IBM T. J. Watson Research Center

XML Screamer: Integrated, High-Performance XML Parsing, Validation and Deserialization

Margaret Kostoulas, Morris Matsa, Noah Mendelsohn, Eric Perkins, Abraham Heifets, Martha Mercaldi

24 May 2006

Outline

- **Introduction**
- **Why is XML Parsing Slow**
- **XML Screamer: Design**
- **XML Screamer: Performance**
- **Conclusion**

XML Performance

- **XML is everywhere**
- **Increasingly, XML is being used in processes that demand high-performance**
- **XML is widely seen as underperforming**
- **Validation is even worse**
- **Often, though, XML is used for exactly the kinds of things you want to validate**

Why are traditional XML parsers slow?

How fast is your computer?

- **An XML Parser must read through its input (and update the bytes of its output)**
- **A 1 GHz Pentium can read through an input buffer at approximately 100 Mbytes/sec which is approximately 10 cycles/byte.**

How fast are traditional XML processors?

- **Xerces-C 2.6, non-validating, using SAX: approximately 6 Mbytes/Sec/GHz, which is ~165 cycles/byte**
- **Expat version 1.95.8 for Windows using its UTF-8 API is about 12 Mbytes/sec/GHz (~80 cycles/byte)**
- **Remember: processor character scan rate = 100 MByte/sec/GHz (10 cycles/byte)**
- **What in the world are these XML processors doing for 80-160 cycles with each byte they pick up? That may be on the order of 300+ instructions per byte!**

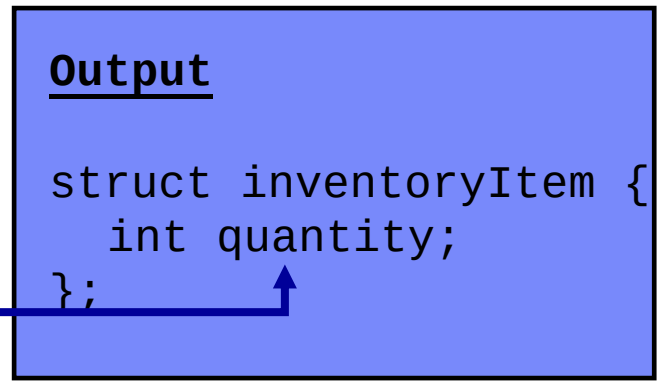
An Example

Input:

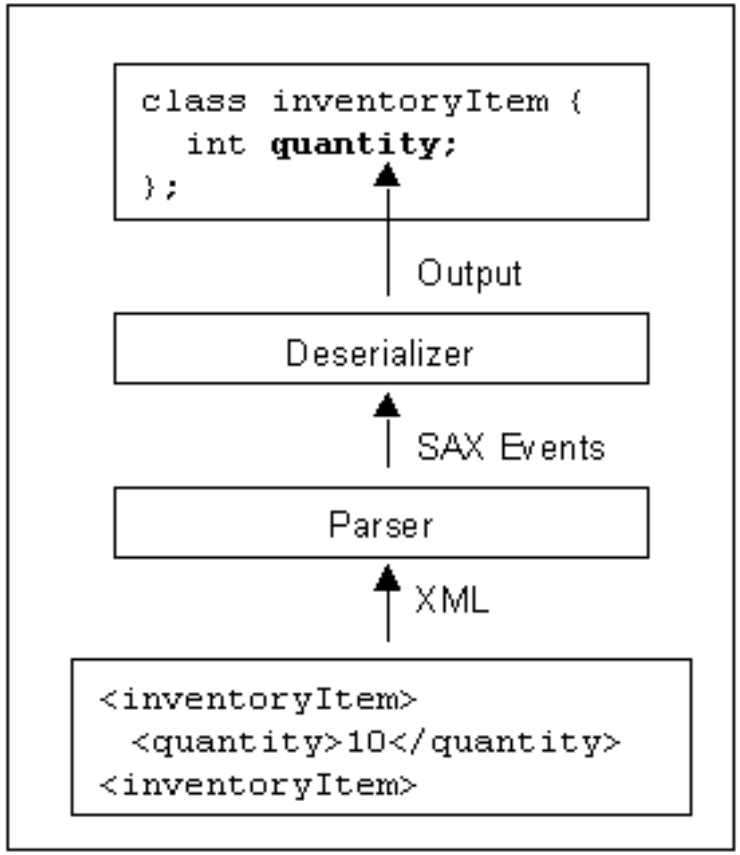
```
<inventoryItem>
   <quantity>10</quantity>
   ...
<inventoryItem>
```

