

소프트웨어 마에스트로 특강

Rust 크로스 플랫폼 프로그래밍

Chris Ohk

utilForever@gmail.com

발표자 소개

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- 옥찬호 (Chris Ohk)
 - (현) Momenti Engine Engineer
 - (전) Nexon Korea Game Programmer
 - Microsoft Developer Technologies MVP
 - C++ Korea Founder & Administrator
 - Reinforcement Learning KR Administrator
 - IT 전문서 집필 및 번역 다수
 - 게임샐러드로 코드 한 줄 없이 게임 만들기 (2013)
 - 유니티 Shader와 Effect 제작 (2014)
 - 2D 게임 프로그래밍 (2014), 러스트 핵심 노트 (2017)
 - 모던 C++ 입문 (2017), C++ 최적화 (2019)

utilForever@gmail.com



utilForever



목차

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- 크로스 플랫폼 프로그래밍을 하게 된 이유
- Rust로 크로스 플랫폼 프로그래밍 해보기
- 몇 가지 팁들

이직 후 첫번째 업무

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- 여러 플랫폼에서 사용할 수 있는 코어 엔진을 만들어주세요.
 - iOS
 - Android
 - Backend
 - Web
 - ...

기존에는 어떻게 개발되고 있었는가

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- 똑같은 기능을 하는 엔진 코드가 여러 플랫폼에 각각 구현되어 있었다.
 - iOS 앱 → C + Objective-C 기반
 - 프론트엔드 → TypeScript 기반
- 새로운 기능을 구현하거나 기존 기능을 수정해달라는 요청이 들어오면,
 - iOS 앱 → C 코드를 구현한 뒤, Objective-C 코드를 통해 앱에 적용한다.
 - 프론트엔드 → TypeScript 코드를 구현해 웹에 적용한다.

- 어떤 기능을 추가/삭제하거나 변경해야 할 때 지원하는 플랫폼마다 작업을 해줘야 한다.
 - 유지보수 측면에서 비효율적이다.
 - 지원하는 플랫폼이 늘어날수록 작업량이 늘어난다.
- 똑같은 기능을 수행했을 때 플랫폼마다 동작 결과가 달라질 수 있다.
 - 플랫폼마다 서로 다른 사람이 구현하기 때문에 발생하는 문제다.
 - 서로 다른 동작으로 인해 사용자에게 불편을 초래할 수 있다.

어떻게 개선할 것인가

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- 플랫폼마다 로직을 구현하지 않고, 한 곳에서 작업하면 좋겠다.
- 다양한 플랫폼에 어떻게 대응해야 할까?
 - iOS 앱 → Swift API 제공
 - Android 앱 → Kotlin API 제공
 - 백엔드 → Elixir API 제공
 - 웹 사이트 → 웹어셈블리 바이너리 파일 제공

