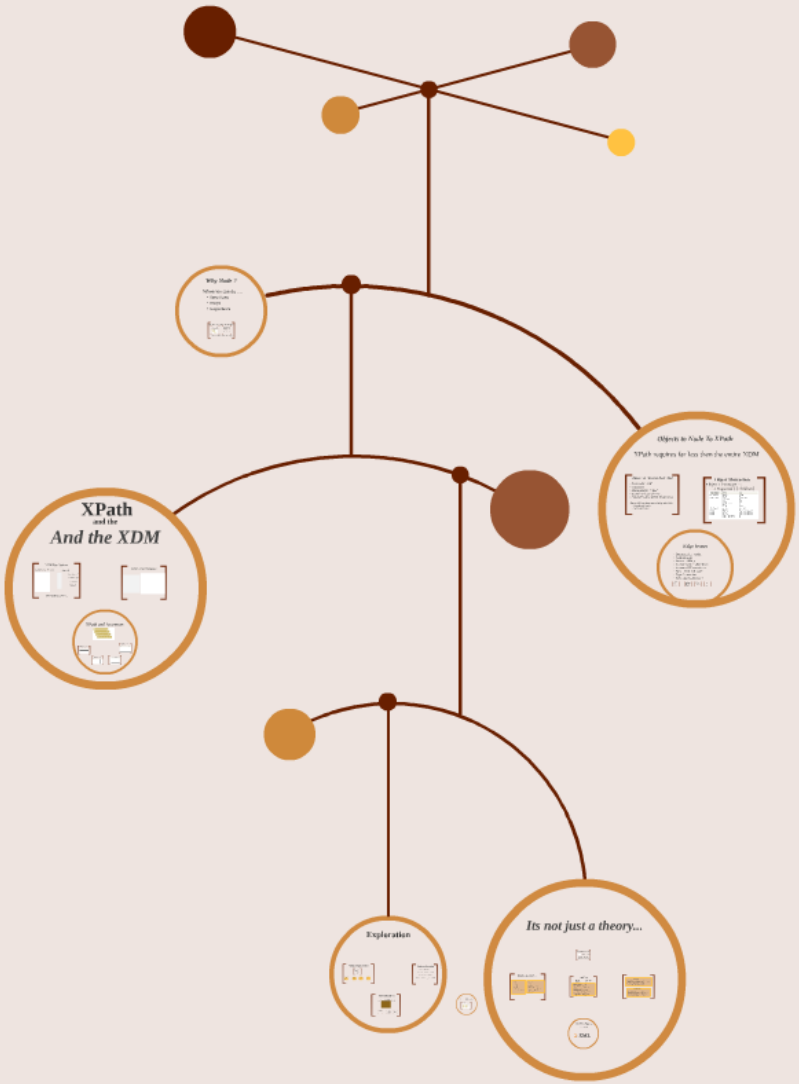
Not Only XML

David A. Lee
dlee@marklogic.com

Extending the relevance of XPath by breaking the chains of the DOM



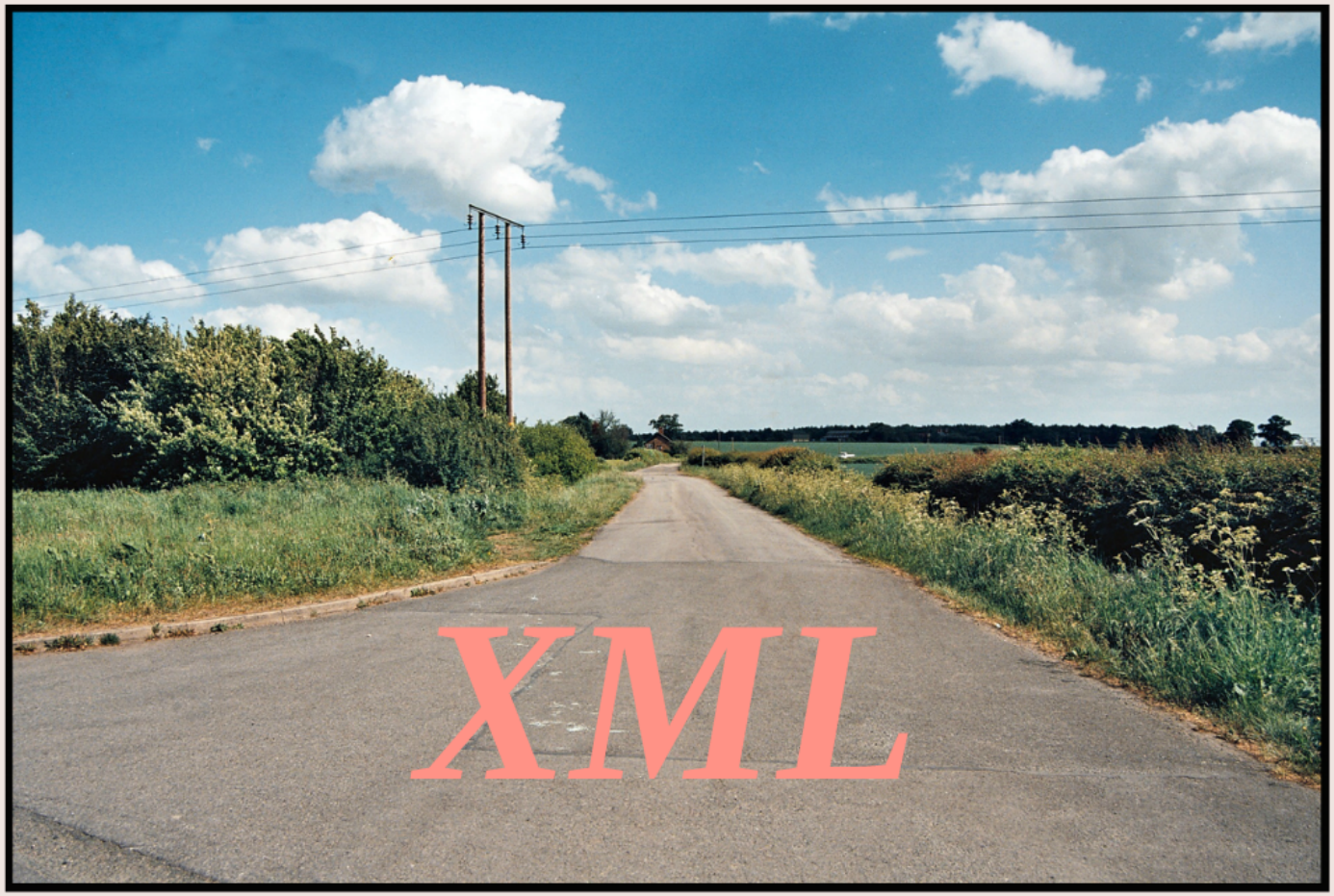


Not Only XML

David A. Lee
dlee@marklogic.com

Extending the relevance of XPath by breaking the chains of the DOM

A Quick trip down XML History Lane

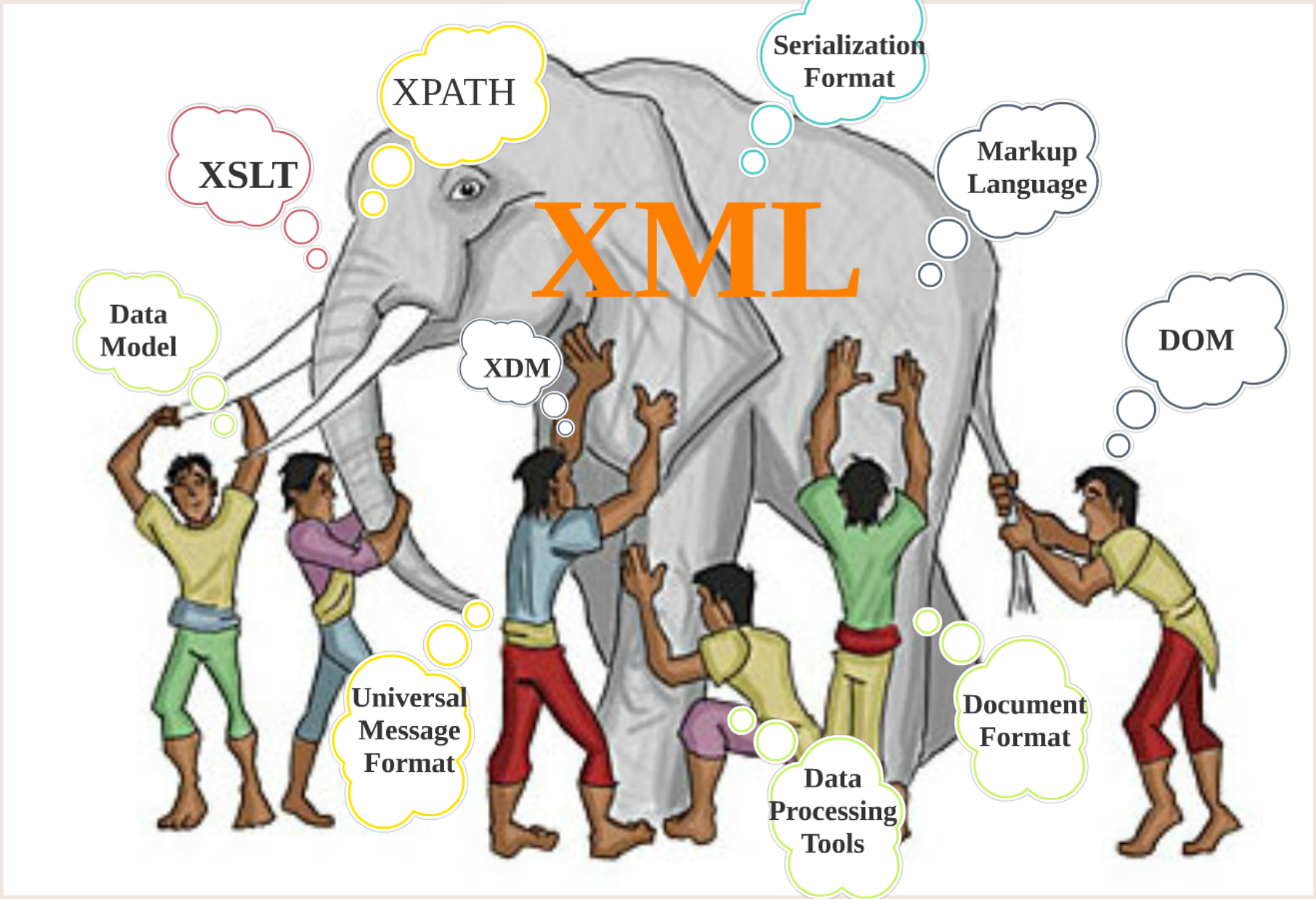


Some
Rose colored
Glasses

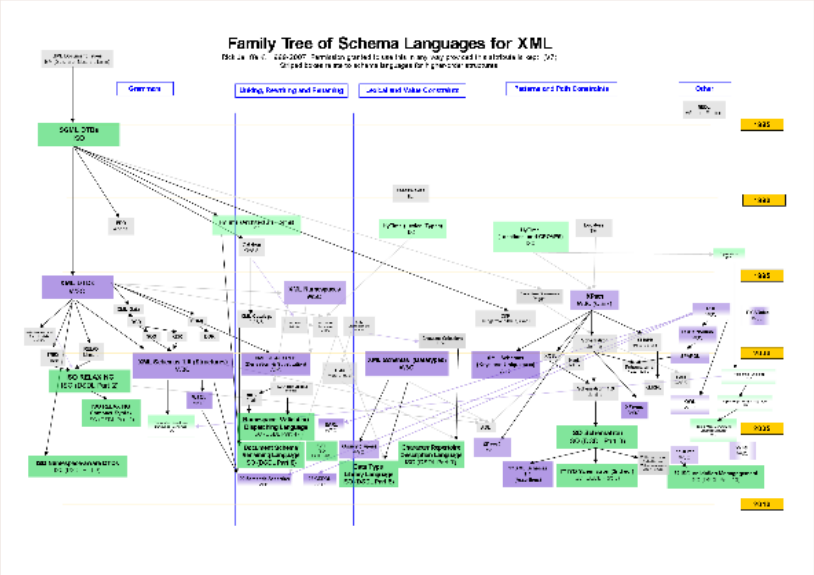
And a Pipe

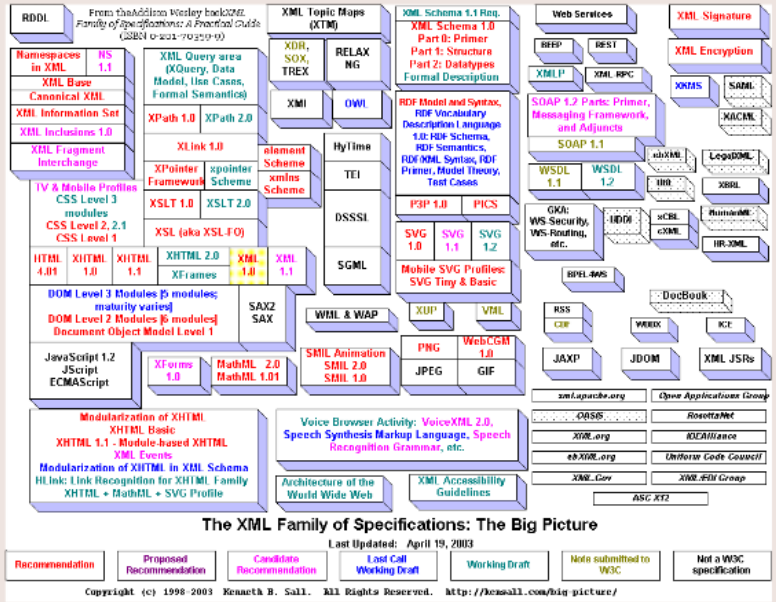
No, This pipe

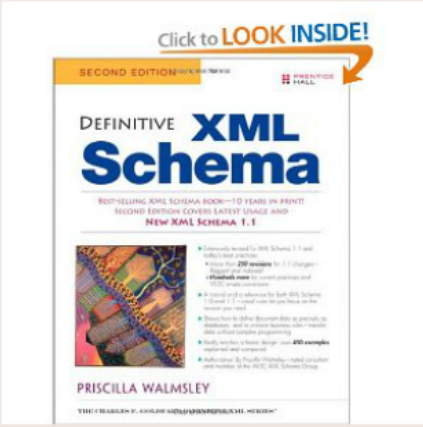
The XML "Experience"

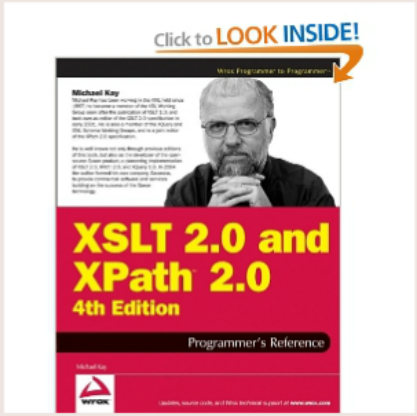


The XML "Reality" ?









Is this "Reality" any more useful ?

Sometimes your own reality is more useful

"There is nothing better ...
