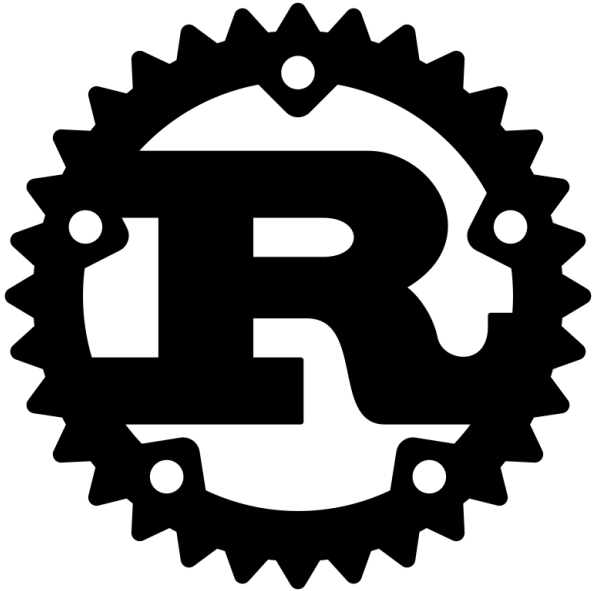
Rust ❤️ LLVM

Nikita Popov

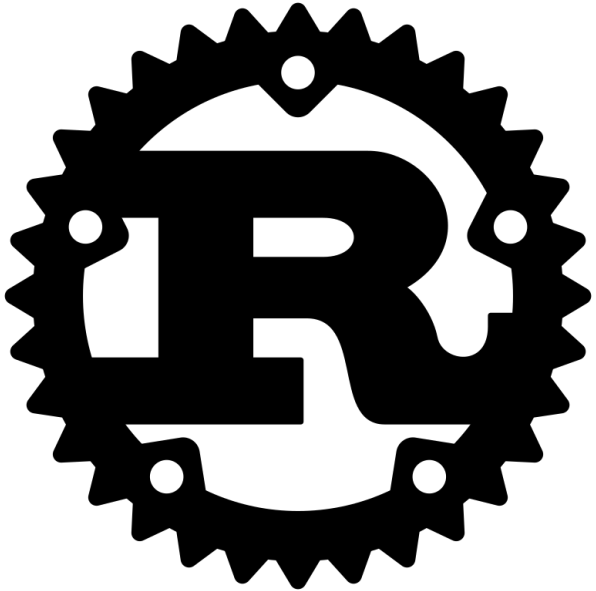
LLVM Developers Meeting 2024

Rust: Memory safe systems programming language



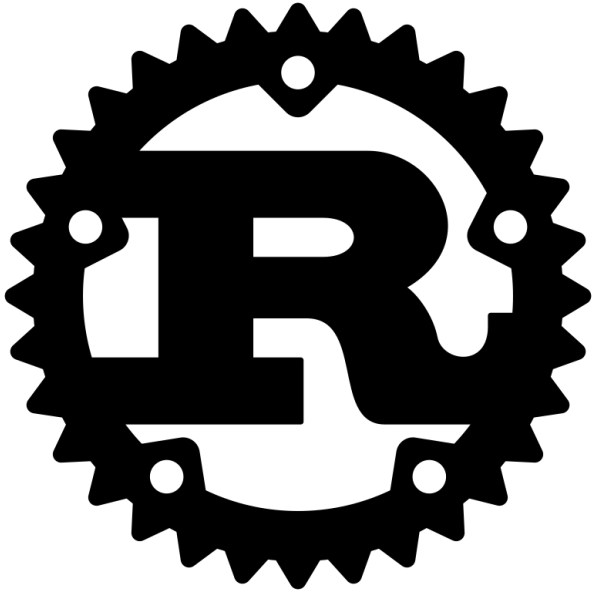
Rust: Memory safe systems programming language

safe



Rust: Memory safe systems programming language

safe



fast

70% of security bugs are memory safety issues

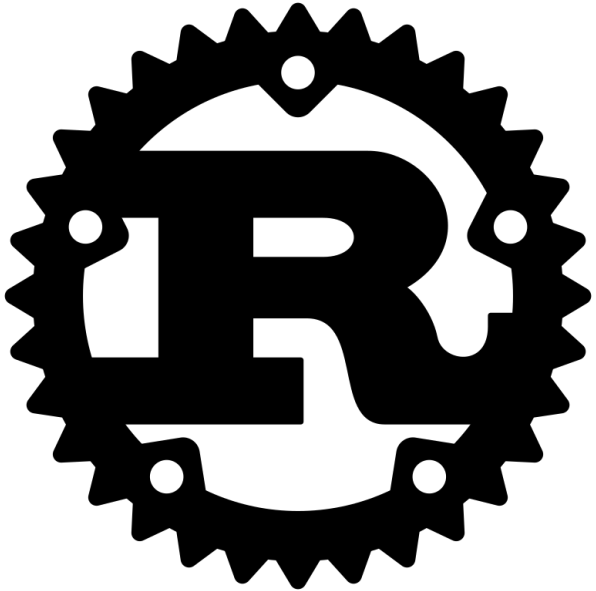
https://www.chromium.org/Home/chromium-security/memory-safety/
https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/

