Shipping Production Ready Rust for Android

Tarik Eshaq

About me:

moz://a

About me:

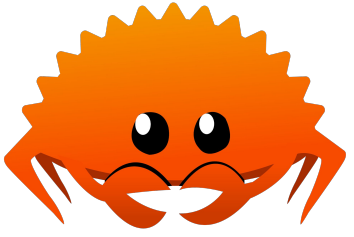
moz://a

Firefox

About me:

moz://a

Firefox

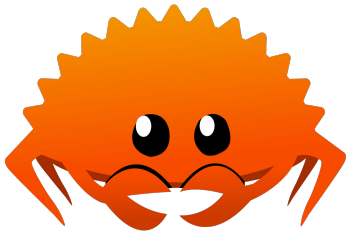


@tarikeshaq

Agenda

1. What is Rust
2. Why Rust & Memory Safety
3. When to use Rust in Android
4. How to use Rust in Android
5. Problems with using Rust in Android
6. Remaining limitations with using Rust in Android

What is Rust



What is Rust

Multipurpose programming language

What is Rust

Multipurpose programming language

No Automated Garbage Collection or Runtime

Like C

What is Rust

Multipurpose programming language

Like C

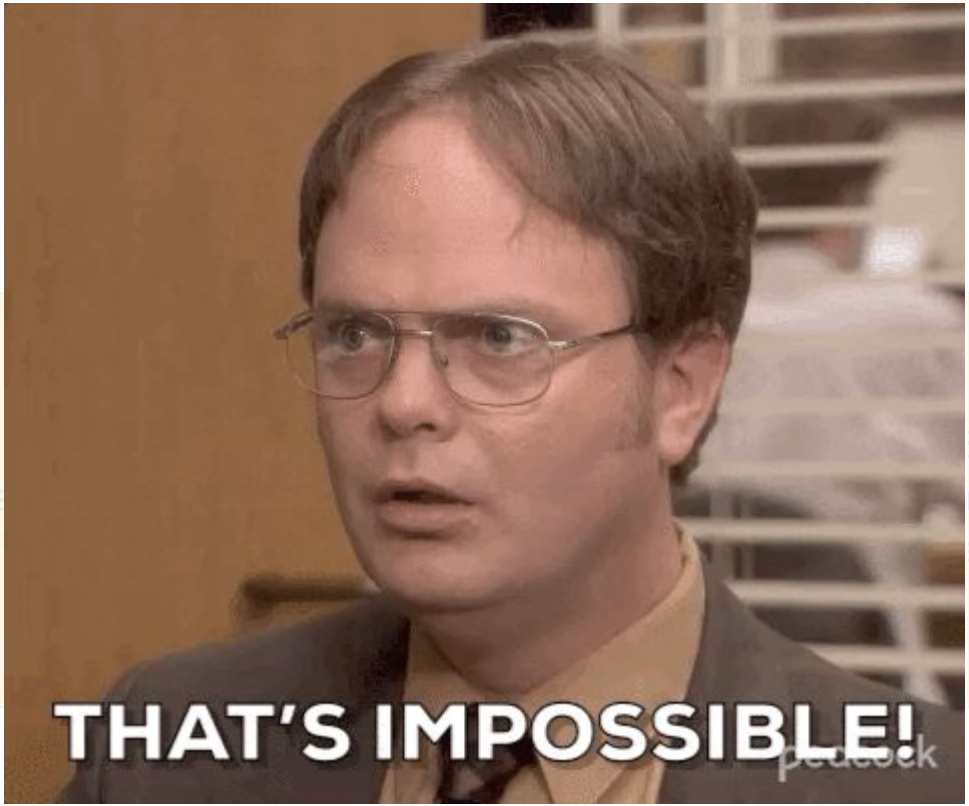
No Automated Garbage Collection or Runtime

Unlike C

Memory Safety

No use-after-free, segmentation faults, null pointers

What is Rust




COMPILER MAGIC
imgflip.com

=

- Ownership
- Lifetimes
- Strong Type System

C

```c
#include <stdio.h>


int* f() {
    int a = 3;
    return &a;
}


int main() {
    printf("a: %i\n", *f()); // prints a: 3
}
```

```
➜  droid-con-2023 gcc -o f f.c
f.c:5:13: warning: address of stack memory associated with local variable 'a' returned [-Wreturn-stack-address]
    return &a;
           ^
1 warning generated.
➜  droid-con-2023 ./f
a: 3
➜  droid-con-2023
```

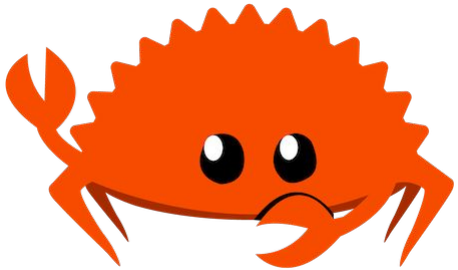
```
fn f() -> &i32 {
    let a = 3;
    return &a;
}

fn main() {
    println!("a: {}", *f()); // does not compile
}
```

```
➜  droid-con-2023 rustc f.rs -o f
error[E0106]: missing lifetime specifier
 --> f.rs:1:11
  |
1 | fn f() -> &i32 {
  |           ^ expected named lifetime parameter
  |
  = help: this function's return type contains a borrowed value, but there is no value for it to be borrowed from
help: consider using the `'static` lifetime
  |
1 | fn f() -> &'static i32 {
  |            +++++++

error: aborting due to previous error

For more information about this error, try `rustc --explain E0106`.
```

```
fn f() -> &'static i32 {
    let a = 3;
    return &a;
}

fn main() {
    println!("a: {}", *f()); // does not compile
}
```

```
→  droid-con-2023 rustc f.rs -o f
error[E0515]: cannot return reference to local variable `a`
 ⟶ f.rs:3:12
  │
3 │     return &a;
  │            ^^ returns a reference to data owned by the current function

error: aborting due to previous error

For more information about this error, try `rustc --explain E0515`.
→  droid-con-2023
```

Why Rust & Memory Safety



Why Rust & Memory Safety (Android)

Memory Safe Languages in Android 13

December 1, 2022

Posted by Jeffrey Vander Stoep

For more than a decade, memory safety vulnerabilities have consistently represented more than 65% of vulnerabilities across products, and across the industry. On Android, we're now seeing something different - a significant drop in memory safety vulnerabilities and an associated drop in the severity of our vulnerabilities.

Why Rust & Memory Safety (Chrome)

An update on Memory Safety in Chrome

September 21, 2021

Adrian Taylor, Andrew Whalley, Dana Jansens and Nasko Oskov, Chrome security team

Security is a cat-and-mouse game. As attackers innovate, browsers always have to mount new defenses to stay ahead, and Chrome has invested in ever-stronger multi-process architecture built on sandboxing and site isolation. Combined with fuzzing, these are still our primary lines of defense, but they are reaching their limits, and we can no longer solely rely on this strategy to defeat in-the-wild attacks.

