KU LEUVEN

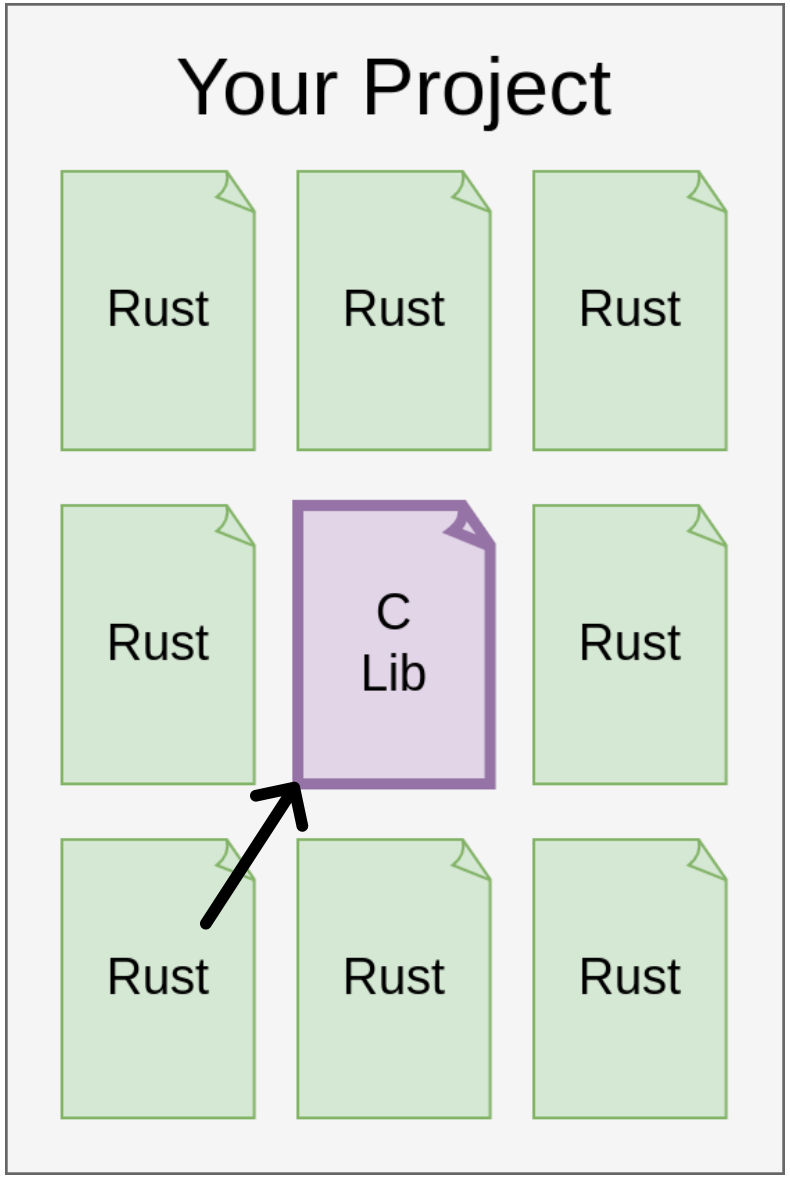
DistriNet

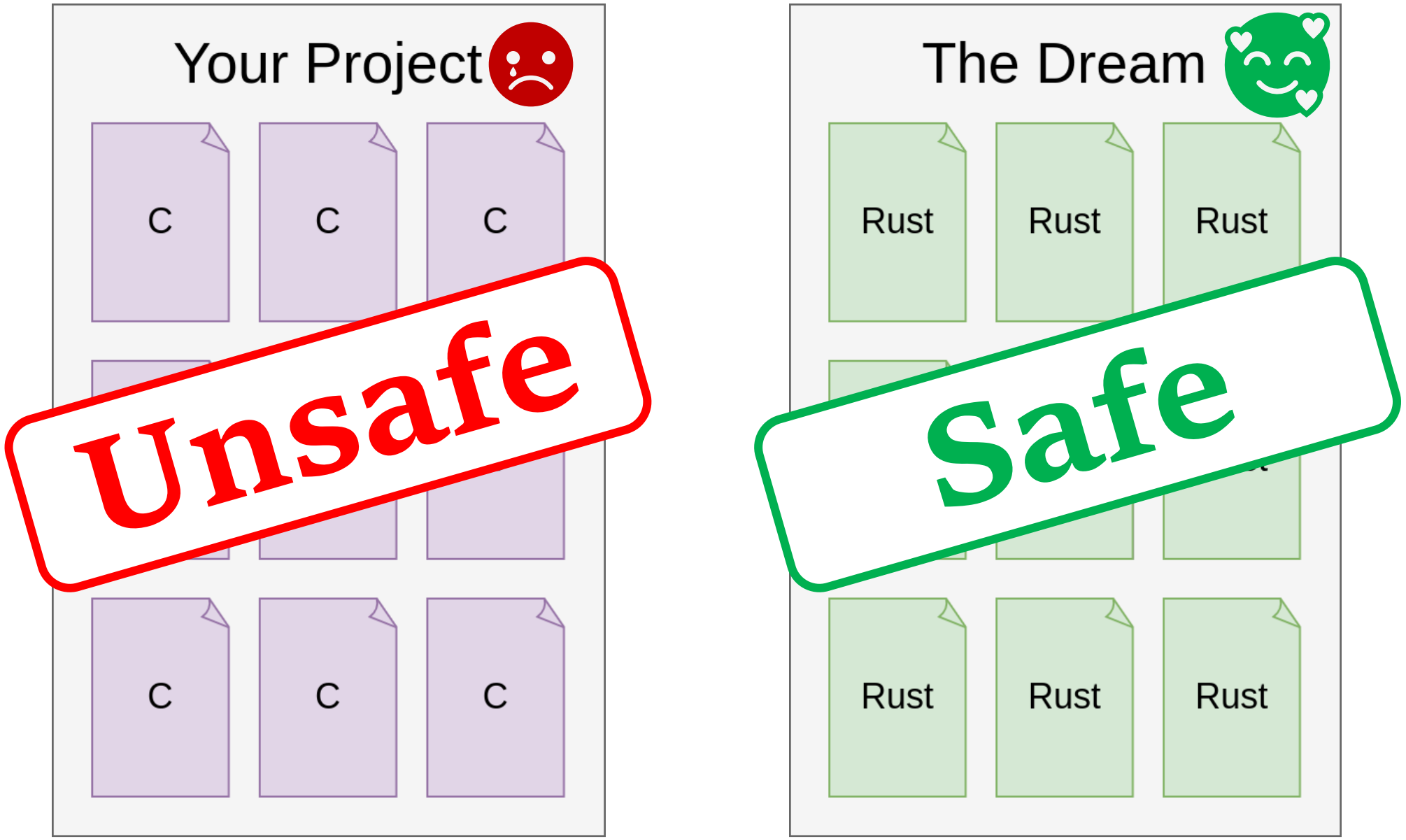
Incremental Migration from C to Rust

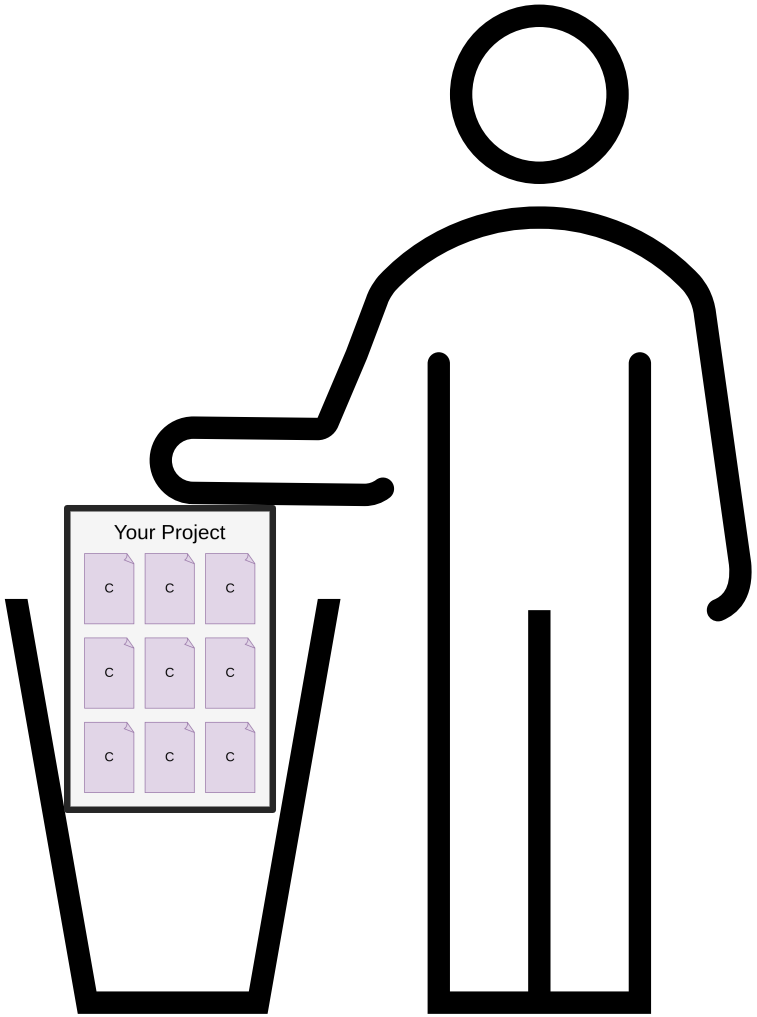
C and Rust Interoperability

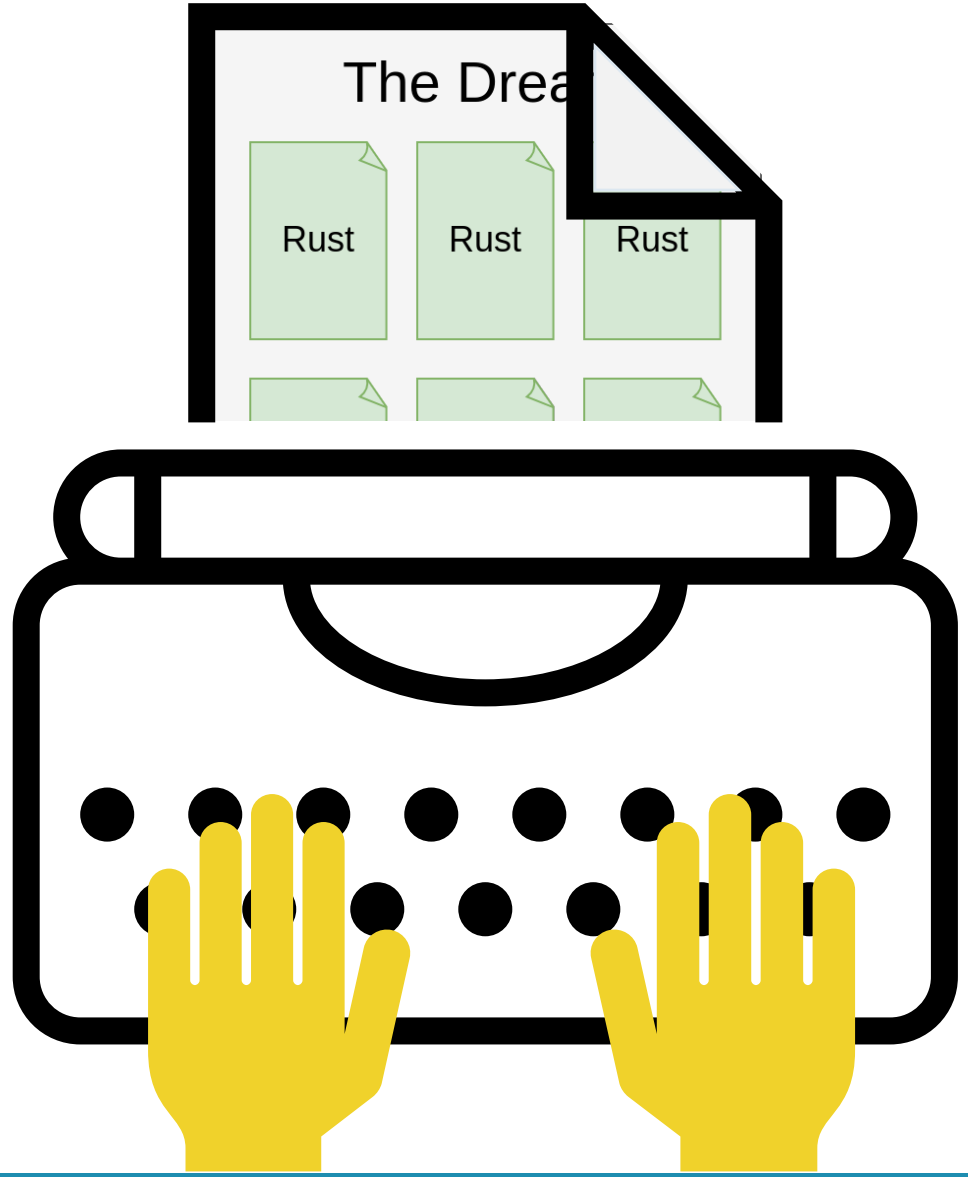