왜 Rust를 선택했는가

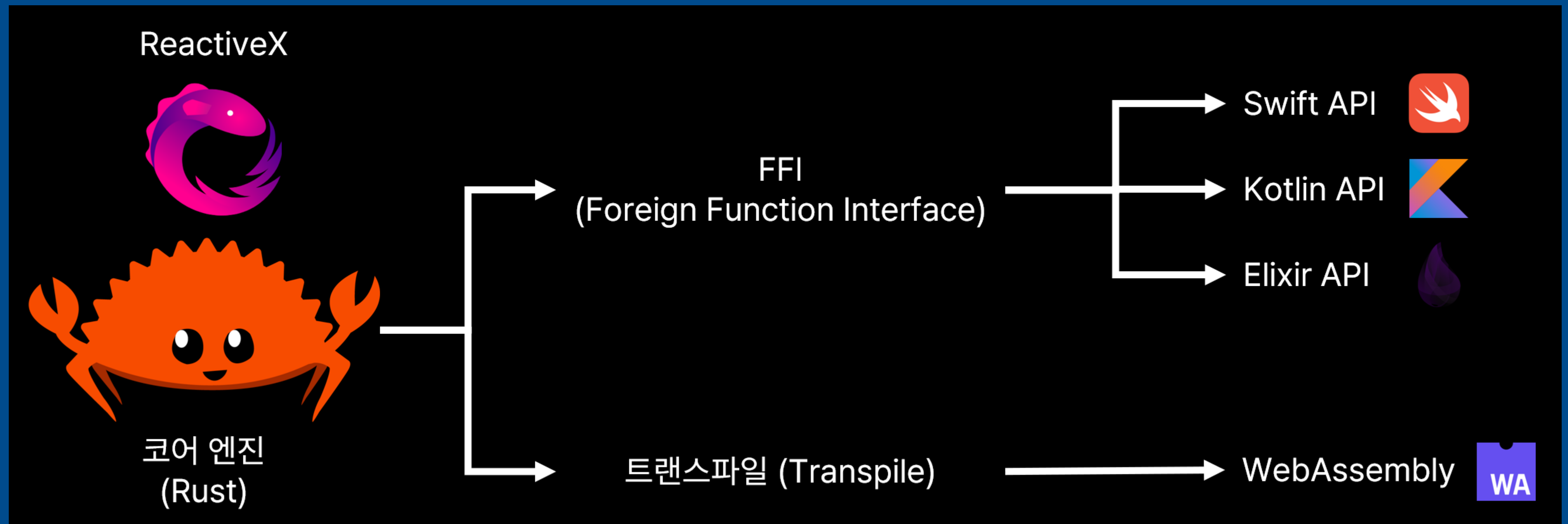
소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- 다양한 언어를 지원하기 위한 API와 웹 어셈블리 코드를 만들 수 있는 언어
 - C++, Rust, Java, Go, ...
- Rust를 선택한 이유
 - 타입 안전성
 - 메모리 안전성
 - 동시성 프로그래밍 안전성
 - FFI를 통해 API 지원을 쉽게 할 수 있음
 - WebAssembly 바이너리를 쉽게 만들 수 있음

