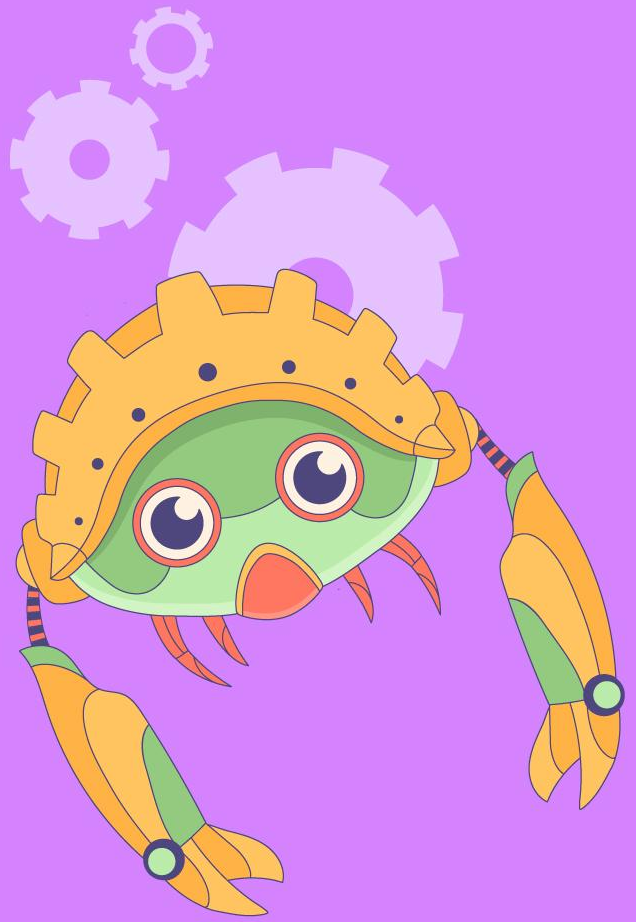
Dylan Brown

Senior Engineering Manager @ Rising Academies

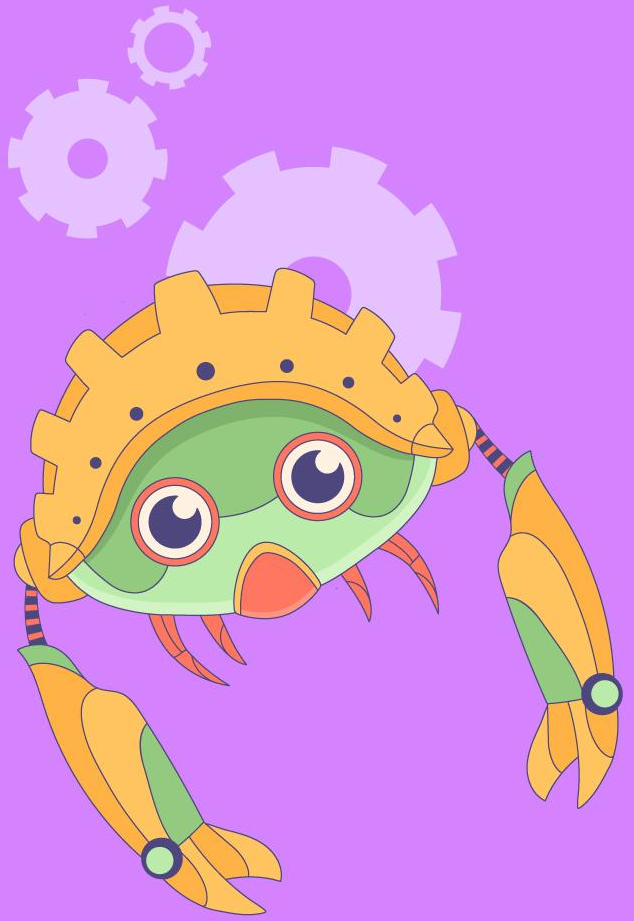
Lessons from African ed-tech and the case for Rust beyond systems programming



