A vision for Multi-Platform App Development in Rust

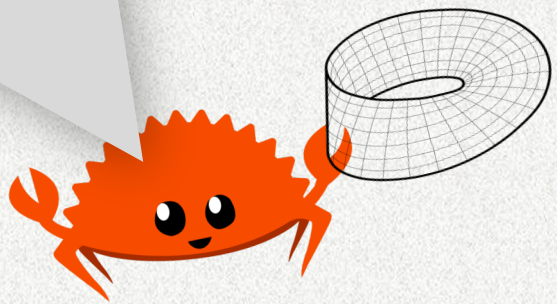
Kevin Boos, Futurewei Technologies

GOSIM 2023

1. Create a cross-platform app dev toolkit completely in Rust
   - Devs can leverage the robust, safe, and performant Rust ecosystem
   - No need to complicate things with other languages

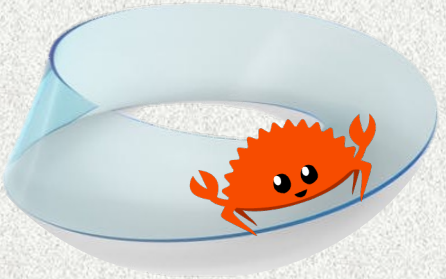
2. The Rust experience on Mobile feels neglected
   - Make Mobile a first-class concern in the Rust community

Robius at a glance



Disclaimer

Not formally affiliated with the Rust project or Rust Foundation



Motivation

What motivates our vision?

1. App devs want to use Rust, but aren't sure where/how to start
   - Paralysis of choice among dozens of partial solutions
   - Unclear how to integrate many disparate projects
2. Business advantages are clear
   - Consistent experience for devs *and* customers
   - Avoid redundant dev effort → save money, faster time to market
3. Rust is great! (not for cross-platform apps, yet)
   - Safety → increased correctness, reliability
   - Performance efficiency → responsive, jank-free UI
   - Potential to unify two dev worlds: systems + frontend apps

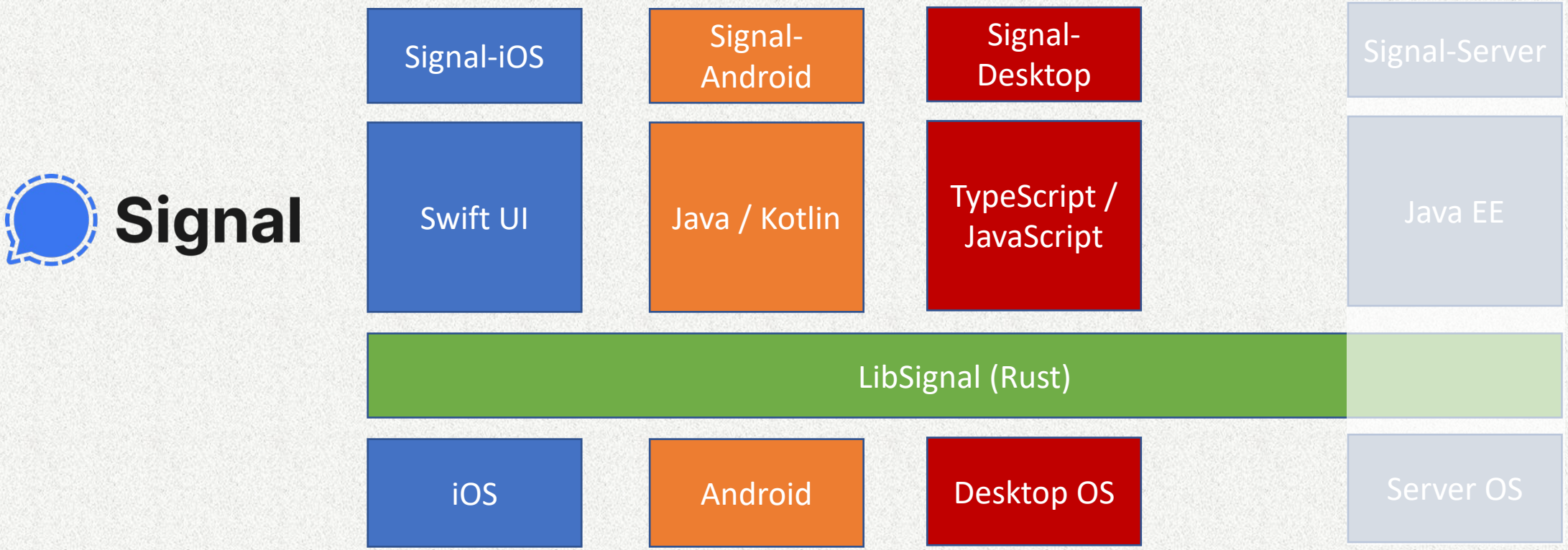
→ Attract front-end developers to the world of Rust

1. State of the Rust app dev ecosystem is rough

- Fractured ecosystem – many competing crates
  - No official recommendation for any specific toolkit(s)
  - Discovery of crates is difficult, non-standardized
  - Many crates offer overlapping features, are abandoned/incomplete
- Unclear integration opportunities
  - Which crates work with which others?
- Poor documentation; lack of examples, tutorials

2. A strong business case for Rust app dev

- Apps developed (partially) in Rust have platform-related pain points



Credit: Yong He, Futurewei

2. A strong business case for Rust app dev

- Signal maintains 3 separate app projects, with little overlap

iOS

```
http://cloc.sourceforge.net v 1.64  T=44.61 s (60.7 files/s, 13899.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Swift                         1630          78143          42943         401181
Objective C                    214          10058           3493          50473
JSON                           568              2              0           8247
C/C++ Header                   228           3537           2198           7811
Python                          25           1469            608           5632
SQL                              1            143              0           1319
Protocol Buffers                11            237            221           1225
Bourne Again Shell              12             74             29            243
YAML                             8             59             46            205
make                             3             20              7             77
Bourne Shell                     7             30             42             64
Perl                             1             21            136             27
-------------------------------------------------------------------------------
SUM:                          2708          93793          49723         476504
-------------------------------------------------------------------------------
```

450K LoC

Android

```
http://cloc.sourceforge.net v 1.64  T=82.82 s (76.2 files/s, 12097.5 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
XML                           2113          47590         108376         334431
Java                          2483          60351          14490         250763
Kotlin                        1573          25629           7564         142292
Protocol Buffers                21            383            163           1884
Groovy                          32            278             21           1556
JSON                            45              0              0            722
Handlebars                       9             55              9            359
Python                           2             34              7            147
Bourne Shell                     2             28            108            105
IDL                             17             13              0             87
CSS                              1             18              0             80
YAML                             5             20              6             79
DOS Batch                        1             21              2             66
Prolog                           1             12              0             46
C++                              1             13              0             32
Javascript                       1              2              0             16
C/C++ Header                     1              3             12             14
ASP.Net                          1              1              0              4
-------------------------------------------------------------------------------
SUM:                          6310         134451         130758         736683
-------------------------------------------------------------------------------
```

350K LoC

Desktop

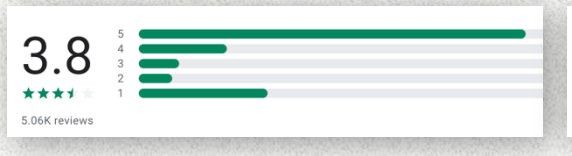
```
http://cloc.sourceforge.net v 1.64  T=35.06 s (43.2 files/s, 33541.2 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JSON                            155            141              0         887029
TypeScript                     1114          28077           8417         172721
Javascript                       35           1233           1565          47192
SASS                            174           3997            662          22226
Protocol Buffers                14            251            100           1257
YAML                            [illegible]    78             47            477
HTML                            [illegible]    18             25            392
Bourne Shell                    [illegible]    16             14             62
-------------------------------------------------------------------------------
SUM:                          1514          33811          10830        1131336
-------------------------------------------------------------------------------
```

