Rust 异步 FFI 系统
以 Rust2Go 为例

2024.09.08 Shanghai

第 四 届 中 国 R u s t 开 发 者 大 会

关于我

茌海(Chi Hai)
GitHub/WeChat: ihciah

- 毕业于复旦大学
- Rust 语言 & 开源爱好者

字节跳动高级研发工程师；高性能网关方向负责人
- Rust Async Runtime（Monoio）
- Rust 通用网关框架（MonoLake）
- 从零手搓了公司内大量 Rust 基础库 / 基础设施

CONTENTS ▶

FFI 与跨语言通信

PART 01

关键问题与设计

PART 02

框架实现

PART 03

性能优化

PART 04

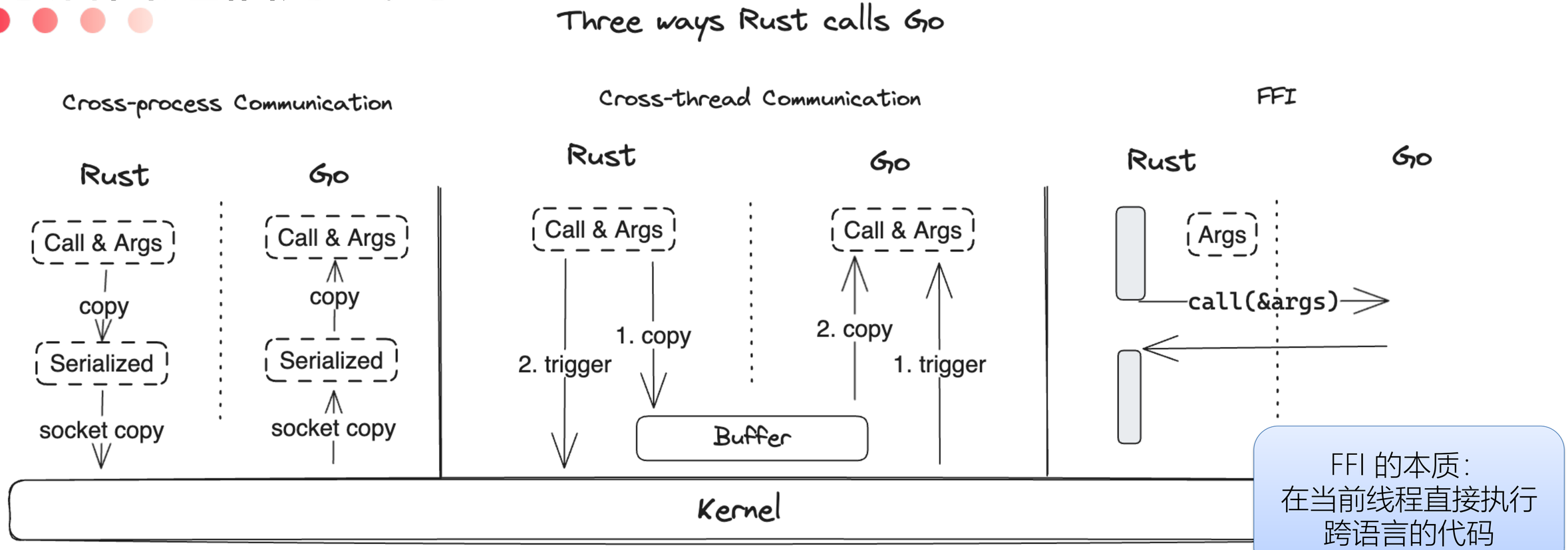
https://github.com/ihciah/rust2go

01

FFI 与跨语言通信

跨语言通信方案对比；FFI 基础

跨语言通信方式对比



FFI 的本质：
在当前线程直接执行
跨语言的代码

跨进程通信

Cost：序列化+反序列化+io syscall

典型例子：RPC

跨线程通信

Cost：内存拷贝+trigger syscall

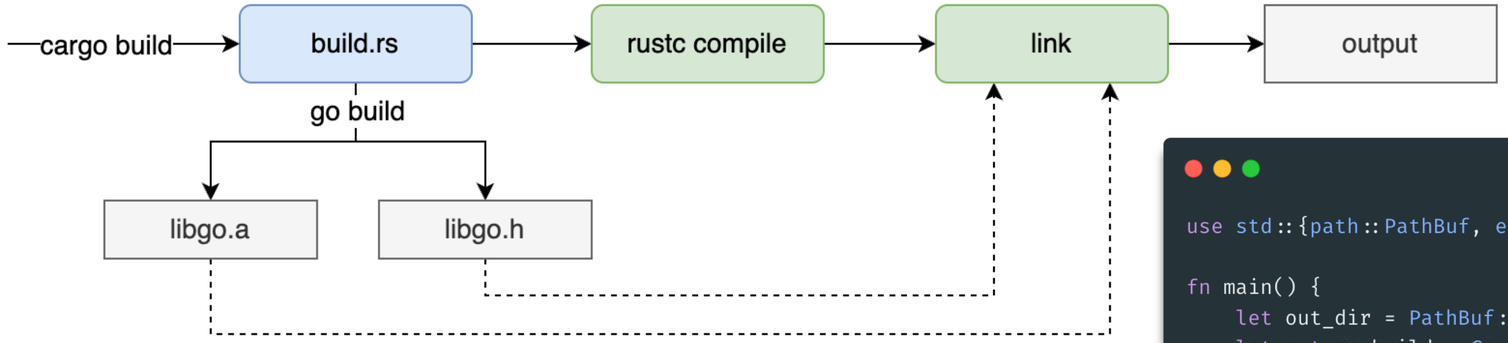
典型例子：ShmIPC

FFI：

Cost：内存对齐

典型例子：Rust2Go

FFI Hello World




```go
package main

import "C"

//export CDemoCall
func CDemoCall() {
    // ...
}

func main() {}
```

main.go

```rust
extern "C" {
    pub fn CDemoCall();
}
fn main() {
    unsafe {
        CDemoCall();
    }
}
```

main.rs

```rust
use std::{path::PathBuf, env, process::Command};

fn main() {
    let out_dir = PathBuf::from(env::var("OUT_DIR").unwrap());
    let mut go_build = Command::new("go");
    go_build
        .arg("build")
        .arg("-buildmode=c-archive")
        .arg("-o")
        .arg(out_dir.join("libgo.a"))
        .arg("path-to-go-file");

    go_build.status().expect("Go build failed");

    println!("cargo:rerun-if-changed={}", "path-to-go-file");
    println!(
        "cargo:rustc-link-search=native={}",
        out_dir.to_str().unwrap()
    );
    println!("cargo:rustc-link-lib=static=go");
}
```

