



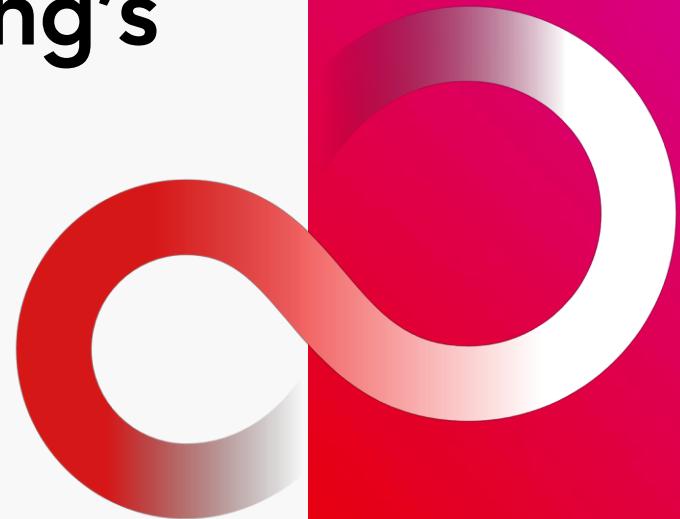
Developers' Meeting

BERLIN 2025

Leveraging “nsw” in Flang’s LLVM IR Generation for Vectorization

Yusuke Minato

LLVM Developers’ Meeting – April 2025



- Vectorization for Flang
 - Flang: the Fortran frontend in the LLVM project
 - Flang relies on the LoopVectorize pass in the backend.
 - Its frontend itself doesn't have a vectorization pass.
 - We're trying to improve the capability of vectorization in the backend.
 - Vectorization plays an important role in optimizations.
- TSVC (Test Suite for Vectorizing Compilers)
 - Available in both Fortran and C
 - Helps distinguish frontend problems from backend issues
 - The frontend can affect vectorization because it may generate LLVM IR that is difficult for the backend to vectorize.

Flang's Capability of Vectorization



- Evaluation using TSVC
 - Options: -O3 -ffast-math -march=armv9-a
- Our contribution
 - Three additional loops can be vectorized since LLVM 20.
 - By introducing several options related to integer overflow into Flang

Vectorizable or not with		Number in LLVM 19	Number in LLVM 20
Clang	Flang		
Vectorizable	Vectorizable	76	80 (+4)
Vectorizable	Non-Vectorizable	11	9 (-3+1)
Non-Vectorizable	Vectorizable	1	1 (0)
Non-Vectorizable	Non-Vectorizable	40	38 (-2)

Analysis of the Result

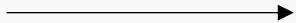


- 11 loops fall into four categories:
 - A) **Suboptimal analysis for array subscripts (3 loops)**
 - B) Reduction variables passed by reference as real arguments (4 loops)
 - C) Gather/scatter with non-constant strides at compile-time (3 loops)
 - D) Loops requiring LTO only for Fortran version (1 loop)
- Categories A and B can be addressed in the frontend.
 - Adding more information to generated MLIR makes it easier for the LoopVectorize pass to vectorize the input LLVM IR.

Address Calculations in Fortran

- Array subscripts: 32 bits, Addresses: 1 word
 - 1 word is equal to 64 bits on 64-bit CPUs.
- Address calculations frequently involve different bit lengths.
 - Lower bounds of subscripts are rarely 0, unlike in C.
 - Address calculations need a step to calculate offsets from subscripts.
- Internal representation of array subscripts gets complicated.
 - This can significantly influence analyses and loop vectorization.

```
real a(n)
do i=1,n-1
  ... a(i+1) ...
end do
```



```
%16 = load i32, ptr %3, align 4 ;; i
%17 = add i32 %16, 1 ;; i + 1
%18 = sext i32 %17 to i64           ↑ subscript
%19 = sub nsw i64 %18, 1 ;; subtract the lower bound
%20 = mul nsw i64 %19, 1 ;; multiply by the stride
%21 = mul nsw i64 %20, 1 ;; multiply by the size of lower dims
%22 = add nsw i64 %21, 0
%23 = mul nsw i64 1, %
%24 = getelementptr float, ptr %0, i64 %22 ;; a + ((i + 1) - 1L)
%25 = load float, ptr %24, align 4 ;; a(i+1)
```

↑ subscript
↓ address

Address calculations are linearized because the shape of an array is often mutable in Fortran.

- Attribute on add/sub/mul/shl/trunc in LLVM IR
 - Shows the result will not overflow the range of signed integer
- Use case of nsw
 - Simplifying calculations involving different bit lengths
 - $(i + 1) - 1L == i$?
- Do not add nsw to instructions whose results could overflow.
 - cf. Rust (release mode)
 - The behavior of integer overflow is defined (wraparound).

- Attribute on add/sub/mul/shl/trunc in LLVM IR
 - Shows the result will not overflow the range of signed integer
- Use case of nsw
 - Simplifying calculations involving different bit lengths
 - $(i + 1) - 1L == i ?$
 - Where $i = \text{INT_MAX}$, (LHS) = $(\text{long})\text{INT_MIN} - 1L$, (RHS) = $(\text{long})\text{INT_MAX}$
 - Equivalent only if the addition has an nsw flag (i.e., $i \leq \text{INT_MAX} - 1$)
 - Do not add nsw to instructions whose results could overflow.
 - cf. Rust (release mode)
 - The behavior of integer overflow is defined (wraparound).

Integer Overflow in Fortran



- The Fortran standard does not mention integer overflow.
 - Leaves the handling of integer overflow up to compiler developers
- We can assume that some operations will never overflow:
 - Increments of DO loop variables
 - Calculations for array subscripts
 - Calculations for loop bounds
- However, code that violates our assumption may exist.
 - The option `-fwrapv` has been introduced into Flang, similar to Clang.

Implementation Details

- Introduced a new flag in FirOpBuilder, a kind of IRBuilder
 - The Builder lowers the AST to IR recursively.
 - This flag controls whether nsw is added to the target operations.
 - Caution: Fortran 2008 and later have intrinsics for bitwise comparisons.
- Patches
 - [#91579](#), [#110060](#), [#113854](#), [#110061](#), [#118933](#)

```
1899 1904     template <typename T>
1900 1905     hlfir::Entity
1901 1906     HlfirDesignatorBuilder::genSubscript(const Fortran::evaluate::Expr<T> &expr) {
1907 +     fir::FirOpBuilder &builder = getBuilder();
1908 +     mlir::arith::IntegerOverflowFlags iofBackup{};
1909 +     if (!getConverter().getLoweringOptions().getIntegerWrapAround()) {
1910 +         iofBackup = builder.getIntegerOverflowFlags();
1911 +         builder.setIntegerOverflowFlags(mlir::arith::IntegerOverflowFlags::nsw);
1912 +     }
1913     auto loweredExpr =
1914         HlfirBuilder(getLoc(), getConverter(), getSymMap(), getStmtCtx())
1915         .gen(expr);
1905 -     fir::FirOpBuilder &builder = getBuilder();
1916 +     if (!getConverter().getLoweringOptions().getIntegerWrapAround())
1917 +         builder.setIntegerOverflowFlags(iofBackup);
    779 +     auto iofi =
    780 +         mlir::dyn_cast<mlir::arith::ArithIntegerOverflowFlagsInterface>(*op);
    781 +     if (iofi) {
    782 +         llvm::StringRef arithIOFAttrName = iofi.getIntegerOverflowAttrName();
    783 +         if (integerOverflowFlags != mlir::arith::IntegerOverflowFlags::none)
    784 +             op->setAttr(arithIOFAttrName,
    785 +                         mlir::arith::IntegerOverflowFlagsAttr::get(
    786 +                             op->getContext(), integerOverflowFlags));
    787 +     }
```

LLVM IR Example with nsw Required



```
1 subroutine sample(a,lb,ub)
2   integer lb,ub,i
3   integer a(lb:ub)
4   do i=lb,ub-1
5     a(i+1) = i-lb
6   end do
7 end subroutine
```



```
1 define void @sample_(ptr captures(none) %0, ptr captures(none) %1, ptr captures(none) %2) {
2   %4 = load i32, ptr %1, align 4, !tbaa !4
3   %5 = sext i32 %4 to i64
4   %6 = load i32, ptr %2, align 4, !tbaa !10
5   %7 = sext i32 %6 to i64
6   %8 = sub i64 %7, %5
7   %9 = add i64 %8, 1
8   %10 = icmp sgt i64 %9, 0
9   %11 = select i1 %10, i64 %9, i64 0
10  %12 = sub nsw i32 %6, 1 ; the upper bound of the loop
11  %13 = sext i32 %12 to i64
12  %14 = trunc i64 %5 to i32
13  %15 = sub i64 %13, %5
14  %16 = add i64 %15, 1
15  br label %17
16
17:                                ; preds = %21, %3
18  %18 = phi i32 [ %32, %21 ], [ %14, %3 ]
19  %19 = phi i64 [ %33, %21 ], [ %16, %3 ]
20  %20 = icmp sgt i64 %19, 0
21  br i1 %20, label %21, label %34
22
23  21:                                         ; preds = %17
24  %22 = load i32, ptr %1, align 4, !tbaa !4
25  %23 = sub i32 %18, %22
26  %24 = add nsw i32 %18, 1 ; the array subscript
27  %25 = sext i32 %24 to i64
28  %26 = sub nsw i64 %25, %5
29  %27 = mul nsw i64 %26, 1
30  %28 = mul nsw i64 %27, 1
31  %29 = add nsw i64 %28, 0
32  %30 = mul nsw i64 1, %11
33  %31 = getelementptr i32, ptr %0, i64 %29
34  store i32 %23, ptr %31, align 4, !tbaa !12
35  %32 = add nsw i32 %18, 1 ; the increment of the DO variable
36  %33 = sub i64 %19, 1
37  br label %17
38
39  34:                                         ; preds = %17
40  ret void
41 }
```

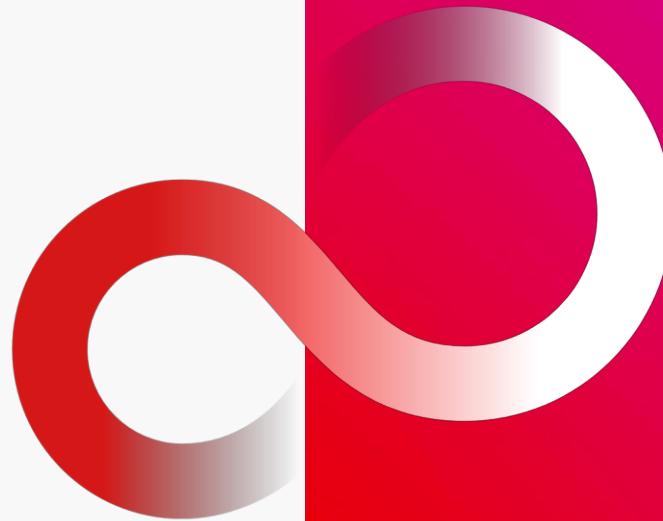
- Performance regression in the LoopStrengthReduce pass
 - While `sdiv` is changed to `udiv` in the InstCombine pass when considering `nsw` on its operands, it blocks analysis for the optimization.
 - <https://github.com/llvm/llvm-project/issues/117318>
- Remaining issues identified by TSVC (categories B and C)
 - Adding `nocapture` attribute to arguments
 - <https://github.com/llvm/llvm-project/issues/106682>
 - Accepting non-constant strides if they are found to be loop-invariant
 - <https://github.com/llvm/llvm-project/issues/110611>

Acknowledgements



- This presentation is based on results obtained from a project, JPNP21029, subsidized by the New Energy and Industrial Technology Development Organization (NEDO).

Thank you





Developers' Meeting

BERLIN 2025



Instruction Cache Prefetching



Oriel Avraham

Compiler Engineer @ Mobileye



Agenda

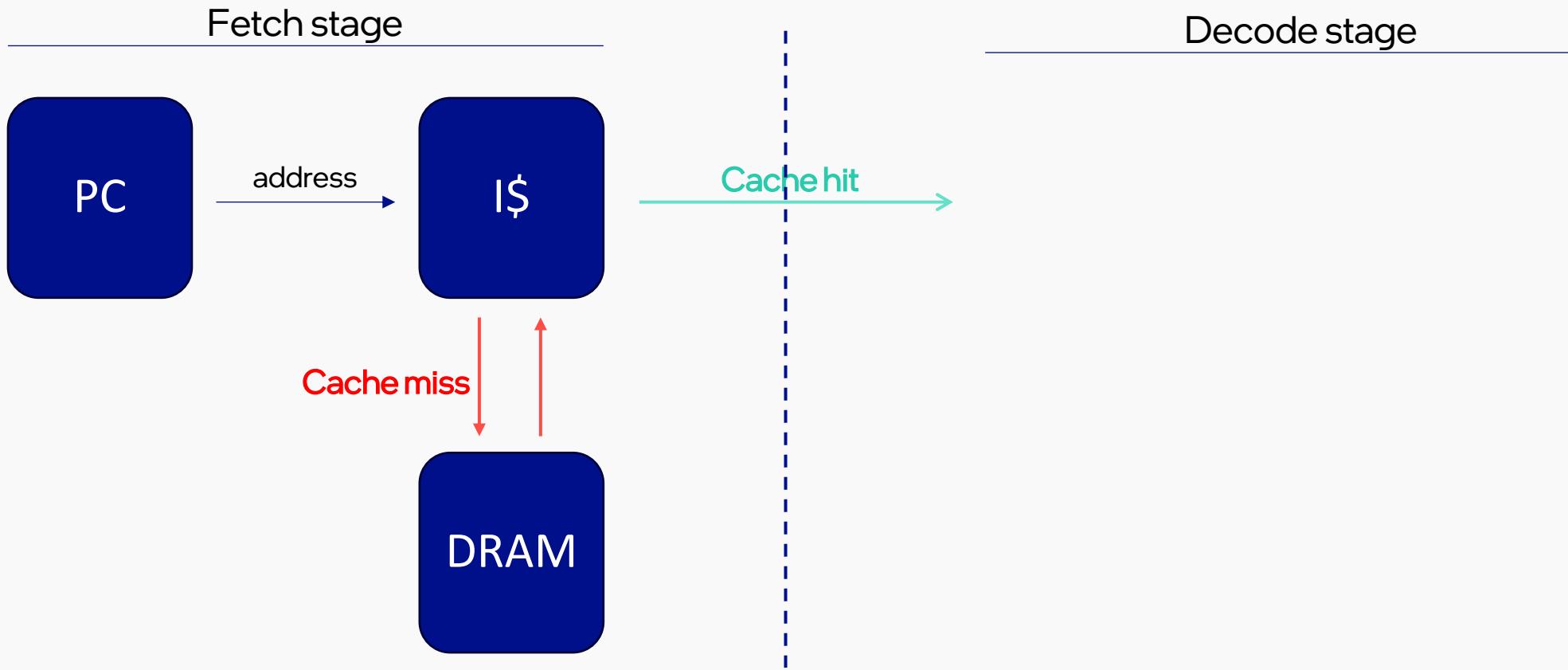
- Instruction Cache
- Prefetch Instruction
- LLVM Analyses
- Performance Impact



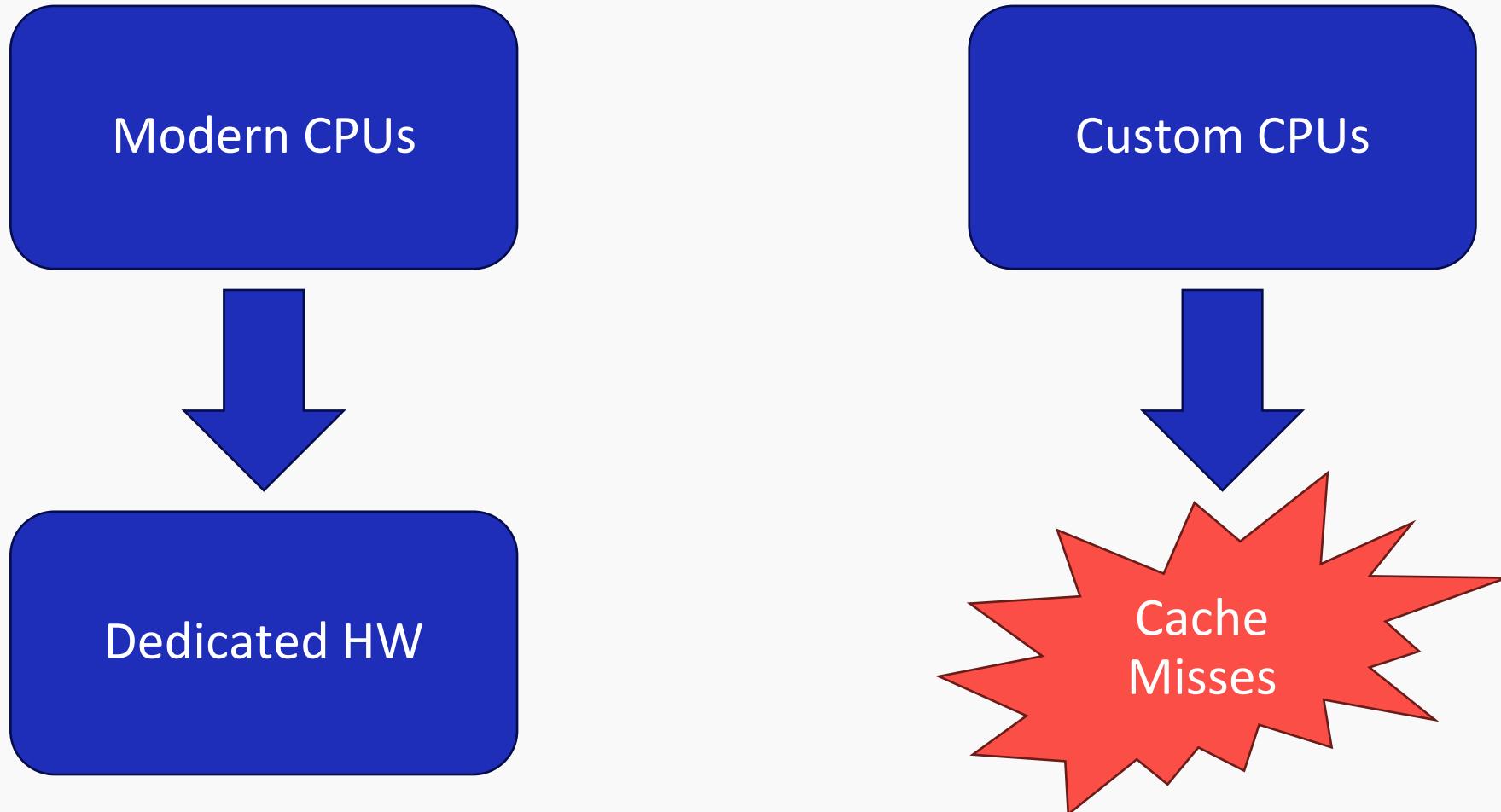
Instruction Cache

What is an Instruction Cache (I\$)

A small, fast memory inside the CPU that stores recently fetched instructions.



How do we keep the I\$ Updated?





Prefetch Instruction

Prefetch Instruction - Specifications

A **prefetch instruction** is a **hint** to the processor to fetch instructions into the I\$.

Syntax

`prefetch(&Label)`

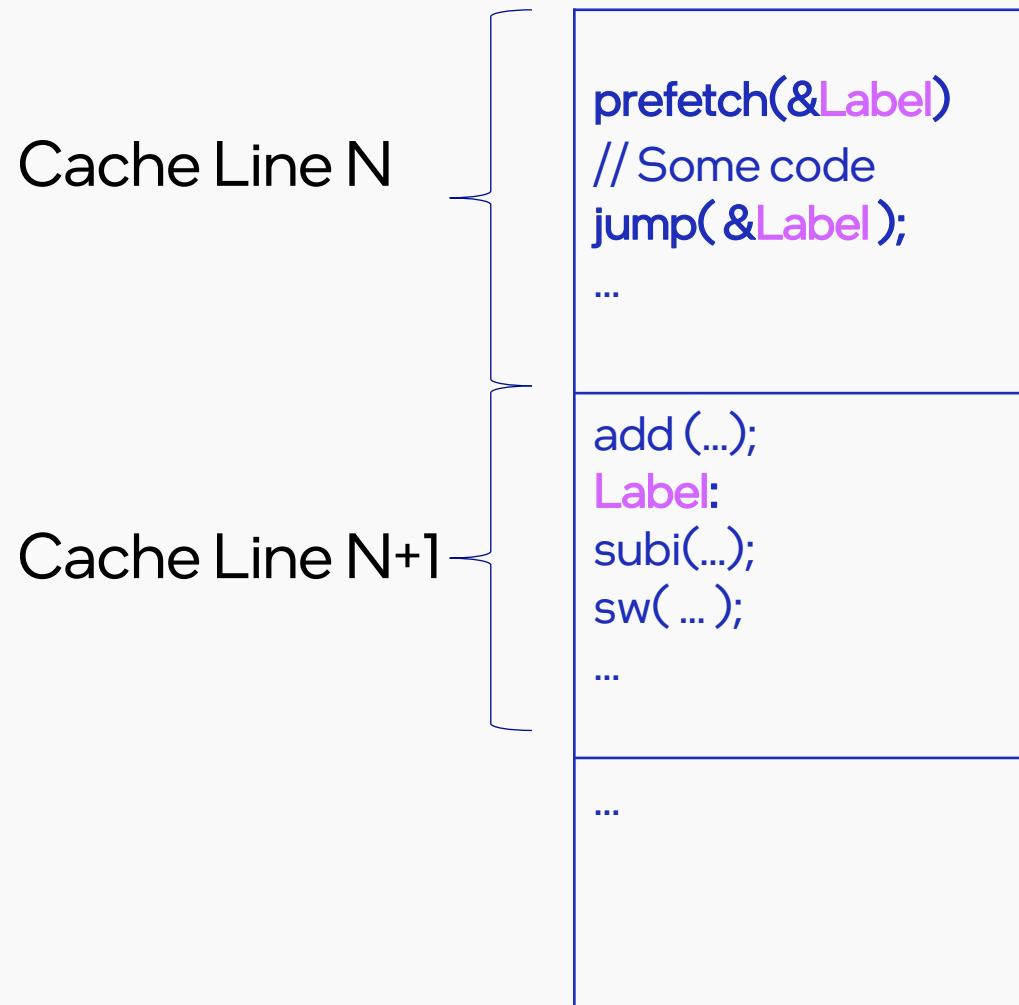
Semantics

Signals to fetch the cache line associated with Label

Sync/Async

Asynchronous non-blocking behavior

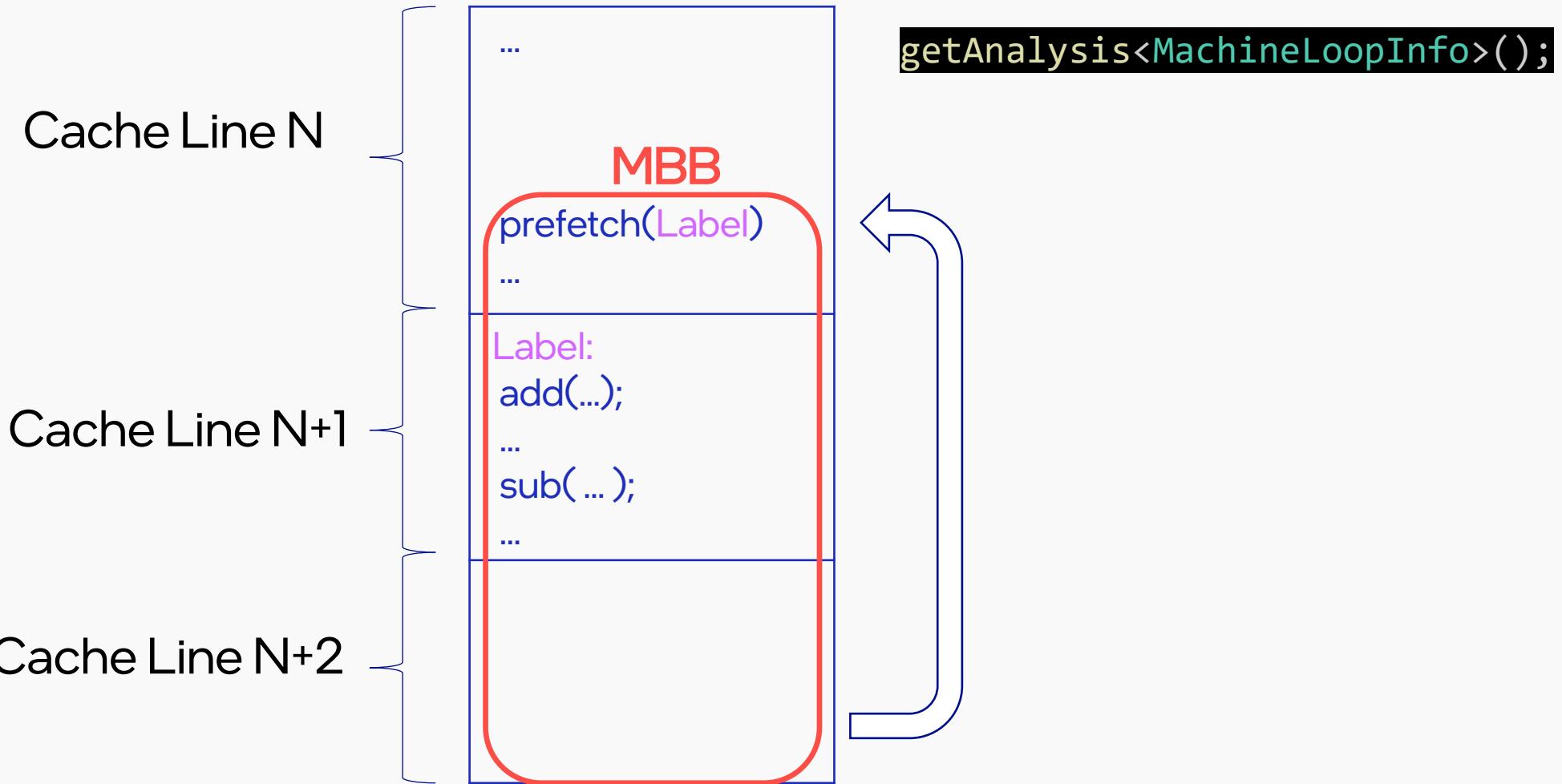
Example



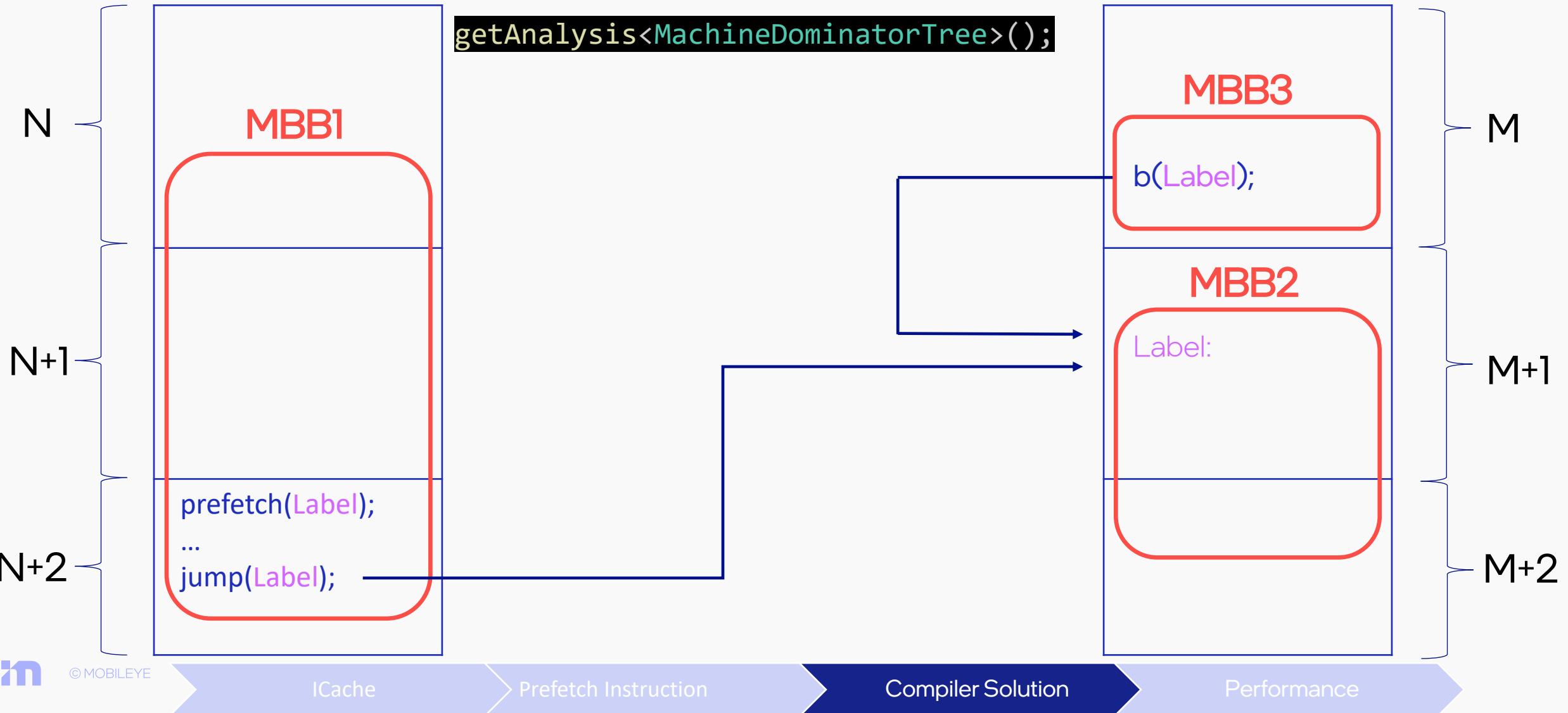
3

LLVM Analyses

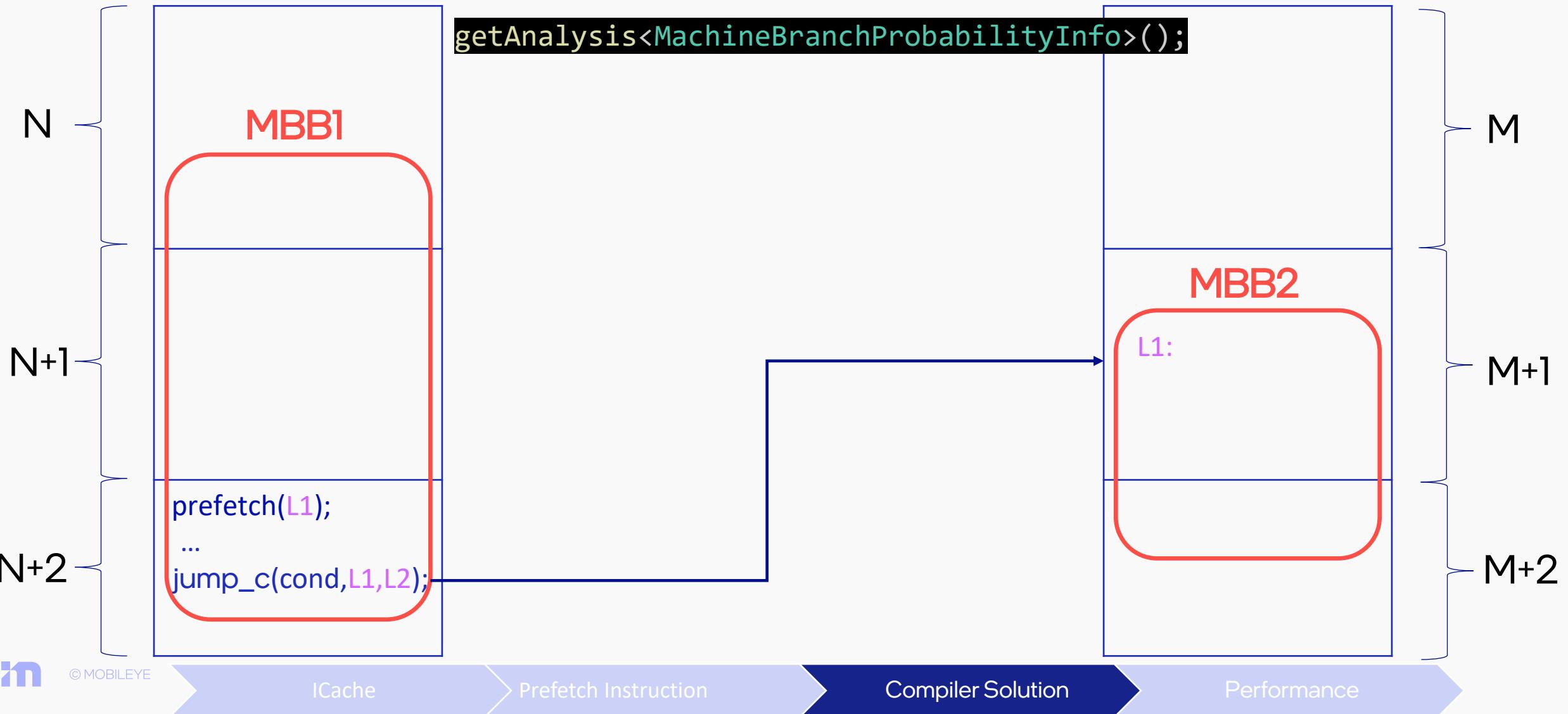
Useful Analyses



Useful Analyses



Useful Analyses





Performance

Performance Impact

I\$ Performance

45% reduction in stalls

Total Performance

3-4%

Conclusion

- Advanced architectures handle I\$ efficiently, but simpler ones don't.
- LLVM framework makes this optimization comfortable to implement.
- Compiler-driven prefetching is a viable software solution.



Thank you!

oriel.avraham@mobileye.com



Developers' Meeting

BERLIN 2025

#embed in clang: one directive to embed them all

Mariya Podchishchaeva

@Fznamznon



What is #embed?

```
# embed <file-name>|"file-name" parameters...  
parameters refers to the syntax of  
no_arg/with_arg(values,...)/vendor::no_arg/vendor::with_arg(tokens...)
```

There are language-defined parameters, for example:

```
const int data[] = {  
#embed "/dev/urandom" limit(512) // no more than 512 bytes  
};
```

P.S. clang doesn't support device files properly yet.

How is that supposed to work?

Users do:

```
const unsigned char data[] = {  
#embed "data.bin"  
};
```

The directive is expanded to comma-separated integer literals:

```
const unsigned char data[] = {  
1, 2, 3  
};
```

where 1, 2, and 3 are byte values from the resource.

How is that supposed to work?

Users do:

```
const unsigned char data[] = {  
#embed "data.bin"  
};
```

The directive is expanded to comma-separated integer literals:

```
const unsigned char data[] = {  
1, 2, 3  
};
```

where 1, 2, and 3 are byte values from the resource.



We try hard to not do exactly this. Why?

What is a ~~bug~~-big deal?

The answer is simple – this is very slow.

Let's do some comparison with “classic” methods...

```
head -c $((1024*1024*NUM_OF_MB)) /dev/urandom > file.bin  
xxd -i file.bin > filexxd.c
```

```
embed.c
```

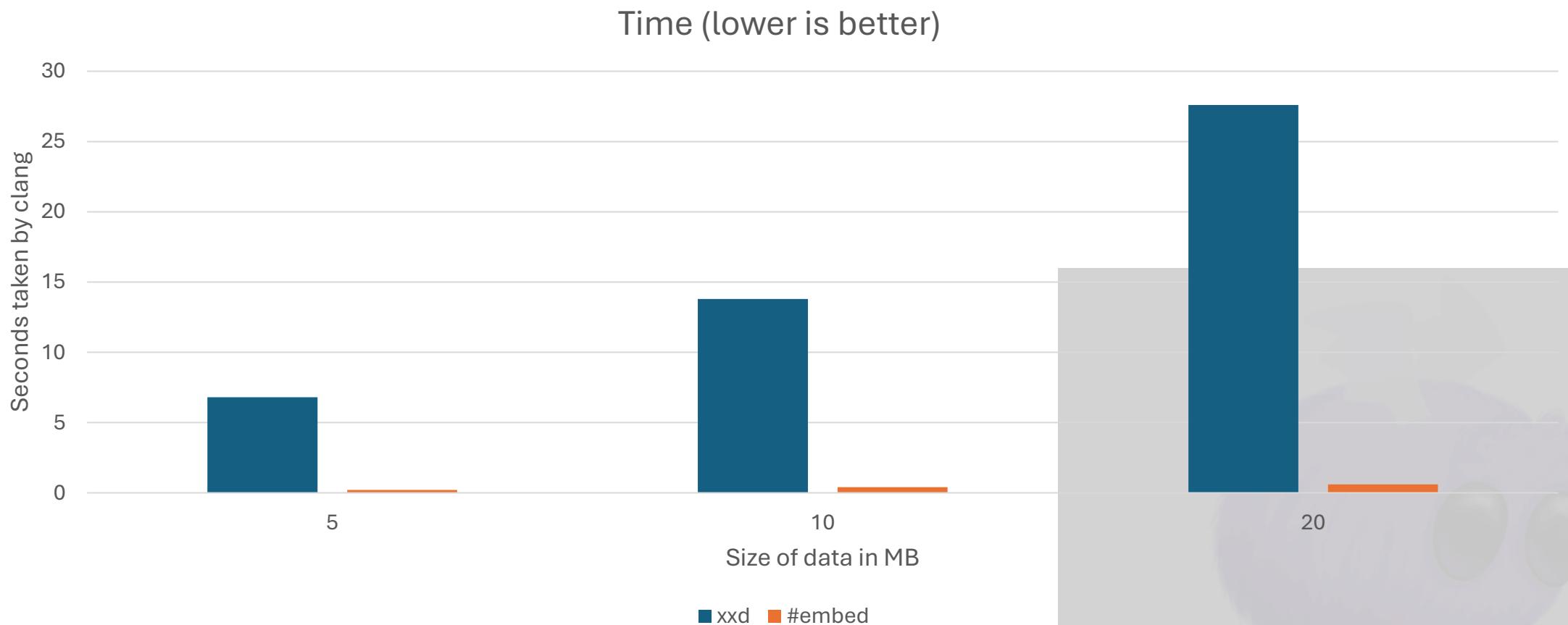
```
unsigned char c[] = {  
#embed "file.bin"  
};
```

```
filexxd.c
```

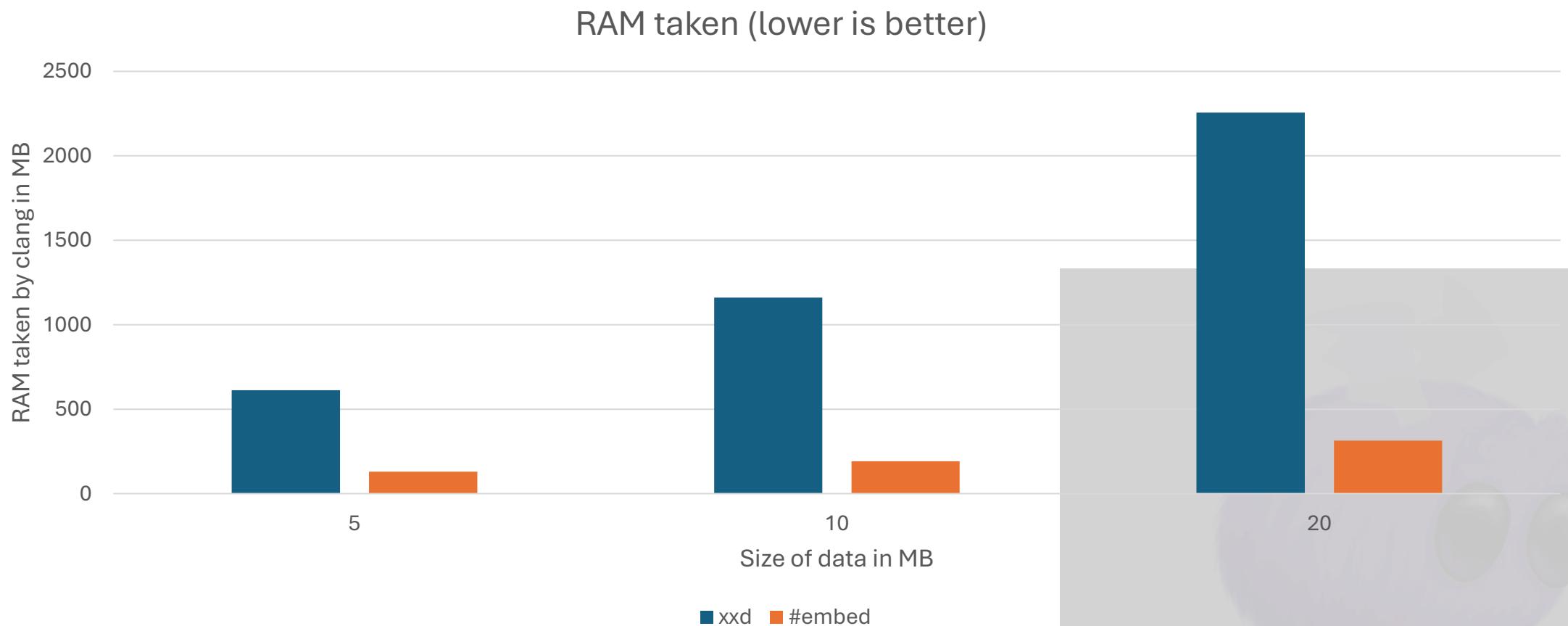
```
unsigned char file_bin[] = {  
    0x82, 0x41, 0x7c, 0xf6,  
    0x7c,...
```

And compare `clang -c -emit-llvm embed.c` vs `clang -c -emit-llvm filexxd.c`

Time difference



RAM consumption difference



How did we get there?

```
unsigned char b[] = {      `"-VarDecl <line:1:1, line:3:1> line:1:15 b
#embed __FILE__                      'unsigned char[46]' cinit
};                                     `"-InitListExpr <col:21, line:3:1> 'unsigned
                                         char[46]'

                                         `"-StringLiteral <line:2:5> 'unsigned
                                         char[46]' "unsigned char b[] = {\n      #embed
                                         __FILE__\n};\n"
```

What to do when strings don't work?

```
int a[2][3] = { 300,  
#embed __FILE__  
};
```

```
-VarDecl <line:2:1, line:4:1> line:2:5 a 'int[2][3]'  
cinit  
`-InitListExpr <col:15, line:4:1> 'int[2][3]'  
|-InitListExpr <line:3:5> 'int[3]'  
| |-array_filler: ImplicitValueInitExpr 0x334a7360  
'int'  
| `--EmbedExpr <col:5> 'int'  
| |-begin: 0  
| `--number of elements: 3  
`-InitListExpr <col:5> 'int[3]'  
|-array_filler: ImplicitValueInitExpr 0x334a7370  
'int'  
`--EmbedExpr <col:5> 'int'  
|-begin: 3  
`--number of elements: 3
```

What is EmbedExpr?

- A reference to embedded data.
- Knows where to take the data and how many of it.
- Represents multiple bytes of data with a single expression.
- One InitListExpr may have several EmbedExprs referencing the same array of data but different parts of this array.
- Created only inside of InitListExpr.
- Handled by AST consumers similarly to array filler.

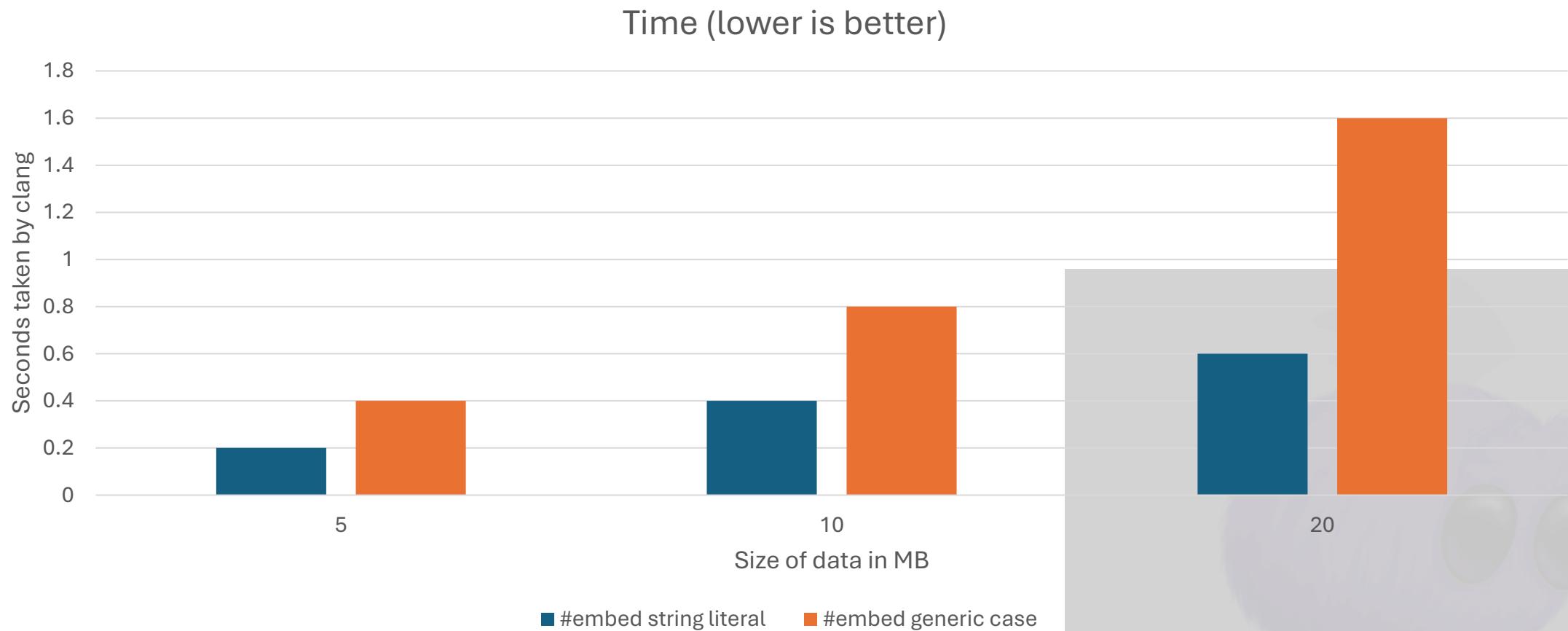
How expensive is that?

Let's check how much time and RAM clang will take with EmbedExpr and compare it to StringLiteral case.

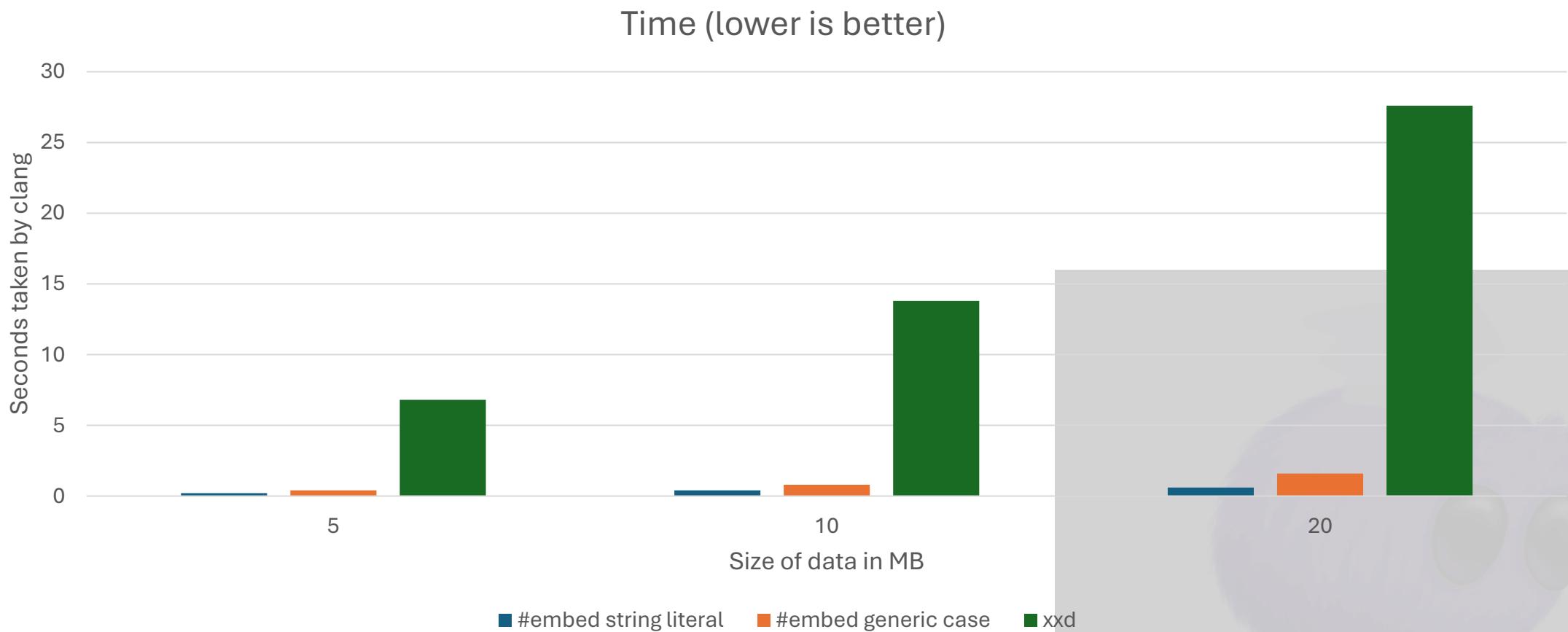
```
// Generic case
int c[] = {1,
#embed "file.bin"
};
```

```
// String literal case
unsigned char b[] = {
#embed "file.bin"
};
```

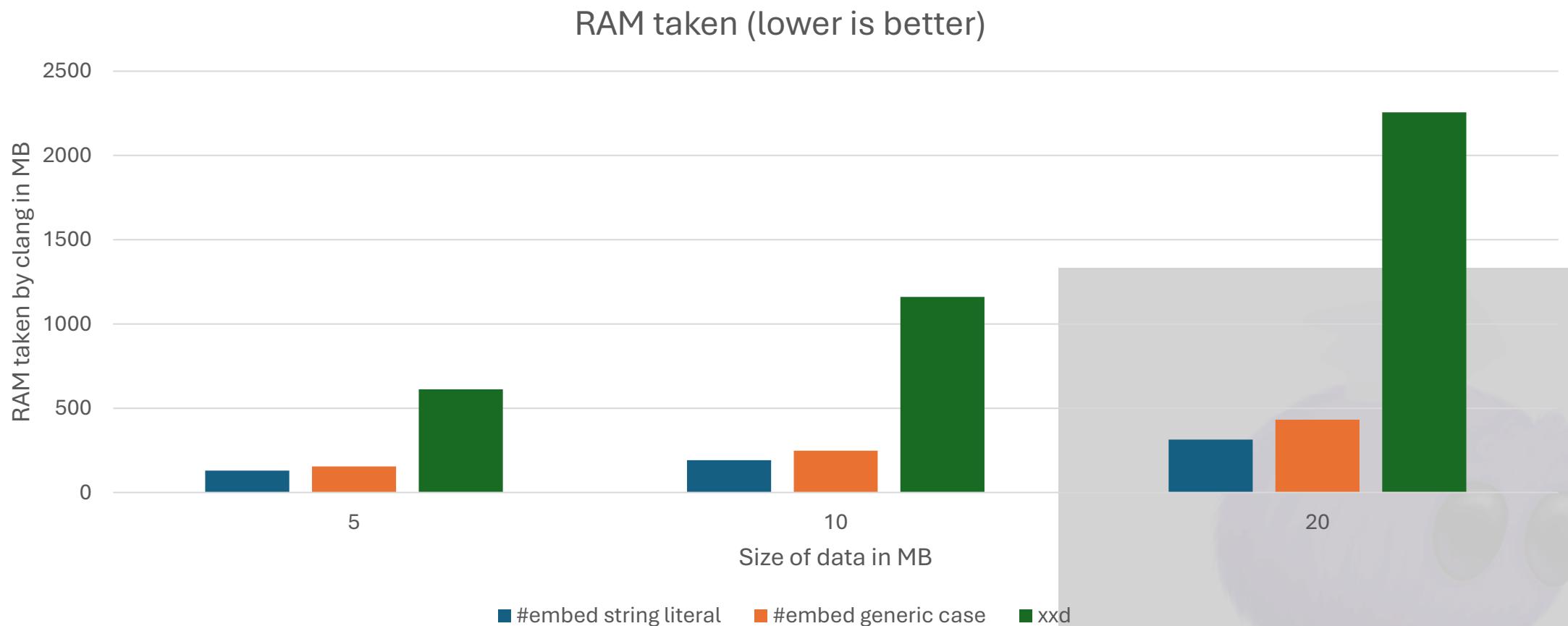
Time difference



Time difference (with xxsd)



RAM consumption difference (with xxz)



What is EmbedExpr?

- A reference to embedded data.
- Knows where to take the data and how many of it.
- Represents multiple tokens of data with a single expression.
- One InitListExpr may have several EmbedExpr referencing the same array of data but different parts of this array.
- Created only inside of InitListExpr.
- Handled by AST consumers similarly to array filler.

#embed in the wild

```
// 47 is '/'
int b = (
#embed __FILE__ limit(2)
);
`-VarDecl <line:6:1, line:8:1> line:6:5 b 'int'
cinit
`-ParenExpr <col:9, line:8:1> 'int'
`-BinaryOperator <line:7:1> 'int' ','
|-IntegerLiteral <col:1> 'int' 47
`-IntegerLiteral <col:1> 'int' 47
```

Status in clang

- Available since clang 19.
- Supported in C23, in older C modes and in C++ supported as clang extension.
- Has bugs (known and coming).
 - <https://github.com/llvm/llvm-project/labels/embed> the GitHub label for `#embed`-specific bugs.
 - <https://github.com/llvm/llvm-project/issues/95222> contains follow-up work to be done/discussed.

Backup

Machine specs

Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz

Ubuntu 24.04

400 GB RAM

#embed annotation token

```
const int self[] = {           int 'int'          [LeadingSpace]      Loc=<<source>:1:7>
    #embed __FILE__ prefix(1,) identifier 'self'      [LeadingSpace]
                                         Loc=<<source>:1:11>
};
                                l_square '['          Loc=<<source>:1:15>
                                r_square ']'         Loc=<<source>:1:16>
                                equal '='        [LeadingSpace]      Loc=<<source>:1:18>
                                l_brace '{'       [LeadingSpace]      Loc=<<source>:1:20>
                                numeric_constant '1' Loc=<<source>:2:26>
                                comma ','        Loc=<<source>:2:27>
                                annot_embed        Loc=<<source>:2:3>
                                r_brace '}'        Loc=<<source>:3:1>
                                semi ';'        Loc=<<source>:3:2>
```

Implementation challenges

- Performance.
 - `#embed` is easy to implement so it conforms to the standard, yet it is hard to make it effective.
- Corner cases of it being a preprocessor directive.
 - Can output multiple tokens per byte of data. Need to make sure all places where comma-separated list can appear handle `#embed` data.
- Preprocessed output.
 - -E output can get huge because of `#embed`.
 - Security concerns.

Why `#embed`?

- Gets binary content easily into applications.
- Platform independent, portable.
- Allows to include data as a constant expression.
- File search mechanism works like well-known `#include` directive.
- An `#embed` directive can be used in any place where a single integer or comma-separated list of integer literals is acceptable.
- Part of C23 standard, accepted in C++26.



Developers' Meeting

BERLIN 2025



LLVM_ENABLE_RUNTIMES=flang-rt

EuroLLVM 2025

Michael Kruse

Advanced Micro Devices GmbH

15th April, 2025

Outline

1 Introduction

2 Legacy Flang Runtime

3 New Flang-RT

4 Remaining Work



Outline

1 Introduction

- Usage
- The Mechanism
- Advanced Options

2 Legacy Flang Runtime

3 New Flang-RT

4 Remaining Work



LLVM_ENABLE_RUNTIMES?

Bootstrapping-Runtimes build

```
cmake ..../llvm -GNinja  
"-DLLVM_ENABLE_PROJECTS=clang;lld;polly"  
"-DLLVM_ENABLE_RUNTIMES=compiler-rt;libc;libcxx;openmp"
```

\
\

LLVM_ENABLE_PROJECTS

- Compiled using host compiler (e.g. GCC)
- Same CMake build-dir as LLVM itself
- Intended to run on the host architecture

LLVM_ENABLE_RUNTIMES

- Compiled using just-built Clang
- Use separate CMake build-dir nested inside LLVM build-dir
- Intended to be used by binaries compiled by Clang
 - Can be a different architecture (cross-compilation)



How does it work?

CMake step

LLVM_ENABLE_PROJECTS

- 1 For each enabled project,

```
add_subdirectory(llvm-sourcedir/<project>)
```

LLVM_ENABLE_RUNTIMES

- 1 Add build targets:

```
runtimes, install-runtimes, <runtime>, check-<runtime>,  
install-<runtime>, ...
```

- 2 For each target architecture,

```
llvm_ExternalProject_Add(runtimes ...)  
    which executes
```

```
cmake -GNinja  
    -S llvm-sourcedir/runtimes  
    -B llvm-build-dir/runtimes/runtimes-bins  
    -DLLVM_BINARY_DIR=llvm-build-dir  
    -DCMAKE_{C,CXX}_COMPILER=llvm-build-dir/bin/clang{++}  
    -DCMAKE_{C,CXX}_COMPILER_WORKS=YES  
    "-DLLVM_ENABLE_RUNTIMES=<runtimes>"
```

- 1 find_package(LLVM), find_package(Clang)
- 2 Find tools such as llvm-lit, FileCheck in llvm-build-dir
- 3 For each enabled runtime,
 add_subdirectory(llvm-sourcedir/<runtime>)

How does it work?

Ninja step

LLVM_ENABLE_PROJECTS: `ninja <project>`

- 1 CMake ensured all necessary dependencies

LLVM_ENABLE_RUNTIMES: `ninja <runtime>`

- 1 Build dependencies such as clang, FileCheck, etc
- 2 Run configure step for runtimes
- 3 Build dependencies for runtimes
- 4 Execute

```
ninja -C llvm-build-dir/runtimes/runtime-bins <runtime>
```

- 1 Build selected runtime



Runtime Options

Pass Options to Runtimes Build

```
cmake ...  
  -DCMAKE_CXX_FLAGS=-fmax-errors=1  
  "-DRUNTIMES_CMAKE_ARGS=-DCMAKE_CXX_FLAGS=-ferror-limit=1"
```

- -fmax-errors=1 is for gcc
- -ferror-limit=1 is for clang

Multiarch & Pass Arch-specific Options

```
cmake ...  
  "-DLLVM_RUNTIME_TARGETS=default;aarch64-linux-gnu"  
  "-DRUNTIMES_aarch64-linux-gnu_CMAKE_CXX_FLAGS=-march=cortex-a57"
```

- Will create one builddir per target

Standalone-Runtimes build

CMake step

```
cmake -GNinja llvm-srcdir/runtimes  
      -DLLVM_BINARY_DIR=llvm-buildDir  
      "-DLLVM_ENABLE_RUNTIMES=<runtimes>"\n      \\\\"
```

- Projects (LLVM, Clang, ...) compiled separately
- Uses default C/C++ compiler (e.g. gcc)

Ninja step

```
ninja <runtime>  
ninja check-<runtime>
```



Outline

1 Introduction

2 Legacy Flang Runtime

- Usage
- The Problem

3 New Flang-RT

4 Remaining Work



Legacy Flang Runtime

flang/runtime/CMakeLists.txt

In-Tree build

- Like a LLVM_ENABLE_RUNTIMES build:

```
add_subdirectory(runtime)
```

Standalone build

```
cmake llvm-srcdir/flang/runtime \
-DLLVM_DIR=... \
-DCLANG_DIR=... \
-DMLIR_DIR=...
```

```
cmake llvm-srcdir/flang/lib/Decimal \
-DLLVM_DIR=... \
-DCLANG_DIR=... \
-DMLIR_DIR=...
```



What is the Problem?

