Pango

An open-source Unicode
text layout engine

Owen Taylor
`otaylor@redhat.com`

25th Internationalization and Unicode Conference
Washington, DC
March/April 2004

Outline

Open source

- Source code made available to user, who can modify, redistribute
- No longer unfamiliar
  - Linux, Apache
  - Libraries such as libpng, libjpeg, libxml, ICU...
- High Quality
  - Development process incorporates code review: small group of core developers take contributions from broad user community
- Responsive to needs of users

Open source and text layout

- Ability to contribute back code very interesting for a layout engine
  - Most users have relatively simple needs
  - But “minor” scripts still have thousands or millions of users
  - Some of these users will be developers interested in contributing back code

The idea of Pango

- General purpose layout library; not restricted to complex scripts
- Highly modular
  - Add new scripts
  - Add new backends
- Use throughout system; adding support for a language to Pango enables it everywhere
  - Config tools, dialog boxes, spreadsheets, web browsers, ...

The Name

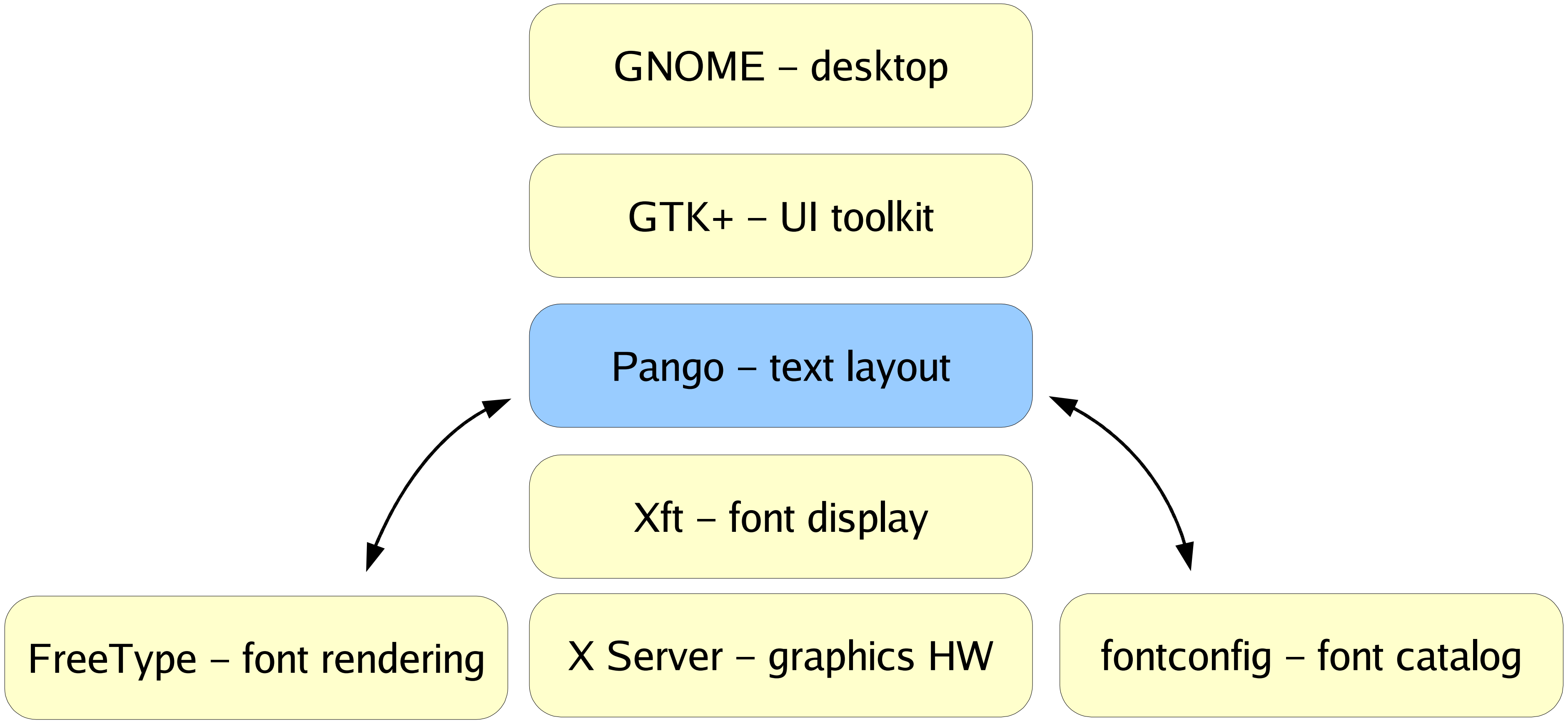
Παν語

- Greek “pan” - all + Japanese “go” language
- “go” character means “spoken language” in Chinese...

