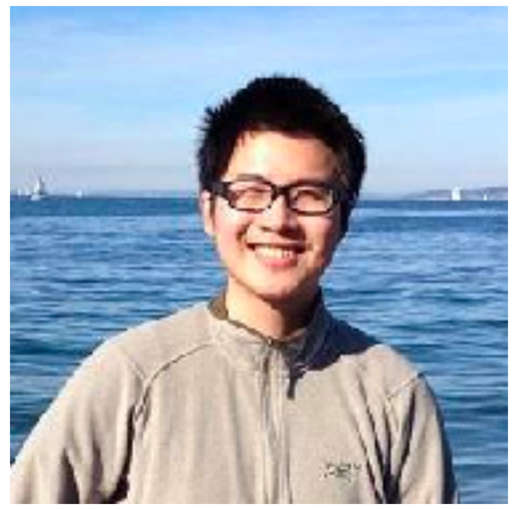
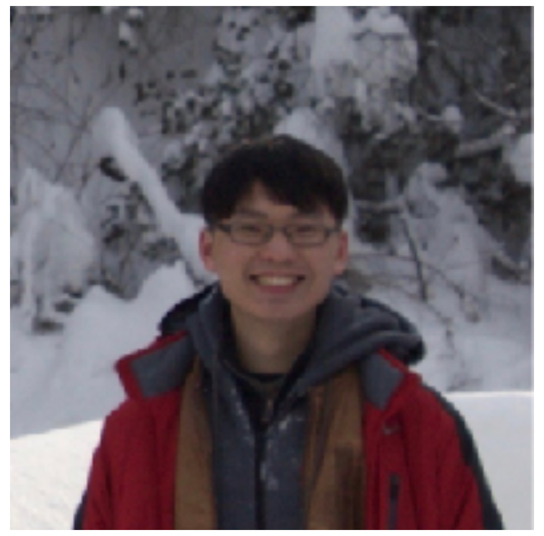
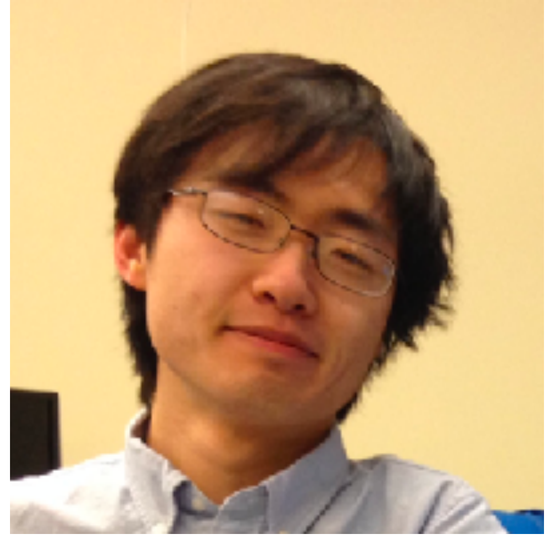
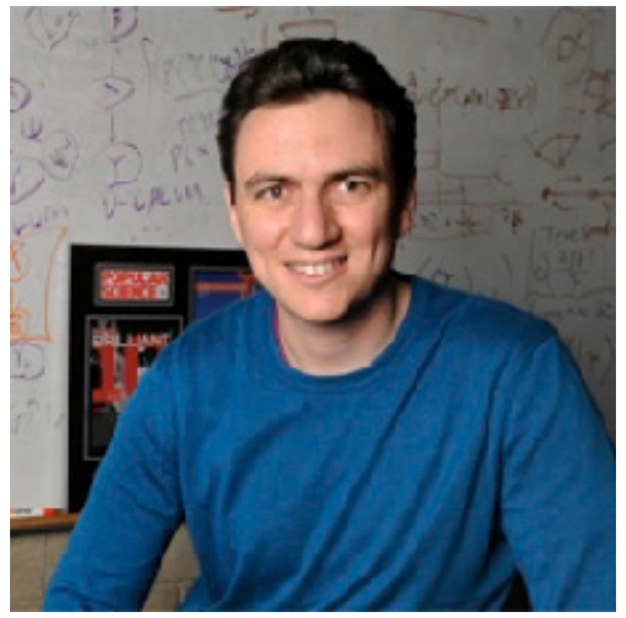
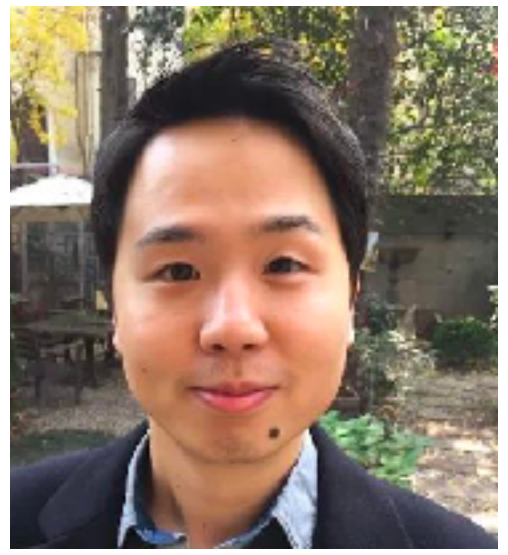
TVM: An Automated End-to-End Optimizing Compiler for Deep Learning

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

Collaborators

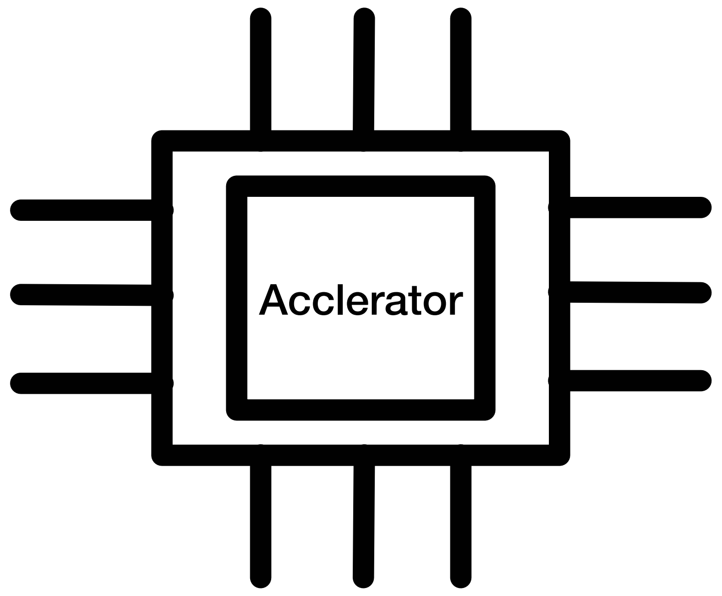
     

SAMPL Lab: https://sampl.cs.washington.edu

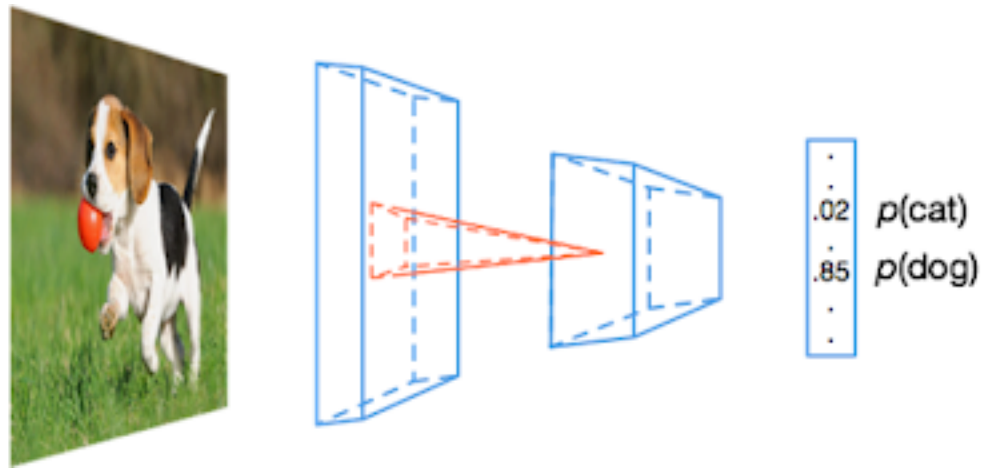
sampl

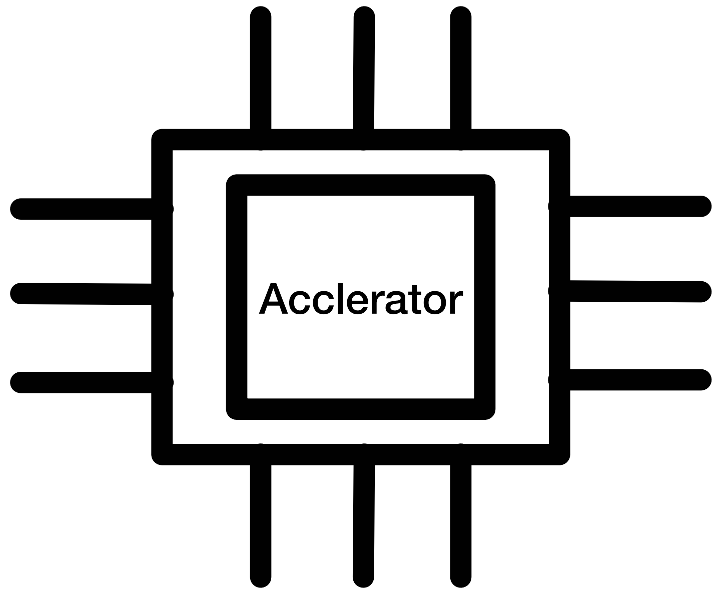
Beginning of Story

Beginning of Story

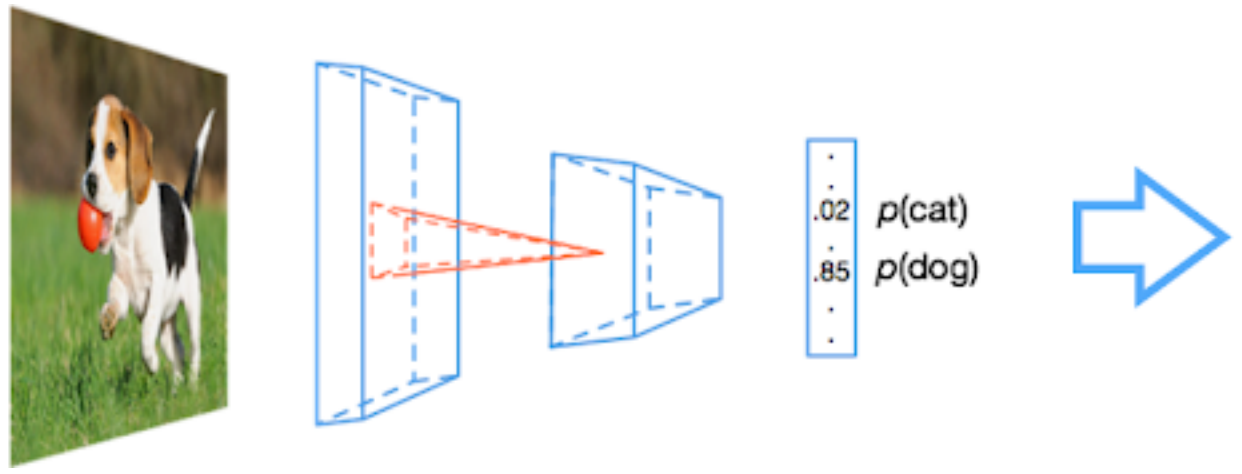


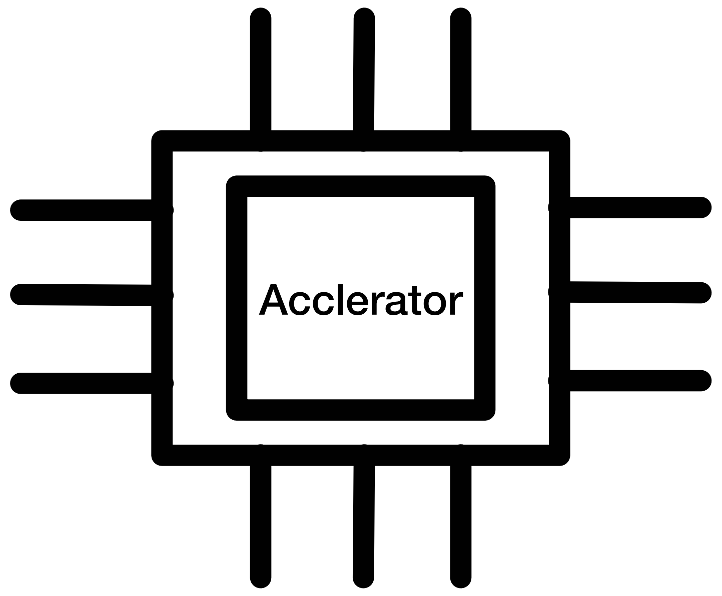
Beginning of Story



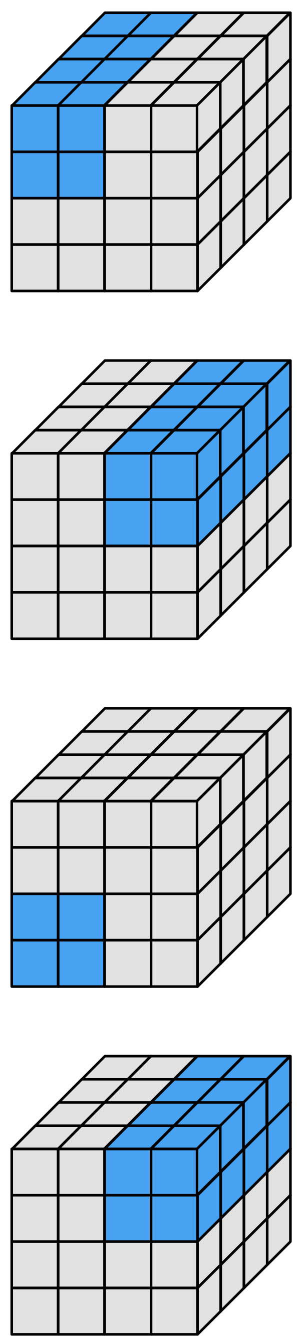


Beginning of Story






```
// 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

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

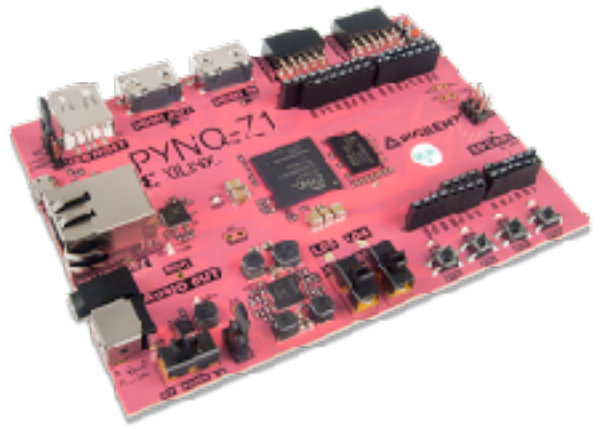
Frameworks

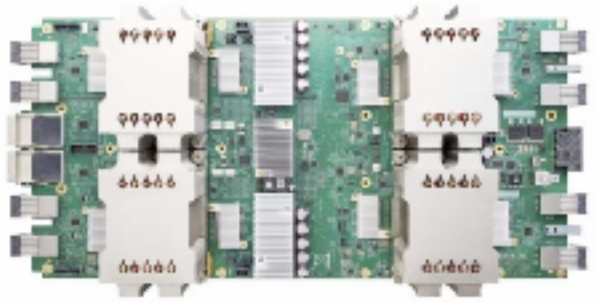
Explosion of models and frameworks

Explosion of hardware backends







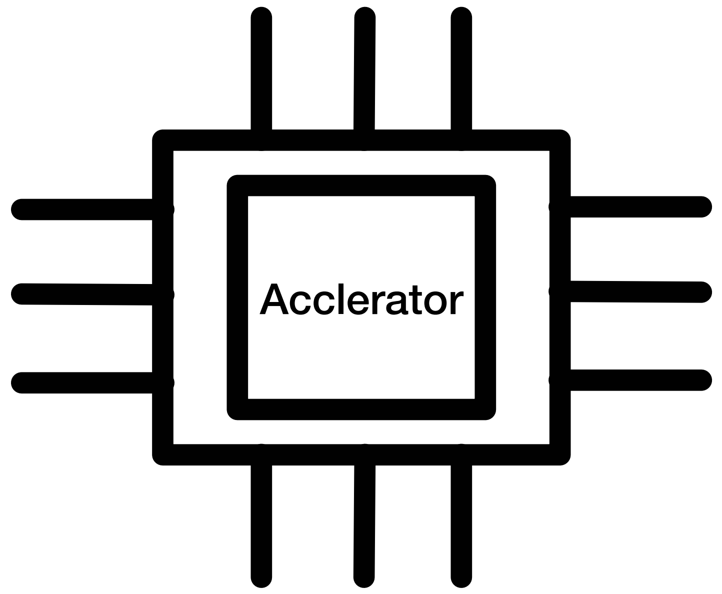


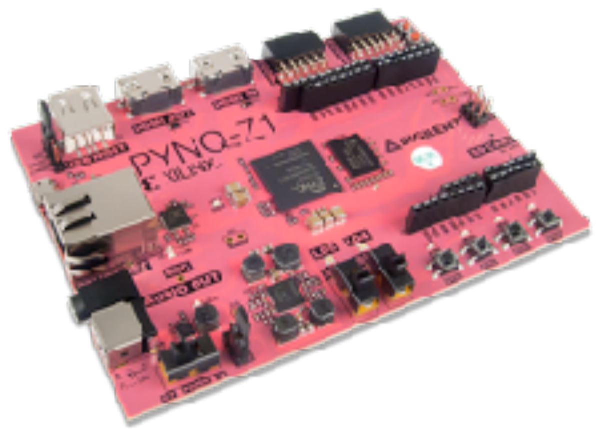
Goal: Deploy Deep Learning Everywhere

Frameworks

Explosion of models and frameworks

Explosion of hardware backends





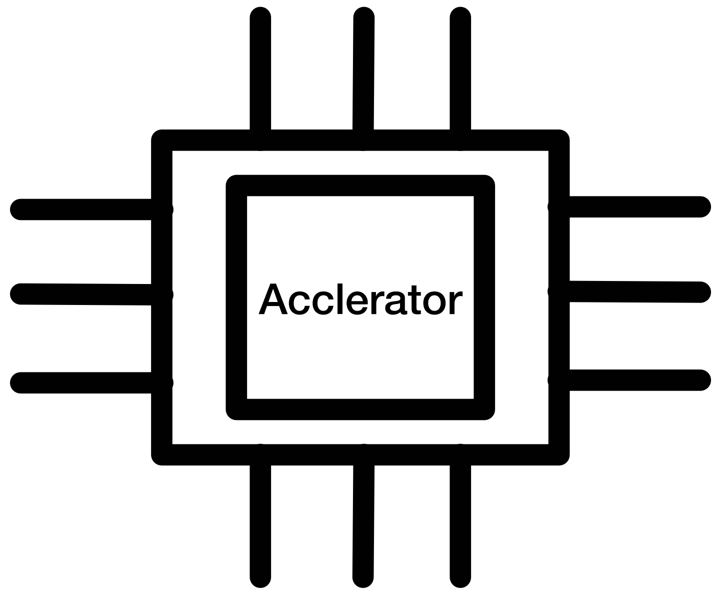
Goal: Deploy Deep Learning Everywhere

Frameworks

Explosion of models and frameworks

Huge gap between model/frameworks and hardware backends

Explosion of hardware backends



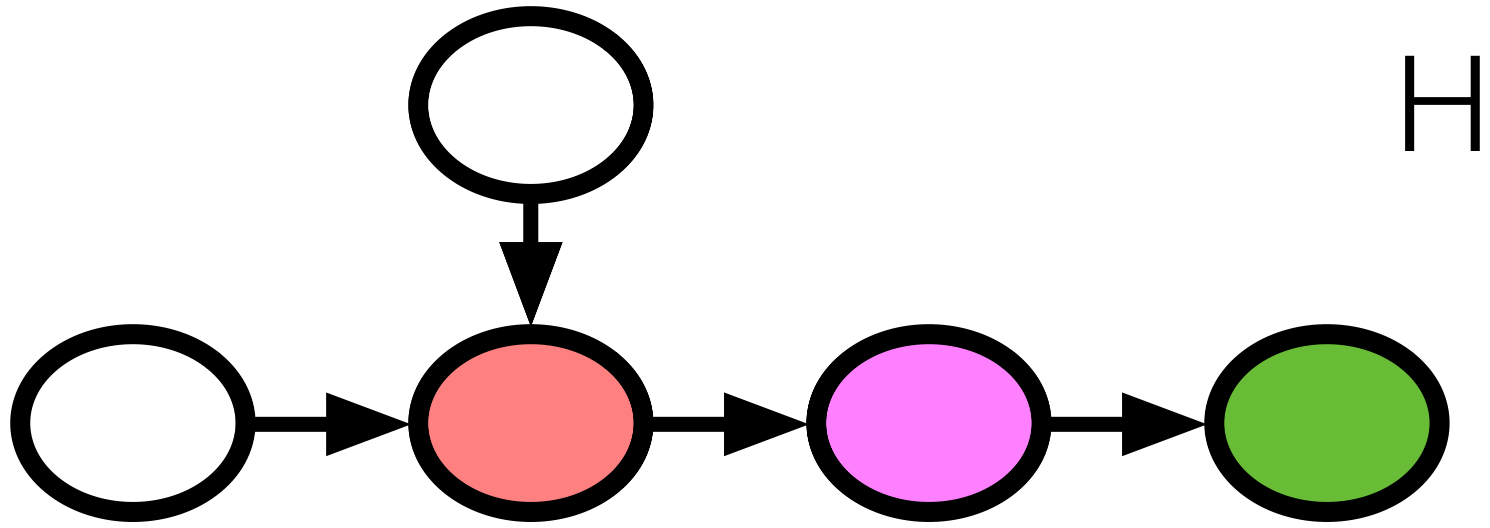
Existing Approach

Frameworks

Hardware

Existing Approach

Frameworks

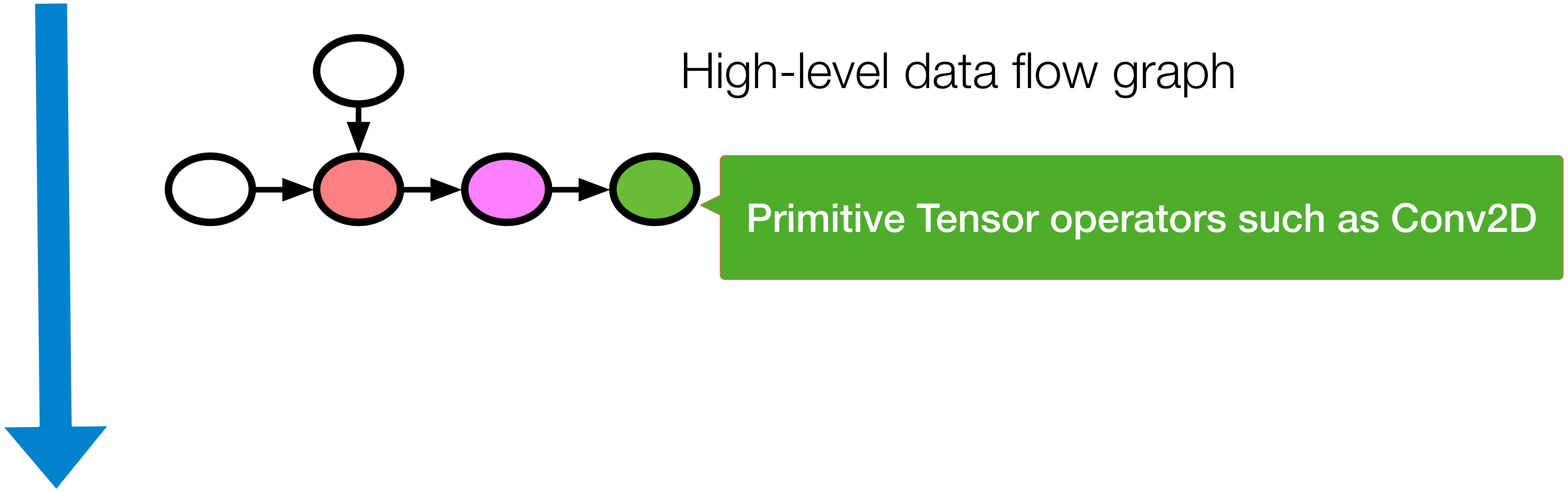


High-level data flow graph

Hardware

Existing Approach

Frameworks

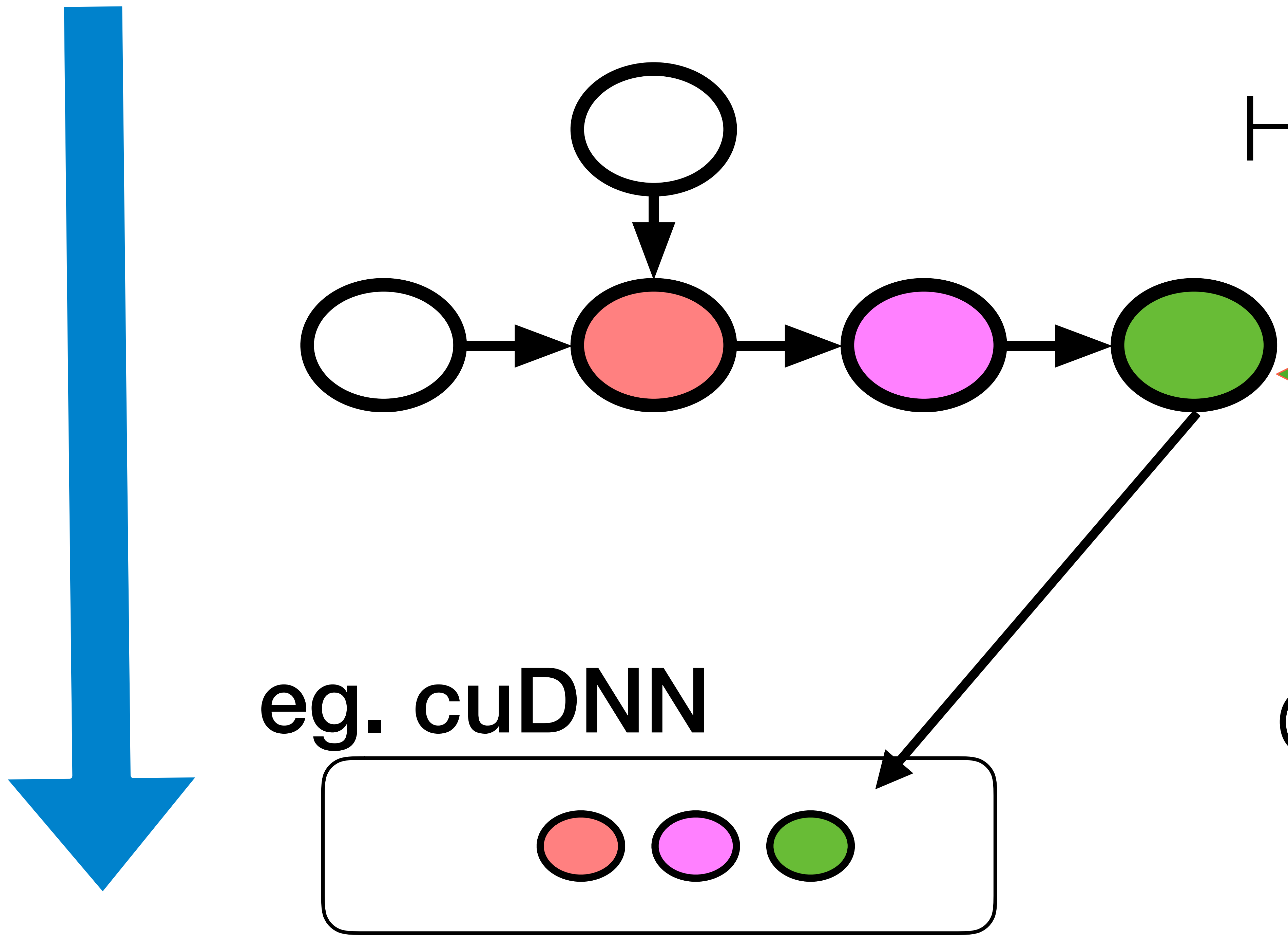


Hardware

Existing Approach

Frameworks

High-level data flow graph

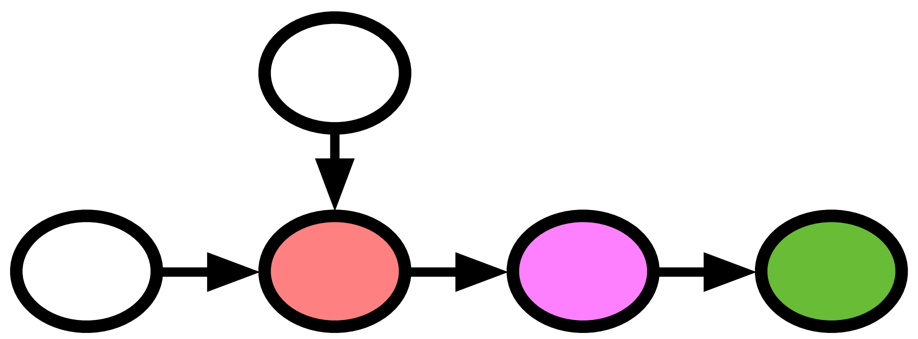


Primitive Tensor operators such as Conv2D

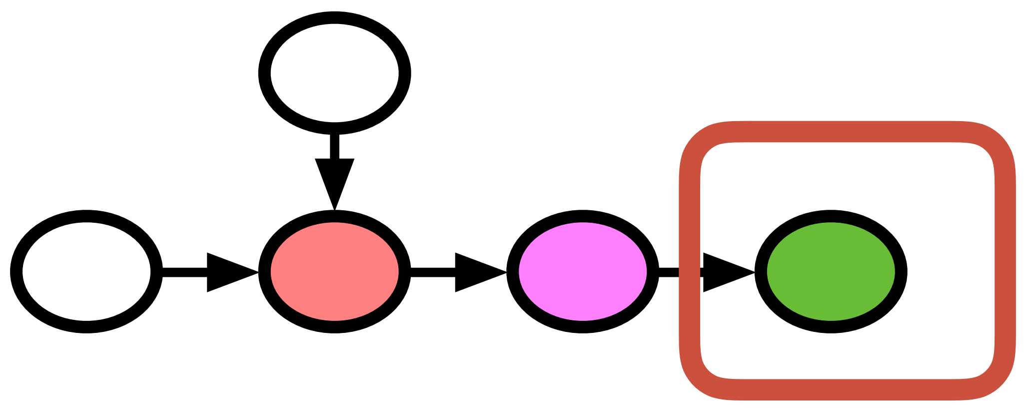
Offload to heavily optimized
DNN operator library

Hardware

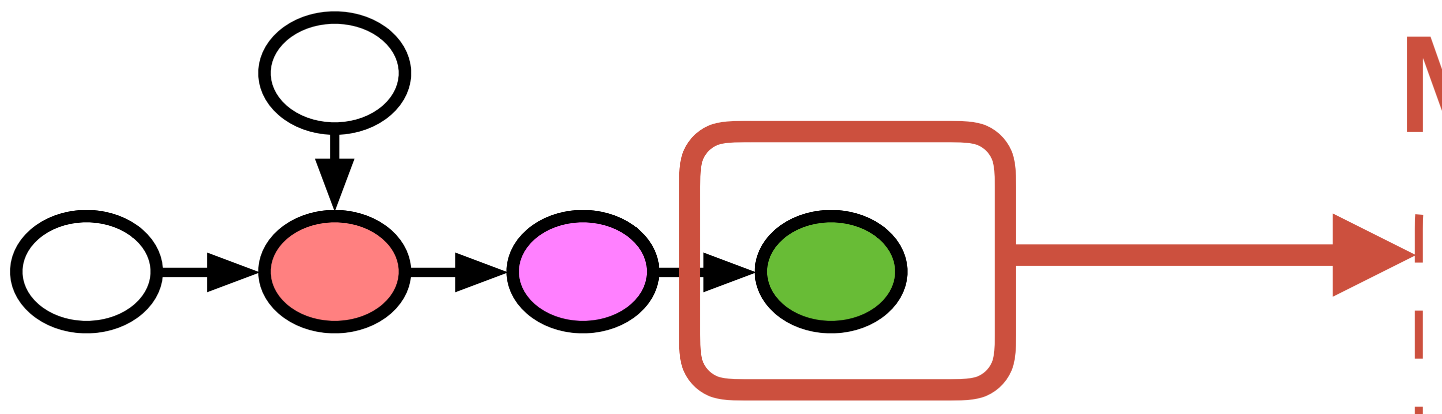
Existing Approach: Engineer Optimized Tensor Operators



Existing Approach: Engineer Optimized Tensor Operators



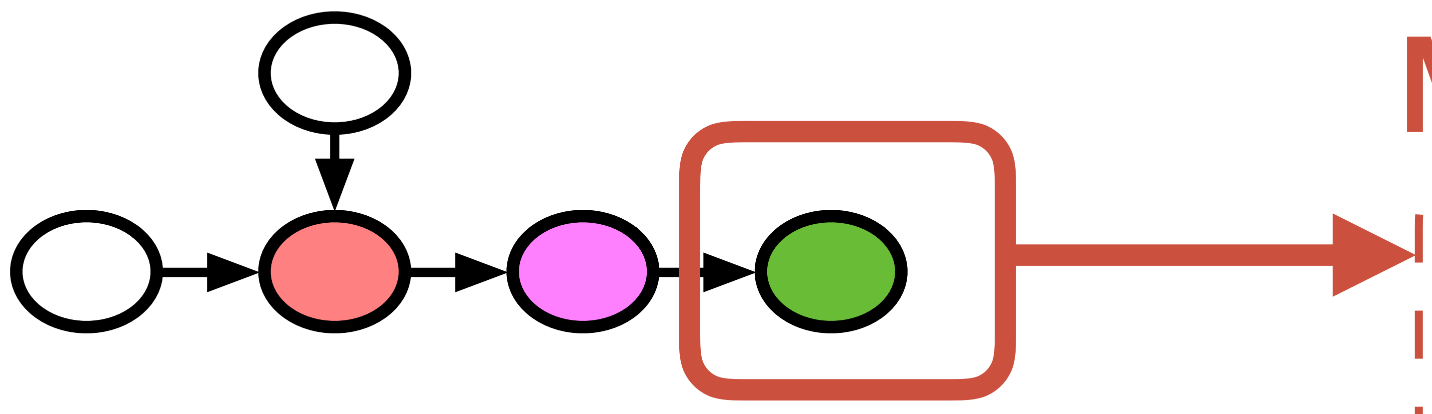
Existing Approach: Engineer Optimized Tensor Operators



Matmul: Operator Specification

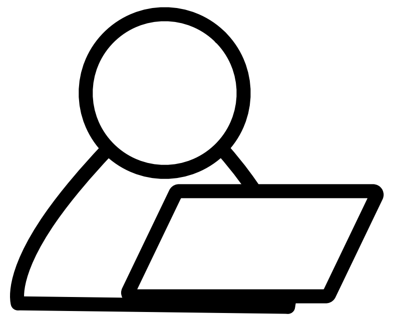
```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Existing Approach: Engineer Optimized Tensor Operators

