Processing Lax XML Element Trees

Phil Fearon – DeltaXML Ltd (Senior Software Engineer)

Gursheen Kaur – DeltaXML Ltd (Software Developer)

DELTAXML

Lax vs Strict – From the Processing Perspective

Lax Content Model

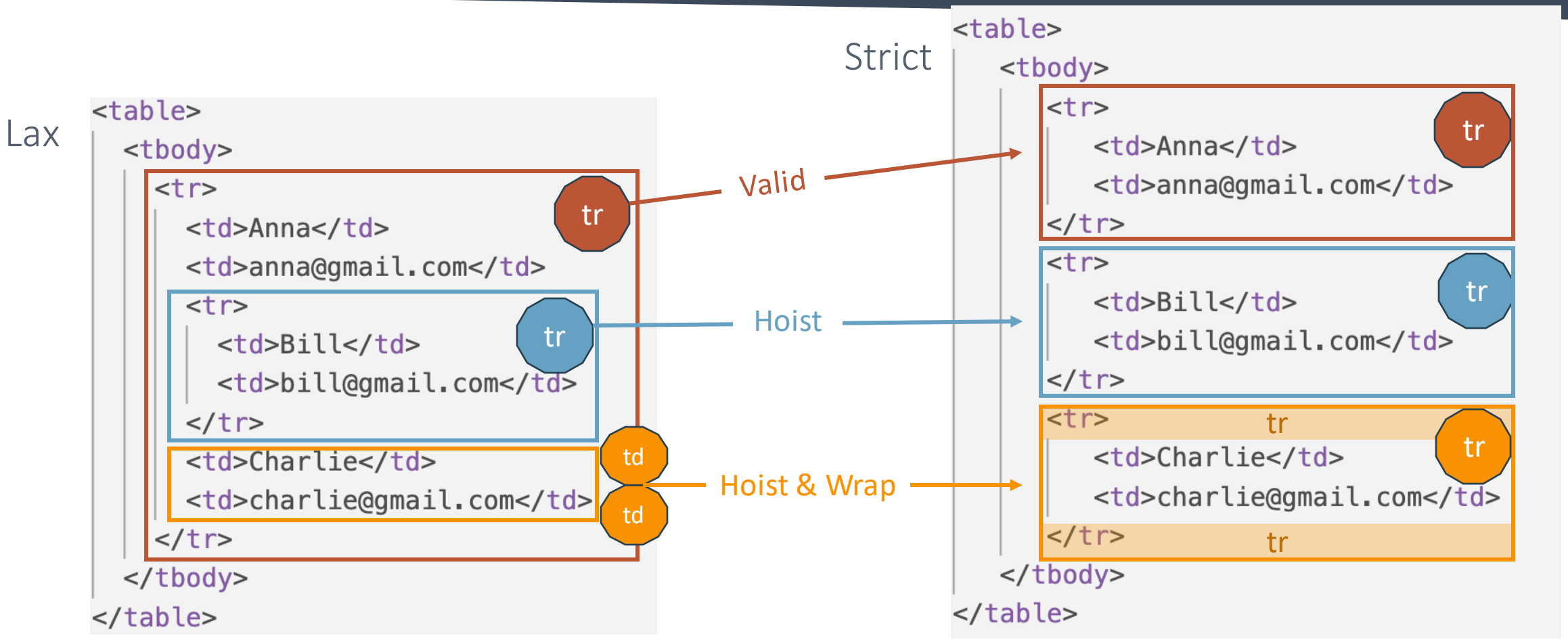
- Many conditional branches
- Complex processing logic
- Reduced opportunity for code reuse
- Comparison of is difficult

Strict Content Model

- Consistent input structure
- Improved processing performance
- Simpler, more robust, code
- Improved test coverage

Lax to Strict – HTML Tables

1. *tr* element is a child of *tbody*
2. *tr* element is a child of tr
3. two *td* elements follow the *tr*

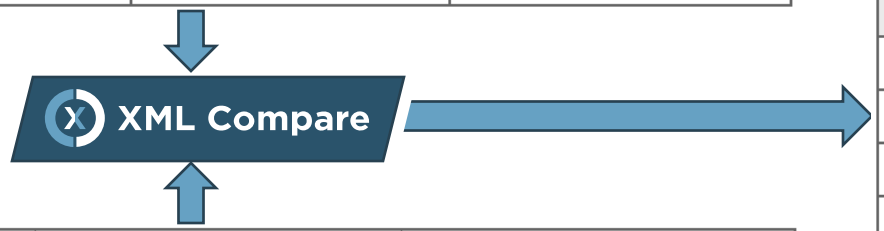


DELTAXML

Goal: Compare Lax HTML Tables

With different element structures AND different table-cell spans

Sr	Name	Code	Email	DOB
1	Joe and Anna GL...			1-1-1985
2				2-2-1985
3	Chris	GL3	chris@gmail.com	3-3-1985 4-4-1985
4	Dave	GL4	dave@gmail.com	



Sr	Name	Email	DOB
1	Joe and Anna and Chris		1-1-1985
2			2-2-1985
3			3-3-1985 4-4-1986
4	Dave	dave@gmail.com	

Sr	Name	Code	Email	DOB
1	Joe and Anna GL...ChrisGL3chris@gmail.com and Chris			1-1-1985
2				2-2-1985
3				3-3-1985 4-4-1985 1986
4	Dave	GL4	dave@gmail.com	

DELTAXML

HTML Table ‘Normalizer’ Considerations

- Equivalence with HTML Standard’s Table Processing Model
- Positioning within XSLT pipeline
- Additional Requirements
- Relative Performance
- Non-Requirements

- Off-the-shelf HTML parser?
- How to Test and Verify?

