XPath 3.1 in the Browser

John Lumley
jωL Research/
Saxonica

Debbie Lockett,
Michael Kay
Saxonica

Synopsis

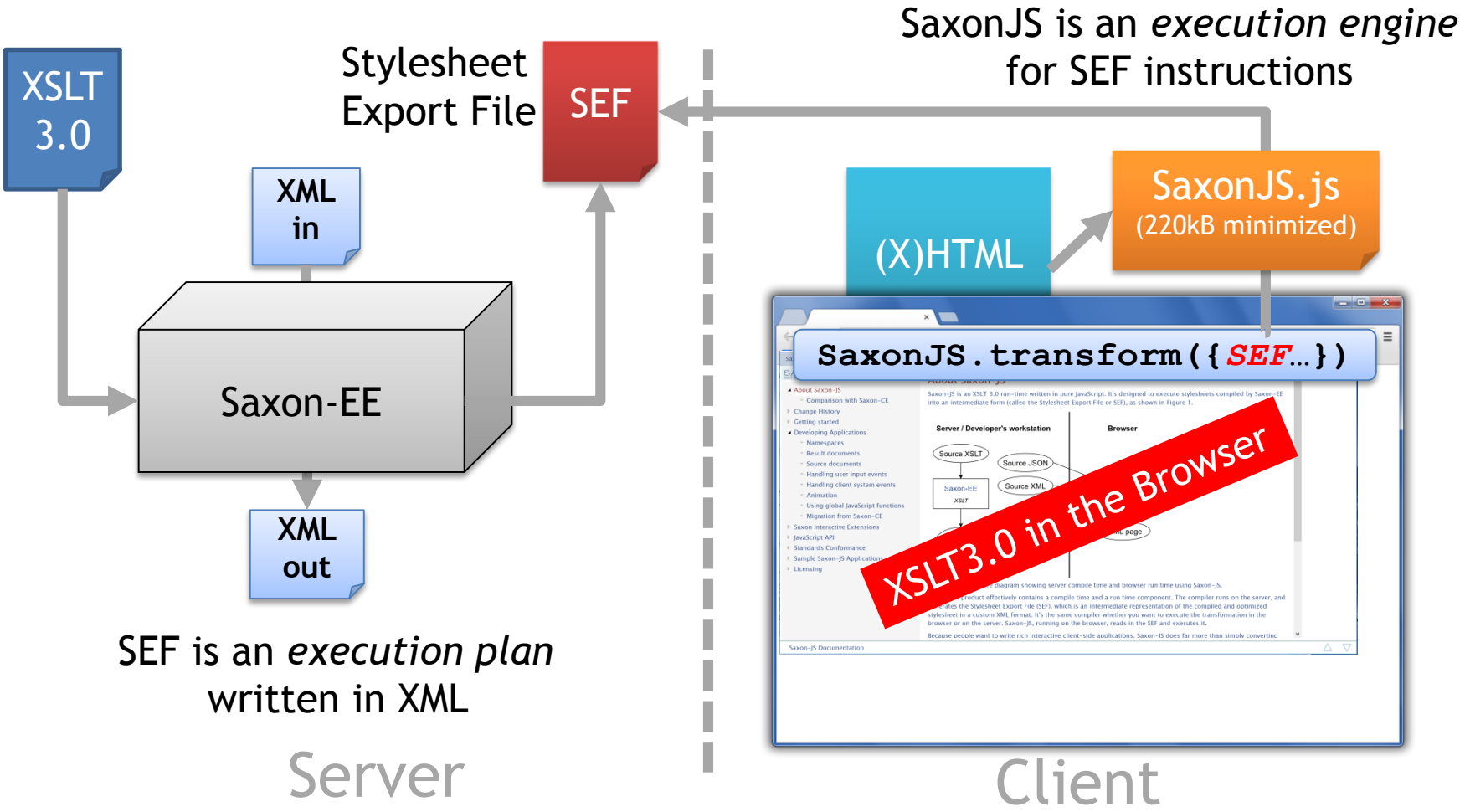
Using the *Saxon-JS* JavaScript runtime, generate *in-browser* a suitable execution plan for an XPath expression to support dynamic XPath evaluation

