面向RISC-V异构AI芯片的“大编译器”设计和实现

伍华林

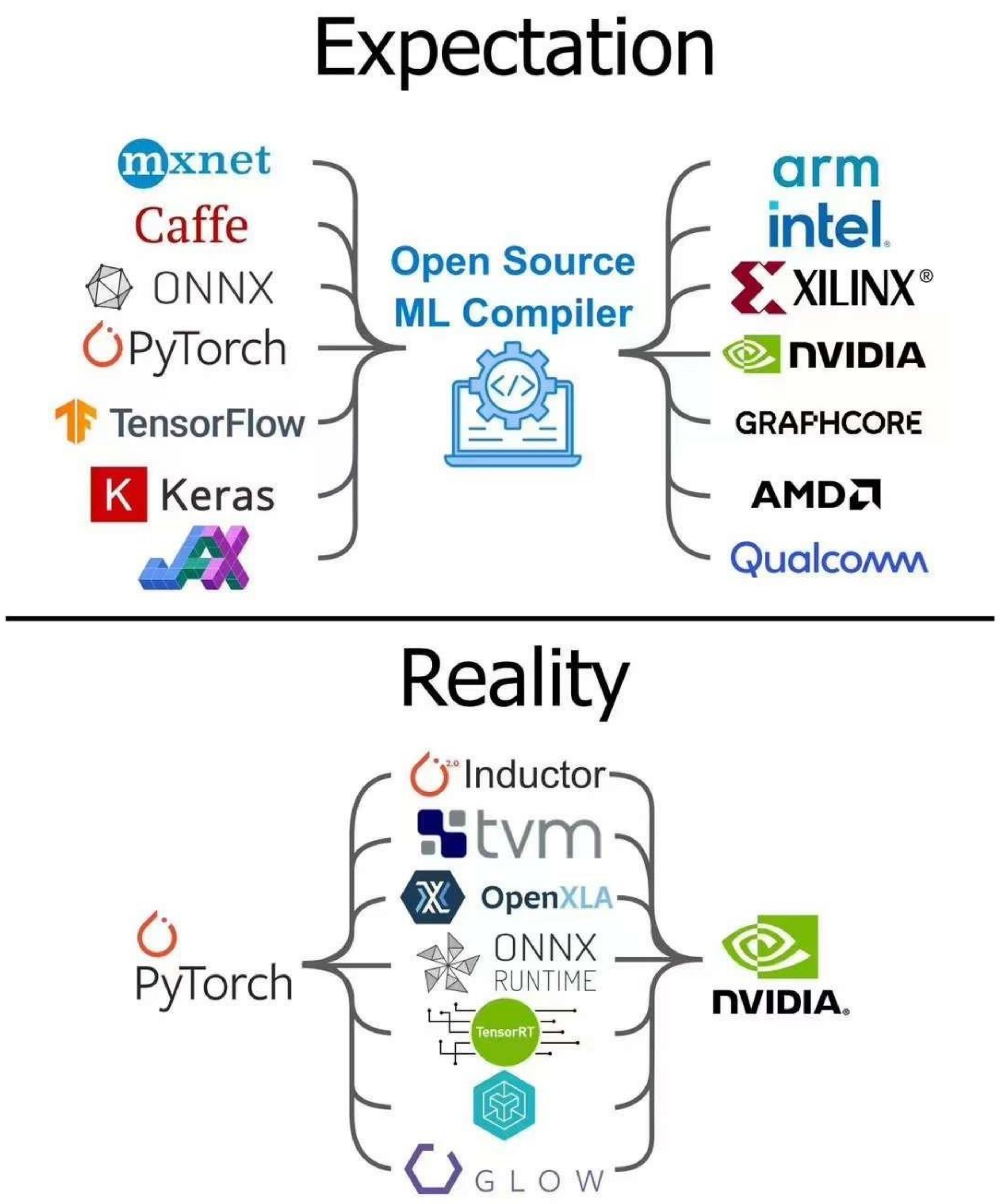
兆松科技（武汉）有限公司

兆松科技
Tera Pines

目录
CONTENTS

动机

- 软件1.0时代 -> 软件2.0时代
- 新模型，AI加速器层🎧不穷
- 模型 x 框架 x 加速器：指数级增长
- 算子库维护成本，CUDA兼容问题

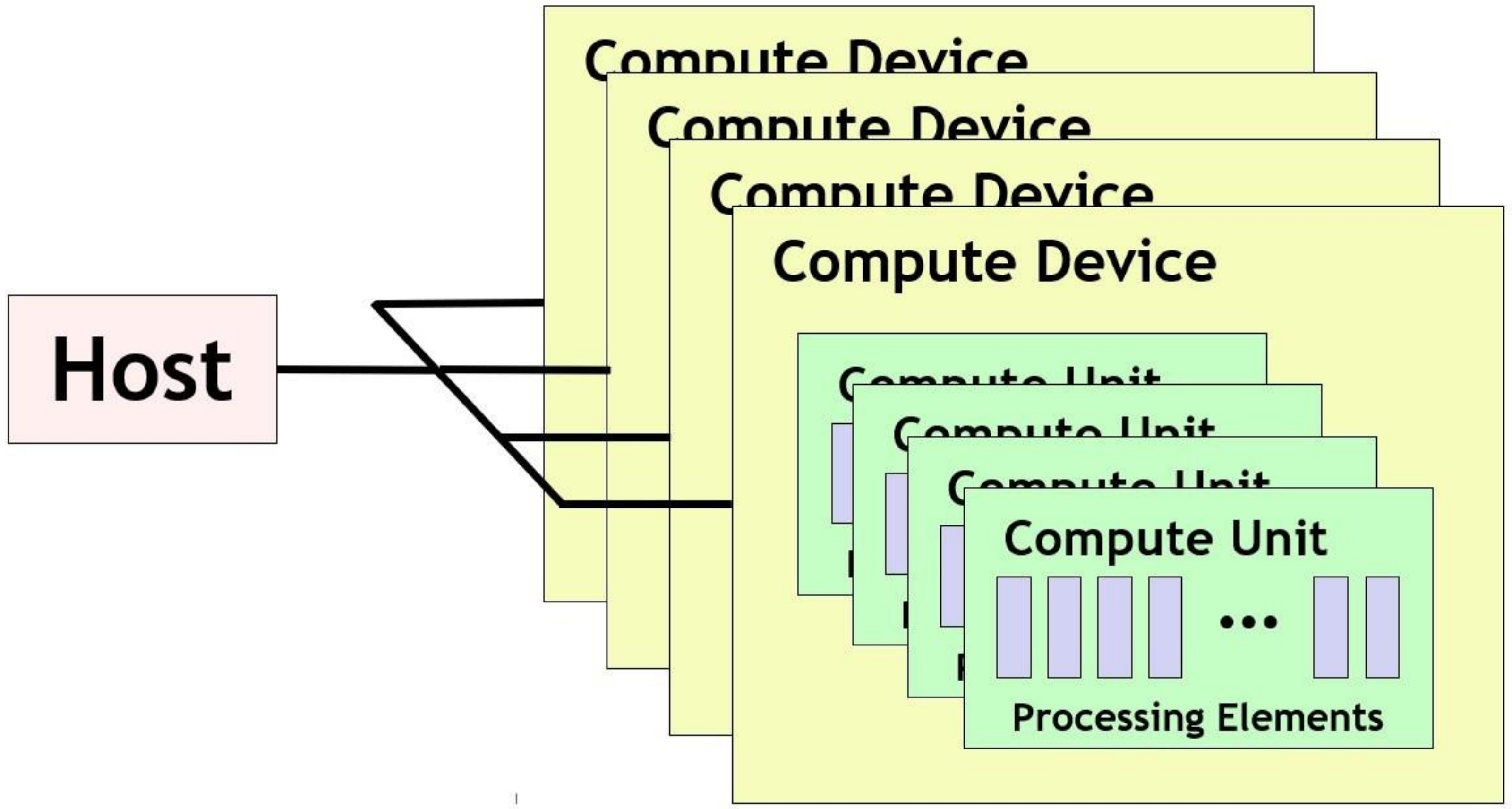


