CV Academy

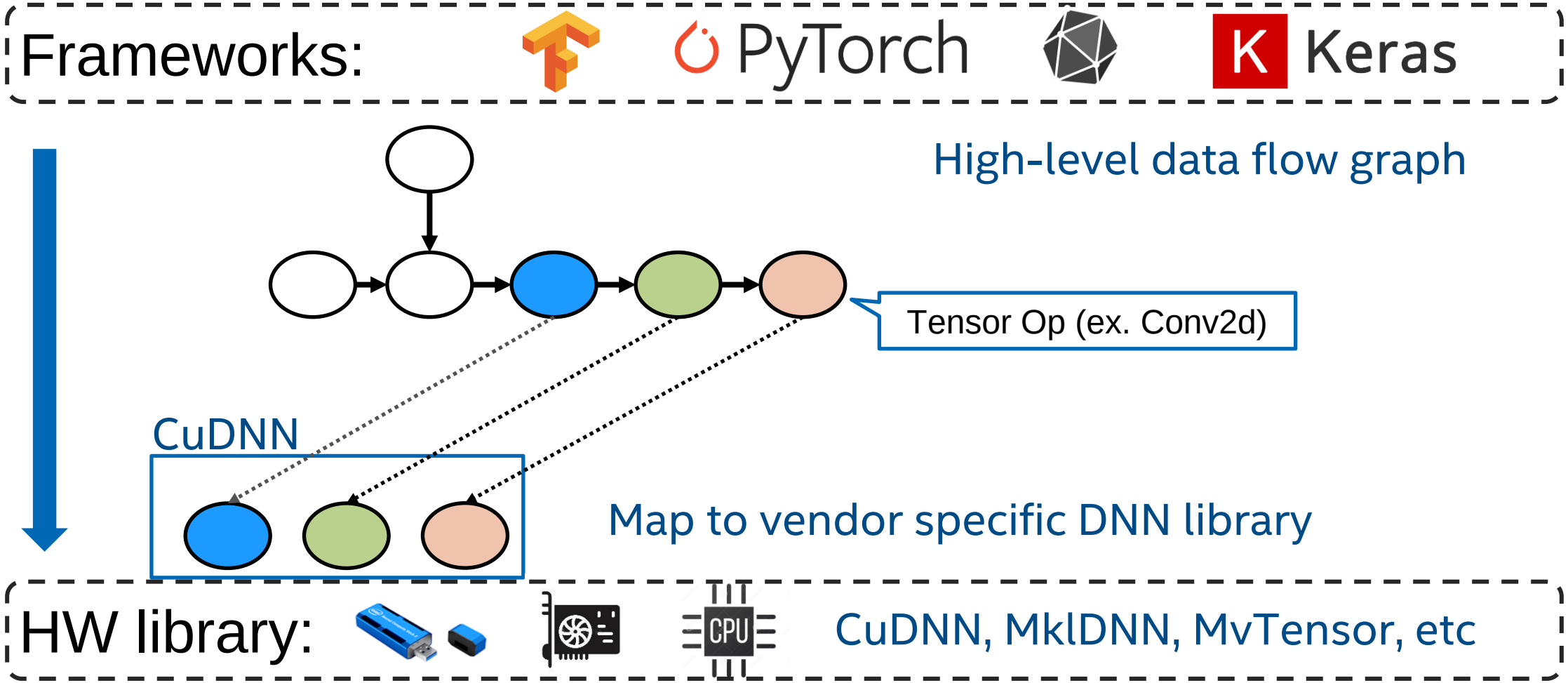
ML Compilers

Novak Alexander

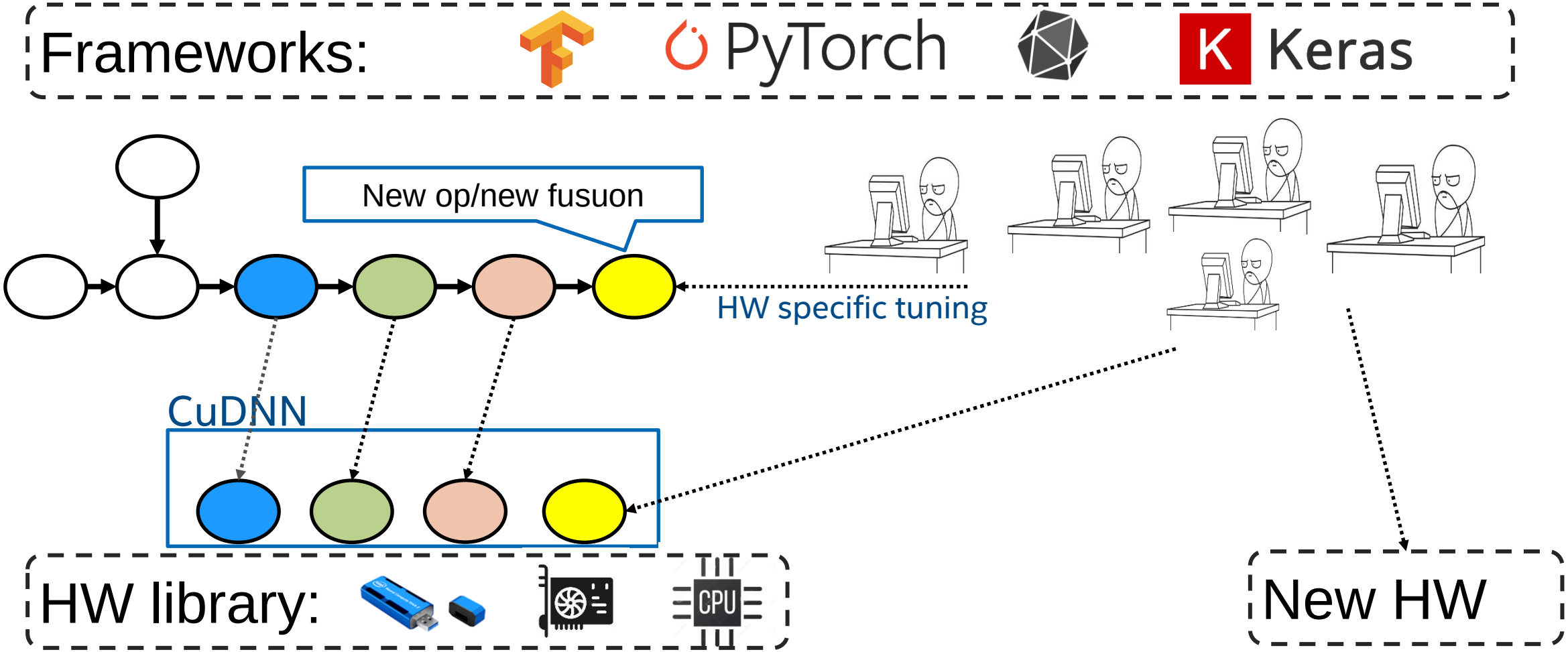
Agenda

- Motivations for ML compiler
- Overview of popular compilers
- VPU compiler in OpenVino
- MLIR
- Summary

Old DL approach



Old DL approach: New platform enablement



Limitations

- Graph representation is too high level
  - It knows nothing about the backend and HW library
- Topologies enablement and layer fusion is limited by vendor specific library
  - Layers mapped to HW or memory bound
- Custom layers
  - What if we don't want to show our layers?

