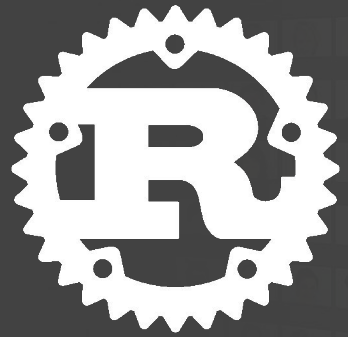
True Story!

How Rust for AArch64 Linux became a Tier-1 Target Platform



arm

Robin Randhawa (Arm)
Florian Gilcher (Ferrous Systems)

Linaro Connect
VIRTUAL • SPRING 2021

Introductions

Florian Gilcher

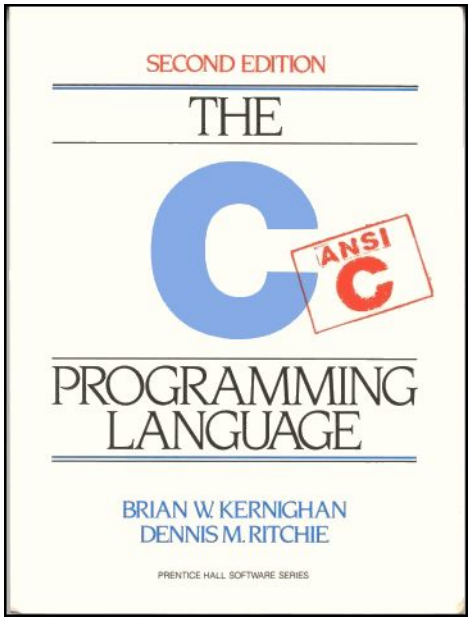
- Rust developer since 2013, Trainer since 2015
- Part of Rust community team and Rust core, Board Member Rust Foundation
- Founder of the Berlin users group and RustFest and Oxidize conferences
- Co-Founder of Ferrous Systems, a group of Rust developers providing
  - Help in adopting Rust
  - Training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

Robin Randhawa

- Embedded Systems Tinkerer
- Programming language enthusiast
- I work in Arm's Open Source software group at Cambridge, UK
- Current focus areas are safety themed software architecture in Automotive, Industrial and Robotics domains
- Long time friend of Linaro since the early days

So, what’s the big deal with Rust and why should anyone care ?

The C Programming Language



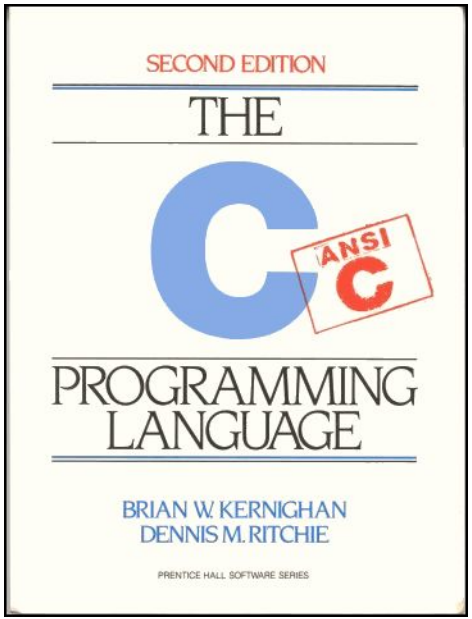
Important design intent and general philosophy:

- Maximal code portability
- Fine grained control over machine code and data layout
- Maximal performance (‘close to the metal’)

To achieve this:

- C assumes an abstract machine model
- Permits a level of ‘intentional ambiguity’ in the language specification

The C Programming Language



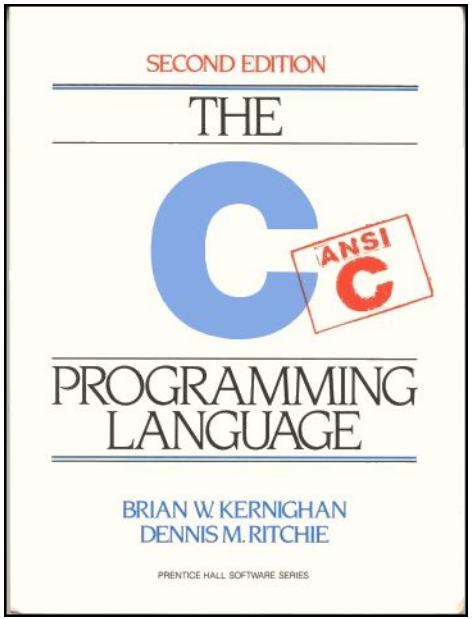
That 'intentional ambiguity' was useful:

- Because it enabled near absolute control of the underlying machine
- Gave rise to the 'C is the assembly of higher level languages' meme

But that brought with it a number of problems:

- Code safety automatically took a back-seat
  - Type size ambiguity
  - Pointer/Integer ambiguity
  - Buffer {over/under}-flow
  - Use-after-free
  - Etc

The C Programming Language



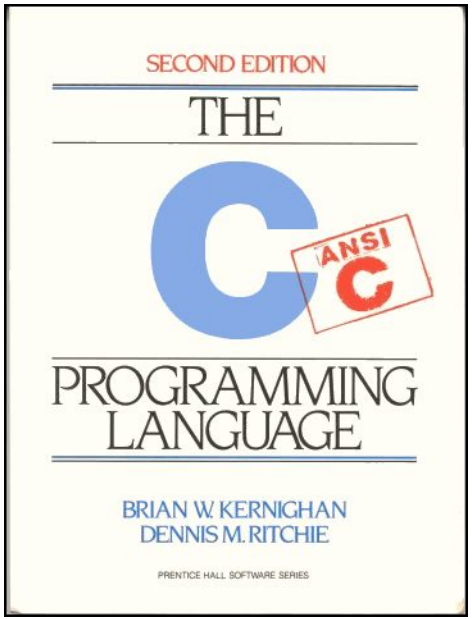
That 'intentional ambiguity' was useful:

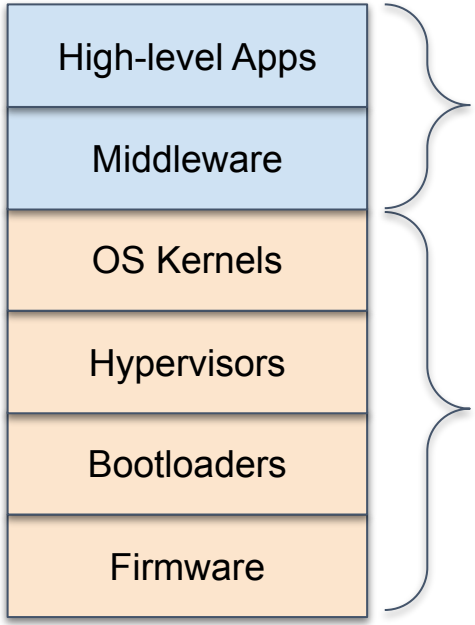
- Because it enabled near absolute control of the underlying machine
- Gave rise to the 'C is the assembly of higher level languages' meme

- That put the onus of writing safe code on the programmer
- That became a ***significant*** problem for large projects
- Especially for concurrent and/or parallel programming
- To rationalise programming for safety and security, languages evolved in ways significantly different from C
  - Dynamic typing
  - Garbage collection
  - (many more)
  -

That made some very pronounced trade-offs very apparent!

The C Programming Language



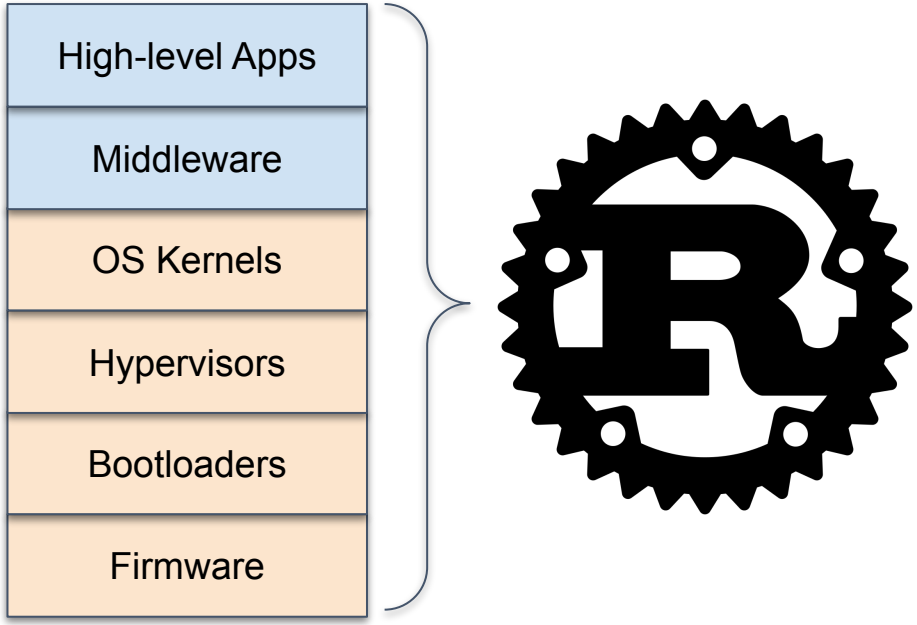


Java, Python, Ruby, C#, Scala et al became dominant
(But applicability for time-determinism sensitive domains became limited)

Rich web servers, Banking applications, HFT, Scientific computing, Etc

C became dominant
(But the onus lay largely on the programmer for safety and security)

And then Rust comes along...