Last year, we showed that more than 70% of our severe security bugs are memory safety problems. That is, mistakes with pointers in the C or C++ languages which cause memory to be misinterpreted.

Why Rust & Memory Safety (Microsoft)

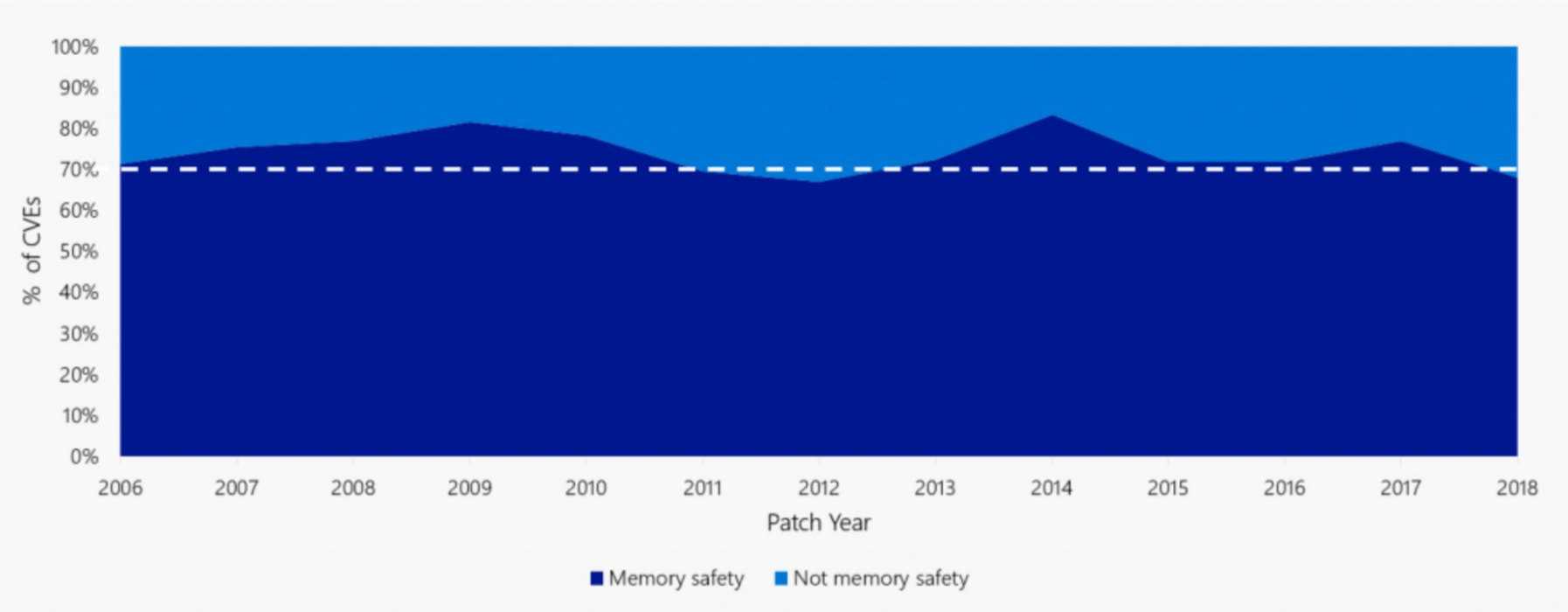


Figure 1: ~70% of the vulnerabilities Microsoft assigns a CVE each year continue to be memory safety issues

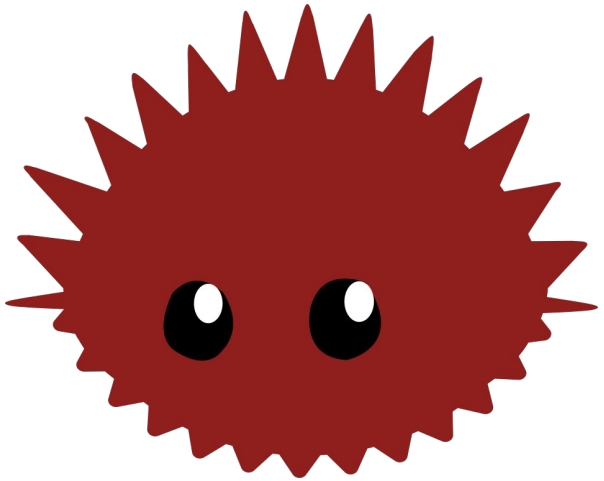
Why Rust & Memory Safety (AWS)

At AWS, Rust has quickly become critical to building infrastructure at scale. Firecracker is an open source virtualization technology that powers AWS Lambda and other serverless offerings. It launched publicly in 2018 as our first notable product implemented in Rust. We use Rust to deliver services such as Amazon Simple Storage Service (Amazon S3), Amazon Elastic Compute Cloud (Amazon EC2), Amazon CloudFront, and more. In 2020, we launched Bottlerocket, a Linux-based container operating system written in Rust, and our Amazon EC2 team uses Rust as the language of choice for new AWS Nitro System components, including sensitive applications, such as Nitro Enclaves.

Why Rust & Memory Safety (Meta)

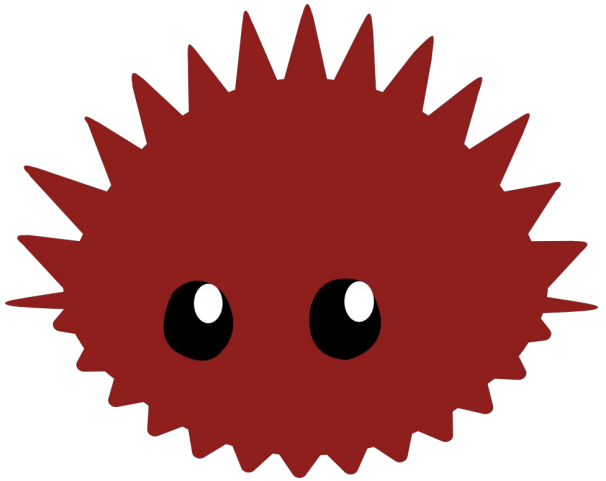
Facebook is embracing Rust, one of the most loved and fastest-growing programming languages available today. In addition to bringing new talent to its Rust team, Facebook has announced that it is officially joining the nonprofit Rust Foundation. Alongside fellow members including Mozilla (the creators of Rust), AWS, Microsoft, and Google, Facebook will be working to sustain and grow the language's open source ecosystem.

When would you use Rust in Android?



