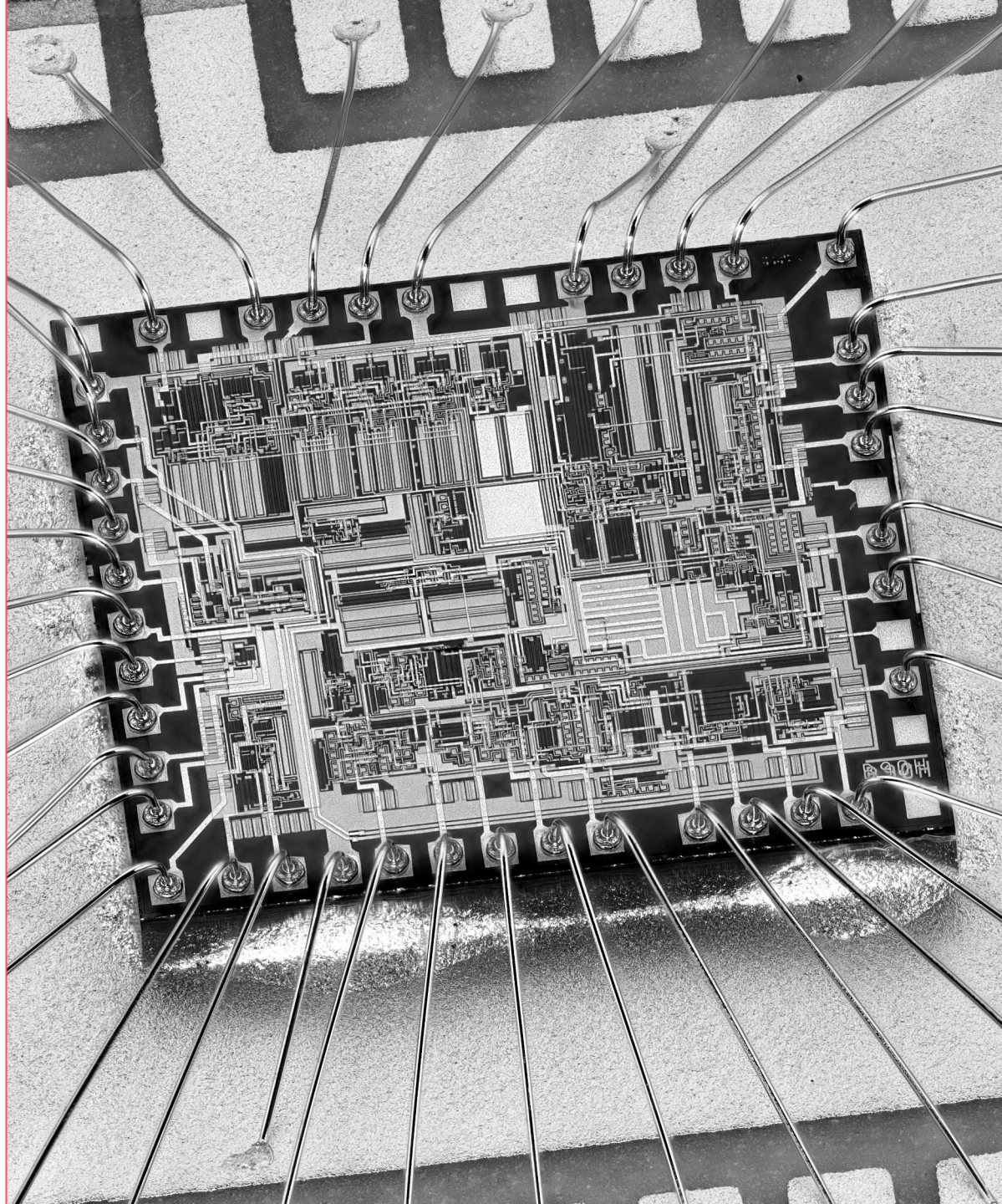
Build your logic design tool with CIRCT

Stefan Wallentowitz
HM Munich University of Applied Sciences
HeiChips 2025



VM or not

- If you are not using VM:

https://github.com/**towoe**/heichips-circt-lab

HM

About Me

Professor at Hochschule München University of Applied Sciences
Computer Architecture, Computer Engineering, Positions available!

Vice-Chair of the RISC-V Board of Directors
Community individuals representative, strategy and events

Director at Free and Open Source Silicon Foundation
Advocacy, community support, events

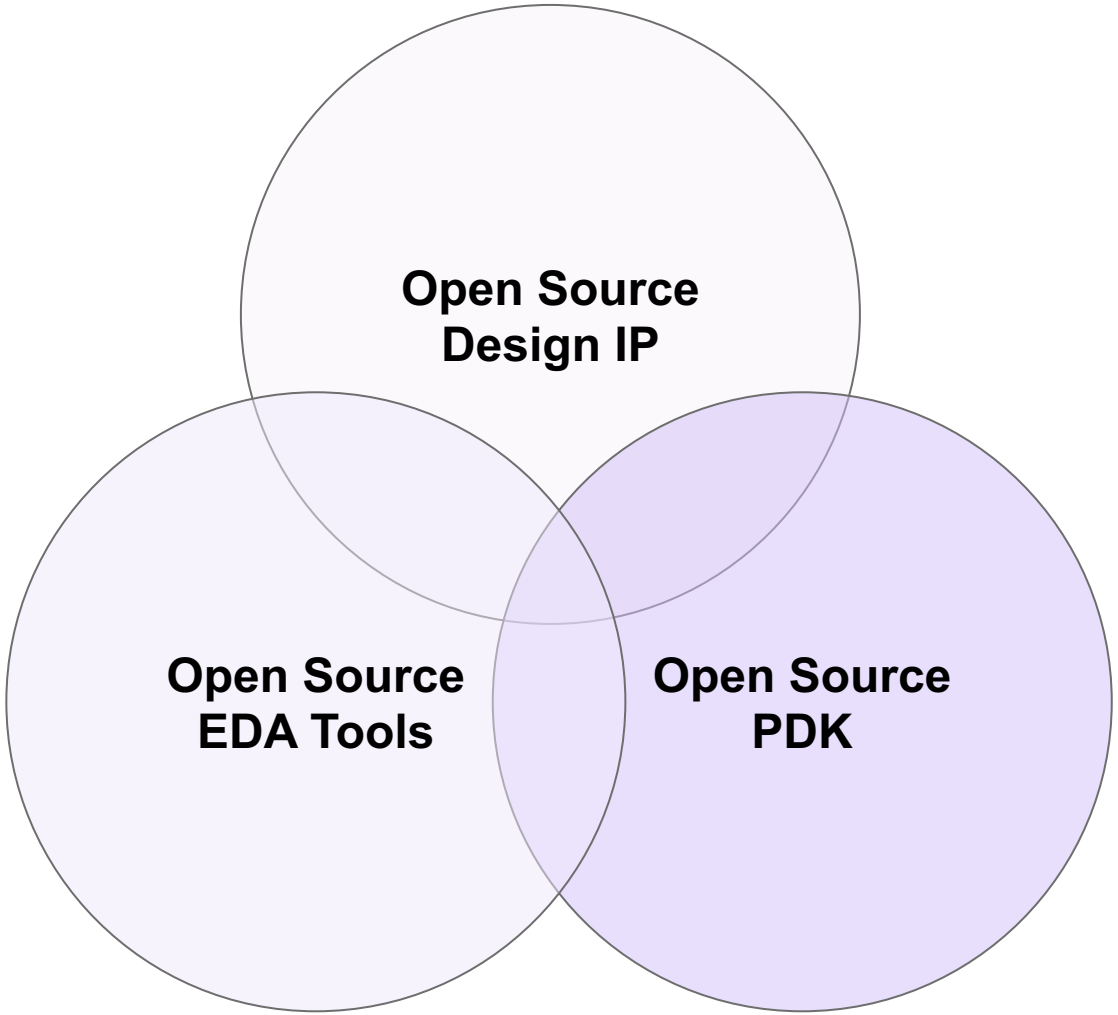
What I want to show you today

- **Goal:** Give you a rough overview of a decent starting point for building EDA tools on various levels of abstraction

- Built on existing materials and point you to good starting points
  - Mike Urbach, *Applying Circuit IR Compilers and Tools (CIRCT) to ML Applications*
  - Andrew Lenharth and Chris Lattner,
    CIRCT: Lifting hardware development out of the 20th century
  - John Demme, *Charting CIRCT*

HM

Open Source Silicon

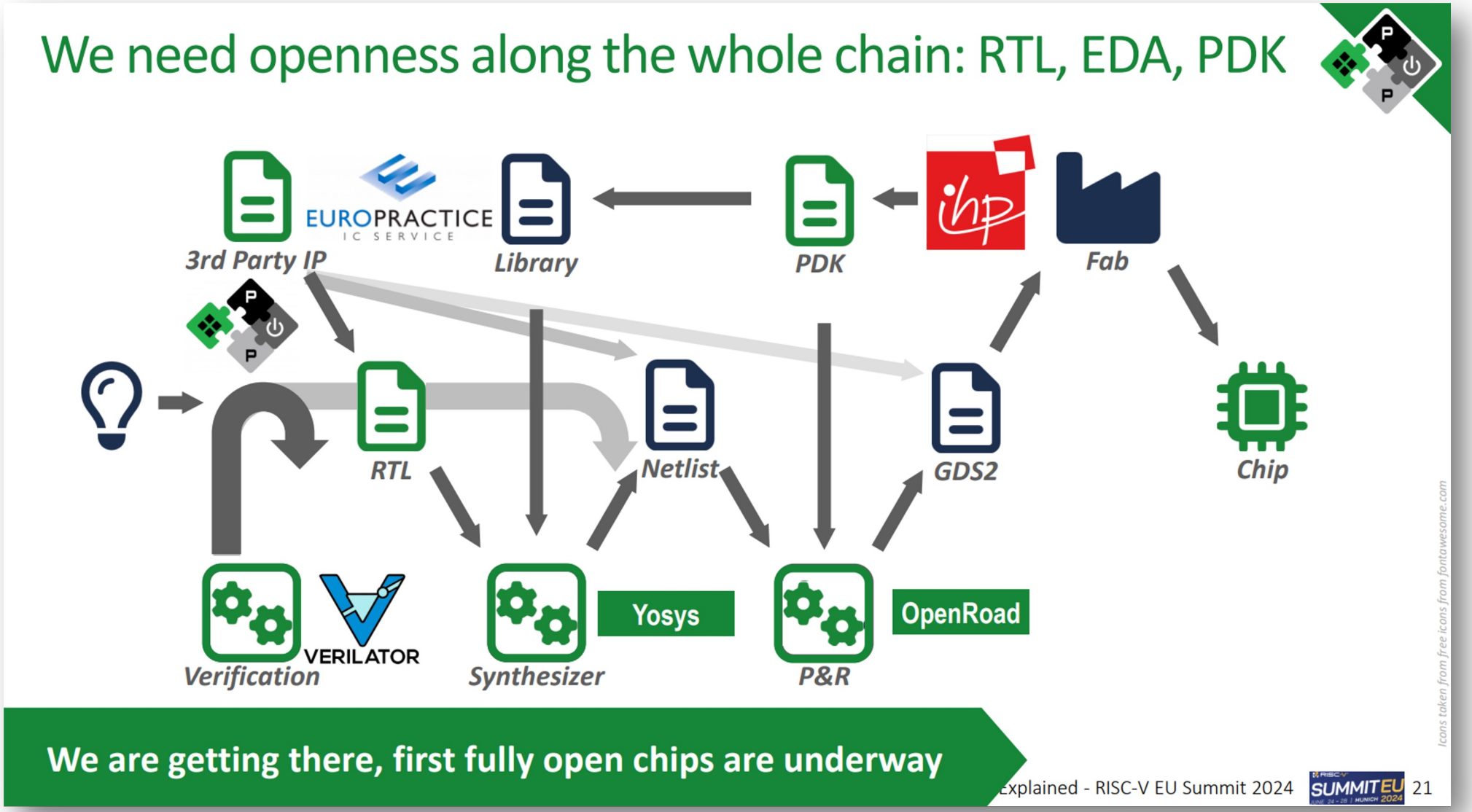


Open Source EDA Tools (digital view)

- Frontend: Logic Synthesis, Technology-independent
- Backend: Mapping, P&R, Technology-dependent
- But also:
  - Design-for-Test
  - Verification
  - High-level synthesis
  - Domain-specific languages
  - Integration and flows
  - Non-functional
  - Productivity tools
  - Your small tool that transforms a netlist
  - …

HM

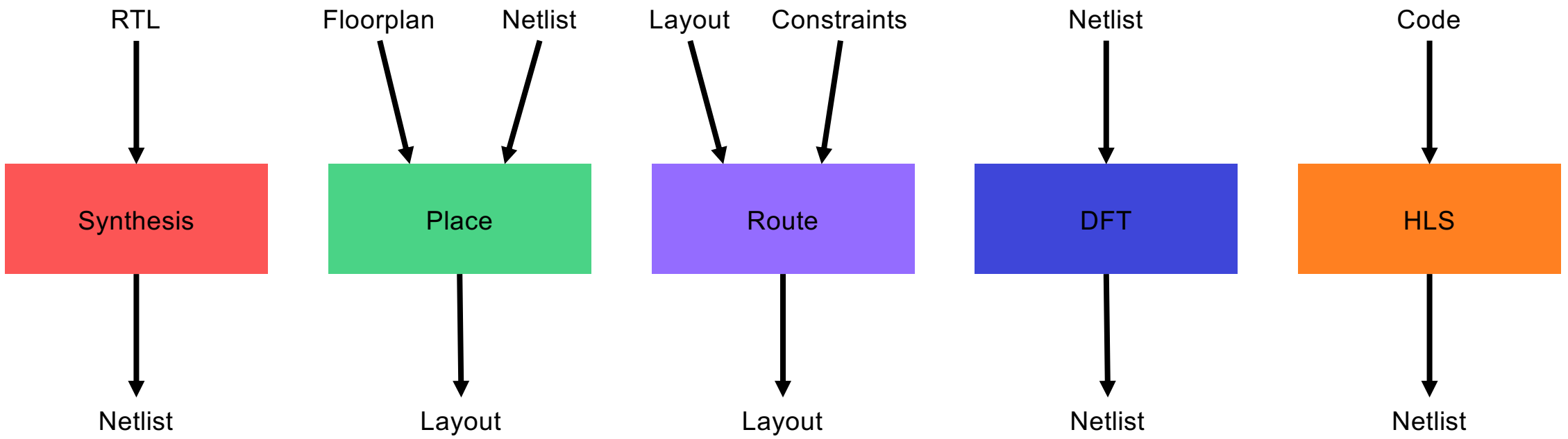
Abstract View on Design Flow



Frank K. Gürkaynak, RISC-V Summit Europe 2024

HM

Typical EDA Tool (illustrative)



How is that relevant for you?

