

Swift + Rust

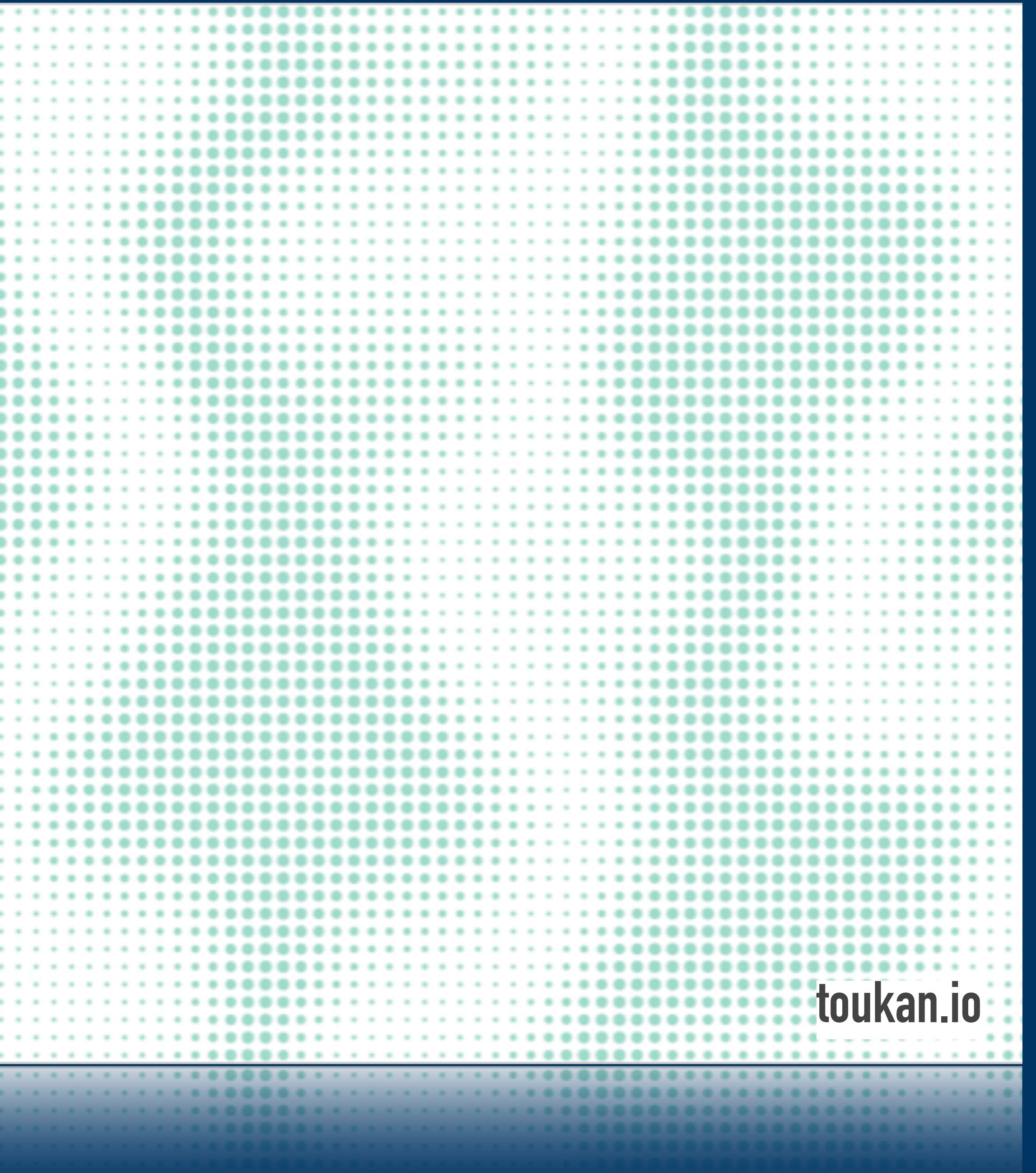
Comment intégrer du code Rust dans des projets Swift

Quentin Mathé

Toukan

Éditeur vectoriel + Design génératif

- Rendre accessible le design génératif à tous les designers
- Challenges
 1. Expérience utilisateur
 2. Portabilité (macOS, iOS, web, Windows)
 3. Performance
- Usage de Rust
 - Conversion entre espaces de couleur → [palette](#)
 - Génération de bruit → [noise.rs](#)
 - Tessellation → [lyon](#)
 - Rendu avec WebGPU → [wgpu](#)





Pourquoi Rust ?