Dylan Brown

Senior Engineering Manager @ Rising Academies

Lessons about Rust from education with Rust



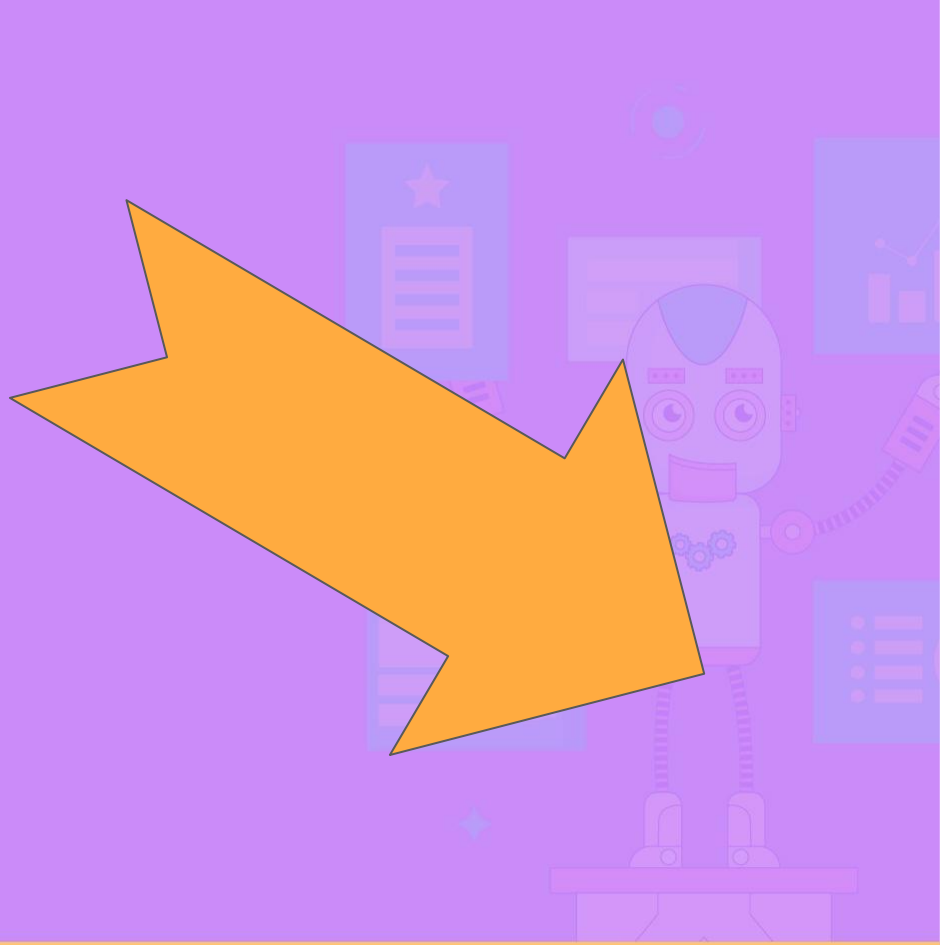
Education in Africa

RISING ACADEMIES

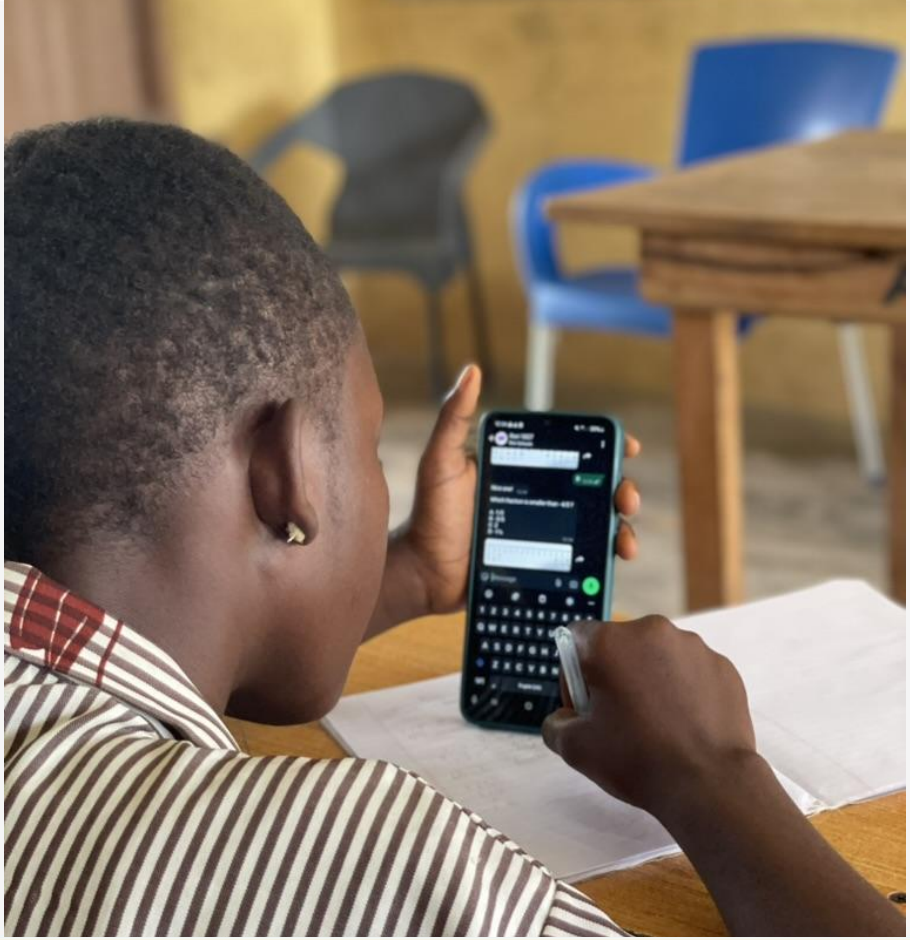


Me

~3 years with Rust
10+ years as a dev
8+ years working with chatbots
~5 years leading engineering teams
3 cats, 1 wife, lots of hyperfocuses, some hobbies
Live in Cape Town, South Africa