- Surge of open source tools, but still gaps
- Your point tool that analyzes/inserts/patches something on the netlist
- New, more efficient design entries
- Productivity improvements/mixing with other design flows

HM

Many approaches to raising design abstraction

Research is producing new HW design models and abstraction approaches

CHISEL

LLHD

Magma

Dahlia

Calyx

SpinalHDL

XLS

migen

bluespec

Aetherling

Incorporating language + type system + compiler tech:

… often directly inspired by software

See also: ASPLOS LATTE'21 Workshop

HM

One typical approach: Verilog "Generators"

Challenges with Verilog:

"We need more metaprogramming (parameterization) to enable reuse of modules"

"Verilog has a weak type system, I want to catch bugs at compile time"

"I want to express complex parameterization in json or another file format"

"I want to be able to build tooling for my designs without having to parse Verilog"

Solution: Don't write Verilog, write a ***program*** to generate Verilog!

- Everything from a Perl script up to a generator *framework*
- Compare to TableGen, PerfectShuffle, Bison, ...

HM

How is that relevant for you?

- Surge of open source tools, but still gaps
- Your point tool that analyzes/inserts/patches something on the netlist
- New, more efficient design entries
- Productivity improvements/mixing with other design flows

→ Various needs to transform, anlayze, optimize and manipulate

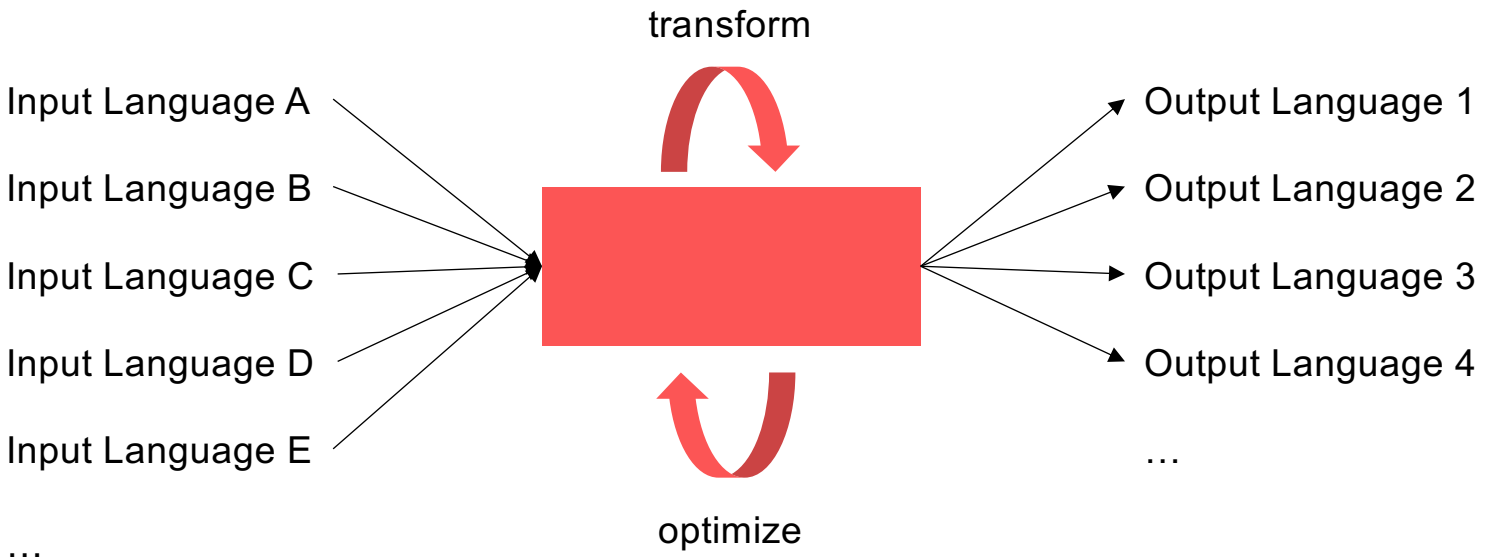
- One format to another
- One netlist to another netlist

HM

Isn’t this similar to a compiler?

Yes!

Rough concept of a compiler



Intermediate Representations

```
  %5 = alloca i8, align 1
  %6 = zext i1 %0 to i8
  store i8 %6, ptr %3, align 1
  %7 = zext i1 %1 to i8
  store i8 %7, ptr %4, align 1
  %8 = load i8, ptr %4, align 1
  %9 = trunc i8 %8 to i1
  br i1 %9, label %13, label %10

10:
  %11 = load i8, ptr %3, align 1
  %12 = trunc i8 %11 to i1
  br label %13
```

- Data structure to represent source in generic way
- Purposes:
  - Abstraction: Still independent from output format
  - Modularity: Divide-and-conquer
  - Portability: Only add input language, output language, or transforms
  - Optimizations: Generic implementation of output-independent optimizations
- Example in software: static single assignment (SSA) form (e.g. LLVM)
- IR for hardware is a major challenge
  - Verilog is not a great IR: A lot of intent gets lost

CIRCT: Circuit IR Compilers and Tools

CIRCT: Compiler Infrastructure for the future of EDA

- **C**ircuit **I**ntermediate **R**epresentation **C**ompilers and **T**ools
- Built using MLIR
- LLVM incubator project
- Composable toolchain for different aspects of electronic design automation (EDA) process
- Common platform with clean interfaces
- Tools for designing accelerators are relevant for programming accelerators

https://circt.llvm.org

HM

MLIR

- Grown as LLVM project
- Generalization framework

What is MLIR?

- Framework to build a compiler IR: define your type system, operations, etc.
- Toolbox covering your compiler infrastructure needs
  - Diagnostics, pass infrastructure, multi-threading, testing tools, etc.
- Batteries-included:
  - Various code-generation / lowering strategies
  - Accelerator support (GPUs)
- Allow different levels of abstraction to freely co-exist
  - Abstractions can better target specific areas with less high-level information lost
  - Progressive lowering simplifies and enhances transformation pipelines
  - No arbitrary boundary of abstraction, e.g. host and device code in the same IR at the same time

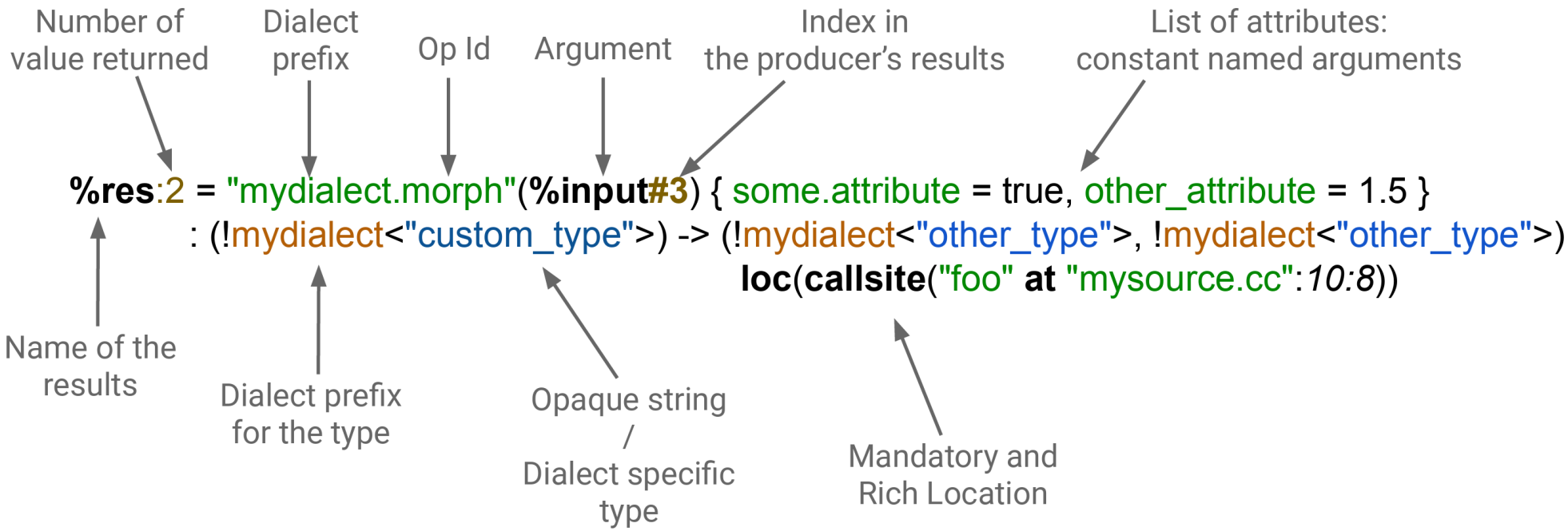
Google

HM

Fundamentals of MLIR

Operations, Not Instructions

- No predefined set of instructions
- Operations are like “opaque functions” to MLIR



Google

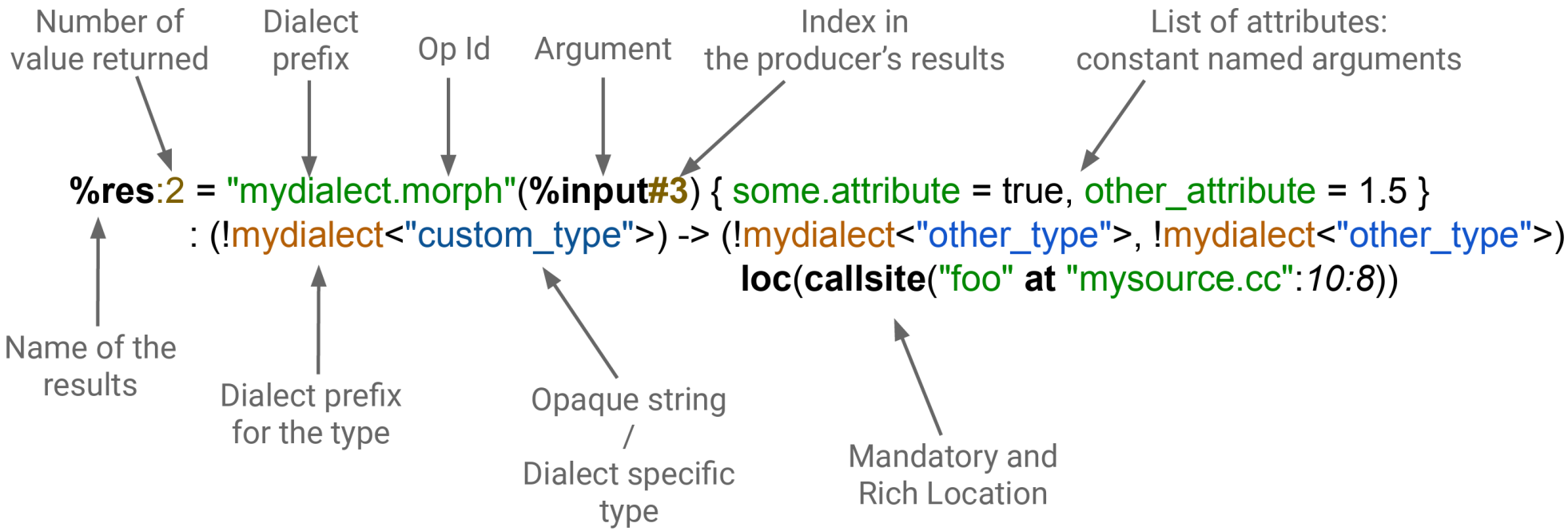
https://mlir.llvm.org/docs/LangRef/#operations
https://github.com/llvm/llvm-project/blob/master/mlir/include/mlir/IR/Operation.h#L27

HM

Fundamentals of MLIR

Operations, Not Instructions

- No predefined set of instructions
- Operations are like “opaque functions” to MLIR



Google

https://mlir.llvm.org/docs/LangRef/#operations
https://github.com/llvm/llvm-project/blob/master/mlir/include/mlir/IR/Operation.h#L27

HM

Fundamentals of MLIR

Dialects: Defining Rules and Semantics for the IR

A MLIR dialect is a logical grouping including:

- A prefix (“namespace” reservation)
- A list of custom types, each its C++ class.
- A list of operations, each its name and C++ class implementation:
  - Verifier for operation invariants (e.g. *toy.print* must have a single operand)
  - Semantics (has-no-side-effects, constant-folding, CSE-allowed, ….)
- Passes: analysis, transformations, and dialect conversions.
- Possibly custom parser and assembly printer

Google

https://mlir.llvm.org/docs/LangRef/#dialects
https://github.com/llvm/llvm-project/blob/master/mlir/include/mlir/IR/Dialect.h#L37
https://mlir.llvm.org/docs/Tutorials/CreatingADialect/

