Rust move optimization on LLVM IR

khei4 @ Kernel/VM探検隊 No16

whoami

☀️LLVM GSoC’23 contributor

- Addressing Rust optimization failures in LLVM
  - Enhance/Accelerate LLVM middle-end(IR optimization) through out Rust issues on Github.

→ Today, I’ll talk about memcpy optimization related to Rust 😀

to talk & not to talk

To talk

🔅Current Rust LLVM codegen around move

🔅**LLVM IR** optimizations to optimize Rust move
I recently related in MemCpyOpt

Not to talk 😭

🔅Rust codegen source level behavior

🔅MLIR, MIR optimizations 😭

🔅LLVM IR Optimization except MemCpyOpt

🔅Optimization performance evaluation 😲

→This is in progress 😀

Outline

- LLVM Middle-end Background
- Rust Move codegen (rustc_llvm)
- Optimizations in MemCpyOpt related to Rust Move

Outline

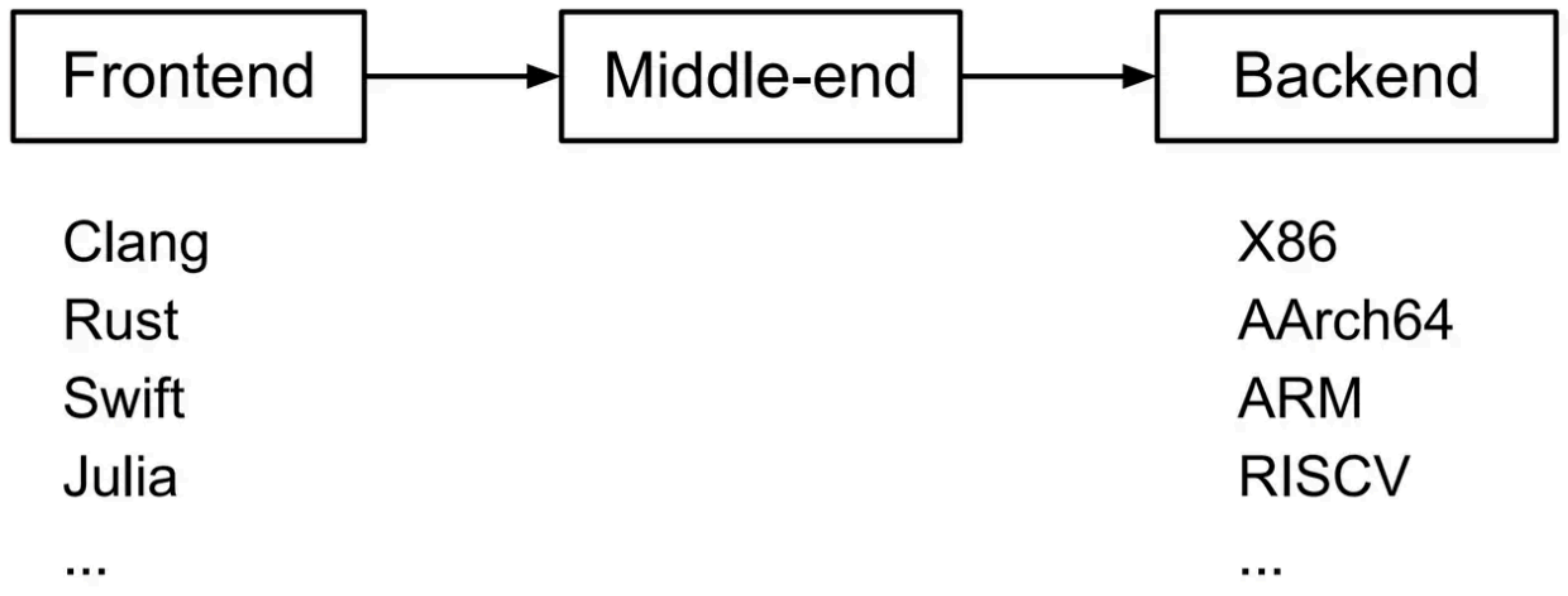
- LLVM Middle-end Background
- Rust Move codegen (rustc_llvm)
- Optimizations in MemCpyOpt related to Rust Move

LLVM IR in 10 seconds

```
define void @basic_memcpy() {
  %src = alloca %struct.Foo, align 4
  %dest = alloca %struct.Foo, align 4
  store %struct.Foo { i32 10, i32 20, i32 30 }, ptr %src

  call void @llvm.memcpy.p0.p0.i64(ptr align 4 %dest,
                                            ptr align 4 %src, i64 12, i1 false)

  ret void
}
```

