SAXONICA.COM
XSLT AND XQUERY PROCESSING

A proposal for XSLT 4.0

Michael Kay, Saxonica
XML Prague 2020

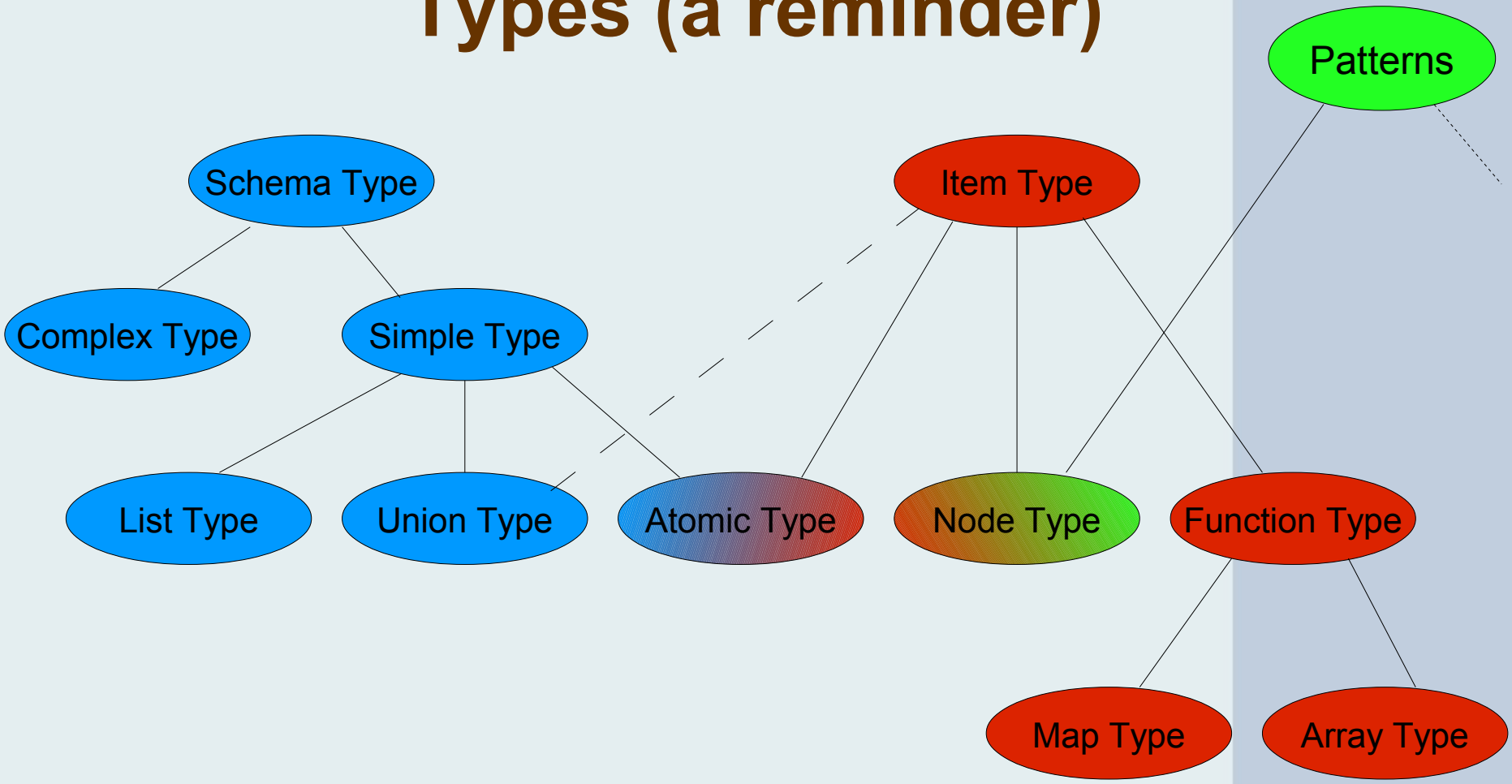
Why?

