Next Generation Input Methods

Presented by
Daiki Ueno, Anish Patil

Today's Topics

I. UX of Future Input Methods

by Anish

ibus-typing-booster developer

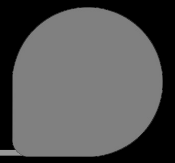
II. Architecture of Future Input Methods

by Daiki

IBus and Japanese engine developer

I. User Experience

Auto completion

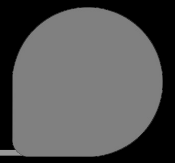


- Need?
- Projects?

Problems?

- Candidate lists
- Good Dictionaries
- Uniform Experience

Solutions



- Tab completion
- Delay while showing suggestions
- Show suggestions inline
- Show suggestions on fixed length bar and select candidates using numbers

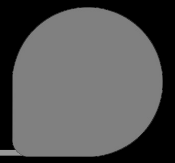
Dictionaries

- Users
- Use cases?

Issues?

- Words are not updated frequently
- Incorrect words in dictionaries
- No unique upstream
- Analysis on present stage

Demo



- http://webwordedit-wwe.rhcloud.com/

II. Architecture

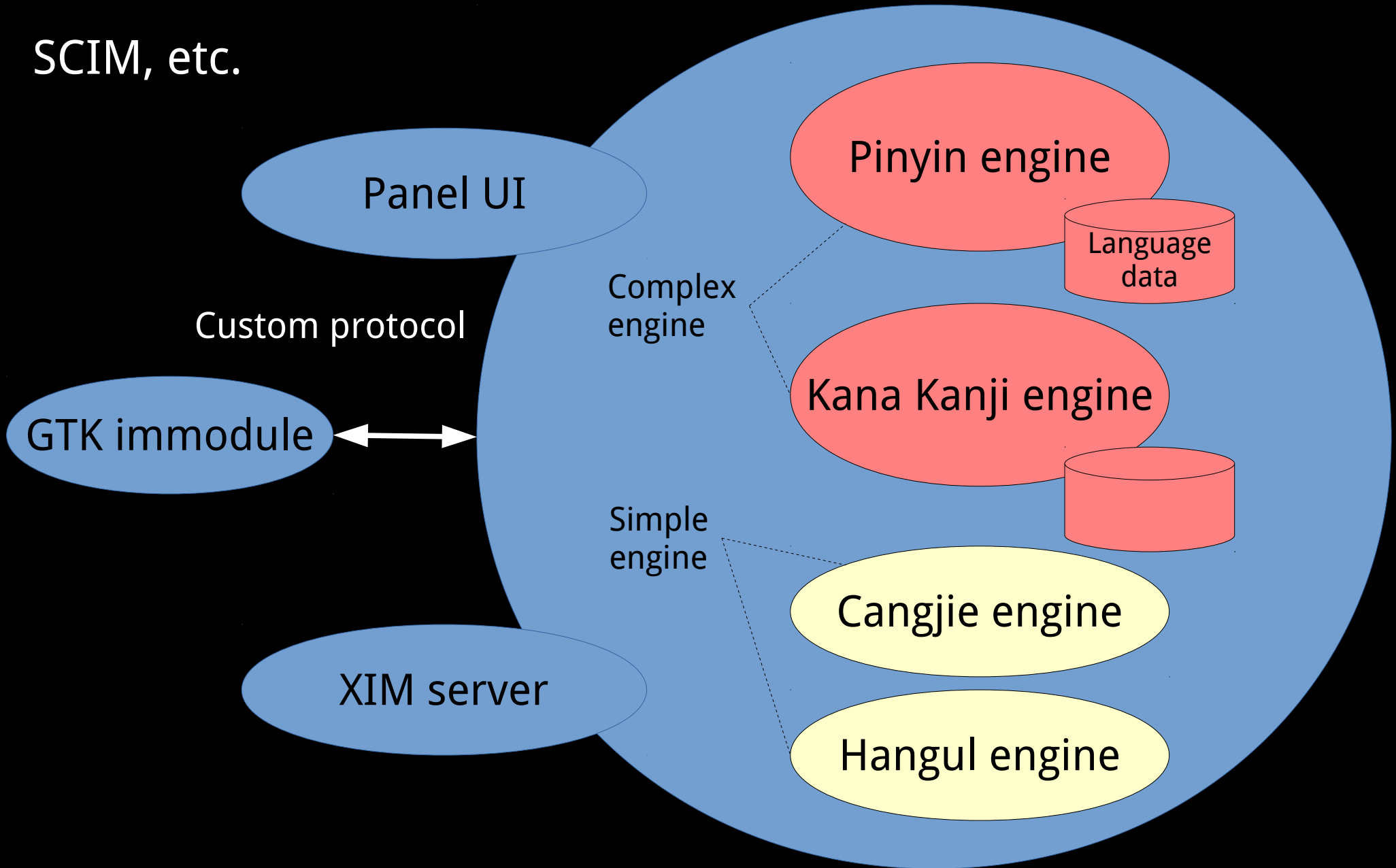
Yet another IM architecture?