Écosystème Swift focalisé sur la
création d'applications



Swift ne peut pas encore remplacer le C

C et C++ propose des bibliothèques à la pelle

- Peu bibliothèques de bas niveau
 - traitement de l'image et du son
 - mathématiques et 3D
 - structure de données
- Maintenance incertaine des bibliothèques existantes
- Performance moins bonne (sans profiling)
 - exemple → compteur atomique avec ARC

Comparaison écosystèmes Rust et Swift

Bibliothèques pour le rendu graphique

- Algèbre linéaire
 -  → euclid (Servo), glam, vectora, nalgebra, cgmath etc.
 -  → LASwift, LANumerics
- Courbes de Bézier
 -  → lyon_geom, flo_curves et vek
 -  → BezierKit (en développement)
- Gestion de la couleur
 -  → qcms (Firefox), palette
 -  → 😢

Écosystème Rust focalisé sur la
programmation système et les besoins du
web



Qu'est-ce qu'apporte Rust en plus ?

Ou en moins par rapport à Swift, C et C++

- Tooling
 - un seul outil pour tout faire (cargo) → compilation, dépendance, documentation, test etc.
- Modularité
 - très modulaire (plus complexe) → contrairement à Swift où chaque bibliothèque est un module
 - plus précis et optimisable → la bibliothèque standard n'est pas un bloc monolithique
- Portabilité
 - plus facile qu'avec C ou C++
 - plus de plateformes supportés que Swift (WASM, embarqué etc.)

D'autres choses encore ?

Quelques exemples

- Maturité → compilateur plus stable, bibliothèque standard de qualité
- Performance → analogue à C et C++ et usage mémoire réduit par rapport à Swift
- Macro → `vec![1, 2, 3];`
- Expression → `let x = if y > 0 { 0 } else { y }`
- Ownership et sécurité → unicité des références mutables
- Concurrence → modèle proche de celui prévu pour Swift 6
- Trait et generics → moins flexible que les protocoles et generics de Swift

Rust vous autorise à décider de tout

Contrairement à Swift

- zero optimisations magiques (ARC, heap allocation, primitive boxing)
- contrôle précis du compilateur
 - représentation données en mémoire
 - allocation de la mémoire (heap et stack, retain count, copy on write)
 - convention d'appel des fonctions
- intégration facilité avec d'autres langages

Rust avec Xcode



Installer Rust

Pour créer des binaires universels iOS et macOS

```
$ curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh  
$ rustup target add aarch64-apple-ios aarch64-apple-ios-sim x86_64-apple-ios  
$ rustup target add aarch64-apple-darwin x86_64-apple-darwin
```

- install and uninstall platform targets
 - **rustup add/remove**
- list the available platform targets
 - **rustup target list**
- show the installed platform targets
 - **rustup show**

Créer une bibliothèque Rust

Utiliser la commande cargo

- Créer bibliothèque
 - cargo new --lib <rust_project_name>
- Compiler
 - cargo build → récupère les dépendances automatiquement
- Tester
 - cargo test
- Formatter
 - cargo fmt
- Linter
 - cargo clippy

Configurer Xcode

Build Settings à ajouter

- Objective-C Bridging Header
- Other Linker Flags
 - -l<rust_project_name>
- Library Search Paths
 - <rust_project_dir>/target/debug
 - <rust_project_dir>/target/x86_64-apple-darwin/release
 - <rust_project_dir>/target/aarch64-apple-darwin/release
 - <rust_project_dir>/target/x86_64-apple-ios/release (simulator)
 - <rust_project_dir>/target/aarch64-apple-ios/release

Automatiser la compilation

Ajout d'une build phase dans Xcode

```
set -e
source "$HOME/.cargo/env"
cd rusty
if [[ -n "${DEVELOPER_SDK_DIR:-}" ]]; then
    export LIBRARY_PATH="${DEVELOPER_SDK_DIR}/MacOSX.sdk/usr/lib:${LIBRARY_PATH:-}"
fi
if [ $CONFIGURATION = "Release" ]; then
    cargo build --release --target=x86_64-apple-darwin
    cargo build --release --target=aarch64-apple-darwin
    cargo build --release --target=x86_64-apple-ios
    cargo build --release --target=aarch64-apple-ios
else
    cargo build # macOS
    cargo build --target=x86_64-apple-ios # iOS (Intel simulator)
    cargo build --target=aarch64-apple--ios-sim # iOS (ARM simulator)
fi;
cbindgen . -o target/rust-generated-header.h
```