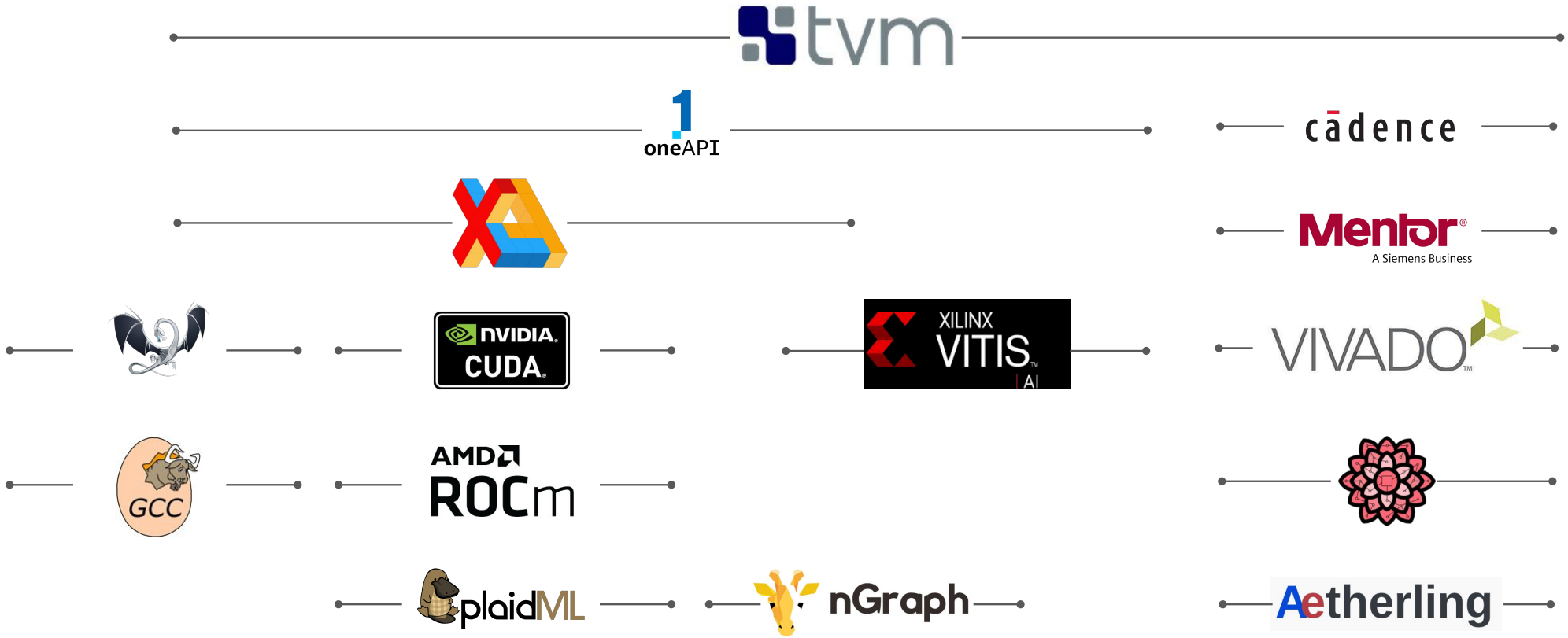
HM

How MLIR and CIRCT Fit In

Fixed processor

Custom processor

CPU, etc. | GPU, etc. | TPU, NPU, etc. | FPGA, CPLD, etc. | ASIC



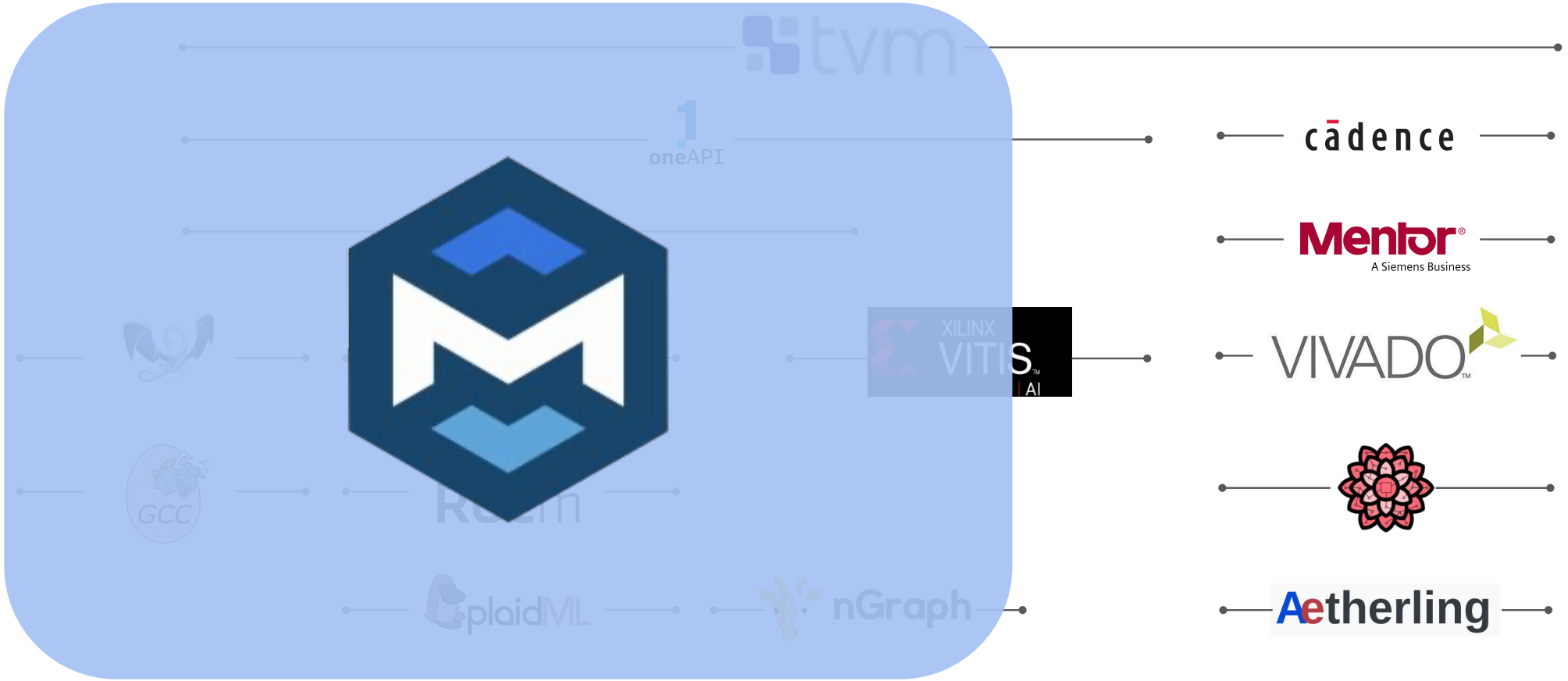
HM

How MLIR and CIRCT Fit In

Fixed processor

Custom processor

CPU, etc. | GPU, etc. | TPU, NPU, etc. | FPGA, CPLD, etc. | ASIC



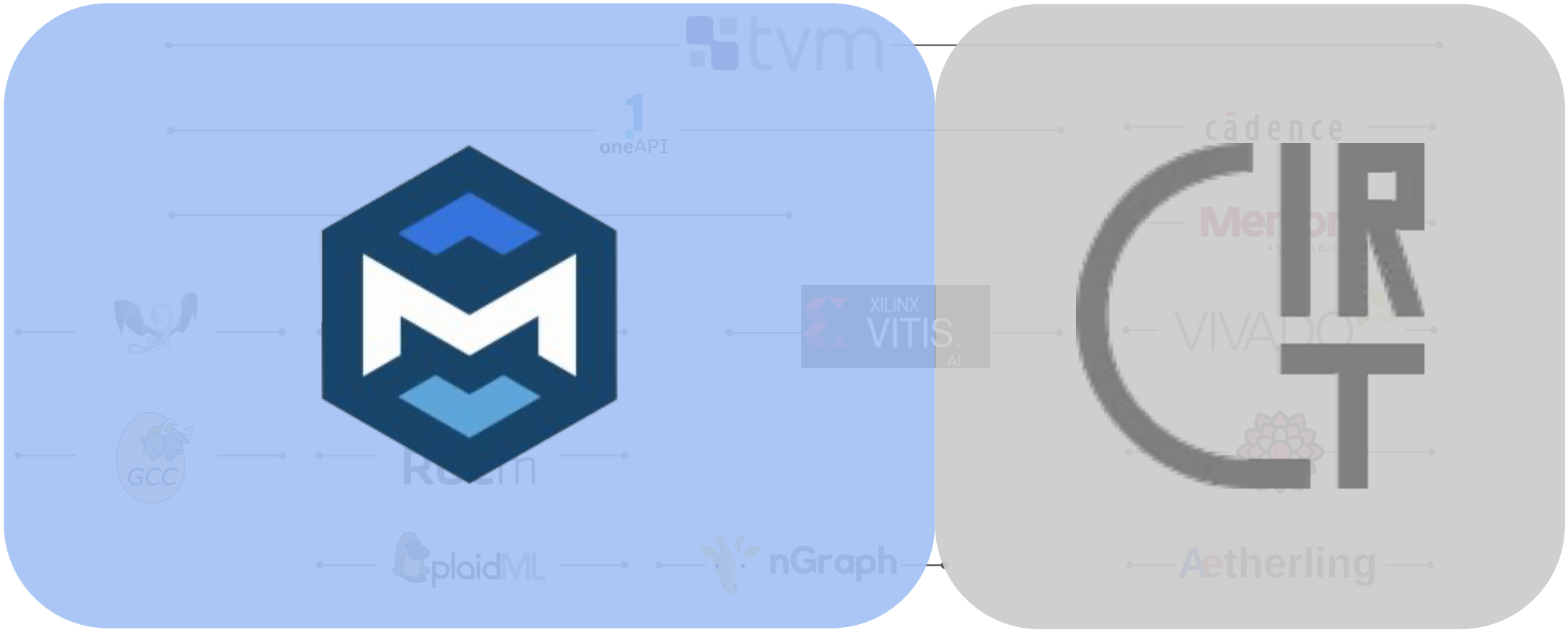
HM

How MLIR and CIRCT Fit In

Fixed processor

Custom processor

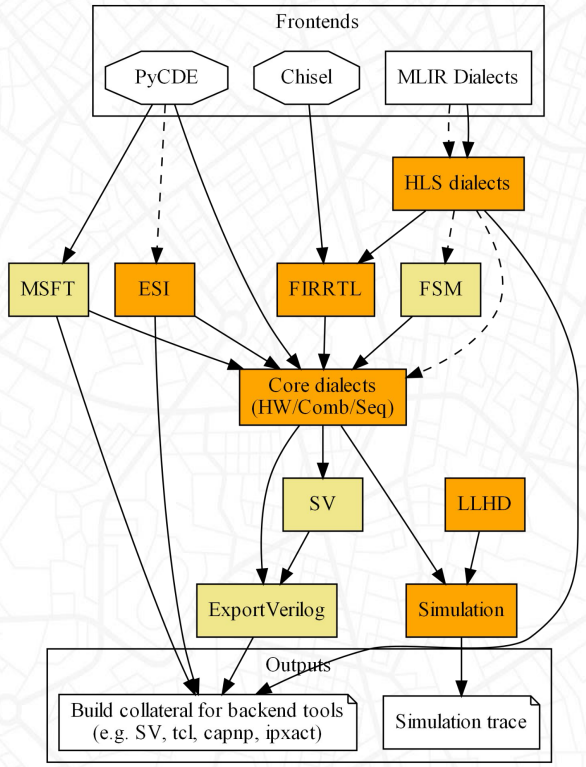
CPU, etc. | GPU, etc. | TPU, NPU, etc. | FPGA, CPLD, etc. | ASIC



HM

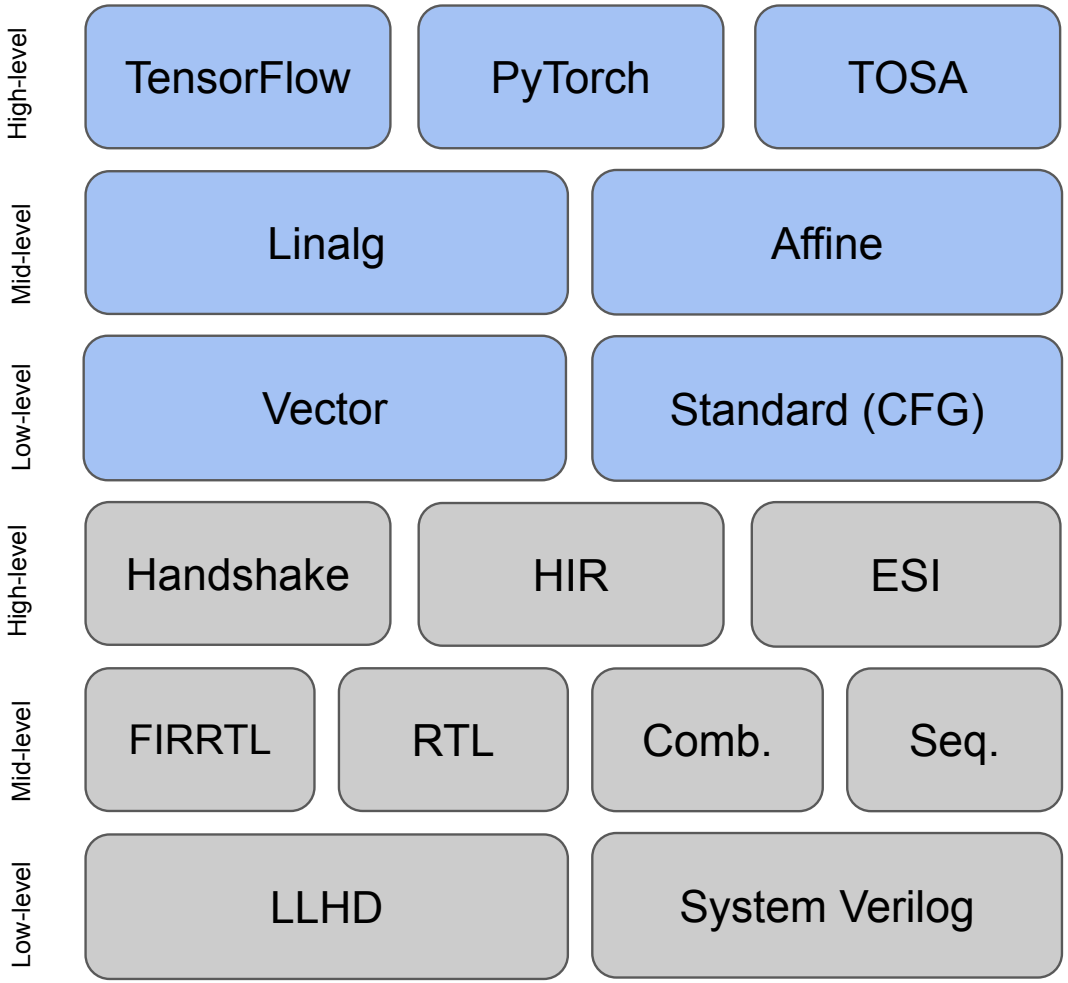
CIRCT: Dialects, Transforms, Passes

Fodor's list of must-see places (Rick Steves mostly agrees)



[{ Overview photo of CIRCT taken from Mars }]