Then something that actually works"

Turn back the clock
45 Years

An interesting Parallel

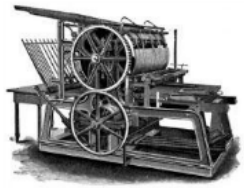
The boys at Bell Labs had the same problem

PAPER TO PAPER

```
:h1.Chapter 1:  Introduction
  :p.GML supported hierarchical
     containers, such as
  :ol
  :li.Ordered lists (like this one),
  :li.Unordered lists, and
```

Paper	• Documents • Key Entry
Files	• GML - 1969 • SGML - 1980
Mainframe	• “Bell” Manuals • Key Entry
PDP-7	• Unix 1969 • C - 1970

XML



THE C PROGRAMMING LANGUAGE

PAPER TO PAPER

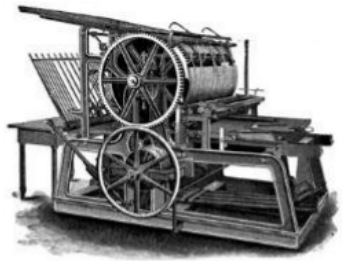
```
:h1.Chapter 1:  Introduction
  :p.GML supported hierarchical
      containers, such as
  :ol
  :li.Ordered lists (like this one),
  :li.Unordered lists, and
```

