen Shirt</name>
      <colorChoices>beige sage</colorChoices>
   </product>
   <product dept="ACC" commenced=”2009-02-04”>
      <number>563</number>
      <name language="en">Ten-Gallon Hat</name>
   </product>
   <product dept="ACC" commenced=”2012-01-01”>
      <number>443</number>
      <name language="en">Golf Umbrella</name>
   </product>
   <product dept="MEN" commenced=”2010-08-09”>
      <number>784</number>
      <name language="en">Rugby Shirt</name>
      <colorChoices>blue/white blue/red</colorChoices>
      <desc>Our <i>best-selling</i> shirt!</desc>
   </product>
</catalog>
```

Example Data

order.xml

```
<order num="00299432" date="2004-09-15" cust="0221A">
   <item dept="WMN" num="557" quantity="1" color="beige"/>
   <item dept="ACC" num="563" quantity="1"/>
   <item dept="ACC" num="443" quantity="2"/>
   <item dept="MEN" num="784" quantity="1" color="blue/white"/>
   <item dept="MEN" num="784" quantity="1" color="blue/red"/>
   <item dept="WMN" num="557" quantity="1" color="sage"/>
</order>
```

Question 1...

1. How do we get a list of departments?

summer school

XQuery Example

input document

```
<catalog>
   <product dept="WMN" commenced="2009-02-04">
      <number>557</number>
      <name language="en">Linen Shirt</name>
      <colorChoices>beige sage</colorChoices>
   </product>
   <product dept="ACC" commenced="2009-02-04">
      <number>563</number>
```

query

```
doc("/db/shop/catalog.xml")/catalog/product/string(@dept)
```

results

WMN ACC ACC MEN

Improved XQuery Example

summer school

input document

```
<catalog>
   <product dept="WMN" commenced="2009-02-04">
      <number>557</number>
      <name language="en">Linen Shirt</name>
      <colorChoices>beige sage</colorChoices>
   </product>
   <product dept="ACC" commenced="2009-02-04">
      <number>563</number>
```

query

```
distinct-values(doc("/db/shop/catalog.xml")/catalog/product/@dept)
```

results

WMN ACC MEN

Q: What is distinct-values?
Q: Why don't we need string()?
Q: What other functions are available?

Question 2...

2. How do we calculate total orders by department?

XQuery Example

summer school

input document

```
<order num="00299432" date="2004-09-15" cust="0221A">
   <item dept="WMN" num="557" quantity="1" color="beige"/>
   <item dept="ACC" num="563" quantity="1"/>
   <item dept="ACC" num="443" quantity="2"/>
   <item dept="MEN" num="784" quantity="1" color="blue/white"/>
   <item dept="MEN" num="784" quantity="1" color="blue/red"/>
   <item dept="WMN" num="557" quantity="1" color="sage"/>
</order>
```

query

```
for $d in distinct-values(doc("/db/shop/order.xml")//item/@dept)
let $items := doc("/db/shop/order.xml")//item[@dept = $d]
order by $d
return
   <department name="{$d}"
      totalQuantity="{sum($items/@quantity)}"/>
```

results

```
<department name="ACC" totalQuantity="3"/>
<department name="MEN" totalQuantity="2"/>
<department name="WMN" totalQuantity="2"/>
```

NOTE: The Context

- You need something to process:
  - External. Set by processor (also external var's!)
  - Request, by explicit function call
    - A document