- Compile time static analysis
- Compile time guarantee of memory safety
- ***Even for concurrent and/or parallel programs***
- ***Even considering synchronisation practicalities***

- Rust is probably the only programming language intended to be a systems programming language that **also** finds popular use in the full stack

Memory safety example: Vector reallocation

```
1  fn main() {
2      let mut vec = vec![1,2,3];
3
4      let elem_ref = &vec[1];
5
6      vec.push(4);
7
8      println!("{}", elem_ref);
9  }
```

Memory safety example: Vector reallocation

```
1  fn main() {
2      let mut vec = vec![1,2,3];
3
4      let elem_ref = &vec[1];
5
6      vec.push(4);
7
8      println!("{}", elem_ref);
9  }
```

References current vector allocation

May reallocate

If reallocated, this is illegal

Memory safety example: Vector reallocation

```
   Compiling playground v0.0.1 (/playground)
error[E0502]: cannot borrow `vec` as mutable because it is also borrowed as immutable
 --> src/main.rs:6:5
  |
4 |     let elem_ref = &vec[1];
  |                     --- immutable borrow occurs here
5 |
6 |     vec.push(4);
  |     ^^^^^^^^^^ mutable borrow occurs here
7 |
8 |     println!("{}", elem_ref);
  |                    -------- immutable borrow later used here
```

Synchronisation example

```
1  use std::sync::Mutex;
2
3  fn lock_and_write(
4      mtx: &Mutex<Vec<u8>>
5  ) {
6      let mut lock = mtx.lock().unwrap();
7
8      lock.push(42);
9  }
```

Synchronisation example

```
1  use std::sync::Mutex;
2
3  fn lock_and_write(
4      mtx: &Mutex<Vec<u8>>
5  ) {
6      let mut lock = mtx.lock().unwrap();
7
8      lock.push(42);
9  }
```

1.
2.
3.

1. Ensures the function can interact with the Mutex, but not manipulate the Mutex itself
2. Ownership tracking makes sure the lock will be released before the end of the function
3. Inner reference tracking (‘borrowing’) will make sure a reference to the locked data cannot be held longer than the lock

Rust effectively provides proper call ordering for resource based APIs

Rust synchronisation hidden gems

- Rust is memory-safe *even in concurrent settings*
- That means the compiler detects when synchronisation of data for concurrent access is necessary
- That *also* means Rust detects when unsynchronised (read-only) access is safely possible!

Rusts core strength is finding bugs early

- The later a bug is found, the more expensive it is
- Rust and its compiler put an emphasis on finding bugs early
- All this through static analysis!

Rust provides good (even fun!) ways to solve hard problems

Report: Rust is the most beloved programming language for five years running

- Rust continues to maintain its spot as the most beloved of programming languages among professional developers. ...
- Rust was followed by **TypeScript** at 67.1%, **Python** at 66.7%, Kotlin at 62.9%, and Go at 62.3% for most loved languages.

2020 developer survey finds Rust 'most loved,' Python 'most wanted,' and Perl, Scala, and Go the 'top paying' languages

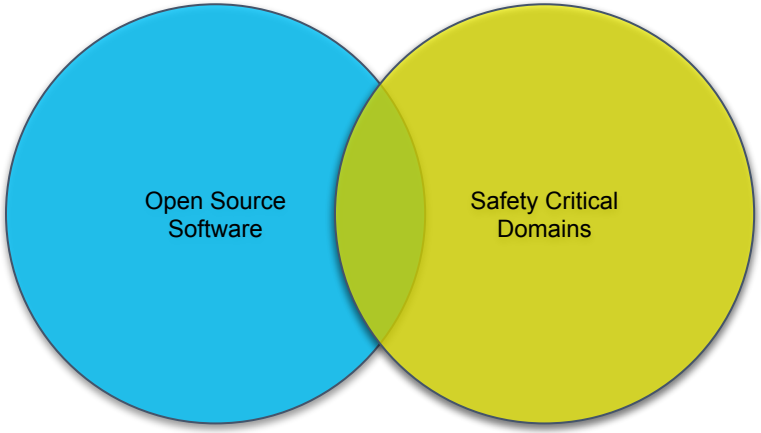
By Paul Krill

Editor at Large, InfoWorld | 2 JUNE 2020 11:00 BST

The Importance of the Tier-1 ‘badge’ and how we made it happen!

The Beginning

I got asked to explore ***if, why, where and how*** Open Source Software can help improve Safety Critical Domains



The Beginning

Safety and Security themed Kernel design

INTEGRITY Real-Time Operating System

QNX

PikeOS 5
CERTIFIED RTOS & HYPERVISOR

VxWorks

Proprietary

Open Source

L4Re

The Beginning

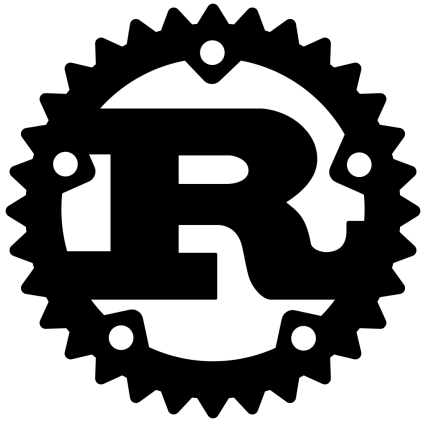
Safety and Security themed Programming Languages

The Beginning

Building a subjective view of merit

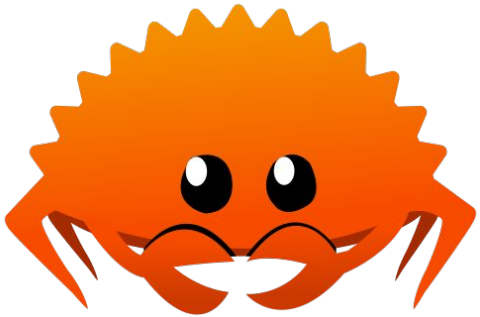
Technical Aspects	Other Aspects
• Safety and Security focused design • Explicit support for memory safety • Even in concurrent and/or parallel scenarios • Programmer expressiveness • Standard library richness • Etc	• Development model • Governance model • Community dynamic • Ecosystem uptake • Etc

The Beginning



The Rust Language

Ferris the Crab



Evaluating the language's claims

https://www.rust-lang.org

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability

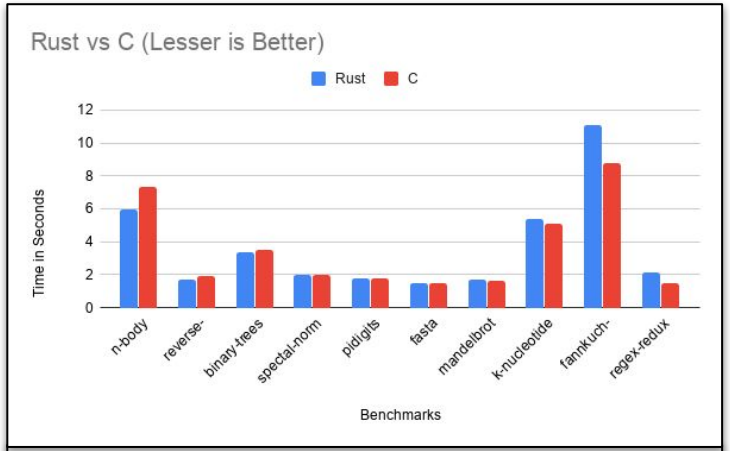
Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

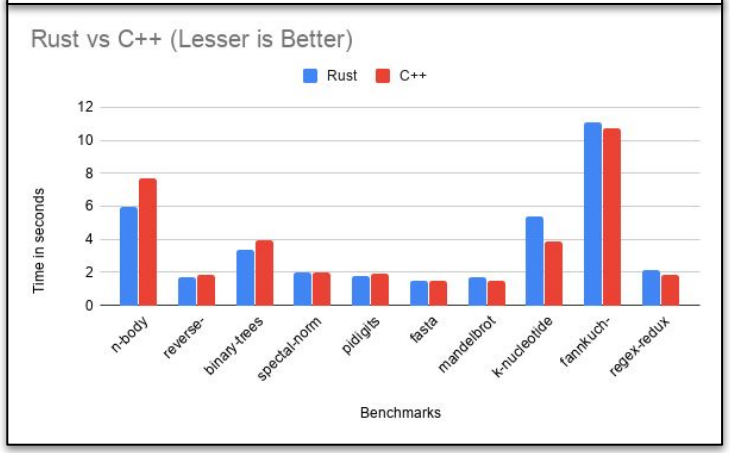
Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

The Beginning
Discovering Value

Evaluating the language's claims
Performance (microbenchmarks)

