Lecture 4: Deep Learning Compilers

CS 256: Systems and Machine Learning

Sangeetha Abdu Jyothi

UCIRVINE

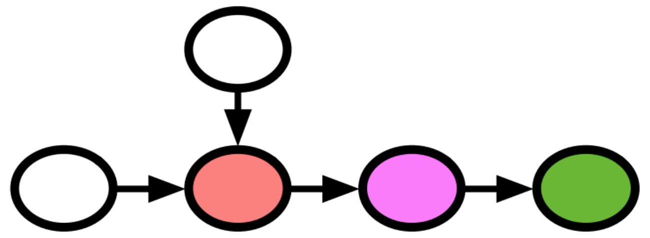
Parts of this lecture were adapted from talks at the TVM conference

Previous lectures

Deep Learning Frameworks

TensorFlow | PyTorch | mxnet | Microsoft Cognitive Toolkit | Caffe2

High-level data flow graph

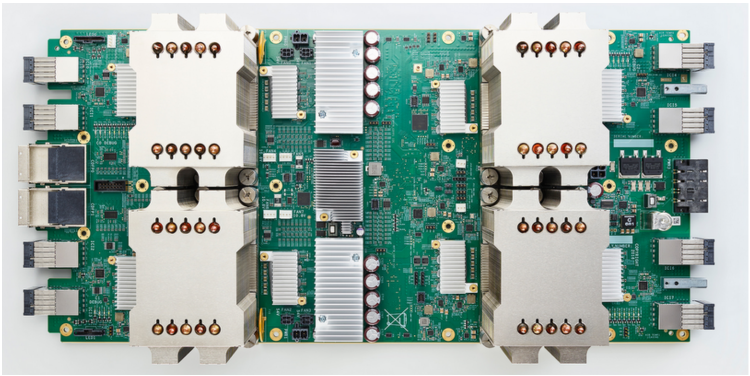


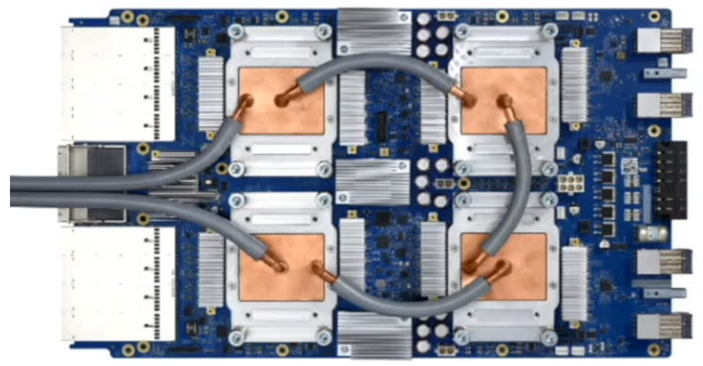
?

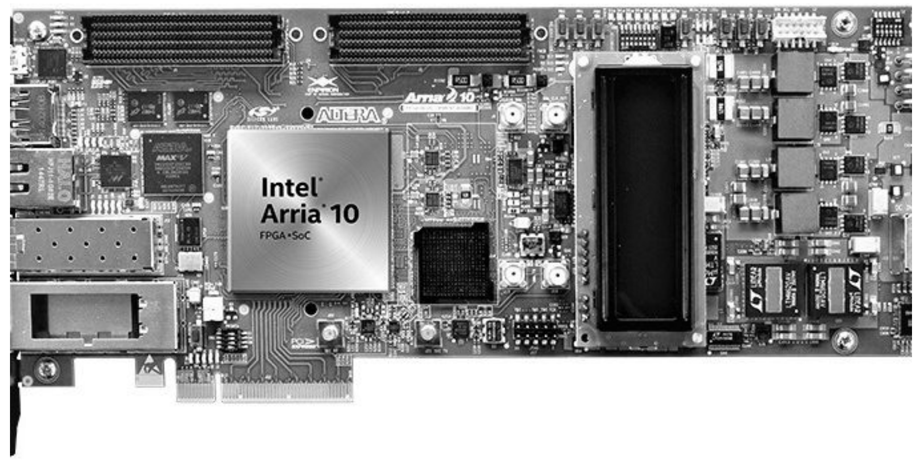
Kernel Libraries

cuDNN | NNPack | MKL-DNN

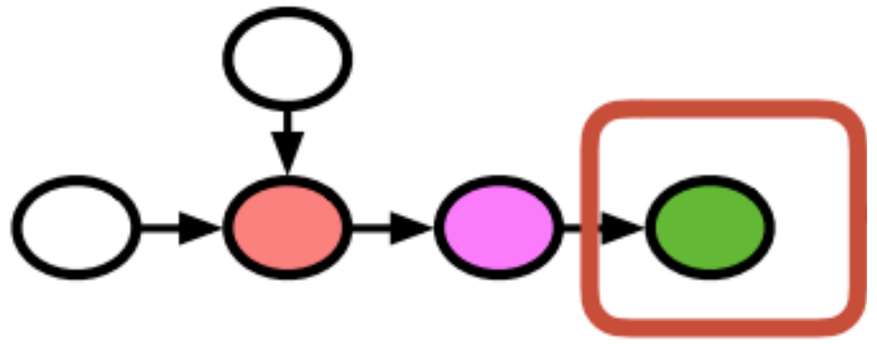
Hardware







Previous Approach: Engineer Optimized Tensor Operators

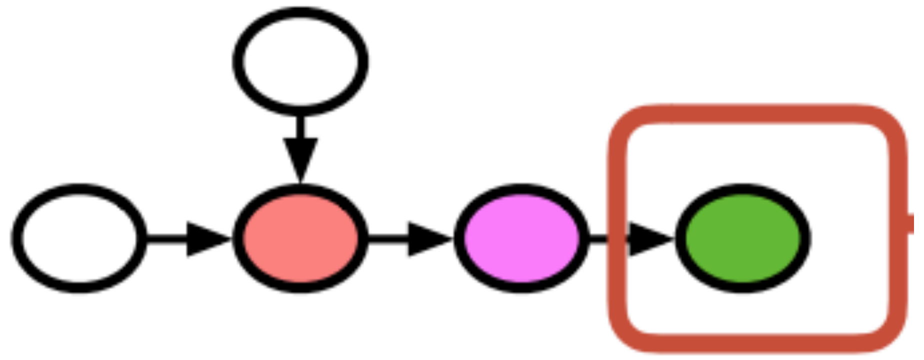


Matmul: Operator Specification

Vanilla Code

```
for y in range(1024):
  for x in range(1024):
    C[y][x] = 0
    for k in range(1024):
      C[y][x] += A[k][y] * B[k][x]
```

Previous Approach: Engineer Optimized Tensor Operators

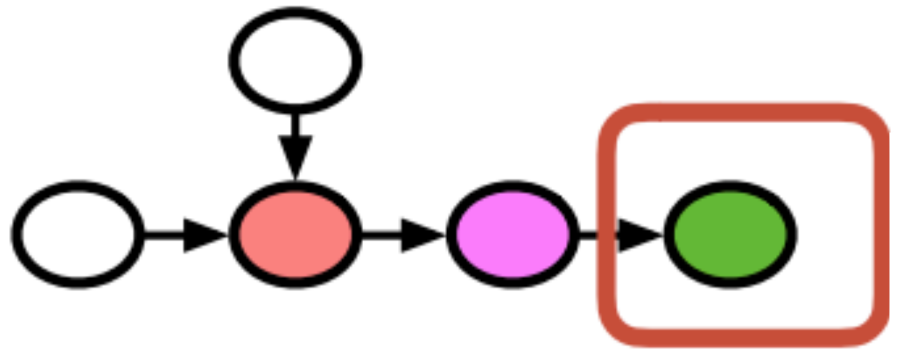


Matmul: Operator Specification

Loop Tiling for Locality

```
for yo in range(128):
  for xo in range(128):
    C[yo*8:yo*8+8][xo*8:xo*8+8] = 0
    for ko in range(128):
      for yi in range(8):
        for xi in range(8):
          for ki in range(8):
            C[yo*8+yi][xo*8+xi] +=
               A[ko*8+ki][yo*8+yi] * B[ko*8+ki][xo*8+xi]
```

Previous Approach: Engineer Optimized Tensor Operators



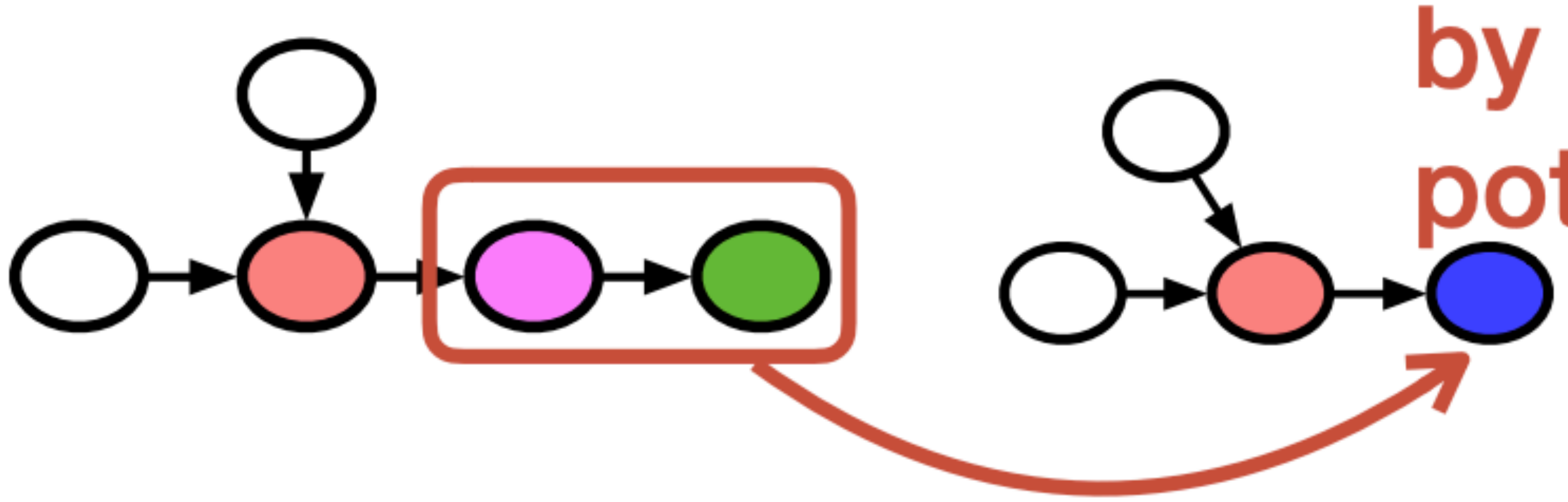
Matmul: Operator Specification

Map to Accelerators

```
inp_buffer AL[8][8], BL[8][8]
acc_buffer CL[8][8]
for yo in range(128):
  for xo in range(128):
    vdla.fill_zero(CL)
    for ko in range(128):
      vdla.dma_copy2d(AL, A[ko*8:ko*8+8][yo*8:yo*8+8])
      vdla.dma_copy2d(BL, B[ko*8:ko*8+8][xo*8:xo*8+8])
      vdla.fused_gemm8x8_add(CL, AL, BL)
    vdla.dma_copy2d(C[yo*8:yo*8+8,xo*8:xo*8+8], CL)
```

Human exploration of optimized code

Previous Approach: Cannot Leverage Operator Fusion



New operator introduced
by operator fusion optimization
potentially benefit: 1.5x speedup

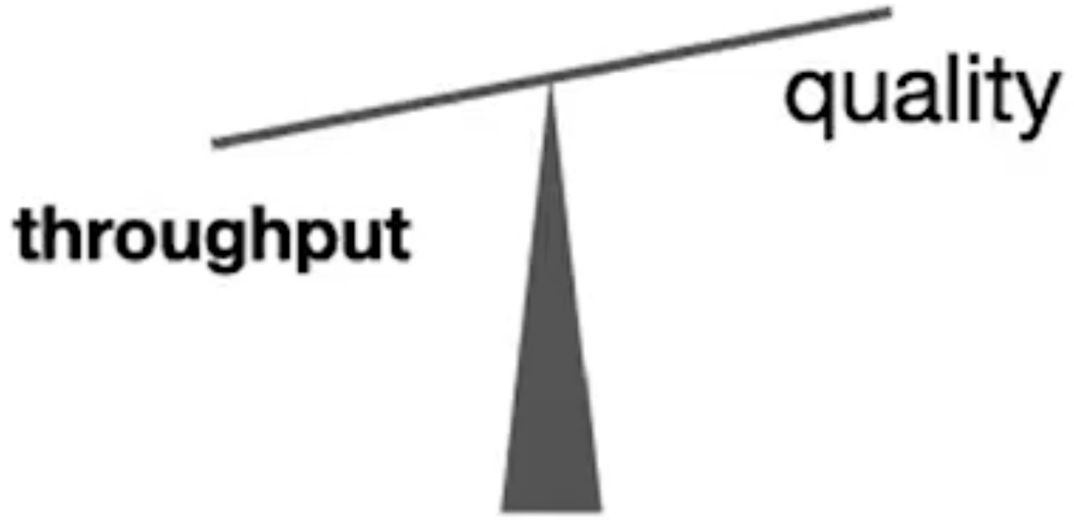
Limitations in this stack

- Every high-level operation in the computational graph requires an optimized implementation in kernel libraries
- Engineering intensive
- Cannot leverage operator fusing

Deployment Challenges

1. On what?

2. How fast / accurate?



3. Inside of what?

Python, C++, C, Rust etc.

