BibTEX++
Towards Higher-order BibTeXing

Laura BARRERO SASTRE Fabien DAGNAT Emmanuel DONIN DE ROSIÈRE

Ronan.KERYELL@enst-bretagne.fr

Nicolas TORNERI

—

Laboratoire Informatique & Télécommunications

Département Informatique

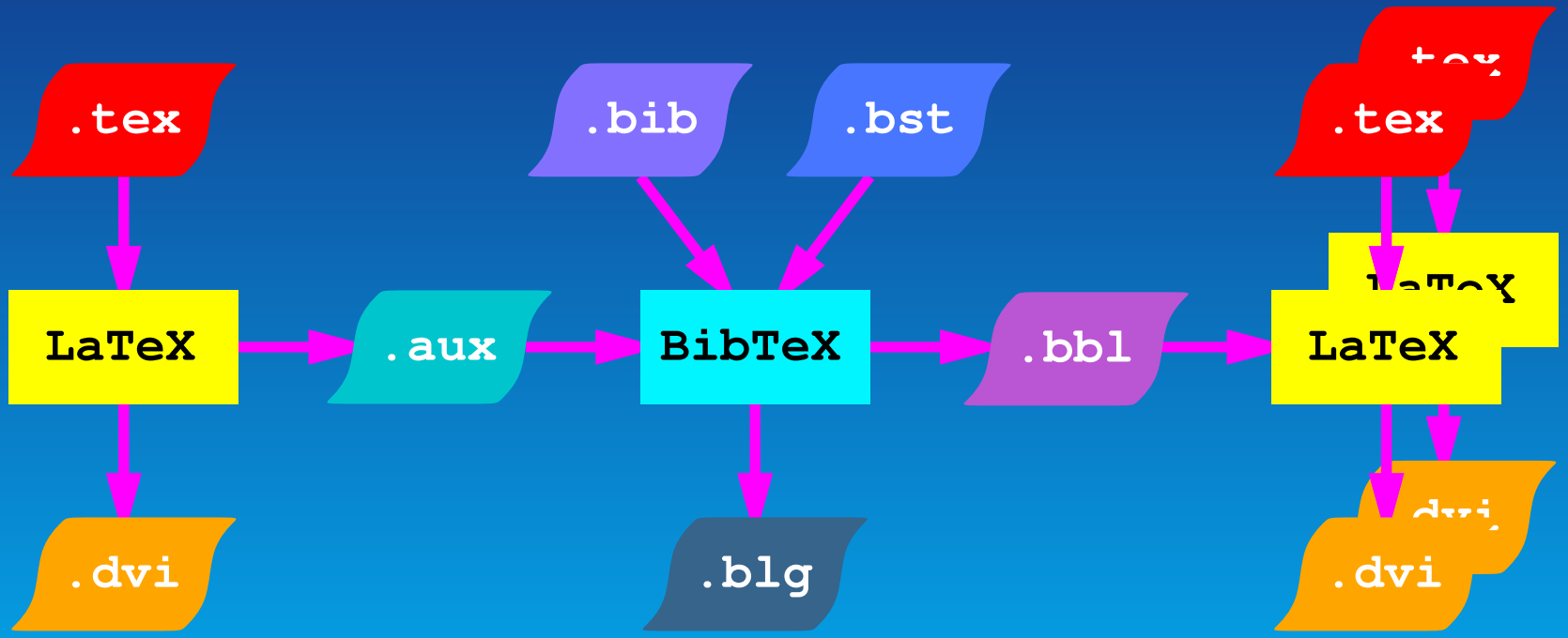
École Nationale Supérieure des Télécommunications de Bretagne

26 juin 2003

BibTEX is good for you...

- *THE* bibliographical tool in LATEX
- Widely used
- Follow logical concepts from the LATEX world
  - Citation database
  - Bibliography style
  - *Automatic* generation from cited references
  - Typeset further by LATEX
- Huge existing matter available
  - Large bibliography databases (`http://citeseer.nj.nec.com/cs`,. . .)
  - Great amount of styles for many journals, books,. . .

BibTEX is good for you...



...but...

- Old tool : from the 80's (well LaTeX too... ☺)
- No longer evolves (only improved to accept 8-bit characters around 1990)
- Programmable... but in an awful 60's stack based language (BST) for aliens from the outerspace
  - Trivial to parse and execute by the computer, easy implementation in BibTeX
  - Just put the burden on the style programmer ☺

```
FUNCTION {sort.format.names}
{ 's :=
  #1 'nameptr :=
  ""
  s num.names$ 'numnames :=
  numnames 'namesleft :=
    { namesleft #0 > }
```

...but...