XSLT: Equivalence with HTML Table Processing Model

HTML

Living Standard — Last Updated 30 June 2023

13 The HTML syntax — Table of Contents

13.2 Parsing HTML documents

13.2.6 Tree construction

13.2.6.4 The rules for parsing tokens in HTML content

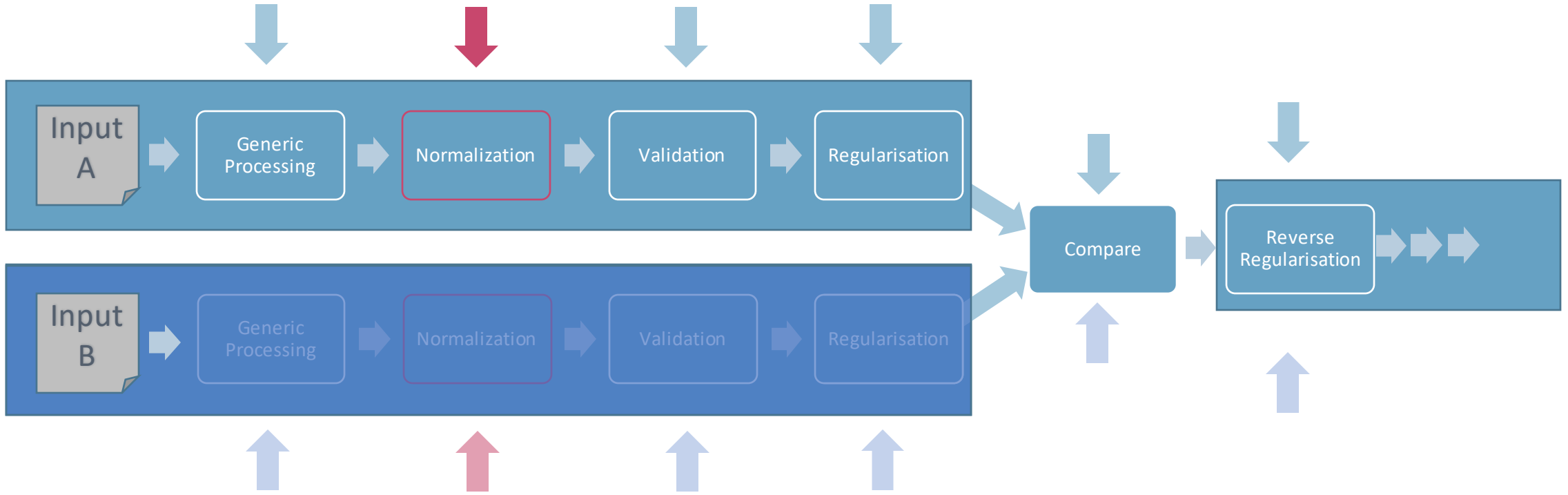
13.2.6.4.9 The "in table" insertion mode
13.2.6.4.10 The "in table text" insertion mode
13.2.6.4.11 The "in caption" insertion mode
13.2.6.4.12 The "in column group" insertion mode
13.2.6.4.13 The "in table body" insertion mode
13.2.6.4.14 The "in row" insertion mode
13.2.6.4.15 The "in cell" insertion mode

XSL Transformations (XSLT) Version 3.0

```
<xsl:variable name="expectedChildren"
              as="map(xs:string, xs:string+)"
  select="map {
    'table':    (
      'caption','colgroup','tbody',
      'thead','tfoot'),
    'tbody':    ('tr'),
    'thead':    ('tr'),
    'tfoot':    ('tr'),
    'tr':       ('td','th'),
    'td':       (''),
    'th':       (''),
    'colgroup': ('col')
  }"/>
```

