# Bindgen, cxx, cpp: Wege, um Rust ins Brownfield zu bringen

Philipp Bormuth // awinia GmbH

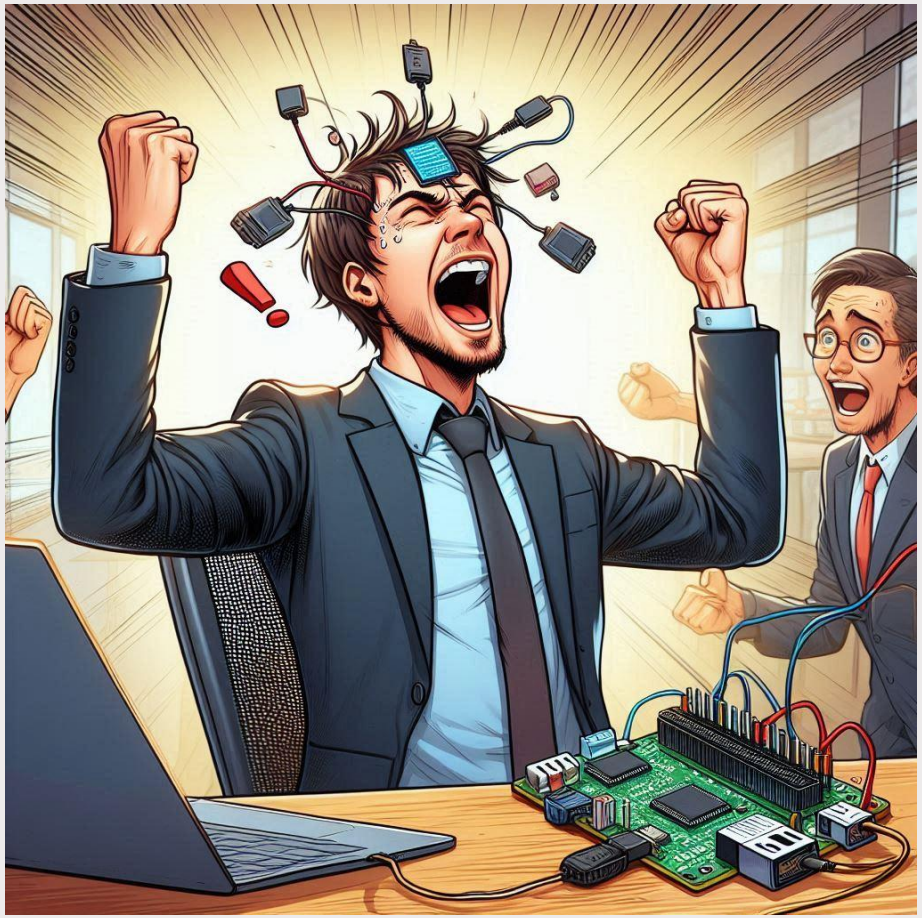
# Agenda

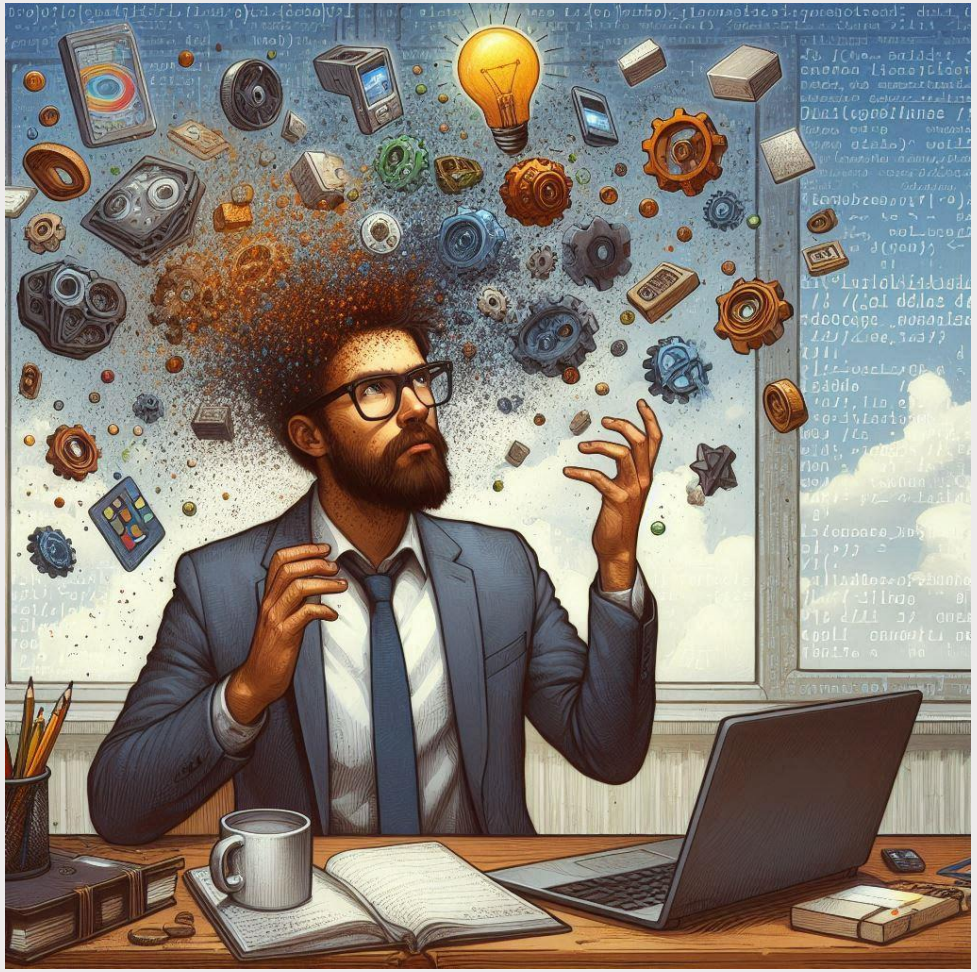
- Disclaimer
- Die Story
- Grundsetup
- Tools

# Disclaimer

- Viel Inhalt, deshalb teilweise etwas kürzer
- Detailfragen? Sprechen Sie mich nach dem Vortrag an!

# Die Story





# Grundsetup

- Rusts C-FFI
- Rust wird als statische C-Lib kompiliert
- Vorgabe:
  - C -> Rust
  - Rust -> C
  - Rust -> C++
  - C++ -> Rust

# C-Interop (1)

- Grundsätzlich einfach…

```
#[no_mangle]
extern "C" fn rs_main() {
    println!("Hello, world!");
}

extern void rs_main();
void main()
{
    rs_main();
}
```

# C-Interop (2)

- Andersrum…

```
extern "C" {
    pub fn k_sleep_rs(timeout: i32) -> i32;
}

int k_sleep_rs(int timeout) { /*…*/}
```

# Automatisieren mit bindgen

- bindgen
  - Erzeugt Rust-FFI Bindings aus Headerfiles
- cbindgen
  - Erzeugt C Headerfiles aus einem Rustmodul

# C-Interop (3)

- Kalamitäten
  - Präprozessor
  - static inline

# C-Interop (4)

```
#define SomeFunctionStyleMacro(a) void foo(int32_t a) { /*…*/}

void SomeFunctionSyleMacroCallableByRust(int32_t a)
{
        SomeFunctionStyleMacro(a)
}
```