```
doc("/db/shop/catalog.xml")
```

    - A collection of documents

```
collection("/db/shop/products")
```

- Or... XQuery generates original data

NOTE: Documents and Collections

- doc() and collection() functions take a URI
- URI may or may not be de-referenced
- Both functions return document node(s).
- What is a Collection?
  - Implementation defined
    - A folder? Hierarchical?
    - URI! A label?
- Typically followed by an expression.

NOTE: Collections vs. Documents

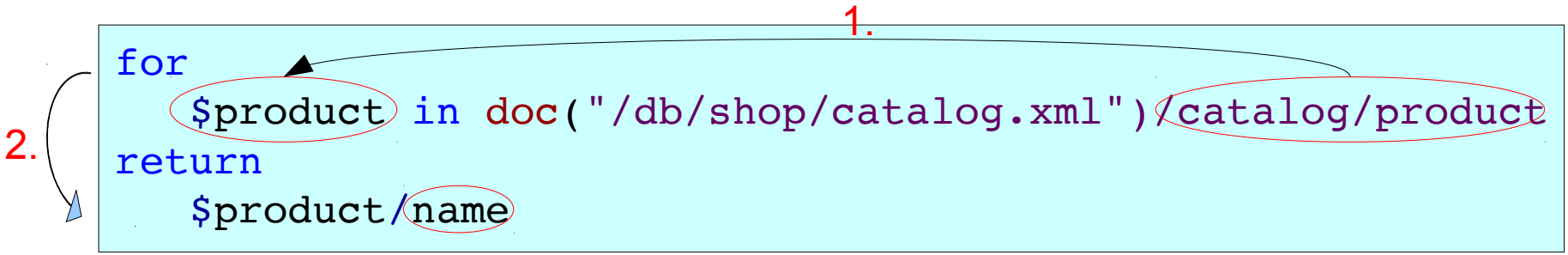
- 1 big document vs. collection of smaller documents
  - Why and When is it best to split?
    - How does your data come to you?
    - How do you deliver your data?
    - What is the unit of demarcation?
  - A Node is a Node is a Node
    - Do you need to know that it was originally X documents?
    - Can Transform back into any shape?
  - How do you update?
  - Processor constraints!

Expressions

- Path Expressions e.g. /xml/summer/school
- FLOWR Expressions:
  - **for**
    - creates a sequence of nodes
  - **let**
    - binds a sequence to a variable
  - **where**
    - filters the nodes on a boolean expression
  - **order by**
    - sorts the nodes
  - **return**
    - gets evaluated once for every node

Simple FLWOR Example

- Just the 'F' and 'R':



1. Bind the $product variable to each /catalog/product node in turn during iteration

2. Return the evaluation of $product/name for each iteration
   i.e. each /catalog/product/name element

Simple FLWOR Example

- Result:

```
<name language="en">Linen Shirt</name>
<name language="en">Ten-Gallon Hat</name>
<name language="en">Golf Umbrella</name>
<name language="en">Rugby Shirt</name>
```

  - An XML Fragment!

Simple FLWOR Example with Position

- Iteration position can be bound with '*at*'

```
for
   $product at $i in doc("catalog.xml")/catalog/product
return
   <product idx="{$i}">{$product/name/text()}</product>
```

- Just like in XSLT, expressions can be evaluated inline using **{***expression***}** notation

- New Nodes can be constructed to change structure
  - Direct Constructors
  - Computed Constructors

```
element product {
    attribute idx { 99 },
    text { "Our New Product" }
}
```

Simple FLWOR Example with Position

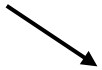
- Result:

```
<product idx="1">Linen Shirt</product>
<product idx="2">Ten-Gallon Hat</product>
<product idx="3">Golf Umbrella</product>
<product idx="4">Rugby Shirt</product>
```

- Result is in Document Order!
- 'Product' is directly constructed element, with attr.
- text() node of Product Name was copied
- This query does not produce a well-formed document!!!

Simple FLWOR Example with Position

```
<products>{
    for
       $product at $i in doc("catalog.xml")/catalog/product
    return
       <product idx="{$i}">{$product/name/text()}</product>
}</products>
```



```
<products>
   <product idx="1">Linen Shirt</product>
   <product idx="2">Ten-Gallon Hat</product>
   <product idx="3">Golf Umbrella</product>
   <product idx="4">Rugby Shirt</product>
</products>
```

- What about *document{ … }?*

FLWOR - Bindings

```
for
   $product in doc("catalog.xml")/catalog/product
let
   $catalog-age := days-from-duration(
     current-date() - xs:date($product/@commenced)
   )
return
    <product ref="{$product/number}">
       Added to the catalog {$catalog-age} days ago.
    </product>
```

1. Bind the $catalog-age variable to each expression during iteration

2. Return the evaluation of
   $catalog-age expression for the iteration

- A FLWOR expression can have any number of bindings

FLWOR – Where Clauses

