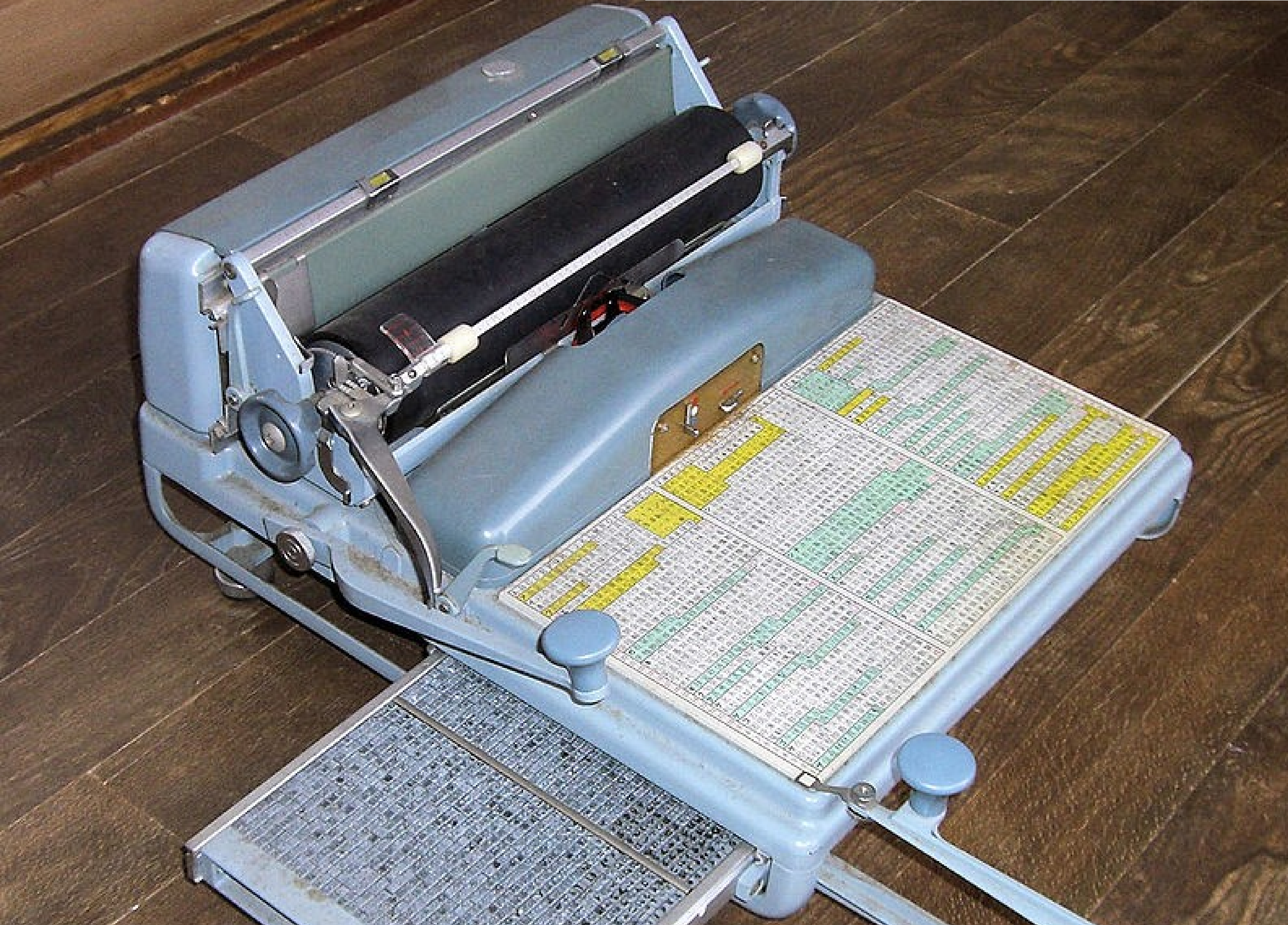
Next Generation Input Methods

Presented by
Daiki Ueno, Anish Patil

Today's Topics

- Japanese input basics 😉
- The algorithm behind it
- Next generation IM features
- Architecture