- There's a substantial user community
  - Top-25 language on StackOverflow
- Users are doing increasingly ambitious things
- There's a business case for investment
- Unfinished business from XSLT 3.0
  - Especially in the area of maps, arrays, JSON, and higher-order functions
- And there's always "low-hanging fruit"

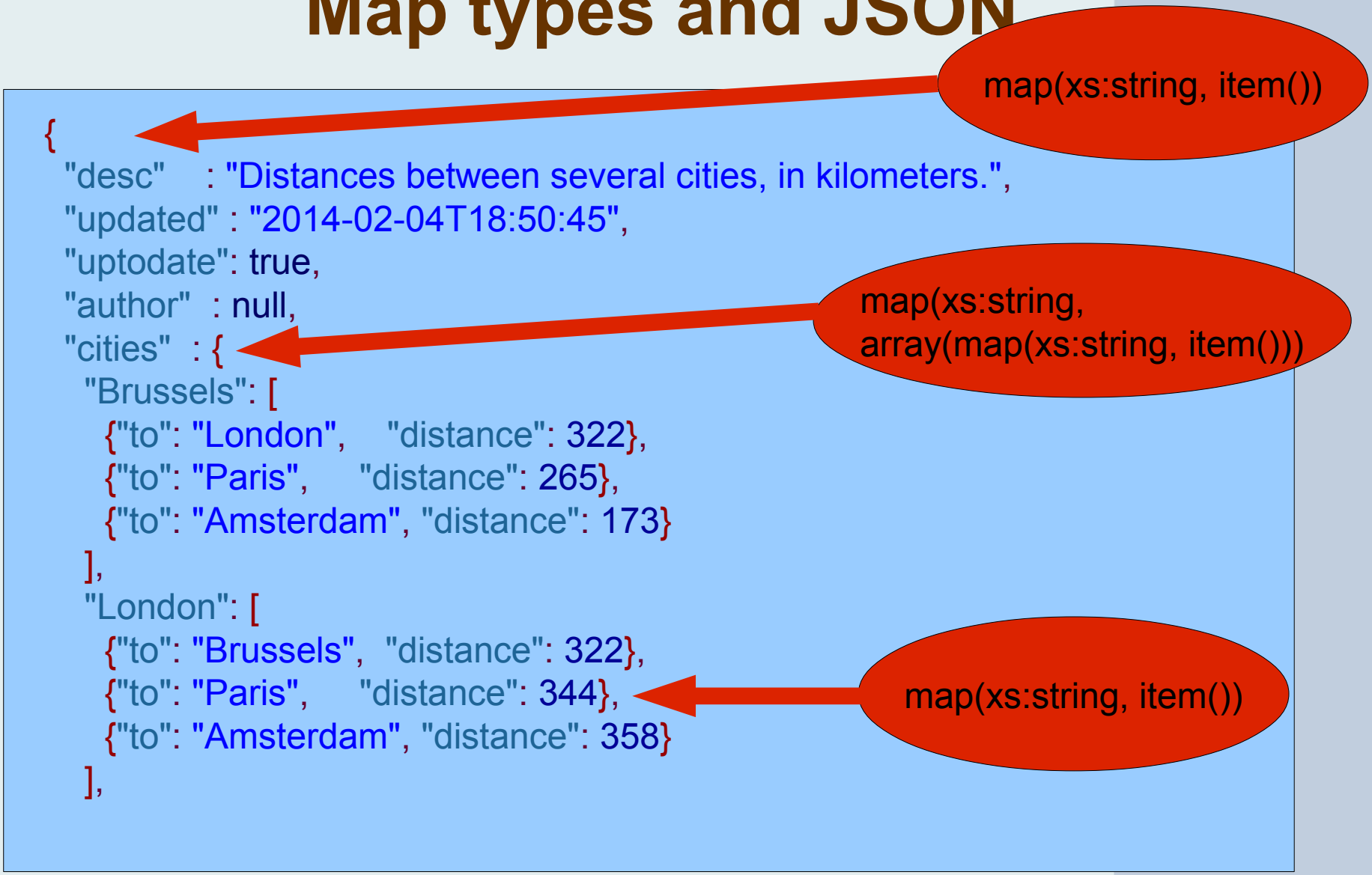
Unfinished Business

- Exploiting maps and arrays
  - stronger typing and pattern matching
  - richer set of operations/functions
- Exploiting higher-order functions
  - more HO functions in standard library
  - easier syntax for inline functions
- XSLT/XPath interoperation
  - XSLT is still too focused on XML data