- 같은 동작을 하는 프로그램을 여러 플랫폼에서 사용할 수 있도록 만들고 있다.
 - iOS 앱 (Swift)
 - Android 앱 (Kotlin)
 - 프론트엔드 (TypeScript)
 - 백엔드 (Elixir)
- 각 플랫폼에서 사용자 입력에 따라 로직을 처리하고 결과물을 보여주기 위한 코드를 개별적으로 작성한다.

프로젝트 구조

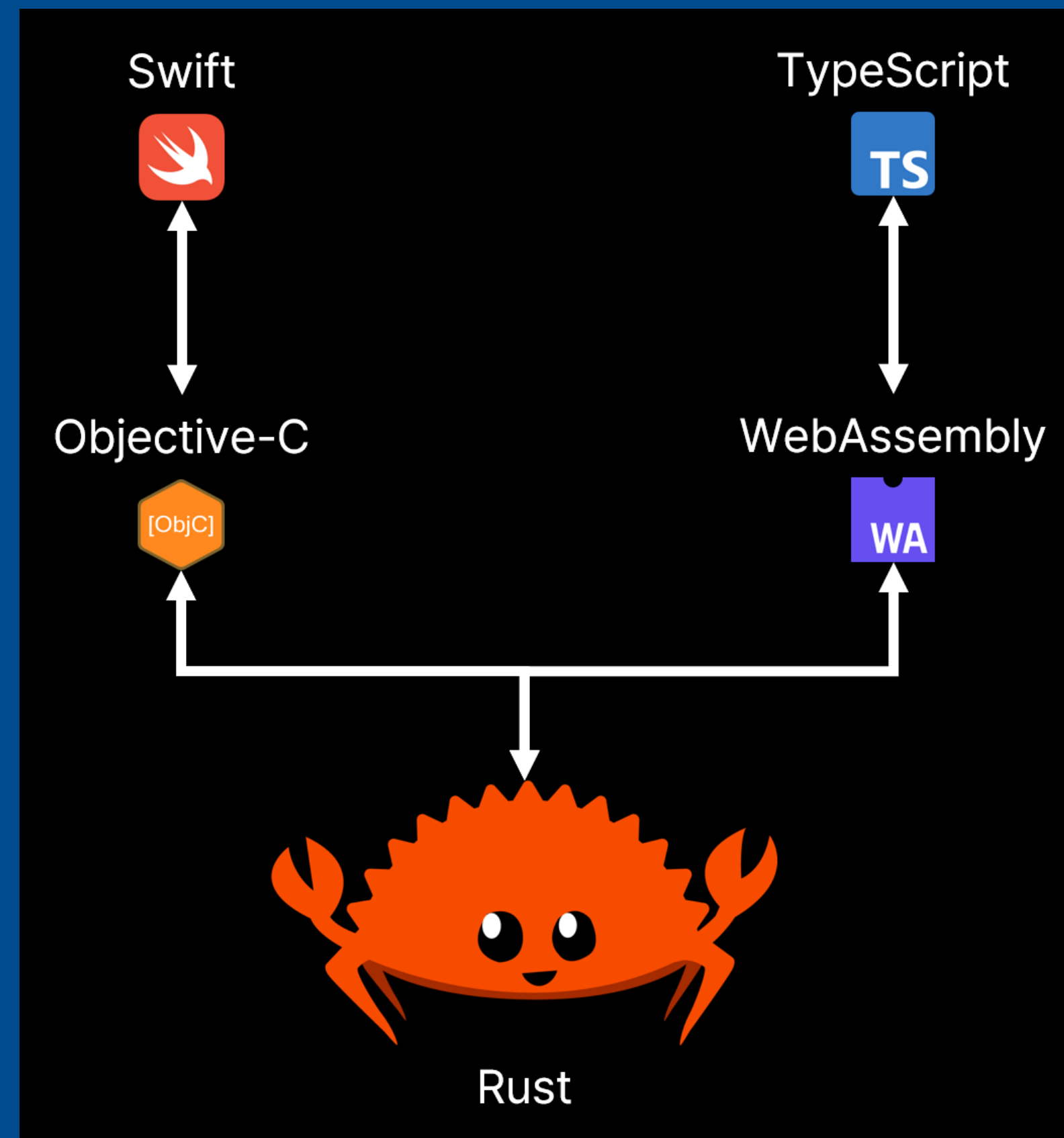
소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍



