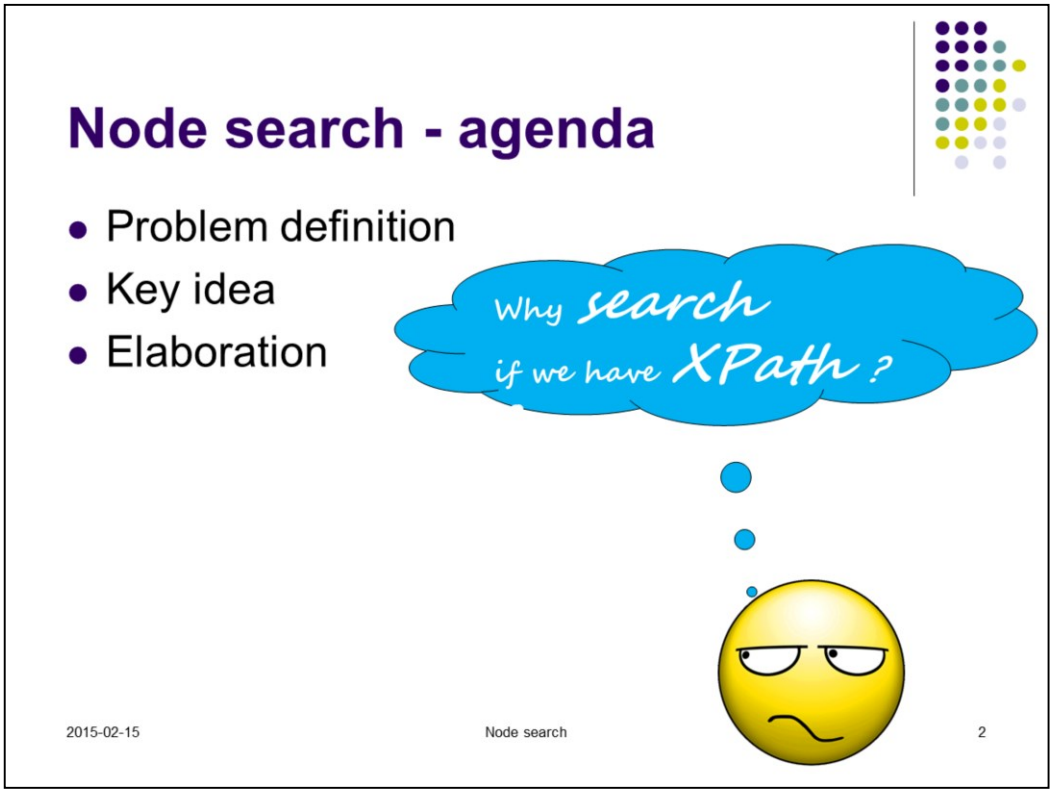
Node search

Node search preceding node construction – XQuery inviting non-XML technologies

Hans-Jürgen Rennau, Traveltainment GmbH
February 15, 2015

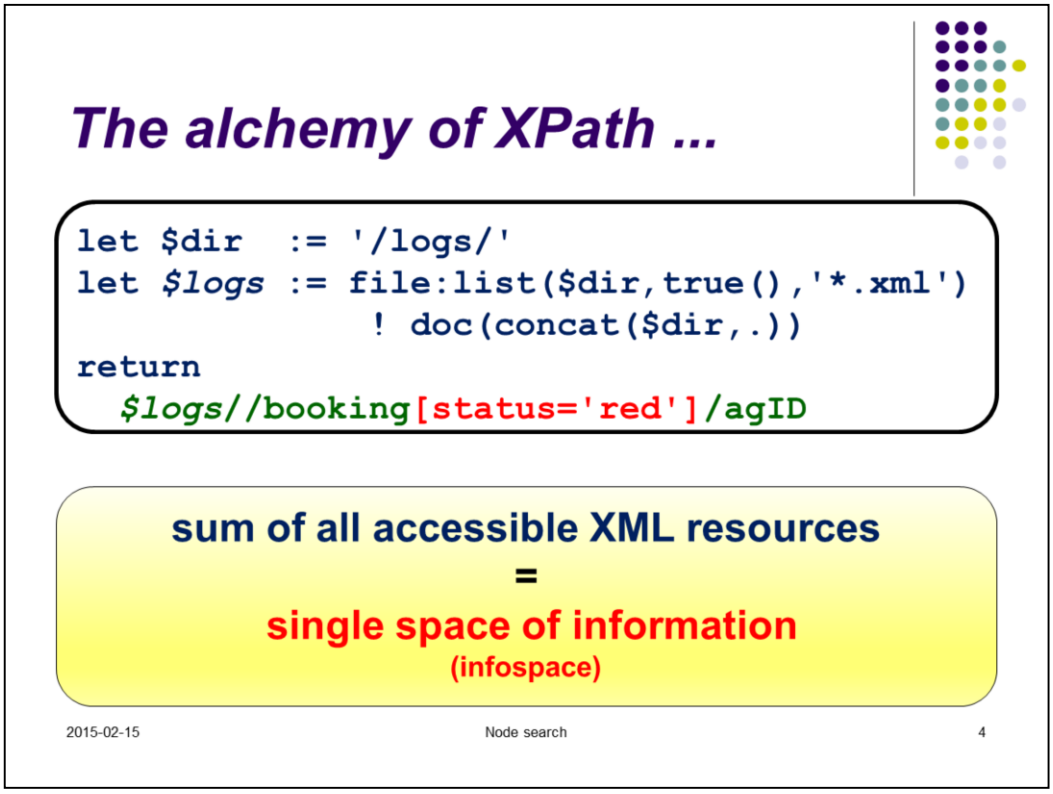
The heart of XML technologies is XPath, a small expression language for finding XML content. This presentation is about an attempt to enhance the capabilities of XPath by a new feature.



As a starting point, a basic limitation of XPath is discussed, which becomes obvious when distinguishing the operations „navigation“ and „search“. My response to this limitation is based on a simple idea. It will lead to new XPath functions enabling a search for nodes which is different from conventional navigation. In particular, it may be powered by XML and non-XML technologies alike.

The problem

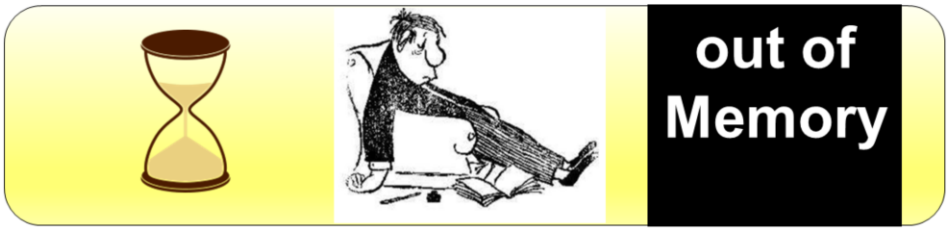
So what‘s the problem? Isn‘t XPath amazing?



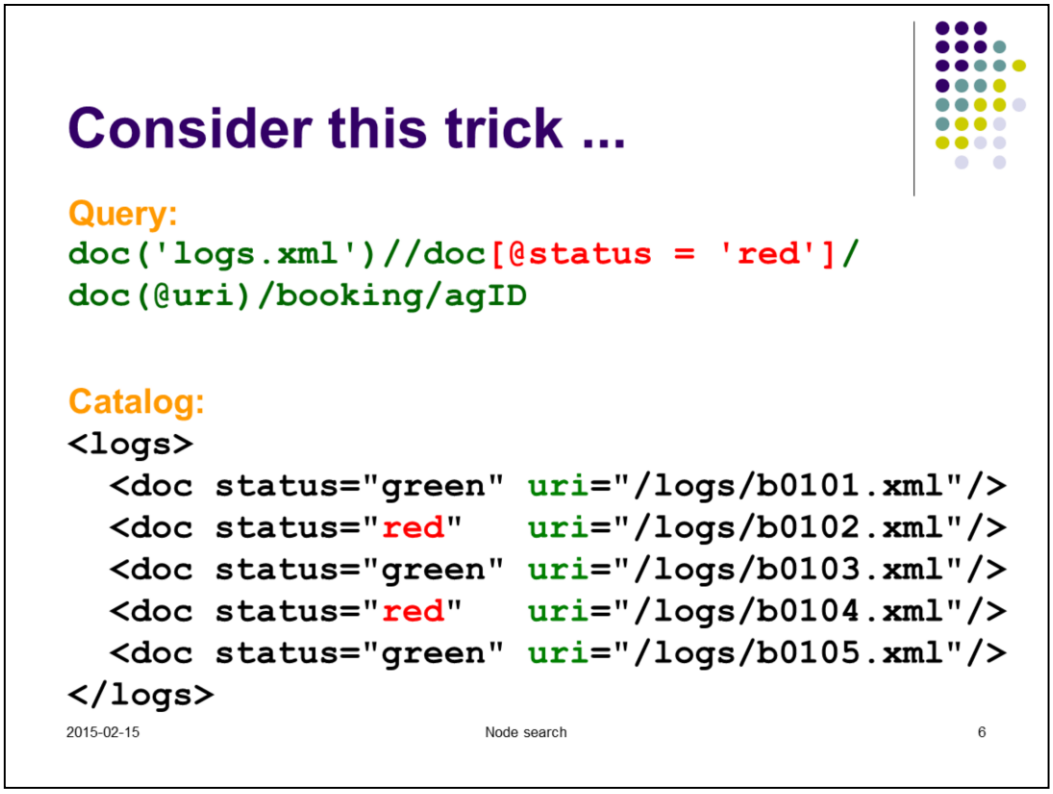
Its effect verges on alchemy. The sheer existence of XPath transforms any XML document – and also any set of XML documents – into a single space of information, ready for navigation. Almost like a database...

REALITY FIGHTING BACK

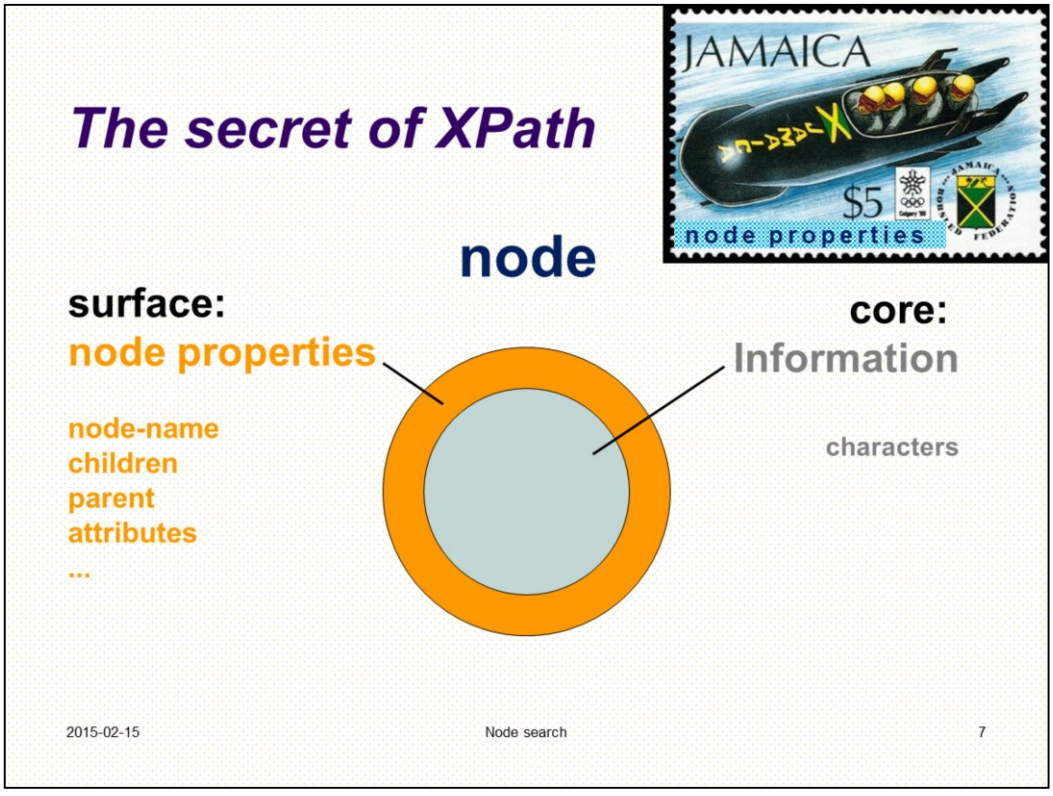
```
let $dir  := '/logs/'
let $logs := file:list($dir,true(),'*.xml')
               ! doc(concat($dir,.))
return
  $logs//booking[status='red']/agID
```