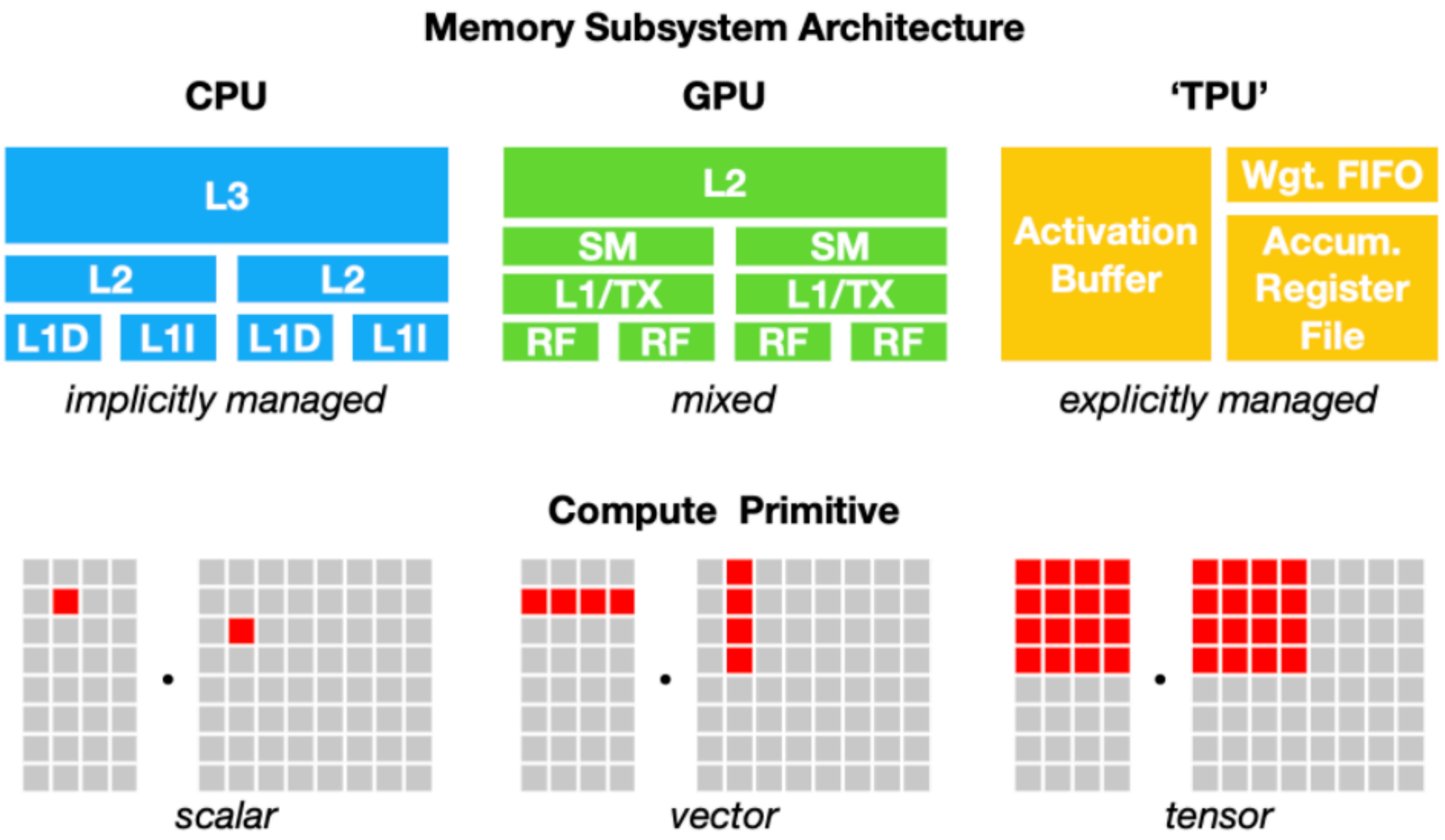
Deployment Challenges

Back End

Front End

PyTorch	?	?	?	?	?	?	?	?	?	?
Caffe2	?	?	?	?	?	?	?	?	?	?
TensorFlow	?	?	?	?	?	?	?	?	?	?
mxnet	?	?	?	?	?	?	?	?	?	?
ONNX	?	?	?	?	?	?	?	?	?	?
TensorFlow Lite	?	?	?	?	?	?	?	?	?	?

