

Node search

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

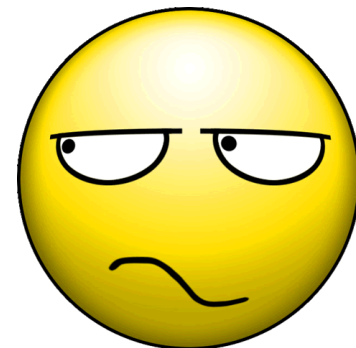
Hans-Jürgen Rennau, Traveltainment GmbH
February 15, 2015





Node search - agenda

- Problem definition
- Key idea
- Elaboration



The problem





The alchemy of XPath ...

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

sum of all accessible XML resources

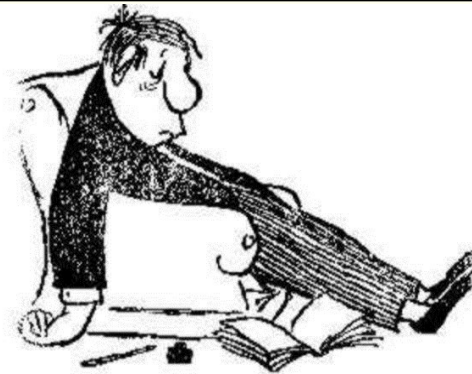
=

**single space of information
(infospace)**



REALITY FIGHTING BACK

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



**out of
Memory**



Consider this trick ...

Query:

```
doc('logs.xml')//doc[@status = 'red']/  
doc(@uri)/booking/agID
```

Catalog:

```
<logs>  
  <doc status="green" uri="/logs/b0101.xml"/>  
  <doc status="red" uri="/logs/b0102.xml"/>  
  <doc status="green" uri="/logs/b0103.xml"/>  
  <doc status="red" uri="/logs/b0104.xml"/>  
  <doc status="green" uri="/logs/b0105.xml"/>  
</logs>
```

The secret of XPath

node

surface:

node properties

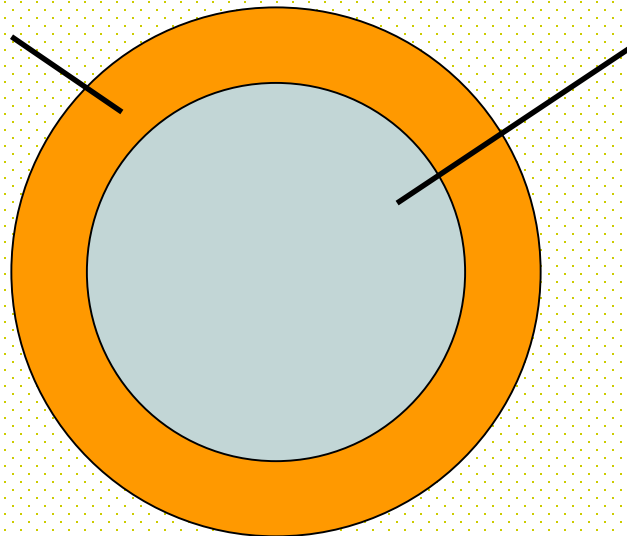
node-name

children

parent

attributes

...



core:
Information

characters





XML navigation vs. search

- Navigation
node(s)

a/b/c



node(s)



- Search
query

fn:doc(\$uri)



node(s)

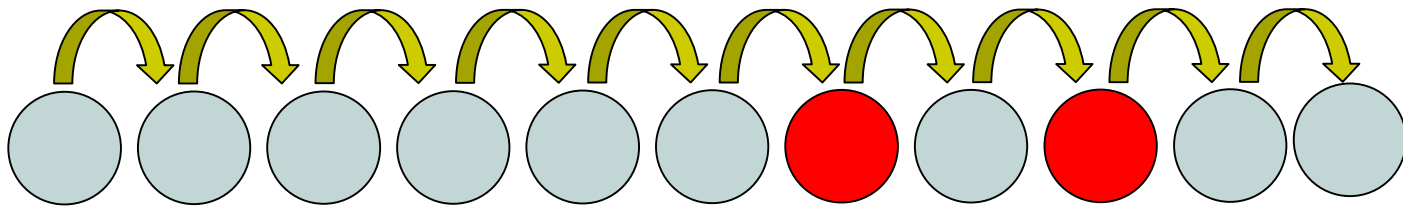
fn:collection(\$uri)