- Inconsistent with other LLVM runtimes
- For cross-compilation targets, must compile each target in standalone build
- GPU offloading: Build auxiliary target runtime separately
 - As done for openmp-offload, libc, compiler-rt, libcxx, ...
- Standalone build does not include `iso_fortran_env_impl.f90`
- Source code shared with Flang and Runtime
 - ABI assumed to be the same
 - Compile code and runtime code have different requirements
 - E.g. runtime code must not link to C++ standard library
 - No clear separation which file belongs where
- Compiled binary shared with Flang and Runtime
 - Runtime built with different flags
 - E.g. `-fno-lto`



Outline

1 Introduction

2 Legacy Flang Runtime

3 New Flang-RT

- Usage
- The Difficulties
- The Changes

4 Remaining Work



Building Flang-RT

Bootstrapping-Runtimes build

```
cmake -GNinja .. llvm \
"-DLLVM_ENABLE_PROJECTS=clang;mlir;flang" \
"-DLLVM_ENABLE_RUNTIMES=flang-rt"
```

Standalone-Runtimes build

```
cmake -GNinja llvm-srcdir/runtimes \
"-DLLVM_ENABLE_RUNTIMES=flang-rt" \
"-DLLVM_BINARY_DIR=llvm-buildDir" \
"-DCMAKE_Fortran_COMPILER=llvm-buildDir/bin/flang" \
"-DCMAKE_Fortran_COMPILER_WORKS=YES"
```

- Flang must built from the same git SHA1
 - No ABI contract
- CMAKE_Fortran_COMPILER_WORKS because flang before the runtime is available cannot produce executables

Things that Must Continue Working

- Shared library
- Quad-precision `math.h` support
 - gcc `libquadmath`
 - Native `sizeof(long double) == 16` with `libm`
 - f128 suffix functions (like `sinf128`) in `libm`
- Conditional `REAL(16)` support in Flang
- Unittests
 - GTest and “non-gtest” testing framework
- Windows static .lib
 - LLVM emits libgcc-ABI function calls, requires `clang_rt.builtins.lib` at link-time
 - msvc ships `clang_rt.builtins-x86_64.a`, but not used by the driver (anymore)
- Experimental OpenMP-offload build
- Experimental CUDA build
 - With `clang -x cuda` and `nvcc`



Library Names

Old Library Names

- libFortranRuntime{.a,.so}
- libFortranDecimal{.a,.so}
- libFortranFloat128Math.a
- libCufRuntime_cuda_\${version}{.a,.so}

New Library Names

- libflang_rt.runtime{.a,.so}
- libflang_rt.quadmath.a
- libflang_rt.cuda_\${version}{.a,.so}

- Same scheme as Compiler-RT: libclang_rt.<component>{.a,.so}
- libFortranDecimal integrated into libflang_rt.runtime
 - Decided by RFC
 - libFortranCommon also used by Flang
 - Made flang depend on libcudart.so



The Big Move

Principles

- Split some headers into a compiler- and a runtime part
- Definitions to flang-rt/lib/\$component/*.cpp
- Non-private headers to flang-rt/include/flang-rt/\$component/*.h
- Files used by both, Flang (the compiler) and Flang-RT (the runtime), remain in flang/
- Move “Common” files only used by Flang (the compiler) to Support
 - Remaining shared components: FortranDecimal, FortranCommon (header-only), FortranRuntime (header-only), FortranTesting

Old	New
flang/runtime/*.h	flang-rt/include/runtime/*.h
flang/runtime/*.cxx	flang-rt/lib/runtime/*.cpp
flang/runtime/Float128Math/*	flang-rt/lib/quadmath/*
flang/runtime/CUDA/*	flang-rt/lib/cuda/*
flang/include/flang/Runtime/*.h	flang/include/flang/Runtime/*.h
flang/include/flang/Common/*.h ¹	flang/include/flang/Support/*.h
flang/unittests/Evaluate/{fp-}testing.h	flang/include/flang/Testing/*.h
flang/lib/Common/*.cpp	flang/lib/Support/*.cpp
flang/unittests/Evaluate/{fp-}testing.cpp	flang/lib/Testing/*.cpp
flang/test/**/* ²	flang-rt/test/**/*.cpp



LLVM_ENABLE_PER_TARGET_RUNTIME_DIR

Library Location

- Old Flang Runtime:

`$(CMAKE_INSTALL_PREFIX)/lib/libflang_rt.runtime.a`

- Clash in multiarch/cross-compile scenarios

- LLVM_ENABLE_PER_TARGET_RUNTIME_DIR=OFF:

`$(CMAKE_INSTALL_PREFIX)/lib/clang/$version/lib/$os/libclang_rt.buildins-$arch.a`

- Windows, Apple, AIX

- LLVM_ENABLE_PER_TARGET_RUNTIME_DIR=ON:

`$(CMAKE_INSTALL_PREFIX)/lib/clang/$version/lib/$triple/libclang_rt.builtins.a`

- Became default for Linux in Clang 19 (Now also BSD, OS390)

- Assumptions leaking into LLVM_ENABLE_PER_TARGET_RUNTIME_DIR=OFF as well 😞

- Only the last supported for flang-rt

- `$(CMAKE_INSTALL_PREFIX)/lib/clang/$version/lib/$triple/libflang_rt.<component>{.a,.so}`

- LLVM_ENABLE_PER_TARGET_RUNTIME_DIR ignored



Shared Library

- Old scheme: `BUILD_SHARED_LIBS=ON`
 - Requires a second standalone build
- New scheme: Build static+shared library at the same time using *object libraries*
 - Done by (almost) every other runtime

CMake Options

- `FLANG_RT_ENABLE_STATIC`
 - Default: ON
- `FLANG_RT_ENABLE_SHARED`
 - Default: OFF
 - Id prefers .so over .a, enabling it would be a breaking change



Experimental GPU Target Support

CUDA

- **clang**: Compile everything with `-x cuda`
- **nvcc**: Treats everything as CUDA source
- Requires `libcudac++` (`libc++` for CUDA), cannot use `<variant>` or `<optional>`
- Declarations must be annotated with `__host__ __device__`

OpenMP

- Compile everything with `-fopenmp --offload-arch`
- Declarations must be annotated with `#pragma declare target`
- Annotations selected with preprocessor macro `RT_API_ATTRS`
- Results in a *fat library*
 - Host and device code in a single file
 - For AMD/OpenMP we would rather compile them separately
 - Multiarch library with device code looked up when launching kernels



Outline

1 Introduction

2 Legacy Flang Runtime

3 New Flang-RT

4 Remaining Work
■ TODOs



To Do Items

- Flang is not (yet) a cross-compiler
Sometimes assumes ABI of host platform, e.g. `sizeof(long)`
- Compile builtin modules in the runtimes build using CMake
 - OpenMP's modules as well
 - Per-target modules
- Flang's Quadmath support must not depend on LLVM build environment
- Multilib support
- Shared library location
- Library versioning



AMD



Developers' Meeting

BERLIN 2025



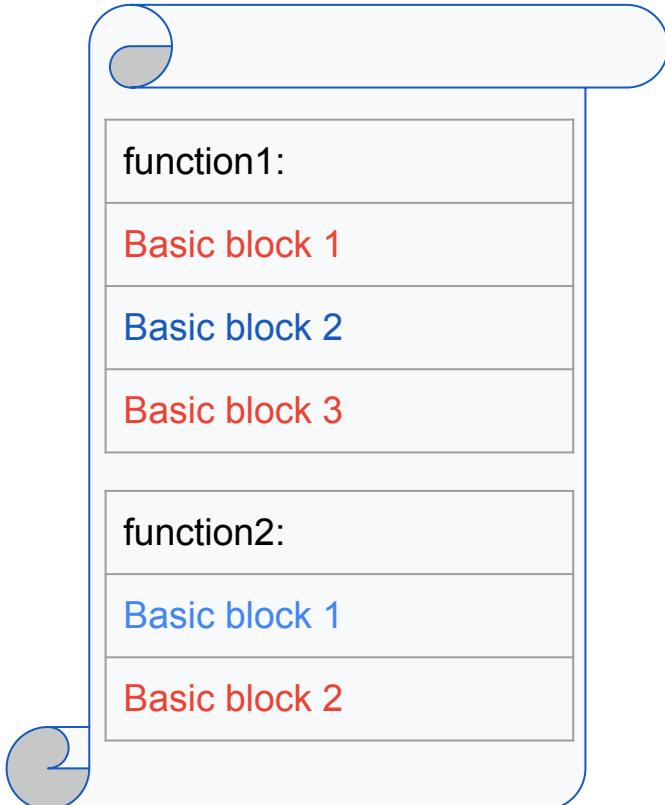
LLDB support for Propeller optimized code

(things programmers believe about functions)

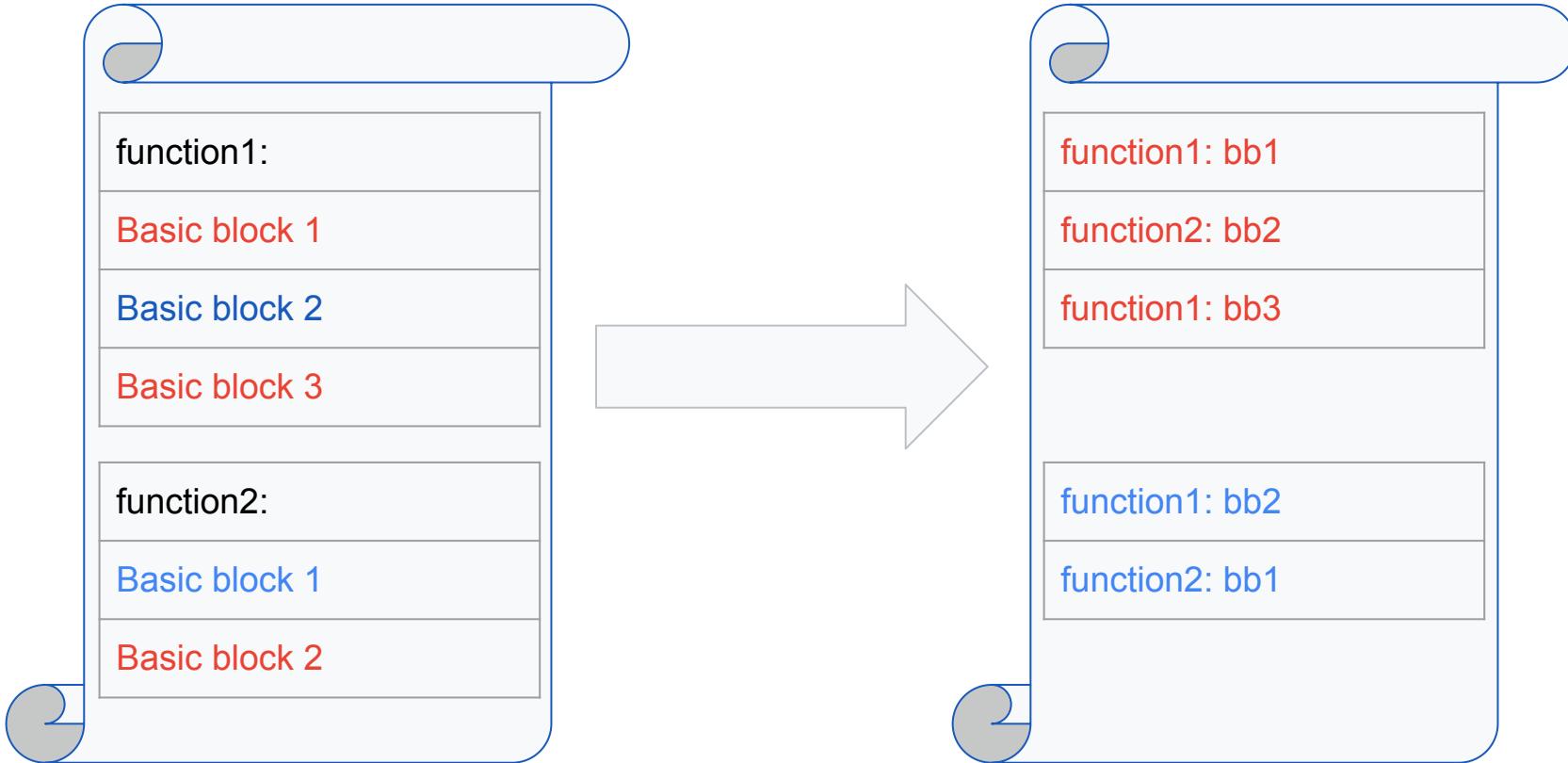


Pavel Labath

Propeller (Bolt, etc.)



Propeller (Bolt, etc.)



Debug info (DWARF)

```
DW_TAG_subprogram
  DW_AT_name    ("function1")
  DW_AT_low_pc  (0x00001000)
  DW_AT_high_pc
(0x00001030)
  ...
DW_TAG_subprogram
  DW_AT_name    ("function2")
  DW_AT_low_pc  (0x00002000)
  DW_AT_high_pc
(0x00002020)
  ...
  ...
```

Debug info (DWARF)

```
DW_TAG_subprogram
```

```
  DW_AT_name      ("function1")
```

```
  DW_AT_low_pc   (0x00001000)
```

```
  DW_AT_high_pc
```

```
(0x00001030)
```

```
...
```

```
DW_TAG_subprogram
```

```
  DW_AT_name      ("function2")
```

```
  DW_AT_low_pc   (0x00002000)
```

```
  DW_AT_high_pc
```

```
(0x00002020)
```

```
...
```



```
DW_TAG_subprogram
```

```
  DW_AT_name      ("function1")
```

```
  DW_AT_ranges   (
```

```
    [0x00001000, 0x00001010)
```

```
    [0x00002000, 0x00002010)
```

```
    [0x00001020, 0x00001030) )
```

```
...
```

```
DW_TAG_subprogram
```

```
  DW_AT_name      ("function2")
```

```
  DW_AT_ranges   (
```

```
    [0x00002010, 0x00002020)
```

```
    [0x00001010, 0x00001020) )
```

```
...
```

The problem

```
class Function {  
    const AddressRange &GetAddressRange() { return m_range; }  
    ...  
};  
  
function1.GetAddressRange() = [0x1000, 0x2010)  
function2.GetAddressRange() = [0x1010, 0x2020)
```

The problem

```
class Function {  
    const AddressRange &GetAddressRange() { return m_range; }  
    ...  
};  
  
function1.GetAddressRange() = [0x1000, 0x2010)  
function2.GetAddressRange() = [0x1010, 0x2020)  
  
function1.GetAddressRange().GetBaseAddress() = 0x1000  
function2.GetAddressRange().GetBaseAddress() = 0x1010
```

The problem

```
class Function {  
    const AddressRange &GetAddressRange() { return m_range; }  
    ...  
};  
  
function1.GetAddressRange() = [0x1000, 0x2010)  
function2.GetAddressRange() = [0x1010, 0x2020)  
  
function1.GetAddressRange().GetBaseAddress() = 0x1000  
function2.GetAddressRange().GetBaseAddress() = 0x1010  
  
class SymbolContext {  
    bool GetAddressRange(uint32_t scope, uint32_t range_idx,  
        bool use_inline_block_range, AddressRange &range) const;  
};
```

Solution

```
class Function {
    AddressRanges GetAddressRanges() { return m_block.GetRanges(); }
    const Address &GetAddress() const { return m_address; }
};

class SymbolContext {
    bool GetAddressRange(uint32_t scope, uint32_t range_idx,
        bool use_inline_block_range, AddressRange &range) const;
    Address GetFunctionOrSymbolAddress() const;
};
```

Recommendations

- Do not assume that functions are contiguous

Recommendations

- Do not assume that functions are contiguous
- Do not assume that all parts (address ranges) of the function are within a single section
 - ```
if (addr1.GetSection() == addr2.GetSection())
 return addr1.GetOffset() - addr2.GetOffset();
```
  - ```
if (addr1.GetModule() == addr2.GetModule())  
    return addr1.GetFileAddress() - addr2.GetFileAddress();
```

Recommendations

- Do not assume that functions are contiguous
- Do not assume that all parts (address ranges) of the function are within a single section
 - ```
if (addr1.GetSection() == addr2.GetSection())
 return addr1.GetOffset() - addr2.GetOffset();
```
  - ```
if (addr1.GetModule() == addr2.GetModule())  
    return addr1.GetFileAddress() - addr2.GetFileAddress();
```
- Do not assume that a function is described by a single `eh_frame` record (line table sequence, etc.)

Recommendations

- Do not assume that functions are contiguous
- Do not assume that all parts (address ranges) of the function are within a single section
 - ```
if (addr1.GetSection() == addr2.GetSection())
 return addr1.GetOffset() - addr2.GetOffset();
```
  - ```
if (addr1.GetModule() == addr2.GetModule())  
    return addr1.GetFileAddress() - addr2.GetFileAddress();
```
- Do not assume that a function is described by a single `eh_frame` record (line table sequence, etc.)
- Do not assume that function entry point is its lowest address

Recommendations

- Do not assume that functions are contiguous
- Do not assume that all parts (address ranges) of the function are within a single section
 - ```
if (addr1.GetSection() == addr2.GetSection())
 return addr1.GetOffset() - addr2.GetOffset();
```
  - ```
if (addr1.GetModule() == addr2.GetModule())  
    return addr1.GetFileAddress() - addr2.GetFileAddress();
```
- Do not assume that a function is described by a single `eh_frame` record (line table sequence, etc.)
- Do not assume that function entry point is its lowest address
 - Corollary: Do not assume that offsets from the entry point are positive

Current state

- Most things “just work”
- Main exception: unwinding

Current state

- Most things “just work”
- Main exception: unwinding
- There are rough edges:

```
(lldb) disassemble --name foo  
my_binary`foo:  
0xdead00 <-1179648>: cmpl    $0x0, %eax  
0xdead03 <-1179645>: jmp     0xdead33          ; <-1179597>  
...
```

Thank you



Developers' Meeting

BERLIN 2025

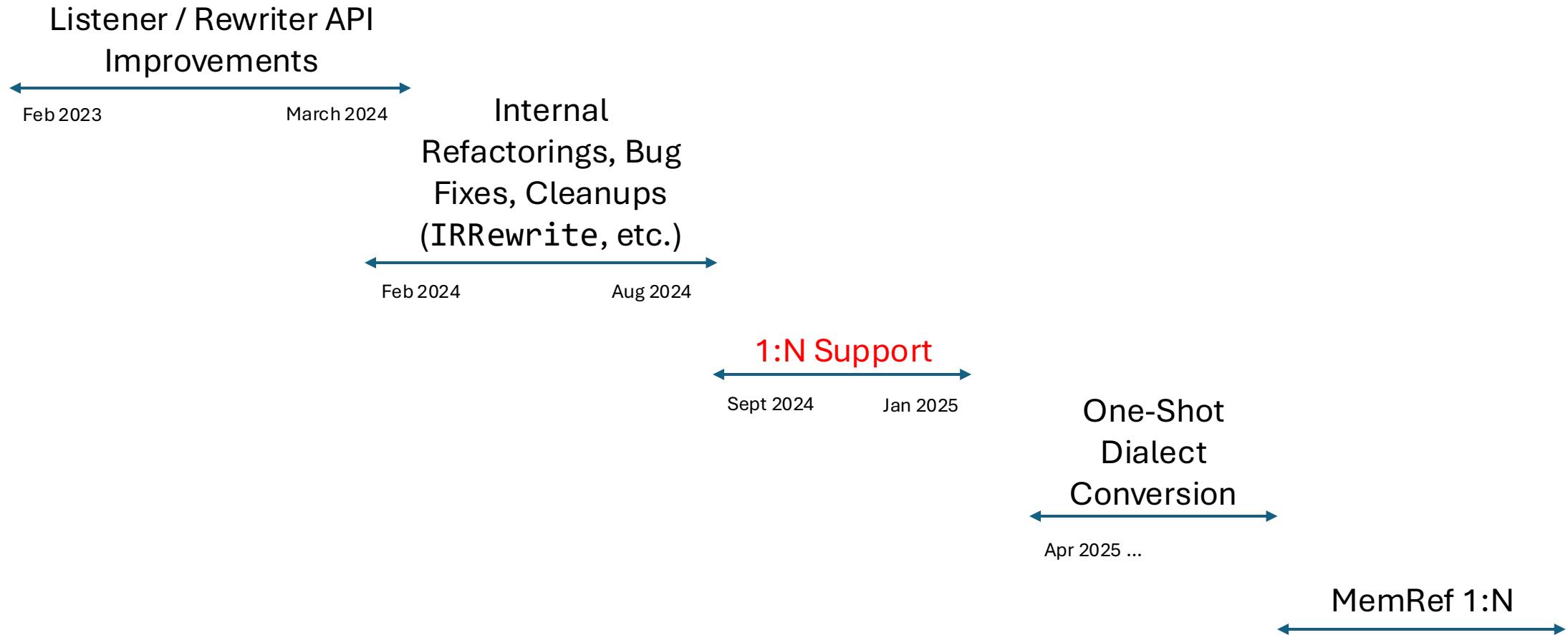


1:N Dialect Conversion

Matthias Springer (NVIDIA)

EuroLLVM 2025 – Quick Talk – April 15, 2025

Timeline



Dialect Conversion

- One of the two main pattern drivers in MLIR, more powerful API than “regular” rewrite API ([PatternRewriter](#))

PatternRewriter	ConversionPatternRewriter
<code>replaceOp(Operation *, ValueRange);</code>	<code>replaceOp(Operation *, ValueRange);</code>
a combination of: <code>createBlock</code> , <code>inlineBlockBefore</code> , <code>create</code> <code>unrealized_conversion_cast ops</code> , <code>replaceAllUsesWith</code> , <code>eraseBlock</code>	<code>applySignatureConversion(Block *, SignatureConversion &, TypeConverter &);</code>

Dialect Conversion

- One of the two main pattern drivers in MLIR, more powerful API than “regular” rewrite API ([PatternRewriter](#))

PatternRewriter	ConversionPatternRewriter
<code>replaceOp(Operation *, ValueRange);</code>	<code>replaceOp(Operation *, ValueRange);</code>
a combination of: <code>createBlock</code> , <code>inlineBlockBefore</code> , <code>create</code> <code>unrealized_conversion_cast ops</code> , <code>replaceAllUsesWith</code> , <code>eraseBlock</code>	<code>applySignatureConversion(Block *, SignatureConversion &, TypeConverter &);</code> <div data-bbox="1446 1224 2304 1396" style="background-color: #2e6b2e; color: white; padding: 10px; border-radius: 10px;"><p>For each block argument, specifies the new type in the converted block.</p></div>

Dialect Conversion

- One of the two main pattern drivers in MLIR, more powerful API than “regular” rewrite API ([PatternRewriter](#))
- What’s new: 1:N op replacements, 1:N conversion patterns

PatternRewriter	ConversionPatternRewriter
<code>replaceOp(Operation *, ValueRange);</code>	<code>replaceOp(Operation *, ValueRange);</code> <code>replaceOpWithMultiple(Operation *, ArrayRef<ValueRange>);</code>
a combination of: <code>createBlock</code> , <code>inlineBlockBefore</code> , <code>create</code> <code>unrealized_conversion_cast</code> ops, <code>replaceAllUsesWith</code> , <code>eraseBlock</code>	<code>applySignatureConversion(Block *, SignatureConversion &, TypeConverter &);</code> <div style="background-color: #2e6b2e; color: white; padding: 5px; text-align: center;"><p>For each block argument, specifies the new types in the converted block.</p></div>

Outline of This Work

- **Cleanup** of duplicate + non-composable frameworks:
 - Incomplete implementation: 1:N used to be partially supported in the dialect conversion framework (only signature conversions).
 - There is a separate 1:N dialect conversion framework that supports 1:N op replacements, but it's not really a dialect conversion.
- Why is this important?
 - 1:N conversions appear in **load-bearing + performance critical** lowering passes of real-world compilers.
 - MLIR: MemRef → LLVM Lowering
 - MLIR: Sparse Tensor → Sparse Tensor Descriptor (codegen path)
 - Triton: Tile → SIMT
 - Multiple NVIDIA-internal projects: cuTile, Cutlass, ...
 - These passes must resort to packing/unpacking to work around dialect conversion limitations. That's **inefficient** (increases compilation time) and makes code/IR **complex**.

Example: MemRef → LLVM

Example: 1:1 MemRef → LLVM Lowering

```
memref<?x?xf32, strided<[?, ?], offset: ?>>
```

1 SSA value → 1 SSA value

```
!llvm.struct<(!llvm.ptr, !llvm.ptr,           // ptr  
              i64,                      // offset  
              !llvm.struct<(i64, i64)>, // sizes  
              !llvm.struct<(i64, i64)>) // strides
```

Example: 1:N MemRef → LLVM Lowering

memref<?x?xf32, strided<[?, ?], offset: ?>>

1 SSA value → 7 SSA values

!llvm.ptr, !llvm.ptr,	// ptr
i64,	// offset
i64, i64	// sizes
i64, i64	// strides

Example: 1:1 MemRef → LLVM Lowering

memref<*xf32>

1 SSA value → 1 SSA value

```
!llvm.struct<(!llvm.ptr,    // ptr to descriptor  
              i64)>      // rank
```

Example: 1:N MemRef → LLVM Lowering

memref<*xf32>

1 SSA value → 2 SSA values

llvm.ptr, // ptr to descriptor
i64 // rank

Test Case

```
// RUN: mlir-opt %s -expand-strided-metadata -convert-to-Llvm

func.func @test_case(%m: memref<5xf32>, %offset: index) -> f32 {
    %c1 = arith.constant 1 : index
    %0 = memref.subview %m[%offset][2][1] : memref<5xf32> to memref<2xf32, strided<[1], offset: ?>>
    %1 = memref.load %0[%c1] : memref<2xf32, strided<[1], offset: ?>>
    return %1 : f32
}
```

Test Case

```
// RUN: mlir-opt %s -convert-to-llvm

func.func @test_case(%m: memref<5xf32>, %offset: index) -> f32 {
    %c1 = arith.constant 1 : index
    %base_buffer, %offset, %sizes, %strides = memref.extract_strided_metadata %m
        : memref<5xf32> -> memref<f32>, index, index, index
    %reinterpret_cast = memrefreinterpret_cast %base_buffer to offset: [%offset], sizes: [2], strides: [1]
        : memref<f32> to memref<2xf32, strided<[1], offset: ?>>
    %0 = memref.load %reinterpret_cast[%c1] : memref<2xf32, strided<[1], offset: ?>>
    return %0 : f32
}
```

Test Case: 1:1 Lowering (current lowering)

```
llvm.func @test_case(%arg0: !llvm.ptr, %arg1: !llvm.ptr,
                     %arg2: i64, %arg3: i64, %arg4: i64,
                     %arg5: i64) -> f32 {
  %0 = llvm.mlir.poison
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
  %1 = llvm.insertvalue %arg0, %0[0]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
  %2 = llvm.insertvalue %arg1, %1[1]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
  %3 = llvm.insertvalue %arg2, %2[2]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
  %4 = llvm.insertvalue %arg3, %3[3, 0]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
  %5 = llvm.insertvalue %arg4, %4[4, 0]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
  %6 = llvm.mlir.constant(1 : index) : i64
  %7 = llvm.extractvalue %5[0]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
  %8 = llvm.extractvalue %5[1]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
  %9 = llvm.mlir.poison : !llvm.struct<(ptr, ptr, i64)>
  %10 = llvm.insertvalue %7, %9[0] : !llvm.struct<(ptr, ptr, i64)>
  %11 = llvm.insertvalue %8, %10[1] : !llvm.struct<(ptr, ptr, i64)>
  %12 = llvm.mlir.constant(0 : index) : i64
  %13 = llvm.insertvalue %12, %11[2] : !llvm.struct<(ptr, ptr, i64)>
  %14 = llvm.extractvalue %5[2]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
  %15 = llvm.extractvalue %5[3, 0]
```

```
: !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
%16 = llvm.extractvalue %5[4, 0]
: !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
%17 = llvm.mlir.poison
: !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
%18 = llvm.extractvalue %13[0] : !llvm.struct<(ptr, ptr, i64)>
%19 = llvm.extractvalue %13[1] : !llvm.struct<(ptr, ptr, i64)>
%20 = llvm.insertvalue %18, %17[0]
: !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
%21 = llvm.insertvalue %19, %20[1]
: !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
%22 = llvm.insertvalue %arg5, %21[2]
: !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
%23 = llvm.mlir.constant(2 : index) : i64
%24 = llvm.insertvalue %23, %22[3, 0]
: !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
%25 = llvm.mlir.constant(1 : index) : i64
%26 = llvm.insertvalue %25, %24[4, 0]
: !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
%27 = llvm.extractvalue %26[1]
: !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
%28 = llvm.extractvalue %26[2]
: !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>)%
%29 = llvm.getelementptr %27[%28] : (!llvm.ptr, i64) -> !llvm.ptr, f32
%30 = llvm.getelementptr %29[%6] : (!llvm.ptr, i64) -> !llvm.ptr, f32
%31 = llvm.load %30 : !llvm.ptr -> f32
llvm.return %31 : f32
}
```

Test Case: 1:1 Lowering (current lowering)

```

llvm.func @test_case(%arg0: !llvm.ptr, %arg1: !llvm.ptr,
                     %arg2: i64, %arg3: i64, %arg4: i64,
                     %arg5: i64) -> f32 {

%0 = llvm.mlir.poison
  : !llvm.struct<(ptr, ptr, i64, target materialization: pack
%1 = llvm.insertvalue %arg0, %0[0]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>>
%2 = llvm.insertvalue %arg1, %1[1]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>>
%3 = llvm.insertvalue %arg2, %2[2]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>>
%4 = llvm.insertvalue %arg3, %3[3, 0]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>>
%5 = llvm.insertvalue %arg4, %4[4, 0]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>>
%6 = llvm.mlir.constant(1 : index) : i64
%7 = llvm extractvalue %5[0]
  : !llvm ExtractStridedMetadataOpLowering: unpack
%8 = llvm.extractvalue %5[1]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>>
%9 = llvm.mlir.poison ; !llvm.struct<(ptr, ptn, i64>
%10 = llvm insertvalue %7[0] ; !llvm.struct<(ptr, ptn, i64>
  : !llvm ExtractStridedMetadataOpLowering: pack
%11 = llvm.insertvalue %8, %10[1] : !llvm.struct<(ptr, ptr, i64>
%12 = llvm.mlir.constant(0 : index) : i64
%13 = llvm.insertvalue %12, %11[2] : !llvm.struct<(ptr, ptr, i64>
%14 = llvm extractvalue %5[2]
  : !llvm ExtractStridedMetadataOpLowering: unpack
%15 = llvm.extractvalue %5[3, 0]
}

```

```

  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>>
%16 = llvm.extractvalue %5[4, 0]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>>
%17 = llvm.mlir.poison
  : !llvm MemRefReinterpretCastOpLowering: unpack
%18 = llvm.extractvalue %13[0] : !llvm.struct<(ptr, ptr, i64>
%19 = llvm.extractvalue %13[1] : !llvm.struct<(ptr, ptr, i64>
%20 = llvm.insertvalue %18, %17[0]
  : !llvm struct<(ptr, ptn, i64, array<1 x i64>, array<1 x i64>>
  : !llvm MemRefReinterpretCastOpLowering: pack
%21 = llvm.insertvalue %19, %20[1]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>>
%22 = llvm.insertvalue %arg5, %21[2]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>>
%23 = llvm.mlir.constant(2 : index) : i64
%24 = llvm.insertvalue %23, %22[3, 0]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>>
%25 = llvm.mlir.constant(1 : index) : i64
%26 = llvm.insertvalue %25, %24[4, 0]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>>
%27 = llvm.extractvalue %26[1]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>>
%28 = llvm.extractvalue %26[2]
  : !llvm.struct<(ptr, ptr, i64, array<1 x i64>, array<1 x i64>>
%29 = llvm.getelementptr %27[%28] : (!llvm.ptr, i64) -> !llvm.ptr, f32
%30 = llvm.getelementptr %29[%6] : (!llvm.ptr, i64) -> !llvm.ptr, f32
%31 = llvm.load %30 : !llvm.ptr -> f32
  llvm.return %31 : f32
}

```

Test Case: 1:N Lowering (better lowering)

```
llvm.func @bar(%arg0: !llvm.ptr, %arg1: !llvm.ptr, %arg2: i64, %arg3: i64, %arg4: i64, %arg5: i64) -> f32 {  
    %0 = llvm.mlir.constant(1 : index) : i64  
    %1 = llvm.mlir.poison : !llvm.ptr  
    %2 = llvm.mlir.poison : i64  
    %3 = llvm.mlir.constant(0 : index) : i64  
    %4 = llvm.mlir.poison : !llvm.ptr  
    %5 = llvm.mlir.poison : i64  
    %6 = llvm.mlir.constant(2 : index) : i64  
    %7 = llvm.mlir.constant(1 : index) : i64  
    %8 = llvm.getelementptr %arg1[%arg5] : (!llvm.ptr, i64) -> !llvm.ptr, f32  
    %9 = llvm.getelementptr %8[%0] : (!llvm.ptr, i64) -> !llvm.ptr, f32  
    %10 = llvm.load %9 : !llvm.ptr -> f32  
    llvm.return %10 : f32  
}
```

1:N Conversion API

Type Converter

```
converter.addConversion([&](MemRefType type,
                           SmallVectorImpl<Type> &result) -> std::optional<LogicalResult> {
    result.push_back(llvmPtrTy);           // allocated ptr
    result.push_back(llvmPtrTy);           // aligned ptr
    result.push_back(i64Ty);               // offset
    for (int64_t i = 0; i < rank; ++i)
        result.push_back(i64Ty);           // sizes
    for (int64_t i = 0; i < rank; ++i)
        result.push_back(i64Ty);           // strides
    return success();
});

// previously: returned !LLvm.struct<...>
```

These are the types of the ValueRange that the adaptor returns.



Conversion Pattern

```
// Simplified: Assume that source is a ranked memref and index is static.

class DimOpLowering : public OpConversionPattern<memref::DimOp> {
    LogicalResult matchAndRewrite(memref::DimOp op, OneToNOpAdaptor adaptor,
                                  ConversionPatternRewriter &rewriter) const override {
        int64_d dim = op.getIndex();
        int64_t descriptorPos = 2 + 1 + dim;
        ValueRange descriptor = adaptor.getSource();
        rewriter.replaceOp(op, descriptor[descriptorPos]);
        return success();
    }
};

// previously: OpAdaptor
```

Conversion Pattern (incorrect example)

```
// Simplified: Assume that source is a ranked memref and index is static.

class DimOpLowering : public OpConversionPattern<memref::DimOp> {
    LogicalResult matchAndRewrite(memref::DimOp op, OpAdaptor adaptor,
                                  ConversionPatternRewriter &rewriter) const override {
        int64_d dim = op.getIndex();
        int64_t descriptorPos = 2 + 1 + dim;
        Value descriptor = adaptor.getSource();
        rewriter.replaceOp(op, ???);
        return success();
    }
};

// previously: OpAdaptor
```

What if I use a regular adaptor in a 1:N conversion?

LLVM fatal error: ‘DimOpLowering’ does not support 1:N conversion

Note: You can use a regular adaptor if all converted values are guaranteed to be single SSA values.

Migration Path from Deprecated 1:N Dialect Conversion

OneToNTypeConversion.h

Migration Guide

- Will be deleted from upstream MLIR after this conference!
- Migrate patterns, driver invocation and tests
 - `OneToNConversionPattern` → `ConversionPattern`
 - `applyPartialOneToNConversion` → `applyPartialConversion`.
You're going to have to define a `ConversionTarget`.
 - `replaceOp(Operation *, ValueRange, OneToNTypeMapping)`
→ `replaceOpWithMultiple(Operation*, ArrayRef<ValueRange>)`
 - `OneToNTypeMapping` → `SignatureConversion`
 - Update test cases: Old framework was actually a greedy pattern rewrite
that performs additional CSE'ing, folding, region simplification.

Questions?

applyPartialConversion

applyPartialOneToNConversion

applySignatureConversion

ConversionPattern

ConversionPatternRewriter

ConversionTarget

LLVMStructType

MemRefType

OpAdaptor

OneToNOpAdaptor

OneToNTypeMapping

replaceOp

replaceOpWithMultiple

SignatureConversion

source materialization

TypeConverter

target materialization

UnrankedMemRefType

unrealized_conversion_cast



Developers' Meeting

BERLIN 2025



Planning Tile & Fuse Transform in MLIR

April 8, 2025

Aviad Cohen, Algorithm engineer

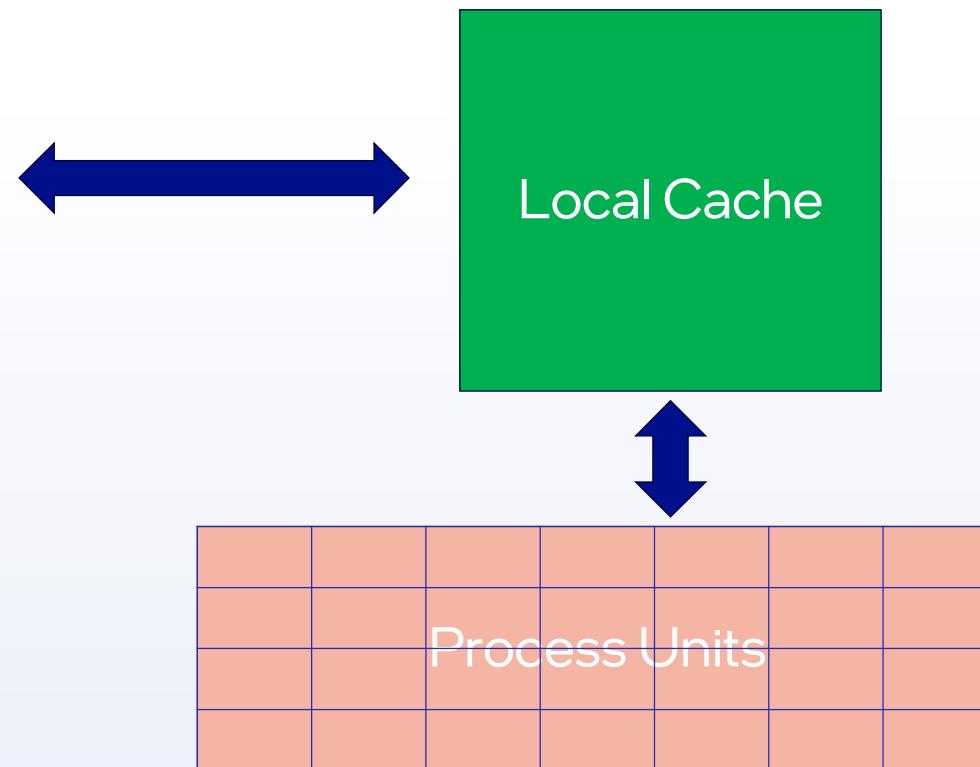
1 Picture, 1000 words



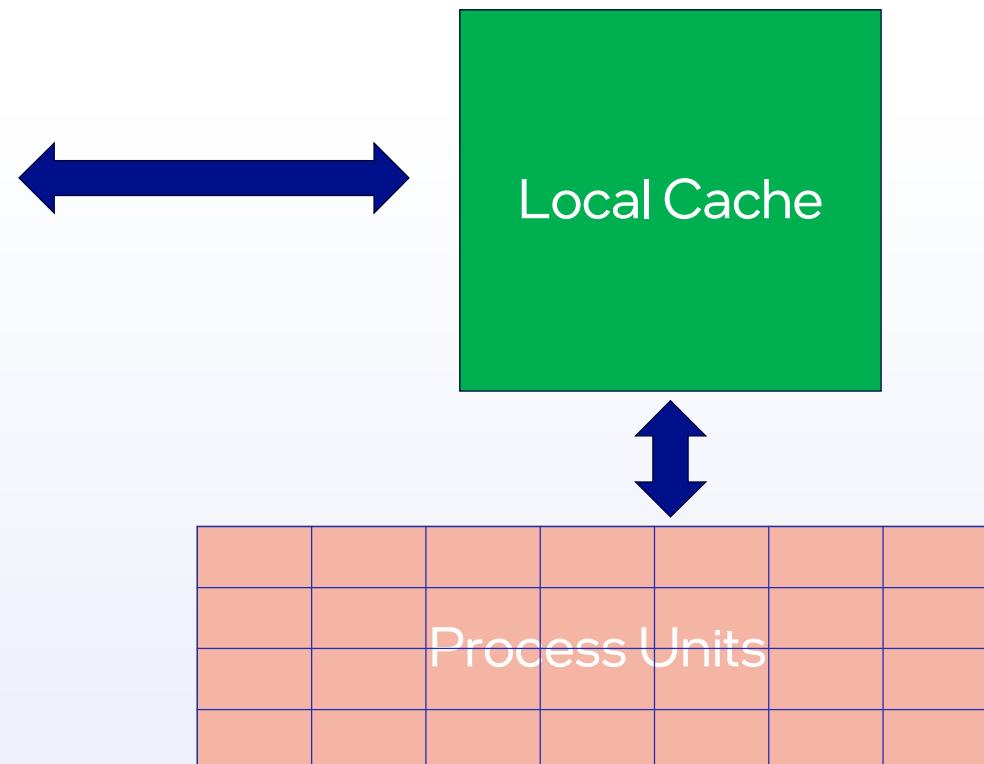
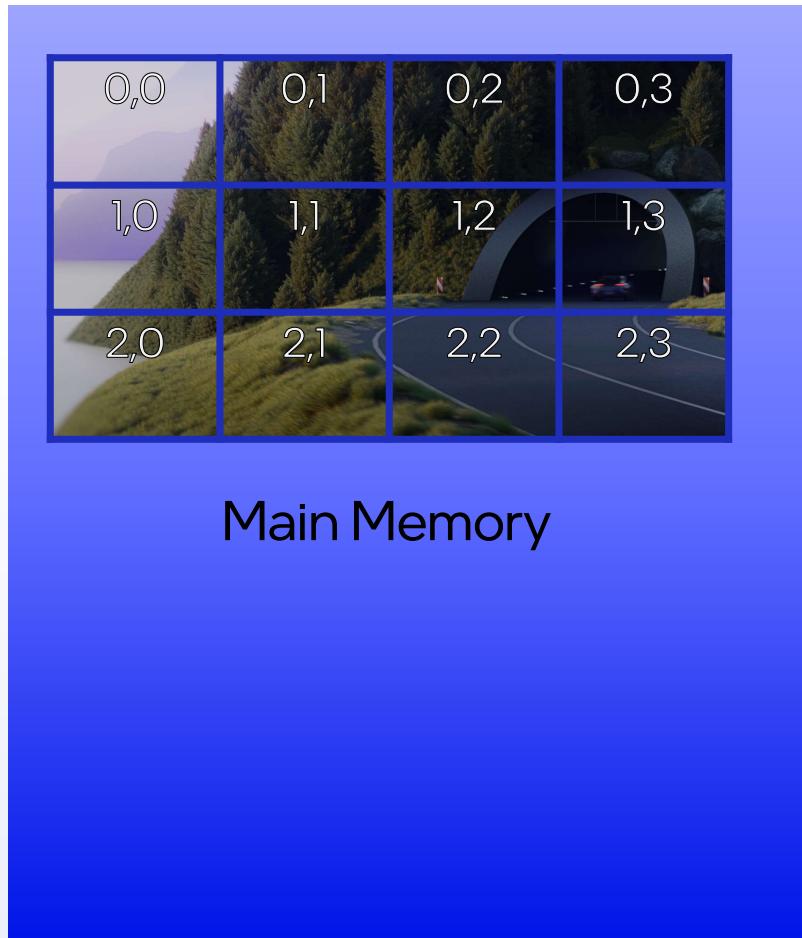
What is tiling all about?



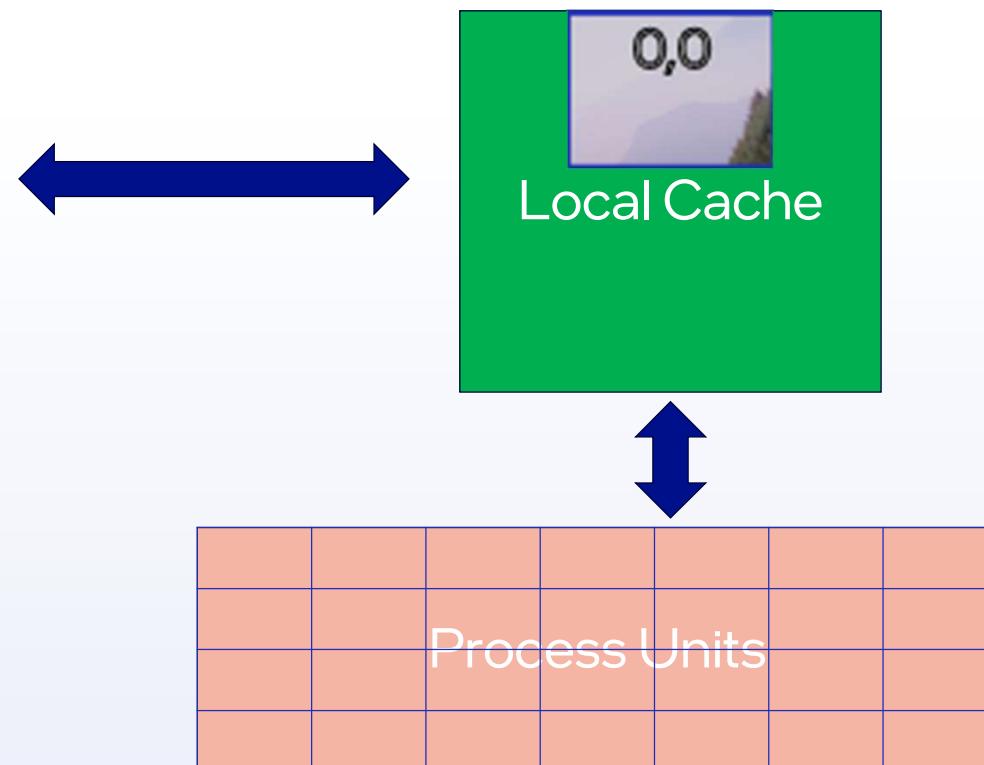
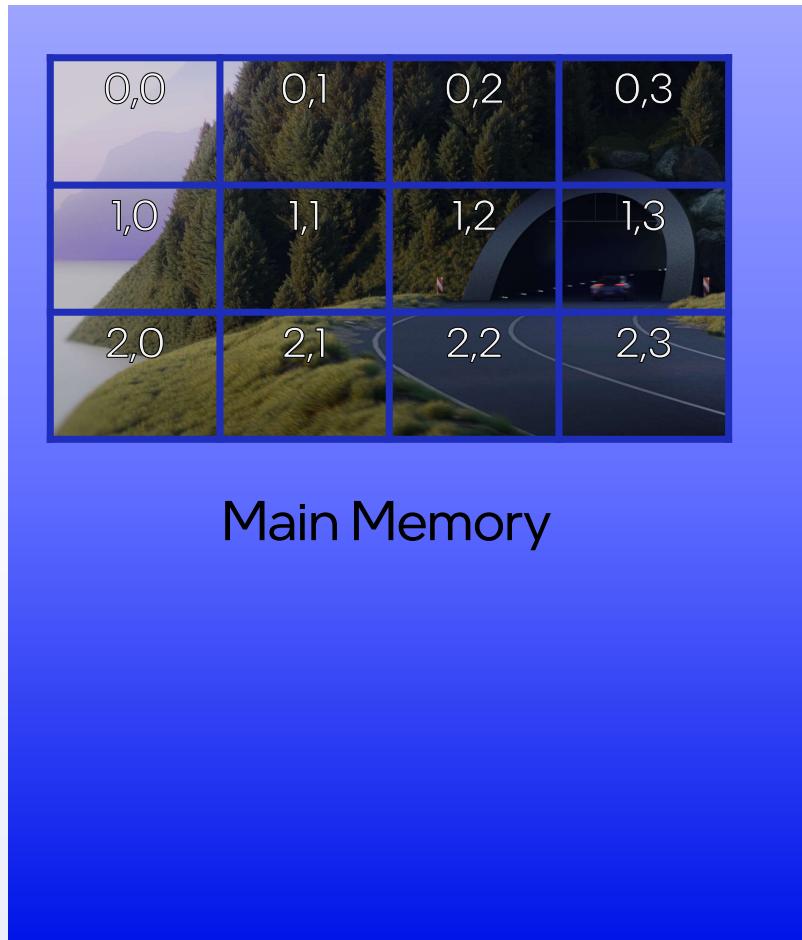
Main Memory



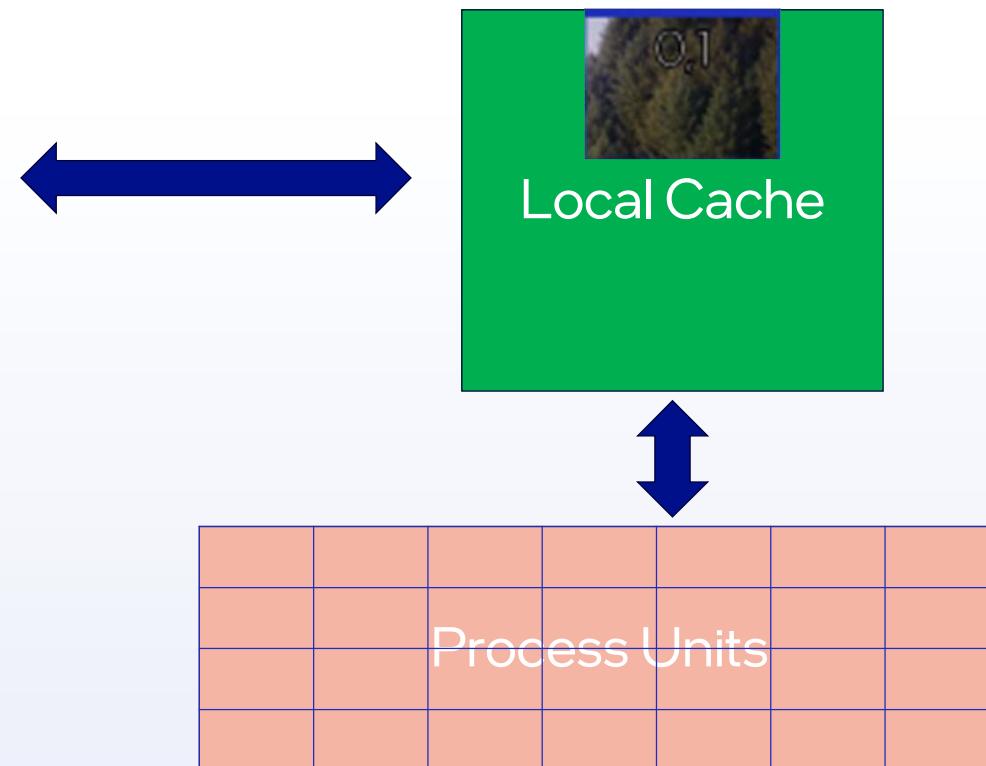
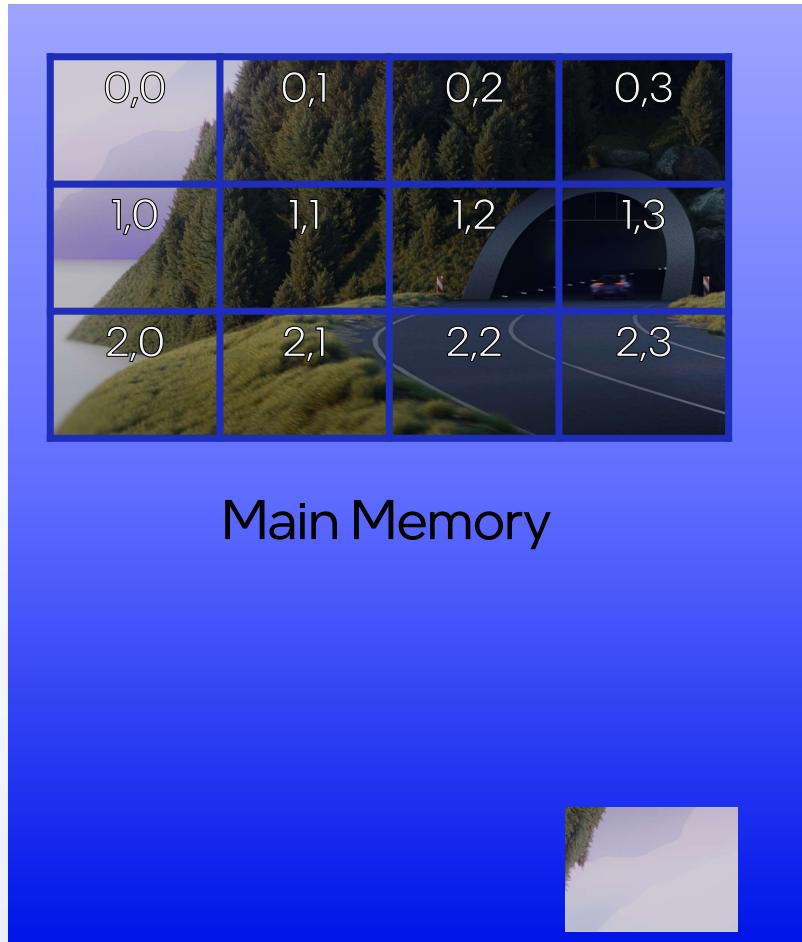
What is tiling all about?



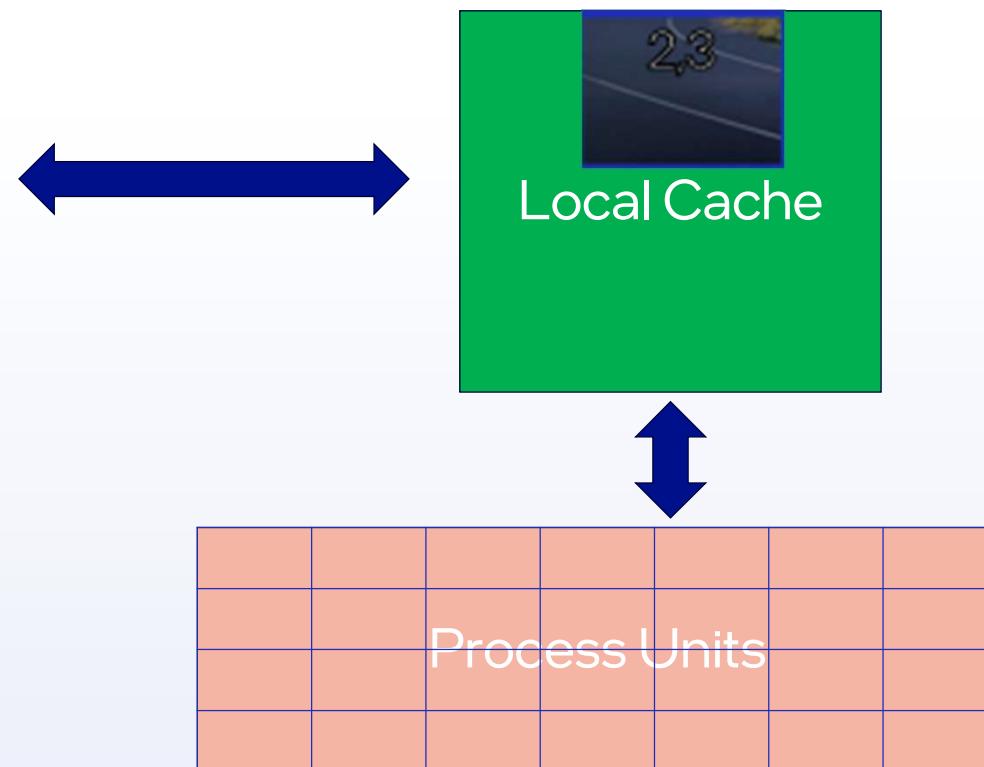
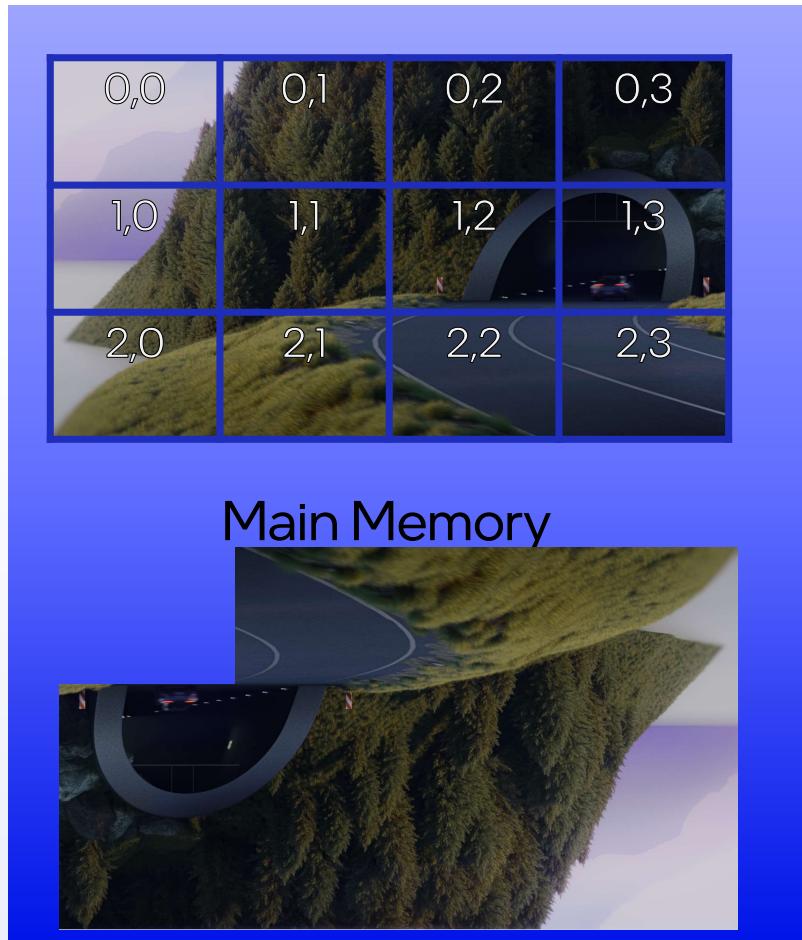
What is tiling all about?



What is tiling all about?



What is tiling all about?

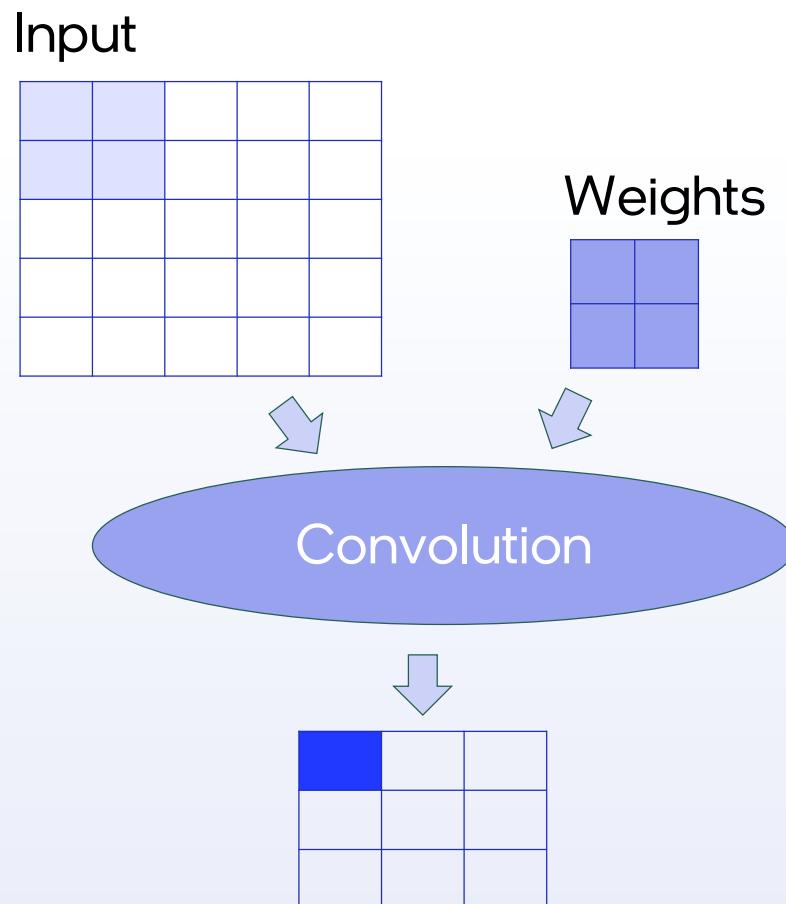


Goal – Reach High Performance

- How?
 - Optimize cache usage by keeping data localized.

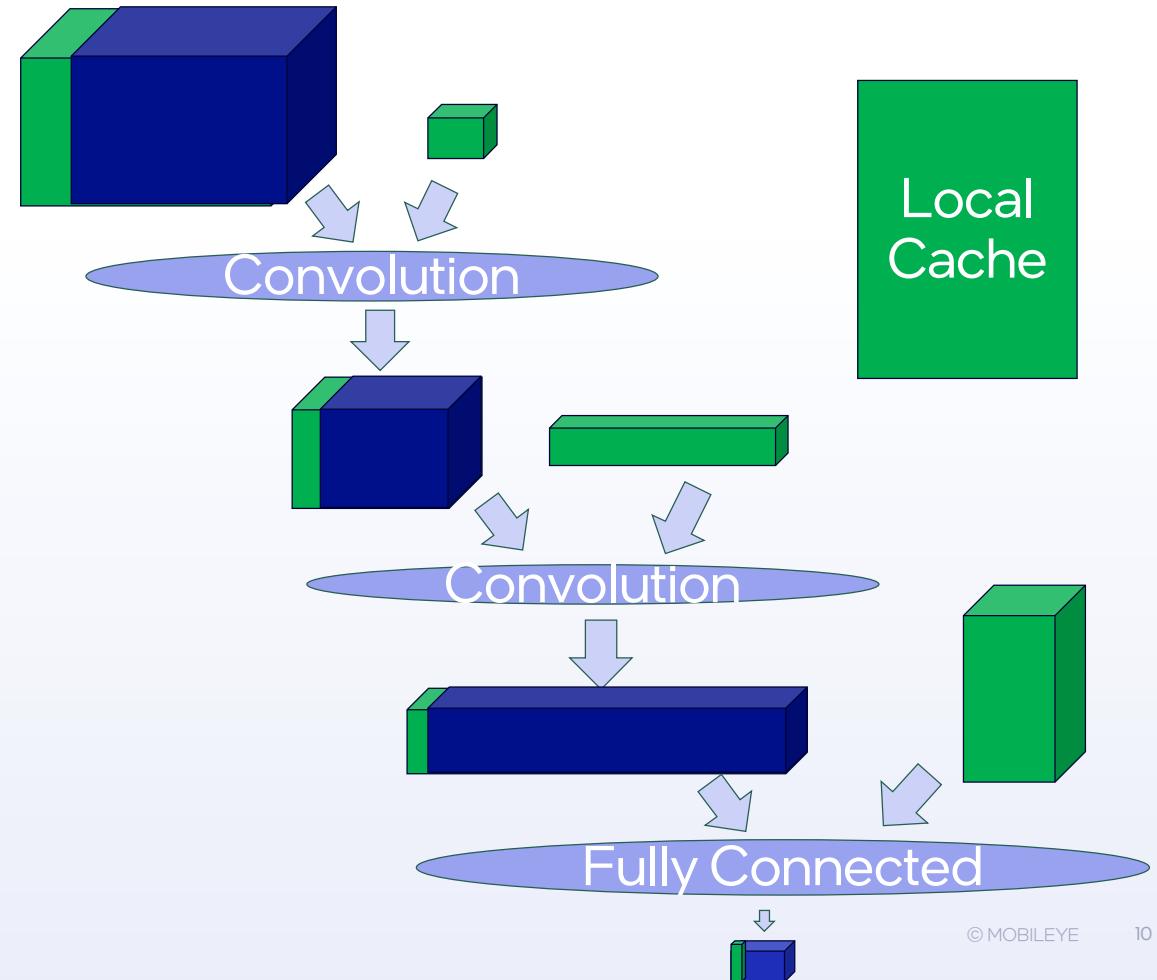
Goal – Reach High Performance

- How?
 - Optimize cache usage by keeping data localized.
 - Retain essential data within the cache for all tiles.