```
for
    $product in doc("catalog.xml")/catalog/product
where
    $product/@dept eq "ACC"
return
    $product
```

- The ***where*** clause *will* be evaluated once for each iteration
  - Familiar for SQL users

- On some implementations it *can* be more efficient to use a **predicate** instead

```
doc("catalog.xml")/catalog/product[@dept eq "ACC"]
```

FLWOR – Ordering Results

```
for
    $product in doc("catalog.xml")/catalog/product
order by
    xs:date($product/@commenced) descending
return
    $product
```

- Ordering may be either ***ascending*** or ***descending***
- You can order on multiple values, e.g.

```
order by
    xs:date($product/@commenced) descending,
    xs:int($product/number) ascending
```

- **Hint:** When ordering, ensure the type of the value

Complete FLWOR Example

```
<products>{
    for
       $product at $i in doc("catalog.xml")/catalog/product
    let
       $catalog-age := days-from-duration(
              current-date() - xs:date($product/@commenced)
       )
    where
       $product/dept eq "ACC"
    order by
       xs:date($product/@commenced) descending,
       xs:int($product/number) ascending
    return
       <product idx="{$i}" ref="{$product/number}">
              <age>{$catalog-age}</age>
              <name>{$product/name/text()}</name>
       </product>
}</products>
```

Variable Scope

- XQuery is functional
  - It has immutable variables
  - Variables are "*bound*" to a value

- What result does the following yield?

```
let $y := 1 return
for $x in (1 to 100)
let $y := $y + $x
return $y
```

a) 1
b) 5050
c) 5051
d) The Sequence (1 to 100)
e) The Sequence (2 to 101)
f) A sequence ending in 5051

Variable Scope

- More comprehensible when rewritten to:

```
let $c := 1
return
    for $x in (1 to 100)
    let $y := $c + $x
    return
        $y
```

Answer) The Sequence (2 to 101)

Variable Scope

- Variable Bindings have a limited Scope

- Is this valid?

```
<something>
    <today>
    {
        let $now := current-date() return
            $now
    }
    </today>
    <tomorrow>
    {
        $now + xs:dayTimeDuration("P1D")
    }
    </tomorrow>
</something>
```

Variable Scope

- ...and is this valid?