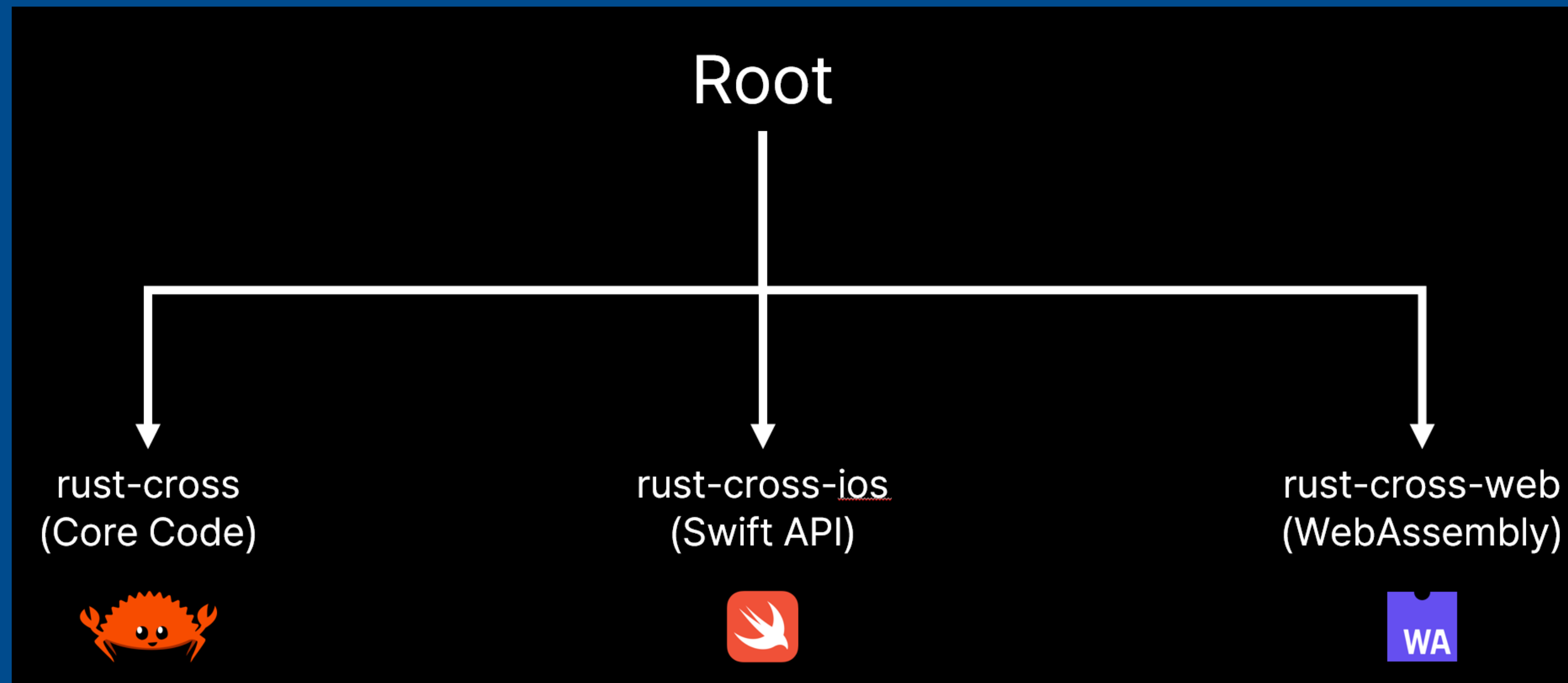
무엇을 만들 것인가

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- 다양한 플랫폼에서 Rust 라이브러리에 있는 add 함수와 sub 함수를 호출해 결과를 받은 뒤 출력하는 예제를 만들어 보자.



- rust-cross : 핵심 코드를 구현하는 크레이트. add 함수와 sub 함수를 구현한다.
- rust-cross-ios : Swift API를 만들기 위한 코드를 구현하는 크레이트.
- rust-cross-web : WebAssembly로 트랜스파일하기 위한 코드를 구현하는 크레이트.



프로젝트 구조 잡기

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍


- 빈 디렉토리에 Cargo.toml 파일을 만든 뒤 다음과 같이 입력한다.

```
[workspace]
members = [
    "rust-cross",
    "rust-cross-ios",
    "rust-cross-web",
]
default-members = ["rust-cross"]
```

새 Rust 라이브러리 프로젝트 만들기

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- cargo new 명령을 통해 만들 수 있다.
 - 라이브러리 프로젝트를 만들려면 --lib를 붙여야 한다.
 - 그렇지 않으면, 바이너리 파일을 생성하는 프로젝트가 만들어진다.



```
cargo new rust-cross --lib
cargo new rust-cross-ios --lib
cargo new rust-cross-web --lib
```

add/sub 함수 구현하기

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- 외부에서 사용할 수 있도록 pub 키워드를 붙이자.
- 구현을 완료했다면 cargo build --release 명령을 통해 빌드가 되는지 확인한다.

```
pub fn add(a: i64, b: i64) -> i64 {  
    a + b  
}  
  
pub fn sub(a: i64, b: i64) -> i64 {  
    a - b  
}
```

iOS를 위한 라이브러리 빌드 준비

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- Xcode를 설치한다.
 - App Store에서 설치할 수 있다.
- Xcode 빌드 툴을 설치한다.
 - `xcode-select --install`
- 그리고 크로스 컴파일이 가능하도록 iOS 아키텍처를 추가한다.
 - `rustup target add aarch64-apple-ios aarch64-apple-ios-sim x86_64-apple-ios`
- 크로스 컴파일을 쉽게 할 수 있도록 도와주는 툴인 `cargo-lipo`를 설치한다.
 - `cargo install cargo-lipo`

- Cargo.toml 파일을 연 뒤, 다음과 같이 수정한다.
 - 정적/동적 라이브러리를 만들기 위해 [lib]의 crate-type을 ["staticlib", "cdylib"]로 설정한다.
 - rust-cross 크레이트에 있는 함수들을 사용하므로 [dependencies]에 추가한다.

```
[package]
name = "rust-cross-ios"
version = "0.1.0"
edition = "2021"

[lib]
crate-type = ["staticlib", "cdylib"]

[dependencies]
rust-cross = { path = "../rust-cross" }
```

C 브릿지에서 호출할 코드 작성

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- 함수의 이름이 망글링되지 않도록 `#[no_mangle]`을 추가한다.
- 망글링(Mangling)이란 소스 코드에 선언된 함수나 변수의 이름을 컴파일 단계에서 일정한 규칙을 갖고 변형하는 걸 말한다.

```
#[no_mangle]
pub extern "C" fn add(a: i64, b: i64) -> i64 {
    rust_cross::add(a, b)
}

#[no_mangle]
pub extern "C" fn sub(a: i64, b: i64) -> i64 {
    rust_cross::sub(a, b)
}
```

