PRODSP TECHNOLOGIES

Welcome at BudLUG/1

Opening + Some Thoughts About Labview and Rust

BudLUG 2024/1, László Balogh, 16th of May, 2024

One of the most Important Thing

Catering

＋ LabVIEW ＝

Sponsored by PRODSP TECHNOLOGIES

Presentations

- László Balogh: Opening + Some Thoughts About Labview and Rustl

  Beer/Coffee Break

- Adrienn Mészáros: Education Services Overview Under New Organization

  Beer/Coffee Break

- Mihály Bánhegyi: OS és nyelv független hálózaton megosztott memória

  Beer/Coffee Break

- Károly Sipos: LabVIEW a hobbielektronikában

  Beer/Coffee + Freeflow Discussion

Next Happening

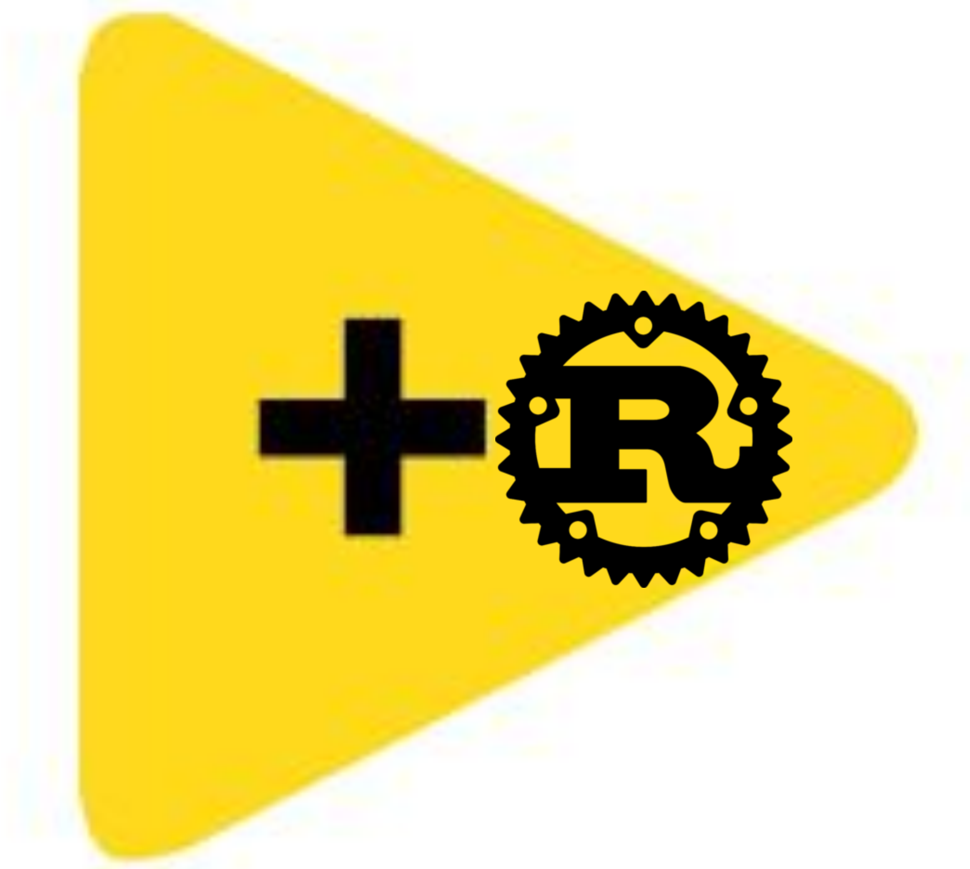
- BudLUG/2 - this autumn
- Where?

Candidate places:

- Knorr-Bremse
- Bosch?

Any presentation is welcome!