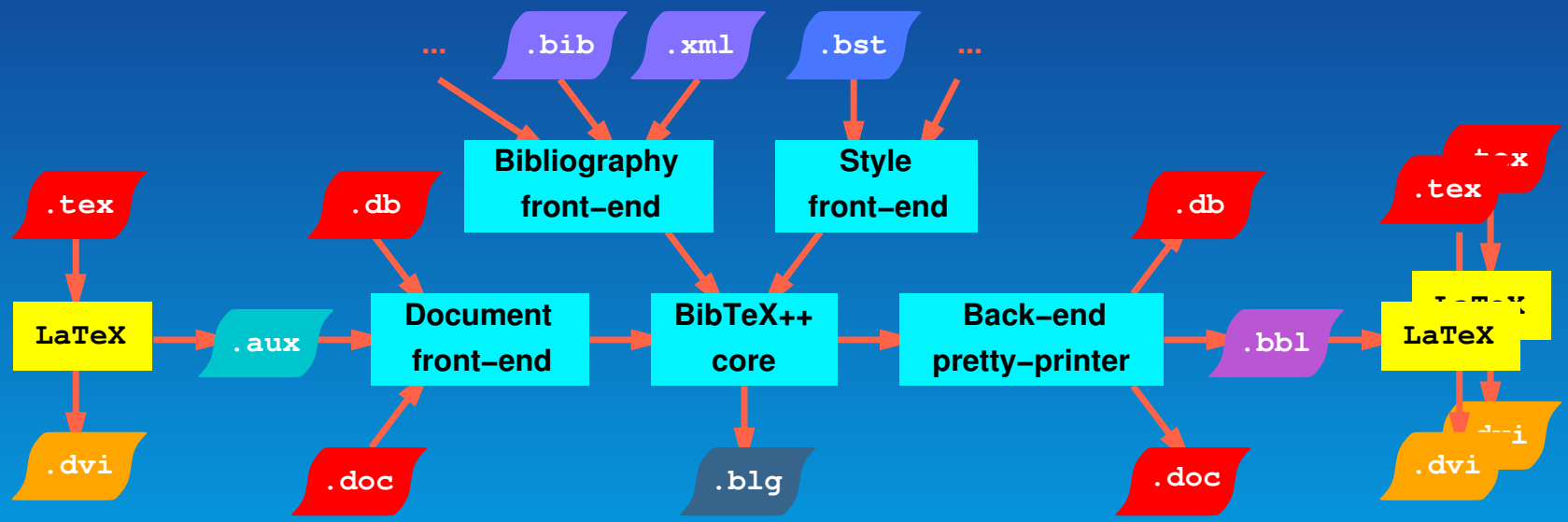
```
    { nameptr #1 >
        { "   " * }
        'skip$
      if$
      s nameptr "{vv{ } }{ll{ }}{  ff{ }}{  jj{ }}" format.name$ 't :=
      nameptr numnames = t "others" = and
        { "et al" * }
        { t sortify * }
      if$
      nameptr #1 + 'nameptr :=
      namesleft #1 - 'namesleft :=
    }
  while$
}
```

1258 such lines in `alpha.bst`...

New needs:

- Multilingual
- UNICODE
- Access to bibliography database from the Internet
- Programming language with far more expressiveness
- Unbound scalability & extensions
- Compatible with existing bibliography styles
- Generic tool suited for many other layout software
- YAB (Yet Another BibTEX)?

BibTEX++ basic architecture



Adopt the object attitude

- Need to choose a clearer language than BST
- Designing a domain specific language?
  - Good for selfishness ☺
  - Yet another (less) cryptic language to learn
  - With lack of expressiveness?
  - Oh. . . just improve the language! ☺
- Or use a classical computer language for all the expressiveness we want!
- Put all the domain specific stuff in objects: bibliographical objects
- OK since no need for performance
- But into what language?