Rust: Memory safe systems programming language

safe

↑

rustc

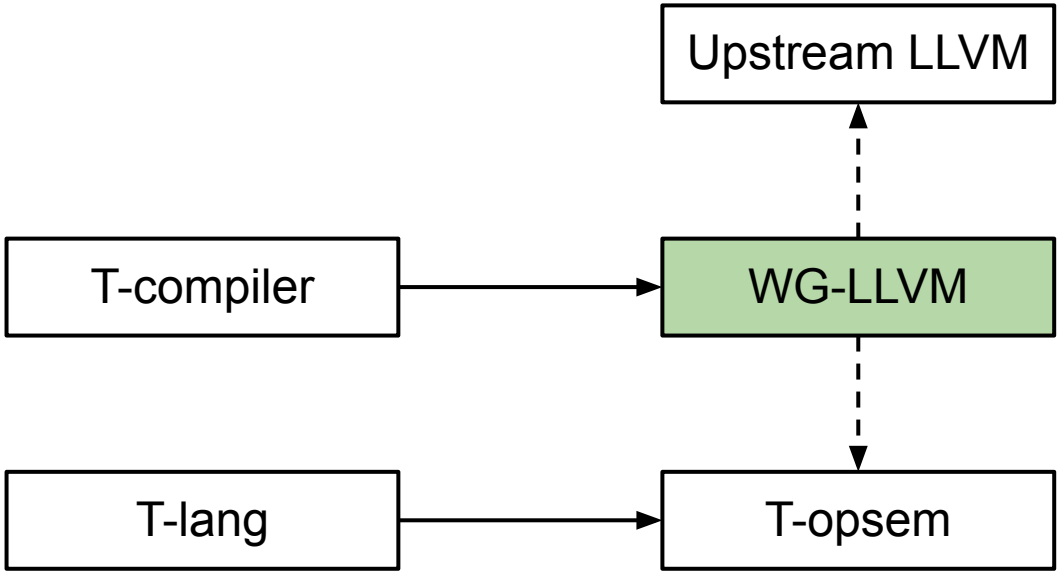


fast

↑

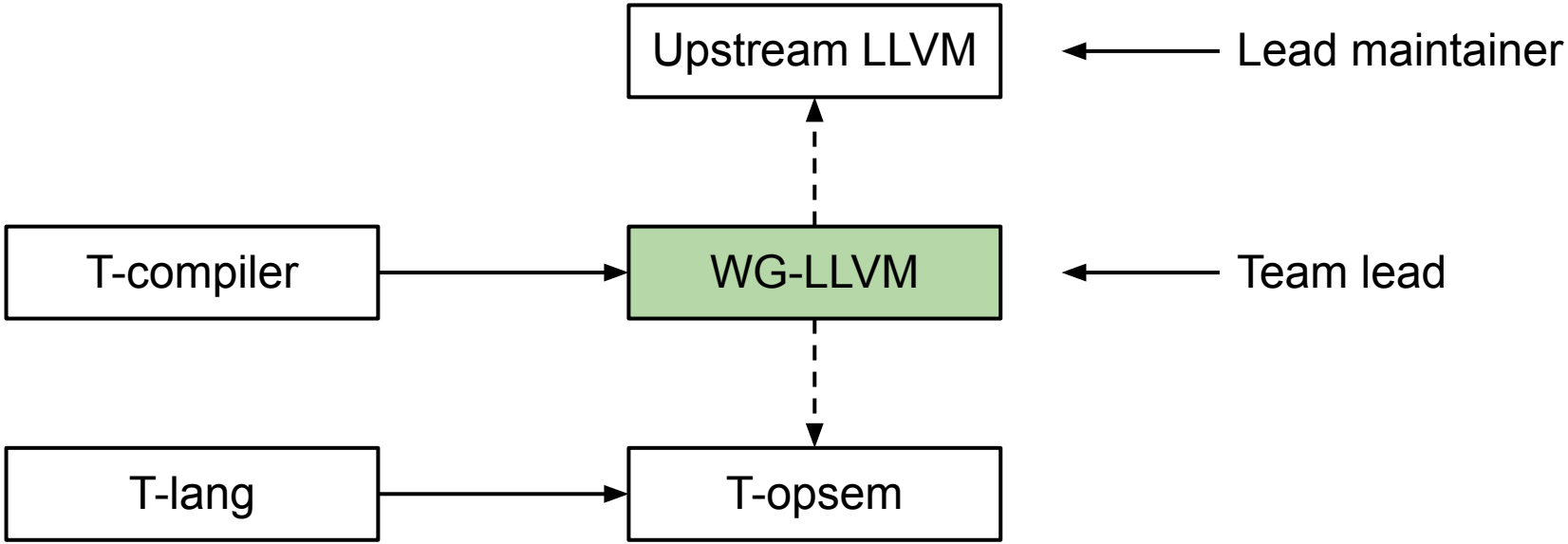
LLVM

Governance



T = Team, WG = Working Group

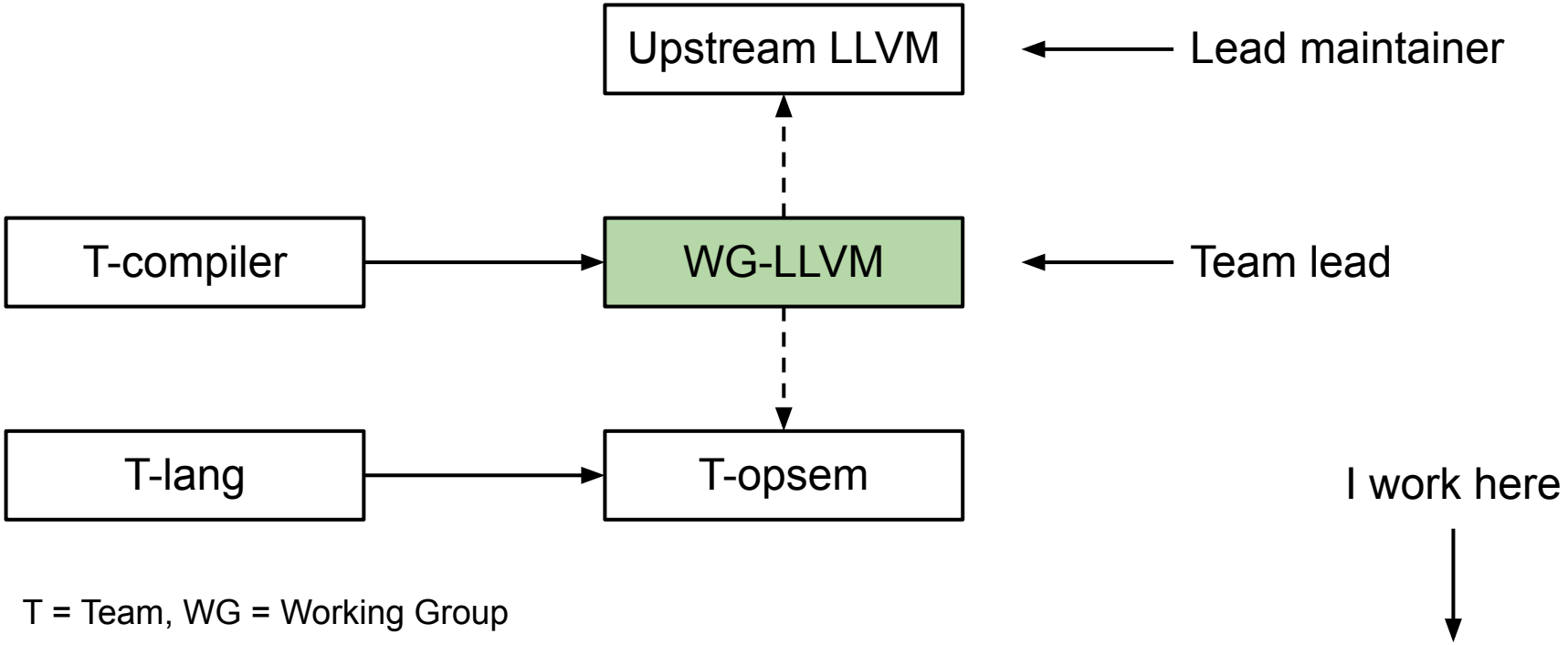
Governance



T = Team, WG = Working Group

Red Hat

Governance



T = Team, WG = Working Group

Red Hat

Supported LLVM versions

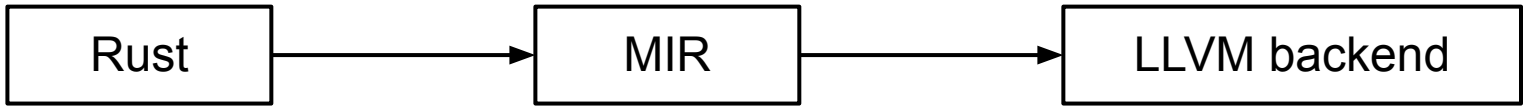
- Current LLVM main (20-dev)
- Current LLVM release (19)
  - Default, used by official rustup binaries
- One or two older LLVM releases (18, 17)
  - For distros

