Seamless Multi-Platform App Development in Rust

Kevin Boos
Principal Architect, Futurewei

May 8th, 2024

1. Create a multi-platform app dev experience fully in Rust

Devs can:

- leverage the robust, safe, and performant Rust ecosystem
- avoid dealing with multiple languages/environments
- never write a single line of platform-specific code

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

Robius at a glance

Background: Project Robius

- Fully open-source, decentralized, and community-driven
  - Independent collaborators across US, Europe, Aus/NZ, S. America, China
- All components written entirely in Rust

But why Robius? … there are tons of great Rust UI toolkits out there

- egui, Iced, Slint, Tauri, Leptos, Dioxus, Makepad, Xilem, Freya, Ribir, …

→ There's **way more to app dev** than just drawing a GUI!

- Many existing Rust apps are "less Rust, more other code"
  - Big platform-specific wrappers around a small Rust core

Motivation

What motivates our work?

1. App devs want to use Rust, but aren't sure where/how to start
   - Paralysis of choice among many offerings
   - Unclear how to integrate a bunch of independent projects
2. Clear business advantages
   - Consistent multi-platform 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 devs to the world of Rust (and vice versa)

What are the best GUI libraries for a potentially “serious”/large project?

I’m thinking of trying to make a binary reverse engineering and static analysis tool, and I need a GUI library for it. GTK was my first thought, but does anyone have any other suggestions?

(Non-)constraints:

1. Accessibility at the GUI framework layer won’t help this kind of program, so that’s not an issue.
2. I need at least a way to display syntax-highlighted text, and preferably the ability to create completely custom widgets.
3. Should be cross-platform, if that isn't implied.

steep learning curve. Our use-case is an embedded GUI and Rust is so much more complex than the conventional front-end languages (javascript, C#) we used before, with no discernible performance upsides and a much more immature and limited ecosystem of library and tooling support. I think

460 Reply Copy link

Why is building a UI in Rust so hard?

What would be the default gui library to choose ?

seeking help & advice

Hi !

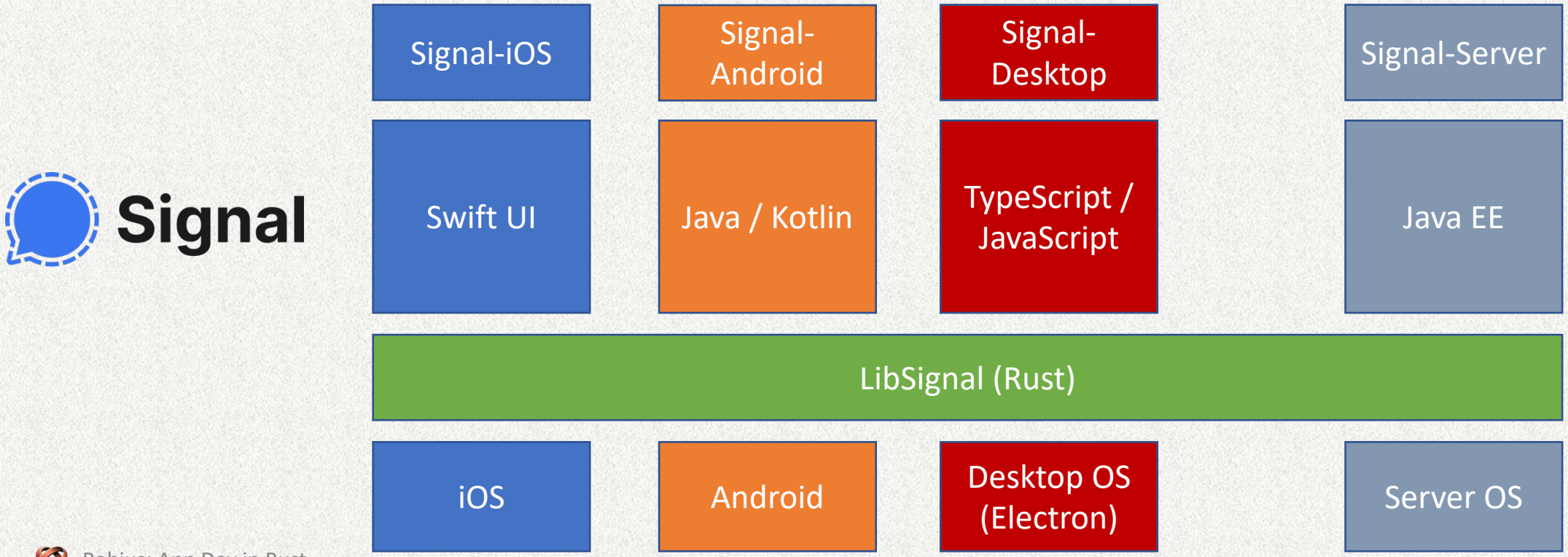
I'm trying to choose a gui library for my project, but after many researches, I'm just confused. There is a lot of parameters to consider, and I don't even know where my project would lead me.

Is there anything similar to that going on in Rust? Any promising ones? Thanks.

ok, ok, we get it

State of mainstream app dev in Rust

- Apps developed using Rust have platform-related pain points
  - Typically use Rust for core business logic only
  - And/or use thin bindings/bridges to other native UI toolkits



State of mainstream app dev in Rust

- Signal maintains 3 separate app projects, with little overlap

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

iOS

450K LoC

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	9	78	47	477
HTML	10	18	25	392
Bourne Shell	3	16	14	62
SUM:	1514	33811	10830	1131336

Desktop

250K LoC

http://cloc.sourceforge.net v 1.64 T=82.82 s (76.2 files/s, 12097.5 lines/s)

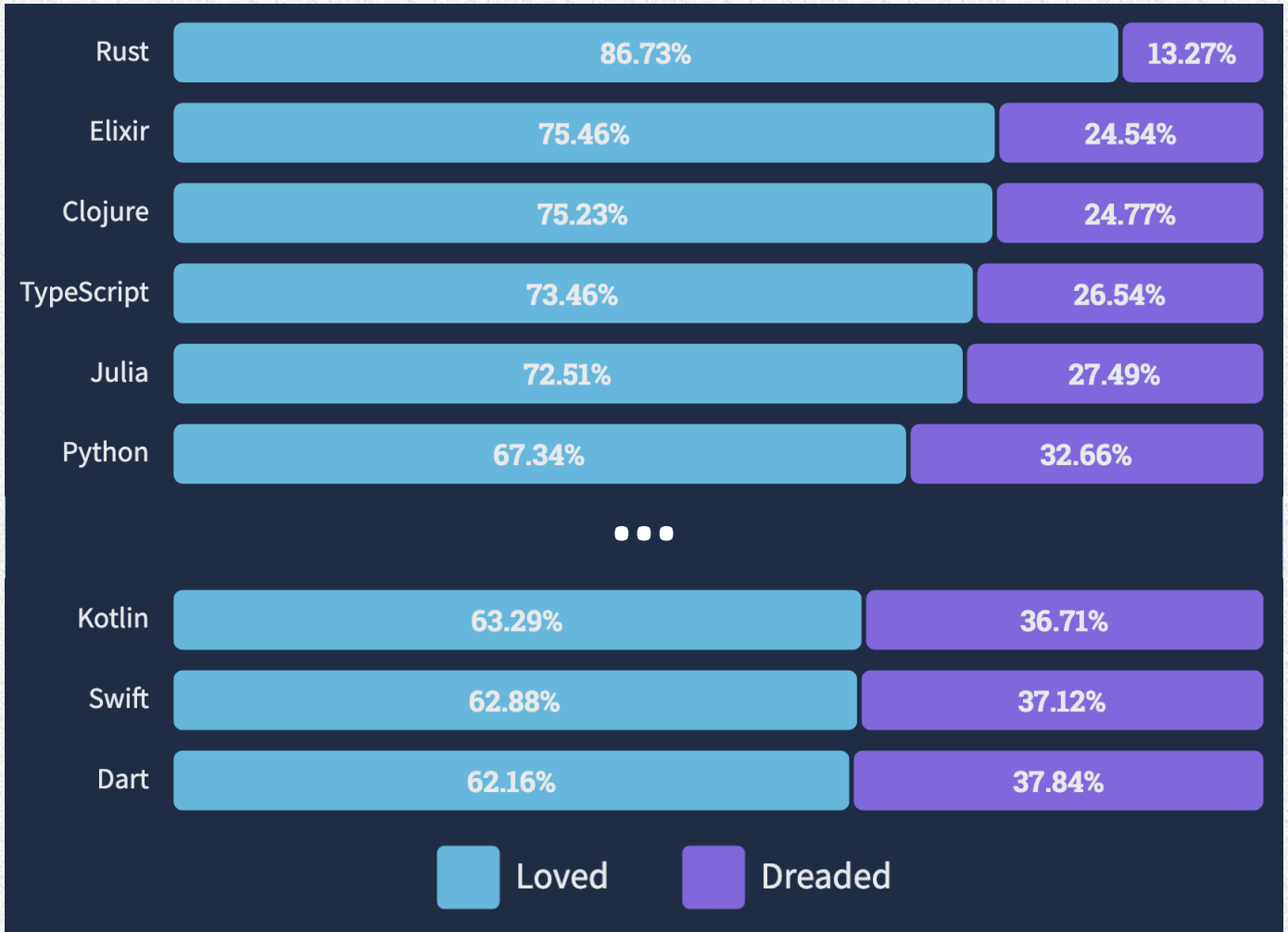
Language	files	blank	comment	code
XML	2113	47590	108376	334431
Java	2483	60351	14490	250763
Kotlin	1573	25629	7564	142292
Protocol Buffers	21	383	163	1884
Groovy	32	278	21	1556
JSON	45	0	0	722
Handlebars	9	55	9	359
Python	2	34	7	147
Bourne Shell	2	28	108	105
IDL	17	13	0	87
CSS	1	18	0	80
YAML	5	20	6	79
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