☀ SSA allows easy API for compiler(optimizer) developer

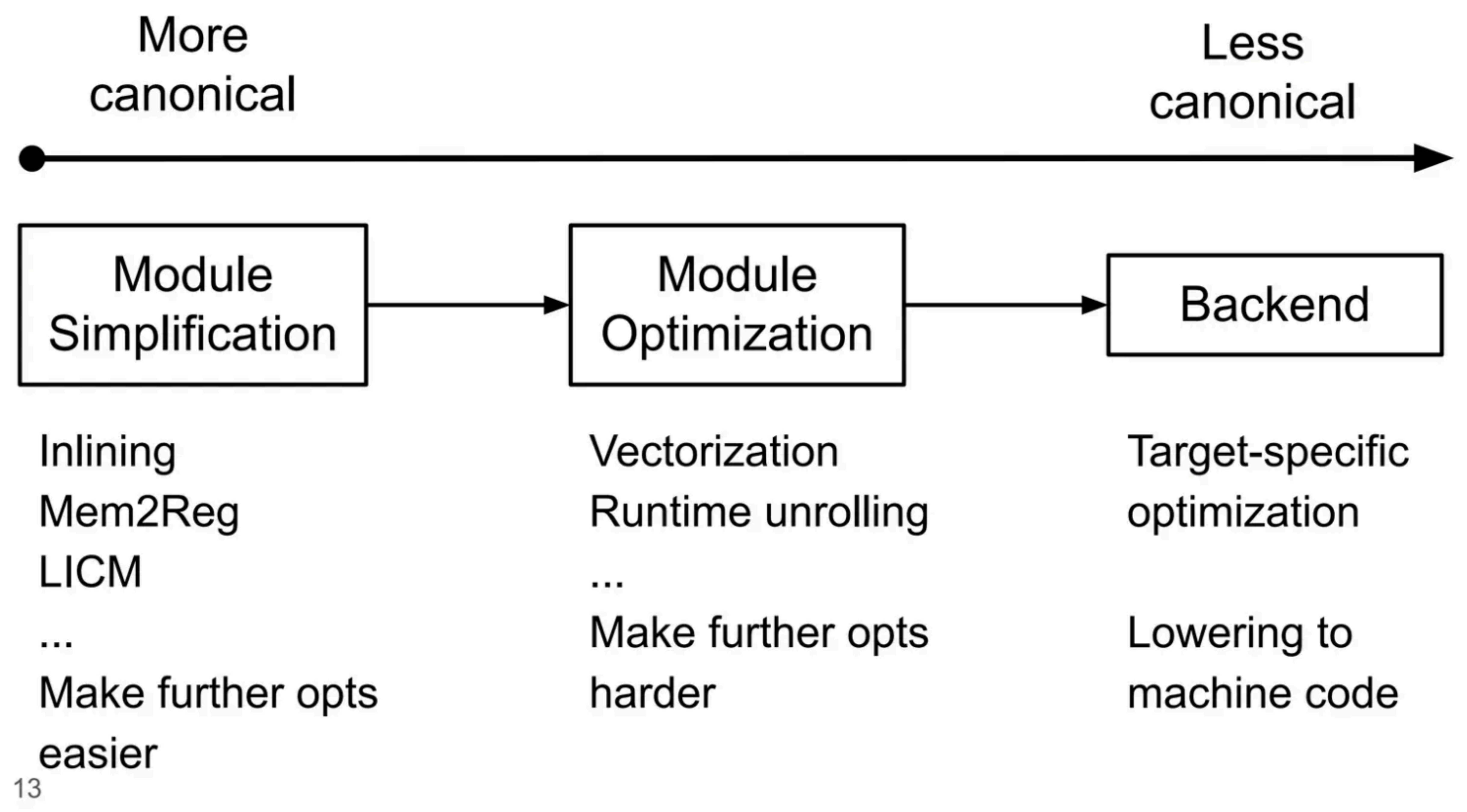
LLVM IR optimization pipeline refs



From 2023 EuroLLVM - Tutorial: A whirlwind tour of the LLVM optimizer by Nikita Popov (slides)

LLVM IR optimization pipeline

- MemCpyOpt is in “Function Simplification” ⊂ “Module Simplification”
- PassBuilderPipelines source contains all pass order information.



From 2023 EuroLLVM - Tutorial: A whirlwind tour of the LLVM optimizer by Nikita Popov (slides)

Outline

- LLVM Middle-end Background
- Rust Move codegen (rustc_llvm)
- Optimizations in MemCpyOpt related to Rust Move

Rust and move codegen

- Move (of ownership) happens when
  1. Rebind other var, 2. Pass function by-value

