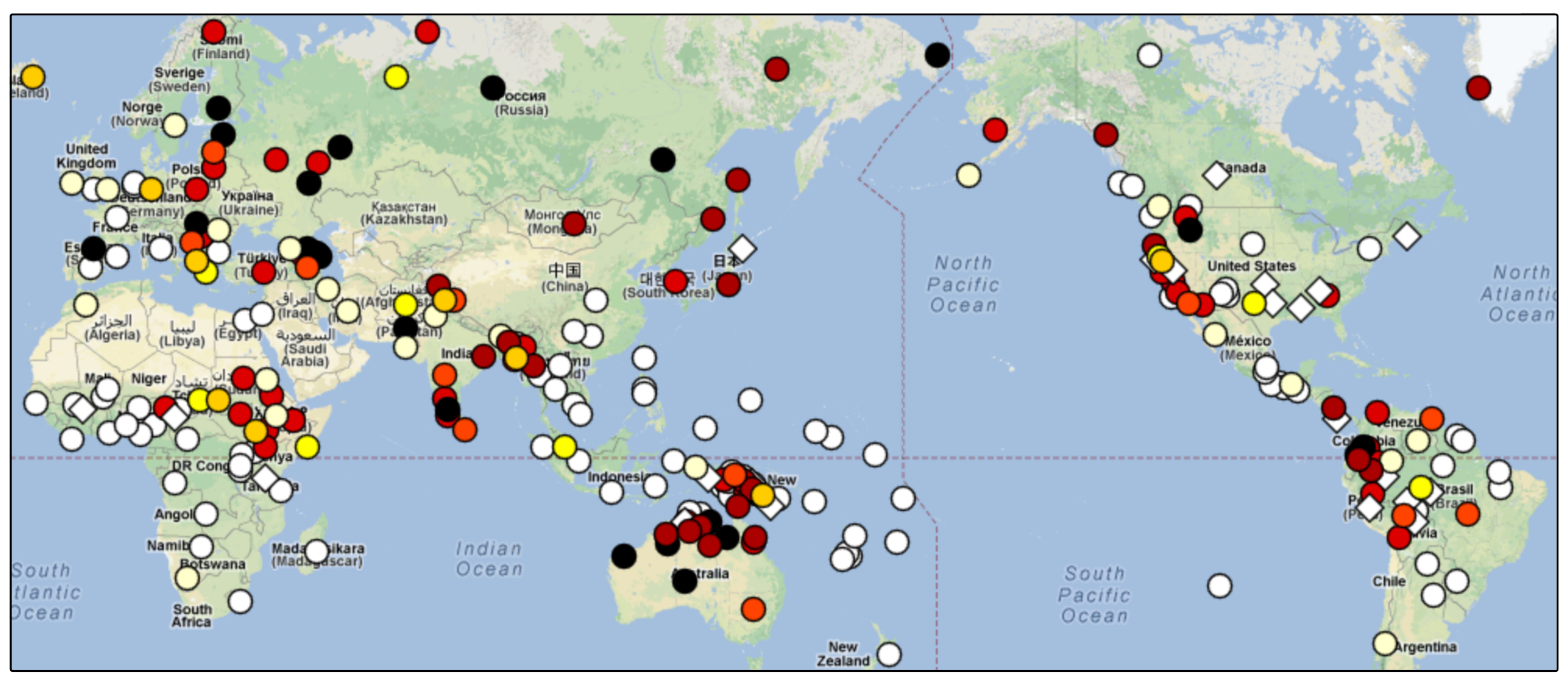
Building an Open-Source Development Infrastructure for Language Technology Projects

Sjur Moshagen°, Tommi Pirinen*, Trond Trosterud‡; Divvun° (*divvun.no*), Giellatekno‡ (*giellatekno.uit.no*), University of Tromsø°‡, University of Helsinki*

How to make language technology available?