Compiler Stack



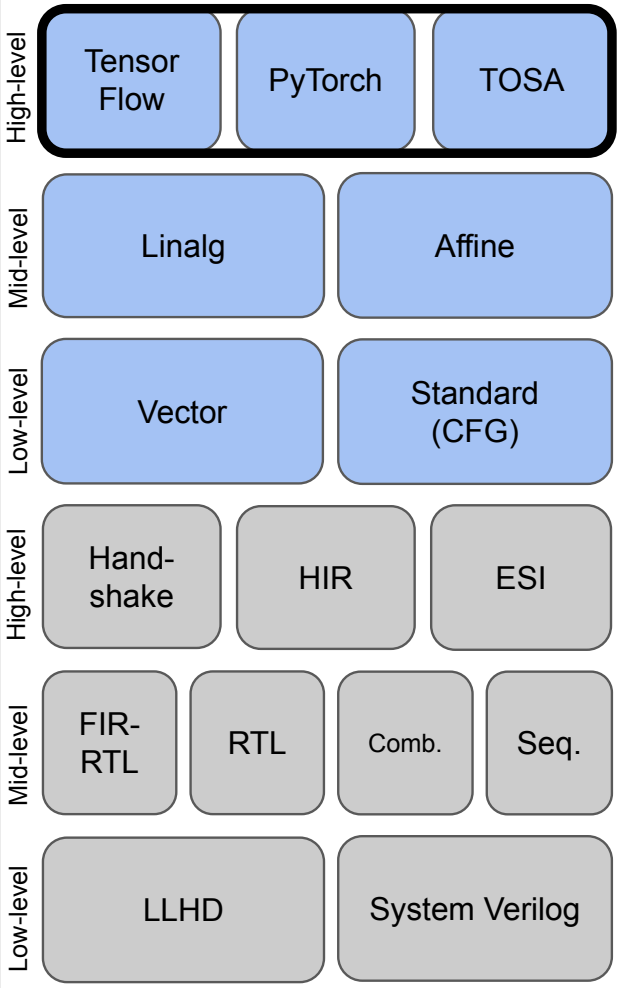
MLIR dialects

CIRCT dialects

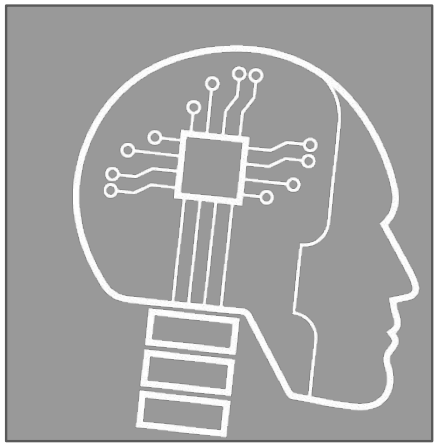
- Start from ML frameworks
- Use MLIR for as much as possible
- Get down to common MLIR layers
- Translate those into CIRCT
- Lower to CIRCT core dialects
- Translate those to exit dialects

HM

High-level dialects: TensorFlow, PyTorch, TOSA

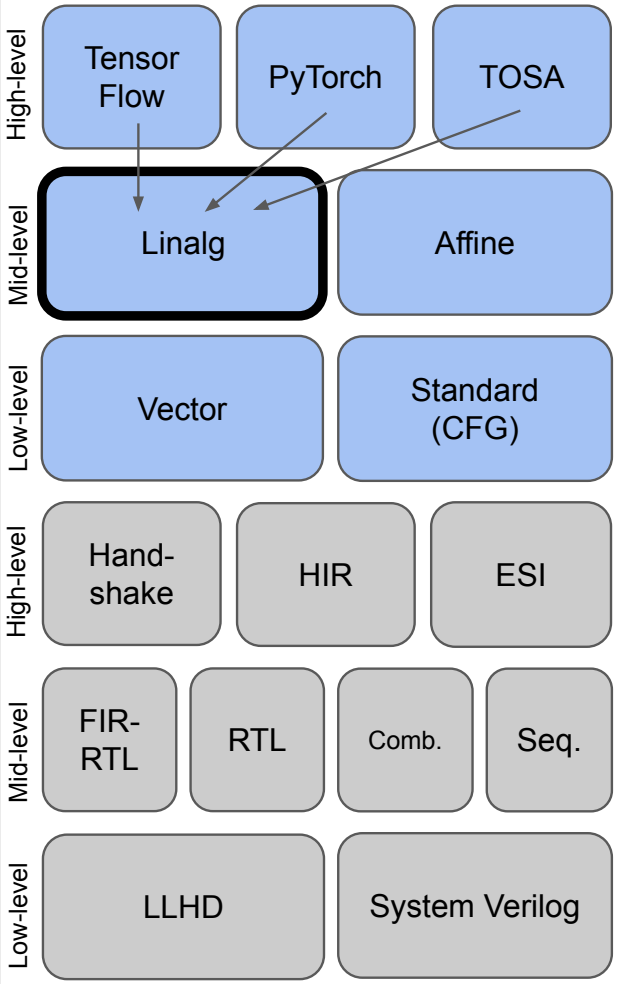



```
func @matmul(%lhs: tensor<5x10xf32>, %rhs: tensor<10x10xf32>) -> (tensor<5x10xf32>) {
  %0 = "tf.BatchMatMulV2"(%lhs, %rhs) : (tensor<5x10xf32>, tensor<10x10xf32>) ->
                                        tensor<5x10xf32>
  return %0 : tensor<5x10xf32>
}
```

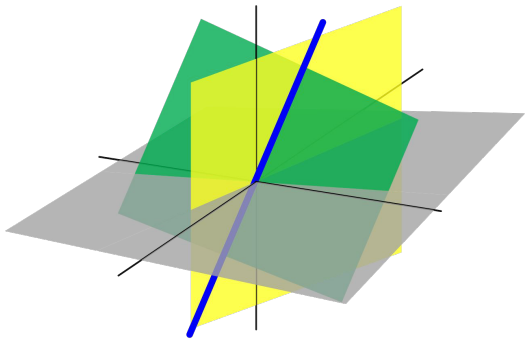


HM

Mid-level dialects: Linalg




```
func @matmul(%A: tensor<5x10xf32>, %B: tensor<10x10xf32>,
             %C: tensor<10x10xf32>) -> tensor<10x10xf32> {
  %0 = linalg.matmul ins(%A, %B: tensor<5x10xf32>, tensor<10x10xf32>)
                     outs(%C: tensor<10x10xf32>) -> tensor<10x10xf32>
  return %0: tensor<10x10xf32>
}
```

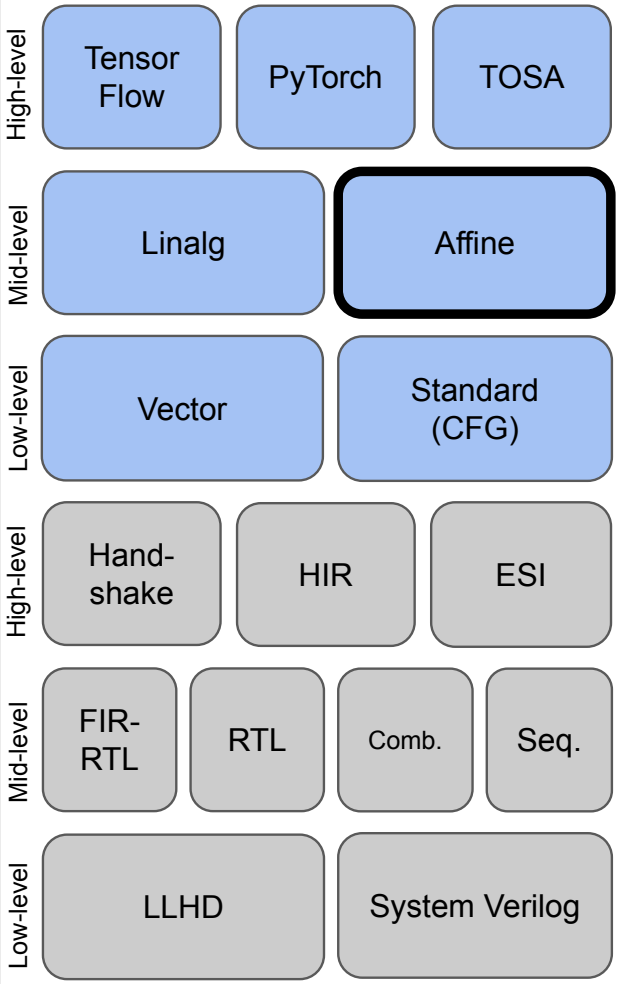


- Builds on lots of prior art
- A handful of generic operations
- Named operations (like above) that map to generic operations

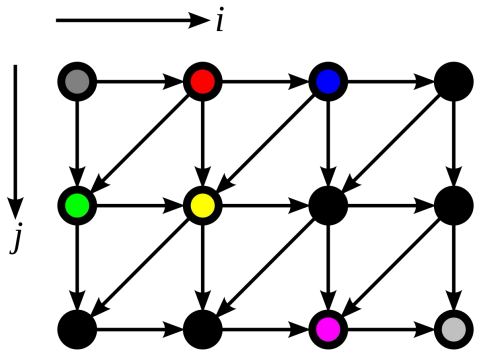
https://mlir.llvm.org/docs/Dialects/Linalg/

HM

Mid-level dialects: Affine



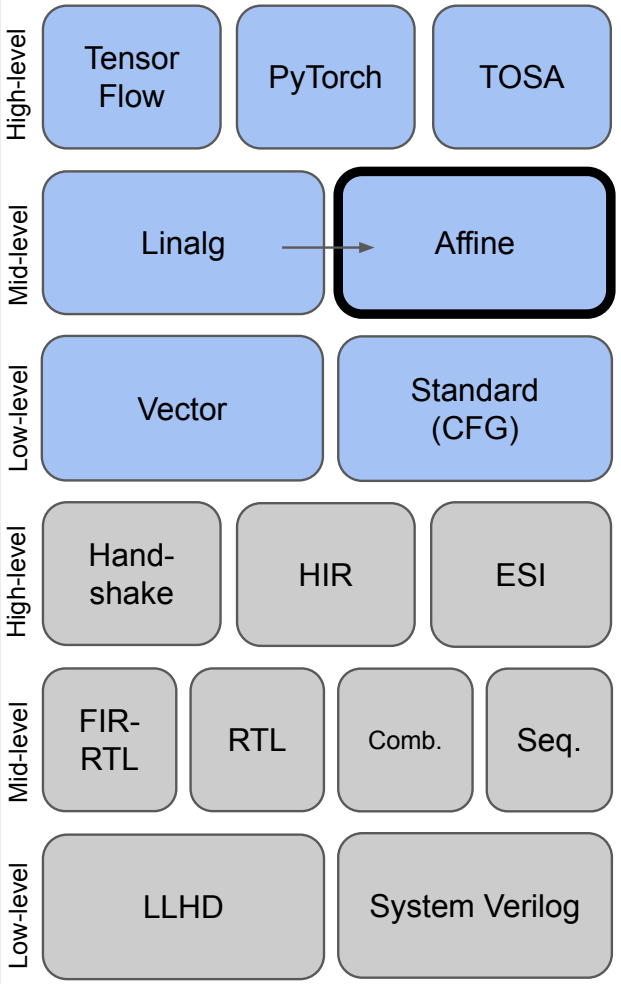

```
func @matmul(%A: memref<5x10xf32>, %B: memref<10x10xf32>,
             %C: memref<10x10xf32>) -> memref<10x10xf32> {
  affine.for %arg3 = 0 to 5 {
    affine.for %arg4 = 0 to 10 {
      affine.for %arg5 = 0 to 10 { ... }
    }
  }
  return %0 : memref<10x10xf32>
}
```



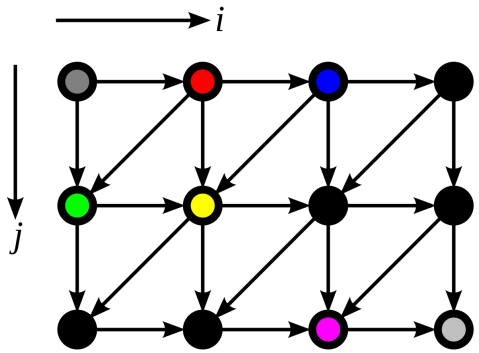
- Polyhedral compilation toolbox
- Affine expressions, maps, and integer sets
- Affine loops and transformations

https://mlir.llvm.org/docs/Dialects/Affine/

Mid-level dialects: Affine




```
func @matmul(%A: memref<5x10xf32>, %B: memref<10x10xf32>,
             %C: memref<10x10xf32>) -> memref<10x10xf32> {
  affine.for %arg3 = 0 to 5 {
    affine.for %arg4 = 0 to 10 {
      affine.for %arg5 = 0 to 10 { ... }
    }
  }
  return %0 : memref<10x10xf32>
}
```

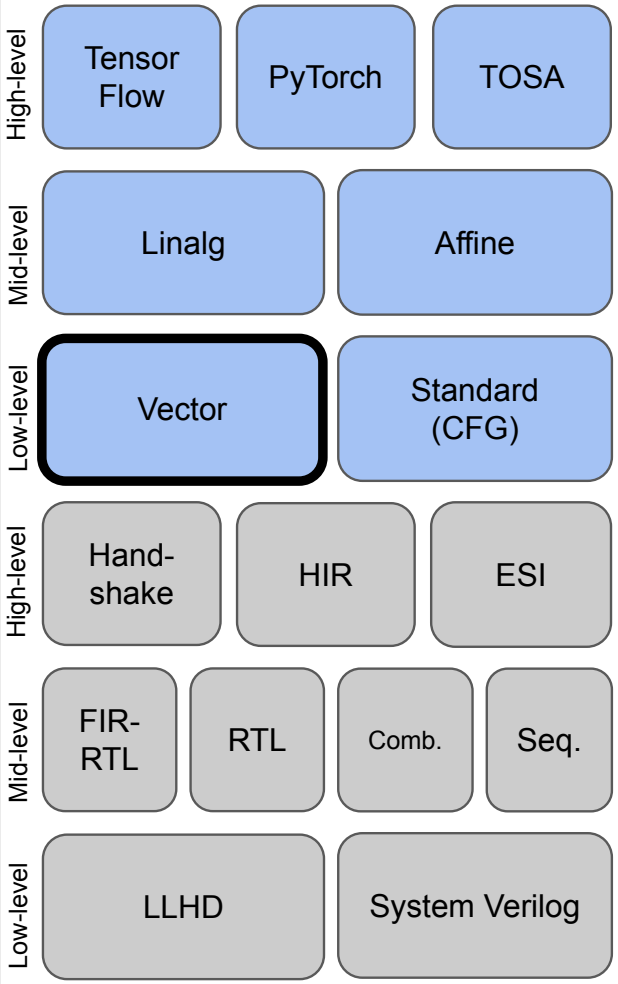


- Polyhedral compilation toolbox
- Affine expressions, maps, and integer sets
- Affine loops and transformations

