PDF and OpenType technology

The ideal match or an uneasy compromise?

Create a PDF file with text

- Nothing can be simpler
  - Choose the right font *(Tf)*
  - Set text matrix *(Tm)* or move text cursor *(Td)*
  - Convert Unicode chars to PDF characters via encoding (CIDs)
  - Output *(Tj / TJ)* and go for a coffee

This is what you see when you get back if your text is in Devanagari script:

Below is the correct result.
How many differences can you find between the two?

Devanagari

Tamil: different appearance, same problems

பே·ன்ற

போன்று

Tamil

ITEXT
dualLab

What is OpenType

- File format combining TrueType and Type1 outlines?
- Not just this: support for all kinds of scripts, world languages and their specific features.

Latin

THE SWASHES, LIgatures & Kerning

Kerning

THE SWASHES, LIgatures & Kerning

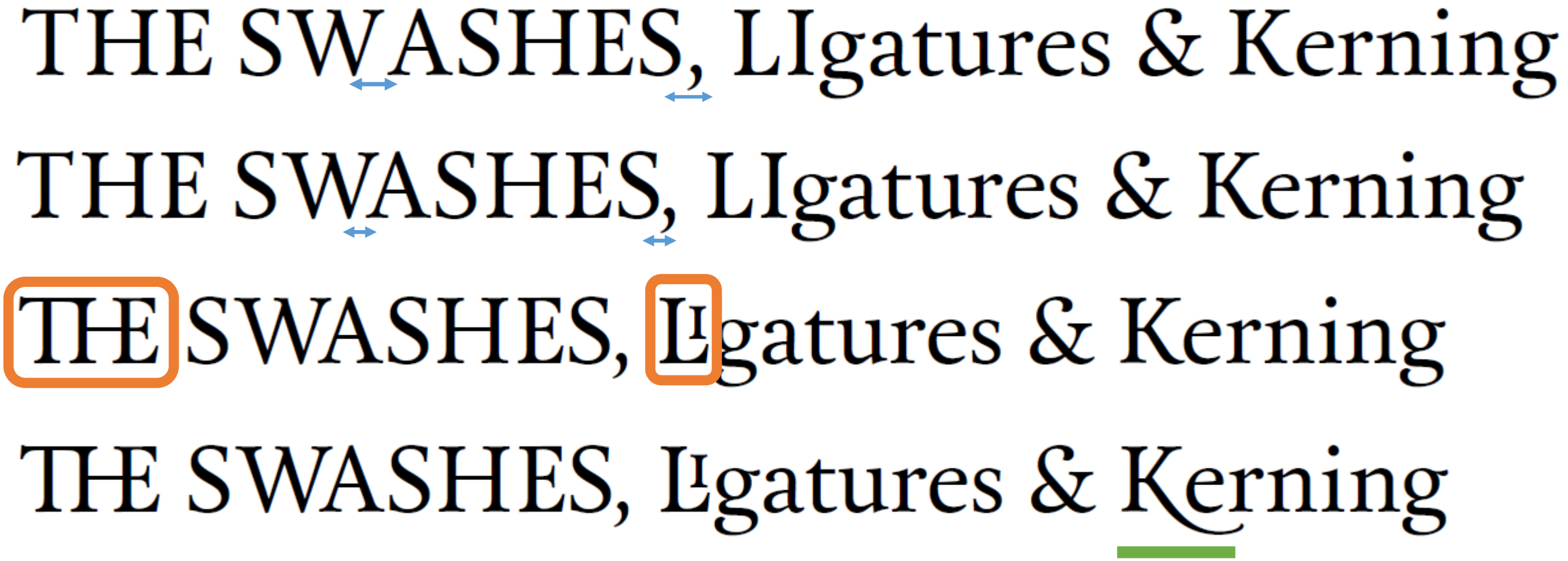
Discretionary ligatures

THE SWASHES, LIgatures & Kerning

THE SWASHES, LIgatures & Kerning

THE SWASHES, LIgatures & Kerning

Swashes


THE SWASHES, LIgatures & Kerning
THE SWASHES, LIgatures & Kerning
THE SWASHES, Ligatures & Kerning
THE SWASHES, Ligatures & Kerning

