PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

TVM Stack: End to End Optimization for Deep Learning

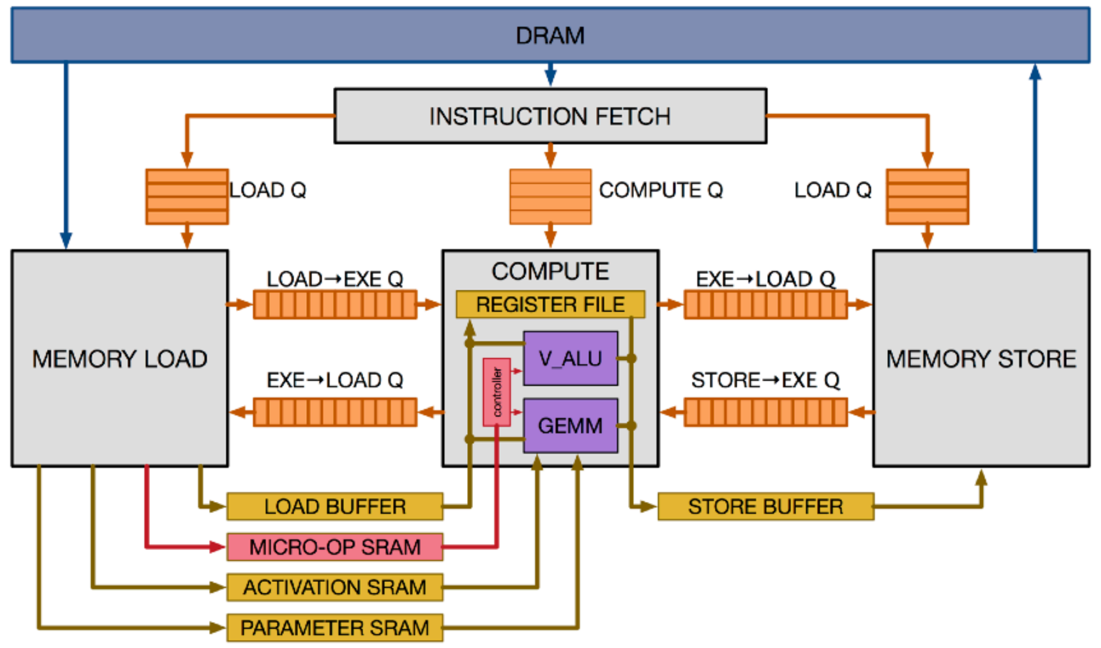
Presenter: Tianqi Chen

Paul G. Allen School of Computer Science & Engineering
University of Washington

Accelerator is more than the Hardware

Frameworks Caffe2 CNTK

But need to rebuilt entire software stack on top of it!