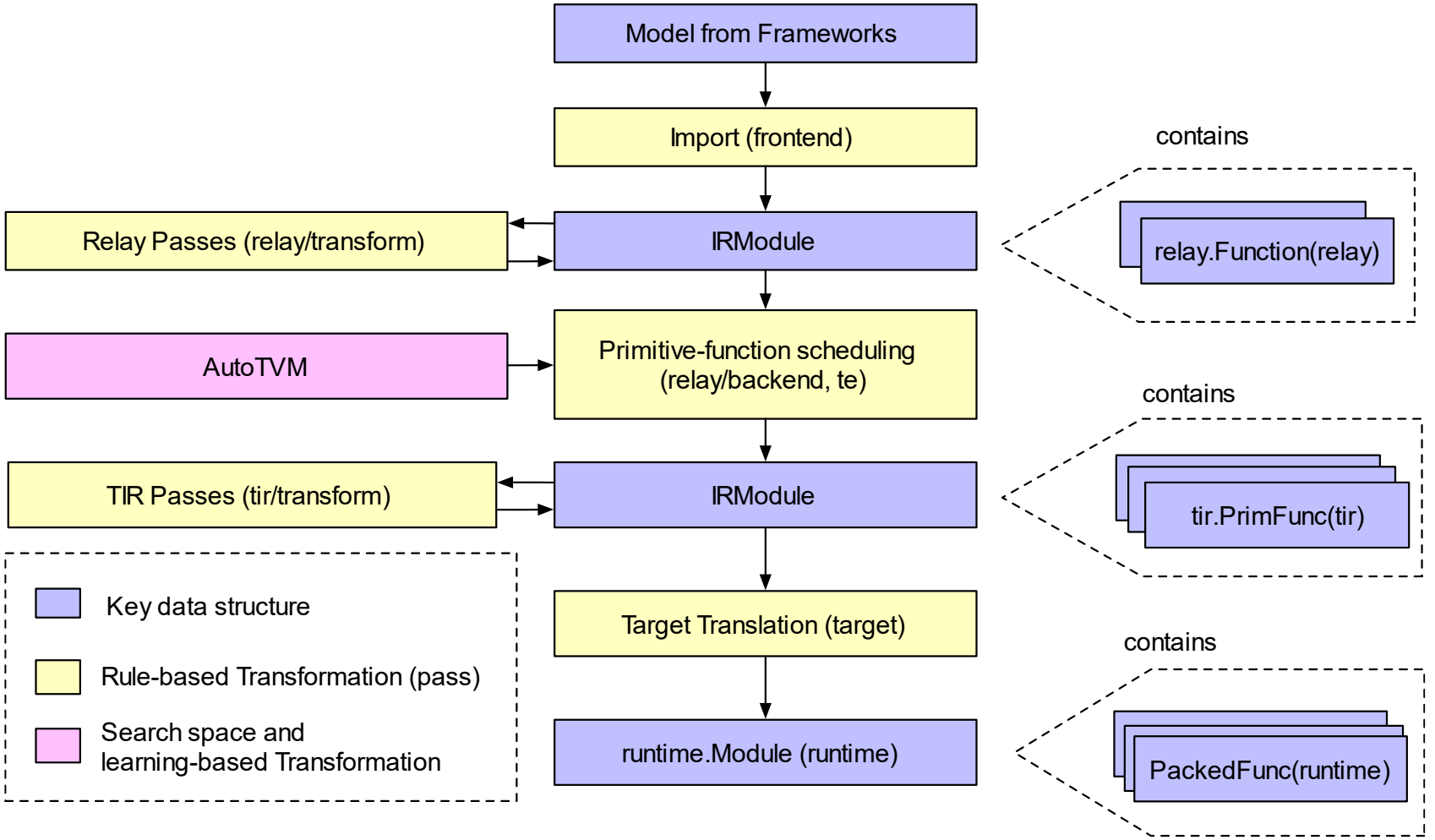
Solutions

- Compiler based approach removes meaning of the layer
- Compiler defines tensor as a native data type leaving layout & memory assignment chooses to the compiler
- Compiler knows HW backend specifics
  - similar access pattern operation fusion
  - optimization choices are based on the cost model or polyhedral analysis
- Assumption: convolution defines speed of the topology
  - the rest is memory

TVM, PlaidML, XLA

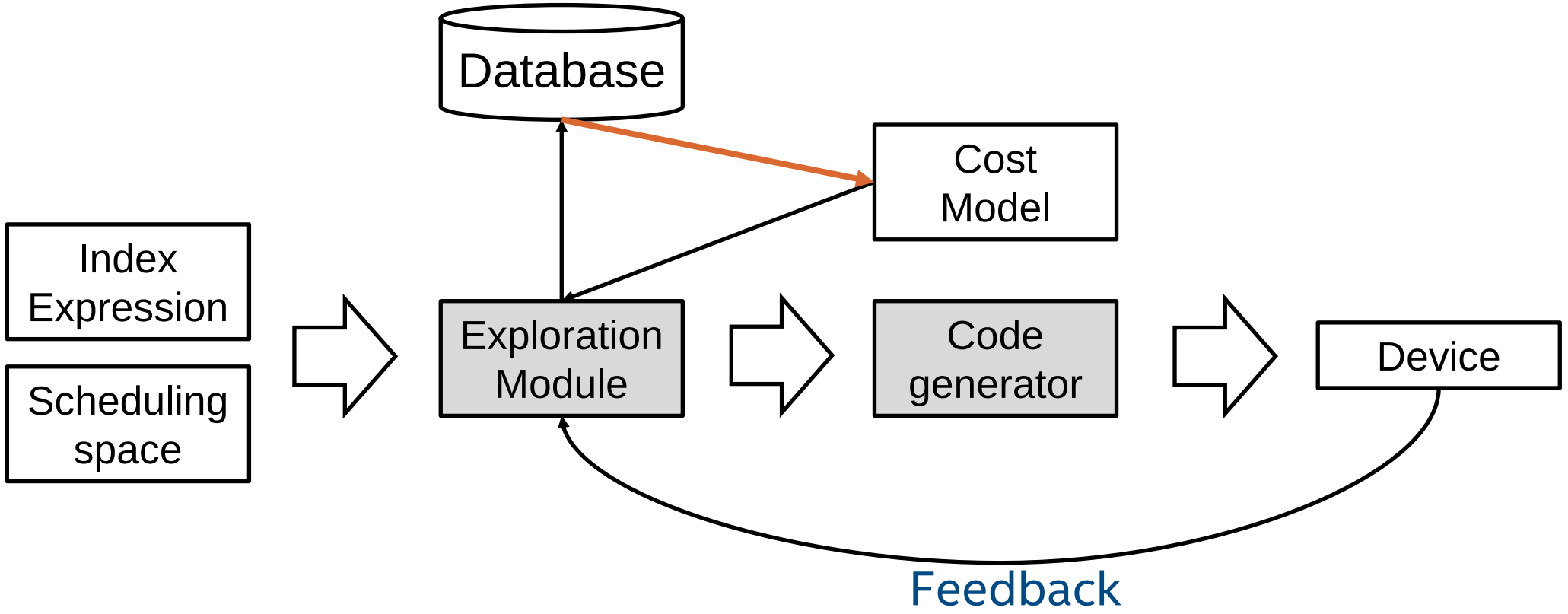
	TVM	**PlaidML**	**VPU GraphTransformer**
High Level Language	Relay	Tile	Ngraph
IR	Tvm IR	MLIR (Stripe)	GT IR
Framework	standalone	standalone	Standalone
Optimizer	ML based	Cost Model	Cost Model
Targets	Nvidia, Mali,x86, arm, FPGA	GPU, CPU	VPU based HW