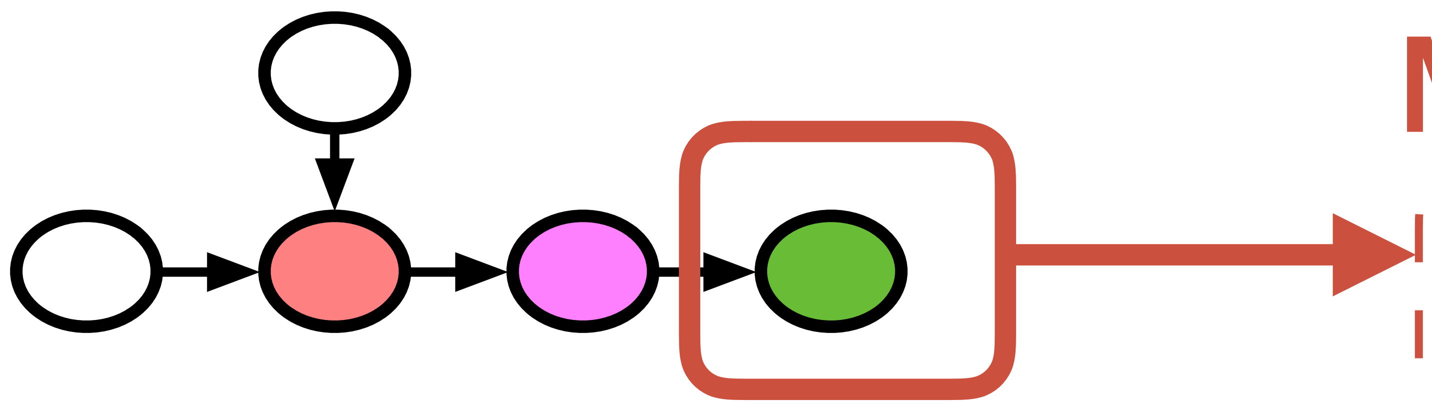


Matmul: Operator Specification

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

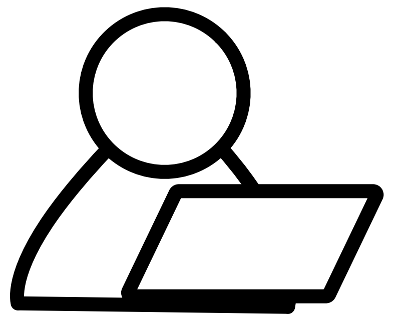


Existing Approach: Engineer Optimized Tensor Operators



Matmul: Operator Specification

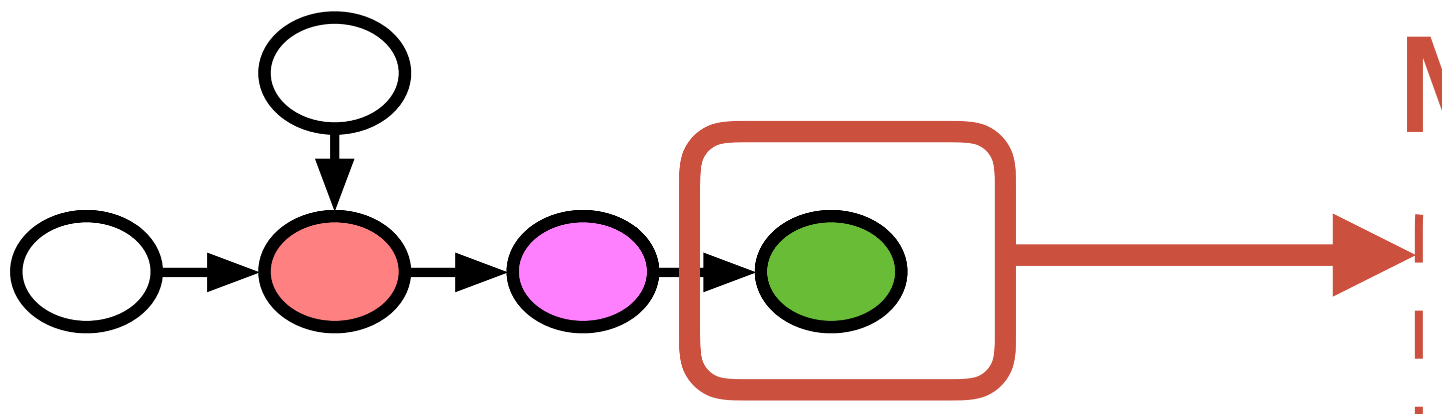
```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```



Vanilla Code

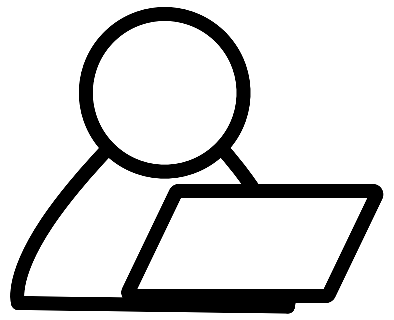
```
for y in range(1024):
  for x in range(1024):
    C[y][x] = 0
    for k in range(1024):
      C[y][x] += A[k][y] * B[k][x]
```

Existing Approach: Engineer Optimized Tensor Operators



Matmul: Operator Specification

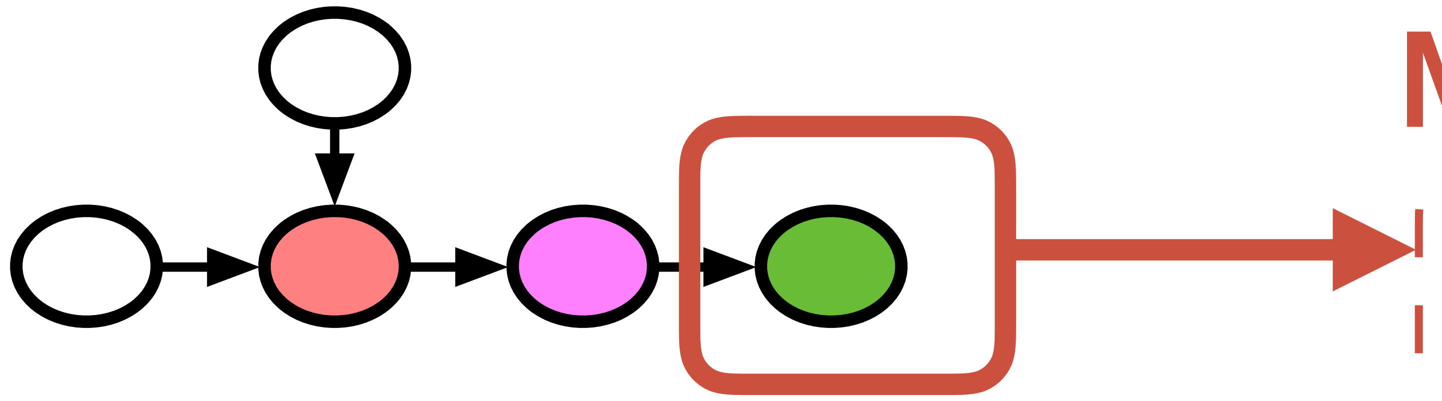
```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```



Loop Tiling for Locality

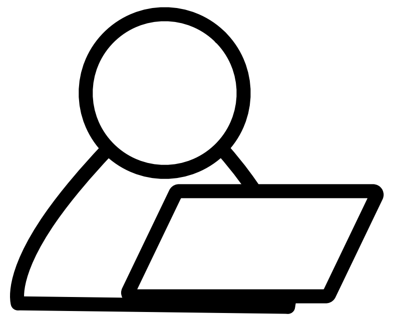
```
for yo in range(128):
  for xo in range(128):
    C[yo*8:yo*8+8][xo*8:xo*8+8] = 0
    for ko in range(128):
      for yi in range(8):
        for xi in range(8):
          for ki in range(8):
            C[yo*8+yi][xo*8+xi] +=
               A[ko*8+ki][yo*8+yi] * B[ko*8+ki][xo*8+xi]
```

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))
```



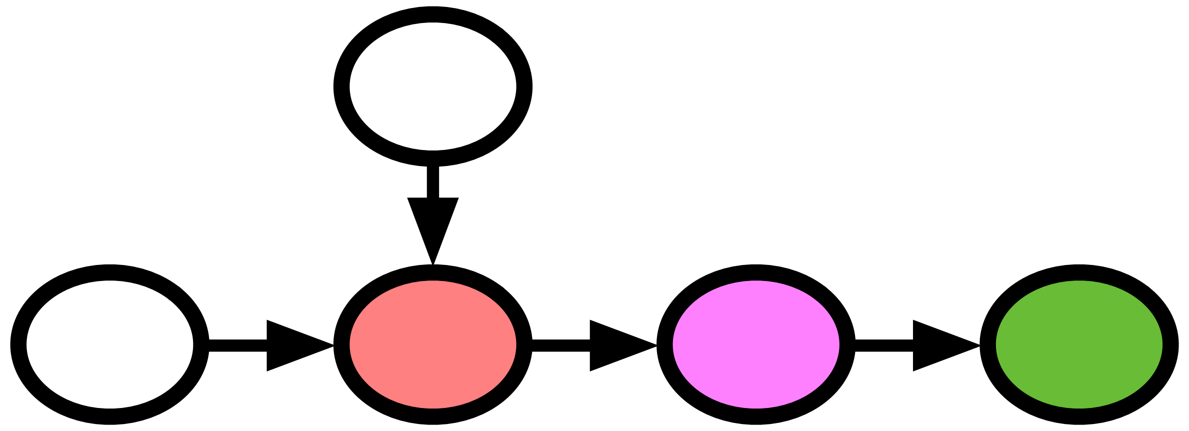
Map to Accelerators