The problem

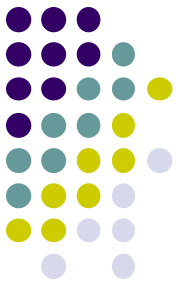


XPath navigation is based on **node properties**

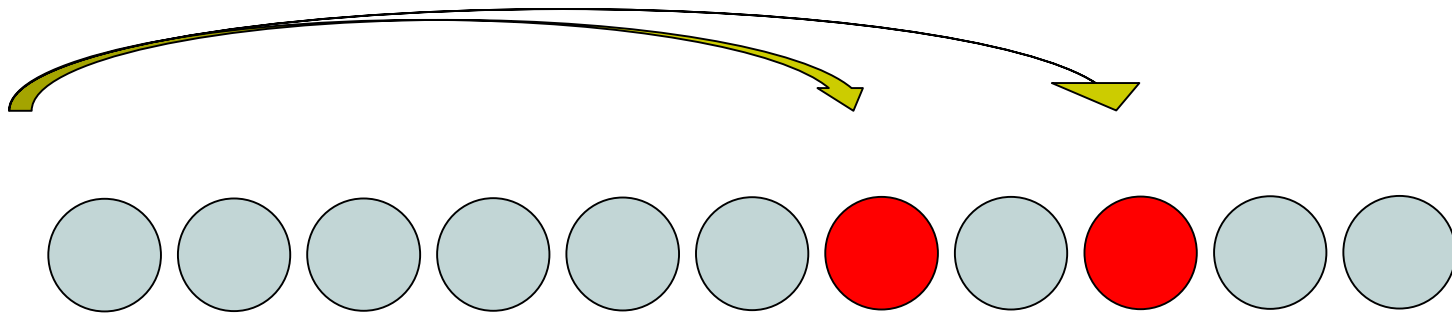


Access to node properties requires **node construction**

The goal

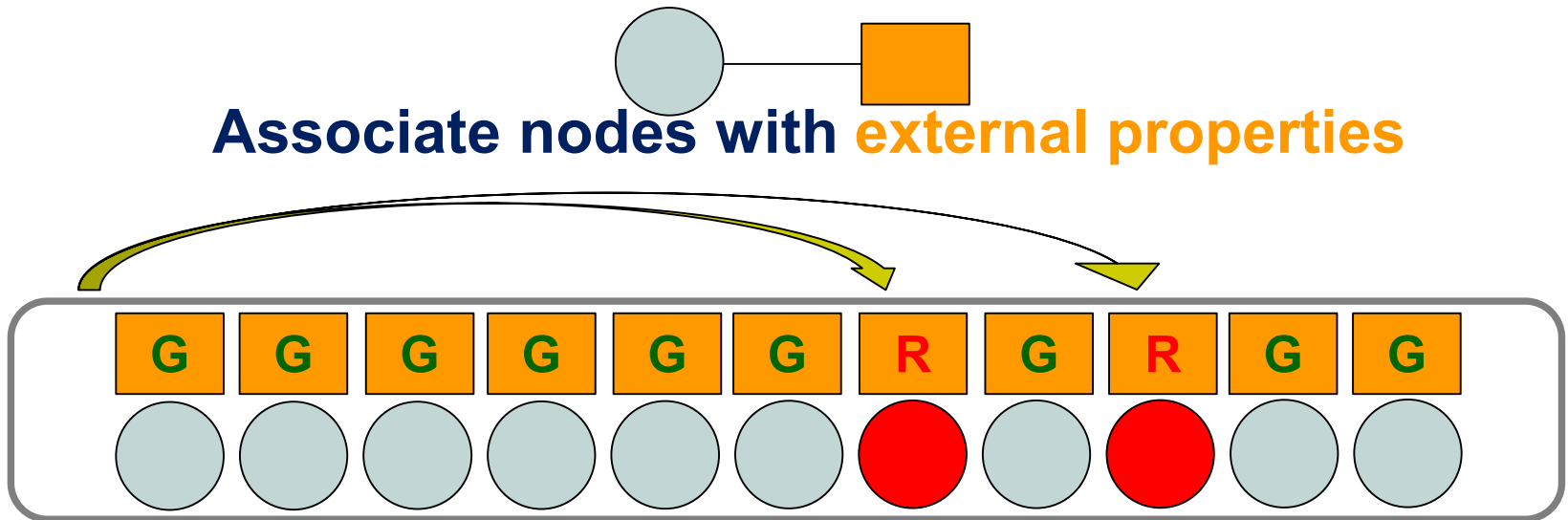


To complement XPath navigation with a **node search**



**which does not require node construction
(and is portable)**

A solution



Node search = filtering nodes by external properties

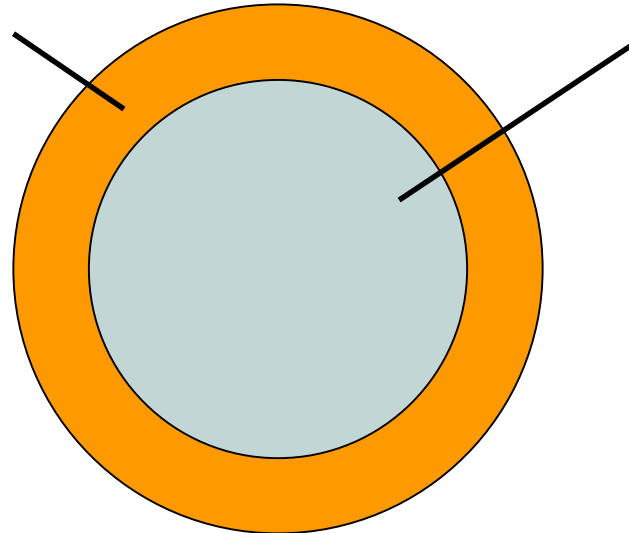


So add ...

node

surface:

node properties



core:

Information

characters

... a second, detached surface!