Schema (abbeviated syntax):

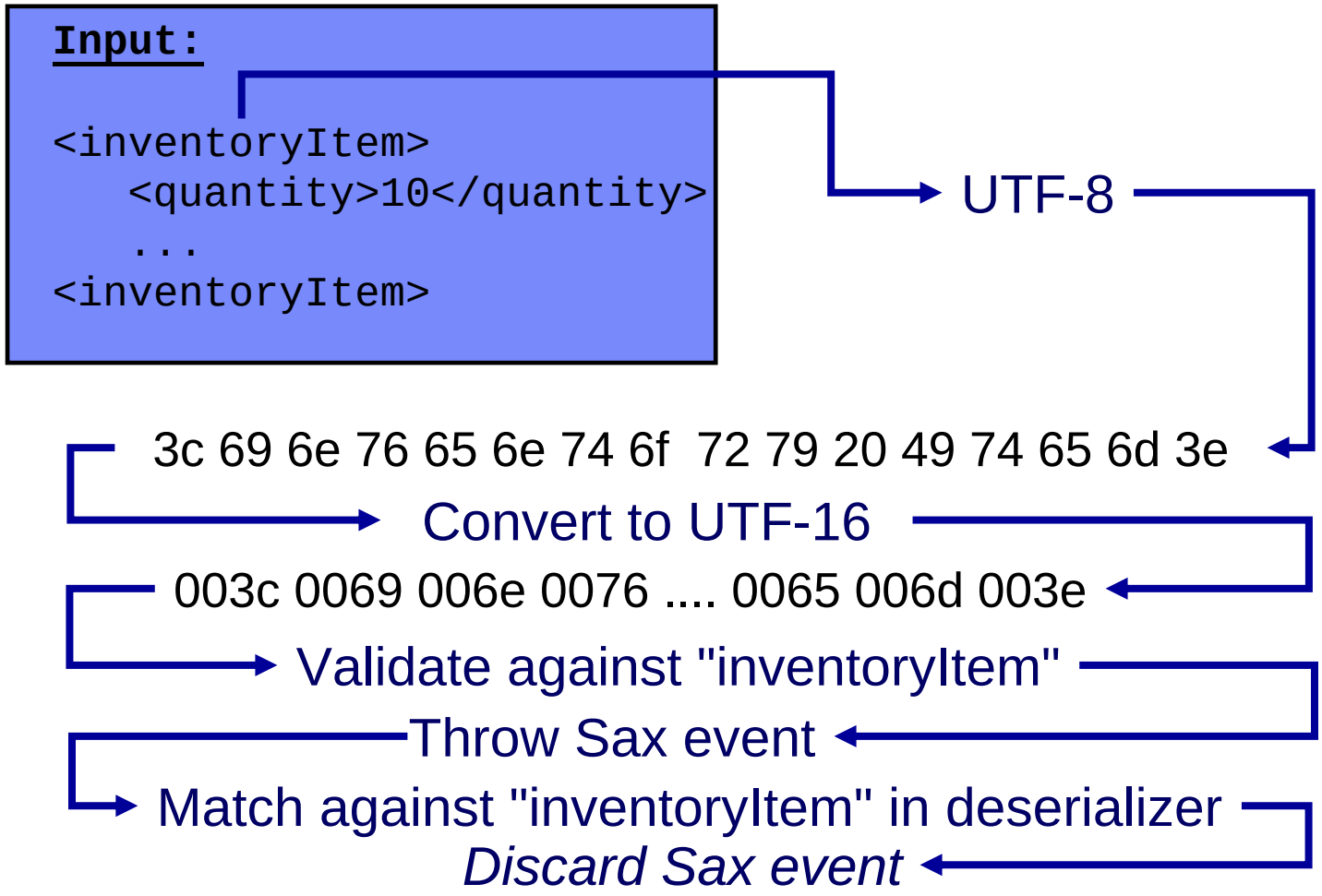
```
<element name="inventoryItem">
  <sequence>
    <element name="quantity">
      <simpleType base="xsd:integer"/>
         <maxInclusive="10000"/>
         <minInclussive="0"/>
      <simpleType>
    </element>
  </sequence>
</element>
```



