The Rust Programming Language

Origin and History

Florian Gilcher (Ferrous Systems)

Why it's so important for Arm today

Robin Randhawa (Arm)

Linaro TECH DAYS

About Florian Gilcher

- Rust developer since 2013, Trainer since 2015
- Founder of the Berlin users group and RustFest and Oxidize conferences
- Part of Rust community team and Rust core
- Co-Founder of Ferrous Systems, a group of Rust developers providing
  - Help in adopting Rust
  - Training
  - Development
  - Maintenance and work on the Rust compiler

ferrous systems

Agenda

- What is Rust?
- A short run through History
- Rust and embedded
- Adoption strategies
- Hallmark projects
- Rust today
- Summary

ferrous systems

The Rust Programming Language

Rust is

- A language in the range between C and C++
- With static memory safety
- With features making it competitive with a wide range of languages
- Industry-friendly and industry-proven
- Rust aims to eliminate *classes of bugs*

ferrous systems

Some code

```
use std::io;
use std::io::prelude::*;
use std::fs::File;

fn main() -> Result<(), io::Error> {
    let open_file = File::open("test");

    let mut file = match open_file {
        Ok(file) => file,
        Err(e) => return Err(e),
    };

    let mut buffer = String::new();
    file.read_to_string(&mut buffer)?;
    println!("{}", buffer);

    Ok(())
}
```

ferrous systems

Rust Values

- Rust is a language for industrial use
  - It is not a research platform
  - Support programming in the large
- Rust is use-case driven
  - Features added to the language need strong claims
- Strong backwards compatibility guarantees
  - With strong QA
  - “Stability without Stagnation”
- Paced iteration
  - Ability to bring fixes to users safely and fast
- Documentation culture
  - Fully documented standard and core libraries
  - Multiple books!
- Approachability
  - Both for individuals and companies

ferrous systems

Rusts core innovations

- Bringing a ***linear type system*** into mainstream use
- Bringing ***region based memory management*** into mainstream use
  - Predictable performance and memory collection
- Establishing usage of ***algebraic data types*** in systems programming
- Establishing a systems programming language with the toolset of modern high level languages
- Establishing a systems programming language that ***provides more information***
  - E.g. in Rust, the aliasing situation of every reference is known.

ferrous systems

Ownership

```
fn main() {
    let mut vec: Vec<u8> = vec![1,2,3,4];

    for i in &mut vec {
        *i += 1;
    }

    println!("{:?}", vec); // [2, 3, 4, 5]
    // vec is deallocated when falling out of scope
}
```

Data introduced into the program is owned *once*.

ferrous systems

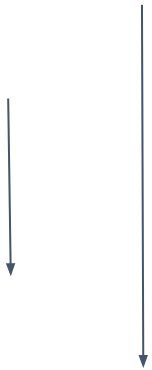
Borrowing

```
fn main() {
    let mut data: Vec<u8> = vec![1,2,3,4];

    let first_two = &mut data[0..2];

    for i in first_two {
        *i += 1;
    }

    println!("{:?}", vec); // [2, 3, 3, 4]
}
```



ferrous systems

Memory Safety

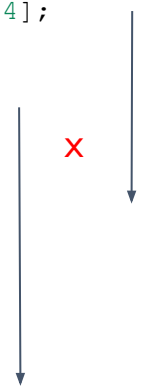
```
fn main() {
    let mut data: Vec<u8> = vec![1,2,3,4];

    let first_two = &mut data[0..2];

    drop(vec); // << deallocation

    for i in first_two {
        *i += 1;
    }

    println!("{:?}", vec); //???
}
```




```
error[E0505]: cannot move out of `vec` because it is borrowed
 --> src/main.rs:6:10
  |
4 |     let first_two = &mut vec[0..2];
  |                     --- borrow of `vec` occurs here
5 |
6 |     drop(vec);
  |          ^^^ move out of `vec` occurs here
7 |
8 |     for i in first_two {
  |              --------- borrow later used here
```

ferrous systems

Rusts memory management rules

- Every value is owned exactly once
  - Either by a function or by other values
- Ownership can be passed on or given up
  - Giving up ownership removes the data from memory
- The owner can hand out references to the value
  - Immutable references
    - As often as liked
  - Mutable references
    - Only exactly once
  - Immutable and mutable references are mutually exclusive

ferrous systems

Rust eliminates shared mutable access!

Common misconceptions

“Borrow checking only applies to heap”

Borrow checking checks all references between data. Indeed, Rust as a language is not aware of the heap.

“Ownership only represents memory management”

Ownership models ownership of resources, which can be:

- Memory
- Kernel handles
- Devices…

Concurrency safety: eliminating shared mutability

```
fn write_to_buffer(buffer: &mut [u8], data: &[u8]) {
  //...
}
```

In this example, *buffer* and *data* can **never** be the same memory location.