Labview and Rust



Background

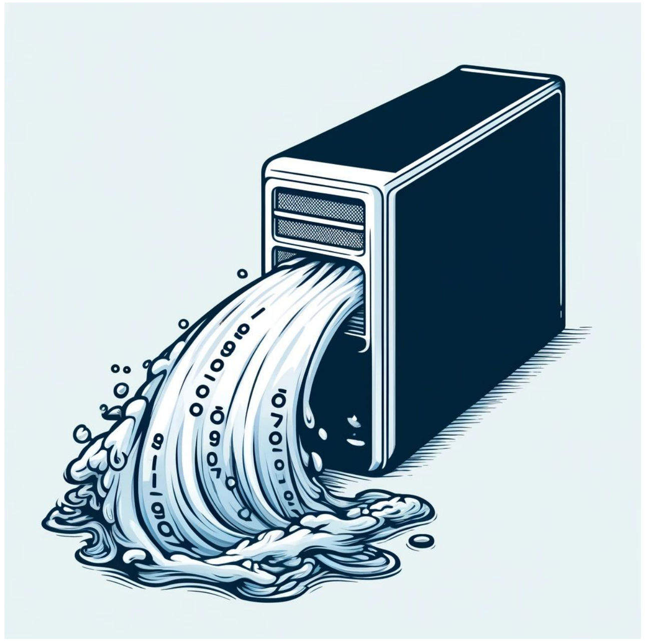
- Tricky memory leak in a LV

⇩

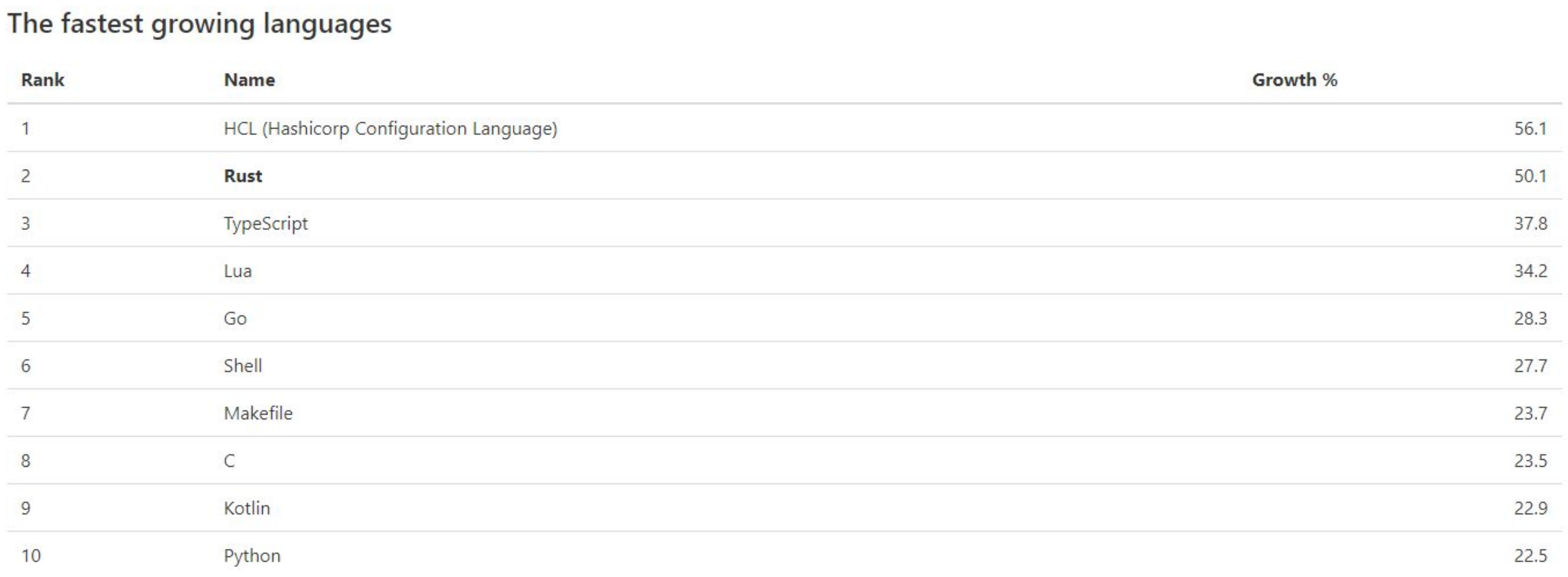
- How to avoid?