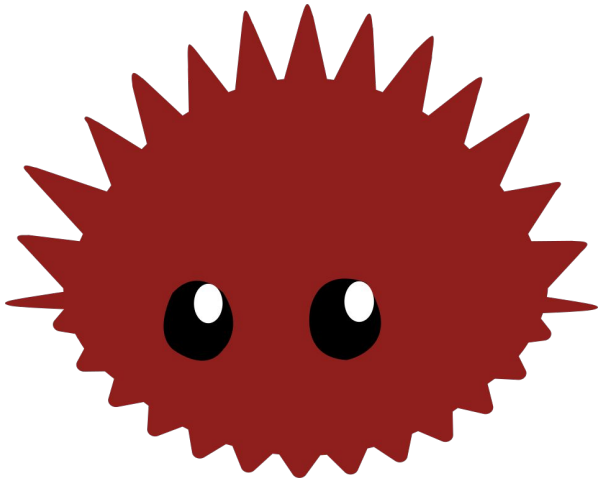
When would you use Rust in Android?

Don't*



When would you use Rust in Android?

Don't*



* Unless you would have used C or C++

When would you use Rust in Android?

1. Desire to ship code to multiple platforms

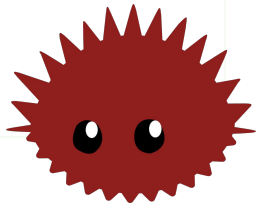
- Especially ones including ones that do not support the JVM

2. Shipping a Garbage collector or Runtime is not viable

- Either it's too expensive, or you would like granular memory management

3. 1 & 2 are true, and you'd like memory safety

- This should almost always be the case, unless you are missing core libraries or ecosystem support



When would you use Rust in Android?

1. Desire to ship code to multiple platforms

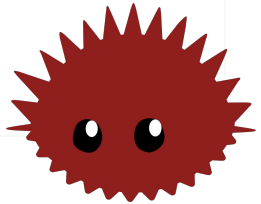
- Especially ones that do not support the JVM

2. Shipping a Garbage collector or Runtime is not viable

- Either it's too expensive, or you would like granular memory management

3. 1 & 2 are true, and you'd like memory safety

- This should almost always be the case, unless you are missing core libraries or ecosystem support



When would you use Rust in Android?

1. Desire to ship code to multiple platforms

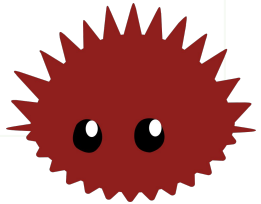
- Especially ones that do not support the JVM

2. Shipping a Garbage collector or Runtime is not viable

- Either it's too expensive, or you would like granular memory management

3. 1 & 2 are true, and you'd like memory safety

- This should almost always be the case, unless you are missing core libraries or ecosystem support



When would you use Rust in Android?

1. Desire to ship code to multiple platforms

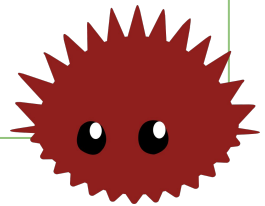
- Especially ones that do not support the JVM

2. Shipping a Garbage collector or Runtime is not viable

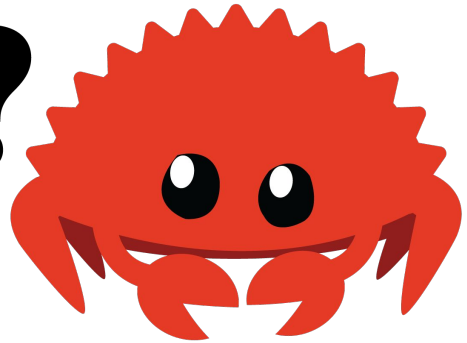
- Either it's too expensive, or you would like granular memory management

3. 1 & 2 are true, and you'd like memory safety

- This should almost always be the case, unless you are missing core libraries or ecosystem support

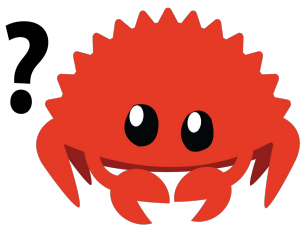


How to use Rust in Android ?



How to use Rust in Android

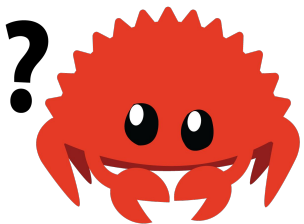
Compile Rust code using Rust compiler



How to use Rust in Android

Compile Rust code using Rust compiler

Link Rust binary with JVM application

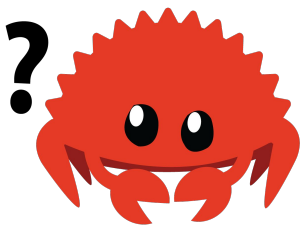


How to use Rust in Android

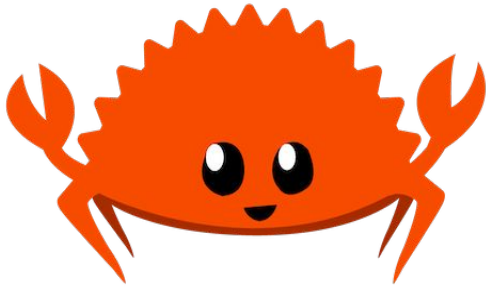
Compile Rust code using Rust compiler

Link Rust binary with JVM application

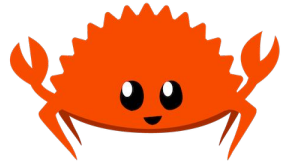
Interface between Kotlin/Java & Rust binary



Problems with Rust in Android



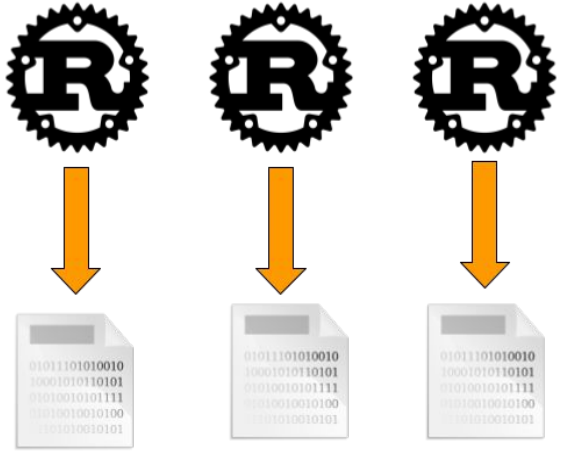
Problems with Rust in Android



Binaries can have duplicate Rust libraries

Problems with Rust in Android

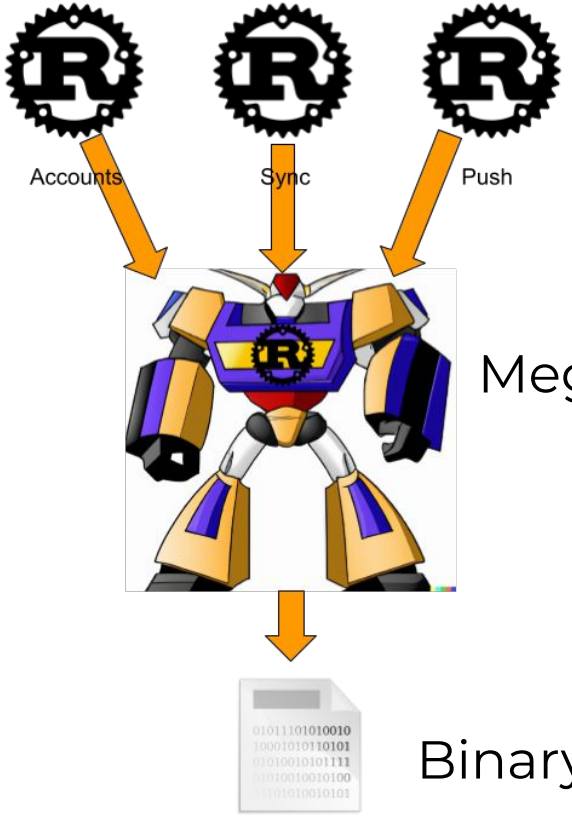
Binaries can have duplicate Rust libraries