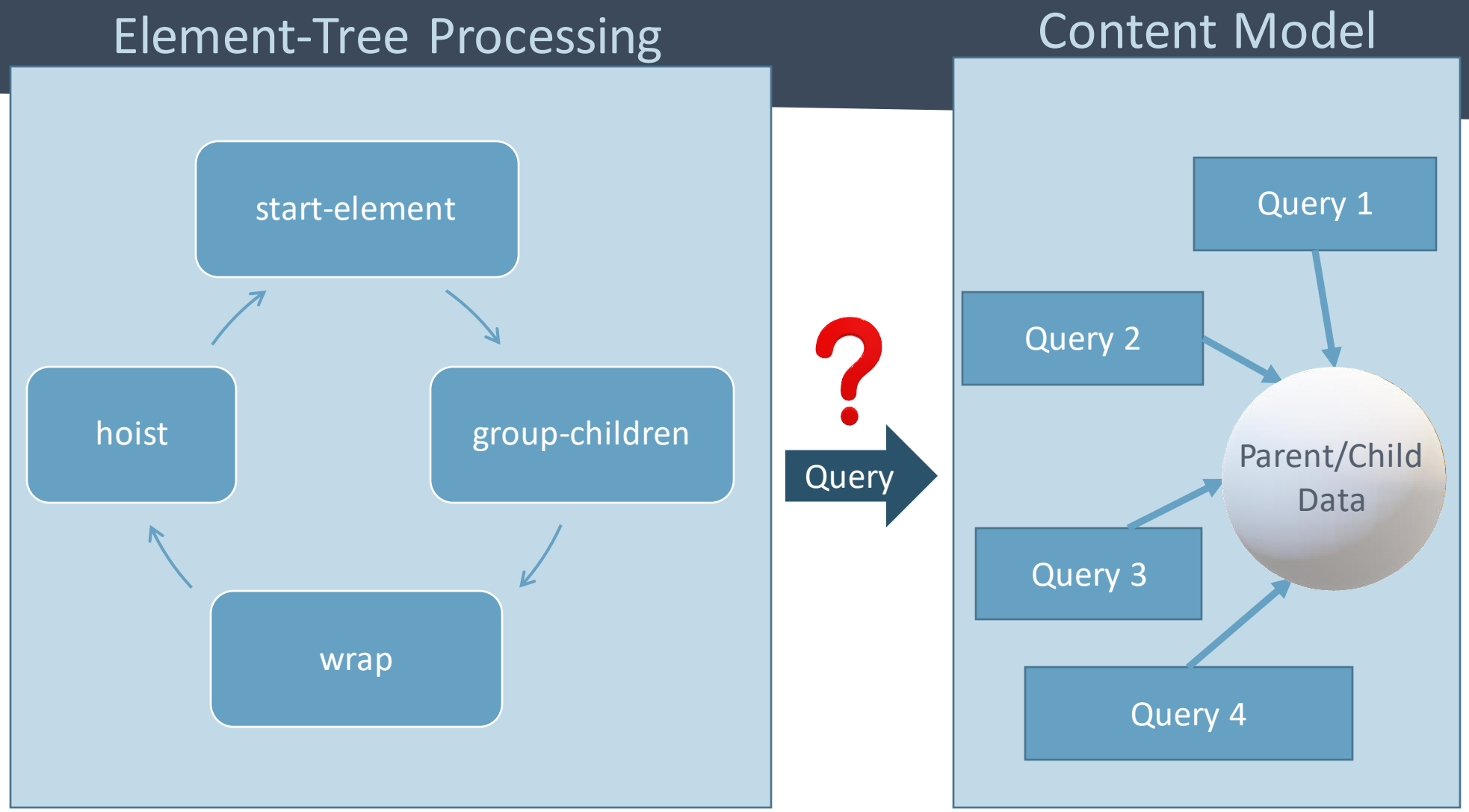
DELTAXML

Context: Positioning the HTML Table Normalizer in the XSLT Pipeline



DELTAXML

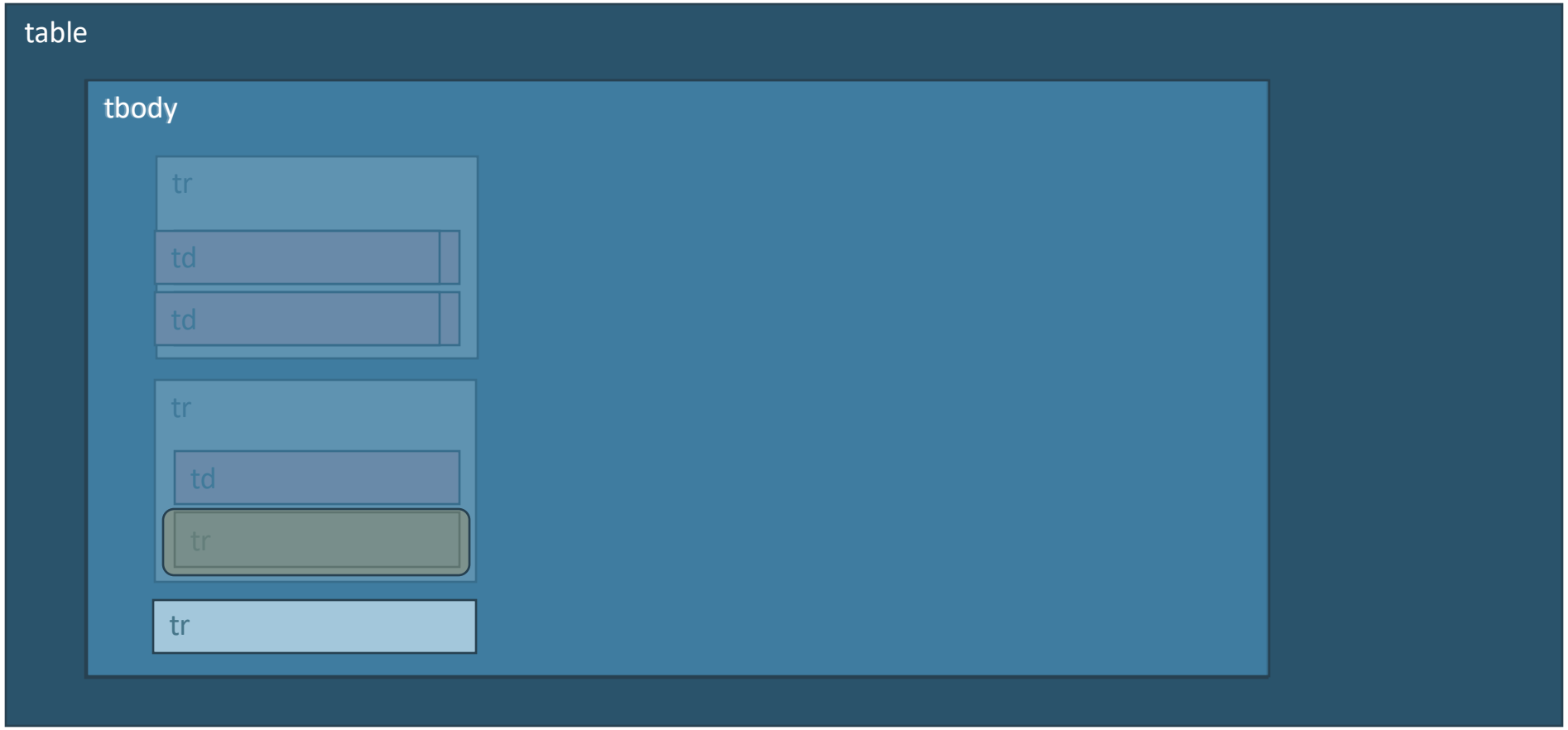
Normalizer: High-Level Design



DELTAXML

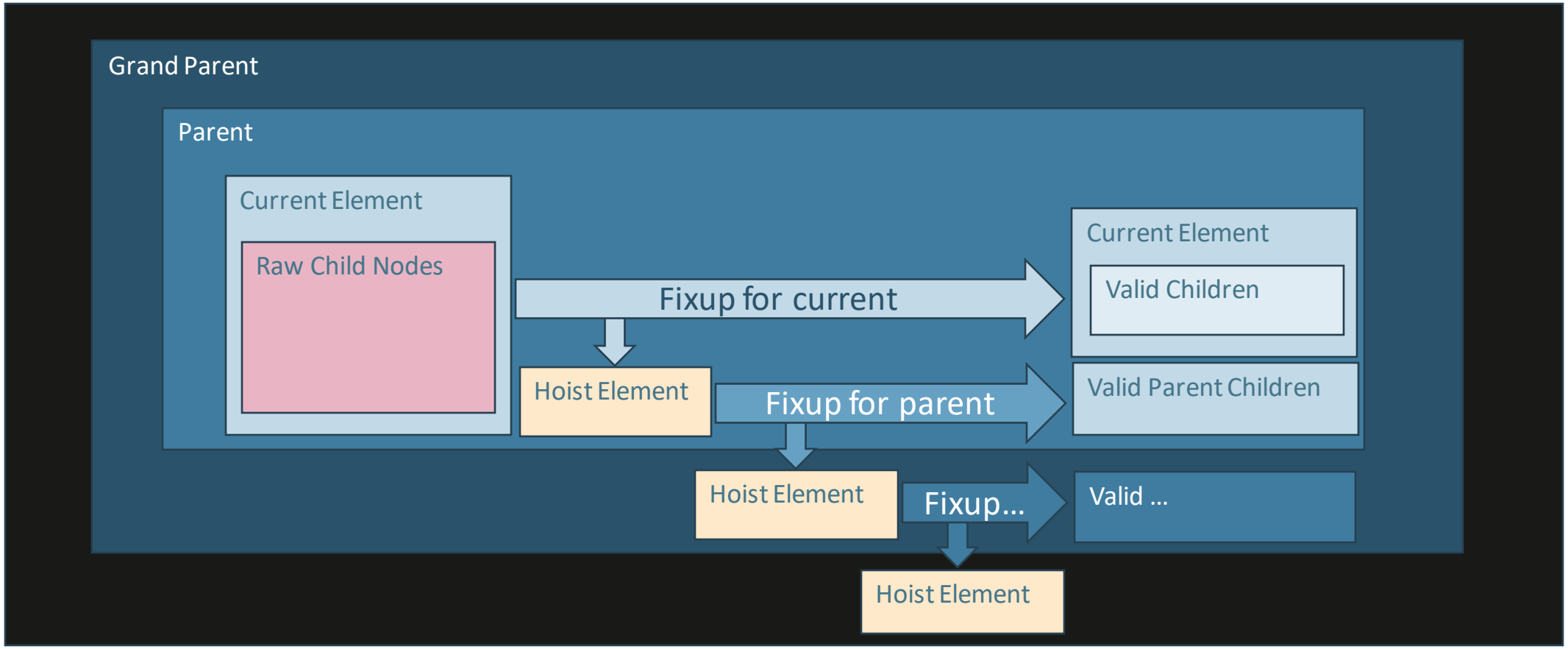
Element Wrapping

Content-Model Controlled Recursive Descent



‘Evading Capture’ (Hoisting)