动机

- LLVM对加速器的支持成熟且容易
- LLVM擅长做标量优化
- MLIR提供了多层抽象，适合做LLVM不擅长的优化
- MLIR后端直接生成LLVM IR方言

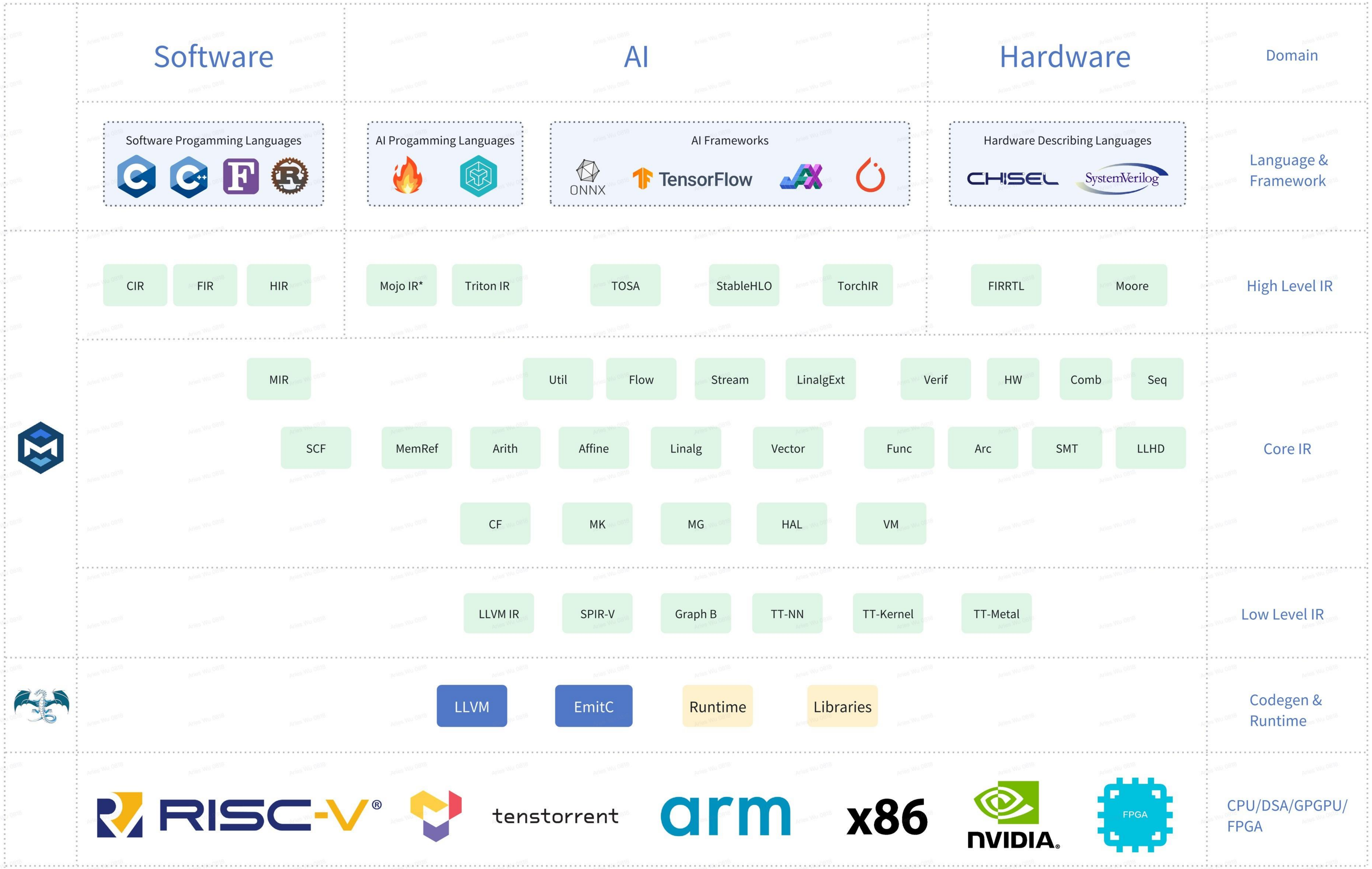
“大编译器”设计

- 传统编译器：针对某一款特定指令集的芯片的编译器
- 大编译器
  - 对控制器和加速器的计算和任务程序，以及两者之间的通信进行抽象
  - 自动生成异构计算芯片的所有代码（包括驱动，加速器代码）



