A data-driven approach to debug info quality

Using the Swift frontend as an example

Emil Pedersen — October 24th, 2024 — LLVM Dev Meeting 2024

Introduction

Emil Pedersen

- Student at Epitech in Paris
- Interned at Apple earlier this year
- This presentation is based on my intern project
- Speaking in a personal capacity

Special thanks to my mentor Adrian Prantl, and the Swift and LLDB teams

Why debug optimized code?

Embedded Programming

Crash Logs

Complex Programs

Optimized code is hard to debug

- Stepping is not always reliable
- Variables are often unavailable
- Variables sometimes disappear altogether

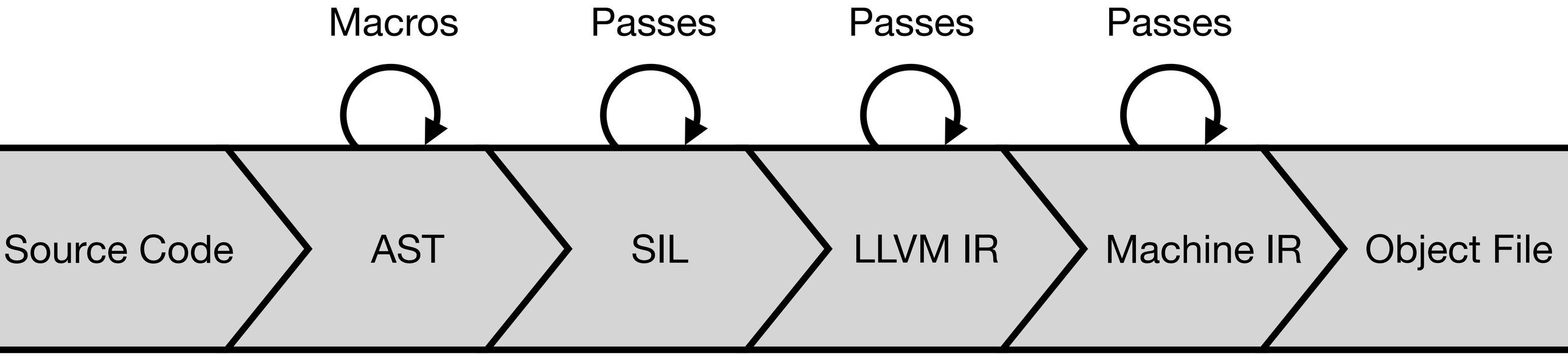
Optimized code is hard to debug

- Stepping is not always reliable
  - **Variables are often unavailable**
  - **Variables sometimes disappear altogether**

This is preventable

Why does this happen?

The Swift compiler pipeline



Variable

VarDecl

debug_value

#dbg_value

DBG_VALUE

DWARF

Why does this happen?

- Variables can be lost in translation
- Optimization passes aren’t designed to retain debug info for variables
- SIL optimization passes often disregard variable debug info altogether

Why does this happen?

- Variables can be lost in translation
  - **Optimization passes aren’t designed to retain debug variables**
  - **SIL optimization passes often disregard debug variables altogether**

How can we improve optimization passes?

How can we improve optimization passes?

Simple approaches

Read optimization passes and find where variables are dropped

Using known examples of lost variables, bisect to find where it is dropped

How can we improve optimization passes?

DExTer: A good tester for a known issue

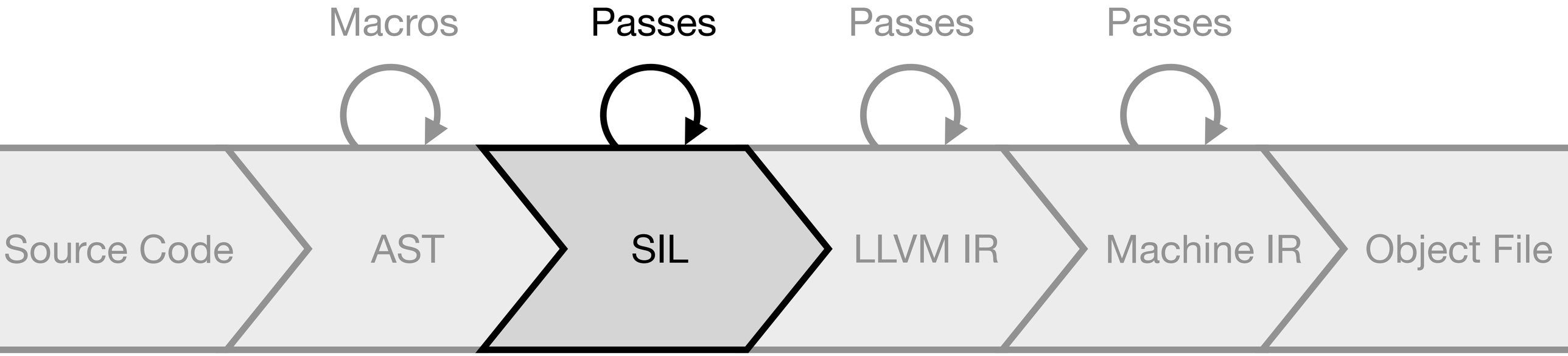
- It can help bisecting LLVM passes
- It requires a code sample with a known debugging problem
- It runs the whole pipeline, which is slow