# C++ Interop

„The only way to be compatible with C++ is to be a C++ compiler“

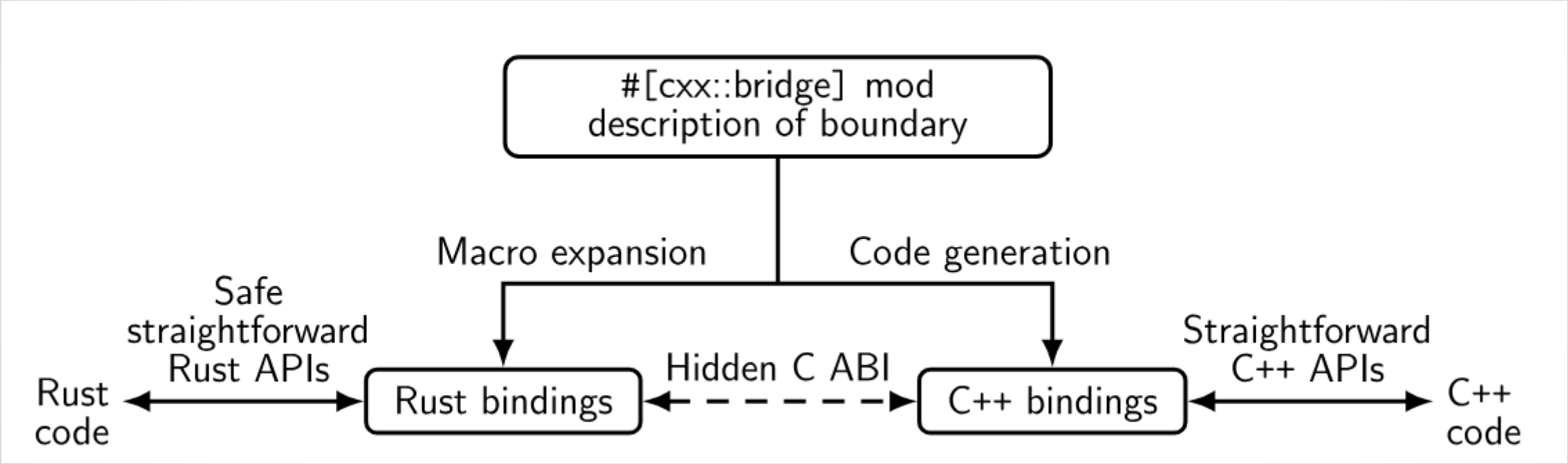
# C++ Interop

- Kalamitäten:
  - „Magic Functions“: C‘tor, CC‘tor, Move C‘tor, Assignments…
  - Name Mangling
  - Move vs. Copy Semantik
  - Exceptions
  - Sprachidiome, z.B. RAII
- Realistisch: C-nahes Interface mit opaken Typen
- Problem: Die API-Oberflächen für Bestandsprojekte sind oft riesig, hier ist leider Fleiß angesagt

# cxx(1)

*"provides a safe mechanism for calling C++ code from Rust and Rust code from C++. It carves out a regime of commonality where Rust and C++ are semantically very similar and guides the programmer to express their language boundary effectively within this regime."*

# cxx(2)


#[cxx::bridge] mod
description of boundary
Macro expansion
Code generation
Safe
straightforward
Rust APIs
Hidden C ABI
Straightforward
C++ APIs
Rust
code
Rust bindings
C++ bindings
C++
code

# CXX(3)

- Entwickler legen ein explizites Interface zwischen den Sprachen an (Muss gepflegt werden)
- Codegenerator erzeugt Gluecode in beide Richtungen

# Einschränkungen

- Ziemlich viel Fleißarbeit nötig bei großer Klassenhierarchie
- Speicherallokationen können die Sprachgrenze nicht übertreten
- Weiterhin: Keine Magicfunctions in Rust (bzw. nur sehr eingeschränkt)
- Heapless: 😟
- Lifetimes: 😱

# Problembeispiel

# Lösungsansätze

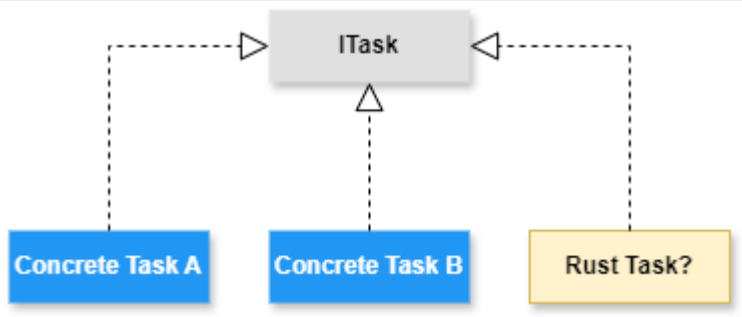
- Pools
  - Gut für langlebige Objekte, die i.d.R. nicht zerstört werden müssen.
  - Ansonsten:
    - Explizites Instanzmanagement auf der Rustseite nötig
    - Bei vielen Objekten: U.U. schwierig die Poolgröße korrekt zu ermitteln (siehe auch Beispiel)
  - Trotzdem RAII? Mehr Arbeit auf der Rustseite nötig.

# autocxx

- Maximale Automatisierung
- Einfach zu verwenden
- no_std/heapless: 😟

# Integration in eine Objekthierarchie(1)

- Die gezeigten Werkzeuge ebnen den Weg für Interoperabilität
- i.d.R. gehört zu einer echten Integration aber mehr



# Integration in eine Objekthierarchie(2)

- C++ Applikationen erwarten i.d.R., dass Erweiterung durch Vererbung bzw. Implementierung von Interfaces passiert.
- Wie implementieren wir ein C++ Interface in Rust?!?!

# Letztes Beispiel ….versprochen

# TL;DL;

(too long, didn‘t listen)

- C Interop ist i.d.R. kein Problem
- C++ bietet Raum für Kreativität

# Vielen Dank

www.awinia.de