Stylistic alternatives

THE SWASHES, LIgatures & Kerning

THE SWASHES, LIgatures & Kerning

THE SWASHES, LIgatures & Kerning

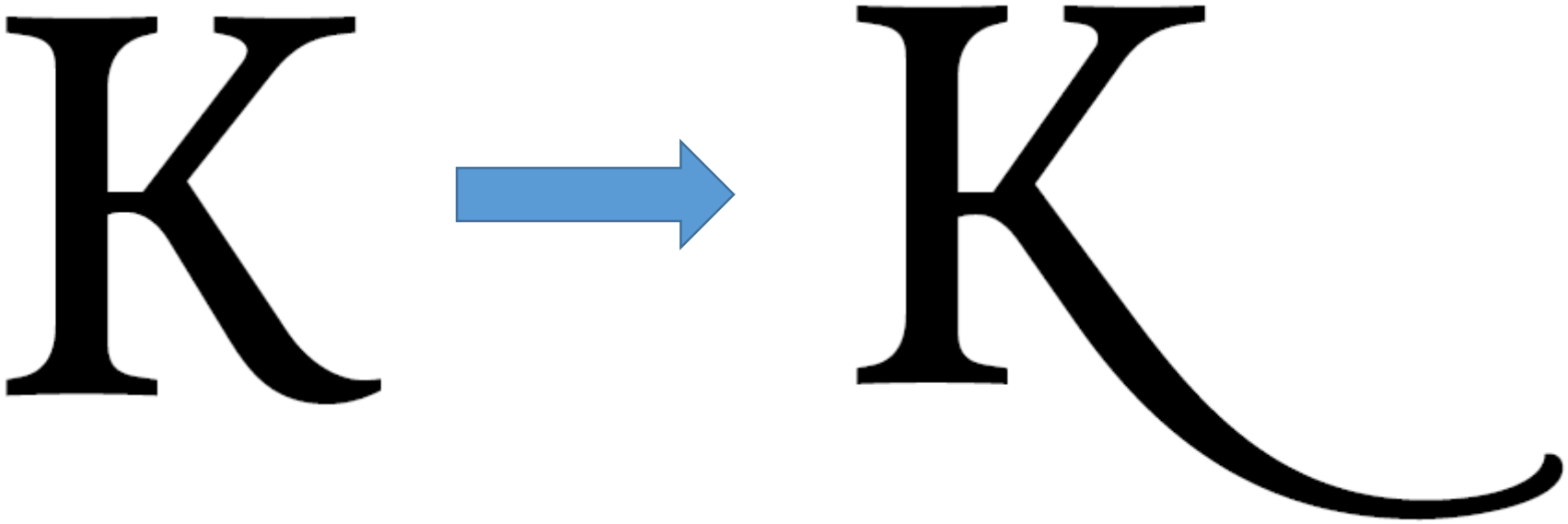
THE SWASHES, LIgatures & Kerning

THE SWASHES, LIgatures et Kerning

PDF Implementation

- Kerning: Text positioning + text showing operators (Tm/Td + Tj), or TJ operator
- [(A) 120 (W) 120 (A) 95 (Y again)] TJ
- Ligatures, swashes, other substitutions: output correct glyph id and specify /ToUnicode mapping correctly (if you want to be able to extract the text from PDF afterwards)

/ToUnicode CMAP



- Different glyph ids, same Unicode
- <002a><002a><004b>
- <00f8><00f8><004b>

/ToUnicode CMAP

- Some ligatures have Unicode values, but some do not
- ff (U+FB00): **'LATIN SMALL LIGATURE FF'**
- <02c4><02c4><005400480045>

