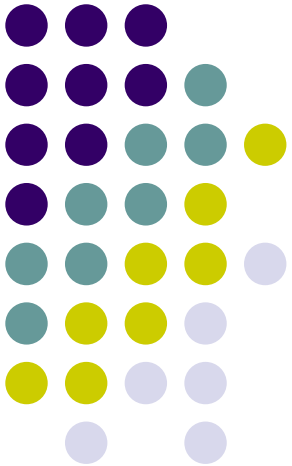
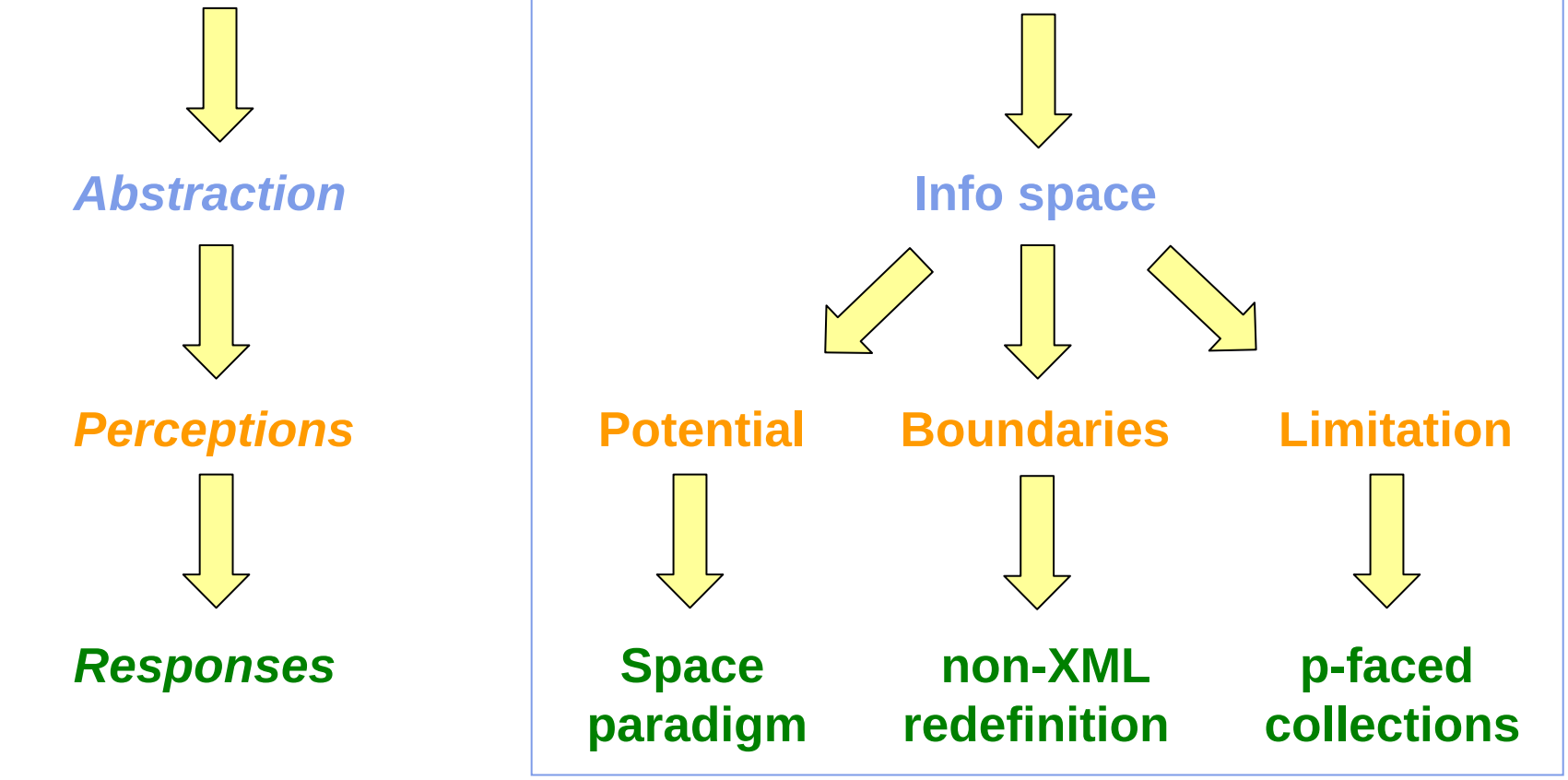
The XML info space

Perceiving the space,
pushing its boundaries,
addressing its limitations

Hans-Jürgen Rennau, Traveltainment GmbH
Presented at Balisage 2013, August 7



Roadmap

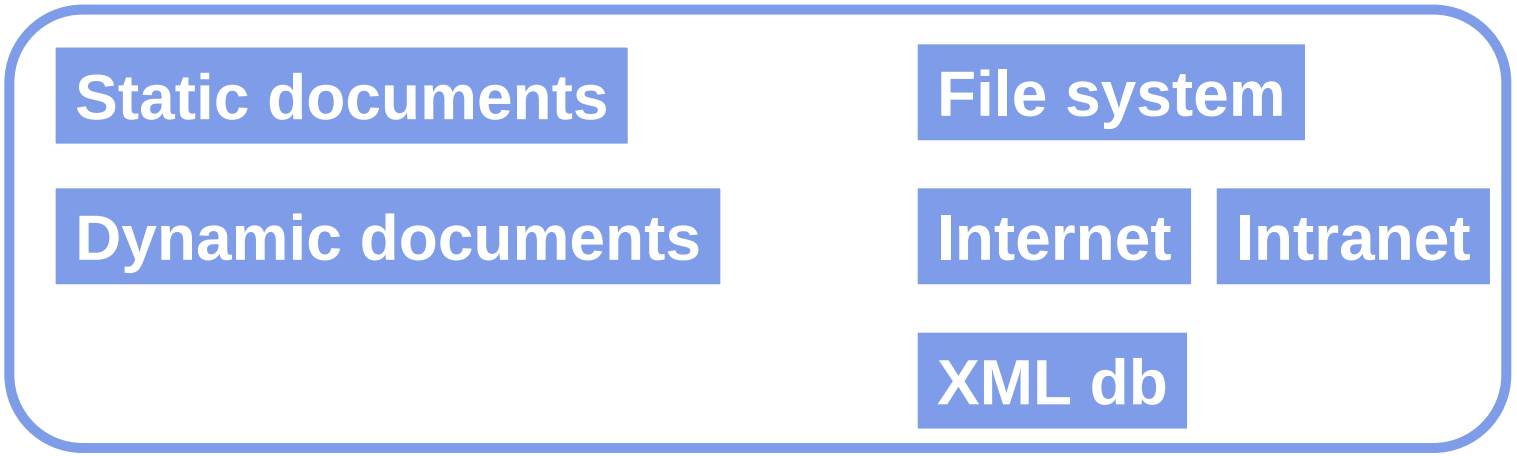

Facts & Experience
Abstraction
Perceptions
Responses
XML technology
Info space
Potential
Boundaries
Limitation
Space paradigm
non-XML redefinition
p-faced collections

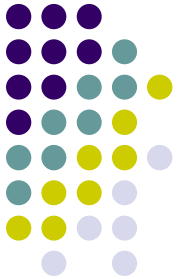
Facts !

