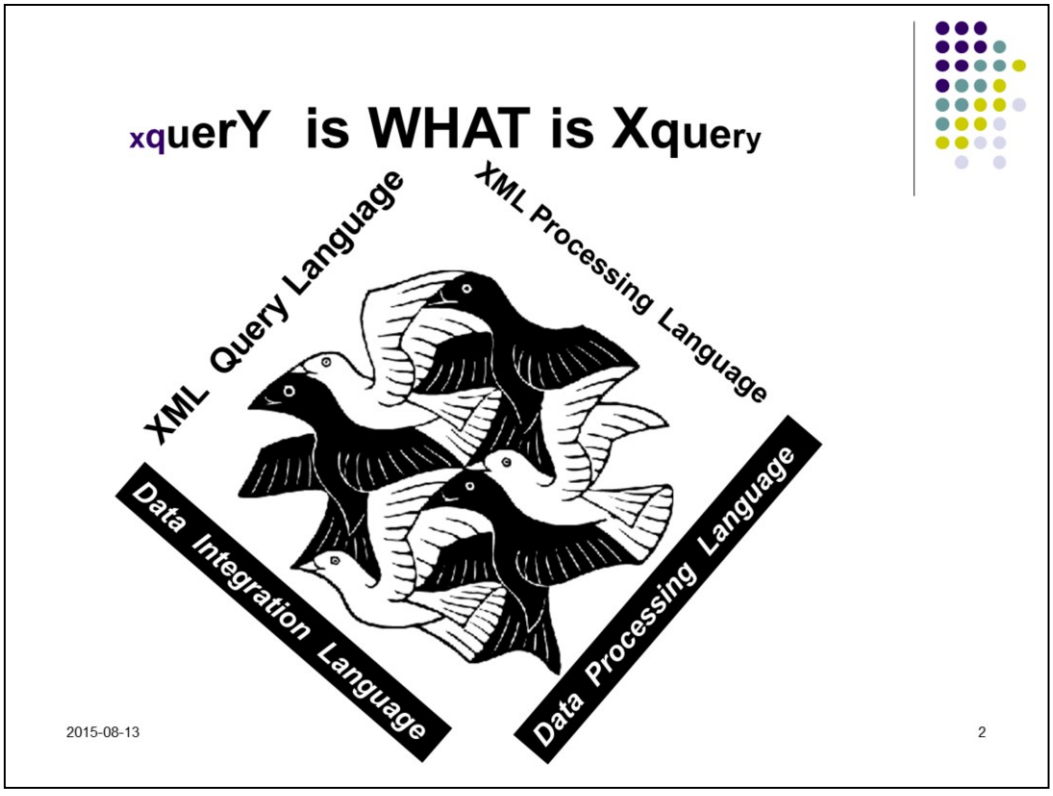
XQuery as a *data integration language*

Assessing the potential.

Hans-Jürgen Rennau, Traveltainment GmbH
Christian Grün, BaseX GmbH
Presented at Balisage 2015, August 13, 2015

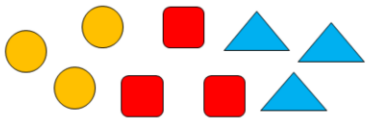
In spite of its name, XQuery is certainly more than a query language. But what is it? Recent extensions of the language, giving access to non-XML resources, viewed in conjunction with the typical strengths of the language, seem to open a new perspective: XQuery as a data integration language. Let us explore.



Well, the language is a set of facts, provided by the language specification, which is clear and precise. But the facts themselves are not yet the picture we see. The picture results from how we view the facts, how we restructure and connect them, and last not least - our imagination. Here, you might see white doves, or black ones, or even both. Similary, when looking at XQuery, it is possible to see very different things.

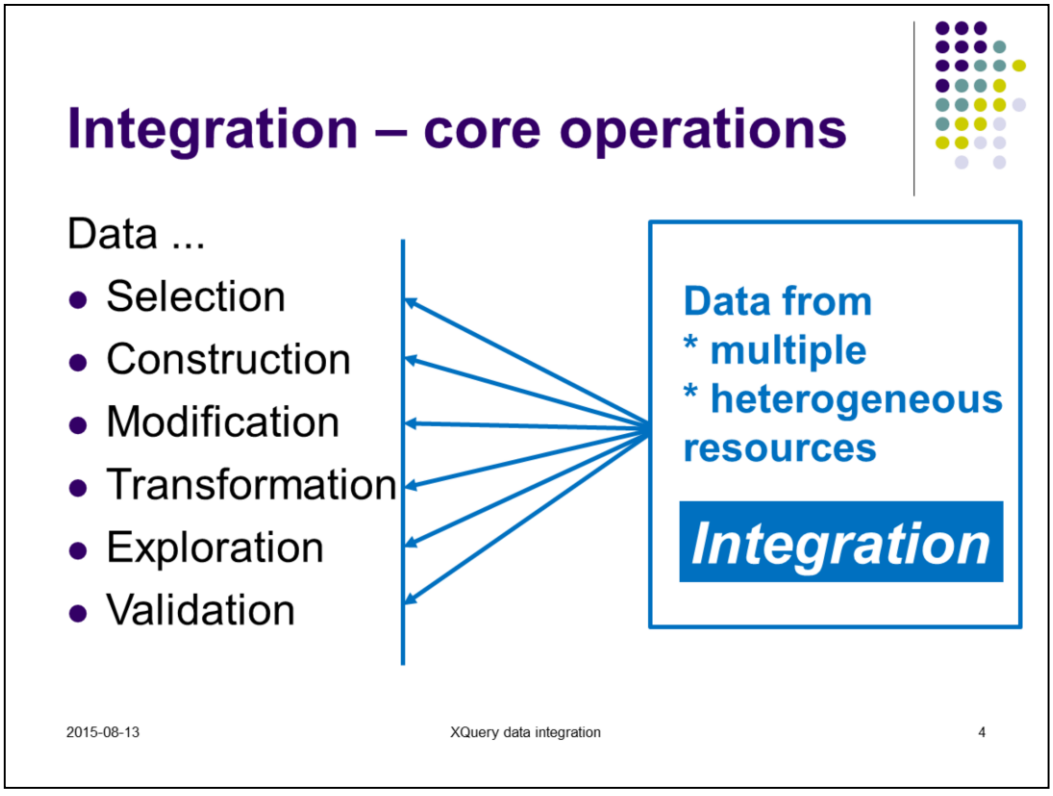
Data integration language ?

- Integration – facing
  multiplicity and heterogeneity



- Data integration language:
  making integration *simpler*

Data integration means dealing with information distributed over several, usually heterogeneous resouces. Any general purpose programming language has APIs for accessing different types of resources - so every general purpose language can be used for data integration. Then why might a language be called a data integration language? The key aspect is simplicity. A data integration language enables simple solutions for typical operations of data integration.



Some typical operations required during data integration. A data integration language should support them well especially in a context of multiple, heterogeneous resources.

Divide and conquer

How to assess the integration capabilities?

Investigation strategy

- XML integration capabilities
  (capabilities in an XML-only environment)
- Access to non-XML resources
- XML integration capabilities applicable to non-XML?

We shall first investigate the integration capabilities of XQuery in a pure XML environment. Then we evaluate XQuery access to non-XML resources. Finally we ask whether the integration capabilities observed in an XML-only environment can be applied to these non-XML resources as well.

Data navigation

Precision
Expressiveness
Naturalness

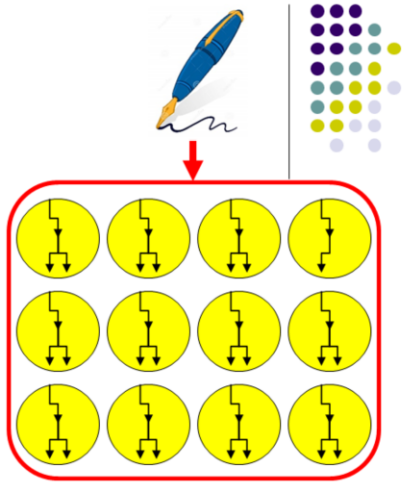
```
//@flightNumber
```

```
//flights
/flight
/@flightNumber
```

```
//flights
/flight [arrival  /@airport = 'IAD']
        [departure/@airport = doc('/usr/data/airports.xml')
                              //airport[@cty!='US']/@code]
/@flightNumber
```


Any understanding of XQuery and its potential uses is based on awareness of XQuery‘s concept of data navigation – XPath.