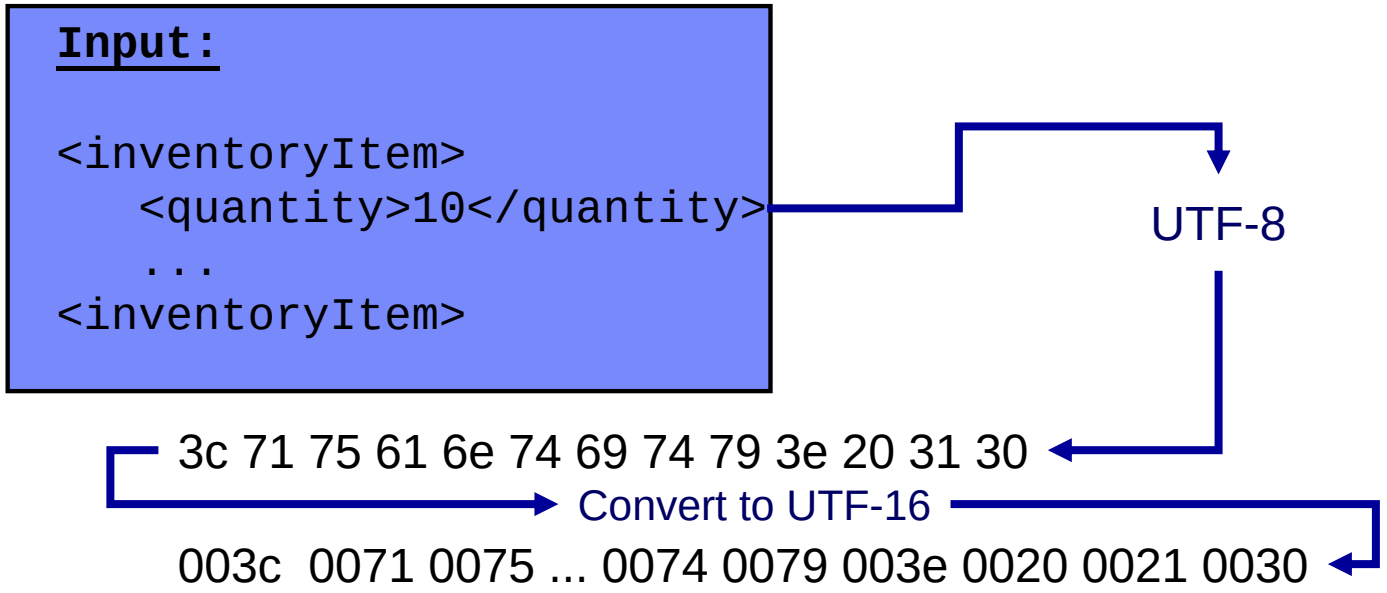
A traditional XML Parser/Deserializer



A traditional XML Parser/Deserializer



A traditional XML Parser/Deserializer



- Validate against “quantity”
- Throw Sax Event for “quantity”
- Match against “quantity” in deserializer
- Convert “10” to UTF-16 and integer
- Validate as 0 < quantity < 10000
- *Discard Integer*

- Sax event for UTF-16 “10”
- Convert “10” to integer (deserializer)
- *Discard Sax event*
- Copy integer to “quantity” field in structure

Traditional parsers: performance issues

- **Lots of expensive UTF-8 to UTF-16 conversions**
- **String compares done in UTF-16 (typically larger)**
- **Work duplicated between validator and deserializer**
- **Repeated data conversions (e.g. string/integer)**
- **Data copying**
- **Possible object & memory management overhead for SAX events**
- **Incremental reporting even when documents are small**

XML Screamer

An Integrated Approach to High Performance XML Parsing, Validation & Deserialization

