SAFETY FIRST WITH NIFS AND RUSTLER

EXTENDING ERLANG/ELIXIR

- Ports
- C node
- Port Drivers
- Native implement function NIF

	Ease of use	Speed	Safty
Ports	👍	👎	💪
C Node	👎	🖕	💪
Ports Driver	👎	👍	🚧🚨🚧
NIFs	👎	😇	🚧🚨🚧

RUSTLER

https://github.com/hansihe/rustler

RUST LANG

- zero-cost abstractions
- move semantics
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inference
- minimal runtime
- efficient C bindings

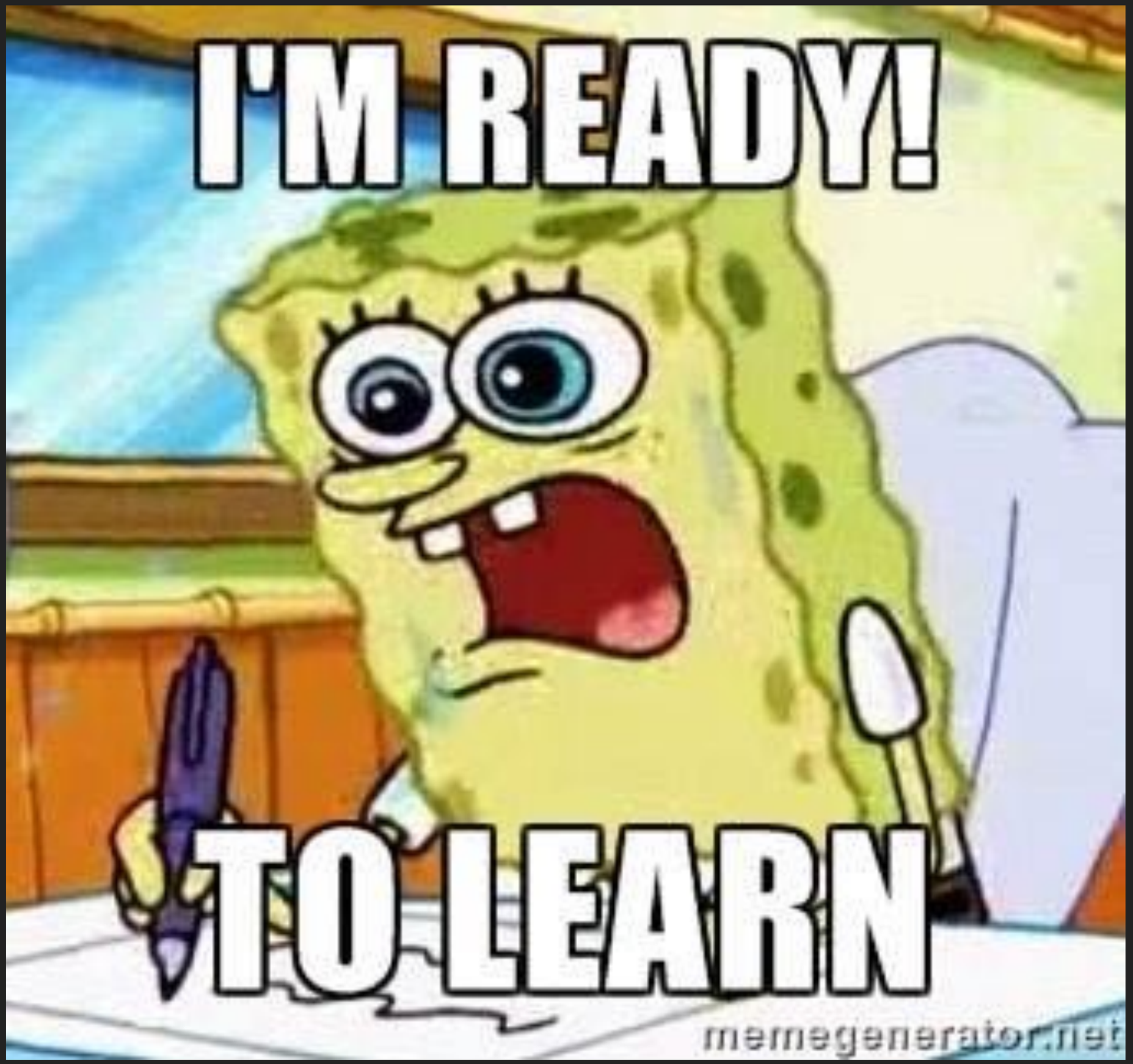
RUST LANG

- zero-cost abstractions
- move semantics

- guaranteed memory safety
- threads without data races

- trait-based generics
- pattern matching
- type inference

- minimal runtime
- efficient C bindings


I'M READY!
TO LEARN
memegenerator.net

ELIXIR BINDINGS TO LIBGIT2

SETUP MIX PROJECT WITH RUSTLER

```
$ mix new gixir

$ git diff mix.exs

     |   defp deps do
     |     [
+    |       {:rustler, "~> 0.16.0"}
     |     ]
     |   end
$ mix deps.get
$ mix rustler.new
```