Memory Layouts and Compute Primitives

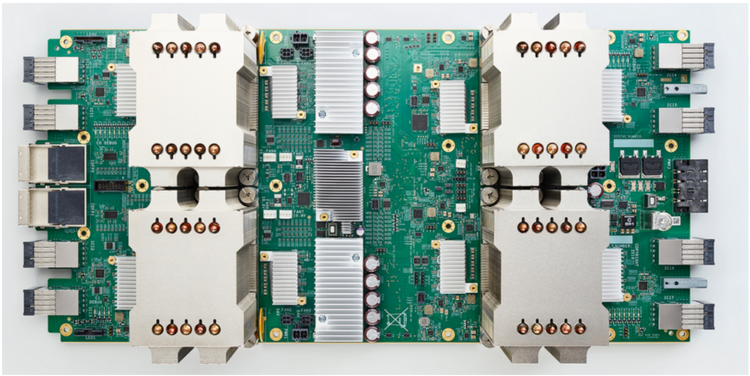


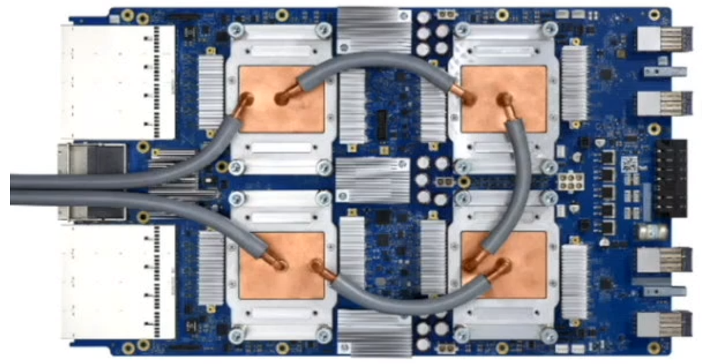
Deep Learning Stack

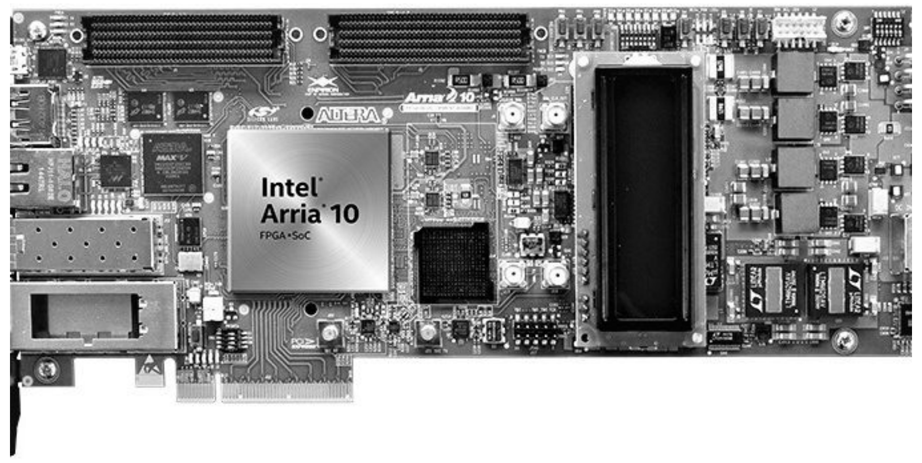
Deep Learning Frameworks

Deep Learning Compilers

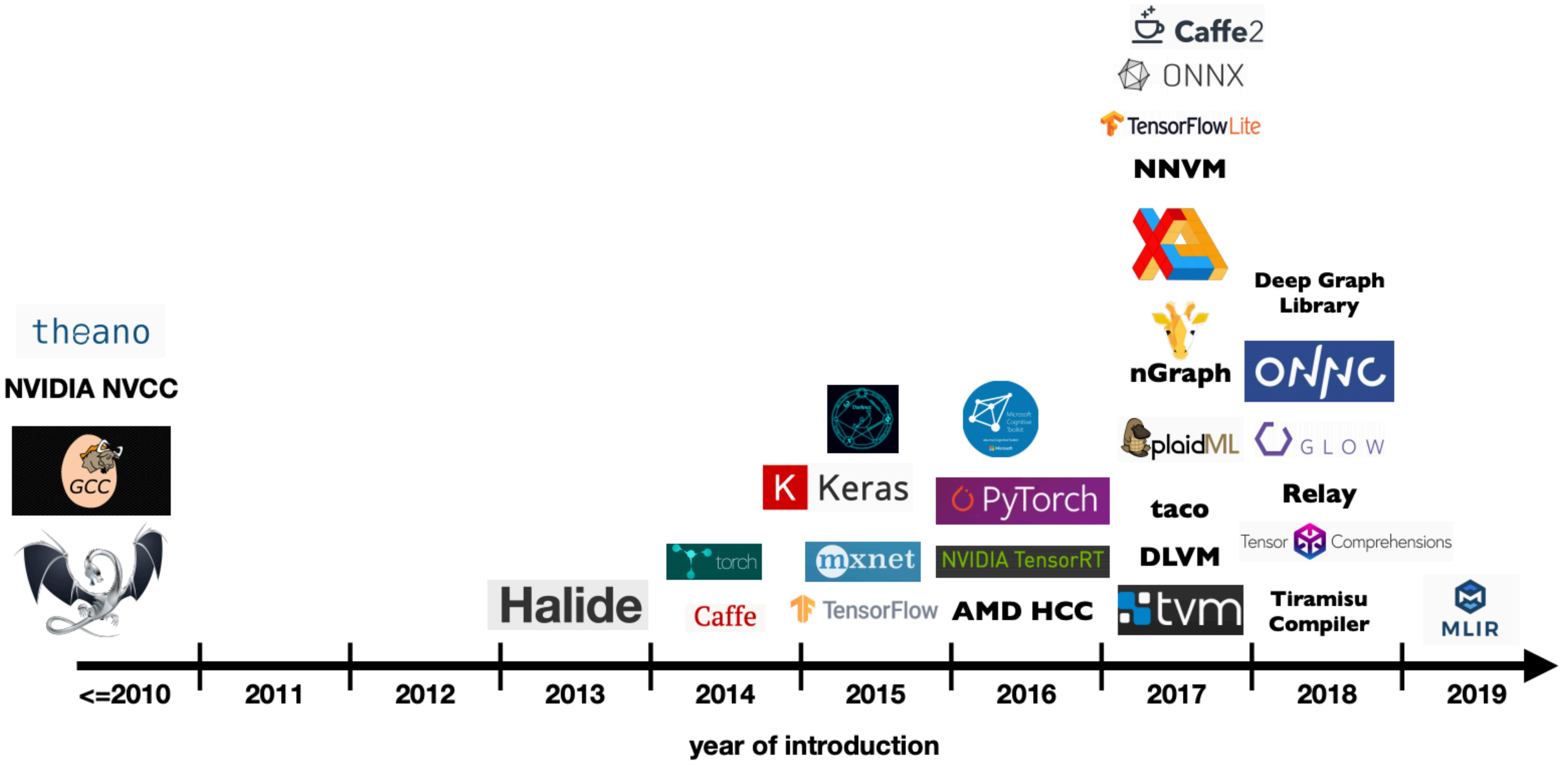
Hardware





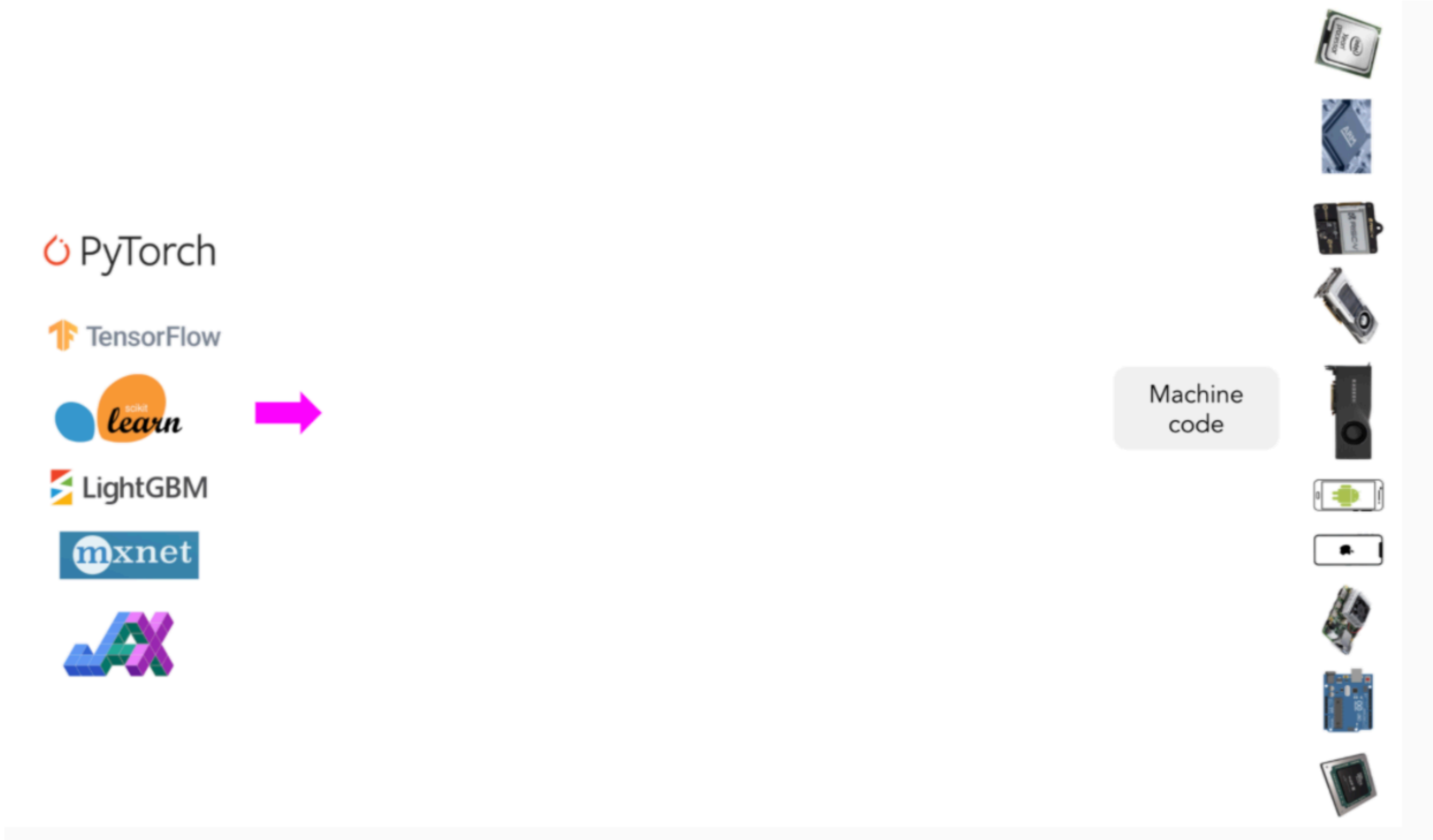


Deep Learning Stack evolution

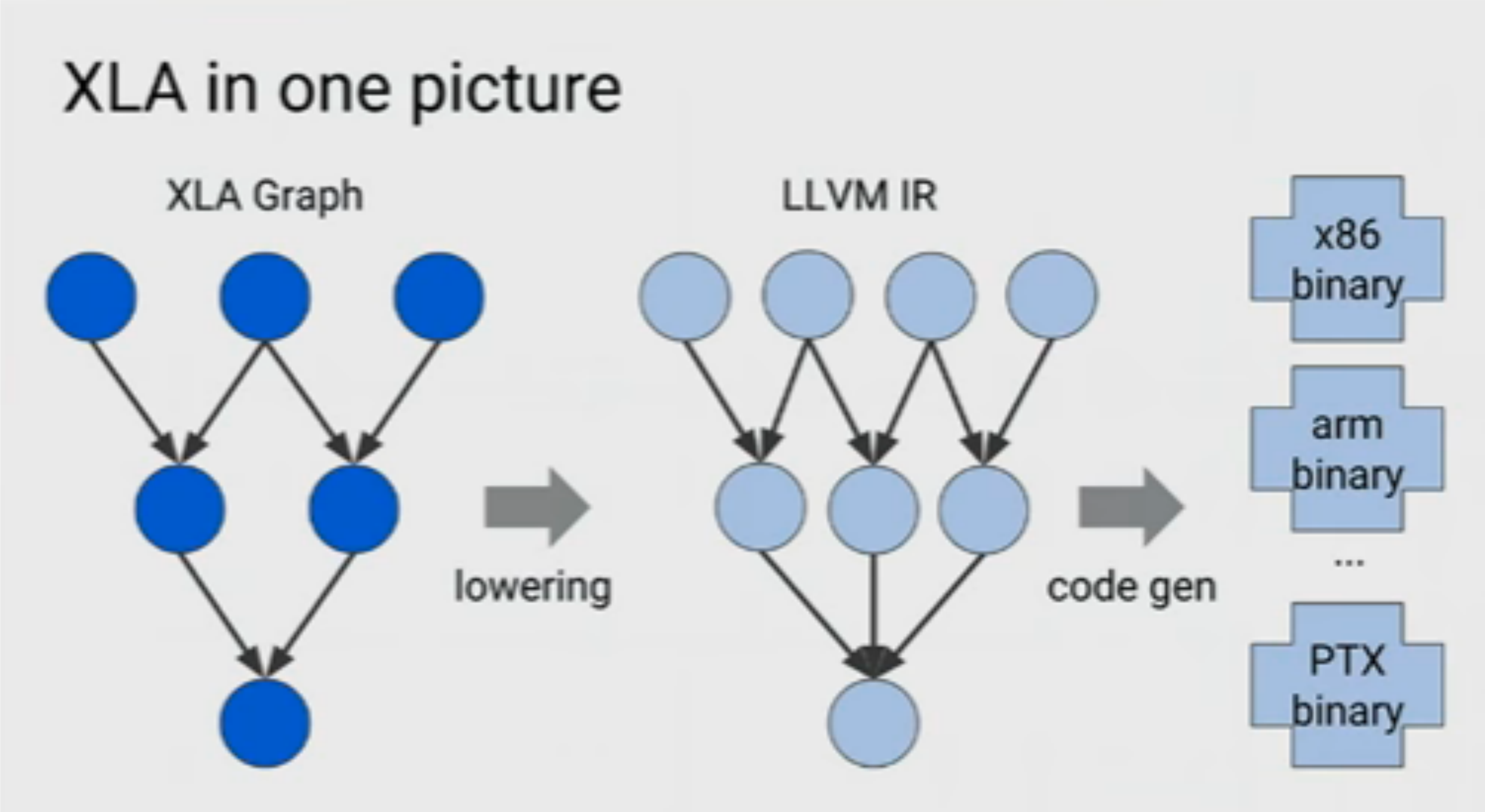


Intermediate Representation

Intermediate Representation (IR)



