XML Compression

XML Compression

- XML files are too big!!!
  - Compression
- Popular tools
  - Pkzip
  - RAR
  - Winzip
- Knowledge Based Compression
  - Uses knowledge about message structure to direct compression/decompression
  - Can be combined with other techniques

XML Compression

compressors

- **XML-conscious**
  - **queryable**
    - **sequential access**
      - •BPLEX
      - •TREECHOP
      - •XGRIND
      - •XPRESS
    - **random access**
      - •XQueC
      - •XSeq
  - **non-queryable**
    - •AXECHOP
    - •DTDPPM
    - •SCMPPM
    - •XComp
    - •XMill
    - •XMLPPM
    - •XWRT
- **generic text**
  - •bzip2
  - •gzip
  - •PPM variants
  - •Etc.

Other Classification Criteria

- Schema-aware or schema-oblivious?
  - Information in schema documents can allow document structure to be encoded more succinctly
  - Some schema languages (e.g., XML Schema) specify data types – this knowledge can be used to guide selection of compression schemes
  - Limited applicability: not all documents have an associated schema document
- Online vs. offline operation
  - Can decompression be carried out incrementally?
- Compression paradigm used
  - Homomorphic or permutation-based?

Schema-aware Compression: Example

```
<!ELEMENT course (name,instructor,students)>
```

Indicates that each <course> element must have <name>, <instructor>, and <students> elements as children (in that order): no unpredictability

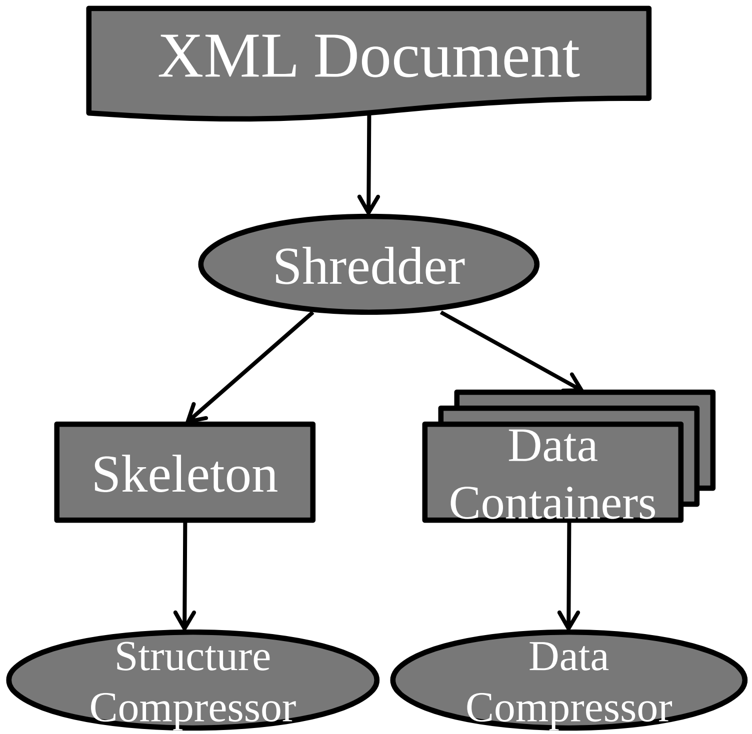
If encoder and decoder both possess the DTD: 0 bits needed to represent structure of <course> elements!

```
<!ELEMENT student (a1,a2,midterm,(finalexam|project))>
```

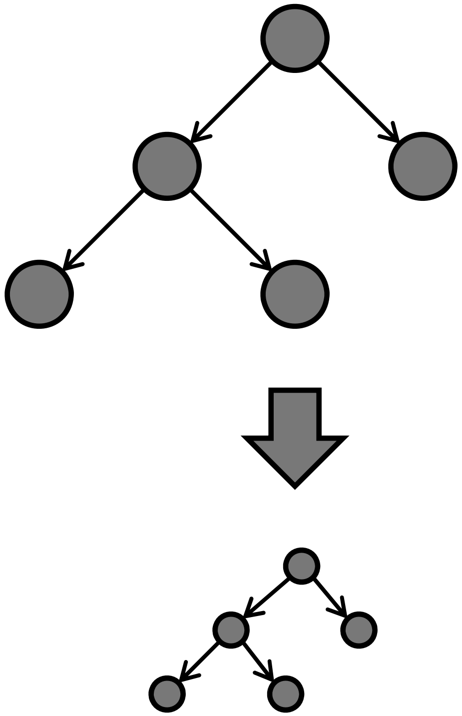
Similarly, only 1 bit is needed to indicate whether <finalexam> or <project> appears as the fourth child of <student>

Permutation-based Approaches

- Document is rearranged to localize repetitions before passing to back-end compressor(s)
  - Data segments are grouped into different containers, typically based on the identity of parent element
  - Tag structure ("skeleton") and data segments are compressed separately



Homomorphic Approaches



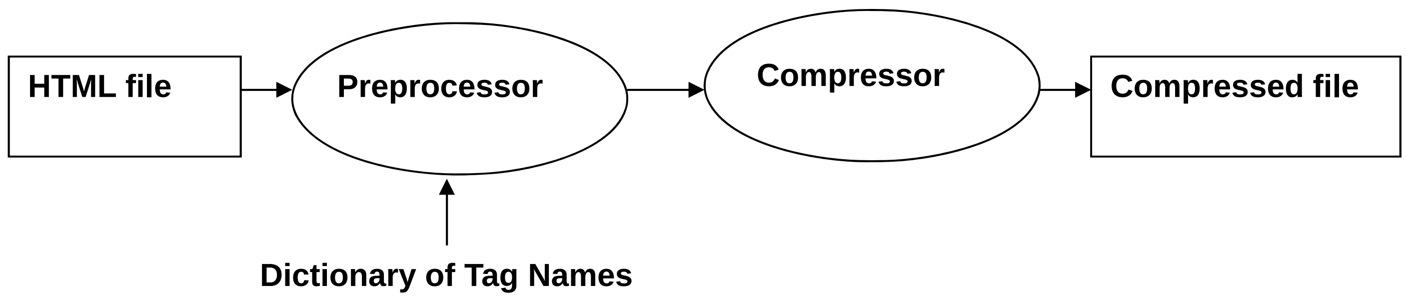
- Each XML token is compressed individually, “in-place”
- Compression process maintains structure of original document
- Poorer compression, but easier to query than permutation-based approaches (less fragmentation)