Alicia Andries

alicia.andries@kuleuven.be

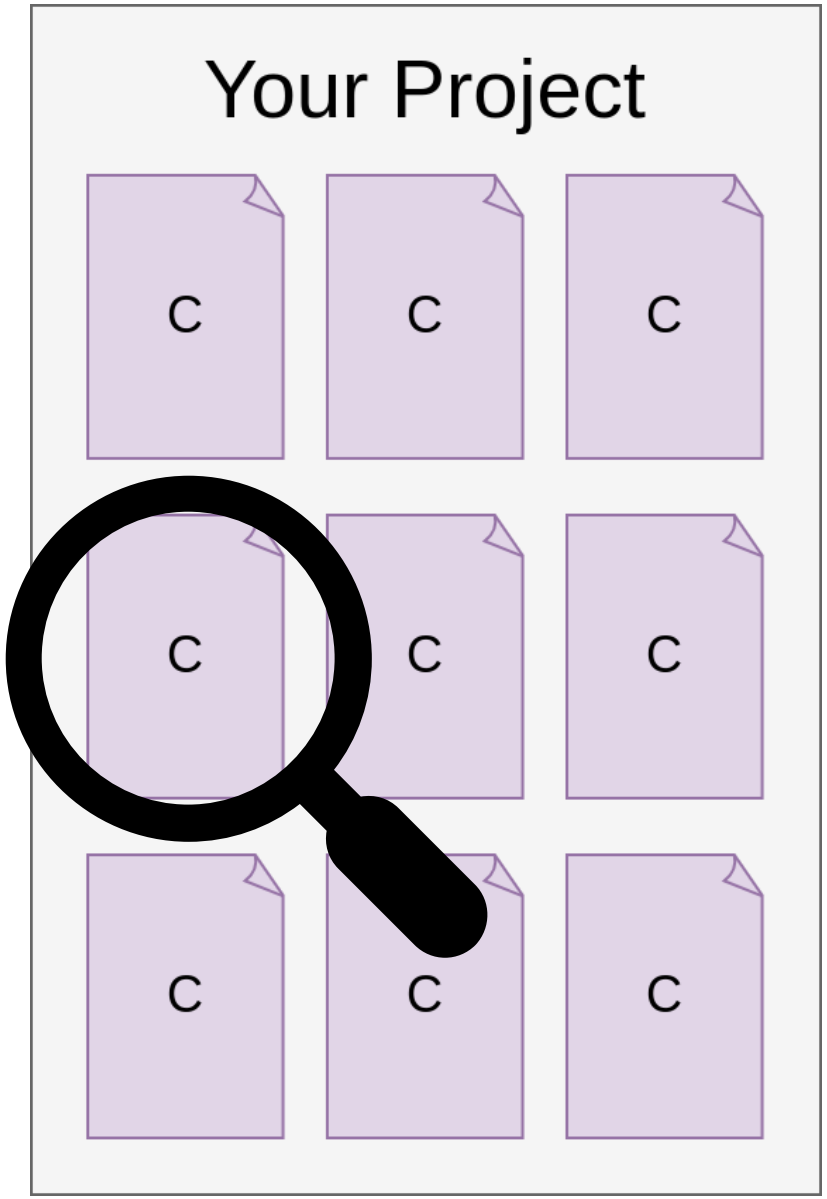


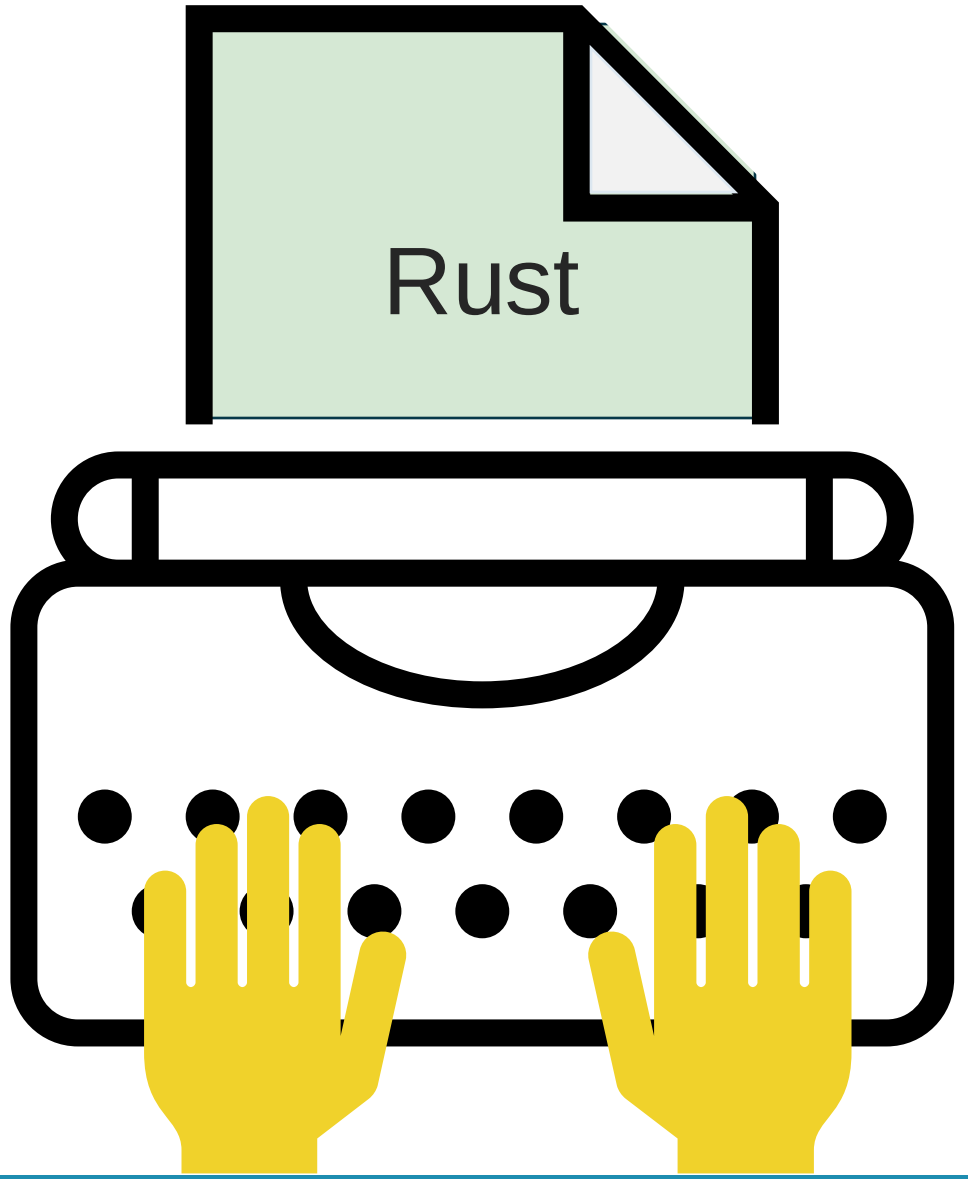

Your Project
C
C
C
C
C
C
C
C
C
Unsafe
The Dream
Rust
Rust
Rust
Rust
Rust
Rust
Rust
Rust
Safe