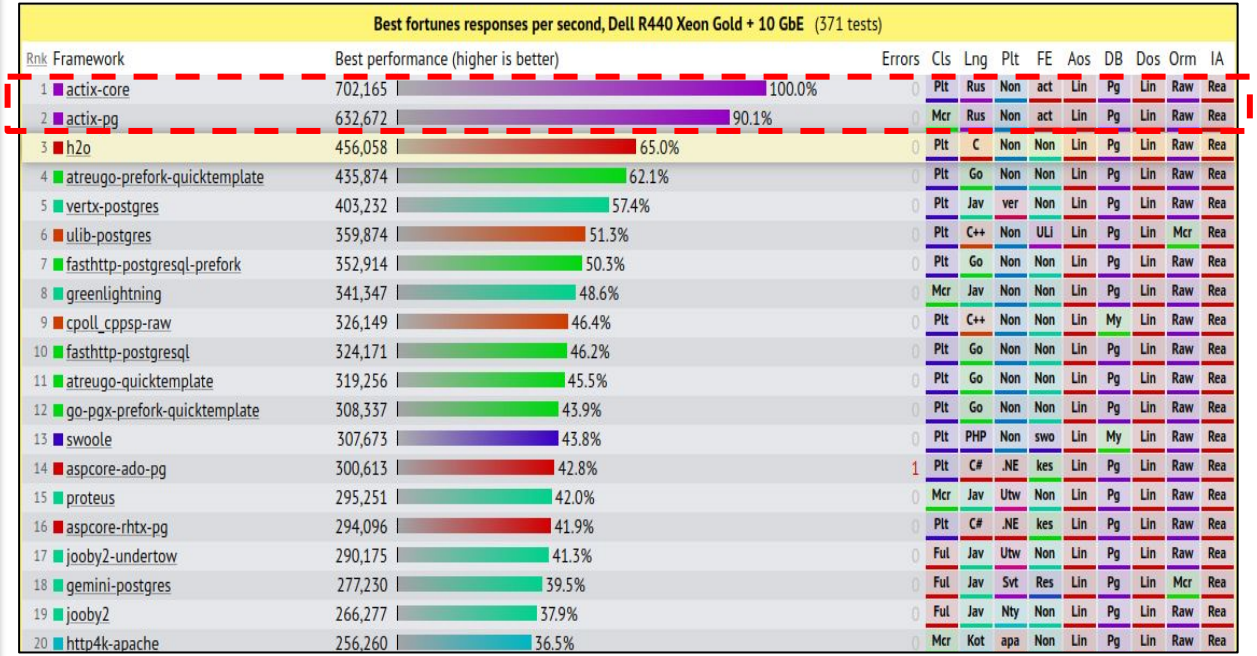




[Computer language benchmarks game]

The Beginning
Discovering Value

Evaluating the language's claims
Performance ('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

TechEmpower web server framework shootout

Evaluating the language's claims
Productivity Tools

A ***very*** helpful compiler!

```
struct Color {
    r: u8,
    g: u8,
    b: u8,
}

fn main() {
    let yellow = Color {
        r: 255,
        g: 255,
        d: 0,
    };

    println!("Yellow = rgb({},{},{})", yellow.r, yellow.g, yellow.b);
}


// ------------ Compile-time error ------------
error[E0560]: struct `Color` has no field named `d`
  --> src/main.rs:11:9
   |
11 |         d: 0,
   |         ^ field does not exist - did you mean `b`?

error: aborting due to previous error
For more information about this error, try `rustc --explain E0560`.
```

Evaluating the language's claims
Productivity Tools

A ***very*** helpful compiler!

````
A `break` statement with an argument appeared in a non-`loop` loop.

Example of erroneous code:
```
let result = while true {
    if satisfied(i) {
        break 2*i; // error: `break` with value from a `while` loop
    }
    i += 1;
};
```

The `break` statement can take an argument (which will be the value of the loop
expression if the `break` statement is executed) in `loop` loops, but not
`for`, `while`, or `while let` loops.

Make sure `break value;` statements only occur in `loop` loops:
```
let result = loop { // ok!
    if satisfied(i) {
        break 2*i;
    }
    i += 1;
};
```
````

Evaluating the language's claims

Developer Tools

[Cargo]

- Powerful package manager
- Dependency management
- Reproducible builds
- Custom package registry support
- License checking across dependencies

Evaluating the language's claims

Tools

rust.analyzer

Bringing a *great* IDE experience
to the Rust programming language.

```
fn main() {
    println!("Hello, world!");      (Heyward Fann 1 month ago) 1
}    #[macro_export]
     macro_rules! println

     Prints to the standard output, with a newline.

     On all platforms, the newline is the LINE FEED character (n/U+000A) alone
     (no additional CARRIAGE RETURN (r/U+000D).

     Use the [format!] syntax to write data to the standard output.
     See [std::fmt] for more information.
```

Emacs, Vim, VSCode supported

Evaluating the language's claims

Misc

Complex Data Types (Sequences, Maps, Sets)	**Built-in support for Modular Development**
Generics	**Lightweight Object Orientedness with Traits**
Unsafe 'escape hatch'	**Rich Error handling**
Async support	**Cross-compilation ease**
No GC	**Focused support for bare-metal (no_std) envs**

The Beginning
Discovering Value
Discovering Gaps

- The ‘out-of-the-box’ Rust experience on AArch64 Linux was poor
- No non-x86_64 target was a Tier-1 supported Rust target
- Rust had a top-class language Reference - but no ISO grade language reference

The Beginning

Discovering Value

Discovering Gaps

Building Bridges

Alex Crichton
(Fastly)

- I reached out to Alex who is a prolific Rust community member and who was intimately aware of the ‘Tier-1 Arm problem’ and who was very well connected
- **Credit where due:**
  - The Tier-1 initiative would probably not have happened without Alex’s timely guidance!

Josh Triplett
(Former Intel - Independent Consultant)

- I reached out to Josh who is also a prolific Rust community member and is responsible for the Tier classification definitions
- Josh is also instrumental in the Rust for Linux kernel drivers initiative
- **Credit where due:**
  - Josh’s guidance with the Tier-1 RFC articulation and the accompanying reports was super useful!

The Beginning

Discovering Value

Discovering Gaps

Building Bridges

Core team

Managing the overall direction of Rust, subteam leadership, and any cross-cutting issues

Members

Aidan Hobson Sayers
GitHub: aidanhs

Ashley Williams
GitHub: ashleygwilliams

Manish Goregaokar
GitHub: Manishearth

Mark Rousskov
GitHub: Mark-Simulacrum

Niko Matsakis
GitHub: nikomatsakis

Pietro Albini
GitHub: pietroalbini

Florian Gilcher
GitHub: skade

Steve Klabnik
GitHub: steveklabnik

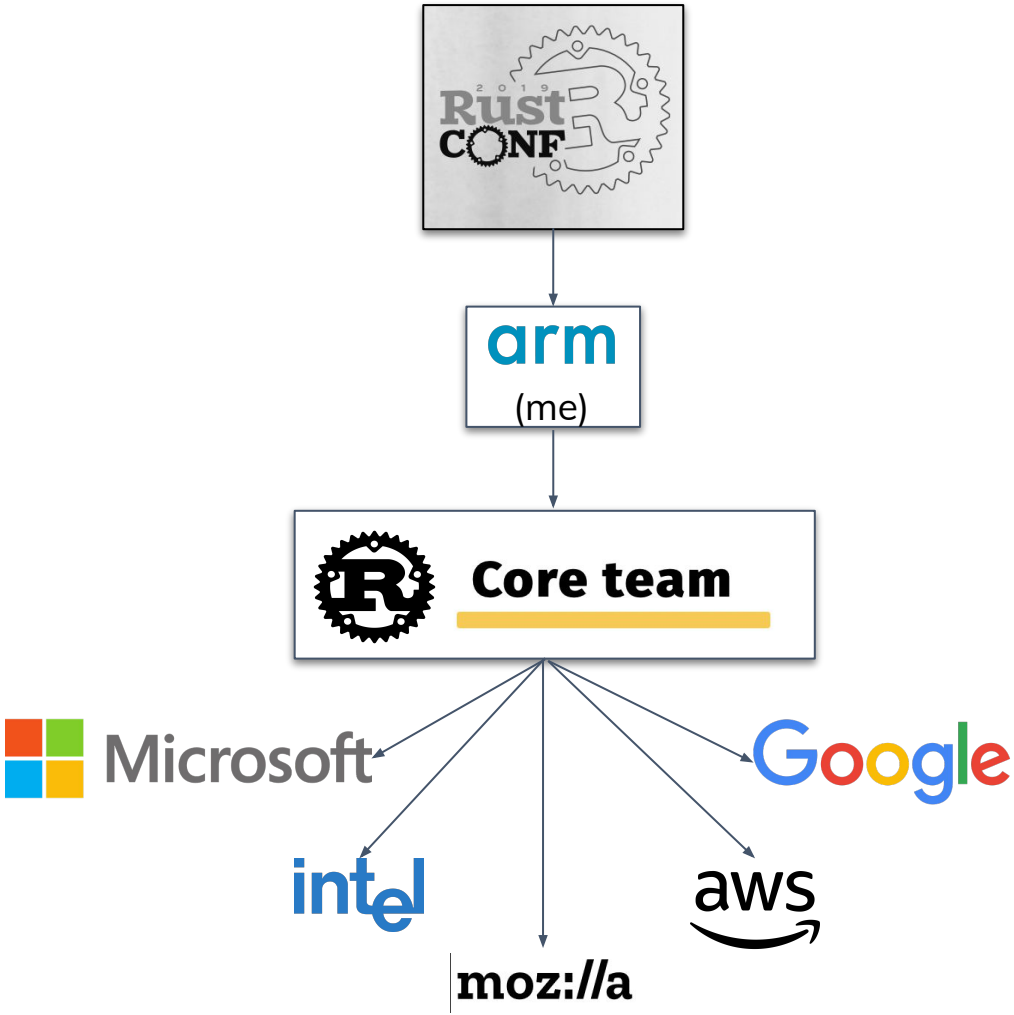
- Alex put me in touch with the Rust Lang Core Team
- A super helpful and wise assemblage of folks with tonnes of experience with language design, deployment and nurturing a community

The Beginning

Discovering Value

Discovering Gaps

Building Bridges



The Beginning
Discovering Value
Discovering Gaps
Building Bridges
Rust's Platform Tier Definitions

Platform Support

Support for different platforms are organized into three tiers, each with a different set of guarantees.