node

surface:

node properties

status=g
agID=10
price=79

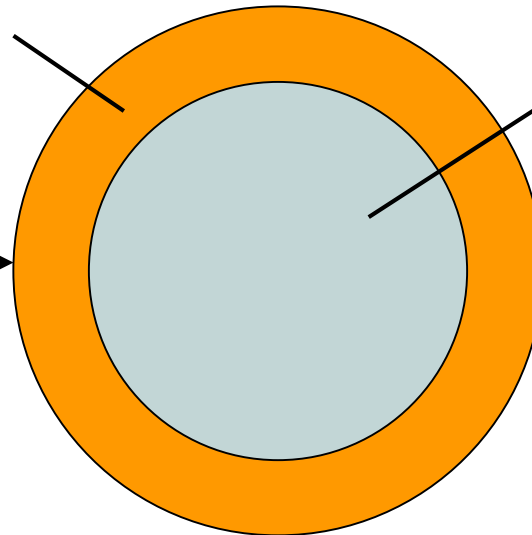
surface #2:

external node properties

core:

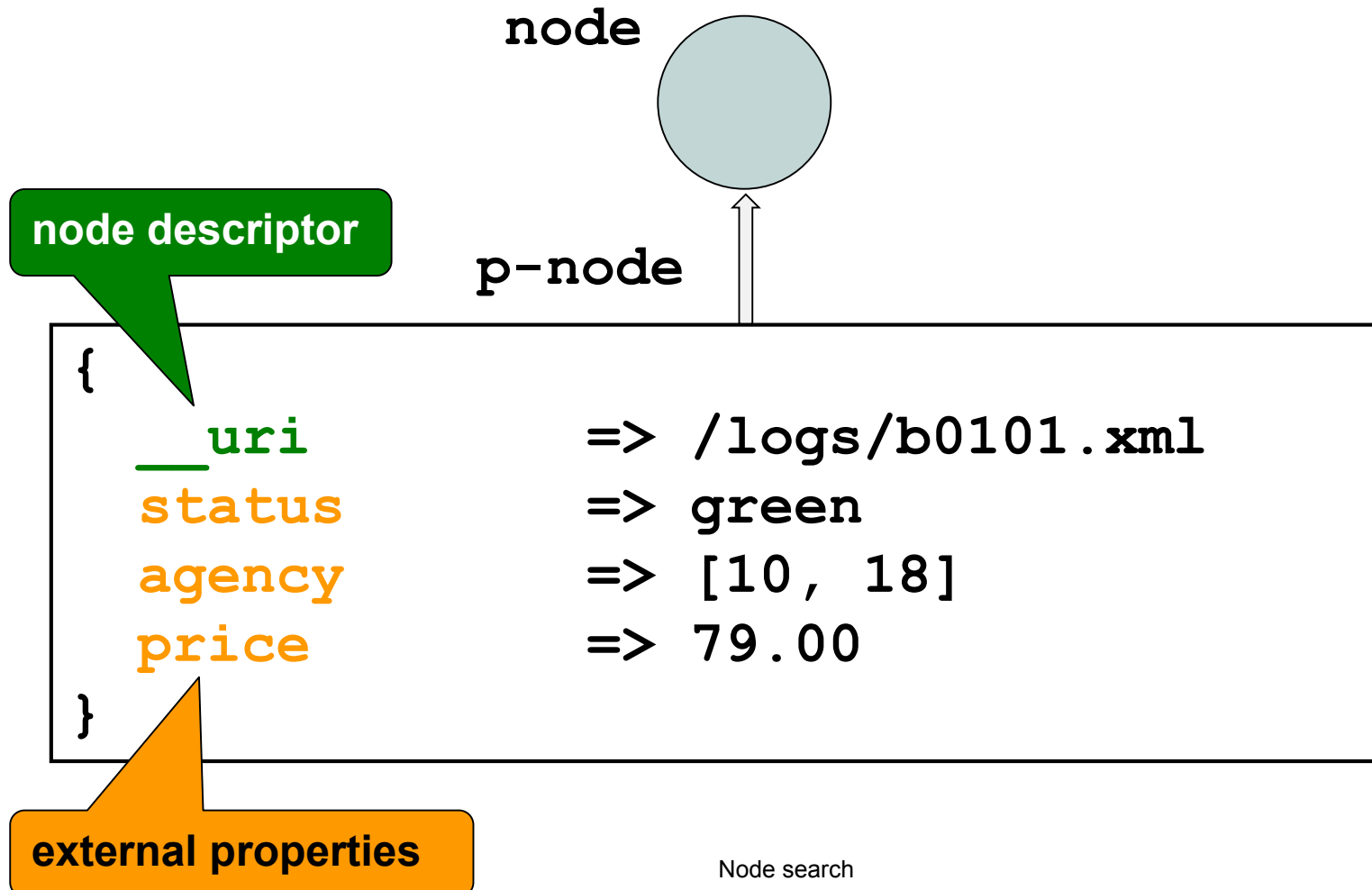