Android

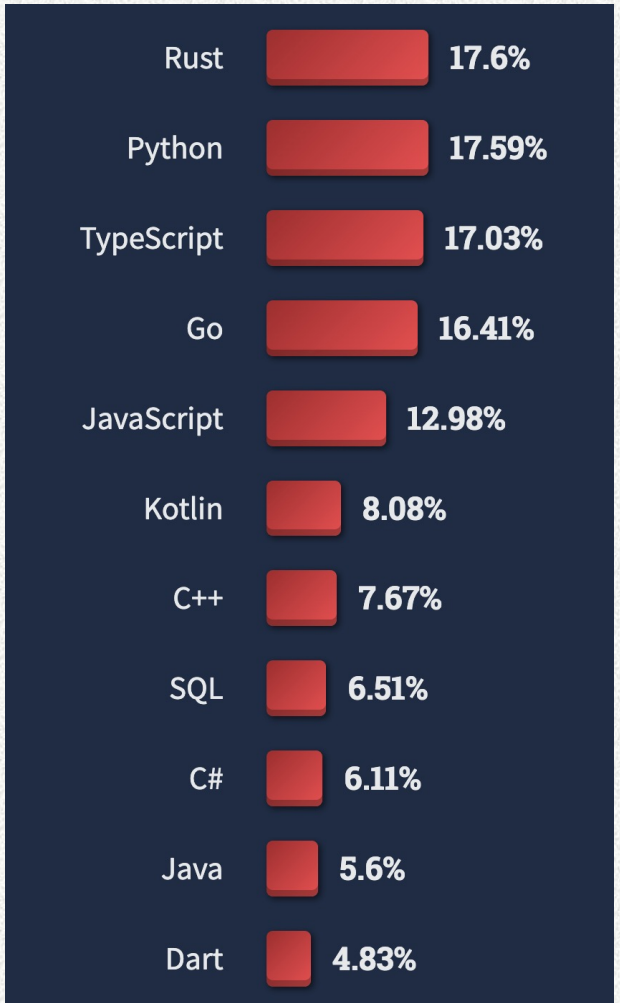
350K LoC

Why pure Rust? Rust itself is the right choice

Most loved language 8 years in a row



... and most wanted



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

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 … (see Nick Nethercote’s work)

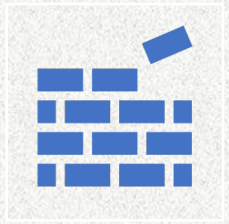
- Potentially steep learning curve for frontend devs

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

-- VERY GOOD VENTURES

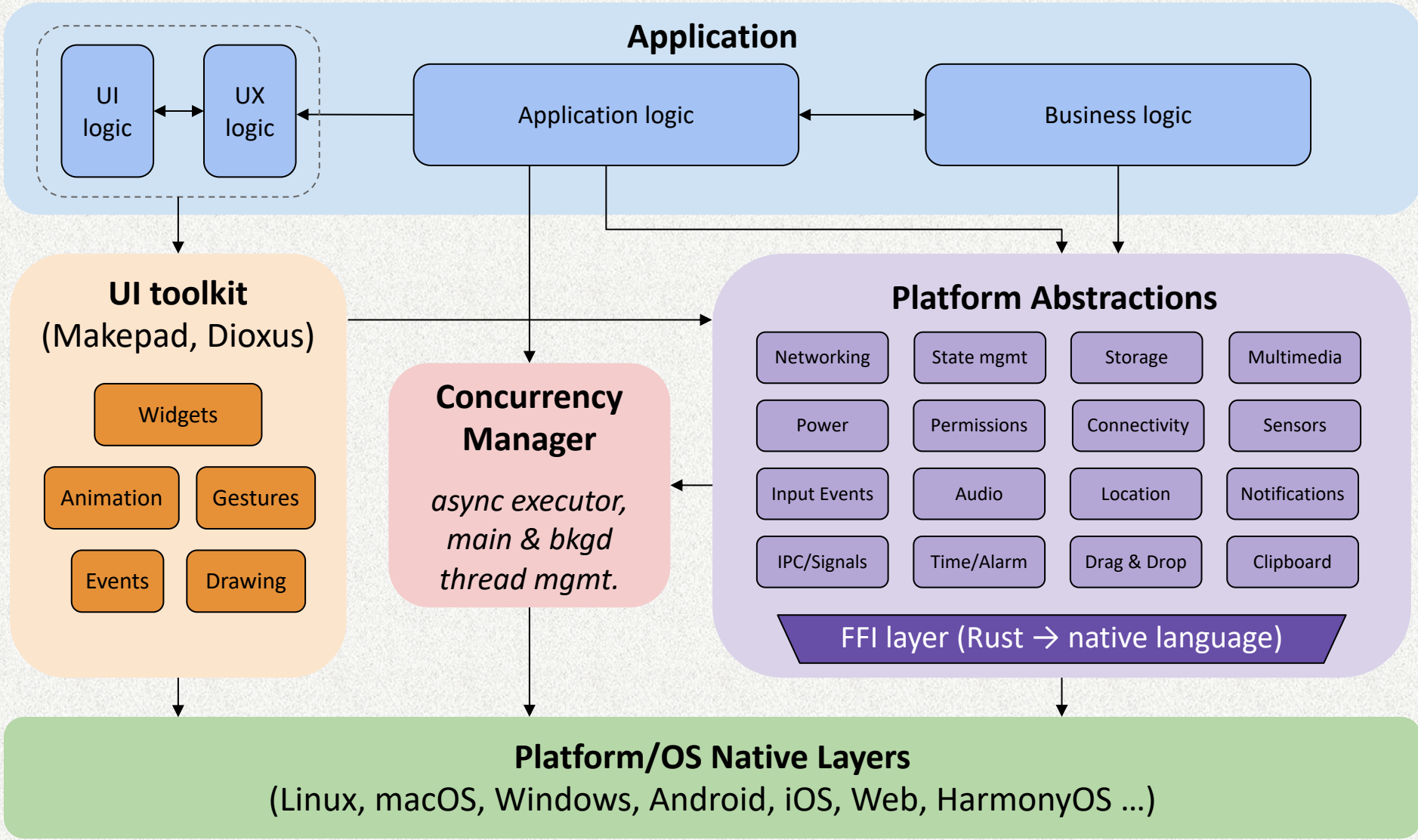
Source: https://verygood.ventures/whitepaper/business-value-of-flutter

Goals (selected)