The XML Screamer Project

- **Goal: show how fast XML and XML Schema Processing can be**
- **Approach: an XML Schema compiler that optimizes across layers that are traditionally separate -- scanning, parsing, validation & deserialization are integrated**

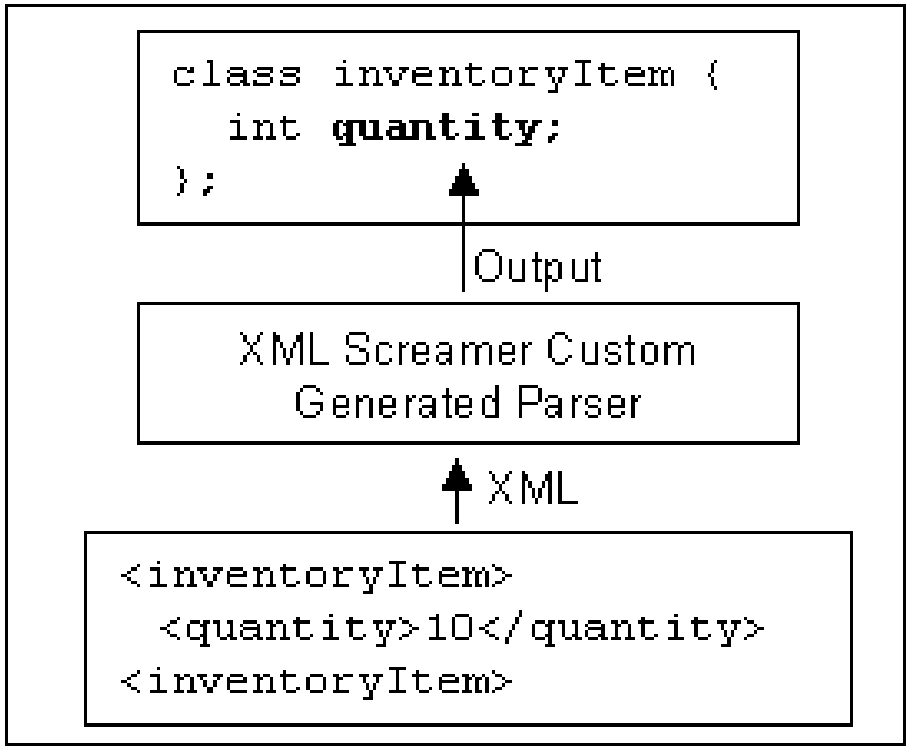
What XML Screamer is…

- **A compiler for XML Schemas**
- **Given a schema and a desired output API, generates a parser that:**
  - Parses and validates against the schema
  - Populates the desired API
- **Screamer compiler is in Java, produces parsers in C or Java**
- **C output much better tuned…much easier to study. All results reported here are for C.**