Information

characters

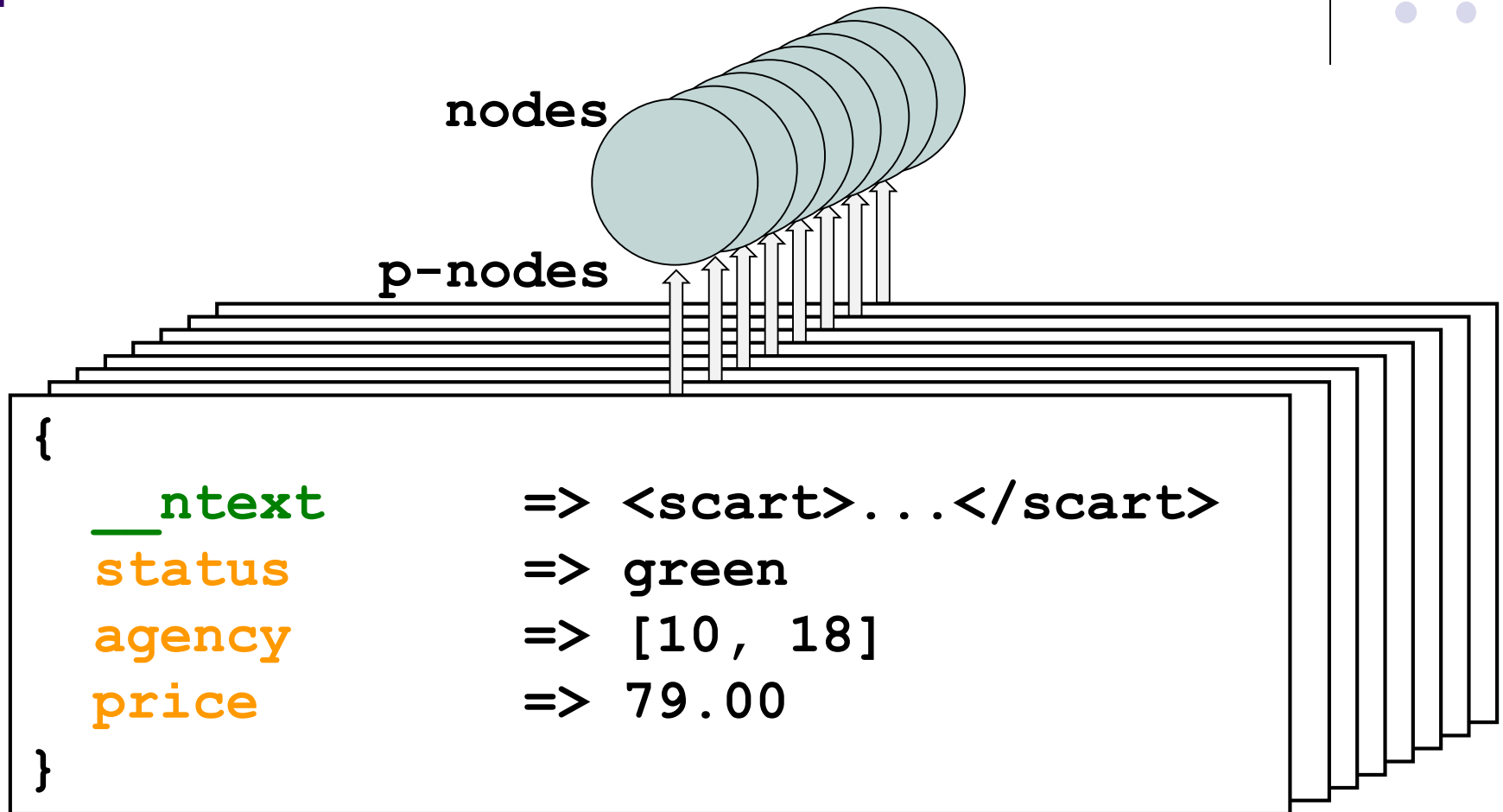




nodes & p-nodes

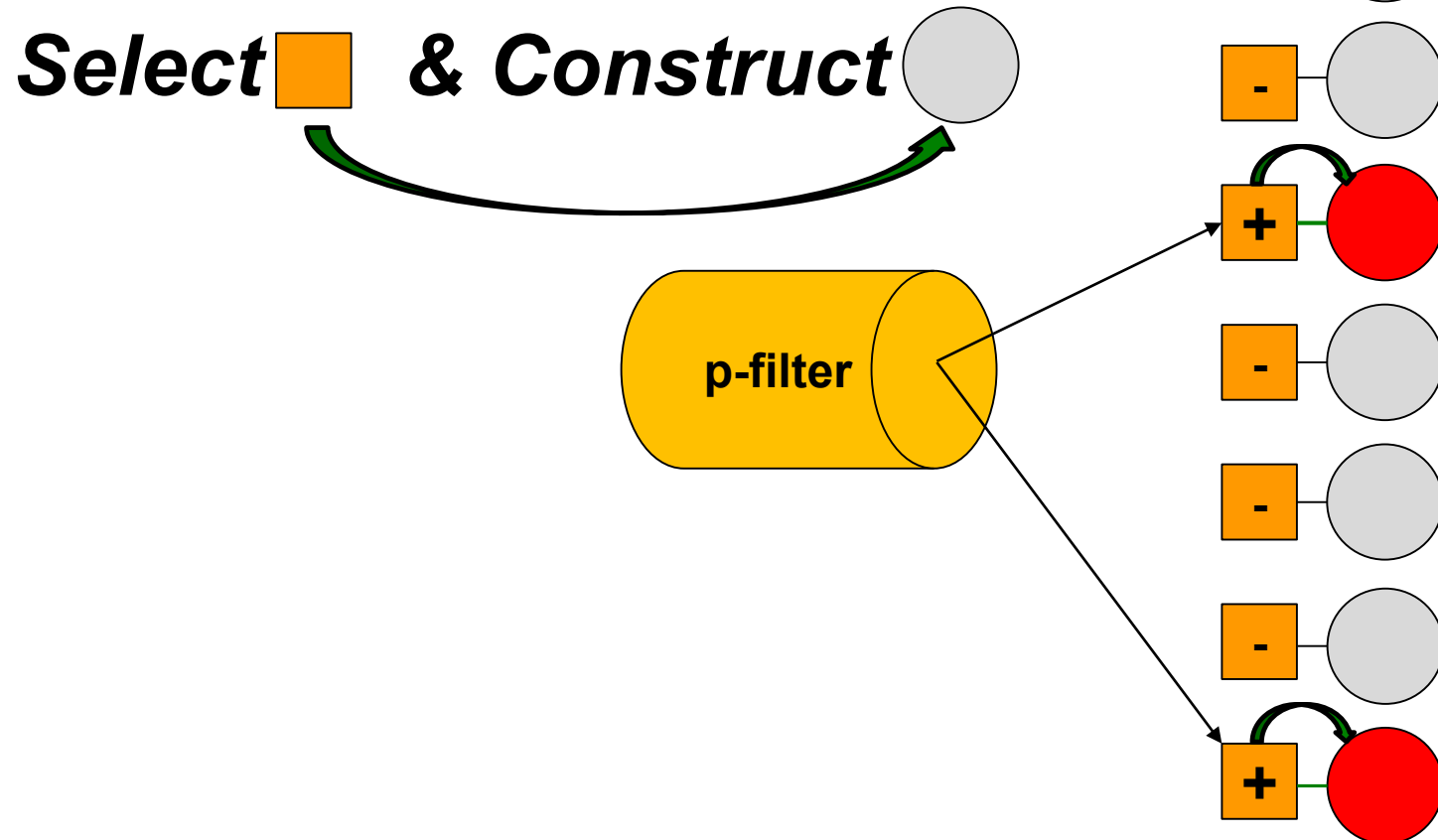


p-collection