Bulk navigation

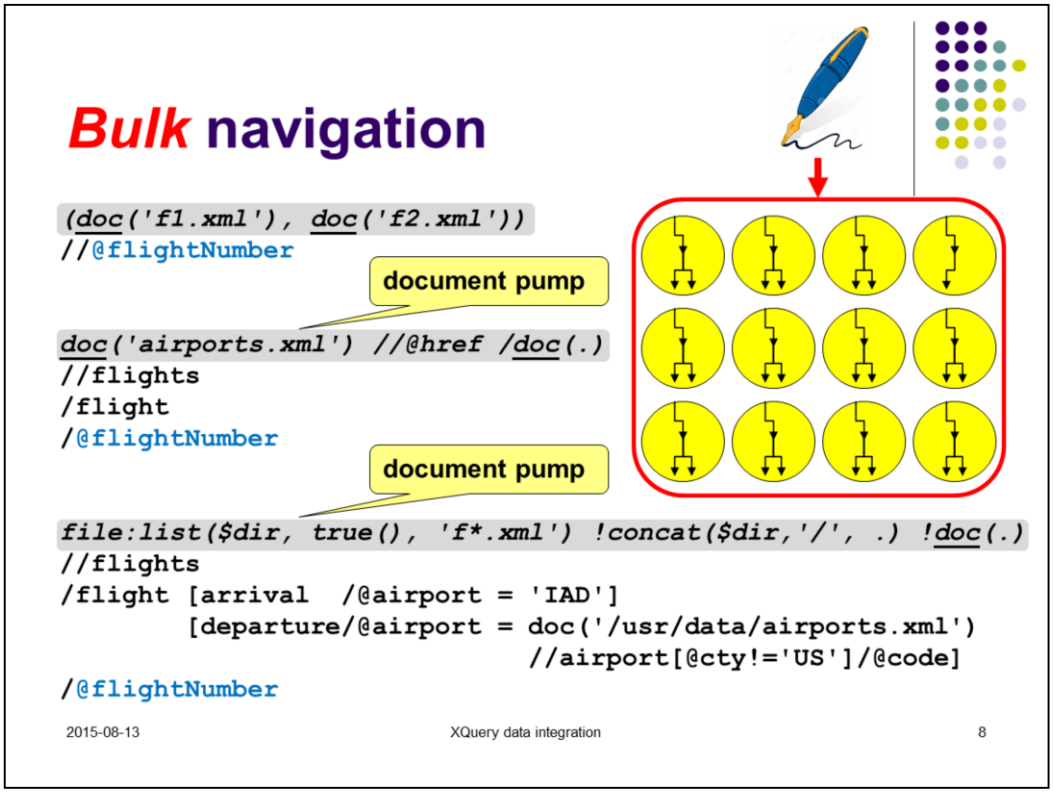


```
//@flightNumber
```

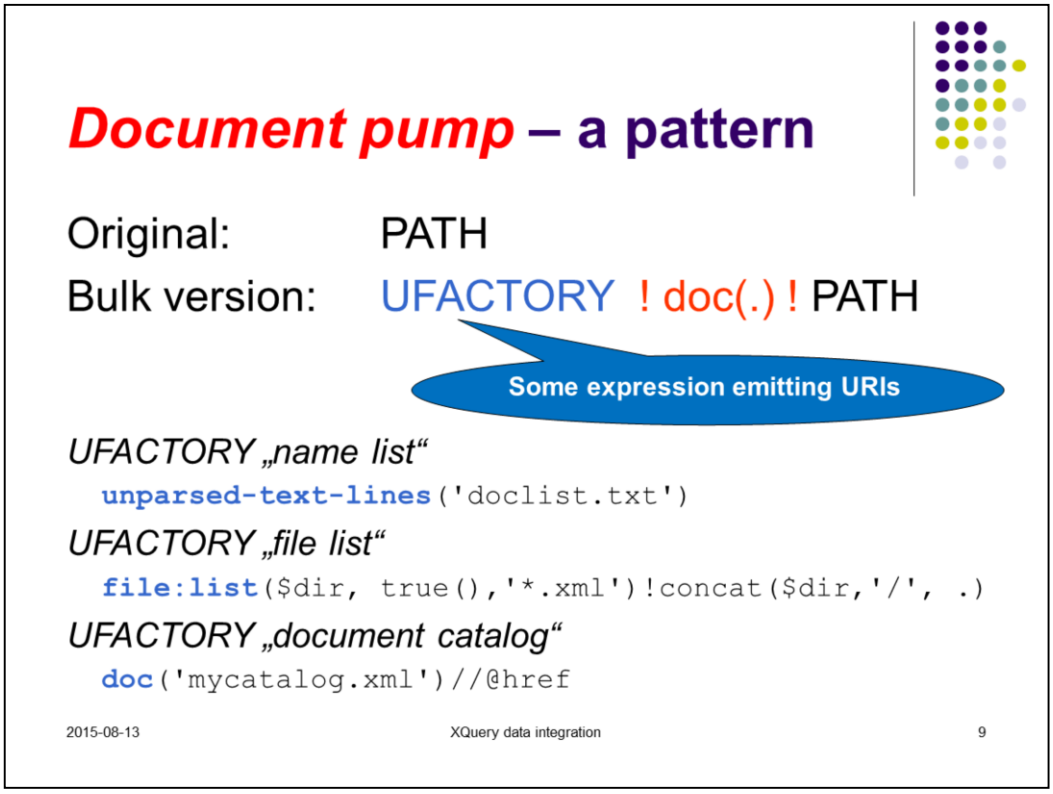
```
//flights
/flight
/@flightNumber
```

```
//flights
/flight [arrival   /@airport = 'IAD']
        [departure/@airport = doc('/usr/data/airports.xml')
                              //airport[@cty!='US']/@code]
/@flightNumber
```


Data integration is not dealing with a single resource, but many resources. Hence our interest in bulk navigation – navigation applied to many resources simultaneously, producing an integrated result.



And here we encounter a first fact which might have an impact on our picture of XQuery. The transition from single document navigation to bulk navigation does not add any complexity to the code. You only have to add an initial step to your path expression, producing the documents to be navigated. This extended path applies the original navigation to all those documents and delivers a merged result.

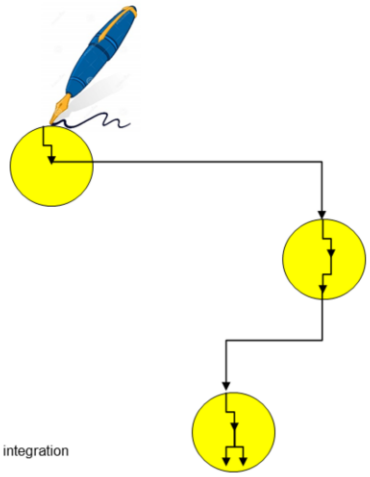


So here we have a simple design pattern how to transform single document navigation into bulk navigation: prefix the path by an expression producing URIs, which are piped into document construction, whose result is then piped into the proper navigation.

Cross-document navigation

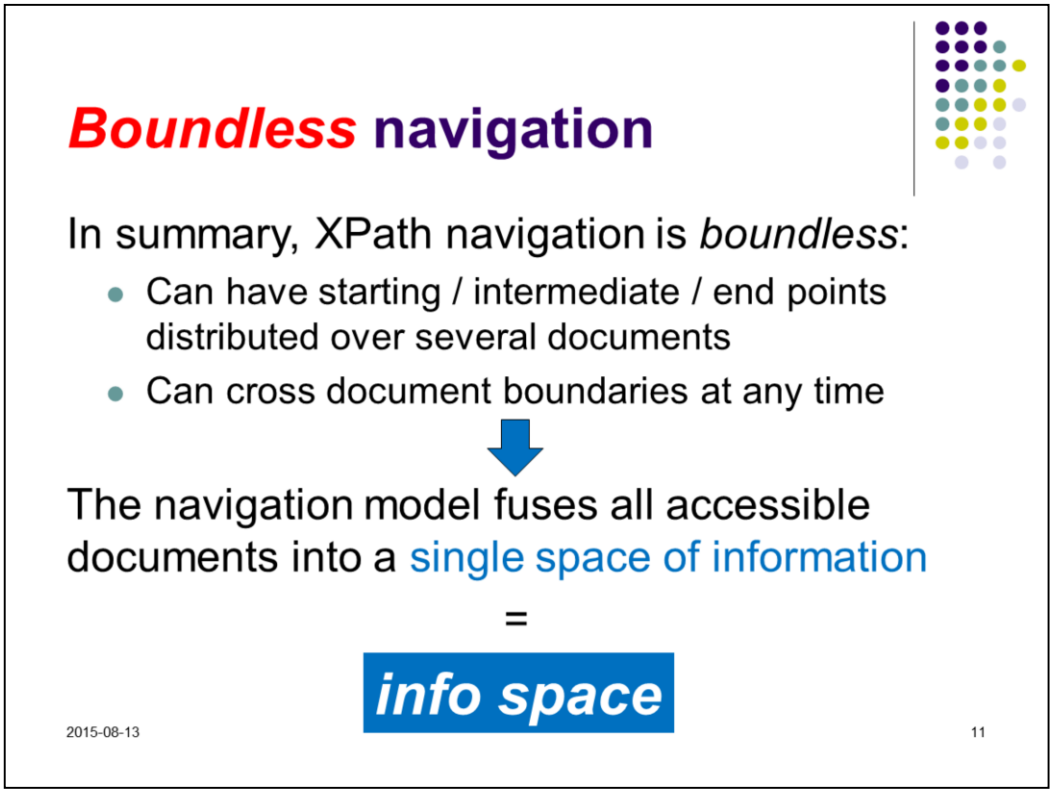
Navigation can cross document boundaries any number of times.

```
//flights
/flight
/termsAndConditions
/@URL
/doc(.)
//payments
/ccProvider
/@href
/doc(.)
//contact
```

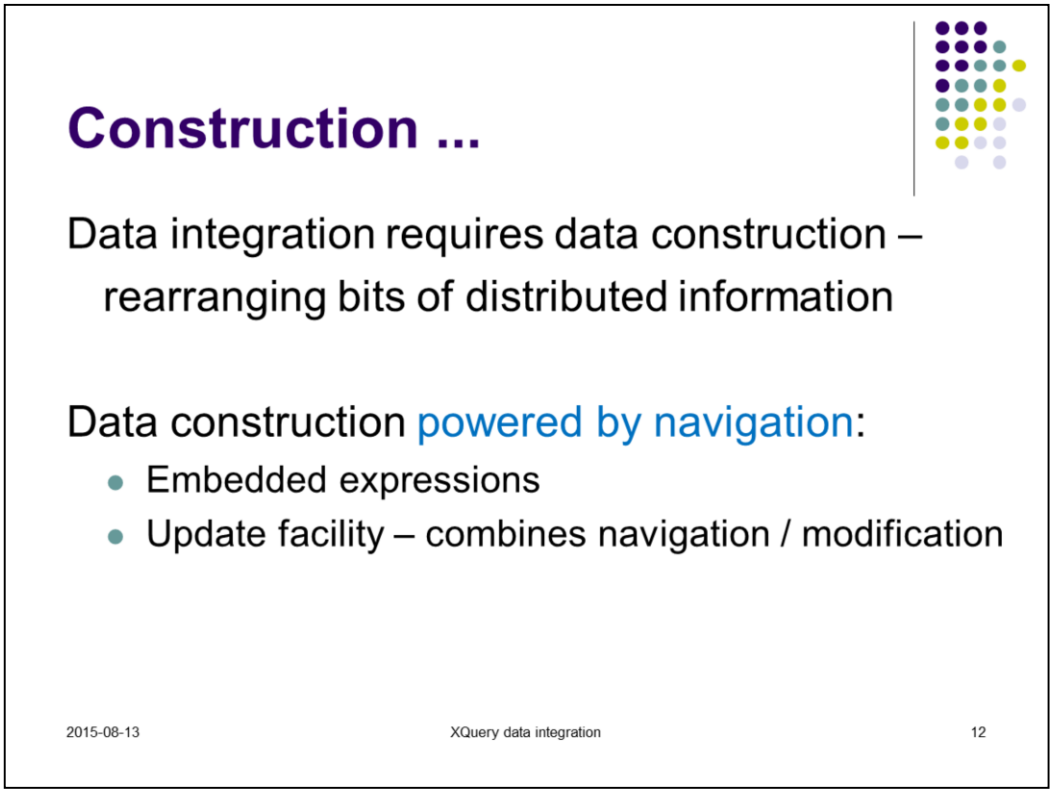


2015-08-13 XQuery data integration 10

Another fact deserving attention when thinking of data integration is cross-document navigation: link traversal. A path step reaches items containing document URIs, the subsequent step leaps to the roots of the referenced documents, whence the navigation continues, drilling down into those documents.