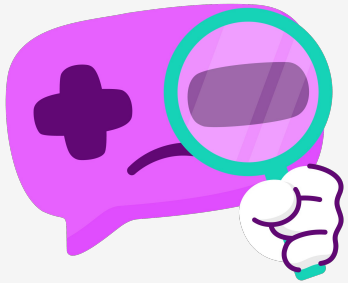

Rori



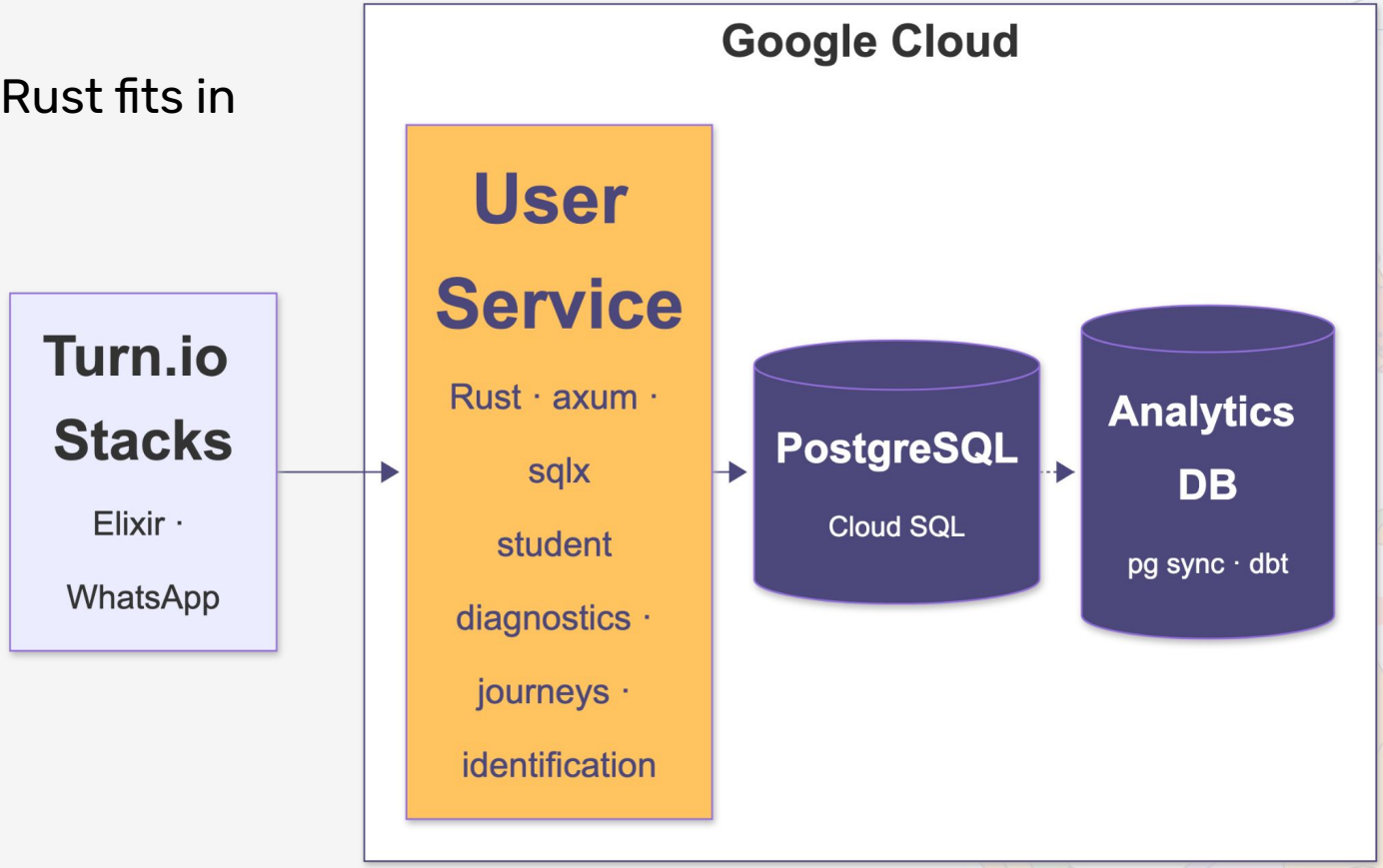


- Not just an LLM driven tutor
- Content crafted by experts
- Custom built answers classification, student identification and content management services

Where Rust fits in



Where Rust fits in



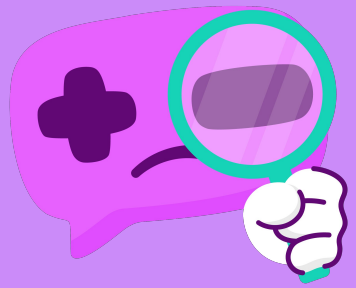
Why Rust?