Supported LLVM versions

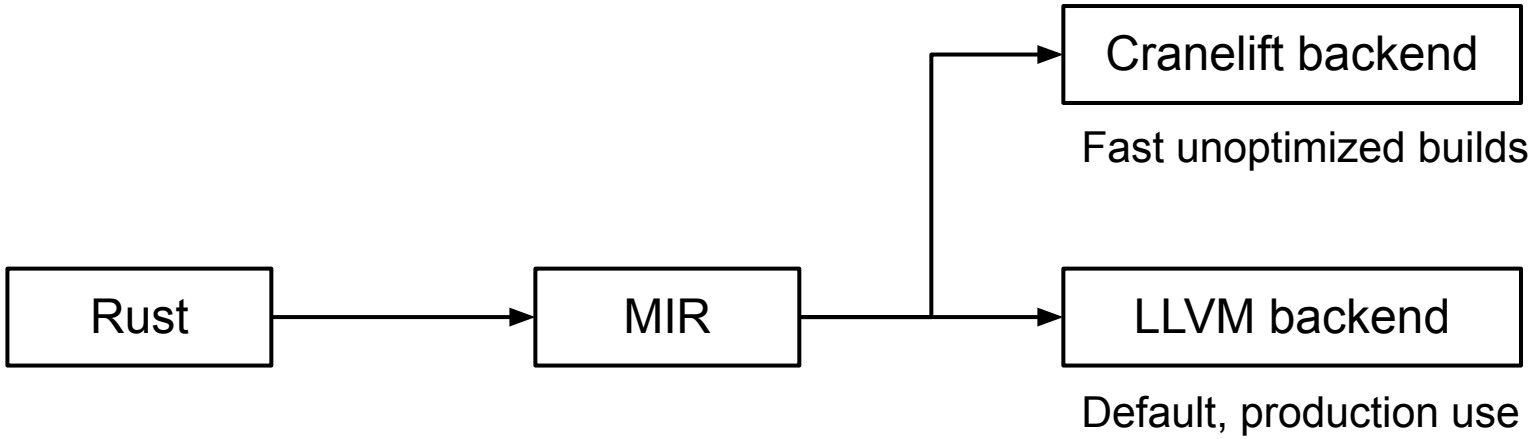
- Current LLVM main (20-dev)
- Current LLVM release (19)
  - Default, used by official rustup binaries
- One or two older LLVM releases (18, 17)
  - For distros