Bridging entre Rust et Swift

C

Swift et Rust ne peuvent pas se parler directement

Appeler une fonction globale Rust depuis Swift

Pour l'instant, c'est simple 😊

Rust

```
#[no_mangle]
pub extern "C" fn average(a: f64, b: f64) -> f64 {
    (a + b) / 2.0
}
```

Header

```
double average(double a, double b);
```

Swift

```
result = average(a, b)
```

cbindgen

Générer un header C à partir du code Rust

cbindgen.toml pour configurer la génération du header

- language = "C"
 - C++ est le choix par défaut
- prefix = "Rusty"
 - permet d'éviter les conflits de symboles → RustyPoint au lieu de Point
 - récupération du nom original côté Swift → typealias Point = RustyPoint
 - autre approche → modifier les includes avec *noincludes* et *sysincludes*

⚠ Une struct doit être accompagnée d'une fonction pour être exportée par cbindgen.

cbindgen et Swift

Attribut spécifique pour cbindgen.toml

Config

```
swift_name_macro = "CF_SWIFT_NAME"  
sys_includes = ["CoreFoundation/CFBase.h"]
```

Rust

```
impl Point {  
    #[no_mangle]  
    #[export_name = "Point_distance"]  
    pub extern "C" fn distance(&self, other: Point) -> f64 {  
        (self.x - other.x).hypot(self.y - other.y)  
    }  
}
```

Header

```
double Point_distance(const struct RustyPoint *self, struct RustyPoint other)  
CF_SWIFT_NAME(Point.distance(self:other:));
```



Pas très utile en pratique

- force les noms d'arguments en keywords
- inutile avec les pointeurs opaques, puisqu'ils sont généralement wraprés dans une struct Swift
- force un pointeur pour le premier argument quand on annote une fonction attachée à une struct

Démo



Si un language alloue de la mémoire, il est responsable de la désallouer

Passer une String de Swift à Rust

Côté Swift, le compilateur génère le code de conversion

Swift

```
pass_c_string_to_rust(swiftString)
```

Header

```
void pass_c_string_to_rust(const char* string);
```

Rust

```
pub extern "C" fn pass_c_string_to_rust(ptr: *const c_char) {
    let str = unsafe {
        CStr::from_ptr(ptr).to_string_lossy()
    };
    // Use the string
}
```

Passer une string de Swift et la stocker côté Rust

Un cas peu plus compliquer où on copie la string côté Rust

Swift

```
store_c_string_into_rust(swiftString)
```

Header

```
void store_c_string_into_rust(const char* string);
```

Rust

```
pub extern "C" fn store_c_string_into_rust(ptr: *const c_char) {
    let str = unsafe {
        CStr::from_ptr(ptr).to_string_lossy().into_owned();
    };
    // Store the string
}
```

Retourner une String de Swift

Création copie côté Swift à désallouer ensuite côté Rust

Swift

```
@_cdecl("c_string_from_swift")
func cStringFromSwift() -> UnsafePointer<CChar> {
    UnsafePointer(strdup(swiftString)!)
}
```

Header

```
const char* string c_string_from_swift();
```

Rust

```
let str = unsafe {
    CStr::from_ptr(c_string_from_swift()).to_string_lossy();
};
// Use the string
unsafe {
    libc::free(ptr as *mut c_void);
}
```

Retourner une String de Swift et la stocker côté Rust

Création de copies côté Swift et Rust

Swift

```
@_cdecl("c_string_from_swift")
func cStringFromSwift() -> UnsafePointer<CChar> {
    UnsafePointer(strdup(swiftString)!)
}
```

Header

```
const char* string c_string_from_swift();
```

Rust

```
let str = unsafe {
    let ptr = c_string_from_swift();
    let str = CStr::from_ptr(ptr).to_string_lossy().into_owned();
    libc::free(ptr as *mut c_void);
    str
};
// Store the string
```



Pour plus de performance

- problème → recherche terminateur de fin à chaque copie
- solution → passer la longueur quand on crée la copie avec strdup et into_owned



L'allocateur de Rust est différent de celui de C et Swift

```
unsafe {  
    libc::free(c_ptr as *mut c_void);  
}
```

