TVM: An Automated End-to-End Optimizing Compiler for Deep Learning

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan,
Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu,
Luis Ceze, Carlos Guestrin, Arvind Krishnamurthy

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Goal: Deploy Deep Learning Everywhere

Frameworks

Goal: Deploy Deep Learning Everywhere

Frameworks

Explosion of models and frameworks

Goal: Deploy Deep Learning Everywhere

Frameworks

Explosion of models and frameworks

Explosion of hardware backends

Goal: Deploy Deep Learning Everywhere

Frameworks

Explosion of models and frameworks

Explosion of hardware backends

Goal: Deploy Deep Learning Everywhere

Frameworks

Explosion of models and frameworks

Explosion of hardware backends

Goal: Deploy Deep Learning Everywhere

Frameworks

Explosion of models and frameworks

Explosion of hardware backends

Goal: Deploy Deep Learning Everywhere

Frameworks

Explosion of models and frameworks

Explosion of hardware backends

Goal: Deploy Deep Learning Everywhere

Frameworks

Explosion of models and frameworks

Explosion of hardware backends

Goal: Deploy Deep Learning Everywhere

Frameworks

Explosion of models and frameworks

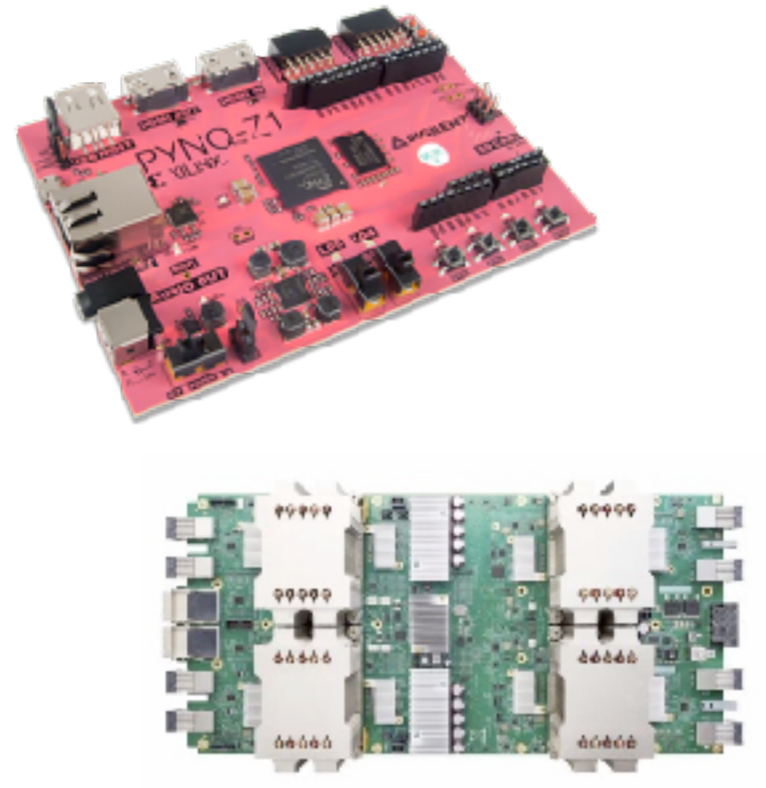
Explosion of hardware backends

Goal: Deploy Deep Learning Everywhere

Frameworks

Explosion of models and frameworks

Explosion of hardware backends

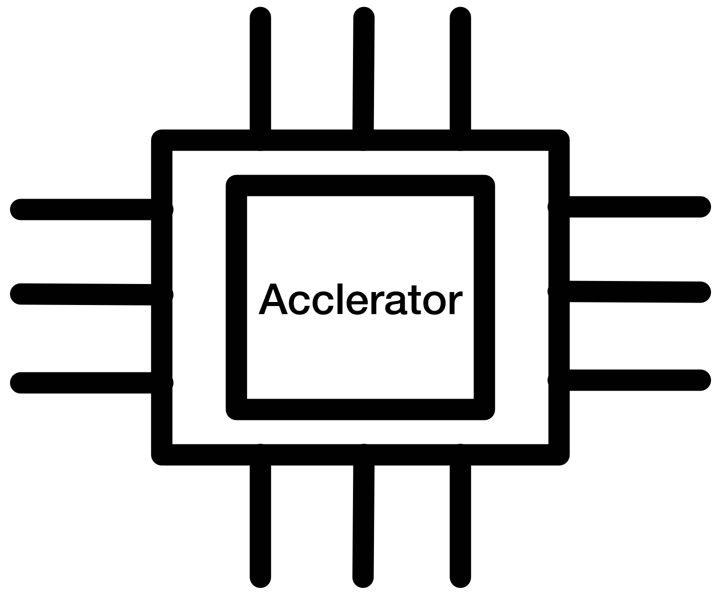


Goal: Deploy Deep Learning Everywhere

Frameworks

Explosion of models and frameworks

Explosion of hardware backends



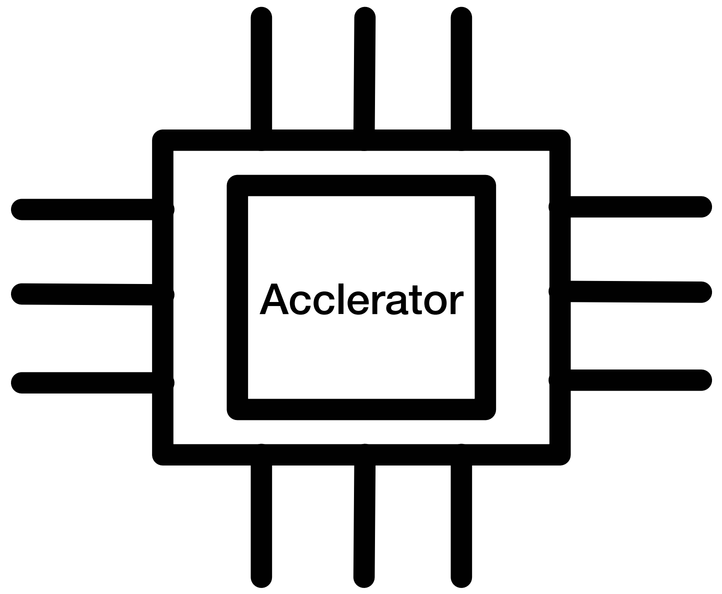
Goal: Deploy Deep Learning Everywhere

Frameworks

Explosion of models and frameworks

Huge gap between model/frameworks and hardware backends

Explosion of hardware backends



```python
import tvm
from tvm import relay

graph, params =
   frontend.from_keras(keras_resnet50)
graph, lib, params =
     relay.build(graph, target)
```

Compile

```python
import tvm
from tvm import relay

graph, params =
   frontend.from_keras(keras_resnet50)
graph, lib, params =
     relay.build(graph, target)
```

Compile

```python
import tvm
from tvm import relay

graph, params =
   frontend.from_keras(keras_resnet50)
graph, lib, params =
     relay.build(graph, target)
```

Compile

Deploy

```python
module = runtime.create(graph, lib, tvm.gpu(0))
module.set_input(**params)
module.run(data=data_array)
output = tvm.nd.empty(out_shape, ctx=tvm.gpu(0))
module.get_output(0, output)
```

input

Deployable Module

prediction **tabby, tabby cat**

tvm

Deploy

```
module = runtime.create(graph, lib, tvm.gpu(0))
module.set_input(**params)
module.run(data=data_array)
output = tvm.nd.empty(out_shape, ctx=tvm.gpu(0))
module.get_output(0, output)
```

input

Deployable Module

prediction **tabby, tabby cat**

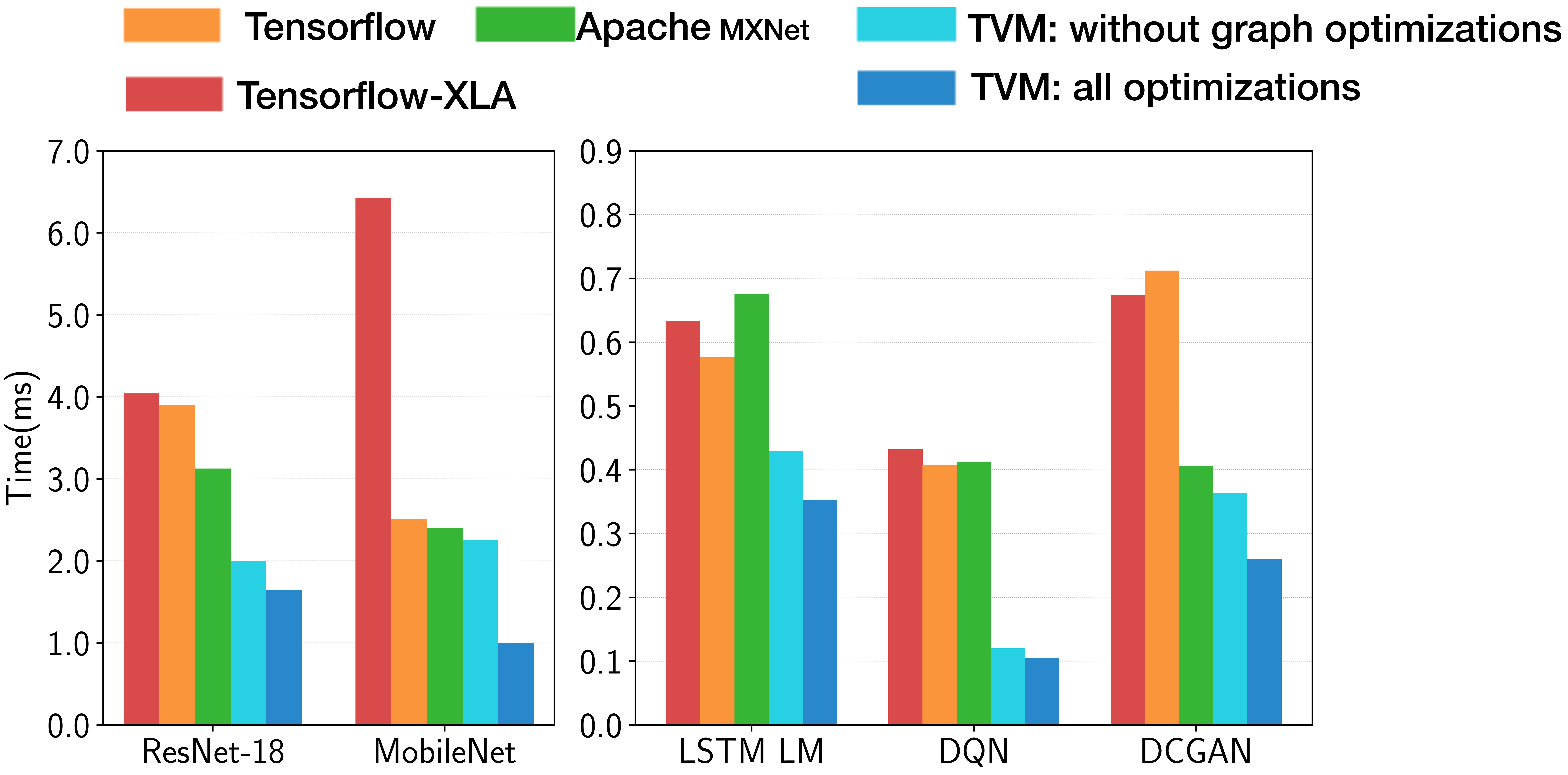
```
import tvm
from tvm import relay

graph, params =
   frontend.from_keras(keras_resnet50)
graph, lib, params =
     relay.build(graph, target)
```

Compile

On languages and platforms you choose

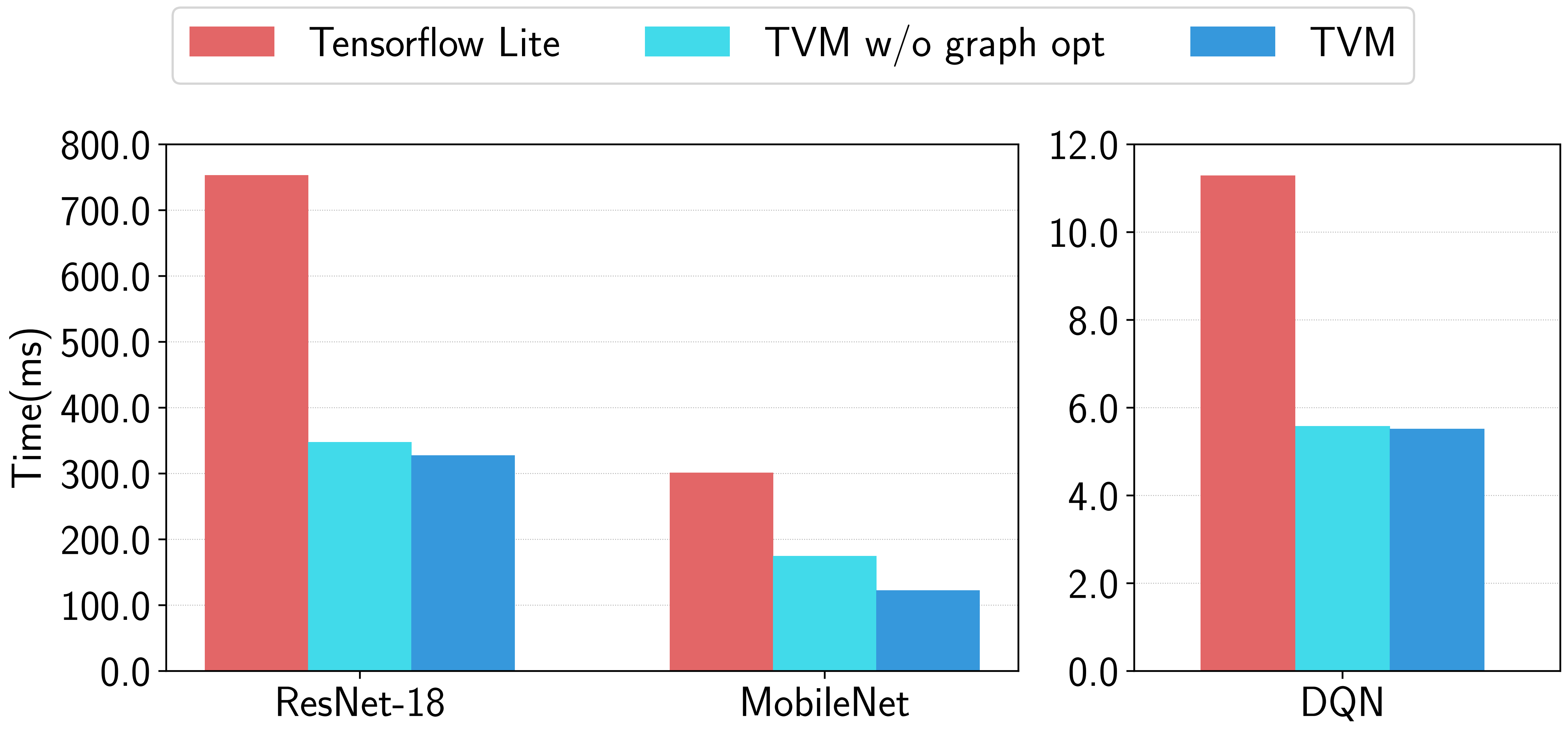
JS Java python iOS

Across Hardware Platforms

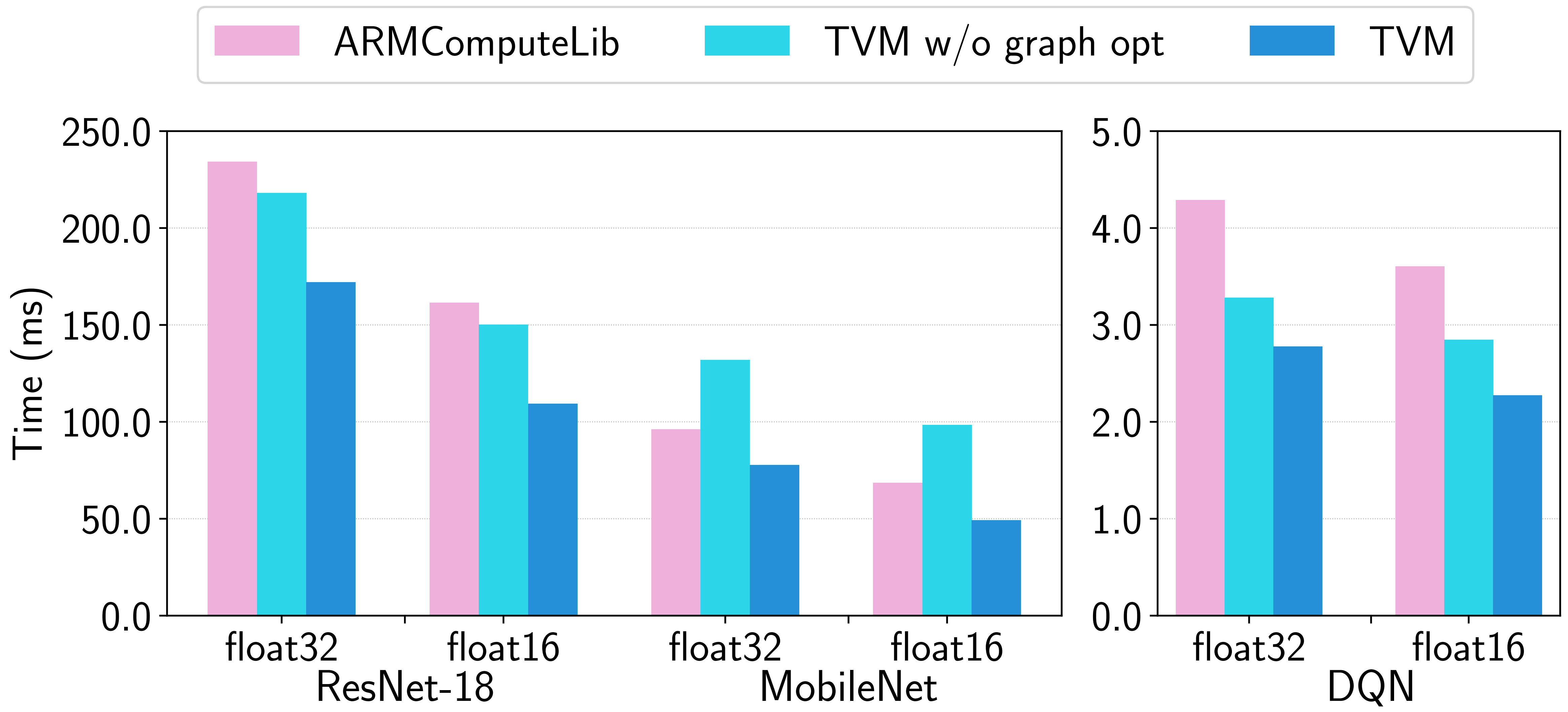


Across Hardware Platforms

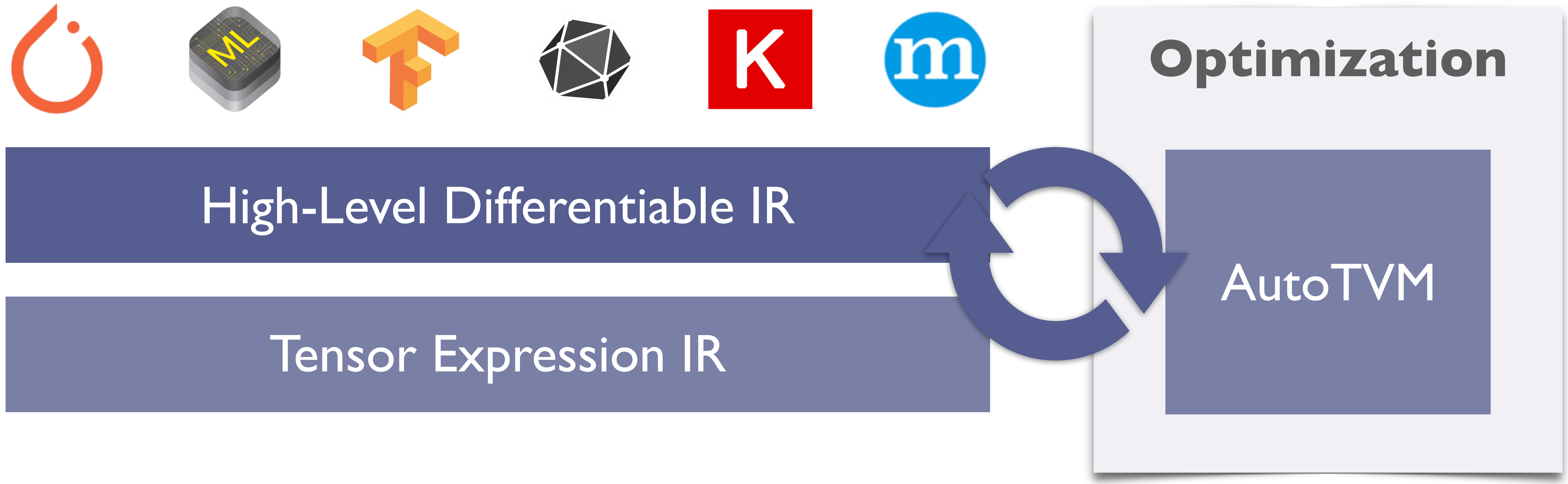
ARM CPU(A53)



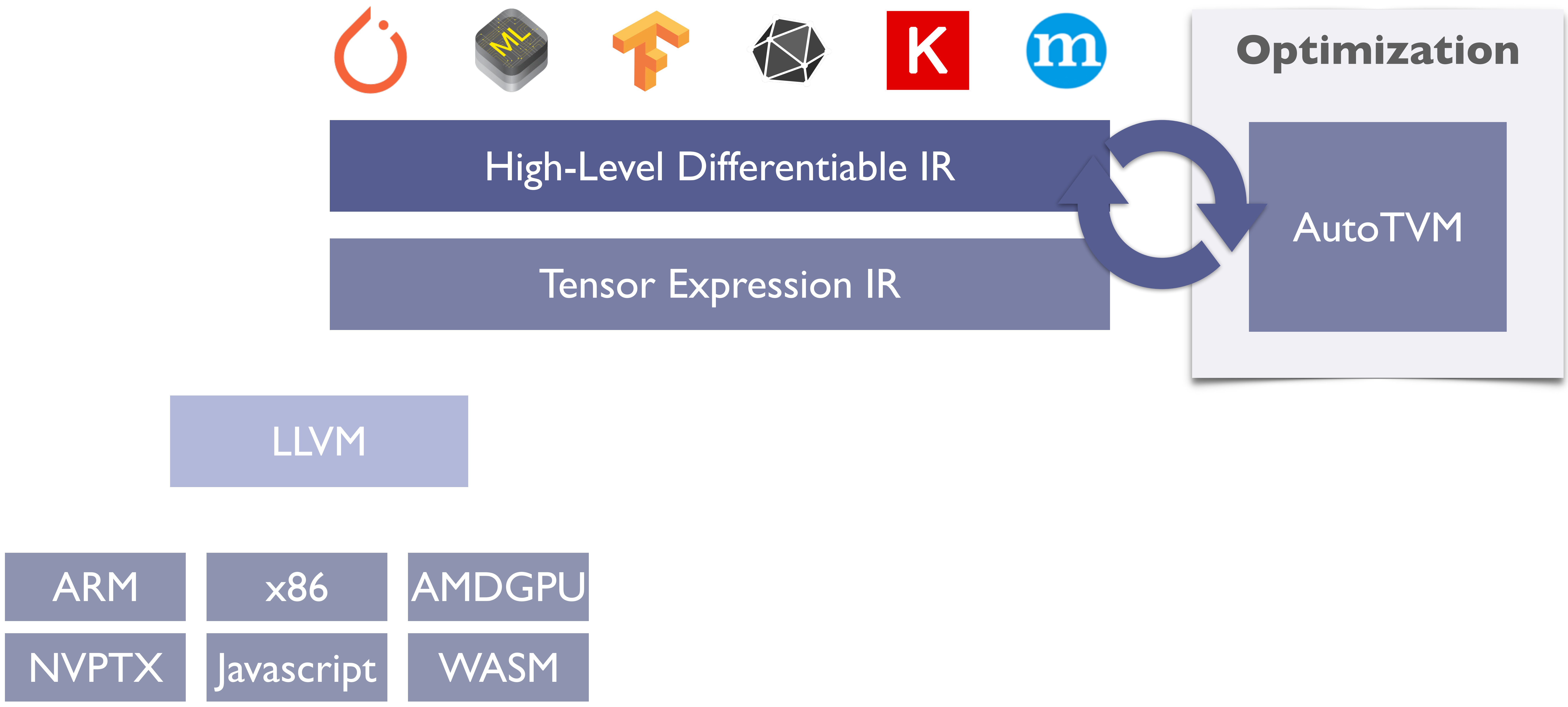
ARM GPU(MALI)



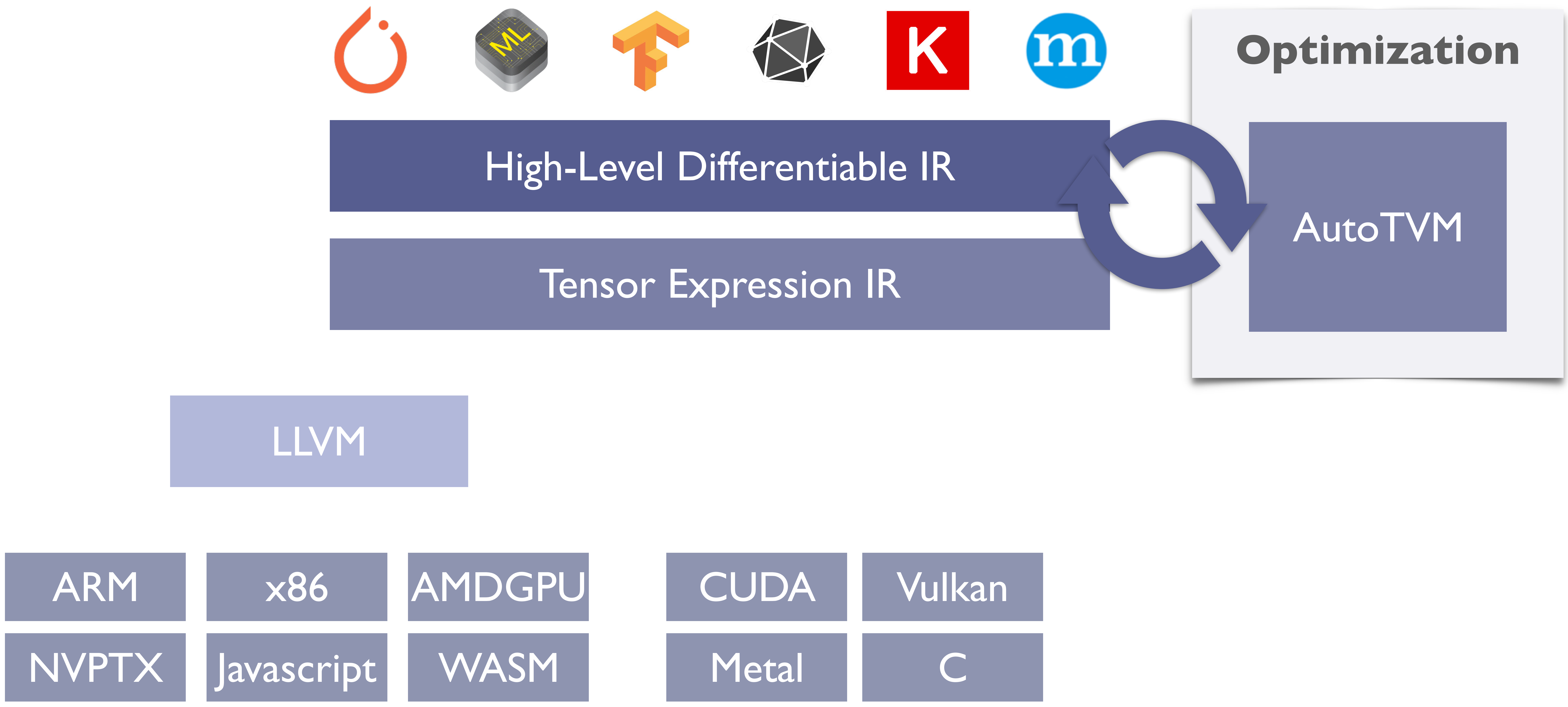
Diverse Hardware backends



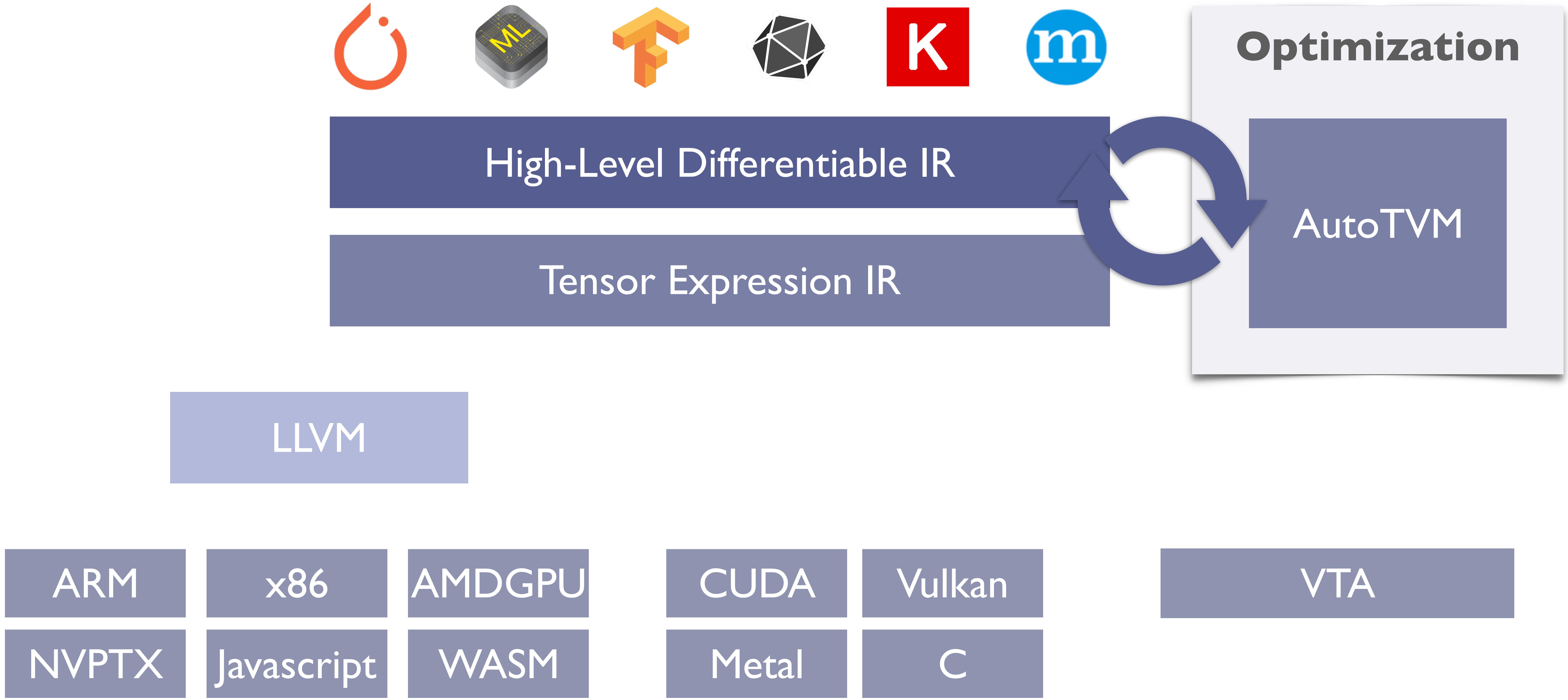
Diverse Hardware backends



Diverse Hardware backends



Diverse Hardware backends



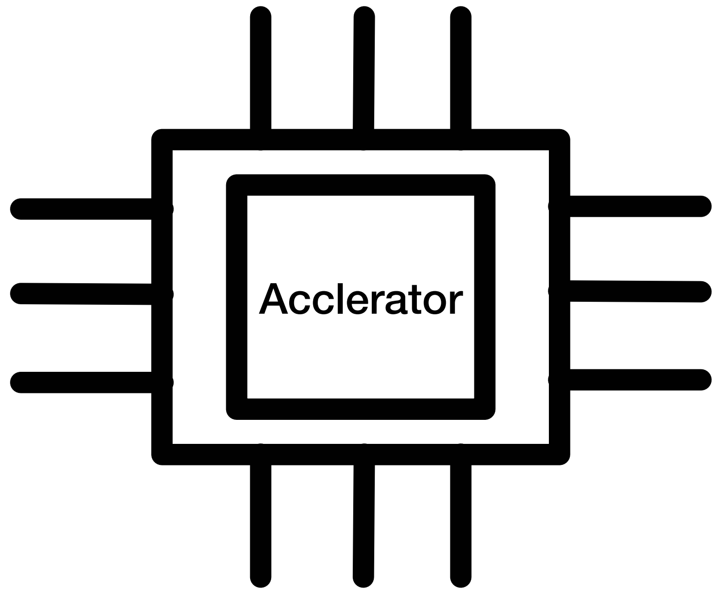
TVM in Productions

TVM in Productions

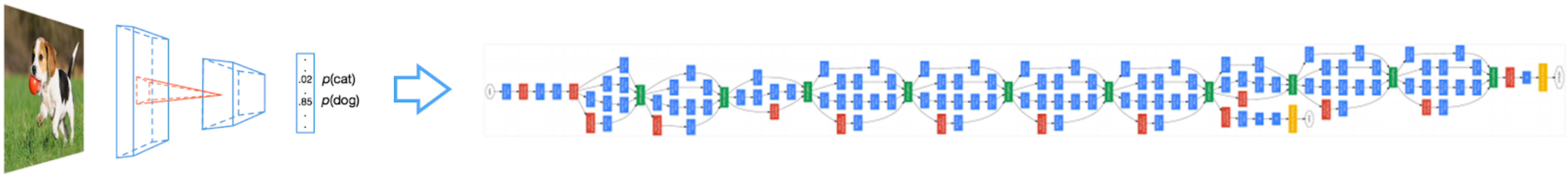
- AWS: Deep Learning Compiler in SageMaker Neo.
- Huawei: Compiler support for Ascent AI ASIC Chip.
- FB: caffe2/pytorch automatic optimization on mobile devices.
- https://sampl.cs.washington.edu/tvmconf/

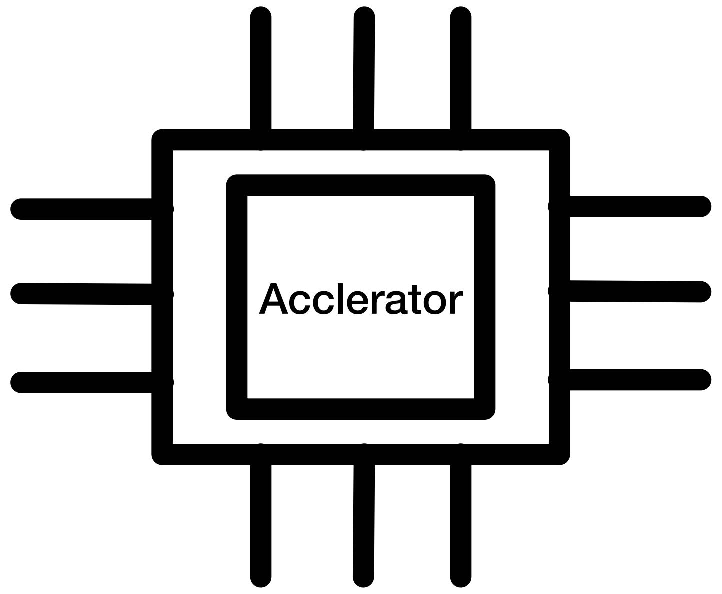
Beginning of Story

Beginning of Story

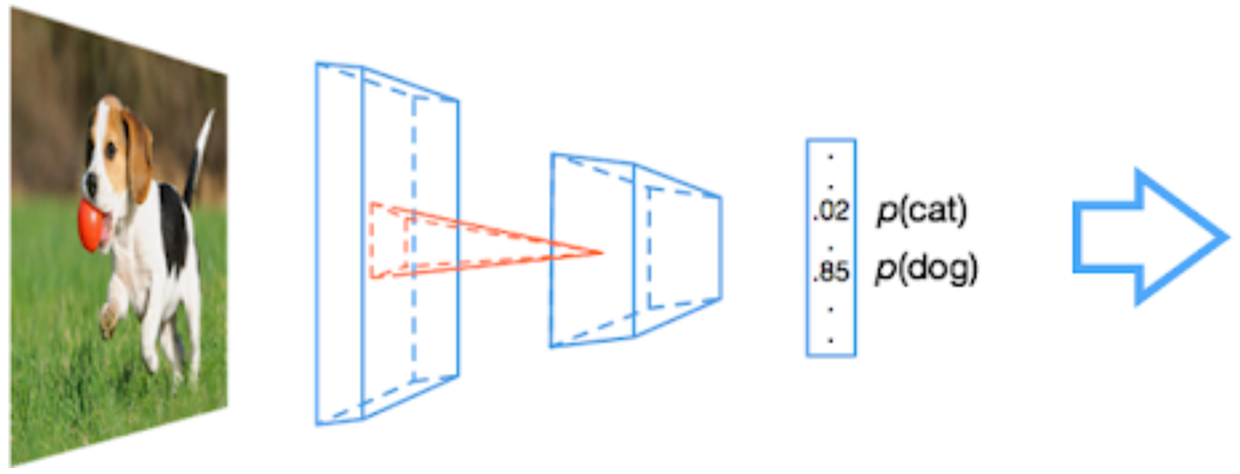


Beginning of Story

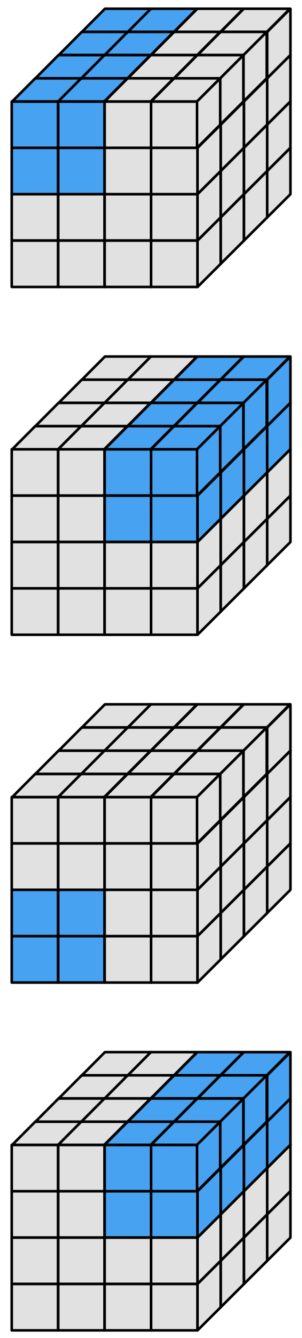




Beginning of Story



```
// Pseudo-code for convolution program for the VIA accelerator
// Virtual Thread 0
0x00: LOAD(PARAM[ 0-71])                                          // LD@TID0
0x01: LOAD(ACTIV[ 0-24])                                          // LD@TID0
0x02: LOAD(LDBUF[ 0-31])                                          // LD@TID0
0x03: PUSH(LD->EX)                                                // LD@TID0
0x04: POP (LD->EX)                                                // EX@TID0
0x05: EXE (ACTIV[ 0-24],PARAM[ 0-71],LDBUF[ 0-31],STBUF[ 0- 7])  // EX@TID0
0x06: PUSH(EX->LD)                                                // EX@TID0
0x07: PUSH(EX->ST)                                                // EX@TID0
0x08: POP (EX->ST)                                                // ST@TID0
0x09: STOR(STBUF[ 0- 7])                                          // ST@TID0
0x0A: PUSH(ST->EX)                                                // ST@TID0
// Virtual Thread 1
0x0B: LOAD(ACTIV[25-50])                                          // LD@TID1
0x0C: LOAD(LDBUF[32-63])                                          // LD@TID1
0x0D: PUSH(LD->EX)                                                // LD@TID1
0x0E: POP (LD->EX)                                                // EX@TID1
0x0F: EXE (ACTIV[25-50],PARAM[ 0-71],LDBUF[32-63],STBUF[32-39])  // EX@TID1
0x10: PUSH(EX->LD)                                                // EX@TID1
0x11: PUSH(EX->ST)                                                // EX@TID1
0x12: POP (EX->ST)                                                // ST@TID1
0x13: STOR(STBUF[32-39])                                          // ST@TID1
0x14: PUSH(ST->EX)                                                // ST@TID1
// Virtual Thread 2
0x15: POP (EX->LD)                                                // LD@TID2
0x16: LOAD(PARAM[ 0-71])                                          // LD@TID2
0x17: LOAD(ACTIV[ 0-24])                                          // LD@TID2
0x18: LOAD(LDBUF[ 0-31])                                          // LD@TID2
0x19: PUSH(LD->EX)                                                // LD@TID2
0x1A: POP (LD->EX)                                                // EX@TID2
0x1B: POP (ST->EX)                                                // EX@TID2
0x1C: EXE (ACTIV[ 0-24],PARAM[ 0-71],LDBUF[ 0-31],STBUF[ 0- 7])  // EX@TID2
0x1D: PUSH(EX->ST)                                                // EX@TID2
0x1E: POP (EX->ST)                                                // ST@TID2
0x1F: STOR(STBUF[ 0- 7])                                          // ST@TID2
// Virtual Thread 3
0x20: POP (EX->LD)                                                // LD@TID3
0x21: LOAD(ACTIV[25-50])                                          // LD@TID3
0x22: LOAD(LDBUF[32-63])                                          // LD@TID3
0x23: PUSH(LD->EX)                                                // LD@TID3
0x24: POP (LD->EX)                                                // EX@TID3
0x25: POP (ST->EX)                                                // EX@TID2
0x26: EXE (ACTIV[25-50],PARAM[ 0-71],LDBUF[32-63],STBUF[32-39])  // EX@TID3
0x27: PUSH(EX->ST)                                                // EX@TID3
0x28: POP (EX->ST)                                                // ST@TID3
0x29: STOR(STBUF[32-39])                                          // ST@TID3
```



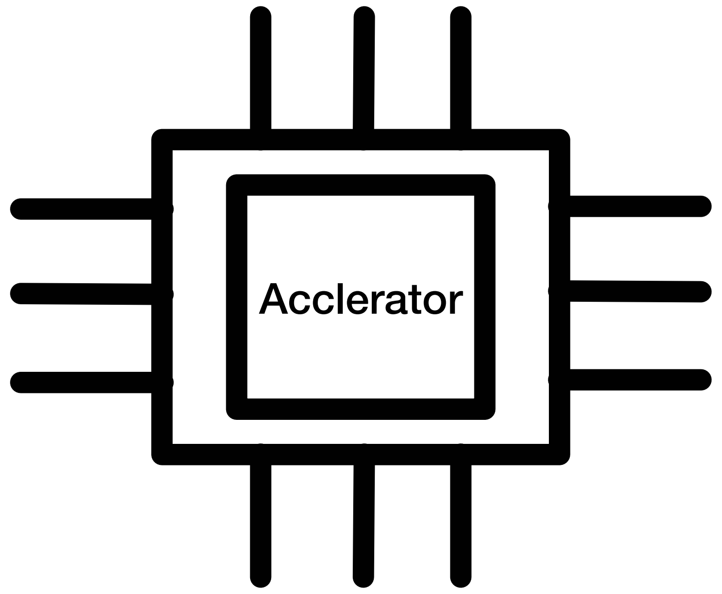
(a) Blocked convolution program with multiple thread contexts

```
// Convolution access pattern dictated by micro-coded program.
// Each register index is derived as a 2-D affine function.
// e.g. idx_rf = a_rf y+b_rf x+c_rf^0, where c_rf^0 is specified by
//      micro op 0 fields.
for y in [0…i)
  for x in [0…j)
    rf[idx_rf^0] += GEVM(act[idx_act^0], par[idx_par^0])
    rf[idx_rf^1] += GEVM(act[idx_act^1], par[idx_par^1])
    …
    rf[idx_rf^n] += GEVM(act[idx_act^n], par[idx_par^n])
```

(b) Convolution micro-coded program

```
// Max-pool, batch normalization and activation function
// access pattern dictated by micro-coded program.
// Each register index is derived as a 2D affine function.
// e.g. idx_dst = a_dst y+b_dst x+c_dst^0, where c_dst^0 is specified by
//      micro op 0 fields.
for y in [0…i)
  for x in [0…j)
    // max pooling
    rf[idx_dst^0] = MAX(rf[idx_dst^0], rf[idx_src^0])
    rf[idx_dst^1] = MAX(rf[idx_dst^1], rf[idx_src^1])
    …
    // batch norm
    rf[idx_dst^m] = MUL(rf[idx_dst^m], rf[idx_src^m])
    rf[idx_dst^m+1] = ADD(rf[idx_dst^m+1], rf[idx_src^m+1])
    rf[idx_dst^m+2] = MUL(rf[idx_dst^m+2], rf[idx_src^m+2])
    rf[idx_dst^m+3] = ADD(rf[idx_dst^m+3], rf[idx_src^m+3])
    …
    // activation
    rf[idx_dst^n-1] = RELU(rf[idx_dst^n-1], rf[idx_src^n-1])
    rf[idx_dst^n] = RELU(rf[idx_dst^n], rf[idx_src^n])
```