- Move directly corresponds to llvm memcpy intrinsic on rustc_llvm. Basically

move codegen today 2023/07

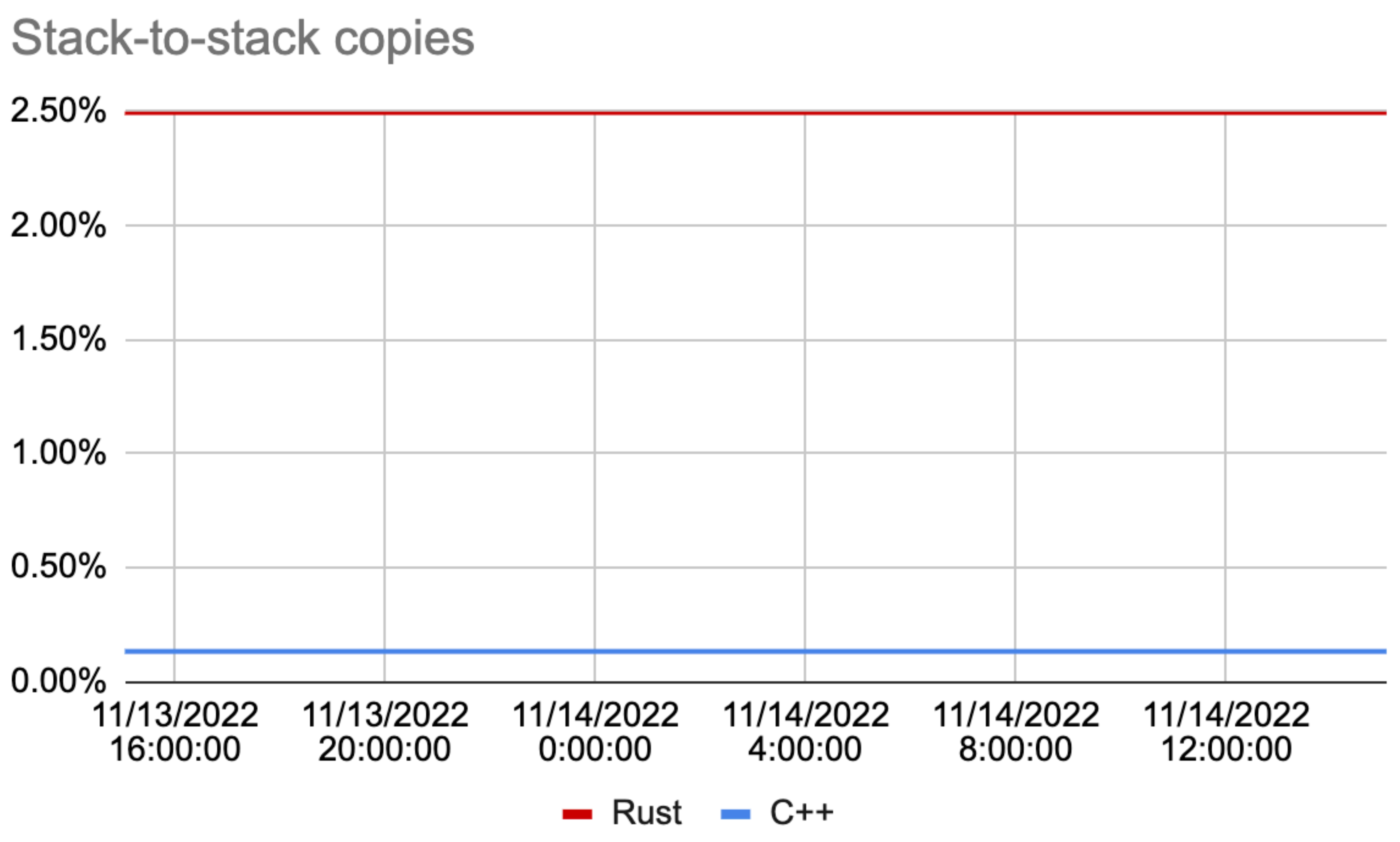
```rust
pub fn clone_string<'a>() -> Vec<String>{
    let mut vector_string = vec![];
    let mut origin = String::from("a");
    repeat_outlined(&mut origin);
    let copied = origin; // memcpy introduced without inlining
    push_outlined(&mut vector_string, copied);
    vector_string
}
```

-C opt-level=3

```llvm
define void @example::clone_string( … ){
start:
  %copied = alloca %"String", align 8
  %origin = alloca %"String", align 8
  %vector_string = alloca %"Vec<String>", align 8
...
bb1:
  call void @llvm.memcpy.p0.p0.i64(ptr … %copied, ptr … %origin,…)
...
```

https://rust.godbolt.org/z/7s5418TYv

Are we stack efficient?



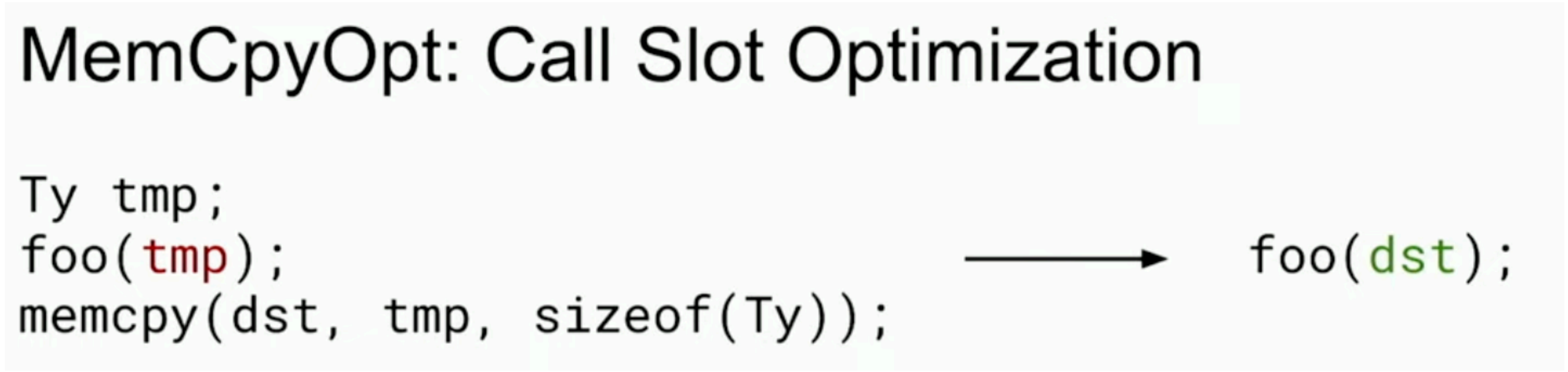
(Although, I'm not sure about what kind of program this is…