⇩

- Let’s try Rust!



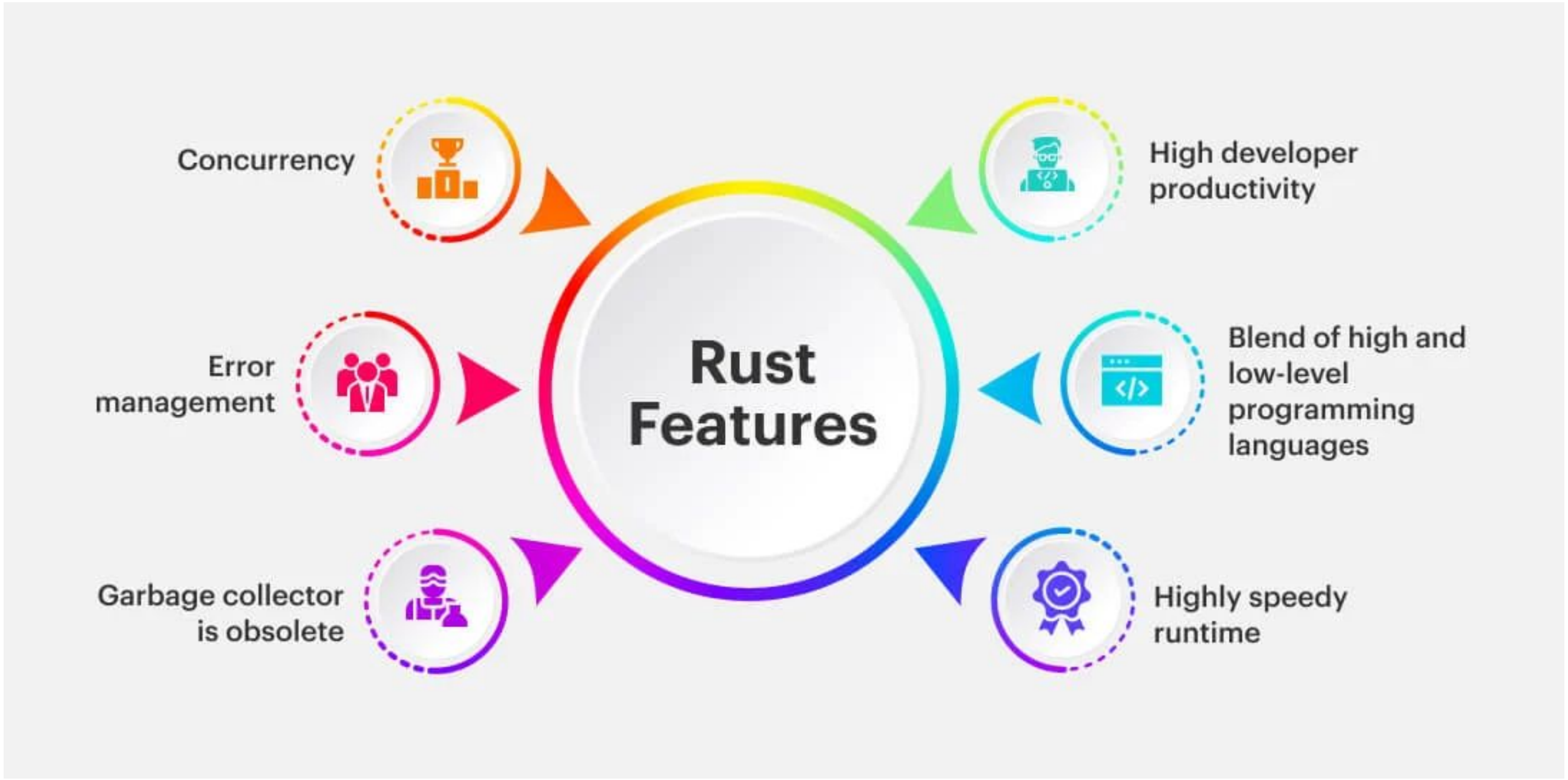
Rust?????