(c) Max pool, batch norm and activation micro-coded program



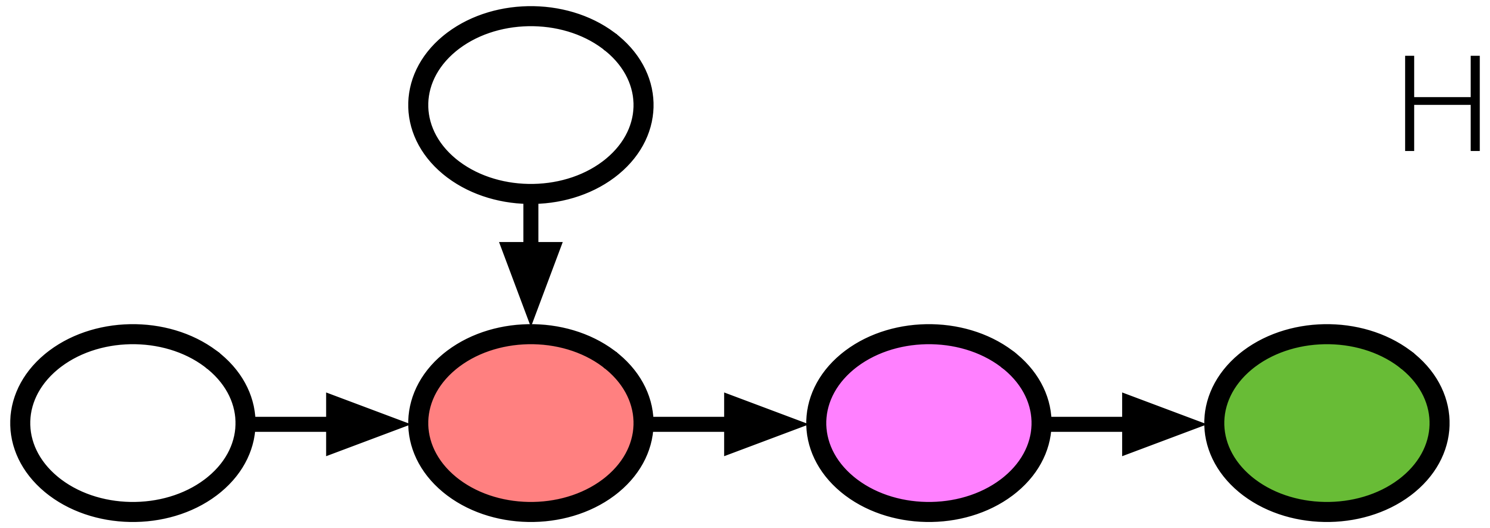
Existing Approach

Frameworks

Hardware

Existing Approach

Frameworks

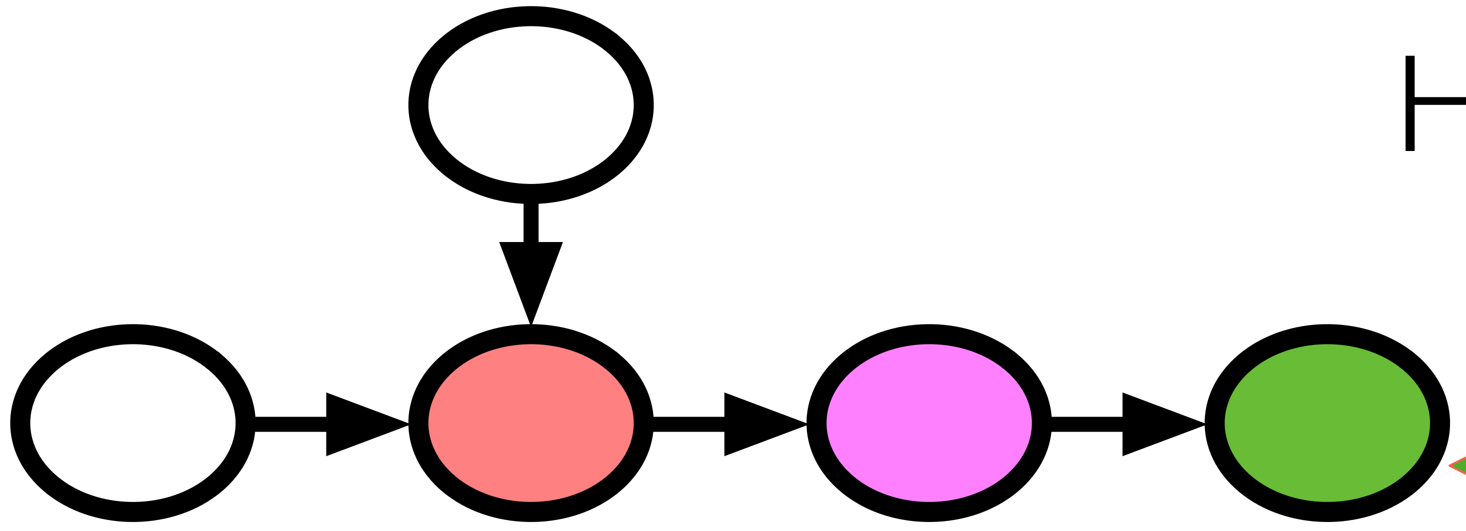


High-level data flow graph

Hardware

Existing Approach

Frameworks

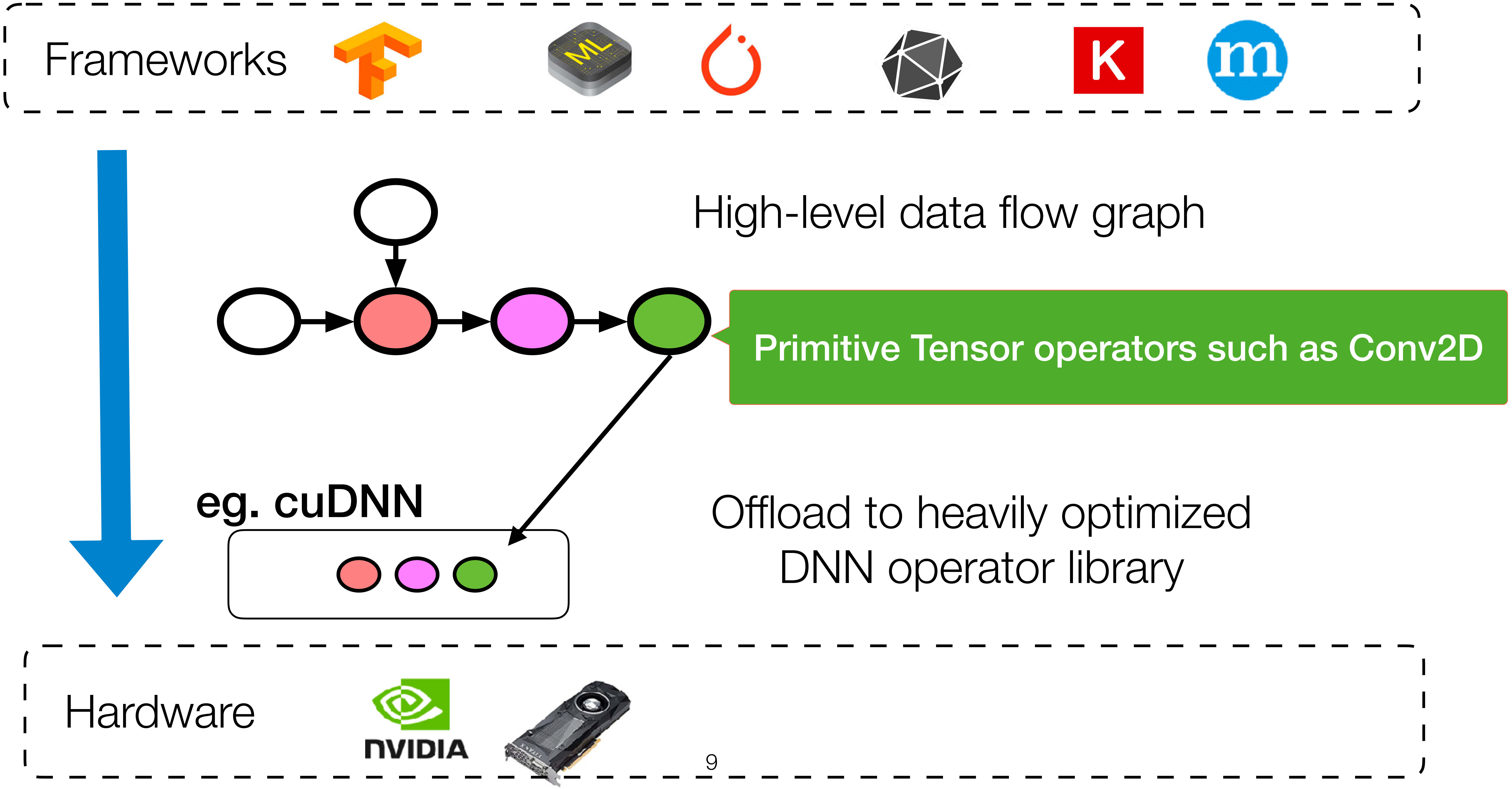


High-level data flow graph

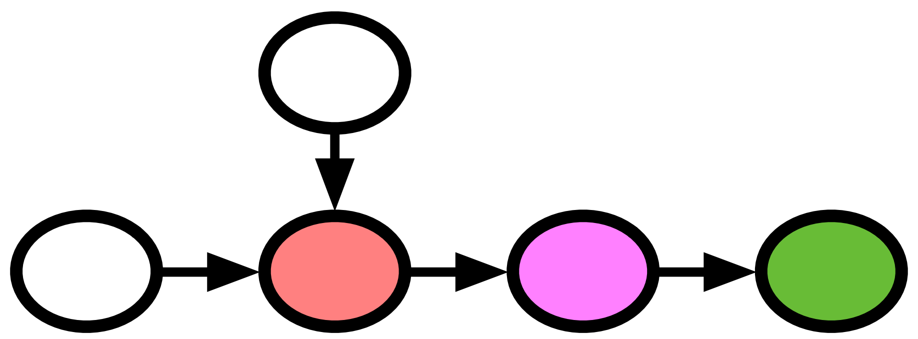
Primitive Tensor operators such as Conv2D

Hardware

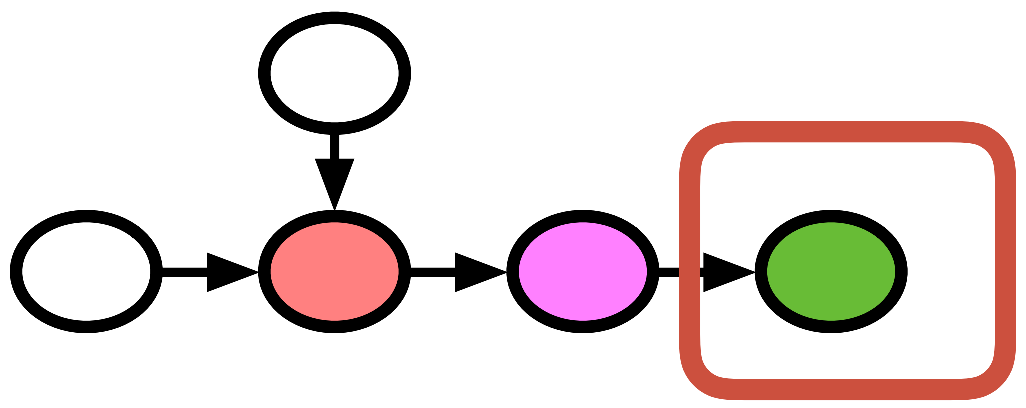
Existing Approach



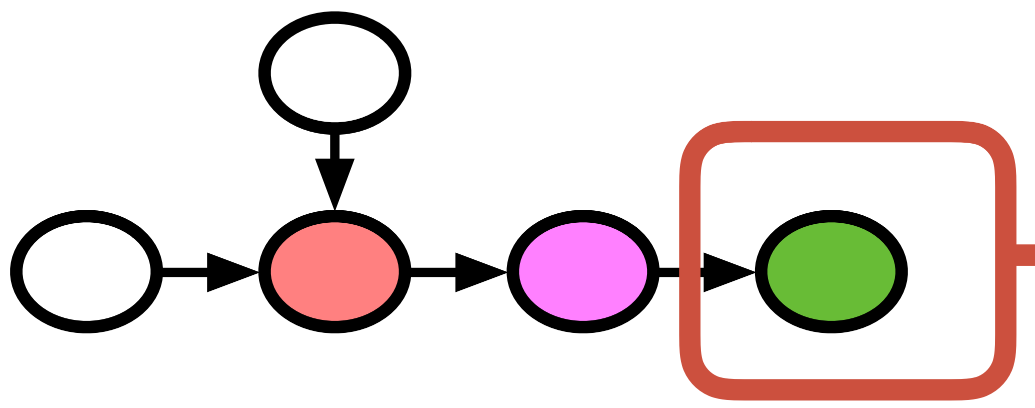
Existing Approach: Engineer Optimized Tensor Operators



Existing Approach: Engineer Optimized Tensor Operators



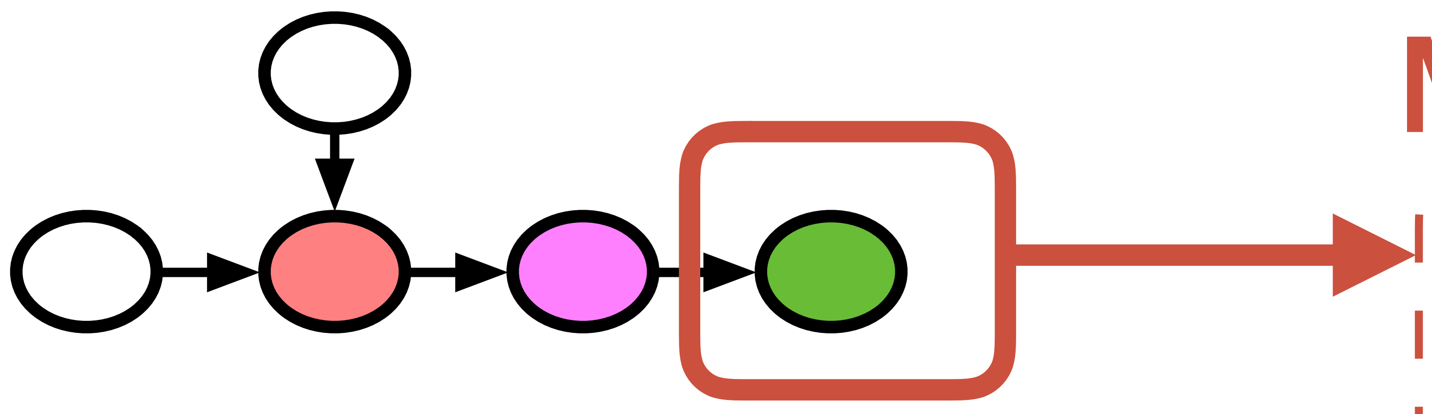
Existing Approach: Engineer Optimized Tensor Operators



Matmul: Operator Specification

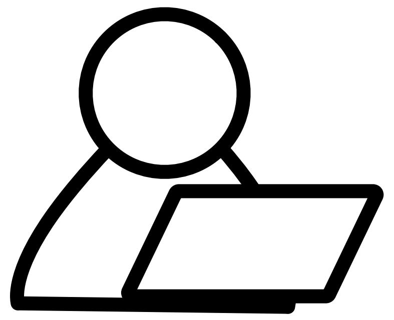
```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Existing Approach: Engineer Optimized Tensor Operators

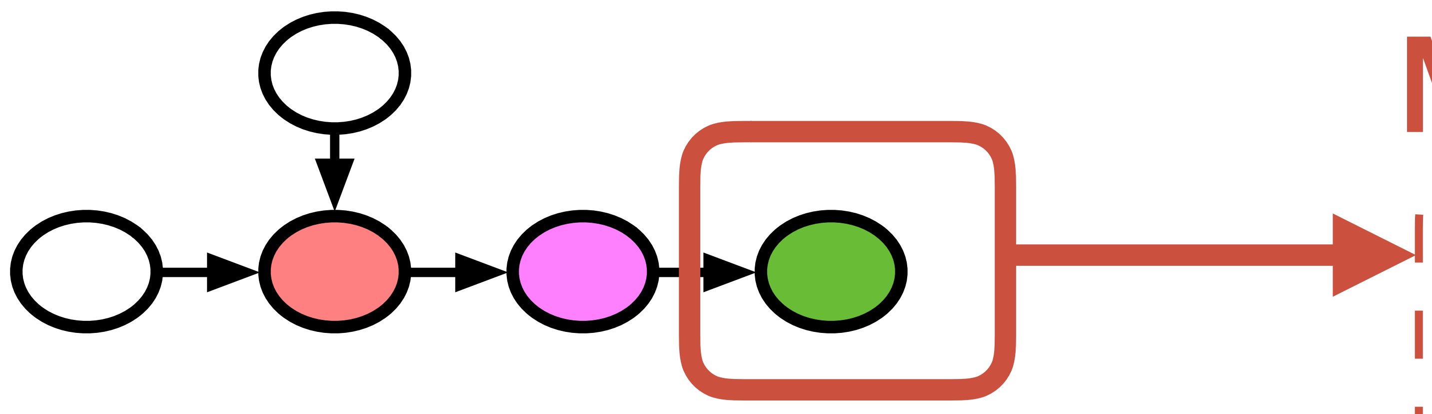


Matmul: Operator Specification

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

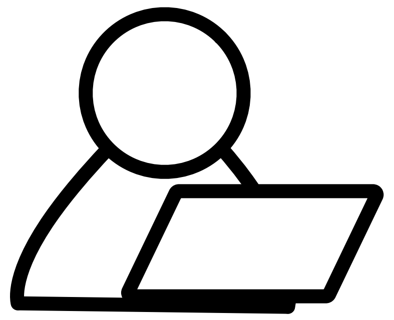


Existing Approach: Engineer Optimized Tensor Operators



Matmul: Operator Specification

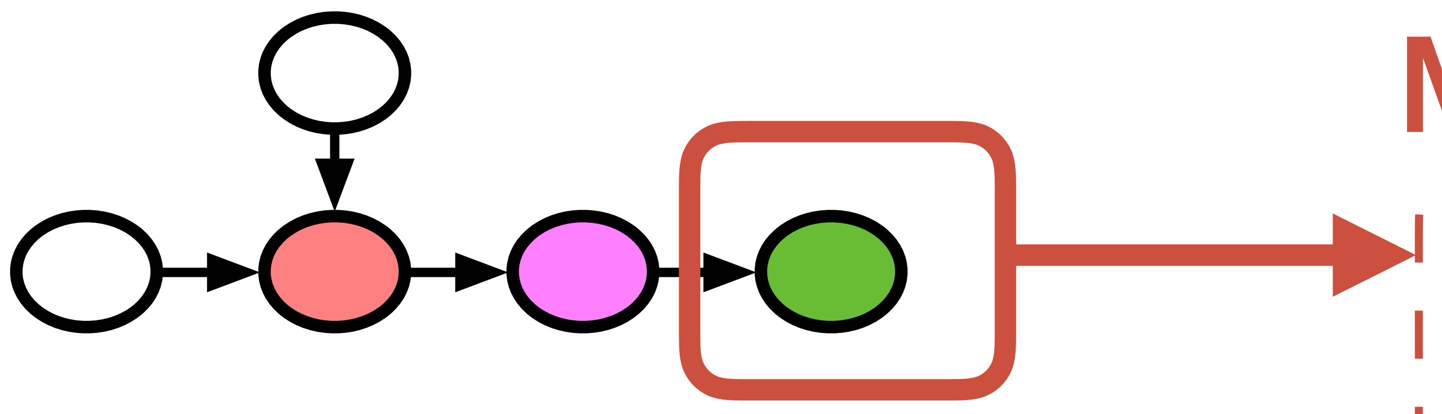
```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```



Vanilla Code

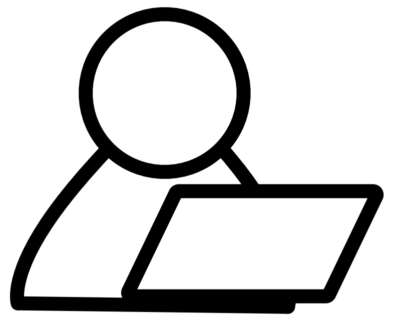
```
for y in range(1024):
  for x in range(1024):
    C[y][x] = 0
    for k in range(1024):
      C[y][x] += A[k][y] * B[k][x]
```

Existing Approach: Engineer Optimized Tensor Operators



Matmul: Operator Specification

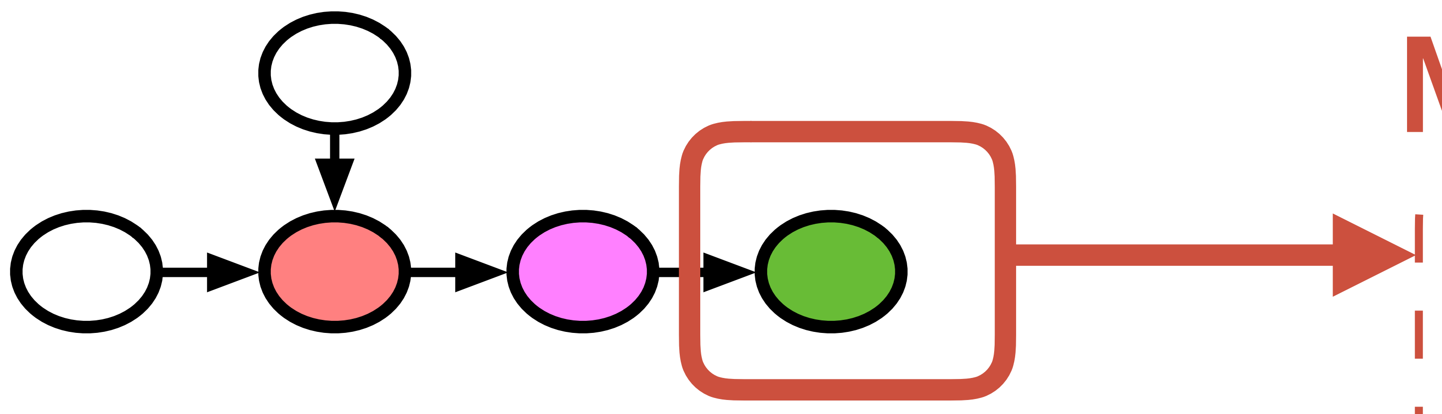
```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```



Loop Tiling for Locality

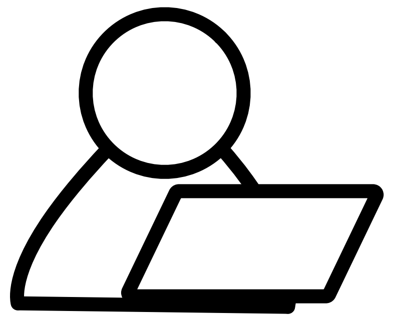
```
for yo in range(128):
  for xo in range(128):
    C[yo*8:yo*8+8][xo*8:xo*8+8] = 0
    for ko in range(128):
      for yi in range(8):
        for xi in range(8):
          for ki in range(8):
            C[yo*8+yi][xo*8+xi] +=
               A[ko*8+ki][yo*8+yi] * B[ko*8+ki][xo*8+xi]
```

Existing Approach: Engineer Optimized Tensor Operators



Matmul: Operator Specification

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



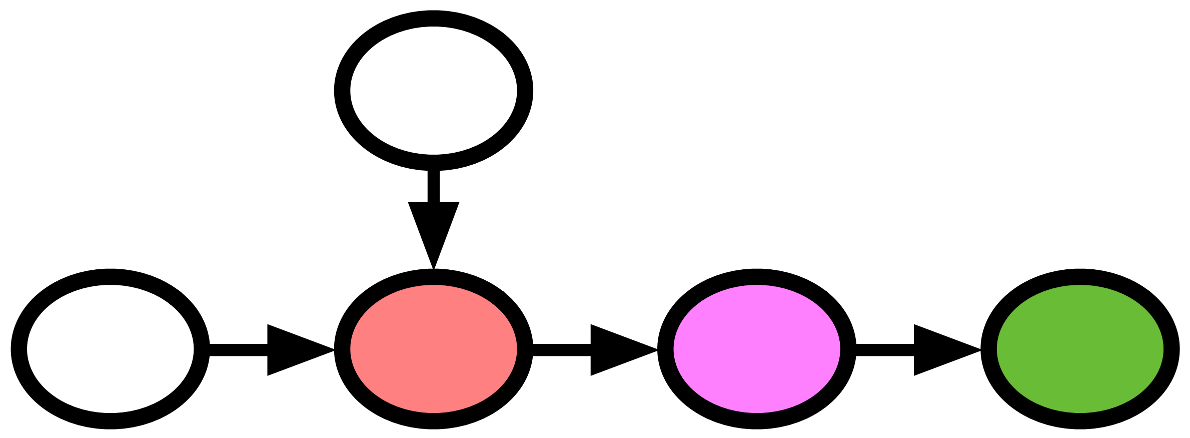
Map to Accelerators

```
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

Limitations of Existing Approach

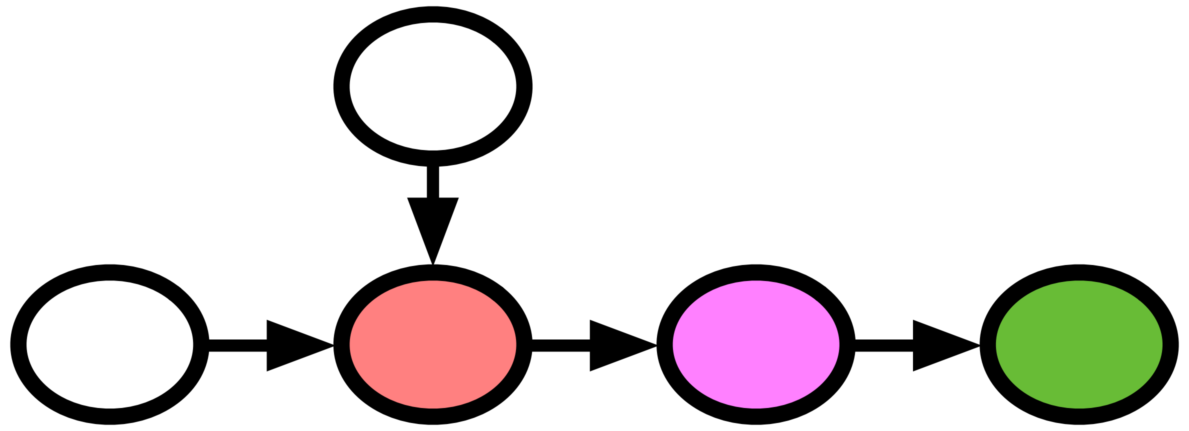
Frameworks



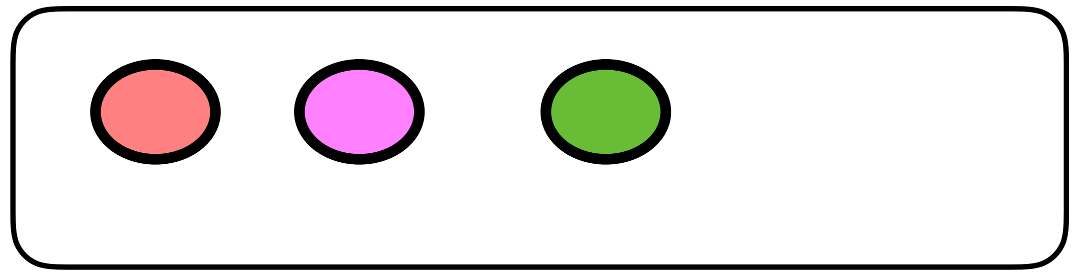
cuDNN

Limitations of Existing Approach

Frameworks

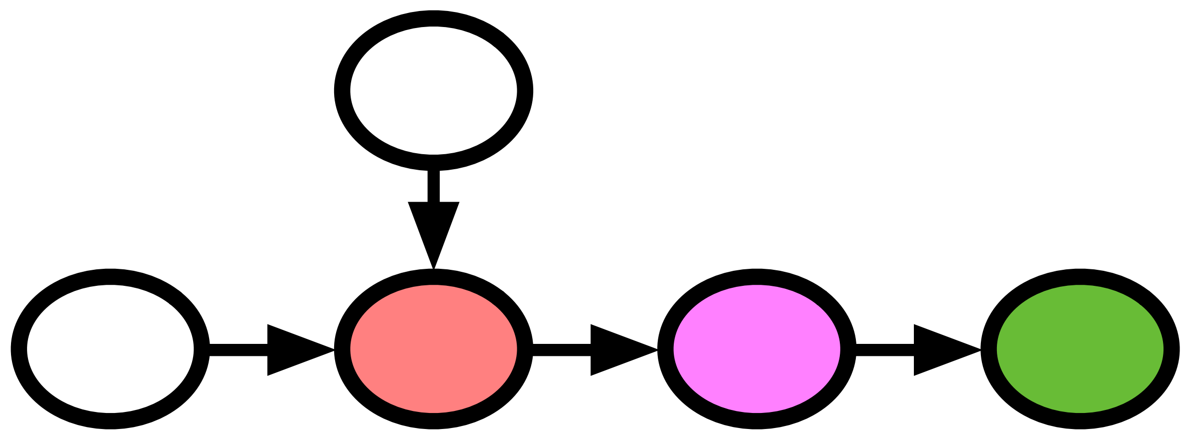


cuDNN

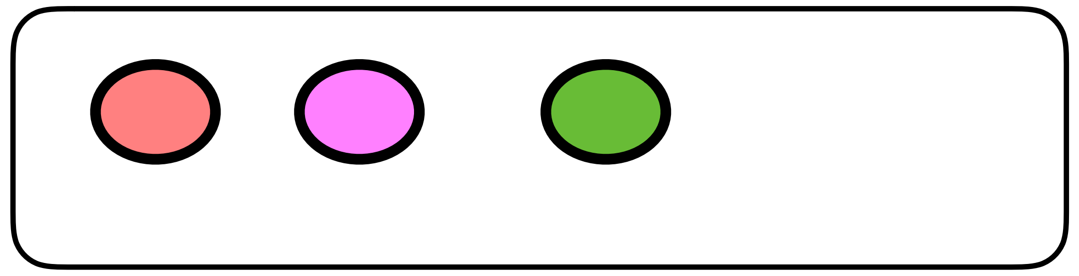


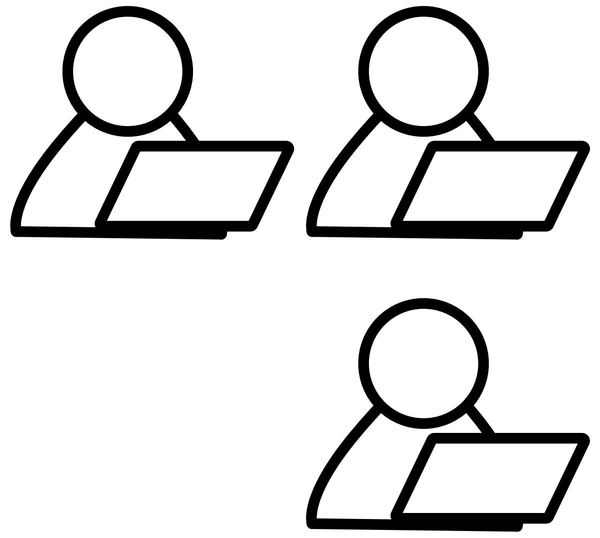
Limitations of Existing Approach

Frameworks



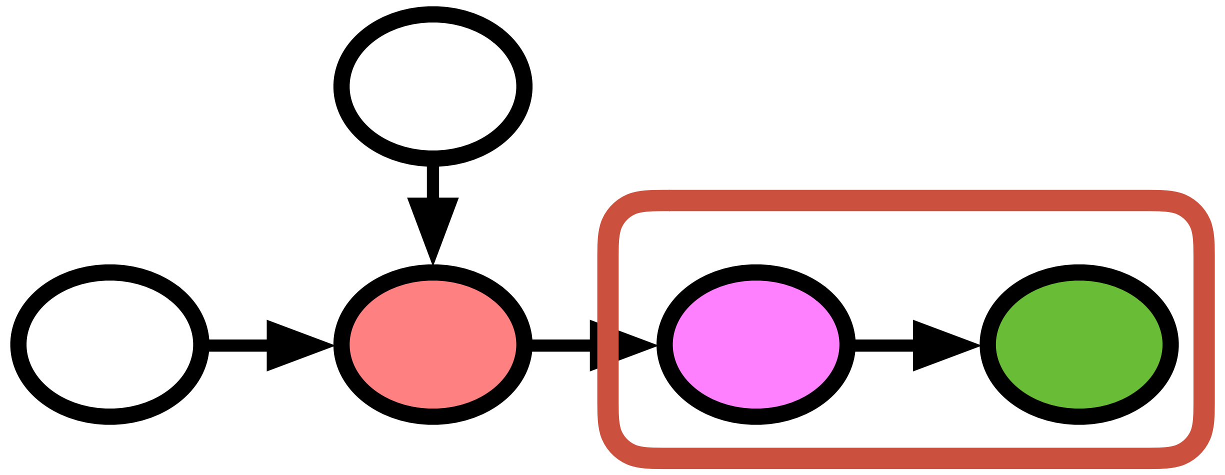
cuDNN



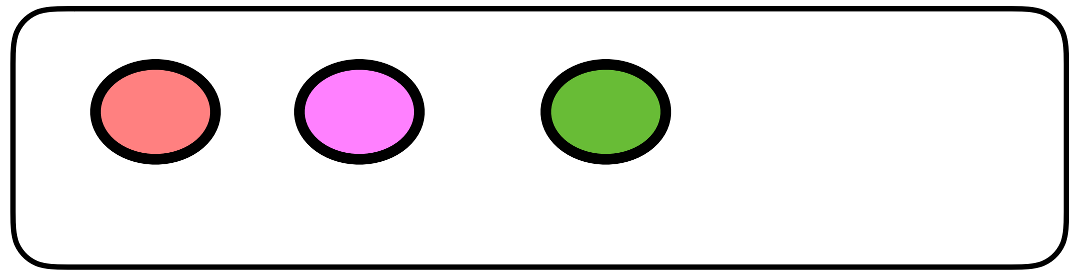


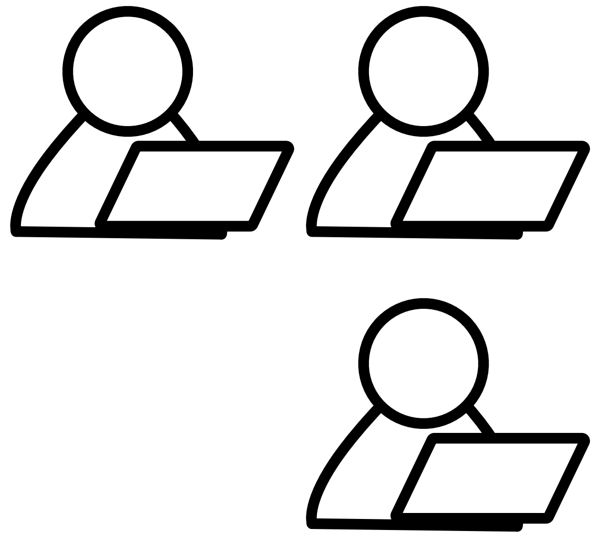
Limitations of Existing Approach

Frameworks



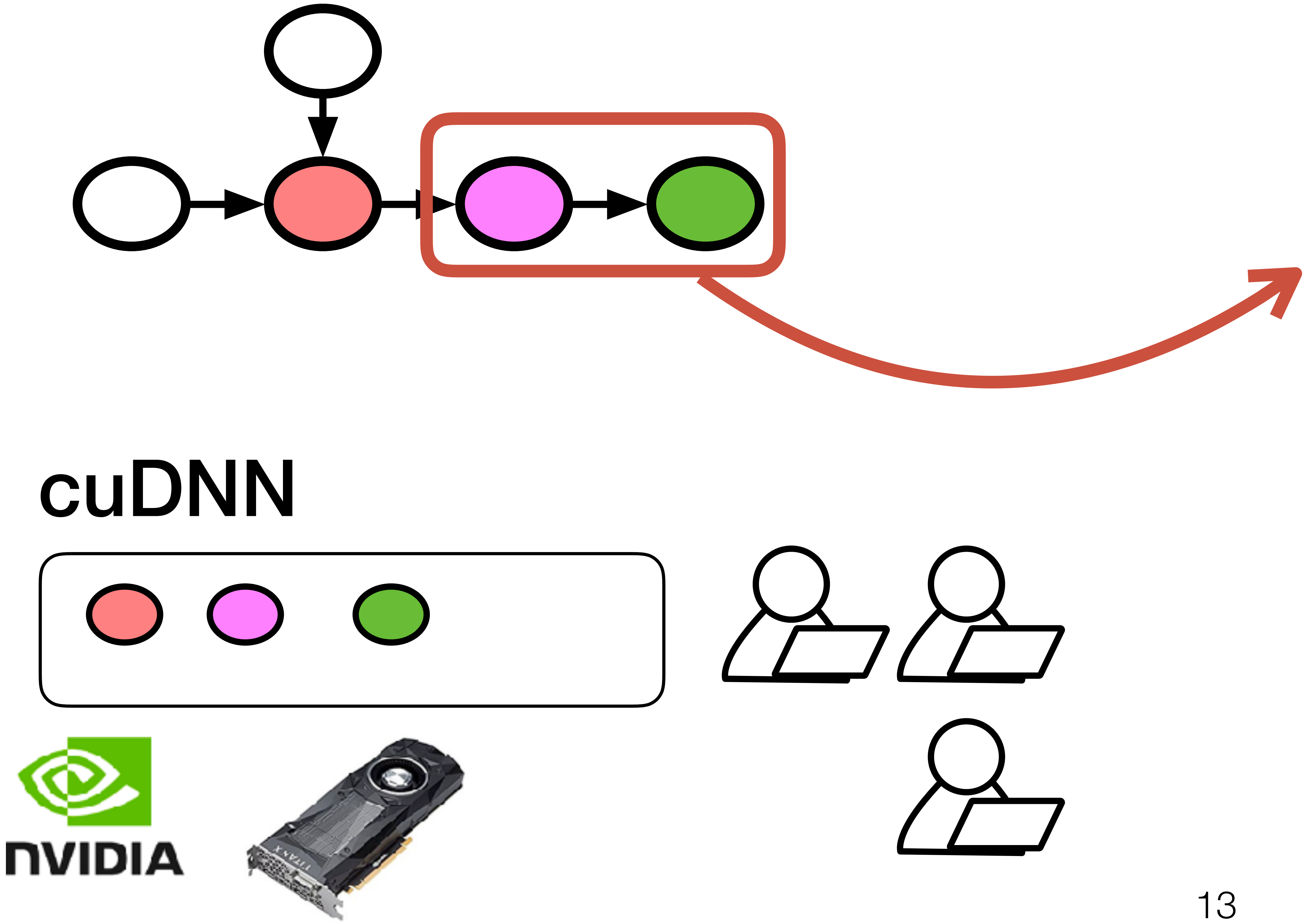
cuDNN





Limitations of Existing Approach

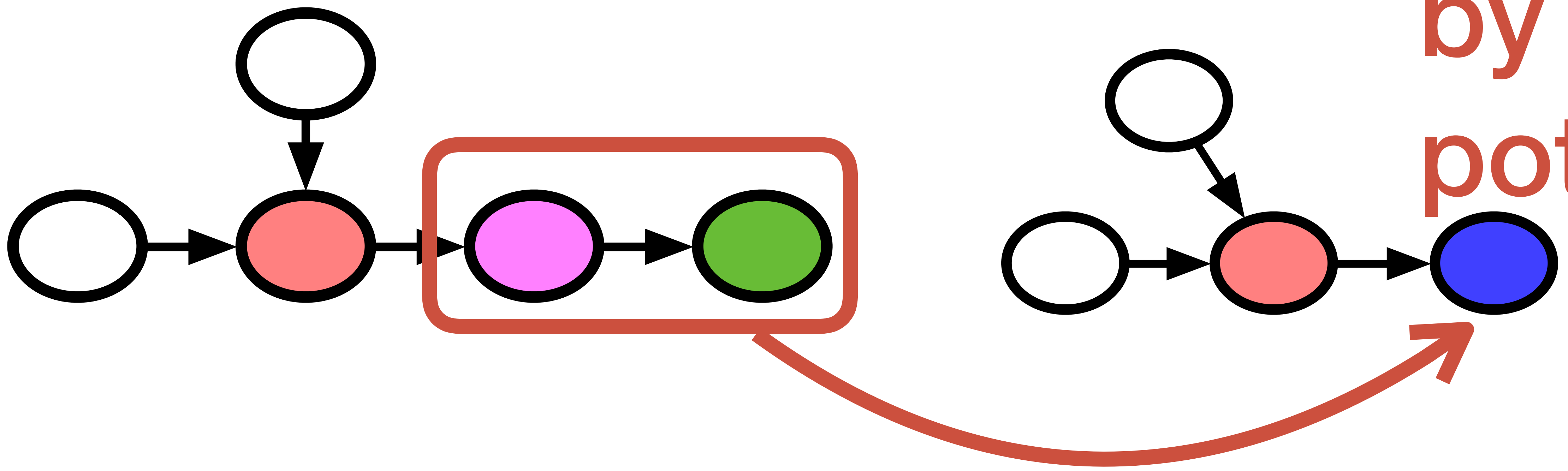
Frameworks



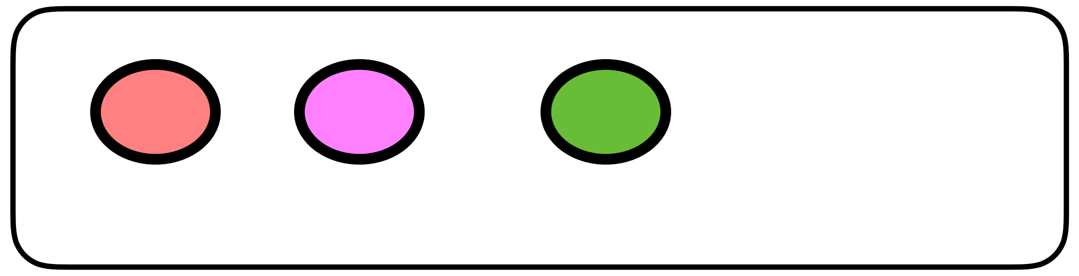
Limitations of Existing Approach

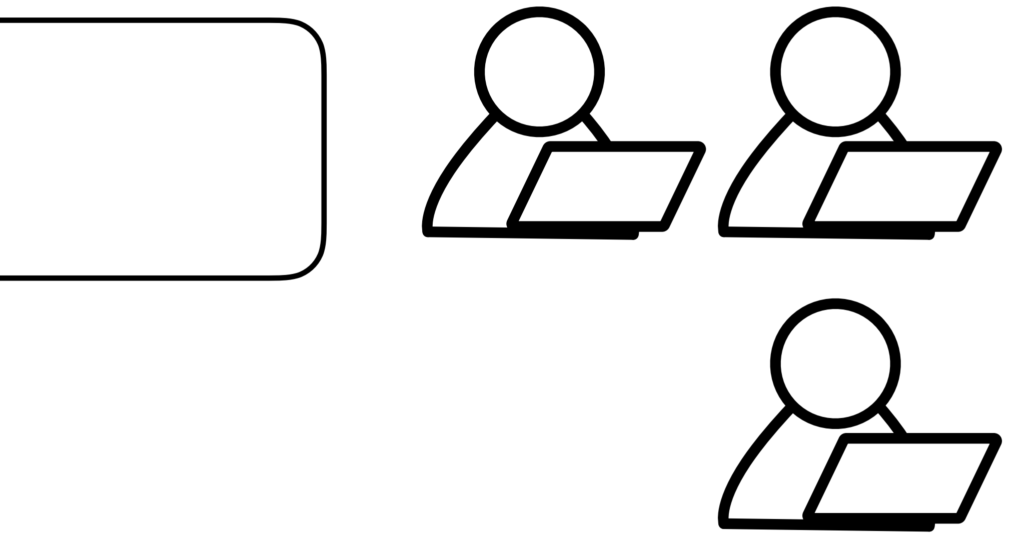
Frameworks

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



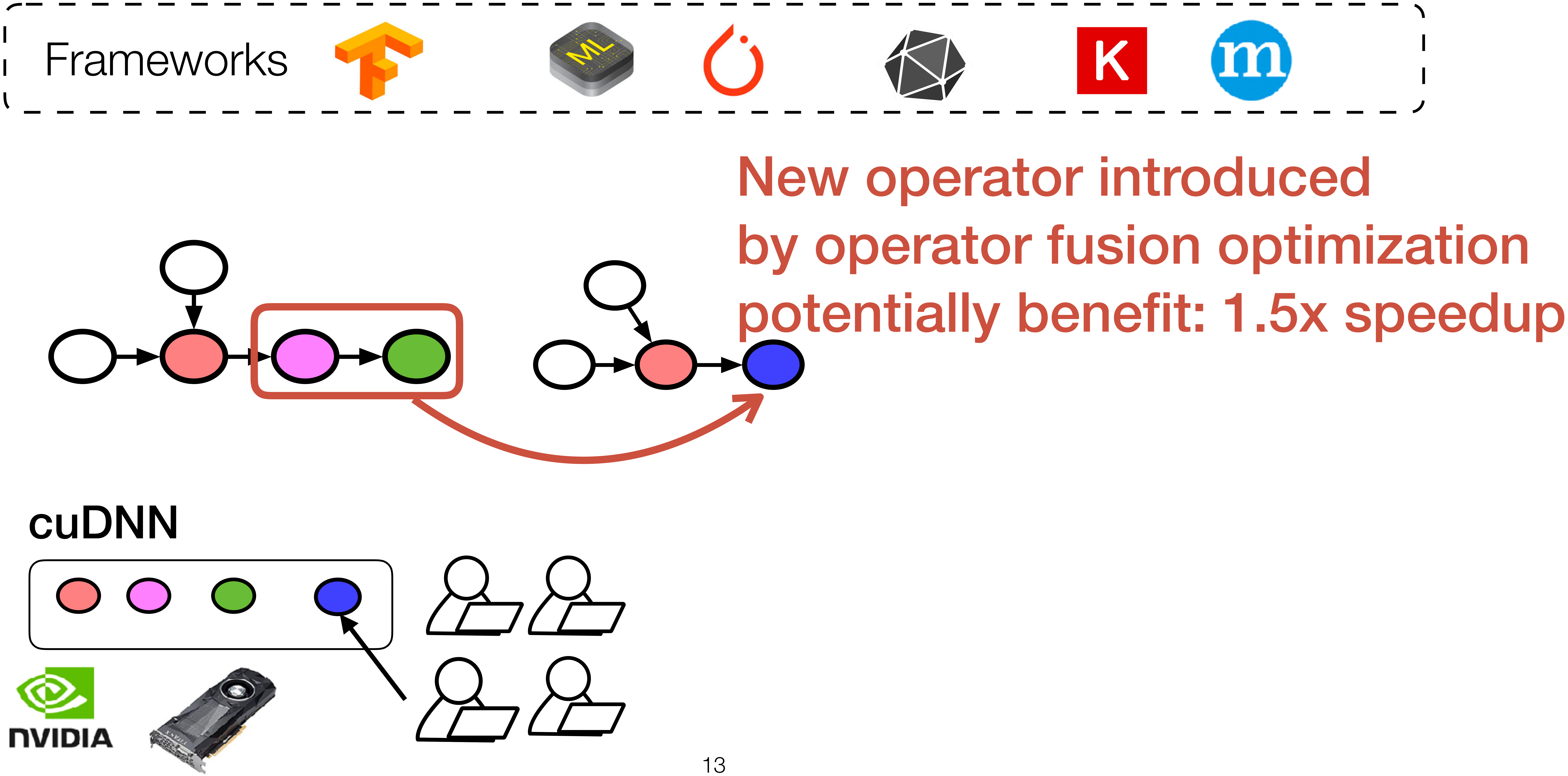
cuDNN





NVIDIA

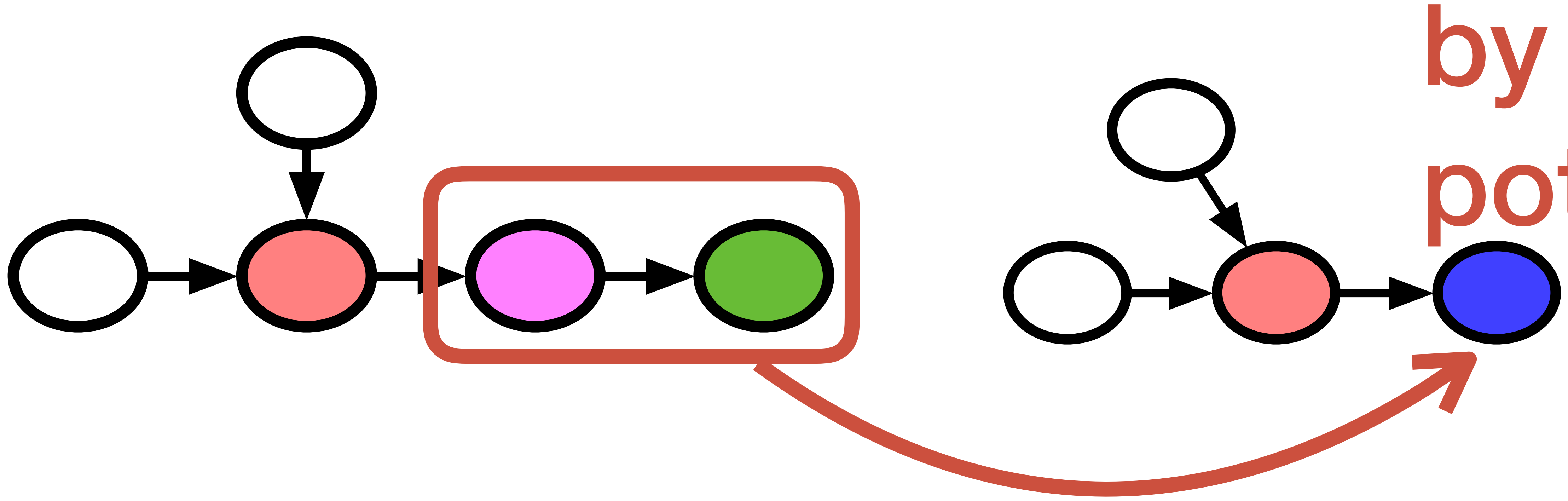
Limitations of Existing Approach



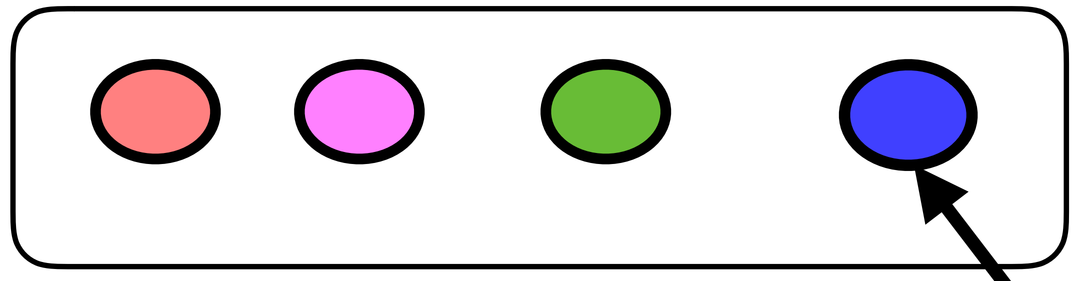
Limitations of Existing Approach

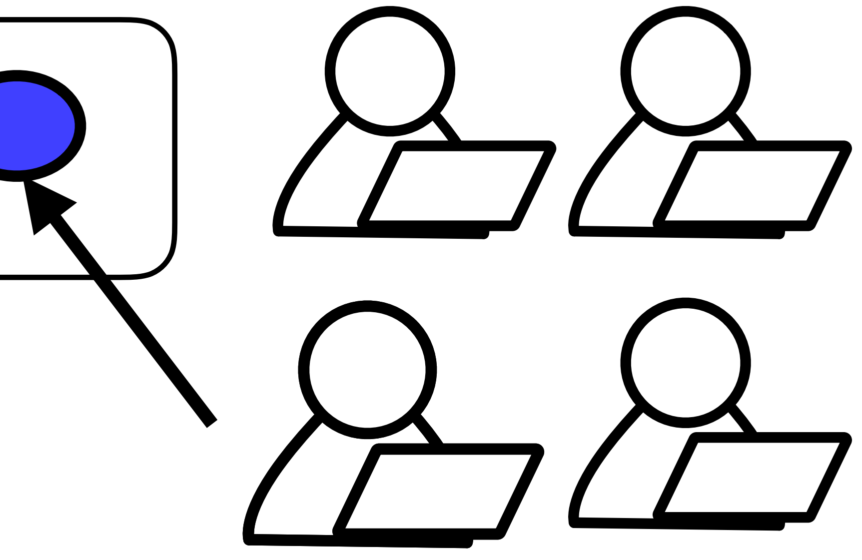
Frameworks

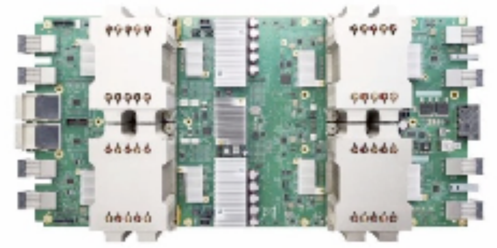
New operator introduced
by operator fusion optimization
potentially benefit: 1.5x speedup