TVM


Model from Frameworks
Import (frontend)
Relay Passes (relay/transform)
IRModule
AutoTVM
Primitive-function scheduling
(relay/backend, te)
TIR Passes (tir/transform)
IRModule
Target Translation (target)
runtime.Module (runtime)
contains
relay.Function(relay)
contains
tir.PrimFunc(tir)
contains
PackedFunc(runtime)
Key data structure
Rule-based Transformation (pass)
Search space and
learning-based Transformation

TVM arch

Auto TVM: ML based optimization framework



Link to details definition

PlaidML

- Initially developed targeting GPU (Nvidia)
- Implements own language for writing Tensor codes – **Tile**
  - no loops & no layout
- Optimal kernels can be produced from HW config given sufficient constraints
- Intermediate representation - **Stripe**
- Represents tensor operations via Nested Polyhedral Model

PlaidML: Tile

- Written directly in polyhedral form; no nested for loops until writing optimized kernels
- For every valid index, compute right hand side; multiple writes to same output merged using the aggregation operation
- Just formula

- Tile
  - C[i, j : I, J]= + (A[i, k] * B[k, j]);
- TVM (Relay)
  - tvm.sum(a[i, k] * b[k, j], axis = k);
- Example for Conv stride = 2
  - function (I[N, X, CI], F[W, CI, CO]) -> (O) {

```
O[n, x, c: N, (X+1)/2, CO] = +(I[n, 2*x + i, d] *
                               F[i, d,c]);
}
```

PlaidML: Stripe

Kernel Library

```
foreach HW Architecture
  foreach HW Version
    foreach Kernel
      foreach Input Shape
        foreach Output Shape
          write_kernel
```

Schedule Space Searching

```
foreach Kernel
  write_algorithm
  foreach HW Architecture
    write_schedule_space
    foreach HW Version
      foreach Input Shape
        foreach Output Shape
          select_schedule
```

Stripe

```
foreach Kernel
  write_algorithm

foreach HW Architecture
  create_stripe_config
  foreach HW Version
    set_config_params
```

- "removes the combinatorial explosion of engineering work"
- "modular and extensible optimization passes" (no redevelopment)
- "doesn't require physical hardware or even a cycle-accurate model – HI TVM! =)

Link: https://arxiv.org/pdf/1903.06498.pdf

Nested Polyhedral Model

Definition 1. An integer polyhedron $\mathcal{P}$ is a set of all $\vec{x} \in \mathbb{Q}^n$ such that

$$A\vec{x} + \vec{b} \geq \vec{0}, \text{ and}$$
$$A\vec{x} + \vec{b} \in \mathbb{Z}^m$$

where $A \in \mathbb{Q}^{m \times n}$ and $\vec{b} \in \mathbb{Q}^m$.

Definition 2. A *parallel polyhedral block* $(\mathcal{P}, \mathcal{S}_\mathcal{P}, \mathcal{D}, \mathcal{A}_\mathcal{D})$ consists of a polyhedron $\mathcal{P}$ called the *iteration space*, a map $\mathcal{S}_\mathcal{P}$ from points in $\mathcal{P}$ to lists of *statements*, a set of I/O *buffers* $\mathcal{D}$, and a map $\mathcal{A}_\mathcal{D}$ from buffers of $\mathcal{D}$ to associative and commutative operations called the *aggregation operations* satisfying the following:

1. Statements in $\mathcal{S}_\mathcal{P}$ may only read or write to buffers in $\mathcal{D}$ or to internally-scoped temporaries that are not shared between iterations. A single statement list $S_p \in \mathcal{S}_\mathcal{P}$ may have arbitrary dependencies between its statements and is interpreted as running serially.

2. If the statements $s_i \in \mathcal{S}_\mathcal{P}$ for iteration $i \in \mathcal{P}$ write to a buffer element $b \in B \in \mathcal{D}$, no statements $s_j \in \mathcal{S}_\mathcal{P}$ for $j \in \mathcal{P}$, $j \neq i$ may read from this buffer element b.