```
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

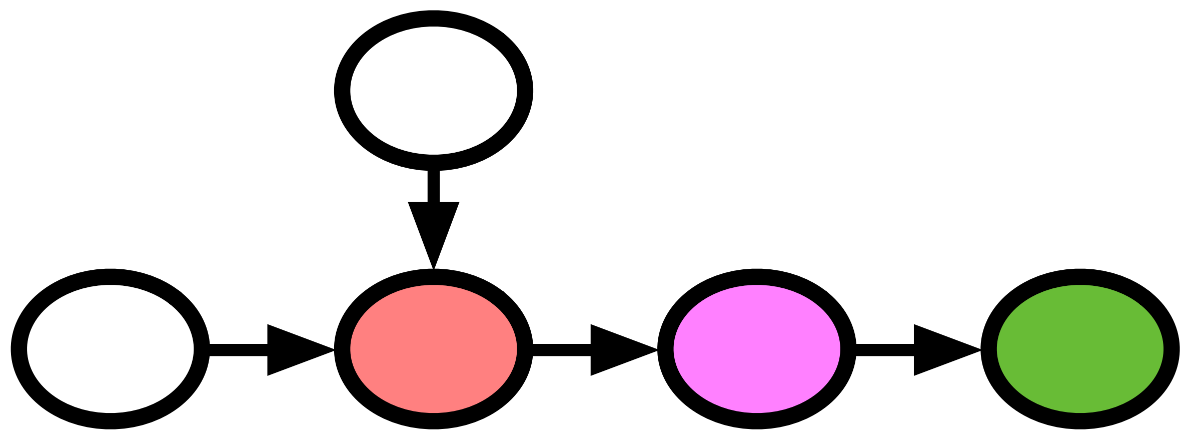
Frameworks



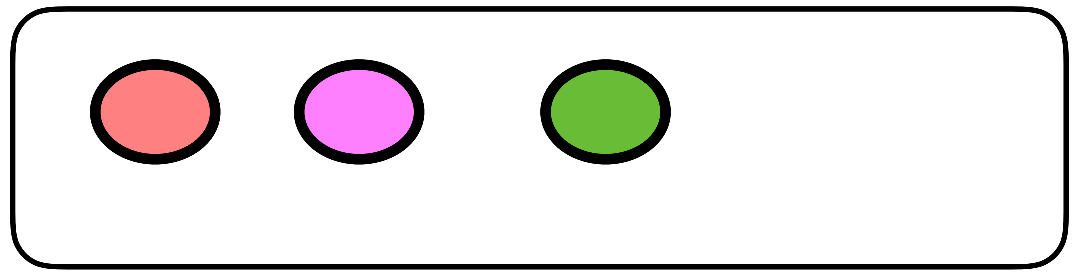
cuDNN

Limitations of Existing Approach

Frameworks

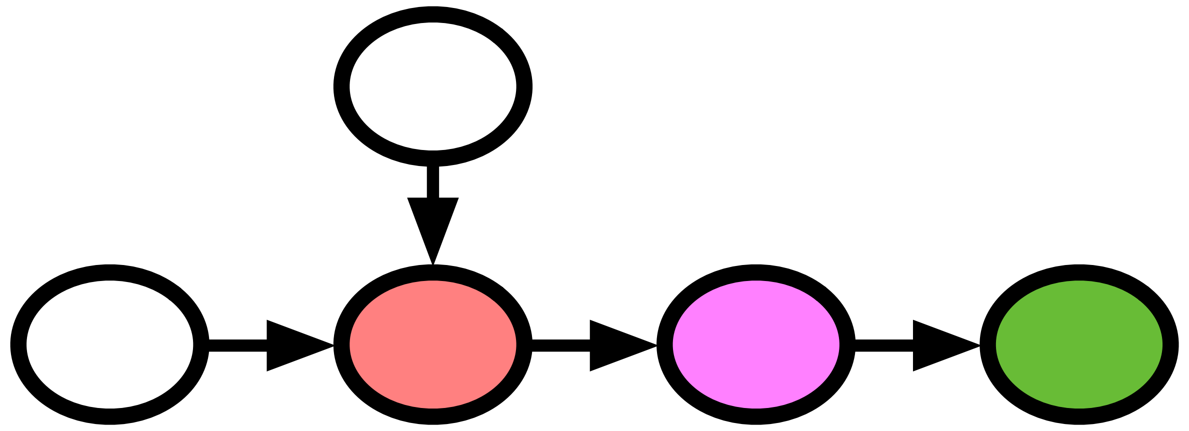


cuDNN

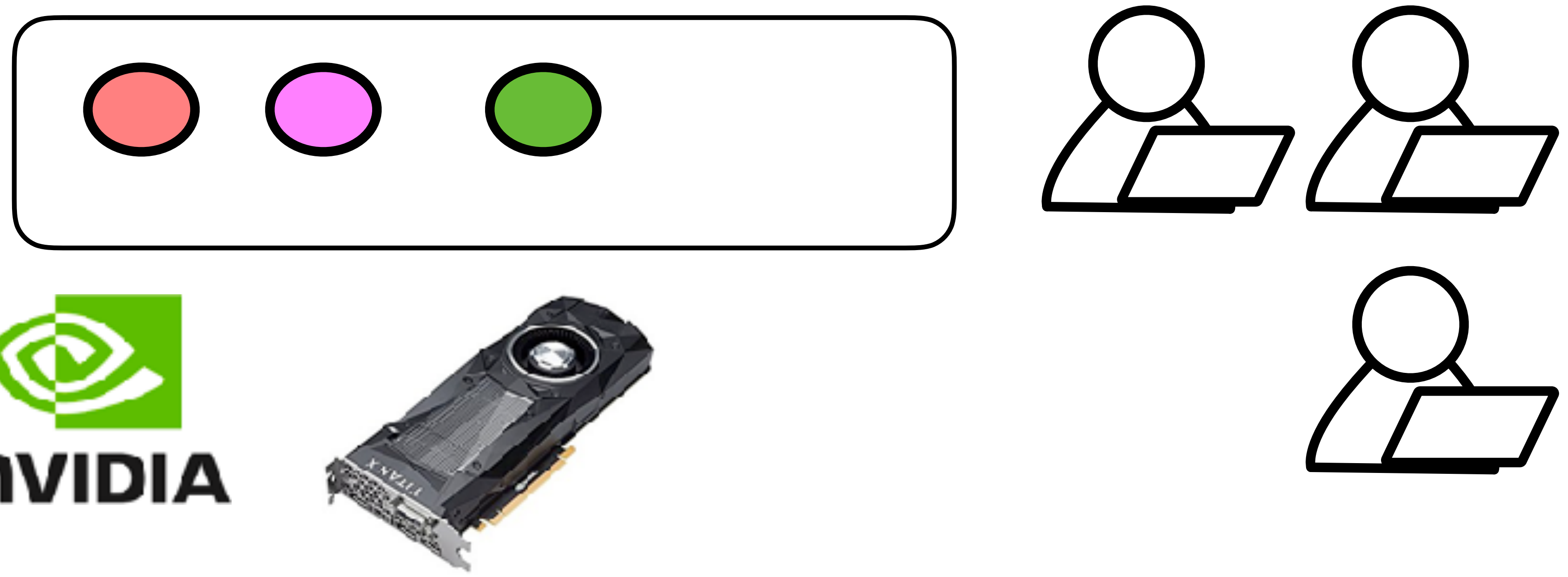


Limitations of Existing Approach

Frameworks

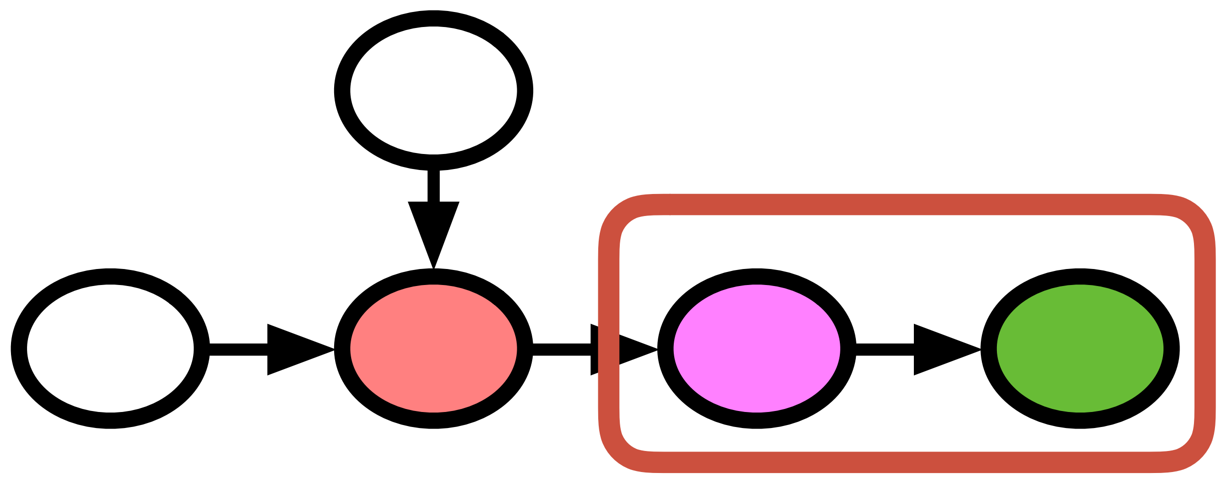


cuDNN

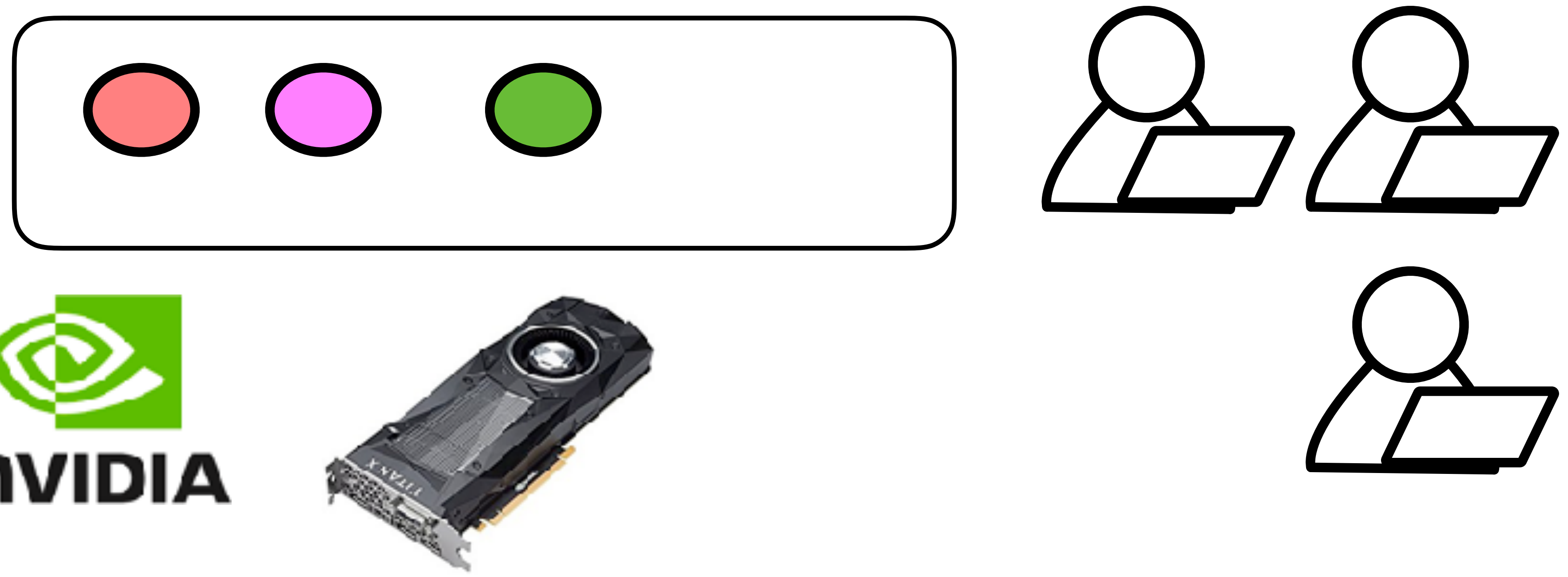


Limitations of Existing Approach

Frameworks



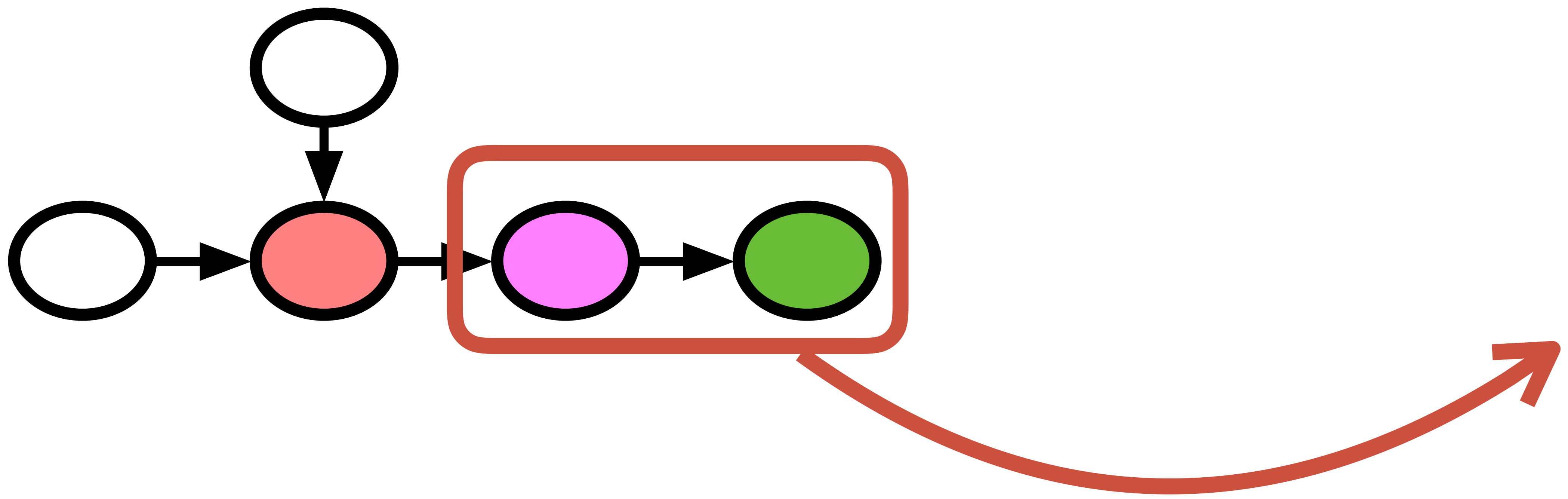
cuDNN



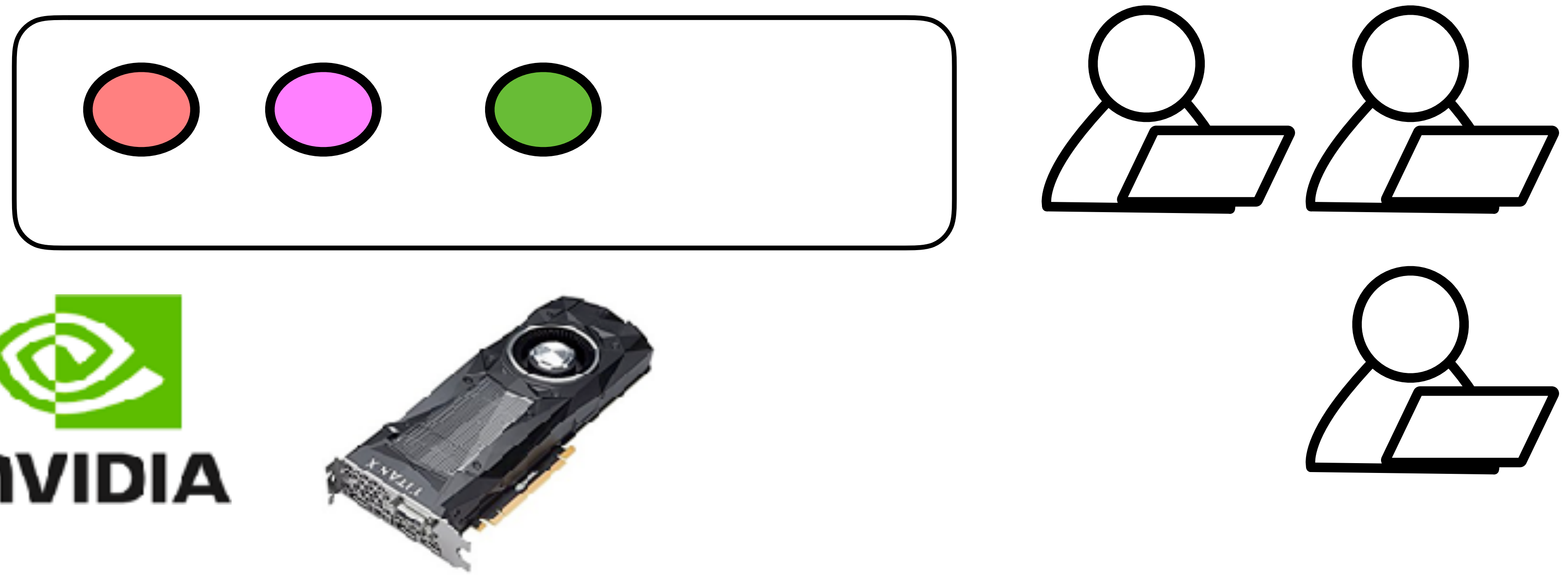
NVIDIA

Limitations of Existing Approach

Frameworks



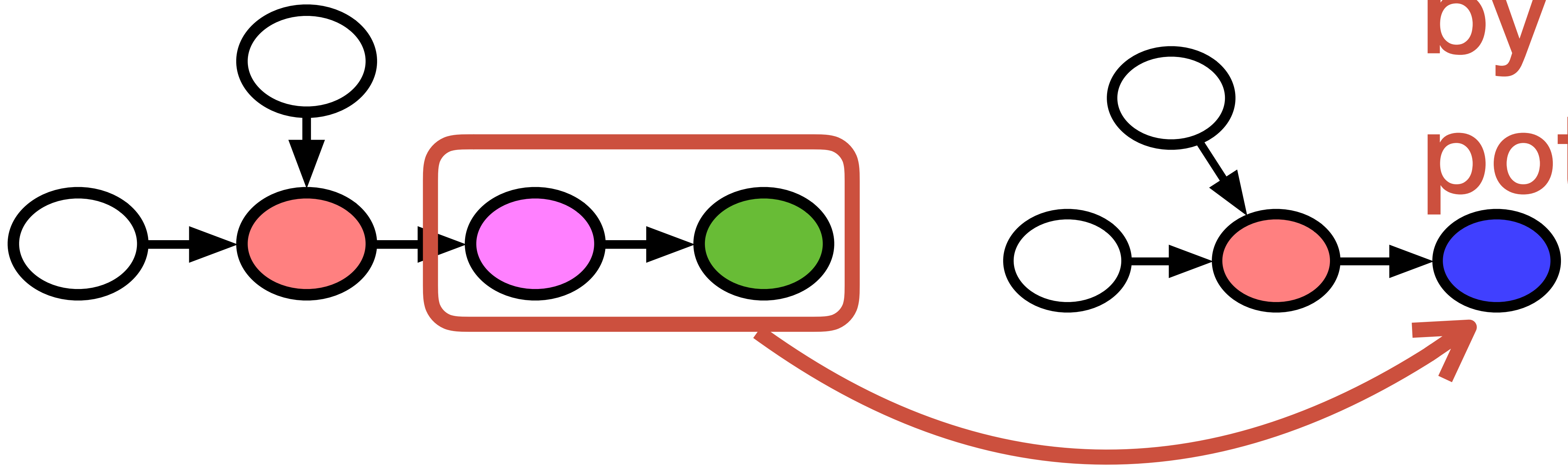
cuDNN



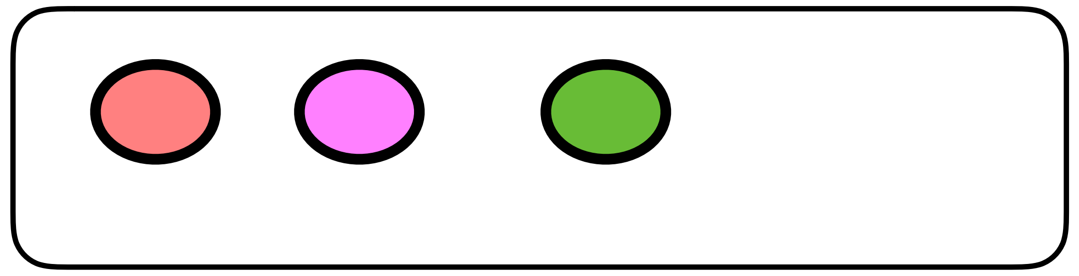
Limitations of Existing Approach

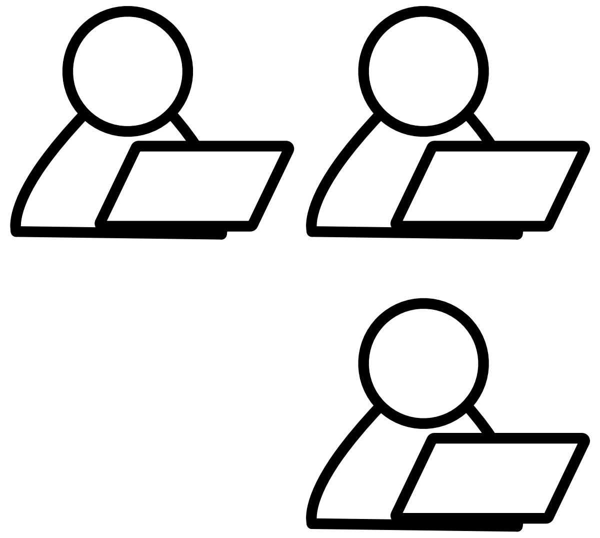
Frameworks

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



cuDNN

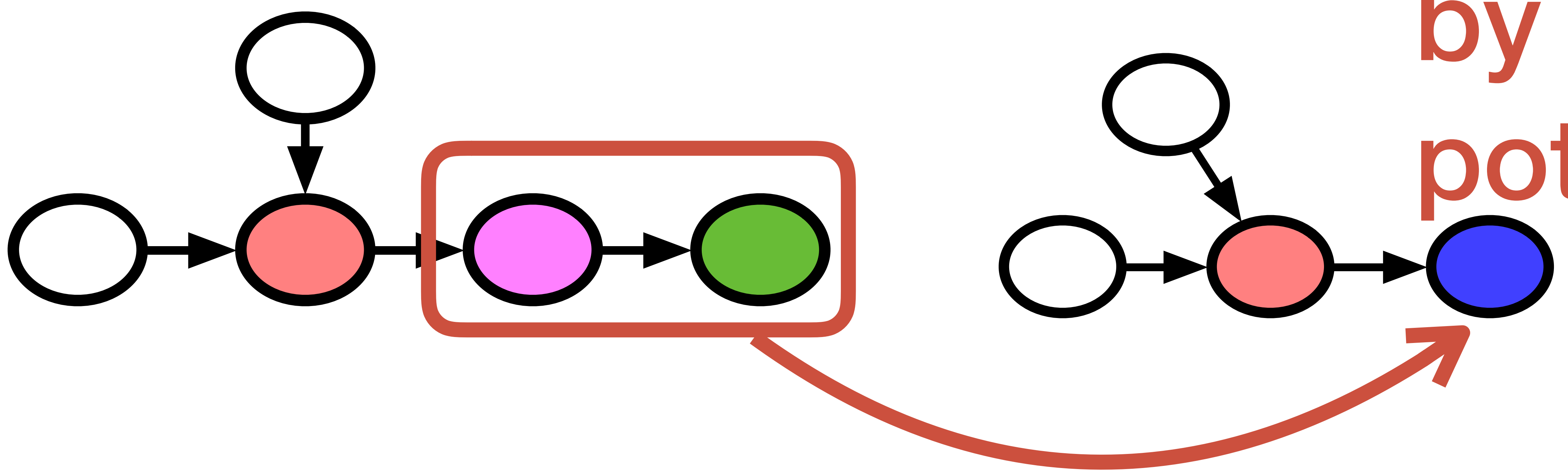




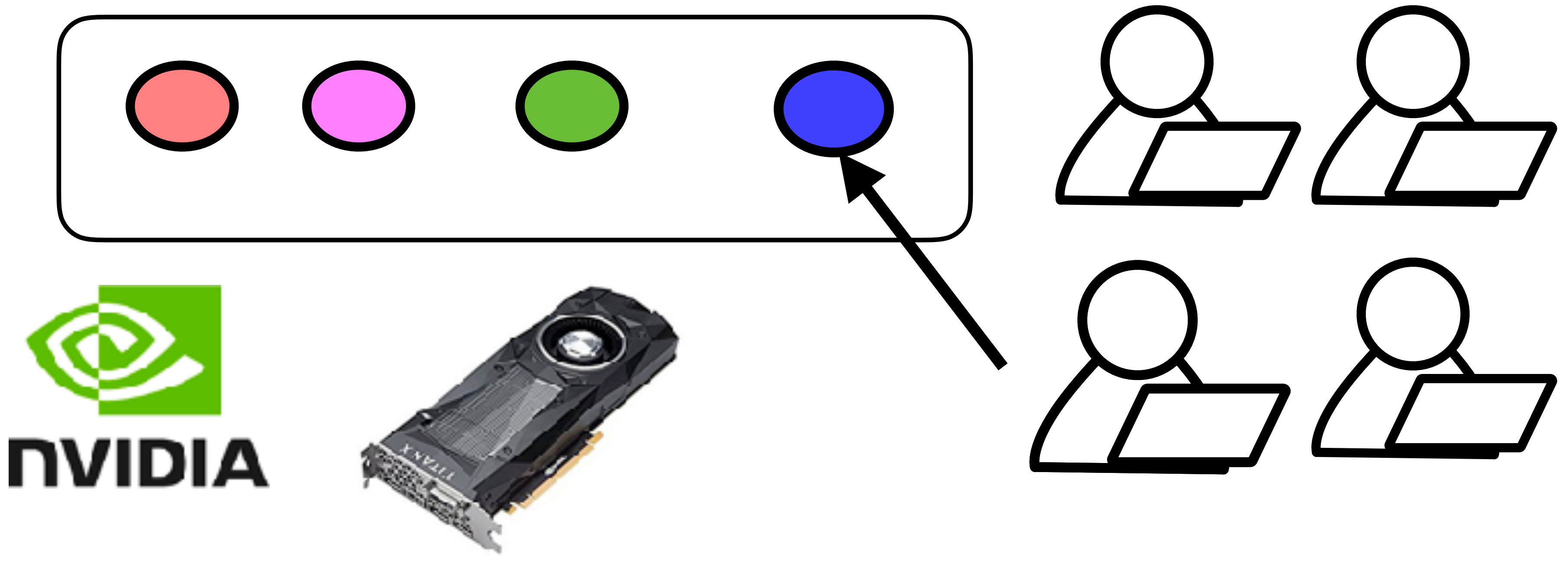
Limitations of Existing Approach

Frameworks

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



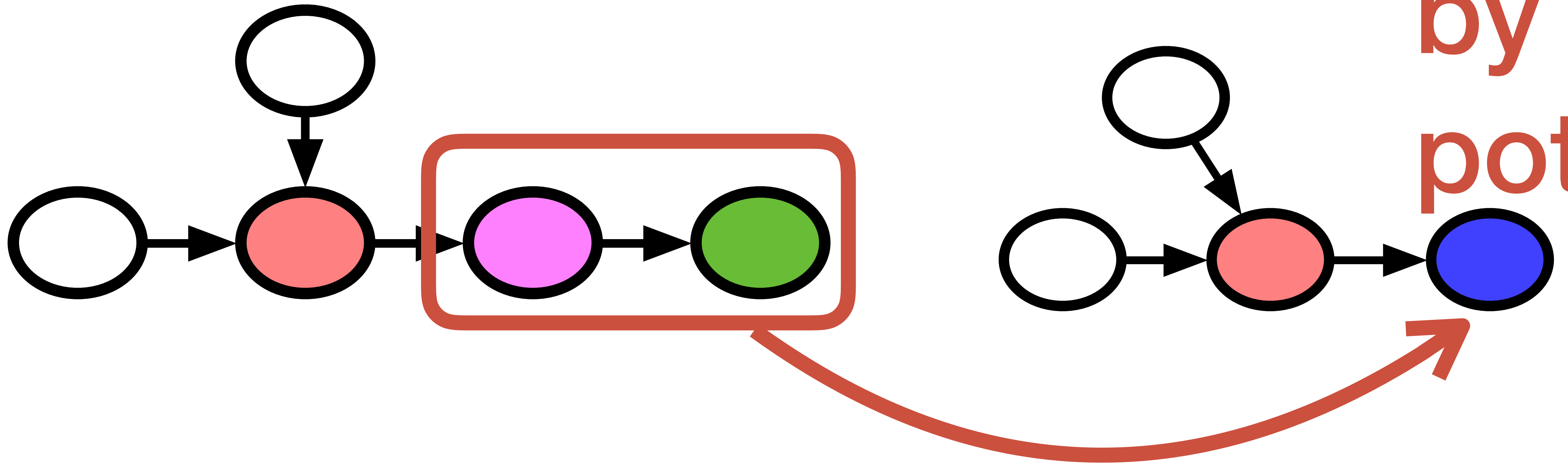
cuDNN



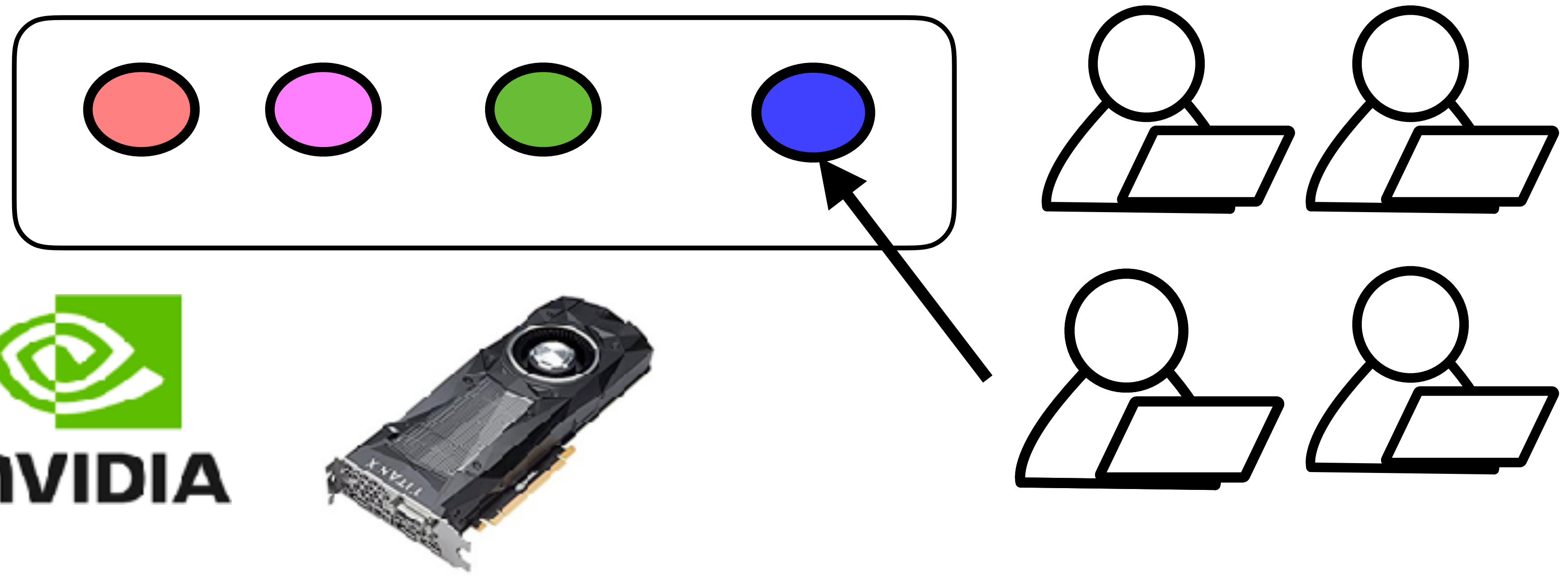
Limitations of Existing Approach

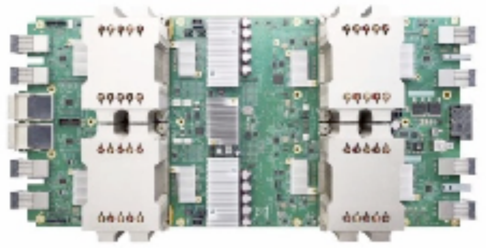
Frameworks

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



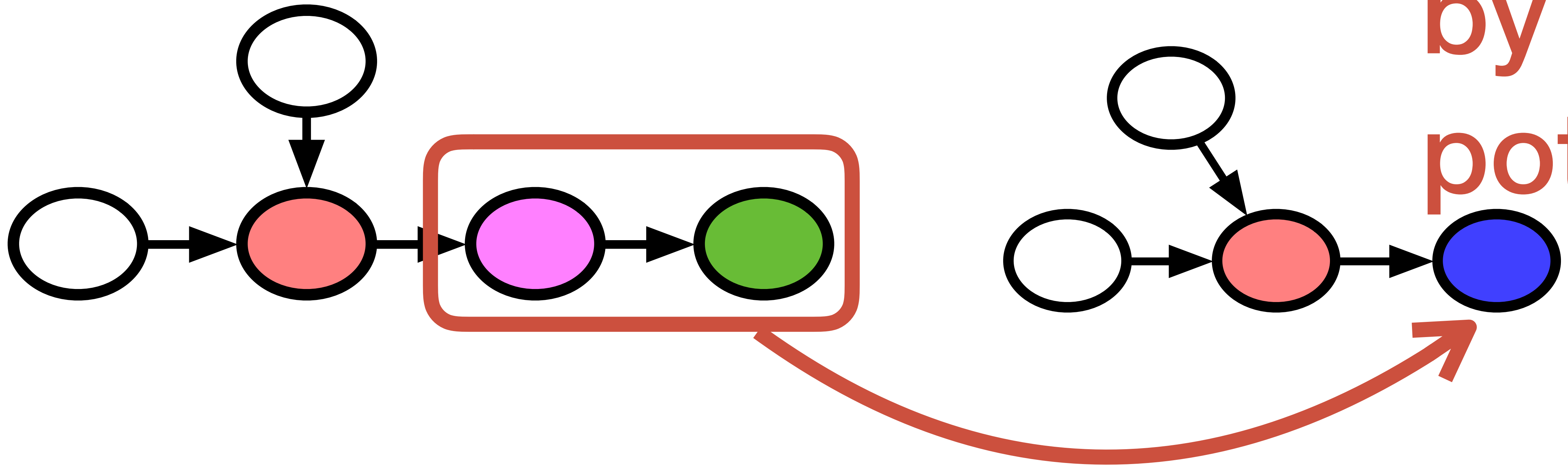
cuDNN





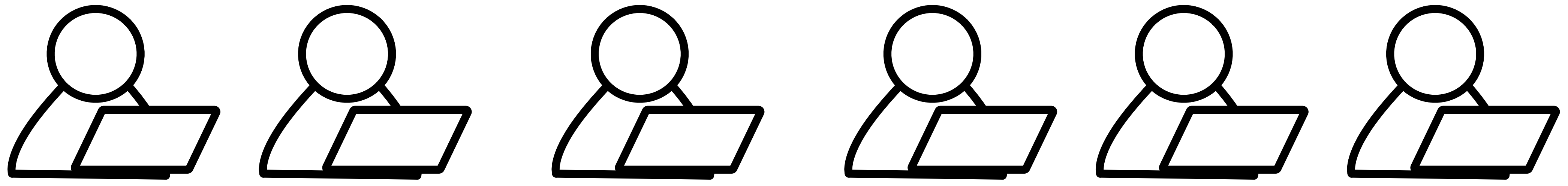
Limitations of Existing Approach

Frameworks

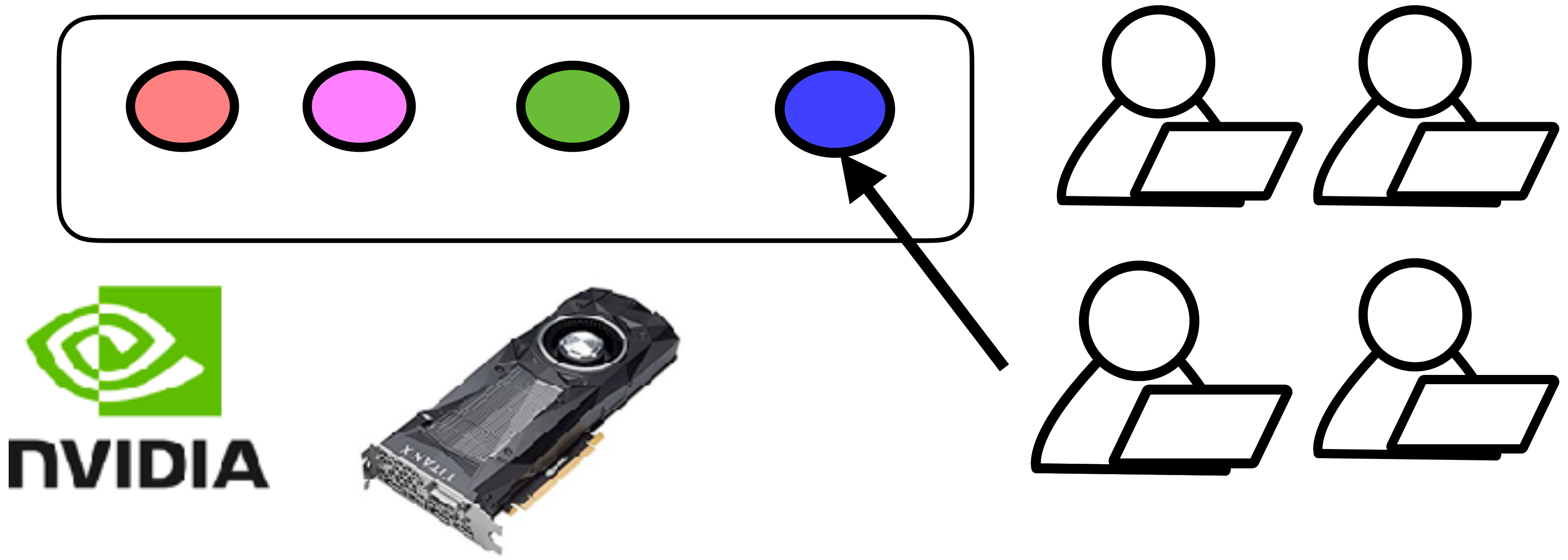


New operator introduced
by operator fusion optimization
potentially benefit: 1.5x speedup

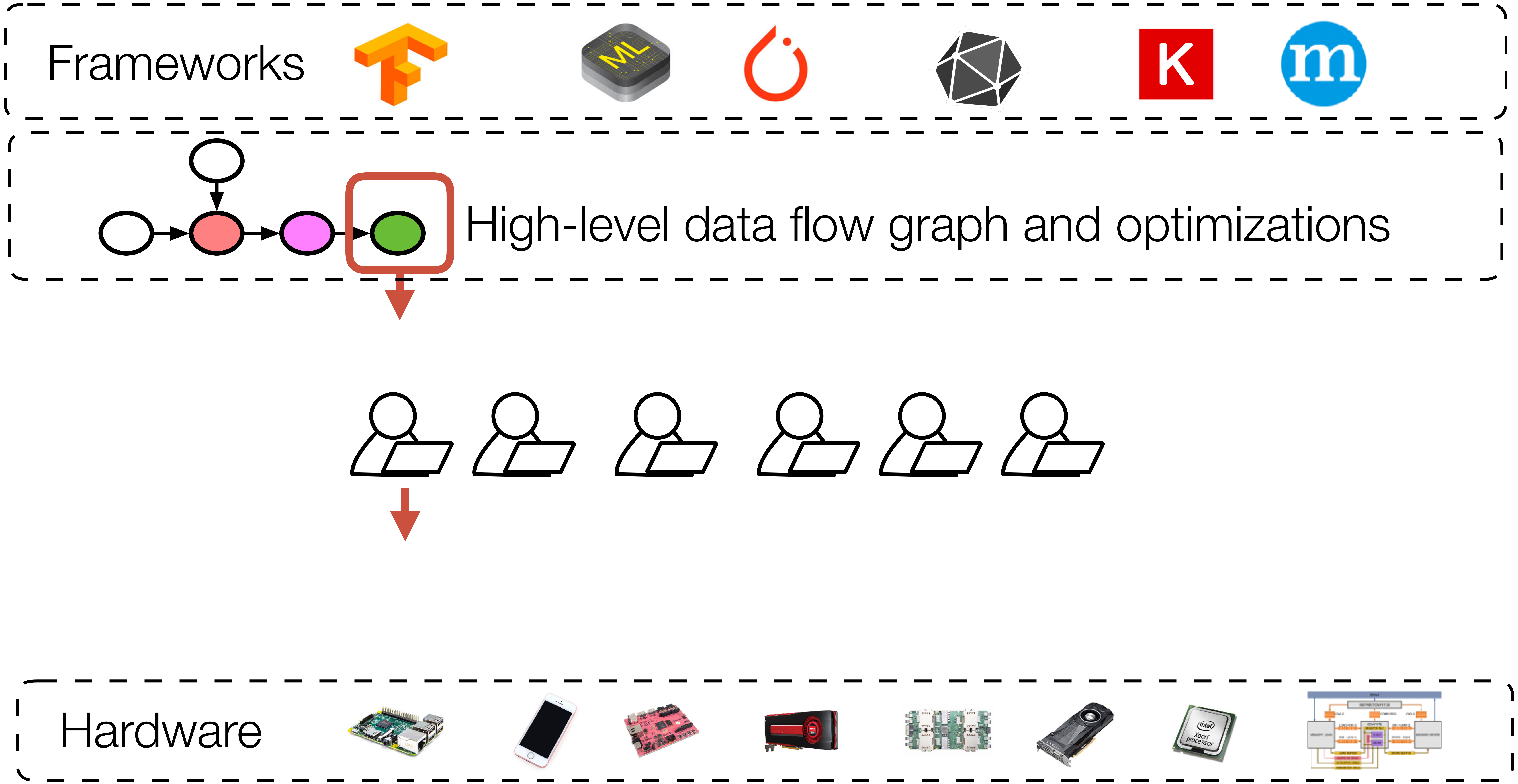
Engineering intensive



cuDNN

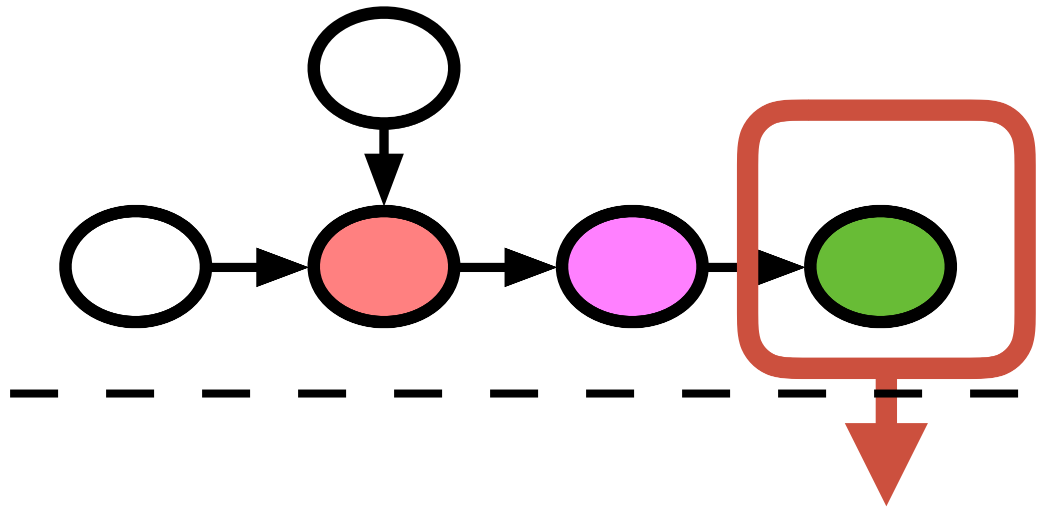


Learning-based Learning System



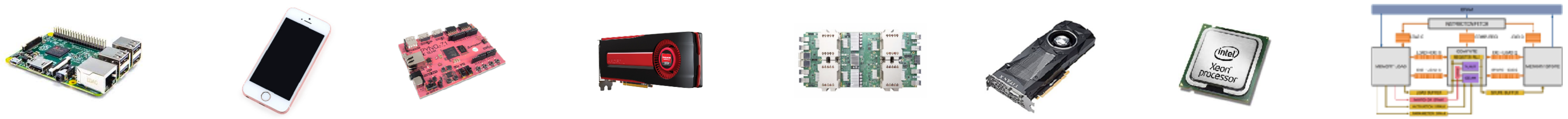
Learning-based Learning System

Frameworks

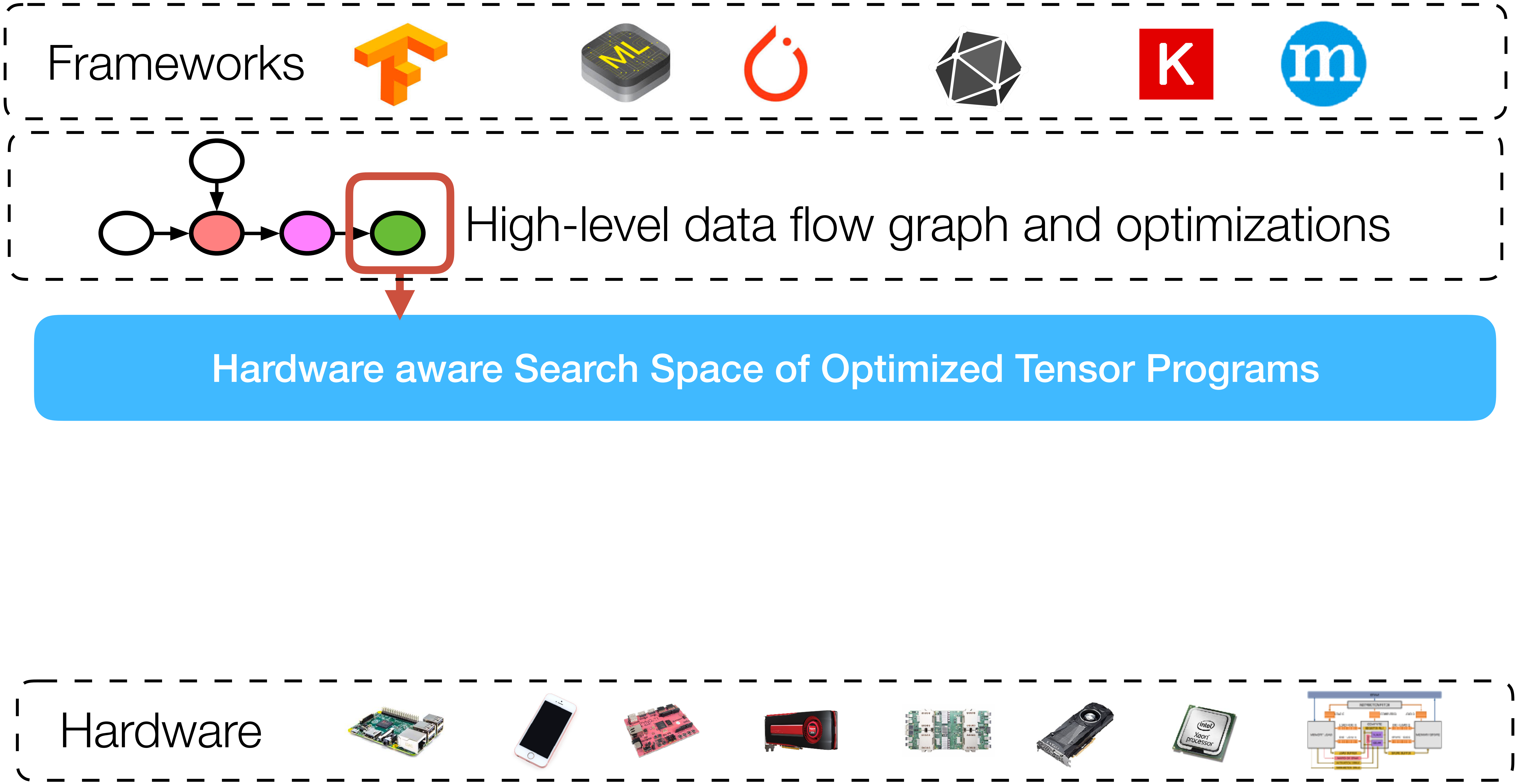


High-level data flow graph and optimizations

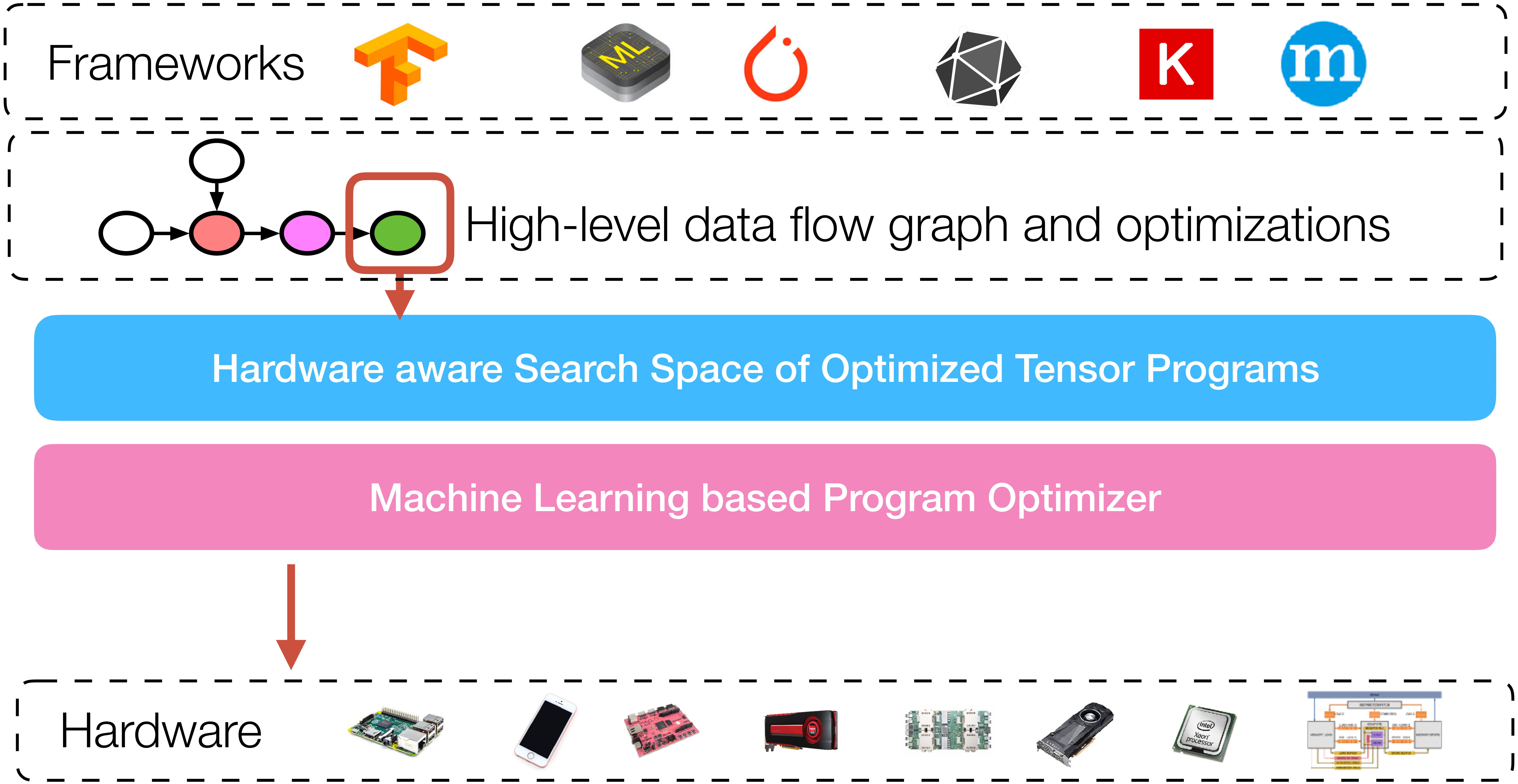
Hardware



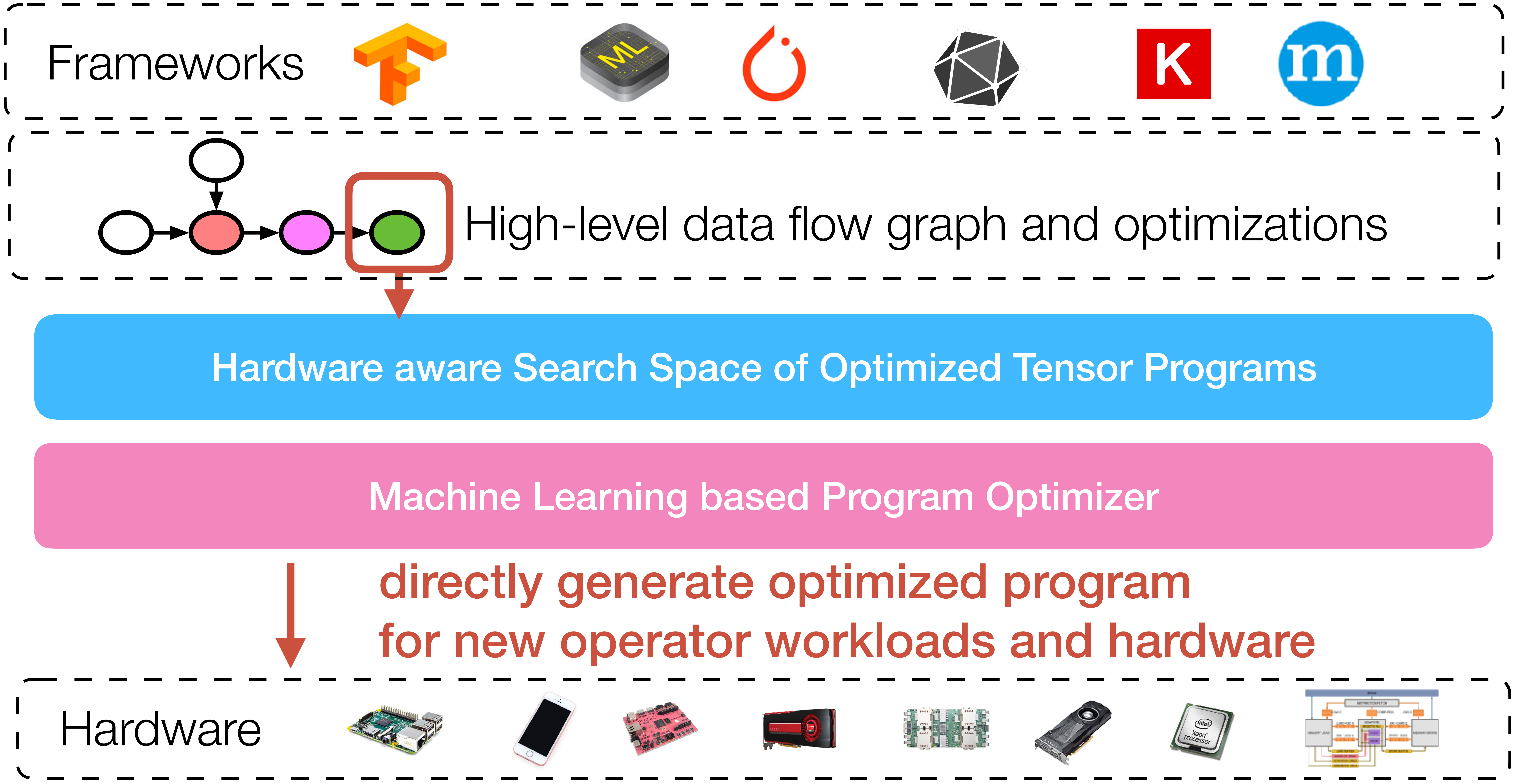
Learning-based Learning System



Learning-based Learning System

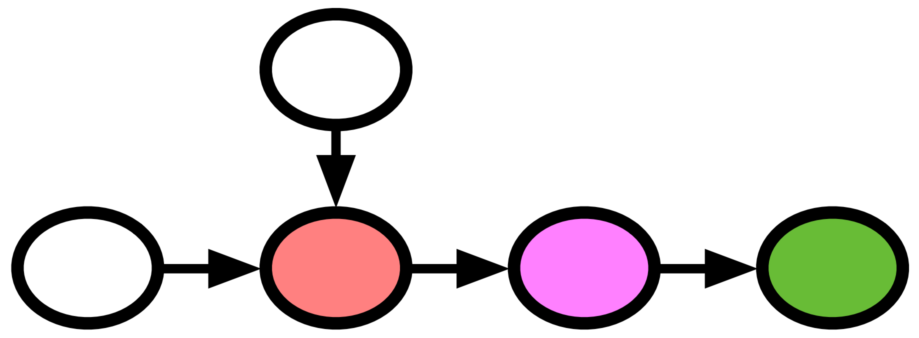


Learning-based Learning System



Learning-based Learning System

Frameworks



High-level data flow graph and optimizations

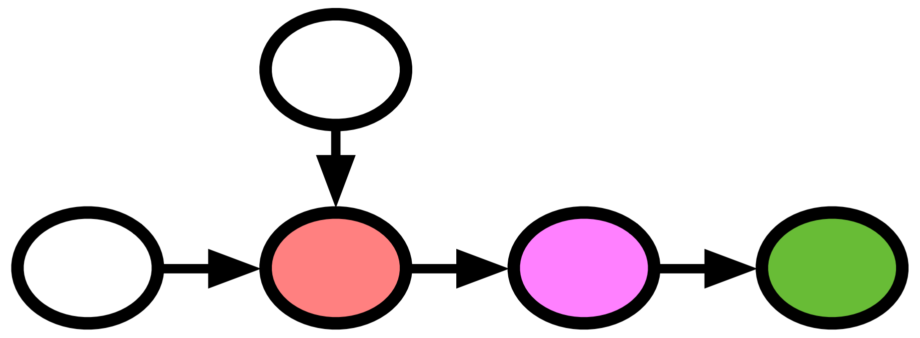
Hardware aware Search Space of Optimized Tensor Programs

