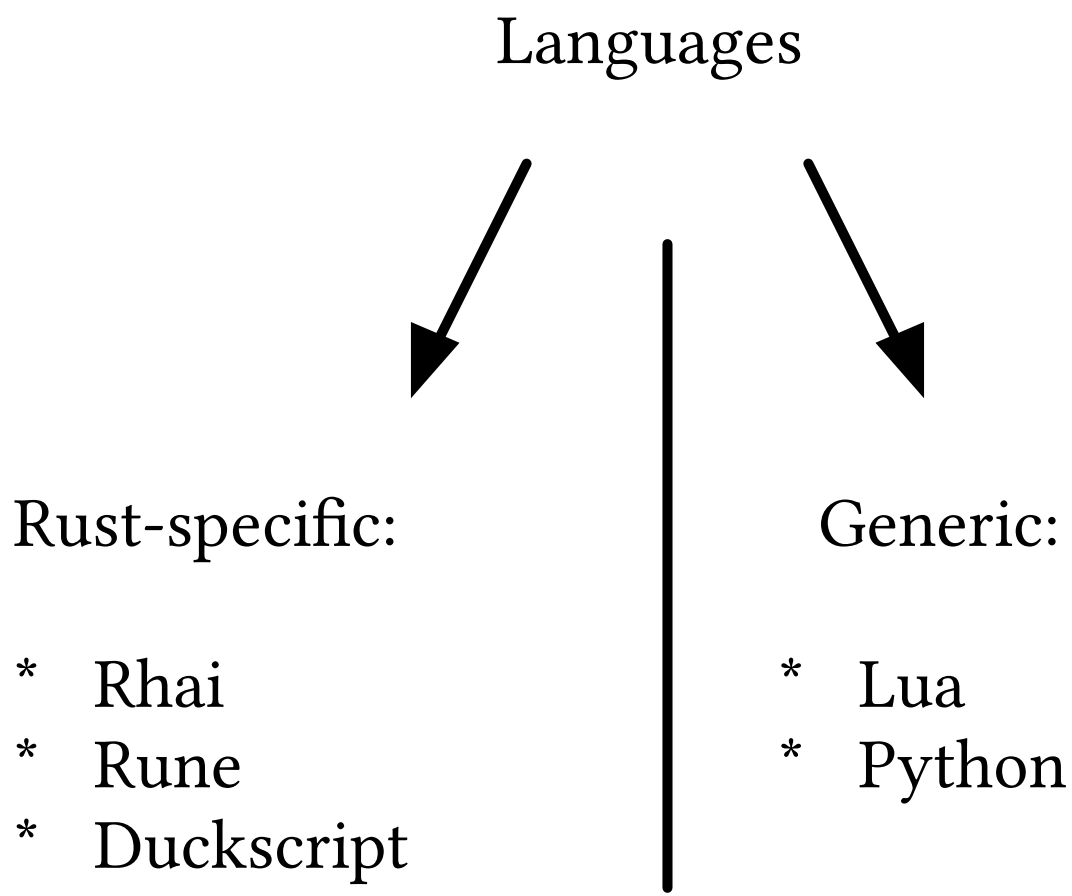
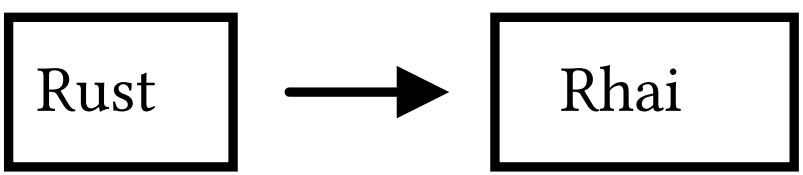
Scripting Rust wit Rhai

Link to this presentation:


Languages
Rust-specific:
* Rhai
* Rune
* Duckscript
Generic:
* Lua
* Python

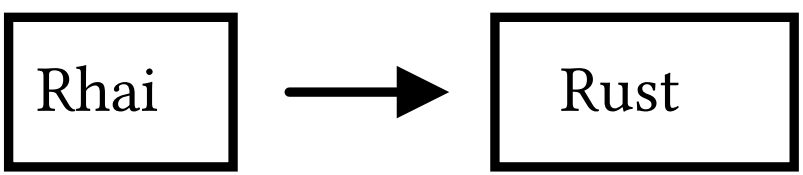
Rhai

This is the option I am familiar with.