HTML Basic

- A HTML file consists of many tag commands, which are used for controlling how the document is displayed in the web browser.
- A begin tag starts with the symbol ‘<’and ends with the symbol‘>’, while the end tag starts with ‘<’ and ends with ‘/>’. Only the text embraced by the pair of tags will be affected by the function of the tag.
- For example, ****Test**** = **Test.**

Compression Model for HTML

- A tag name dictionary is used in the preprocessor for matching the HTML begin tags.
- The output of the preprocessor will be an intermediate text file with indices of tag names and non-tag characters.
- The intermediate file will be further compressed by a double-byte compression algorithm.



Four Cases Handled by Preprocessor - I

Case One: Begin and End Tags Are in Pairs

Assume tag dictionary contains “body” in index position 1, “br” in index position 2, “font” in index position 3, “html”in index position 4 and “I” in index position 5.

<html> <body> This is an <I> example </I> </body> </html>

When encoding, the preprocessor will replace the begin tag names by indices and all end tags by a special ***endtag*** symbol, the output string will become:

4 1 This is an 5 example endtag endtag endtag

Four Cases Handled by Preprocessor - 2

Case Two: With Begin Tags Only or Begin and End Tags Are Not in Pairs

Assume tag dictionary contains “body” in index position 1, “br” in index position 2, “font” in index position 3, “html”in index position 4 and “I” in index position 5.

<html> <body> This is
 an <I> example </I> </body> </html>

When a begin tag appears in single, a ***skiptag*** symbol will be generated and put in an appropriate position of the intermediate file, the output string will become:

4 1 This is 2 an 5 example endtag skiptag endtag endtag

Four Cases Handled by Preprocessor - 3

Case Three: The Tags Have Attribute Fields

Assume tag dictionary contains “body” in index position 1, “br” in index position 2, “font” in index position 3, “html”in index position 4 and “I” in index position 5.

<html> <body> Hello </body> </html>

If a tag with attribute fields is read, two ***attributetag*** symbols will be generated to embrace the attribute fields, the output string will become:

4 1 3 attributetag color=FFFFFF size =+1 attributetag Hello endtag endtag endtag

Four Cases Handled by Preprocessor - 4

Case Four: The Tags Are Not Found in the Tag Dictionary

Assume tag dictionary contains “body” in index position 1, “br” in index position 2, “font” in index position 3, “html”in index position 4 and “I” in index position 5.

<html> <body> <blink> Hello</blink>AA<blink>World</blink></body></html>

If a tag name is not found in the dictionary, two ***escapetag*** symbols will embrace the unmatched tag name. The new tag name will be appended into dictionary. The output string will become:

4 1 escapetag blink escapetag Hello endtag AA 6 World endtag endtag endtag

XMill

- Specialized compressor for XML data
- Makes XML look “small”
- Three principles
  - Compress the structure separately from the data
  - Group the data values according to their types
  - Apply semantic (specialized) compressors
- XMILL: An Efficient Compressor for XML Data by Liefke and Suciu, in SIGMOD'2001

An Example

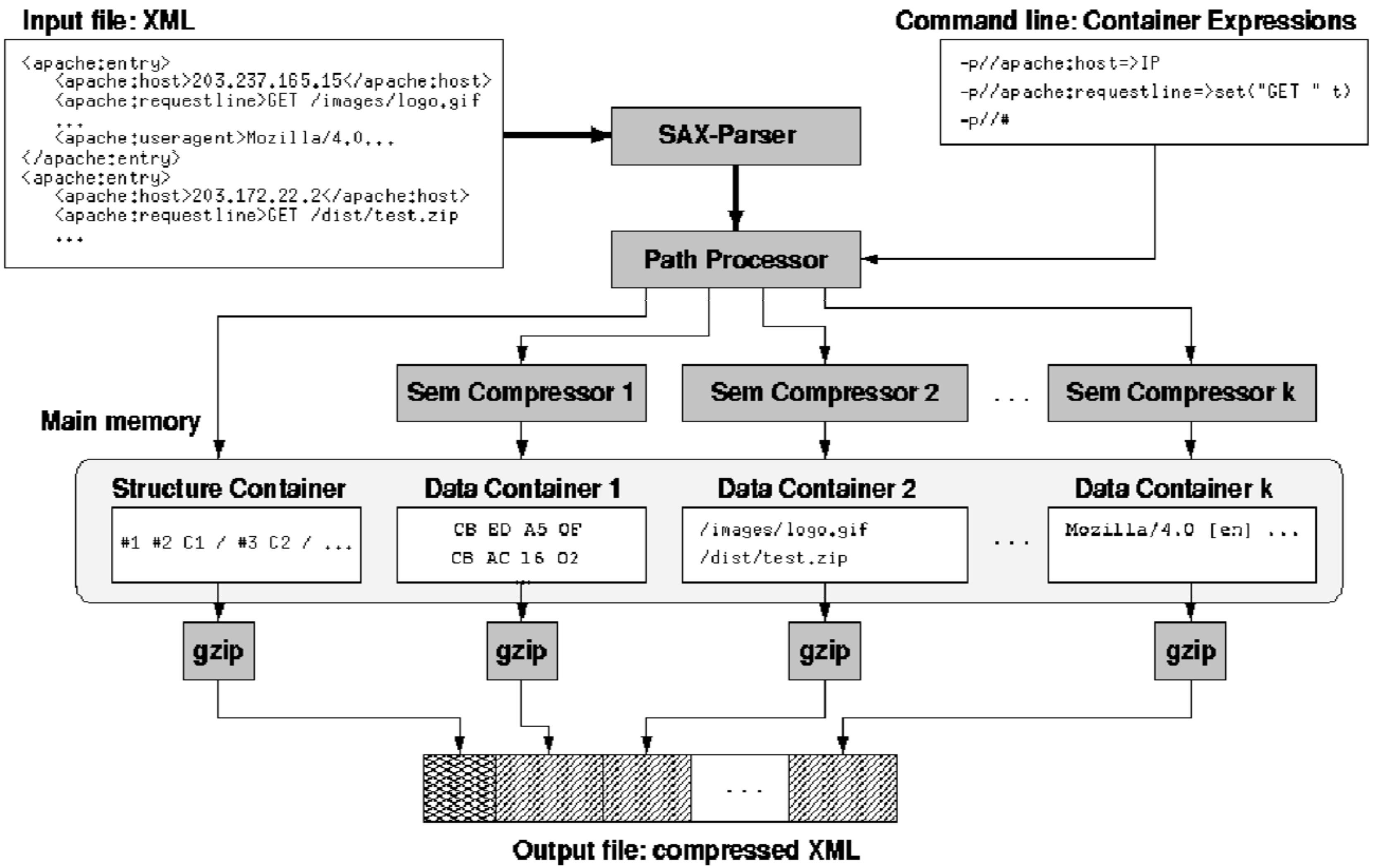
ASCII File 15.9 MB   (gzipped 1.6MB):

