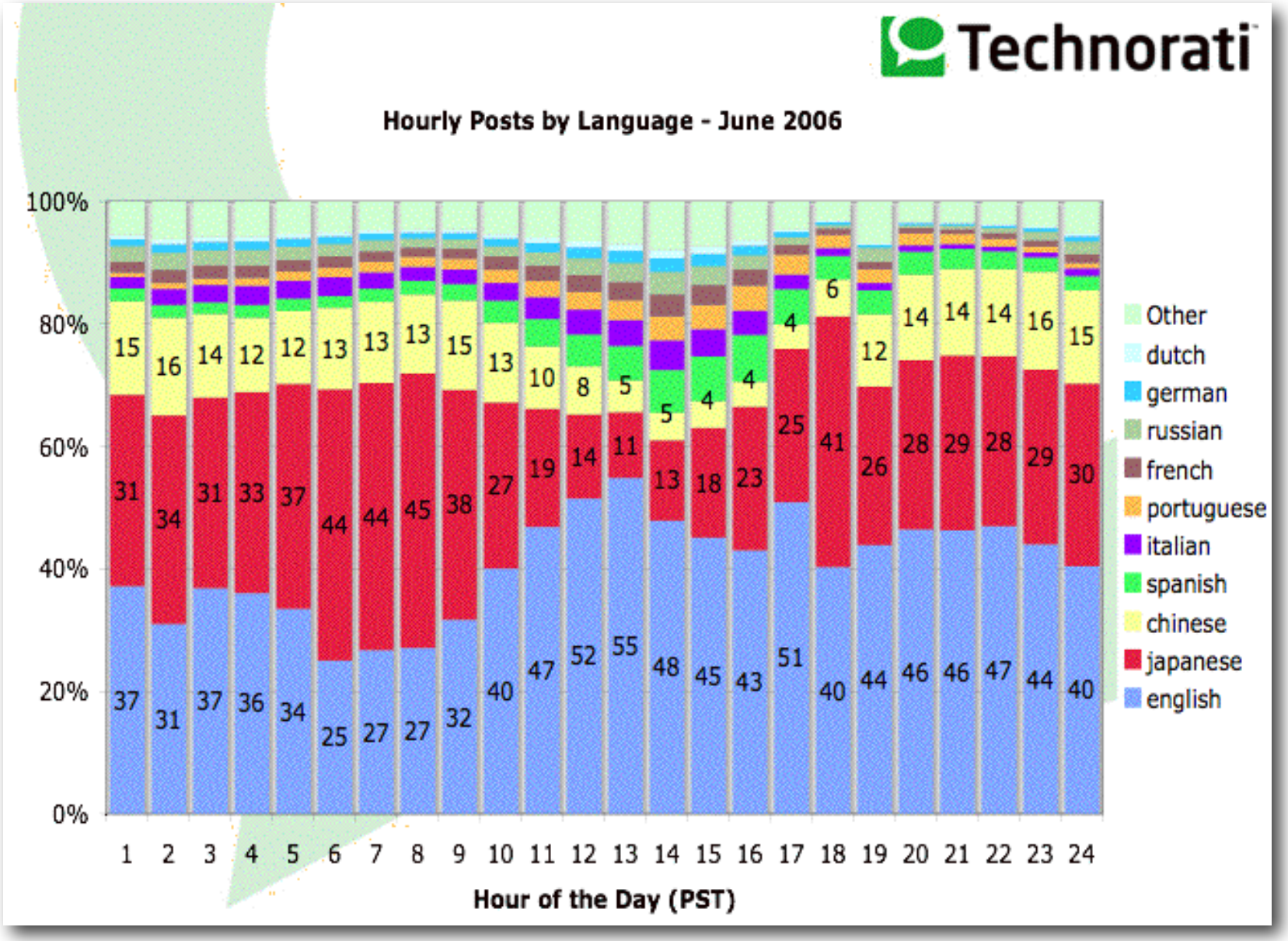
I18N, M17N, UNICODE, AND ALL THAT

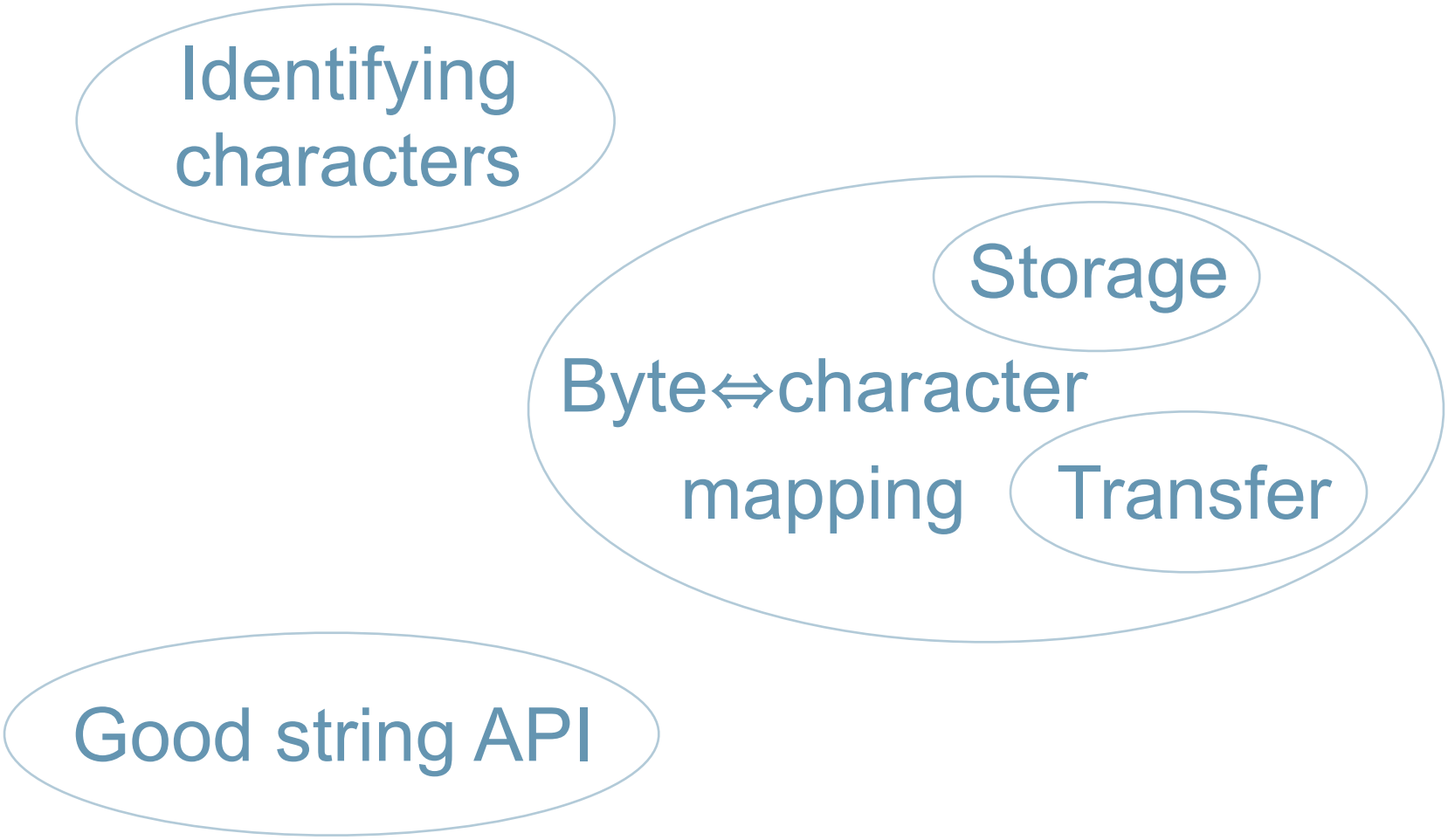
Tim Bray
General-Purpose Web Geek
Sun Microsystems