Japanese input basics


Japanese Typewriter SH-280, CC-BY 3.0, by miya

Japanese input in one slide

- ASCII sequence
  - kyouhaiitenkidesune
- Japanese alphabets (Kana)
  - きょうはいいてんきですね
- Japanese sentences (Kana + Kanji)
  - 今日はいい天気ですね
  - きょうは良い天気ですね
  - …

Character conversion 1:1

Sentence conversion 1:N

There's no single solution, though
extremely rare combinations are not acceptable

How does it work?

1. Split input string into possible substrings
2. Assign Chinese characters to each substring
3. Find the most likely output

1. Split into substrings

- き | ょうはいいてんきですね
- きょ | うはいいてんきですね
- …
- きょう | は | いいてんきですね
- きょう | はい | いてんきですね
- …
- きょう | は | いい | てんきですね

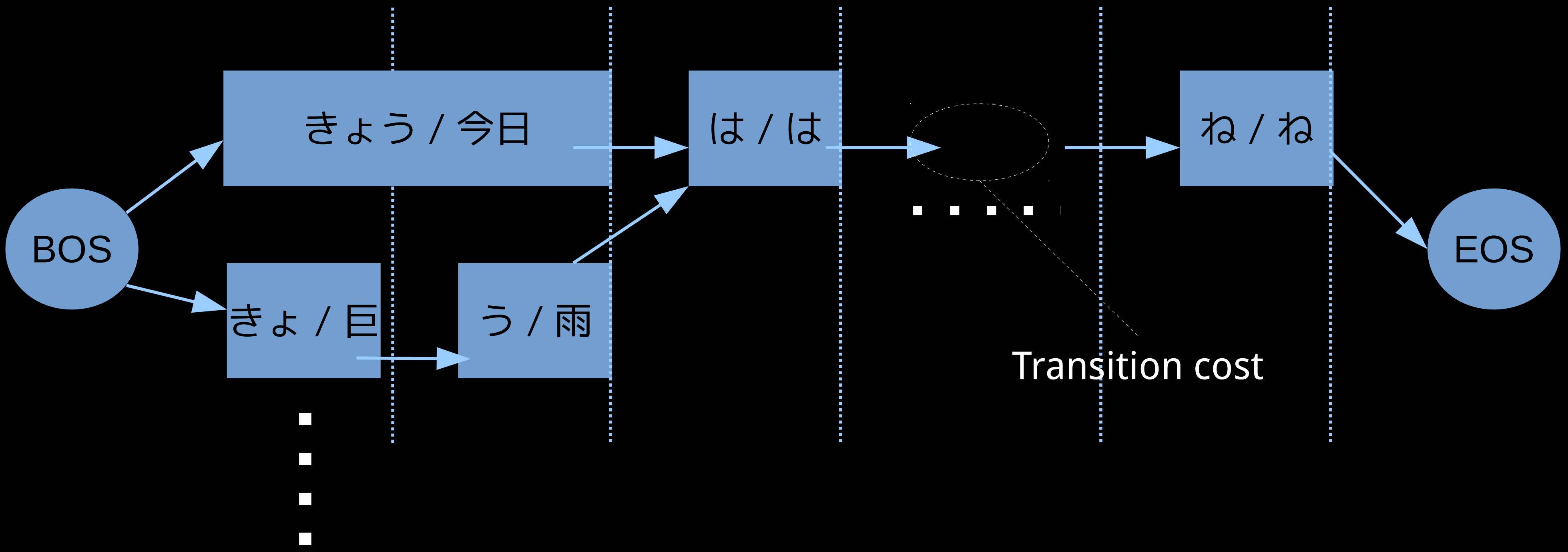
$$N=\frac{n(n-1)}{2}$$

2. Assign Chinese characters

- 木 | ょうはいいてんきですね
- 巨 | うはいいてんきですね
- …
- 今日 | は | いいてんきですね
- 今日 | 杯 | いてんきですね
- …
- 今日 | は | 良い | てんきですね

$$\left.\right\} \quad N' = \sum_{k=1}^{N} C_k$$

3. Find the most likely output



Now it turned into the shortest path problem.
But, how can we assign costs?