Safe & Unsafe Rust

Safe Rust has no:

- Undefined behaviour
- Illegal memory access
- Null pointers
- Data races

Rust has an unsafe sublanguage that allows unsafe things, mainly:

- Raw pointer dereferences
- Calling into foreign code
- Calling known unsafe functions (e.g. raw device access)

“Safe Code means you can take better risks”

- Emily Dunham

Orientation around C/C++

Rust lands *between* C and C++:

- It is a *function* and *data structures* language like C
  - It does not have classes and other higher-level constructs
  - All higher level features still boil down to a function in the end
- It is a generic language like C++
  - You can easily write code generic over multiple data types
- Error management works through *values* like in C
  - With some additions through the type system
- Rusts memory management is very similar to RAII practices from C++
  - Just compile-time checked and validated

We're standing in the shoulders of giants here.

Rust History: Overview

- Rust started 2006 in Mozilla Research as a part of an experiment to write a better, safer browser engine
- Initially developed in OCaml
- Based on the LLVM(*)
- First public announcement June 7th, 2010
- First public release: January 17, 2012
- Looked very Go-like
  - Go final release date: March 2012
  - Who needs Go, but from Mozilla?
- Rust 1.0 release: May 15th, 2015
- Rust 2018 release: November 15th, 2018
  - Considered the “embedded stable release”

ferrous systems

Rust is older than you think!

Interesting aspects

- Rust was developed in an engineering organisation that builds one of the largest C/C++ codebases in the world.
- Rust was always *industrial application of research*
  - All released code was always under QA
  - Rust always had regular releases
  - There was a focus on training the team
    - Regular deprecations
    - RFC based development
- Strong choice of focus
  - Do we write our own code generator? No.
  - Do we write our own optimization framework? No, at least not at first.
  - Do we want to innovate syntax? Only when necessary.
  - Do we want to build a sound type system? Yes!
  - Do we want to have good tooling? Yes!

“Finally, a language from people who understand why we still use C!”

The early years: Experimentation and Removal

- Can you have memory safety without a Garbage Collector?
  - Experiment: port Servo to not using a GC
- Why doesn't Rust have classes?
  - It had them, they didn't really fit.
- Can we write large software in Rust?
  - How about a browser engine?
- Can we do bare-metal programming in Rust?
  - https://zinc.rs
- Tune the "innovation budget"
  - How many novel concepts can we introduce?

The path from 2012 to release was generally marked by *pragmatic removal* over additions while the rest of the language was tuned to a cohesive whole.

2015: Release

- Claim: “Safe, Concurrent, Fast”
- Stable core language
  - Ownership and Borrowing at its core
  - Along with a number of important tools
- Stable for server-side use
  - Runtime details are not stable yet
- Support out of the box for cross-compilation
  - Especially to Arm
  - Reliance on the LLVM is a huge booster
- Start of the release model
  - Every 6 weeks, a beta is forked
  - Beta is in testing for 6 weeks
  - Is released as stable afterwards

ferrous systems

Adoption of Rust 2015

Fast usage in critical and stability-focused service codebases.

- Amazon Firecracker
- Dropbox Backend services
- NPM delivery service
- Chef Habitat
- Microsoft IoT Edge
- Google Fuchsia
- Facebook Libra

- And finally, in 2017: Firefox

ferrous systems

Note: these are individual projects, not organisation-wide adoptions!

2018 Release

- Claim: “Performance, Reliability, Productivity”
- Ships a new language *profile*
  - Irons out a lot of issues with the ergonomics of 2015
  - Codebases with Rust 2018 and 2015 can be mixed
- Stable embedded development is a hallmark feature
- Much more ergonomic memory safety checking
- Migration tooling provided
- Preparation for async/.await

- 2018 is very widely adopted.

ferrous systems

Rust for embedded

- Early experiments have always been around
  - Zinc.rs
  - Early projects informed us of what *not* to do or decide
- For the 2015 release, embedded was set as a non-goal
  - But also with an eye to not *block* usage on embedded systems
- Set as a target for the 2018 release
  - Set up a working group in 2017
  - Working group was done in August 2018
  - Release had to be pushed to November 2018
- Since then, we have a notable uptake of Rust usag

ferrous systems

Some cortex-m code

```
//! Minimal `cortex-m-rt` based program

#![no_main]
#![no_std]

extern crate cortex_m_rt as rt;
extern crate panic_halt;

use rt::entry;

// the program entry point
#[entry]
fn main() -> ! {
    loop {}
}
```

ferrous systems

Notable embedded WG output

- Embedded HAL
  - A hardware abstraction layer based on Rust concepts
- A user-level book
  - https://rust-embedded.github.io/book/
- A low-level book
  - E.g. How do I get started with a new target?
  - https://github.com/rust-embedded/embedonomicon
- Code generation tooling for SVD
  - https://github.com/rust-embedded/svd2rust
- Further improvements to the cross-compilation story

ferrous systems

Developing using a board kit

```
#[entry]
fn main() -> ! {
    let mut dwm1001 = DWM1001::take().unwrap();

    let mut timer = dwm1001.TIMER0.constrain();

    loop {
        dwm1001.leds.D12.enable();
        delay(&mut timer, 20_000); // 20ms
        dwm1001.leds.D12.disable();
        delay(&mut timer, 230_000); // 230ms
    }
}
```

This lets a Decawave DWM1001 development board blink!

ferrous systems

Future: async/.await on bare metal

```
async fn send_to_uart(uart: &mut UartTx, data: &[u8]) -> Result<()> {
    let iter = input.chunks(UART_DMA_SIZE);
    for chunk in input {
        uart.send(chunk).await?;
    }
}
```

This allows us to write linear code while DMA actions happen concurrently!