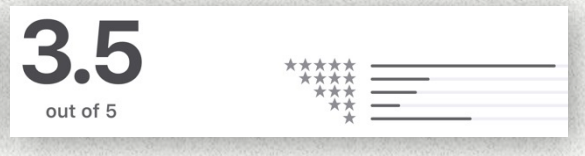
250K LoC

Credit: Yong He, Futurewei

2. A strong business case for Rust app dev

- **element**: first-party Matrix chat client
  - Separate app, SDK repos for each platform
  - Leads to buggy, inconsistent UI/UX behavior





element-x-ios Public
Swift 170 Apache-2.0 31 145 (5 issues need help) 5 Updated 5 hour

element-ios Public template
A glossy Matrix collaboration client for iOS
Swift 1,650 Apache-2.0 464 1,669 (7 issues need help) 20 Updated

element-web Public
A glossy Matrix collaboration client for the web.
TypeScript 10,019 Apache-2.0 1,797 3,308 (84 issues need help) 22

element-desktop Public
A glossy Matrix collaboration client for desktop.
TypeScript 885 Apache-2.0 204 313 22 Updated 2 hours ago

element-x-android Public
Android Matrix messenger application using the Matrix Rust Sdk and Jetpack Compose
Kotlin 296 Apache-2.0 34 96 (1 issue needs help) 9 Updated 3 h

element-android Public
A glossy Matrix collaboration client for Android.
Kotlin 2,976 Apache-2.0 616 2,014 (10 issues need help) 30 Upda

2. A strong business case for Rust app dev

- Element's experience of redundant effort led to *Compound*

- "Design-only Flutter"

Compound

Compound is the work-in-progress design system for Element.

What's Compound?

Compound is a shared language and set of UI component implementations for the web, iOS, & Android.

Compound is broken down into the following primitives:

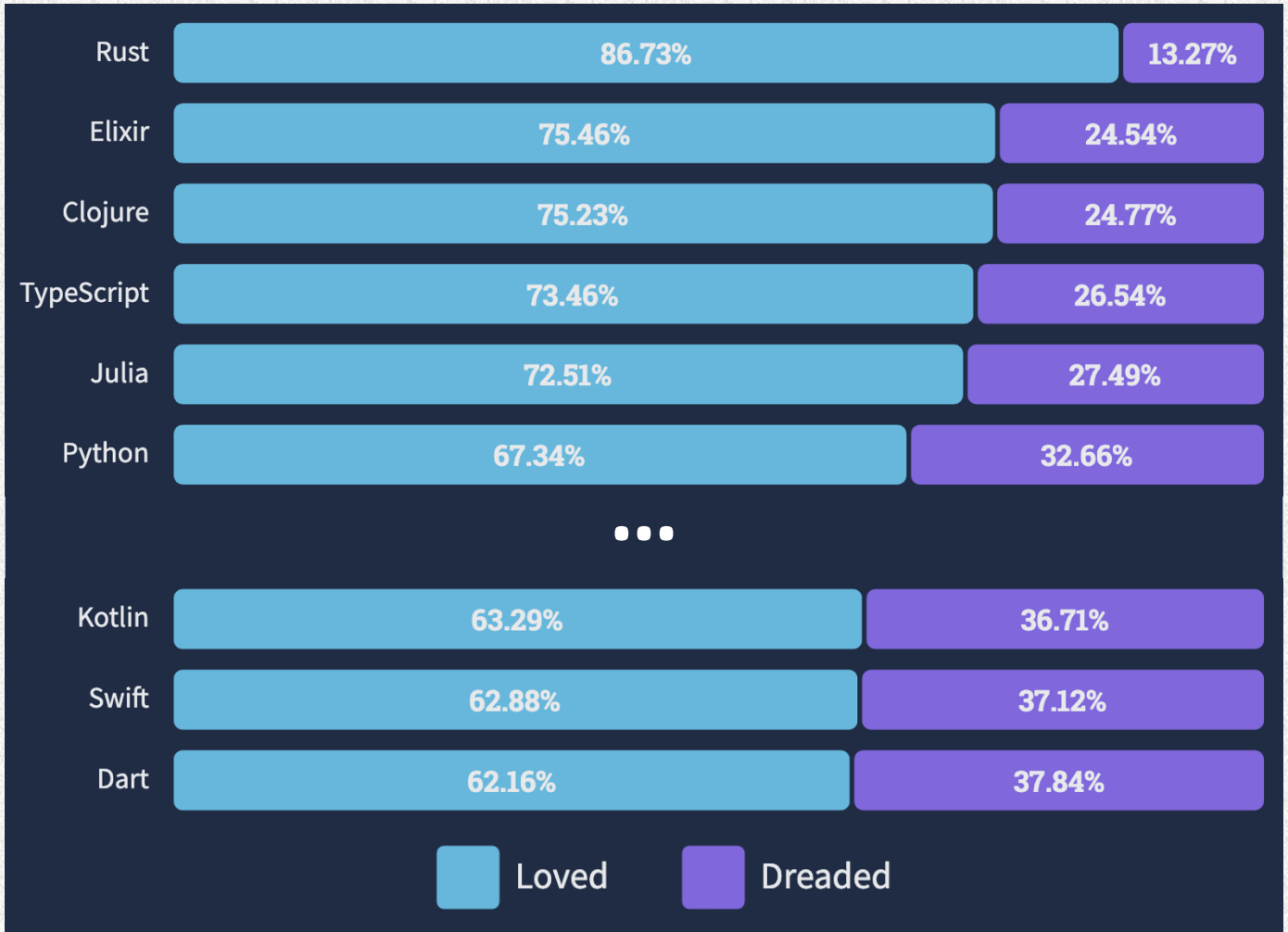
- Foundations: Guiding principles which inform the basis of any implementation.
- Styles: Systems for applying colour, type, spacing, sizing and materials.
- Components: Defined components used to build end user experiences.

Platforms

- ElementX (iOS & Android), Element Web, ...

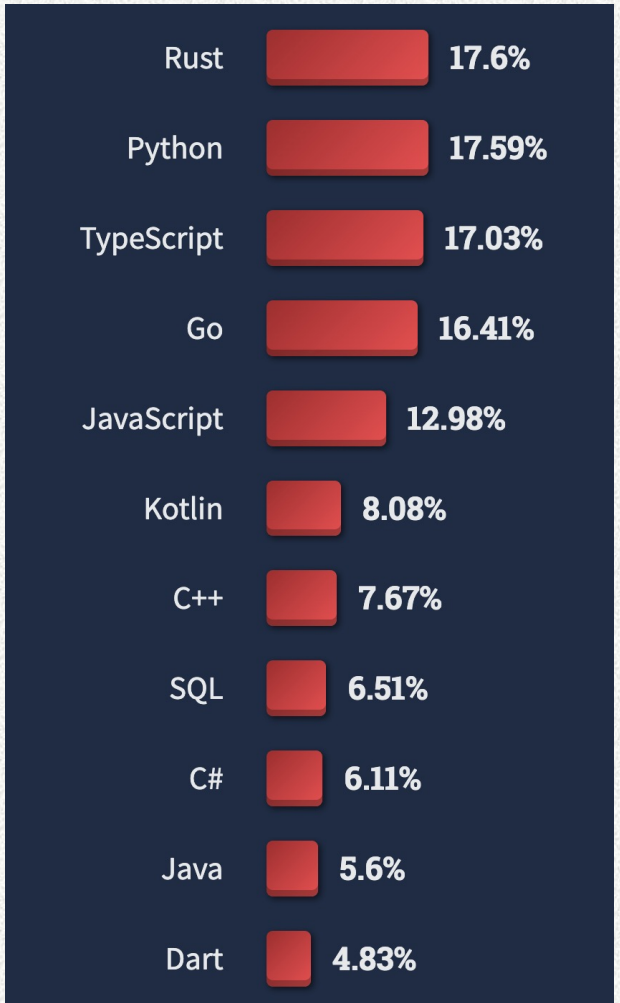
3. Rust itself is the right choice

Most loved language 8 years in a row



...

and most wanted



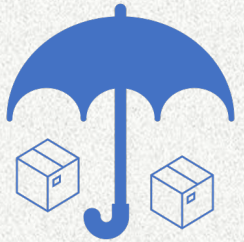
3. Rust itself is the right choice

Leverage Rust's great features for multi-platform app dev:

Safety, runtime efficiency

Easy, clear dependency mgmt

Simple cross compilation



Ecosystem covers many domains

3. Rust is great, but …

- Some shortcomings from a UI standpoint
  - Lack of familiar OOP patterns; UI folks are accustomed to inheritance
  - Difficult to realize shared mutable state
    - Typical closure-based callback pattern → overuse of `Rc<RefCell<…>>`
  - Compilation is “slow”
    - But constantly improving … wait for Nick’s talk

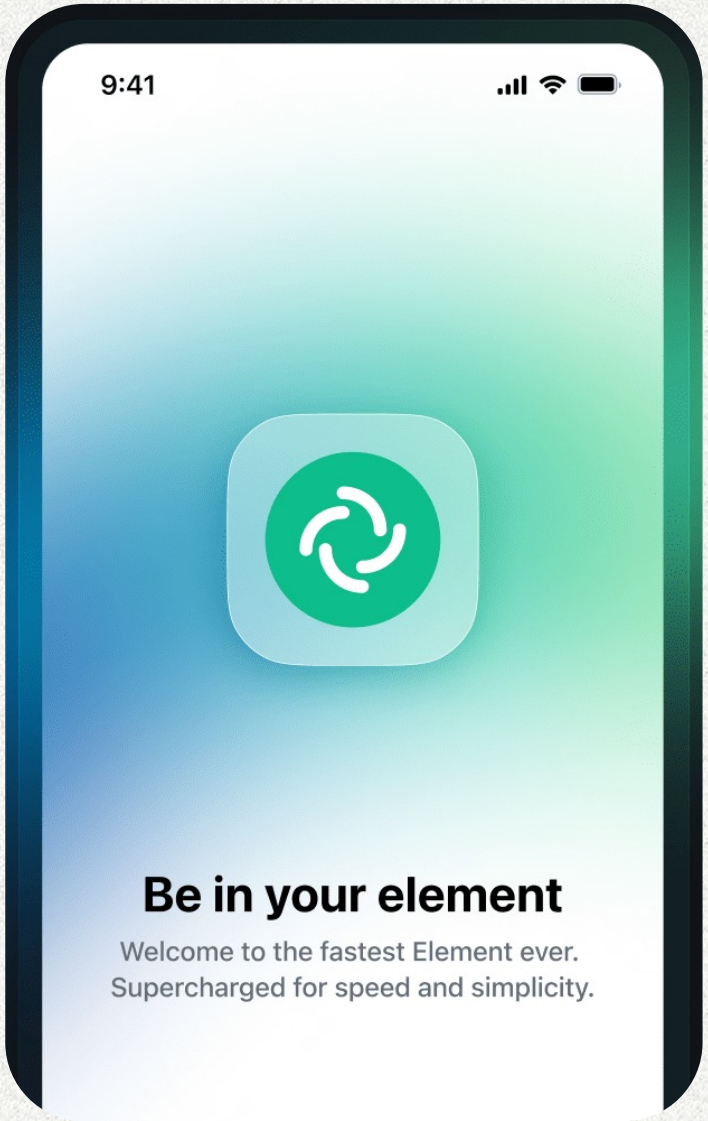
- Potentially steep learning curve for frontend devs

Mobile devs needed ~3.5 months of retraining to switch to Flutter

-- VERY GOOD VENTURES

Source: https://verygood.ventures/whitepaper/business-value-of-flutter

Theme: switching to Rust for *huge* gains 💪



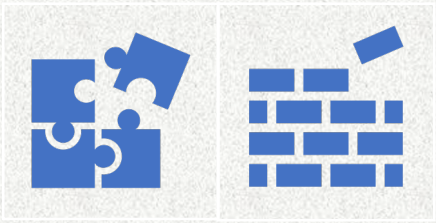
- Element X: “up to 6000 times faster”
  - Rewritten in Rust, with native frameworks
  - Rust Matrix SDK + SwiftUI / Jetpack Compose
- Speaks to **the attitude surrounding Rust**
- Motivates a cross-platform approach:

> We’re releasing Element X today on iOS; you can get it from the App Store (plus it runs well on macOS too, if you have Apple Silicon.) The Android build will follow [soon]…
>
> — *Element blog. July 6, 2023*

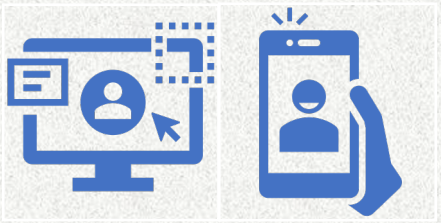
Source: https://element.io/blog/element-x-experience-the-future-of-element/

Goals

Establish community, drive ecosystem



Create reference design of full system stack



Develop flagship apps as proof-of-concept

Give feedback to Rust & Cargo teams

[Future]

Establish community, drive ecosystem

- Create working group
  - Track status on an “*Are we app yet?*” website

- Collect set of maintained, functional crates
  - Attract and support key contributors across system stack

- Drive holistic ecosystem documentation, tutorials

- Evangelize & demonstrate power of Rust for app dev
  - *It’s not just for systems or embedded projects anymore!*

Reference design of full system stack

- Complete, working example of everything beneath app
  - Support major platforms/OS environments

details coming later…

Proof-of-concept demos + flagship app

- A series of open-source, ready to use example apps

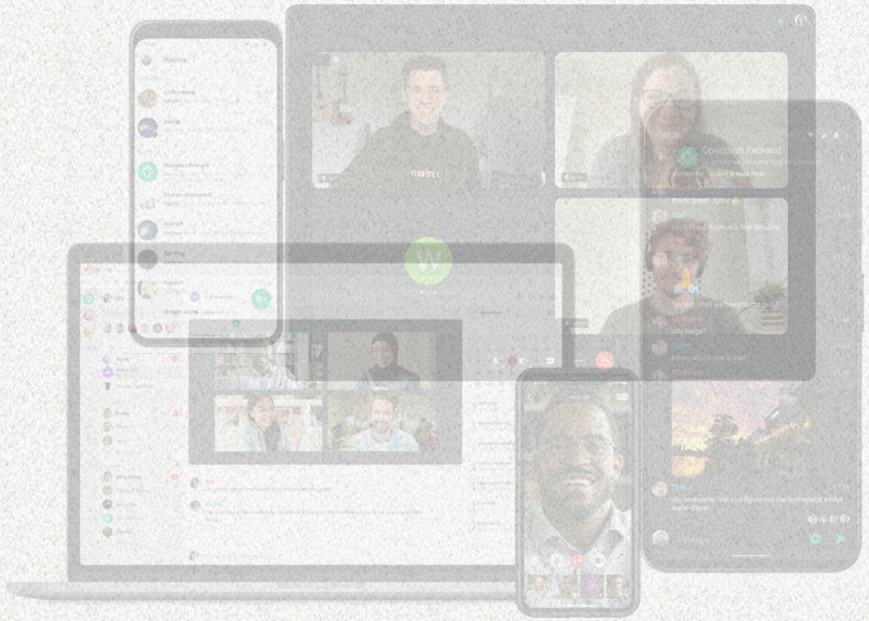
- One major enterprise-grade flagship app

Meta-goal: easy for new devs to fork & modify

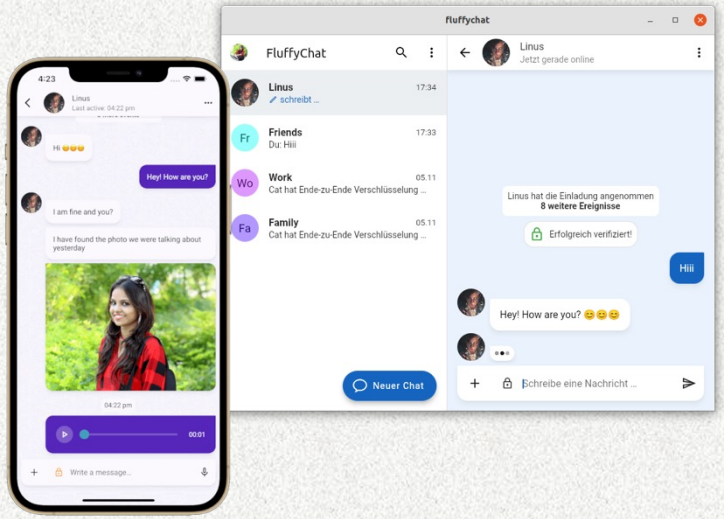
- Well-commented, well-structured code, plus docs