OpenType features

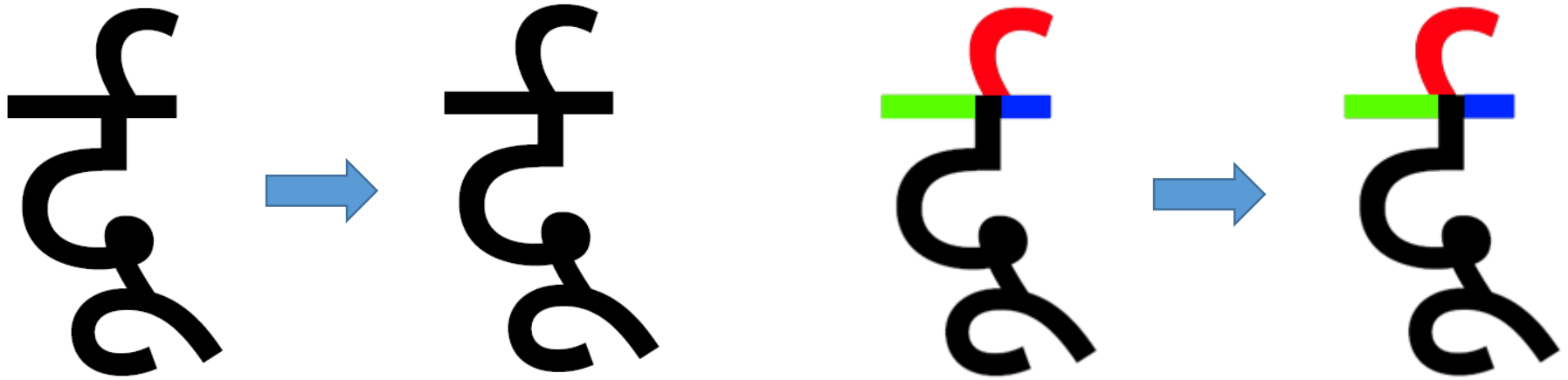
- 'aalt' Access All Alternates
- 'abvf' Above-base Forms
- 'abvm' Above-base Mark Positioning
- 'abvs' Above-base Substitutions
- 'afrc' Alternative Fractions
- 'akhn' Akhands
- …

And ≈130 more!