- What we want:
  - Portable
  - Clean object support and syntax
  - Can deal with big programs
  - Lot of library for all the modern way of life: UNICODE, Internet,. . .
  - Well known to avoid the *yet another weird-language to learn* syndrome
- Trade-off
- Let's go for Java

- Bibliography styles are *directly* written in Java for expressiveness and simplicity
- Generic library functions and classes to deal with bibliographies in BibTEX++ rewritten in Java
- Portable
- UNICODE natively accepted
- Inherit all the Java locale stuff and even specialized collators for international sorting so important in BibTEX
- Lots of `hook`s to modify the internal behaviour
- Keep the information message `.blg`

Bibliography back-end

- The easy part
- Just a prettyprinter of the abstract internal representation
- Add a prettyprinter for each output language
  - Generate a `.bbl` file for LaTeX
  - ...

Front-ends

- Deal with all BibTEX++ inputs
- Use the SableCC parser generator (to build kinds of super-DOM generic parser and library)
- Build the abstract internal representations
- Available front-ends:
  - Document front-end analyzes the `.aux` file or other to pick requested citations
  - Bibliography database front-end accepts bibliography items
  - Style front-end deal with style programming

A BST compiler

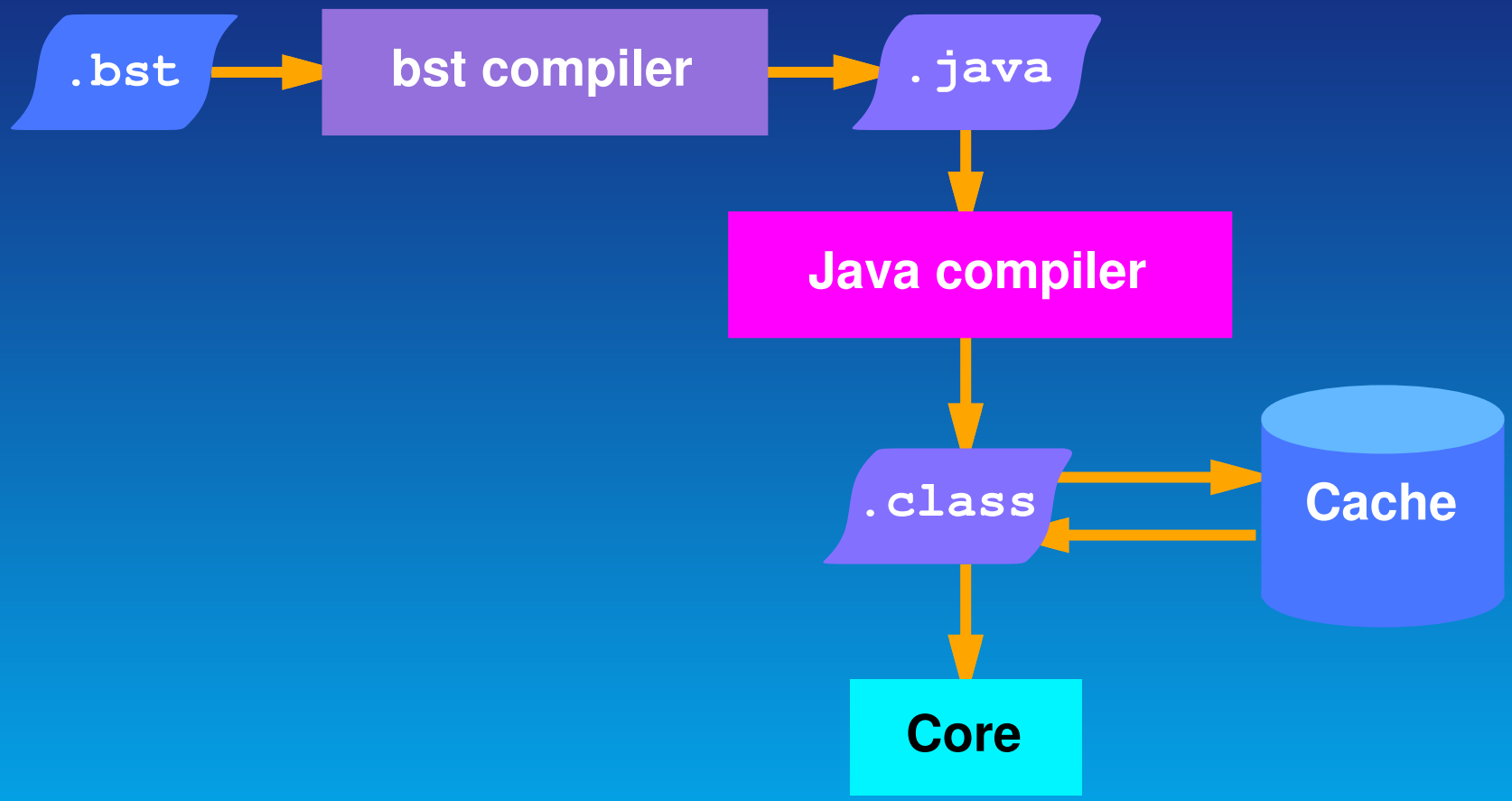
- Big legacy BibTEX style (`.bst`) available : 150 files in mikTEX