RUST CRATE STRUCTURE

```
tree native/

native/
└── gixir
    ├── Cargo.lock
    ├── Cargo.toml
    ├── README.md
    └── src
        └── lib.rs


2 directories, 4 files
```

AUTOGENERATED LIB.RS

File: **native/gixir/src/lib.rs**

```rust
#[macro_use] extern crate rustler;
#[macro_use] extern crate rustler_codegen;
#[macro_use] extern crate lazy_static;

use rustler::{NifEnv, NifTerm, NifResult, NifEncoder};

mod atoms {
    rustler_atoms! {
        atom ok;
        //atom error;
        //atom __true__ = "true";
        //atom __false__ = "false";
    }
}

rustler_export_nifs! {
    "Elixir.Gixir",
    [("add", 2, add)],
    None
}

fn add<'a>(env: NifEnv<'a>, args: &[NifTerm<'a>]) -> NifResult<NifTerm<'a>> {
    let num1: i64 = try!(args[0].decode());
    let num2: i64 = try!(args[1].decode());

    Ok((atoms::ok(), num1 + num2).encode(env))
}
```

INITIAL CONFIGURATION

File: mix.exs

```
1      defmodule Gixir.MixProject do
2        use Mix.Project
3
4        def project do
5          [
6            app: :gixir,
7            version: "0.1.0",
8            elixir: "~> 1.6",
9            start_permanent: Mix.env() == :prod,
10 +         compilers: [:rustler] ++ Mix.compilers(),
11 +         rustler_crates: rustler_crates(),
12           deps: deps()
13         ]
14       end
15
16       # Run "mix help compile.app" to learn about applications.
17       def application do
18         [
19           extra_applications: [:logger]
20         ]
21       end
22
23       # Run "mix help deps" to learn about dependencies.
24       defp deps do
25         [
26 +         {:rustler, "~> 0.16.0"}
27           # {:dep_from_hexpm, "~> 0.3.0"},
28           # {:dep_from_git, git: "https://github.com/elixir-lang/my_dep.git", tag: "0.1.0"},
29         ]
30       end
31 +
32 +     defp rustler_crates do
33 +       [
34 +         io: [
35 +           path: "native/gixir",
36 +           mode: if(Mix.env() == :prod, do: :release, else: :debug)
37 +         ]
38 +       ]
39 +     end
40     end
```

OPEN REPOSITORY

```rust
fn repo_open<'a>(env: NifEnv<'a>, args: &[NifTerm<'a>]) -> NifResult<NifTerm<'a>> {
    let path: String = try!(args[0].decode());

    let repo = match Repository::open(path) {
        Ok(repo) => repo,
        Err(e) => return Ok((atoms::error(), (e.raw_code(), e.message().to_string())).encode(env)),
    };
    Ok(atoms::ok().encode(env))
}
```

```
iex(2)> Gixir.repo_open(1)
** (ArgumentError) argument error
    (gixir) Gixir.repo_open(1)
iex(2)> Gixir.repo_open(".")
:ok
iex(3)> Gixir.repo_open("/tmp")
{:error, {-3, "could not find repository from '/tmp'"}}
```

RETURNING REFERENCE TO OBJECT

```rust
use rustler::{NifEncoder, NifEnv, NifResult, NifTerm};
use rustler::resource::ResourceArc;
use git2::Repository;

mod atoms {
    rustler_atoms! {
        atom ok;
        atom error;
        //atom __true__ = "true";
        //atom __false__ = "false";
    }
}

rustler_export_nifs! {
    "Elixir.Gixir",
    [("add", 2, add), ("repo_open", 1, repo_open)],
    Some(on_load)
```

```
iex(1)> Gixir.repo_open(".")
{:ok, #Reference<0.621351194.2609250307.86327>}
iex(2)>
```

```rust
unsafe impl Sync for RepositoryResource {}

fn on_load<'a>(env: NifEnv<'a>, _load_info: NifTerm<'a>) -> bool {
    resource_struct_init!(RepositoryResource, env);
    true
}

fn repo_open<'a>(env: NifEnv<'a>, args: &[NifTerm<'a>]) -> NifResult<NifTerm<'a>> {
    let path: String = try!(args[0].decode());

    let repo = match Repository::open(path) {
        Ok(repo) => ResourceArc::new(RepositoryResource { repo: repo }),
        Err(e) => return Ok((atoms::error(), (e.raw_code(), e.message().to_string())).encode(env)),
    };

    Ok((atoms::ok(), repo).encode(env))
}
```