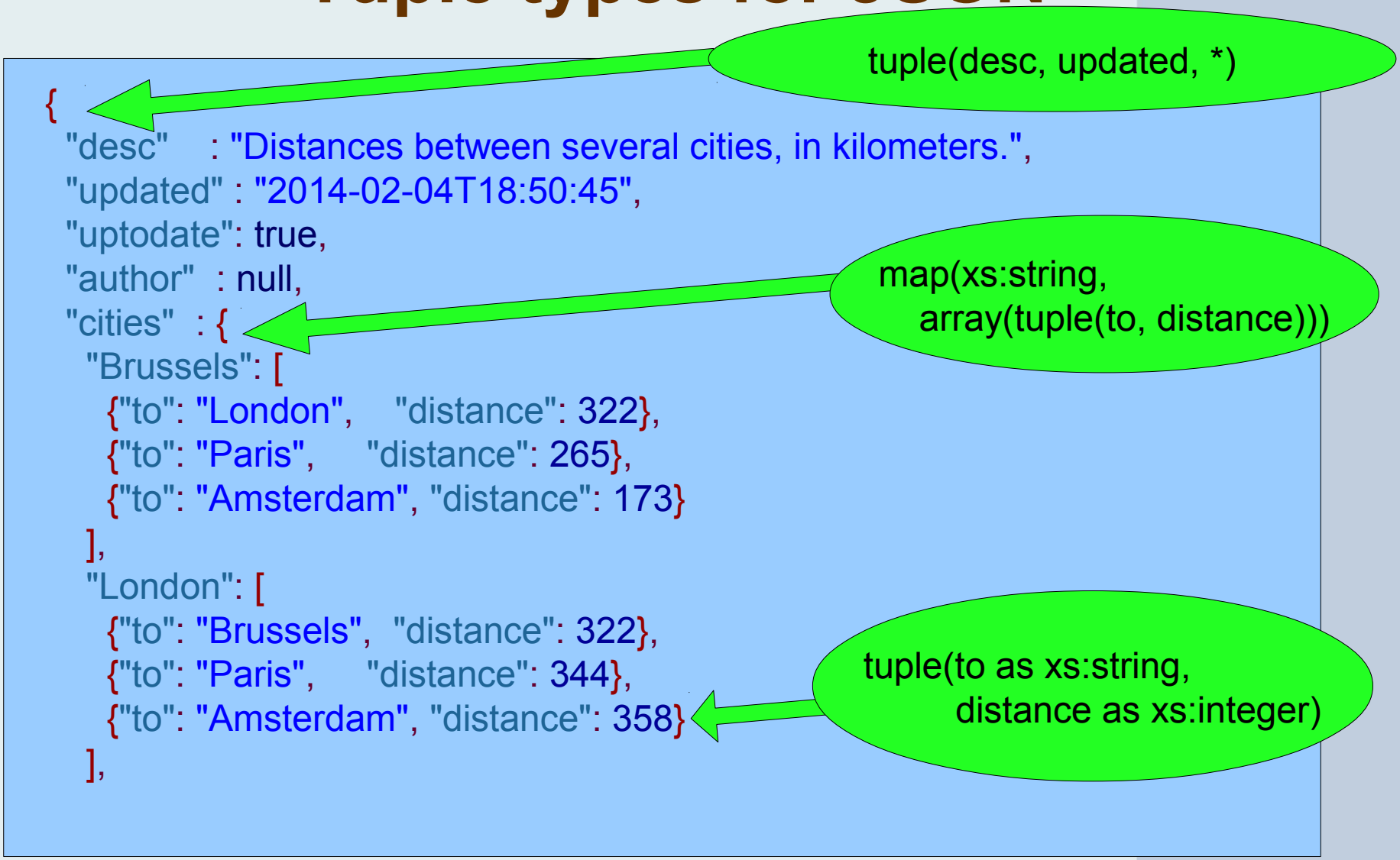
Types (a reminder)



Map types and JSON



Tuple types for JSON



Tuples

```
<xsl:param
      name="site-employees"
      as="tuple(location as xs:string,
                emps as element(employee)*)"/>
```

- Tuple types are a new way of describing (or constraining) maps
- Used as item types and as patterns
- No new operations

Tuples: technical details

- A map with statically known key values
  - tuple(longitude as xs:double, latitude as xs:double)
- Keys are strings (using quotes if necessary)
  - tuple("first name", "last name", "date of birth")
- Field types default to item()+
- Entries may be optional
  - tuple (first as xs:string, middle as xs:string?, last as xs:string)
- Type may be extensible
  - tuple (a as xs:integer, b as element(), *)

