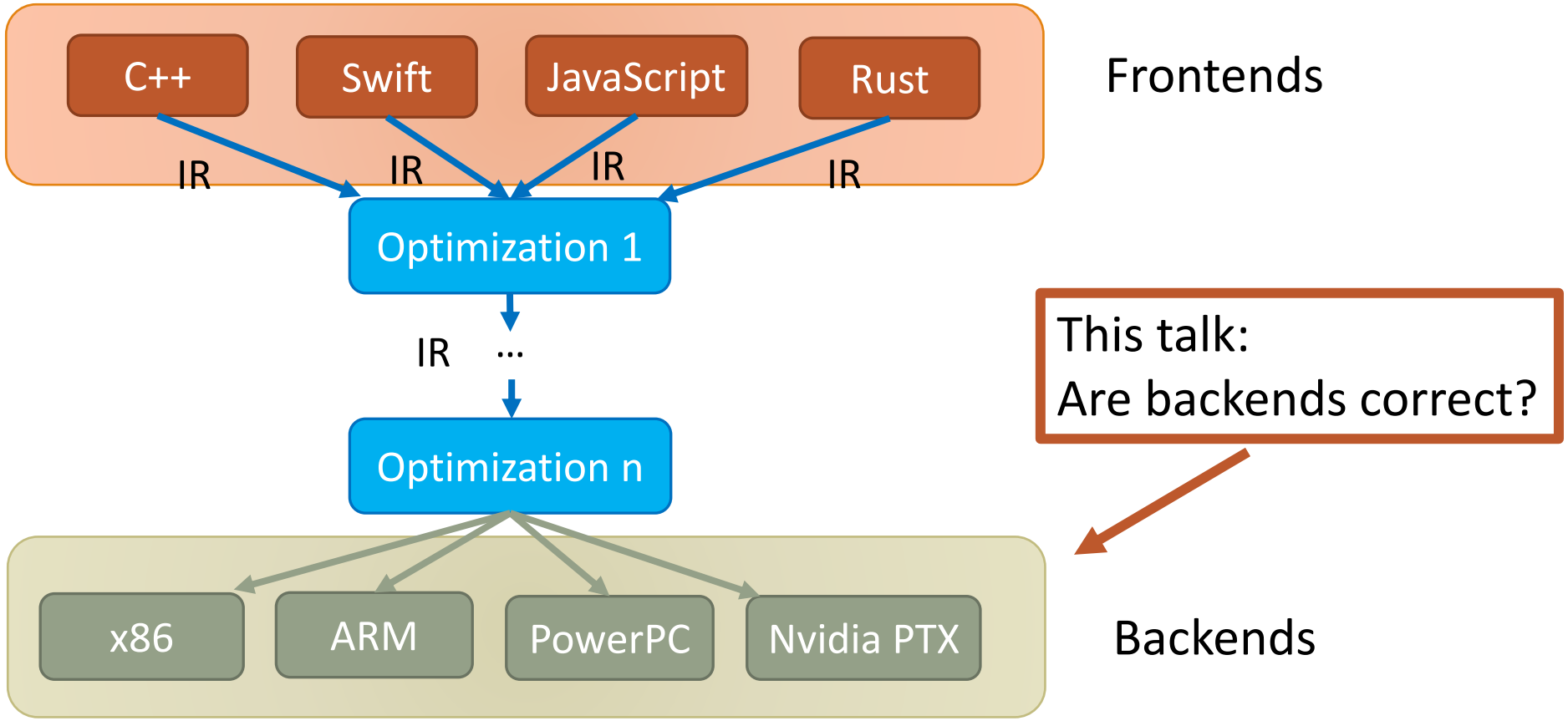
Translation Validation for LLVM's AArch64 Backend

Ryan Berger (Nvidia), Mitch Briles (Utah), Nader Bushehri (Utah), Nicholas Coughlin (Queensland), Kait Lam (Queensland), Nuno Lopes (Lisbon), Stefan Mada (Utah), Tanmay Tirpankar (Utah), John Regehr (Utah)

Typical Compiler



What's in a Backend?

Historical answer:

- Register allocation
- Instruction selection
- Instruction scheduling

Modern answer is the old stuff +

- Dead code elimination
- Loop invariant code motion
- Algebraic simplifications
- Common subexpression elimination
- Stack coloring
- ...

Are Compiler Backends Correct?

LLVM's Target and CodeGen directories have > 1M LoC

This much code…

- Creates plenty of opportunities for bugs
- Is challenging to adequately test

Compiler Bugs = Security Vulnerabilities

```
%a = load <16 x i8>, ptr %p
%b = insertelement <16 x i8> %a, i8 %val, i32 %idx
store <16 x i8> %b, ptr %p
```

```
strb    w1, [x0, w2, uxtw]  ; *(%p + %idx) = new byte value
```

Can store anywhere!