https://arewestackefficientyet.com/ by pcwalton

Outline

- LLVM Middle-end Background
- Rust Move codegen (rustc_llvm)
- Optimizations in MemCpyOpt related to Rust Move

MemCpyOpt



MemCpyOpt: Call Slot Optimization

```
Ty tmp;
foo(tmp);                                   →     foo(dst);
memcpy(dst, tmp, sizeof(Ty));
```

From 2023 EuroLLVM - Tutorial: A whirlwind tour of the LLVM optimizer by Nikita Popov (slides)

☀️ I implemented two memcpy optimizations for Rust

- Immutable Argument processing
- Stack Move Optimization

Attributes

☀ LLVM has the concept called attributes,
which can give optimization-helpful info for
1. Function return value, 2. Function arg, etc.

- align N … this param/arg is guaranteed to N bytes-aligned.
- readonly … this param/arg is only read, not written in the function
- noalias … no other alias exists during the execution of the function

ImmutArg 2023/01

```rust
pub fn should_be_no_op(val: Foo) -> Foo {
    val
}

pub fn sum_slices_2(val: Foo) -> u32 {
    let val = should_be_no_op(val);
    sum(&val)
}

pub fn sum(val: &Foo) -> u32{
    val.0
}
```

```llvm
…
call void @llvm.lifetime.start.p0(i64 48, ptr nonnull %val1)
call void @llvm.memcpy.p0.p0.i64(ptr align 8 %val1, ptr align 8 %val,…)
%0 = call noundef i32 @example::sum(ptr noalias readonly align 8 %val1)
call void @llvm.lifetime.end.p0(i64 48, ptr nonnull %val1)
…
```

From https://github.com/rust-lang/rust/issues/107436

ImmutArg 2023/01

```
pub fn should_be_no_op(val: Foo) -> Foo {
    val
}

pub fn sum_slices_2(val: Foo) -> u32 {
    let val = should_be_no_op(val);
    sum(&val)
}

pub fn sum(val: &Foo) -> u32{
    val.0
}
```

Immutable!

```
…
call void @llvm.lifetime.start.p0(i64 48, ptr nonnull %val1)
call void @llvm.memcpy.p0.p0.i64(ptr align 8 %val1, ptr align 8 %val,…)
%0 = call noundef i32 @example::sum(ptr noalias readonly align 8 %val1)
call void @llvm.lifetime.end.p0(i64 48, ptr nonnull %val1)
…
```

From https://github.com/rust-lang/rust/issues/107436

ImmutArg 2023/01

```rust
pub fn should_be_no_op(val: Foo) -> Foo {
    val
}

pub fn sum_slices_2(val: Foo) -> u32 {
    let val = should_be_no_op(val);
    sum(&val)
}

pub fn sum(val: &Foo) -> u32{
    val.0
}
```

Redundant!　　Immutable!

```
…
call void @llvm.lifetime.start.p0(i64 48, ptr nonnull %val1)
call void @llvm.memcpy.p0.p0.i64(ptr align 8 %val1, ptr align 8 %val,…)
%0 = call noundef i32 @example::sum(ptr noalias readonly align 8 %val1)
call void @llvm.lifetime.end.p0(i64 48, ptr nonnull %val1)
…
```

From https://github.com/rust-lang/rust/issues/107436

ImmutArg: IR -> IR 2023/06

```
declare void @f(ptr)
define void @immut_param(ptr align 4 noalias %val) {
  %val1 = alloca i8, align 4
  call void @llvm.memcpy.p0.p0.i64(ptr align 4 %val1, ptr align 4 %val, i64 1, i1 false)
  call void @f(ptr align 4 nocapture noalias readonly %val1)
  ret void
}
```

