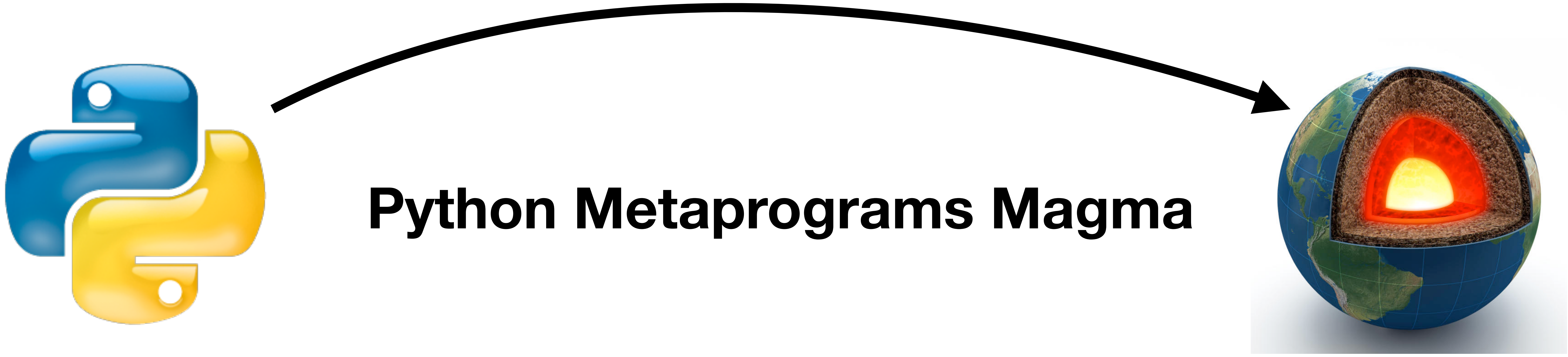
The use of MLIR in Magma

Raj Setaluri

AHA Affiliates Meeting 4/6/22

Magma background



Generates Circuits

Represents Circuits

Python Metaprograms Magma

Python

Magma

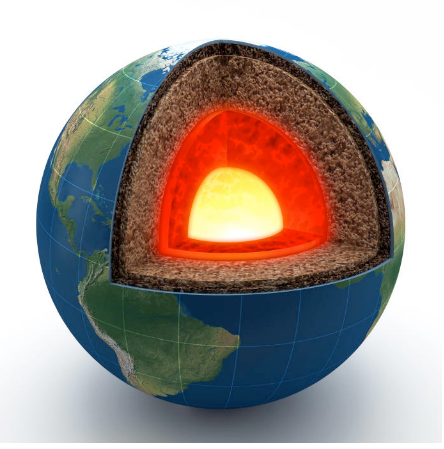
Magma background

```
class Foo(m.Circuit):
    T = m.UInt[8]
    io = m.IO(a=m.In(T), b=m.In(T), y=m.Out(T))
    io.y @= io.a + io.b


class Top(m.Circuit):
    N, T = 2, m.UInt[8]
    io = m.IO(I=m.In(m.Array[N, T]), O=m.Out(T))
    curr = io.I[0]
    for i in range(1, N):
        y = Foo()(curr, io.I[i])
        curr = m.register(y)
    io.O @= curr
```

Python

- **Dynamically-typed, polymorphic**
- **Widely used**



Magma

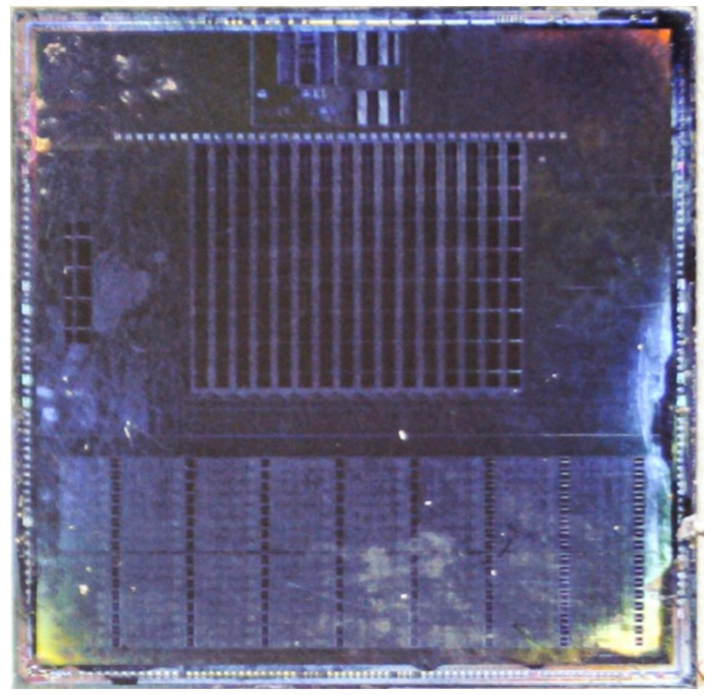
- **Magma embedded in Python**
- **Similar syntax, shared lexical state**
- **Static product types**
- **Combinational and sequential logic**
- **Guaranteed to be synthesizable**

Python-Magma system

- **Python metaprograms Magma**

Magma background

- Magma used in academia: *Garnet*, *Amber*, *Onyx* chips
- Magma used in industry
    - Major block in custom accelerator @ Facebook
    - 5mm^2 at 5nm technology (~150M-200M transistors)
- More users, more complex designs -> more demands!