DRAM

INSTRUCTION FETCH

LOAD Q

COMPUTE Q

LOAD Q

MEMORY LOAD

LOAD→EXE Q

EXE→LOAD Q

COMPUTE

REGISTER FILE

V_ALU

GEMM

controller

EXE→LOAD Q

STORE→EXE Q

MEMORY STORE

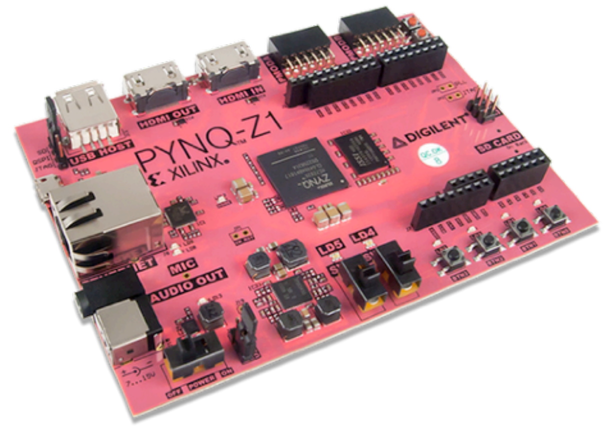
LOAD BUFFER

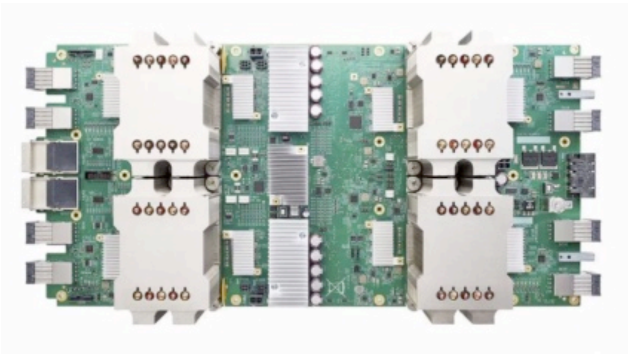
STORE BUFFER

MICRO-OP SRAM

ACTIVATION SRAM

PARAMETER SRAM





We can build new accelerators

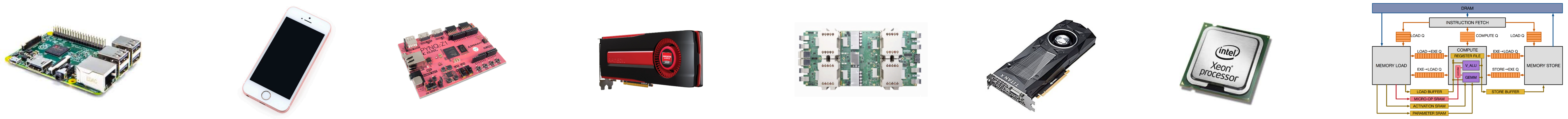
TVM: End to End Optimization Stack

Frameworks Caffe2 CNTK CoreML

Pytorch, caffe2, cntk supported via onnx

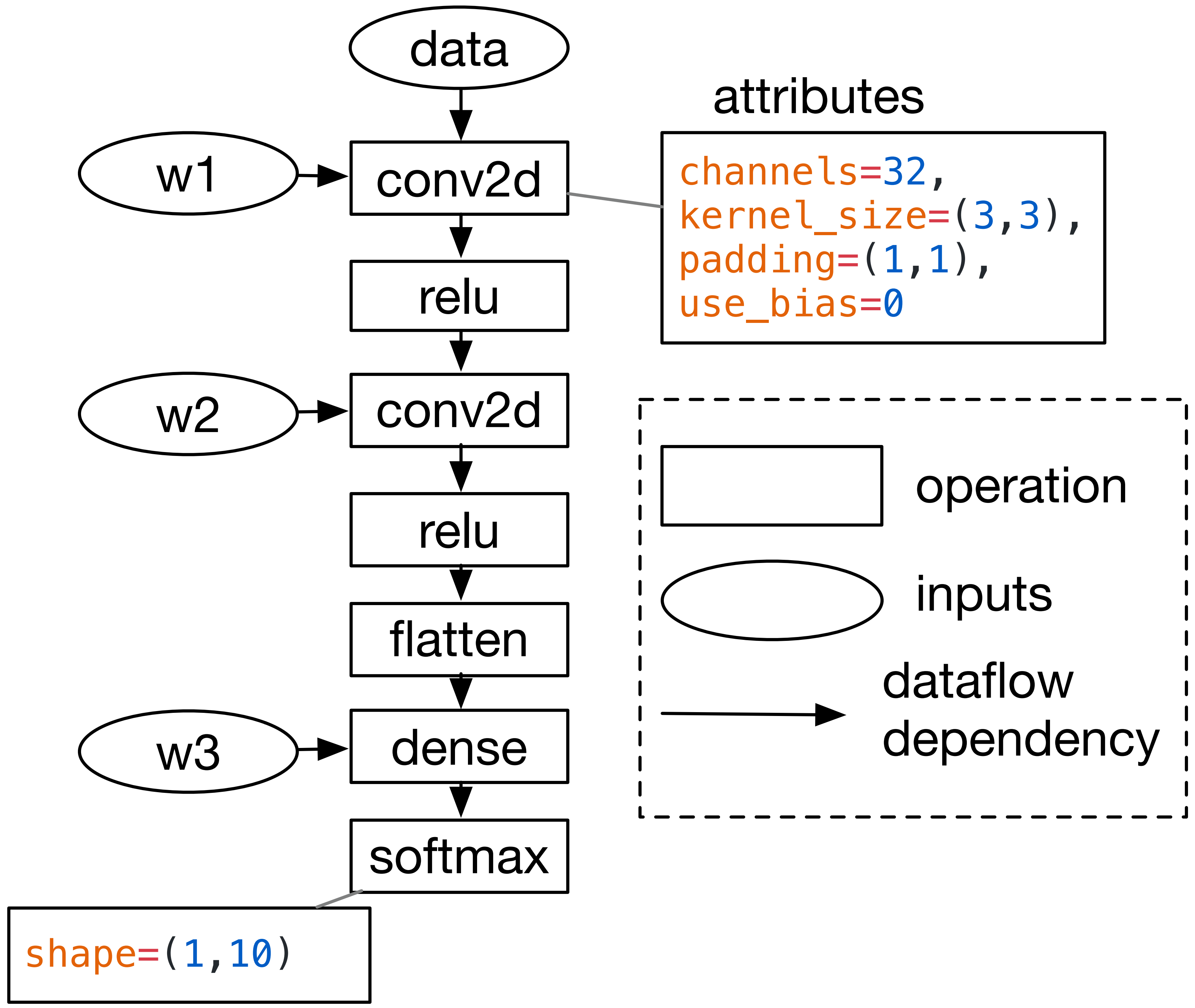
Computational Graph Optimization

Hardware

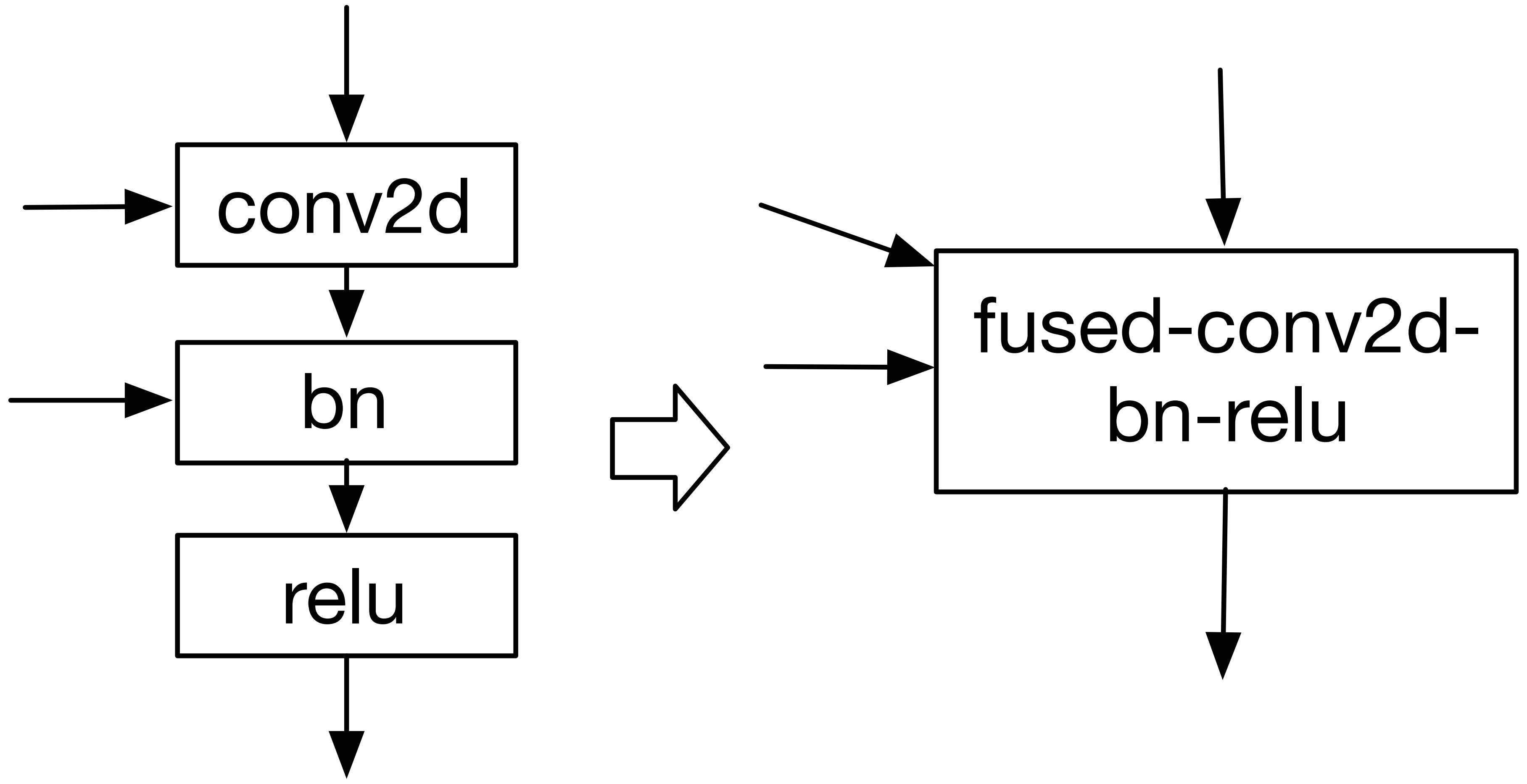


Computational Graph Optimization

Represent High level Deep Learning Computations



Effective Equivalent Transformations to Optimize the Graph



Similar approach used by TensorFlow XLA, NGraph …

TVM: End to End Optimization Stack

Frameworks Caffe2 CNTK CoreML K

Pytorch, caffe2, cntk supported via onnx

Computational Graph Optimization

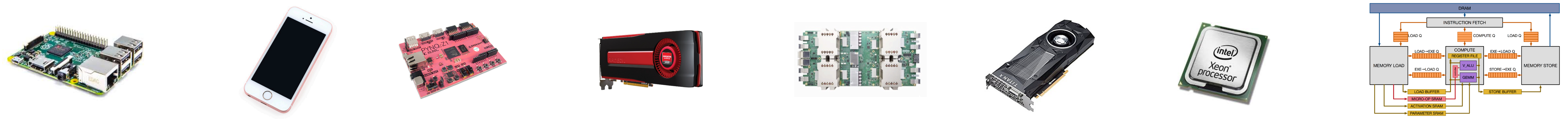
Memory Plan

Operator Fusion

Data Layout Transform ...

huge gap remains: need to build and optimize operators for each hardware, variant of layout, precision, threading pattern …