- Context
- Parsing the XPath
- Compiling to Saxon-JS instructions
- Testing & Results
- Demonstration
- Conclusions

XPath-ish in the browser

- XSLT/XPointer → XPath 1.0 (1999)
  - Browser-native XSLT1.0 *(killed by the mobile phone)*
  - JavaScript native (2010+): "*Wicked good XPath*", *XPath-js*
- XSLT 2.0/XQuery → XPath 2.0 (2007)
  - *Saxon-CE* (Saxon Java crosscompiled/GWT → JavaScript)
  - JavaScript native: *jQuery.xpath* (Illinsky), *XQIB* (XQuery)
- XSLT/XQuery 3.x → XPath 3.x (2015)
  - *Saxon-JS*
    (Saxon compilation of XSLT; JavaScript execution engine)

Saxon-JS



Stylesheet Export File

```
let $t:=current-time(),
 $hours:= hours-from-time($t) mod 12,
 $quarters := minutes-from-time($t) idiv 15
 return (format-time($t,'[H]:[m]'),
        (1 to (if($hours eq 0) then 12 else $hours)) ! 'BONG',
        (1 to $quarters) !'ding')
```

→ ("13:48", "BONG", "ding","ding","ding")

```
<let name="Q{}t" slot="1"…>
 <fn name="current-time"/>
 <let name="Q{}hours" slot="2">
  <arith op="mod" calc="i%i">
    <fn name="hours-from-time">
      <varRef name="Q{}t" slot="1" diag="0|0||hours-from-time"/>
    </fn>
    <int val="12"/>
  </arith>
  <let name="Q{}quarters" slot="3">
    <arith op="idiv" calc="i~i">
     <fn name="minutes-from-time">
       <varRef name="Q{}t" slot="1" diag="0|0||minutes-from-time"/>
     </fn>
     <int val="15"/>
    </arith>
    <sequence…/> </…
```

Variable slots

Nested Lets

Type-determined operations

```
<sequence>
 <fn name="format-time">
   <varRef name="Q{}t" slot="1" diag="0|0||format-time"/>
   <str val="[H]:[m]" diag="0|1||format-time"/>
 </fn>
 <forEach>
   <to>
     <int val="1"/>
     <choose>
       <vc card="1:1" comp="…" op="eq">
          <varRef name="Q{}hours" slot="2" diag="1|0||eq"/>
          <int val="0" diag="1|1||eq"/>
       </vc>
       <int val="12"/>
       <true/>
       <cast as="xs:integer?" diag="1|1||to">
         <varRef name="Q{}hours" slot="2"/>
       </cast>
     </choose>
   </to>
   <str val="BONG"/>
 </forEach>
 <forEach>
  <to> …
  <str val="ding"/>
 </forEach>
</sequence>
```

Function call

if → choose

Type-inferred cast

Saxon-JS features

XSLT3.0	Most functionality
New XDM types	`map(*), array(*), JSON`
Functions & Operators	Most of the 3.1 set