- Are you creating an IBus alternative?

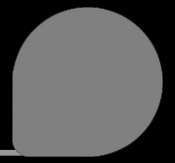
  No!

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

Traditional IM architecture



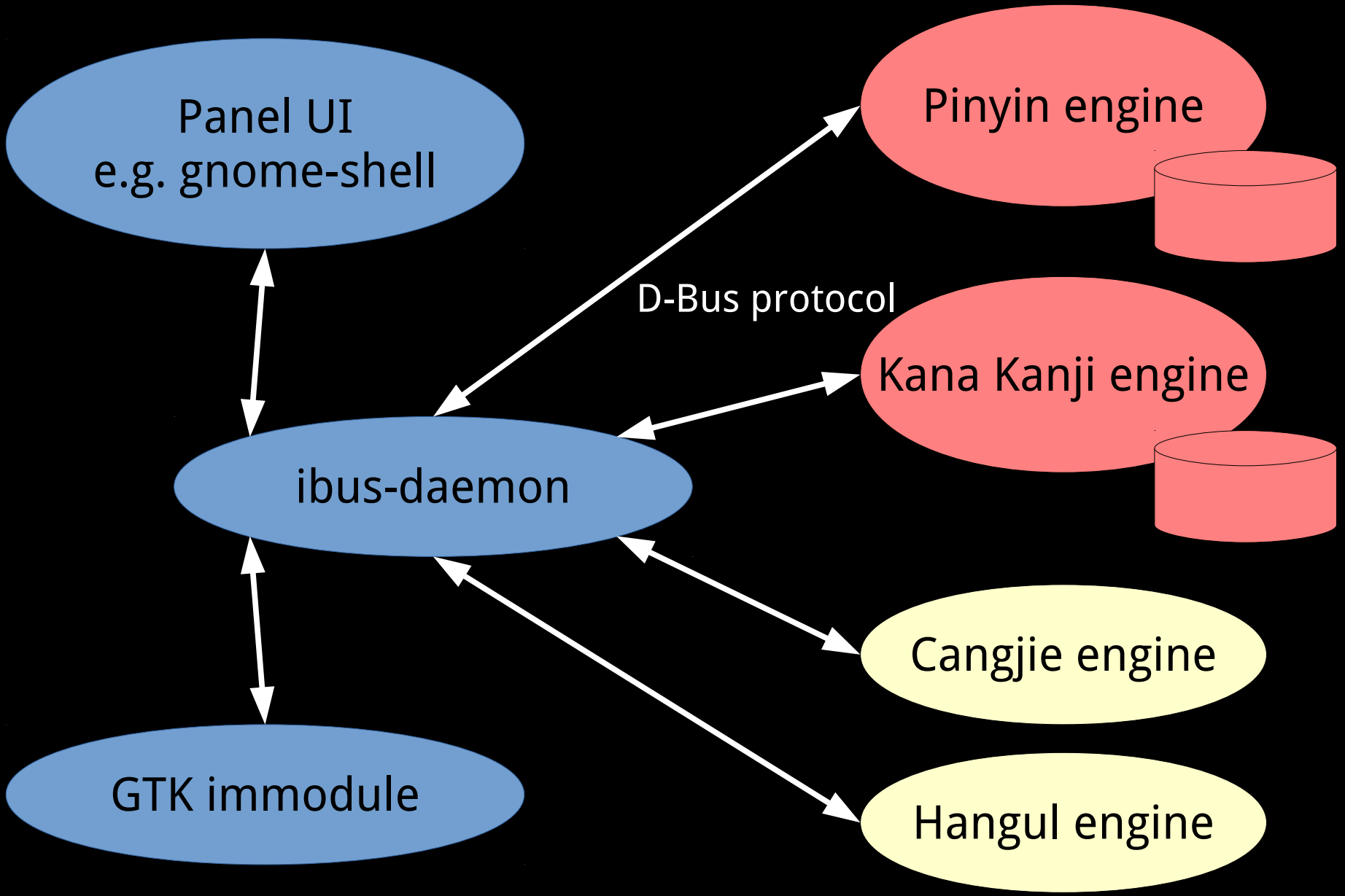
Traditional IM architecture



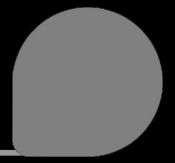
Almost all components run in a single process

- Pros
  - Fast response
- Cons
  - One engine can make the whole system unusable
    - Some engines are very complex by nature and can be irresponsive on high resource usage

IBus architecture



IBus architecture



Every component run as a separate process

- Pros
  - Crash resistant
  - Stable frontend (panel) API, based on D-Bus
- Cons
  - Slow response – IPC costs
  - Complicated implementation

