Developers' Meeting

BERLIN 2025

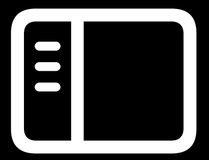
What is LLDB-DAP?

Jonas Devlieghere

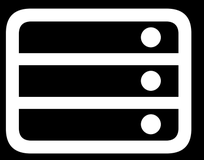
EuroLLVM 2025

Debug Adapter Protocol

Debug Adapter Protocol



client

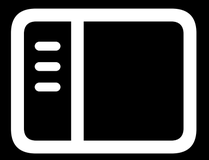


lldb-dap

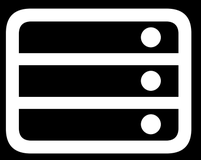
```
{
  "command": "breakpointLocations",
  "arguments": {
    "source": {
      "name": "Driver.cpp",
      "path": "/path/to/Driver.cpp"
    },
    "line": 780
  },
  "type": "request",
  "seq": 3
}
```

https://microsoft.github.io/debug-adapter-protocol/specification#Requests_BreakpointLocations

Debug Adapter Protocol



client

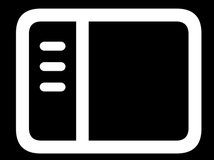


lldb-dap

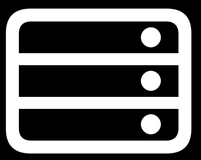
```
{
  "command": "breakpointLocations",
  "arguments": {
    "source": {
      "name": "Driver.cpp",
      "path": "/path/to/Driver.cpp"
    },
    "line": 780
  },
  "type": "request",
  "seq": 3
}
```

https://microsoft.github.io/debug-adapter-protocol/specification#Requests_BreakpointLocations

Debug Adapter Protocol



client



lldb-dap


```
{
  "body": {
    "breakpoints": [
      {
        "column": 19,
        "line": 780
      }
    ]
  },
  "command": "breakpointLocations",
  "request_seq": 3,
  "seq": 1,
  "success": true,
  "type": "response"
}
```


https://microsoft.github.io/debug-adapter-protocol/specification#Requests_BreakpointLocations

DAP in LLDB

DAP in LLDB

lldb-dap

The DAP server

LLDB-DAP

Visual Studio Code Extension

lldb-dap

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

LLDB-DAP

The Visual Studio Code Extension

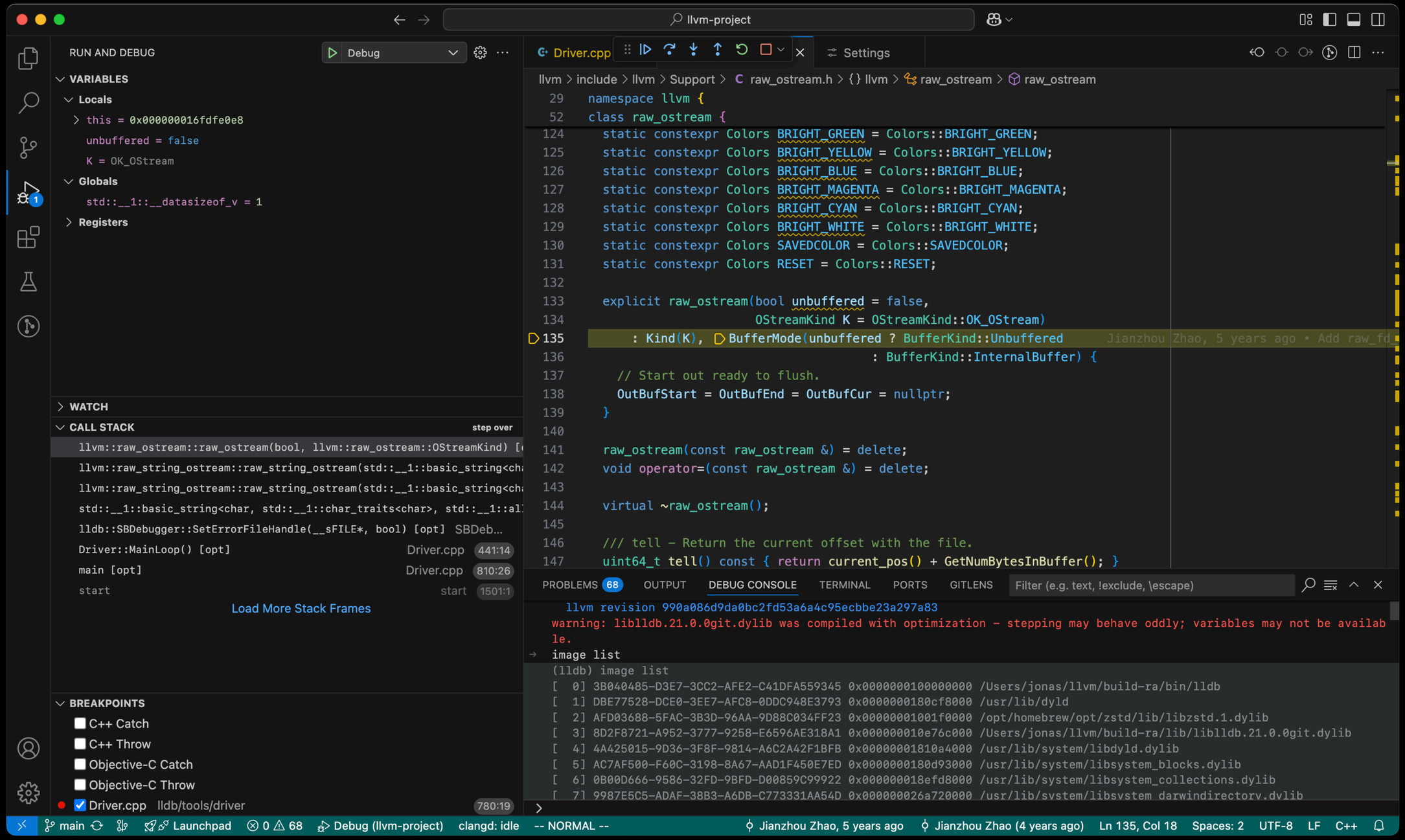
Visual Studio Code extension

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

Integration into VS Code

- Settings UI
- Launch configuration templates

https://marketplace.visualstudio.com/items?itemName=llvm-vs-code-extensions.lldb-dap


llvm-project
RUN AND DEBUG
Debug
VARIABLES
Locals
this = 0x000000016fdfe0e8
unbuffered = false
K = OK_OStream
Globals
std::__1::__datasizeof_v = 1
Registers
WATCH
CALL STACK
step over
llvm::raw_ostream::raw_ostream(bool, llvm::raw_ostream::OStreamKind) [
llvm::raw_string_ostream::raw_string_ostream(std::__1::basic_string<cha
llvm::raw_string_ostream::raw_string_ostream(std::__1::basic_string<cha
std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::al
lldb::SBDebugger::SetErrorFileHandle(__sFILE*, bool) [opt] SBDeb...
Driver::MainLoop() [opt] Driver.cpp 441:14
main [opt] Driver.cpp 810:26
start start 1501:1
Load More Stack Frames
BREAKPOINTS
C++ Catch
C++ Throw
Objective-C Catch
Objective-C Throw
Driver.cpp lldb/tools/driver 780:19
Driver.cpp
Settings
llvm > include > llvm > Support > raw_ostream.h > { } llvm > raw_ostream > raw_ostream
29 namespace llvm {
52 class raw_ostream {
124 static constexpr Colors BRIGHT_GREEN = Colors::BRIGHT_GREEN;
125 static constexpr Colors BRIGHT_YELLOW = Colors::BRIGHT_YELLOW;
126 static constexpr Colors BRIGHT_BLUE = Colors::BRIGHT_BLUE;
127 static constexpr Colors BRIGHT_MAGENTA = Colors::BRIGHT_MAGENTA;
128 static constexpr Colors BRIGHT_CYAN = Colors::BRIGHT_CYAN;
129 static constexpr Colors BRIGHT_WHITE = Colors::BRIGHT_WHITE;
130 static constexpr Colors SAVEDCOLOR = Colors::SAVEDCOLOR;
131 static constexpr Colors RESET = Colors::RESET;
132
133 explicit raw_ostream(bool unbuffered = false,
134 OStreamKind K = OStreamKind::OK_OStream)
135 : Kind(K), BufferMode(unbuffered ? BufferKind::Unbuffered Jianzhou Zhao, 5 years ago • Add raw_fd
136 : BufferKind::InternalBuffer) {
137 // Start out ready to flush.
138 OutBufStart = OutBufEnd = OutBufCur = nullptr;
139 }
140
141 raw_ostream(const raw_ostream &) = delete;
142 void operator=(const raw_ostream &) = delete;
143
144 virtual ~raw_ostream();
145
146 /// tell - Return the current offset with the file.
147 uint64_t tell() const { return current_pos() + GetNumBytesInBuffer(); }
PROBLEMS 68
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS
Filter (e.g. text, !exclude, \escape)
llvm revision 990a086d9da0bc2fd53a6a4c95ecbbe23a297a83
warning: liblldb.21.0.0git.dylib was compiled with optimization - stepping may behave oddly; variables may not be availab
le.
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-9D88C034FF23 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
main
Launchpad
0 ⚠ 68
Debug (llvm-project)
clangd: idle
-- NORMAL --
Jianzhou Zhao, 5 years ago
Jianzhou Zhao (4 years ago)
Ln 135, Col 18
Spaces: 2
UTF-8
LF
C++

Developers' Meeting

BERLIN 2025

THE UNIVERSITY *of* EDINBURGH

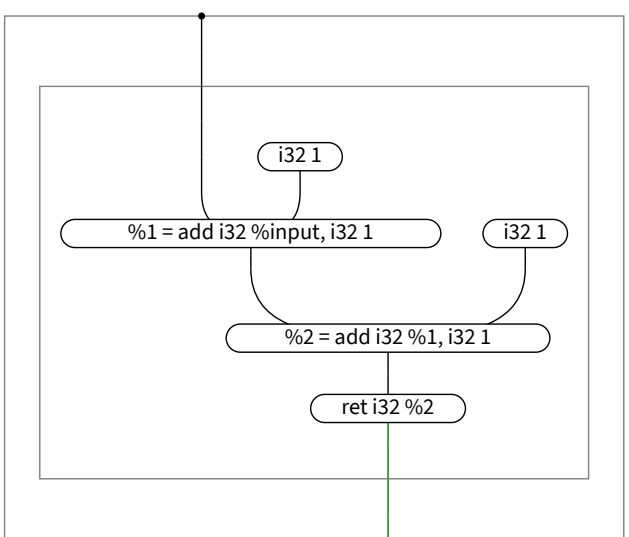
sd-visualiser: Interactive graph visualisation for SSA-based IRs

Alex Rice, Nick Hu, Calin Tataru

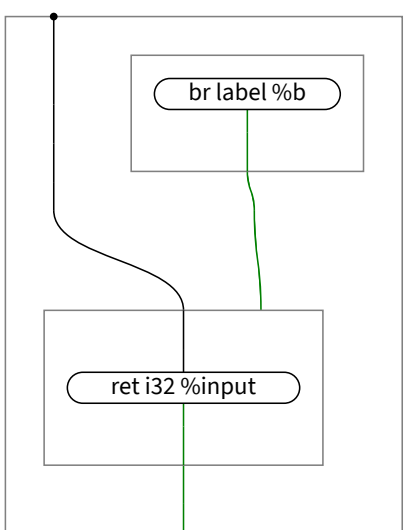
EuroLLVM 2025

Intermediate representations as a graph

Data flow

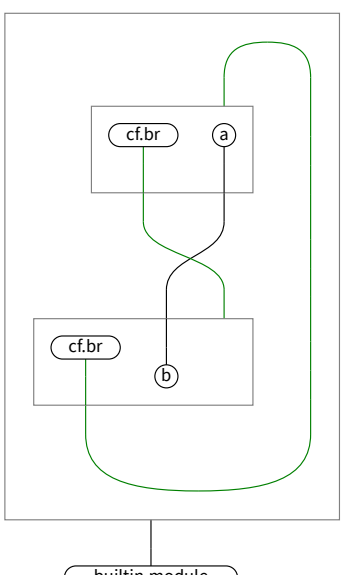


Control flow

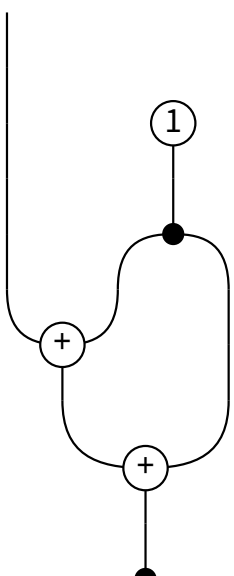


Representing IR features

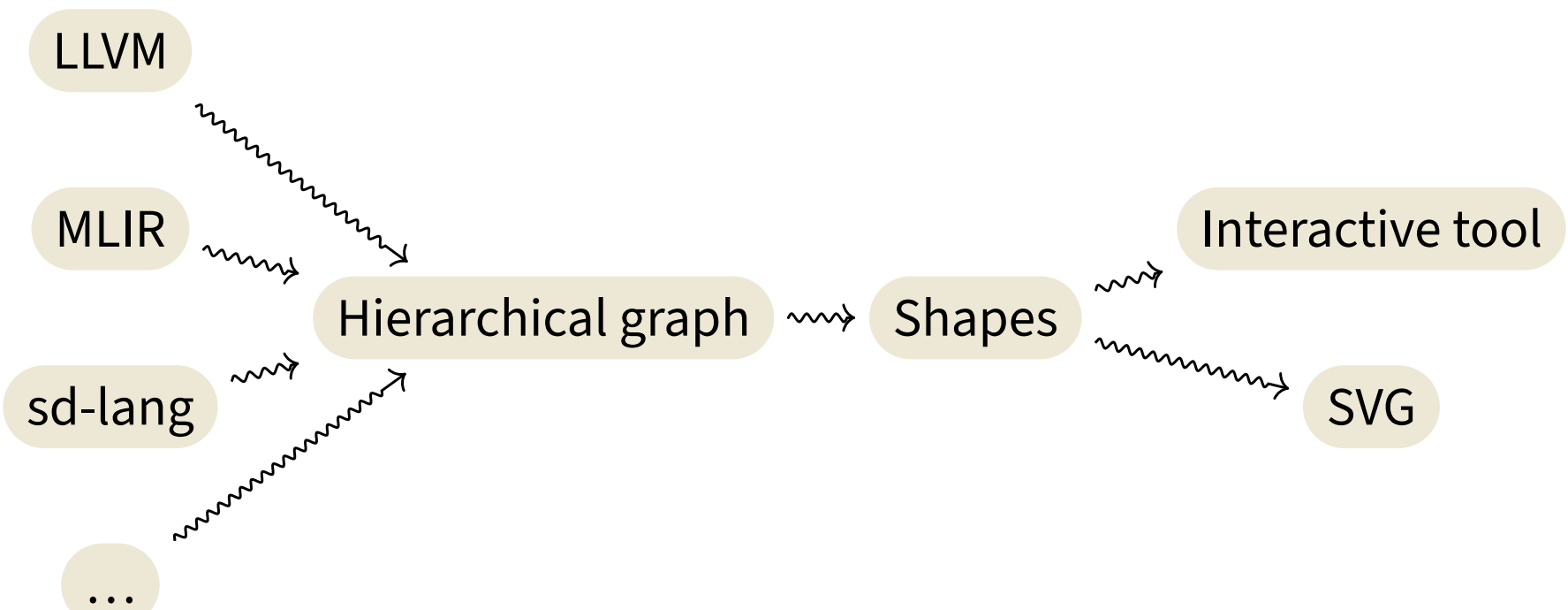
Blocks and regions



Copying and deletion

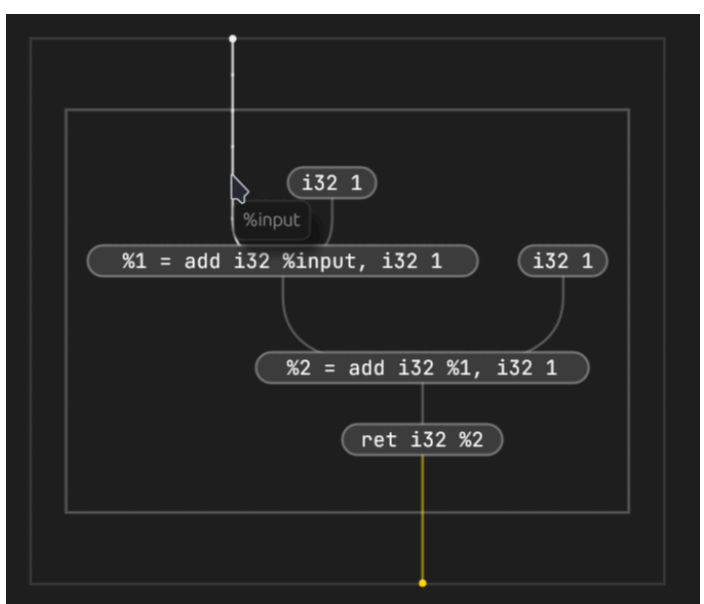


Visualising IRs

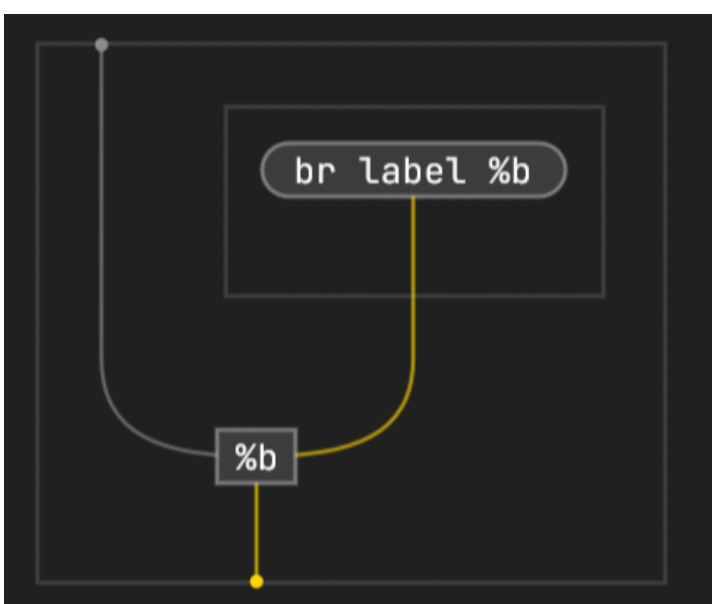


Interactive features

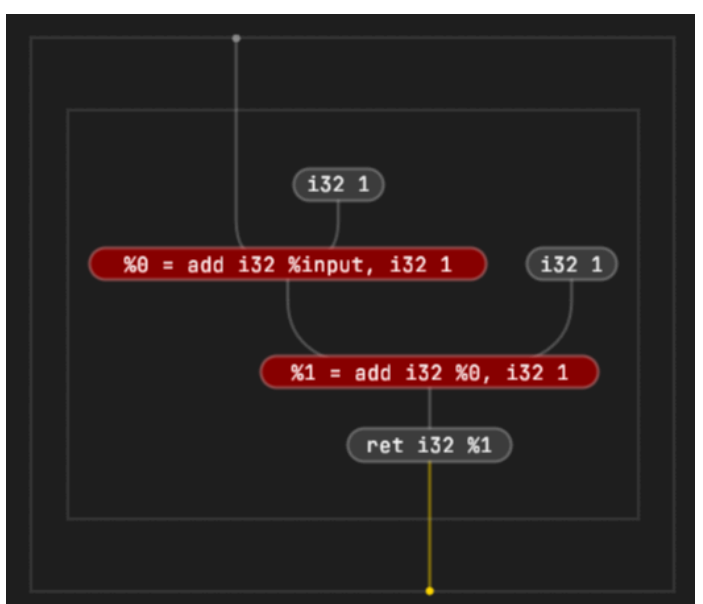
Highlighting



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 Cheng**[1] (chenglong86@huawei.com)

Co-authors: Lu Li[2], Zhongchun Zheng[1, 5], Rodrigo Caetano de Oliveira Rocha[3], Wei Wei[1], Tianyi Liu[4], Li Zhou[1]

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

HUAWEI

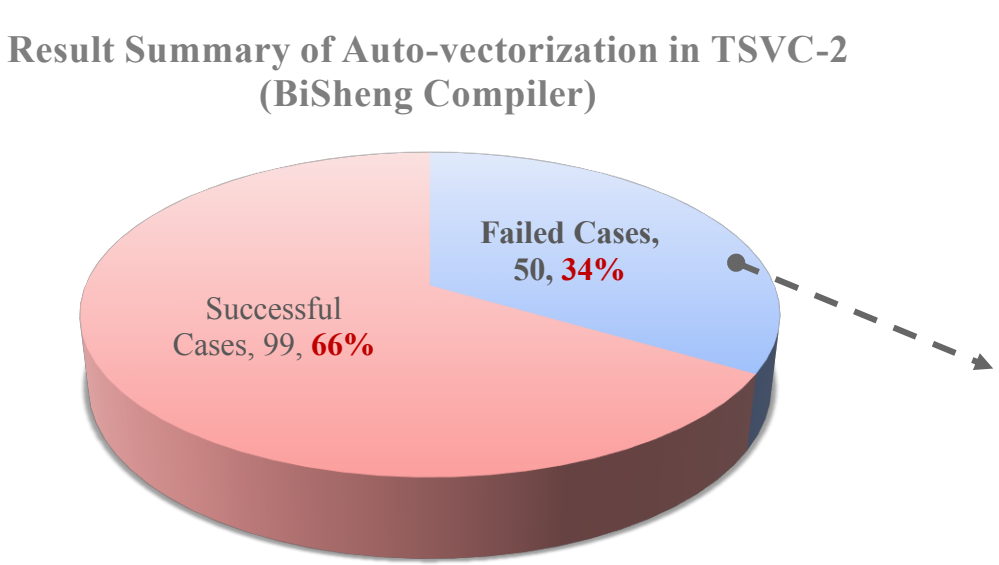
SUN YAT-SEN UNIVERSITY

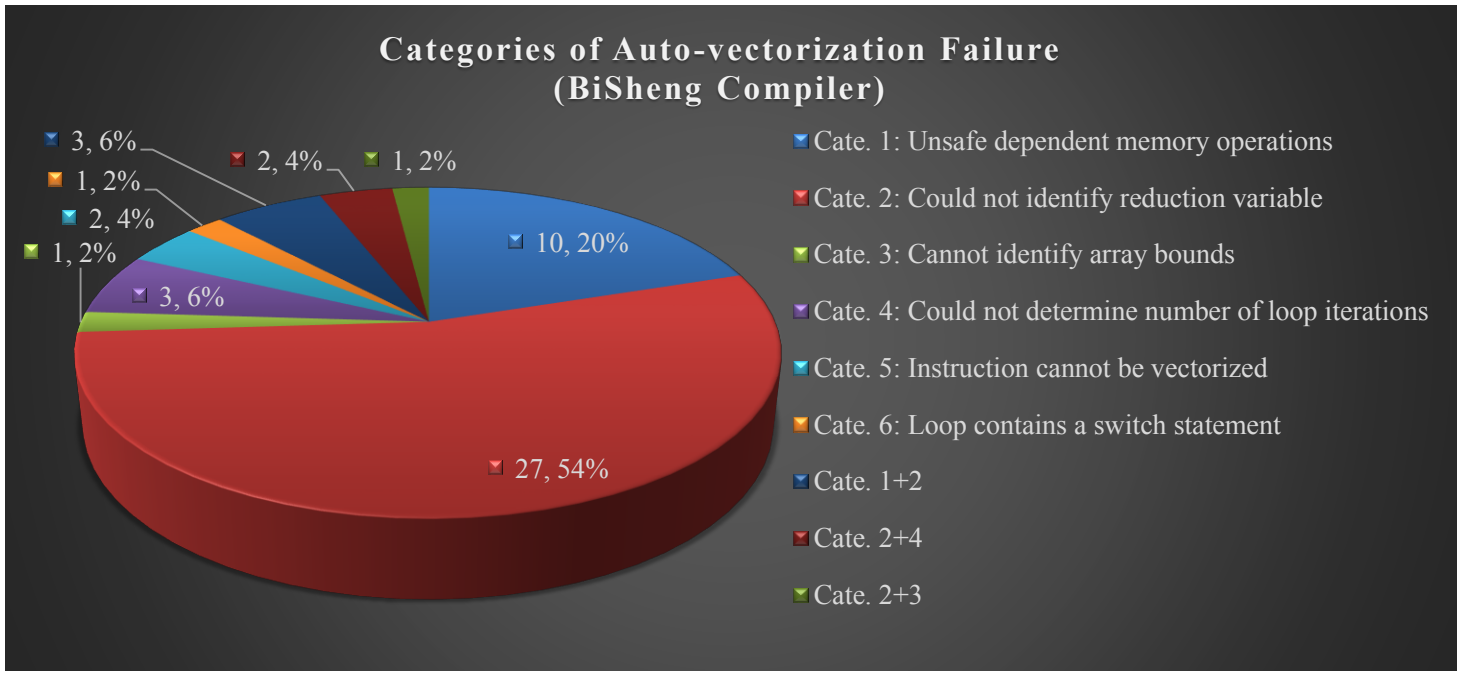
【More Information, please Refer to the Full Preprint Paper】

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**.