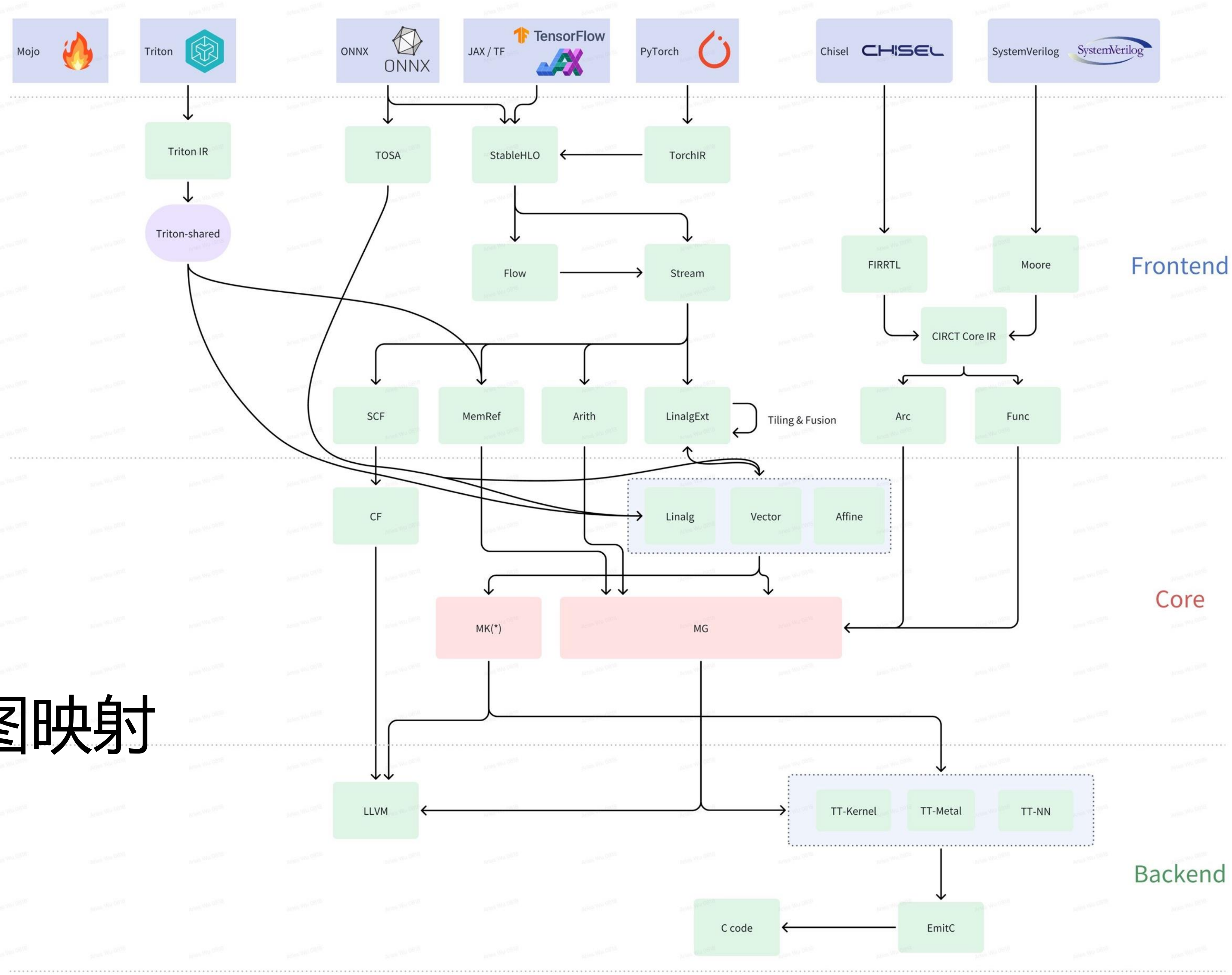
异构计算系统案例 - OpenCL Platform Model

“大编译器” 设计


Software
AI
Hardware
Domain
Software Progamming Languages
AI Progamming Languages
AI Frameworks
ONNX
TensorFlow
Hardware Describing Languages
CHISEL
SystemVerilog
Language & Framework
CIR
FIR
HIR
Mojo IR*
Triton IR
TOSA
StableHLO
TorchIR
FIRRTL
Moore
High Level IR
MIR
Util
Flow
Stream
LinalgExt
Verif
HW
Comb
Seq
SCF
MemRef
Arith
Affine
Linalg
Vector
Func
Arc
SMT
LLHD
CF
MK
MG
HAL
VM
Core IR
LLVM IR
SPIR-V
Graph B
TT-NN
TT-Kernel
TT-Metal
Low Level IR
LLVM
EmitC
Runtime
Libraries
Codegen & Runtime
RISC-V
tenstorrent
arm
x86
NVIDIA
FPGA
CPU/DSA/GPGPU/FPGA

“大编译器”设计 – MLIR前端

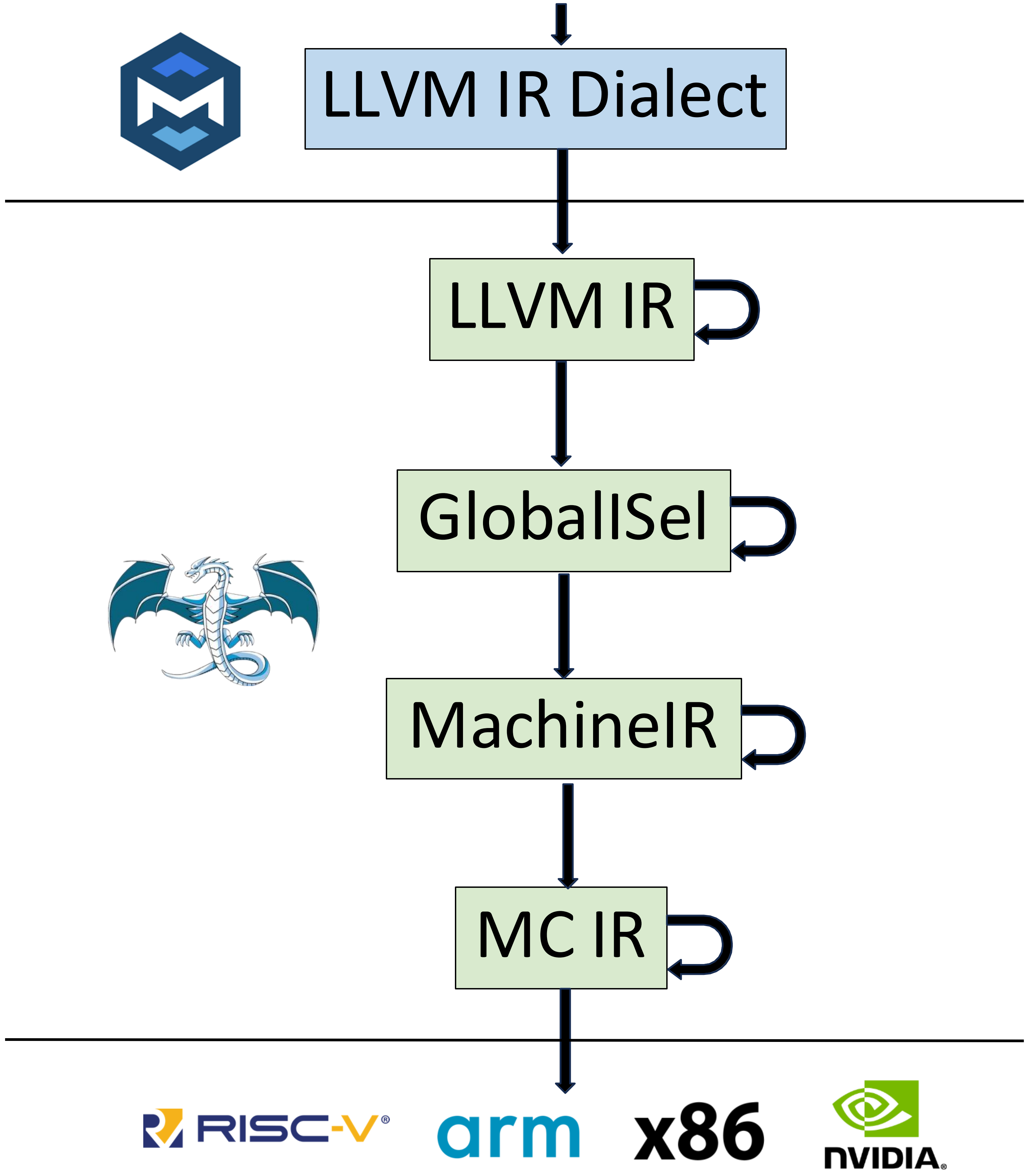
- 基于IREE/MLIR的一些核心方言
- 支持Triton/Mojo*编程语言
- 支持多种模型文件导入
- 支持Dataflow芯片计算图生成
- 统一的算子和计算图抽象层
- 支持RTL计算图到Dataflow计算图映射