See also: JSON Schema

Tuple types as Patterns

```
<xsl:template match="tuple(first, last, 'date of birth', *)">
  <person firstName="{?first}"
          lastName="{?last}"
          dateOfBirth="{?'date of birth'}">
    <xsl:apply-templates select="?children"/>
  </person>
</xsl:template>

<xsl:template match="array(tuple(first, last, 'date of birth', *))">
  <children>
    <xsl:apply-templates select="?*"/>
  </children>
</xsl:template>
```

Type aliases

```
<xsl:item-type name="person" as="tuple(first, last, 'date of birth', *)"/>

<xsl:template match="type(person)">
  <person firstName="{?first}"
          lastName="{?last}"
          dateOfBirth="{?'date of birth'}">
    <xsl:apply-templates select="?children"/>
  </person>
</xsl:template>

<xsl:template match="array(type(person))">
  <children>
    <xsl:apply-templates select="?*"/>
  </children>
</xsl:template>
```

Inline union types

```
<xsl:function name="f:parse-date"
      as="union(xs:date, xs:time, xs:dateTime)">

<xsl:item-type name="option-name"
      as="union(xs:string, xs:QName)"/>

<xsl:variable name="options"
      as="map(type(option-name), item()*)"/>

<xsl:sequence
      select="$in cast as union(xs:integer, xs:string)"/>
```

Higher Order Functions

- One of the most powerful additions to XSLT 3.0
- Under-used, because:
  - optional feature
  - clumsy syntax
  - limited repertoire of HOFs in the standard function library

Concise inline functions
(1) Dot Functions

- Instead of
  - sort(//employees, (),
    function($emp as item())
    as xs:string {$emp/lastName})
- allow
  - sort(//employees, (), .{lastName})

Concise inline functions
(2) Underscore Functions

- Instead of

```
fn:fold-left(1 to 5, 0,
    function($a as xs:integer, $b as xs:integer)
    as xs:integer { $a + $b })
```

- allow

```
fn:fold-left(1 to 5, 0, _{ $1 + $2 })
```

New Higher-Order Functions
replace-with()

```
replace-with("Chapter 13", "[0-9]+", .{xs:integer(.) + 1})
```

- "Chapter 14"

New Higher-Order Functions
map:replace()

```
let $M := map{'height': 12, 'width': 93}
let $toMM := .{. * 25.4 || 'mm'}
return

$M => map:replace("height", $toMM)
   => map:replace("width", $toMM)
```

➤ "map{'height': '304.8mm', 'width': '2362.2mm'}"

New Higher-Order Functions
items-before(), -to(), &c

```
let $in := (<p/>, <q/>, <r/>, <s/>, <t/>)

items-before($in, .{self::s})
```
➤ <p/><q/><r/>

```
items-to($in, .{self::s})
```
➤ <p/><q/><r/><s/>

```
items-from($in, .{self::s})
```
➤ <s/><t/>

```
items-after($in, .{self::s})
```
➤ <t/>

New Higher-Order Functions
highest(), lowest()

highest(//employee, .{salary}) / fullName

➤ "Jeff Bezos"

Flexible Function Arity

- Parameter defaults in xsl:function

```
<xsl:function name="f:compare-widgets">
   <xsl:param name="$widget1"/>
   <xsl:param name="$widget2"/>
   <xsl:param name="$options" select="map{}"/>
</xsl:function>
```

- Function coercion to accept a lower-arity function:
  - fn:filter(//employees, true#0)

Conditionals

```
@price - (@discount otherwise 0)
```

```
<xsl:if test="empty($x)"
        then="()"
        else="head($x) + f(tail($x))"/>
```

```
<xsl:choose>
   <xsl:when test="$a = 'small' " select="1"/>
   <xsl:when test="$a = 'medium' " select="2"/>
   <xsl:when test="$a = 'large' " select="3"/>
   <xsl:otherwise select="4"/>
</xsl:choose>
```

Other Goodies (see paper...)

- Namespaces:
  - /a/b/c means /*:a/*:b/*:c
  - unprefixed type names match xs:*
- Modes:
  - xsl:template as child of xsl:mode
  - xsl:mode as="*return-type*"
- Instructions to process arrays
- Node constructor functions
  - new-attribute('code', '123')

Questions

- Q: What's the plan?
- A: You tell me.

Any Other Questions?