Facts (1)

- XML document = node tree
- fn:doc(u1) = node tree at URI u1
- u1, u2, u3, … = node forest

One single node forest





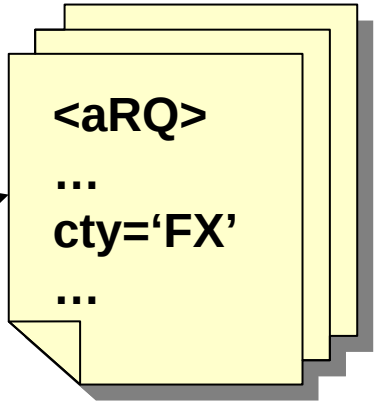
Facts (2)

- Uniform content model
  - Content = node
  - Defined by a fixed set of node properties
- Uniform location model
  - Location = node
  - Expressed by URI & node relationships
- Uniform navigation model
  - Start, End = nodes
  - Expressed by URI & node relationships
- Uniform processing model
  - In, out = sequence of (nodes | atoms)

Coding experience

Task

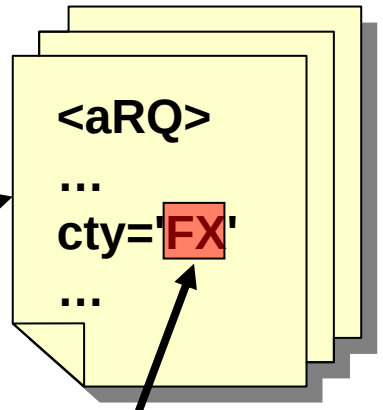
```
<projects>
  <project id="p1">
    <xsd>
      <aRQ href="/user/xsds/aRQ.xsd"/>
      <aRS href="/user/xsds/bRS.xsd"/>
    </xsd>
    <msgs>
      <aRQ href="/user/msgs/aRQ.xml"/>
      <aRS href="/user/msgs/aRS.xml"/>
      <aER href="/user/msgs/aER.xml"/>
    </msgs>
  </project>
  <project id="p2">
    …
</projects>
```



Task:
Check all messages of project "p1" for invalid country codes.
(Valid codes are provided by a doc.)

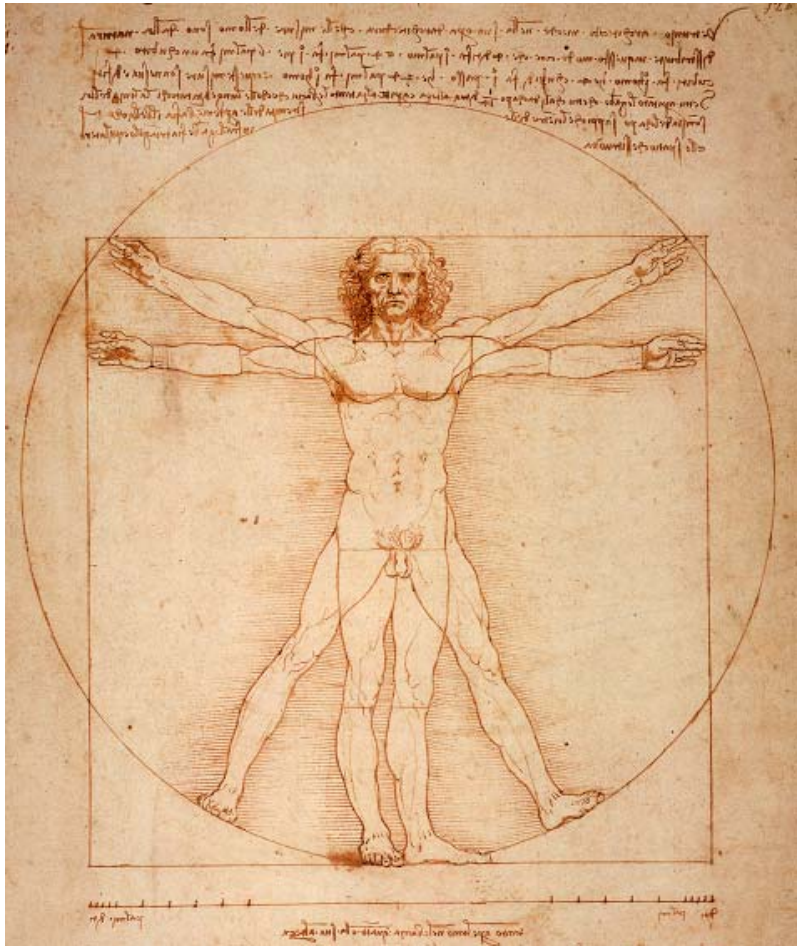
Solution

```
<projects>
  <project id="p1">
    <xsd>
      <aRQ href="/user/xsds/aRQ.xsd"/>
      <aRS href="/user/xsds/bRS.xsd"/>
    </xsd>
    <msgs>
      <aRQ href="/user/msgs/aRQ.xml"/>
      <aRS href="/user/msgs/aRS.xml"/>
    </msgs>
  </project>
  <project id="p2">
</projects>
```




```
doc('p.xml') //
 project [@id='p1'] / msgs // @href /
 doc(.) // @cty /
    [not(. = doc('c.xml' ) // code)]
```