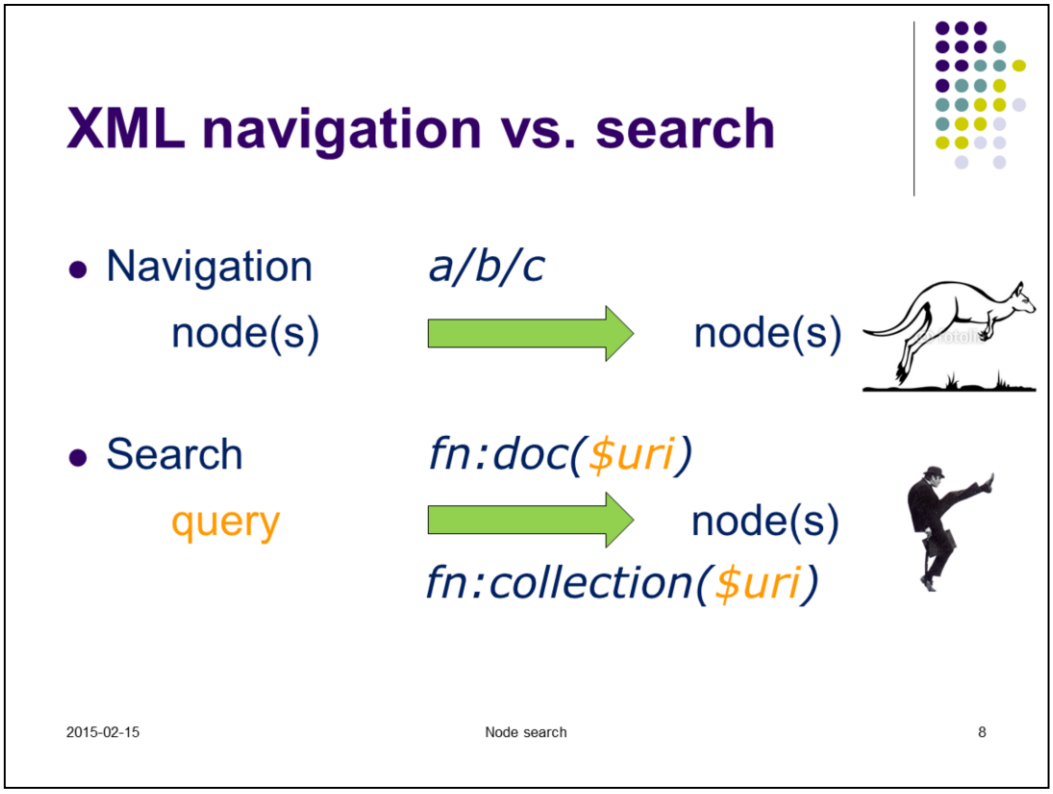
But when there are very many or very large documents, our query still works... and works ... works ... and stops. Out of memory error. Frustrating! A primitive grep command would never let us down like this. The problem: the query cannot filter documents without parsing them – before filtering, every candidate document must be transformed into a node tree. We have a scaling problem.



Now consider a trick. We create a catalog in which the query can look up the status of a document, without parsing the document. It is simple. Just associate URI strings and status values. Relying on such a catalog, we can rewrite our query so that it does not parse any documents which do not have a red status.

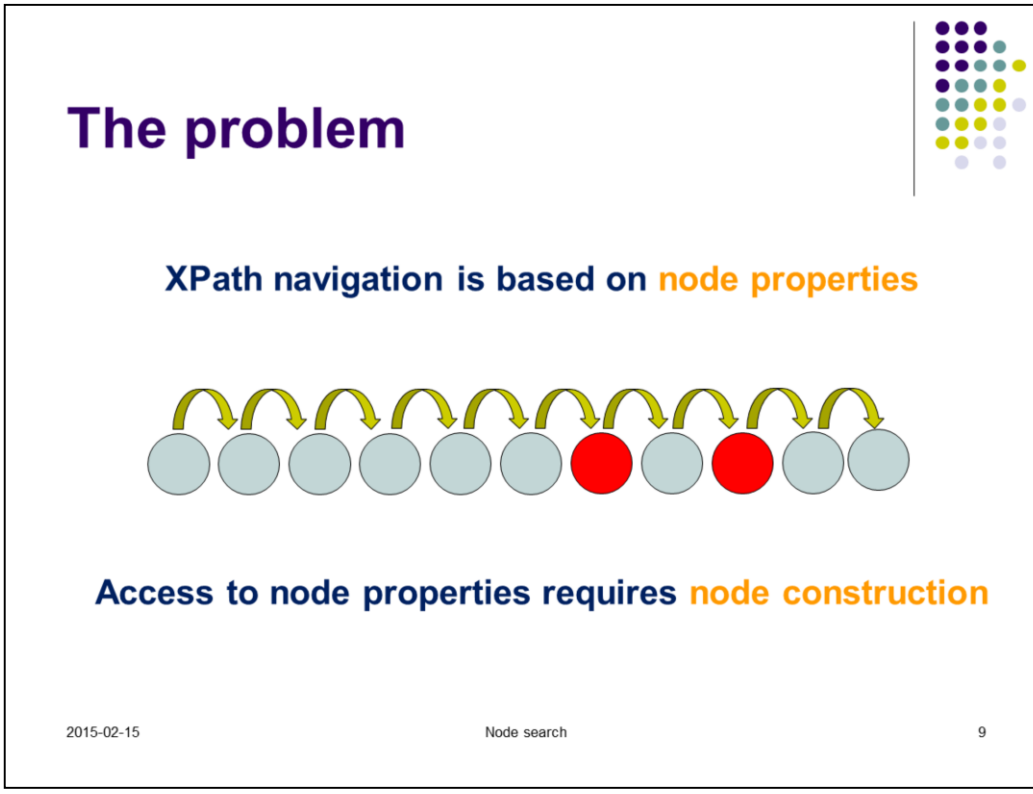


The power of XPath is based on a secret, which is the modelling of XML items (documents, elements, attributes) as nodes. A node associates a core of information with a surface of node properties (node name, child nodes, etc.). These properties provide the surface along which XPath navigation rushes. Navigation is guided by node properties. In fact, it DEPENDS on node properties.



XPath excels in navigation – the movement from nodes to nodes, guided by node properties. But there is very little support for a related operation: node search, which also leads to nodes, but starts without nodes, with only a query - information about the nodes of interest. XPath support for search is restricted to the functions fn:doc and fn:collection. The only search query which XPath understands is a single URI...

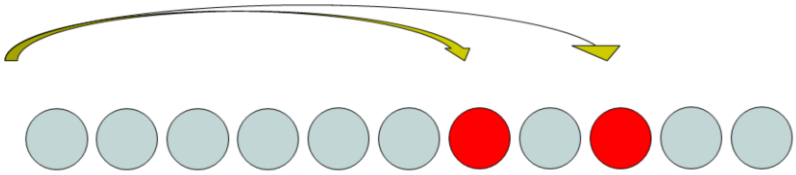
The problem



So the smartness of XPath is navigation, not search. XPath navigation is based on node properties. Access to node properties requires node construction, so node construction must precede navigation. XPath cannot search for red bookings, it must construct all node trees and then navigate to the status elements. If the processing involves the selection of few target nodes from very many candidate nodes, we run into a scaling problem. XML databases solve the problem by maintaining persistent representations of the node trees, as well as indexes. But there are also several issues, above all the lack of underlying standards, the lack of portability and the closed nature of the technology. More about that later.

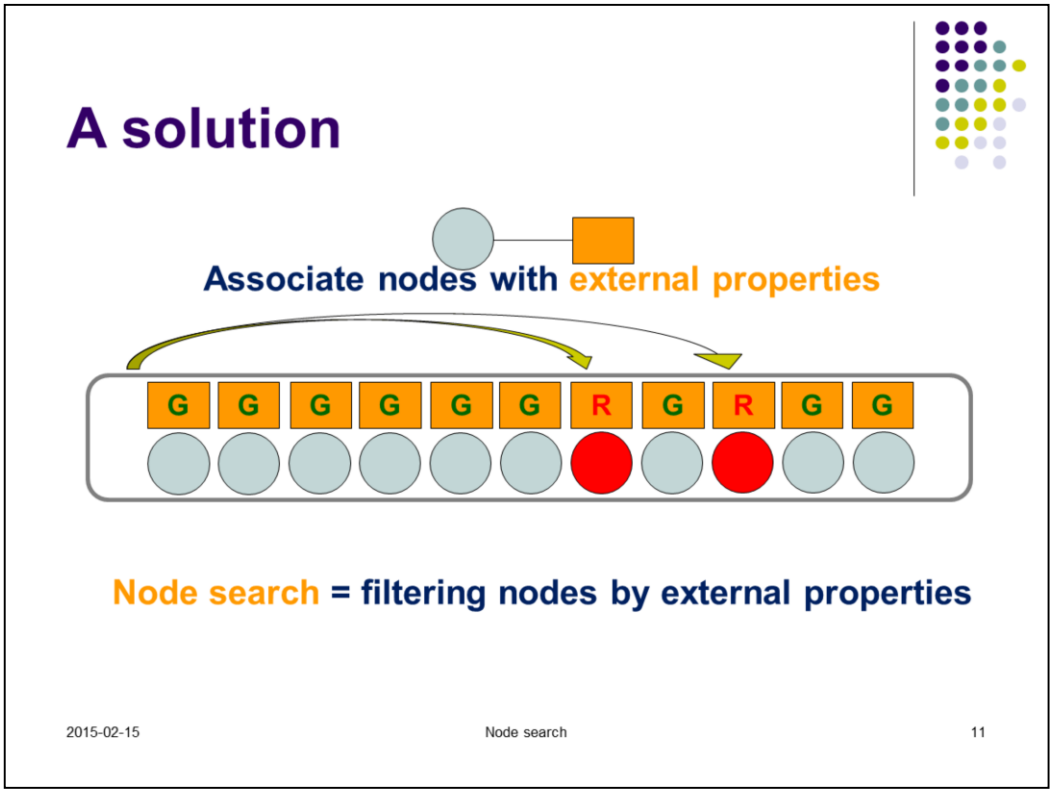
The goal

To complement XPath navigation with a node search

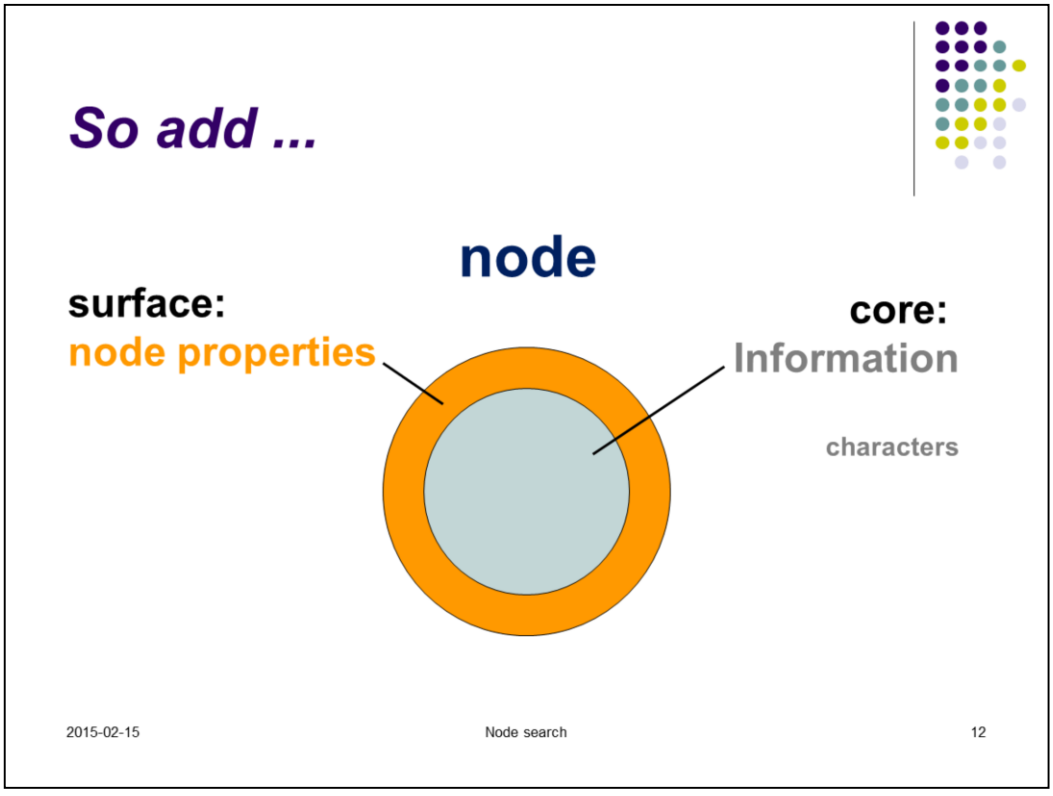


which does not require node construction (and is portable)

