RUST CHINA CONF 2020

首届中国 Rust 开发者大会

2020.12.26–27 深圳

Rust China Conf 2020

Shenzhen, China

2020conf.rustcc.cn

用于 Rust 的嵌入式脚本语言

Embedded scripting language for Rust

陈天泽 icey@icey.tech
GitHub: ICEYSELF

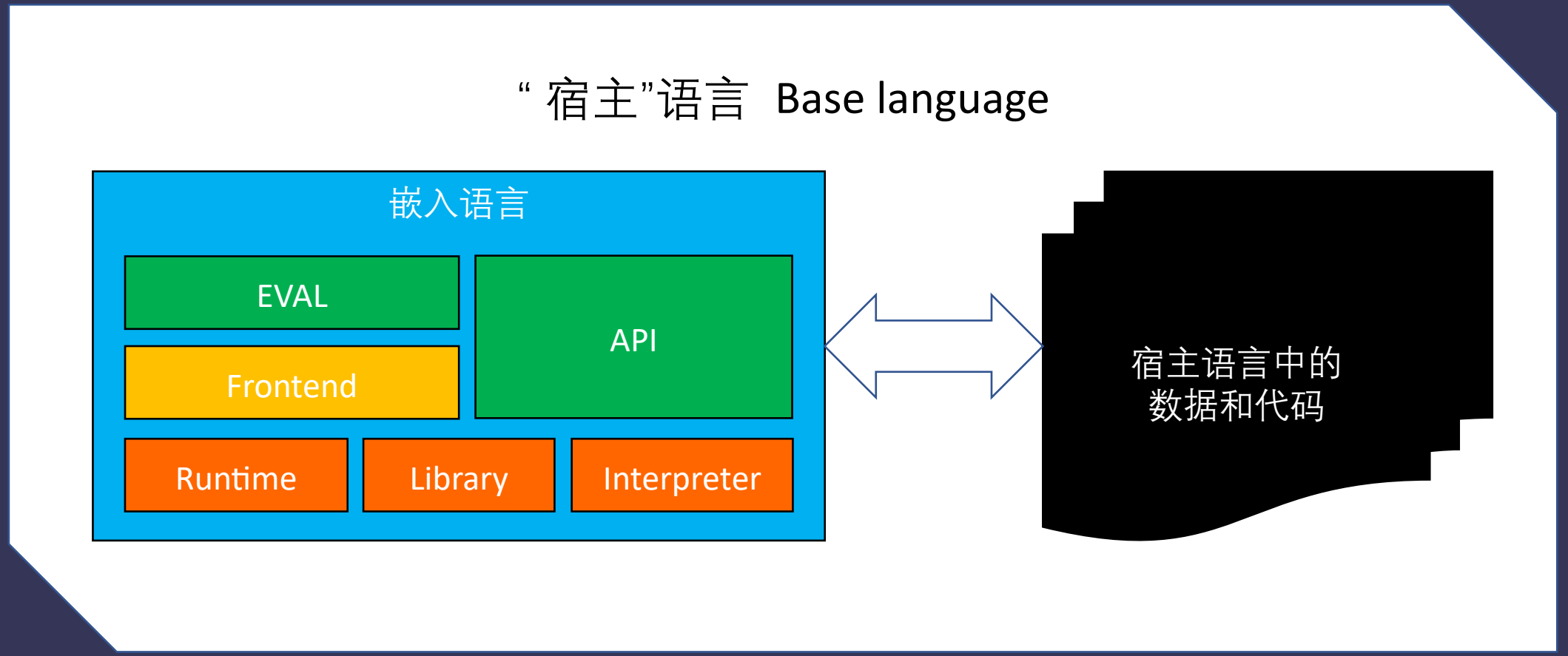
Embedding: What and Why

何为嵌入式语言

Definition from wikipedia:

An **embedded style language** is a kind of computer language whose commands appear intermixed with those of a base language. Such languages can either have their own syntax, which is translated into that of the base language, or can provide an API with which to invoke the behaviors of the language. Embedded domain-specific languages are common examples of embedded style languages that rely upon translation.

