

# C++, Rust and Qt: Easier than you think

Joshua Goins

# C++, Rust AND Qt? Really?

# Overview

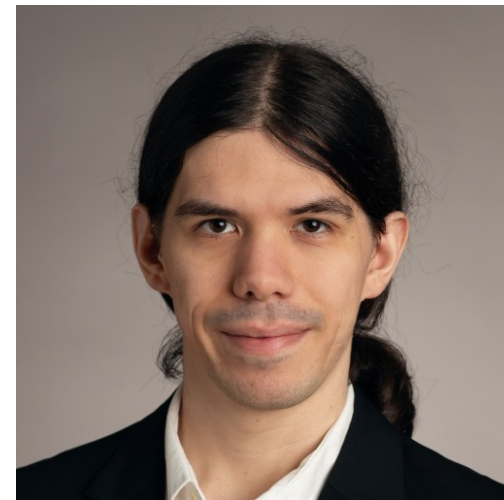
- Non-technical bits:
  - What is Rust?
  - Rust in KDE
    - What we have and what we could have
- For technical wizards:
  - Possible benefits of Rust (for a C++ programmer)
  - Integration strategies for Qt, Rust and C++
    - Existing applications to take inspiration from

# No Previous Qt + Rust Talk?

- I quickly skimmed past programs:
  - Tobias Hunger talking about Rust + Slint UI (2023)
  - Méven talking about Rust in KDE (2020)
  - Emma showcasing Rust support in KDevelop (2017)
- We have plenty of stuff to talk about!

# Who Am I?

- Joshua Goins
- Software Engineer at KDAB
- Been hacking away at KDE for a couple of years now
- Have been using Rust for lots of personal projects, I want to use Qt in them!



# Rust Basics

# What Is Rust?

- Systems language, like C++
- Officially released in 2015, but still a relatively new language
- Focuses on “safety” and lots of interesting compiler-based checks
- Similar syntax to C++ and C, in some surface level aspects

# Rust Features

- “Batteries included”
  - Testing suite
  - Package system for developers
  - Build system
  - Benchmarking (soon!)
  - Standard libraries
  - Toolchains for obscure targets



# Rust “Promises”

- Rust tries to be more safe than C++
- Checks for:
  - Memory safety
  - Thread safety
  - Ownership
- We’ll go over an example later in the technical section



# Rust in KDE

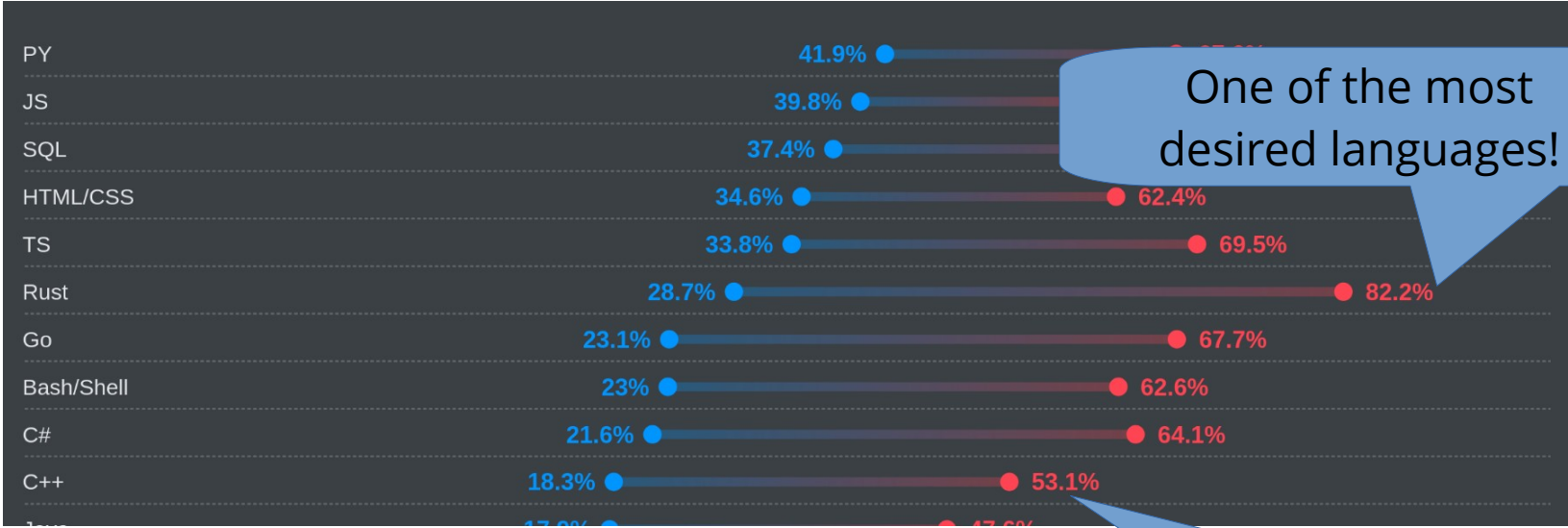
# Why Are We Even Here?