The solution

Variable drop statistics

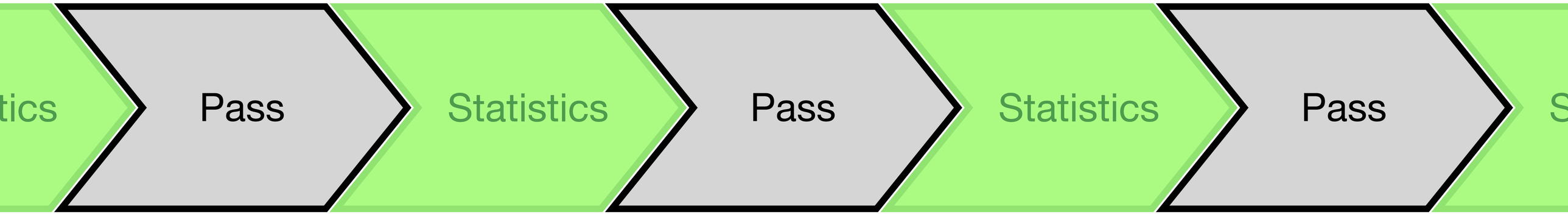
Variable drop statistics

Within the optimization pipeline



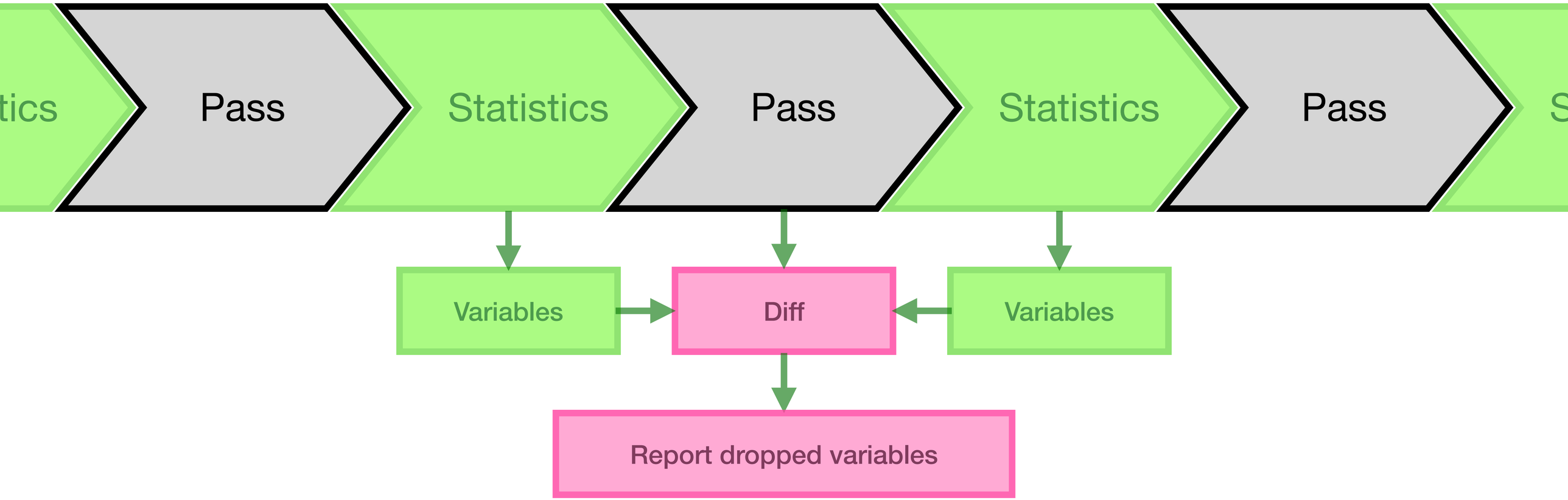
Variable drop statistics

Within the optimization pipeline



Variable drop statistics

Within the optimization pipeline



What constitutes a variable?