First flagship app – a Matrix chat client

- element: first-party Matrix chat client
  - Separate app and SDK for each platform
  - Leads to buggy, inconsistent UI/UX behavior



- Good existing example: **FluffyChat**
  - Written using Flutter → consistent behavior
  - Still requires some platform-specific code
    - Primarily for building on iOS, Android, web, Windows, Linux, and for different app stores



Advertise Robius's business value

- Attract devs/customers via consistent UI+UX

- Ability to hire fewer devs with deeper expertise
  - Only need to know Rust + framework, not multiple platforms

- Easier maintainability
  - Rapid, consistent bug fixes → happier customers

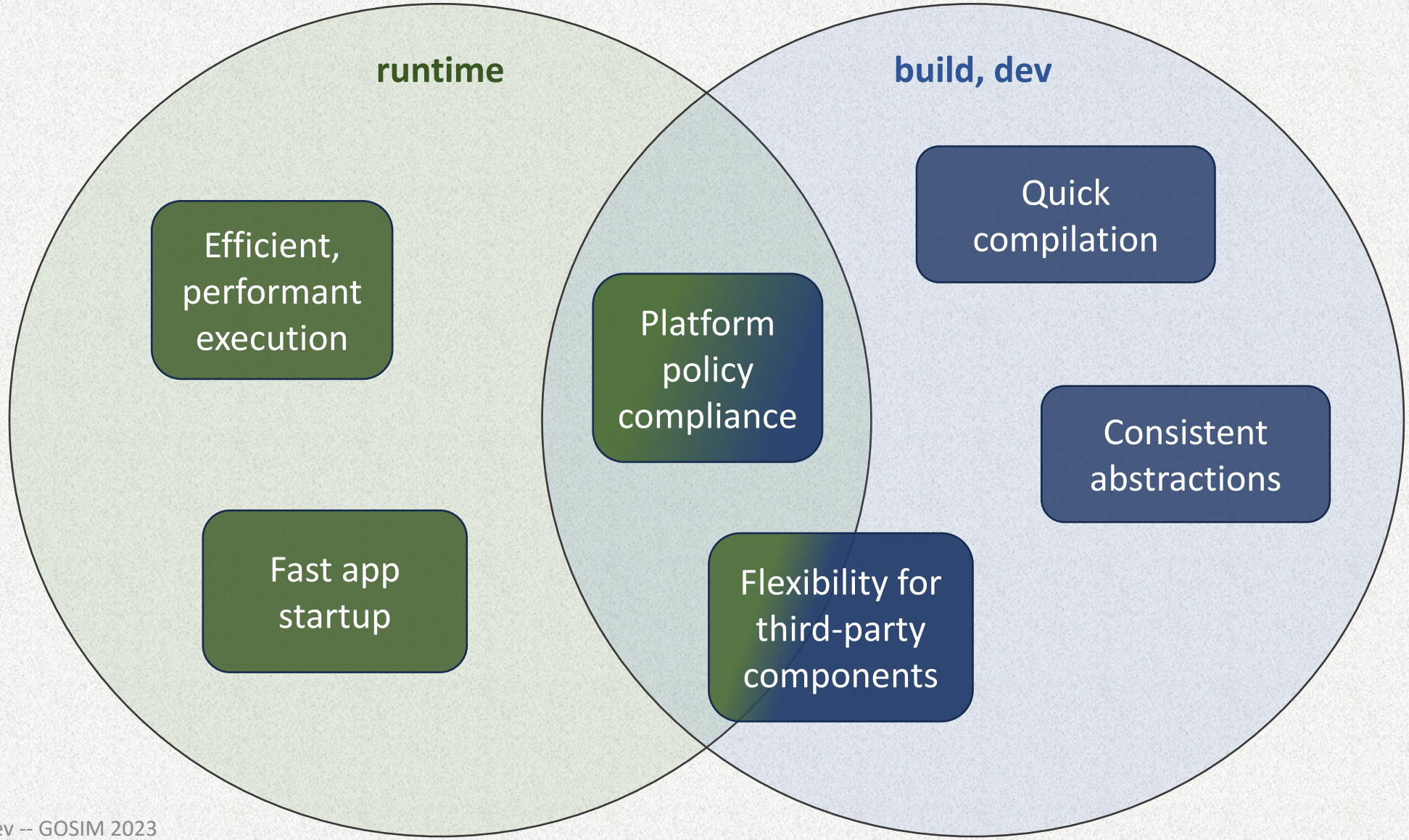
- Smooth integration with Rust's large crate ecosystem
  - Rust FOSS ecosystem is quite deep, especially beyond UI works

- Open-source, distributed, decentralized project structure

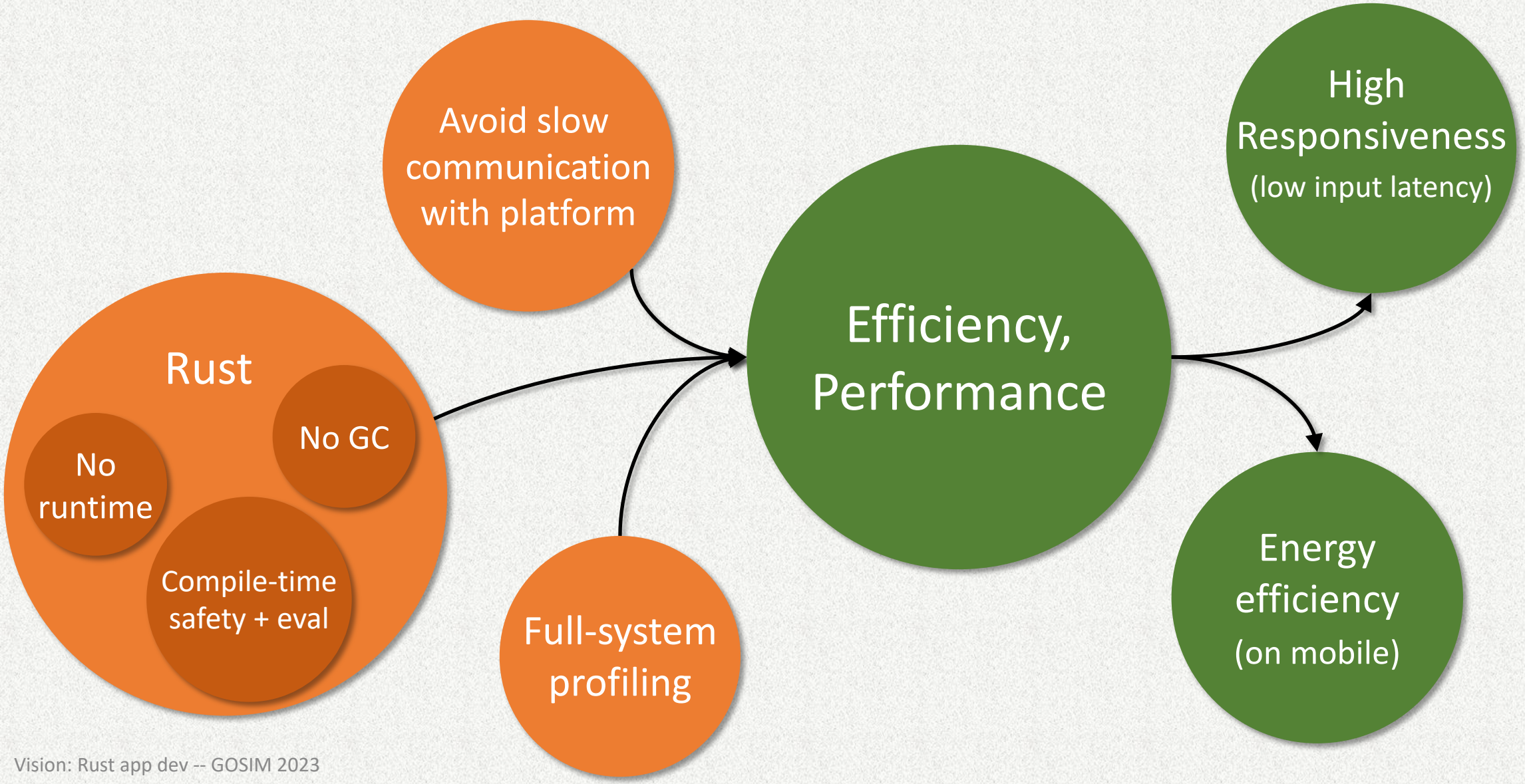
... and you get to use Rust!