build.rs

02

关键问题与设计

参数和返回值传递；异步支持；复杂类型支持；内存安全

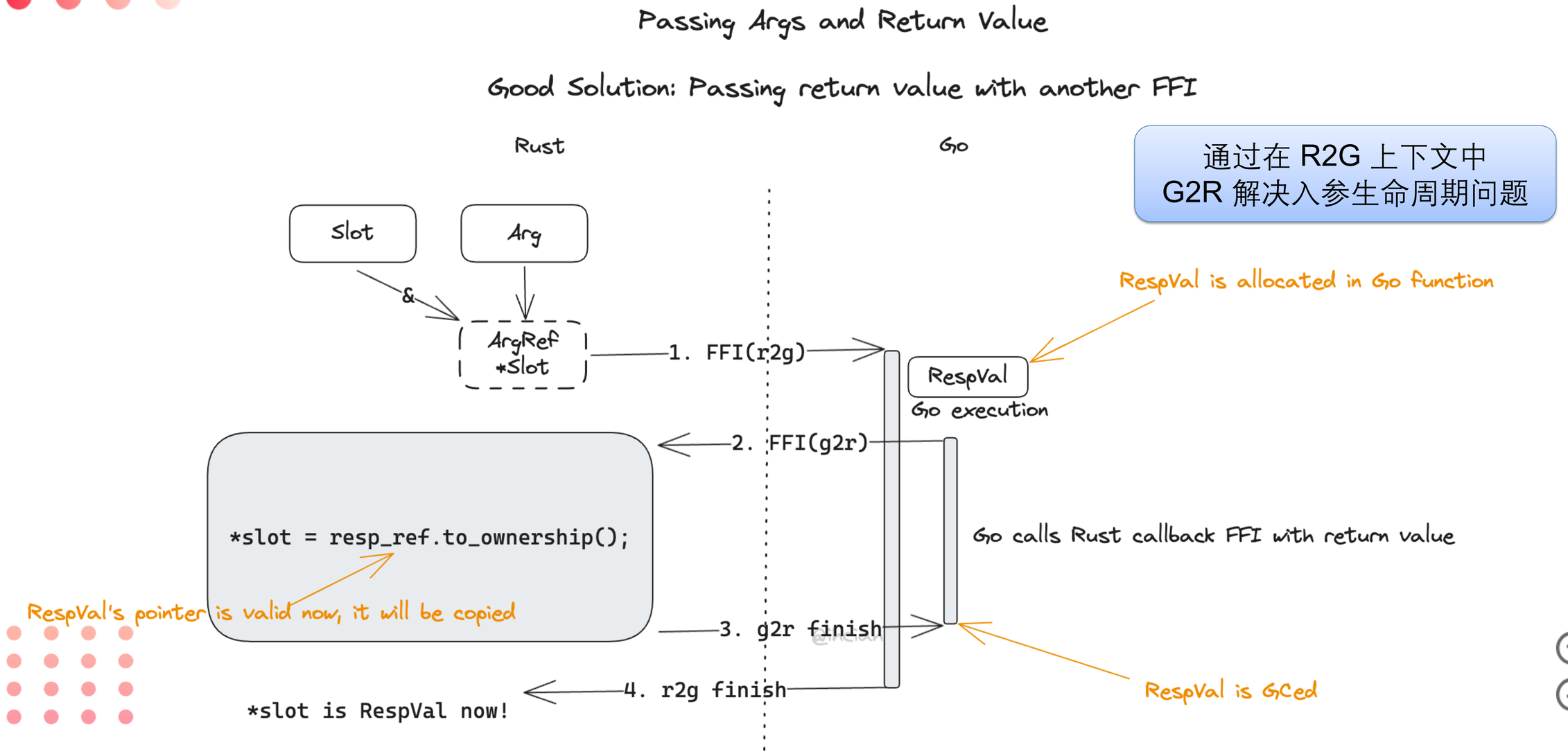
1. 参数和返回值传递 – 入参表示

Rust China Conf

传递基础类型（u8, u16, …）没问题，但如何传递 String 或其他嵌套更深的堆数据呢？

序列化 或 传递引用

1. 参数和返回值传递 – 生命周期管理



通过在 R2G 上下文中
G2R 解决入参生命周期问题

1. 参数和返回值传递 – Go call Rust

Rust China Conf

无法扩栈!

Go -> ASM -> Rust(C ABI)

Go -> CGO -> Rust(C ABI)

ABI 转换

产物管理和编译流程复杂!

Rust as cdylib

```rust
#[no_mangle]
unsafe extern "C" fn xxx_cb( ... ) {
    ...
}
```

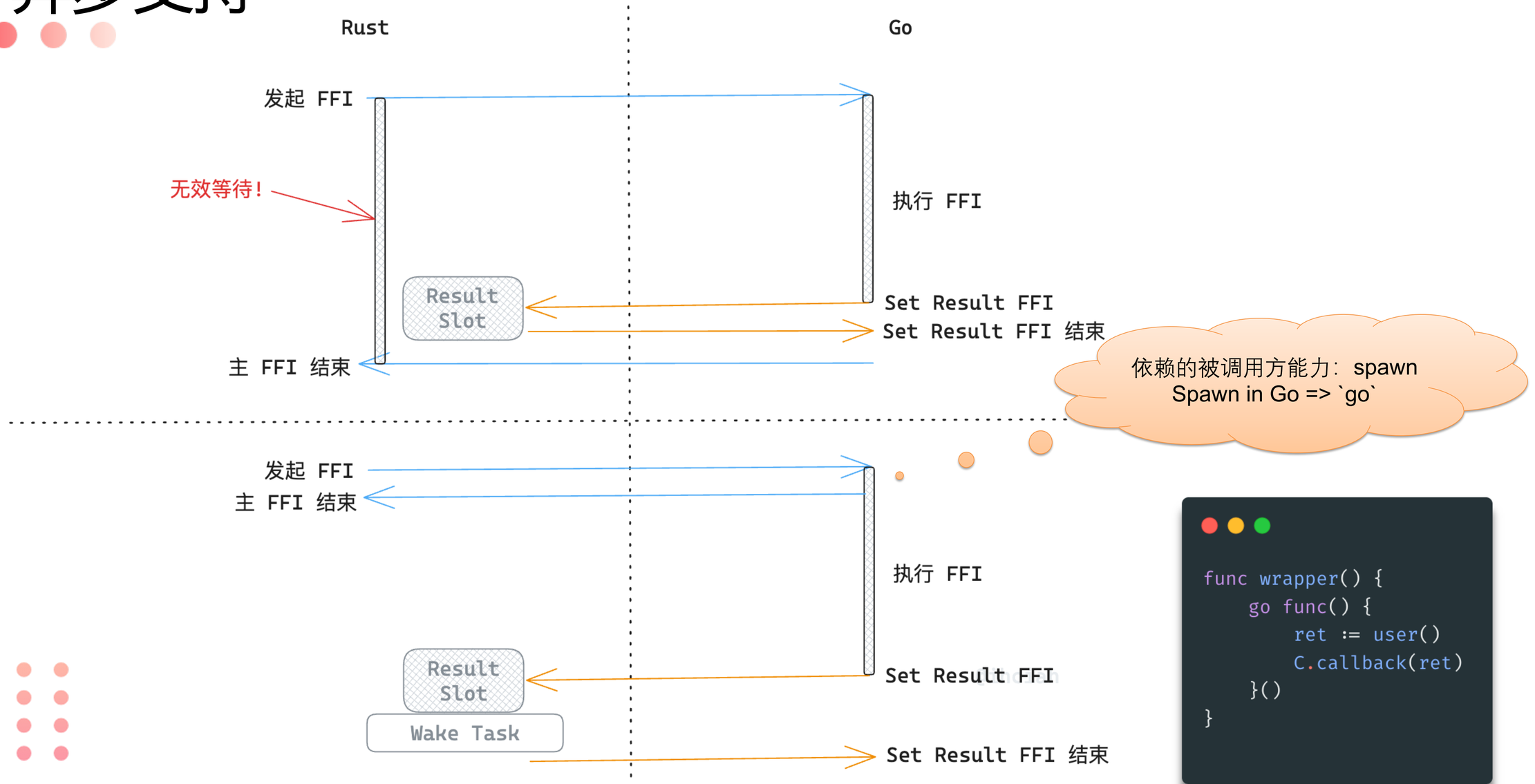
Pass Pointer & Static Cast

```go
/*
__attribute__((weak))
inline void xxx_cb(const void *f_ptr, ... ) {
    ((void (*)( ... ))f_ptr)( ... );
}
*/
import "C"

func go_xxx_cb(cb *C.void) {
    C.xxx_cb(unsafe.Pointer(cb),  ... )
}
```