Goal – Reach High Performance

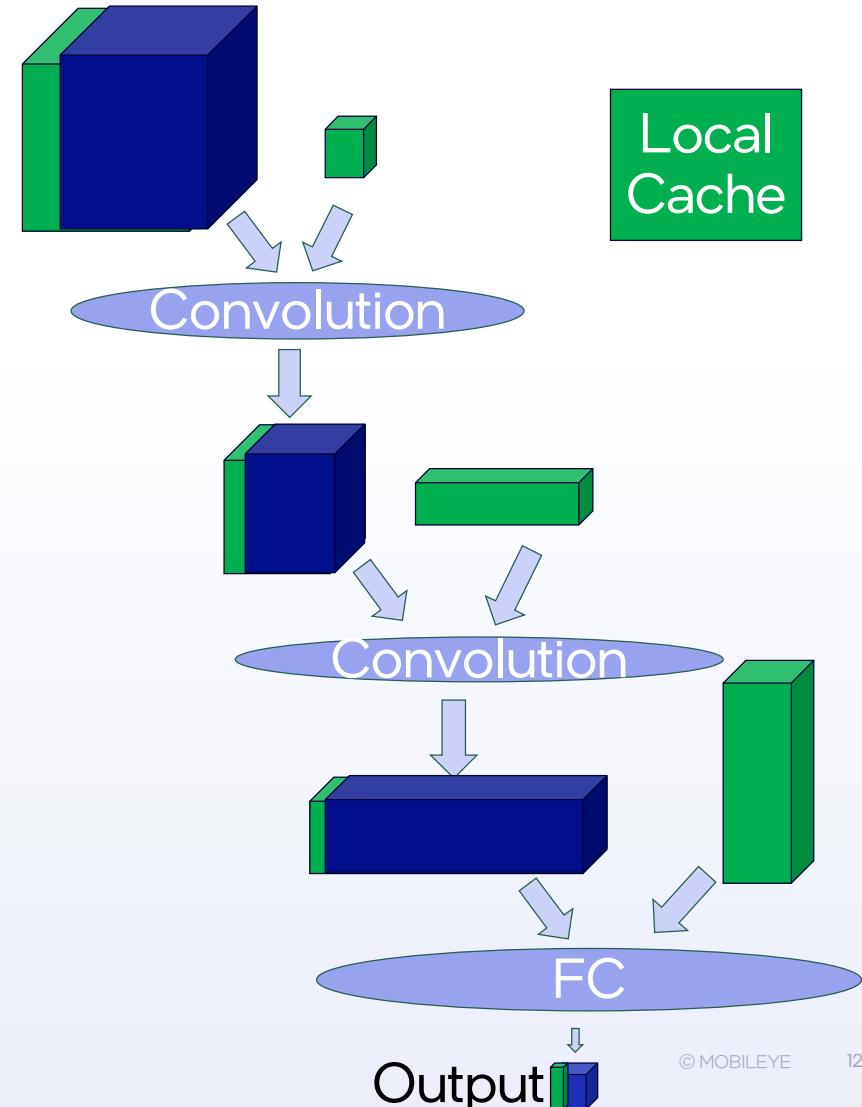
- How?
 - Optimize cache usage by keeping data localized.
 - Retain essential data within the cache for all tiles.
 - Utilize larger tiles to minimize control flow overhead.

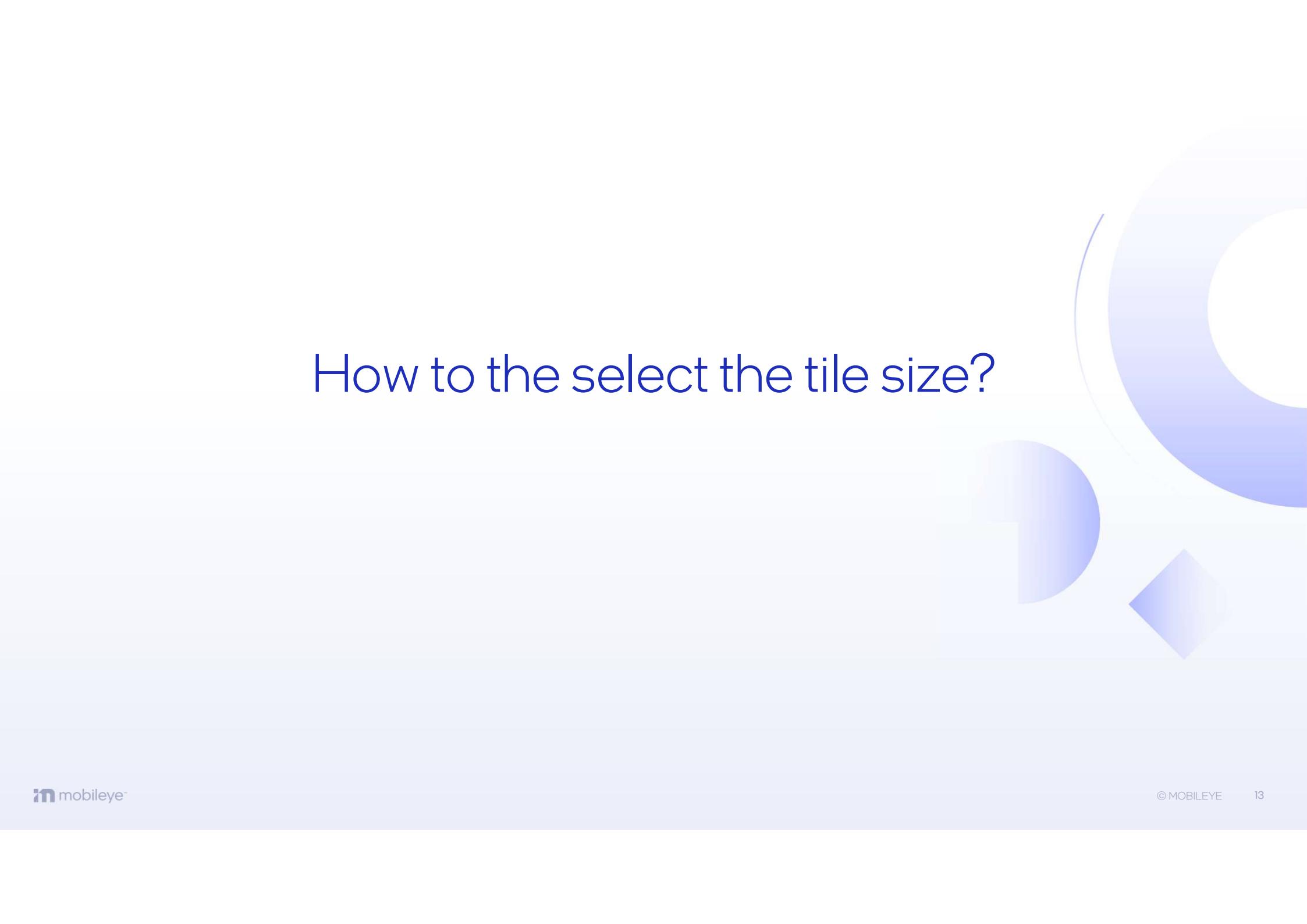


Fortunately, transform already exists!

Tile & Fuse - The transform

- Demands Tiling Interface.
- General flow:
 - Tile a root operation into loops.
 - Fuse consumer/producer operations into the existing loops.
- Can be controlled by a control function.

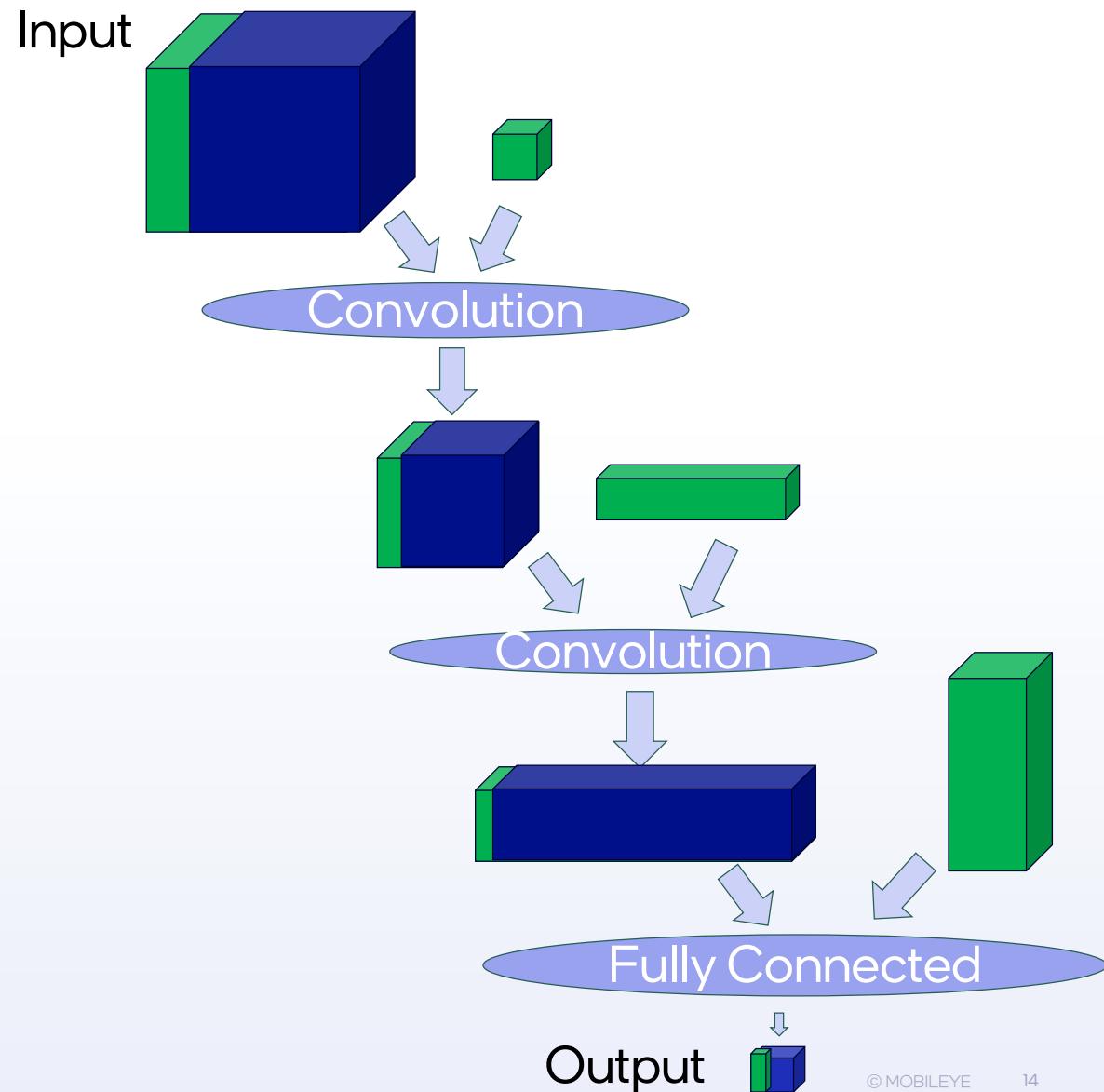




How to select the tile size?

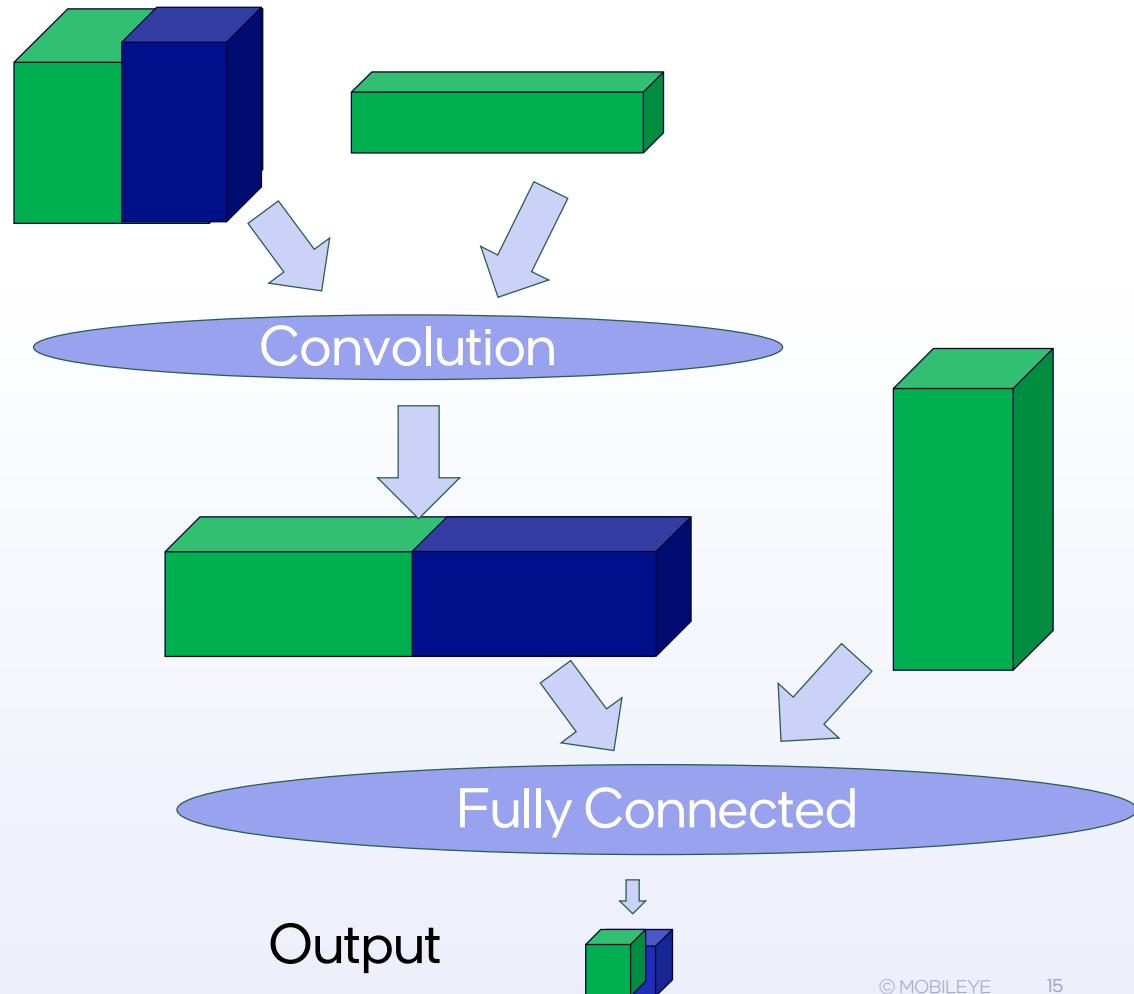
Vertical approach

Increase **number of fused operations over tile size**

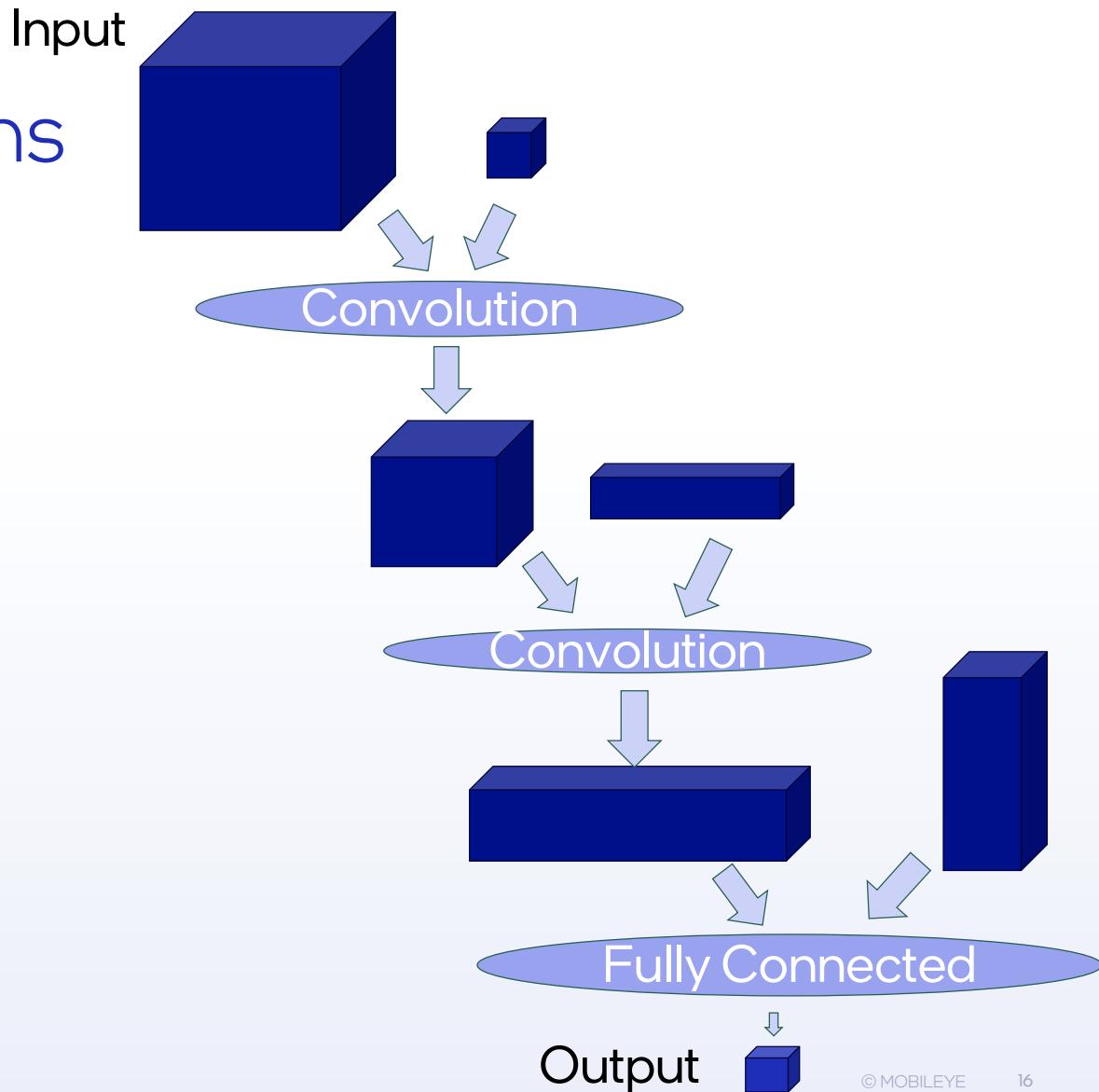


Horizontal Tiling

Increase **tile size**
over **number of**
fused operations



Planning Considerations



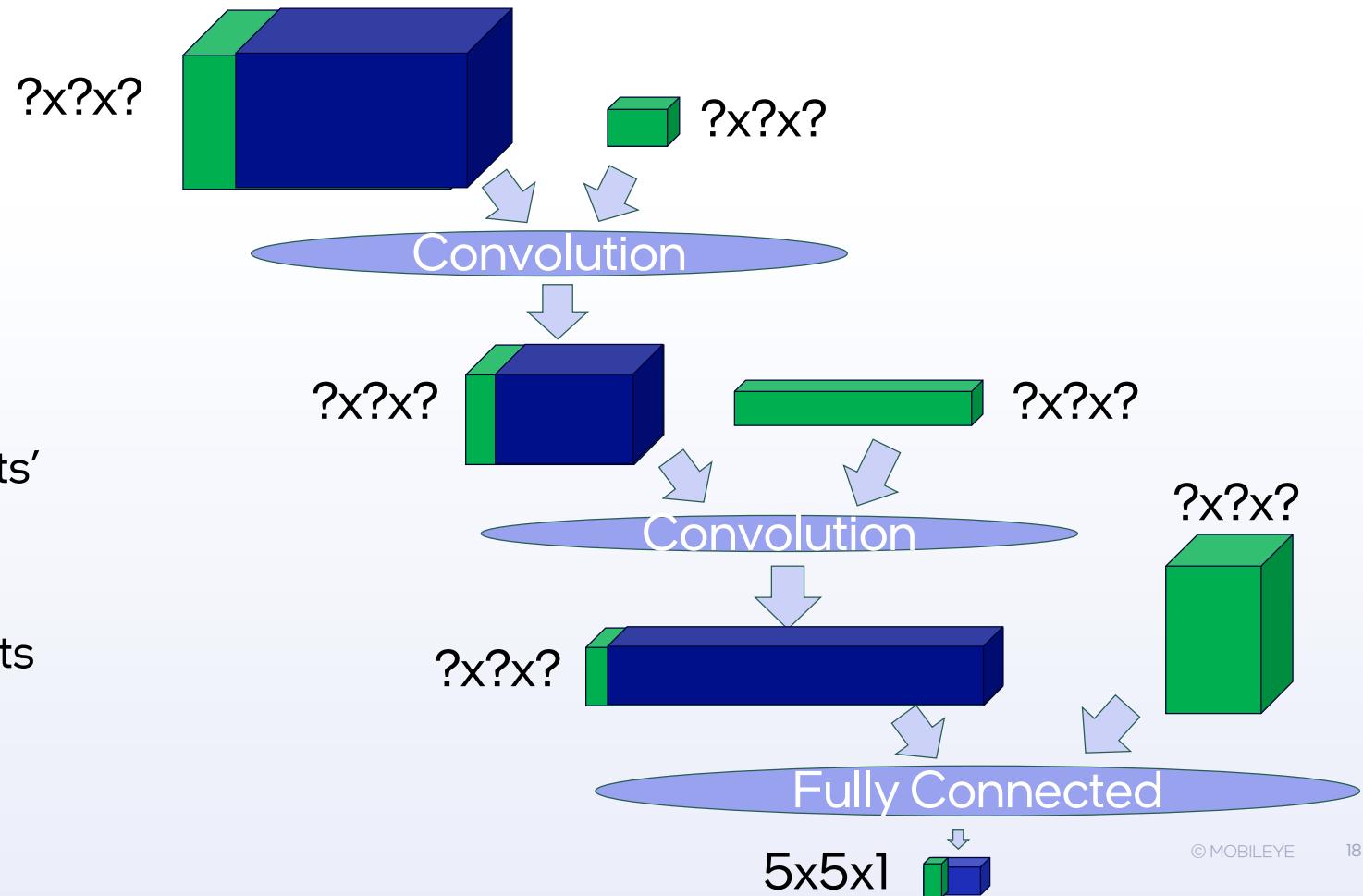
- Find the sweet spot between tile size and fusion to minimize bandwidth
- Overlapping tiles
- Scheduling approaches
- Too big control flow



So, what is missing?

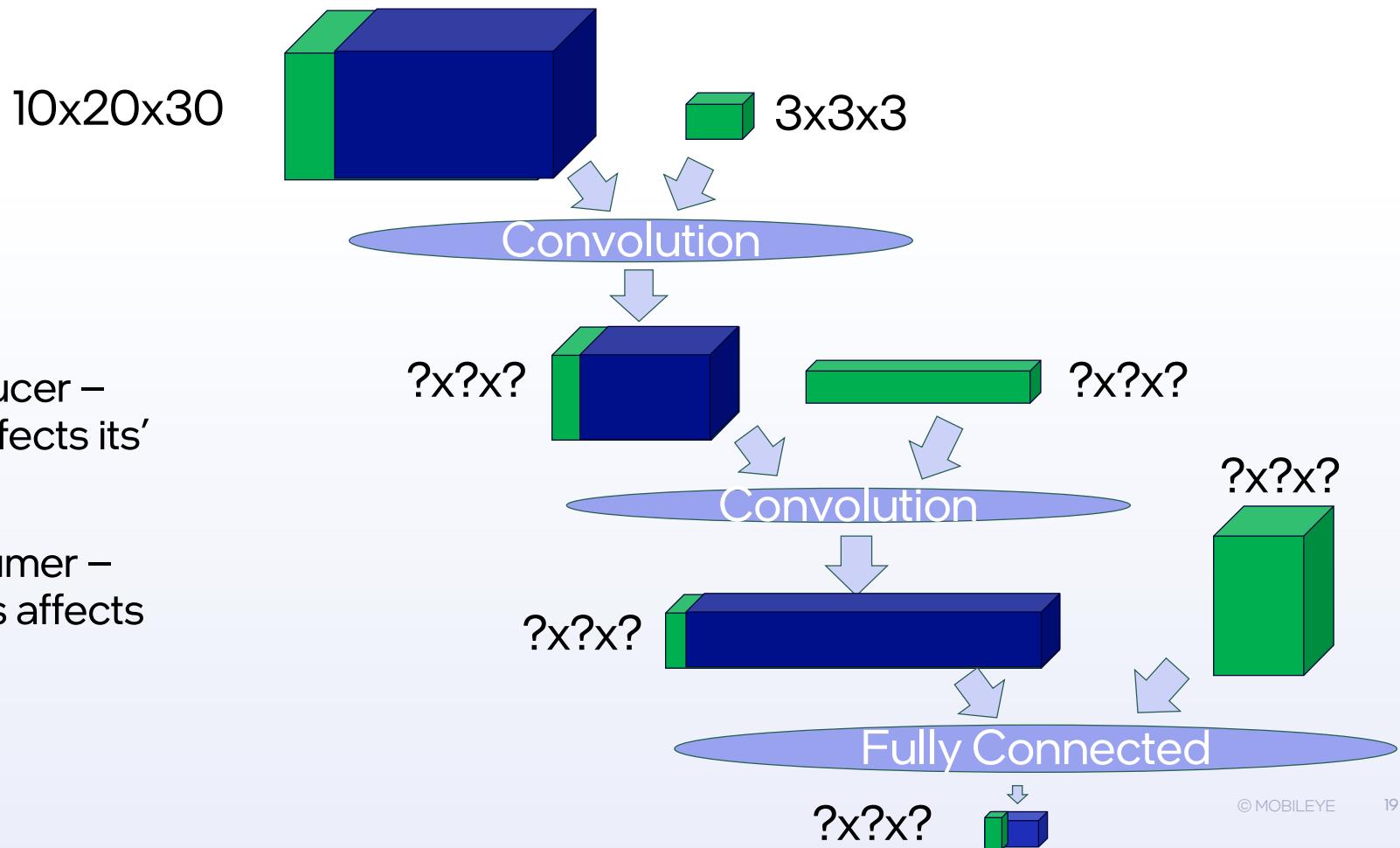
Proposal – Fusion Interface

- Given tiled producer – how its' results affects its' consumers?
- Given tiled consumer – how its' operands affects its' producers?



Proposal – Fusion Interface

- Given tiled producer – how its' results affects its' consumers?
- Given tiled consumer – how its' operands affects its' producers?



RFC - Fusion Analysis Interface for Compute Operations

- Additional method can be added, let's enhance the interface together!
- For more details follow the RFC:
<https://discourse.llvm.org/t/fusion-analysis-interface-for-compute-operations/85743>



Thank you!

Aviad Cohen

Aviad.cohen2@mobileye.com



Developers' Meeting

BERLIN 2025



BOLT'ED CLANG: HOW GOOD IS IT ON AARCH64?

ELVINA YAKUBOVA, SJOERD MEIJER

EUROLLVM 2025

HOW GOOD IS BOLT ON AARCH64?

Questions we wanted to investigate:

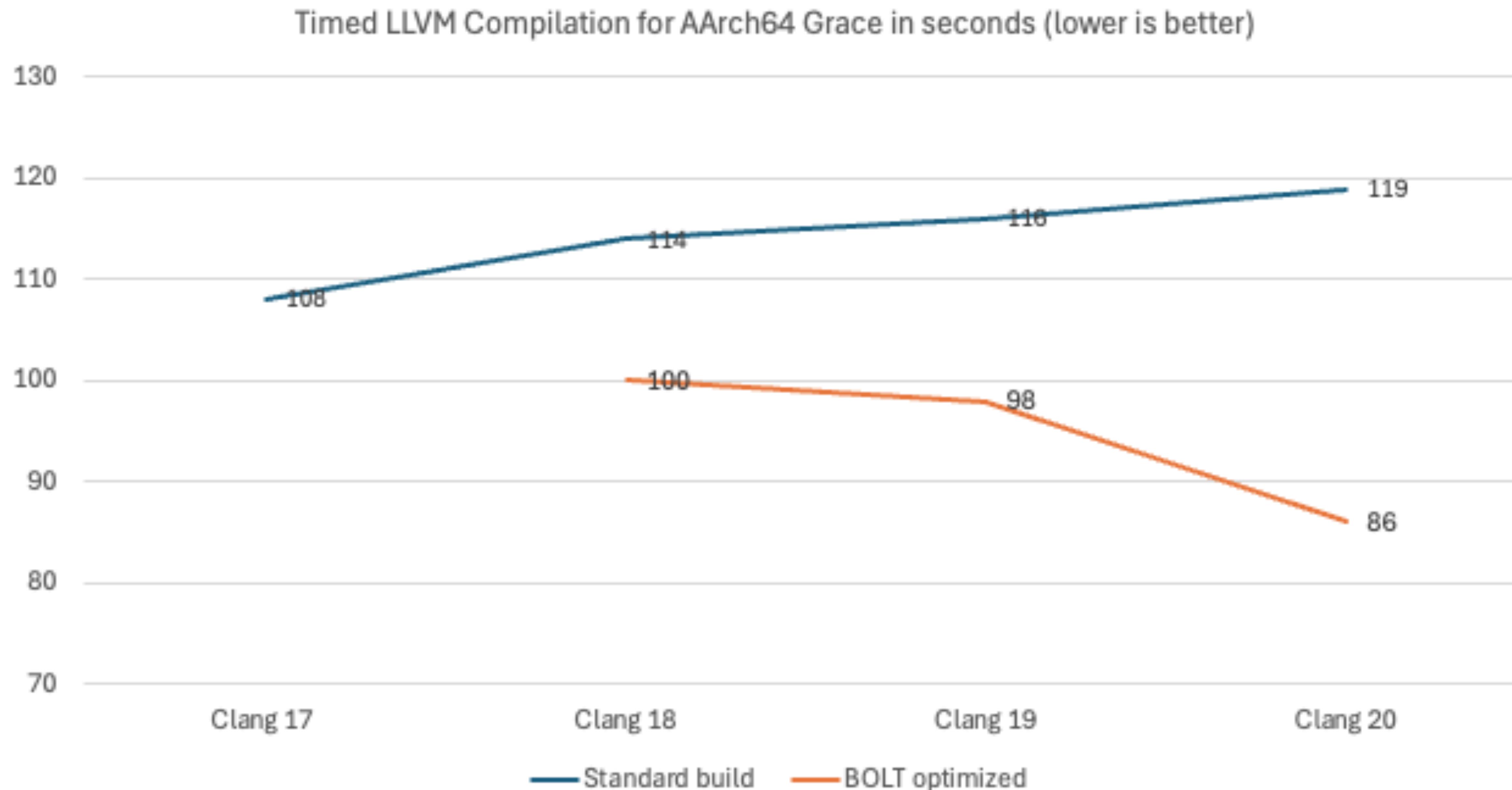
1. What performance improvements can we obtain by using BOLT on AArch64 platforms?
2. Should we BOLT our Clang build?
3. And can we do better and change build process?

We will show:

- Performance results for timed compilation of LLVM and CTMark (LLVM test-suite)
- The trends for Clang-18, Clang-19, and Clang-20
- Show experiments with more and different profiles

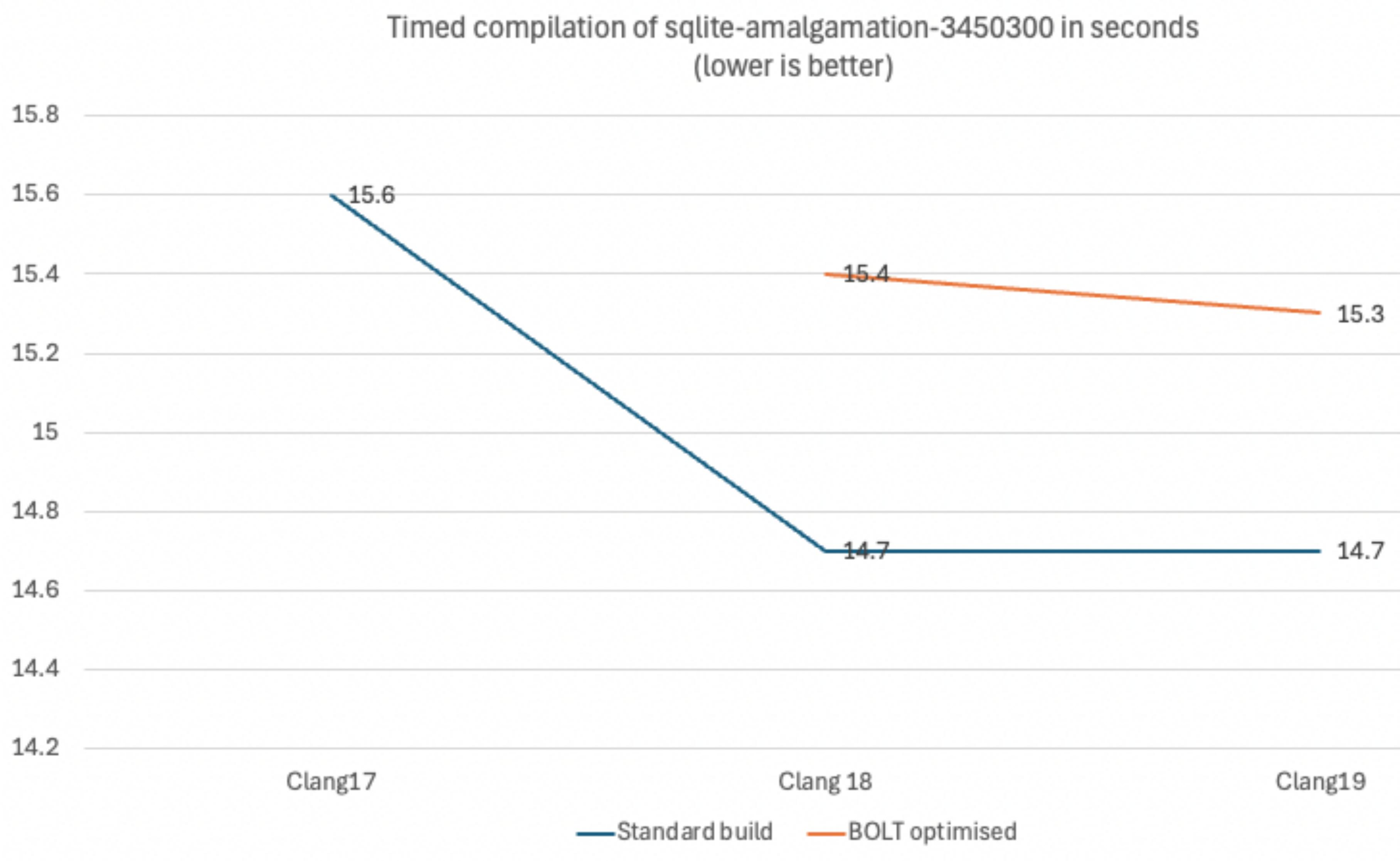
BOLT'ED CLANG: LTO/PGO/BOLT OPTIMISED COMPILER

- BOLT optimised AArc64 Clang toolchain
 - A.k.a.: how do we create a fast compiler?
 - Metric: measure compilation time of the BOLT'ed compiler: 27% speed-up of BOLT'ed Clang-20 compared to a standard build:



IS IT GOOD FOR ALL WORKLOADS?

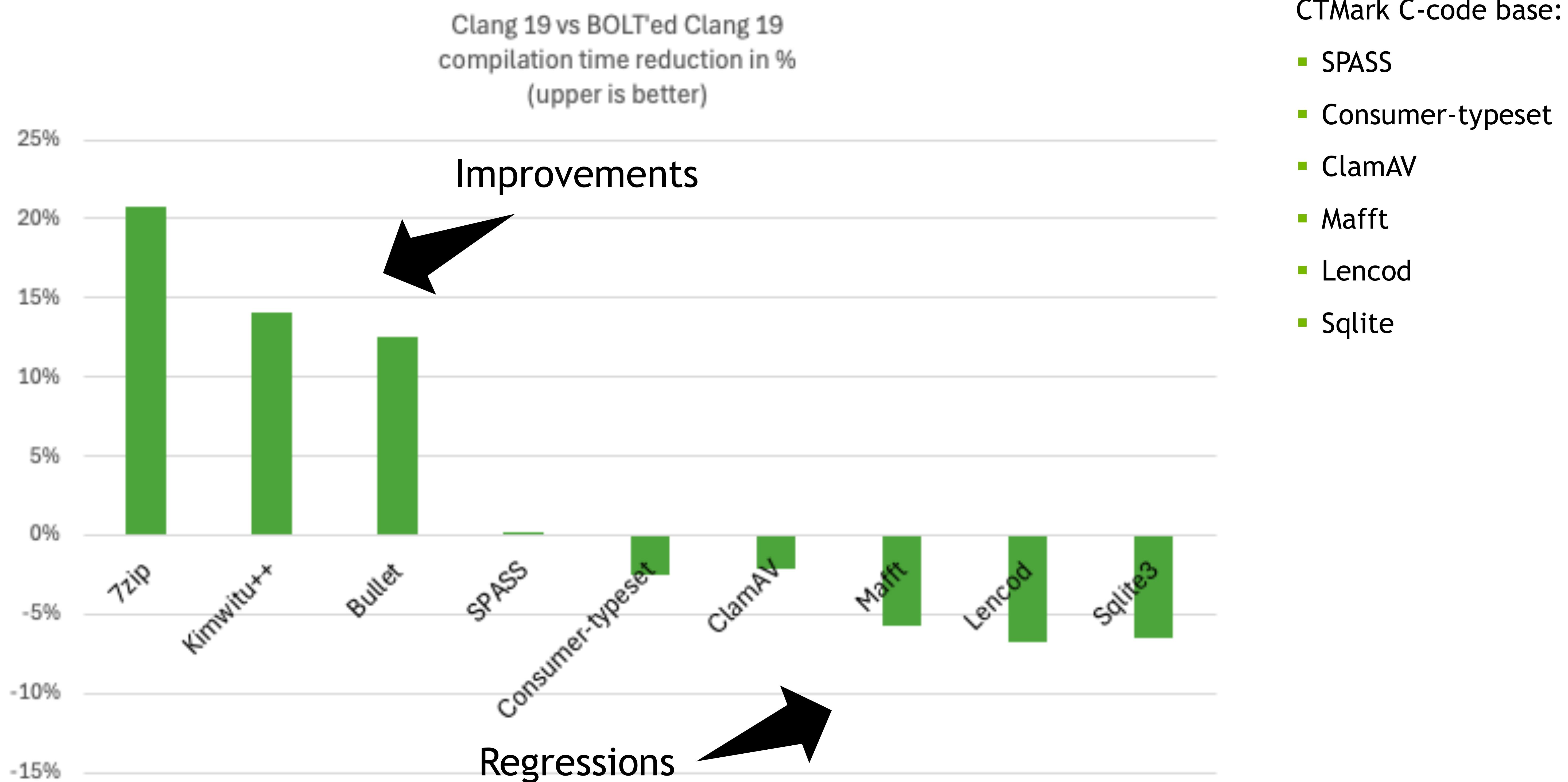
- 4% compile-time regression for SQLite with BOLT'ed Clang:



Hypothesis:

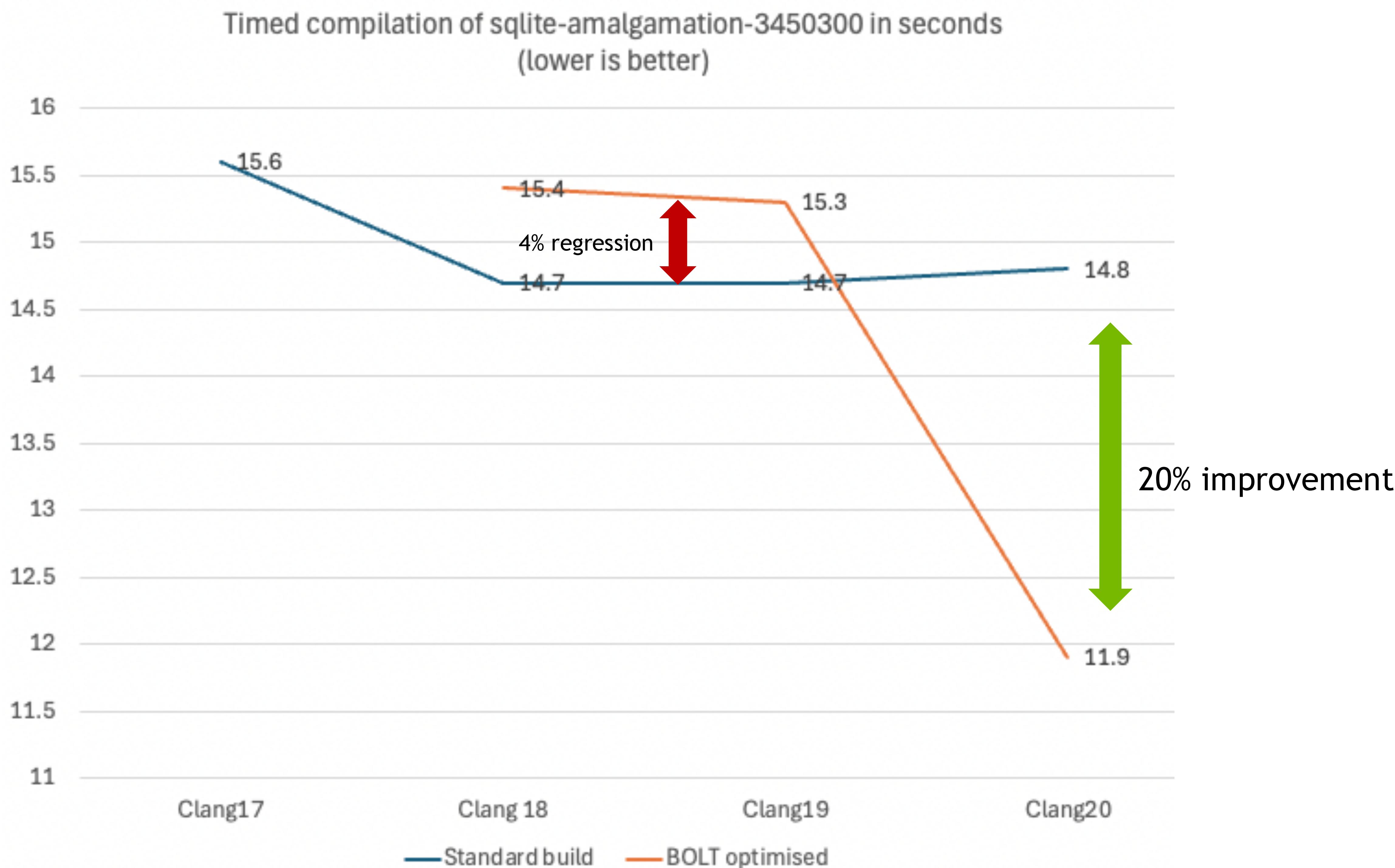
- BOLT'ed Clang is trained on a modern C++ code (LLVM),
- Maybe this works less well for SQLite that is C-code.

IT'S NOT ALL GOOD, MORE REGRESSIONS...

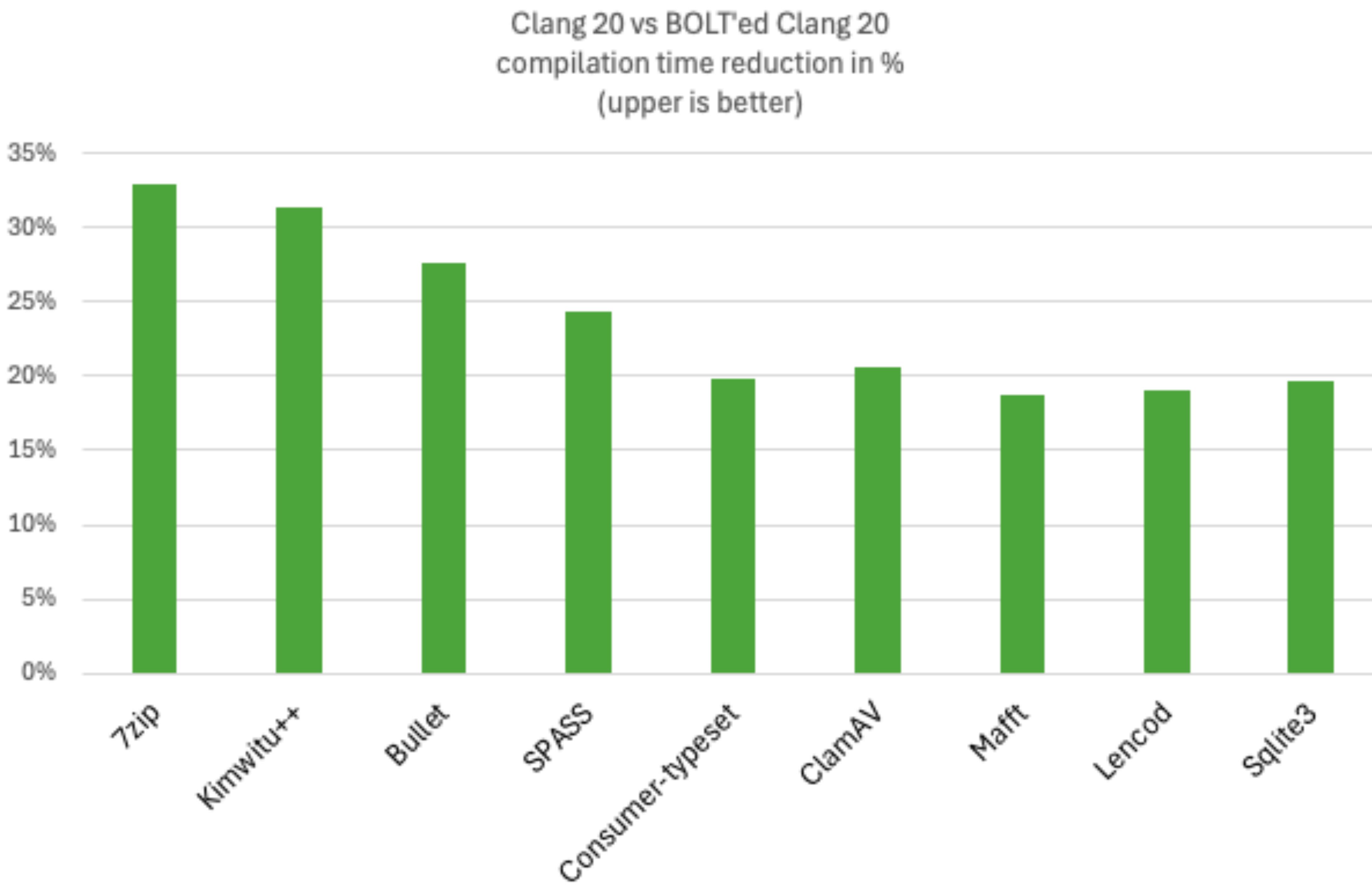


CLANG 20: GAME CHANGER

- Massive 20% SQLite compile-time improvement!



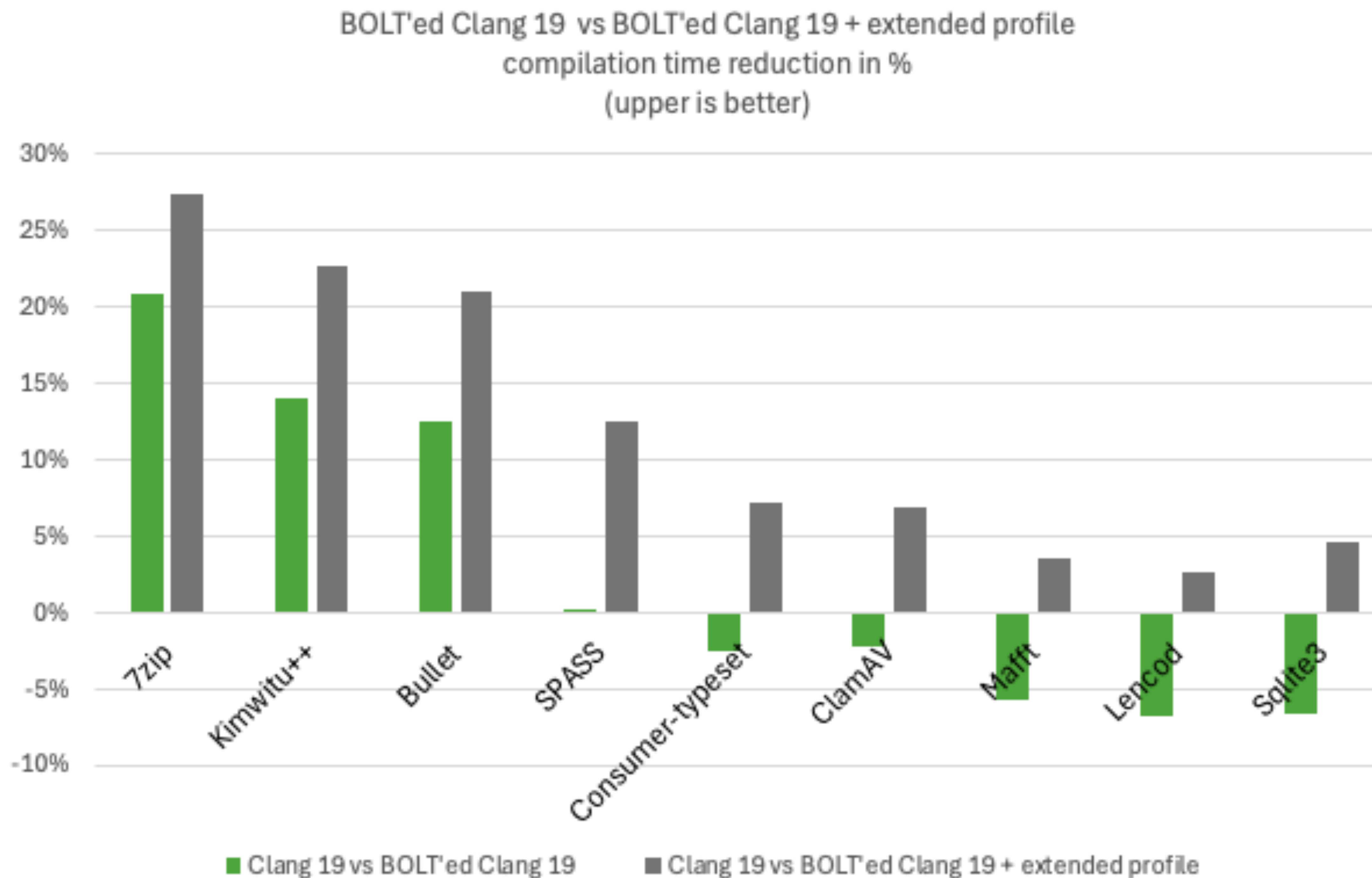
ONLY IMPROVEMENTS WITH CLANG 20



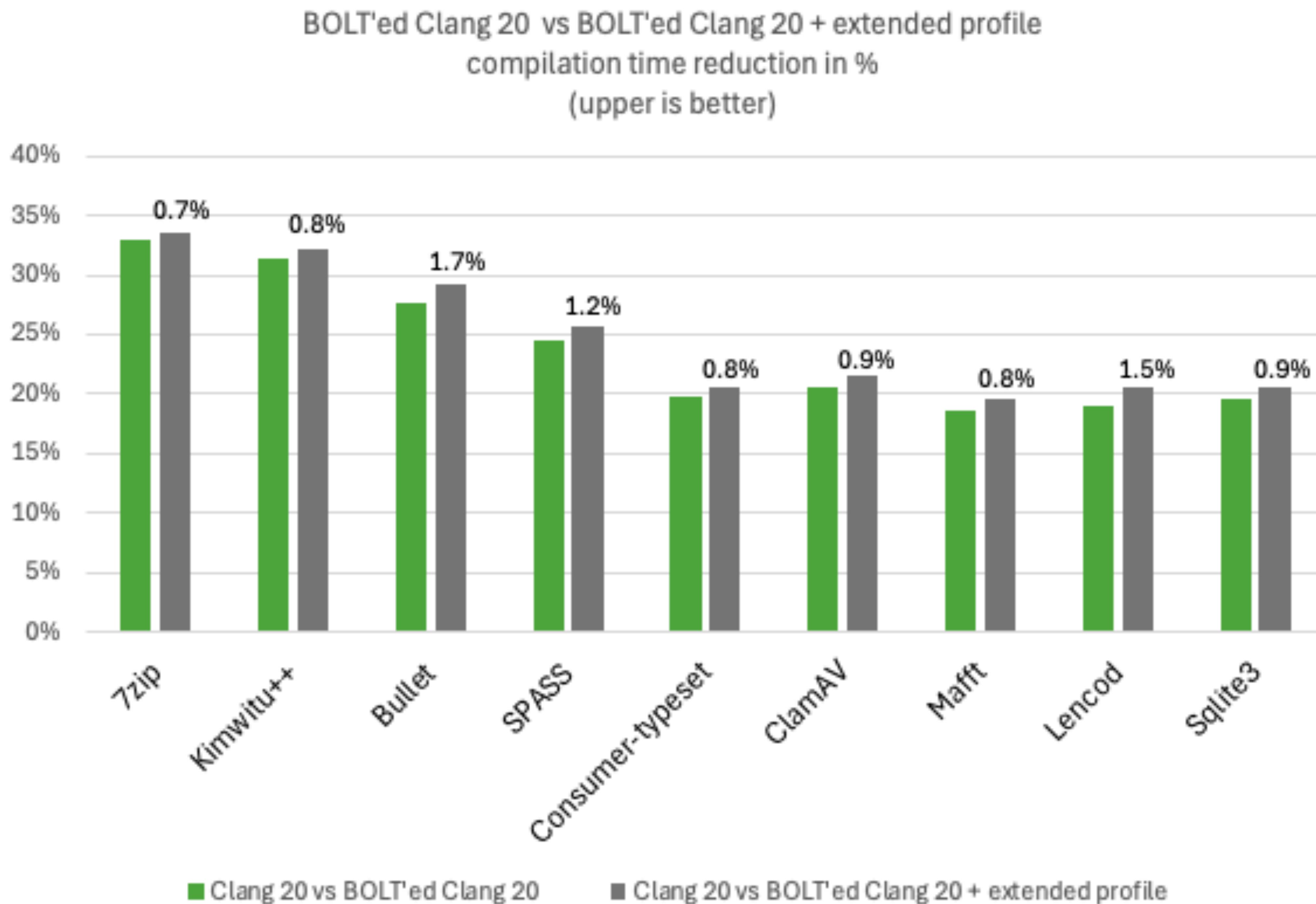
CAN WE DO BETTER?

- BOLT'ed Clang-20 improvements could come from:
 - CMake configurations: different options passed on,
 - BOLT learned new optimisations,
 - More/better profiles?
- Can we do better?
 - BOLT'ed Clang is trained on a modern C++ code-base (LLVM),
 - Should we extend the training stage with more/different profiles?
- Extended Profile:
 - Collect profiles for CTMark from the LLVM test-suite (C code-base),
 - Collect profiles for LLVM (C++ code-base),
 - Merge LLVM + CTMark profiles and use that as input to BOLT.