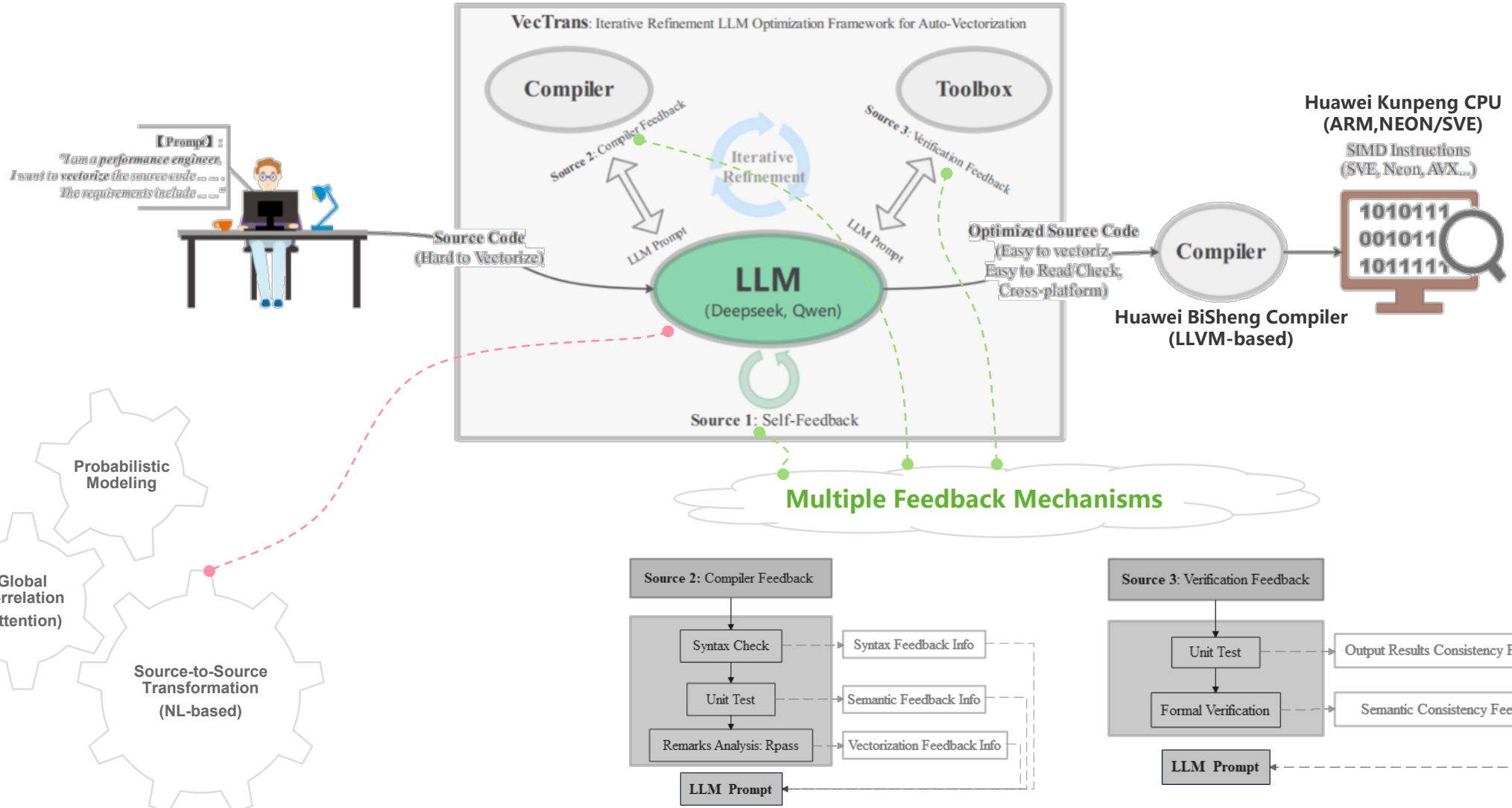


- **Compiler**: BiSheng Compiler(LLVM-based, Industrial-level),
- **Hardware**: Kunpeng CPU-ARM(NEON/SVE)
- **Optimization Flags**: (1) -O3 -ffast-math; (2) -Rpass=loop-vectorize; (3) -Rpass-analysis=loop-vectorize

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)

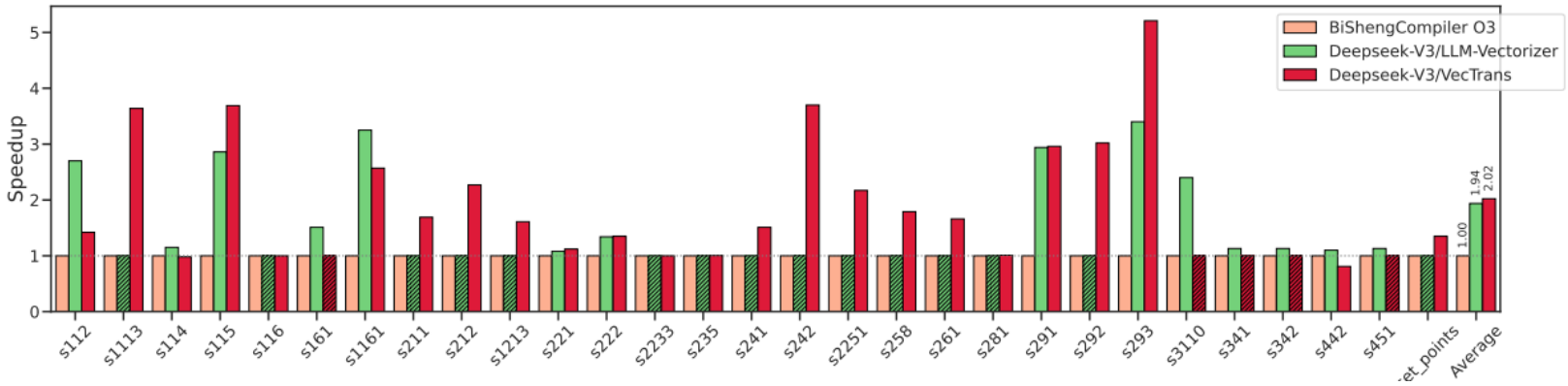
```
void s1113(int iterations, int LEN_1D, float* a,
    float* b)
{
    for (int nl = 0; nl < 2*iterations; nl++) {
        for (int i = 0; i < LEN_1D; i++) {
            a[i] = a[LEN_1D/2] + b[i];
        }
    }
}
```

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)

LLM-Vectorizer(C -> SIMD Intrinsic)
Taneja J, Laird A, Yan C, et al. *Llm-vectorizer: Llm-based verified loop vectorizer.* CGO 2025: 137-149.



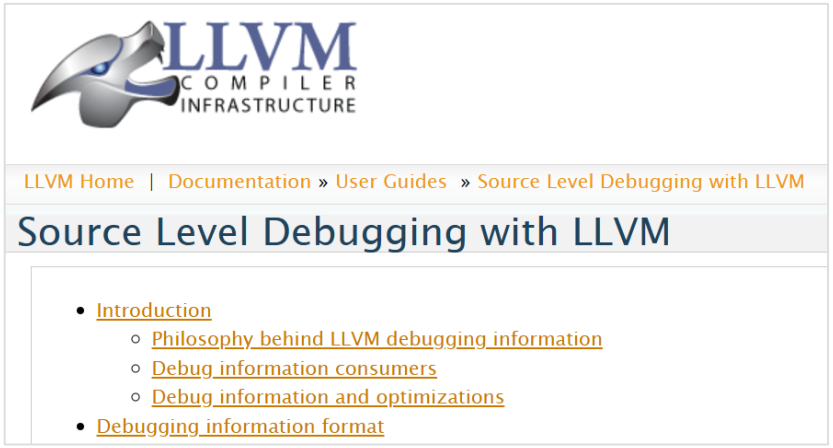
Auto-vectorization Ablation Experiments

(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)



OpenEuler https://www.openeuler.org/en/

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

Developers' Meeting

BERLIN 2025

Sony Interactive Entertainment | PlayStation

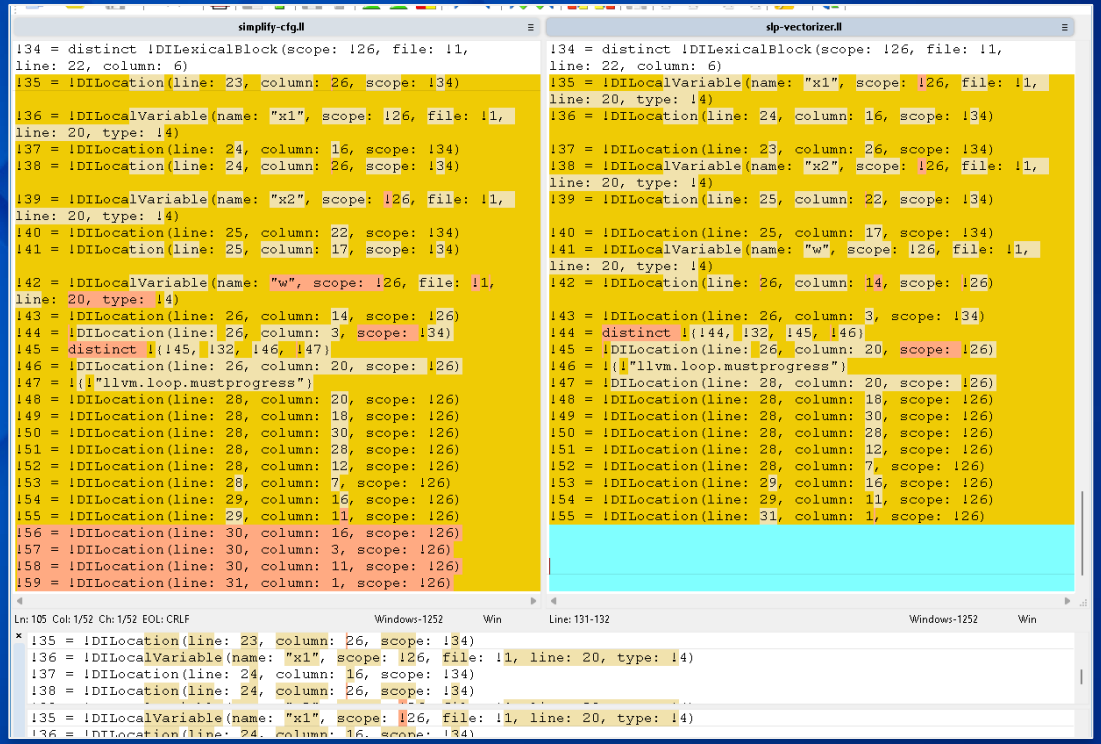
Why add an IR reader to llvm-debuginfo-analyzer tool