C 브릿지 헤더 파일 작성

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- Rust에서 사용한 매개 변수 타입과 리턴 타입에 호환되는 타입을 사용해야 한다.
 - 예 : i32 또는 i64 → int
- cargo lipo -release 명령을 통해 라이브러리 파일 (.a)가 생성되는지 확인한다.

```
#include <stdint.h>

int add(int a, int b);
int sub(int a, int b);
```

WebAssembly 빌드 준비

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- Rust로 WebAssembly 바이너리 파일을 빌드해주는 툴인 wasm-pack을 설치한다.
 - `curl https://rustwasm.github.io/wasm-pack/installer/init.sh -sSf | sh`
- JavaScript 번들러를 설치하고 실행할 수 있도록 패키지 매니저인 npm을 설치한다.
 - `npm install npm@latest -g`

- Cargo.toml 파일을 연 뒤, 다음과 같이 수정한다.
 - 동적 시스템/러스트 라이브러리를 만들기 위해 [lib]의 crate-type을 ["cdylib", "rlib"]로 설정한다.
 - rust-cross 크레이트에 있는 함수들을 사용하므로 [dependencies]에 추가한다.
 - WebAssembly 빌드를 위해 wasm-bindgen을 [dependencies]에 추가한다.

```
[package]
name = "rust-cross-web"
version = "0.1.0"
edition = "2021"

[lib]
crate-type = ["cdylib", "rlib"]

[dependencies]
rust-cross = { path = "../rust-cross" }
wasm-bindgen = "0.2.48"
```

WebAssembly로 빌드할 코드 작성

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- 필요할 경우 Wrapper로 처리할 수 있도록 `#[wasm_bindgen]`을 추가한다.
- `wasm-pack build` 명령을 통해 wasm 파일이 생성되는지 확인한다.

```
use wasm_bindgen::prelude::*;

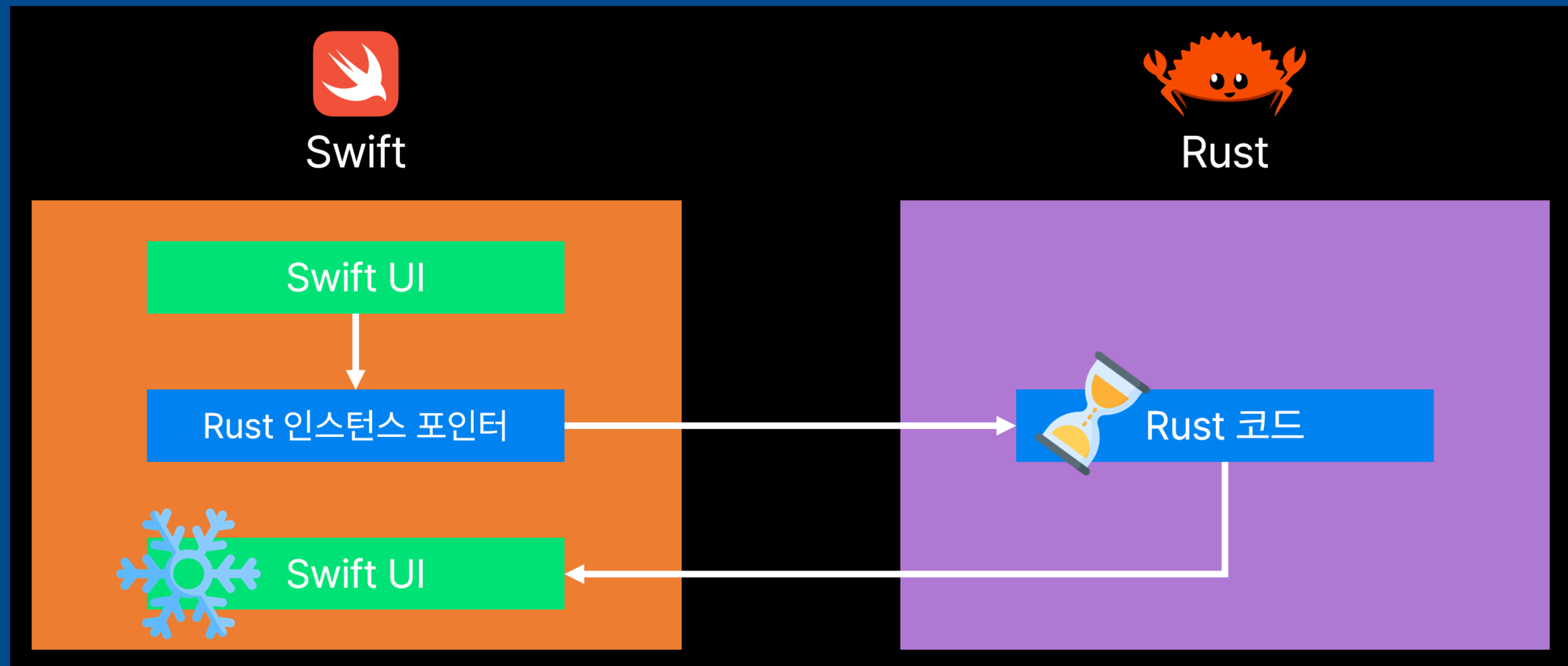
#[wasm_bindgen]
pub fn add(a: i64, b: i64) -> i64 {
    rust_cross::add(a, b)
}

#[wasm_bindgen]
pub fn sub(a: i64, b: i64) -> i64 {
    rust_cross::sub(a, b)
}
```