OpenType features: basic operations

- Substitute glyphs

- Adjust the positions of glyphs

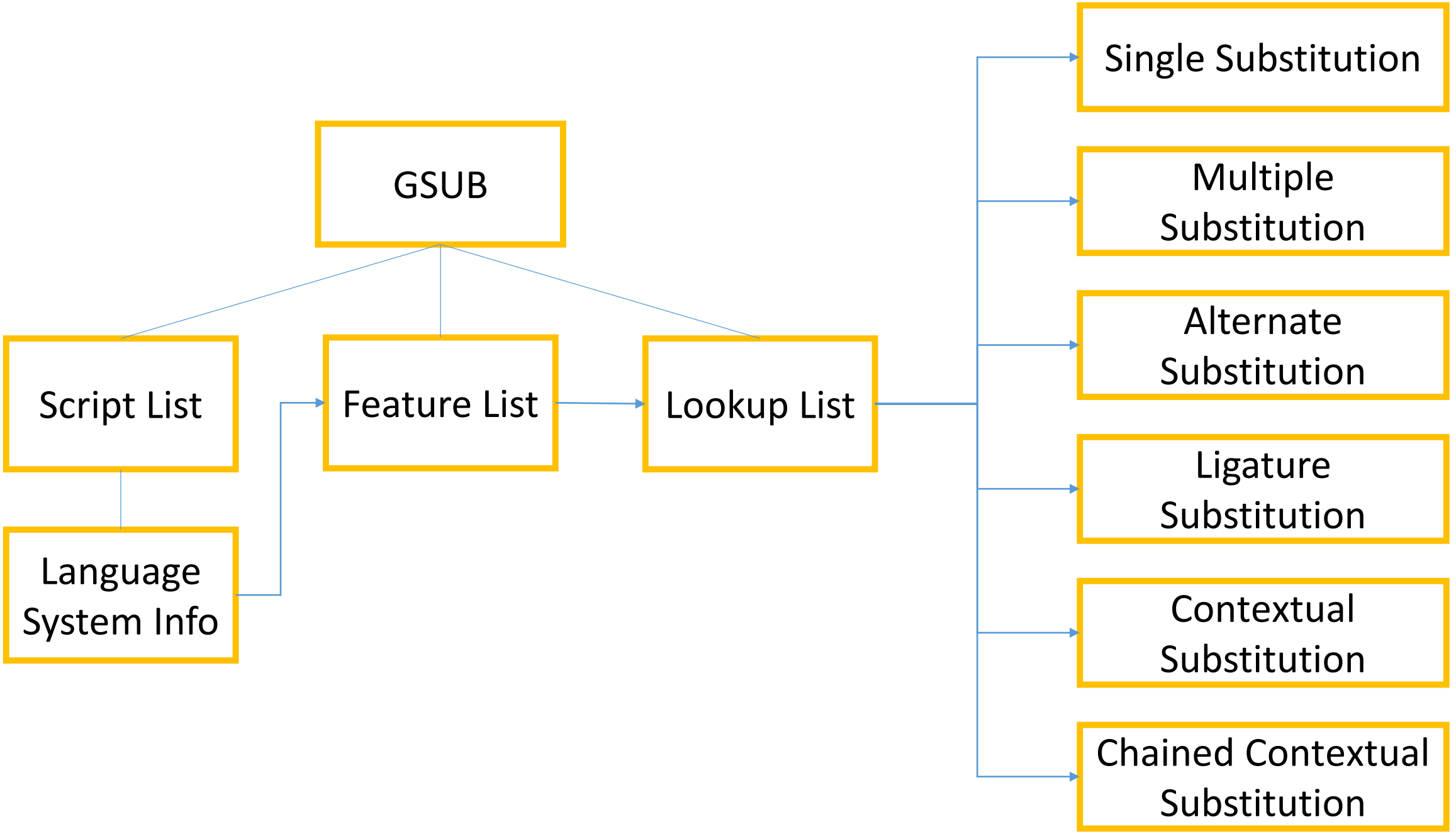


OpenType Layout tag registry

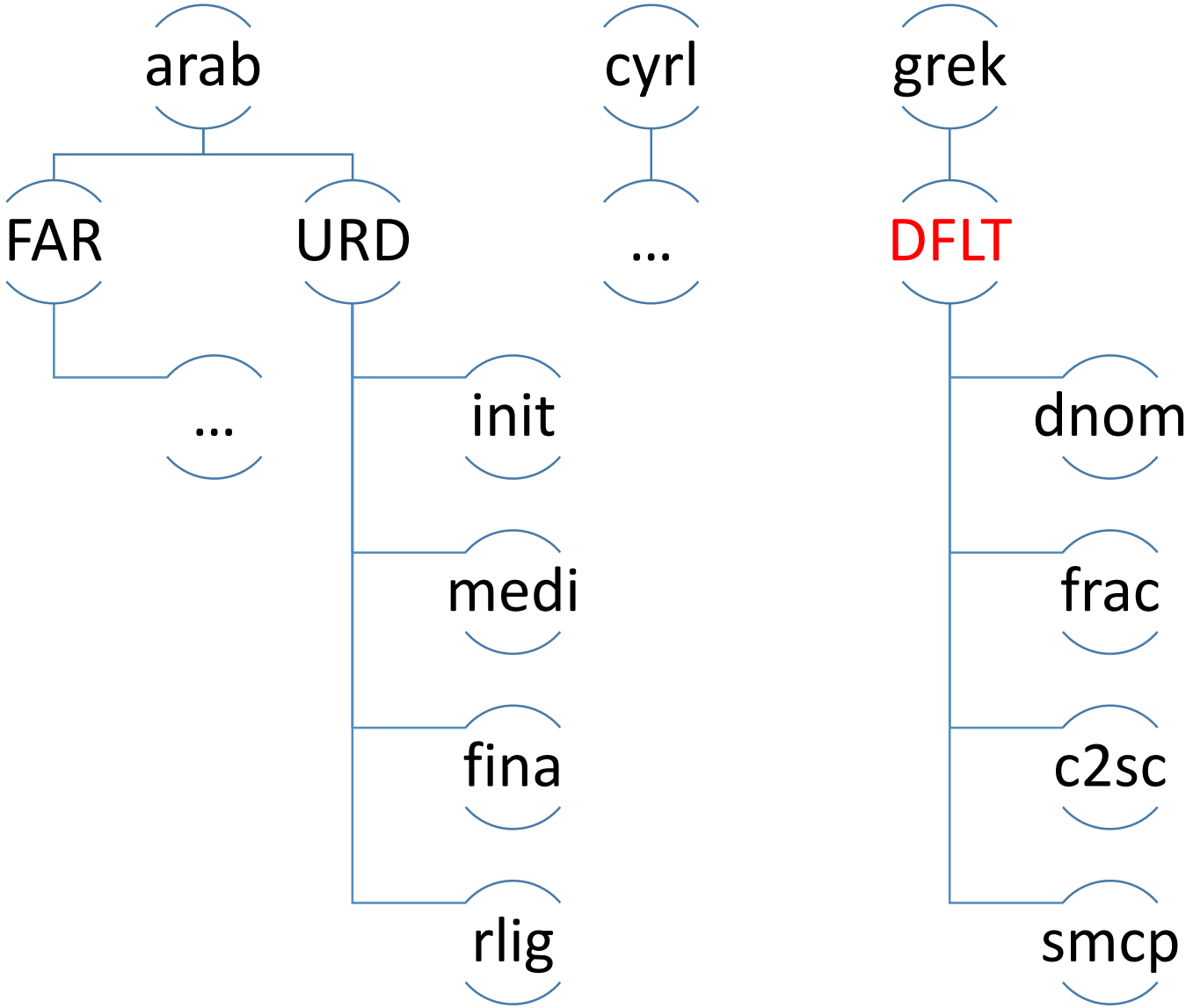
- ≈ 150 Script tags
- For some scripts there are old and new implementations (e.g. deva and dev2)
- ≈ 500 Language tags
- ≈ 140 Feature tags
- Font developers also may define and register their own features
- How is everything organized?