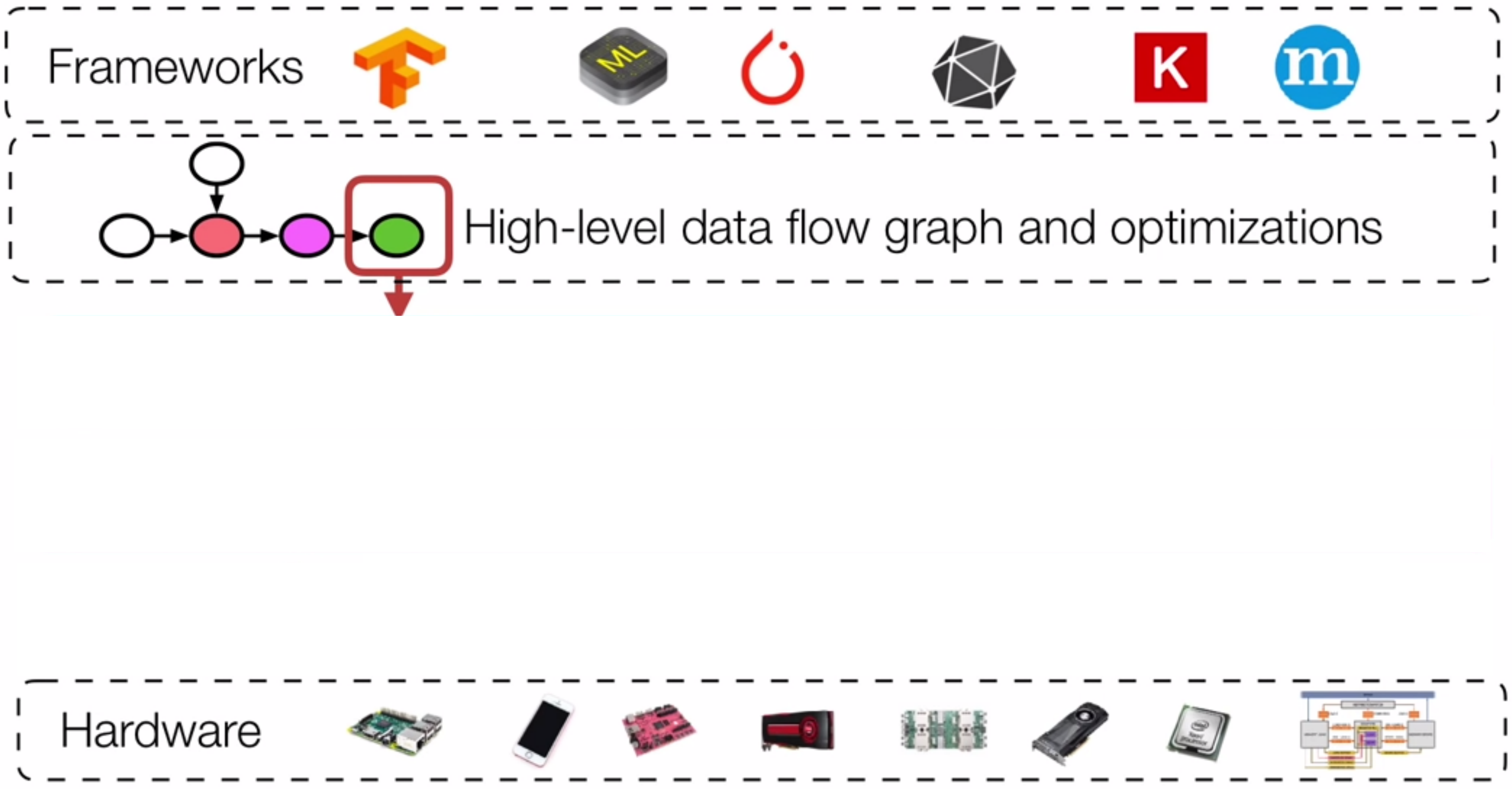
Example: XLA



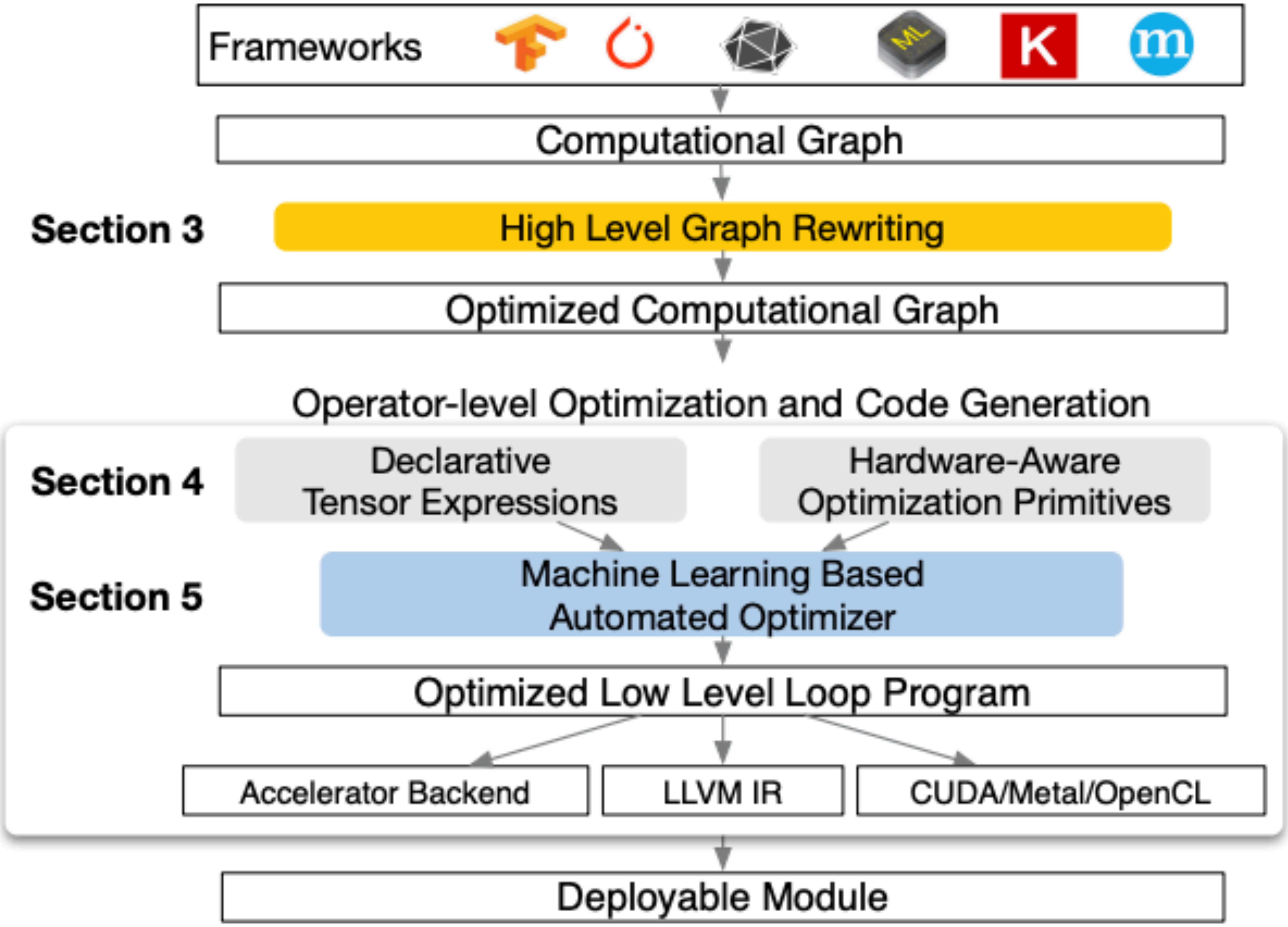
XLA in one picture
XLA Graph
LLVM IR
lowering
code gen
x86 binary
arm binary
...
PTX binary

TVM

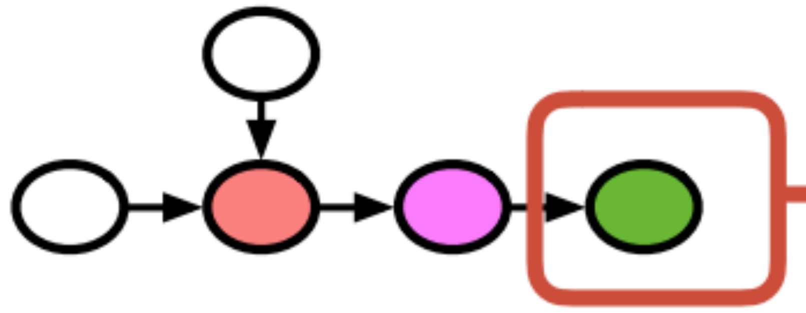
TVM: Learning Based Deep Learning Compiler



System Overview



Hardware Aware Search Space



Tensor Expression Language (Specification)

```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Define search space of hardware aware mappings from expression to hardware program

Based on Halide's compute/schedule separation

Halide Programming Model

Functional definition: what should this function do?

```
// The algorithm - no storage or order
blur_x(x, y) = (input(x-1, y) + input(x, y) + input(x+1, y))/3;
blur_y(x, y) = (blur_x(x, y-1) + blur_x(x, y) + blur_x(x, y+1))/3;
```

Halide Programming Model

Functional definition: what should this function do?

```
// The algorithm - no storage or order
blur_x(x, y) = (input(x-1, y) + input(x, y) + input(x+1, y))/3;
blur_y(x, y) = (blur_x(x, y-1) + blur_x(x, y) + blur_x(x, y+1))/3;
```

Schedule definitions: how should the function do it?

```
// The schedule - defines order, locality; implies storage
blur_y.tile(x, y, xi, yi, 256, 32)
      .vectorize(xi, 8).parallel(y);
blur_x.compute_at(blur_y, x).vectorize(x, 8);
```

Matrix Multiply Example

Tensor-Expression DSL defines the algorithm and the schedule

```
# Algorithm
k = te.reduce_axis((0, K), "k")
A = te.placeholder((M, K), name="A")
B = te.placeholder((K, N), name="B")
C = te.compute((M, N), lambda x, y: te.sum(A[x, k] * B[k, y], axis=k), name="C")

# Default schedule
s = te.create_schedule(C.op)
```

vanilla schedule

Matrix Multiply Optimized Schedule

vanilla schedule

compute tiling

split reduction axis

```
bn = 32
s = te.create_schedule(C.op)

# Blocking by loop tiling
xo, yo, xi, yi = s[C].tile(C.op.axis[0], C.op.axis[1], bn, bn)
(k,) = s[C].op.reduce_axis
ko, ki = s[C].split(k, factor=4)

# Hoist reduction domain outside the blocking loop
s[C].reorder(xo, yo, ko, ki, xi, yi)
```

Matrix Multiply Optimized Schedule

vanilla schedule

compute tiling

split reduction axis

```python
bn = 32
s = te.create_schedule(C.op)

# Blocking by loop tiling
xo, yo, xi, yi = s[C].tile(C.op.axis[0], C.op.axis[1], bn, bn)
(k,) = s[C].op.reduce_axis
ko, ki = s[C].split(k, factor=4)

# Hoist reduction domain outside the blocking loop
s[C].reorder(xo, yo, ko, ki, xi, yi)
```

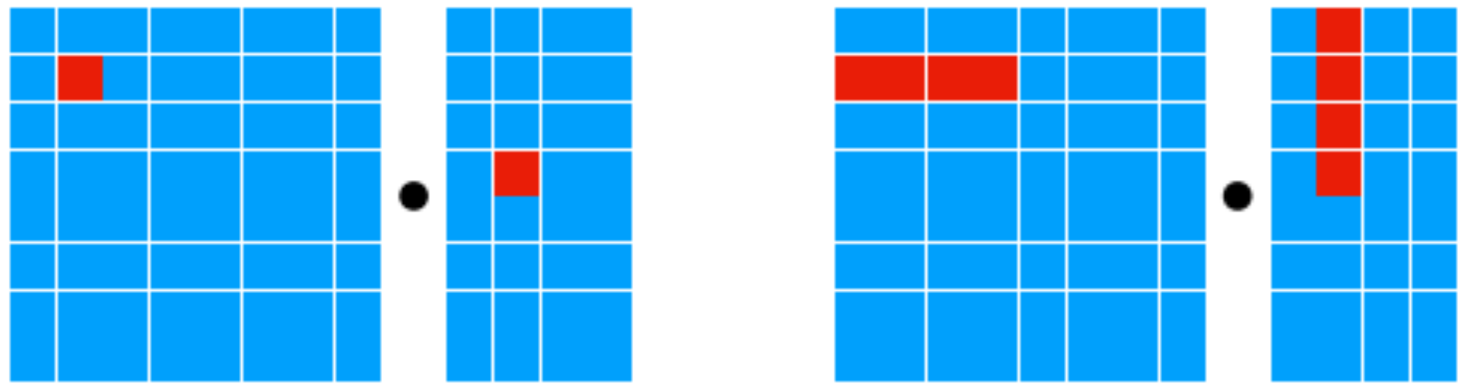
6 nested loops

