2019 LLVM DEV MEETING

\Orchestrating a brighter world

NEC

NEC SX-Aurora as a Scalable Vector Playground

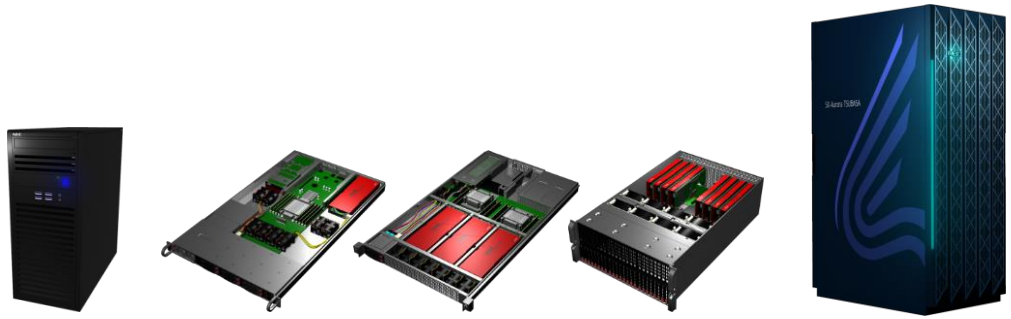
Kazuhisa Ishizaka, Yoshiyuki Ohno, Yuta Ideguchi, Erich Focht, Simon Moll

simon.moll@emea.nec.com

NEC Corporation

SX-Aurora TSUBASA Vector Computer

High performance computer ranged from deskside to cloud and supercomputer



Vector Processor on a Vector Engine PCIe accelerator card

Vector Engine (VE)

(PCIe accelerator card)

Vector Processor with 6 HBM2s

Theoretical Performance

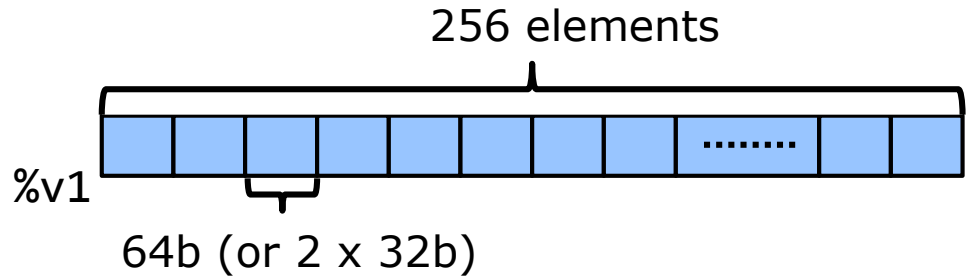
Computation:
FP32: 4.30 TFlops
FP64: 2.15 TFlops

Memory:
Capacity: 48GB
Bandwidth: 1.2 TB/s

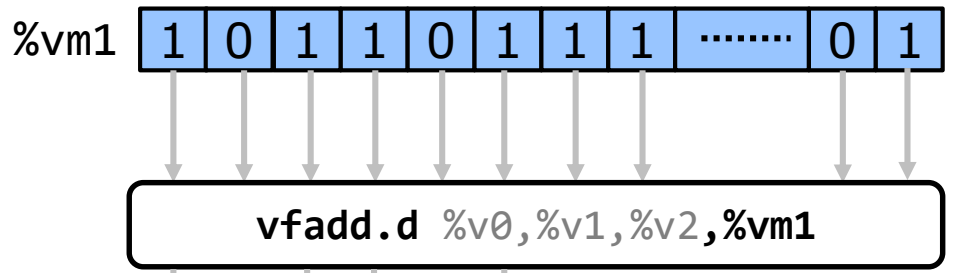
Vector Engine(VE) Scalable Vector ISA

Basics

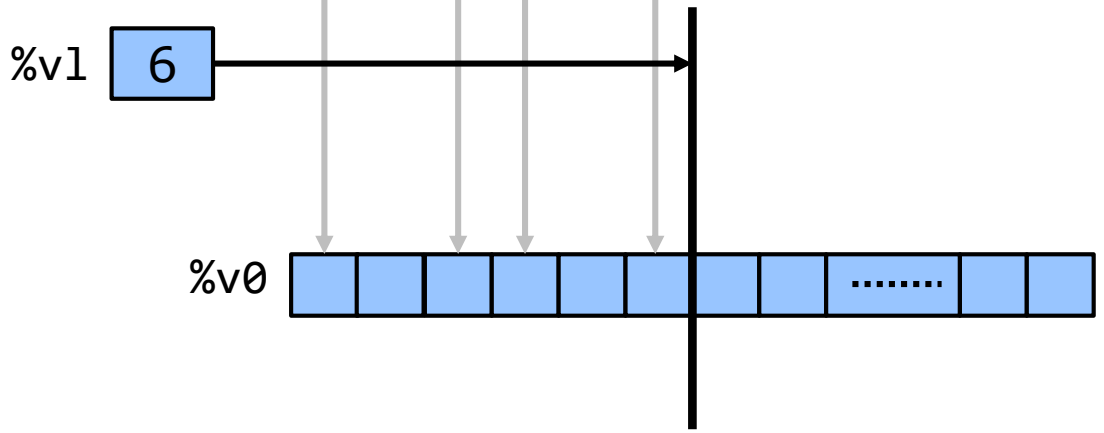
- **Wide vector registers (256 x 64bit)**
  - 64 Vector registers (%v***i***)



- **Full vector predication**
  - 16 Vector mask registers (%vm***i***)
  - ***Explicit*** operand



- **(Active) Vector Length Register**
  - One, global, VL Register
  - ***Implicit*** dependence



Vector Engine Challenges in LLVM

1) Implicit dependency through VL register

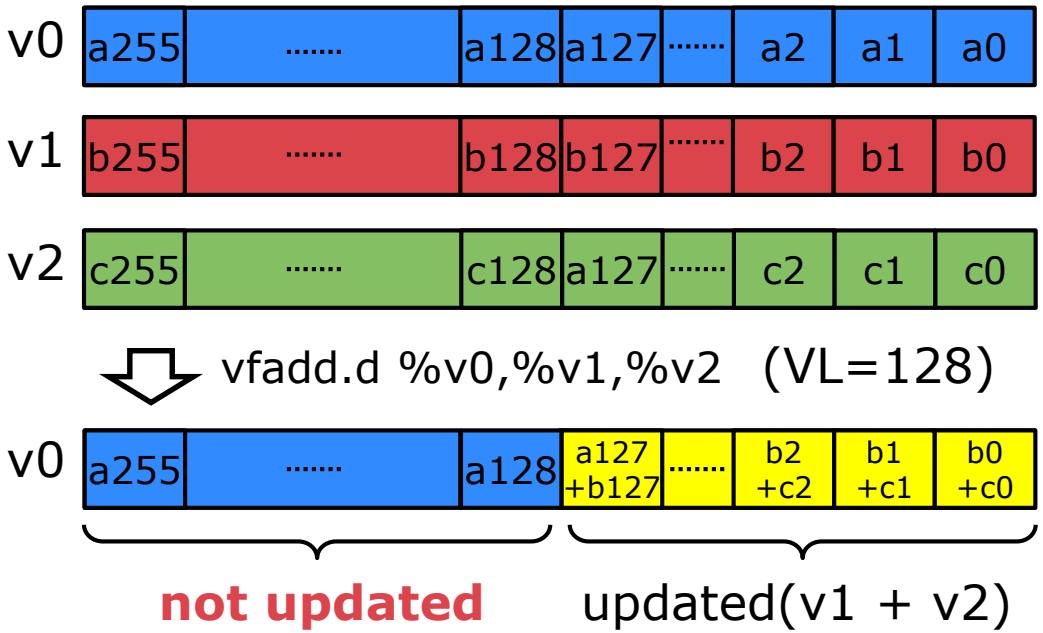
```
lvl %s37
vld %v3,8,%s0
vld %v4,8,%s1
vfmad.d %v3,%v3,%s2,%v4
```

define VL

use VL

How do we implement implicit def-use?

2) Partial update of a destination vector register



How do we introduce partial update?

Our Solution: IR and Backend

1. Vector IR based on LLVM-VP

- VL as parameter

```
$v3 = vfadd.d $v4,$v5,$pt,$vl
# for i=0,256
#     v3[i] = i < vl ? v4[i] + v5[i] : pt[i]
```

+

2. Automatic LVL generation in backend

- Inserting LVL instruction from $vl argument in IR
- Minimizing LVL instruction by current VL inference

Status and Roadmap

Status

- LLVM-VE is available at https://github.com/sx-aurora-dev/llvm
- Scalar code backend + vector intrinsics
- Application: TensorFlow for SX-Aurora

Roadmap

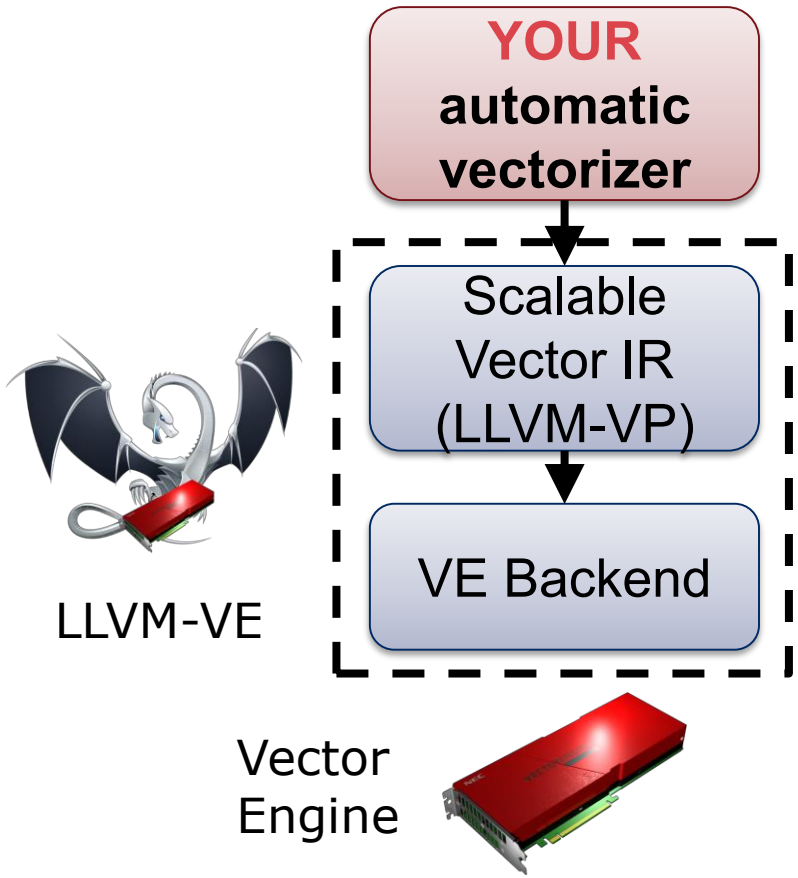
- Upstreaming! https://reviews.llvm.org/D69103
- Vector predication with LLVM-VP (D57504)

Welcome collaborators

- *for automatic vectorization, etc.*

Come see us at the poster session!

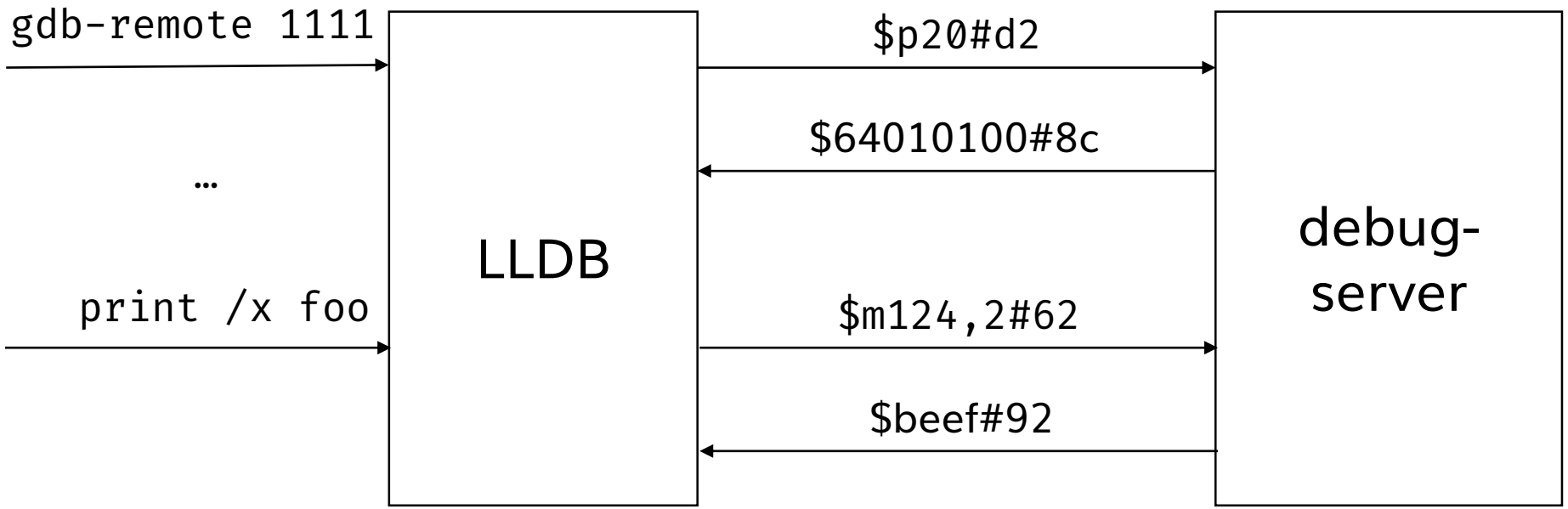
"NEC SX-Aurora as a Scalable Vector Playground"