Reference design of full system stack

runtime components

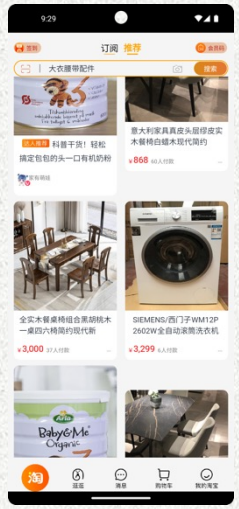


Proof-of-concepts demos + flagship apps

1. A series of ready-to-use simple example/demo apps





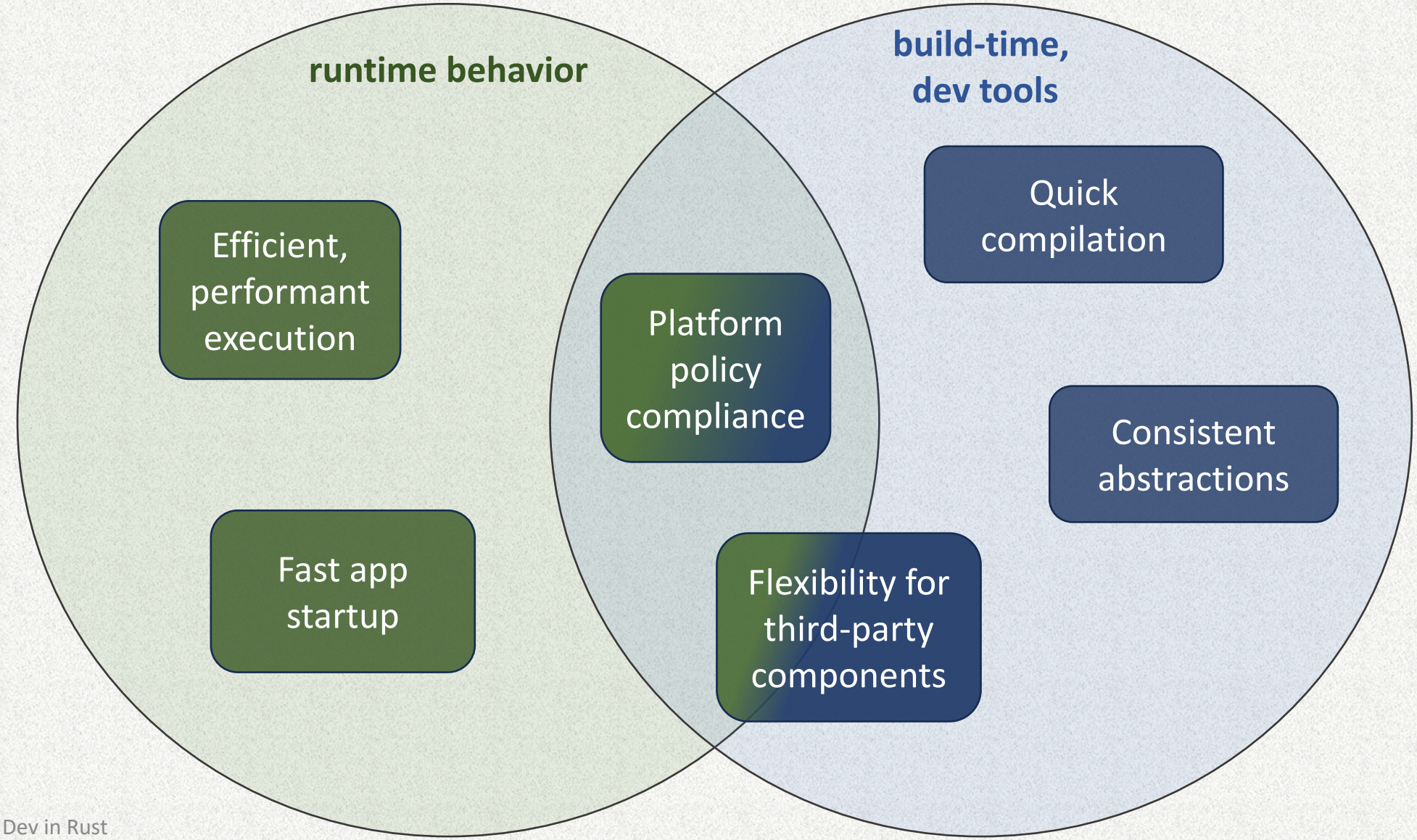



2. A few major commercial-grade flagship apps

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

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

Select technical goals



Quick compilation

"rustc is slow" — *everyone*

- Modern Rust UI kits combat this with **live reloading** of DSL elements
- Structure of many small crates
  - Benefit: **parallel** compilation units
  - **Minimize** dependencies ruthlessly
- 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 principle:

- Platform-agnostic code imposes no overhead
  - As close to underlying native platform as possible

Major Challenges

Significant systems challenges

- Exposing & abstracting platform APIs / OS services
- Handling complex multi-step builds
  - Rust compilation: invoking & configuring Cargo
  - Standardizing compilation of non-Rust system glue layers
  - Supporting platform-specific toolchains and linking conventions
  - Defining and executing pre- and post-compilation steps
- Signing, bundling/packaging, distributing apps
  - Required for displaying notifications (and so much more)

all across disparate platforms

- Managing concurrency: multi-threading, async/sync, hetero CPUs…
  - UI & other select workloads must run on “main” thread
  - Certain APIs *require* async, others *forbid* async
- Contending with multiple *vastly* different UI toolkits
  - Platform support crates must be easy to integrate with UI system internals

for more info, see our upcoming joint whitepaper:

The state of Rust GUI systems

(being organized by Nico Burns)

Project Architecture

Project Robius: a diverse community

Key community projects:

- **Robius**:
  - overall integration & app dev
  - platform feature abstraction crates
- **OSIRIS**:
  - signing, bundling/packaging, distribution (wrapping cargo)
  - OS service wrappers/FFI crates
- **Quake-build**:
  - meta-build tool (à la `make`)
  - orchestrate pre- & post-compile steps
- **Others** (me)
  - concurrency management
  - generic caching of widgets (populating & drawing UI data)

Targeted UI toolkits:

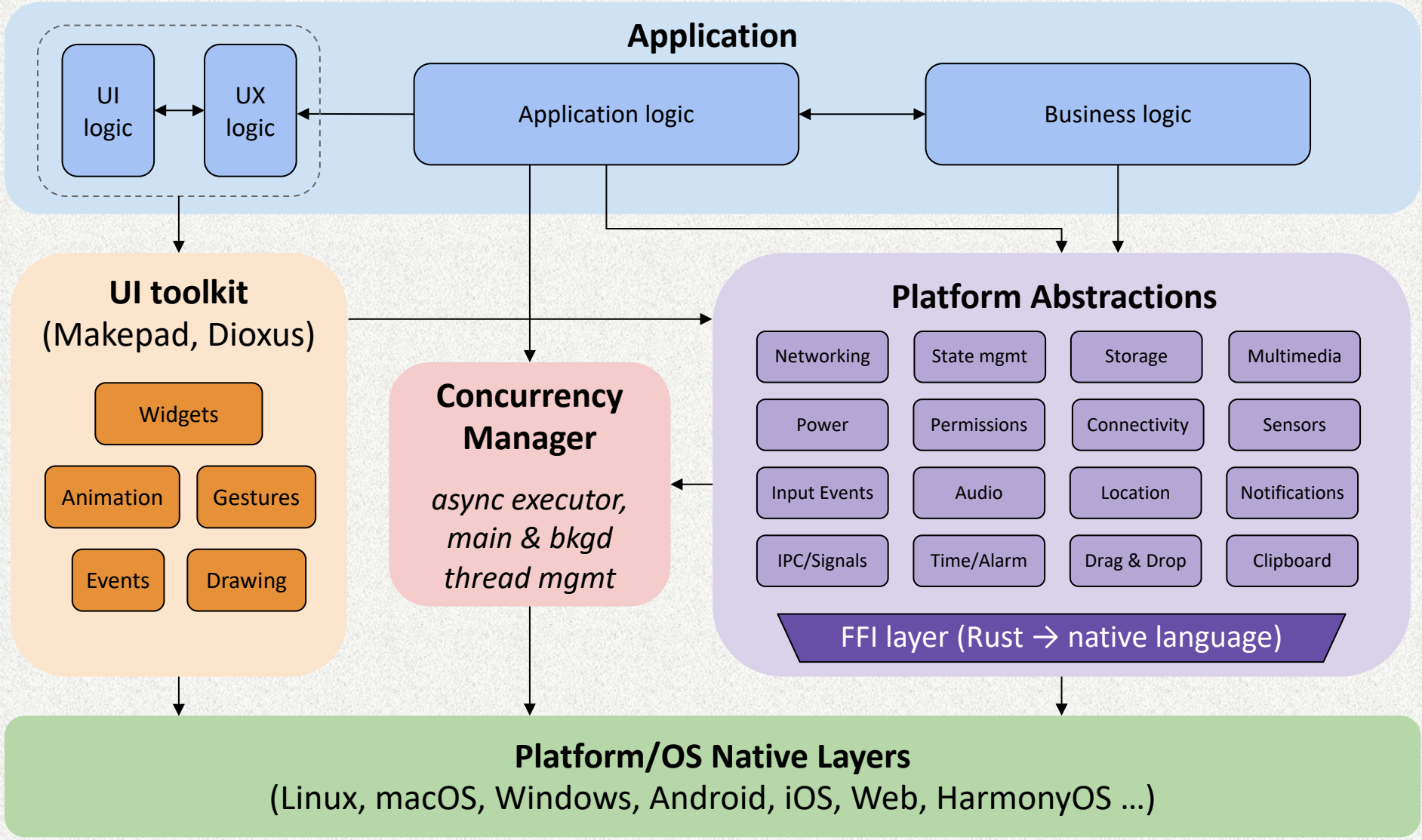
1) **Makepad**
   - Close daily collaboration
   - Robius makes UI & system contributions back to Makepad
2) **Dioxus**
   - Goal: fill gaps in `dioxus-std`
   - Acts as diverse point in space of integration implementation
3) **Linebender project** (Xilem, etc)
   - Periodic exchange of ideas
   - Co-leverage Android knowledge
4) **Tauri**
   - Utilize existing desktop crates
   - Offer mobile crates back (like Dioxus)