- KDE has always been written in C++, why should we care?
- What can Rust bring to the table?
- Why Rust and not X, Y or [insert your favorite language]?
  - I can do what I want!
  - Rust has lots of tools for C/C++ integration

# What Rust Can Bring to the Table

- New contributors?
  - Less and less people are interested in C++
  - They say its “legacy” and “Rust is more interesting”
  - And also that C++ is a waste of time :-)
  - Regardless if that’s true, Rust contributors are a huge untapped market
  - If we want a healthy flow of new contributors, we have to start thinking about this stuff!

# What Rust Can Bring to the Table



One of the most desired languages!

C++ is still loved!



Source: [StackOverflow Developer Survey, 2024](#)

# What Rust Can Bring to the Table

- Thinking outside the box
  - Rust is in dire need of GUI applications
  - As KDE, we want people to use Qt. By extension, they can be pulled into KDE to use our frameworks!
  - Making it easier to write Qt-enabled Rust benefits KDE and the greater Rust ecosystem
  - Using CXX, we can also write bindings for our frameworks
    - More on this later!

# What Rust Can Bring to the Table

- Memory safe code
  - Like in Akonadi, tasks like “HTML parsing” is pretty separated from the UI layer
  - There are *\*real\** crashes we are solving via Rust
  - What are some other areas we could apply Rust to?

# What Rust Can Bring to the Table

- Useful libraries exist in Rust!
  - Libraries that lack a C API, can be hoisted into existing KDE applications with a little bit of Rust glue.
  - Corrosion makes this extremely easy and we're already shipping it in some cases
    - More about this later in the technical section



# A Real World Case of Libraries

- Servo is a newer and work-in-progress web engine written in Rust
- We can use CXX-Qt to expose it in Rust!
  - Not limited to Rust-enabled Qt applications, since it's exposed as a regular QML component!



<https://github.com/KDABLabs/cxx-qt-servo-webview>

# What To Avoid

- Don't think: "rewrite in Rust"
  - Fruitless endeavor, costly, and really no one wants to do that
  - There are benefits but those tend to be very little compared to the drawbacks
  - Not *\*everything\** needs to be written in Rust either

# Technical Part Begin

- Now it's time to talk technical bits
- This can be more boring or more interesting, depending on you

# Integrating Rust and C++

# It's Happening!?

- “Rewrite in Rust” is already happening?!?
- **Angelfish** (Web Browser)
  - Uses an adblocking crate
- **Pikasso** (Drawing)
  - Uses a crate for tessellation
- **Akonadi** (Personal Information Management)
  - Uses a crate for HTML parsing (woo! security!)



# Rust “Safety”

- Includes lots of safety features, tries to pave over some common pitfalls we see in C++
- ... How?

# C++ Example

```
class SomeClass {
public:
    void do_your_thing();
    std::vector<int> values;
};

void SomeClass::do_your_thing() {
    values.clear();
}

int main() {
    SomeClass *c = new SomeClass();
    c->values = {1, 2, 3, 4, 5};
    for (const auto &value : c->values) {
        if (value == 2)
            c->do_your_thing();
        std::cout << value << std::endl;
    }
    return 0;
}
```

Invalidating the  
iterator, destroying  
values, oh no!

1  
2  
3  
4  
5

Nonsensical output

Compiler is completely  
happy :-)

```
struct SomeStruct {
    values: Vec<i32>
}

impl SomeStruct {
    fn do_your_thing(&mut self) {
        self.values.clear();
    }
}

fn main() {
    let mut s = SomeStruct {
        values: vec![1, 2, 3, 4];
    };

    for val in &s.values {
        if *val == 2 {
            s.do_your_thing();
        }
        println!("{val}")
    }
}
```

The same, destructive action we're doing in C++.

What will happen?