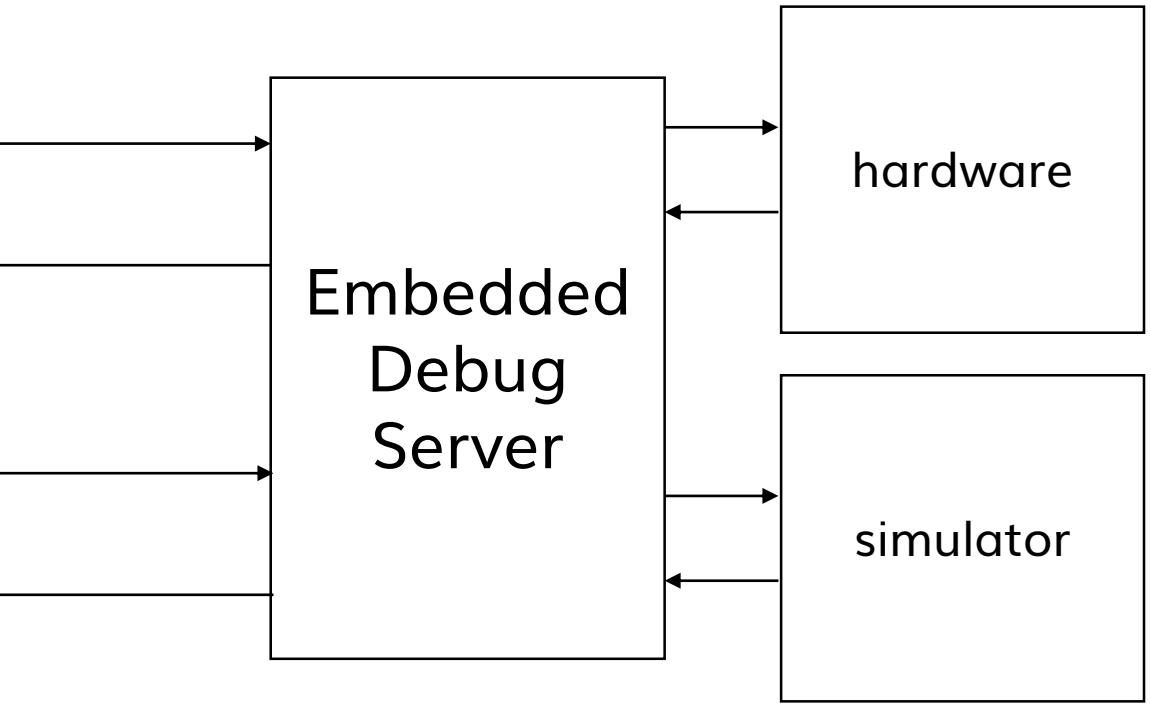


2019 LLVM DEV MEETING

A Unified Debug Server for Deeply Embedded Systems and LLDB

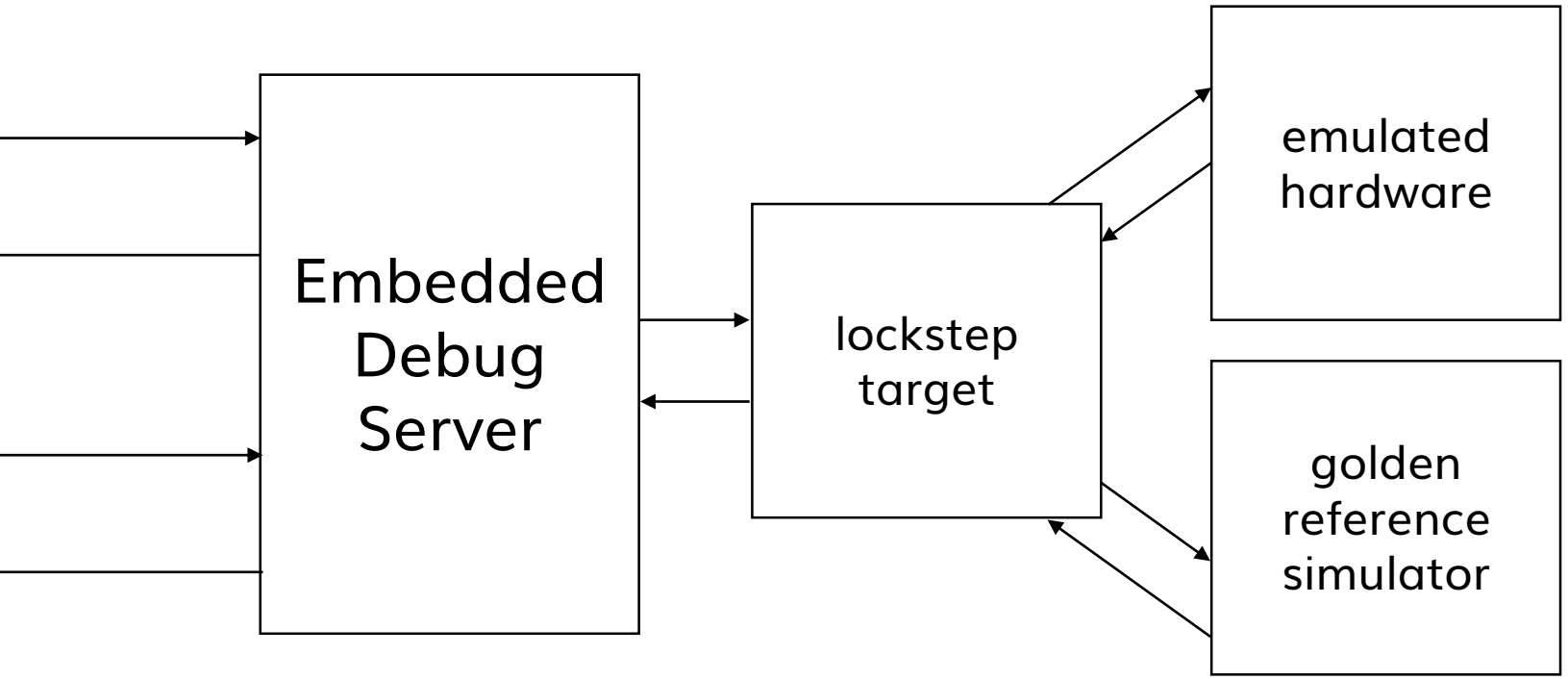
Simon Cook


gdb-remote 1111
…
print /x foo
LLDB
$p20#d2
$64010100#8c
$m124,2#62
$beef#92
debug-server




```
class ExampleTarget : public Target {
public:
  // Timers
  uint64_t getCycleCount() const;
  uint64_t getInstrCount() const;

  // Read-write memory and registers
  std::size_t readRegister(const int reg,
                           uint_reg_t &value);
  std::size_t writeRegister(const int reg,
                           const uint_reg_t value);
  std::size_t read(const uint_addr_t addr,
                   uint8_t *buffer,
                   const std::size_t size);
  std::size_t write(const uint_addr_t addr,
                    const uint8_t *buffer,
                    const std::size_t size);

  // Execution control
  bool prepare(const std::vector<ResumeType> &actions);
  bool resume(void) override;
  WaitRes wait(std::vector<ResumeRes> &results);
  ...
};
```

EMBECOSM®




```
std::size_t
LockstepTarget::readRegister(const int reg,
                             uint_reg_t &value) {
  uint_reg_t vL;
  uint_reg_t vR;
  std::size_t rL = _l->readRegister(reg, vL);
  std::size_t rR = _r->readRegister(reg, vR);

  // Report inconsistency server side.
  if ((rL != rR) || (vL != vR))
    std::cerr << "Lockstep error: register"
              << " inconsistency."
              << std::endl;


  // TODO: Allow this to be configured.
  // For now we just choose the left value
  // since readRegister cannot fail.
  value = vL;
  return rL;
}
```

EMBECOSM®

Lockstep

```
$ lldb
(lldb) gdb-remote 51000
Process 1 stopped
* thread #1, stop reason = signal SIGTRAP
frame #0: 0x0001015c
-> 0x1015c: addi a5, zero, 5
0x10160: mv t6, a5
0x10164: mv a5, zero
0x10168: mv a0, a5
(lldb) si
Process 1 stopped
* thread #1, stop reason = instruction step into
frame #0: 0x00010160
-> 0x10160: mv t6, a5
0x10164: mv a5, zero
0x10168: mv a0, a5
0x1016c: lw s0, 12(sp)
(lldb) si
Process 1 stopped
* thread #1, stop reason = signal SIGSYS
frame #0: 0x00010164
-> 0x10164: mv a5, zero
0x10168: mv a0, a5
0x1016c: lw s0, 12(sp)
0x10170: addi sp, sp, 16
```

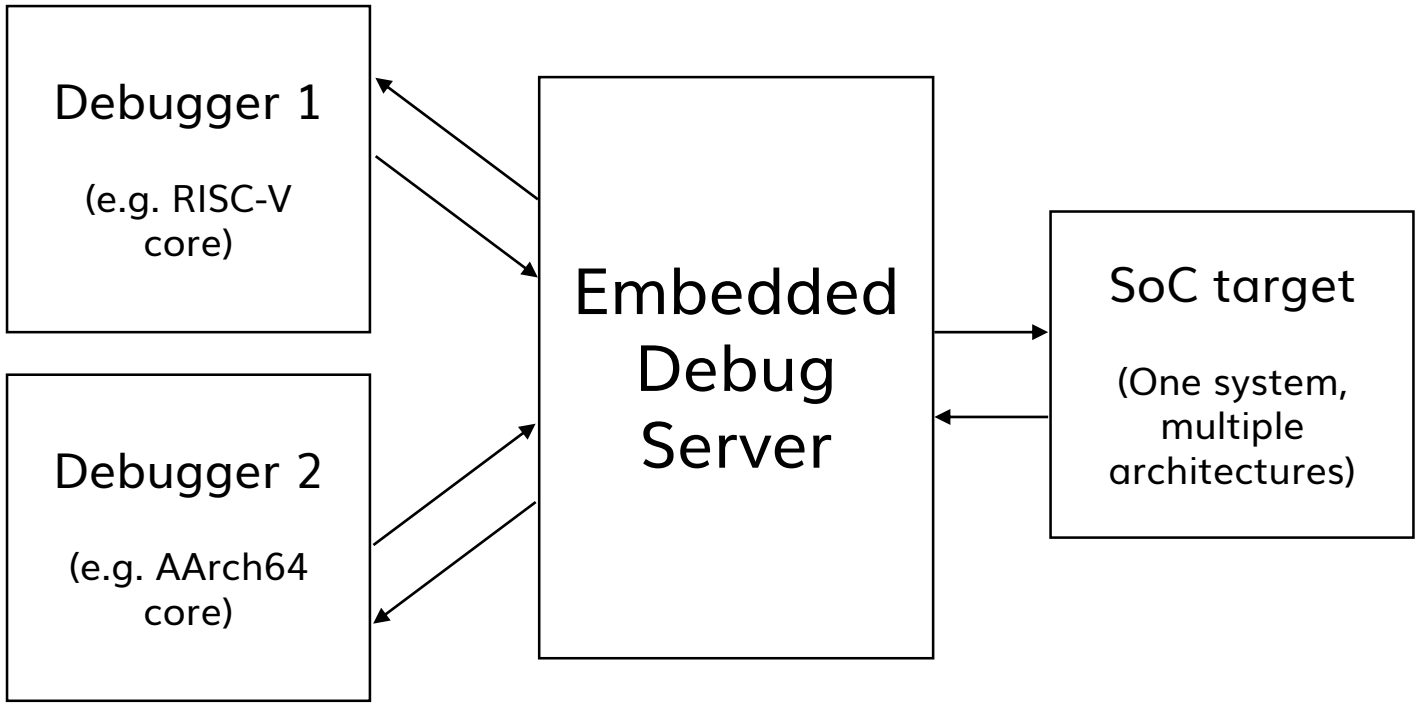
SIGSYS returned, indicating divergence

```
(lldb) register read --all
general:
x0 = 0x00000000
x1 = 0x000100d8
x2 = 0xfffffff0
x3 = 0x00011da8
< ... >
x28 = 0x00000000
x29 = 0x00000000
x30 = 0x00000000
pc = 0x00010164
1 registers were unavailable.

lockstep-left:
x31-left = 0x00000005
32 registers were unavailable.

lockstep-right:
x31-right = 0x00000004
32 registers were unavailable.
```

x31 not shown, since both targets disagree on value

both targets x31 shown, allowing investigation into divergence

EMBECOSM®


Debugger 1
(e.g. RISC-V core)
Debugger 2
(e.g. AArch64 core)
Embedded Debug Server
SoC target
(One system, multiple architectures)

Questions?

www.embecosm.com

2019 LLVM DEV MEETING

Speculative Compilation in ORC

ORC

- LLVM Modular Just in Time Compilation Library
- Custom compilers, program representations...
- Supports concurrent compilation

JIT Variants

- Eager JIT - high startup time, zero compiler interactions at runtime

- Lazy JIT - ⚡ startup time, compilation overhead on first call

Can we do better? Can we have benefits of two worlds? 

Let’s guess it!

```
void Driving(Signal S)

        switch(S){

        case red:stop();tweet();

        break;

        case yellow:

        like_reply_to_a_tweet();break;

        case green :think_next_tweet();

        break;

}}
```

What if we guess the signal’s outcome and do action!

Likewise, we guess control flow path and compile the *likely* functions before calling them.

Let’s guess it!

```
void Driving(Signal S)

        switch(S){

        case red:stop();tweet();

        break;

        case yellow:

        like_reply_to_a_tweet();break;

        case green :think_next_tweet();

        break;

}}
```

What if we guess the signal’s outcome and do action!

Likewise, we guess control flow path and compile the *likely* functions before calling them.

Speculative Decisions

- Compile only the most likely next executable functions
- Speculate based on CFG edge probabilities and hot blocks heuristics
- Implemented as SpeculateQuery function objects, you can try your own ideas 😊
    - Map<Function, LikelyFunctionSymbols> SpeculateQuery(Function& F);

➥ “Jump into JIT through IR Instrumentation”

Mix‘n’Match

```
ObjectLinkingLayer LinkLayer;

IRCompileLayer<...> CompileLayer(LinkLayer, ConcurrentCompiler);

IRSpeculationLayer SpeculateLayer(...,CompileLayer,Speculator,SpeculateQuery)

CompileOnDemandLayer<...> CODLayer(SpeculateLayer, ...);

CODLayer.addModule(Mod, MemMgr, SymResolver);

auto FooSym = CODLayer.findSymbol(“foo”, true);

auto Foo = reinterpret_cast<int(*)()>(FooSym.getAddress());

int Result = Foo(); // <-- Call foo’s stub.
```

Before

```
define dso_local void @Driving(i32 %Signal)

  switch i32 %Signal,label %exit[i32 0,label %red

  i32 1, label %yellow i32 2, label %green]

red:

  call void @stop()  call void @tweet()

yellow:

  call void @like_reply_to_a_tweet()

green:

  call void @think_next_tweet()
```

After...

```
@__orc_speculator = external global %Class.Speculator

declare void @__orc_speculate_for(%Class.Speculator* %0, i64 %1)

define dso_local void @Driving(i32 %0) #0 {

  call void @__orc_speculate_for(%Class.Speculator* @__orc_speculator,

  i64 ptrtoint (i32 ()* @Driving to i64)) // Jump into JIT 👉

 ...

  switch i32 %3, label %7 [ i32 0, label %Red

  i32 1, label %Yellow

  i32 2, label %Green]

...
```

Performance

7× Speed-Down 💔💔

Multiple Jumps into JIT ❌

ExecutionSession::Lookup’s are not free✅

We want Performance👊

- Relatively easy fix, guard the orc_speculate_for call
- Jump into JIT only on the first call
- This will give us - what we want

```
@__orc_speculate.guard.for.main = internal local_unnamed_addr global i8 0, align 1

define dso_local void @Driving(i32 %0) {

__orc_speculate.decision.block:

  %guard.value = load i8, i8* @__orc_speculate.guard.for.main

  %compare.to.speculate = icmp eq i8 %guard.value, 0

  br i1 %compare.to.speculate, label %__orc_speculate.block, label %program.entry

__orc_speculate.block:

  call void @__orc_speculate_for(%Class.Speculator* @__orc_speculator,

  i64 ptrtoint (i32 ()* @Driving to i64))

  store i8 1, i8* @__orc_speculate.guard.for.main

  br label %program.entry
```

Performance

We see significant speedup with our proof-of-concept speculative jit

For SPEC 403.gcc benchmark, reduce exec time from 17.4 seconds to 10.5 seconds (4 threads) 😃

What’s Next?

- Finish dynamic profiling support to collect branch probability information
- Reduce the scope of speculation region in a function
- Implementing more SpeculateQueries
- Performance tuning

Thank you so much - Lang Hames and David Blaikie

Thank you LLVM Foundation and Conf Sponsors.

2019 LLVM DEV MEETING

Loom

Weaving Instrumentation for Program Analysis

Brian Kidney (Presenter)
Jonathan Anderson
Memorial University

But Instrumentation is done, right?

Why another instrumentation tool

- There are lots of instrumentation tools
  - Intel Pin, XRay, CSI, DTrace…
- Most tools focus on performance
- We needed something different
  - No custom compiler frontend or backend
  - “Non-traditional” instrumentation
  - The ability to transform code when needed
- Our focus was security
  - We wanted something more general-purpose

What if you wanted to instrument every PAM Authentication?

Instrumenting PAM

And you could do it with this:

```
strategy: callout
dtrace: userspace
functions:
- callee: [entry]
  metadata:
    name: auth
    id: 1
  name: pam_authenticate
```

Instrumenting PAM

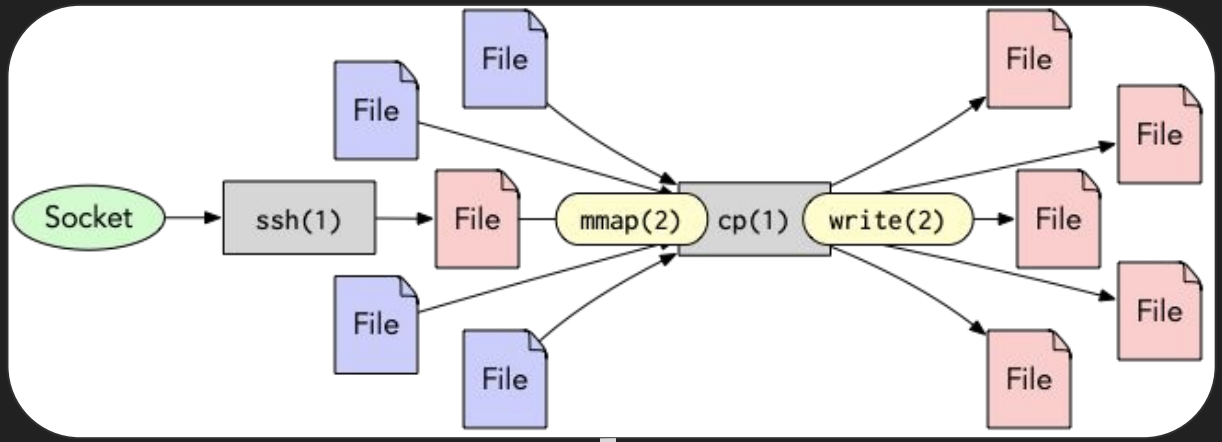
And you could do it with this:

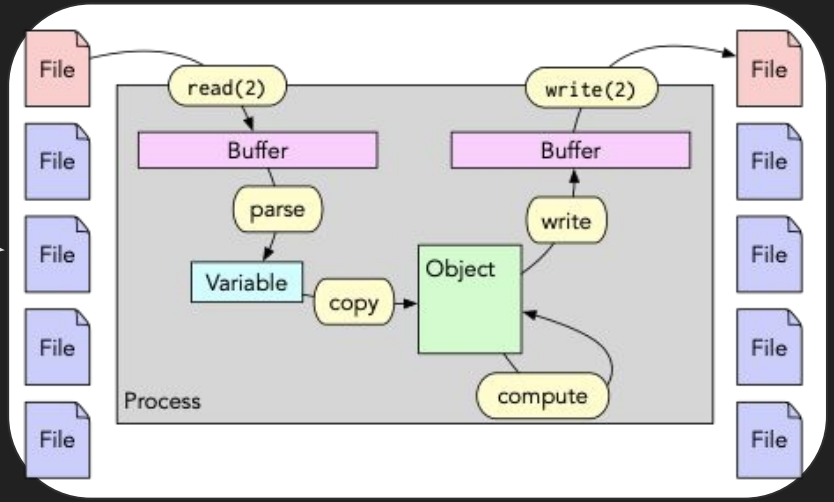
```
strategy: callout
dtrace: userspace
functions:
- callee: [entry]
  metadata:
    name: auth
    id: 1
  name: pam_authenticate
```

And you get this:

```
dtrace: script './pam.d' matched 6 probes
CPU     ID                    FUNCTION:NAME
  2  61725                    none:dt-probe
Pam Authentication - execname: sshd, user: bkidney
  0  61725                    none:dt-probe
Pam Authentication - execname: sshd, user: steve
  2  61725                    none:dt-probe
Pam Authentication - execname: sshd, user: badguy
```

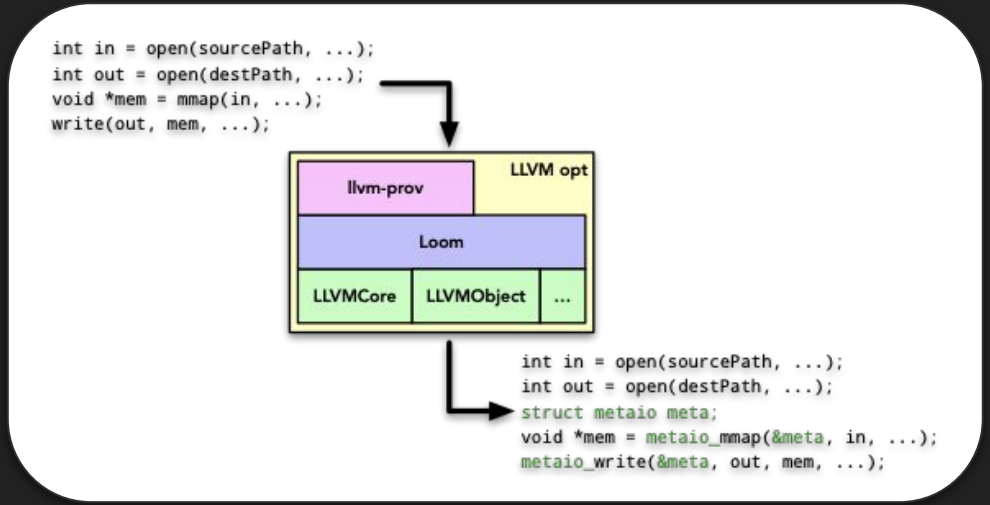
What if you wanted to transform system calls for provenance instrumentation?