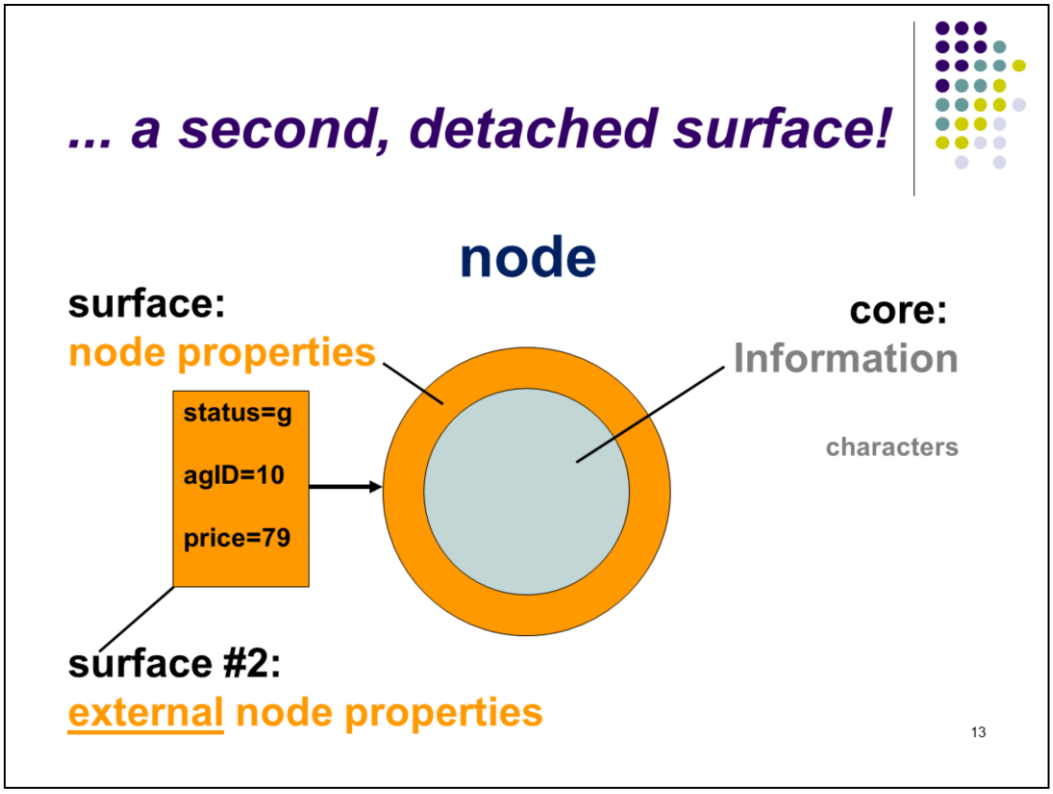
Problem understood, goal defined.



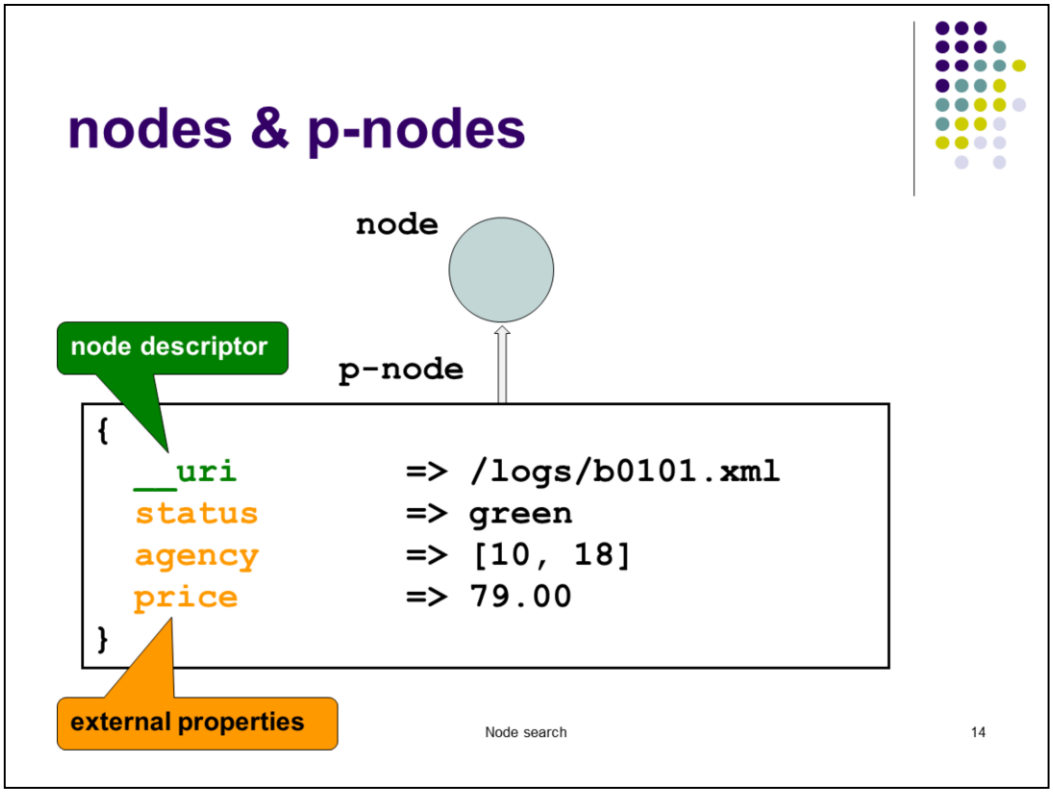
A possible solution has been suggested by our trick. The gist is to associate nodes with information describing the node, but kept outside of the node. We may call this information external properties of the node. They enable node selection not preceded by node construction. Node search is understood as the filtering of a collection by the external properties of its members.


So add ...
node
surface:
node properties
core:
Information
characters
2015-02-15
Node search
12

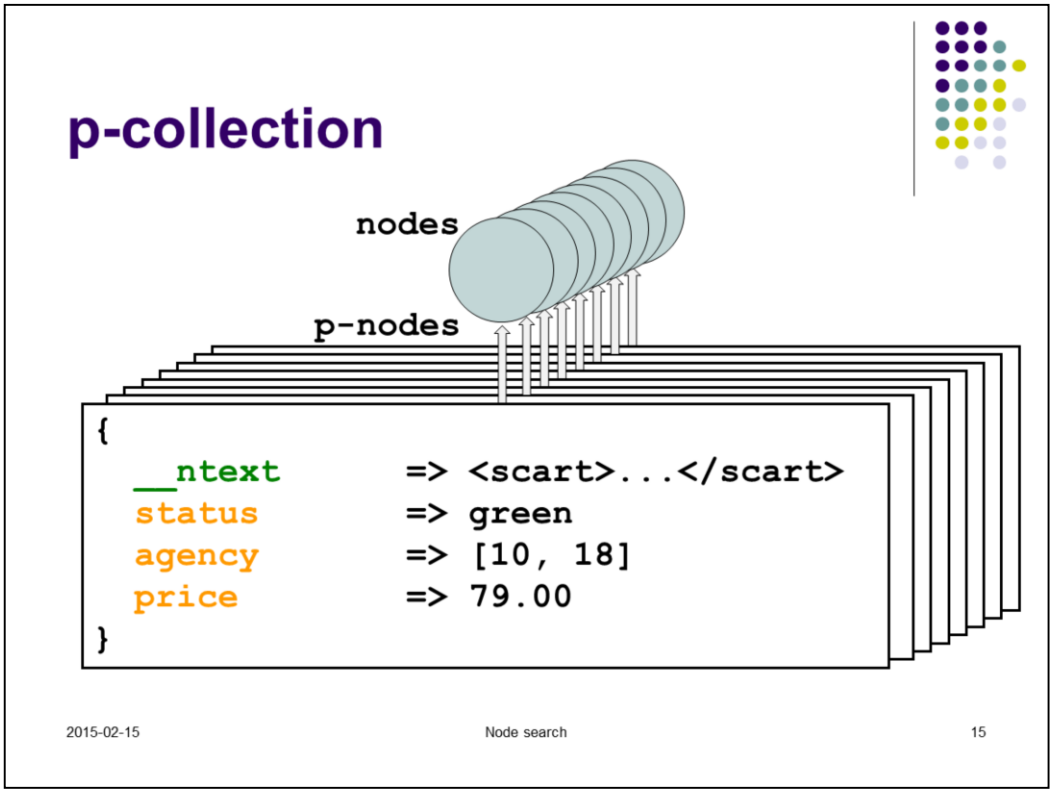
So we add ...



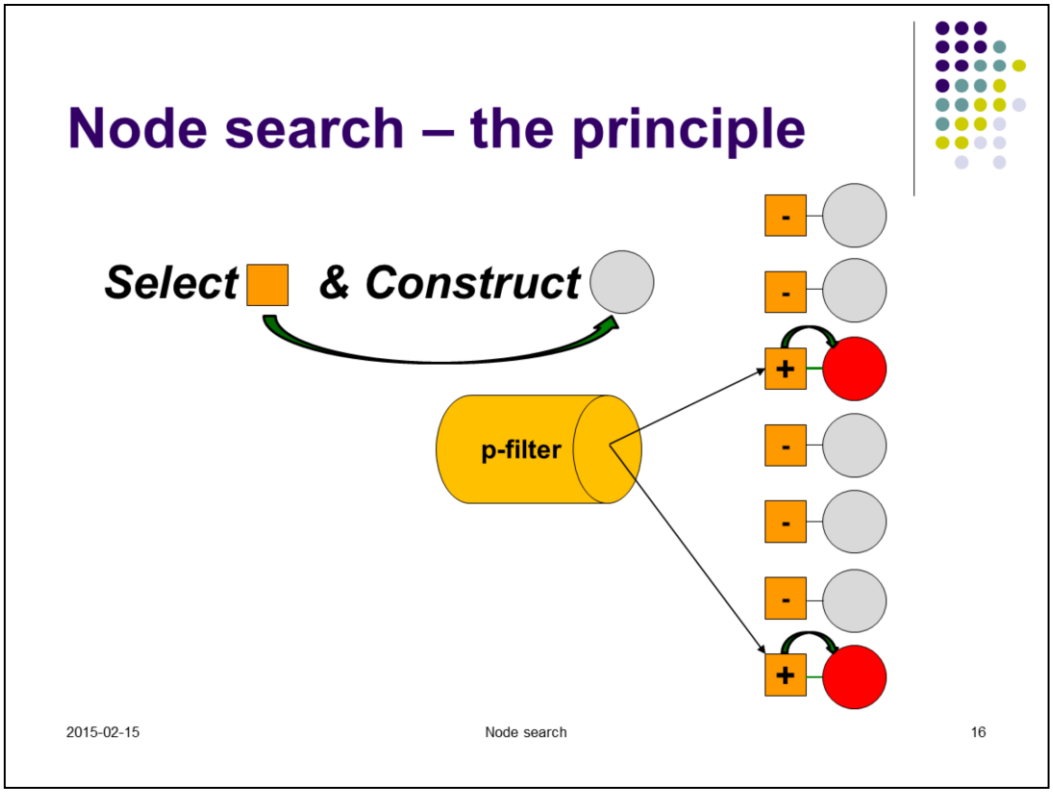
to the nodes a second, detached surface. Being detached, these surfaces can be stored and queried without constructing nodes.