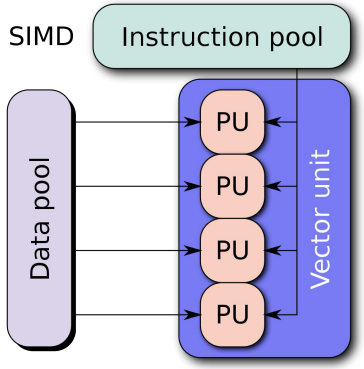
https://mlir.llvm.org/docs/Dialects/Affine/

HM

Low-level dialects: Vector




```
func @transpose(%arg0: vector<3x7xf32>) -> vector<7x3xf32> {
  %0 = vector.transpose %arg0, [1, 0] : vector<3x7xf32> to vector<7x3xf32>
  return %0 : vector<7x3xf32>
}
```

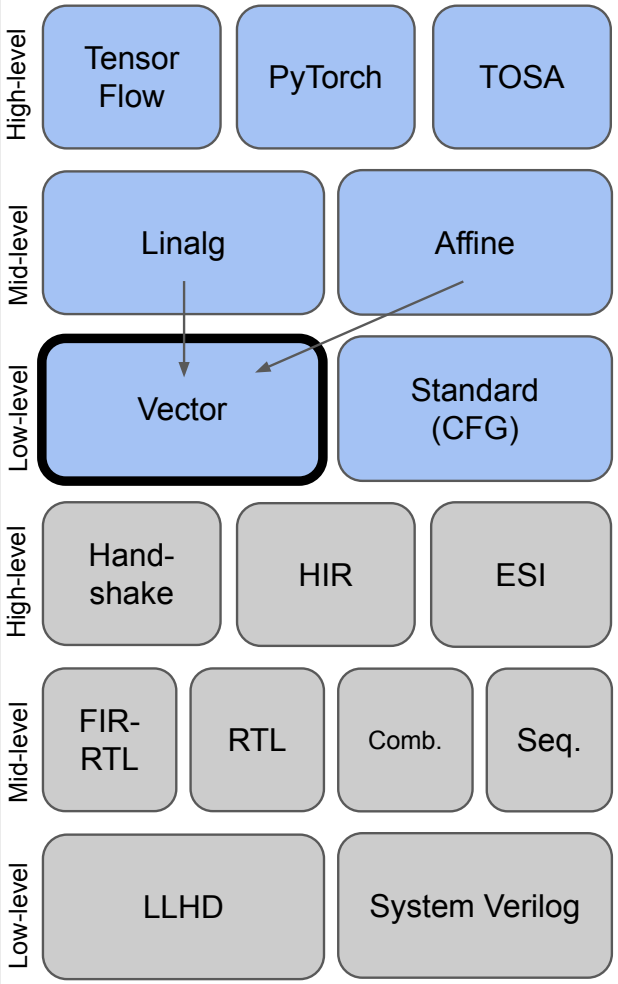


- Virtual n-D vectors and transformations on them
- Hardware vectors that match a specific processor's ISA

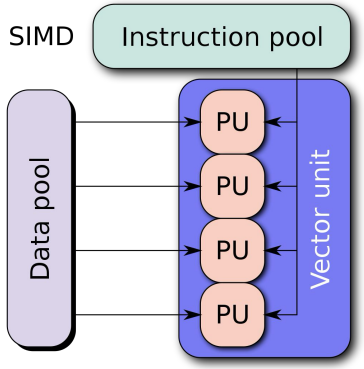
https://mlir.llvm.org/docs/Dialects/Vector/

HM

Low-level dialects: Vector



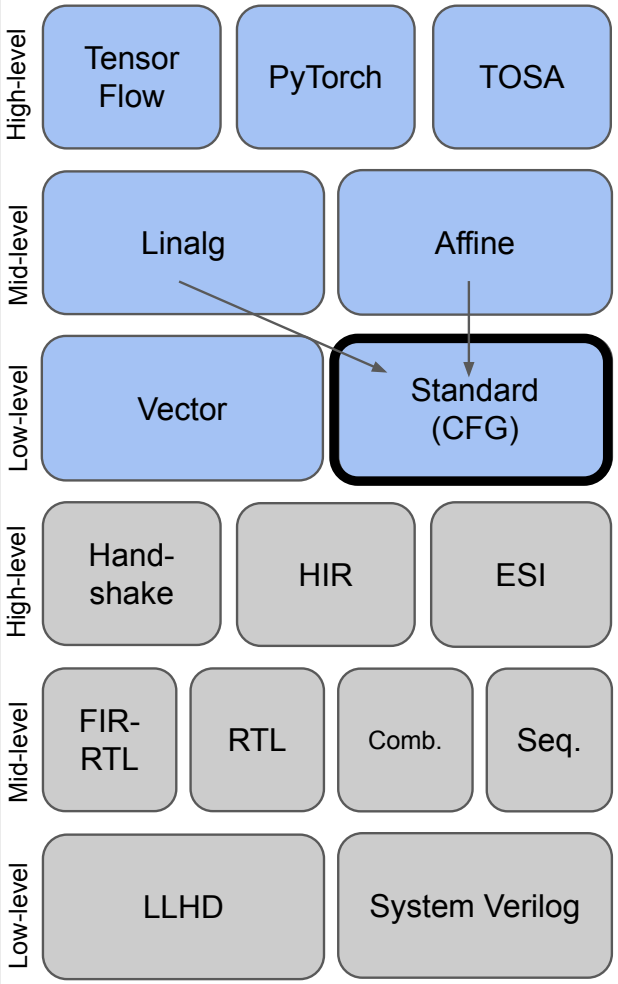

```
func @transpose(%arg0: vector<3x7xf32>) -> vector<7x3xf32> {
  %0 = vector.transpose %arg0, [1, 0] : vector<3x7xf32> to vector<7x3xf32>
  return %0 : vector<7x3xf32>
}
```



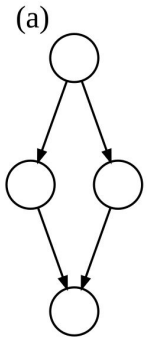
- Virtual n-D vectors and transformations on them
- Hardware vectors that match a specific processor's ISA

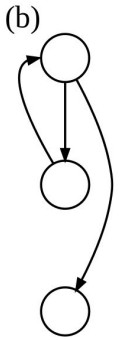
https://mlir.llvm.org/docs/Dialects/Vector/

Low-level dialects: Standard control-flow graph




```
func @simple_loop() {
^bb0:
  br ^bb1
^bb1:   // pred: ^bb0
  %c1 = constant 1 : index
  %c42 = constant 42 : index
  br ^bb2(%c1 : index)
^bb2(%0: index):        // 2 preds: ^bb1, ^bb3
  %1 = cmpi slt, %0, %c42 : index
  cond_br %1, ^bb3, ^bb4
```



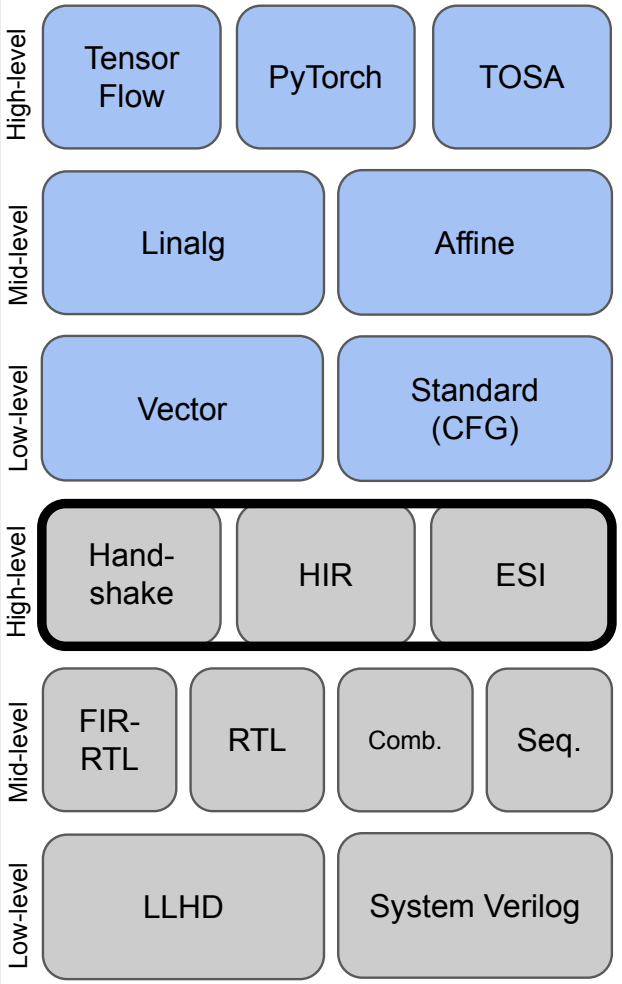


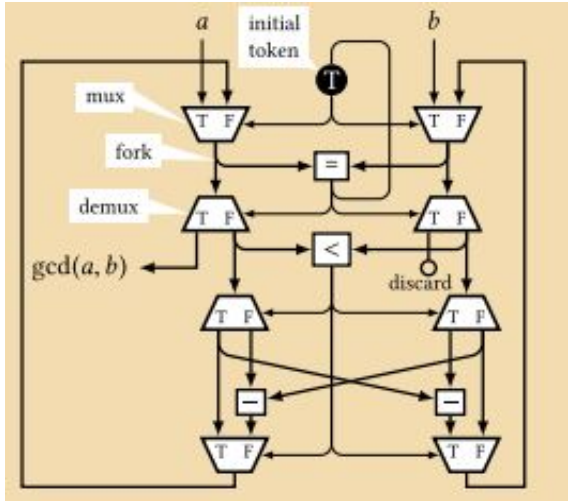
- MLIR's standard dialect
- Basic blocks and branches
- Evolution of LLVM IR

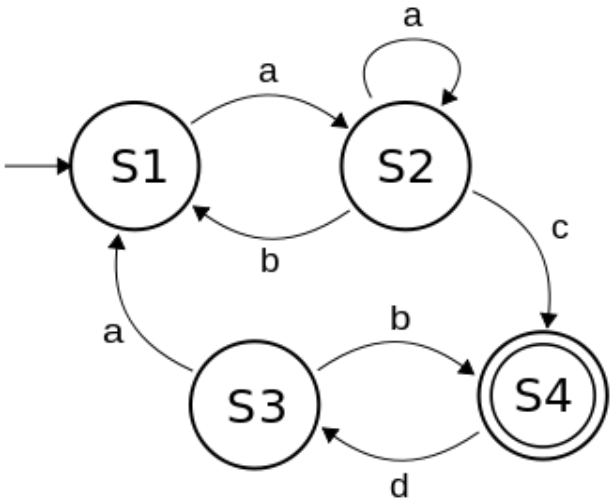
https://mlir.llvm.org/docs/Dialects/Standard/