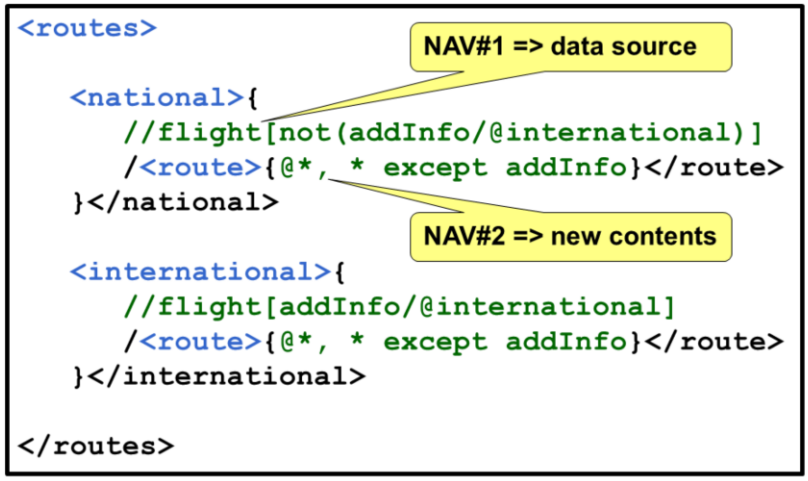


Joining these insights into a single whole – the precision and simplicity, the bulk friendliness, the irrelevance of document boundaries – a picture emerges which shows XPath navigation as boundess. And the space of this navigation – the sum total of accessible XML documents – is in turn fused by XPath into a single, homogeneous substrate, which I like to call the info space. By definition, this space is an ideal stage for data integration.



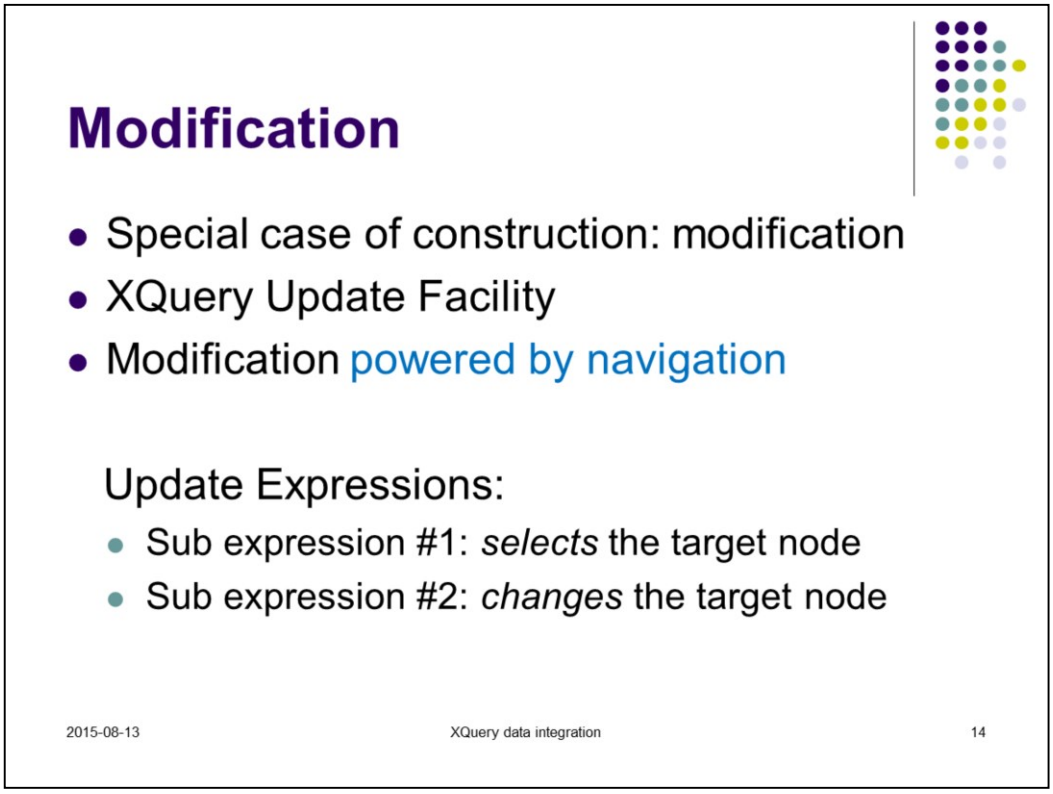
The immediate effect of data navigation is data selection. But data integration needs more than data selection – distributed information must not only be selected, but reassembled into new entities. Construction is as essntial a part of integration as selection.

... powered by navigation

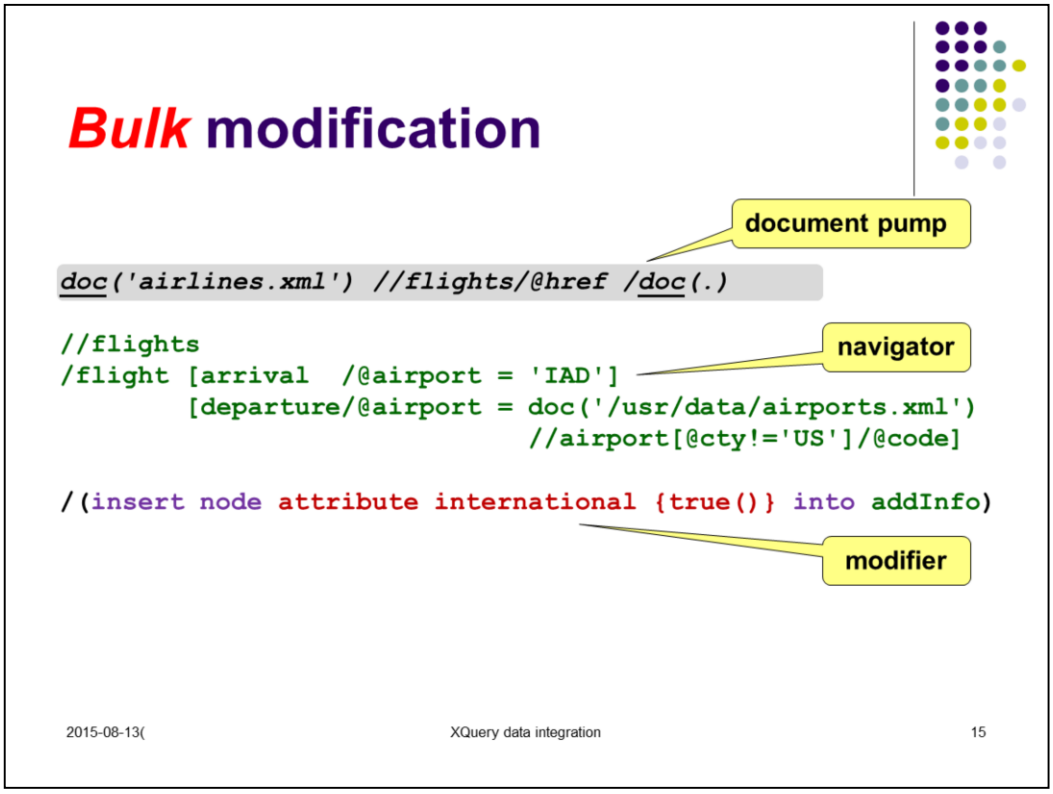



```
<routes>

   <national>{
      //flight[not(addInfo/@international)]
      /<route>{@*, * except addInfo}</route>
   }</national>

   <international>{
      //flight[addInfo/@international]
      /<route>{@*, * except addInfo}</route>
   }</international>

</routes>
```

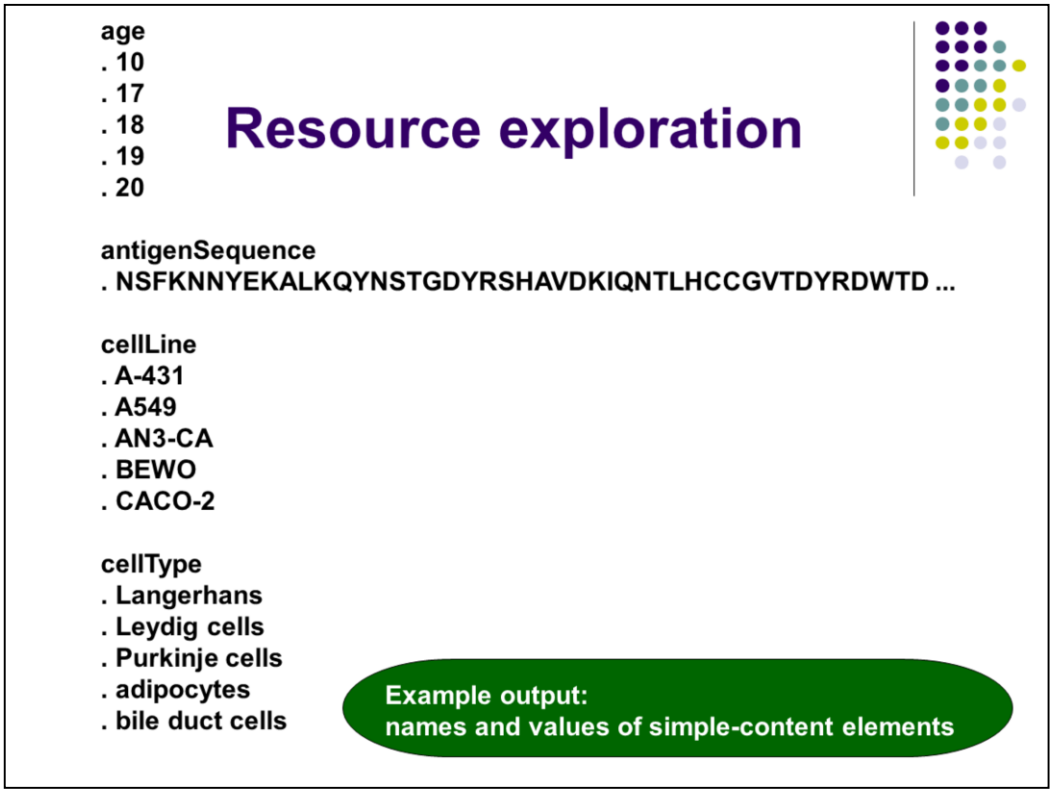

In this example, different navigations are used to select data sources (flight elements) and new element contents, which are a filtered copy of the data source contents. The effort of construction is minimal, thanks to navigation.