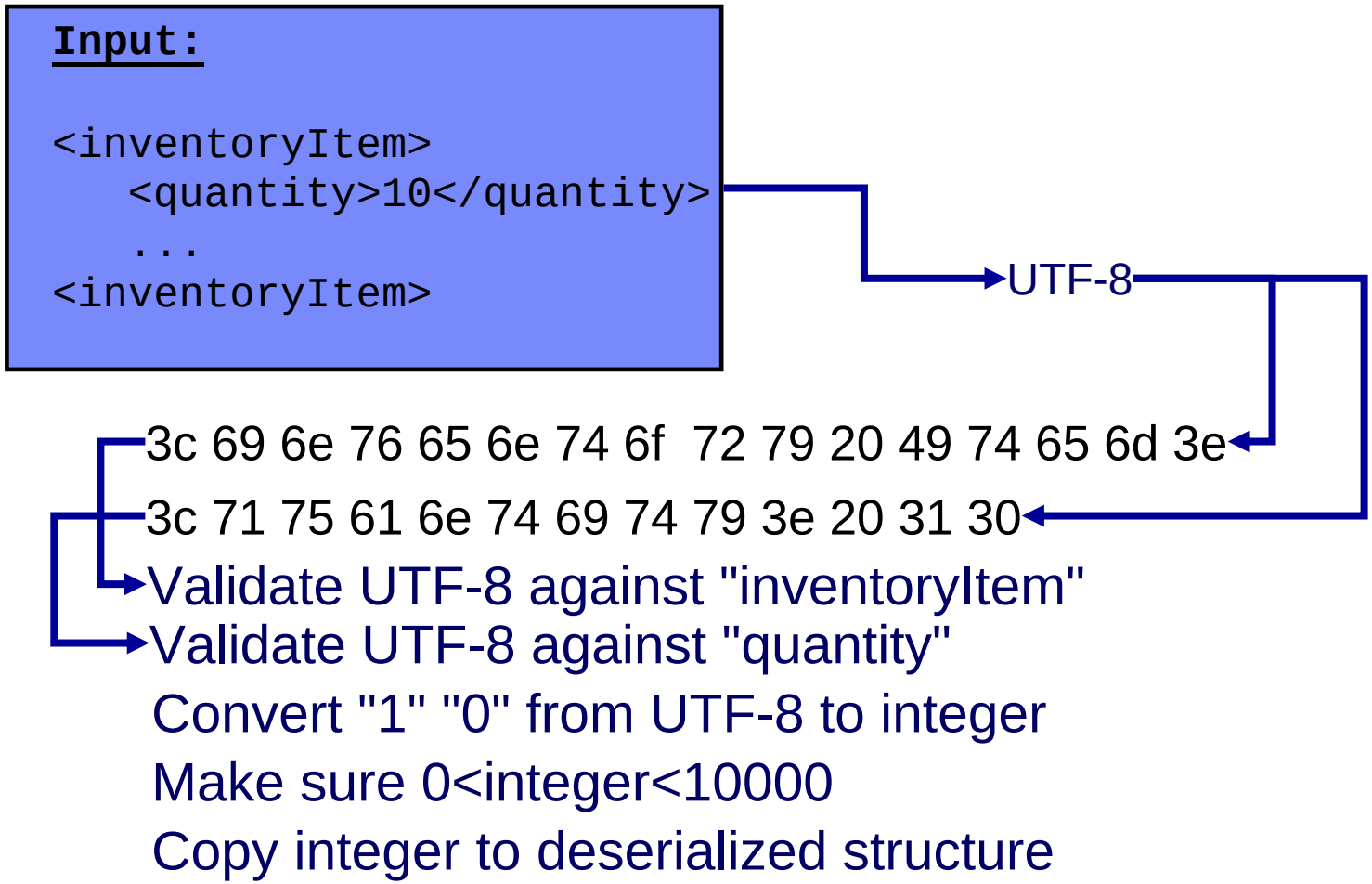
Screamer Parser APIs

- **NoAPI:**
  - just reports well-formedness and root validity
- **Business object**
  - like gSOAP, or JAX-RPC, SDO, etc.
- **SAX**
  - knowledge of schema allows pre-computation of some SAX output

An XML Screamer Parser/Deserializer



An XML Screamer Parser/Deserializer



What makes XML Screamer fast?

- **Optimizing across layers**
- **Avoid intermediate forms**
  - Don’t use SAX if you don’t need it
- **Avoid format conversions**
  - Work in input encoding wherever possible
- **Attention to detail**
- **In short: think like a compiler writer!**

How fast is XML Screamer?