License

- GNU *Library* General Public License (LGPL)
- If an application just links to Pango
  - No requirement to reveal source code
  - No royalties
- If you modify Pango, must make the source code available to your users

Big Picture on X



- Also used on Win32, in embedded systems, etc.

Timeline

- 1999 – work started
- 2001 – 1.0 release; used in version 2.0 of GTK+ user interface toolkit.
- 2002 – 1.2 released; Indic OpenType fonts, fontconfig, Uniscribe backend on Win32
- 2004 – 1.4 release; Unicode-4.0 support, GPOS positioning for Arabic

Basic idea

- Input
  - Unicode text
  - Attributes (font family, language tags, colors,etc.)
- Output
  - Positioned *glyphs*
- Positioned glyphs
  - rendered to the screen, printer
  - converted to outlines for a drawing program
  - ...

An example

```
PangoContext *context;
PangoLayout *layout;
int width, height;

context = pango_xft_get_context (display, screen);
layout = pango_layout_new (context);
pango_layout_set_text (layout, "Hello, world");
/* ... or ... */
pango_layout_set_markup (layout,
     <span size='x-large'>Big</span> text”);

pango_layout_get_pixel_size (layout, &width, &height);
pango_xft_layout_render (xft_draw, xft_color,
                         layout, 10, 10);
```

PangoLayout

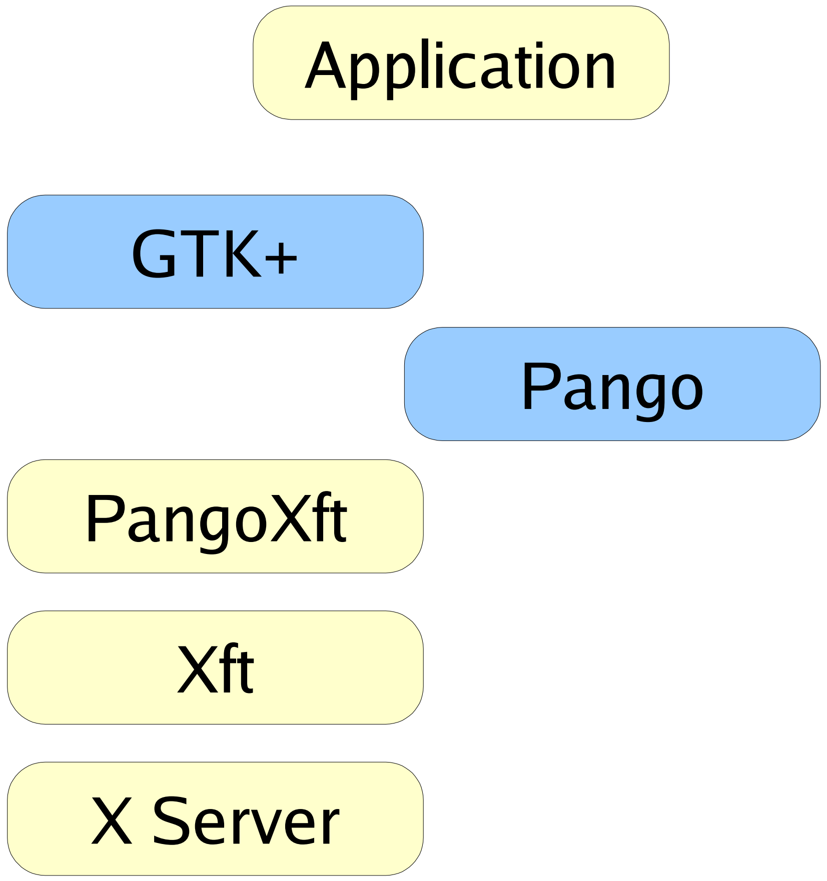
- High-level PangoLayout object holds one or more paragraphs of text + attributes
- Interfaces provided for:
  - Hit testing
  - Determining cursor locations
  - Iterating through text in visual or logical order
  - etc.