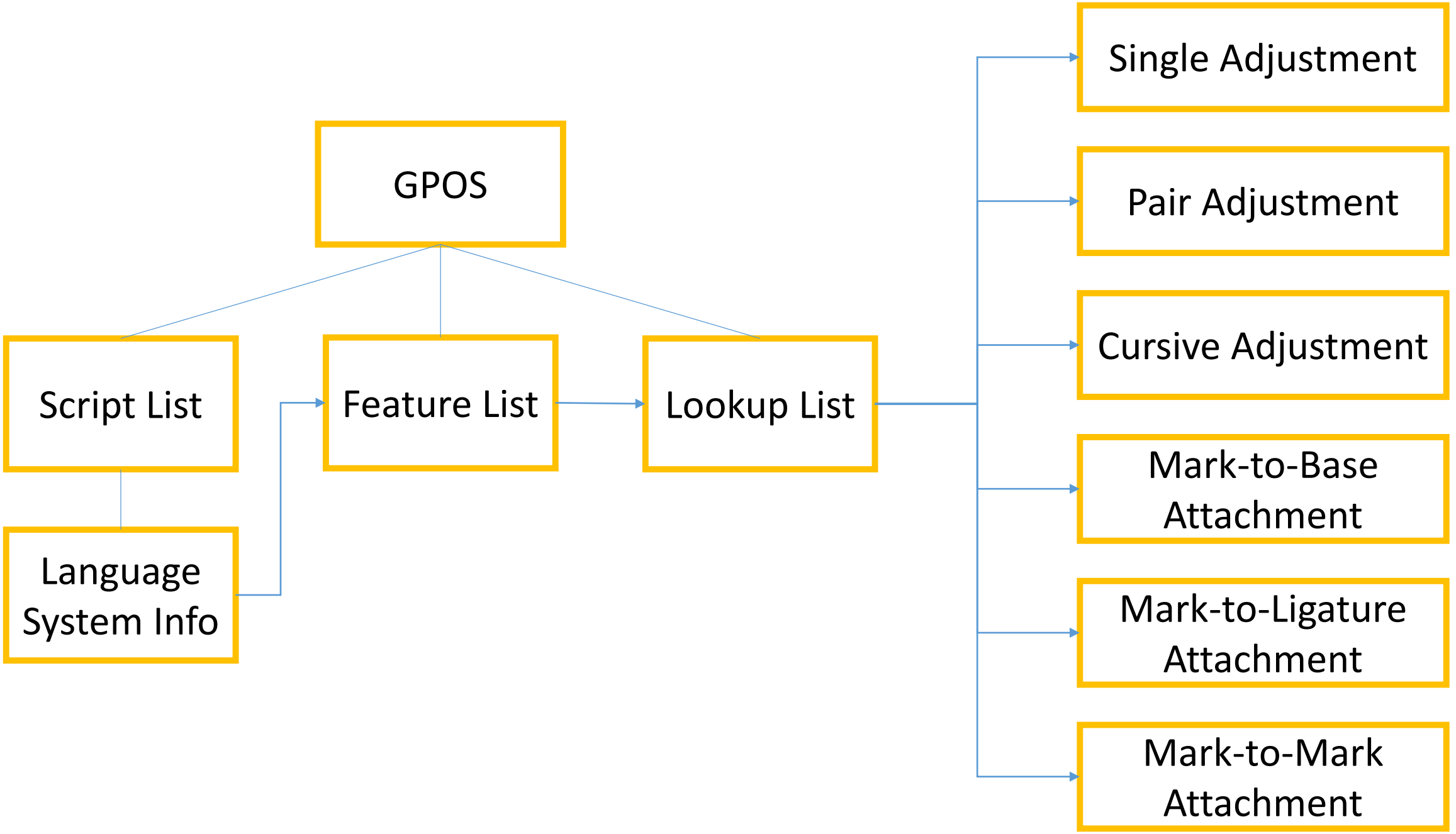
GSUB and GPOS tables in the OpenType font