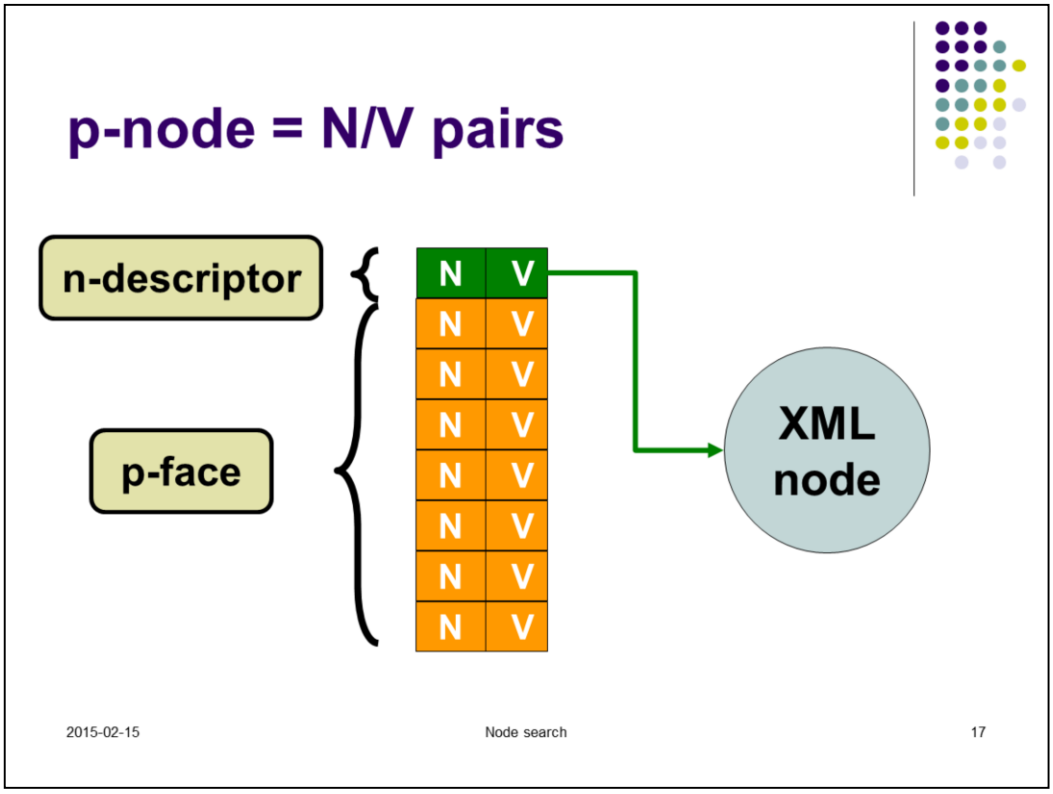
A new unit of information – called a p-node – associates the external properties of a node with a node descriptor, which is a string enabling the construction of the node should the node be needed. The node descriptor might be a URI, or some other kind of reference, but it may also be the serialized node itself, which is just a string.



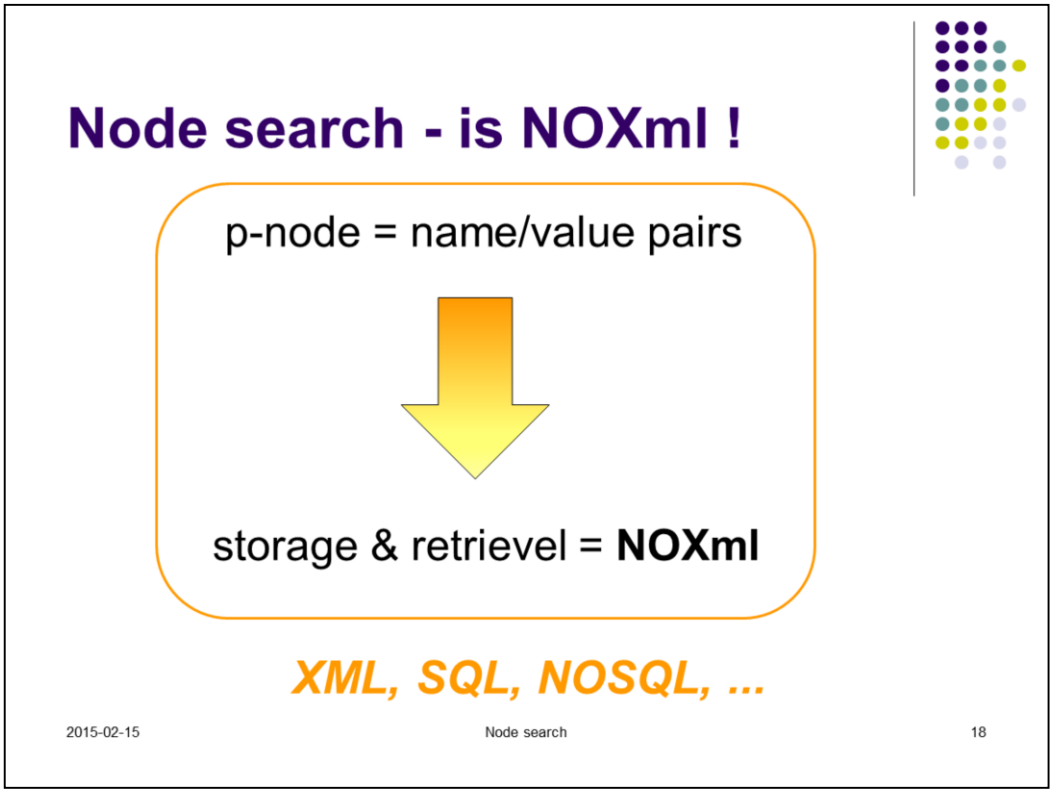
A p-collection is a set of p-nodes, which represent a set of XML nodes, but can be stored and queried without constructing those XML nodes.



The principle of node search is a two-step procedure: (1) filter the p-nodes, (2) construct the nodes associated with the selected p-nodes.

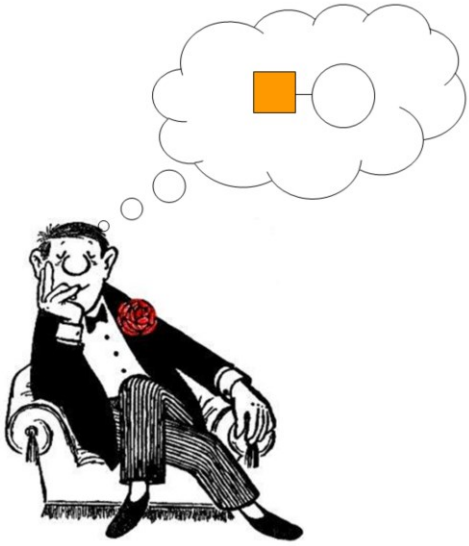


Note that a p-node is just a set of name/value pairs: one pair is the node descriptor, enabling node construction; the other pairs represent the external properties of the node, enabling node selection.



As p-nodes are just sets of name/value pairs, their storage and retrieval might be handled by XML and non-XML technologies alike. Node search is NOXml – not only XML.

The idea.

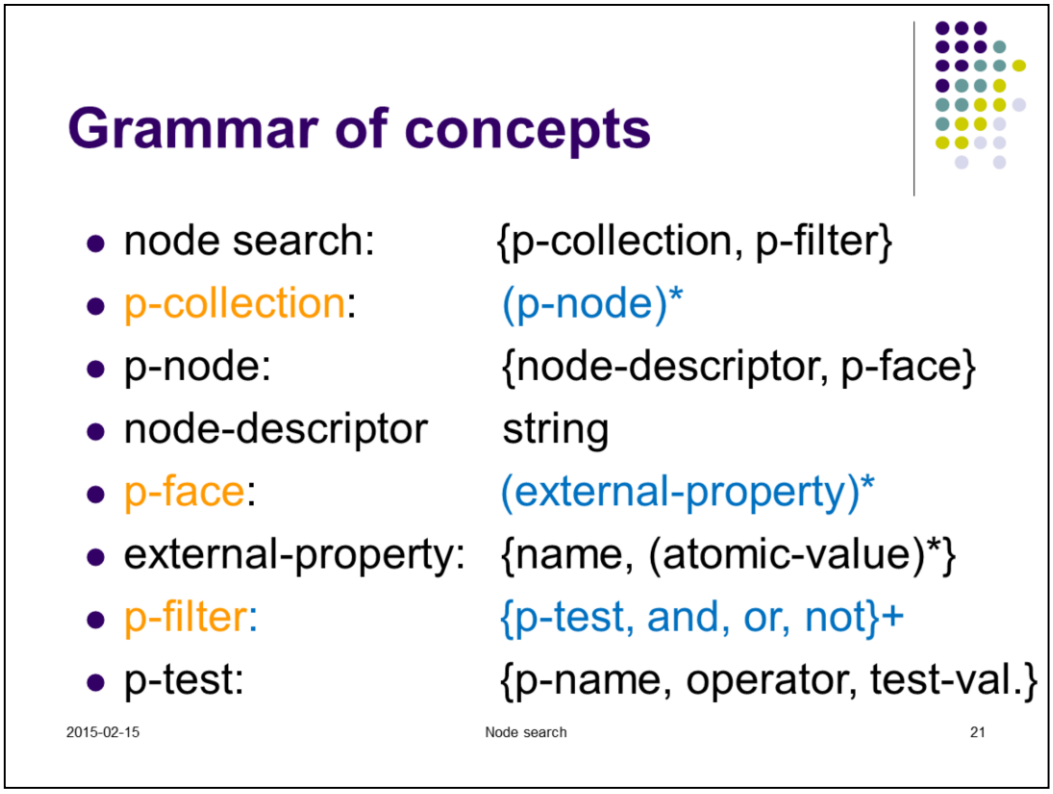


So much for the idea.

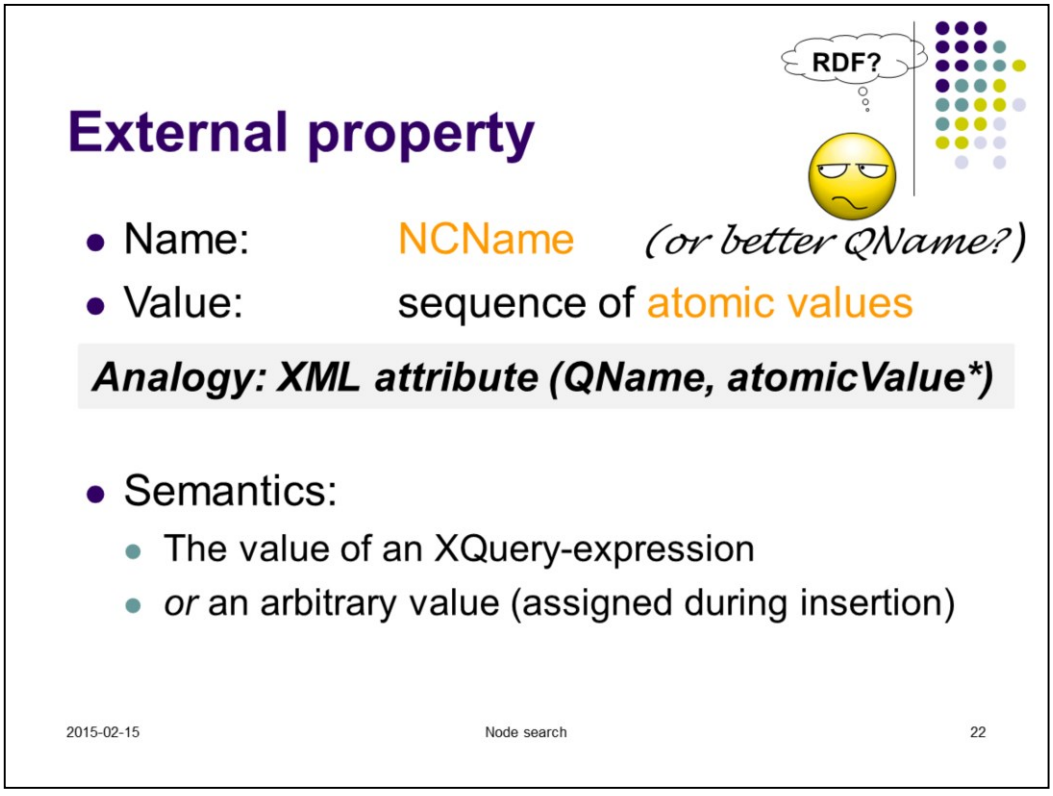
The elaboration

- grammar of concepts
- models
  - p-face, p-model, p-filter
- APIs
  - node search
  - collection management

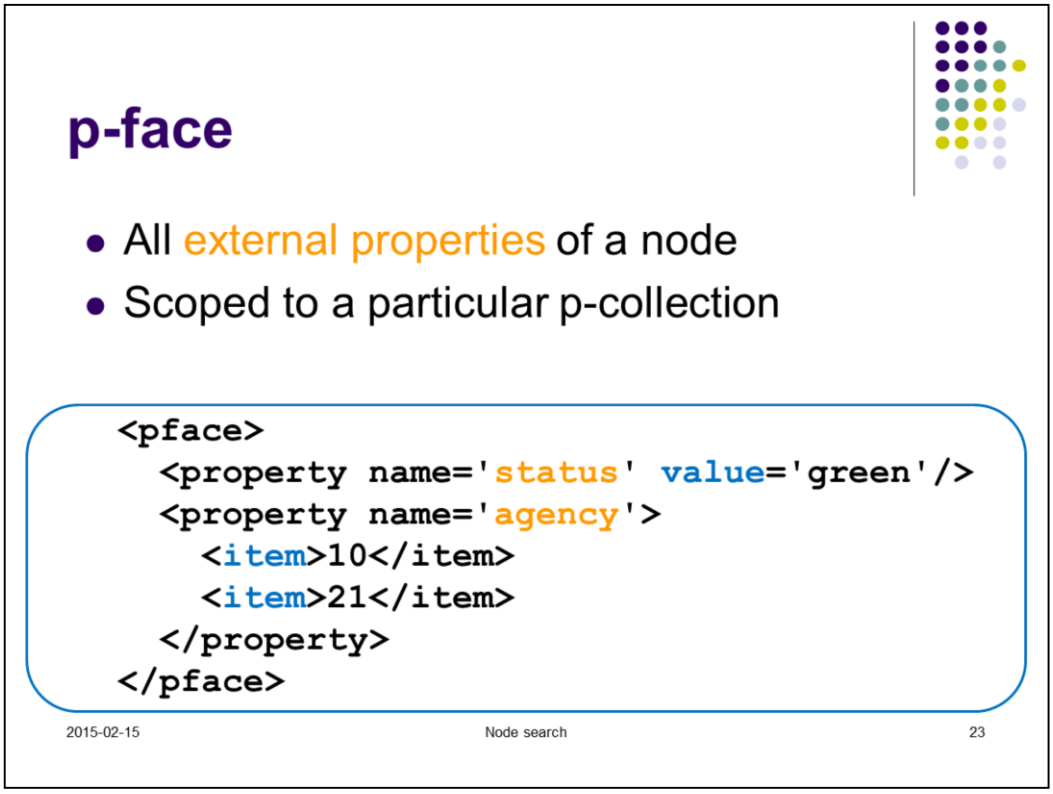
What is the idea worth? Its value depends on its elaboration.