Tier 3

Tier 3 platforms are those which the Rust codebase has support for, but which are not built or tested automatically, and may not work. Official builds are not available.

Tier 2

Tier 2 platforms can be thought of as "guaranteed to build". Automated tests are not run so it's not guaranteed to produce a working build, but platforms often work to quite a good degree and patches are always welcome! Specifically, these platforms are required to have each of the following:

- Official binary releases are provided for the platform.
- Automated building is set up, but may not be running tests.
- Landing changes to the `rust-lang/rust` repository's master branch is gated on platforms **building**. For some platforms only the standard library is compiled, but for others `rustc` and `cargo` are too.

Tier 1

Tier 1 platforms can be thought of as "guaranteed to work". Specifically they will each satisfy the following requirements:

- Official binary releases are provided for the platform.
- Automated testing is set up to run tests for the platform.
- Landing changes to the `rust-lang/rust` repository's master branch is gated on tests passing.
- Documentation for how to use and how to build the platform is available.

The Beginning

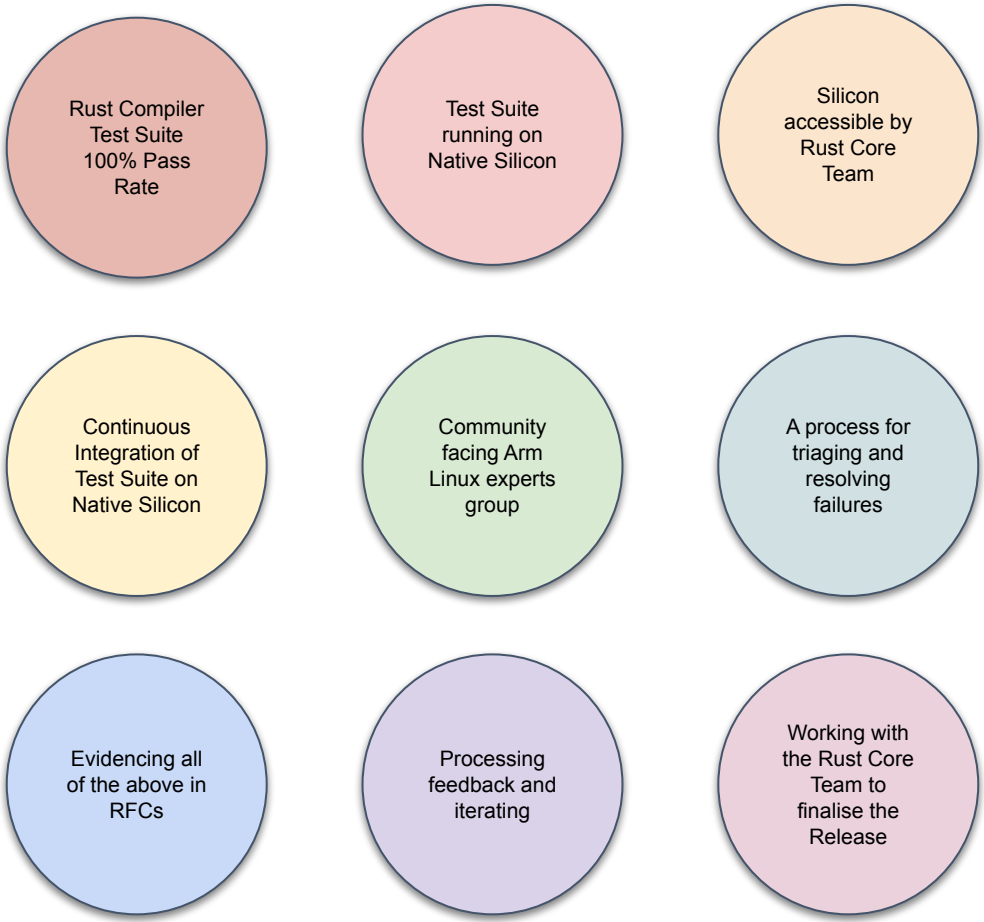
Discovering Value

Discovering Gaps

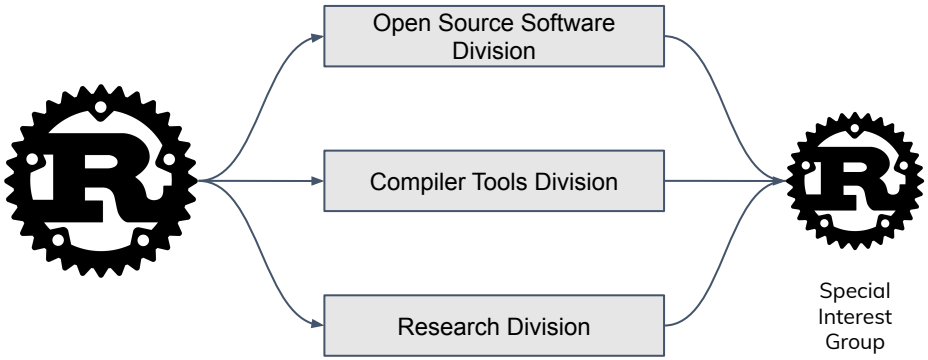
Building Bridges

Rust's Platform Tier Definitions

Scoping the Tier-1 Problem

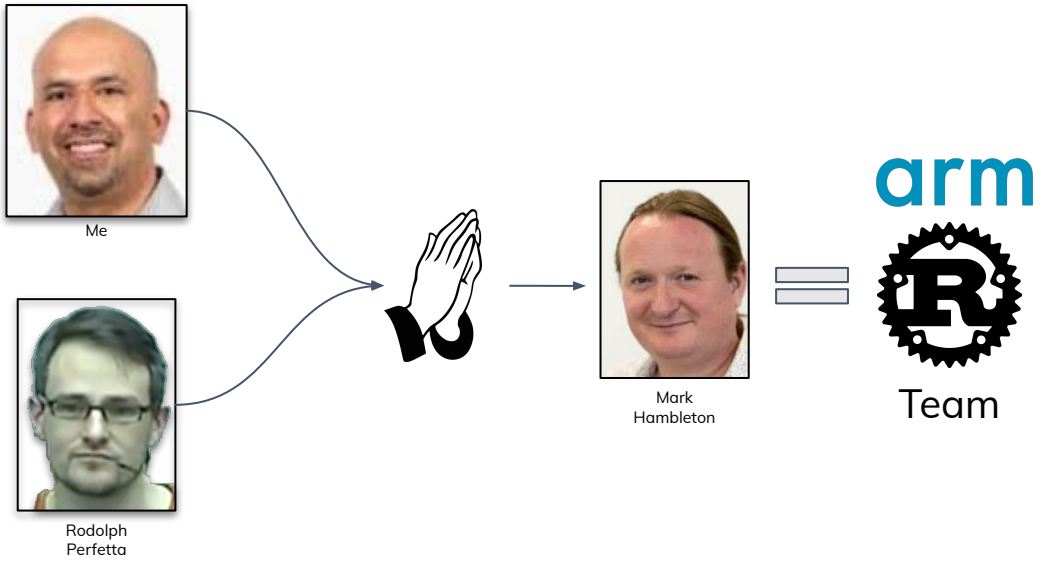


arm



- A number of folks within Arm were already evaluating Rust and were liking it
- I helped set up a cross-org SIG which quickly became very popular

The Beginning
Discovering Value
Discovering Gaps
Building Bridges
Rust's Platform Tier Definitions
Scoping the Tier-1 Problem
Building Mindshare at Arm



- **Credit where due:**
  - Rodolph - for being a fellow conspirator who understood the value here and for supporting me in making the case
  - Mark - for listening, understanding the value and supporting the cause!

- So we got a small team within the Arm Open Source Division to progress the Tier-1 initiative

- The Beginning
- Discovering Value
- Discovering Gaps
- Building Bridges
- Rust's Platform Tier Definitions
- Scoping the Tier-1 Problem
- Building Mindshare at Arm
- Fixing Rust Compiler Tests

arm

Team

- Compiler cross-builds
- Compiler native builds
- Test suite failure triage
- Fixups
- Upstream submissions
- Engagements to Merge

Compiletest test suites

The compiletest tests are located in the tree in the `src/test` directory. Immediately within you will see a series of subdirectories (e.g. `ui`, `run-make`, and so forth). Each of those directories is called a **test suite** – they house a group of tests that are run in a distinct mode.

Here is a brief summary of the test suites and what they mean. In some cases, the test suites are linked to parts of the manual that give more details.

- `ui` – tests that check the exact stdout/stderr from compilation and/or running the test
- `run-pass-valgrind` – tests that ought to run with valgrind
- `pretty` – tests targeting the Rust "pretty
AST
- `debuginfo` – tests that run in gdb or lldb
- `codegen` – tests that compile and then te
optimizations we want are taking effect. S
- `codegen-units` – tests for the monomor
- `assembly` – similar to `codegen` tests, bu
backend can handle provided code.
- `mir-opt` – tests that check parts of the g
correctly or doing the optimizations we e
- `incremental` – tests for incremental con
are performed, we are able to reuse the r
- `run-make` – tests that basically just execu
annoying to write.
- `rustdoc` – tests for rustdoc, making sure
documentation.
- `rustfix` – tests for applying diagnostic s
- `*-fulldeps` – same as above, but indica
(and hence those things must be built)

Other Tests

The Rust build system handles running tests for various other things, including:

- **Tidy** – This is a custom tool used for validating source code style and formatting conventions, such as rejecting long lines. There is more information in the section on coding conventions.