Guiding philosophy: stay out of UI's way

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

Robius, OSIRIS, etc are supplementary

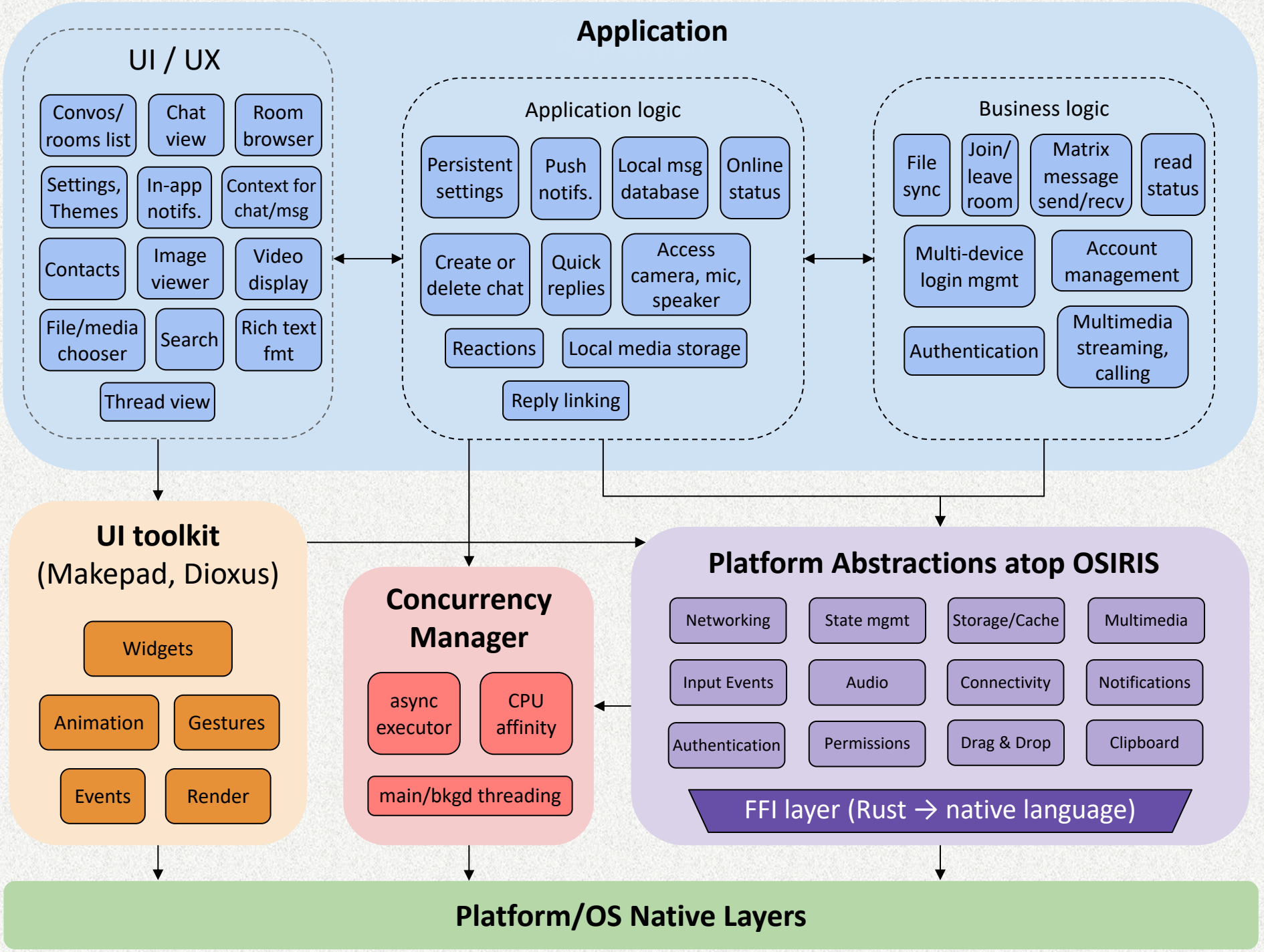
- Independent from UI
  - Offer easy integration w/ supported UI toolkits
- **Passive** libraries driven by app



Robrix App Architecture

(Matrix Chat Client)

Robrix's needs determine feature development and implementation priority



Targeted platform abstractions / OS service APIs

- System notifications
- Biometrics & authentication
- File/image picker dialogs
- Context menus (OS-native)
- Menu bar (desktop only)
- Rich clipboard
- Drag & Drop
- Default URI/Intent handling
- Content sharing to foreign app
- Native font access
- Input Method Editors (IME)
- Native gesture recognition
- Connectivity management
  - WiFi, Bluetooth, NFC, nearby devices
- GPS/Location access
- Multimedia capture
  - Device discovery & configuration
  - Camera – snapshots, video, settings
  - Audio – input, MIDI, playback
- Sensors
- Native alarms, timers
- Keychain (secret storage)
- Power management/status
- App lifecycle state transitions

and so much more …

Status — just getting started, lots to do!

- System notifications
- Biometrics & authentication
- File/image picker dialogs
- Context menus (OS-native)
- Menu bar (desktop only)
- Rich clipboard
- Drag & Drop
- Default URI/Intent handling
- Content sharing to foreign app
- Native font access
- Input Method Editors (IME)
- Native gesture recognition
- Connectivity management
  - WiFi, Bluetooth, NFC, nearby devices
- GPS/Location access
- Multimedia capture
  - Device discovery & configuration
  - Camera – snapshots, video, settings
  - Audio – input, MIDI, playback
- Sensors
- Native alarms, timers
- Keychain (secret storage)
- Power management/status
- App lifecycle state transitions

and so much more …

Simple philosophy:

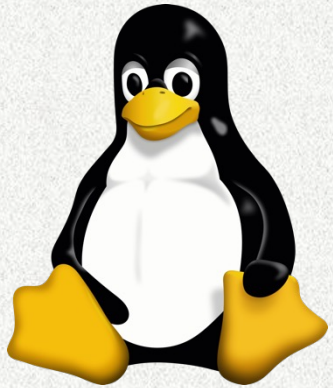
1. Native-first, or as close to it as possible
2. Little to no (avoidable) dependencies