Hardware



TVM: End to End Optimization Stack

Frameworks Caffe2 CNTK CoreML

Pytorch, caffe2, cntk supported via onnx

Computational Graph Optimization

Tensor Expression Language

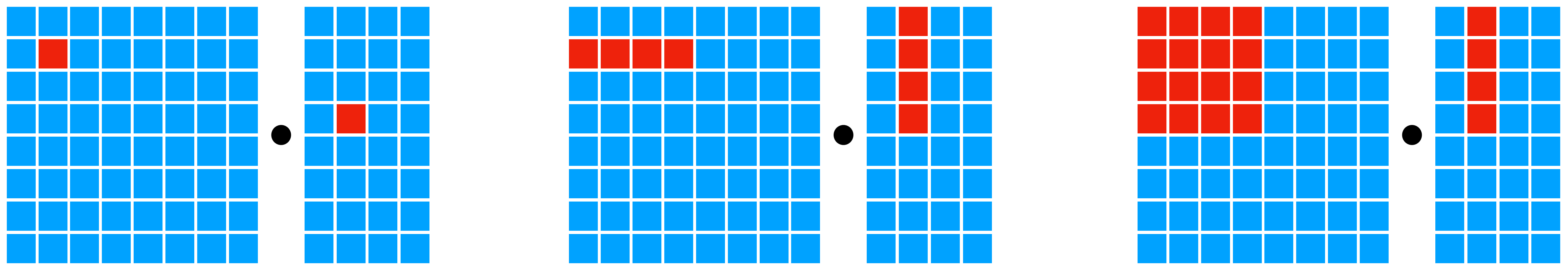
```
C = tvm.compute((m, n),
   lambda i, j: tvm.sum(A[i, k] * B[j, k], axis=k))
```