Paper

- Documents
- Key Entry

Files

- GML - 1969
- SGML - 1980



Mainframe

- “Bell” Manuals
- Key Entry

PDP-7

- Unix 1969
- C - 1970

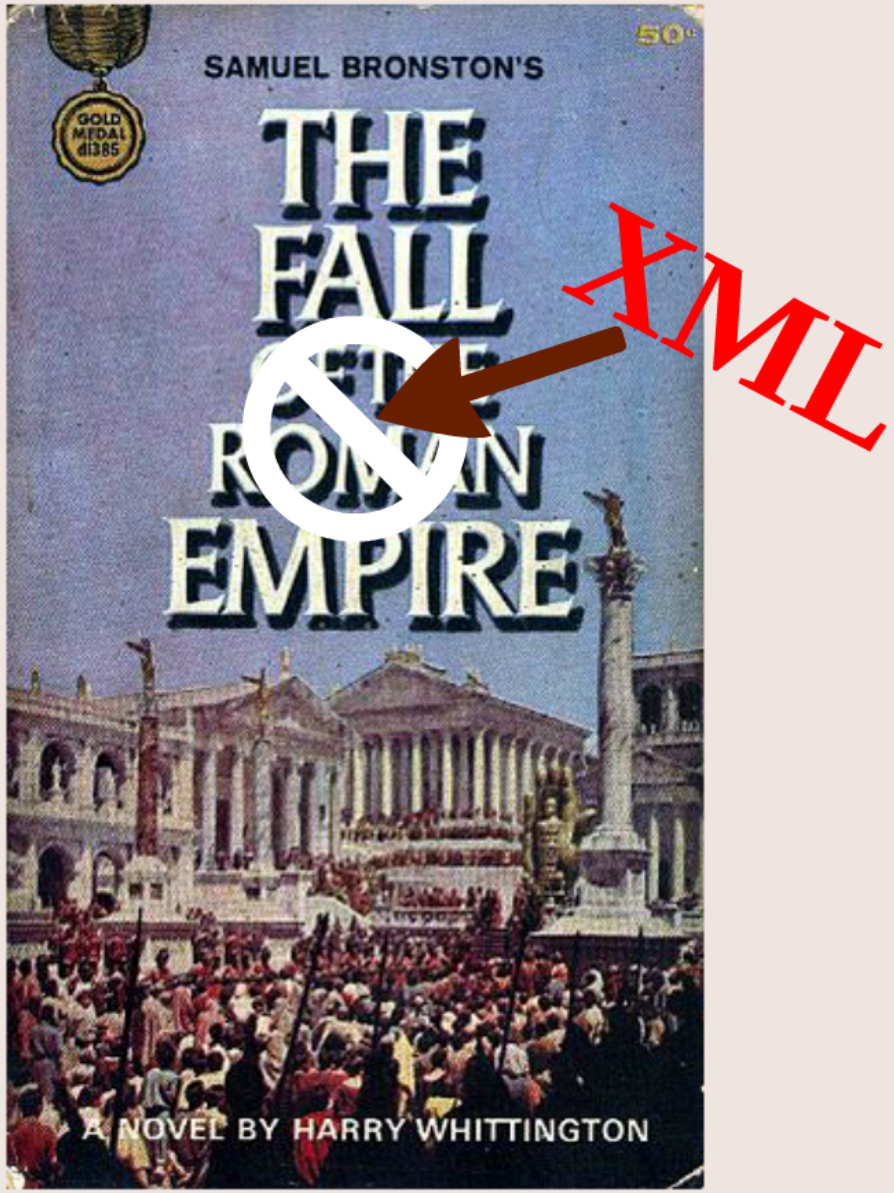
Time Ahead 30 years

We dreamed of ..

One Format to Rule them All

The Rise of XML

XML Is Dead


SAMUEL BRONSTON'S
GOLD MEDAL d1385
THE FALL
ROMAN EMPIRE
A NOVEL BY HARRY WHITTINGTON
XML


We Failed


We Won !
XML

We Failed

The Dream of Universal Adoption

Total admiration
Love, Respect, and
World Domination

One Markup for
Everything and
everyone

XML replacing HTML

Staying "Kewl" for a decade

Predicting the next Generation
Would be JUST LIKE US

We Won !

Massive adoption of XML worldwide

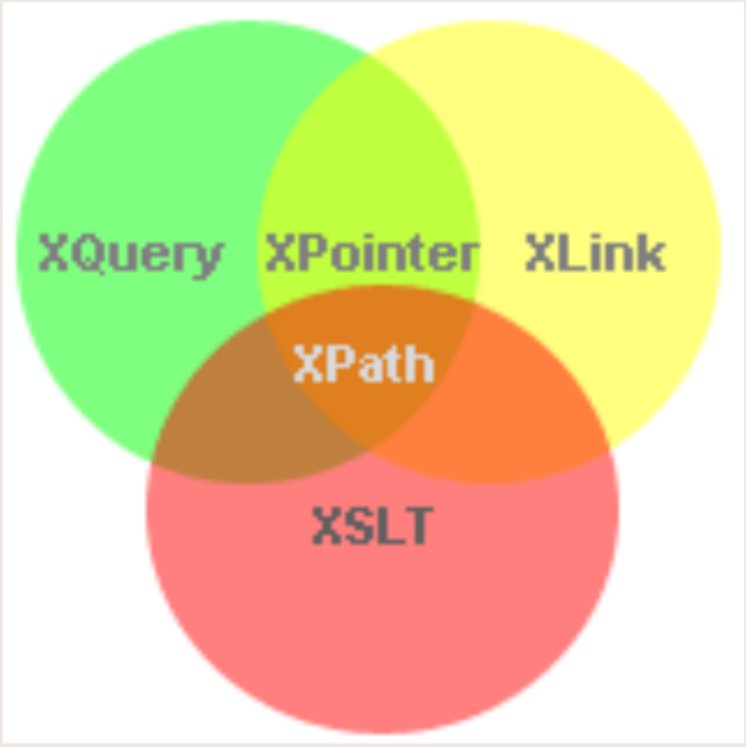
Incredible research and Engineering

XML

Semantic Markup is Mainstream

Tools Tools and More tools !

The One Ring To Rule them All



XPATH

XPath
and the
And the XDM

XDM Type System

Rich Atomic Type Hierarchy

Node Model

Object Hierarchy ?

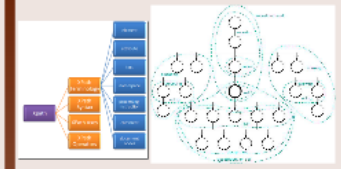
Substitution Group?

Property Model ?

"Kind"

Other Stuff tossed here ...

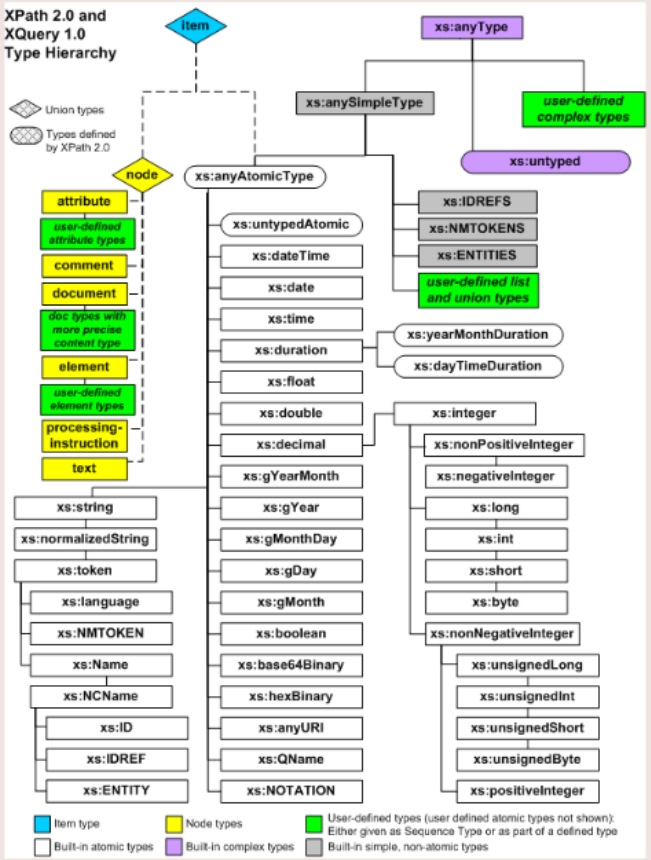
XPath - Tree Navigation



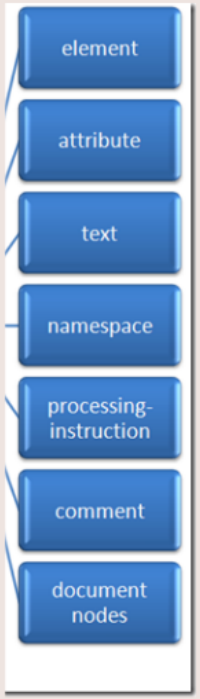
XPath and Accessors

XDM Type System

Rich Atomic Type Hierarchy



Node Model



Object Hierarchy ?

Substitution Group?