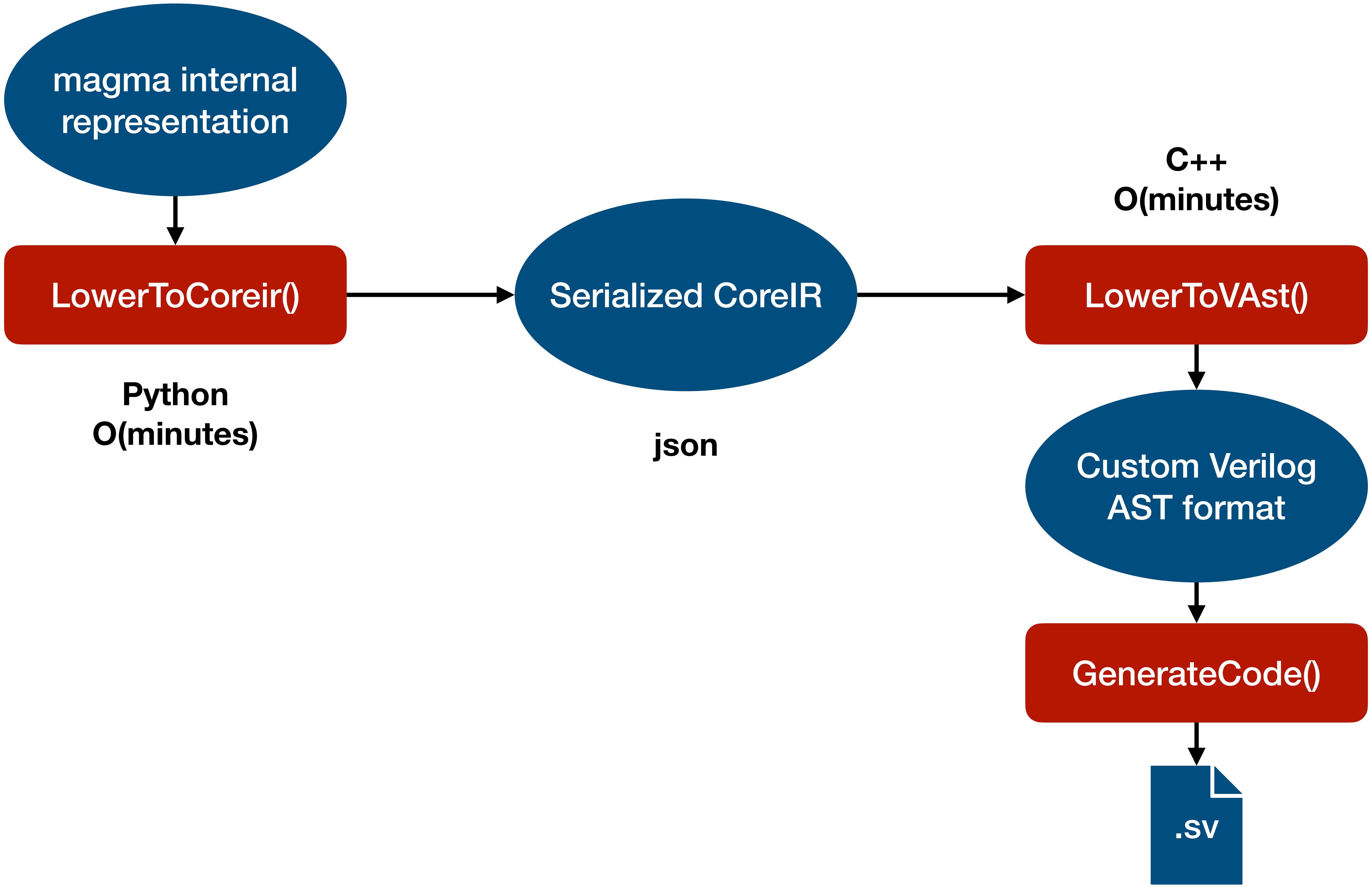


Amber

Scaling Magma

- Runtime performance
- Debuggability
- Code generation
- CAD flow integration

Magma toolchain



magma internal representation

LowerToCoreir()