- LLVM fork for backport management only, no Rust-specific patches

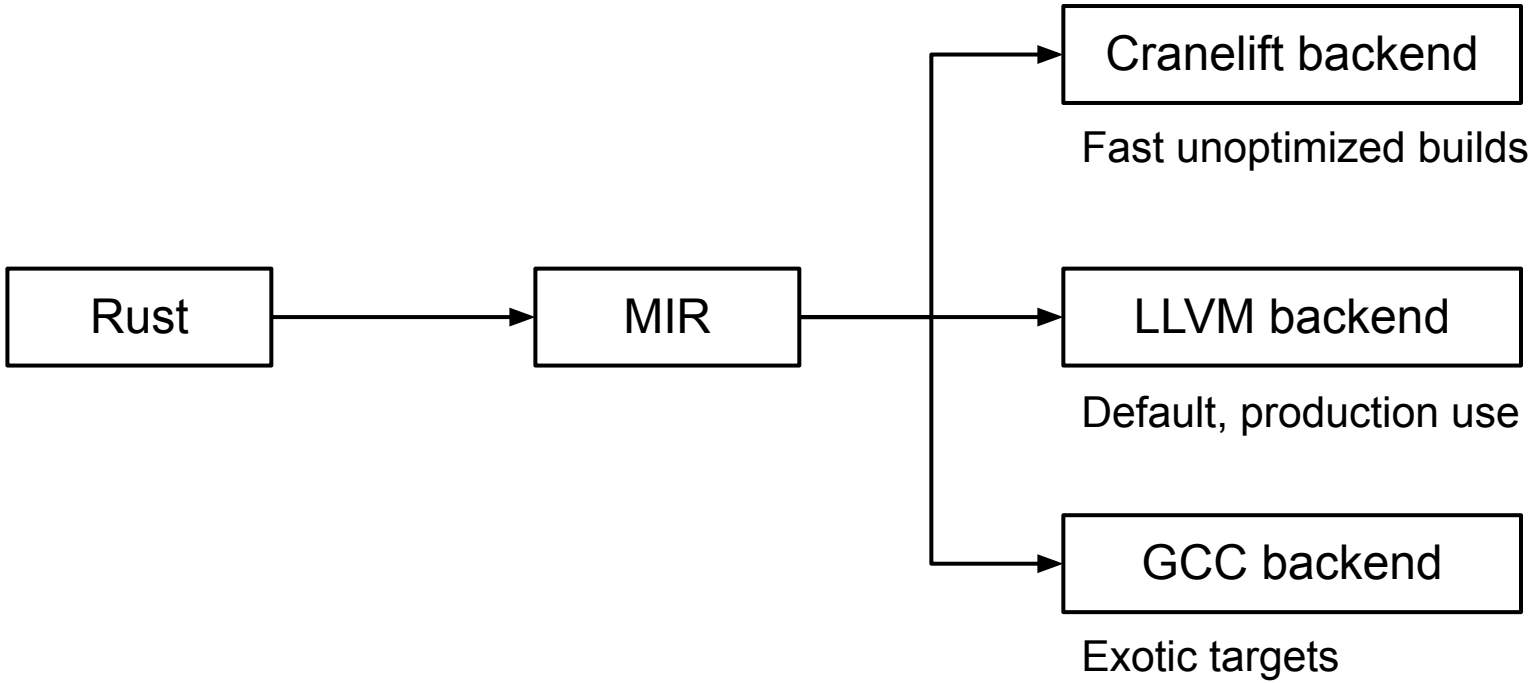
Rust Lowering



Rust Lowering

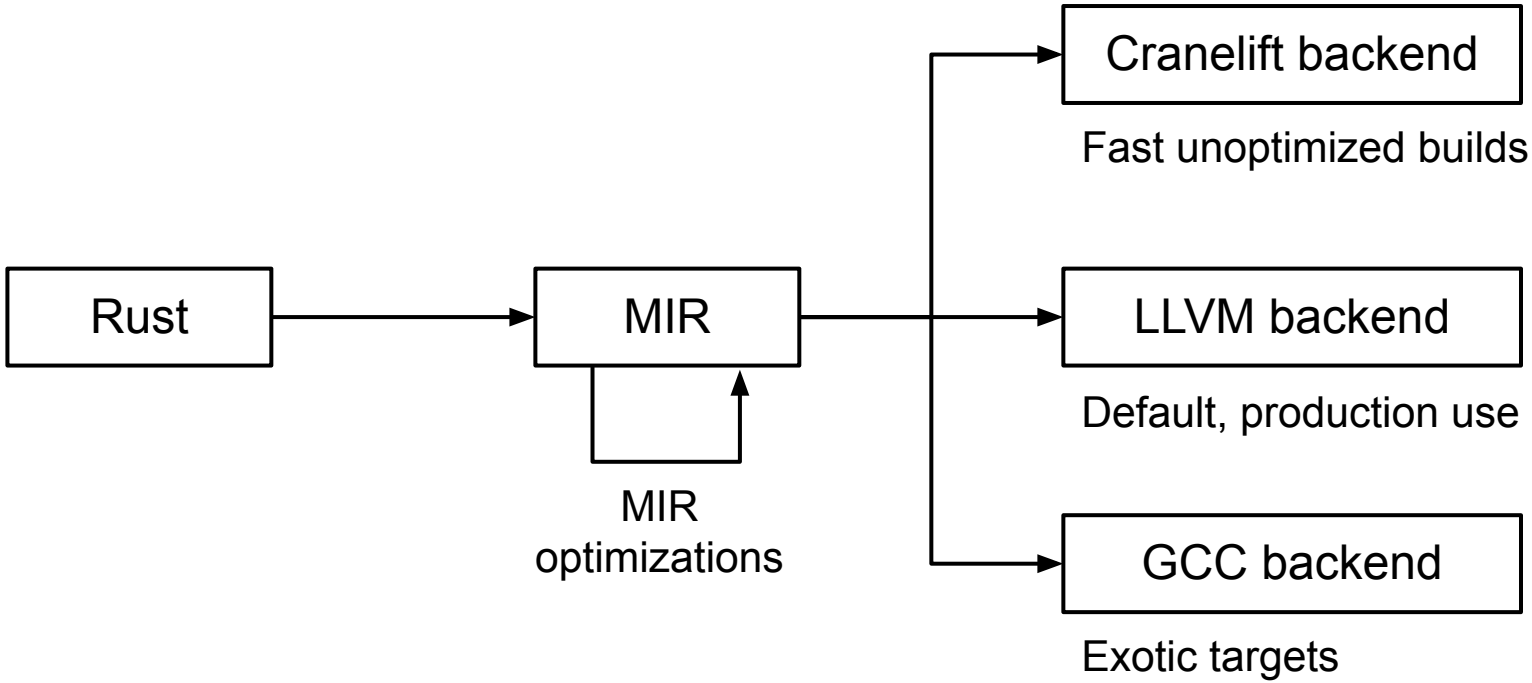


Rust Lowering



Red Hat

Rust Lowering



Challenges

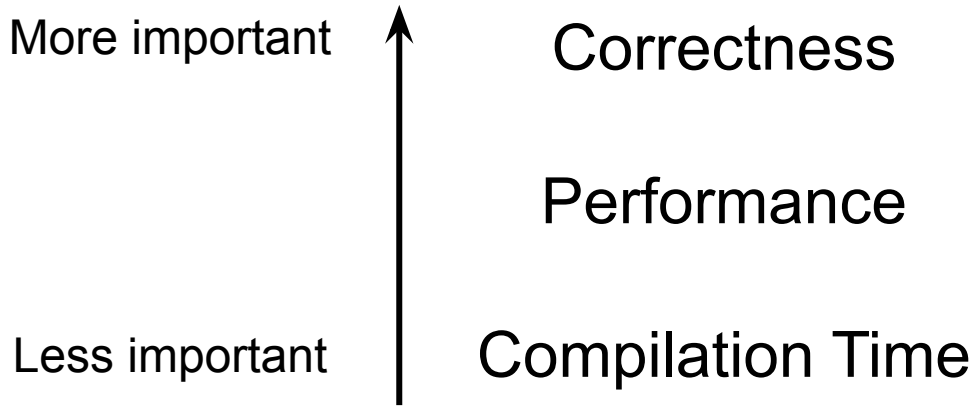
Challenges

Correctness

Performance

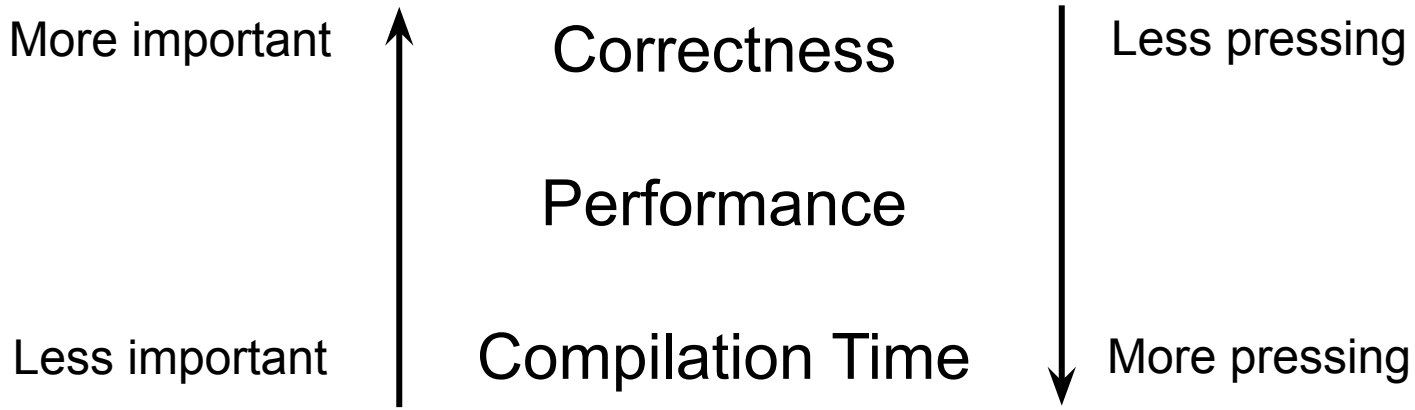
Compilation Time

Challenges



Red Hat

Challenges



Compilation Time

Compilation Time

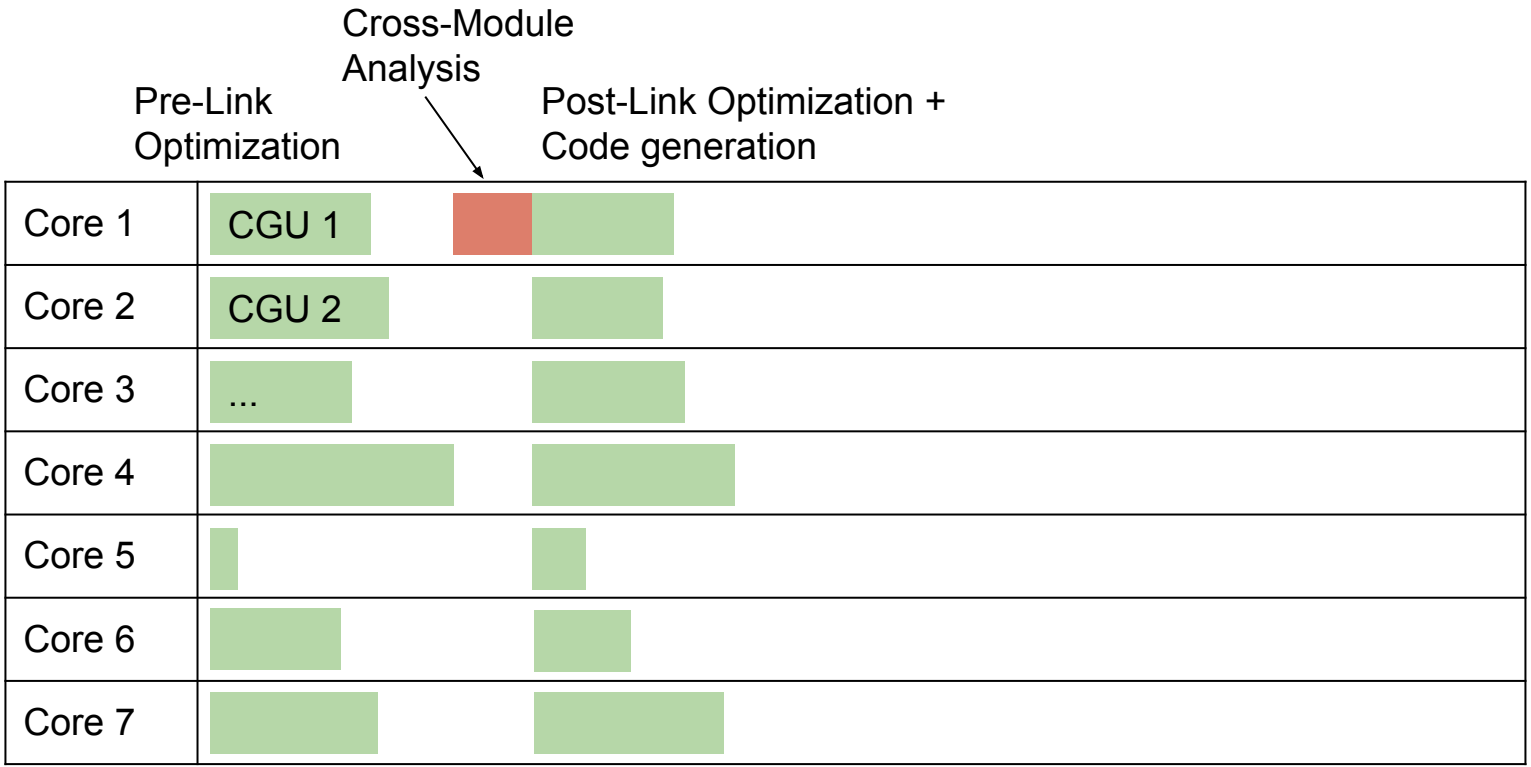
- Compilation unit is crate (vs file in C/C++)
  - Mitigation: CGU partitioning + Crate-local ThinLTO