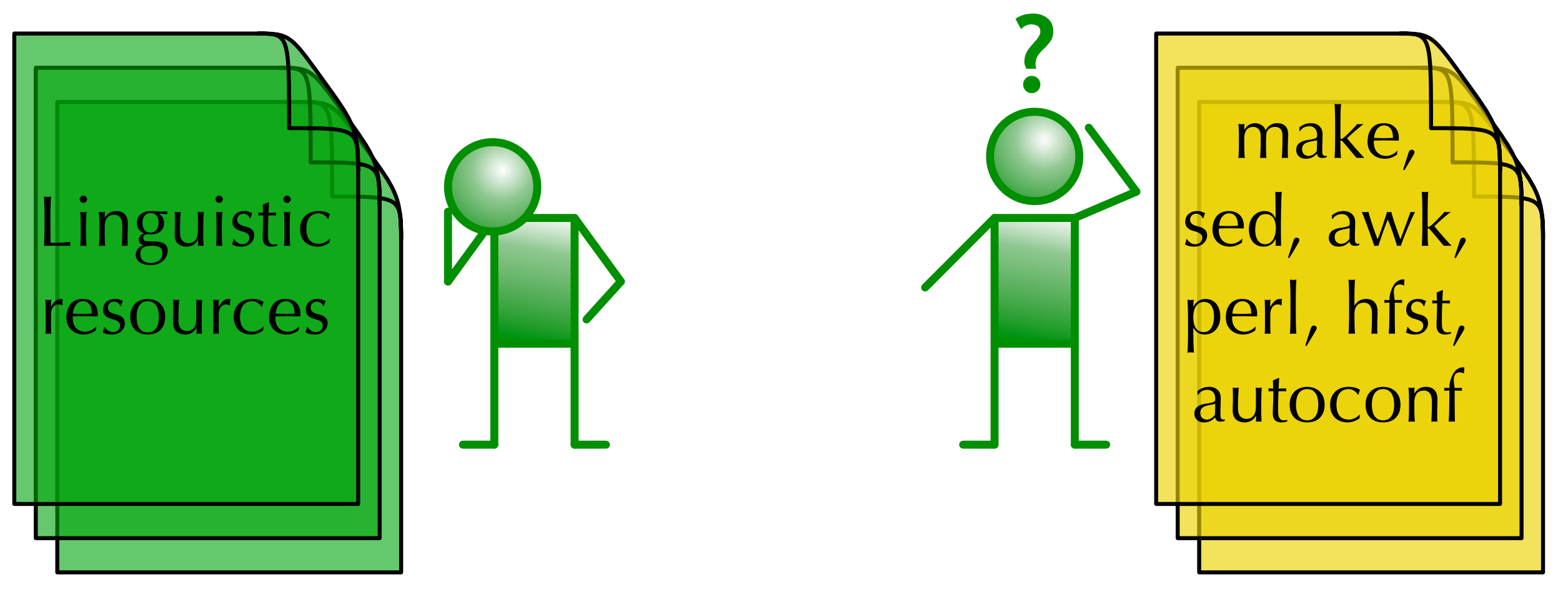
1. 7000 languages, 2200 with standard orthography
2. 150 languages with basic localisation support
3. a handful of languages with good language tech



Most languages have rich morphology and a paucity of texts Cf. map showing number of cases, ranging from white = 0 to black = 10 or more. The white spots in Africa and America represent languages with rich verbal inflection (and in Africa gender classes), whereas white spots in the Pacific represent languages with less morphology (Source: wals.info, map 49A).

Our solution:

- open infrastructure
- open source tools and language source files
- rule-based tools
- separation of concerns:
  - linguistic data and resources *vs.*
  - infrastructure and required tools



⇒ ***plug in a new language and get the tools for free***

Language technology for the languages of the world!

Source code and documentation:

Source: `svn co https://victorio.uit.no/langtech/trunk`
Documentation: http://divvun.no/doc/infra/GettingStarted.html

The infrastructure consists of 5 components

1. A core component of common scripts, filters, ...
2. A set of templates for adding new languages
3. One file set for each individual language
4. Autotools to configure builds and support portability
5. Tools to compile the individual components

Technical details:

Bring best common practices from software engineering to computational linguistics:

1. Code re-use by separation of common parts and language specifics: a) build rules are common for all languages; b) lexical data, linguistic rules specific for each language; c) sharing data is possible, via a scripted template system that can merge or replace data
2. Proper use of GNU standard Autotools[1] suite for easy building, distribution and deployment: Standard build by `` `./configure && make install` ``
3. Standard form obligatory commenting of code for automatic documentation and testing (cf. literate programming, python's doccomments[2] and doctest[3], etc.)
4. Maintainability is achieved through standard coding style, automatic testing, and open source style project maintenance: Automatic test suite by `` `make check` ``
5. Distribution and deployment is achieved by autotools standards: a) Installation by `` `make install` `` (with `` `DESTDIR` `` support etc.); b) Automatically distributable language packages by `` `make dist` ``