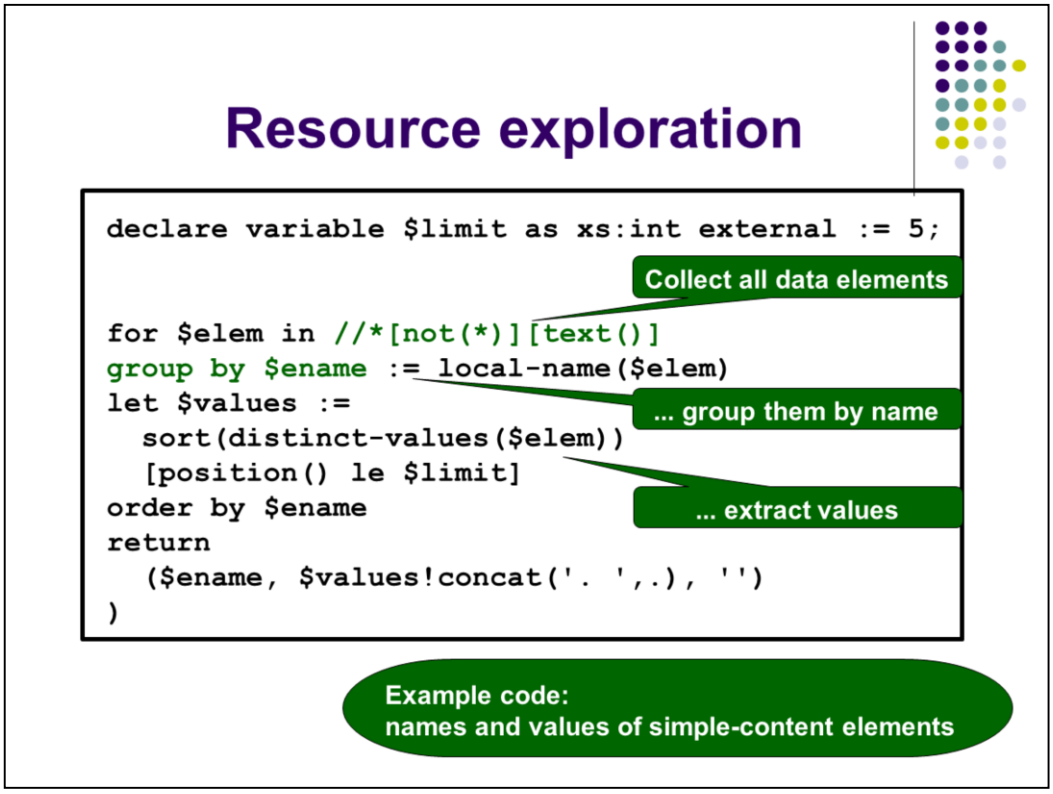
It is worthwhile to consider a special case of construction – modification. The XQuery Update Facility enables an elegant combination of navigation and change: an independent sub expression selects the target node, another expression provides the new value to be inserted, which more often than not is in turn expressed by some navigation - for example into a configuration file.



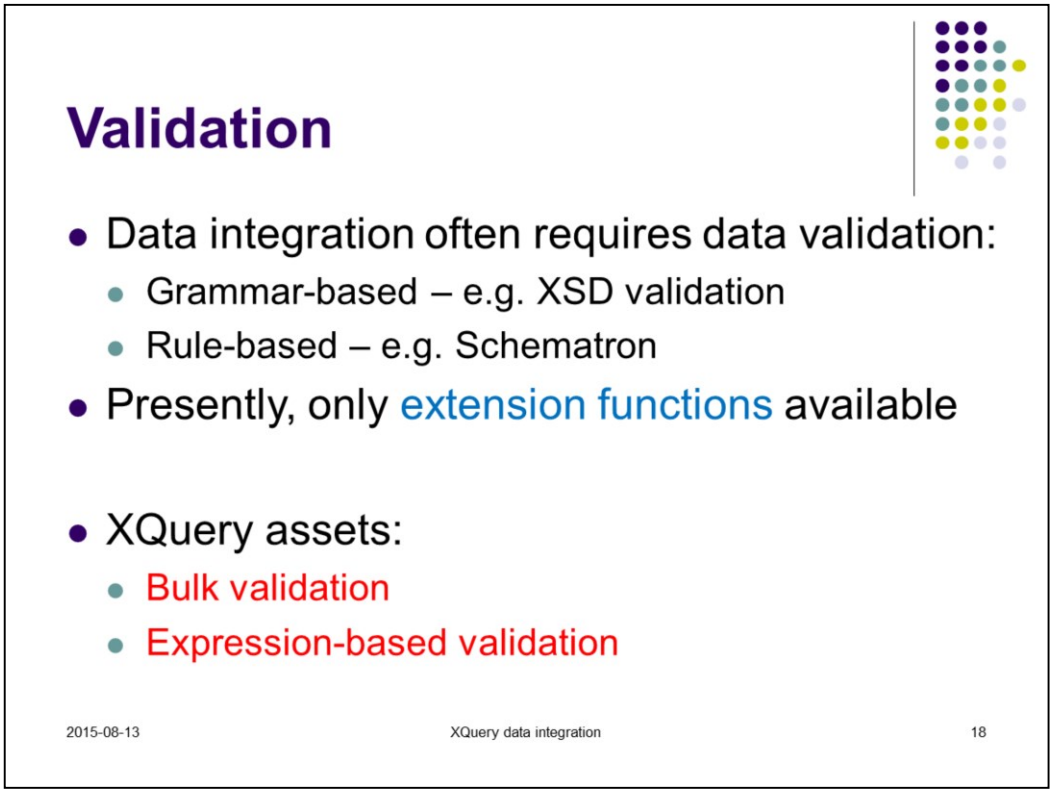
Let us look at an example of bulk modification. The first line pumps documents into the navigation to flight elements which describe a flight from a non-US airport to Washington. The last line modifies the contents of these flight elements, inserting an „international“ attribute into its addInfo child element.



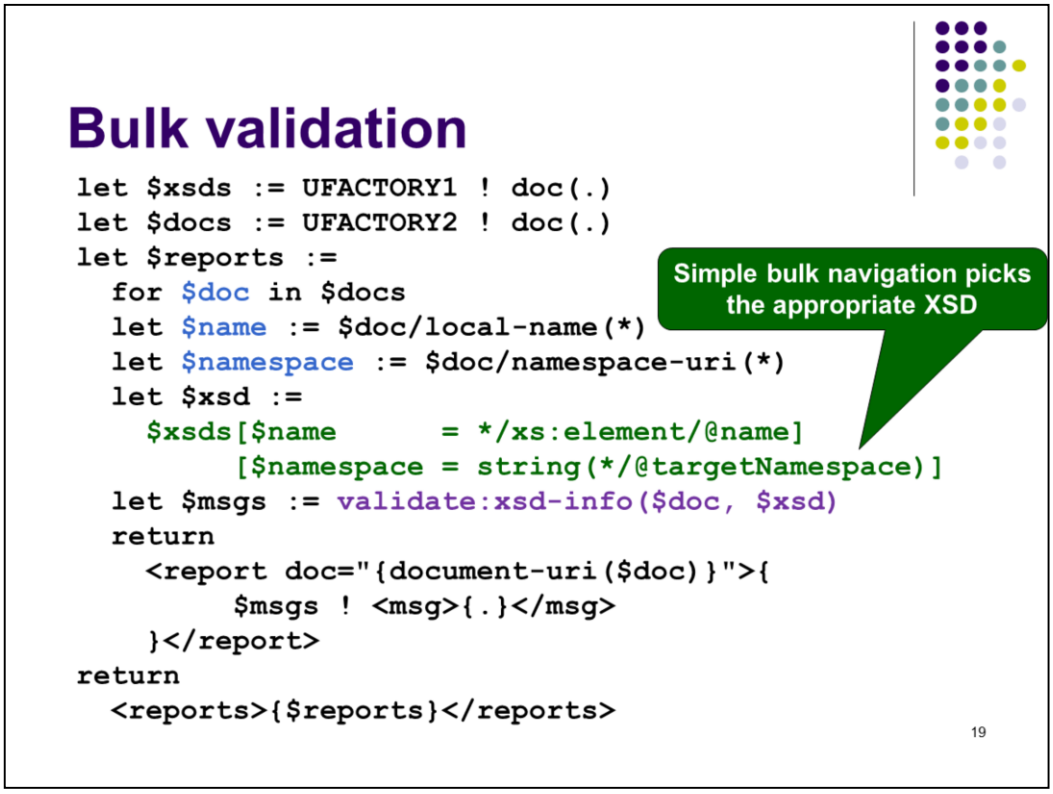
Resource exploration is an important activity when attempting data integration. It is about gaining insight about resource contents. XQuery is a great tool for such tasks. For example, when confronted with an unknown type of XML documents, you may want to learn the names and sampled contents of its data elements.



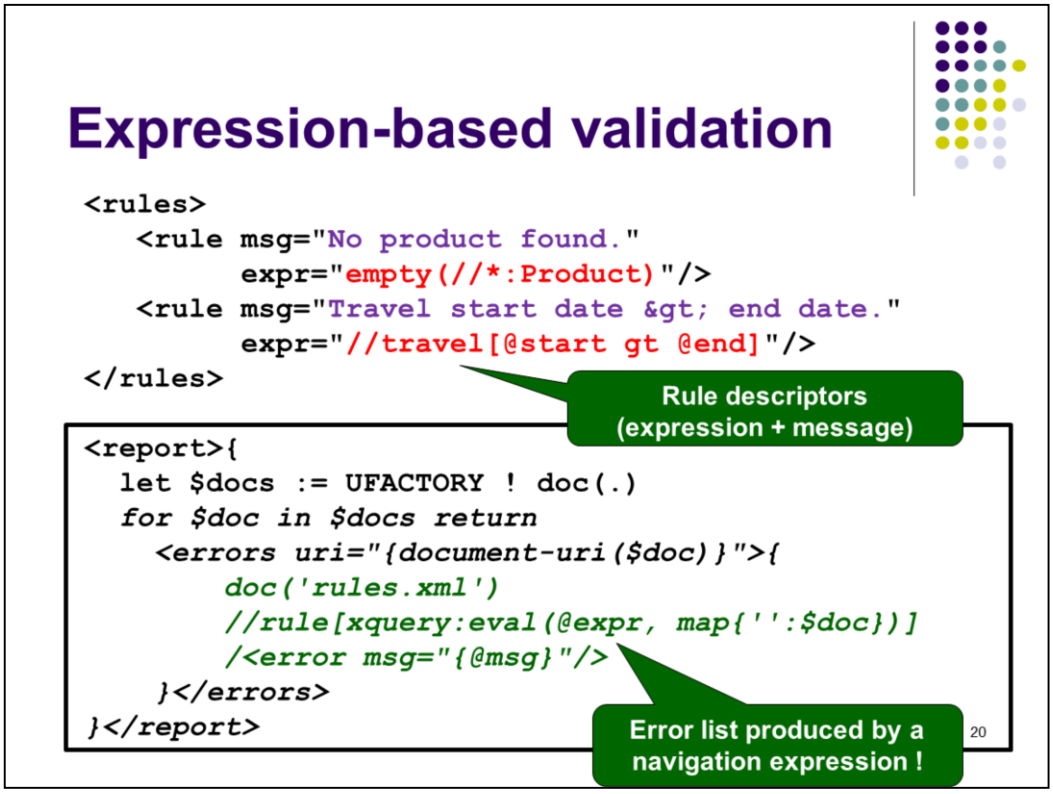
A few lines of XQuery code give you the answer. It goes without saying that transition to bulk exploration is just a couple of code lines away.