Backends

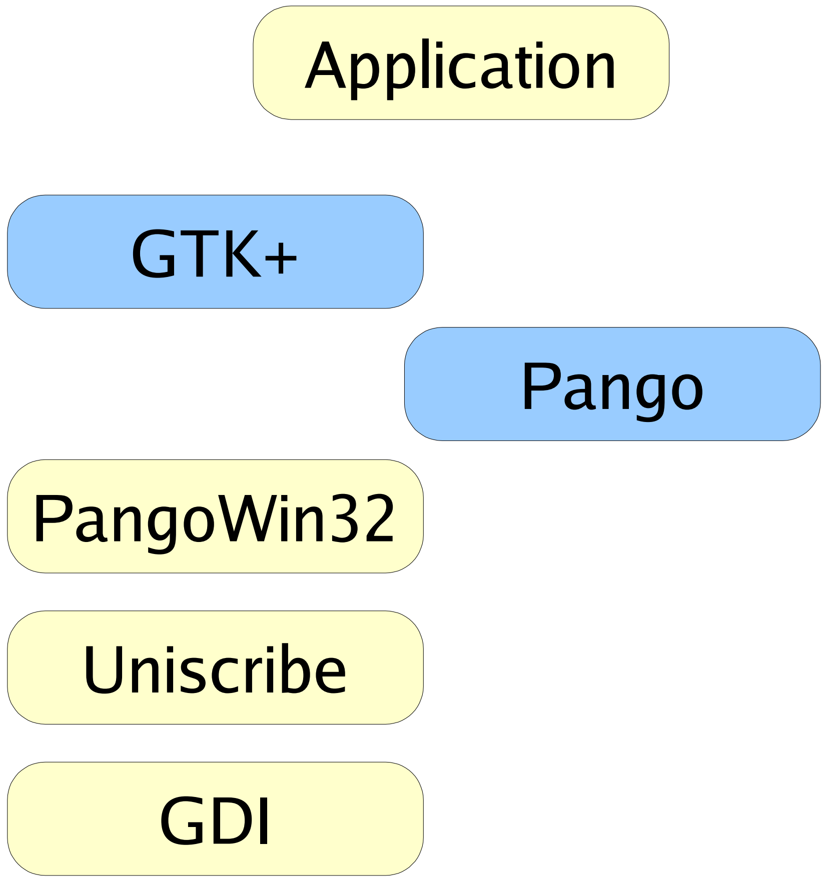
- Pango doesn't shield user from the backend
- A larger system (e.g., GTK+) can

```
PangoContext *context;
PangoLayout *layout;
int width, height;

context = gtk_widget_get_pango_context (widget);
layout = pango_layout_new (context);
pango_layout_set_text (layout, "Hello, world");

pango_layout_get_pixel_size (layout, &width, &height);
gdk_draw_layout (widget->window, widget->style->black_gc,
                 10, 10, layout);
```

Backends (cont.)