Recursive Descent with Element Hoisting



DELTAXML

The xsl:template

‘Fix’ child nodes and insert as children and/or siblings of the current element

For each table element, assign wrapped child-nodes and hoist-position to variables


```
<xsl:template match="*[local-name() = $tableElementNames]" mode="inTable">
  <xsl:variable name="fixedChildNodes" as="node()*" select="fn:fixupChildNodes(local-name(), node())"/>
  <xsl:variable name="firstHoistPosition" as="xs:integer" select="fn:firstHoistPosition($fixedChildNodes)"/>

  <xsl:choose>
    <xsl:when test="$firstHoistPosition gt 0">
      <xsl:copy>
        <xsl:sequence select="@*, subsequence($fixedChildNodes, 1, $firstHoistPosition - 1)"/>
      </xsl:copy>
      <xsl:variable name="parentName" as="xs:string" select="fn:findParentName(local-name())"/>
      <xsl:variable name="hoistElementsForParent" as="node()*">
        <xsl:apply-templates select="subsequence($fixedChildNodes, $firstHoistPosition)" mode="stripHoistWrapper"/>
      </xsl:variable>
      <xsl:message use-when="$debug" select="fn:msg-on-hoist(., $fixedChildNodes, $firstHoistPosition, $parentName,
      <xsl:sequence select="fn:fixupChildNodes($parentName, $hoistElementsForParent)"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy>
        <xsl:sequence select="@*, $fixedChildNodes"/>
      </xsl:copy>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

DELTAXML

Insert wrapped nodes before hoist-position into context-element copy


```
<xsl:template match="*[local-name() = $tableElementNames]" mode="inTable">
  <xsl:variable name="fixedChildNodes" as="node()*" select="fn:fixupChildNodes(local-name(), node())"/>
  <xsl:variable name="firstHoistPosition" as="xs:integer" select="fn:firstHoistPosition($fixedChildNodes)"/>

  <xsl:choose>
    <xsl:when test="$firstHoistPosition gt 0">
      <xsl:copy>
        <xsl:sequence select="@*, subsequence($fixedChildNodes, 1, $firstHoistPosition - 1)"/>
      </xsl:copy>
      <xsl:variable name="parentName" as="xs:string" select="fn:findParentName(local-name())"/>
      <xsl:variable name="hoistElementsForParent" as="node()*">
        <xsl:apply-templates select="subsequence($fixedChildNodes, $firstHoistPosition)" mode="stripHoistWrapper"/>
      </xsl:variable>
      <xsl:message use-when="$debug" select="fn:msg-on-hoist(., $fixedChildNodes, $firstHoistPosition, $parentName,
      <xsl:sequence select="fn:fixupChildNodes($parentName, $hoistElementsForParent)"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy>
        <xsl:sequence select="@*, $fixedChildNodes"/>
      </xsl:copy>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

DELTAXML

Fix remaining nodes for the **context-parent** and append to result sequence


```
<xsl:template match="*[local-name() = $tableElementNames]" mode="inTable">
  <xsl:variable name="fixedChildNodes" as="node()*" select="fn:fixupChildNodes(local-name(), node())"/>
  <xsl:variable name="firstHoistPosition" as="xs:integer" select="fn:firstHoistPosition($fixedChildNodes)"/>

  <xsl:choose>
    <xsl:when test="$firstHoistPosition gt 0">
      <xsl:copy>
        <xsl:sequence select="@*, subsequence($fixedChildNodes, 1, $firstHoistPosition - 1)"/>
      </xsl:copy>
      <xsl:variable name="parentName" as="xs:string" select="fn:findParentName(local-name())"/>
      <xsl:variable name="hoistElementsForParent" as="node()*">
        <xsl:apply-templates select="subsequence($fixedChildNodes, $firstHoistPosition)" mode="stripHoistWrapper"/>
      </xsl:variable>
      <xsl:message use-when="$debug" select="fn:msg-on-hoist(., $fixedChildNodes, $firstHoistPosition, $parentName,
      <xsl:sequence select="fn:fixupChildNodes($parentName, $hoistElementsForParent)"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy>
        <xsl:sequence select="@*, $fixedChildNodes"/>
      </xsl:copy>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

DELTAXML

The Node Wrapping Function

Group nodes for wrapping and invoke <xsl:apply-templates>

[illegible] using <xsl:iterate> to track wrap-state

```
<xsl:function name="fn:fixupChildNodes" as="node()*">
  <xsl:param name="parentName" as="xs:string"/>
  <xsl:param name="childNodes" as="node()*"/>
  <xsl:iterate select="$childNodes">
    <xsl:param name="wrapperName" as="xs:string" select="''"/>
    <xsl:param name="wrapStart" as="xs:integer" select="0"/>

    <xsl:on-completion>
      <xsl:sequence select="fn:addWrapper($childNodes, $wrapperName, $wrapStart, count($childNodes) + 1)"/>
    </xsl:on-completion>

    <xsl:variable name="isWrapStarted" as="xs:boolean" select="$wrapStart gt 0"/>
    <xsl:variable name="nodeName" as="xs:string" select="local-name()"/>
    <xsl:variable name="validName" as="xs:string?" select="fn:findValidChild($nodeName, $parentName)"/>
    <xsl:variable name="state" as="map(xs:string, xs:boolean)"
      select="fn:calcWrapState(., $nodeName, $validName, $wrapperName, $isWrapStarted)"/>
    <xsl:if test="$state?hoist or $state?startWrap or $state?endWrap">
      <xsl:sequence select="fn:addWrapper($childNodes, $wrapperName, $wrapStart, position())"/>
    </xsl:if>
    <xsl:choose>
```

...

On ‘hoist’, enclose all remaining nodes in ‘hoist’ element and break-iteration