Does not require a runtime!

https://ferrous-systems.com/blog/embedded-async-await/

ferrous systems

Tooling

- Rust ships important tooling along with the language
  - Cargo: A build tool and dependency manager
  - Rustup: A toolchain manager
  - Rustdoc: Documentation tool and tester(!)
  - Clippy: A code linter
  - Rls: Rust language server, for IDE support
  - Rustfmt: A standard code formatter
- All of them are cross-compile aware

To cross-compile our code from above:

```
rustup target install thumbv7em-none-eabihf
cargo build --target thumbv7em-none-eabihf
```

ferrous systems

Adoption Strategies

- Great tooling for binding Rust to C and C to Rust
- Most codebases adopting Rust are gradually moved over
  - Identify parts that are problematic
  - Evaluate if they play to Rusts strength
  - Use Rust there first
- Make sure that the first project is non-critical and well-understood technically

Tooling for cross-language integration is very advanced, due to the Firefox integration.

Hallmark Projects

- Iqlusion armistice: https://github.com/iqlusioninc/armistice
  - Armistice is a programmable platform for running security sensitive applications and provides access to the cryptographic keys stored on the device.
  - Targeting the F-Secure USB Armory (Cortex-A)
  - Without running an embedded Linux
- Amazon Firecracker: https://firecracker-microvm.github.io/
  - A lightweight virtualisation system
  - Powering Amazon Lambda

Hallmark Projects

- TockOS: https://www.tockos.org/
  - “An embedded operating system designed for running multiple concurrent, mutually distrustful applications on low-memory and low-power microcontrollers.”
  - Started development on May 19th, 2015
- Google OpenSK: https://github.com/google/OpenSK
  - A FIDO2 Authenticator base developed as a Tock application for Nordic nrf52

Rust Today

- 250 people project
- 4 times most loved language on Stackoverflow survey
- Split over working groups and teams
  - Core
  - Language
  - Compiler
  - Libraries
  - Release
  - Triage
  - Community/Comms/Marketing
  - Event support
  - ....
- Still a very lean project!
- Huge international community, especially also China, Japan, Russia and South Africa
- Involvement in the project can be done from 2 hours a week to 40 hours a week!

Quality assurance

Rust has very high standards on quality assurance:

- Clean master (all code inserted into the project is already fully tested)
- Industry-Standard security practices (no infrastructure access to anyone who doesn't need it, 2FA everywhere)
- We ship no unfinished things
  - Unstable features are not present in the release version
- Checks and balances
  - Core has no technical power
  - New features need signoff by the release team
  - 6 weeks release cycle keeps cost of blocking low
- Collaboration with vendors
  - For example, collaborating with Arm for better quality assurance

Quality Assurance: Crater

What if we are not sure if a fix breaks code?

We download all code we know about, build it and check for regressions!

This includes working with the libraries authors, should they accidentally rely on type system implementation bugs.

ferrous systems

Rust as an innovation driver

- Rust has inspired Swift (and Swift has inspired Rust)
- Tooling is seen as a reference to live up to
- Error messages and diagnostics systems are seen as top of the class
  - GCC 9 error messages are very inspired by ours!
- Governance structure is seen as a successful divergence from Design by Committee and BDFL (Benevolent Dictator)
  - Especially the holistic approach to governance, marrying the technical and the structural sides of programming languages
- Type level strategies have made it in multiple other languages
- Is used as research base at multiple universities
- “Edition” concept is evaluated as adoption into other languages

Rust improves other languages as well!

Sealed Rust

The next logical step for us is enabling Rust in the safety critical space.

Ferrous Systems has started an initiative called “Sealed Rust”, to qualify the Rust Language and Compiler for use in the Safety Critical domain.

https://ferrous-systems.com/blog/sealed-rust-the-pitch/

https://ferrous-systems.com/blog/sealed-rust-the-plan/

2020: Rust is here to stay

- Rust has crossed the threshold from *individual projects* to *organisation-wide adoption*
- All major players now have substantial Rust codebases
- It has become a general language of choice for startups in the field

“We are adopting Rust.” - Ryan Levick, Microsoft, RustFest 2019 Keynote

ferrous systems

We'd like to make it a pleasant stay!