  Example: `./x.py test tidy`

- **Formatting** – Rustfmt is integrated with the build system to enforce uniform style across the compiler. In the CI, we check that the formatting is correct. The formatting check is also automatically run by the Tidy tool mentioned above.

  Example: `./x.py fmt --check` checks formatting and exits with an error if formatting is needed.

  Example: `./x.py fmt` runs rustfmt on the codebase.

  Example: `./x.py test tidy --bless` does formatting before doing other tidy checks.

- **Unit tests** – The Rust standard library and many of the Rust packages include typical Rust `#[test]` unittests. Under the hood, `x.py` will run `cargo test` on each package to run all the tests.

  Example: `./x.py test library/std`

- **Doc tests** – Example code embedded within Rust documentation is executed via `rustdoc --test`. Examples:

  `./x.py test src/doc` – Runs `rustdoc --test` for all documentation in `src/doc`.

  `./x.py test --doc library/std` – Runs `rustdoc --test` on the standard library.

- **Link checker** – A small tool for verifying `href` links within documentation.

  Example: `./x.py test src/tools/linkchecker`

- **Dist check** – This verifies that the source distribution tarball created by the build system will unpack, build, and run all tests.

  Example: `./x.py test distcheck`

- **Tool tests** – Packages that are included with Rust have all of their tests run as well (typically by running `cargo test` within their directory). This includes things such as cargo, clippy, rustfmt, rls, miri, bootstrap (testing the Rust build system itself), etc.

- **Cargo test** – This is a small tool which runs `cargo test` on a few significant projects (such as `servo`, `ripgrep`, `tokei`, etc.) just to ensure there aren't any significant regressions.

Guide to rustc development

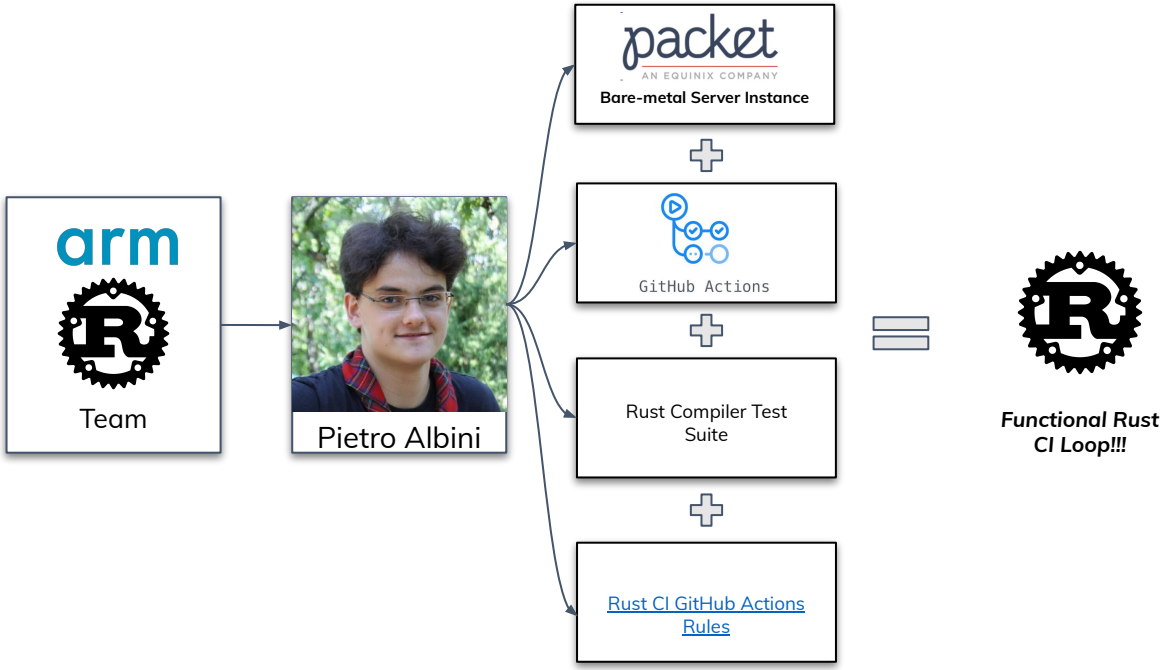
- The Beginning
- Discovering Value
- Discovering Gaps
- Building Bridges
- Rust's Platform Tier Definitions
- Scoping the Tier-1 Problem
- Building Mindshare at Arm
- Fixing Rust Compiler Tests
- Enabling Silicon Access

packet
AN EQUINIX COMPANY

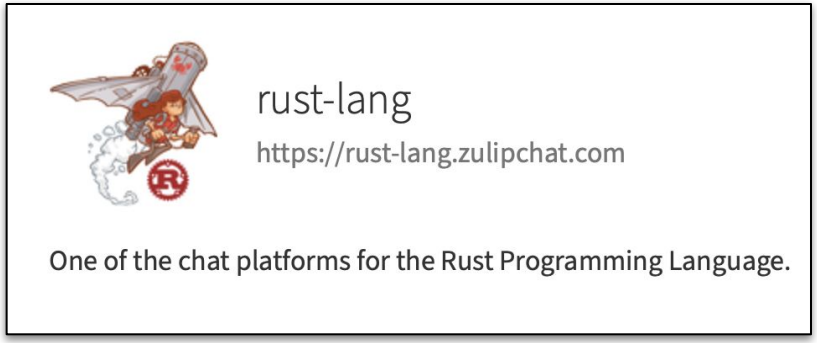
AMPERE®

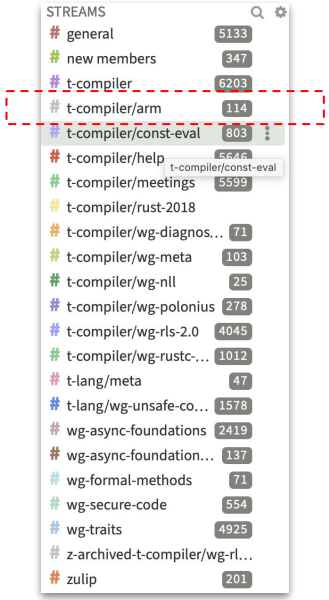
Hardware Specs	
CPU	32 Physical Cores @ 3.3 GHz (1 X AMPERE EMAG)
Memory	128 GB of DDR4 ECC RAM
Storage	480 GB SSD
Network	Dual Port Mellanox NIC (2 × 10GBPS W/ LACP)

The Beginning
Discovering Value
Discovering Gaps
Building Bridges
Rust's Platform Tier Definitions
Scoping the Tier-1 Problem
Building Mindshare at Arm
Fixing Rust Compiler Tests
Enabling Silicon Access
Enabling Rust CI Loops

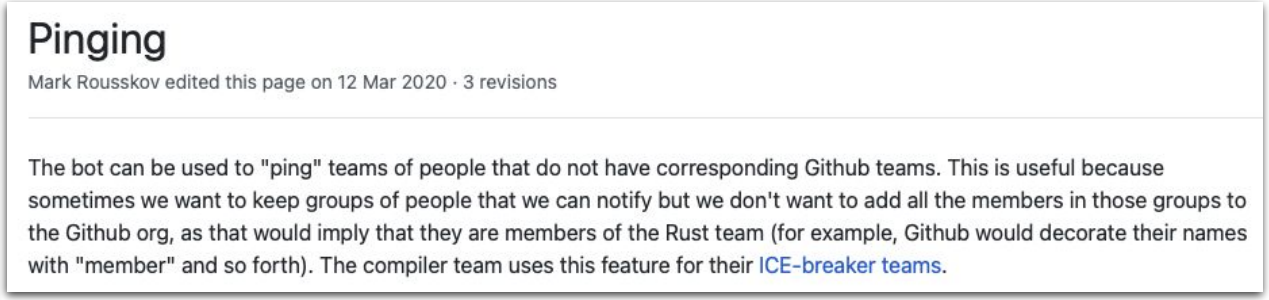


- **Credit where due:**
  - Pietro - for making GitHub Actions fit for purpose and for putting all the bits together to get a complete CI loop working
  - Kushal Koolwal (Arm) and Ed Valmietti (Packet) for understanding the value props and enabling access to silicon





- The Core Team set us up with a temporary Zulip channel focused on Arm Tier-1 attainment
- In addition, a new and perpetual Zulip channel to help with triage of Arm regressions was created (t-compiler/arm) with the Arm Rust Team and some of Rust Core Team
- This satisfied an important Tier-1 requirement



Pinging

Mark Rousskov edited this page on 12 Mar 2020 · 3 revisions

The bot can be used to "ping" teams of people that do not have corresponding Github teams. This is useful because sometimes we want to keep groups of people that we can notify but we don't want to add all the members in those groups to the Github org, as that would imply that they are members of the Rust team (for example, Github would decorate their names with "member" and so forth). The compiler team uses this feature for their ICE-breaker teams.