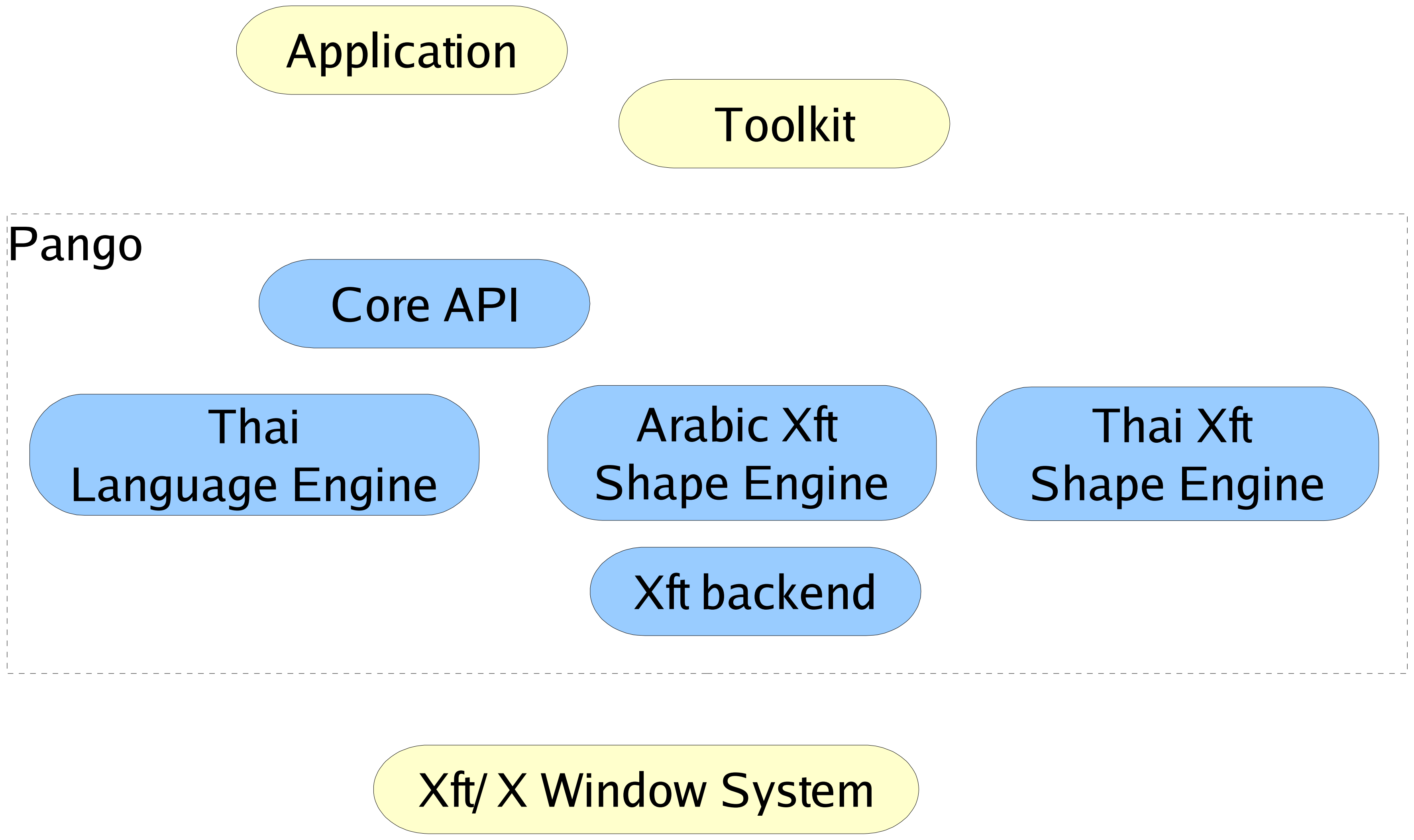


On Linux/Unix



On Win32

Internal architecture



Pango pieces

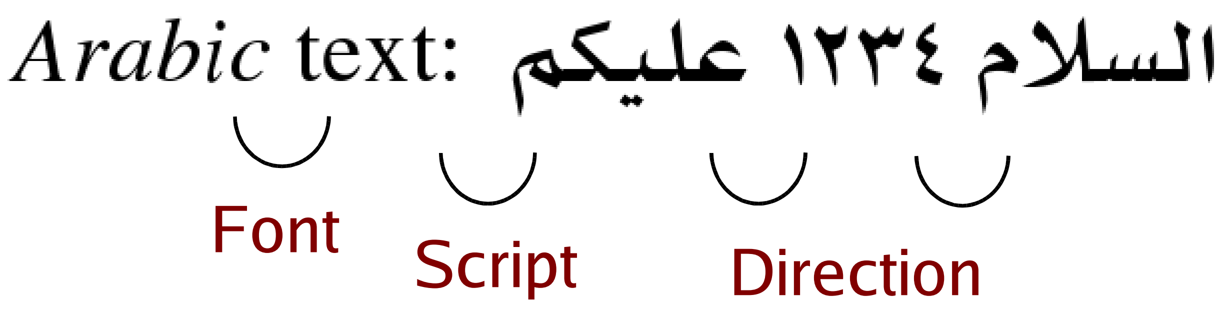
- Core: PangoLayout, layout pipeline driver logic
- Language engines: language specific logic for line breaks, etc.
- Backend library: public/private interfaces for a particular backend
- Shape engines: layout logic for a particular backend/script combination