Compilation Time

- Compilation unit is crate (vs file in C/C++)
  - Mitigation: CGU partitioning + Crate-local ThinLTO

Core 1	Whole crate
Core 2	
Core 3	
Core 4	
Core 5	
Core 6	
Core 7	

CGU partitioning and Crate-Local ThinLTO

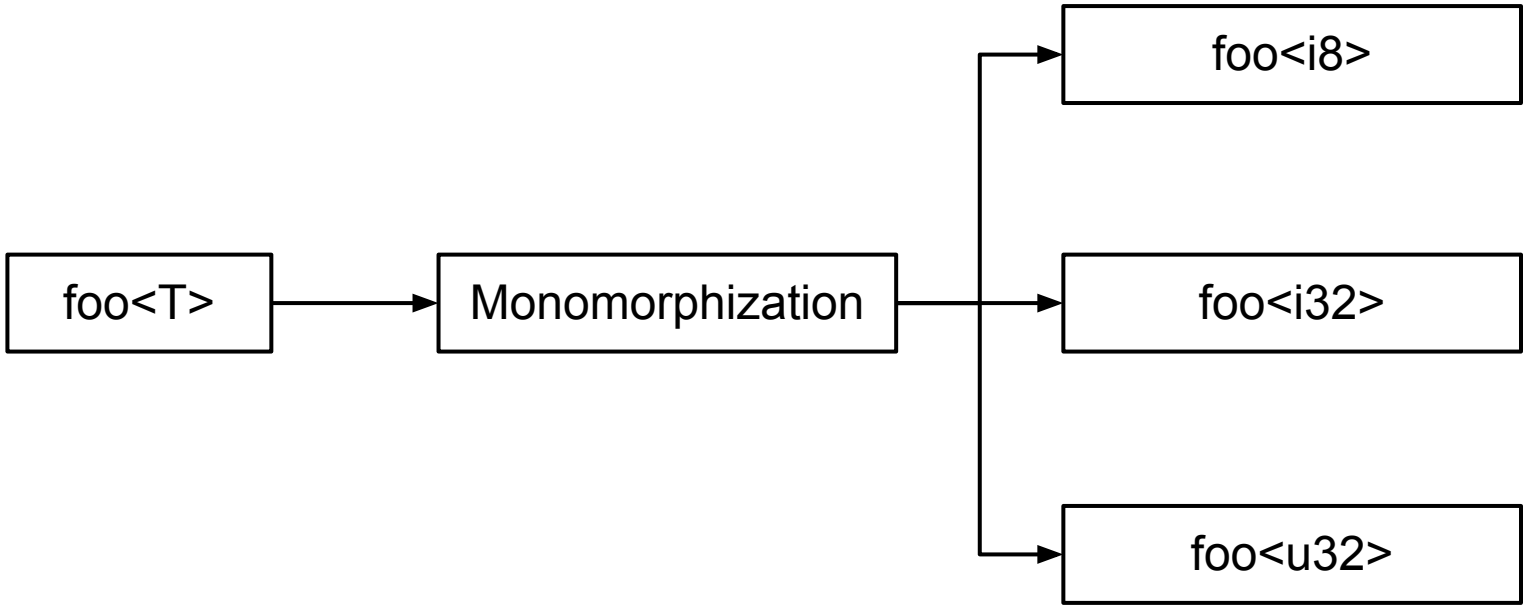


Red Hat

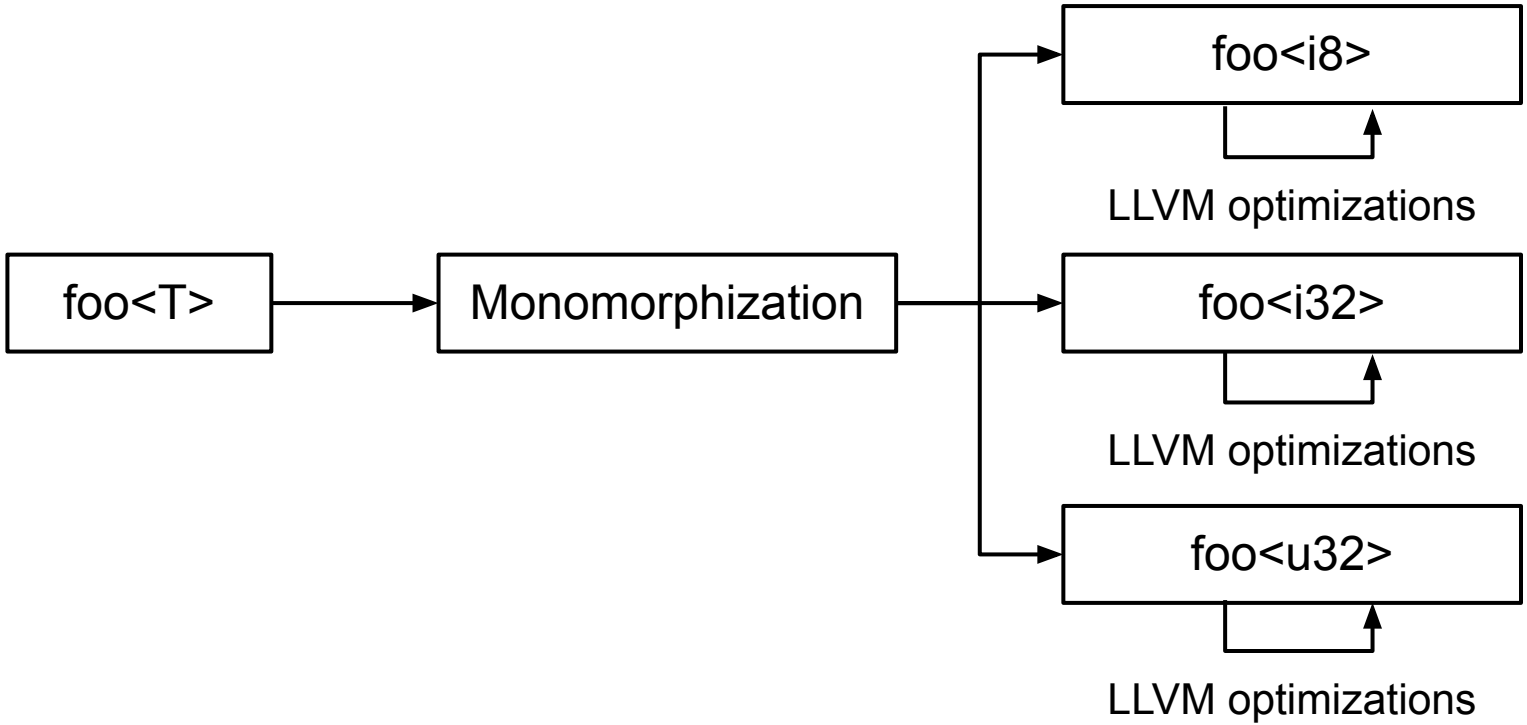
Compilation Time

- Compilation unit is crate (vs file in C/C++)
  - Mitigation: CGU partitioning + Crate-local ThinLTO
- Generics produce huge amounts of IR
  - Mitigation: MIR optimization, share-generics, polymorphization(?)