函数 Locate

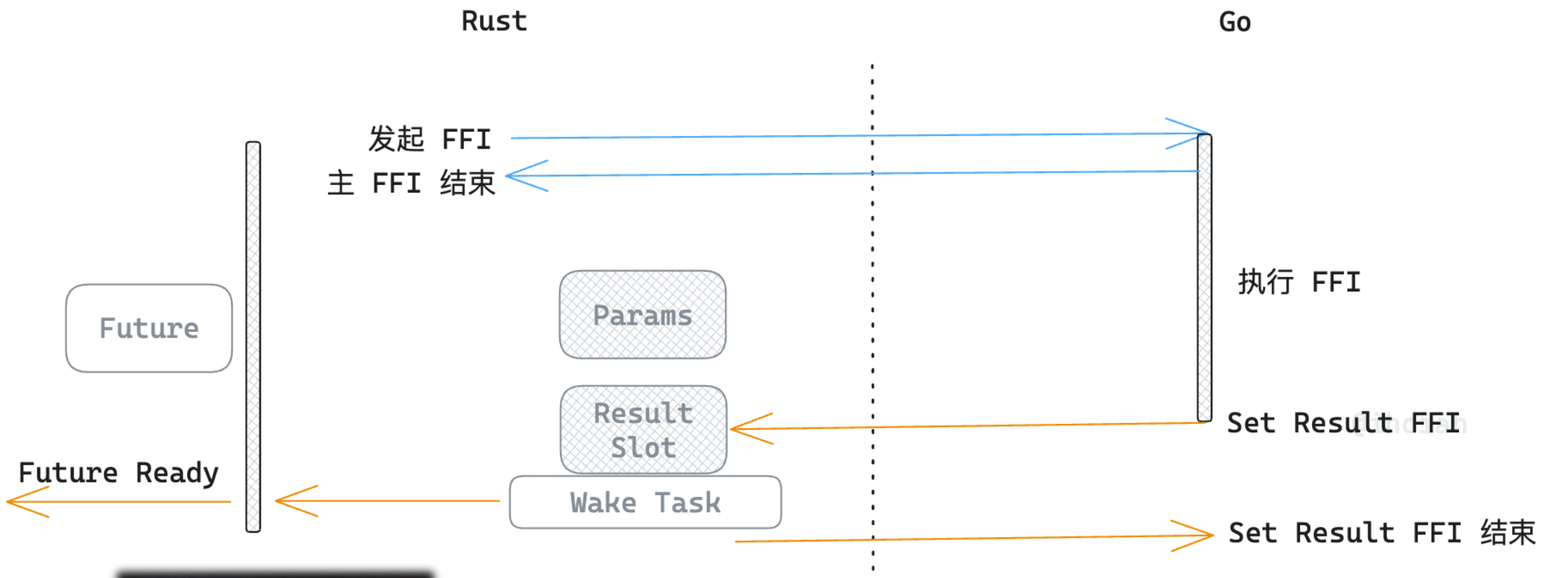
2. 异步支持



依赖的被调用方能力：spawn
Spawn in Go => `go`

```go
func wrapper() {
    go func() {
        ret := user()
        C.callback(ret)
    }()
}
```

2. 异步支持 – 并发安全



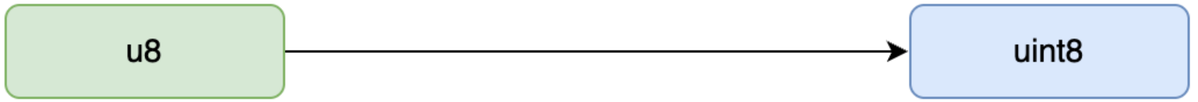
1. Future Early Drop?
2. Future woken up by mistake?

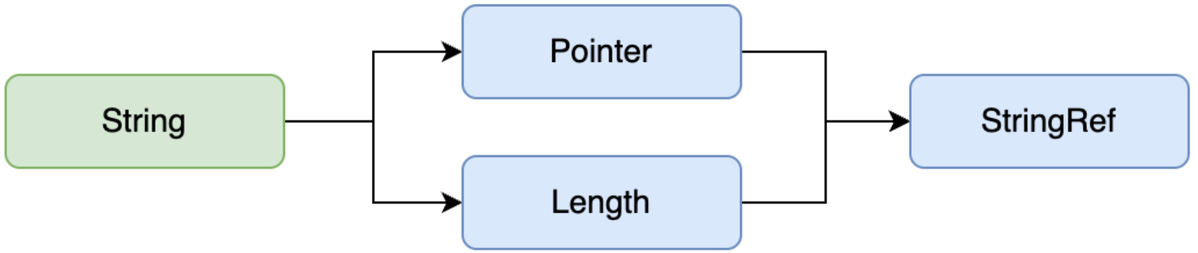
Atomic Slot!