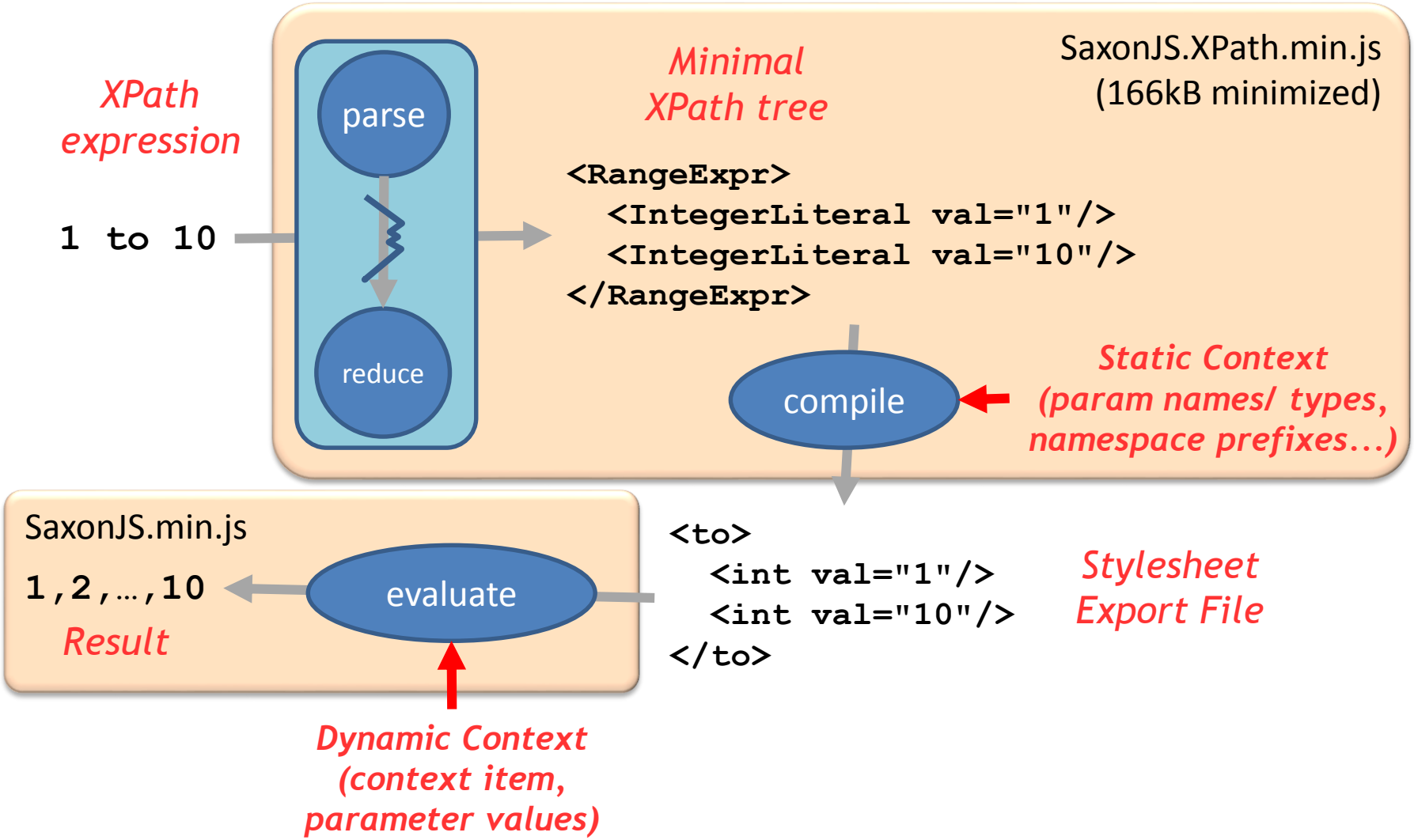
Saxon-JS limitations

No streaming! No packaging!	Life's complicated enough as it is
No higher-order functions	No `function(*)` type
No accumulators	
No **`xsl:evaluate`**	No '*XPath compiler*' in Saxon-JS runtime

Saxon-JS *requires* the execution plan to be *very* correct

- Function argument arity and type
- Operand types
- Run-time casts and type checks