Machine Learning based Program Optimizer

Hardware

Learning-based Learning System

Frameworks

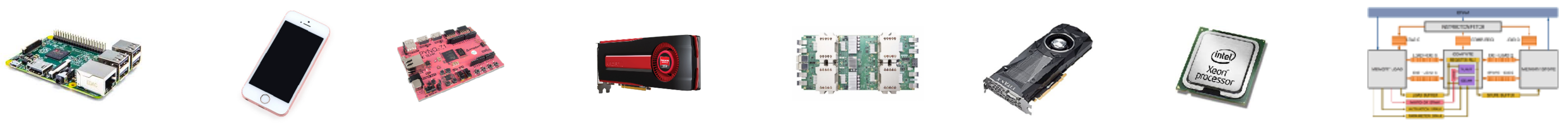


High-level data flow graph and optimizations

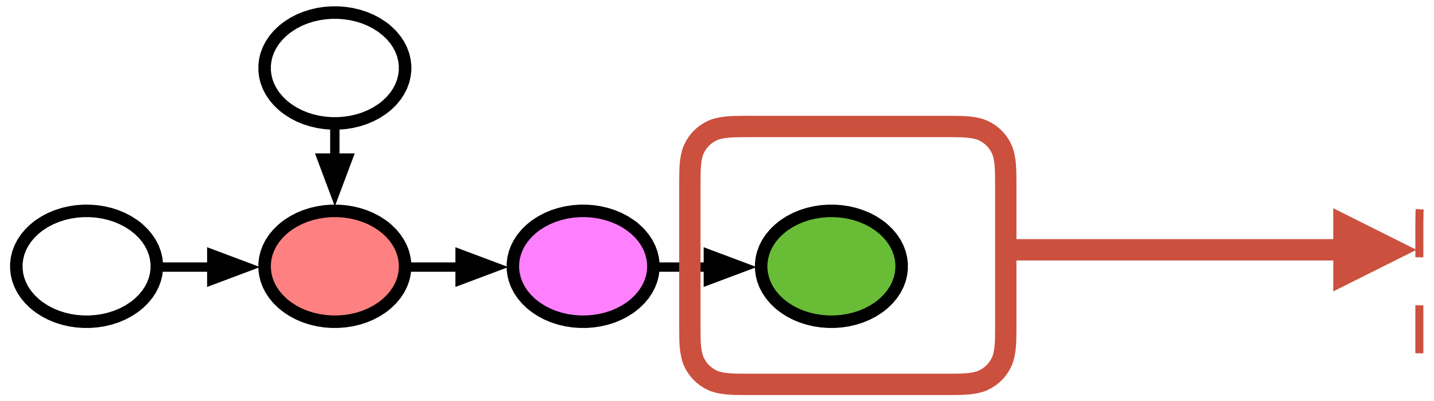
Hardware aware Search Space of Optimized Tensor Programs

Machine Learning based Program Optimizer

Hardware



Hardware-aware Search Space

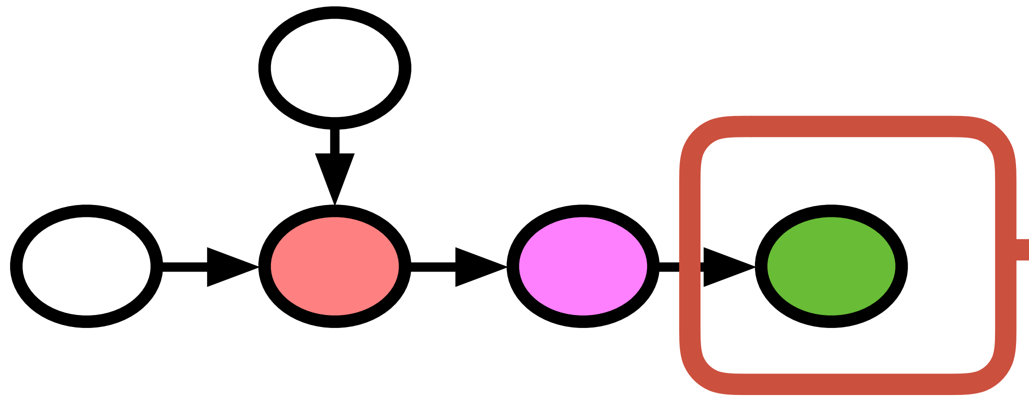


Tensor Expression Language (Specification)

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

Hardware

Hardware-aware Search Space



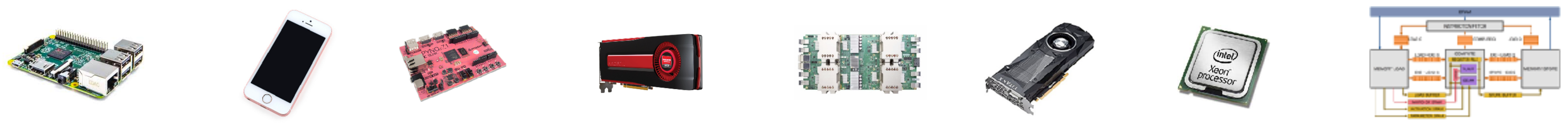
Tensor Expression Language (Specification)

```
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

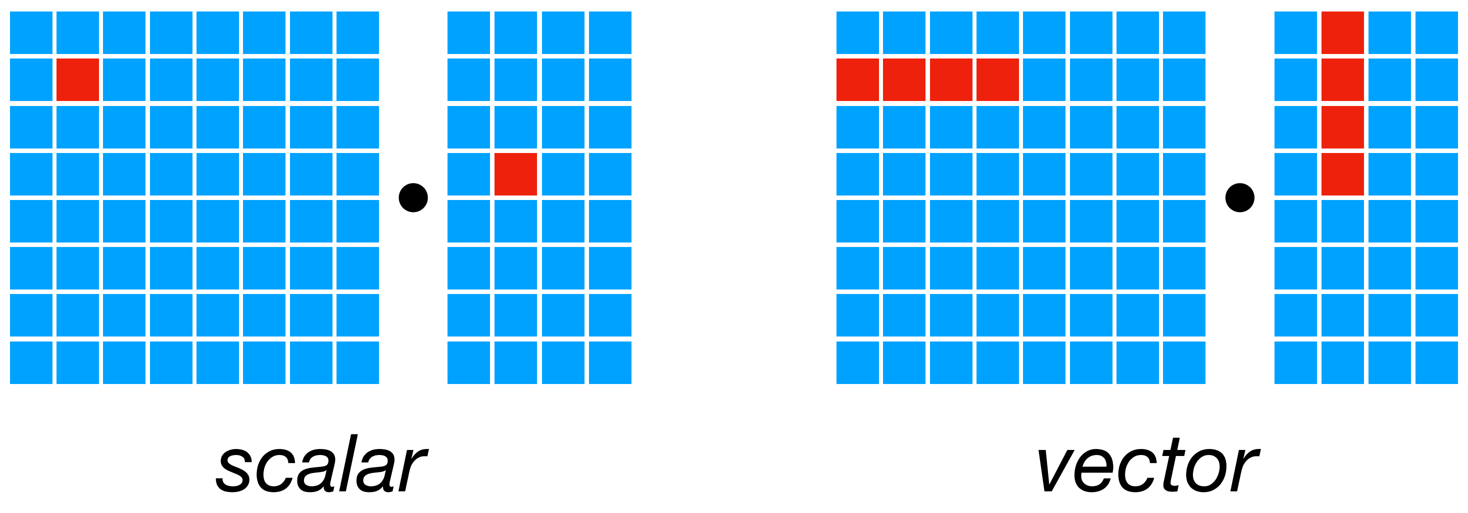
Hardware



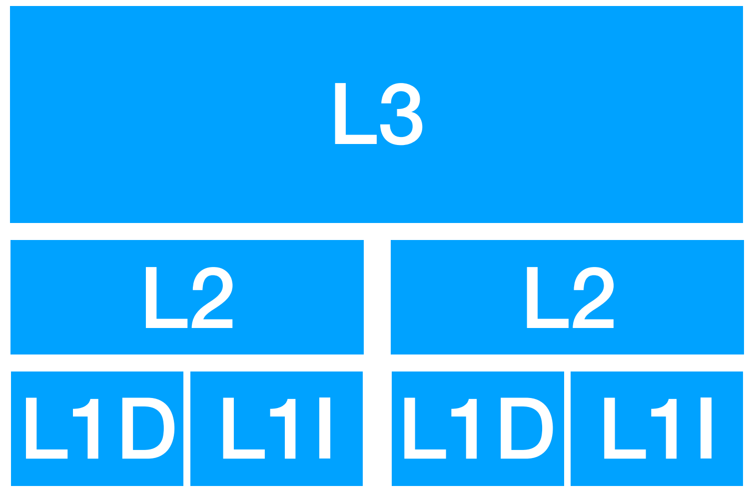
Hardware-aware Search Space

CPUs

Compute Primitives



Memory Subsystem



implicitly managed

Loop Transformations

Cache Locality

Vectorization

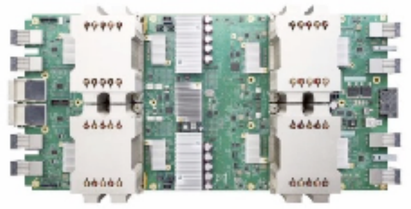
Reuse primitives from prior work:
Halide, Loopy

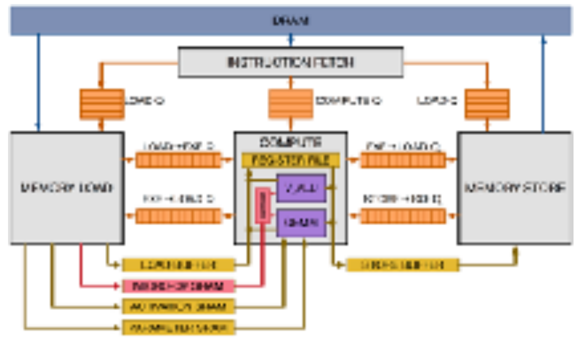
Challenge to Support Diverse Hardware Backends

CPUs

GPUs

TPU-like specialized Accelerators

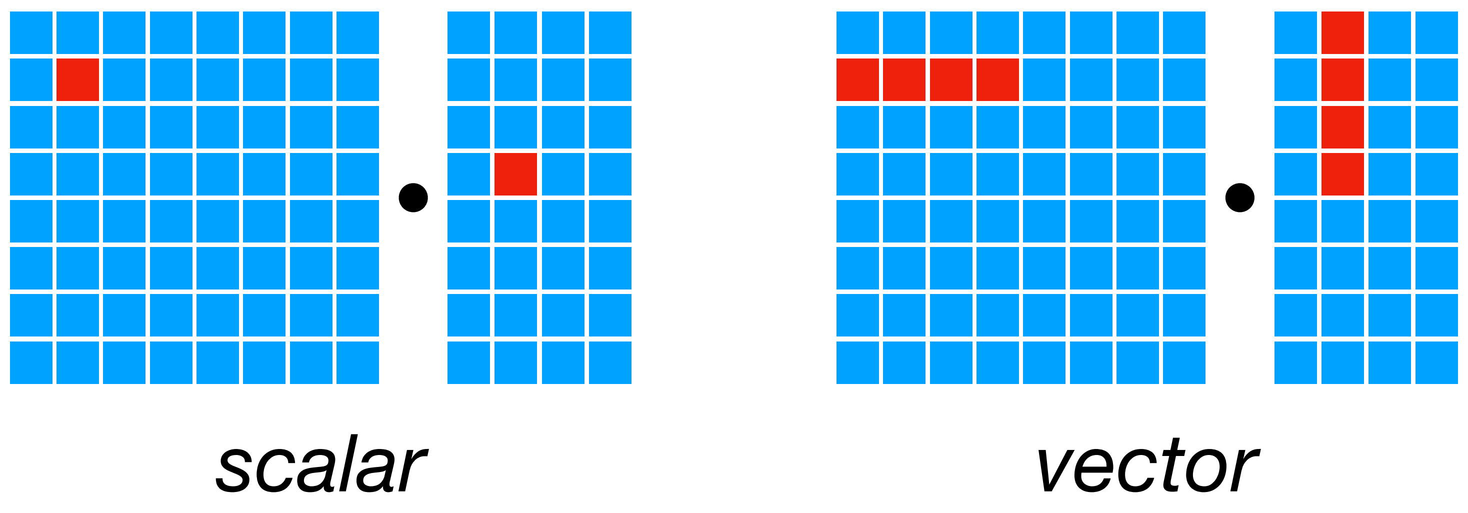




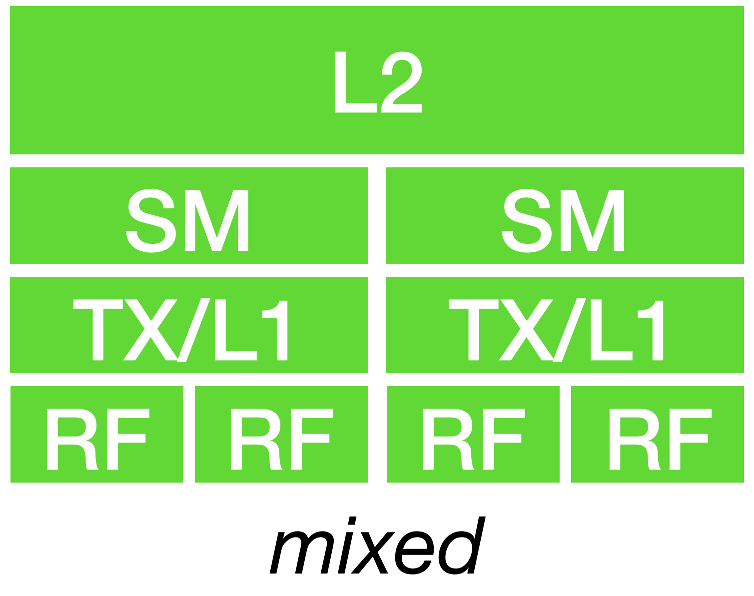
Hardware-aware Search Space

GPUs

Compute Primitives



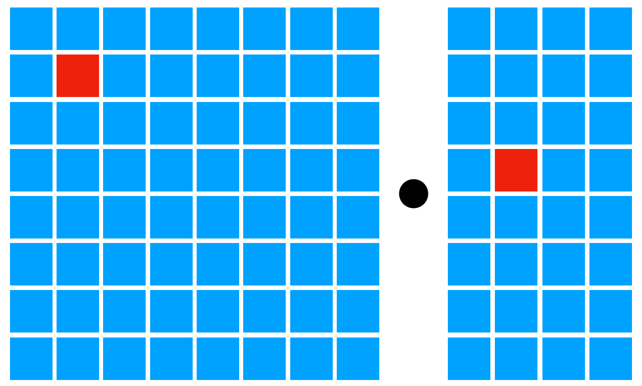
Memory Subsystem



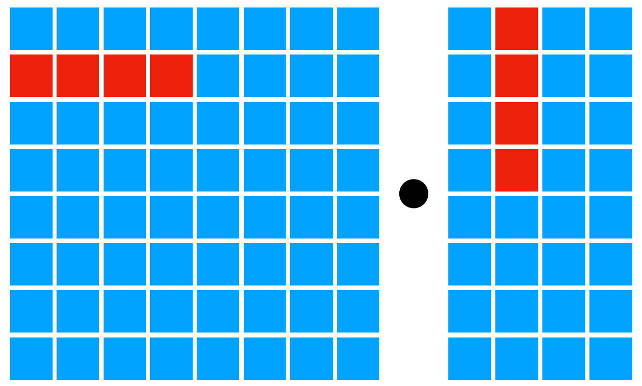
Hardware-aware Search Space

GPUs

Compute Primitives

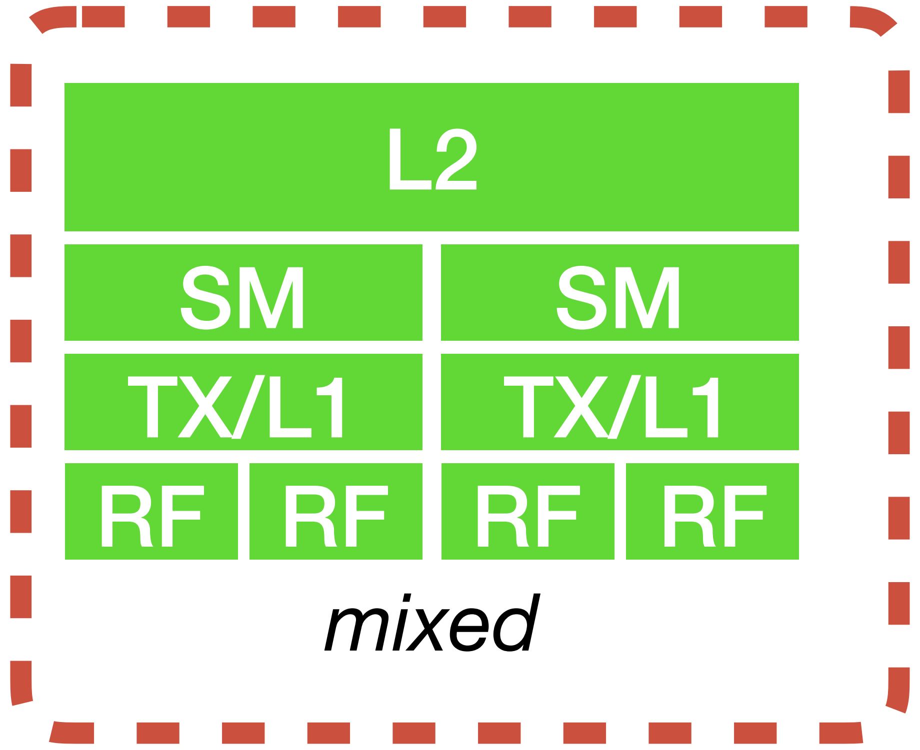


scalar



vector

Memory Subsystem



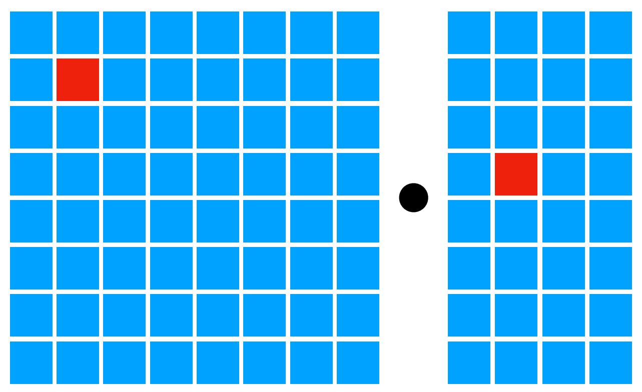
mixed

Shared memory among compute cores

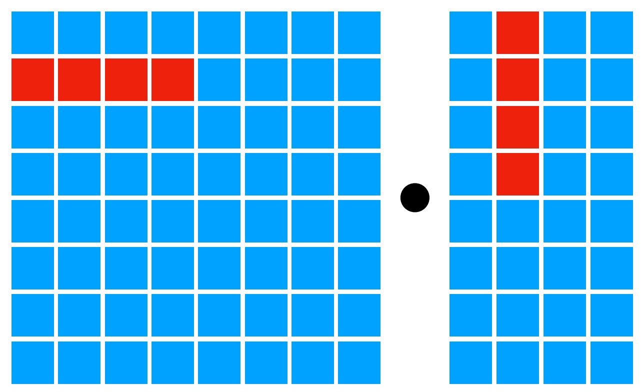
Hardware-aware Search Space

GPUs

Compute Primitives

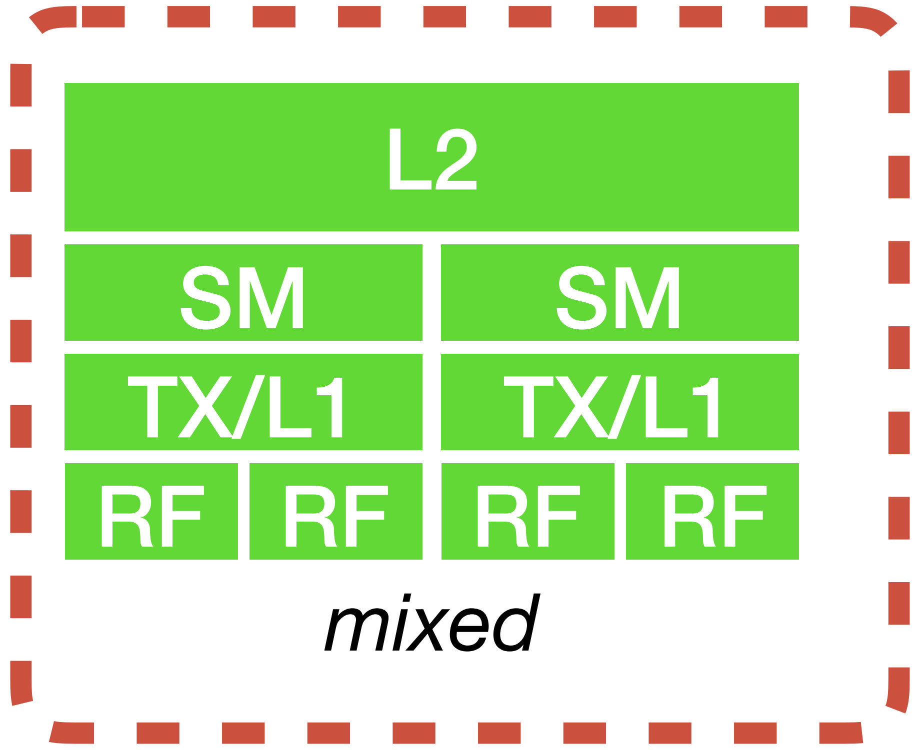


scalar



vector

Memory Subsystem



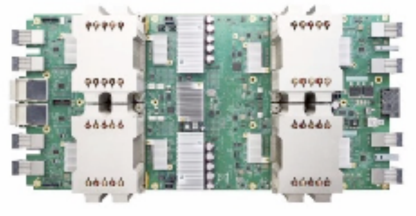
Shared memory among compute cores

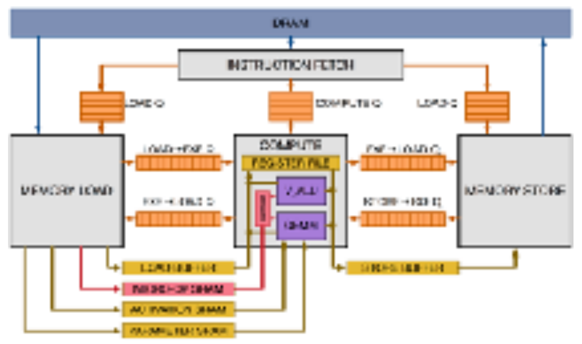
Use of Shared Memory

Thread Cooperation

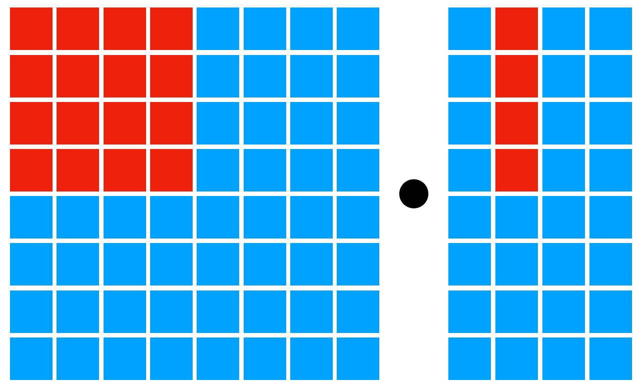
Hardware-aware Search Space

TPU-like Specialized Accelerators



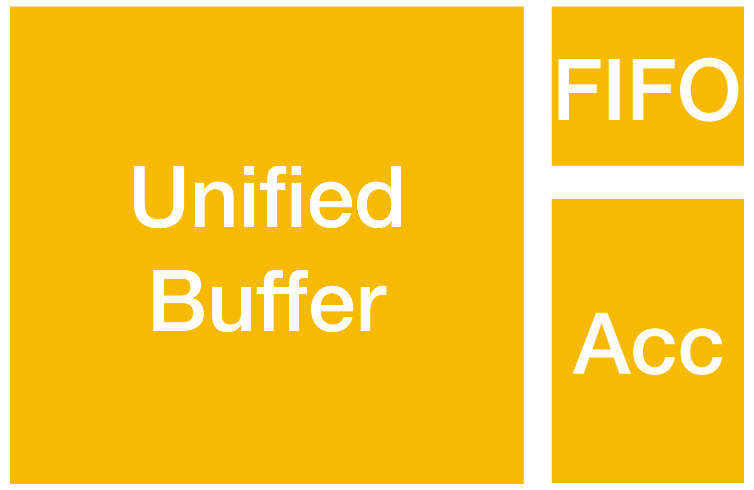


Compute Primitives



tensor

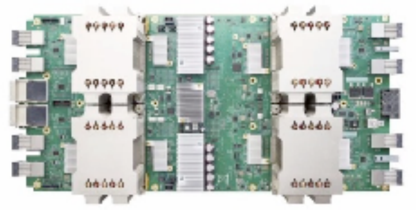
Memory Subsystem

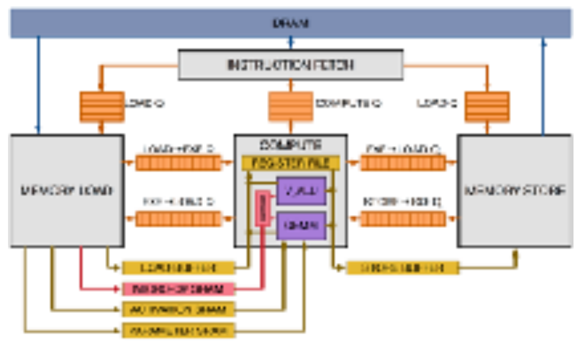


explicitly managed

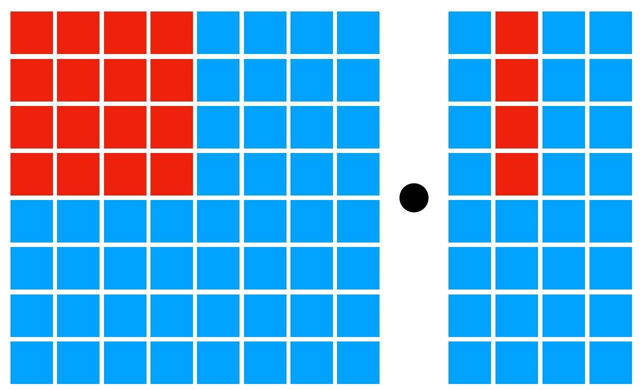
Hardware-aware Search Space

TPU-like Specialized Accelerators



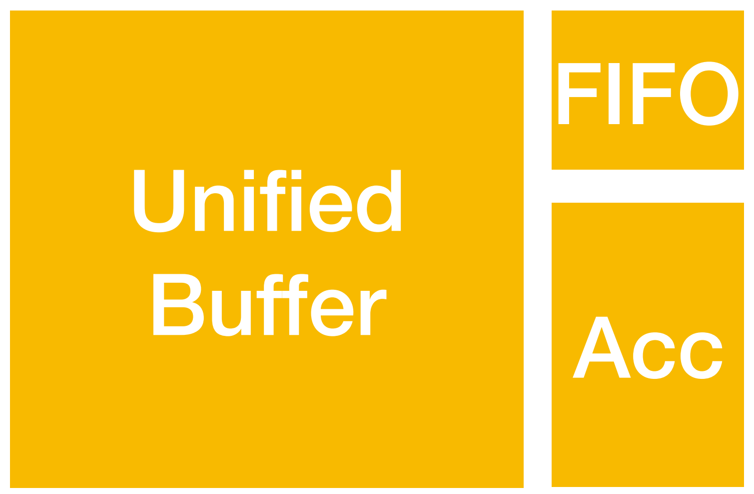


Compute Primitives



tensor

Memory Subsystem



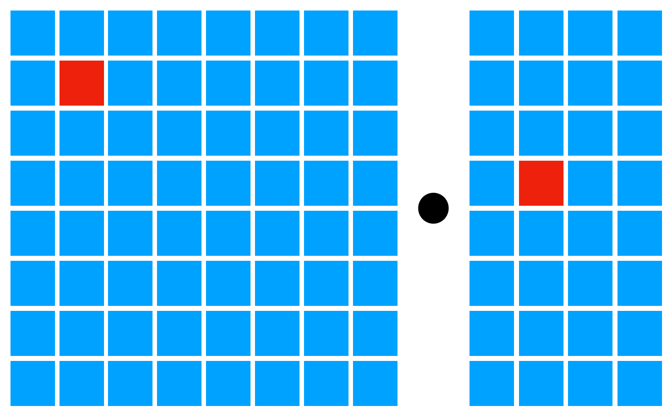
explicitly managed

Tensorization Challenge

Compute primitives

Tensorization Challenge

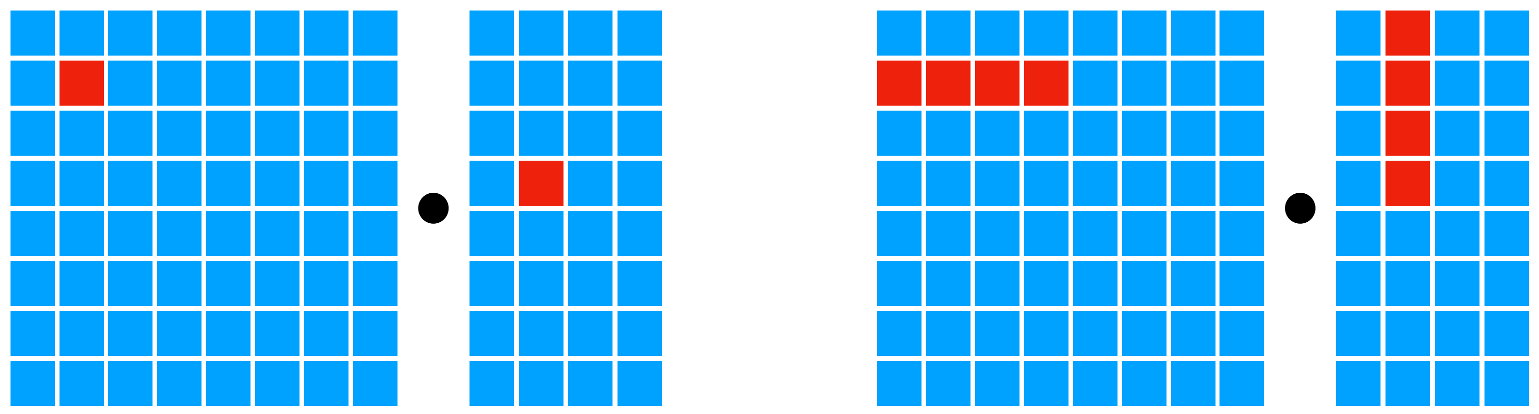
Compute primitives



scalar

Tensorization Challenge

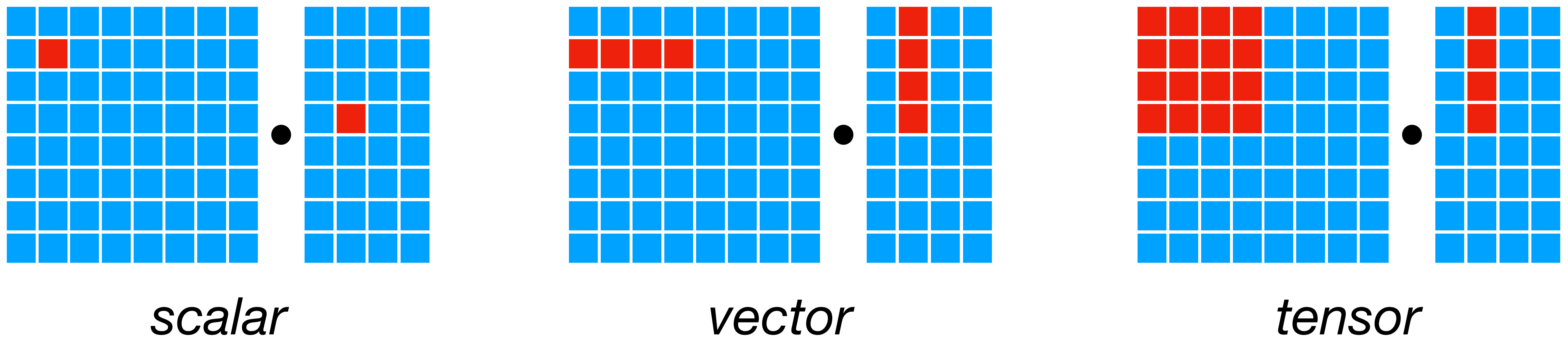
Compute primitives



scalar *vector*

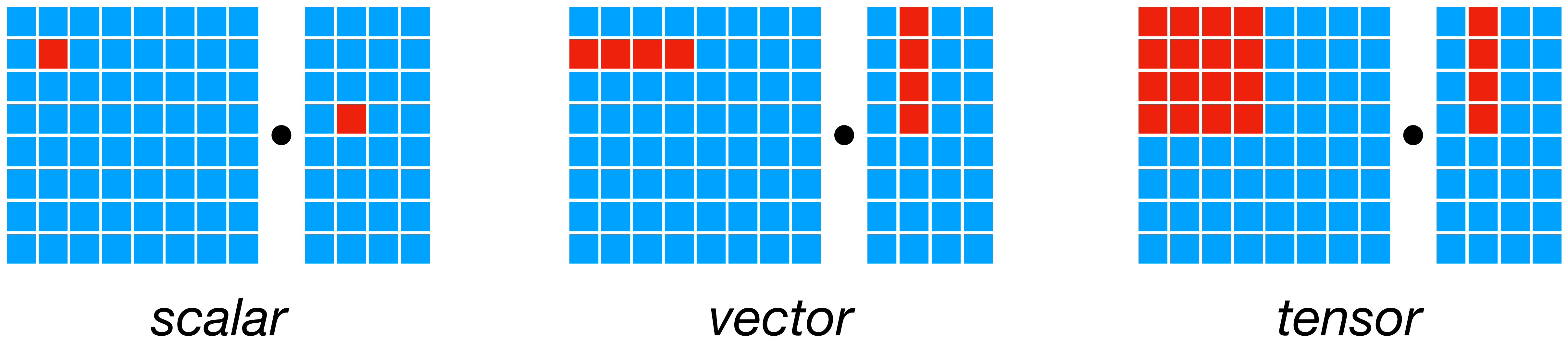
Tensorization Challenge

Compute primitives



Tensorization Challenge

Compute primitives



Hardware designer: declare tensor instruction interface with Tensor Expression

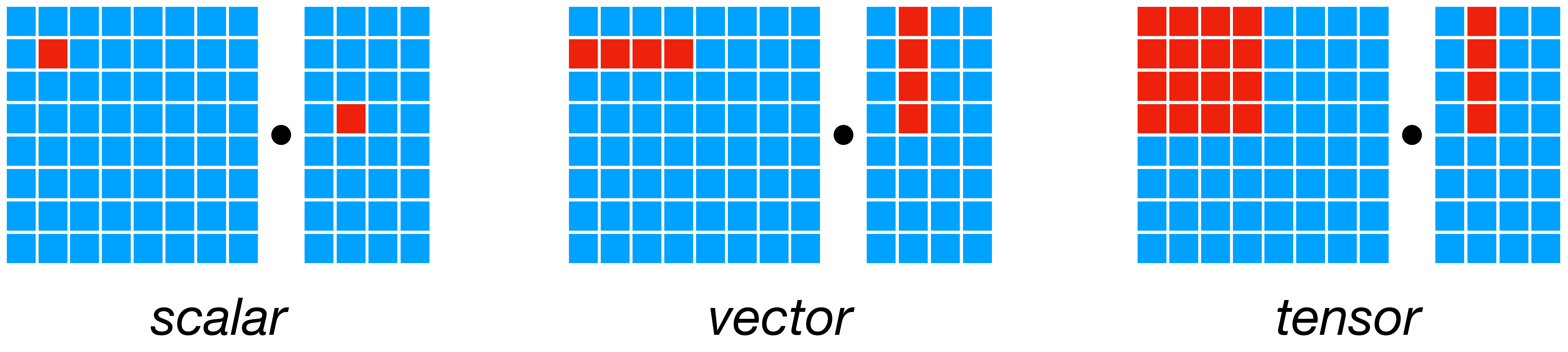
```
w, x = t.placeholder((8, 8)), t.placeholder((8, 8))
k = t.reduce_axis((0, 8))
y = t.compute((8, 8), lambda i, j:
              t.sum(w[i, k] * x[j, k], axis=k))