- Same name
- Same scope (including inlining information)
- Same source location

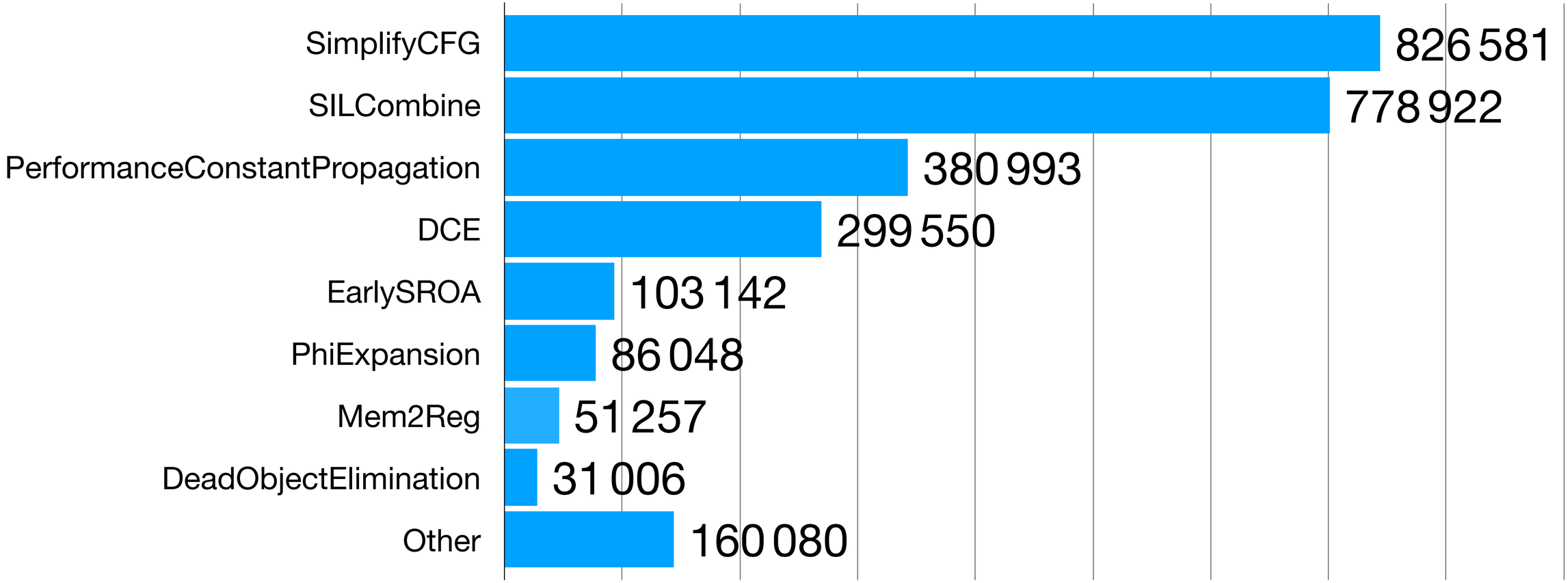
Which Swift code base to use?

Standard Library — Impacts all code

Source Compatibility Test Suite — 200+ open source projects

Report results (SIL)

Variables dropped by SIL passes, -Ospeed (Top 8)



Number of dropped variables in the source compatibility test suite
March 2024

Some variables should be dropped

Variables that can be completely removed

- When a function or scope is unreachable

  ➡ DCE, DeadFunctionElimination

- When a function is marked transparent

  ➡ DeadAllocationElimination

```
func buttonPress() {
    if false {
        let code = 1
        exit(code)
    }
}
```

The variable will not appear in the debugger

Some variables should be dropped

Variable values that must be dropped

- When the variable is unused, or moved
  - ➡ SimplifyCFG, Dead Object Elimination
- When a value is folded and not recoverable
  - ➡ SILCombine, Constant Propagation

```
func buttonPress() {
    let code = cos(0)
    if false {
        exit(code)
    }
}
```