Carlos Alberto Enciso

Sony Interactive Entertainment | PlayStation

Reduce the noisiness of comparing the debuginfo in LLVM IR

Reduce the noisiness of comparing the debuginfo in LLVM IR



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

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

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

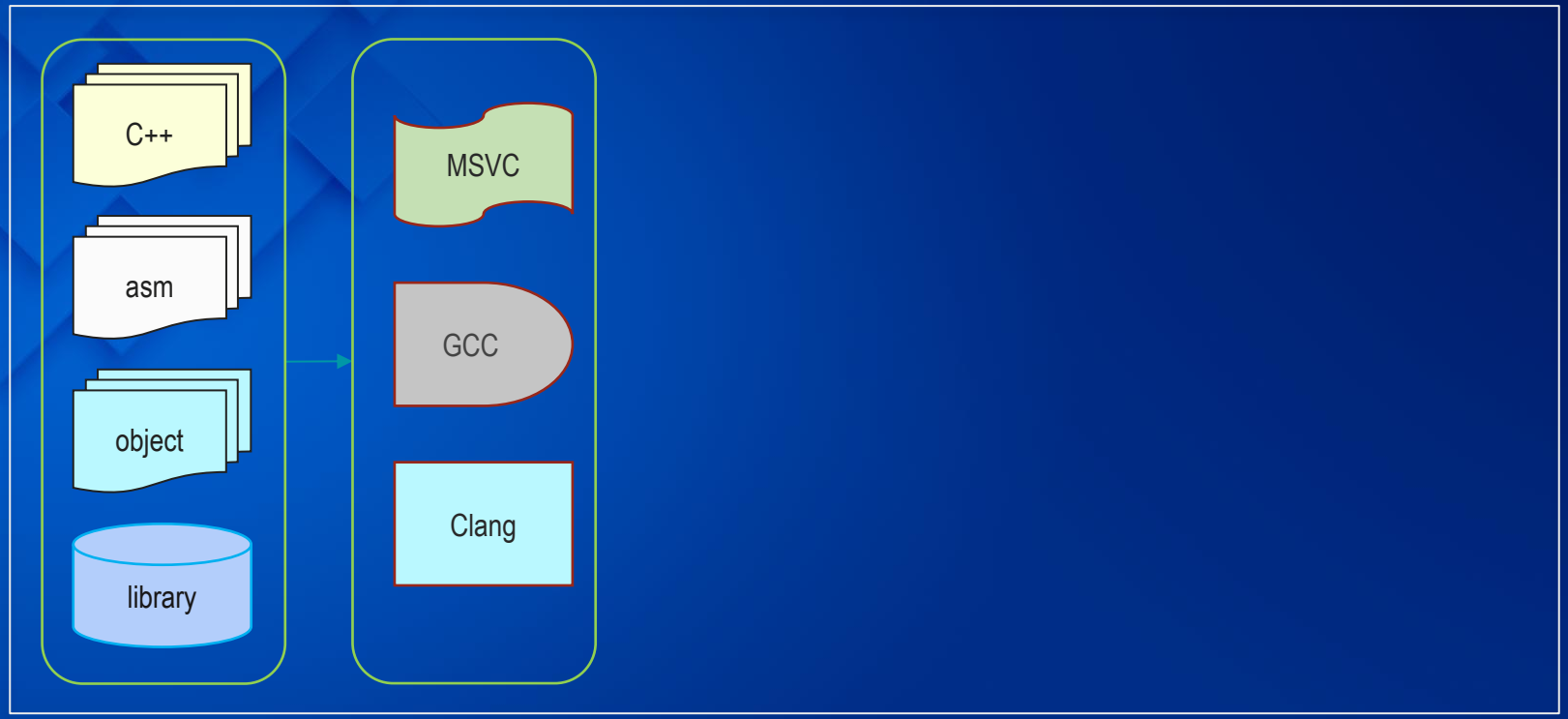
llvm-debuginfo-analyzer

- Basic introduction
- Print logical view
- Compare logical view

LLVM and debug information - inputs

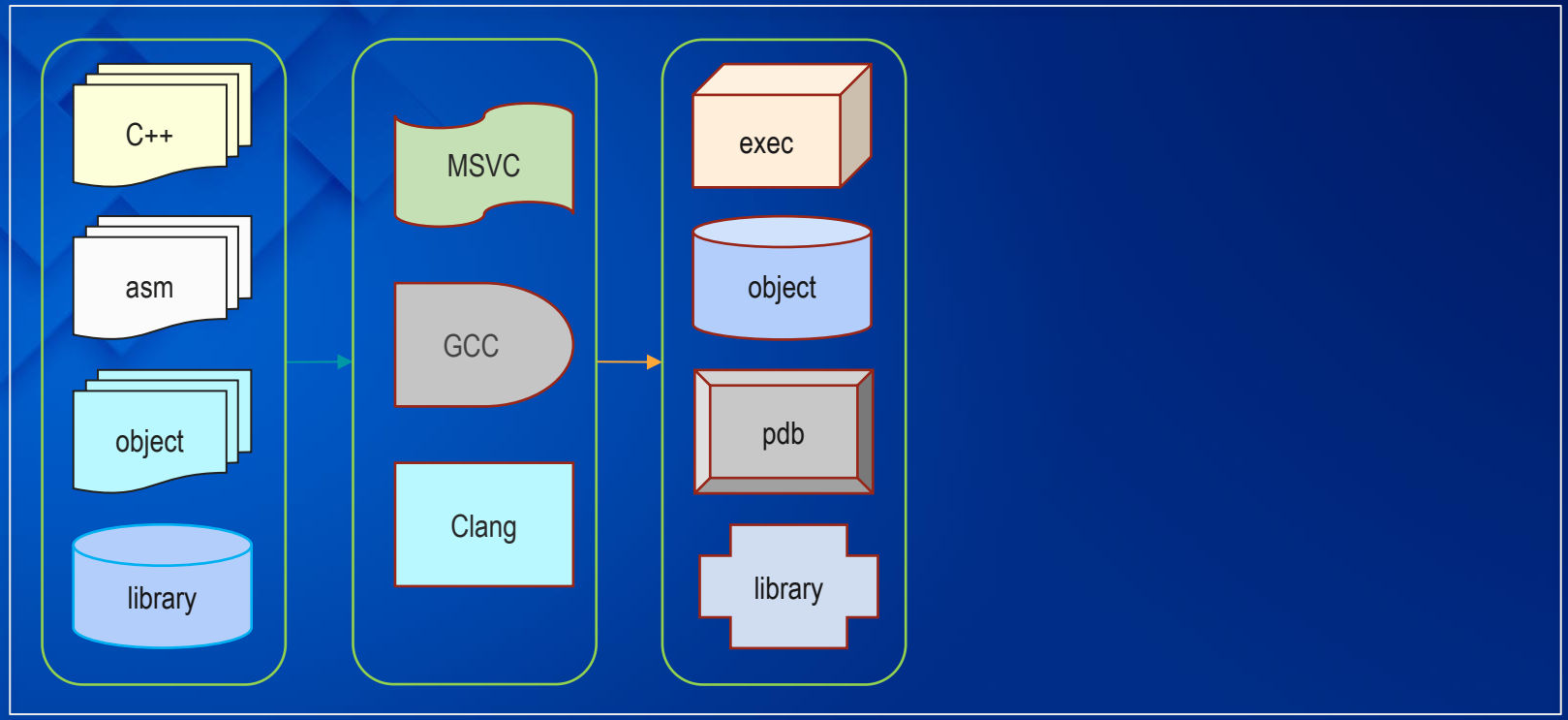
Different inputs

LLVM and debug information - toolchains



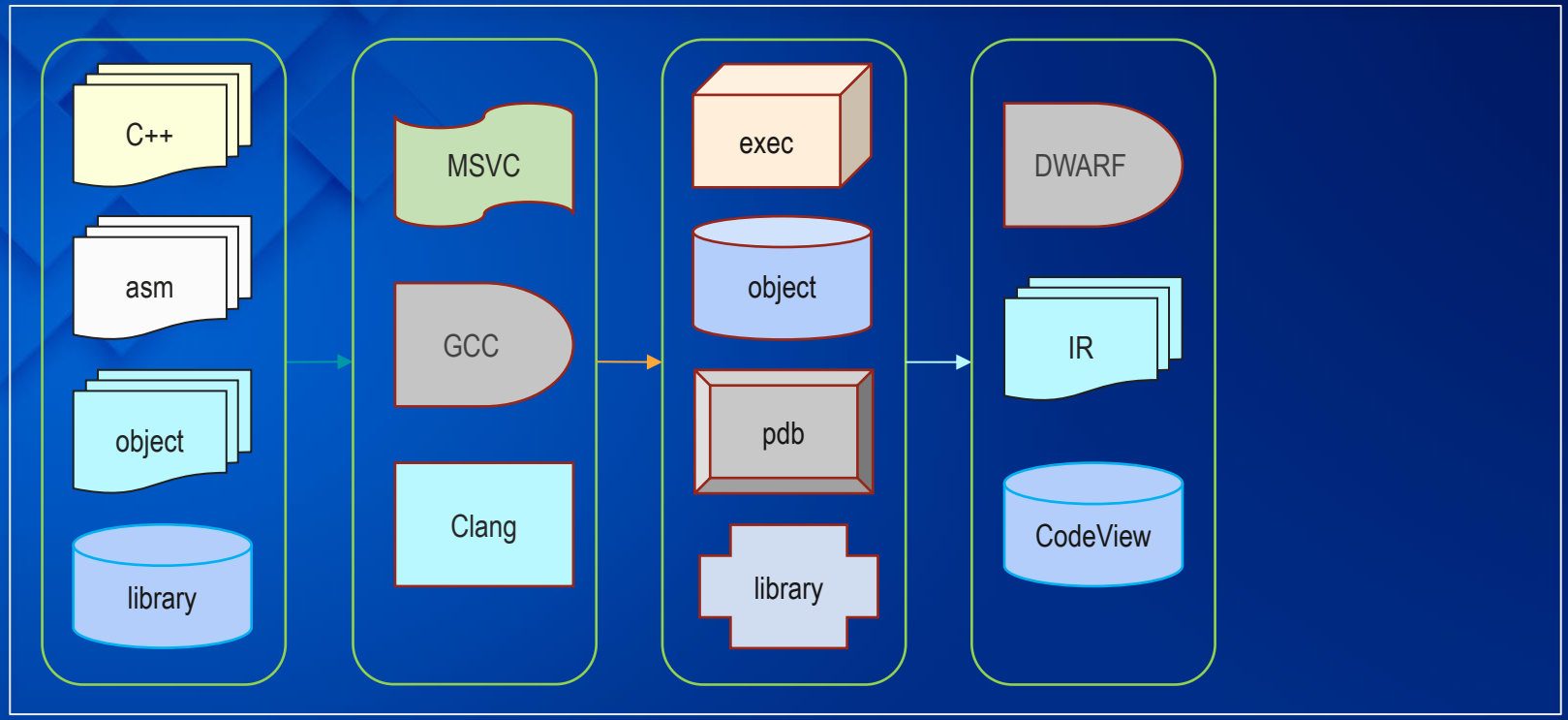
Different toolchains

LLVM and debug information - binary file formats



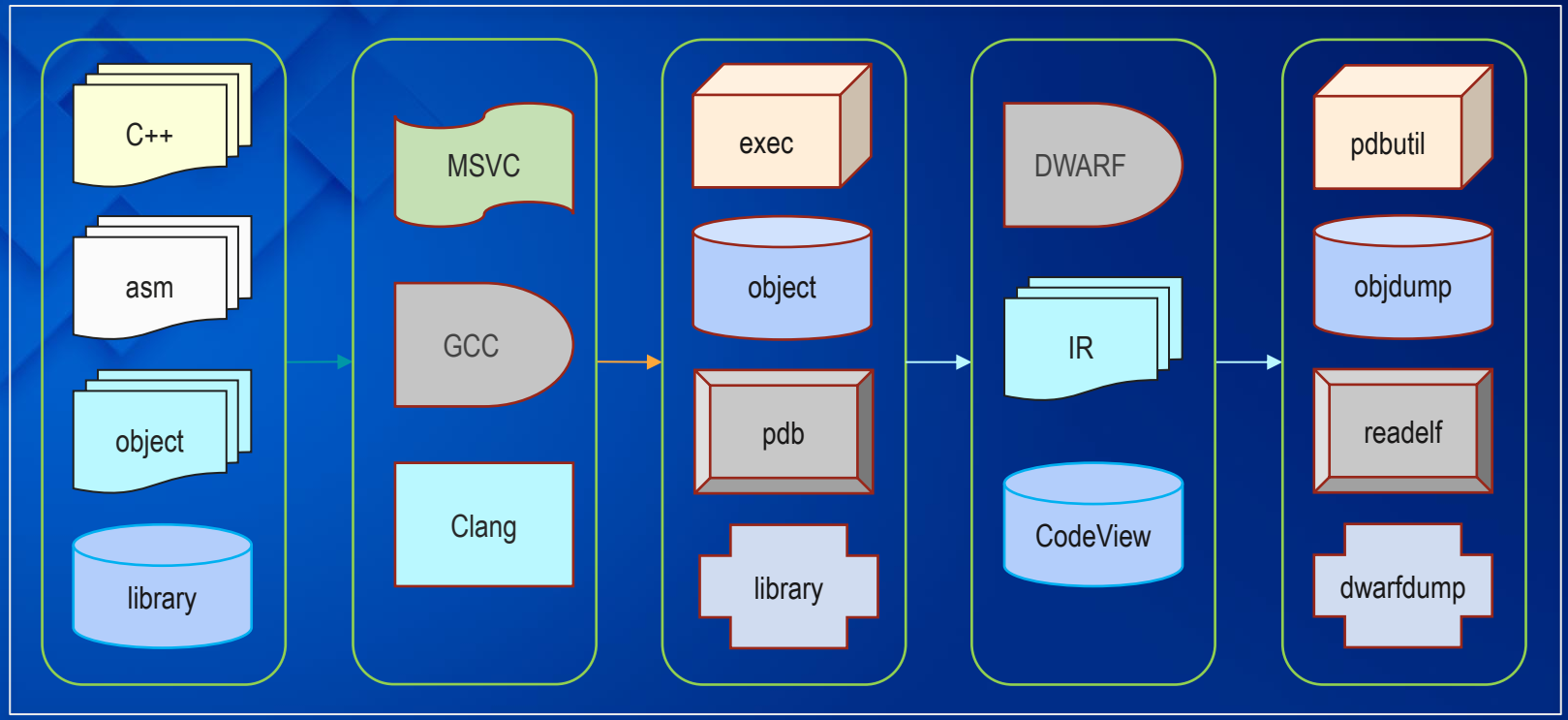
Different binary formats

LLVM and debug information - debug information formats



Different debug information formats

LLVM and debug information - debug information tools



Different debug information tools

Common problems with debug information

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

Common problems with debug information

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

Analyzing optimizer IR output

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

Analyzing optimizer IR output

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.

Analyzing optimizer IR output

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 }
```

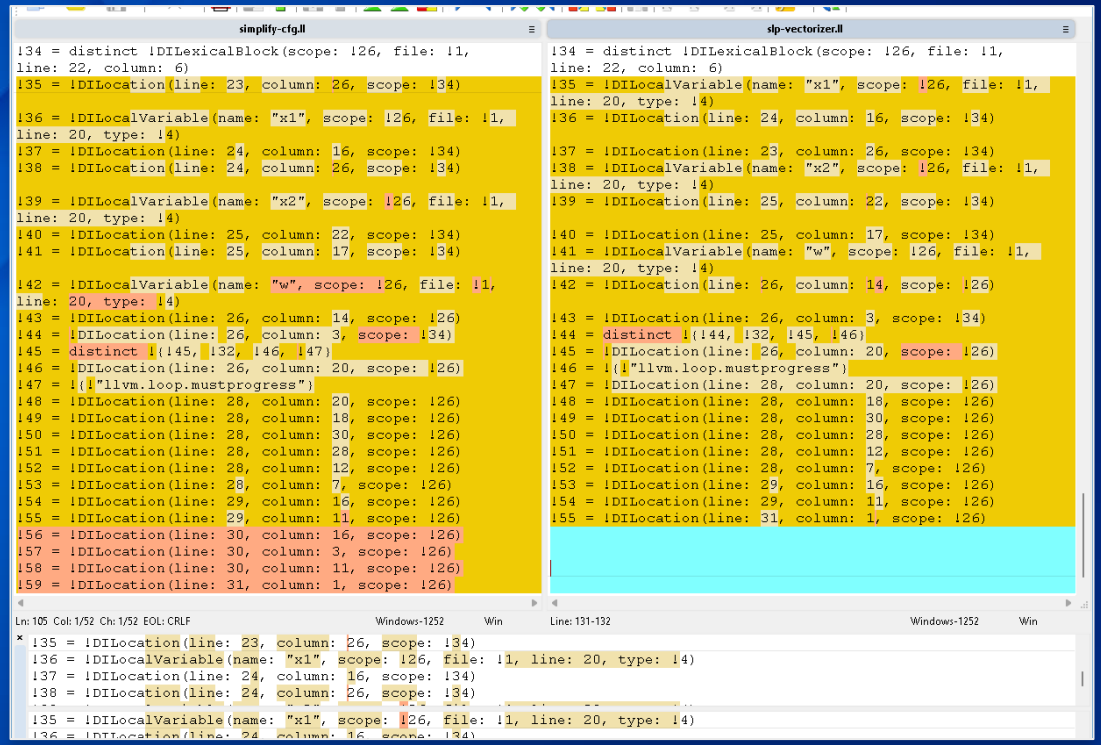
```
!35 = !DILocation(line: 23, column: 26, scope: !34)
!36 = !DILocalVariable(name: "x1", scope: !26, file:
!1, line: 20, type: !4)
!37 = !DILocation(line: 24, column: 16, scope: !34)
!38 = !DILocation(line: 24, column: 26, scope: !34)
!39 = !DILocalVariable(name: "x2", scope: !26, file:
!1, line: 20, 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

```
!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)
```

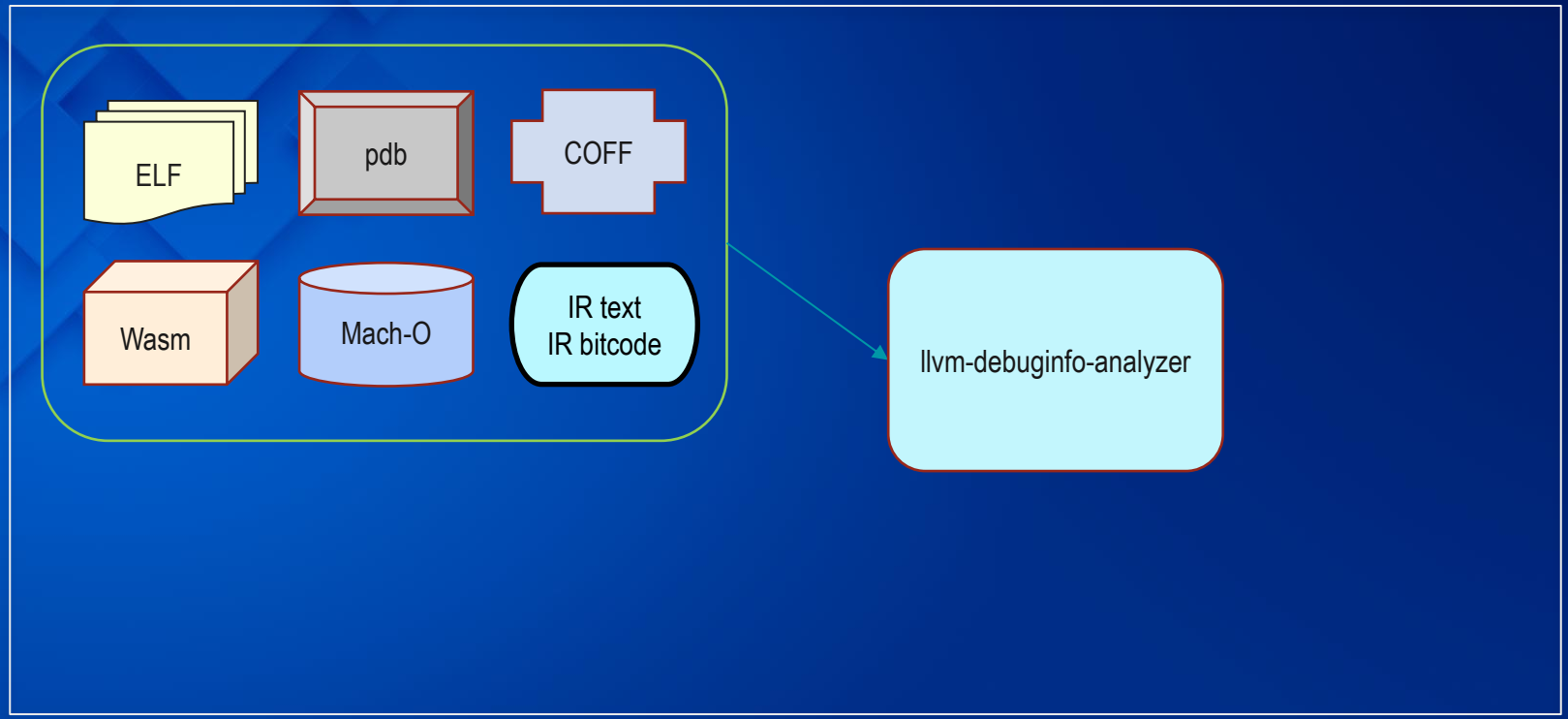
IR metadata after SLP Vectorizer

IR metadata comparison - comparison tool



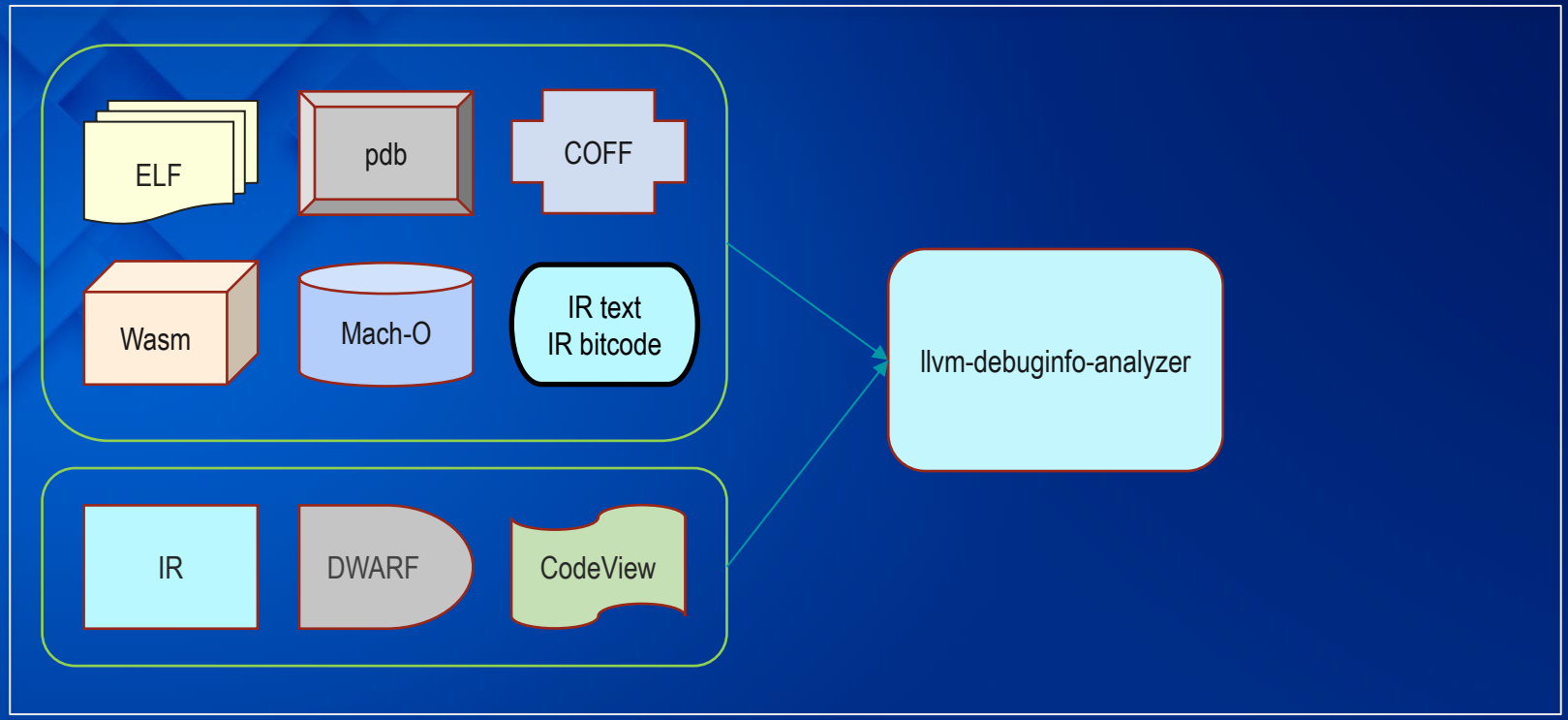
IR changes: comparison tool

llvm-debuginfo-analyzer - binary file formats



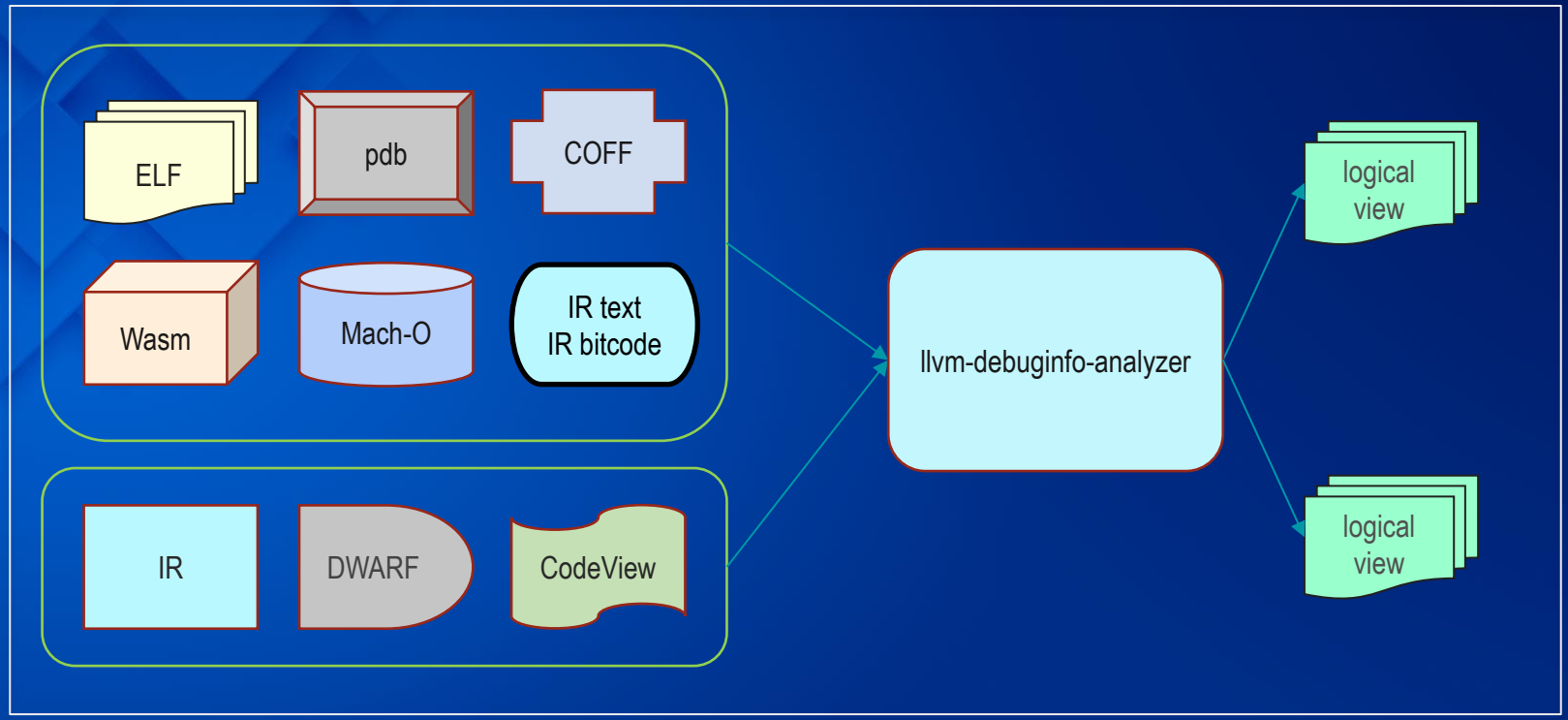
Supported binary file formats

llvm-debuginfo-analyzer - debug information formats



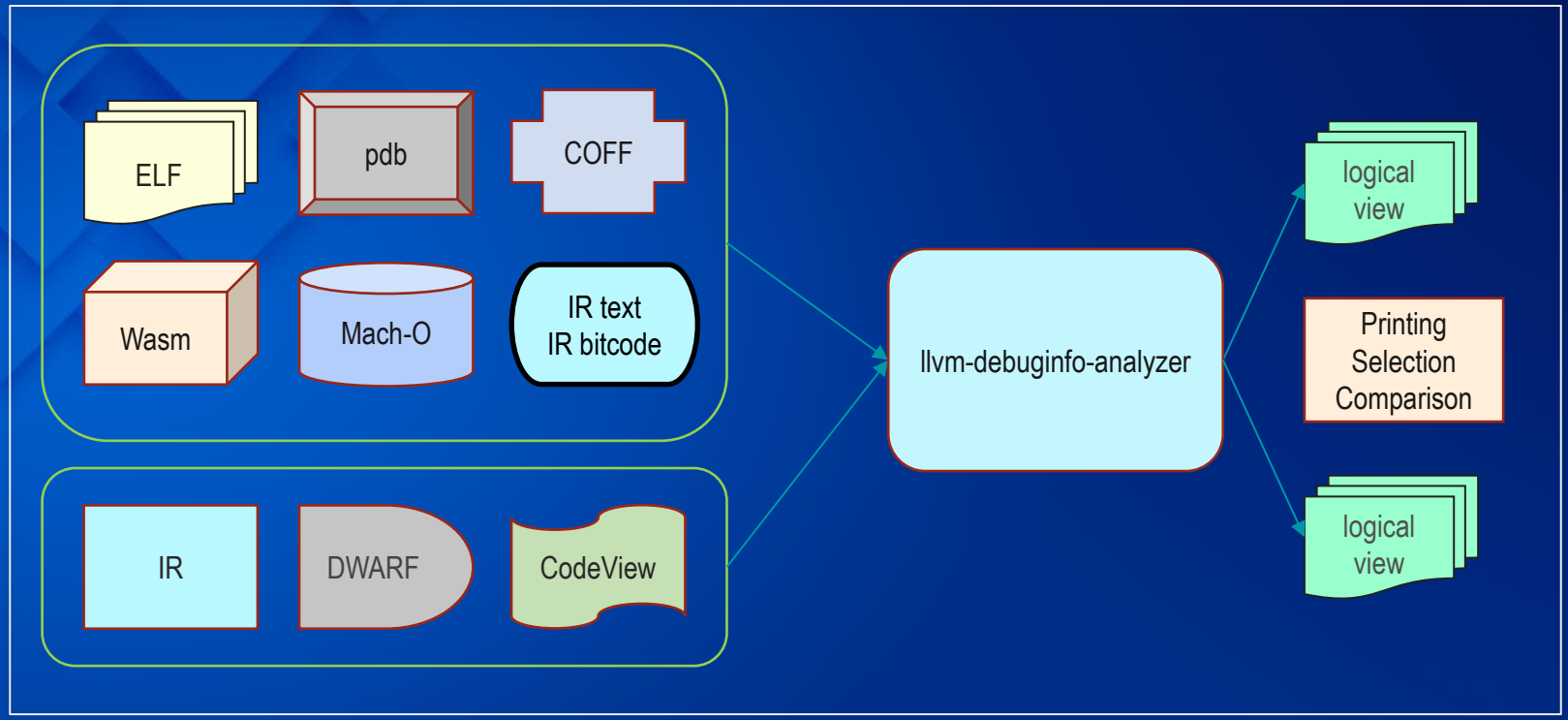
Supported debug information formats

llvm-debuginfo-analyzer - canonical logical views



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

llvm-debuginfo-analyzer - printing, selection, comparison



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  (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

```
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_OBJNAME [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

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  (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

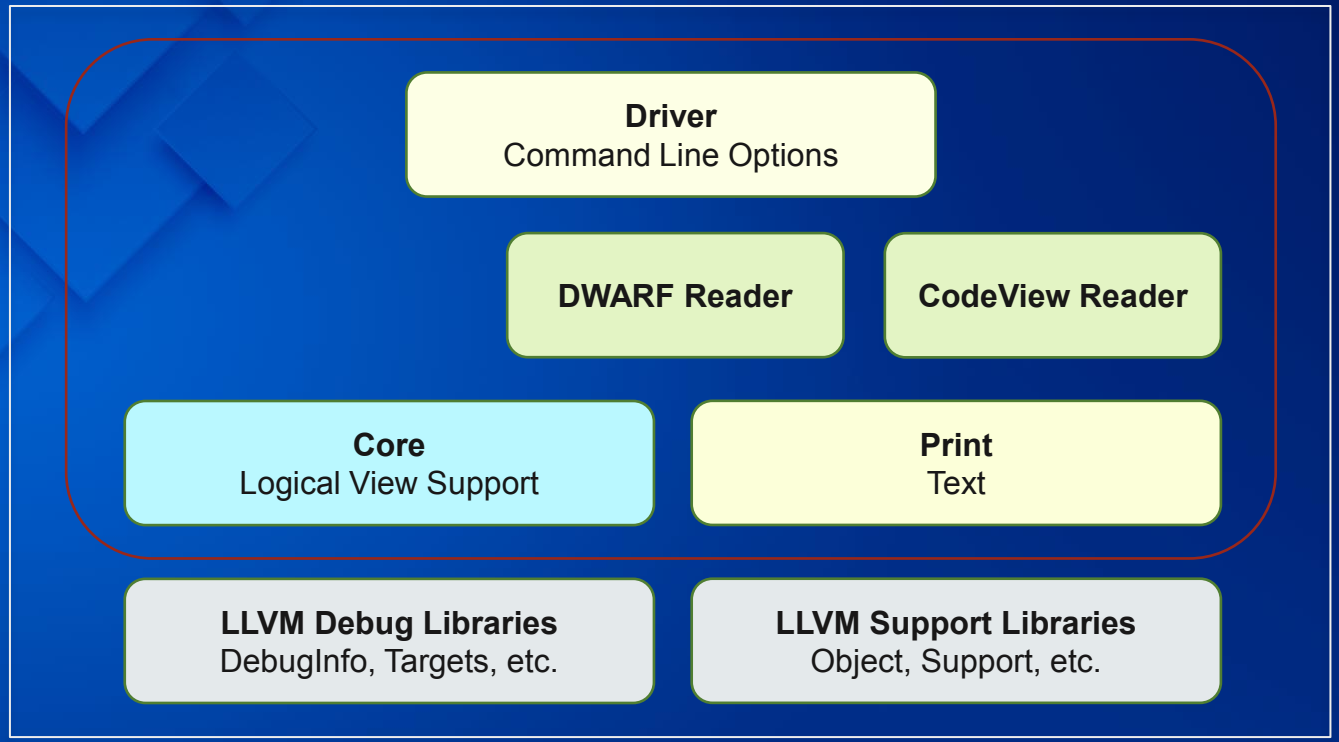
```
Logical View:
    {File} 'hello-world-clang.o'