Company Contact:
Florian Gilcher
florian.gilcher@ferrous-systems.com
+49 172 8122469 (Signal, Whatsapp, Telegram)
office@ferrous-systems.com

Project Contact:
community@rust-lang.org
core@rust-lang.org (for trusted inquiries)

skade on https://www.rust-lang.org/community

ferrous systems

Thank you

Accelerating deployment in the Arm Ecosystem

Linaro

TECH DAYS

Rust and Arm

Robin Randhawa

Linaro TECH DAYS

About me

- I work in Arm's Open Source software group at Cambridge, UK
- I've been a friend of Linaro since it's inception
- I was the Technical Lead of Arm's OS Power-Perf Team for about 7 years
- I've been looking at ***Safety themed architecture*** for the last 2 years
- This is called the ***OSS Safety Track***
- Primary focus area - ***The overlap between Safety domains and Open Source***

Safety Track Charter

"Promote the uptake of Arm IP in safety critical domains using Open Source software as a medium"

Safety and security areas of focus

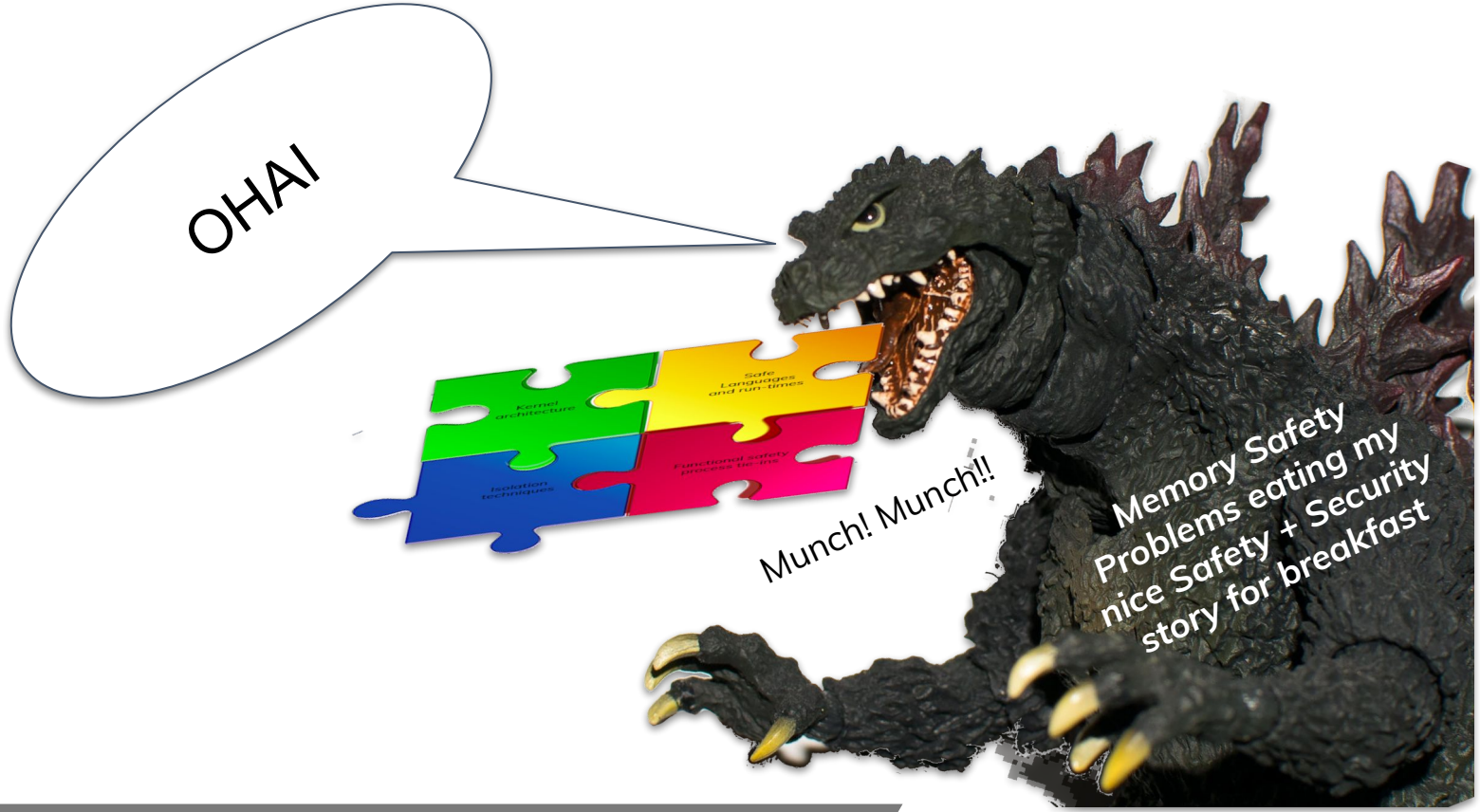
- Microkernel vs Monolithic
- What do the proprietary OS' do and why ?
- Is that sensible ?
- Can it be done with OSS ?
- Is there some precedent ?