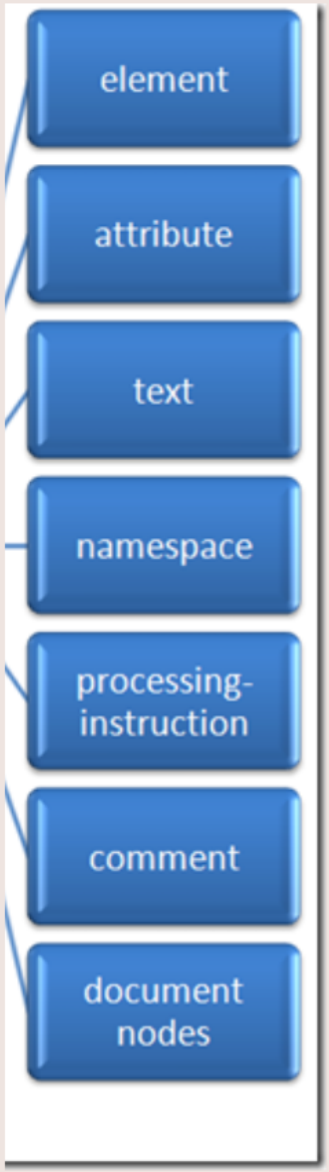
Property Model ?

"Kind"

Other Stuff tossed here ...

rchy

Node Model



Object Hierarchy ?

Substitution Group?

Property Model ?

"Kind"

A segue

Terminology :

XDM "node" == "XML Node"

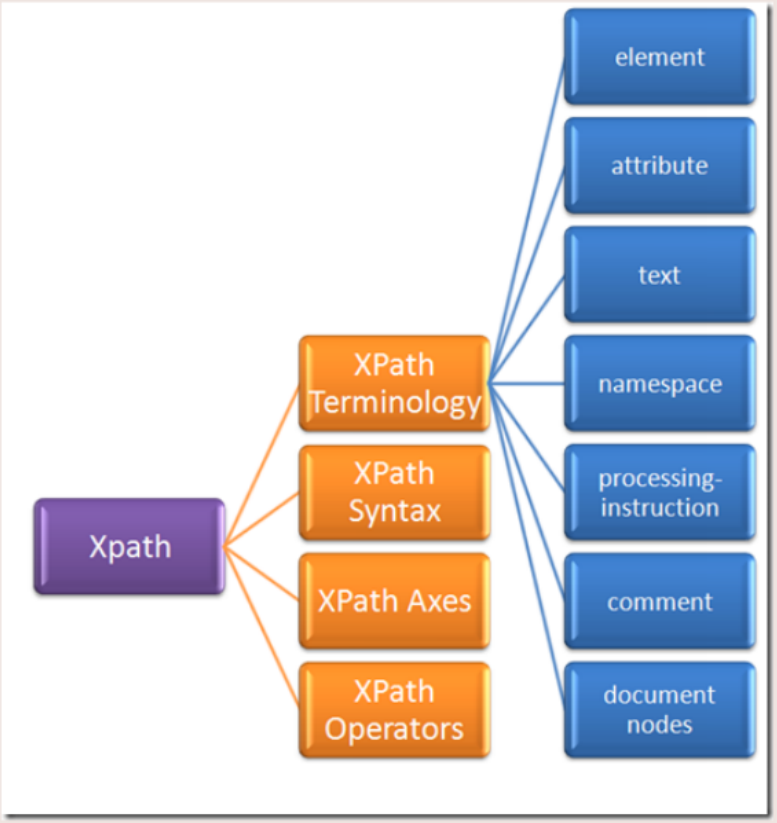
Why?

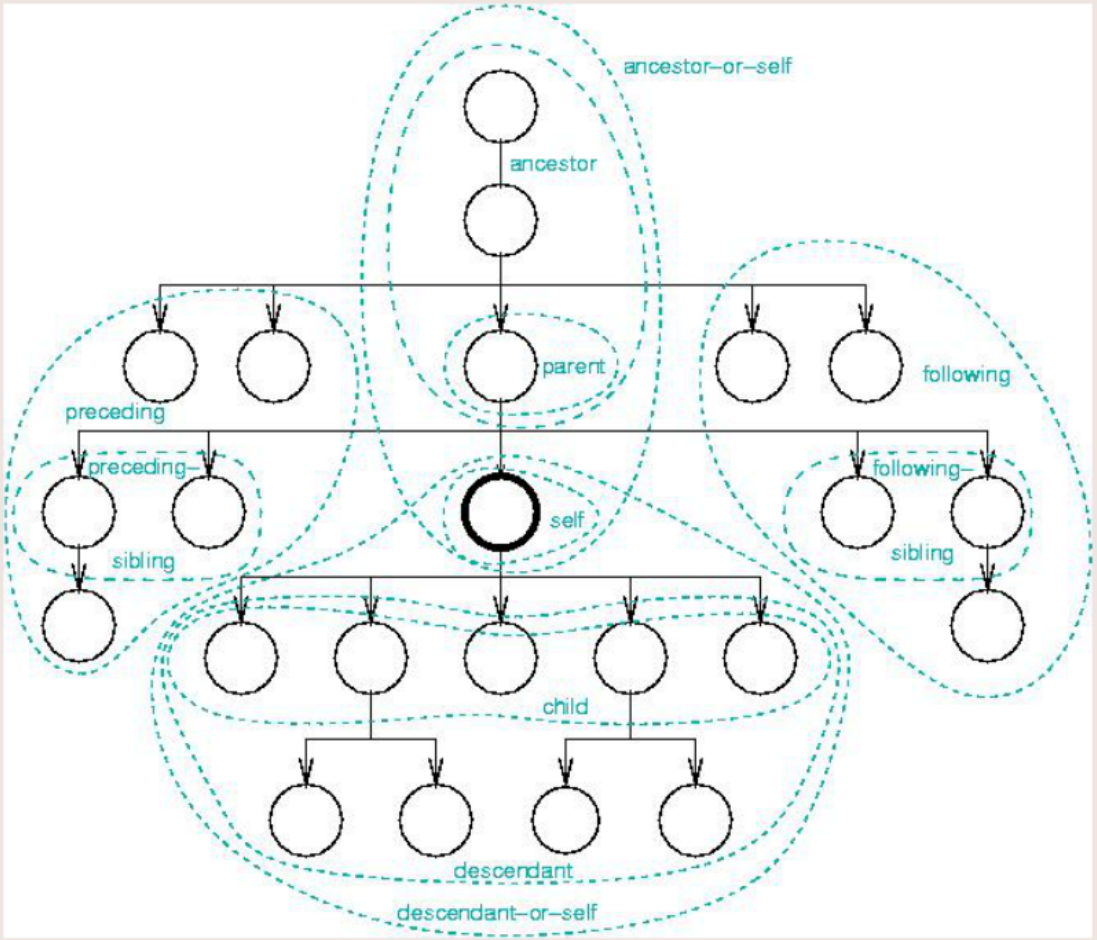
The Serialization format came before the Data model

An artifact of history ?

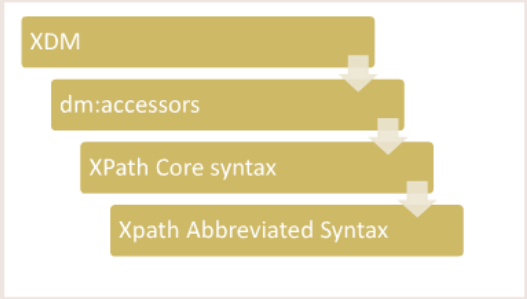
Or an intrinsic necessity ?

