Type-based compression of XML data

DCC • 29 March 2007

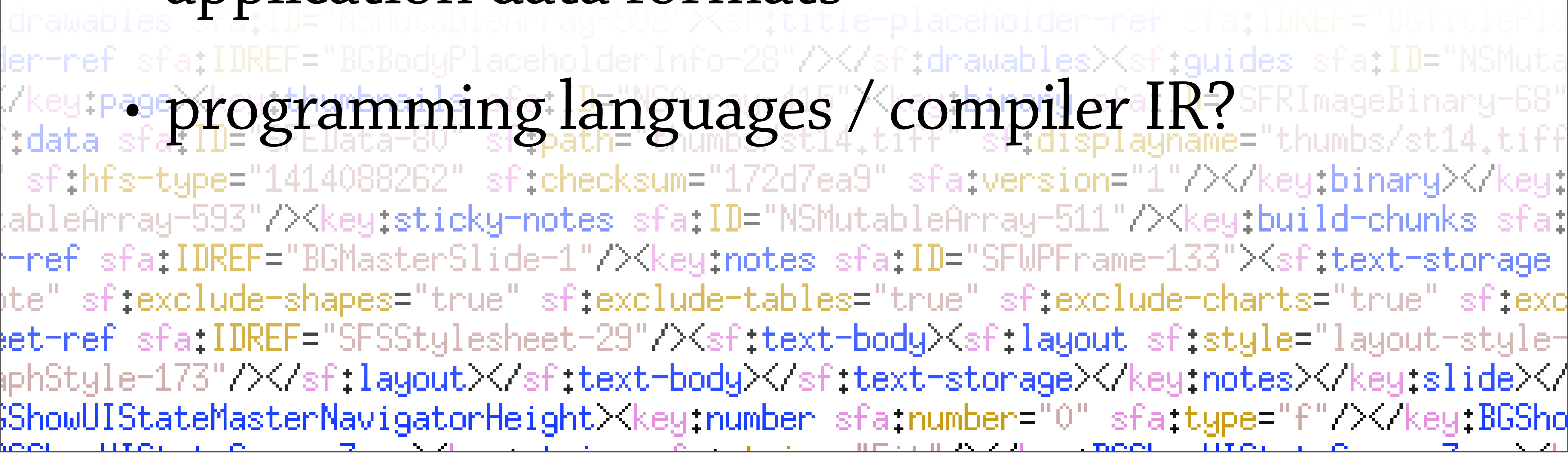
Christopher League
Kenjone Eng

LONG ISLAND UNIVERSITY

```
 sf:hfs-type="1414088262" sf:checksum=                    sfa:version="1"/></key:binary></key:
ableArray-593"/><key:sticky-notes sfa:ID="NSMutableArray-511"/><key:build-chunks sfa:
-ref sfa:IDREF="BGMasterSlide-1"/><key:notes sfa:ID="SFWPFrame-133"><sf:text-storage
te" sf:exclude-shapes="true" sf:exclude-tables="true" sf:exclude-charts="true" sf:exc
eet-ref sfa:IDREF="SFSStylesheet-29"/><sf:text-body><sf:layout sf:style="layout-style-
phStyle-173"/></sf:layout></sf:text-body></sf:text-storage></key:notes></key:slide></
ShowUIStateMasterNavigatorHeight><key:number sfa:number="0" sfa:type="f"/></key:BGSho
```