GETTING BRANCH LISTS

```rust
fn repo_list_branches<'a>(env: NifEnv<'a>, args: &[NifTerm<'a>]) -> NifResult<NifTerm<'a>> {
    let repo_arc: ResourceArc<RepositoryResource> = try!(args[0].decode());
    let repo = &repo_arc.repo;

    let branches = match repo.branches(Some(BranchType::Local)) {
        Ok(branches) => branches,
        Err(e) => return Ok((atoms::error(), e.raw_code()).encode(env)),
```

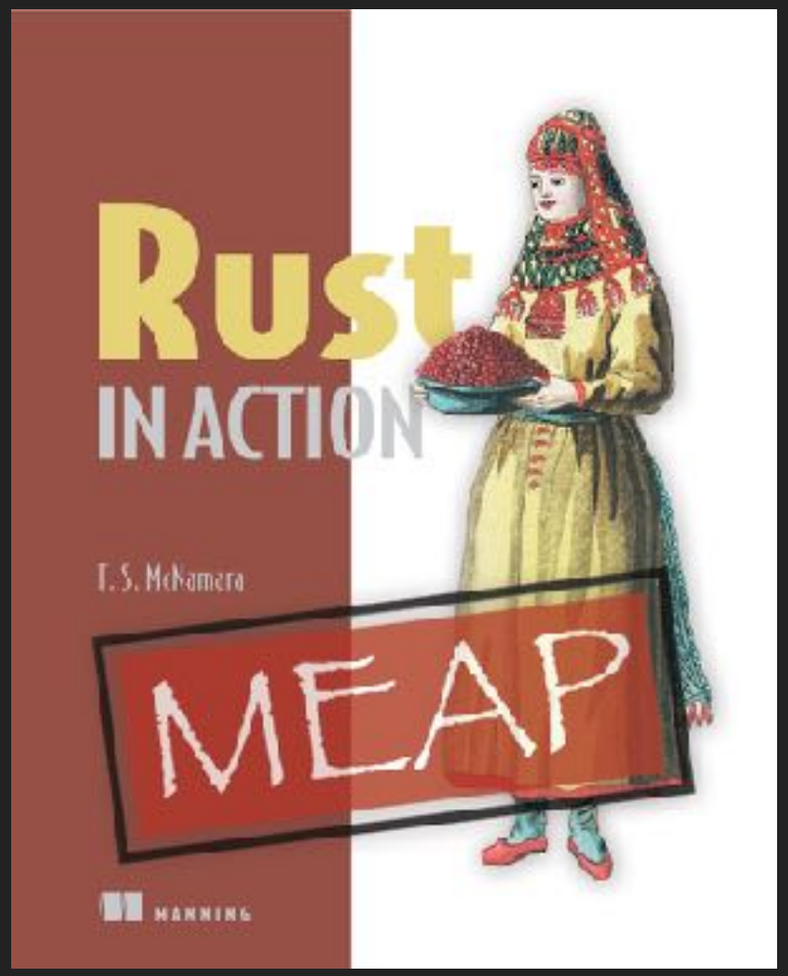
```
iex(1)> {:ok, ref} = Gixir.repo_open(".")
{:ok, #Reference<0.1045791663.3897688067.158734>}
iex(2)> Gixir.
MixProject                add/2                    repo_list_branches/1
repo_open/1
iex(2)> Gixir.repo_list_branches(ref)
{:ok, ["master"]}
iex(3)>
```

```rust
            Err(e) => return Ok((atoms::error(), e.raw_code()).encode(env)),
        };
        let branch_name = match branch_name {
            Some(v) => format!("{}", v),
            None => return Ok((atoms::error(), 2).encode(env)),
        };
        v.push(branch_name)
    }

    Ok((atoms::ok(), v).encode(env))
}
```

The Rust Programming Language

https://github.com/rust-lang/book



Rust in Action

QUESTIONS?