Python
O(minutes)

Serialized CoreIR

json

C++
O(minutes)

LowerToVAst()

Custom Verilog AST format

GenerateCode()

.sv

Existing HW toolchains

- FIRRTL (Chisel IR)
    - Lo-FIRRTL, Hi-FIRRTL
- LLHD IR
    - Behavioral IR, Structural IR, Netlist IR
- RTLIL (yosys IR), LNAST, LGraph, …

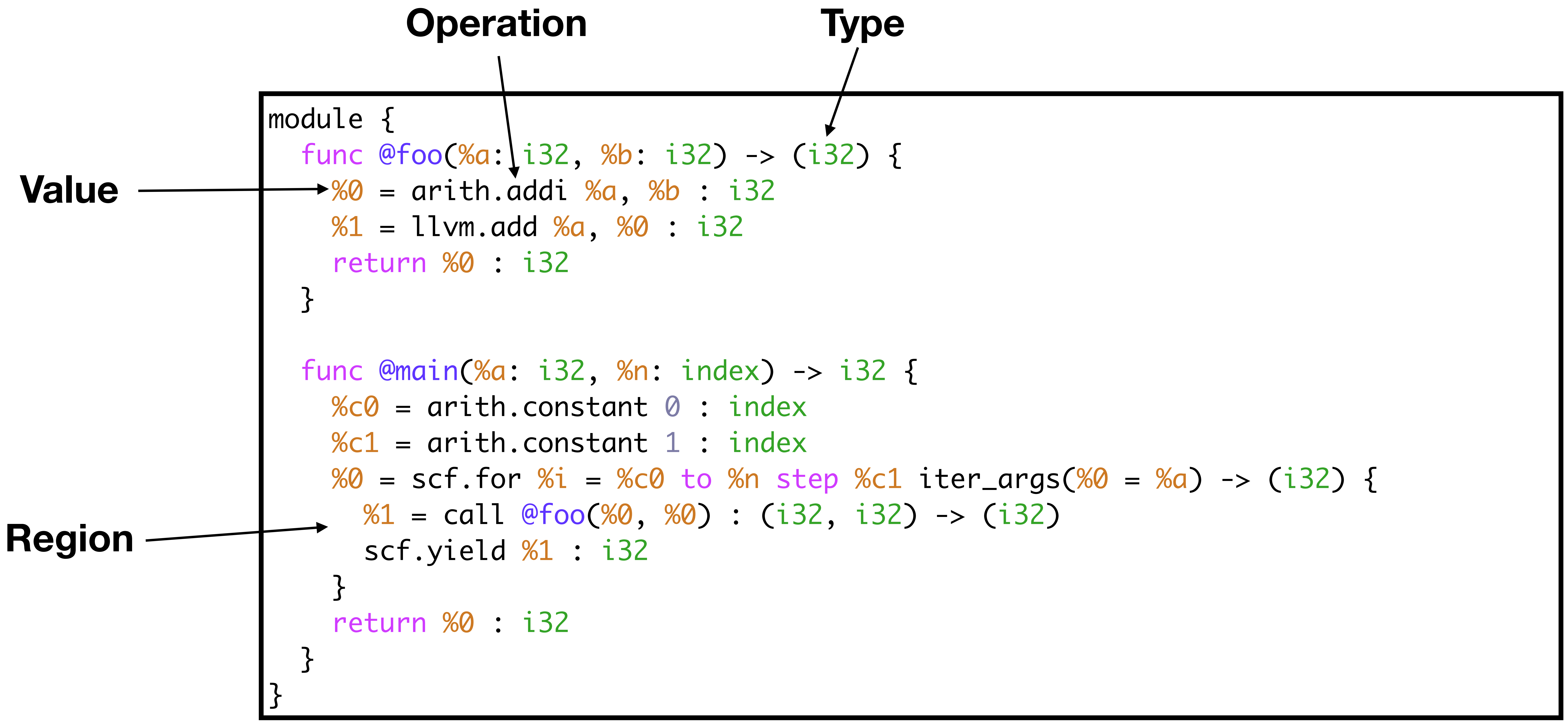
MLIR toolchain

- *Multi-level Intermediate Representation*
- Originally Google open source project
    - Intended as machine learning DSL target
- Super flexible IR builder + community of IR’s

MLIR structure

- Graph of *Operations* (nodes) and typed *Values* (edges)
- *Dialect*: collection of types, operations, passes
- Recursive structure
    - *Regions* contain *Blocks*
    - *Blocks* contain *Operations*
    - *Operations* can contain *Regions*