Language model

- Assigns probability of sentence or words
  - 1-gram: 1 word
  - 2-gram: 2 consecutive words
  - 3-gram: 3 consecutive words
  - …
- Generated from a large set of examples
  - Based on **features** of each word
    - Notation, part of speech, length, …

Implementation: libkkc

- Language model
  - 3-gram language model generated from:
    - Wikipedia (Japanese): 100,000 sentences
    - Yahoo! Chiebukuro (Q&A site): 20,000 sentences
  - Only using notation of each word
- > 90% accuracy
  - To recover sentences from newspaper articles

Next generation IM features

Problems

- The language is changing
- User's language skills are spoiled by computers

Language change

Natural language reflects current events

- あべ (pronunciation: əbe) is a popular Japanese family name, written as:
  - 阿部 , 安倍 , 安部 , or 阿倍
- When Mr. 安倍 was appointed as the Japanese prime minister
  - あべしゅしょう should be 安倍首相 , not 阿部首相
  - あべせいけん should be 安倍政権 , not 阿部政権

Language change (cont'd)

Misuse sometimes becomes formal

- 怒り心頭に<u>達する</u> = たっする
- 怒り心頭に<u>発する</u> = はっする

Lots of new words / phrases emerge from slangs

Possible solutions

- Do conversion online
  - Privacy issues
- Release language model frequently
  - It could be large and require bandwidth
- Interpolate language model with updates
  - May affect accuracy

Language skills are spoiled

- Cumbersome to type the whole sentence
- Can't remember the formal usage of a word
- Can't remember the pronunciation of a character
  - We have thousands of characters

Possible solutions

- Predictive input
- Handwriting input

Predictive input

Suggest next possible word or phrase, from the previously input words and history

- Implementations
  - POBox, MS-IME, Google Japanese input, ibus-typing-booster
- Issues
  - Privacy: history carries sensitive information

Handwriting input

Find a character by handwriting shape, drawn using a pointing device

- Implementations
  - Mac OS X, ibus-handwrite
- Issues
  - Accuracy
  - Writing speed

Common issues

- New UI elements are needed
  - No user distractions
  - Don't interfere with other applications
    - e.g. Web browser suggestions
- The current IBus implementations are PoC
  - Implemented as a separate IBus engine
  - Aren't backed by real engines

Architecture

Yet another IM architecture?