      {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_OBJNAME [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`
```

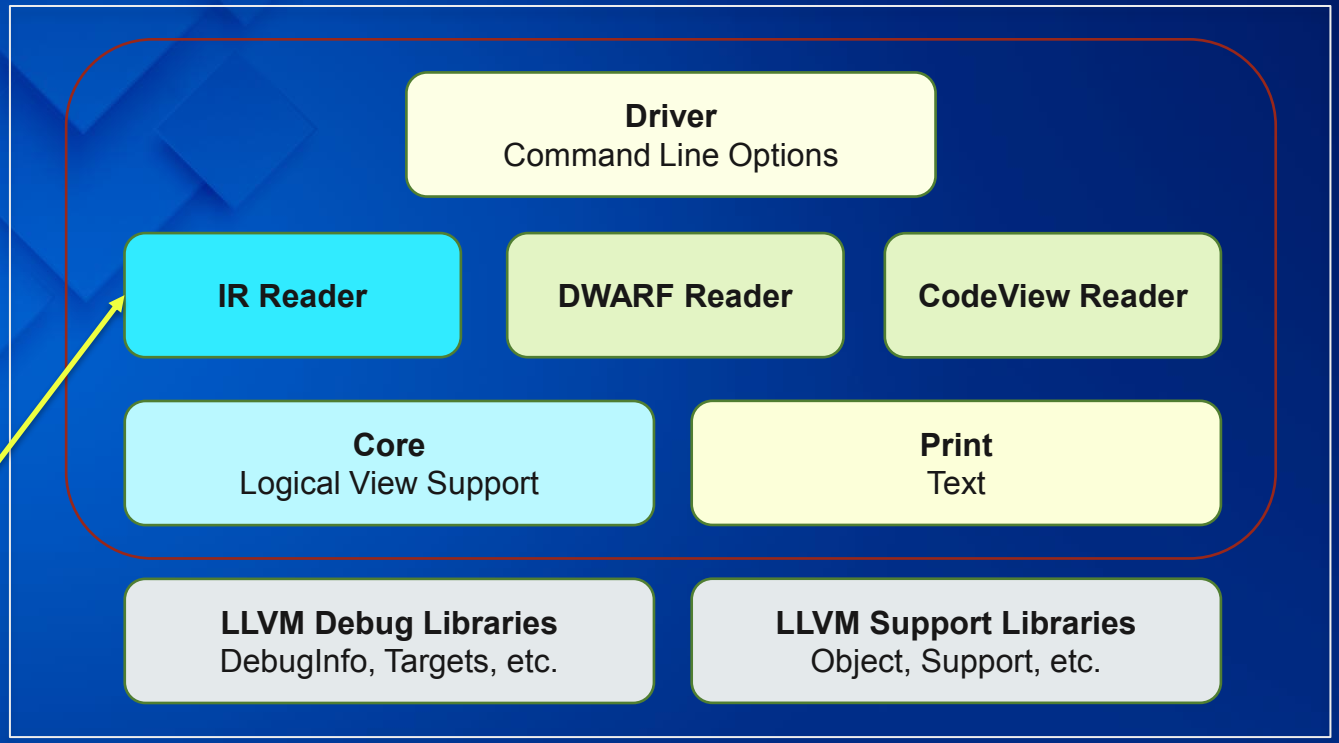
CodeView debug information

llvm-debuginfo-analyzer - current components



Current components

llvm-debuginfo-analyzer - IR reader component



IR Reader component

Print logical views

Common options to print logical views when dealing with IR

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

Print logical views

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

Print logical views

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.dbg.cu = !{!0}
!llvm.module.flags = !{!5, !6, !7, !8, !9, !10, !11}
!llvm.ident = !{!12}
!0 = distinct !DICompileUnit(language: DW_LANG_C_plus_plus_14, file: !1, producer: "clang", isOptimized: false, runtimeVersion: 0,
emissionKind: FullDebug, retainedTypes: !2, splitDebugInlining: false, nameTableKind: None)
!1 = !DIFile(filename: "test.cpp", directory: "")
!2 = !{!3}
!3 = !DIDerivedType(tag: DW_TAG_typedef, name: "F32", file: !1, line: 7, baseType: !4)
!4 = !DIBasicType(name: "float", size: 32, encoding: DW_ATE_float)
!5 = !{i32 7, !"Dwarf Version", i32 5}
!6 = !{i32 2, !"Debug Info Version", i32 3}
!12 = !{!"clang"}
!13 = distinct !DISubprogram(name: "RandF32", linkageName: "_Z7RandF32v", scope: !1, file: !1, line: 13, type: !14, scopeLine: 13, flags:
DIFlagPrototyped, spFlags: DISPFlagDefinition, unit: !0, retainedNodes: !16)
!14 = !DISubroutineType(types: !15)
!15 = !{!4}
!16 = !{}
!17 = !DILocation(line: 14, column: 15, scope: !13)
!18 = !DILocalVariable(name: "uRand", scope: !13, file: !1, line: 14, type: !19)
!19 = !DIDerivedType(tag: 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: 15, column: 21, scope: !13)
!23 = !DILocation(line: 15, column: 27, scope: !13)
!24 = !DILocalVariable(name: "fRand", scope: !13, file: !1, line: 15, type: !3)
!25 = !DILocation(line: 16, column: 3, scope: !13)
!26 = distinct !DISubprogram(name: "randGauss", linkageName: "_Z9randGaussPf", scope: !1, file: !1, line: 19, type: !27, scopeLine: 19,
flags: DIFlagPrototyped, spFlags: DISPFlagDefinition, unit: !0, retainedNodes: !16)
!27 = !DISubroutineType(types: !28)
!28 = !{null, !29}
!29 = !DIDerivedType(tag: DW_TAG_pointer_type, baseType: !4, size: 64)
!30 = !DILocalVariable(name: "work", arg: 1, scope: !26, file: !1, line: 19, type: !29)
!31 = !DILocation(line: 0, scope: !26)
!32 = !DILocation(line: 22, column: 3, scope: !26)
!33 = !DILocation(line: 23, column: 16, scope: !34)
!34 = distinct !DILexicalBlock(scope: !26, 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: 18, 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

Logical views for Simplify CFG and SLP Vectorizer passes

```
!llvm.dbg.cu = !{!0}
!llvm.module.flags = !{!5, !6, !7, !8, !9, !10, !11}
!llvm.ident = !{!12}
!0 = distinct !DICompileUnit(language: DW_LANG_C_plus_plus_14, file: !1, producer: "clang", isOptimized: false, runtimeVersion: 0,
emissionKind: FullDebug, retainedTypes: !2, splitDebugInlining: false, nameTableKind: None)
!1 = !DIFile(filename: "test.cpp", directory: "")
!2 = !{!3}
!3 = !DIDerivedType(tag: DW_TAG_typedef, name: "F32", file: !1, line: 7, baseType: !4)
!4 = !DIBasicType(name: "float", size: 32, encoding: DW_ATE_float)
!5 = !{i32 7, !"Dwarf Version", i32 5}
!6 = !{i32 2, !"Debug Info Version", i32 3}
!12 = !{!"clang"}
!13 = distinct !DISubprogram(name: "RandF32", linkageName: "_Z7RandF32v", scope: !1, file: !1, line: 13, type: !14, scopeLine: 13, flags:
DIFlagPrototyped, spFlags: DISPFlagDefinition, unit: !0, retainedNodes: !16)
!14 = !DISubroutineType(types: !15)
!15 = !{!4}
!16 = !{}
!17 = !DILocation(line: 14, column: 15, scope: !13)
!18 = !DILocalVariable(name: "uRand", scope: !13, file: !1, line: 14, type: !19)
!19 = !DIDerivedType(tag: 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: 15, column: 21, scope: !13)
!23 = !DILocation(line: 15, column: 27, scope: !13)
!24 = !DILocalVariable(name: "fRand", scope: !13, file: !1, line: 15, type: !3)
!25 = !DILocation(line: 16, column: 3, scope: !13)
!26 = distinct !DISubprogram(name: "randGauss", linkageName: "_Z9randGaussPf", scope: !1, file: !1, line: 19, type: !27, scopeLine: 19,
flags: DIFlagPrototyped, spFlags: DISPFlagDefinition, unit: !0, retainedNodes: !16)
!27 = !DISubroutineType(types: !28)
!28 = !{null, !29}
!29 = !DIDerivedType(tag: DW_TAG_pointer_type, baseType: !4, size: 64)
!30 = !DILocalVariable(name: "work", arg: 1, scope: !26, file: !1, line: 19, type: !29)
!31 = !DILocation(line: 0, scope: !26)
!32 = !DILocation(line: 22, column: 3, scope: !26)
!33 = !DILocation(line: 23, column: 16, scope: !34)
!34 = distinct !DILexicalBlock(scope: !26, 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: 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

```
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.dbg.cu = !{!0}
!llvm.module.flags = !{!5, !6, !7, !8, !9, !10, !11}
!llvm.ident = !{!12}
!0 = distinct !DICompileUnit(language: DW_LANG_C_plus_plus_14, file: !1, producer: "clang", isOptimized: false, runtimeVersion: 0,
emissionKind: FullDebug, retainedTypes: !2, splitDebugInlining: false, nameTableKind: None)
!1 = !DIFile(filename: "test.cpp", directory: "")
!2 = !{!3}
!3 = !DIDerivedType(tag: DW_TAG_typedef, name: "F32", file: !1, line: 7, baseType: !4)
!4 = !DIBasicType(name: "float", size: 32, encoding: DW_ATE_float)
!5 = !{i32 7, !"Dwarf Version", i32 5}
!6 = !{i32 2, !"Debug Info Version", i32 3}
!12 = !{!"clang"}
!13 = distinct !DISubprogram(name: "RandF32", linkageName: "_Z7RandF32v", scope: !1, file: !1, line: 13, type: !14, scopeLine: 13, flags:
DIFlagPrototyped, spFlags: DISPFlagDefinition, unit: !0, retainedNodes: !16)
!14 = !DISubroutineType(types: !15)
!15 = !{!4}
!16 = !{}
!17 = !DILocation(line: 14, column: 15, scope: !13)
!18 = !DILocalVariable(name: "uRand", scope: !13, file: !1, line: 14, type: !19)
!19 = !DIDerivedType(tag: 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: 15, column: 21, scope: !13)
!23 = !DILocation(line: 15, column: 27, scope: !13)
!24 = !DILocalVariable(name: "fRand", scope: !13, file: !1, line: 15, type: !3)
!25 = !DILocation(line: 16, column: 3, scope: !13)
!26 = distinct !DISubprogram(name: "randGauss", linkageName: "_Z9randGaussPf", scope: !1, file: !1, line: 19, type: !27, scopeLine: 19,
flags: DIFlagPrototyped, spFlags: DISPFlagDefinition, unit: !0, retainedNodes: !16)
!27 = !DISubroutineType(types: !28)
!28 = !{null, !29}
!29 = !DIDerivedType(tag: DW_TAG_pointer_type, baseType: !4, size: 64)
!30 = !DILocalVariable(name: "work", arg: 1, scope: !26, file: !1, line: 19, type: !29)
!31 = !DILocation(line: 0, scope: !26)
!32 = !DILocation(line: 22, column: 3, scope: !26)
!33 = !DILocation(line: 23, column: 16, scope: !34)
!34 = distinct !DILexicalBlock(scope: !26, 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: 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

```
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

Compare logical views

Common options to compare logical views when dealing with IR

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

Compare logical views

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

Compare logical views

Common options to compare logical views when dealing with IR

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

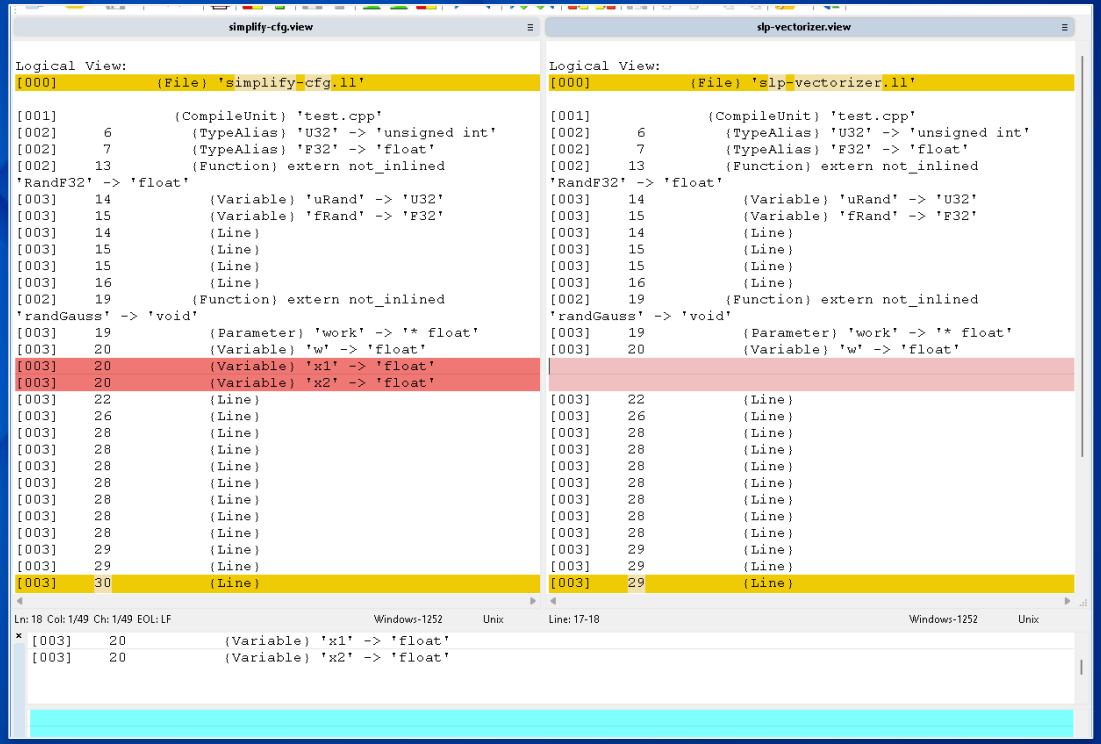
IR tests

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

llvm-debuginfo-analyzer command line

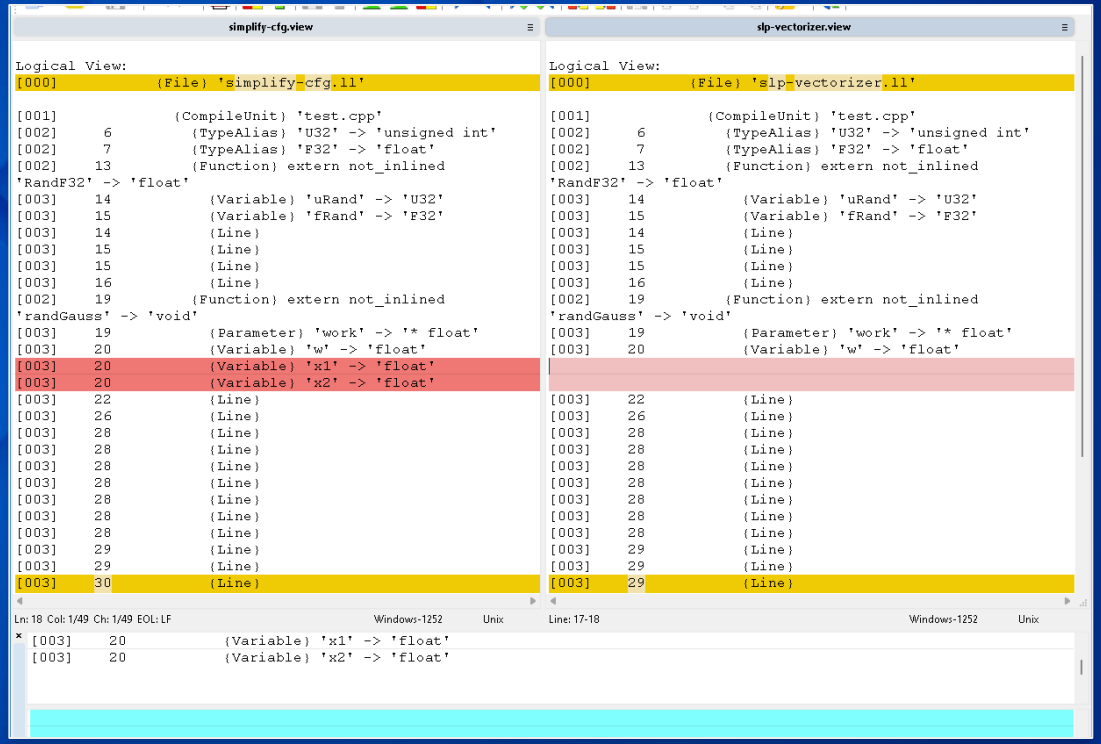
- --compare=symbols,lines --report=list --print=symbols,lines simplify-cfg.ll slp-vectorizer.ll
- --compare=symbols,lines --report=view --print=symbols simplify-cfg.ll slp-vectorizer.ll