cuDNN

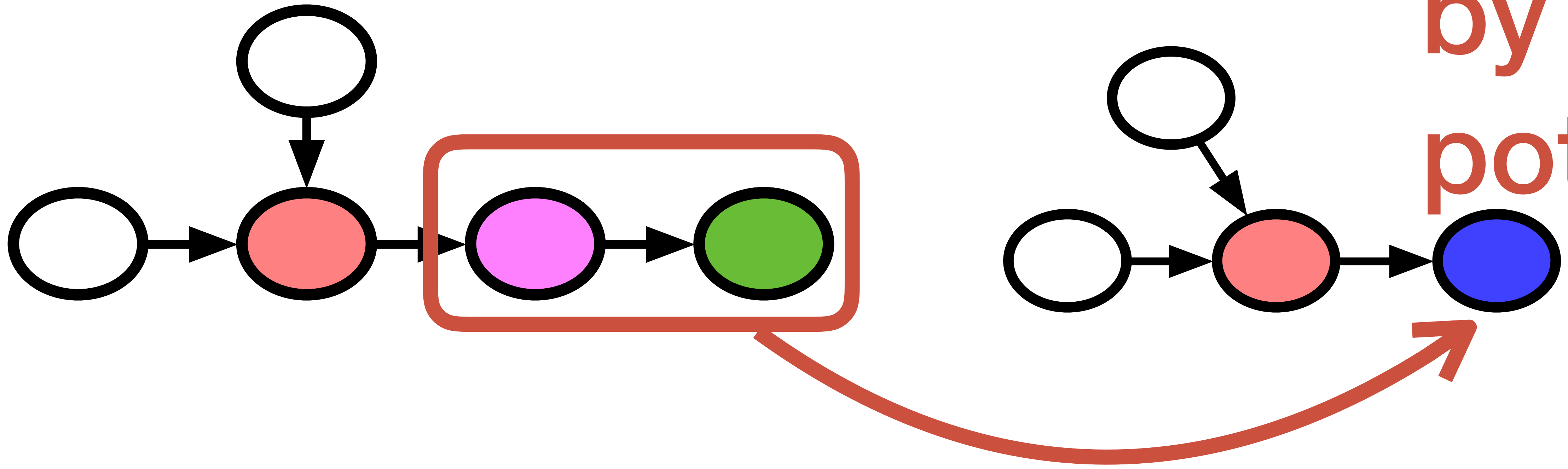






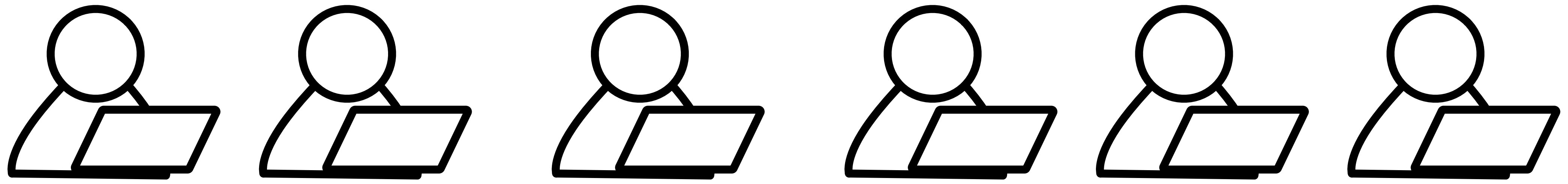
Limitations of Existing Approach

Frameworks

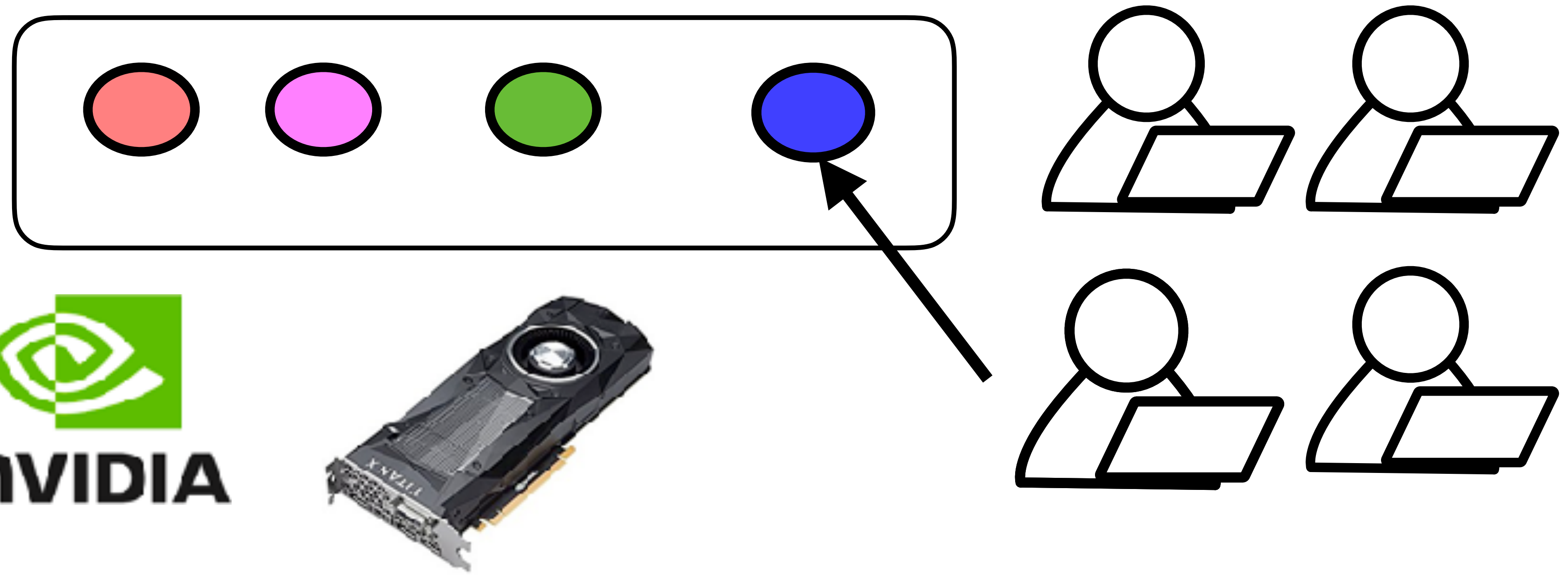


New operator introduced
by operator fusion optimization
potentially benefit: 1.5x speedup

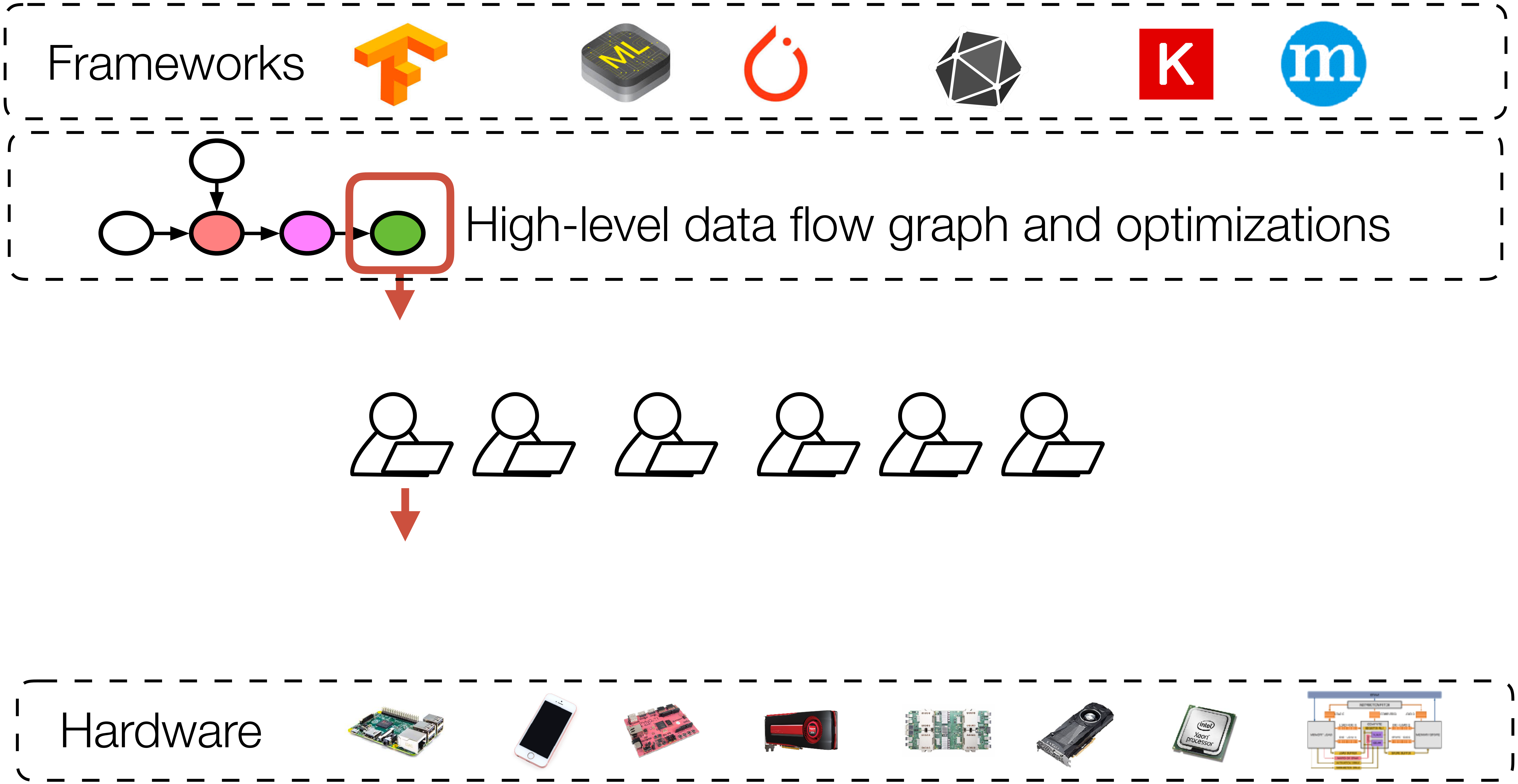
Engineering intensive



cuDNN

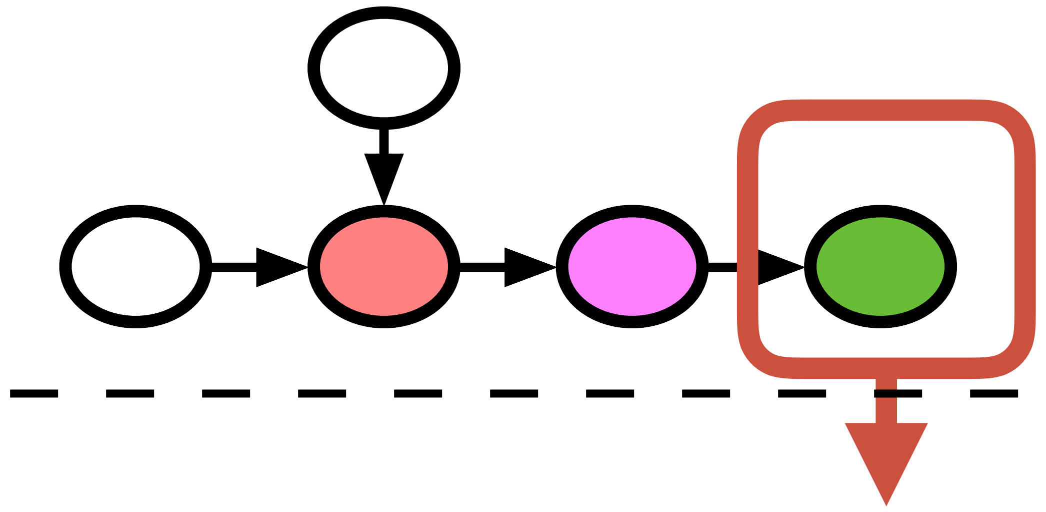


Learning-based Learning System



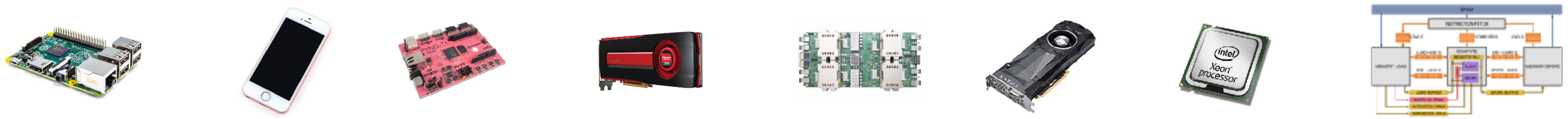
Learning-based Learning System

Frameworks

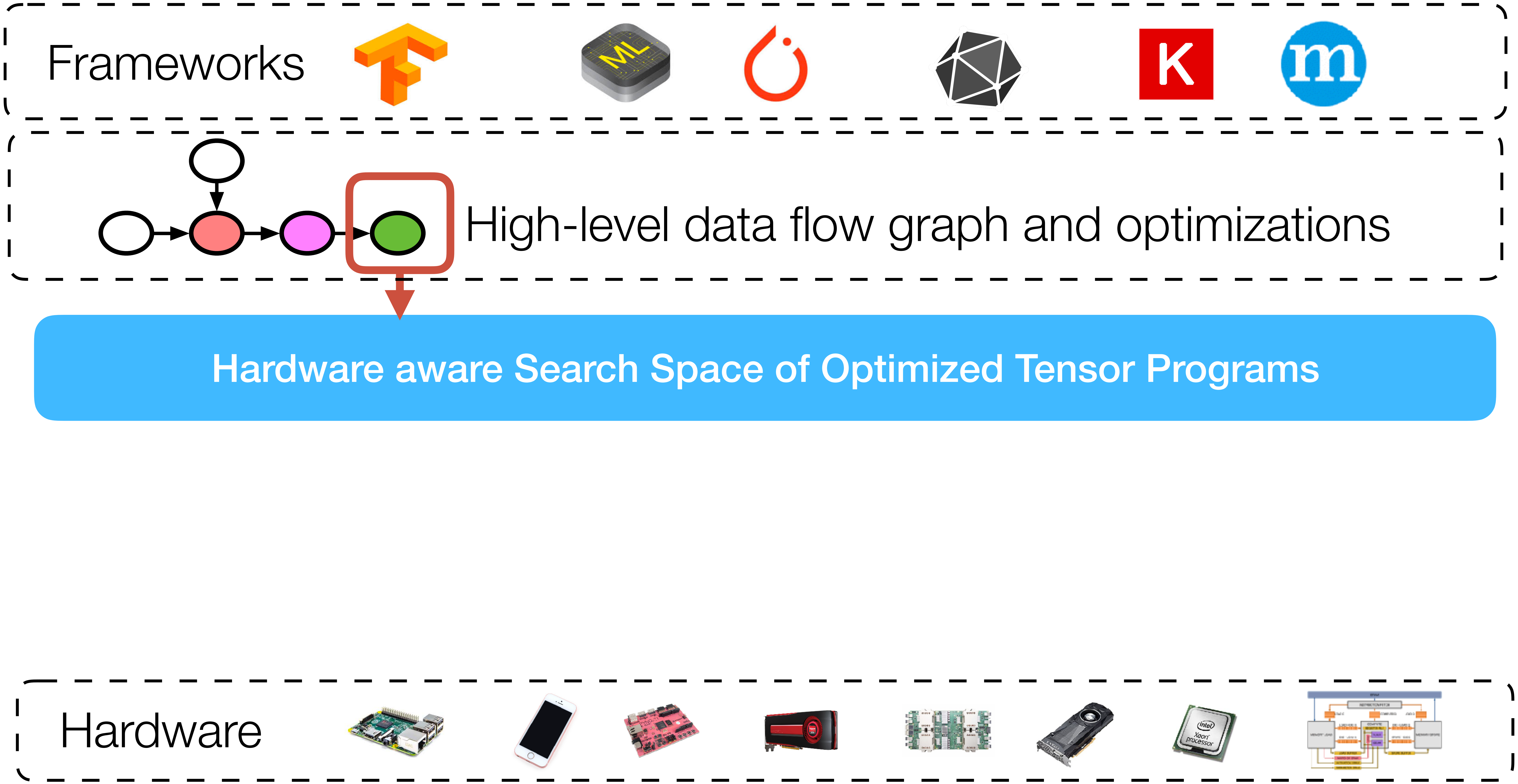


High-level data flow graph and optimizations

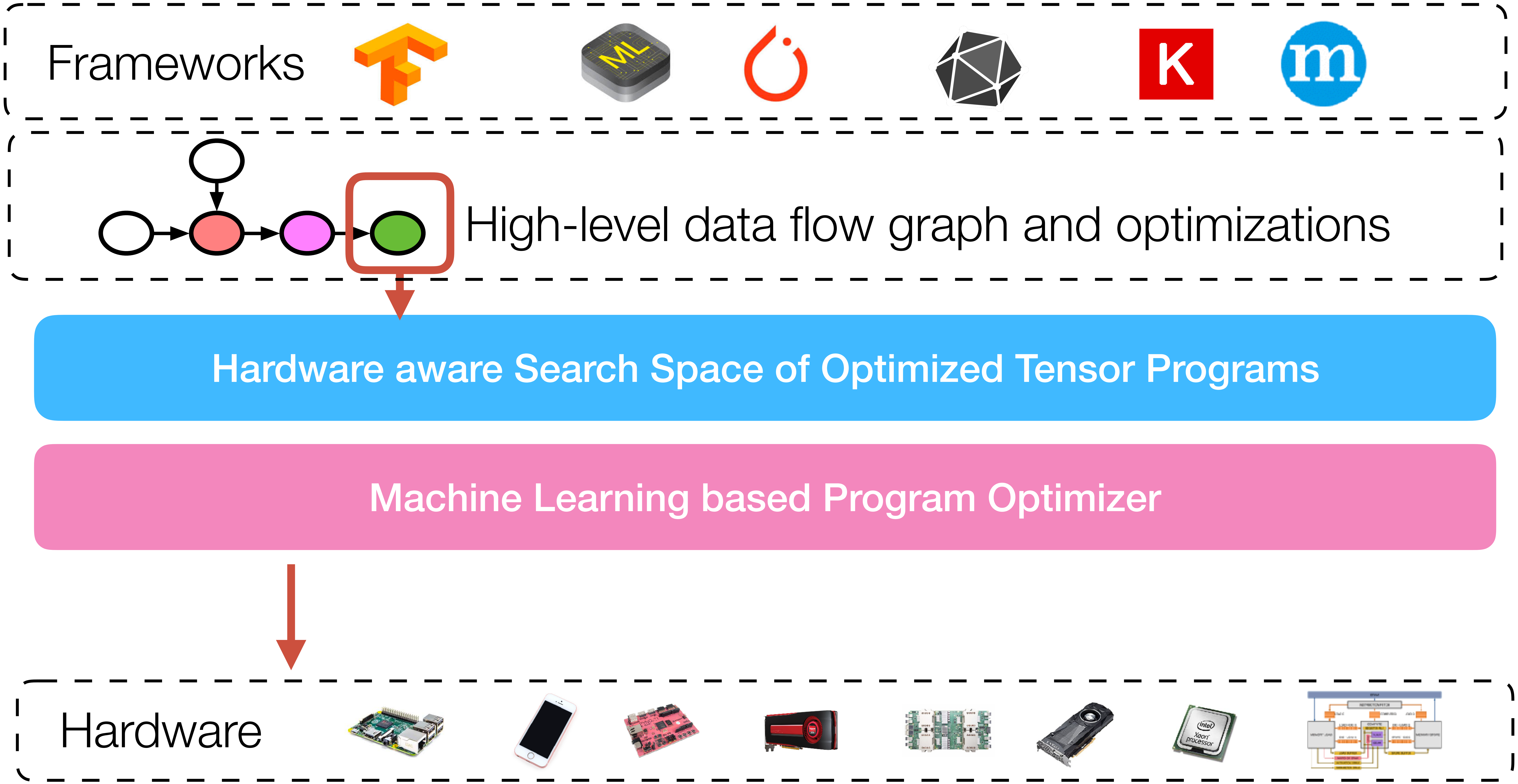
Hardware



Learning-based Learning System

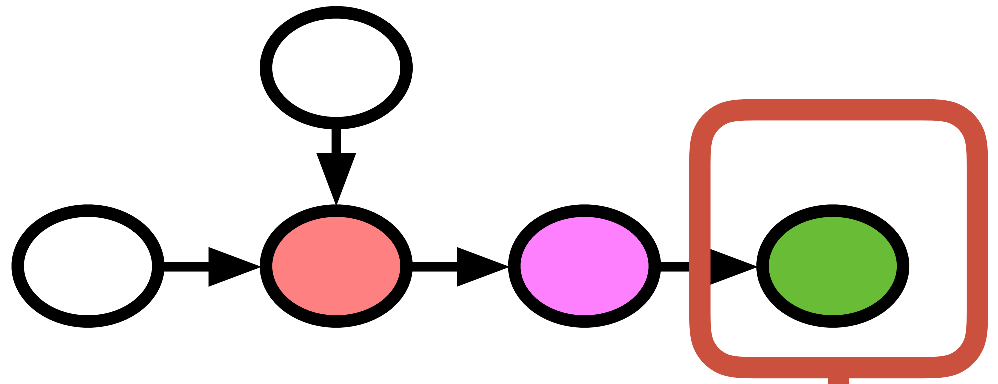


Learning-based Learning System



Learning-based Learning System

Frameworks



High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

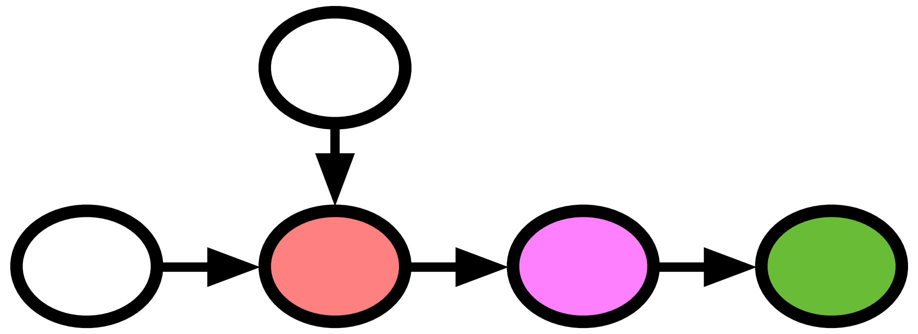
Machine Learning based Program Optimizer

directly generate optimized program
for new operator workloads and hardware

Hardware

Learning-based Learning System

Frameworks



High-level data flow graph and optimizations

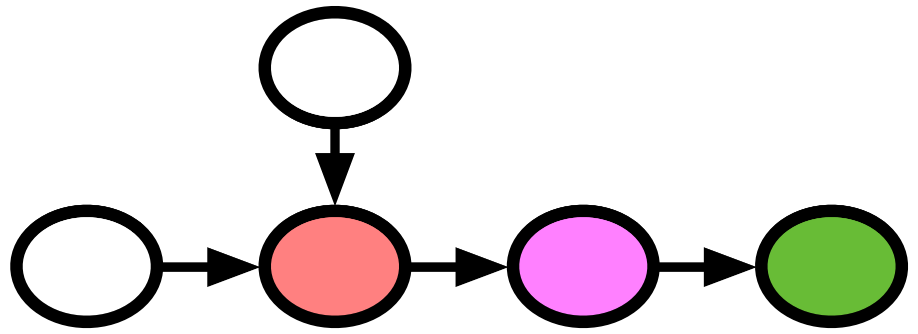
Hardware aware Search Space of Optimized Tensor Programs

Machine Learning based Program Optimizer

Hardware

Learning-based Learning System

Frameworks



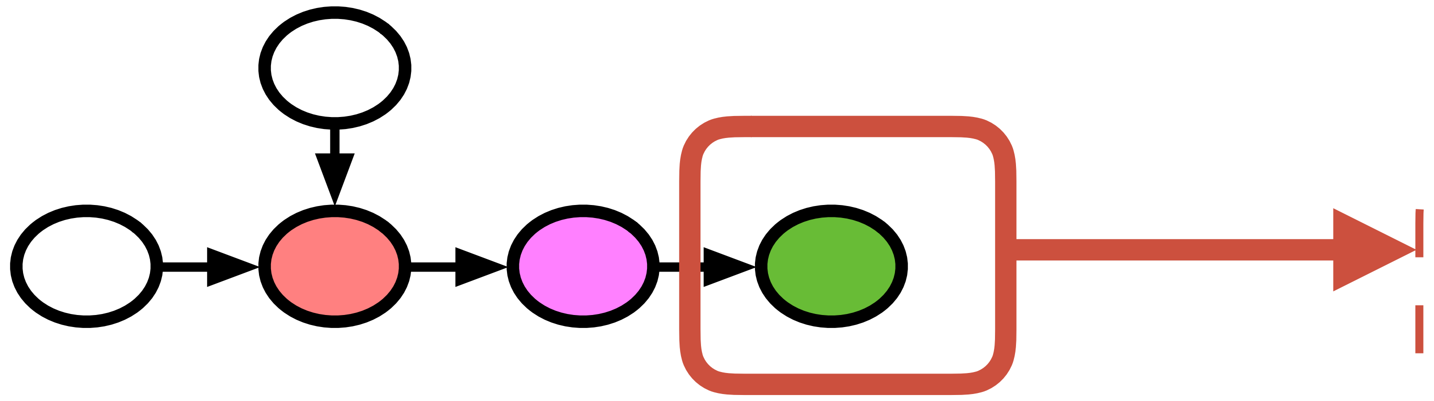
High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

Machine Learning based Program Optimizer

Hardware

Hardware-aware Search Space

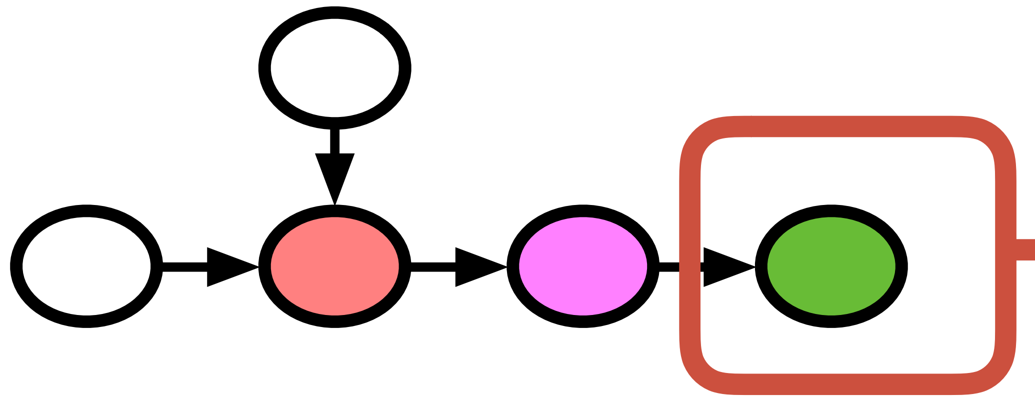


Tensor Expression Language (Specification)

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

Hardware

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

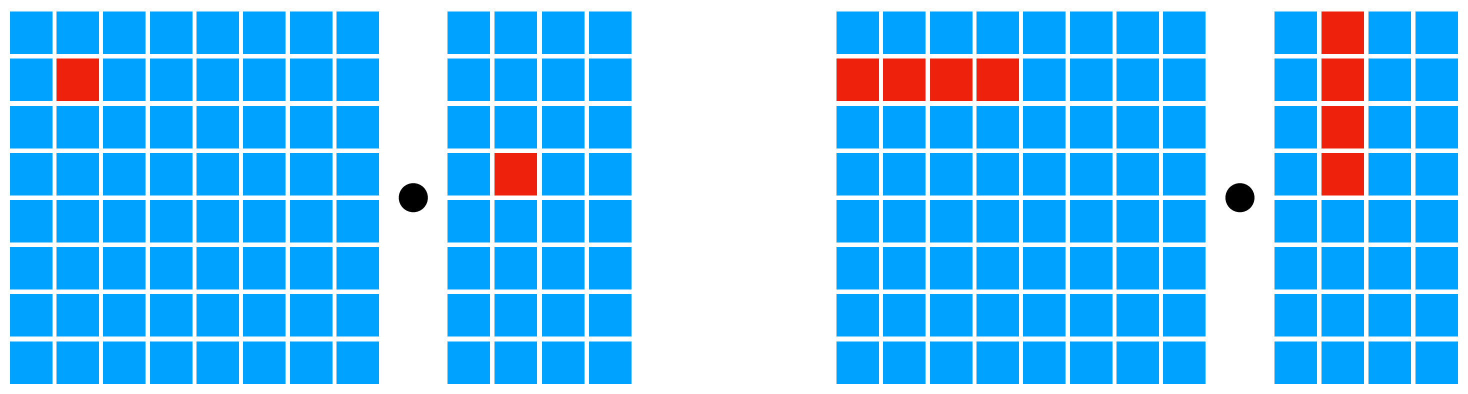
Based on Halide's compute/schedule separation

Hardware

Hardware-aware Search Space

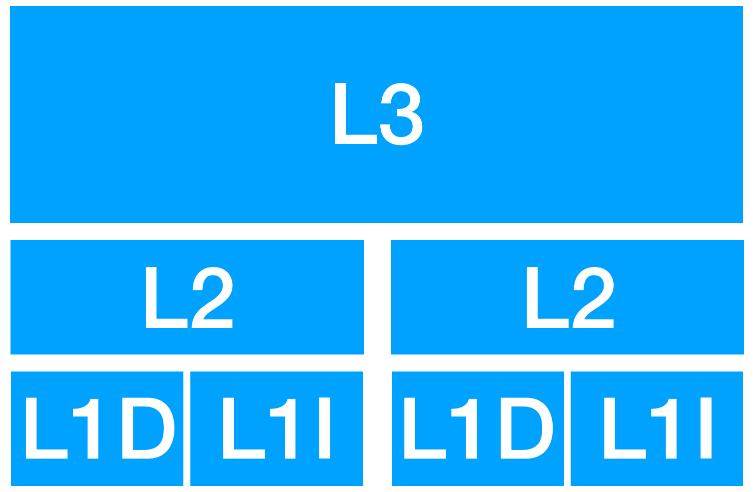
CPUs

Compute Primitives



scalar *vector*

Memory Subsystem



implicitly managed

Loop Transformations

Cache Locality

Vectorization

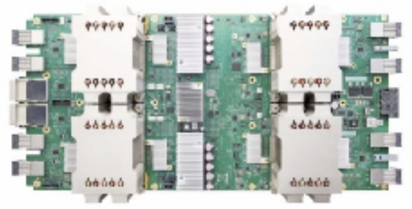
Reuse primitives from prior work:
Halide, Loopy

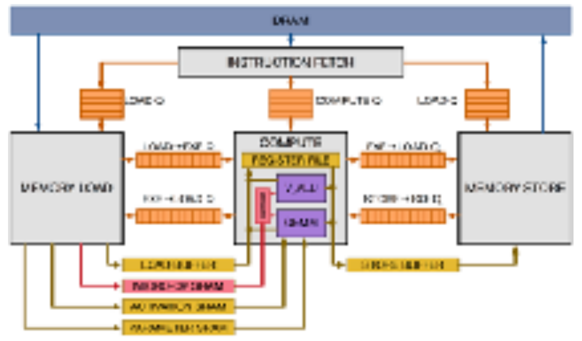
Challenge to Support Diverse Hardware Backends