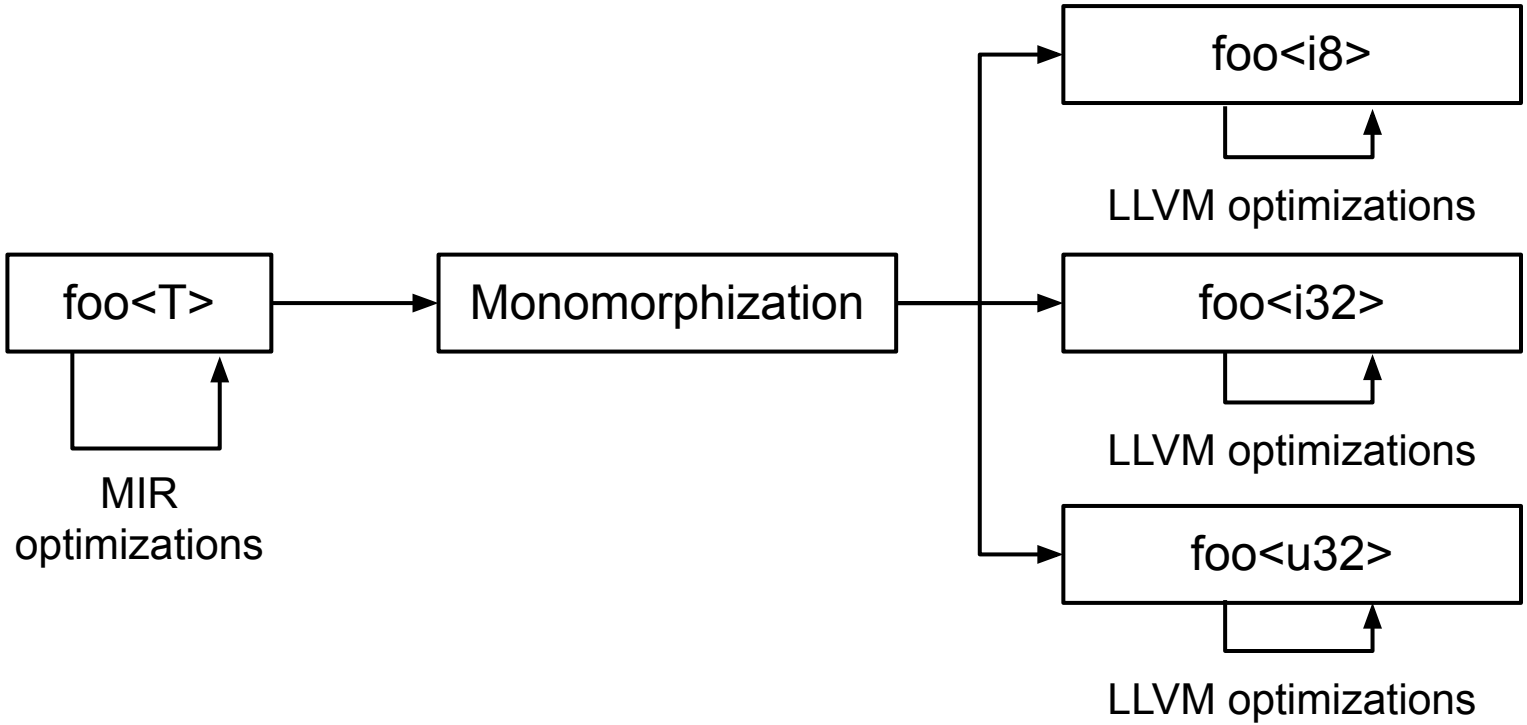
Monomorphization



Monomorphization



Monomorphization



Compilation Time

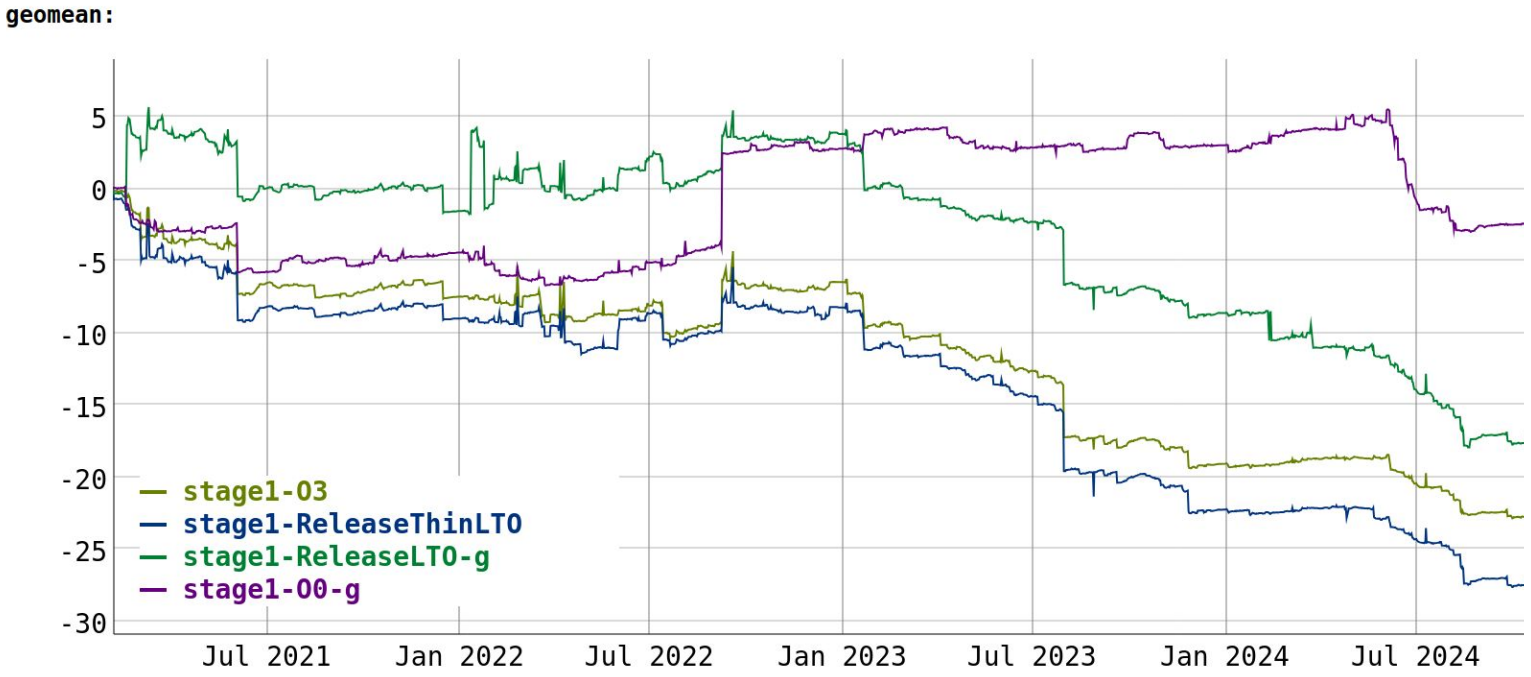
- Compilation unit is crate (vs file in C/C++)
  - Mitigation: CGU partitioning + Crate-local ThinLTO
- Generics produce huge amounts of IR
  - Mitigation: MIR optimization, share-generics, polymorphization(?)
- LLVM is slow
  - Mitigation: Make LLVM faster