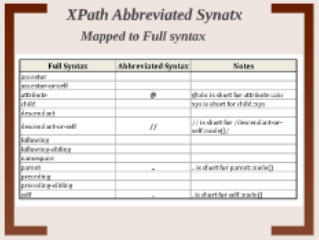
XPath - Tree Navigation

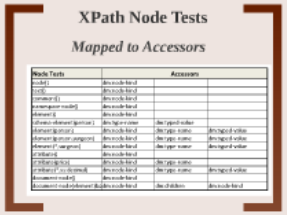


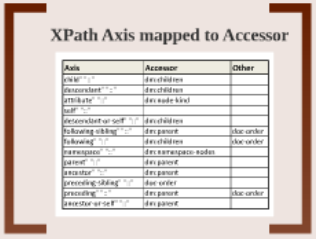


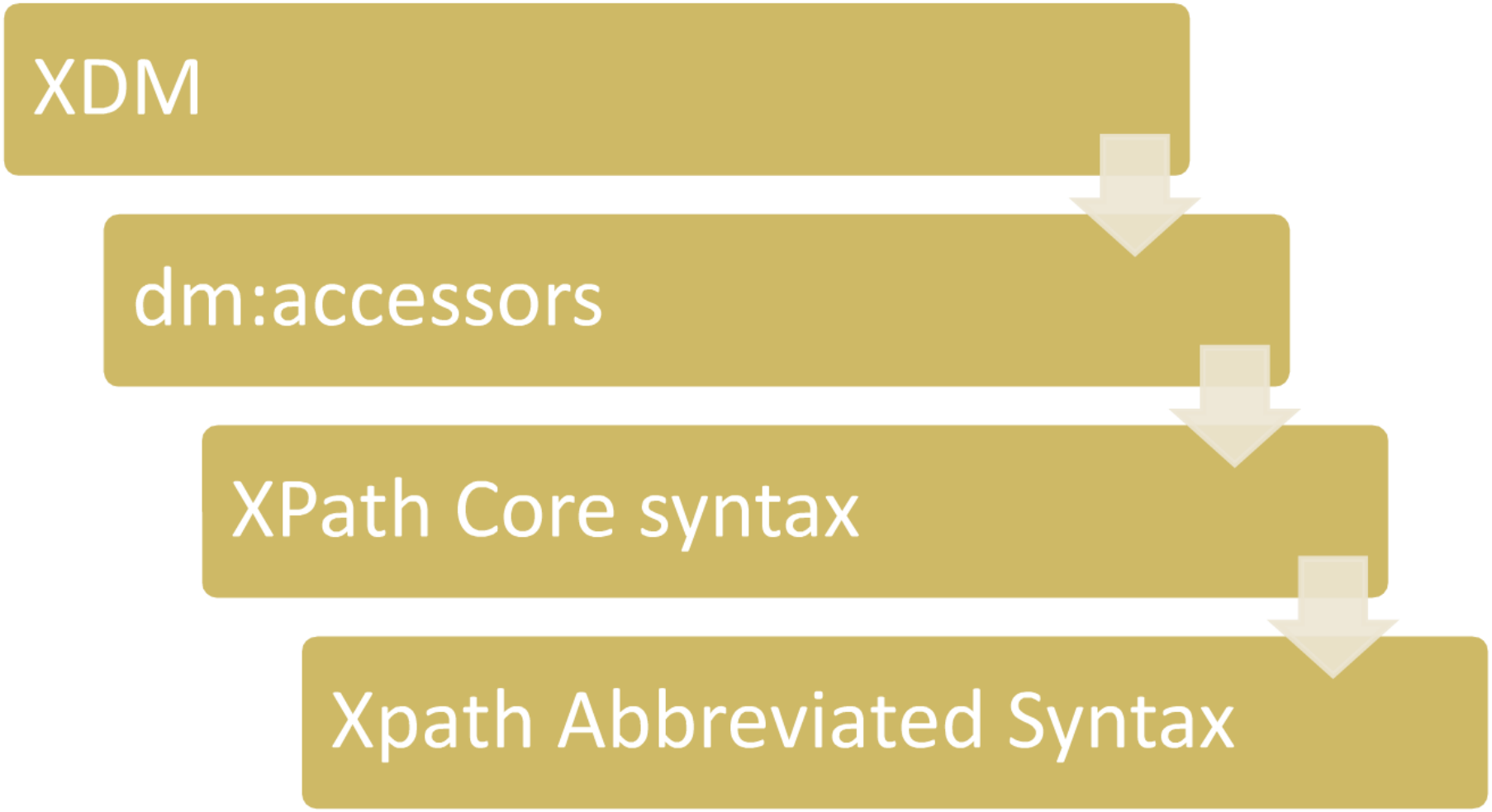
XPath and Accessors










XDM
dm:accessors
XPath Core syntax
Xpath Abbreviated Syntax

"dm" Accessors

XPath's view of the Node

Accessor	Document	Element	attribute	text	namespace	PI	comment
dm:attributes	()	*	()	()	()	()	()
dm:base-uri	?	?	?	?	()	Y	?
dm:children	*	*	()	()	()	()	()
dm:document-uri	?	()	()	()	()	()	()
dm:is-id	()	Y	Y	()	()	()	()
dm:is-idrefs	()	*	*	()	()	()	()
dm:namespace-nodes	()	*	()	()	()	()	()
dm:nilled	()	Y	()	()	()	()	()
dm:node-kind	document	element	attribute	text	namespace	pi	comment
dm:node-name	()	Y	Y	()	?	Y	()
dm:parent	()	Y	Y	Y	Y	Y	Y
dm:string-value	()	?	Y	Y	Y	Y	Y
dm:type-name	Y	Y	Y	Y	()	()	()
dm:typed-value	Y	?	?	Y	Y	Y	()
dm:unparsed-entity-public-id	?	()	()	()	()	()	()
dm:unparsed-entity-system-id	()	()	()	()	()	()	()

XPath's view of the Node

Accessor	Document	Element	attribute	text	namespace	PI	comment
dm:attributes	()	*	()	()	()	()	()
dm:base-uri	?	?	?	?	()	Y	?
dm:children	*	*	()	()	()	()	()
dm:document-uri	?	()	()	()	()	()	()
dm:is-id	()	Y	Y	()	()	()	()
dm:is-idrefs	()	*	*	()	()	()	()
dm:namespace-nodes	()	*	()	()	()	()	()
dm:nilled	()	Y	()	()	()	()	()
dm:node-kind	document	element	attribute	text	namespace	pi	comment
dm:node-name	()	Y	Y	()	?	Y	()
dm:parent	()	Y	Y	Y	Y	Y	Y
dm:string-value	()	?	Y	Y	Y	Y	Y
dm:type-name	Y	Y	Y	Y	()	()	()
dm:typed-value	Y	?	?	Y	Y	Y	()
dm:unparsed-entity-public-id	?	()	()	()	()	()	()
dm:unparsed-entity-system-id	()	()	()	()	()	()	()

XPath Node Tests

Mapped to Accessors

Node Tests	Accessors		
node()	dm:node-kind		
text()	dm:node-kind		
comment()	dm:node-kind		
namespace-node()	dm:node-kind		
element()	dm:node-kind		
schema-element(person)	dm:type-name	dm:typed-value	
element(person)	dm:node-kind	dm:type-name	dm:typed-value
element(person,surgeon)	dm:node-kind	dm:type-name	dm:typed-value
element(*,surgeon)	dm:node-kind	dm:type-name	dm:typed-value
attribute()	dm:node-kind		
attribute(price)	dm:node-kind	dm:type-name	
attribute(*,xs:decimal)	dm:node-kind	dm:type-name	dm:typed-value
document-node()	dm:node-kind		
document-node(element(bo	dm:node-kind	dm:children	dm:node-kind

Mapped to Accessors

Node Tests	Accessors		
node()	dm:node-kind		
text()	dm:node-kind		
comment()	dm:node-kind		
namespace-node()	dm:node-kind		
element()	dm:node-kind		
schema-element(person)	dm:type-name	dm:typed-value	
element(person)	dm:node-kind	dm:type-name	dm:typed-value
element(person,surgeon)	dm:node-kind	dm:type-name	dm:typed-value
element(*,surgeon)	dm:node-kind	dm:type-name	dm:typed-value
attribute()	dm:node-kind		
attribute(price)	dm:node-kind	dm:type-name	
attribute(*,xs:decimal)	dm:node-kind	dm:type-name	dm:typed-value
document-node()	dm:node-kind		
document-node(element(bo	dm:node-kind	dm:children	dm:node-kind

XPath Axis mapped to Accessor

Axis	Accessor	Other
child" "::"	dm:children	
descendant" "::"	dm:children	
attribute" "::"	dm:node-kind	
self" "::"		
descendant-or-self" "::"	dm:children	
following-sibling" "::"	dm:parent	doc-order
following" "::"	dm:children	doc-order
namespace" "::"	dm:namespace-nodes	
parent" "::"	dm:parent	
ancestor" "::"	dm:parent	
preceding-sibling" "::"	doc-order	
preceding" "::"	dm:parent	doc-order
ancestor-or-self" "::"	dm:parent	

Axis	Accessor	Other
child" "::"	dm:children	
descendant" "::"	dm:children	
attribute" "::"	dm:node-kind	
self" "::"		
descendant-or-self" "::"	dm:children	
following-sibling" "::"	dm:parent	doc-order
following" "::"	dm:children	doc-order
namespace" "::"	dm:namespace-nodes	
parent" "::"	dm:parent	
ancestor" "::"	dm:parent	
preceding-sibling" "::"	doc-order	
preceding" "::"	dm:parent	doc-order
ancestor-or-self" "::"	dm:parent	

XPath Abbreviated Synatx

Mapped to Full syntax

Full Syntax	Abbreviated Syntax	Notes
ancestor		
ancestor-or-self		
attribute	@	@abc is short for attribute::abc
child		xyz is short for child::xyz
descendant		
descendant-or-self	//	// is short for /descendant-or-self::node()/
following		
following-sibling		
namespace		
parent	..	.. is short for parent::node()
preceding		
preceding-sibling		
self	.	. is short for self::node()

Mapped to Full syntax

Full Syntax	Abbreviated Syntax	Notes
ancestor		
ancestor-or-self		
attribute	@	@abc is short for attribute::abc
child		xyz is short for child::xyz
descendant		
descendant-or-self	//	// is short for /descendant-or-self::node()/
following		
following-sibling		
namespace		
parent	..	.. is short for parent::node()
preceding		
preceding-sibling		
self	.	. is short for self::node()

Why Node ?