CPUs

GPUs

TPU-like specialized Accelerators

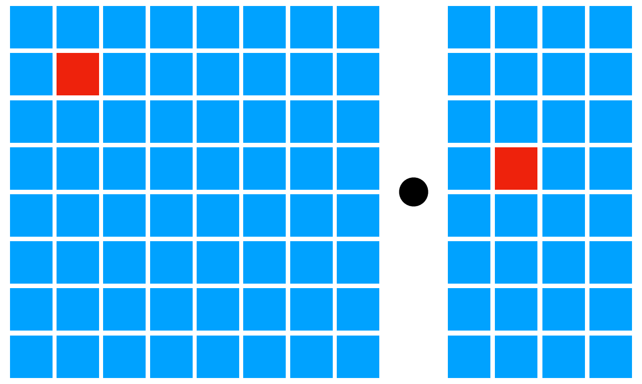




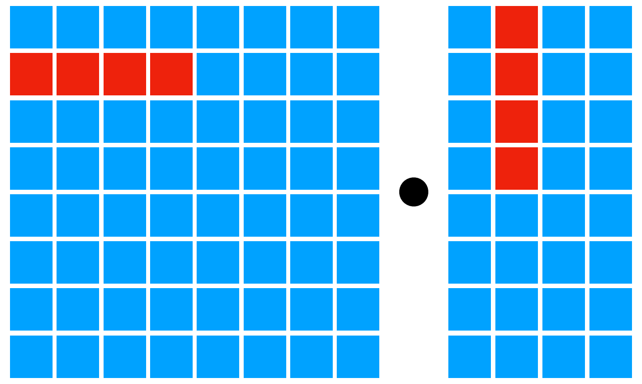
Hardware-aware Search Space

GPUs

Compute Primitives



scalar



vector

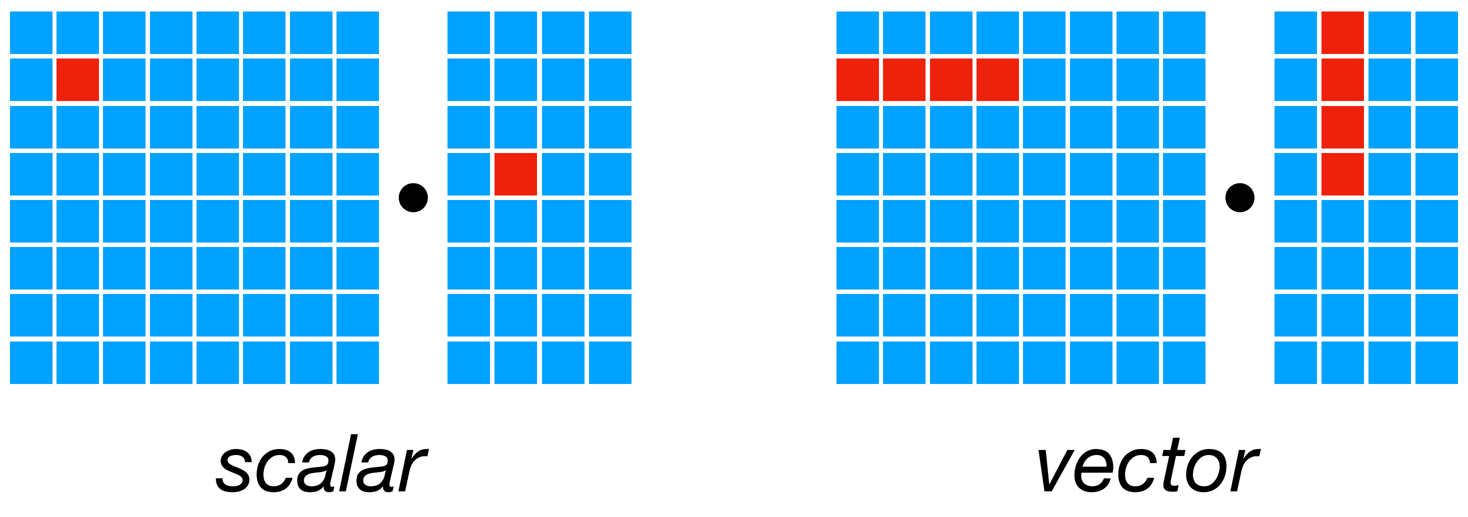
Memory Subsystem

mixed

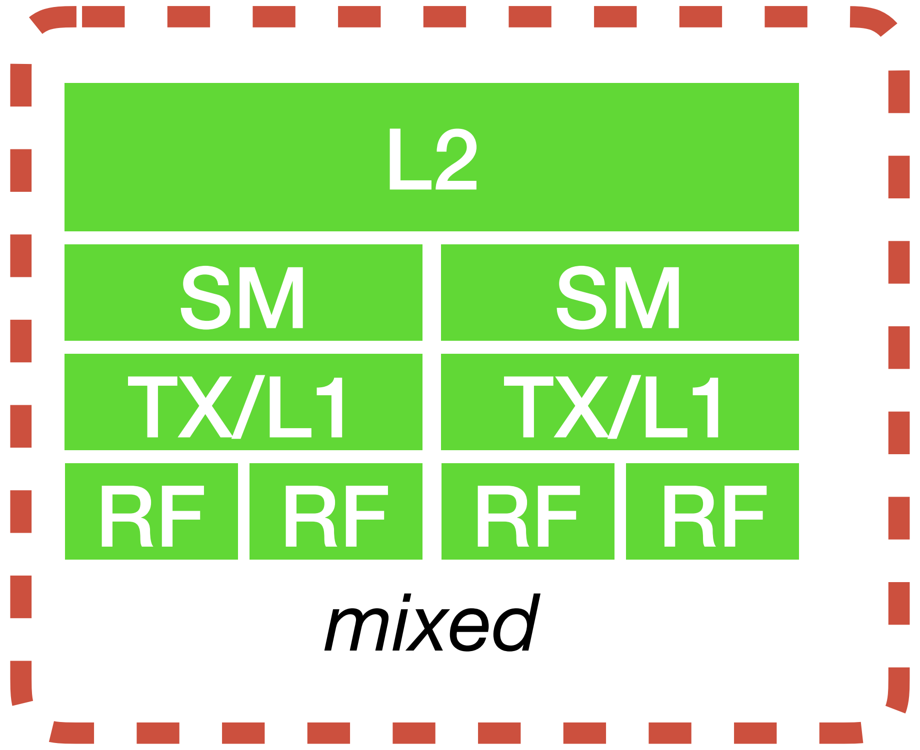
Hardware-aware Search Space

GPUs

Compute Primitives



Memory Subsystem

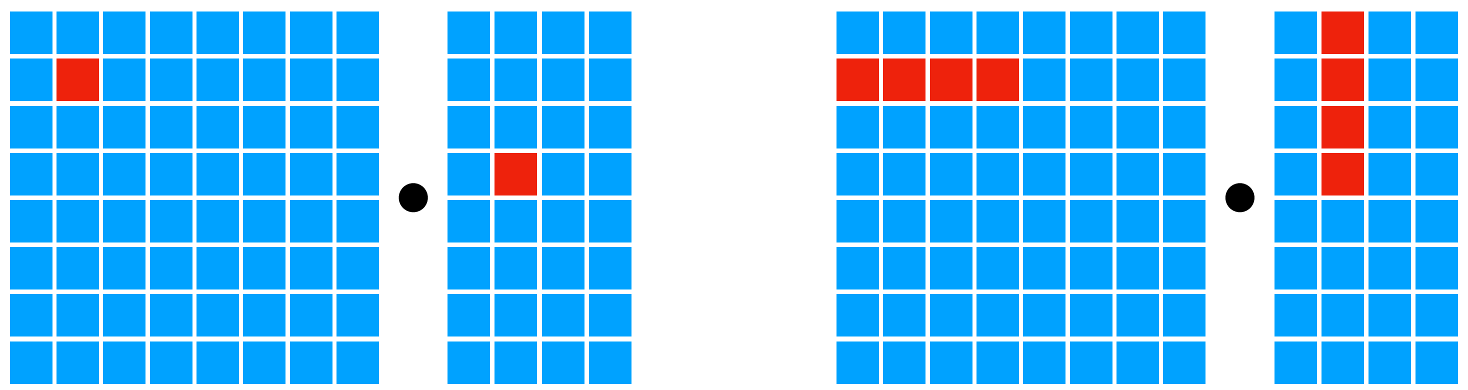


Shared memory among compute cores

Hardware-aware Search Space

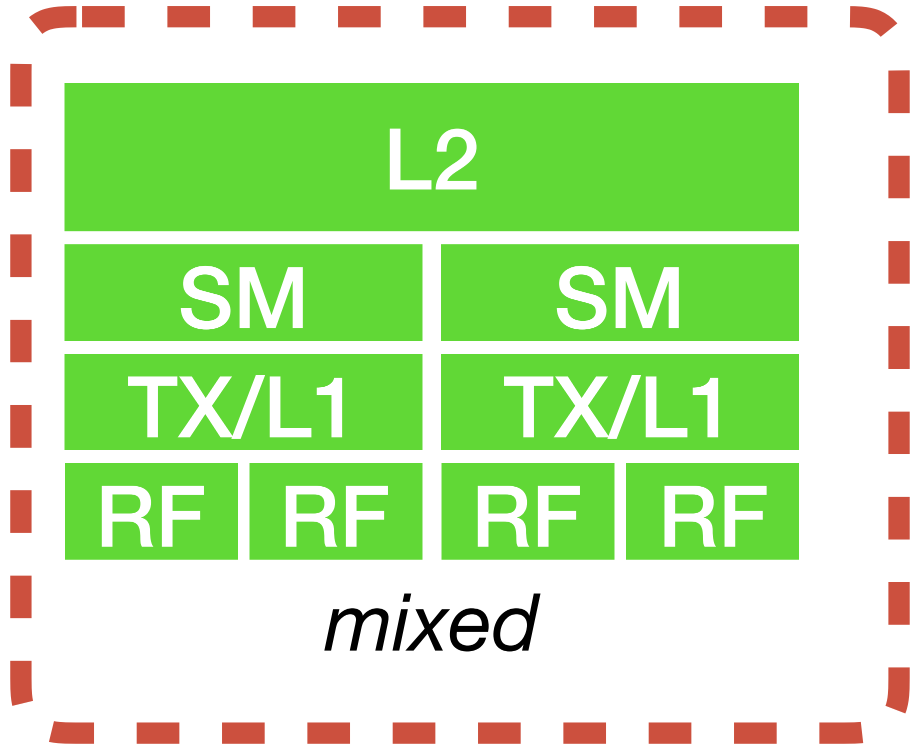
GPUs

Compute Primitives



scalar *vector*

Memory Subsystem



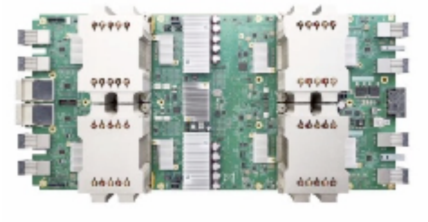
Shared memory among compute cores

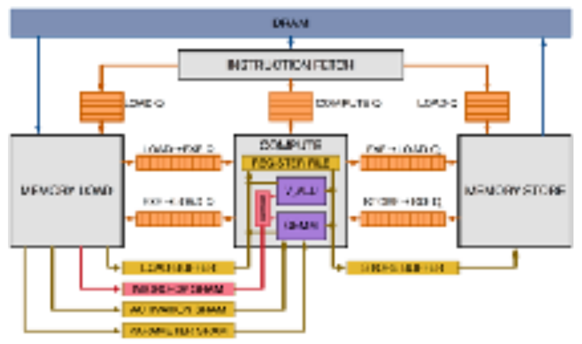
Use of Shared Memory

Thread Cooperation

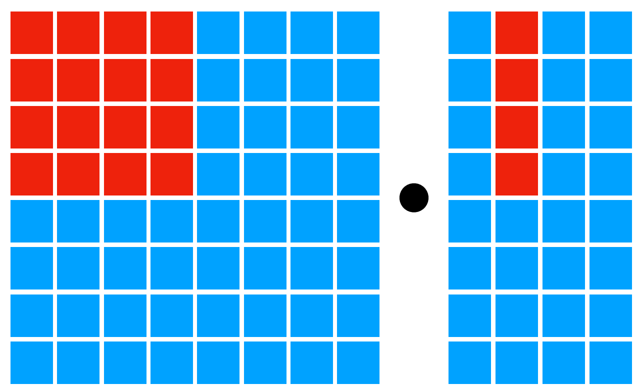
Hardware-aware Search Space

TPU-like Specialized Accelerators



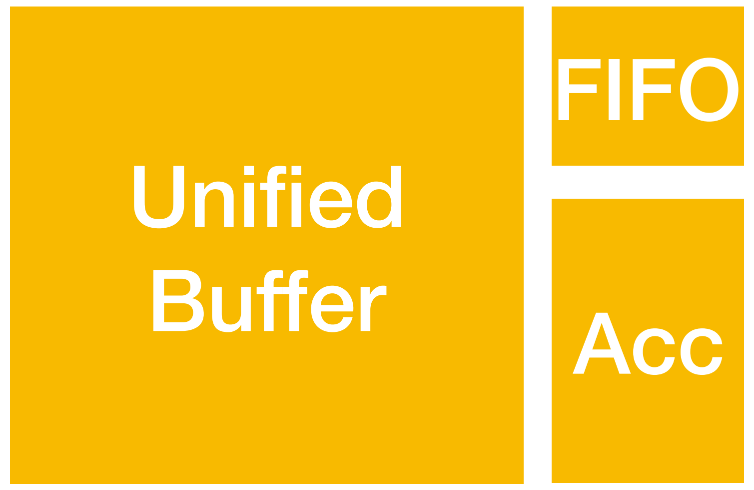


Compute Primitives



tensor

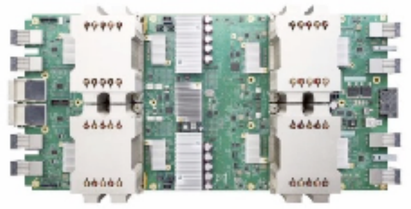
Memory Subsystem

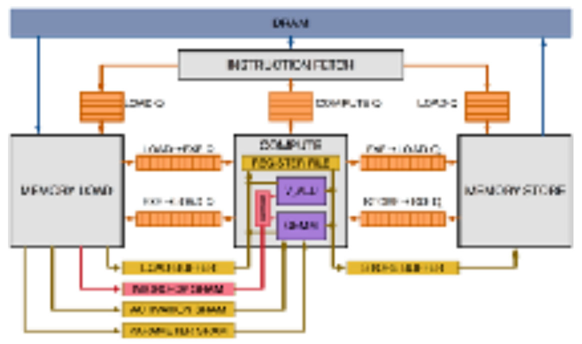


explicitly managed

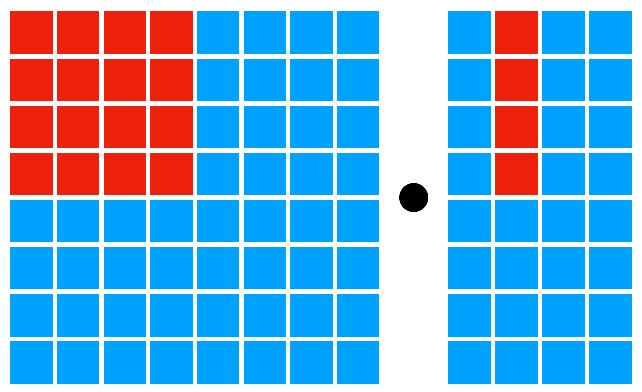
Hardware-aware Search Space

TPU-like Specialized Accelerators



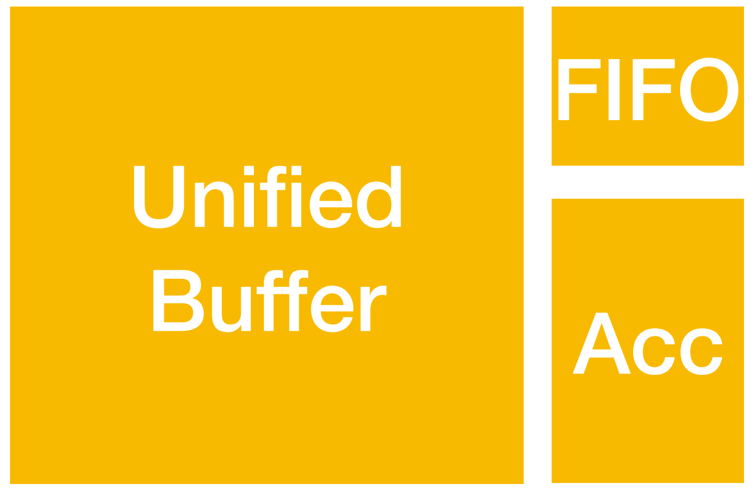


Compute Primitives



tensor

Memory Subsystem



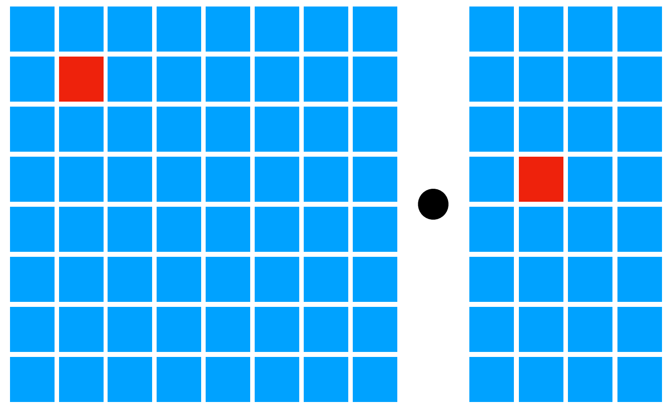
explicitly managed

Tensorization Challenge

Compute primitives

Tensorization Challenge

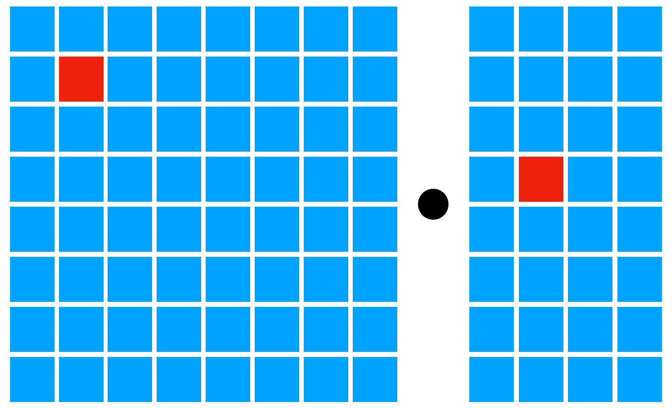
Compute primitives



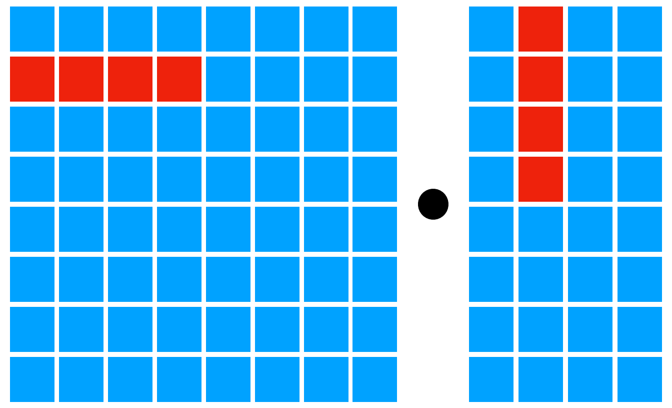
scalar

Tensorization Challenge

Compute primitives



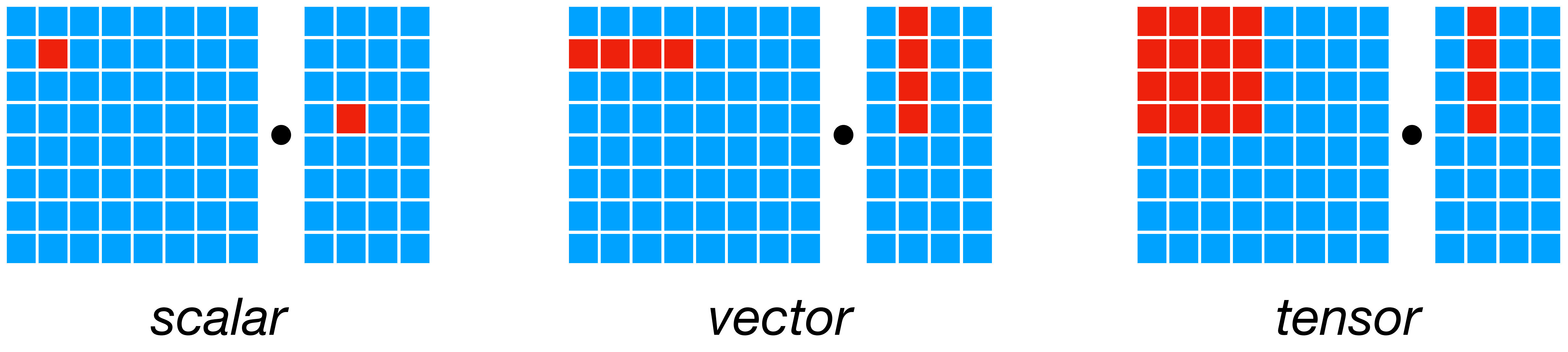
scalar



vector

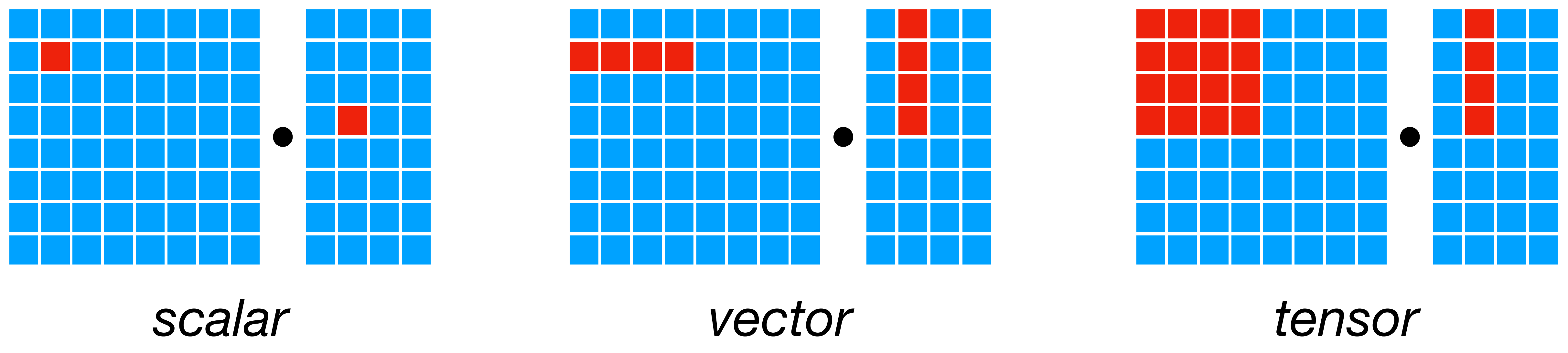
Tensorization Challenge

Compute primitives



Tensorization Challenge

Compute primitives



Hardware designer: declare tensor instruction interface with Tensor Expression

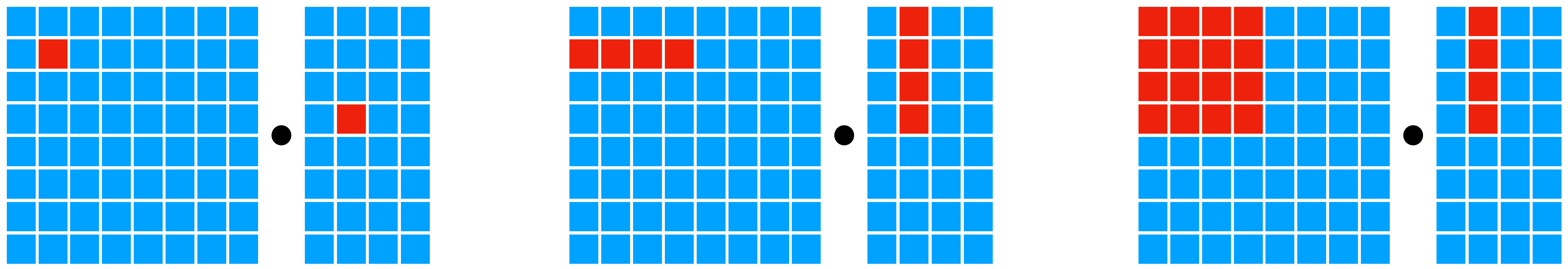
```
w, x = t.placeholder((8, 8)), t.placeholder((8, 8))
k = t.reduce_axis((0, 8))
y = t.compute((8, 8), lambda i, j:
              t.sum(w[i, k] * x[j, k], axis=k))

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

Tensorization Challenge

Compute primitives



scalar *vector* *tensor*

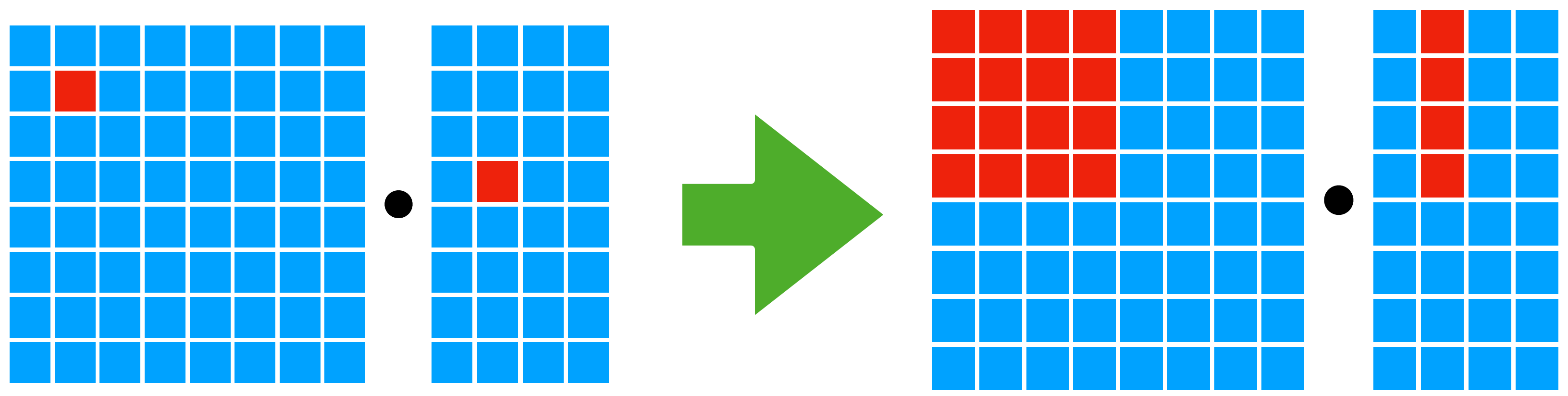
Hardware designer: declare tensor instruction interface with Tensor Expression

```
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

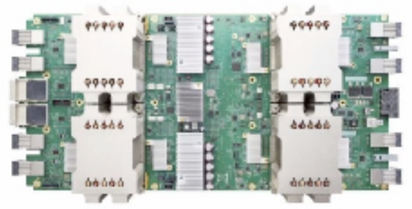
Tensorize: transform program to use tensor instructions

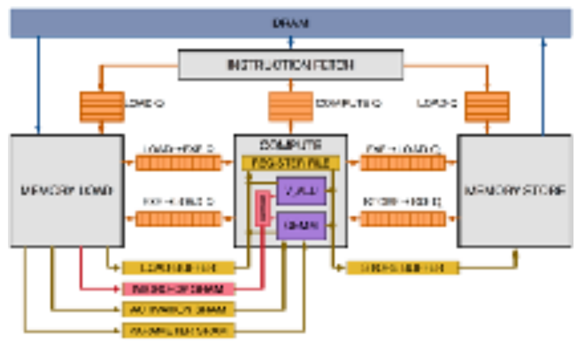


scalar *tensor*

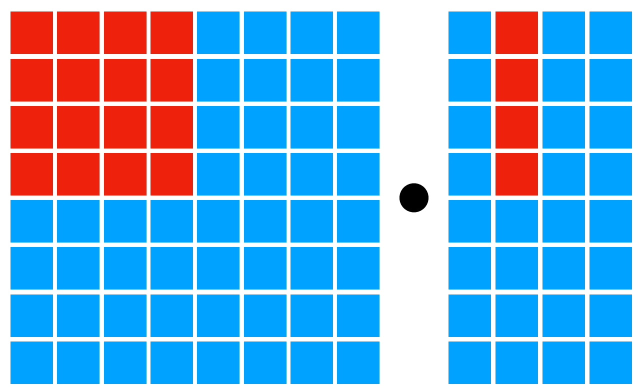
Hardware-aware Search Space

TPU-like Specialized Accelerators



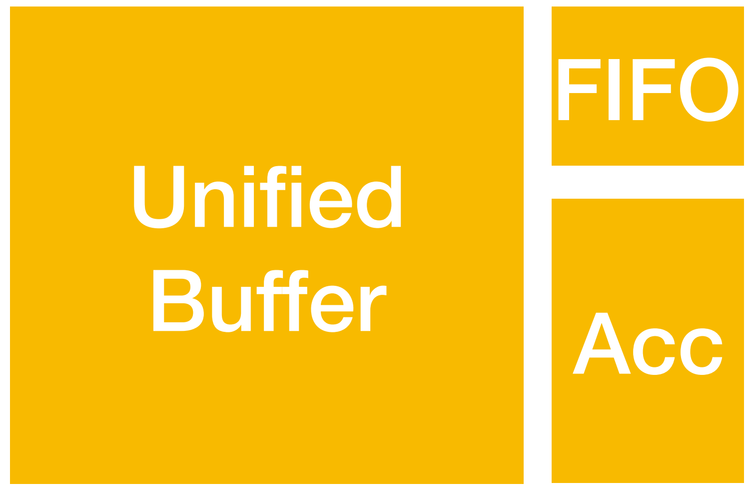


Compute Primitives



tensor

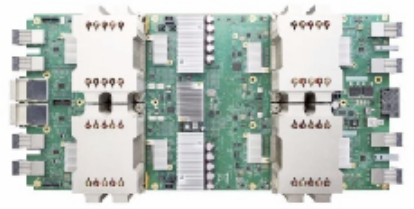
Memory Subsystem

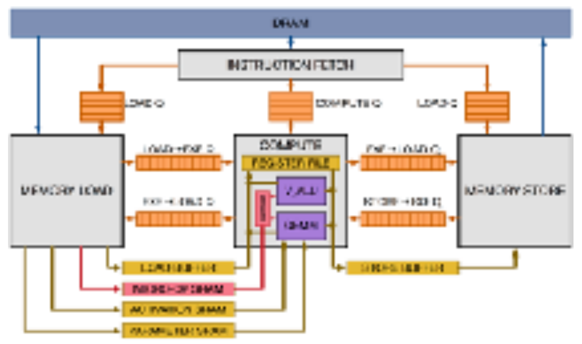


explicitly managed

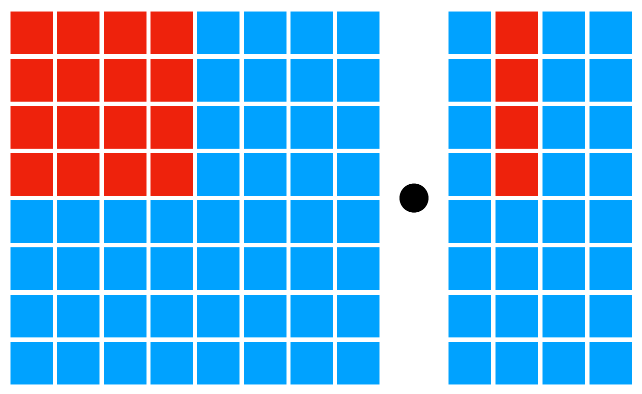
Hardware-aware Search Space

TPU-like Specialized Accelerators



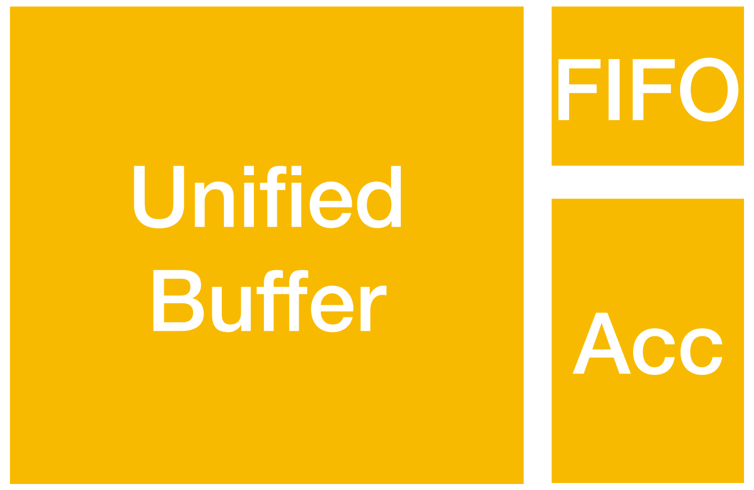


Compute Primitives



tensor

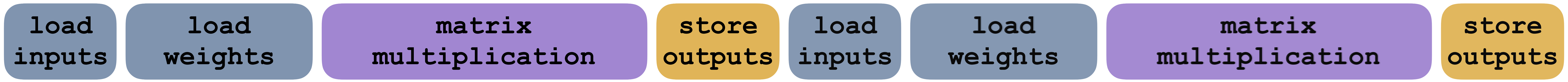
Memory Subsystem



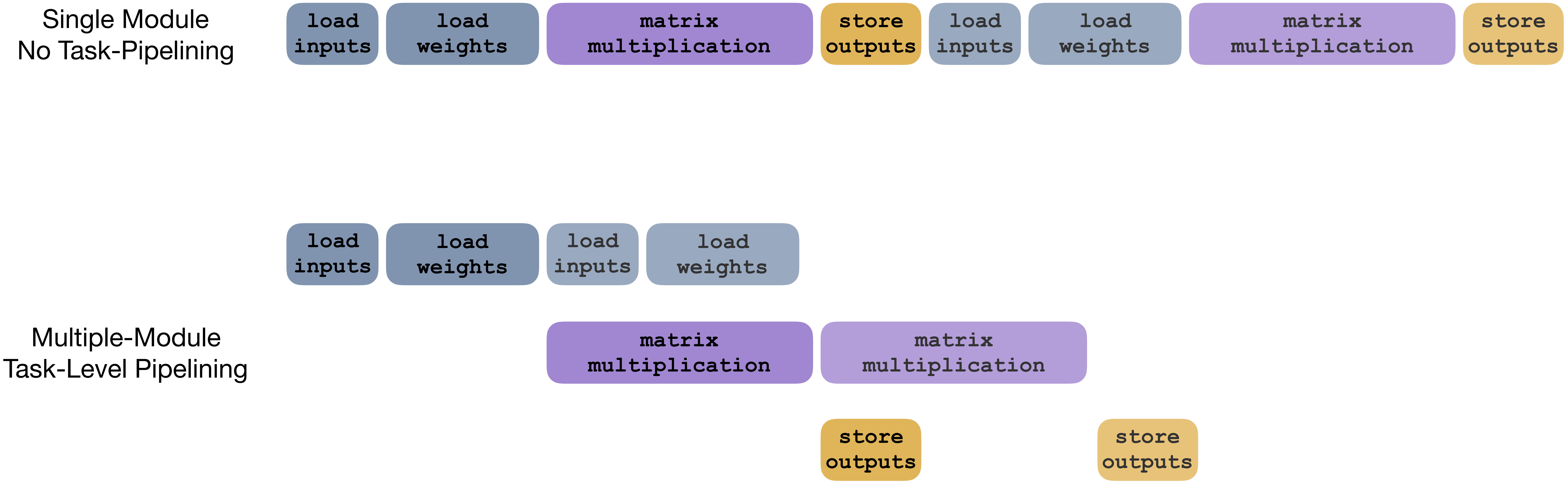
explicitly managed

Software Support for Latency Hiding

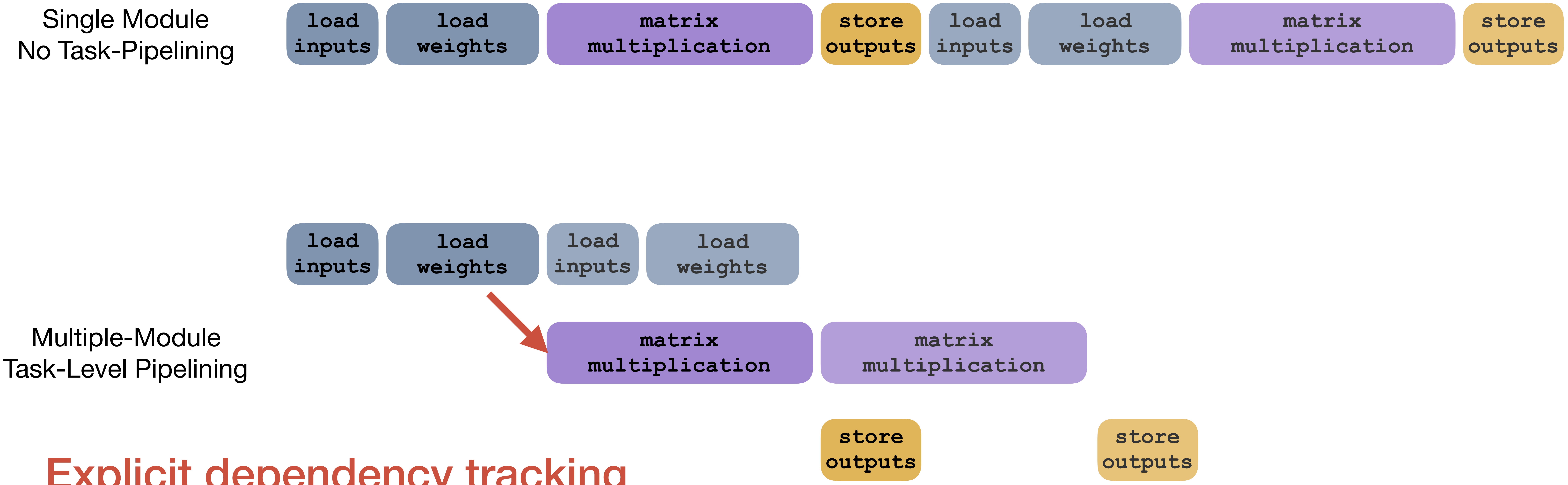
Single Module
No Task-Pipelining



Software Support for Latency Hiding

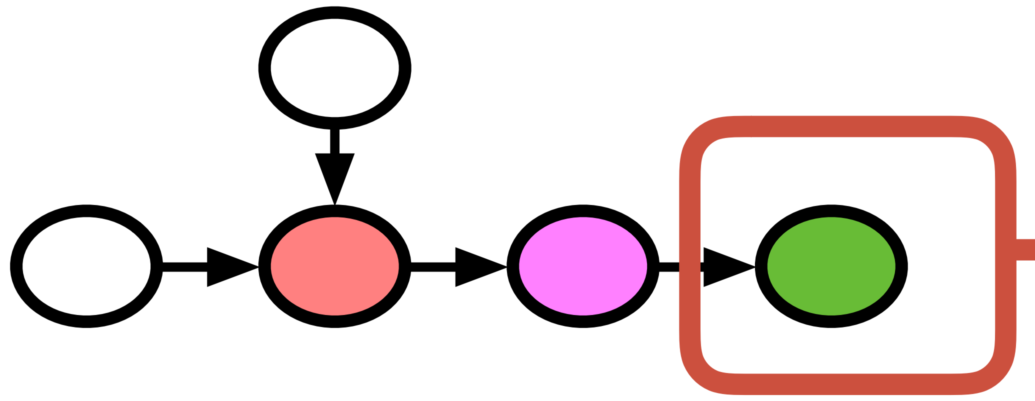


Software Support for Latency Hiding



Explicit dependency tracking
managed by software to hide memory latency

Hardware-aware Search Space

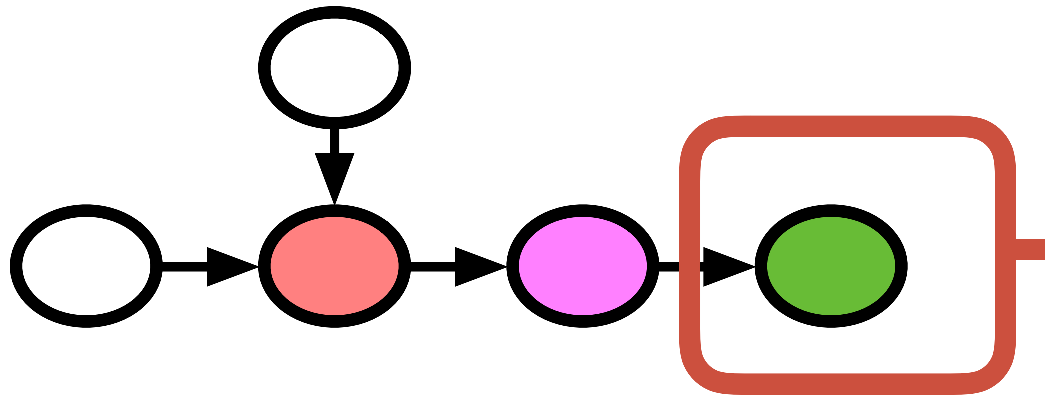


Tensor Expression Language

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

Hardware

Hardware-aware Search Space



Tensor Expression Language

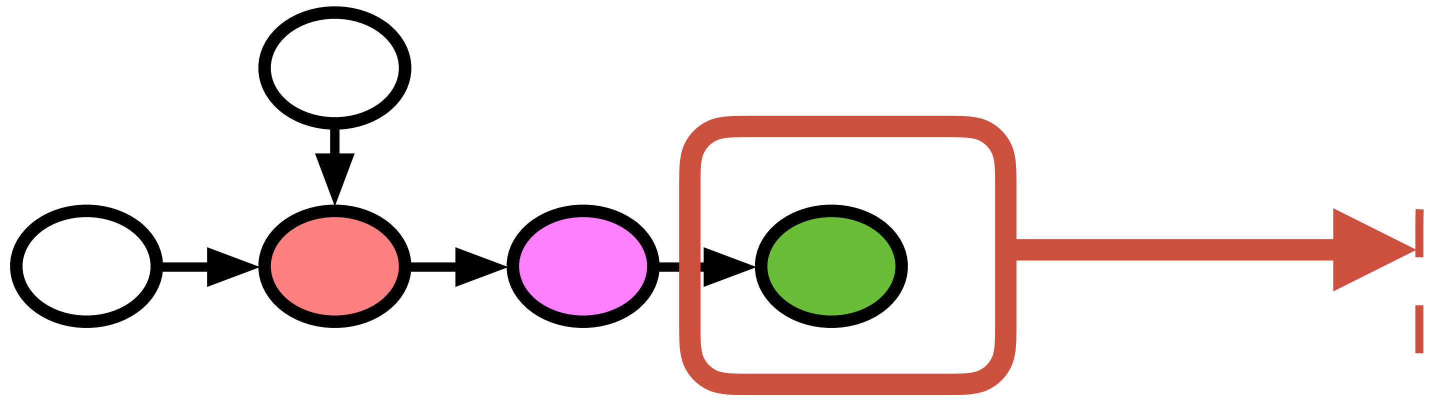
```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Primitives in prior work:
Halide, Loopy

Loop Transformations | Thread Bindings | Cache Locality

Hardware

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

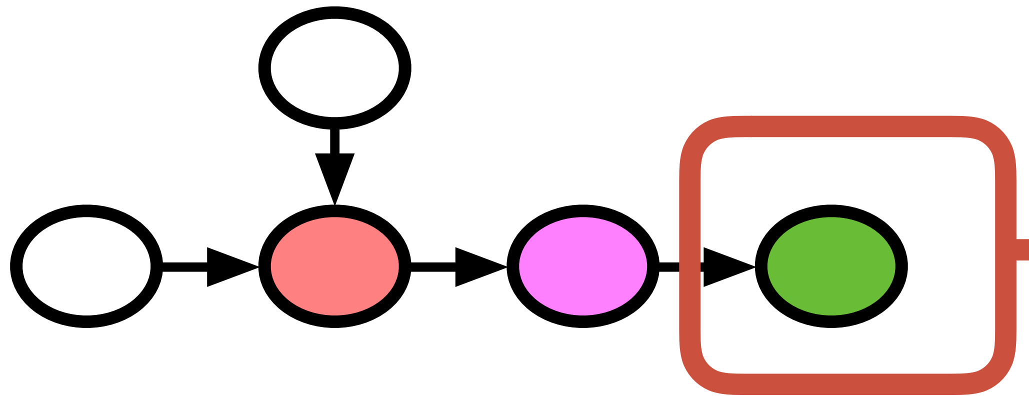
- Loop Transformations
- Thread Bindings
- Cache Locality

New primitives for GPUs, and enable TPU-like Accelerators

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

Hardware

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))
```