- Morphologically complex languages require state-of-the-art tools to handle morphology and syntax, e.g. Xerox fst/hfst tools, Constraint Grammar tools, others
- Maintainability with different tools and syntaxes requires rigorous following of coding style, commenting and making examples for test cases
- Autotools setup ties together different tool sets with completely different modes of operation by neatly built simple rules and extensive test phase
- Code commenting practices ensure code suites are up-to-date with rules

URLS:
[1]: http://en.wikipedia.org/wiki/GNU_build_system
[2]: http://www.python.org/dev/peps/pep-0257/
[3]: http://docs.python.org/3.3/library/doctest.html

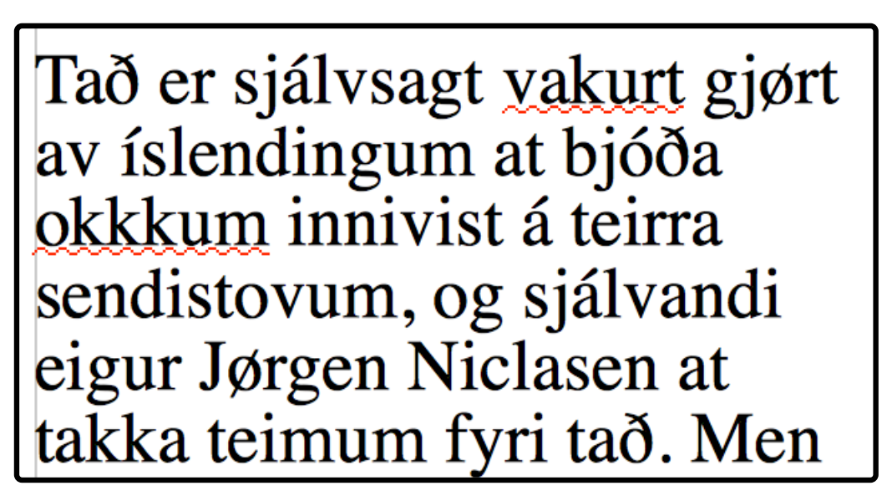
Make your own speller (try it here!)

```
1)edit your source file(s),
2)then:
   $ make
   $ sudo make install
3)open LibreOffice, and see your changes in the speller!
```

The tools we produce:

- morphological analysers/generators, parsers
- spell checkers and hyphenators
- electronic dictionaries with morphology
- speech synthesis (new, not yet finished)
- ICALL tools with linguistic analysis

Example spellchecker: Faroese. The first error is a false alarm, the second one is not.



Tað er sjálvsagt vakurt gjørt av íslendingum at bjóða okkkum innivist á teirra sendistovum, og sjálvandi eigur Jørgen Niclasen at takka teimum fyri tað. Men

How to build a speller:

1. collect word list from corpus
2. build a lexicon & morphology
3. make an error model
4. choose speller type:
   – Microsoft Office
   – Linux/Unix (aspell, hunspell, etc)
5. build a speller engine
6. integrate into applications
7. test the coverage
8. test the error model
9. test the speller engine
10. test the application integration

The red-coloured areas are automated in our infrastructure, partly as a direct result of the way it is built, and partly by leveraging development work done by other groups, mainly Voikko[4] and Hfst[5]. What remains is purely linguistic work.

The linguistic nature of the work is further emphasised by our focus on reuse and morphologic (and syntactic) analysis – the lexicon and morphology are directly usable for both morphological analysis and generation. This property of the infrastructure and the tools we build are further utilized in the automated testing, to ensure that all and only the expected word forms are accepted or generated.