XML has become indispensible

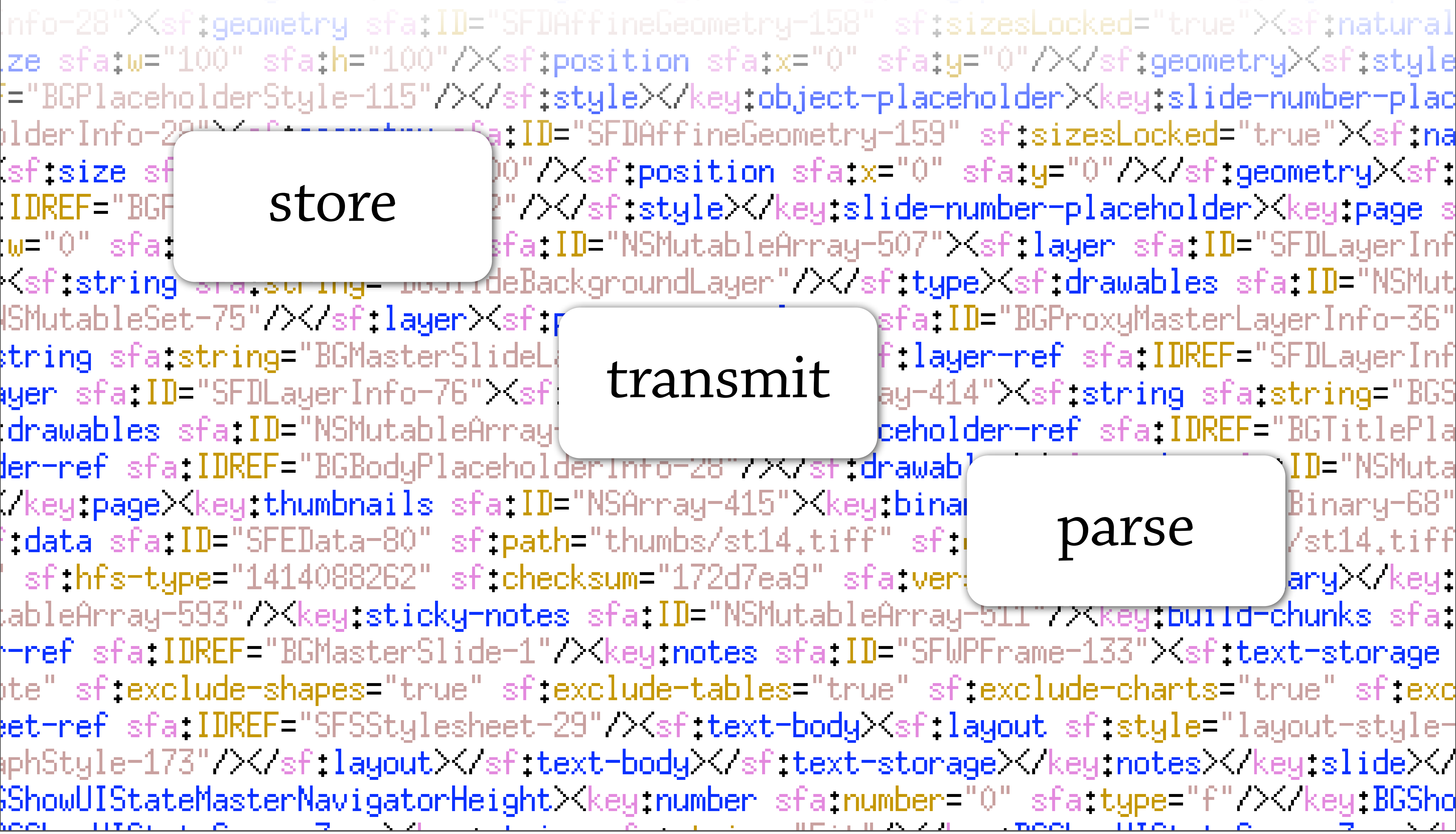
- web services
- document markup
- conduits between databases
- application data formats
- programming languages / compiler IR?



```
:drawables sfa:ID="NSMutableArray-592"><sf:title-placeholder-ref sfa:IDREF="BGTitlePla
der-ref sfa:IDREF="BGBodyPlaceholderInfo-28"/></sf:drawables><sf:guides sfa:ID="NSMuta
</key:page><key:thumbnails sfa:ID="NSArray-415"/><key:binary sfa:ID="SFRImageBinary-68"
:data sfa:ID="SFEData-80" sf:path="thumbs/st14.tiff" sf:displayname="thumbs/st14.tiff
" sf:hfs-type="1414088262" sf:checksum="172d7ea9" sfa:version="1"/></key:binary></key:
tableArray-593"/><key:sticky-notes sfa:ID="NSMutableArray-511"/><key:build-chunks sfa:
-ref sfa:IDREF="BGMasterSlide-1"/><key:notes sfa:ID="SFWPFrame-133"><sf:text-storage
te" sf:exclude-shapes="true" sf:exclude-tables="true" sf:exclude-charts="true" sf:exc
eet-ref sfa:IDREF="SFSStylesheet-29"/><sf:text-body><sf:layout sf:style="layout-style-
aphStyle-173"/></sf:layout></sf:text-body></sf:text-storage></key:notes></key:slide></
SShowUIStateMasterNavigatorHeight><key:number sfa:number="0" sfa:type="f"/></key:BGSho
```

Thanks. XML has become indispensible in for many different applications. What brought me to this effort was the idea of representing PLs and compiler IRs in XML. ¶ But of course, …

XML is verbose



nfo-28"><sf:geometry sfa:ID="SFDAffineGeometry-158" sf:sizesLocked="true"><sf:natural
ize sfa:w="100" sfa:h="100"/><sf:position sfa:x="0" sfa:y="0"/></sf:geometry><sf:style
="BGPlaceholderStyle-115"/></sf:style></key:object-placeholder><key:slide-number-plac
lderInfo-29" a:ID="SFDAffineGeometry-159" sf:sizesLocked="true"><sf:na
<sf:size sf 00"/><sf:position sfa:x="0" sfa:y="0"/></sf:geometry><sf:
:IDREF="BGF 2"/></sf:style></key:slide-number-placeholder><key:page s
w="0" sfa: sfa:ID="NSMutableArray-507"><sf:layer sfa:ID="SFDLayerInf
<sf:string deBackgroundLayer"/></sf:type><sf:drawables sfa:ID="NSMut
NSMutableSet-75"/></sf:layer><sf:p sfa:ID="BGProxyMasterLayerInfo-36"
string sfa:string="BGMasterSlideL f:layer-ref sfa:IDREF="SFDLayerInf
ayer sfa:ID="SFDLayerInfo-76"><sf: ay-414"><sf:string sfa:string="BGS
drawables sfa:ID="NSMutableArray- ceholder-ref sfa:IDREF="BGTitlePla
der-ref sfa:IDREF="BGBodyPlaceholderInfo-28"/></sf:drawab ID="NSMuta
</key:page><key:thumbnails sfa:ID="NSArray-415"><key:binar Binary-68"
:data sfa:ID="SFEData-80" sf:path="thumbs/st14.tiff" sf: /st14.tiff
 sf:hfs-type="1414088262" sf:checksum="172d7ea9" sfa:ver ary></key:
ableArray-593"/><key:sticky-notes sfa:ID="NSMutableArray-511"/><key:build-chunks sfa:
-ref sfa:IDREF="BGMasterSlide-1"/><key:notes sfa:ID="SFWPFrame-133"><sf:text-storage
te" sf:exclude-shapes="true" sf:exclude-tables="true" sf:exclude-charts="true" sf:exc
eet-ref sfa:IDREF="SFSStylesheet-29"/><sf:text-body><sf:layout sf:style="layout-style-
phStyle-173"/></sf:layout></sf:text-body></sf:text-storage></key:notes></key:slide></
SShowUIStateMasterNavigatorHeight><key:number sfa:number="0" sfa:type="f"/></key:BGSho

store

transmit

parse

XML is very verbose. Shown here is part of the representation of this slide show, which is stored as gzipped XML. § So, because of its size, XML takes more space to store, more bandwidth to transmit, and more time to parse (compared to a custom binary format).