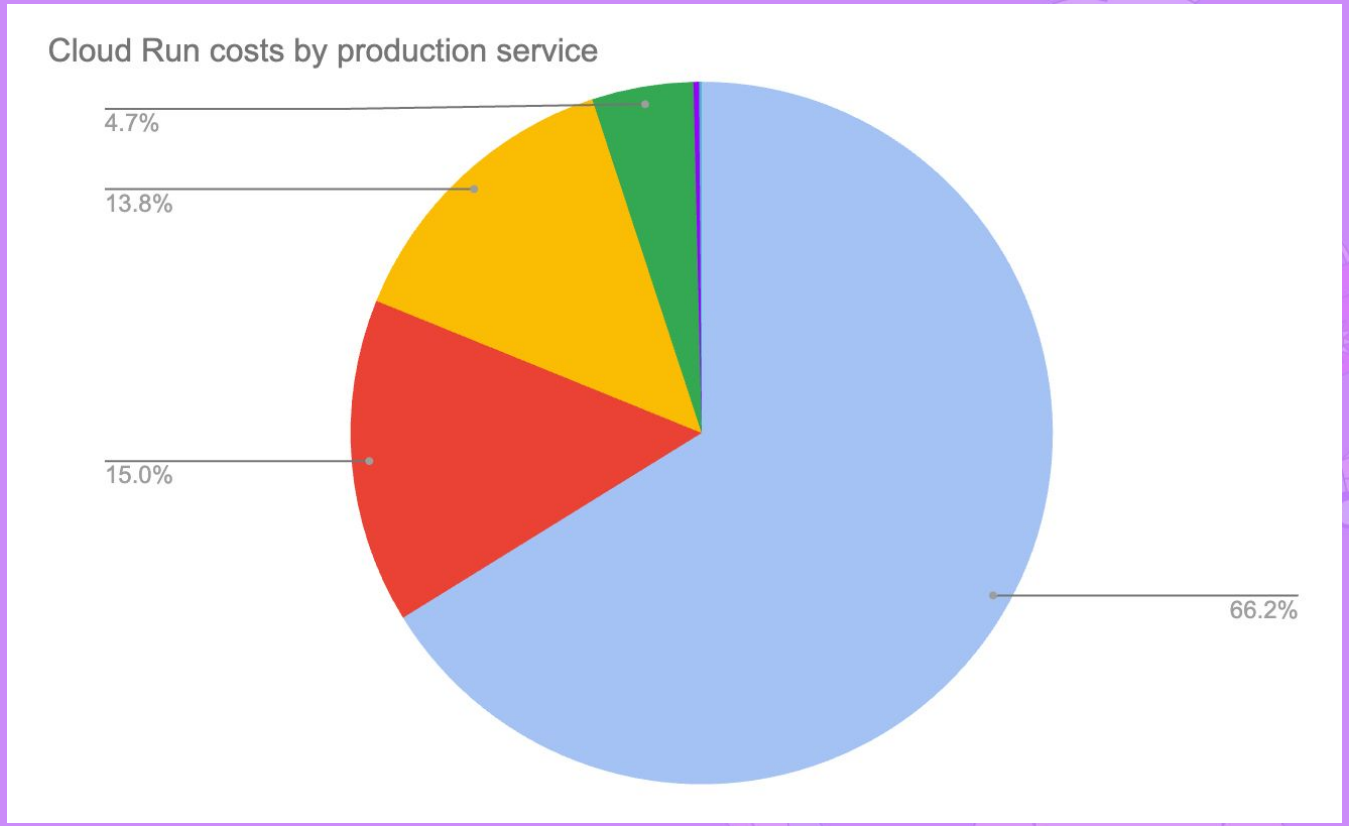
Education is a long game



Education is a long game

Why Rust?





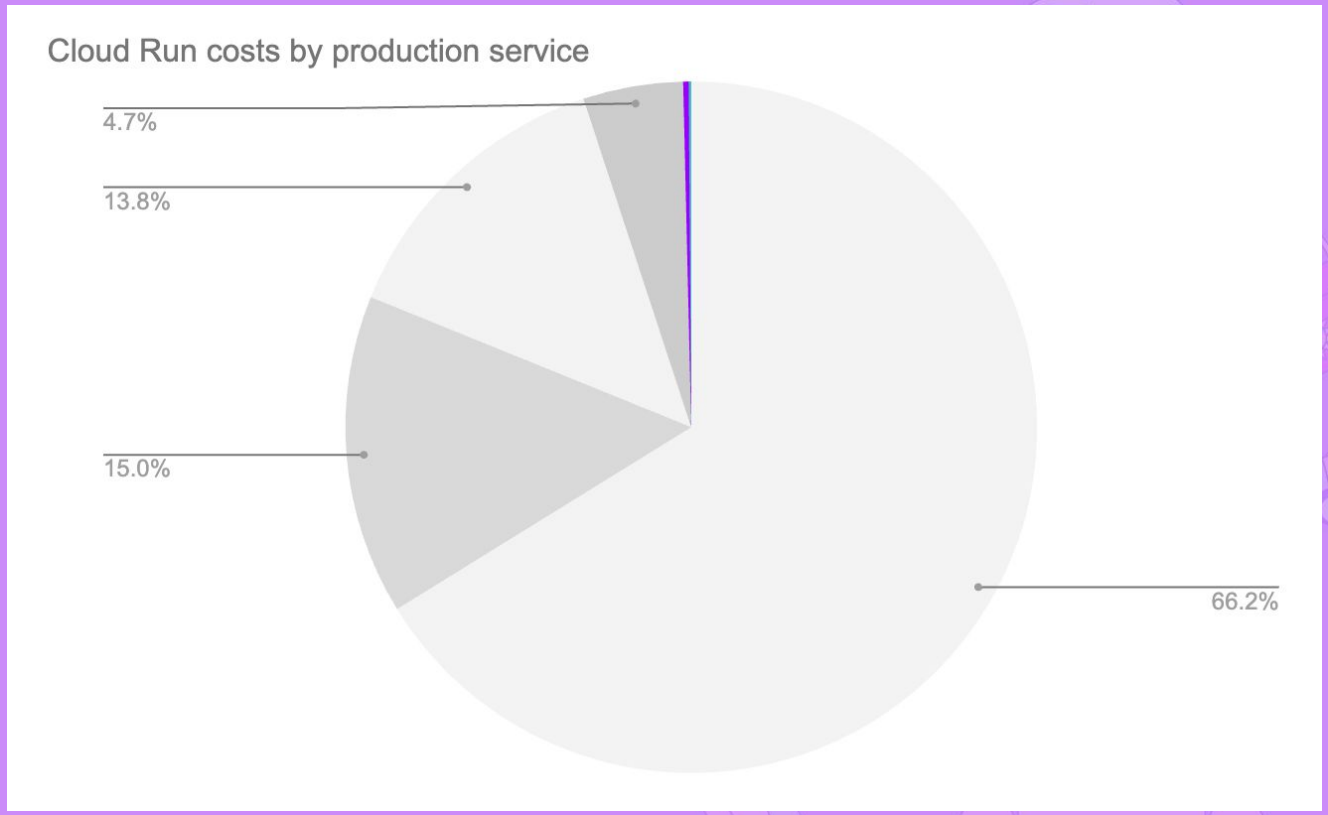
Education is a long game

Why Rust?

1:1.47
Requests/s

1:0.017
$ cost





Education is a long game

Why Rust?