MLIR structure



Operation

Type

Value

Region

```
module {
  func @foo(%a: i32, %b: i32) -> (i32) {
    %0 = arith.addi %a, %b : i32
    %1 = llvm.add %a, %0 : i32
    return %0 : i32
  }

  func @main(%a: i32, %n: index) -> i32 {
    %c0 = arith.constant 0 : index
    %c1 = arith.constant 1 : index
    %0 = scf.for %i = %c0 to %n step %c1 iter_args(%0 = %a) -> (i32) {
      %1 = call @foo(%0, %0) : (i32, i32) -> (i32)
      scf.yield %1 : i32
    }
    return %0 : i32
  }
}
```

MLIR structure

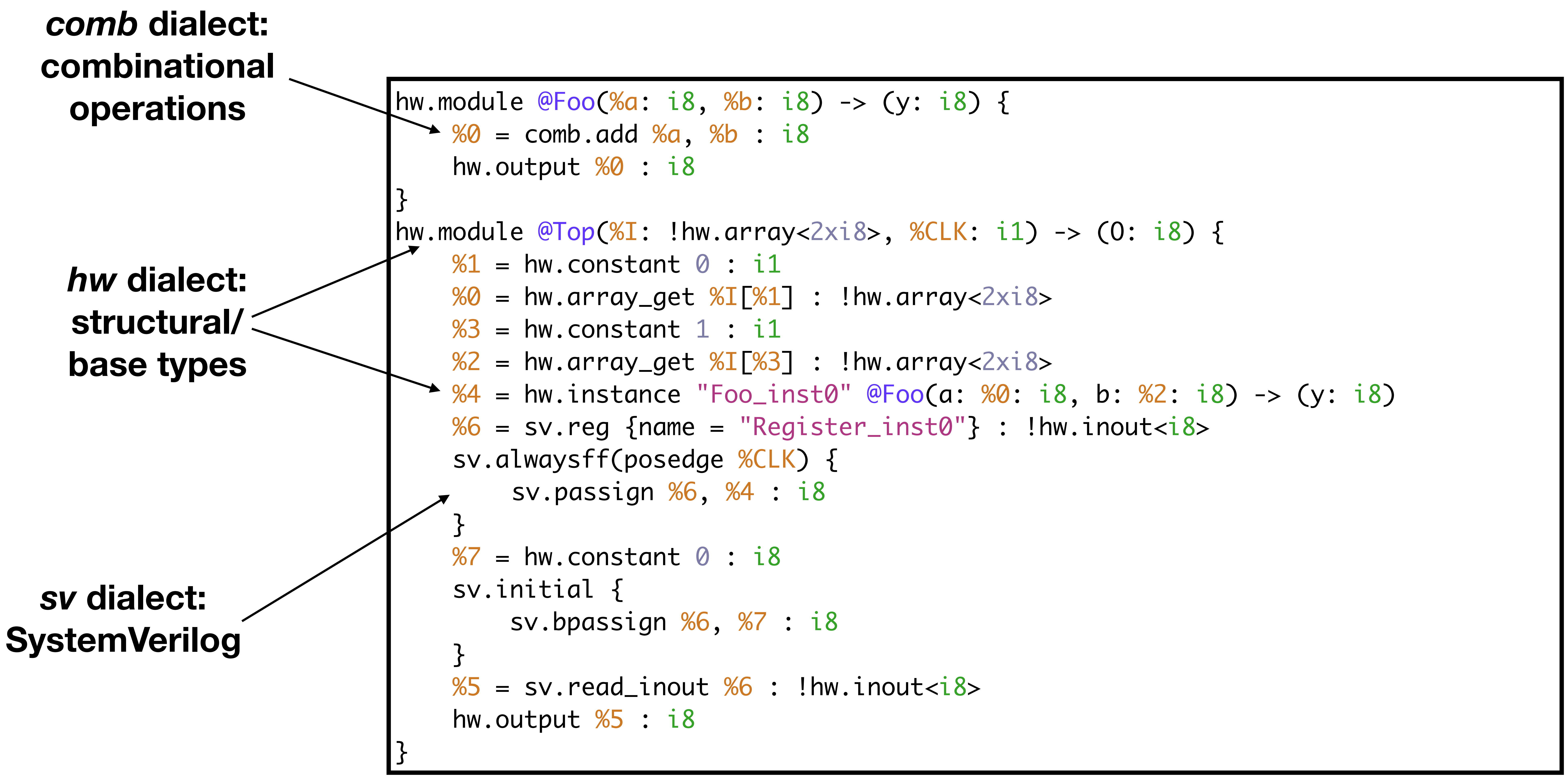
- Inter-operation of specialized dialects + common representation =
    - Reusable common passes, e.g. common subexpression elim.
    - Domain (dialect) specific passes, e.g. loop transformations

CIRCT community

- *Circuit IR Compilers and Tools*
- Collection of hardware specific MLIR dialects!
- Strong industry involvement