Result Slot: Rust 读 && Go 写

Params & Slot lifetime: 双边都可能先退出

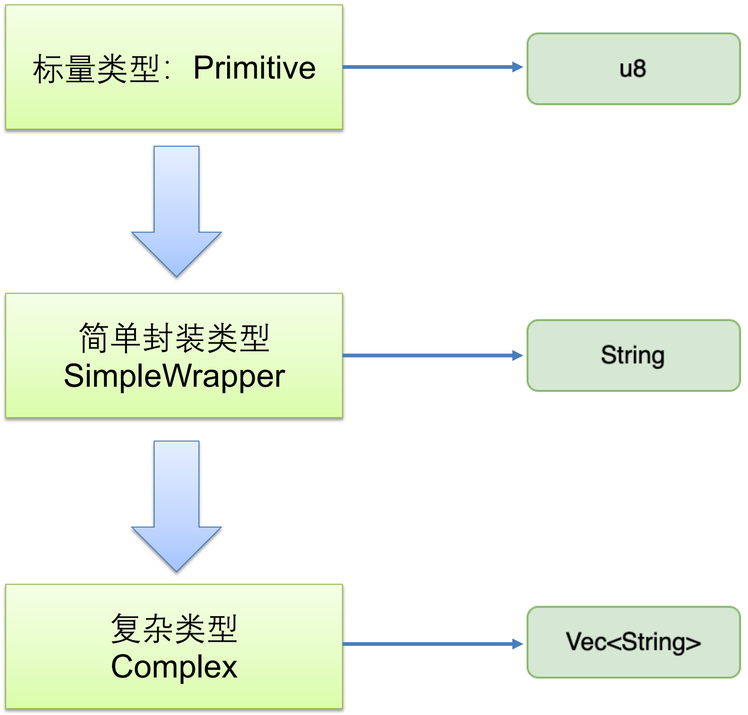
3. 复杂类型支持





Vec<String>

3. 复杂类型支持




```rust
pub enum MemType {
    Primitive,
    SimpleWrapper,
    Complex,
}

pub trait ToRef {
    const MEM_TYPE: MemType;

    type Ref;
    fn to_size(&self, acc: &mut usize);
    fn to_ref(&self, buffer: &mut Writer) -> Self::Ref;
}
```

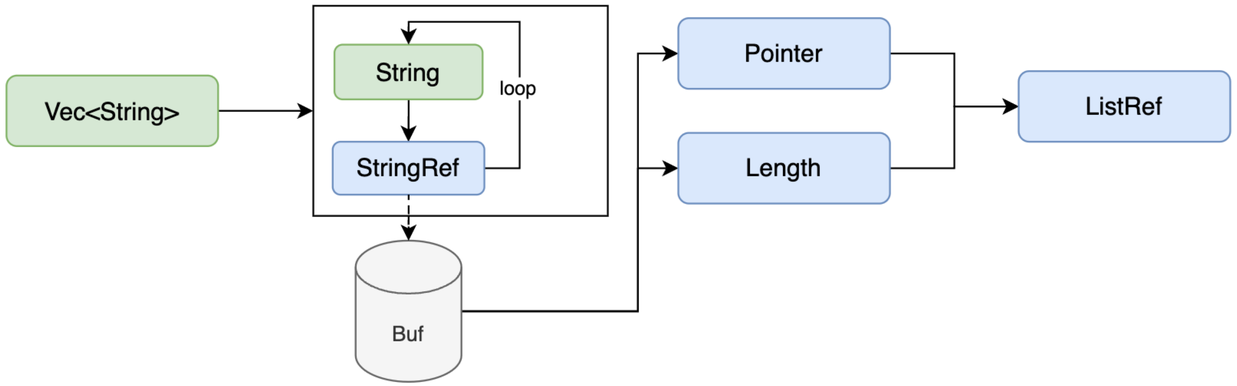
Rule1: 由多种内存类型组成的 struct 对应内存类型为所有 fields 中最大的类型

例：struct{u16, String, Vec<String>} 内存类型为 Complex

Rule2: Container<T> 的内存类型是 T 内存类型的下一个更大的类型

例：Vec<{u16, u32}> 内存类型为 SimpleWrapper; Vec<{u8, String}>内存类型为 Complex

3. 复杂类型支持



原始内存

标量类型：Primitive

传递新创建的、指向原始内存的结构

简单封装类型
SimpleWrapper

传递新创建的，指向新创建内存的结构

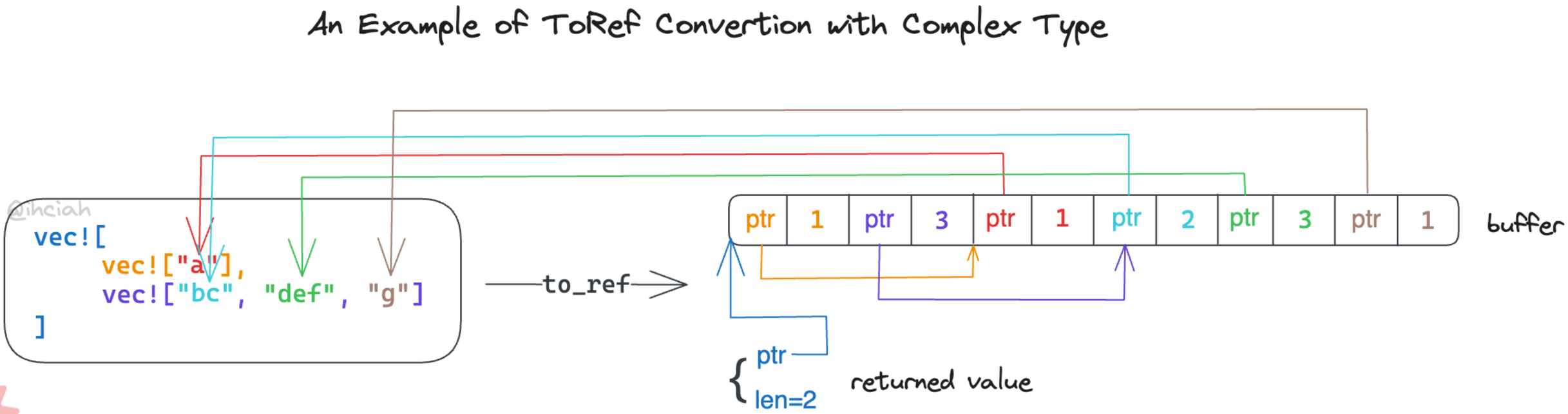
复杂类型
Complex

```rust
pub enum MemType {
    Primitive,
    SimpleWrapper,
    Complex,
}

pub trait ToRef {
    const MEM_TYPE: MemType;

    type Ref;
    fn to_size(&self, acc: &mut usize);
    fn to_ref(&self, buffer: &mut Writer) -> Self::Ref;
}
```

3. 复杂类型支持


An Example of ToRef Convertion with Complex Type
@ihciah
vec![
vec!["a"],
vec!["bc", "def", "g"]
]
to_ref
ptr 1 ptr 3 ptr 1 ptr 2 ptr 3 ptr 1
buffer
ptr
len=2
returned value