The variable will appear unavailable in the debugger

Some variables shouldn’t be dropped

Variable values that must NOT be dropped

- When the value changes memory location
  - ➡ SROA, Mem2Reg, PhiExpansion
  - ➡ AllocBoxToStack, AllocStackHoisting, LoadableByAddress
- When a constant is folded
  - ➡ DiagnosticConstantPropagation

Some variables shouldn’t be dropped

Variable values that must NOT be dropped

- When the value changes memory location

  ➡ SROA, Mem2Reg, PhiExpansion

  ➡ **AllocBoxToStack**, **AllocStackHoisting**, **LoadableByAddress**

- When a constant is folded

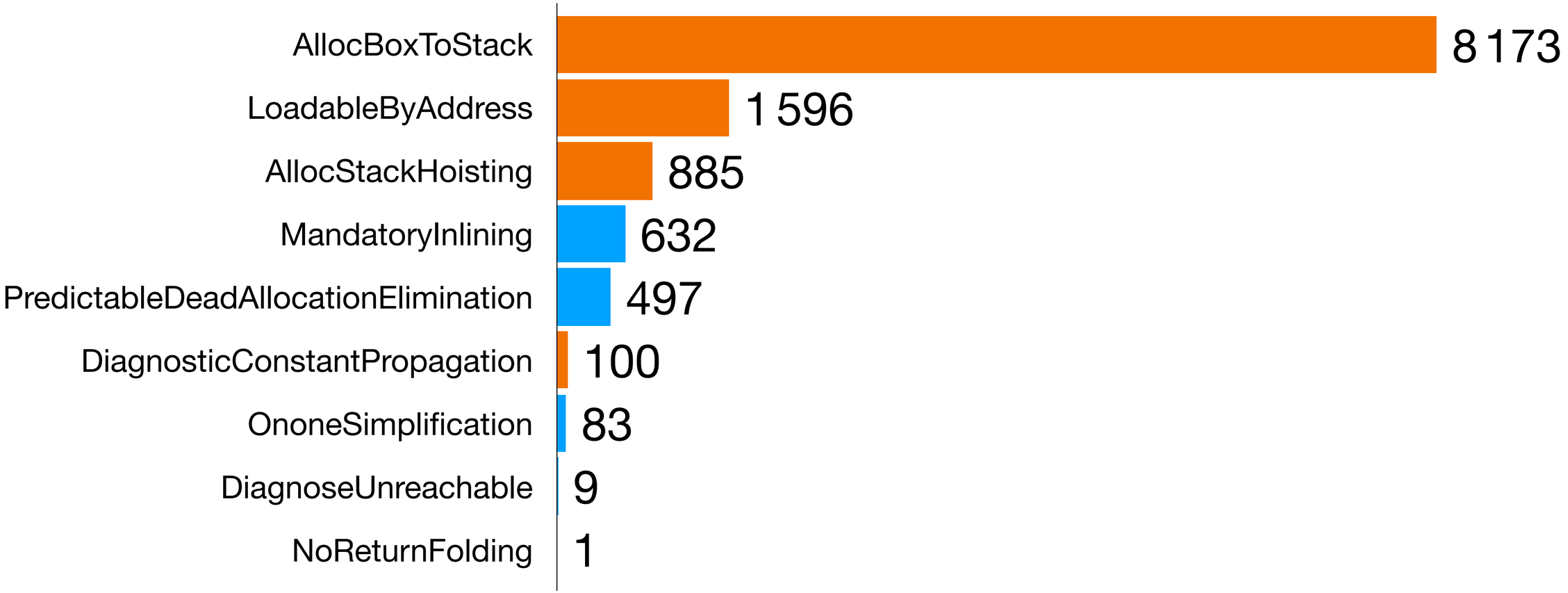
  ➡ **DiagnosticConstantPropagation**

These SIL passes are run for debug builds

Report results (SIL)