MORE PROFILES FIX THE CLANG 19 RESULTS

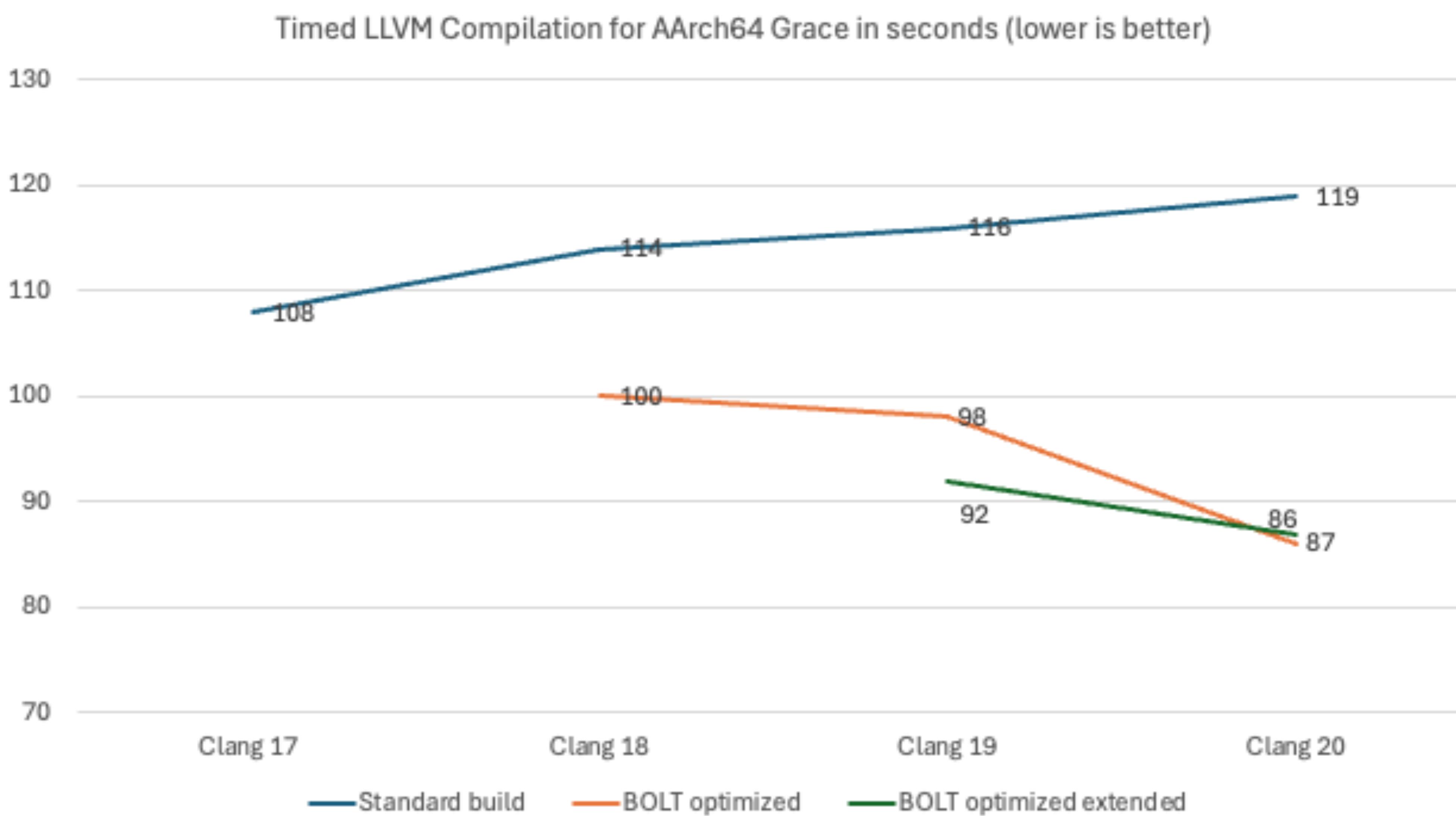


EXTENDED PROFILES: ADDITIONAL IMPROVEMENTS LESS THAN 2%



LTO/PGO/BOLT OPTIMISED CLANG COMPILER

- Additional 6% improvement on Clang 19
- No difference on Clang 20



CONCLUSIONS

- Clang-20 and BOLT-20 are great, also on AArch64!
 - BOLT'ed Clang-20 is 27% faster than Clang-20
 - Universally good: the same or better performance (for LLVM and CTMark)
 - Fixes the issues with Clang-19 and older versions that were not so great yet.
- Yes, Clang releases should be BOLT'ed
 - They started with the latest release
- Extended profiles:
 - Clang-20 is now also trained on libLLVMSupport: big improvement
 - Training it even more with CTMark: almost makes no difference!
 - The current CMake Clang/BOLT build process and configuration is enough
- Future work:
 - CTMark apps are small, investigate more/bigger apps,
 - With extended profiles, they should also be added to the PGO stage (i.e. not only BOLT)



Thank you





Developers' Meeting

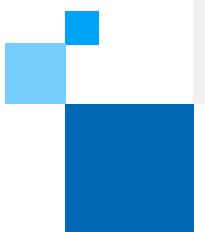
BERLIN 2025

EuroLLVM 2025 - Berlin

MLIR Tensor Compiler

design group & charter update

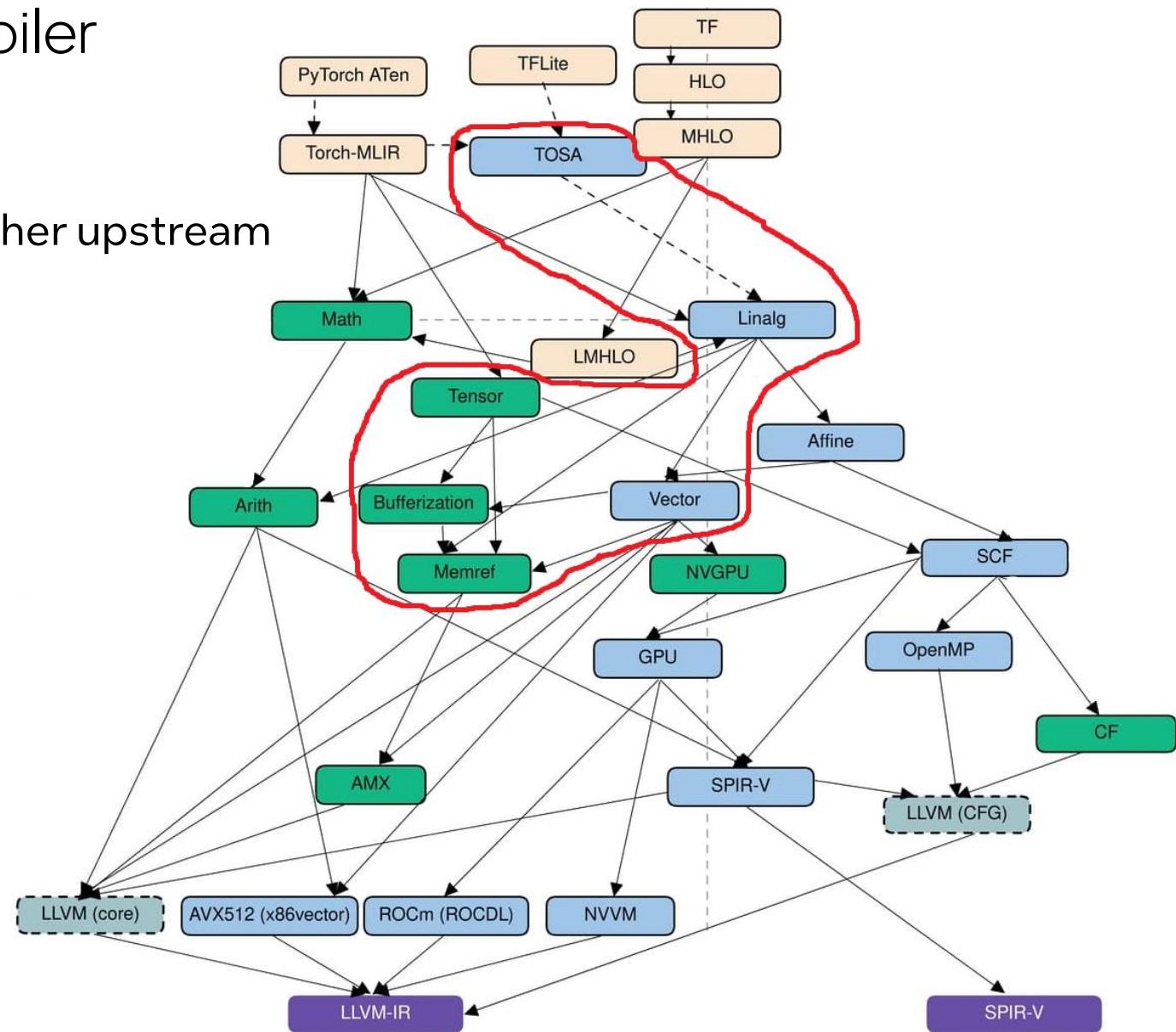
Rolf Morel & Renato Golin



intel

Upstream MLIR Tensor Compiler

- MLIR-based toolkit for ML compilers
 - Great value in developing this together upstream
- Main dialects:
 - **linalg**
 - **tensor**
 - **memref**
 - **vector**
 - **bufferization**
 - **tosa**



MLIR organization – dialect groupings

- Originates with MLIR Governance efforts from last year

 [\[RFC\] MLIR Project Charter and Restructuring](#) by Renato Golin, Stella Laurenzo, Chris Lattner, Alex Zinenko, Jacques Pienaar, Mehdi Amini

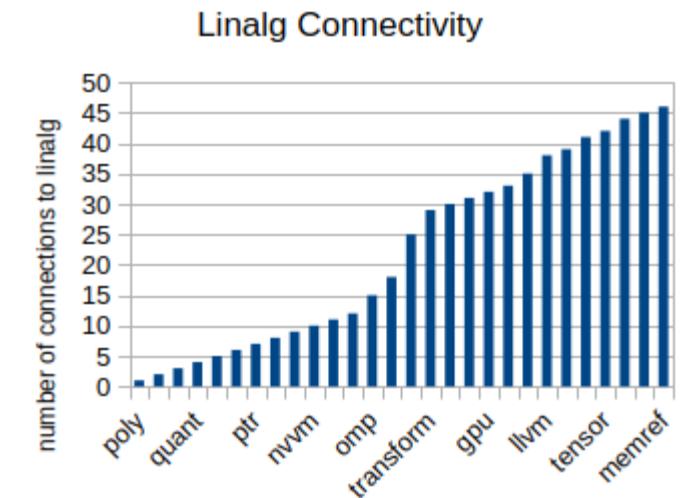
 [\[Survey\] MLIR Project Charter and Restructuring Survey](#) by Renato

 [MLIR Organization & Charter](#) by Renato, Stella, Alex, Jacques, Chris, and Andrzej Warzynski, Nicolas Vasilache, Mahesh Ravishankar

MLIR organization – dialect groupings

- Originates with MLIR Governance efforts from last year
 - [\[RFC\] MLIR Project Charter and Restructuring](#) by Renato Golin, Stella Laurenzo, Chris Lattner, Alex Zinenko, Jacques Pienaar, Mehdi Amini
 - [\[Survey\] MLIR Project Charter and Restructuring Survey](#) by Renato
 - [MLIR Organization & Charter](#) by Renato, Stella, Alex, Jacques, Chris, and Andrzej Warzynski , Nicolas Vasilache, Mahesh Ravishankar

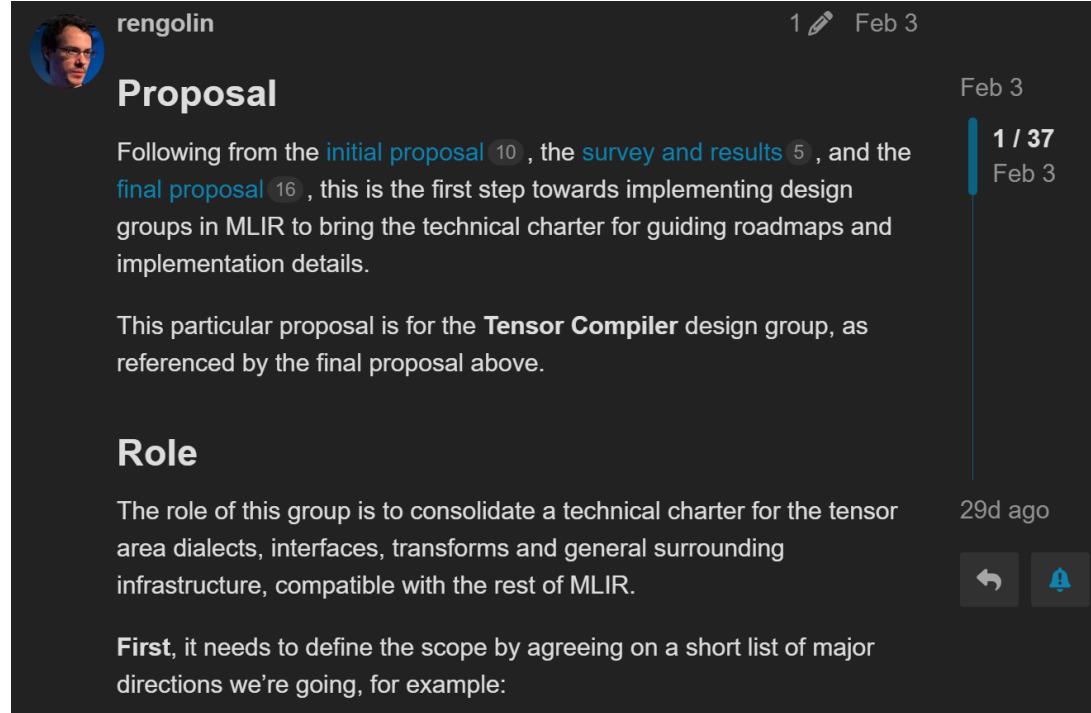
- Tensor Compiler dialect grouping
 - **linalg, tensor, memref, vector, bufferization, tosa**
 - Own community of stakeholders
 - In need of an overarching charter
 - Clear governance: upstream consensus



Tensor Compiler Design Group

 [MLIR Tensor Compiler Design Group](#) by Renato

Goal: charter across relevant dialects, interfaces, transforms, surrounding infrastructure



renolin · [Proposal](#) · Feb 3 · 1 / 37 · Feb 3

Following from the [initial proposal](#) 10, the [survey and results](#) 5, and the [final proposal](#) 16, this is the first step towards implementing design groups in MLIR to bring the technical charter for guiding roadmaps and implementation details.

This particular proposal is for the **Tensor Compiler** design group, as referenced by the final proposal above.

Role

The role of this group is to consolidate a technical charter for the tensor area dialects, interfaces, transforms and general surrounding infrastructure, compatible with the rest of MLIR.

First, it needs to define the scope by agreeing on a short list of major directions we're going, for example:



Tensor Compiler Design Group

 [MLIR Tensor Compiler Design Group](#) by Renato

Goal: charter across relevant dialects, interfaces, transforms, surrounding infrastructure

Scope

- linear algebra – tensor semantics – bufferization – memref semantics – vector semantics
- canonicalization (op aliasing within linalg) – ingress & egress – dialect design – flows

Tensor Compiler Design Group

 [MLIR Tensor Compiler Design Group](#) by Renato

Goal: charter across relevant dialects, interfaces, transforms, surrounding infrastructure

Scope

- linear algebra
- tensor semantics
- bufferization
- memref semantics
- vector semantics
- canonicalization (op aliasing within linalg)
- ingress & egress
- dialect design
- flows

Documentation updates

- Consolidate conflicting roadmaps
- Update outdated rationale docs
- Start an overarching charter

Tensor Compiler Design Group

 [MLIR Tensor Compiler Design Group](#) by Renato

Goal: charter across relevant dialects, interfaces, transforms, surrounding infrastructure

Scope

- linear algebra
- tensor semantics
- bufferization
- memref semantics
- vector semantics
- canonicalization (op aliasing within linalg)
- ingress & egress
- dialect design
- flows

Documentation updates

- Consolidate conflicting roadmaps
- Update outdated rationale docs
- Start an overarching charter

Infrastructure for distributed usage

- Devise a path for more flexibility for users (also across upstream projects)
- E.g., easier dialect extensions, canonicalized transform requirements, better coverage

Tensor Compiler Design Group – Members

 [Volunteers](#) from active contributors and representative stakeholders

Alex Zinenko
Brium

Renato Golin
Intel

Jacques Pienaar
Google

Matthias Springer
Nvidia

Quinn Dawkins
AMD

Javed Absar
Qualcomm

Jakub Kuderski
AMD

Suraj Sudhir
Arm

Rolf Morel
Intel

Andrzej Warzynski
Arm

Diego Caballero
Nvidia

Kunwar Grover
AMD

Tensor Compiler Design Group – Members

 [Volunteers](#) from active contributors and representative stakeholders

Alex Zinenko
Brium

Renato Golin
Intel

Jacques Pienaar
Google

Matthias Springer
Nvidia

Quinn Dawkins
AMD

Javed Absar
Qualcomm

Jakub Kuderski
AMD

Suraj Sudhir
Arm

Rolf Morel
Intel

Andrzej Warzynski
Arm

Diego Caballero
Nvidia

Kunwar Grover
AMD

***In bold**: MLIR Area Team, a distinct governance effort



Tensor Compiler Design Group – Members

Volunteers from

Modular

Democratizing AI Compute, Part 8:
What about the MLIR compiler infrastructure?



CHRIS LATTNER

Alex Zinenko
Brium

Quinn Dawkins
AMD

Rolf Morel
Intel

Matthias Springer
Nvidia

Suraj Sudhir
Arm

Kunwar Grover
AMD

A New Hope: Improved MLIR Governance

The tensions have simmered for years—and they're deeply felt across the broader LLVM and MLIR communities.

Fortunately, **there's a new hope**: LLVM is a meritocratic community with a long track record of aligning engineers—even when their companies are at war in the market.

The MLIR community is filled with amazing engineers who have poured years of their hearts and souls into improving the project to work through these challenges, and progress is now happening!

MLIR now has a new Area Team to help guide its evolution, along with a new organizational structure and charter and governance group. The charter defines separate area groups: MLIR Core (the domain-independent infrastructure), and the dialects (like the machine learning-specific pieces). I am extremely thankful to everyone who is spending time to improve MLIR and work through these issues—such work has a profound impact on everyone building into the ecosystem as well as the downstream users.

***In bold**: MLIR Area Team, a distinct governance effort



Tensor Compiler Design Group – Modus Operandi

- Regular meetings
 - Bi-weekly group meetings
 - Agenda & notes: [!\[\]\(e08ab59d3d686bd6105d67ebbb31fce5_img.jpg\) MLIR Tensor Compiler Design Group - Meetings](#)
 - ... only had three (3) so far
 - First Open Design Meeting scheduled late April (date TBC)

Tensor Compiler Design Group – Modus Operandi

- Regular meetings
 - Bi-weekly group meetings
 - Agenda & notes: [!\[\]\(ac263877d47c6367d00a5ecd10c3462b_img.jpg\) MLIR Tensor Compiler Design Group - Meetings](#)
 - ... only had three (3) so far
 - First Open Design Meeting scheduled late April (date TBC)
- In practice, members bring in-the-pipeline, community-relevant topics
 - In essence *pre-RFCs* — workgroup provides space for quick iteration
 - Rapid top-of-mind responses for low overhead feedback
 - Next: posted as RFC, with same consensus process as any other RFC

Tensor Compiler Design Group – Modus Operandi

- Regular meetings
 - Bi-weekly group meetings
 - Agenda & notes: [!\[\]\(ffa66d35b38fa61d1ea0ce291a91d082_img.jpg\) MLIR Tensor Compiler Design Group - Meetings](#)
 - ... only had three (3) so far
 - First Open Design Meeting scheduled late April (date TBC)
- In practice, members bring in-the-pipeline, community-relevant topics
 - In essence *pre-RFCs* — workgroup provides space for quick iteration
 - Rapid top-of-mind responses for low overhead feedback
 - Next: posted as RFC, with same consensus process as any other RFC
- All workgroup-generated documentation is public
 - [!\[\]\(f8c89cfe56739d1cce53f5841cedce81_img.jpg\) MLIR Tensor Compiler Design Group - Overview document](#)

Tensor Compiler Design Group – Progress on Vector

[MLIR Tensor Compiler Design Group - Vector Dialect: Refactoring + Re-design ideas](#)

RFCs which benefitted from live discussion:

[\[RFC\] Allow pointers as element type of `vector`](#)

- Upshot: VectorElementTypeInterface with semantics of only allowing “atomic” elements

[\[RFC\] Improving gather codegen for Vector Dialect](#)

- Addresses abstraction gap which lead to early loss of structured indexing

[\[RFC\] Generalize tiling to operate on ShapedType](#)

- Proposes extending infrastructure for tiling/blocking beyond linalg-on-tensor/memref
- In-the-pipeline RFC on reducing references to LLVM LangRef in vector dialect docs

Tensor Compiler – other recent significant changes

[Transpose attribute for Linalg matmul operations](#)

- Linalg.matmul (and batch_matmul) now have an affine_maps attribute

[Introduce linalg.contract](#): $D[J] = (\bigoplus_{((I^A \cup I^B) \setminus J)} A[I^A]^* B[I^B]) \oplus C[J]$

- Op generalizing all contraction ops (e.g. matmul variants and matvec and dot and ...)

[Extend Linalg elemwise named ops semantics](#)

- linalg.elementwise with affine_maps replacing linalg.elem_wise_{unary,binary}

[Move `tensor.pack` and `tensor.unpack` into Linalg](#)

- Fix layering issue; facilitate interactions with linalg ops; allow for  [packing on memrefs](#)

Topics we are looking to make progress on

1.  [RFC] Should we restrict the usage of 0-D vectors in the Vector dialect?
2. Remove ops with overlapping functionality from Vector, e.g. extractelement -> extract
3. Various issues in Vector regarding consistent naming and semantics
4. Pros & cons of splitting vector dialect into high-level and low-level parts
5. Vector vs Tensor types – another go at clarifying their distinctive roles

Topics we are looking to make progress on

1.  [\[RFC\] Should we restrict the usage of 0-D vectors in the Vector dialect?](#)
2. Remove ops with overlapping functionality from Vector, e.g. extractelement -> extract
3. Various issues in Vector regarding consistent naming and semantics
4. Pros & cons of splitting vector dialect into high-level and low-level parts
5. Vector vs Tensor types – another go at clarifying their distinctive roles
6. Revisit how to attach layouts to tensors & vectors
7. Enshrine in the charter the significance of projected permutation affine_maps
8. Pick up stalled progress on common-ing up linalg conv ops
9. Compute type on linalg ops given correspondence with imperfect loop nest
10. Op aliasing in linalg, e.g. linalg.matmul vs linalg.contract vs linalg.generic
 - Equiv. class perspective on matching w.r.t.  [\[RFC\] Linalg operation tree](#)

Tensor Compiler Design Group – next steps

- **More focus on documentation updates**
 - Many dialect-level documents are outdated, as are the rationale ones
 - We would love help with this!

Tensor Compiler Design Group – next steps

- **More focus on documentation updates**
 - Many dialect-level documents are outdated, as are the rationale ones
 - We would love help with this!
- **Start formulating the Tensor Compiler-spanning, cross-dialect charter**
 - New documents laying out intended coherent usage across dialects

Tensor Compiler Design Group – next steps

- **More focus on documentation updates**
 - Many dialect-level documents are outdated, as are the rationale ones
 - We would love help with this!
- **Start formulating the Tensor Compiler-spanning, cross-dialect charter**
 - New documents laying out intended coherent usage across dialects
- **Start codifying flows through the Tensor Compiler**
 - E.g., integration tests spanning just the Tensor Compiler dialects
 - In addition to explicitly described flows in the charter

Tensor Compiler Design Group – get in touch!



Alex Zinenko
([@ftynse](#))



Renato Golin
([@rengolin](#))



Jacques Pienaar
([@jpienaar](#))



Matthias Springer
([@matthias-springer](#))



Quinn Dawkins
([@qed](#))



Javed Absar
([@javedabsar](#))



Jakub Kuderski
([@kuhar](#))



Suraj Sudhir
([@sjarus](#))



Rolf Morel
([@rolfmorel](#))



Andrzej Warzynski
([@banach-space](#))



Diego Caballero
([@dcaballe](#))



Kunwar Grover
([@groverkss](#))



Developers' Meeting

BERLIN 2025



Accurate Runtime Performance Estimation for Predictably Training ML Guided Register Eviction Policies

Aiden Grossman

EuroLLVM 2025



Why do this in the first place?

Rewards are **critical** for training learned heuristics.



Trace-based Cost Modeling

Agenda

01

Trace Based Cost Modeling for Register Allocation

02

(Distributed) Training of Models

03

Further Improvements

Traces vs PGO Data

PGO-Based

```
.loop:  
    add    eax, dword ptr [rdi + 4*rdx]  
    inc    rdx  
    cmp    rcx, rdx  
    jne    .loop
```

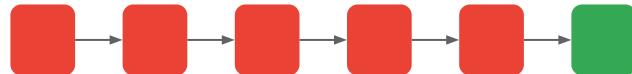
X5

```
.exit:  
    add    eax, 5  
    ret
```

X1

- Reward is a linear combination of instruction types multiplied by their block frequency.

Trace-Based



- Reward is some operation (typically cycles) over the sequence of instructions. We have several options to choose from:
 - Analytical CPU pipeline models.
 - ML based CPU models.
 - Raw Instruction Counting.

Sourcing Trace Data

- Previous work, llvm-mcad (EuroLLVM 2022 mshockwave@), (<https://youtu.be/ZGEP7JEIKNo>) that guided us down this direction used a QEMU plugin.
- Use DynamoRIO to produce traces due to good internal support.

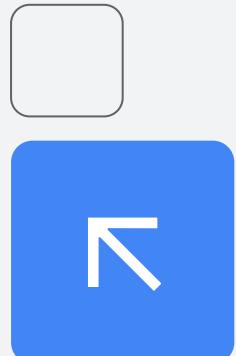


Basic Block Trace Modeling

We need to be able to collect a single trace and apply it to many variants of the same binary as rerunning each time defeats the point.

Sourcing Basic Block Information

Take Advantage of some of the Propeller Infrastructure
(Basic Block Address Maps).



BB Trace Extraction

How to turn an instruction stream into a BB stream that can be replayed.

- Add a basic basic block to the trace every time we encounter an instruction with a PC starting at the beginning of a BB.
- Sometimes we need to split BBs.
 - Call Instructions
 - TCMalloc RSeq
 - Inline Assembly

```
.loop:  
    add    eax, [rdi + 4*rdx]  
    inc    rdx  
    cmp    rcx, rdx  
    jne    .loop  
.exit:  
    mov    add, 5  
    ret
```

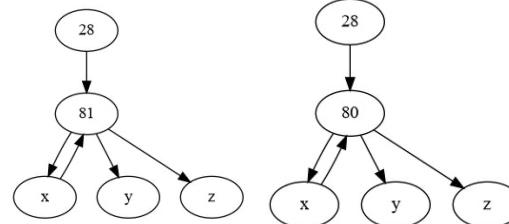
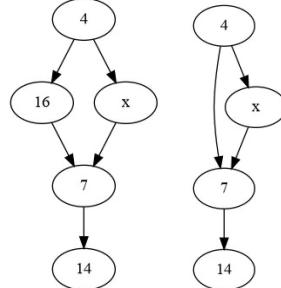


Basic Block Trace Modelling

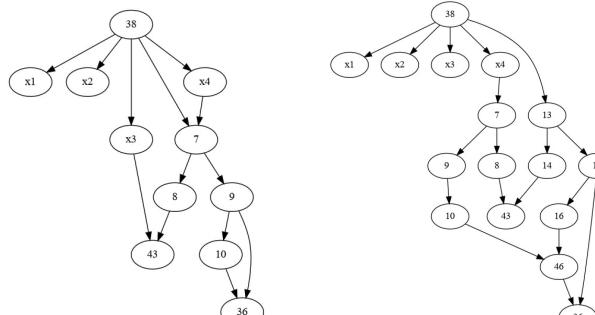
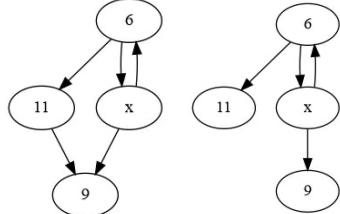
Becomes as simple as loading BBs from the
binary/compiled corpus.



Reconciling CFG Differences - The Problem



- Most cases are trivially reconcilable.
- Some cases are impossible to reconcile without additional information.



Reconciling CFG Differences - The Side Step



Disabling 3.5
Passes eliminates all
CFG Differences

Findings

1. Disabling three (believed to be) non-regalloc-coupled passes eliminates all CFG differences.
2. Disable one option on another pass (the half a pass).
3. Simple trick allows for much simpler BB trace modelling design.

BB Traces are Reasonably Close to Source Traces



0.5% missing instructions

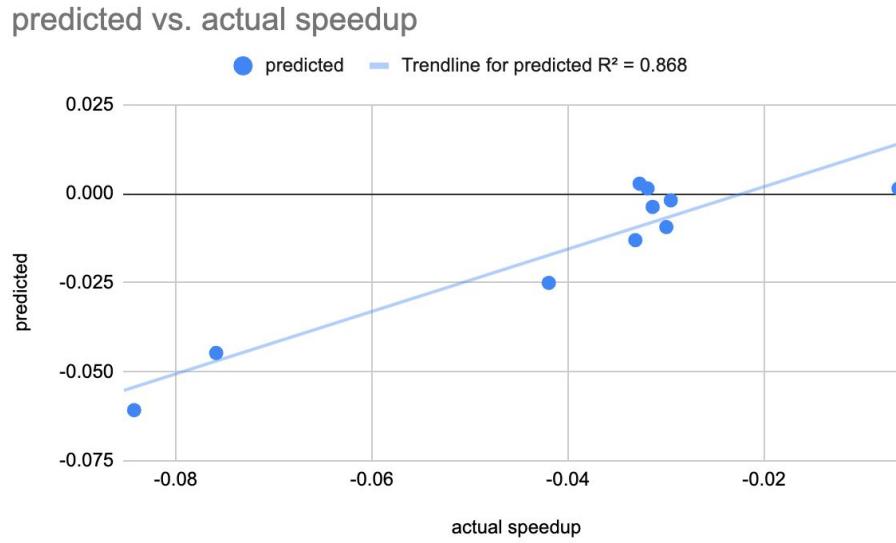
On traces upwards of 10M instructions.

Some cases we are not handling currently like interrupted restartable sequences and symbols from assembly files. No theoretical obstacles to completely fixing the gap.

It Even Works With PGO+CSPGO+ThinLTO!

For skylake, measuring runtime in cycles. Albeit with a large constant offset.

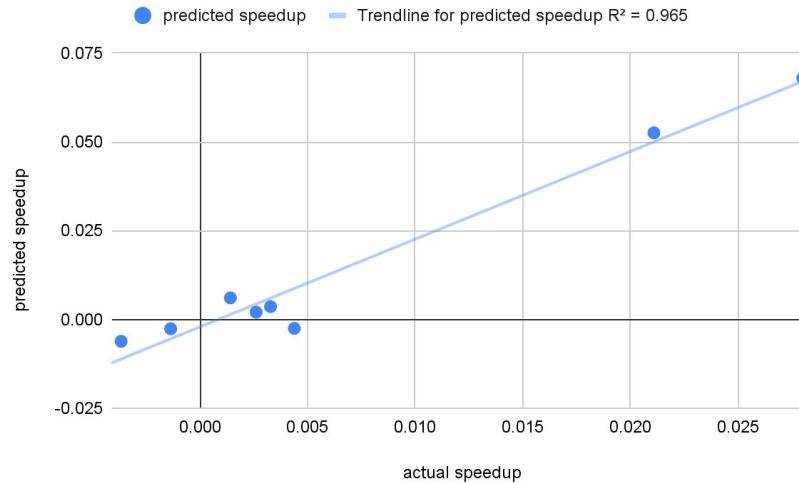
Predicted Speedup vs. Actual Speedup



The Non-PGO case works as well:

`opt -passes="default<O3>" -disable-output` on StructuralHash.cpp from LLVM. ~10M retired instructions.

Predicted Speedup vs. Actual Speedup

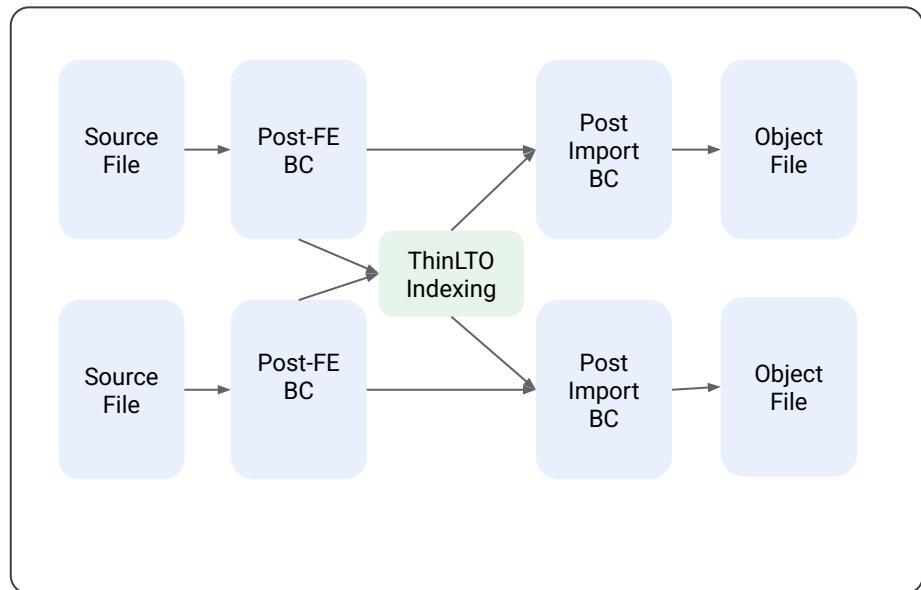


Training with Reinforcement Learning

Background - The Corpus

How should we efficiently collect training data?

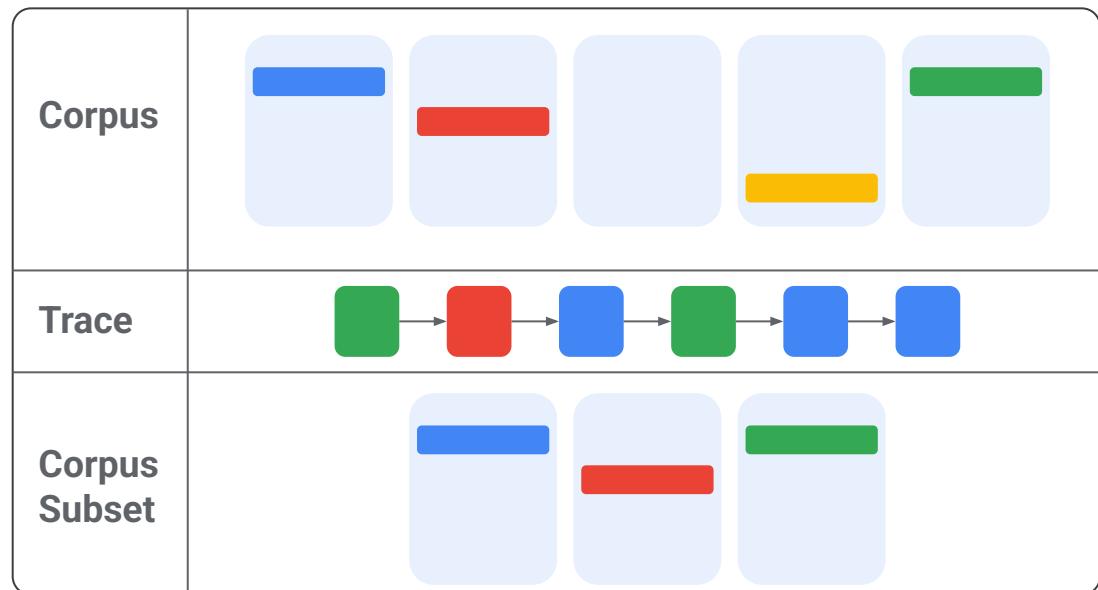
We collect **LLVM Bitcode** for all translation units involved in the final link.



Corpus Subsetting

How do we efficiently evaluate models?

- Find the minimum set of translation units covering the entire set of functions in the trace.
- Pull them to the side.



Background - Training Setup

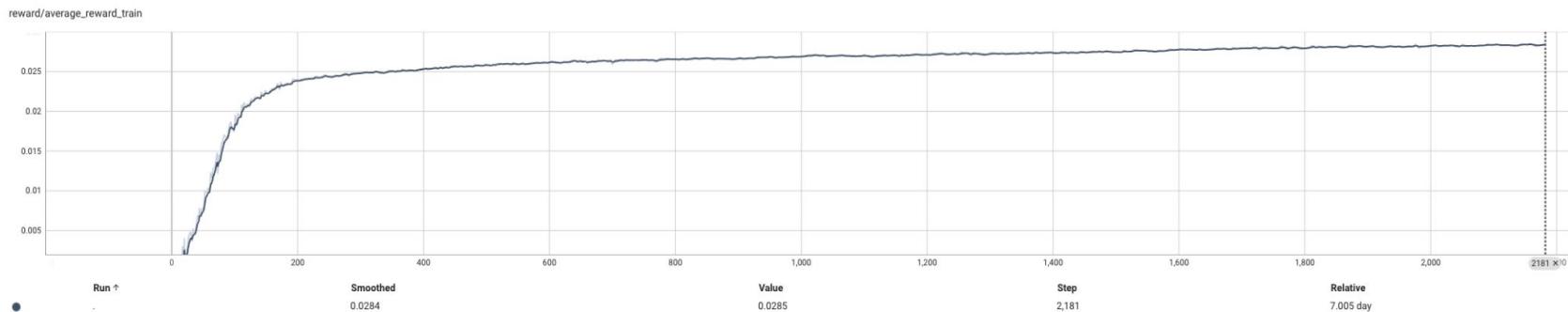
How do the requirements on the ML side interface with the cost model/compiler side?

- We use ES (Evolutionary Strategies) as our training algorithm.
 - Simple math, relatively easy to understand.
 - Enables long trajectories - We can give feedback on many individual decisions and the algorithm will still adjust the policies appropriately.
 - Has a set of perturbations for each iteration.
- Utilize existing training infrastructure
 - But scaled given now we need to compile an entire corpus subset to get a signal rather than a handful of modules.
- First experiments were performed with the same opt invocation from earlier. ~10M retired instructions.

Training Results

It trains! Somewhat slowly... (for LLVM opt)

Reward (Predicted speedup over baseline)



- 100 Machine Slices (1600 Threads)
- 100 Perturbations per iteration
- ~800 Modules
- ~7 Days of training time
- 0.5% Real World Performance Improvement over an already peak optimized (PGO+CSPGO+ThinLTO) binary.

So Why Does This Matter?



We have a **validated**,
predictive cost
model for real
applications.

Distributed Training

Distributing the Training Process

Using more machines will at least help.

01

Parallelize individual workers

We use a threadpool within each worker to enable parallel compilation with the modelling component already being parallelized.

02

Spawn a bunch of distributed workers.

We use XM to manage experiments, with each worker being given about 32CPU cores, giving us reasonable scalability.

03

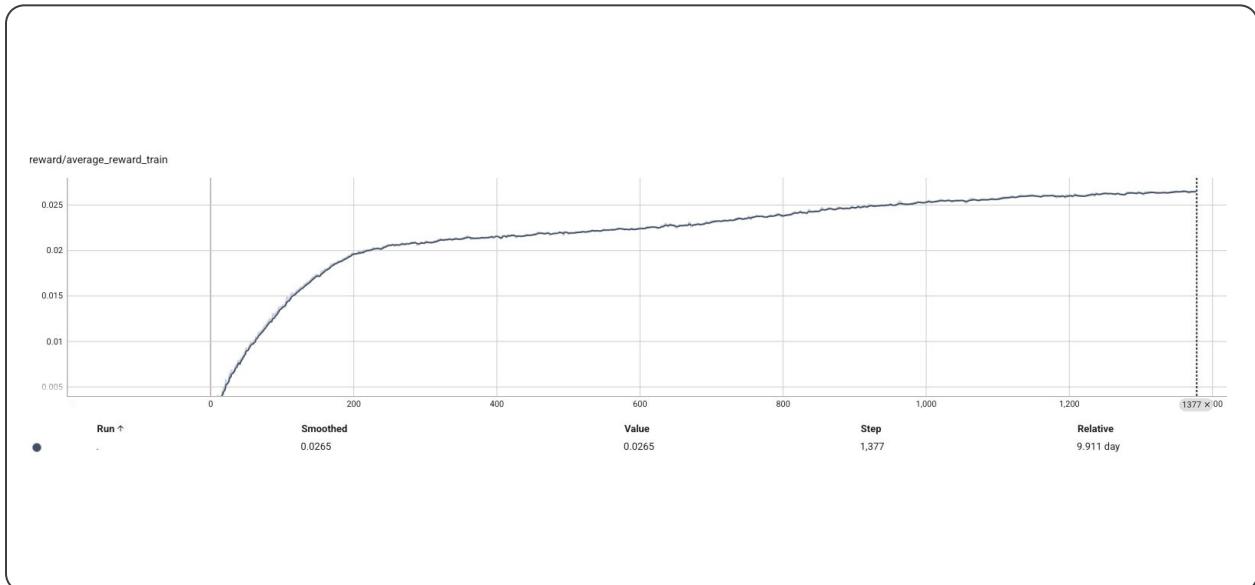
Profit (Somewhat)

This dropped iterations times to about five minutes. The latency of an individual model evaluation precludes us from going faster. We can evaluate many perturbations in parallel.

Shipping a Model

Training - Training Hyperparameter Tuning/RL

RL training started to go significantly faster when we realized the learning rate could be increased 10x with no ill-effect. More experimentation is still needed.



Some Reasonable Initial Performance Improvements

This is from a new [model](#) trained on a single workload. It ends up generalizing reasonably well.



Internal Server App 1

- 0.34ms latency on action 1
- 0.24ms latency on action 2



Internal Search App 1

```
Benchmark Setup 1:  
---- total:qps ----  
diff: +0.32% ±0.088%  
---- <workload1 cpu kcycles> ----  
diff: -0.75% ±0.198%  
Benchmark Setup 2:  
---- <workload2 cpu kcycles> ----  
diff: -0.46% ±0.268%
```



Internal Search App 2

```
-- <round trip latency> --  
diff: -0.64% ±0.372%
```

Current and Future Work



During training, only compiling functions with regalloc decisions causes compile time drops from 3-4 minutes to 5-10s.

Modeling time remains about the same as it is bottlenecked by MCA. We have some ideas on how to fix that...



Reducing Modeling Costs

Now that compile times have been drastically reduced, modeling costs dominate. It would be good to reduce them too.

- Only modeling changing functions might provide significant benefits.
 - Benefits depends upon how many functions they call.
 - Needs empirical validation.
 - Natural extensions (like excluding blocks from functions that get called) also need separate validation.
- Trace subsetting - Only evaluate a subset of the traces on each invocation.

Future Work

Future Work

Understanding Constant Offsets

- Understand why our model is producing large constant offsets.
- Hopefully leads to better models.
- Experiment with other modelling techniques (ML based, etc.)

Ship Better Models

- Utilize more efficient training techniques, train better models and ship them.

What do we need to do to Generalize for other Optimizations?

- Control flow graph reconciliation?
- Keep track of data/inputs to interpret (M)IR?
- Something else?
- Better cost modelling techniques?

Thank You!





Developers' Meeting

BERLIN 2025



An Update on LLVM Premerge Testing

The New Beta System

Beta Premerge System Status

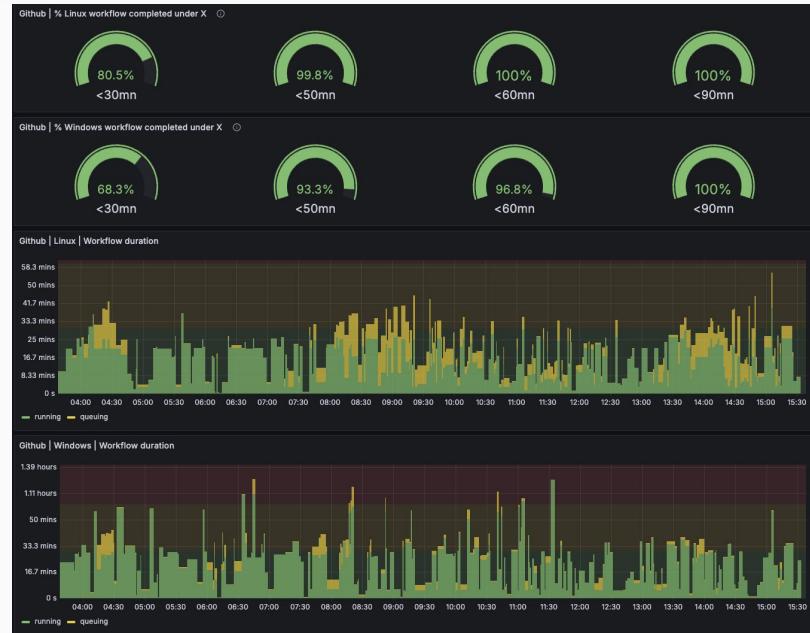
- Both old and new systems are running in parallel
- But the new system marks all jobs as passing
- Sorting out a few stability problems
- Launch expected in Q2

Beta System Features

- Autoscaling means more machines at peak load (workday afternoons Pacific time), meaning better latency and typically <20 minutes of queue time.
- Analytics
- Oncall team support during working hours
- Dedicated engineering staff for maintenance and improvements

Metrics

- Displays key metrics that have significantly impacted premerge functionality in the past.
- Metrics is coupled with alerting to notify an on-call rotation when things go awry.



Beta System Features/Improvements

Done:

- Computing what projects to test is now much simpler and unit tested.

Exploring:

- Improving node/container setup times.
- Full reproduction instructions using containers.
- Faster toolchain on Windows?

Beta System Launch Q2

- Launch in this case means marking the new system premerge tests as authoritative, meaning a failing test will report as failed to GitHub
- At this time we will also turn down the old presubmit infrastructure

Testing Resources Prioritization

- There's always a trade-off between test coverage and premerge latency
- More test coverage means more premerge latency
- Google will do its best to deliver a high-reliability and high-performance system
- But beyond that, these tradeoffs are best made by the community
- We will defer coverage decisions to the Infrastructure Area Leads

Credits

From within Google:

- Aiden Grossman
- Caroline Tice
- Guillaume Chatelet
- Lucile Rose Nihlen
- Nathan Gauër

And the following OSS contributors:

- David Spickett
- Tom Stellard

Questions?

Thank You!

(Questions?)





Developers' Meeting

BERLIN 2025



Measuring the Health of the LLVM Community

Jeremy Bennett

Copyright © 2025 Embecosm. Freely available under a
Creative Commons Attribution-ShareAlike license.





The *git* Repository

Copyright © 2025 Embecosm. Freely available under a Creative Commons Attribution-ShareAlike license.

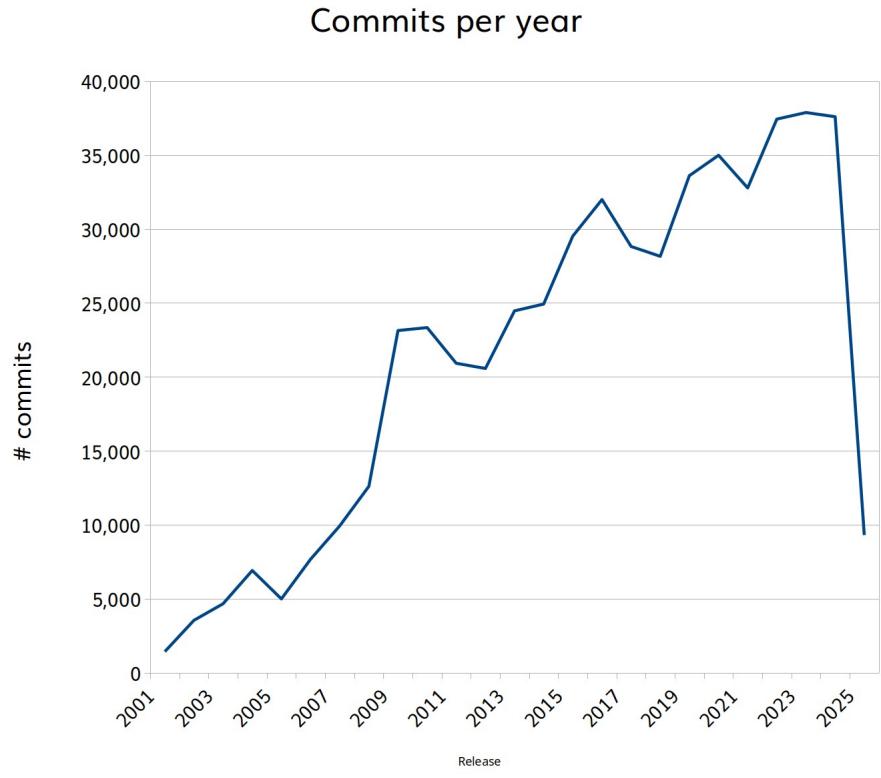
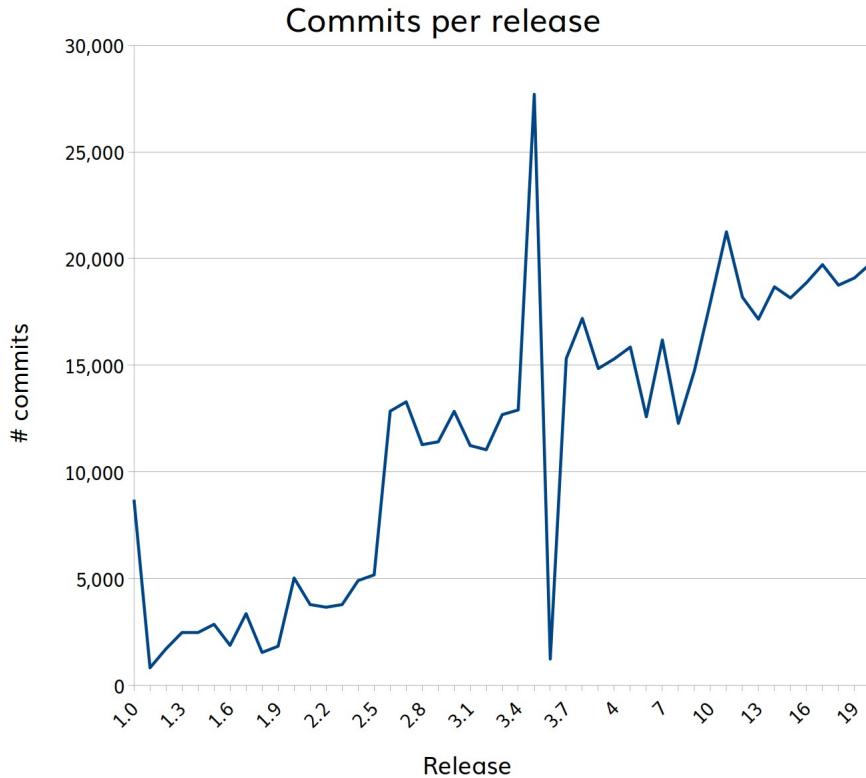


Counting commits

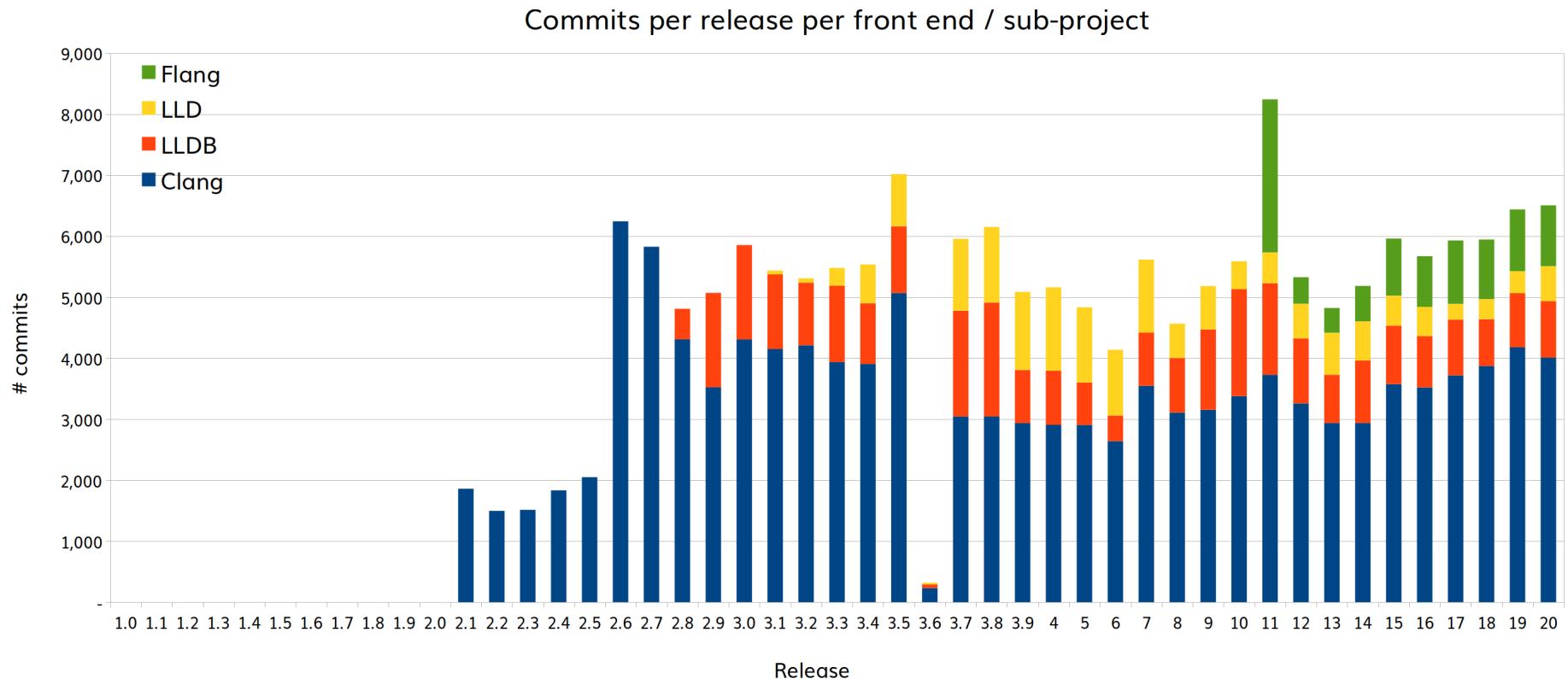
```
git log --oneline --no-merges remotes/upstream/release/20.x \  
^remotes/upstream/release/19.x | wc -l
```

```
git log --oneline --no-merges --since-as-filter="2024-01-01" \  
--until="2024-12-31" | wc -l
```

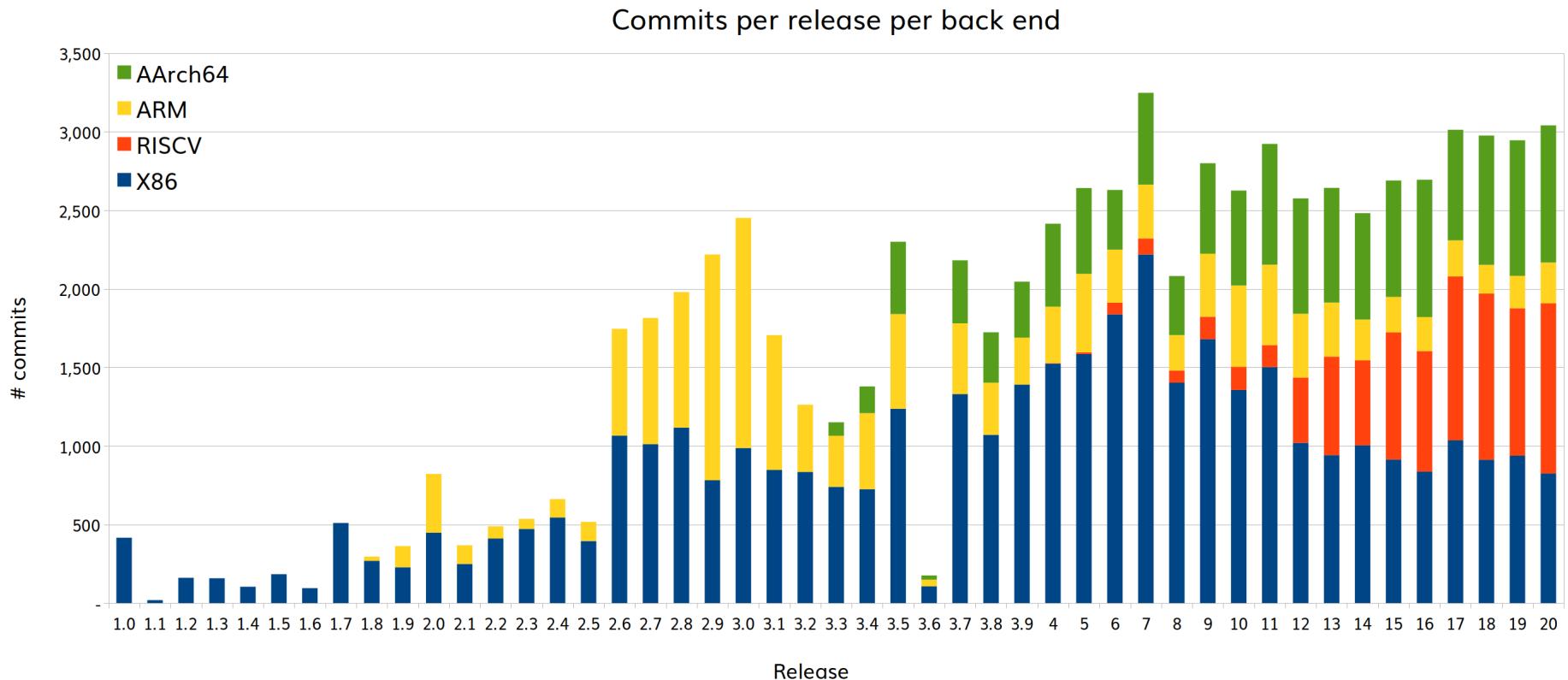
Counting Commits



Front End & Sub-Project Activity (Commits)



Back End Activity (Commits)



Counting Contributors

```
git log --no-merges remotes/upstream/release/20.x \
    ^remotes/upstream/release/19.x \
    --pretty="format:%an" | sort -u | wc -l
```

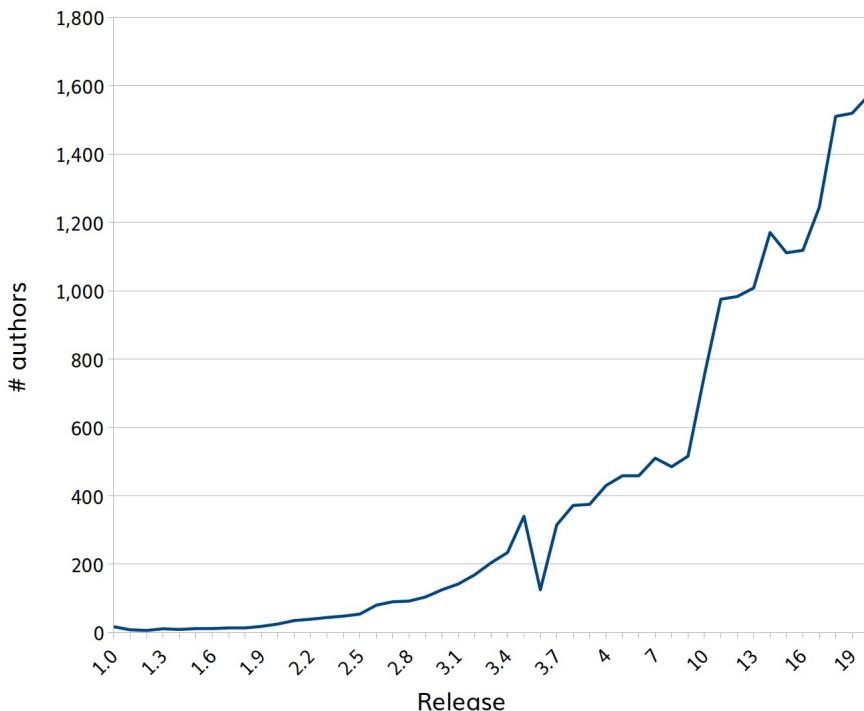
```
git log --no-merges remotes/upstream/release/20.x \
    ^remotes/upstream/release/19.x \
    --pretty="format:%ae" | sed -e 's/^.\+\@//' | sort -u
```

Then post-process to remove generic email domains (gmail etc) and any domain with a single contributors

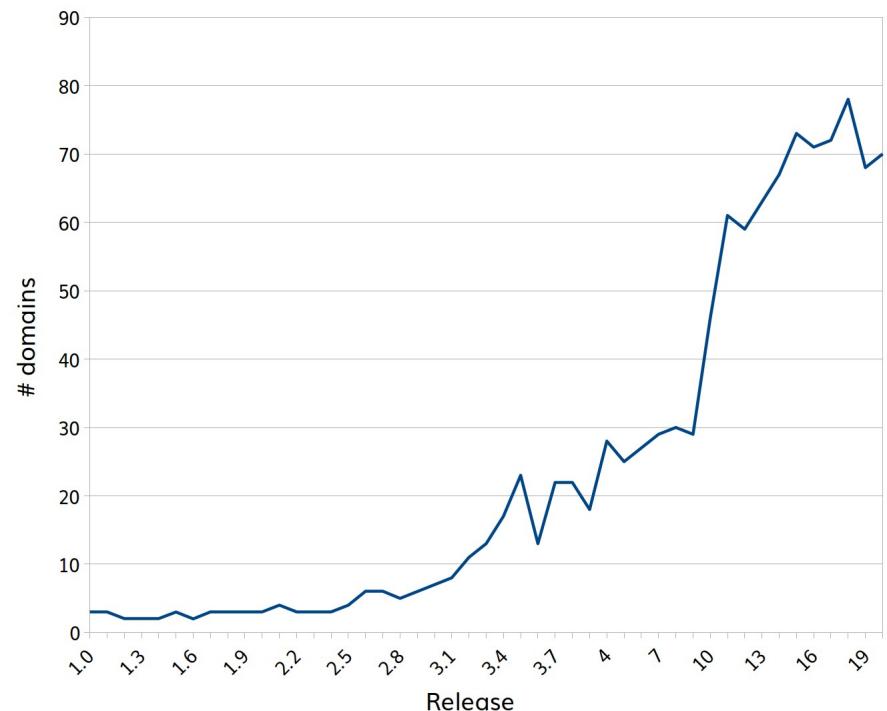


Counting Contributors

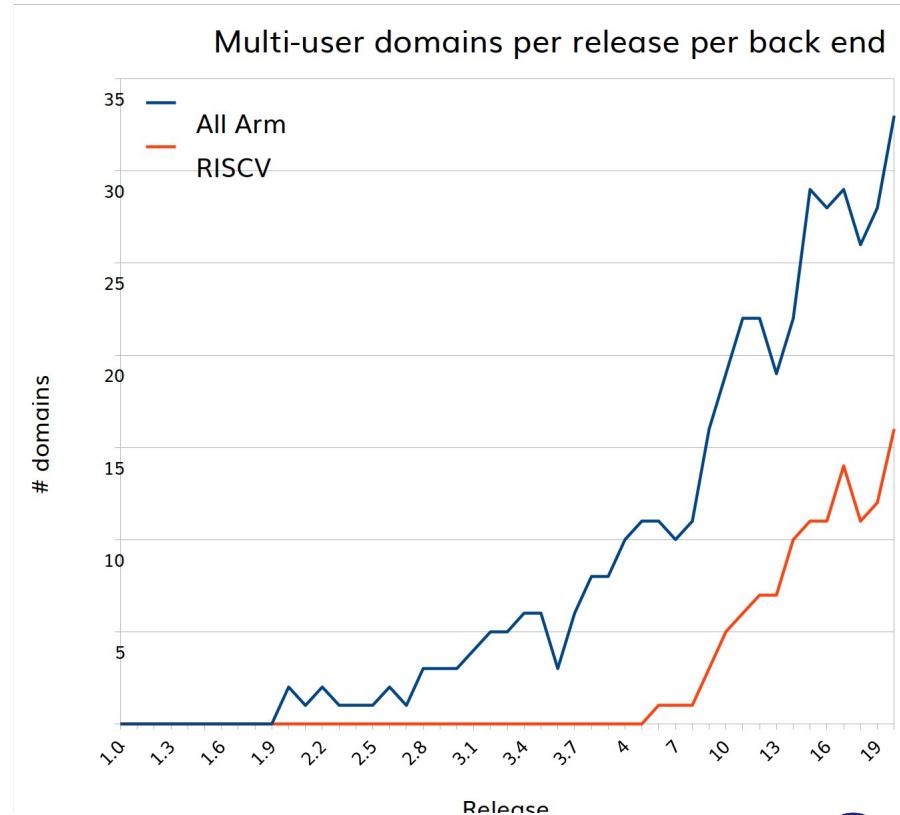
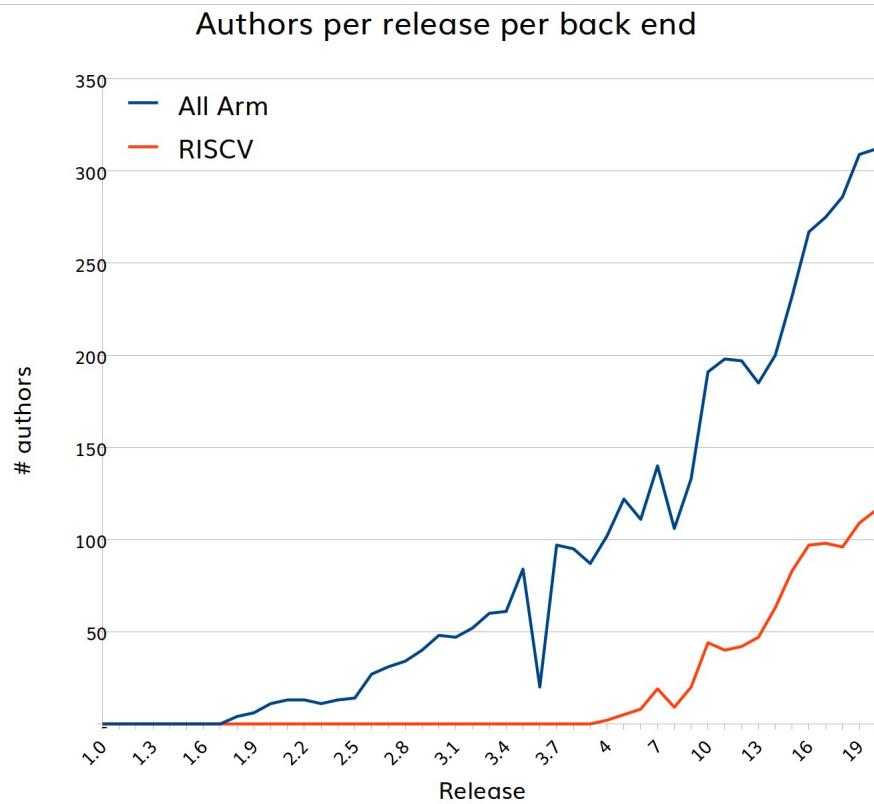
Authors per release



Multi-user domains per release



The Detail: Comparing Back Ends



The Detail: LLVM Supercommitters

Release 20

- 1,571 individuals
- 70 “corporate” domains
- 63 individuals committed
more than once per week
- Highest individual: 767
commits



The Detail: LLVM Supercommitters

Release 20

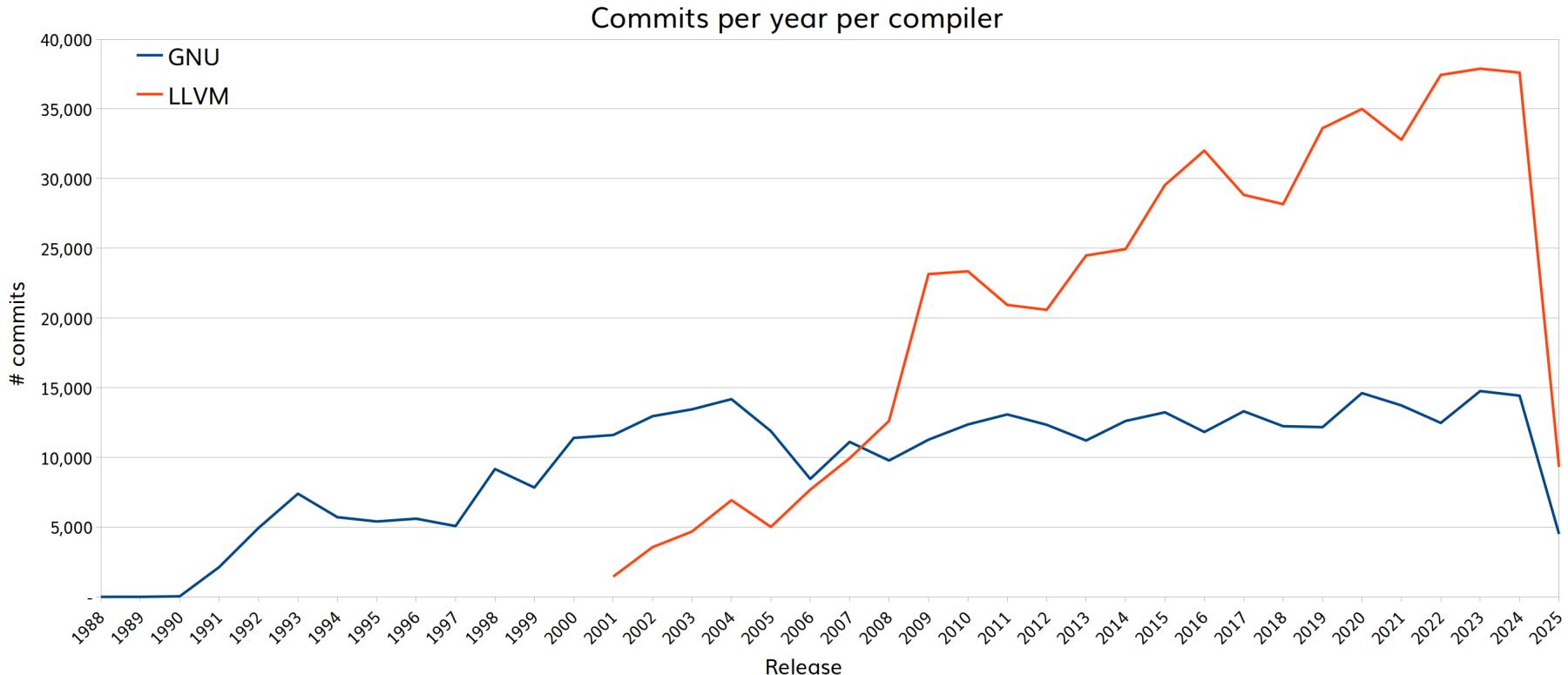
- 1,571 individuals
- 70 “corporate” domains
- 63 individuals committed more than once per week
- Highest individual: 767 commits

Release 1.0

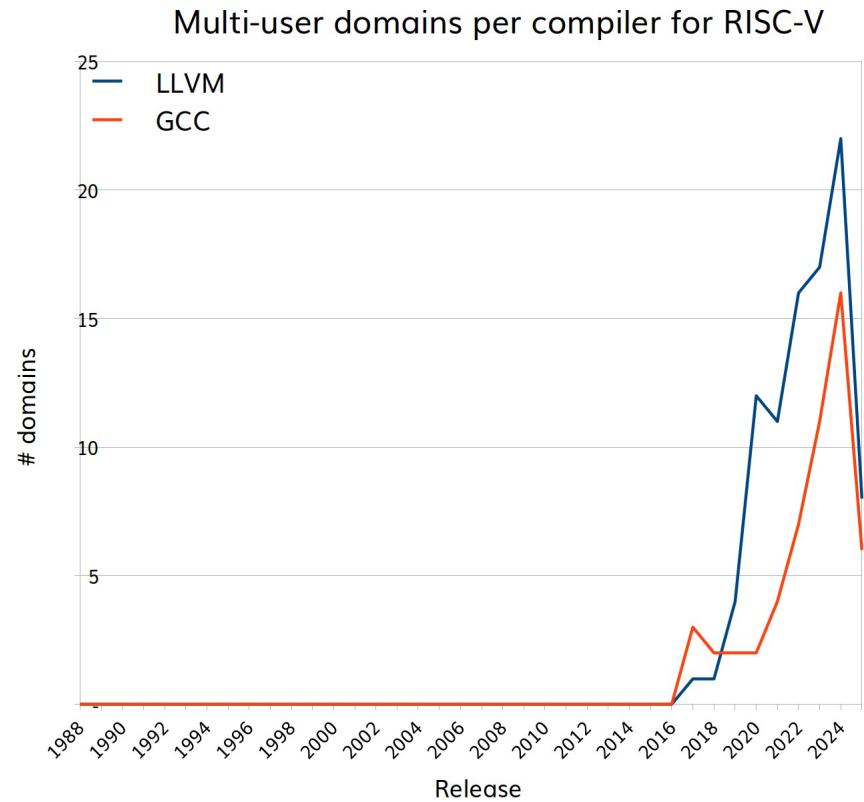
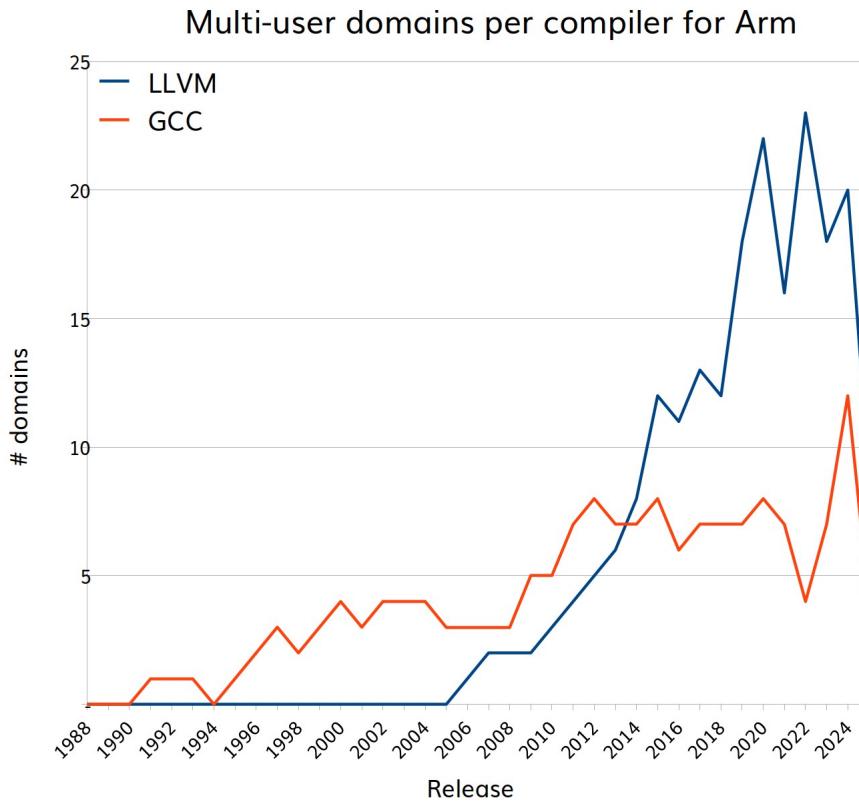
- 17 individuals
- 2 “corporate” domains
- 7 individuals committed more than once per week
- Highest individual: 6,511 commits



Comparing Compilers: Commits per Year



The Detail: Comparing Compiler Back Ends





GitHub using *gh*

Copyright © 2025 Embecosm. Freely available under a Creative Commons Attribution-ShareAlike license.