File
File
File
File
File
File
File
File
File
File
Socket
ssh(1)
File
mmap(2)
cp(1)
write(2)


File
File
File
File
File
File
File
File
File
File
read(2)
write(2)
Buffer
Buffer
parse
write
Variable
copy
Object
compute
Process

llvm-prov

- Too complex an application for the policy file
- Transforms system call API to new API
- Domain specific logic to determine code of interest
- Loom emits code transformation



Loom

- Simple Policy files to apply instrumentation without code modification
- Framework to build custom tools when you need more

Loom

- Simple Policy files to apply instrumentation without code modification
- Framework to build custom tools when you need more

Currently supports instrumenting:

- Struct fields
- Functions
  - Callee / Caller
- Global Variables
- Pointer Instructions

Supported Outputs:

- Logging
  - KTrace, DTrace, Text, Json, XML
- Code transformation

Loom

- Simple Policy files to apply instrumentation without code modification
- Framework to build custom tools when you need more

- Work has started on instrumentation language
  - DAG matching for code transformation

Currently supports instrumenting:

- Struct fields
- Functions
  - Callee / Caller
- Global Variables
- Pointer Instructions

Supported Outputs:

- Logging
  - KTrace, DTrace, Text, Json, XML
- Code transformation

Come see more
during poster session!

2019 LLVM DEV MEETING

2019 LLVM DEV MEETING

Supporting 64 bit pointers in RISCV 32 bit LLVM backend

Reshabh Sharma

Background:

Prof. Taylor's Bespoke Silicon Group is developing a GP-GPU based on RISC-V 32 bit ISA (RV32)

Why 32 bit ISA?

It is good for very high energy efficiency and density.

What is a good first thing you would expect from a GP-GPU?

Access to 64 bit addressable DRAM

We provide access to 64 bit addresses in DRAM using custom load and store instructions.

LDW rd, rs1, rs2

SDW rd, rs1, rs2

Where rs1 and rs2 always store 32 bit halves of a 64 bit address

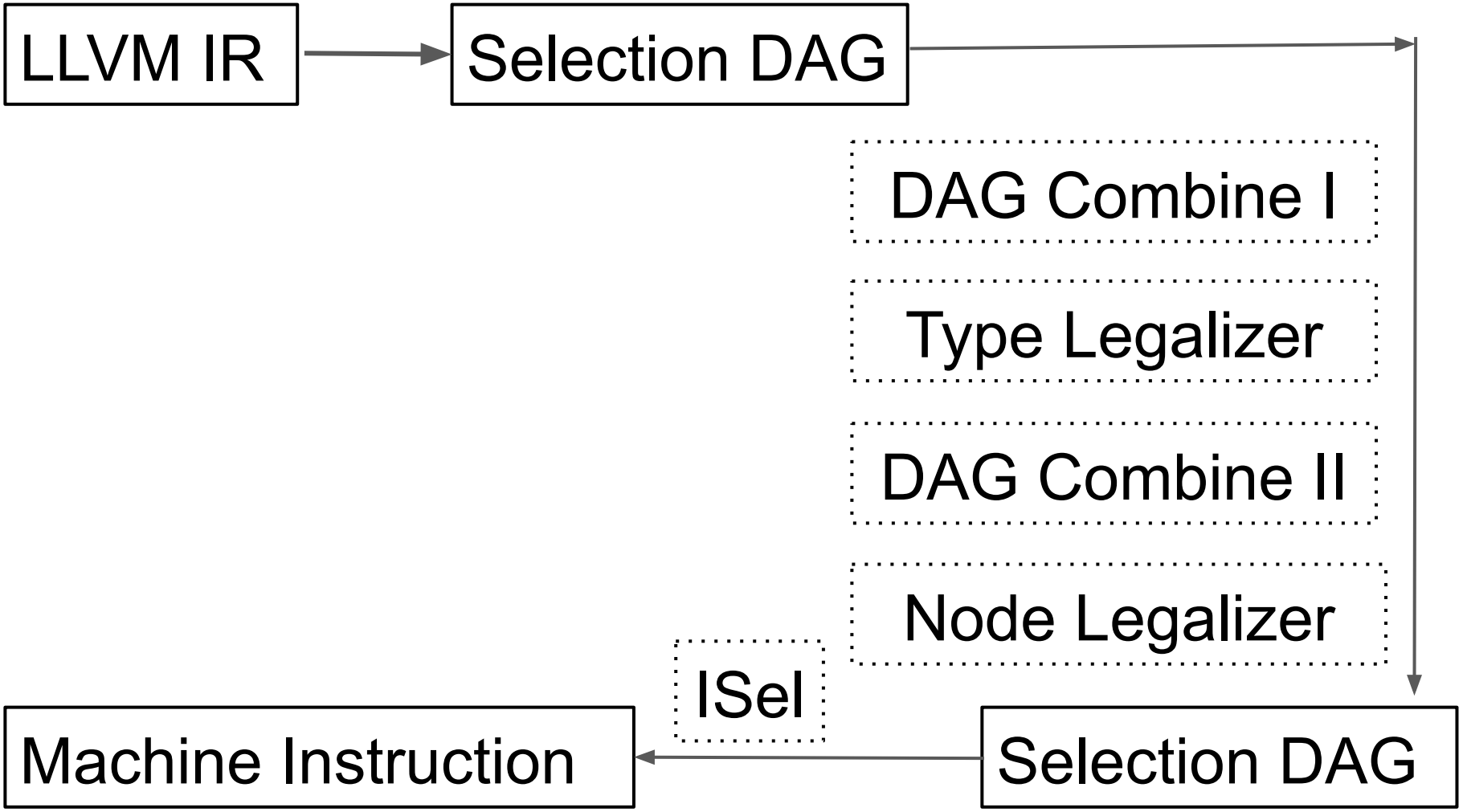
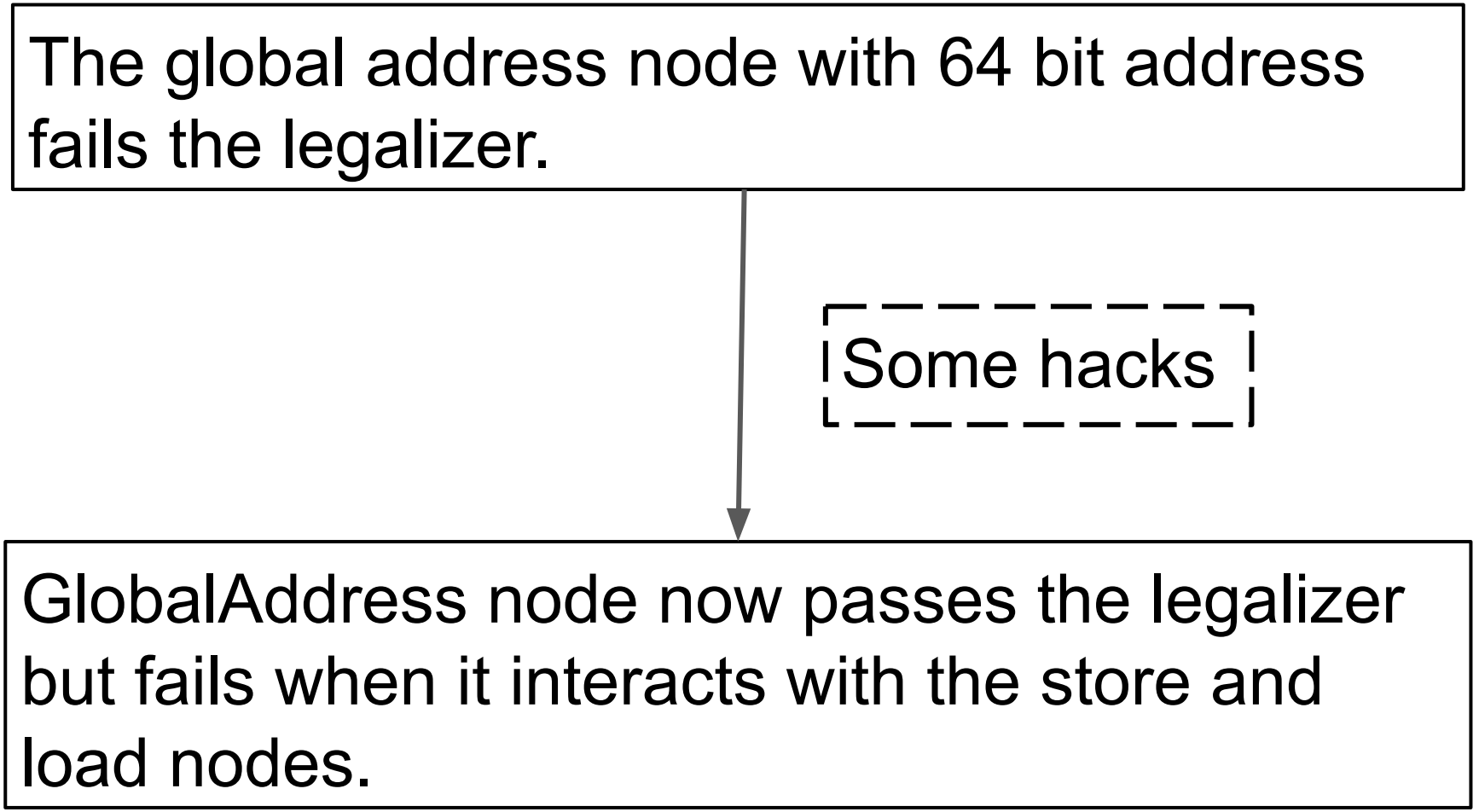
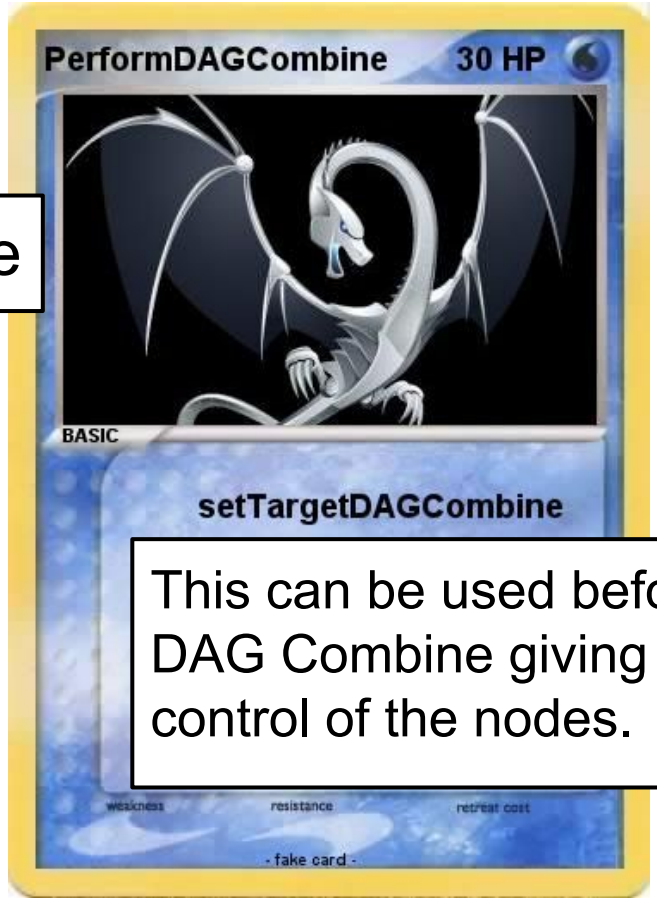

LLVM IR
Selection DAG
DAG Combine I
Type Legalizer
DAG Combine II
Node Legalizer
ISel
Machine Instruction
Selection DAG

Image credits:
Dragon illustration: Vintage vector created by stockgiu - www.freepik.com
Smiley image: The logo belong to the awesome band nirvana


The global address node with 64 bit address fails the legalizer.
Some hacks
GlobalAddress node now passes the legalizer but fails when it interacts with the store and load nodes.

LLVM gives us many good ways to manipulate the nodes before ISel

PerformDAGCombine



This can be used before any DAG Combine giving complete control of the nodes.

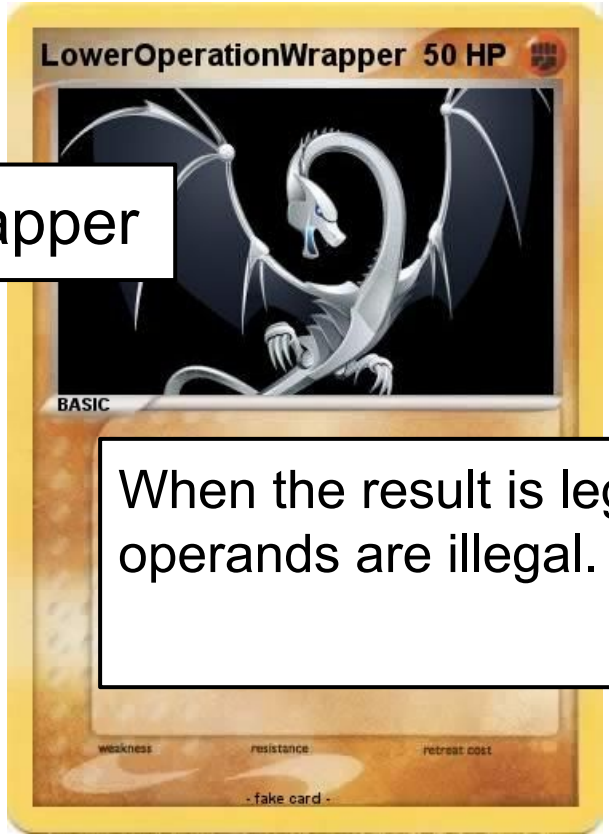
LLVM gives us many good ways to manipulate the nodes before ISel

ReplaceNodeResults

Replaces illegal return type.

LLVM gives us many good ways to manipulate the nodes before ISel

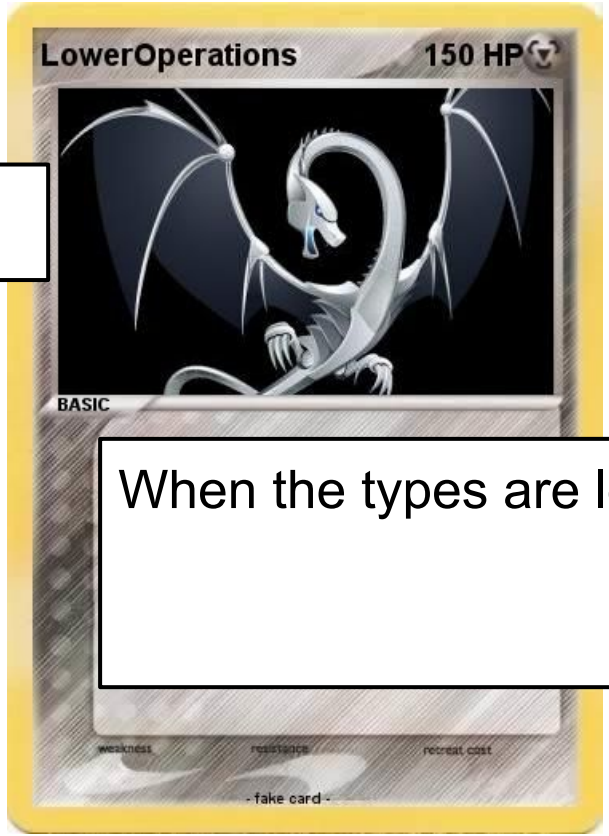
LowerOperationWrapper



When the result is legal and operands are illegal.