# Rust Example

```
error[E0502]: cannot borrow `s` as mutable because it is also borrowed
as immutable
```

```
--> src/main.rs:18:13
```

```
16 |         for val in &s.values {
    |                   -----
    |                   |
    |                   immutable borrow occurs here
    |                   immutable borrow later used here
17 |             if *val == 2 {
18 |                 s.do_your_thing();
    |                 ^^^^^^^^^^^^^^^^^ mutable borrow occurs here
```

For more information about this error, try `rustc --explain E0502`.

# Rust “Safety”

- **NOT** a silver bullet
  - Logic bugs are not protected, that’s still possible in both Rust and C++
- Memory and static type safety are touted as security benefits
  - There’s still benefits to taking advantage of it even in non-security scenarios
- Rust can make you a better C++ programmer, and vice versa

# Sliding Scale

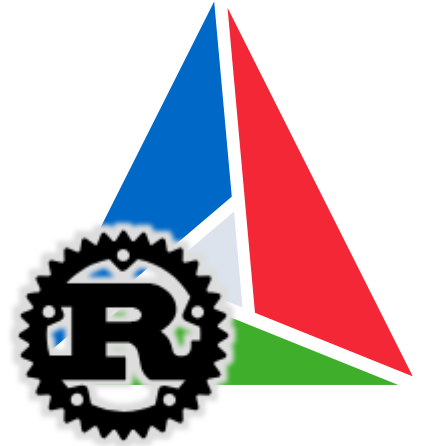
- Integrating Rust is **a sliding scale** and there's rarely a perfect solution for every project
  - Falling into the “rewrite in Rust” mindset will lead to disappointment
- Start small and work your way up in existing applications

# Methods to Integrate Rust

- There are many, many ways to expose Rust to C++ and vice versa
  - We'll focus on just CXX today, but there are many more solutions out there
- Bindings take a non-trivial amount of work to write and design initially, there's no avoiding that with any tool
- CXX will allow us to express and use C++ types without having to worry about C FFI

# Corrosion

- Tool used by almost every Rust & C++ application
- Hooks together CMake with Rust's Cargo
  - Exposes Rust libraries as regular CMake targets
- <https://github.com/corrosion-rs/corrosion>



# CXX


- Glues C++ and Rust together
- Very opinionated and tries to produce straightforward and native-looking APIs for both languages
  - Tries to remove the ugliness of dealing with FFI
- Handles C++ compilation in Cargo as well, if you want
- <https://github.com/dtolnay/cxx>



# CXX

- Example can be found in KDE!
- See `akonadi-search`

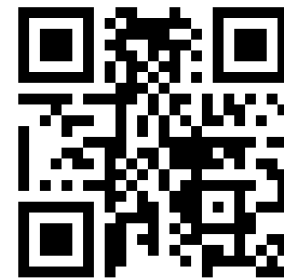
```
pub fn convert_to_text(html: String) -> String {  
    from_read(html.as_bytes(), html.len())  
}  
  
#[cxx::bridge]  
mod ffi {  
    extern "Rust" {  
        fn convert_to_text(html: String) -> String;  
    }  
}
```



It can be that  
simple? Wow!

# CXX-Qt (Sponsored )

- Created in 2021 by KDAB, still maintained by Andrew Hadzen and Leon Matthes
  - Updated continuously ever since
- Maintainers are receptive to contributions
  - No CLA!
- Is one of the largest and feature-filled Qt bindings in Rust currently
  - We have a comparison chart in the README
- <https://github.com/KDAB/cxx-qt>





# CXX-Qt (Sponsored )

```
#[cxx_qt::bridge]
pub mod qobject {
    ...
    unsafe extern "RustQt" {
        #[qobject]
        #[qml_element]
        #[qproperty(i32, number)]
        #[qproperty(QString, string)]
        type MyObject = super::MyObjectRust,
    }
    ...
}
```

Equivalent to Q\_OBJECT  
and QML\_ELEMENT

Equivalent to  
Q\_PROPERTY  
(Property getters/setters  
can be auto-created)