```
<something>
    {
        let $now := current-date() return
            <today>$now</today>
            ,
            <tomorrow>
            {
                $now + xs:dayTimeDuration("P1D")
            }
            </tomorrow>
    }
</something>
```

summer school

XQuery Challenge 1

What you need to know...

- Server
  http://play.evolvedbinary.com:8080/exist/apps/eXide
- Collection
  /db/products
- Functions
  - xmldb:get-child-collections($collection-path)
  - xmldb:get-child-resources($collection-path)
  - https://www.w3.org/TR/xpath-functions-31/#quickcontents

Challenges - Part 1

- How many documents are in /db/products/examples, what does their structure look like?
- What sub-collections of /db/products exist?
- How many:
  - Tablets? How many reviews for Tablets?
  - Tablets and Laptops?
  - How many total reviews?

Challenges – Part 2

Please test first with the */db/products/examples* collection

- Which product(s) have the most reviews?
- Which product(s) have the highest average "Overall" score?
- Which year(s) have the most laptop reviews?

summer school

Introduction to XML Databases

Why XML Databases?

Why might you need an XML Database?

Why use an XML Database?

- Why a database, why not use a File System?
  - How to retrieve?
    - By file-path or some sort of lookup table?
    - i.e. Is a 'Directory' the same as a 'Collection'?
  - Where to keep metadata?
  - How to Query?
    - grep?
    - Integrate a search-engine (full-text), e.g. Solr / Elastic?
    - No direct XPath access!
  - How to Update?

What is an XML Database?

What is an XML Database?

"An XML database is a data persistence software system that allows data to be specified, and sometimes stored, in XML format.

These data can then be queried, transformed, exported and returned to a calling system. XML databases are a flavor of document-oriented databases which are in turn a category of NoSQL database (meaning Not (only) SQL)."

-- https://en.wikipedia.org/wiki/XML_database

What is an XML Database?

- More than just a filesystem!
- Unit of storage is the “Document”
- It ingests (and *may* return) XML documents
- Node aware, e.g. across and within document access
- CRUD operations on document(s)/node(s)
- Some form of query facility/language

What is an XML Database?

- Full-Text capabilities
- Indexes defined for document queries
- Often defines "*Collection*"s
- May support non-XML content
  - e.g. Key/Value, Tabular, JSON, Binary, Graph etc.
- Single or Multi-user: Client/Server and/or Embedded

Types of XML Database

- XML Enabled Database
  - Existing database product which added support for XML
  - Predominant Data Model and purpose is NOT XML
  - Heterogenous data models
    - Typically used when only small amounts of XML are involved

- Native XML Database (NXDB)
  - Designed for XML storage/retrieval/query from the start
  - Primary concern and data model is hierarchical (tree)
  - Highly optimised for XML storage and query
    - Typically used when the majority (or all) of the data is XML

- Polyglot Persistence - i.e. 'Use the Right Tool for the Job'

XML Enabled/Hybrid Database

- RDBMS approaches:
  - XML Stored in CLOB
  - XML Shredding into tables. e.g. Oracle XML Schema Table.
  - ISO XML Type for columns

– Good for small amounts of standalone XML

– Bad for complex queries across XML and Tables

– Commercial: Oracle RDBMS, IBM DB2, SQL Server
– Open Source: PostgreSQL

Example - XMLType and SQL

id	issn	short_name	vol	journal
1	0012-1606	Dev. Biol.	369	<journal> <name>Developmental Biology</name> <publisher>Elsevier</publisher> </journal>
2	8756-8233	Drugs Soc.	11	<journal> <name>Drugs and Society</name> <publisher>Taylor & Francis</publisher> </journal>