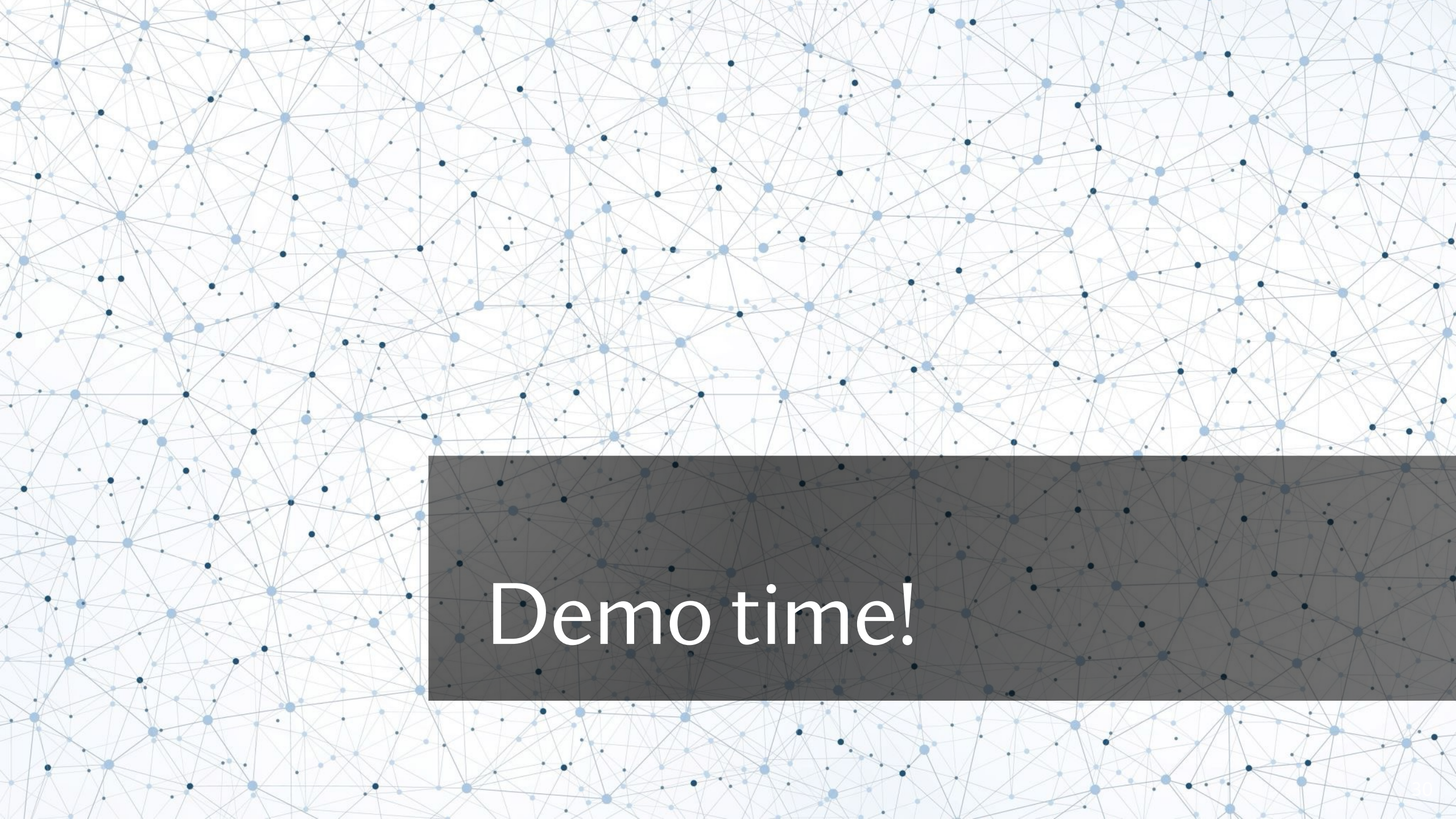
Platforms of interest

- Desktop
  - macOS
  - Linux (primarily Debian-based)
  - Windows
- Mobile
  - Android
  - iOS
  - [Future] OpenHarmony OS
- Web, wasm
- Others?
  - Select microcontrollers/peripherals
  - tvOS, watchOS

WEBASSEMBLY

OpenHarmony




Demo time!

Demos across multiple platforms

Flagship apps

- Robrix – Matrix chat client
- Moxin – run LLMs locally (à la LM studio)

Simple apps

- `robius-demo-simple`
  - Shows platform features: authentication, opening URIs
- Wonderous copycat app

Libraries & Infrastructure

(Examples from Android)

- `android-build`
  - Build Java source for Android from a Cargo build script
- `robius-android-env`
  - Abstraction for hassle-free access to UI toolkit-owned Android state

<demo>

Demo overview for future offline viewers

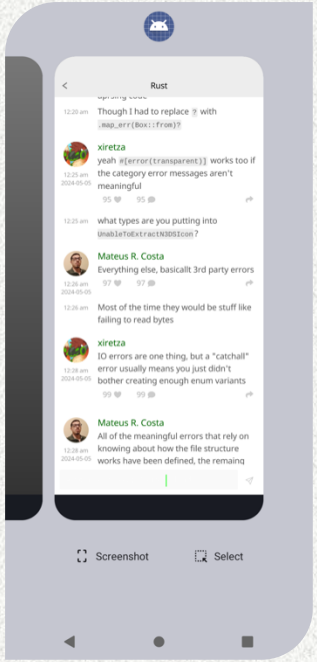
- Demo examples of robius-android-env
  - Plus a before & after of robius-authentication/robius-open using it
    - And also robius-demo-simple using it (previously had platform-specific code for android only, now have nothing)
  - Mention how we strive to remove the burden of implementing glue code from *both* the app developer and the UI toolkit maintainer
    - It should “just work”

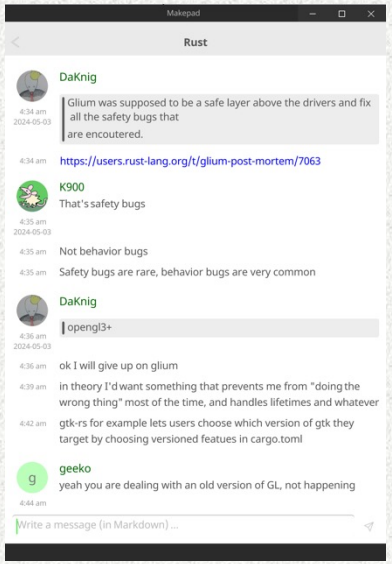
Demo overview for future offline viewers

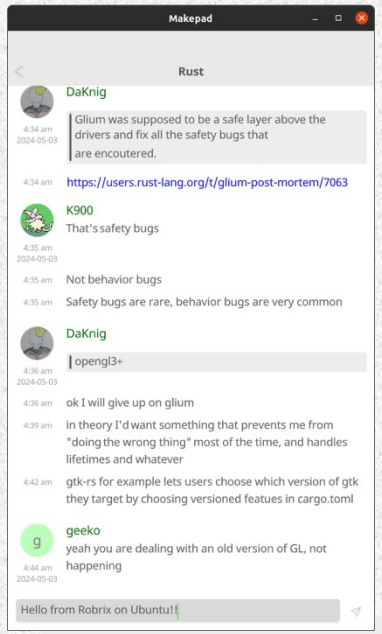
- Demo of how to use android-build, similar to `cc-rs`
  - Plus a before & after of using it in robius-authentication's build.rs script (showing how much simpler it is)
  - Mention that lots of work went into it to ensure it works on all host OS platforms (Linux, macOS, Windows)
  - Convenience features like auto-detection of the Java home directory and your Android SDK installation
  - Establishes standardized env vars for overriding/customizing build behavior
  - Future: support defining more parameters via cargo metadata