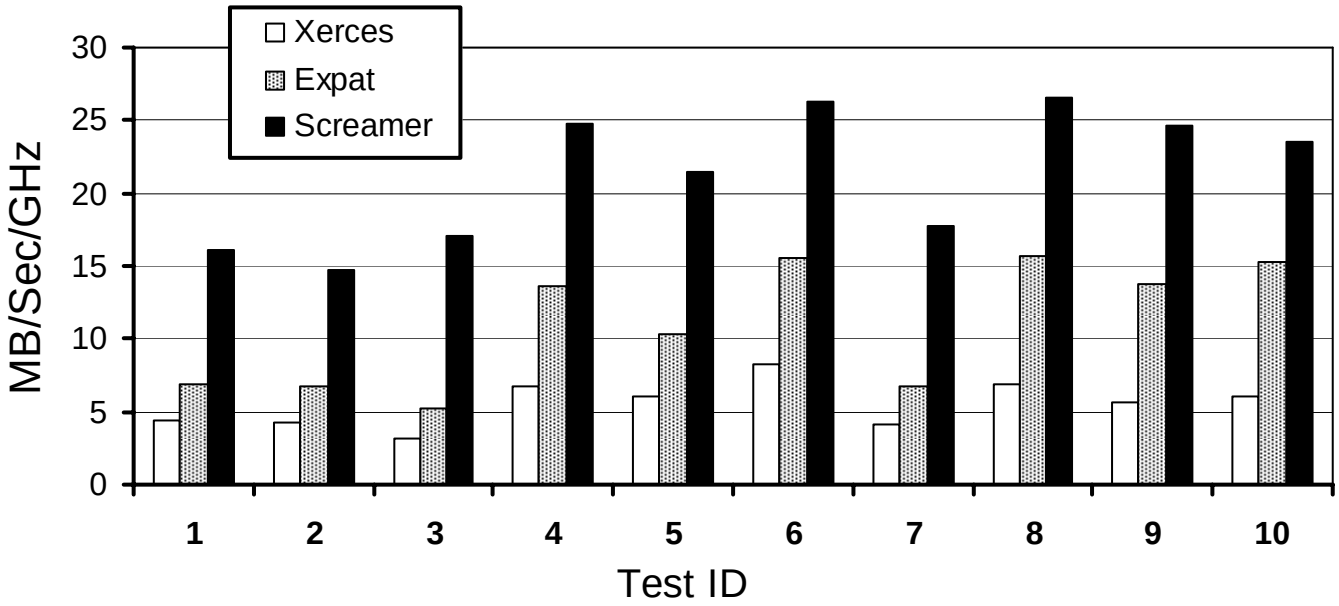
Benchmark Reporting

- **Test environment**
  - IBM eServer xSeries Model 235, 3.2 GHz Intel Xeon, 2 GB RAM, Microsoft Windows Server 2003 Service Pack 1. Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 13.10.3077.
  - Other machines checked for consistency – see paper
- **Results normalized to 1 GHz Pentium Processor**
  - Therefore: throughput of 3.2 Mbytes/sec would be reported as 1 MByte/sec/GHz
  - Scales well across Pentiums, Xeons, etc. of various clock speeds (Centrino is bit faster per cycle)

Test cases

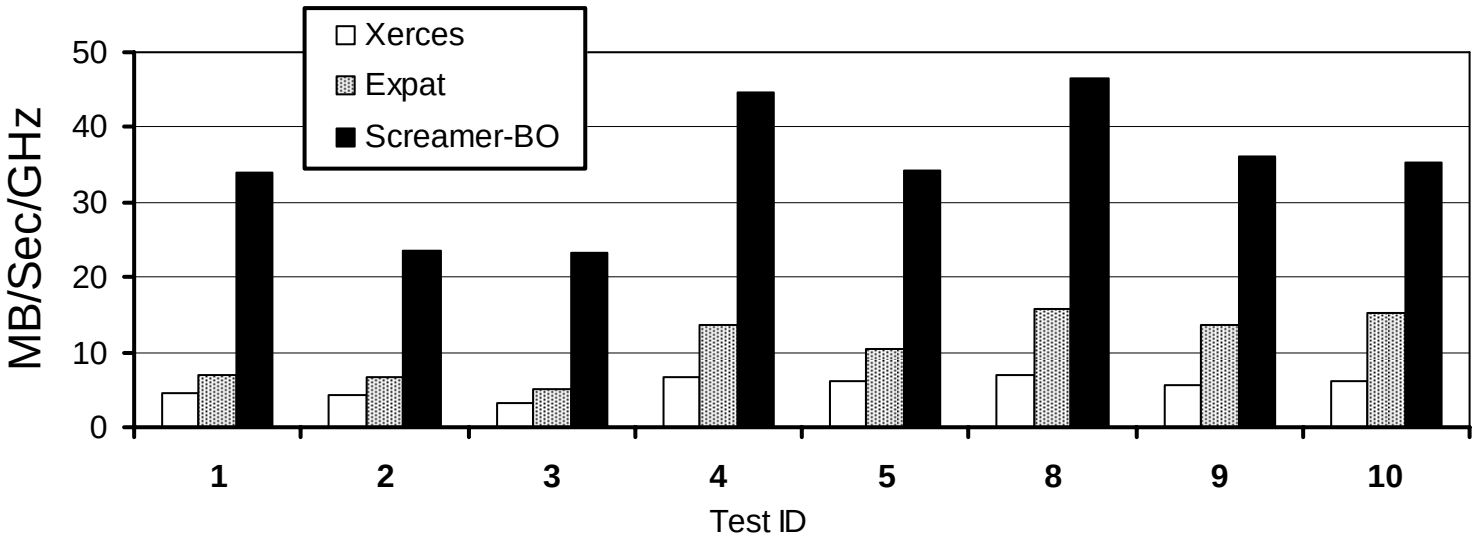
- **Report on 10 separate tests over 6 schemas**
- **All UTF-8 instances, single buffer, fits in memory.**
- **Sizes range from 990 bytes through 116.5KBytes.**