XML-ized inflates to 24.2 MB   (gzipped 2.1MB):

Web Server Logs

```
<apache:entry>
   <apache:host> 202.239.238.16 </apache:host>
   <apache:requestLine> GET / HTTP/1.0 </apache:requestLine>
   <apache:contentType> text/html </apache:contentType>
   <apache:statusCode> 200</apache:statusCode>
   <apache:date> 1997/10/01-00:00:02</apache:date>
   <apache:byteCount> 4478</apache:byteCount>
   <apache:referer> http://www.net.jp/ </apache:referer>
   <apache:userAgent> Mozilla/3.1$[$ja$]$(I)</apache:userAgent>
</apache:entry>
```

The Architecture of XMill



Separating Structure

- Example
  - XML Document

    <Book> <Title lang="English"> Views </Title>

    <Author> Miller </Author>

    <Author> Tai </Author>

    </Book>
  - Tags

    Book = #1, Title = #2, @lang = #3, Author = #4
  - Data values

    English = C1, Views = C2, Miller, Tai = C3
  - Converted Structure

    Structure = #1 #2 #3 C1 / C2 / #4 C3 / #4 C3 / /

Separating Structure

- Example
  - XML Document

<Employees>
<Employee id="1">Homer Simpson</Employee>
<Employee id="2">Frank Grimes</Employee>
</Employees>

Dictionary
T1 =>Employees
T2 => Employee
T3 => @id

Structure Container
T1 T2 T3 C3 / C4 / T2 T3 C3 / C4 / /

C3
1
2

C4
Homer Simpson
Frank Grimes

Grouping Data Values

- Data Container
  - Each data value is uniquely assigned to one data container
  - Mapping rules
    - the data value's path
    - the user-specified container expression
- Container Expression
  - e ::= label | * | # | e1/e2 | e1//e2 | (e1|e2) | (e)+
  - All are XPath constructs except (e)+ and #
  - # : any tag or attribute(much like *),
    but each match of # will determine a new container

Grouping Data Values

- Container Expression
  - Expression example
    - //Name
    - //Person/Title
    - //#
    - //Person/#

Semantic Compressors

- Atomic Compressor

Compressor	Description	Compressor	Description
t	Default text compressor	**u**	For positive int
i	Compressor for int	**u8**	For positive int < 256
di	Delta compressor for int	**rl**	Run-length encoder
e	Enumeration encoder	**"…"**	Constant compressor

  – Extended container expression
    - C ::= c | c => s
    - c : container expression
    - s : sementic compressor

Semantic Compressors

- Combined Compressors
  - *Sequence Compressor* **`seq(s1 s2 …)`**
    - IP Addr. : 104.44.29.21
    - seq(u8 “.” u8 “.” u8 “.” u8)
  - *Alternate Compressor* **`or(s1 s2 …)`**
    - page reference : 145-199, 145
    - or(seq(u “-” u) u)
  - *Repetition Compressor* **`rep(d s)`**
    - d : delimiter, s: anonther semantic compressor
    - a seq. of comma separate keywords
    - rep(“,” e)

Semantic Compressors

- User-defined Compressors
  - Highly specialized compressor
    - DNA sequence
  - Procedure
    - write their own compressor/decompressors
    - link them into Xmill and Xdemill

Software

- Publicly available
  - Window
  - Linux
- Tested with 6 different sets of data
- Download
  - www.research.att.com/sw/tools/xmill/

XMill Example

```
<customers>
  <customer id="c1">
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <invoice total="300">
      <items>
          <item>item1</item>
          <item>item2</item>
      </items>
    </invoice>
  </customer>

<customer id="c2">
    <firstName>Bill</firstName>
    <lastName>Luis</lastName>
</customer>
</customers>
```

XML file

Elements table

1	/customers
2	/customers/customer
3	/customers/customer/firstName
4	/cusomters/customer/lastName
5	/cusomters/customer/invoice
6	/cusomters/customer/invoice/items
7	/cusomters/customer/invoice/items/item