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



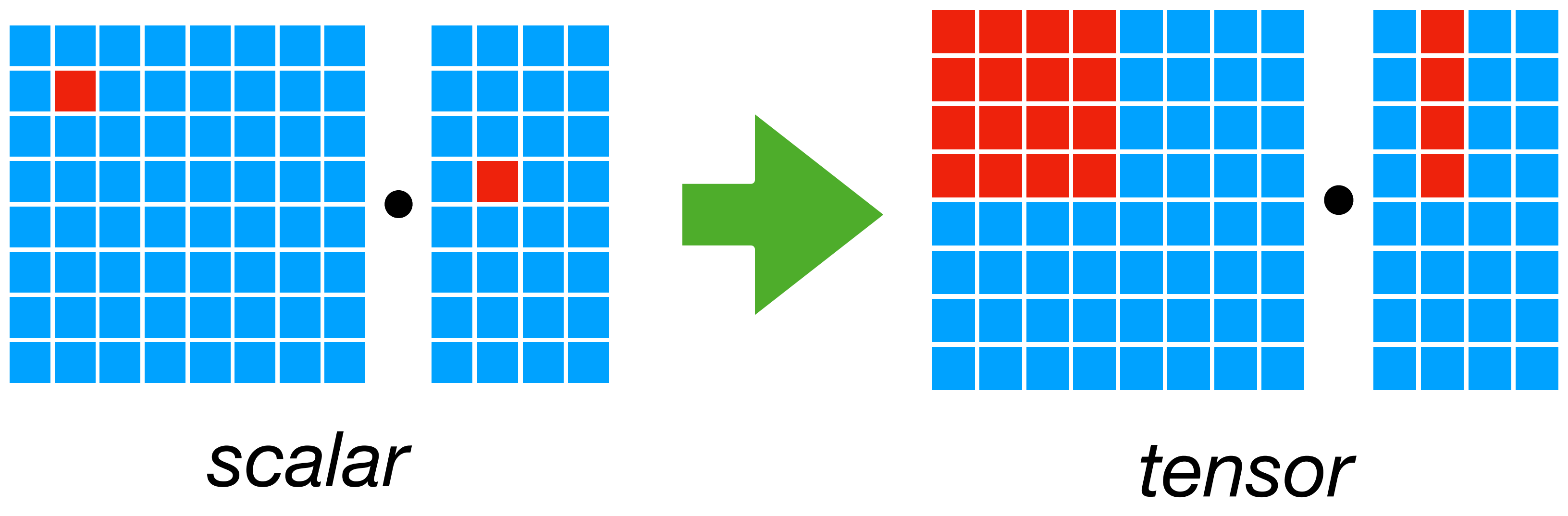
Hardware designer: declare tensor instruction interface with Tensor Expression

```
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

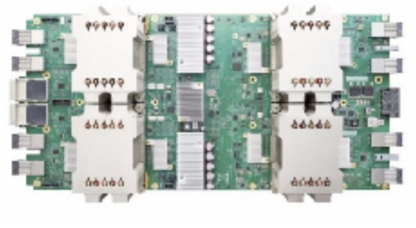
lowering rule to generate hardware intrinsics to carry out the computation

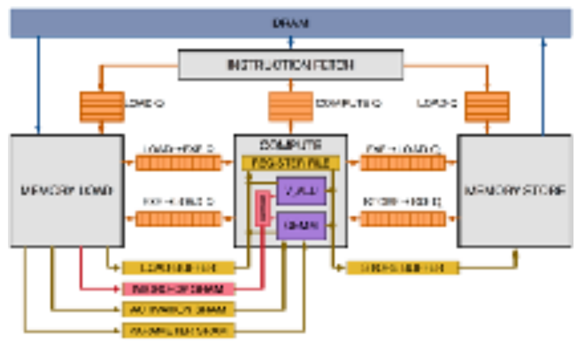
Tensorize: transform program to use tensor instructions



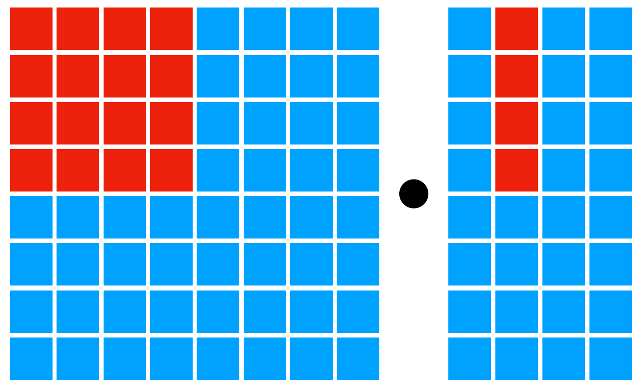
Hardware-aware Search Space

TPU-like Specialized Accelerators



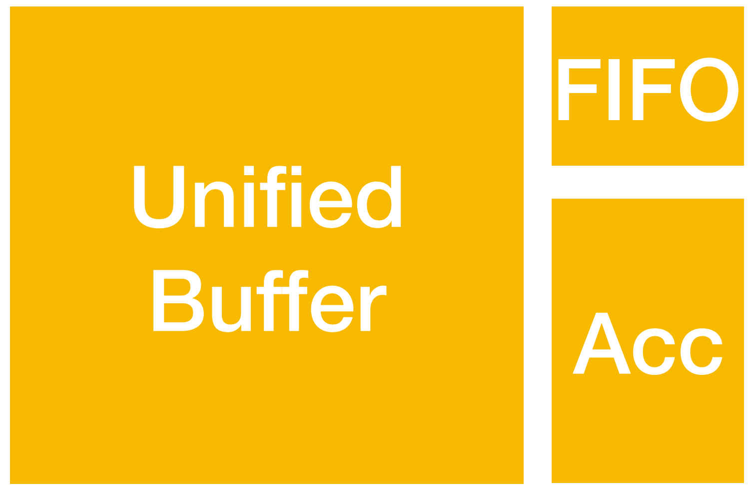


Compute Primitives



tensor

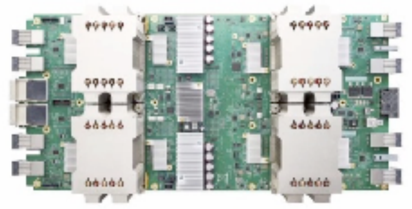
Memory Subsystem

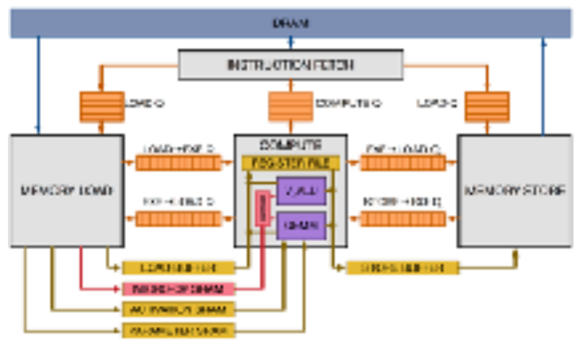


explicitly managed

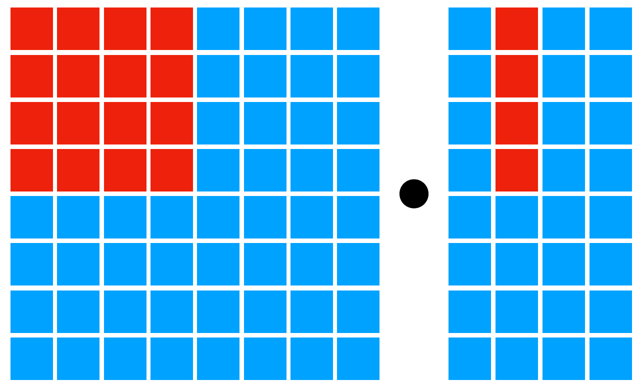
Hardware-aware Search Space

TPU-like Specialized Accelerators



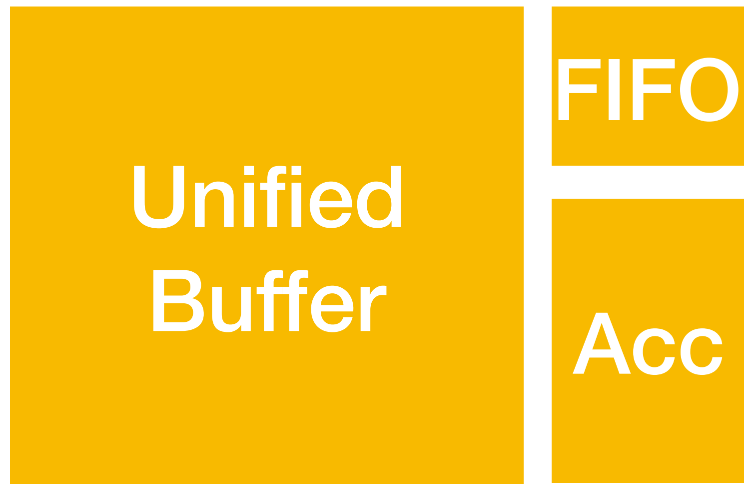


Compute Primitives



tensor

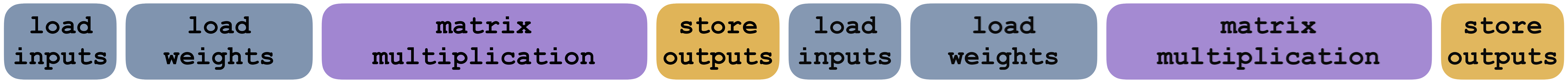
Memory Subsystem



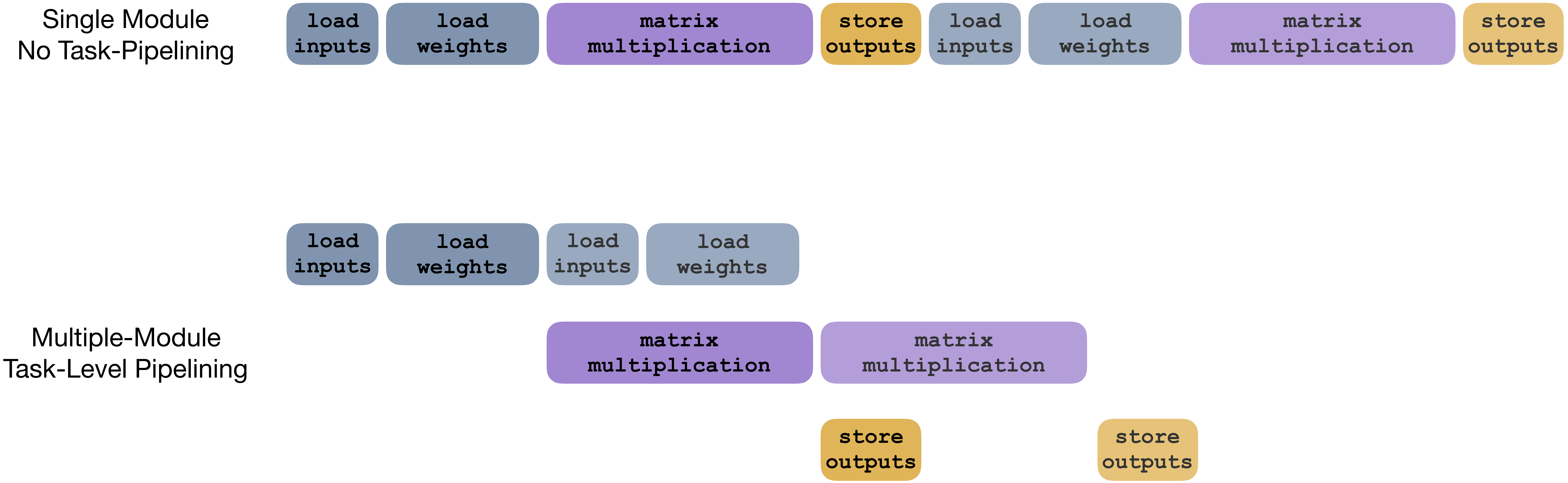
explicitly managed

Software Support for Latency Hiding

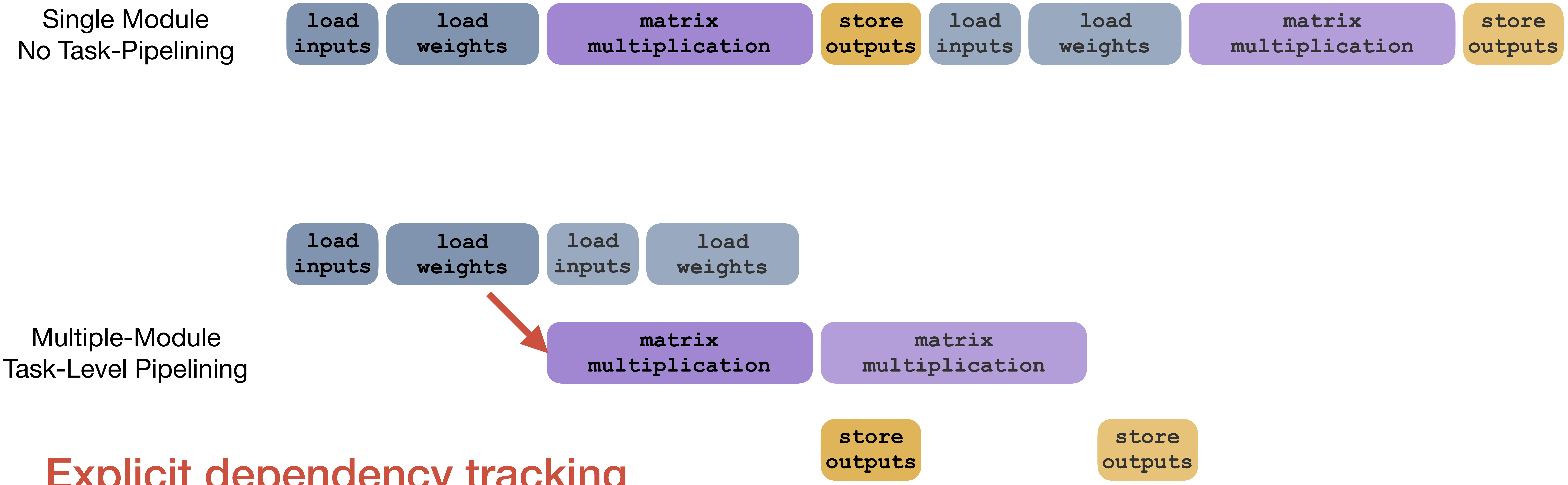
Single Module
No Task-Pipelining



Software Support for Latency Hiding

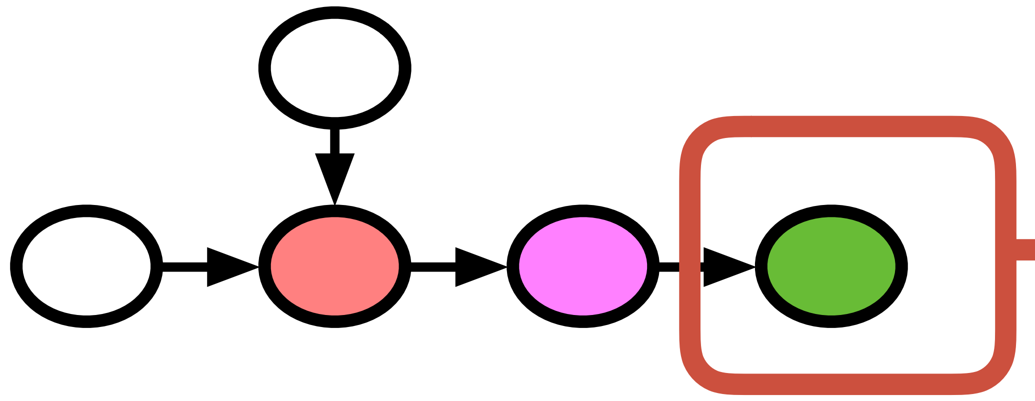


Software Support for Latency Hiding



Explicit dependency tracking
managed by software to hide memory latency

Hardware-aware Search Space

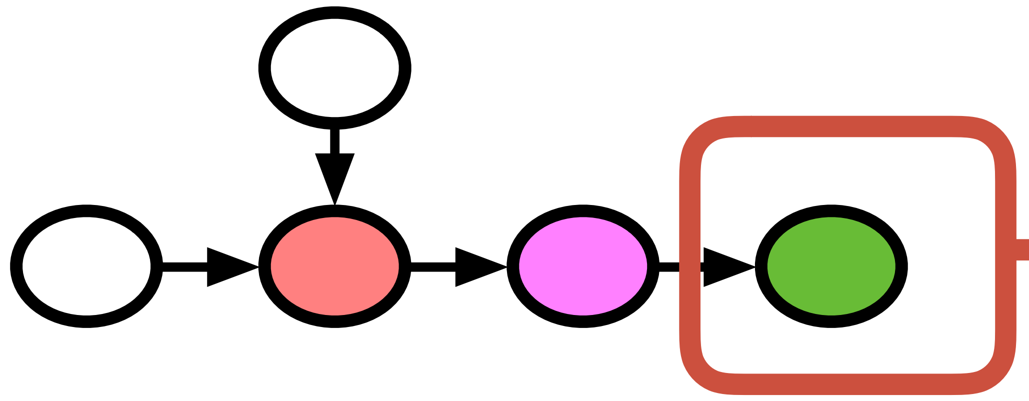


Tensor Expression Language

```
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

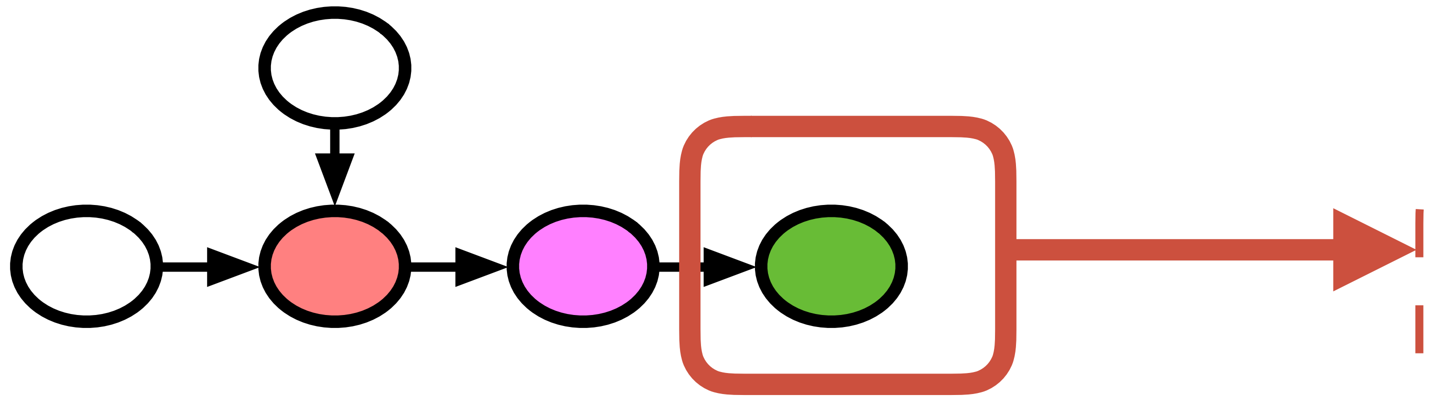
```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

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))
```

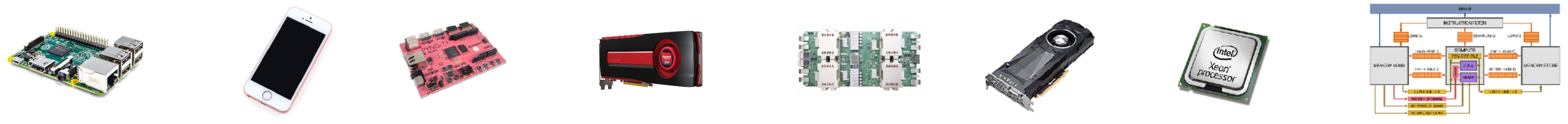
Primitives in prior work: Halide, Loopy

- Loop Transformations
- Thread Bindings
- Cache Locality

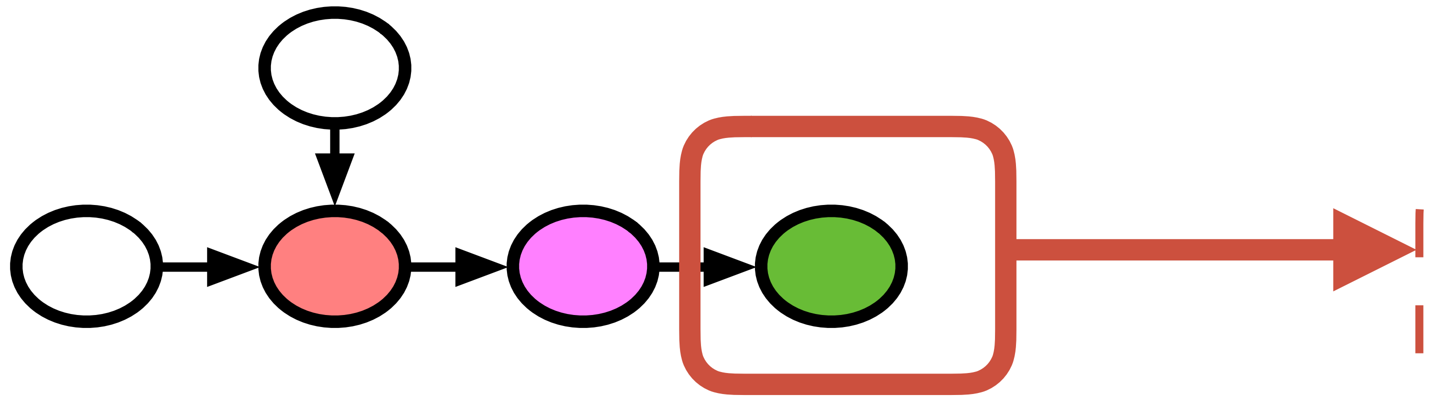
New primitives for GPUs, and enable TPU-like Accelerators