*未来支持

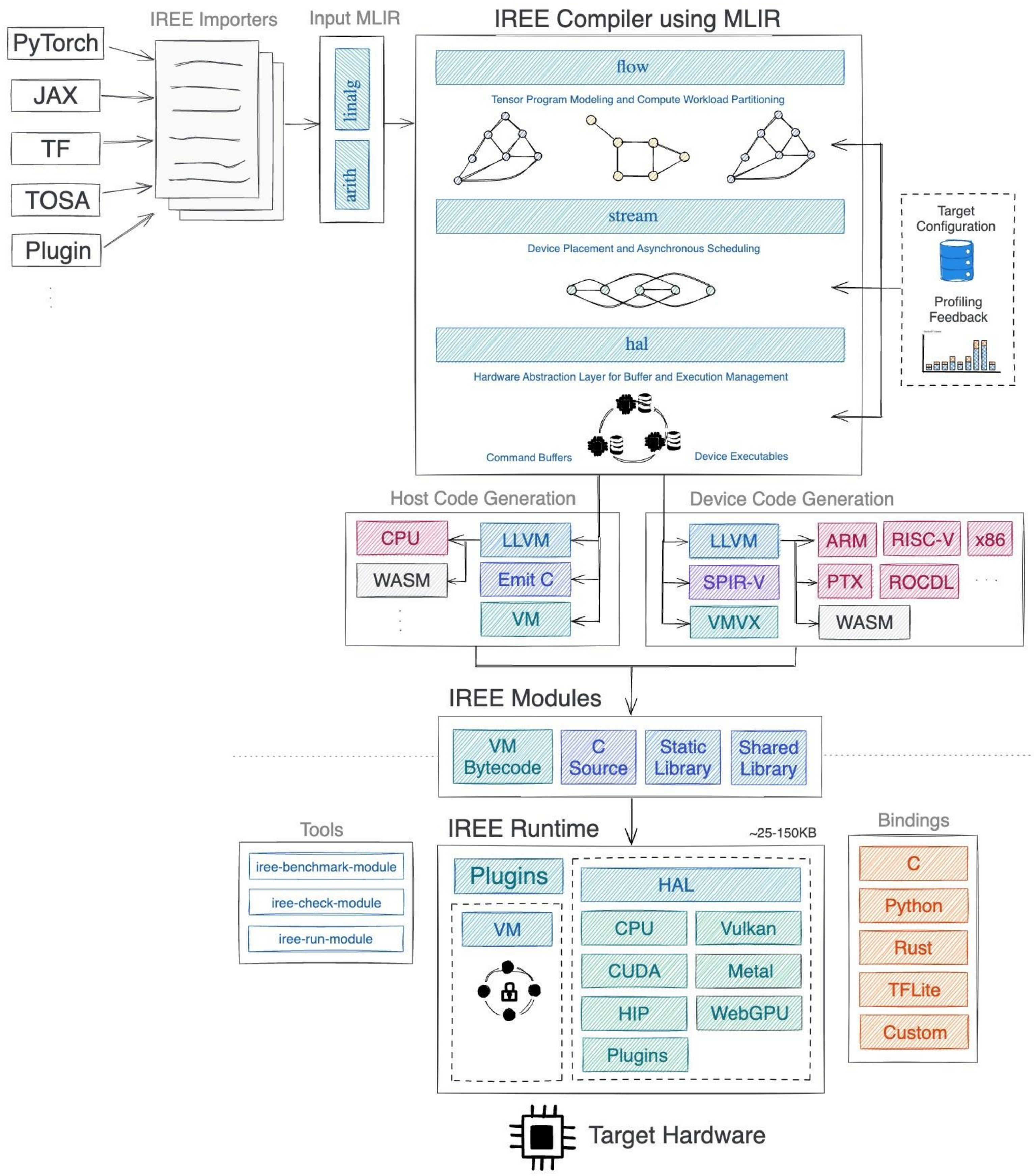
“大编译器” 设计 – LLVM后端

- 无缝衔接MLIR
- 支持多种硬件平台
- 处理标量相关的优化
- 指令选择，寄存器分配，指令排序等
- 快速适配各类RISC-V DSA



“大编译器” 实现 – 模型导入

- 基于IREE的模型导入
- 支持导入PyTorch, JAX/Tensorflow, ONNX
  - Dialect: TorchIR, StableHLO, TOSA
- 统一下降到LinalgExt, Arith, Flow etc