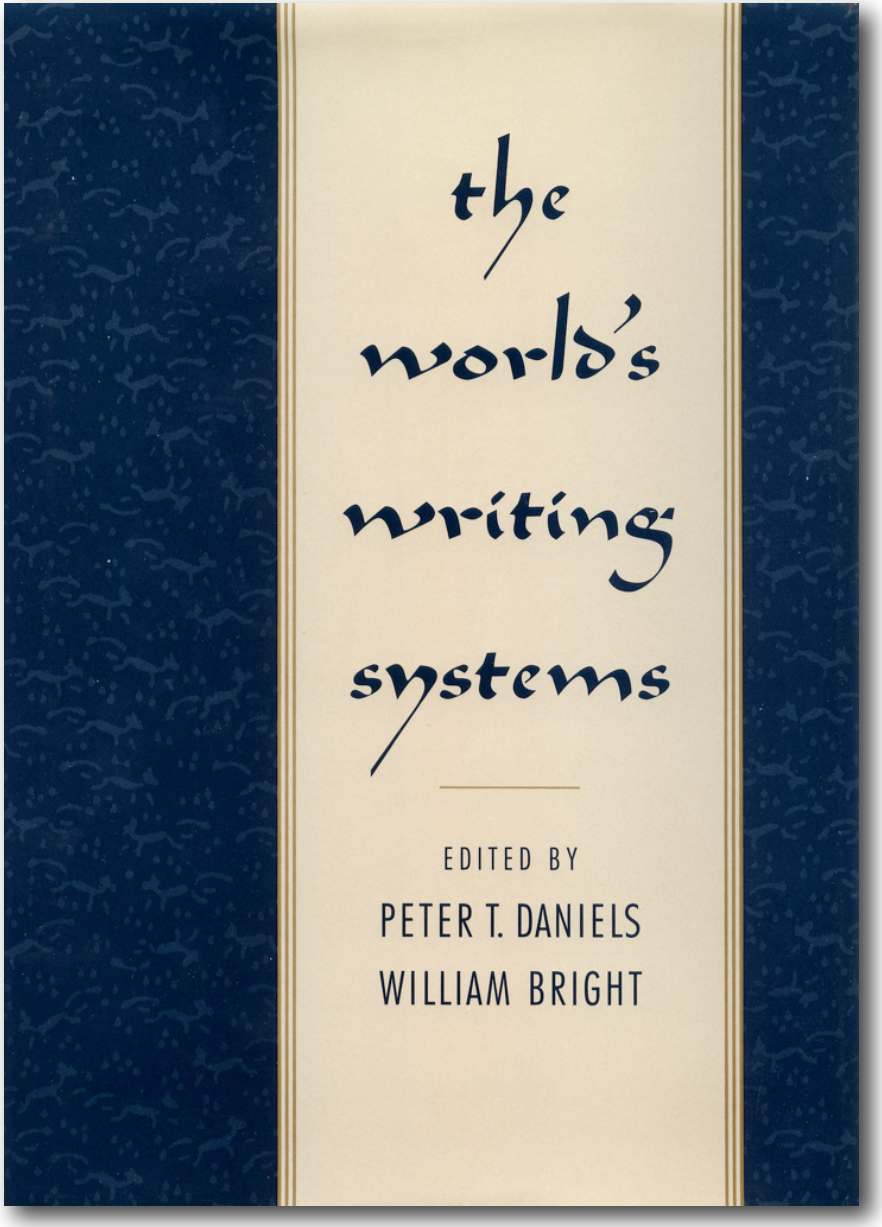

Technorati
Hourly Posts by Language - June 2006
100%
80%
60%
40%
20%
0%
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Hour of the Day (PST)
Other
dutch
german
russian
french
portuguese
italian
spanish
chinese
japanese
english
15 16 14 12 12 13 13 13 15 13 10 8 5 5 4 4 4 25 41 6 12 14 14 14 16 15
31 34 31 33 37 44 44 45 38 27 19 14 11 13 18 23 25 41 26 28 29 28 29 30
37 31 37 36 34 25 27 27 32 40 47 52 55 48 45 43 51 40 44 46 46 47 44 40

```
/[a-zA-Z]+/
```