4. 内存安全

Rust China Conf

```rust
#[rust2go::r2g]
pub trait DemoCall {
    fn demo_oneway(req: &DemoUser);
    fn demo_check(req: &DemoComplicatedRequest) -> DemoResponse;
    fn demo_check_async(
        req: &DemoComplicatedRequest,
    ) -> impl std::future::Future<Output = DemoResponse>;
    #[drop_safe]
    fn demo_check_async_safe(
        req: DemoComplicatedRequest,
    ) -> impl std::future::Future<Output = DemoResponse>;
    #[drop_safe_ret]
    fn demo_check_async_safe2(
        req: DemoComplicatedRequest,
    ) -> impl std::future::Future<Output = DemoResponse>;
}
```

同步调用：安全！
由同步调用本身保证

异步传引用，不安全！
Drop Future 后可 Drop Params，对端内存访问异常

异步传所有权，安全！

将 Params 置于堆上，并由双边通过引用计数管理

Q：若 Golang 侧没有遵循约定，outlive 了借用的变量怎么办？
A：不提供此类安全保证，这是 Golang 侧（用户代码）需要保证的，Rust 侧不需要提供保证。

03

框架实现

调用约定描述；代码生成

Rust 友好的调用约定描述

Rust China Conf

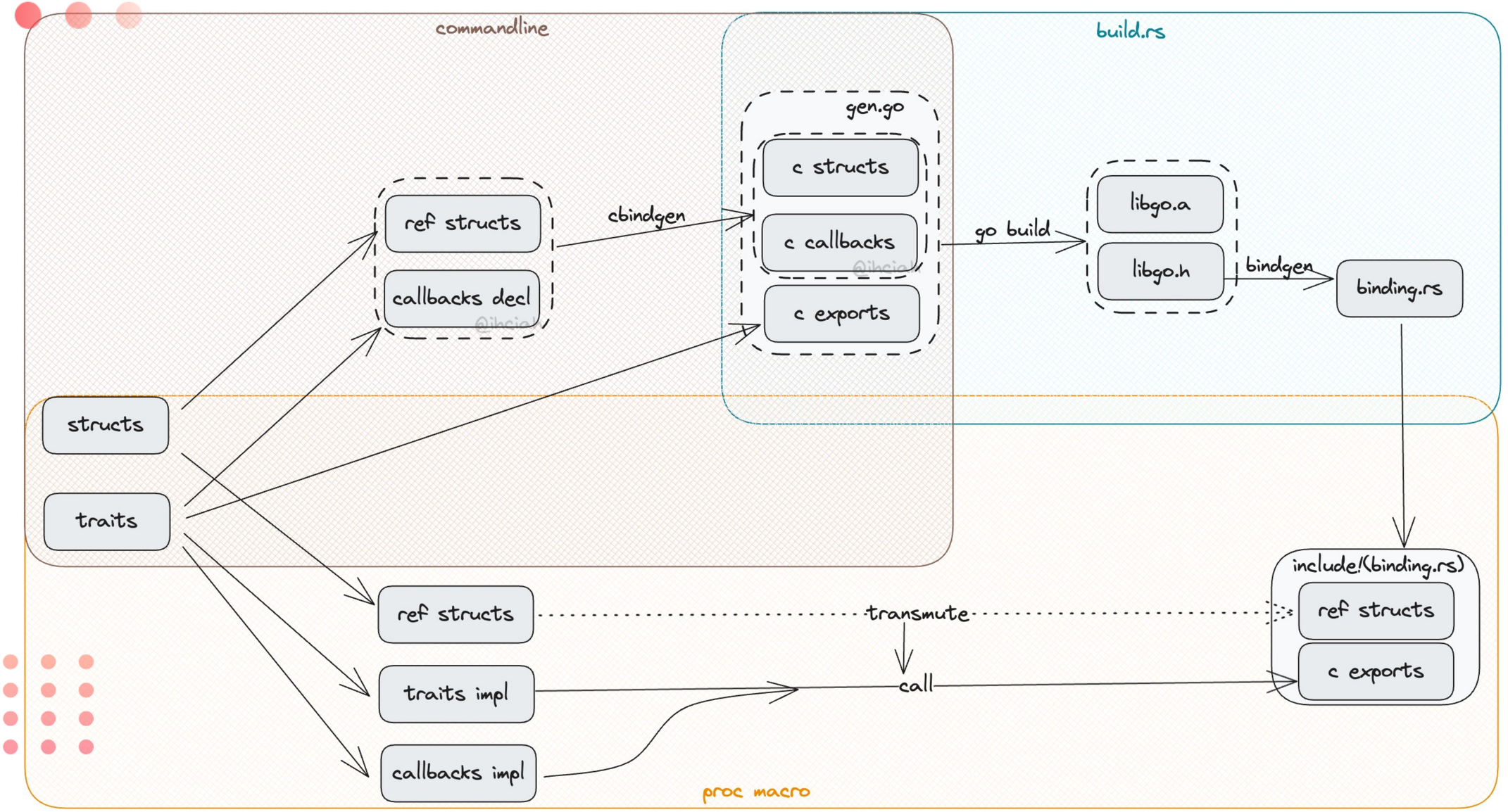
描述 FFI Call 与 Struct 定义

IDL？ Go Style？ Rust Style？ C Style？

选择 Rust 原生描述，可以利用 syn 方便 parse，并省略 Rust 侧代码生成，且不引入学习成本。

```rust
#[rust2go::r2g]
pub trait DemoCall {
    fn demo_oneway(req: &DemoUser);
    fn demo_check(req: &DemoComplicatedRequest) -> DemoResponse;
    fn demo_check_async(
        req: &DemoComplicatedRequest,
    ) -> impl std::future::Future<Output = DemoResponse>;
    #[drop_safe]
    fn demo_check_async_safe(
        req: DemoComplicatedRequest,
    ) -> impl std::future::Future<Output = DemoResponse>;
    #[drop_safe_ret]
    fn demo_check_async_safe2(
        req: DemoComplicatedRequest,
    ) -> impl std::future::Future<Output = DemoResponse>;
}
```

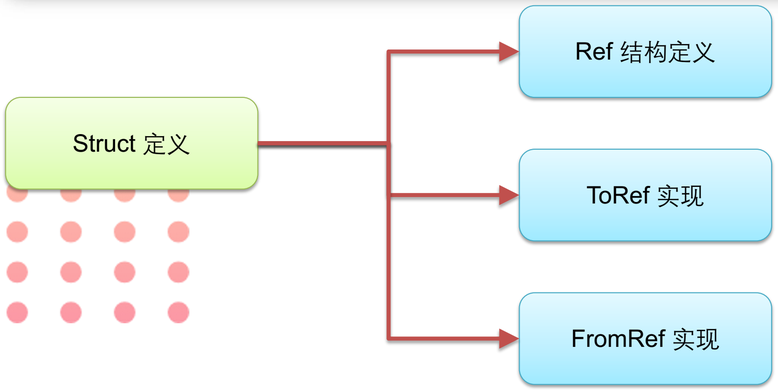
基于 cli 和过程宏的代码生成



基于 cli 和过程宏的代码生成 – 结构定义

Rust China Conf

```rust
#[derive(rust2go::R2G, Clone)]
pub struct DemoUser {
    pub name: String,
    pub age: u8,
}
```