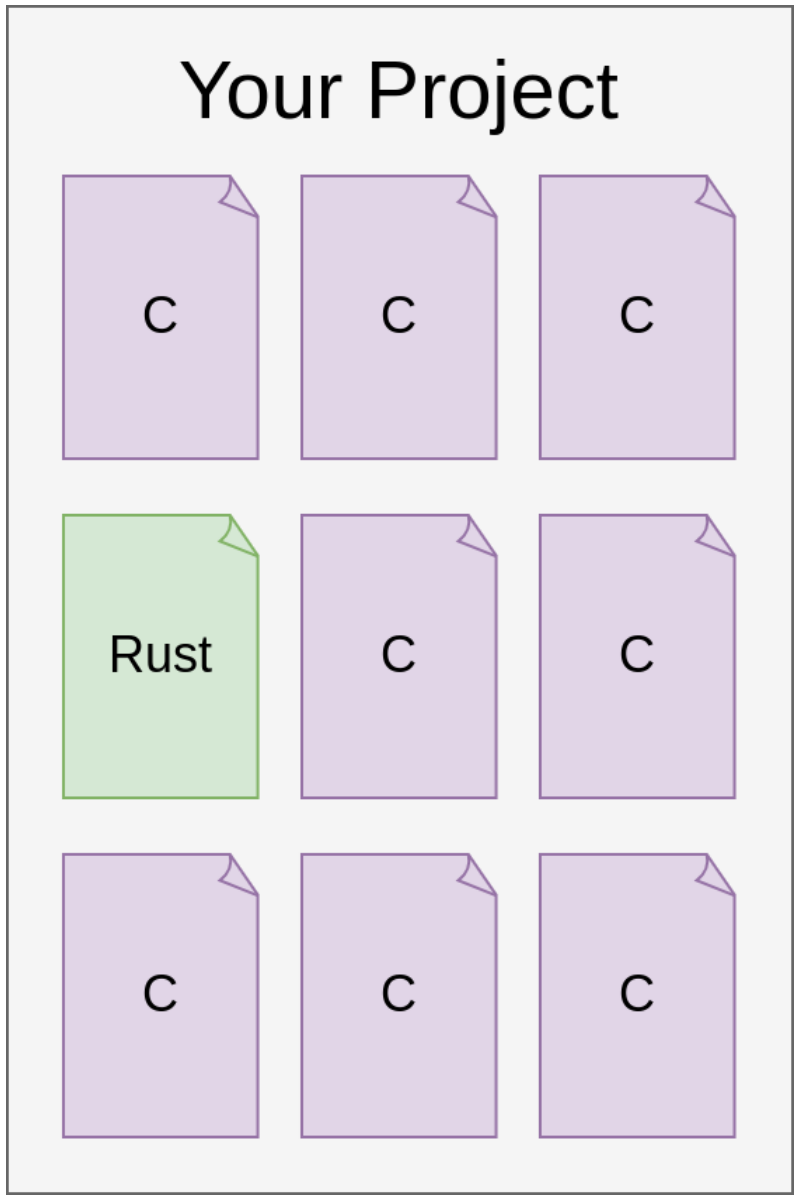


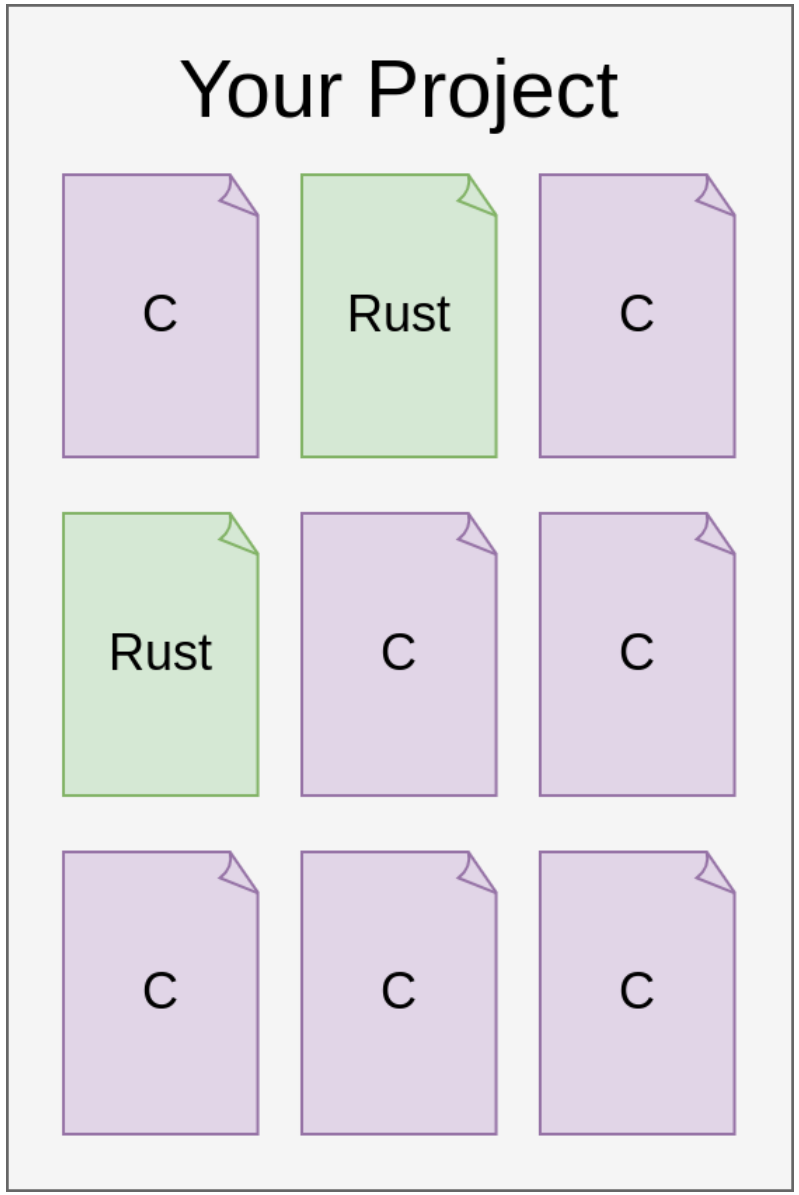


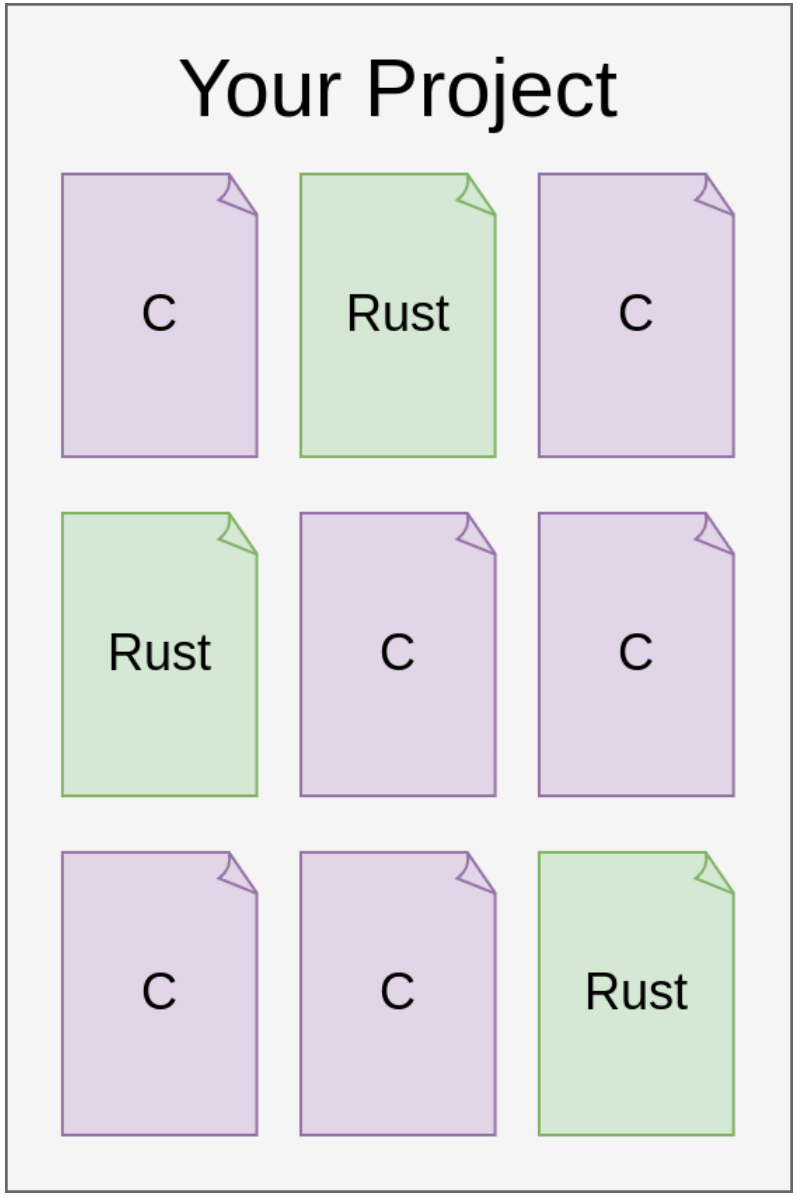
KU LEUVEN

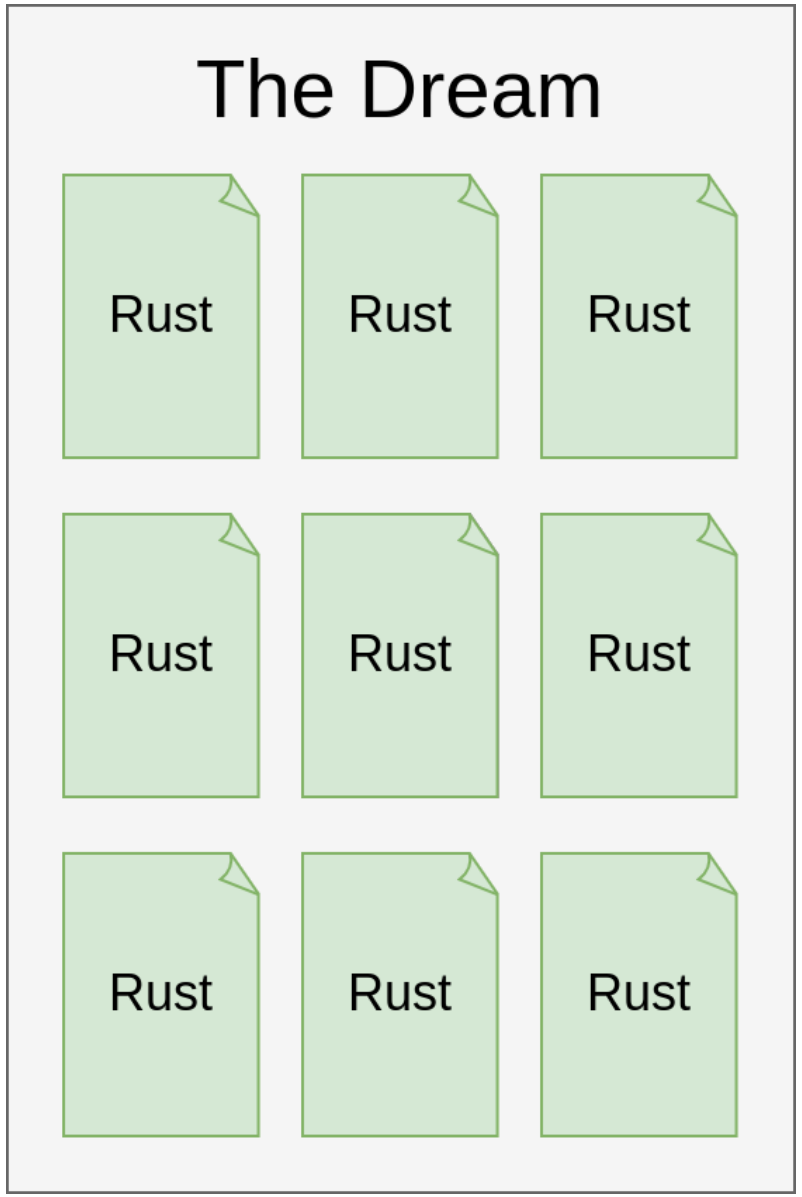

Your Project
C
C
C
C
C
C
C
C
C
C
C
C


Rust

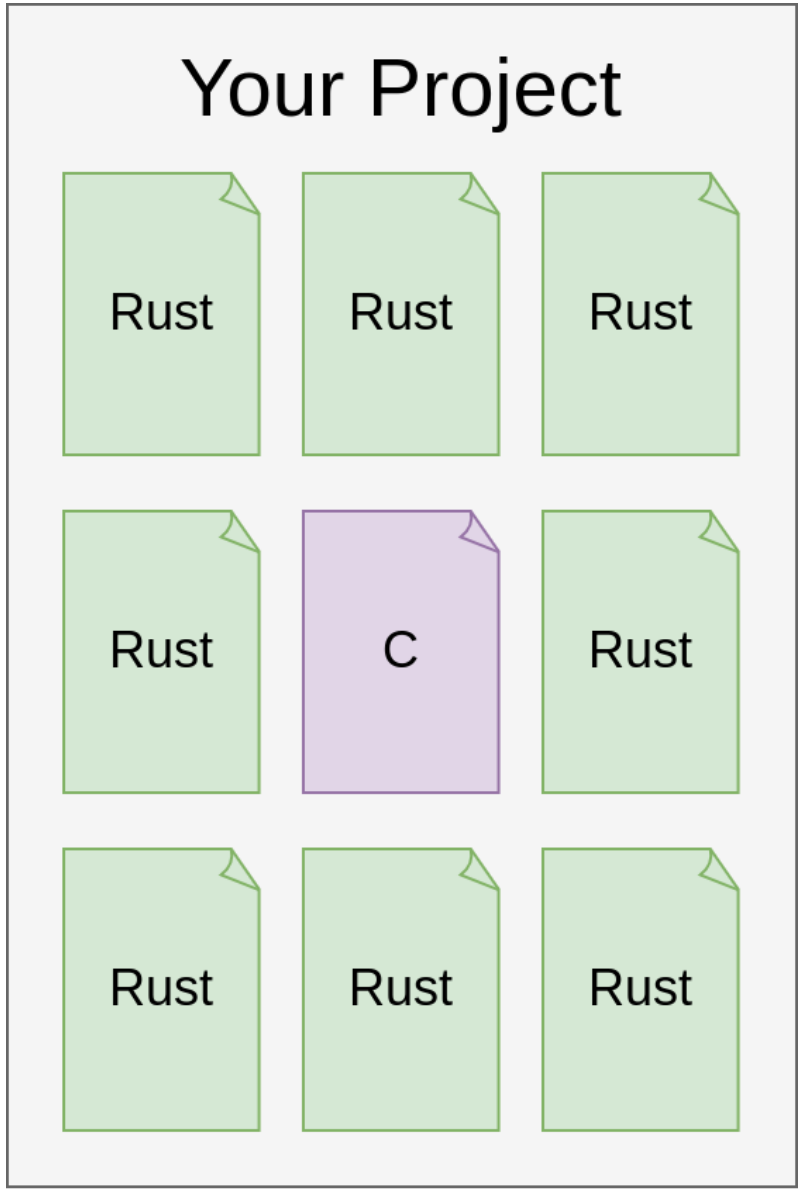





Your Project
C
Rust
C
Rust
C
C
C
C
Rust


The Dream
Rust
Rust
Rust
Rust
Rust
Rust
Rust
Rust
Rust

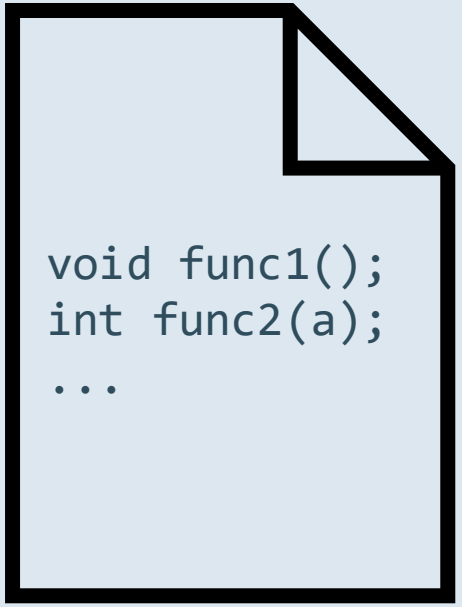
How to Call C Code From Rust?



Call C Code from Rust

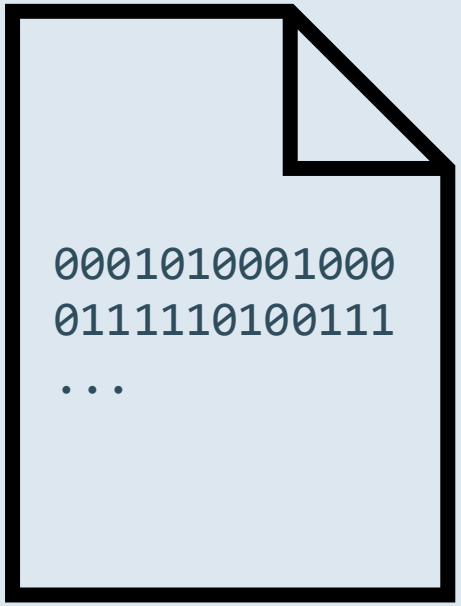
Interface

interface.h



Shared/Static Library

library.so



Rust Binding From C Code

doggo.h

```c
typedef struct doggo {
    int many;
    char wow;
} doggo;

void grade(doggo* pupper);
```

doggo.rs

```rust
#[repr(C)]
pub struct doggo {
    pub many: ::std::os::raw::c_int,
    pub wow: ::std::os::raw::c_char,
}

extern "C" {
    pub fn grade(pupper: *mut doggo);
}
```

main.rs

```rust
fn main() {