```
primfn(A_1: handle, B_1: handle, C_1: handle) -> ()
  attr = {"global_symbol": "main", "tir.noalias": True}
  buffers = {C: Buffer(C_2: Pointer(float32), float32, [1024, 1024], []),
             B: Buffer(B_2: Pointer(float32), float32, [1024, 1024], []),
             A: Buffer(A_2: Pointer(float32), float32, [1024, 1024], [])}
  buffer_map = {A_1: A, B_1: B, C_1: C} {
  for (x.outer: int32, 0, 32) {
    for (y.outer: int32, 0, 32) {
      for (x.inner.init: int32, 0, 32) {
        for (y.inner.init: int32, 0, 32) {
          C_2[((((x.outer*32768) + (x.inner.init*1024)) + (y.outer*32)) + y.inner.init)] = 0f32
        }
      }
      for (k.outer: int32, 0, 256) {
        for (k.inner: int32, 0, 4) {
          for (x.inner: int32, 0, 32) {
            for (y.inner: int32, 0, 32) {
              C_2[((((x.outer*32768) + (x.inner*1024)) + (y.outer*32)) + y.inner)] = ((float32*)C_2[((((x.outer*32768) + (x.inner*1024)) + (y.outer*32)) + y.inner)] +
((float32*)A_2[((((x.outer*32768) + (x.inner*1024)) + (k.outer*4)) + k.inner)]*(float32*)B_2[((((k.outer*4096) + (k.inner*1024)) + (y.outer*32)) + y.inner)]))
            }
          }
        }
      }
    }
  }
}
```

Hardware Aware Search Space: CPUs

CPUs

Compute Primitives



scalar *vector*

Memory Subsystem

implicitly managed

Loop Transformations

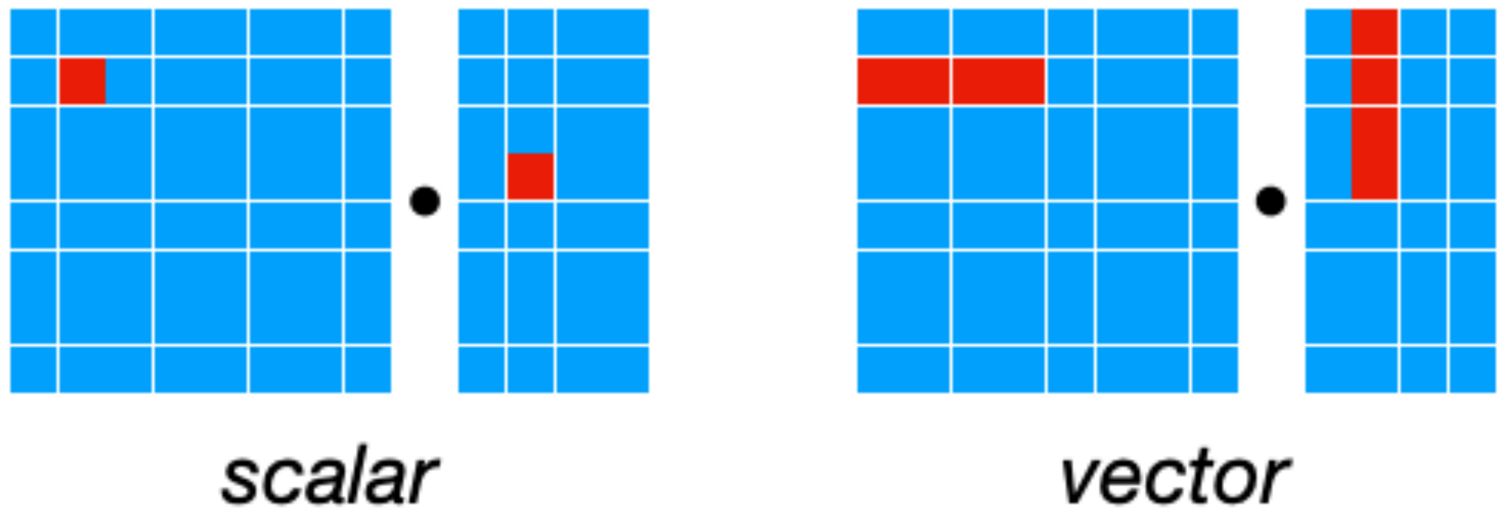
Cache Locality

Vectorization

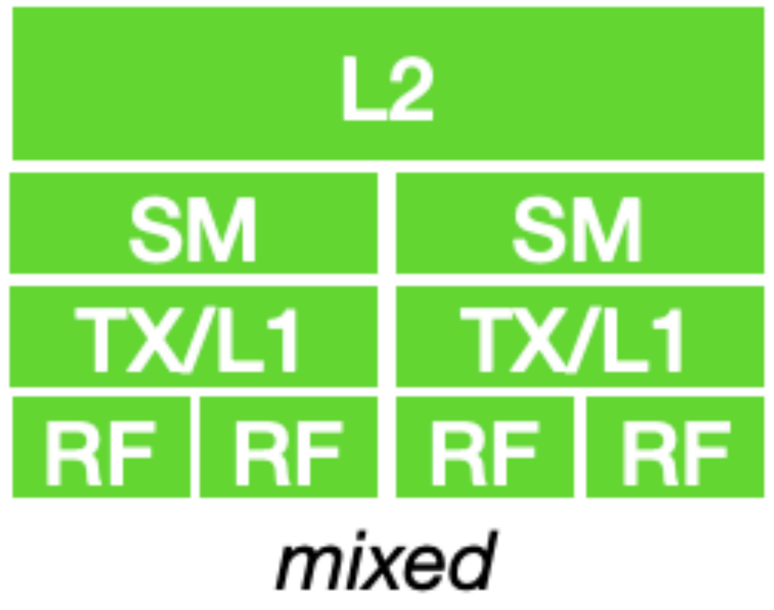
Hardware Aware Search Space: GPUs

GPUs

Compute Primitives



Memory Subsystem

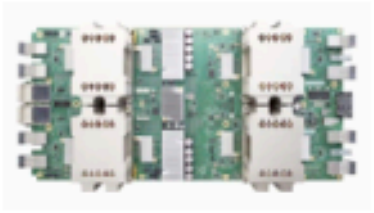


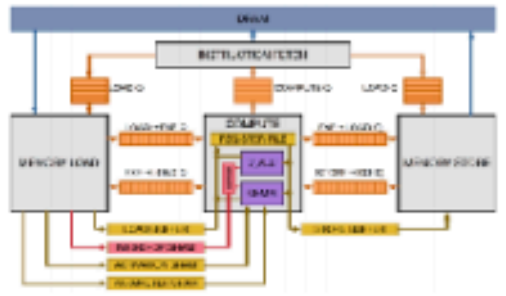
Use of Shared Memory

Thread Cooperation

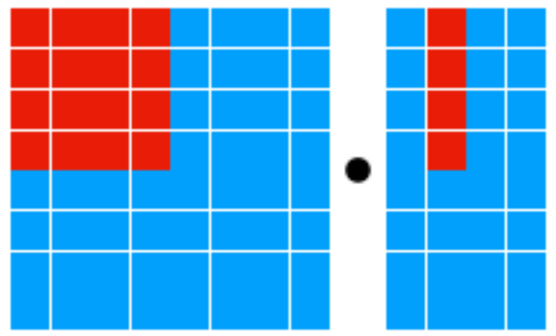
Hardware Aware Search Space: TPUs

TPU-like Specialized Accelerators





Compute Primitives



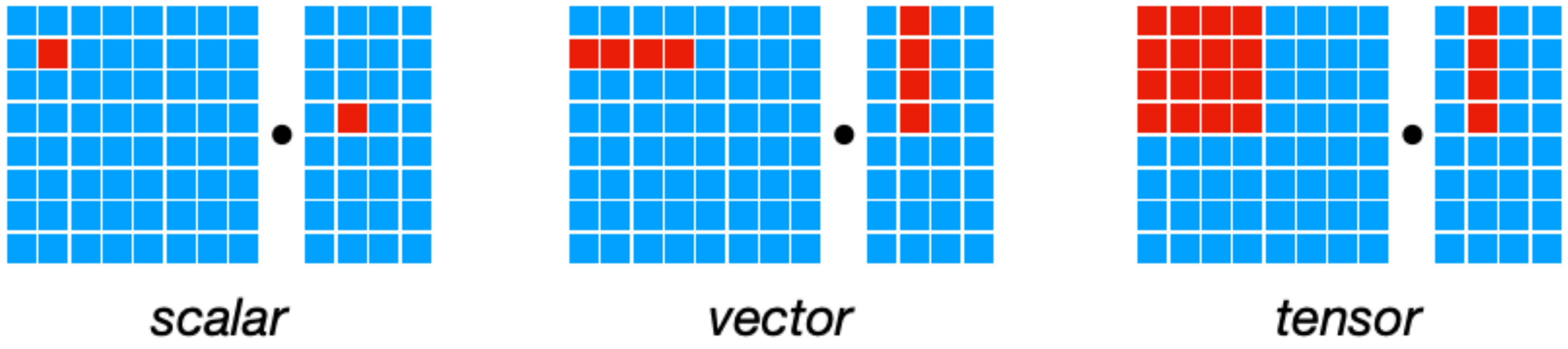
tensor

Memory Subsystem

explicitly managed

Tensorization Challenge: TPUs

Compute primitives



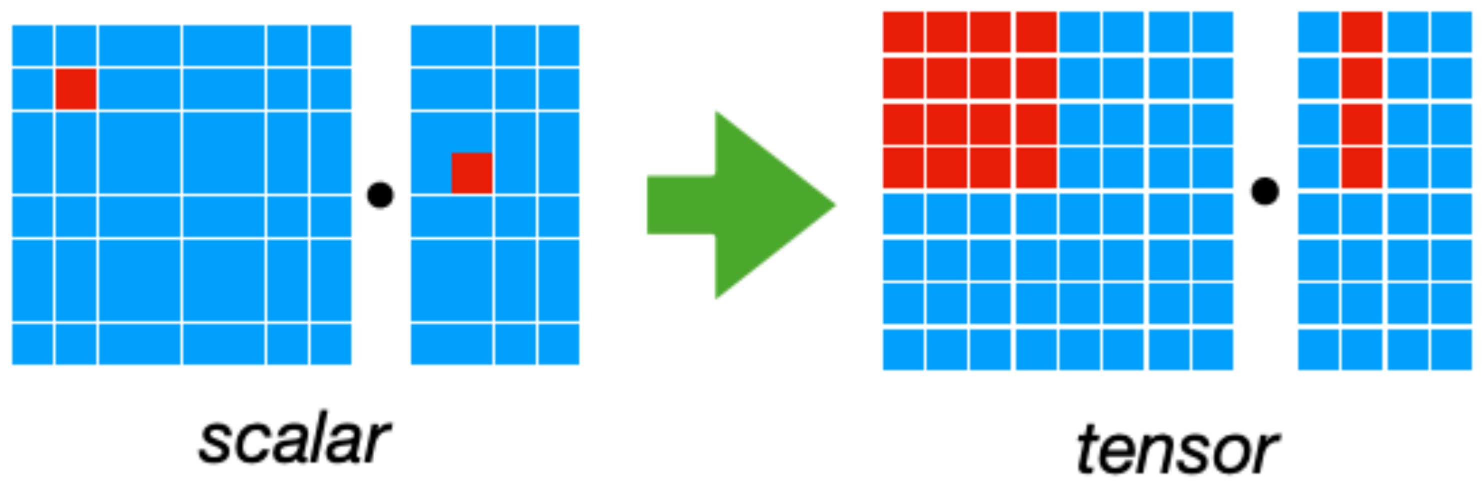
Hardware designer: declare tensor instruction interface with Tensor Expression

```
w, x = t.placeholder((8, 8)), t.placeholder((8, 8))
k = t.reduce_axis((0, 8))
y = t.compute((8, 8), lambda i, j:
                t.sum(w[i, k] * x[j, k], axis=k))

def gemm_intrin_lower(inputs, outputs):
   ww_ptr = inputs[0].access_ptr("r")
   xx_ptr = inputs[1].access_ptr("r")
   zz_ptr = outputs[0].access_ptr("w")
   compute = t.hardware_intrin("gemm8x8", ww_ptr, xx_ptr, zz_ptr)
   reset = t.hardware_intrin("fill_zero", zz_ptr)
   update = t.hardware_intrin("fuse_gemm8x8_add", ww_ptr, xx_ptr, zz_ptr)
   return compute, reset, update

gemm8x8 = t.decl_tensor_intrin(y.op, gemm_intrin_lower)
```

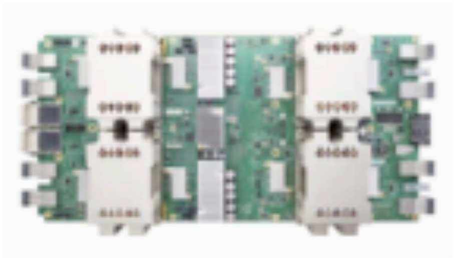
declare behavior

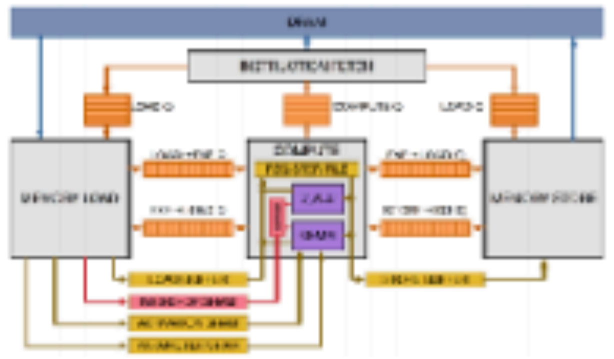
lowering rule to generate hardware intrinsics to carry out the computation

Tensorize: transform program to use tensor instructions

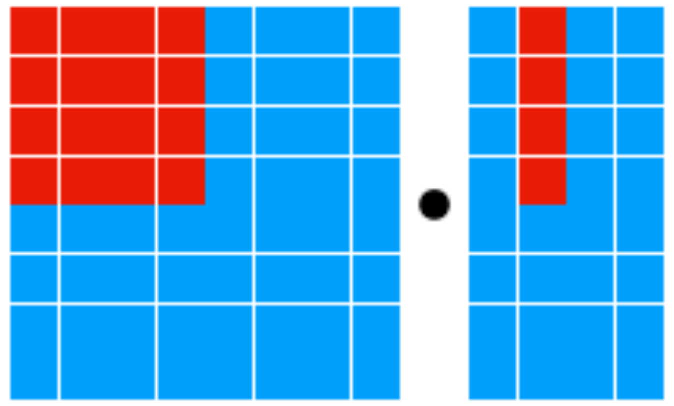


TPU-like Specialized Accelerators





Compute Primitives

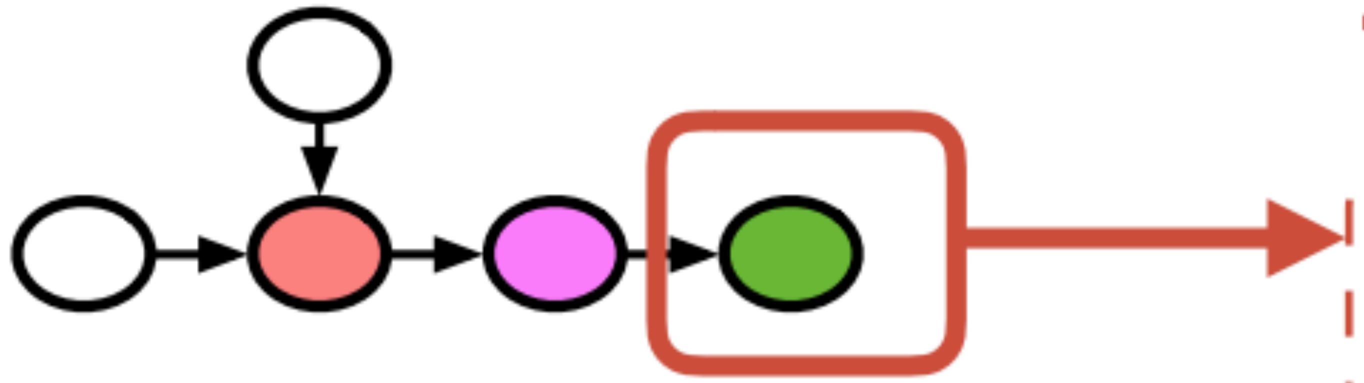


tensor

Memory Subsystem

explicitly managed

Hardware Aware Search Space



Tensor Expression Language

```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Primitives in prior work: Halide, Loopy