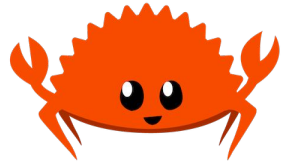
Binary Files

Problems with Rust in Android

Binaries can have duplicate Rust libraries



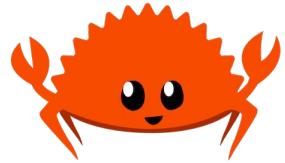
Problems with Rust in Android



Binaries can have duplicate Rust libraries

```
[lib]
crate-type = ["cdylib"]


[dependencies]
account = { path = "./account" }
sync = { path = "./sync" }
push = { path = "./push" }
```

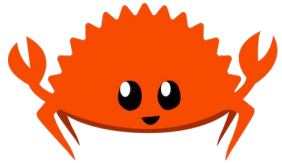
Problems with Rust in Android



Binaries can have duplicate Rust libraries

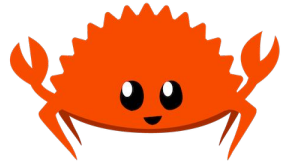
```
pub use account;
pub use sync;
pub use push;
```

Problems with Rust in Android



Integrating the Rust compiler with an Android build system is not easy

Problems with Rust in Android



Integrating the Rust compiler with an Android build system is not easy

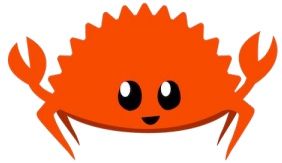
README.md

Rust Android Gradle Plugin

Cross compile Rust Cargo projects for Android targets.