- Is there a truly compelling safety argument ?
- How burdensome is it for the programmer ?
- Does it have mass appeal ?
- Is there evidence of industrial pick-up ?
- Is there a walled garden ?
- Is there an ivory tower ?

- If cost wasn't an issue, how robust can hardware be ?
- If it is an issue, how much can we rely on purely software isolation techniques ?
- Are both needed ?
- What impact do those techniques have up and down the stack ?

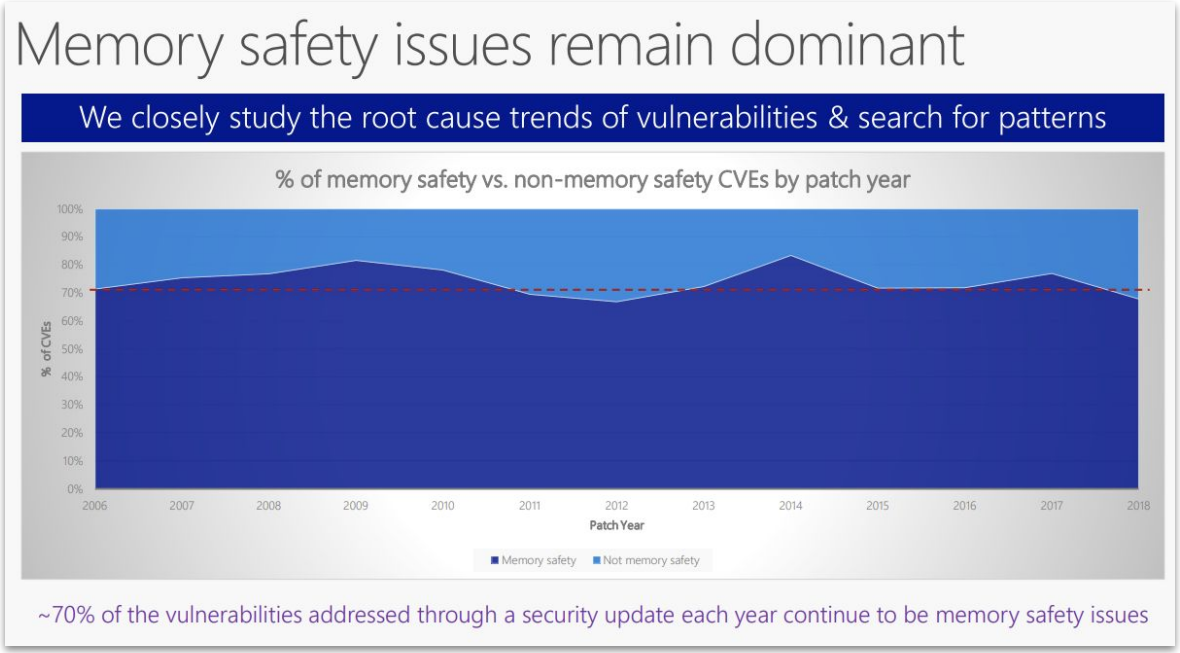
- What are these standards ?
- Do they truly improve safety and security ?
- What costs do they impose ?
- Can OSS work with those standards ?
- If not - is there another way ?

That memory safety problem...



Memory safety is the state of being protected from various software bugs and security vulnerabilities when dealing with memory access, such as buffer overflows and dangling pointers

Memory safety problems have been and continue to be the bane of safety and security architecture



Courtesy: Friends in Microsoft Research (who are Rust fans BTW)

The cost of memory safety

Tooling

Hardware design

Talent

Software Development Standards

Indemnity

Culture

Safety Assessment

Safety Certification

C Language Extensions

- Is everywhere… No really - EVERYWHERE
- Everywhere is not a nice place to be (from a safety and security PoV)
- Arm invests in finding better options for its partners
- Technically sound options with the right attributes are attractive to Arm
- Rust has many attractive attributes

The language spectrum

- Fine-grained control of code and data placement (close to the metal)
- Potential to get close to wire-speed throughput from the hardware
- Potential to engineer deterministic functionality
- Programmer bears the responsibility of memory allocation and reclamation
- Generally restricted programming model
- Generally super easy to shoot oneself in the foot (memory safety, concurrency, parallelism quagmires)
- Anything “funky” is out of the language scope and a part of commodity libs (Hashes etc)
- General tendency to be governed by committee and standards bodies

- Memory alloc/dealloc taken care of by language and run-time entities (VMs etc)
- Generally more pleasant to program in (batteries included nature of the language and core libs)
- Generally less prone to the low-level language safety pitfalls
- Wire-speed throughput is generally hard/impossible
- Deterministic function is generally hard/impossible
- Generally not usable for low level systems programming
- Don’t typically find patronage/use in deep safety related domains