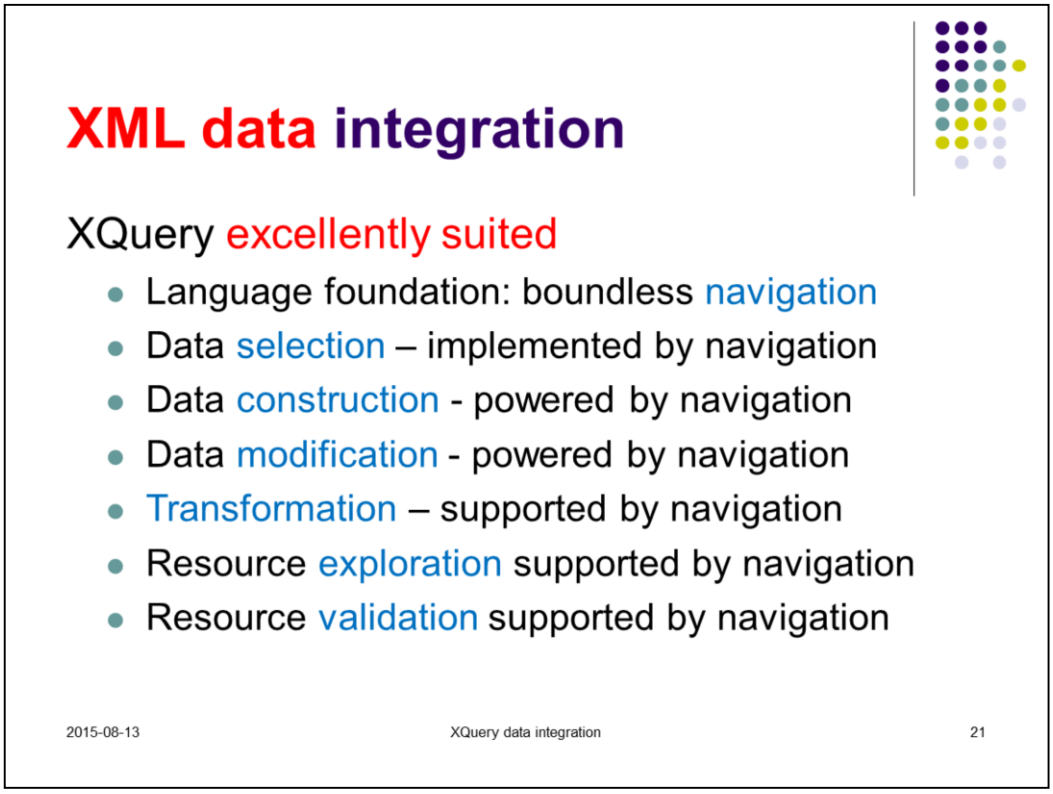
Validation is important, and it is very important in the context of data integration. It is a pity that there are not yet standard functions available providing XSD and other validation. Fortunately, popular XQuery processors offer extension functions for validation. What XQuery has to offer is excellent support for bulk validation and expression-based validation.



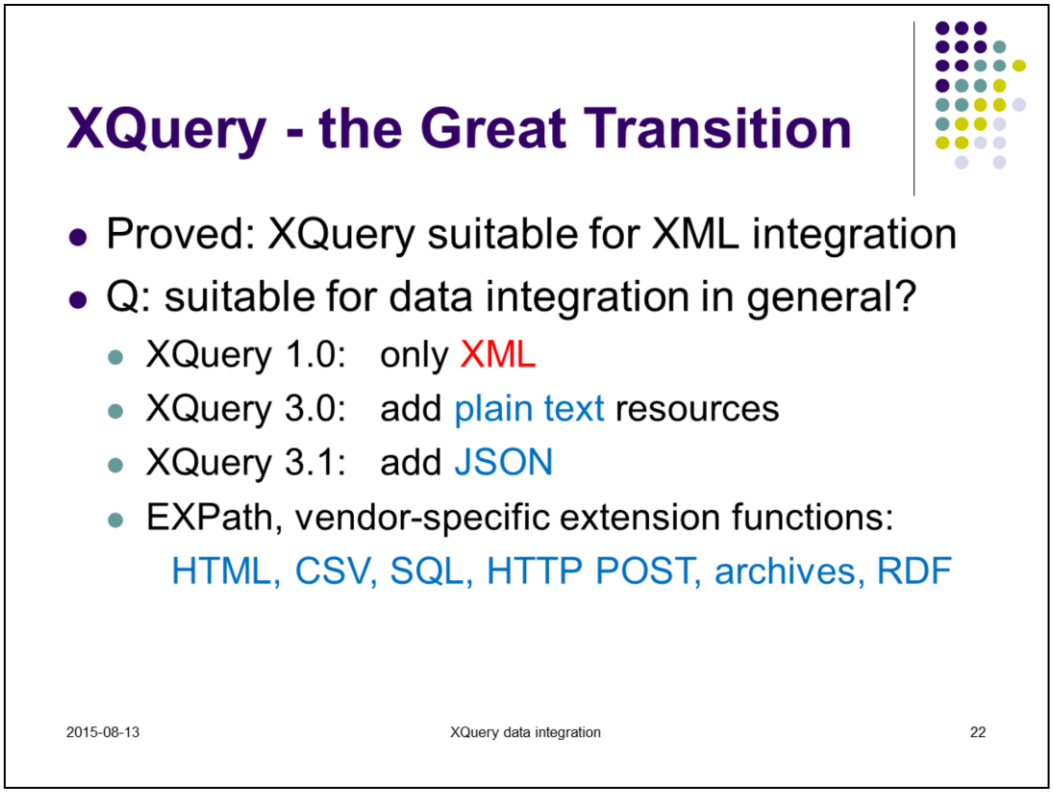
Interesting in the context of data integration is bulk validation: validating a heterogeneous set of instance documents against a collection of XSDs, letting the bulk validator sort out for each instance document which schema to pick. This is a trivial task of bulk navigation – inspect the name and namespace of the instance document root, and pick the schema with matching target namespace and containing a top-level element declaration with a matching name.



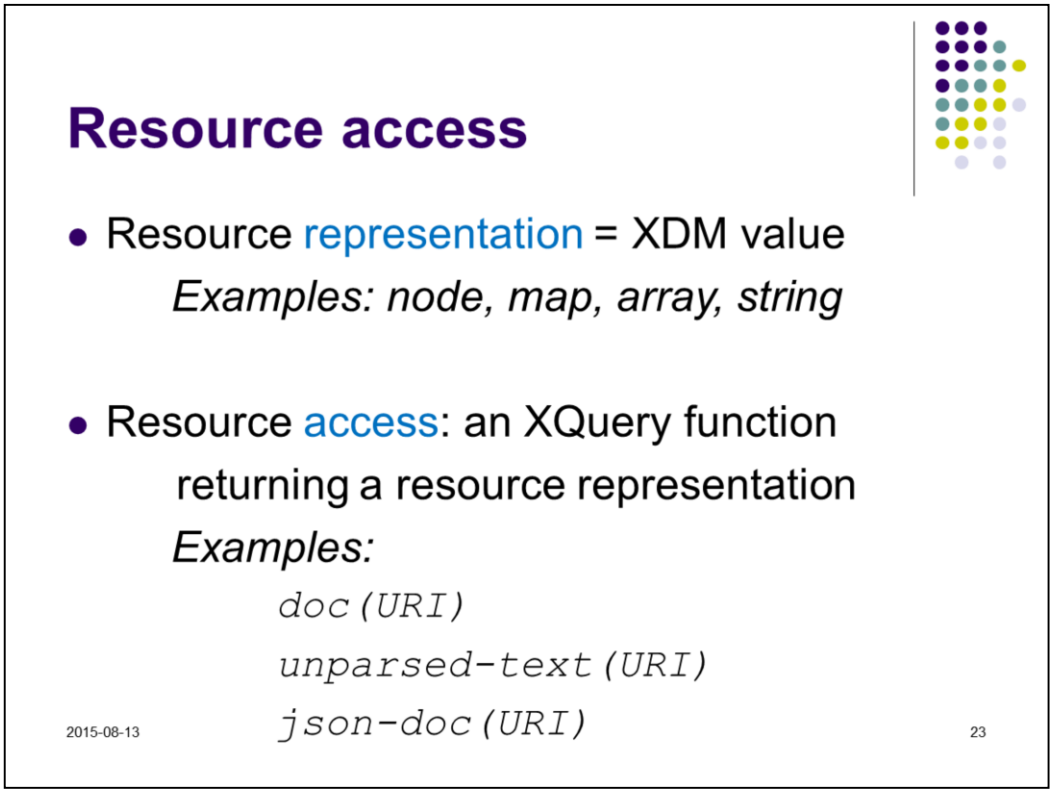
Grammar-based validation cannot check business rules. XQuery espressions are an ideal way of both defining and checking a rule. This example shows a simple generic rule checker. The validation is essentially executed by navigation: visit the rule descriptors, filter them by applying their expression to the target document, and emit the associated message. Note that this code uses a vendor-defined extension function for the dynamic evaluation of XQuery expressions found in the configuration.