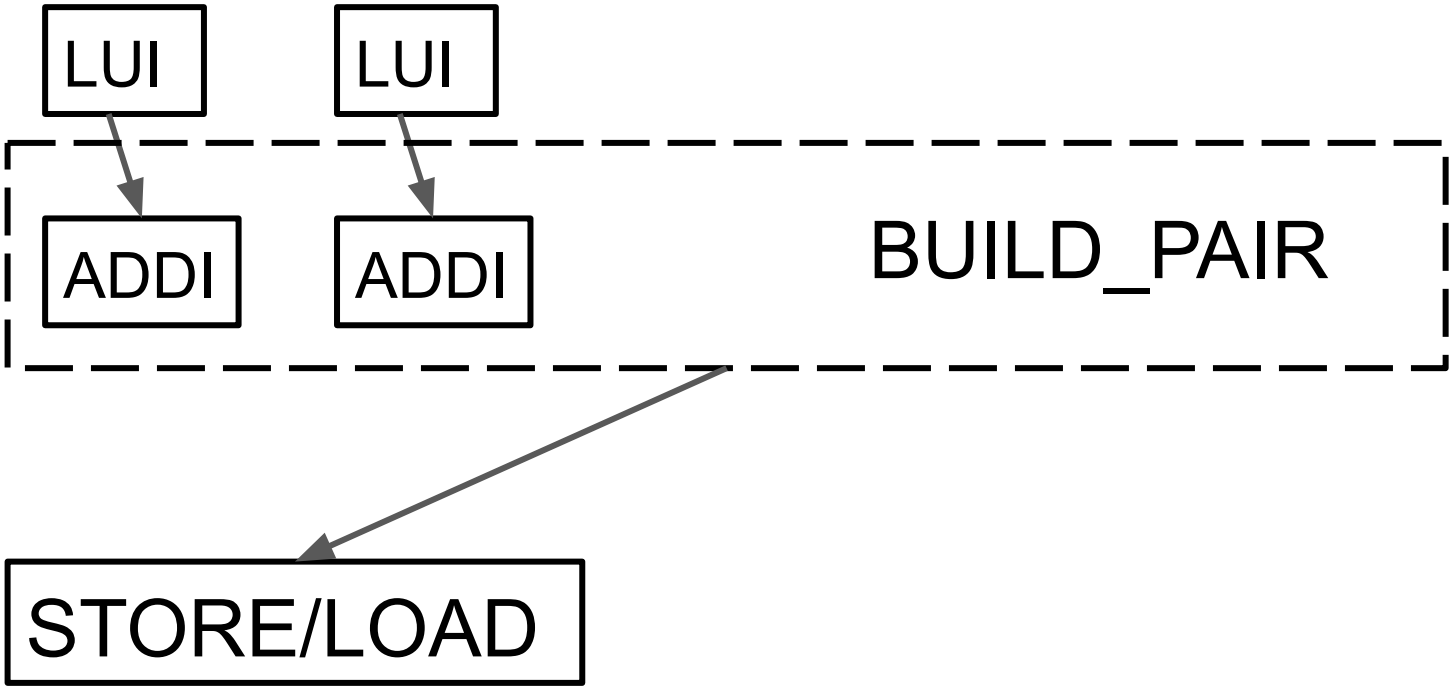
LLVM gives us many good ways to manipulate the nodes before ISel

LowerOperations



How we lowered global address?

GlobalAddress 0x64-bit-address



How we managed to handle store and load with GlobalAddress node?

The only option we had was to lower at the farthest point possible.

Load node is lowered before the Type Legalizer at DAG Combine 1

Store node is lowered at the Node Legalizer

Thank you!

2019 LLVM DEV MEETING

Clang Interface Stubs

Syntax Directed Stub Library Generation

Puyan Lotfi
Facebook

Interface Stubs

```
echo "" | clang -shared -fPIC -x c - -o - | llvm-objdump -section-headers

a.out:  file format ELF64-x86-64

Sections:
Idx Name          Size     Address          Type
  0               00000000 0000000000000000
  1 .dynsym       00000108 00000000000001d0
  2 .dynstr       0000008f 00000000000002d8
  3 .symtab       00000498 0000000000000000
  4 .strtab       00000178 0000000000000000
  5 .shstrtab     000000cc 0000000000000000
  6 .gnu.hash     0000003c 0000000000000190
  7 .gnu.version  00000016 0000000000000368
  8 .gnu.version_r 00000020 0000000000000380
  9 .rela.dyn     000000a8 00000000000003a0
 10 .init         00000017 0000000000000448 TEXT
 11 .plt          00000010 0000000000000460 TEXT
 12 .plt.got      00000008 0000000000000470 TEXT
 13 .text         000000c6 0000000000000480 TEXT
 14 .fini         00000009 0000000000000548 TEXT
 15 .eh_frame_hdr 00000024 0000000000000554 DATA
 16 .eh_frame     0000007c 0000000000000578 DATA
 17 .init_array   00000008 0000000000200e40
 18 .fini_array   00000008 0000000000200e48
 19 .dynamic      00000190 0000000000200e50
 20 .got          00000020 0000000000200fe0 DATA
 21 .got.plt      00000018 0000000000201000 DATA
 22 .data         00000008 0000000000201018 DATA
 23 .bss          00000008 0000000000201020 BSS
 24 .comment      0000008f 0000000000000000
```

Interface Stubs

```
echo "" | clang -shared -fPIC -x c - -o - | llvm-objdump -section-headers

a.out:  file format ELF64-x86-64

Sections:
Idx Name          Size     Address          Type
  0               00000000 0000000000000000
  1 .dynsym       00000108 00000000000001d0
  2 .dynstr       0000008f 00000000000002d8
  3 .symtab       00000498 0000000000000000
  4 .strtab       00000178 0000000000000000
  5 .shstrtab     000000cc 0000000000000000
  6 .gnu.hash     0000003c 0000000000000190
  7 .gnu.version  00000016 0000000000000368
  8 .gnu.version_r 00000020 0000000000000380
  9 .rela.dyn     000000a8 00000000000003a0
 10 .init         00000017 0000000000000448 TEXT
 11 .plt          00000010 0000000000000460 TEXT
 12 .plt.got      00000008 0000000000000470 TEXT
 13 .text         000000c6 0000000000000480 TEXT
 14 .fini         00000009 0000000000000548 TEXT
 15 .eh_frame_hdr 00000024 0000000000000554 DATA
 16 .eh_frame     0000007c 0000000000000578 DATA
 17 .init_array   00000008 0000000000200e40
 18 .fini_array   00000008 0000000000200e48
 19 .dynamic      00000190 0000000000200e50
 20 .got          00000020 0000000000200fe0 DATA
 21 .got.plt      00000018 0000000000201000 DATA
 22 .data         00000008 0000000000201018 DATA
 23 .bss          00000008 0000000000201020 BSS
 24 .comment      0000008f 0000000000000000
```

Motivation

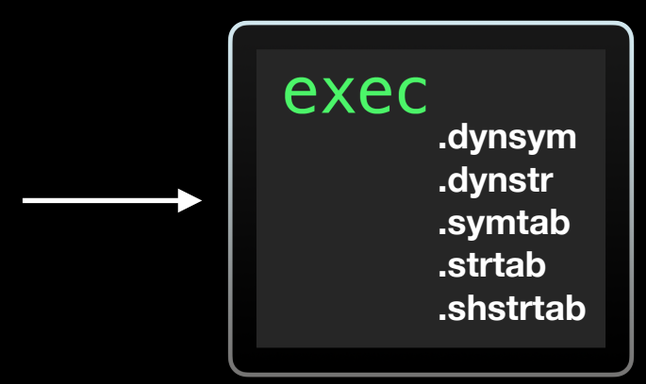
- Generation of lean SDKs
  - No Code
  - Explicit Symbol Exposure
- Code as Source of Truth: Syntax Directed
  - Make use of visibility attributes (ie __attribute__((__visibility__(“hidden”))))

Motivation

- Generation of lean SDKs
  - No Code
  - Explicit Symbol Exposure
- Code as Source of Truth: Syntax Directed
  - Make use of visibility attributes (ie __attribute__((__visibility__(“hidden”))))

clang -emit-interface-stubs -o libfoo.so a.cpp c.cpp sq.cpp

```
#define hidden __attribute__(( \
  __visibility__("hidden")))

hidden int b;
int red() { return b; }
hidden void green() { }
hidden void blue() { }
int a;
hidden int c;
```



Prior Art & Approaches

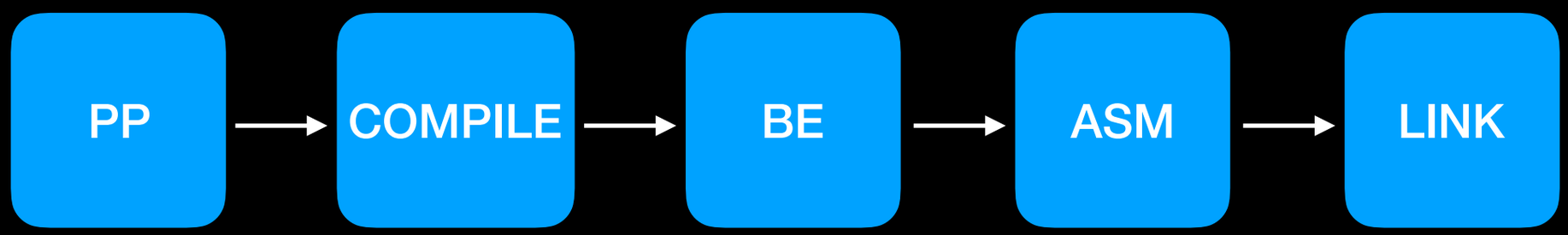
- Microsoft's Import Libraries
  - Generate stub code from compiler & linker
    - Syntax directed through __declspec(dllimport/dllexport)
- Apple's TAPI (Text API)
  - Header Scanning & stub generation
  - .so/.dylib/.dll scanning & stripping

Clang Interface Stubs

- Repurposes visibility attribute to direct symbol exposure using code syntax
  - Fine grain control (internal SPI vs external API)
- Aggregates exposure across compilation units via text
- Supports ELF
- Yields smaller SDK and faster link times

Clang Driver Pipeline

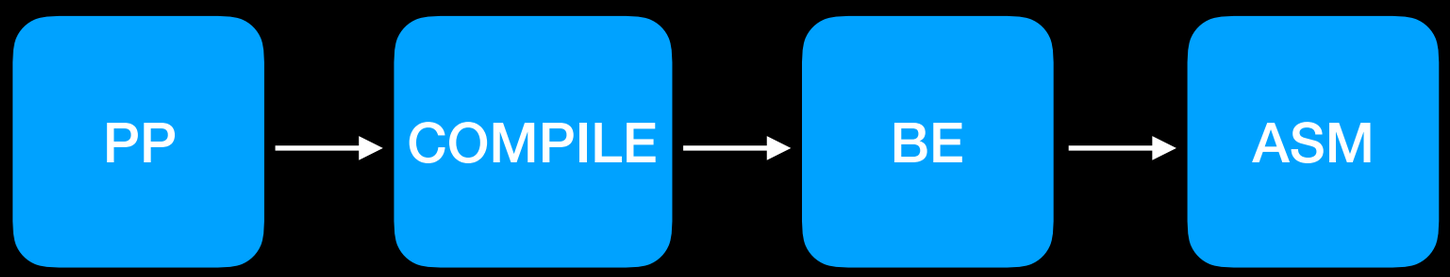
- Traditional Pipeline: Preprocess, Compile, Backend, Assemble, and Link Phases



Clang Driver Pipeline

- Traditional Pipeline: Preprocess, Compile, Backend, Assemble, and Link Phases

```
clang -c
```



Clang Driver Pipeline

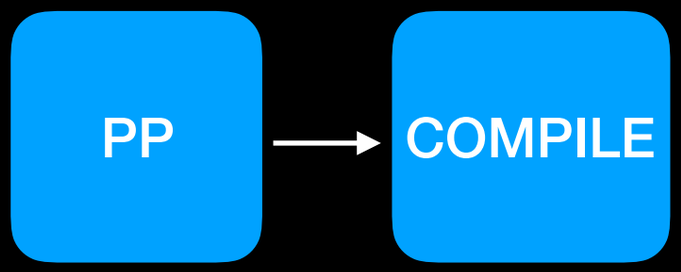
- Traditional Pipeline: Preprocess, Compile, Backend, Assemble, and Link Phases

```
clang -S
```

Clang Driver Pipeline

- Traditional Pipeline: Preprocess, Compile, Backend, Assemble, and Link Phases

```
clang -fsyntax_only
```



Clang Driver Pipeline

- Traditional Pipeline: Preprocess, Compile, Backend, Assemble, and Link Phases

```
clang -E
```

PP

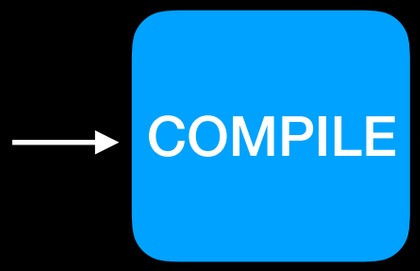
Driver Pipeline

- Clang Interface Stubs Pipeline: Preprocess, Compile, and Merge phases
- Compile Phase: Generates intermediate text (.ifs files)
- Merge Phase: Invokes llvm-ifs to consume & merge .ifs files to produce ELF .so

Driver Pipeline

- Clang Interface Stubs Pipeline: Preprocess, Compile, and Merge phases
- Compile Phase: Generates intermediate text (.ifs files)
- Merge Phase: Invokes llvm-ifs to consume & merge .ifs files to produce ELF .so
- Compile Phase invokes InterfaceStubFunctionsConsumer (clang -cc1)
  - Walks the AST scanning for visible decls

```
#define weak \
  __attribute__((__weak__))
#define hidden __attribute__(( \
  __visibility__("hidden")))

hidden int b;
int red() { return b; }
weak void green() { }
hidden void blue() { }
int a;
hidden int c;
```

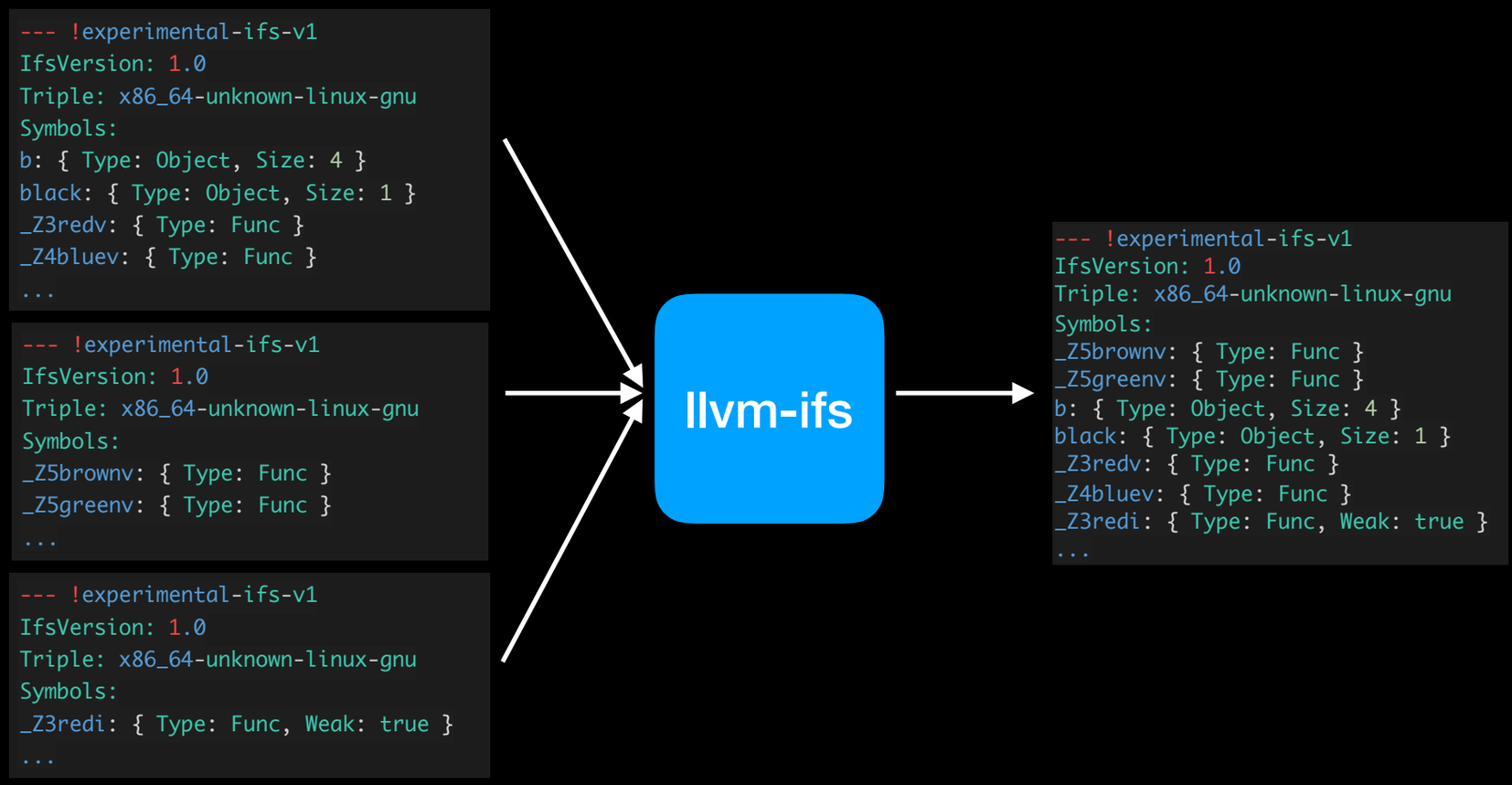



```
--- !experimental-ifs-v1
IfsVersion: 1.0
Triple: x86_64-unknown-linux-gnu
Symbols:
_Z3redv: { Type: Func }
_Z5greenv: { Type: Func,
             Weak: true }
a: { Type: Object, Size: 4 }
...
```