```
select id, vol, xmlquery('$j/name', passing journal as "j") as name
from journals
where
   xmlexists('$j[publisher="Elsevier"]', passing journal as "j")
```

id	vol	name
1	369	<name>Developmental Biology</name>

Native XML Databases

summer school

- Reasons not to use an RDBMS
  - XML is **NOT** "*just text*"! (varchar / BLOB / CLOB)
  - Shredding
    - Every set of children is a table. Many *many* tables!
    - Manual vs. Auto.
    - How to Query/Transform/Retrieve doc?
- Many RDBMS offer XML storage (e.g. XMLType)
  - Oracle shred's behind the scenes, requires XML Schema.
  - Querying is often still driven from SQL
  - Joining XML and non-XML data is hard
- How to Update? Full-text Search? Aggregate?

summer school

Choosing an XML Database

Native XML Database Options 1/2

- BaseX
  - Open Source. BSD License
  - XQuery 3.1*, XSLT 2.0, XQuery Update 1.0, RESTXQ, EXPath, XQuery Full-Text 1.0
- eXist
  - Open Source. LGPL v2.1
  - XQuery 3.1*, XSLT 2.0, XQuery Update, RESTXQ, EXPath, Bespoke Full-Text, XProc, XForms 1.1, Customisable Extension Modules
  - Master-Slave Replication with Slave promotion.
- Marklogic
  - Commercial
  - XQuery 1.0/3.0*, XSLT 2.0, Bespoke Update, Bespoke Full-Text, XForms 1.1
  - Shared-Nothing Clustering
- Others: Sedna / EMC Documentum xDB / Zorba / etc...

Native XML Database Options 2/2

- BaseX
  - Serializable
  - Auto (short) locking of database for multiple readers / single-writer
  - System Transactions
  - Manual locking prolog options - *query:read-lock* / *query:write-lock*

- eXist
  - Dirty Reads
  - Auto (short) locking of resources for multiple readers / single writer
  - No PUL for XQuery Update
  - System Transactions for Write Ahead Journal
  - Manual locking functions – *util:shared-lock* / *util:exclusive-lock*

- MarkLogic
  - MVCC – Snapshot Isolation
  - Auto or User controllable Transactions
  - Manual locking functions - *xdmp:lock-acquire* / *xdmp:lock-for-update*

summer school

Getting started with eXist-db

How to get setup?

- eXist-db is written in Java
  - You need Oracle/Open JRE 8
- Download and install v3.4.1 from
  - http://exist-db.org/#download
  - Source code: https://github.com/exist-db/exist
- Consists of:
  - Database and Web Server
  - Simple GUI Admin Client
  - Web IDE (eXide) and Dashboard

Collections in eXist-db

- Documents are stored in Collections
- Root collection is **/db**
- Collections can contain sub-collections
- The collection hierarchy is inherited!

Quiz

How do I get all of the marketing collection?

What does **collection**("/db/journals") return?

What does **collection**("/db/books/blogs") return?

summer school

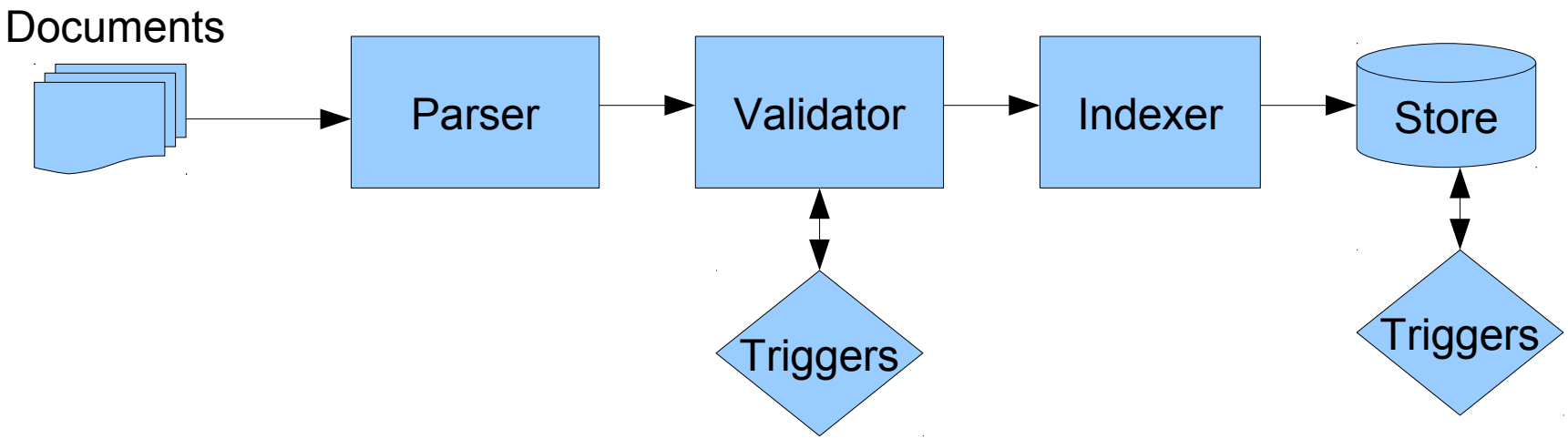
Store/Retrieve with an XML Database (eXist-db)

How to store data into a NXDB?

- Storing an XML Document into eXist-db
  - Upload to the database via API
  - For Web Developers
    - Demo REST
  - For Authors/Editors
    - Demo WebDAV
  - For Programmers
    - Demo Java
    - Demo Python
  - Many other options available...

How to store data into a NXDB?

- Where has my XML Document gone?



  - Demo – webapp/WEB-INF/data
  - Highly Optimised storage format and indexes

How to retrieve data from a NXDB?

- Retrieving an XML Document from eXist-db
  - Download from the database via API
  - **Canonical form!**
  - For Web Developers
    - Demo REST
  - For Authors/Editors
    - Demo WebDAV
  - For Programmers
    - Demo Java
  - Many other options available...

Basic Database Queries

- REST API
  - HTTP GET