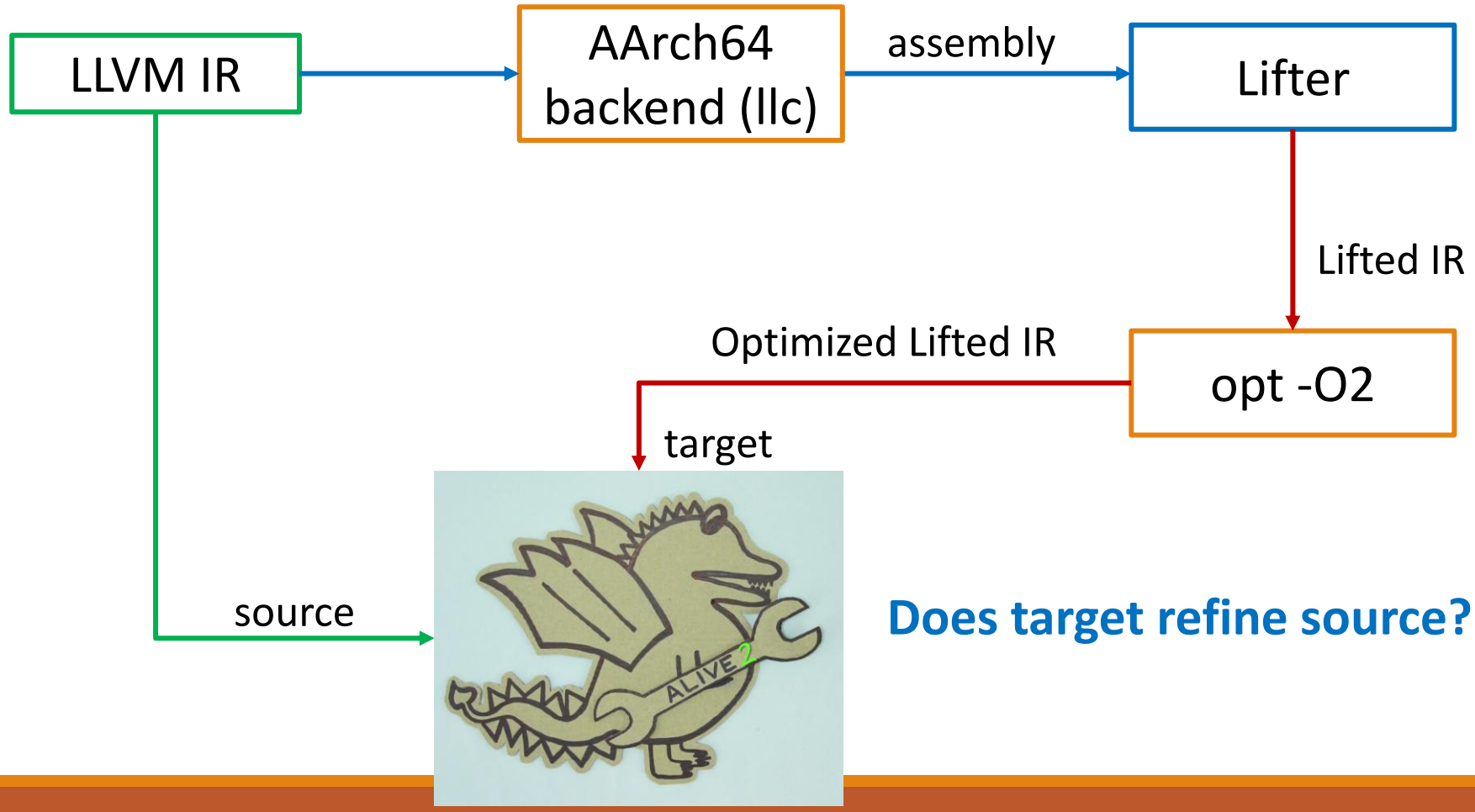
"If idx exceeds the length of val for a fixed-length vector, the result is a poison value." (LLVM IR spec aka LangRef)

```
and     x8, x2, #0xf  ; reduce %idx modulo 16
strb    w1, [x0, x8]  ; *(%p + %idx) = new byte value
```

Translation Validation (TV) to the Rescue

- Proving that a single execution of a compiler is correct
- Alive2 successfully productized TV for compiler optimizations
  - Found hundreds of bugs in LLVM
  - 1,300+ LLVM commits mention Alive2 (to certify the correctness of the commit)
- This paper: applying TV for Aarch64 (ARM64) backend