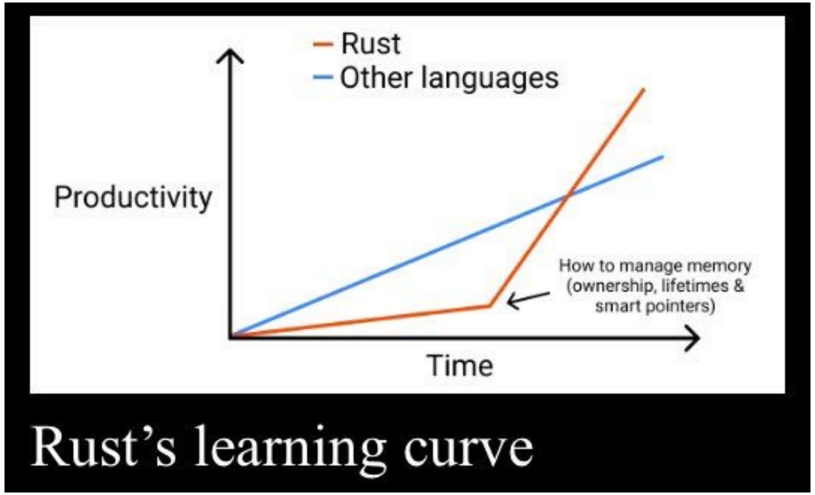
The fastest growing languages

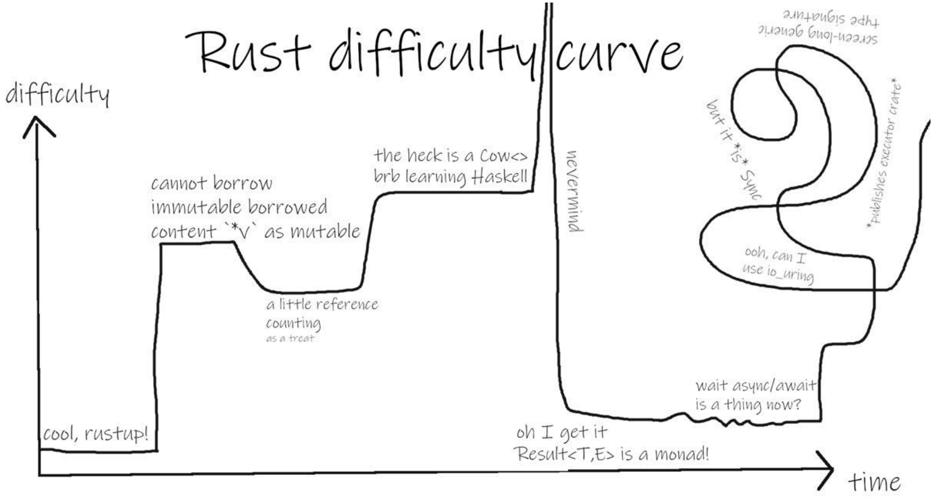
Rank	Name	Growth %
1	HCL (Hashicorp Configuration Language)	56.1
2	**Rust**	50.1
3	TypeScript	37.8
4	Lua	34.2
5	Go	28.3
6	Shell	27.7
7	Makefile	23.7
8	C	23.5
9	Kotlin	22.9
10	Python	22.5

Features


Concurrency
Error management
Garbage collector is obsolete
Rust Features
High developer productivity
Blend of high and low-level programming languages
Highly speedy runtime