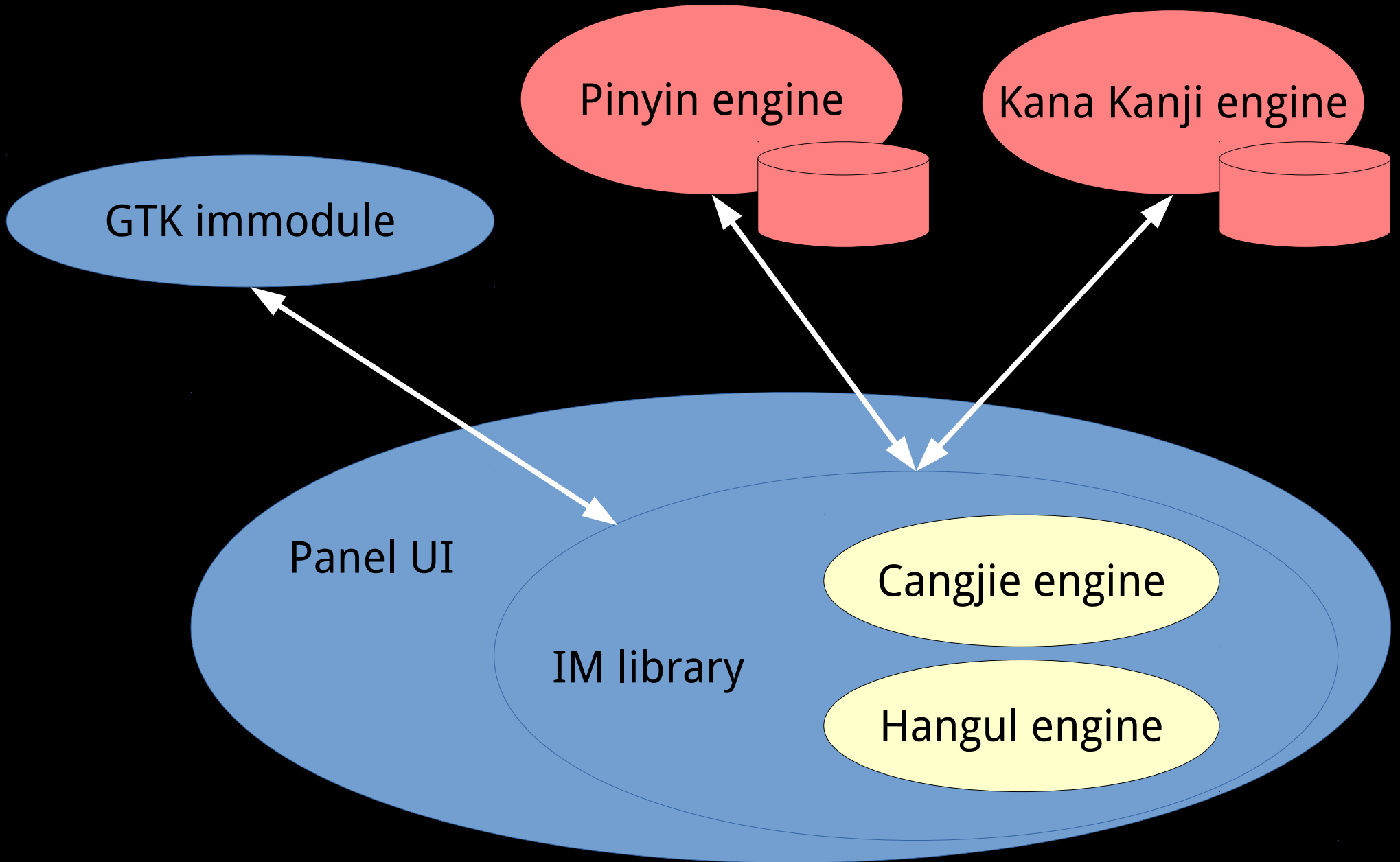
IBus implementation issues

- Slow switching of engines
  - The backend API is not fully asynchronous nor cancellable
- Process management glitches
  - No mechanism to recover crashed engine
  - Newly installed engines are not recognized until ibus-daemon restarts
- Small number of test cases
  - ~30% code coverage

Our approach

- Separate out unstable components only
  = Complex engines like Pinyin and Kana Kanji
- Provide the IM architecture itself as a library
  - gnome-shell can directly use it with the same API as IBus, through gobject-introspection,

Hybrid architecture



Hybrid architecture

Only unstable components run separately

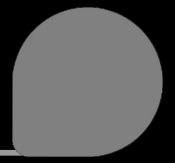
- Pros
  - Crash resistant
  - Fast response from simple engines
  - Stable frontend API, through gobject-introspection
- Cons
  - Even more complicated implementation
    - But we can do better, using the recent inventions

      gdbus-codegen, GCancellable, GTask,

Prototype: libtextinput

- Claim invocation type in the XML description
  - In-process or out-process
- Automatic crash recovery
- Demo client as a Wayland input-method
- Reuse IBus engines as a shared library (WIP)

Summary



- IBus frontend API is stable and good, however...
- The architecture needs renovation
  - Fully asynchronous backend API
  - Smarter process management
  - Reduced IPC costs
  - More tests

Questions?