Schema specifies structure

⊢

```
<book>
 <title>Alaska</title>
 <author>James A. Michener</author>
 <price currency='USD'>11.96</price>
</book>
```

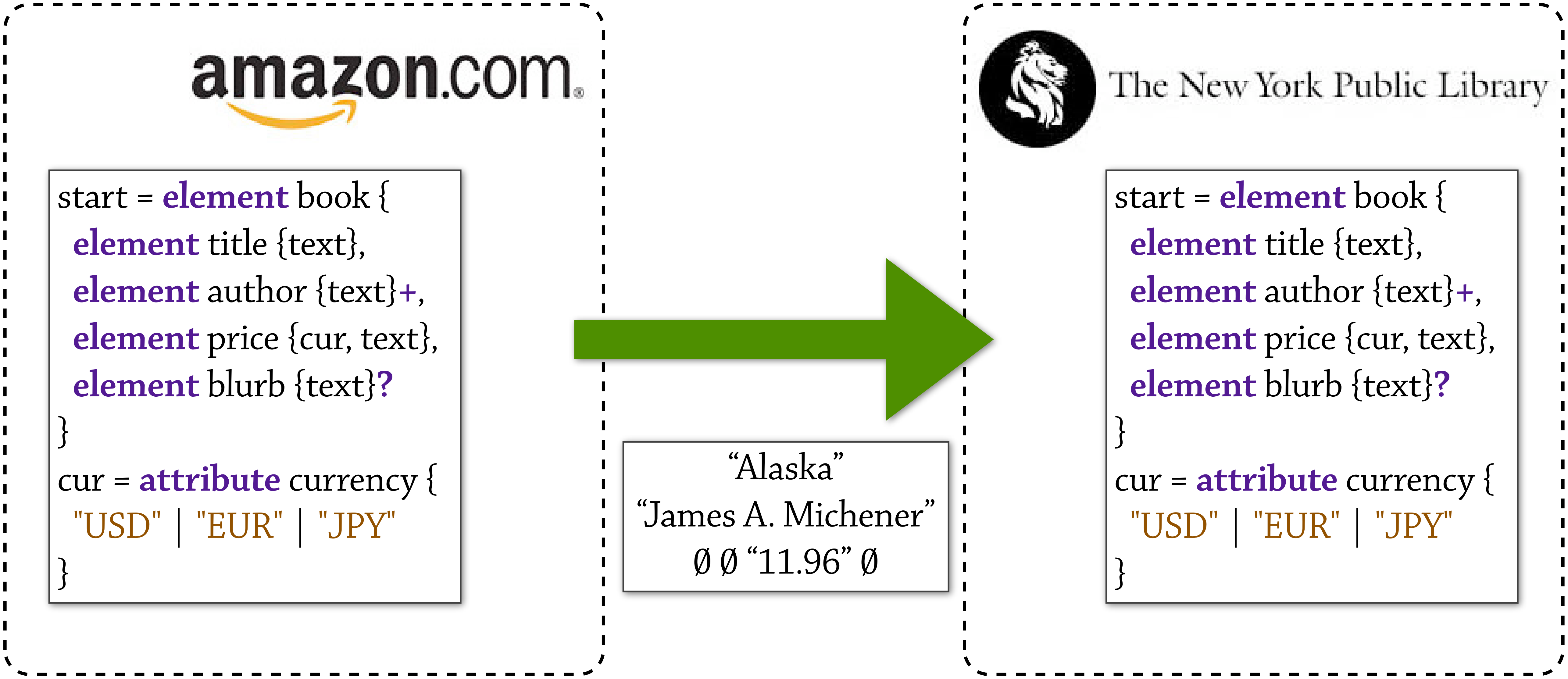
:

```
start = element book {
 element title {text},
 element author {text}+,
 element price {cur, text},
 element blurb {text}?
}
cur = attribute currency {
 "USD" | "EUR" | "JPY"
}
```

- schema languages:
  DTD, XML Schema (XSD), **Relax NG,** ...

Our approach begins with the concept of a **schema,** or document type. The schema specifies the structure of the XML: what tags and attributes are allowed where. ¶ Here's an example of a schema and an instance. I'm using a schema language called "Relax NG." Note the regex operators.