```
<xsl:choose>
  <xsl:when test="$state?hoist">
    <deltaxml:hoist-nodes>
      <xsl:sequence select="subsequence($childNodes, position())"/>
    </deltaxml:hoist-nodes>
    <xsl:break/>
  </xsl:when>
  <xsl:when test="$state?startWrap">
    <xsl:next-iteration>
      <xsl:with-param name="wrapperName" select="$validName"/>
      <xsl:with-param name="wrapStart" select="position()"/>
    </xsl:next-iteration>
  </xsl:when>
  <xsl:when test="$state?endWrap">
    <xsl:apply-templates select="." mode="inTable"/>
    <xsl:next-iteration>
      <xsl:with-param name="wrapperName" select="''"/>
      <xsl:with-param name="wrapStart" select="0"/>
    </xsl:next-iteration>
  </xsl:when>
```

...

DELTAXML

On ‘startWrap’ set the wrap-state: $wrapperName and $wrapStart


```
<xsl:choose>
  <xsl:when test="$state?hoist">
    <deltaxml:hoist-nodes>
      <xsl:sequence select="subsequence($childNodes, position())"/>
    </deltaxml:hoist-nodes>
    <xsl:break/>
  </xsl:when>
  <xsl:when test="$state?startWrap">
    <xsl:next-iteration>
      <xsl:with-param name="wrapperName" select="$validName"/>
      <xsl:with-param name="wrapStart" select="position()"/>
    </xsl:next-iteration>
  </xsl:when>
  <xsl:when test="$state?endWrap">
    <xsl:apply-templates select="." mode="inTable"/>
    <xsl:next-iteration>
      <xsl:with-param name="wrapperName" select="''"/>
      <xsl:with-param name="wrapStart" select="0"/>
    </xsl:next-iteration>
  </xsl:when>
```

...

On ‘endWrap’, apply templates to current node and reset the wrap-state


```
<xsl:choose>
  <xsl:when test="$state?hoist">
    <deltaxml:hoist-nodes>
      <xsl:sequence select="subsequence($childNodes, position())"/>
    </deltaxml:hoist-nodes>
    <xsl:break/>
  </xsl:when>
  <xsl:when test="$state?startWrap">
    <xsl:next-iteration>
      <xsl:with-param name="wrapperName" select="$validName"/>
      <xsl:with-param name="wrapStart" select="position()"/>
    </xsl:next-iteration>
  </xsl:when>
  <xsl:when test="$state?endWrap">
    <xsl:apply-templates select="." mode="inTable"/>
    <xsl:next-iteration>
      <xsl:with-param name="wrapperName" select="''"/>
      <xsl:with-param name="wrapStart" select="0"/>
    </xsl:next-iteration>
  </xsl:when>
```

...

DELTAXML

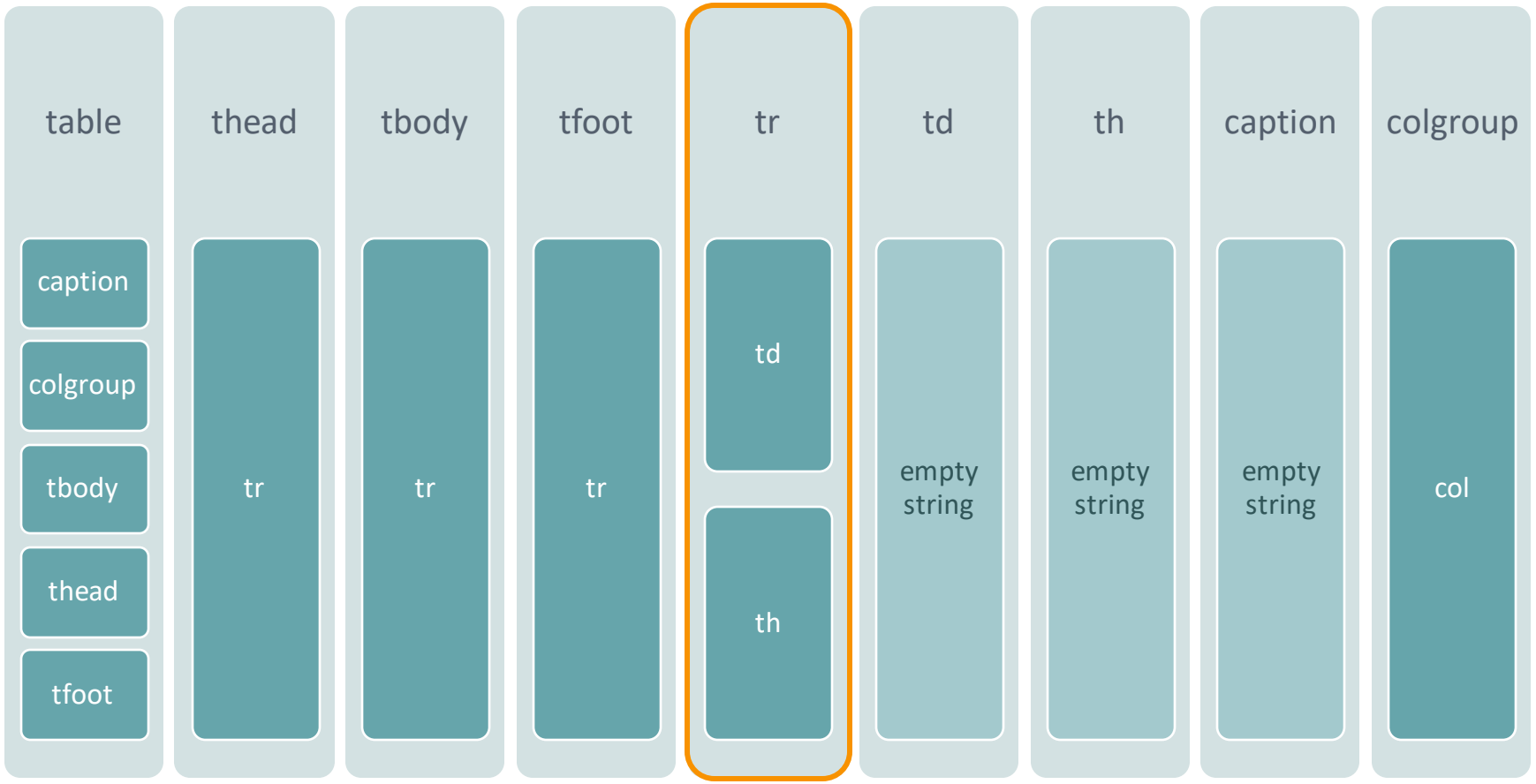
Otherwise (state has not changed), <xsl:apply-templates> to current-node, if not ‘buffering’