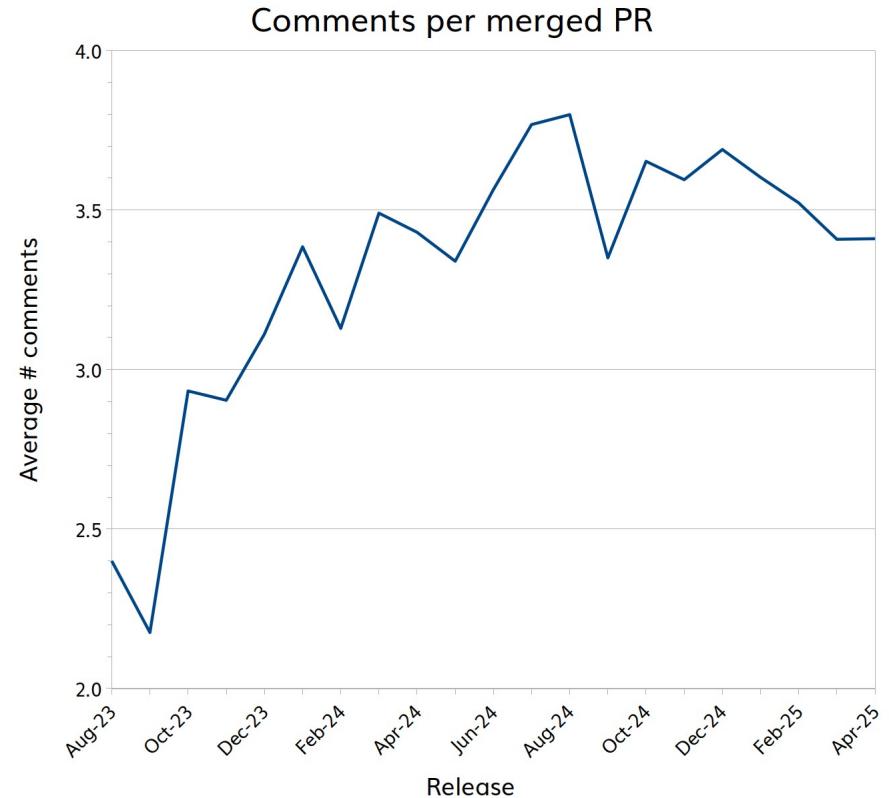
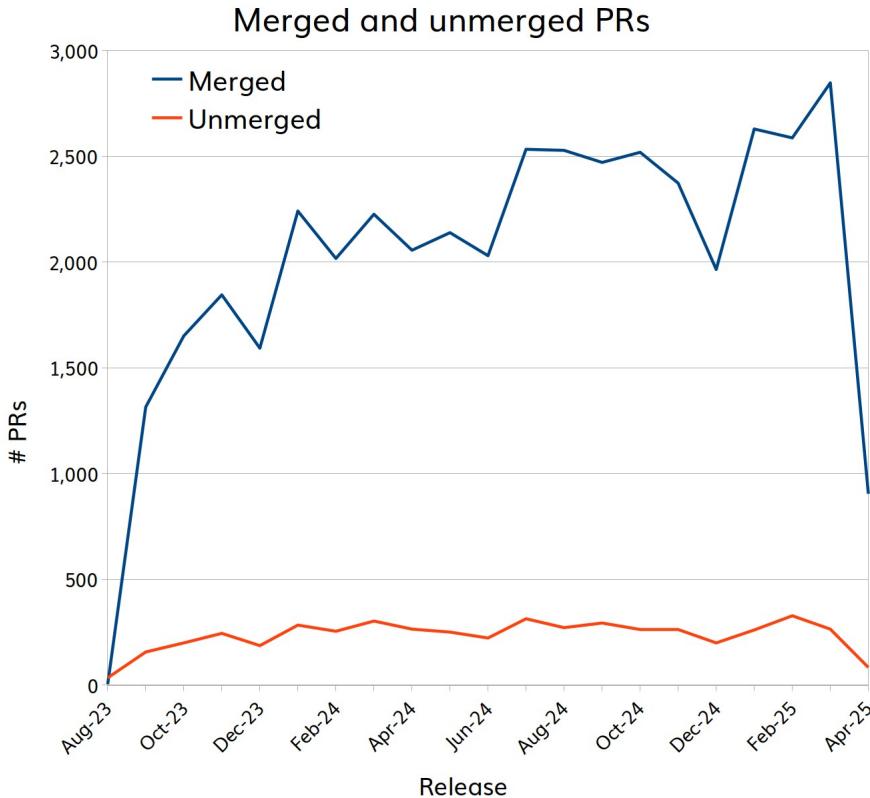
Analysing Pull Requests

```
gh pr list -L 1000000 -s closed \
    --json id,createdAt,closedAt,state,comments \
    -q '.. | .id?,.createdAt?,.closedAt?,.state?'
```

Post process with an `awk` script to generate a CSV file.



PRs: Comments per Commit



More Detailed Analysis: Diversity and Inclusion

The Inspiration



Prof Andrea Capiluppi
University of Groningen

- www.rug.nl/staff/a.capiluppi/
- Infer gender activity
 - how robust is this?
- Statistical analysis
- youtu.be/sTAtuSxwCss
- doi.org/10.1093/iwc/iwt047



Thank You

jeremy.bennett@embecosm.com

github.com/embecosm/toolchain-analyze

embecosm.com

Jeremy Bennett



Developers' Meeting

BERLIN 2025

Lessons learned from leveling up RISC-V LLVM testing

Alex Bradbury asb@igalia.com

EuroLLVM, 2025-04-15



Background

- RISC-V
- LLVM's buildbot infrastructure
 - Important: this is focused on post-commit CI



Challenges



Challenge:

Lack of high performance commodity RISC-V hardware with support for all desired instruction set extensions.



Challenge:

A full bootstrap build and test under qemu-system is incredibly slow (>16h).



Challenge:

Limitations in bug finding ability of a two-stage Clang build followed by running LLVM's unit tests.



Challenge:

Missing a flow for iterative developing and testing builder configurations prior to commit and deployment from llvm-zorg.



Challenge:

Existing buildbot worker configurations/recipes are hard to customise to the needs of the RISC-V setup. e.g. cross-compilation followed by running tests under qemu-system.



Challenge:

Difficult to reproduce a failure locally.



Challenge:

It can be time consuming to manually root cause a regression with multiple candidate commits.



Challenge:

There is no single “best” builder approach given the pros/cons of different emulation choices and trade-offs of test coverage and speed.



Challenge:

The buildbot interface is hard to navigate to check the status of RISC-V bots at a glance, so failures can go unnoticed.



Dashboard: <https://igalia.github.io/riscv-llvm-ci/>

RISC-V LLVM CI status

Generated at 2025-04-06 13:59. All times were recalculated in your local timezone (Europe/London). Regenerated approximately every 20 minutes.

Bot	In progress build	Previous build
clang-riscv-rva20-2stage ► Info	● #802 1h31m ago	● #801 1h32m · 2025-04-06 12:28
clang-riscv-rva23-evl-vec-2stage ► Info	● #641 13m ago	● #640 1h57m · 2025-04-06 13:46
clang-riscv-rva23-2stage ► Info	● #531 5h59m ago	● #530 15h21m · 2025-04-06 07:59
libc-riscv64-debian-dbg ► Info		● #12799 9m · 2025-04-06 12:36
libc-riscv64-debian-fullbuild-dbg ► Info	● #11903 1m ago	● #11902 4m · 2025-04-06 12:41
libc-riscv32-qemu-yocto-fullbuild-dbg ► Info	● #6722 1m ago	● #6721 37m · 2025-04-06 13:36

This dashboard and the clang-* bots operated by Igalia, supported by RISE▲.



Thank you

- Patch reviewers
- Everyone contributing to development and upkeep of LLVM's testing infrastructure.
- RISE for supporting this work.



Questions?

asb@igalia.com



Overflow

- Release testing

Future work:

- Performance testing and tracking.
- Wider ecosystem testing (e.g. large corpus of Linux packages)
- On-demand pre-commit CI.





Developers' Meeting

BERLIN 2025



What is LLDB-DAP?

Jonas Devlieghere

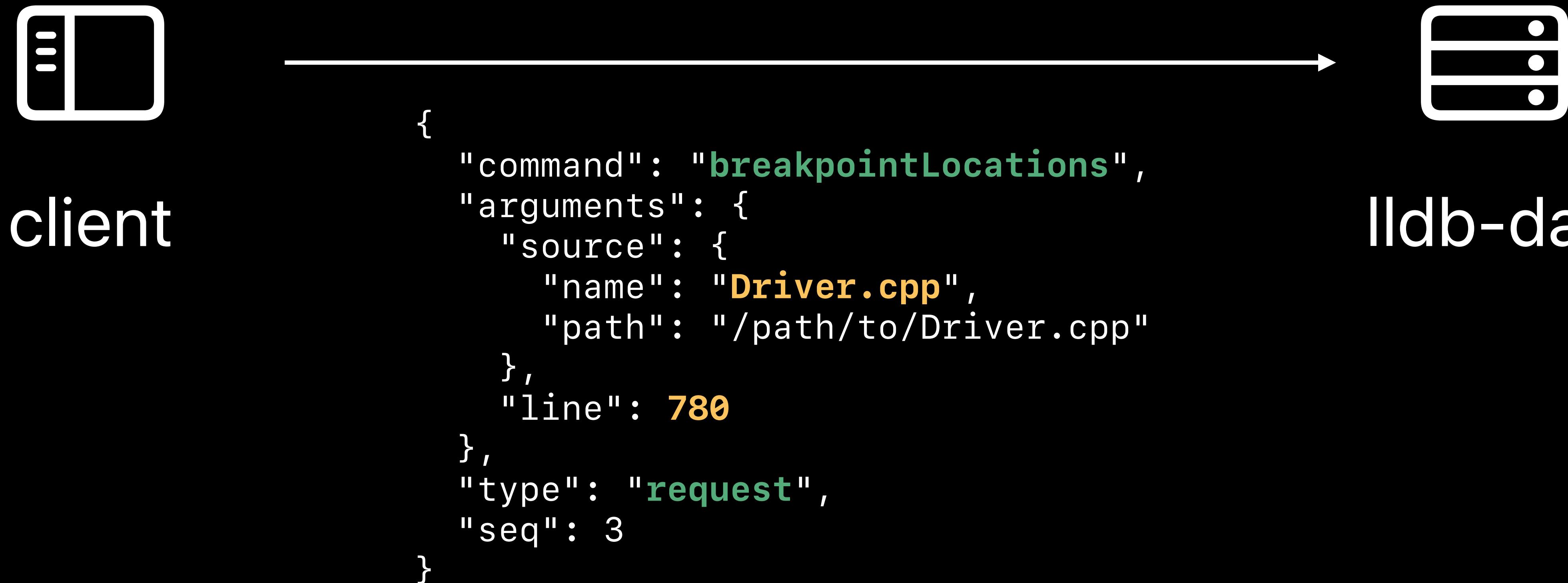
EuroLLVM 2025

Debug Adapter Protocol

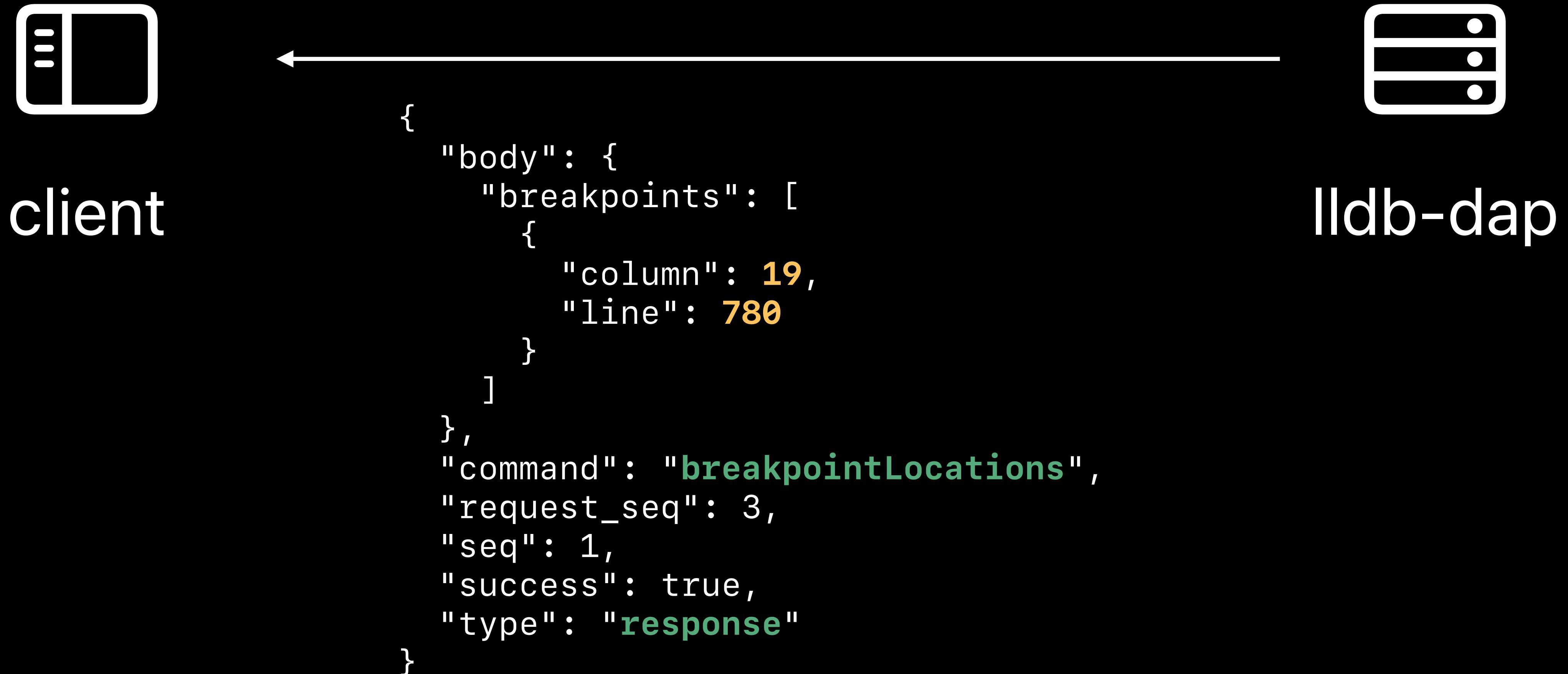
Debug Adapter Protocol



↗ Debug Adapter Protocol

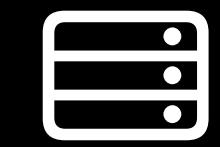


Debug Adapter Protocol



DAP in LLDB

DAP in LLDB



lldb-dap

The DAP server



LLDB-DAP

Visual Studio Code Extension



The DAP server

Standalone DAP server binary

- Part of your toolchain (part of Xcode 16 & LLVM 19)
- Uses the SB API and links against libLLDB

Can be used with any editor supporting the protocol

- Vim, Sublime, VS Code, Emacs, etc



The Visual Studio Code Extension

Visual Studio Code extension

- Available on the Visual Studio Marketplace
- Relies on the lldb-dap binary (*not included!*)

Integration into VS Code

- Settings UI
- Launch configuration templates

The screenshot shows a developer environment for the LLVM project, specifically in debug mode. The interface includes:

- Left Sidebar:** RUN AND DEBUG, VARIABLES (Locals, Globals, Registers), WATCH, CALL STACK, BREAKPOINTS.
- Main Area:** A code editor showing `Driver.cpp` with the following snippet:

```
namespace llvm {  
class raw_ostream {  
    static constexpr Colors BRIGHT_GREEN = Colors::BRIGHT_GREEN;  
    static constexpr Colors BRIGHT_YELLOW = Colors::BRIGHT_YELLOW;  
    static constexpr Colors BRIGHT_BLUE = Colors::BRIGHT_BLUE;  
    static constexpr Colors BRIGHT_MAGENTA = Colors::BRIGHT_MAGENTA;  
    static constexpr Colors BRIGHT_CYAN = Colors::BRIGHT_CYAN;  
    static constexpr Colors BRIGHT_WHITE = Colors::BRIGHT_WHITE;  
    static constexpr Colors SAVEDCOLOR = Colors::SAVEDCOLOR;  
    static constexpr Colors RESET = Colors::RESET;  
  
    explicit raw_ostream(bool unbuffered = false,  
                        OStreamKind K = OStreamKind::OK_OStream)  
        : Kind(K), BufferMode(unbuffered ? BufferKind::Unbuffered  
                                : BufferKind::InternalBuffer) {  
        // Start out ready to flush.  
        OutBufStart = OutBufEnd = OutBufCur = nullptr;  
    }  
  
    raw_ostream(const raw_ostream &) = delete;  
    void operator=(const raw_ostream &) = delete;  
  
    virtual ~raw_ostream();  
  
    /// tell - Return the current offset with the file.  
    uint64_t tell() const { return current_pos() + GetNumBytesInBuffer(); }  
};
```
- Bottom:** PROBLEMS (68), OUTPUT, DEBUG CONSOLE (active), TERMINAL, PORTS, GITLENS, Filter (e.g. text, !exclude, \escape). The DEBUG CONSOLE tab shows lldb output:

```
lldb revision 990a086d9da0bc2fd53a6a4c95ecbbe23a297a83  
warning: liblldb.21.0.0git.dylib was compiled with optimization - stepping may behave oddly; variables may not be available.  
→ image list  
(lldb) image list  
[ 0] 3B040485-D3E7-3CC2-AFE2-C41DFA559345 0x0000000100000000 /Users/jonas/llvm/build-ra/bin/lldb  
[ 1] DBE77528-DCE0-3EE7-AFC8-0DDC948E3793 0x0000000180cf8000 /usr/lib/dyld  
[ 2] AFD03688-5FAC-3B3D-96AA-9D88C034F23 0x00000001001f0000 /opt/homebrew/opt/zstd/lib/libzstd.1.dylib  
[ 3] 8D2F8721-A952-3777-9258-E6596AE318A1 0x000000010e76c000 /Users/jonas/llvm/build-ra/lib/liblldb.21.0.0git.dylib  
[ 4] 4A425015-9D36-3F8F-9814-A6C2A42F1BFB 0x00000001810a4000 /usr/lib/system/libdyld.dylib  
[ 5] AC7AF500-F60C-3198-8A67-AAD1F450E7ED 0x0000000180d93000 /usr/lib/system/libsystem_blocks.dylib  
[ 6] 0B00D666-9586-32FD-9BFD-D00859C99922 0x000000018efd8000 /usr/lib/system/libsystem_collections.dylib  
[ 7] 9987E5C5-ADAF-38B3-A6DB-C773331AA54D 0x000000026a720000 /usr/lib/system/libsystem_darwindirectory.dylib
```



TM and © 2025 Apple Inc. All rights reserved.



Developers' Meeting

BERLIN 2025



THE UNIVERSITY
of EDINBURGH

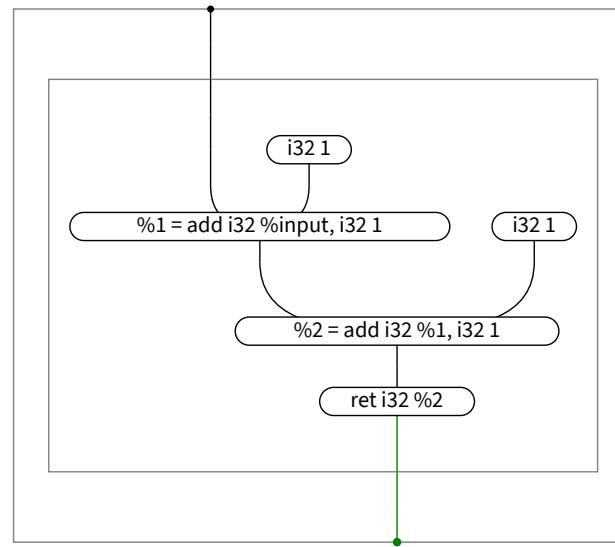
sd-visualiser: Interactive graph visualisation for SSA-based IRs

Alex Rice, Nick Hu, Calin Tataru

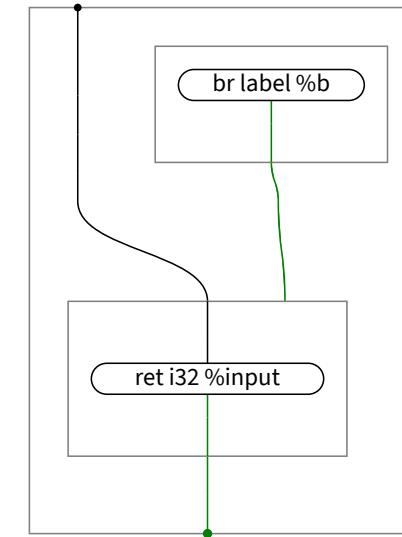
EuroLLVM 2025

Intermediate representations as a graph

Data flow

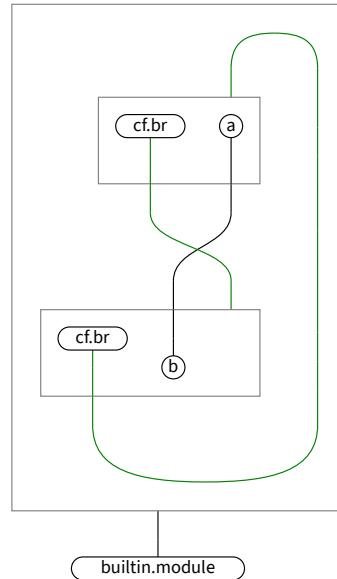


Control flow

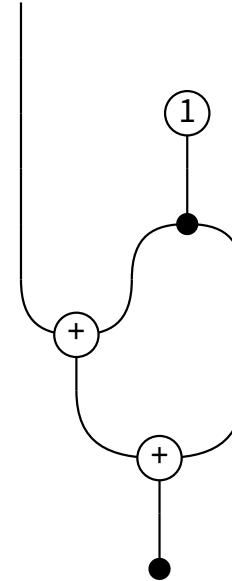


Representing IR features

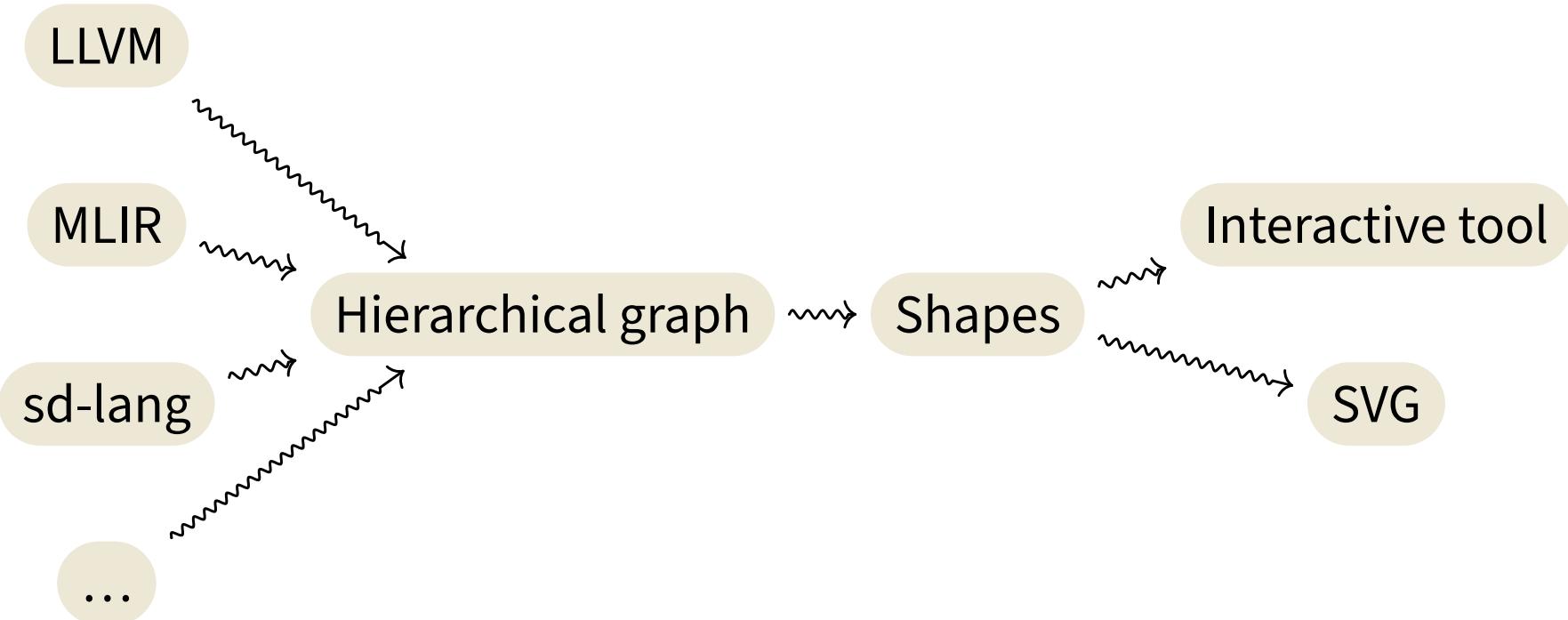
Blocks and regions



Copying and deletion

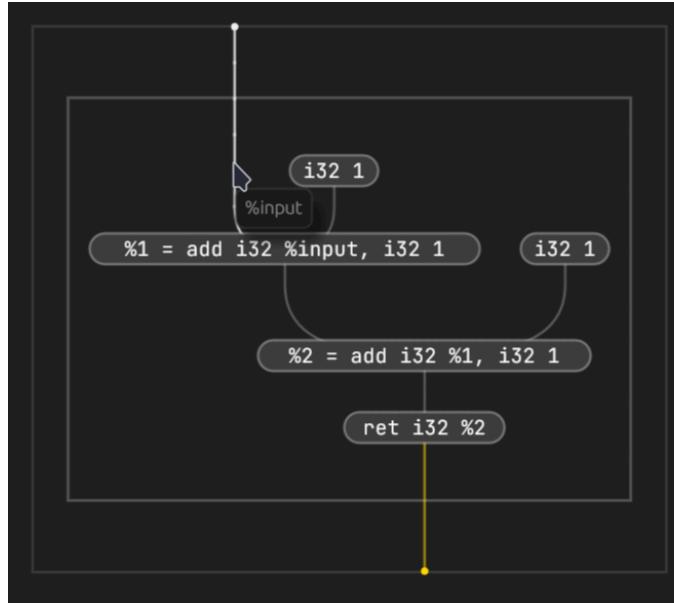


Visualising IRs

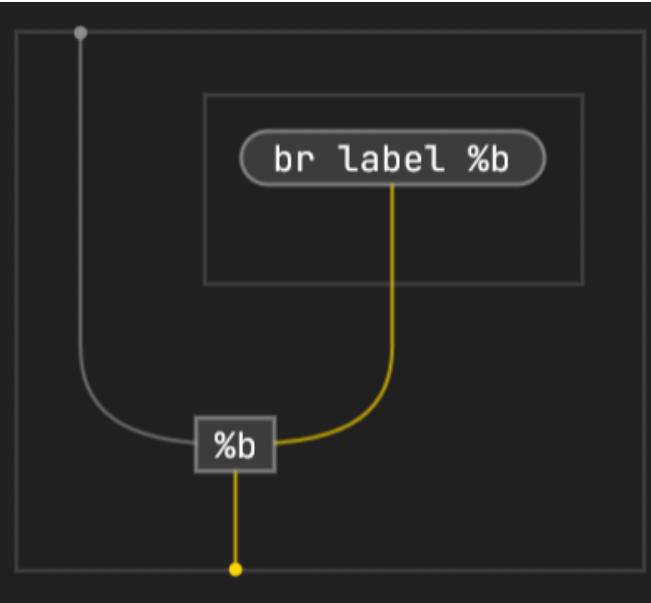


Interactive features

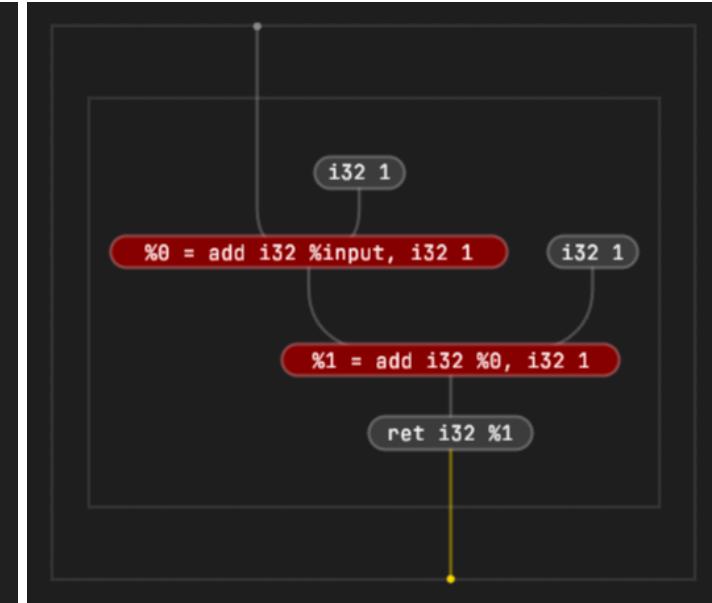
Highlighting



Collapsing



Searching



Try the tool today!

<https://sd-visualiser.github.io/sd-visualiser>





Developers' Meeting

BERLIN 2025



Beyond Pattern-based Optimization: What Can LLM Reshape *Auto-vectorization*?

Presenter: Long Cheng¹ (chenglong86@huawei.com)

Co-authors: Lu Li², Zhongchun Zheng^{1, 5}, Rodrigo Caetano de Oliveira Rocha³, Wei Wei¹, Tianyi Liu⁴, Li Zhou¹

Affiliations: ¹ Compiler Lab., Huawei Technologies Co., LTD, China

² Huawei Hongkong Research Institute, China HongKong

³ Huawei Edinburgh Research Institute, UK Edinburgh

⁴ Huawei Cambridge Research Institute, UK Cambridge

⁵ Sun Yat-sen University, GuangZhou, China



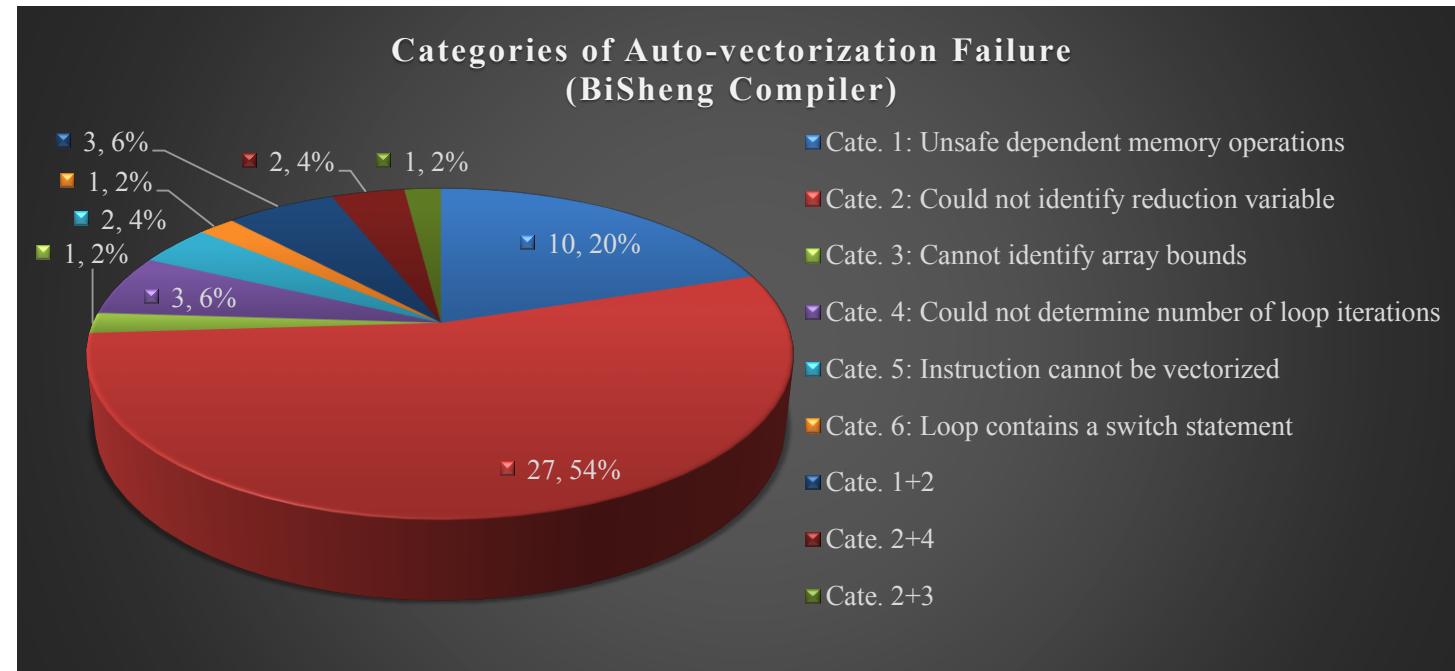
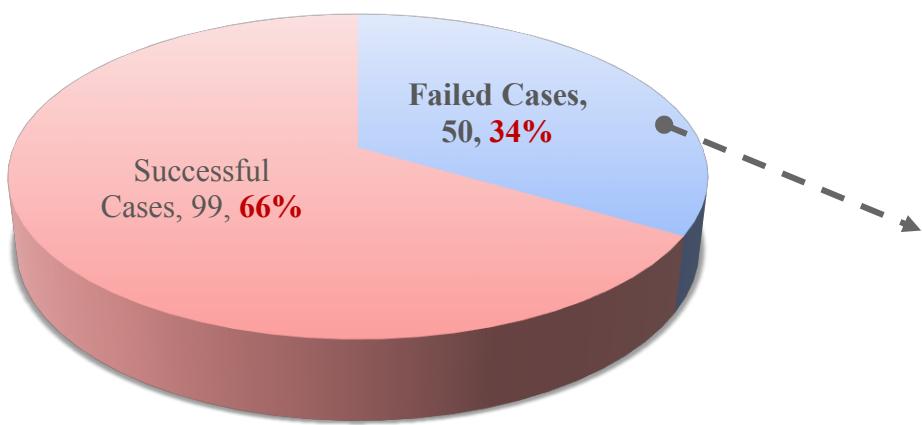
【More Information, please Refer to the Full Preprint Paper】

Zheng Z, Cheng L*, Li L, et al. VecTrans: LLM Transformation Framework for Better Auto-vectorization on High-performance CPU[J]. arXiv preprint arXiv:2503.19449, 2025.

Is It Good Enough for Auto-vectorization in Industrial Compilers?

- (1) Even for the simple benchmark - TSVC-2, there is still about **50/149** cases that cannot be vectorized by the LLVM-based industrial compiler(BiSheng Compiler) with **-O3** flag;
- (2) The dominant failure causes are: **limited analysis for memory dependencies and reduction**;
- (3) For **industrial applications** like HPC and mobile rendering, code scenarios are more complicated, which makes it hard to successfully **analyze the high-level semantics** purely utilizing traditional industrial compiler. **Hence, the answer to the title question is NO.**

Result Summary of Auto-vectorization in TSVC-2
(BiSheng Compiler)

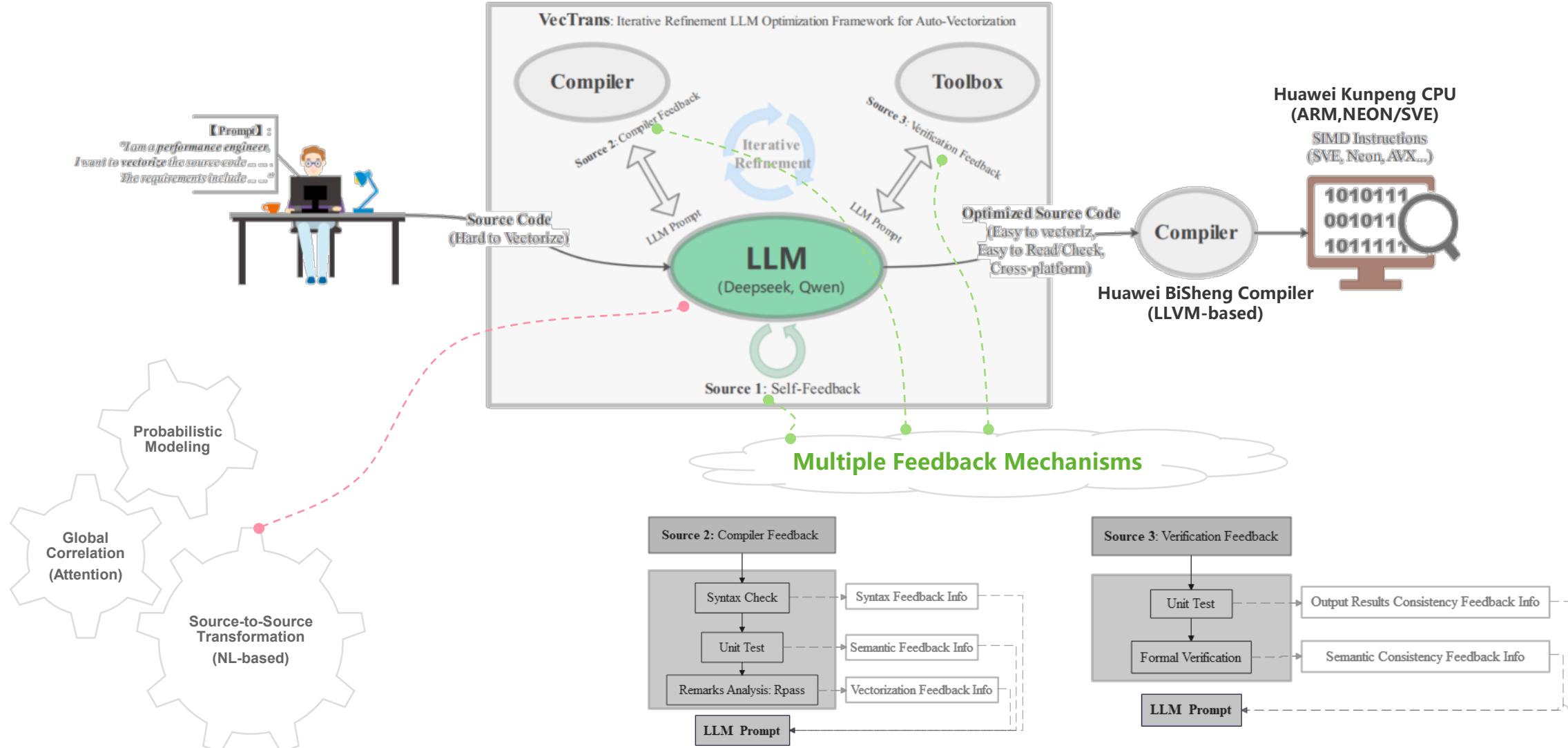


- Compiler: [BiSheng Compiler](#)(LLVM-based, Industrial-level),
- Hardware: Kunpeng CPU-ARM(NEON/SVE)
- Optimization Flags: (1) -O3 -ffast-math; (2) -Rpass=loop-vectorize; (3) -Rpass-analysis=loop-vectorize

VecTrans: A LLM Compiler Agent Framework to Enhance Auto-vectorization

We built **VecTrans** to massively enhance the capability of auto-vectorization in traditional compilers. It has the following features:

- (1) *LLM-guided Source Code Transformations for Vectorization;*
- (2) *Explores LLM-compiler Collaboration Paradigm;*
- (3) *Naively Support Cross-platform Verification;*
- (4) *Generates Effective Results*



Current Results

Benchmark Code(TSVC-2, s1113)

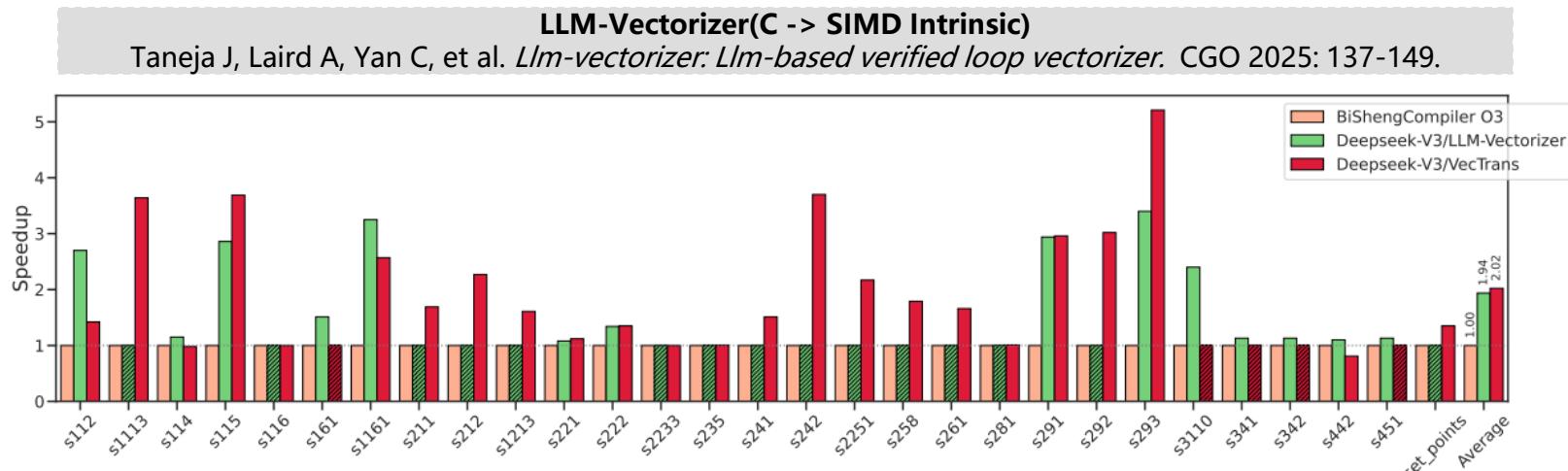
```
void s1113(int iterations, int LEN_1D, float* a,
           float* b)
{
    for (int nl = 0; nl < 2*iterations; nl++) {
        for (int i = 0; i < LEN_1D; i++) {
            a[i] = a[LEN_1D/2] + b[i];
        }
    }
}
```



VecTrans

```
void s1113_opt(int iterations, int LEN_1D, float*
                a, float* b) {
    for (int nl = 0; nl < 2*iterations; nl++) {
        int mid = LEN_1D / 2;
        float temp = a[mid];
        for (int i = 0; i < mid; i++) {
            a[i] = temp + b[i];
        }
        a[mid] = temp + b[mid];
        temp = a[mid];
        for (int i = mid + 1; i < LEN_1D; i++) {
            a[i] = temp + b[i];
        }
    }
}
```

Comparison with SOTA(TSVC-2)



Auto-vectorization Ablation Experiments (BiSheng Compiler+Kunpeng CPU-Neon/SVE)

Configurations	Success Ratio	Average Iteration Times
DeepSeek-V3/ VecTrans	46.2%	8.76
DeepSeek-V3/LLM-Vectorizer	28.8%	13.39
DeepSeek-V3/Base Model	17.3%	5.41
Qwen2.5-32B/VecTrans	34.6%	13.94
Qwen2.5-72B/VecTrans	38.5%	12.26
DeepSeek-V3/Without Formal Verification	32.7%	8.49
DeepSeek-V3/ Without Compiler Feedback	21.2%	5.21
DeepSeek-V3/Without Unit Test	25.0	9.66

Discussion and Future Work

1. If we can open the debug information in LLVM infrastructure to designate more precise program analysis information to LLM, we believe that the concept of ***LLM as a Compiler optimizer*** will become more practical for industrial applications;
2. The current VecTrans work will be open source in openEuler community. (<https://gitee.com/openeuler/llvm-project>)

The screenshot shows the LLVM Compiler Infrastructure website. At the top is the LLVM logo. Below it is a navigation bar with links: LLVM Home, Documentation, User Guides, and Source Level Debugging with LLVM. The main content area is titled "Source Level Debugging with LLVM". Under this title is a bulleted list of topics: Introduction, Philosophy behind LLVM debugging information, Debug information consumers, Debug information and optimizations, and Debugging information format.

The screenshot shows the openEuler website homepage. At the top is the openEuler logo and a link to the English version. Below the logo is a banner with the text "100% Support for Mainstream Computing Architectures" and a list of supported architectures: Arm, x86, RISC-V, SW-64, LoongArch, NPUs, GPUs, DPUs, 100+ devices, and 300+ boards. Below the banner are four icons representing different computing environments: Server, Cloud & Cloud Native, Edge Computing, and Embedded.

openEuler is an open source project incubated and operated by the OpenAtom Foundation.



Developers' Meeting

BERLIN 2025

Why add an IR reader to llvm-debuginfo-analyzer tool

Carlos Alberto Enciso

Reduce the noisiness of comparing the debuginfo in LLVM IR

Reduce the noisiness of comparing the debuginfo in LLVM IR



The screenshot shows two LLVM IR code editors side-by-side. The left editor is titled 'simplify-cfg.ll' and the right is 'slp-vectorizer.ll'. Both editors show nearly identical code, with some lines highlighted in yellow. The bottom status bar indicates the file name, encoding, and line numbers.

```
simplify-cfg.ll
134 = distinct !DILexicalBlock(scope: 126, file: 11,
line: 22, column: 6)
135 = !DILocation(line: 23, column: 26, scope: 134)
136 = !DILocalVariable(name: "x1", scope: 126, file: 11,
line: 20, type: 14)
137 = !DILocation(line: 24, column: 16, scope: 134)
138 = !DILocation(line: 24, column: 26, scope: 134)
139 = !DILocalVariable(name: "x2", scope: 126, file: 11,
line: 20, type: 14)
140 = !DILocation(line: 25, column: 22, scope: 134)
141 = !DILocation(line: 25, column: 17, scope: 134)
142 = !DILocalVariable(name: "w", scope: 126, file: 11,
line: 20, type: 14)
143 = !DILocation(line: 26, column: 14, scope: 126)
144 = !DILocation(line: 26, column: 3, scope: 134)
145 = distinct !(!45, !32, !46, !47)
146 = !DILocation(line: 26, column: 20, scope: 126)
147 = !(!"llvm.loop.mustprogress")
148 = !DILocation(line: 28, column: 20, scope: 126)
149 = !DILocation(line: 28, column: 18, scope: 126)
150 = !DILocation(line: 28, column: 30, scope: 126)
151 = !DILocation(line: 28, column: 28, scope: 126)
152 = !DILocation(line: 28, column: 12, scope: 126)
153 = !DILocation(line: 28, column: 7, scope: 126)
154 = !DILocation(line: 29, column: 16, scope: 126)
155 = !DILocation(line: 29, column: 11, scope: 126)
156 = !DILocation(line: 30, column: 16, scope: 126)
157 = !DILocation(line: 30, column: 3, scope: 126)
158 = !DILocation(line: 30, column: 11, scope: 126)
159 = !DILocation(line: 31, column: 1, scope: 126)

Ln:105 Col:1/52 Ch:1/52 EOL:CRLF          Windows-1252      Win          Line:131-132          Windows-1252      Win
* 135 = !DILocation(line: 23, column: 26, scope: 134)
136 = !DILocalVariable(name: "x1", scope: 126, file: 11, line: 20, type: 14)
137 = !DILocation(line: 24, column: 16, scope: 134)
138 = !DILocation(line: 24, column: 26, scope: 134)
139 = !DILocalVariable(name: "x2", scope: 126, file: 11, line: 20, type: 14)
140 = !DILocation(line: 25, column: 22, scope: 134)
141 = !DILocalVariable(name: "w", scope: 126, file: 11, line: 20, type: 14)
142 = !DILocation(line: 26, column: 14, scope: 126)
143 = !DILocation(line: 26, column: 3, scope: 134)
144 = distinct !(!44, !32, !45, !46)
145 = !DILocation(line: 26, column: 20, scope: 126)
146 = !(!"llvm.loop.mustprogress")
147 = !DILocation(line: 28, column: 20, scope: 126)
148 = !DILocation(line: 28, column: 18, scope: 126)
149 = !DILocation(line: 28, column: 30, scope: 126)
150 = !DILocation(line: 28, column: 28, scope: 126)
151 = !DILocation(line: 28, column: 12, scope: 126)
152 = !DILocation(line: 28, column: 7, scope: 126)
153 = !DILocation(line: 29, column: 16, scope: 126)
154 = !DILocation(line: 29, column: 11, scope: 126)
155 = !DILocation(line: 31, column: 1, scope: 126)
```

IR changes: comparison tool

The screenshot shows a comparison interface between 'simplify-cfg.ll' and 'slp-vectorizer.ll'. It highlights differences in symbols and lines. The bottom section shows the logical view of the code.

```
Reference: 'simplify-cfg.ll'
Target:   'slp-vectorizer.ll'

(2) Missing Symbols:
- 20 {Variable} 'x1' -> 'float'
- 20 {Variable} 'x2' -> 'float'

(5) Missing Lines:
- 24 {Line}
- 30 {Line}
- 30 {Line}
- 30 {Line}
- 24 {Line}

Reference: 'simplify-cfg.ll'
Target:   'slp-vectorizer.ll'

Logical View:
  {File} 'simplify-cfg.ll'

    {CompileUnit} 'test.cpp'
      {Function} extern not_inlined 'RandF32' -> 'float'
      {Variable} 'uRand' -> 'U32'
      {Variable} 'fRand' -> 'F32'
      {Function} extern not_inlined 'randGauss' -> 'void'
      {Parameter} 'work' -> '* float'
      20 {Variable} 'w' -> 'float'
      - 20 {Variable} 'x1' -> 'float'
      - 20 {Variable} 'x2' -> 'float'
```

IR changes: llvm-debuginfo-analyzer

LLVM and debug information

- Different formats, toolchain, tools
- Common problems

Analyzing optimizer IR output

- IR before/after an optimizer pass
- Test case: SLP Vectorizer pass drops debug information