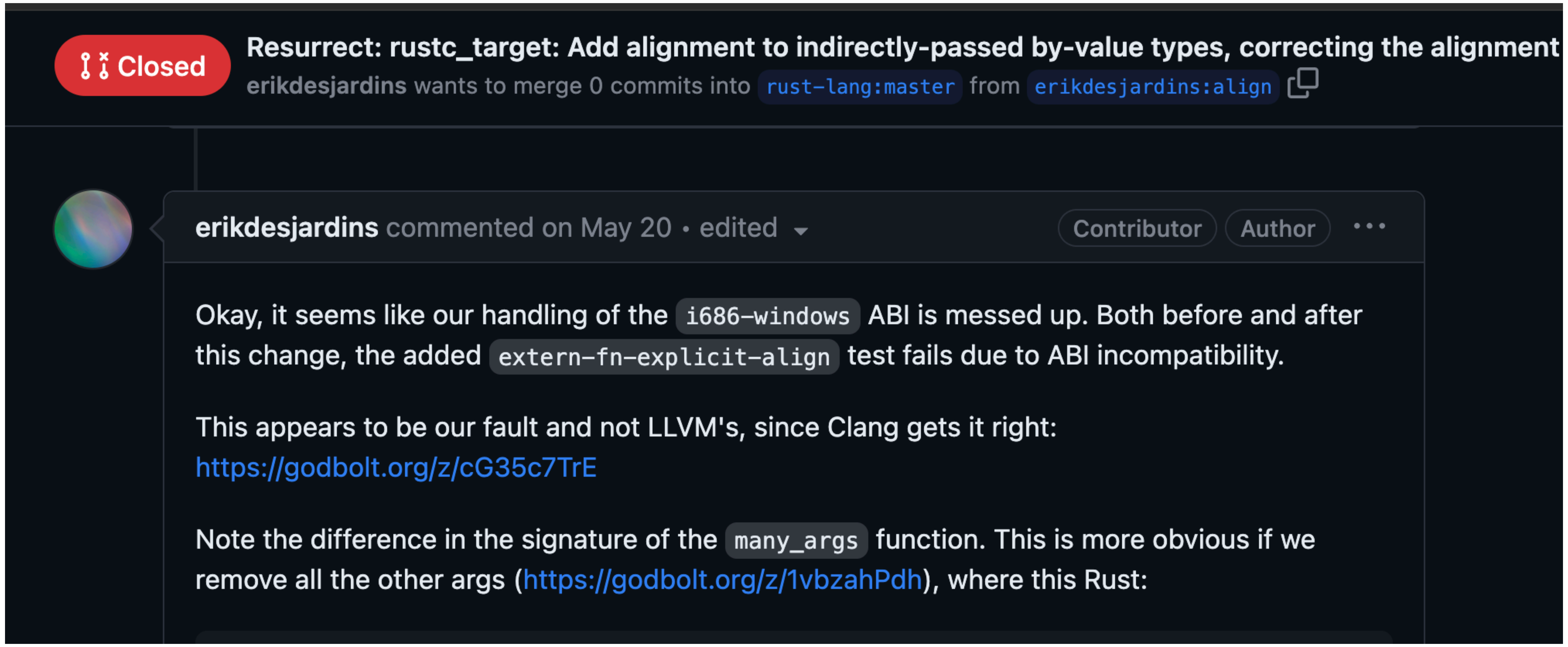
```
define void @immut_param(ptr noalias align 4 %val) {
  call void @f(ptr noalias nocapture readonly align 4 %val)
  ret void
}
```

https://reviews.llvm.org/D150970

No blockers,
Bo Compile regressions
But…

Alignment attr problem on rustc

☀ This optimization (and similar optimization on InstCombine) requires arg/param is attributed with align. msvc blocks it…



Closed

Resurrect: rustc_target: Add alignment to indirectly-passed by-value types, correcting the alignment

erikdesjardins wants to merge 0 commits into `rust-lang:master` from `erikdesjardins:align`

erikdesjardins commented on May 20 • edited

Contributor Author

Okay, it seems like our handling of the `i686-windows` ABI is messed up. Both before and after this change, the added `extern-fn-explicit-align` test fails due to ABI incompatibility.

This appears to be our fault and not LLVM's, since Clang gets it right: https://godbolt.org/z/cG35c7TrE

Note the difference in the signature of the `many_args` function. This is more obvious if we remove all the other args (https://godbolt.org/z/1vbzahPdh), where this Rust:

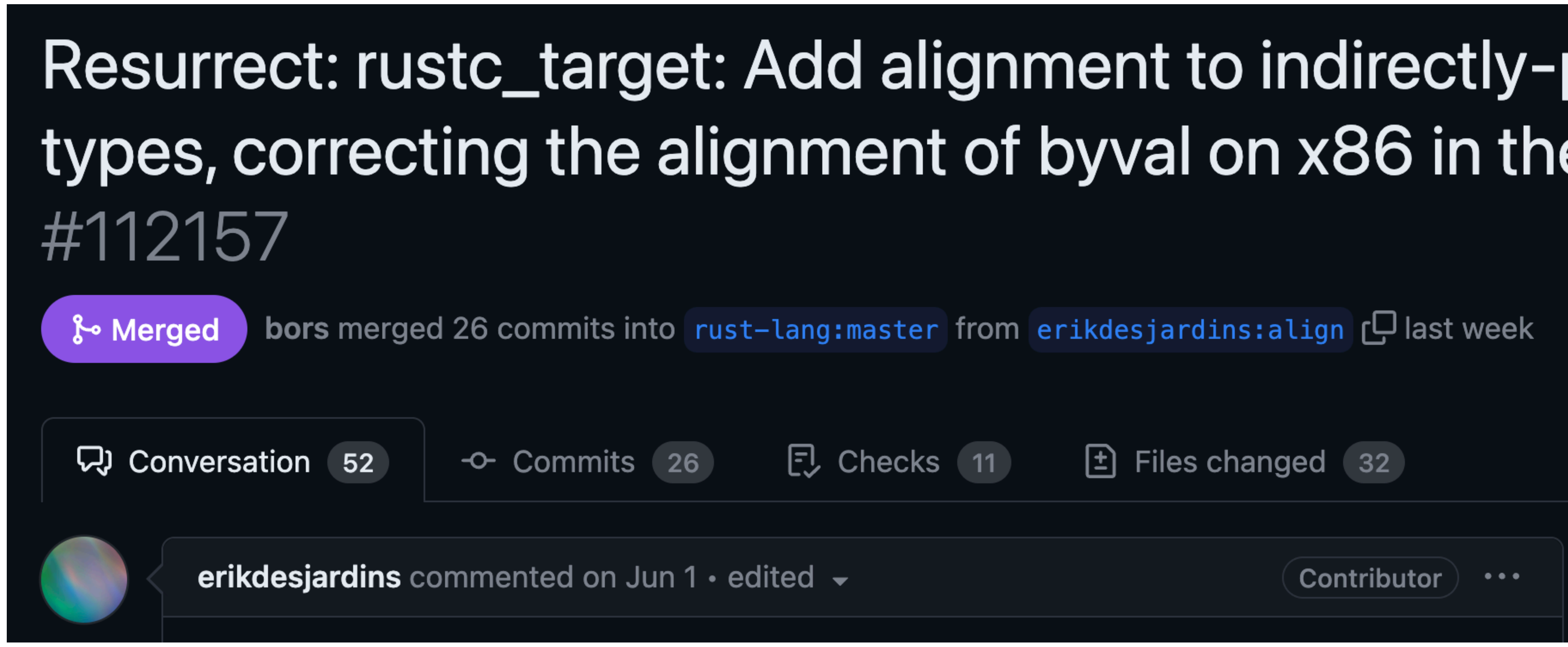
https://github.com/rust-lang/rust/pull/111551

ImmutArg 2023/07(nightly) 🎉🎉🎉

```rust
pub fn should_be_no_op(val: Foo) -> Foo {
    val
}

pub fn sum_slices_2(val: Foo) -> u32 {
    let val = should_be_no_op(val);
    sum(&val)
}

pub fn sum(val: &Foo) -> u32{
    val.0
}
```

-C opt-level=3

```
...
%0 = call noundef i32 @example::sum(ptr noalias readonly align 8 %val)
...
```


Resurrect: rustc_target: Add alignment to indirectly-
types, correcting the alignment of byval on x86 in th
#112157

Merged bors merged 26 commits into rust-lang:master from erikdesjardins:align last week

https://github.com/rust-lang/rust/pull/112157

https://rust.godbolt.org/z/9zss56xjK

StackMoveOptzn

```rust
pub fn clone_string<'a>() -> Vec<String>{
    let mut vector_string = vec![];
    let mut origin = String::from("a");
    repeat_outlined(&mut origin);
    let copied = origin; // memcpy introduced without inlining
    push_outlined(&mut vector_string, copied);
    vector_string
}
```

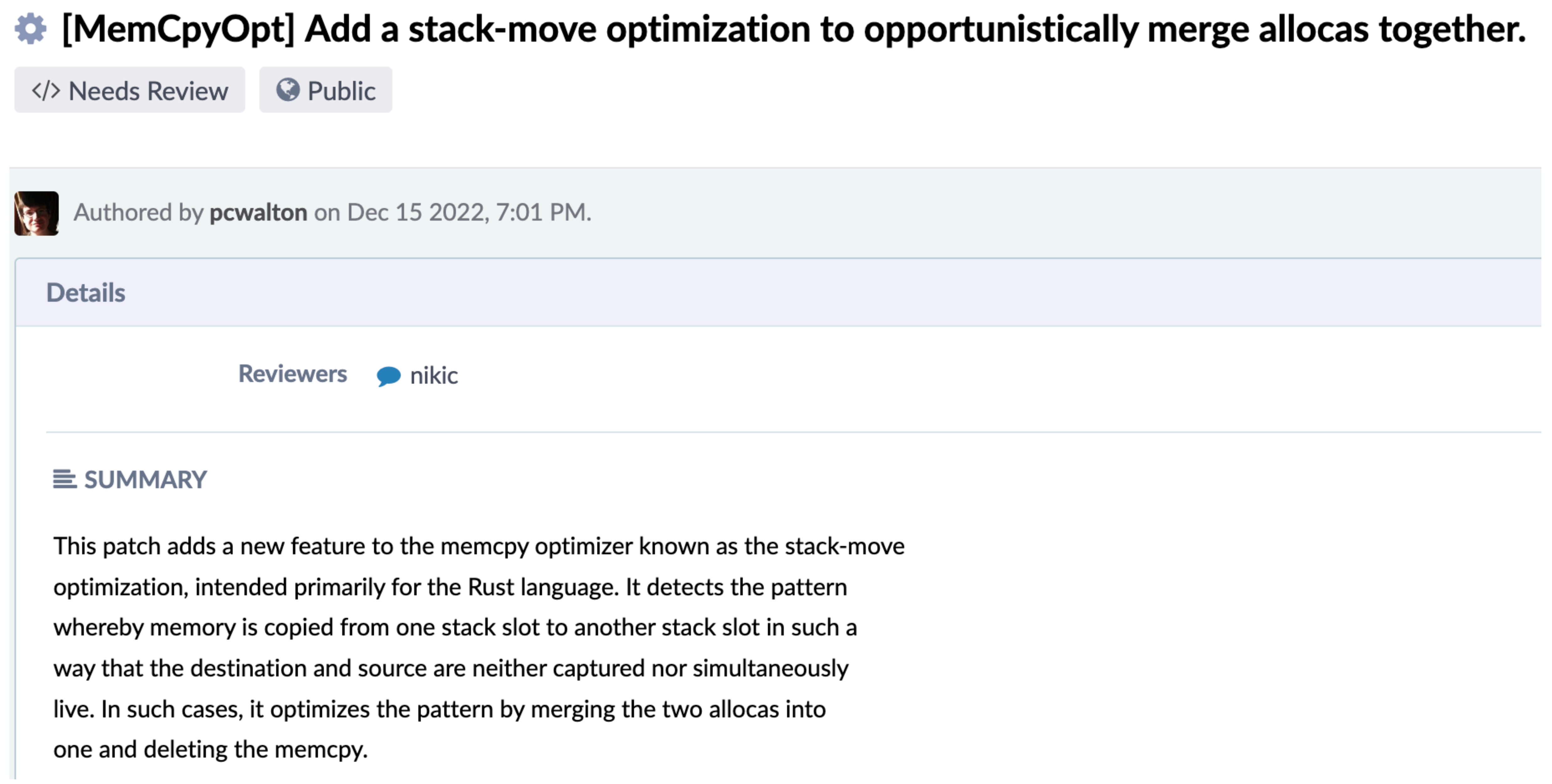
copied and origin are static, unescaped
also have no simultaneous uses