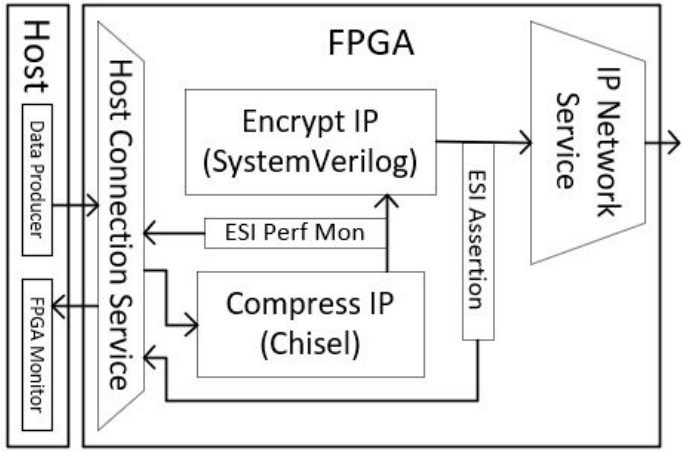
HM

High-level dialects

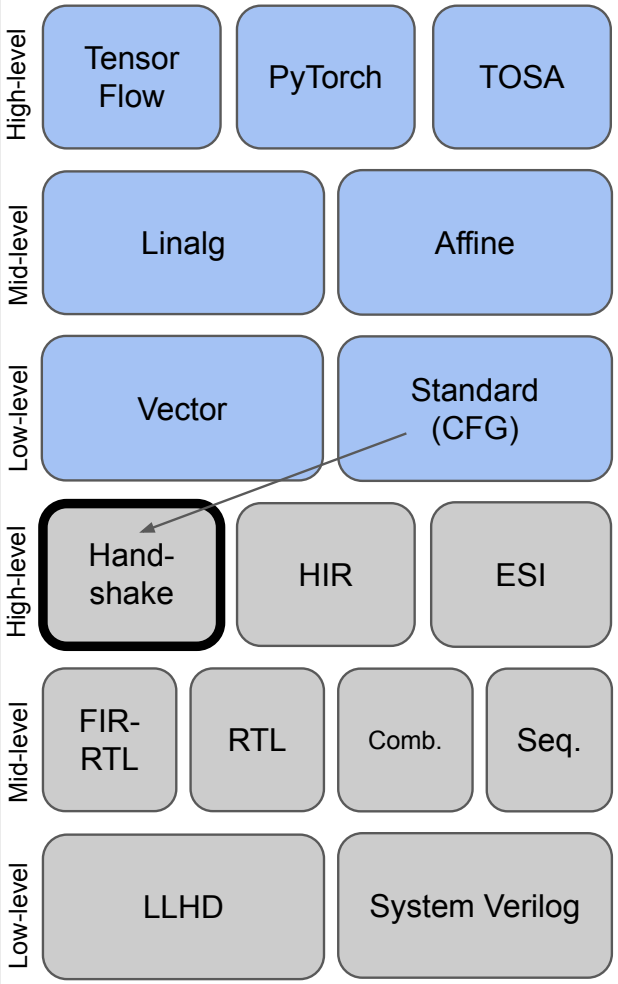




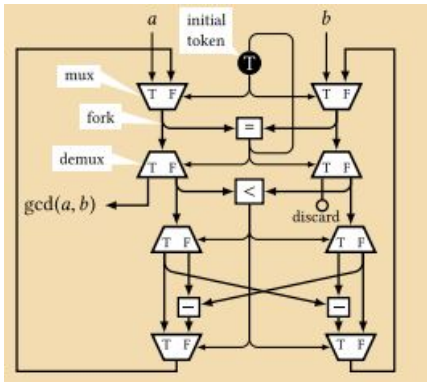




High-level dialects: Handshake



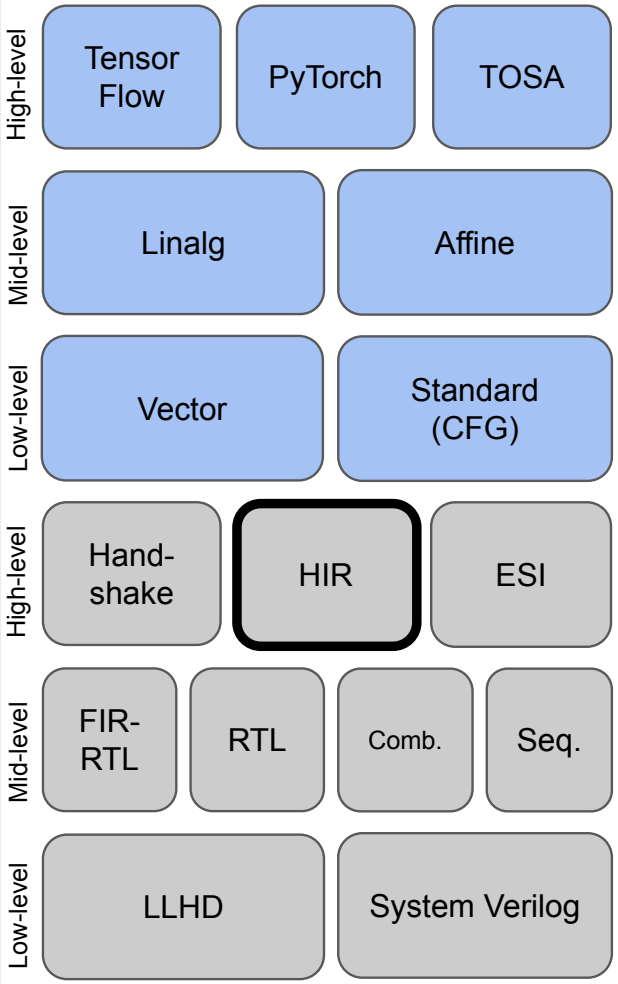

```
handshake.func @fork(%arg0: none, %arg1: none) -> (none, none, none) {
  %0:2 = "handshake.fork"(%arg0) {control = true} : (none) -> (none, none)
  %1 = "handshake.join"(%0#0, %0#1) {control = true}: (none, none) -> none
  handshake.return %1, %arg1 : none, none, none
}
```



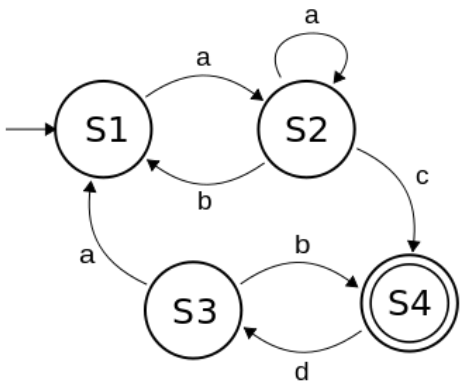
- Composable set of control flow operators like fork and join
- Rooted in Kahn Process Networks and Petri Nets
- Efficient hardware implementation

https://circt.llvm.org/docs/Dialects/Handshake/

High-level dialects: HIR



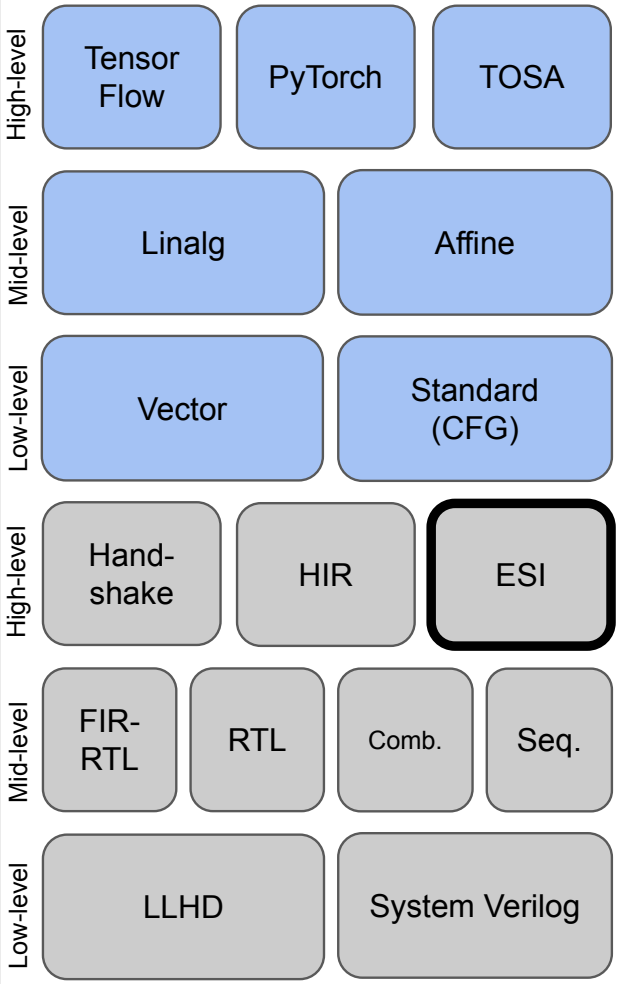

```
hir.func @mac at %t (%a :i32, %b :i32, %c :i32) -> (i32 delay 3) {
  %1 = hir.constant 1
  %2 = hir.constant 2
  %m = hir.call @mult_3stage (%a,%b) at %t : (i32, i32) -> (i32 delay 3)
  %c2= hir.delay %c by %2 at %t : i32 -> i32
  %res = hir.add (%m,%c2) : (i32, i32) -> (i32)
  %res1 = hir.delay %res by %1 at %t offset %2 : i32 -> i32
  hir.return (%res1) : (i32)
}
```



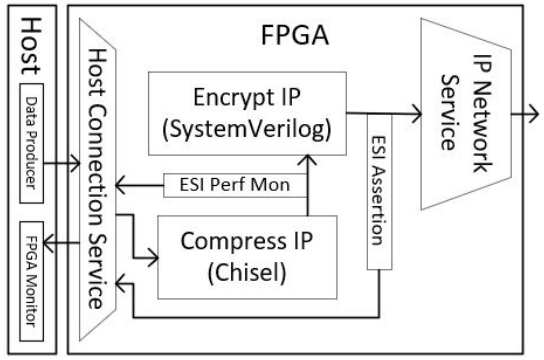
- Zero cost abstractions
- Represents finite state machines in a high-level way
- Supports banked memories

[HIR: An MLIR-based IR for Hardware Accelerator Description]

High-level dialects: Elastic Silicon Interconnect (ESI)



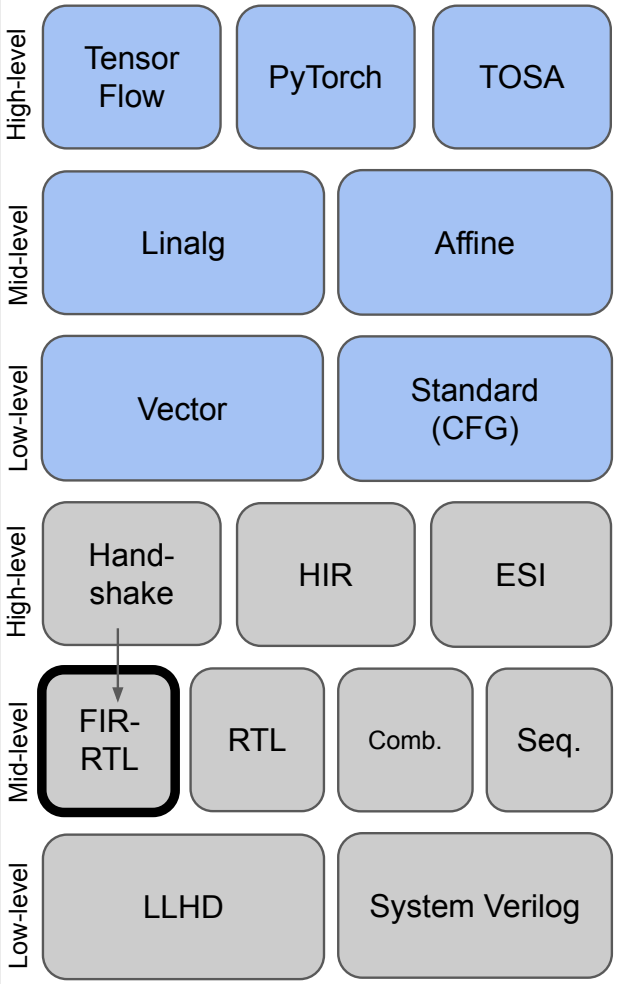

```
rtl.module @esi(%clock: i1, %reset: i1) {
  %esiChannel, %0 = rtl.instance "sender" @Sender (%clock) : (i1) -> (!esi.channel<i4>, i8)
  %bufferedChannel = esi.buffer %clock, %reset, %esiChannel { stages = 4 } : i4
  rtl.instance "receiver" @Reciever (%bufferedChannel, %clock) : (!esi.channel<i4>, i1) -> ()
}
```



- Type system for channels in hardware
- Connect on-chip and off-chip components
- Generates elastic connector circuits
- Supports cosimulation

https://circt.llvm.org/docs/Dialects/ESI/

Mid-level dialects: FIRRTL




```
firrtl.module @bundle0(%a : !firrtl.bundle<f1: uint<1>, f2: flip<sint<1>>>,
                       %b : !firrtl.bundle<f1: flip<uint<1>>, f2: sint<1>>) {
  firrtl.connect %b, %a : !firrtl.bundle<f1: flip<uint<1>>, f2: sint<1>>,
                          !firrtl.bundle<f1: uint<1>, f2: flip<sint<1>>>
}
```

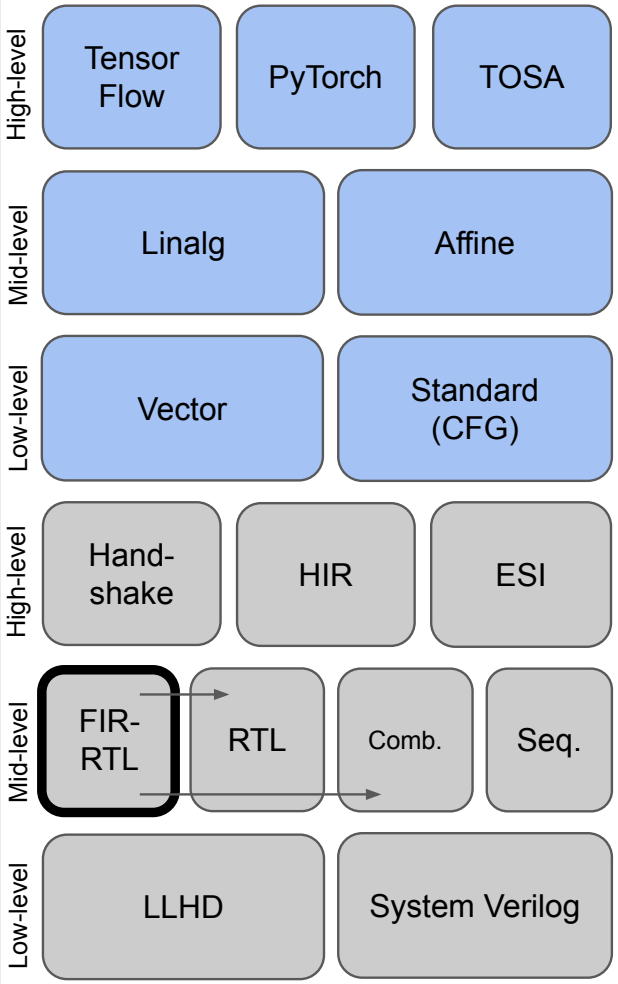
FIRRTL

- Core IR from Chisel compiler
- Abstractions that are below Chisel but above System Verilog
- Lots of effort to improve on the design in CIRCT's core dialects

https://circt.llvm.org/docs/Dialects/FIRRTL/

HM

Mid-level dialects: FIRRTL




```
firrtl.module @bundle0(%a : !firrtl.bundle<f1: uint<1>, f2: flip<sint<1>>>,
                       %b : !firrtl.bundle<f1: flip<uint<1>>, f2: sint<1>>) {
  firrtl.connect %b, %a : !firrtl.bundle<f1: flip<uint<1>>, f2: sint<1>>,
                          !firrtl.bundle<f1: uint<1>, f2: flip<sint<1>>>
}
```

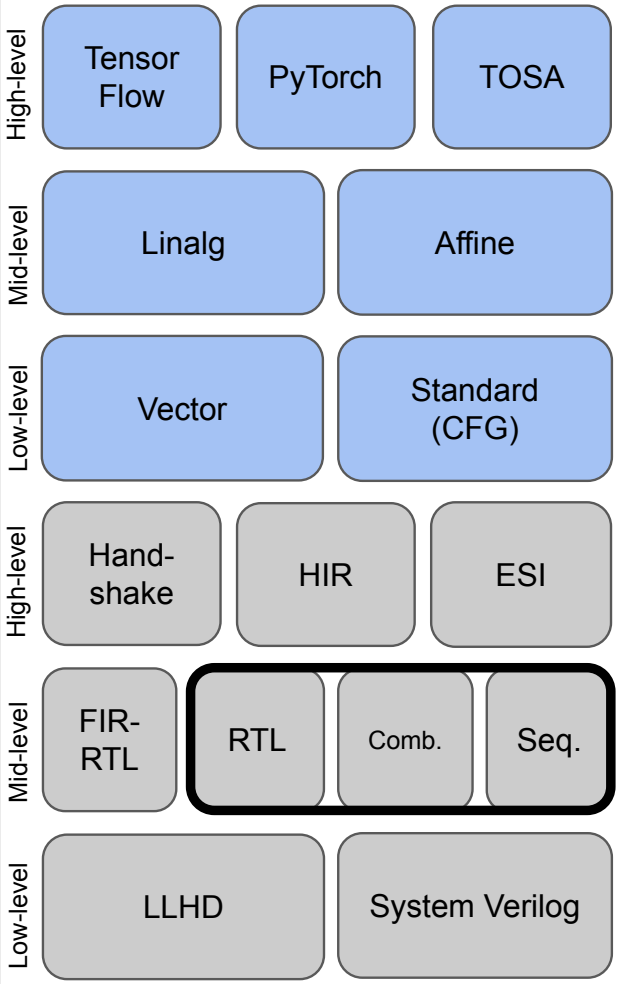
FIRRTL

- Core IR from Chisel compiler
- Abstractions that are below Chisel but above System Verilog
- Lots of effort to improve on the design in CIRCT's core dialects