The goal of portability requires a seamless integration of node search into XPath. Node search must be defined in terms of concepts which are aligned with the XPath data model (XDM).



An external property is a name/value pair. The name is an NCName and the value is a sequence of atomic values. The semantics of the property are not constrained. The value may be some value extracted from the node, or, more generally, the value of an XQuery expression evaluated in the context of the node. But the value may also be independent of the node content, perhaps providing contextual information, like the timestamp of node creation. One more remark. Constraining the propertiy names to be NCNames made things easier, but may not be the last word. Perhaps QNames are better, as they might enable us to use RDF triples as properties.



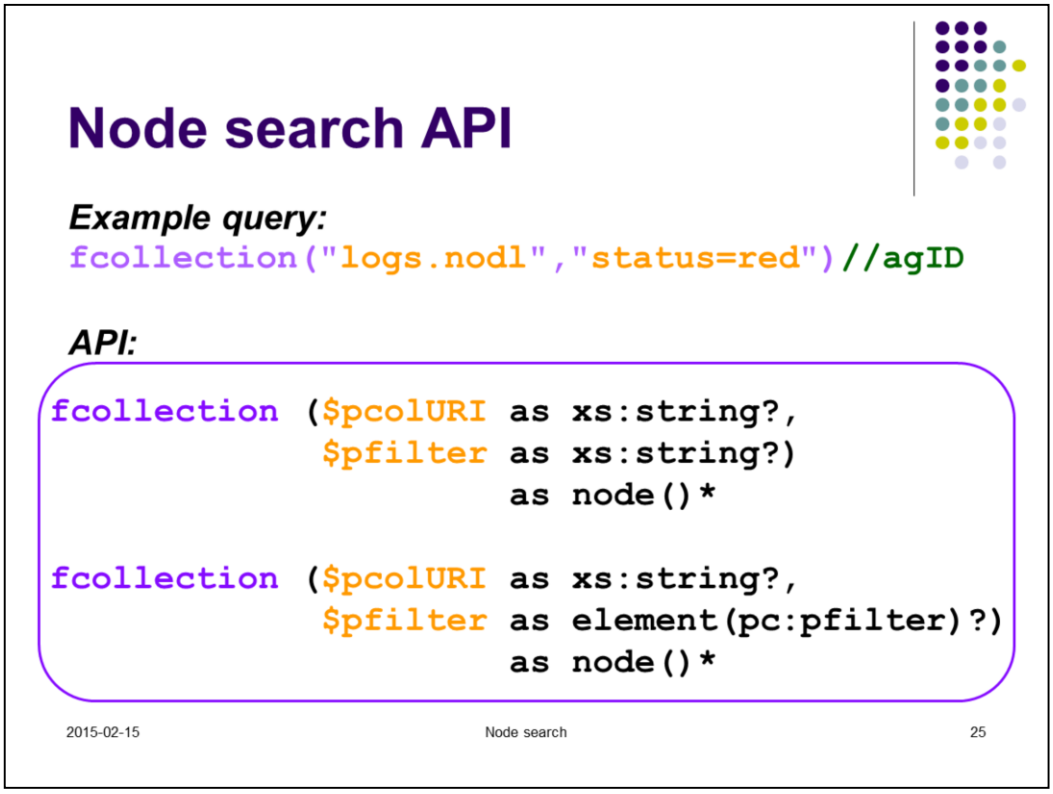
A p-face is the set of all external properties which a given node has in the context of a given collection.

p-filter

```
<pfilter>
  <and>
    <p name='status' value='red'/>
    <or>
      <p name='price' op='le' value='0'/>
      <p name='agency'>
        <item>10</item>
        <item>18</item>
      </p>
    </or>
  </and>
</pfilter>
```

status=red && (price<=0 || agency=(10,18))

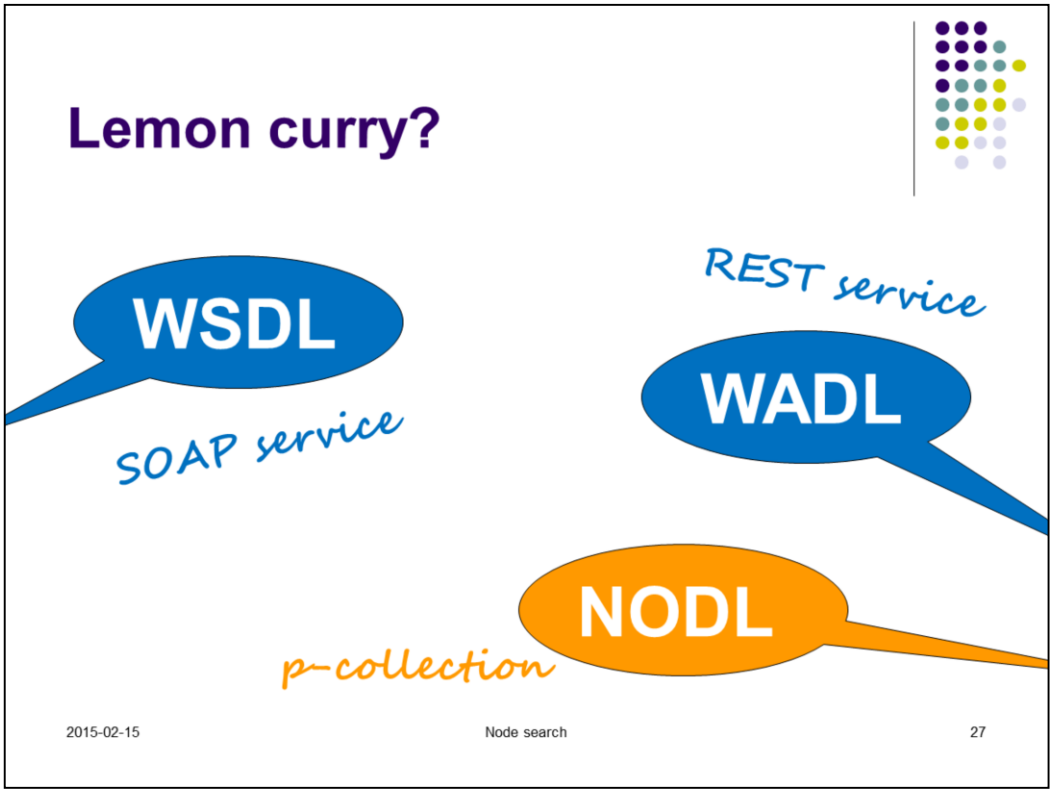
A p-filter is a condition applied to the p-face of a node. A p-filter consists of one or more p-tests combined by Boolean operators. A p-test is a triple consisting of property name, test value and comparison operator, e.g. „equal“ or „less than“. A p-filter can be expressed by a pfilter element, or by an equivalent query string.



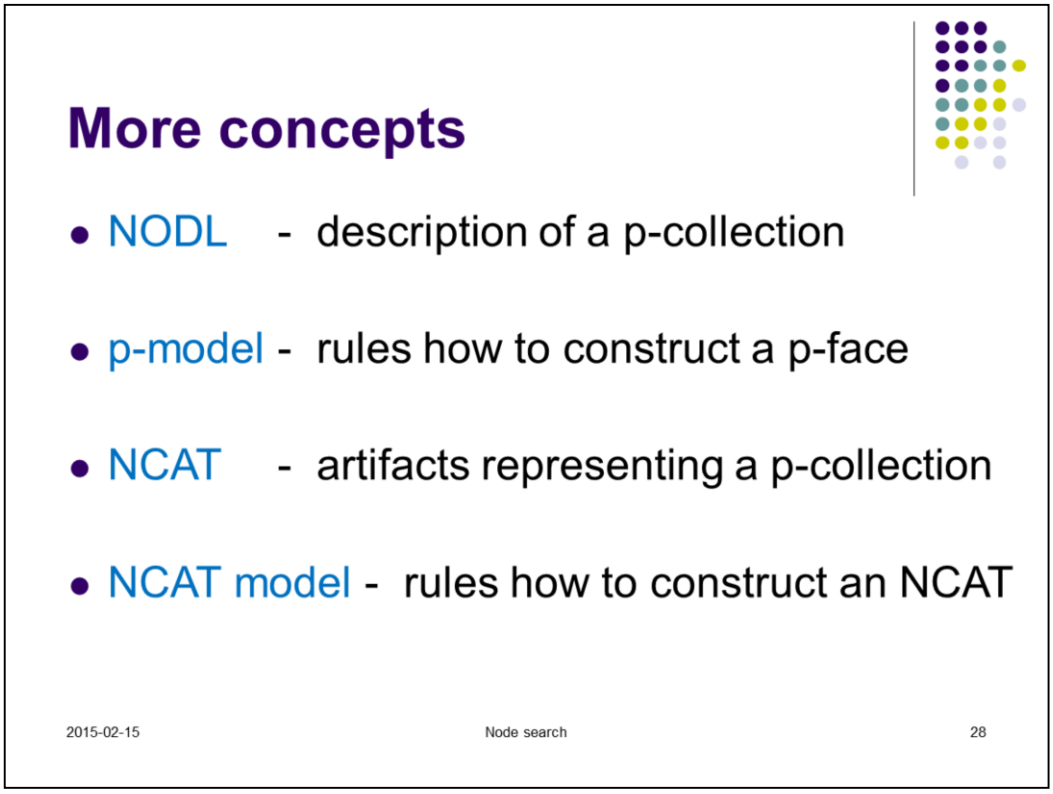
Now we are ready for the node search API. Node search means to apply a p-filter to a p-collection. Accordingly, the node search API is a single function with two parameters, the collection URI and a filter. It returns the nodes obtained by applying the filter to the collection.

p-collection management

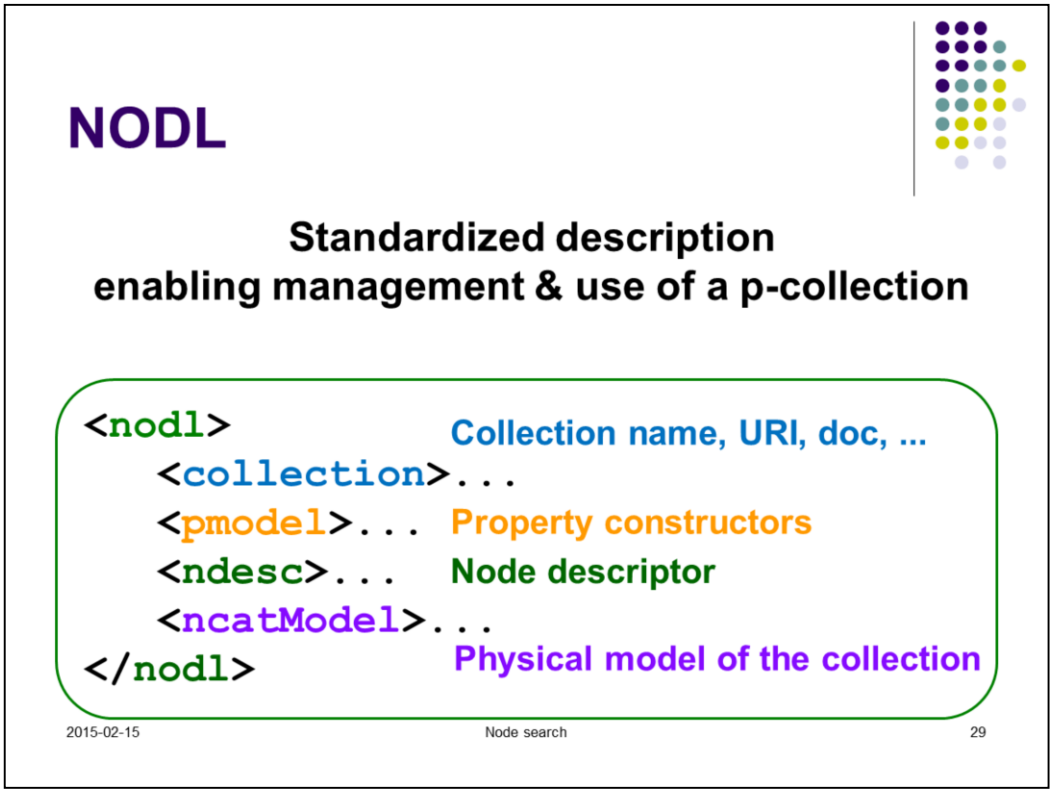
The usefulness of a node search based on p-collections stands and falls with the ease with which p-collections can be created and managed. p-collection management must be an integral part of the concept of node search.