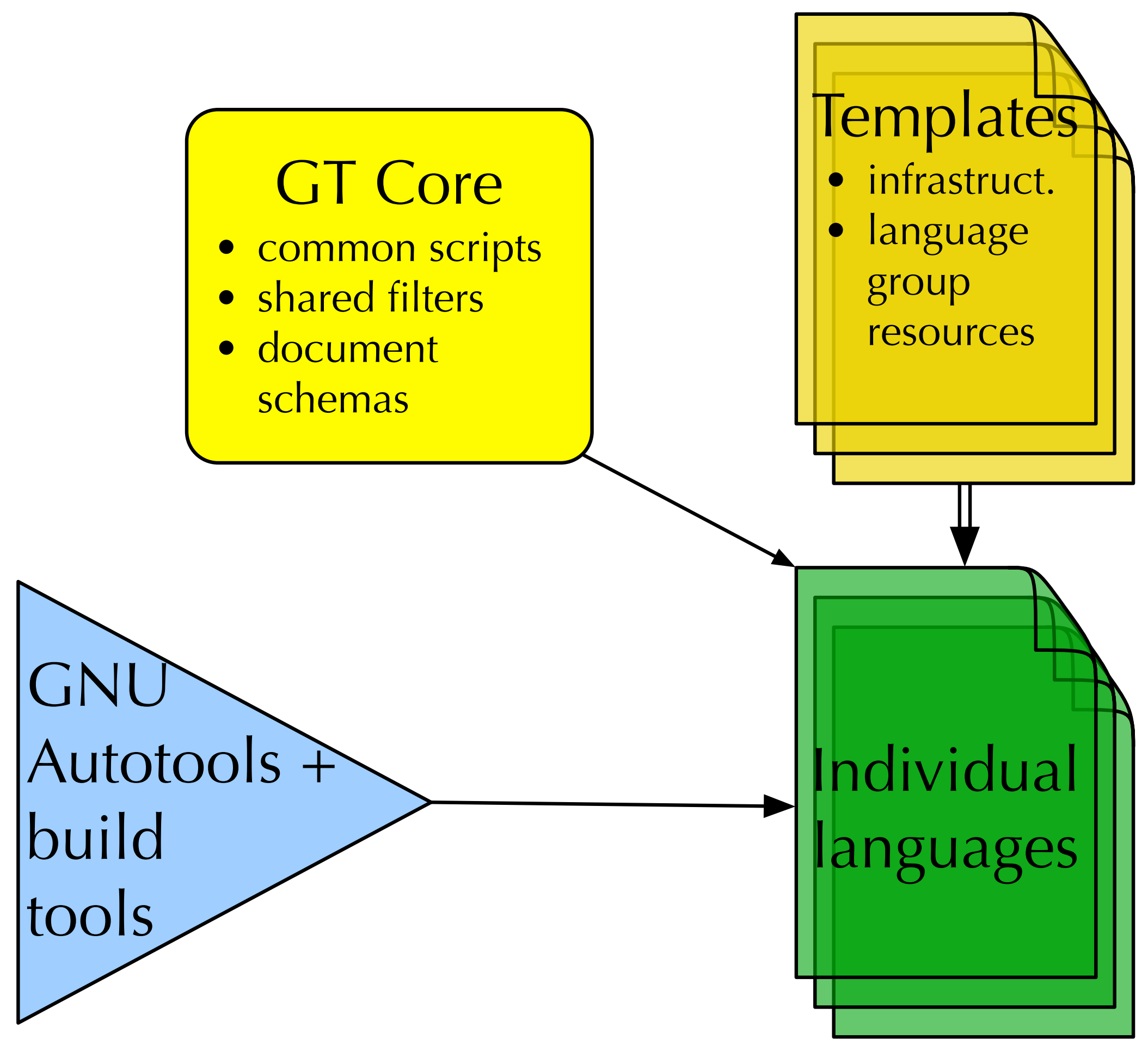
URL's:
[4]: http://voikko.sourceforge.net
[5]: http://hfst.sourceforge.net

Units Language	Roots	Affixes	States	Edges	On-Disk Size	Coverage	HFST	Xerox
Lule Saami	73,101	2,943	103,504	260,401	7 MiB	94.1 %	2 min 44 s	30.9 s
Finnish	71,761	55,895	179,891	403,944	9.6 MiB	85.1 %	18.7 s	16.4 s
Erzya	106,550	2,860	65,571	174,427	5.8 MiB	84.4 %	26.7 s	2.1 s
Komi-Zyrian	88,734	1,336	56,146	147,618	4.4 MiB	83.4 %	59.3 s	4.6 s
Faroese	87,567	1,460	80,969	1,282,798	32 MiB	81.5 %	6 min 25 s	3 min 55 s
Greenlandic	72,243	147,717	398,622	13,018,901	170 MiB	94.2 %	1 h 11 min	37 min 12 s
Eastern Mari	56,684	180	35,139	87,578	2.8 MiB	72.6 %	1 min 21 s	0.9 s
Western Mari	53,236	180	28,734	70,890	2.2 MiB	69.8 %	36.7 s	2.7 s
Udmurt	37,101	84	35,986	65,296	1.8 MiB	43.3 %	29.3 s	0.9 s

Some of our transducers, their size and performance. *Coverage* tests against Wikipedia or similar corpora (~100000 wds), and *HFST* and *Xerox* show compilation time. The variation is dependent upon coding style: Finnish is made with parallel affixes instead of morphophonological rules. Greenlandic encodes suffix strings as separate "words", to make a spellchecker, this gives a large FST. For the other FSTs affix size reflects morphological complexity and coverage.