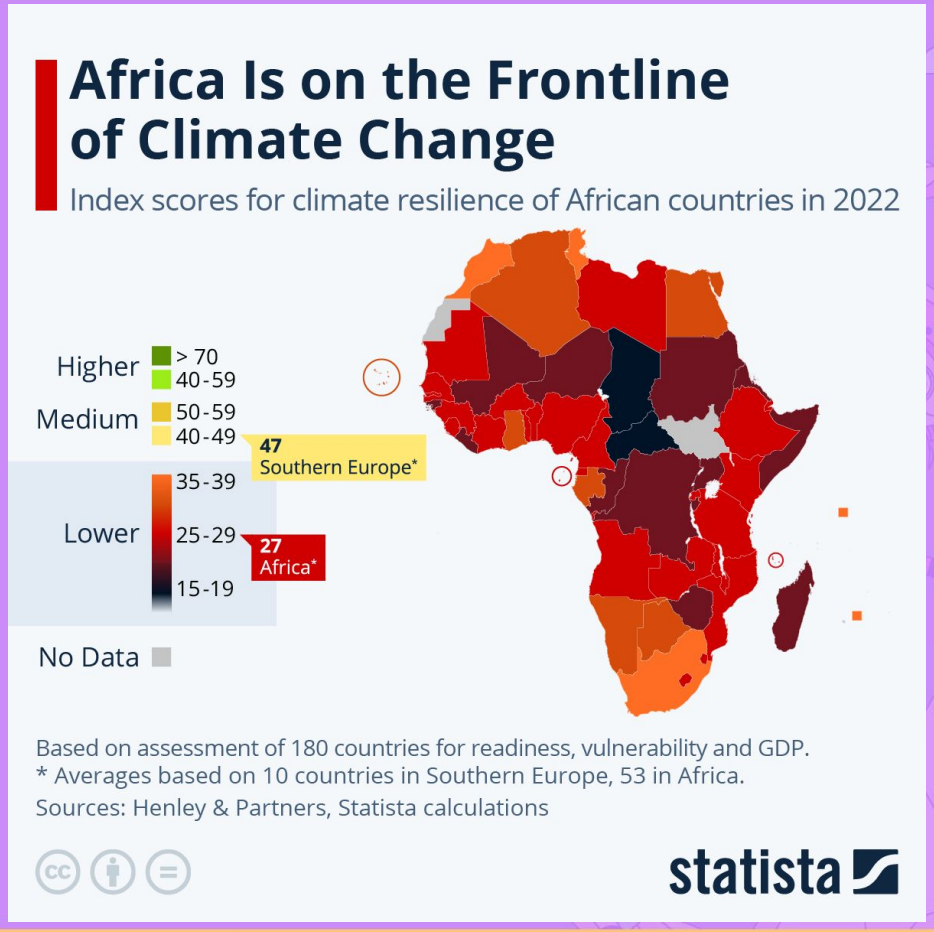
Maintenance and Stability



Education is a long game

Why Rust?

“Africa is responsible for less than 10 percent of global greenhouse gas emissions. But it is the continent which is the least able to cope with the negative impacts of climate change.” - WMO Secretary-General Prof. Petteri Taalas.
[source]



Why Rust?

Shoutout to Laabidi Raissi from RustLab 2023
[source]



Learnings

What we learnt along the way



Rust Mindset



- How you approach a problem
- Identify in other languages too

Rust Mindset



- Change is inevitable
- Unexpected s*** happens
- The compiler is your friend

Change is inevitable



- Aware of *how* things will change
- Make it *safe and easy* to embrace change

Change is inevitable

```rust
// The signal back to the client for the next activity
enum NextActivitySignal {
    Continue,
    SkipUp,
    SkipDown,
}
// The confirmation from the client for a skip
enum SkipConfirmation {
    SkipUp,
    SkipDown,
}
```

Change is inevitable

```
// The signal back to the client for the next activity
enum NextActivitySignal {
    Continue,
    SkipUp,
    SkipDown,
}
// The confirmation from the client for a skip
enum SkipConfirmation {
    SkipUp,
    SkipDown,
}
```

Change is inevitable

```
// The signal back to the client for the next activity
enum NextActivitySignal {
    Continue,
    Skip(SkipDirection),
}
// A shared enum that gives us tighter coupling for both directions
enum SkipDirection {
    SkipUp,
    SkipDown,
}
```

Unexpected s*** happens



- Hubris to believe we can anticipate everything
- How can we prepare for the unexpected?

Unexpected s*** happens

```python
# let's look a python example
def get_user(user_id):
  if not is_human(user_id):
    return None # We only expect to deal with humans right?
  return users.get(user_id)
# The issue is passed on to the call site
def greet(user_id):
    user = get_user(user_id)
    return f"Hello, {user.name}"  # AttributeError
```

Unexpected s*** happens

```rust
fn get_user(user_id: UserId) -> Option<User> {
    if !is_human(&user_id) {
        return None;
    }
    Some(load_user(user_id))
}

fn greet(user_id: UserId) -> String {
    let user = get_user(user_id);
    // ERROR: no field `name` on `Option<User>`
    format!("Hello, {}", user.name)
}
```

The compiler is your friend



- Work with the compiler not against it
- Use the compiler as a reflection of your design