“Low-level” languages

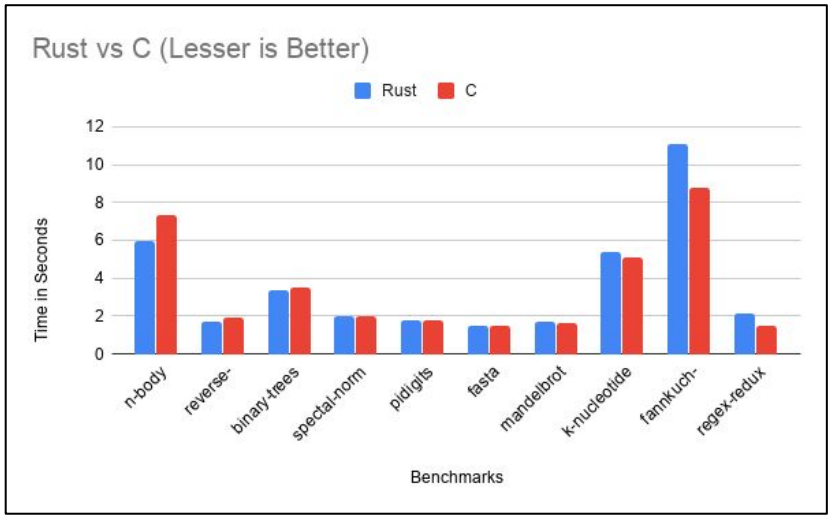
“Hi-level” languages

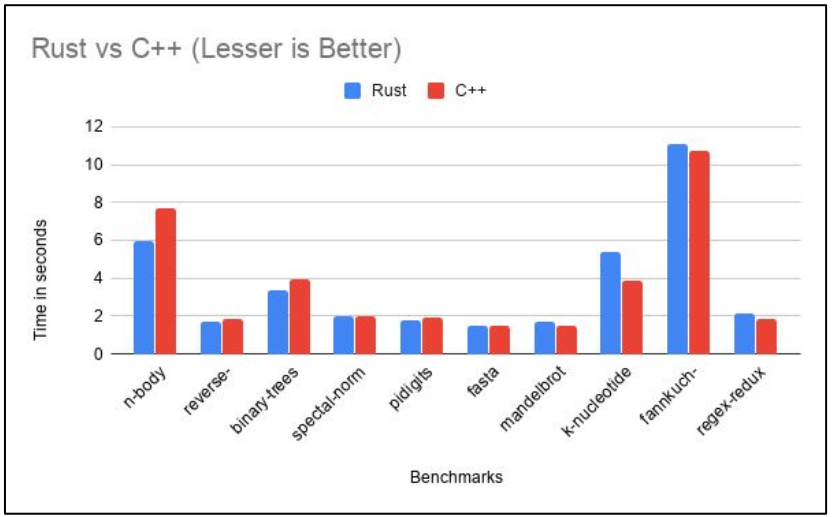
The Best of both Worlds

Think C like speed, With C++ like capabilities, Expressiveness like Python and Ruby
and
Compile Time Memory Safety

Contrasting programming languages is hard....

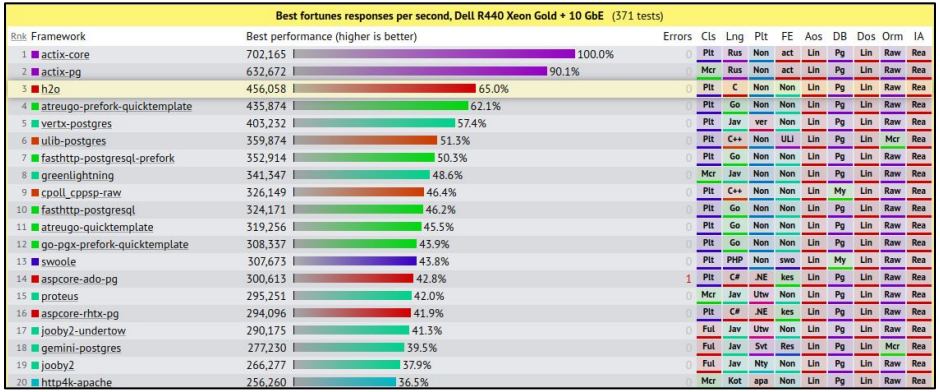
Microbenchmarks...





- The Computer Language Benchmarks Game (x86_64 host, AArch64 hosted numbers WiP)
- But microbenchmarks aren't representative!
- Yes but these are the only publically available collection of algorithms
- That are implemented idiomatically in multiple languages
- Also - don't forget that microbenchmarks are essential to system design
- Not to be taken lightly....
- The Rust numbers are definitely compelling

(Some) Macro Benchmarks



Best fortunes responses per second, Dell R440 Xeon Gold + 10 GbE (371 tests)