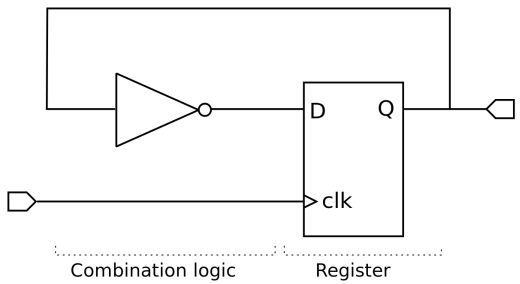
https://circt.llvm.org/docs/Dialects/FIRRTL/

HM

Mid-level dialects: RTL, Combinational, Sequential




```
rtl.module @test1(%a: i12) -> (i32){
  %b = comb.add %a, %a : i12
  %c = comb.mul %a, %b : i12
  %d = comb.sext %c : (i12) -> i32
  rtl.output %d : i32
}
```

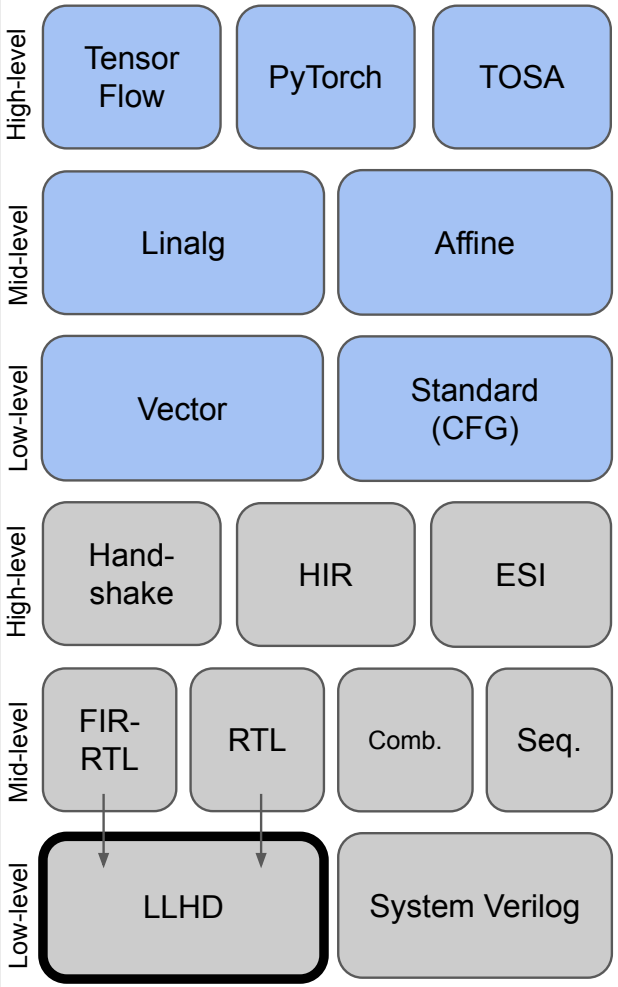


- Structure (modules and ports)
- Combinational logic (and, or, xor)
- Sequential logic (registers)
- Work in progress

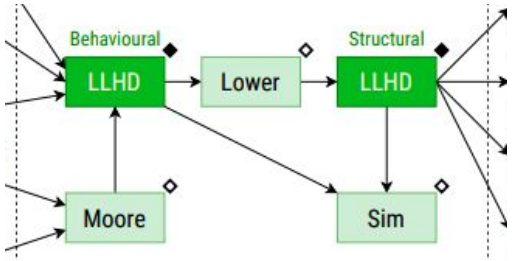
https://circt.llvm.org/docs/Dialects/RTL/

HM

Low-level dialects: LLHD



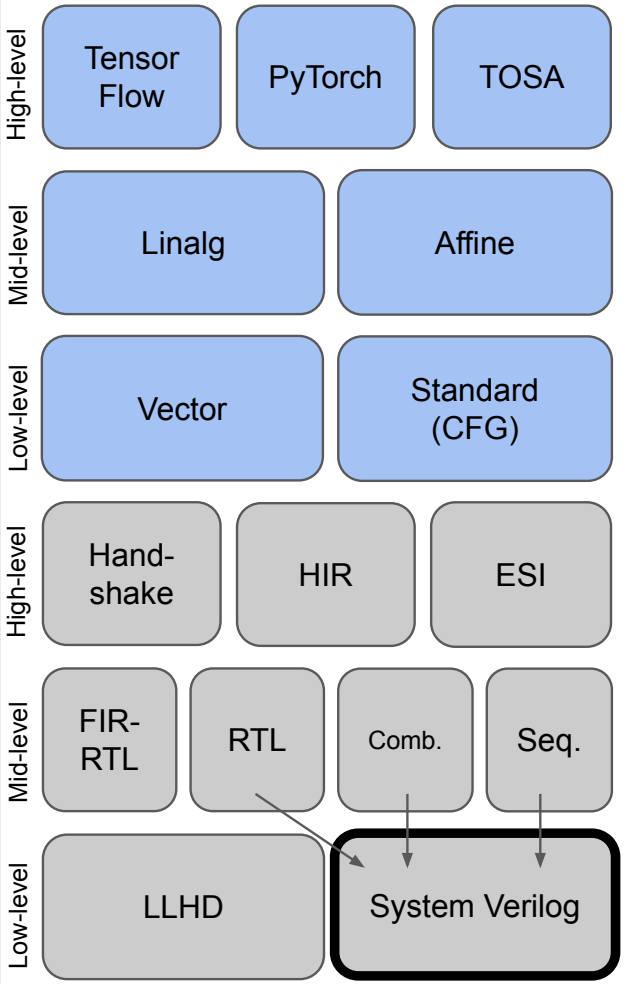

```
llhd.entity @top (%arg0 : !llhd.sig<i1>) -> (%arg1 : !llhd.sig<i1>) {
  %t = #llhd.time<1ns, 2d, 3e> : !llhd.time
  %0 = llhd.prb %arg0 : !llhd.sig<i1>
  llhd.drv %arg1, %0 after %t : !llhd.sig<i1>
}
```



- Low-level, even for hardware
- Supports 9-valued logic
- Encodes time in the type system
- Includes MLIR-based simulator

https://circt.llvm.org/docs/Dialects/LLHD/

Low-level dialects: System Verilog




```
rtl.module @sytemverilog(%clock: i1, %reset: i1) {
  %c0 = rtl.constant 0 : i32
  %c42 = rtl.constant 42 : i32
  %reg = sv.reg : !rtl.inout<i32>
  sv.alwaysff(posedge %clock) {
   sv.passign %reg, %c42 : i32
  } (asyncreset : posedge %reset) {
   sv.passign %reg, %c0 : i32
  }
}
```

SystemVerilog

- Represents System Verilog syntax
- Designed for export, not transformation
- Readability and other syntactical improvements happen here
- Guides towards compatibility, but offers escape hatch

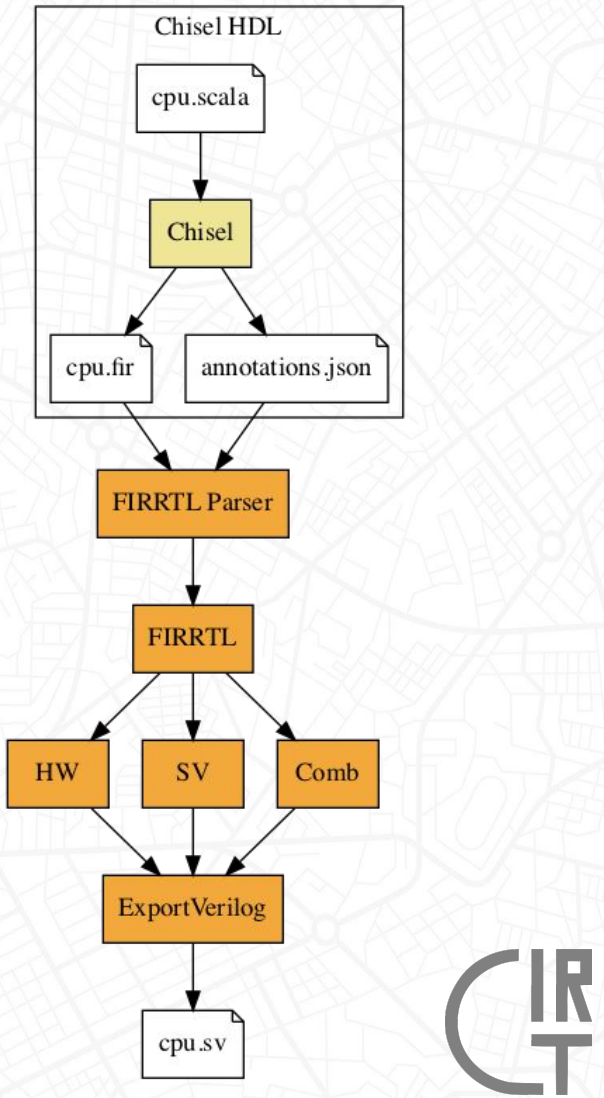
https://circt.llvm.org/docs/Dialects/SV/

HM

Example: CHISEL Ecosystem

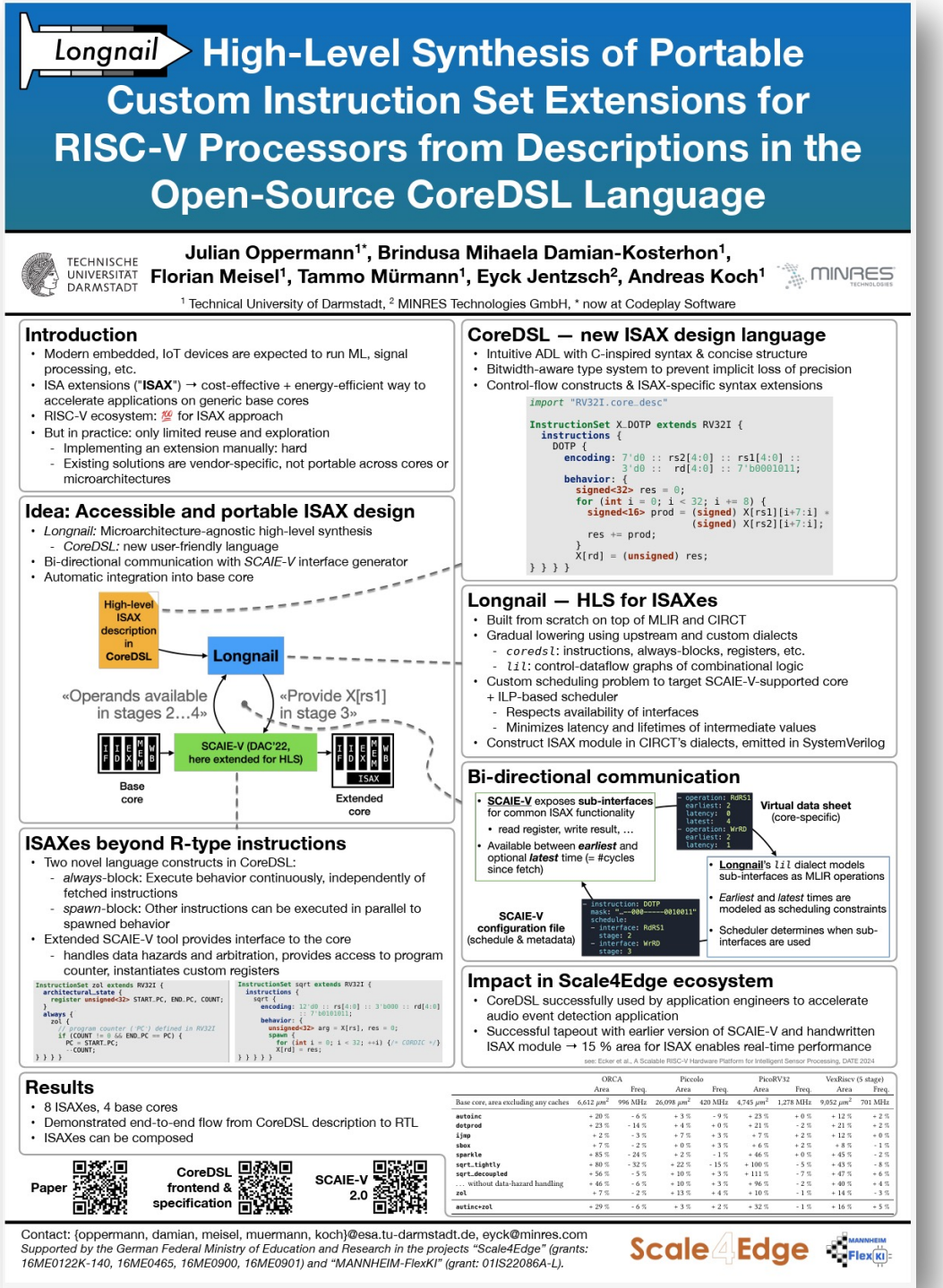
FIRRTL: Supporting Chisel

- FIRRTL is the name of the compiler IR used by the Chisel hardware description language (HDL)
- HDLs are DSLs used to describe circuit structure
- In CIRCT we are writing a drop-in replacement for the Scala FIRRTL compiler
- FIRRTL IR parser imports to the FIRRTL Dialect
- FIRRTL Dialect lowers to common CIRCT Dialects



Example: Longnail

- RISC-V Instruction Set Extensions
- From DSL to implementation



Longnail

High-Level Synthesis of Portable Custom Instruction Set Extensions for RISC-V Processors from Descriptions in the Open-Source CoreDSL Language

TECHNISCHE UNIVERSITÄT DARMSTADT

Julian Oppermann[1*], Brindusa Mihaela Damian-Kosterhon[1], Florian Meisel[1], Tammo Mürmann[1], Eyck Jentzsch[2], Andreas Koch[1]

MINRES TECHNOLOGIES

[1] Technical University of Darmstadt, [2] MINRES Technologies GmbH, * now at Codeplay Software

Introduction

- Modern embedded, IoT devices are expected to run ML, signal processing, etc.
- ISA extensions ("**ISAX**") → cost-effective + energy-efficient way to accelerate applications on generic base cores
- RISC-V ecosystem: 💯 for ISAX approach
- But in practice: only limited reuse and exploration
  - Implementing an extension manually: hard
  - Existing solutions are vendor-specific, not portable across cores or microarchitectures