- The Core Team set us up with a ‘Triagebot’ that would automatically ping us folks on #t-compiler/arm

- The Beginning
- Discovering Value
- Discovering Gaps
- Building Bridges
- Rust's Platform Tier Definitions
- Scoping the Tier-1 Problem
- Building Mindshare at Arm
- Fixing Rust Compiler Tests
- Enabling Silicon Access
- Enabling Rust CI Loops
- Creating an Arm Focus Group
- Creating a Arm Triagebot
- The First RFC Submission

raw-bin commented on 17 Jul 2020 • edited by pietroalbini ▾ Contributor ☺ ⋯

This commit contains an RFC proposal to promote the aarch64-unknown-linux-gnu Rust target to Tier-1.

Rendered

👍 100 🎉 38 ❤️ 40 🚀 35 👀 7

- Feature Name: `promote-aarch64-unknown-linux-gnu-to-tier-1`
- Start Date: 2020-07-17
- RFC PR: rust-lang/rfcs#2959
- Rust Issue: rust-lang/rust#78251

Summary

Promote the Arm aarch64-unknown-linux-gnu Rust target to Tier-1.

The next section provides a justification for the promotion.

Please note that the following are required next steps that should ideally emerge from ensuing discussions:

- An approval from the Compiler Team that Tier-1 target requirements have been met.
- An approval from the Infrastructure Team that the target in question may be integrated into CI.
- An approval from the Release Team that supporting the target in question is viable in the long term.

Justifications provided, with evidence

1.a. The Rust compiler and compiler tests must all build and pass reliably for the target in question.

1.b. All necessary supporting infrastructure, including dedicated hardware, to build and run the Rust compiler and compiler tests reliably must be available openly.

1.c. There must exist a robust and convenient CI integration for the target in question.

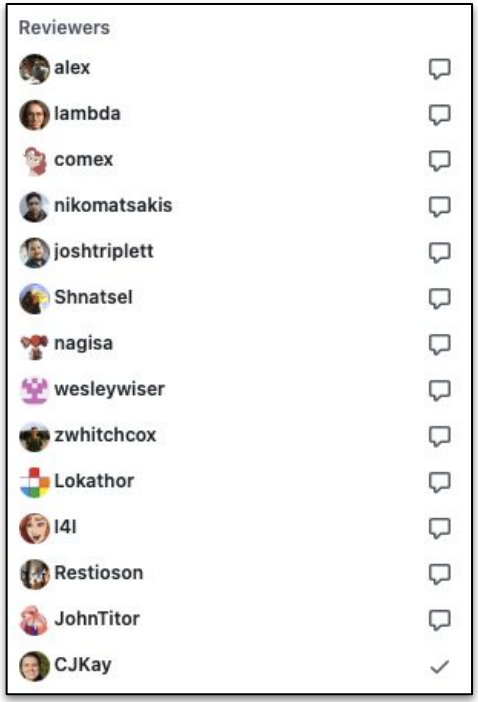
2.a. The long term viability of the existence of a target specific ecosystem should be clear.

2.b. The long term viability of supporting the target should be clear.

2.c. The target must have substantial and widespread interest within the Rust developer community.

2.d. The target must serve the interests of multiple production users of Rust across multiple organizations or projects.

The Beginning
Discovering Value
Discovering Gaps
Building Bridges
Rust's Platform Tier Definitions
Scoping the Tier-1 Problem
Building Mindshare at Arm
Fixing Rust Compiler Tests
Enabling Silicon Access
Enabling Rust CI Loops
Creating an Arm Focus Group
Creating a Arm Triagebot
The First RFC Submission
Processing Feedback



- Most issues to do with education/awareness and were easily resolved
- One key issue was to do with cleaning up misconceptions in the compiler test suites about x86_64 tests and their applicability to AArch64 - resolved by herculean crowd sourcing of **all** the tests!
- Key blocker!!! :(
  - The AArch64 LLVM back-end doesn't have generic support for stack-probing mitigations against stack clashing

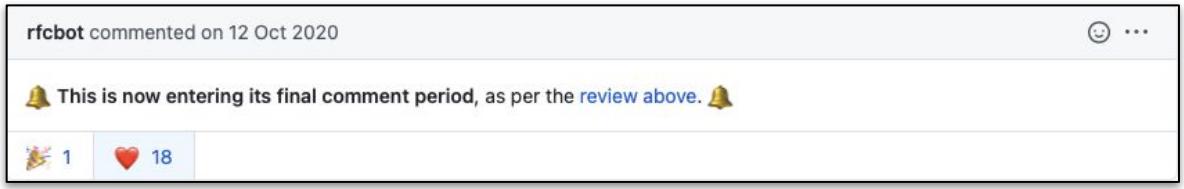
The Beginning
Discovering Value
Discovering Gaps
Building Bridges
Rust's Platform Tier Definitions
Scoping the Tier-1 Problem
Building Mindshare at Arm
Fixing Rust Compiler Tests
Enabling Silicon Access
Enabling Rust CI Loops
Creating an Arm Focus Group
Creating a GitHub Arm PingBot
The First RFC Submission
Processing Feedback
Linaro to the Rescue

Kristof Beyls
(Arm)

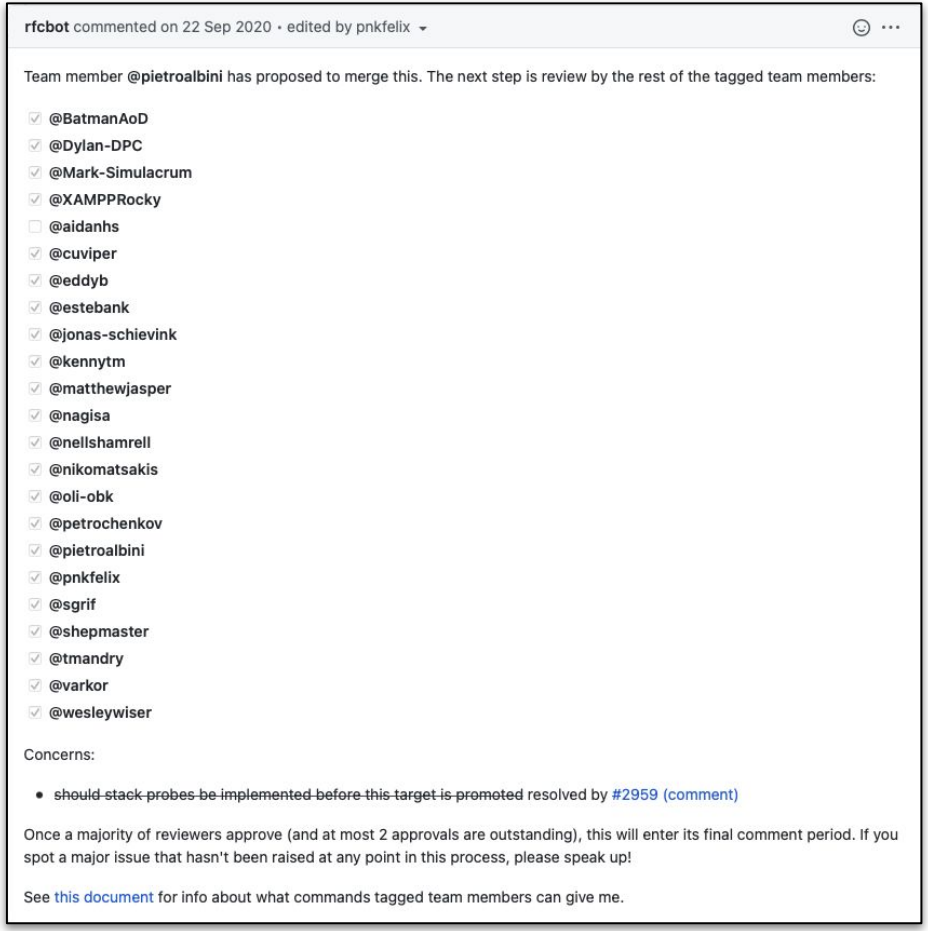
Oliver Stannard
(Linaro)

- **Credit where due:**
  - Kristof looks after LLVM things at Arm and I sought his advice
  - He put me in touch with Oliver who then proceeded to look at adding stack-probing support to LLVM for AArch64!
  - Crisis averted!

The Beginning
Discovering Value
Discovering Gaps
Building Bridges
Rust's Platform Tier Definitions
Scoping the Tier-1 Problem
Building Mindshare at Arm
Fixing Rust Compiler Tests
Enabling Silicon Access
Enabling Rust CI Loops
Creating an Arm Focus Group
Creating a GitHub Arm PingBot
The First RFC Submission
Processing Feedback
Linaro to the Rescue
The Final RFC Submission



The Beginning
Discovering Value
Discovering Gaps
Building Bridges
Rust's Platform Tier Definitions
Scoping the Tier-1 Problem
Building Mindshare at Arm
Fixing Rust Compiler Tests
Enabling Silicon Access
Enabling Rust CI Loops
Creating an Arm Focus Group
Creating a GitHub Arm PingBot
The First RFC Submission
Processing Feedback
Linaro to the Rescue
The Final RFC Submission
The Vote



rfcbot commented on 22 Sep 2020 • edited by pnkfelix

Team member **@pietroalbini** has proposed to merge this. The next step is review by the rest of the tagged team members:

- [x] **@BatmanAoD**
- [x] **@Dylan-DPC**
- [x] **@Mark-Simulacrum**
- [x] **@XAMPPRocky**
- [ ] **@aidanhs**
- [x] **@cuviper**
- [x] **@eddyb**
- [x] **@estebank**
- [x] **@jonas-schievink**
- [x] **@kennytm**
- [x] **@matthewjasper**
- [x] **@nagisa**
- [x] **@nellshamrell**
- [x] **@nikomatsakis**
- [x] **@oli-obk**
- [x] **@petrochenkov**
- [x] **@pietroalbini**
- [x] **@pnkfelix**
- [x] **@sgrif**
- [x] **@shepmaster**
- [x] **@tmandry**
- [x] **@varkor**
- [x] **@wesleywiser**