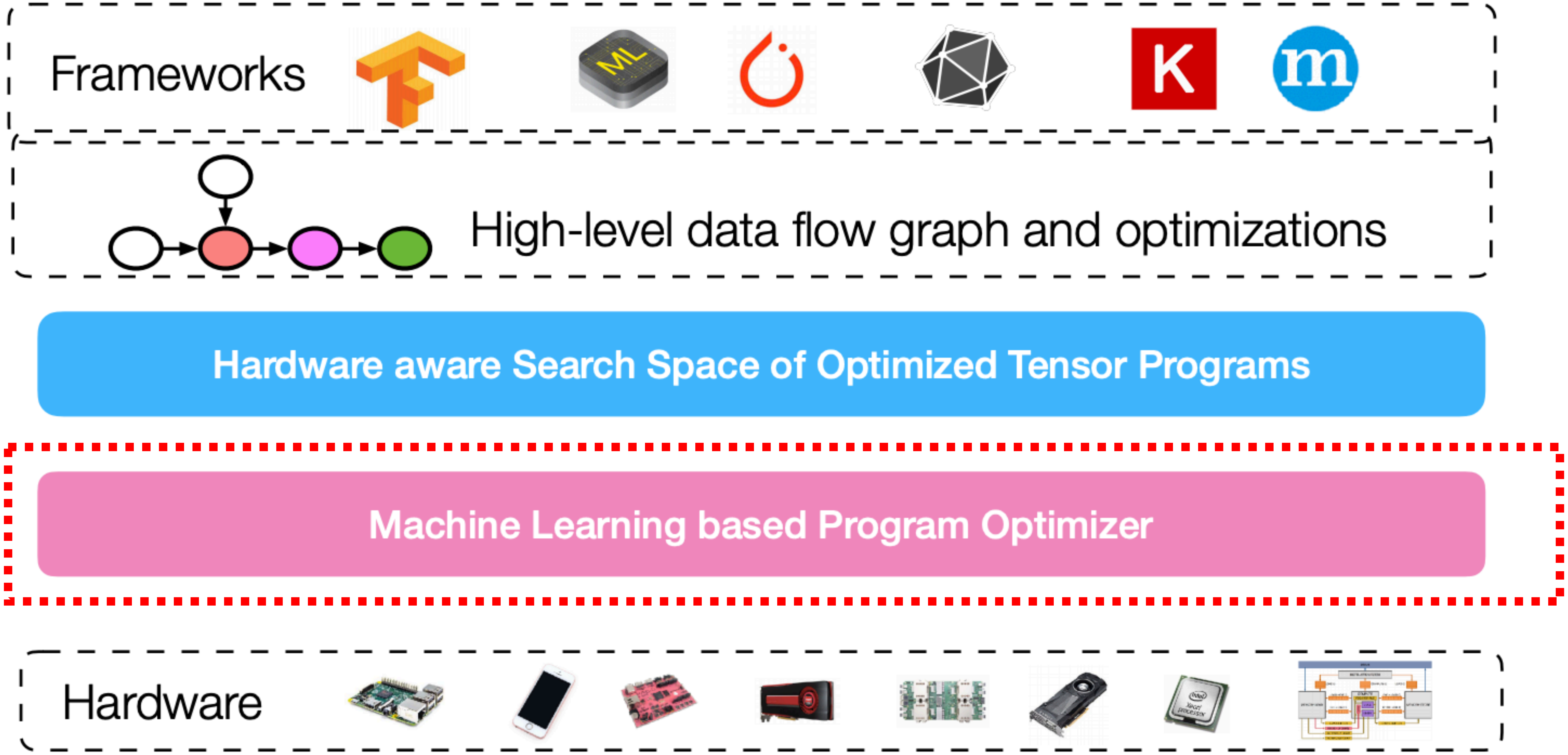
- Loop Transformations
- Thread Bindings
- Cache Locality

New primitives for GPUs, and enable TPU-like Accelerators

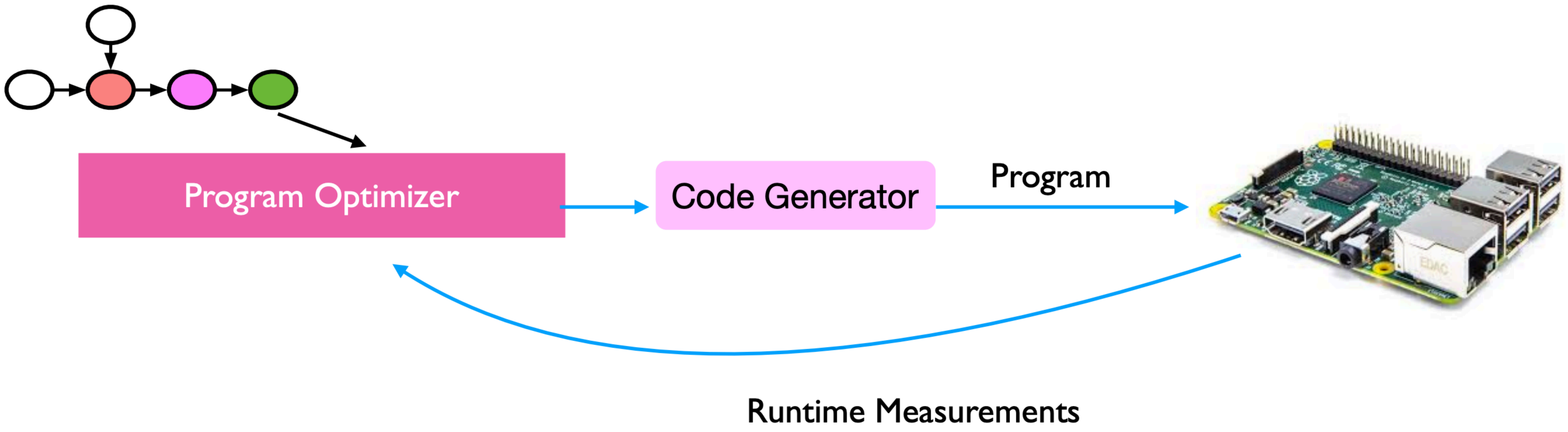
- Thread Cooperation
- Tensorization
- Latency Hiding
- ...

Hardware

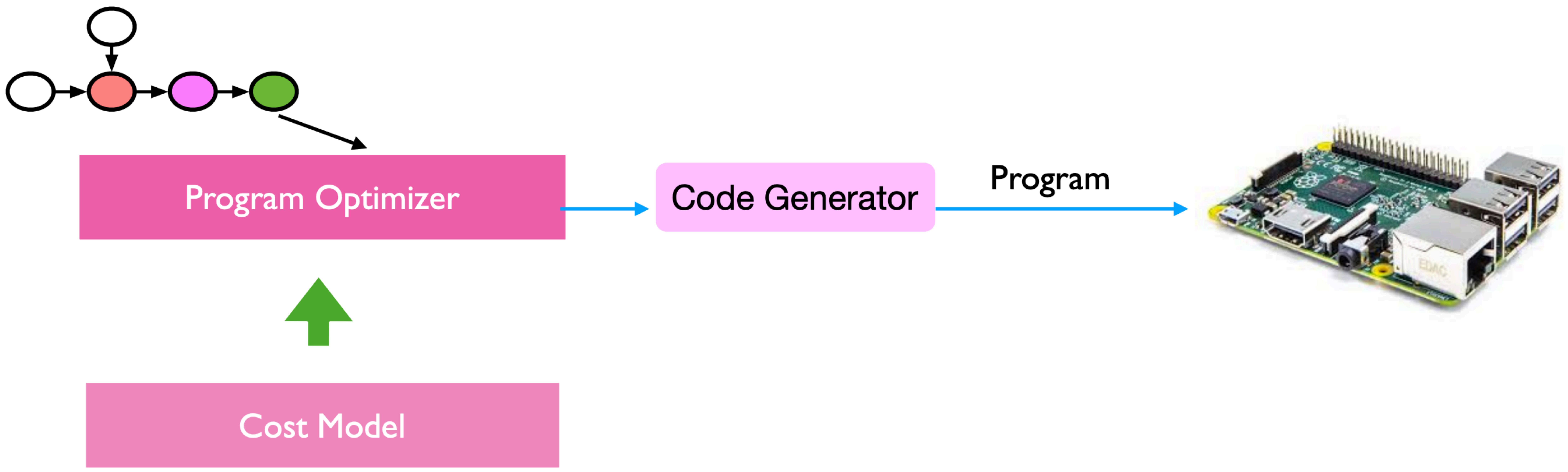
Learning Based Learning System



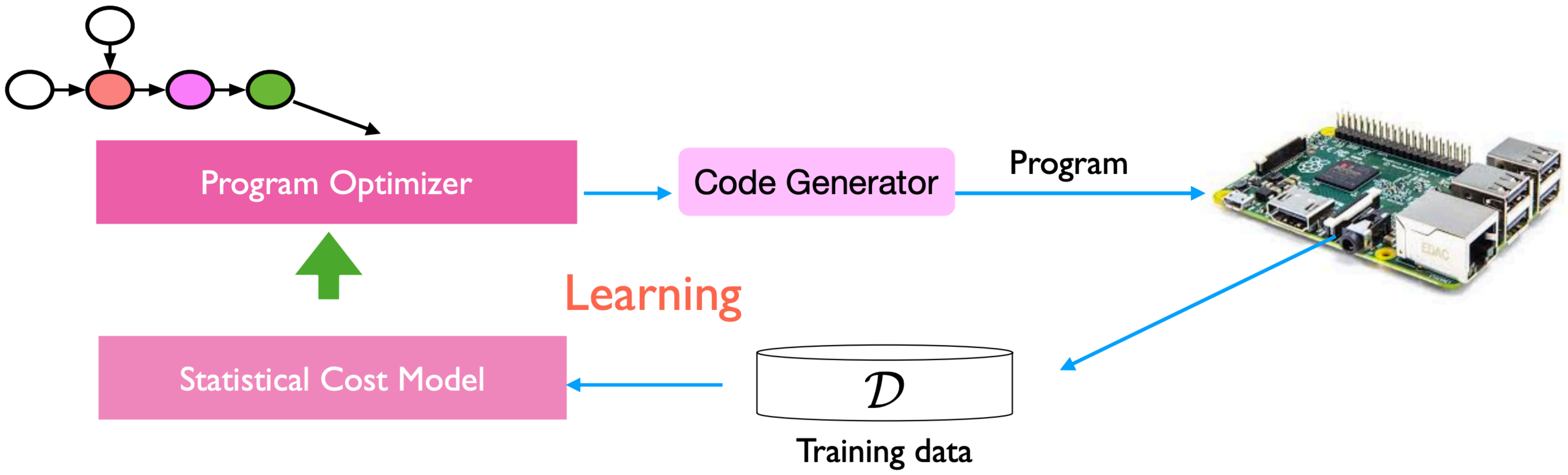
Program Optimizer Vanilla Approach



Cost-based Program Optimizer



Learning Based Program Optimizer



Program Aware Cost Modeling

High-Level Configuration

Program Aware Cost Modeling

High-Level Configuration

```
for y in range(8):
  for x in range(8):
    C[y][x]=0
    for k in range(8):
      C[y][x]+=A[k][y]*B[k][x]
```

Low-level Abstract Syntax Tree (shared between tasks)

Program Aware Cost Modeling

High-Level Configuration

```
for y in range(8):
  for x in range(8):
    C[y][x]=0
    for k in range(8):
      C[y][x]+=A[k][y]*B[k][x]
```

Low-level Abstract Syntax Tree
(shared between tasks)

touched memory

	C	A	B
y	64	64	64
x	8	8	64
k	1	8	8

outer loop length

y	1
x	8
k	64

statistical features

Boosted Tree Ensembles

Program Aware Cost Modeling

High-Level Configuration

```
for y in range(8):
  for x in range(8):
    C[y][x]=0
    for k in range(8):
      C[y][x]+=A[k][y]*B[k][x]
```