IFS Text Format

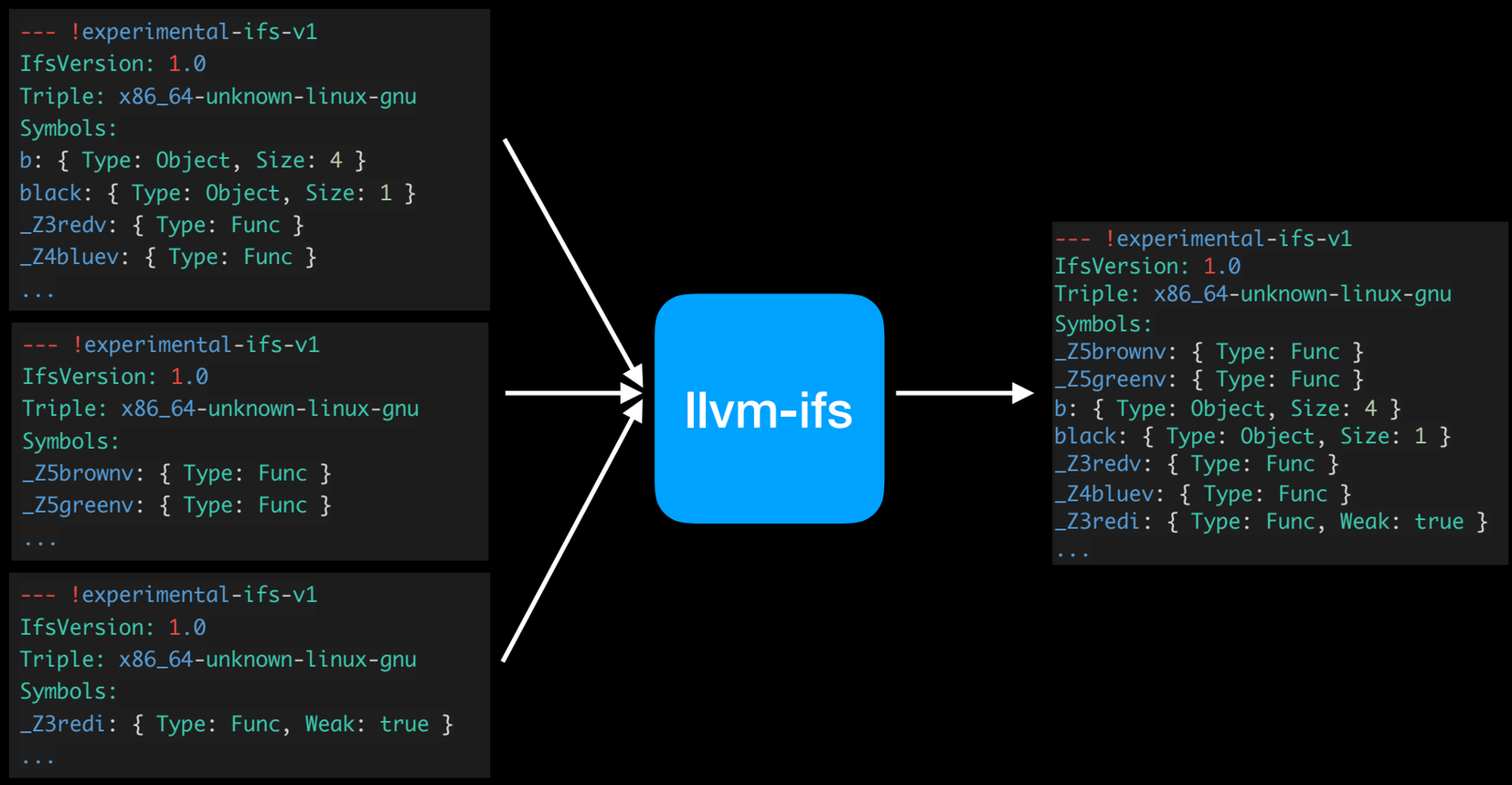
```
--- !experimental-ifs-v1
IfsVersion: 1.0
Triple: x86_64-unknown-linux-gnu
Symbols:
_Z5brownv: { Type: Func }
_Z5greenv: { Type: Func }
b: { Type: Object, Size: 4 }
black: { Type: Object, Size: 1 }
_Z3redv: { Type: Func }
_Z4bluev: { Type: Func }
_Z3redi: { Type: Func, Weak: true }
...
```

IFS Text Format



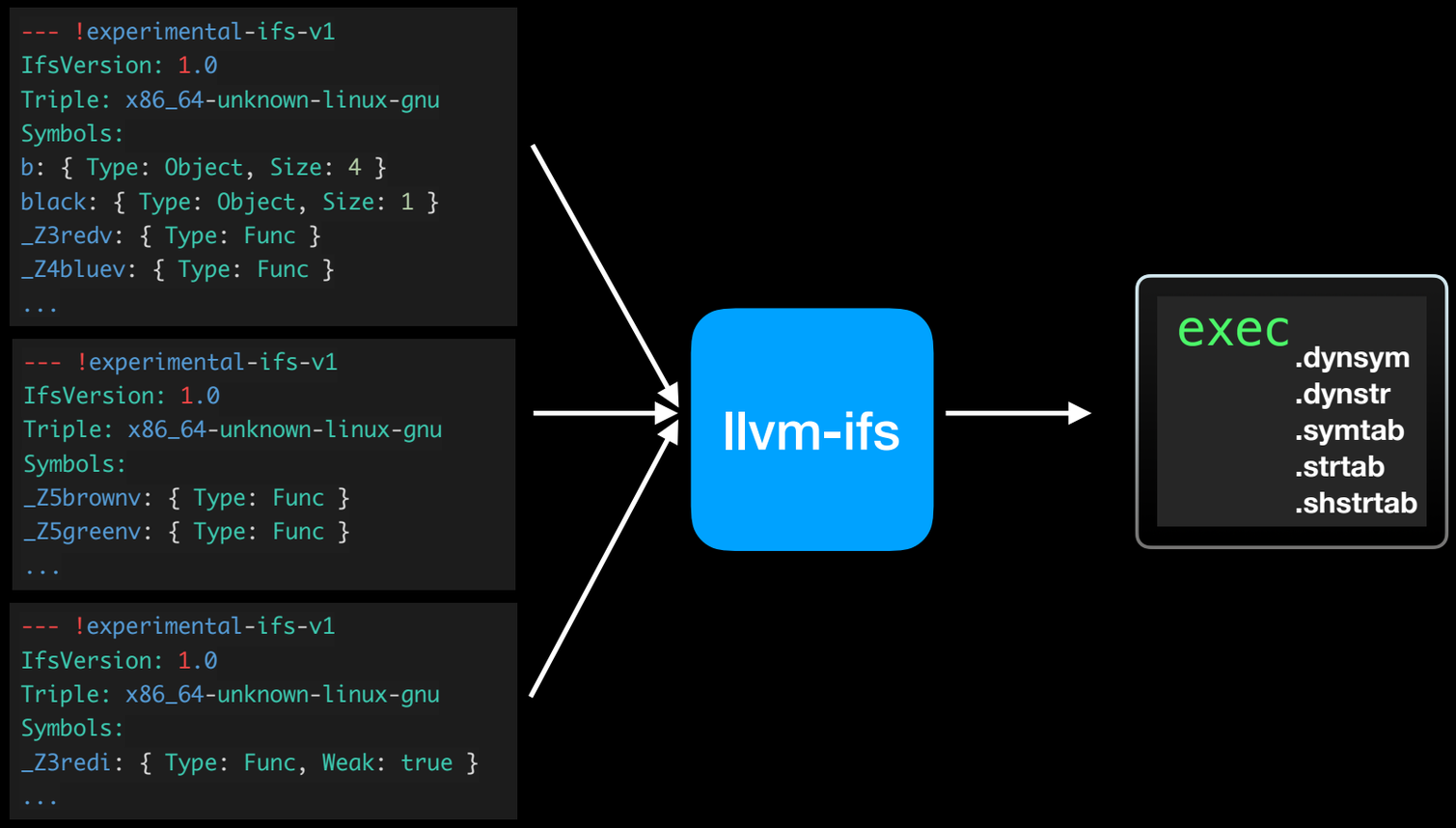
llvm-ifs

- A new llvm tool for consuming IFS files: produces a merged IFS file, or ELF shared object (.dynsym, .dynstr, .symtab only)



llvm-ifs

- A new llvm tool for consuming IFS files: produces a merged IFS file, or ELF shared object (.dynsym, .dynstr, .symtab only)



Example

clang -emit-interface-stubs -o libfoo.so a.cpp c.cpp sq.cpp

```
void brown() {  }
int green() { return 42; }
```

→ COMPILE →

```
--- !experimental-ifs-v1
IfsVersion: 1.0
Triple: x86_64-unknown-linux-gnu
Symbols:
_Z5brownv: { Type: Func }
_Z5greenv: { Type: Func }
...
```

→ MERGE (llvm-ifs)

Example

clang -emit-interface-stubs -o libfoo.so a.cpp c.cpp sq.cpp

```
#define hidden __attribute__(( \
  __visibility__("hidden")))

hidden int b;
hidden char black;
int orange() { return b; }
void blue() { }
hidden int c;
```

COMPILE

```
--- !experimental-ifs-v1
IfsVersion: 1.0
Triple: x86_64-unknown-linux-gnu
Symbols:
_Z3orangev: { Type: Func }
_Z4bluev: { Type: Func }
...
```

```
void brown() {  }
int green() { return 42; }
```

COMPILE

```
--- !experimental-ifs-v1
IfsVersion: 1.0
Triple: x86_64-unknown-linux-gnu
Symbols:
_Z5brownv: { Type: Func }
_Z5greenv: { Type: Func }
...
```

```
__attribute__((__weak__))
int red(int num) {
  return num * num;
}
```

COMPILE

```
--- !experimental-ifs-v1
IfsVersion: 1.0
Triple: x86_64-unknown-linux-gnu
Symbols:
_Z3redi: { Type: Func,
           Weak: true }
...
```

MERGE (llvm-ifs)

Example

clang -emit-interface-stubs -o libfoo.so a.cpp c.cpp sq.cpp

```
#define hidden __attribute__(( \
  __visibility__("hidden")))

hidden int b;
hidden char black;
int orange() { return b; }
void blue() { }
hidden int c;
```

COMPILE

```
--- !experimental-ifs-v1
IfsVersion: 1.0
Triple: x86_64-unknown-linux-gnu
Symbols:
_Z3orangev: { Type: Func }
_Z4bluev: { Type: Func }
...
```

```
void brown() {  }
int green() { return 42; }
```

COMPILE

```
--- !experimental-ifs-v1
IfsVersion: 1.0
Triple: x86_64-unknown-linux-gnu
Symbols:
_Z5brownv: { Type: Func }
_Z5greenv: { Type: Func }
...
```

```
__attribute__((__weak__))
int red(int num) {
  return num * num;
}
```

COMPILE

```
--- !experimental-ifs-v1
IfsVersion: 1.0
Triple: x86_64-unknown-linux-gnu
Symbols:
_Z3redi: { Type: Func,
           Weak: true }
...
```

MERGE (llvm-ifs)

```
--- !experimental-ifs-v1
IfsVersion: 1.0
Triple: x86_64-unknown-linux-gnu
Symbols:
_Z5brownv: { Type: Func }
_Z5greenv: { Type: Func }
b: { Type: Object, Size: 4 }
black: { Type: Object, Size: 1 }
_Z3orangev: { Type: Func }
_Z4bluev: { Type: Func }
_Z3redi: { Type: Func, Weak: true }
...
```

Example

clang -emit-interface-stubs -o libfoo.so a.cpp c.cpp sq.cpp

```
#define hidden __attribute__(( \
  __visibility__("hidden")))

hidden int b;
hidden char black;
int orange() { return b; }
void blue() { }
hidden int c;
```

COMPILE

```
--- !experimental-ifs-v1
IfsVersion: 1.0
Triple: x86_64-unknown-linux-gnu
Symbols:
_Z3orangev: { Type: Func }
_Z4bluev: { Type: Func }
...
```

```
void brown() {  }
int green() { return 42; }
```

COMPILE

```
--- !experimental-ifs-v1
IfsVersion: 1.0
Triple: x86_64-unknown-linux-gnu
Symbols:
_Z5brownv: { Type: Func }
_Z5greenv: { Type: Func }
...
```

```
__attribute__((__weak__))
int red(int num) {
  return num * num;
}
```

COMPILE

```
--- !experimental-ifs-v1
IfsVersion: 1.0
Triple: x86_64-unknown-linux-gnu
Symbols:
_Z3redi: { Type: Func,
           Weak: true }
...
```

MERGE (llvm-ifs)

exec

.dynsym
.dynstr
.symtab
.strtab
.shstrtab

Challenges

- Refactoring for alternate pipeline setup required changes in Driver::BuildActions and getCompilationPhases
- Handling corner cases in the driver setup
- Handling Templates, Specializations, and non-trivial decls
- Converging on the text format required many iterations

Future Work

- Hardening IFS ASTConsumer and llvm-ifs
- Generating interface stubs for libc++ builds
- Inline Assembly
- Support for new formats: MS Import Libs, Darwin TAPI

2019

LLVM DEV MEETING

FLANG UPDATE

Steve Scalpone, LLVM Developers' Meeting, October 23, 2019

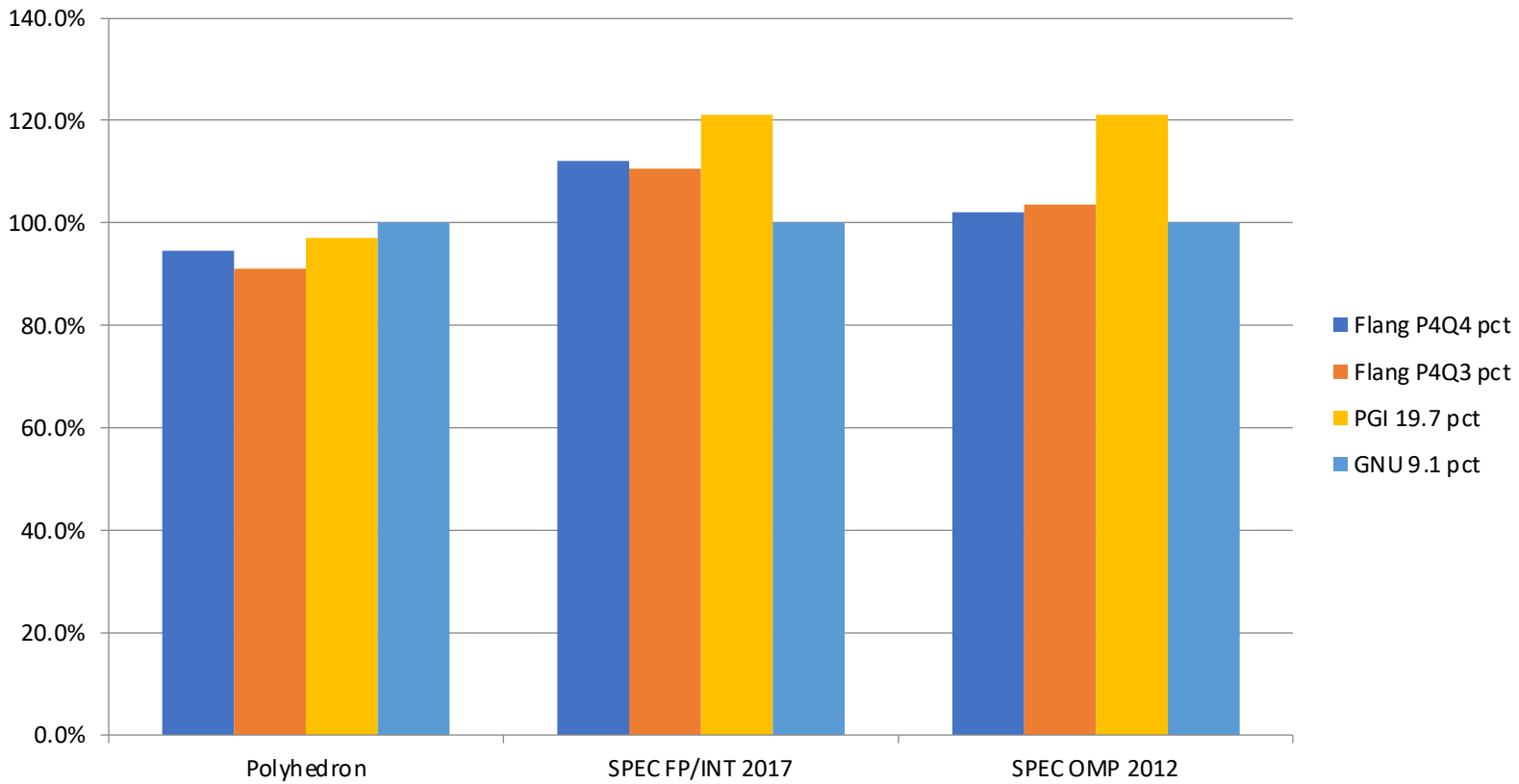
THE FLANG PROJECT

Now affectionately known as ”Old Flang”

- A multi-year collaboration between the NNSA labs and NVIDIA/PGI
  - Open-source Fortran compiler since May 2017
  - Arm and AMD are shipping compilers built using Flang
  - Wide variety of contributors; most recently Microsoft signed the CCLA
- The end of the CLA
  - No longer require a CCLA or CLA to contribute
  - Adding Arm and other community members as committers
  - Relicensing from Apache 2.0 to Apache 2.0 with LLVM Exception

FLANG PERFORMANCE

All runs on dual-socket Intel Xeon Skylake



Performance measured September 2019 and are considered ESTIMATES per SPEC run and reporting rules.
Two 20 core Skylake Intel® Gold 6148 CPU @ 2.40GHz CPUs @ 2.4GHz w/ 256GB memory. SPEC® is a registered trademark of the Standard Performance Evaluation Corporation (www.spec.org).