Requirements

Technical Requirements



Efficient, performant execution



Fast app startup

- Mobile guidelines: 1.5–2 sec
- Good: Rust AOT compilation
  - No JIT overhead at startup
  - More consistent general perf, lower theoretical max perf
- Bad: large app binary sizes
- Side benefit: helps shorten iterative development cycle

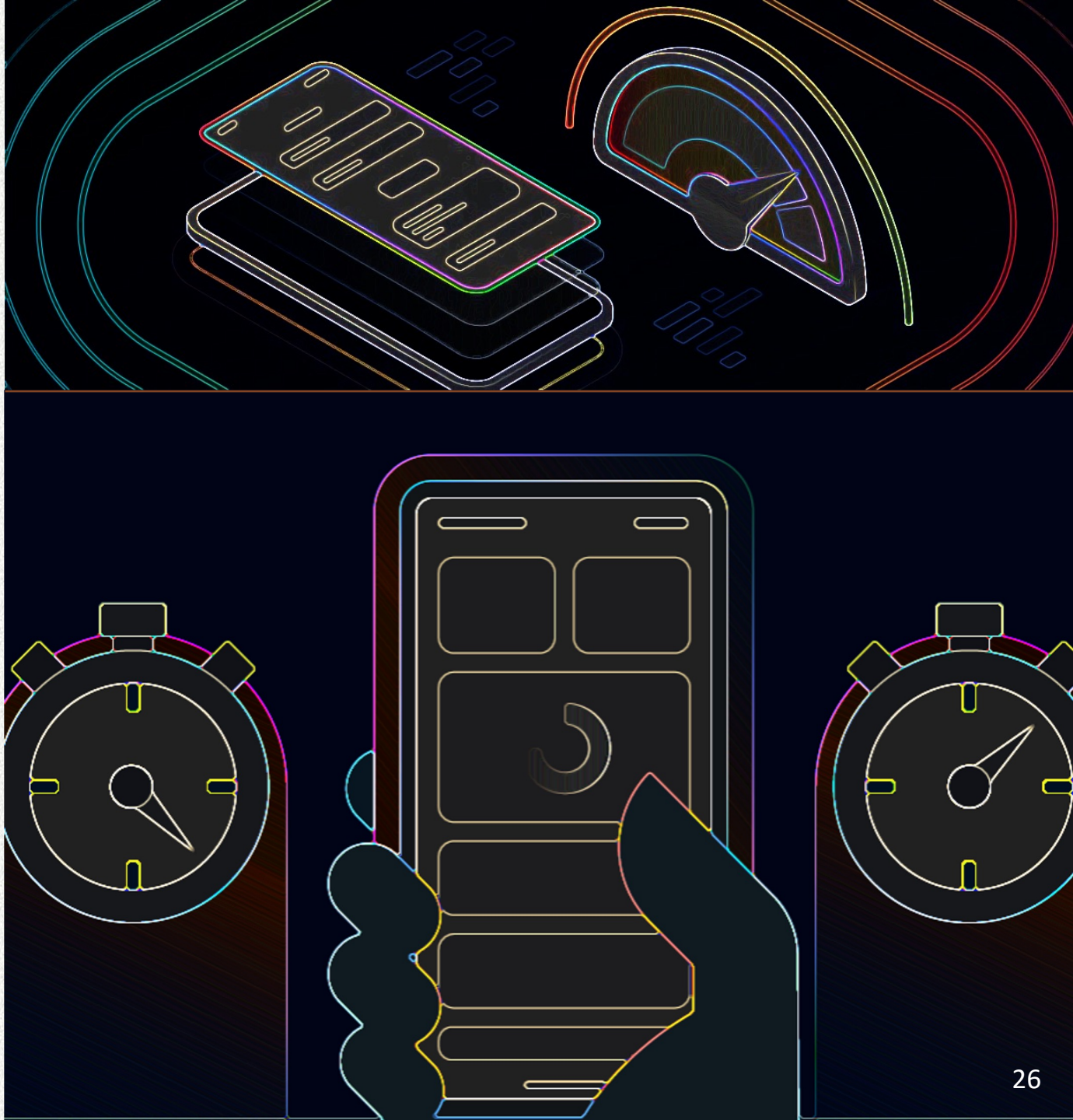


Image source: https://www.instabug.com/blog/app-launch-optimization

Quick compilation

"Rustc is slow" — *the internet*

- Modern Rust UI kits combat this with **live reloading** of DSL elements
- Structure of many small crates
  - Benefit: **parallel** compilation units
- Fast ***incremental*** compilation (at the very least)

Consistency across multiple platforms

Key principle:

- Devs shouldn't be *required* to touch platform-specific code
  - But should be *able* to if desired, with relative ease
- Secondary: platform-agnostic code imposes no overhead
  - Or as close to underlying native platform as possible

Flexibility to incorporate arbitrary components

- Build system and tooling must easily allow adding custom deps
- Initially, support choice of one of multiple UI toolkits
  - Future: replace default components with custom substitutes

Full compliance with platform policies

- Generated app packages must be publishable to app stores
  - ✅ Platform abstractions we provide must adhere to publisher-specific policies
  - 🚫 Private APIs, JIT, loading arbitrary executables