Low-level Abstract Syntax Tree
(shared between tasks)

touched memory

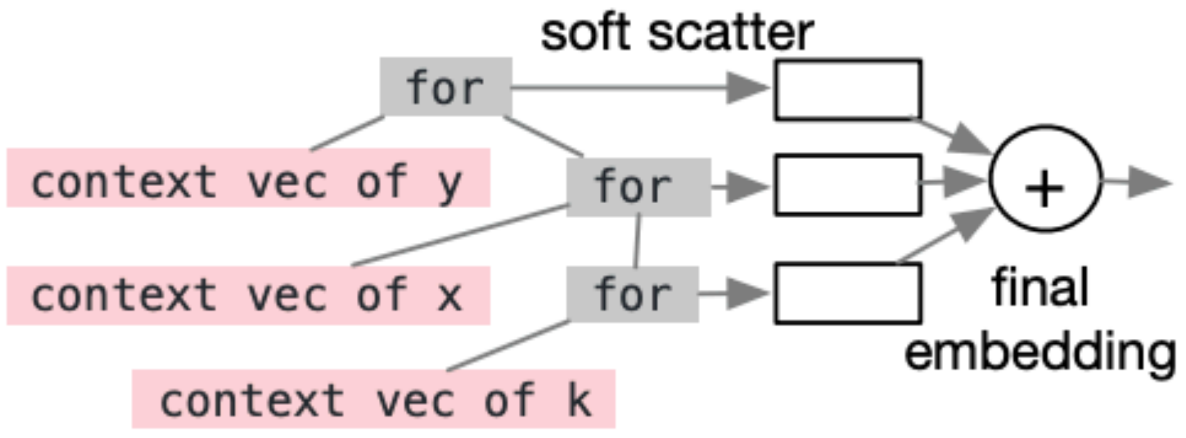
	C	A	B
y	64	64	64
x	8	8	64
k	1	8	8

outer loop length

y	1
x	8
k	64

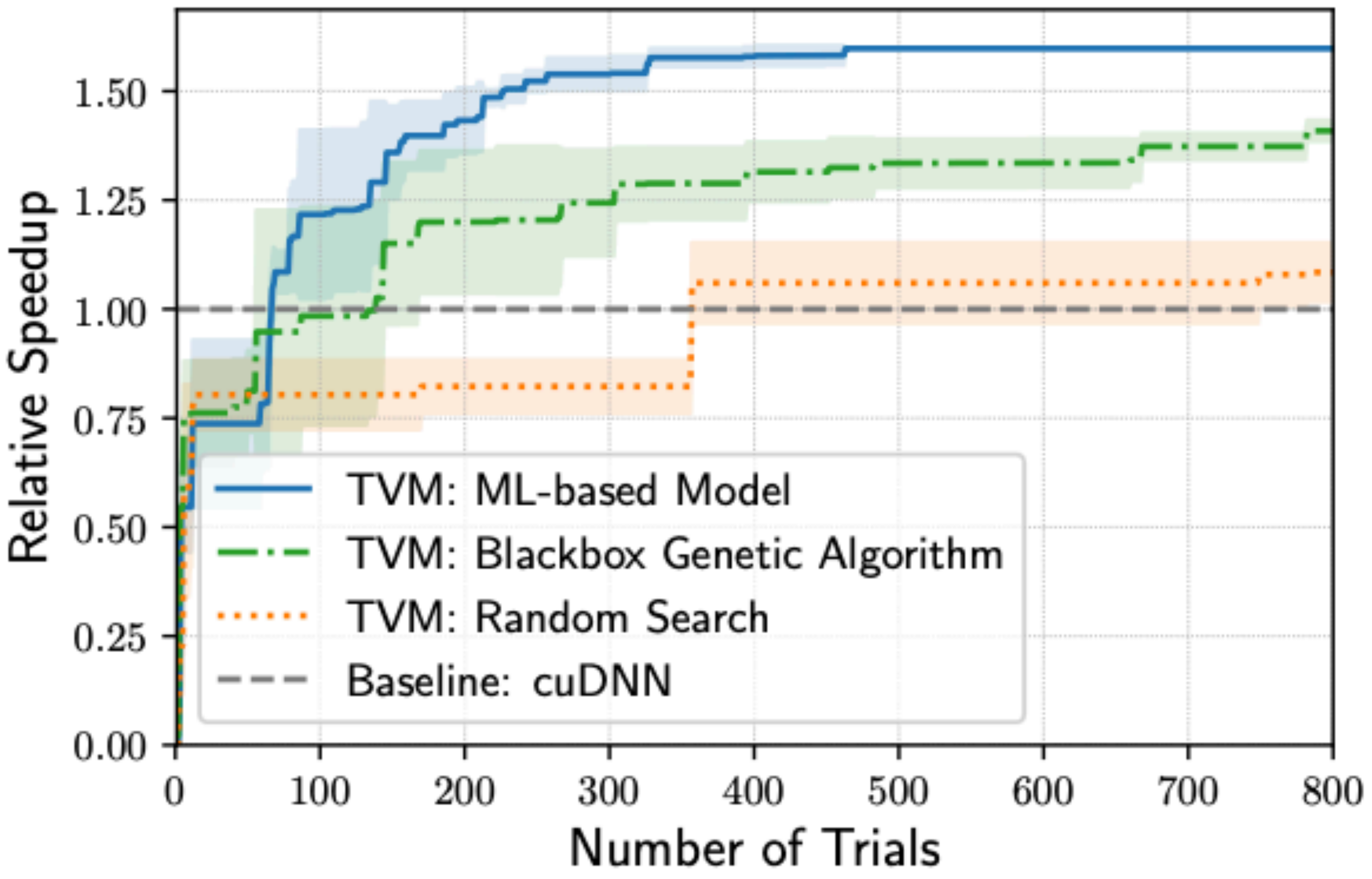
statistical features

Boosted Tree Ensembles

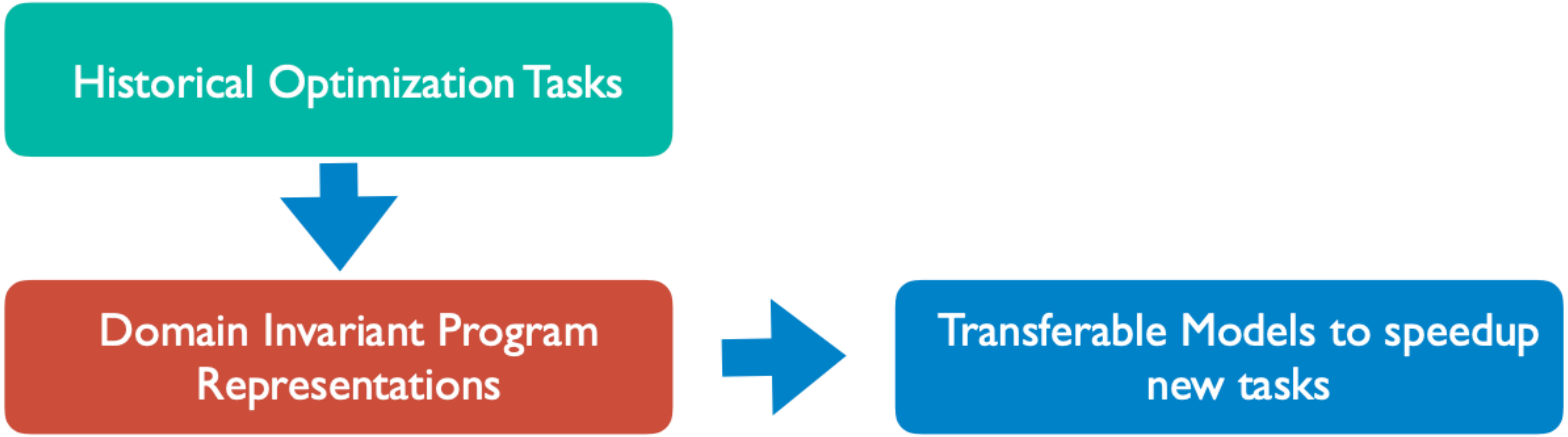


TreeGRU

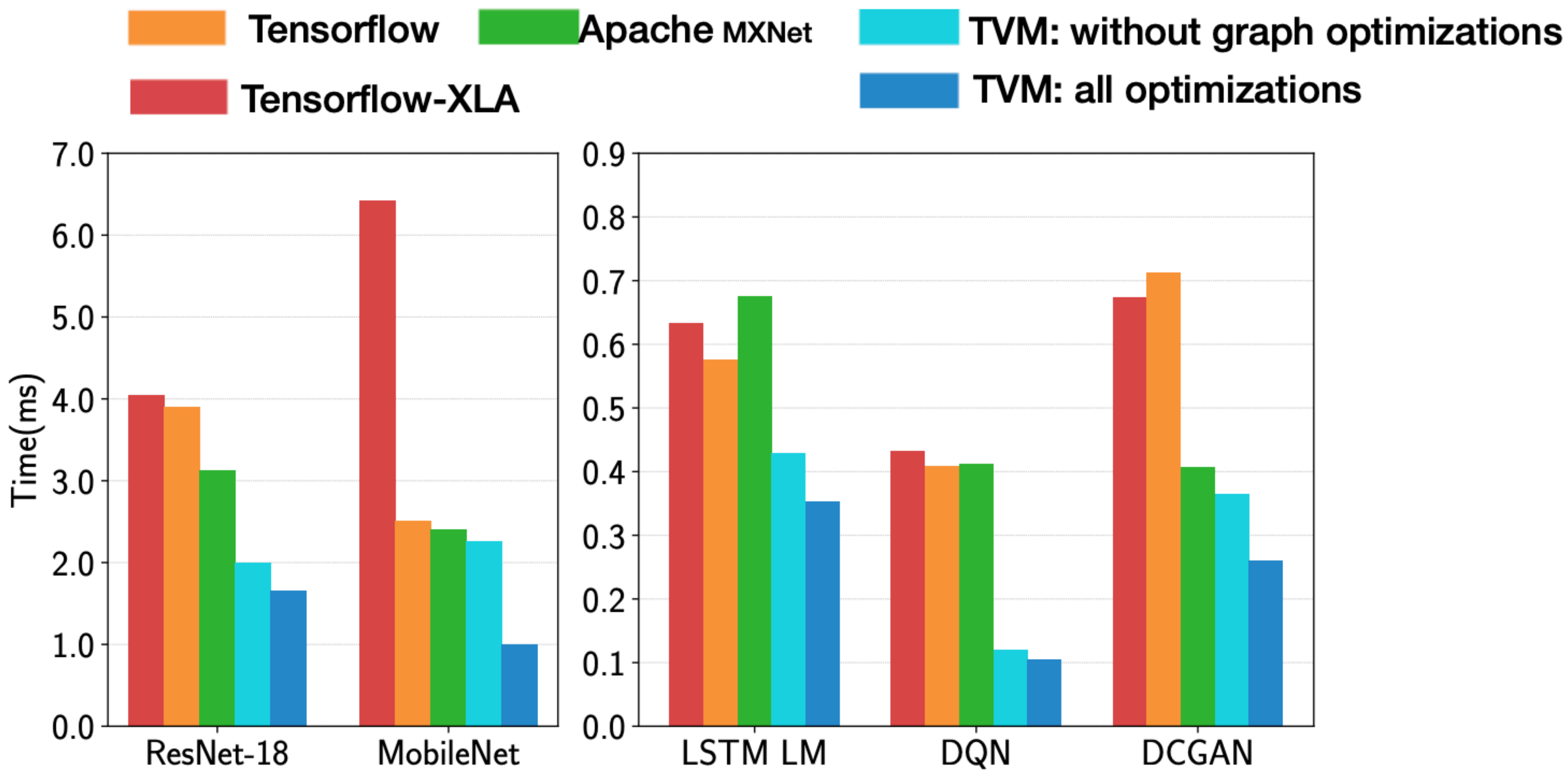
Effectiveness of ML Based Model



Transfer Learning Among Different Workloads

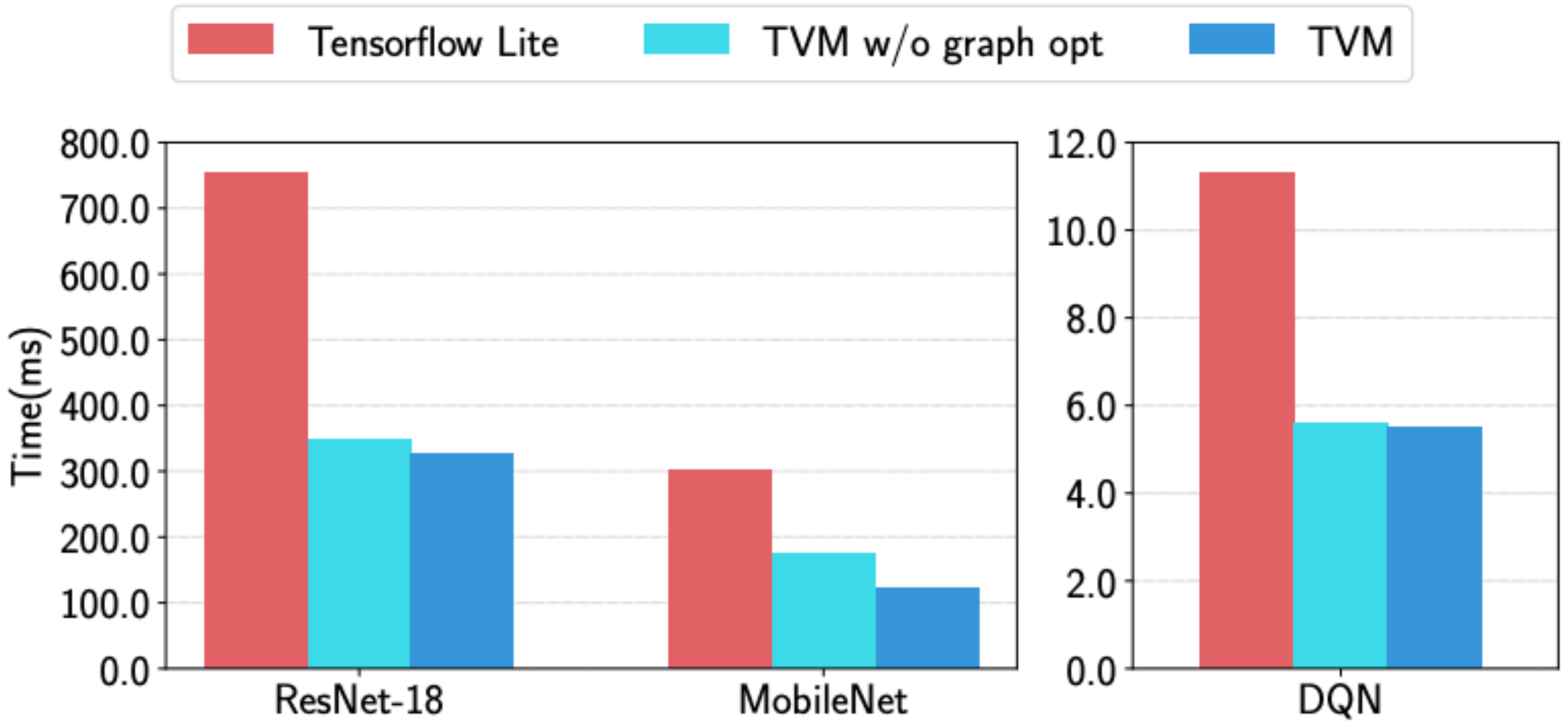


End to End Inference Performance

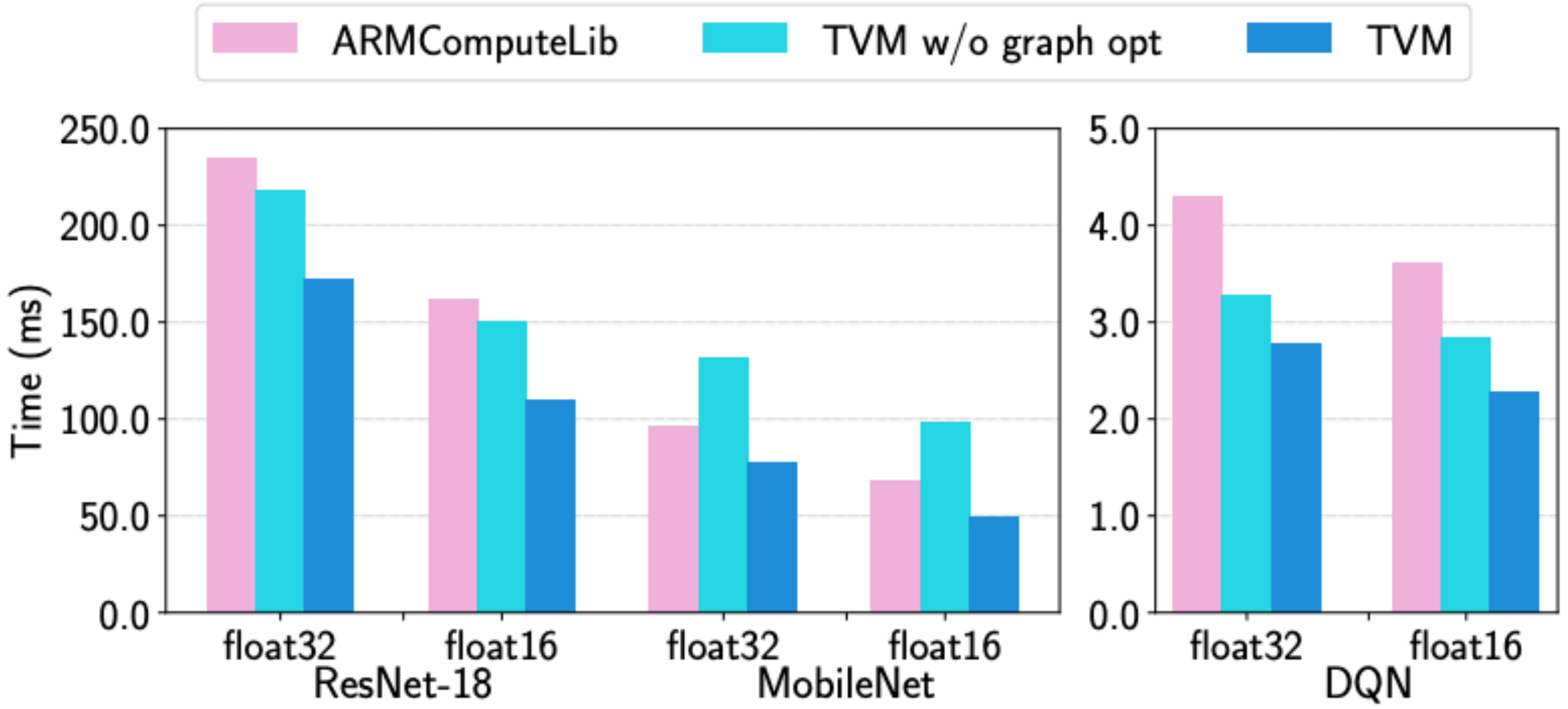


Performance Across Hardware Platforms

ARM CPU(A53)



ARM GPU(MALI)



TVM: What problems does TVM address?

Portability:

- When there are limited hardware options to deploy your model

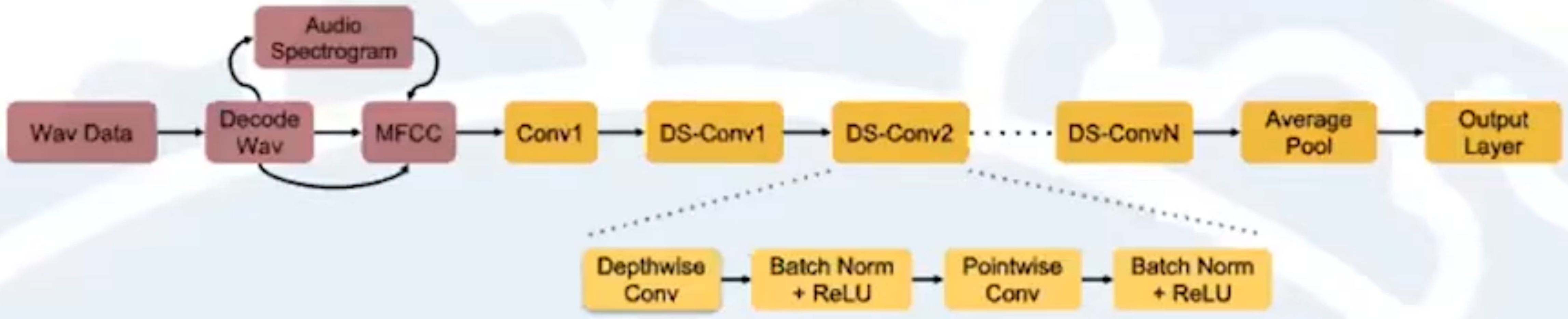
Efficiency:

- When you need to squeeze as much efficiency out of your target platform

Software support:

- When you need to build a software stack for your hardware system

TVM Use Cases in Real-World: Portability



Keyword spotting DS-CNN model architecture from TF

AzureSphere: Microsoft's secure edge IoT device

limited SRAM, bare bones operating system, no C++ support, no dynamic linking …

Deploying deep learning on AzureSphere using TVM, Hessar et al., Blog post

TVM Use Cases in Real-World: Efficiency

Workload: WaveRNN style model architecture

- Compute dominated by GRU and FC layers
- 24kHz sampling frequency requires 40us inference net runtime