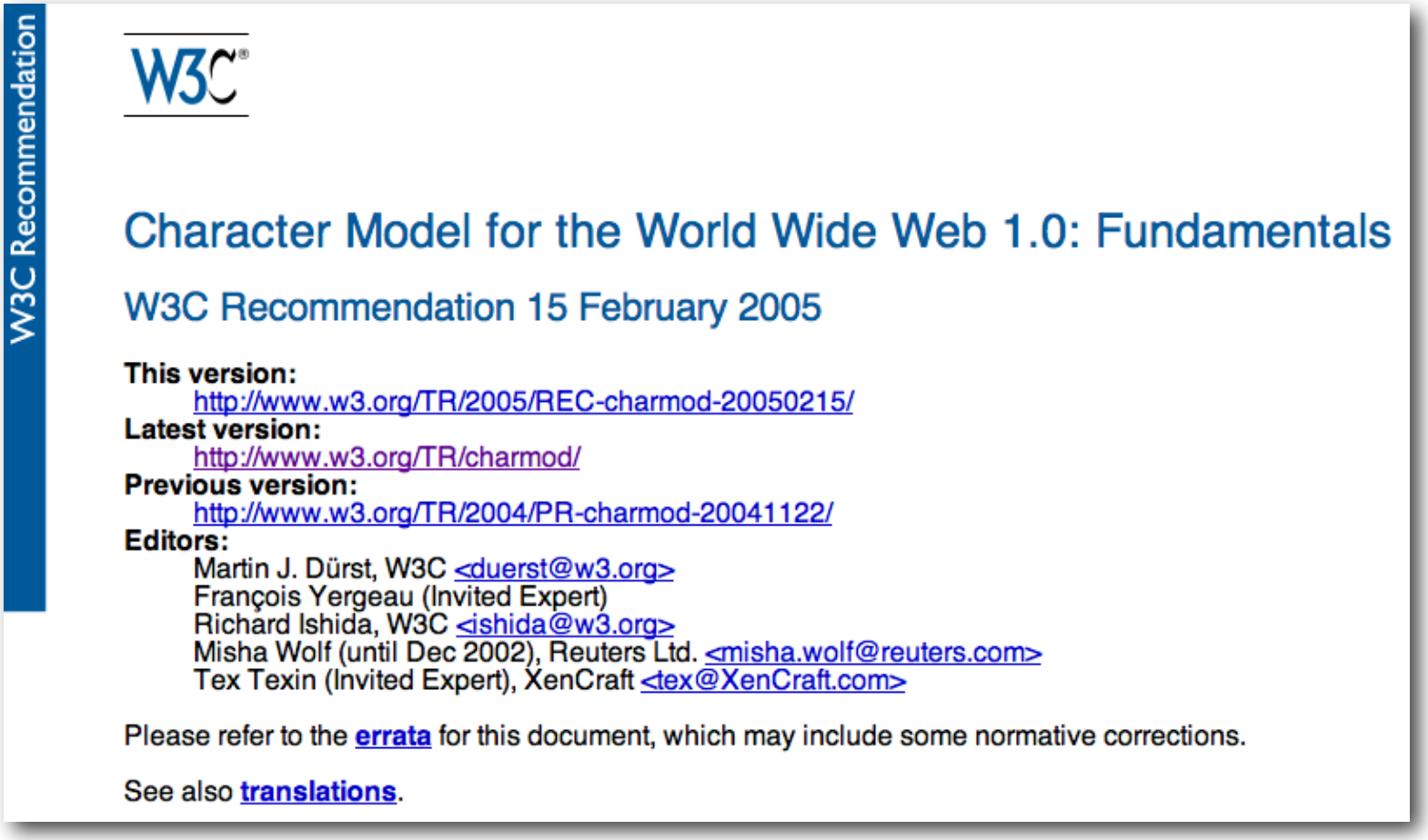
This is probably a bug.