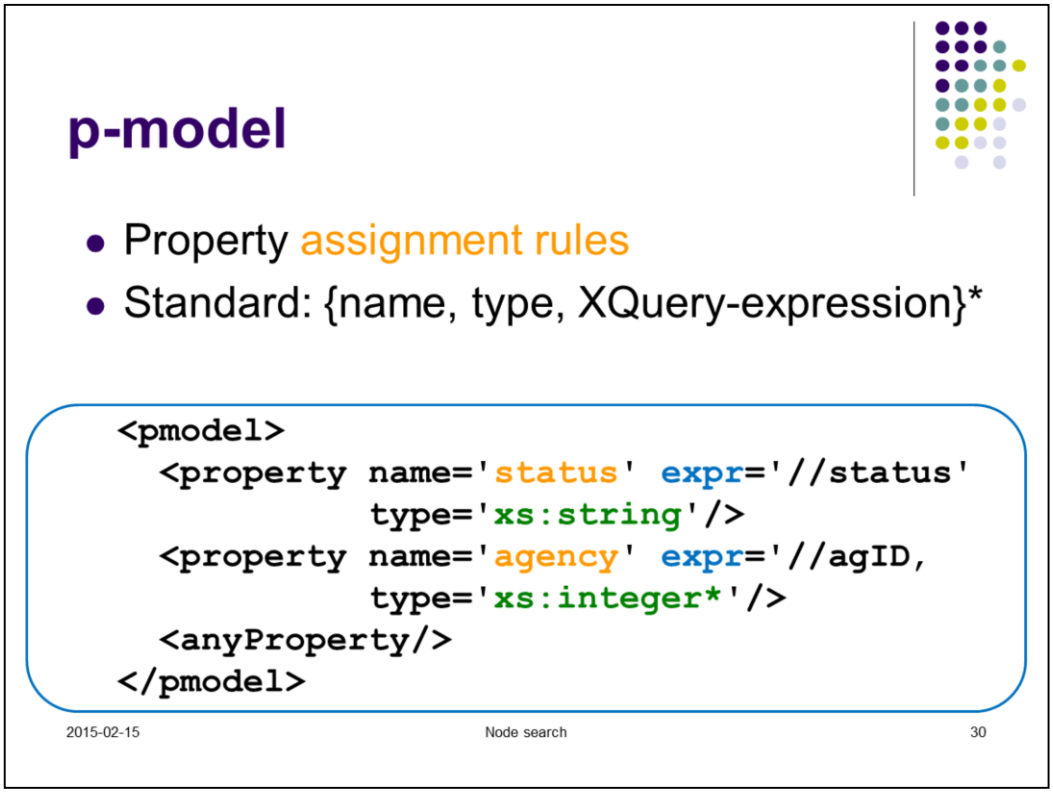
Here, the key concept is a NODL document, which plays a similar role in dealing with p-collections as WSDLs and WADLs play when dealing with web services.



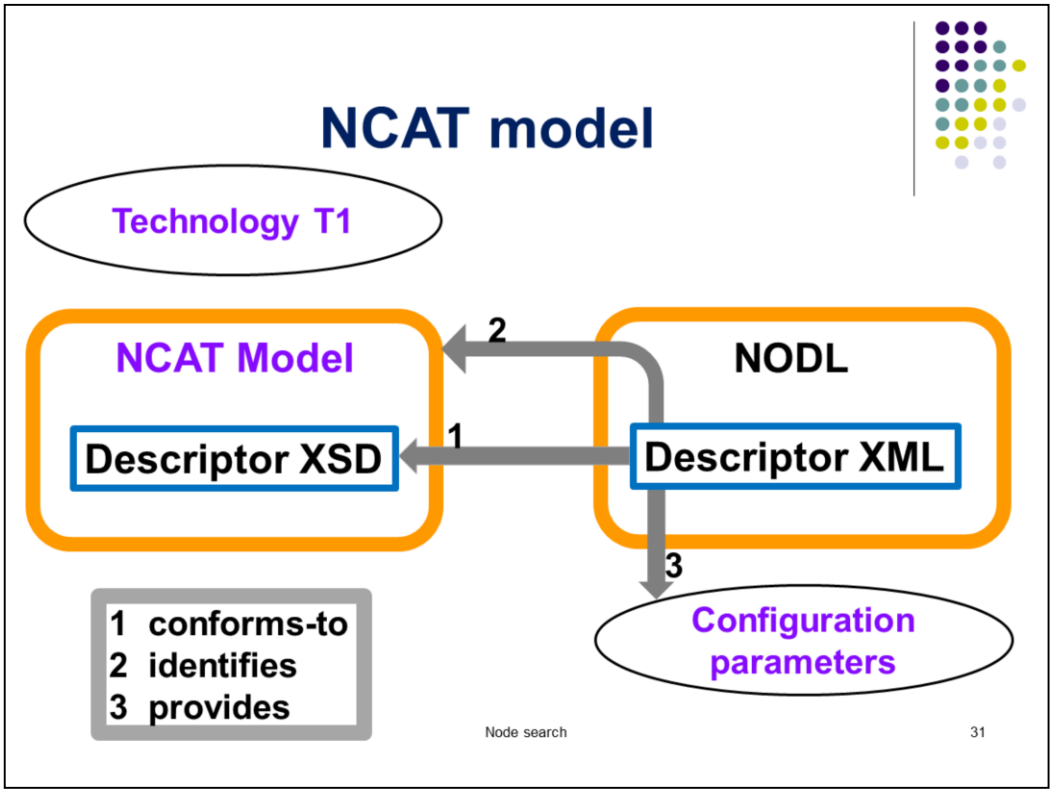
p-collection management relies on a few further concepts.



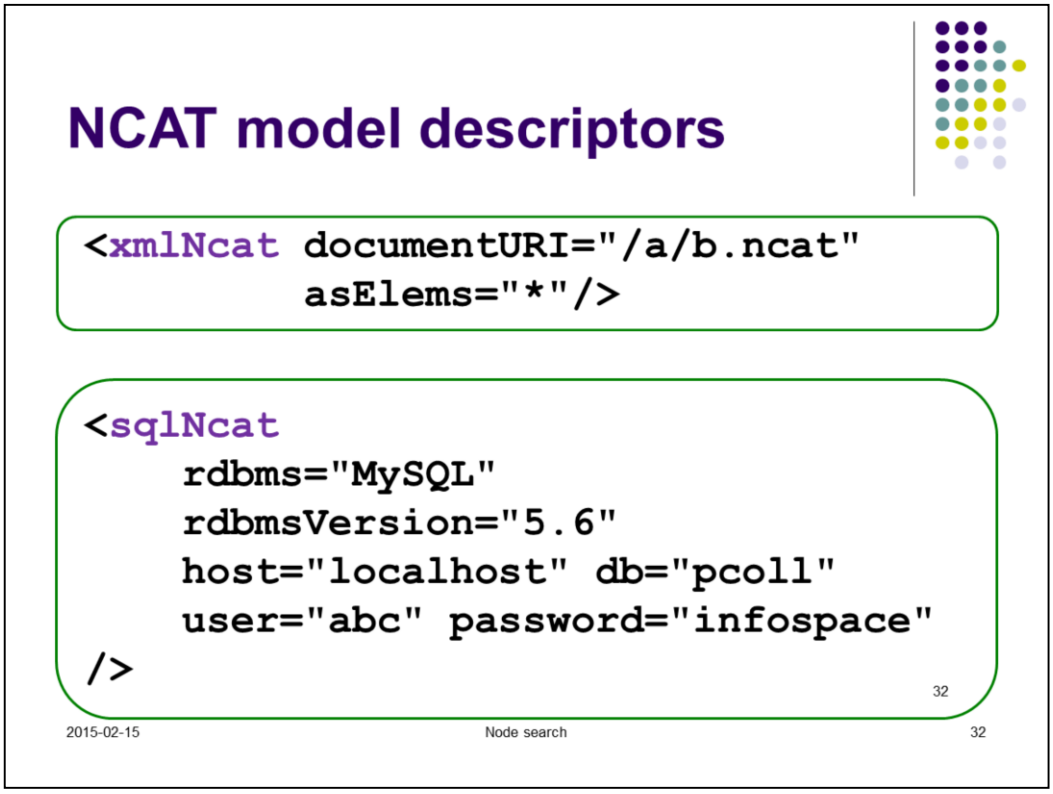
A NODL encapsulates all information which software dealing with p-collections needs to know. In particular, NODLs provide the information required for creating, updating and filtering collections. A NODL has four sections, describing the collection as a logical entity, the p-model, the node descriptor and the NCAT model.



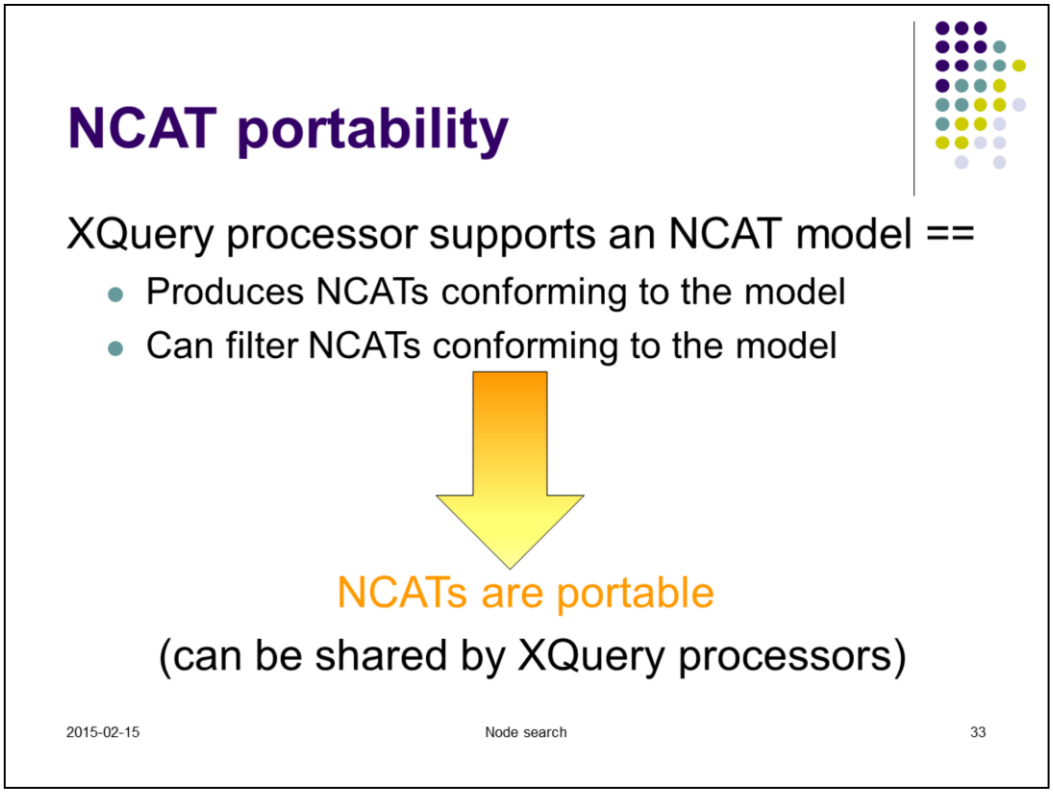
A p-model defines the external properties which a collection confers to its member nodes. The standard form of a p-model is a set of names associated with XQuery expressions and type declarations. The property value is then the value obtained by evaluating the expression in the context of the node in question. The model may also contain a wildcard denoting additional properties whose names and values are set arbitrarily.