Attributes table

100	/cusomters/customer/@id
101	/cusomters/customer/invoice/@total

/customers/customer/firstName

John
Bill

/customers/customer/lastName

Smith
Luis

/customers/customer/invoice/items/item

item1
item2

/customers/customer/@id

C1
C2

/customers/customer/invoice/@total

300

XGRIND

- XMill
  - The compressed XML is not *queryable*.
- XGrind : A Query-friendly XML Compressor
  - The resulting compressed XML document maintains the original structure. Compressed XML is also
    - XML
    - Queryable
    - DTD validatable
- P. M. Tolani, J. R. Haritsa, XGRIND: A Query-friendly XML Compressor, In Proc. of SIGMOD, 2002

Two Phases

- Phase 1:
  - Scan XML document
  - Compute statistics for each element and attribute for compression in Phase 2
- Phase 2:
  - Scan XML document
  - Make compressed result XML document
    1. Meta-Data Compression (start tag, end tag, attribute)
    2. Enumerated-type attribute value compression
    3. General element/attribute value compression

XGRIND

- Meta-Data Encoding
  - Each start-tag of an element is encoded by a ‘T’ followed by a uniquely assigned element-ID.
  - All end-tags are encoded by ‘/’s.
  - Attribute names are similarly encoded by the character ‘A’ followed by a uniquely assigned attribute-ID.

XGRIND

- Enumerated-type Attribute Value
  - Identifies the value by examining the DTD of the document and encodes their values using a simple $\log_2 K$ encoding scheme.

XGRIND

- General Element/Attribute Value
  - – A context-free compression scheme is required for efficiently querying.
    - The code assigned to a string in the document is independent of its location in the document.
    - This feature allows, given an arbitrary string, to locate occurrences of that string in the compressed document directly, without decompressing it.

XGRIND

- General Element/Attribute Value
  - Context-free compression scheme
    - First pass : Compute a separate frequency distribution table for each element and non-enumerated attribute.
    - Second pass : Compress at the granularity of individual element/attribute values.

XGRIND

- Homomorphic Compression
  - **The output, like the input, is semi-structured** in nature.
    - **The variety of efficient techniques** available for parsing/querying XML documents can be used.
    - **Indexes** can be built on the compressed document.
    - **Updates** to the XML document can be directly executed on the compressed version.
    - A compressed document can be checked for validity against the compressed version of its DTD.

Example

<song>
<title>No.2</title>
<artist>BoB</artist>
<publisher>R record</publisher>
<length>4:30</length>
<date>1999/1/1</date>
</song>
<song>
…

Song.xml

T1
T2 nahuff(No.2) /
T3 nahuff(Bob) /
T4 nahuff(R record) /
T5 nahuff(4:30) /
T6 nahuff(1999/1/1) /
/
T1
…

Output XML

+

Statistic Information (frequency)

Querying a Compressed XML

- Query can be carried out over the compressed document without fully decompressing it.
  - XPath Query
    - /song[artist="BoB"]/title ➔ /T1[T3=nahuff(Bob)]/T2
  - By query data value (such as "BoB")
    - exact-match
    - prefix-match

    direct processing

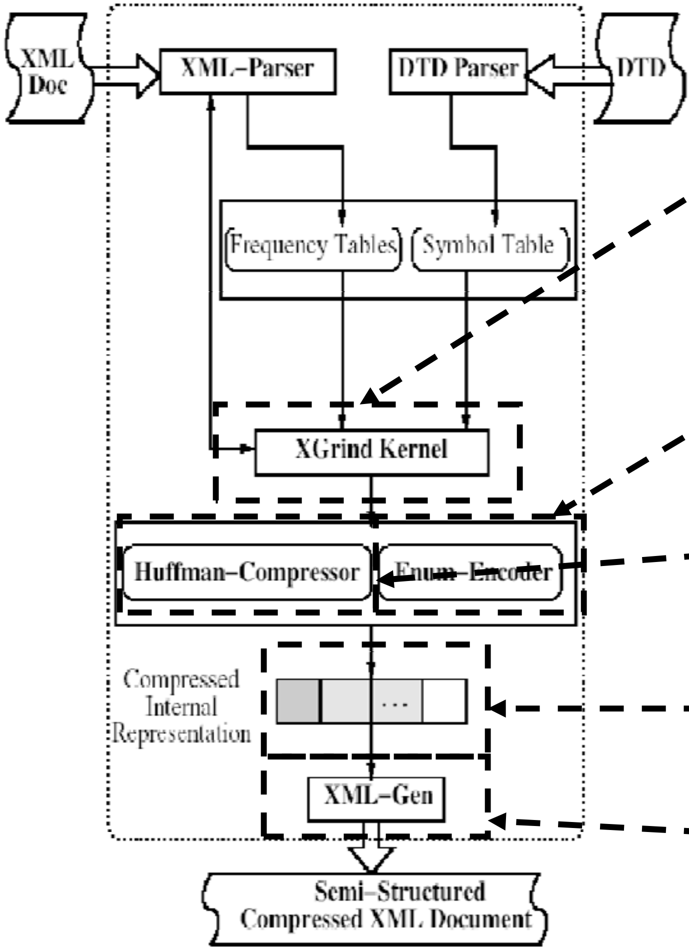
    - range-match
    - partial-match

    need partial decompression

T1
T2 nahuff(No.2) /
T3 nahuff(Bob) /
T4 nahuff(R record) /
T5 nahuff(4:30) /
T6 nahuff(1999/1/1) /
/
T1
…

Output XML

System Architecture



- **XGrind Kernel**

: **the heart of the compressor**
  - Invokes the DTD Parser.
  - Invokes the XML Parser.
- **Enum-Encoder**
  - **Used for meta-data and enumerated-type data items**.
- **Huffman-Compressor**
  - **Used for non-enumerated data items**.
  - Implements the non-adaptive Huffman coding compression scheme.
- **Compressed Internal Representation (CIR)**
  - The compressed output of the encoders.
- **XML-Gen**
  - Converts the CIR into a semi-structured compressed XML document.