Schedule Optimizations

Hardware

Tensorization Challenge

Compute primitives



scalar

vector

tensor

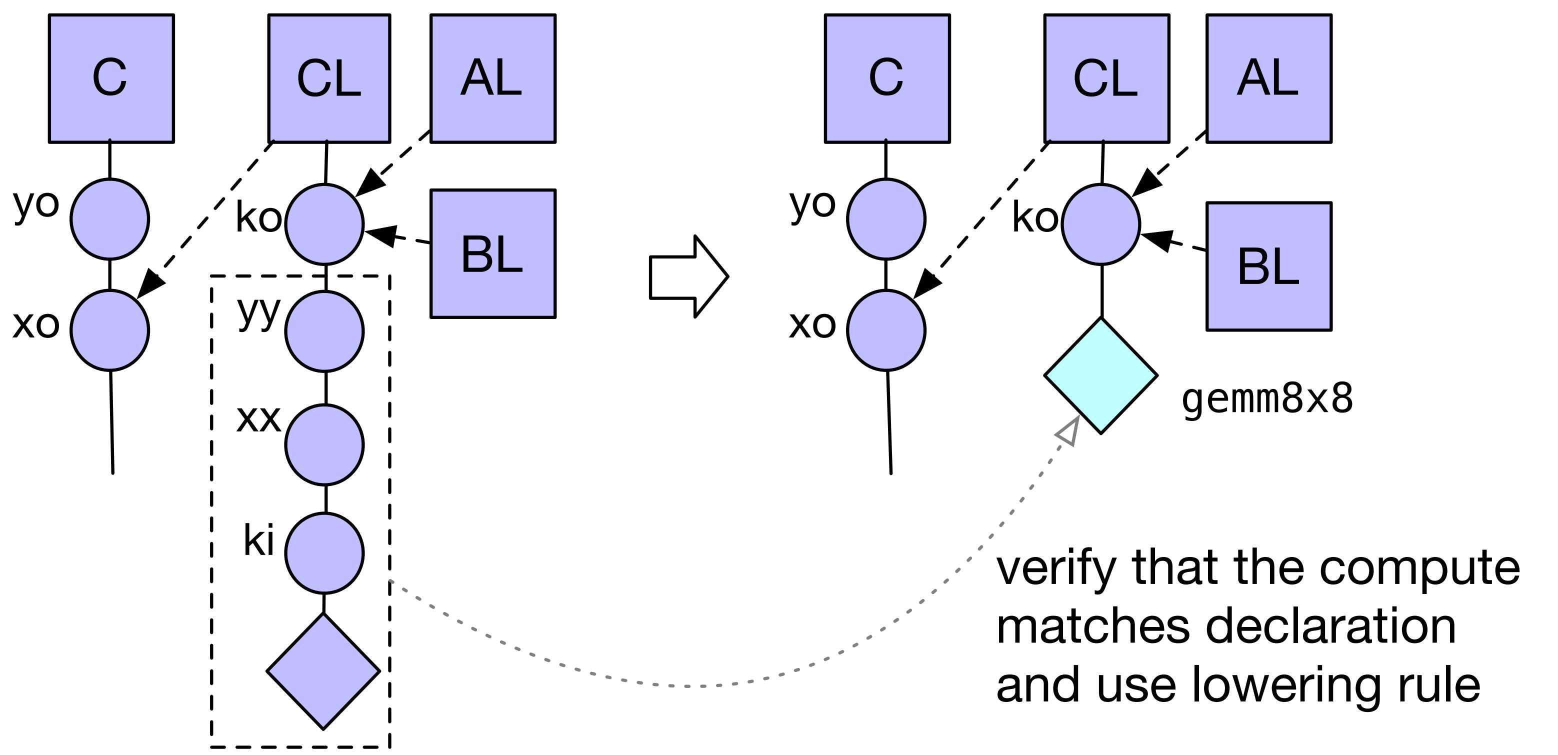
Hardware designer: declare tensor instruction interface

```
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

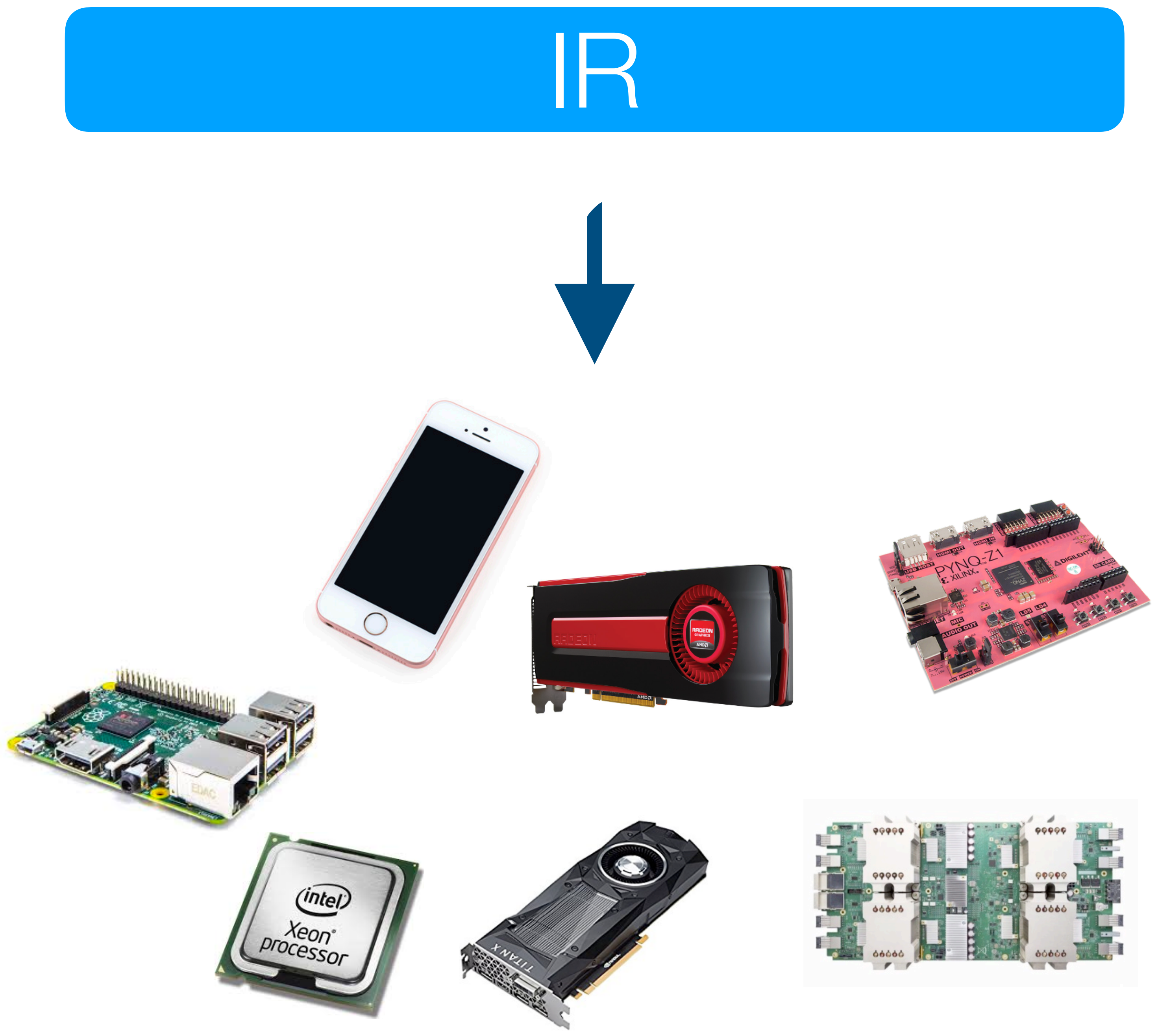
lowering rule to generate hardware intrinsics to carry out the computation

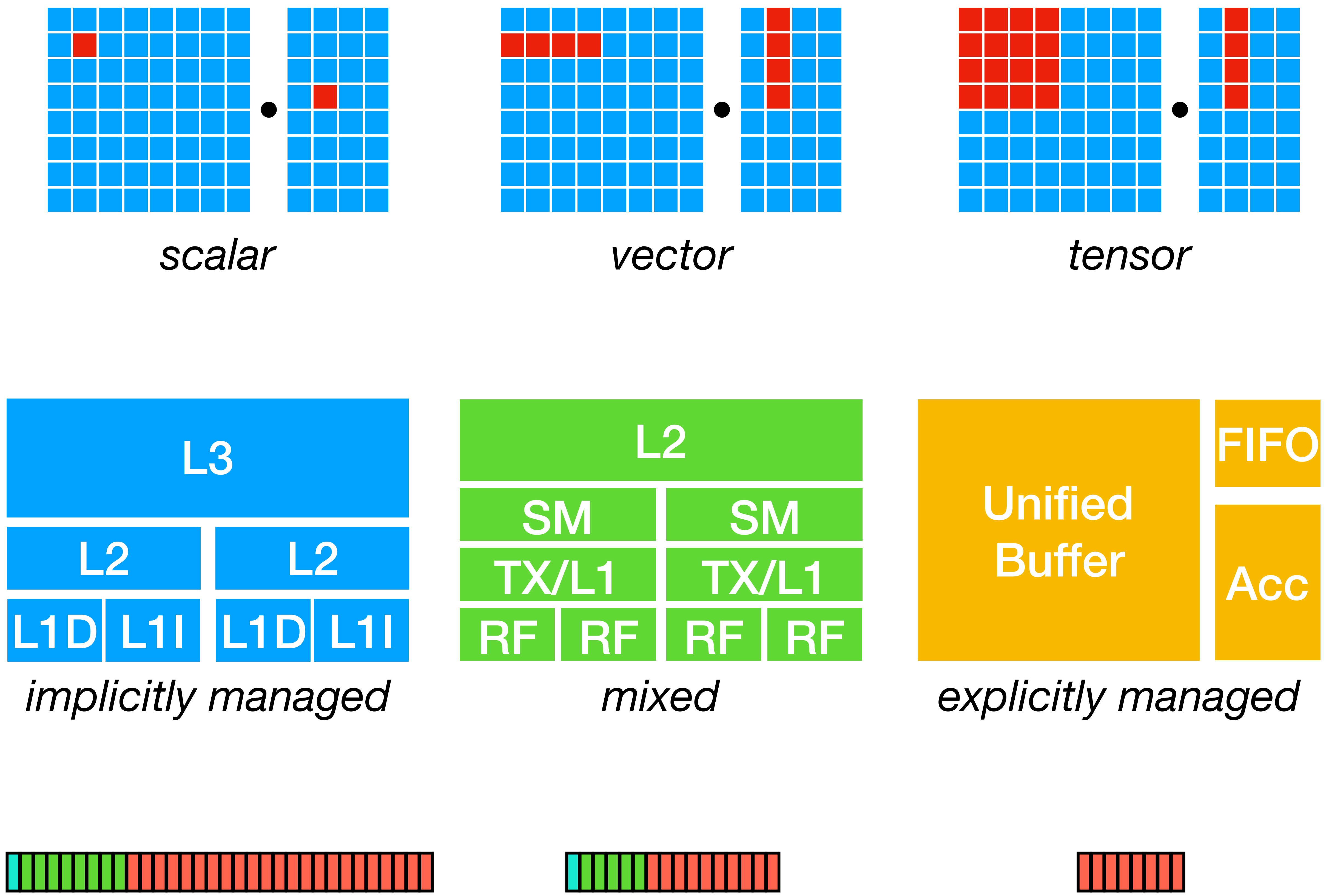
Tensorize: transform program to use tensor instructions



verify that the compute matches declaration and use lowering rule

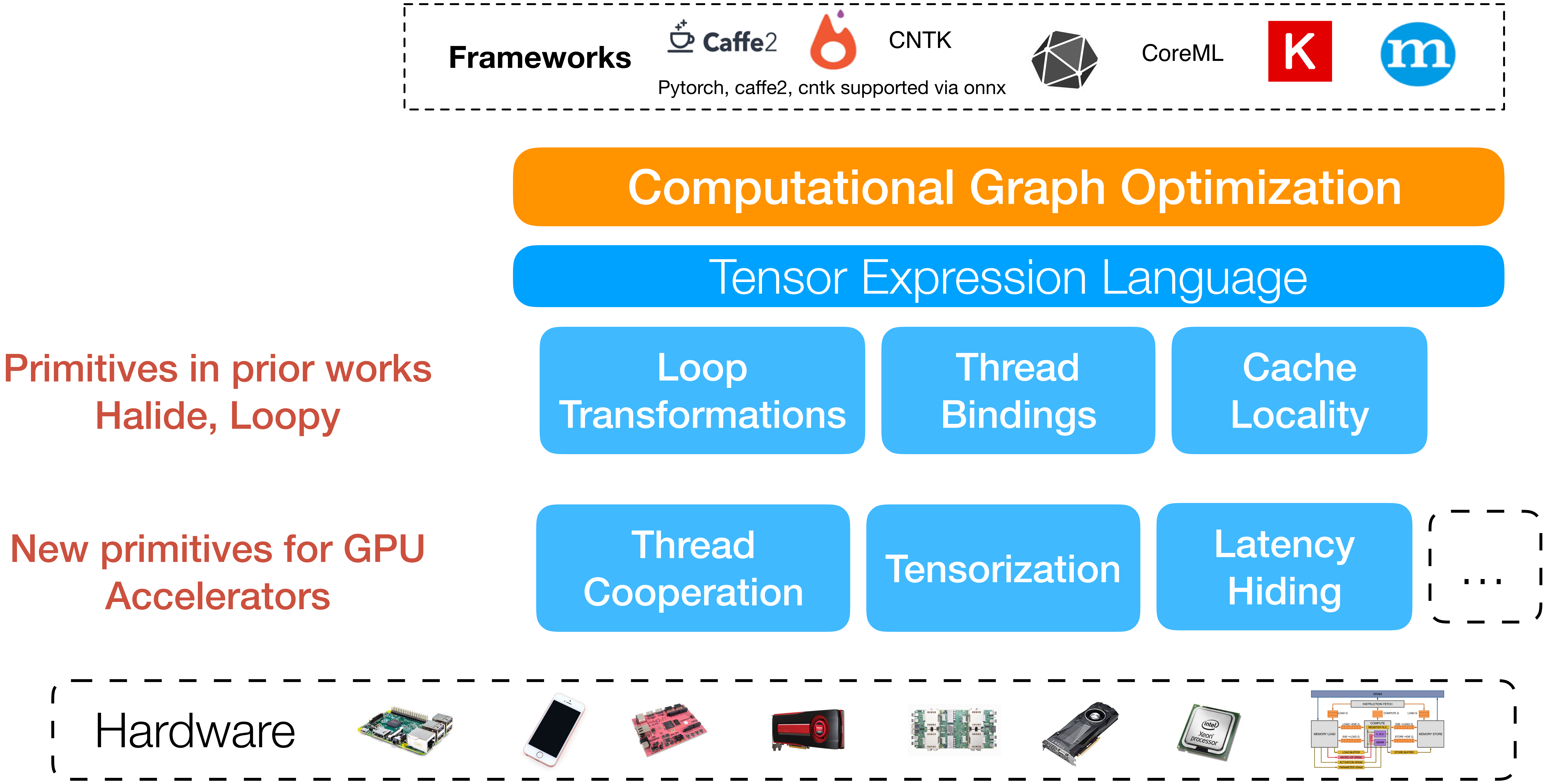
More Hardware Challenges