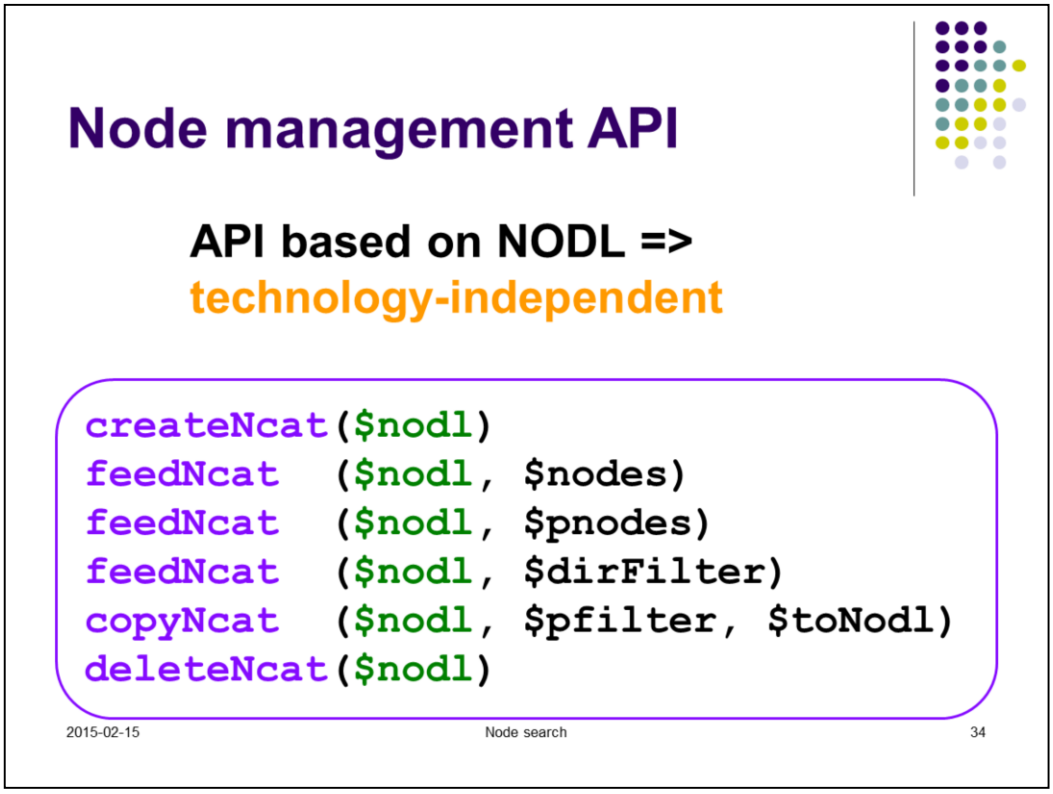
p-nodes can be implemented using different technologies, so the question arises how implementations of node search and collection maintenance can be guided in a unified and standardized way. The answer is the use of technology-specific NCAT models The term NCAT denotes the artifacts used to represent a p-collection – e.g. XML documents, a set of relational tables, or a NOSQL collection. For each supported technology, a generic NCAT model is defined. It specifies all details required to create, update and query an NCAT based on the respective technology. An NCAT model includes the schema of an NCAT model descriptor, an XML element supplying all required configuration parameters. A NODL contains exactly one NCAT model descriptor.



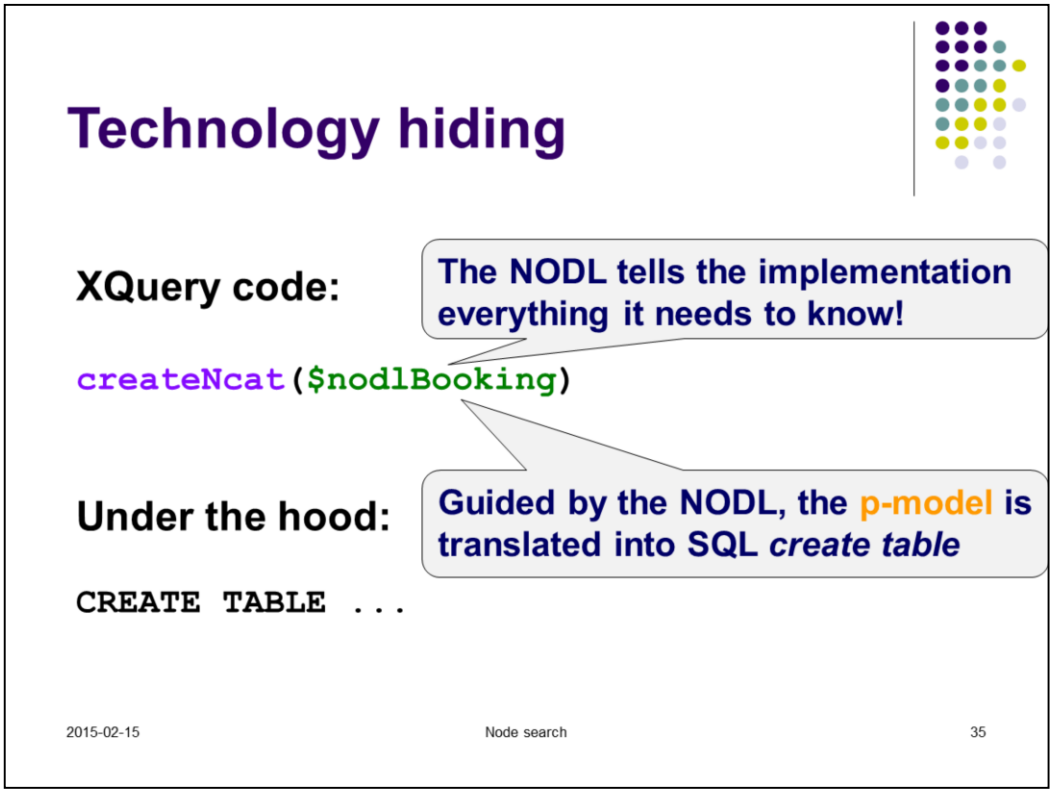
Currently, two NCAT models have been defined: an XML NCAT model for NCATs based on an XML document. And a SQL NCAT model for NCATs based on a relational database. Hopefully, this is only a start – and further NCAT models will follow.



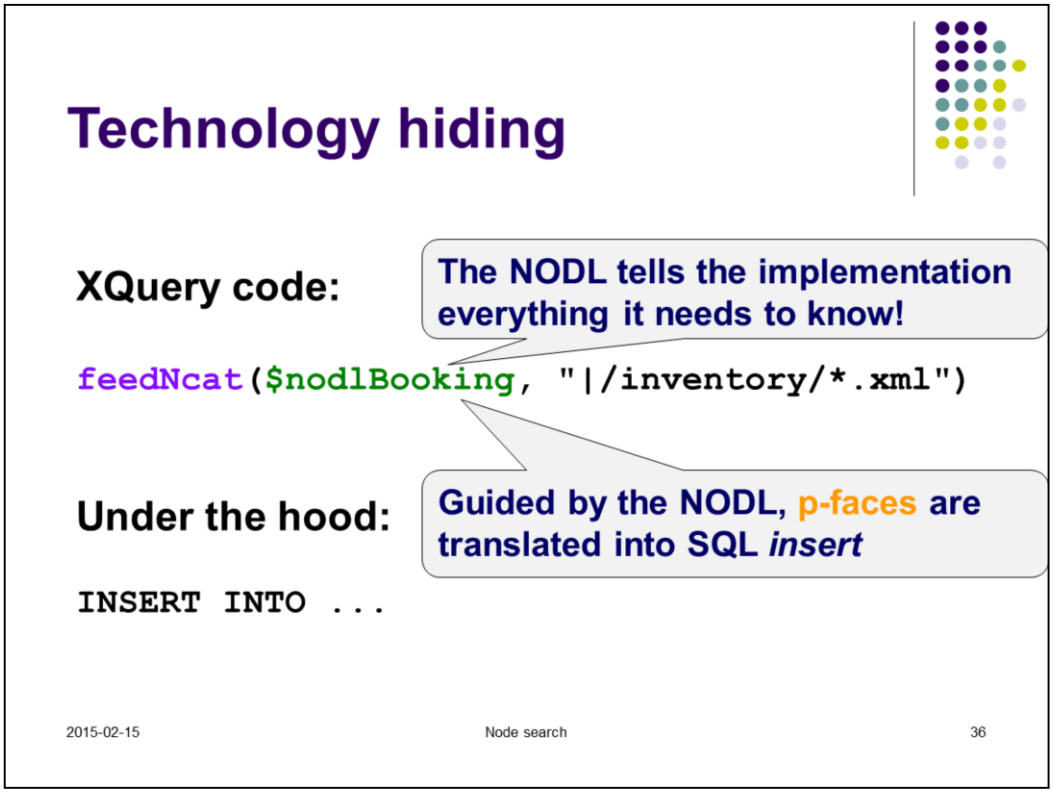
NCAT models are the ground on which portability is built.



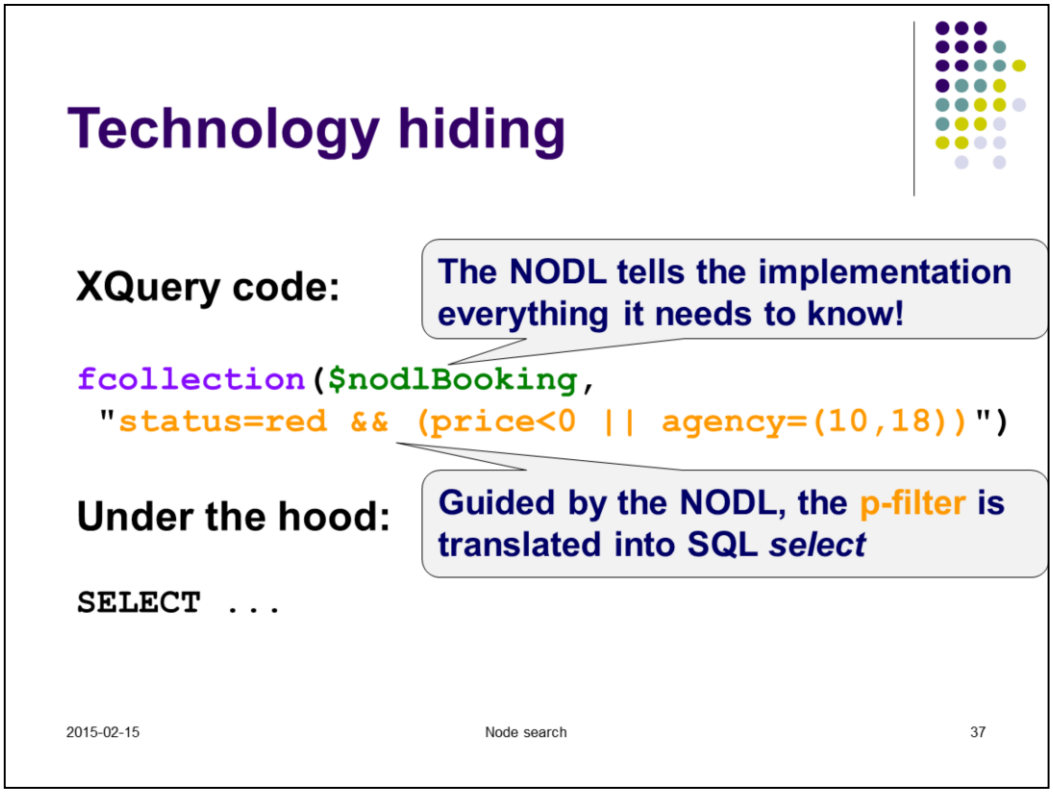
The node management API supports the operations of creating, updating and deleting an NCAT. Note that the API is technology-independent. Every function receives a NODL document as function parameter, and the NODL tells the implementation everything it needs to know, including the choice of technology and any technology-related configuration parameters.