太长不看版

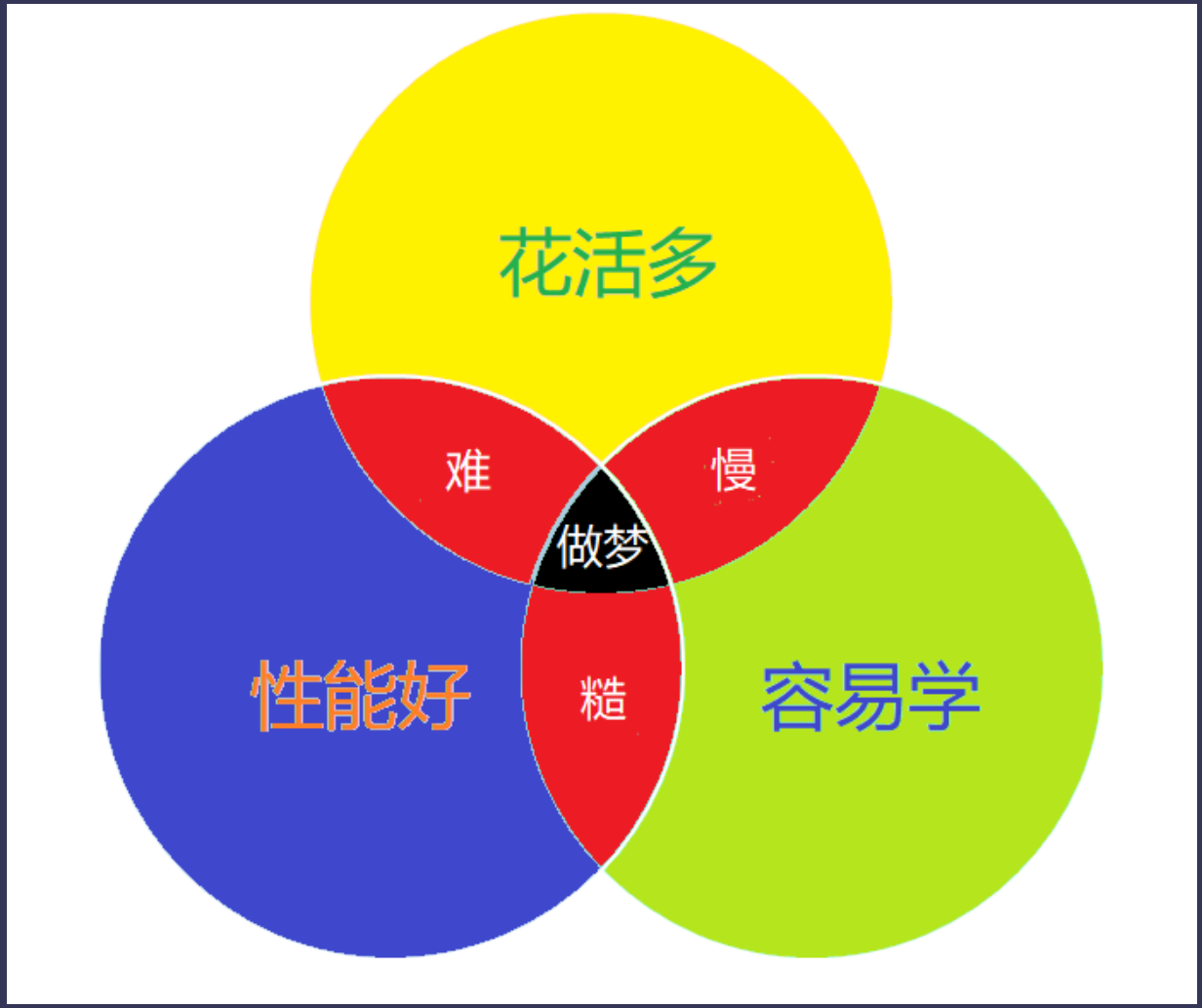

“ 宿主”语言 Base language
嵌入语言
EVAL
Frontend
API
Runtime
Library
Interpreter
宿主语言中的
数据和代码

为什么需要嵌入语言

> I can't seem to understand the reason as to why multiple programming languages are used in the same product or software?

It is quite simple: **there is no single programming language suitable for all needs and goals.**

https://softwareengineering.stackexchange.com/questions/370135/why-are-multiple-programming-languages-used-in-the-development-of-one-product-or/370146#370146