- GSUB = Glyph Substitution
- GPOS = Glyph Positioning


GSUB
Script List
Language System Info
Feature List
Lookup List
Single Substitution
Multiple Substitution
Alternate Substitution
Ligature Substitution
Contextual Substitution
Chained Contextual Substitution

Example


arab
cyrl
grek
FAR
URD
…
DFLT
…
init
dnom
medi
frac
fina
c2sc
rlig
smcp


GPOS
Script List
Language System Info
Feature List
Lookup List
Single Adjustment
Pair Adjustment
Cursive Adjustment
Mark-to-Base Attachment
Mark-to-Ligature Attachment
Mark-to-Mark Attachment

Features

- How do we know which features to apply?
- How and when to apply them?

Feature	Feature function	Layout operation	Required
Language based forms:			
ccmp	Character composition/decomposition substitution	GSUB	
Typographical forms:			
liga	Standard ligature substitution	GSUB	
clig	Contextual ligature substitution	GSUB	
Positioning features:			
dist	Distances	GPOS	X
kern	Pair kerning	GPOS	
mark	Mark to base positioning	GPOS	X
mkmk	Mark to mark positioning	GPOS	X

[GSUB = glyph substitution, GPOS = glyph positioning]

Indic scripts: overview of the algorithm

- Clustering into syllables (Unicode)
- Reordering (Unicode, aside from font)
- Substitutions (OpenType features)
  - one to one
  - one to many
  - many to one
  - contextual
- Positioning (OpenType features)
  - kerning
  - mark positioning

Indic shaping algorithm: Unicode

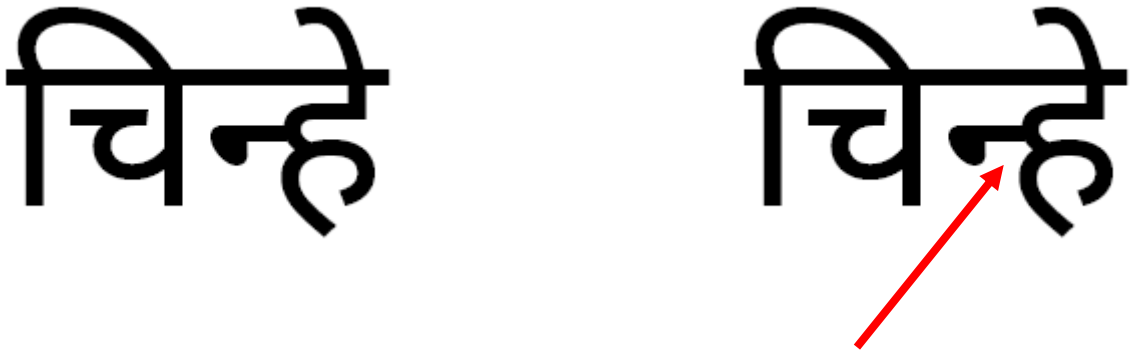
- Initial
  - \u091A\u093F\u0928\u094D\u0939\u0947
- Clustering into syllables
  - \u091A\u093F\u0928\u094D\u0939\u0947
- Reordering
  - \u093F\u091A\u0928\u094D\u0939\u0947

Indic shaping algorithm: OpenType

- Initial

- GSUB

- GPOS



ITEXT

dualLab

GSUB features implementation

- Single substitution:
  - One to one
  - Replacement glyph might not have Unicode value (swashes)
  - Remember Unicode value and replace glyph id.
- Multiple substitution:
  - One to many
  - Do not confuse with Unicode decomposition
  - Same approach
  - How to enable copying? (/ToUnicode)

GSUB features implementation

- Alternate substitution:

  - One to one of many
  - Same approach
- Ligature substitution
  - Many to one
  - Same approach and define /ToUnicode as described before