- 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

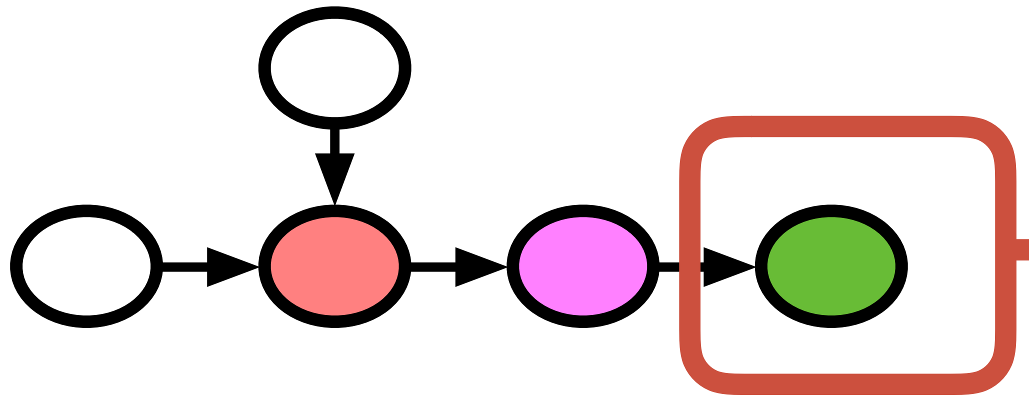
Thread Cooperation

Tensorization

Latency Hiding

Hardware

Hardware-aware Search Space



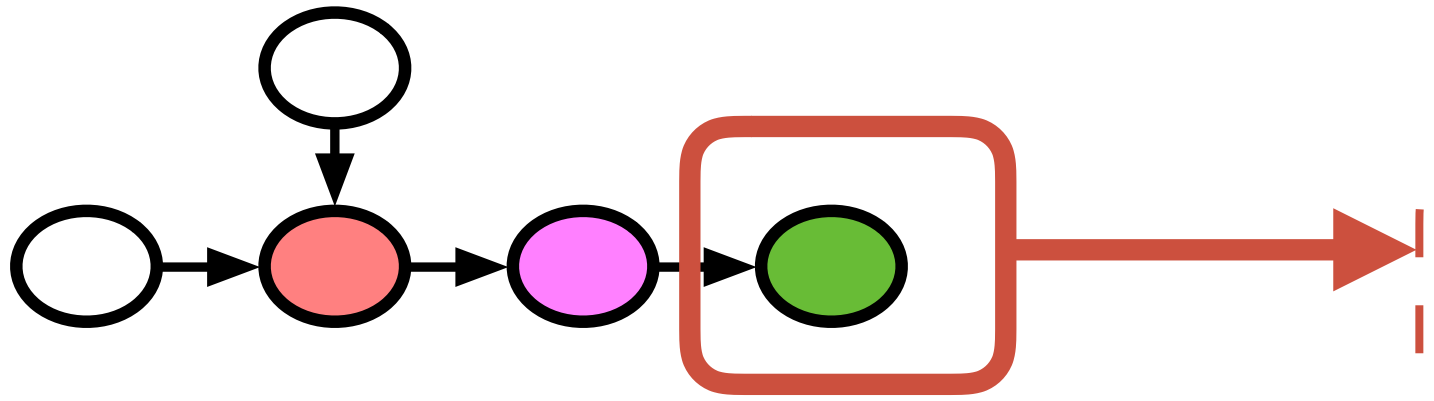
Tensor Expression Language

```
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



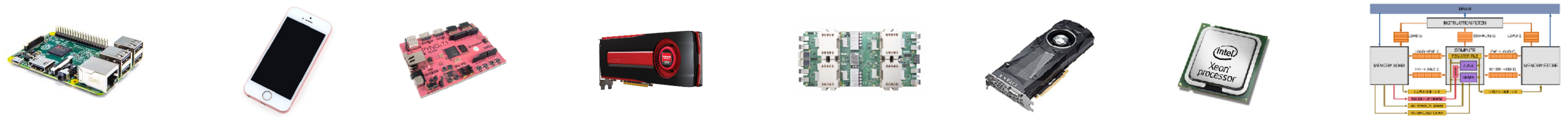
Tensor Expression Language

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

Billions of possible optimization choices

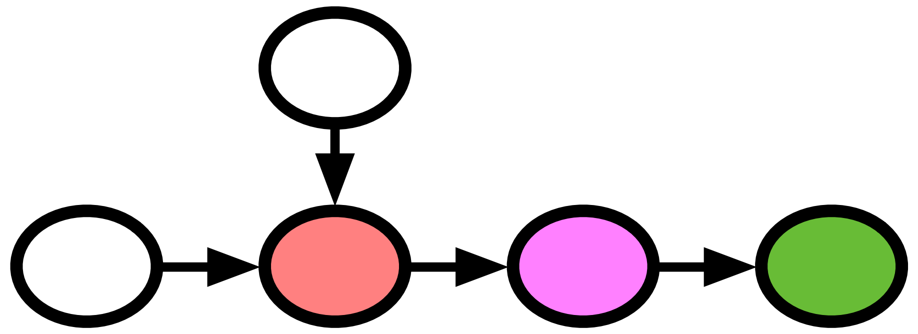
- Loop Transformations
- Thread Bindings
- Cache Locality
- Thread Cooperation
- Tensorization
- Latency Hiding

Hardware



Learning-based Learning System

Frameworks


High-level data flow graph and optimizations

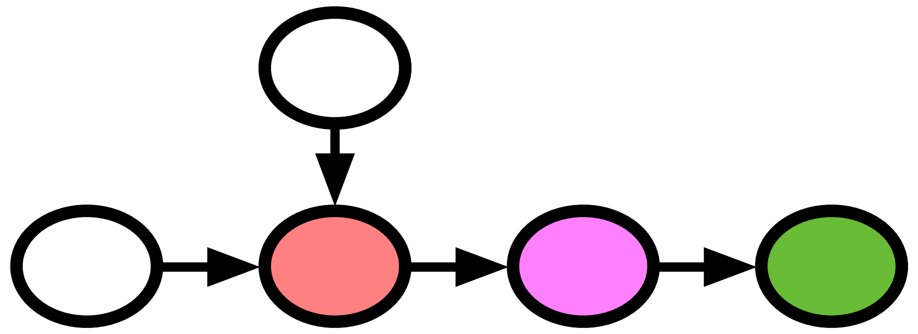
Hardware aware Search Space of Optimized Tensor Programs

Machine Learning based Program Optimizer

Hardware

Learning-based Learning System

Frameworks



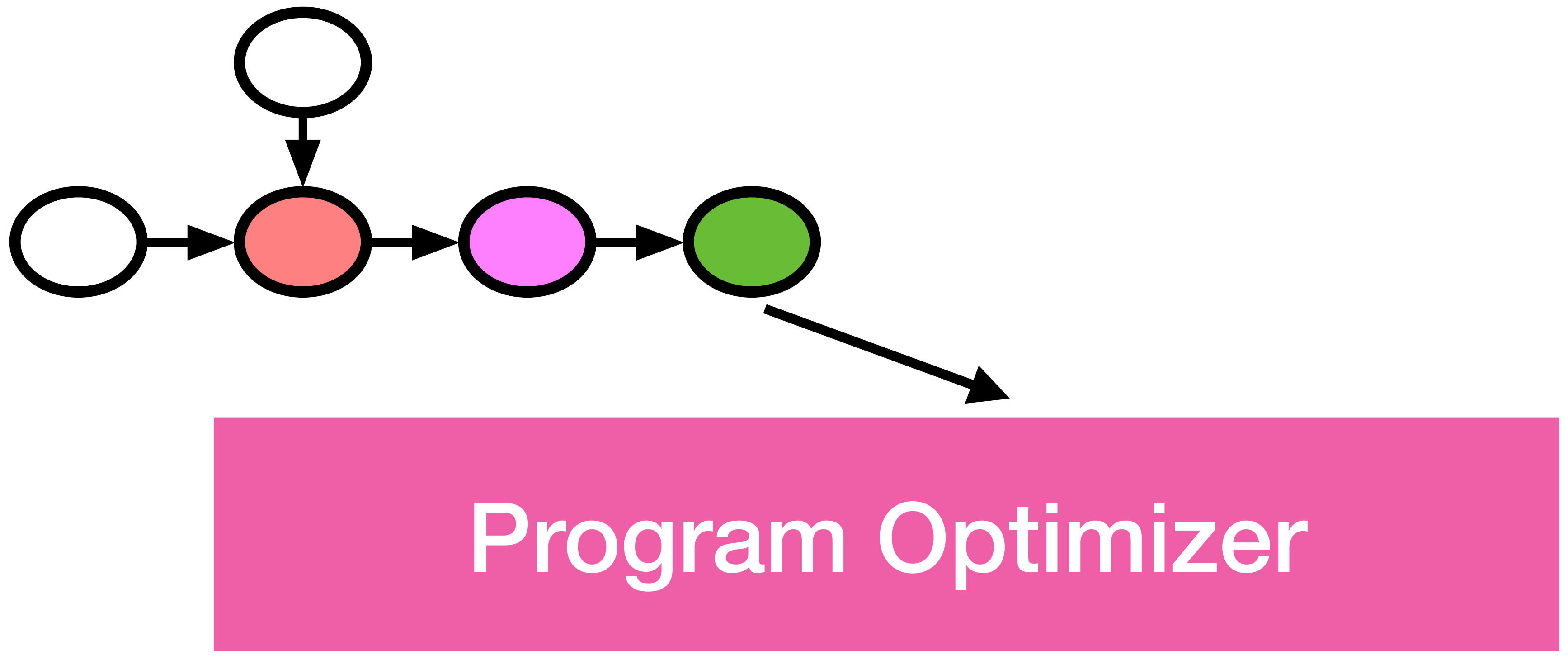
High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

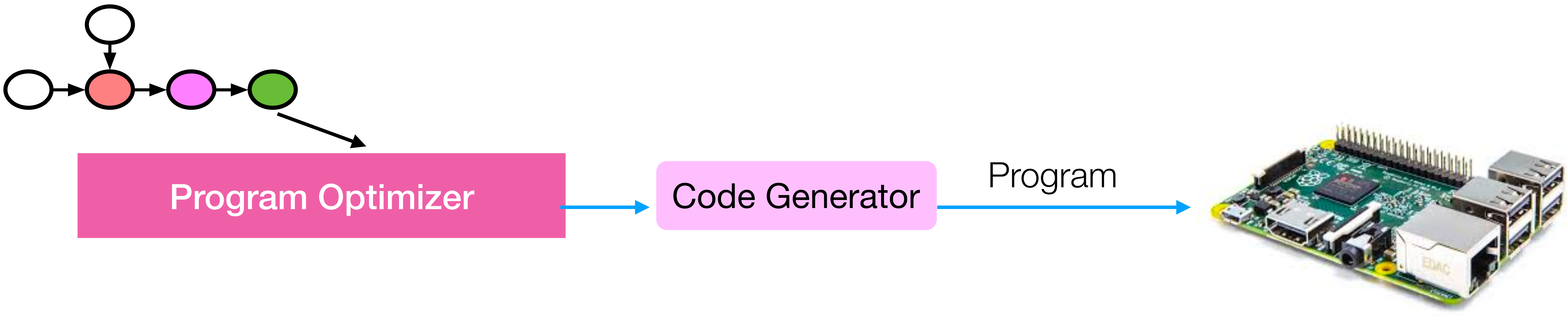
Machine Learning based Program Optimizer

Hardware

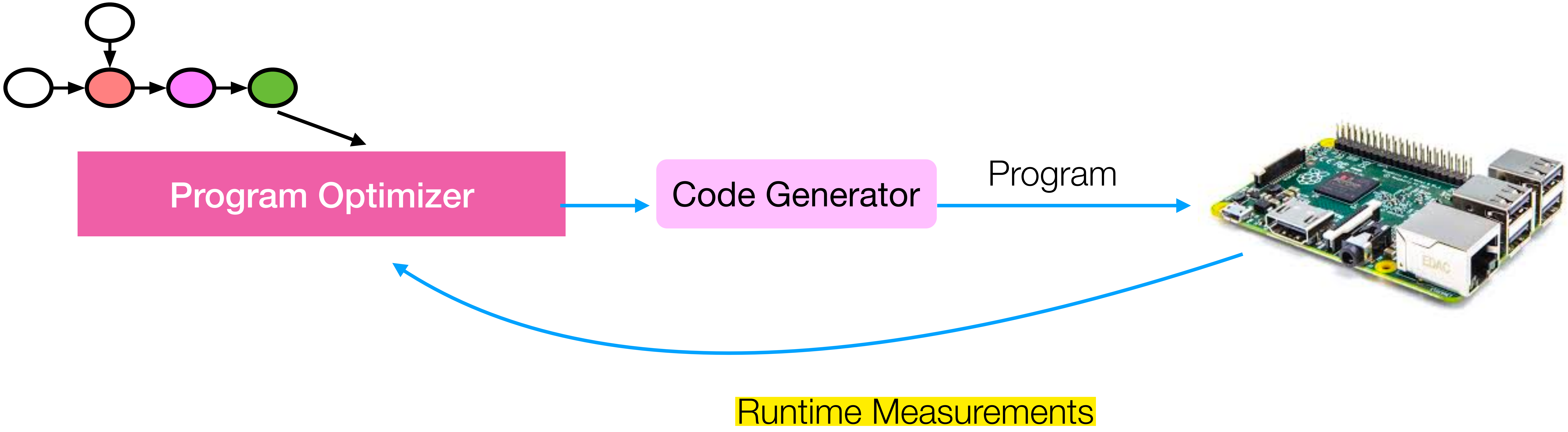
Learning-based Program Optimizer



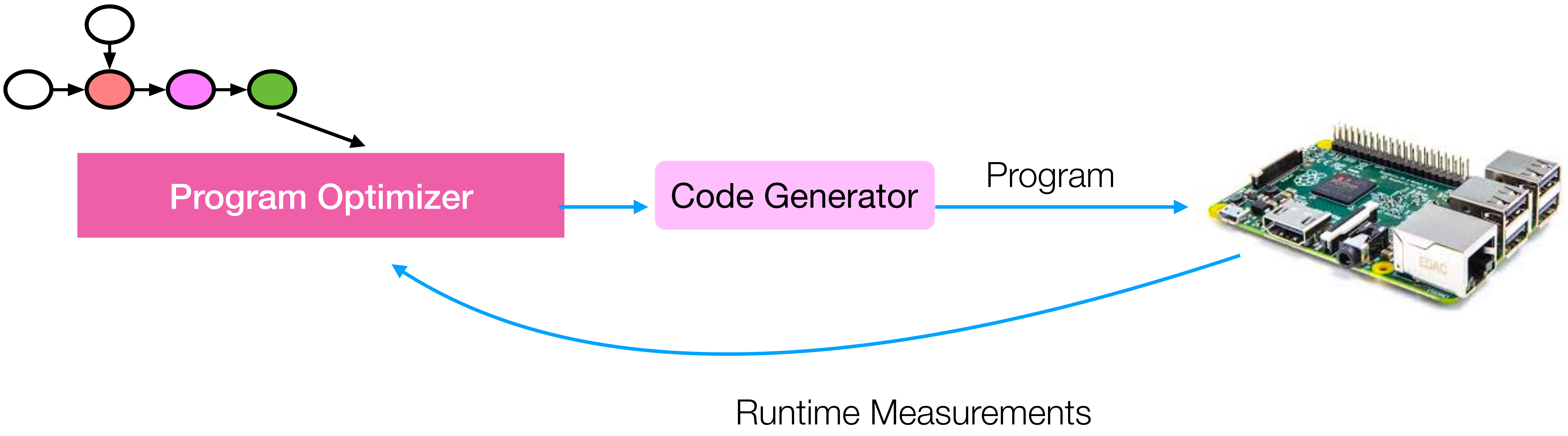
Learning-based Program Optimizer



Learning-based Program Optimizer

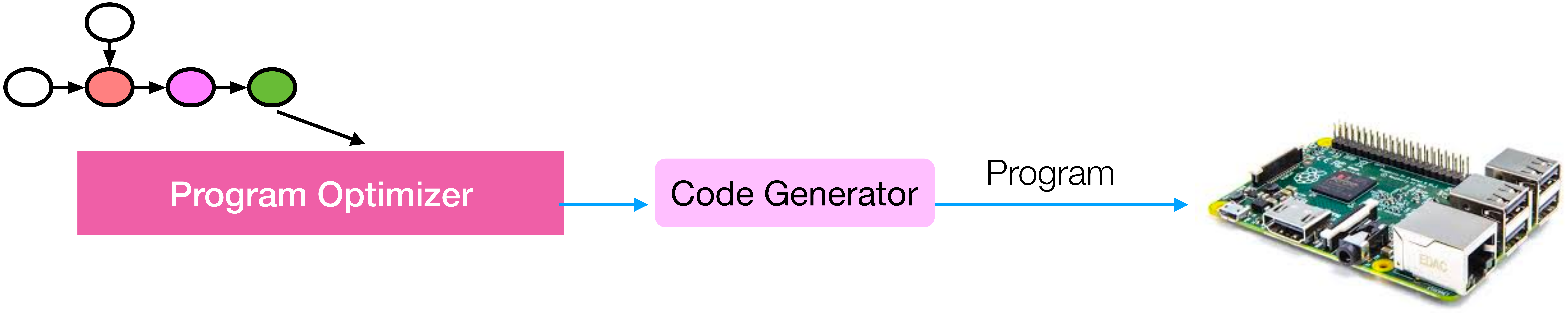


Learning-based Program Optimizer

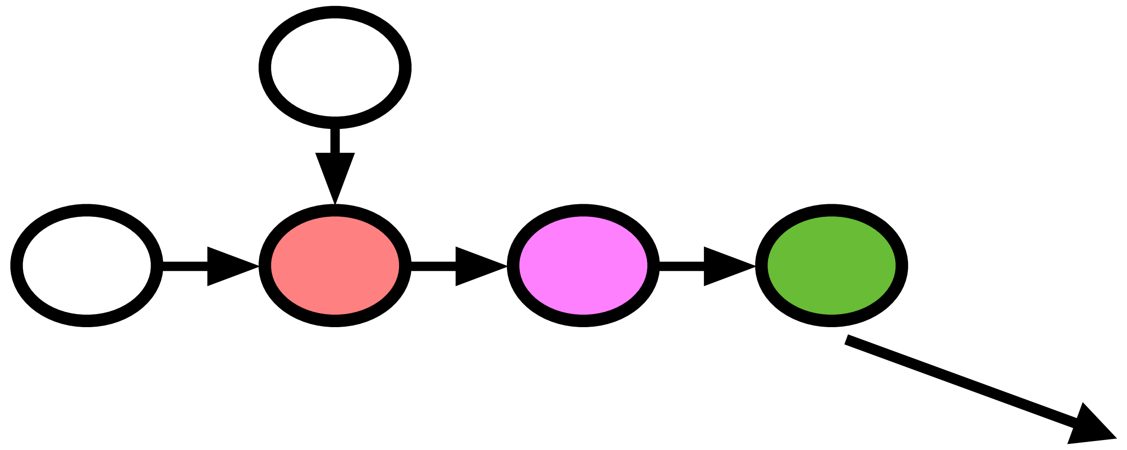


High experiment cost,
each trial costs ~1second

Learning-based Program Optimizer



Learning-based Program Optimizer



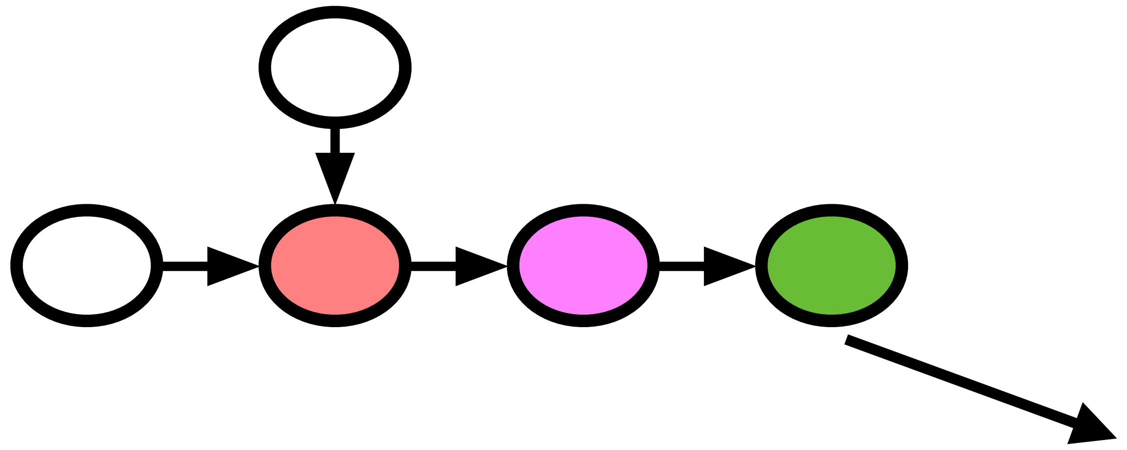
Program Optimizer

Code Generator

Program

Cost Model

Learning-based Program Optimizer



Program Optimizer

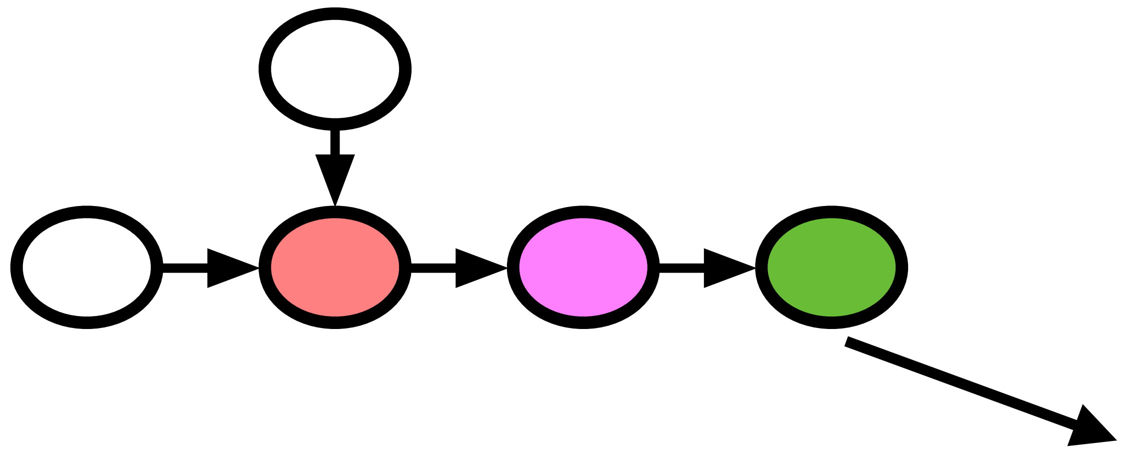