“大编译器”实现 – 算子库

- Triton 算子库 -> MLIR Dialect TTIR
- 纯算法，跨平台的开源算子库
- MK (Magic Kernel) 抽象不同硬件的算子库
- 通过MK映射到各硬件平台的算子库
  - 自动生成Kernel runtime

```
#any_device = #tt.operand_constraint<dram|l1|scalar|tile|any_device|any_device_tile>
module attributes {tt.system_desc = #tt.system_desc<[{arch = <wormhole_b0>, grid = <8x8>
  func.func @forward(%arg0: tensor<64x128xf32>, %arg1: tensor<64x128xf32>) -> tensor<64x
    %0 = tensor.empty() : tensor<64x128xf32>
    %1 = "ttir.multiply"(%arg0, %arg1, %0) <{operandSegmentSizes = array<i32: 2, 1>, ope
    return %1 : tensor<64x128xf32>
  }
}
```

```
  func.func @forward(%arg0: tensor<64x128xf32, #layout>, %arg1: tensor<64x128xf32, #layo
    %0 = "ttmetal.alloc"() <{address = 262144 : i64, memory_space = #l1_, size = 32768 :
    %1 = "ttmetal.host_write"(%arg0, %0) : (tensor<64x128xf32, #layout>, tensor<64x128xf
    %2 = "ttmetal.alloc"() <{address = 294912 : i64, memory_space = #l1_, size = 32768 :
    %3 = "ttmetal.host_write"(%arg1, %2) : (tensor<64x128xf32, #layout>, tensor<64x128xf
    %4 = "ttmetal.alloc"() <{address = 327680 : i64, memory_space = #l1_, size = 32768 :
    %5 = "ttmetal.dispatch"(%1, %3, %4) <{core_ranges = [#ttmetal.core_range<0x0, 1x1>,
    ^bb0(%arg2: !ttkernel.cb<0, 0, memref<64x128xf32, #l1_>>, %arg3: !ttkernel.cb<0, 1,
      "ttkernel.cb_push_back"(%arg2) : (!ttkernel.cb<0, 0, memref<64x128xf32, #l1_>>) ->
      "ttkernel.return"() : () -> ()
    }, {
    ^bb0(%arg2: !ttkernel.cb<0, 0, memref<64x128xf32, #l1_>>, %arg3: !ttkernel.cb<0, 1,
      "ttkernel.cb_push_back"(%arg3) : (!ttkernel.cb<0, 1, memref<64x128xf32, #l1_>>) ->
      "ttkernel.return"() : () -> ()
    }, {
    ^bb0(%arg2: !ttkernel.cb<0, 0, memref<64x128xf32, #l1_>>, %arg3: !ttkernel.cb<0, 1,
      "ttkernel.builtin"(%arg2, %arg3, %arg4) <{kind = @eltwise, op = @mulitply}> : (!tt
      "ttkernel.return"() : () -> ()
    }) : (tensor<64x128xf32, #layout1>, tensor<64x128xf32, #layout1>, tensor<64x128xf32,
    "ttmetal.dealloc"(%2) : (tensor<64x128xf32, #layout1>) -> ()
    "ttmetal.dealloc"(%0) : (tensor<64x128xf32, #layout1>) -> ()
    %6 = "ttmetal.alloc"() <{address = 0 : i64, memory_space = #system, size = 32768 : i
    %7 = "ttmetal.host_read"(%5, %6) : (tensor<64x128xf32, #layout1>, tensor<64x128xf32,
    "ttmetal.dealloc"(%4) : (tensor<64x128xf32, #layout1>) -> ()
    return %7 : tensor<64x128xf32, #layout>
  }
}
```