ARM-TV: Lifter + Alive2



ARM-TV Example

```
1  define i32 @foo() {
2    br label %1
3  1:
4    %2 = phi i32 [ 1, %0 ], [ %4, %1 ]
5    %3 = phi i8 [ 0, %0 ], [ %5, %1 ]
6    %4 = and i32 %2, 4
7    %5 = add i8 %3, -1
8    %6 = icmp eq i8 %5, 0
9    br i1 %6, label %7, label %1
10 7:
11   ret i32 %4
12 }
```

⬇ LLVM's llc

```
1 foo:
2   mov     w8, #0
3   mov     w0, #1
4 .LBB0_1:
5   and     w0, w0, #0x4
6   sub     w8, w8, #1
7   tst     w8, #0xff
8   b.ne    .LBB0_1
9   ret
```

Is refined by?

⇒

ARM-TV lifter

```
1  define i32 @foo() {
2  i8_loop:
3    %X8_0 = zext i32 0 to i64
4    %X0_0 = zext i32 1 to i64
5    br label %.LBB0_1
6 .LBB0_1:
7    %X0_1 = phi i64 [ %X0_0, %i8_loop ], [ %X0_4, %.LBB0_1]
8    %X8_1 = phi i64 [ %X8_0, %i8_loop ], [ %X8_4, %.LBB0_1]
9    %X0_2 = trunc i64 %X0_1 to i32
10   %X0_3 = and i32 %X0_2, 4
11   %X0_4 = zext i32 %X0_3 to i64
12   %X8_2 = trunc i64 %X8_1 to i32
13   %X8_3 = sub i32 %X8_2, 1
14   %X8_4 = zext i32 %X8_3 to i64
15   %Z_0 = trunc i64 %X8_4 to i32
16   %Z_1 = and i32 %Z_0, 255
17   %Z_2 = icmp eq i32 %Z_1, 0
18   %Z_3 = xor i1 %Z_2, 1
19   br i1 %Z_3, label %.LBB0_1, label %i8_loop-tgt1
20 i8_loop-tgt1:
21   %RES = trunc i64 %X0_4 to i32
22   ret i32 %RES
23 }
```

Generating the Lifter w/ Partial Evaluation From ARM Spec

- Keep only relevant machine state (e.g., always in user mode)

```
bits(datasize) operand1 = V[n];
bits(datasize) operand2 = V[m];
bits(datasize) result;
for e = 0 to elements-1
   bits(esize) element1 = Elem[operand1, e, esize];
   bits(esize) element2 = Elem[operand2, e, esize];
   if sub_op then
      Elem[result, e, esize] = element1 - element2;
   else
      Elem[result, e, esize] = element1 + element2;
V[d] = result;
```

ASL specification

```
V[0] = ZeroExtend(add_vec(
     V[0][0 +: 64], V[1][0 +: 64], 8), 128);
```

Reduced ASL

```
%a4_5 = load i128, ptr %Q0, align 1
%a4_6 = trunc i128 %a4_5 to i64
%a4_7 = load i128, ptr %Q1, align 1
%a4_8 = trunc i128 %a4_7 to i64
%a4_9 = bitcast i64 %a4_6 to <8 x i8>
%a4_10 = bitcast i64 %a4_8 to <8 x i8>
%a4_11 = add <8 x i8> %a4_9, %a4_10
%a4_12 = bitcast <8 x i8> %a4_11 to i64
%a4_14 = zext i64 %a4_12 to i128
store i64 %a4_14, ptr %Q0, align 1
```

```
add v0.8b, v0.8b, v1.8b
```

Checking ABI Compliance

- Caller and callee saved registers
- Sign/zero-extended arguments
- Stack alignment
- Padding of non-byte-aligned memory accesses

Alive2: Assembly Mode

- Lifted LLVM IR is valid, but it is not a refinement of source!
- Assembly uses a flat memory model, IR doesn't

```
void fn(int *p, int *q) {
  if (p == q)
    *p = 3;
  else
    *q = 3;
}
```

```
mov     w8, #3
str     w8, [x1]    ; store though q
ret
```

Alive2: Assembly Mode

- Extended Alive2 with an assembly mode:
  - Added flat memory model
  - Allow source/target to have different memory models
  - No undef/poison in assembly
  - A few SMT encoding optimizations

Conclusion

- Compilers have bugs
  - Some introduce security vulnerabilities in compiled programs
- Translation validation (TV) is a practical approach to identify bugs in compilers
- ARM-TV reuses existing TV framework to find 45 bugs in Aarch64 backend:
  - Lifts assembly to LLVM IR
  - Checks ABI compliance
  - New assembly mode for Alive2
  - Lifter generated automatically from ARM specification!