Abstraction



The info space

Node =

uniform point of content & location

Sum of nodes =

homogenous space of located information

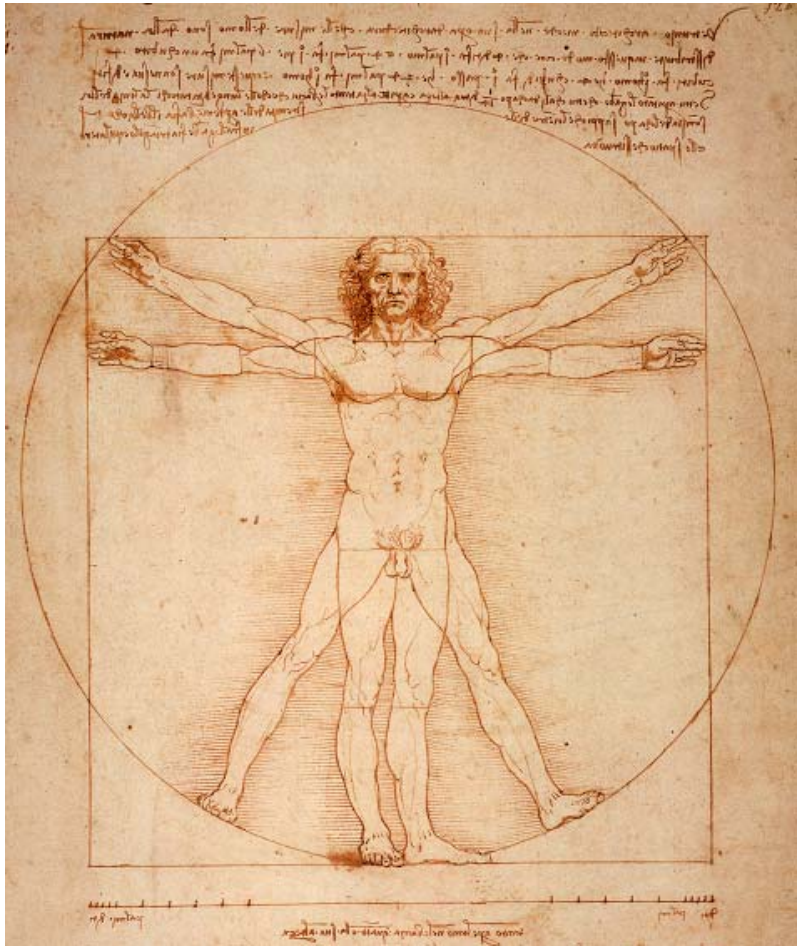
[Definition] info space =

sum total of all accessible XML resources

Unified …

content

location



processing

navigation

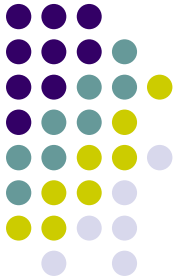
Specs & the info space

- The specs define nodes
- Nodes imply the *images “tree”, “space”*
- Image *“tree”* enables definition “axis”
- Image *“space”* enables definition “…”?

Tree image ➔ foundation of the navigation model

Space image ➔ extension of the navigation model?

Three key perceptions



Awareness of an info space enables three key perceptions:

- The space - its potential (its usefulness)
- The space - its boundaries
- The space - its limitations

The potential

Space Paradigm

REST

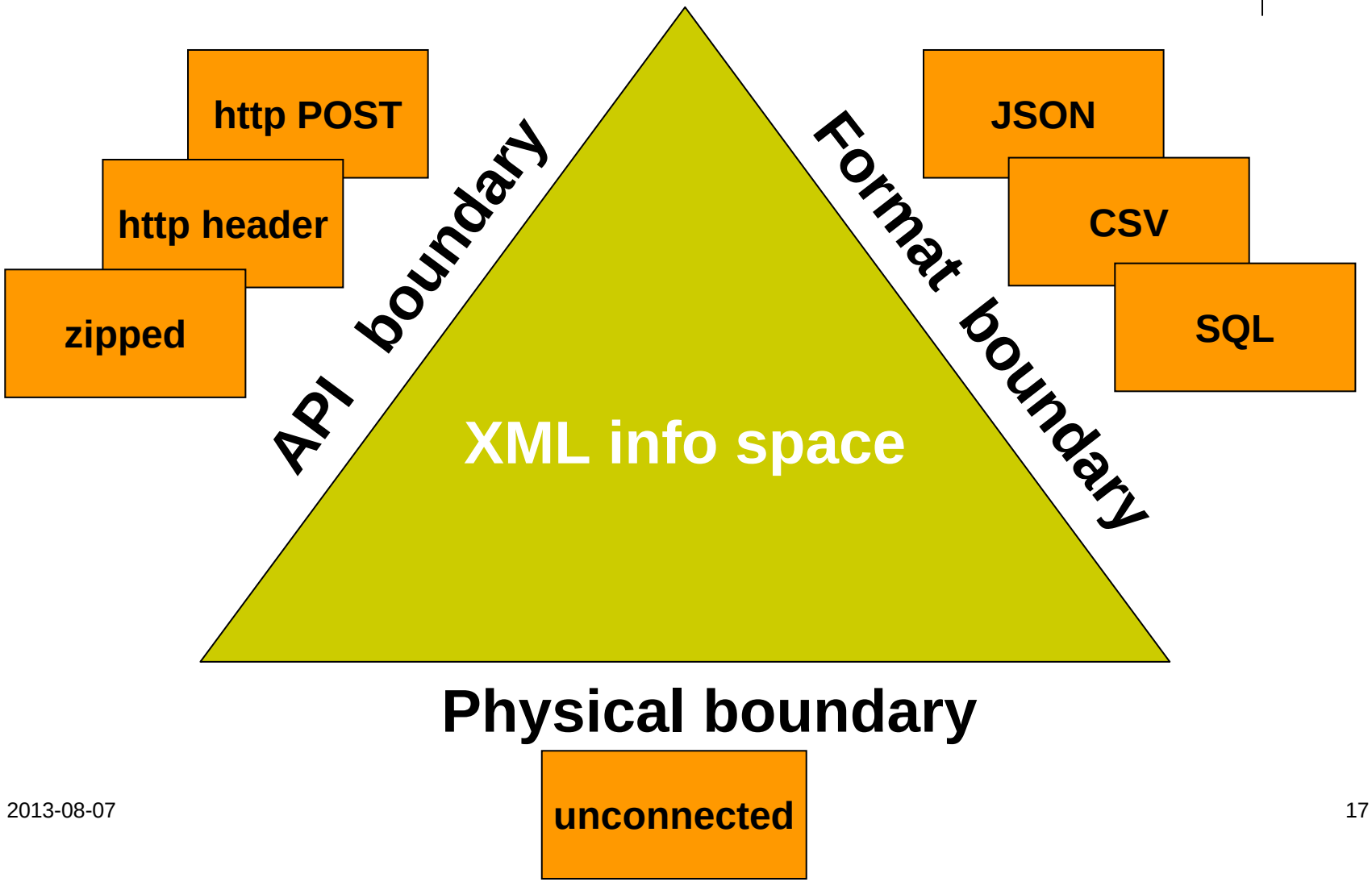
Information … into the space!

Evaluation … within the space!

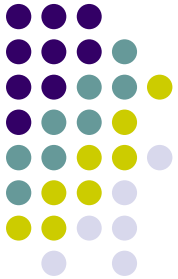
XQuery / XSLT

Pushing the boundaries!

Three boundaries



The API boundary



Add standard functions accessing documents.