“大编译器” 实现 – 运行时抽象

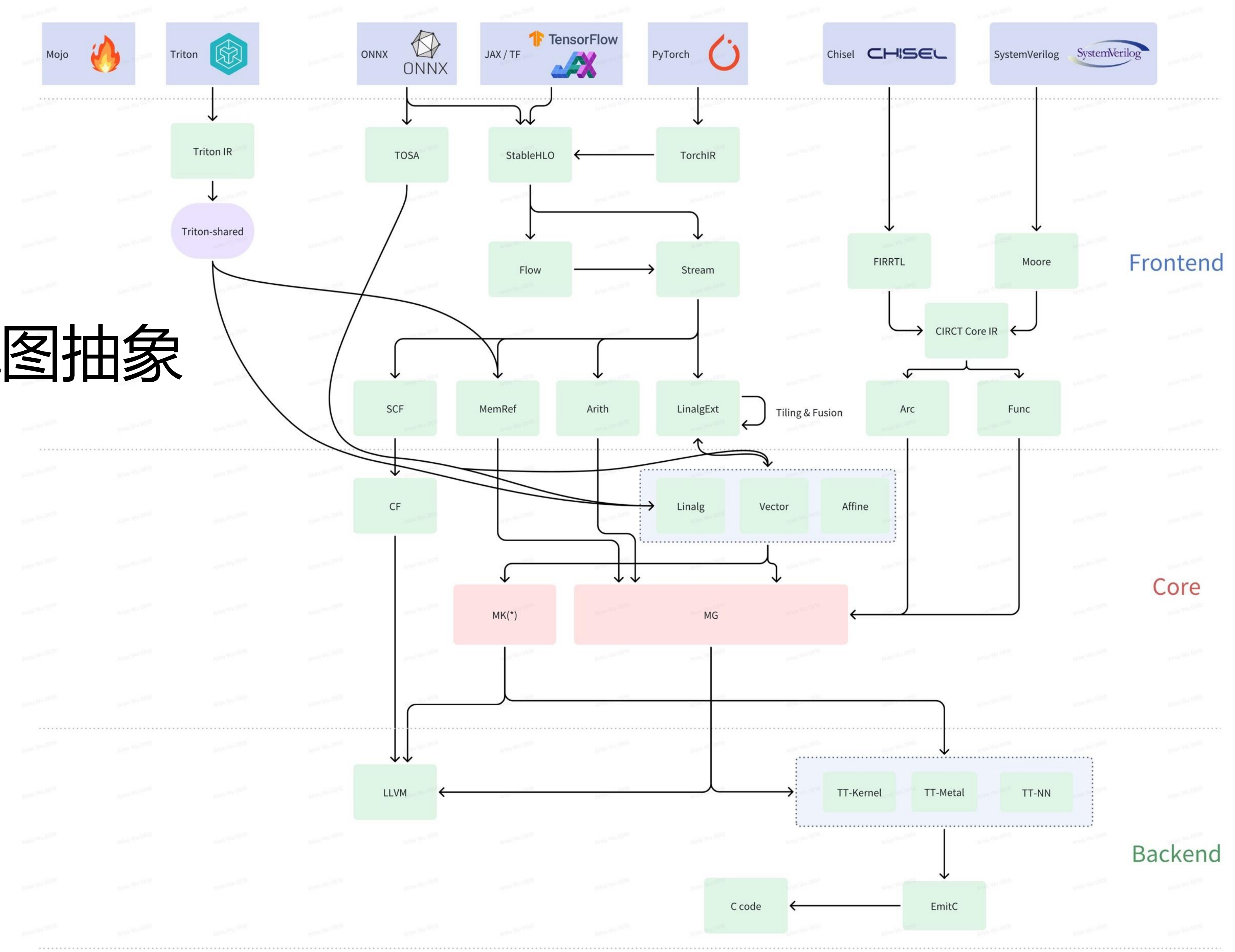
- 和平台无关的运行时抽象
- 自动生成控制器代码
- 自动生成平台相关算子
- 算子库编写者无需考虑硬件平台API
  - 硬件平台API通过编译器隐藏和自动生成

```
#any_device = #tt.operand_constraint<dram|l1|scalar|tile|any_device|any_device_tile>
module attributes {tt.system_desc = #tt.system_desc<[{arch = <wormhole_b0>, grid = <8x8>
  func.func @forward(%arg0: tensor<64x128xf32>, %arg1: tensor<64x128xf32>) -> tensor<64x
    %0 = tensor.empty() : tensor<64x128xf32>
    %1 = "ttir.multiply"(%arg0, %arg1, %0) <{operandSegmentSizes = array<i32: 2, 1>, ope
    return %1 : tensor<64x128xf32>
  }
}
```

```
module attributes {tt.device = #tt.device<#tt.grid<8x8, (d0, d1) -> (0, d0, d1)>, [0]>,
  func.func @forward(%arg0: tensor<64x128xf32, #layout>, %arg1: tensor<64x128xf32, #layo
    %0 = "ttmetal.alloc"() <{address = 262144 : i64, memory_space = #l1_, size = 32768 :
    %1 = "ttmetal.host_write"(%arg0, %0) : (tensor<64x128xf32, #layout>, tensor<64x128xf
    %2 = "ttmetal.alloc"() <{address = 294912 : i64, memory_space = #l1_, size = 32768 :
    %3 = "ttmetal.host_write"(%arg1, %2) : (tensor<64x128xf32, #layout>, tensor<64x128xf
    %4 = "ttmetal.alloc"() <{address = 327680 : i64, memory_space = #l1_, size = 32768 :
    %5 = "ttmetal.dispatch"(%1, %3, %4) <{core_ranges = [#ttmetal.core_range<0x0, 1x1>,
    ^bb0(%arg2: !ttkernel.cb<0, 0, memref<64x128xf32, #l1_>>, %arg3: !ttkernel.cb<0, 1,
      "ttkernel.cb_push_back"(%arg2) : (!ttkernel.cb<0, 0, memref<64x128xf32, #l1_>>) ->
      "ttkernel.return"() : () -> ()
    }, {
    ^bb0(%arg2: !ttkernel.cb<0, 0, memref<64x128xf32, #l1_>>, %arg3: !ttkernel.cb<0, 1,
      "ttkernel.cb_push_back"(%arg3) : (!ttkernel.cb<0, 1, memref<64x128xf32, #l1_>>) ->
      "ttkernel.return"() : () -> ()
    }, {
    ^bb0(%arg2: !ttkernel.cb<0, 0, memref<64x128xf32, #l1_>>, %arg3: !ttkernel.cb<0, 1,
      "ttkernel.builtin"(%arg2, %arg3, %arg4) <{kind = @eltwise, op = @mulitply}> : (!ttl
      "ttkernel.return"() : () -> ()
    }) : (tensor<64x128xf32, #layout1>, tensor<64x128xf32, #layout1>, tensor<64x128xf32,
    "ttmetal.dealloc"(%2) : (tensor<64x128xf32, #layout1>) -> ()
    "ttmetal.dealloc"(%0) : (tensor<64x128xf32, #layout1>) -> ()
    %6 = "ttmetal.alloc"() <{address = 0 : i64, memory_space = #system, size = 32768 : i
    %7 = "ttmetal.host_read"(%5, %6) : (tensor<64x128xf32, #layout1>, tensor<64x128xf32,
    "ttmetal.dealloc"(%4) : (tensor<64x128xf32, #layout1>) -> ()
    return %7 : tensor<64x128xf32, #layout>
  }
}
```

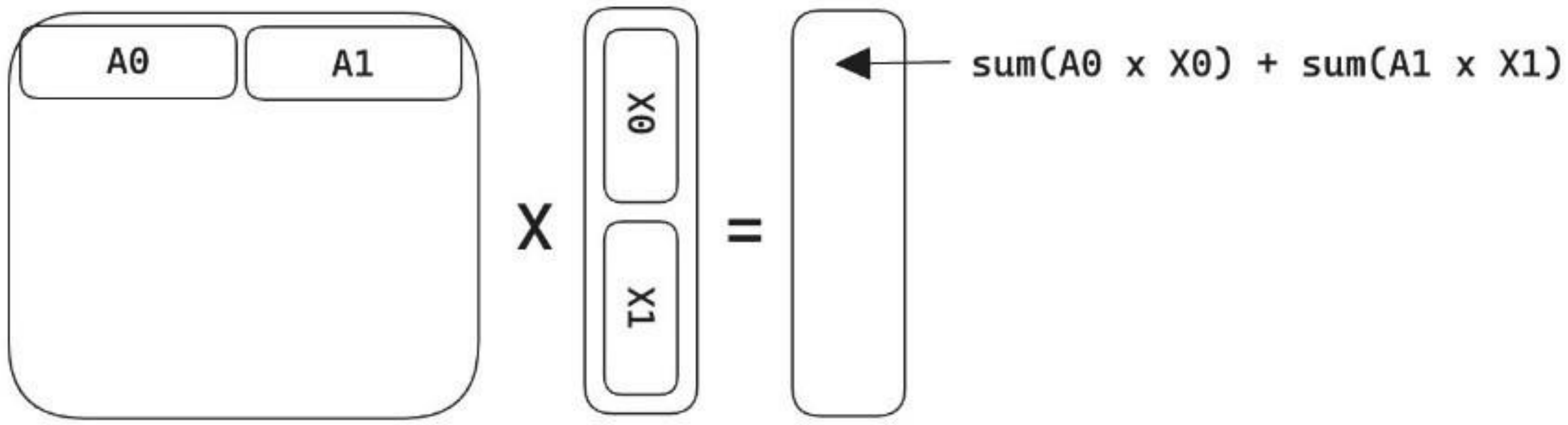
“大编译器” 实现 – 计算图

- Flow – AI模型计算图抽象
- Stream – 硬件流水线抽象
- MG (Magic Graph) – 通用计算图抽象
- 图切割算法
  - 动态计算时间模型
  - 数据通信模型
  - 全局变量重排
  - 缩减def-use链