Logical view changes - comparison tool



IR changes: comparison tool

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

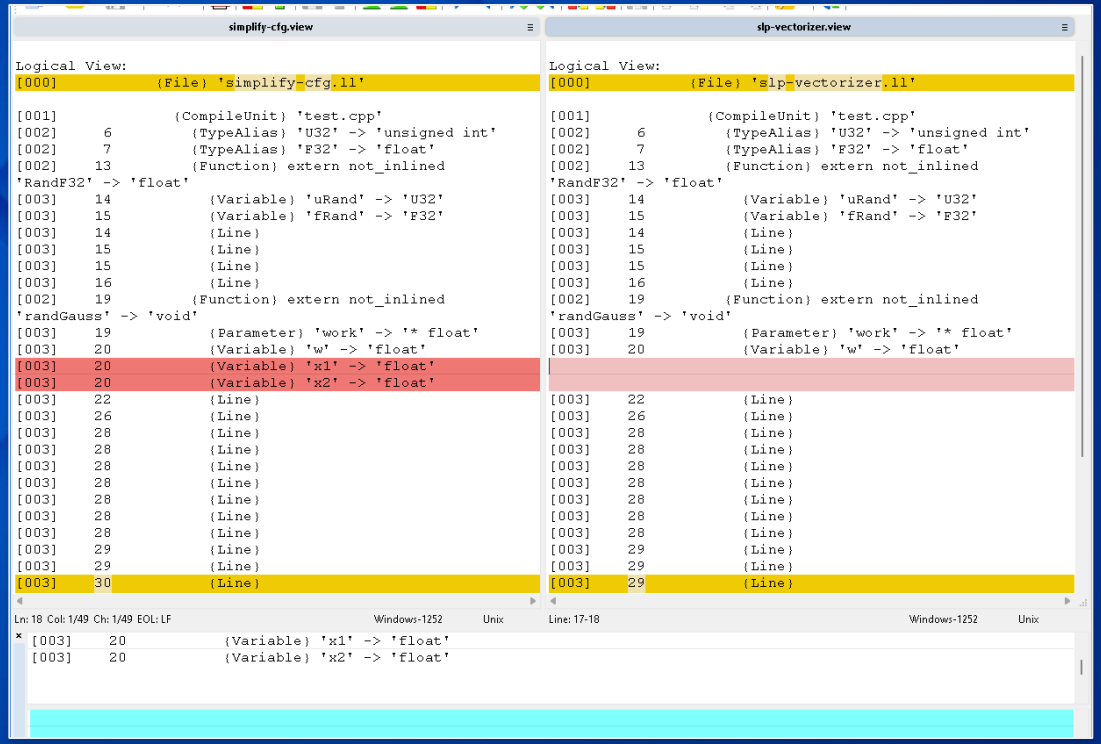


IR changes: comparison tool

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

IR changes: llvm-debuginfo-analyzer

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

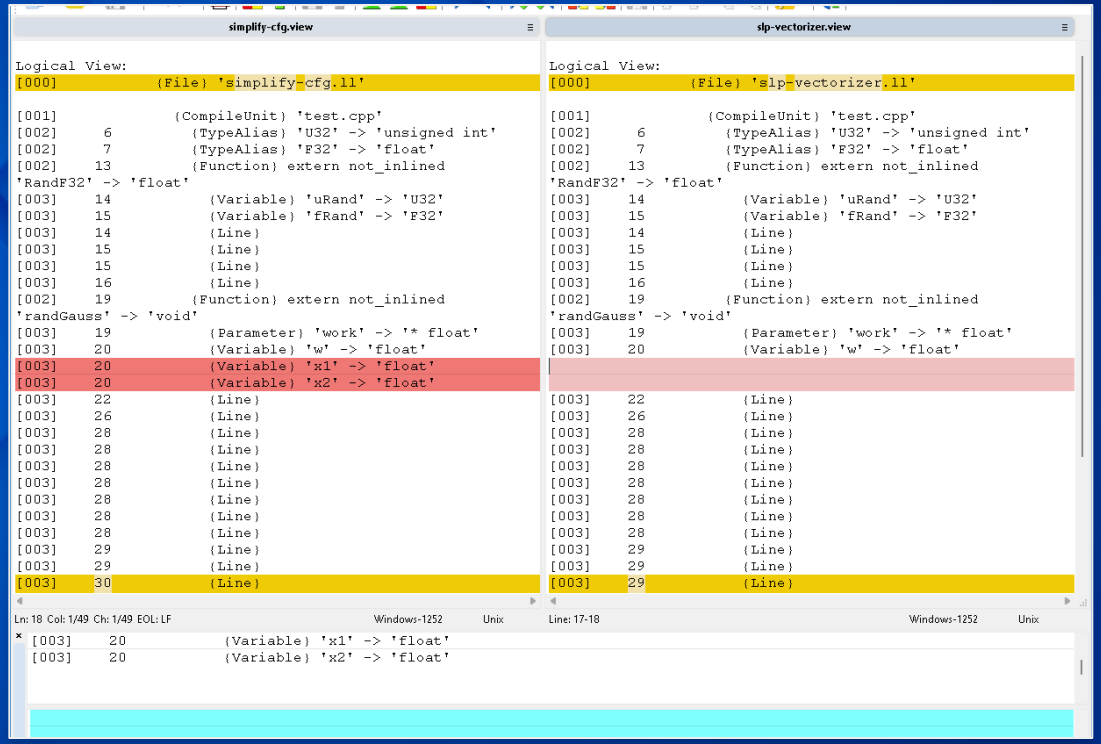


IR changes: comparison tool

```
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)

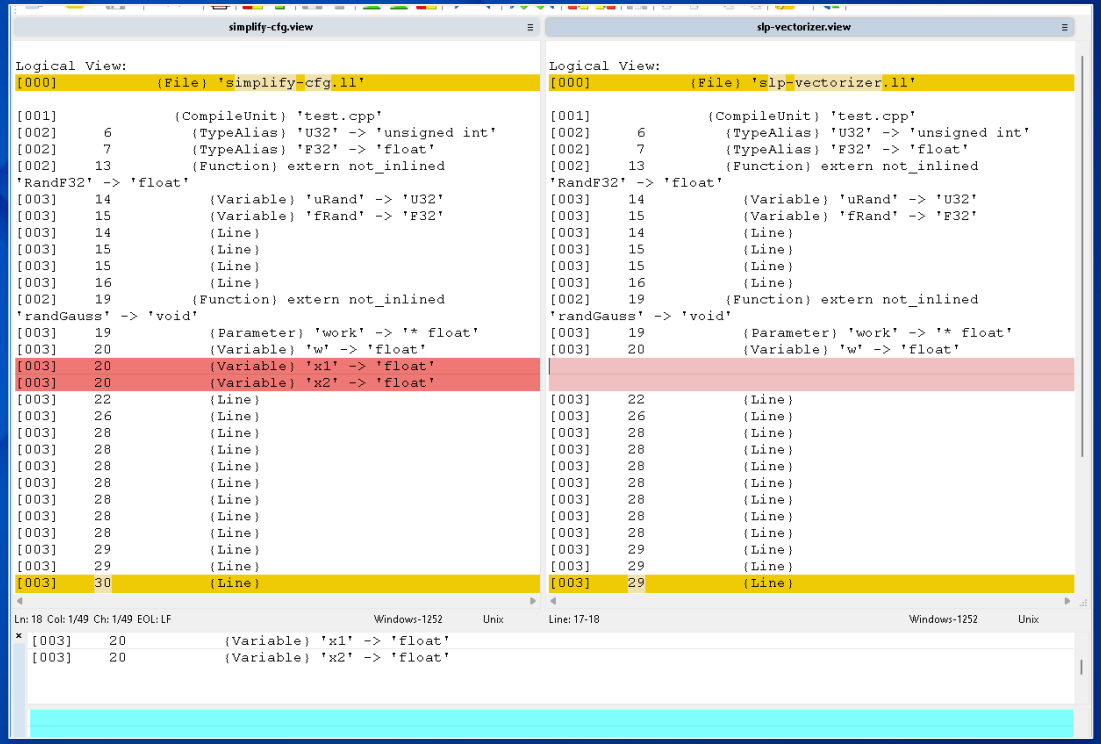


IR changes: comparison tool

```
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}
```

IR changes: llvm-debuginfo-analyzer

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



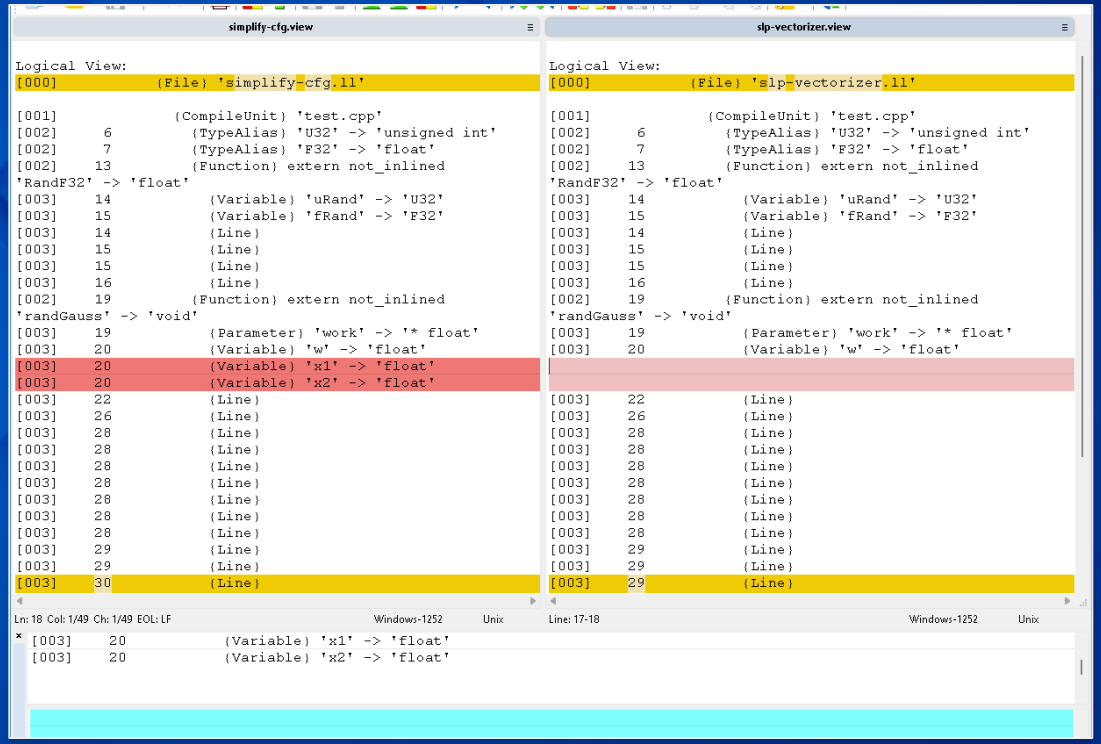
IR changes: comparison tool

```
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)



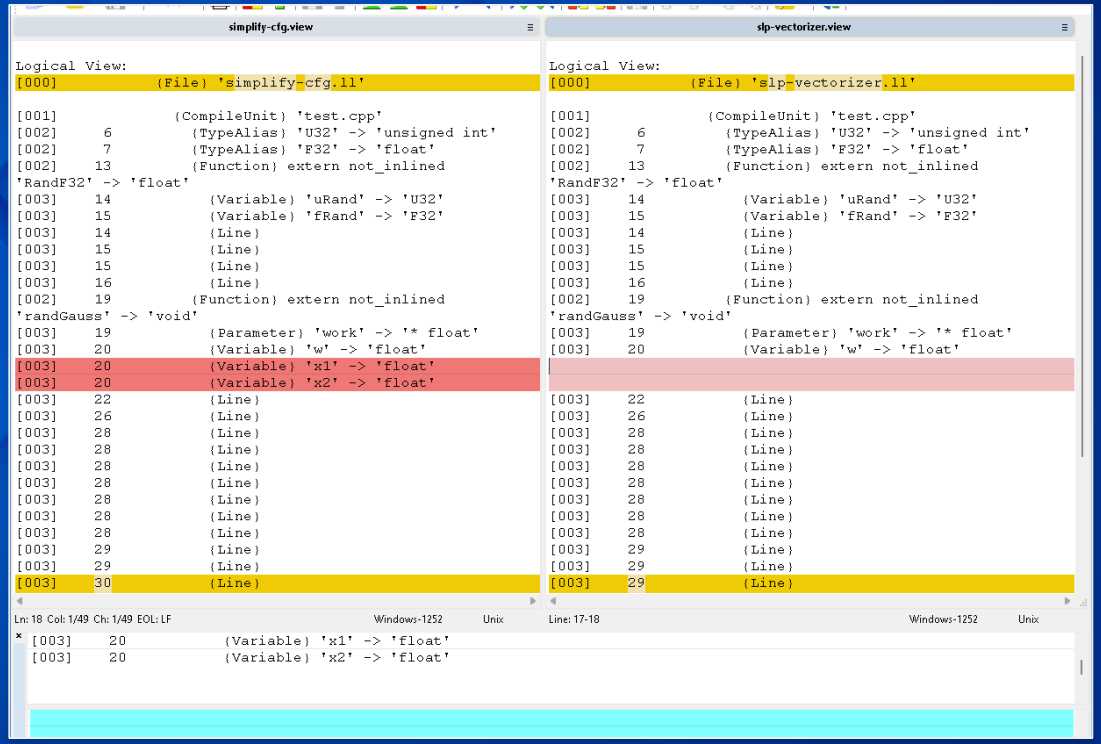
IR changes: comparison tool

```
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)



IR changes: comparison tool

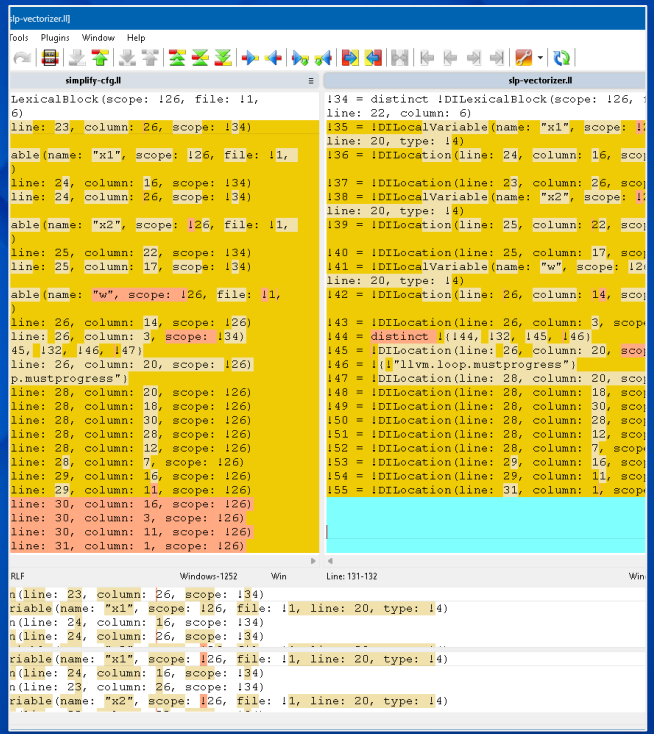
```
Reference: ‘simplify-cfg.ll'
Target:    ‘slp-vectorizer.ll'

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

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

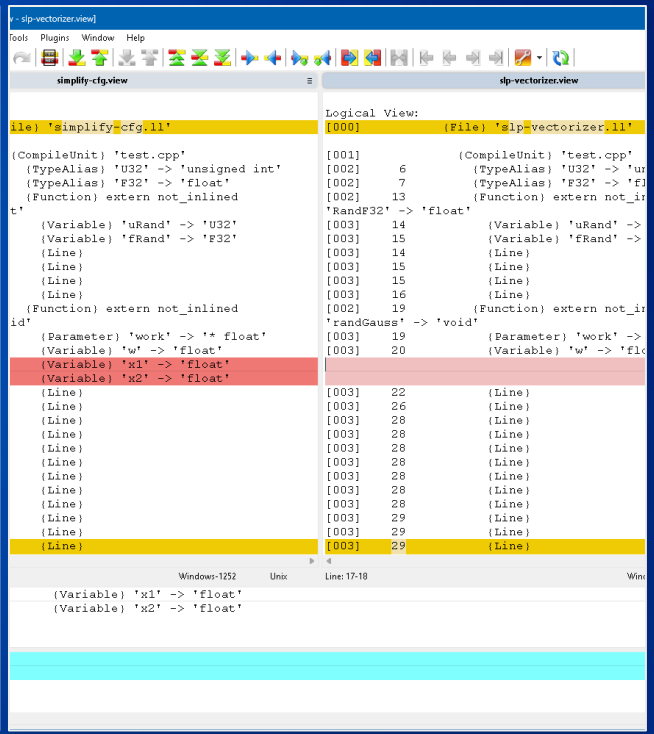
```
Reference: ‘simplify-cfg.ll'
Target:    ‘slp-vectorizer.ll'

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





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}  '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

Sony Interactive Entertainment | PlayStation

Thank you!

Developers' Meeting

BERLIN 2025

Samsung Research

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 & ~rs2;	Scalar	Zbb or Zbkb
orn rd, rs1, rs2	rd = rs1 | ~rs2;		
xnor rd, rs1, rs2	rd = rs1 ^ ~rs2;		
vandn.vv vd, vs1, vs2	for (int i = 0; i < vl; i++) vd[i] = vs1[i] & ~vs2[i];	Vector	Zvkb
vandn.vx vd, vs1, rs2	for (int i = 0; i < vl; i++) vd[i] = vs1[i] & ~rs2;		

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 = [HasStdExtZbbOrZbkb] 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 << 12	“Load Upper Immediate”
lui rd, simm20 addi rd, rd, simm12	rd = (simm20 << 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
lui a1, HI+1 addi a1, a1, -1 or a0, a0, a1	lui a1, ~HI orn a0, a0, a1

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

Samsung Research

Thank you

Developers' Meeting

BERLIN 2025

EuroLLVM ’25

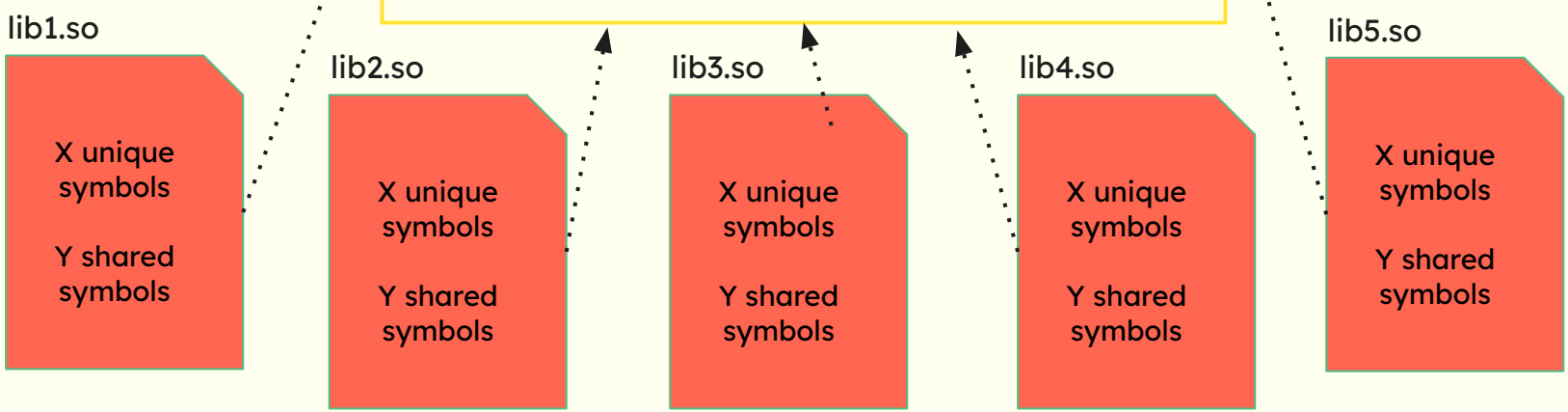
Artem Pianykh
Software Engineer @ Meta

Accidentally quadratic in `compiler-rt/asan`

Motivating Example / 1

https://github.com/artempyanykh/eurollvm25

```cpp
#include <iostream>

int main(int argc, char *argv[]) {
   std::cout << "Hello EuroLLVM!" << std::endl;
}

extern "C" const char *__asan_default_options() {
   return "detect_odr_violation=1";
}
```



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.

~~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, __asa
                nop
   0.02 810:  ┌→movq    0x8(%r15),%r15
   0.55       │ testq   %r15,%r15
   0.14       │↓ je      910
        81d:  │ movq    0x38(%r12),%rcx
   1.28       │ movq    (%r15),%rax
   9.71       │ cmpq    0x38(%rax),%rcx
  88.22       └ jne     810
                movq    $0x54fce0,%rcx
   0.00         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*)
```

Is this a linked list?

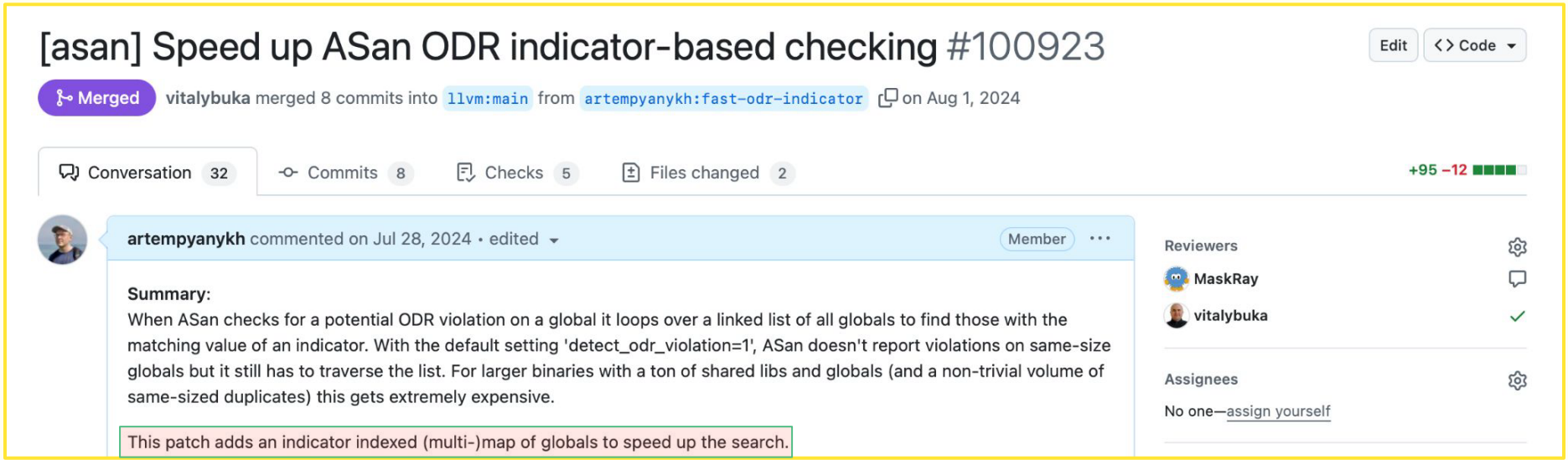
Credits: SVTFOE (c) by Daron Nefcy

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_odr_violation >= 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)



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

Edit | Code

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 | +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

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

Thoughts

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

Ivan Ho

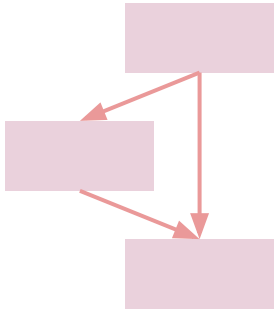
PhD Student

Supervised by Tobias Grosser

UNIVERSITY OF CAMBRIDGE

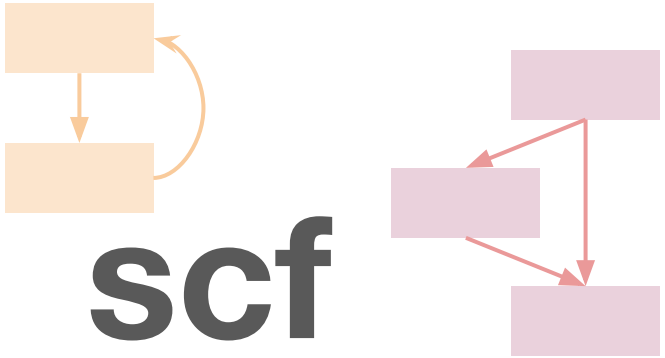
arith

arith



scf

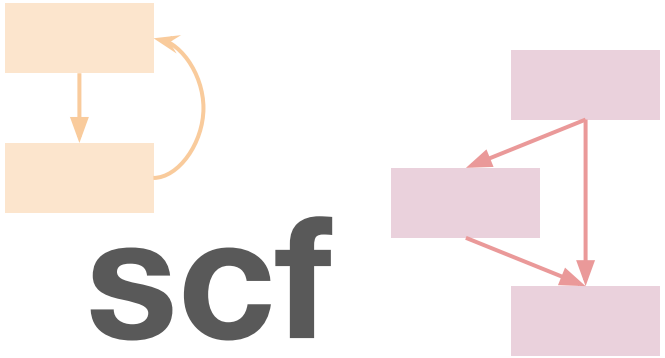
arith



scf

func

arith



scf

func

transform

What if we had dialects as a dialect?

What if we had dialects as a dialect?

IRDL

\\\\٩(๑`^´๑)۶////

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 --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)
  }
}
```

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

```
$ 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)
  }
}
```

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

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

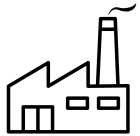
❌ unsatisfied constraint

IRDL: IR Definition Language

Concise

Introspectable

Dynamic



Generatable

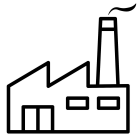
And it *just* works…

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
```

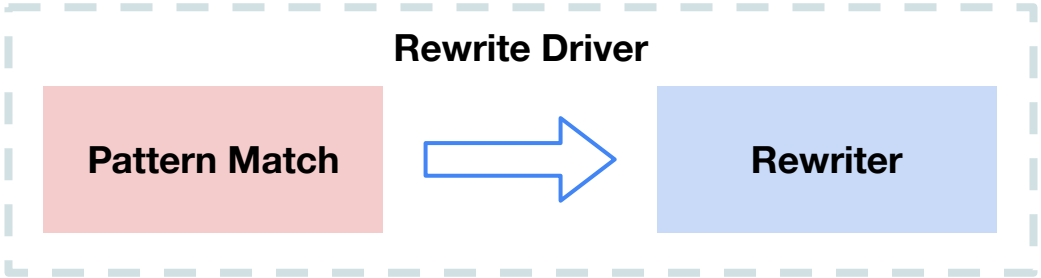
Is parsing this program enough?