Concerns:

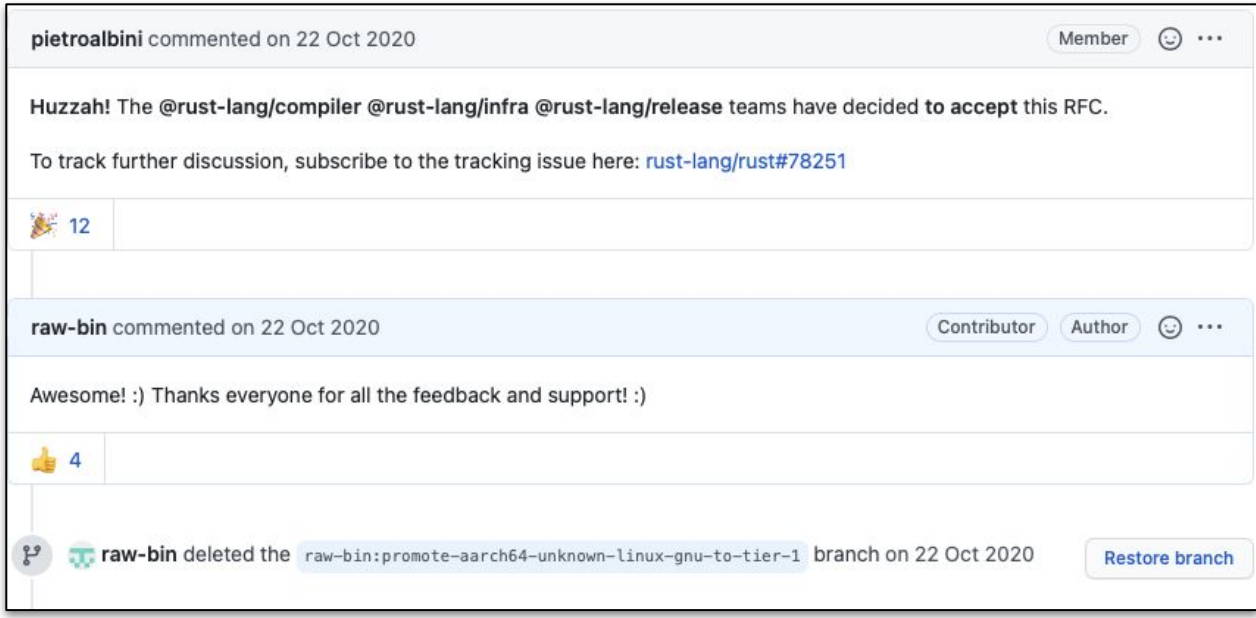
- ~~should stack probes be implemented before this target is promoted~~ resolved by #2959 (comment)

Once a majority of reviewers approve (and at most 2 approvals are outstanding), this will enter its final comment period. If you spot a major issue that hasn't been raised at any point in this process, please speak up!

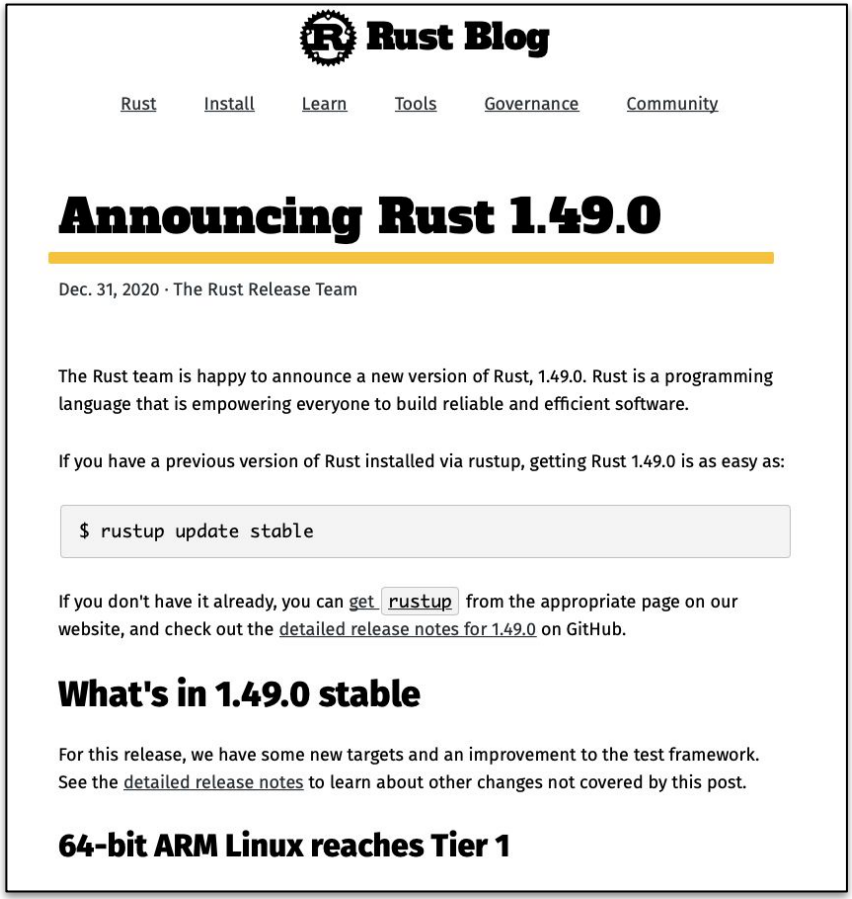
See this document for info about what commands tagged team members can give me.

- **After all the issues raised were empirically addressed, we got unanimous approval from all key stakeholders!!!**

The Beginning
Discovering Value
Discovering Gaps
Building Bridges
Rust's Platform Tier Definitions
Scoping the Tier-1 Problem
Building Mindshare at Arm
Fixing Rust Compiler Tests
Enabling Silicon Access
Enabling Rust CI Loops
Creating an Arm Focus Group
Creating a GitHub Arm PingBot
The First RFC Submission
Processing Feedback
Linaro to the Rescue
The Final RFC Submission
The Vote



The Beginning
Discovering Value
Discovering Gaps
Building Bridges
Rust's Platform Tier Definitions
Scoping the Tier-1 Problem
Building Mindshare at Arm
Fixing Rust Compiler Tests
Enabling Silicon Access
Enabling Rust CI Loops
Creating an Arm Focus Group
Creating a GitHub Arm PingBot
The First RFC Submission
Processing Feedback
Linaro to the Rescue
The Final RFC Submission
The Vote
Attaining Tier-1 Status



Rust Blog

Rust Install Learn Tools Governance Community

Announcing Rust 1.49.0

Dec. 31, 2020 · The Rust Release Team

The Rust team is happy to announce a new version of Rust, 1.49.0. Rust is a programming language that is empowering everyone to build reliable and efficient software.

If you have a previous version of Rust installed via rustup, getting Rust 1.49.0 is as easy as:

```
$ rustup update stable
```

If you don't have it already, you can get `rustup` from the appropriate page on our website, and check out the detailed release notes for 1.49.0 on GitHub.

What's in 1.49.0 stable

For this release, we have some new targets and an improvement to the test framework. See the detailed release notes to learn about other changes not covered by this post.

64-bit ARM Linux reaches Tier 1

Some final words about tiers

- Tier 2 are still high quality targets
  - Tier 2 can still be release blockers and will be shipped with every release
- Tier 1 is high assurance
  - Runs full test and quality assurance test suites for every commit
  - Engineering support really helps
- Tier 1 targets influence the quality of Tier 2 on the same platform positively
- RFC 2803: https://github.com/rust-lang/rfcs/pull/2803

The importance of ARM on Tier 1

- Rust is currently in phase of fast adoption
- Two architectures being tier1 and a number of other ones Tier 2 has a lot of importance
- ARM reaching out *way before* the fast adoption phase meant time for a proper release
- In the hot phase
  - Programming language adoption is not unidirectional
  - Companies and projects adopting Rust are aiming for well supported architectures with preference
  - Broadening architecture support shows growth in the right directions
- Rust is often used in greenfield projects, where hardware choice is free(er)

Filling the other gaps!