Portable language technology

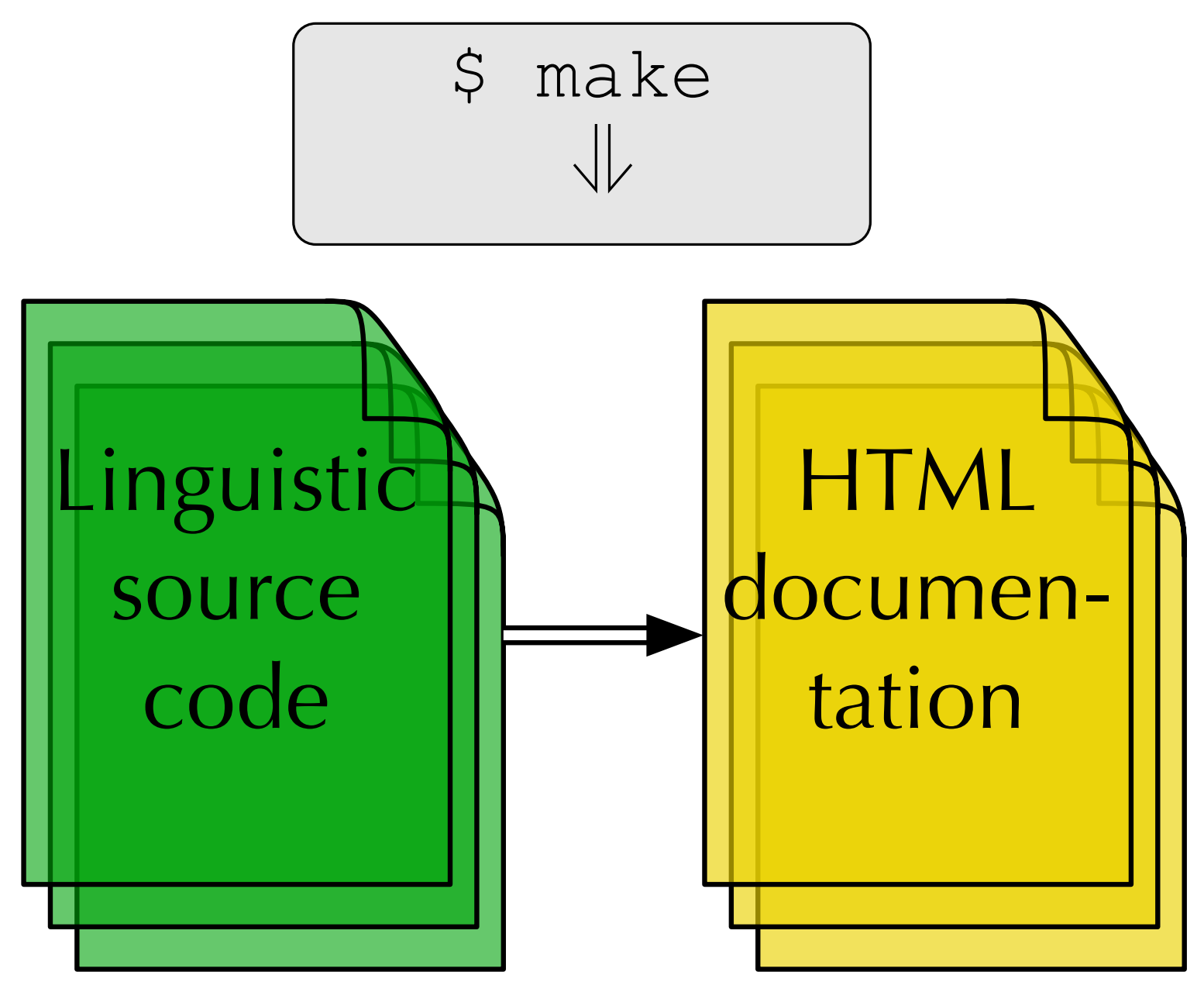
The Giellatekno/Divvun (GT) infrastructure:



Automated testing and documentation extraction

We use the support for automated testing and documentation building built into Autotools, to automatically run a set of tests and generate documentation for ourselves and for others.

```
$ make check
      ⇓
PASS
FAIL
======================
4 of 7 tests failed
(2 tests were not run)
Please report to bugs@
======================
```


$ make
⇓
Linguistic source code
HTML documen-tation