Variables dropped by SIL passes, -Onone

BEFORE

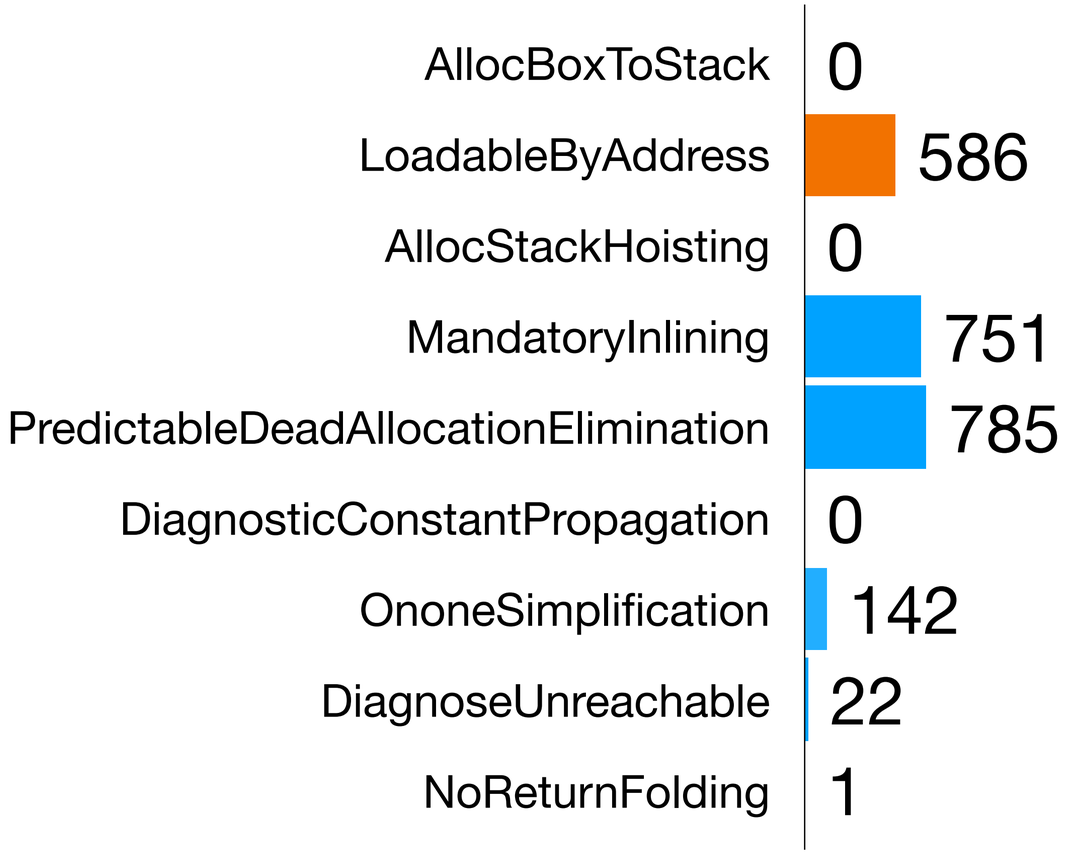


Number of dropped variables in the source compatibility test suite (-Onone)
March 2024

Report after changes (SIL)