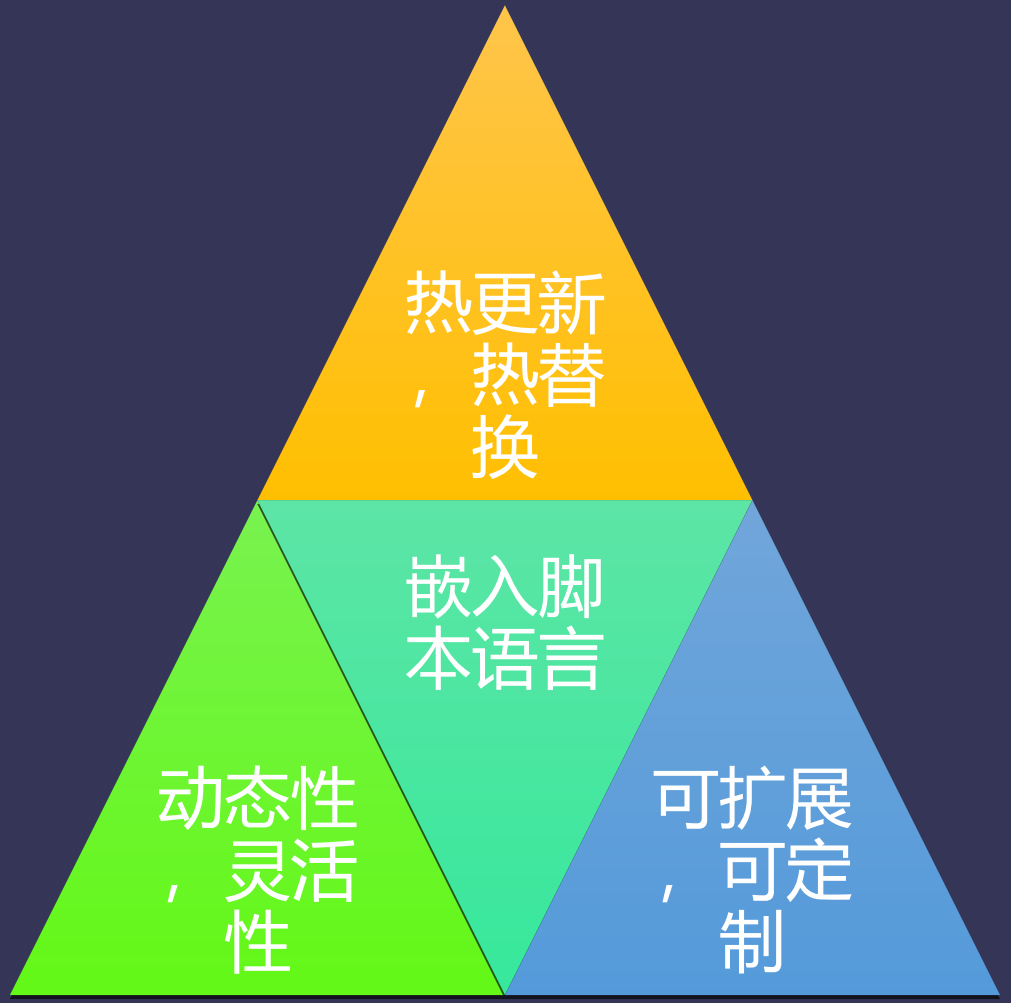


为什么需要嵌入语言


AUTOCAD


The
Battle for Wesnoth


Office 365


热更新，热替换
嵌入脚本语言
动态性，灵活性
可扩展，可定制


unity

Rust: Existing Solutions

现有解决方案

```rust
// Normal function that returns a standard type
// Remember to use 'ImmutableString' and not 'String'
fn add_len(x: i64, s: ImmutableString) -> i64 {
    x + s.len()
}
// Alternatively, '&str' maps directly to 'ImmutableString'
fn add_len_str(x: i64, s: &str) -> i64 {
    x + s.len()
}

// Function that returns a 'Dynamic' value - must return a 'Result'
fn get_any_value() -> Result<Dynamic, Box<EvalAltResult>> {
    Ok((42_i64).into())                    // standard types can use 'into()'
}

let mut engine = Engine::new();

engine
    .register_fn("add", add_len)
    .register_fn("add_str", add_len_str);

let result = engine.eval::<i64>(r#"add(40, "xx")"#)?;

println!("Answer: {}", result);             // prints 42
```

Project-47

47 工程 (Project-47)

- 为 Rust 量身打造
  - 在 Rust Result 和 Pr47 Exception 之间自动转换
  - 在 Rust Option 和 Pr47 Null value 之间自动转换
  - 支持绑定引用类型 &T, &mut T ，将数据在 Rust 和 Pr47 之间共享
  - 复制 Copy 类型，移动 !Copy 类型
- 半静态类型系统
  - 有助于“编译期”检查，提高程序可靠性
  - 有助于“编译期”优化，提高程序运行性能
  - 仍然能提供不错的动态性
- 提供垃圾回收

```rust
fn foo(a: &Type1, b: &mut Type2) -> i64 {
    /* ... */
}

/* ... */

let mut compiler = compiler::Compiler::new();
compiler.register_type::<Type1>("T1");
compiler.register_type::<Type2>("T2");
compiler.register_fn("foo", foo);
let mut program = compiler.compile(r#"
    func application_start() {
        var t1 T1 = /*...*/;
        var t2 T2 = /*...*/;
        print(foo(t1, t2));
    }
"#);

program.run(&mut vm::VM::new());
```

处理 Rust 所有权和生存期

Useful

Garbage

```
fn foo(t1: MoveType) { /* MoveType is !Copy */ }
fn bar(t2: CopyType) { /* CopyType is Copy */ }
fn baz(t3: &NobodyCares, t4: &mut WhoCares) { /* Passing by reference */ }
```

Moved

Shared

Mut Shared

不同类型需要不同的代码

```rust
unsafe fn take_value<T /*not ref*/>(value: &vm::Value) -> T {
    value.mark_as_moved();
    let void_ptr: *mut() = value.get_ptr();
    *Box::from_raw(void_ptr as *mut T)
}

unsafe fn get_as_ref<T>(value: &vm::Value) -> &T {
    value.mark_as_borrowed();
    let void_ptr: *mut () = value.get_ptr();
    (void_ptr as *mut T as *const T)
}
```