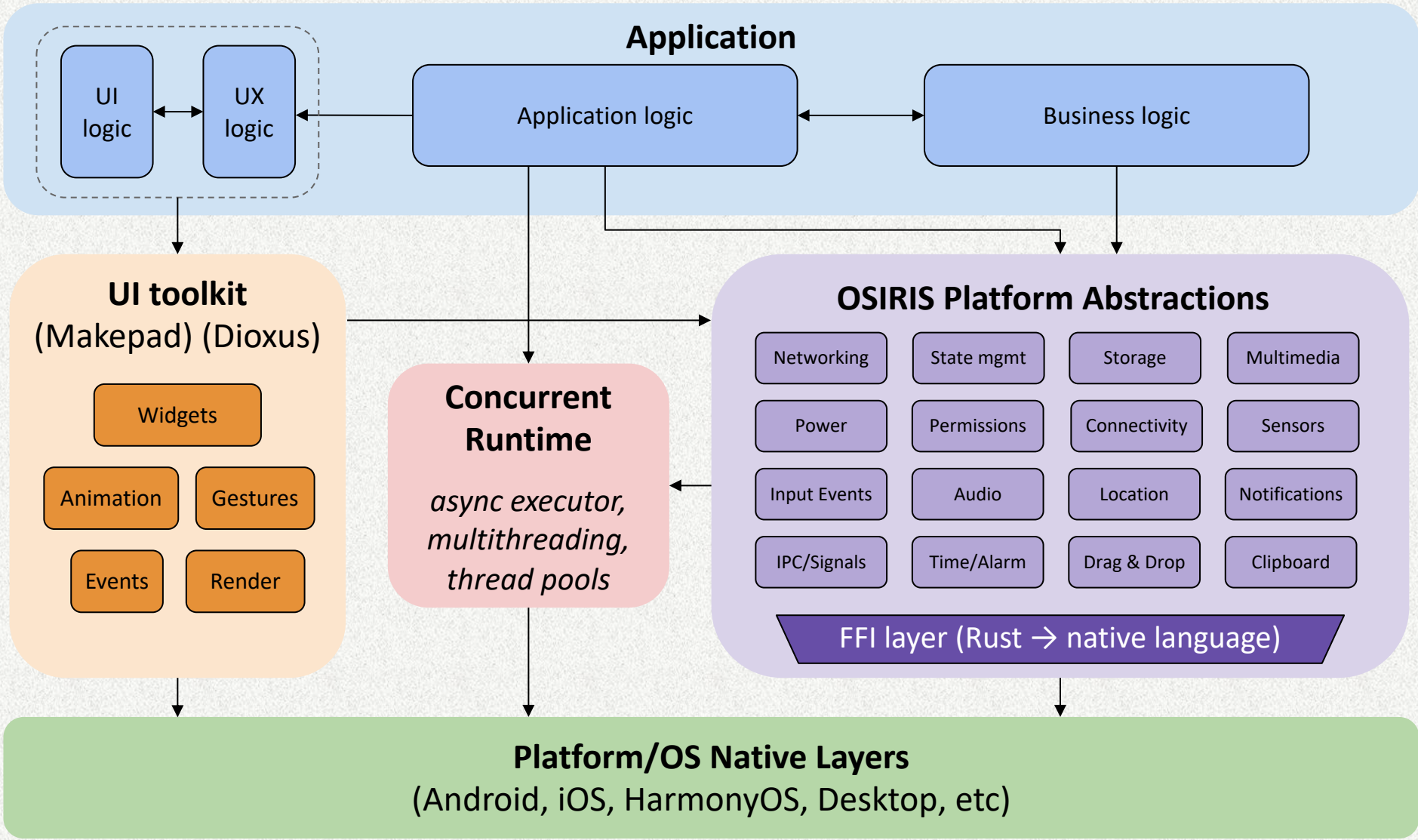
Reference Design

A sketch of system architecture

- Incorporate many components across open-source ecosystem
  - UI toolkits
  - Game engines
  - Abstractions of OS/platform native interfaces & services
  - Concurrency abstractions: async tasking, multithreading
  - [Future] Client-side libraries for remote services (Open Mobile Hub)

- Why call it a “reference design”?
  - Pre-integrated solution for a known-working app dev experience
  - Architecture of loosely-coupled components; reusable elsewhere
    - Advanced devs can tailor stack to meet complex needs

Reference Design of Full System Stack

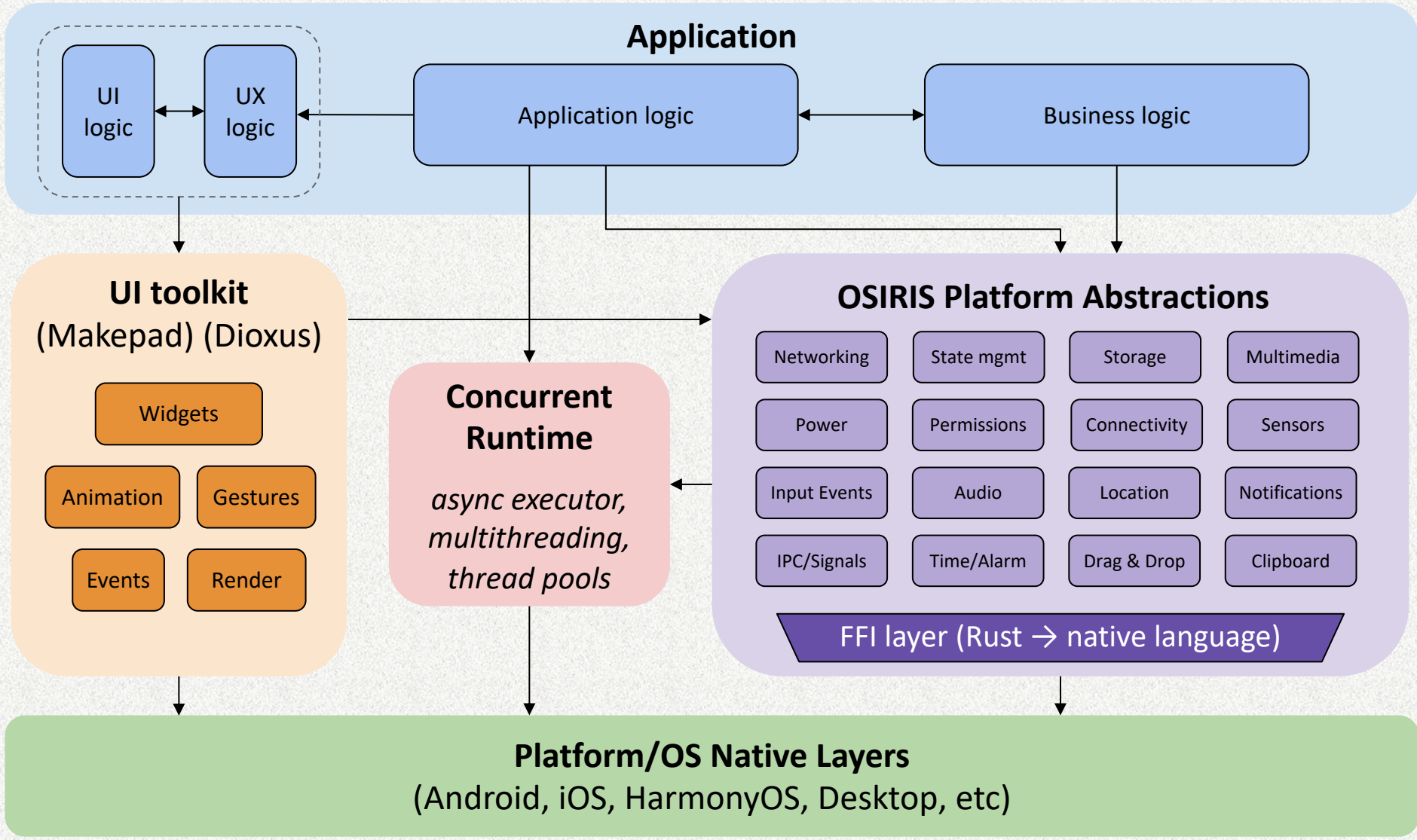


Guiding philosophy: stay out of UI's way

- Let UI experts do what they do best
- UI toolkit + app logic act as active driver