	CPU	GPU	Accelerators
Compute primitives	*scalar*	*vector*	*tensor*
Memory subsystem	*implicitly managed*	*mixed*	*explicitly managed*
Data type	`fp32`	`fp16`	`int8`

TVM: End to End Optimization Stack



Frameworks

Caffe2

CNTK

CoreML

Pytorch, caffe2, cntk supported via onnx

Computational Graph Optimization

Tensor Expression Language

Primitives in prior works
Halide, Loopy

Loop Transformations

Thread Bindings

Cache Locality

New primitives for GPU
Accelerators

Thread Cooperation

Tensorization

Latency Hiding

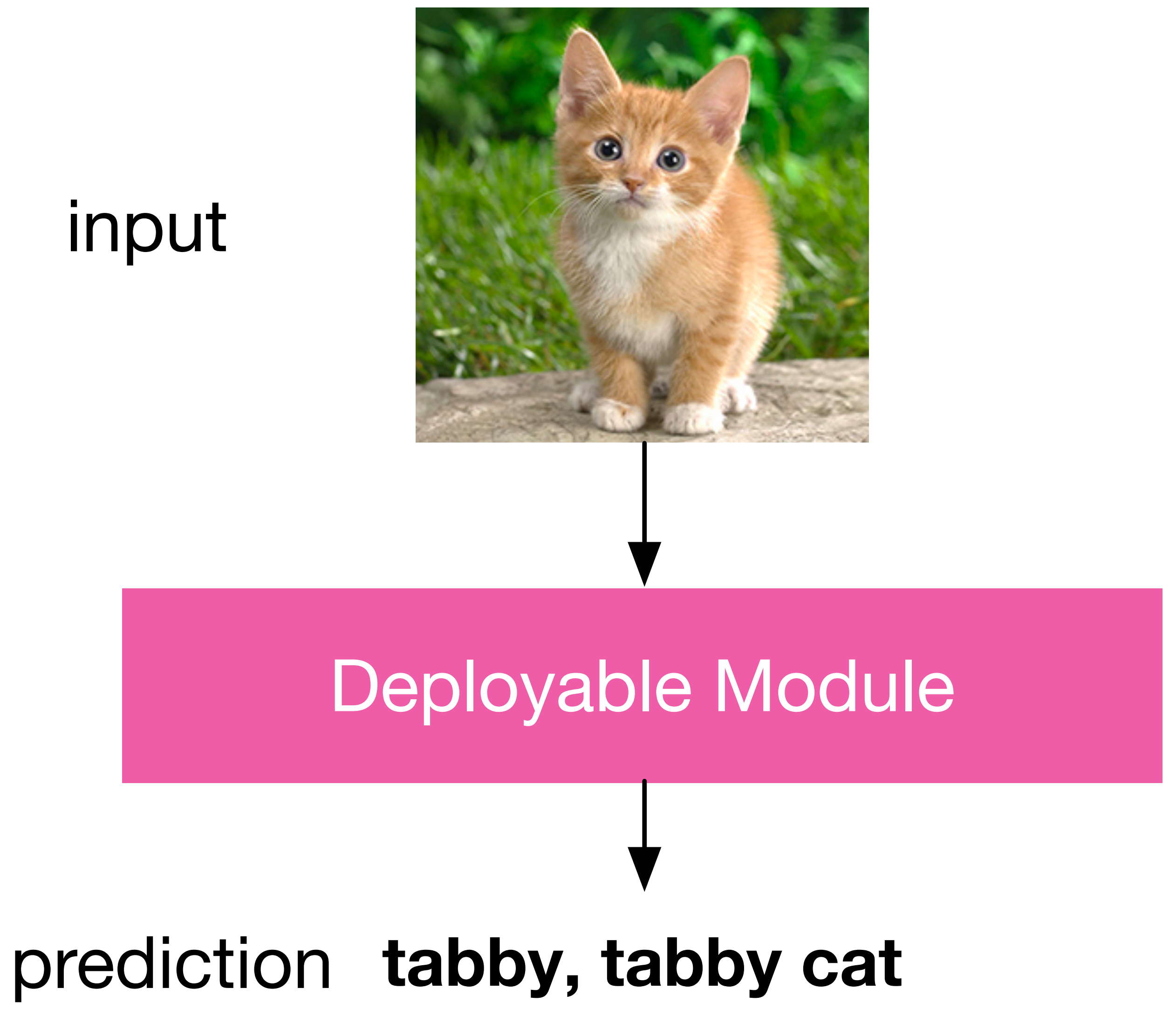
…

Hardware

Model in, Deployable Module Out