Loop Transformations

Thread Bindings

Cache Locality

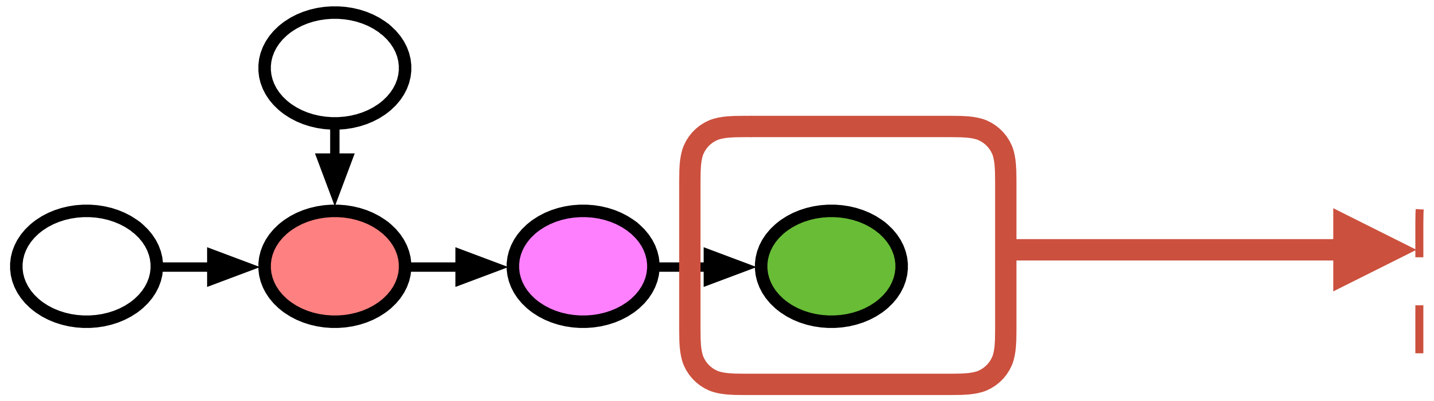
Thread Cooperation

Tensorization

Latency Hiding

Hardware

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))
```

Loop Transformations

Thread Bindings

Cache Locality

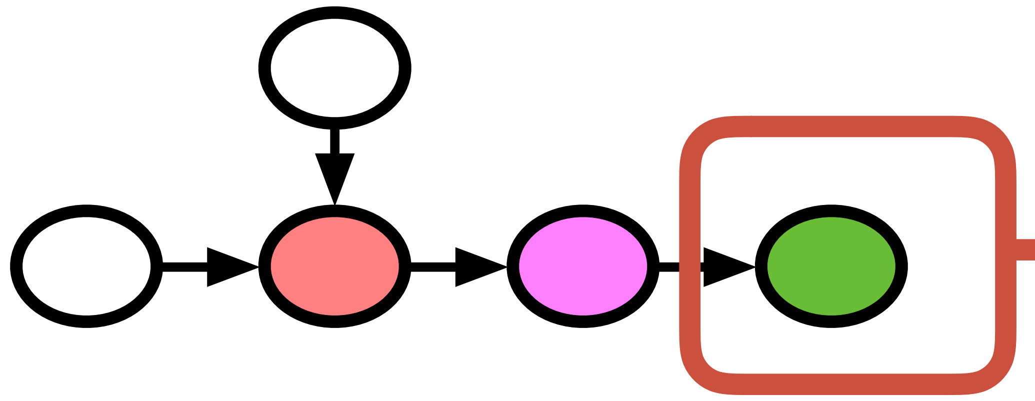
Thread Cooperation

Tensorization

Latency Hiding

Hardware

Hardware-aware Search Space



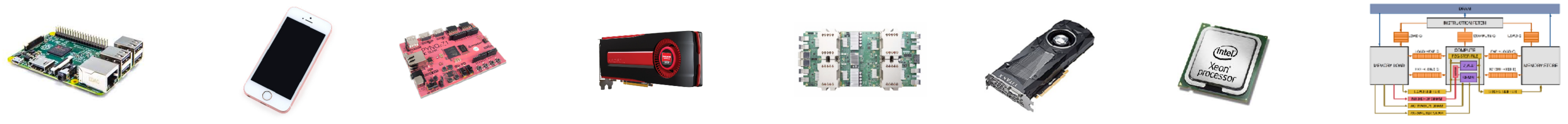
Tensor Expression Language

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

Billions of possible optimization choices

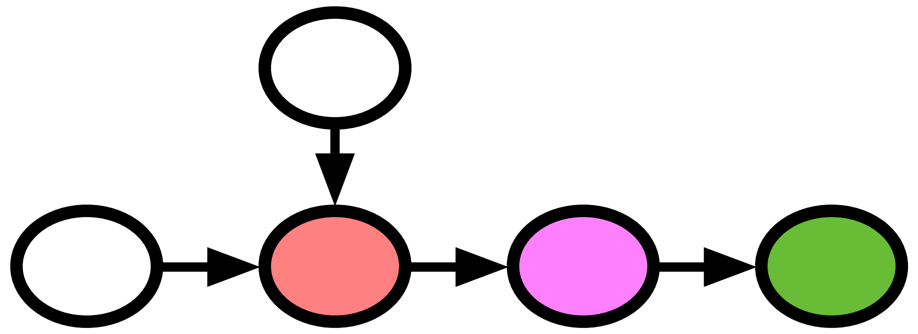
- Loop Transformations
- Thread Bindings
- Cache Locality
- Thread Cooperation
- Tensorization
- Latency Hiding

Hardware



Learning-based Learning System

Frameworks



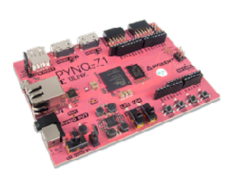
High-level data flow graph and optimizations

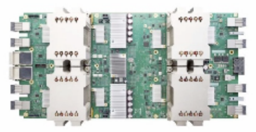
Hardware aware Search Space of Optimized Tensor Programs

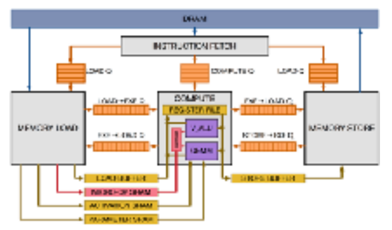
Machine Learning based Program Optimizer

Hardware



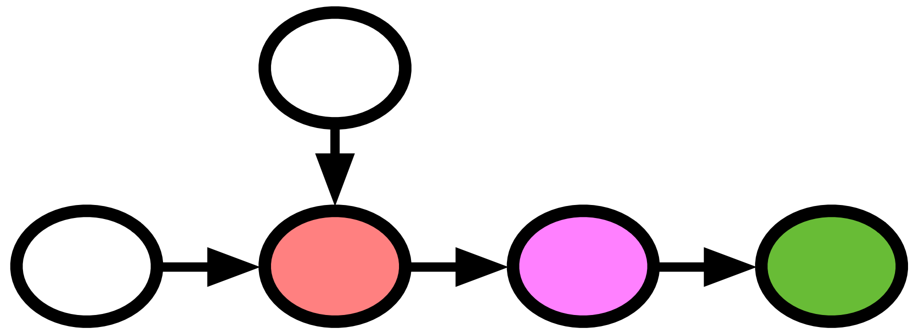






Learning-based Learning System

Frameworks



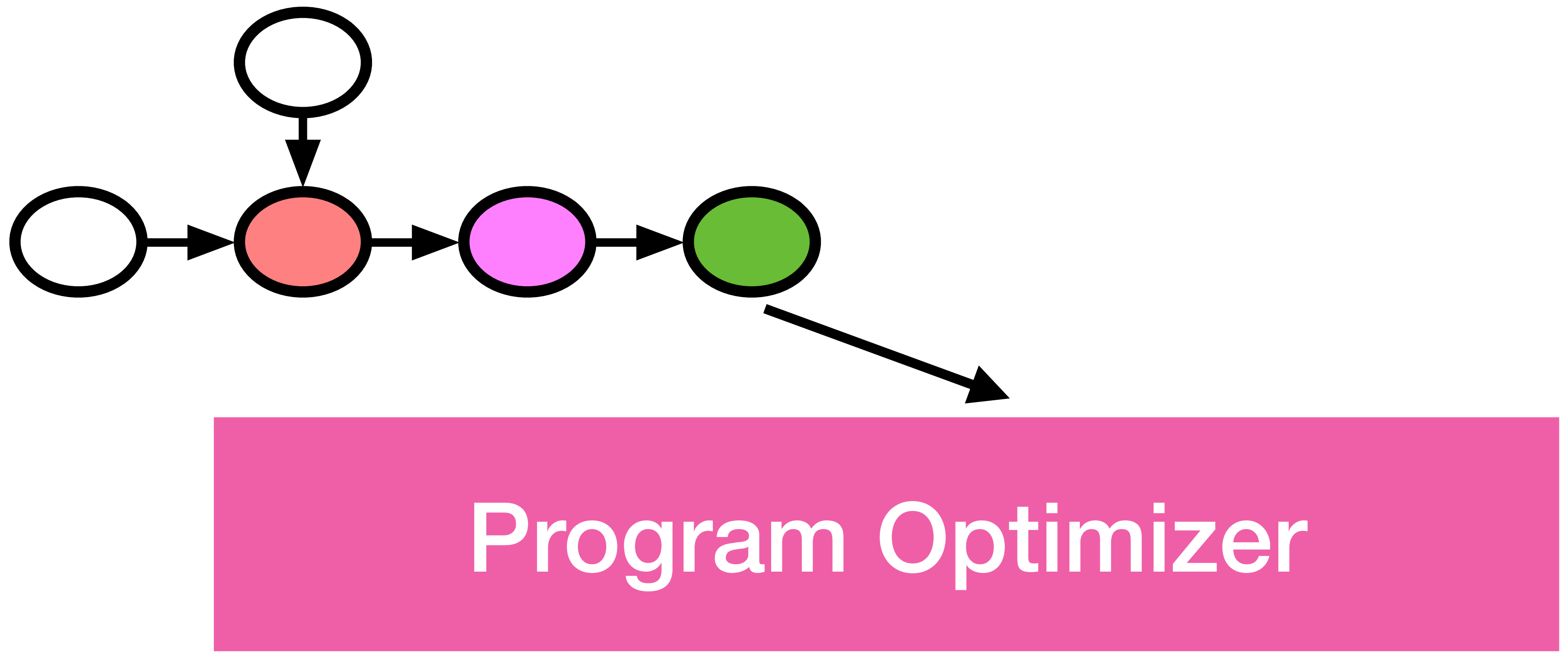
High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

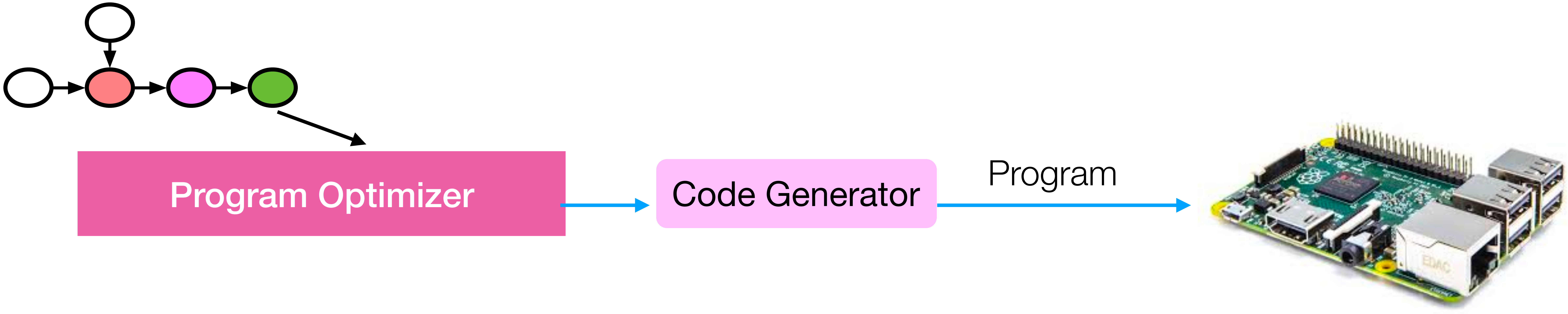
Machine Learning based Program Optimizer

Hardware

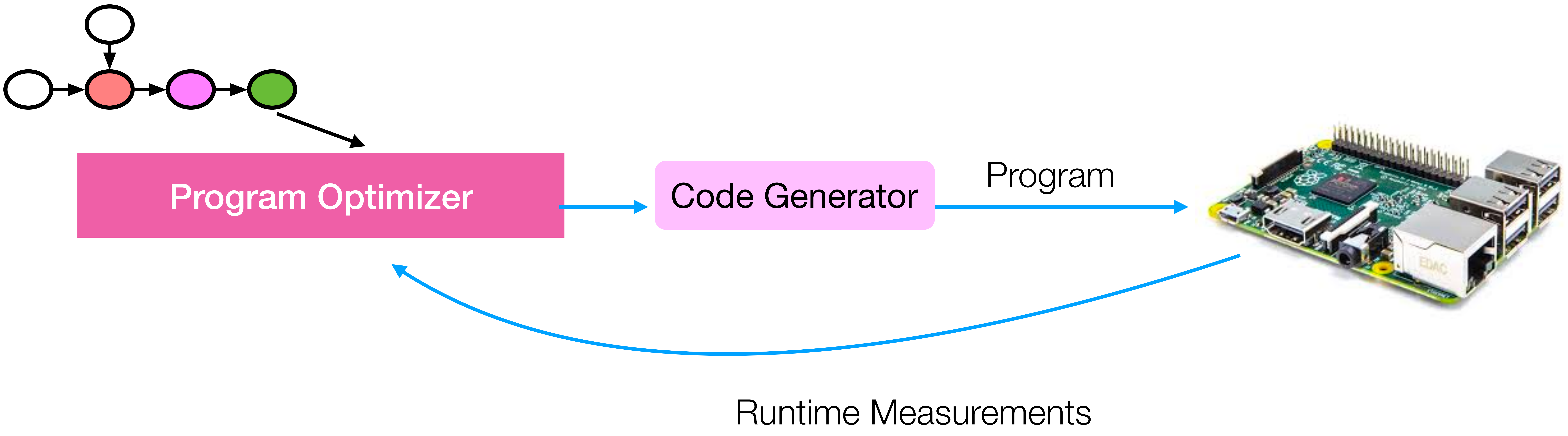
Learning-based Program Optimizer



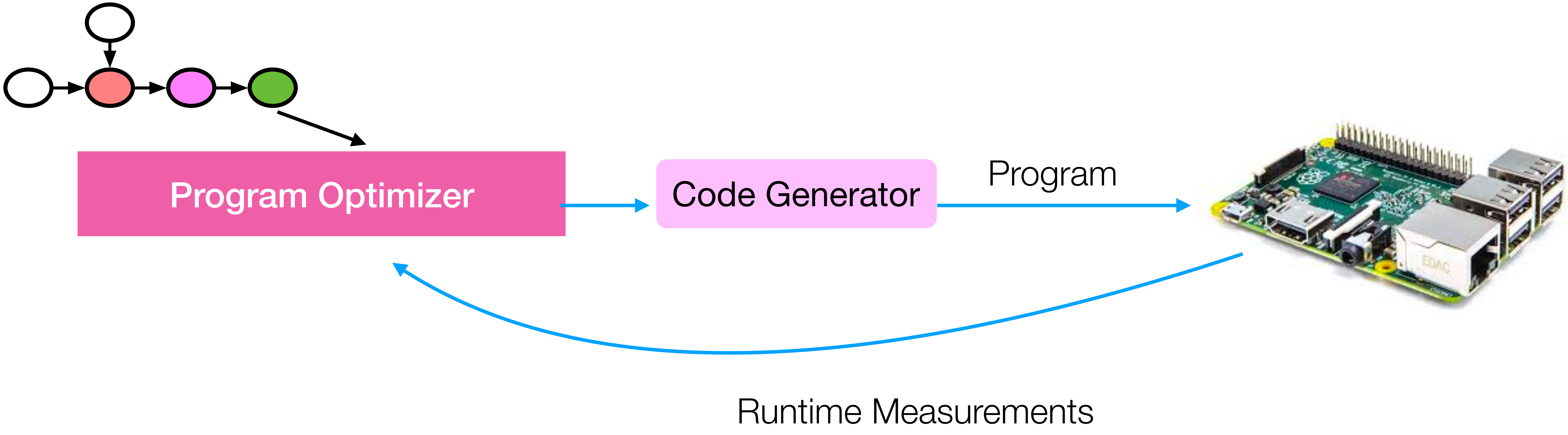
Learning-based Program Optimizer



Learning-based Program Optimizer

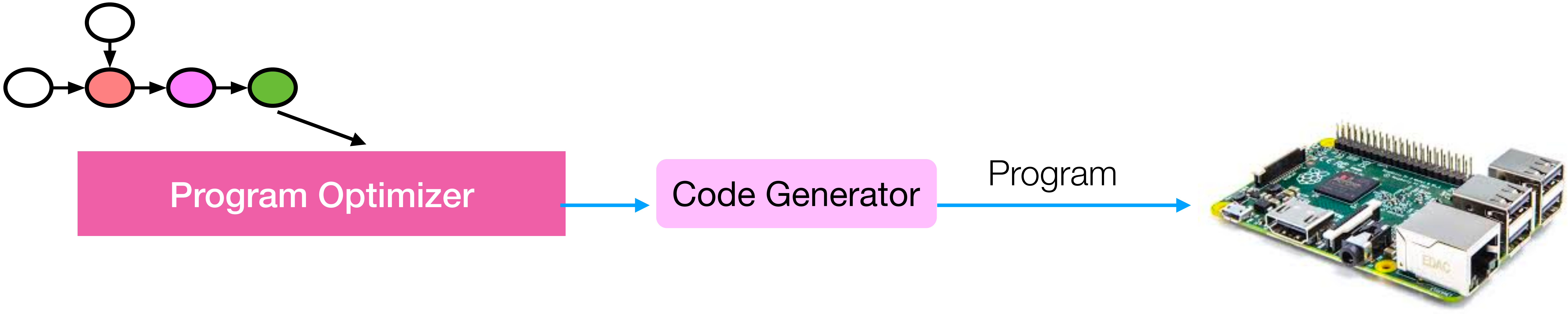


Learning-based Program Optimizer

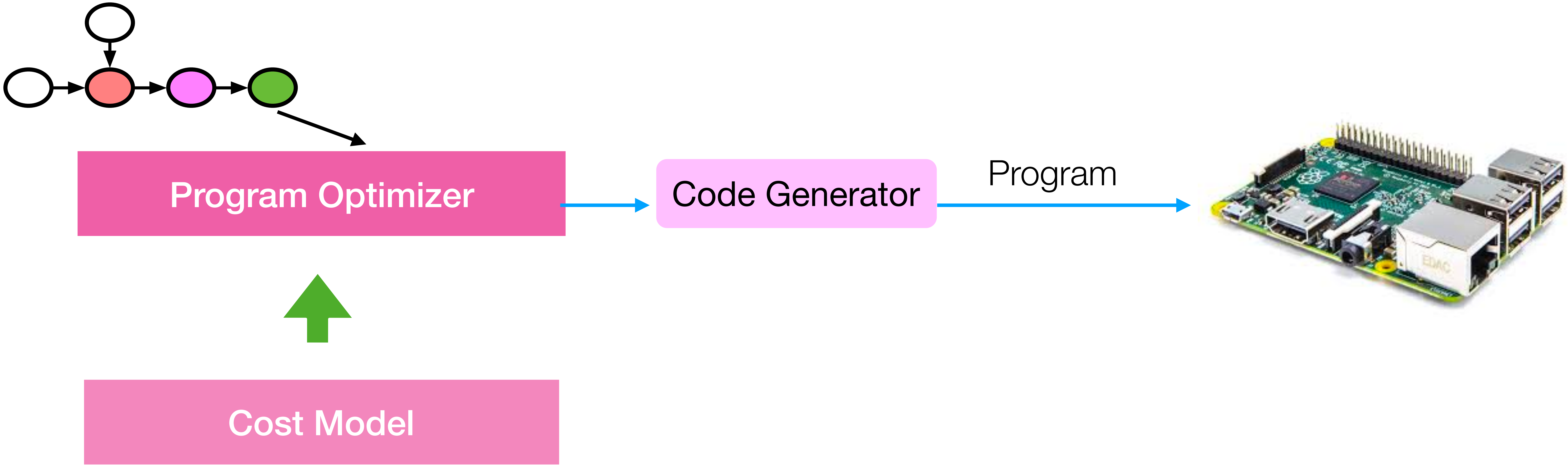


High experiment cost,
each trial costs ~1second

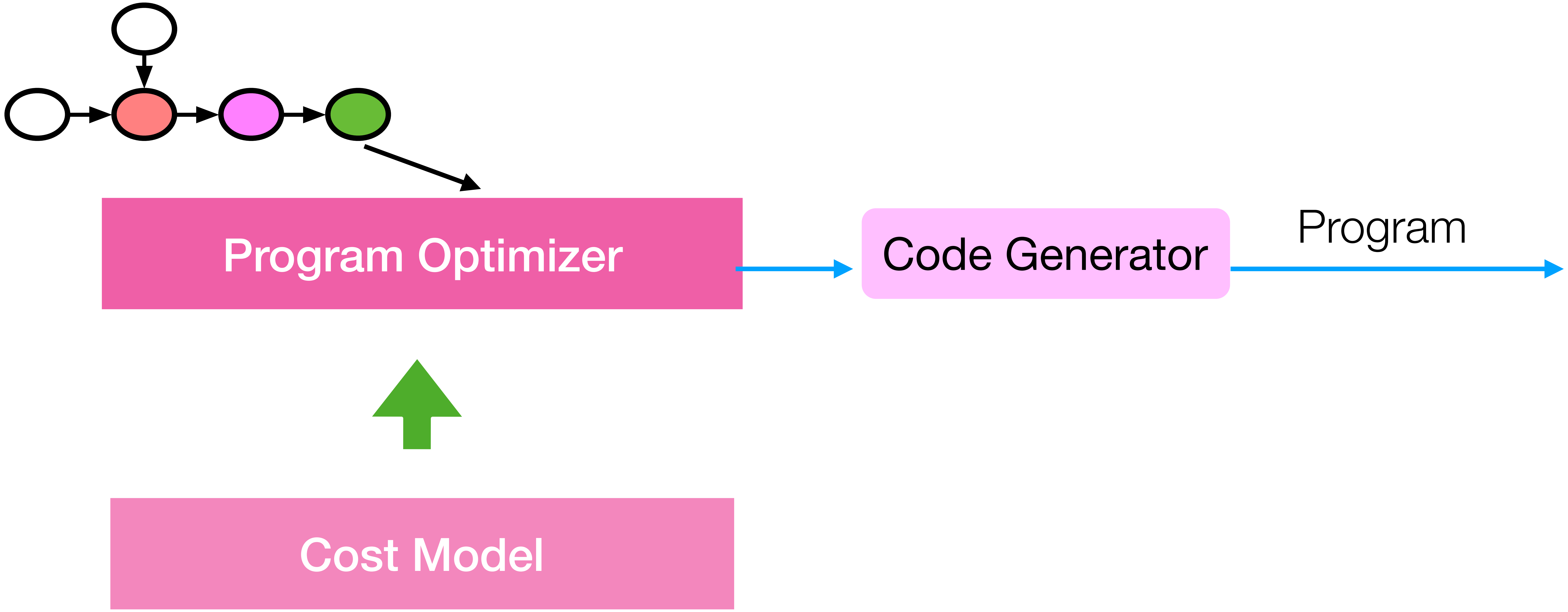
Learning-based Program Optimizer



Learning-based Program Optimizer

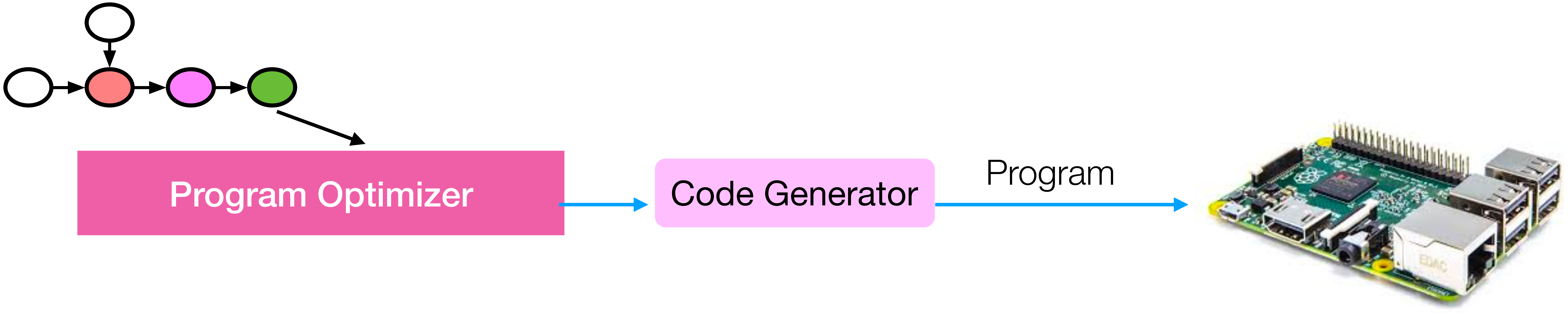


Learning-based Program Optimizer

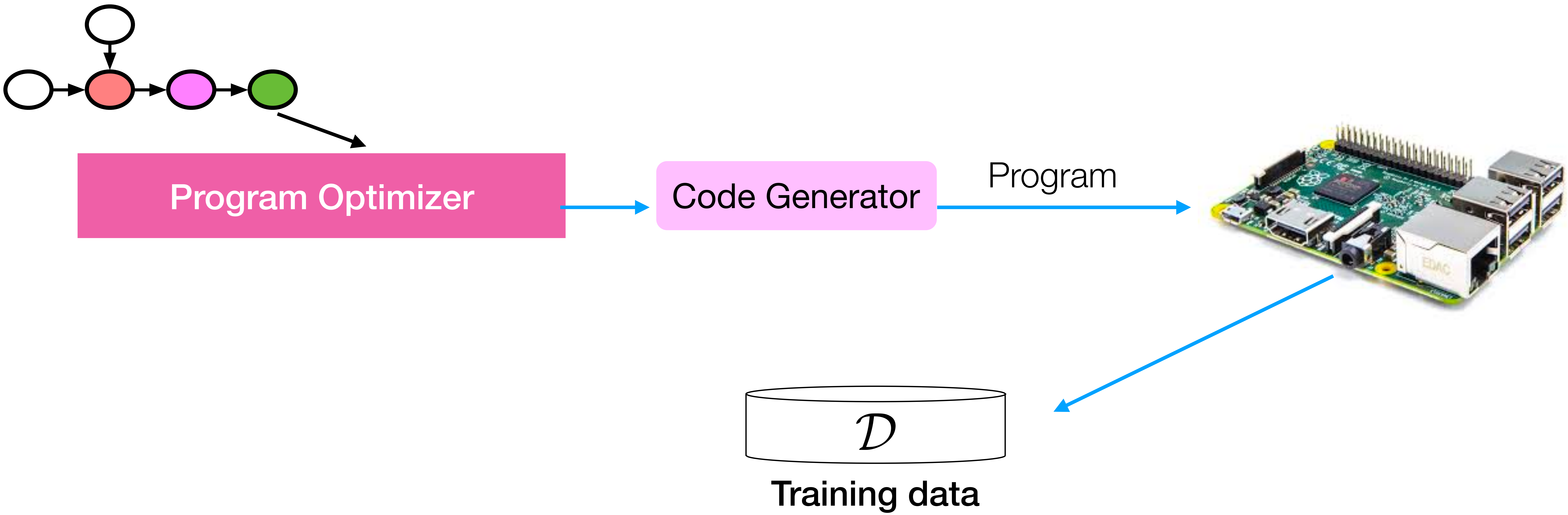


Need reliable cost model per hardware

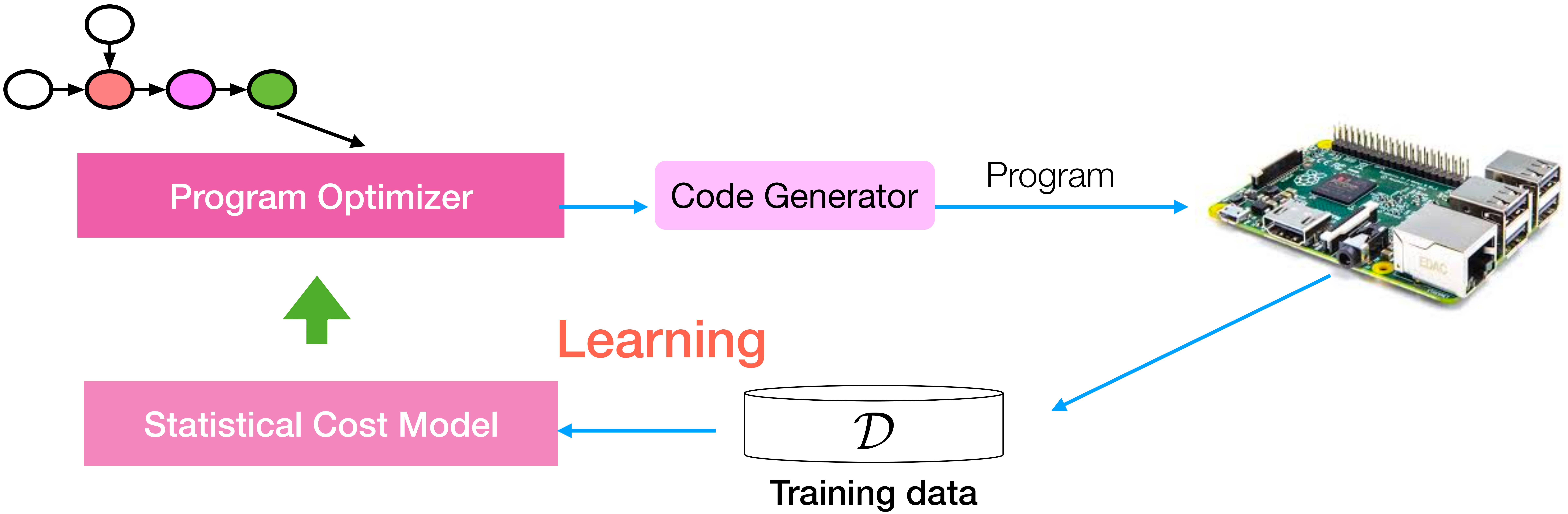
Learning-based Program Optimizer



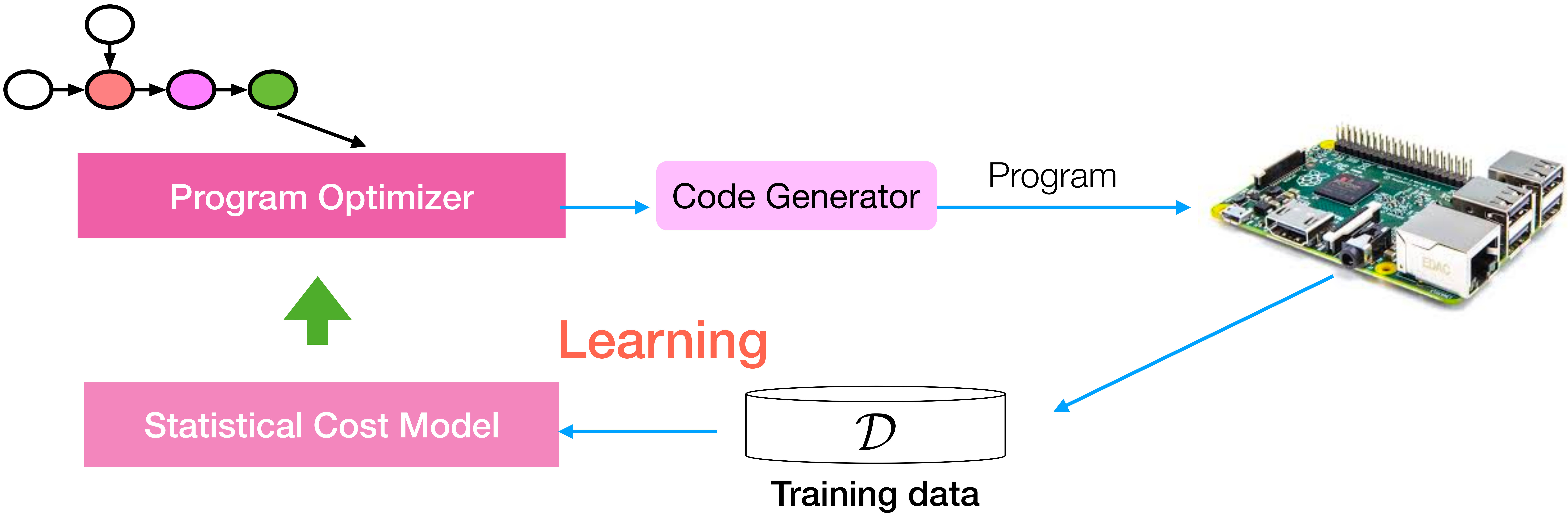
Learning-based Program Optimizer



Learning-based Program Optimizer



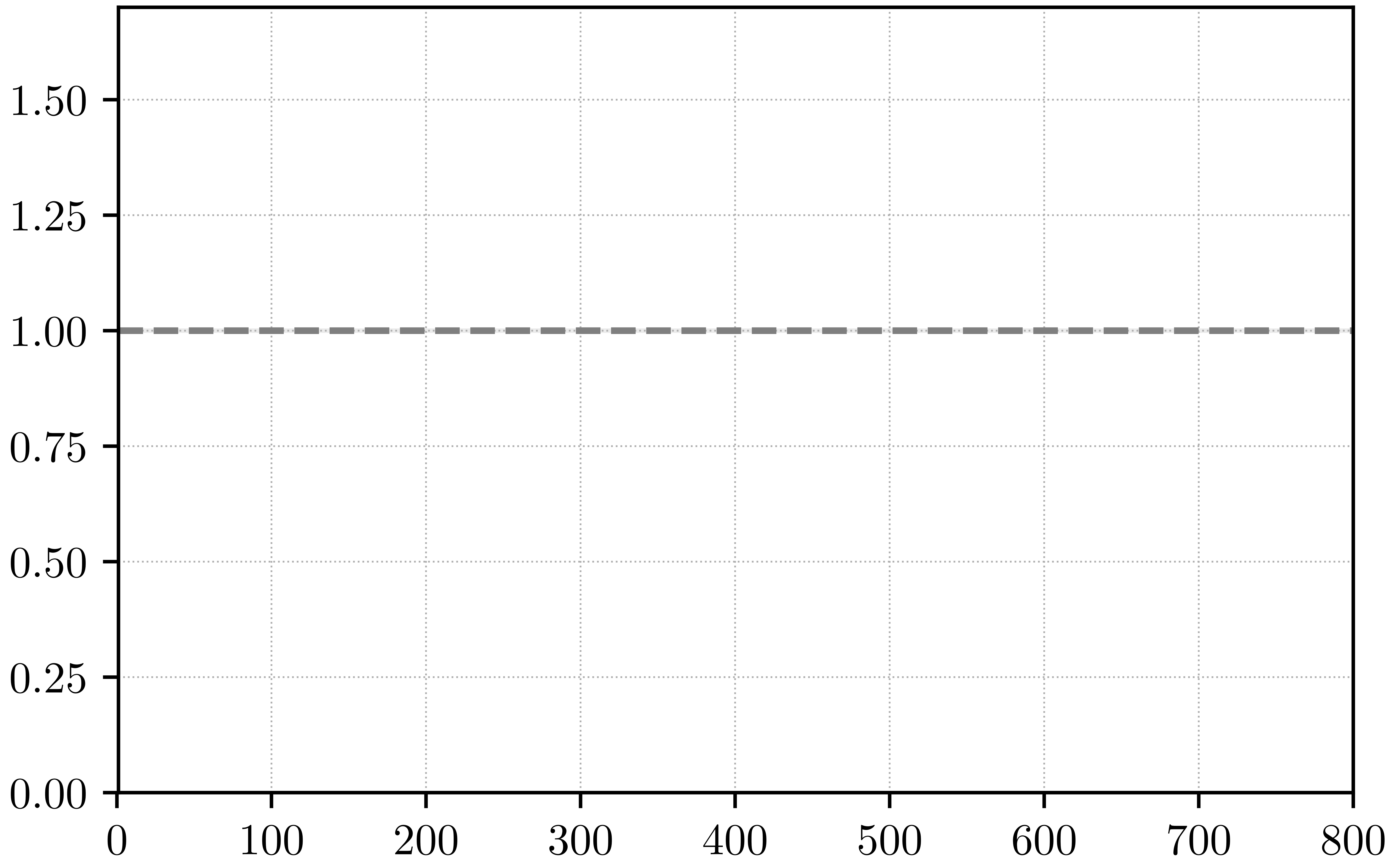
Learning-based Program Optimizer



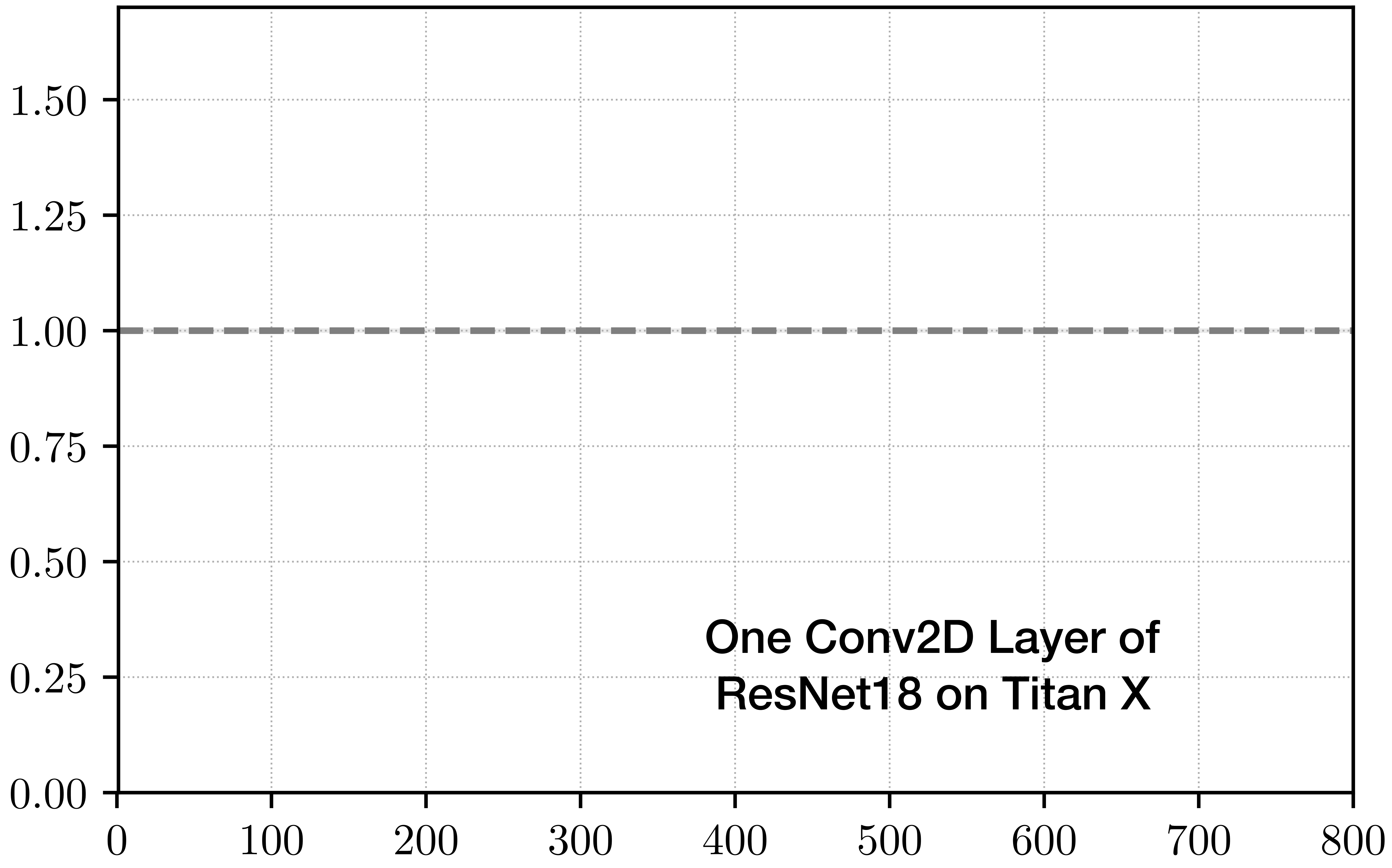
Adapt to hardware type by learning

Make prediction in 1ms level

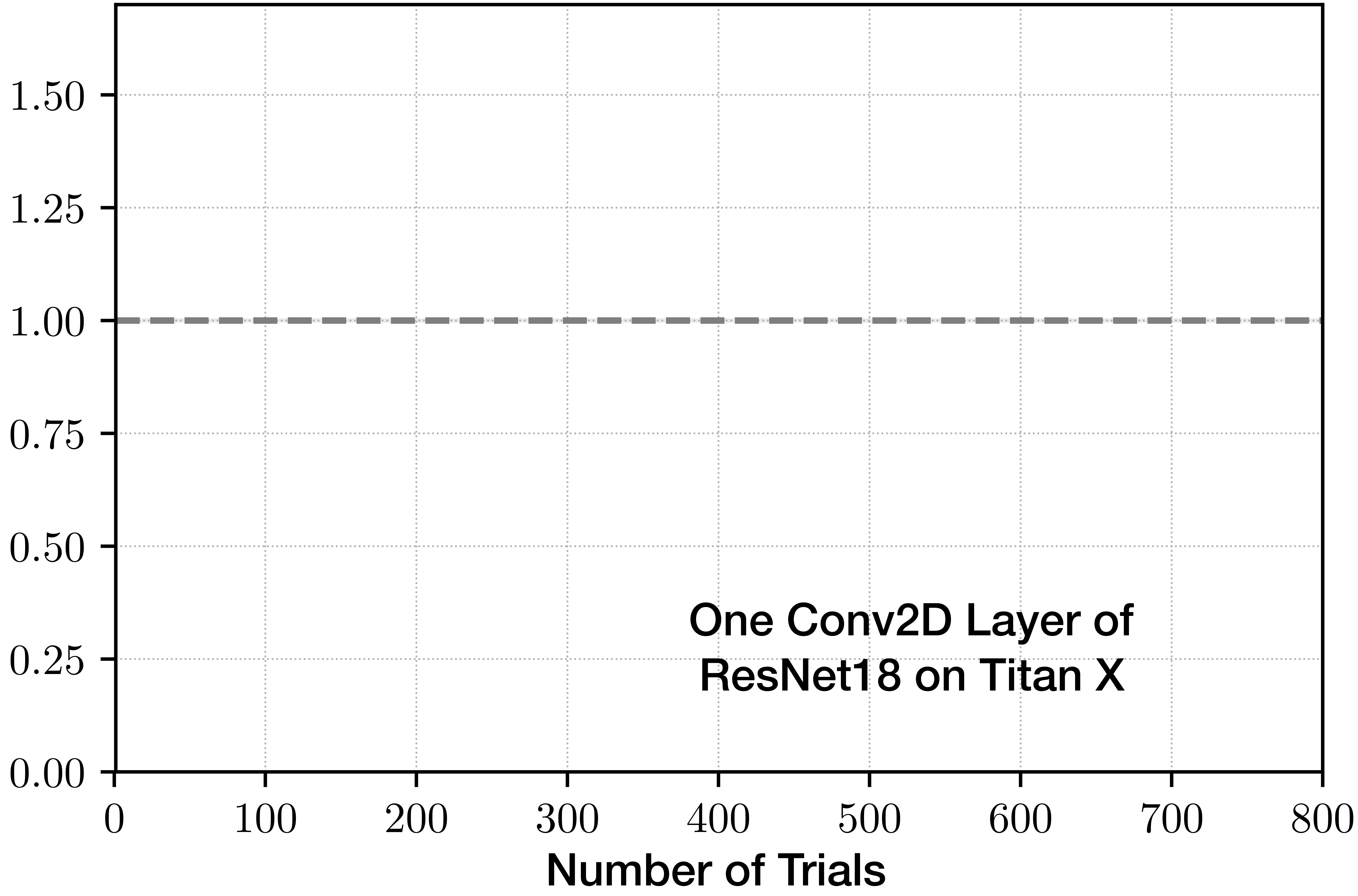
Effectiveness of ML based Model



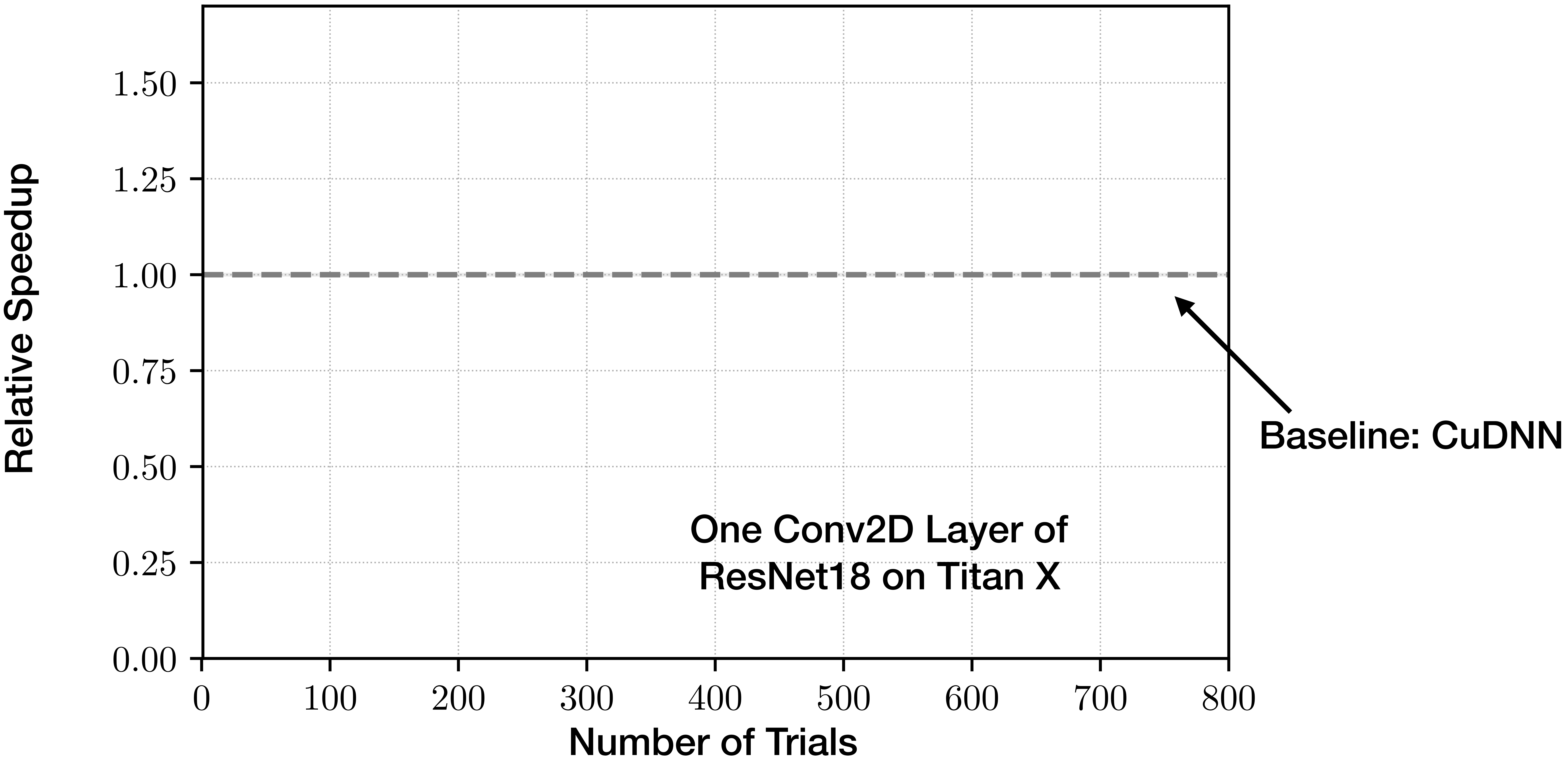
Effectiveness of ML based Model



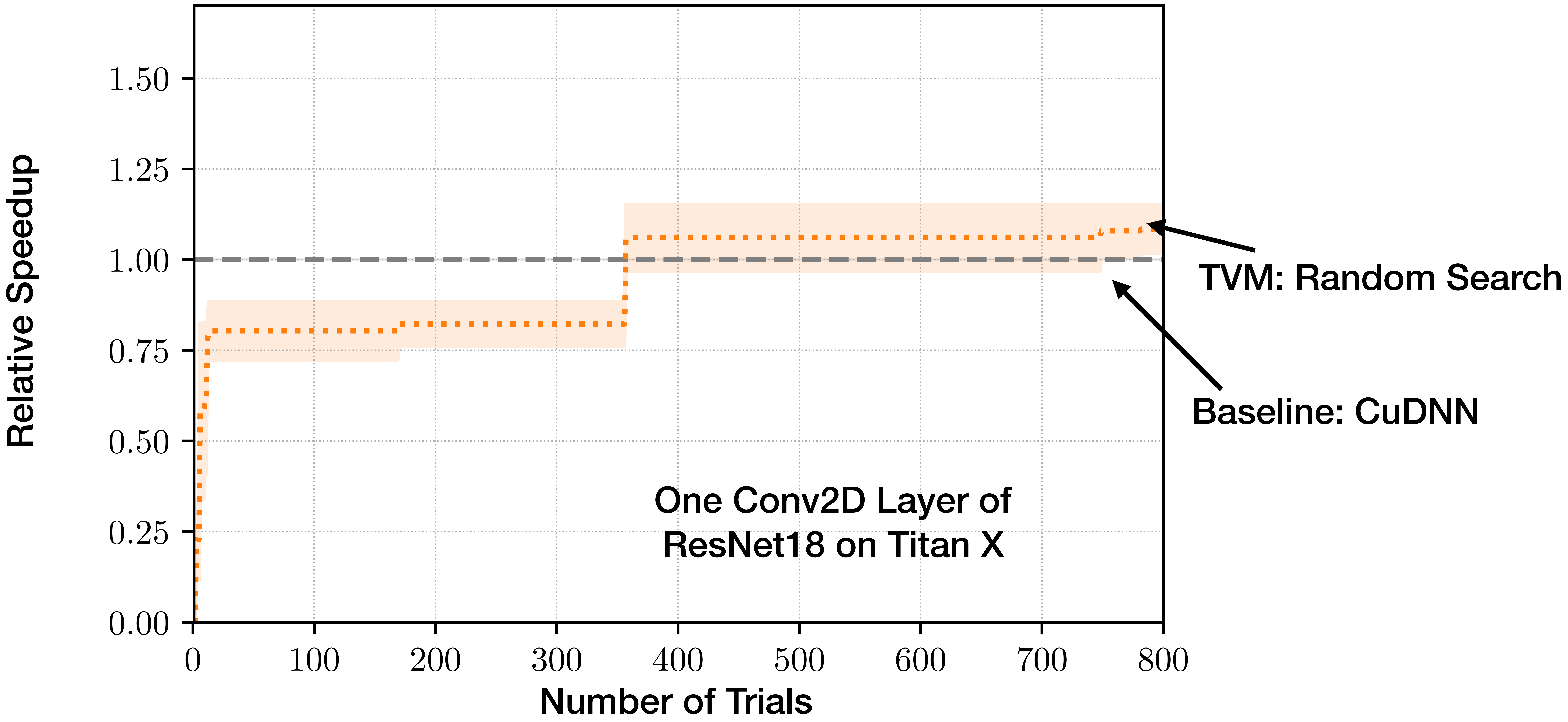
Effectiveness of ML based Model



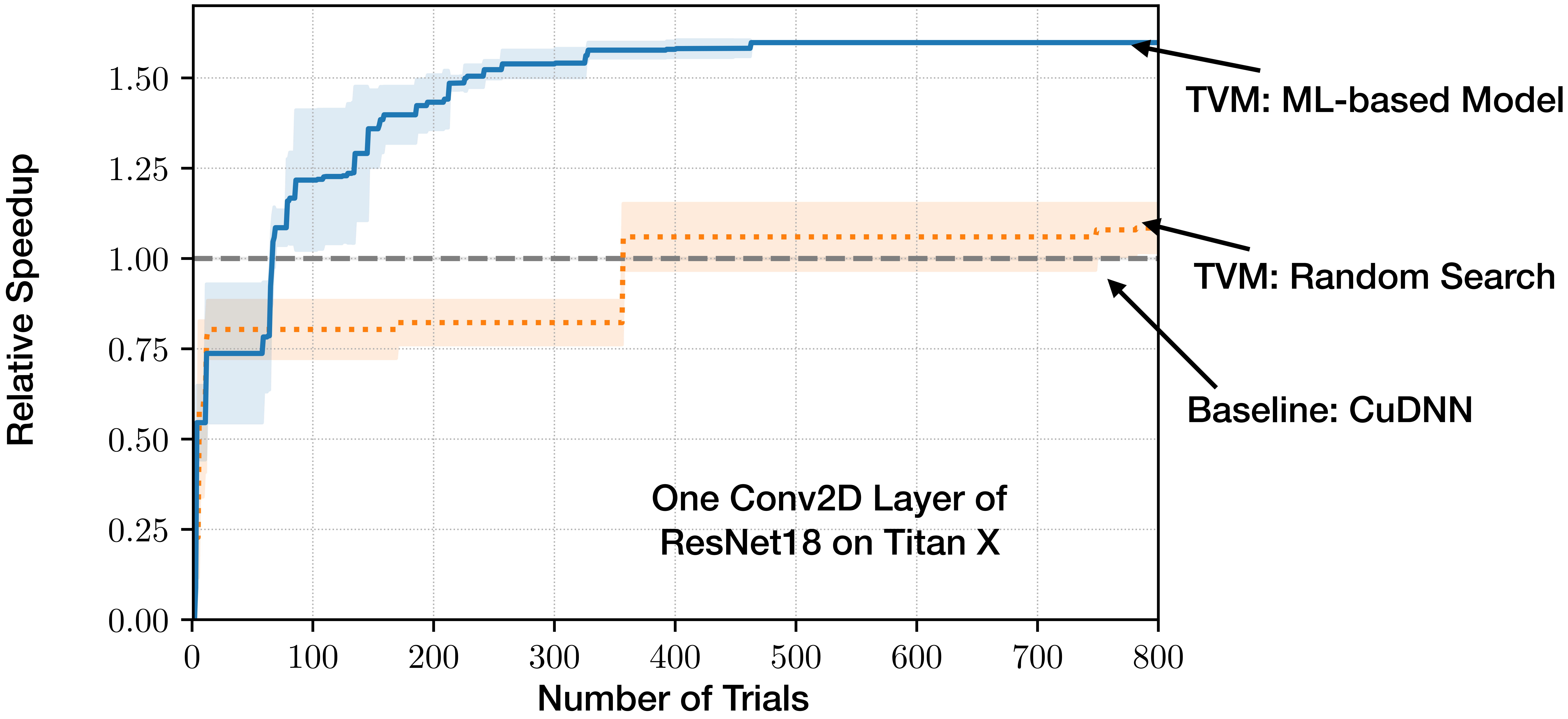
Effectiveness of ML based Model



Effectiveness of ML based Model



Effectiveness of ML based Model



Transfer and Lifelong Learning

Tensor Operator A

Exploration Module

Proposal

Code Generator

Transfer and Lifelong Learning

Tensor Operator A

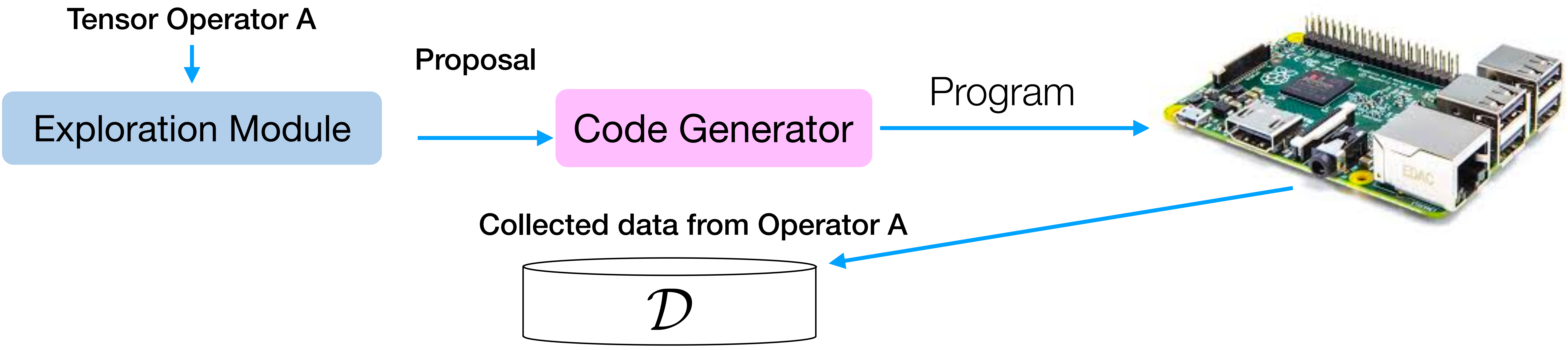
Exploration Module

Proposal

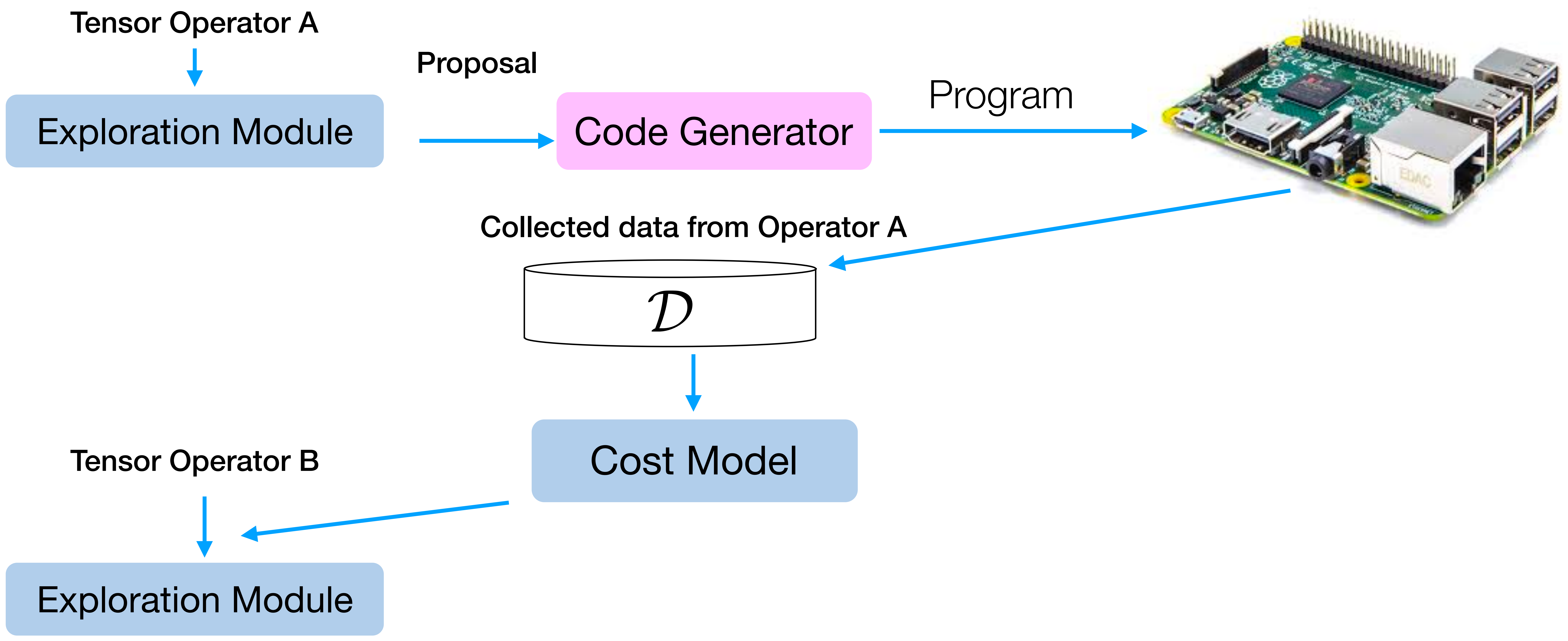
Code Generator

Program

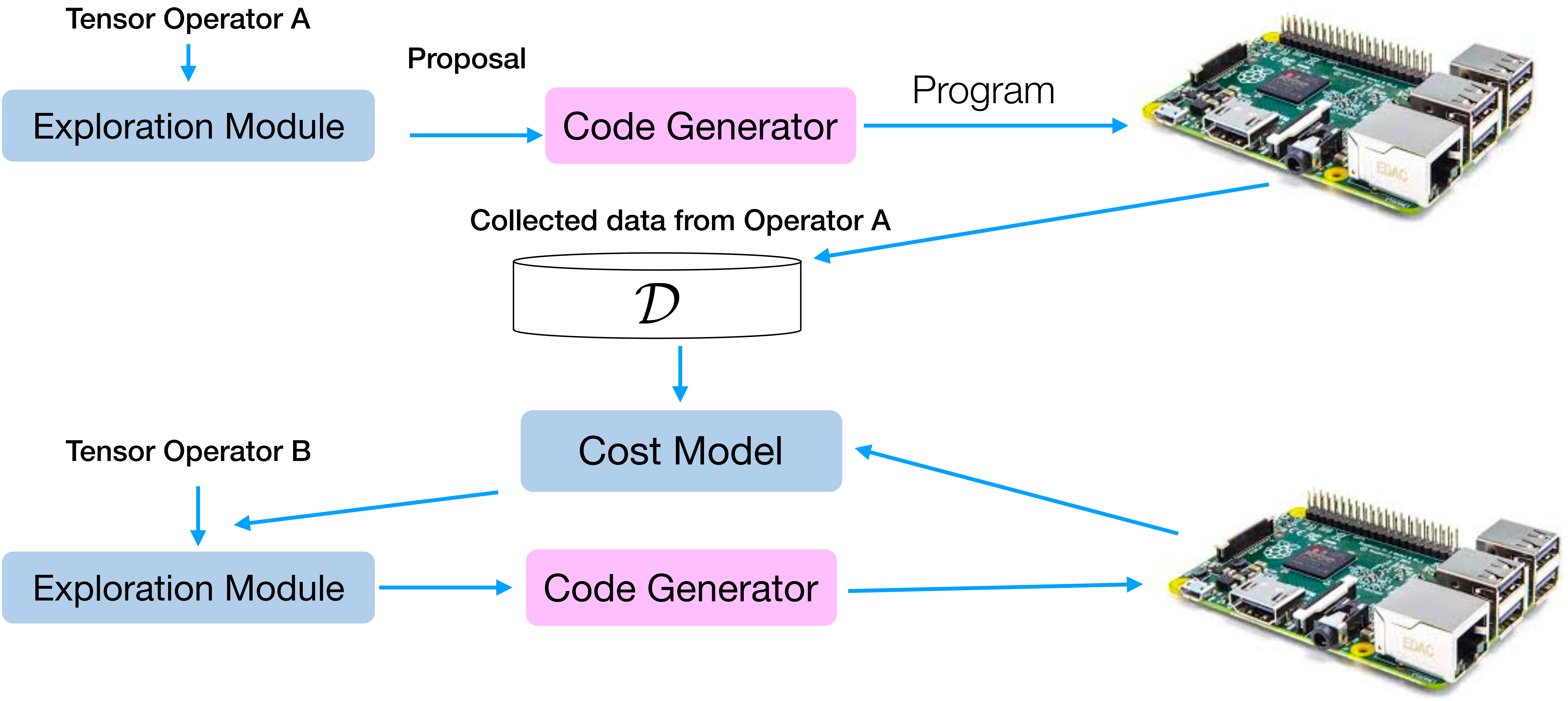
Transfer and Lifelong Learning



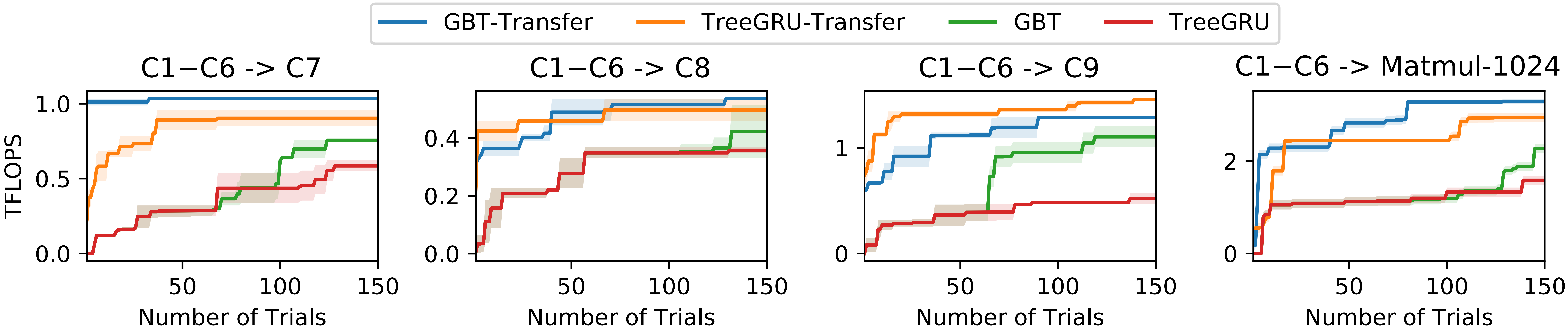
Transfer and Lifelong Learning



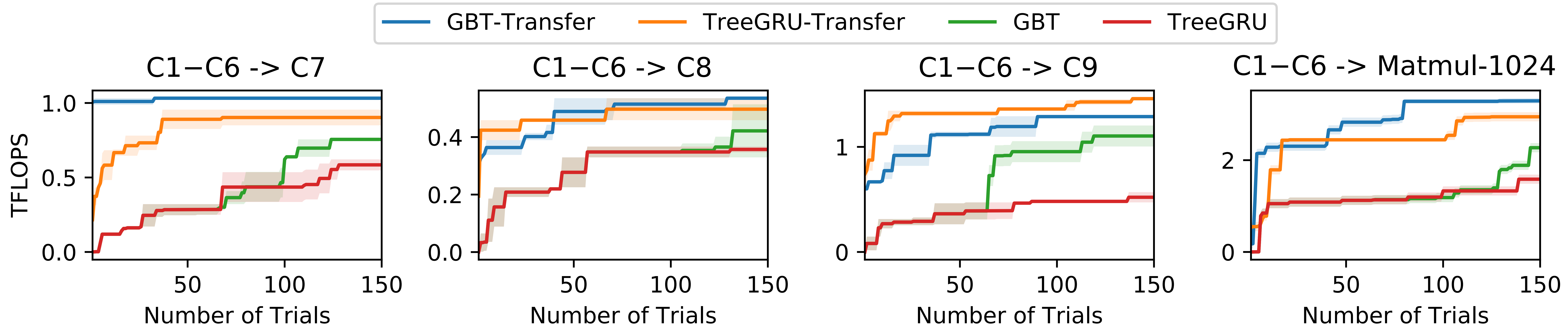
Transfer and Lifelong Learning



Impact of Transfer Learning



Impact of Transfer Learning



3x to 10x speedup over non-transfer case

AutoTVM API

Developer Search Space Specification

normal schedule: single configuration