THE FLANG PROJECT

Some call it “f18” or “new flang”

New Flang compiler was started from scratch

Modern & well-documented design written in C++17

NNSA, ECP, NVIDIA, Arm and others have contributed

It’s an LLVM subproject despite not yet having files in the repo

Developed 100% in open source guided by LLVM coding standards

THE FLANG PROJECT

Using some interesting technologies

- FIR implementation is as an MLIR dialect
  - MLIR replaced the original f18/FIR infrastructure
  - Working with the MLIR team to extend MLIR as needed for FIR
- OpenMP IRBuilder drives language-independent lowering
  - Defining an OpenMP dialect in MLIR
  - Optz and transformations may be in OpenMP IRBuilder, MLIR framework, or LLVM

THE FLANG PROJECT

SOURCE CODE

https://github.com/flang-compiler

MAILING LIST

flang-dev@lists.llvm.org

PROJECT CALL

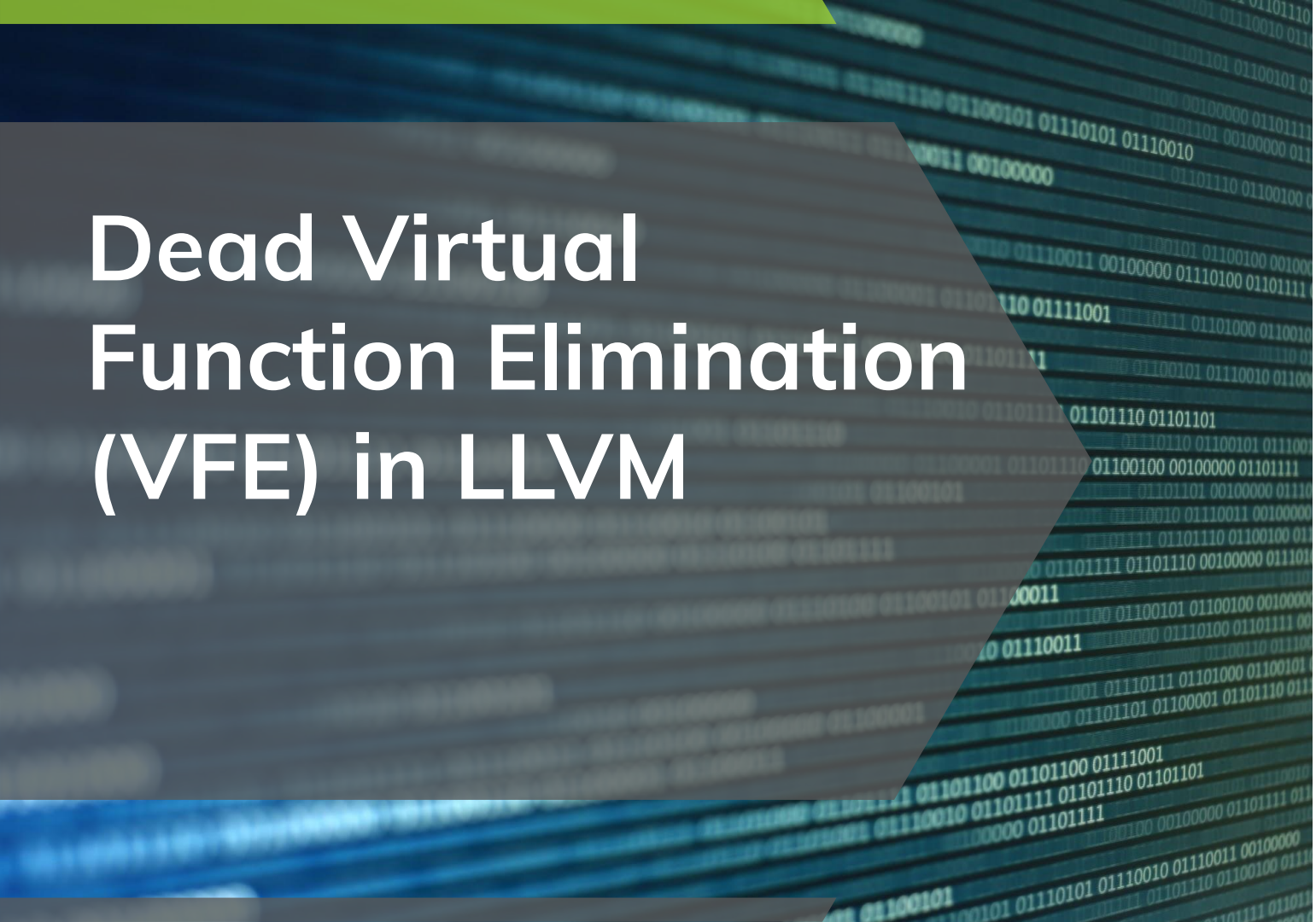
Every other Wednesday 8:30am Pacific
See the mailing list for details

TECHNICAL CALL

Every other Monday 8:30 Pacific
See the mailing list for details

2019 LLVM DEV MEETING

Dead Virtual Function Elimination (VFE) in LLVM



Linaro

An example we'd like to optimise

```
struct A {
  virtual int foo() { return 1; }
  virtual int bar() { return 2; }
};

struct B : A {
  virtual int foo() { return 3; }
  virtual int bar() { return 4; }
};

A* make_A() { return new A(); }
B* make_B() { return new B(); }

int call_1(A* p) { return p->foo(); }
int call_2(B* p) { return p->bar(); }
```

An example we'd like to optimise

```
struct A {
  virtual int foo() { return 1; }
  virtual int bar() { return 2; }
};

struct B : A {
  virtual int foo() { return 3; }
  virtual int bar() { return 4; }
};

A* make_A() { return new A(); }
B* make_B() { return new B(); }

int call_1(A* p) { return p->foo(); }
int call_2(B* p) { return p->bar(); }
```

Could call A::foo or B::foo

Can only call B::bar

An example we'd like to optimise

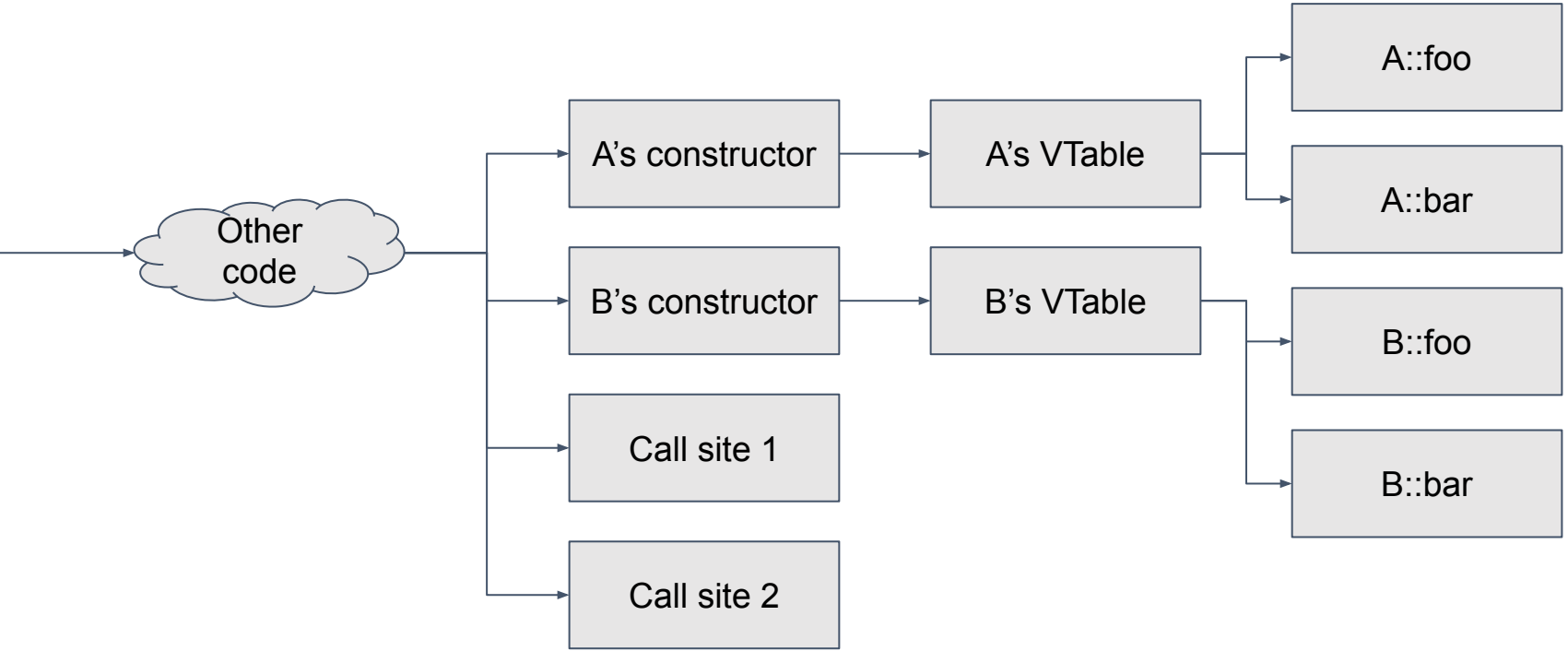
```
struct A {
  virtual int foo() { return 1; }
  virtual int bar() { return 2; }
};

struct B : A {
  virtual int foo() { return 3; }
  virtual int bar() { return 4; }
};

A* make_A() { return new A(); }
B* make_B() { return new B(); }

int call_1(A* p) { return p->foo(); }
int call_2(B* p) { return p->bar(); }
```

No calls to this function

Could call A::foo or B::foo

Can only call B::bar

What does this looks like to GlobalDCE?



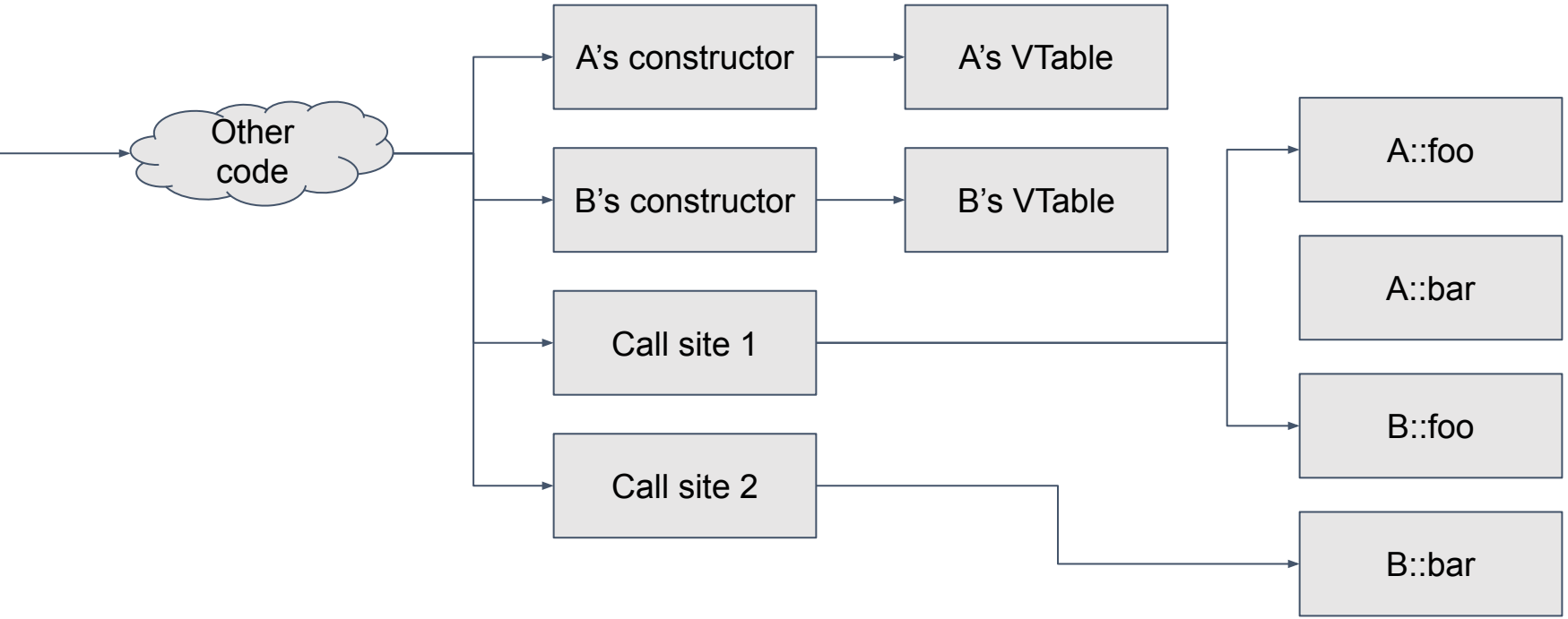
Type metadata

- Optionally added to vtables, virtual call sites
- Provides link between call sites and vtable slots
- Currently used for devirtualisation, control flow integrity
- Needs some changes to call sites to allow VFE

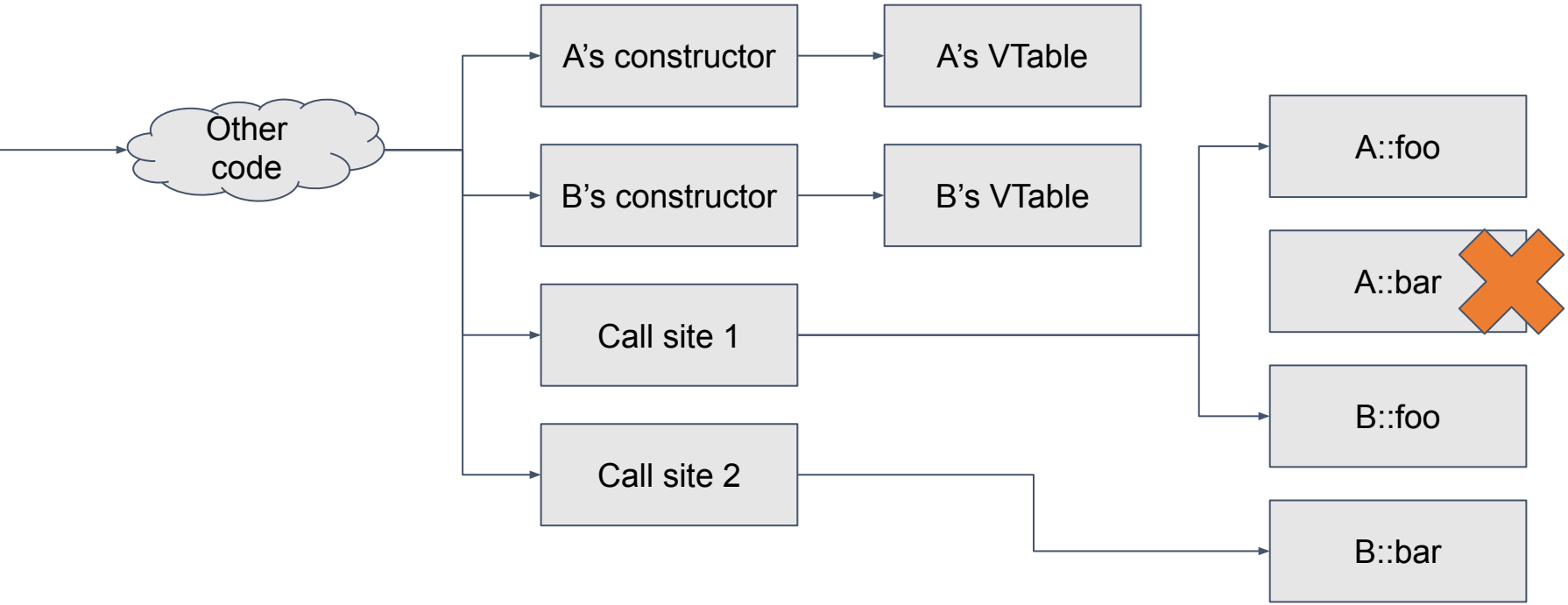
```
@_ZTV1A = constant { [4 x i8*] } ...,  !type !0
@_ZTV1B = constant { [4 x i8*] } ...,  !type !0, !type !4

!0 = !{i64 16, !"_ZTS1A"}
!4 = !{i64 16, !"_ZTS1B"}

%vtable1 = load i8*, i8** %0, align 8, !tbaa !9
%1 = tail call { i8*, i1 } @llvm.type.checked.load(
                                i8* %vtable1, i32  8, metadata !"_ZTS1B")
```

Linaro

How does this change GlobalDCE's view?



How does this change GlobalDCE's view?

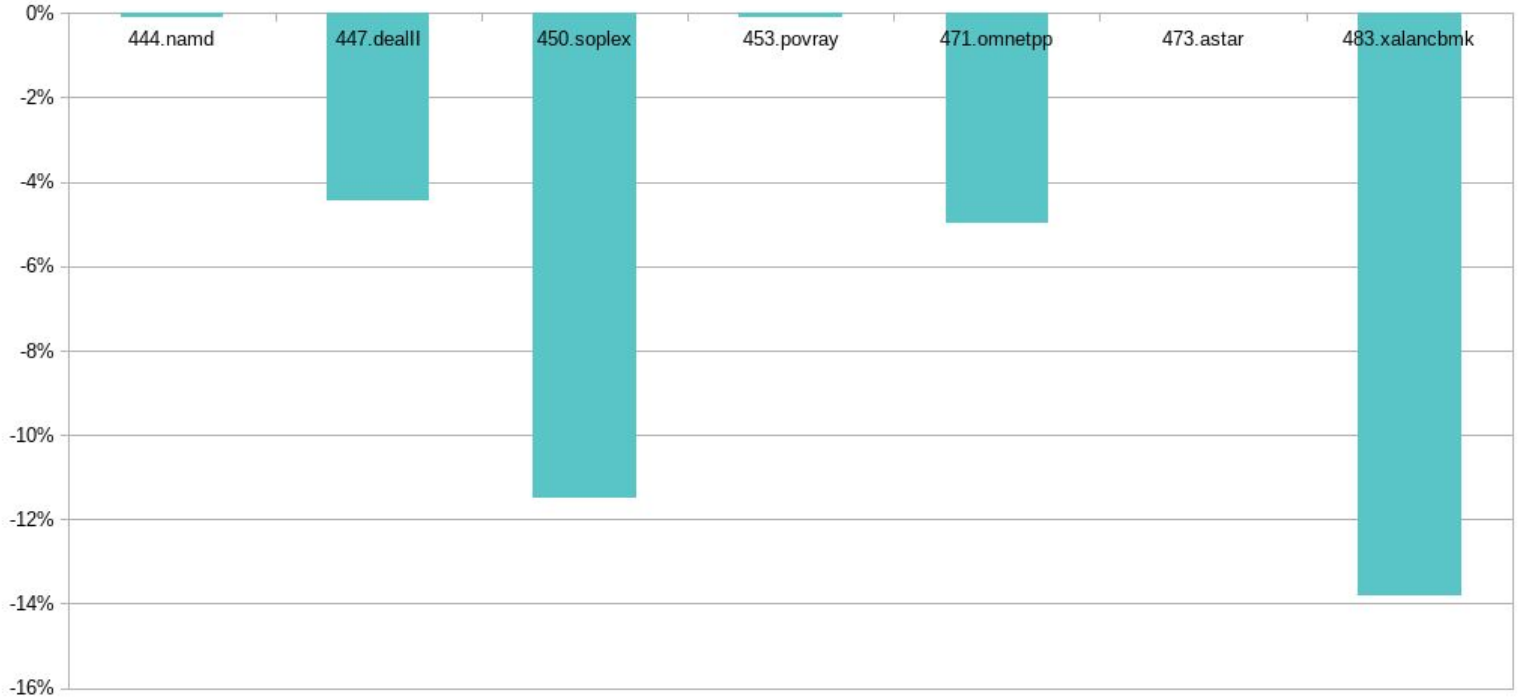


When is this optimisation valid?