Retourner une string de Rust à Swift

Logique similaire à ce que l'on a vu précédent

Voir `RustySwift` pour un exemple

- `RustIntegration.swift`
 - `Polygon.description`
 - `polygon.rs`
 - `Polygon.description()`
 - `polygon_description()`
 - `polygon_free_description()`

Struct Rust partagée avec Swift

Implémenter et exposer une struct valeur côté Rust

```
#[repr(C)]
#[derive(Copy, Clone)]
pub struct Point {
    pub x: f64,
    pub y: f64,
}

impl Point {

    fn distance(&self, other: Point) -> f64 {
        (self.x - other.x).hypot(self.y - other.y)
    }
}

#[no_mangle]
pub extern "C" fn distance_to(point: Point, other: Point) -> f64 {
    point.distance(other)
}
```

Struct Rust partagée avec Swift

Utiliser une struct valeur côté Swift

Header

```
typedef struct Point {  
    double x;  
    double y;  
} Point;  
  
double distance_to(struct Point point, struct Point other);
```

Swift

```
extension Point {  
    static let zero: Point = .init(x: 0, y: 0)  
  
    func distance(to other: Point) -> Double {  
        distance_to(self, other)  
    }  
}
```

Démo

Pointeur opaque Rust et wrapper Swift

Implémenter une struct référence côté Rust

Rust

```
#[derive(Clone)]
pub struct Polygon {
    id: Uuid,
    points: Vec<Point>
}

impl Polygon {
    pub fn new() -> Polygon {
        Polygon { id: random_uuid_no_copy(), points: vec![] }
    }

    pub fn length(&self) -> f64 {
        // Code
    }
}
```

Pointeur opaque Rust et wrapper Swift

Exposer une struct référence côté Rust

```
#[no_mangle]
pub extern "C" fn polygon_new() -> *mut Polygon {
    Box::into_raw(Box::new(Polygon::new()))
}
```

```
#[no_mangle]
pub extern "C" fn polygon_free(ptr: *mut Polygon) {
    if ptr.is_null() { return; }
    unsafe { Box::from_raw(ptr); }
}
```

```
#[no_mangle]
pub extern "C" fn polygon_length(ptr: *mut Polygon) -> f64 {
    let polygon = unsafe {
        assert!(!ptr.is_null());
        &mut *ptr
    };
    polygon.length()
}
```

Pointeur opaque Rust et wrapper Swift

Utiliser une struct référence côté Swift

Header

```
typedef struct Polygon Polygon;
struct Polygon *polygon_new(void);
void polygon_free(struct Polygon *ptr);
double polygon_length(struct Polygon *ptr);
```

Swift

```
class Polygon {
    let raw: OpaquePointer = polygon_new()
    var length: Double { polygon_length(raw) }
    deinit {
        polygon_free(raw)
    }
}
```

Échanger des tableaux

Minimiser les copies en mémoire

- 3 approches
 - tableau C
 - optimisation avec functions de mutation (add, remove etc.) dans une extension
 - Swift Array wrapper
 - Rust Vec wrapper
 - possibilité de conformance à RangeReplaceableCollection
- pour échanger d'autres sortes de collections (Set, Dictionary etc.)
 - représentation en C impossible → créer des wrappers Swift ou Rust

Retourner un tableau Rust à Swift

Se méfier des tableaux vides 😞

Voir `RustySwift` pour un exemple

- `RustyIntegration.swift`
 - `Polygon.points`
 - `polygon.rs`
 - `Polygon.points`
 - `polygon_points()`
 - `polygon_set_points()`
 - `free_points()`