-C opt-level=3

```llvm
define void @example::clone_string( … ){
start:
  %copied = alloca %"String", align 8
  %origin = alloca %"String", align 8
  %vector_string = alloca %"Vec<String>", align 8
...
bb1:
  call void @llvm.memcpy.p0.p0.i64(ptr … %copied, ptr … %origin,…)
...
```

https://rust.godbolt.org/z/7s5418TYv

StackMoveOptzn by pcwalton 2022/12



[MemCpyOpt] Add a stack-move optimization to opportunistically merge allocas together.

</> Needs Review Public

Authored by **pcwalton** on Dec 15 2022, 7:01 PM.

Details

Reviewers: nikic

SUMMARY

This patch adds a new feature to the memcpy optimizer known as the stack-move optimization, intended primarily for the Rust language. It detects the pattern whereby memory is copied from one stack slot to another stack slot in such a way that the destination and source are neither captured nor simultaneously live. In such cases, it optimizes the pattern by merging the two allocas into one and deleting the memcpy.

☀ Extend CaptureTracking with Liveness analysis

→ I am now going to land incrementally 😀

https://reviews.llvm.org/D140089

StackMoveOptzn for single-BB 2023/07

```
define void @basic_memcpy() {
  %src = alloca %struct.Foo, align 4
  %dest = alloca %struct.Foo, align 4
  store %struct.Foo { i32 10, i32 20, i32 30 }, ptr %src
  %1 = call i32 @use_nocapture(ptr nocapture %src)

  call void @llvm.memcpy.p0.p0.i64(ptr align 4 %dest, ptr align 4 %src, i64 12, i1 false)

  %2 = call i32 @use_nocapture(ptr nocapture %dest)
  ret void
}
```

-passes=memcpyopt

```
define void @basic_memcpy() {
  %src = alloca %struct.Foo, align 4
  store %struct.Foo { i32 10, i32 20, i32 30 }, ptr %src
  %1 = call i32 @use_nocapture(ptr nocapture %src)
  %2 = call i32 @use_nocapture(ptr nocapture %src)
  ret void
}
```

☀ **Aliasing Xor Mutability** like check for src and dest of memcpy

Not so much practical yet…

StackMoveOptzn 2023/07 (not merged)

```
…
  %1 = call i32 @use_nocapture(ptr noundef nocapture %src)
  br i1 %b0, label %bb0, label %exit

exit:
  %2 = call i32 @use_nocapture(ptr noundef nocapture %src)
  ret void

bb0:
  call void @llvm.memcpy.p0.p0.i64(ptr align 4 %dest, ptr align 4 %src, i64 12, i1 false)
  %3 = call i32 @use_nocapture(ptr noundef nocapture %src)
  ret void
…
```