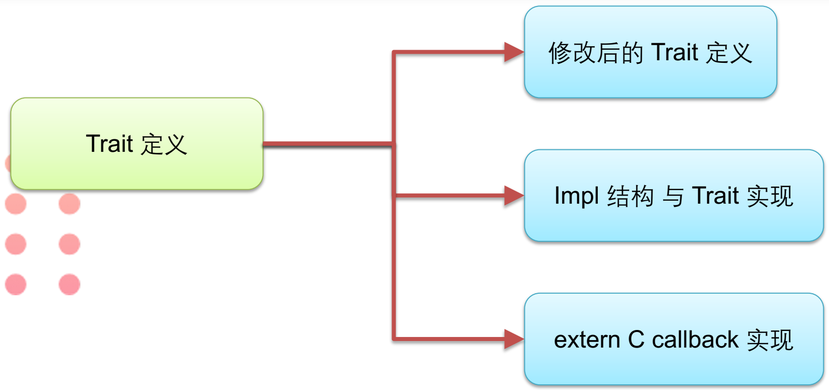



```rust
#[repr(C)]
pub struct DemoUserRef {
    name: ::rust2go::StringRef,
    age: u8,
}
impl ::rust2go::ToRef for DemoUser {
    const MEM_TYPE: ::rust2go::MemType = ::rust2go::max_mem_type!(String, u8);
    type Ref = DemoUserRef;
    fn to_size(&self, acc: &mut usize) {
        if matches!(Self::MEM_TYPE, ::rust2go::MemType::Complex) {
            self.name.to_size(acc);
            self.age.to_size(acc);
        }
    }
    fn to_ref(&self, buffer: &mut ::rust2go::Writer) -> Self::Ref {
        DemoUserRef {
            name: ::rust2go::ToRef::to_ref(&self.name, buffer),
            age: ::rust2go::ToRef::to_ref(&self.age, buffer),
        }
    }
}
impl ::rust2go::FromRef for DemoUser {
    type Ref = DemoUserRef;
    fn from_ref(ref_: &Self::Ref) -> Self {
        Self {
            name: ::rust2go::FromRef::from_ref(&ref_.name),
            age: ::rust2go::FromRef::from_ref(&ref_.age),
        }
    }
}
```

基于 cli 和过程宏的代码生成 – trait 定义

Rust China Conf

```rust
#[rust2go::r2g]
pub trait DemoCall {
    fn demo_oneway(req: &DemoUser);
    fn demo_check(req: &DemoComplicatedRequest) -> DemoResponse;
    fn demo_check_async(
        req: &DemoComplicatedRequest,
    ) -> impl std::future::Future<Output = DemoResponse>;
    #[drop_safe_ret]
    fn demo_check_async_safe(
        req: DemoComplicatedRequest,
    ) -> impl std::future::Future<Output = DemoResponse>;
}
```




```rust
pub trait DemoCall {
    fn demo_oneway(req: &DemoUser);
    fn demo_check(req: &DemoComplicatedRequest) -> DemoResponse;
    unsafe fn demo_check_async(req: &DemoComplicatedRequest) -> impl Future<Output = DemoResponse>;
}
pub struct DemoCallImpl;

impl DemoCall for DemoCallImpl {
    fn demo_oneway(req: &DemoUser) {
        let (_buf, req) = ::rust2go::ToRef::calc_ref(req);
        #[allow(clippy::useless_transmute)]
        unsafe {
            binding::CDemoCall_demo_oneway(::std::mem::transmute(req))
        }
    }
    fn demo_check(req: &DemoComplicatedRequest) -> DemoResponse {
        let mut slot = None;
        let (_buf, req) = ::rust2go::ToRef::calc_ref(req);
        #[allow(clippy::useless_transmute)]
        unsafe {
            binding::CDemoCall_demo_check(
                ::std::mem::transmute(req),
                &slot as *const _ as *const () as *mut _,
                Self::demo_check_cb as *const () as *mut _,
            )
        };
        slot.take().unwrap()
    }
    unsafe fn demo_check_async(
        req: &DemoComplicatedRequest,
    ) -> impl ::std::future::Future<Output = DemoResponse> {
        ...
    }
}
impl DemoCallImpl {
    #[allow(clippy::useless_transmute)]
    #[no_mangle]
    unsafe extern "C" fn demo_check_cb(resp: binding::DemoResponseRef, slot: *const ()) {
        *(slot as *mut Option<DemoResponse>) =
            Some(::rust2go::FromRef::from_ref(::std::mem::transmute(&resp)));
    }
    #[allow(clippy::useless_transmute)]
    #[no_mangle]
    unsafe extern "C" fn demo_check_async_cb(resp: binding::DemoResponseRef, slot: *const ()) {
        ...
    }
}
```

广泛基于泛型的 Golang 代码生成

Rust China Conf

examples > example-tokio > go > gen.go > ...

```go
func new_list_mapper[T1, T2 any](f func(T1) T2) func(C.ListRef) []T2 {
    return func(x C.ListRef) []T2 {
        input := unsafe.Slice((*T1)(unsafe.Pointer(x.ptr)), x.len)
        output := make([]T2, len(input))
        for i, v := range input {
            output[i] = f(v)
        }
        return output
    }
}
func new_list_mapper_primitive[T1, T2 any](_ func(T1) T2) func(C.ListRef) []T2 {
    return func(x C.ListRef) []T2 {
        return unsafe.Slice((*T2)(unsafe.Pointer(x.ptr)), x.len)
    }
}

// only handle non-primitive type T
func cnt_list_mapper[T, R any](f func(s *T, cnt *uint) [0]R) func(s *[]T, cnt *uint) [0]C.ListRef {
    return func(s *[]T, cnt *uint) [0]C.ListRef {
        for _, v := range *s {
            f(&v, cnt)
        }
        *cnt += uint(len(*s)) * size_of[R]()
        return [0]C.ListRef{}
    }
}

// only handle primitive type T
func cnt_list_mapper_primitive[T, R any](_ func(s *T, cnt *uint) [0]R) func(s *[]T, cnt *uint) [0]C.ListRef {
    return func(s *[]T, cnt *uint) [0]C.ListRef { return [0]C.ListRef{} }
}
```