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


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

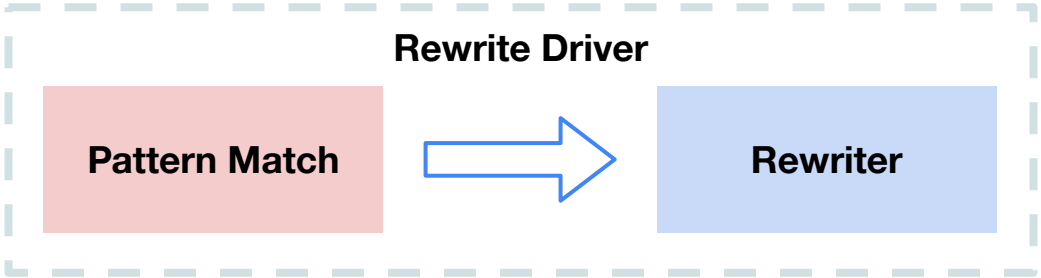
How to Rewrite IR in MLIR

(the abridged version)



How to Rewrite IR in MLIR

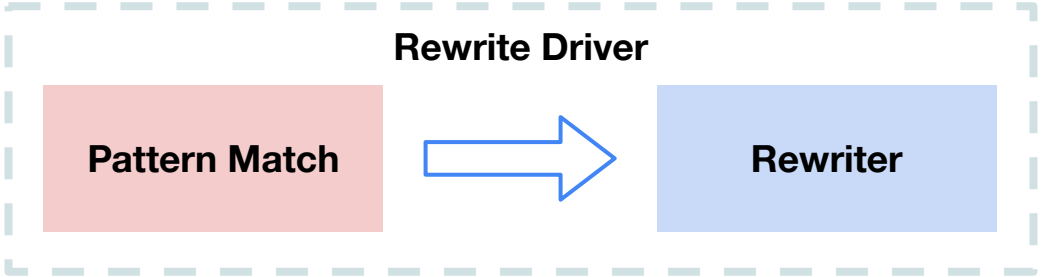
(the abridged version)




```
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**!

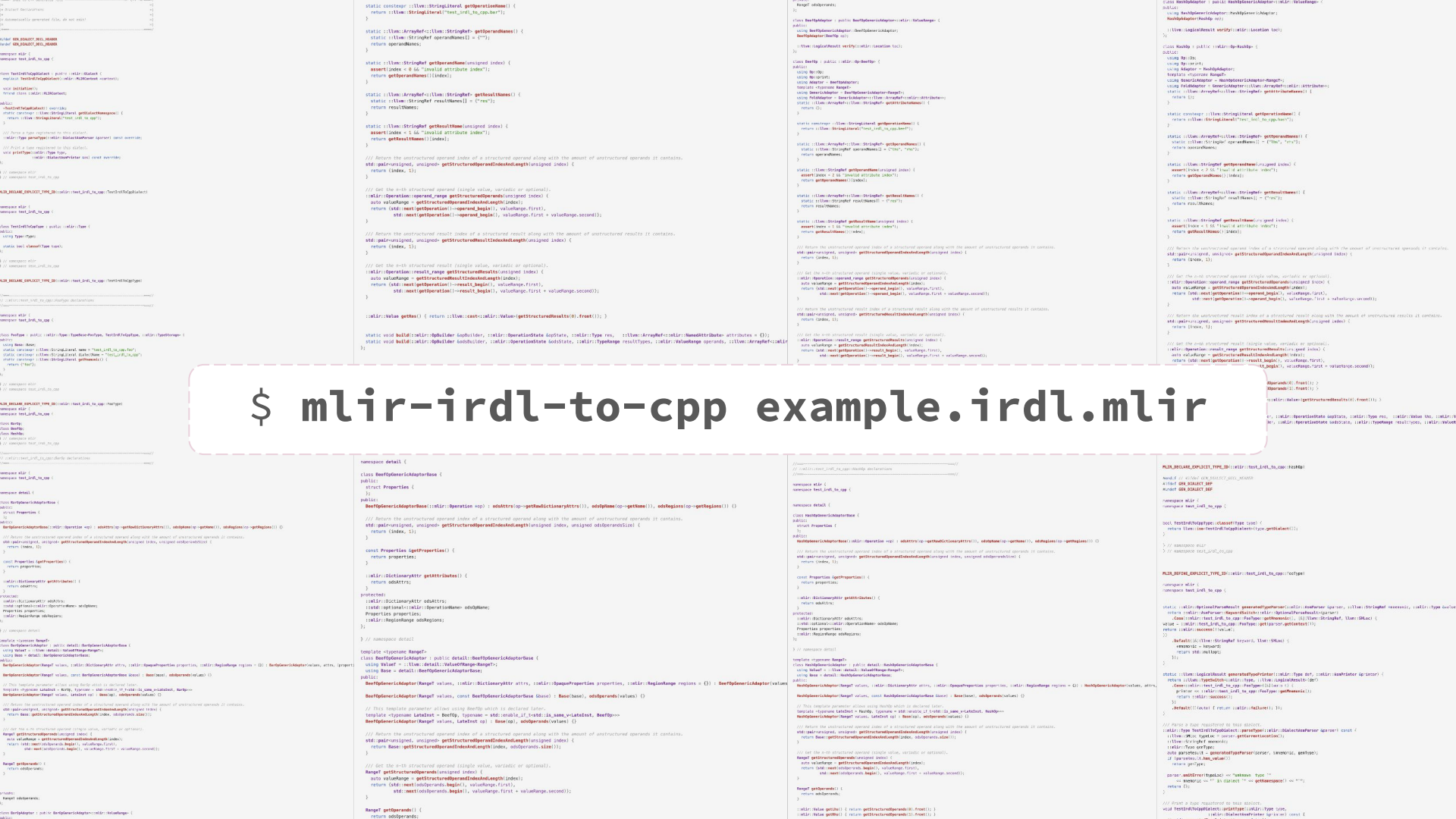
```
mlir-opt --irdl-file=cmath.irdl.mlir example.cmath.mlir
```

IRDL dialects are loaded **dynamically**

IRDL d didn't work with the existing MLIR C++ infrastructure

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

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


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

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

CMakeLists.txt

```
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

Merged?

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

Open **hhkit** wants to merge 90 commits into `llvm:main` from `opencompl:hhkit/irdl-to-cpp`

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.