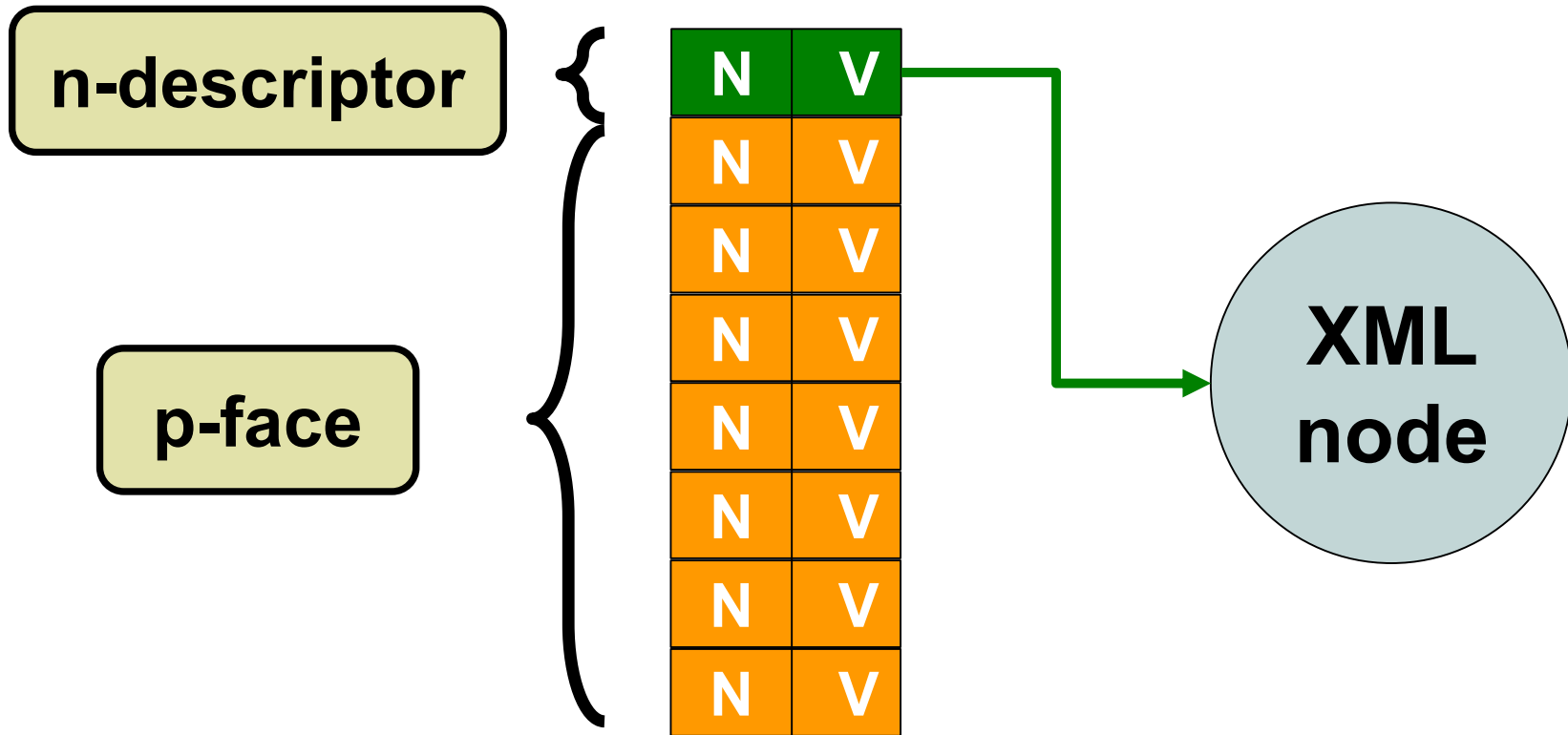




Node search – the principle



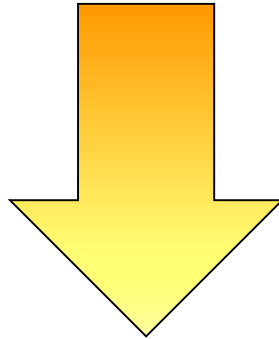
p-node = N/V pairs





Node search - is NOXml !