If sender & receiver agree on schema, transmission can be streamlined



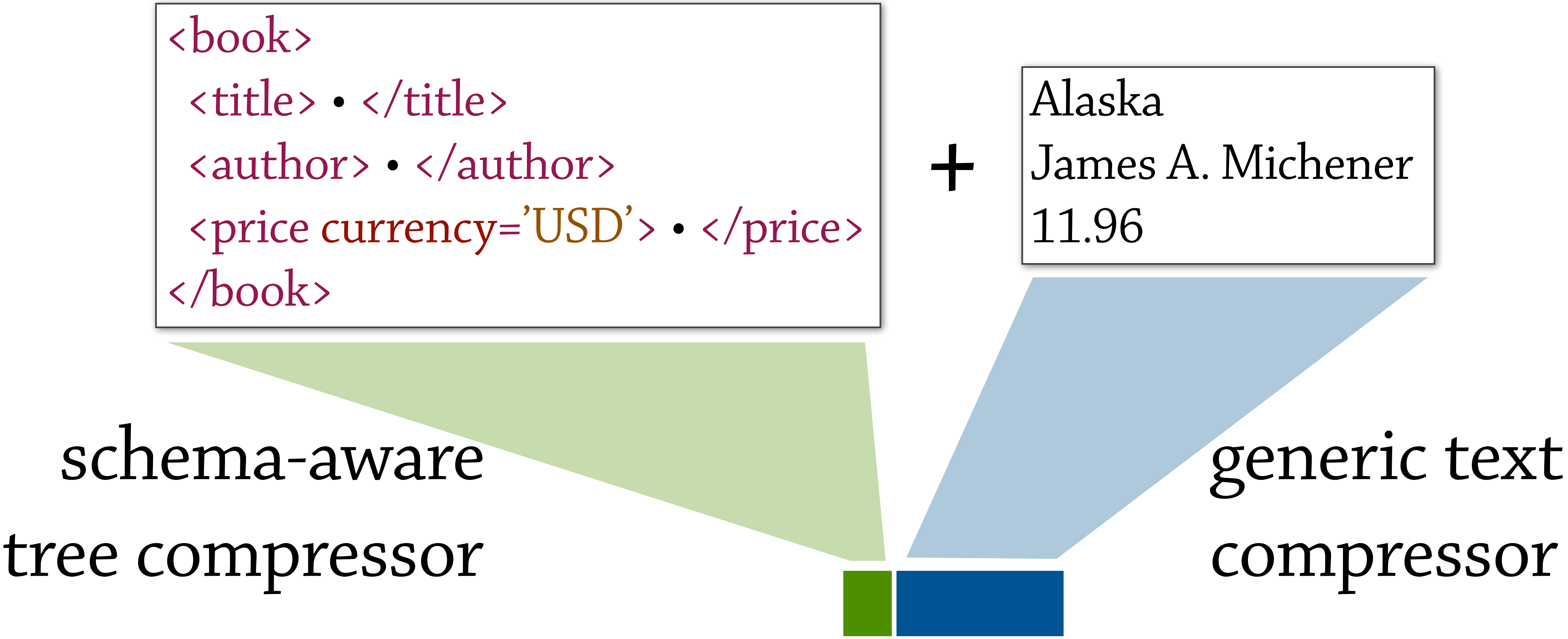
```
start = element book {
  element title {text},
  element author {text}+,
  element price {cur, text},
  element blurb {text}?
}
cur = attribute currency {
  "USD" | "EUR" | "JPY"
}
```

“Alaska”
“James A. Michener”
Ø Ø “11.96” Ø

```
start = element book {
  element title {text},
  element author {text}+,
  element price {cur, text},
  element blurb {text}?
}
cur = attribute currency {
  "USD" | "EUR" | "JPY"
}
```

Here’s the key insight: if sender & receiver (or reader & writer) agree on a schema, the transmitted information can be greatly abbreviated. Here we’re just sending the textual data, and a few numbers to indicate there’s just 1 author, no blurb, and the price is in U.S. dollars.

Extract tree structure from text and encode them separately



```
<book>
 <title> • </title>
 <author> • </author>
 <price currency='USD'> • </price>
</book>
```