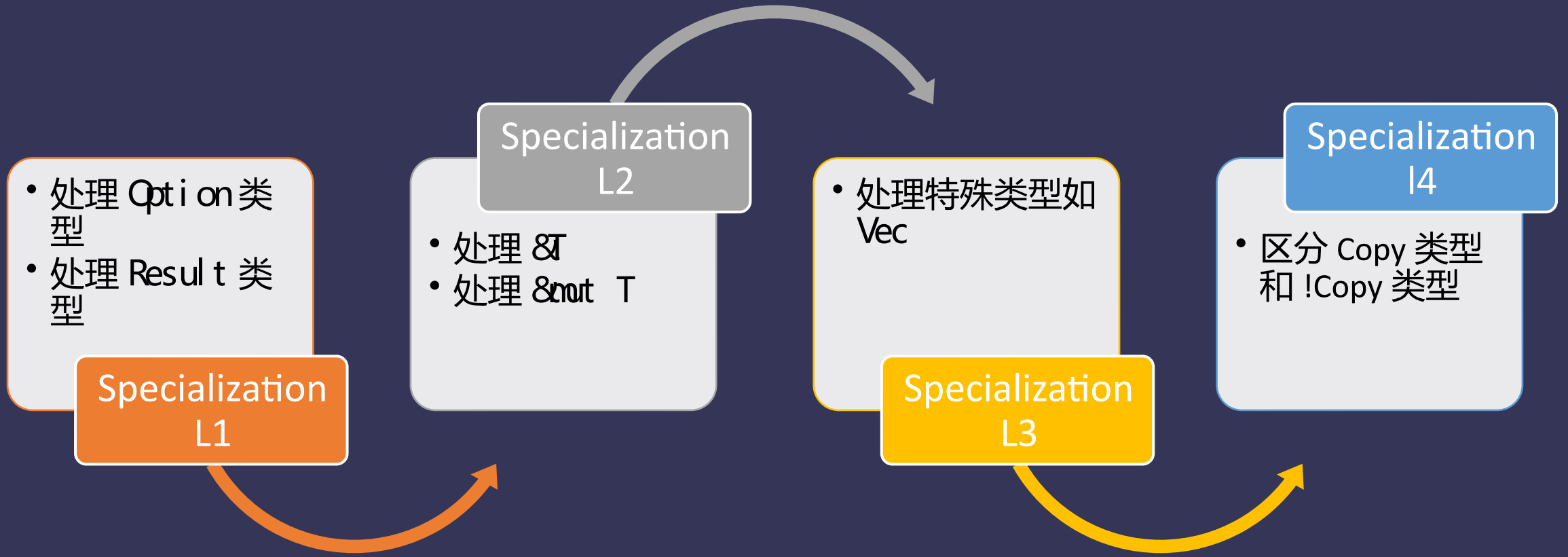
#[feature(specialization)]

```rust
trait Greeting {
    fn greeting(&self);
}

impl<T> Greeting for T {
    default fn greeting(&self) {
        println!("Hi")
    }
}

impl Greeting for Russkiy {
    fn greeting(&self) {
        println!("Privet")
    }
}
```

Specialization: 为不同类型提供不同实现



```rust
pub trait FromValue<'a, T> { /*...*/ }
pub trait FromValueL1<'a, T> { /*...*/ }
pub trait FromValueL2<'a, T> { /*...*/ }
pub trait FromValueL3<'a, T> { /*...*/ }

impl<'a, T> FromValue<'a, T> for Void where Void: FromValueL1<'a, T> { /*...*/ }
/*spec*/ impl<'a, T> FromValue<'a, Option<T>> for Void where Void: FromValueL1<'a, T>

impl<'a, T> FromValueL1<'a, T> for Void where Void: FromValueL2<'a, T> { /*...*/ }
/*spec*/ impl<'a, T> FromValueL1<'a, &'a T> for Void where Void: FromValueL2<'a, T> { /*...*/ }
/*spec*/ impl<'a, T> FromValueL1<'a, &'a mut T> for Void where Void: FromValueL2<'a, T> { /*...*/ }

impl<'a, T> FromValueL2<'a, T> for Void where Void: FromValueL3<'a, T> { /*...*/ }
/*spec*/ impl<'a> FromValueL2<'a, i64> for Void { /*...*/ }
/*spec*/ impl<'a> FromValueL2<'a, f64> for Void { /*...*/ }

impl<'a, T> FromValueL3<'a, T> for Void where Void: StaticBase<T> { /*...*/ }
/*spec*/ impl<'a, T> FromValueL3<'a, T> for Void where Void: StaticBase<T>, T: Copy { /*...*/ }
```

静态类型，静态分析

```
func untyped_add(
    return a + b
}
```

```
func typed_add(a int, b int) int {
    return a + b
}
```

静态类型，静态检查

```
func untyped_add(a any, b any) any {
    return a + b
}
```

```
%1 = load-any @a
%2 = load-any @b
%t1 = typeof %1
%t2 = typeof %2
check-addable %t1, %t2
%3 = dispatch add, %1, %2
ret %3
```