    let dog = Box::new(doggo{10,10});
    unsafe{
        grade(dog.into_raw());
    }
}
```

bindgen

```
$ bindgen doggo.h -o doggo.rs
```

cargo.toml

```
[build-dependencies]
bindgen = "0.65.1"
```

wrapper.h

```
#include "doggo.h"
#include "cat.h"
```

build.rs

```
bindgen
```

https://rust-lang.github.io/rust-bindgen/tutorial-3.html

Build Integration

Cargo.toml

```
[build-dependencies]
cmake = "0.1"
```

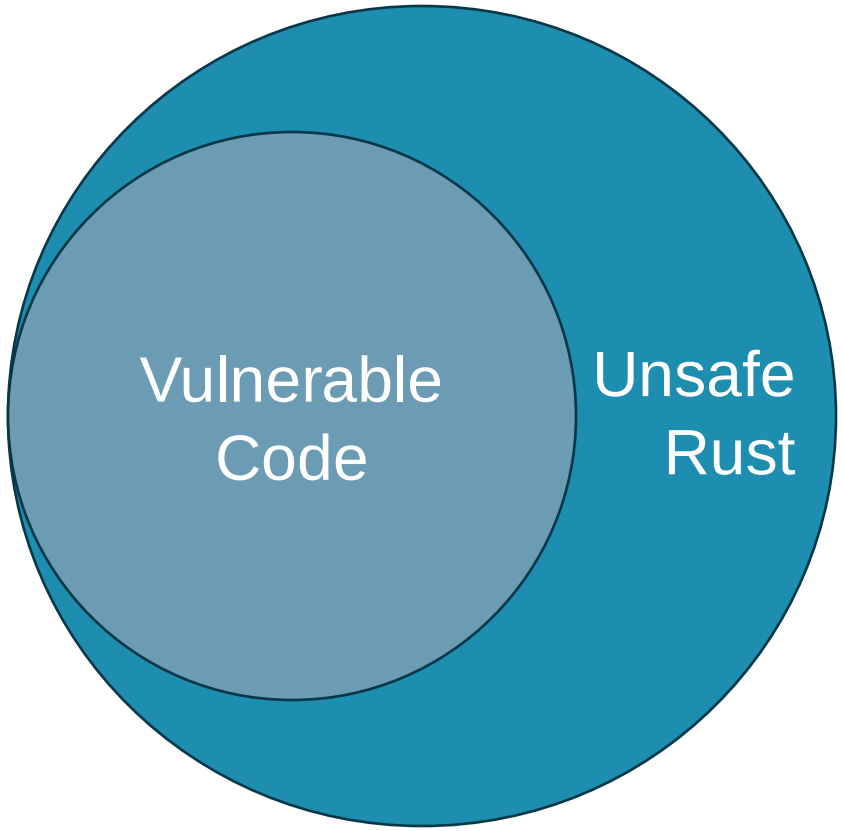
Build.rs

```
use cmake::Config;

let dst = Config::new("libfoo")
                 .define("FOO", "BAR")
                 .cflag("-foo")
                 .build();
println!("cargo:rustc-link-search=native={}", dst.display());
println!("cargo:rustc-link-lib=static=foo");
```


https://crates.io/crates/cmake

Unsafe Rust
≠
Vulnerable Code*



* This is theoretically true but practically probably not

How to Deal With *Unsafe* C

doggo.rs

```rust
#[repr(C)]
pub struct doggo {
    pub many: ::std::os::raw::c_int,
    pub wow: ::std::os::raw::c_char,
}

extern "C" {
    pub fn grade(pupper: *mut doggo);
}
```

main.rs

```rust
fn main() {
    let dog = Box::new(doggo{10,10});
    unsafe{
        grade(dog.into_raw());
    }
}
```


https://doc.rust-lang.org/nomicon/ffi.html

How to Deal With *Unsafe* C

doggo.rs

```rust
#[repr(C)]
pub struct doggo {
    pub many: ::std::os::raw::c_int,
    pub wow: ::std::os::raw::c_char,
}

extern "C" {
    pub fn grade(pupper: *mut doggo);
}
```

doggo_wrapper.rs

```rust
pub fn safe_grade(pupper: &mut doggo){
    // # Safety
    // Calling a foreign function with a
    // valid pointer
    unsafe{
        grade(doggo as *mut doggo);
    }
}
```

main.rs

```rust
fn main() {
    let dog = Box::new(doggo{10,10});
    safe_grade(dog);
}
```


https://doc.rust-lang.org/nomicon/ffi.html

What Makes a Function *Unsafe*?