LLVM 10 upgrade

▶	clap-rs	opt	full	llvm	16.26%
▶	clap-rs	opt	incr-patched: println	llvm	16.03%
▶	syn	opt	incr-patched: println	llvm	15.35%
▶	clap-rs	opt	incr-full	llvm	14.36%
▶	cargo	opt	incr-patched: println	llvm	13.88%
▶	syn	opt	incr-full	llvm	13.82%
▶	regex	opt	full	llvm	12.71%
▶	cargo	opt	incr-full	llvm	12.63%
▶	regex	opt	incr-patched: sparse set	llvm	12.57%
▶	regex	opt	incr-patched: Job	llvm	12.53%
▶	regex	opt	incr-patched: compile one	llvm	12.53%
▶	regex	opt	incr-patched: Compiler new	llvm	12.51%
▶	regex	opt	incr-patched: reverse	llvm	12.49%
▶	tokio-webpush-simple	opt	incr-patched: minor change	llvm	12.31%

https://perf.rust-lang.org/compare.html?start=c5840f9d252c2f5cc16698dbf385a29c5de3ca07&end=97588aeda139309169b11654fc809e1ac5fd246c

LLVM compilation time tracker



https://llvm-compile-time-tracker.com/graphs.php?startDate=2021-02-04&interval=100&relative=on&bench=geomean&width=800

Red Hat

LLVM 19 upgrade

▶	regex-1.5.5	debug	incr-patched: Job	llvm	-15.78%
▶	regex-1.5.5	debug	full	llvm	-15.12%
▶	regex-1.5.5	opt	full	llvm	-13.80%
▶	regex-1.5.5	debug	incr-full	llvm	-13.77%
▶	cargo-0.60.0	debug	full	llvm	-13.53%
▶	webrender-2022	opt	full	llvm	-13.51%
▶	regex-1.5.5	opt	incr-patched: Job	llvm	-12.99%
▶	webrender-2022	opt	incr-full	llvm	-12.73%
▶	image-0.24.1	debug	full	llvm	-12.53%
▶	regex-1.5.5	opt	incr-full	llvm	-12.39%
▶	ripgrep-13.0.0	debug	full	llvm	-12.15%
▶	cargo-0.60.0	debug	incr-full	llvm	-12.05%
▶	webrender-2022	debug	full	llvm	-11.82%
▶	image-0.24.1	debug	incr-full	llvm	-11.32%

https://perf.rust-lang.org/compare.html?start=e552c168c72c95dc28950a9aae8ed7030199aa0d&end=0b5eb7ba7bd796fb39c8bb6acd9ef6c140f28b65

Performance

Specific optimization problems

- Bounds check elimination
  - Usually works, but not reliable
  - Good: Constant bounds, straight-line code. Bad: Checks in loops.

Specific optimization problems

- Bounds check elimination
  - Usually works, but not reliable
  - Good: Constant bounds, straight-line code. Bad: Checks in loops.
- memcpy elimination
  - Rust has no NRVO -> many memcpys

Specific optimization problems

- Bounds check elimination
  - Usually works, but not reliable
  - Good: Constant bounds, straight-line code. Bad: Checks in loops.
- memcpy elimination
  - Rust has no NRVO -> many memcpys
- Inclusive ranges
  - `0..n` often optimized much better than `0..=n`
  - Conditional increment to handle `n == u32::MAX` correctly

Performance: Telling LLVM about Rust semantics

- Rust has many very strong guarantees
- Conveyed to LLVM using attributes, metadata and assumes
  - noalias, readonly, dereferenceable, nonnull, range, etc.

Performance: Telling LLVM about Rust semantics

- Rust has many very strong guarantees
- Conveyed to LLVM using attributes, metadata and assumes
  - noalias, readonly, dereferenceable, nonnull, range, etc.
- Problem: Metadata/attributes get lost. Assumes don't get lost enough.

Attributes motivated by Rust needs

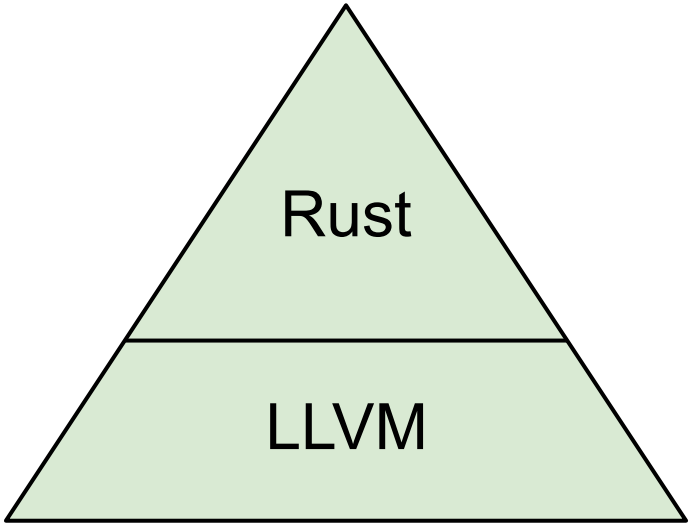
- Allocator attributes
  - Teach LLVM about Rust's custom allocation functions
- Range attributes
  - Previously only available as load metadata. Can now annotate function args.
- Dead_on_unwind, writable
  - Allow more memcpy optimization
- Getelementptr nuw
  - Let LLVM know the array index is not negative

Attributes motivated by Rust needs

- Allocator attributes
  - Teach LLVM about Rust's custom allocation functions
- Range attributes
  - Previously only available as load metadata. Can now annotate function args.
- Dead_on_unwind, writable
  - Allow more memcpy optimization
- Getelementptr nuw
  - Let LLVM know the array index is not negative
- All of these benefit C++ and other languages as well