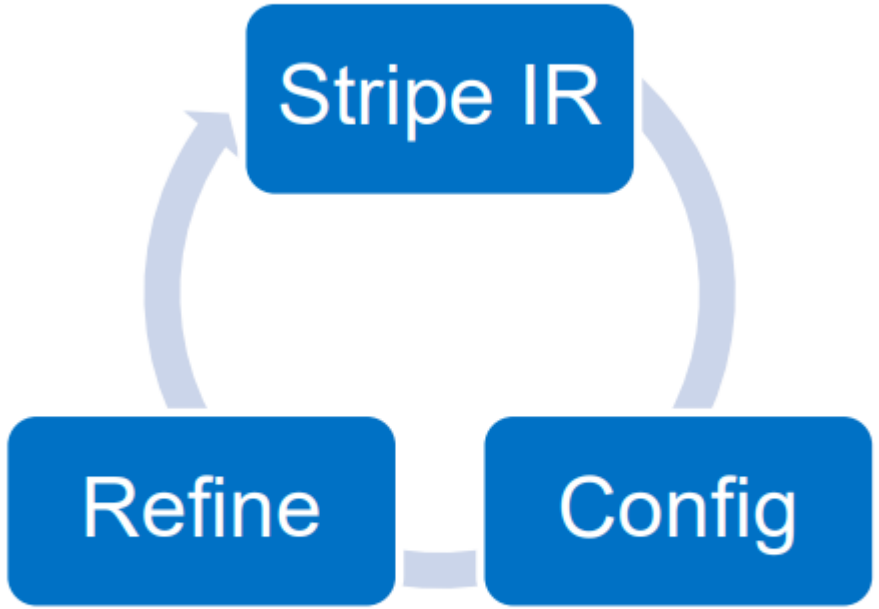
3. When a buffer element $b \in B \in \mathcal{D}$ is written to by statements in statement lists $S_{i_0}, \ldots, S_{i_n} \in \mathcal{S}_\mathcal{P}$ for multiple index values $i_0, \ldots, i_n \in \mathcal{P}$, the value written to b is

$$a_B(v_{i_0}, \ldots, v_{i_n})$$

where $v_{i_0}, \ldots, v_{i_n}$ are the values for b computed by the statement lists $S_{i_0}, \ldots, S_{i_n}$ and $a_B \in \mathcal{A}_\mathcal{D}$ is the aggregation operation associated with B.

PlaidML: Stripe

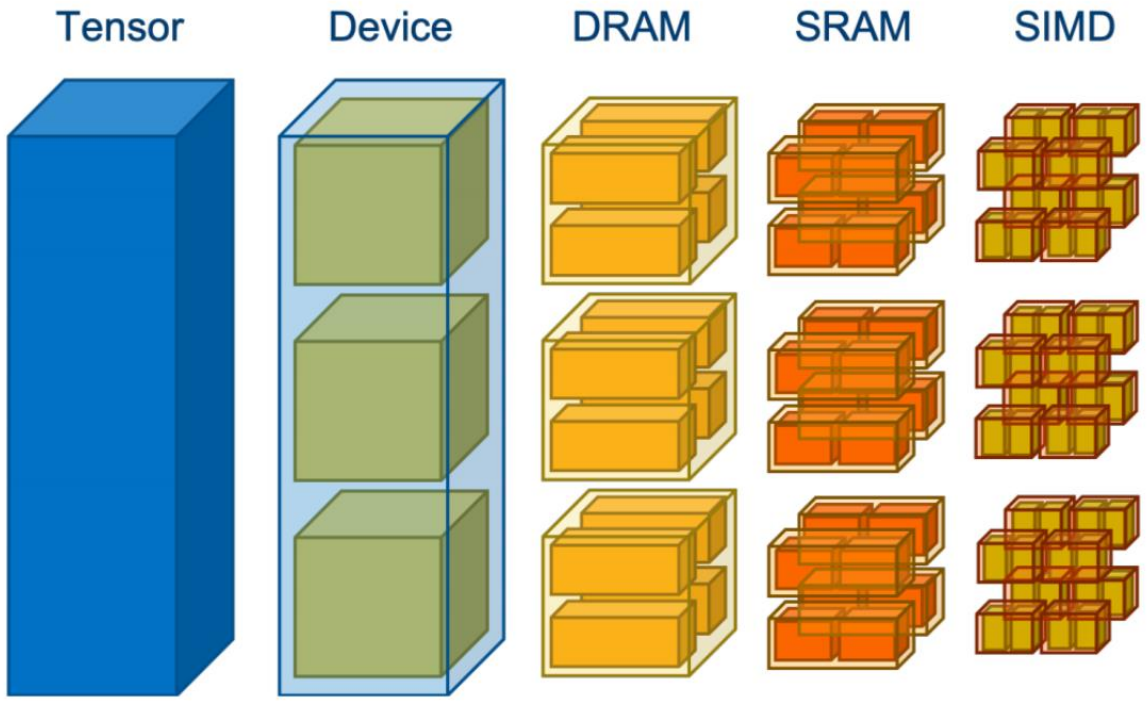
- Stripe a polyhedral IR that is highly amenable to optimization
- Stripe enables distinct passes that process stripe and emit more stripe
- Stripe fundamentally represents operations over a polyhedral tensor space