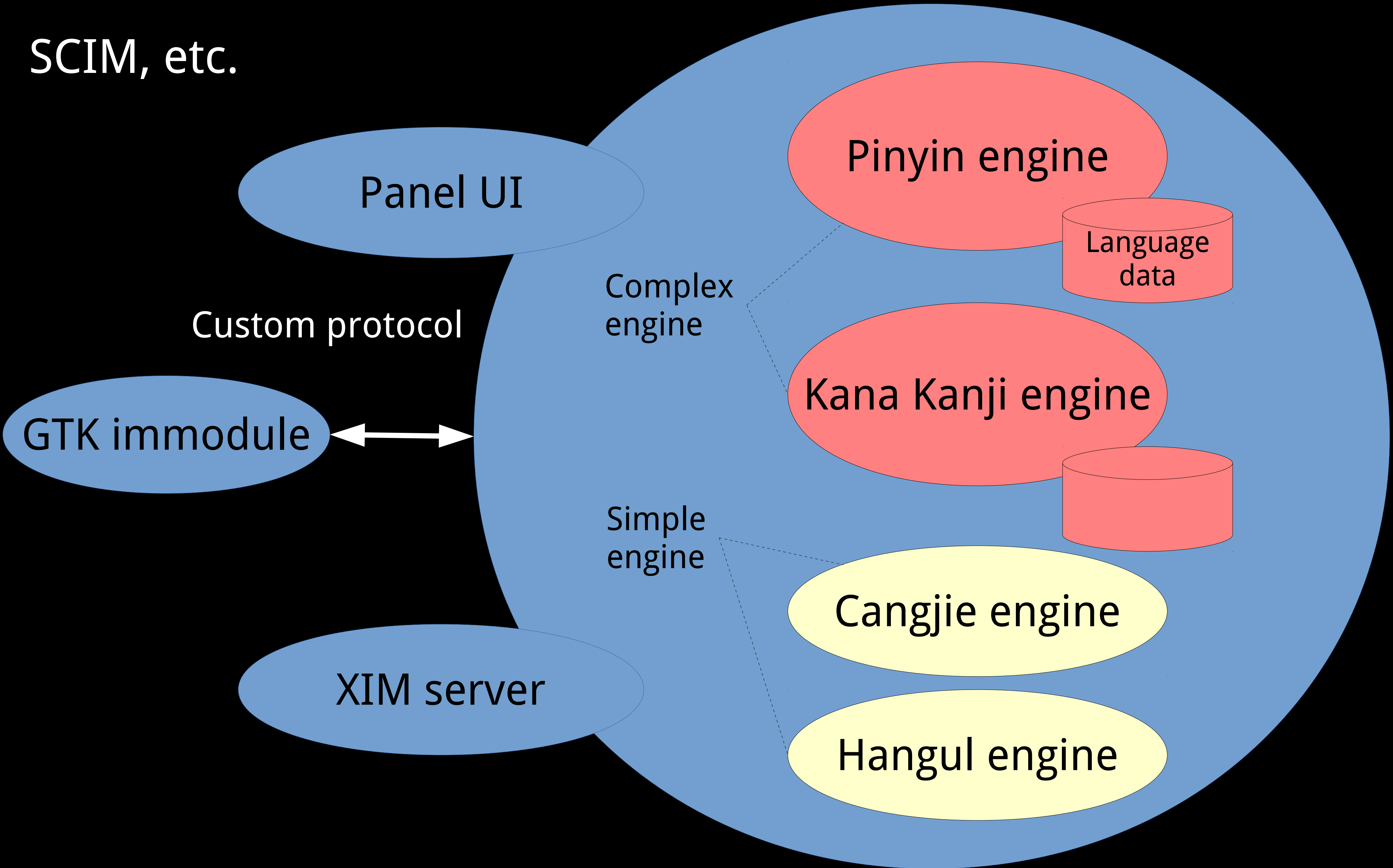
- Are you proposing an IBus alternative?