+

Alaska
James A. Michener
11.96

schema-aware tree compressor

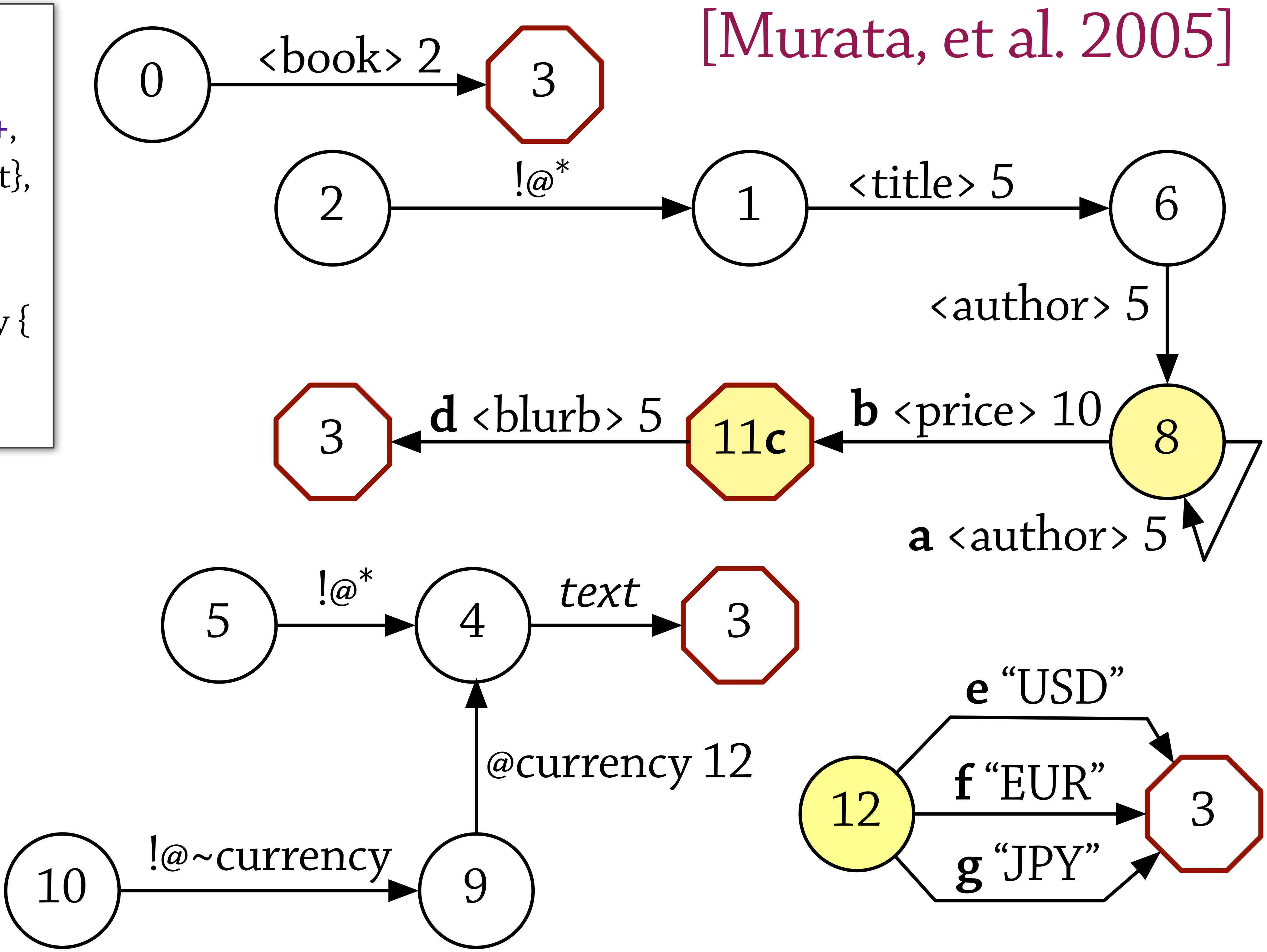
generic text compressor

We're going to extract the tree structure from the text and encode them separately. For the tree structure, we'll need the schema, and I'll describe that in a moment. For the text, we just pass it to a generic text compressor, such as gzip.

‘Relax NG’ schema induces a *tree automaton* used for validation

```
start = element book {
  element title {text},
  element author {text}+,
  element price {cur, text},
  element blurb {text}?
}
cur = attribute currency {
  "USD" | "EUR" | "JPY"
}
```

[Murata, et al. 2005]



One of the nice things about Relax NG is that it has a very clean formal model, in the form of tree automata. So I’ll show you how the book schema translates to an automaton. We start at state 0, and transitions are labeled with a tag name and state number. The red octagon is a final state. Subroutine...

```
<book><title>Alaska</title>
 <author>James A. Michener</author>
 <price currency='USD'>11.96</price>
</book>
```

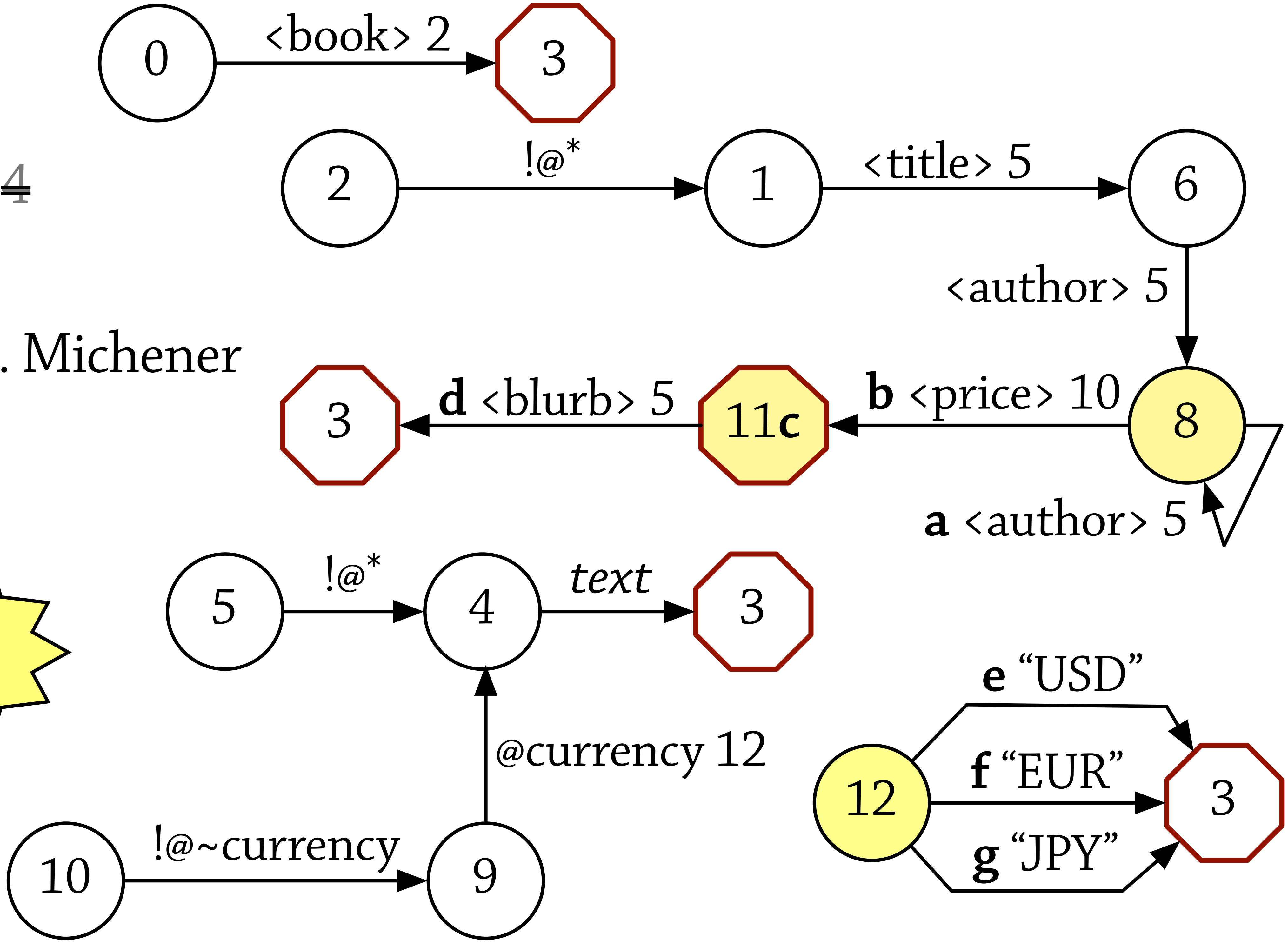
Note path taken at each choice point

stack: ~~3~~ ~~6~~ ~~8~~ ~~11~~ ~~4~~

data: Alaska
James A. Michener
11.96

tree: **b e c**

4 bits



The automaton was developed to validate XML documents against the schema, but I can use it to compress and decompress. We'll do a small example. § We'll need a stack to keep track of subroutine calls. …