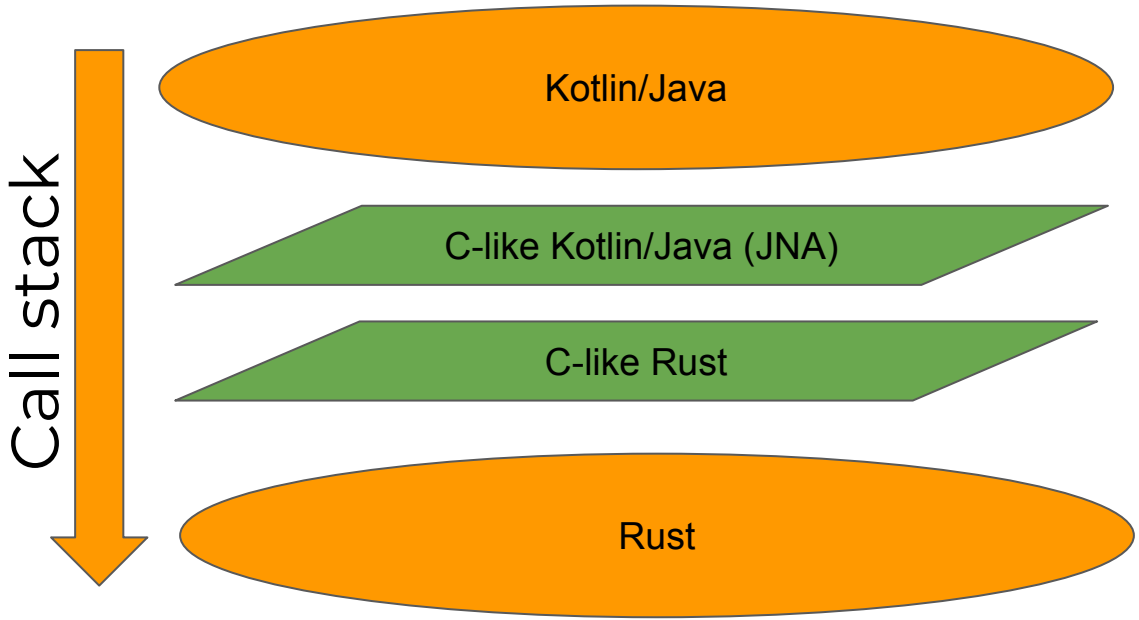
rust-android-gradle v0.9.3

Problems with Rust in Android

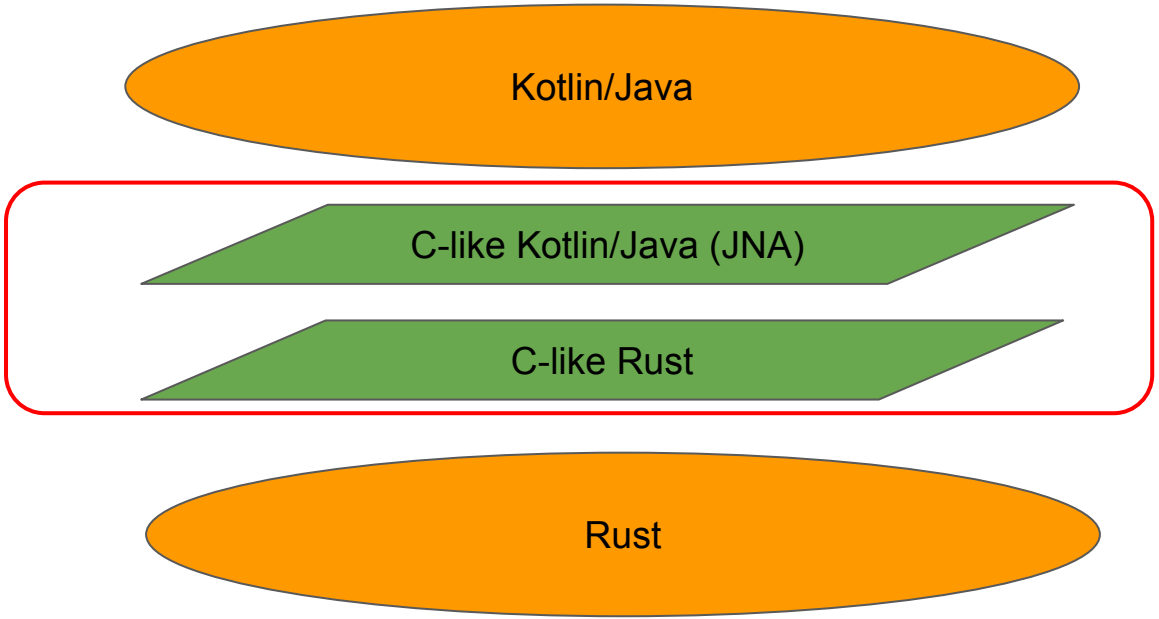


Interfacing between Kotlin and Rust is difficult

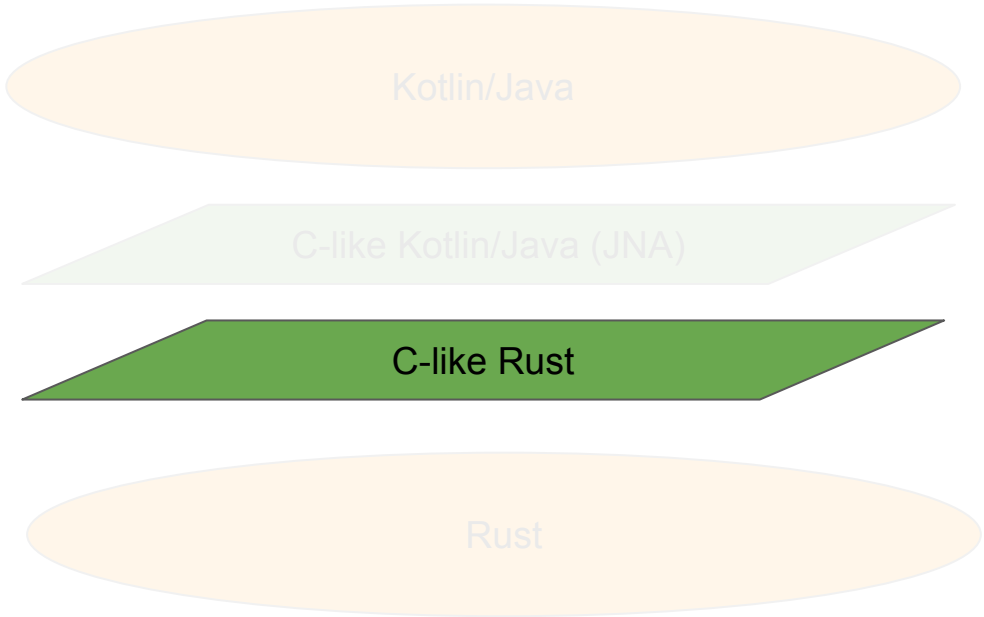
The FFI Sandwich



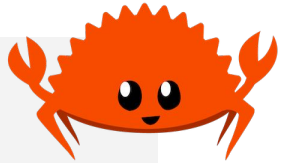
The FFI Sandwich



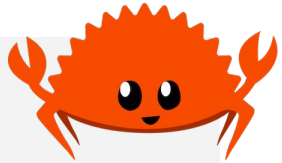
Foreign Function Interface

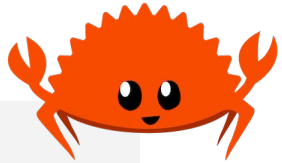

Kotlin/Java
C-like Kotlin/Java (JNA)
C-like Rust
Rust

```
#[no_mangle]
pub extern "C" fn sync() {
    // do sync stuff
    // Call other Rust functions normally
}
```

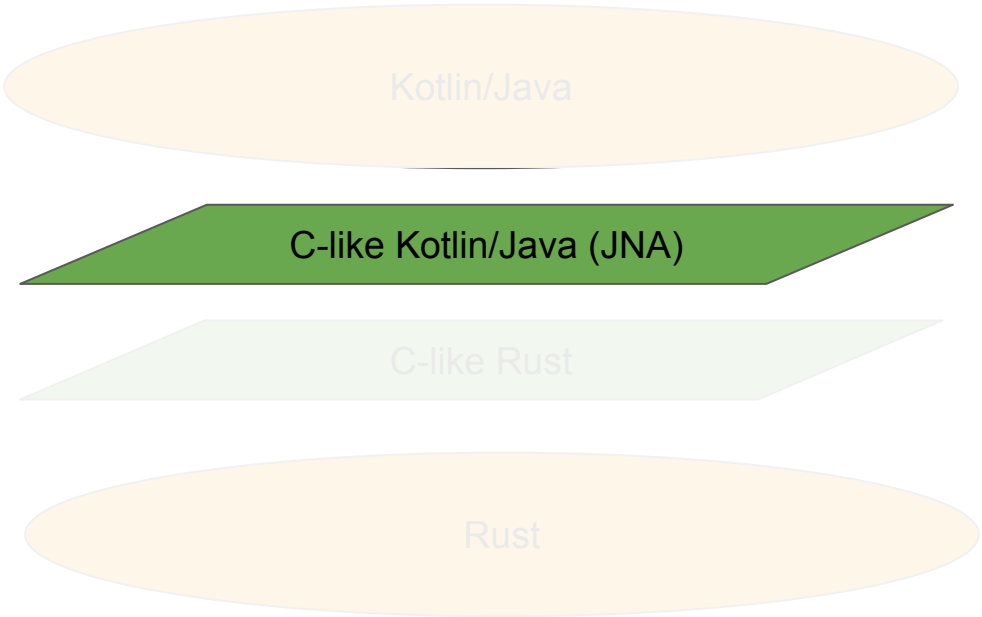


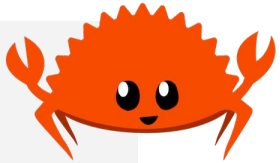
```
#[no_mangle]
pub extern "C" fn sync() -> i32 {
    match sync_impl() {
        Ok(()) => 0,
        // to_error_code implemented elsewhere
        Err(e) => e.to_error_code()
    }
}
```





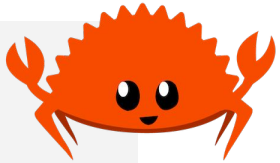
```
#[no_mangle]
pub extern "C" fn sync() -> *const u8 {
    match sync_impl() {
        // to_ffi_buffer implemented elsewhere
        Ok(res) => res.to_ffi_buffer(),
        Err(e) => e.to_ffi_buffer()
    }
}
```


Kotlin/Java
C-like Kotlin/Java (JNA)
C-like Rust
Rust

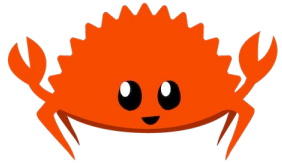


```
internal interface RustLib : Library {
    companion object {
        internal val INSTANCE: RustLib by lazy {
            loadIndirect<RustLib>(componentName = "sync")
        }
    }
    fun sync(): Int
}
```

```
fun sync() {
  val status = RustLib.INSTANCE.sync()
  if status != 0 {
     throw new SyncFFIException(status)
  }
}
```