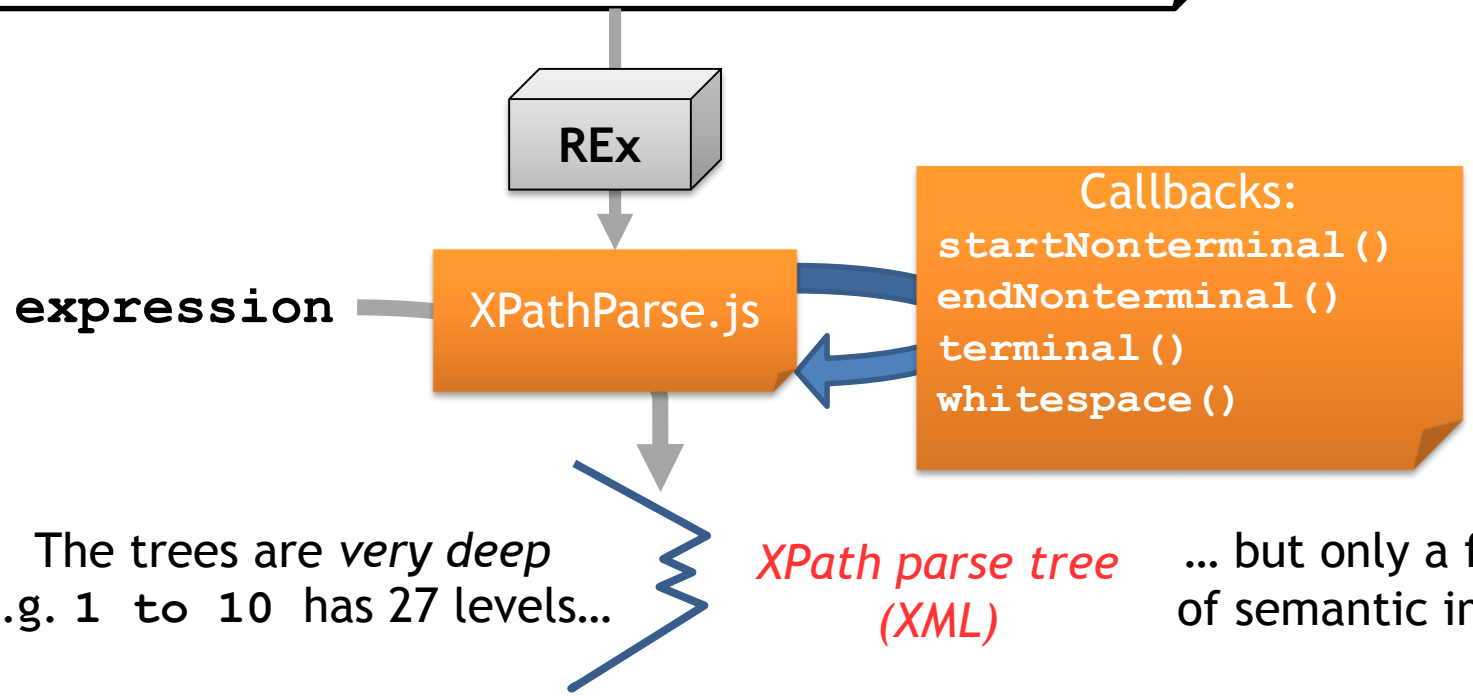
Main processes



Parsing the expression

XPath 3.1 EBNF

```
Xpath ::=
…
OrExpr    ::= AndExpr ( 'or' AndExpr )*
AndExpr   ::= ComparisonExpr ( 'and' ComparisonExpr )*
…
```



The trees are *very deep*
e.g. `1 to 10` has 27 levels…

XPath parse tree (XML)

… but only a fraction is of semantic importance

Reducing the tree

`1 * 4 to ceiling(10.1)`

```
… <RangeExpr>
  …<MultiplicativeExpr>
  …   <IntegerLiteral>1</IntegerLiteral> …
        <TOKEN>*</TOKEN>
  …   <IntegerLiteral>4</IntegerLiteral> …
   </MultiplicativeExpr>...
   <TOKEN>to</TOKEN>
   … <FunctionCall>
      <FunctionEQName>
       <FunctionName>
         <QName>ceiling</QName>
       </FunctionName>
      </FunctionEQName>
      <ArgumentList>
       <TOKEN>(</TOKEN>
       <Argument>
       … <DecimalLiteral>10.1</DecimalLiteral> …
       </Argument>
       <TOKEN>)</TOKEN>
      </ArgumentList>
    </FunctionCall> …
  </RangeExpr>
```