The compiler is your friend

```rust
fn get_user(user_id: UserId) -> Option<User> {
    if !is_human(&user_id) {
        return None;
    }
    Some(load_user(user_id))
}

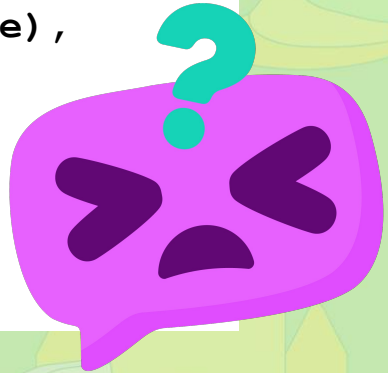
fn greet(user_id: UserId) -> String {
    let user = get_user(user_id);
    // ERROR: no field `name` on `Option<User>`
    format!("Hello, {}", user.name)
}
```

The compiler is your friend

```rust
fn greet(user_id: UserId) -> String {
    // We could resolve the compiler error with an unwrap
    let user = get_user(user_id).unwrap();
    format!("Hello, {}", user.name)
}
```

The compiler is your friend

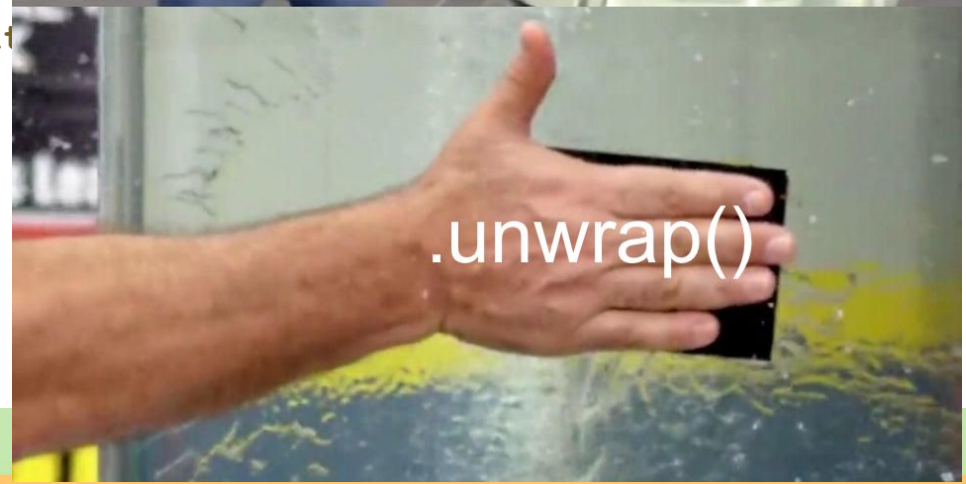
```rust
fn greet(user_id: UserId) -> String {
    let user_option = get_user(user_id);
    // Expanded match handling. No error now
    match(user_option) {
        Some(user) => format!("Hello, {}", user.name),
        None => "Be gone monster!".to_string(),
    }
}
```



- The compiler is

```
fn greet(u
    // We                                    n unwrap
    let us
    format
}
```





Toothbrushes



Make the healthy habit easy

Make the Rust mindset easy to follow

- Strong types
- Use Options and Results wisely
- More compiler, less human reliance



Make the healthy habit easy

Make the Rust mindset easy to follow

- Strong types -> *Change is inevitable*
- Use Options and Results wisely -> *Unexpected s*** happens*
- More compiler, less human reliance -> *The compiler is your friend*

Challenges

Learning doesn’t happen without challenges



Challenges

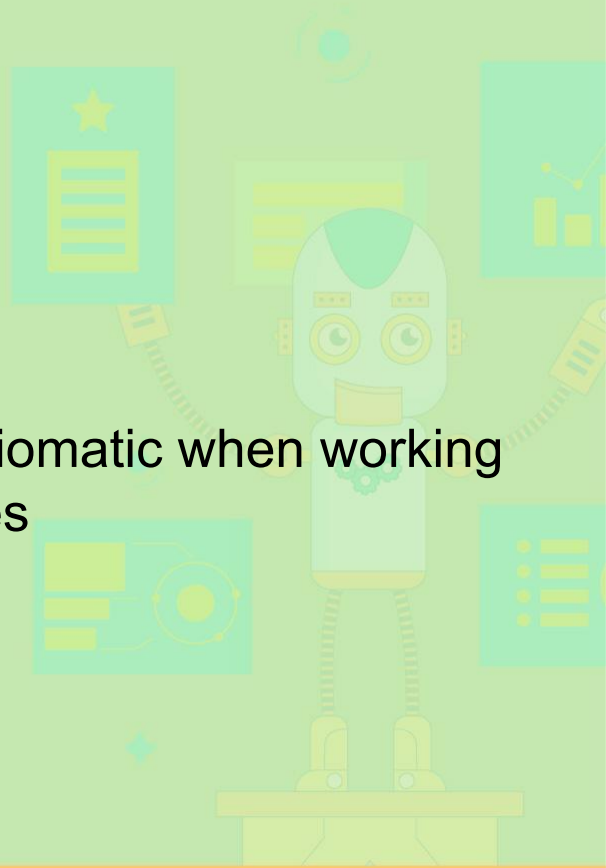
Bringing Rust into a small team comes with challenges