```
def matmul(N, L, M, dtype):
    A = tvm.placeholder((N, L), name='A', dtype=dtype)
    B = tvm.placeholder((L, M), name='B', dtype=dtype)
    k = tvm.reduce_axis((0, L), name='k')
    C = tvm.compute((N, M), lambda i, j:
         tvm.sum(A[i, k] * B[k, j], axis=k), name='C')
    s = tvm.create_schedule(C.op)

    # schedule
    y, x = s[C].op.axis
    k = s[C].op.reduce_axis[0]
    yo, yi = s[C].split(y, 8)
    xo, xi = s[C].split(x, 8)
    s[C].reorder(yo, xo, k, yi, xi)
    return s, [A, B, C]
```

template schedule: space of configs

```
@autotvm.template
def matmul(N, L, M, dtype):
    A = tvm.placeholder((N, L), name='A', dtype=dtype)
    B = tvm.placeholder((L, M), name='B', dtype=dtype)
    k = tvm.reduce_axis((0, L), name='k')
    C = tvm.compute((N, M), lambda i, j:
           tvm.sum(A[i, k] * B[k, j], axis=k), name='C')
    s = tvm.create_schedule(C.op)

    y, x = s[C].op.axis
    k = s[C].op.reduce_axis[0]
    cfg = autotvm.get_config()
    cfg.define_split("tile_y", y, num_outputs=2)
    cfg.define_split("tile_x", x, num_outputs=2)

    # schedule according to config
    yo, yi = cfg["tile_y"].apply(s, C, y)
    xo, xi = cfg["tile_x"].apply(s, C, x)
    s[C].reorder(yo, xo, k, yi, xi)
    return s, [A, B, C]
```

Developer Search Space Specification

normal schedule: single configuration

```
def matmul(N, L, M, dtype):
    A = tvm.placeholder((N, L), name='A', dtype=dtype)
    B = tvm.placeholder((L, M), name='B', dtype=dtype)
    k = tvm.reduce_axis((0, L), name='k')
    C = tvm.compute((N, M), lambda i, j:
         tvm.sum(A[i, k] * B[k, j], axis=k), name='C')
    s = tvm.create_schedule(C.op)

    # schedule
    y, x = s[C].op.axis
    k = s[C].op.reduce_axis[0]
    yo, yi = s[C].split(y, 8)
    xo, xi = s[C].split(x, 8)
    s[C].reorder(yo, xo, k, yi, xi)
    return s, [A, B, C]
```

template schedule: space of configs

```
@autotvm.template
def matmul(N, L, M, dtype):
    A = tvm.placeholder((N, L), name='A', dtype=dtype)
    B = tvm.placeholder((L, M), name='B', dtype=dtype)
    k = tvm.reduce_axis((0, L), name='k')
    C = tvm.compute((N, M), lambda i, j:
           tvm.sum(A[i, k] * B[k, j], axis=k), name='C')
    s = tvm.create_schedule(C.op)

    y, x = s[C].op.axis
    k = s[C].op.reduce_axis[0]
    cfg = autotvm.get_config()
    cfg.define_split("tile_y", y, num_outputs=2)
    cfg.define_split("tile_x", x, num_outputs=2)

    # schedule according to config
    yo, yi = cfg["tile_y"].apply(s, C, y)
    xo, xi = cfg["tile_x"].apply(s, C, x)
    s[C].reorder(yo, xo, k, yi, xi)
    return s, [A, B, C]
```

Space of configurations

The Master Template

```
@autotvm.template
def matmul(N, L, M, dtype):
    A = tvm.placeholder((N, L), name='A', dtype=dtype)
    B = tvm.placeholder((L, M), name='B', dtype=dtype)
    k = tvm.reduce_axis((0, L), name='k')
    C = tvm.compute((N, M), lambda i, j:
           tvm.sum(A[i, k] * B[k, j], axis=k), name='C')

    s = autovm.master_schedule(C.op)

    return s, [A, B, C]
```

Remove the need of writing templates

Analyze the axis relation

Enumerate all possible patterns

Leave the job to the optimizer

Promising results on CUDA

Active on-going research

The Master Template

```python
@autotvm.template
def matmul(N, L, M, dtype):
    A = tvm.placeholder((N, L), name='A', dtype=dtype)
    B = tvm.placeholder((L, M), name='B', dtype=dtype)
    k = tvm.reduce_axis((0, L), name='k')
    C = tvm.compute((N, M), lambda i, j:
           tvm.sum(A[i, k] * B[k, j], axis=k), name='C')

    s = autovm.master_schedule(C.op)

    return s, [A, B, C]
```

Remove the need of writing templates

Analyze the axis relation

Enumerate all possible patterns

Leave the job to the optimizer

Promising results on CUDA

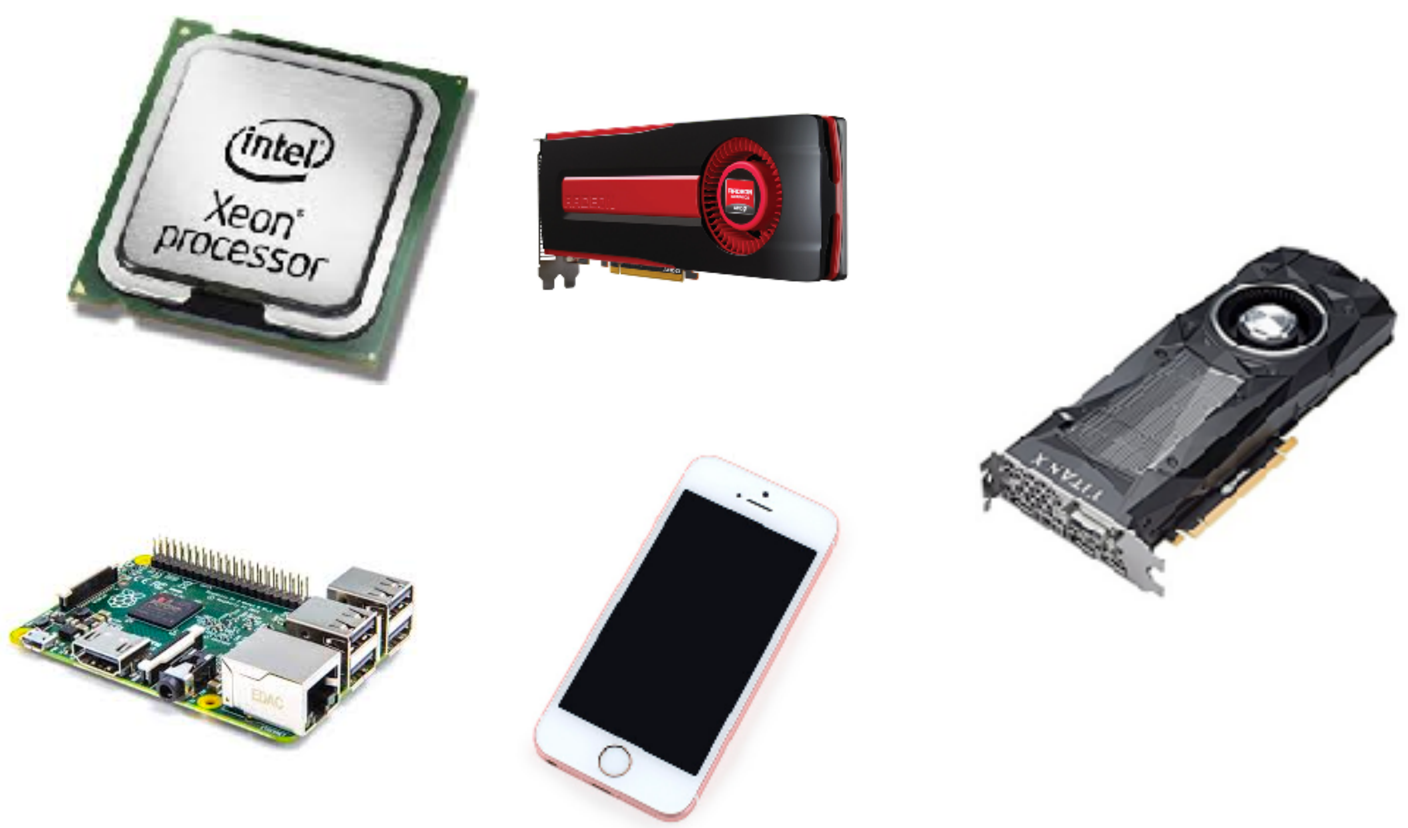
Active on-going research

Summarizes common Optimization lessons

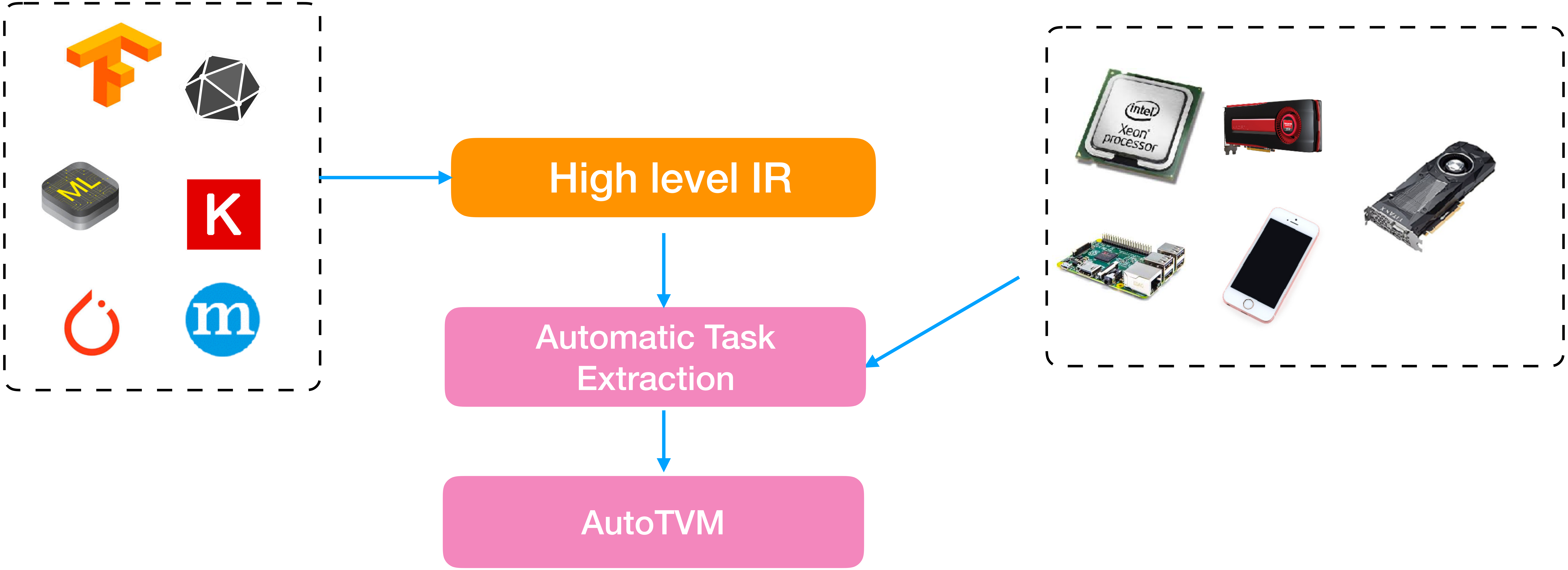
End to End Integration

End to End Integration

High level IR

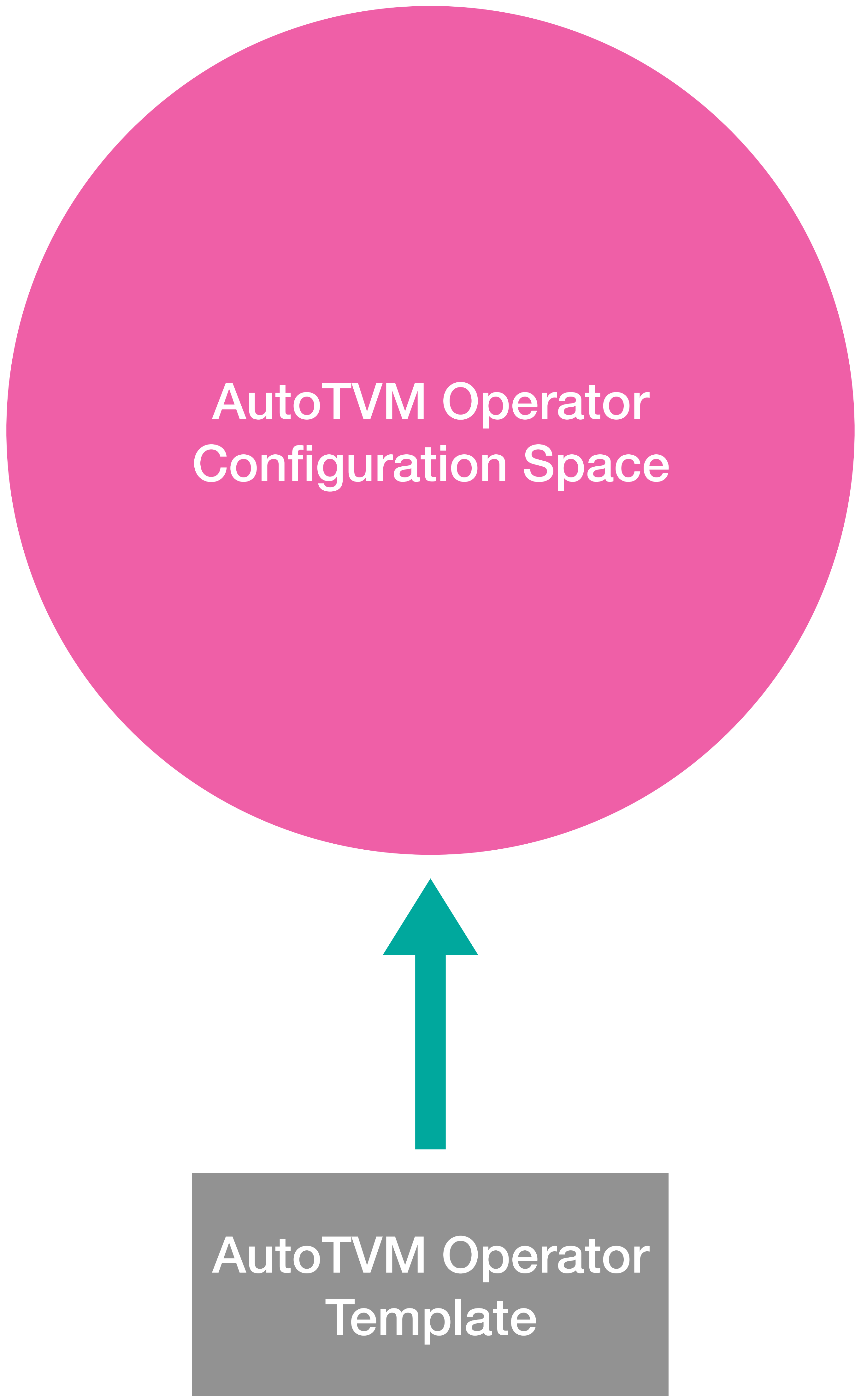


End to End Integration

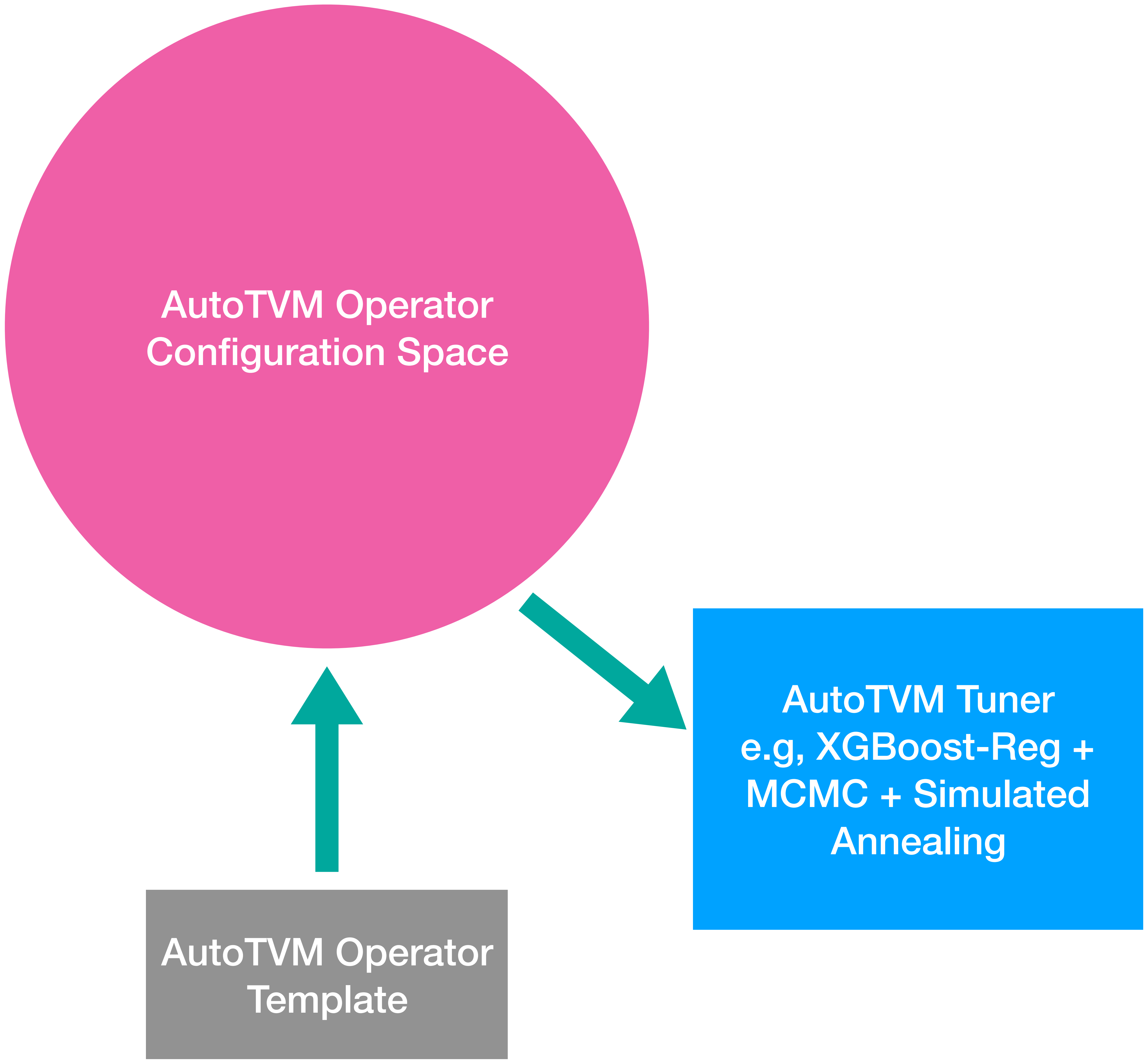


Device Fleet: Scaling up AutoTVM

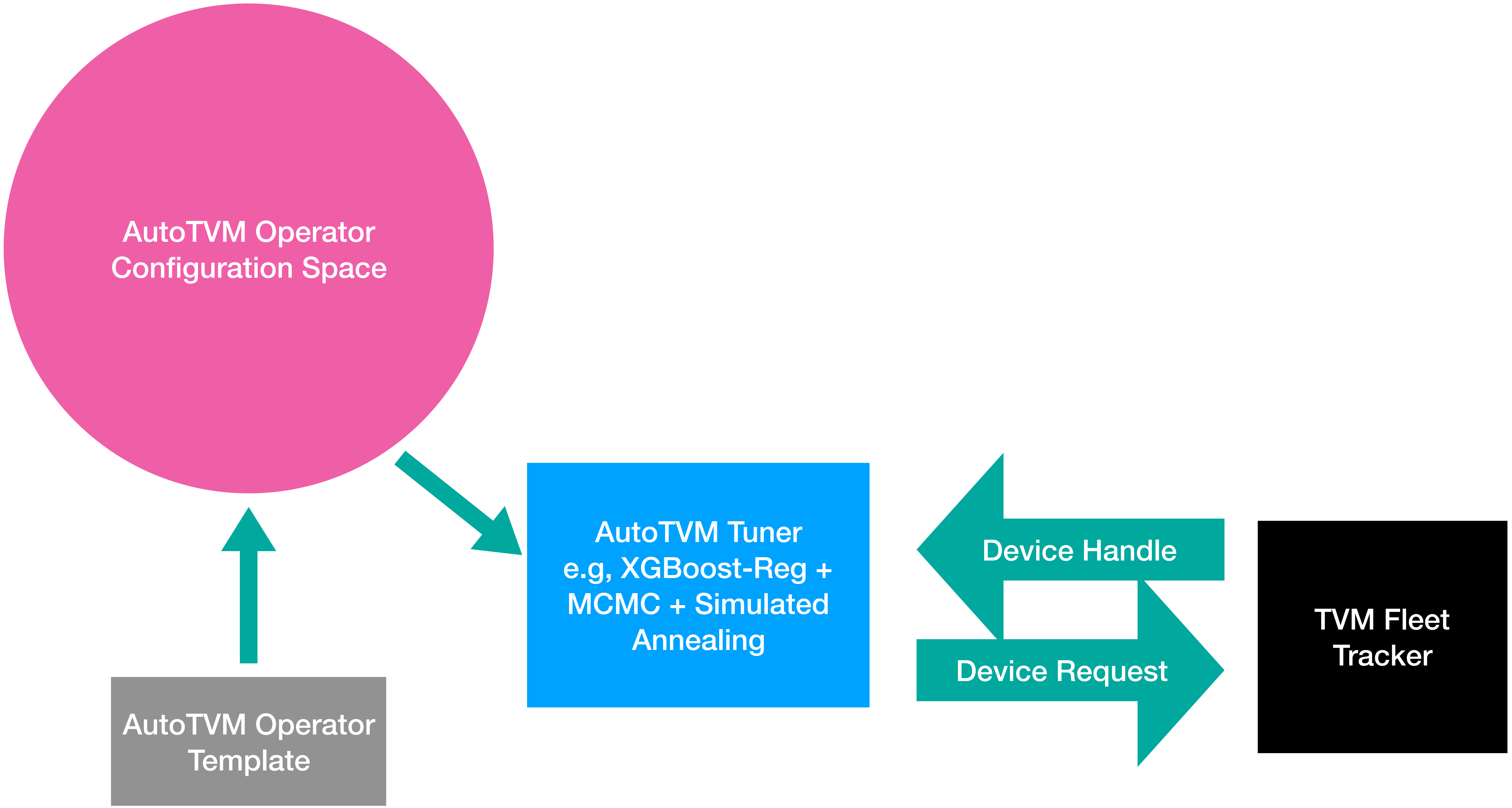
Device Fleet: Scaling up AutoTVM



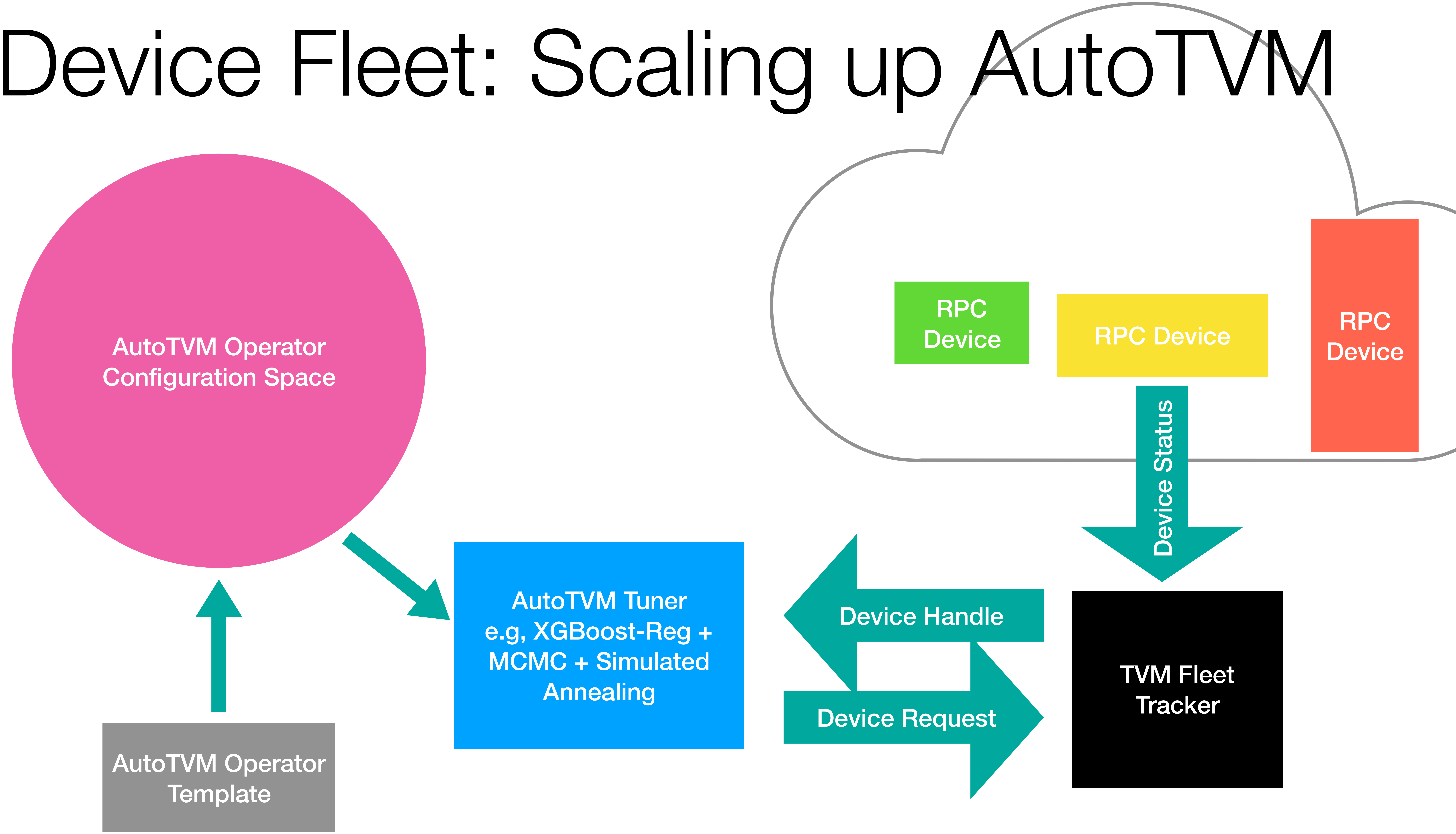
Device Fleet: Scaling up AutoTVM



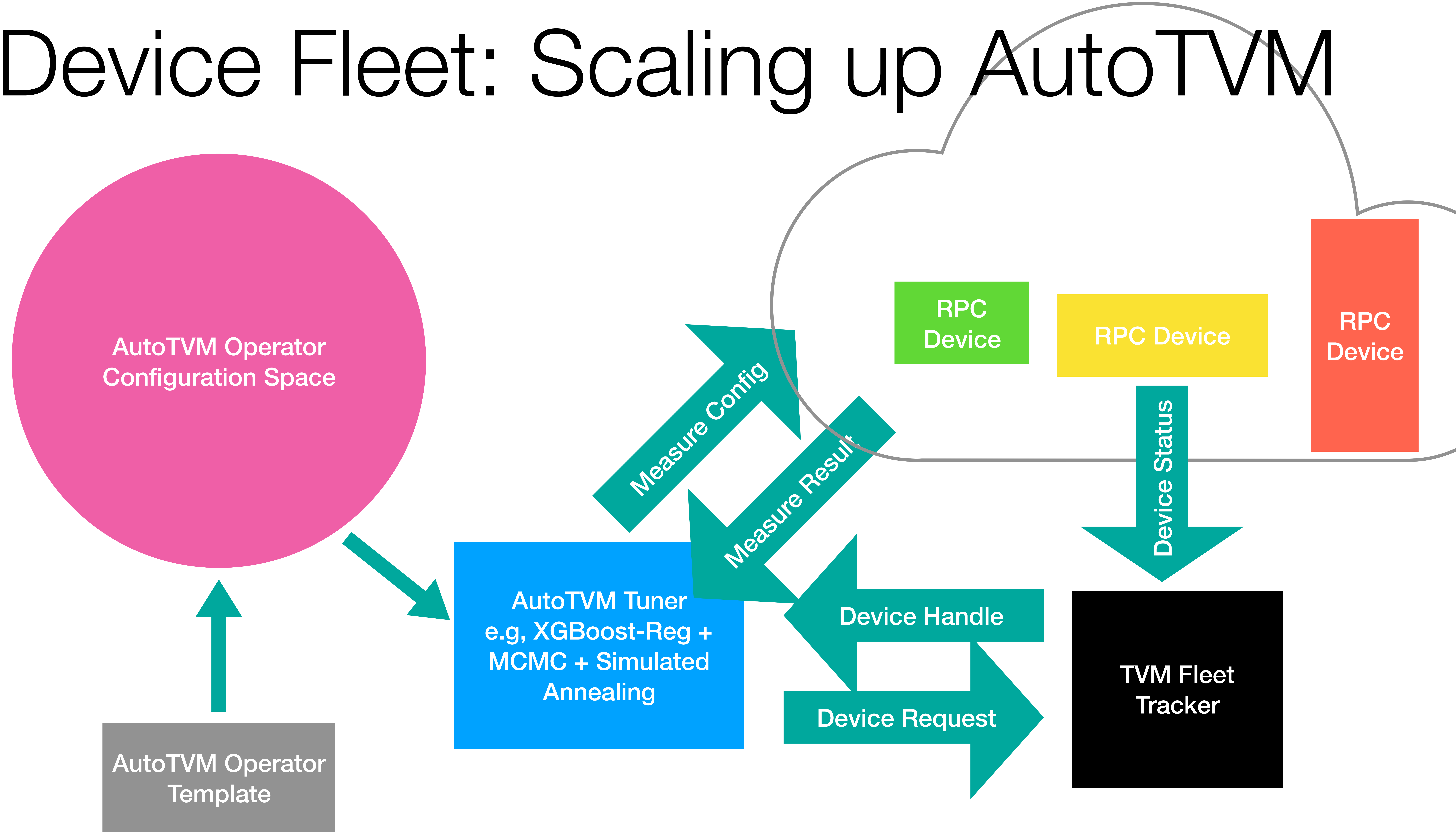
Device Fleet: Scaling up AutoTVM



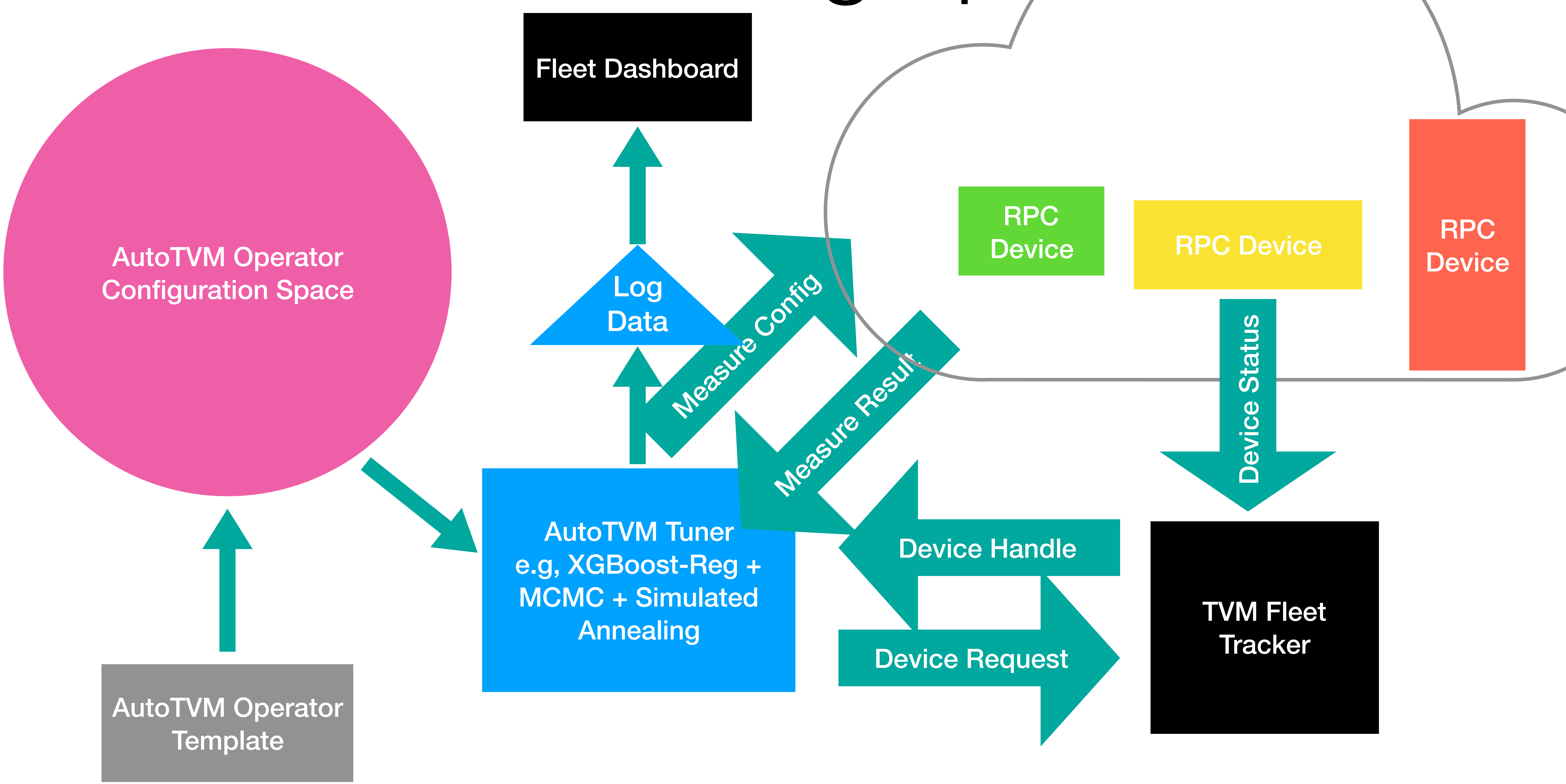
Device Fleet: Scaling up AutoTVM



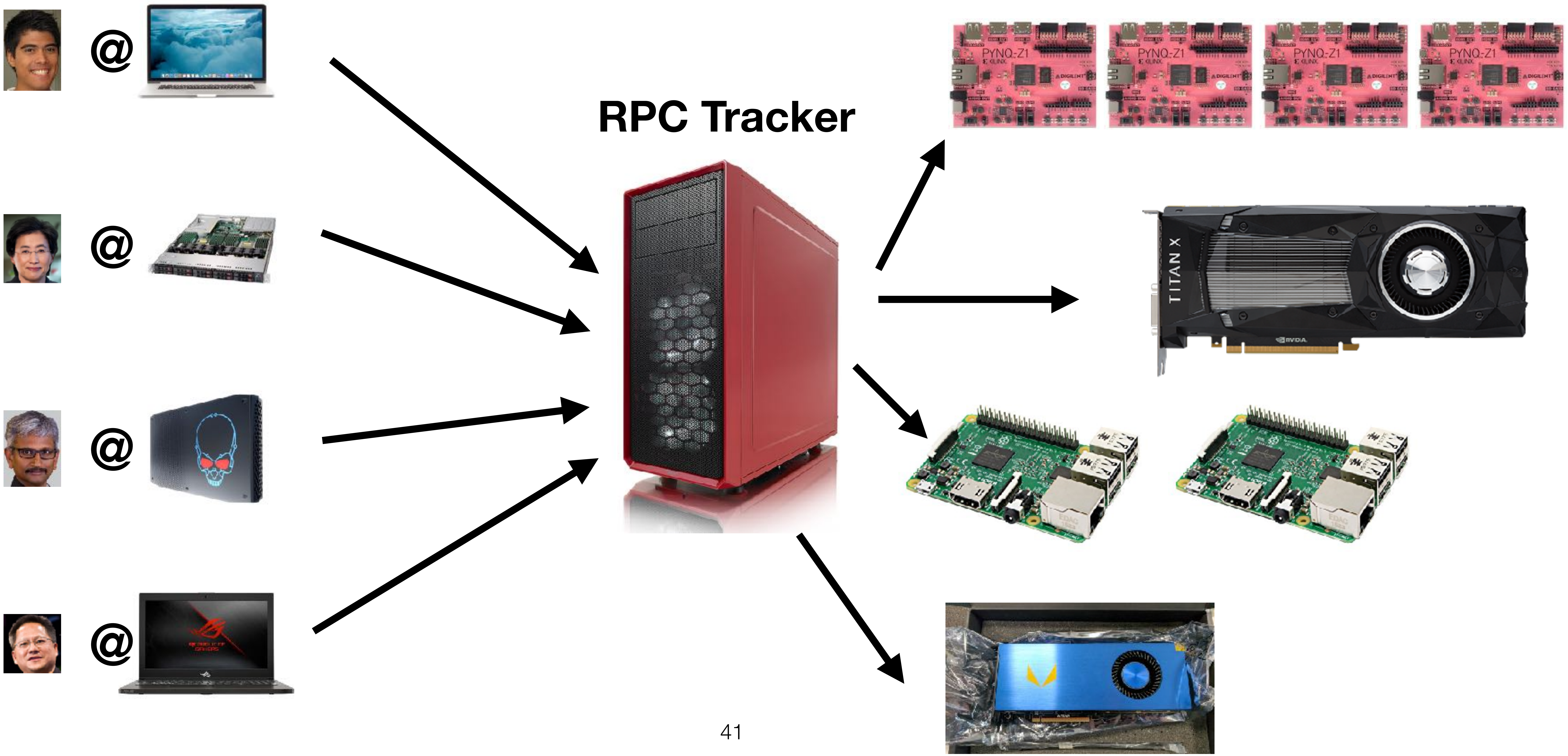
Device Fleet: Scaling up AutoTVM



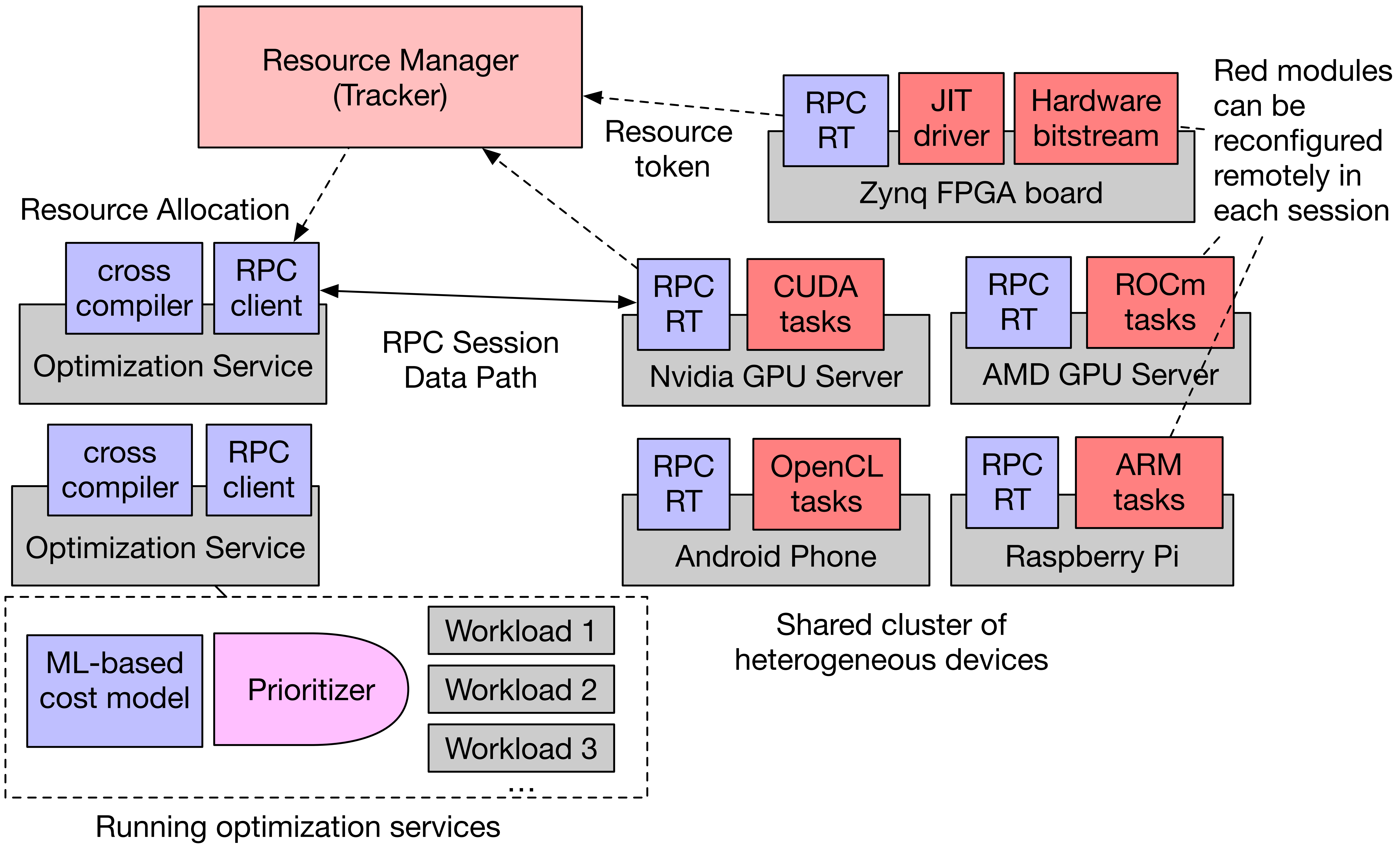
Device Fleet: Scaling up AutoTVM



RPC + Tracker: **develop** *locally*, **execute** *remotely*



Low Level: Portable RPC Tracker + Server



Fleet Tracker Example

Fleet Tracker Example

```
tracker = rpc.connect_tracker(tracker_host, tracker_port)
```

Fleet Tracker Example

```
tracker = rpc.connect_tracker(tracker_host, tracker_port)
remote = tracker.request(key, priority=0, session_timeout=60)
```

Fleet Tracker Example

```
tracker = rpc.connect_tracker(tracker_host, tracker_port)
remote = tracker.request(key, priority=0, session_timeout=60)
context = remote.cl(0)
```

Fleet Tracker Example

```
tracker = rpc.connect_tracker(tracker_host, tracker_port)
remote = tracker.request(key, priority=0, session_timeout=60)
context = remote.cl(0)
remote.upload('my_library.so')
```

Fleet Tracker Example