- Initial model runs in PT with 3,400us inference net runtime
- **85x slower than target**

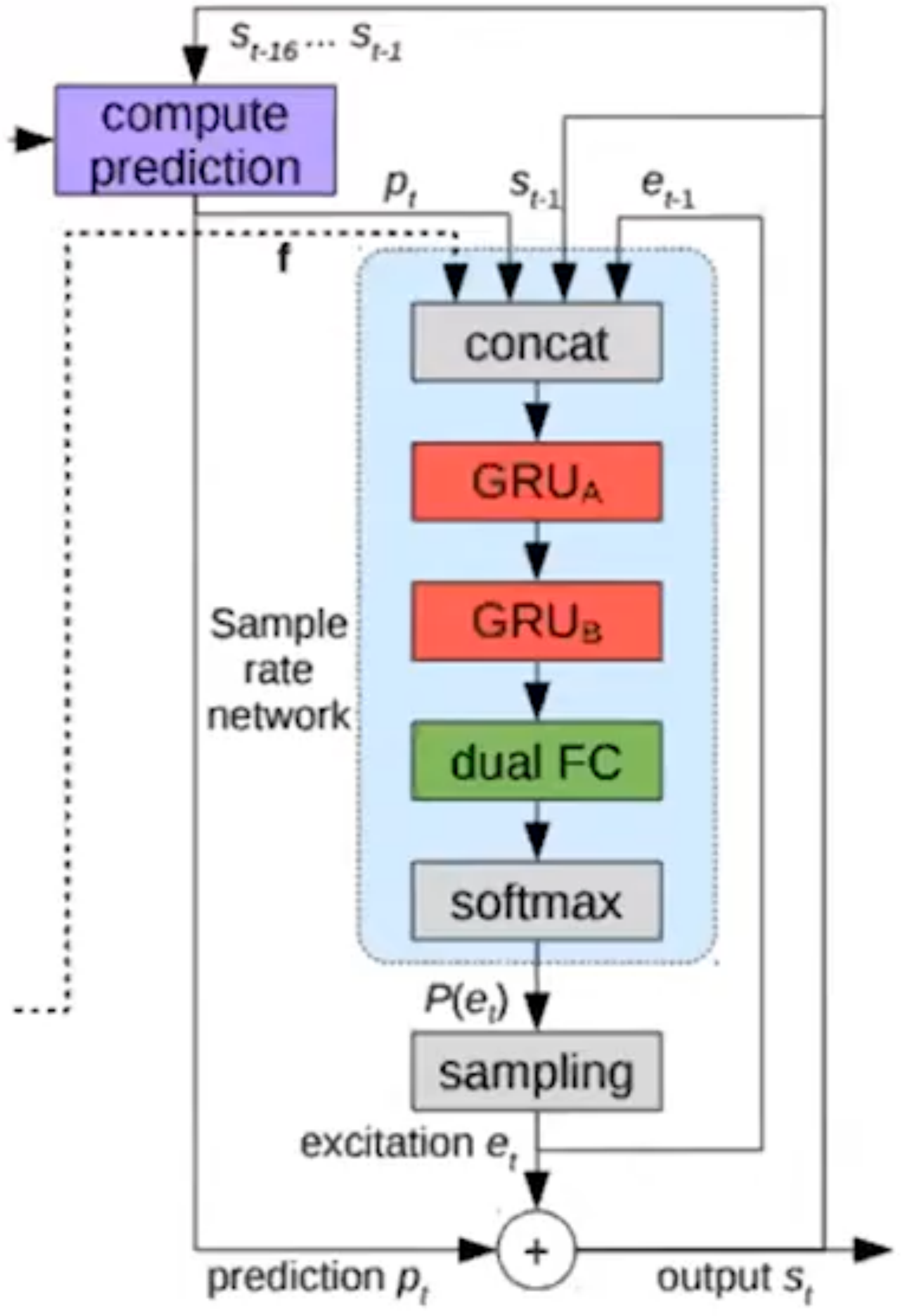


Image from LPCNet

TVM improved performance more than 100X in this environment

TVM @ Facebook, Tulloch et al., TVM Conf 2019

Industry-wide Impact

aws — Every "Alexa" wake-up today across all devices uses a model optimized with TVM

Facebook — "[TVM enabled] real-time on mobile CPUs for free...We are excited about the performance TVM achieves." More than 85x speed-up for speech recognition model.

Microsoft — Bing query understanding: 112ms (Tensorflow) -> 34ms (TVM). QnA bot: 73ms->28ms (CPU), 10.1ms->5.5ms (GPU)

Qualcomm — "TVM is key to ML Access on Hexagon"

CF: TVM Conference 2019 (Dec) - videos and slides available online

What Next?

- Apache TVM is a fast-growing open-source community
- Efforts related to TVM:
  - Support for more dynamism (e.g., dynamic graphs)
  - Integrate with VTA (Open Hardware Accelerator)
    - Software-Hardware Codesign
  - Unified runtime for heterogeneous devices

Other Compilers

- NVCC (NVIDIA CUDA Compiler)
  - works only with CUDA. Closed-source.
- XLA (Accelerated Linear Algebra, Google)
  - originally intended to speed up TensorFlow models, but has been adopted by JAX. Open-source as part of the TensorFlow repository.
- PyTorch Glow (Facebook)
  - PyTorch has adopted XLA to enable PyTorch on TPUs, but for other hardware, it relies on PyTorch Glow. Open-source as part of the PyTorch repository.

Thanks!