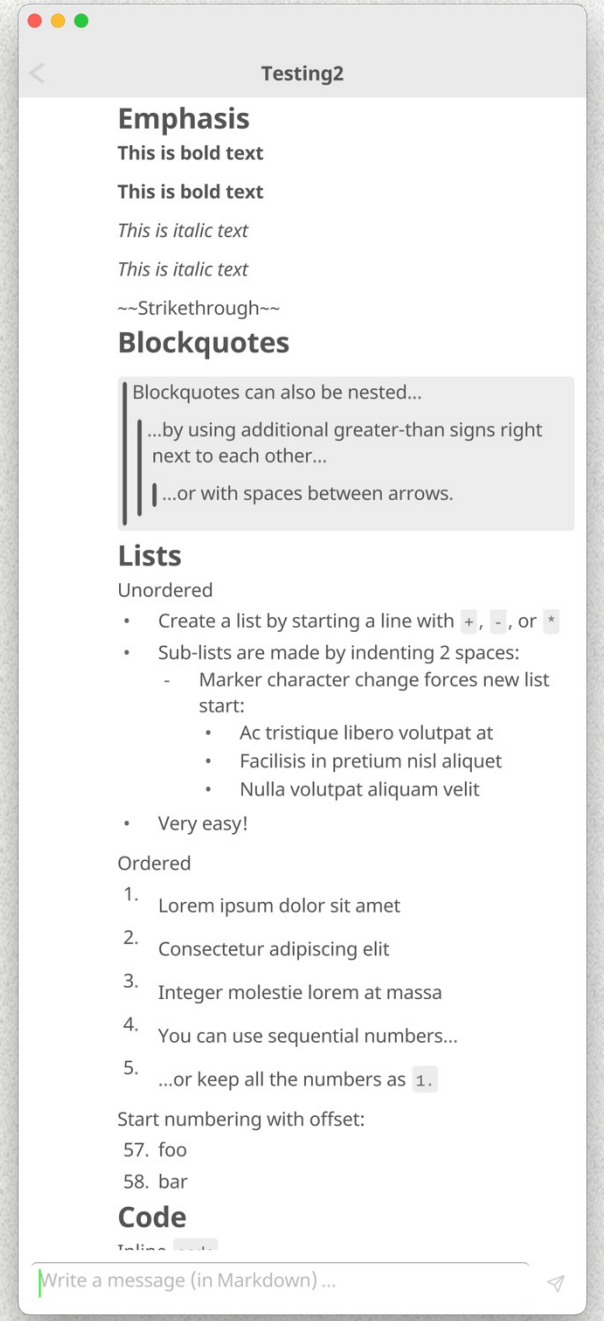
Robrix

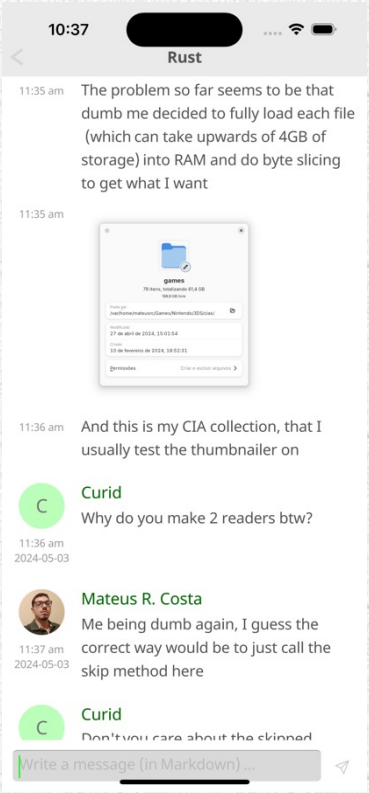
One code base, many platforms

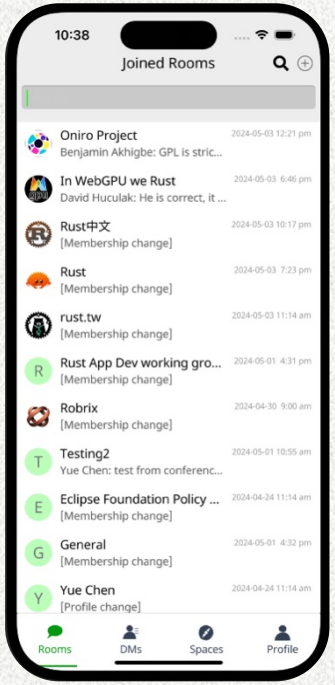


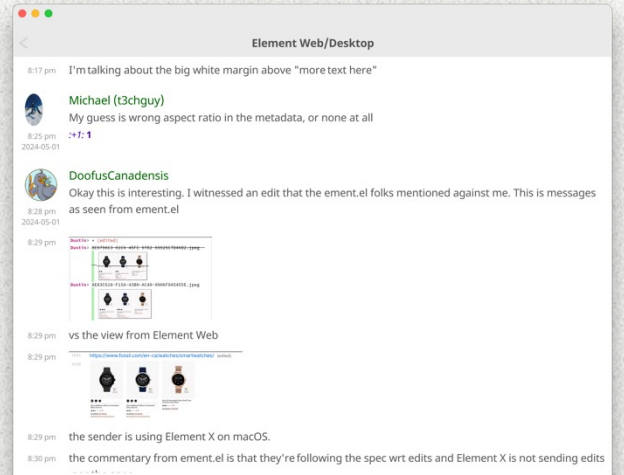


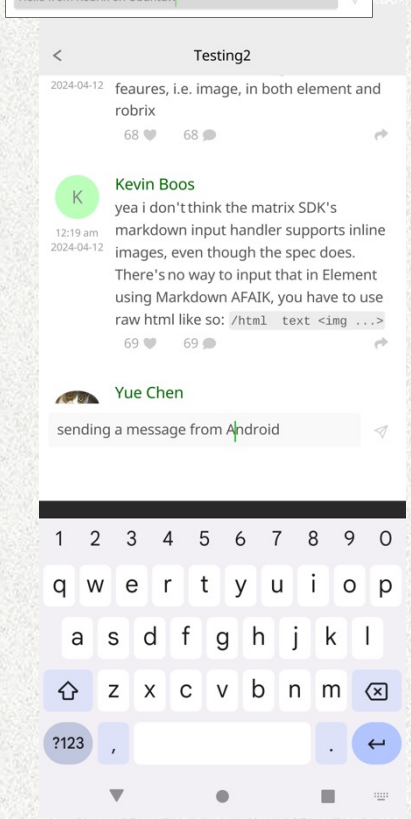




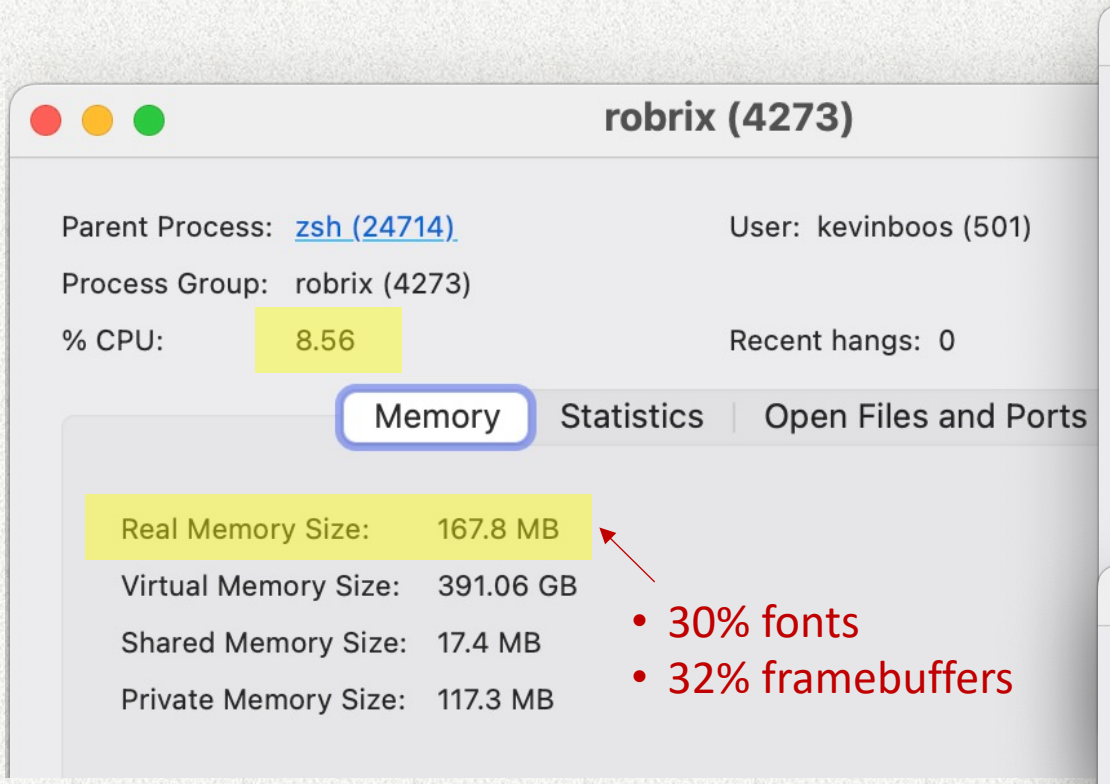




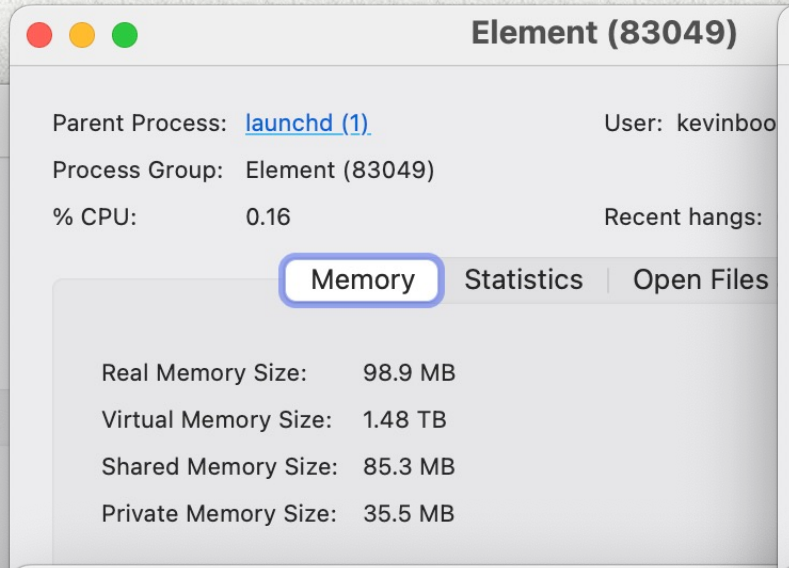


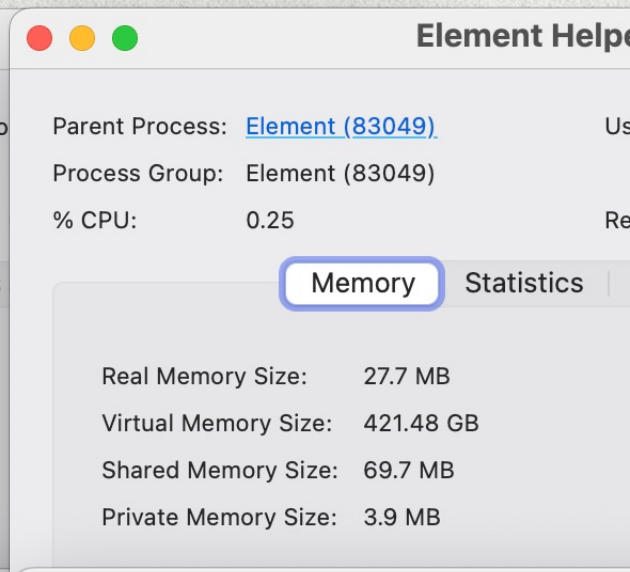


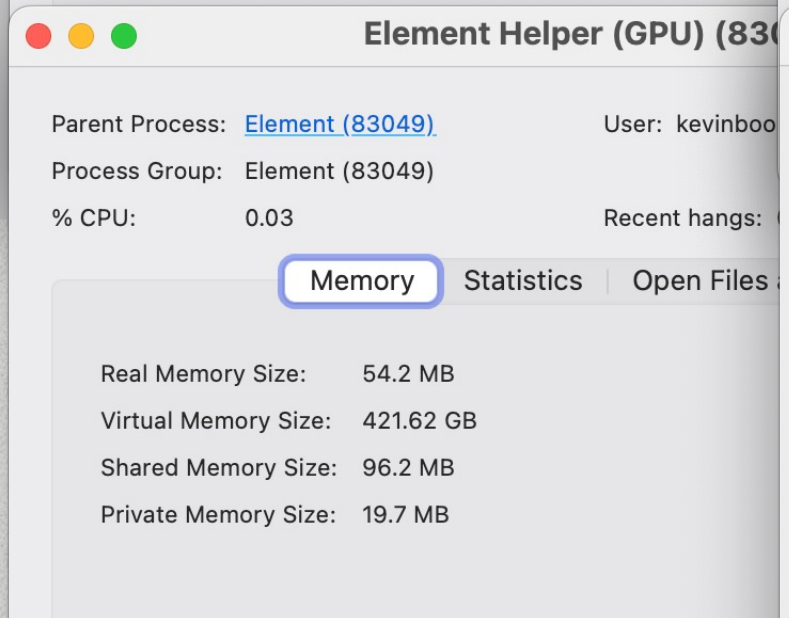
Makepad + Robius stack is efficient

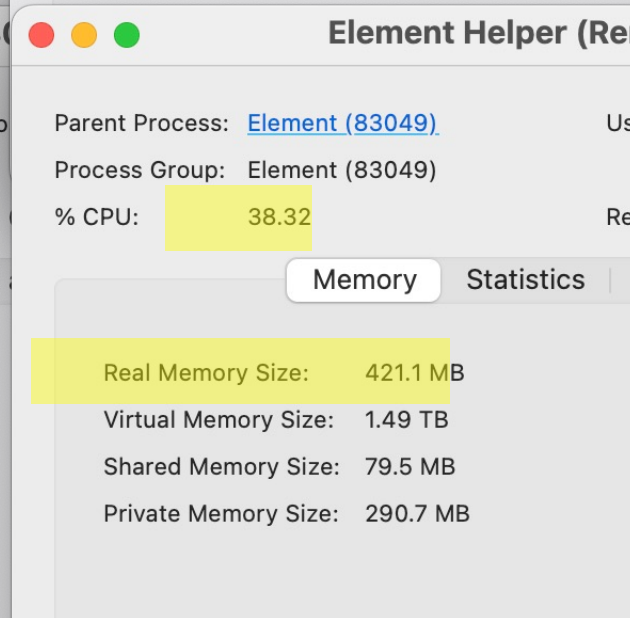


- 30% fonts
- 32% framebuffers









→ Related benchmarks for Makepad:
robius.rs/blog/performance-benchmarking-2/

Robrix development revealed a concurrency challenge

A typical first attempt to fetch and draw an image

```
impl Widget for TimelineView {
  fn draw_walk(&mut self, cx: &mut Cx2d, scope: &mut Scope, walk: Walk) -> DrawStep {
    while let Some(item_id) = self.list.next_visible_item(cx) {
      let Some(timeline_item) = self.tl_items.get(tl_idx) else { ... };
      if let TimelineItemKind::Event(event_tl_item) = match timeline_item.kind() {
        if let TimelineItemContent::Message(message) = event_tl_item.content() {
          if let MessageType::Image(image) = message.msgtype() {
            let (item, existed) = list.item (cx, item_id, live_id!(ImageMessage))?;
            let Some(mimetype, width, height) = image.info.as_ref() else { return };
            let text_or_image_widget = item.text_or_image(id!(content.message));
            let media_request = MediaRequest { source: image.source, format: ... };
            let data = matrix_client.media().get_media_content(&media_request, true).await?;
            text_or_image_widget.show_image(|img| utils::load_png_or_jpg(&img, cx, &data));
            ...
          }
        }
      }
    }
  }
}
```

```
error[E0728]: `await` is only allowed inside `async` functions and blocks
     |
1048 | fn draw_walk(
     | ------------- this is not `async`
...
1132 | let data = matrix_client.media().get_media_content(&media_request, true).await?;