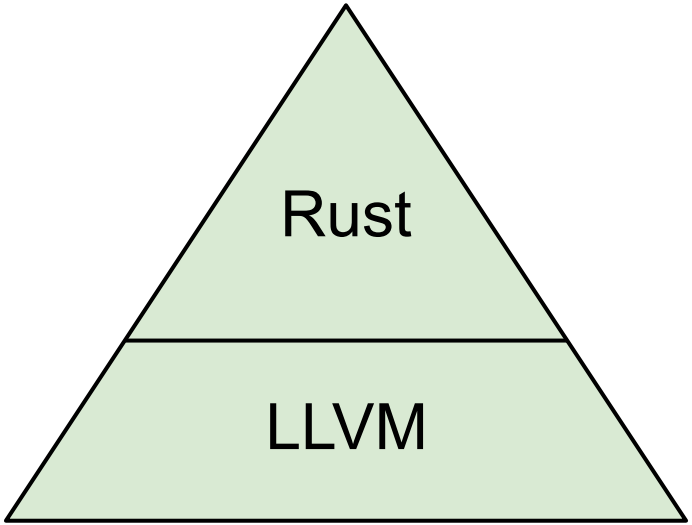
Correctness

Rust semantics based on LLVM semantics



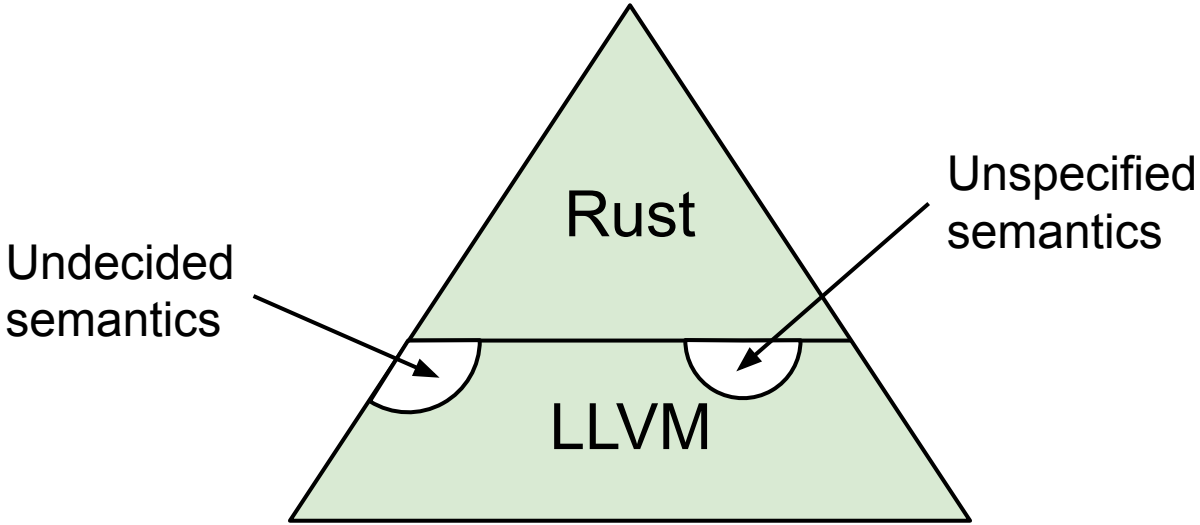
Rust semantics based on LLVM semantics

- Rust can only pick semantics that are supported by LLVM
  - ...and have good optimization support



Rust semantics based on LLVM semantics

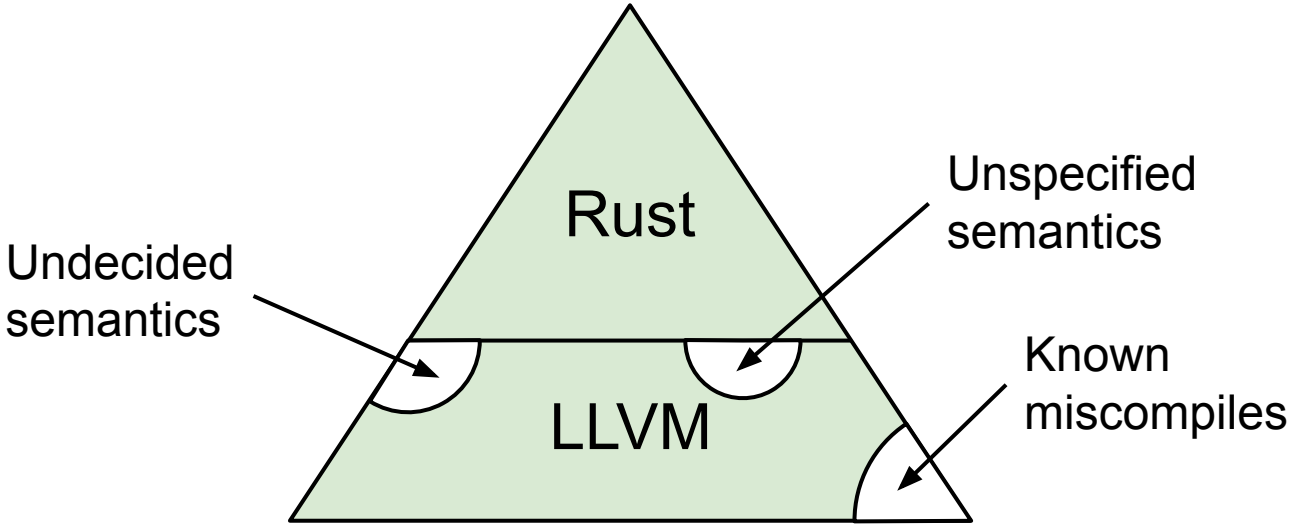
- Rust can only pick semantics that are supported by LLVM
  - ...and have good optimization support



Red Hat

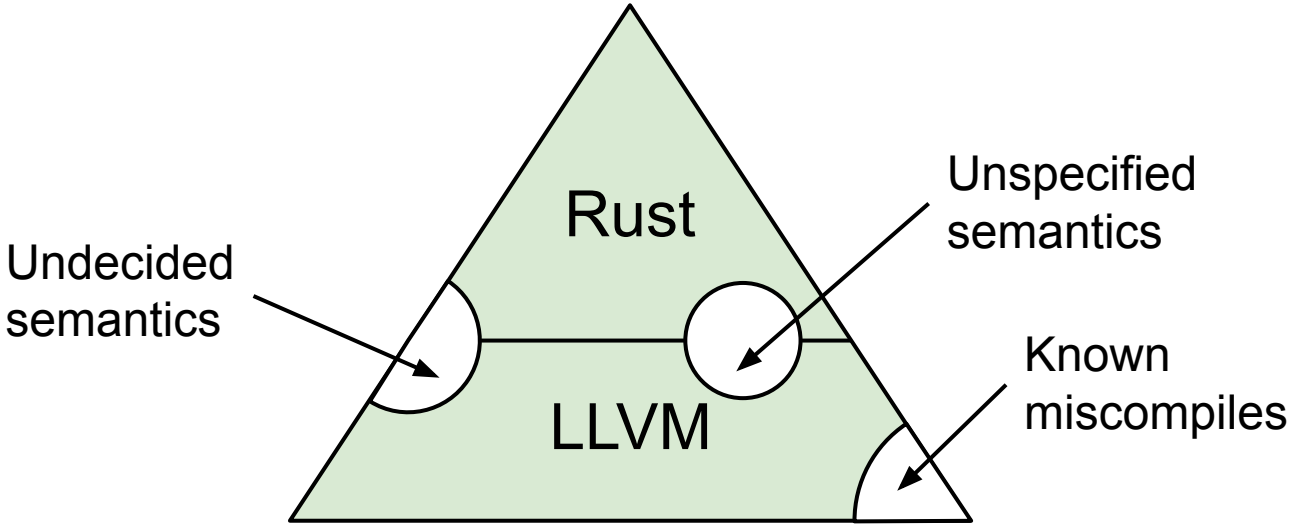
Rust semantics based on LLVM semantics

- Rust can only pick semantics that are supported by LLVM
  - ...and have good optimization support



Rust semantics based on LLVM semantics

- Rust can only pick semantics that are supported by LLVM
  - ...and have good optimization support



Historical issues

- Side-effect-free infinite loop UB
  - Opt-in via mustprogress

Historical issues

- Side-effect-free infinite loop UB
  - Opt-in via mustprogress
- noalias (`restrict` C)
  - restrict rarely used in C, ubiquitous in Rust

Call ABI

- How values are passed/returned

Call ABI

- How values are passed/returned
- C ABI handling must be implemented in each frontend
  - LLVM type system not enough for ABI handling
  - Must reimplement ~10k lines of clang TargetInfo

Call ABI

- How values are passed/returned
- C ABI handling must be implemented in each frontend
  - LLVM type system not enough for ABI handling
  - Must reimplement ~10k lines of clang TargetInfo
- LLVM confuses "available instruction sets" and "call ABI"
  - Target features like `+avx` affect both

Off the beaten path

- Rust is currently adding f16 and f128 types
- Beyond X86/ARM: Lots of bugs
- Backend often "wrong by default" instead of "correct by default"

Rust  LLVM

Thank You!

Questions?