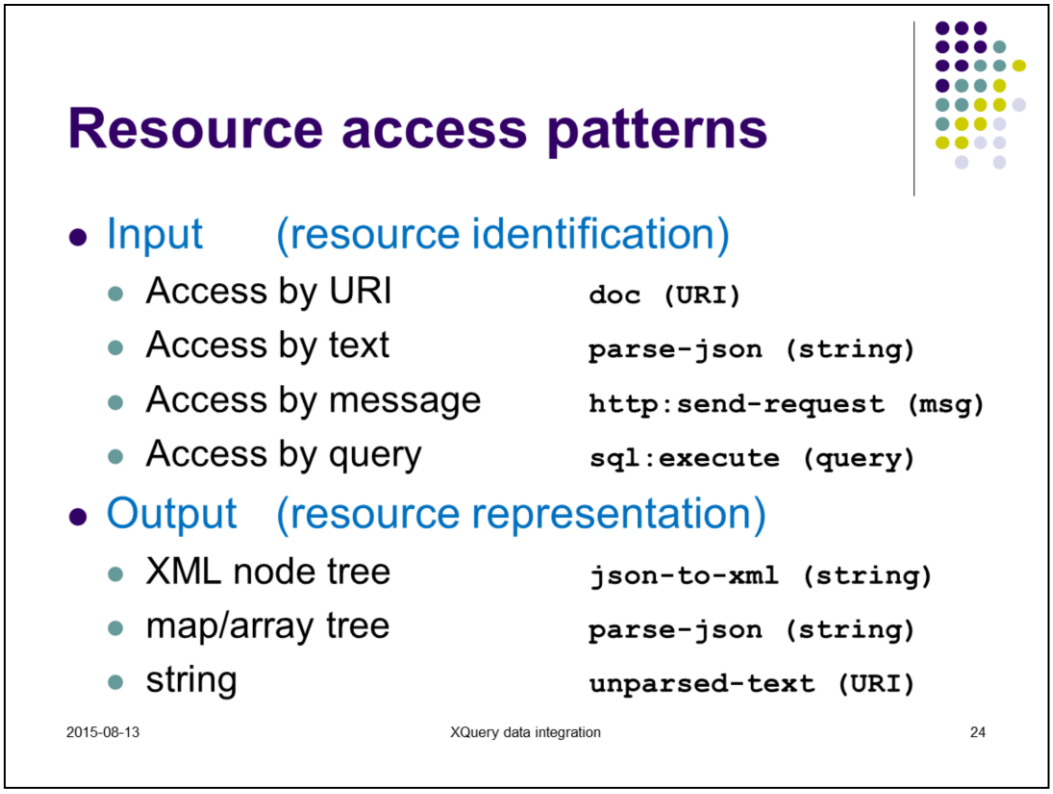
Ok – dealing with an XML-only environment, XQuery is excellently suited for data integration – thanks to a language structure laying a groundwork of boundless data navigation and using it to power construction and modification. Navigation also provides the programmer with strong support for transformation, exploration and validation.



Take a deep breath – our first result is that XQuery is a great tool for XML integration. Recent versions of the language (3.0, 3.1) added access to non-XML resources – plain text and JSON. And in the mean time popular XQuery processors already provided access to numerous other resource types. So let us ask: is XQuery suitable for data integration in general?

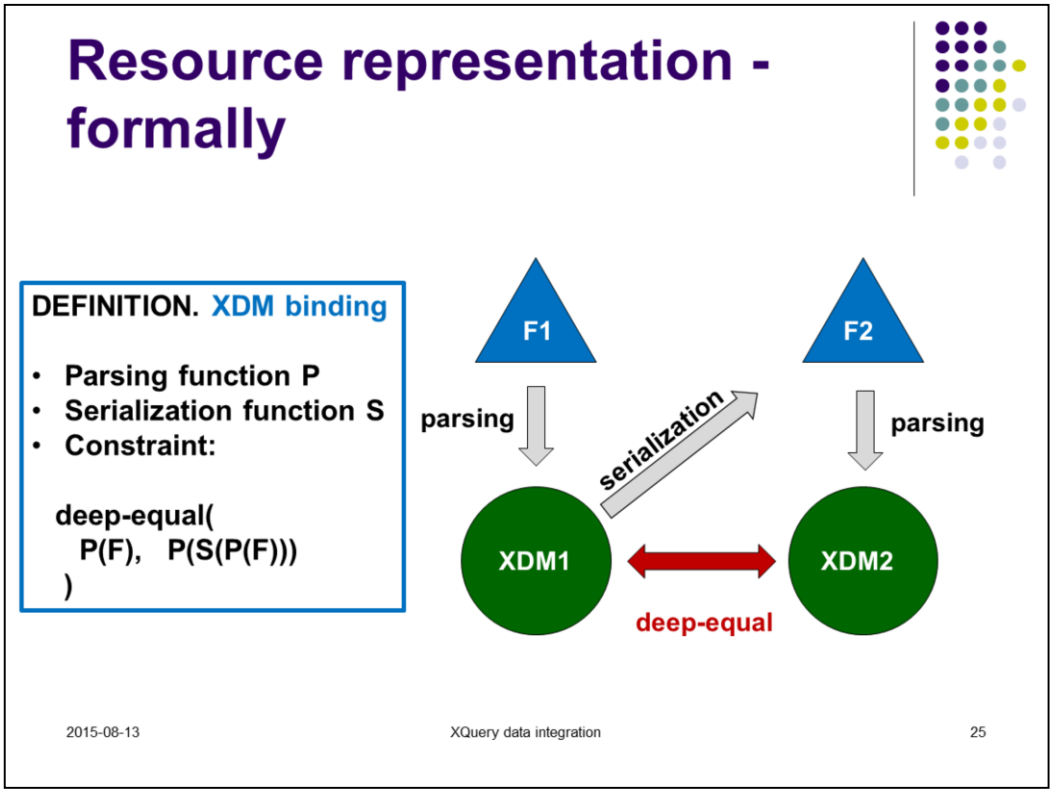


Let us first consider the concept of resource access. In XQuery, resource contents must be represented by an XDM value. Resource access is provided by an XQuery function returning a resource representation.



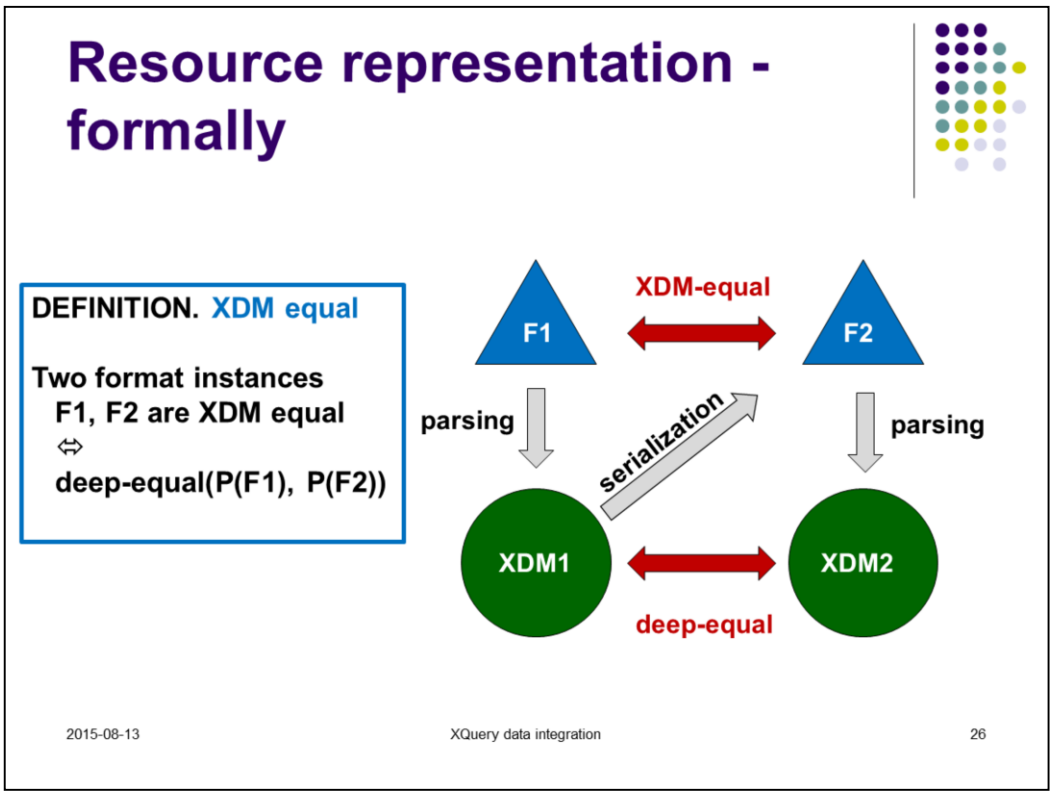
We should be aware of various access pattterns. Resource contents are delivered in exchange to a resource URI, or to the resource text itself, or a request message, or a database query. The resource representation can be an XML node tree, a tree of nested maps/arrays, or simply a string.

Resource representation - formally



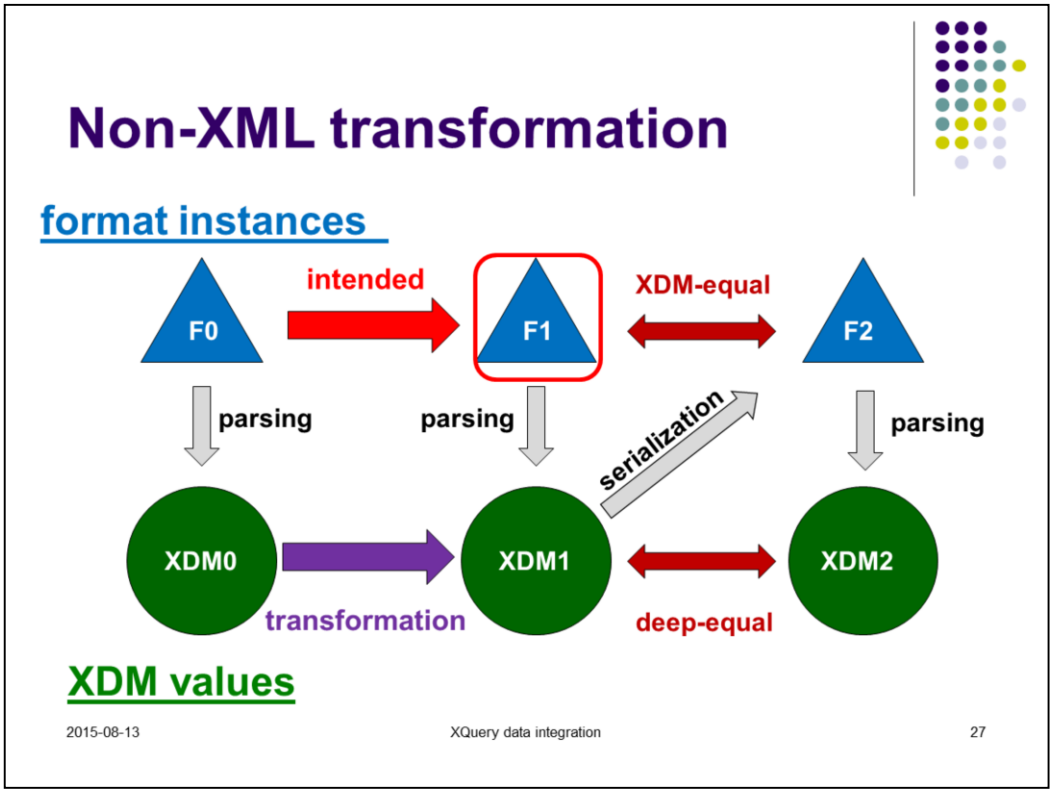
If data integration becomes a strategic goal, we may need new concepts defining the relationships between XDM values and resource instances, expressing constraints on information loss and the effects of round-tripping. Here comes a proposal for a formal definition of resource representation.