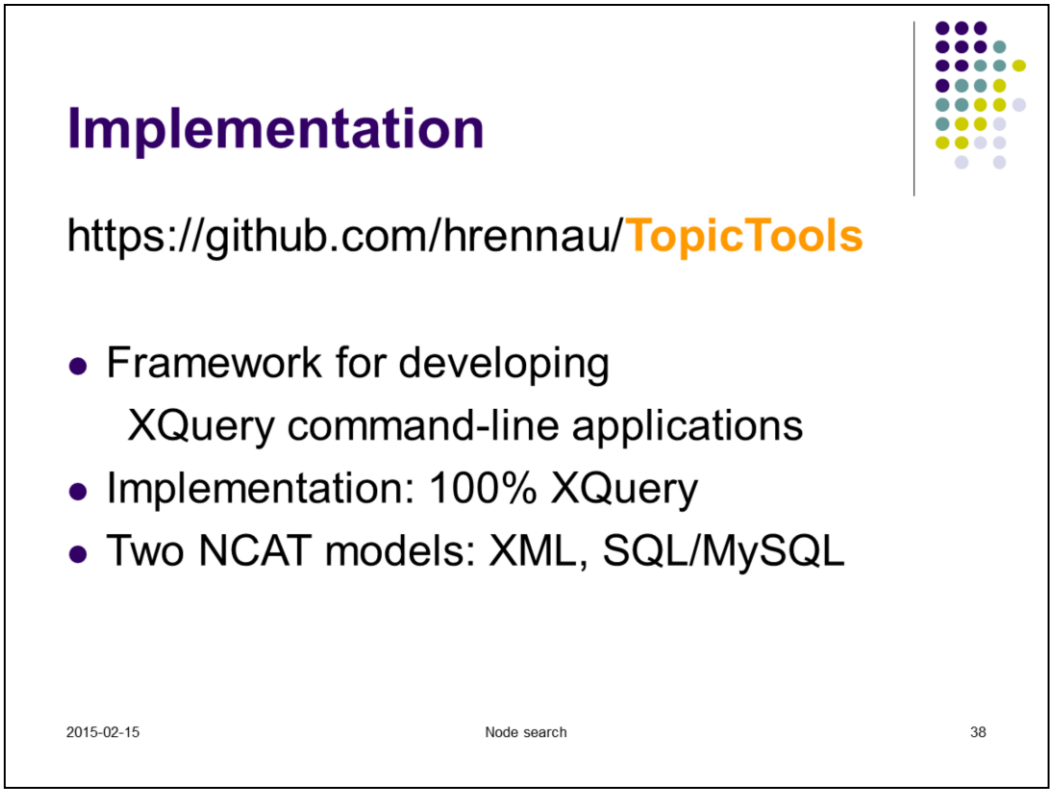
Let us take a look at technology hiding. Suppose the NODL at hand is SQL-based. When I create an NCAT, the API function translates the p-model into SQL create table statements.



When I feed the collection with documents, the API function inspects the documents, constructs their p-faces and translates each p-face into a set of SQL INSERT commands.

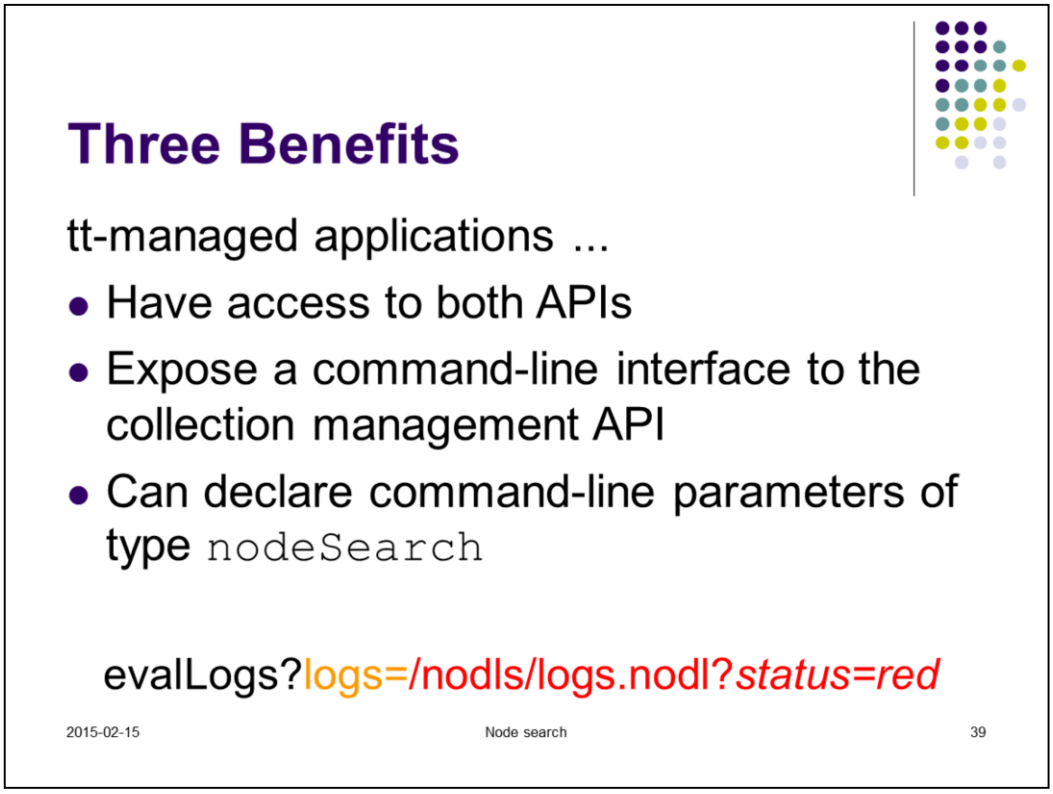


When I filter the collection, the API function translates the p-filter into a SQL SELECT command and resolves the node descriptors of the selected p-nodes to XML document nodes.

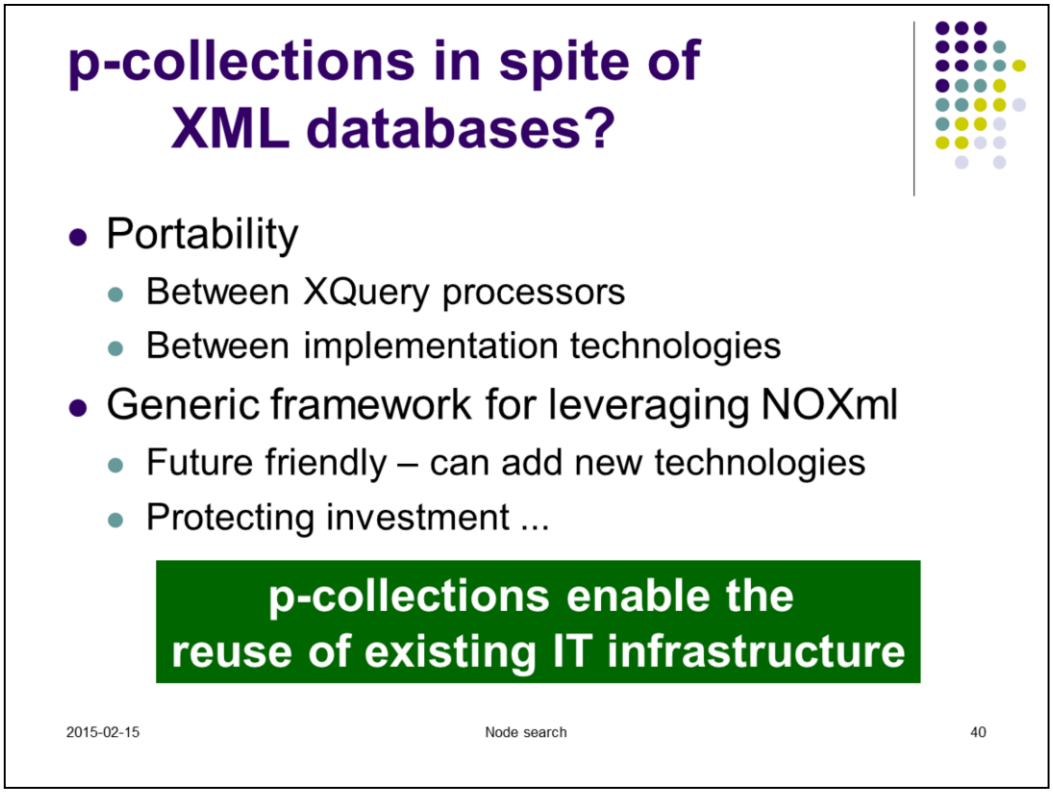


A first implementation of the APIs for node search and collection management is available via github. The implementation is an integral part of TopicTools, which is a framework for creating XQuery command-line tools. TopicTools supports your application via code generation and a rich infrastructure connecting your code to user input, validating and augmenting user input as guided by annotations. The current version supports two NCAT models, XML and SQL.

The implementation is 100% XQuery-based, that is, it does not involve any changes of XQuery processor code. This entails the limitation that SQL collections can only be used with XQuery processors which offer extension functions for executing SQL commands. Currently, SQL support is therefore restricted to BaseX. Support for other processors offering SQL support will be added, as soon as user interest becomes apparent.

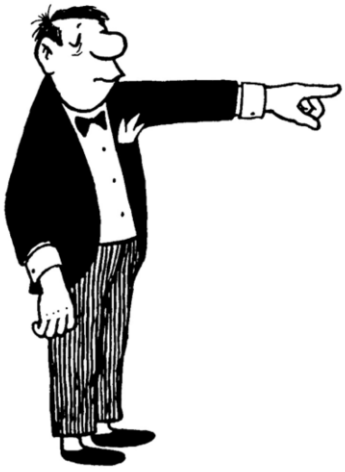


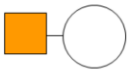
For the developer of a TopicTools-based application, support for p-collections has three aspects. First, your application code has access to the APIs described in this presentation. Second, your application inherits a command-line interface to the collection management API. Third, the application can use annotations in order to declare command-line parameters of various types, among them the type nodeSearch. In summary, you enable your users to manage p-collections and to supply application input as filtered collections.



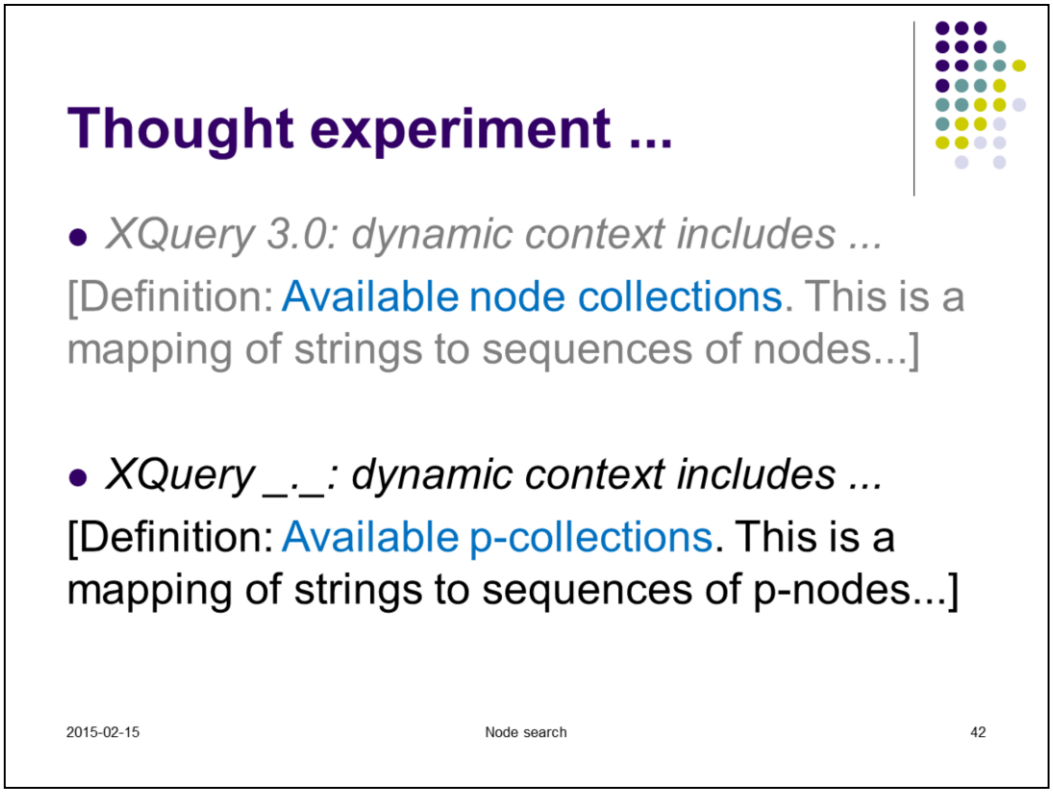
XML databases offer a node search which is incomparably richer than what p-collections can offer. Is there any point in p-collections?

Recommended!





Is it imaginable to include p-collections in standard XQuery?



Apart from extending the XDM by p-nodes, the main change required would be the extension of the dynamic context by a further component: „available p-collections“. Here it poses with its older sister, „available node collections“.

Thank you, W3C!