                                                                  only allowed inside `async` functions and blocks ^^^^^
```

Robrix development revealed a concurrency challenge

Ok, let's try the request in a spawned async task

```
impl Widget for TimelineView {
  fn draw_walk(&mut self, cx: &mut Cx2d, scope: &mut Scope, walk: Walk) -> DrawStep {
    while let Some(item_id) = self.list.next_visible_item(cx) {
      let Some(timeline_item) = self.tl_items.get(tl_idx) else { ... };
      if let TimelineItemKind::Event(event_tl_item) = match timeline_item.kind() {
        if let TimelineItemContent::Message(message) = event_tl_item.content() {
          if let MessageType::Image(image) = message.msgtype() {
            let (item, existed) = list.item (cx, item_id, live_id!(ImageMessage))?;
            let Some(mimetype, width, height) = image.info.as_ref() else { return };
            let text_or_image_widget = item.text_or_image(id!(content.message));
            let media_request = MediaRequest { source: image.source, format: ... };
            let data = Handle::current().spawn(async {
              matrix_client.media().get_media_content(&media_request, true).await?
            });
            text_or_image_widget.show_image(|img| utils::load_png_or_jpg(&img, cx, &data));
            ...
} } } } } }
```

```
thread 'main' panicked at src/home/room_screen.rs:1128:25:
there is no reactor running, must be called from the context of a Tokio 1.x runtime
```

Robrix development revealed a concurrency challenge

Obvious: *must not block the main UI thread*

- Invoking a blocking (sync) function from the main thread will hang the UI
  - Especially if it causes a syscall, where the OS may put the thread to sleep

Less obvious: **async** functions can ***also*** block the main thread

- Even if the function is async, that native thread could still be blocked while the async executor is polling the future being `awaited`
- Same issue can occur on background threads, too (but that's ok)