Démo

Utiliser Swift depuis Rust

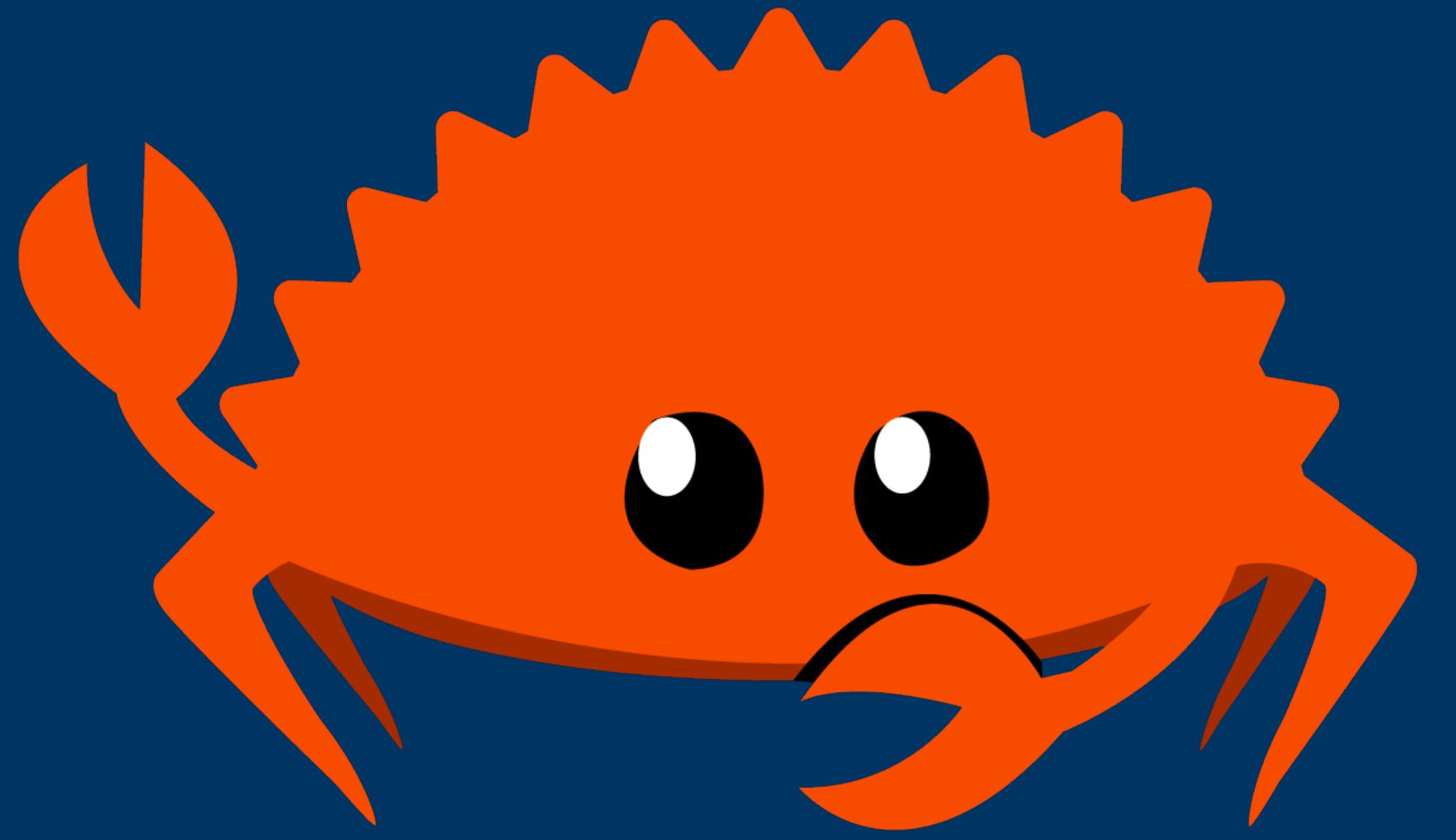
Plus compliqué que l'inverse

- Exposer des fonctions Swift globales marqué avec @_cdecl
- Appeler des fonctions Swift via des pointeurs de fonctions C
 - callbacks
 - classe Swift wrapée côté Rust
- Écrire des applis iOS ou macOS avec Rust
 - How to create a macOS app in Rust
 - Using Rust in Cocoa apps
 - rust-objc → bridge entre Rust et Objective-C
 - core-foundation-rs → ensemble de bindings Rust pour macOS et iOS (CoreFoundation, CoreGraphics, Cocoa etc.)
 - metal-rs → bindings Rust pour Metal sur macOS et iOS

Pour aller plus loin

Exemples, outils et articles

- Démo → github.com/qmathe/RustySwift
- cbindgen → github.com/eqrion/cbindgen
- swift-bridge → github.com/chinedufn/swift-bridge
- [Rust FFI](#)
- [Rust FFI Omnibus](#)
- [Sharing Owned Objects Between Rust and Swift](#)
- [Sharing View Model Between Rust and Swift Threads](#)
- [Dangers of Rust FFI](#)
- [Linking Swift code into Rust app](#)
- [How Swift Achieved Dynamic Linking Where Rust Couldn't](#)
- [Swift Guide to Rust](#)



github.com/qmathe/RustySwift