The Problems We Have To Solve



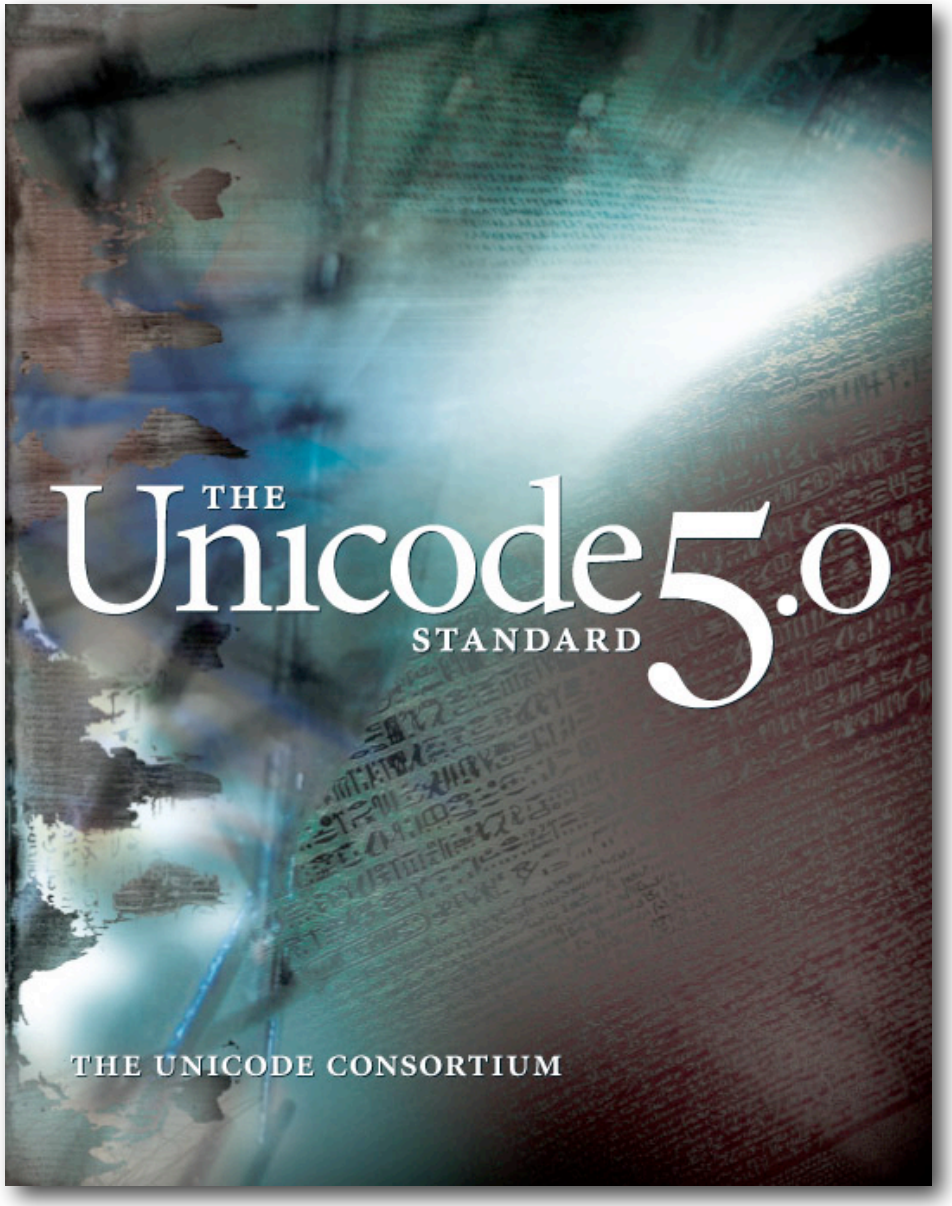


Published in 1996; it has 74 major sections, most of which discuss whole families of writing systems.