Layout pipeline

- Can go directly to low-level layout process
  - But usually, PangoLayout is more convenient
- Steps are
  - Itemization
  - Text boundary determination
  - Shaping
  - Line breaking
- Note similarity to Uniscribe
  - helps when layering Pango on top of Uniscribe on Win32

Itemization

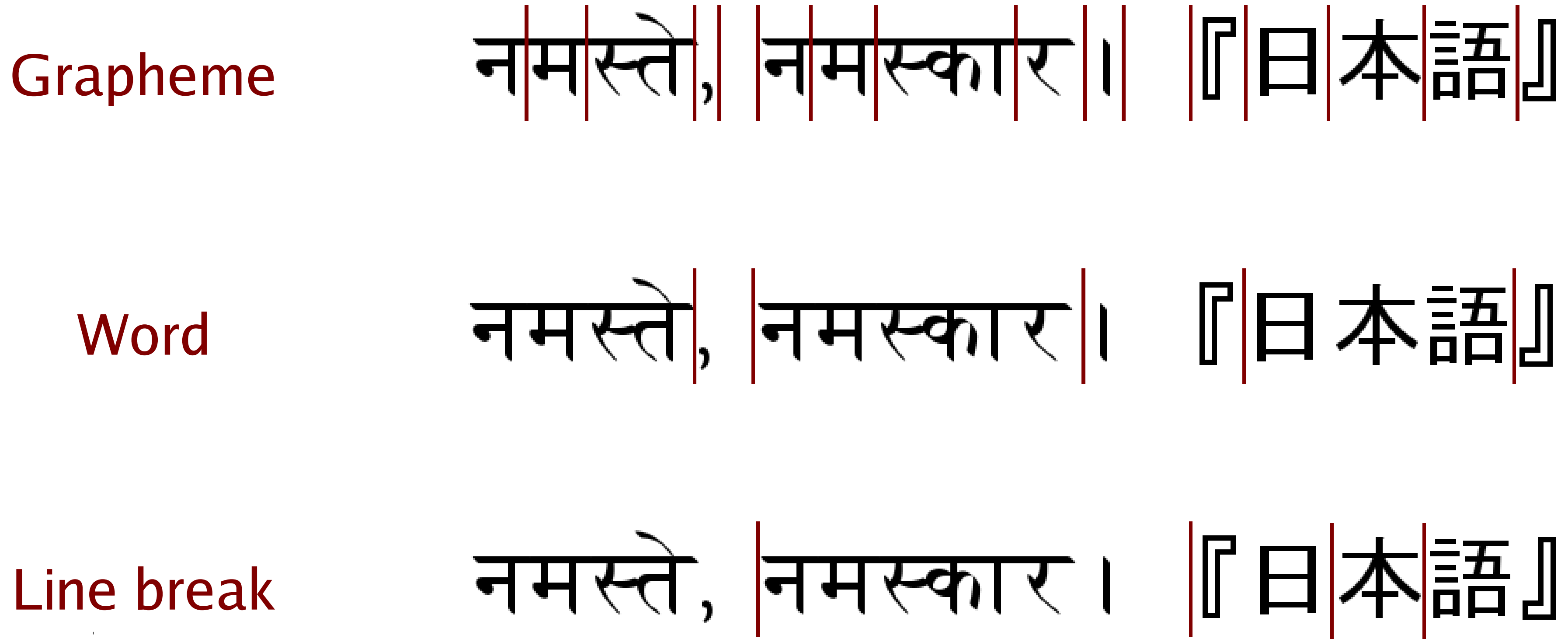
- text broken into segments with unique font, direction, shape engine