For example:

- fn:httpDoc

  httpDoc ($uri, $username, $password, $headerFields, $body)

- fn:zipDocs

  zipDoc ($uri, $directories, $namePatterns)

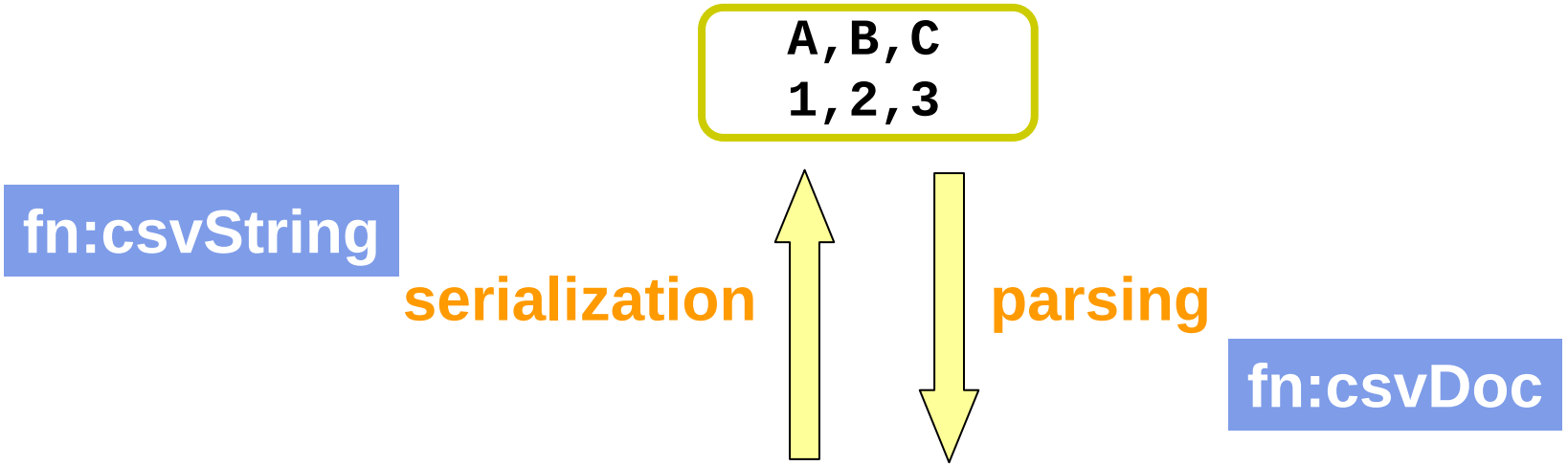
The format boundary

XML syntax = representation of a node tree

data syntax = representation of a node tree

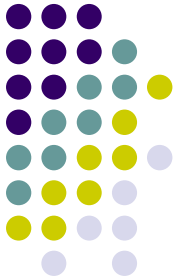
Redefinition of non-XML

Simple example: CSV, redefined




```
<table xmlns="http://www.w3.org/infospace/csv">
   <row><col>A</col><col>B</col><col>C</col></row>
   <row><col>1</col><col>2</col><col>3</col></row>
</table>
```

Meta-model: format redefinition



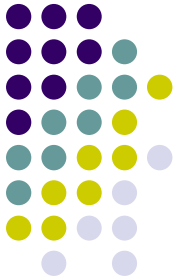
Given a non-XML format F:

Rule sets

R1	Mapping rules: F => node tree
R2	Mapping rules: node tree => F
R3	Rules: *can* a given node tree be mapped to F*?*

Constraints

C1	For any format instance G: R1(G) satisfies R3
C2	Node tree N satisfies R3 ➔ R1(R2(N)) is deep-equal to N

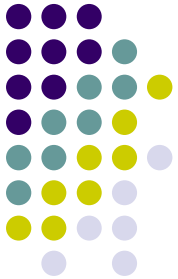


Types of format redefinition

- Vocabulary based
  - Format instances can be captured by a format-specific vocabulary
  - *Examples: CSV, CSS, SQL, Java-Properties*

- Structure based
  - Node tree structure reflects format instance structure
  - *Example: JSON*

JSON & the info space



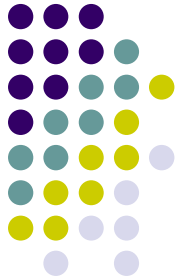
- JSONiq
  - integrates JSON into XQuery…
  - but does *not* model JSON as node trees

- Mapping approaches
  - Map JSON to an XML document …
  - but *add nodes* (e.g. @name) to “remember” the source JSON

- The info space approach:
  - JSON = node tree
  - But the redefinition is problematic – JSON names are strings !
    Redefinition requires an extended node model :
    new node properties [key], [model]