CIRCT community

CIRCT example



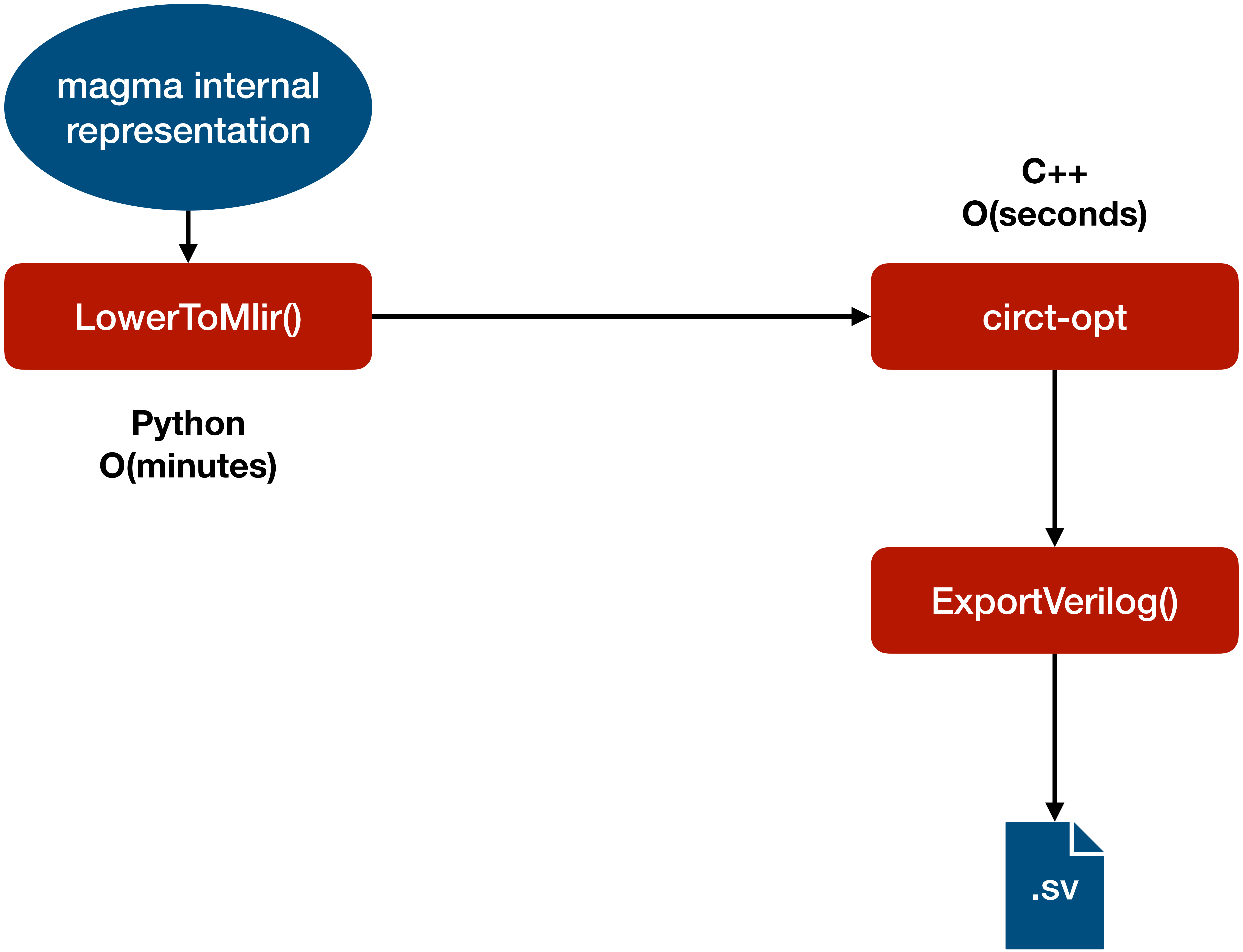
***comb* dialect: combinational operations**

***hw* dialect: structural/ base types**

***sv* dialect: SystemVerilog**

```
hw.module @Foo(%a: i8, %b: i8) -> (y: i8) {
    %0 = comb.add %a, %b : i8
    hw.output %0 : i8
}
hw.module @Top(%I: !hw.array<2xi8>, %CLK: i1) -> (O: i8) {
    %1 = hw.constant 0 : i1
    %0 = hw.array_get %I[%1] : !hw.array<2xi8>
    %3 = hw.constant 1 : i1
    %2 = hw.array_get %I[%3] : !hw.array<2xi8>
    %4 = hw.instance "Foo_inst0" @Foo(a: %0: i8, b: %2: i8) -> (y: i8)
    %6 = sv.reg {name = "Register_inst0"} : !hw.inout<i8>
    sv.alwaysff(posedge %CLK) {
        sv.passign %6, %4 : i8
    }
    %7 = hw.constant 0 : i8
    sv.initial {
        sv.bpassign %6, %7 : i8
    }
    %5 = sv.read_inout %6 : !hw.inout<i8>
    hw.output %5 : i8
}
```