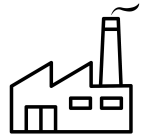
IRDL: IR Definition Language

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)
  }
}
```

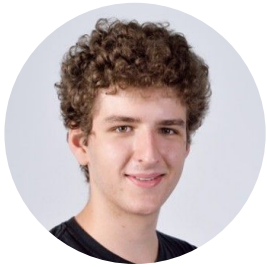
Dynamic



Generatable

IRDL Devs

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



Théo Degioanni
github: Moxinilian
discord: moxinilian

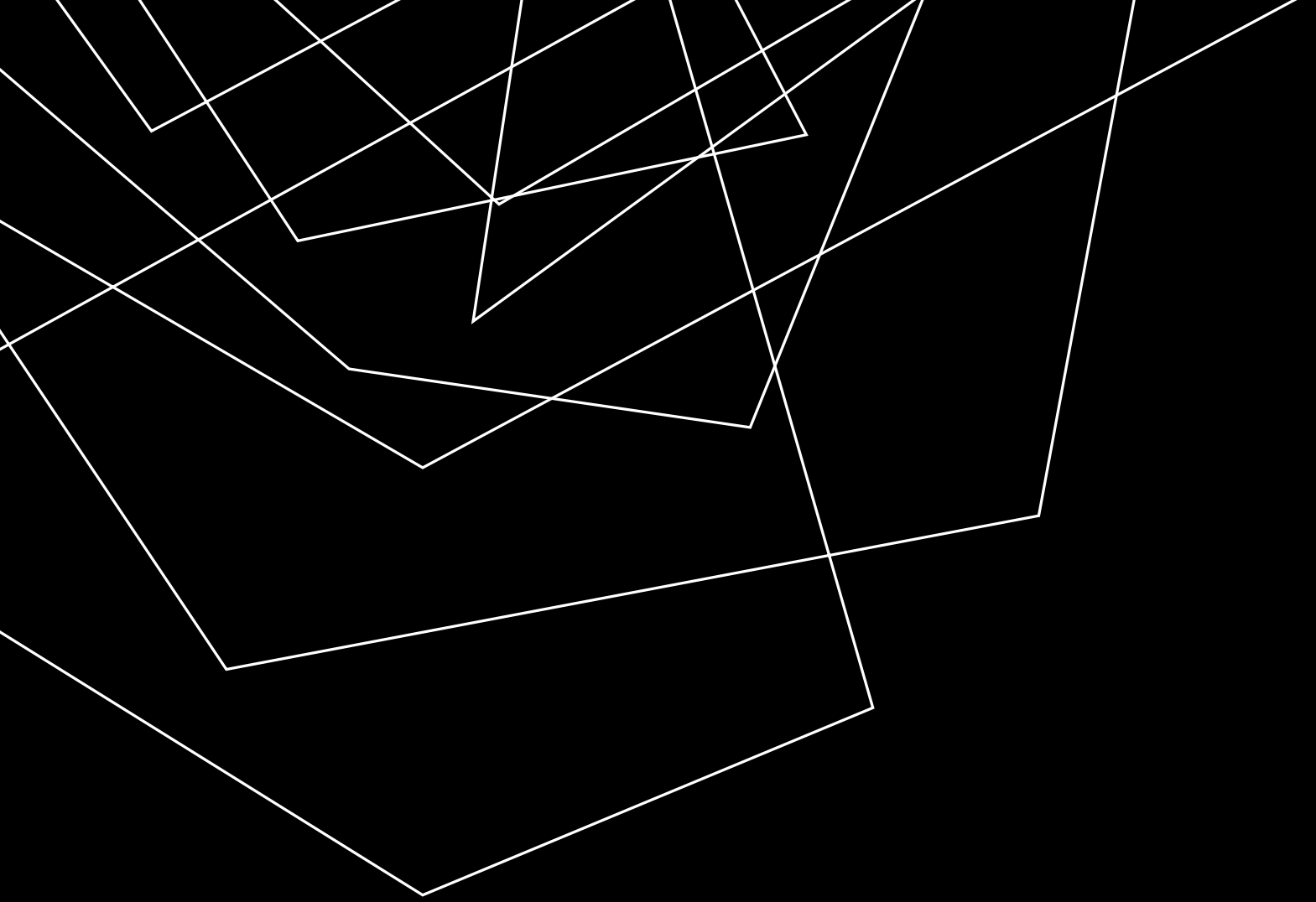
Ivan Ho
email: ivan@hhkit.dev
github: hhkit

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

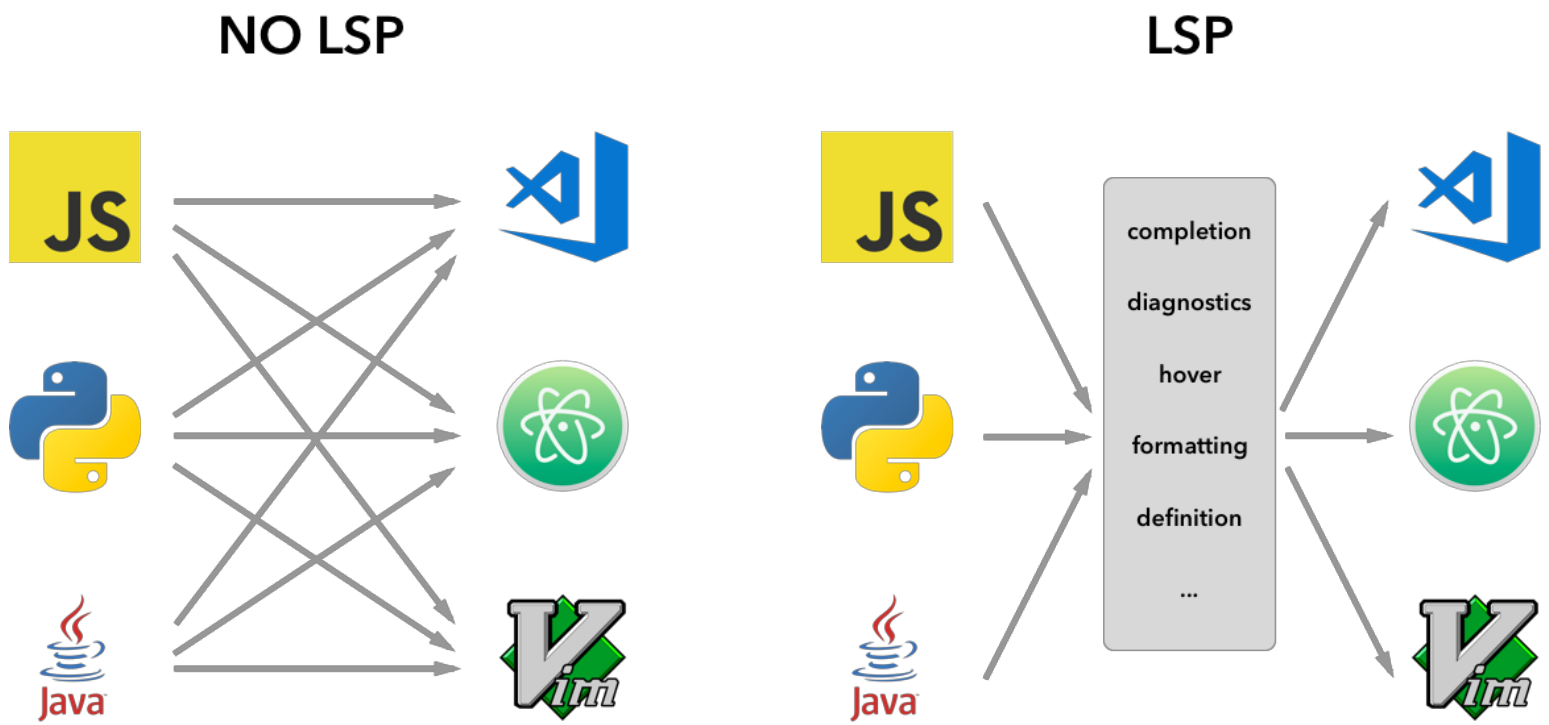
- Coding guidelines for using C++ in safety-related and critical systems
- Started as MISRA (Motor Industry Software Reliability Association) C++ in 2008
- Updated for newer C++ standards AUTOSAR (AUTOMOTIVE OPEN SYSTEM ARCHITECTURE) C++14
- Updated again as MISRA C++: 2023 for C++ 23

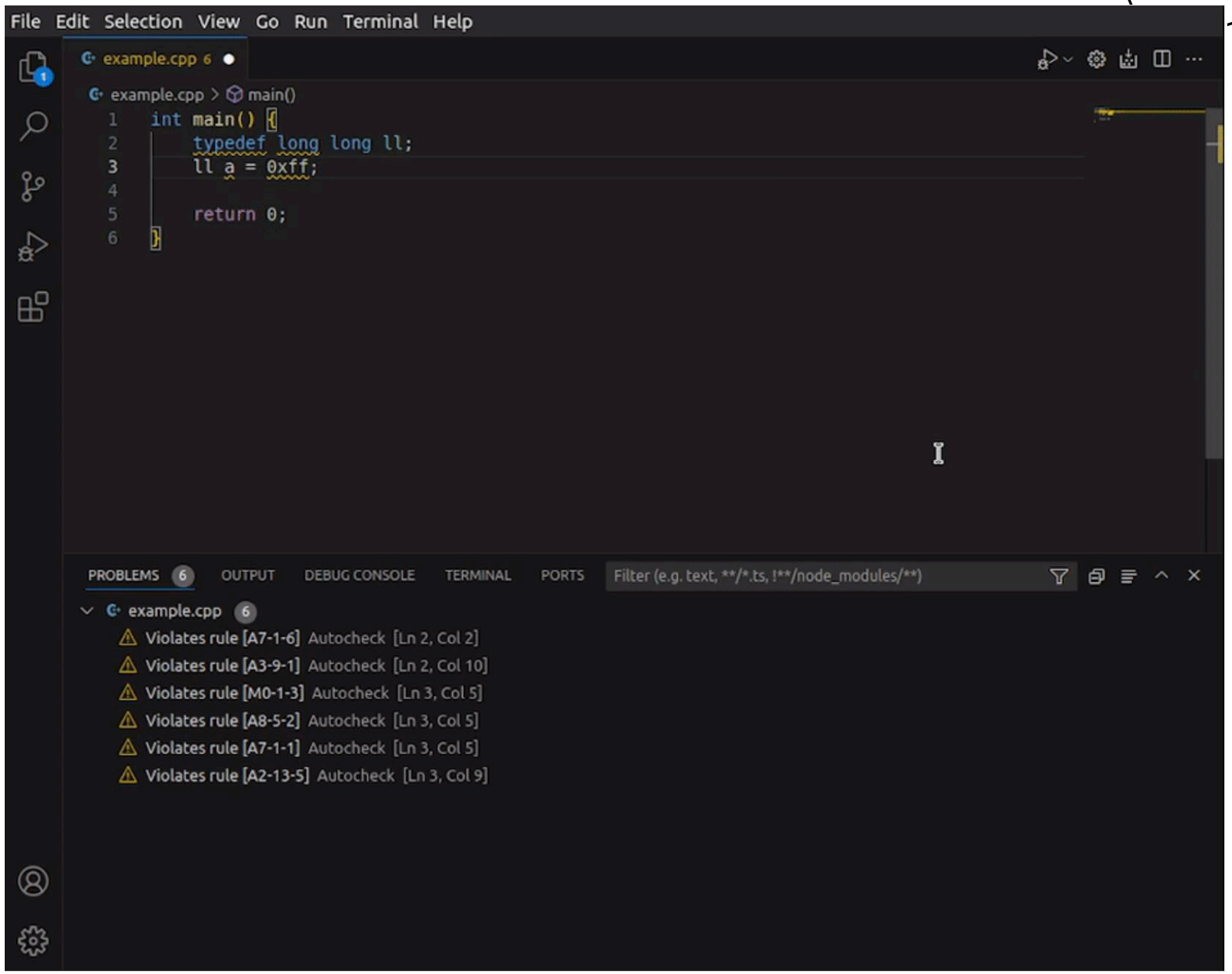
IMPLEMENTATION

- 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...



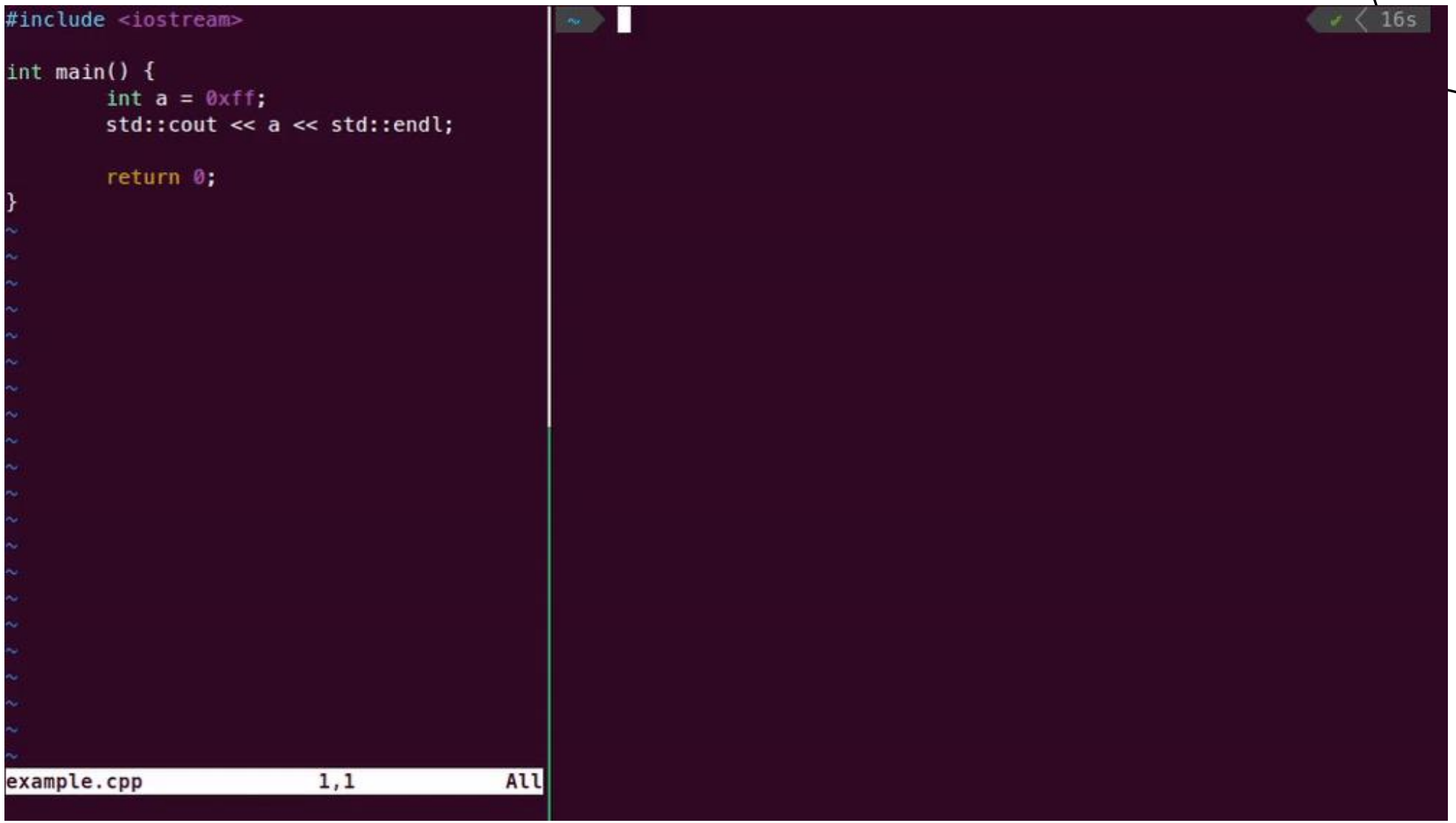

File Edit Selection View Go Run Terminal Help
example.cpp 6
example.cpp > main()
1 int main() {
2 typedef long long ll;
3 ll a = 0xff;
4
5 return 0;
6 }
PROBLEMS 6
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, **/*.ts, !**/node_modules/**)
example.cpp 6
Violates rule [A7-1-6] Autocheck [Ln 2, 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]

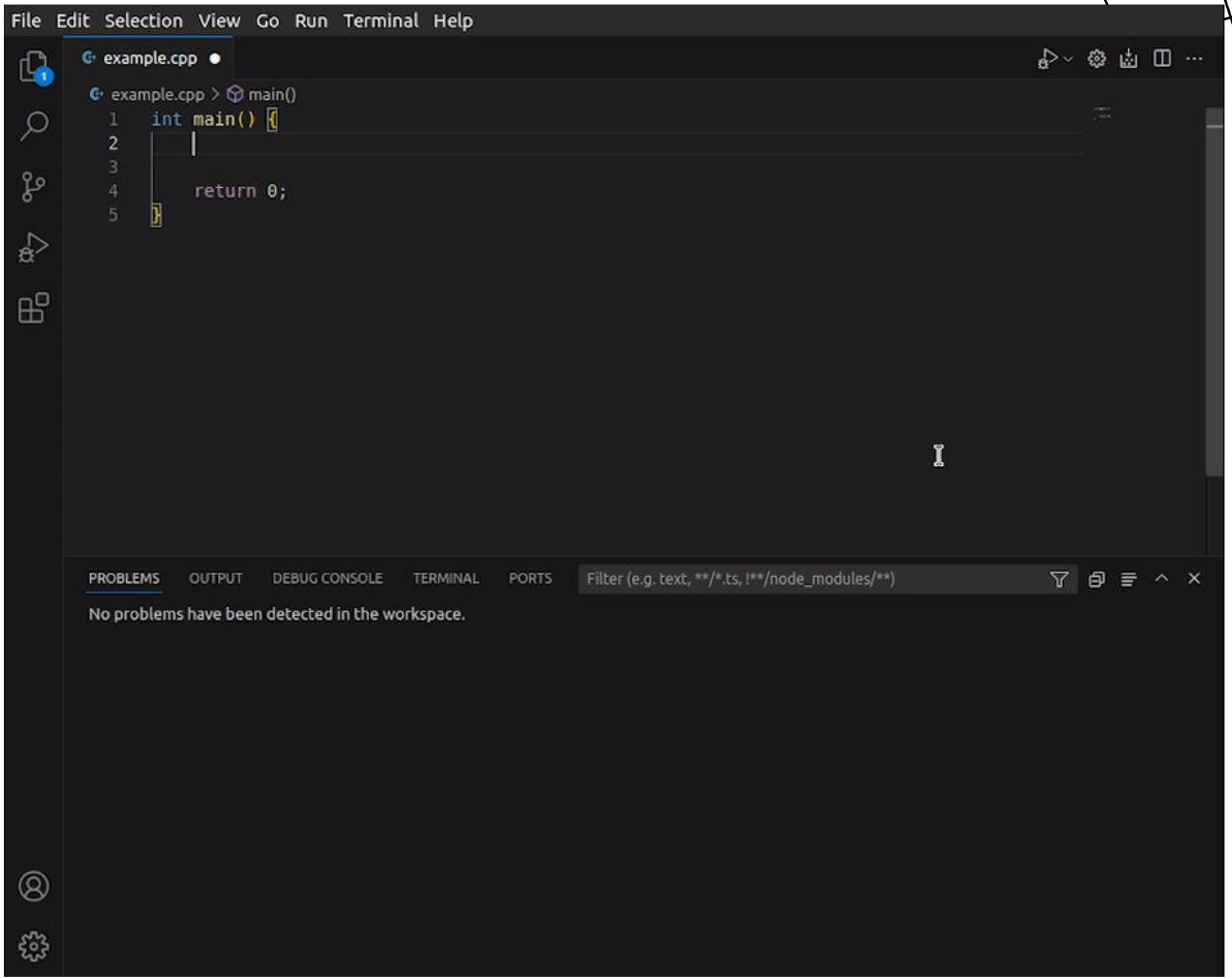
AI INTEGRATION

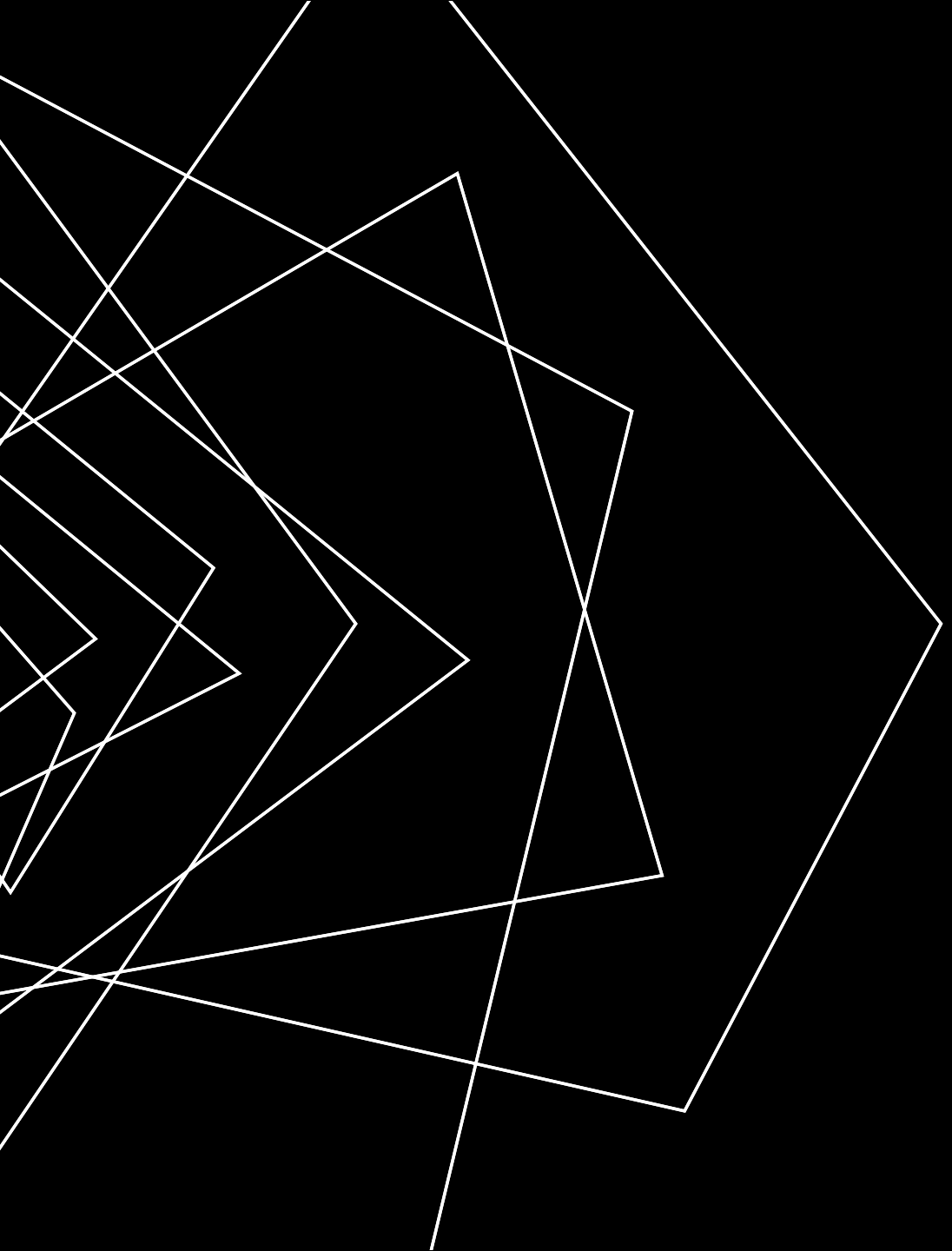
- Used for automatic code rewriting
- Local LLM instance using llama.cpp
- Llama.cpp enables inference of different LLMs (not just llama) locally either on a CPU, or GPU (CUDA, HIP)
- Used for rules where compiler technology cannot be applied – e.g. adding comments to functions that are missing it

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**




File Edit Selection View Go Run Terminal Help
example.cpp
example.cpp > main()
1 int main() {
2
3
4 return 0;
5 }
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Filter (e.g. text, **/*.ts, !**/node_modules/**)
No problems have been detected in the workspace.



THANK YOU

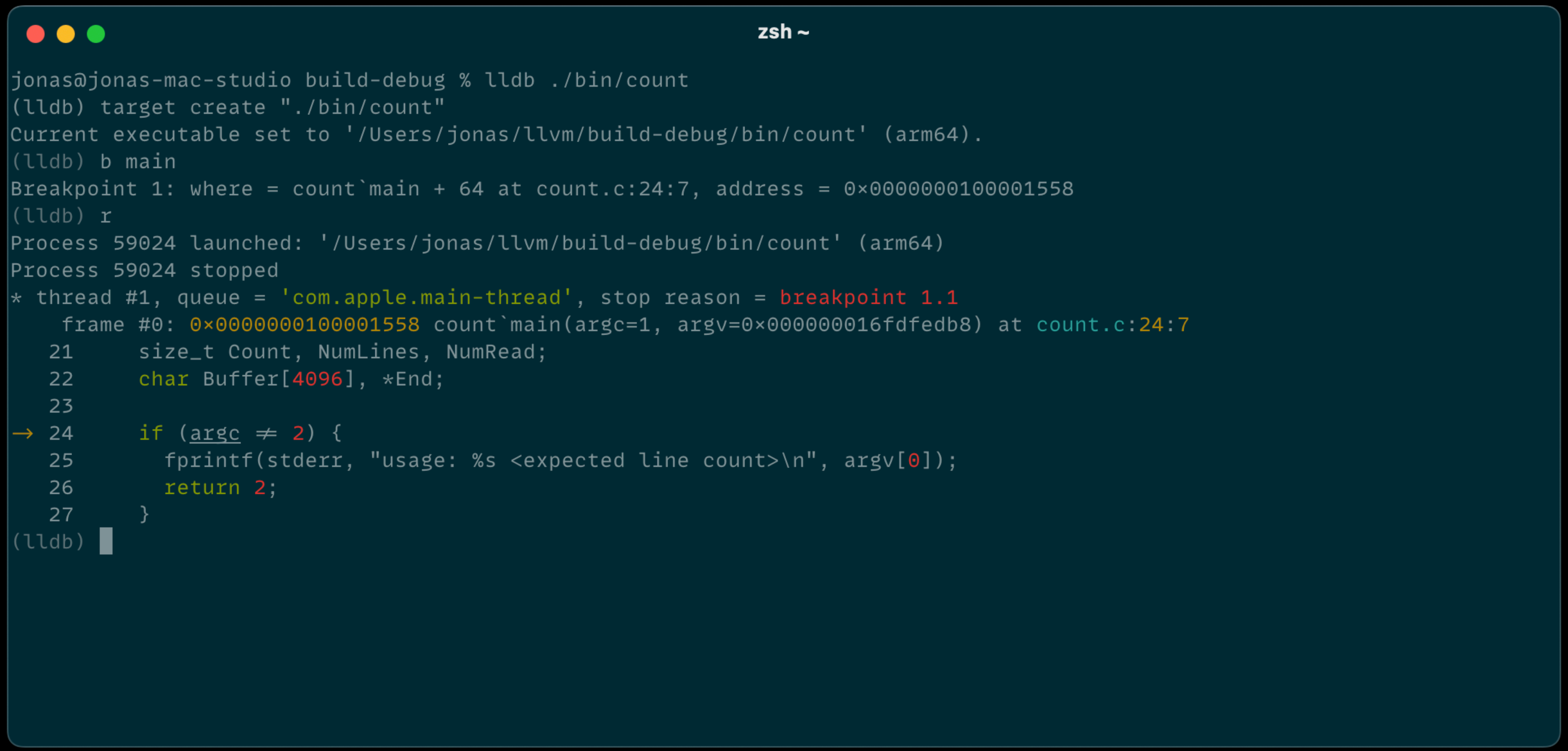
Developers' Meeting

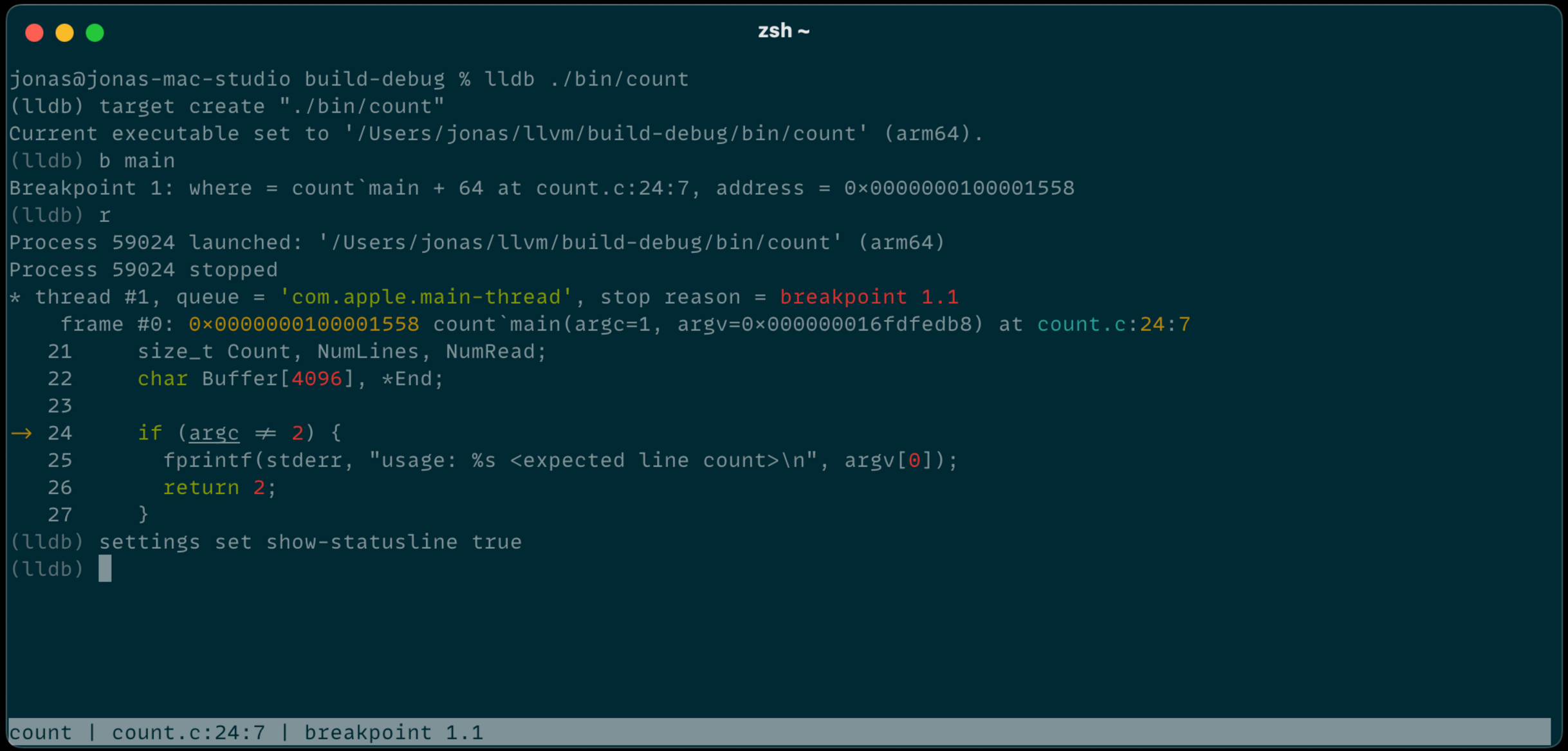
BERLIN 2025

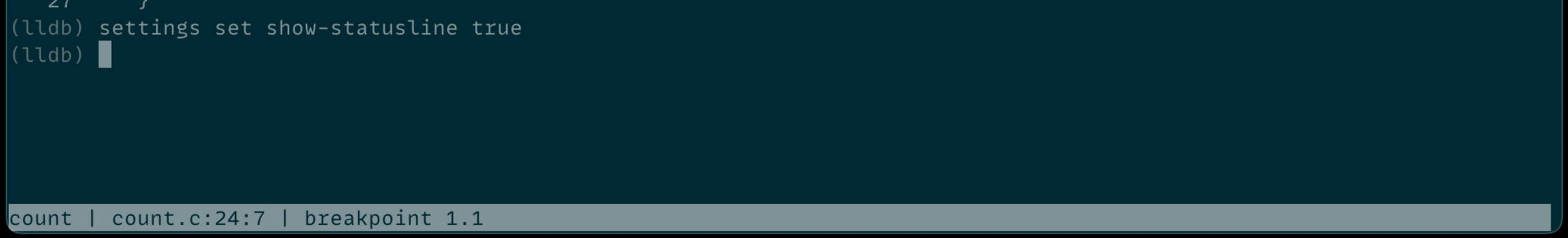
LLDB Statusline

Jonas Devlieghere

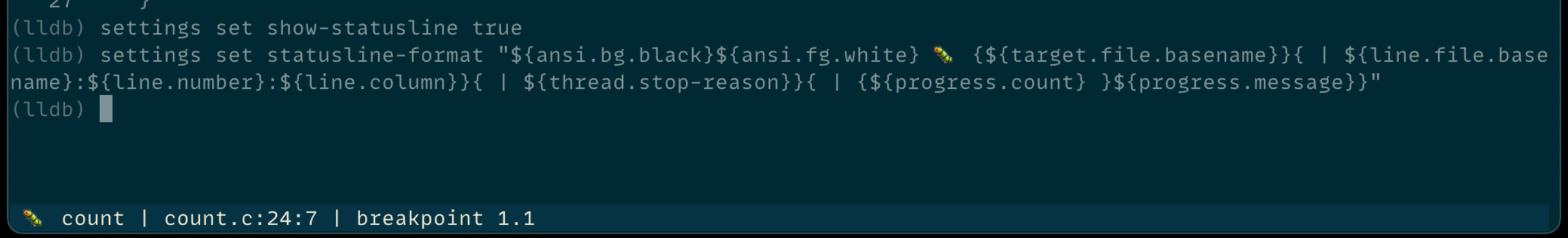
EuroLLVM 2025


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)

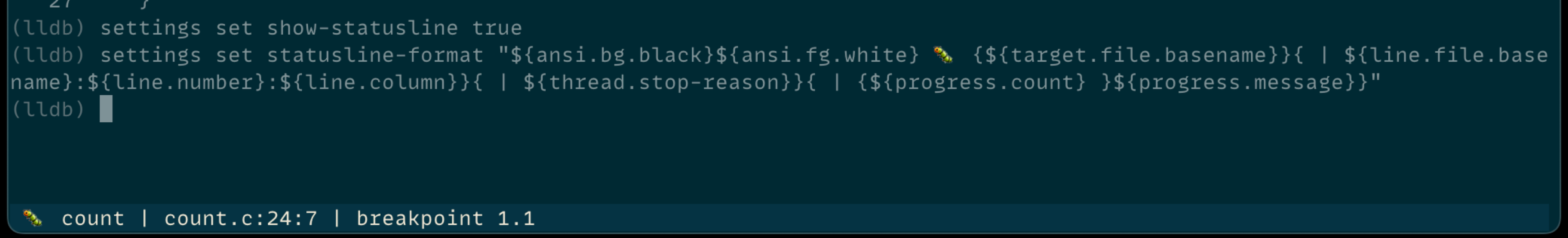

zsh ~
jonas@jonas-mac-studio build-debug % lldb ./bin/count
(lldb) target create "./bin/count"
Current executable set to '/Users/jonas/llvm/build-debug/bin/count' (arm64).
(lldb) b main
Breakpoint 1: where = count`main + 64 at count.c:24:7, address = 0x0000000100001558
(lldb) r
Process 59024 launched: '/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



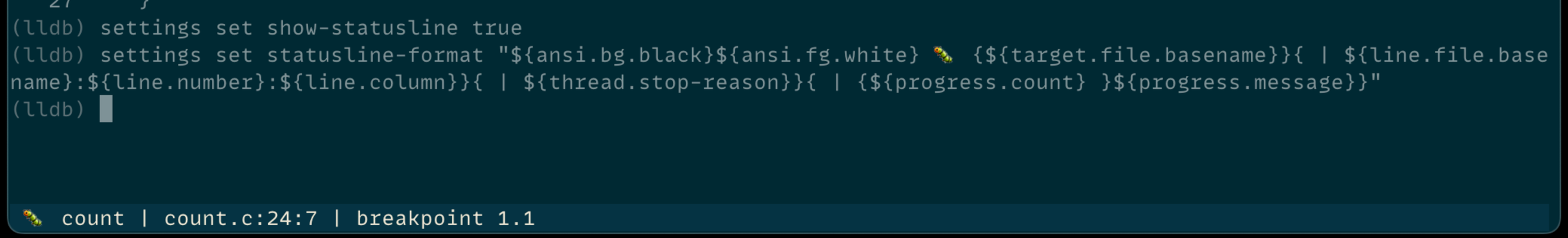

```
(lldb) settings show statusline-format
statusline-format (format-string) =
 "${ansi.negative}
  {${target.file.basename}}
  { | ${line.file.basename}:${line.number}:${line.column}}
  { | ${thread.stop-reason}}
  { | {${progress.count} }${progress.message}}"
```



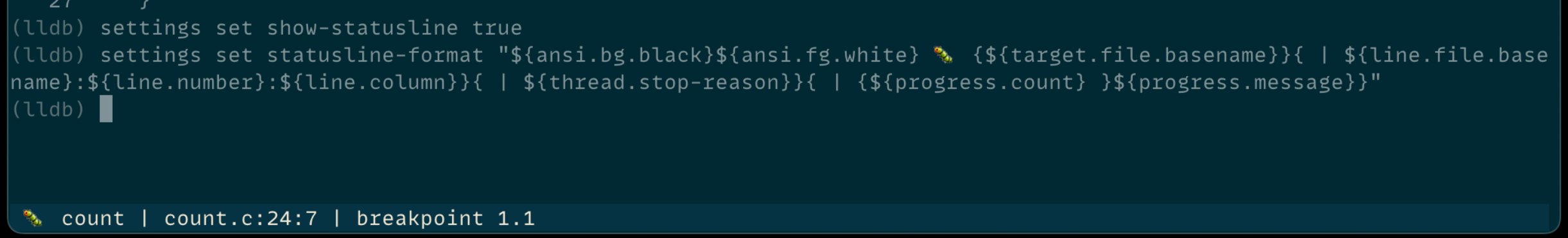

```
(lldb) settings show statusline-format
statusline-format (format-string) =
 "${ansi.bg.black}${ansi.fg.white} 🪲
  {${target.file.basename}}
  { | ${line.file.basename}:${line.number}:${line.column}}
  { | ${thread.stop-reason}}
  { | {${progress.count} }${progress.message}}"
```



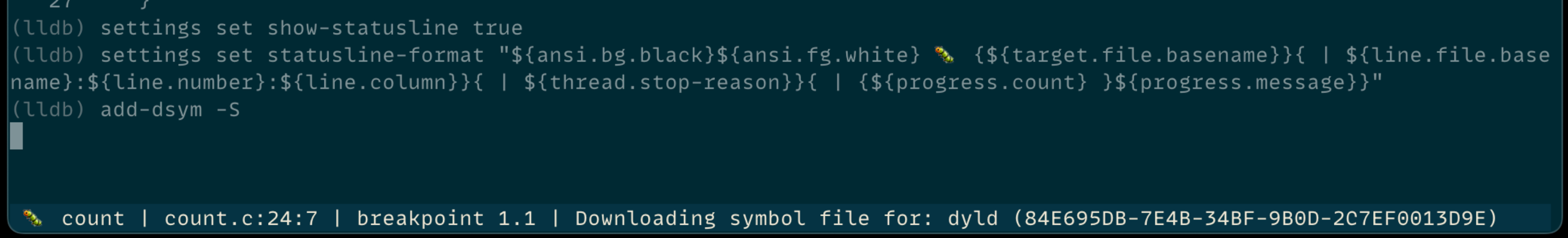

```
(lldb) settings show statusline-format
statusline-format (format-string) =
 "${ansi.bg.black}${ansi.fg.white} 🪲
   {${target.file.basename}}
   { | ${line.file.basename}:${line.number}:${line.column}}
   { | ${thread.stop-reason}}
   { | {${progress.count} }${progress.message}}"
```