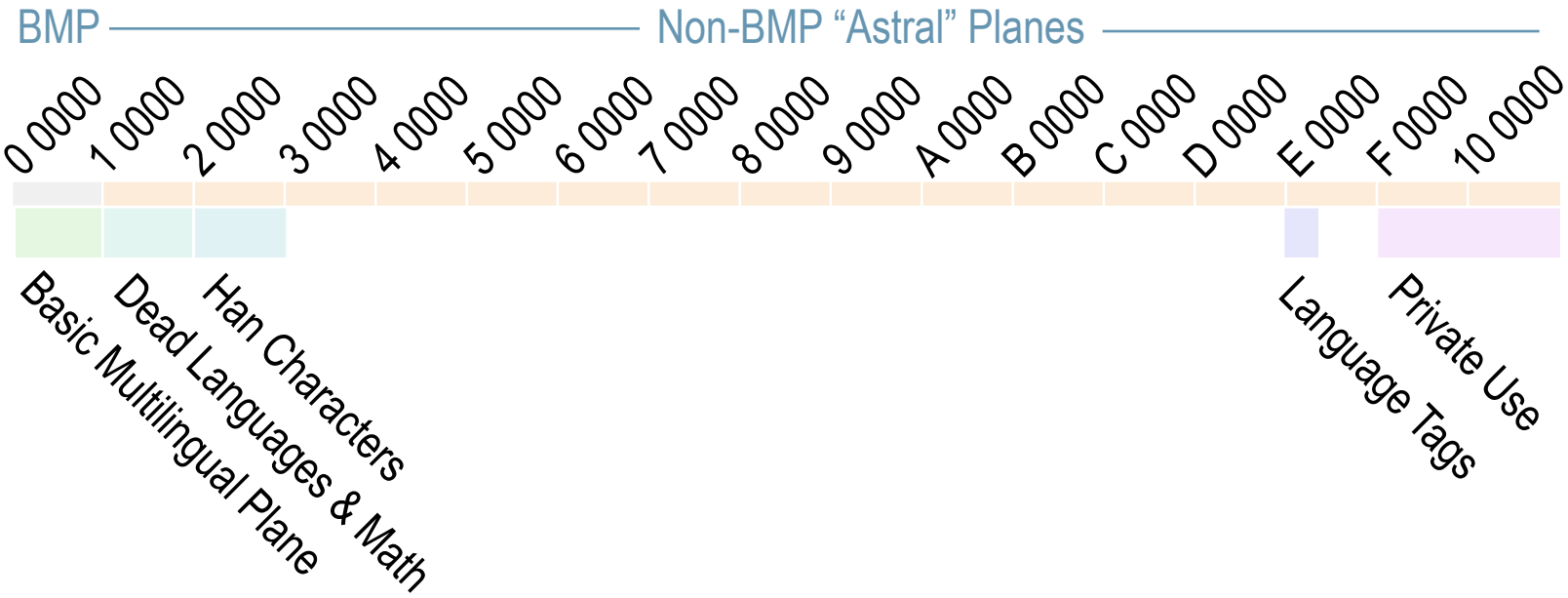
www.w3.org/TR/charmod

Identifying Characters



1,114,112 Unicode Code Points

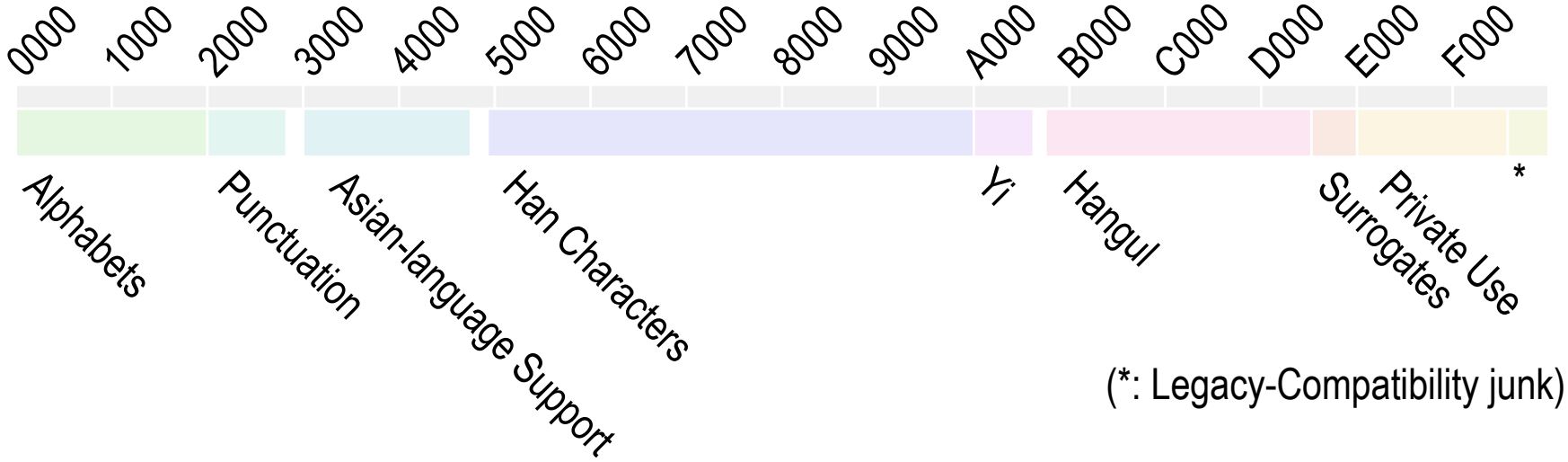
17 “Planes” each with 64k code points: U+0000 – U+10FFFF



99,024 characters defined in Unicode 5.0

The Basic Multilingual Plane (BMP)

U+0000 – U+FFFF



(*: Legacy-Compatibility junk)

Unicode Character Database

```
00C8;LATIN CAPITAL LETTER E WITH
GRAVE;Lu;0;L;0045 0300;;;;N;LATIN
CAPITAL LETTER E GRAVE;;;00E8;
```

"Character #200 is LATIN CAPITAL LETTER E WITH GRAVE, a lower-case letter, combining class 0, renders L-to-R, can be composed by U+0045/U+0300, had a different name in Unicode 1, isn't a number, lowercase is U+00E8."