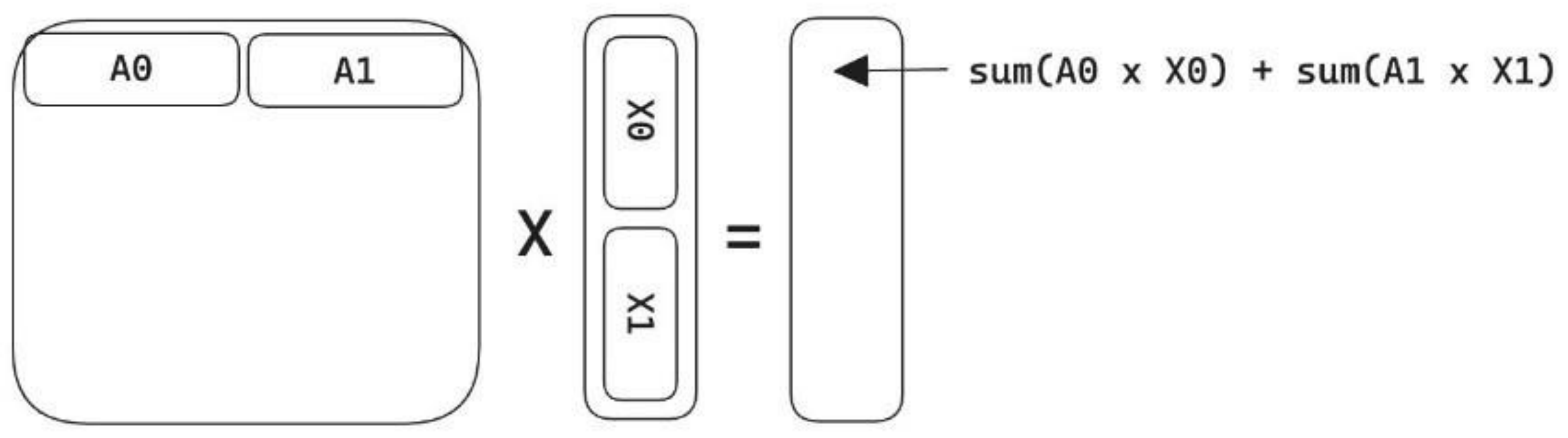
Example - 简化的 GEMV in Triton

```
# Y = A @ X, where A is a matrix of M x N, X is a vector of N. @triton.jit
def gemv_kernel(Y, A, X, M, N, stride_am, BLOCK_SIZE_M: tl.constexpr, BLOCK_SIZE_N: tl.constexpr):
    start_m = tl.program_id(0)
    rm = start_m * BLOCK_SIZE_M + tl.arange(0, BLOCK_SIZE_M) rn =
    tl.arange(0, BLOCK_SIZE_N)

    A = A + (rm[:, None] * stride_am + rn[None, :])
    X = X + rn

    acc = tl.zeros((BLOCK_SIZE_M, ), dtype=tl.float32) for n in
    range(N, 0, -BLOCK_SIZE_N):
        a = tl.load(A)
        x = tl.load(X)
        acc += tl.sum(a * x[None, :], axis=1) A +=
        BLOCK_SIZE_N
        X += BLOCK_SIZE_N

    Y = Y + rm
    tl.store(Y, acc)
```