Arabic	text:	السلا	١٢٣٤	عليكم

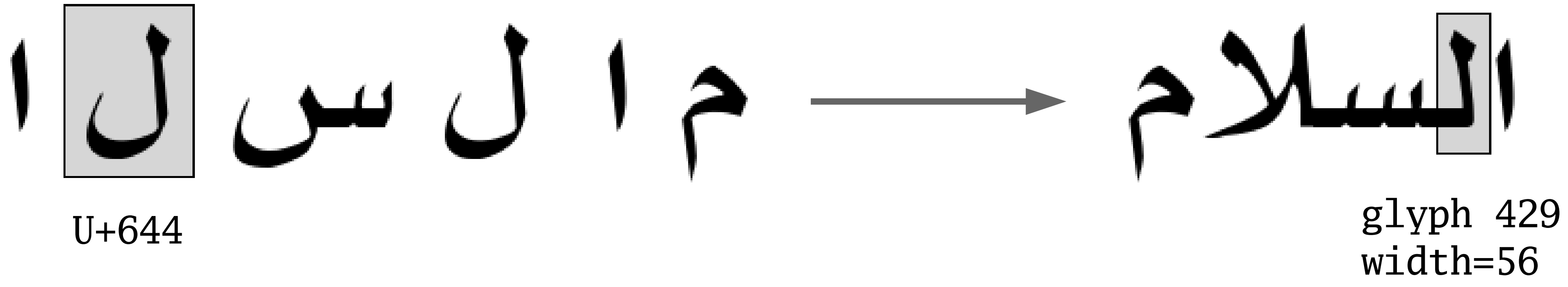
`Nimbus Roman Italic`
`Bidi-level=0`
`Basic shaper`
	`Nimbus Roman`
	`Bidi-level=0`
	`Basic shaper`
		`KacstLetter`
		`Bidi-level=1`
		`Arabic shaper`
			`KacstLetter`
			`Bidi-level=2`
			`Arabic shaper`
				`KacstLetter`
				`Bidi-level=1`
				`Arabic shaper`

Text boundaries


Grapheme
न|म|स्ते|,| |न|म|स्का|र| |।| |『|日|本|語|』
Word
नमस्ते|, |नमस्कार|। 『|日本語|』
Line break
नमस्ते, |नमस्कार। |『日|本|語』

- Line break boundaries affect shaping
- Grapheme, word boundaries for editing

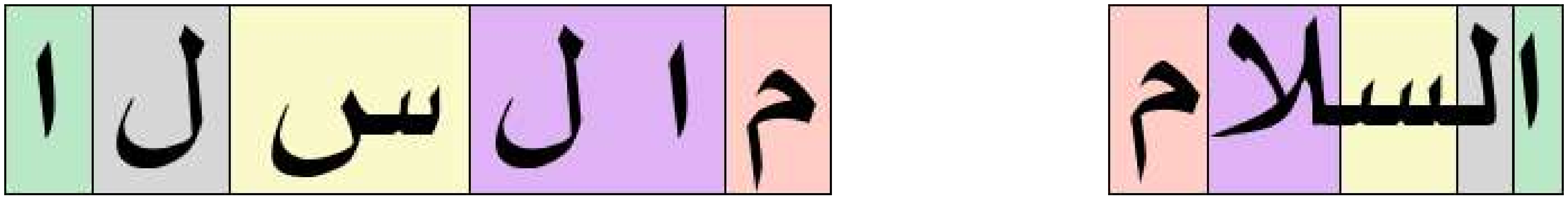
Shaping



- Input: font, Unicode text
- Output: positioned glyphs
- Done by script-specific “shape engine”