实现效果

Rust China Conf

Step 1: 定义参数和返回值

```rust
pub mod binding {
    #![allow(warnings)]
    rust2go::r2g_include_binding!();
}

// Define your own structs. You must derive `rust2go::R2G` for each struct.
#[derive(rust2go::R2G, Clone)]
pub struct DemoUser {
    pub name: String,
    pub age: u8,
}

// Define your own structs. You must derive `rust2go::R2G` for each struct.
#[derive(rust2go::R2G, Clone)]
pub struct DemoComplicatedRequest {
    pub users: Vec<DemoUser>,
    pub balabala: Vec<u8>,
}

// Define your own structs. You must derive `rust2go::R2G` for each struct.
#[derive(rust2go::R2G, Clone, Copy)]
pub struct DemoResponse {
    pub pass: bool,
}
```

call.rs

实现效果

Rust China Conf

Step 2: 定义 Call Trait

```rust
#[rust2go::r2g]
pub trait DemoCall {
    fn demo_oneway(req: &DemoUser);
    fn demo_check(req: &DemoComplicatedRequest) -> DemoResponse;
    fn demo_check_async(
        req: &DemoComplicatedRequest,
    ) -> impl std::future::Future<Output = DemoResponse>;
    #[drop_safe_ret]
    fn demo_check_async_safe(
        req: DemoComplicatedRequest,
    ) -> impl std::future::Future<Output = DemoResponse>;
}
```

call.rs

实现效果

Rust China Conf

Step 3: 生成 Go 代码并实现 Go Interface

```
rust2go-cli --src src/call.rs --dst go/gen.go
```

```go
package main

import (
    "fmt"
    "time"
)

type Demo struct{}

func init() {
    DemoCallImpl = Demo{}
}

func (Demo) demo_oneway(req DemoUser) {
    fmt.Printf("[Go-oneway] Golang received name: %s, age: %d\n", req.name, req.age)
}

func (Demo) demo_check(req DemoComplicatedRequest) DemoResponse {
    fmt.Printf("[Go-call] Golang received req: %d users\n", len(req.users))
    fmt.Printf("[Go-call] Golang returned result\n")
    return DemoResponse{pass: true}
}

func (Demo) demo_check_async(req DemoComplicatedRequest) DemoResponse {
    fmt.Printf("[Go-call async] Golang received req, will sleep 1s\n")
    time.Sleep(1 * time.Second)
    fmt.Printf("[Go-call async] Golang returned result\n")
    return DemoResponse{pass: true}
}

func (Demo) demo_check_async_safe(req DemoComplicatedRequest) DemoResponse {
    fmt.Printf("[Go-call async drop_safe] Golang received req, will sleep 1s\n")
    time.Sleep(1 * time.Second)
    resp := DemoResponse{pass: req.balabala[0] == 1}
    return resp
}
```

impl.go

实现效果

Rust China Conf

Step 4: 添加 build.rs 并正常调用

```rust
fn main() {
    rust2go::Builder::new()
        .with_go_src("./go")
        .build();
}
```

build.rs

```rust
mod user;

use user::{DemoCall, DemoCallImpl, DemoComplicatedRequest, DemoUser};

#[tokio::main]
async fn main() {
    let user = DemoUser {
        name: "chihai".to_string(),
        age: 28,
    };
    DemoCallImpl::demo_oneway(&user);
    println!("[Rust-oneway] done");

    let req = DemoComplicatedRequest {
        users: vec![user.clone(), user],
        balabala: vec![1],
    };
    println!(
        "[Rust-sync] User pass: {}",
        DemoCallImpl::demo_check(&req).pass
    );
}
```

main.rs

实现效果

Rust China Conf

Step 5: 常规方式编译 / 运行即可

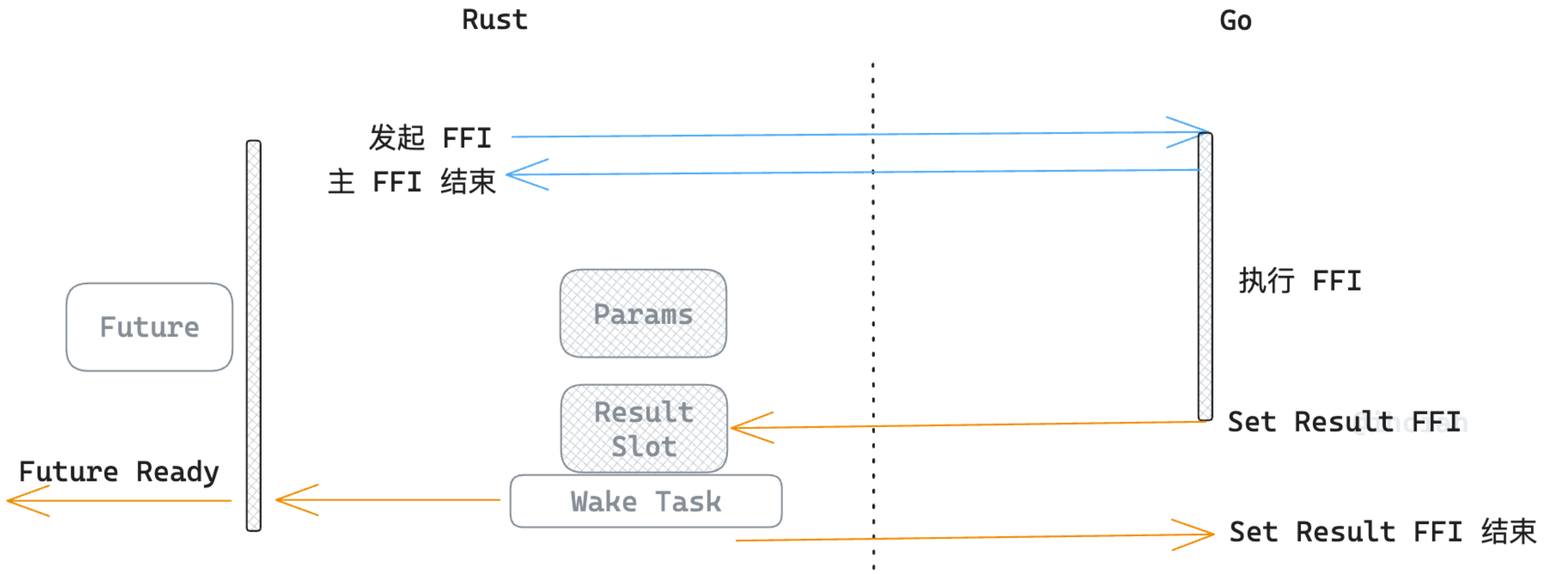
```
   Compiling example-tokio v0.1.0 (/home/ihciah/code/ihciah/rust2go/examples/example-tokio)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 4.02s
     Running `target/debug/example`
[Go-oneway] Golang received name: chihai, age: 28
[Rust-oneway] done
[Go-call] Golang received req: 2 users
[Go-call] Golang returned result
[Rust-sync] User pass: true
 *  终端将被任务重用，按任意键关闭。
```