```
(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}}"
```



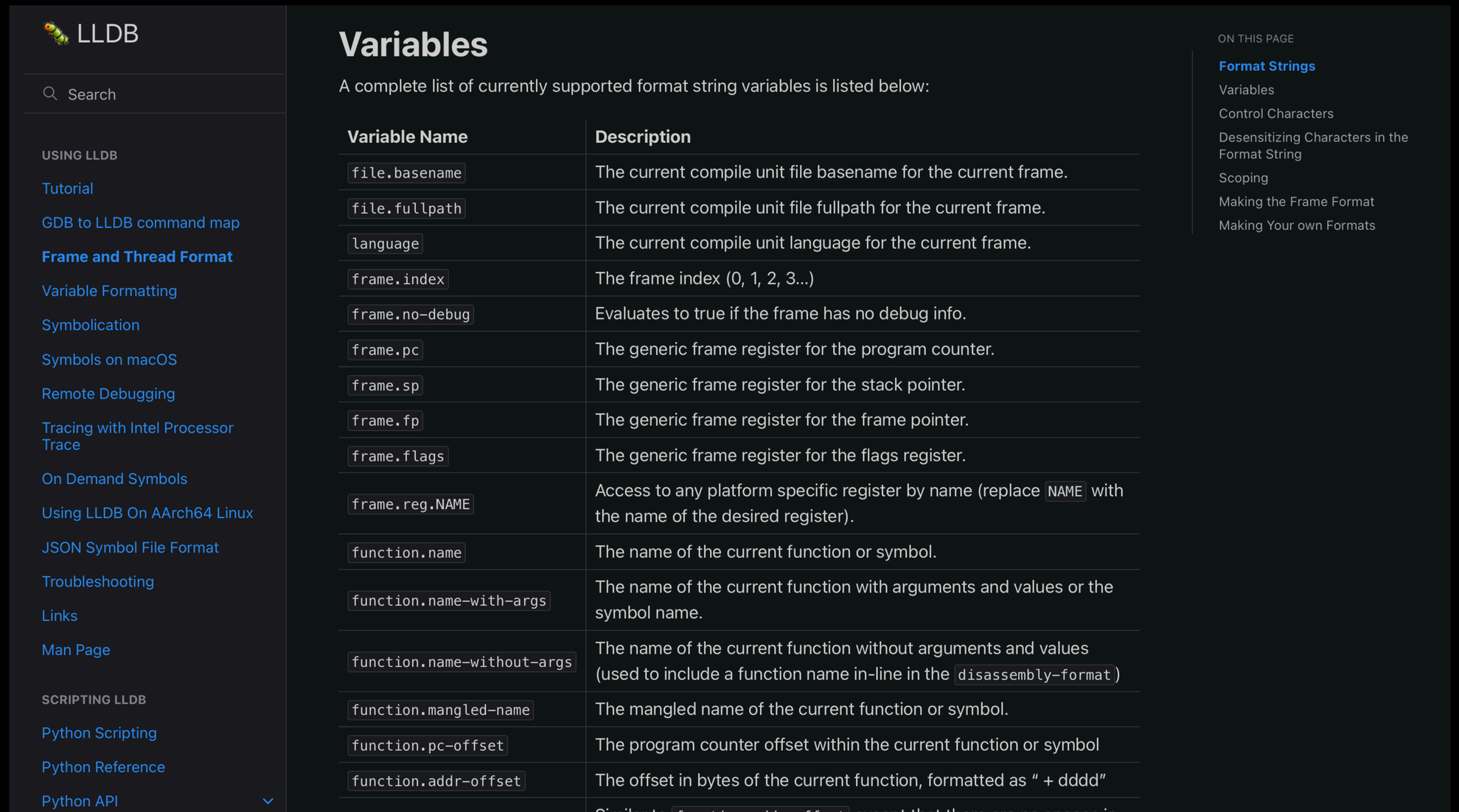

```
(lldb) settings show statusline-format
statusline-format (format-string) =
 "${ansi.bg.black}${ansi.fg.white} 🪲
  {${target.file.basename}}
  { | ${line.file.basename}:${line.number}:${line.column}}
  { | ${thread.stop-reason}}
  { | {${progress.count} }${progress.message}}}"
```




```
(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



LLDB

Search

USING LLDB

- Tutorial
- GDB to LLDB command map
- **Frame and Thread Format**
- Variable Formatting
- Symbolication
- Symbols on macOS
- Remote Debugging
- Tracing with Intel Processor Trace
- On Demand Symbols
- Using LLDB On AArch64 Linux
- JSON Symbol File Format
- Troubleshooting
- Links
- Man Page

SCRIPTING LLDB

- Python Scripting
- Python Reference
- Python API

Variables

A complete list of currently supported format string variables is listed below:

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

Developers' Meeting

BERLIN 2025

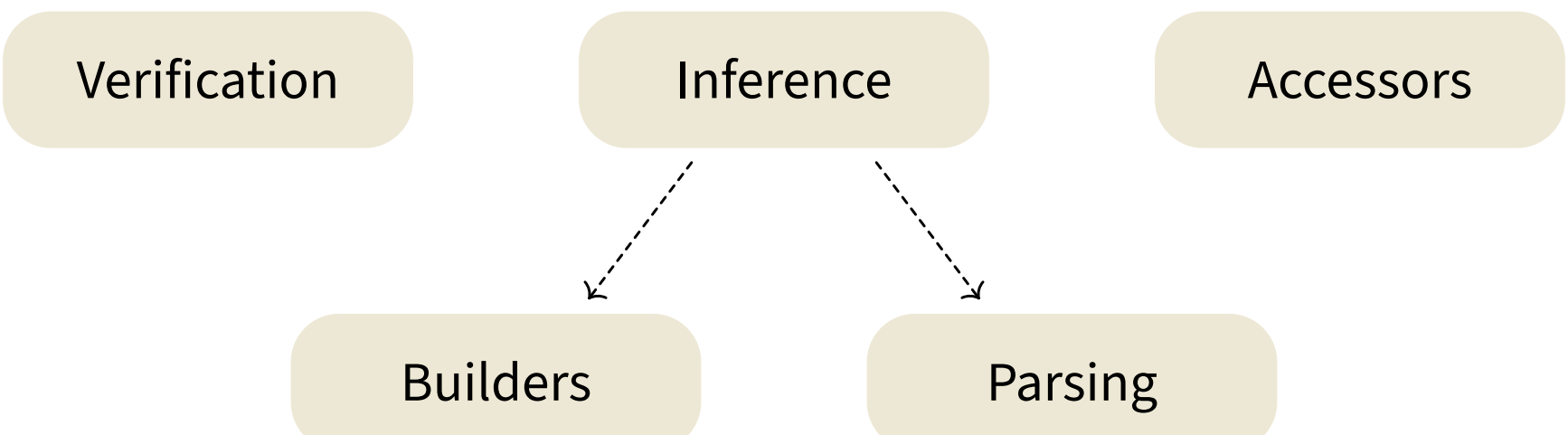
THE UNIVERSITY *of* EDINBURGH

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:

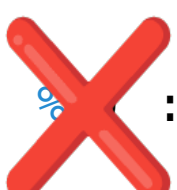
```
"arith.addi"(%1, %2) : (i32, i64) -> i8
```

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

linaro™

Arm Solutions at Lightspeed

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

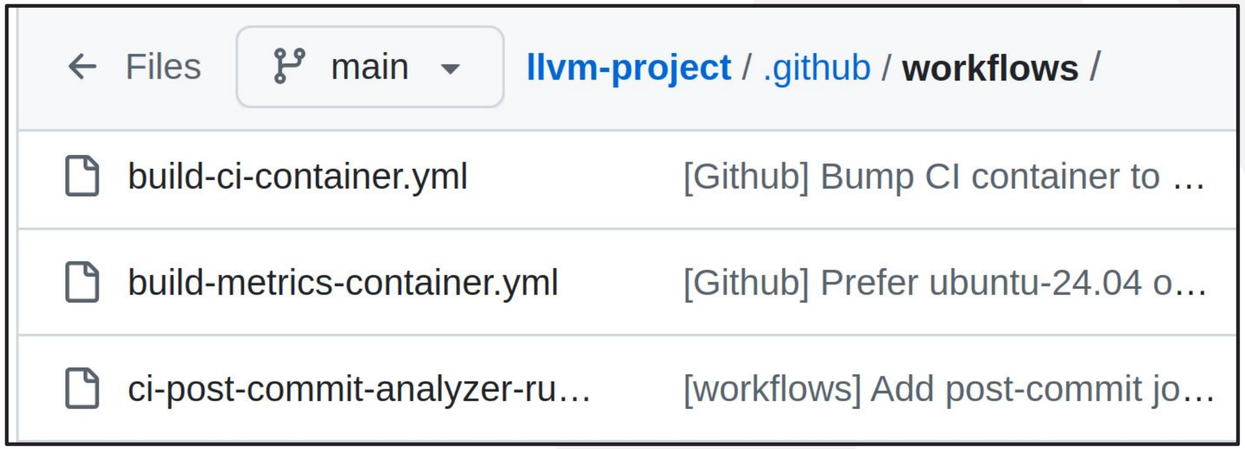
David Spickett, Arm / Linaro

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

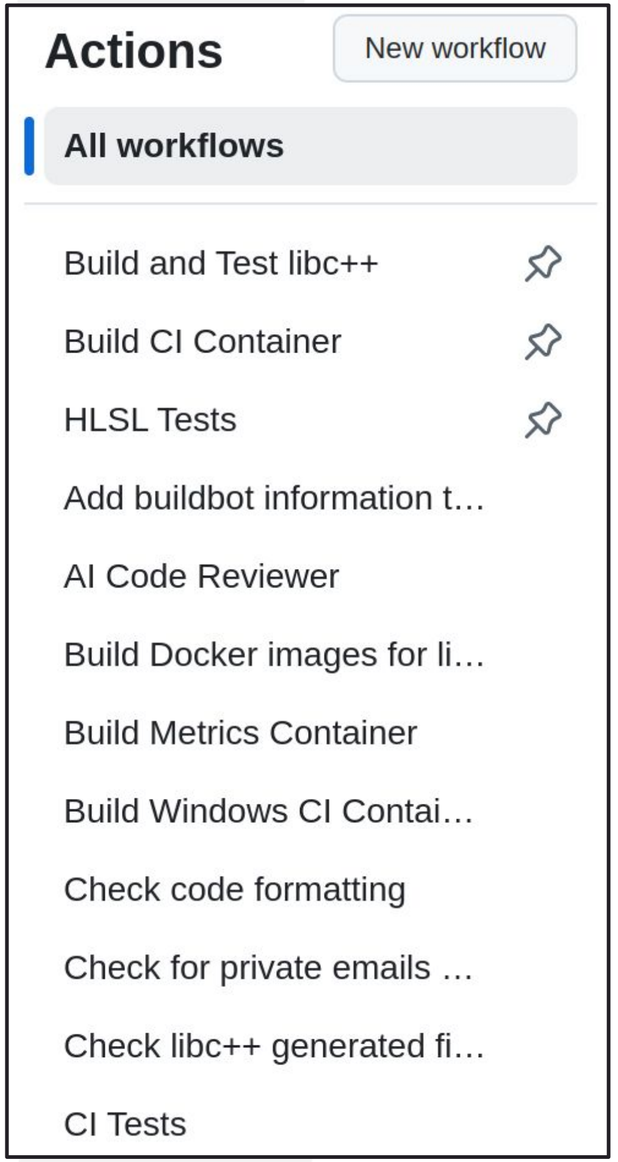
- Automate tasks for a GitHub repository
- “workflows” define what to do and when
- Run on remote “Runners” provided by GitHub
- Workflows defined in YAML
- `.github/workflows` in llvm-project



LLVM Workflows

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

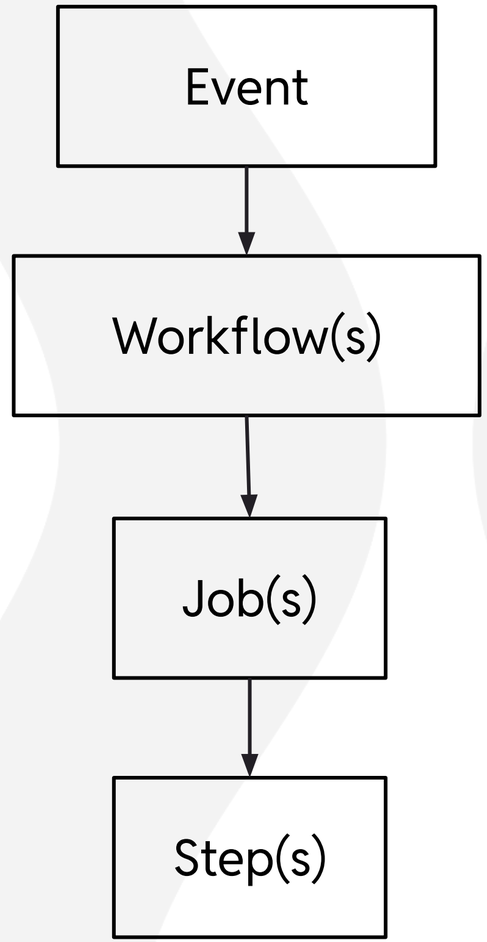
- Pull Request (PR) Labelling
- Checking code formatting
- Backporting to releases
- ...more examples later...



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
```



Note: YAML shown is for illustration purposes, not correctly formatted.

So You Want to Write a Workflow

You need:

- A Github account
- Patience

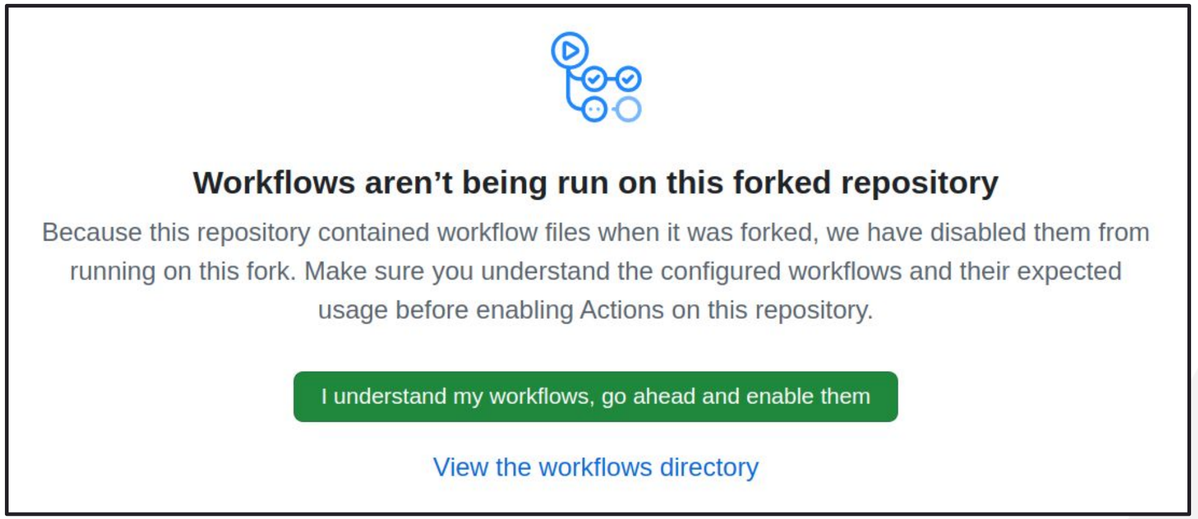
No special membership required, even the Runners are free to everyone*.

* 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

Workflows and Runners will be available by default.

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



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…

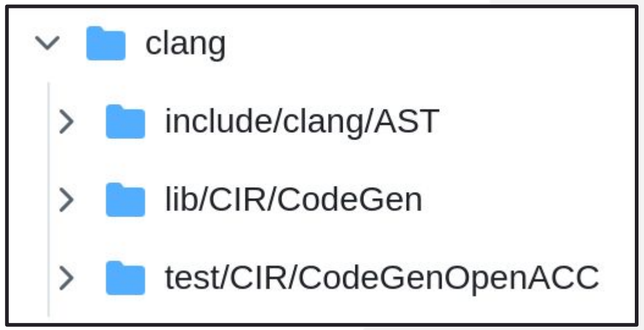
New PR Workflow - new-prs.yml

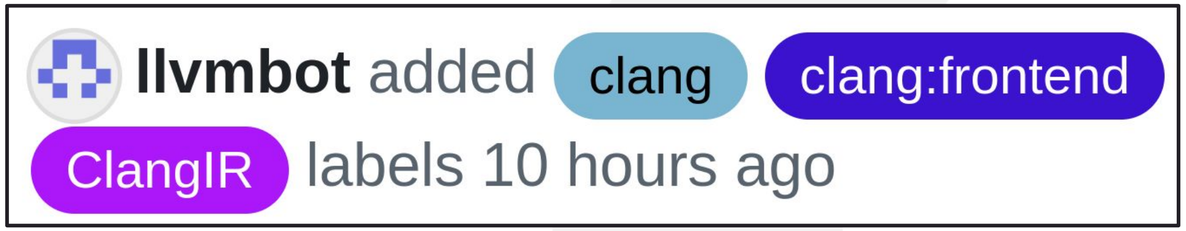
- Runs when a new PR is opened.

```
on:
 pull_request_target:
  types:
   - opened
   - reopened
   <...>
```

- Labels it based on the changes.

```
steps:
 - uses: actions/labeler
```





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)

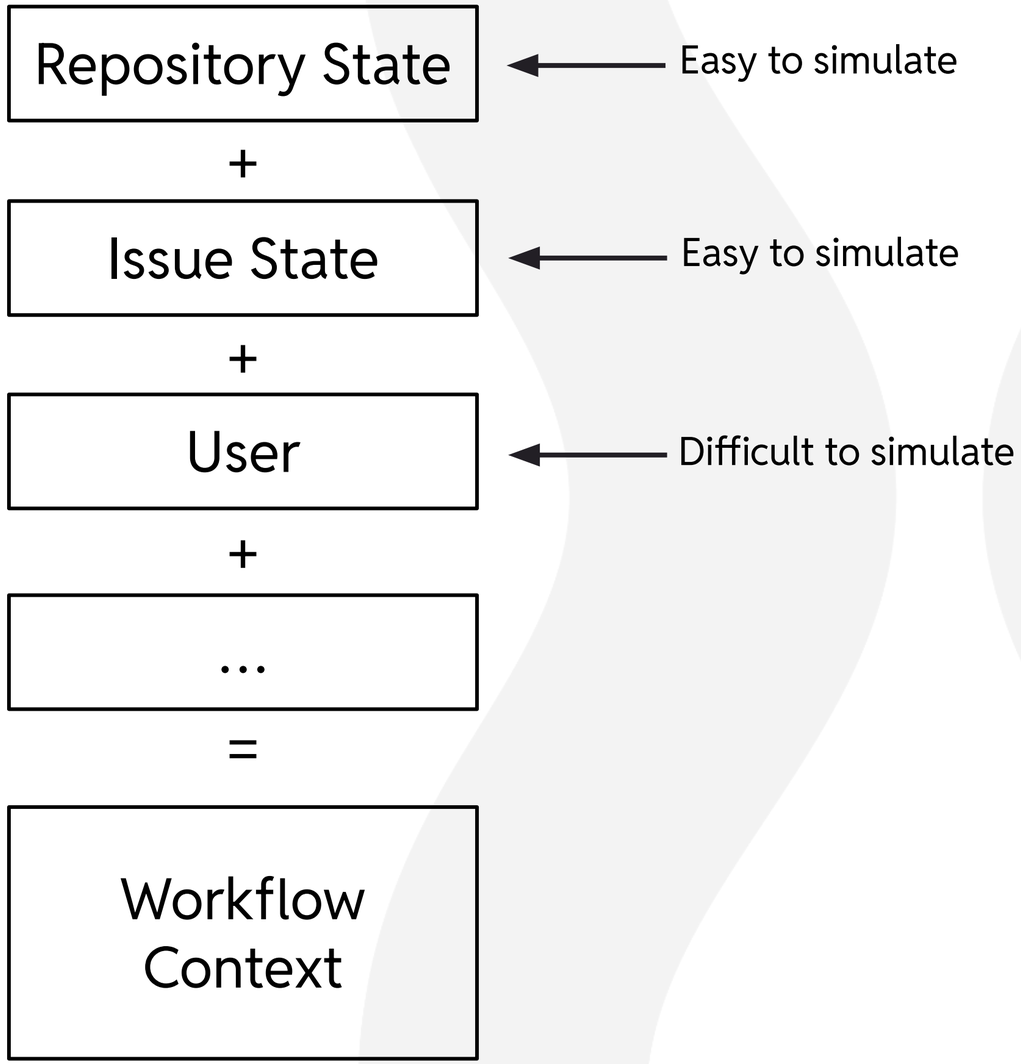
Write the rest of the workflow

- 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.



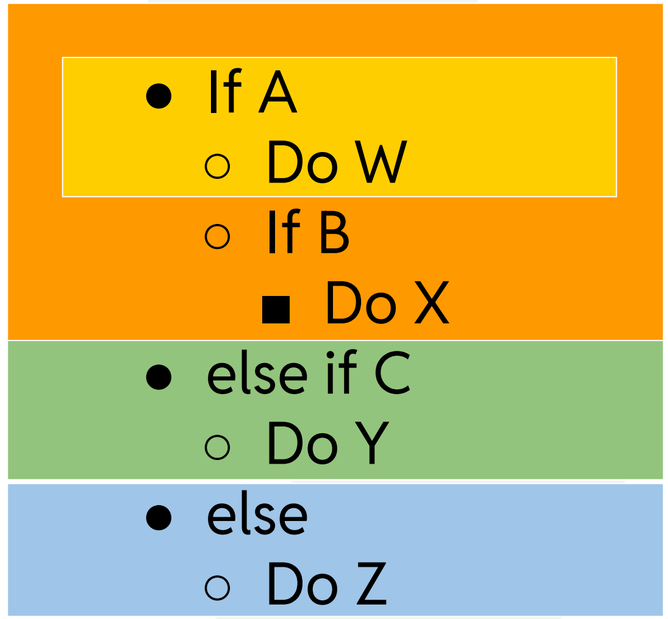
Testing

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') &&
  <...>
```

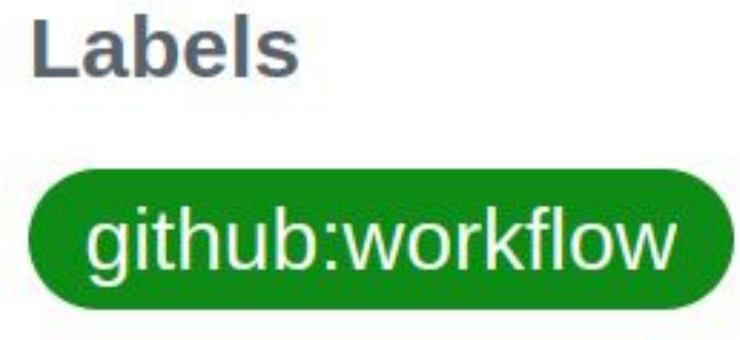
- Does the check.
- **Do** contribute this.

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.



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™ Arm Solutions at Lightspeed

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