  No, no

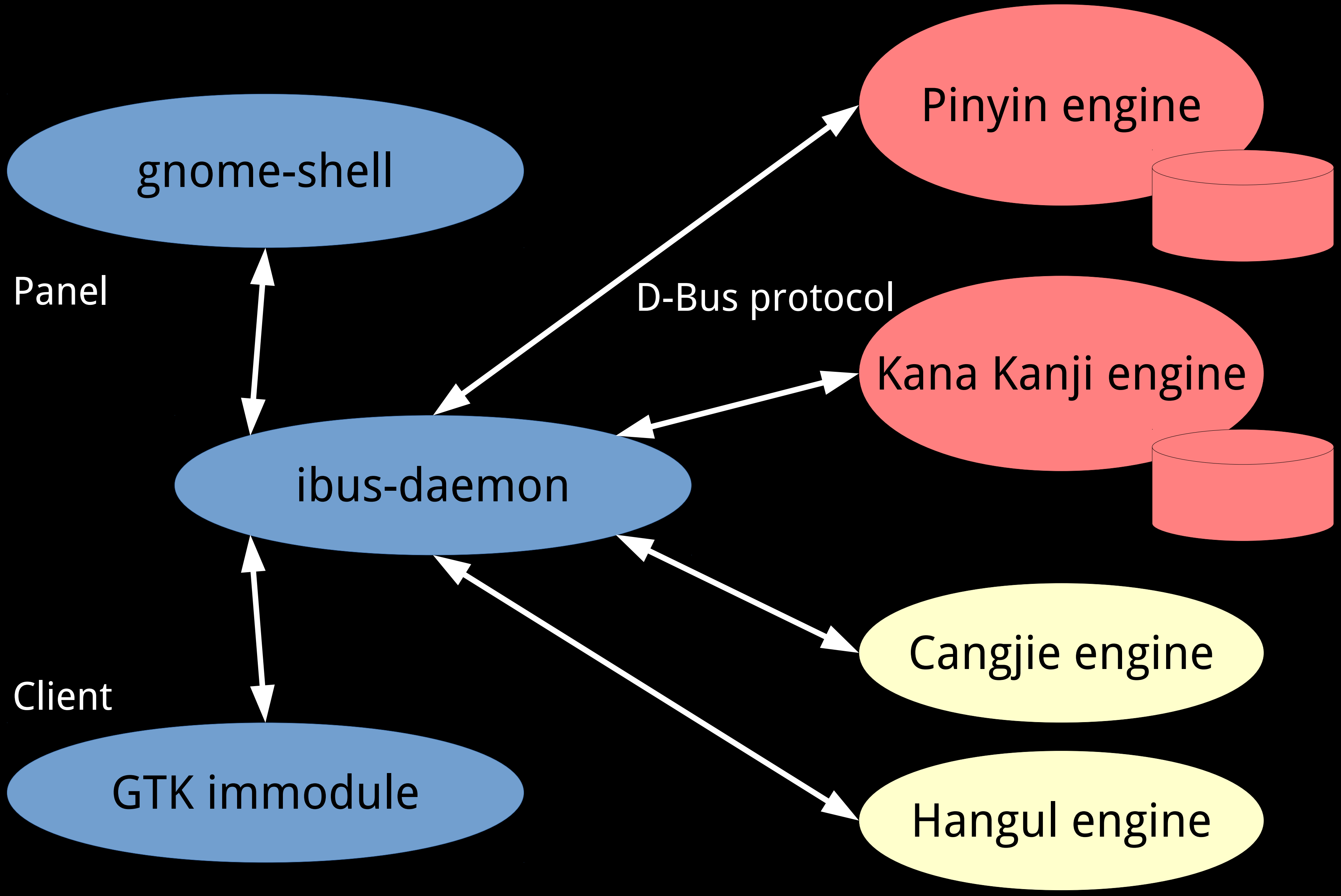
- This is a renovation project
- What's wrong with the IBus architecture?

Traditional IM architecture

SCIM, etc.



IBus architecture



IBus architecture (cont'd)

- ibus-daemon
  - Re-implementation of dbus-daemon
  - Manage engine registration and input-contexts
- Engine
  - Do actual language-specific input conversion
- Panel
  - Provide UI stuff

IBus architecture (cont'd)

- Pros
  - Crash resistant
  - Stable panel API, based on D-Bus
- Cons
  - Slow response for input events
  - Implementation issues

Implementation issues

- Unresponsiveness
  - The API is not fully asynchronous
  - Newly installed engines are not recognized immediately
  - Don't recover crashed engine
- Small number of test cases
  - ~30% code coverage