Rnk	Framework	Best performance (higher is better)	Errors	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA
1	actix-core	702,165 100.0%	0	Plt	Rus	Non	act	Lin	Pg	Lin	Raw	Rea
2	actix-pg	632,672 90.1%	0	Mcr	Rus	Non	act	Lin	Pg	Lin	Raw	Rea
3	h2o	456,058 65.0%	0	Plt	C	Non	Non	Lin	Pg	Lin	Raw	Rea
4	atreugo-prefork-quicktemplate	435,874 62.1%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
5	vertx-postgres	403,232 57.4%	0	Plt	Jav	ver	Non	Lin	Pg	Lin	Raw	Rea
6	ulib-postgres	359,874 51.3%	0	Plt	C++	Non	ULi	Lin	Pg	Lin	Mcr	Rea
7	fasthttp-postgresql-prefork	352,914 50.3%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
8	greenlightning	341,347 48.6%	0	Mcr	Jav	Non	Non	Lin	Pg	Lin	Raw	Rea
9	cpoll_cppsp-raw	326,149 46.4%	0	Plt	C++	Non	Non	Lin	My	Lin	Raw	Rea
10	fasthttp-postgresql	324,171 46.2%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
11	atreugo-quicktemplate	319,256 45.5%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
12	go-pgx-prefork-quicktemplate	308,337 43.9%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
13	swoole	307,673 43.8%	0	Plt	PHP	Non	swo	Lin	My	Lin	Raw	Rea
14	aspcore-ado-pg	300,613 42.8%	1	Plt	C#	.NE	kes	Lin	Pg	Lin	Raw	Rea
15	proteus	295,251 42.0%	0	Mcr	Jav	Utw	Non	Lin	Pg	Lin	Raw	Rea
16	aspcore-rhtx-pg	294,096 41.9%	0	Plt	C#	.NE	kes	Lin	Pg	Lin	Raw	Rea
17	jooby2-undertow	290,175 41.3%	0	Ful	Jav	Utw	Non	Lin	Pg	Lin	Raw	Rea
18	gemini-postgres	277,230 39.5%	0	Ful	Jav	Svt	Res	Lin	Pg	Lin	Mcr	Rea
19	jooby2	266,277 37.9%	0	Ful	Jav	Nty	Non	Lin	Pg	Lin	Raw	Rea
20	http4k-apache	256,260 36.5%	0	Mcr	Kot	apa	Non	Lin	Pg	Lin	Raw	Rea

[Taken from the latest TechEmpower web server framework shootout]

Gaps/Criticisms of Rust

General

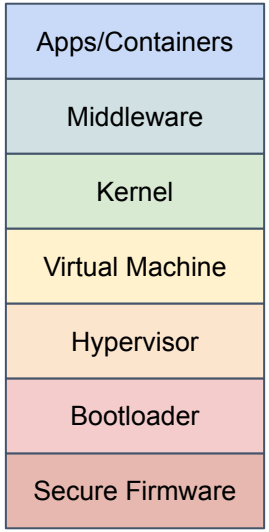
- Initial learning curve
- High compilation times
- Idiomatic fitness problems (GUIs)
- ABI stability
- No ISO style language specification

Arm specific

- AArch64 targets are not classified as Tier-1 Rust targets (yet)
  - Tier class definitions and policies

Rust for Arm

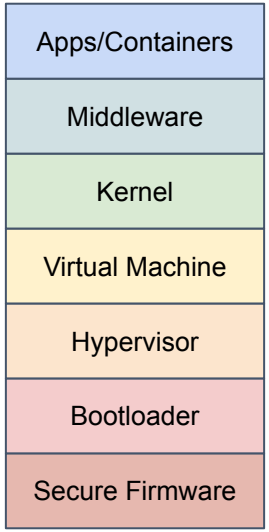
At a high level, most modern software stacks look like this