One ambition is to enable Rust uptake in system SW layers for Arm by working with interested parties

Cortex-A Crate

- Autogenerate safe Rust wrappers for system control register access from Arm ISA XML specs
- Cortex-R adaptation

MMU Crate

- Rust traits for architectural configuration of MMUs and for basic page-to-frame attribute expression + mapping schemes

GIC Crate

- Rust traits for architectural configuration of GICs and for basic IRQ-to-handler attribute expression + mapping schemes

DeviceTree Crate

- Safe parsing and manipulation of standards compliant Device-Tree blobs

PSCI Crate

- Rust traits for PSCI services
- Enables different back-end implementations in Rust (bare-metal/{RT}OS hosted)

POSIX Clib Crate

- Rust traits for POSIX APIs
- Enables different and back-end implementations in Rust (bare-metal/{RT}OS hosted)
- C linkage via Rust FFI

Firmware

- PSA-FF-A compliant firmware implementation in Rust
- PSA-FF-M

Bootloader

- EBBR compliant bootloader implementation in Rust

Hypervisor

- Type-1 static partitioning hypervisor in Rust
- Misc Type-1 Hypervisor Enablement via rust-vmm

Architecture support

- Rust support for Advanced NEON/SIMD intrinsics
- SVE
- Future safety and security centric Arm architecture extensions

Ferrocene

- A concrete path to making Rust viable for ISO2626, ISO61508
- Led by Ferrous Systems GmbH

Tier-1 maintenance

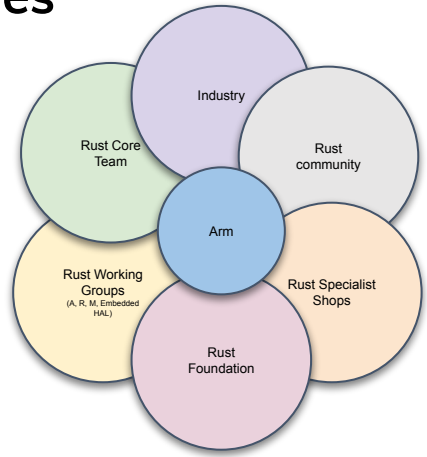
- CI
- Triage
- Fix
- Rust compiler test suite extensions/cleanups

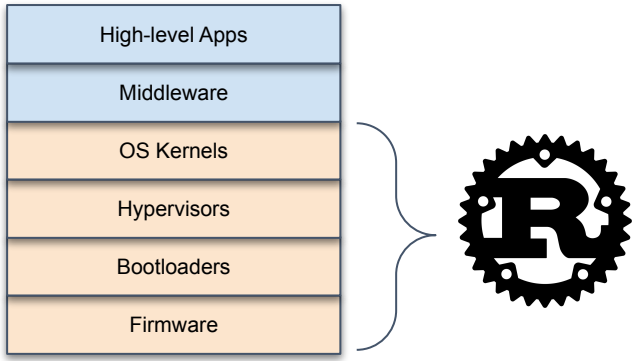
Other AArch64 Triples

- AArch64 macOS
- AArch64 FreeBSD
- AArch64 Windows

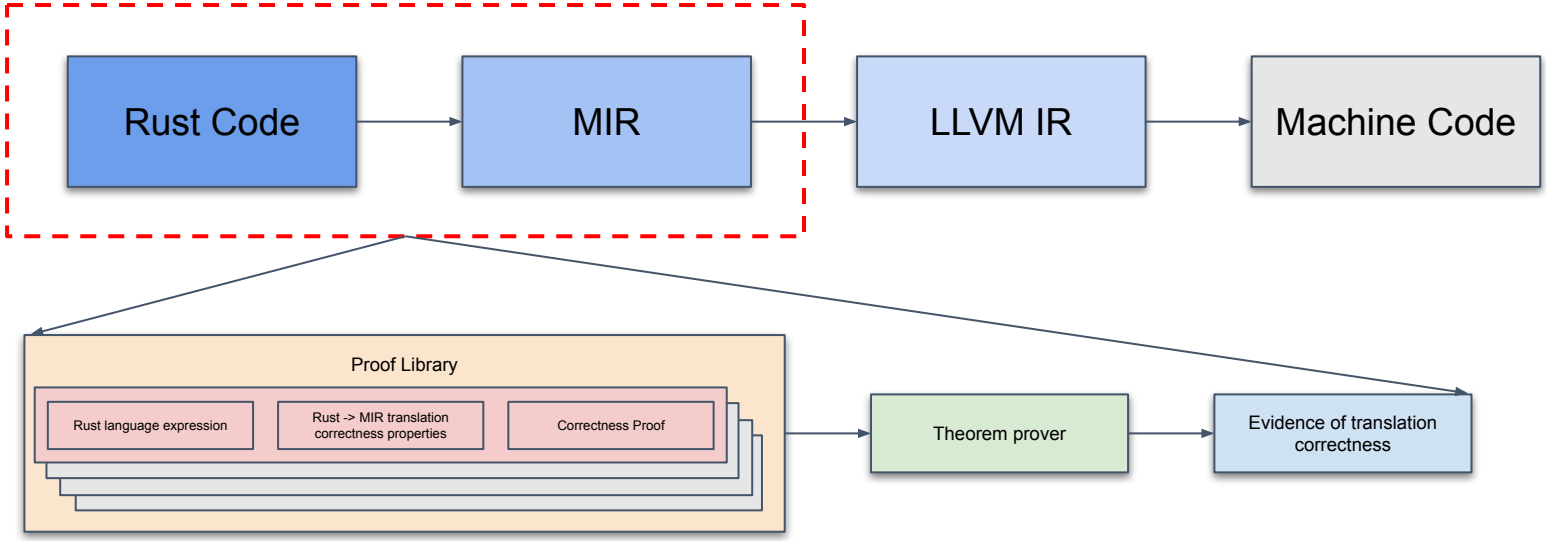
Rust for Linux Drivers

- Rust wrappers to Kernel Driver APIs





The other ambition is to enable Rust's path to ISO26262/ISO61508 deployments using Formal Methods

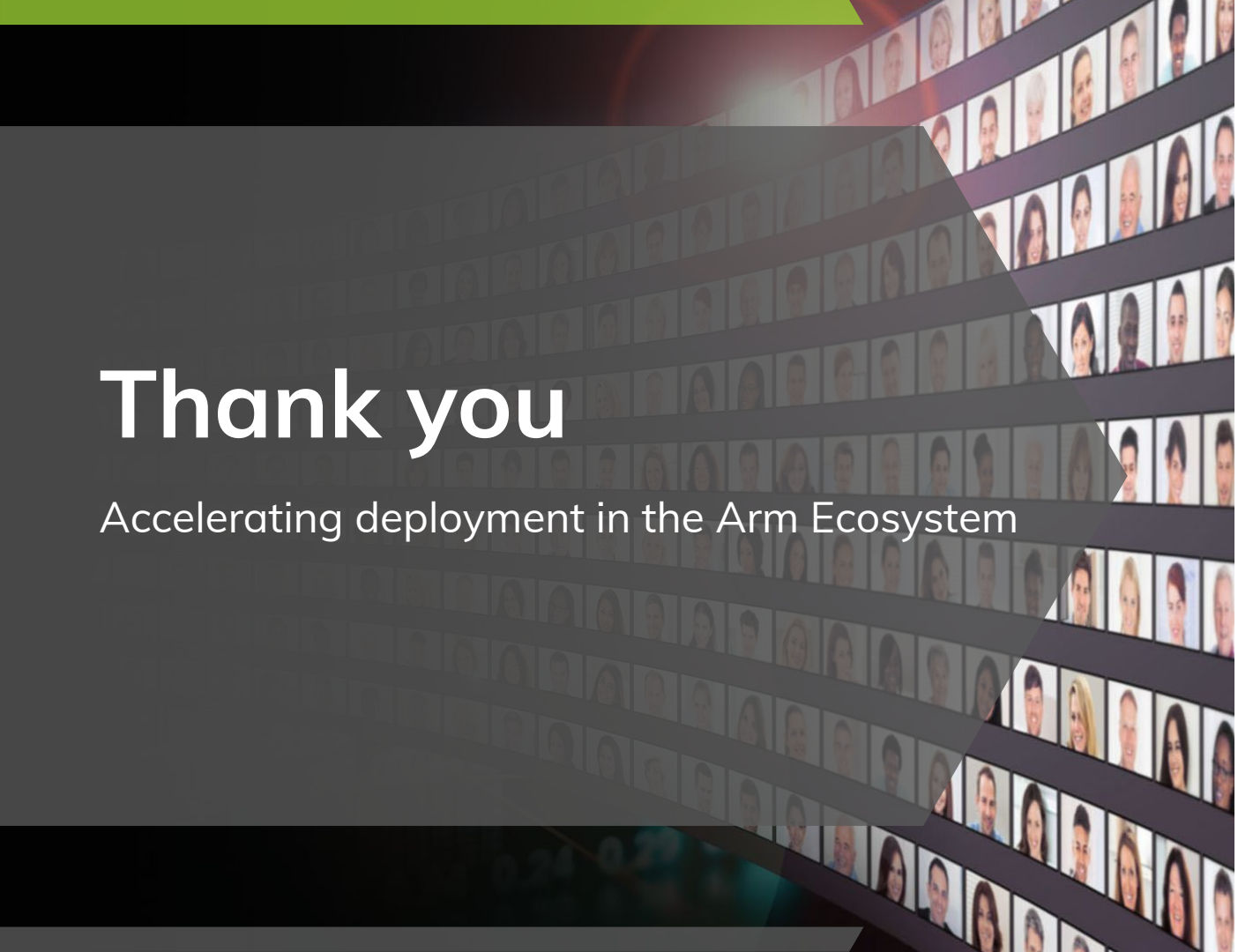


ferrocene

- Increase trust in the Rust compiler
- Model the correctness of the translation of Rust code to MIR using formal methods
  - Correctness properties expressed as higher order formal proofs
  - Modern techniques using theorem solvers (Coq et al) are used to verify the proofs
  - The proofs act as a runnable language description and a test suite
  - This keeps the focus on the language without getting into toolchain specifics

Outlook / Future

- Libraries and bindings
- Tooling
  - https://knurling.ferrous-systems.com/
  - https://probe.rs/
- Support for safety-critical environments
  - Qualification of compilers and libraries (e.g. ISO 26262)
  - Review and safety guidelines
  - Tooling for safety-critical industries
  - Long Term Support services
  - https://ferrous-systems.com/ferrocene



Thank you

Accelerating deployment in the Arm Ecosystem

Linaro Connect
VIRTUAL • SPRING 2021