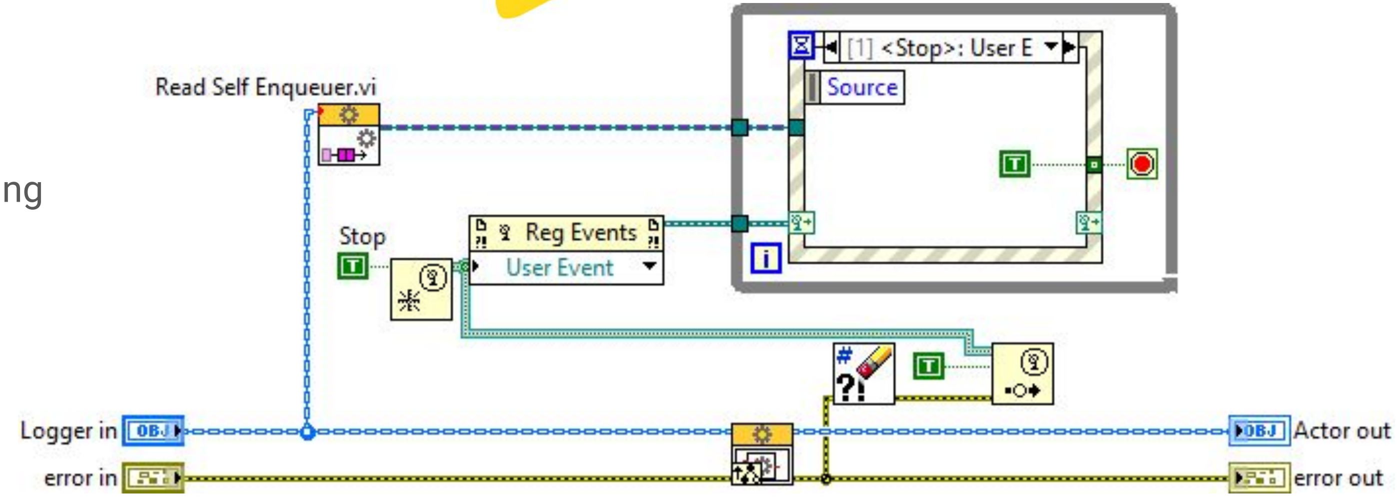
Learning Curve





Comparison Concept

- Memory management
- Case Structure, Matching
- Error Handling
- OOP
- Parallel Programming



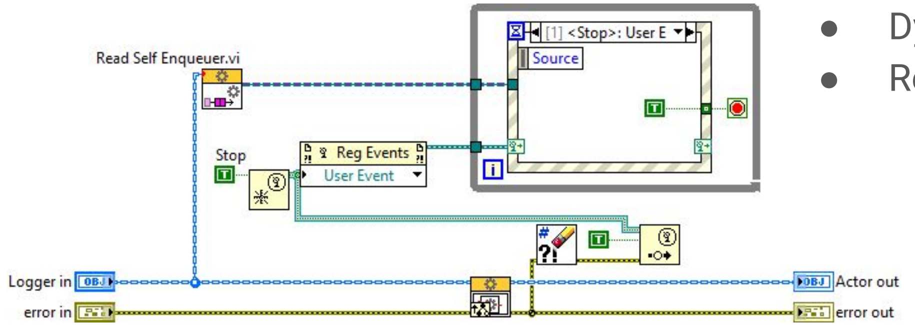
Memory Management

```
let v = vec![1, 2, 3];

let v2 = v;

println!("v[0] is: {}", v[0]);
```

```
let v = vec![1, 2, 3];

let mut v2 = v;
```

- Borrow checker
- Ownership
- Stack, heap usage (dynamic memory)
- Zero overhead
- Unsafe code

- Garbage collection
- Supereasy to use
- No allocation, just definition by wire
- In place code
- Dynamic memory: array
- Reference (dvr)