OSIRIS, others are supplementary

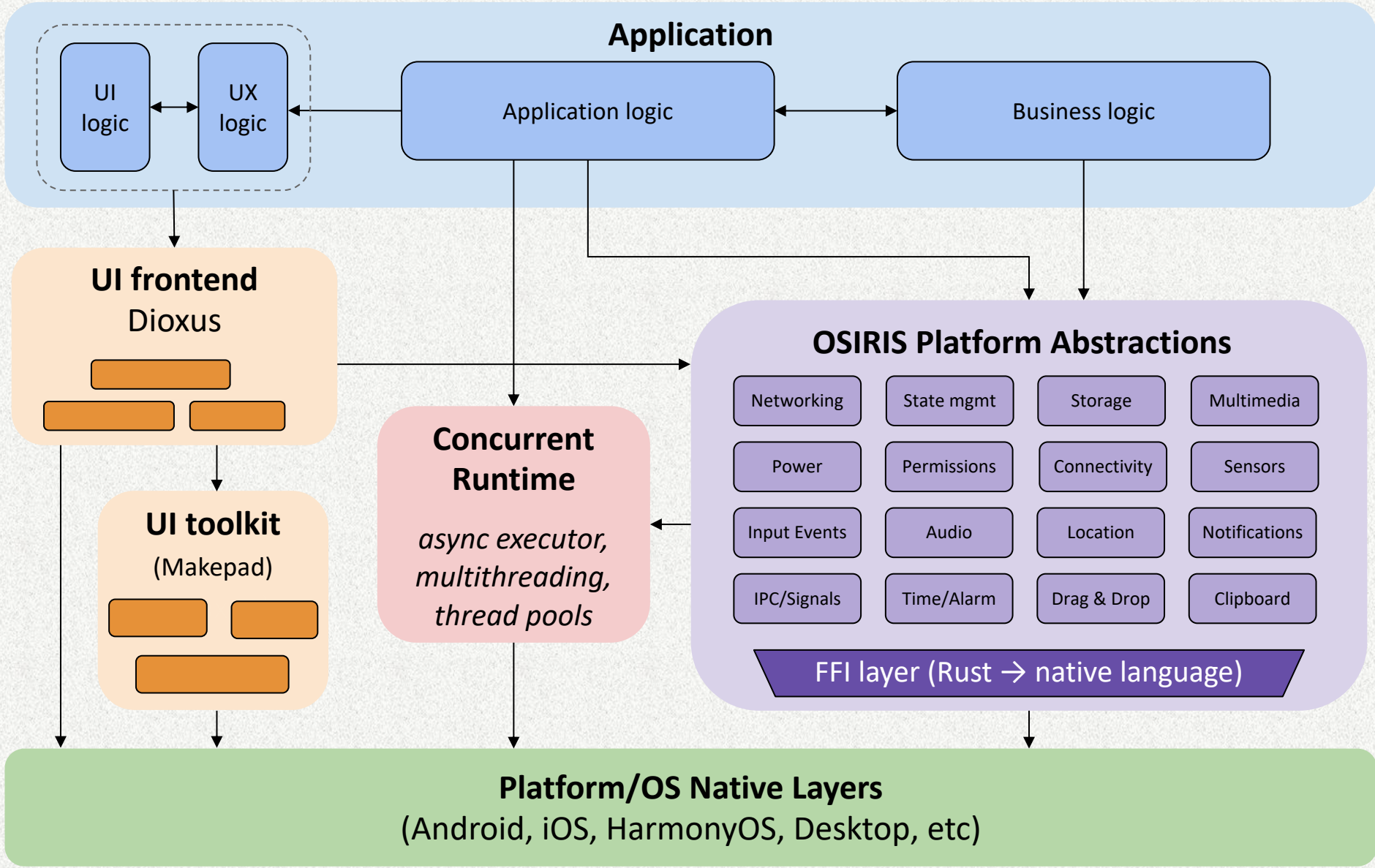
- Independent from UI
- Passive libraries, driven by app



Flexibly integrate multiple UI kits

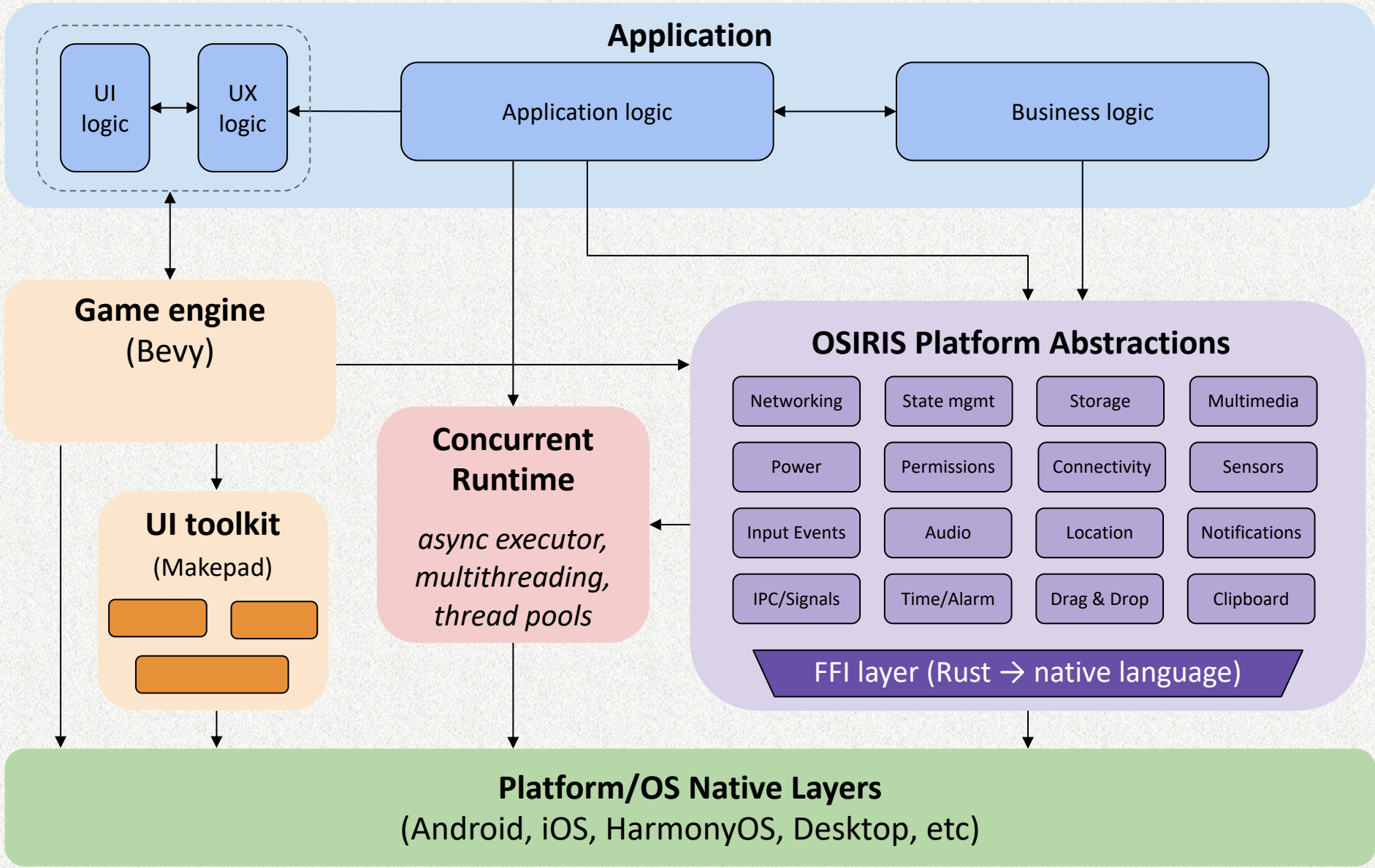
- Devs can choose a familiar UI model
- Challenge: how to compose UI toolkits