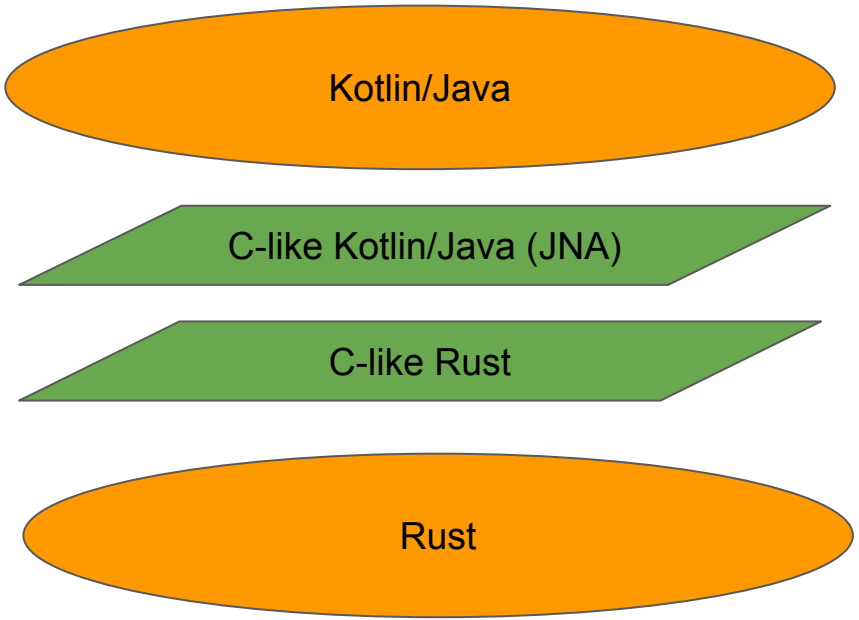
Problems with Rust in Android



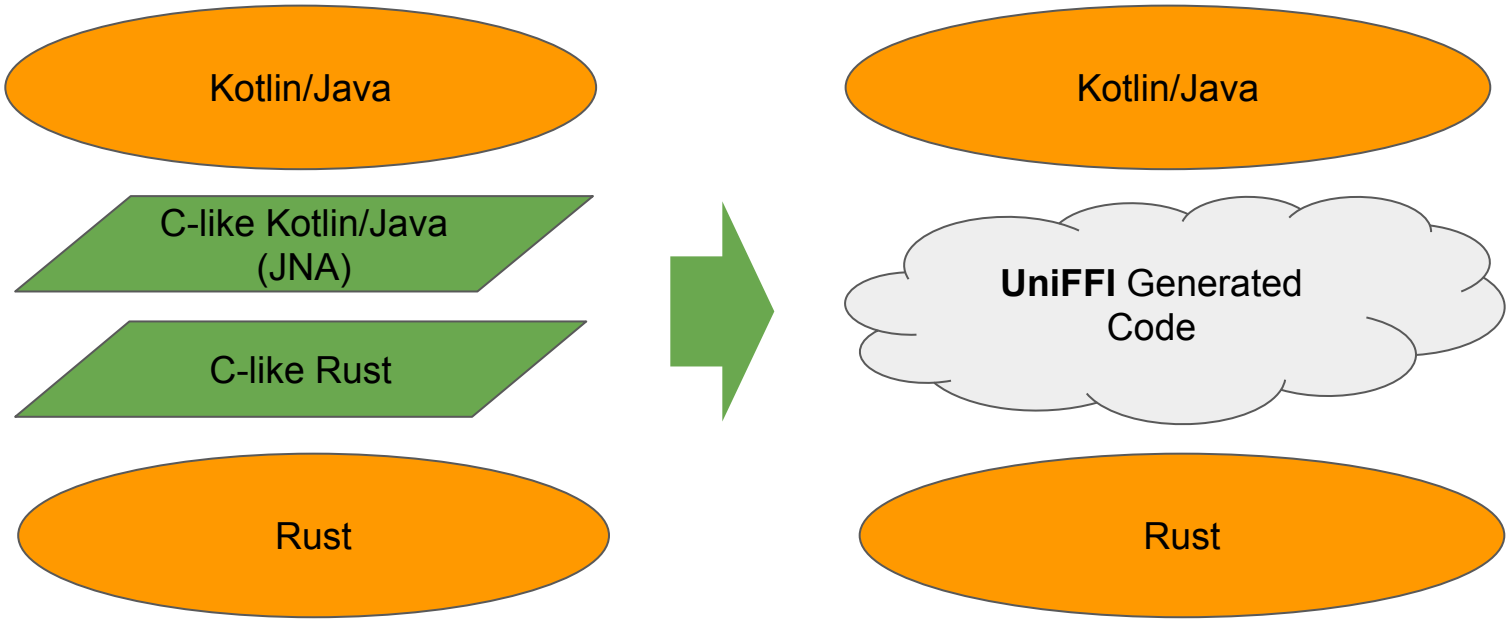
Interfacing between Kotlin and Rust is difficult

- **Many FFI operations are unsafe**
- **Restricted to types compatible with C**
- **A large cognitive load on both Kotlin and Rust engineers**

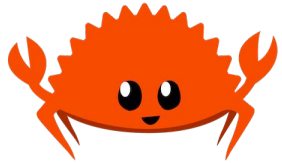
Problems with Rust in Android



Problems with Rust in Android



UniFFI

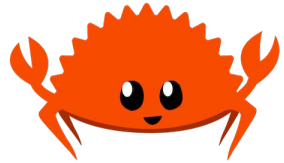


UniFFI is an open source tool built at Mozilla

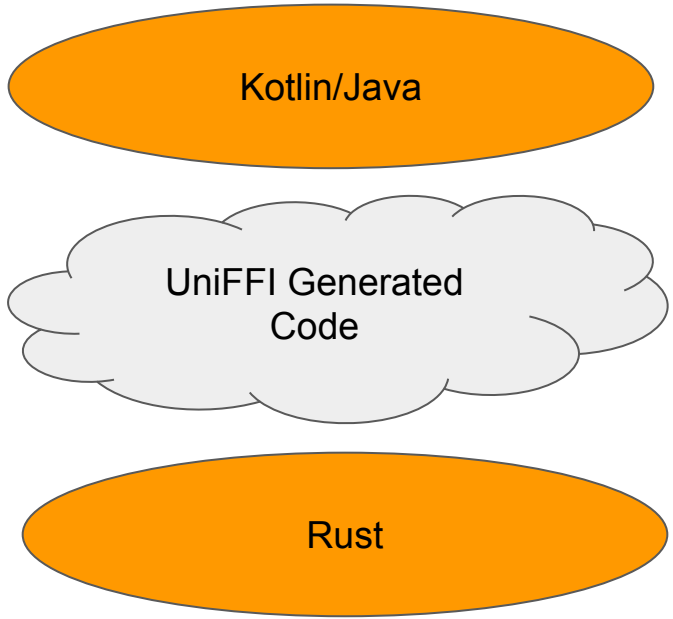
Generates the inside of the FFI sandwich

You only worry about the buns!