When we can do ....

- functions
- maps
- sequences

The "node" is the foundation of XPath

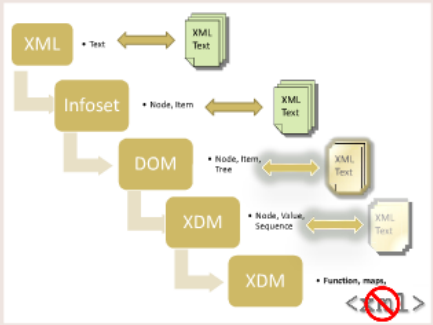
Shackled to History

Greatest Strength of XML Languages

Extreme coupling from Data Model to Serialization

Worst Enemy for the future of XML Languages

Insistence on Extreme coupling from Data Model to Serialization



XPath may be the key

- Single common expression language
- Powerful and terse
- Simple abstraction unifying all XDM Types

But XPath needs "Node"

Type Bifurcation abandons XPath

maps / arrays / json /

XPath is the foundation of the XML Languages

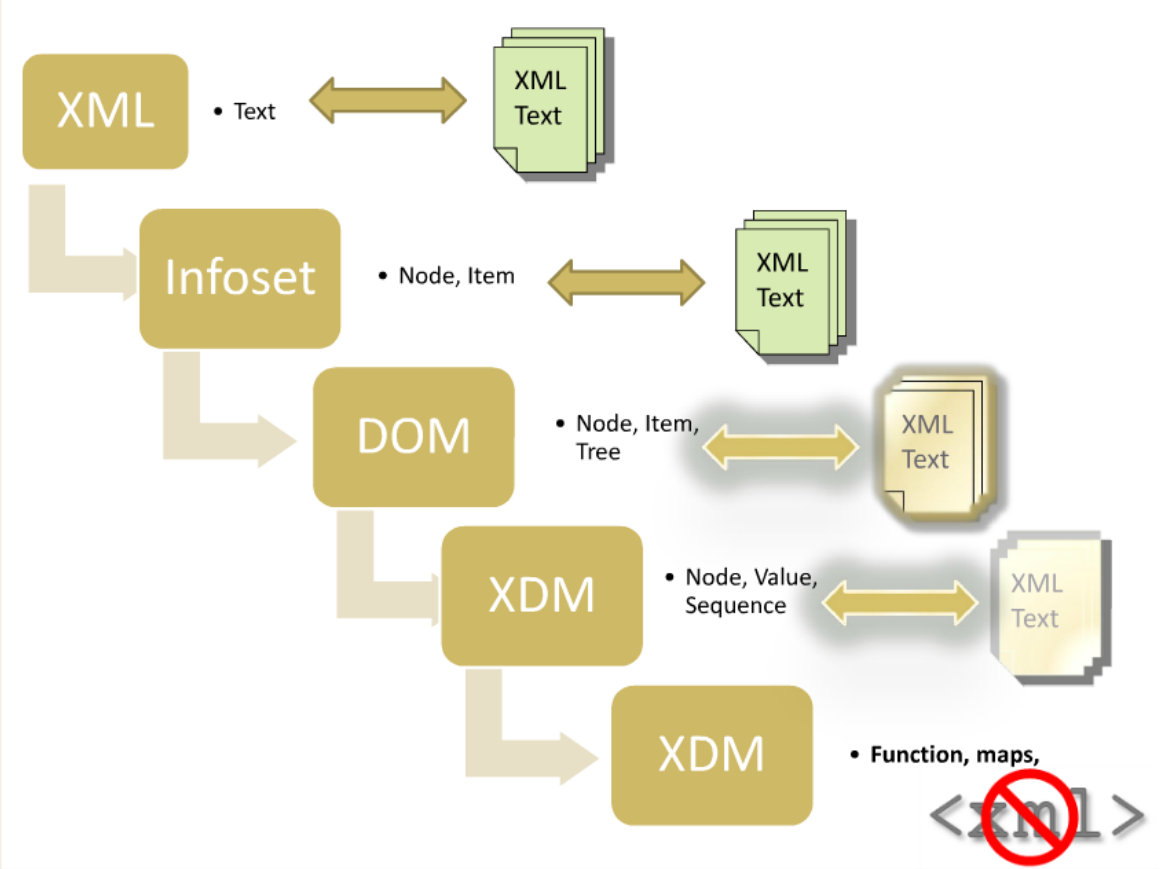
Shackled to History

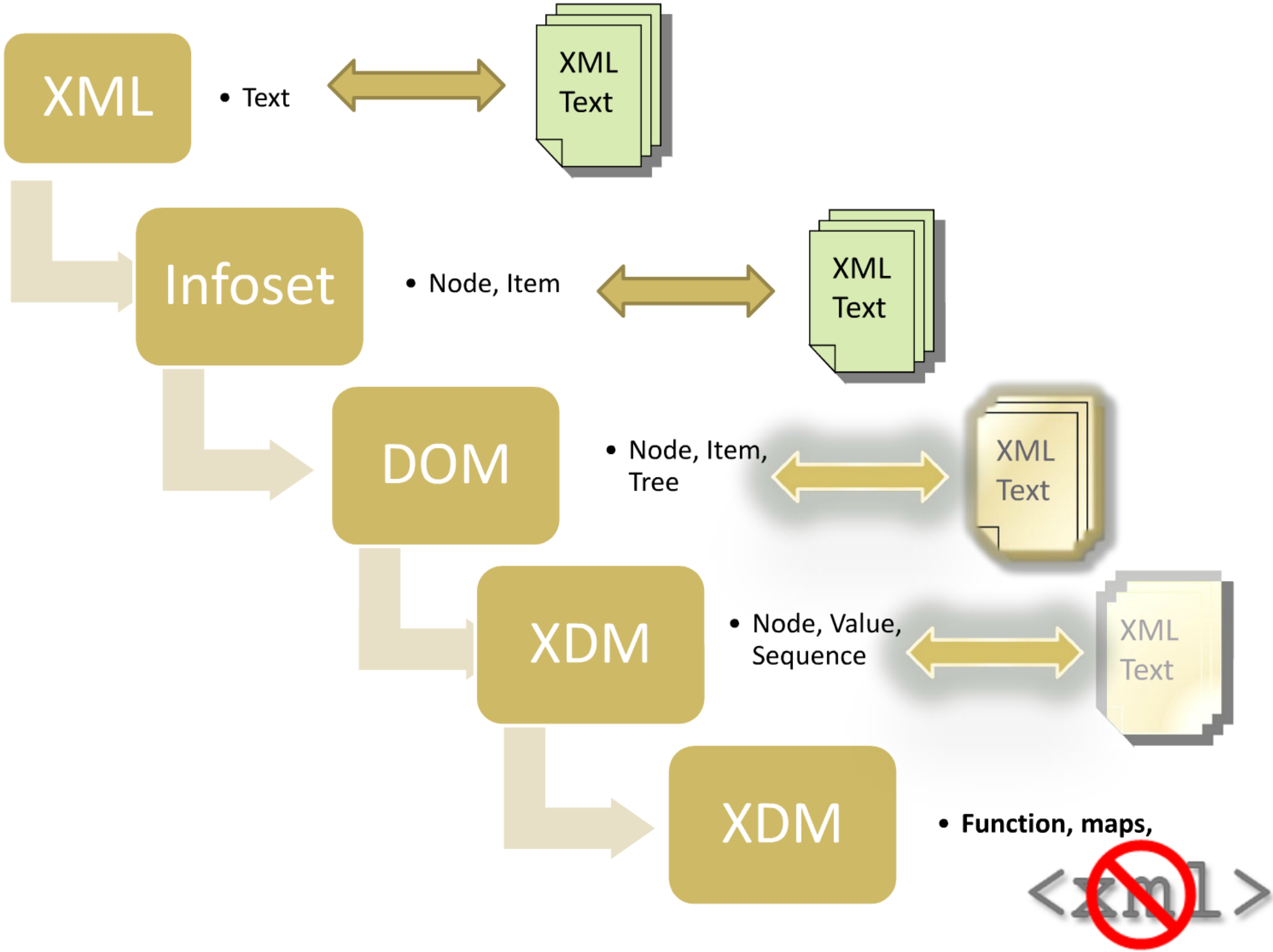
Greatest Strength of XML Languages

Extreme coupling from Data Model to Serialization

Worst Enemy for the future of XML Languages

Insistence on Extreme coupling from Data Model to Serialization




XML
• Text
XML Text
Infoset
• Node, Item
XML Text
DOM
• Node, Item, Tree
XML Text
XDM
• Node, Value, Sequence
XML Text
XDM
• Function, maps,
<xml>