Resource representation - formally

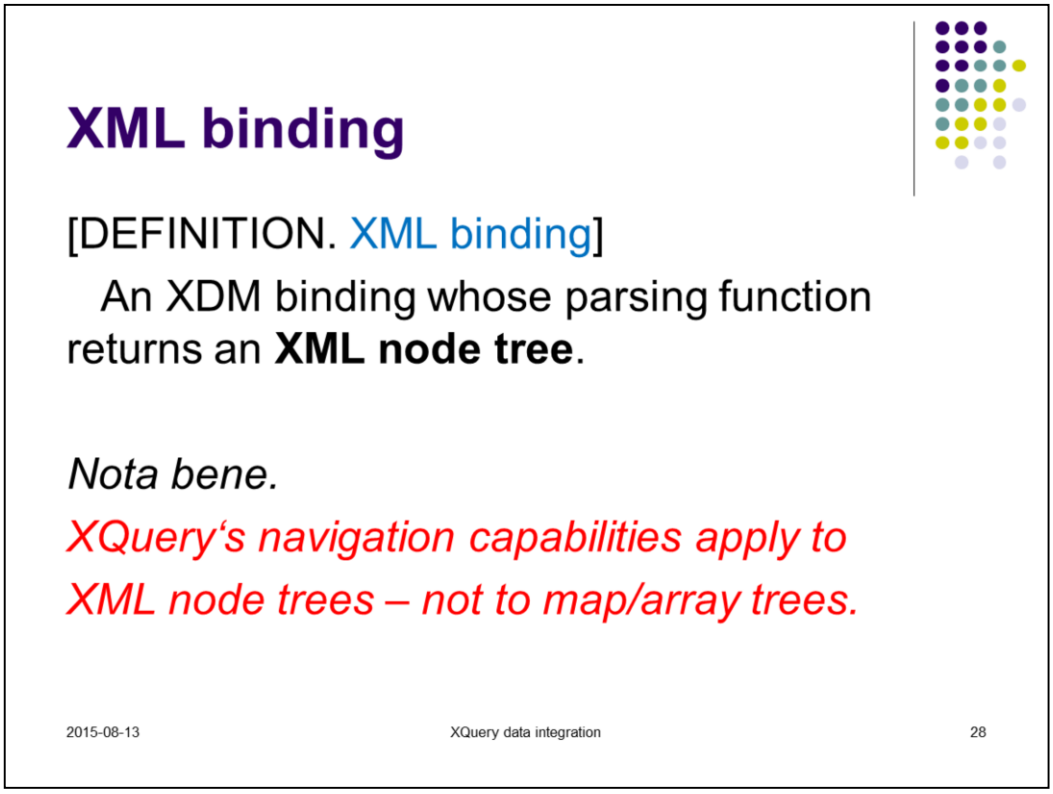


XDM bindings create a formal bridge between format instances and XDM values. Now we can compare format instances by comparing the associated XDM values

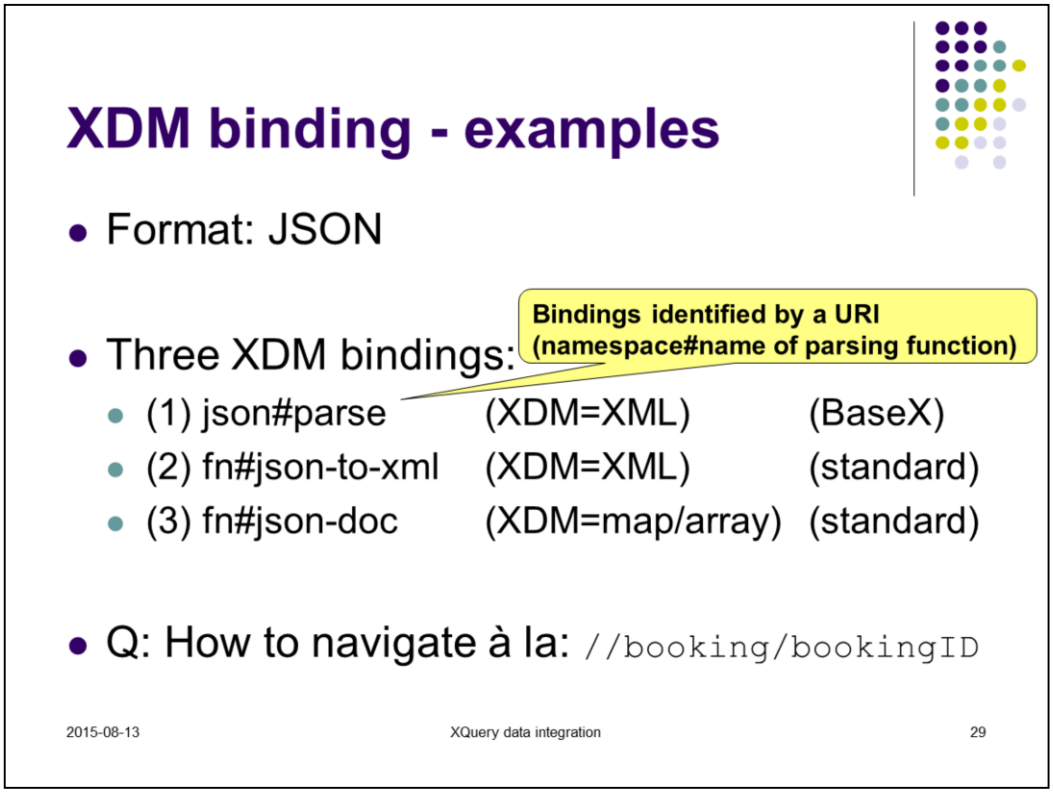
.



And now we can define the transformation between format instances in terms of XDM manipulations. A transformation between two format instances can be accomplished as the transformation between the corresponding XDM values as defined by a particular XDM binding. The result of the XDM transformation can be serialized into a format instance which is guaranteed to be XDM equal to the intended end result.



XML binding is just a special case of XDM binding – the XDM represention is an XML node tree. This is an important aspect as XPath navigation can only be applied to node trees, not to map/array trees.



Let us consider examples of XDM binding. Dealing with JSON resources and using the BaseX processor, three XDM bindings are available. The first is given by a BaseX-defined parsing function, which produces a BaseX-defined XML-representation. The second is given by a standard function, which produces a W3C-defined XML representation. And the third is given by a standard function returning a map/array representation. Now let us ask how to perform a navigation which in an XML context would be expressed by //booking/bookingID.

Navigation - XML binding

- Case 1: json:parse (XDM=XML (BaseX))

```
//booking/(., _)/bookingID
```

- Case 2: fn:json-to-xml (XDM=XML (W3C))

```
//*[@key eq 'booking']
/descendant-or-self::j:map[1]
/*[@key eq 'bookingID']
```

30

In the case of an XML representation the navigation is accomplished by a more or less simple path expression. The expression varies, dependent on which XML representation is used – the BaseX-defined representation, or the W3C-defined representation.

Navigation - non-XML binding

- Case 3: fn:json-doc (XDM=map/array)

```
declare function local:descendant($item as item(), $fieldName as xs:string) {
   typeswitch($item)
   case map(*) return (
     let $field := $item($fieldName)
     return
       if ($field instance of array(*)) then $field?* else $field,
     map:keys($item)[. ne $fieldName] !
       local:descendant($item(.), $fieldName)
   )
   case array(*) return $item?* ! local:descendant(., $fieldName)
   default return ()
};
local:descendant($data, 'booking')?bookingID
```

31

If the resource is represented by nested maps and arrays, the navigation can only be accomplished by writing complex code. This is not navigation any more – this is just a piece of tough programming work. Sorry - the XDM value obtained for the JSON document is not in scope of XQuery‘s outstanding navigation capabilities.

Map/array navigation issues

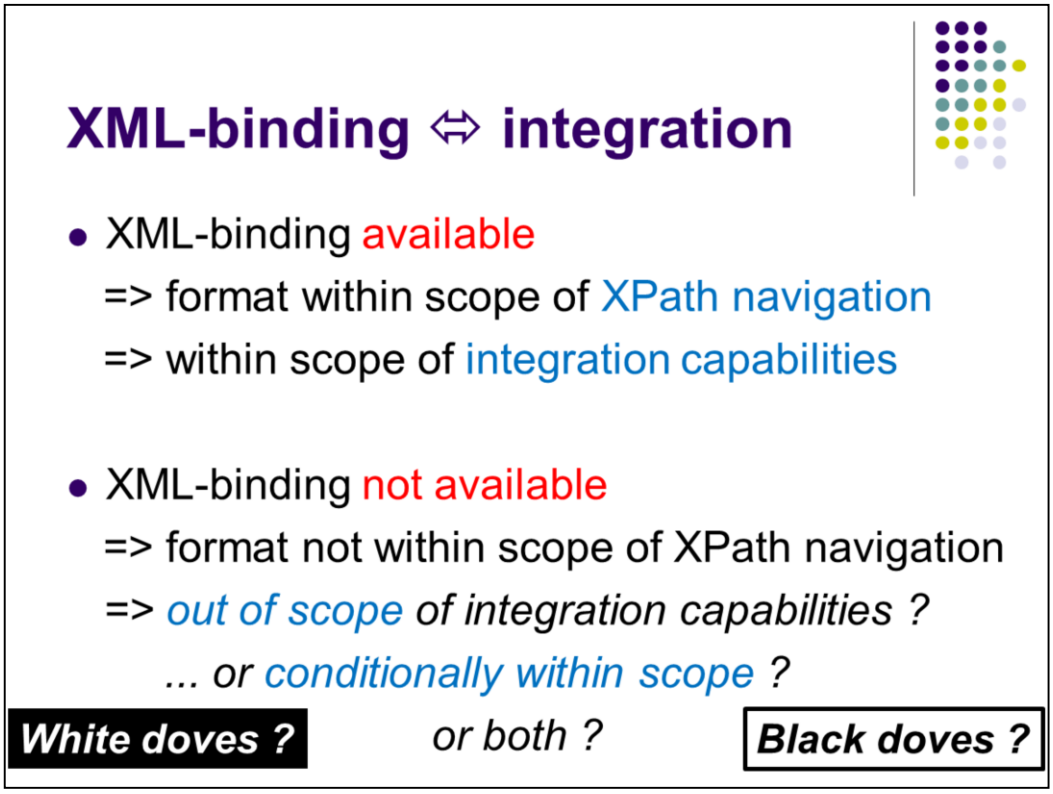
Primitive navigation: `a?b?c`

- Only child axis – no deep, no upward navigation
  - No deep navigation — no `//foo`
  - No upward navigation — no `ancestor::booking`
- Result heterogeneous (map, array, string, ...)
- Result anonymous — no `$res/name()`
- Knowledge of array-steps required