```
import tvm
import nnvm.frontend
import nnvm.compiler

graph, params =
nnvm.frontend.from_mxnet(mx_resnet50)
graph, lib, params =
     nnvm.compiler.build(graph, target)
```

```
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)
```



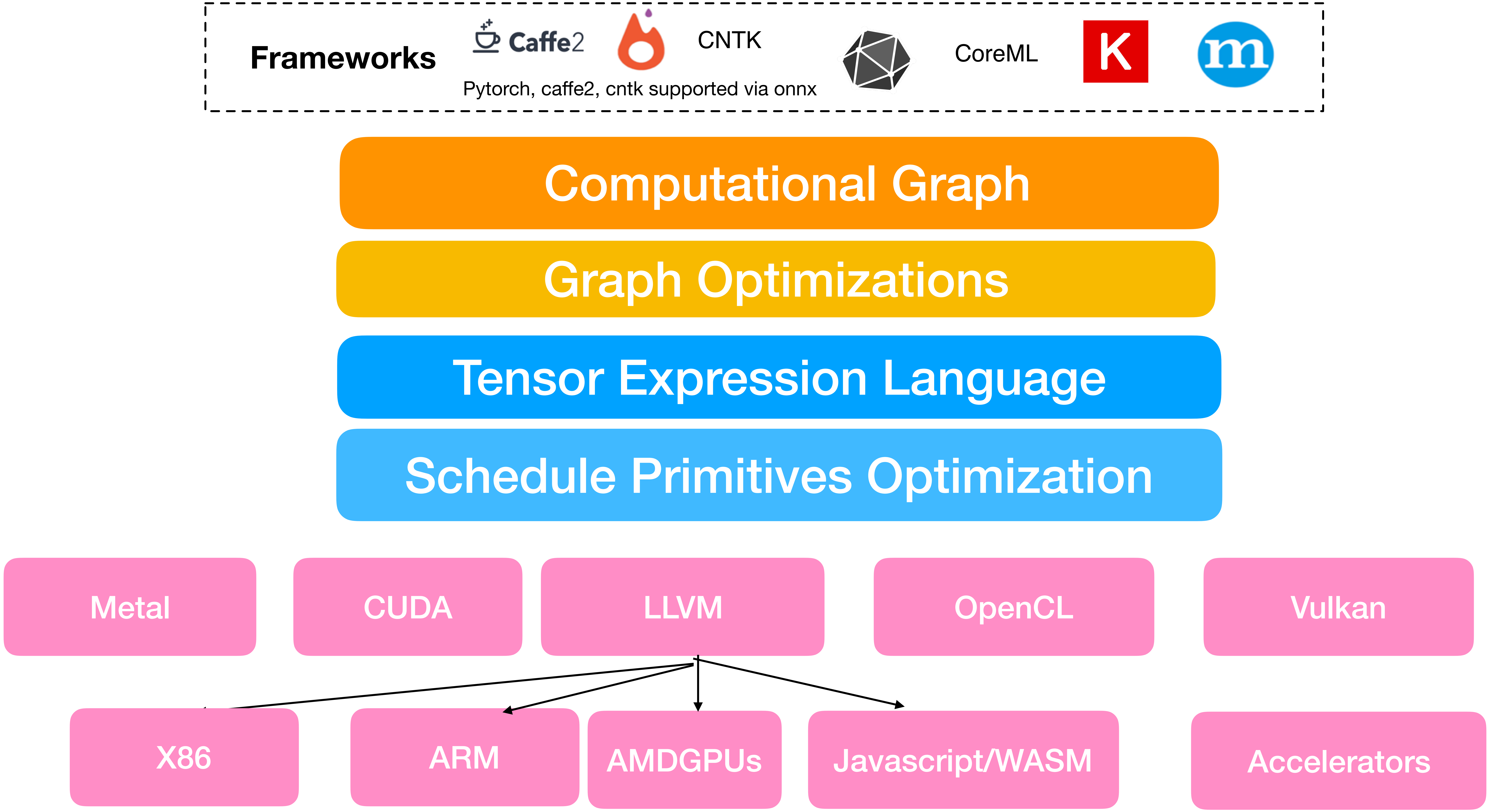
input

Deployable Module

prediction **tabby, tabby cat**

On languages and platforms you choose

TVM: End to End Stack For Deep Learning