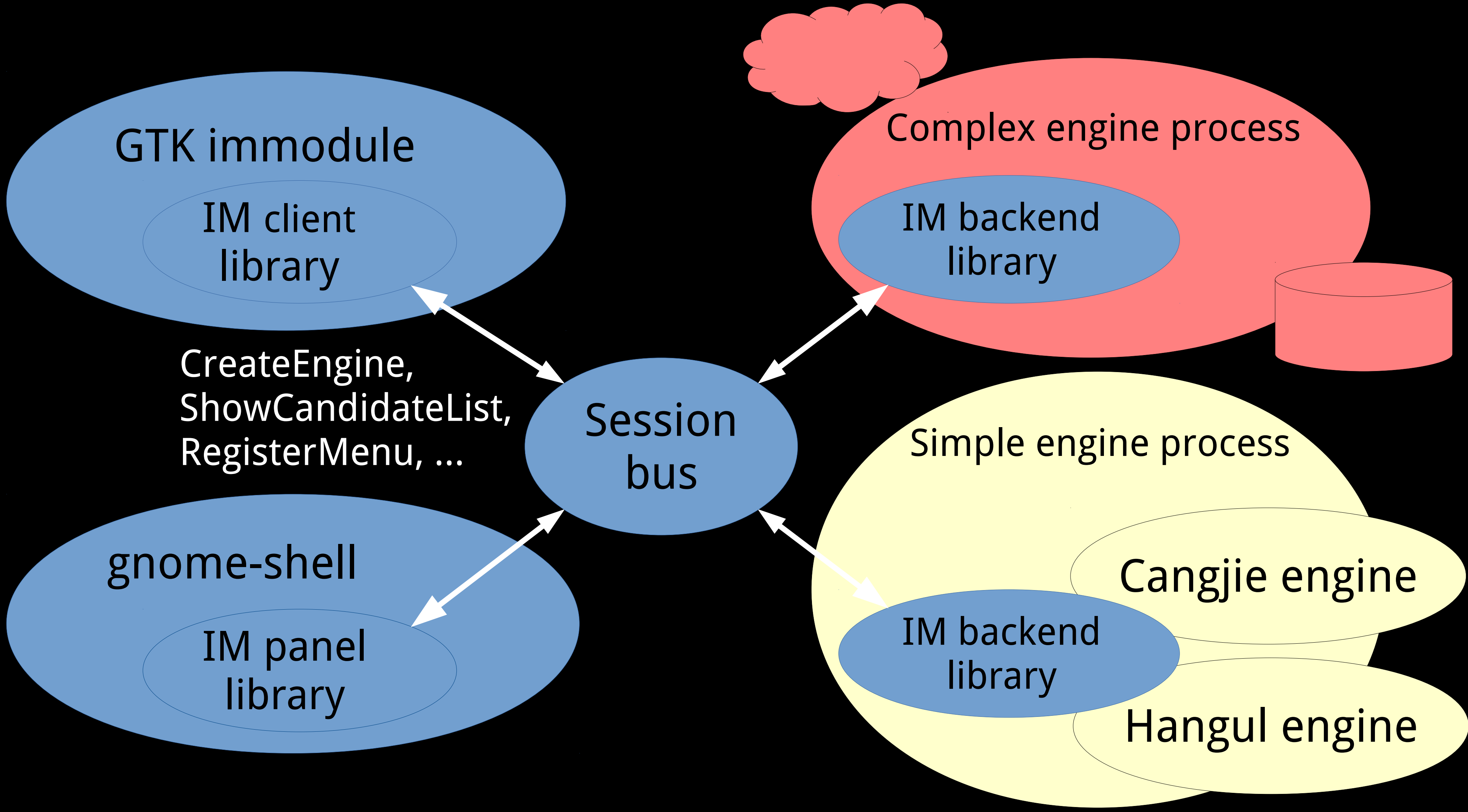
Goals

- Unified UI for predictive/handwriting input
- Privacy
- Performance

Our approach

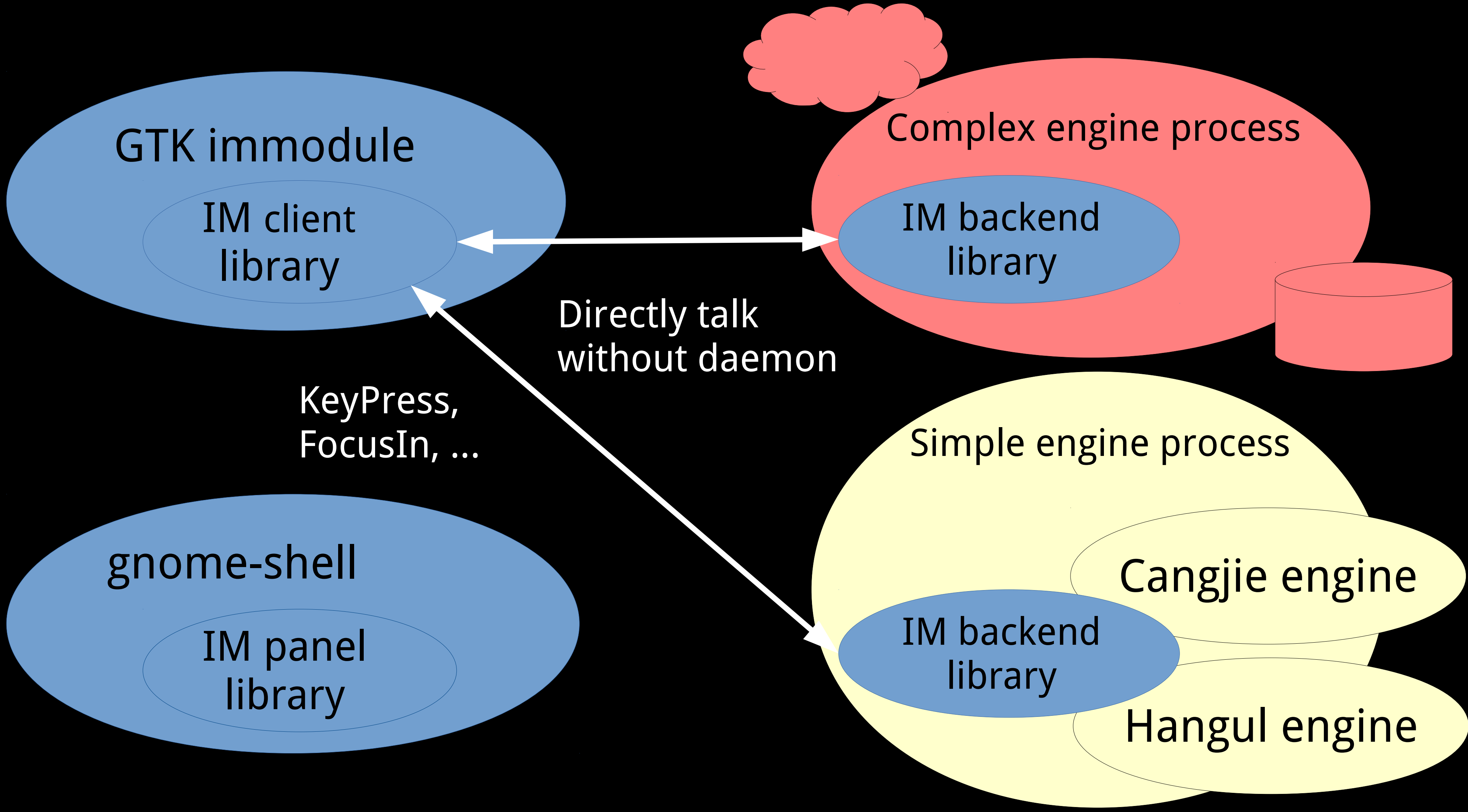
- Make engines more like an ordinary GNOME application / service
  - Shall be registered through a .desktop file
  - Take advantage of sandboxing?
- Eliminate ibus-daemon
  - Use session bus for: UI and engine activation
  - Use peer-to-peer connection for input events

Proposed architecture



CreateEngine,
ShowCandidateList,
RegisterMenu, ...

Proposed architecture



Libraries?

- Provide compatibility with IBus API, through GI
- Make the panel API extensible
- Manage connections between client and engines

Questions?