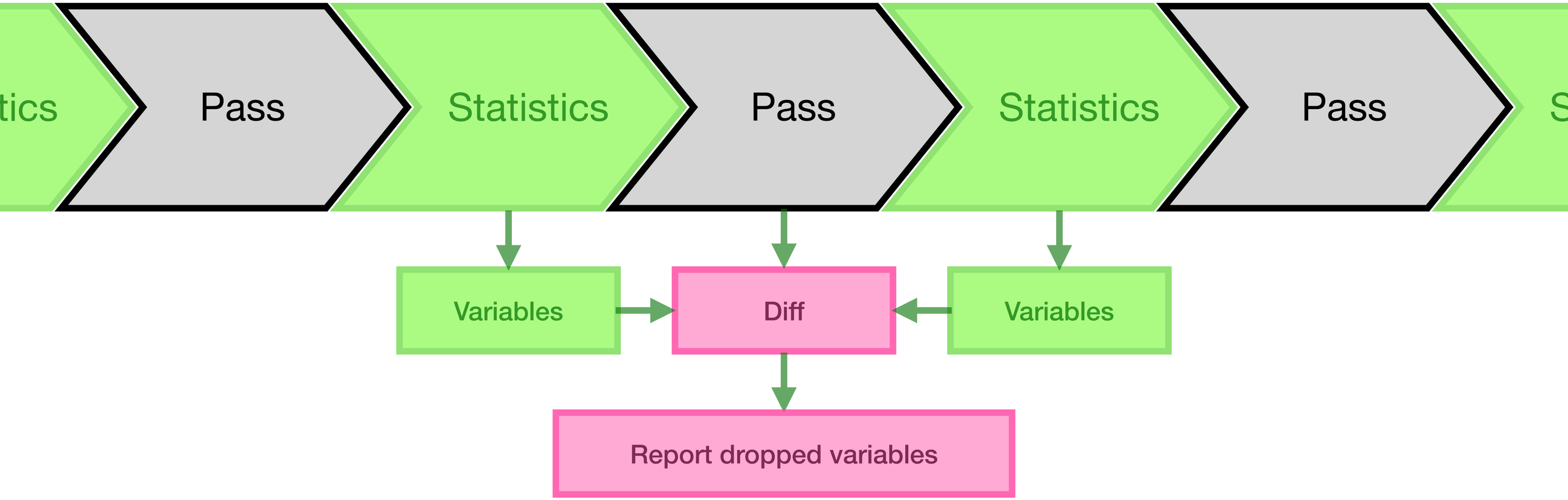
Variables dropped by SIL passes, -Onone

AFTER

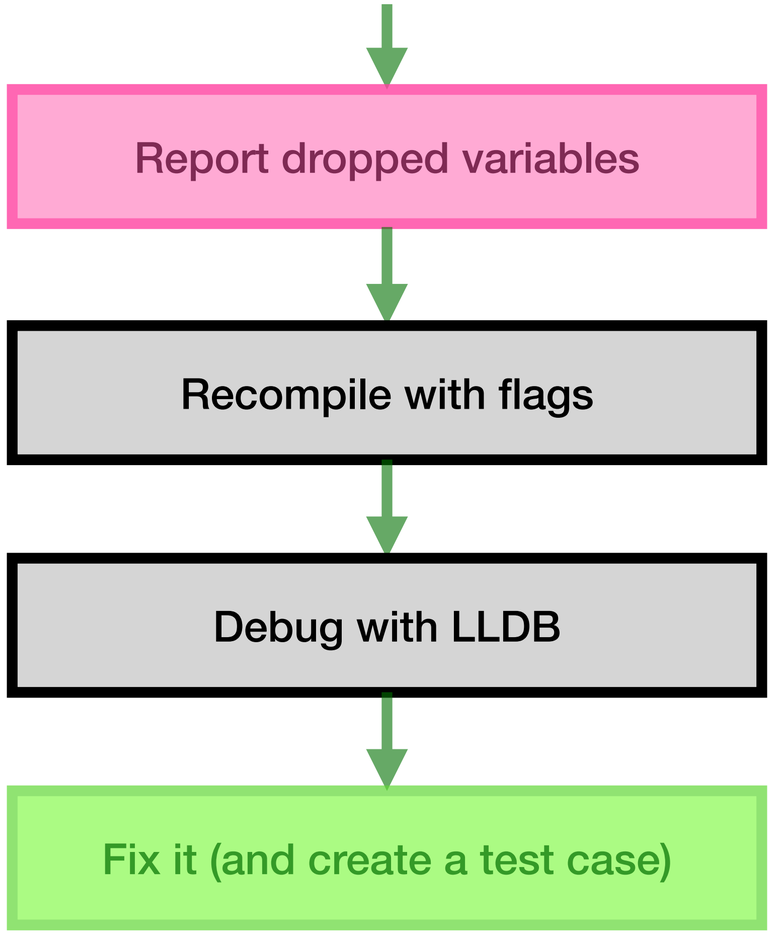


Number of dropped variables in the source compatibility test suite (-Onone)
Present

How to find mistakes in the passes?