```
http://localhost:8080/exist/rest/db/?
_query=<date>{current-dateTime()}</date>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<exist:result xmlns:exist="http://exist.sourceforge.net/NS/exist" exist:hits="1"
exist:start="1" exist:count="1">
   <date>2012-09-07T15:44:23.275+01:00</date>
</exist:result>
```

  - HTTP POST

```
http://localhost:8080/exist/rest/db/
```

```
<query xmlns="http://exist.sourceforge.net/NS/exist">
   <text><![CDATA[
         <date>{current-date()}</date>
      ]]></text>
</query>
```

Querying the Database

- Demo - GUI Admin Client

- Demo - eXide
- RESTXQ, SOAP / WebDAV / XML-RPC / Java / PHP / Python, etc.
- Stored Queries
  - XQuerys can be stored into the database
  - Executed later e.g. REST Server by URI
  - Demo

summer school

Advanced XQuery

Advanced XQuery

- XQuery is a full functional Programming Language
- Aspects of XQuery we will examine
  - Conditional Expressions
  - Functions

Conditional Expressions

- If, then, else syntax:

```
if($date lt current-date())then
    <result>The date {$date} is in the past</result>
else if($date gt current-date())then
    <result>The date {$date} is in the future</result>
else
    <result>The date {$date} is today!</result>
```

- Parentheses must surround the expression for **if**
- **if** expressions can be chained, i.e. **else if**
- **else** is always required
  - Can just use the empty sequence i.e. **else** ()

Effective Boolean Value

- Expression of **`if`** statement must be boolean
  - if not, its *effective boolean value* is found
- effective boolean value is *false* for:
  - the **`xs:boolean`** value **`false`**
  - the number **`0`** or **`NaN`**
  - the empty sequence
  - ...also, a zero-length string!
- otherwise it is *true* (e.g. a list of elements)

```
if(doc("order.xml")//item)then
    <result>Found some items</result>
else
    <result>Error: Zero items in the order</result>
```

Logical Expressions

- Combine boolean values: **and**, **or**
  - **and** has precedence over **or**
  - use parentheses to manage precedence

```
if($is-discounted and ($discount gt 10 or $discount lt 0))then
    10
else
    $discount
```

- Use **not** function to invert boolean value

```
if(not($is-discounted))then
    0
else
    $discount
```

- **not** function will also resolve *effective* boolean value

Type Conditional Expressions

- typeswitch syntax:

```
typeswitch($something)

    case $n as element(name) return
        <result>found the name: {$n/text()}</result>

    case $e as element() return
        <result>found an element: {local-name($e)}</result>

    case $t as text() return
        <result>found the text: {$t}</result>

    case $i as xs:integer return
        <result>found the integer: {$i}</result>

    default return
        ()
```

Type Conditional Expressions

- Identity Transform using typeswitch:

```
declare function local:transform($node) {
    for $n in $node return
        typeswitch($n)

            (: TODO add you overrides here:)

            case document-node() return
                document {
                    local:transform($n/*)
                }

            case element() return
                element {node-name($n)} {
                    local:transform($n/@*|$n/node())
                }

            default return
                $n
};
```

Functions

- XQuery has many built-in functions
  - Defined in W3C Spec:
    - XQuery 1.0 and XPath 2.0 Functions and Operators
      http://www.w3.org/TR/xpath-functions/
  - So far we have seen:

    doc collection distinct-values sum current-date days-from-duration not local-name node-name

  - There are >150 functions available in XQuery 1.0
  - There are >220 functions available in XQuery 3.0
  - There are >260 functions available in XQuery 3.1

Extension Functions

- Vendors/Processors may also provide extension functions
- e.g. SQL Queries, Sending Email, DB Management
  - eXist-db provides > 600 functions!
- Extension functions are processor specific
  - ...Non-Portable XQuery code
  - EXPath and EXQuery Projects try to standardise

http://www.expath.org
http://www.exquery.org

User Defined Functions

- You can also write your own functions in XQuery
- Functions must have a fully qualified name
  - e.g. **my:function1**
  - *The "local" prefix may be used for functions in a main module. e.g.* **local:function1**
- Functions may be placed in library modules
  - Which are imported by other modules or main module
- Many common functions available at FunctX

  http://www.xqueryfunctions.com/

User Defined Functions

- Function Declaration:

```
declare function local:my-first-function() {

    (: TODO your function body code goes here! :)

};
```

  - Functions cannot have an empty body (i.e. above!)

- Functions may take parameters

```
declare function local:my-first-function($thing, $other) {
    <this>{$thing}</this>
};
```

- Parameters may be explicitly typed

```
declare function local:my-second-function($when as xs:time) {
    ...
```

User Defined Functions

- Parameters may be sequences with constraints
  - Cardinality
    - One

```
declare function local:my-third-function($date as xs:date)
```

    - Zero or One **?**

```
declare function local:my-third-function($date as xs:date?)
```

    - One or More **+**

```
declare function local:my-third-function($dates as xs:date+)
```

    - Zero or More *****

```
declare function local:my-third-function($dates as xs:date*)
```

User Defined Functions

- Return values may be explicitly typed

```
declare function local:my-third-function($date as xs:date)
 as element(calendar) {

   <calendar>Starting from: {$date}</calendar>
};
```

- Return values may specify a cardinality constraint

```
declare function local:my-third-function($dates as xs:date)
 as element(calendar)+ {

    for $date in $dates return
        <calendar>Starting from: {$date}</calendar>
};
```

Function Modules

- XQuery code lives in Modules
  - Two types of module:
    - Main Module
      - Everything you have seen up to now!
      - Query Body (and maybe functions)
      - Can Import other Libraries
      - .xqy file, or...
    - Library Module
      - Has a namespace!
      - Just functions
      - Can Import other Libraries
      - .xqm file, or...

Function Modules

- Main Module

```
xquery version "1.0";

import module namespace my = "http://my-function-module"
 at "my-funcs.xqm";

<greetings>{
    my:say-hello("Adam Retter")
}</greetings>
```

- Library Module

```
xquery version "1.0";

module namespace my = "http://my-function-module";

declare function my:say-hello($name) {
    <hello>{$name}</hello>
};
```

summer school

XQuery Challenge 2

What you need to know...

- Server
  http://play.evolvedbinary.com:8080/exist/apps/eXide

- Data Collection: /db/pubmed/data
  - Examine: /PubmedArticleSet/PubmedArticle[1]

- Create your own App Collection
  /db/<***your-app-name***>

- Functions
  - xmldb:collection-available("/db/***your-app-name***")
  - xmldb:create-collection("/db", "***your-app-name***")

Challenges - Part 1

summer school

- *Functions*
  - *https://www.w3.org/TR/xpath-functions-31/#quickcontents*
  - *http://exist-db.org/exist/apps/fundocs/index.html*
  - *HINT: request:get-parameter($param-name, $default-val)*

- Create and store a HTML form (form1.html) into your Collection
  - Form should take some sort of query, e.g. date, name or keyword
  - http://play.evolvedbinary.com:8080/exist/rest/db/***your-app-name***/form1.html

- *Create an XQuery that processes the form, queries the /db/pubmed/data collection, and shows the result as XML*

Challenges - Part 2

- *Functions*
  - *https://www.w3.org/TR/xpath-functions-31/#quickcontents*
  - *http://exist-db.org/exist/apps/fundocs/index.html*
  - *HINT: transform:transform($xml-node, $xslt-node, ())*
  - *HINT:* fn:serialize($xml-node, $serialization-parameters)

- Modify your query from part 1 to optionally return HTML (use XSLT for the transform)

- *Modify your query again, to optionally return JSON*
  - *<output:serialization-parameters><output:method value="json"/></output:serialization-parameters>*

Challenges - Part 3

- *Functions*
  - *http://exist-db.org/exist/apps/doc/lucene.xml#D2.2.5*
  - *ft:query($xml-node, $query-expression)*
- Examine the Full Text index definitions in: /db/system/conf/db/pubmed/data/collection.xconf
- *Modify your HTML form and XQuery again, to also offer a full-text query option*