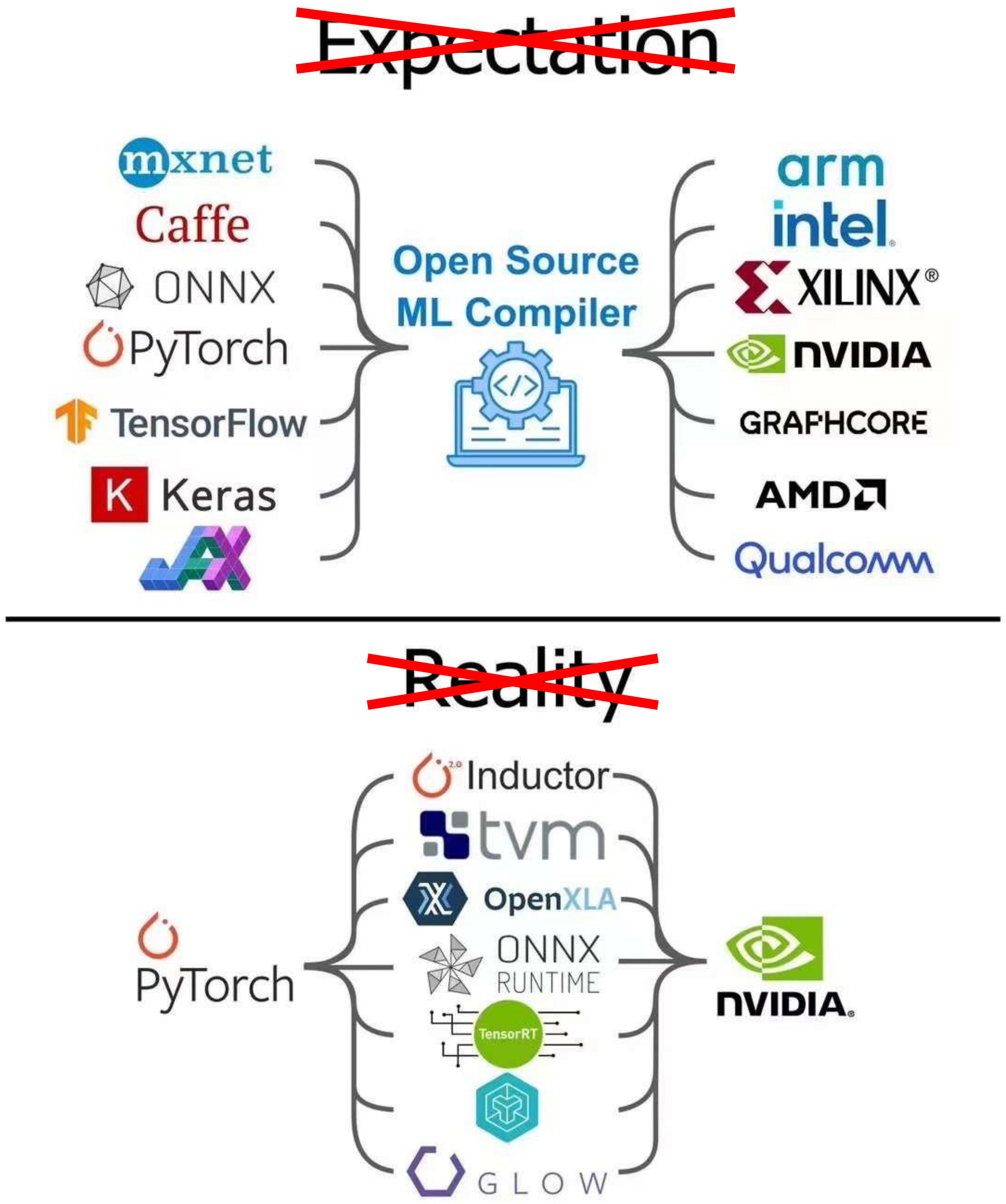


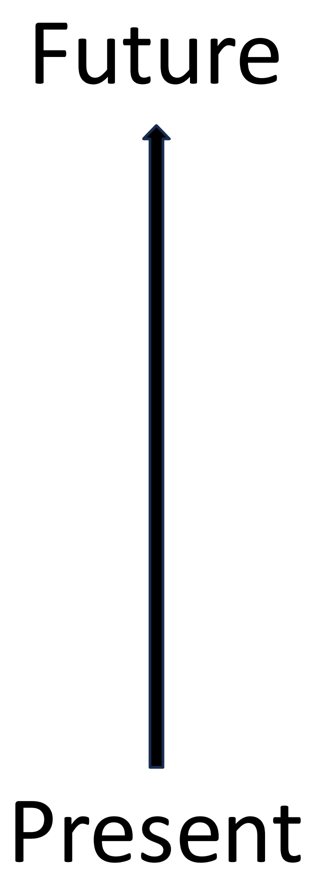
Example - 简化的 GEMV in TTIR

```
module {
  tt.func public @gemv_kernel(%arg0: !tt.ptr<f32> loc("/path/to/gemv.py":10:0), %arg1: !tt.ptr<f32> loc("/path/to/gemv.py":10:0), %arg
    %cst = arith.constant dense<0.000000e+00> : tensor<32xf32> loc(#loc1)
    %c0_i32 = arith.constant 0 : i32 loc(#loc1)
    %cst_0 = arith.constant dense<32> : tensor<32xi32> loc(#loc1)
    %cst_1 = arith.constant dense<32> : tensor<32x32xi32> loc(#loc1)
    %c32_i32 = arith.constant 32 : i32 loc(#loc1)
    %0 = tt.get_program_id x : i32 loc(#loc2)
    %1 = arith.muli %0, %c32_i32 : i32 loc(#loc3)
    %2 = tt.make_range {end = 32 : i32, start = 0 : i32} : tensor<32xi32> loc(#loc4)
    %3 = tt.splat %1 : i32 -> tensor<32xi32> loc(#loc5)
    %4 = arith.addi %3, %2 : tensor<32xi32> loc(#loc5)
    %5 = tt.expand_dims %4 {axis = 1 : i32} : tensor<32xi32> -> tensor<32x1xi32> loc(#loc6)
    %6 = tt.splat %arg5 : i32 -> tensor<32x1xi32> loc(#loc7)
    %7 = arith.muli %5, %6 : tensor<32x1xi32> loc(#loc7)
    %8 = tt.expand_dims %2 {axis = 0 : i32} : tensor<32xi32> -> tensor<1x32xi32> loc(#loc8)
    %9 = tt.broadcast %7 : tensor<32x1xi32> -> tensor<32x32xi32> loc(#loc9)
    %10 = tt.broadcast %8 : tensor<1x32xi32> -> tensor<32x32xi32> loc(#loc9)
    %11 = arith.addi %9, %10 : tensor<32x32xi32> loc(#loc9)
    %12 = tt.splat %arg1 : !tt.ptr<f32> -> tensor<32x32x!tt.ptr<f32>> loc(#loc10)
    %13 = tt.addptr %12, %11 : tensor<32x32x!tt.ptr<f32>>, tensor<32x32xi32> loc(#loc10)
    %14 = tt.splat %arg2 : !tt.ptr<f32> -> tensor<32x!tt.ptr<f32>> loc(#loc11)
    %15 = tt.addptr %14, %2 : tensor<32x!tt.ptr<f32>>, tensor<32xi32> loc(#loc11)
    %16:3 = scf.for %arg6 = %c0_i32 to %arg4 step %c32_i32 iter_args(%arg7 = %cst, %arg8 = %13, %arg9 = %15) -> (tensor<32xf32>, tenso
      %19 = tt.load %arg8 : tensor<32x32x!tt.ptr<f32>> loc(#loc13)
      %20 = tt.load %arg9 : tensor<32x!tt.ptr<f32>> loc(#loc14)
      %21 = tt.expand_dims %20 {axis = 0 : i32} : tensor<32xf32> -> tensor<1x32xf32> loc(#loc15)
      %22 = tt.broadcast %21 : tensor<1x32xf32> -> tensor<32x32xf32> loc(#loc16)
      %23 = arith.mulf %19, %22 : tensor<32x32xf32> loc(#loc16)
      %24 = "tt.reduce"(%23) <{axis = 1 : i32}> ({
      ^bb0(%arg10: f32 loc(unknown), %arg11: f32 loc(unknown)):
        %28 = arith.addf %arg10, %arg11 : f32 loc(#loc29)
        tt.reduce.return %28 : f32 loc(#loc27)
      }) : (tensor<32x32xf32>) -> tensor<32xf32> loc(#loc27)
      %25 = arith.addf %arg7, %24 : tensor<32xf32> loc(#loc20)
      %26 = tt.addptr %arg8, %cst_1 : tensor<32x32x!tt.ptr<f32>>, tensor<32x32xi32> loc(#loc21)
      %27 = tt.addptr %arg9, %cst_0 : tensor<32x!tt.ptr<f32>>, tensor<32xi32> loc(#loc22)
      scf.yield %25, %26, %27 : tensor<32xf32>, tensor<32x32x!tt.ptr<f32>>, tensor<32x!tt.ptr<f32>>
    } loc(#loc12)
    %17 = tt.splat %arg0 : !tt.ptr<f32> -> tensor<32x!tt.ptr<f32>> loc(#loc24)
    %18 = tt.addptr %17, %4 : tensor<32x!tt.ptr<f32>>, tensor<32xi32> loc(#loc24)
    tt.store %18, %16#0 : tensor<32x!tt.ptr<f32>> loc(#loc25)
    tt.return loc(#loc26)
  } loc(#loc)
} loc(#loc)
```



总结


~~Expectation~~
mxnet
Caffe
ONNX
PyTorch
TensorFlow
Keras
Open Source
ML Compiler
arm
intel
XILINX
NVIDIA
GRAPHCORE
AMD
Qualcomm
~~Reality~~
PyTorch
Inductor
tvm
OpenXLA
ONNX
RUNTIME
TensorRT
GLOW
NVIDIA


Future
Present

Thanks