```
tracker = rpc.connect_tracker(tracker_host, tracker_port)
remote = tracker.request(key, priority=0, session_timeout=60)
context = remote.cl(0)
remote.upload('my_library.so')
a = tvm.nd.array(a_numpy_array, context)
```

Fleet Tracker Example

```
tracker = rpc.connect_tracker(tracker_host, tracker_port)
remote = tracker.request(key, priority=0, session_timeout=60)
context = remote.cl(0)
remote.upload('my_library.so')
a = tvm.nd.array(a_numpy_array, context)
b = tvm.nd.array(b_numpy_array, context)
```

Fleet Tracker Example

```
tracker = rpc.connect_tracker(tracker_host, tracker_port)
remote = tracker.request(key, priority=0, session_timeout=60)
context = remote.cl(0)
remote.upload('my_library.so')
a = tvm.nd.array(a_numpy_array, context)
b = tvm.nd.array(b_numpy_array, context)
func = remote.load_module('my_library.so')
```

Fleet Tracker Example

```
tracker = rpc.connect_tracker(tracker_host, tracker_port)
remote = tracker.request(key, priority=0, session_timeout=60)
context = remote.cl(0)
remote.upload('my_library.so')
a = tvm.nd.array(a_numpy_array, context)
b = tvm.nd.array(b_numpy_array, context)
func = remote.load_module('my_library.so')
func_timer = func.time_evaluator(func.entry_name, context,
number=10)
```

Fleet Tracker Example

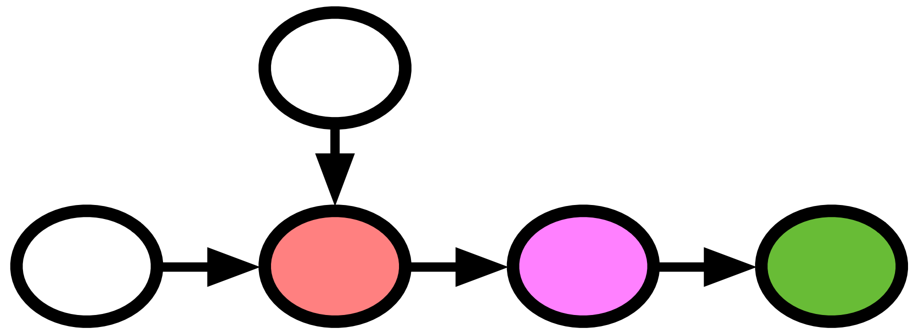
```
tracker = rpc.connect_tracker(tracker_host, tracker_port)
remote = tracker.request(key, priority=0, session_timeout=60)
context = remote.cl(0)
remote.upload('my_library.so')
a = tvm.nd.array(a_numpy_array, context)
b = tvm.nd.array(b_numpy_array, context)
func = remote.load_module('my_library.so')
func_timer = func.time_evaluator(func.entry_name, context,
number=10)
cost = func_timer(a, b).mean
```

Fleet Tracker Example

```
tracker = rpc.connect_tracker(tracker_host, tracker_port)
remote = tracker.request(key, priority=0, session_timeout=60)
context = remote.cl(0)
remote.upload('my_library.so')
a = tvm.nd.array(a_numpy_array, context)
b = tvm.nd.array(b_numpy_array, context)
func = remote.load_module('my_library.so')
func_timer = func.time_evaluator(func.entry_name, context,
number=10)
cost = func_timer(a, b).mean
```

Learning-based Learning System

Frameworks



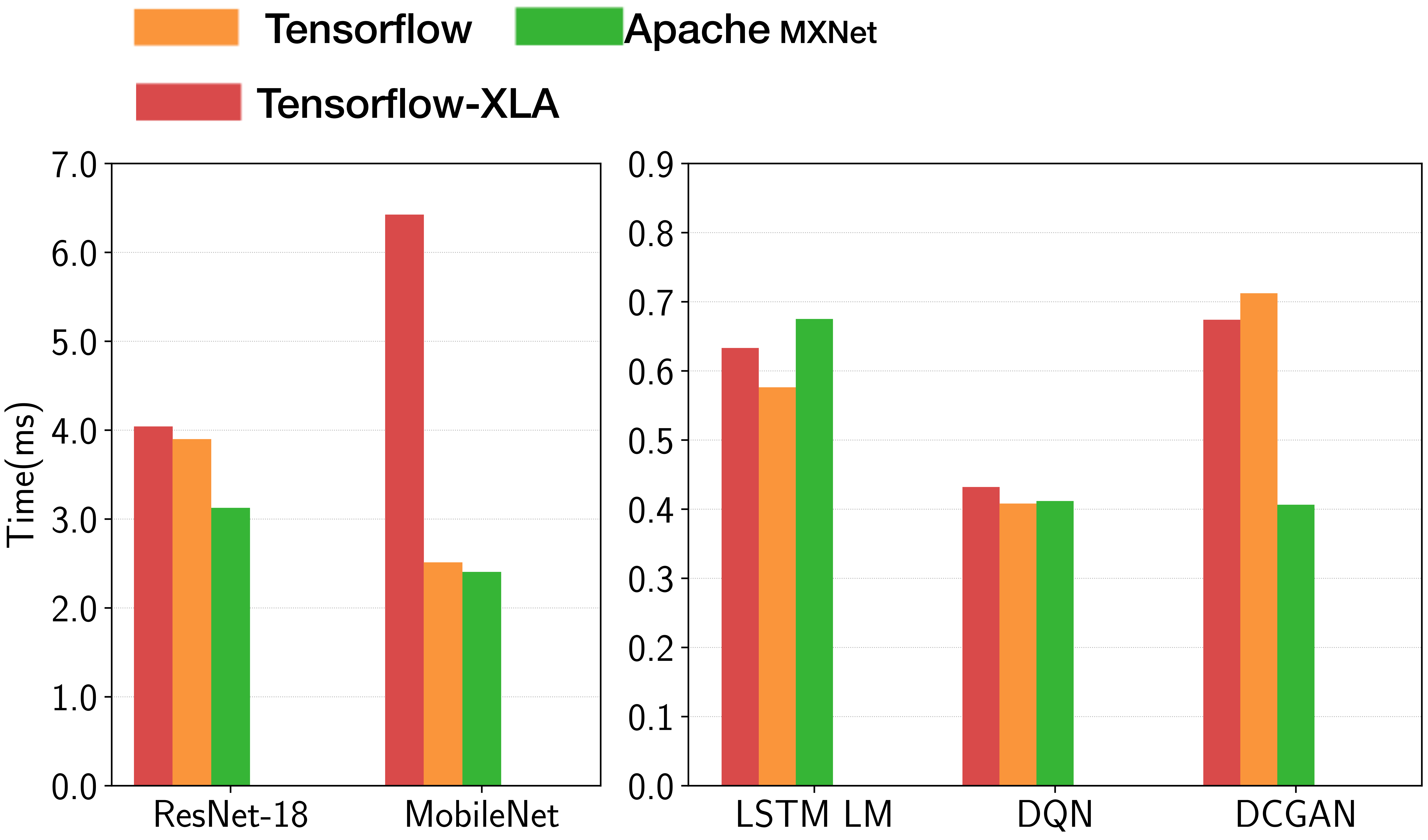
High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

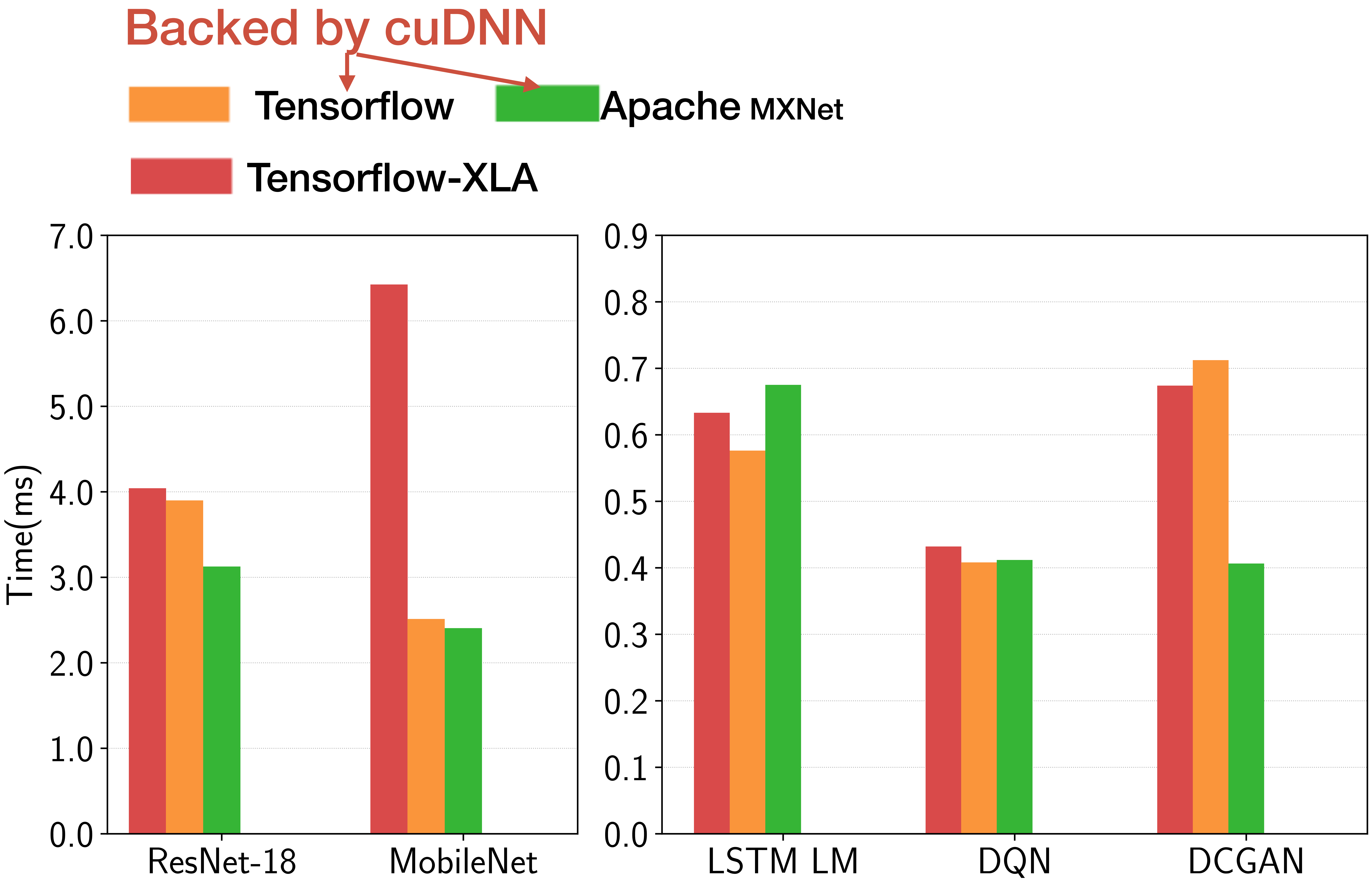
Machine Learning based Program Optimizer

Hardware

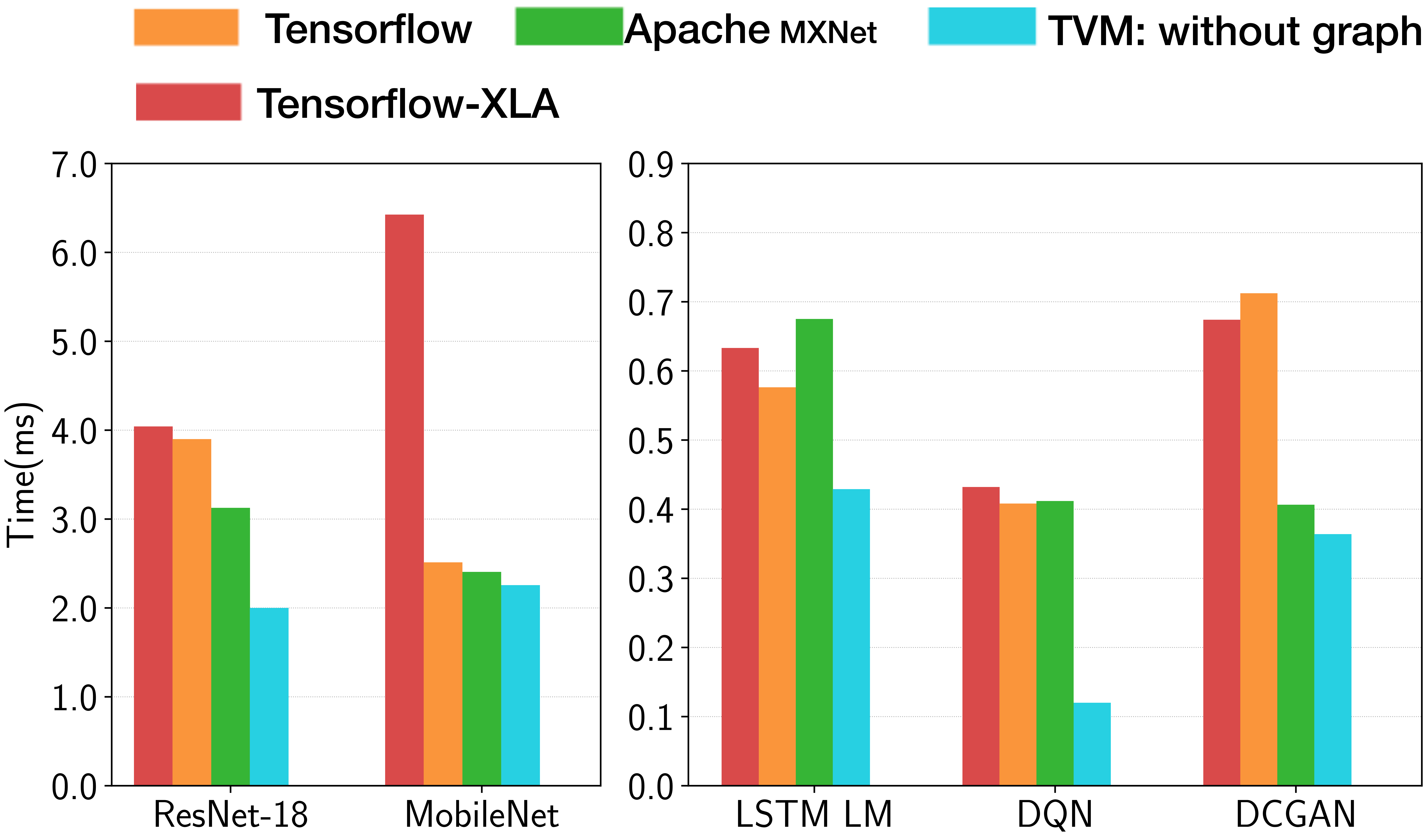
End to End Inference Performance (Nvidia Titan X)



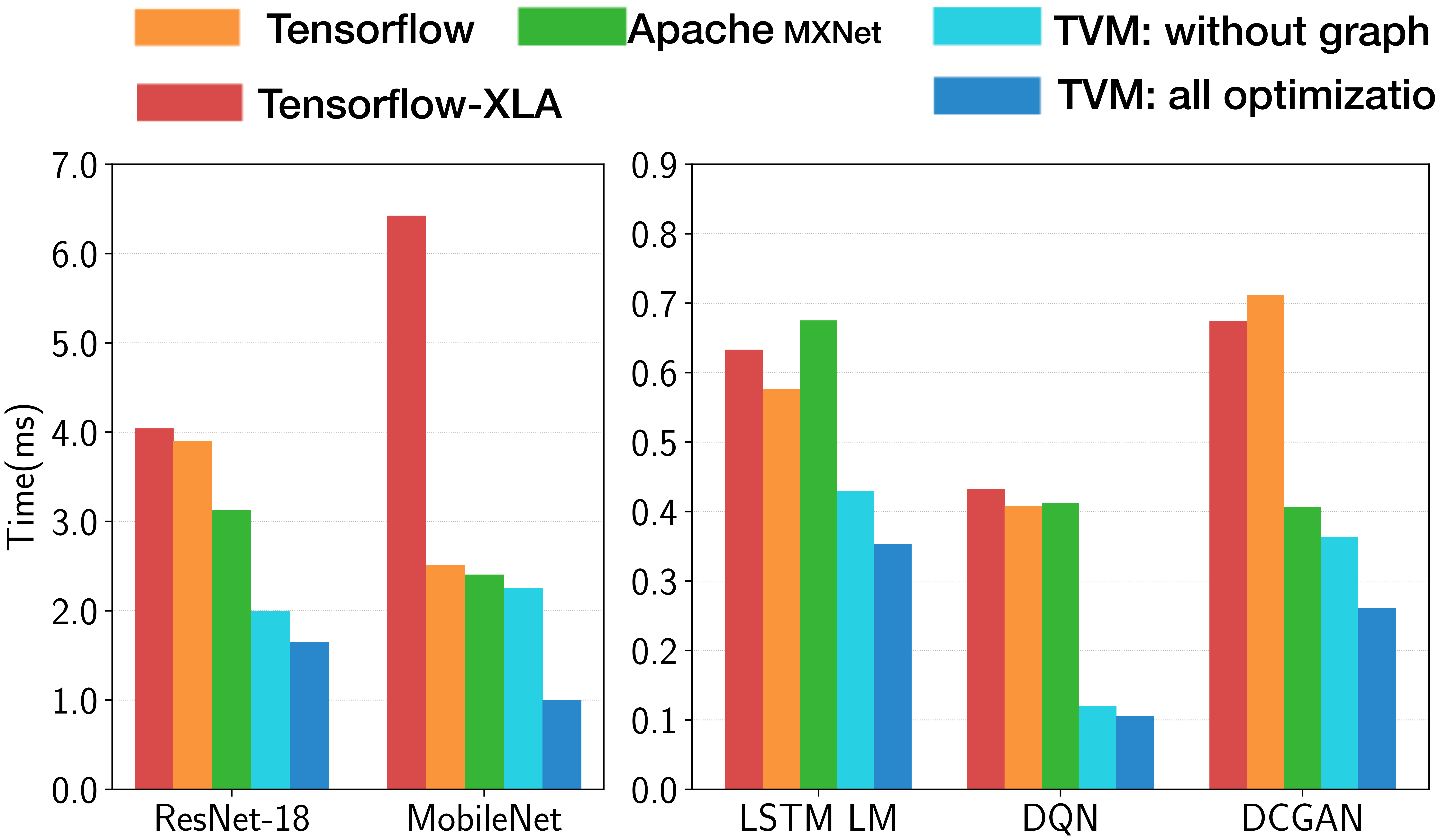
End to End Inference Performance (Nvidia Titan X)



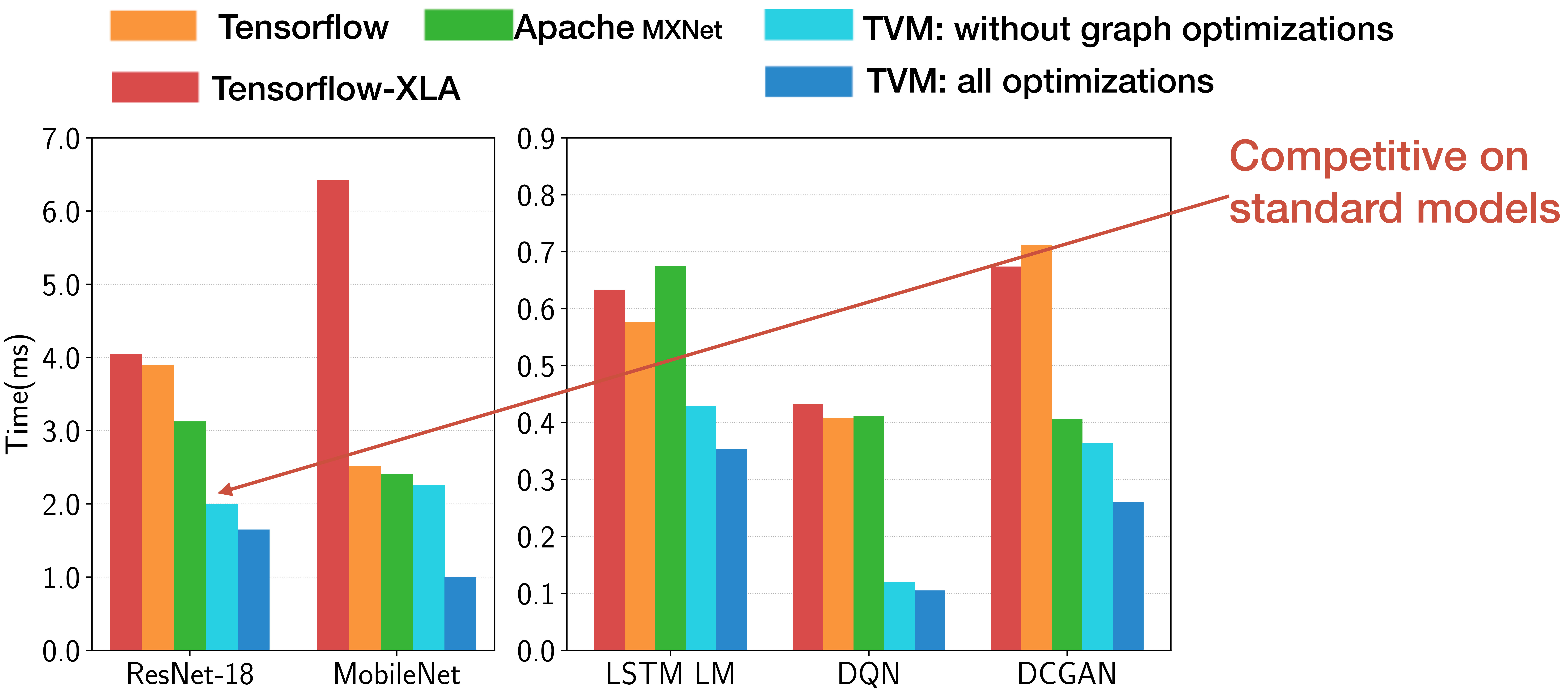
End to End Inference Performance (Nvidia Titan X)



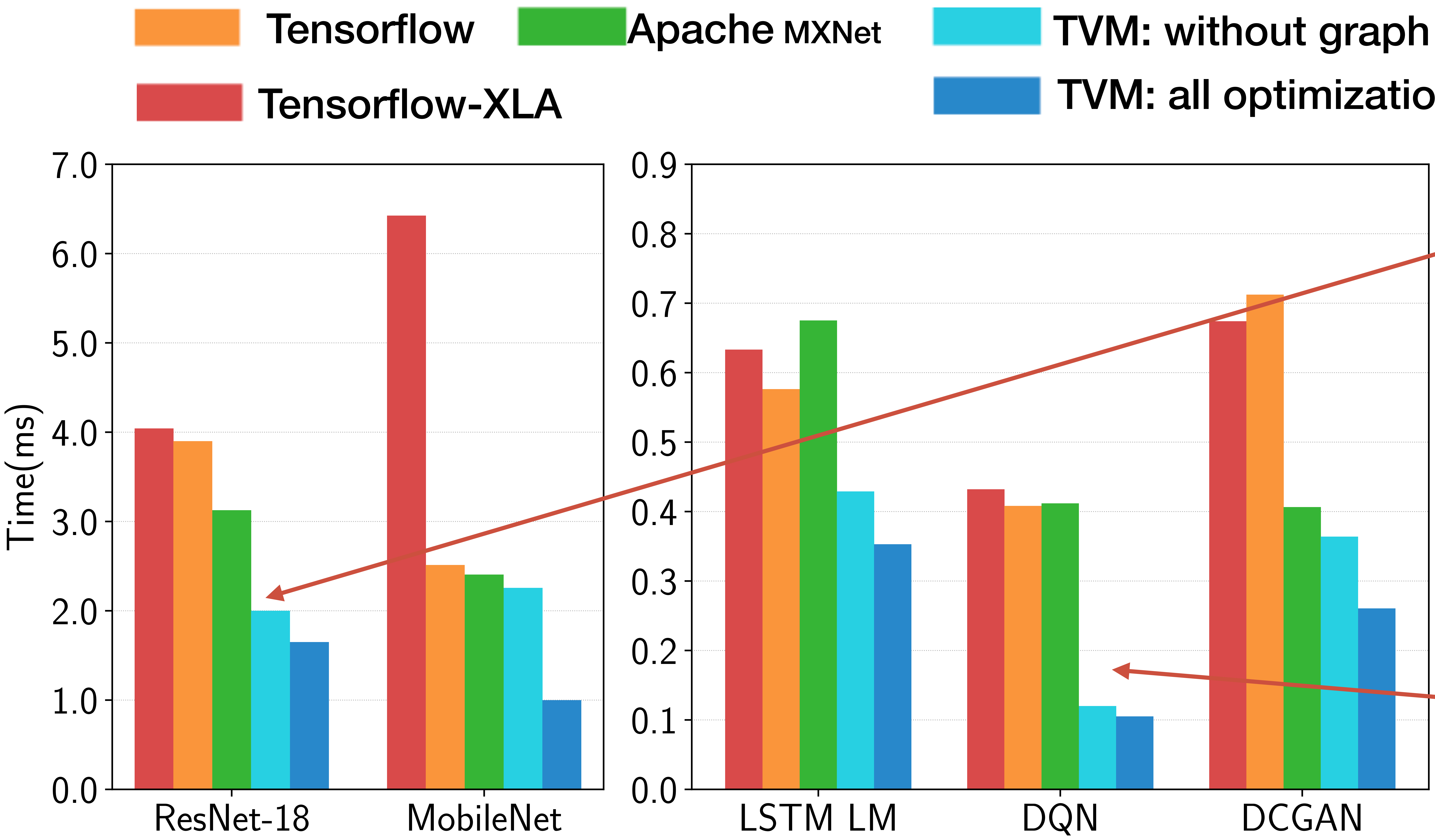
End to End Inference Performance (Nvidia Titan X)



End to End Inference Performance (Nvidia Titan X)



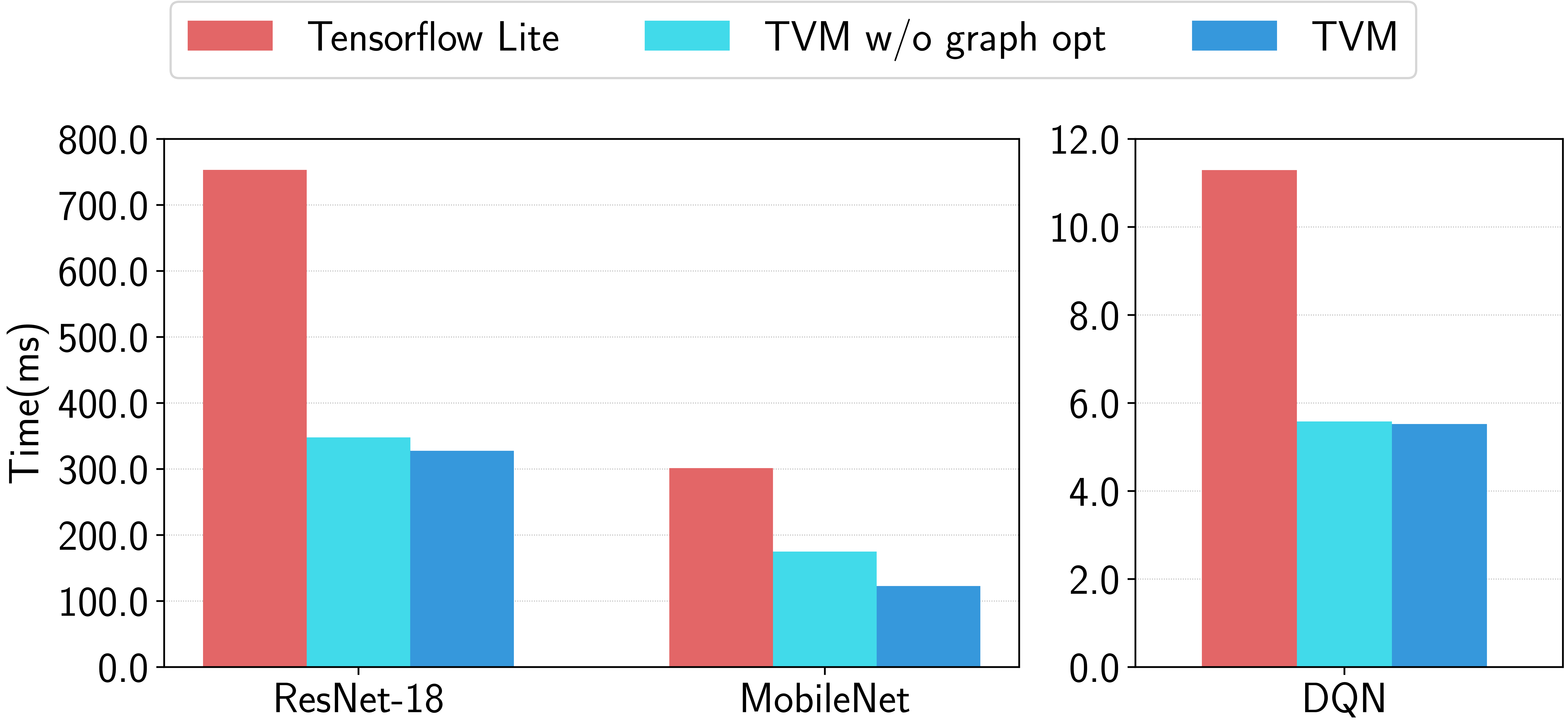
End to End Inference Performance (Nvidia Titan X)



Competitive on standard models

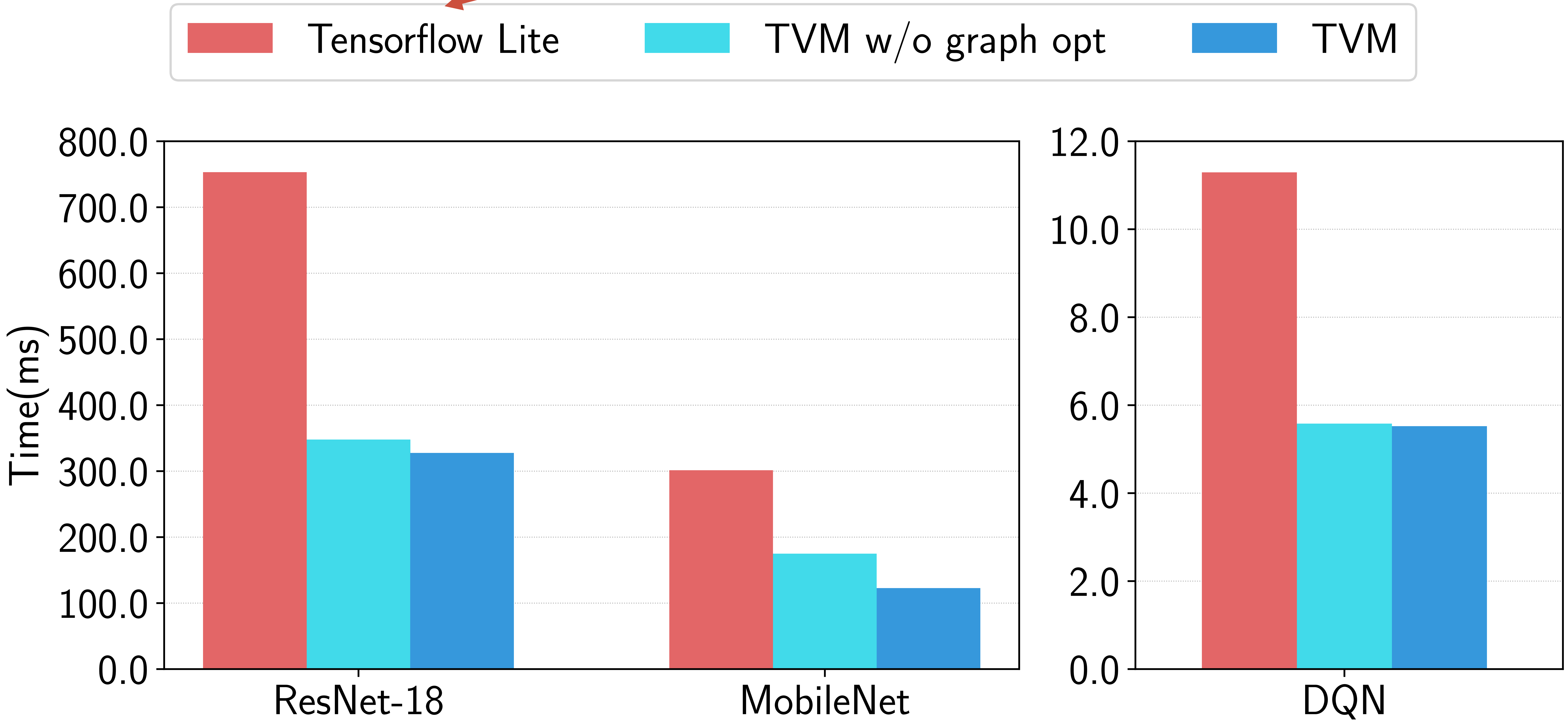
Bigger gap on less conventional models

End to End Performance(ARM Cortex-A53)

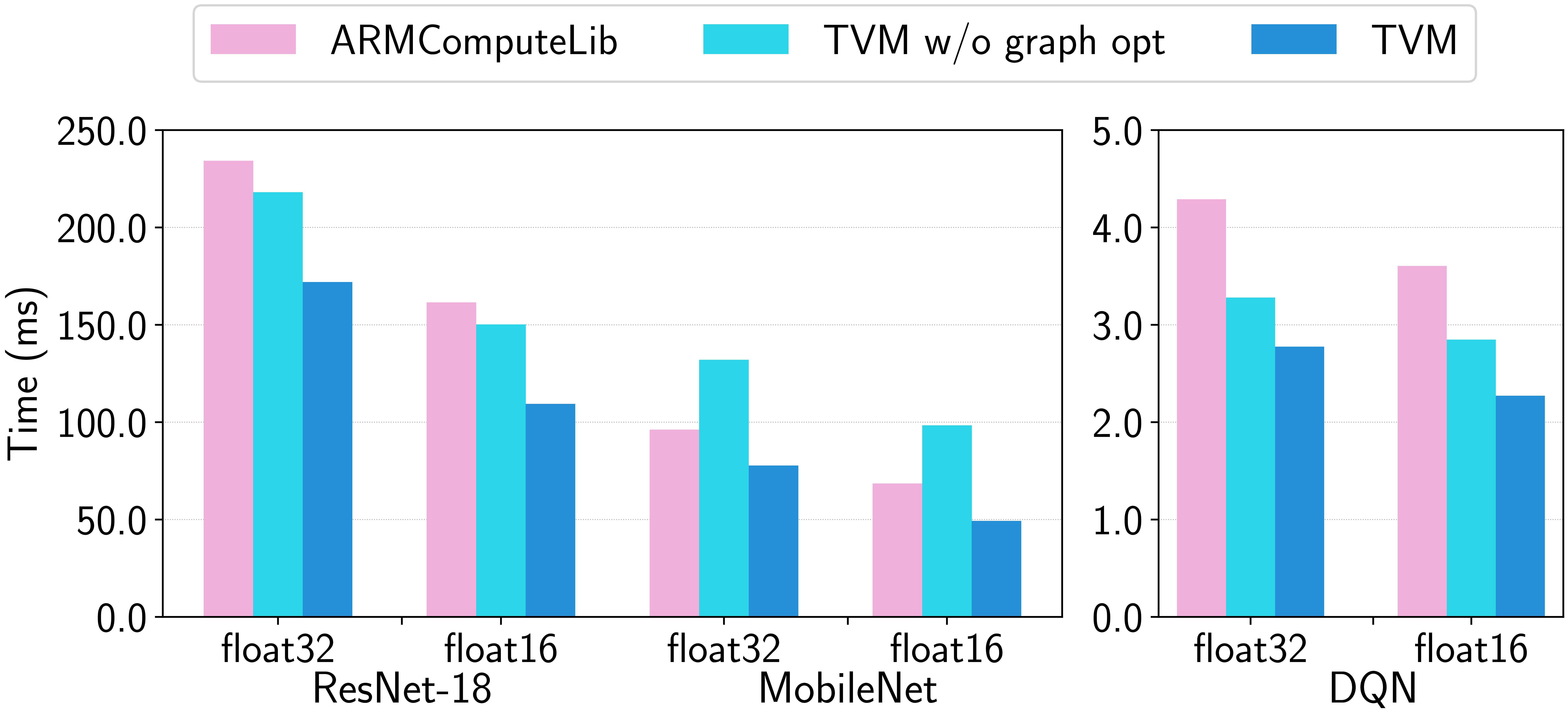


End to End Performance(ARM Cortex-A53)

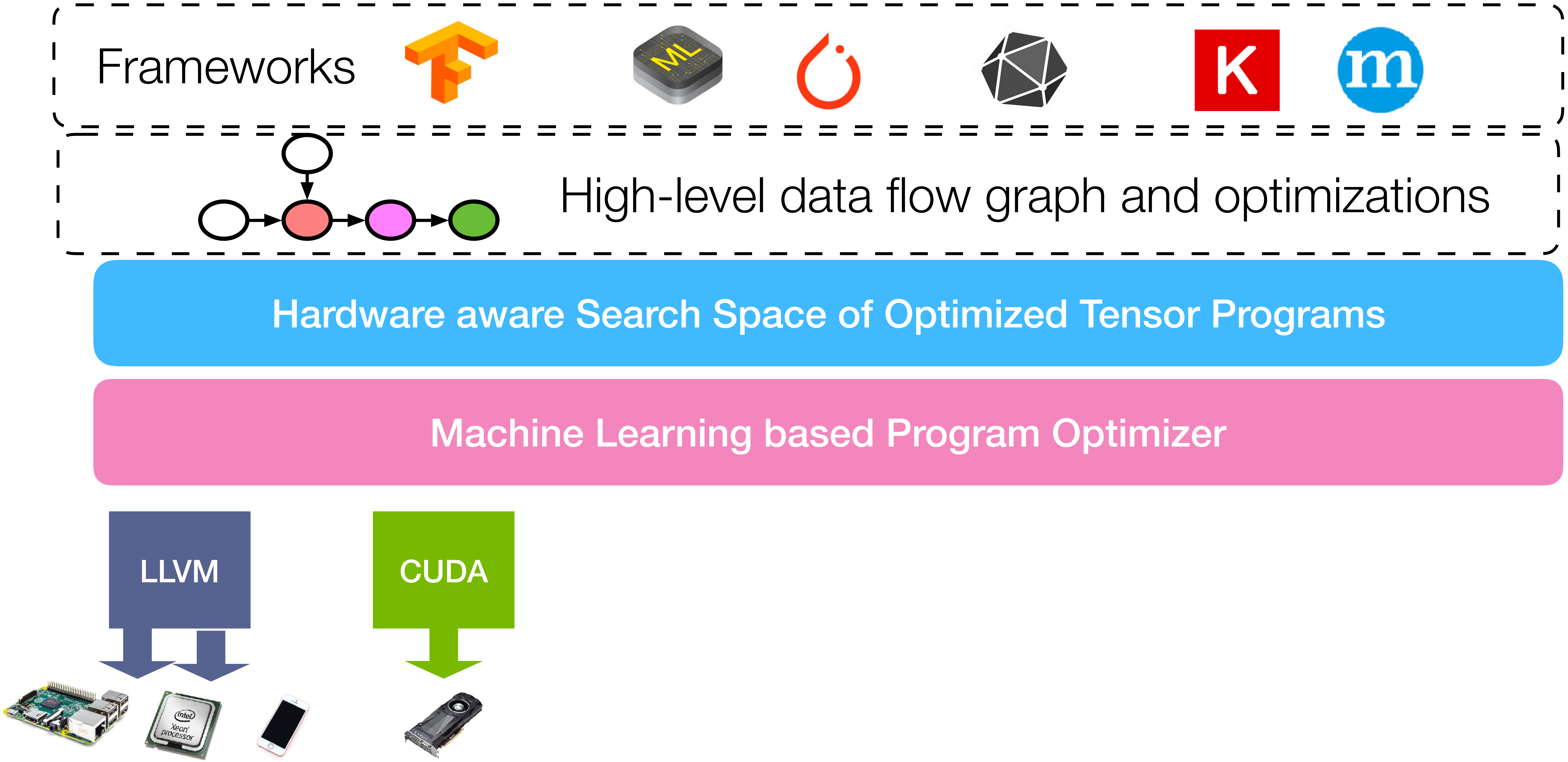
Specially optimized for Embedded system(ARM)



End to End Performance(ARM GPU)

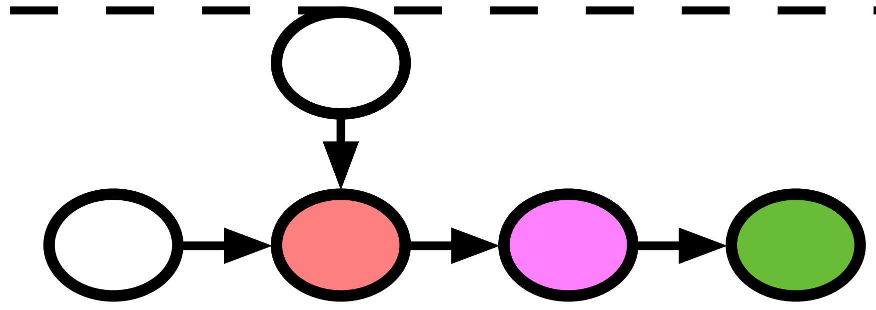


Supporting New Specialized Accelerators



Supporting New Specialized Accelerators

Frameworks



High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

Machine Learning based Program Optimizer

LLVM

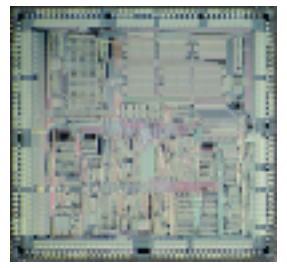
CUDA

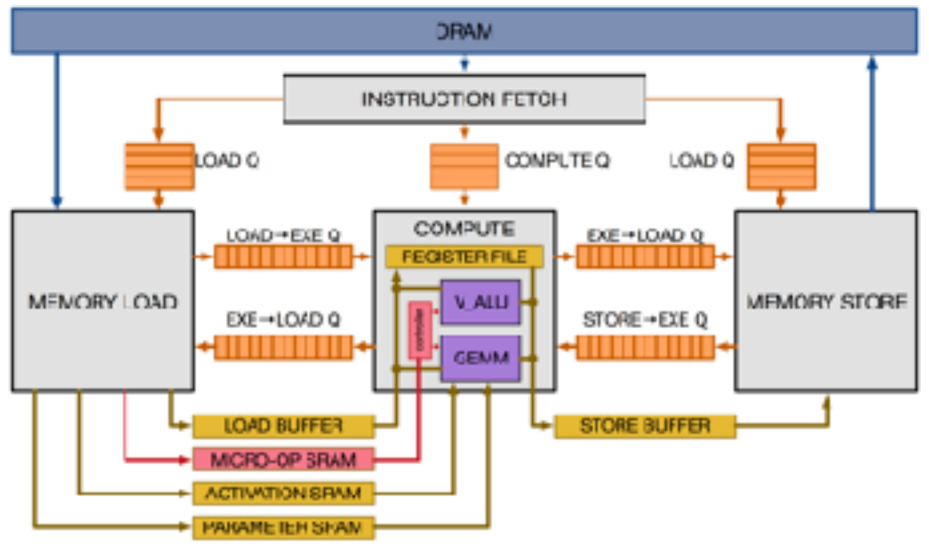
VTA: Open, Customizable Deep Learning Accelerator

Edge FPGA

Data Center FPGA

ASIC



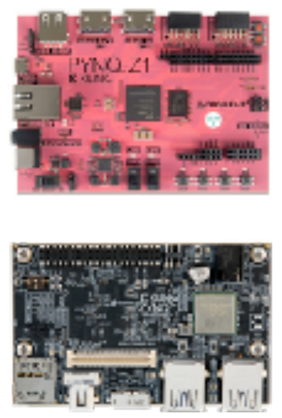


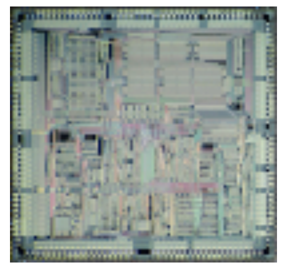
TVM/VTA: Full Stack Open Source System

High-level Optimizations

Tensor Program Search Space

ML-based Optimizer





TVM/VTA: Full Stack Open Source System

High-level Optimizations

Tensor Program Search Space

ML-based Optimizer

VTA MicroArchitecture

TVM/VTA: Full Stack Open Source System

High-level Optimizations

Tensor Program Search Space

ML-based Optimizer

VTA Hardware/Software Interface (ISA)

VTA MicroArchitecture

TVM/VTA: Full Stack Open Source System

High-level Optimizations

Tensor Program Search Space

ML-based Optimizer

VTA Runtime & JIT Compiler

VTA Hardware/Software Interface (ISA)

VTA MicroArchitecture

TVM/VTA: Full Stack Open Source System

High-level Optimizations

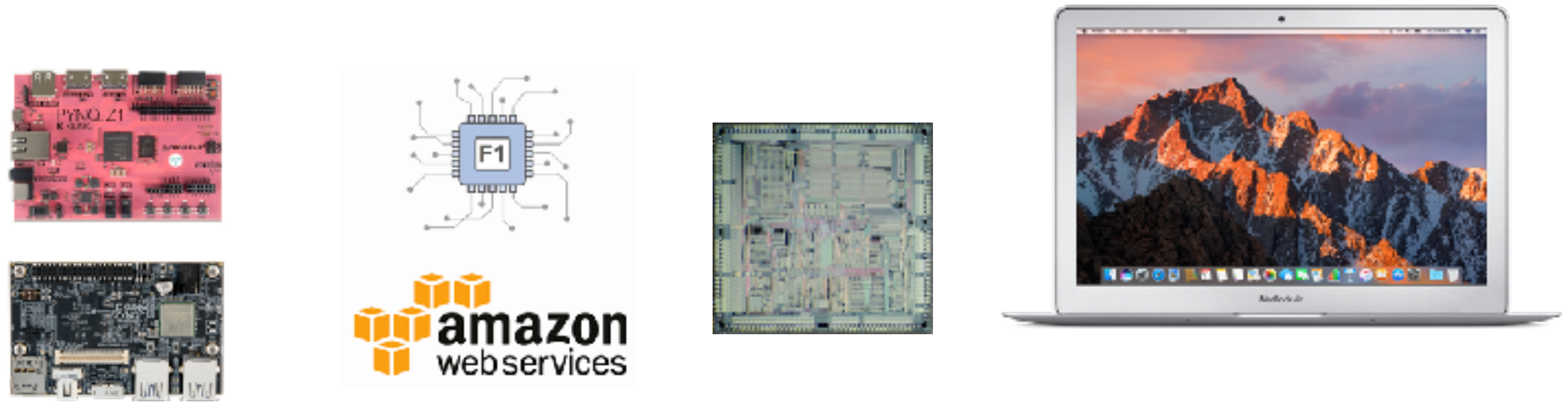
Tensor Program Search Space

ML-based Optimizer

VTA Runtime & JIT Compiler

VTA Hardware/Software Interface (ISA)

VTA MicroArchitecture | VTA Simulator



TVM/VTA: Full Stack Open Source System

- JIT compile accelerator micro code
- Support heterogenous devices, 10x better than CPU on the same board.
- Move hardware complexity to software

TVM/VTA: Full Stack Open Source System

High-level Optimizations

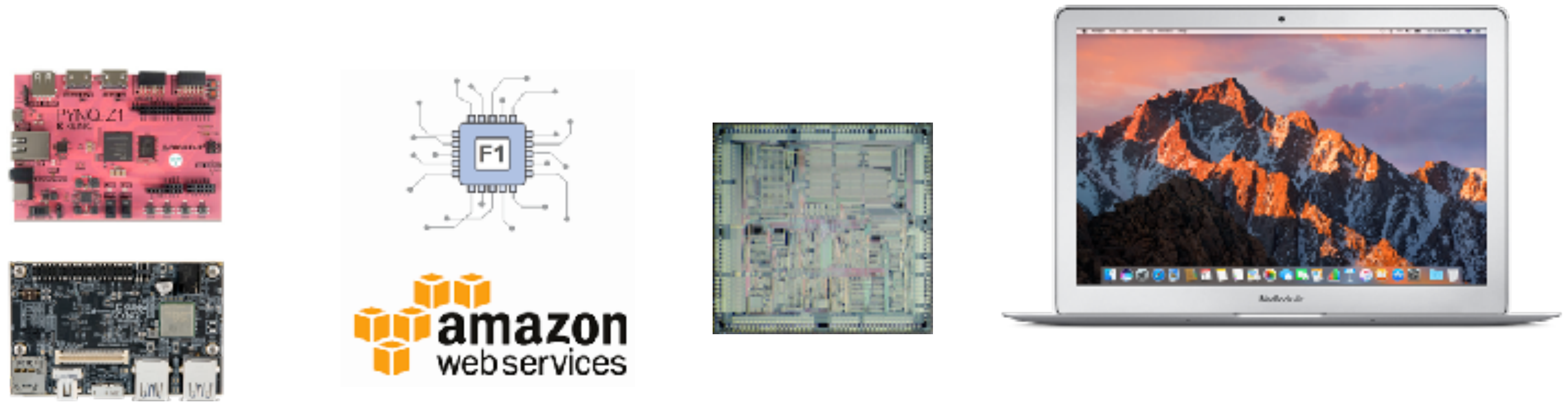
Tensor Program Search Space

ML-based Optimizer

VTA Runtime & JIT Compiler

VTA Hardware/Software Interface (ISA)

VTA MicroArchitecture | VTA Simulator



- JIT compile accelerator micro code
- Support heterogenous devices, 10x better than CPU on the same board.
- Move hardware complexity to software

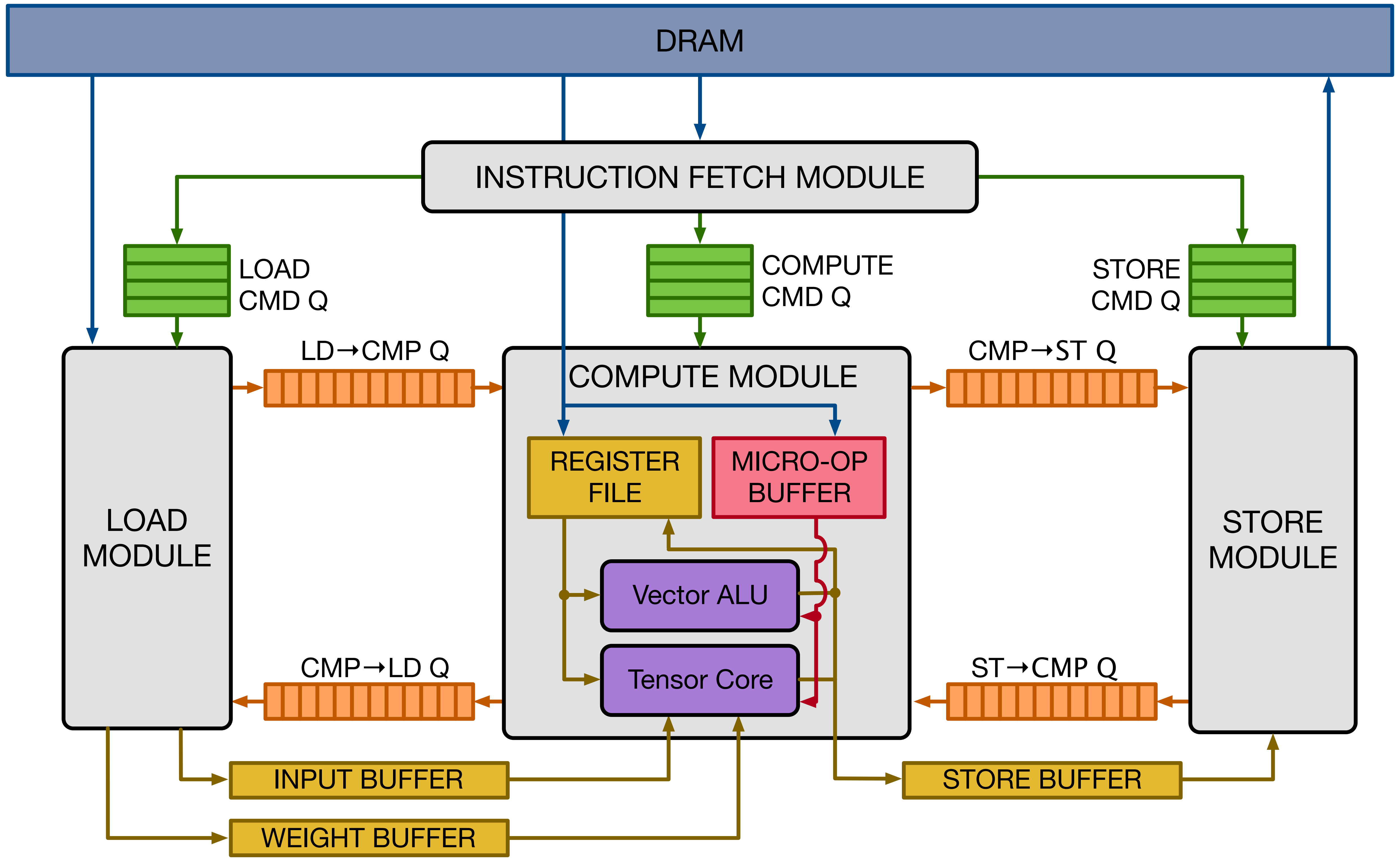
compiler, driver,
hardware design
full stack open source

VTA Hardware Architecture

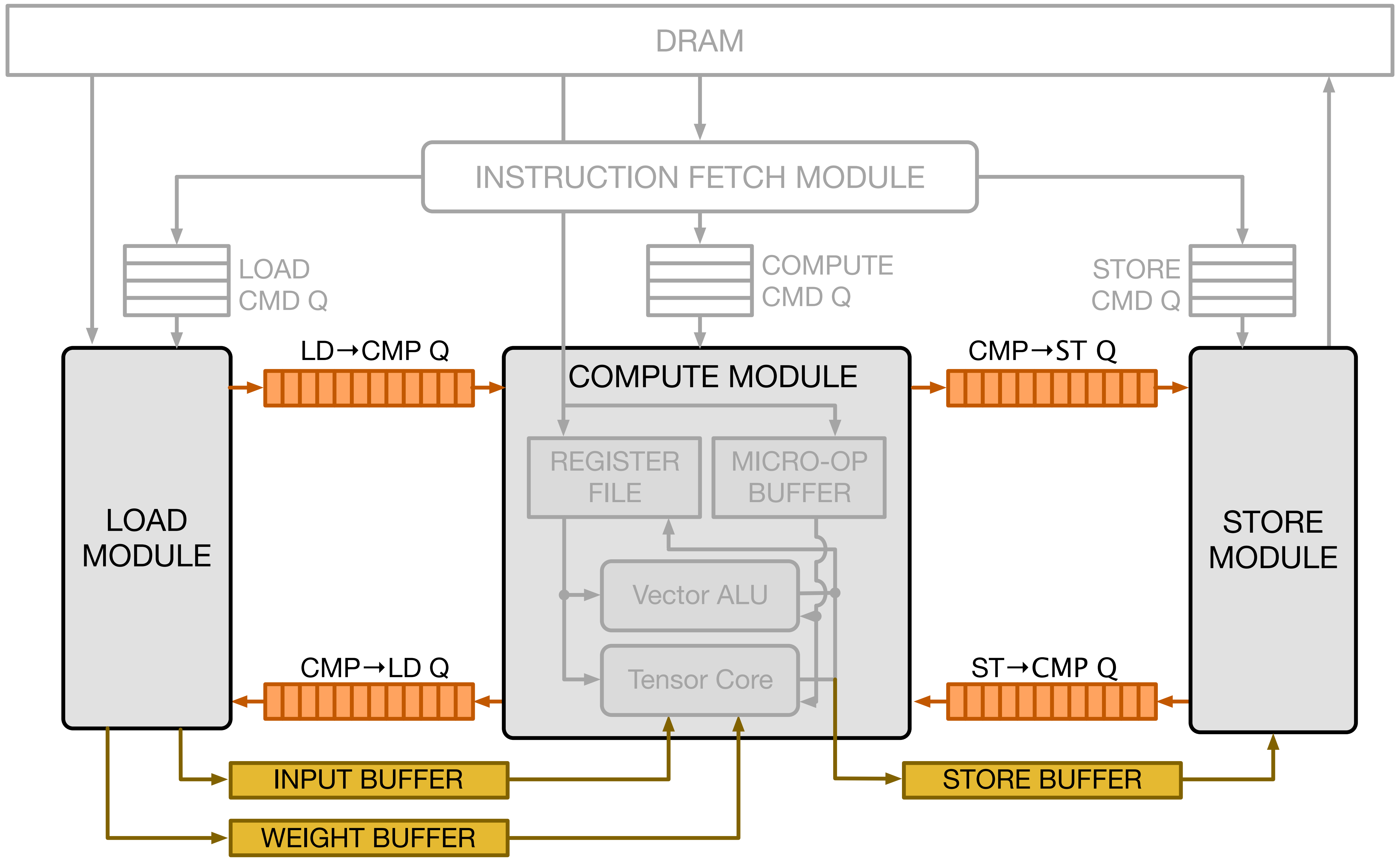
Philosophy: simple hardware, provide software-defined flexibility

VTA Hardware Architecture

Philosophy: simple hardware, provide software-defined flexibility



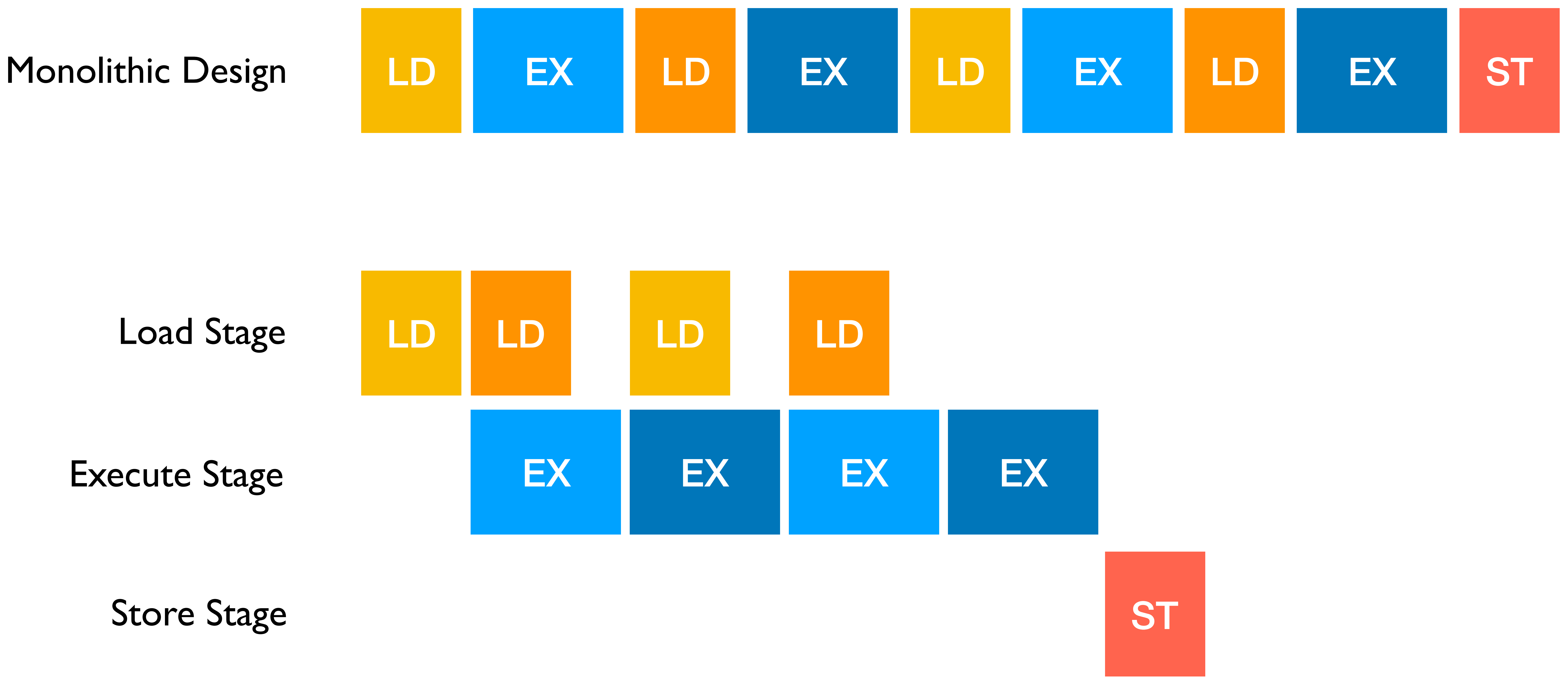
VTA Hardware Architecture



Pipelining Tasks to Hide Memory Latency

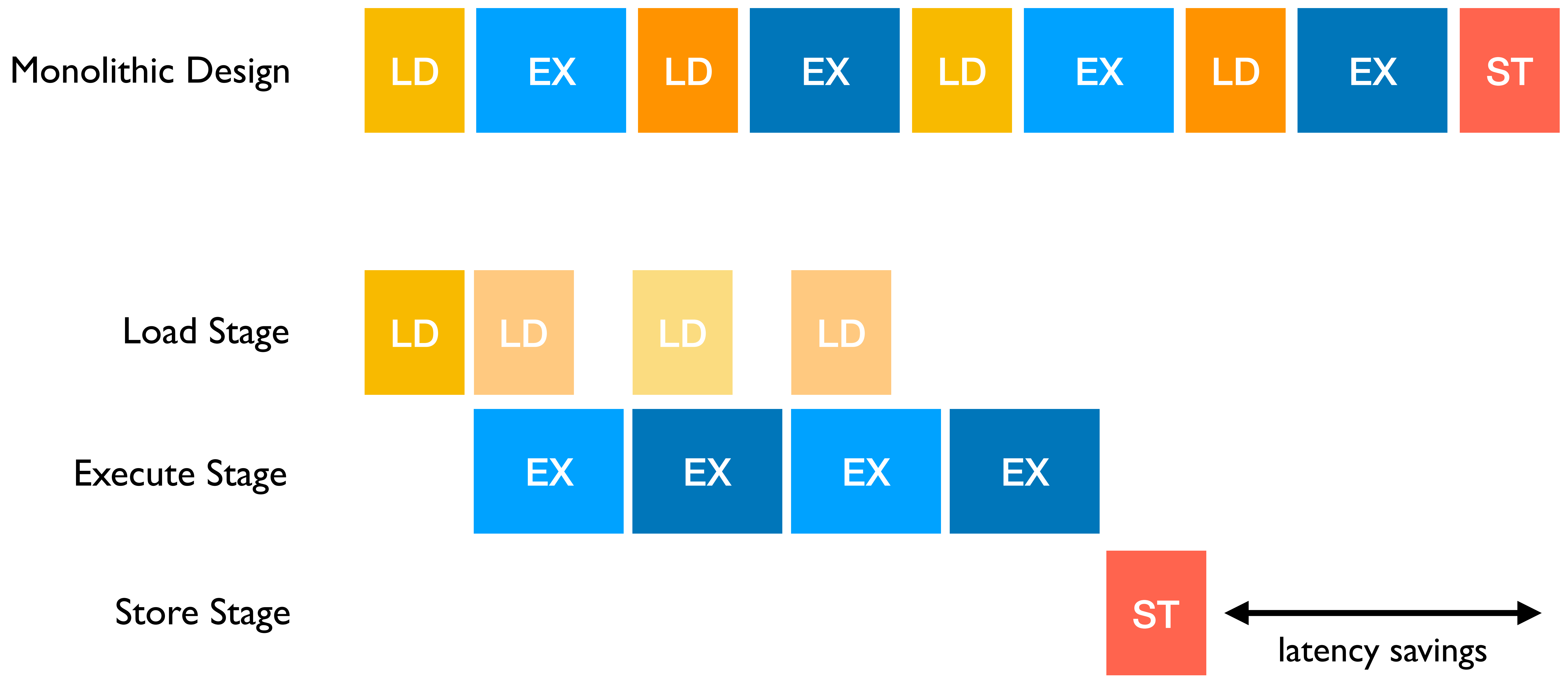
LD: load
EX: compute
ST: store

Pipelining Tasks to Hide Memory Latency



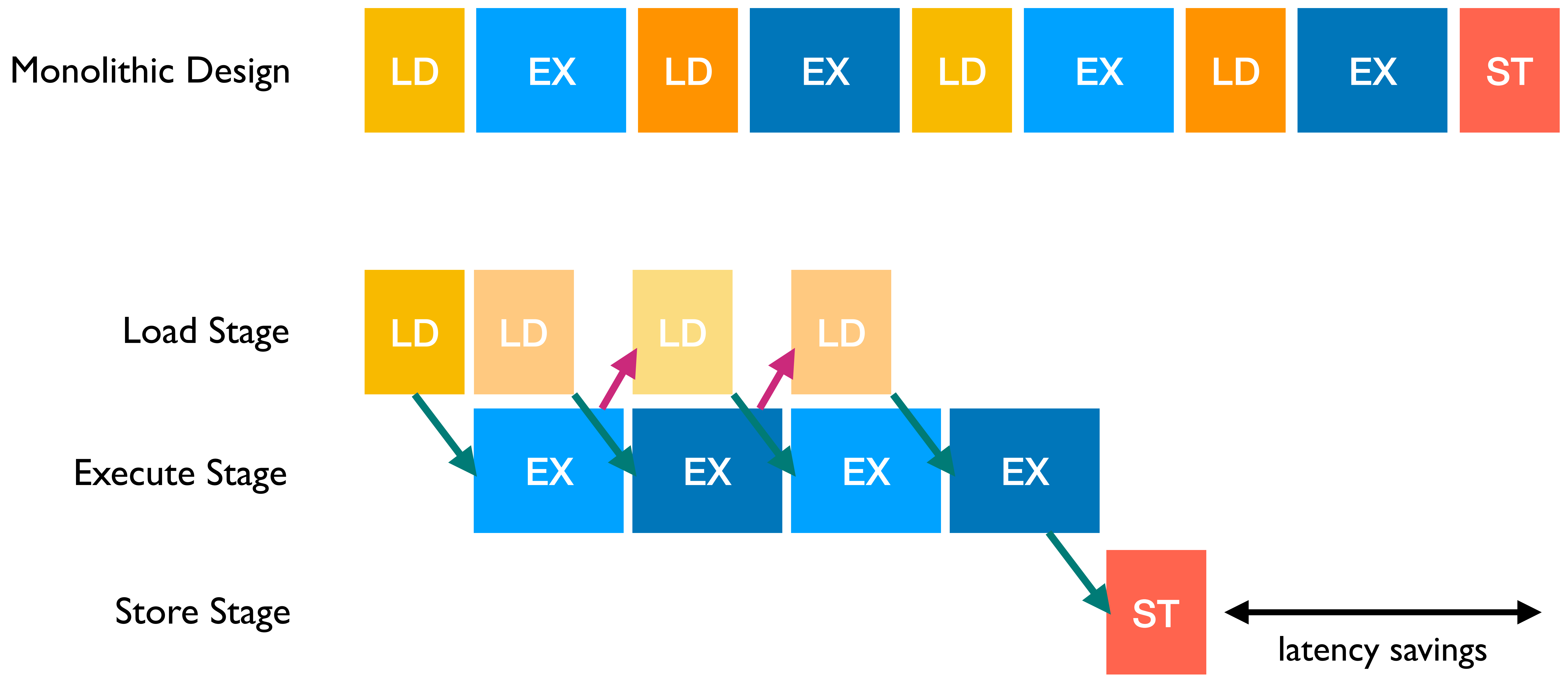
LD: load
EX: compute
ST: store

Pipelining Tasks to Hide Memory Latency



LD: load
EX: compute
ST: store

Pipelining Tasks to Hide Memory Latency



low-level synchronization between tasks is explicitly managed by the software

LD: load
EX: compute
ST: store

Two-Level ISA Overview

Provides the right tradeoff between expressiveness and code compactness

Two-Level ISA Overview

Provides the right tradeoff between expressiveness and code compactness

- Use CISC instructions to perform multi-cycle tasks

DENSE ALU LOAD STORE

Two-Level ISA Overview

Provides the right tradeoff between expressiveness and code compactness

- Use CISC instructions to perform multi-cycle tasks

DENSE	ALU	LOAD	STORE

- Use RISC micro-ops to perform single-cycle tensor operations

Two-Level ISA Overview

Provides the right tradeoff between expressiveness and code compactness

- Use CISC instructions to perform multi-cycle tasks

DENSE	ALU	LOAD	STORE

- Use RISC micro-ops to perform single-cycle tensor operations