Challenges

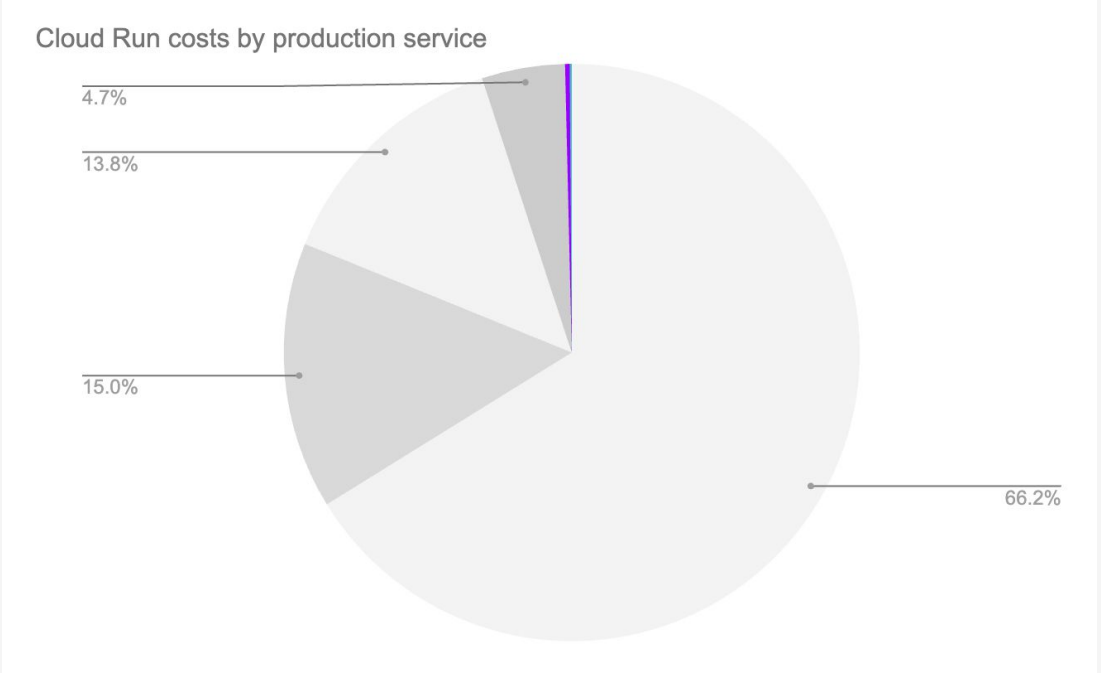
Bringing Rust into a small team comes with challenges

- Learning
- Keeping Rust idiomatic when working at the boundaries



Make learning easier

It’s a purposeful choice you need to make



Make learning easier

It’s a purposeful choice you need to make

- Rust doesn’t have to be difficult to read
- Work with the compiler
- Make changes safer

Messy input

Necessary flexibility

- “Kids say the darndest things”
- Don’t want to throw everything to an LLM
- Don’t want to lose the gains of Rust because we need to have flexible solutions

Messy Input

```rust
use partial_date::extract::extract;
use partial_date::models::{Config, Input};

let config = Some(Config::default());

extract(Input { utterance: "19 Oct 2014".into(), config: config.clone() });
// day: Found(19)  month: Found(October)  year: Found(2014)

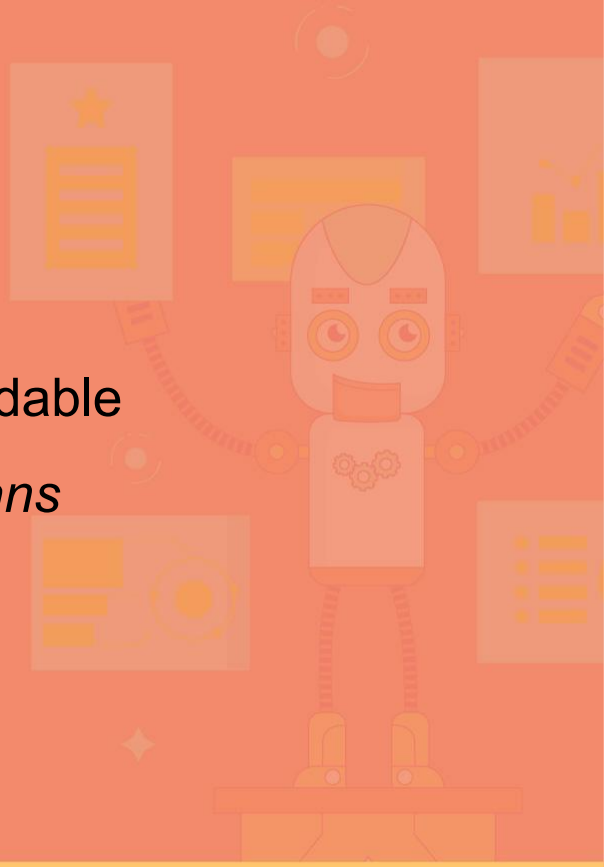
extract(Input { utterance: "Jun 2024".into(), config: config.clone() });
// day: NotFound   month: Found(June)     year: Found(2024)

extract(Input { utterance: "the 15th".into(), config: config.clone() });
// day: Found(15)  month: NotFound        year: NotFound
```

https://github.com/DylanIvanBrown/partial-date

Coding tools (LLMs)

Port everything to Rust!



- Rust *can* be readable
- Rust is for *humans*

Learnings

Coding tools (LLMs)

Compiler trap



Coding tools (LLMs)

Code is the most solid foundation



Coding tools (LLMs)

Don’t become a reverse centaur



Coding tools (LLMs)

