Roto

A fast and safe scripting language for Rust

Oct 09 2025, EuroRust

Terts Diepraam, NLnet Labs

Introduction

Terts Diepraam

Software Engineer at NLnet Labs

Organizer of RustWeek

NLnet Labs

Non-profit organization

Been around for 25 years

Focused on DNS and routing

NSD, Unbound, Routinator, etc.

NLnet Labs

Historically all in C

All *new* projects are in Rust

E.g. Cascade, Krill, Routinator, Rotonda and more

Rotonda

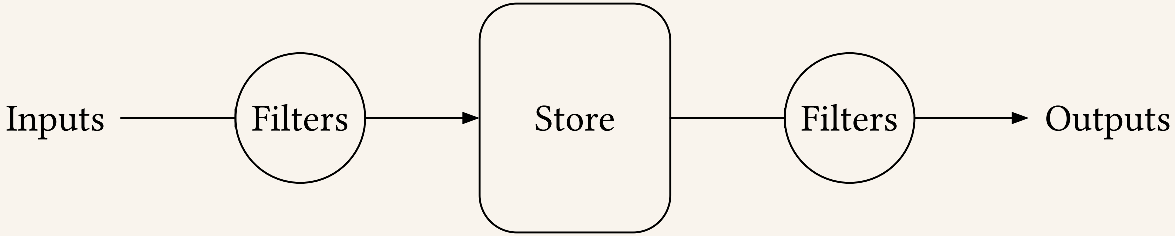
A BGP collector written in Rust

read: specialized database

Lots of data coming in and going out

We want to give people flexibility!

Rotonda: simplified



What language do users write those filters in?

Solution: scripting language?

All have one or more drawbacks:

- Too constrained
- Too slow
- No type checking

We obviously did the only sensible thing…

… we made our own

Enter: **Roto**

Roto in a nutshell

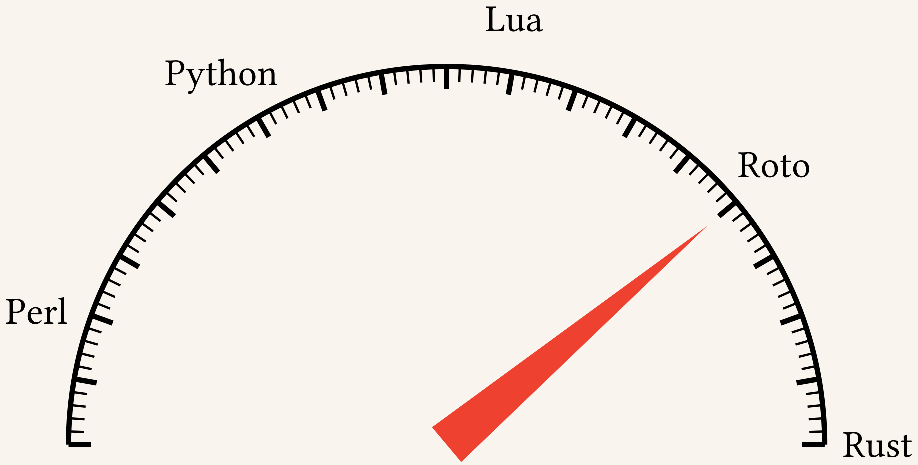
Embedded in Rust applications

Statically & strongly typed

Friendly error messages

Compiled to machine code

Handwavy speed expectations



(not benchmarked much yet, but looks good so far)

Example: A simple script

We write a simple Roto script...

```
fn clamp(x: i32) -> i32 {
    print(f"Got the value: {x}");
    if x > 100 {
      print("It's too big!");
      x = 100;
    }
    x
}
```

Example: compiling a script

…then we compile it and run it from Rust!

```rust
use roto::Runtime;
let rt = Runtime::new();
let mut pkg = rt.compile("script.roto").unwrap();
let f = pkg.get_function::<_, fn(i32) -> i32>("clamp").unwrap();
let y = f.call(&mut (), x);
```

Example: Error messages

```
fn clamp(x: i32) -> i32 {
    print(f"Got the value: {x}");
    if x > 100 {
      print("It's too big!");
      x = 100.0;
    }
    x
}
```

Example: Error messages (cont'd)

```
Error: Type error: mismatched types
   ╭─[ script.roto:5:13 ]
   │
 5 │        x = 100.0;
   │            ──┬──
   │              ╰──── expected `i32`, found `{float}`
───╯
```

I try to channel my inner @ekuber for the messages.