Link: https://arxiv.org/pdf/1903.06498.pdf

PlaidML: Stripe

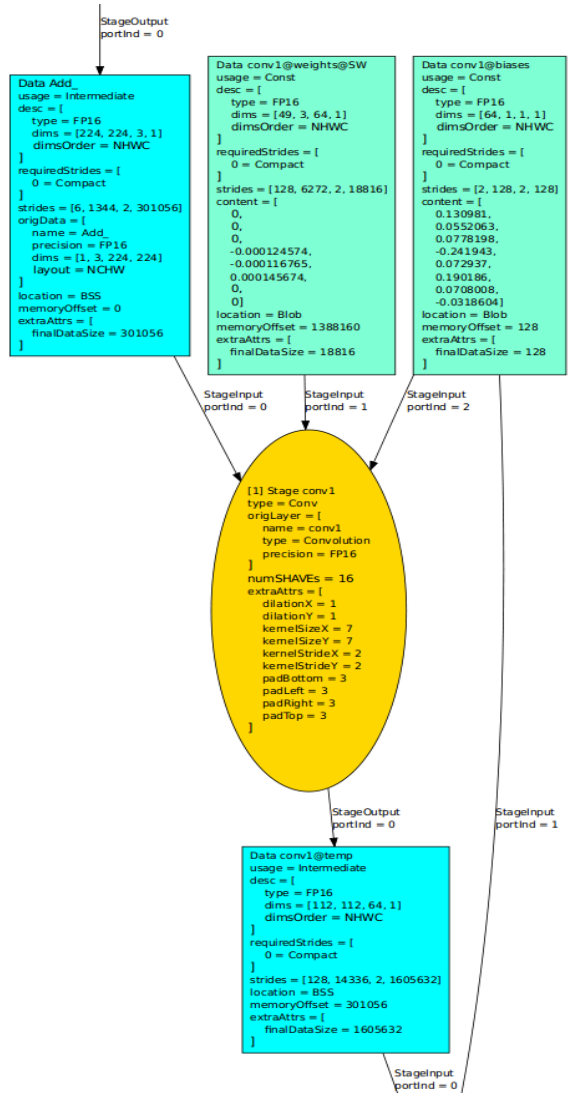
- Stripe enables:
  - Arbitrary tensorization
  - Affine vertical fusion
  - Arbitrarily complex memory hierarchy
  - Heterogenous compute topologies
  - Detailed performance / cost estimates