Frameworks

Caffe2

CNTK

Pytorch, caffe2, cntk supported via onnx

CoreML

Computational Graph

Graph Optimizations

Tensor Expression Language

Schedule Primitives Optimization

Metal

CUDA

LLVM

OpenCL

Vulkan

X86

ARM

AMDGPUs

Javascript/WASM

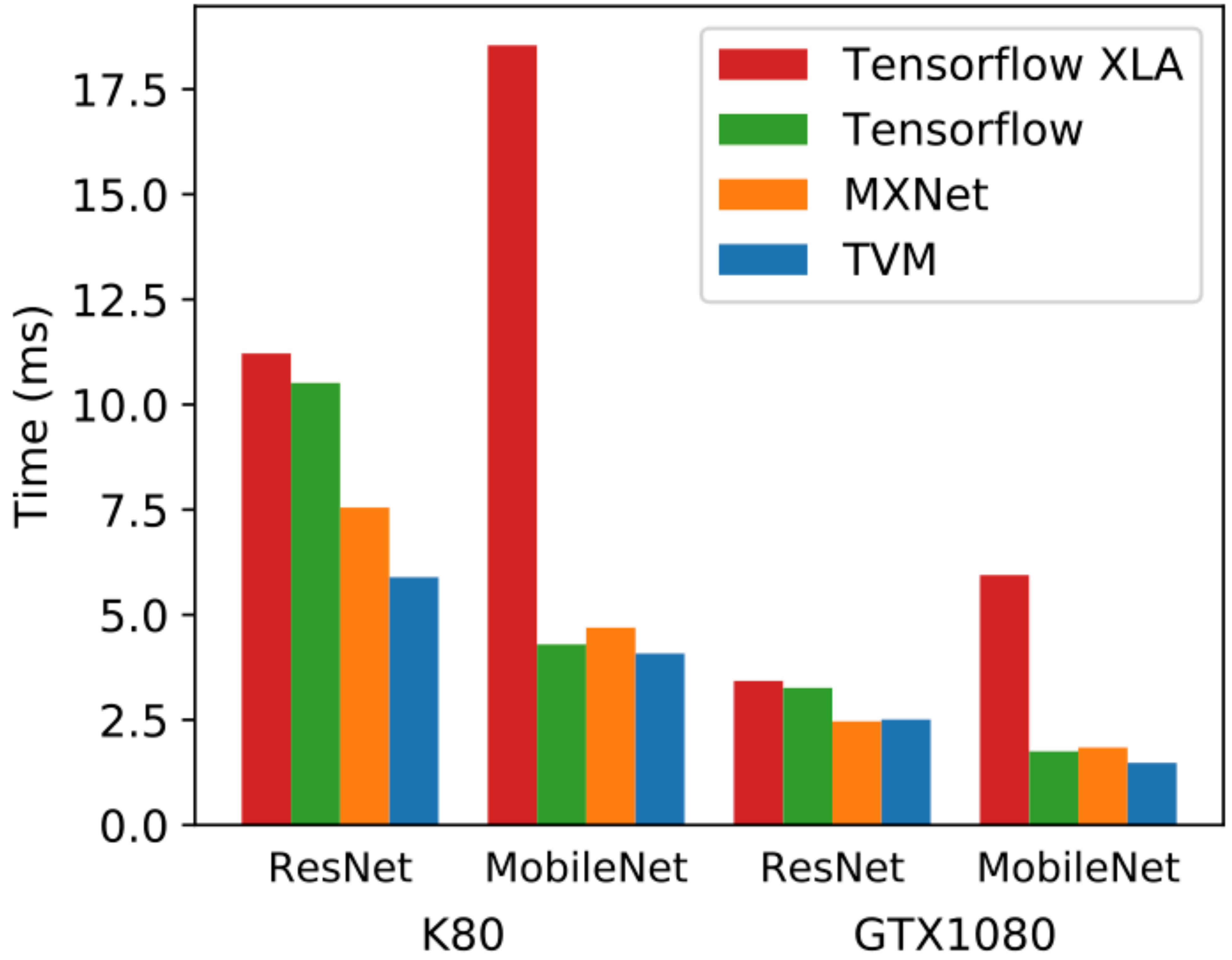
Accelerators

Experimental Results

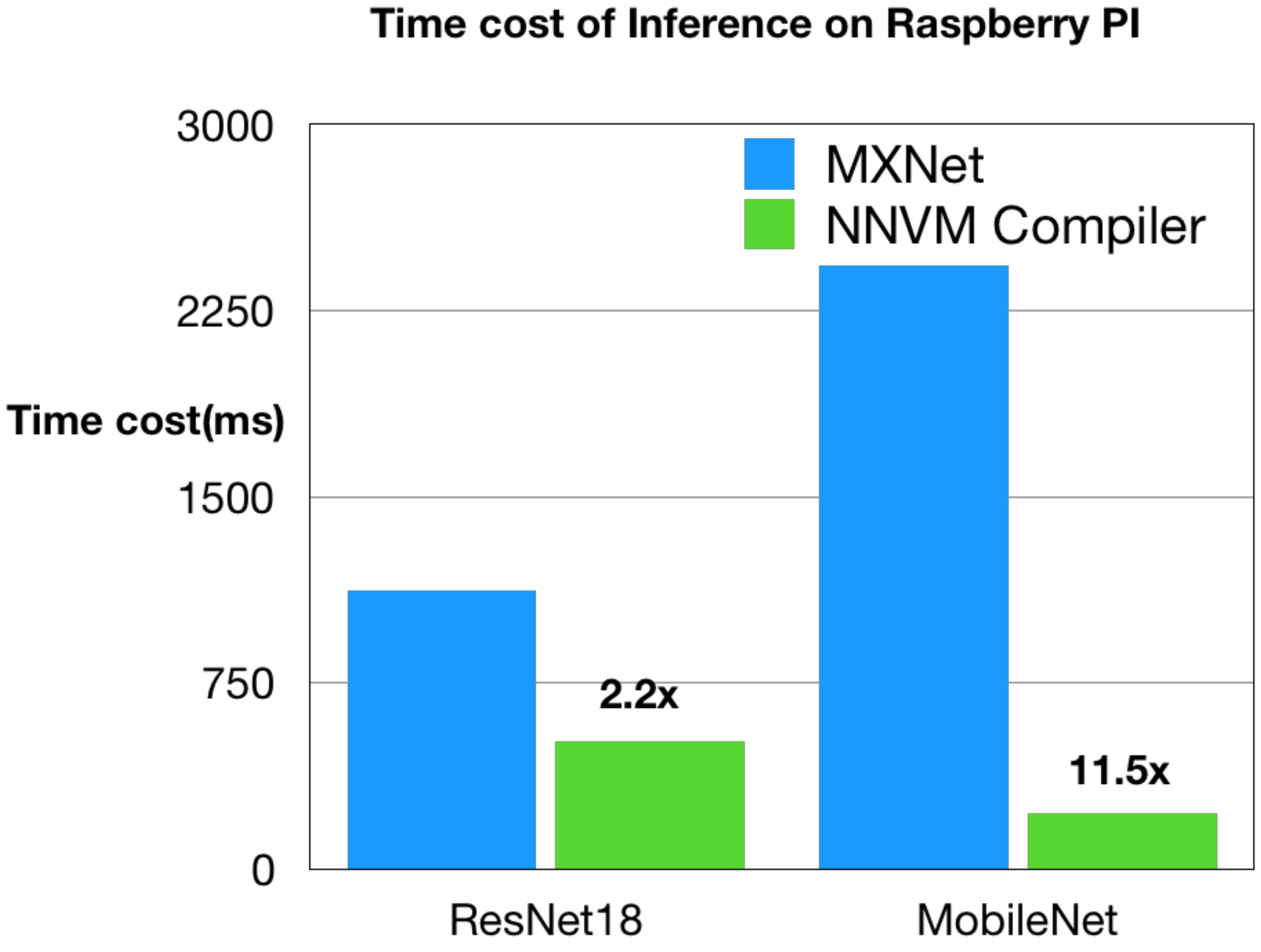
Compare TVM Stack solution to