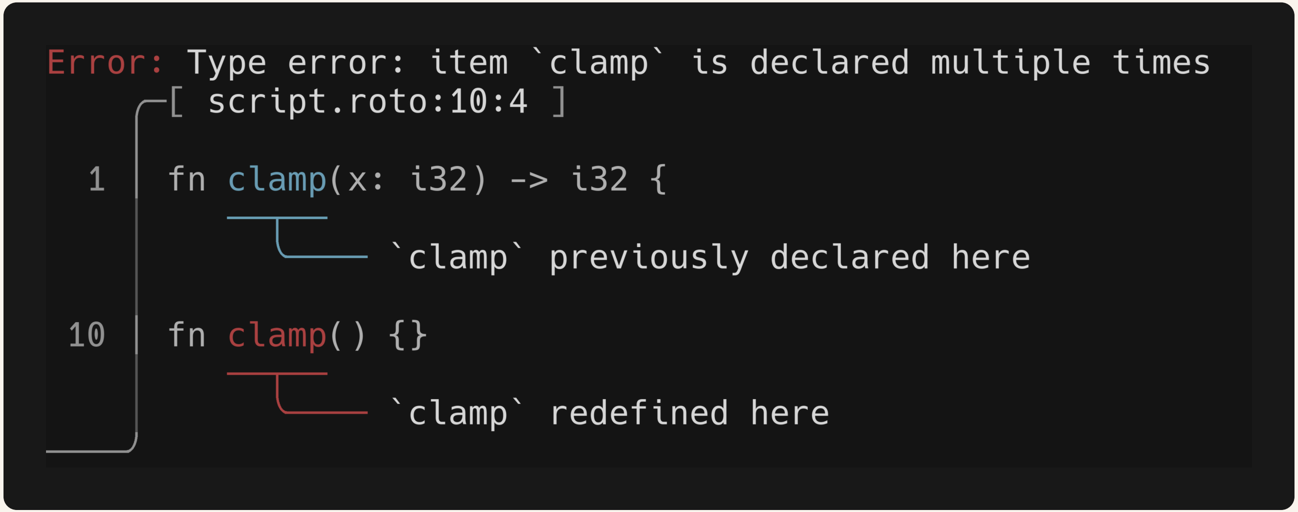
The output is powered by `ariadne`

Example: Error messages 2

```
fn clamp(x: i32) -> i32 {
    print(f"Got the value: {x}");
    if x > 100 {
      print("It's too big!");
      x = 100;
    }
    x
}

fn clamp() {}
```

Example: Error messages



```
Error: Type error: item `clamp` is declared multiple times
    ╭─[ script.roto:10:4 ]
    │
  1 │ fn clamp(x: i32) -> i32 {
    │    ──┬──
    │      ╰──── `clamp` previously declared here
    │
 10 │ fn clamp() {}
    │    ──┬──
    │      ╰──── `clamp` redefined here
────╯
```

How it works

We have the following steps:

- Parsing
- Type checking
- Lowering to Cranelift IR
- Compile to machine code

Cranelift is great!

The unsafest unsafe

But Cranelift gives us just a pointer and a buffer of code.

So we have to transmute that `*const u8` to a function pointer!

Super-duper unsafe!

Mitigation: Valgrind

Example: Registration

```rust
use glam::Vec3; // just some random type
use roto::{Runtime, Val, library};

let lib = library! {
    #[copy] type Vec3 = Val<Vec3>;

    impl Val<Vec3> {
        fn x(v: Val<Vec3>) -> f32 {
            v.x
        }
    }
};

let rt = Runtime::from_lib(lib).unwrap();
```

Example: Registration

```
fn add_x_components(a: Vec3, b: Vec3) -> f32 {
    a.x() + b.x()
}
```

Example: Registration

```rust
let mut pkg = rt.compile("script.roto").unwrap();
let f = pkg.get_function("add_x_components").unwrap()

let a = Vec3::new(3.0, 0.0, 0.0);
let b = Vec3::new(5.0, 0.0, 0.0);
let out: f32 = f.call(&mut (), Val(a), Val(b));
assert_eq!(out, 8.0);
```

Registration restrictions

All registered types must implement `Clone`

Everything else should be in `Rc` or `Arc`

Registered types must be wrapped in `Val(..)`

No serialization necessary!

DEMO

Tooling

Tree-sitter grammar for syntax highlighting

Documentation generator for Sphinx

More in the future (...lsp?)

Current limitations

Roto might not fit your needs yet:

- No lists yet
- Therefore, no `for` loops (only `while`)
- No constants defined in scripts
- No custom `enum`
- No generics

We take this thing seriously!

Backed by a non-profit organization.

Integral part of a major product.

Free and open source forever.

Join us for RustWeek 2026!

RustWeek

by RustNL

May 18-23, 2026 – Utrecht, The Netherlands

See rustweek.org

Links

More about Roto

- github.com/NLnetLabs/roto
- roto.docs.nlnetlabs.nl

Find me online

- terts.dev
- terts@nlnetlabs.nl
- @mastodon.online@tertsdiepraam

Feel free to come up and talk to me!

Slides made with Typst.

Slides, recording & links:

https://terts.dev/talks/roto-eurorust25