```
...
    <xsl:otherwise>
      <xsl:apply-templates select="if ($isWrapStarted) then () else ." mode="inTable"/>
    </xsl:otherwise>
  </xsl:choose>
```

The Content Model

The ‘Parent / Child ‘ Data Map

<xsl:variable name=“expectedChildren” as=“map (xs:string, xs:string+)”…

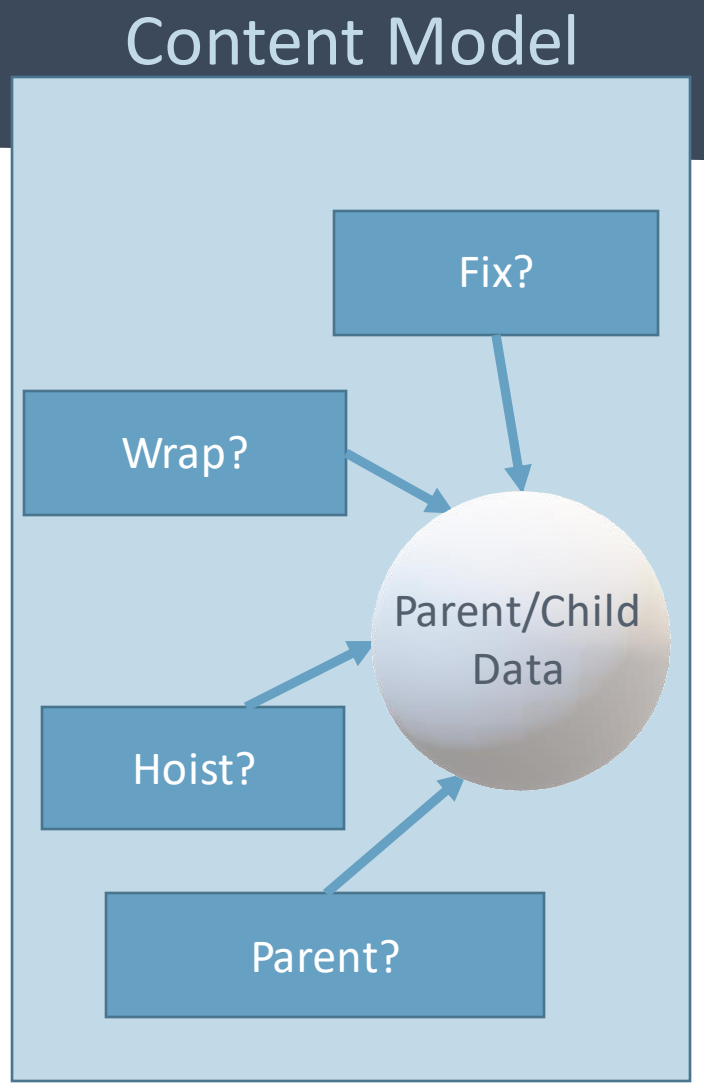


DELTAXML

Content Model Function Summary

For any XML element:

- **Fix?** Is it in the table-elements list?
- **Wrap?** What wrapper is expected, given the parent element?
- **Hoist?** Must this be hoisted, given the parent element?
- **Parent?** What is the valid parent element for this table-element?



DELTAXML

Tracing XSLT Execution

Using <xsl:message> with coloring and formatting

#1 Wrap all <table> children in <tbody>

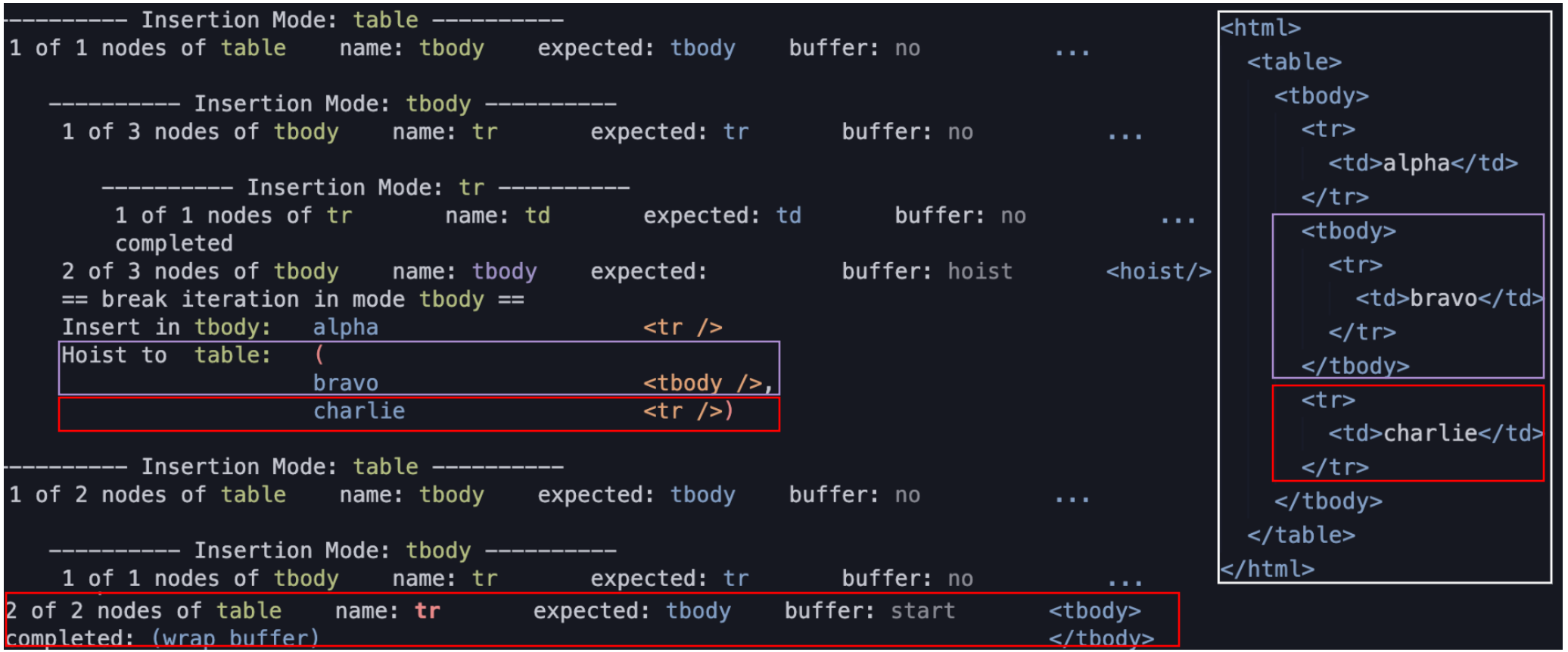

```
---------- Insertion Mode: table ----------
1 of 5 nodes of table    name: tr       expected: tbody    buffer: start       <tbody>
2 of 5 nodes of table    name: td       expected: tbody    buffer: continue    ...
3 of 5 nodes of table    name: td       expected: tbody    buffer: continue    ...
4 of 5 nodes of table    name: td       expected: tbody    buffer: continue    ...
5 of 5 nodes of table    name: tr       expected: tbody    buffer: continue    ...
completed: (wrap buffer)                                                     </tbody>

   ---------- Insertion Mode: tbody ----------
    1 of 5 nodes of tbody    name: tr       expected: tr       buffer: no          ...