Arguments exist that can induce memory unsafe behaviour

```
pub fn safe_grade(pupper: &mut doggo){
    // # Safety
    // Calling a foreign function with a
    // valid pointer
    unsafe{
        grade(doggo.into_raw());
    }
}
```


https://doc.rust-lang.org/nomicon/ffi.html

doggo.h

```
doggo doggos[SIZE];

void insert(doggo pupper, int index){
    doggos[index] = pupper;
}
```

doggo.rs

```
extern "C" {
    /// # Safety
    /// The index should always be smaller than "SIZE"
    fn insert(pupper: doggo, index: i32);
}
```

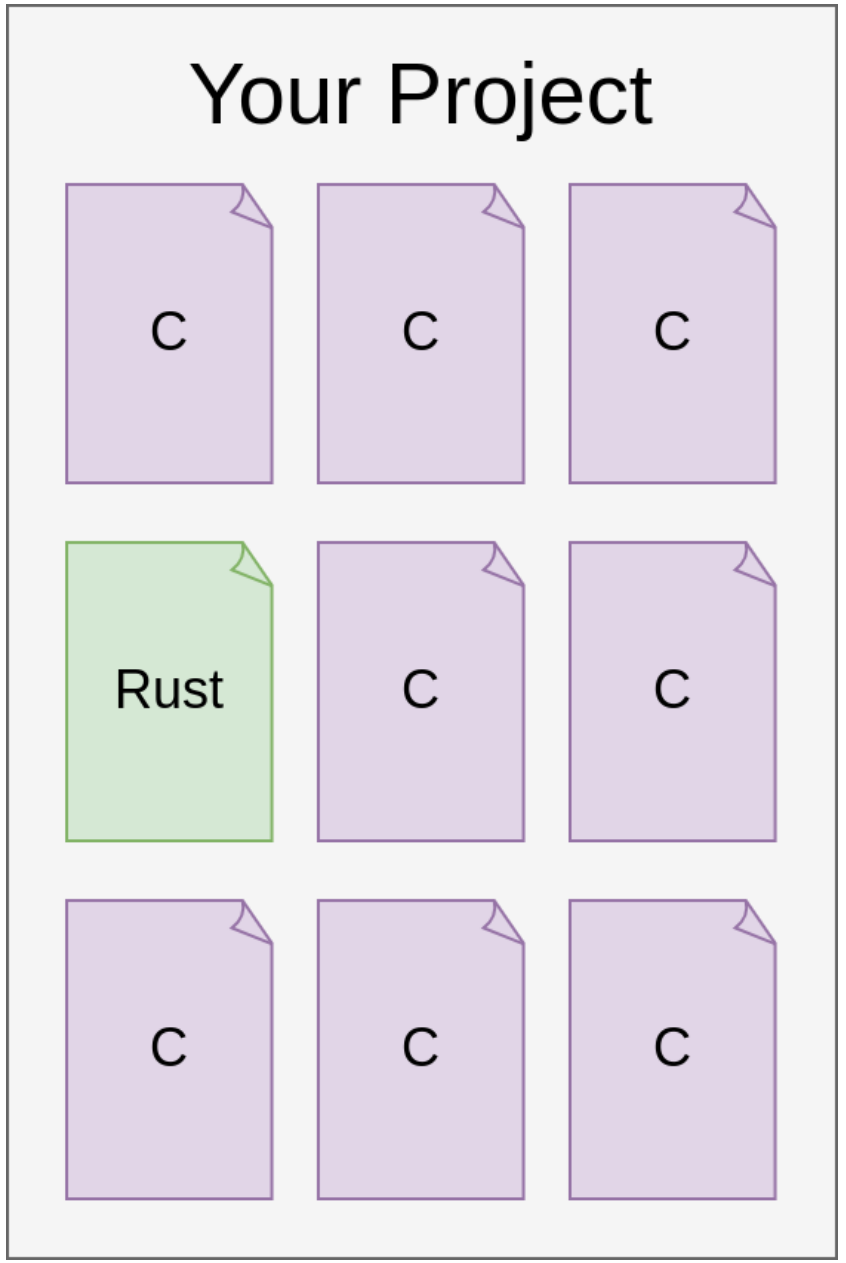
doggo.rs

```
extern "C" {
    /// # Safety
    /// Index must be smaller than "SIZE"
    fn insert(pupper: doggo, index: i32);
}
```

main.rs

```
fn main() {
    let dog = Box::new(doggo{many: 10, wow: 'a' });
    // # Safety:
    // This is safe because the index is smaller than
    // "SIZE"
    unsafe{
        insert(dog, 8);
    }
}-
```

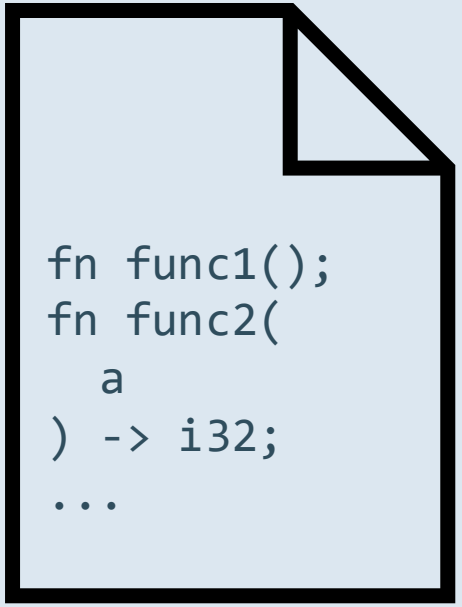
How to Call Rust From C Code



Call Rust Code from C

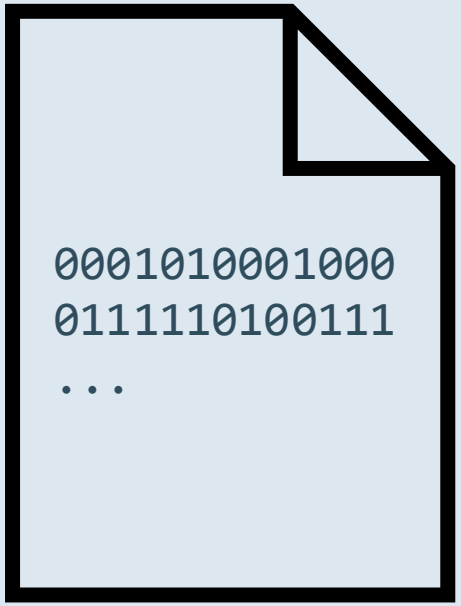
Interface

interface.rs



Shared/Static Library

library.so



C Binding From Rust Code

doggo.rs

```
pub struct doggo {
    pub many: ::std::os::raw::c_int,
    pub wow: ::std::os::raw::c_char,
}

pub fn grade(pupper: *mut
    doggo);
}
```

doggo.rs

```
#[repr(C)]
pub struct doggo {
    pub many: ::std::os::raw::c_int,
    pub wow: ::std::os::raw::c_char,
}

#[no_mangle]
pub extern "C" fn grade(pupper: *mut
    doggo);
}
```

C Binding From Rust Code

doggo.rs

```rust
#[repr(C)]
pub struct doggo {
    pub many: ::std::os::raw::c_int,
    pub wow: ::std::os::raw::c_char,
}

#[no_mangle]
pub extern "C" fn grade(
    pupper: *mut doggo
);
```

doggo.h

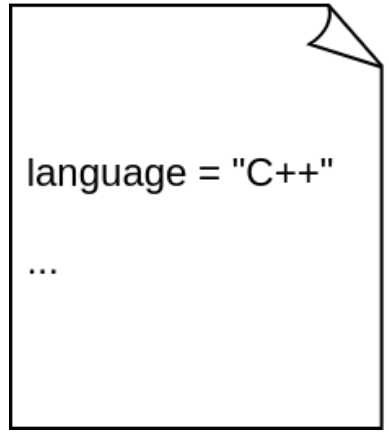
```c
typedef struct doggo {
    int many;
    char wow;
} doggo;

void grade(doggo* pupper);
```

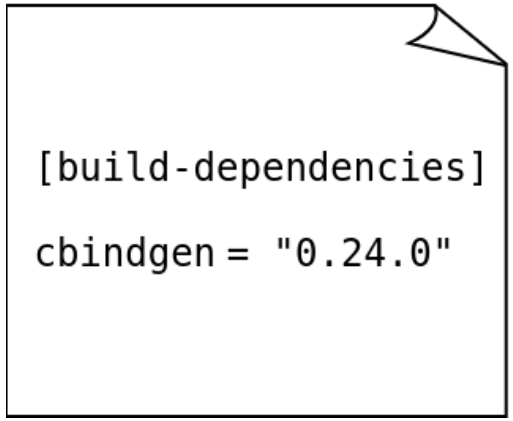
cbindgen

```
$ cbindgen --config cbindgen.toml --crate my_rust_library
> --output my_header.h
```