애플리케이션 프리징 문제

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

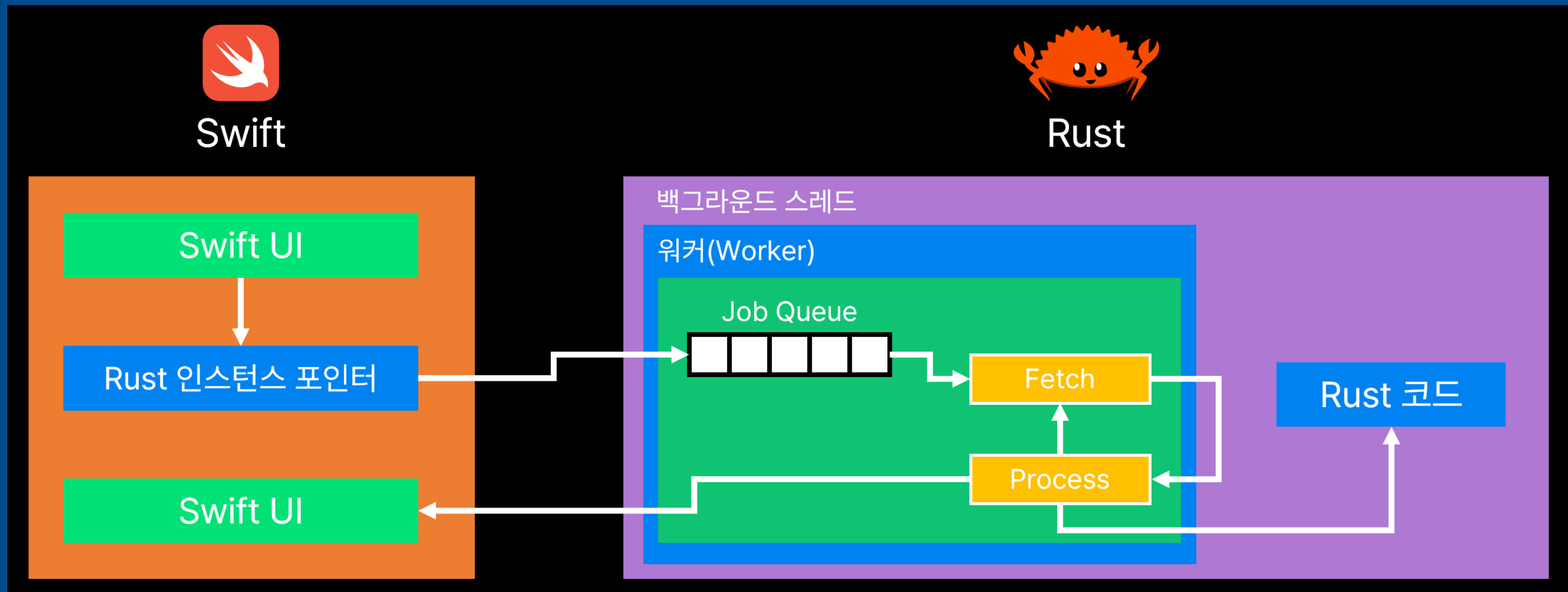
- Swift 앱에서 어떤 동작을 수행하기 위해 UI에서 버튼을 누르면, 엔진에서 처리되는 동안에 앱이 멈추는 문제가 발생한다.
- 원인은 Swift에서 필요한 함수들을 직접 호출하며, 모든 함수 호출은 동기적으로 동작하기 때문이다.