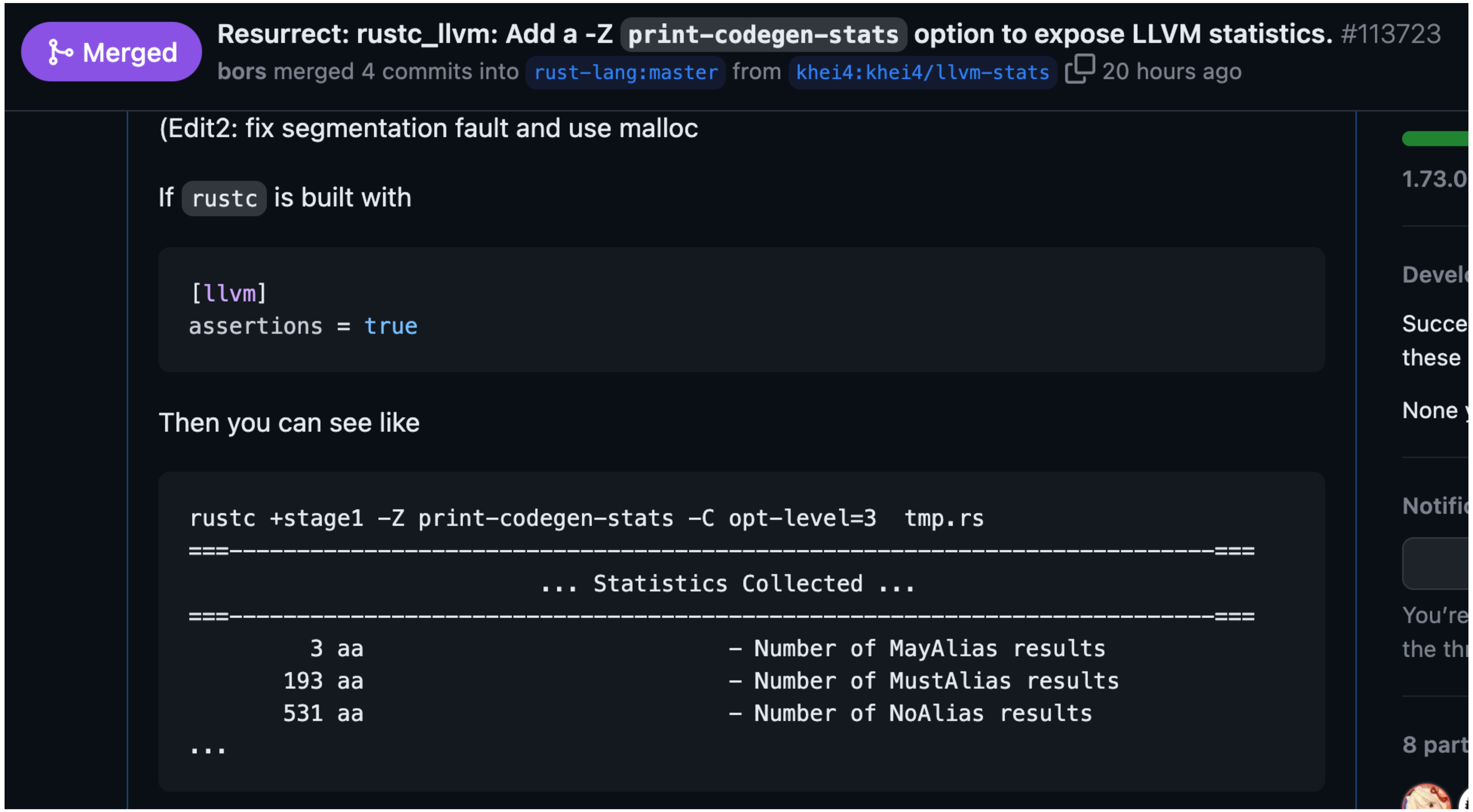
-passes=memcpyopt

```
…
  %1 = call i32 @use_nocapture(ptr noundef nocapture %src)
  br i1 %b0, label %bb0, label %exit

exit:
  %2 = call i32 @use_nocapture(ptr noundef nocapture %src)
  ret void

bb0:
  %3 = call i32 @use_nocapture(ptr noundef nocapture %src)
  ret void
…
```

☀ Still in review

rustc -Z print-codegen-stats

☀️ You can see results on rustc soon. (Hopefully)



Merged

Resurrect: rustc_llvm: Add a -Z `print-codegen-stats` option to expose LLVM statistics. #113723

bors merged 4 commits into `rust-lang:master` from `khei4:khei4/llvm-stats` 20 hours ago

(Edit2: fix segmentation fault and use malloc

If `rustc` is built with

```
[llvm]
assertions = true
```

Then you can see like

```
rustc +stage1 -Z print-codegen-stats -C opt-level=3  tmp.rs
===-------------------------------------------------------------------------===
                          ... Statistics Collected ...
===-------------------------------------------------------------------------===
         3 aa                          - Number of MayAlias results
       193 aa                          - Number of MustAlias results
       531 aa                          - Number of NoAlias results
...
```

https://github.com/rust-lang/rust/pull/113723

Acknowledgment

- ☀️ Almost all idea of move optimization idea attributes to Patrick Walton (@pcwalton)

- ☀️ All my patches are very eagerly reviewed by my mentor, Nikita Popov(@nikic)

- ☀️ Many crash reports from ongoing LLVM phabricator revisions.

- ☀️ So many feedbacks about proposal from my previous colleagues.

I appreciate all their helps! 😊