XPath may be the key

- **Single common expression language**
- **Powerful and terse**
- **Simple abstraction unifying all XDM Types**

But XPath needs "Node"

Type Bifurcation abandons XPath

maps / arrays / json /

- **Powerful and terse**

- **Simple abstraction unifying all XDM Types**

But XPath needs "Node"

Type Bifurcation abandons XPath

maps / arrays / json /

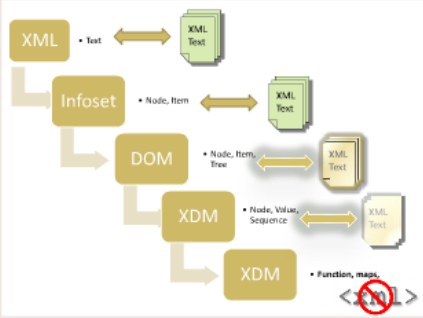
Shackled to History

Greatest Strength of XML Languages

Extreme coupling from Data Model to Serialization

Worst Enemy for the future of XML Languages

Insistence on Extreme coupling from Data Model to Serialization



XPath may be the key

- Single common expression language
- Powerful and terse
- Simple abstraction unifying all XDM Types

But XPath needs "Node"

Type Bifurcation abandons XPath

maps / arrays / json /

XPath is the foundation of the XML Languages

Objects to Node To XPath

XPath requires far less then the entire XDM

Object : an "Abstract Node" kind

- New node "kind"
- Accessors
- Document ID "rules"
- Container / Containment
- "Stretch" some builtin XPath rules

Object Abstraction

Object < Prototype > :

{ Properties } [Children]

Object Propertis	Accessor	Value
Named Properties	dm:attributes	? Possible
	dm:base-uri	()
Contained Objects	dm:children	Contained Object
	dm:document-uri	()

Object : an "Abstract Node" kind

- New node "kind"
- Accessors
- Document ID "rules"
- Container / Containment
- "Stretch" some builtin XPath rules

Not too different then some existing node kinds

- ***Constructed Nodes***
- ***Document Nodes***

Object Abstraction

Object < Prototype > :
{ Properties } [Children]

Object Propertis	Accessor	Value
Named Properties	dm:attributes	? Possible
	dm:base-uri	()
Contained Objects	dm:children	Contained Object
	dm:document-uri	()
	dm:is-id	"false"
	dm:is-idrefs	"false"
	dm:namespace-nodes	()
	dm:nilled	?
	dm:node-kind	"object"
"Name" Property	dm:node-name	"name" or ()
	dm:parent	()
Definable	dm:string-value	Depends on object properties
Definable	dm:type-name	Depends on object properties
Definable	dm:typed-value	Depends on object properties
	dm:unparsed-entity-public-id	()
	dm:unparsed-entity-system-id	()

{ Properties } [Children]

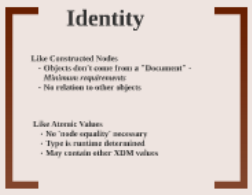
Object Propertis	Accessor	Value
Named Properties	dm:attributes	? Possible
	dm:base-uri	()
Contained Objects	dm:children	Contained Object
	dm:document-uri	()
	dm:is-id	"false"
	dm:is-idrefs	"false"
	dm:namespace-nodes	()
	dm:nilled	?
	dm:node-kind	"object"
"Name" Property	dm:node-name	"name" or ()
	dm:parent	()
Definable	dm:string-value	Depends on object properties
Definable	dm:type-name	Depends on object properties
Definable	dm:typed-value	Depends on object properties
	dm:unparsed-entity-public-id	()
	dm:unparsed-entity-system-id	()

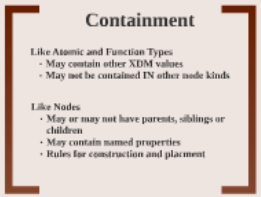
Edge issues

- Document ID / Order
- Node equality
- Parent / Sibling
- Containment of other types
- Contained IN other types
- New atomic null type?
- Type Conversion
- New axis / node tests ?

Edge issues

- Document ID / Order
- Node equality
- Parent / Sibling
- Containment of other types
- Contained IN other types
- New atomic null type?
- Type Conversion
- New axis / node tests ?





Identity

Like Constructed Nodes

- **Objects don't come from a "Document" - *Minimum requirements***
- **No relation to other objects**

Like Atomic Values

- **No 'node equality' necessary**
- **Type is runtime determined**
- **May contain other XDM values**

Containment

Like Atomic and Function Types

- **May contain other XDM values**
- **May not be contained IN other node kinds**

Like Nodes

- **May or may not have parents, siblings or children**
- **May contain named properties**
- **Rules for construction and placment**

Type Conversion

May supply conversion rules ...

- **To other types**
- **From other types**
- **When placed in other types**
- **When values are inserted**

New Axis ?

- ***May be useful ...***
- ***May not be required ...***

New node tests

- ***Probably needed***

Exploration

Object Type System

Object is a "Meta" Node

- Substitutable for Node
- Substitutable for Item
- Generic container
- Properties / Arrays / Meta Data
- Instances of Object can express different behaviour

Object "Types"

Concept:
- C++ Template
- Java Generic
- JavaScript Prototype
- Schema

```
Type : object<
    { Prototype }
    { Constraints }
    { rules }
    { Signature }
>
```

An element like node

Prototype

```
object< xs:QName ? ,
    object<xs:string,xs:string>*,
    node()* >
```

Example

```
object {  "MyElement" ,
    { "attribute" : "value" } ,
        <child>text<child> ,
        text { "mixed" } }
```

An Array like Node

Prototype

```
object< xs:anyType* >
```

Example

```
object { "one" , 2 ,
    object { document { ... } },
    function () ...
}
```

An JSON Node ?

Prototype

```
object<
    object { xs:string, xs:anyType } * |
    object* >
```

Example

```
object {
    { "field1" , 10 },
    { "field2 , { "string" , 1.5 , () }
}
```

Object is a "Meta" Node

- Substitutable for Node
- Substitutable for Item
- Generic container
- Properties / Arrays / Meta Data
- Instances of Object can express different behaviour

Object is a "Meta" Node

- Substitutable for Node
- Substitutable for Item
- Generic container
- Properties / Arrays / Meta Data
- Instances of Object can express different behaviour

Object "Types"

Concept:

- C++ Template
- Java Generic
- JavaScript Prototype
- Schema

```
Type : object<
        { Prototype }
        { Constraints }
        { rules }
        { Signature }
>
```

An element like node

Prototype

```
object< xs:QName ? ,
    object<xs:string,xs:string>*,
    node()* >
```

Example

```
object {  "MyElement" ,
      { "attribute" : "value" } ,
            <child>text<child> ,
            text { "mixed" } }
```

An Array like Node

Prototype

```
object< xs:anyType*  >
```

Example

```
object { "one" , 2 ,
    object { document { ... } },
    function () ...
  }
```

An JSON Node ?

Prototype

```
object<
    object { xs:string, xs:anyType } * |
    object* >
```

Example

```
object {
   { "field1" , 10  },
  { "field2 , { "string" , 1.5 , {} }
 }
```

Rules and Constraints

Constraints refine object types

Tight Constraints ease modeling specific types

Loose Constraints allow flexibility

Rules can describe type conversions

Serialization

The Dream ...
of perfect round trip lossless serialization

Wake up!

We lost that a long time ago ...

It works well when you have ...
One Format
One Data Model
Universal Agreement
Fixed Requirements
Forever ....

but ... XML

We still have XML !

It works great ... don't touch it.

So great we want the tools For Everything !

The Solution

- Decouple Serialization from the Data Model
- New types should leverage XPath
- Serialization formats come and go
- New XDM Types should be Format Neutral

Wake up!

We lost that a long time ago ...

It works well when you have ...
One Format
One Data Model
Universal Agreement
Fixed Requirements
Forever ....

but ... XML

We still have XML !

It works great ... don't touch it.

So great we want the tools
For Everything !

The Solution

- Decouple Serialization from the Data Model
- New types should leverage XPath
- Serialization formats come and go
- New XDM Types should be Format Neutral

But why all this work ?