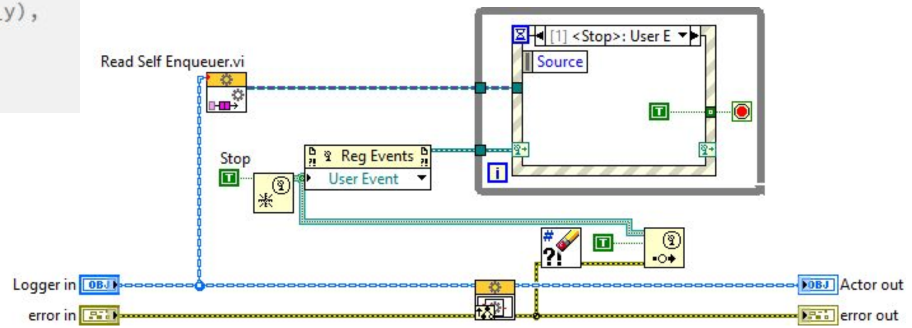
Case Structure, Matching

- Complex and improved enums
- Match structure

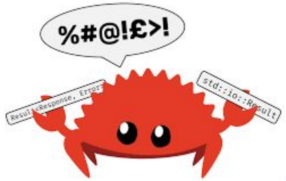
```
enum Message {
    Quit,
    ChangeColor(i32, i32, i32),
    Move { x: i32, y: i32 },
    Write(String),
}
```

```
fn process_message(msg: Message) {
    match msg {
        Message::Quit => quit(),
        Message::ChangeColor(r, g, b) => change_color(r, g, b),
        Message::Move { x, y: new_name_for_y } => move_cursor(x, new_name_for_y),
        Message::Write(s) => println!("{}", s),
    };
}
```

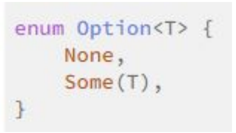
- Well known case structure
- Classic enum
- Variant for general data
- Message class and dynamic dispatch



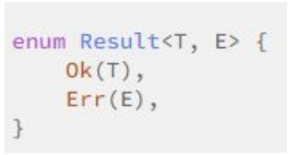
Error Handling



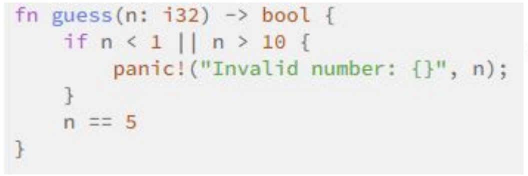
- Unrecoverable error: panic
- Rust: Failure is not an option<T>, it is a Result<T,E>
- unwrap, unwrap_and_result



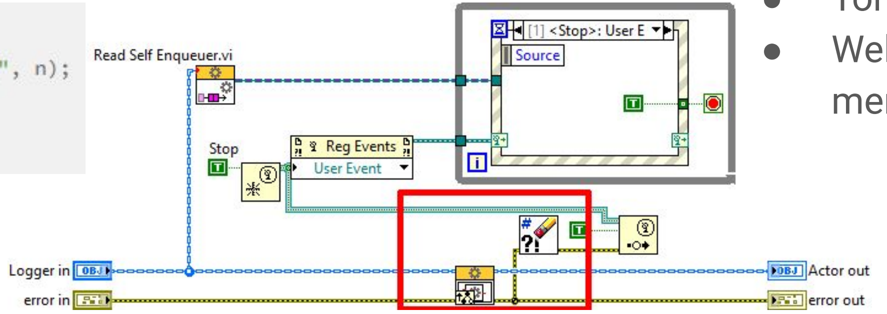
```
enum Option<T> {
    None,
    Some(T),
}
```



```
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```



```
fn guess(n: i32) -> bool {
    if n < 1 || n > 10 {
        panic!("Invalid number: {}", n);
    }
    n == 5
}
```



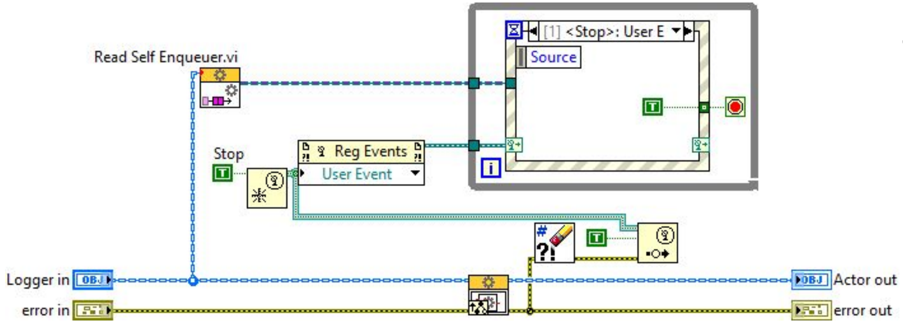
- Error wire
- Automatic error handling
- Not execute if error at the input
- Tons of error codes
- Well defined functions (clear, merge, ….)

Object Oriented Programming

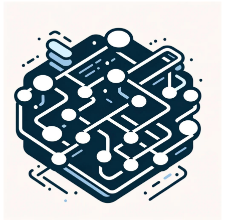
```rust
struct Circle {
    x: f64,
    y: f64,
    radius: f64,
}