Code Generator

Program

Cost Model

Need reliable cost model per hardware

Learning-based Program Optimizer

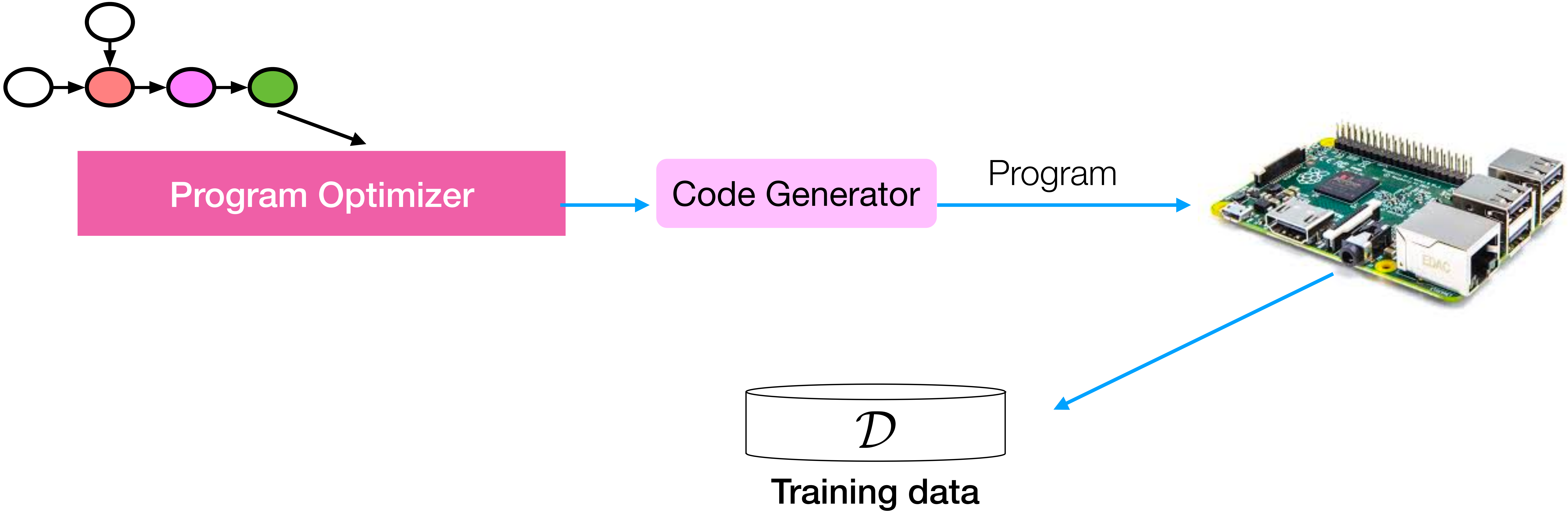


Program Optimizer

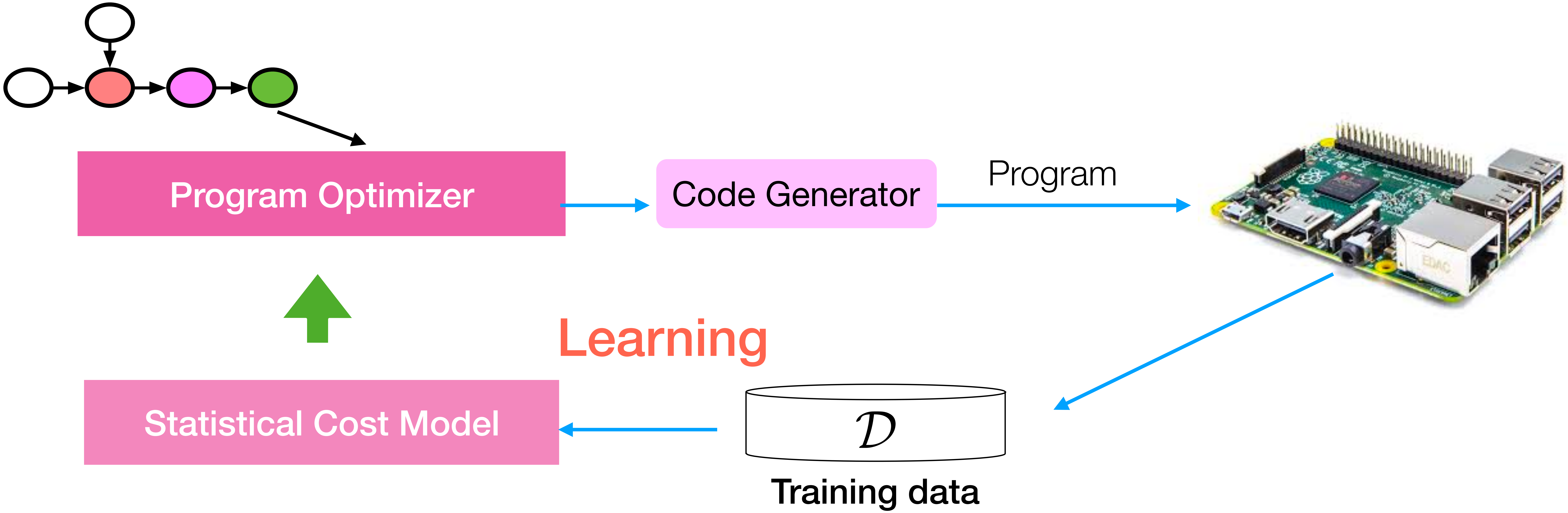
Code Generator

Program

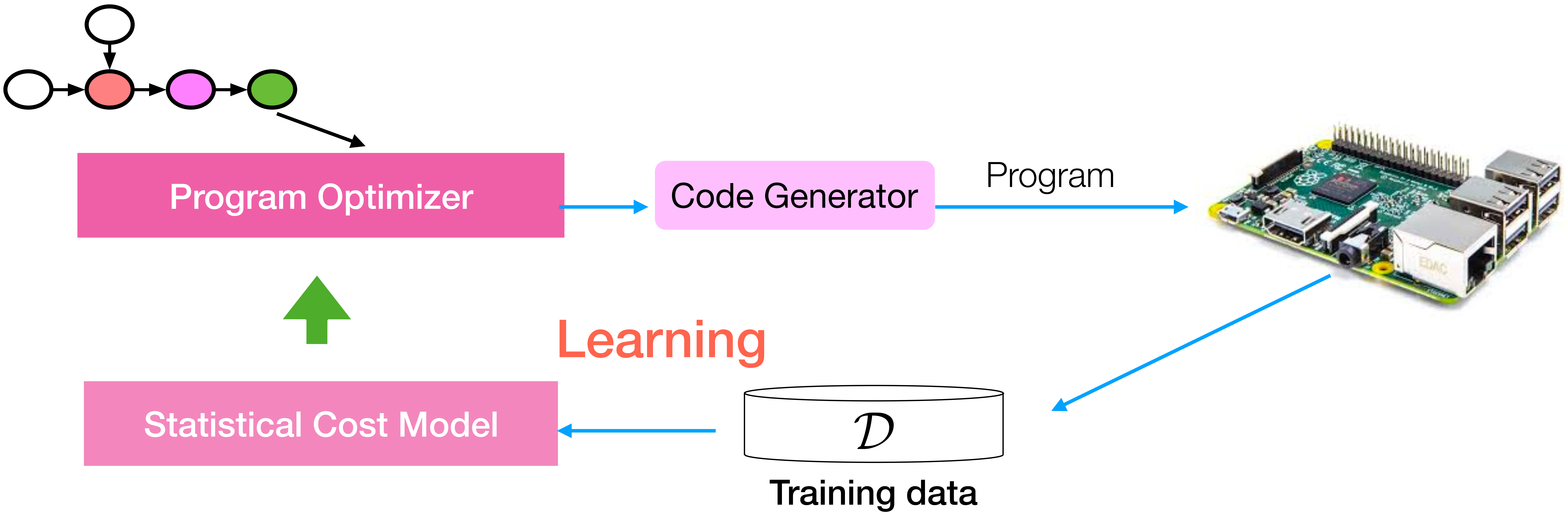
Learning-based Program Optimizer



Learning-based Program Optimizer



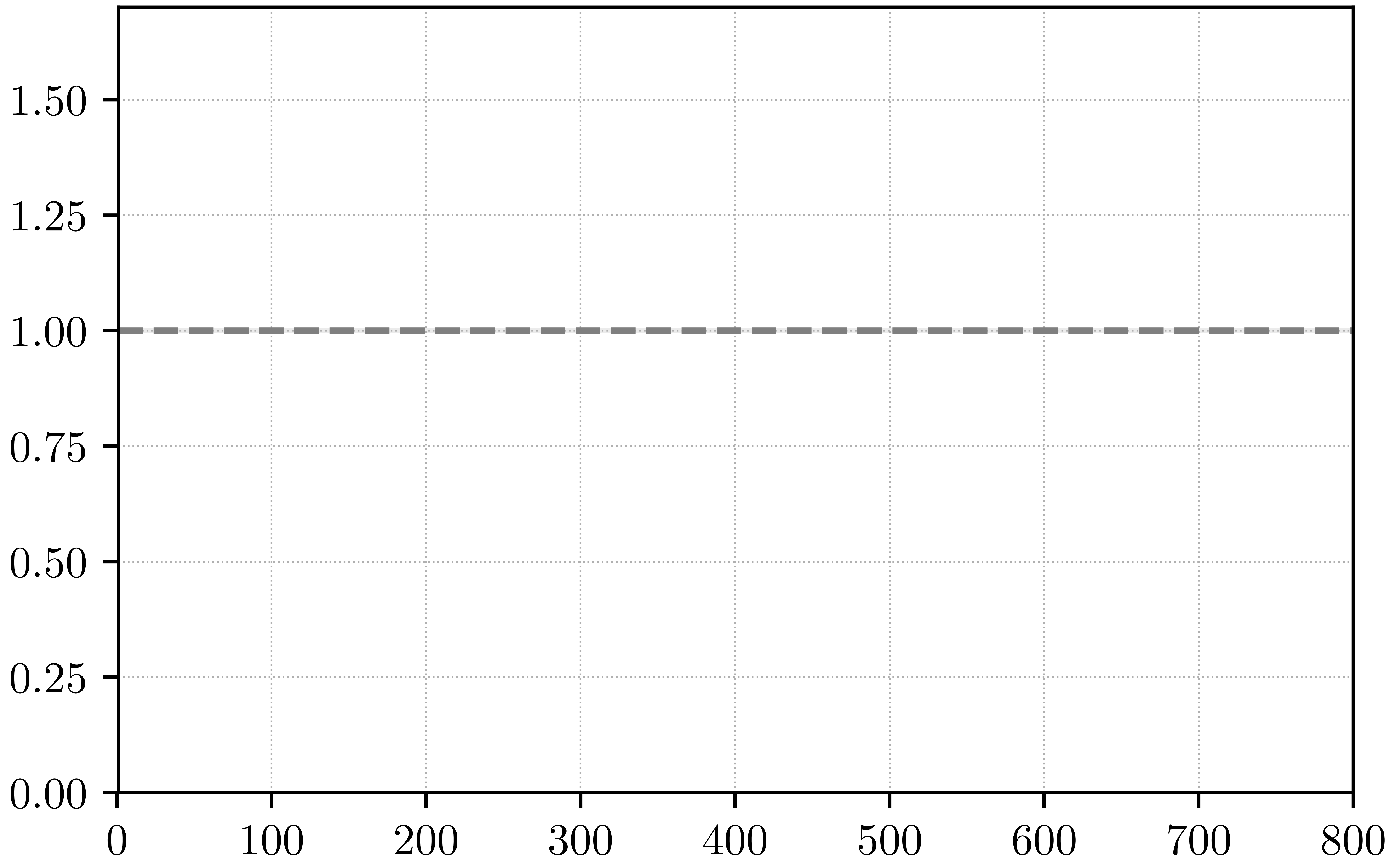
Learning-based Program Optimizer



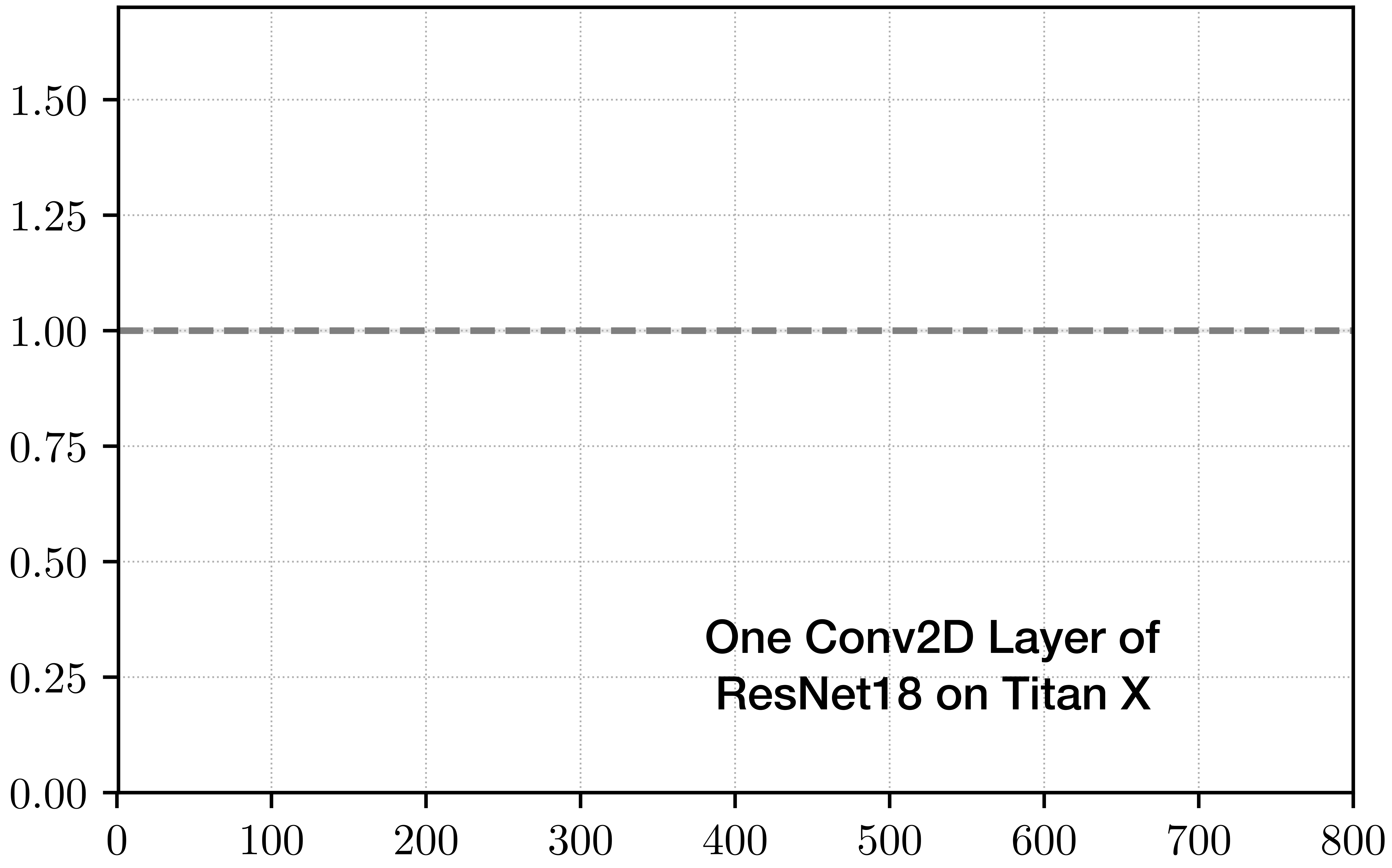
Adapt to hardware type by learning

Make prediction in 1ms level

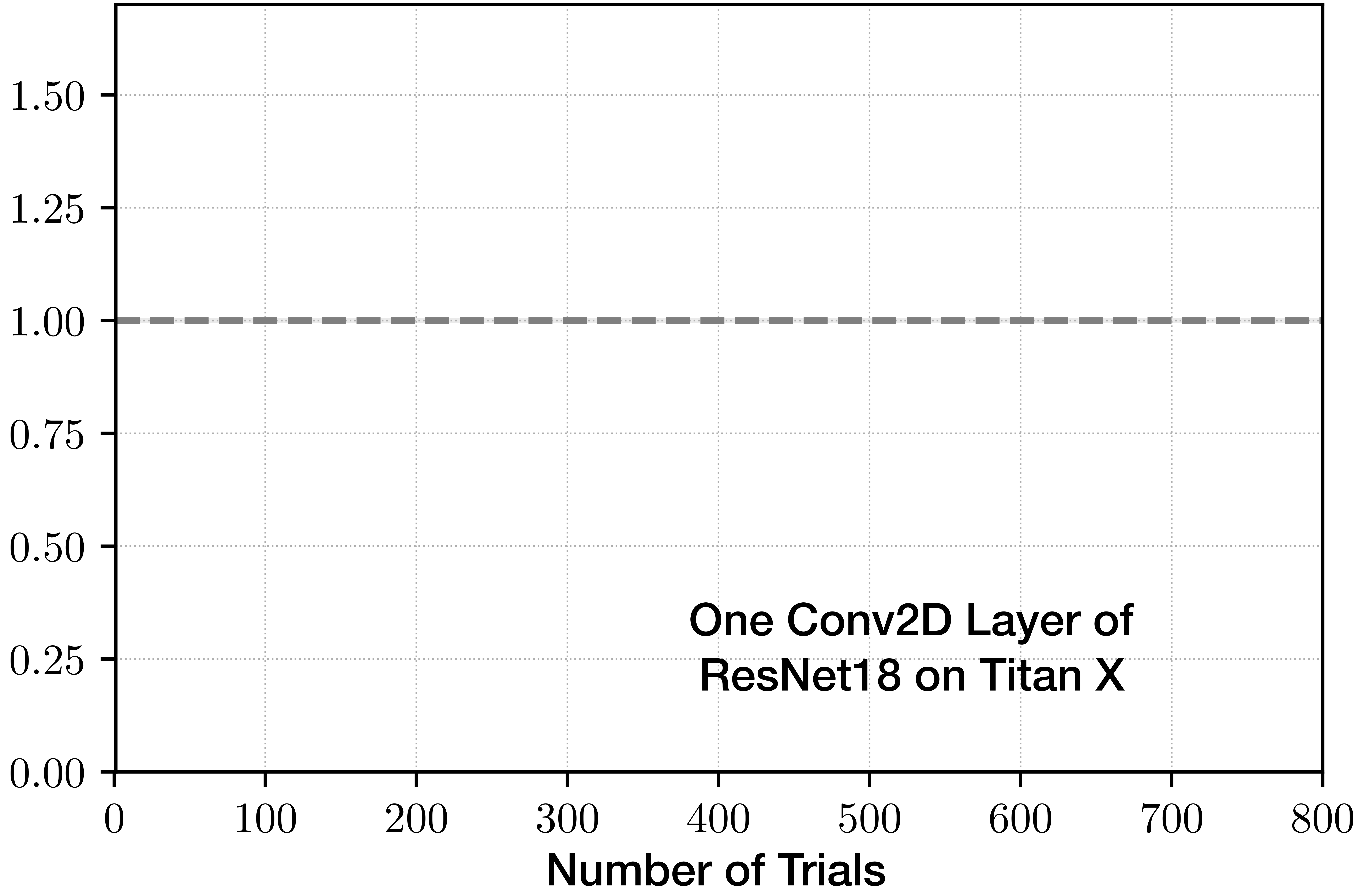
Effectiveness of ML based Model



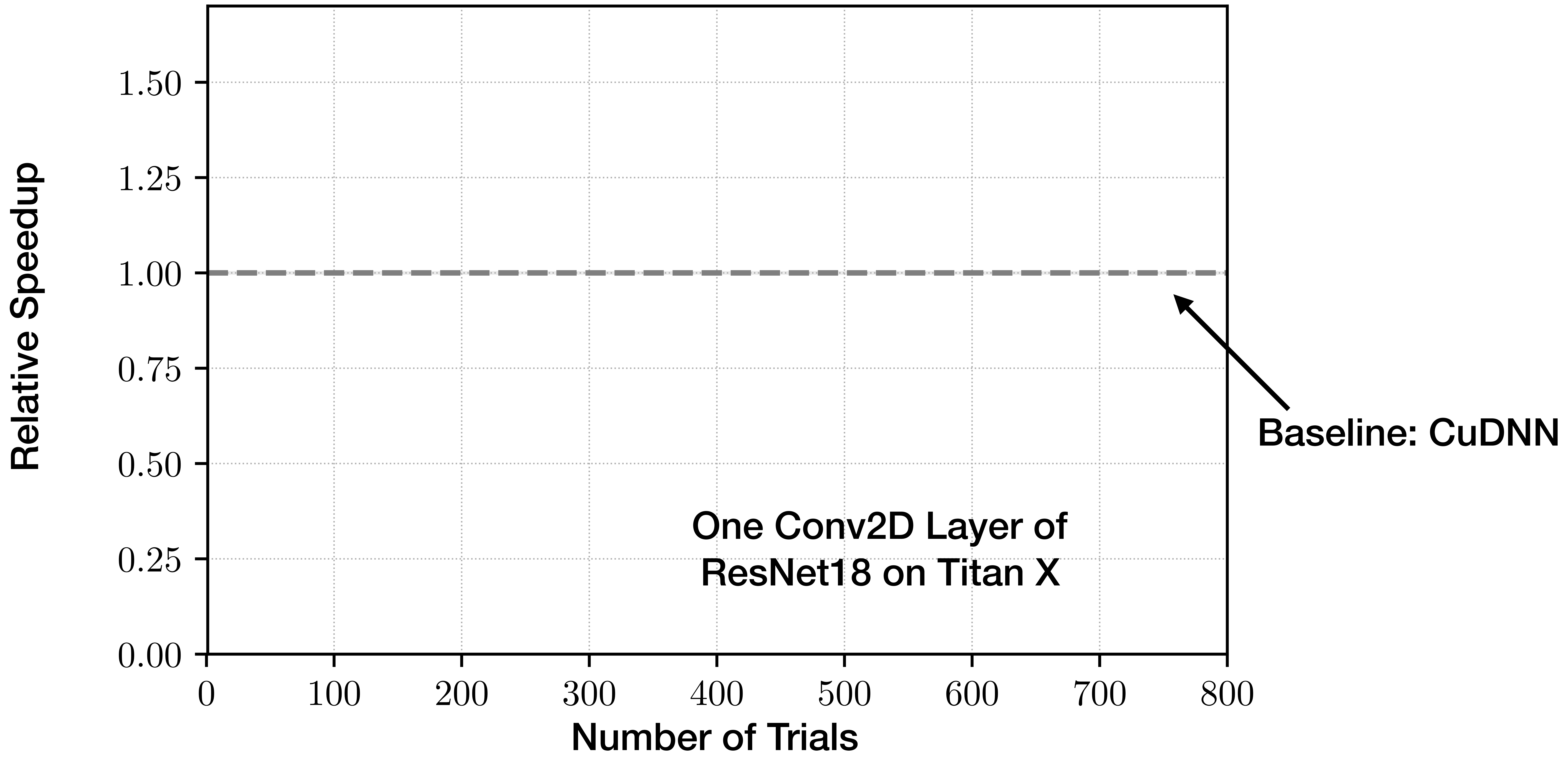
Effectiveness of ML based Model



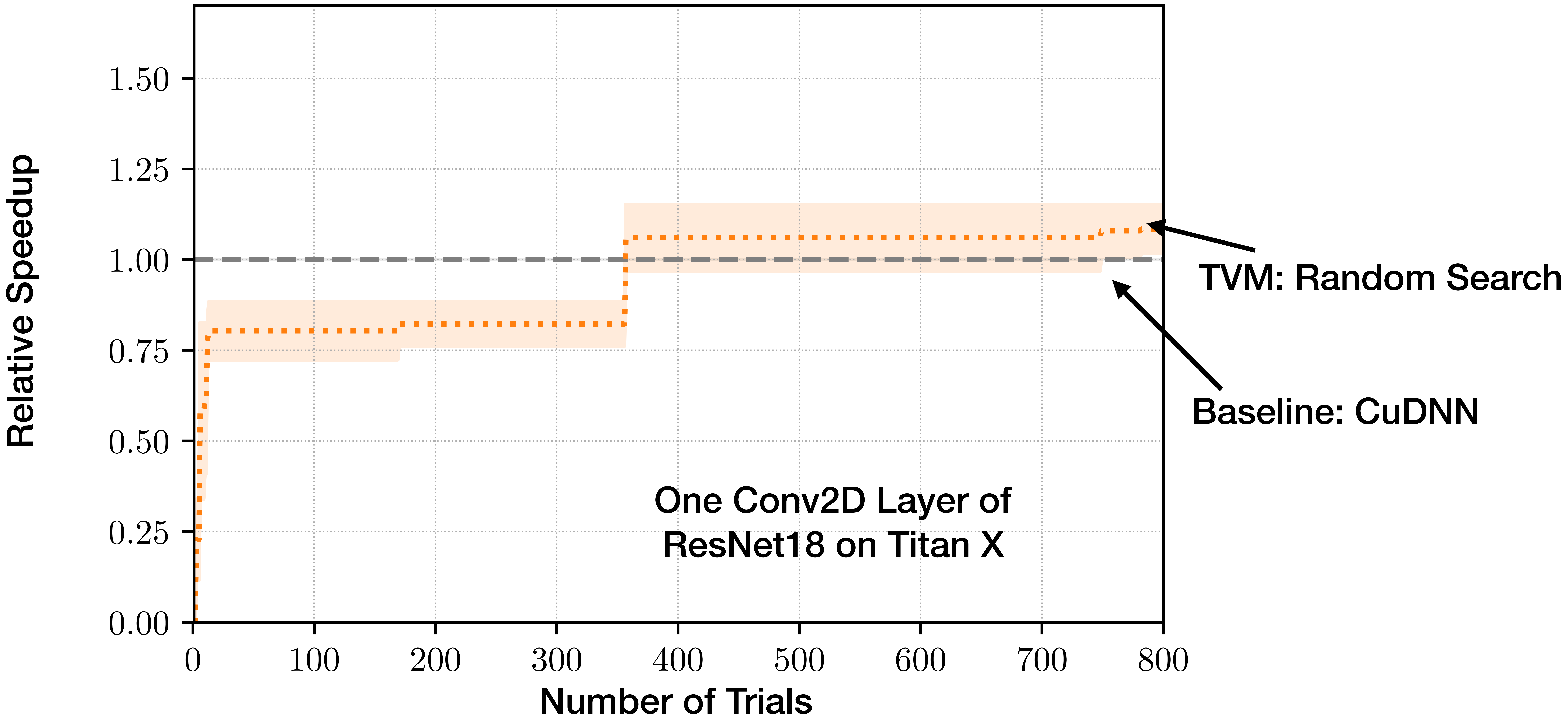
Effectiveness of ML based Model



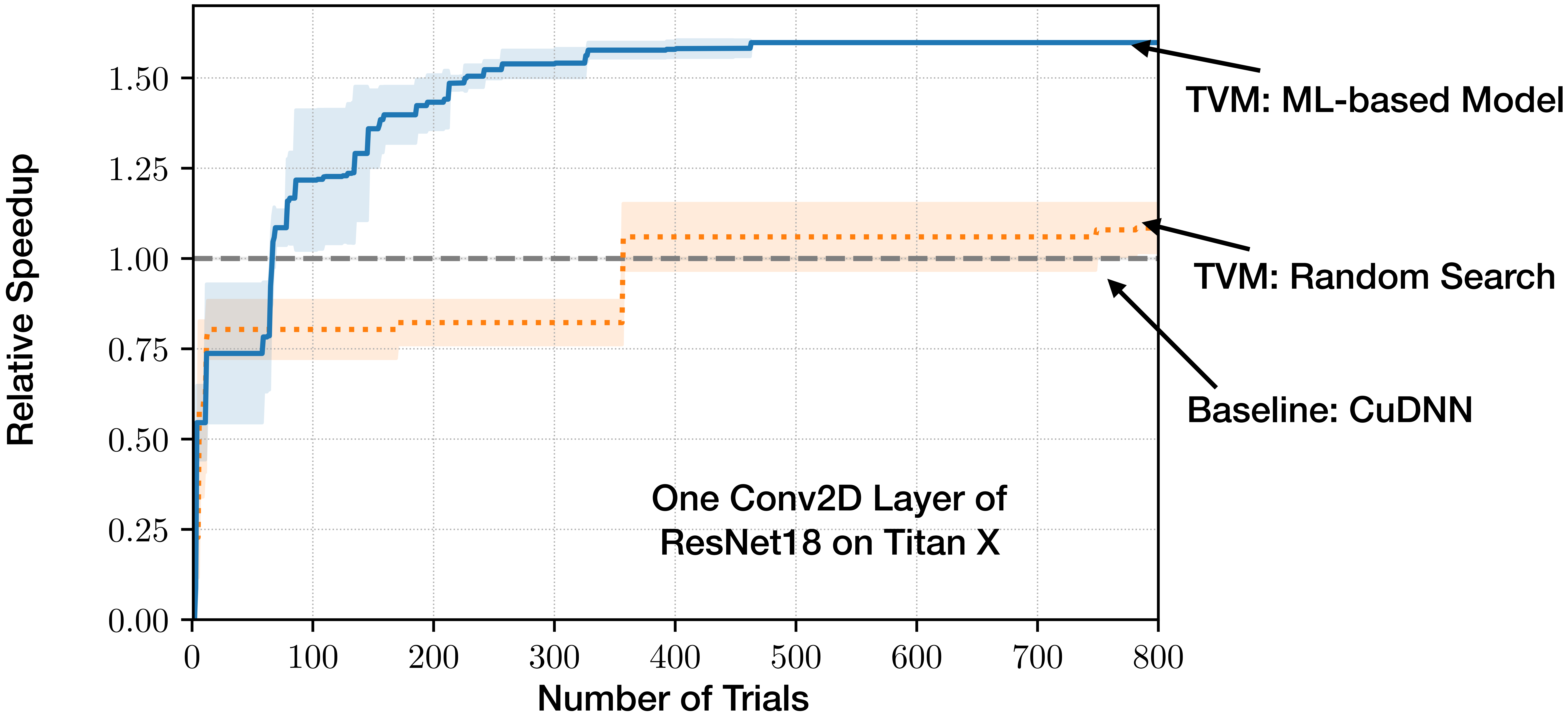
Effectiveness of ML based Model



Effectiveness of ML based Model

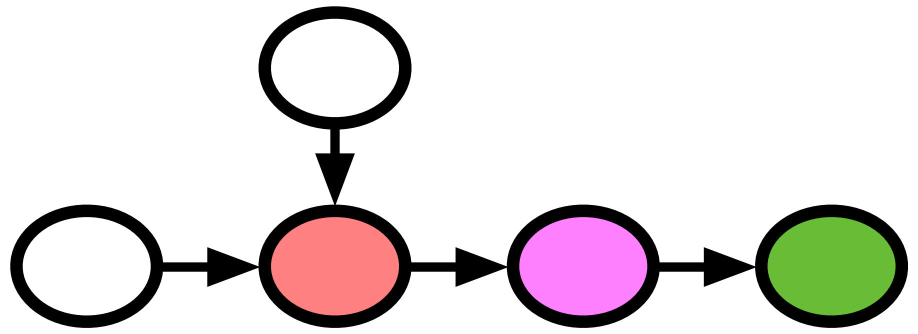


Effectiveness of ML based Model



Learning-based Learning System

Frameworks

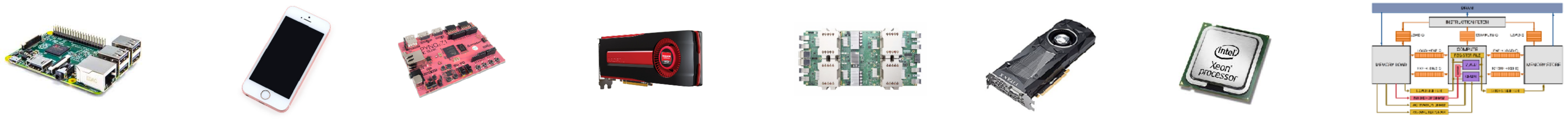


High-level data flow graph and optimizations

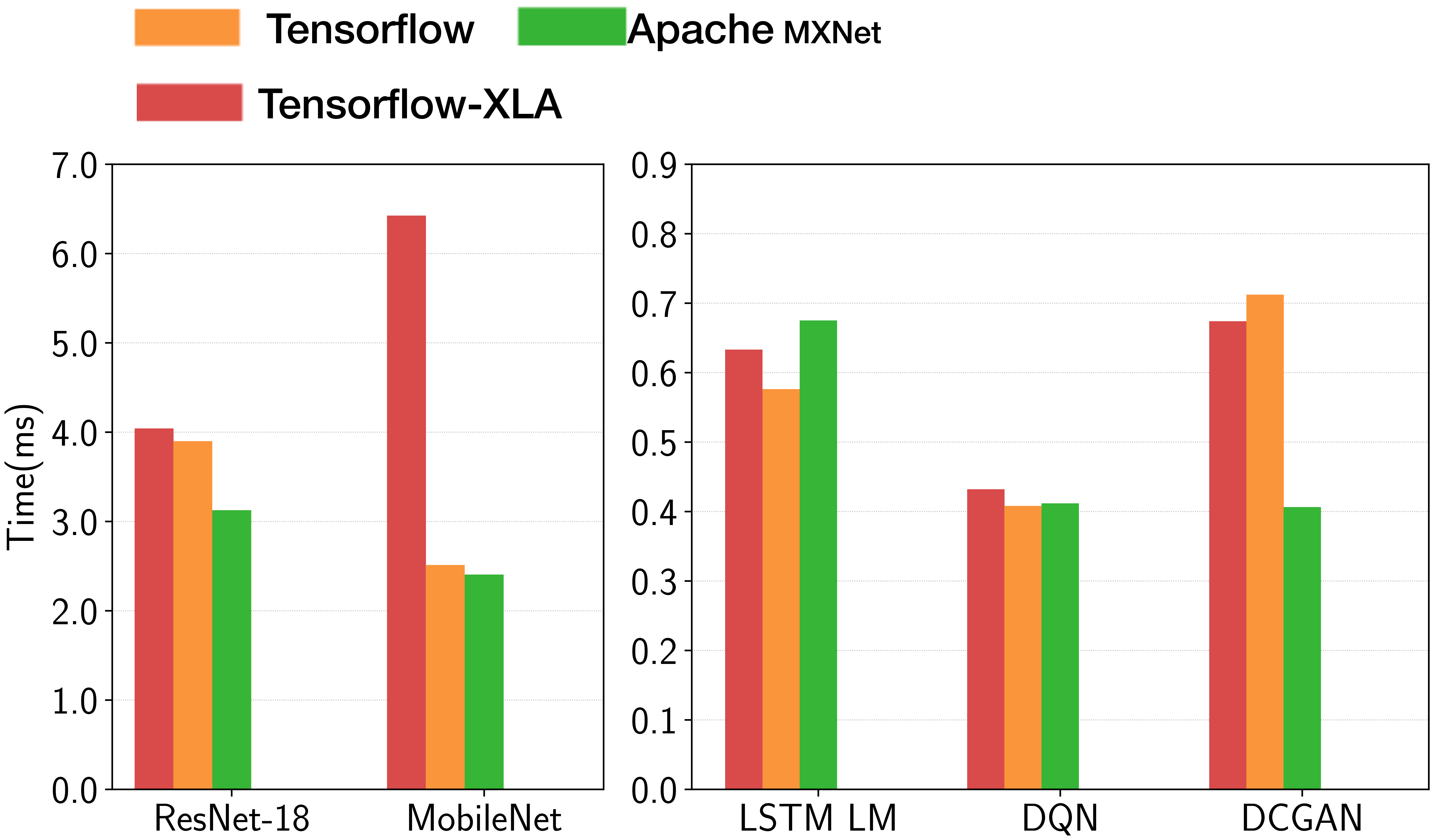
Hardware aware Search Space of Optimized Tensor Programs

Machine Learning based Program Optimizer

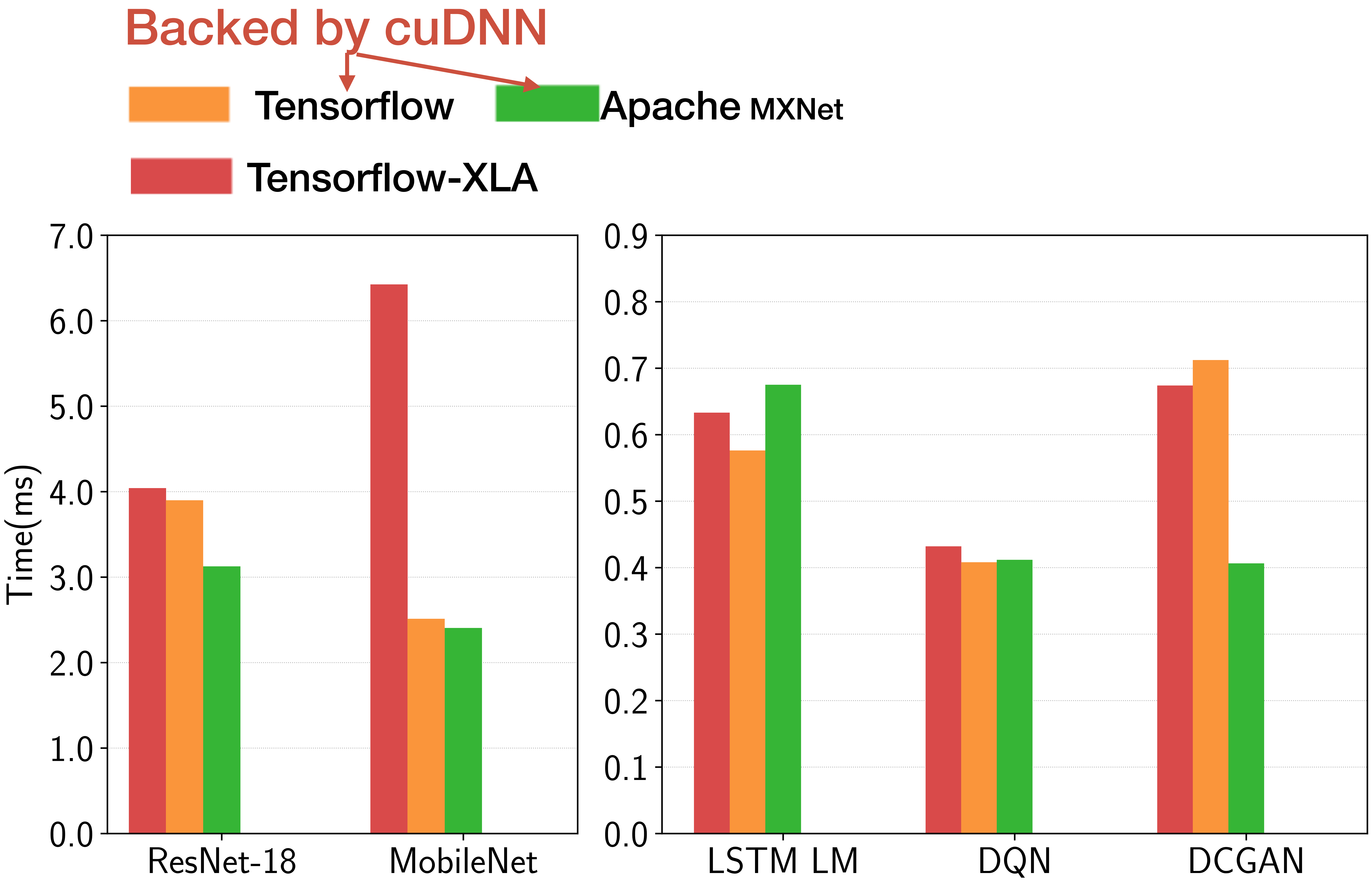
Hardware



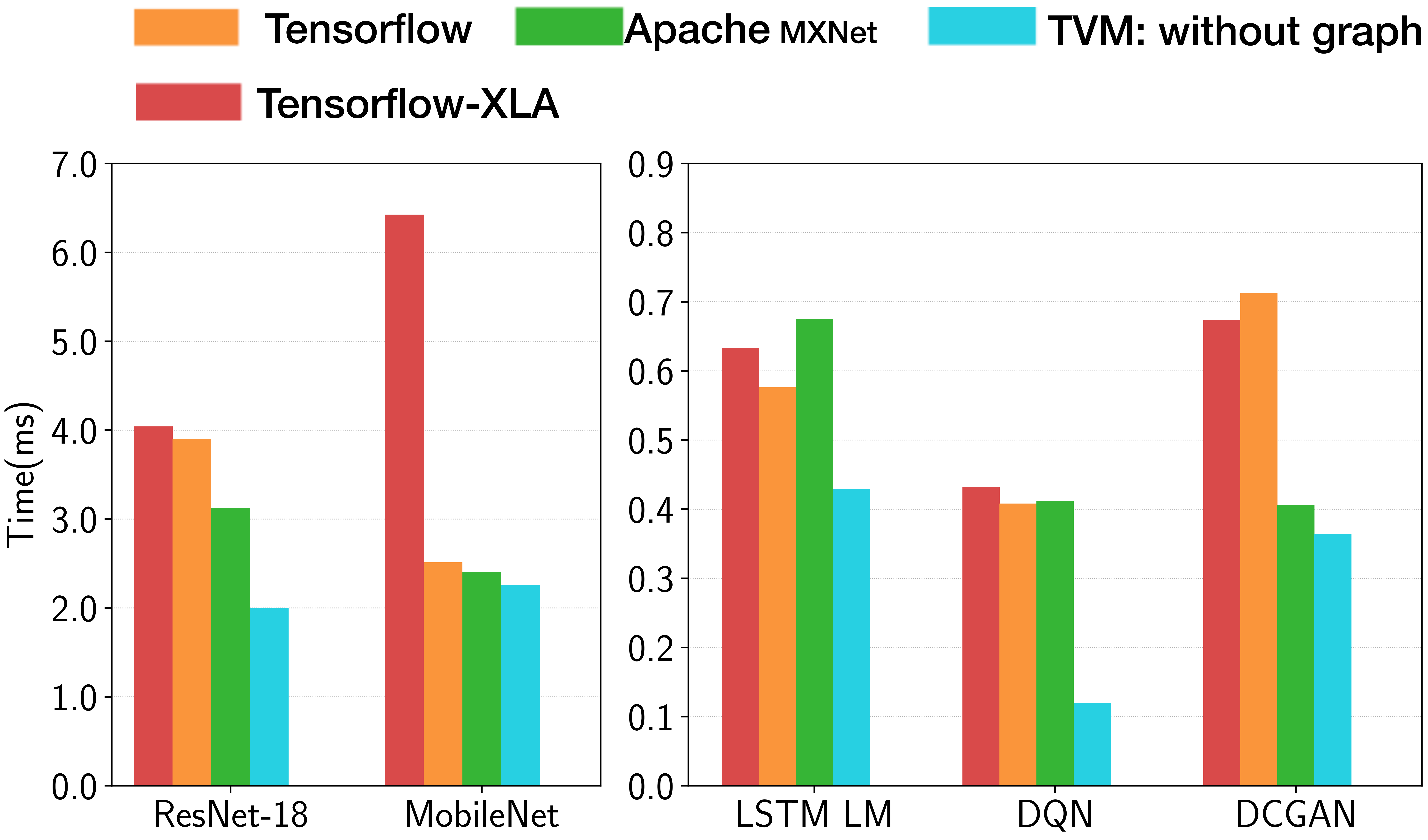
End to End Inference Performance (Nvidia Titan X)



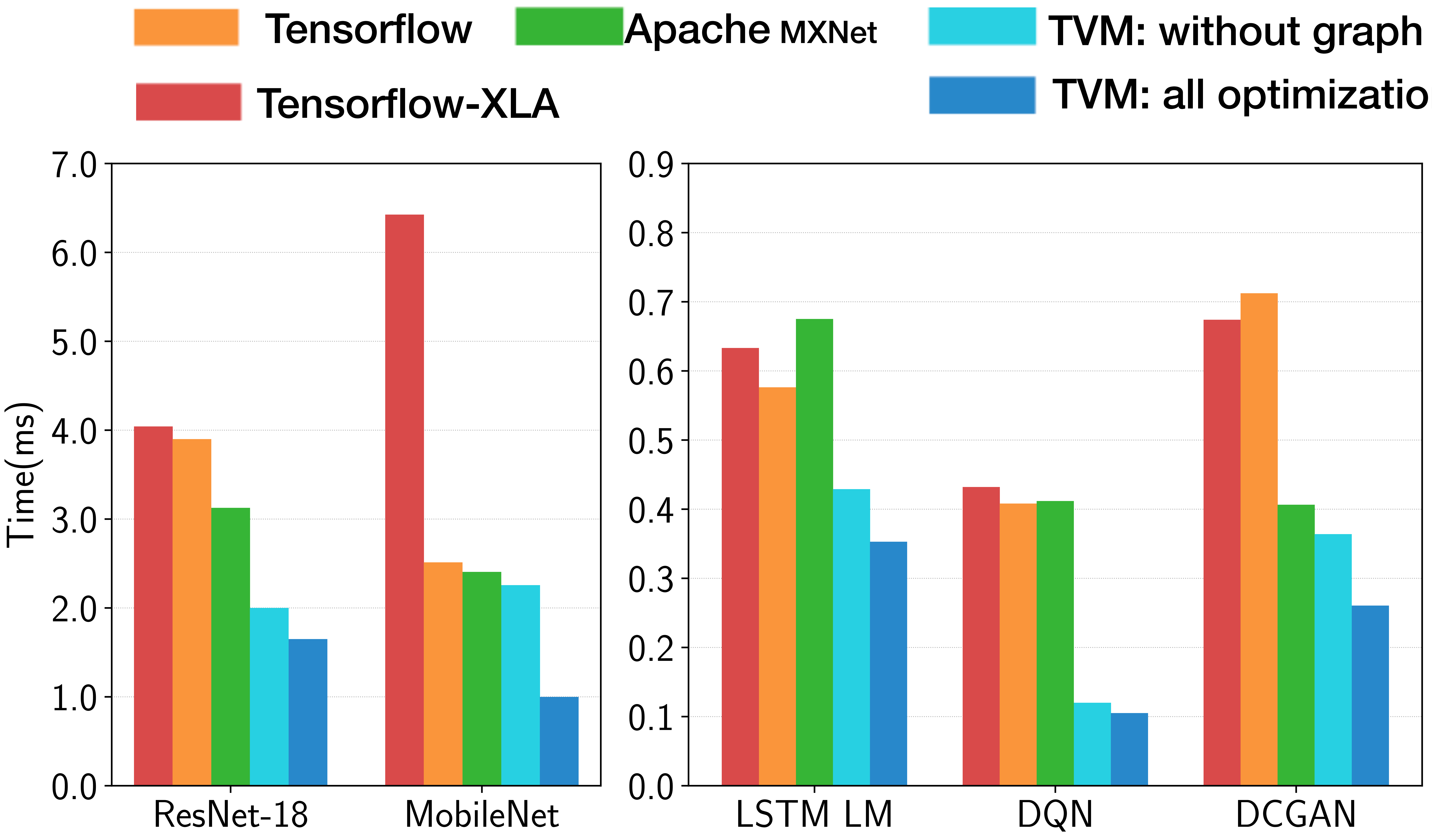
End to End Inference Performance (Nvidia Titan X)



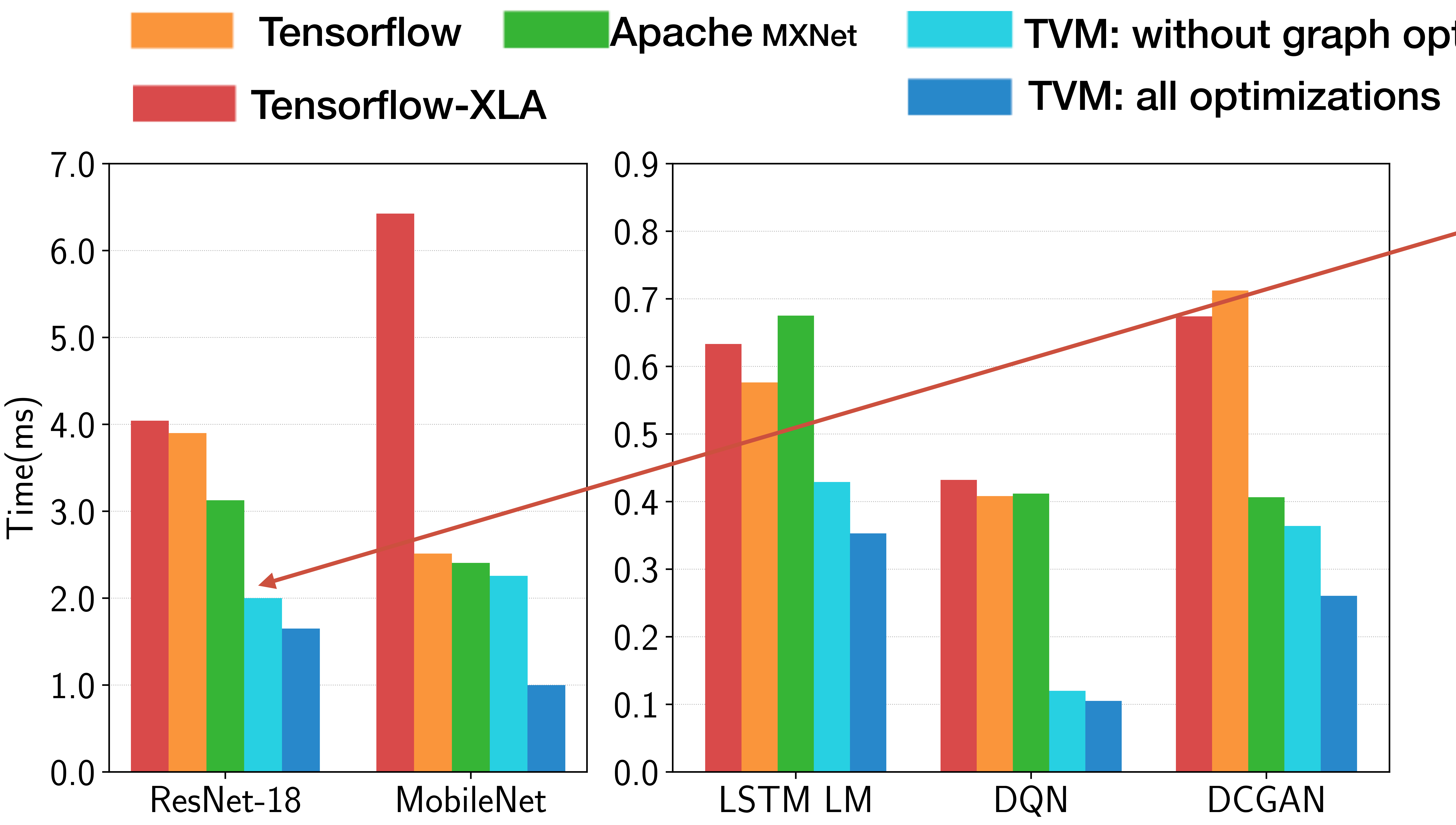
End to End Inference Performance (Nvidia Titan X)



End to End Inference Performance (Nvidia Titan X)

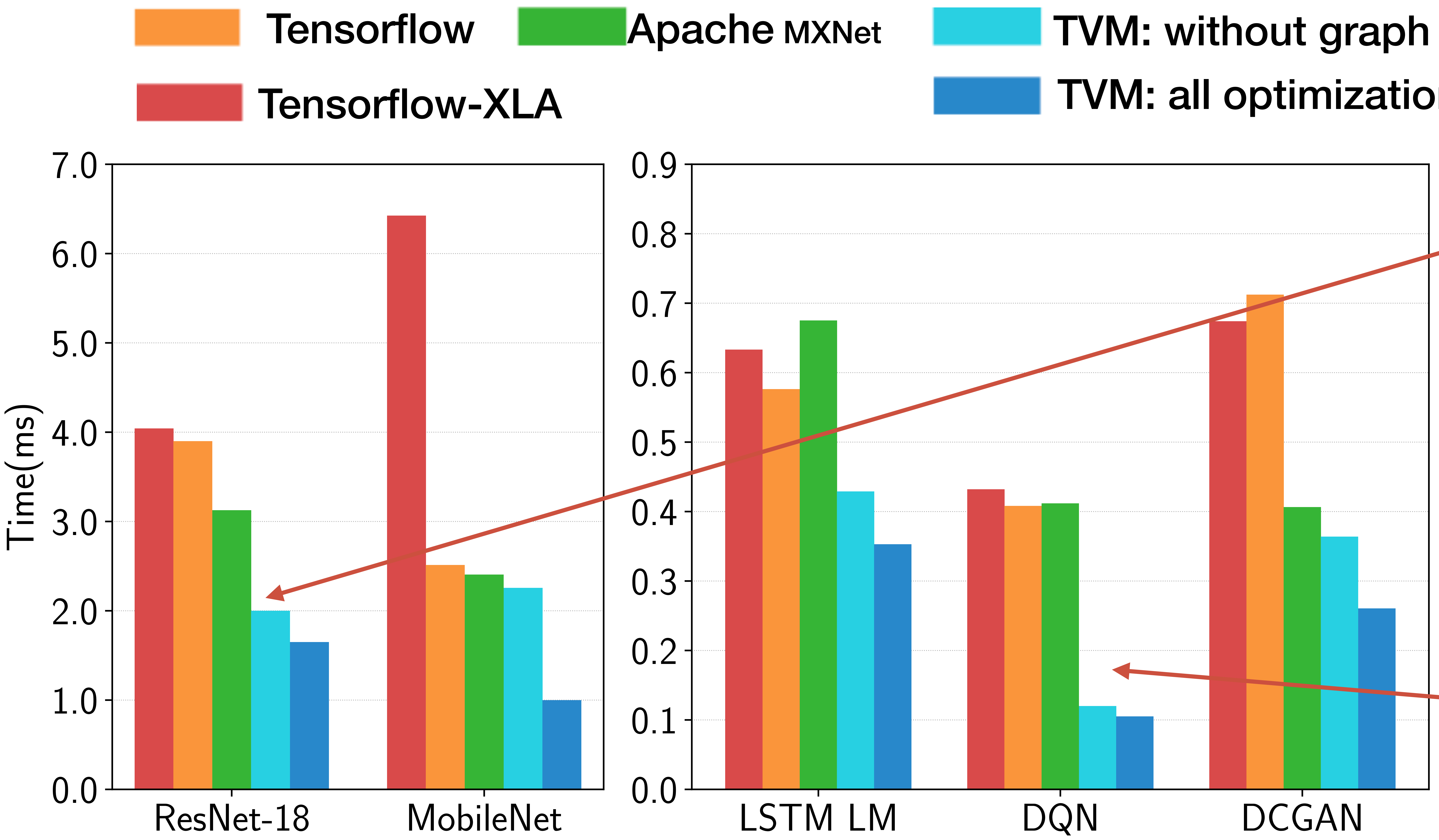


End to End Inference Performance (Nvidia Titan X)



Competitive on standard models