  ⇝ Straight traduction of BST code to clearer Java

  Rely on advanced compilation and program re-engineering techniques such as those developed in the PIPS project at the École des Mines de Paris

- A generate Java style can be used later as a basis for new style developments

A BST compiler



- Use a cache of compiled BST styles to speed up operations

```
public String sort_format_names( String s0 )
{
    String s1;
    int i1 , i2;
    s = s0;
    nameptr = 1;
    s0 = "";
    i1 = BuiltIn.numnames( s );
    numnames = i1;
    namesleft = i1;
    while( i1 > 0 )
    {
        if( nameptr > 1 )
        {
```

10

```
    s0 = s0 + " ";
}
s1 = BuiltIn.formatName( s , nameptr , "{vv{ } }{ll{ }}{ ff{ }}
t = s1;
i1 = BuiltIn.equal( numnames , nameptr );
i2 = BuiltIn.equal( "others" , s1 );
i1 = and( i1 , i2 );
if( i1 > 0 )
{
    s0 = s0 + "et al";
}
else
{
    s1 = sortify( t );
```

```
        s0 = s0 + s1;
      }                                                    30
      nameptr = nameptr + 1;
      i1 = namesleft − 1;
      namesleft = namesleft − 1;
    }
    return( s0 );
  }
```

- How does it work?

Stack removal

- Well. . . BST is a stack oriented language but Java is not ☹
- Up to this year this stack was still in the generated Java code
- Stack removal is old stuff in computer science but it comes back because of efficient JIT JVM implementation
- Code transformation from stack based to imperative style with variables borrowed from Forth to C compiler

- BST is not a typed language
- A stack element can hold anything
- But polymorphism is not really used by BST operators, it is rather a design simplification
- No function signature (what arguments? what returned values?)
- Java is a typed language : nice if lacking BST types can be infered ⇝ more understandable Java code

Type reconstruction

- Bottom-up approach from BibTEX operators well-known type

  `format.name$`

  produces a string on the stack from a string, an integer and a string on the stack

- Propagate all the known types interprocedurally through all the code

- When ambiguous, keep polymorphic `Cell` objects

Unbalanced stack

- Stack removal assumes that each BST block's stack usage can be mapped on a fixed amount of variables
- What if a BST code stack usage depends on values? Code generation with static variable allocation not possible ☹
- If stack removal fails, keep all the clean code and wrap it in a stack environment
- Such codes do really exist!
  Just in `plain.bst`!

Unbalanced stack

```
FUNCTION {format.names}
{ 's :=  #1 'nameptr :=
 s num.names$ 'numnames :=
 numnames 'namesleft :=
  { namesleft #0 > }
  { s nameptr "{ff~}{vv~}{ll}{, jj}"
      format.name$ 't :=
     nameptr #1 >
      { namesleft #1 >
         { ", " * t * }
         { numnames #2 >
            { "," * }
            'skip$
          if$
          t "others" =
            { " et~al." * }
            { " and " * t * }
            if$
          }
        if$
       }
        't
    if$
    nameptr #1 + 'nameptr :=
    namesleft #1 - 'namesleft :=
  }
 while$
}
```

In the outer `if` the `then` branch does not modify the stack depth but the `else` branch push the string `t` on the stack ☹, only during the last iteration. . . ☺

Unbalanced stack

- Correction:
  - Theoretical answer: just build an intelligent compiler to understand the program ☺
  - Approximation: abstract interpretation + loop peeling + partial evaluation
  - Real life right now: pattern matching
- Some other usages of stack nasty things: simulating exception-line mechanism with markers on the stack. A BibTEX function can throw away an exception that is caught elsewhere by emptying the stack up to the marker

- Allow dynamic arbitrary code extension in BibTEX++: new styles, new bibliography sources, new format,. . .
  - *hook* added in BibTEX++ to modify the behaviour
  - Classes specialization
- `\bibliography{:plugin:ENSTBr/computer-science-base}` could fetch the bibliography of our computer science lab with whatever protocol
  - `\cite{ENSTBr:keryell88}`
- `\bibliography{:plugin:citeseer}`
  - `\cite{citeseer:keryell93activity}` would fetch from

    `http://citeseer.nj.nec.com/keryell93activity.html`
  - CiteSeer gives many BibTEX entries but not often very canonical ones. . .