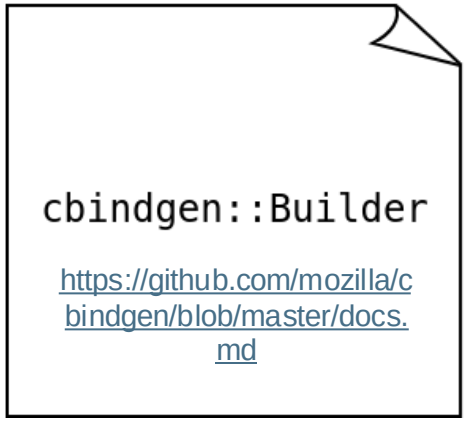
cbindgen.toml



cargo.toml



build.rs



https://github.com/mozilla/cbindgen/blob/master/docs.md

Build Rust

Cargo.toml

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

rustc

```
rustc --crate-type=dylib doggo.rs
```

Build System Integration

CMakeLists.txt (simplified)

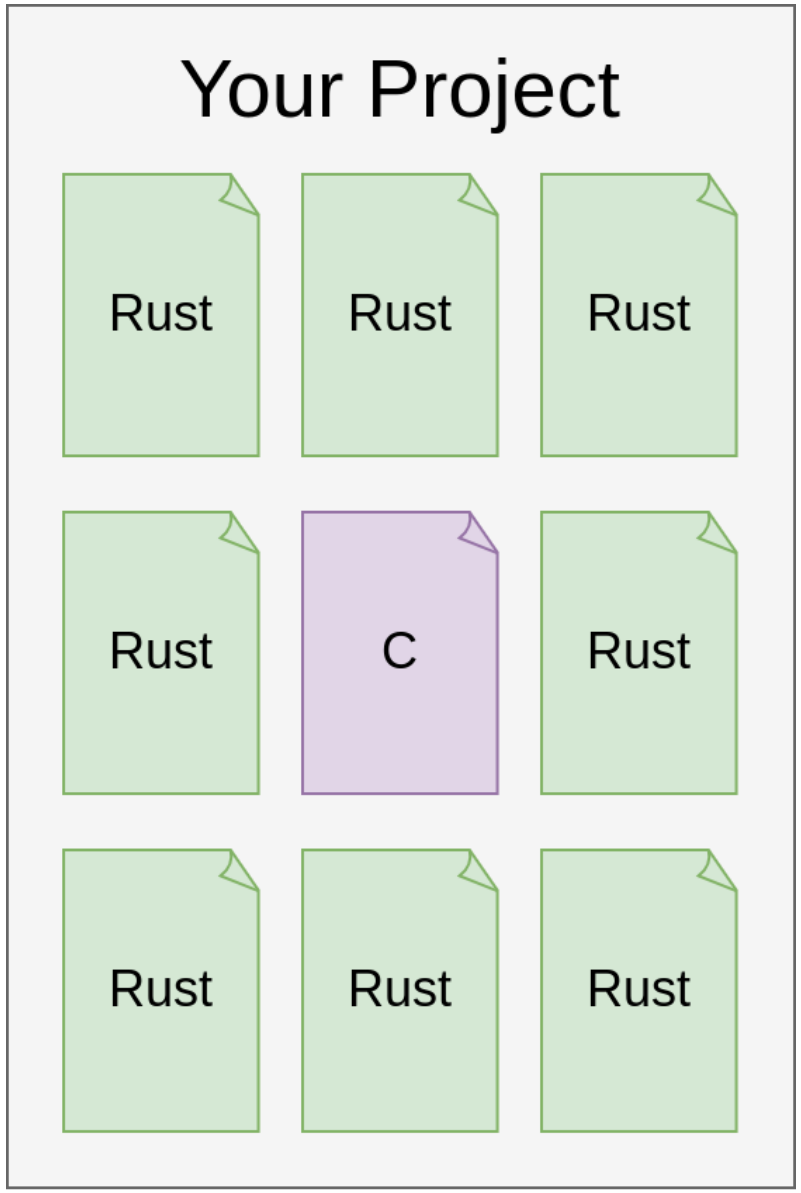
```
add_custom_target(vector ALL
  COMMENT "Compiling vector"
  COMMAND CARGO_TARGET_DIR=${CMAKE_CURRENT_BINARY_DIR} ${CARGO_CMD}
)
```

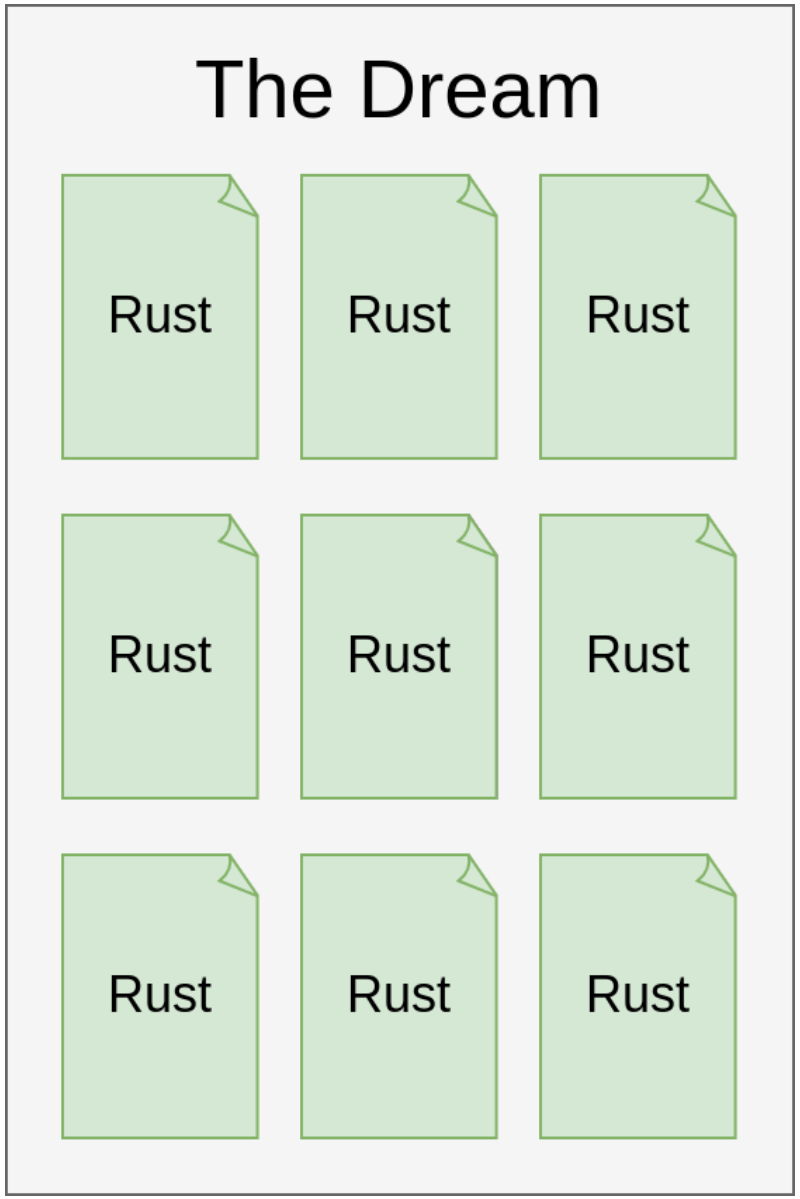
Conventions

- Add comments that explain what inputs are valid

doggo.rs

```
/// # Safety
/// Do not pass null pointer or otherwise invalid pointer
#[no_mangle]
pub extern "C" fn grade(pupper: *mut
    doggo);
```