```
R0: R0 + GEMM(A8, W3)
```

Two-Level ISA Overview

Provides the right tradeoff between expressiveness and code compactness

- Use CISC instructions to perform multi-cycle tasks

DENSE	ALU	LOAD	STORE

- Use RISC micro-ops to perform single-cycle tensor operations

```
R0: R0 + GEMM(A8, W3)
R2: MAX(R0, ZERO)
```

VTA RISC Micro-Kernels

VTA RISC Micro-Kernels

multiple RISC instructions define a **micro-kernel**,
which can be invoked by a CISC instruction

VTA RISC Micro-Kernels

multiple RISC instructions define a **micro-kernel**,
which can be invoked by a CISC instruction

```
CONV2D: layout=NCHW, chan=128, kernel=(3,3), padding=(1,1), strides=(1,1)
```

VTA RISC Micro-Kernels

multiple RISC instructions define a **micro-kernel**,
which can be invoked by a CISC instruction

```
CONV2D: layout=NCHW, chan=128, kernel=(3,3), padding=(1,1), strides=(1,1)
```

```
CONV2D: layout=NCHW, chan=256, kernel=(1,1), padding=(0,0), strides=(2,2)
```

VTA RISC Micro-Kernels

multiple RISC instructions define a **micro-kernel**,
which can be invoked by a CISC instruction

```
CONV2D: layout=NCHW, chan=128, kernel=(3,3), padding=(1,1), strides=(1,1)
```

```
CONV2D: layout=NCHW, chan=256, kernel=(1,1), padding=(0,0), strides=(2,2)
```

```
CONV2D_TRANSPOSE: ...
```

VTA RISC Micro-Kernels

multiple RISC instructions define a **micro-kernel**,
which can be invoked by a CISC instruction

```
CONV2D: layout=NCHW, chan=128, kernel=(3,3), padding=(1,1), strides=(1,1)
```

```
CONV2D: layout=NCHW, chan=256, kernel=(1,1), padding=(0,0), strides=(2,2)
```

```
CONV2D_TRANSPOSE: ...
```

```
GROUP_CONV2D: ...
```

VTA RISC Micro-Kernels

micro-kernel programming gives us
software-defined flexibility

DCGAN

ResNet50

TVM: Learning-based Deep Learning Compiler

TVM: Learning-based Deep Learning Compiler

High-Level Differentiable IR

TVM: Learning-based Deep Learning Compiler

High-Level Differentiable IR

Tensor Expression IR

TVM: Learning-based Deep Learning Compiler

High-Level Differentiable IR

Tensor Expression IR

LLVM, CUDA, Metal

TVM: Learning-based Deep Learning Compiler

High-Level Differentiable IR

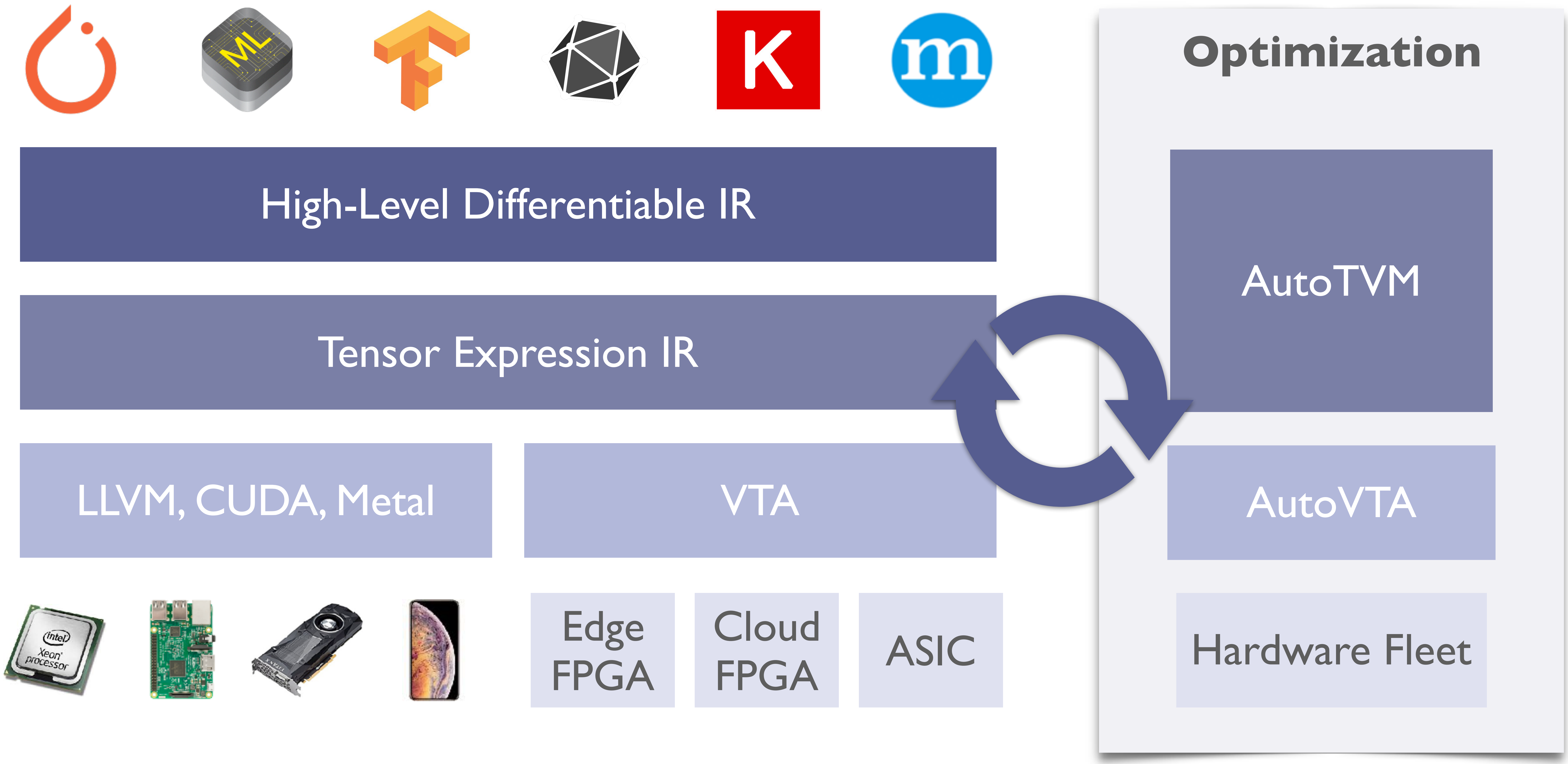
Tensor Expression IR

LLVM, CUDA, Metal | VTA

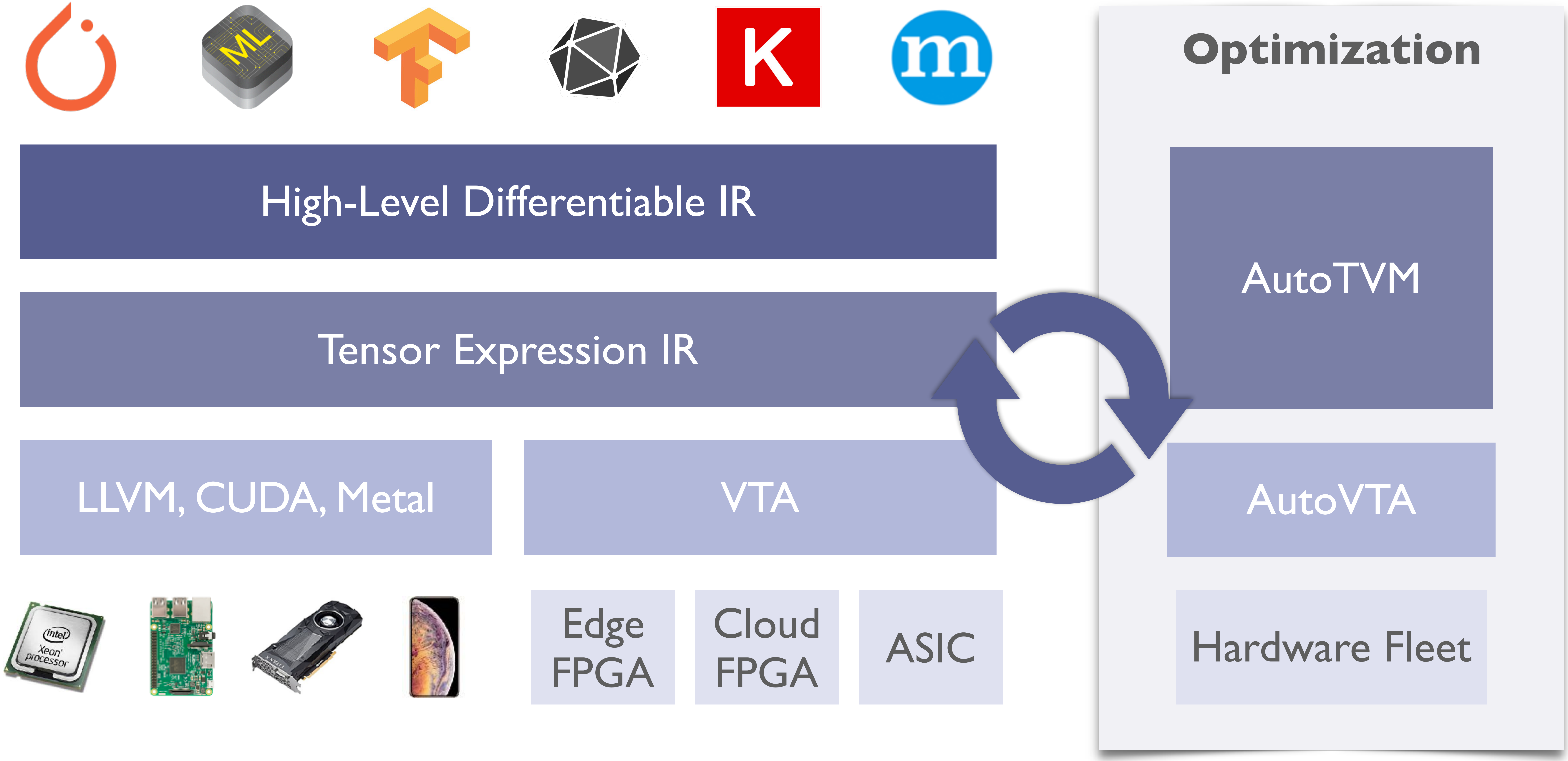
Edge FPGA | Cloud FPGA | ASIC

TVM: Learning-based Deep Learning Compiler

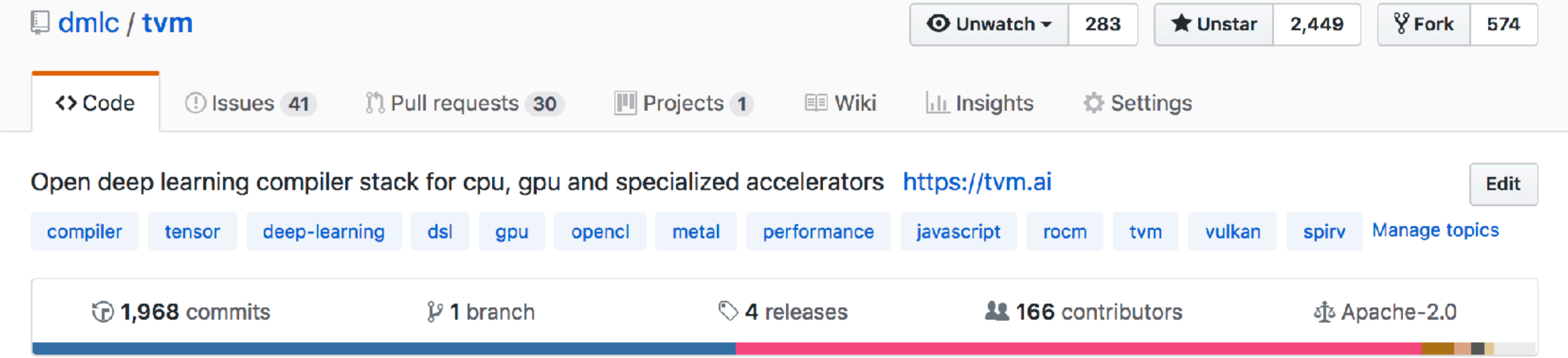
TVM: Learning-based Deep Learning Compiler



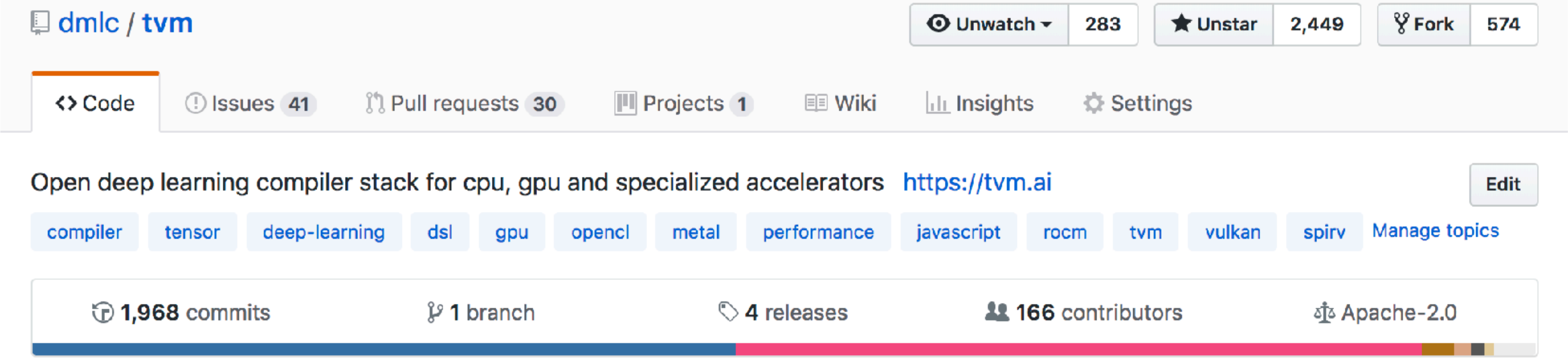
TVM: Learning-based Deep Learning Compiler



TVM Open Source Community



TVM Open Source Community



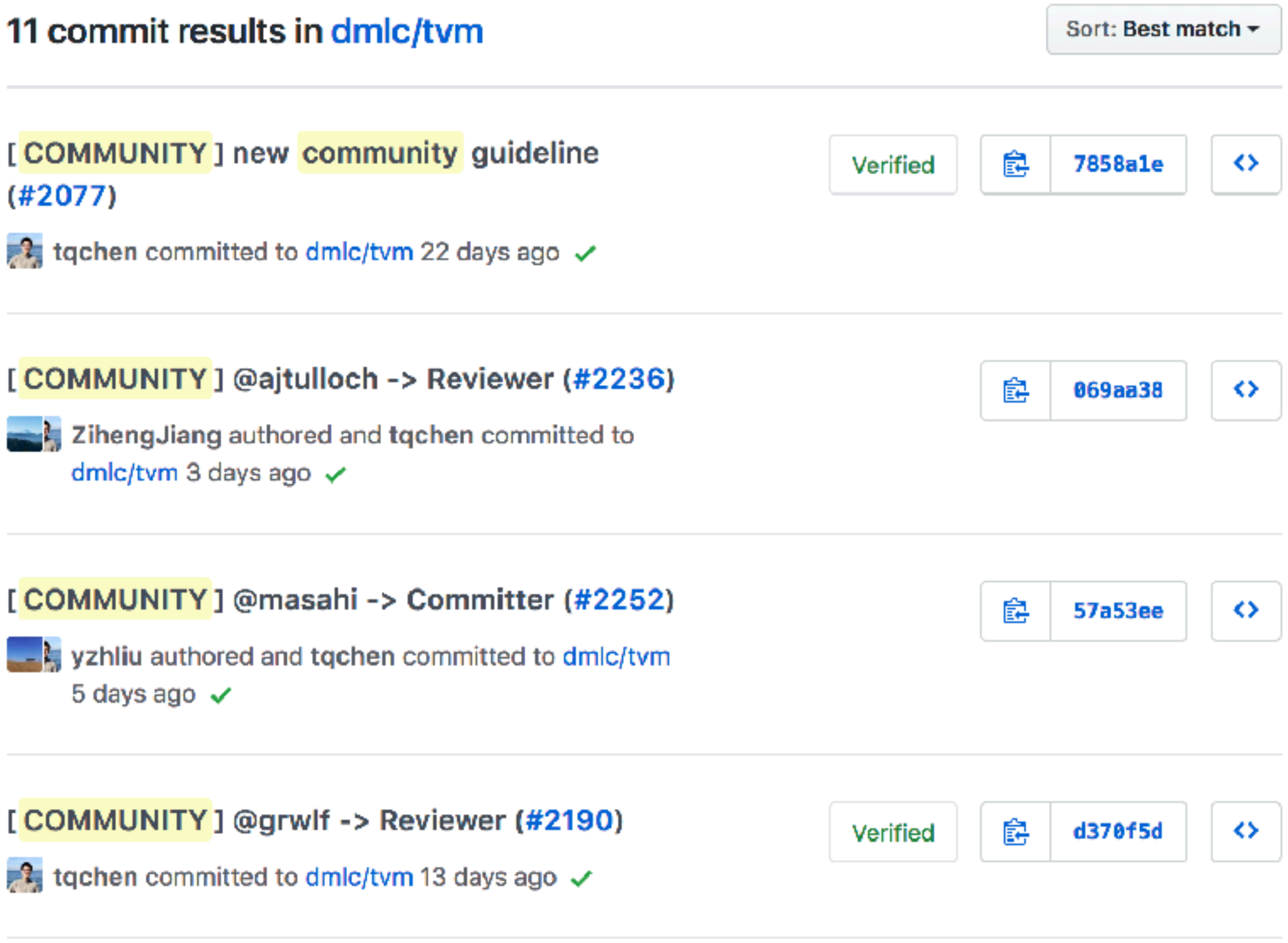
Apache governance model: grant project ownership by merit.

11 committers, 29 reviewers, 166 contributors.

Contributed by the community, for the community.

TVM **Open** Source Community

- Prefer public archivable discussion
- Open RFC discussion
- Bring in new members by merit



https://docs.tvm.ai/contribute/community.html

TVM in Productions

TVM in Productions

- AWS: Deep Learning Compiler in SageMaker Neo.
- Huawei: Compiler support for Ascent AI ASIC Chip.
- FB: caffe2/pytorch automatic optimization on mobile devices.
- https://sampl.cs.washington.edu/tvmconf/

TVM: Learning-based Deep Learning Compiler

TVM: Learning-based Deep Learning Compiler

High-Level Differentiable IR

TVM: Learning-based Deep Learning Compiler

High-Level Differentiable IR

Tensor Expression IR

TVM: Learning-based Deep Learning Compiler

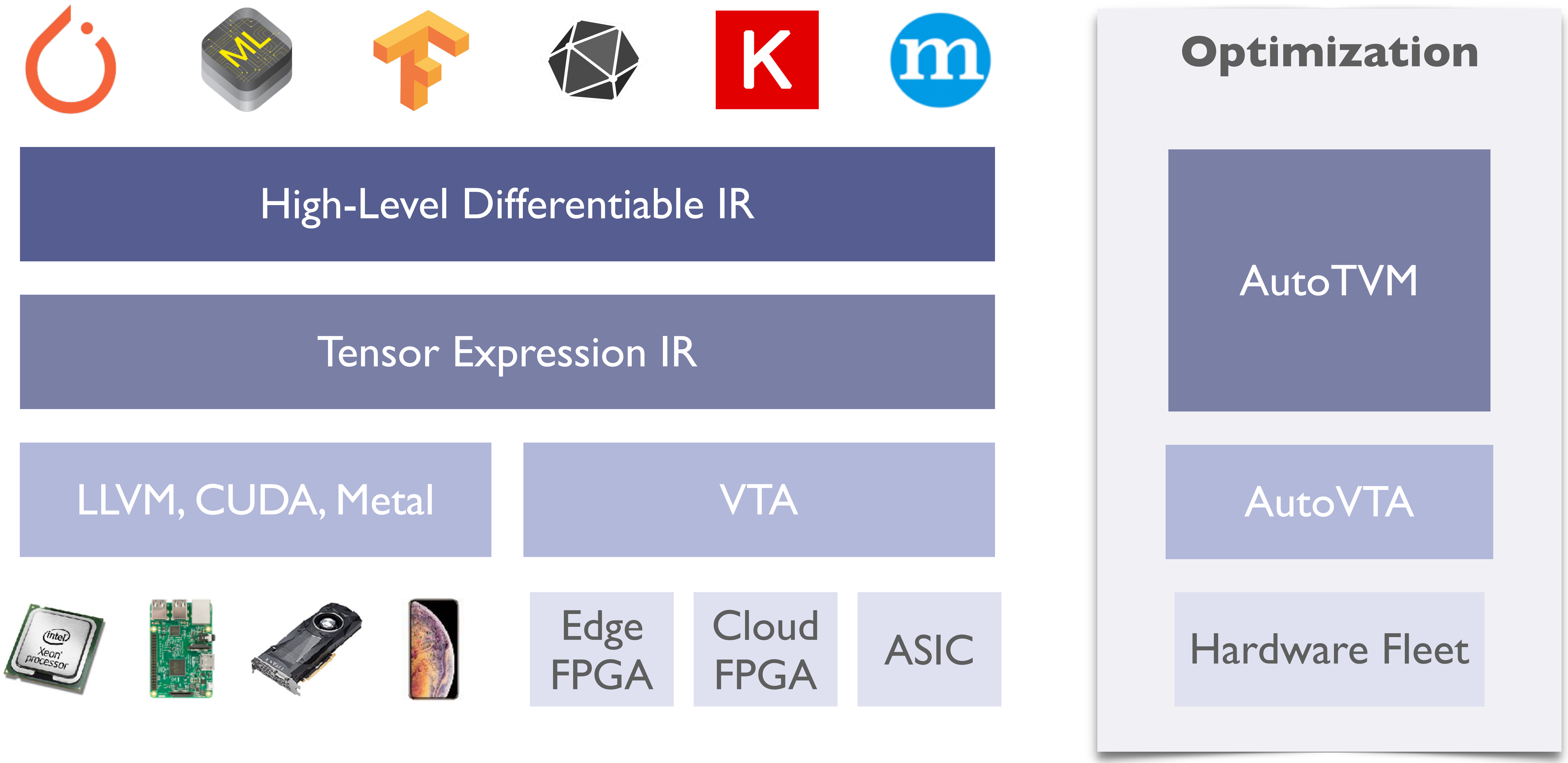
High-Level Differentiable IR

Tensor Expression IR

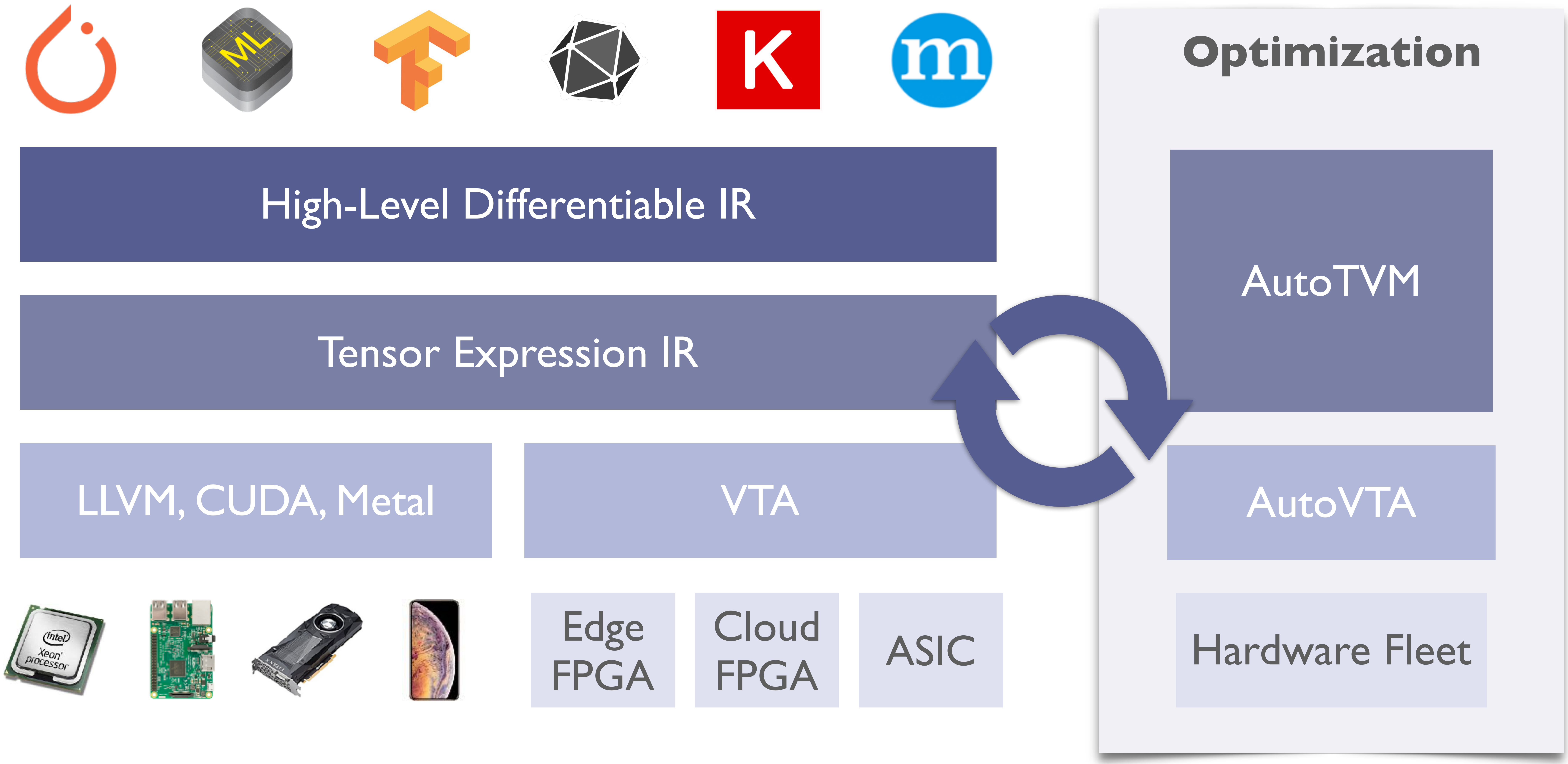
LLVM, CUDA, Metal

TVM: Learning-based Deep Learning Compiler

TVM: Learning-based Deep Learning Compiler



TVM: Learning-based Deep Learning Compiler



TVM: Learning-based Deep Learning Compiler