trait HasArea {
    fn area(&self) -> f64;
}

impl HasArea for Circle {
    fn area(&self) -> f64 {
        std::f64::consts::PI * (self.radius * self.radius)
    }
}
```

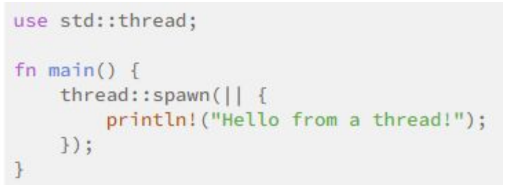
- encapsulation
  - methods for enum/struct
  - public keyword
- NO inheritance
  - default trait
- polymorphism
  - trait bounds
  - trait objects

- classes, objects, interfaces
- inheritance
  - dynamic dispatch
- Design patterns

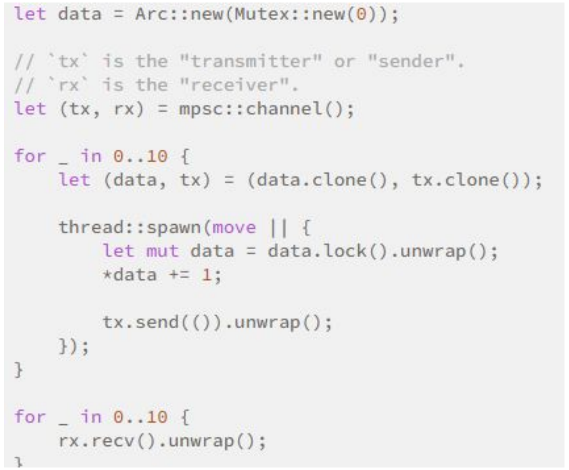


Parallelism





```
use std::thread;

fn main() {
    thread::spawn(|| {
        println!("Hello from a thread!");
    });
}
```



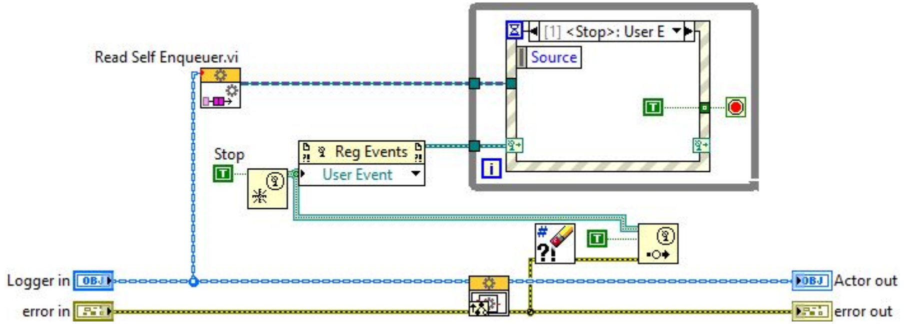
```
let data = Arc::new(Mutex::new(0));

// `tx` is the "transmitter" or "sender".
// `rx` is the "receiver".
let (tx, rx) = mpsc::channel();

for _ in 0..10 {
    let (data, tx) = (data.clone(), tx.clone());

    thread::spawn(move || {
        let mut data = data.lock().unwrap();
        *data += 1;