Your Project
Rust
Rust
Rust
Rust
C
Rust
Rust
Rust
Rust

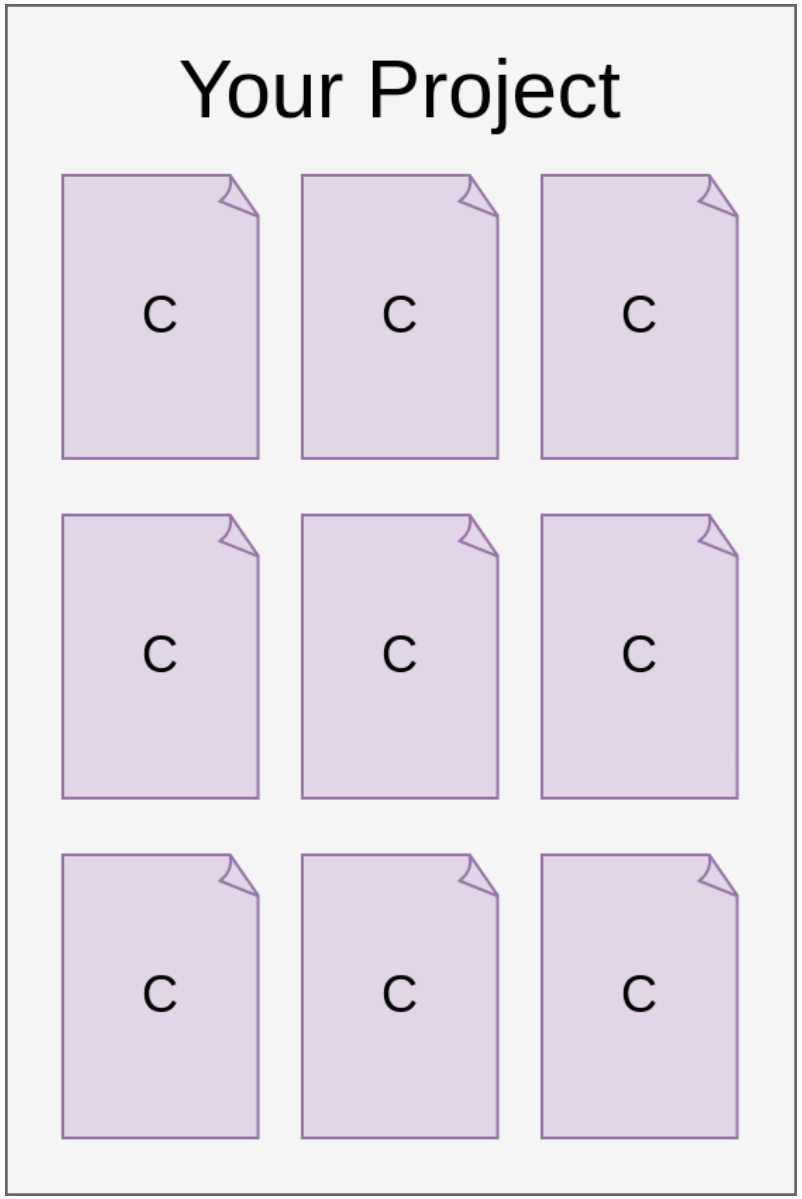

The Dream
Rust
Rust
Rust
Rust
Rust
Rust
Rust
Rust
Rust

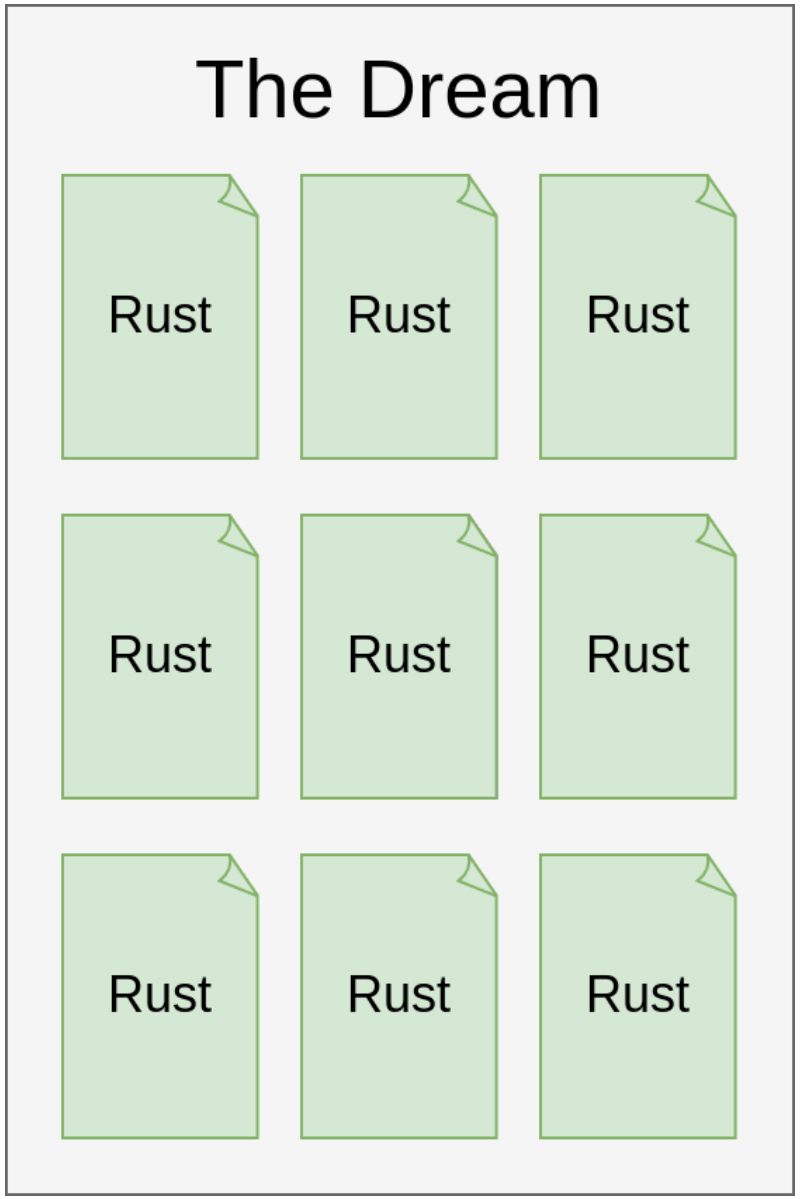
C2Rust

- Syntactic translation
- Emits *unsafe* code
- Uses nightly only features

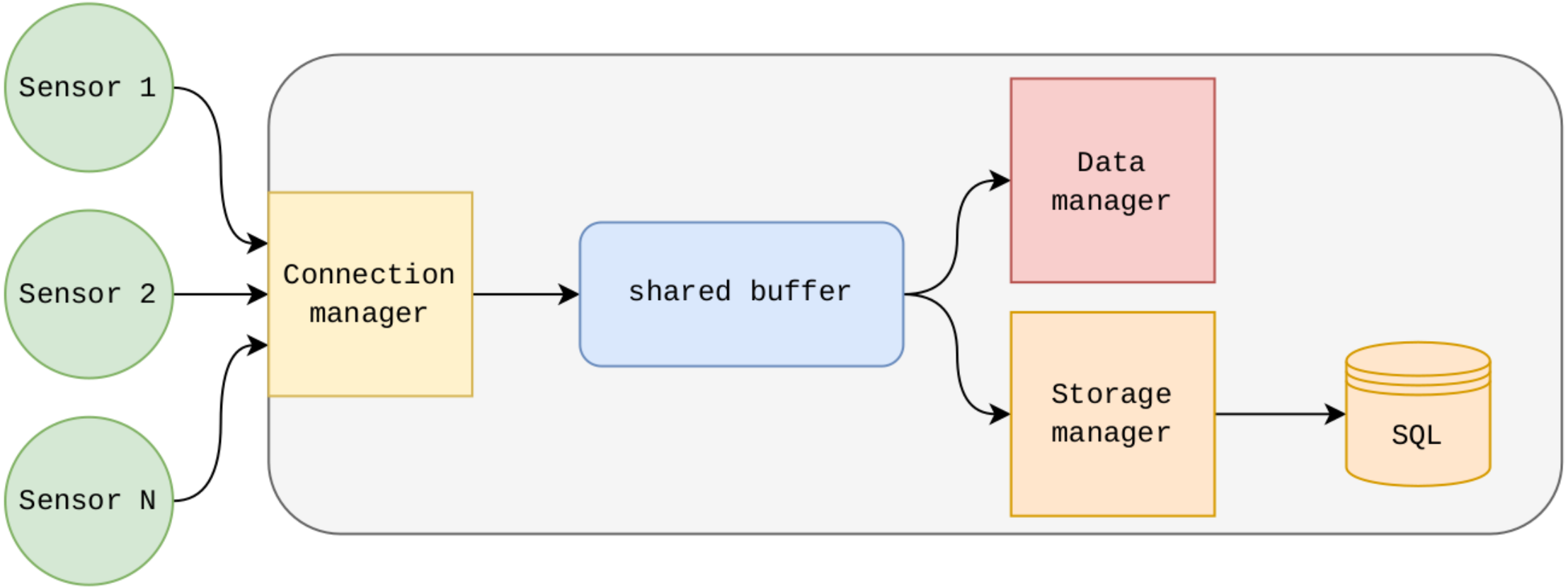
https://c2rust.com/

The Workshop


Your Project
C
C
C
C
C
C
C
C
C
C
C
C


The Dream
Rust
Rust
Rust
Rust
Rust
Rust
Rust
Rust
Rust

Workshop


Sensor 1
Sensor 2
Sensor N
Connection manager
shared buffer
Data manager
Storage manager
SQL

Difficulties

References Versus Pointers

Is a pointer valid or not?

Rust Lifetime System

What if Rust frees an object C is still using?

Type Incompatibilities

size_t <-> usize

int <-> i32

Wrapping <-> Not wrapping

Questions?