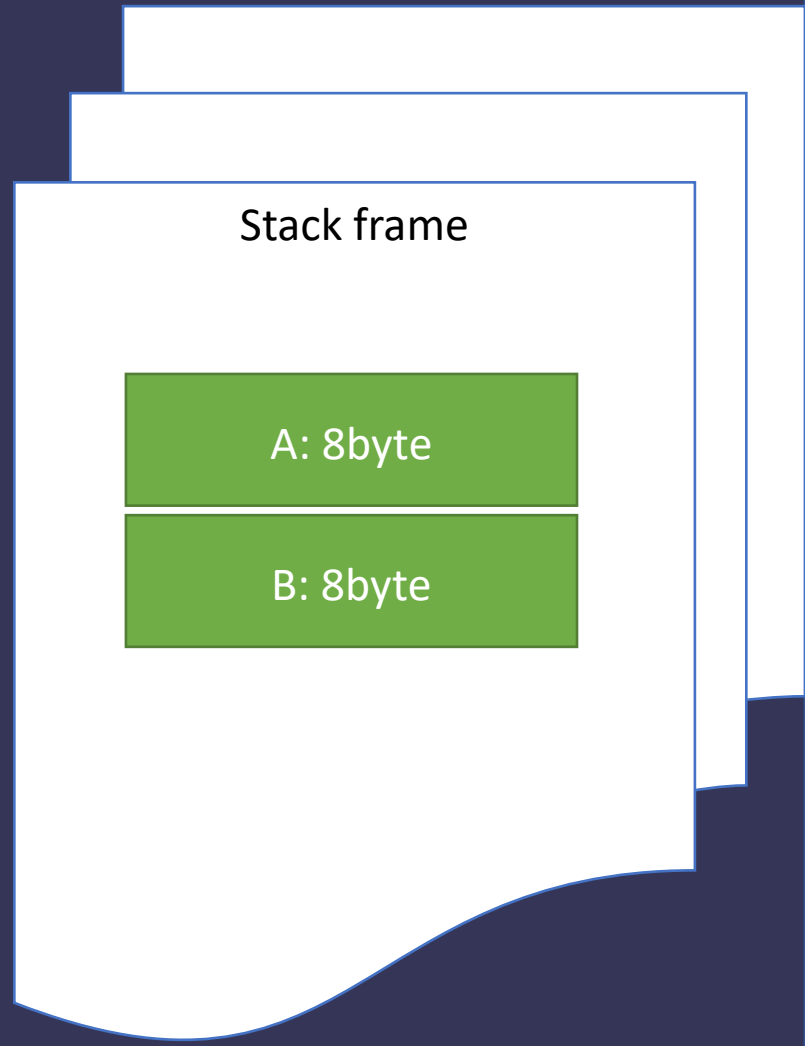
静态类型，静态检查

```
func typed_add(a int, b int) int {
    return a + b
}
```

```
%1 = load-int @a
%2 = load-int @b
%3 = add-int %1, %2
ret %3
```