parameters → attributes

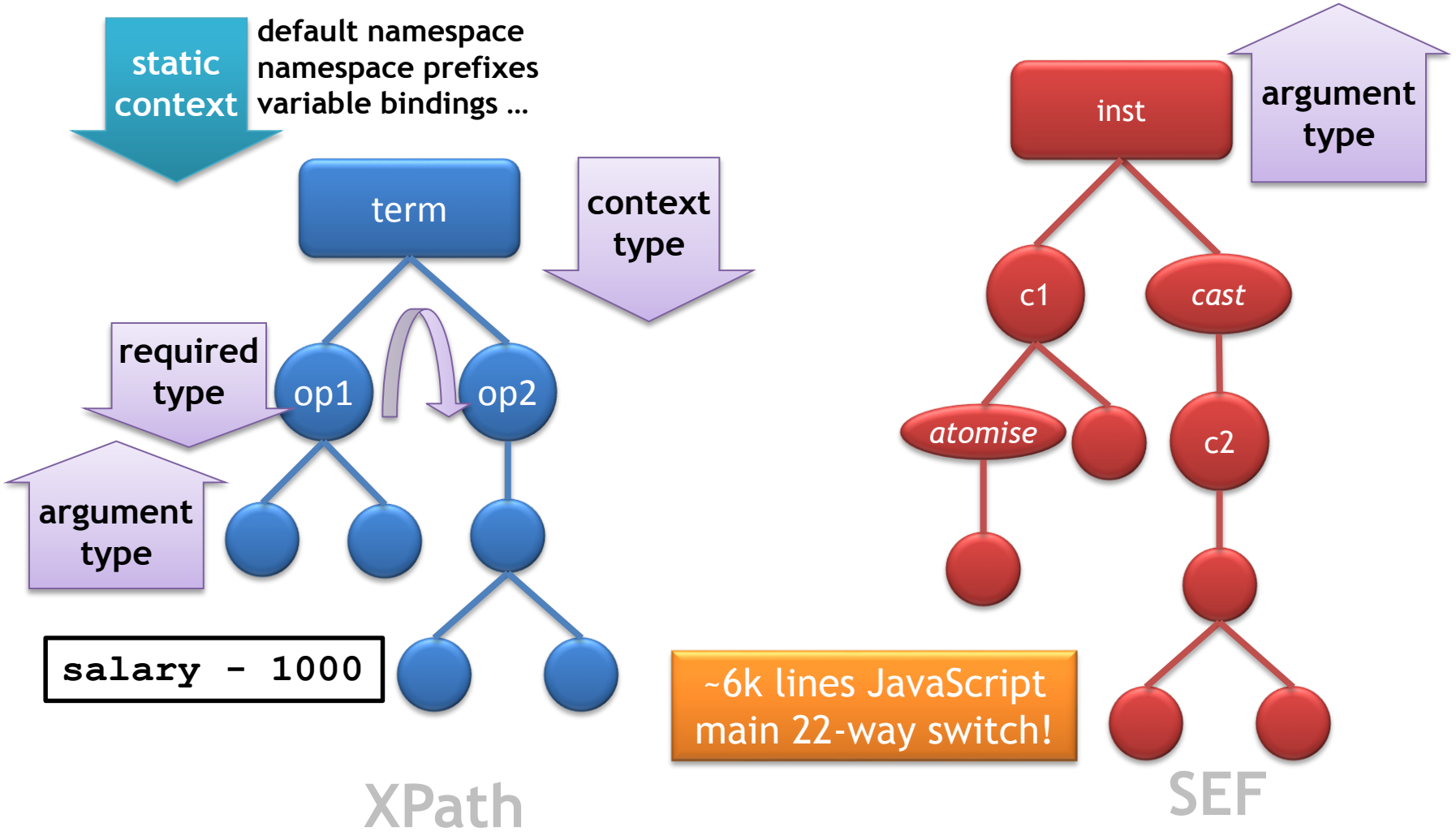
redundant tokens removed

single-child reduction

canonical nesting

```
<RangeExpr>
 <MultiplicativeExpr op="*">
  <IntegerLiteral value="1"/>
  <IntegerLiteral value="4"/>
 </MultiplicativeExpr>
 <FunctionCall name="ceiling">
   <DecimalLiteral value="10.1"/>
 </FunctionCall>
</RangeExpr>
```