www.unicode.org/Public/Unidata

U+0024 DOLLAR SIGN

U+017D LATIN CAPITAL LETTER Z WITH CARON

U+00AE REGISTERED SIGN

U+03AE GREEK SMALL LETTER ETA WITH TONOS

U+0416 CYRILLIC CAPITAL LETTER ZHE

U+05D0 HEBREW LETTER ALEF

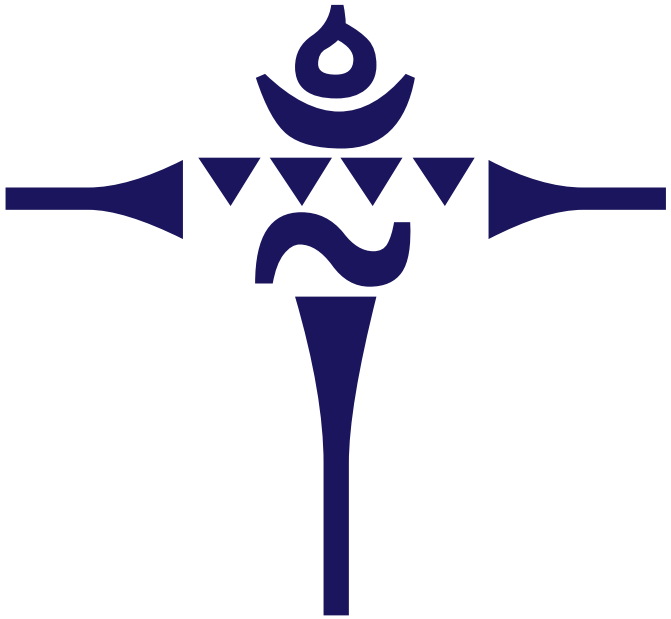
U+0638 ARABIC LETTER ZAH

ਗਾ

U+0A17 GURMUKHI LETTER GA

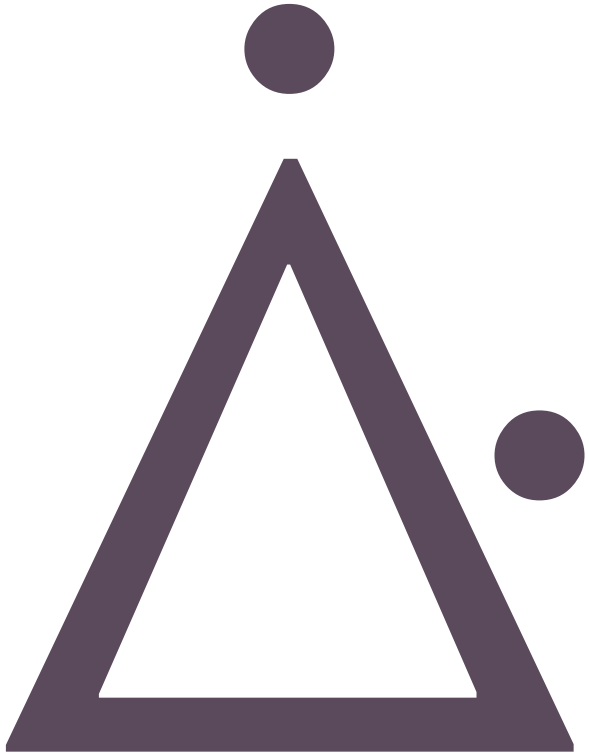
U+0A88 GUJARATI LETTER II

U+0E06 THAI CHARACTER KHO RAKHANG

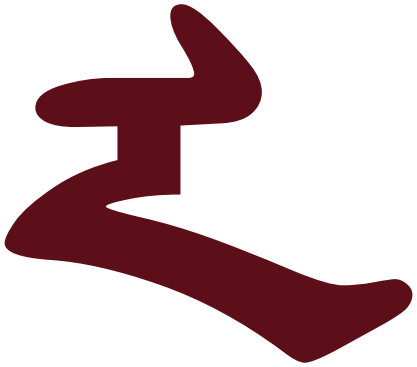


U+0F12 TIBETAN MARK RGYA GRAM SHAD

U+13BA CHEROKEE LETTER ME



U+1411 CANADIAN SYLLABICS WEST-CREE WII

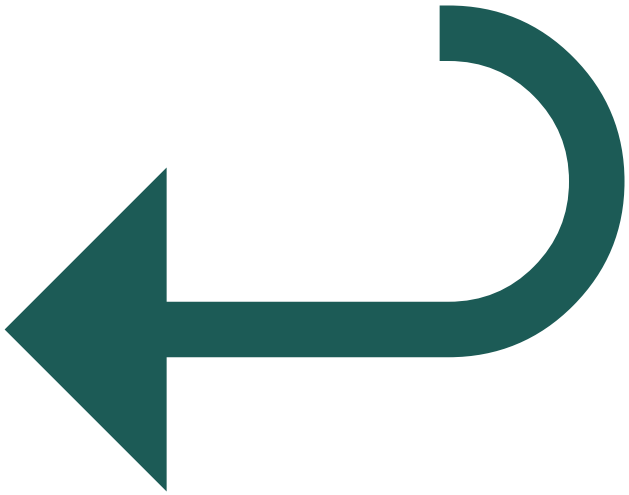


U+1820 MONGOLIAN LETTER ANG

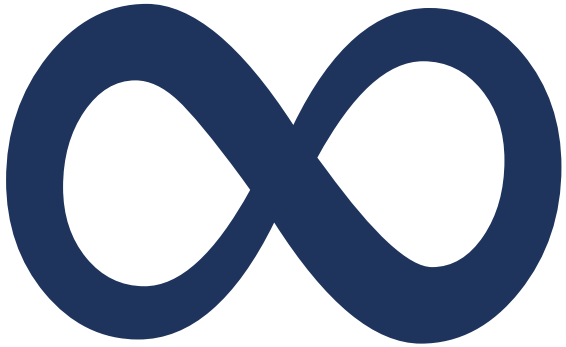
‰

U+2030 PER MILLE SIGN

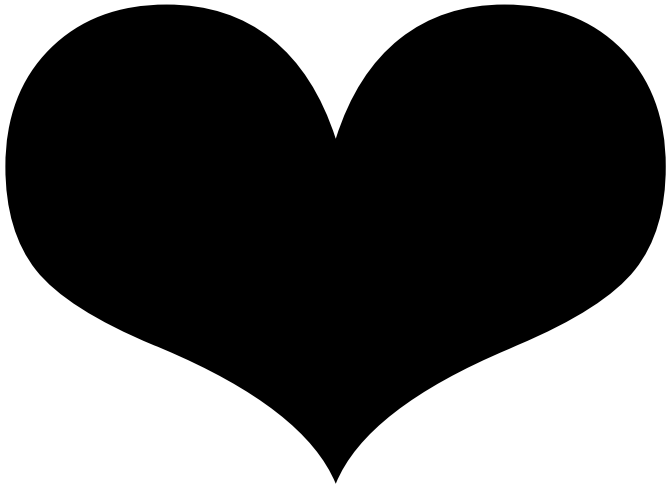
U+215D VULGAR FRACTION FIVE EIGHTHS



U+21A9 LEFTWARDS ARROW WITH HOOK



U+221E INFINITY



U+2764 HEAVY BLACK HEART

U+3055 HIRAGANA LETTER SA

U+30C0 KATAKANA LETTER DA

U+4E2D (Han character)

U+8A9E (Han character)

U+AC7A (Hangul syllabic)

U+1D12B (Non-BMP) Musical Symbol Double Flat

U+2004E (Non-BMP) (Han character)

Nice Things About Unicode

Huge repertoire
Room for growth
Private use areas
Sane process
Unicode character database
Ubiquitous standards/tools support

Difficulties With Unicode

Combining forms
Awkward historical compromises
Han unification

Han Unification

Pro: en.wikipedia.org/wiki/Han_Unification

Contra: tronweb.super-nova.co.jp/characcodehist.html

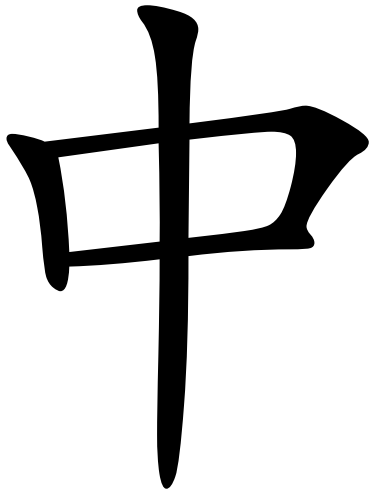
Neutral: www.jbrowse.com/text/unij.html

Alternatives

For Japanese scholarly/historical work: Mojikyo, www.mojikyo.org; also see Tron, GTCode.

Also see Wittern, *Embedding Glyph Identifiers in XML Documents*.

Byte⇔Character Mapping



U+4E2D (Han character)
How do I encode 0x4E2D in bytes for computer processing?