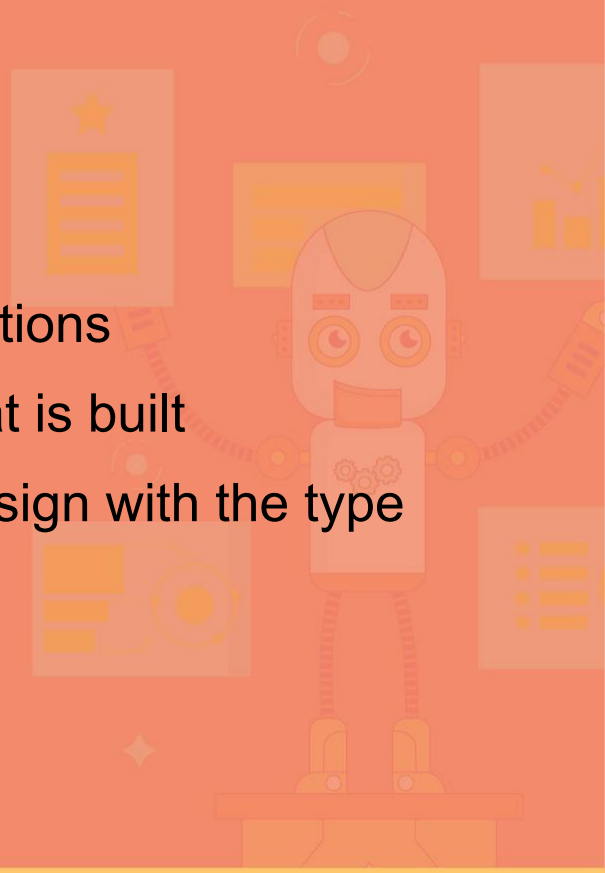
Don’t end up with a weird bathroom



Coding tools (LLMs)

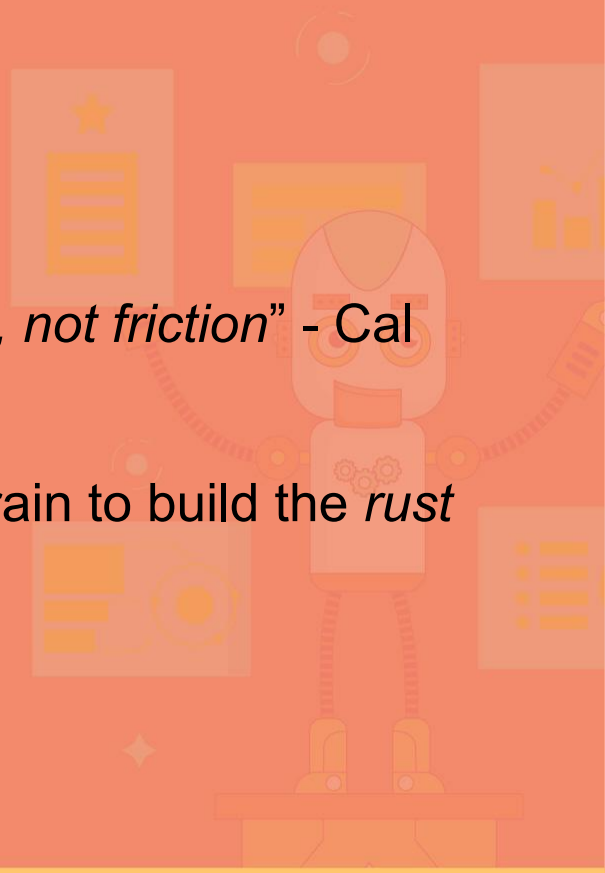
Human code as a foundation

- Craft the foundations
- Understand what is built
- Enforce your design with the type system



Coding tools (LLMs)

- “*Reduce tedium, not friction*” - Cal Newport
- Exercise your brain to build the *rust mindset*



Buy-in

Don’t talk tech

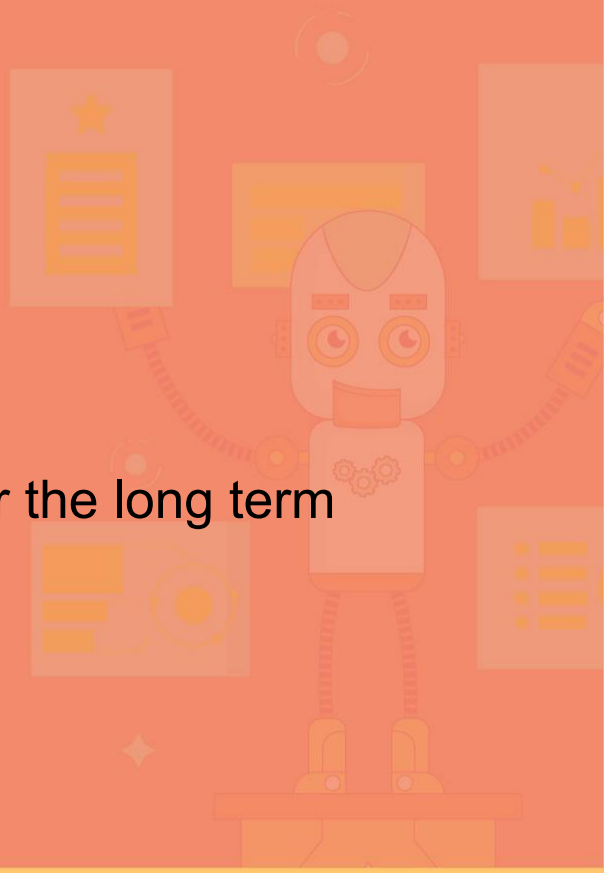




Buy-in

Focus on

- Costs
- Stability
- Productivity over the long term



What is next?

What we built is *not special*. In this era of increasing power consumption for compute, Rust offers us an alternative approach to make technology *more efficient and lower the environmental impact* of what we build.

This is why we should be thinking about Rust's application *beyond systems programming*.

Thank you

Dylan Brown

Blog: https://dylanivanbrown.github.io/
Github: https://github.com/DylanIvanBrown
LinkedIn:
https://www.linkedin.com/in/dylanivanbrown/
Email: dylan.brown@risingacademies.com