Generating the execution plan



Conversion sample

```
function prepare(node, context) {
… switch(node.nodeName) {…
    case "RangeExpr": {
      n = makeElement("to");
      lhs = prepare(node.firstChild, context);
      rhs = prepare(node.lastChild, context);
      var diag = diagnostic("BINARY_EXPR", 0, "", "to");
      var role = makeTypeRole("XPTY0004", diag);
      n.appendChild(typeCheck(lhs,
        SequenceType.OPTIONAL_INTEGER, role, diag, context));
      diag = diagnostic("BINARY_EXPR", 1, "", "to");
      role = makeTypeRole("XPTY0004", diag);
      n.appendChild(typeCheck(rhs,
        SequenceType.OPTIONAL_INTEGER, role, diag, context));
      setType(n, SequenceType.INTEGER_SEQUENCE);
      return n;
    }
…
```

Static type-checking

- *The* critical component:
  - Detect semantic (type) errors
  - Add type coercion instructions (`cast`)
  - Add run-time type checking instructions
- Transcription of Saxon `class TypeChecker`
  - Full type system (`baseType + cardinality`)
- Full function signature catalogue

Function calls

<fn name="*fn-name*">*arg*$_1$, *arg*$_2$, … </>

- Function catalogue:
  - Generated from W3C spec.
  - Signatures – arity, argument types, defaults
  - Check correct calls, needed casts
- Type constructors (e.g. **`xs:decimal(1.01)`**) converted to cast instructions
- Implicit casts
  - **`string-length()`** → **`string-length(string(.))`**

Variables

- Invocation parameters or **`let`** subjects
- Added to (static) context at definition
  - Value slot allocation
  - Inferred type
  - Scope **`following-sibling::*/descendant-or-self::*`**
- Interpolation through **`varRef`** instruction
  - Type & slot reference from context

Development with *the oracle*

XPath expression

```
1 to 10
```

XSLT 3.0

Saxon-EE

SEF

parse

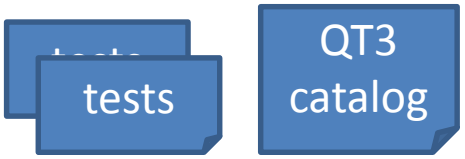
? ==

```
<RangeExpr>
 <IntegerLiteral value="1"/>
 <IntegerLiteral value="10"/>
</RangeExpr>
```

compile

```
<to>
  <int val="1"/>
  <int val="10"/>
</to>
```

Testing via QT3



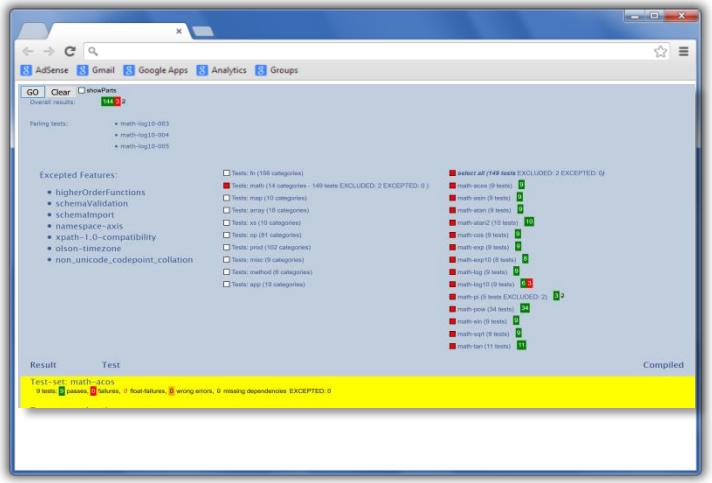
QT3 catalog

```
<test-case name="op-concat-19">
…
  <test>12 || 34 - 50</test>
  <result>
    <all-of>
      <assert-eq>"12-16"</assert-eq>
      <assert-type>xs:string</assert-type>
    </all-of>
  </result>
</test-case>
```