Performance

- Result
  - Compression performance is, on the average, about 77% of XMill.
    (The worst case is within 68% of XMill.)
  - Compression time is always within about twice the time taken by XMill.
    (But, the compression is a ‘one-time’ operation, querying is a repeated occurrence.)

XPress

- To overcome the verbosity problem
- Some XML compressors do not support querying compressed data (XMILL)
- Other XML compressors which support querying compressed data blindly encode tags and data values using predefined encoding methods. (XGRIND)
  - XPRESS: A Queriable Compression for XML Data, Jun-Ki Min, Myung-Jae Park, Chin-Wan Chung, SIGMOD 2003

Contribution

- Supports direct and efficient evaluations of queries on compressed XML data
- Novel combination of characteristics:
  - ✓ Reverse Arithmetic Encoding
  - ✓ Automatic Type Inference
  - ✓ Apply Diverse Encoding Methods to Different Types
  - ✓ Semi-adaptive Approach
  - ✓ Homomorphic Compression

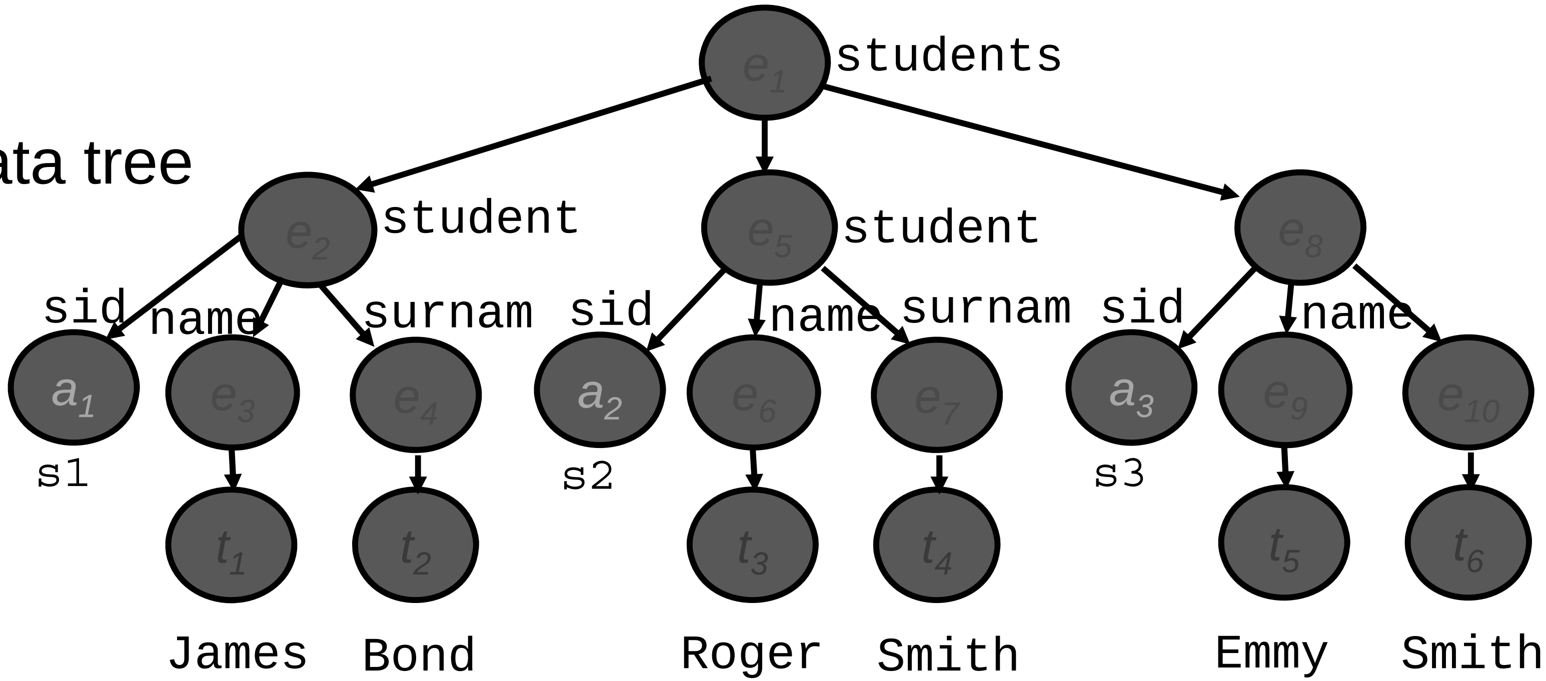
XQueC

- Query- unaware systems:
  - XMill: exploits data semantics
  - XMLPPM: exploits classical PPM (Predicted by Partial Matching) algorithms
- Query- aware systems:
  - XGrind: an “homomorphic” compressor/basic query processor
  - XPress: an “homomorphic” compressor/slightly extended query processor
- XQueC
  - improves over previous works, by covering arbitrarily complex queries in the compressed domain
    - A. Arion, A. Bonifati, I. Manolescu, and A. Pugliese. XQueC: A Query-Conscious Compressed XML Database. *Transactions on Internet Technology*, 7(2), ACM, 2007.