Magma-to-MLIR flow



magma internal representation

LowerToMlir()

Python
O(minutes)

C++
O(seconds)

circt-opt

ExportVerilog()

.sv

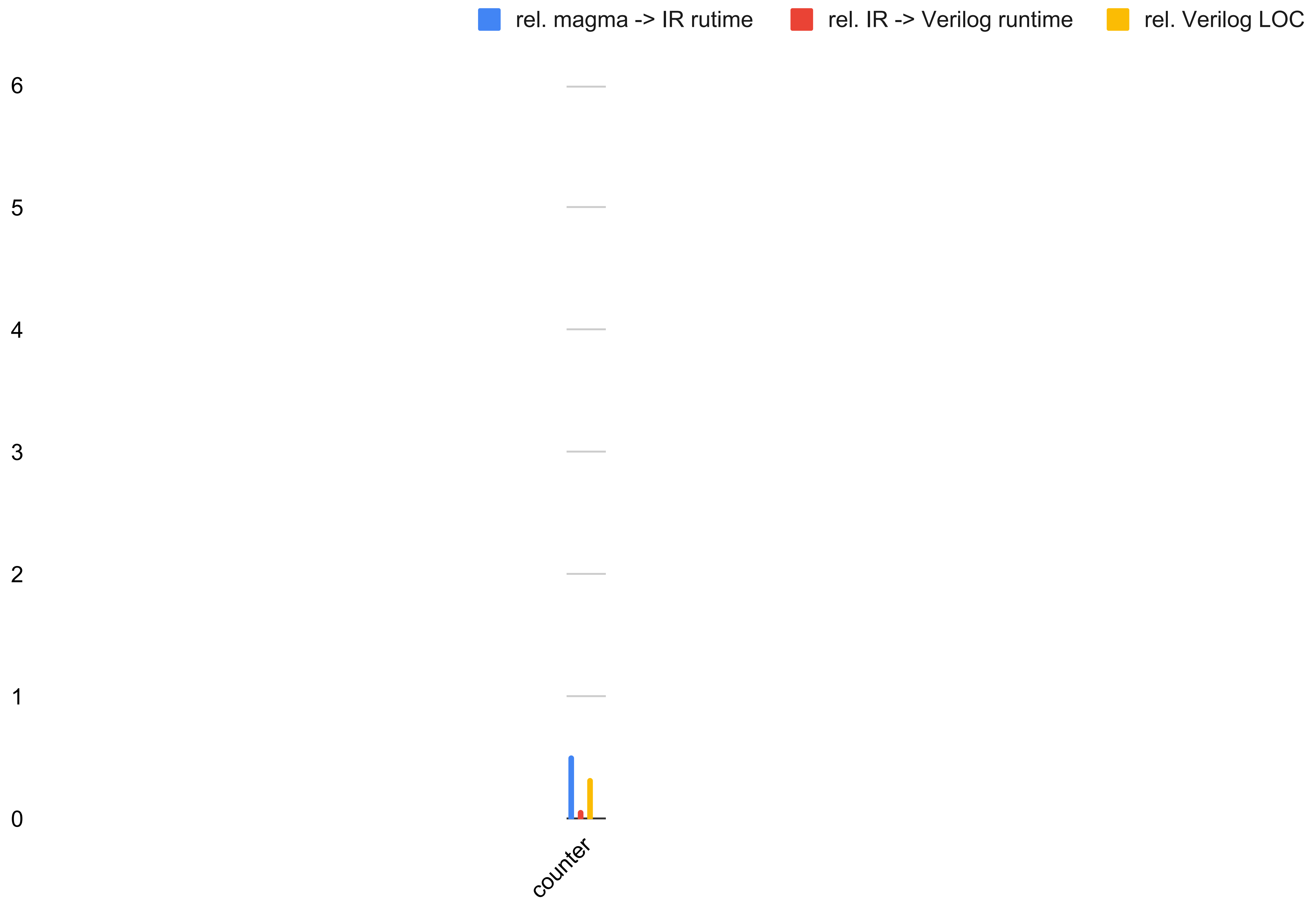
Results

Runtime performance

- Facebook flows bottlenecked by magma performance
- 10min to generate Verilog, ~split evenly between:
    - Python frontend execution
    - Magma-to-IR generation
    - IR-to-Verilog generation