애플리케이션 프리징 문제

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- 이를 해결하려면, 엔진에 대한 모든 요청을 처리하는 워커를 구현해 백그라운드 스레드에서 동작하게 만들어야 한다.
- 백그라운드 스레드에서 워커(Worker)를 통해 요청들을 Job Queue에 넣으면 작업을 비동기적으로 수행한다.



WebAssembly 함수 맵글링 문제

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- Rust + WebAssembly로 빌드하면 함수 이름들이 맵글링된다.
- 맵글링된 함수 이름 때문에 콜 스택을 제대로 확인할 수 없는 문제가 있다.

```
panicked at 3:22 consoleObservable.ts:43

Stack:
Error
  at 
  at 
  at 
  at http://192.168.1.26:8085/c9d8136...module.wasm:wasm-function[7827]:0x15f69e
  at http://192.168.1.26:8085/c9d8136...module.wasm:wasm-function[1448]:0xc0b5c
  at http://192.168.1.26:8085/c9d8136...module.wasm:wasm-function[8213]:0x16308a
  at http://192.168.1.26:8085/c9d8136...module.wasm:wasm-function[7479]:0x15b626
  at http://192.168.1.26:8085/c9d8136...module.wasm:wasm-function[2927]:0x101871
  at http://192.168.1.26:8085/c9d8136...module.wasm:wasm-function[4591]:0x12fd0a
  at http://192.168.1.26:8085/c9d8136...module.wasm:wasm-function[8050]:0x161951

console.<computed> @ consoleObservable
(anonymous) @ 
logError @ 
__wbg_error_09919627ac0992f5 @ 
$func3992 @ c9d8136...odule.
$func1448 @ c9d8136...odule.
$func8213 @ c9d8136...odule.
$func7479 @ c9d8136...odule.
$func2927 @ c9d8136...odule.
$func4591 @ c9d8136...odule.
$func8050 @ c9d8136...odule.
$func8196 @ c9d8136...odule.
$func7466 @ c9d8136...odule.
$func8594 @ c9d8136...odule.
$func8672 @ c9d8136...odule.
```

WebAssembly 함수 망글링 문제

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- 이 문제를 해결하려면 WebAssembly 코드를 배포하는 크레이트의 Cargo.toml에 각 빌드에 따른 설정을 해줘야 한다.
- 디버깅을 위해, dev 빌드에서 demangle-name-section을 true로 설정하고 wasm-opt에 플래그 -g를 추가한다.
- 이제 wasm 파일이 빌드되면 웹쪽 코드에 pkg 폴더를 붙여넣기하고 실행하면 된다.

```
[package.metadata.wasm-pack.profile.dev]
wasm-opt = ['-O', '-g']

[package.metadata.wasm-pack.profile.dev.wasm-bindgen]
# Should we enable wasm-bindgen's debug assertions in its generated JS glue?
debug-js-glue = true
# Should wasm-bindgen demangle the symbols in the "name" custom section?
demangle-name-section = true
# Should we emit the DWARF debug info custom sections?
dwarf-debug-info = false

[package.metadata.wasm-pack.profile.profiling]
wasm-opt = ['-O']

[package.metadata.wasm-pack.profile.profiling.wasm-bindgen]
debug-js-glue = false
demangle-name-section = false
dwarf-debug-info = false

[package.metadata.wasm-pack.profile.release]
wasm-opt = ['-O']

[package.metadata.wasm-pack.profile.release.wasm-bindgen]
debug-js-glue = false
demangle-name-section = false
dwarf-debug-info = false
```

WebAssembly 함수 맵링 문제

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- npm을 실행했을 때 Failed to decode custom "name" section @2024365; ignoring (Maximum call stack size exceeded) 오류가 발생하는 경우가 있다.
이 때는 node를 실행할 때 --stack-size=10000을 추가해서 스택 크기를 늘려야 한다.
- 하지만 보안 이슈로 인해 NODE_OPTIONS를 사용할 수는 없고,
package.json에서 npm start에 해당하는 부분에 추가하면 된다.
- 심볼 정보를 전부 포함한 채로 실행하기 때문에 불러오는데 꽤 오랜 시간이 걸릴 수 있다. (인내하고 기다리자!)