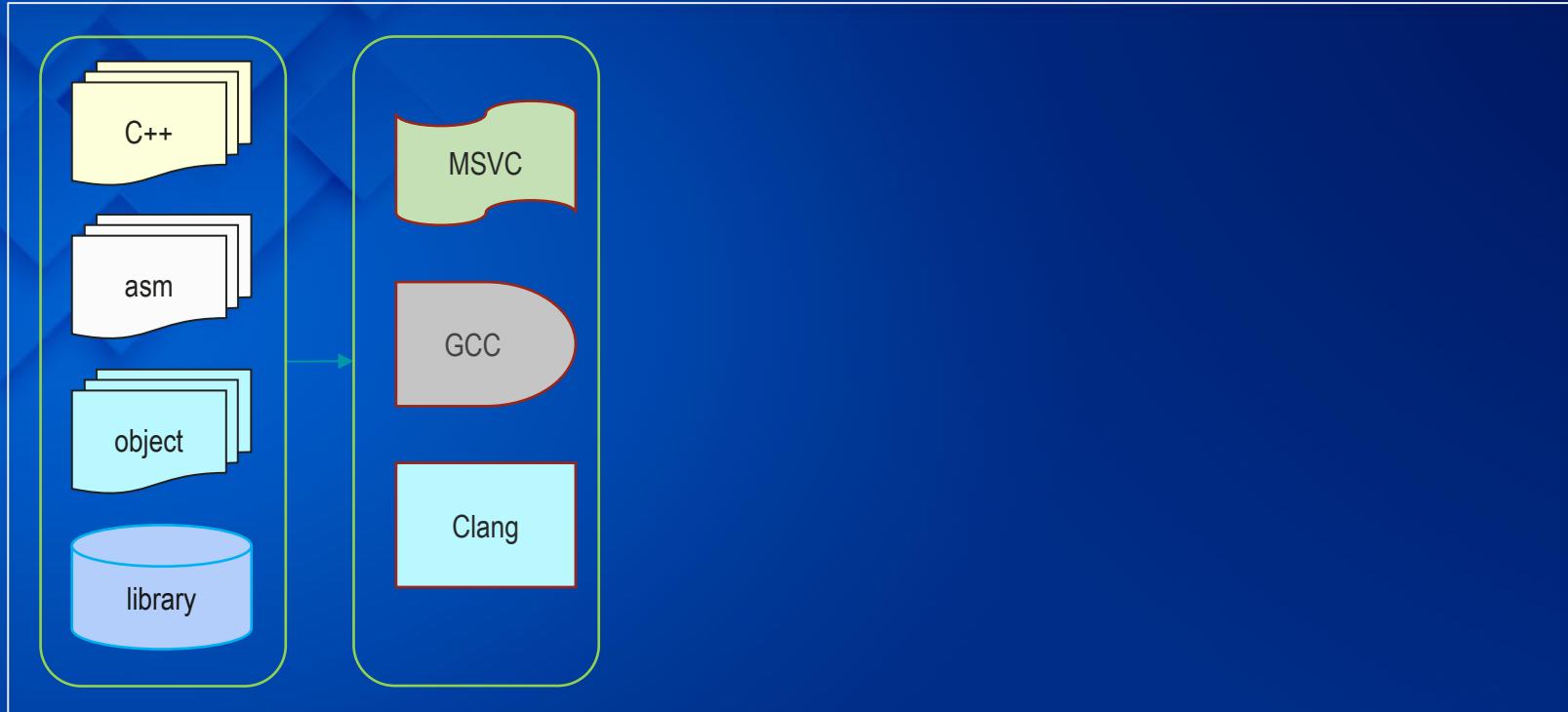
llvm-debuginfo-analyzer

- Basic introduction
- Print logical view
- Compare logical view

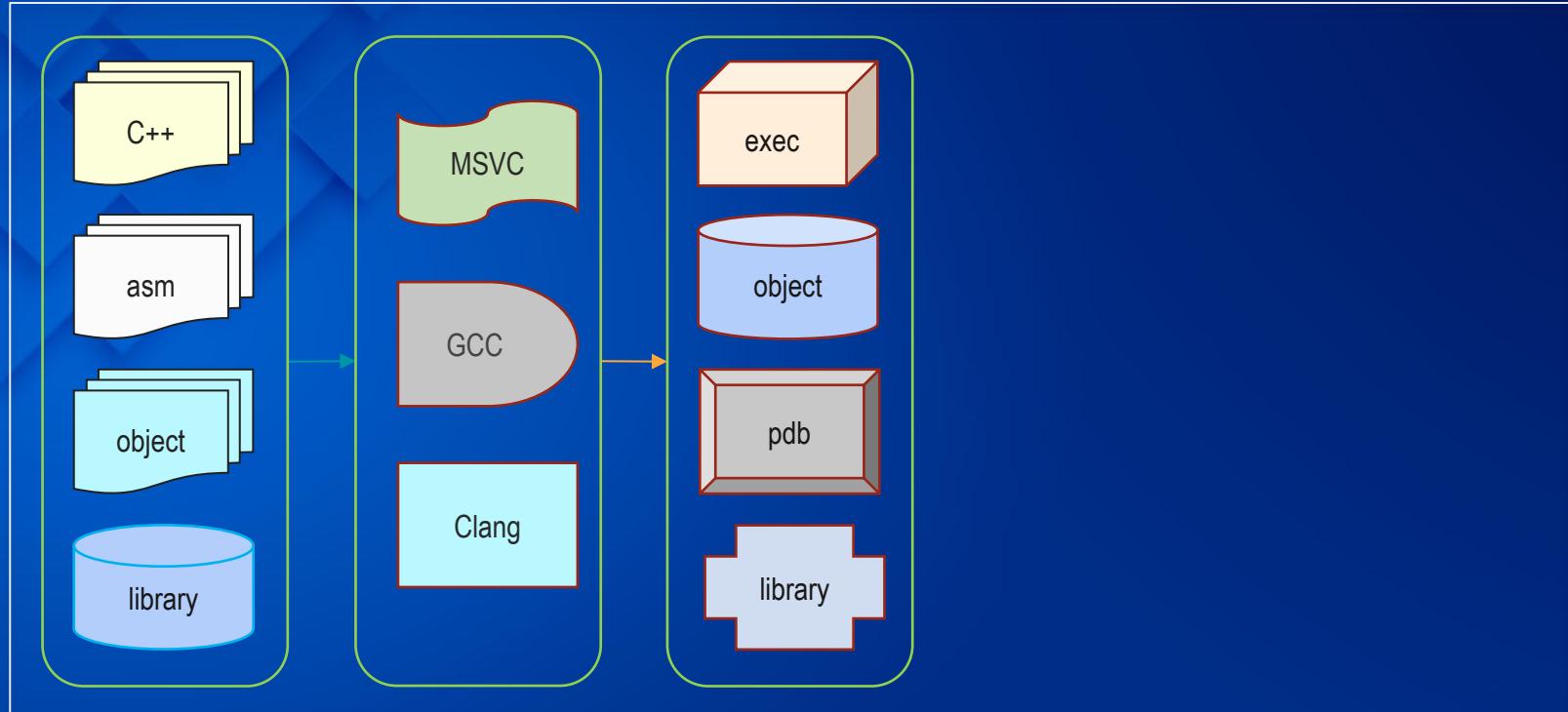
LLVM and debug information - inputs



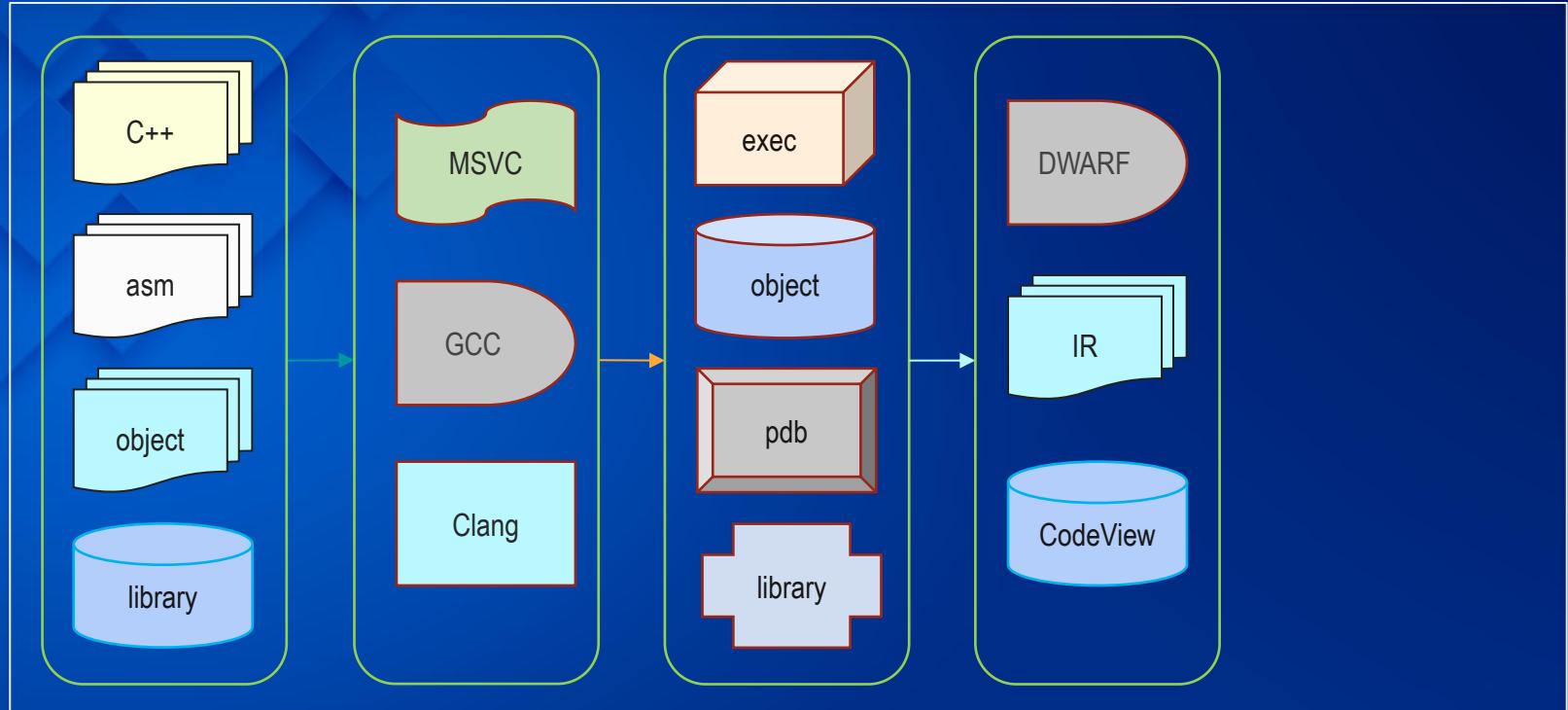
Different inputs



LLVM and debug information - binary file formats

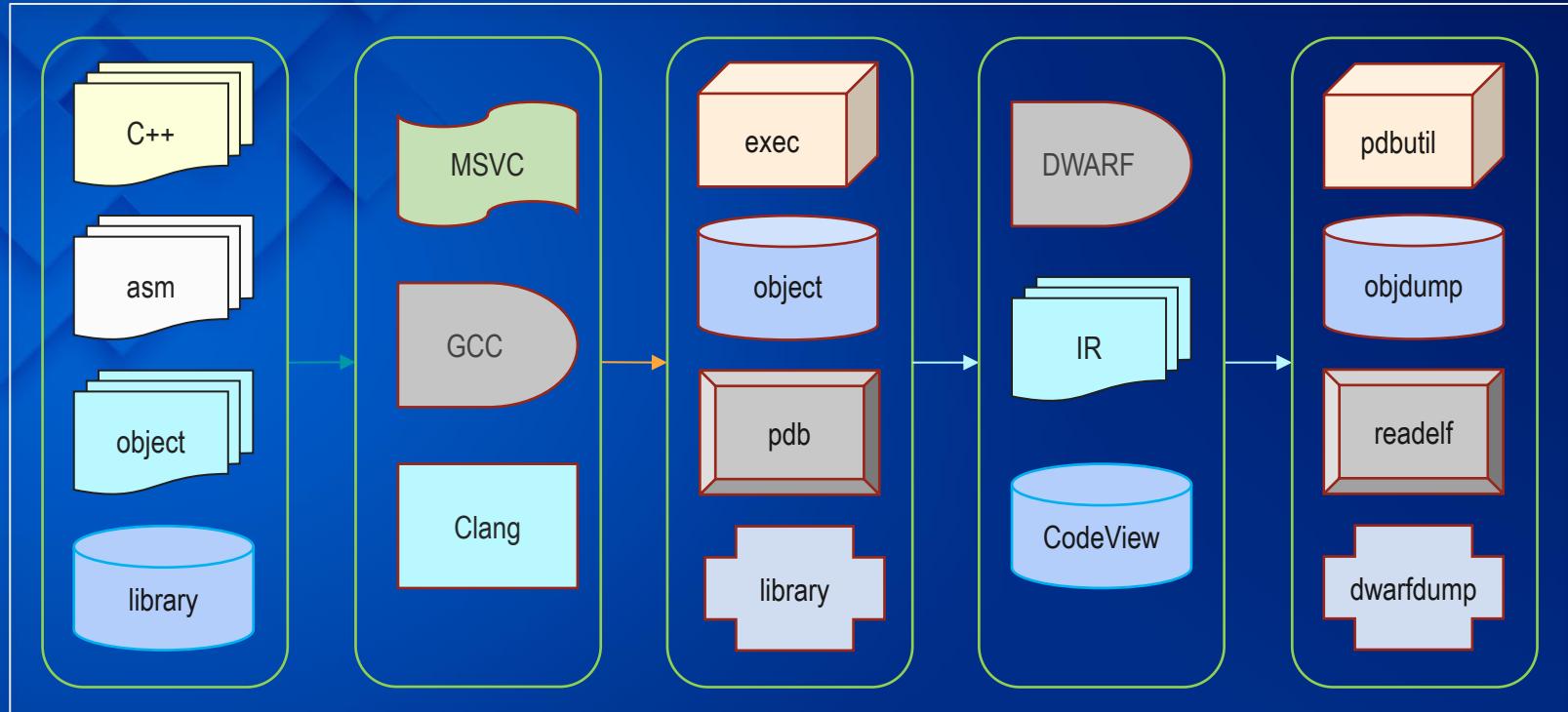


Different binary formats



Different debug information formats

LLVM and debug information - debug information tools



Different debug information tools

Does the debug information represent the original source

- Which variables are dropped due to optimization
- Why I cannot stop at a particular line
- Which lines are associated to a specific code range
- Size changes due to toolchain features

Does the debug information represent the original source

- Which variables are dropped due to optimization
- Why I cannot stop at a particular line
- Which lines are associated to a specific code range
- Size changes due to toolchain features

Semantic differences in the generated debug information

- By different toolchain versions (same platform)
- Same source code compiled in different platforms

IR before/after an optimizer pass

The IR output is very rich and noisy

- The metadata identifiers changes between passes
- Difficult to see the changes using a comparison tool
- Requires knowledge about the IR and passes
- By bisection, we can pin down which pass causes the change

IR before/after an optimizer pass

The IR output is very rich and noisy

- The metadata identifiers changes between passes
- Difficult to see the changes using a comparison tool
- Requires knowledge about the IR and passes
- By bisection, we can pin down which pass causes the change

The optimizer includes a wide set of debug printing options

- `--print-before`, `--print-before-all`, `--print-after`, `--print-after-all`, etc.

IR before/after an optimizer pass

The IR output is very rich and noisy

- The metadata identifiers changes between passes
- Difficult to see the changes using a comparison tool
- Requires knowledge about the IR and passes
- By bisection, we can pin down which pass causes the change

The optimizer includes a wide set of debug printing options

- --print-before, --print-before-all, --print-after, --print-after-all, etc.

Test case: SLP Vectorizer pass drops debug information

- [DebugInfo@O2] <https://github.com/llvm/llvm-project/issues/45507>
- Drops location information for local variables x1 and x2
- Generate IR after Simplify CFG and SLP Vectorizer passes

IR metadata - Simplify CFG and SLP Vectorizer passes



```
1 // Compile with clang -g -O2.
2 // The SLP Vectorizer pass drops location
3 // information for the local variables:
4 // x1 and x2.
5
6 typedef unsigned int U32;
7 typedef float F32;
8 extern "C" double log(float);
9 extern "C" double sqrt(float);
10
11 extern unsigned RandU32();
12
13 float RandF32() {
14     U32 uRand = RandU32();
15     F32 fRand = ((F32)uRand / 4294967810.0f);
16     return (fRand);
17 }
18
19 void randGauss(float work[2]) {
20     float x1, x2, w;
21
22     do {
23         x1 = 2.f * RandF32() - 1.f;
24         x2 = 2.f * RandF32() - 1.f;
25         w = x1 * x1 + x2 * x2;
26     } while (w >= 1.f);
27
28     w = sqrt((-2.f * log(w)) / w);
29     work[0] = x1 * w;
30     work[1] = x2 * w;
31 }
```

IR metadata - Simplify CFG and SLP Vectorizer passes



```
1 // Compile with clang -g -O2.
2 // The SLP Vectorizer pass drops location
3 // information for the local variables:
4 // x1 and x2.
5
6 typedef unsigned int U32;
7 typedef float F32;
8 extern "C" double log(float);
9 extern "C" double sqrt(float);
10
11 extern unsigned RandU32();
12
13 float RandF32() {
14     U32 uRand = RandU32();
15     F32 fRand = ((F32)uRand / 4294967810.0f);
16     return (fRand);
17 }
18
19 void randGauss(float work[2]) {
20     float x1, x2, w;
21
22     do {
23         x1 = 2.f * RandF32() - 1.f;
24         x2 = 2.f * RandF32() - 1.f;
25         w = x1 * x1 + x2 * x2;
26     } while (w >= 1.f);
27
28     w = sqrt((-2.f * log(w)) / w);
29     work[0] = x1 * w;
30     work[1] = x2 * w;
31 }
```

```
!35 = !DILocation(line: 23, column: 26, scope: !34)
!36 = !DILocalVariable(name: "x1", scope: !26, file:
!1, line: 20, type: !4)
!37 = !DILocation(line: 24, column: 16, scope: !34)
!38 = !DILocation(line: 24, column: 26, scope: !34)
!39 = !DILocalVariable(name: "x2", scope: !26, file:
!1, line: 20, type: !4)
!40 = !DILocation(line: 25, column: 22, scope: !34)
!41 = !DILocation(line: 25, column: 17, scope: !34)
!42 = !DILocalVariable(name: "w", scope: !26, file:
!1, line: 20, type: !4)
!43 = !DILocation(line: 26, column: 14, scope: !26)
!44 = !DILocation(line: 26, column: 3, scope: !34)
!45 = distinct !{!45, !32, !46, !47}
!46 = !DILocation(line: 26, column: 20, scope: !26)
!47 = !{"!llvm.loop.mustprogress"}
!48 = !DILocation(line: 28, column: 20, scope: !26)
!49 = !DILocation(line: 28, column: 18, scope: !26)
!50 = !DILocation(line: 28, column: 30, scope: !26)
!51 = !DILocation(line: 28, column: 28, scope: !26)
!52 = !DILocation(line: 28, column: 12, scope: !26)
!53 = !DILocation(line: 28, column: 7, scope: !26)
!54 = !DILocation(line: 29, column: 16, scope: !26)
!55 = !DILocation(line: 29, column: 11, scope: !26)
!56 = !DILocation(line: 30, column: 16, scope: !26)
!57 = !DILocation(line: 30, column: 3, scope: !26)
!58 = !DILocation(line: 30, column: 11, scope: !26)
!59 = !DILocation(line: 31, column: 1, scope: !26)
```

IR metadata after Simplify CFG

IR metadata - Simplify CFG and SLP Vectorizer passes

```
1 // Compile with clang -g -O2.
2 // The SLP Vectorizer pass drops location
3 // information for the local variables:
4 // x1 and x2.
5
6 typedef unsigned int U32;
7 typedef float F32;
8 extern "C" double log(float);
9 extern "C" double sqrt(float);
10
11 extern unsigned RandU32();
12
13 float RandF32() {
14     U32 uRand = RandU32();
15     F32 fRand = ((F32)uRand / 4294967810.0f);
16     return (fRand);
17 }
18
19 void randGauss(float work[2]) {
20     float x1, x2, w;
21
22     do {
23         x1 = 2.f * RandF32() - 1.f;
24         x2 = 2.f * RandF32() - 1.f;
25         w = x1 * x1 + x2 * x2;
26     } while (w >= 1.f);
27
28     w = sqrt((-2.f * log(w)) / w);
29     work[0] = x1 * w;
30     work[1] = x2 * w;
31 }
```

```
!35 = !DILocation(line: 23, column: 26, scope: !34)
!36 = !DILocalVariable(name: "x1", scope: !26, file:
!1, line: 20, type: !4)
!37 = !DILocation(line: 24, column: 16, scope: !34)
!38 = !DILocation(line: 24, column: 26, scope: !34)
!39 = !DILocalVariable(name: "x2", scope: !26, file:
!1, line: 20, type: !4)
!40 = !DILocation(line: 25, column: 22, scope: !34)
!41 = !DILocation(line: 25, column: 17, scope: !34)
!42 = !DILocalVariable(name: "w", scope: !26, file:
!1, line: 20, type: !4)
!43 = !DILocation(line: 26, column: 14, scope: !26)
!44 = !DILocation(line: 26, column: 3, scope: !34)
!45 = distinct !(!45, !32, !46, !47)
!46 = !DILocation(line: 26, column: 20, scope: !26)
!47 = !{"!llvm.loop.mustprogress"}
!48 = !DILocation(line: 28, column: 20, scope: !26)
!49 = !DILocation(line: 28, column: 18, scope: !26)
!50 = !DILocation(line: 28, column: 30, scope: !26)
!51 = !DILocation(line: 28, column: 28, scope: !26)
!52 = !DILocation(line: 28, column: 12, scope: !26)
!53 = !DILocation(line: 28, column: 7, scope: !26)
!54 = !DILocation(line: 29, column: 16, scope: !26)
!55 = !DILocation(line: 29, column: 11, scope: !26)
!56 = !DILocation(line: 30, column: 16, scope: !26)
!57 = !DILocation(line: 30, column: 3, scope: !26)
!58 = !DILocation(line: 30, column: 11, scope: !26)
!59 = !DILocation(line: 31, column: 1, scope: !26)
```

IR metadata after Simplify CFG

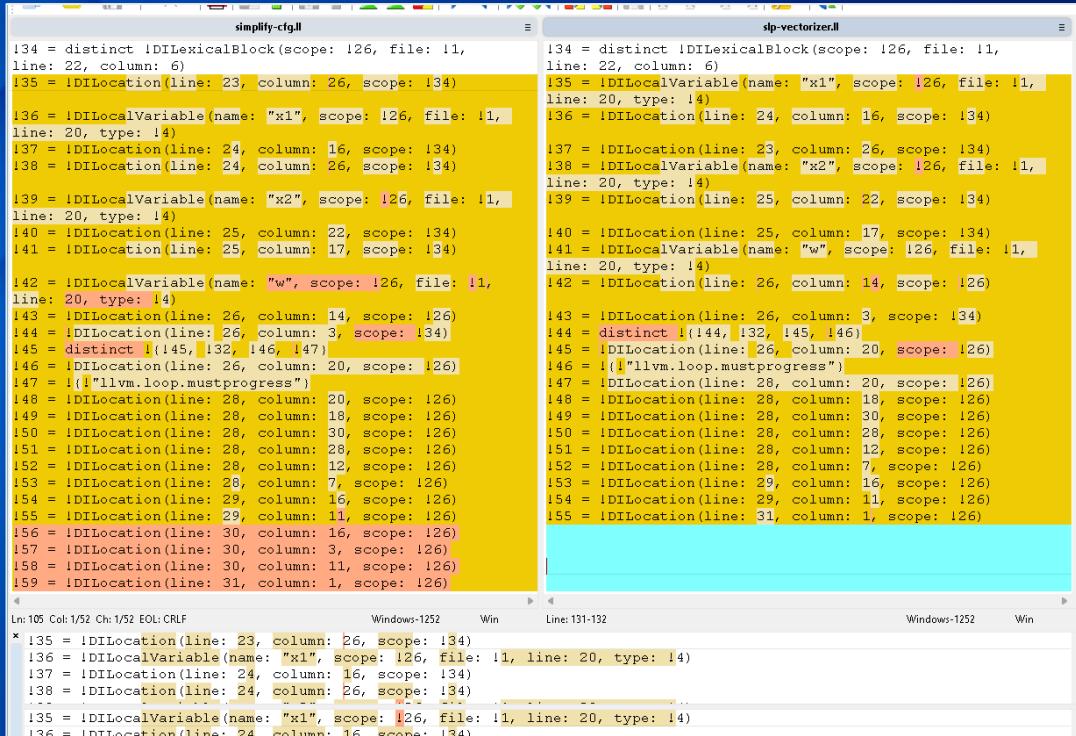


Sony
Interactive
Entertainment



```
!35 = !DILocalVariable(name: "x1", scope: !26, file:
!1, line: 20, type: !4)
!36 = !DILocation(line: 24, column: 16, scope: !34)
!37 = !DILocation(line: 23, column: 26, scope: !34)
!38 = !DILocalVariable(name: "x2", scope: !26, file:
!1, line: 20, type: !4)
!39 = !DILocation(line: 25, column: 22, scope: !34)
!40 = !DILocation(line: 25, column: 17, scope: !34)
!41 = !DILocalVariable(name: "w", scope: !26, file:
!1, line: 20, type: !4)
!42 = !DILocation(line: 26, column: 14, scope: !26)
!43 = !DILocation(line: 26, column: 3, scope: !34)
!44 = distinct !(!44, !32, !45, !46)
!45 = !DILocation(line: 26, column: 20, scope: !26)
!46 = !{"!llvm.loop.mustprogress"}
!47 = !DILocation(line: 28, column: 20, scope: !26)
!48 = !DILocation(line: 28, column: 18, scope: !26)
!49 = !DILocation(line: 28, column: 30, scope: !26)
!50 = !DILocation(line: 28, column: 28, scope: !26)
!51 = !DILocation(line: 28, column: 12, scope: !26)
!52 = !DILocation(line: 28, column: 7, scope: !26)
!53 = !DILocation(line: 29, column: 16, scope: !26)
!54 = !DILocation(line: 29, column: 11, scope: !26)
!55 = !DILocation(line: 31, column: 1, scope: !26)
```

IR metadata after SLP Vectorizer

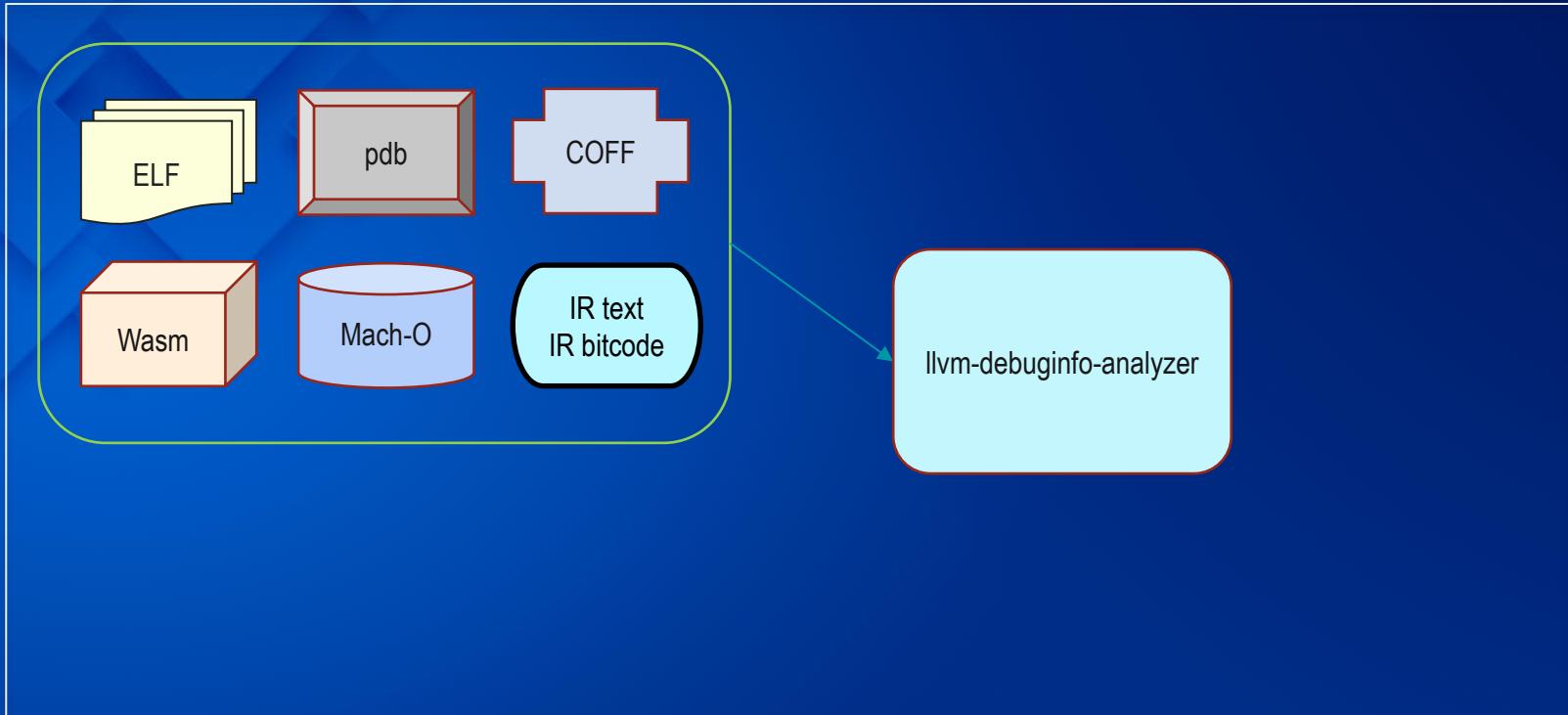


```
simplify-cfg.ll          slp-vectorizer.ll
134 = distinct !DILexicalBlock(scope: 126, file: !1,
line: 22, column: 6)
135 = !DILocation(line: 23, column: 26, scope: 134)
136 = !DILocalVariable(name: "x1", scope: 126, file: !1,
line: 20, type: !4)
137 = !DILocation(line: 24, column: 16, scope: 134)
138 = !DILocation(line: 24, column: 26, scope: 134)
139 = !DILocalVariable(name: "x2", scope: 126, file: !1,
line: 20, type: !4)
140 = !DILocation(line: 25, column: 22, scope: 134)
141 = !DILocation(line: 25, column: 17, scope: 134)
142 = !DILocalVariable(name: "w", scope: 126, file: !1,
line: 20, type: !4)
143 = !DILocation(line: 26, column: 14, scope: 126)
144 = !DILocation(line: 26, column: 3, scope: 134)
145 = distinct !(145, 132, 146, 147)
146 = !DILocation(line: 26, column: 20, scope: 126)
147 = !(1!"llvm.loop.mustprogress")
148 = !DILocation(line: 28, column: 20, scope: 126)
149 = !DILocation(line: 28, column: 18, scope: 126)
150 = !DILocation(line: 28, column: 30, scope: 126)
151 = !DILocation(line: 28, column: 28, scope: 126)
152 = !DILocation(line: 28, column: 12, scope: 126)
153 = !DILocation(line: 28, column: 7, scope: 126)
154 = !DILocation(line: 29, column: 16, scope: 126)
155 = !DILocation(line: 29, column: 11, scope: 126)
156 = !DILocation(line: 30, column: 16, scope: 126)
157 = !DILocation(line: 30, column: 3, scope: 126)
158 = !DILocation(line: 30, column: 11, scope: 126)
159 = !DILocation(line: 31, column: 1, scope: 126)

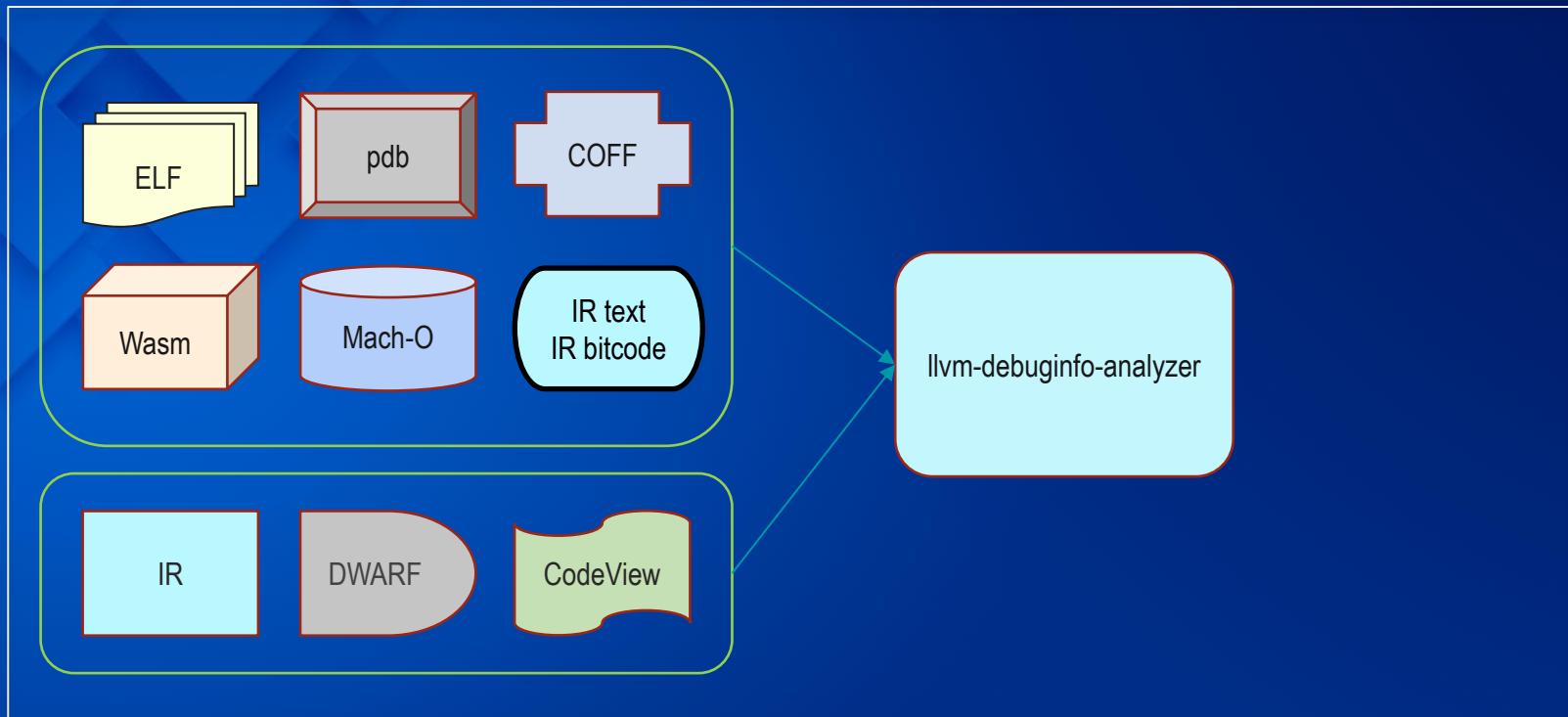
Ln:105 Col:1/52 Ch:1/52 EOL:CRLF          Windows-1252   Win      Line:131-132
* 135 = !DILocation(line: 23, column: 26, scope: 134)
136 = !DILocalVariable(name: "x1", scope: 126, file: !1, line: 20, type: !4)
137 = !DILocation(line: 24, column: 16, scope: 134)
138 = !DILocation(line: 24, column: 26, scope: 134)
139 = !DILocalVariable(name: "x1", scope: 126, file: !1, line: 20, type: !4)
140 = !DILocation(line: 25, column: 22, scope: 134)
141 = !DILocalVariable(name: "w", scope: 126, file: !1,
line: 20, type: !4)
142 = !DILocation(line: 26, column: 14, scope: 126)
143 = !DILocation(line: 26, column: 3, scope: 134)
144 = distinct !(144, 132, 145, 146)
145 = !DILocation(line: 26, column: 20, scope: 126)
146 = !(1!"llvm.loop.mustprogress")
147 = !DILocation(line: 28, column: 20, scope: 126)
148 = !DILocation(line: 28, column: 18, scope: 126)
149 = !DILocation(line: 28, column: 30, scope: 126)
150 = !DILocation(line: 28, column: 28, scope: 126)
151 = !DILocation(line: 28, column: 12, scope: 126)
152 = !DILocation(line: 28, column: 7, scope: 126)
153 = !DILocation(line: 29, column: 16, scope: 126)
154 = !DILocation(line: 29, column: 11, scope: 126)
155 = !DILocation(line: 31, column: 1, scope: 126)

Ln:105 Col:1/52 Ch:1/52 EOL:CRLF          Windows-1252   Win      Line:131-132
* 135 = !DILocation(line: 23, column: 26, scope: 134)
136 = !DILocalVariable(name: "x1", scope: 126, file: !1, line: 20, type: !4)
137 = !DILocation(line: 24, column: 16, scope: 134)
138 = !DILocation(line: 24, column: 26, scope: 134)
139 = !DILocalVariable(name: "x1", scope: 126, file: !1, line: 20, type: !4)
140 = !DILocation(line: 25, column: 22, scope: 134)
```

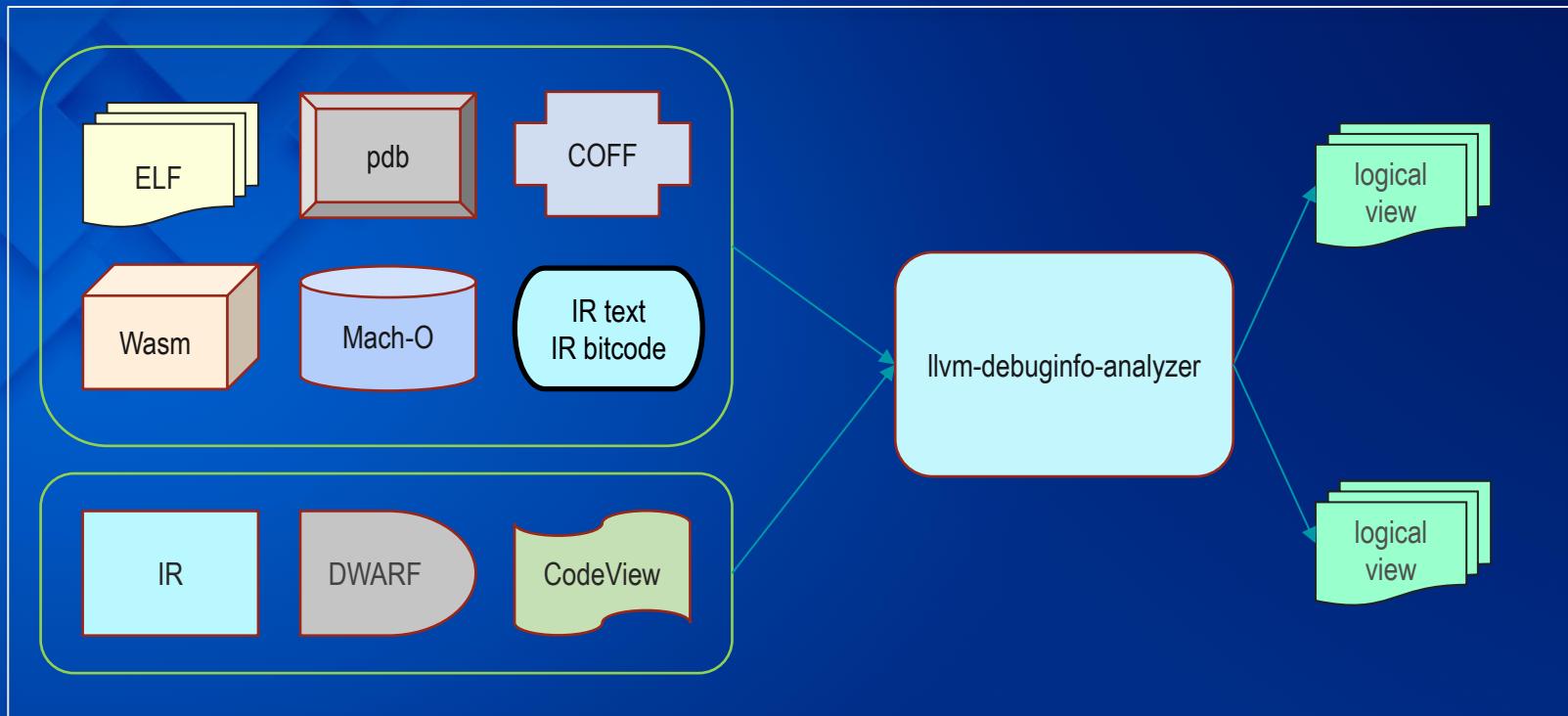
IR changes: comparison tool



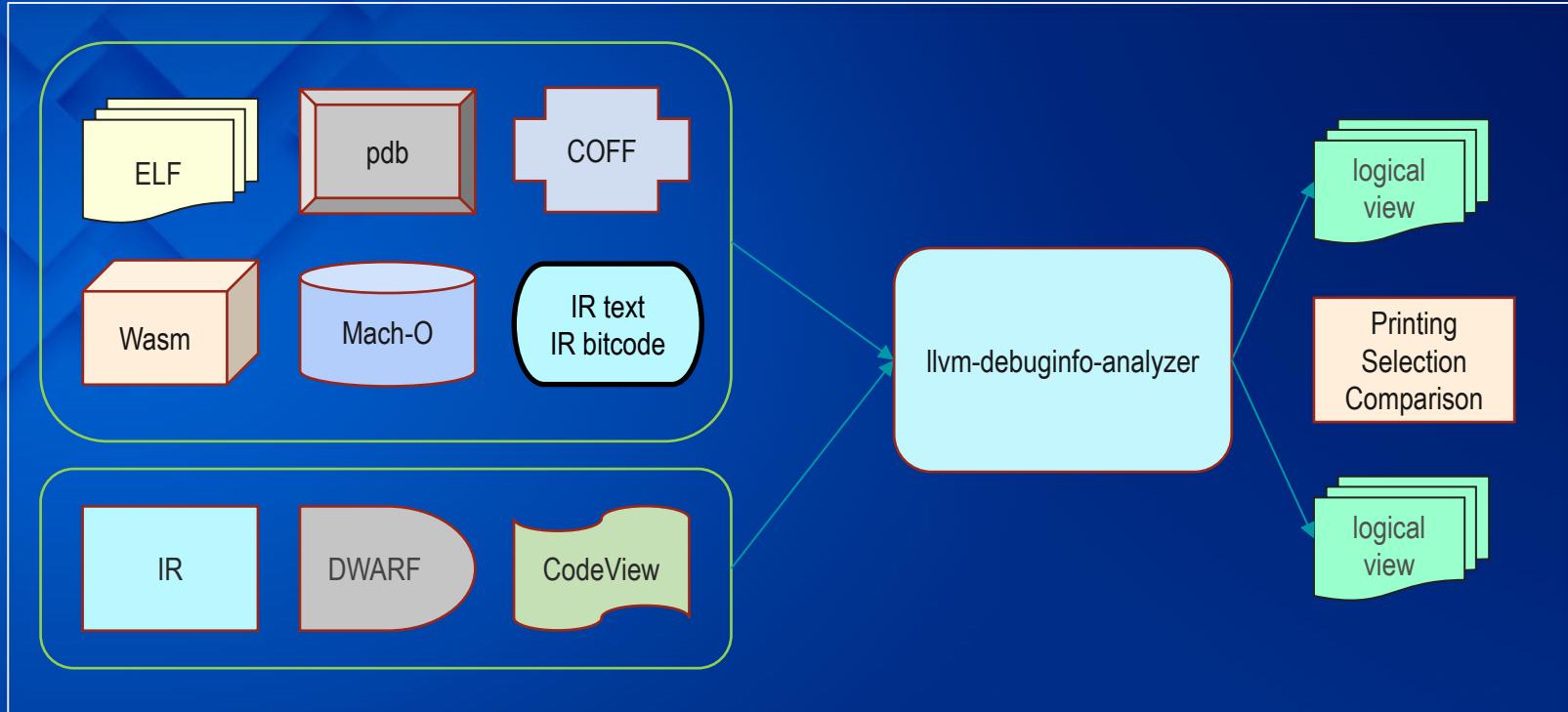
Supported binary file formats



Supported debug information formats



Logical view is a high-level representation of the debug information



Logical views can be printed, selected and compared

DWARF debug information (llvm-dwarfdump)



```
DW_TAG_compile_unit
  DW_AT_producer  ("clang")
  DW_AT_language   (DW_LANG_C_plus_plus_14)
  DW_AT_name      ("hello-world.cpp")
  DW_AT_low_pc    (0x0000000000000000)
  DW_AT_high_pc   (0x0000000000000014)
  DW_AT_addr_base (0x00000008)

DW_TAG_variable
  DW_AT_type      (0x0000002d "const char[13]")
  DW_AT_decl_line (4)
  DW_AT_location   (DW_OP_addrx 0x0)

DW_TAG_array_type
  DW_AT_type      (0x00000039 "const char")

DW_TAG_subrange_type
  DW_AT_type      (0x00000042 "__ARRAY_SIZE_TYPE__")
  DW_AT_count     (0x0d)

DW_TAG_subprogram
  DW_AT_low_pc    (0x0000000000000000)
  DW_AT_high_pc   (0x0000000000000014)
  DW_AT_frame_base (DW_OP_reg6 RBP)
  DW_AT_linkage_name ("_Z3foov")
  DW_AT_name      ("foo")
  DW_AT_decl_line  (3)
  DW_AT_external   (true)
```

DWARF debug information

CodeView debug information (llvm-pdbutil)

```
DW_TAG_compile_unit
  DW_AT_producer  ("clang")
  DW_AT_language   (DW_LANG_C_plus_plus_14)
  DW_AT_name      ("hello-world.cpp")
  DW_AT_low_pc    (0x0000000000000000)
  DW_AT_high_pc   (0x0000000000000014)
  DW_AT_addr_base (0x00000008)

DW_TAG_variable
  DW_AT_type      (0x0000002d "const char[13]")
  DW_AT_decl_line (4)
  DW_AT_location   (DW_OP_addrx 0x0)

DW_TAG_array_type
  DW_AT_type      (0x00000039 "const char")

DW_TAG_subrange_type
  DW_AT_type      (0x00000042 "__ARRAY_SIZE_TYPE__")
  DW_AT_count     (0xd)

DW_TAG_subprogram
  DW_AT_low_pc    (0x0000000000000000)
  DW_AT_high_pc   (0x0000000000000014)
  DW_AT_frame_base (DW_OP_reg6 RBP)
  DW_AT_linkage_name ("Z3fooV")
  DW_AT_name      ("foo")
  DW_AT_decl_line (3)
  DW_AT_external   (true)
```

DWARF debug information

```
Types (.debug$T)
=====
0x1000 | LF_ARGLIST [size = 8]
0x1001 | LF PROCEDURE [size = 16]
  return type = 0x0003 (void), # args = 0,
  param list = 0x1000
  calling conv = cdecl, options = None
0x1002 | LF FUNC_ID [size = 16]
  name = foo, type = 0x1001, parent scope =
<no type>
0x1004 | LF STRING_ID [size = 24] ID: <no type>,
String: hello-world.cpp

Symbols
=====
Mod 0000 | ` .debug$S`:
  0 | S_OBJCNAME [size = 64] sig=0, `hello-
world-clang-cv.o`
  0 | S_COMPILE3 [size = 156]
    machine = intel x86-x64, Ver = clang
version 21.0.0, language = c++
  frontend = 21.0.0 flags = none
  0 | S_GPROC32_ID [size = 44] `foo`
    parent = 0, end = 0, addr = 0000:0000
    type = `0x1002 (foo)`, debug start = 0,
debug end = 0, flags = noinline | opt debuginfo
  0 | S_FRAMEPROC [size = 32]
    size = 40, padding size = 0 padding = 0
    bytes of callee saved registers = 0,
exception handler addr = 0000:0000
  local fp reg = RSP, param fp reg = RSP
  flags = safe buffers
  0 | S_PROC_ID_END [size = 4]
  0 | S_BUILDINFO [size = 8] BuildId = `0x1008`
```

CodeView debug information

DWARF & CodeView canonical logical view

```
DW_TAG_compile_unit
DW_AT_producer ("clang")
DW_AT_language (DW_LANG_C_plus_plus_14)
DW_AT_name ("hello-world.cpp")
DW_AT_low_pc (0x0000000000000000)
DW_AT_high_pc (0x0000000000000014)
DW_AT_addr_base (0x00000008)

DW_TAG_variable
DW_AT_type (0x0000002d "const char[13]")
DW_AT_decl_line (4)
DW_AT_location (DW_OP_addrx 0x0)

DW_TAG_array_type
DW_AT_type (0x00000039 "const char")

DW_TAG_subrange_type
DW_AT_type (0x00000042 "__ARRAY_SIZE_TYPE__")
DW_AT_count (0xd)

DW_TAG_subprogram
DW_AT_low_pc (0x0000000000000000)
DW_AT_high_pc (0x0000000000000014)
DW_AT_frame_base (DW_OP_reg6 RBP)
DW_AT_linkage_name ("_Z3fooV")
DW_AT_name ("foo")
DW_AT_decl_line (3)
DW_AT_external (true)
```

DWARF debug information

```
Logical View:
{File} 'hello-world-clang.o'

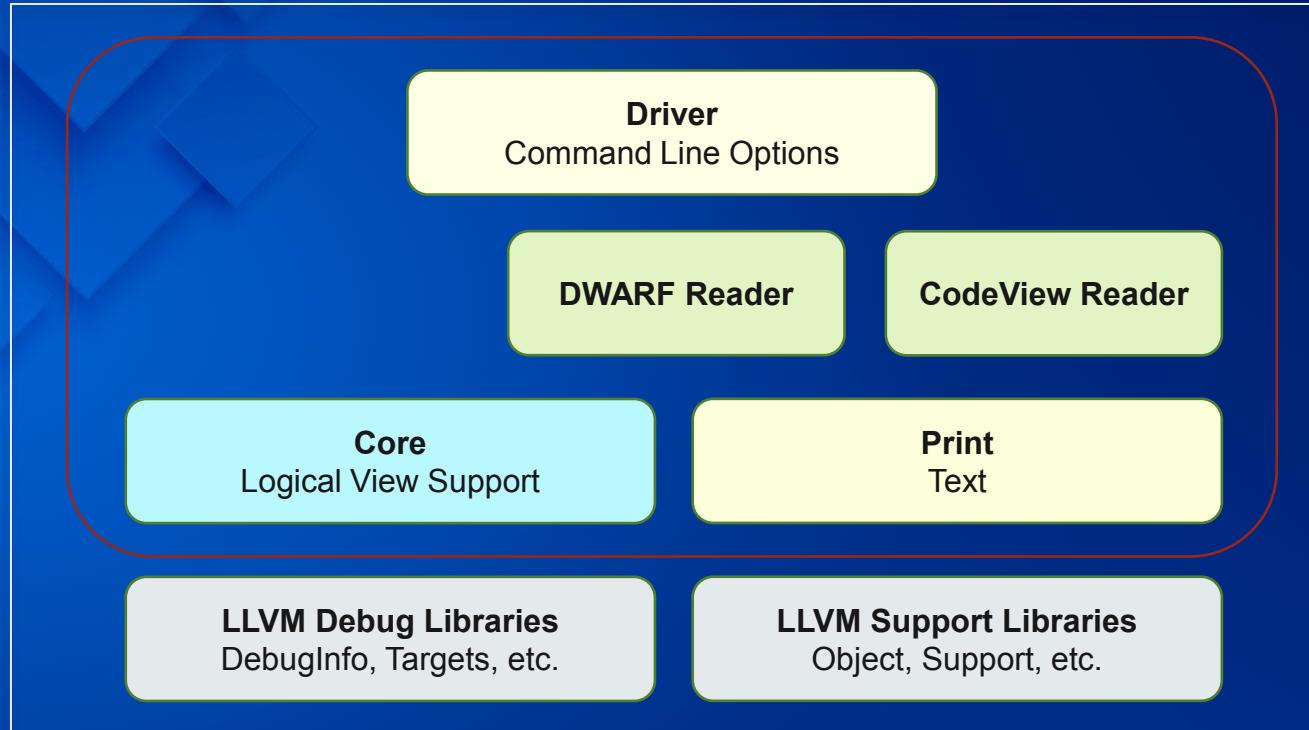
{CompileUnit} 'hello-world.cpp'
{Array} 'const char [13]'
3 {Function} not_inlined 'foo' -> 'void'
3 {Line}
{Code} 'pushq' %rbp
{Code} 'movq' %rsp, %rbp'
4 {Line}
{Code} 'leaq' (%rip), %rdi
{Code} 'movb' $0x0, %al
{Code} 'callq' 0x0
5 {Line}
{Code} 'popq' %rbp'
{Code} 'retq'
5 {Line}
4 {Variable} '' -> 'const char [13]'
```

Canonical logical view

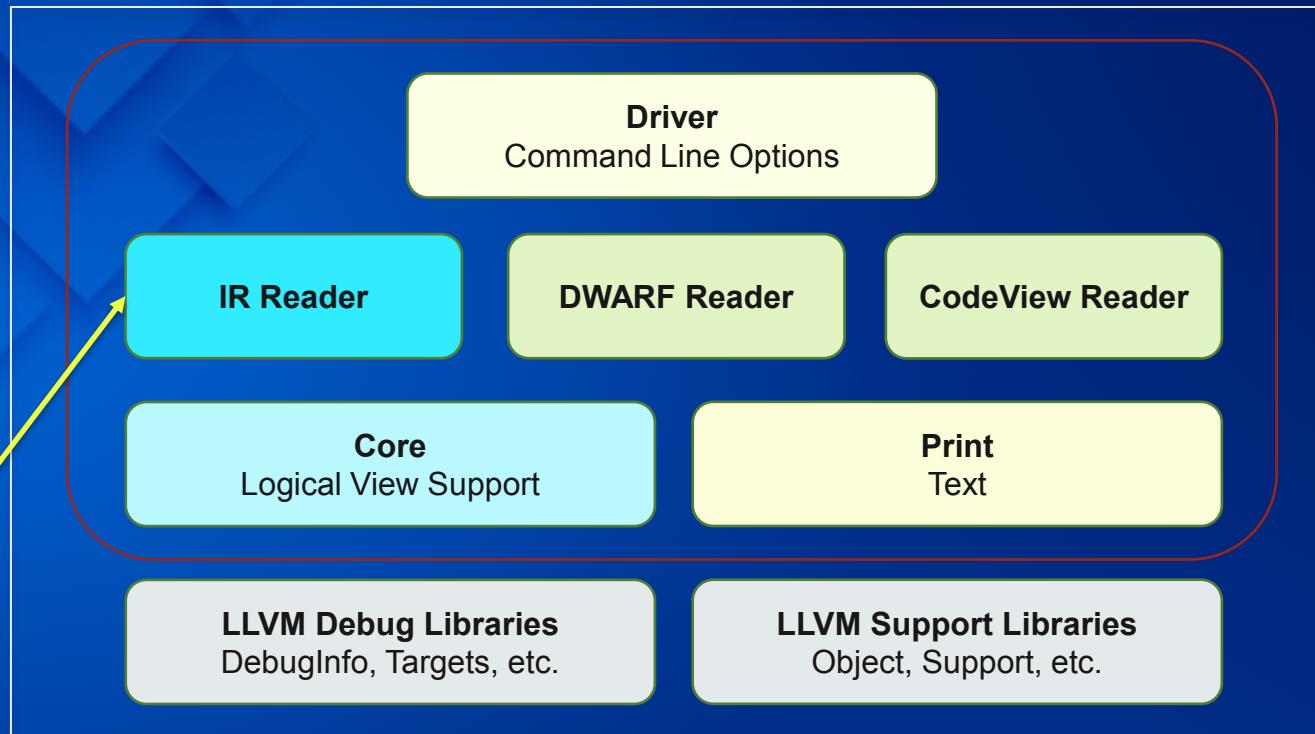
```
Types (.debug$T)
=====
0x1000 | LF_ARGLIST [size = 8]
0x1001 | LF PROCEDURE [size = 16]
    return type = 0x0003 (void), # args = 0,
param list = 0x1000
    calling conv = cdecl, options = None
0x1002 | LF FUNC_ID [size = 16]
    name = foo, type = 0x1001, parent scope =
<no type>
0x1004 | LF STRING_ID [size = 24] ID: <no type>,
String: hello-world.cpp

Symbols
=====
Mod 0000 | ` .debug$S` :
0 | S_OBJCNAME [size = 64] sig=0, `hello-
world-clang-cv.o`
0 | S_COMPILE3 [size = 156]
    machine = intel x86-x64, Ver = clang
version 21.0.0, language = c++
    frontend = 21.0.0.0 flags = none
0 | S_GPROC32_ID [size = 44] `foo`
    parent = 0, end = 0, addr = 0000:0000
    type = `0x1002 (foo)`, debug start = 0,
debug end = 0, flags = noinline | opt debuginfo
0 | S_FRAMEPROC [size = 32]
    size = 40, padding size = 0 padding = 0
    bytes of callee saved registers = 0,
exception handler addr = 0000:0000
    local fp reg = RSP, param fp reg = RSP
    flags = safe buffers
0 | S_PROC_ID_END [size = 4]
0 | S_BUILDINFO [size = 8] BuildId = `0x1008`
```

CodeView debug information



Current components



IR Reader component

Common options to print logical views when dealing with IR

- --attribute=level
- --print=scopes,types,symbols,lines

Common options to print logical views when dealing with IR

- --attribute=level
- --print=scopes,types,symbols,lines

IR tests

- After Simplify CFG pass: simplify-cfg.ll
- After SLP Vectorizer pass: slp-vectorizer.ll

Common options to print logical views when dealing with IR

- `--attribute=level`
- `--print=scopes,types,symbols,lines`

IR tests

- After Simplify CFG pass: `simplify-cfg.ll`
- After SLP Vectorizer pass: `slp-vectorizer.ll`

`llvm-debuginfo-analyzer` command line

- `--attribute=level --print=scopes,symbols,types simplify-cfg.ll`
- `--attribute=level --print=scopes,symbols,types slp-vectorizer.ll`

Logical views for Simplify CFG and SLP Vectorizer passes



```
!llvm.debug = !(0)
!llvm.module.flags = [{!5, !6, !7, !8, !9, !10, !11}
!llvm.debug.info = 0
!D = !DICompileUnit(language: DW_LANG_C_plus_plus, !4, file: !1, producer: "clang", isOptimized: false, runtimeVersion: 0,
    emissionKind: FullDebug, retainedTypes: !2, splitDebuggingInfo: false, nameTableKind: None)
!1 = !DIFile(filename: "test.cpp", directory: "")
!2 = !DIT3
!3 = !DILabeledType(DW_TAG_typedef, name: "T32", file: !1, line: 2, baseType: !4)
!4 = !DIBasicType(name: "kint", size: 32, encoding: DW_ATE_beat)
!5 = !DIT2, !DwarfVersion, !32, !5
!6 = !DIT2, !DebugInfoVersion, !32, !3
!12 = !DIT3("clang")
!13 = distinct !DISubprogram(name: "RandF32", linkageName: ".ZRandf32v", scope: !1, file: !1, line: 13, type: !14, scopeLine: 13, flags: !15)
!14 = !DISPFDefn, spfFlags: DISPFFlagDefinition, unit: !10, retainedNodes: !16
!14 = !DISubroutineType(types: !15)
!15 = !4
!16 = !4
!17 = !DILocation(line: 14, column: 15, scope: !13)
!18 = !DILocalVariable(name: "iRand", scope: !13, file: !1, line: 14, type: !19)
!19 = !DIBasicType(DW_TAG_typedef, name: "U32", file: !1, line: 6, baseType: !20)
!20 = !DIBasicType(name: "unsigned int", size: 32, encoding: DW_ATE_unsigned)
!21 = !DILocation(line: 0, scope: !13)
!22 = !DILocation(line: 21, column: 21, scope: !13)
!23 = !DILocation(line: 15, column: 27, scope: !13)
!24 = !DILocalVariable(name: "Rand", scope: !13, file: !1, line: 15, type: !3)
!25 = !DILocation(line: 16, column: 3, scope: !13)
!26 = distinct !DISubprogram(name: "randGauss", linkageName: ".ZrandGaussPT", scope: !1, file: !1, line: 19, type: !27, scopeLine: 19,
    flags: !18)
!27 = !DISubroutineType(types: !28)
!28 = !4
!29 = !DILocation(DW_TAG_pointer_type, baseType: !4, size: 64)
!30 = !DILocalVariable(name: "work", file: !1, scope: !26, file: !1, line: 19, type: !29)
!31 = !DILocation(line: 0, scope: !29)
!32 = !DILocation(line: 22, column: 3, scope: !30)
!33 = !DILocation(line: 23, column: 16, scope: !34)
!34 = distinct !DILexicalBlock(scope: !25, file: !1, line: 22, column: 6)
!35 = !DILocation(line: 23, column: 26, scope: !34)
!36 = !DILocalVariable(name: "x1", scope: !26, file: !1, line: 20, type: !4)
!37 = !DILocation(line: 24, column: 16, scope: !34)
!38 = !DILocation(line: 24, column: 26, scope: !34)
!39 = !DILocalVariable(name: "x2", scope: !26, file: !1, line: 20, type: !4)
!40 = !DILocation(line: 25, column: 22, scope: !34)
!41 = !DILocation(line: 25, column: 23, scope: !34)
!42 = !DILocation(line: 26, column: 4, scope: !26, file: !1, line: 20, type: !4)
!43 = !DILocation(line: 26, column: 14, scope: !26)
!44 = !DILocation(line: 26, column: 3, scope: !34)
!45 = distinct !44, !32, !46, !47)
!46 = !DILocation(line: 26, column: 20, scope: !26)
!47 = !DILocation(line: 26, column: 20, scope: !26)
!48 = !DILocation(line: 28, column: 20, scope: !26)
!49 = !DILocation(line: 28, column: 18, scope: !26)
!50 = !DILocation(line: 28, column: 30, scope: !26)
!51 = !DILocation(line: 28, column: 28, scope: !26)
!52 = !DILocation(line: 28, column: 12, scope: !26)
!53 = !DILocation(line: 28, column: 7, scope: !26)
!54 = !DILocation(line: 28, column: 16, scope: !26)
!55 = !DILocation(line: 28, column: 26, scope: !26)
!56 = !DILocation(line: 28, column: 16, scope: !26)
!57 = !DILocation(line: 30, column: 3, scope: !26)
!58 = !DILocation(line: 30, column: 11, scope: !26)
!59 = !DILocation(line: 31, column: 1, scope: !26)
```

IR metadata after Simplify CFG

Logical views for Simplify CFG and SLP Vectorizer passes

```
!llvm.debug = ![]()
!llvm.module.flags = [{!5, !6, !7, !8, !9, !10, !11}]
!llvm.debug.info = ![]()

0 = distinct ID!CompileUnit(language: DW_LANG_C_plus_plus, !4, file: !1, producer: "clang", isOptimized: false, runtimeVersion: 0,
    emissionKind: FullDebug, retainedTypes: !2, splitDebuggingInfo: false, nameTableKind: None)
1 = ID!File(filename: "test.cpp", directory: "")
2 = ![]()
3 = ID!DerivedType!tag DW_TAG_typedef_name: "F32", file: !1, line: 2, baseType: !4
4 = ID!BasicType!name: "knot", size: 32, encoding: DW_ATE_float
5 = !{!2, !7, "Dwarf Version", !3, !5}
6 = !{!2, !7, "Debug Info Version", !3, !6}
12 = !{!7, "clang"}
13 = distinct ID!Subprogram(name: "RandF32", linkageName: ".ZTRandf32v", scope: !1, file: !1, line: 13, type: !4, scopeLine: 13, flags: !5, splitPrototyped: !6, splitFlags: !7, splitFlagDefinition: !8, unit: !9, retainedNodes: !10)
14 = ID!Subroutine!type!types: !15
15 = ![]()
16 = ![]()
17 = !ID!Location!line: 14, column: 15, scope: !13
18 = !ID!LocalVariable!name: "uRand", scope: !13, file: !1, line: 14, type: !19
19 = ID!DerivedType!tag DW_TAG_typedef_name: "U32", file: !1, line: 6, baseType: !20
20 = ID!BasicType!name: "unsigned int", size: 32, encoding: DW_ATE_unsigned
21 = !ID!Location!line: 0, scope: !13
22 = !ID!Location!line: 15, column: 21, scope: !13
23 = !ID!Location!line: 15, column: 27, scope: !13
24 = !ID!LocalVariable!name: "Rand", scope: !13, file: !1, line: 15, type: !3
25 = !ID!Location!line: 16, column: 3, scope: !13
26 = distinct ID!Subprogram(name: "randGauss", linkageName: ".ZrandGaussPT", scope: !1, file: !1, line: 19, type: !27, scopeLine: 19, flags: !28, splitPrototyped: !29, splitFlags: !30, splitFlagDefinition: !31, unit: !32, retainedNodes: !16)
27 = ID!Subroutine!type!types: !28
28 = !ID!Location!line: 29, column: 1, scope: !29
29 = ID!Type!pointer DW_TAG_pointer_type, baseType: !4, size: 64
30 = ID!LocalVariable!name: "work", file: !1, scope: !26, file: !1, line: 19, type: !29
31 = !ID!Location!line: 0, scope: !28
32 = !ID!Location!line: 22, column: 3, scope: !26
33 = !ID!Location!line: 23, column: 16, scope: !34
34 = distinct ID!LexicalBlock!scope: !25, file: !1, line: 22, column: !6
35 = !ID!Location!line: 23, column: 26, scope: !34
36 = !ID!LocalVariable!name: "x1", scope: !26, file: !1, line: 20, type: !4
37 = !ID!Location!line: 24, column: 16, scope: !34
38 = !ID!Location!line: 24, column: 26, scope: !34
39 = !ID!LocalVariable!name: "x2", scope: !26, file: !1, line: 20, type: !4
40 = !ID!Location!line: 25, column: 22, scope: !34
41 = !ID!Location!line: 25, column: 26, scope: !34
42 = !ID!LocalVariable!name: "y", scope: !26, file: !1, line: 20, type: !4
43 = !ID!Location!line: 26, column: 14, scope: !26
44 = !ID!Location!line: 26, column: 3, scope: !34
45 = distinct !{!45, !32, !46, !47}
46 = !ID!Location!line: 26, column: 20, scope: !26
47 = !{"!llvm.loop.mustProgress"}
48 = !ID!Location!line: 28, column: 20, scope: !26
49 = !ID!Location!line: 28, column: 18, scope: !26
50 = !ID!Location!line: 28, column: 30, scope: !26
51 = !ID!Location!line: 28, column: 28, scope: !26
52 = !ID!Location!line: 28, column: 12, scope: !26
53 = !ID!Location!line: 28, column: 7, scope: !26
54 = !ID!Location!line: 28, column: 16, scope: !26
55 = !ID!Location!line: 28, column: 26, scope: !26
56 = !ID!Location!line: 28, column: 28, scope: !26
57 = !ID!Location!line: 28, column: 30, scope: !26
58 = !ID!Location!line: 30, column: 11, scope: !26
59 = !ID!Location!line: 31, column: 1, scope: !26
```