- Need a wrapper to clean up the HTML

- `\bibliography{:plugin:DBLP}` to fetch the bibliography from `http://dblp.uni-trier.de/`
  - XML base `ftp://ftp.informatik.uni-trier.de/pub/users/Ley/bib/records.tar.gz`
  - DTD `http://SunSITE.Informatik.RWTH-Aachen.DE/dblp/db/about/dblp.dtd`
- `\bibliography{:plugin:fermivista}`
- `\bibliographystyle{:plugin:ENSTBr/computer-science-style}` fetches the `ENSTBr/computer-science-style` Java style
- ...

Meta-plugins

- Plugins that fetch other plugins from plugin servers or generate new ones
- Heavily rely on Java class dynamic loading and introspection
- `\bibliography{:plugin:ENSTBr/metaplugin:http://...}`
- Directly use a style for a journal from the journal editor server

Need to extend BibTEX++ with *plugins*, meta-*plugins*, from everywhere...

¿¿¿What about BibTEX++ virus??? ☹

- Word™: VBScript macros
- HTML browser
  - JavaScript and buggy execution
  - Applet : Java may escape a buggy sand-box
  - All the plugins (*flash*,...) in the browser
  - Automatically opened in some mailers (OutLook, Eudora,...)
- TEX with a execution *shell* on `\write18` if allowed
- `dvips \special{‘...}` if no `-R`
- PostScript : a true computer language *and* operating system ⇝

arbitrary code execution if not in secured mode

- PDF
  - JavaScript
  - Various plugins
  - Can launch a viewer on `http://...` links

⇝ ¡Need to finely control the execution!...

Securing BibTEX++ mobilets

- Written in Java ⇝ freely and heavily relies on Java security model
- All actions of a class can be precisely authorized by a `SecurityManager` object : file access, network access,. . .
- Specialization of a `SecurityManager` for a kind of BibTEX++ styles or plugins
- Mainly only authorized to fill the bibliography cache

(Wo)Man power on the project

- 1 first year programming project (PAP) in 2000 with 4 students: project basis
- 1 first year programming project (PAP) in 2001 with 5 students: concept of plugin, preview of stack removal
- 1 third year bibliography study in 2001 on stack removal
- 1 master internship on sorting in 2002
- 1 master thesis running on getting all this stuff to production (6 months)

What's next

- Streamline the installation phase
- Plugin mechanism to be implemented
- Object specialization framework. But what/how?
  - Subclassing
  - Aspect programming
  - Reflection & introspection
  - ...
- Code transformation framework for automatic localization and more
- *Back to typography*: think again about bibliography: how to deal with complete mix up of Latin, Arabic, Chinese,... different entries in the *same* bibliography?

What's next

- Deal with other worlds than LaTeX: DocBook,. . .
- Compile BST for other targets : Bibulus,. . .

Conclusion

- There is even computer science research work in the BibTEX world!
- Compatible with BibTEX right now, tested on Linux & Windows
- Ready for scalability:
  - Clean portable object oriented language
  - Native UNICODE
  - Recycle legacy dusty deck BST and BIB files through advanced compiler technologies
  - Free software
- Can reach the great unification: for example
  - Word™ document
  - XML bibliography database from the Internet

- ► `.bst` BibTEX for a journal from the Internet
- Tested against all the teTEX distribution. . . Some `.bst` files in the distribution are wrong! ☹
- First packaged public version this summer
- CVS available at `http://picolibre.enst-bretagne.fr`

Table des transparents

Introduction

1 BibTEX is good for you...

3 . . . but. . .

5 ...BibTEX++

6 BibTEX++ basic architecture

Architecture

7 Adopt the object attitude

8 Programming language

9 BibTEX++ core

10 Bibliography back-end

11 Front-ends

12 A BST compiler

BST compiler

17 Stack removal

18 Typing

19 Type reconstruction

20 Unbalanced stack

23 Plugins

Plugins

25 Meta-plugins

26 Security and mobile code (*mobilet*)

Security

28 Securing BibTEX++ mobilets

29 (Wo)Man power on the project

30 What's next

32 Conclusion

Conclusion

34 Table of contents