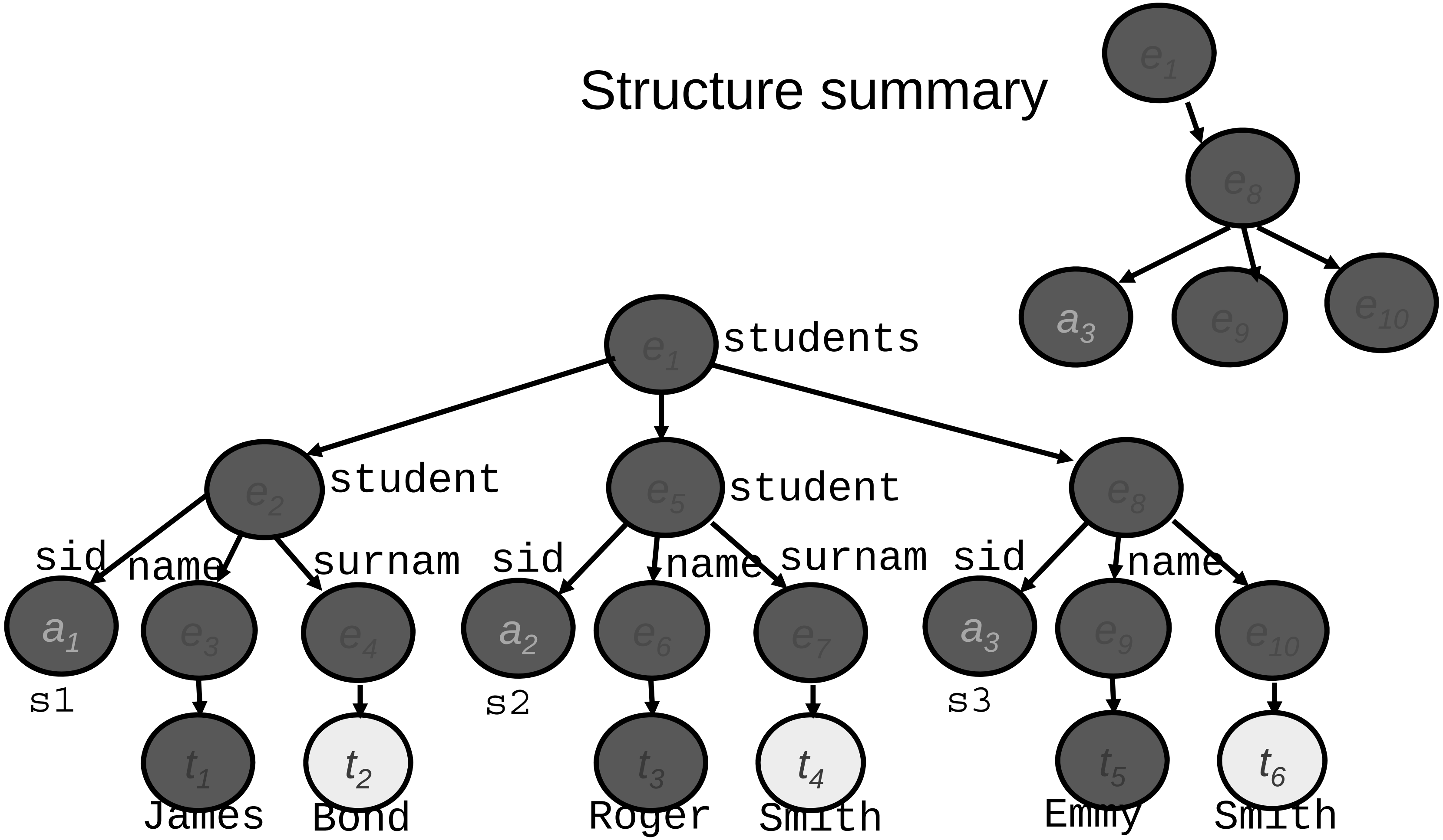
XQueC Idea

- Associate a container to each <type, root-to-leaf Path Expression>
- Keep the content separated from the structure by means of pointers