→ Help new devs gradually transition to Rust, lower layers



Bring UI and game engines together

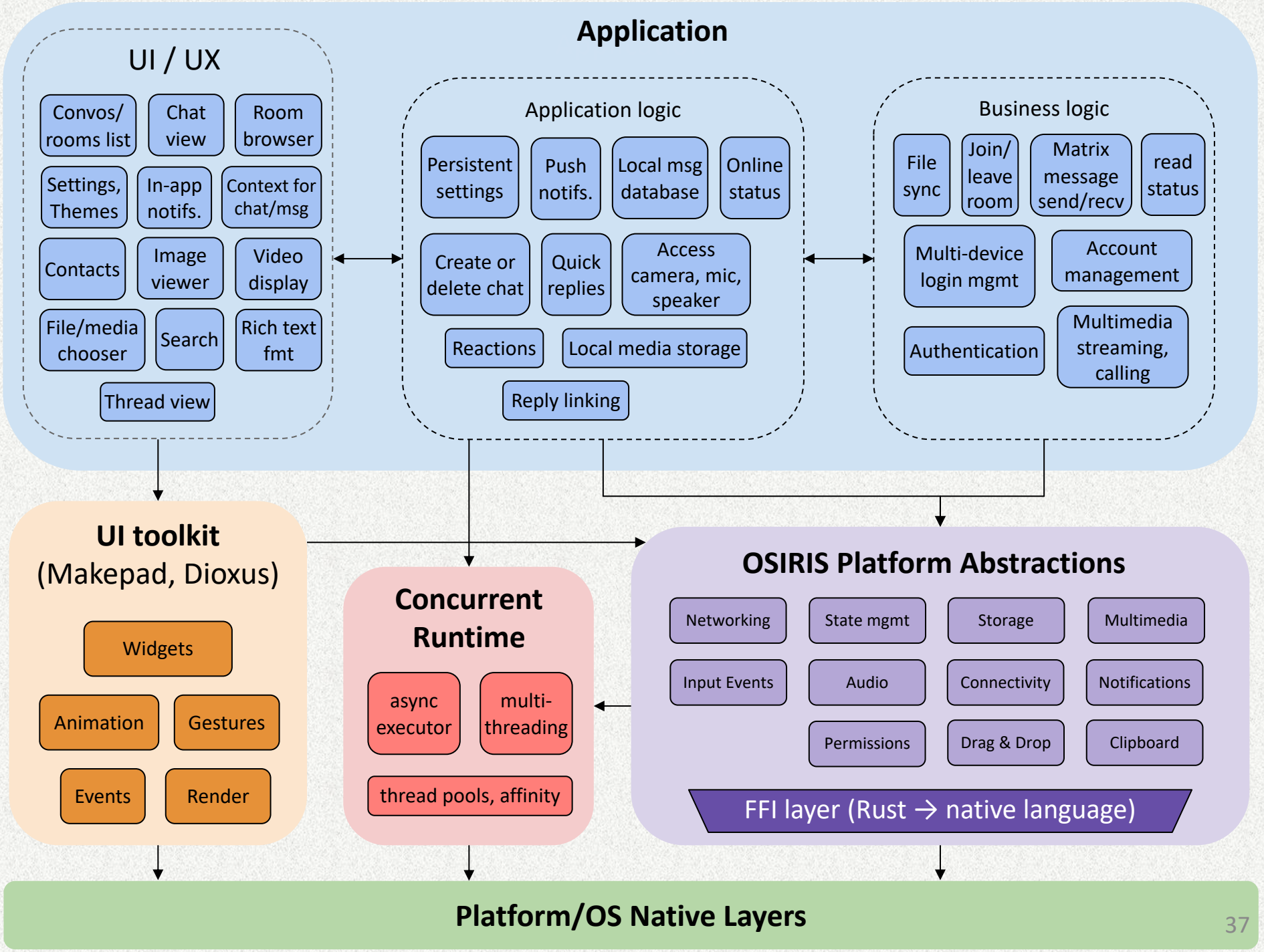
- Game engine can use and drive UI framework
- Incorporate UI features and OSIRIS to simplify non-gameplay function



Flagship App Architecture

(Matrix Chat Client)

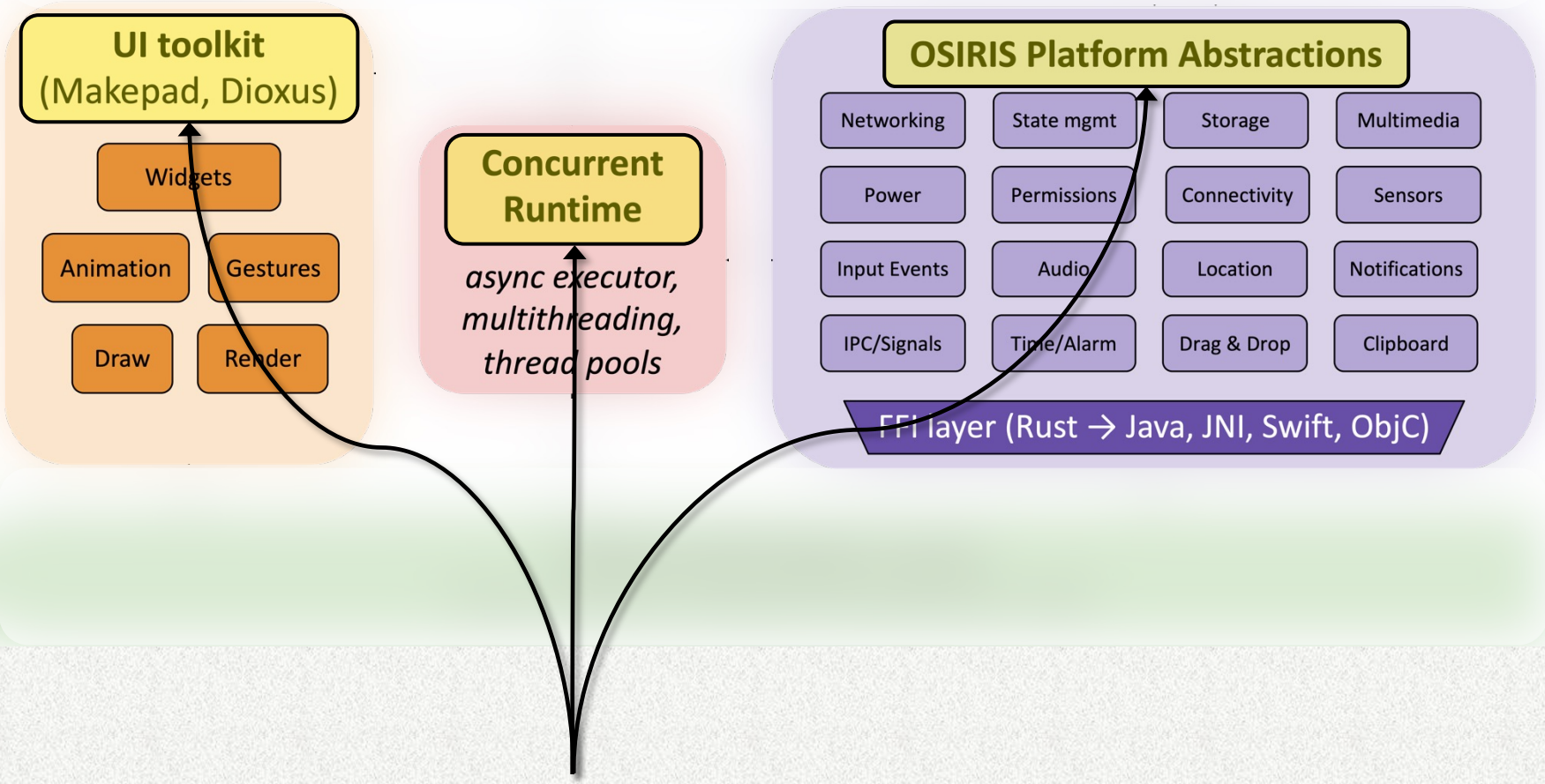
Feature development for this app will steer OSIRIS and UI toolkit integration



Parting thoughts

The path towards Robius

- Newly embarking on this journey
  - Working group announcement coming soon
  - Initial pre-alpha reference design in several months



Detailed status in upcoming talks

GOSIM®
GLOBAL OPEN-SOURCE INNOVATION MEETUP

Thanks!

Interested? Please reach out:

-  @project-robius
-  @kevinaboos
- kboos@futurewei.com

BACKUP SLIDES

Flutter vs. Robius – fundamentally different approaches

- Flutter devs must learn new language, Dart
- Robius is a decentralized, loosely-coupled effort
  - Not owned by a single entity
  - Not a black box like Flutter or React Native
- Fills the vacuum in Rust
  - Efforts already ongoing across many related subdomains in Rust
  - Built in a bottom-up manner to serve Rust community first
    - Rust community usage is our primary concern
  - Architecture supports selection of components specific to your needs
    - Devs can descend into and understand lower layers
      to optimize their apps and the system stack itself accordingly
- No intent to compete with or replace Flutter

Incorporate feedback into Rust/cargo

- Leverage app & framework design experience to improve Rust
- Already in-progress, driven by UI toolkits and system packagers
  - Need better top-level build tool surrounding cargo
  - Need ability to hook into cargo commands, e.g., post-build action
  - Need to inject metadata into build, query it later
  - Better unified configuration space to define build inclusion
    - Outside cargo (e.g., Make vars), within cargo (--features), within rustc (--cfg)
  - Better, more flexible features with saner default-features toggle
  - Existential dependencies

The role of WebAssembly (WASM)

- We *may* be able to leverage WASM for easier deployment
  - Mostly for apps on the web, plus Desktop platforms
  - Mobile app stores disallow downloading and running arbitrary code
- Rust has first-class support for targeting WASM
  - Most of the WASM ecosystem prioritizes Rust integration
- Component model can help mitigate large-sized app binaries
  - Delivery over the web could only transfer app binary
  - Re-use libraries, shared components already on the client