WebAssembly 함수 맵글링 문제

소프트웨어 마에스트로 특강
Rust 크로스 플랫폼 프로그래밍

- 이제 잘 실행되고 디맵글링된 함수 이름들을 볼 수 있다.

```
✖ panicked at [REDACTED] consoleObservable.ts:43
3:22

Stack:
Error
  at [REDACTED]
  at [REDACTED]
  at [REDACTED]
  at console_error_panic_hook::Error::new::habaaa7a517e69b73 (http://192.168.1.26:8085/380a79c...module.wasm:wasm-function[7827]:0x15f78
7)
  at console_error_panic_hook::hook_impl::he97eccf8e518f1c3 (http://192.168.1.26:8085/380a79c...module.wasm:wasm-function[1448]:0xc0c07)
  at console_error_panic_hook::hook::h1ef0976bed306fb0 (http://192.168.1.26:8085/380a79c...module.wasm:wasm-function[8213]:0x163173)
  at core::ops::function::Fn::call::hfd142f2b7bbd24f1 (http://192.168.1.26:8085/380a79c...module.wasm:wasm-function[7479]:0x15b70f)
  at std::panicking::rust_panic_with_hook::h56d066e3564322fb (http://192.168.1.26:8085/380a79c...module.wasm:wasm-function[2927]:0x10192
a)
  at std::panicking::begin_panic_handler::{{closure}}::he926425bf39194d7 (http://192.168.1.26:8085/380a79c...module.wasm:wasm-function[4
591]:0x12fdf1)
  at rust_begin_unwind (http://192.168.1.26:8085/380a79c...module.wasm:wasm-function[8050]:0x161a3a)

console.<computed> @ consoleObservable.ts:43
(anonymous) @ [REDACTED]
logError @ [REDACTED]
__wbg_error_09919627ac0992f5 @ [REDACTED]
$console_error_panic_hook::error::h881975d7e773fe08 @ 380a79c...module.wasm:wasm-function[7827]:0x15f78
$console_error_panic_hook::hook_impl::he97eccf8e518f1c3 @ 380a79c...module.wasm:wasm-function[1448]:0xc0c07
$console_error_panic_hook::hook::h1ef0976bed306fb0 @ 380a79c...module.wasm:wasm-function[8213]:0x163173
$core::ops::function::Fn::call::hfd142f2b7bbd24f1 @ 380a79c...module.wasm:wasm-function[7479]:0x15b70f
$std::panicking::rust_panic_with_hook::h56d066e3564322fb @ 380a79c...module.wasm:wasm-function[2927]:0x10192
$std::panicking::begin_panic_handler::{{closure}}::he926425bf39194d7 @ 380a79c...module.wasm:wasm-function[4591]:0x12fdf1
$rust_begin_unwind @ 380a79c...module.wasm:wasm-function[8050]:0x161a3a
$core::panicking::panic_fmt::h0a20ba97f16cb738 @ 380a79c...module.wasm:wasm-function[7479]:0x15b70f
$core::panicking::panic_display::h02eec950c6198561 @ 380a79c...module.wasm:wasm-function[8213]:0x163173
$core::panicking::panic_str::haedd4881c3b18d7d @ 380a79c...module.wasm:wasm-function[7479]:0x15b70f
$core::option::expect_failed::h7b17b0c26e58b8b9 @ 380a79c...module.wasm:wasm-function[7479]:0x15b70f
$core::option::Option<T>::expect::h63b3ac7991ec1912 @ 380a79c...module.wasm:wasm-function[7479]:0x15b70f
[REDACTED] @ 380a79c...module.wasm:wasm-function[7479]:0x15b70f
$<rxrust::observable::observable_next::ObserverN<N,Item> as rxrust::observer::Observer>::next::h64e99edc0e0cd0af @ 380a79c...module.wasm:wasm-function[7479]:0x15b70f
```

Thank you!