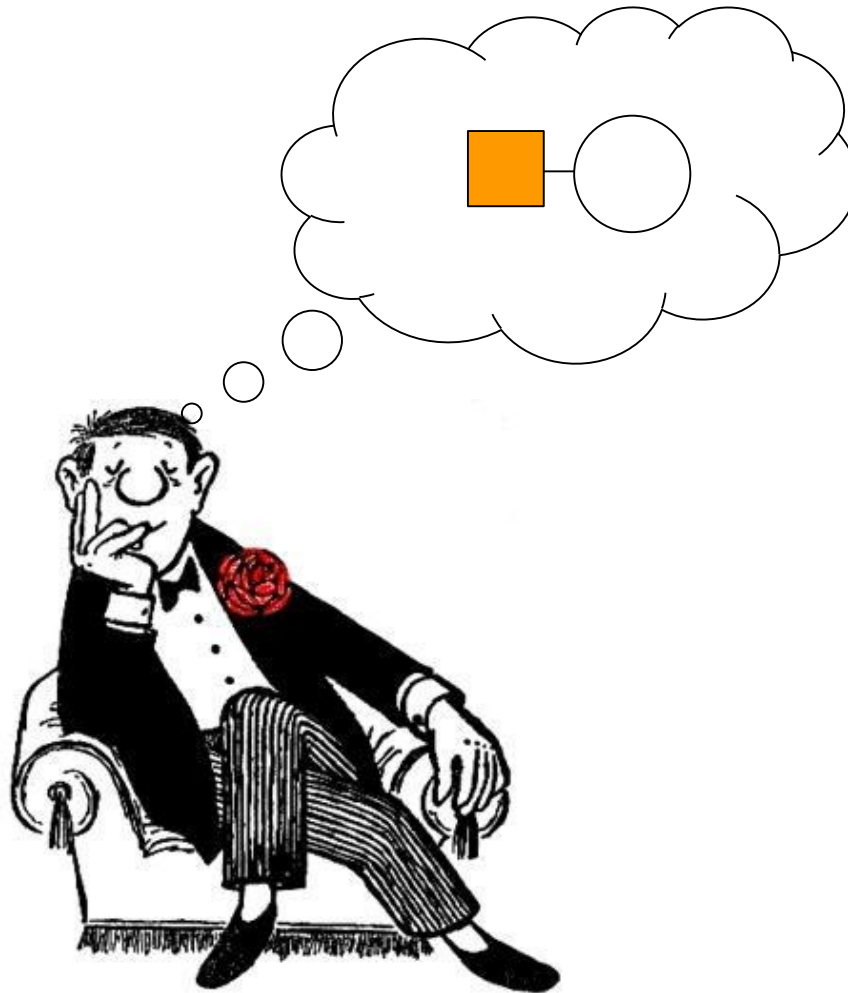
p-node = name/value pairs



storage & retrieval = **NOXml**

XML, SQL, NOSQL, ...

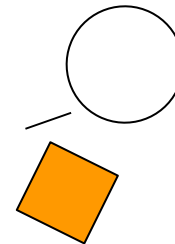
The idea.