Clusters

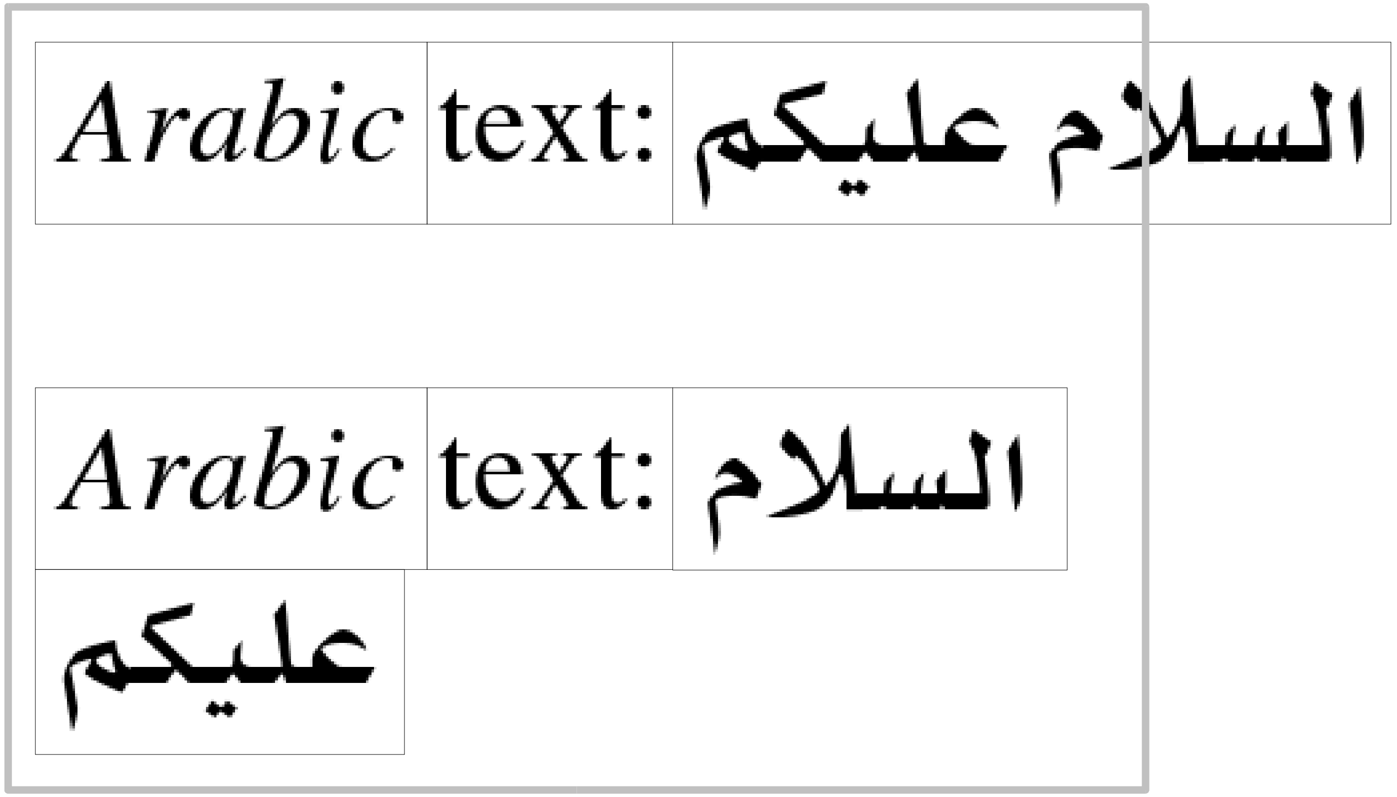
- Each output glyph is assigned to a cluster of input characters
  - needed for hit testing, drawing selections, etc.



- Can be
  - N characters to 1 glyph: ligature
  - 1 character to M glyphs: decomposition
  - N characters to M glyphs (e.g., Indic syllables)

Line Breaking

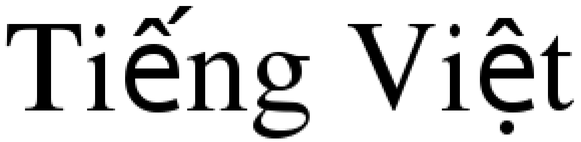
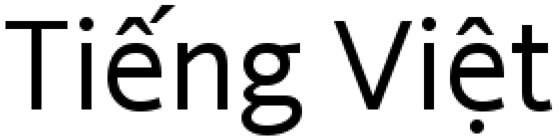
- Measure shaped items
- If necessary, break item at line break position, reshape pieces



Scripts

- Shape engine primarily selected based on script (Cyrillic, Arabic, Devanagari, Latin, Han, ...)
- *Neutral characters* – combining marks, whitespace, zero-width characters – need to be passed to same shaper
  - Performance; don't want lots of little items
  - Correctness; characters such as ZWJ affect
- Identify *script runs*
  - Algorithm borrowed from ICU