The limitations

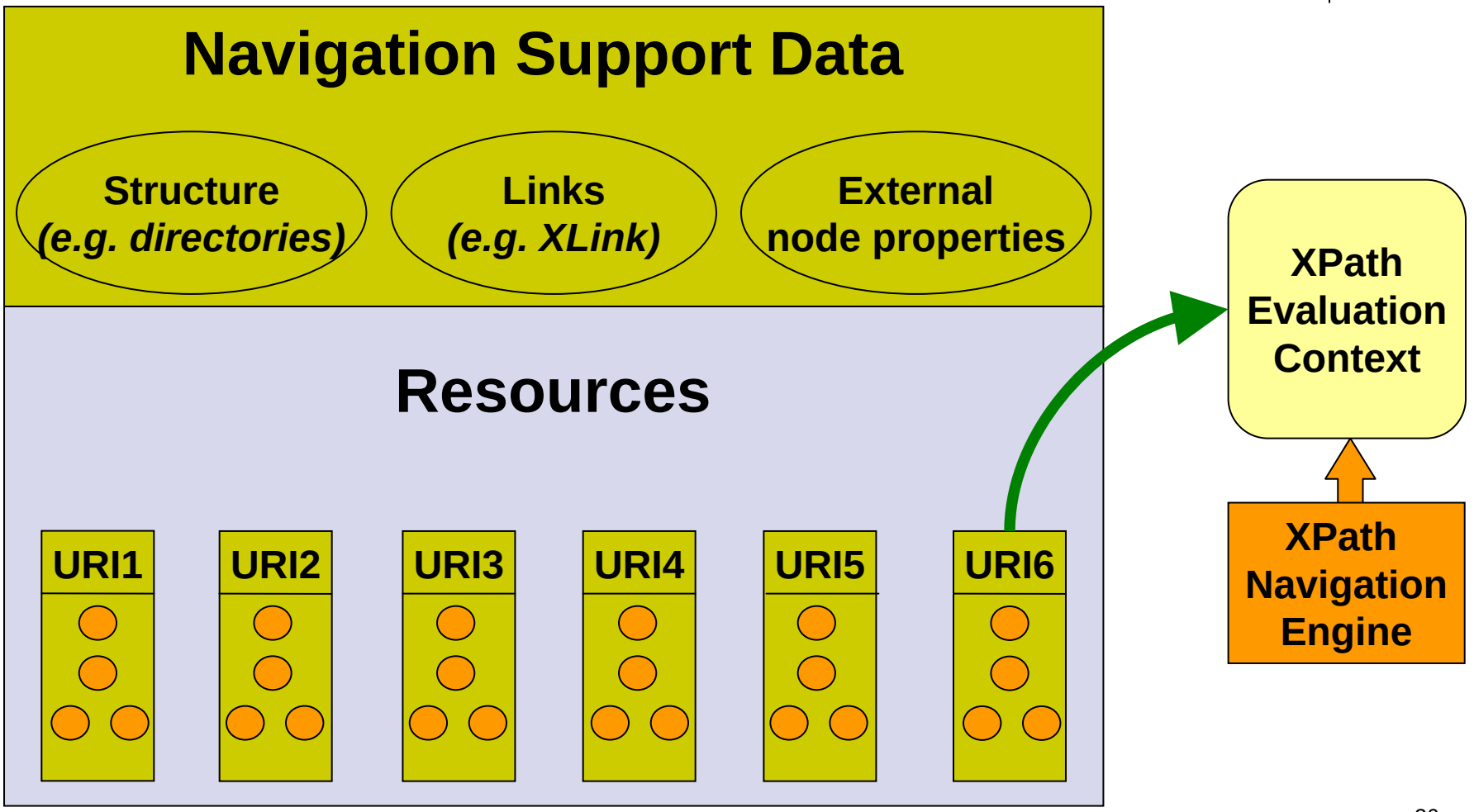
Unstructured set of trees



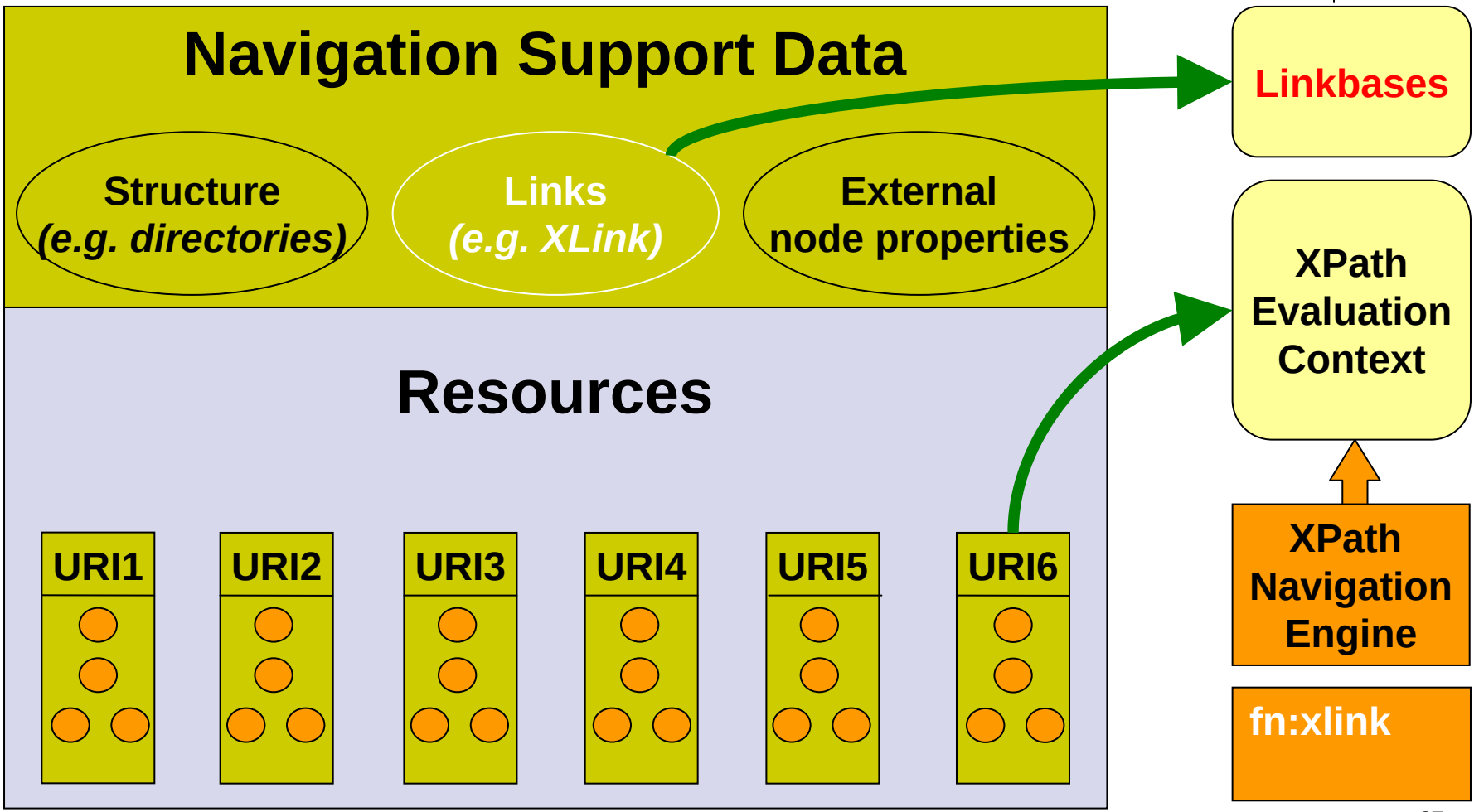
- XPath navigation
  - strong - within-document (path expressions)
  - weak - between-document (fn:doc, fn:collection)

- Info space structure
  - node tree = highly structured set of nodes
  - info space = unordered collection of trees

Navigation support data



XLink data


Navigation Support Data
Structure
(e.g. directories)
Links
(e.g. XLink)
External
node properties
Resources
URI1
URI2
URI3
URI4
URI5
URI6
Linkbases
XPath
Evaluation
Context
XPath
Navigation
Engine
fn:xlink