Comparison to Non-validating Parsers



- **Screamer is validating, and report SAX events**
- **Median Performance improvement: *1.9x Expat 3.8x Xerces***

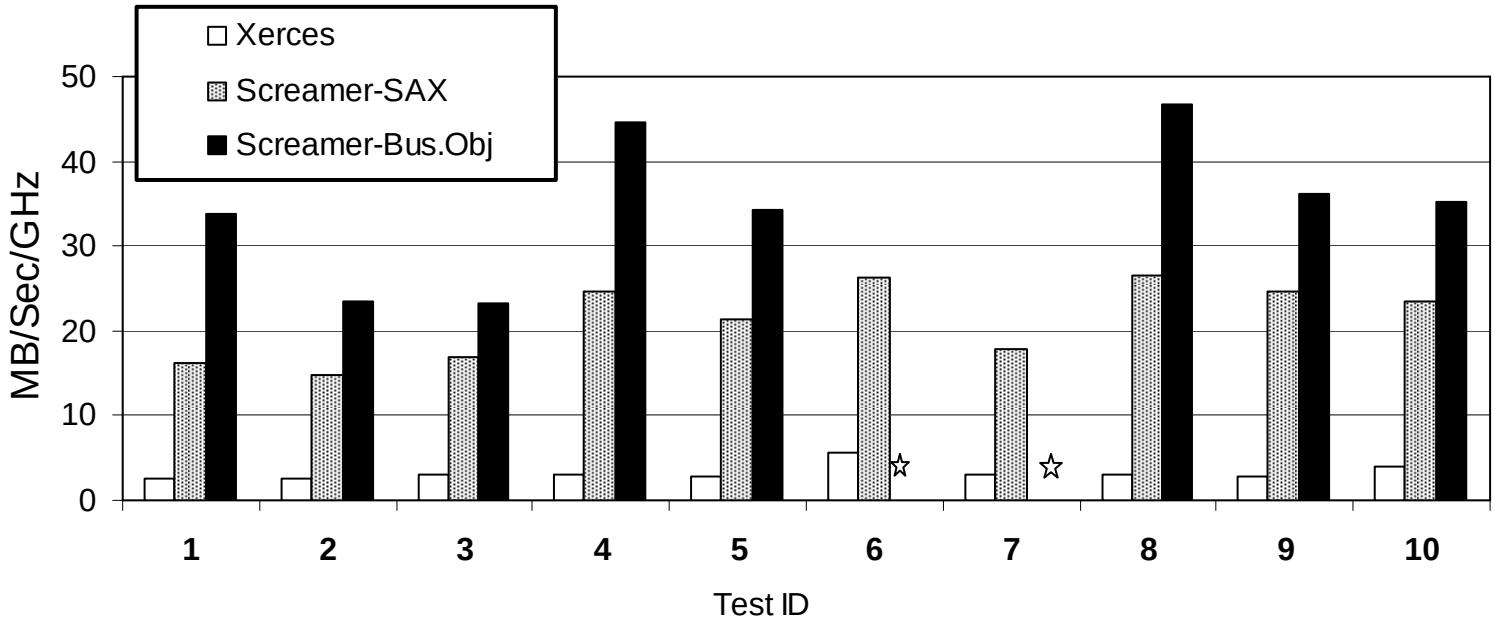
Business Object Creation



- **Xerces and Expat included for reference**
- **Median Performance improvement: *2.9x Expat 5.9x Xerces***