XSLT

`xsl:evaluate`

`check-assert()`



Saxon-EE

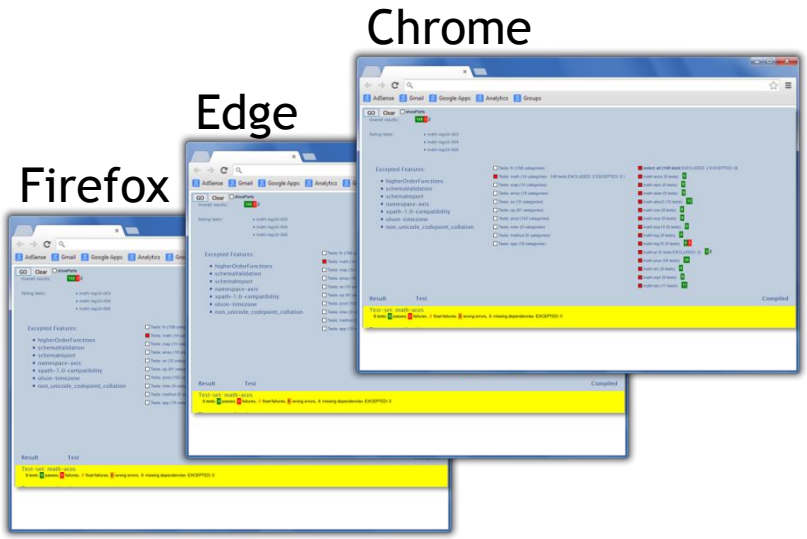
SEF

Chrome:

18444 passes
115 failures
173 wrong errors

2mins 45 seconds

Comparing across browsers



Save As

<html>+
.webarchive

XSLT

...

QT3tests - Saxon JS - browser comparison

Browsers tested:

Browser	Results					
Chrome	18347	116	173	~~4212~~	*49*	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (K
Edge	18325	138	173	~~4212~~	*49*	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (K
Unknown(IE?)	18303	188	173	~~4212~~	*21*	Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NI
Firefox	18344	117	175	~~4212~~	*49*	Mozilla/5.0 (Windows NT 10.0; WOW64; rv:50.0) Gecko/20100101 I
Opera	18347	116	173	~~4212~~	*49*	Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KH
Safari	18315	177	172	~~4212~~	*21*	Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/534.57.2 (KH

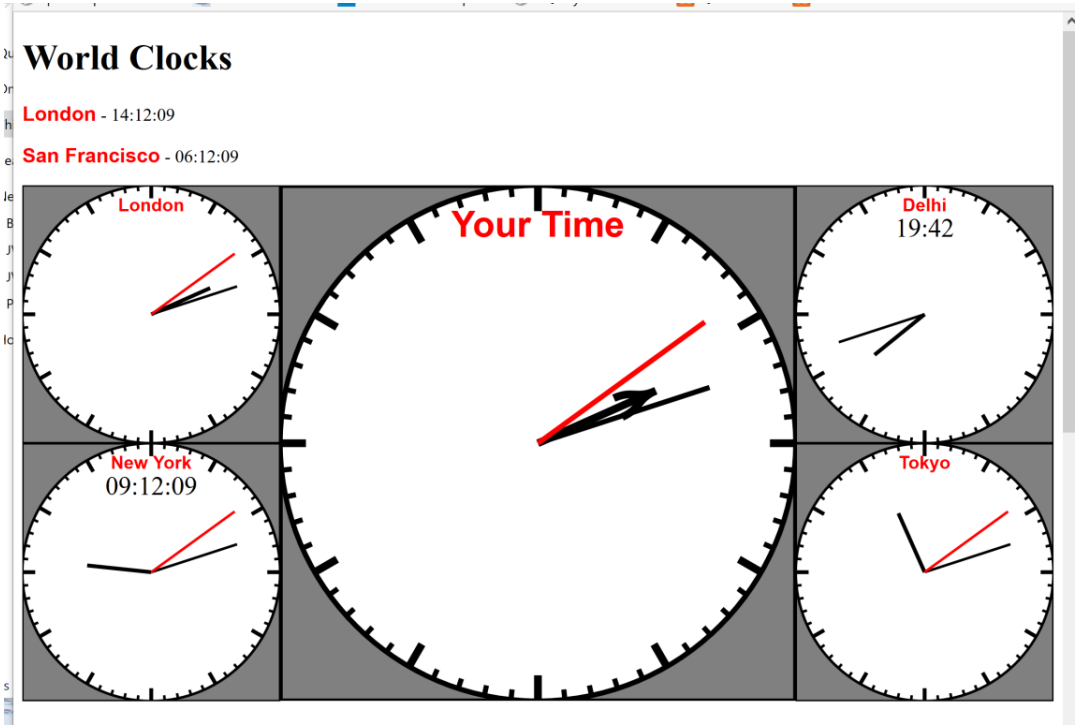
Excluded features **Excepted test-sets**