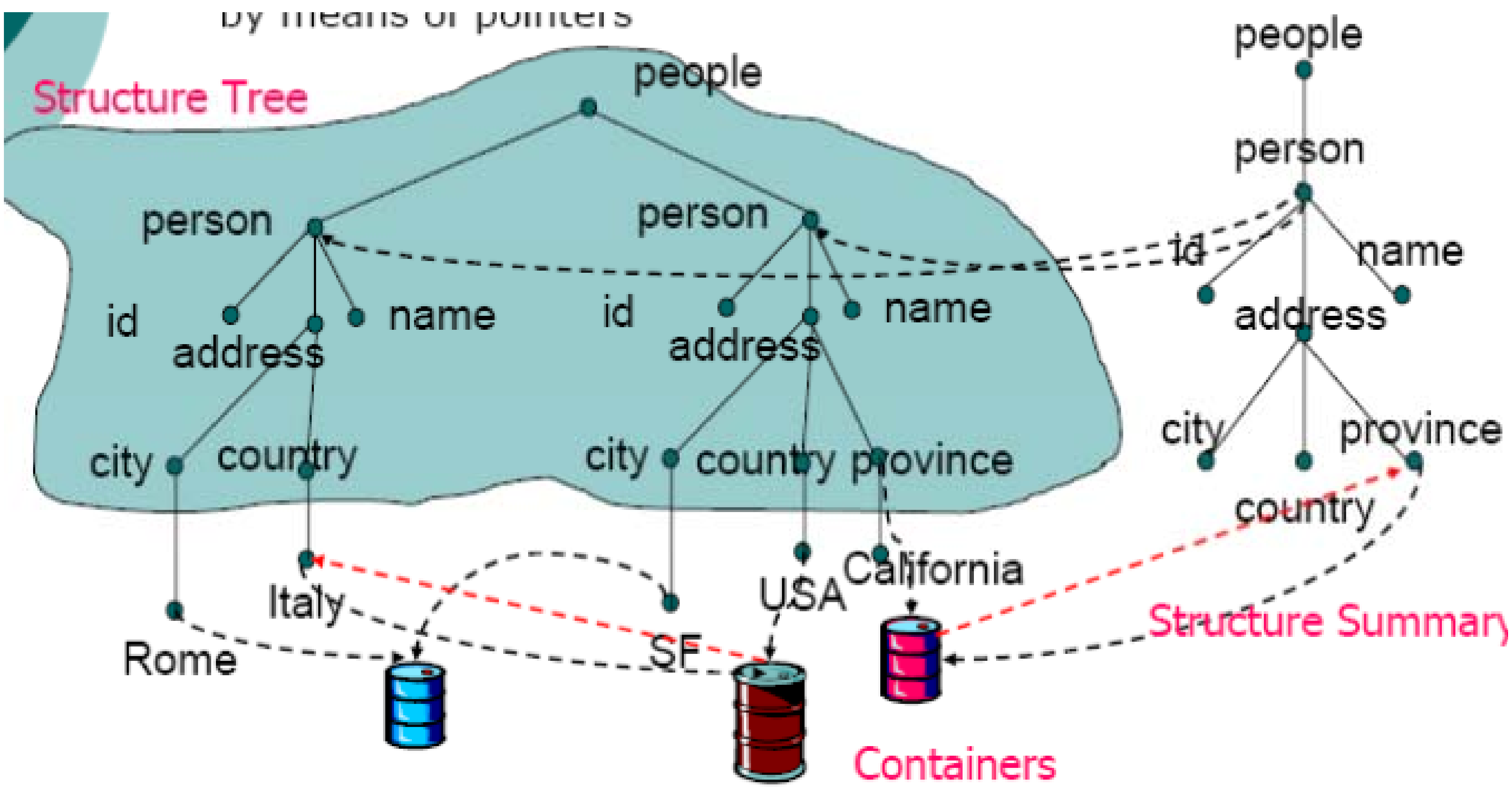
Data tree



XQueC Idea


Structure summary
e1
e8
a3
e9
e10
e1
students
e2
student
e5
student
e8
sid
name
surnam
sid
name
surnam
sid
name
a1
e3
e4
a2
e6
e7
a3
e9
e10
s1
s2
s3
t1
t2
t3
t4
t5
t6
James
Bond
Roger
Smith
Emmy
Smith

Structure Summary and Containers



XQueC

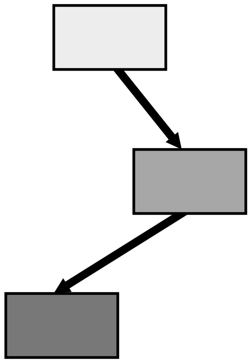
- Disobeying “homomorphism” is not a drawback
- Consider //A/C in XGrind
  - <a> <c> hello </c>
  - T1 T2 nahuff(hello) / /
- In XGrind, same parsing as in decompressed data

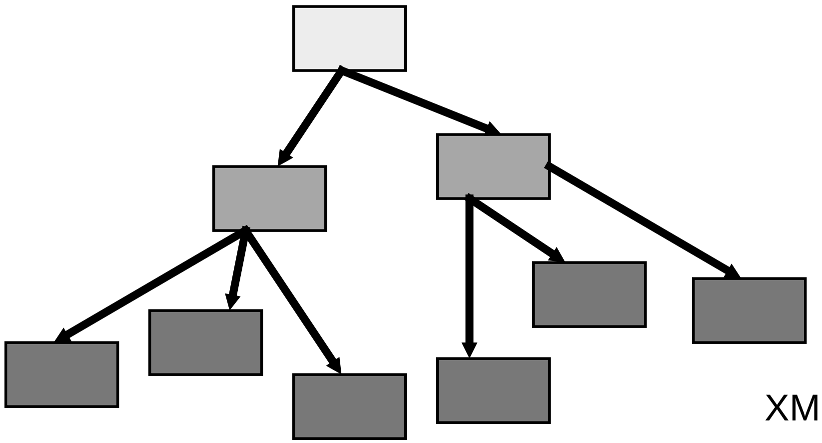
<a>	**<c>**	**boy**	**</c>**	****			**<a>**	**<c>**	**hello**	**</c>**	****		

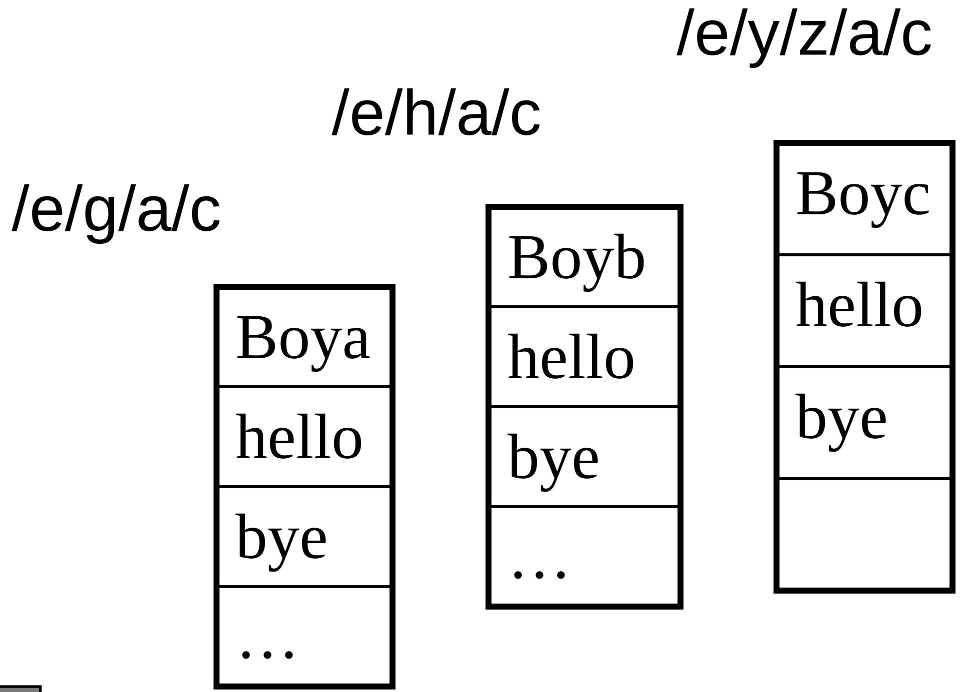
Starting

XQueC

- Disobeying “homomorphism” is not a drawback
  - XQueC tries to simplify the querying
    - E.g. //a/c