Key point:

→ Don't invoke ***any*** blocking functions, even "non-blocking" async

(on the main UI thread)

Robrix development revealed a concurrency challenge

```
impl Widget for TimelineView {
  fn draw_walk(&mut self, cx: &mut Cx2d, scope: &mut Scope, walk: Walk) -> DrawStep {
    while let Some(item_id) = self.list.next_visible_item(cx) {
      let Some(timeline_item) = self.tl_items.get(tl_idx) else { ... };
      if let TimelineItemKind::Event(event_tl_item) = match timeline_item.kind() {
        if let TimelineItemContent::Message(message) = event_tl_item.content() {
          if let MessageType::Image(image) = message.msgtype() {
            let (item, existed) = list.item (cx, item_id, live_id!(ImageMessage)).unwrap();
            let Some(mimetype, width, height) = image.info.as_ref() else { return };
            let text_or_image_widget = item.text_or_image(id!(content.message));
            if let MediaSource::Plain(mxc_uri) => &image.source {
              match self.media_cache.try_get_media_or_fetch(mxc_uri.clone(), None) {
                MediaCacheEntry::Loaded(data) => {
                  text_or_image_widget.show_image(|img| utils::load_png_or_jpg(&img, cx, &data));
                  ...
                MediaCacheEntry::Requested => {
                  text_or_image_widget.set_text(&format!("Fetching image from {:?}", mxc_uri));
                }
                MediaCacheEntry::Failed => {
                  text_or_image_widget.set_text(&format!("Failed to fetch image from {:?}", mxc_uri));
                }
              } ...
```

Solution: →

A concurrency-aware impl that requests an async function to be run in a different context

Robrix development revealed a concurrency challenge

```
/// Tries to get the media from the cache, or submits an async request to fetch it.
///
/// This method *does not* block or wait for the media to be fetched,
/// and will return `MediaCache::Requested` while the async request is in flight.
/// If a request is already in flight, this will not issue a new redundant request.
pub fn try_get_media_or_fetch(&mut self, mxc_uri: OwnedMxcUri ...) -> MediaCacheEntry {
  let value_ref = match self.entry(mxc_uri.clone()) {
    Entry::Vacant(vacant) => vacant.insert(MediaCacheEntry::Requested),
    Entry::Occupied(occupied) => return occupied.get()...,
  };

  async_ctx::submit_async_request(
    MatrixRequest::FetchMedia {
      media_request: mxc_uri.into(),
    },
  );
  ...
}
```

```
/// Submits a request to the worker thread
/// to be executed in an async context.
pub fn submit_async_request(req: MatrixRequest)
{
  REQUEST_SENDER.get().send(req);
}
```

Robrix development revealed a concurrency challenge

```rust
/// Submits a request to the worker thread
/// to be executed in an async context.
pub fn submit_async_request(req: MatrixRequest)
{
  REQUEST_SENDER.get().send(req);
}
```

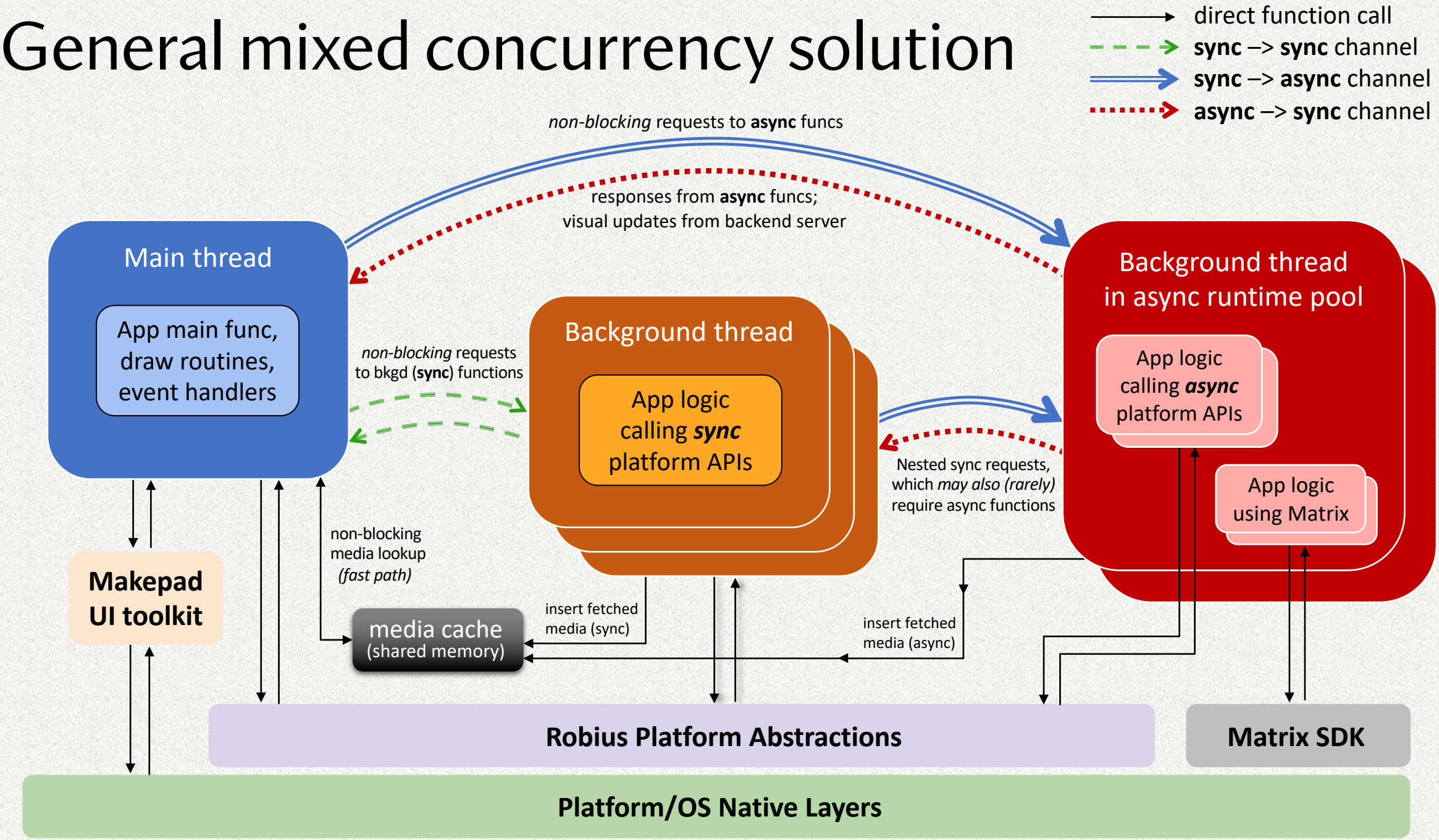
triggers async request handler

```rust
pub enum MatrixRequest {
  FetchMedia {
    media_request: MediaRequest,
    on_fetched: fn(&Mutex<MediaCacheEntry>, MediaRequest, ...),
    destination: Arc<Mutex<MediaCacheEntry>>,
    update_sender: crossbeam_channel::Sender<TimelineUpdate>,
    ...,
} }
```

```rust
async fn async_worker(mut receiver: UnboundedReceiver<MatrixRequest>) -> Result<()> {
  while let Some(request) = receiver.recv().await {
    match request {
      MatrixRequest::FetchMedia { media_request, on_fetched, destination, update_sender } => {
        Handle::current().spawn(async move {
          let img_data = client.media().get_media_content(&media_request, true).await;
          on_fetched(&destination, media_request, img_data, update_sender);
        });
} } } }
```

This same pattern is used for back pagination, sending messages, etc

General mixed concurrency solution



Wrapping up, but looking forward

- Robius is only just getting started!
  - Many challenges remain: UI, platform, build, tooling
  - Active collaboration with UI toolkits, who are improving every day

Roadmap for 2024:

Continue developing key **platform feature abstraction** crates

Publish first flagship apps to app stores & package managers

End-to-end **guided walkthroughs**: creating a full app atop Robius

Acknowledgments

Rik Arends

Eddy Bruël

Edward Tan

David Rheinsberg

Klim Tsoutsman

Sebastian Michailidis

Jonathan Kelley

Jorge Bejar

Julian Montes de Oca

Cassaundra Smith

Thanks!

Interested? Please reach out:

@project-robius

@kevinaboos

#robius@matrix.org