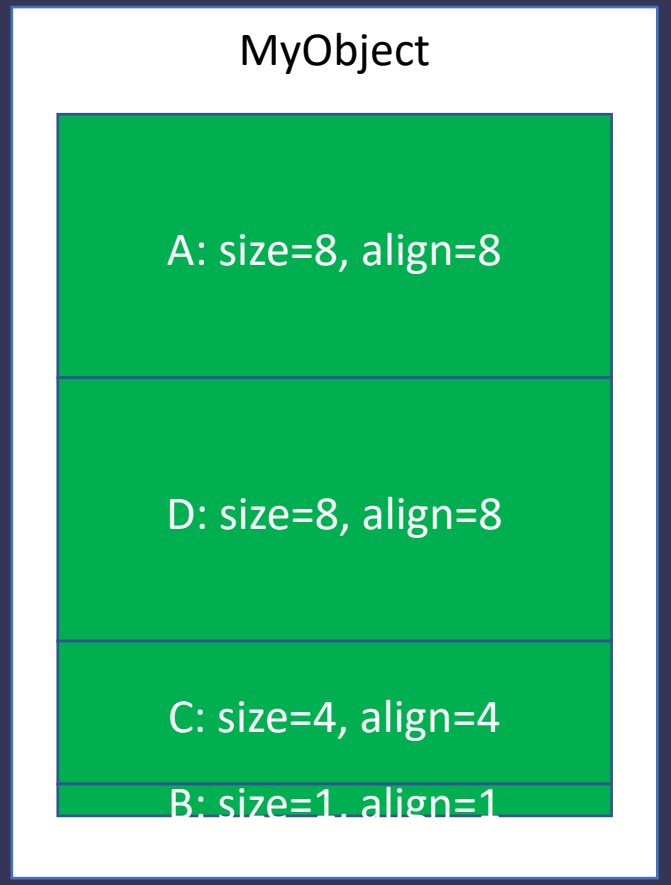
静态类型，确定分配

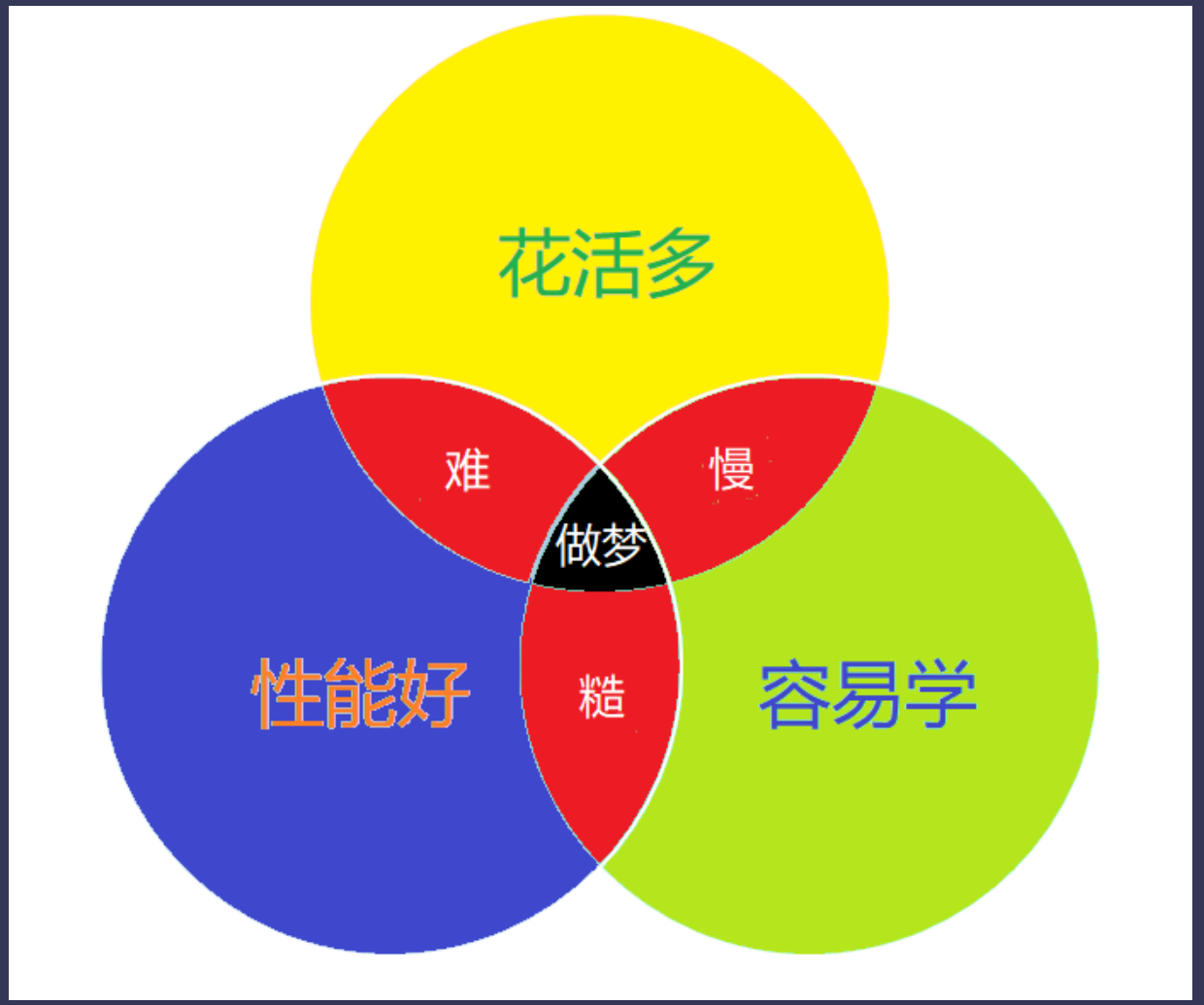
```
func typed_add(a int, b int) int {
    return a + b
}
```



静态类型，确定分配

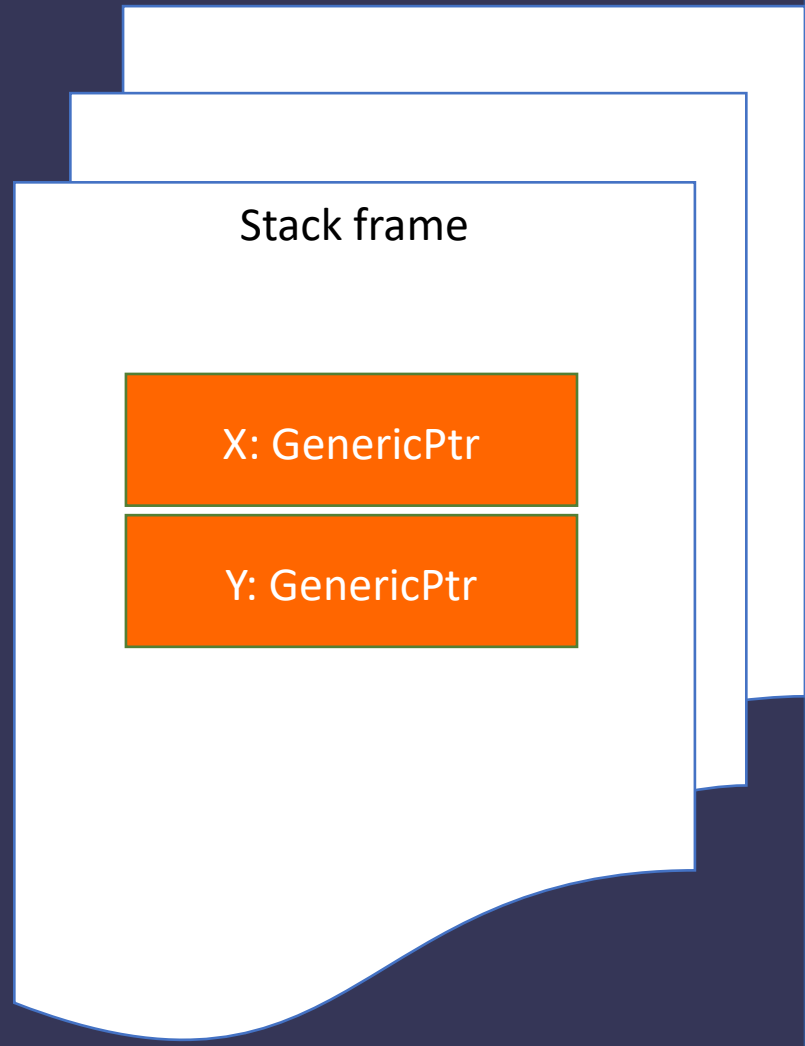
```
struct MyObject {
    a int,
    b bool,
    c char,
    d float
}
```