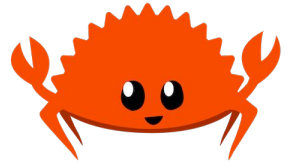
Problems with Rust in Android



Interfacing between Kotlin and Rust is difficult



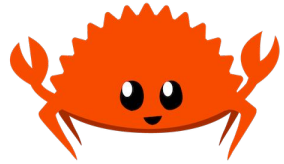
Problems with Rust in Android



Interfacing between Kotlin and Rust is difficult

```
fn sync() ->
Result<SyncResult, SyncError> {
    // Sync stuff
}
```

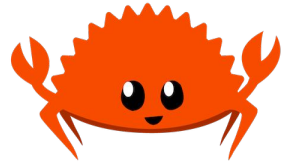
Problems with Rust in Android



Interfacing between Kotlin and Rust is difficult

```
[Throws=SyncError]
SyncResult sync();


[Error]
enum SyncError { .. };


dictionary SyncResult { .. };
```

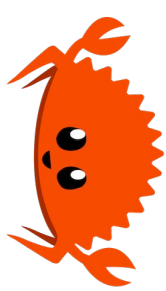
Problems with Rust in Android

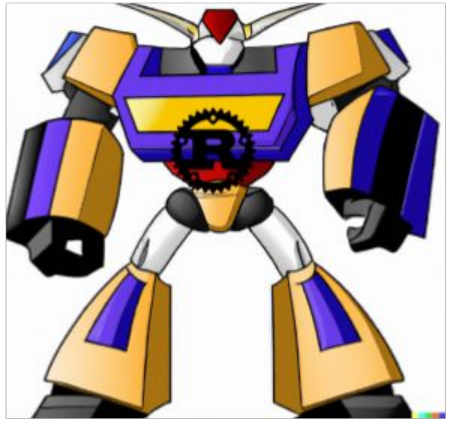


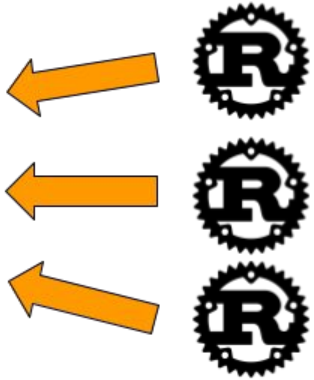
Interfacing between Kotlin and Rust is difficult

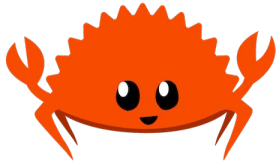
```
fun caller() {
    try {
        sync()
    } catch (e: SyncException) {
        ...
    }
}
```