Link: https://arxiv.org/pdf/1903.06498.pdf

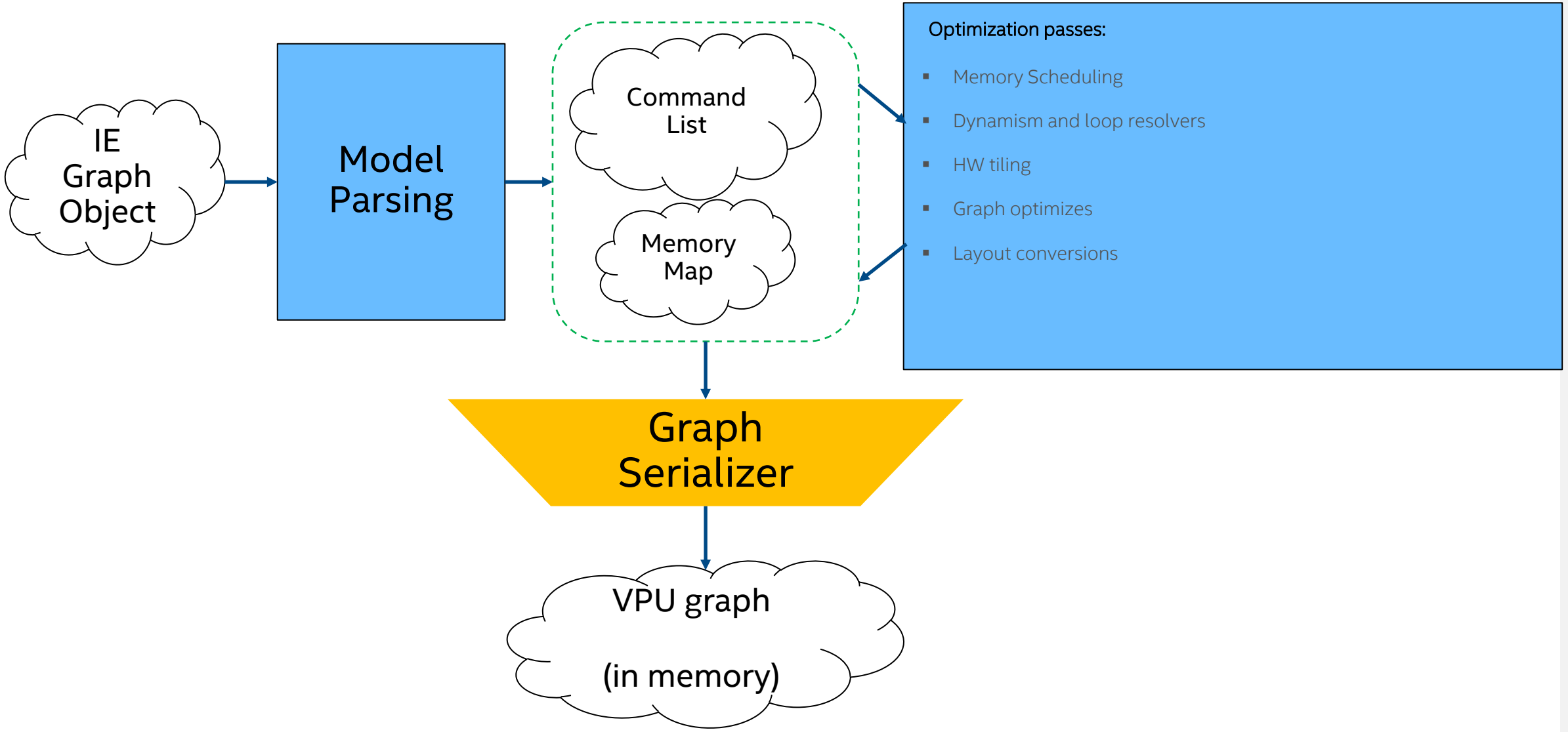
VPU Compiler

- Targeting to Intel Movidius devices (>10)
- Input format- nGraph
- Intermediate representation – Own Graph Model
- Based on cost model function



Link: [vpu_compiler]

VPU Graph Transformer



Link: vpu_compiler

Main concepts

- Represent ML graph rotations as optimizing compiler problem
- Tensorization defines the rest optimizations
- No layer concept
- No layout concept
- Tensor language for lazy loop generation