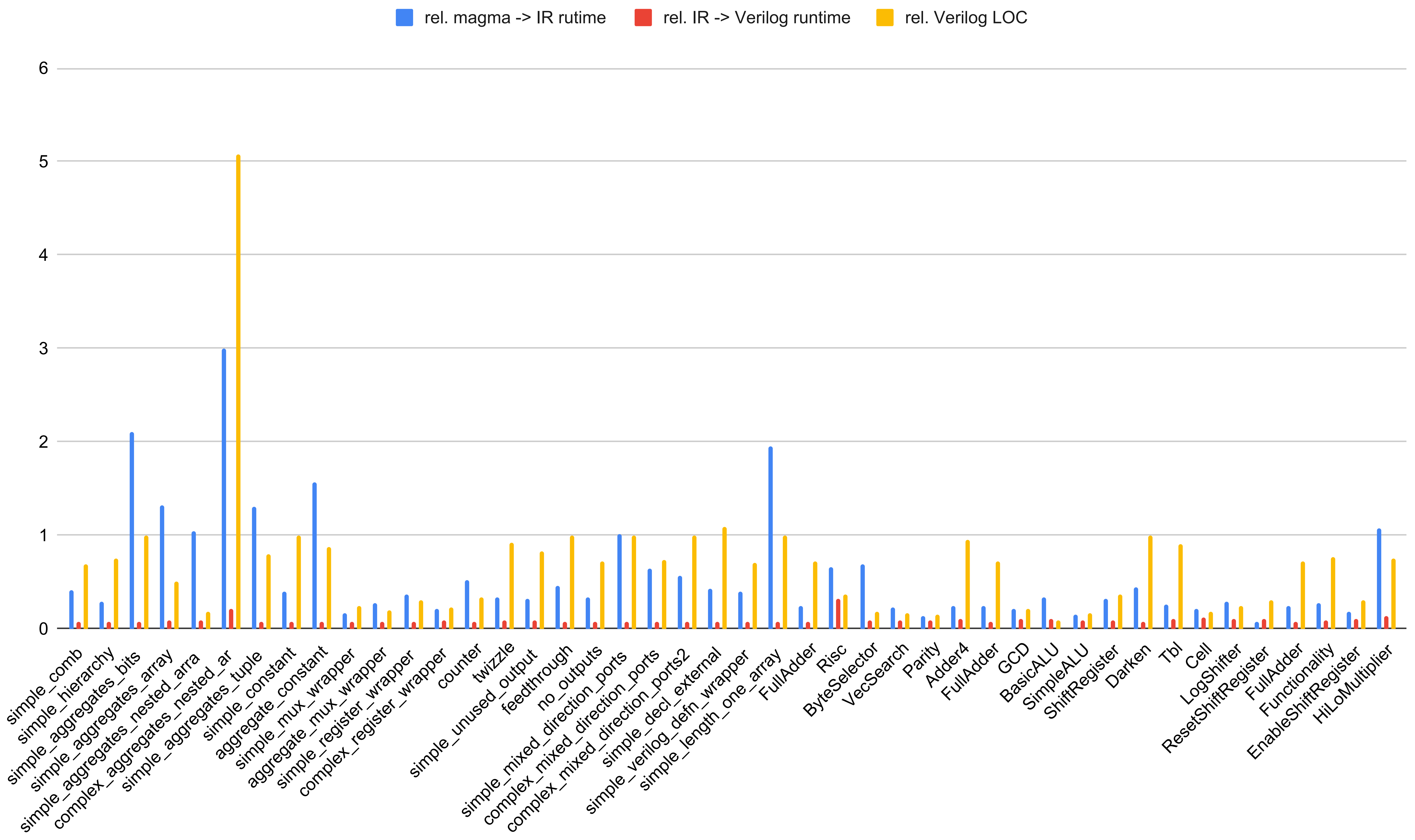
Runtime performance

Microbenchmarks



Runtime performance

Microbenchmarks



Debuggability

Source locations

```
hw.module @Foo(%a: i8, %b: i8) -> (y: i8) {
    %0 = comb.add %a, %b : i8 loc("example.py":11:17)
    hw.output %0 : i8 loc("example.py":11:9)
} loc("example.py":8:0)
```

example.mlir

```
module Foo(     // example.py:8
  input  [7:0] a, b,
  output [7:0] y);

  assign y = a + b;     // example.py:11:{9,17}
endmodule
```

example.sv

Debuggability

- Referenceable symbols are first-class citizens
- *Upcoming work*: Integrate with Keyi’s deugging framework

CAD flow integration

- Verification
  - SV bind flow
  - Assertions, Logging
- Synthesis
  - Single golden RTL for PD and DV using `ifdef

Code generation

Clock-gated registers: CoreIR

Synthesis can’t infer a clock gate!

```
module Register (
    input [7:0] I,
    output [7:0] O,
    input CE,
    input CLK,
    input ASYNCRESET
);
  wire [7:0] I_enable = CE ? I : 0;
  reg [7:0] data;
  always @(posedge CLK, posedge ASYNCRESET) begin
    if (ASYNCRESET) data <= 0;
    else data <= I_enable;
  end
  assign O = data;
);
endmodule
```