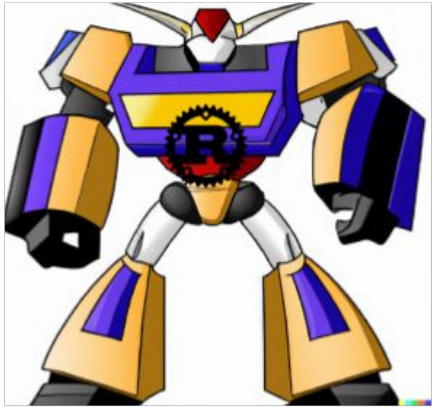
卍 卍 卍

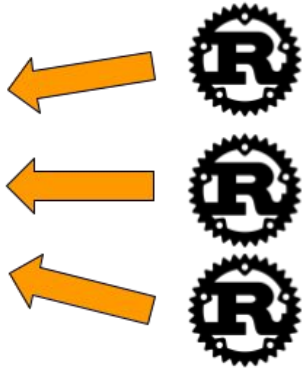


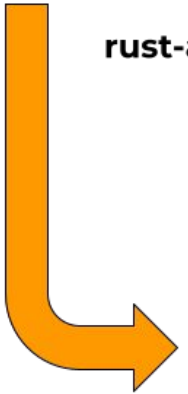




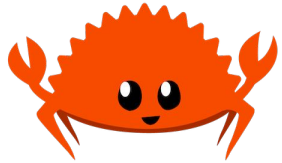


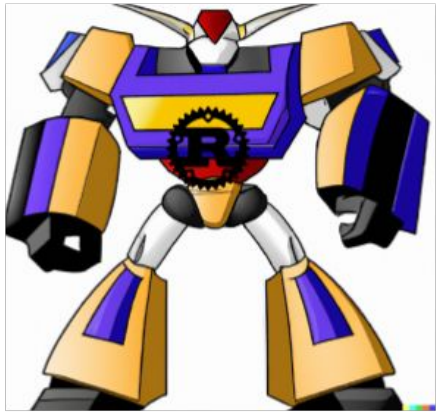


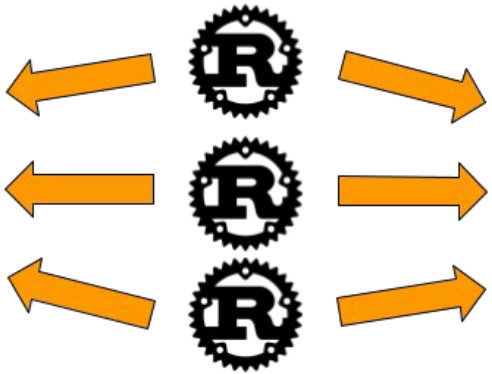




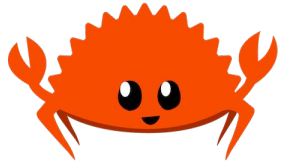
rust-android-gradle

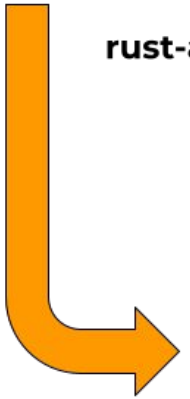




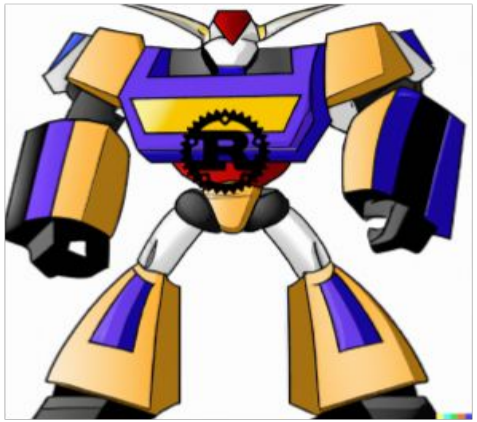


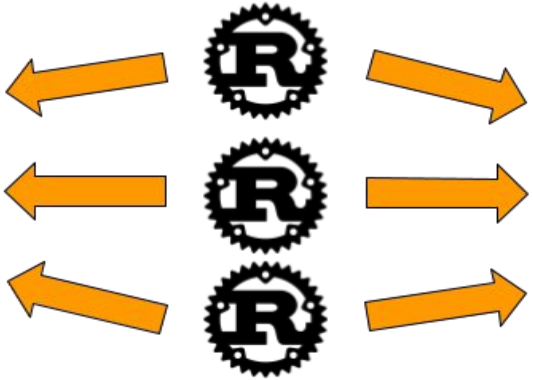
UniFFI





rust-android-gradle





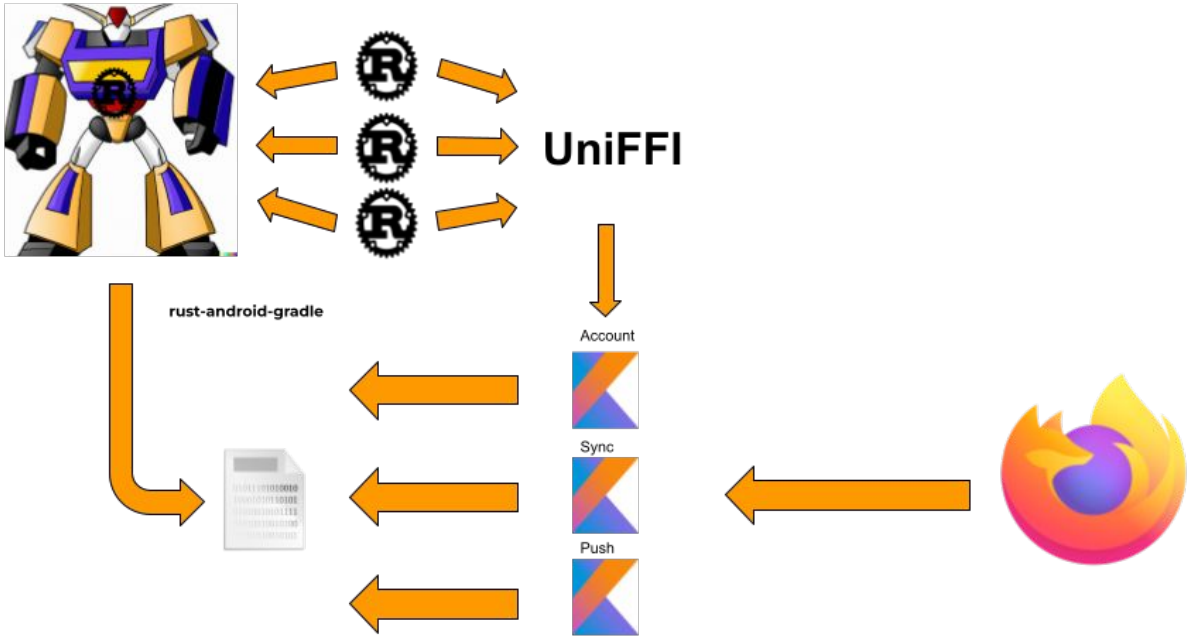
UniFFI

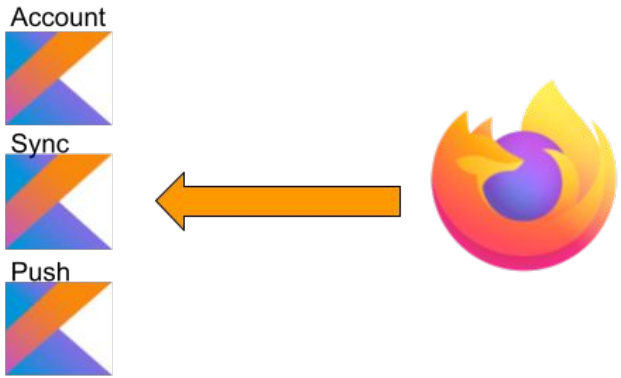
rust-android-gradle

Account

Sync

Push


UniFFI
rust-android-gradle
Account
Sync
Push


Account
Sync
Push

Remaining Limitations with Rust in Android

Remaining Limitations with ~~Rust~~ Native Languages in Android

Remaining Limitation with Native Languages in Android

1. Restricted Threading

- The Android operating system is strict with background threads
- Difficult to manage app lifecycle events from outside the JVM

2. Asynchrony needs to be managed by the application

- Can't use async runtimes
- I/O blocks calling thread
- Difficulty separating blocking and non-blocking calls

3. Developer Experience

- Rust Engineers crossing over to Android & Android engineers crossing over to Rust creates friction

Remaining Limitation with Native Languages in Android

1. Restricted Threading

- The Android operating system is strict with background threads
- Difficult to manage app lifecycle events from outside the JVM

2. Asynchrony needs to be managed by the application

- Can't use async runtimes
- I/O blocks calling thread
- Difficulty separating blocking and non-blocking calls

3. Developer Experience

- Rust Engineers crossing over to Android & Android engineers crossing over to Rust creates friction

Remaining Limitation with Native Languages in Android

1. Restricted Threading

- The Android operating system is strict with background threads
- Difficult to manage app lifecycle events from outside the JVM

2. Asynchrony needs to be managed by the application

- Can't use async runtimes
- I/O blocks calling thread
- Difficulty separating blocking and non-blocking calls

3. Developer Experience

- Rust Engineers crossing over to Android & Android engineers crossing over to Rust creates friction

Remaining Limitation with Native Languages in Android

1. Restricted Threading

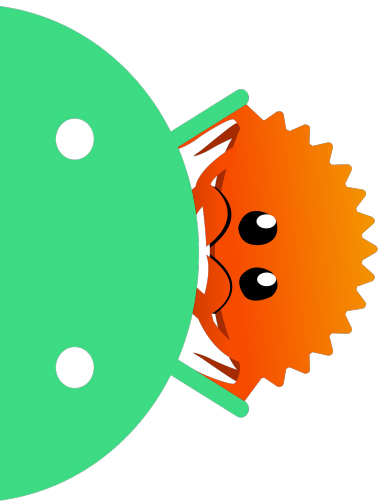
- The Android operating system is strict with background threads
- Difficult to manage app lifecycle events from outside the JVM

2. Asynchrony needs to be managed by the application

- Can't use async runtimes
- I/O blocks calling thread
- Difficulty separating blocking and non-blocking calls

3. Developer Experience

- Rust Engineers crossing over to Android & Android engineers crossing over to Rust creates friction



Thank You

Tarik Eshaq

References

- https://security.googleblog.com/2022/12/memory-safe-languages-in-android-13.html
- https://security.googleblog.com/2021/09/an-update-on-memory-safety-in-chrome.html
- https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/
- https://aws.amazon.com/blogs/opensource/sustainability-with-rust/
- https://alexgaynor.net/2020/may/27/science-on-memory-unsafety-and-security/

Resources

- Rust Programming Language Book (https://doc.rust-lang.org/book/)
- Rust android gradle plugin (https://github.com/mozilla/rust-android-gradle)
- UniFFI (https://mozilla.github.io/uniffi-rs/)
- JNA (https://github.com/java-native-access/jna)