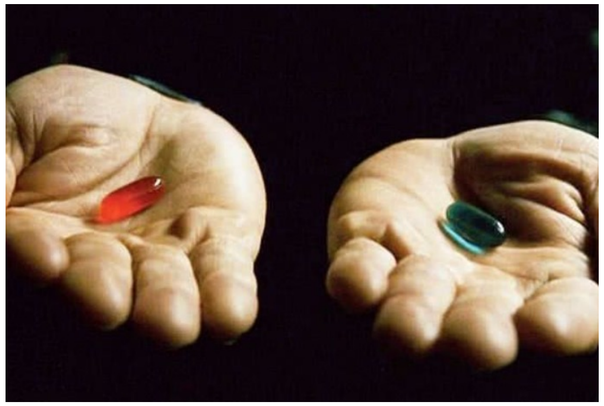
04

性能优化

使用基于共享内存的通信方式替代 CGO

Rethinking Async FFI

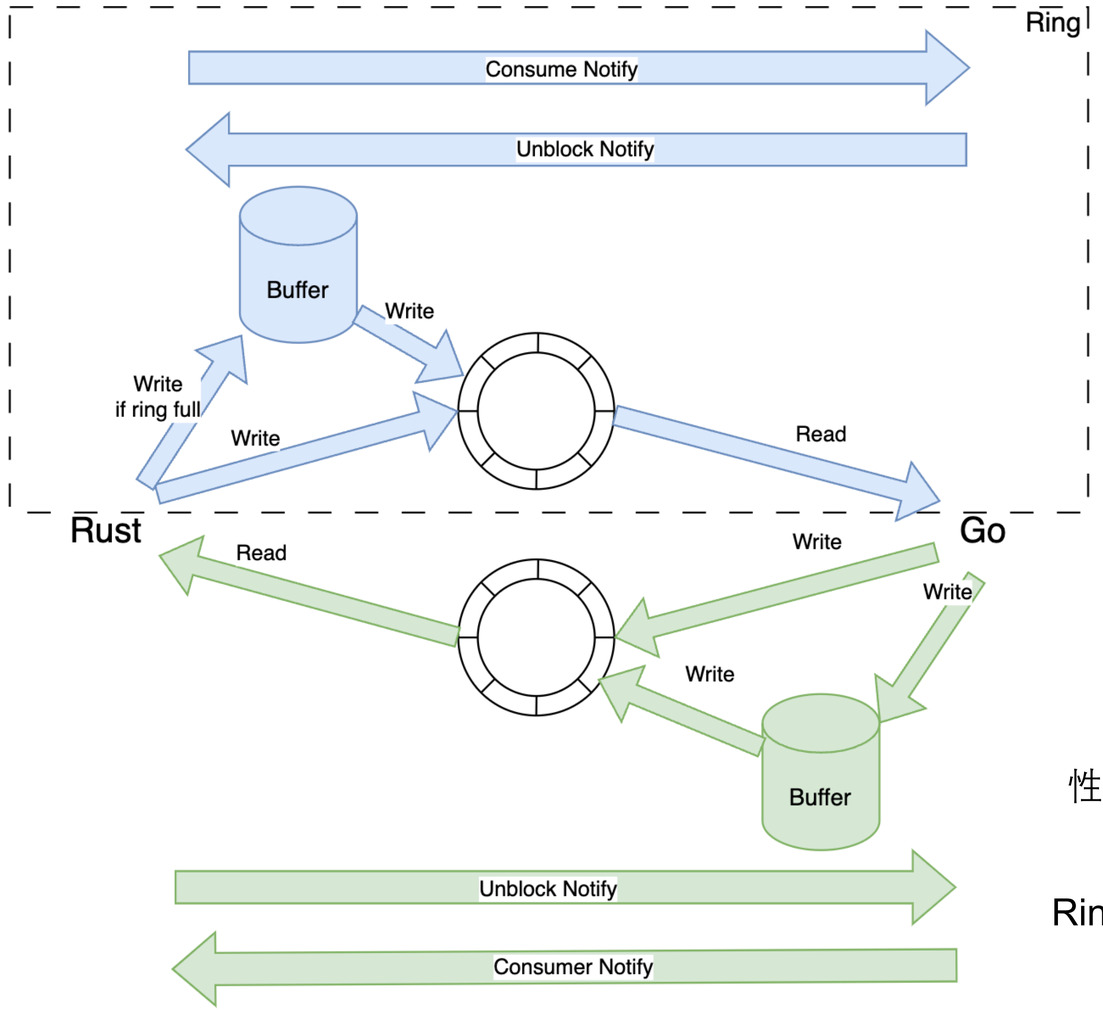




Blue Pill: 基于同步 FFI 实现异步 FFI，这很好!

Red Pill: 同步 FFI 与异步 FFI 完全不同，异步 FFI 是一种通信!

基于共享内存的 FFI



1.基于双无锁 Ring 实现的双向通信
2.基于 eventfd / UDS 实现的通知机制
3.后备 buffer 提供准确发送保障
4.Consume/Unblock Notify 事件驱动

性能：Go 1.18 下相比基于 CGO 版本最多 CPU 降低 20%

Ring 也可以单独使用：https://crates.io/crates/mem-ring

落地项目

项目开源地址：https://github.com/ihciah/rust2go

Rust China Conf

字节 API 网关 Rust 新版本	支撑了几乎所有 app HTTP 入流量处理与协议转换 依赖 Rust2Go 执行通用 Go 插件（核心能力之一）
字节大模型安全网关	支撑豆包等大模型服务假名化能力 依赖 Rust2Go 开发业务逻辑
某开源Rust 区块链应用	依赖 Rust2Go 集成 Golang 测试
某开源Rust HTTP 客户端	基于某开源 Golang 客户端封装，支持细粒度控制能力 依赖 Rust2Go 提供核心 FFI 调用能力

未来规划

项目开源地址：https://github.com/ihciah/rust2go

Rust China Conf

- 支持 Go -> Rust 调用
- 支持更高性能的结构转换
- 发掘更多应用场景和用户
- 提升框架易用性

其他工作

Service Async
纯异步 Service

DynThrift
高性能协议转换

CertainMap
可靠的跨Service 信息传递

Monoio-
http/tls/…

MonoLake：高性能代理框架 – 即将开源

Monoio：高性能 Rust Async Runtime – 已开源

Thanks

更多阅读:

1. https://github.com/ihciah/rust2go
2. https://www.ihcblog.com/rust2go
3. https://www.v2ex.com/t/1070796

欢迎 Follow/Star！

也欢迎找我讨论（ihciah@gmail.com）！