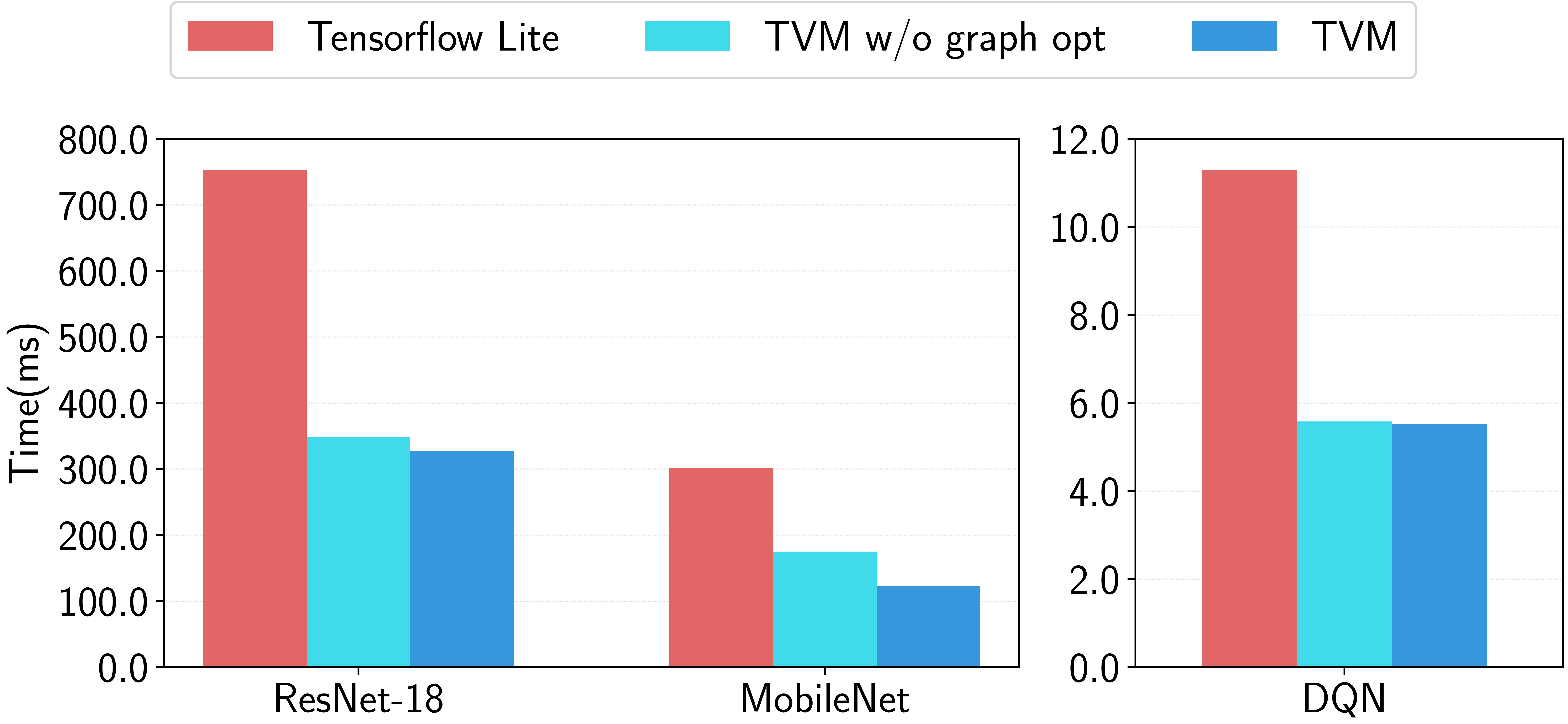
End to End Inference Performance (Nvidia Titan X)



Competitive on standard models

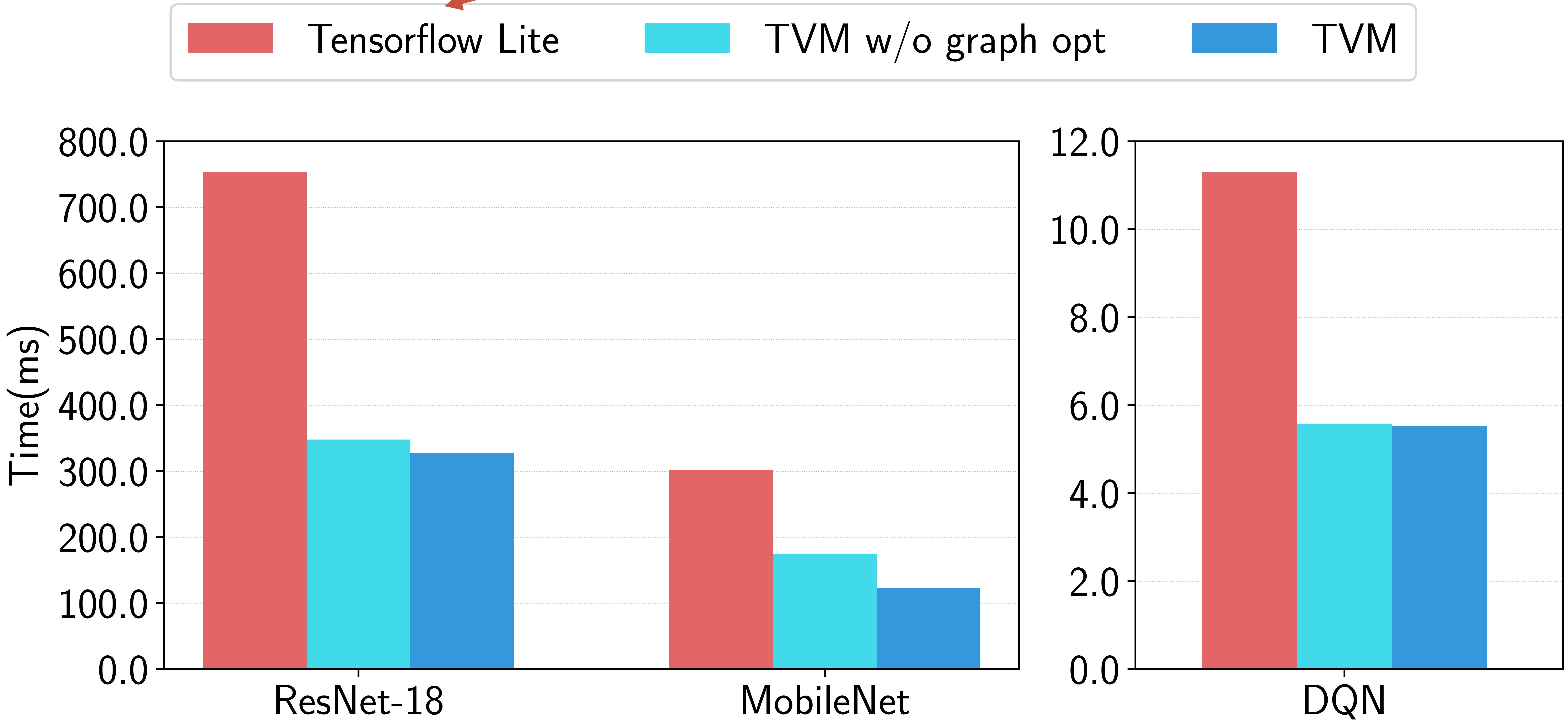
Bigger gap on less conventional models

End to End Performance(ARM Cortex-A53)

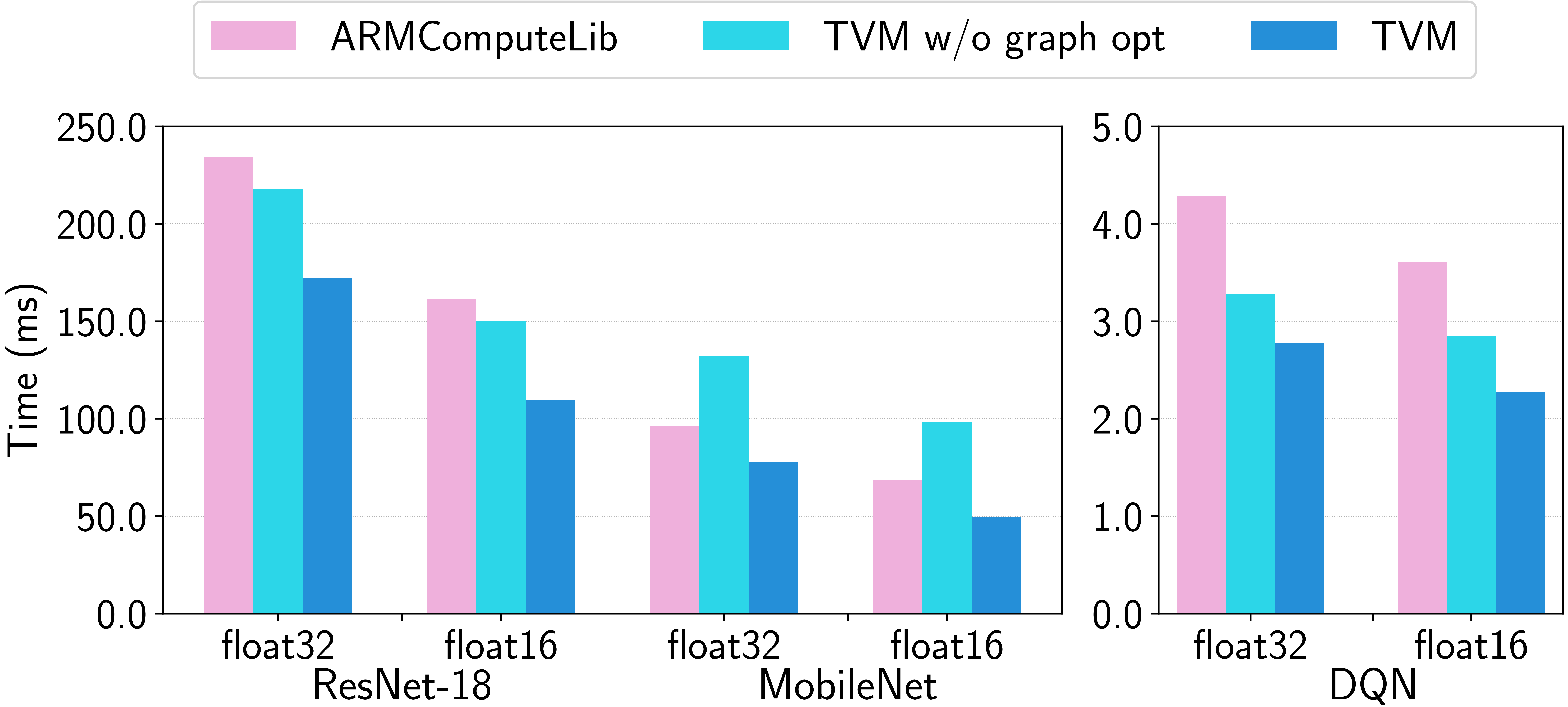


End to End Performance(ARM Cortex-A53)

Specially optimized for
Embedded system(ARM)

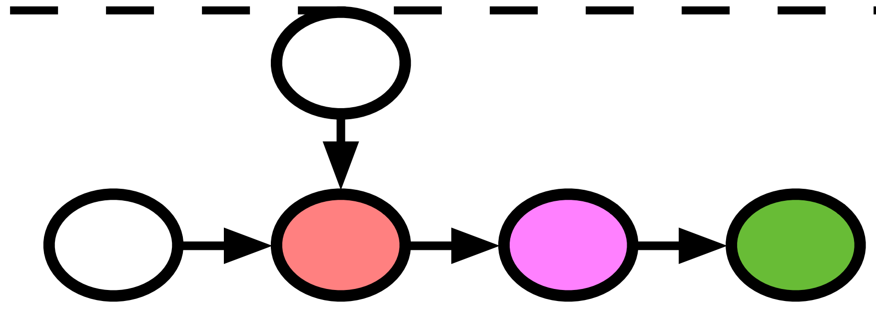


End to End Performance(ARM GPU)



Supporting New Specialized Accelerators

Frameworks



High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

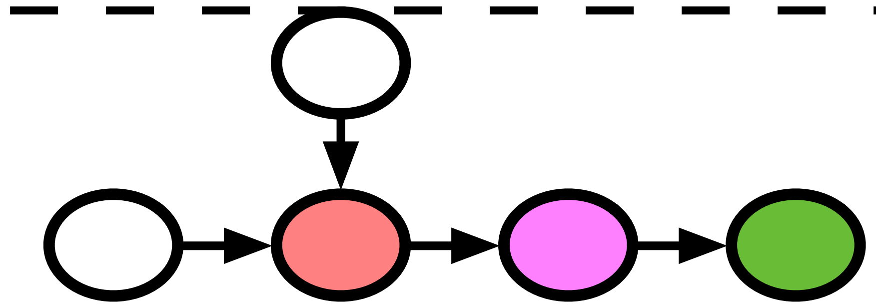
Machine Learning based Program Optimizer

LLVM

CUDA

Supporting New Specialized Accelerators

Frameworks

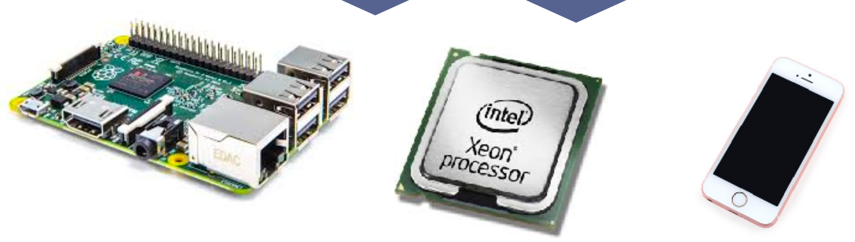


High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

Machine Learning based Program Optimizer

LLVM

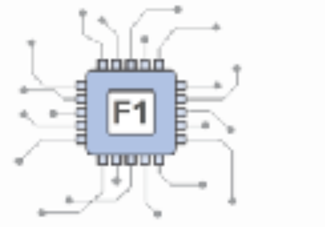


CUDA

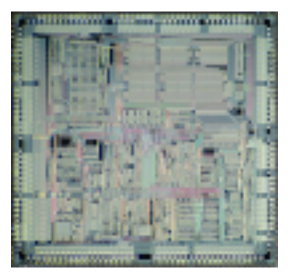
VTA: Open, Customizable Deep Learning Accelerator

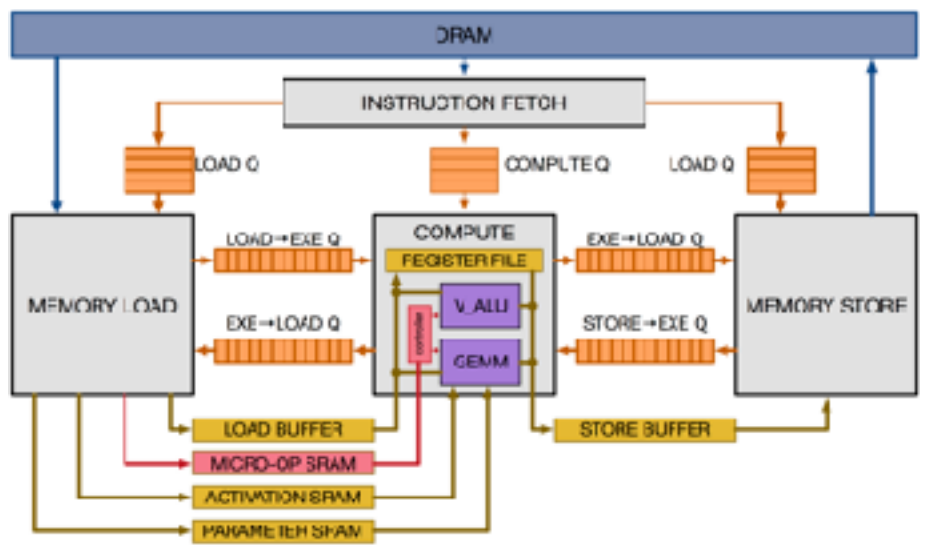
Edge FPGA

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

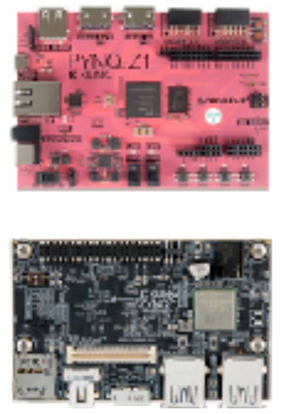
Tensor Program Search Space

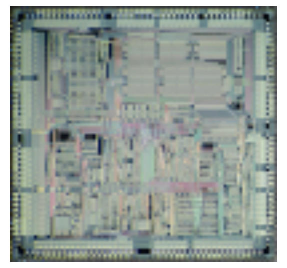
ML-based Optimizer

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

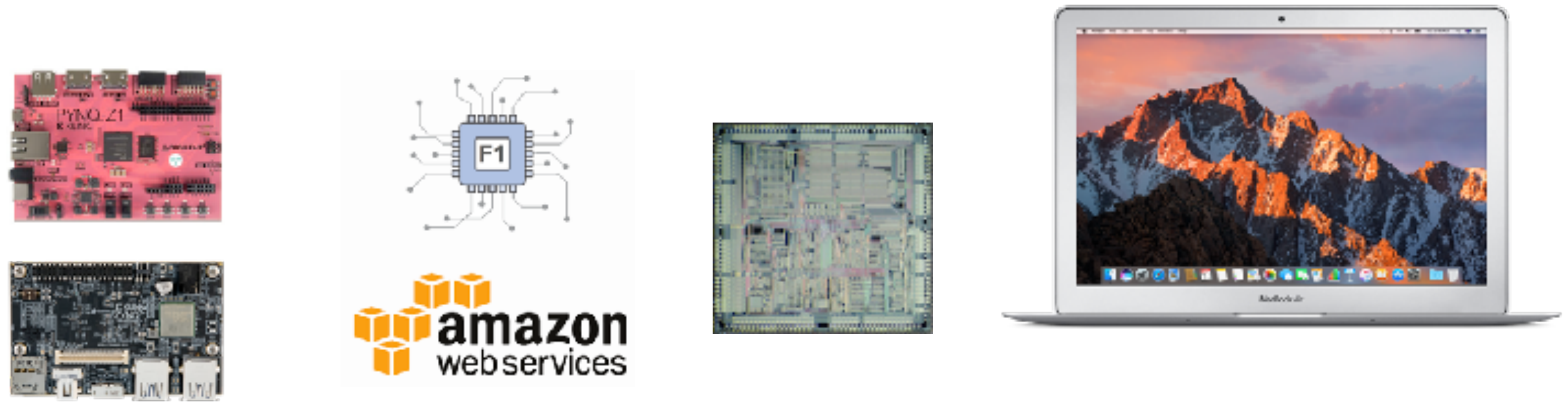
ML-based Optimizer

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

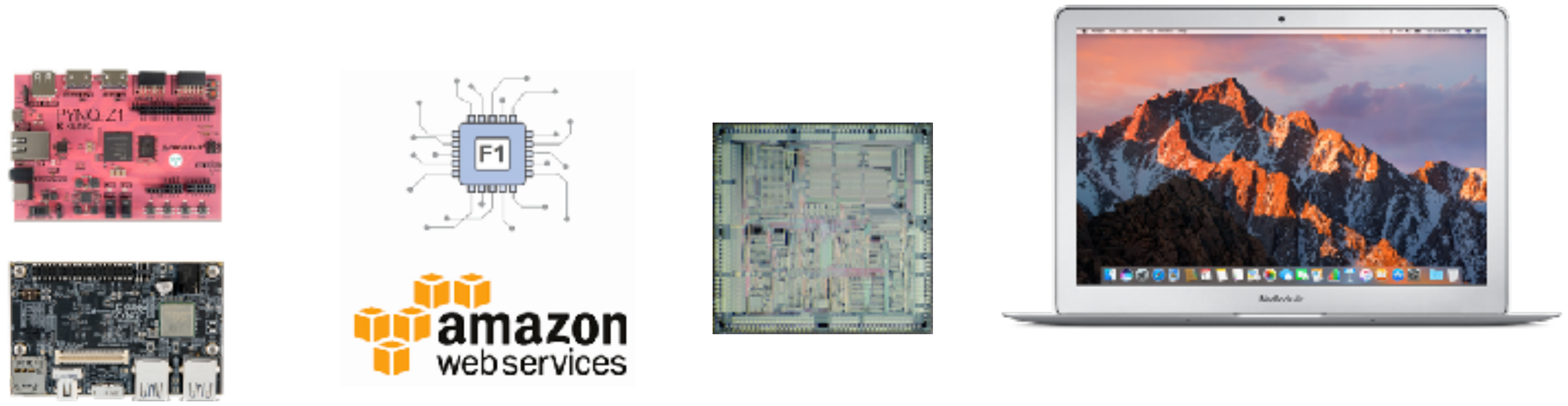
Tensor Program Search Space

ML-based Optimizer

VTA Runtime & JIT Compiler

VTA Hardware/Software Interface (ISA)

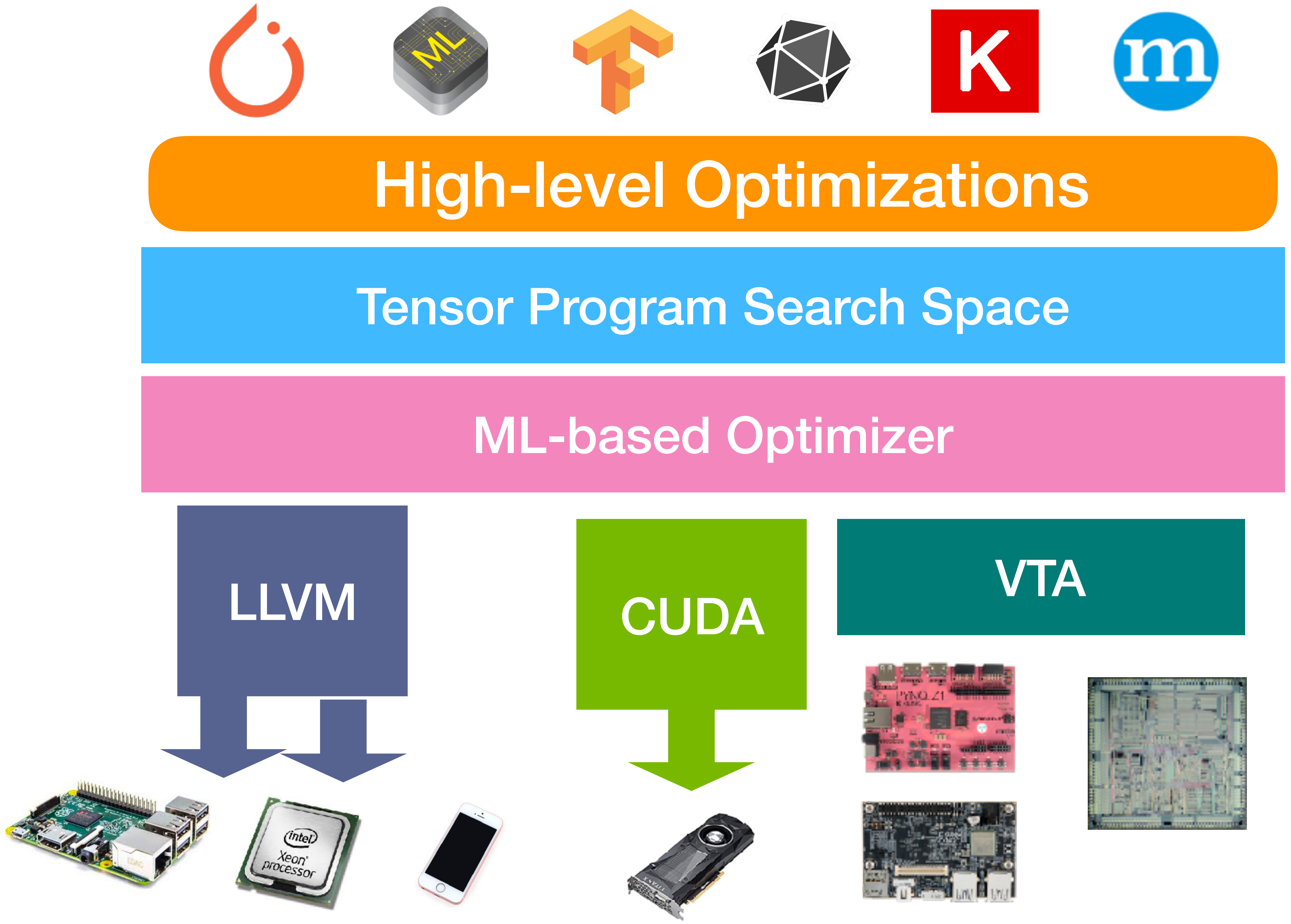
VTA MicroArchitecture | VTA Simulator



- 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

TVM: Learning-based Learning System



Check it out!

tvm.ai