CONTAINERS

More on Containers

- They are kept lexicographically ordered
- Document order can be easily reconstructed looking at the structure summary
- Containers happen to be efficient pre-processed indexes (resemble B+trees on value columns)
- Binary search on containers: logarithmic cost

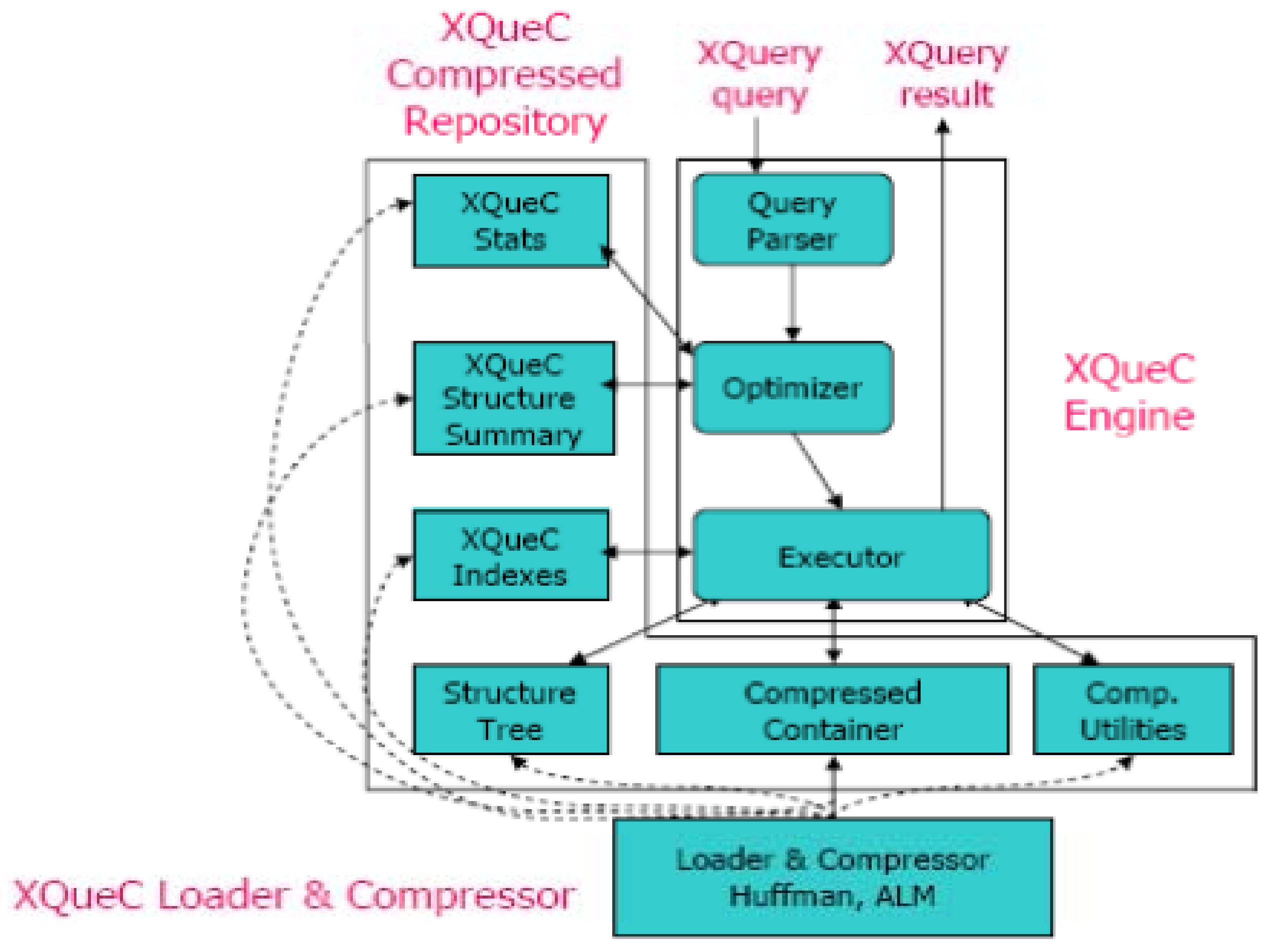
Querying

- Consider a query, like:

```
FOR $c IN /A/B/C,
$b IN /A/B
WHERE $c/text() > $b/ text()
RETURN $c
```

- In XQueC, direct access to the corresponding containers
  – pay the effort of fetching in memory only two containers plus (part of) the structure summary
- In XGrind and XPress,
  – have to fetch in memory all the XML file, keep part of it and decompress

XQueC Components



XQueC

- XQueC uses data fragmentation for advantageous query evaluation
- http://dns.isi.cs.cnr.it/isi/bonifati/xquec/

Further Work

- Impetus for incremental update of existing XML data sets is increasing
  - XML-based office document standards: ODF, OOXML
  - Increased volume of persistent XML data
- W3C has recently proposed an extension to XQuery for expressing node-level updates
- So far, most approaches to XML compression have assumed a "read-only" model
  - How amenable are the existing schemes to updates?