- Need to see every possible call site involving a class
- Generally requires `-fvisibility=hidden`, and LTO
- All base classes (with some exceptions) must also be hidden
  - Otherwise, more virtual calls could be outside the LTO unit
- Added new `!vcall_visibility` metadata to represent this

Benchmark - SPEC2006 (C++ subset)

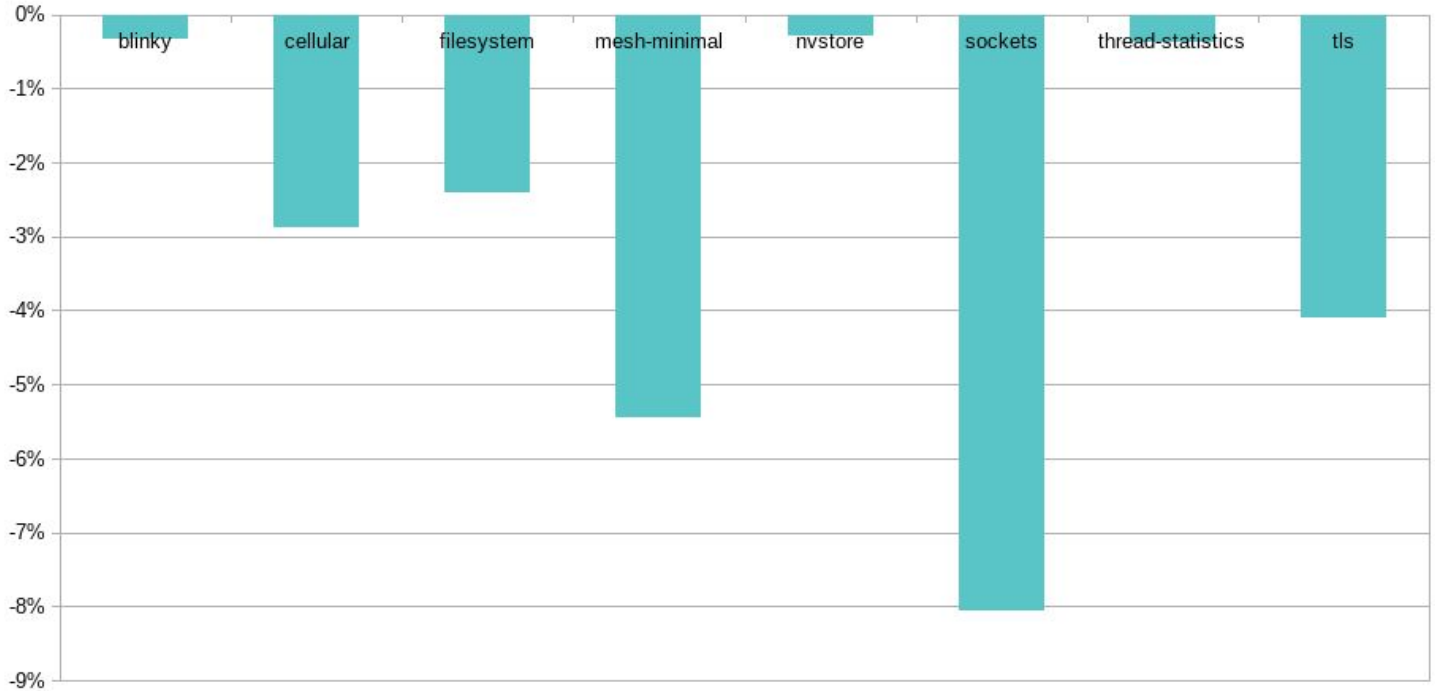
Code size change, compared to -Oz -flto -fvisibility=hidden



Linaro

Benchmark - mbed-os examples

Code size change, compared to -Oz -flto -fvisibility=hidden



Linaro

Future work

- ThinLTO
- Dead RTTI elimination

2019 LLVM DEV MEETING

Making a Language Cross Platform

Libraries and Tooling

Gwen Mittertreiner

Facebook

Goals

- Make you think about cross platform support
- High level, but still have practical advice.

Paths and File Systems

Paths and File Systems

- On Unix, paths are easy to parse right? /usr/bin/easy/peasy

Paths and File Systems

- On Unix, paths are easy to parse right? /usr/bin/easy/peasy
- On Windows, paths are easy

Paths and File Systems

- On Unix, paths are easy to parse right? /usr/bin/easy/peasy
- On Windows, paths are easy to get wrong.

Paths and File Systems

- On Unix, paths are easy to parse right? /usr/bin/easy/peasy
- On Windows, paths are easy to get wrong.
- Which of these are valid paths on Windows?
  - A:\foo\bar
  - B:foo\bar
  - C:/foo/bar
  - D:/foo\bar
  - \\?\E:\foo\bar
  - \\.\F:\foo\bar
  - \??\G:\foo\bar
  - \usr\bin\clang
  - \\?\UNC\abc\xyz
  - \\?\GLOBALROOT\??\UNC\abc\xyz

Paths and File Systems

- On Unix, paths are easy to parse right? /usr/bin/easy/peasy
- On Windows, paths are easy to get wrong.
- Which of these are valid paths on Windows?
  - A:\foo\bar
  - B:foo\bar
  - C:/foo/bar
  - D:/foo\bar
  - \\?\E:\foo\bar
  - \\.\F:\foo\bar
  - \??\G:\foo\bar
  - \usr\bin\clang
  - \\?\UNC\abc\xyz
  - \\?\GLOBALROOT\??\UNC\abc\xyz
- All of them!

Paths and File Systems

- On Unix, paths are easy to parse right? /usr/bin/easy/peasy
- On Windows, paths are easy to get wrong.
- Which of these are valid paths on Windows?
  - A:\foo\bar
  - B:foo\bar
  - C:/foo/bar
  - D:/foo\bar
  - \\?\E:\foo\bar
  - \\.\F:\foo\bar
  - \??\G:\foo\bar
  - \usr\bin\clang
  - \\?\UNC\abc\xyz
  - \\?\GLOBALROOT\??\UNC\abc\xyz
- All of them! * (mostly)

*https://googleprojectzero.blogspot.com/2016/02/the-definitive-guide-on-win32-to-nt.html

Paths and File Systems

Paths and File Systems

- Let system APIs do the lifting

Paths and File Systems

- Let system APIs do the lifting
- Avoid matching paths by streq

Building and Testing

Building and Testing

- Keep cross compiling in mind through out

Building and Testing

- Keep cross compiling in mind through out
- Be very careful with Unix commands

Building and Testing

- Keep cross compiling in mind through out
- Be very careful with Unix commands
- Linker formats

Building and Testing

- Keep cross compiling in mind through out
- Be very careful with Unix commands
- Linker formats
  - Different formats support different things
    - Weak Symbols
    - Symbol Replacement
    - Thread Local Storage
    - DLL Storage
    - Two level namespaces

Porting Existing Libraries

Porting Existing Libraries

- UTF-8 is pretty standard now...

Porting Existing Libraries

- UTF-8 is pretty standard now...
  - Except Windows and Apple's Cocoa any sufficiently old platform! Those use UTF-16.

Porting Existing Libraries

- UTF-8 is pretty standard now...
  - Except Windows and Apple's Cocoa any sufficiently old platform! Those use UTF-16.
- Two Approaches:

Porting Existing Libraries

- UTF-8 is pretty standard now...
  - Except Windows and Apple's Cocoa any sufficiently old platform! Those use UTF-16.
- Two Approaches:
  - Define a character type that's different sizes on different platforms

Porting Existing Libraries

- UTF-8 is pretty standard now...
  - Except Windows and Apple's Cocoa any sufficiently old platform! Those use UTF-16.
- Two Approaches:
  - Define a character type that's different sizes on different platforms
  - Convert everything to a single encoding

Porting Existing Libraries

- UTF-8 is pretty standard now...
  - Except Windows and Apple's Cocoa any sufficiently old platform! Those use UTF-16.
- Two Approaches:
  - Define a character type that's different sizes on different platforms
  - Convert everything to a single encoding
- Be okay with things not making sense

Porting Existing Libraries

- UTF-8 is pretty standard now...
  - Except Windows and Apple's Cocoa any sufficiently old platform! Those use UTF-16.
- Two Approaches:
  - Define a character type that's different sizes on different platforms
  - Convert everything to a single encoding
- Be okay with things not making sense :(

Porting Existing Libraries

- UTF-8 is pretty standard now...
  - Except Windows and Apple's Cocoa any sufficiently old platform! Those use UTF-16.
- Two Approaches:
  - Define a character type that's different sizes on different platforms
  - Convert everything to a single encoding
- Be okay with things not making sense :(
- POSIX apis are a handy tool

Porting Existing Libraries

- UTF-8 is pretty standard now...
  - Except Windows and Apple's Cocoa any sufficiently old platform! Those use UTF-16.
- Two Approaches:
  - Define a character type that's different sizes on different platforms
  - Convert everything to a single encoding
- Be okay with things not making sense :(
- POSIX apis are a handy tool
  - Windows supports many POSIX APIs

Porting Existing Libraries

- UTF-8 is pretty standard now...
  - Except Windows and Apple's Cocoa any sufficiently old platform! Those use UTF-16.
- Two Approaches:
  - Define a character type that's different sizes on different platforms
  - Convert everything to a single encoding
- Be okay with things not making sense :(
- POSIX apis are a handy tool
  - Windows supports many POSIX APIs
    - With some limitations

Porting Existing Libraries

- UTF-8 is pretty standard now...
  - Except Windows and Apple's Cocoa any sufficiently old platform! Those use UTF-16.
- Two Approaches:
  - Define a character type that's different sizes on different platforms
  - Convert everything to a single encoding
- Be okay with things not making sense :(
- POSIX apis are a handy tool
  - Windows supports many POSIX APIs
    - With some limitations
  - Android doesn't support some POSIX APIs

Porting Existing Libraries

- UTF-8 is pretty standard now...
  - Except Windows and Apple's Cocoa any sufficiently old platform! Those use UTF-16.
- Two Approaches:
  - Define a character type that's different sizes on different platforms
  - Convert everything to a single encoding
- Be okay with things not making sense :(
- POSIX apis are a handy tool
  - Windows supports many POSIX APIs
    - With some limitations
  - Android doesn't support some POSIX APIs
    - Depends on API level

Porting Existing Libraries

- UTF-8 is pretty standard now...
  - Except Windows and Apple's Cocoa any sufficiently old platform! Those use UTF-16.
- Two Approaches:
  - Define a character type that's different sizes on different platforms
  - Convert everything to a single encoding
- Be okay with things not making sense :(
- POSIX apis are a handy tool
  - Windows supports many POSIX APIs
    - With some limitations
  - Android doesn't support some POSIX APIs
    - Depends on API level
  - Reimplementing the original target's API set

Thanks!

- I hope you'll all go and make someone porting your project happy.

2019 LLVM DEV MEETING

PURDUE UNIVERSITY®

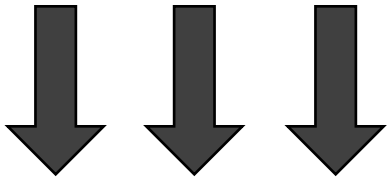
Grafter: A Clang tool for fusing tree traversals

Laith Sakka Kirshanthan Sundararajah Milind Kulkarni

What is Grafter?

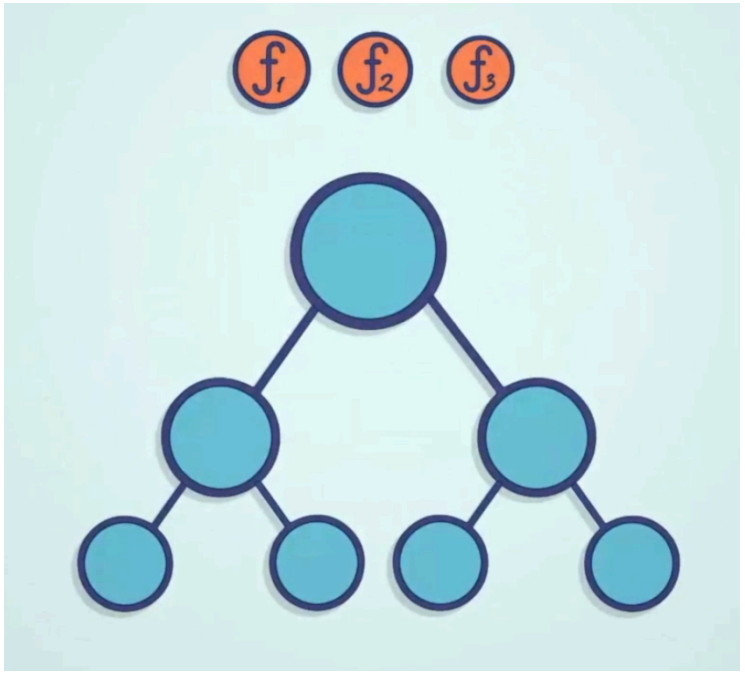
Grafter performs source to source transformations to fuse recursive functions that traverse trees

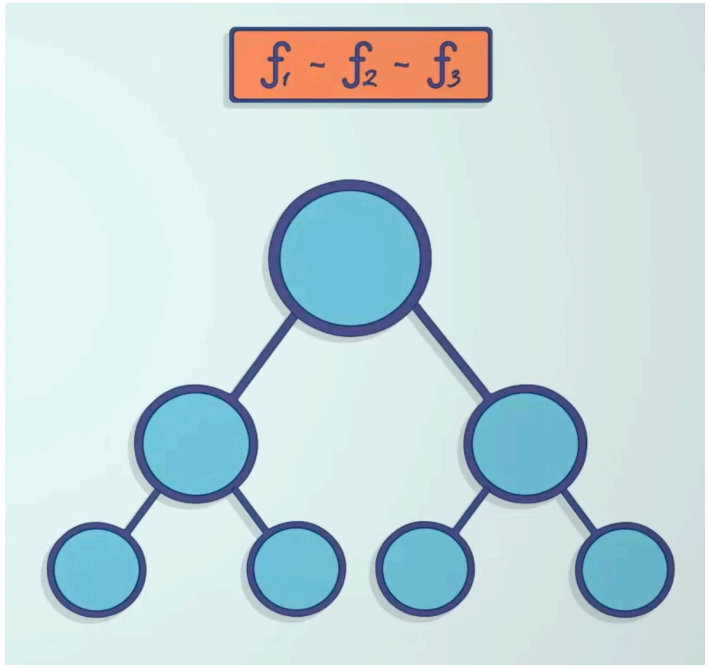
```
computeHight (render_tree);
computeWidth(render_tree);
computePos(render_tree);
```



```
computeHight_computeWidth_computePos(render_tree);
```

What is Grafter?





Sound, Fine-Grained Traversal Fusion for Heterogeneous Trees (PLDI 2019)

https://dl.acm.org/citation.cfm?id=3314221.3314626

https://www.youtube.com/watch?v=j2henSFtZds

https://github.com/laithsakka/Grafter

This talk is not about Grafter!
Its about utilizing Clang to implement Grafter

1. Embedded DSL in C++

2. Static analysis

3. Code generation

Source to source

1. Easier to understand tree/graph programs on the source level.

2. Analyze dependences in tree/graph programs

3. Define high level transformations as source level rewrites

Embedded DSL in C++

- Annotate components
- Verify annotated components against a set of rules.