IR metadata after Simplify CFG

Logical View:

```
[000]      {File} 'simplify-cfg.ll'
[001]      {CompileUnit} 'test.cpp'
[002]      6      {TypeAlias} 'U32' -> 'unsigned int'
[002]      7      {TypeAlias} 'F32' -> 'float'
[002]      13     {Function} 'RandF32' -> 'float'
[003]      14     {Variable} 'uRand' -> 'U32'
[003]      15     {Variable} 'fRand' -> 'F32'
[003]      14     {Line}
[003]      15     {Line}
[003]      15     {Line}
[003]      16     {Line}
[002]      19     {Function} 'randGauss' -> 'void'
[003]      19     {Parameter} 'work' -> '* float'
[003]      20     {Variable} 'w' -> 'float'
[003]      20     {Variable} 'x1' -> 'float'
[003]      20     {Variable} 'x2' -> 'float'
[003]      28     {Line}
[003]      29     {Line}
[003]      30     {Line}
[003]      30     {Line}
[003]      30     {Line}
[003]      31     {Line}
[003]      23     {Line}
[003]      24     {Line}
[003]      25     {Line}
```

Logical view after Simplify CFG

Logical views for Simplify CFG and SLP Vectorizer passes



```

llvm.debug = !(0)
llvm.moduleFlags = !(5, 16, 17, 18, 19, !10, !11)
llvm.debug = !(0)
0 = distinct IDCompileUnit(language: DW_LANG_C_plus_plus, file: !1, producer: "clang", isOptimized: false, runtimeVersion: 0,
    emissionKind: FullDebug, retainedTypes: 2, splitDebugLineinfo: false, nameTableKind: None)
11 = IDFile(fileName: "test.cpp", directory: "")
12 = !{!3}
13 = IDCompileUnit(language: DW_LANG_C_plus_plus, file: !2, baseType: 14)
14 = IDBasicType(name: "knot", size: 32, encoding: DW_ATE_float)
15 = !{!2,2, !DwarfVersion, !32, 5}
16 = !{!2,2, !DebugInfoVersion, !3, 3}
112 = !{!clang}
113 = distinct IDSubprogram(name: "RandF32", linkageName: ".ZTRandf32v", scope: !1, file: !1, line: 13, type: !14, scopeLine: 13, flags: DiflagPrototyped, splitFlags: DiflagDefinition, unit: !10, retainedNodes: !16)
114 = IDSubroutineType(types: 15)
115 = !{!4}
116 = !{!5}
117 = !DILocation(line: 14, column: 15, scope: !13)
118 = IDLocalVariable(name: "uRand", scope: !13, file: !1, line: 14, type: !19)
119 = IDDerivedType(tag: DW_TAG_typedef, name: "U32", file: !1, line: 6, baseType: !20)
120 = IDBasicType(name: "unsigned int", size: 32, encoding: DW_ATE_unsigned)
121 = !DILocation(line: 0, scope: !13)
122 = !DILocation(line: 15, column: 21, scope: !13)
123 = !DILocation(line: 15, column: 27, scope: !13)
124 = IDLocalVariable(name: "Rand", scope: !13, file: !1, line: 15, type: !13)
125 = !DILocation(line: 16, column: 3, scope: !13)
126 = distinct IDSubprogram(name: "randGauss", linkageName: ".ZrandGaussPT", scope: !1, file: !1, line: 19, type: !27, scopeLine: 19, flags: DiflagPrototyped, splitFlags: DiflagDefinition, unit: !10, retainedNodes: !16)
127 = IDSubroutineType(types: 28)
128 = !{!frnkt, !25}
129 = IDDerivedType(tag: DW_TAG_pointer_type, baseType: !4, size: 64)
130 = IDLocalVariable(name: "work", file: !1, scope: !26, file: !1, line: 19, type: !29)
131 = !DILocation(line: 0, scope: !28)
132 = !DILocation(line: 22, column: 3, scope: !26)
133 = !DILocation(line: 23, column: 16, scope: !34)
134 = distinct IDLexicalBlock(scope: !26, file: !1, line: 22, column: !6)
135 = !DILocation(line: 23, column: 26, scope: !34)
136 = IDLocalVariable(name: "x1", scope: !26, file: !1, line: 20, type: !4)
137 = !DILocation(line: 24, column: 16, scope: !34)
138 = !DILocation(line: 24, column: 26, scope: !34)
139 = IDLocalVariable(name: "x2", scope: !26, file: !1, line: 20, type: !4)
140 = !DILocation(line: 25, column: 22, scope: !34)
141 = !DILocation(line: 25, column: 28, scope: !34)
142 = IDLocalVariable(name: "w", scope: !26, file: !1, line: 20, type: !4)
143 = !DILocation(line: 26, column: 14, scope: !26)
144 = !DILocation(line: 26, column: 3, scope: !34)
145 = distinct !{!45, !32, !46, !47}
146 = !DILocation(line: 26, column: 20, scope: !26)
147 = !{!lm.loop.mustProgress}
148 = !DILocation(line: 28, column: 20, scope: !26)
149 = !DILocation(line: 28, column: 18, scope: !26)
150 = !DILocation(line: 28, column: 30, scope: !26)
151 = !DILocation(line: 28, column: 28, scope: !26)
152 = !DILocation(line: 28, column: 12, scope: !26)
153 = !DILocation(line: 28, column: 7, scope: !26)
154 = !DILocation(line: 28, column: 16, scope: !26)
155 = !DILocation(line: 28, column: 24, scope: !26)
156 = !DILocation(line: 28, column: 26, scope: !26)
157 = !DILocation(line: 30, column: 3, scope: !26)
158 = !DILocation(line: 30, column: 11, scope: !26)
159 = !DILocation(line: 31, column: 1, scope: !26)

```

IR metadata after Simplify CFG

©2025 Sony Interactive Entertainment

Logical View:

```

[000]      {File} 'simplify-cfg.ll'
[001]      {CompileUnit} 'test.cpp'
[002]      6      {TypeAlias} 'U32' -> 'unsigned int'
[002]      7      {TypeAlias} 'F32' -> 'float'
[002]      13     {Function} 'RandF32' -> 'float'
[003]      14     {Variable} 'uRand' -> 'U32'
[003]      15     {Variable} 'fRand' -> 'F32'
[003]      14     {Line}
[003]      15     {Line}
[003]      15     {Line}
[003]      16     {Line}
[002]      19     {Function} 'randGauss' -> 'void'
[003]      19     {Parameter} 'work' -> '* float'
[003]      20     {Variable} 'w' -> 'float'
[003]      20     {Variable} 'x1' -> 'float'
[003]      20     {Variable} 'x2' -> 'float'
[003]      28     {Line}
[003]      29     {Line}
[003]      30     {Line}
[003]      30     {Line}
[003]      30     {Line}
[003]      31     {Line}
[003]      23     {Line}
[003]      24     {Line}
[003]      25     {Line}

```

Logical view after Simplify CFG

Logical View:

```

[000]      {File} 'slp-vectorizer.ll'
[001]      {CompileUnit} 'test.cpp'
[002]      6      {TypeAlias} 'U32' -> 'unsigned int'
[002]      7      {TypeAlias} 'F32' -> 'float'
[002]      13     {Function} 'RandF32' -> 'float'
[003]      14     {Variable} 'uRand' -> 'U32'
[003]      15     {Variable} 'fRand' -> 'F32'
[003]      14     {Line}
[003]      15     {Line}
[003]      15     {Line}
[003]      16     {Line}
[002]      19     {Function} 'randGauss' -> 'void'
[003]      19     {Parameter} 'work' -> '* float'
[003]      20     {Variable} 'w' -> 'float'
[003]      28     {Line}
[003]      29     {Line}
[003]      29     {Line}
[003]      29     {Line}
[003]      31     {Line}
[003]      23     {Line}
[003]      23     {Line}
[003]      23     {Line}
[003]      25     {Line}
[003]      25     {Line}

```

Logical view after SLP Vectorizer

Common options to compare logical views when dealing with IR

- --report=list --report=view
- --print=scopes,types,symbols,lines

Common options to compare logical views when dealing with IR

- `--report=list` `--report=view`
- `--print=scopes,types,symbols,lines`

IR tests

- After Simplify CFG pass: `simplify-cfg.ll`
- After SLP Vectorizer pass: `slp-vectorizer.ll`

Common options to compare logical views when dealing with IR

- --report=list --report=view
- --print=scopes,types,symbols,lines

IR tests

- After Simplify CFG pass: simplify-cfg.ll
- After SLP Vectorizer pass: slp-vectorizer.ll

llvm-debuginfo-analyzer command line

- --compare=symbols,lines --report=list --print=symbols,lines simplify-cfg.ll slp-vectorizer.ll
- --compare=symbols,lines --report=view --print=symbols simplify-cfg.ll slp-vectorizer.ll

Logical view changes - comparison tool

The screenshot shows two code editors side-by-side, each displaying a logical view of an intermediate representation (IR) for different files.

Left Editor (simplify-cfg.view):

```
Logical View:  
[000] (File) 'simplify-cfg.ll'  
  
[001]     (CompileUnit) 'test.cpp'  
[002]     6      (TypeAlias) 'U32' -> 'unsigned int'  
[002]     7      (TypeAlias) 'F32' -> 'float'  
[002]     13     (Function) extern not_inlined  
'RandF32' -> 'float'  
[003]     14     (Variable) 'uRand' -> 'U32'  
[003]     15     (Variable) 'fRand' -> 'F32'  
[003]     14     (Line)  
[003]     15     (Line)  
[003]     15     (Line)  
[003]     16     (Line)  
[002]     19     (Function) extern not_inlined  
'randGauss' -> 'void'  
[003]     19     (Parameter) 'work' -> '* float'  
[003]     20     (Variable) 'w' -> 'float'  
[003]     20     (Variable) 'x1' -> 'float'  
[003]     20     (Variable) 'x2' -> 'float'  
  
[003]     22     (Line)  
[003]     26     (Line)  
[003]     28     (Line)  
[003]     29     (Line)  
[003]     29     (Line)  
[003]     30     (Line)
```

Right Editor (slp-vectorizer.view):

```
Logical View:  
[000] (File) 'slp-vectorizer.ll'  
  
[001]     (CompileUnit) 'test.cpp'  
[002]     6      (TypeAlias) 'U32' -> 'unsigned int'  
[002]     7      (TypeAlias) 'F32' -> 'float'  
[002]     13     (Function) extern not_inlined  
'RandF32' -> 'float'  
[003]     14     (Variable) 'uRand' -> 'U32'  
[003]     15     (Variable) 'fRand' -> 'F32'  
[003]     14     (Line)  
[003]     15     (Line)  
[003]     15     (Line)  
[003]     16     (Line)  
[002]     19     (Function) extern not_inlined  
'randGauss' -> 'void'  
[003]     19     (Parameter) 'work' -> '* float'  
[003]     20     (Variable) 'w' -> 'float'  
  
[003]     22     (Line)  
[003]     26     (Line)  
[003]     28     (Line)  
[003]     29     (Line)  
[003]     29     (Line)  
[003]     29     (Line)
```

The bottom status bar shows file names, encodings (Windows-1252), and line numbers (17-18). A command-line interface at the bottom displays the command used to run the tool:

```
* [003] 20     (Variable) 'x1' -> 'float'  
[003] 20     (Variable) 'x2' -> 'float'
```

IR changes: comparison tool

Logical view changes - built-in compare (report mode)



The screenshot shows two terminal windows side-by-side. Both windows have a title bar with the name of the view and a 'Logical View:' header.

Left Terminal (simplify-cfg.view):

```
Logical View:  
[000] (File) 'simplify-cfg.ll'  
  
[001]      (CompileUnit) 'test.cpp'  
[002]      6      (TypeAlias) 'U32' -> 'unsigned int'  
[002]      7      (TypeAlias) 'F32' -> 'float'  
[002]      13     (Function) extern not_inlined  
'RandF32' -> 'float'  
[003]      14     (Variable) 'uRand' -> 'U32'  
[003]      15     (Variable) 'fRand' -> 'F32'  
[003]      14     (Line)  
[003]      15     (Line)  
[003]      15     (Line)  
[003]      16     (Line)  
[002]      19     (Function) extern not_inlined  
'randGauss' -> 'void'  
[003]      19     (Parameter) 'work' -> '* float'  
[003]      20     (Variable) 'w' -> 'float'  
[003]      20     (Variable) 'x1' -> 'float'  
[003]      20     (Variable) 'x2' -> 'float'  
  
[003]      22     (Line)  
[003]      26     (Line)  
[003]      28     (Line)  
[003]      29     (Line)  
[003]      29     (Line)  
[003]      29     (Line)  
[003]      30     (Line)  
  
Ln: 18 Col: 1/49 Ch: 1/49 EOL: LF          Windows-1252      Unix          Line: 17-18          Windows-1252      Unix
```

Right Terminal (slp-vectorizer.view):

```
Logical View:  
[000] (File) 'slp-vectorizer.ll'  
  
[001]      (CompileUnit) 'test.cpp'  
[002]      6      (TypeAlias) 'U32' -> 'unsigned int'  
[002]      7      (TypeAlias) 'F32' -> 'float'  
[002]      13     (Function) extern not_inlined  
'RandF32' -> 'float'  
[003]      14     (Variable) 'uRand' -> 'U32'  
[003]      15     (Variable) 'fRand' -> 'F32'  
[003]      14     (Line)  
[003]      15     (Line)  
[003]      15     (Line)  
[003]      16     (Line)  
[002]      19     (Function) extern not_inlined  
'randGauss' -> 'void'  
[003]      19     (Parameter) 'work' -> '* float'  
[003]      20     (Variable) 'w' -> 'float'  
  
[003]      22     (Line)  
[003]      26     (Line)  
[003]      28     (Line)  
[003]      29     (Line)  
[003]      29     (Line)  
[003]      29     (Line)  
[003]      29     (Line)  
  
Ln: 18 Col: 1/49 Ch: 1/49 EOL: LF          Windows-1252      Unix          Line: 17-18          Windows-1252      Unix
```

The bottom status bar of both terminals shows the line, column, character, and end-of-line information.

IR changes: comparison tool

Reference: 'simplify-cfg.ll'
Target: 'slp-vectorizer.ll'

IR changes: llvm-debuginfo-analyzer

Logical view changes - built-in compare (report mode)



simplify-cfg.view slp-vectorizer.view

```

Logical View:                                     Logical View:
[000]   (File) 'simplify-cfg.ll'                [000]   (File) 'slp-vectorizer.ll'

[001]       (CompileUnit) 'test.cpp'             [001]       (CompileUnit) 'test.cpp'
[002]   6           (TypeAlias) 'U32' -> 'unsigned int'  [002]   6           (TypeAlias) 'U32' -> 'unsigned int'
[002]   7           (TypeAlias) 'F32' -> 'float'        [002]   7           (TypeAlias) 'F32' -> 'float'
[002]   13          (Function) extern not_inlined    [002]   13          (Function) extern not_inlined
'RandF32' -> 'float'                          'RandF32' -> 'float'
[003]   14          (Variable) 'uRand' -> 'U32'        [003]   14          (Variable) 'uRand' -> 'U32'
[003]   15          (Variable) 'fRand' -> 'F32'        [003]   15          (Variable) 'fRand' -> 'F32'
[003]   14          (Line)                           [003]   14          (Line)
[003]   15          (Line)                           [003]   15          (Line)
[003]   15          (Line)                           [003]   15          (Line)
[003]   16          (Line)                           [003]   16          (Line)
[002]   19          (Function) extern not_inlined    [002]   19          (Function) extern not_inlined
'randGauss' -> 'void'                         'randGauss' -> 'void'
[003]   19          (Parameter) 'work' -> '* float'  [003]   19          (Parameter) 'work' -> '* float'
[003]   20          (Variable) 'w' -> 'float'         [003]   20          (Variable) 'w' -> 'float'
[003]   20          (Variable) 'x1' -> 'float'         [003]   20          (Variable) 'x1' -> 'float'
[003]   20          (Variable) 'x2' -> 'float'         [003]   20          (Variable) 'x2' -> 'float'

[003]   22          (Line)                           [003]   22          (Line)
[003]   26          (Line)                           [003]   26          (Line)
[003]   28          (Line)                           [003]   28          (Line)
[003]   29          (Line)                           [003]   29          (Line)
[003]   29          (Line)                           [003]   29          (Line)
[003]   30          (Line)                           [003]   29          (Line)

Ln:18 Col:1/49 Ch:1/49 EOL:LF      Windows-1252      Unix      Line:17-18      Windows-1252      Unix
* [003] 20          (Variable) 'x1' -> 'float'      * [003] 20          (Variable) 'x1' -> 'float'
* [003] 20          (Variable) 'x2' -> 'float'      * [003] 20          (Variable) 'x2' -> 'float'

```

IR changes: comparison tool

Reference: 'simplify-cfg.ll'
Target: 'slp-vectorizer.ll'

(2) Missing Symbols:

- 20 {Variable} 'x1' -> 'float'
- 20 {Variable} 'x2' -> 'float'

IR changes: llvm-debuginfo-analyzer

Logical view changes - built-in compare (report mode)

The screenshot shows two windows side-by-side, each displaying a logical view of LLVM IR. The left window is titled 'simplify-cfg.view' and the right window is titled 'slp-vectorizer.view'. Both windows show the same code structure but with different line numbers. The bottom of the left window has a status bar with 'Ln: 18 Col: 1/49 Ch: 1/49 EOL: LF' and a command line with 'x [003] 20 (Variable) 'x1' -> 'float''. The bottom of the right window also has a status bar with 'Ln: 17-18 Windows-1252 Unix'.

```

Logical View: simplify-cfg.view
[000] (File) 'simplify-cfg.ll'
[001]     (CompileUnit) 'test.cpp'
[002]     6     (TypeAlias) 'U32' -> 'unsigned int'
[002]     7     (TypeAlias) 'F32' -> 'float'
[002]     13    (Function) extern not_inlined
'RandF32' -> 'float'
[003]     14     (Variable) 'uRand' -> 'U32'
[003]     15     (Variable) 'fRand' -> 'F32'
[003]     14     (Line)
[003]     15     (Line)
[003]     15     (Line)
[003]     16     (Line)
[002]     19    (Function) extern not_inlined
'randGauss' -> 'void'
[003]     19     (Parameter) 'work' -> '* float'
[003]     20     (Variable) 'w' -> 'float'
[003]     20     (Variable) 'x1' -> 'float'
[003]     20     (Variable) 'x2' -> 'float'
[003]     22     (Line)
[003]     26     (Line)
[003]     28     (Line)
[003]     29     (Line)
[003]     29     (Line)
[003]     30     (Line)

Logical View: slp-vectorizer.view
[000] (File) 'slp-vectorizer.ll'
[001]     (CompileUnit) 'test.cpp'
[002]     6     (TypeAlias) 'U32' -> 'unsigned int'
[002]     7     (TypeAlias) 'F32' -> 'float'
[002]     13    (Function) extern not_inlined
'RandF32' -> 'float'
[003]     14     (Variable) 'uRand' -> 'U32'
[003]     15     (Variable) 'fRand' -> 'F32'
[003]     14     (Line)
[003]     15     (Line)
[003]     15     (Line)
[003]     16     (Line)
[002]     19    (Function) extern not_inlined
'randGauss' -> 'void'
[003]     19     (Parameter) 'work' -> '* float'
[003]     20     (Variable) 'w' -> 'float'
[003]     22     (Line)
[003]     26     (Line)
[003]     28     (Line)
[003]     29     (Line)
[003]     29     (Line)
[003]     29     (Line)
[003]     29     (Line)

```

IR changes: comparison tool

Reference: 'simplify-cfg.ll'
Target: 'slp-vectorizer.ll'

(2) Missing Symbols:
- 20 {Variable} 'x1' -> 'float'
- 20 {Variable} 'x2' -> 'float'

(5) Missing Lines:
- 24 {Line}
- 30 {Line}
- 30 {Line}
- 30 {Line}
- 24 {Line}

The screenshot shows a terminal window with the output of the LLVM debuginfo analyzer. It lists 'Missing Symbols' and 'Missing Lines' between the 'simplify-cfg.ll' reference and the 'slp-vectorizer.ll' target. The 'Missing Symbols' section lists two entries: '20 {Variable} 'x1' -> 'float'' and '20 {Variable} 'x2' -> 'float''. The 'Missing Lines' section lists five entries: '24 {Line}', '30 {Line}', '30 {Line}', '30 {Line}', and '24 {Line}'.

IR changes: llvm-debuginfo-analyzer

Logical view changes - built-in compare (view mode)



The screenshot shows two windows side-by-side, each displaying a logical view of LLVM IR. The left window is titled 'simplify-cfg.view' and the right window is titled 'slp-vectorizer.view'. Both windows have a header 'Logical View:' followed by a list of IR instructions.

simplify-cfg.view (Left Window):

```
[000] (File) 'simplify-cfg.ll'
[001]     (CompileUnit) 'test.cpp'
[002]     6      (TypeAlias) 'U32' -> 'unsigned int'
[002]     7      (TypeAlias) 'F32' -> 'float'
[002]     13     (Function) extern not_inlined
'RandF32' -> 'float'
[003]     14     (Variable) 'uRand' -> 'U32'
[003]     15     (Variable) 'fRand' -> 'F32'
[003]     14     (Line)
[003]     15     (Line)
[003]     15     (Line)
[003]     16     (Line)
[002]     19     (Function) extern not_inlined
'randGauss' -> 'void'
[003]     19     (Parameter) 'work' -> '* float'
[003]     20     (Variable) 'w' -> 'float'
[003]     20     (Variable) 'x1' -> 'float'
[003]     20     (Variable) 'x2' -> 'float'
[003]     22     (Line)
[003]     26     (Line)
[003]     28     (Line)
[003]     29     (Line)
[003]     29     (Line)
[003]     29     (Line)
[003]     30     (Line)
Ln: 18 Col: 1/49 Ch: 1/49 EOL: LF          Windows-1252   Unix
* [003] 20     (Variable) 'x1' -> 'float'
[003] 20     (Variable) 'x2' -> 'float'
```

slp-vectorizer.view (Right Window):

```
[000] (File) 'slp-vectorizer.ll'
[001]     (CompileUnit) 'test.cpp'
[002]     6      (TypeAlias) 'U32' -> 'unsigned int'
[002]     7      (TypeAlias) 'F32' -> 'float'
[002]     13     (Function) extern not_inlined
'RandF32' -> 'float'
[003]     14     (Variable) 'uRand' -> 'U32'
[003]     15     (Variable) 'fRand' -> 'F32'
[003]     14     (Line)
[003]     15     (Line)
[003]     15     (Line)
[003]     16     (Line)
[002]     19     (Function) extern not_inlined
'randGauss' -> 'void'
[003]     19     (Parameter) 'work' -> '* float'
[003]     20     (Variable) 'w' -> 'float'
[003]     22     (Line)
[003]     26     (Line)
[003]     28     (Line)
[003]     29     (Line)
[003]     29     (Line)
[003]     29     (Line)
[003]     29     (Line)
Ln: 17-18          Windows-1252   Unix
```

IR changes: comparison tool

Reference: 'simplify-cfg.ll'
Target: 'slp-vectorizer.ll'

(2) Missing Symbols:

- 20 {Variable} 'x1' -> 'float'
- 20 {Variable} 'x2' -> 'float'

(5) Missing Lines:

- 24 {Line}
- 30 {line}
- 30 {line}
- 30 {line}
- 24 {line}

Reference: 'simplify-cfg.ll'
Target: 'slp-vectorizer.ll'

IR changes: llvm-debuginfo-analyzer

Logical view changes - built-in compare (view mode)



Logical View:

	simplify-cfg.view	slp-vectorizer.view
[000]	(File) 'simplify-cfg.ll'	(File) 'slp-vectorizer.ll'
[001]	(CompileUnit) 'test.cpp'	(CompileUnit) 'test.cpp'
[002]	6 (TypeAlias) 'U32' -> 'unsigned int'	6 (TypeAlias) 'U32' -> 'unsigned int'
[002]	7 (TypeAlias) 'F32' -> 'float'	7 (TypeAlias) 'F32' -> 'float'
[002]	13 (Function) extern not_inlined 'RandF32' -> 'float'	13 (Function) extern not_inlined 'RandF32' -> 'float'
[003]	14 (Variable) 'uRand' -> 'U32'	14 (Variable) 'uRand' -> 'U32'
[003]	15 (Variable) 'fRand' -> 'F32'	15 (Variable) 'fRand' -> 'F32'
[003]	14 (Line)	14 (Line)
[003]	15 (Line)	15 (Line)
[003]	15 (Line)	15 (Line)
[003]	16 (Line)	16 (Line)
[002]	19 (Function) extern not_inlined 'randGauss' -> 'void'	19 (Function) extern not_inlined 'randGauss' -> 'void'
[003]	19 (Parameter) 'work' -> '* float'	19 (Parameter) 'work' -> '* float'
[003]	20 (Variable) 'w' -> 'float'	20 (Variable) 'w' -> 'float'
[003]	20 (Variable) 'x1' -> 'float'	20 (Variable) 'x1' -> 'float'
[003]	20 (Variable) 'x2' -> 'float'	20 (Variable) 'x2' -> 'float'
[003]	22 (Line)	22 (Line)
[003]	26 (Line)	26 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	29 (Line)	29 (Line)
[003]	29 (Line)	29 (Line)
[003]	30 (Line)	29 (Line)

Ln:18 Col:1/49 Ch:1/49 EOL:LF Windows-1252 Unix Line:17-18 Windows-1252 Unix

* [003] 20 (Variable) 'x1' -> 'float'
* [003] 20 (Variable) 'x2' -> 'float'

IR changes: comparison tool

Reference: 'simplify-cfg.ll'
Target: 'slp-vectorizer.ll'

(2) Missing Symbols:
- 20 {Variable} 'x1' -> 'float'
- 20 {Variable} 'x2' -> 'float'

(5) Missing Lines:
- 24 {Line}
- 30 {Line}
- 30 {Line}
- 30 {Line}
- 24 {Line}

Reference: 'simplify-cfg.ll'
Target: 'slp-vectorizer.ll'

Logical View:
(File) 'simplify-cfg.ll'

```
{CompileUnit} 'test.cpp'
13 {Function} extern not_inlined 'RandF32' -> 'float'
14 {Variable} 'uRand' -> 'U32'
15 {Variable} 'fRand' -> 'F32'
19 {Function} extern not_inlined 'randGauss' -> 'void'
19 {Parameter} 'work' -> '* float'
20 {Variable} 'w' -> 'float'
```

IR changes: llvm-debuginfo-analyzer

Logical view changes - built-in compare (view mode)



Logical View:

	simplify-cfg.view	slp-vectorizer.view
[000]	(File) 'simplify-cfg.ll'	(File) 'slp-vectorizer.ll'
[001]	(CompileUnit) 'test.cpp'	(CompileUnit) 'test.cpp'
[002]	6 (TypeAlias) 'U32' -> 'unsigned int'	6 (TypeAlias) 'U32' -> 'unsigned int'
[002]	7 (TypeAlias) 'F32' -> 'float'	7 (TypeAlias) 'F32' -> 'float'
[002]	13 (Function) extern not_inlined 'RandF32' -> 'float'	13 (Function) extern not_inlined 'RandF32' -> 'float'
[003]	14 (Variable) 'uRand' -> 'U32'	14 (Variable) 'uRand' -> 'U32'
[003]	15 (Variable) 'fRand' -> 'F32'	15 (Variable) 'fRand' -> 'F32'
[003]	14 (Line)	14 (Line)
[003]	15 (Line)	15 (Line)
[003]	15 (Line)	15 (Line)
[003]	16 (Line)	16 (Line)
[002]	19 (Function) extern not_inlined 'randGauss' -> 'void'	19 (Function) extern not_inlined 'randGauss' -> 'void'
[003]	19 (Parameter) 'work' -> '* float'	19 (Parameter) 'work' -> '* float'
[003]	20 (Variable) 'w' -> 'float'	20 (Variable) 'w' -> 'float'
[003]	20 (Variable) 'x1' -> 'float'	20 (Variable) 'x1' -> 'float'
[003]	20 (Variable) 'x2' -> 'float'	20 (Variable) 'x2' -> 'float'
[003]	22 (Line)	22 (Line)
[003]	26 (Line)	26 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	28 (Line)	28 (Line)
[003]	29 (Line)	29 (Line)
[003]	29 (Line)	29 (Line)
[003]	30 (Line)	29 (Line)

Ln:18 Col:1/49 Ch:1/49 EOL:LF Windows-1252 Unix Line:17-18 Windows-1252 Unix

* [003] 20 (Variable) 'x1' -> 'float'
* [003] 20 (Variable) 'x2' -> 'float'

IR changes: comparison tool

Reference: 'simplify-cfg.ll'
Target: 'slp-vectorizer.ll'

(2) Missing Symbols:

- 20 {Variable} 'x1' -> 'float'
- 20 {Variable} 'x2' -> 'float'

(5) Missing Lines:

- 24 {Line}
- 30 {line}
- 30 {line}
- 30 {line}
- 24 {Line}

Reference: 'simplify-cfg.ll'
Target: 'slp-vectorizer.ll'

Logical View:

	{File} 'simplify-cfg.ll'
	{CompileUnit} 'test.cpp'
13	{Function} extern not_inlined 'RandF32' -> 'float'
14	{Variable} 'uRand' -> 'U32'
15	{Variable} 'fRand' -> 'F32'
19	{Function} extern not_inlined 'randGauss' -> 'void'
19	{Parameter} 'work' -> '* float'
20	{Variable} 'w' -> 'float'
- 20	{Variable} 'x1' -> 'float'
- 20	{Variable} 'x2' -> 'float'

IR changes: llvm-debuginfo-analyzer

Conclusion

Reduce the noisiness of comparing the debuginfo in LLVM IR



slp-vectorizer.ll

Tools Plugins Window Help

File: slp-vectorizer.ll

```
LexicalBlock(scope: 126, file: 11, line: 23, column: 26, scope: 134)
able(name: "x1", scope: 126, file: 11, line: 24, column: 16, scope: 134)
able(name: "x2", scope: 126, file: 11, line: 25, column: 22, scope: 134)
able(name: "w", scope: 126, file: 11, line: 26, column: 14, scope: 126)
line: 26, column: 3, scope: 134)
45, [132, 146, 147]
line: 26, column: 20, scope: 126)
p.mustprogress"]
line: 28, column: 20, scope: 126)
line: 28, column: 16, scope: 126)
line: 28, column: 30, scope: 126)
line: 28, column: 28, scope: 126)
line: 28, column: 12, scope: 126)
line: 28, column: 7, scope: 126)
line: 29, column: 16, scope: 126)
line: 29, column: 11, scope: 126)
line: 30, column: 16, scope: 126)
line: 30, column: 3, scope: 126)
line: 30, column: 11, scope: 126)
line: 31, column: 1, scope: 126)
```

RLF Windows-1252 Win Line:131-132 Win

n(line: 23, column: 26, scope: 134)
riable(name: "x1", scope: 126, file: 11, line: 20, type: 14)
n(line: 24, column: 16, scope: 134)
n(line: 24, column: 26, scope: 134)
riable(name: "x1", scope: 126, file: 11, line: 20, type: 14)
n(line: 24, column: 16, scope: 134)
n(line: 23, column: 26, scope: 134)
riable(name: "x2", scope: 126, file: 11, line: 20, type: 14)

IR changes: comparison tool

slp-vectorizer.ll

Tools Plugins Window Help

File: simplify-cfg.ll

```
Logical View:
[000] (File) 'slp-vectorizer.ll'
[001] (CompileUnit) 'test.cpp'
[002] 6 (TypeAlias) 'U32' -> 'unsigned int'
[003] 7 (TypeAlias) 'F32' -> 'float'
[004] 13 (Function) extern not_inlined
[005] 14 (Variable) 'uRand' -> 'U32'
[006] 15 (Variable) 'fRand' -> 'F32'
[007] 14 (Line)
[008] 15 (Line)
[009] 15 (Line)
[010] 16 (Line)
[011] 19 (Function) extern not_inlined
[012] 19 (Parameter) 'work' -> '* float'
[013] 20 (Variable) 'w' -> 'float'
[014] 22 (Line)
[015] 26 (Line)
[016] 28 (Line)
[017] 28 (Line)
[018] 28 (Line)
[019] 28 (Line)
[020] 29 (Line)
[021] 29 (Line)
[022] 29 (Line)
[023] 29 (Line)
[024] 29 (Line)
```

Windows-1252 Unix Line:17-18 Win

(Variable) 'x1' -> 'float'
(Variable) 'x2' -> 'float'

Reference: 'simplify-cfg.ll'
Target: 'slp-vectorizer.ll'

(2) Missing Symbols:
- 20 {Variable} 'x1' -> 'float'
- 20 {Variable} 'x2' -> 'float'

(5) Missing Lines:
- 24 {Line}
- 30 {Line}
- 30 {Line}
- 30 {Line}
- 24 {Line}

Reference: 'simplify-cfg.ll'
Target: 'slp-vectorizer.ll'

Logical View:

{File} 'simplify-cfg.ll'

{CompileUnit} 'test.cpp'

13 {Function} 'RandF32' -> 'float'

14 {Variable} 'uRand' -> 'U32'

15 {Variable} 'fRand' -> 'F32'

19 {Function} 'randGauss' -> 'void'

19 {Parameter} 'work' -> '* float'

20 {Variable} 'w' -> 'float'

- 20 {Variable} 'x1' -> 'float'

- 20 {Variable} 'x2' -> 'float'

IR changes: llvm-debuginfo-analyzer



Thank you!



Developers' Meeting

BERLIN 2025

To be OR NOT to be

Piotr Fusik <p.fusik@samsung.com>

2025-04-16

Instruction	Operation	Unit	RISC-V Extension
andn rd, rs1, rs2	$rd = rs1 \& \sim rs2;$	Scalar	Zbb or Zbkb
orn rd, rs1, rs2	$rd = rs1 \sim rs2;$		
xnor rd, rs1, rs2	$rd = rs1 ^ \sim rs2;$		
vandn.vv vd, vs1, vs2	<pre>for (int i = 0; i < vl; i++) vd[i] = vs1[i] & ~vs2[i];</pre>	Vector	Zvkb
vandn.vx vd, vs1, rs2	<pre>for (int i = 0; i < vl; i++) vd[i] = vs1[i] & ~rs2;</pre>		

Input	Output
(and rs1, (not rs2))	(andn rs1, rs2)
(or rs1, (not rs2))	(orn rs1, rs2)
(xor rs1, (not rs2))	(xnor rs1, rs2)
(vand vs1, (not vs2))	(vandn vs1, vs2)
(vand vs1, (not rs2))	(vandn vs1, rs2)

Essentially:



Input	Output
(and rs1, (not rs2))	(andn rs1, rs2)
(or rs1, (not rs2))	(orn rs1, rs2)
(xor rs1, (not rs2))	(xnor rs1, rs2)
(vand vs1, (not vs2))	(vandn vs1, vs2)
(vand vs1, (not rs2))	(vandn vs1, rs2)

Real patterns:



```

let Predicates = [HasStdExtZbbOrZbb] in {
  def : Pat<(XLenVT (and GPR:$rs1, (not GPR:$rs2))), (ANDN GPR:$rs1, GPR:$rs2)>;
  def : Pat<(XLenVT (or GPR:$rs1, (not GPR:$rs2))), (ORN GPR:$rs1, GPR:$rs2)>;
  def : Pat<(XLenVT (xor GPR:$rs1, (not GPR:$rs2))), (XNOR GPR:$rs1, GPR:$rs2)>;
}

foreach vti = AllIntegerVectors in {
  let Predicates = !listconcat([HasStdExtZvkb],
    GetVTypePredicates<vti>.Predicates) in {
    def : Pat<(vti.Vector (and (riscv_vnot vti.RegClass:$rs1),
      vti.RegClass:$rs2)),
      (!cast<Instruction>"PseudoVANDN_VV_"#vti.LMul.MX)
      (vti.Vector (IMPLICIT_DEF)),
      vti.RegClass:$rs2,
      vti.RegClass:$rs1,
      vti.AVL, vti.Log2SEW, TA_MA)>;
    def : Pat<(vti.Vector (and (riscv_splat_vector
      (not vti.ScalarRegClass:$rs1)),
      vti.RegClass:$rs2)),
      (!cast<Instruction>"PseudoVANDN_VX_"#vti.LMul.MX)
      (vti.Vector (IMPLICIT_DEF)),
      vti.RegClass:$rs2,
      vti.ScalarRegClass:$rs1,
      vti.AVL, vti.Log2SEW, TA_MA)>;
  }
}

foreach vti = AllIntegerVectors in {
  let Predicates = !listconcat([HasStdExtZvkb],
    GetVTypePredicates<vti>.Predicates) in {
    def : Pat<(vti.Vector (riscv_and_vl (riscv_xor_vl
      (vti.Vector vti.RegClass:$rs1),
      (riscv_splat_vector -1),
      (vti.Vector vti.RegClass:$passthru),
      (vti.Mask V0),
      VLOpFrag),
      (vti.Vector vti.RegClass:$rs2),
      (vti.Vector vti.RegClass:$passthru),
      (vti.Mask V0),
      VLOpFrag)),
      (!cast<Instruction>"PseudoVANDN_VV_"#vti.LMul.MX#"_MASK")
      vti.RegClass:$passthru,
      vti.RegClass:$rs2,
      vti.RegClass:$rs1,
      (vti.Mask V0),
      GPR:$vl,
      vti.Log2SEW,
      TAIL_AGNOSTIC)>;
    def : Pat<(vti.Vector (riscv_and_vl (riscv_splat_vector
      (not vti.ScalarRegClass:$rs1)),
      (vti.Vector vti.RegClass:$rs2),
      (vti.Vector vti.RegClass:$passthru),
      (vti.Mask V0),
      VLOpFrag)),
      (!cast<Instruction>"PseudoVANDN_VX_"#vti.LMul.MX#"_MASK")
      vti.RegClass:$passthru,
      vti.RegClass:$rs2,
      vti.ScalarRegClass:$rs1,
      (vti.Mask V0),
      GPR:$vl,
      vti.Log2SEW,
      TAIL_AGNOSTIC)>;
  }
}
}

```

Optimization #1

Op with constant
↓
Inverted op
with inverted
constant

Optimization #2

Inversion
before the loop
↓
Inverted op in
the loop

Optimization #3

Target-
independent
transforms
emitting VANDN

- RISC-V instructions are 32-bit (plus optional 16-bit “compressed” instructions)
- Many instructions have variants with a sign-extended 12-bit immediate

Instructions	Operation	Comment
addi rd, x0, simm12	$rd = 0 + simm12$	Load small constants by adding to register X0 which is hardwired to zero
lui rd, simm20	$rd = simm20 \ll 12$	“Load Upper Immediate”
lui rd, simm20 addi rd, rd, simm12	$rd = (simm20 \ll 12) + simm12$	“Load Upper Immediate” followed by an addition

Input	Output
(and rs1, C)	(andn rs1, ~C)
(or rs1, C)	(orn rs1, ~C)
(xor rs1, C)	(xnor rs1, ~C)
(vand vs1, C)	(vandn vs1, ~C)

Example

Before	After
<code>lui a1, HI+1 addi a1, a1, -1 or a0, a0, a1</code>	<code>lui a1, ~HI orn a0, a0, a1</code>

llvm/lib/Target/RISCV/MCTargetDesc/RISCVMatInt.cpp
(>500 LOC)

Emits a sequence of LUI, ADDI, ADDIW, SLLI, SLLI.UW, RORI, NOT, BSETI, BCLRI, PACK, SH1ADD, SH2ADD, SH3ADD instructions

How to tell if inverting the constant is profitable?

```
int OrigImmCost = RISCVMatInt::getIntMatCost(APInt(64, Imm), ...);  
int NegImmCost = RISCVMatInt::getIntMatCost(APInt(64, ~Imm), ...);  
if (NegImmCost < OrigImmCost)  
    PerformTransform();
```

```
inverted = (not mask);  
for (...) { result = (and src, inverted); }  
  
for (...) { result = (and src, (not mask)); }  
  
for (...) { result = (andn src, mask); }
```

and similarly for "or", "xor" and "vand".

Low-hanging fruit: port this to x86 and Arm.

<https://github.com/llvm/llvm-project/issues/108840>

```
bool RISCVTargetLowering::hasAndNotCompare(SDValue Y) const {
    EVT VT = Y.getValueType();

    // FIXME: Support vectors once we have tests.
    if (VT.isVector())
        return false;

    return (Subtarget.hasStdExtZbb() || Subtarget.hasStdExtZbkb()) &&
           (!isa<ConstantSDNode>(Y) || cast<ConstantSDNode>(Y)->isOpaque());
}
```

Don't study LLVM internals too much (i.e. **how** things work), instead:

1. Decide **what** change to make
 - a. Invent a transform yourself (example: my optimization #1)
 - b. Pick up a GitHub ticket (example: my optimization #2)
 - c. Read the code (TODO/FIXME, example: my optimization #3)
2. Find **where** to do your change
 - Dump intermediate representations
 - Look for similar transforms
3. Decide **how** to do your change last
 - Often obvious if you know **what&where**
 - Other contributors can help you



Thank you



Developers' Meeting

BERLIN 2025

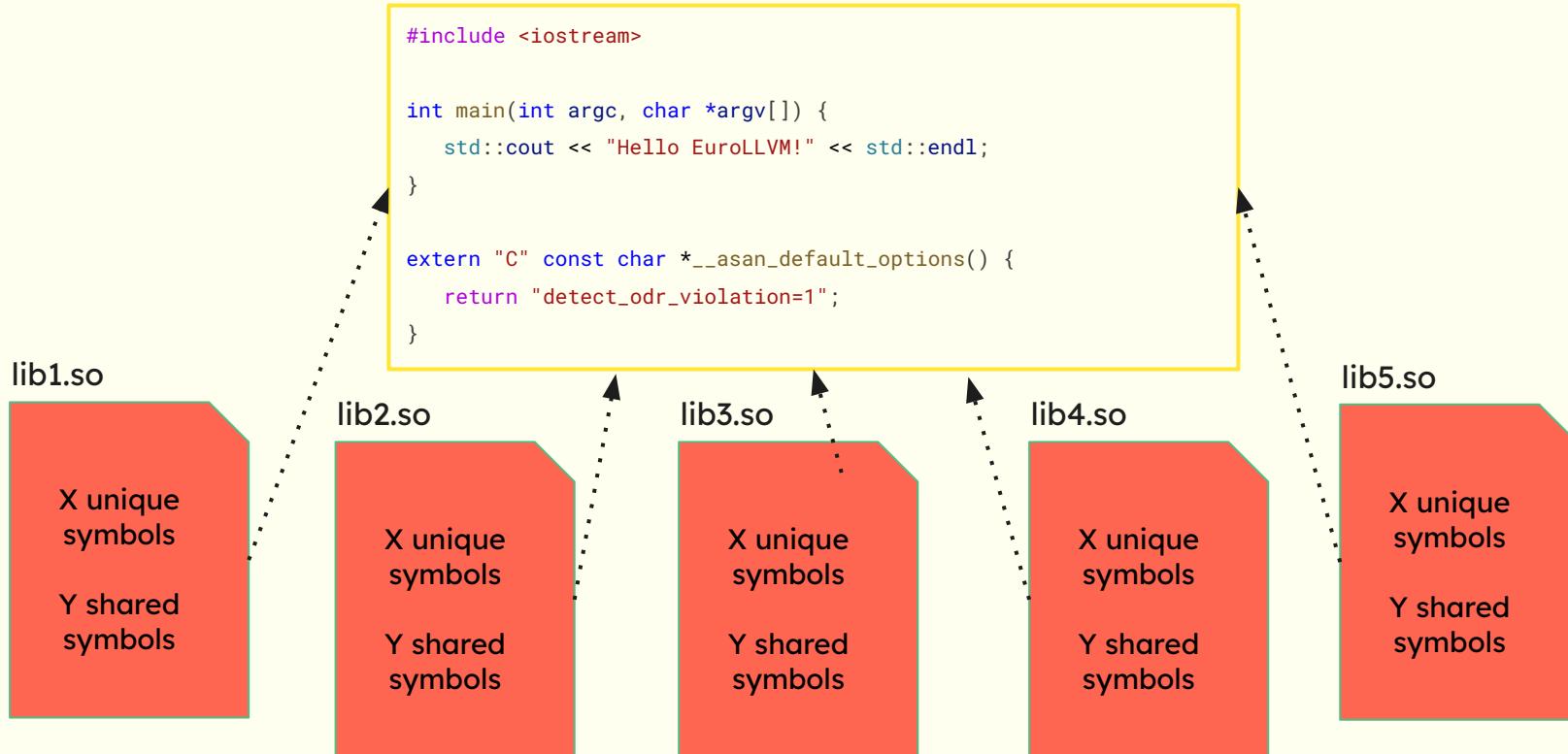
EuroLLVM '25

Artem Pianykh
Software Engineer @ Meta

Accidentally quadratic in compiler-rt/asan

Motivating Example / 1

<https://github.com/artempyanykh/eurollvm25>



Motivating Example / 2

```
~/d/eurollvm25 > time ./before_patch
```

```
Hello EuroLLVM!
```

Executed in	5.70 secs	fish	external
usr time	5.60 secs	0.00 micros	5.60 secs
sys time	0.03 secs	518.00 micros	0.03 secs

```
~/d/eurollvm25 > █
```

```
~/d/eurollvm25 > time ./after_patch
```

```
Hello EuroLLVM!
```

Executed in	96.93 millis	fish	external
usr time	46.83 millis	263.00 micros	46.57 millis
sys time	49.78 millis	218.00 micros	49.56 millis

```
~/d/eurollvm25 > █
```



**Simplified, but
representative of our
production**

perf report

Samples: 22K of event 'cycles:Pu', Event count (approx.): 22956920047				
Children	Self	Command	Shared Object	Symbol
+ 99.81%	0.00%	before_patch	ld-linux-x86-64.so.2	[.] _dl_start_user
+ 99.42%	0.00%	before_patch	ld-linux-x86-64.so.2	[.] _dl_init
+ 99.42%	0.00%	before_patch	ld-linux-x86-64.so.2	[.] call_init
- 99.42%	0.00%	before_patch	before_patch	[.] __asan_register_elf_globals
- __asan_register_elf_globals				
99.39% __asan_register_globals				
+ 99.39%	99.10%	before_patch	before_patch	[.] __asan_register_globals
0.38%	0.00%	before_patch	ld-linux-x86-64.so.2	[.] _dl_start
0.38%	0.00%	before_patch	ld-linux-x86-64.so.2	[.] _dl_sysdep_start
0.38%	0.00%	before_patch	ld-linux-x86-64.so.2	[.] dl_main
0.38%	0.07%	before_patch	ld-linux-x86-64.so.2	[.] _dl_relocate_object
0.27%	0.10%	before_patch	ld-linux-x86-64.so.2	[.] _dl_lookup_symbol_x
0.16%	0.15%	before_patch	ld-linux-x86-64.so.2	[.] do_lookup_x

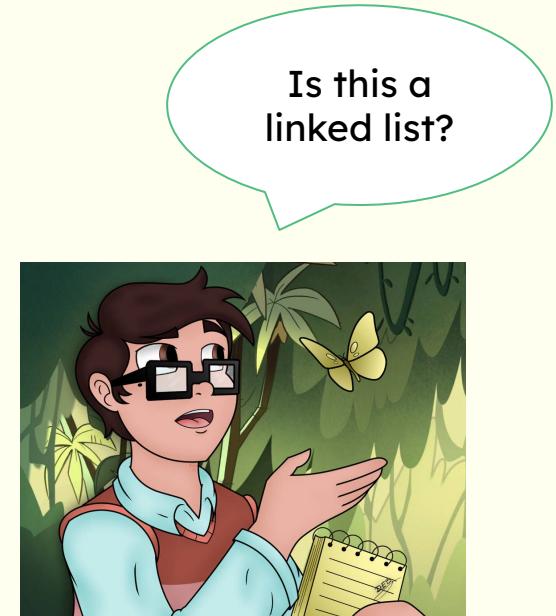




~~ASan is too slow! I better turn it off.~~
We need ASan! Let's dig deeper.

perf report cont.

```
Samples: 22K of event 'cycles:Pu', 4000 Hz, Event count (approx.): 22956920047
__asan_register_globals /home/arr/dev/eurollvm25/before_patch [Percent: local period]
Percent          xorl    %esi,%esi
                  movl    $0x0,%ecx
                  testq   %rdi,__asan::mu_for_globals
                  ↓ je     b41
7fa:             movq    %r12,%rdi
                  → callq  __asan::ReportODRViolation(__asan_global const*, unsigned int, __asan_
nop
0.02              movq    0x8(%r15),%r15
0.55              testq   %r15,%r15
0.14              ↓ je     910
81d:             movq    0x38(%r12),%rcx
                  movq    (%r15),%rax
                  movq    0x38(%rax),%rcx
                  cmpq
9.71
88.22             jne    810
                  movq    $0x54fce0,%rcx
                  cmpl    $0x1,0x64(%rcx)
                  ↓ jg     841
                  movq    (%rbx),%rcx
                  cmpq    0x8(%rax),%rcx
                  ↑ je     810
841:             movq    0x10(%rbx),%rdi
                  → callq  __asan::IsODRViolationSuppressed(char const*)
0.00
```



Culprit

```
static void CheckODRViolationViaIndicator(const Global *g) {
...
// If *odr_indicator is DEFINED ...
for (const auto &l : list_of_all_globals) {
    if (g->odr_indicator == l.g->odr_indicator &&
        (flags()->detect_odrViolation >= 2 || g->size != l.g->size) && !IsODRViolationSuppressed(g->name)) {
        ReportODRViolation(g, FindRegistrationSite(g), l.g,
                            FindRegistrationSite(l.g));
    }
}
```

*A potential violation triggers iteration over the **list** of **all** globals.*

Fix (list -> map)

9

[asan] Speed up ASan ODR indicator-based checking #100923

Merged vitalybuka merged 8 commits into `llvm:main` from `artempyanykh:fast-odr-indicator` on Aug 1, 2024

Conversation 32

Commits 8

Checks 5

Files changed 2

Edit <> Code

+95 -12



artempyanykh commented on Jul 28, 2024 · edited

Member

...

Summary:

When ASan checks for a potential ODR violation on a global it loops over a linked list of all globals to find those with the matching value of an indicator. With the default setting 'detect_odr_violation=1', ASan doesn't report violations on same-size globals but it still has to traverse the list. For larger binaries with a ton of shared libs and globals (and a non-trivial volume of same-sized duplicates) this gets extremely expensive.

This patch adds an indicator indexed (multi-)map of globals to speed up the search.

Reviewers

MaskRay

vitalybuka

Assignees

No one—assign yourself

<https://github.com/llvm/llvm-project/pull/100923>

Thanks to Vitaly Buka for the review and context on the compiler-rt codebase!

Fix (list -> map)

```

38 39 typedef IntrusiveList<GlobalListNode> ListOfGlobals;
.. 40 typedef DenseMap<uptr, ListOfGlobals> MapOfGlobals;
39 41
40 42 static Mutex mu_for_globals;
41 43 static ListOfGlobals list_of_all_globals;
.. 44 static MapOfGlobals map_of_globals_by_indicator;
42 45
43 46 static const int kDynamicInitGlobalsInitialCapacity = 512;
44 47 struct DynInitGlobal {

```

compiler-rt/lib/asan/asan_globals.cpp --- 3/3 --- C++

```

149 if (g->odr_indicator == UINTPTR_MAX)
150     return;
...
...
...
151 u8 *odr_indicator = reinterpret_cast<u8 *>(g->odr_indicator);
152 if (*odr_indicator == UNREGISTERED) {
153     *odr_indicator = REGISTERED;
154     return;
155 }
156 // If *odr_indicator is DEFINED, some module have already registered
157 // externally visible symbol with the same name. This is an ODR violation.
158 for (const auto &l : list_of_all_globals) {
159     if (g->odr_indicator == l.g->odr_indicator &&
160         (flags()->detect_odr_violation >= 2 || g->size != l.g->size) &&
161         !IsODRViolationSuppressed(g->name)) {
162         ReportODRViolation(g, FindRegistrationSite(g), l.g,
163                            FindRegistrationSite(l.g));
164     }
165 }
...
...
...
166 }
167
168 if (g->odr_indicator == UINTPTR_MAX)
169     return;
170
171 ListOfGlobals &relevant_globals =
172     map_of_globals_by_indicator[g->odr_indicator];
173
174 u8 *odr_indicator = reinterpret_cast<u8 *>(g->odr_indicator);
175 if (*odr_indicator == REGISTERED) {
...
...
176     // If *odr_indicator is REGISTERED, some module have already registered
177     // externally visible symbol with the same name. This is an ODR violation.
178     for (const auto &l : relevant_globals) {
179         if ((flags()->detect_odr_violation >= 2 || g->size != l.g->size) &&
...
180             !IsODRViolationSuppressed(g->name))
181             ReportODRViolation(g, FindRegistrationSite(g), l.g,
182                                FindRegistrationSite(l.g));
...
183     }
184 } else { // UNREGISTERED
185     *odr_indicator = REGISTERED;
186 }
187 AddGlobalToList(relevant_globals, g);
188 }
189

```

Thoughts

11

Developers' perception

“Sanitizers are too slow” is the most likely reaction. Can we provide better upper bounds and runtime diagnostics?

Benchmarks

We fixed one pathological case. How do we protect from regressions going forward?

ODR checker defaults

The current defaults lead to too many FPs. FNs are also possible. What does “good” mean for the ODR checker?

Thank you!

Artem Pianykh

Presentation reference:

<https://github.com/artempyanykh/eurollvm25>

Contacts:

arTEM.PiAnyKh@gmail.com

<https://pianykh.com/>



Developers' Meeting

BERLIN 2025

Dialects as a Dialect

Bringing native C++ registration to IRDL



UNIVERSITY OF
CAMBRIDGE

Ivan Ho

PhD Student

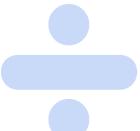
Supervised by Tobias Grosser



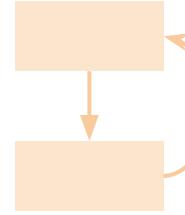
  

arith

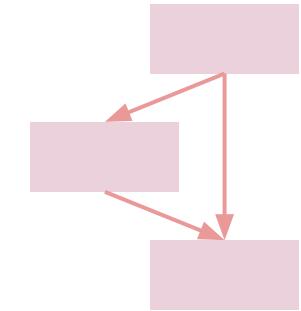




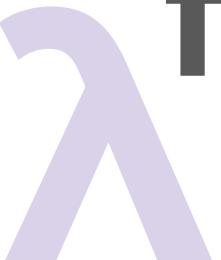
arith

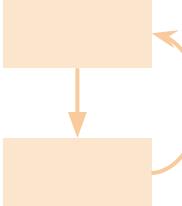
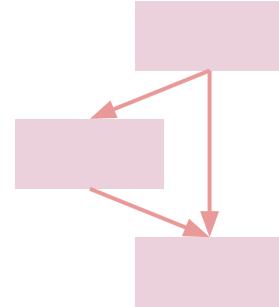


scf



  
arith

 **func** 


scf 

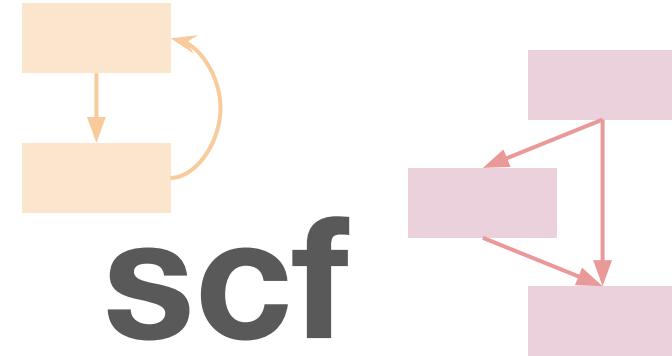
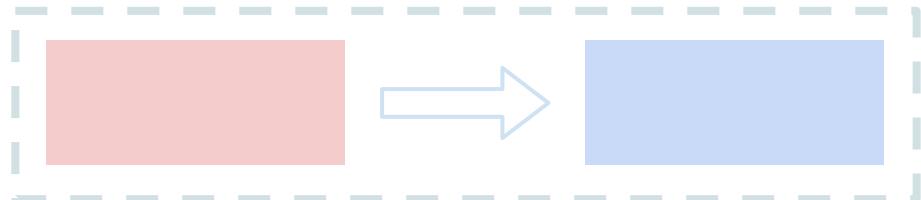
arith



func



transform



What if we had dialects as a dialect?



**What if we had
dialects as a dialect?**



IRDL

\\\\\\|⁹(◎`^'◎)⁹//||//

Why a Dialect for Dialects?

```
irdl.dialect cmath {  
  
    irdl.type complex {  
        %0 = irdl.is f32  
        %1 = irdl.is f64  
        %2 = irdl.any_of(%0, %1)  
  
        irdl.parameters (elem: %2)  
    }  
  
    irdl.operation norm {  
        %0 = irdl.any  
        %1 = irdl.parametric @cmath:::@complex(%0)  
  
        irdl.operands (in: %1)  
        irdl.results (res: %0)  
    }  
}
```

Why a Dialect for Dialects?

```
irdl.dialect cmath {
```

A *Dialect* definition is a single operation.

```
}
```

Why a Dialect for Dialects?

```
irdl.dialect cmath {  
    irdl.type complex {  
        %0 = irdl.is f32  
        %1 = irdl.is f64  
        %2 = irdl.any_of(%0, %1)  
  
        irdl.parameters (elem: %2)  
    }  
  
}
```

A *Dialect* definition is a single operation.

Types may have constraints.

Why a Dialect for Dialects?

```
irdl.dialect cmath {  
    irdl.type complex {  
        %0 = irdl.is f32  
        %1 = irdl.is f64  
        %2 = irdl.any_of(%0, %1)  
  
        irdl.parameters (elem: %2)  
    }  
  
    irdl.operation norm {  
        %0 = irdl.any  
        %1 = irdl.parametric @cmath:::@complex(%0)  
  
        irdl.operands (in: %1)  
        irdl.results (res: %0)  
    }  
}
```

A *Dialect* definition is a single operation.

Types may have constraints.

An *Operation* is defined similarly.

```
irdl.dialect cmath {  
  
    irdl.type complex {  
        // ...  
    }  
  
    irdl.operation norm {  
        %0 = irdl.any  
        %1 = irdl.parametric @cmath::@complex(%0)  
  
        irdl.operands (in: %1)  
        irdl.results (res: %0)  
    }  
}
```

```
$ mlir-opt --irdl-file=cmath.irdl.mlir example.cmath.mlir
```

```
irdl.dialect cmath {  
  
    irdl.type complex {  
        // ...  
    }  
  
    irdl.operation norm {  
        %0 = irdl.any  
        %1 = irdl.parametric @cmath::@complex(%0)  
  
        irdl.operands (in: %1)  
        irdl.results (res: %0)  
    }  
}
```

IRDL dialects are loaded **dynamically**

```
$ mlir-opt --irdl-file=cmath.irdl.mlir example.cmath.mlir
```

```
irdl.dialect cmath {  
  
    irdl.type complex {  
        // ...  
    }  
  
    irdl.operation norm {  
        %0 = irdl.any  
        %1 = irdl.parametric @cmath::@complex(%0)  
  
        irdl.operands (in: %1)  
        irdl.results (res: %0)  
    }  
}
```

```
$ mlir-opt --irfile=cmath.irdl.mlir example.cmath.mlir
```

```
irdl.dialect cmath {  
  
    irdl.type complex {  
        // ...  
    }  
  
    irdl.operation norm {  
        %0 = irdl.any  
        %1 = irdl.parametric @cmath::@complex(%0)  
  
        irdl.operands (in: %1)  
        irdl.results (res: %0)  
    }  
}
```

```
%0 = "cmath.norm" (%arg0)  
      : (!cmath.complex<f32>) -> f32
```

```
$ mlir-opt --irfile=cmath.irdl.mlir example.cmath.mlir
```

```
irdl.dialect cmath {  
  
    irdl.type complex {  
        // ...  
    }  
  
    irdl.operation norm {  
        %0 = irdl.any  
        %1 = irdl.parametric @cmath::@complex(%0)  
  
        irdl.operands (in: %1)  
        irdl.results (res: %0)  
    }  
}
```

```
%0 = "cmath.norm" (%arg0)  
: (!cmath.complex<f32>) -> f32
```

```
%0 = "cmath.norm" (%arg0)  
: (!cmath.complex<f32>) -> f64
```

✗ unsatisfied constraint

IRDL: IR Definition Language



Concise



Introspectable



Dynamic



Generatable

And it *just* works...

IRDL: IR Definition Language



Concise



Introspectable



Dynamic



Generatable

And it *just works... kind of*

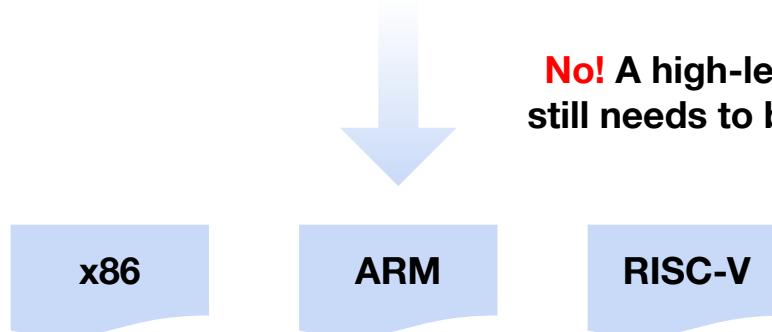
Is parsing this program enough?

```
%0 = "cmath.norm" (%arg0) : (!cmath.complex<f32>) -> f32
```

Is parsing this program enough?