The elaboration



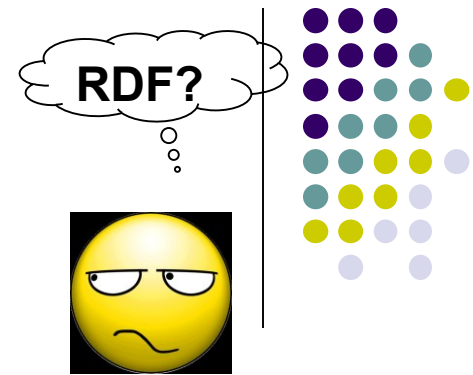
- grammar of concepts
- models
 - p-face, p-model, p-filter
- APIs
 - node search
 - collection management





Grammar of concepts

- node search: {p-collection, p-filter}
- p-collection: (p-node)*
- p-node: {node-descriptor, p-face}
- node-descriptor string
- p-face: (external-property)*
- external-property: {name, (atomic-value)*}
- p-filter: {p-test, and, or, not}+
- p-test: {p-name, operator, test-val.}



External property

- Name: **NCName** *(or better QName?)*
- Value: sequence of **atomic values**

Analogy: XML attribute (QName, atomicValue*)

- Semantics:
 - The value of an XQuery-expression
 - *or* an arbitrary value (assigned during insertion)



p-face

- All **external properties** of a node
- Scoped to a particular p-collection

```
<pface>
  <property name='status' value='green' />
  <property name='agency'>
    <item>10</item>
    <item>21</item>
  </property>
</pface>
```



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))



Node search API

Example query:

```
fcollection("logs.nod1", "status=red") //agID
```

API:

```
fcollection ($pcolURI as xs:string?,  
             $pfilter as xs:string?)  
            as node () *
```

```
fcollection ($pcolURI as xs:string?,  
             $pfilter as element(pc:pfilter)?)  
            as node () *
```

p-collection management



Lemon curry?



WSDL

SOAP service

REST service

WADL

NODL

p-collection



More concepts

- **NODL** - description of a p-collection
- **p-model** - rules how to construct a p-face
- **NCAT** - artifacts representing a p-collection
- **NCAT model** - rules how to construct an NCAT

NODL



**Standardized description
enabling management & use of a p-collection**

```
<nodl>                                Collection name, URI, doc, ...  
  <collection>...  
  <pmodel>...    Property constructors  
  <ndesc>...     Node descriptor  
  <ncatModel>...  
</nodl>         Physical model of the collection
```



p-model

- Property **assignment rules**
- Standard: {name, type, XQuery-expression}*

```
<pmodel>
  <property name='status'  expr='//status'
            type='xs:string' />
  <property name='agency'  expr='//agID',
            type='xs:integer*' />
  <anyProperty/>
</pmodel>
```



NCAT model

Technology T1

NCAT Model

Descriptor XSD

NODL

Descriptor XML

Configuration parameters

- 1 conforms-to
- 2 identifies
- 3 provides



NCAT model descriptors

```
<xmlNcat documentURI="/a/b.ncat"  
asElems="*" />
```

```
<sqlNcat  
  rdbms="MySQL"  
  rdbmsVersion="5.6"  
  host="localhost" db="pcoll"  
  user="abc" password="infospace"  
/>
```

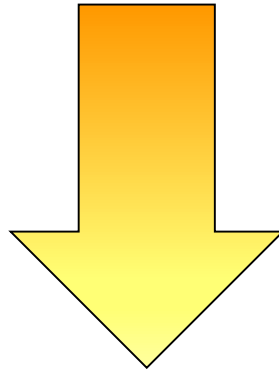
32



NCAT portability

XQuery processor supports an NCAT model ==

- Produces NCATs conforming to the model
- Can filter NCATs conforming to the model



NCATs are portable

(can be shared by XQuery processors)



Node management API

API based on NODL =>
technology-independent

```
createNcat ($nodl)  
feedNcat   ($nodl, $nodes)  
feedNcat   ($nodl, $pnodes)  
feedNcat   ($nodl, $dirFilter)  
copyNcat   ($nodl, $pfilter, $toNodl)  
deleteNcat ($nodl)
```



Technology hiding

XQuery code:

```
createNcat($nodlBooking)
```

The NODL tells the implementation everything it needs to know!

Under the hood:

```
CREATE TABLE ...
```

Guided by the NODL, the **p-model** is translated into SQL *create table*



Technology hiding

XQuery code:

```
feedNcat($nodlBooking, "|/inventory/*.xml")
```

The NODL tells the implementation everything it needs to know!

Under the hood:

INSERT INTO ...

Guided by the NODL, **p-faces** are translated into SQL *insert*



Technology hiding

XQuery code:

The NODL tells the implementation everything it needs to know!

```
fcollection($nodlBooking,  
  "status=red && (price<0 || agency=(10,18))")
```

Under the hood:

Guided by the NODL, the **p-filter** is translated into SQL *select*

```
SELECT ...
```



Implementation

<https://github.com/hrennau/TopicTools>

- Framework for developing XQuery command-line applications
- Implementation: 100% XQuery
- Two NCAT models: XML, SQL/MySQL



Three Benefits

tt-managed applications ...