Idea: Accessible and portable ISAX design

- *Longnail:* Microarchitecture-agnostic high-level synthesis
  - *CoreDSL:* new user-friendly language
- Bi-directional communication with *SCAIE-V* interface generator
- Automatic integration into base core

ISAXes beyond R-type instructions

- Two novel language constructs in CoreDSL:
  - *always*-block: Execute behavior continuously, independently of fetched instructions
  - *spawn*-block: Other instructions can be executed in parallel to spawned behavior
- Extended SCAIE-V tool provides interface to the core
  - handles data hazards and arbitration, provides access to program counter, instantiates custom registers

```
InstructionSet zol extends RV32I {
  architectural_state {
    register unsigned<32> START_PC, END_PC, COUNT;
  }
  always {
    zol {
      // program counter ('PC') defined in RV32I
      if (COUNT != 0 && END_PC == PC) {
        PC = START_PC;
        --COUNT;
      }
    }
  }
}
```

```
InstructionSet sqrt extends RV32I {
  instructions {
    sqrt {
      encoding: 12'd0 :: rs[4:0] :: 3'b000 :: rd[4:0]
                :: 7'b0101011;
      behavior: {
        unsigned<32> arg = X[rs], res = 0;
        spawn {
          for (int i = 0; i < 32; ++i) {/* CORDIC */}
          X[rd] = res;
        }
      }
    }
  }
}
```

CoreDSL — new ISAX design language

- Intuitive ADL with C-inspired syntax & concise structure
- Bitwidth-aware type system to prevent implicit loss of precision
- Control-flow constructs & ISAX-specific syntax extensions

```
import "RV32I.core_desc"

InstructionSet X_DOTP extends RV32I {
  instructions {
    DOTP {
      encoding: 7'd0 :: rs2[4:0] :: rs1[4:0] ::
                3'd0 ::  rd[4:0] :: 7'b0001011;
      behavior: {
        signed<32> res = 0;
        for (int i = 0; i < 32; i += 8) {
          signed<16> prod = (signed) X[rs1][i+7:i] *
                            (signed) X[rs2][i+7:i];
          res += prod;
        }
        X[rd] = (unsigned) res;
      }
    }
  }
}
```

Longnail — HLS for ISAXes

- Built from scratch on top of MLIR and CIRCT
- Gradual lowering using upstream and custom dialects
  - `coredsl`: instructions, always-blocks, registers, etc.
  - `lil`: control-dataflow graphs of combinational logic
- Custom scheduling problem to target SCAIE-V-supported core + ILP-based scheduler
  - Respects availability of interfaces
  - Minimizes latency and lifetimes of intermediate values
- Construct ISAX module in CIRCT's dialects, emitted in SystemVerilog

Bi-directional communication

- **SCAIE-V** exposes **sub-interfaces** for common ISAX functionality
  - read register, write result, …
- Available between ***earliest*** and optional ***latest*** time (= #cycles since fetch)

```
- operation: RdRS1
  earliest: 2
  latest:   4
- operation: WrRD
  earliest: 2
  latest:   1
```

Virtual data sheet (core-specific)

- **Longnail**'s `lil` dialect models sub-interfaces as MLIR operations
- *Earliest* and *latest* times are modeled as scheduling constraints
- Scheduler determines when sub-interfaces are used

SCAIE-V configuration file (schedule & metadata)

```
- instruction: DOTP
  mask: "-----000-----0010011"
  schedule:
  - interface: RdRS1
    stage: 2
  - interface: WrRD
    stage: 3
```

Impact in Scale4Edge ecosystem

- CoreDSL successfully used by application engineers to accelerate audio event detection application
- Successful tapeout with earlier version of SCAIE-V and handwritten ISAX module → 15 % area for ISAX enables real-time performance

see: Ecker et al., A Scalable RISC-V Hardware Platform for Intelligent Sensor Processing, DATE 2024

Results

- 8 ISAXes, 4 base cores
- Demonstrated end-to-end flow from CoreDSL description to RTL
- ISAXes can be composed

	ORCA		Piccolo		PicoRV32		VexRiscv (5 stage)	
	Area	Freq.	Area	Freq.	Area	Freq.	Area	Freq.
Base core, area excluding any caches	6,612 μm^2	996 MHz	26,098 μm^2	420 MHz	4,745 μm^2	1,278 MHz	9,852 μm^2	701 MHz
autoinc	+ 20 %	- 6 %	+ 3 %	- 9 %	+ 23 %	+ 0 %	+ 12 %	+ 2 %
dotprod	+ 23 %	- 14 %	+ 4 %	+ 0 %	+ 21 %	- 2 %	+ 21 %	+ 2 %
ijmp	+ 2 %	- 3 %	+ 7 %	+ 3 %	+ 7 %	+ 2 %	+ 12 %	+ 0 %
sbox	+ 7 %	- 2 %	+ 0 %	+ 3 %	+ 6 %	+ 2 %	+ 8 %	- 1 %
sparkle	+ 85 %	- 24 %	+ 2 %	- 1 %	+ 46 %	+ 0 %	+ 45 %	- 2 %
sqrt_tightly	+ 80 %	- 32 %	+ 22 %	- 15 %	+ 100 %	- 5 %	+ 43 %	- 8 %
sqrt_decoupled	+ 56 %	- 5 %	+ 10 %	+ 3 %	+ 111 %	- 7 %	+ 47 %	+ 6 %
… without data-hazard handling	+ 46 %	- 6 %	+ 10 %	+ 3 %	+ 96 %	- 2 %	+ 40 %	+ 4 %
zol	+ 7 %	- 2 %	+ 13 %	+ 4 %	+ 10 %	- 1 %	+ 14 %	- 3 %
autinc+zol	+ 29 %	- 6 %	+ 3 %	+ 2 %	+ 32 %	- 1 %	+ 16 %	+ 5 %

Paper

CoreDSL frontend & specification

SCAIE-V 2.0

Contact: {oppermann, damian, meisel, muermann, koch}@esa.tu-darmstadt.de, eyck@minres.com

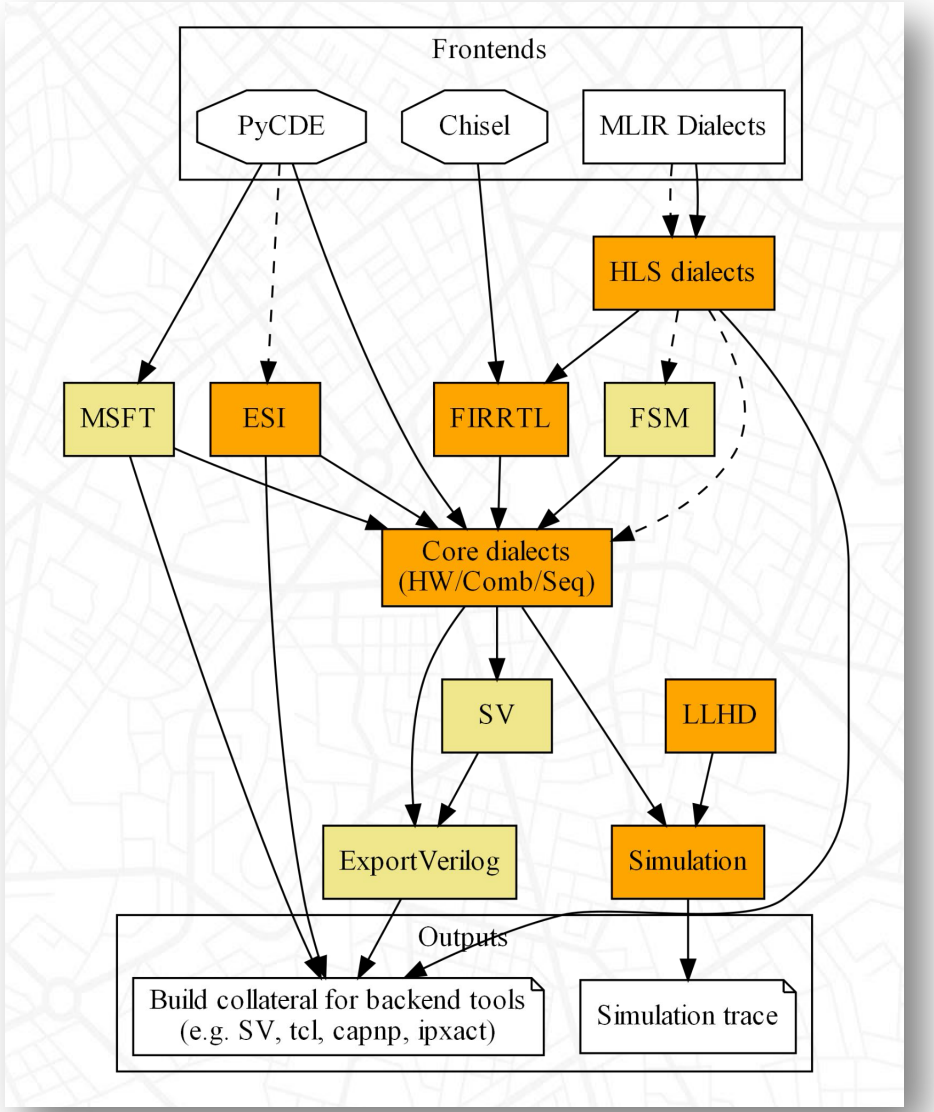
Supported by the German Federal Ministry of Education and Research in the projects "Scale4Edge" (grants: 16ME0122K-140, 16ME0465, 16ME0900, 16ME0901) and "MANNHEIM-FlexKI" (grant: 01IS22086A-L).

Scale4Edge

MANNHEIM FlexKI

Core Dialects

- HW: core abstractions
  - Operations like module, instances
  - Standard data types
- Comb: Combinational logic
  - Computational operations
  - All operators etc.
- Seq: Sequential logic
  - Clocked storage elements
  - Sense of timing



System Verilog support

- Recently System Verilog support was added
- Use of slang frontend and transforms to dialects

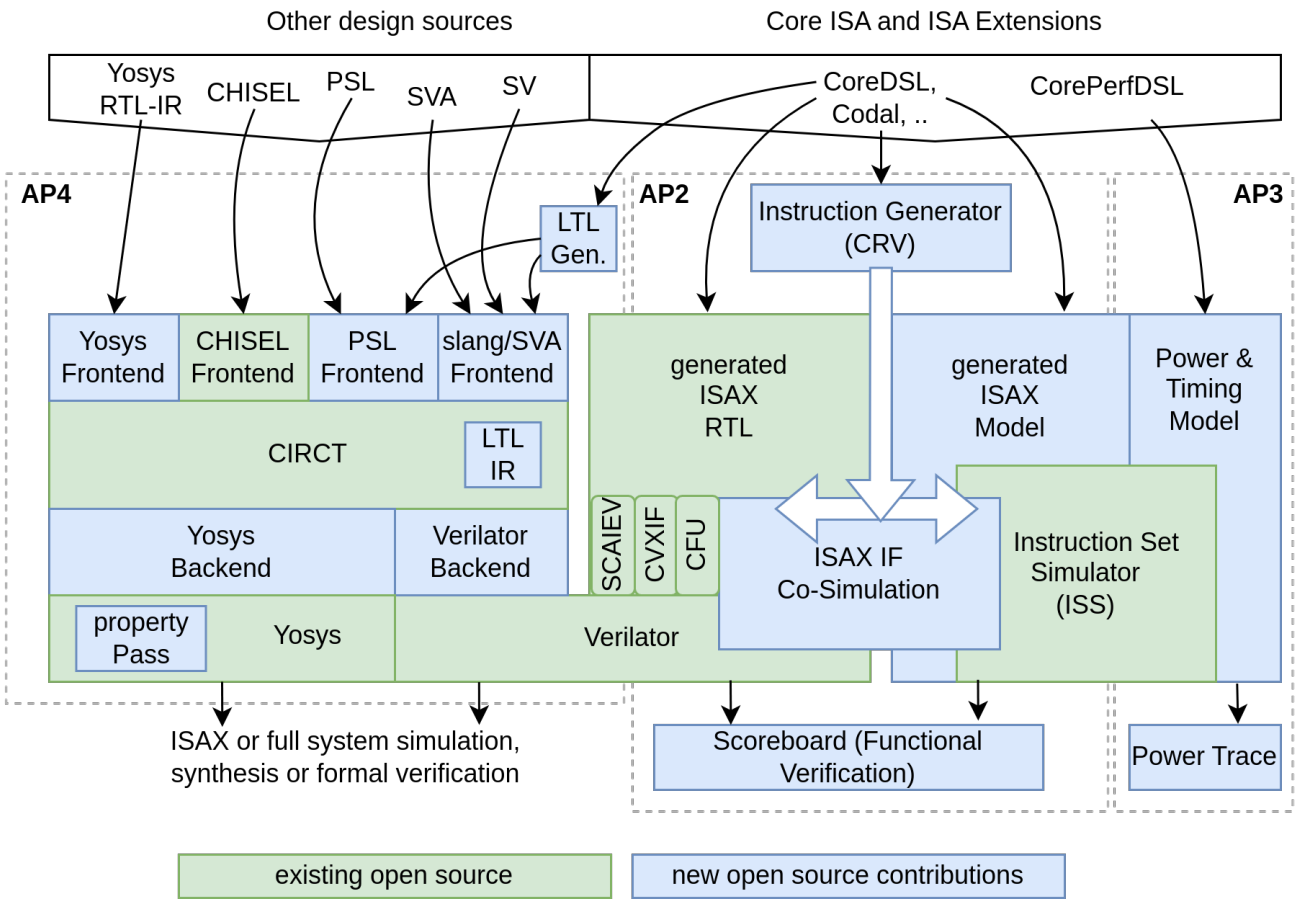
Demo: Toy Example

- Built into the upstream implementation:
  - Renames all wires to foo_<id++>
- Build your own transform: Requires basic compiler knowledge

Our activities: OSVISE

- Open Source Verification of Instruction Set Extensions

- HM activities
  - Support for linear temporal logic
  - Improve SV frontend
  - Interoperability
    - Yosys<->CIRCT
    - CIRCT->Verilator



Open and Upcoming PhD and Postdoc Positions

- Open Source EDA tooling
  - Netlist-to-netlist transformations
  - Improved integration and orchestration
  - Improved inference of DSP primitives
  - Scalable formal verification framework (with Yosys)
  - Integration of Surfer waveform viewer and Verilator
  - Strongly typed hardware design language and CIRCT

- RISC-V Instruction Set Extensions
  - Integrated safety and security features
  - Focus topic: variable length instructions for better code density
  - Support for DSP workloads

HM

Thank You