  **** Business Objects not supported for tests 6 & 7***

Validation Comparison



- **Median Performance improvement:**

  5.5x for Sax, 11.6x for Business Objects

  **** Business Objects not supported for tests 6 & 7***

XML Performance Summary

- **XML can be parsed, validated, and deserialized into high performance API at median throughput of about 35MBytes/sec. on a 1 GHz Pentium.**
  - That is 35% of the speed of raw character scan rate.
- **On modern 4GHz processor, that's 140MBytes/sec, or 14,000 10K Byte msgs/sec.**
  - If you want to devote only 10% of CPU to parsing, you can still do 14 Mbytes/sec, or 1,400 10Kbyte messages per second

Conclusions

- **Parsing, validation, and deserialization can run at speeds within 20-40% of raw character scan rate.**
  - Probably close to the true limits of XML performance.
- **Validation can mean a net *gain* in performance, if you have the option to compile in advance.**
- **XML Stack is *designed* in layers, but in *implementations*, layers disrupt performance.**
  - This has implications beyond just parsing and validation.
- **API Choice can make a significant difference in performance.**

Related publications

- Perkins, E., Kostoulas, M., Heifets, A., Matsa, M., Mendelsohn, N. *Performance Analysis of XML APIs,* XML 2005 Conference (http://www.idealliance.org/proceedings/xml05/abstracts/paper246.HTML)
- Perkins, E., Matsa, M., Kostoulas, M., Heifets, A., Mendelsohn, N. Generation of Efficient Parsers through Direct Compilation of XML Schema. IBM Systems Journal, 45, No. 2, (May 2006).

END

Backup 1: Performance Measurements

Test case			Throughput (Mytes/Sec/ProcessorGHz)							Comparisons			
			Xerces - SAX		Expat	XML Screamer							
						Validating			any Type				
ID	Schema Filename	Size	Non val	Val	Non val	NOAPI	Business Object API	SAX	SAX	Screamer SAX vs Expat	Screamer Business Object vs. Expat	Screamer SAX vs. Xerces Val SAX	Screamer: Schema vs anyType Sax
1	po	990.00	4.41	2.65	6.85	35.08	33.88	16.12	12.75	**2.4x**	**4.9x**	**6.1x**	**1.3x**
2	ipo	1,406.00	4.24	2.51	6.81	23.23	23.56	14.76	14.25	**2.2x**	**3.5x**	**5.9x**	**1.0x**
3	MI_AUS_RESPONSE2_1	1,572.00	3.21	2.98	5.21	25.95	23.21	17.00	7.51	**3.3x**	**4.5x**	**5.7x**	**2.3x**
4	po	8,062.00	6.79	3.01	13.68	48.00	44.67	24.73	16.08	**1.8x**	**3.3x**	**8.2x**	**1.5x**
5	ipo	8,077.00	6.08	2.90	10.31	38.40	34.21	21.44	16.46	**2.1x**	**3.3x**	**7.4x**	**1.3x**
6	bibteXML	8,609.00	8.28	5.58	15.54	47.49	NA	26.25	19.77	**1.7x**	**NA**	**4.7x**	**1.3x**
7	MI_AUS_REQUEST2_1	9,429.00	4.06	3.16	6.74	23.15	NA	17.79	8.52	**2.6x**	**NA**	**5.6x**	**2.1x**
8	po	63,754.00	6.88	3.02	15.65	49.87	46.63	26.58	16.37	**1.7x**	**3.0x**	**8.8x**	**1.6x**
9	ipo	64,233.00	5.68	2.85	13.75	44.00	36.15	24.65	16.67	**1.8x**	**2.6x**	**8.6x**	**1.5x**
10	periodic_table	116,506.00	6.03	3.99	15.25	34.61	35.28	23.47	14.16	**1.5x**	**2.3x**	**5.9x**	**1.7x**

Backup 2: SAX Pre-computation