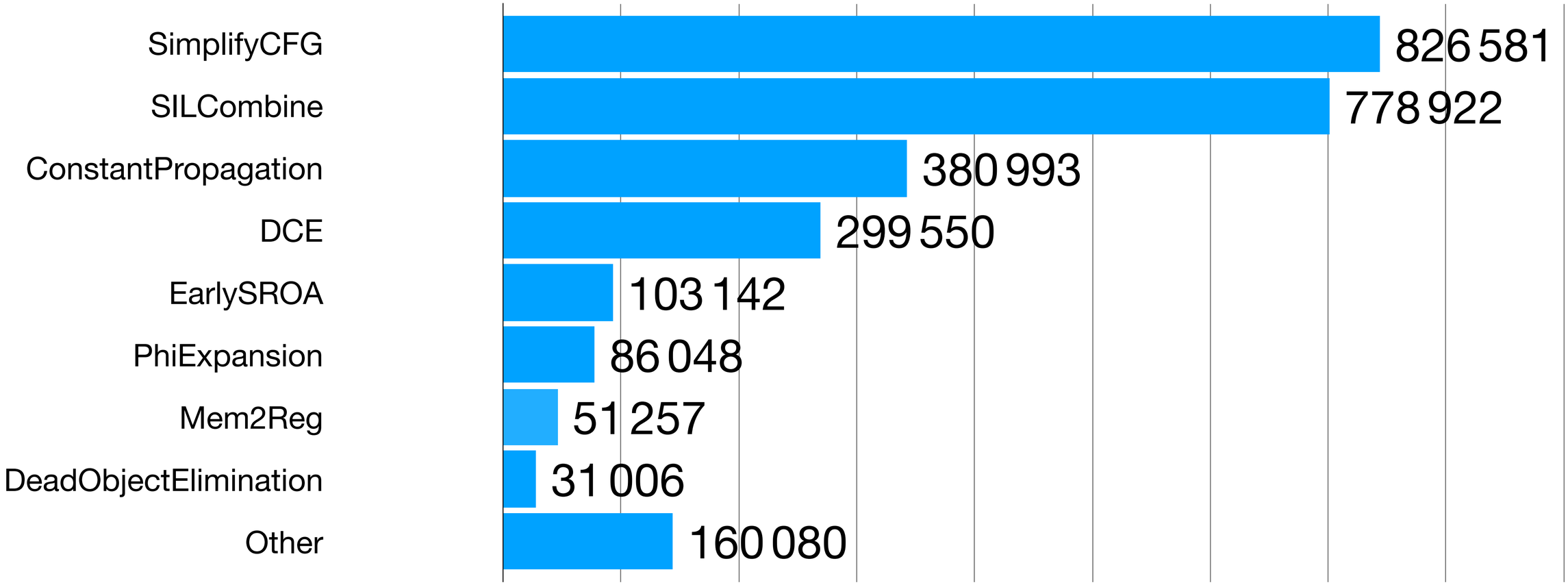
How to find mistakes in the passes?



Report results (SIL)

Variables dropped by SIL passes, -Ospeed (Top 8)

BEFORE

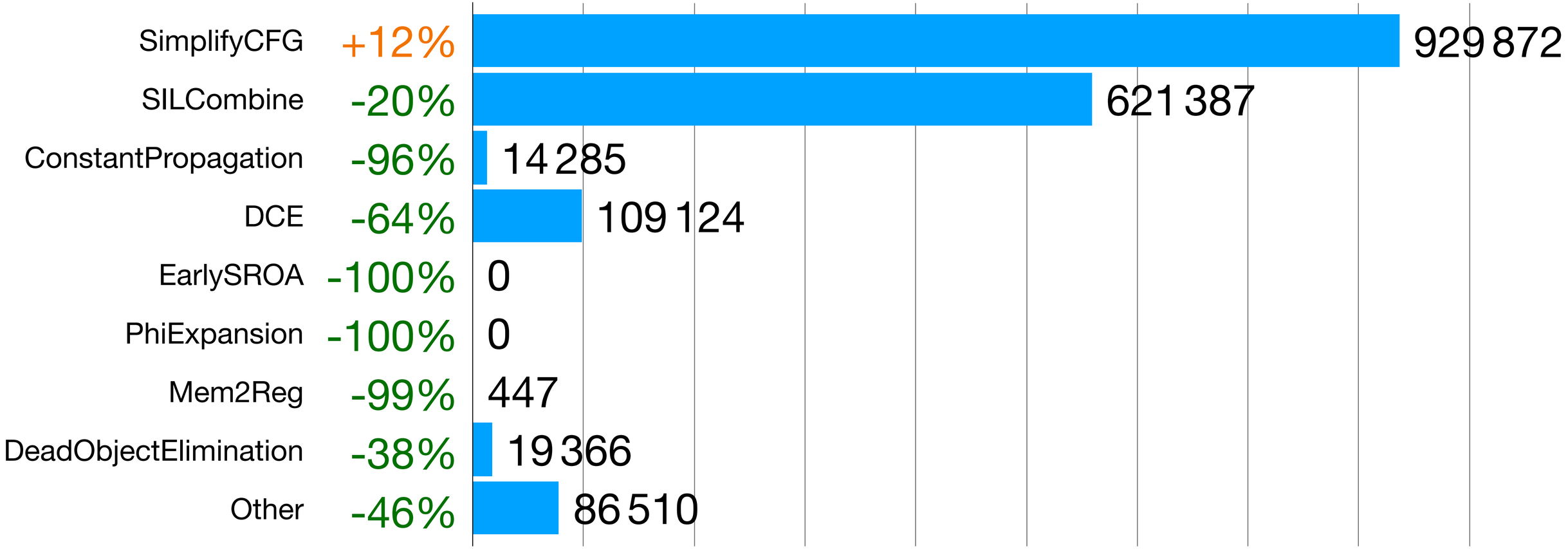


Number of dropped variables in the source compatibility test suite
March 2024

Report after changes (SIL)