XLink per XPath

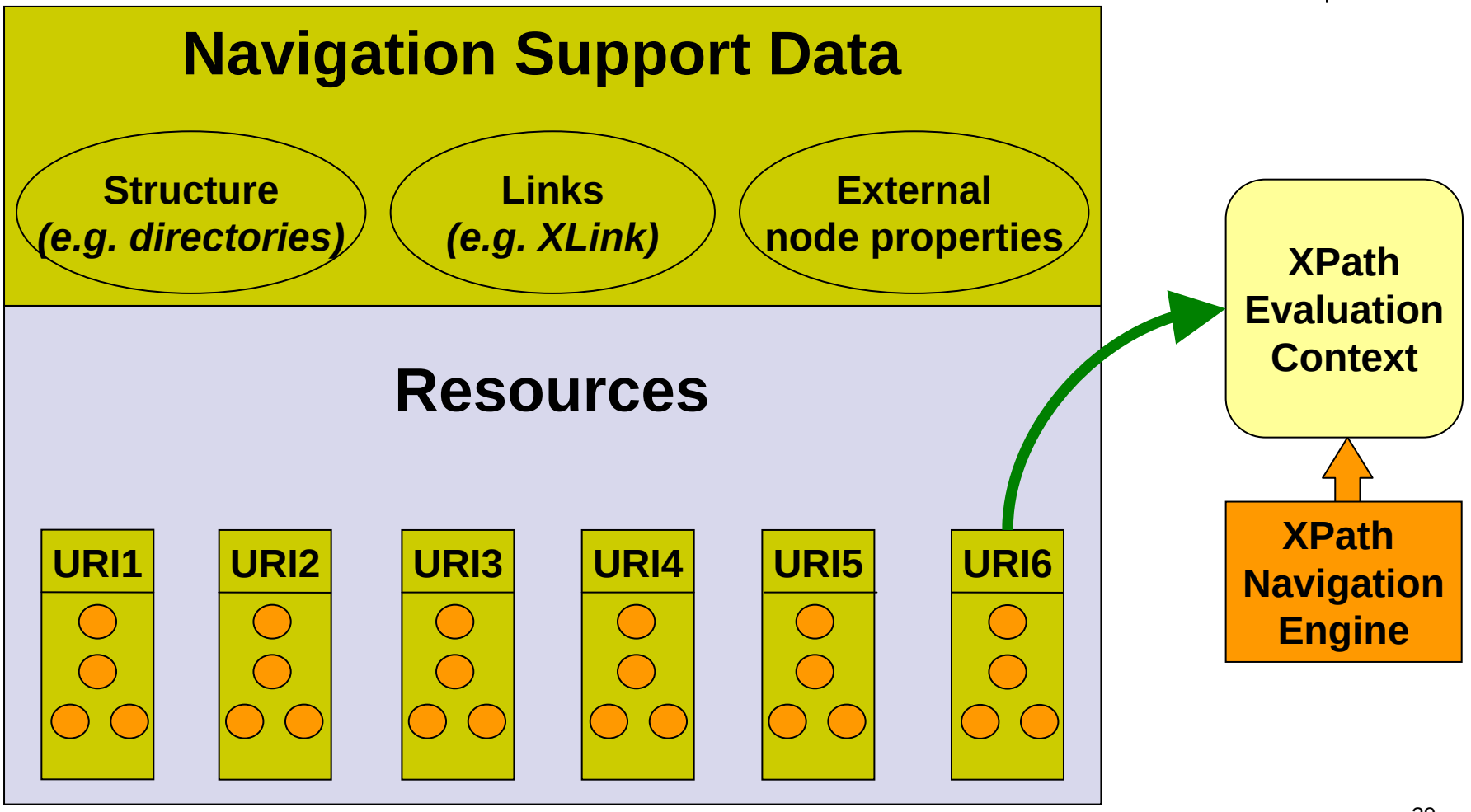
XLink arc: a mapping S1 => S2
where S1 and S2 are sets of resources

XPath arc step: ***"return the ending resources of selected arcs"***

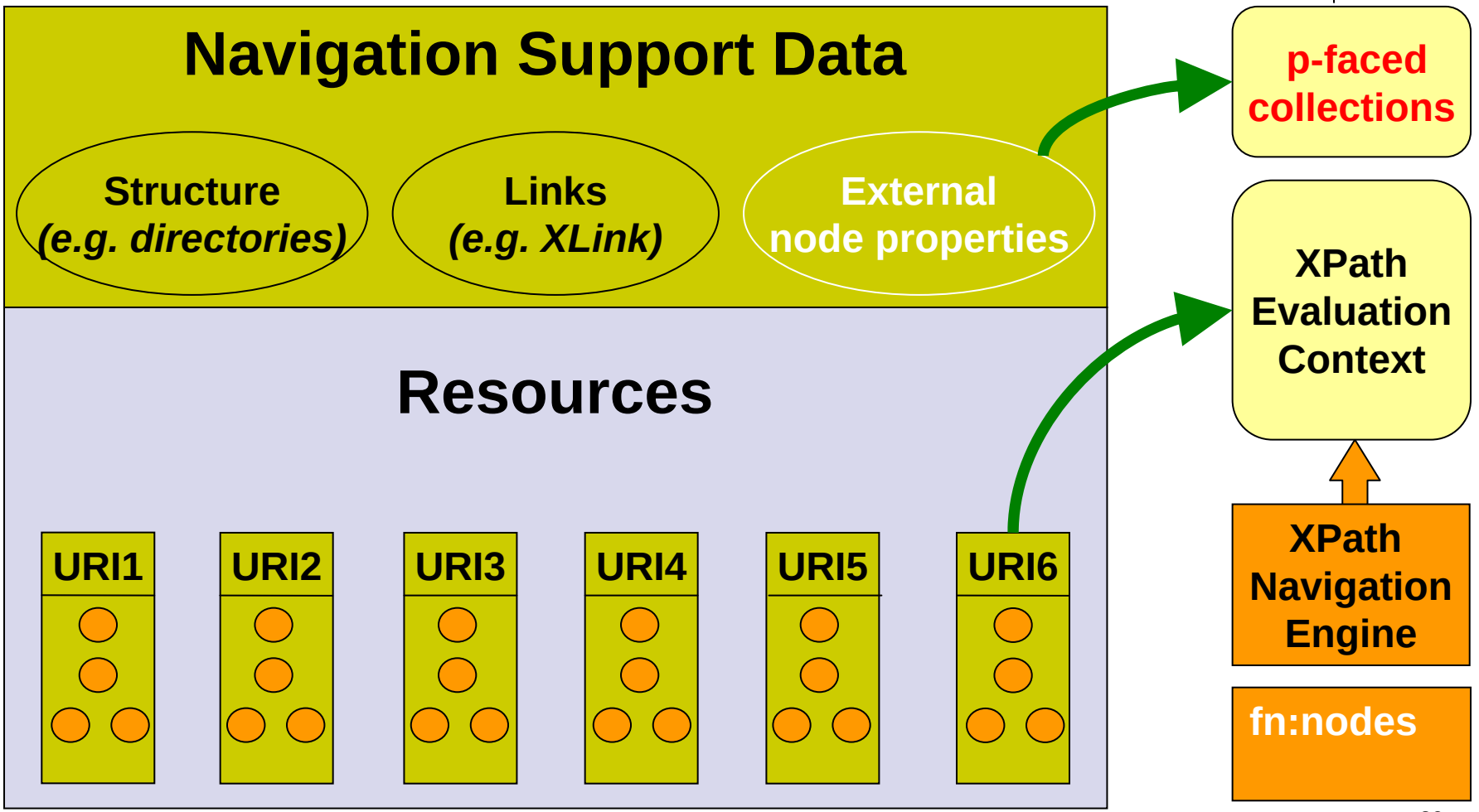
- Arc selection:
  - Arc must "match" a QName
  - The context node must be among the starting resources

- Expressions - axis or function
  - **fn:xlink** **`($arcTest as xs:QName) as node()*`**
  - **xlink::x:y**

Navigation support data



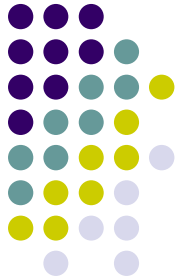
External node properties



The info space – *without collections*

The info space – *with a collection*

Filtered collection

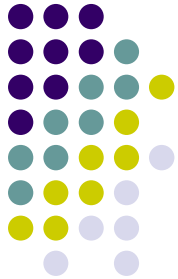


collection URI

```
fn:collection("http://proteins")
              [.//xref/@id eq "P51587"]
```