        tx.send(()).unwrap();
    });
}

for _ in 0..10 {
    rx.recv().unwrap();
}
```

Fearless Concurrency

- Mutex
- mspc::channel
  - N tx, 1 rx
- Low level

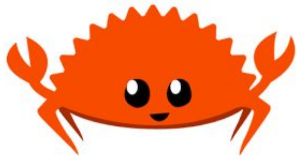
- FGV
- Queue
- Event
- High level solutions
  - passed by value

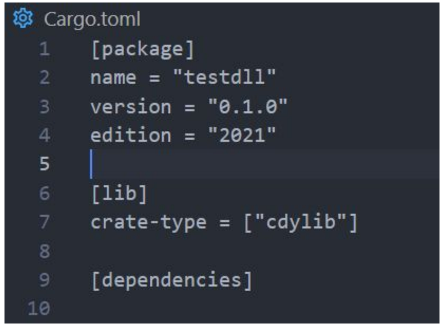
Almost like QMH



Call Rust Code from LabVIEW

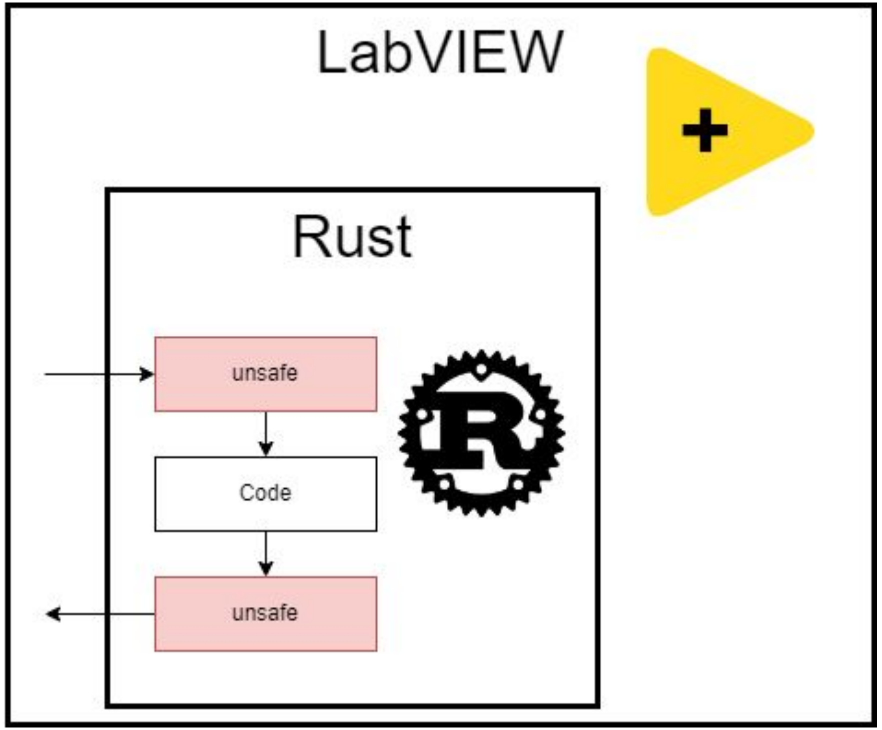
- Create C like dll
- Very Simple
- Needs to use *unsafe* for complex in, out data (for example arrays)





Cargo.toml

```toml
[package]
name = "testdll"
version = "0.1.0"
edition = "2021"

[lib]
crate-type = ["cdylib"]

[dependencies]

```



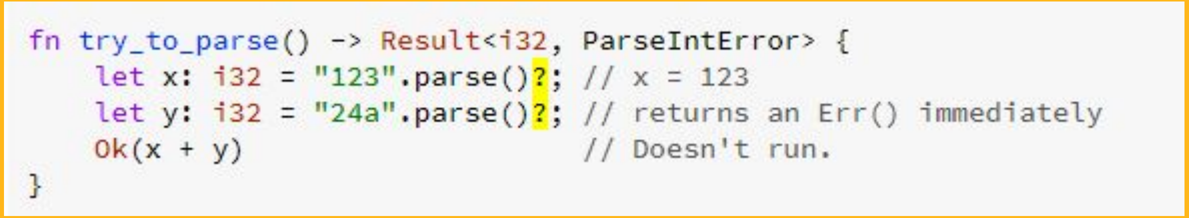
Summary

- More time to get used to it
- Naming: pain to learn
- Built in tons of experience
  - Not so far from Labview patterns
- Fun
- Easy to interface to Labview

Do not stop here

- Package manager
- Patterns and matching
- Closures
- Struct

PRODSP TECHNOLOGIES



```
fn try_to_parse() -> Result<i32, ParseIntError> {
    let x: i32 = "123".parse()?; // x = 123
    let y: i32 = "24a".parse()?; // returns an Err() immediately
    Ok(x + y)                    // Doesn't run.
}
```