Implementation exists (in Java)

```
Usage: rngzip [options] [file ...]

Options:
 -c --stdout                write to standard output; do not touch files
 -D --debug                 trace compressor; replaces normal output
 -E --tree-encoder=CODER    use method CODER for encoding the XML tree
 -f --force                 force overwrite of output file
    --ignore-checksum       decompress even if schema changed (not recommended)
 -k --keep                  do not remove input files
 -p --pretty-print[=TAB]    line-break and indent decompressed output [2]
 -q --quiet                 suppress all warnings
 -s --schema=FILE|URL       use this schema (required to compress)
 -S --suffix=.SUF           use suffix .SUF on compressed files [.rnz]
 -t --timings               output timings (implies -v)
 -T --tree-compressor=CM    compress the encoded XML tree using CM
 -v --verbose               report statistics about processed files
 -Z --data-compressor=CM    compress the data stream using CM

Modes:                      compress is the default; this requires -s
 -d --decompress            decompress instead of compress
 -i --identify              print information about compressed files
 -h --help                  provide this help
 -V --version               display version number, copyright, and license
    --exact-version         output complete darcs patch context

Coders: fixed *huffman
Compressors: none *gz bz2
```

[thanks to *Bali* library by Kawaguchi]

In implementing this technique, we benefitted greatly from the Bali library by Kawaguchi (@Sun). It parses the Relax NG specs and builds the automata. We had to post-process the automata slightly, but without that library, we never would've gotten off the ground. Our tool is in Java because the library is in Java.

Data sources

- gene — genome metadata from NCBI
- pubmed — bibliographic data from NCBI
- movies, actors — film data from IMDB
- sigmod, issue, proc — bibliographic data (ACM)
- niagara — from Niagara Query Engine (Wisc.)
- uw — course catalogs from U. Washington
- shakes — Shakespeare in XML (Jon Bosak)

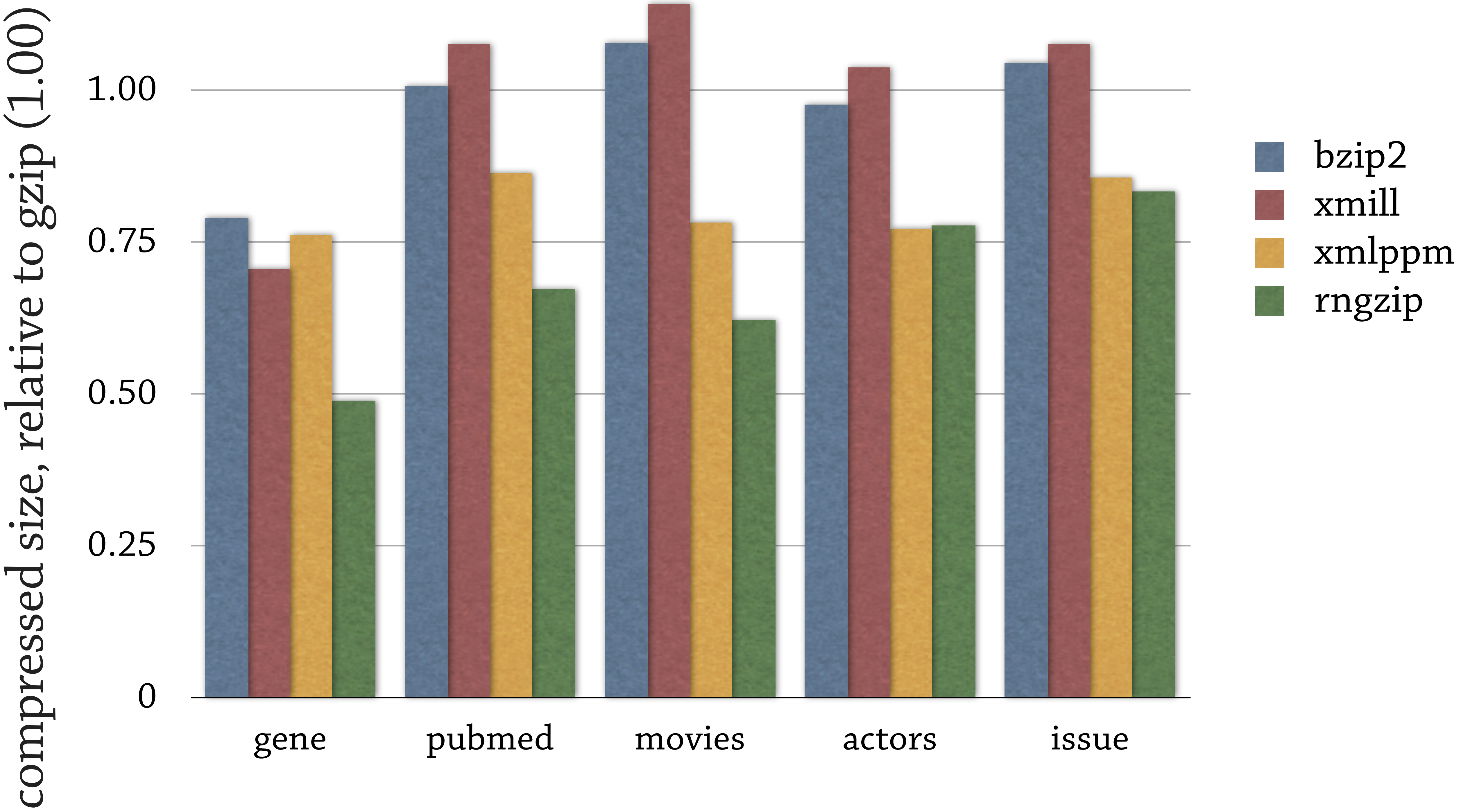
Now, for some empirical analysis. We used a variety of different data sources: some small, some large. Some are mostly tags, others mostly text, some in between.