Code generation

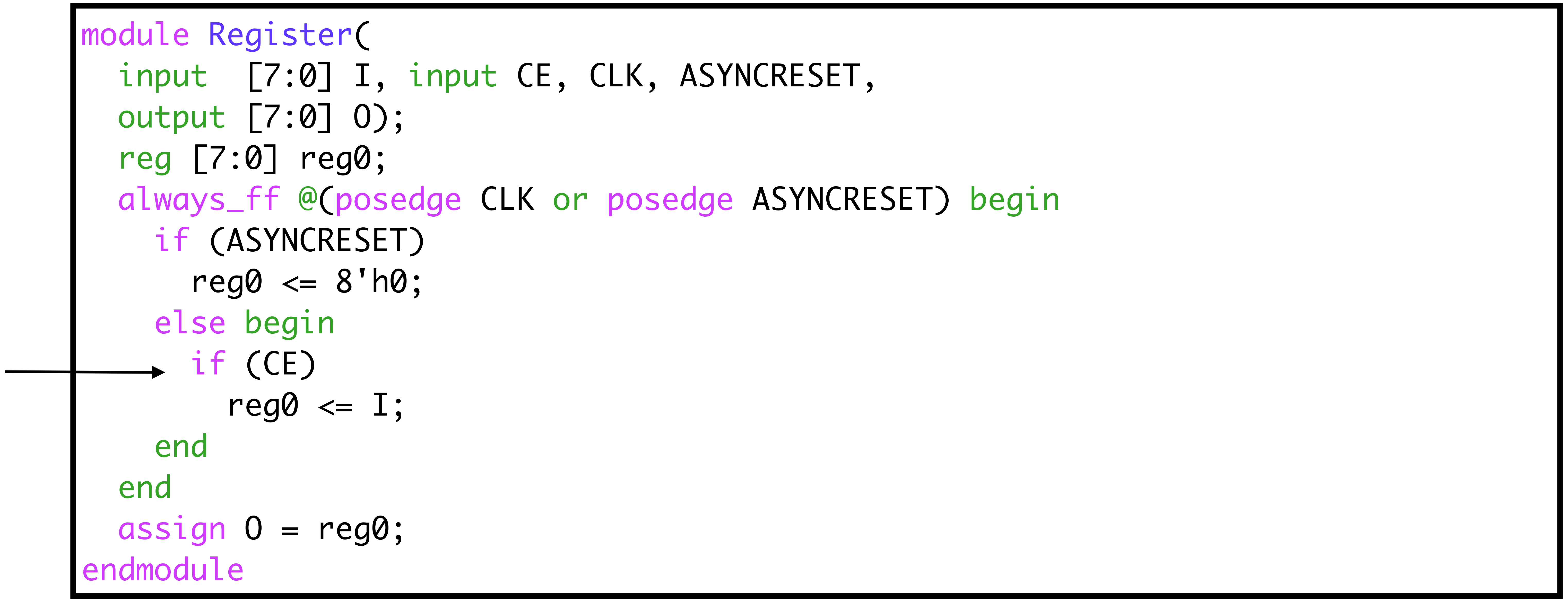
Clock-gated registers: MLIR

```
hw.module @Register(%I: i8, %CE: i1, %CLK: i1, %ASYNCRESET: i1) -> (O: i8) {
    %1 = sv.reg {name = "reg0"} : !hw.inout<i8>
    %2 = hw.constant 0 : i8
    sv.alwaysff(posedge %CLK) {
        sv.if %CE {
            sv.passign %1, %I : i8
        }
    } (asyncreset : posedge %ASYNCRESET) {
        sv.passign %1, %2 : i8
    }
    %0 = sv.read_inout %1 : !hw.inout<i8>
    hw.output %0 : i8
}
```

First-class support for reset, if-stmt

Code generation

Clock-gated registers: MLIR



```
module Register(
  input  [7:0] I, input CE, CLK, ASYNCRESET,
  output [7:0] O);
  reg [7:0] reg0;
  always_ff @(posedge CLK or posedge ASYNCRESET) begin
    if (ASYNCRESET)
      reg0 <= 8'h0;
    else begin
      if (CE)
        reg0 <= I;
    end
  end
  assign O = reg0;
endmodule
```

Synthesis infers clock gate!

Code generation

- SV structs
- Direct support for verbatim SV (escape hatch)
- Code “prettification”: e.g. CSE, DCE, canonicalization
    - Can greatly reduce code size

Conclusion

- CoreIR represents one point in the hardware IR space
    - MLIR allows us to mix multiple levels of representation
- Benefit of large OS community and mature, performant code base

Try magma!

https://github.com/phanrahan/magma