- Polyhedral IR analysis

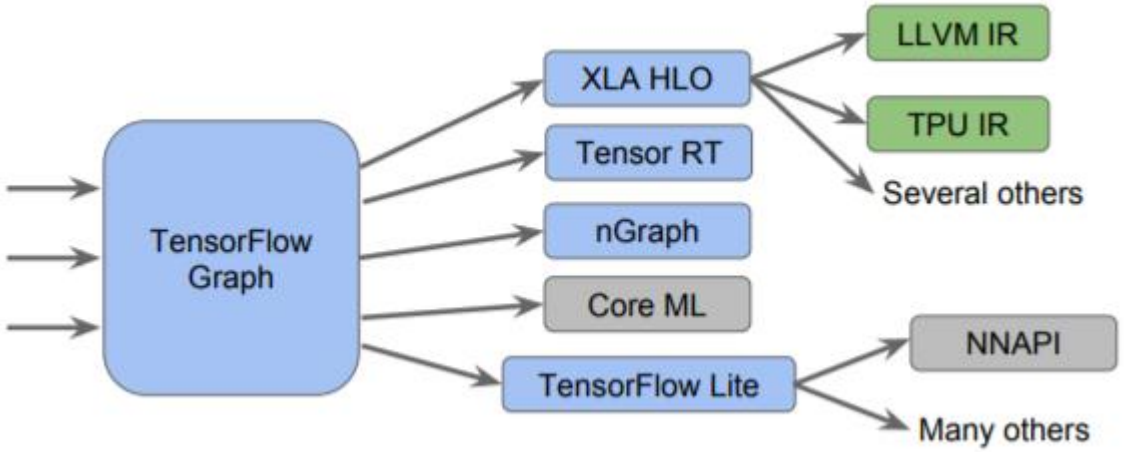
Questions?
Let’s think about the issues….

Issues

- The structure is the same, but not….
- Passes can’t be reused between compilers
- New HW ->need update all compilers

XLA (accelerated linear algebra) to MLIR

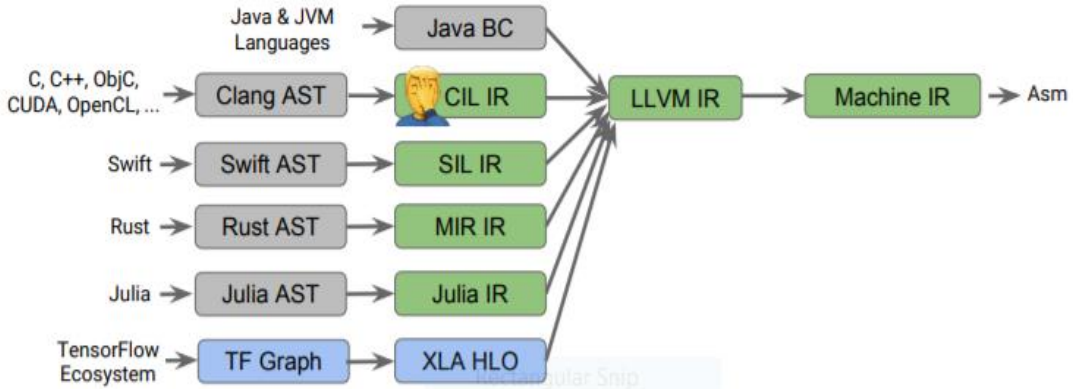
- Supplementary to TensorFlow
  1. “classic” mode: kernel based precompiled with TF
  2. XLA bridge mode: compiles TF codes to HLO for further optimization