The competition

• gzip, bzip2	
• XMill	[Liefke & Suciu 2000]
• XMLPPM*	[Cheney 2001]
• DTDPPM	[Cheney 2005]

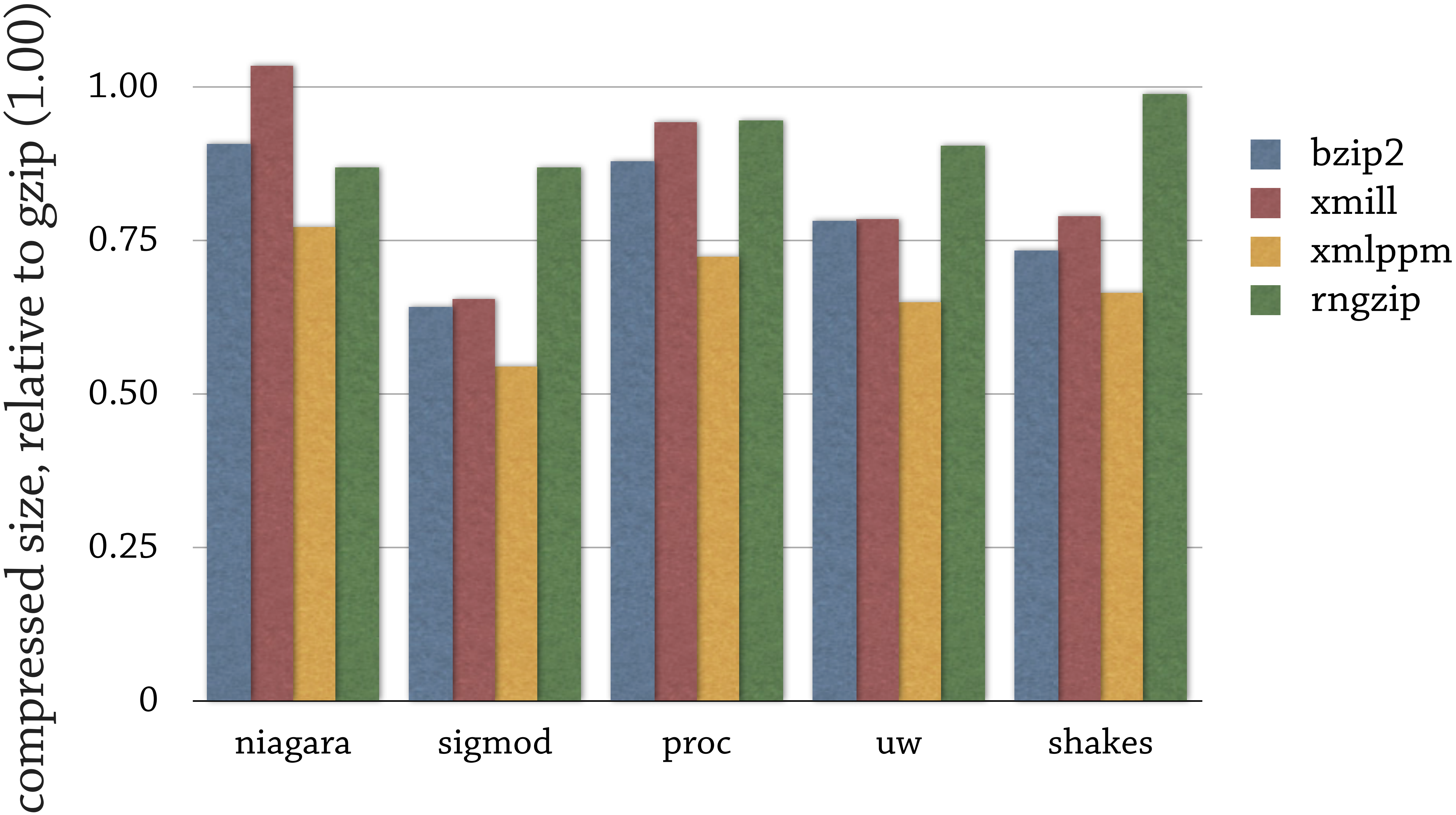
Our competition was the general-purpose compressors gzip and bzip, and some XML-aware systems: XMill and XMLPPM. DTDPPM is another schema-aware system, but it never did much better than XMLPPM. In our tests, XMLPPM was the one to beat.