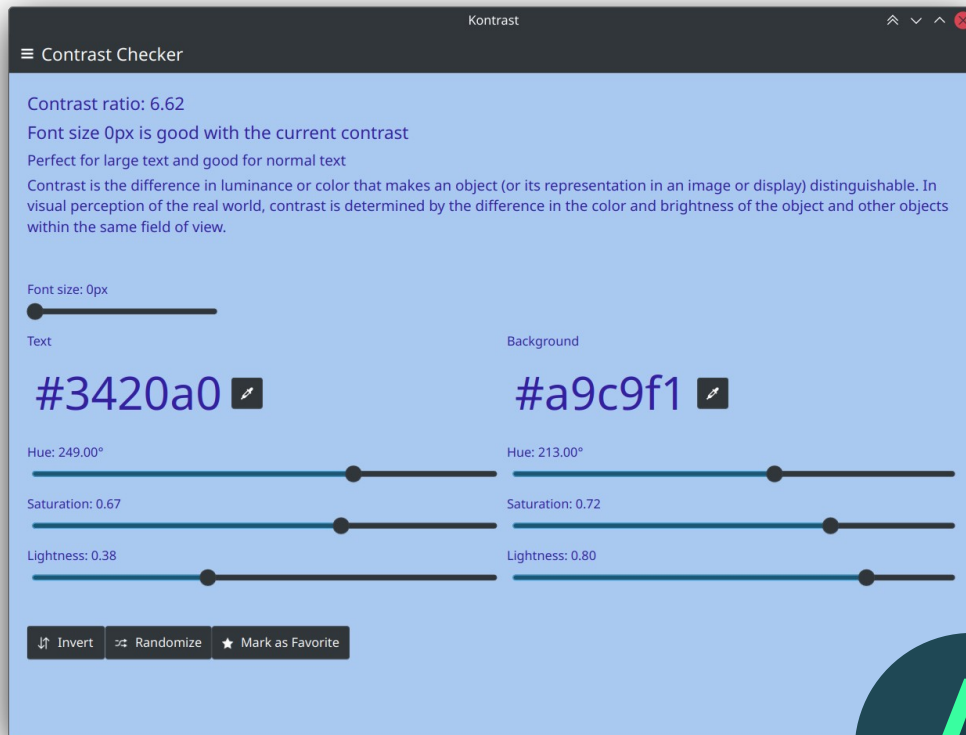
The QObject type  
(Exposed to C++)

The Rust type

# CXX-Qt (Sponsored )

- Lots of other Qt-isms are supported
  - Invokables via qinvokable
    - Usable from both Rust, C++, and QML
  - Overriding base methods (usually for QabstractItemModel)
  - Threading
  - QEnum, Qnamespace
  - Bindings for a lot of the Qt API, and more is always being added
  - Examples for most of these are in the cxx-qt repository

# Kontrast



# Kontrast

- Proof of concept showing what a CXX-Qt KDE application may look like
- Written by Darshan Phaldesai
- Features Rust + QML
- <https://github.com/mystchonky/kontrast-rs>



# Kontrast

- Since it's supposed to be the same application, we can do some naive comparisons!
- Backend code is a single class
  - C++: 127 LoC (header) + 362 LoC (source)
  - Rust: 339 LoC
- Comparable in LoC, but this is only a single and simple case

# cxx-kde-frameworks

- CXX bindings for some KDE Frameworks
  - Written by Darshan Phaldesai
  - Frameworks is ideal for this thanks to it's ABI and general API stability
- Still a work-in-progress, not supposed to be official
- <https://github.com/mystchonky/cxx-kde-frameworks>



# cxx-kde-frameworks

```
KLocalizedString::set_application_domain(&QByteArray::from("konstrast"));

let mut about_data = KAboutData::from(
    QString::from("konstrast"),
    i18nc("@title", "Konstrast"),
    QString::from("TEST"),
    i18nc("@title", "A constrast checker application. Now oxidized!"),
    License::GPL_V3,
);

KAboutData::set_application_data(about_data);

...

if let Some(mut engine) = engine.as_mut() {
    KLocalizedString::initialize_engine(engine.as_mut().as_qqmlengine());
    engine.load(&QUrl::from("qrc:/qt/qml/org/kde/konstrast/src/qml/Main.qml"));
}
```

loadFromModule is being worked on

# cxx-kde-frameworks

- Open questions:
  - If we want this to become official, should the bindings be centralized in one repository?
  - We should review which frameworks are typically used QML applications (on the C++ side)
  - Packaging and distribution



# Wrapping Up

# Conclusion

- Rust is a relatively new and exciting language for newer developers
- We have new and upcoming bindings for Qt and KDE Frameworks
- Rust can be offered as an option, not a replacement

# Join in!

- CXX-Qt developers are available on Zulip
  - <https://cxx-qt.zulipchat.com/>
- GitHub
  - <https://github.com/KDAB/cxx-qt>
- KDE Rust Matrix
  - [#kde-rust:kde.org](https://matrix.org/#kde-rust:kde.org)



# Q&A

- Questions?
- Slide deck is available online



CXX-Qt Getting Started

Thank you!  
joshua.goins@kdab.com