Rust
Rhai

```rust
fn main() -> Result<(), Box<dyn std::error::Error + Send + Sync>>
{
    let e = rhai::Engine::new();
    let ast = e.compile("fn qqq(r) { r + r }")?;
    let mut scope = rhai::Scope::new();
    let x : u32 = e.call_fn(&mut scope, &ast, "qqq", (4u32,))?;
    assert_eq!(x, 8);
    Ok(())
}
```

```
$ cargo bloat --crates
 File  .text     Size Crate
 8.9%  73.3%   1.1MiB rhai
 3.1%  25.8% 389.5KiB std
```


Rhai
Rust

```rust
fn doubler(x: i64) -> i64 { x * 2 }
fn printer(x : i64) {  println!("{x}") }

fn main() -> Result<(), Box<dyn std::error::Error + Send + Sync>>{
    let mut e = rhai::Engine::new_raw();
    e.register_fn("doubler", doubler);
    e.register_fn("printer", printer);
    e.run("printer(doubler(doubler(5)))")?;
    Ok(())
}
```

```
$ cargo bloat --crates
 File  .text     Size Crate
 6.4%  51.5% 419.7KiB rhai
 5.8%  47.0% 383.5KiB std
```

```rust
#[derive(Clone)]
struct My(u32);

fn main() -> Result<(), Box<dyn std::error::Error + Send + Sync>>{
    let mut e = rhai::Engine::new_raw();
    e.register_fn("create", ||My(4));
    e.register_fn("print", |x: My|{println!("{}", x.0)});
    e.run("print(create())")?;
    Ok(())
}
```


Rhai
Rust
Rhai

```
use std::sync::Arc;
use rhai::{EvalAltResult, FnPtr, NativeCallContext, AST,
Dynamic};

struct MyContext {
    ast: AST,
}
...
```

```
...
fn twice(ctx: NativeCallContext, f: FnPtr) -> Result<(),
Box<EvalAltResult>> {
    let myctx : Arc<MyContext> =
ctx.tag().unwrap().clone().cast();
    let _ : () = f.call(ctx.engine(), &myctx.ast, ())?;
    let _ : () = f.call(ctx.engine(), &myctx.ast, ())?;
    Ok(())
}
...
```

```rust
fn main() -> Result<(), Box<dyn std::error::Error + Send + Sync>>{
    let mut e = rhai::Engine::new_raw();
    e.register_fn("twice", twice);
    e.register_fn("print", |x:i64|{println!("{x}")});
    let program = "twice(|| print(3)); twice(|| print(5));";
    let ast = e.compile(program)?;
    let myctx = Arc::new(MyContext{ast});
    e.set_default_tag(Dynamic::from(myctx.clone()));
    e.run_ast(&myctx.ast)?;
    Ok(())
}
```

Allows async workaround, though leads to a callback hell.

Before disabling features (min-size build):

```
File  .text     Size Crate
7.8%  73.9% 403.0KiB rhai
2.4%  22.3% 121.8KiB core
```

After disabling features: `no_custom_syntax,no_float,no_index no_module,no_object,no_optimize,no_position,no_time`

```
6.6%  69.0% 190.3KiB rhai
2.2%  23.5%  64.7KiB core
```

```
$ ls -sh ./mini
328K ./mini

$ ./mini
3
3
5
5
```

Still moderately bloated, but has its market niche.

The end