花活多
难
慢
做梦
性能好
糙
容易学

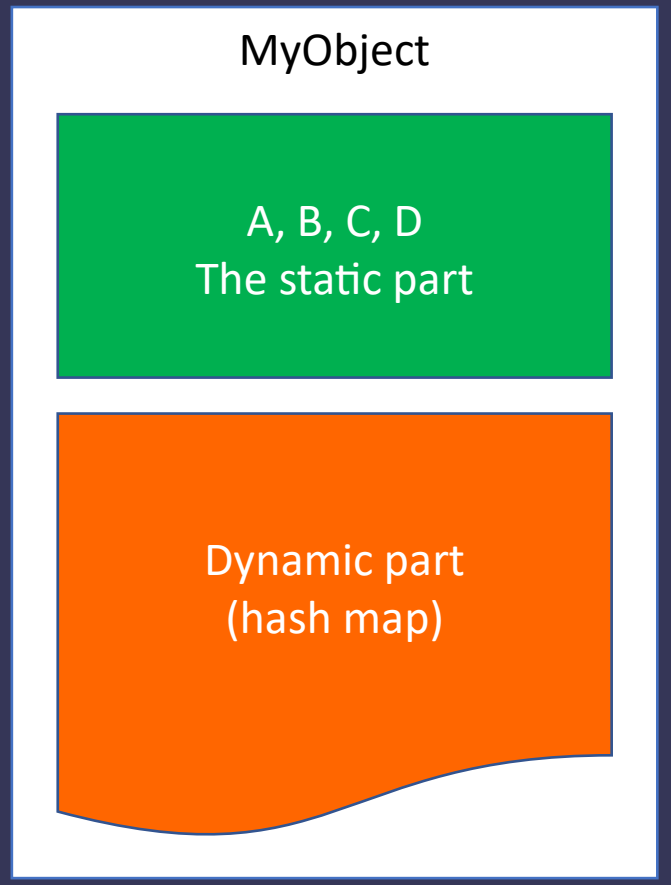
走向动态：动态类型

```
var x = createSomeObject();
var y any = createSomeObject();
```



走向动态：运行时可变的 fields

```
object MyObject {
    a int,
    b bool,
    c char,
    d float
}
```



走向动态：运行时重载

```
struct S1 { /* ... */ }
struct S2 { /* ... */ }

func foo(s1 S1, ...) { /* ... */ }
func foo(s2 S2, ...) { /* ... */ }

var s = functionReturnsEitherS1OrS2();
s.foo();
```

Remaining Problems

首先，#[allow(incomplete_features)]

Specialization 还没稳定，甚至...还没 complete

nikomatsakis commented on 24 Feb 2016 • edited ▾ Contributor •••

This is a tracking issue for specialization (rust-lang/rfcs#1210).

Major implementation steps:

- [x] Land #30652 =)
- [ ] Restrictions around lifetime dispatch (currently a **soundness hole**)
- [ ] `default impl` (#37653)
- [ ] Integration with associated consts
- [ ] Bounds not always properly enforced (#33017)
- [ ] Should we permit empty impls if parent has no `default` members? #48444
- [ ] implement "always applicable" impls #48538
- [ ] describe and test the precise cycle conditions around creating the specialization graph (see e.g. this comment, which noted that we have some very careful logic here today)

Assignees

No one assigned

Labels

A-specialization A-traits B-RFC-approved B-RFC-implemented B-unstable C-tracking-issue F-specialization T-lang

Projects

None yet

其次， Soundness 很重要

```rust
struct S ();

struct Foo<'a> {
    inner: RefCell<&'a S>
}

fn leak<'a>(foo: &'a Foo<'a>, s: &'a S) {
    foo.inner.replace(s);
}
```

```
func createLeak(foo Foo) {
  var s S = createS();
  leak(foo, s);
}

func application_start() {
  var foo Foo = createFoo();
  createLeak(foo);
  // garbage collection
  useFoolnner(foo);
}
```

ICEYSELF commented 9 days ago • edited Member Author •••

[MID] According to the talk with schungx the maintainer of Rhai, we've found that it is difficult (almost impossible) to support reference parameters, `T: 'a` andd `Fn: 'a` all together. By this time,