- Have access to both APIs
- Expose a command-line interface to the collection management API
- Can declare command-line parameters of type `nodeSearch`

`evalLogs?logs=/nodls/logs.nodl?status=red`

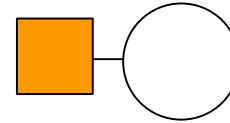
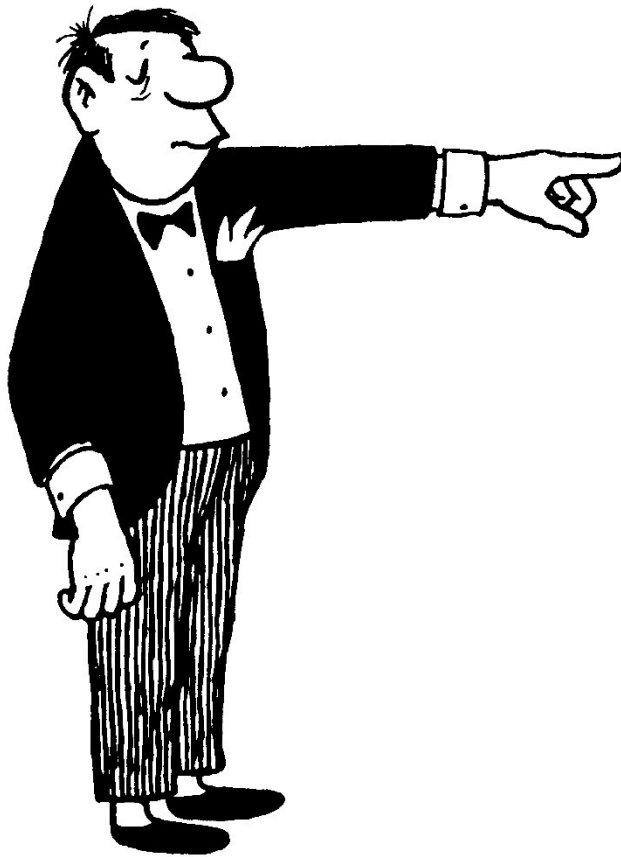
p-collections in spite of XML databases?



- Portability
 - Between XQuery processors
 - Between implementation technologies
- Generic framework for leveraging NOXml
 - Future friendly – can add new technologies
 - Protecting investment ...

**p-collections enable the
reuse of existing IT infrastructure**

Recommended!





Thought experiment ...

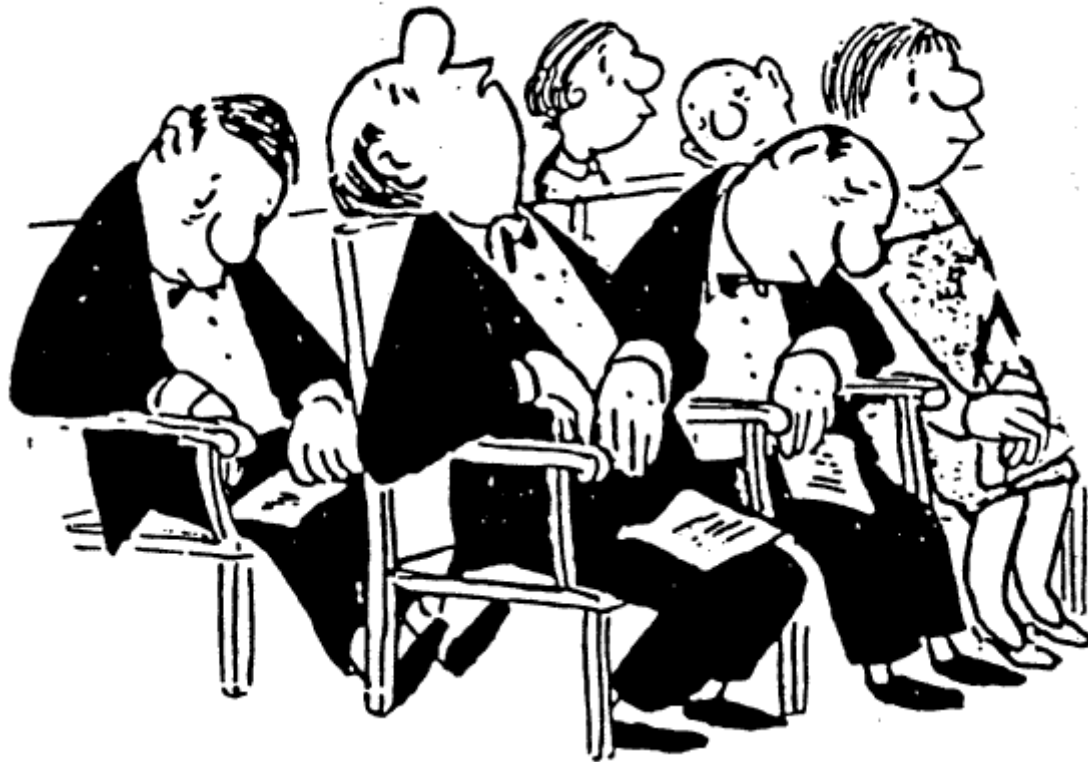
- *XQuery 3.0: dynamic context includes ...*

[Definition: **Available node collections**. This is a mapping of strings to sequences of nodes...]

- *XQuery _._: dynamic context includes ...*

[Definition: **Available p-collections**. This is a mapping of strings to sequences of p-nodes...]

Thank you, W3C!



Loriot