       ---------- Insertion Mode: tr ----------
        1 of 3 nodes of tr       name: td       expected: td       buffer: no          ..
        2 of 3 nodes of tr       name: td       expected: td       buffer: no          ..
        3 of 3 nodes of tr       name: td       expected: td       buffer: no          ..
        completed
    2 of 5 nodes of tbody    name: td       expected: tr       buffer: start       <tr>
    3 of 5 nodes of tbody    name: td       expected: tr       buffer: continue    ...
    4 of 5 nodes of tbody    name: td       expected: tr       buffer: continue    ...
    5 of 5 nodes of tbody    name: tr       expected: tr       buffer: end         </tr>

       ---------- Insertion Mode: tr ----------
        1 of 3 nodes of tr       name: td       expected: td       buffer: no          ..
        2 of 3 nodes of tr       name: td       expected: td       buffer: no          ..
        3 of 3 nodes of tr       name: td       expected: td       buffer: no          ..
        completed

       ---------- Insertion Mode: tr ----------
        1 of 3 nodes of tr       name: td       expected: td       buffer: no          ...
        2 of 3 nodes of tr       name: td       expected: td       buffer: no          ...
        3 of 3 nodes of tr       name: td       expected: td       buffer: no          ...
```

Input XML

```
<html lang="en">
<title>test</title>
<table>
    <tr>
        <td>half</td>
        <td>quarter</td>
        <td>third</td>
    </tr>
    <td>one</td>
    <td>two</td>
    <td>three</td>
    <tr>
        <td>ten</td>
        <td>twenty</td>
        <td>thirty</td>
    </tr>
</table>
</html>
```

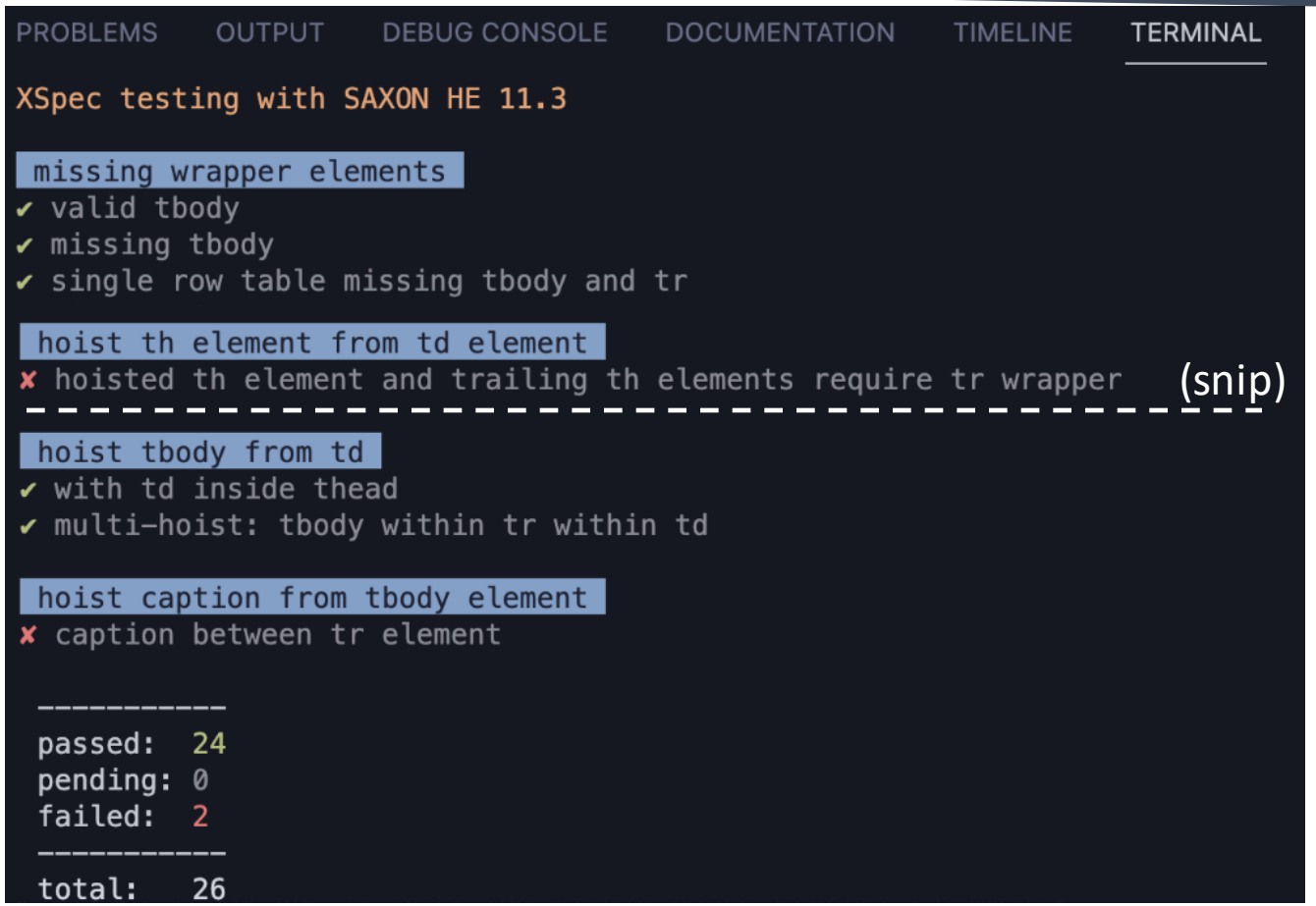
DELTAXML

Simple Element Hoisting Example



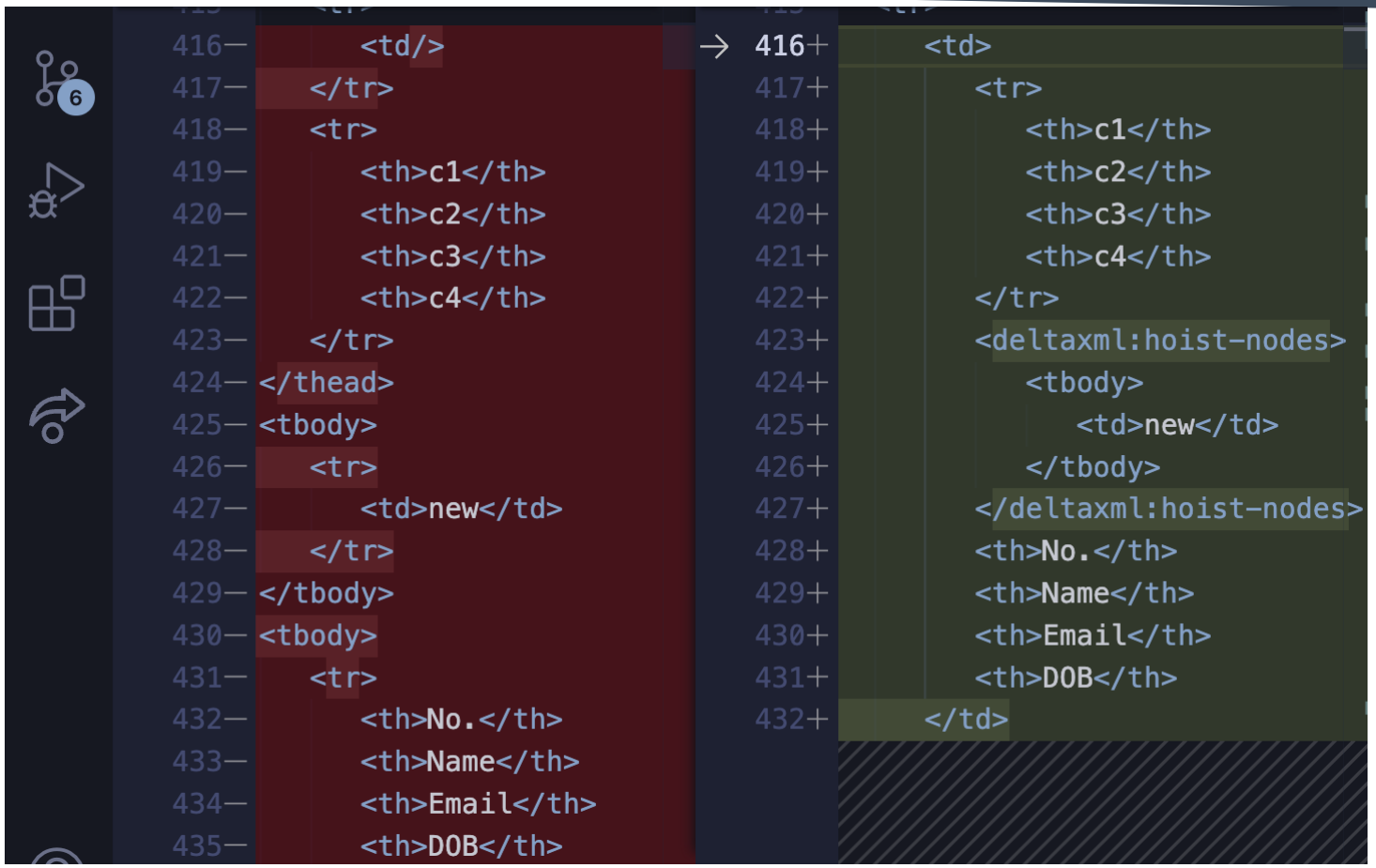
DELTAXML

Testing XSLT with XSpec (in VS Code)



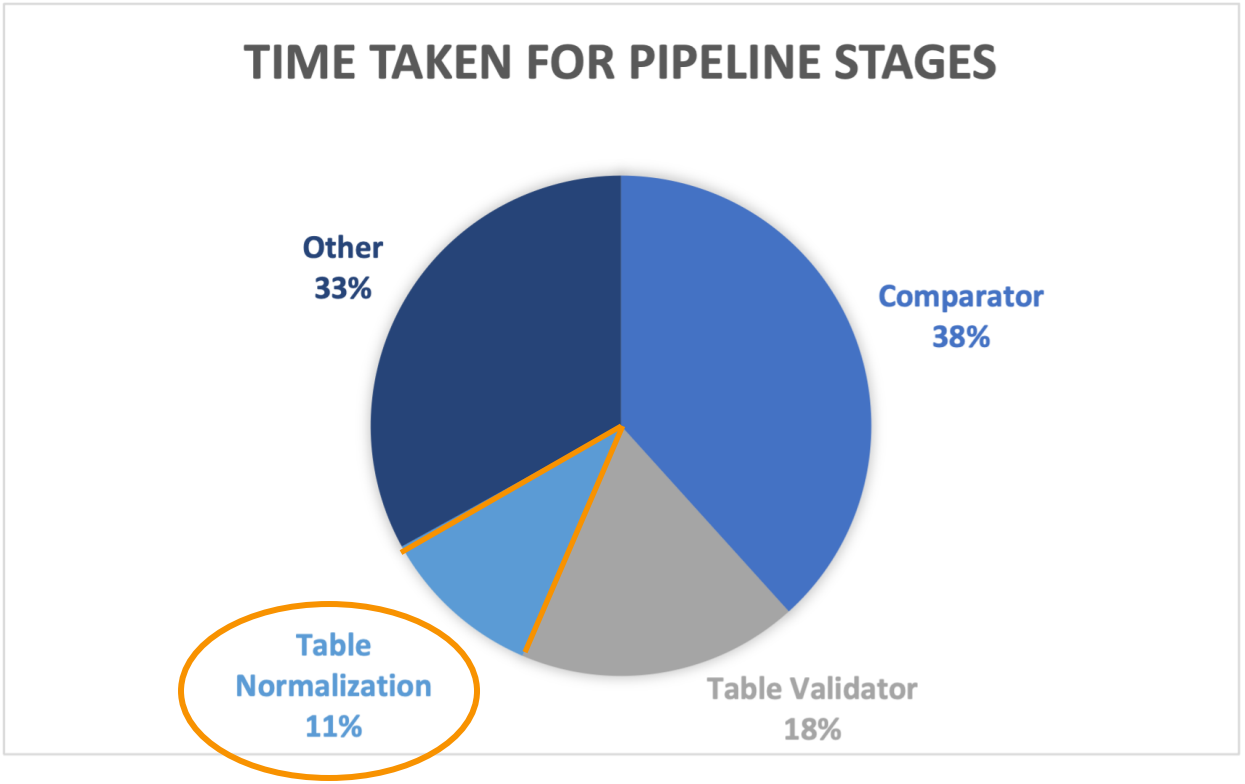
DELTAXML

Analysing Failed XSpec Tests



DELTAXML

Performance



DELTAXML

Summary

- Benefits of a Strict Content Model within a Pipeline
- Separation of Content Model from XML Structure Processing
- Recursive element ‘Wrapping’ during descent
- ‘Hoisting’ achieved by ‘Evading Capture’ during descent
- Tracing recursive execution with <xsl:message/>

<xsl:on-completion/>