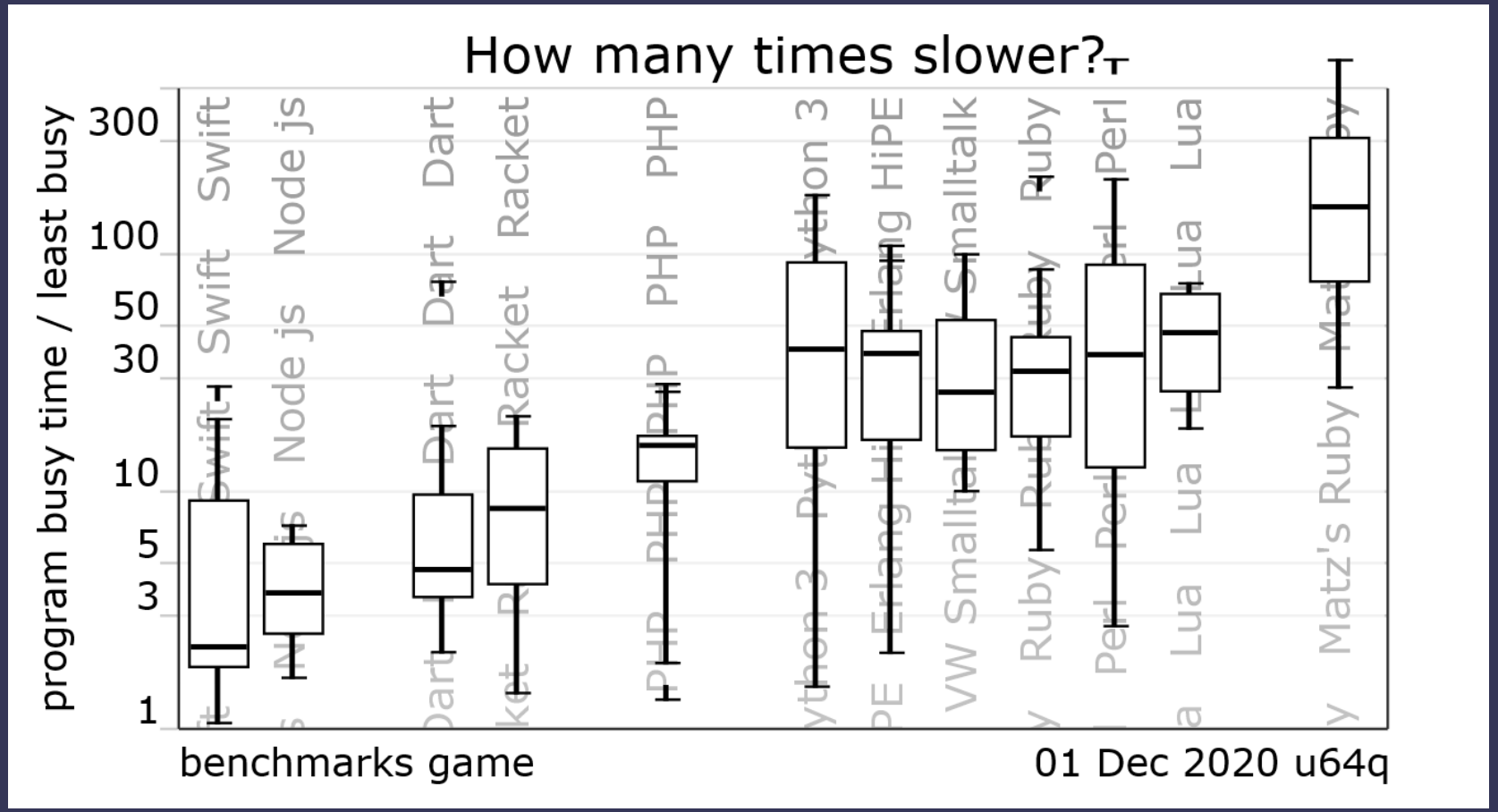
> **[schungx]**: I'd expect that the major problem with references will be to deal with lifetimes. Right now Rhai cheats by making all functions 'static. That means you can never return something that contains a reference unless it is 'static.

In fact, even in Rhai, without certain restrictions there can be problems:

```
fn unsound<'a>(orig: &'a mut i64) {
    let the_cell: RefCell<&'a mut i64> = RefCell::new(orig);
    let unsound_closure = |r: &'a mut i64| {
        the_cell.replace(r);
    };
    // rhai.register(unsound_closure);
}
```

If the `register` function does not restrict the function to be `'static`, the code above can silently create a dangling pointer. As a result the binding of Rhai is limited. Such limitations can also be used by Pr47 to improve the overall soundness.

最后，性能也很重要


How many times slower?
program busy time / least busy
300
100
50
30
10
5
3
1
Swift
Node js
Dart
Racket
PHP
Python 3
Erlang HiPE
VW Smalltalk
Ruby
Perl
Lua
Matz's Ruby
benchmarks game
01 Dec 2020 u64q

- DR0012: Severe issue when sharing data with Rust FFI functions [Bug] [Level-3] [Soundness]
  #14 opened 18 days ago by ICEYSELF
- I0009: 现代诗 [invalid] [wontfix]
  #11 opened 23 days ago by ICEYSELF
- P0008: Nullable value types [Level-2] [Proposal]
  #9 opened on 14 Nov by ICEYSELF
- DR0007: Nullability on value types [Bug] [Level-2]
  #8 opened on 14 Nov by ICEYSELF
- P0005: Reference/pointer to value typed objects [Level-3] [Proposal] [Soundness]
  #5 opened on 13 Nov by ICEYSELF
- DR0003: Semantics of value-typed objects [Bug] [Level-2] [help wanted]
  #3 opened on 11 Nov by ICEYSELF
- P0002: Structural & checked exception for Pr47 [Level-1] [Proposal] [enhancement] [help wanted]
  #2 opened on 10 Nov by ICEYSELF
- I0001: nullability/lifetime marker [Level-1] [enhancement] [help wanted]
  #1 opened on 3 Nov by ICEYSELF

感谢聆听
THANKS

语言设计文档 : https://github/Pr47/doc47
POC 实现 : https://github.com/Pr47/T10

RUST CHINA CONF 2020

首届中国 Rust 开发者大会

2020.12.26–27 深圳