The good news



And here are the results. Good news first! These are compressed file sizes, relative to gzip, at 1.0. On these data sets, we are doing significantly better than or comparable to xmlppm. gene and pubmed are very taggy but highly regimented. But it's not all good news…

Not-so-good news

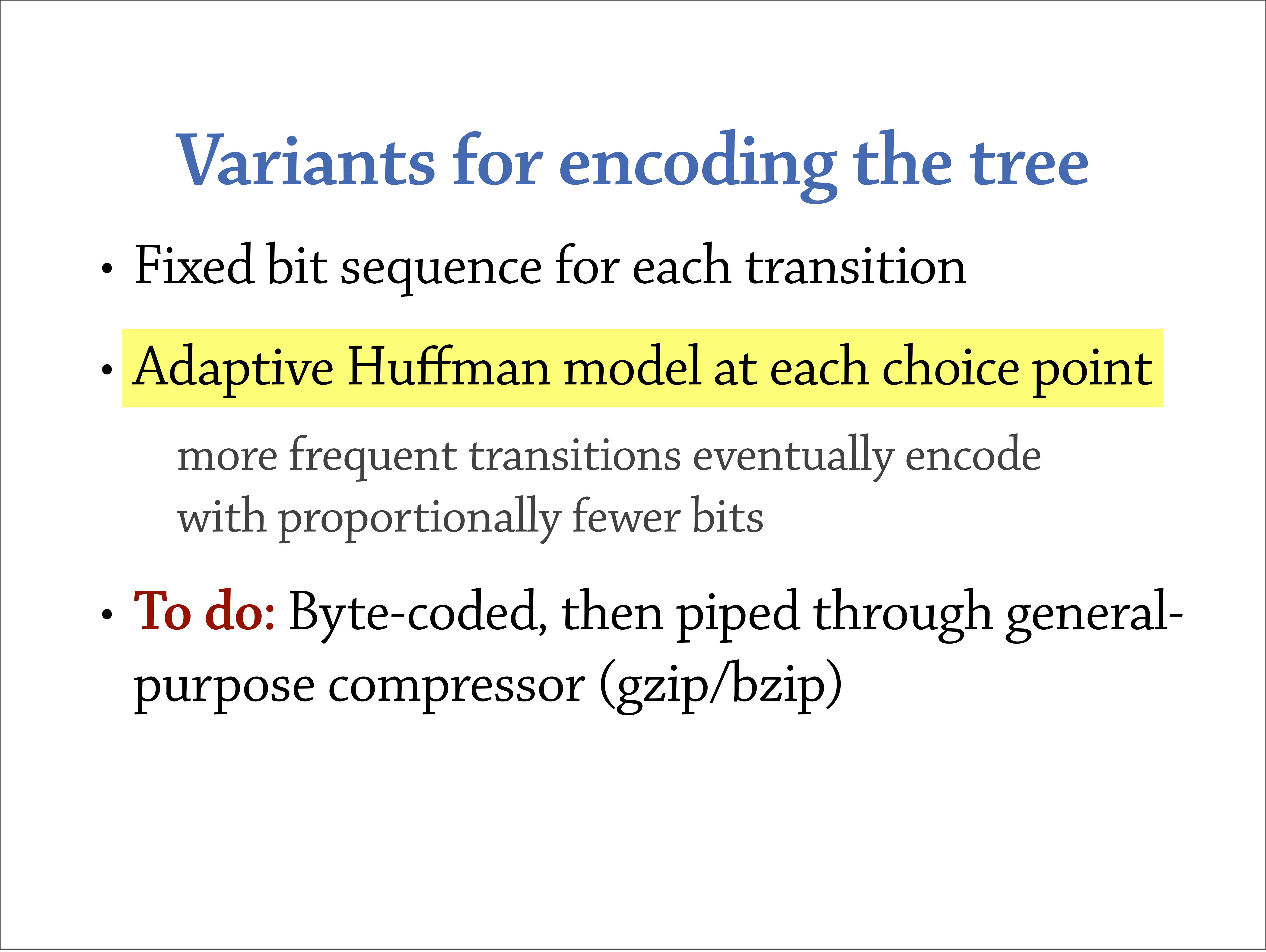


On some data sets, we do significantly worse. But, our textual data is just compressed with gzip, so it makes sense that our performance would degrade to that of gzip for text-oriented documents like Shakespeare. (Can easily substitute bzip.) ¶ So the results seem mixed, but I have high hopes. Here’s why…

Variants for encoding the text

- Pipe through gzip
- Pipe through bzip2
- **To do:** Use parent tags to fill separate data containers, like XMill
- **To do:** Use parent tags as context for prediction by partial match, like XMLPPM

For the textual part of the documents, we're being very naive. It's completely possible to adapt orthogonal techniques from XMill and/or XMLPPM here, using the tags to provide some additional context for compressing text nodes.



Variants for encoding the tree

- Fixed bit sequence for each transition
- Adaptive Huffman model at each choice point

  more frequent transitions eventually encode with proportionally fewer bits

- **To do:** Byte-coded, then piped through general-purpose compressor (gzip/bzip)

For encoding the tree, I mentioned in the example that we'd use a fixed number of bits for each choice point, based on the number of transitions. We also implemented an adaptive Huffman encoding so that It was suggested by a reviewer to try byte-coding…

Related work

- Sundaresan & Moussa [2002] proposed "differential DTD compression"
  - report poor run-time performance;
  - unable to compress *Hamlet*

Some other researchers have thought of using document type information in similar ways. …

More related work

- Toman [2004] dynamically infers custom grammar for each document; uses automata
    - never beats xmlppm

Most related work

- Levene & Wood [2002] have nearly the same idea, for DTD
  - but no implementation / empirical results
  - they prove an optimality result — assuming non-recursive document type

Future directions

- Enable streaming / online compression
  - a property of Relax NG makes it difficult
- Native Relax NG data sets? (OpenLaszlo)
- Support Relax NG datatype library
  - need specialized encoders for dates, *n*-bit integers, base-64 binary, DNA sequences, ...

Thanks!

christopher.league@liu.edu