Variables dropped by SIL passes, -Ospeed (Top 8)

AFTER

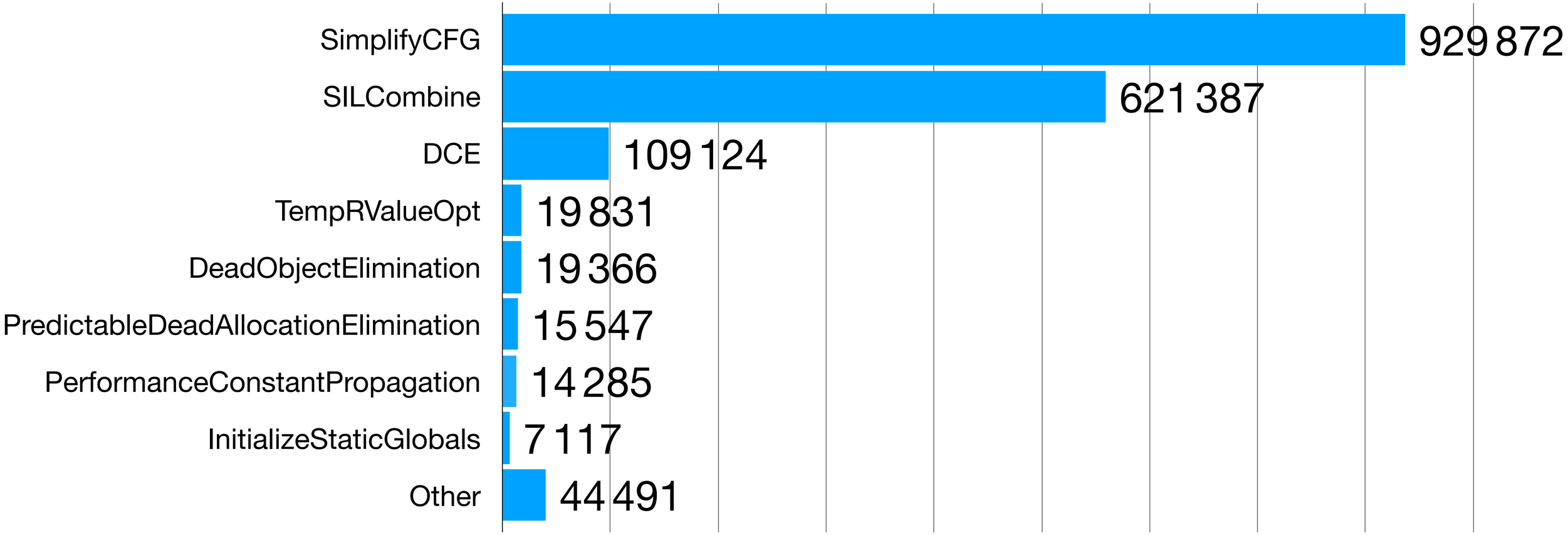


Number of dropped variables in the source compatibility test suite
Present

Report after changes (SIL)

Variables dropped by SIL passes, -Ospeed (Top 8)

AFTER



Number of dropped variables in the source compatibility test suite
Present

Pull Requests

For the variable drop statistics

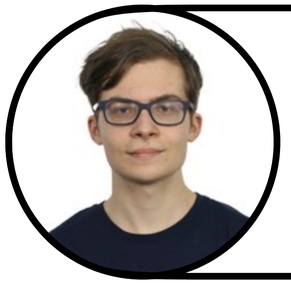
- Swift
  - swiftlang/swift#73334
- LLVM
  - llvm/llvm-project#102233 (by Shubham Rastogi)

Future Directions

- Merge the patch for LLVM IR
- Add this feature to Machine IR
- Add a similar detection when lowering (IRGen, ISel, etc.)
- Fine-grained statistics for InstCombine, SILCombine, etc.
- Discriminate between the reasons why a variable was dropped

Thank you!

Questions?



Emil Pedersen

contact@emil.codes

github.com/Snowy1803

in/emil-b-pedersen