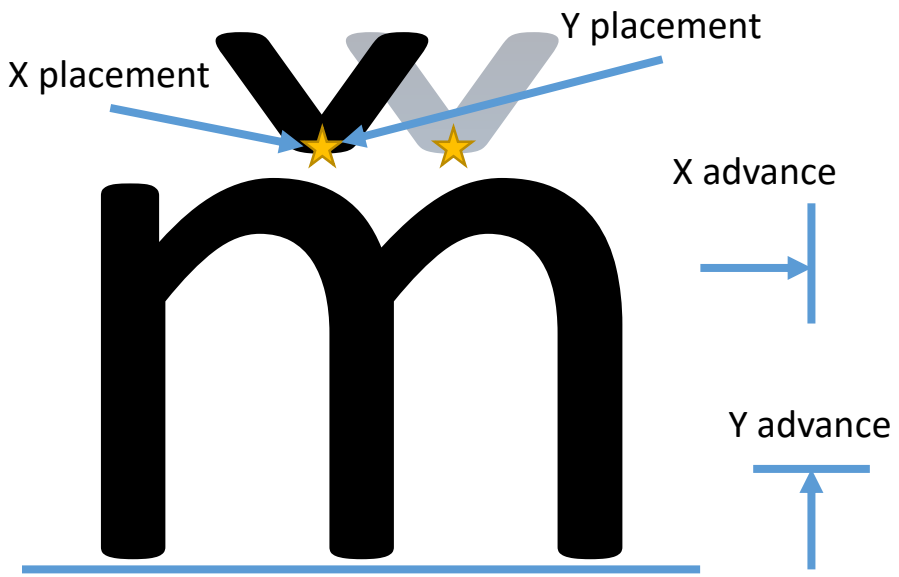
GPOS features

- Single adjustments: superscript or subscript
- Pair adjustments: kerning
- Cursive attachment: connect glyphs with attachment points
- MarkToBase attachment: position mark characters with respect to base glyph
- MarkToLigature attachment: associate mark with one of the ligature glyph’s components
- MarkToMark attachment: attach one mark to another

GPOS features implementation

- Placement
- Advance
- Glyph attachment points
- Offset to attaching point

GPOS features implementation



GPOS features implementation

- Check if current glyph has placement
  - Move the cursor to the position of the glyph the current glyph is attached to *(**Td**)*
  - Apply xPlacement and yPlacement to move the origin to the anchor *(**Td**)*
  - Show current glyph *(Tj / TJ)*
  - Roll back the cursor to the initial position *(**Td**)*
- Apply xAdvance and yAdvance *(**Td**)*

GPOS features implementation

- Horizontal placement => *Tj + Td* can be replaced with *TJ*
- Vertical placement (yPlacement != 0 **or** yAdvance != 0) => *TJ* **is not enough** => **need to use** *Td*
- Might be a problem for text extraction

Back to /ToUnicode

The underlying Unicode sequence is:

\u0935\u0930\u094D\u0923\u094B\u0902

(व + र + ◌् + ण + ◌ो + ◌ं)

Content stream glyph ids: 39, 27, 1C4

???

\u0935\u0930\u094D\u0923\u094B\u0902

(व + र + ् + ण + ो + ं)

Two buffer approach

- Text editors keep two buffers
- Buffer with Unicode string
- Buffer with glyph ids
- Easy correspondence for non-breakable parts (syllables)
- Cursor goes over syllables
- Windows: Uniscribe
- Linux: ICU - International Components for Unicode

Two syllables

- Cursor in your browser knows that!
- \u0935\u0930\u094D\u0923\u094B\u0902

PDF Approach

- Have only content stream and glyphs written there
- /ToUnicode
- Have to map all the glyphs to Unicode characters

1C4 ??

1C4

\u0935\u0930\u094D\u0923\u094B\u0902

Indic does not work that way

◌्र	◌्र	◌्र	◌्रं
◌्र	◌्र	◌्र	◌्रं
◌्र	◌्र	◌्र	◌्रं
◌्र	◌्र	◌्रं	◌्रं
◌्र	र्	र्◌ं	र्◌ं
◌्र	र्	र्◌ं	र्◌ं

ITEXT
dualLab

How to map glyphs to Unicode?

- 39, 27, 1C4
- **\u0935\u0930\u094D\u0923\u094B\u0902**
- 39 <-> **u0935**
- 27 <-> **u0923**
- 1C4 <-> ???
- Easy without reordering, but not in our case
- **\u0935|\u0930\u094D\u0923|\u094B\u0902**
- Incorrect when copying single glyphs
- **Incorrect when adding new words**

How to map glyphs to Unicode?