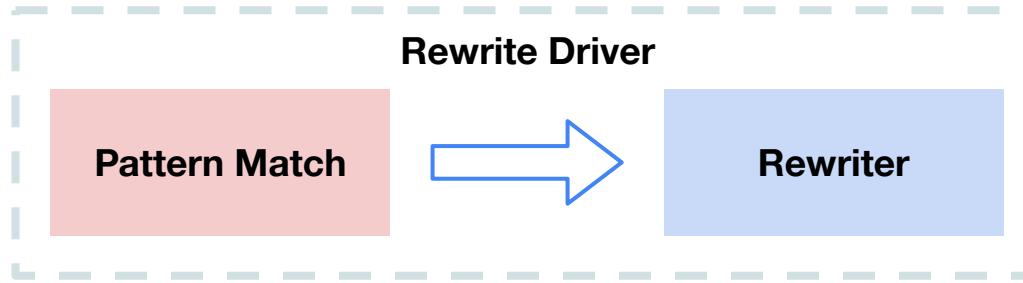
```
%0 = "cmath.norm" (%arg0) : (!cmath.complex<f32>) -> f32
```

No! A high-level dialect
still needs to be lowered!



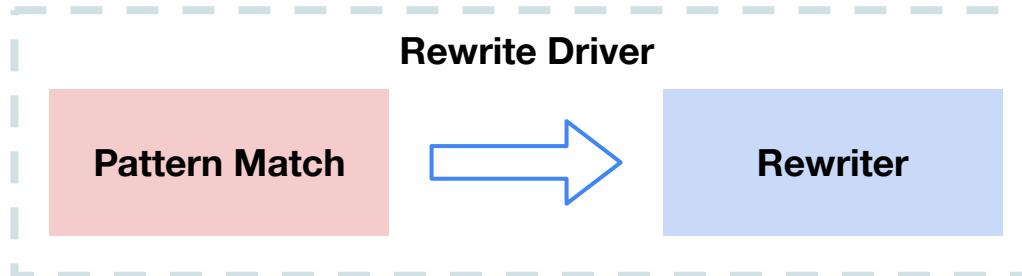
How to Rewrite IR in MLIR

(the abridged version)



How to Rewrite IR in MLIR

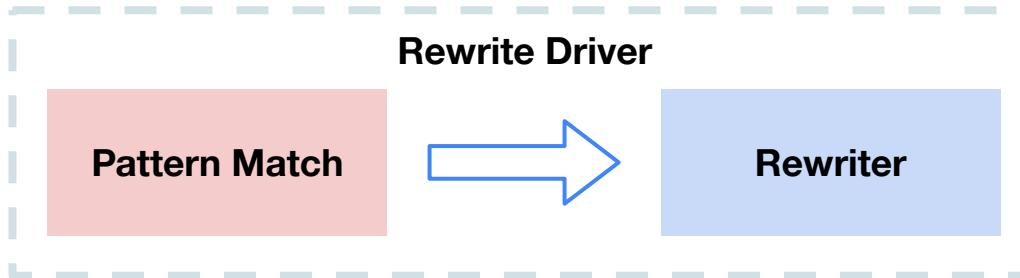
(the abridged version)



```
void matchAndRewrite(arith::AddFOp op, OpAdaptor op,
                     ConversionPatternRewriter& rewriter)
{
    // ...
    rewriter.create<arith::MulFOp>(loc, ...)
    rewriter.replaceWithNewOp<arith::MulFOp>(loc, op, ...)
}
```

How to Rewrite IR in MLIR

(the abridged version)



```
void matchAndRewrite(arith::AddFOp op, OpAdaptor op,
                     ConversionPatternRewriter& rewriter)
{
    // ...
    rewriter.create<arith::MulFOp>(loc, ...);
    rewriter.replaceWithNewOp<arith::MulFOp>(loc, op, ...);
}
```

MLIR infrastructure
uses **static C++ types!**

```
mlir-opt --irdl-file=cmath.irdl.mlir example.cmath.mlir
```

IRDL dialects are loaded **dynamically**

MLIR infrastructure
uses **static C++ types!**

```
mlir-opt --irdl-file=cmath.irdl.mlir example.cmath.mlir
```

IRDL dialects are loaded **dynamically**

**IRDL doesn't work with the
existing MLIR C++ infrastructure**

MLIR infrastructure
uses **static C++ types!**

```
mlir-opt --irdl-file=cmath.irdl.mlir example.cmath.mlir
```

IRDL dialects are loaded **dynamically**

IRDL  work with the
existing MLIR C++ infrastructure

MLIR infrastructure
uses **static C++ types!**

```
$ mlir-irdl-to-cpp example.irdl.mlir
```

↳ `mlir-ir-to-cpp example.1rlt.mlir`

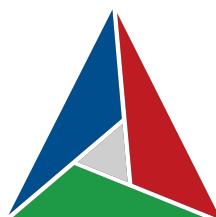
```
void matchAndRewrite(cmath::NormOp op, OpAdaptor op,
                     ConversionPatternRewriter& rewriter)
{
    // ...
    rewriter.create<cmath::RealOp>(loc, ...)
    rewriter.replaceWithNewOp<cmath::ImagOp>(loc, op, ...)
}
```



```
add_irdl_to_cpp_target(TestIRDLToCppGen test_irdl_to_cpp.irdl.mlir)
```

```
// CHECK: func.func @test() {
// CHECK: %[[v0:[^ ]*]] = "test_irdl_to_cpp.bar"() : () -> i32
// CHECK: %[[v1:[^ ]*]] = "test_irdl_to_cpp.bar"() : () -> i32
// CHECK: %[[v2:[^ ]*]] = "test_irdl_to_cpp.hash"(%[[v0]], %[[v0]]) : (i32, i32) -> i32
// CHECK: return
// CHECK: }
func.func @test() {
    %0 = "test_irdl_to_cpp.bar"() : () -> i32
    %1 = "test_irdl_to_cpp.beef"(%0, %0) : (i32, i32) -> i32
    return
}
```

[test_conversion.testd.mlir](#)



[MLIR][IRDL] Added IRDL to C++ Translation #133982

Merged?

[! Open](#) hhkit wants to merge 90 commits into [llvm:main](#) from [opencompl:hhkit/irdl-to-cpp](#) [Diff](#)

Conversation 31 · Commits 90 · Checks 7 · Files changed 39



hhkit commented last week · edited

Member

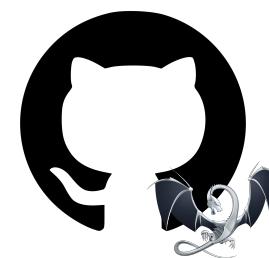
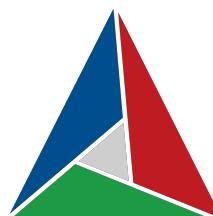
...

This PR introduces a new tool, `mlir-irdl-to-cpp`, that converts IRDL to C++ definitions.

The C++ definitions allow use of the IRDL-defined dialect in MLIR C++ infrastructure, enabling the use of conversion patterns with IRDL dialects for example. This PR also adds CMake utilities to easily integrate the IRDL dialects into MLIR projects.

Note that most IRDL features are not supported. In general, we are only able to define simple types and operations.

- The only type constraint supported is `irdl.any`.
- Variadic operands and results are not supported.
- Verifiers for the IRDL constraints are not generated.
- Attributes are not supported.



IRDL: IR Definition Language



Concise



Introspectable

```
irdl.dialect cmath {  
  
    irdl.type complex {  
        %0 = irdl.is f32,      %1 = irdl.is f64  
        %2 = irdl.any_of(%0, %1)  
  
        irdl.parameters (elem: %2)  
    }  
  
    irdl.operation norm {  
        %0 = irdl.any  
        %1 = irdl.parametric @cmath::@complex(%0)  
  
        irdl.operands (in: %1)  
        irdl.results (res: %0)  
    }  
}
```



Dynamic



Generatable

IRDL Devs



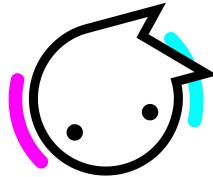
Mathieu Fehr

email: mathieufehr@gmail.com
github: math-fehr



Théo Degioanni

github: Moxinilian
discord: moxinilian



Ivan Ho

email: ivan@hhkit.dev
github: hhkit

Resources

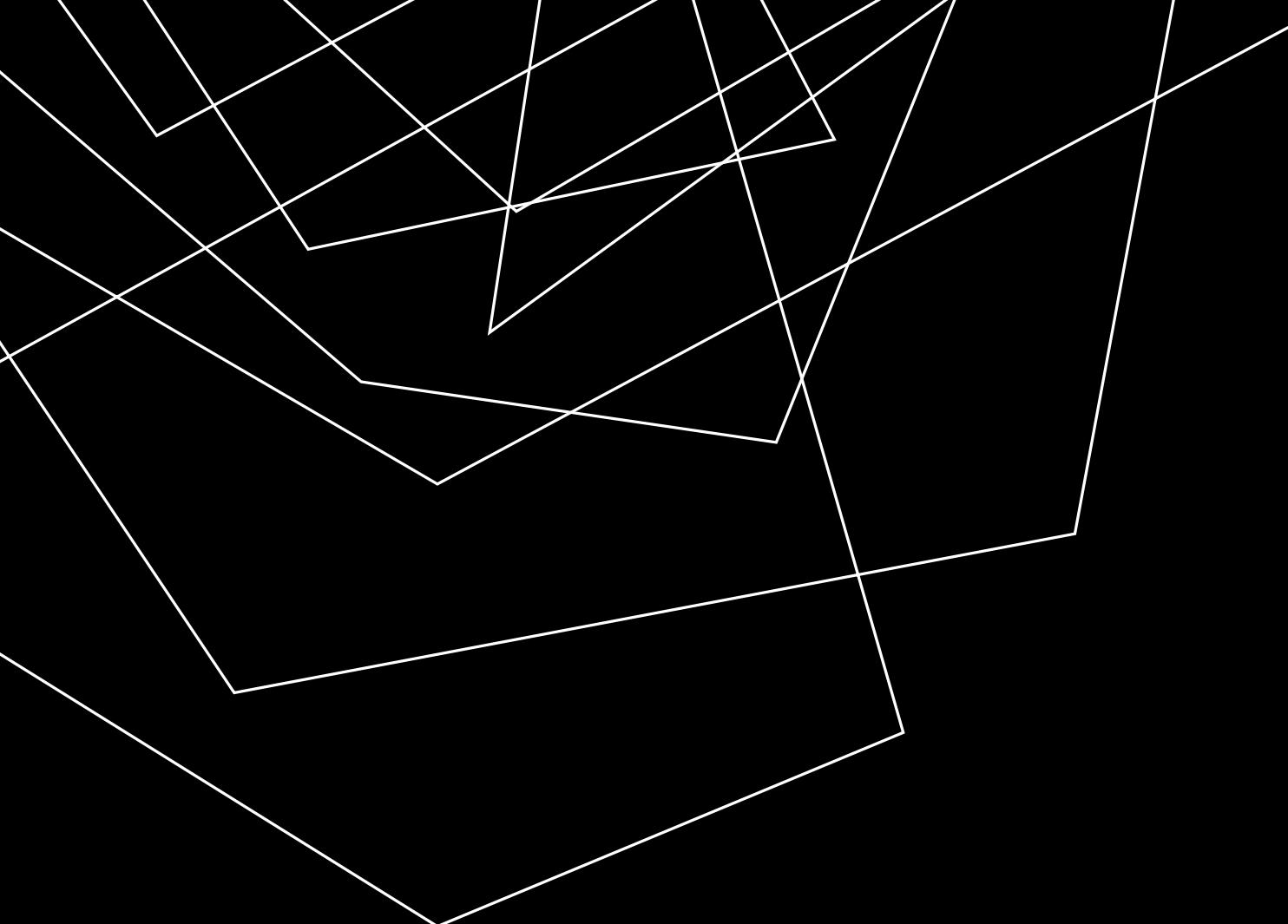


<https://godbolt.org/z/Ya54Whvd8>



Developers' Meeting

BERLIN 2025



AUTOCHECK
DJORDJE TODOROVIC
VLADIMIR VUKSANOVIC
PETAR JOVANOVIC
HTEC GROUP

ABOUT AUTOCHECK

- Static analysis tool for checking compliance with AUTOSAR/MISRA C++ Guidelines
- Command line tool
- Generate rule violation reports for a project provided
- IDE Integration (VSCode)
- AI Integration (llama.cpp)
- Presented at **2020 LLVM Developers' Meeting**
 - Extending Clang for Checking Compliance With Automotive Coding Standards - Milena Vujosevic Janicic - <https://www.youtube.com/watch?v=-6dL-7xkIV0>

AUTOSAR / MISRA C++ GUIDELINES

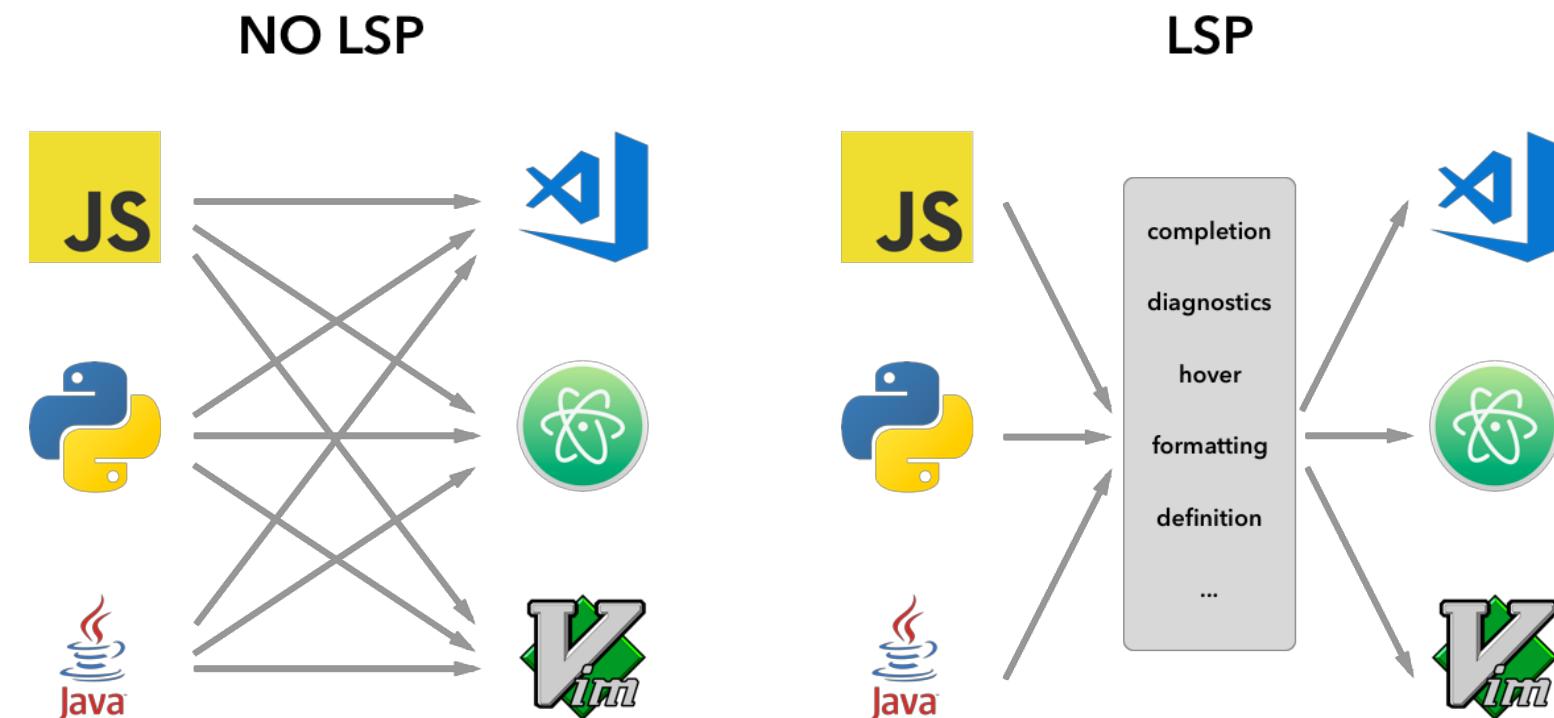
- Coding guidelines for using C++ in safety-related and critical systems
- Started as MISRA (Motor Industry Software Reliability Association) C++ in 2008
- Updated for newer C++ standards AUTOSAR (AUTOMOTIVE OPEN SYSTEM ARCHITECTURE) C++14
- Updated again as MISRA C++: 2023 for C++ 23

IMPLEMENTATION

- Based on the LLVM Compiler Infrastructure, like clang-tidy
 - libClang
- Works on multiple compilation stages:
 - Preprocessor
 - Lexer
 - Abstract syntax tree (AST)
 - AST Visitors
 - AST Matchers
 - Static Analyzer
 - LLVM Intermediate Representation (IR)

IDE INTEGRATION

- Language Server Protocol
- Supported by Visual Studio, VSCode, IntelliJ, vim, emacs, Qt Creator...



The screenshot shows a dark-themed code editor interface. At the top, a menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, Find, and Run.

The main workspace displays the content of `example.cpp`:

```
int main() {
    typedef long long ll;
    ll a = 0xff;
    return 0;
}
```

Below the code editor is a navigation bar with tabs: PROBLEMS (6), OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The PROBLEMS tab is active, showing a list of 6 errors:

- Violates rule [A7-1-6] Autocheck [Ln 2, Col 2]
- Violates rule [A3-9-1] Autocheck [Ln 2, Col 10]
- Violates rule [M0-1-3] Autocheck [Ln 3, Col 5]
- Violates rule [A8-5-2] Autocheck [Ln 3, Col 5]
- Violates rule [A7-1-1] Autocheck [Ln 3, Col 5]
- Violates rule [A2-13-5] Autocheck [Ln 3, Col 9]

On the left side, there are several small icons: a file with a blue dot, a magnifying glass, a gear, and a square with a cross. On the right side, there are icons for a terminal, settings, and other tools.

AI INTEGRATION

- Used for automatic code rewriting
- Local LLM instance using llama.cpp
- Llama.cpp enables inference of different LLMs (not just llama) locally either on a CPU, or GPU (CUDA, HIP)
- Used for rules where compiler technology cannot be applied – e.g. adding comments to functions that are missing it

CURRENT STATUS

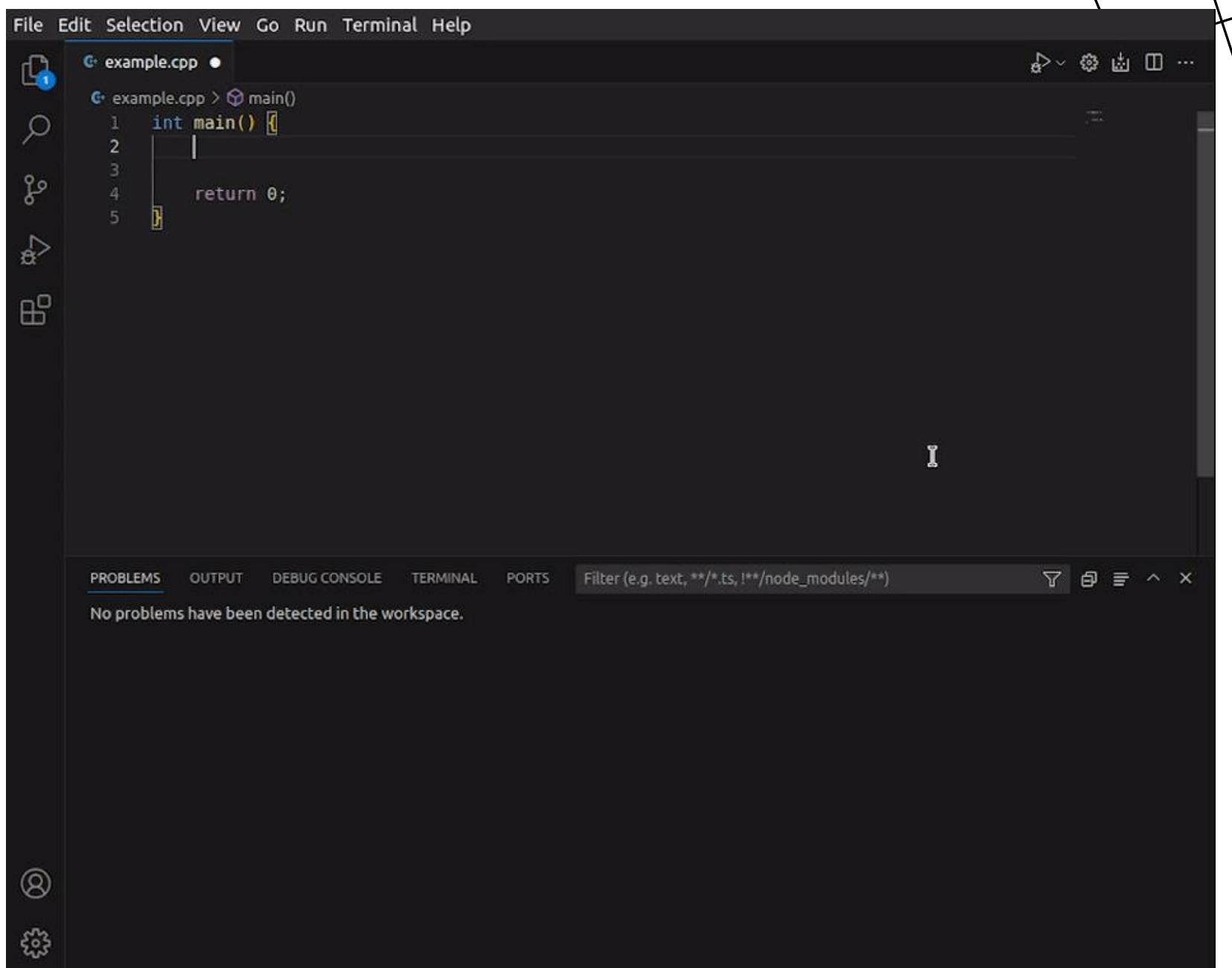
- Private repo
- Willing to open source it
 - <https://www.phoronix.com/news/Autocheck-LLVM-Safety-Critical>
 - <https://discourse.llvm.org/t/rfc-autocheck-a-source-code-analysis-tool-based-on-clang-llvm-for-automotive/76333>
- Problem with Licensees
 - MISRA/AUTOSAR
 - You have an advice?
 - Drop us a message at
 - djordje.todorovic@htecgroup.com
 - petar.jovanovic@htecgroup.com

```
#include <iostream>

int main() {
    int a = 0xff;
    std::cout << a << std::endl;

    return 0;
}
```

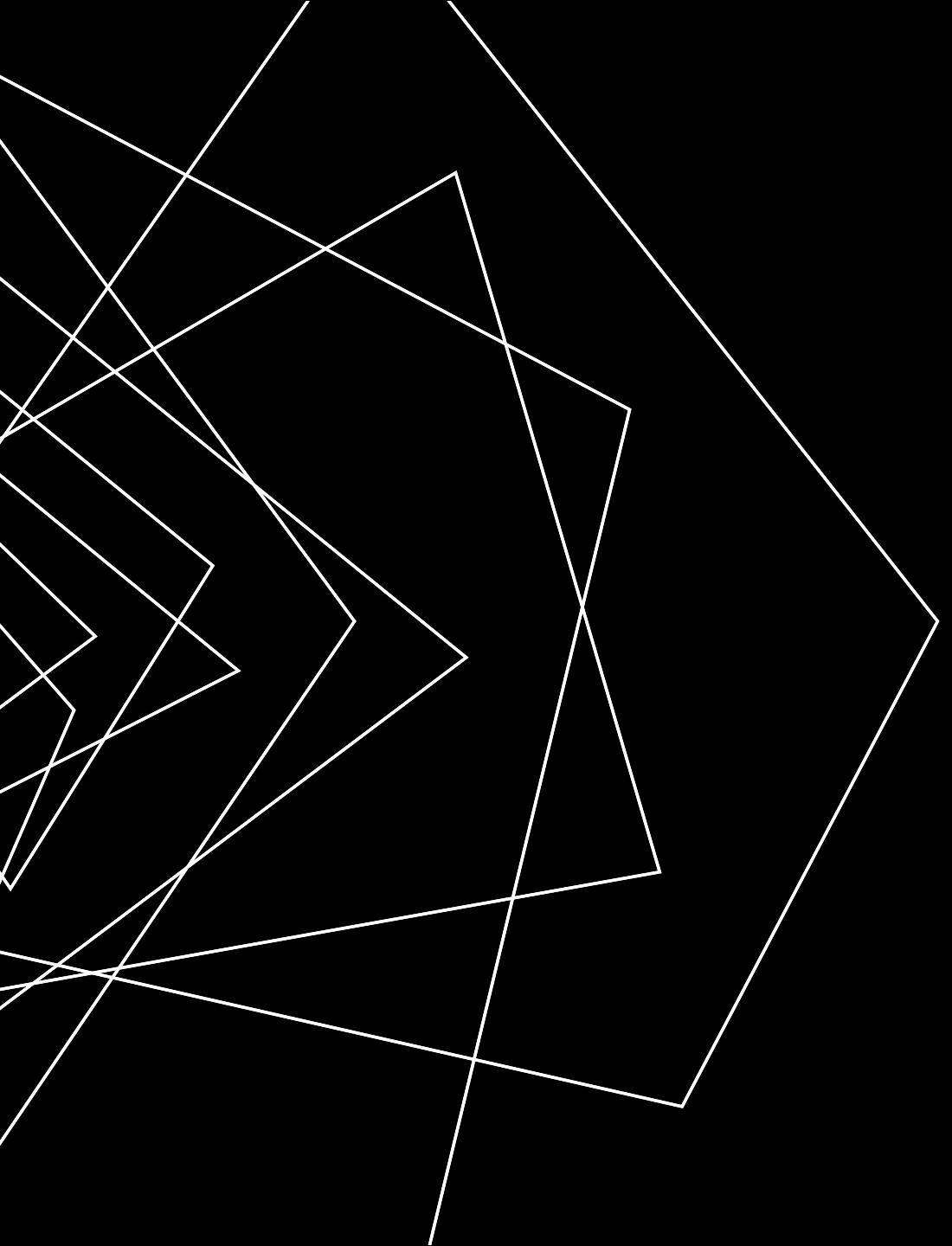
example.cpp 1,1 All



The screenshot shows a dark-themed code editor window. At the top, the menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. A sidebar on the left contains icons for file operations like Open, Save, Find, and others. The main editor area displays the following C++ code:

```
example.cpp
int main() {
    |
    return 0;
}
```

Below the editor is a status bar with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The PROBLEMS tab is active, showing the message "No problems have been detected in the workspace." There is also a "Filter (e.g. text, **/*.ts, !**/node_modules/**)" input field and some small icons.

An abstract geometric pattern is positioned on the left side of the image. It consists of numerous thin white lines forming various shapes, primarily triangles and chevrons, that radiate from the bottom-left corner towards the top-right. Some lines intersect, creating a complex web of geometric forms.

THANK YOU



Developers' Meeting

BERLIN 2025



LLDB Statusline

Jonas Devlieghere

EuroLLVM 2025



zsh ~

```
jonas@jonas-mac-studio build-debug % lldb ./bin/count
(lldb) target create "./bin/count"
Current executable set to '/Users/jonas/llvm/build-debug/bin/count' (arm64).
(lldb) b main
Breakpoint 1: where = count`main + 64 at count.c:24:7, address = 0x0000000100001558
(lldb) r
Process 59024 launched: '/Users/jonas/llvm/build-debug/bin/count' (arm64)
Process 59024 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
  frame #0: 0x0000000100001558 count`main(argc=1, argv=0x000000016fdfedb8) at count.c:24:7
21     size_t Count, NumLines, NumRead;
22     char Buffer[4096], *End;
23
→ 24     if (argc != 2) {
25         fprintf(stderr, "usage: %s <expected line count>\n", argv[0]);
26         return 2;
27     }
(lldb) █
```



zsh ~

```
jonas@jonas-mac-studio build-debug % lldb ./bin/count
(lldb) target create "./bin/count"
Current executable set to '/Users/jonas/llvm/build-debug/bin/count' (arm64).
(lldb) b main
Breakpoint 1: where = count`main + 64 at count.c:24:7, address = 0x0000000100001558
(lldb) r
Process 59024 launched: '/Users/jonas/llvm/build-debug/bin/count' (arm64)
Process 59024 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
  frame #0: 0x0000000100001558 count`main(argc=1, argv=0x000000016fdfedb8) at count.c:24:7
  21     size_t Count, NumLines, NumRead;
  22     char Buffer[4096], *End;
  23
→ 24     if (argc != 2) {
  25         fprintf(stderr, "usage: %s <expected line count>\n", argv[0]);
  26         return 2;
  27     }
(lldb) settings set show-statusline true
(lldb) █
```

```
count | count.c:24:7 | breakpoint 1.1
```

```
27  
(lldb) settings set show-statusline true  
(lldb) █
```

```
count | count.c:24:7 | breakpoint 1.1
```

```
(lldb) settings show statusline-format  
statusline-format (format-string) =  
"${ansi.negative}  
${target.file.basename}  
{ | ${line.file.basename}:${line.number}:${line.column}}  
{ | ${thread.stop-reason}}  
{ | ${progress.count} }${progress.message}"
```

```
27
(lldb) settings set show-statusline true
(lldb) settings set statusline-format "${ansi.bg.black}${ansi.fg.white} 🐛 ${target.file.basename}{ | ${line.file.basename}: ${line.number}: ${line.column}}{ | ${thread.stop-reason}}{ | ${progress.count} }${progress.message}""
(lldb) █
```

🐛 count | count.c:24:7 | breakpoint 1.1

(lldb) settings show statusline-format

statusline-format (format-string) =

"\${ansi.bg.black}\${ansi.fg.white} 🐛"

{\${target.file.basename}}

{ | \${line.file.basename}: \${line.number}: \${line.column}}

{ | \${thread.stop-reason}}

{ | \${progress.count} }\${progress.message}"

```
27
(lldb) settings set show-statusline true
(lldb) settings set statusline-format "${ansi.bg.black}${ansi.fg.white} 🐛 ${target.file.basename}{ | ${line.file.basename}:${line.number}:${line.column}}{ | ${thread.stop-reason}}{ | ${progress.count} }${progress.message}""
(lldb) █
```

🐛 count | count.c:24:7 | breakpoint 1.1

(lldb) settings show statusline-format

statusline-format (format-string) =

"\${ansi.bg.black}\${ansi.fg.white} 🐛"

{\${target.file.basename}}

{ | \${line.file.basename}:\${line.number}:\${line.column}}

{ | \${thread.stop-reason}}

{ | \${progress.count} }\${progress.message}"

```
27  
(lldb) settings set show-statusline true  
(lldb) settings set statusline-format "${ansi.bg.black}${ansi.fg.white} 🐛 ${target.file.basename}{ | ${line.file.basename}: ${line.number}: ${line.column}}{ | ${thread.stop-reason}}{ | ${progress.count} }${progress.message}"  
(lldb) █
```

🐛 count | count.c:24:7 | breakpoint 1.1

(lldb) settings show statusline-format

statusline-format (format-string) =

"\${ansi.bg.black}\${ansi.fg.white} 🐛"

{\${target.file.basename}}

{ | \${line.file.basename}: \${line.number}: \${line.column}}

{ | \${thread.stop-reason}}

{ | \${progress.count} }\${progress.message}"

```
27
(lldb) settings set show-statusline true
(lldb) settings set statusline-format "${ansi.bg.black}${ansi.fg.white} 🐛 ${target.file.basename}{ | ${line.file.basename}: ${line.number}: ${line.column}}{ | ${thread.stop-reason}}{ | ${progress.count} }${progress.message}""
(lldb) █
```

🐛 count | count.c:24:7 | breakpoint 1.1

(lldb) settings show statusline-format

statusline-format (format-string) =

"\${ansi.bg.black}\${ansi.fg.white} 🐛"

{\${target.file.basename}}

{ | \${line.file.basename}: \${line.number}: \${line.column}}

{ | \${thread.stop-reason}}

{ | \${progress.count} }\${progress.message}}"

```
27  
(lldb) settings set show-statusline true  
(lldb) settings set statusline-format "${ansi.bg.black}${ansi.fg.white} 🐛 ${target.file.basename}{ | ${line.file.basename}:${line.number}:${line.column}}{ | ${thread.stop-reason}}{ | ${progress.count} ${progress.message}}"  
(lldb) add-dsym -S
```

```
🐛 count | count.c:24:7 | breakpoint 1.1 | Downloading symbol file for: dyld (84E695DB-7E4B-34BF-9B0D-2C7EF0013D9E)
```

```
(lldb) settings show statusline-format  
statusline-format (format-string) =  
"${ansi.bg.black}${ansi.fg.white}  
${target.file.basename}  
{ | ${line.file.basename}:${line.number}:${line.column}}  
{ | ${thread.stop-reason}}  
{ | ${progress.count} ${progress.message}}"
```

Format Strings

Reference on lldb.llvm.org

Variable Name	Description
<code>file.basename</code>	The current compile unit file basename for the current frame.
<code>file.fullpath</code>	The current compile unit file fullpath for the current frame.
<code>language</code>	The current compile unit language for the current frame.
<code>frame.index</code>	The frame index (0, 1, 2, 3...)
<code>frame.no-debug</code>	Evaluates to true if the frame has no debug info.
<code>frame.pc</code>	The generic frame register for the program counter.
<code>frame.sp</code>	The generic frame register for the stack pointer.
<code>frame.fp</code>	The generic frame register for the frame pointer.
<code>frame.flags</code>	The generic frame register for the flags register.
<code>frame.reg.NAME</code>	Access to any platform specific register by name (replace <code>NAME</code> with the name of the desired register).
<code>function.name</code>	The name of the current function or symbol.
<code>function.name-with-args</code>	The name of the current function with arguments and values or the symbol name.
<code>function.name-without-args</code>	The name of the current function without arguments and values (used to include a function name in-line in the <code>disassembly-format</code>)
<code>function.mangled-name</code>	The mangled name of the current function or symbol.
<code>function.pc-offset</code>	The program counter offset within the current function or symbol
<code>function.addr-offset</code>	The offset in bytes of the current function, formatted as " + dddd"

ON THIS PAGE

[Format Strings](#)

[Variables](#)

[Control Characters](#)

[Desensitizing Characters in the Format String](#)

[Scoping](#)

[Making the Frame Format](#)

[Making Your own Formats](#)



Motivation

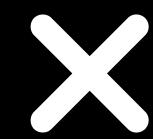
Progress Events

- Inline progress events are fragile
- Holding back new progress events

Customization

- Users like to customize lldb
- Popular request is to configure the prompt

Implementation



ncurses

- Efficient and established way to create textual interfaces
- Needs to clear the screen to control it



ANSI escape codes

- How we implement colors
- multiline editing
- Escape sequence to adjust the scroll window

Future Work

Extend the format strings

- Support more format variables
- Default values
- Alignment & padding



TM and © 2025 Apple Inc. All rights reserved.



Developers' Meeting

BERLIN 2025



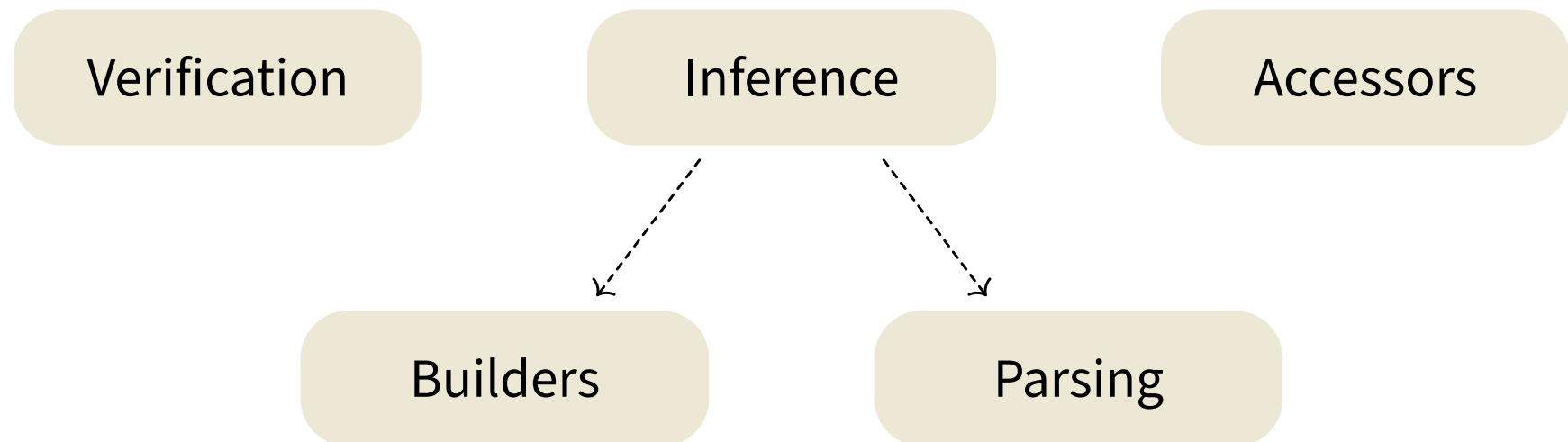
THE UNIVERSITY
of EDINBURGH

Defining and Verifying MLIR Operations with Constraints

Alex Rice

EuroLLVM 2025

What do operation definitions do?



Example: arith.addi

```
%0 = arith.addi %1, %2 : i32
```

Example: arith.addi

```
%0 = arith.addi %1, %2 : i32
```

Constraints cannot be defined independently:

```
"arith.addi"(%1, %2) : (i32, i64) -> i8
```

Example: arith.addi

```
%0 = arith.addi %1, %2 : i32
```

Constraints cannot be defined independently:

"arith.addi"(%1, %2 : (i32, i64) -> i8



Constraints for arith.addi

```
class AddIOp:  
    _T: ClassVar = VarConstraint("T", signlessIntegerLike)  
    lhs = operand_def(_T)  
    rhs = operand_def(_T)  
    result = result_def(_T)  
  
    assembly_format = "$lhs ` ,` $rhs attr-dict ` :` type($result)"
```

Towards more complex constraints

```
class InsertOp:  
    name = "vector.insert"  
  
    _T: ClassVar = VarConstraint("T", AnyAttr())  
    _V: ClassVar = VarConstraint("V", VectorType.constr(_T))  
  
    source = operand_def(VectorType.constr(_T) | _T)  
    dest   = operand_def(_V)  
    result = result_def(_V)  
    ...
```



Developers' Meeting

BERLIN 2025



Arm Solutions at Lightspeed

Small Changes, Big Impact: GitHub Workflows for the LLVM Project

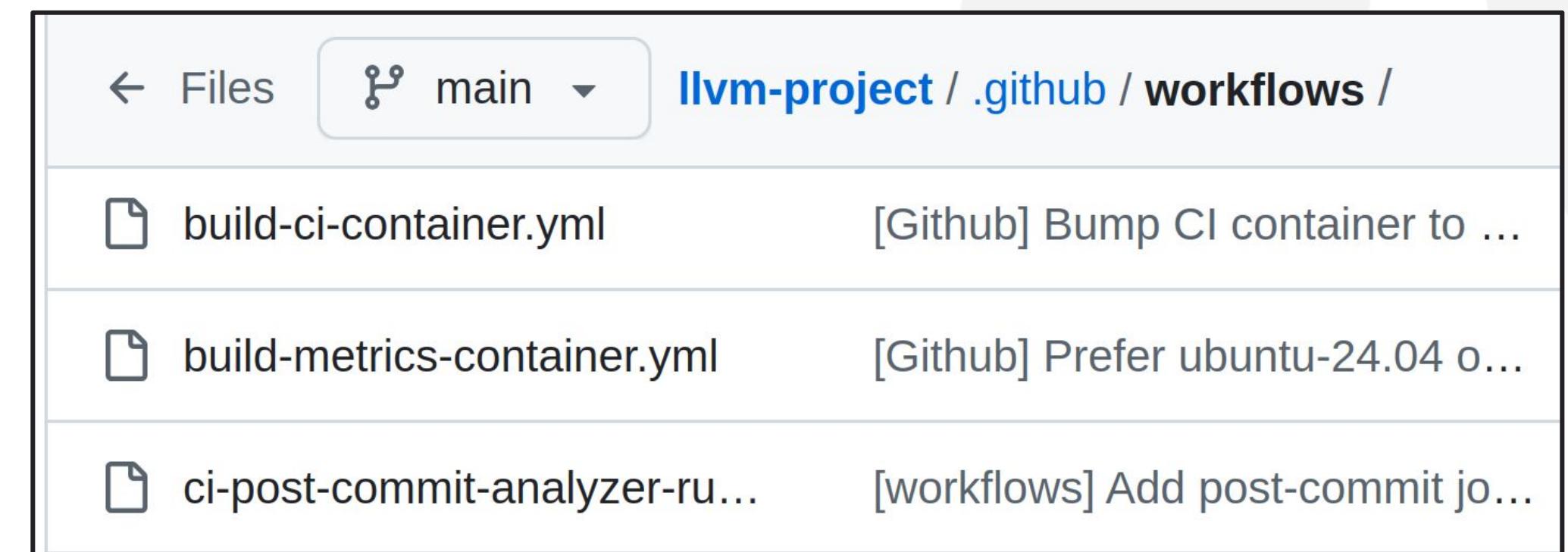
David Spickett, Arm / Linaro

FYI

- This is quite high level.
- There will be more text than you can read.
- I will talk about the most important bits.
- Come back to the slides later if you find this interesting.

GitHub Actions Workflows

- Automate tasks for a GitHub repository
- “workflows” define what to do and when
- Run on remote “Runners” provided by GitHub
- Workflows defined in YAML
- [.github/workflows](#) in llvm-project



A screenshot of a GitHub repository interface. The URL in the address bar is "llvm-project/.github/workflows/". The page shows three files in the ".github/workflows" directory:

File	Description
build-ci-container.yml	[Github] Bump CI container to ...
build-metrics-container.yml	[Github] Prefer ubuntu-24.04 o...
ci-post-commit-analyzer-ru...	[workflows] Add post-commit jo...

LLVM Workflows

If you have contributed to LLVM, you have seen a workflow in action.

- Pull Request (PR) Labelling
- Checking code formatting
- Backporting to releases
- ...more examples later...

Actions [New workflow](#)

All workflows

Build and Test libc++	
Build CI Container	
HLSL Tests	
Add buildbot information t...	
AI Code Reviewer	
Build Docker images for li...	
Build Metrics Container	
Build Windows CI Contai...	
Check code formatting	
Check for private emails ...	
Check libc++ generated fi...	
CI Tests	

YAML Example

What can it access?



When should it run?



What should it run?



Where should it run?



What work should it do?

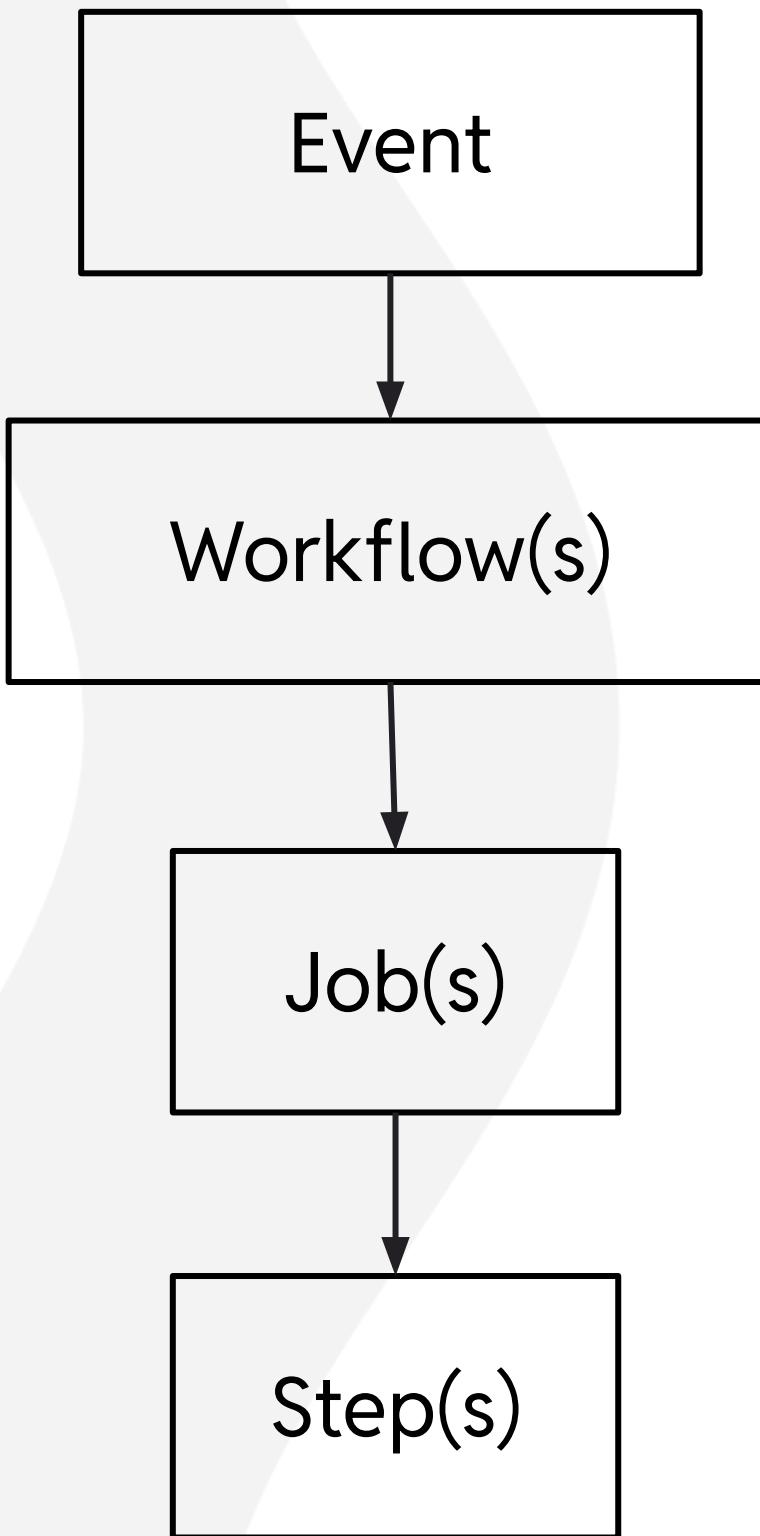


```
name: "Test documentation build"

permissions:
  contents: read

on:
  push:

jobs:
  check-docs-build:
    name: "Test documentation build"
    runs-on: ubuntu-24.04
    steps:
      - name: Fetch LLVM sources
```



So You Want to Write a Workflow

You need:

- A Github account
- Patience

No special membership required, even the Runners are free to everyone*.

* <https://docs.github.com/en/billing/managing-billing-for-your-products/managing-billing-for-github-actions/about-billing-for-github-actions#included-storage-and-minutes>

Fork LLVM



Fork

12.9k



Workflows and Runners will be available by default.

You will need enable the existing workflows from the “Actions” tab:



Workflows aren't being run on this forked repository

Because this repository contained workflow files when it was forked, we have disabled them from running on this fork. Make sure you understand the configured workflows and their expected usage before enabling Actions on this repository.

[I understand my workflows, go ahead and enable them](#)

[View the workflows directory](#)

LLVM's workflows have repository name checks as well, more on this later.

Starting Points

- YAML is tricky, GitHub's errors make it worse.
- Copy an existing workflow.

Look for similar:

- Trigger Events (push, new PR, new issue, ...)
- Use Case (checking code, managing labels, inspecting builds, ...)
- Results (writing a comment, creating a file, adding a label, ...)

For example...

New PR Workflow - new-prs.yml

- Runs when a new PR is opened.
- Labels it based on the changes.

on:

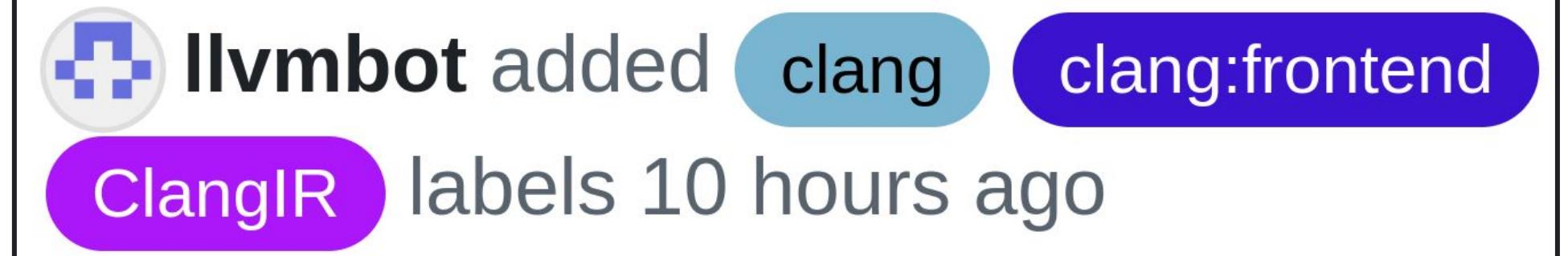
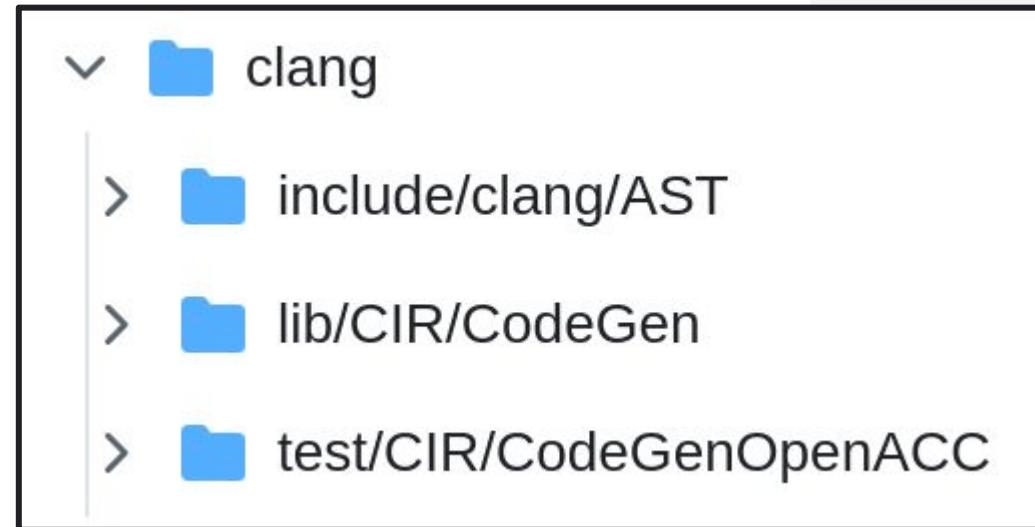
pull_request_target:

types:

- opened
- reopened
- < . . . >

steps:

- uses: actions/labeller



New PR Workflow - new-prs.yml

- Greet new contributors.



```
- name: Greet Author  
run:  
  python3 ./github-automation.py <...> pr-greeter <...>
```

`github-automation.py` - a collection of uses of the GitHub API.

- Specific to LLVM
- Built on [PyGitHub](#)

Enable On Your Fork

- Workflow files may have extra checks:

```
if: github.repository == 'llvm/llvm-project'
```

- Change to your username, change back when submitting to LLVM.

```
if: github.repository == 'your-username/llvm-project'
```

(you only need to do this for the workflow you are editing)

(and no, the UI [does not tell](#) you [why](#) it was skipped, that would be far too useful)



This job was skipped

Write the rest of the workflow



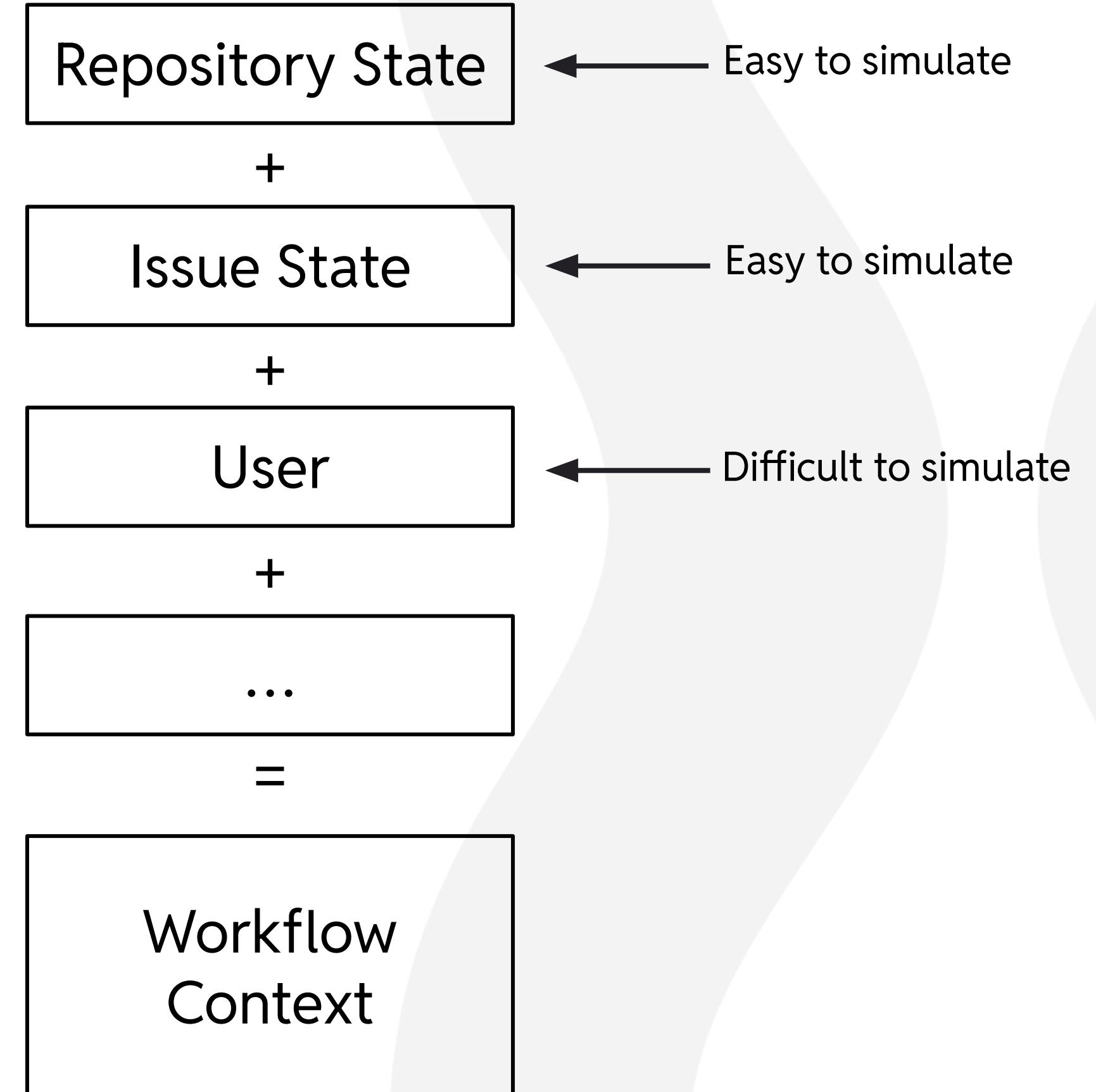
- Use a YAML aware editor.
- Edit in small chunks and commit often.
- Put complex logic into script steps or script files, instead of YAML.*
- Use the GitHub [documentation](#).
- GitHub Code Search to find examples.+

* Unless doing it in YAML saves a lot of compute time.

+ You might be the first of 100M users to notice that something is [broken](#).

Testing

- GitHub Runners are not under our control.
- No local testing.
- No test framework.
- Cannot simulate all conditions.



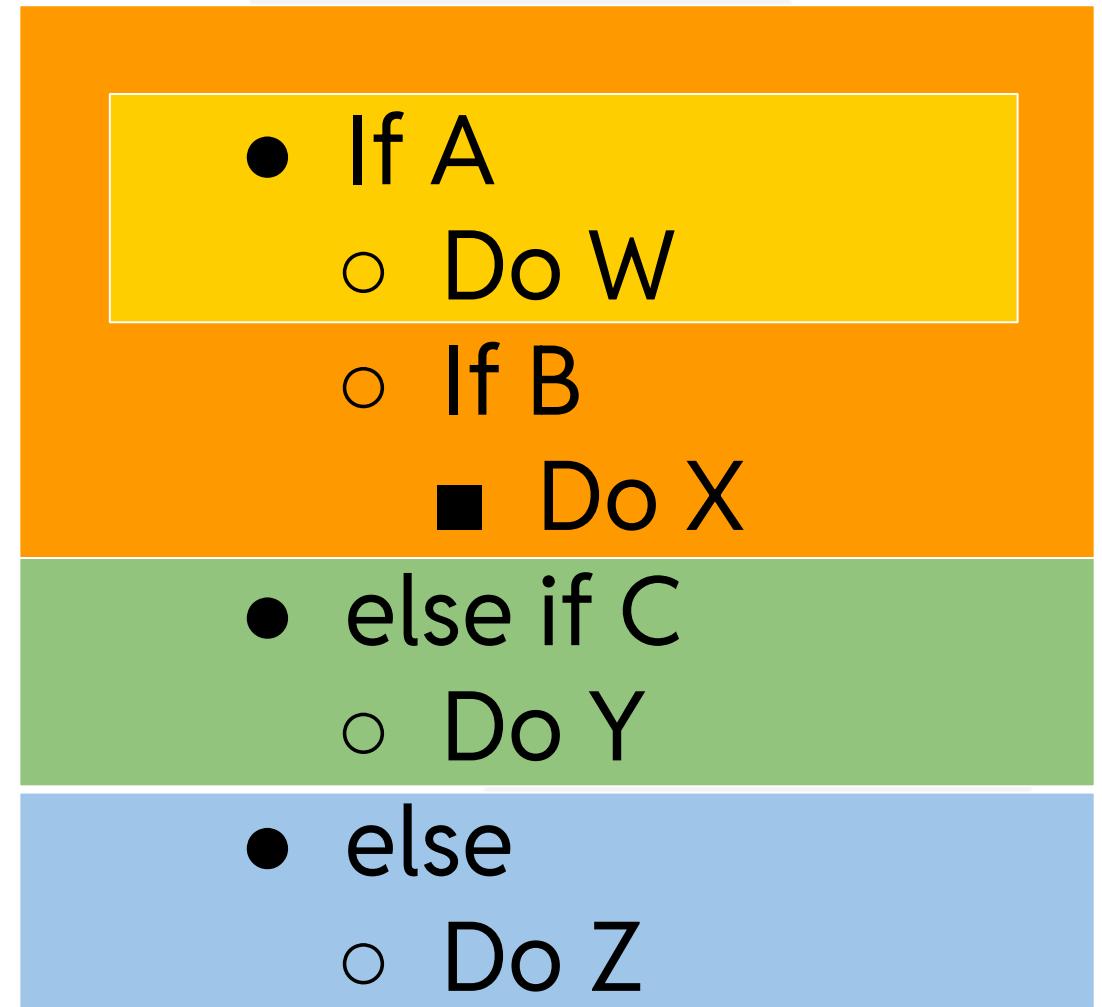
Testing

Solution:

- Manual testing.
- Cannot change all inputs but -
- Can temporarily change the code, **as if** we had changed the inputs.

Testing - Scenarios

- Write out all the scenarios that you need to handle
- Split scenarios into parts if needed
- Make changes to simulate one part
- Test
- Reset changes
- Repeat for all parts



(each colour is one test run)

Testing - New PR Example

Scenario 1: New PR from a new user

Expected: Labels and greeting comment added

Test with:

```
if: >-
(github.repository == 'you/llvm-project') &&
(github.event.action == 'opened')
```

- No check (as if user is new)
- **Do not contribute this.**

Scenario 2: New PR from an existing user

Expected: Labels added

Test with:

```
if: >-
(github.repository == 'you/llvm-project') &&
(github.event.action == 'opened') &&
(github.event.pull_request.author_association != 'COLLABORATOR') &&
<...>
```

- Does the check.
- **Do contribute this.**

Scenario 1 + Scenario 2 = Full Coverage*

* As much as we can hope for given we are not “testing what we ship”

Contributing to LLVM

- Save a copy of the branch, in case you need the history.
- Remove testing hacks, skips and manual triggers.
- Reset “repo==” checks to “**llvm/llvm-project**”.
- Squash into one commit.
- [Send a PR](#) to LLVM as you would for any other change.

Labels

github:workflow

Ideas

Existing

- Labelling PRs
- Easy backports
- Formatting checks
- Managing commit access

In progress*

- Pre-merge testing⁺
- Clean up user branches
- Find someone to merge a PR for you

Future?

- Per-file [reviewer notes](#)
- Unusual builds on demand
- Recommended tests
- Domain specific linters
- ...

* As of 2025-02-04

+ Pre-merge testing is moving from Buildkite to GitHub

Conclusions

- You all have the ability to edit and create your own workflows!
- Make life easier for 100s of contributors every day.



linaro.org

Thank you

“Writing a new workflow” workflow

- Fork llvm-project
- Copy a YAML file with a similar use case
- Enable the workflow on your fork
- Write the rest of the workflow
- Push to your fork’s main branch
- Trigger the Workflow
- See the result in the “Actions” tab
- Fix mistakes, push, test, repeat until working.
- Squash
- Send PR to LLVM



Developers' Meeting

BERLIN 2025