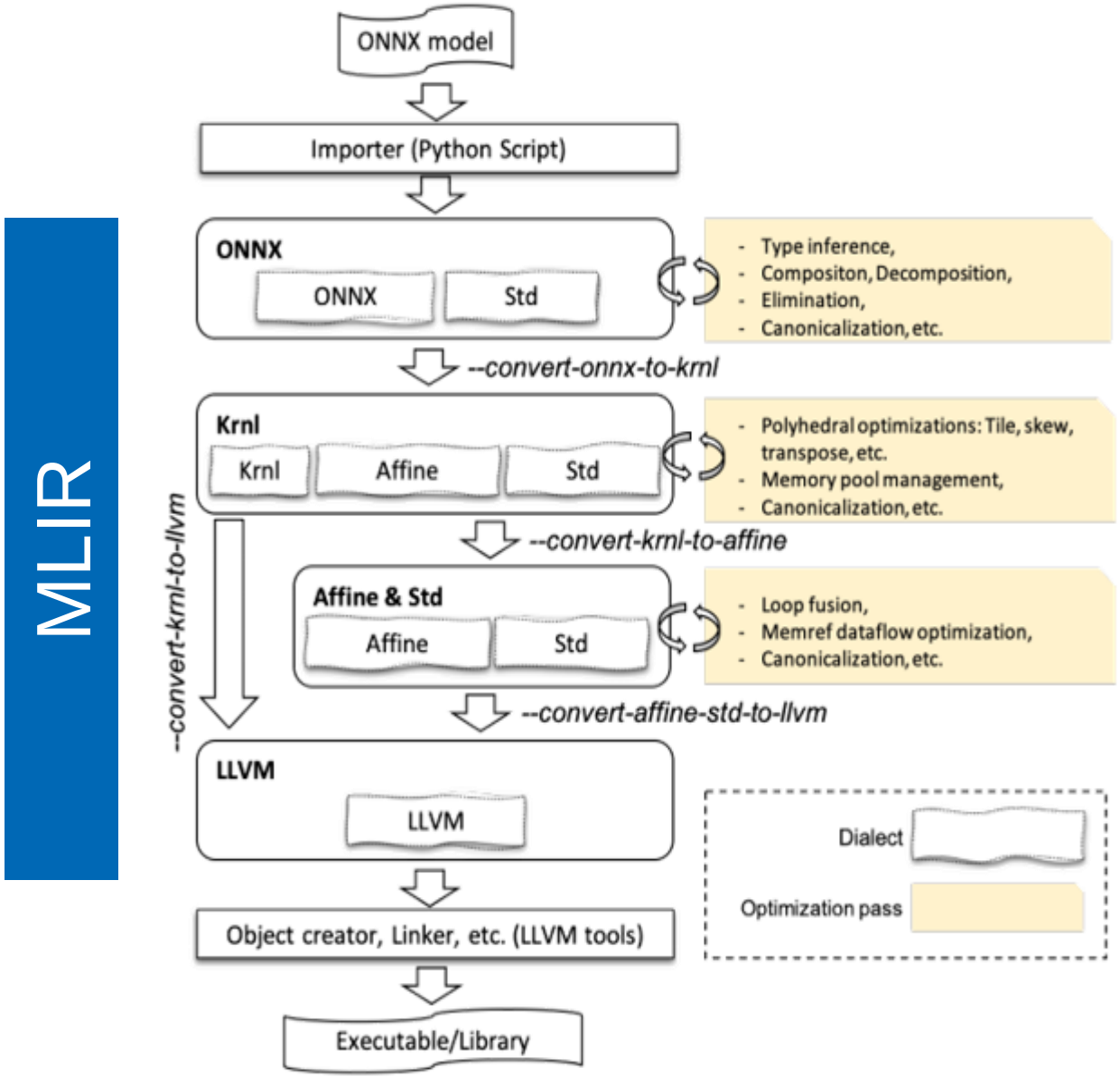
- Many “Graph” IRs, each with challenges:
- Similar-but-different proprietary technologies: not going away anytime soon
- Fragile, poor UI when failures happen: e.g. poor/no location info, or even crashes
- Duplication of infrastructure at all levels

MLIR

- Framework for compiler development
- Generic enough for various domains (not NN only)
  - For example, Fortran compiler
- Multi-Level Intermediate Representation
  - Can represent different levels of abstractions (NN layers, OpenCL code, LLVM assembler)
- Support for various techniques (polyhedral model, for example)
- Compilation performance (multi-threaded compilation included)



Flow example



Operations and Values

```
%t_tensor = "toy.transpose"(%tensor) {inplace = true} : (tensor<2x3xf64>) -> tensor<3x2xf64>
loc("example/file/path":12:1)
```

- **"toy.transpose"** – Operation
  - **toy** – Dialect namespace
  - **transpose** – Operation name
- **%tensor** – input Value (always has % as first name symbol)
- **%t_tensor** – output Value (SSA)
- **{inplace = true}** – Operation attributes dictionary
- **(tensor<2x3xf64>) -> tensor<3x2xf64>** – input/output Values Types
- **loc(...)** – Operation location information (used for extended error messages)

Tensor compiler vs Generic compiler

ML optimizing compiler

- Layout assignment
- Graph simplification
- Tensorization
- Layer fusion

Generic optimizing compiler

- Type inference
- Loop transformation
- Vectorization
- Inter procedural analysis


intel