- 39, 27, 1C4
- 27 <-> **u0923**
- **\u0935|\u0930\u094D\u0923|\u094B\u0902**
- \u0935\u0930\u094D\u0917\u0940\u0915\u0930**\u0923**
- Extra chars will be copied along with the word
- \u0935\u0930\u094D\u0917\u0940\u0915\u0930\u0930\u094D\u0923
- Challenge for most of the PDF producers even today

/ActualText comes to save us

- Can be specified for content that does translate into text but that is represented in a nonstandard way *(**ISO 32000-1**)*
- Replacement text can be specified for the following items:
  - A structure element, by means of the optional **ActualText** entry *(PDF 1.4)* of the structure element dictionary.
  - A marked-content sequence, through an **ActualText** entry in a property list attached to the marked-content sequence with a **Span** tag.

/ActualText comes to save us

- **\u0935**\u0930\u094D\u0923\u094B\u0902
- [39, 27, 1C4]
- /ToUnicode CMAP:
  - 39 <-> **u0935**
  - 27 <-> **u0923**
  - 1C4 <-> \u094B\u0930\u094D\u0902

/Span <</ActualText <FEFF 0930 094D 0923 094B 0902> >> BDC

<002701C4>Tj

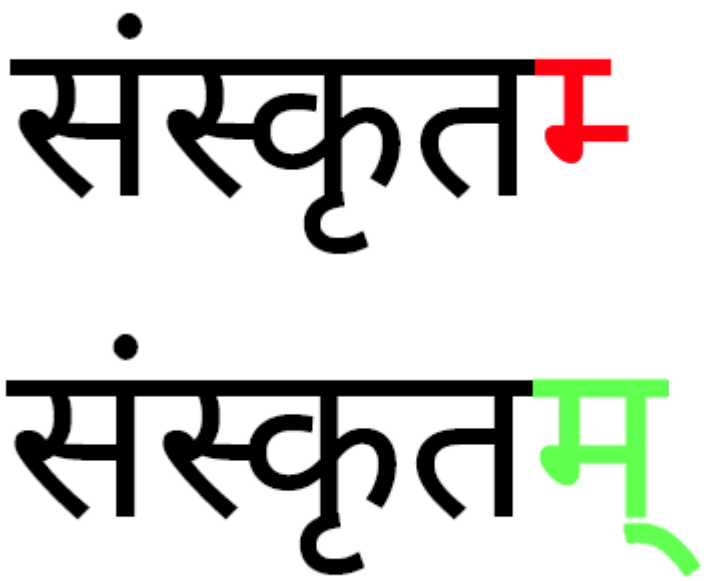
EMC

/ActualText

- Not supported in many PDF viewers
- Problems with determining spaces when extracting text

Features + algorithms

- Lookup tables don’t know script rules
- Half characters
  - त + व = त्व *tva*
  - ण + ढ= ण्ढ *ṇḍha*
  - स + थ = स्थ *stha*
- Don’t blindly apply all features
- Set up masks for features during preprocessing



Arabic

- Right-to-left
- Unicode => logical order
- init, medi, fina, liga
- /ReversedChars
  - /ReversedChars BMC
  - (olleH) Tj
  - –200 0 Td
  - (. dlrow) Tj
  - EMC

Arabic

يلامجالا رعسلا

السعر الاجمالي

السعر الاجمالي

Why OpenType?

- All non-obligatory font-specific features + positioning
- Many ligatures do not have Unicode equivalent as there are too many of them because of script-specific rules => encode them in **lookup tables**
- Different correct representations of a text: some glyphs might be present in a font, some may not => too hard to check all options => encode transformations in **lookup tables**

Conclusions

- OpenType features
  - Obligatory (Indic, Arabic shaping)
  - Non-obligatory (Latin Swashes, Kerning)
  - Unicode preprocessing for complex scripts
  - Work in pair with algorithms and script rules
- PDF + OpenType = solid (and necessary) match, but…
  - *Td* even for showing a single word (vertical positioning)
  - /ActualText for complex scripts text extraction (two buffer analogue)

References

- OpenType specification - https://www.microsoft.com/en-us/Typography/OpenTypeSpecification.aspx
- Microsoft Typography - https://www.microsoft.com/en-us/Typography/default.aspx
- FontForge Open Source tool - https://fontforge.github.io
- OpenType CookBook - http://opentypecookbook.com/index.html

Questions? प्रश्न? ?أسئلة

- Benoît Lagae benoit.lagae@itextpdf.com
- Alexey Subach alexey.subach@duallab.com