Rust for Arm

At a high level, most modern software stacks look like this

It doesn't make sense to "oxidize" everything

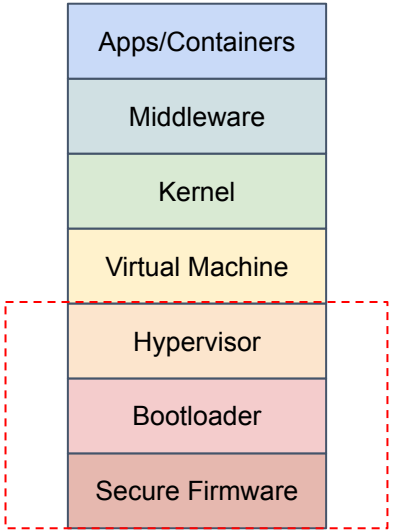


Rust for Arm

At a high level, most modern software stacks look like this

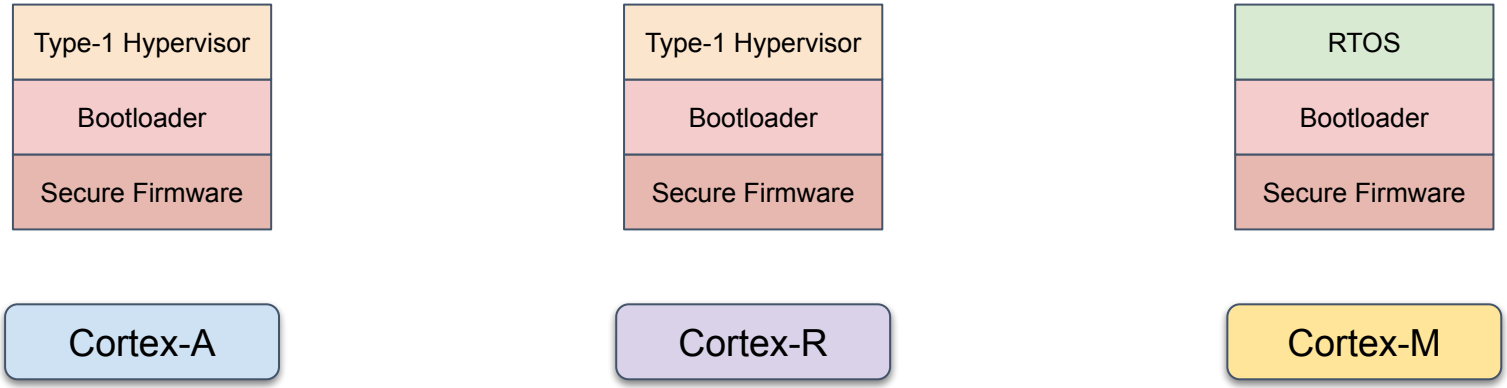
It doesn't make sense to "oxidize" everything

But given's Rust's strengths, some layers are really good candidates for "oxidation"

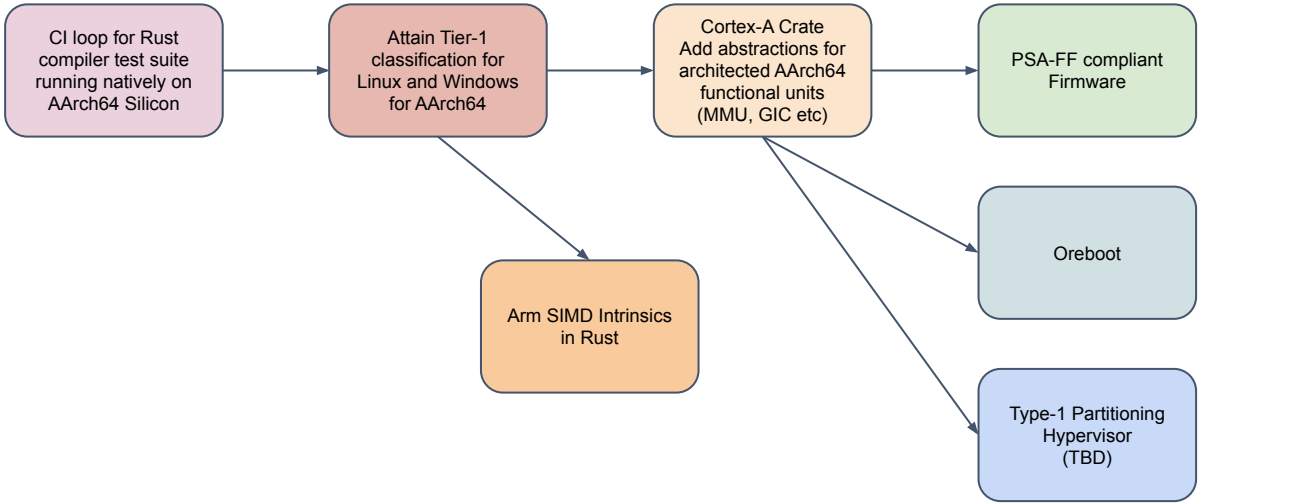


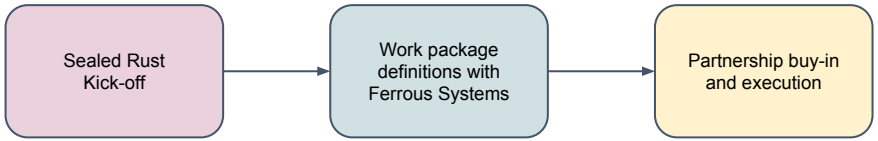
Rust for Arm

Finding ways to work with the Rust community on these aspects...



The future


CI loop for Rust compiler test suite running natively on AArch64 Silicon
Attain Tier-1 classification for Linux and Windows for AArch64
Cortex-A Crate Add abstractions for architected AArch64 functional units (MMU, GIC etc)
PSA-FF compliant Firmware
Oreboot
Arm SIMD Intrinsics in Rust
Type-1 Partitioning Hypervisor (TBD)


Sealed Rust Kick-off
Work package definitions with Ferrous Systems
Partnership buy-in and execution

Get in touch!

- If you would like to share ideas and find ways to collaborate:

Mail: robin.randhawa@arm.com
Github: @raw-bin
Twitter: @throbbin_robin

Thank you

Accelerating deployment in the Arm Ecosystem

Linaro

TECH DAYS