	Safari	42	3			4	3	✓✓✓✗✓✓	fn-unparsed-text-available-050	
fn-unparsed-text-lines	Chrome	36	7		5	4	3	C E U F O S	Test	
								✗✗✗✗✗✗	fn-unparsed-text-lines-038	fn-unparsed-text-lines-007 Will generate runtime error in untaken conditional
	Edge	36	7		5	4	3	✗✗✗✗✗✗	fn-unparsed-text-lines-049	
								✗✗✗✗✗✗	fn-unparsed-text-lines-050	fn-unparsed-text-lines-009 Will generate runtime error in untaken conditional
	Unknown(IE?)	34	9		5	4	3	✗✗✗✓✗✗	fn-unparsed-text-lines-051	
								✗✗✗✓✗✗	fn-unparsed-text-lines-052	fn-unparsed-text-lines-011 Will generate runtime error in untaken conditional
	Firefox	34	7		7	4	3	✗✗✗✗✗✗	fn-unparsed-text-lines-053	
								✗✗✗✗✗✗	fn-unparsed-text-lines-054	
	Opera	36	7		5	4	3	✓✓✗✓✓✓	fn-unparsed-text-lines-031	
								✓✓✗✓✓✓	fn-unparsed-text-lines-032	
	Safari	36	7		5	4	3	✓✓✓✗✓✓	fn-unparsed-text-lines-043	
								✓✓✓✗✓✓	fn-unparsed-text-lines-044	
fn-upper-case	Chrome	25					4	C E U F O S	Test	
	Edge	24	1				4	✓✗✓✓✓✓	fn-upper-case-20	fn-upper-case-18 Latin Eszett (German beta) to "SS" capital

A pure JavaScript API

`XPath.evaluate(xpath,contextItem?,options?)`

```
function setClocks() {
  var clocks = SaxonJS.XPath.evaluate(
    ".//*[tokenize(@class)='clock']",
     thisDoc,
     {resultForm:"array"});
  for(var i =0;
       i &lt; clocks.length;
       i++) {
    setClock(clocks[i]);
  }
}
```



Future

- Optimisation?
  - Faster parse tree reduction
  - Constant sub-expression evaluation
    - Only really worth it for repeated execution
    - Split to `compile()` and `eval()` in JS API
- JSON SEF format?
- Compiler in XSLT?
- `fn:transform()` in the browser?

Conclusions

- With an XPath *instruction execution engine*, dynamic in-browser XPath upto v3.1 is possible with
  1. An efficient and accurate XPath parser
  2. Very accurate type checking code.
  3. An *oracle* to show correct execution plans
  4. Early construction of an in-browser QT3 test harness
- We have shown very high conformance levels and rapid execution speeds

OTHER MATERIAL

introductory apologies

- I assume you have some familiarity with XPath
- Examples may be daft, but they're short

If not, then this talk might still amuse

I wish to preserve your sanity