Font selection

- Need to resolve multiple-font aliases, like “Serif”
- Using first font in alias for each character gives “ransom-note” typography: 
Tiếng Việt
- Language tags help; prefer fonts in alias that have *all* the characters needed to write the language: 
Tiếng Việt

Underlying technology

- GLib: data structures, portability routines, Unicode algorithms/properties
- GObject: object oriented programming in C
- fribidi: Unicode bidirectional algorithm

Linux/Unix font handling

- FreeType: font loading and
  - Pango also uses code from FreeType project to parse OpenType tables
- fontconfig: font catalog; font naming
- Xft: display fonts with antialiasing
- OpenType Indic code from ICU

Supported scripts

- “Basic” scripts
- Arabic
- Hangul
- Hebrew
- Indic: Bengali, Devanagari, Gurmukhi, Gujarati, Kannada, Malayalam, Oriya, Tamil, Telugu,
- Thai

Current users

- GTK+ toolkit
  - GNOME desktop
  - GIMP, other cross-platform applications
- Core text library for GNOME desktop
  - Red Hat Enterprise Linux, Sun Java Desktop, etc.
- XSLT stylesheets `(http://pangopdf.sourceforge.net)`
- Mozilla web browser

Future directions

- More scripts: Khmer (patches exist), Tibetan, etc.
- SIL Graphite
- Better typography
  - Justification
  - Hyphenation
  - Vertical layout for CJK

Contributors

Abigail Brady, Hans Breuer, Matthias Clasen,
Sivaraj Doddannan, Dov Grobgeld, James Henstridge,
Theppitak Karoonboonyanan, Karl Koehler, Alex Larsson,
Noah Levitt, Tor Lillqvist, Eric Mader, Keith Packard,
Havoc Pennington, Roozbeh Pournader, Changwoo Ryu,
Jungshik Shin, Chookij Vanatham, Qingjiang (Brian) Yuan

... and more than 90 others

More information:

- These slides:

  `http://people.redhat.com/otaylor/iuc25/`

- Pango web page:

  `http://www.pango.org`

Extra Slides

UTF-8

Language tag refinement

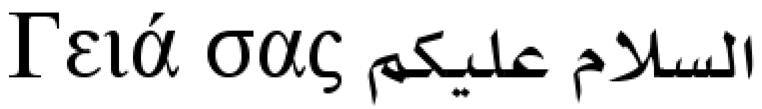
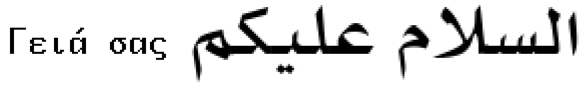
Normalization

Weird mark combinations

UTF-8

- UTF-8 used for both input and output
- Advantages:
  - Compatibility with existing code
  - Seemless beyond BMP handling
  - Emphasizes strings based, not char based methods
- Disadvantages
  - Algorithmic complexity (but not more than UTF-16)
  - Mismatch with UTF-16 based systems

Wrong language tags

- “Serif” alias
  - GreekFont: el,en
  - ArabicFont: ar,en
  - UglyFont: ar,el,en,ru,ab,...
- No language tags: 
  Γειά σας السلام عليكم
- 'ar' language tag: 
  Γειά σας السلام عليكم
  - UglyFont is preferrred to GreekFont for Greek

Language Tag Refinement

- Initial language tags come from document or user's environment
- May not match text: 'ar' language tag applied to Greek script
- In case of mismatch, default language tag for the script is used instead: 'el' for Greek script.

Normalization

- Display should be independent of normalization form
- What doesn't work:
  - Require normalized text as input: hard on application developers
  - Normalize on input: need to preserve mapping
- Ideas
  - Make individual shapers handle it
  - Normalize to NFD before passing to shaper; post-process clusters

Weird mark combinations

- During itemization text split by font
- Base character with mark from a different font?
- Ideas:
  - Forbid: use fallback (dotted circle)
  - Have a “mark font” that is logically merged in with every font. (32 bit glyph indices give spare space)