Storing Unicode in Bytes

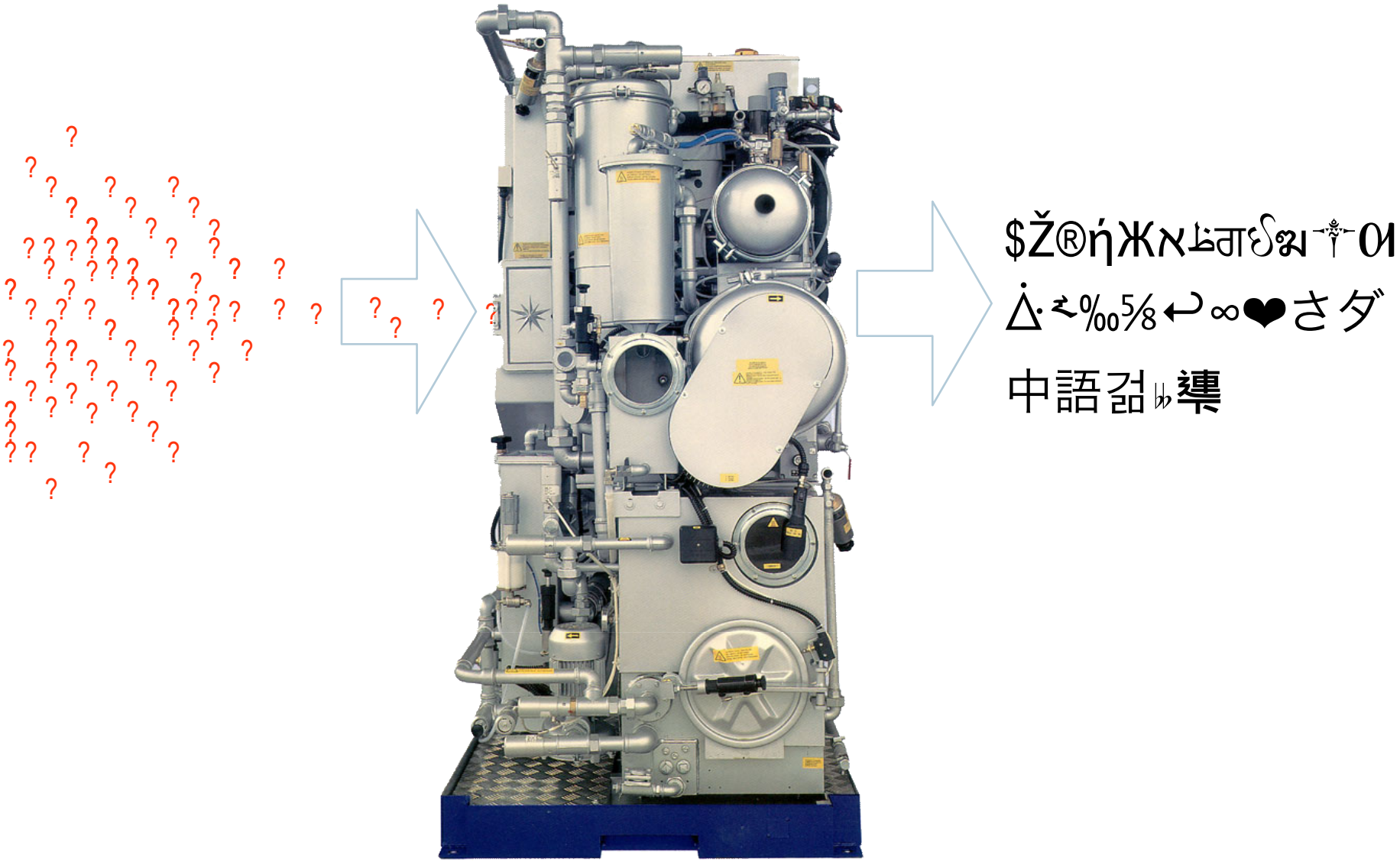
Official encodings: UTF-8, UTF-16, UTF-32
Practical encodings: ASCII, EBCDIC, Shift-JIS, Big5, GB18030, EUC-JP, EUC-KR, ISCII, KOI8, Microsoft code pages, ISO-8859-*, and others.

UTF-* Trade-offs

UTF-8: Most compact for Western languages, C-friendly, non-BMP processing is transparent.
UTF-16: Most compact for Eastern languages, Java/C#-friendly, C-unfriendly, non-BMP processing is horrible.
UTF-32: *wchar_t*, semi-C-friendly, 4 bytes/char.
Note: Video is 100MB/minute...

Web search: “characters vs. bytes”

Text Arriving Over the Network



"An XML document knows what encoding it's in."

- Larry Wall

What Java Does

Strings are Unicode. A Java “char” is actually a UTF-16 code point, so non-BMP handling is shaky. Strings and byte buffers are separate; there are no unsigned bytes. The implementation is generally solid and fast. The APIs are a bit clumsy and there’s no special regexp syntax.

What Perl Does

Perl 5 has Unicode support, in theory. In a typical real-world application, with a Web interface and files and a database, it is very difficult to round-trip Unicode without damage. However, regexp support is excellent. Perl 6 is supposed to fix all the problems...

What Python 3000 Will Do

String Types Reform (Guido's Slide)

- bytes and str instead of str and unicode
  - bytes is a mutable array of int (in range(256))
  - encode/decode API? bytes(s, "Latin-1")?
  - bytes have some str-ish methods (e.g. b1.find(b2))
  - but not others (e.g. not b.upper())
- All data is either binary or text
  - all text data is represented as Unicode
  - conversions happen at I/O time
- Different APIs for binary and text streams
  - how to establish file encoding? (Platform decides)

What Ruby Does

```
% * + << <=> == =~ [] []= capitalize
capitalize! casecmp center chomp chomp!
chop chop! concat count crypt delete
delete! downcase downcase! dump each
each_byte each_line empty? eql? gsub
gsub! hash hex include? index
initialize_copy insert inspect intern
length ljust lstrip lstrip! match new
next next! oct replace reverse reverse!
rindex rjust rstrip rstrip! scan size
slice slice! split squeeze squeeze! strip
strip! sub sub! succ succ! sum swapcase
swapcase! to_f to_i to_s to_str to_sym tr
tr! tr_s tr_s! unpack upcase upcase! upto
```

Core Methods With I18n Issues

```
== =~ [] []= eql? gsub gsub! index length
lstrip lstrip! match rindex rstrip
rstrip! scan size slice slice! strip
strip! sub sub! tr tr!
```

Missing String Method

`each_char`

Needs to be correct and efficient; should serve as the basis for many other methods. Should “just know” about encoding issues.