Annotations

```
class __tree_structure__ ValueNode : public Node {
public:
  int Value;
  __tree_child__ Node *Left, *Right;

  __tree_traversal__ void search(int Key, bool ValidCall) override {
    Found = false;
    if (Key == Value) {
      Found = true;
      return;
    }
    Left->search(Key, Key < Value);
    Right->search(Key, Key >= Value);
    Found = Left->Found || Right->Found;
  }
}
```

No aliasing

```
this->Left = this->Right;
this->Left = new ValueNode();
```

No condition calls

```
if(...)
  Left->insert(...)

Left->insert(...)
```

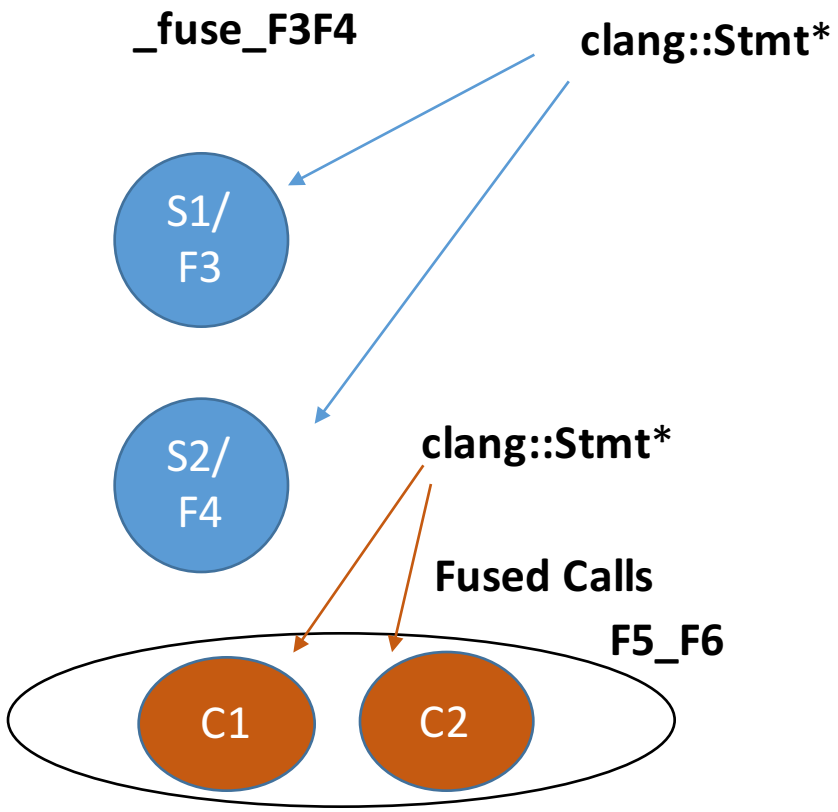
Static analysis

We used clang AST to collect information needed for Grafter static analysis

1. A function is represented as a sequence of Clang::Stmt* and Clang::CallExpr* nodes.

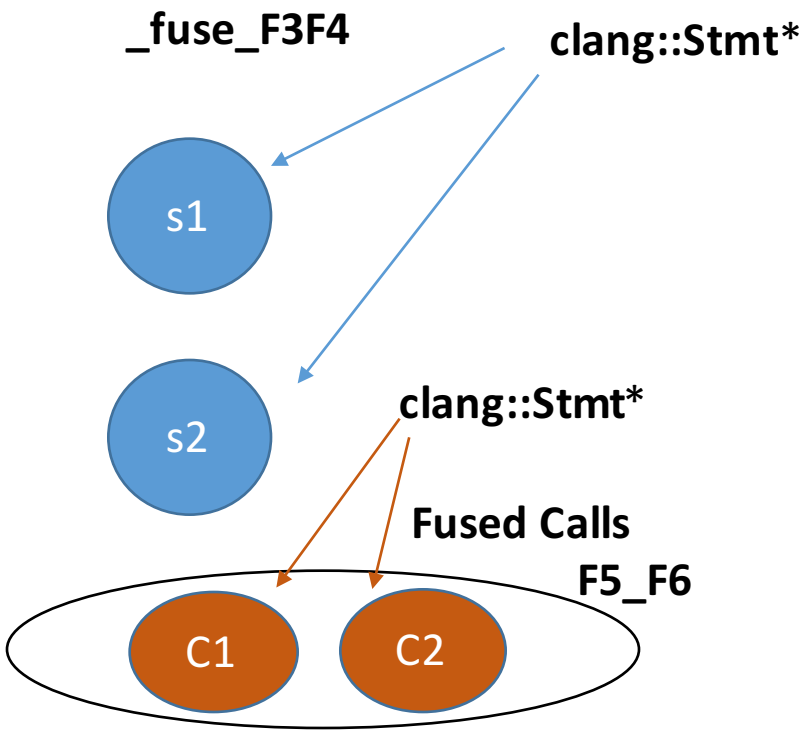
   Different schedules are achieved by reordering statements and collapsing calls.

2. Understand accesses of statements and build call graphs to analyze dependences.



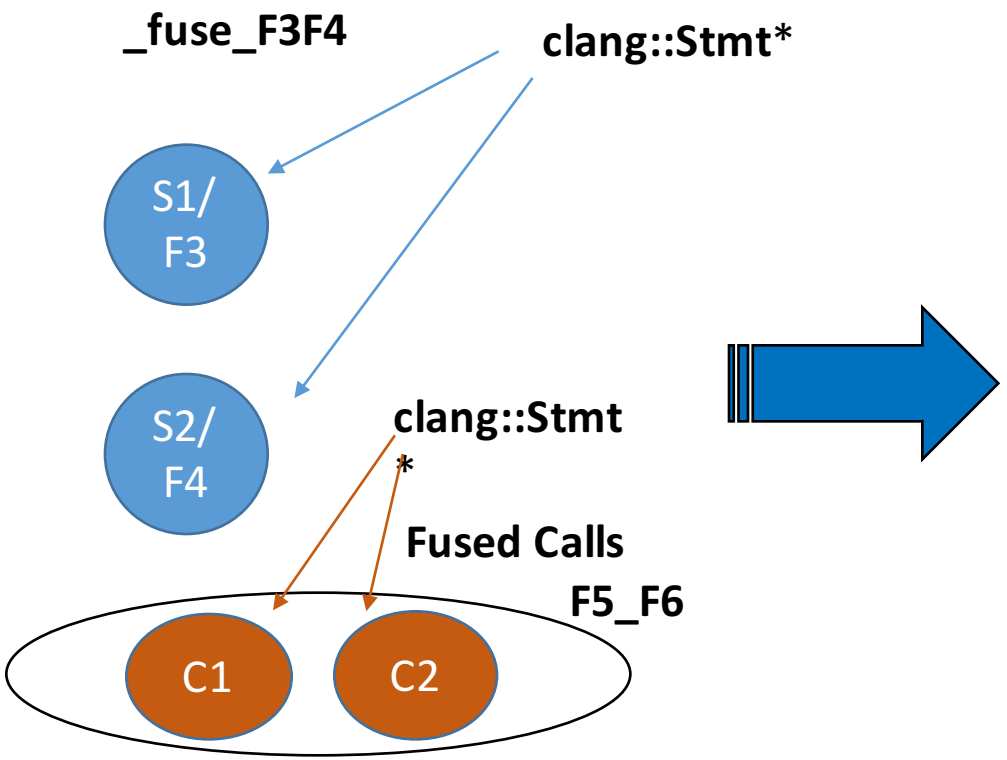
Code Generation

- Grafter build the fused functions incrementally following a set of rewrite rules while tracking the original source code




```
3  void _fuse__F3F4(TextBox *_r, int active_flags) {
4    TextBox *_r_f0 = (TextBox *)(_r);
5    TextBox *_r_f1 = (TextBox *)(_r);
6    if (active_flags & 0b11) /*call*/ {
7      unsigned int call_flags = 0;
8      call_flags <<= 1;
9      call_flags |= (0b01 & (active_flags >> 1));
10     call_flags <<= 1;
11     call_flags |= (0b01 & (active_flags >> 0));
12     _r_f0->Next->__stub1(call_flags);
13   }
14   if (active_flags & 0b1) {
15     _r_f0->Width = _r_f0->Text.Length;
16     _r_f0->TotalWidth = _r_f0->Next->Width + _r_f0
           ->Width;
17   }
18   if (active_flags & 0b10) {
19     _r_f1->Height = _r_f1->Text.Length * (_r_f1->
           Width / CHAR_WIDTH) + 1;
20     _r_f1->MaxHeight = _r_f1->Height;
21     if (_r_f1->Next->Height > _r_f1->Height) {
22       _r_f1->MaxHeight = _r_f1->Next->Height;
23     }
24   }
25 };
```

Code generation example



Parameters

Track active traversals

Call fused functions

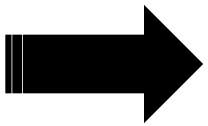
Computations

```
3  void _fuse__F3F4(TextBox *_r, int active_flags) {
4    TextBox *_r_f0 = (TextBox *)(_r);
5    TextBox *_r_f1 = (TextBox *)(_r);
6    if (active_flags & 0b11) /*call*/ {
7      unsigned int call_flags = 0;
8      call_flags <<= 1;
9      call_flags |= (0b01 & (active_flags >> 1));
10     call_flags <<= 1;
11     call_flags |= (0b01 & (active_flags >> 0));
12     _r_f0->Next->__stub1(call_flags);
13   }
14   if (active_flags & 0b1) {
15     _r_f0->Width = _r_f0->Text.Length;
16     _r_f0->TotalWidth = _r_f0->Next->Width + _r_f0
           ->Width;
17   }
18   if (active_flags & 0b10) {
19     _r_f1->Height = _r_f1->Text.Length * (_r_f1->
           Width / CHAR_WIDTH) + 1;
20     _r_f1->MaxHeight = _r_f1->Height;
21     if (_r_f1->Next->Height > _r_f1->Height) {
22       _r_f1->MaxHeight = _r_f1->Next->Height;
23     }
24   }
25 };
```

Code generation example

- Replace original calls with calls to fused functions

```
int main(){
  Element *ElementsList = ...;
  ElementsList->computeWidth();
  ElementsList->computeHeight();
}
```



```
void TextBox::__stub1(int active_flags) {
    _fuse__F3F4(this, active_flags);
}
void Group::__stub1(int active_flags) {
    _fuse__F5F6(this, active_flags);
}
void End::__stub1(int active_flags) {
    _fuse__F1F2(this, active_flags);
}
int main() {
    Group *ElementsList;
    //ElementsList->computeWidth();
    //ElementsList->computeHeight();
    ElementsList->__stub1(0b11);
}
```

It does scale..

- We run Grafter on programs with more than 50 functions
- Generating thousands lines of code
- Achieve significant speedups

Conclusions

- Clang can be useful in doing **performance-oriented** domain-specific source-level transformations

- Easy to implement an embedded DSL in C++

- Collect source-level information needed for static analysis

- Generate new source code while tracking the input program

2019 LLVM DEV MEETING

Improving Your TABLEGEN Description

Javed Absar

GRAPHCORE

WHAT IS ‘TABLEGEN’ ?

- DSL invented for LLVM
- Used extensively
- Describe records of –
  - Instructions, registers, intrinsics, attributes, scheduler model, warnings, …

THIS TALK IS **NOT…**

- Introduction to TableGen
- Complete Guide to TableGen

BUT…

Look at interesting features that you may

- Not know exist
- Know, but haven’t used
- Used, but not this way

‘FOREACH’ AND ‘CONCAT’

Listing One-by-One

```
def F9Dwarf  : DwarfMapping<28>;
def F11Dwarf : DwarfMapping<29>;
def F13Dwarf : DwarfMapping<30>;
def F15Dwarf : DwarfMapping<31>;


def F16Dwarf : DwarfMapping<68>;
def F18Dwarf : DwarfMapping<69>;
def F20Dwarf : DwarfMapping<70>;
def F22Dwarf : DwarfMapping<71>;
```

Source:

‘FOREACH’ AND ‘CONCAT’

Listing One-by-One

```
def F9Dwarf  : DwarfMapping<28>;
def F11Dwarf : DwarfMapping<29>;
def F13Dwarf : DwarfMapping<30>;
def F15Dwarf : DwarfMapping<31>;


def F16Dwarf : DwarfMapping<68>;
def F18Dwarf : DwarfMapping<69>;
def F20Dwarf : DwarfMapping<70>;
def F22Dwarf : DwarfMapping<71>;
```

Generating using ‘foreach’

```
1. def RegMap  {
2.   list<list<int>> val = [[9,28], [11,29], [13,30],
                            [15, 31],  [16, 68], [18, 69],
                            [20, 70], [22, 71]];
3. }


4. foreach N = RegMap.val  in {
5.   def F#!head(N)#Dwarf :
                    DwarfMapping<!head(!tail(N))>;
6. }
```

Source:

‘FOLD’

Listing One-by-One

```
def {
  list<int> Ascending = [1, 2, 3, 4, 5, 6, 7, 8, 9];
  list<int> Descending = [9, 8, 7, 6, 5, 4, 3, 2, 1];
}
```

Source:

‘FOLD’

Listing One-by-One

```
1. def {
2.   list<int> Ascending = [1, 2, 3, 4, 5, 6, 7, 8, 9];

3.   // list<int> Descending = [9, 8, 7, 6, 5, 4, 3, 2, 1];
4.   list<int> Descending = reverse<Ascending>.val;
5. }
```

Generating using ‘foldl’

```
1. class reverse<list<int> L> {
2.    list<int> val = !foldl([ ]<int>, L, a, b, [b] # a);
3. }
```

Source:

‘SORT’ USING TABLEGEN

Merge Sort Algorithm

```
def mergesort(L):
    if len(L) < 2:
        return L
    else:
        h = len(L) div 2
        return merge(mergesort(L[:h]), mergesort(L[h:]))
```

Source:

‘SORT’ USING TABLEGEN

Merge Sort Algorithm

```
def mergesort(L):
    if len(L) < 2:
        return L
    else:
        h = len(L) div 2
        return merge(mergesort(L[:h]), mergesort(L[h:]))
```

Using TableGen

```
1. class mergeSort<list<int> L> {
2.   list<int> val = !if(!lt(!size(L),2), L,
3.                       merge<
4.                             mergeSort< split<L>.lower >.val,
5.                             mergeSort< split<L>.upper >.val
6.                           >.val );
7. }
```

Source:

‘SORT’ USING TABLEGEN

Using TableGen

```
1. class split<list<int> L> {
2.    int sizeW = !size(L);

3.    int sizeL = !srl(!add(sizeW,1),1);
4.    int sizeU = !add(sizeW,!mul(sizeL,-1));

5.    list<int> lower = !foldl([ ]<int>, L, a, b, !if(!lt(!size(a),sizeL),
6.                                                    a # [b],
7.                                                    a));
8.   list<int> upper = !foldl(L, L, a, b, !if(!gt(!size(a), sizeU),
9.                                             !tail(a),
10.                                            a));
11.}
```

Source:

‘SORT’ USING TABLEGEN

Merge Function (Functional Programming)

```
def merge(a, b):
    if len(a) == 0: return b
    elif len(b) == 0: return a
    elif a[0] < b[0]:
        return [a[0]] + merge(a[1:], b)
    else:
        return [b[0]] + merge(a, b[1:])
```

Source:

‘SORT’ USING TABLEGEN

Merge Function (Functional Programming)

```
def merge(a, b):
    if len(a) == 0: return b
    elif len(b) == 0: return a
    elif a[0] < b[0]:
        return [a[0]] + merge(a[1:], b)
    else:
        return [b[0]] + merge(a, b[1:])
```

Using TableGen

```
1. class merge<list<int> a, list<int> b> {
2.    list<int> val = !if(!eq(!size(a), 0), b,
3.                        !if(!eq(!size(b),0), a,
4.                            !if(!lt(!head(a), !head(b)), [!head(a)] # merge<!tail(a), b>.val,
5.                                [!head(b)] # merge<a, !tail(b)>.val)));
6. }
```

Source:

STATIC POLYMORPHISM WITH TABLEGEN

Test Code

```
1. class GeoObj { string draw = "Invalid"; }

2. class Line<int N> : GeoObj { string draw = "Line:" # N; }
3. class Rectangle<int N> : GeoObj { string draw = "Rectangle:" # N; }

4. class myDraw<list<GeoObj> objs> {
5.       list<string> drawings = !foreach(obj, objs, obj.draw);
6. }

7. def main {
8.       GeoObj o1 = Line<1>;   GeoObj o2 = Line<2>;   GeoObj o3 = Rectangle<3>;
9.        list<string> D = myDraw<[o1, o2, o3]>.drawings;
10. }
```

Source:

STATIC POLYMORPHISM WITH TABLEGEN

Test Code

```
1. class GeoObj { string draw = "Invalid"; }
2. class Line<int N> : GeoObj { string draw = "Line:" # N; }
3. class Rectangle<int N> : GeoObj { string draw = "Rectangle:" # N; }
4. class myDraw<list<GeoObj> objs> {
5.       list<string> drawings = !foreach(obj, objs, obj.draw);
6. }

7. def main {
8.       GeoObj o1 = Line<1>;   GeoObj o2 = Line<2>;   GeoObj o3 = Rectangle<3>;
9.        list<string> D = myDraw<[o1, o2, o3]>.drawings;
10. }
```

Generated Definition

```
$ ./bin/llvm-tblgen static_polymorphism.td
def main {
  …
 list<string> D = ["Line:1", "Line:2", "Rectangle:3"];
}
```

Source:

‘COND’

Using Nested-‘IF’s

```
class getSubRegs<int size> {
  list<SubRegIndex> ret2 = [sub0, sub1];
  list<SubRegIndex> ret3 = [sub0, sub1, sub2];
  list<SubRegIndex> ret4 = [sub0, sub1, sub2, sub3];
  list<SubRegIndex> ret5 = [sub0, sub1, sub2, sub3, sub4];
  list<SubRegIndex> ret8 = [sub0, sub1, sub2, sub3, sub4, sub5,
sub6, sub7];
  list<SubRegIndex> ret16 = [sub0, sub1, sub2, sub3,
                     sub4, sub5, sub6, sub7,
                     sub8, sub9, sub10, sub11,
                     sub12, sub13, sub14, sub15];
  list<SubRegIndex> ret = !if(!eq(size, 2), ret2,
                                !if(!eq(size, 3), ret3,
                                   !if(!eq(size, 4), ret4,
                                      !if(!eq(size, 5), ret5,
                                         !if(!eq(size, 8), ret8,
ret16))))); }
```

Using ‘COND’

```
class getSubRegs<int size> {
  list<SubRegIndex> ret2 = [sub0, sub1];
  list<SubRegIndex> ret3 = [sub0, sub1, sub2];
  list<SubRegIndex> ret4 = [sub0, sub1, sub2, sub3];
  list<SubRegIndex> ret5 = [sub0, sub1, sub2, sub3, sub4];
  list<SubRegIndex> ret8 = [sub0, sub1, sub2, sub3, sub4, sub5,
sub6, sub7];

 list<SubRegIndex> ret_opt = !cond(!eq(size, 2):  ret2,
                                   !eq(size, 3):  ret3,
                                   !eq(size, 4):  ret4,
                                   !eq(size, 5):  ret5,
                                   !eq(size, 8):  ret8,
                                   !eq(1, 1):     ret16);
}
```

Source:

CONCLUSION

Take a look at the video and apply it to your context!

2019 LLVM DEV MEETING