W A S T E F U L L
(construct all nodes)

p-faced collection



collection URI

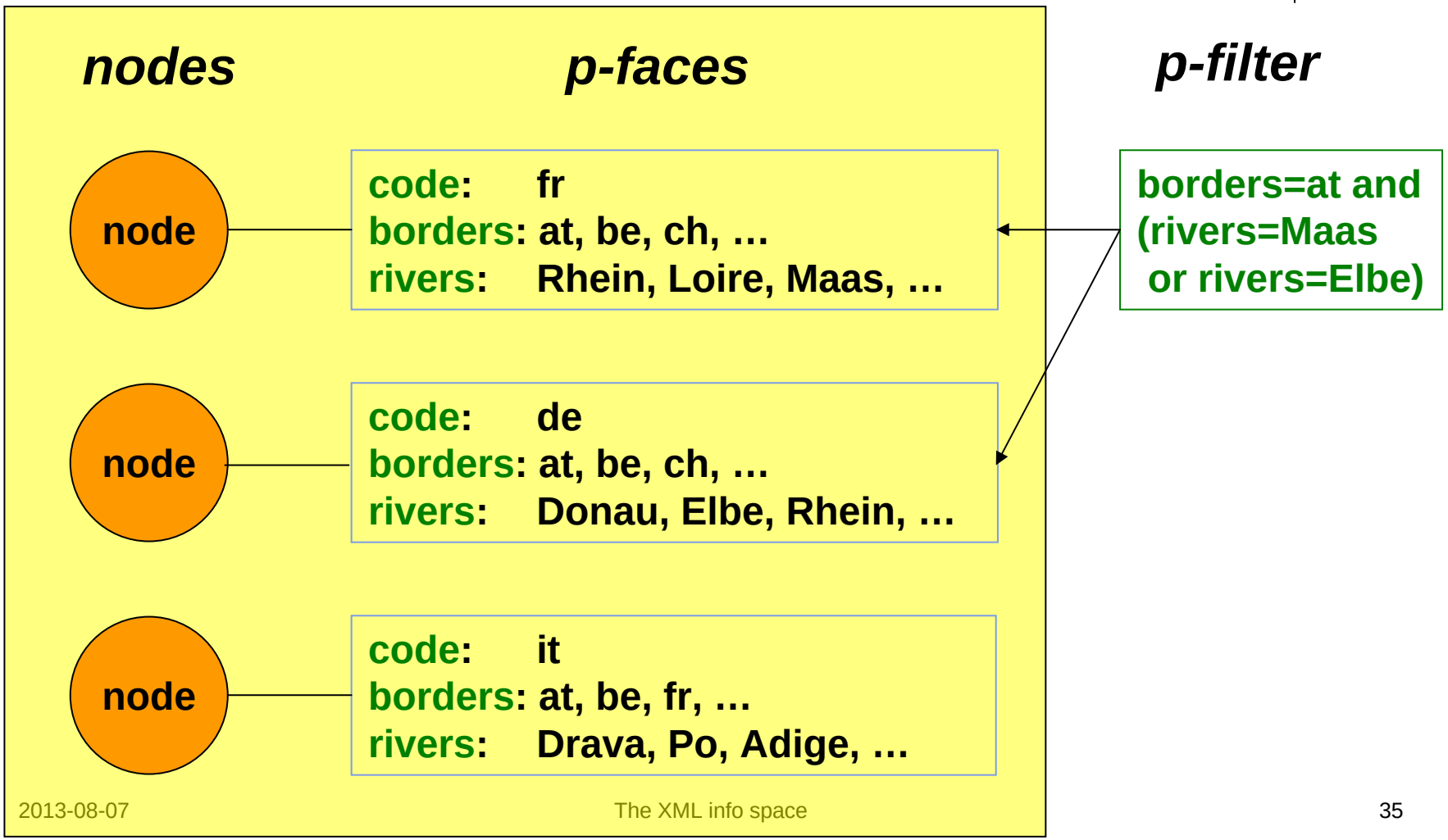
```
fn:nodes("http://proteins?id=P51587")
```

p-filter

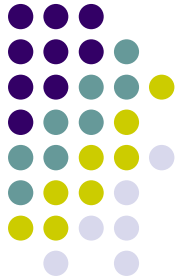
N O W A S T E

(construct only selected nodes)

p-faced collection

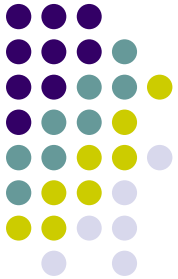


Underlying models



- p-face
  - A set of name/value pairs
  - names - NCName (or better QNames?)
  - values - sequence of atomic values

- p-filter
  - condition tree
  - leaf nodes – p-value comparisons (foo=X, bar>1, zoo~a.*)
  - inner nodes – Boolean operations (and / or / not)