Alternatively, change String#each

1. Allow regexp as well as String argument.
2. Change the default to /./mu from "\n".
3. include Enumerable.

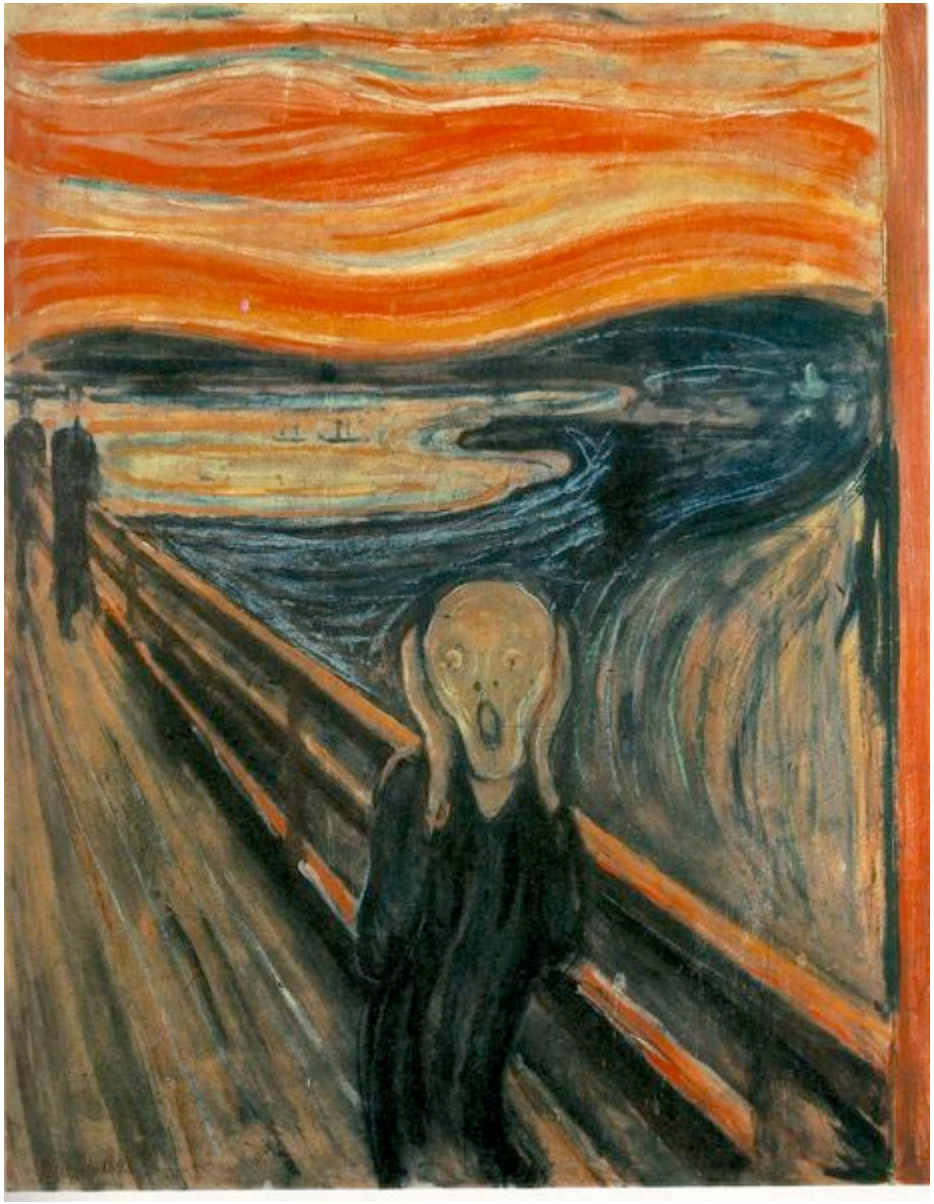
On Byte-buffers and Strings

[] for addressing bytes is OK, because characters are normally read in sequence.

```
def substr(start, len)
  index = -start
  s = ''
  each_char do |c|
    break if index == len
    s << c unless index < 0
    index += 1
  end
  s
end
def charAt(index) substr(index, 1); end
```

On Case-folding

Lower-case ‘I’: ‘i’ or ‘ı’?
Upper-case ‘i’: ‘I’ or ‘İ’?
Upper-case ‘ß’?
Upper-case ‘é’?
Just Say No!



Dangerous String Methods

```
capitalize capitalize! casecmp downcase
downcase! swapcase swapcase! upcase
upcase!
```

Avoid case-folding hell.

Advanced String Methods

```
[] each_byte unpack
```

99.99999% of the time, programmers want to deal with characters not bytes. I know of one exception: running a state machine on UTF8-encoded text. This is done by the Expat XML parser.

Regexp and Unicode

```
stag = "<[^/]([^>]*[^/>])?>"
etag = "</[^>]*>"
empty = "<[^>]*/>"

alnum = '\p{L}|\p{N}|' +
   '[\x{4e00}-\x{9fa5}]|' +
   '\x{3007}|[\x{3021}-\x{3029}]'
wordChars =
   '\p{L}|\p{N}|' + "[-._:']|" +
   '\x{2019}|[\x{4e00}-\x{9fa5}]|\x{3007}|' +
   '[\x{3021}-\x{3029}]'

word = "((#{alnum})((#{wordChars})*(#{alnum}))?)"
text = "(#{stag})|(#{etag})|(#{empty})|#{word}"
regex = /#{text}/
```

Oniguruma can't do these

e.g. "won't-go"

Referring to Characters

```
if in_euro_area?
  append 0x20ac # Euro
elsif in_japan?
  append 0xa5   # Yen
else
  append '$'
end
```

Common idiom while writing XML.

Question: Does Ruby need a Character class?

What Should Ruby Do?

In 2006, programmers around the world expect that, in modern languages, strings are Unicode and string APIs provide Unicode semantics correctly & efficiently, by default. Otherwise, they perceive this as an offense against their language and their culture. Humanities-computing academics often need to work outside Unicode. Few others do.

Who's Working on the Problem?

Matz: M17n for Ruby 2
Julik: ActiveSupport::MultiByte (in edge Rails)
Nikolai: Character encodings project (rubyforge.org/projects/char-encodings/)
JRuby guys: Ruby on a Unicode platform

Thank You!

Tim.Bray@sun.com
www.tbray.org/ongoing/
this talk: www.tbray.org/talks/rubyconf2006.pdf