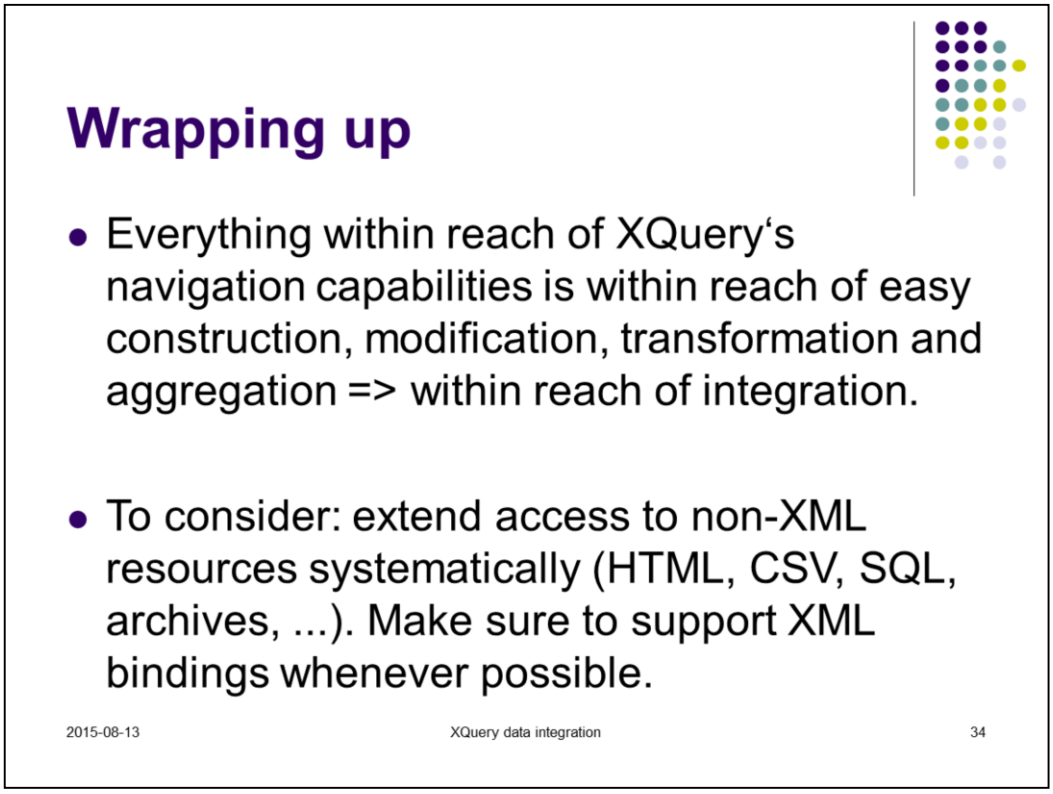
```
flights?*?departure?airport
```

- No modification — try `$j?a?b!map:put(.,'c','X')`

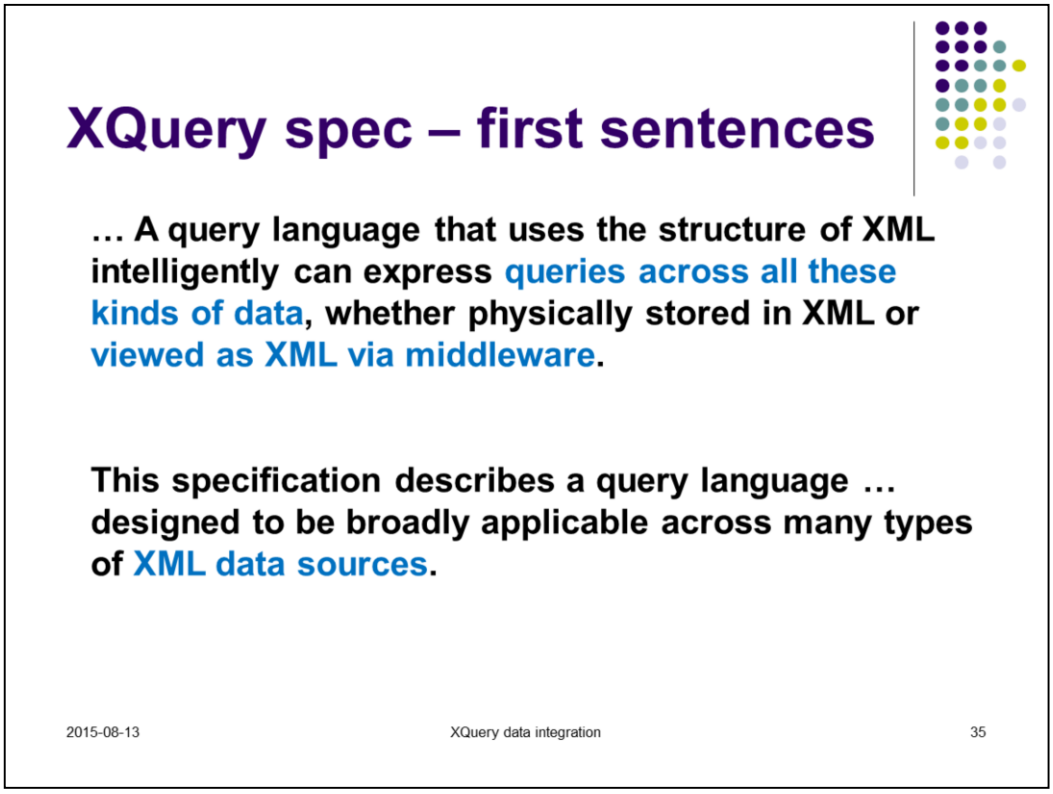
Map/array trees can be navigated in a primitive way, using the lookup operator (?). It is important to realize that this navigation simply cannot be compared with XPath navigation.



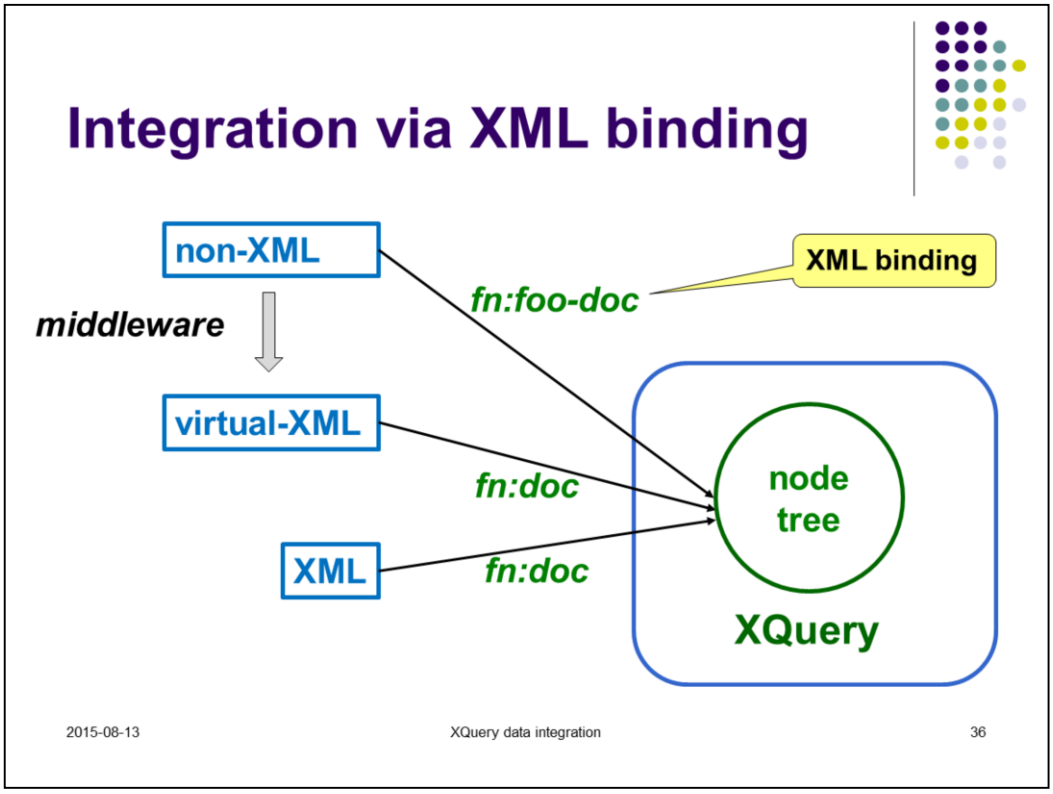
If for a given format F an XML binding exists, it is fully within scope of XQuery‘s navigation capabilities. This seems to imply that the format is within scope of XQuery‘s integration capabilities. But what if only a map/array binding is available? Certainly, the format is not within scope of XQuery‘s navigation capabilities, and as a rule the ease of integration is reduced. But to which degree? Does the format fall out of integration, or is the potential of integration by an large preserved? The answer also depends on the complexity of the format instances – the less complex, the smaller the disadvantage of having to do without strong navigation. The answer might be similar to the answer whether Escher‘s graphic shows white or black doves.



Wrapping up, XQuery's capabilities of integration are tightly coupled to its capabilities of navigation. If the XQuery Working Group should identify data integration as a strategic goal of XQuery, non-XML resource access would need to be extended systematically, and a strong emphasis on XML bindings would be important.



Coming to an end, I return to the beginning of XQuery, the first sentences of the spec... These sentences are about data integration. They envision XML as a unified view on many kinds of data, a view made available by middleware services creating XML on the fly. This picture turns XQuery into a data integration language, which, however, depends on middleware – infrastructure outside of the language. But by now the acceptance of XML is decreasing and such middleware cannot be taken for granted.



But the vision of XQuery as a data integration language might be preserved by introducing a second way how non-XML data might become node trees: not only by middleware services, but by parsing functions, those of XML bindings.

A conceivable extension

… A query language that uses the structure of XML intelligently can express queries across all these kinds of data, whether physically stored in XML or viewed as XML via middleware, or bound to a node tree representation within XQuery itself.

This specification describes a query language … designed to be broadly applicable across many types of XML and XML-bindable data sources.

So... Let‘s edit the text a little...

Thank you for looking!

... and call it a day (or night) – thank you for listening.