Collection = set of maps

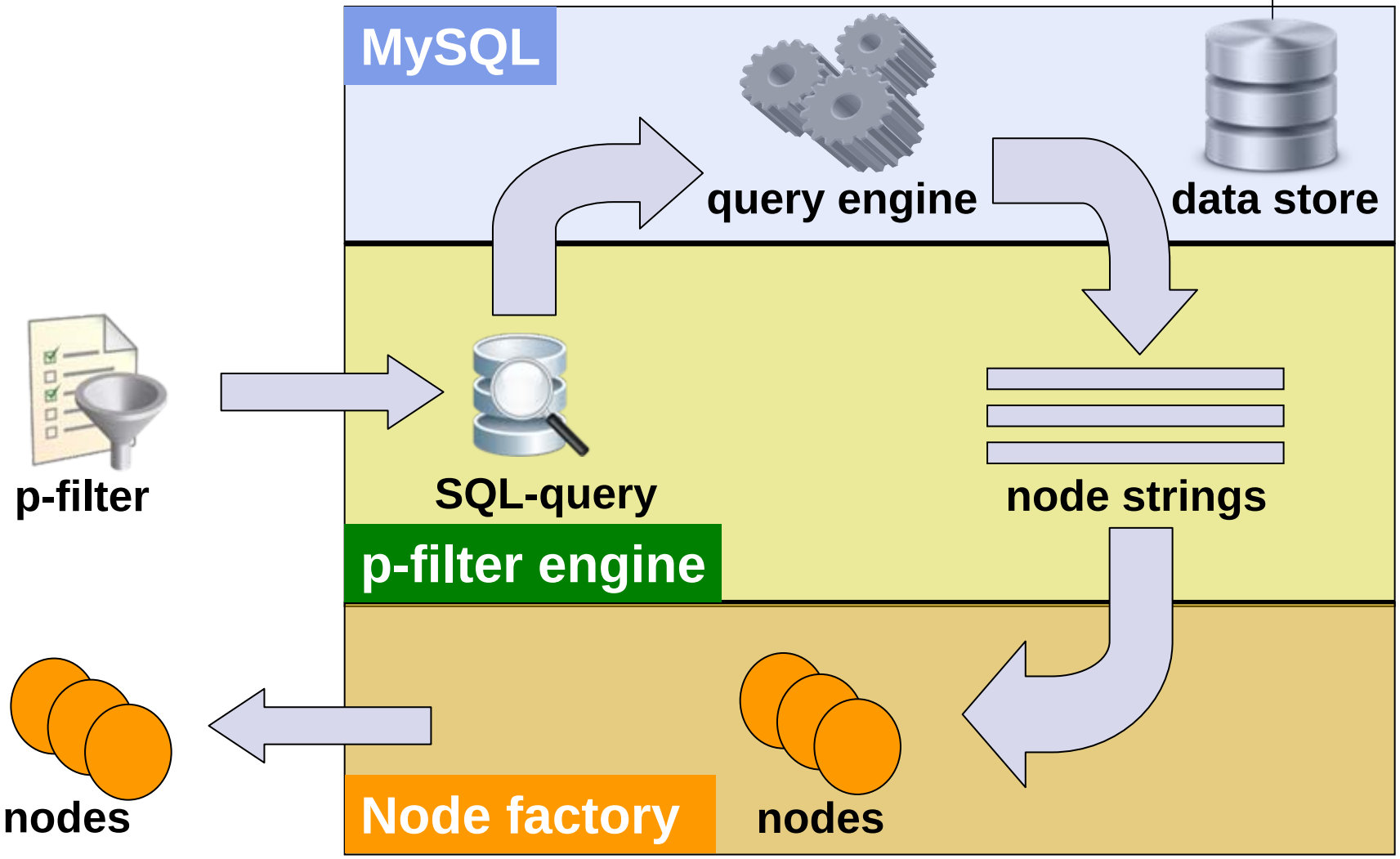
Collection entry = a logical map

- _node = **node string** (URI, serialization, …)
- property1 = **property-value-1**
- property2 = **property-value-2**
- property3 = **property-value-3**

Collection - example

node URI	code	borders	rivers
file://countries/fr.xml	fr	at, be, ch, ...	Rhein, Loire, …
file://countries/de.xml	de	at, be, ch,…	Donau, Elbe, …
file://countries/it.xml	it	at, ch, fr,…	Drava, Po, …

Using non-XML technology



P-filter syntax

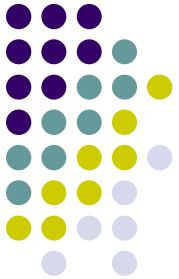
- XML syntax

```
<p:filter>
   <p:or>
      <foo>X</foo>
      <bar op="gt">1</bar>
   </p:or>
</p:filter>
```

- Command syntax

```
foo="X" or bar>"1"
```

REST & fn:nodes



- REST
  - URI = collection-URI + ? + p-filter
  - Result = selected nodes wrapped in a root element

- fn:nodes

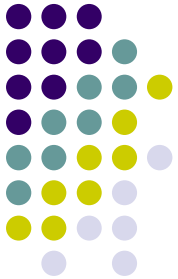
  a) URI = URI of a *deployed collection* ?

  > use p-filter engine and node factory

  b) otherwise

  > invoke REST and deliver child nodes of response root

p-benefits



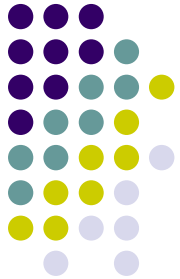
p-faced collections offer two distinct benefits

- Augments XPath navigation **fn:nodes**
- Enables the integration of non-XML technologies into XML data storage and XPath navigation

MySQL

MongoDB

Space – three potentials



- Status quo – follow the space paradigm

- Pushing the boundaries
  - API boundary – new XML access functions
  - Format boundary – non-XML read/write functions

- Addressing the limitations
  - p-faced collections - fn:nodes
  - XLink - fn:xlink

Space – levels of response

- Application level

  extension functions *my:nodes*

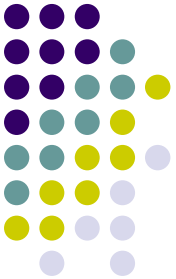
- Processor level

  extension functions *vendor:nodes*

- Standards level

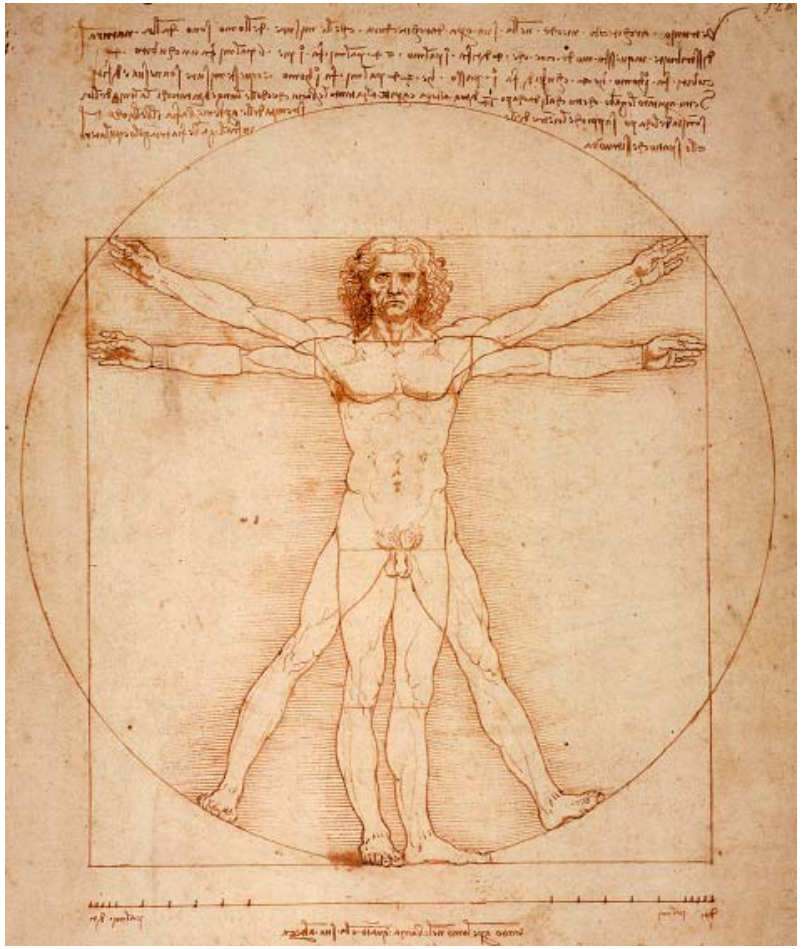
  standard functions *fn:nodes*

Thank you!



content

location



processing

navigation