An Historic Metaphor

To evolve is to survive

To abstract is to evolve

We need to try ...

To Save Everything

An Historic Metaphor

To evolve is to survive

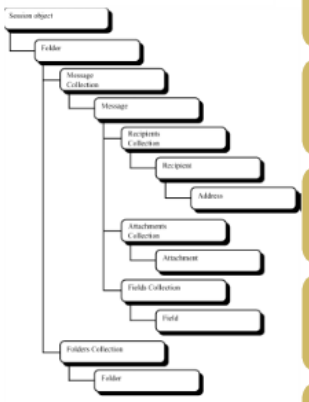
To abstract is to evolve

PAPER TO PAPER

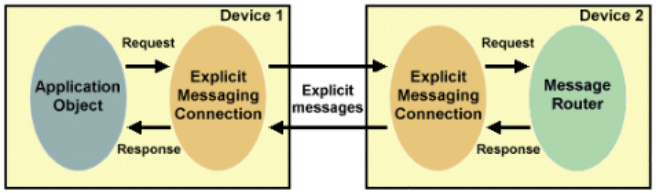
Past

Specific

Paper	•Documents •Key Entry
Files	•GML - 1968 •SGML - 1980
DOM	•XML - 1998 •APIs / XSLT
XDM	•Objects •XQuery
Messages	•Streaming •Data Mapping
Data	• XQuery / XSLT • X* languages
Browser	•JavaScript •XSLT(*) ...
Next Year ?	•Up to us



Mainframe	• Bell Manuals • Key Entry
PDP-7	•Unix 1969 •C - 1970
CPUS	• (K&R) C
Architecture	• C / C99 / C++
VM	•C / Java / C# ..
Cloud	•C / Java / C#
Browser	•JavaScript as a VM •Java / C / Ruby ...
Next Year ?	•Java / C++ / KeWl



Now

Abstract

ANYTHING TO ANYTHING

We need to try ...

To Save Everything

Its not just a theory...

Coincidentally, available now

Version 8.0 - EA 2

Native XDM Object Types
XQuery / XPath / JavaScript

XPath over JSON

```
{
  "a":  {
    "b" : "v1",
    "c1" : 1,
    "c2" : 2,
    "d" : null,
    "e" : {
      "f" : true,
      "g" : [ "x1", "x2", "x3" ]
    }
  }
}
```

```
/a/b => "v1"
/a/c1 => 1
/a/d => null
/a/e/f => true
/a/e/g => ("x1", "x2", "x3")
/a/e/g[2] => "x2"
/a[c1=1] => {"b":"v1", "c1":1, "c2":2,
    "d":null, "e":{"f":true,
      "g":["x1", "x2", "x3"]}}
/a[c1=3] => ()
```

Node Tests

- object-node()
- array-node()
- number-node()
- boolean-node()
- null-node()
- text()

```
$node//number-node() => (1,2)
$node/a/number-node() => (1,2)
$node//text() => ("v1", "x1", "x2", "x3")

$node//object-node() => (
{"a":{"b":"v1", "c1":1, "c2":2, "d":null,
    "e":{"f":true, "g":["x1", "x2", "x3"]}}}
{"b":"v1", "c1":1, "c2":2, "d":null,
    "e":{"f":true, "g":["x1", "x2", "x3"]}}
{"f":true, "g":["x1", "x2", "x3"]} )
```

Named Nodes

```
let $node := array { 1, 2, object { "foo" : 3 } }
return fn:node-name($node/number-node()[. eq 1]) => ()
let $node := xdmp:unquote('{"foo" : "v1"}')
return fn:node-name($node/object-node()) => ()
```

Unnamed Nodes

```
let $node := object { "foo" : "v1", "bar" : array {1, 2} }
return fn:node-name($node/foo) => "foo"
let $node := object {"foo" : "v1", "bar" : array {1, 2} }
return fn:node-name($node/bar[2]) => "bar"
```

Its time Now ...

To Save XML With

NOt Only XML

But why all this work ?

Coincidentally, available now

Version 8.0 - EA2

Native XDM Object Types

XQuery / XPath / JavaScript

XPath over JSON

```
{
 "a":   {
    "b" : "v1",
    "c1" : 1,
    "c2" : 2,
    "d" : null,
     "e":  {
       "f" : true,
       "g" : [ "s1", "s2", "s3" ]
     }
  }
}
```

```
/a/b =>  "v1"
/a/c1 => 1
/a/d => null
/a/e/f => true
/a/e/g => ("s1", "s2", "s3")
/a/e/g[2] => "s2"
/a[c1=1] => {"b":"v1", "c1":1, "c2":2,
      "d":null, "e":{"f":true,
       "g":["s1", "s2",  "s3"]}}
/a[c1=3] => ()
```

Node Tests

- object-node()
- array-node()
- number-node()
- boolean-node()
- null-node()
- text()

```
$node//number-node()  =>  (1,2)
$node/a/number-node()  =>  (1,2)
$node//text()  => ("v1", "s1", "s2", "s3")

$node//object-node()  => (
{"a":{"b":"v1", "c1":1, "c2":2, "d":null,
      "e":{"f":true, "g":["s1", "s2", "s3"]}}}
{"b":"v1", "c1":1, "c2":2, "d":null,
    "e":{"f":true, "g":["s1", "s2", "s3"]}}
{"f":true, "g":["s1", "s2", "s3"]}  )
```

- object-node()
- array-node()
- number-node()
- boolean-node()
- null-node()
- text()

```
$node//number-node()  =>  (1,2)
$node/a/number-node()  =>  (1,2)
$node//text()  => ("v1", "s1", "s2", "s3")

$node//object-node()  => (
{"a":{"b":"v1", "c1":1, "c2":2, "d":null,
      "e":{"f":true, "g":["s1", "s2", "s3"]}}}
{"b":"v1", "c1":1, "c2":2, "d":null,
    "e":{"f":true, "g":["s1", "s2", "s3"]}}
{"f":true, "g":["s1", "s2", "s3"]}  )
```

Named Nodes

```
let $node := array  { 1, 2,  object { "foo" : 3 } }
return fn:node-name($node//number-node()[. eq 1]) => ()
let $node := xdmp:unquote('{"foo" : "v1"}')
return fn:node-name($node/object-node()) => ()
```

Unnamed Nodes

```
let $node := object {"foo" : "v1", "bar" : array {1, 2} }
return fn:node-name($node/foo) => "foo"
let $node := object {"foo" : "v1", "bar" : array {1, 2} }
return fn:node-name($node/bar[2]) => "bar"
```

Its time Now ...

To Save XML With

Not Only XML

Not Only XML

Coincidentally, available now

Version 8.0 - EA2

Native XDM Object Types
XQuery / XPath / JavaScript

XPath over JSON

```
{
 "a":  {
   "b" : "v1",
   "c1" : 1,
   "c2" : 2,
   "d" : null,
   "e":  {
     "f" : true,
     "g" : [ "s1", "s2", "s3" ]
   }
 }
}
```

```
/a/b =>  "v1"
/a/c1 => 1
/a/d => null
/a/e/f => true
/a/e/g => ("s1", "s2", "s3")
/a/e/g[2] => "s2"
/a[c1=1] => {"b":"v1", "c1":1, "c2":2,
     "d":null, "e":{"f":true,
       "g":["s1", "s2",  "s3"]}}
/a[c1=3] => ()
```

Node Tests

- object-node()
- array-node()
- number-node()
- boolean-node()
- null-node()
- text()

```
$node/number-node()  =>  (1,2)
$node/a/number-node()  =>  (1,2)
$node/text()  => ("v1", "s1", "s2", "s3")

$node/object-node()  => (
{"a":{"b":"v1", "c1":1, "c2":2, "d":null,
      "e":{"f":true, "g":["s1", "s2", "s3"]}}}
{"b":"v1", "c1":1, "c2":2, "d":null,
   "e":{"f":true, "g":["s1", "s2", "s3"]}}
{"f":true, "g":["s1", "s2", "s3"]}  )
```

Named Nodes

```
let $node := array  { 1, 2,  object { "foo" : 3 } }
return fn:node-name($node/number-node()[. eq 1]) => ()
let $node := xdmp:unquote('{"foo" : "v1"}')
return fn:node-name($node/object-node()) => ()
```

Unnamed Nodes

```
let $node := object {"foo" : "v1", "bar" : array {1, 2} }
return fn:node-name($node/foo) => "foo"
let $node := object {"foo" : "v1", "bar" : array {1, 2} }
return fn:node-name($node/bar[2]) => "bar"
```

Its time Now ...

To Save XML With

Not Only XML

Not Only XML

David A. Lee
dlee@marklogic.com

Extending the relevance of XPath by breaking the chains of the DOM