Existing solutions which relies on manually optimized libraries

End to End Performance across Hardwares

Nvidia GPUs

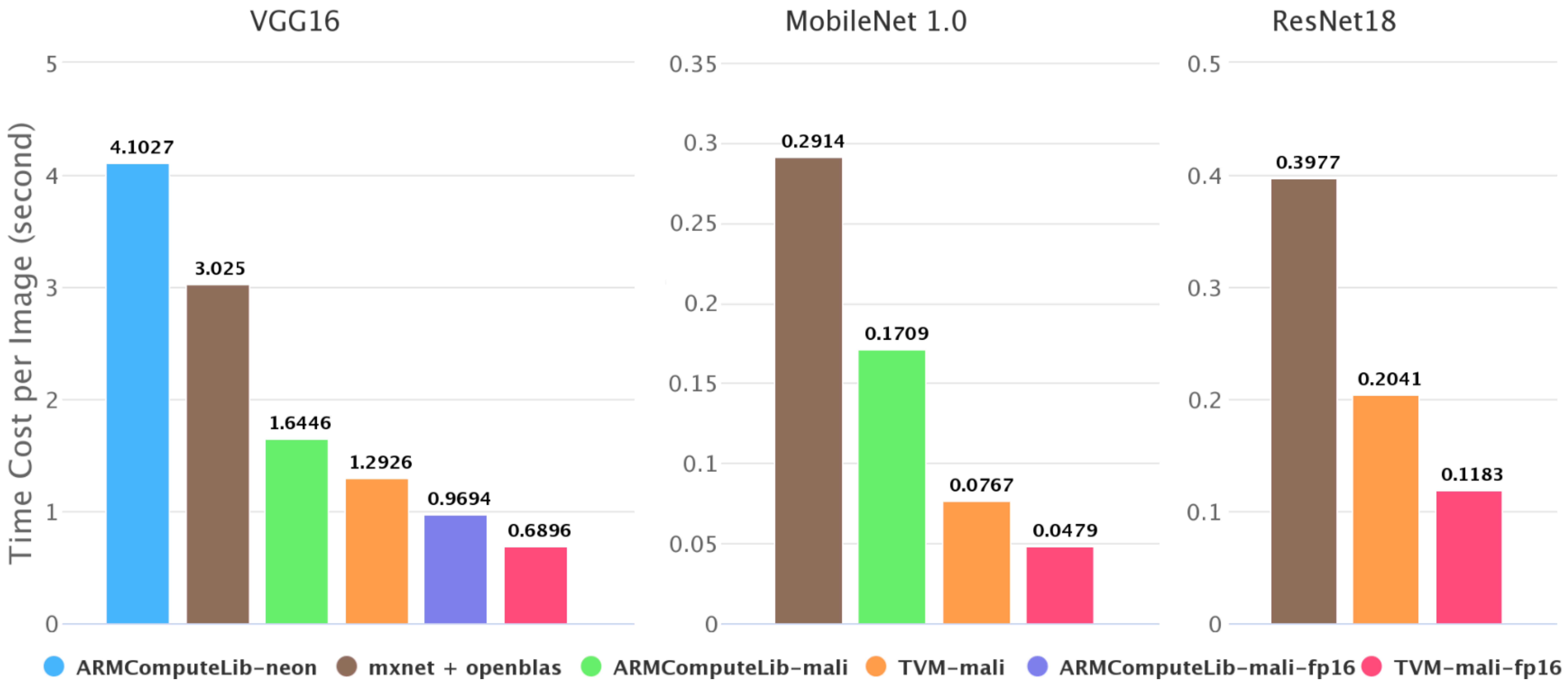


Raspberry Pis



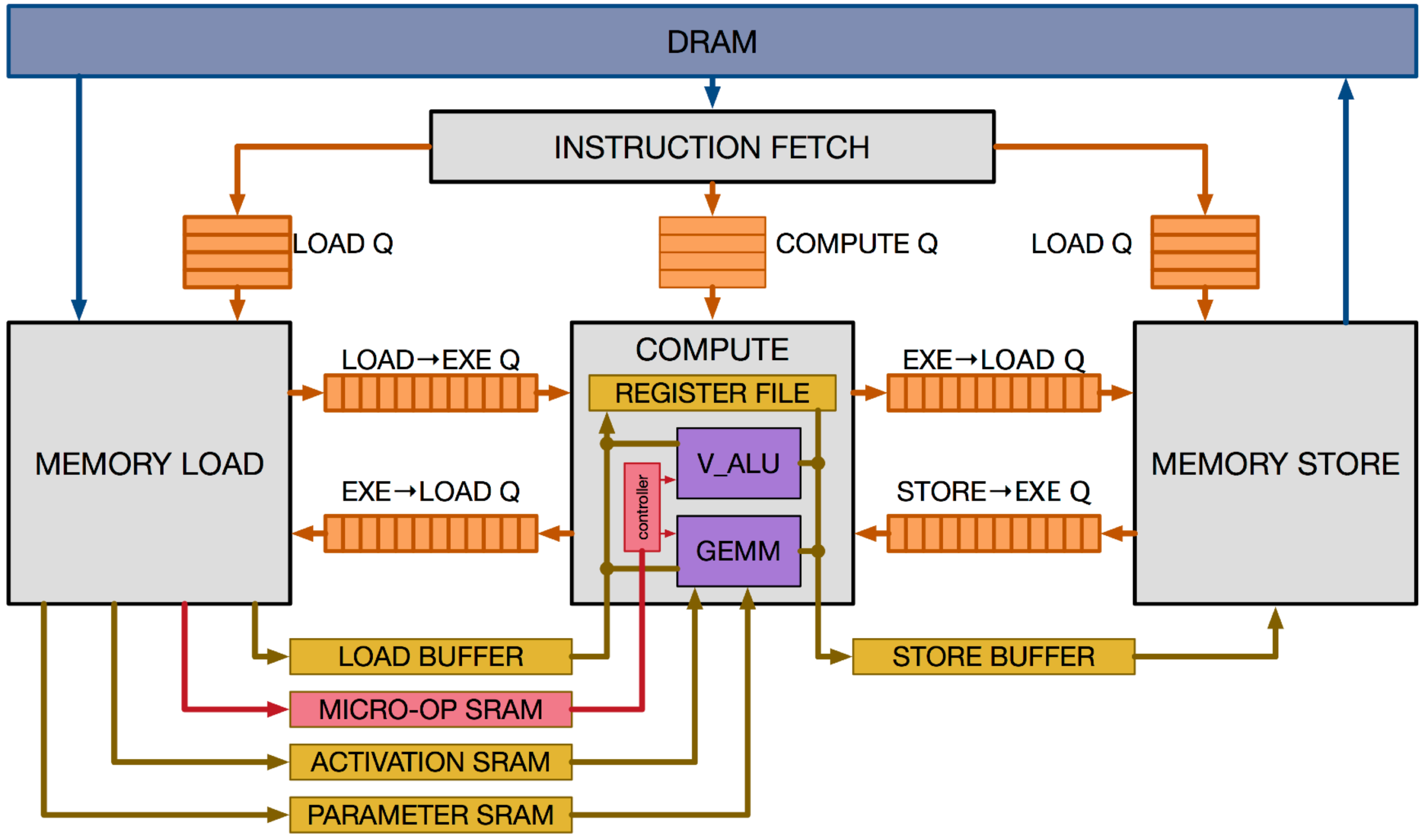
Credit: Leyuan Wang(AWS/UCDavis), Yuwei Hu(TuSimple), Zheng Jiang(AWS/FDU), Lianmin Zheng(SJTU)

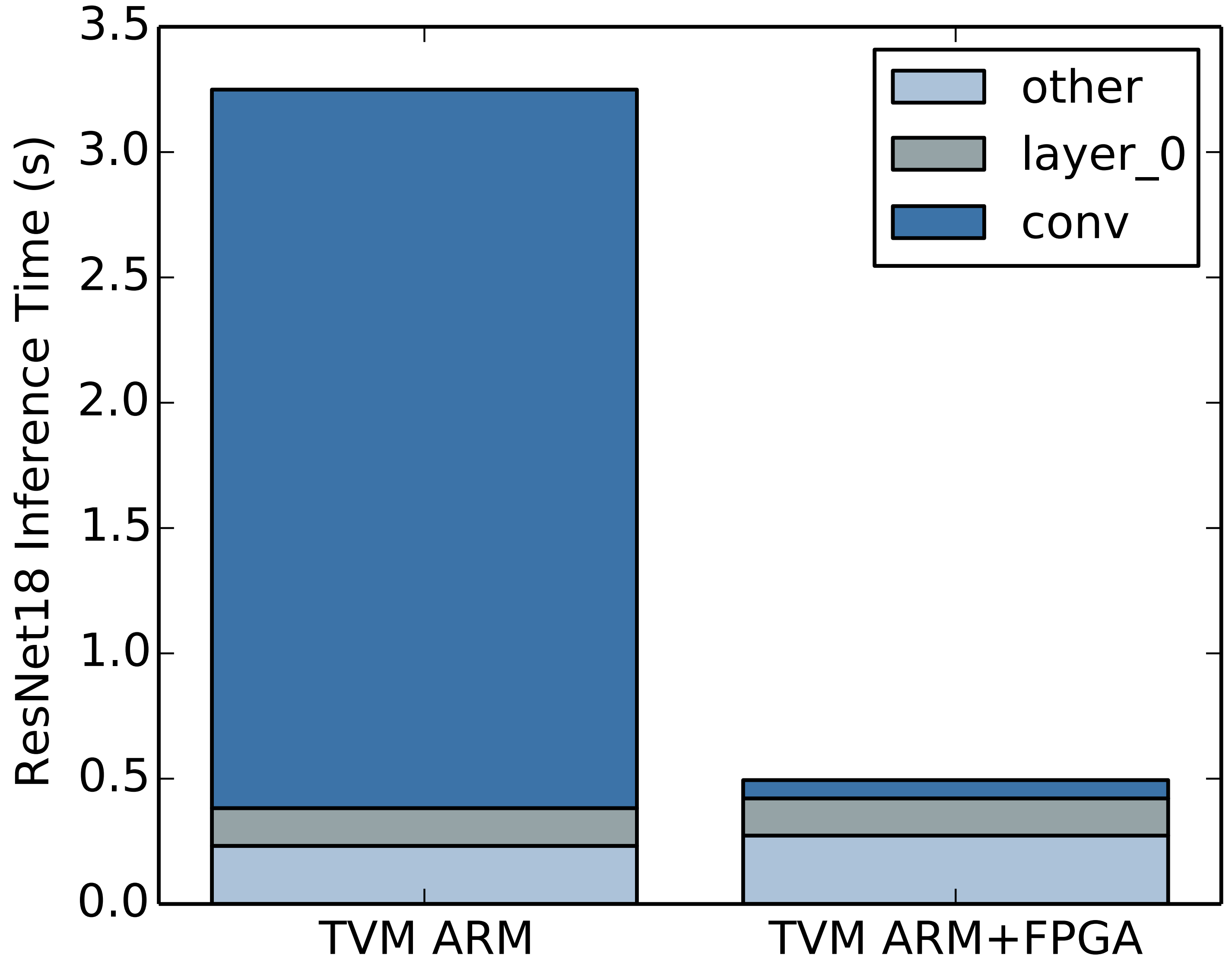
End to End Performance on Mobile GPUs(ARM Mali)

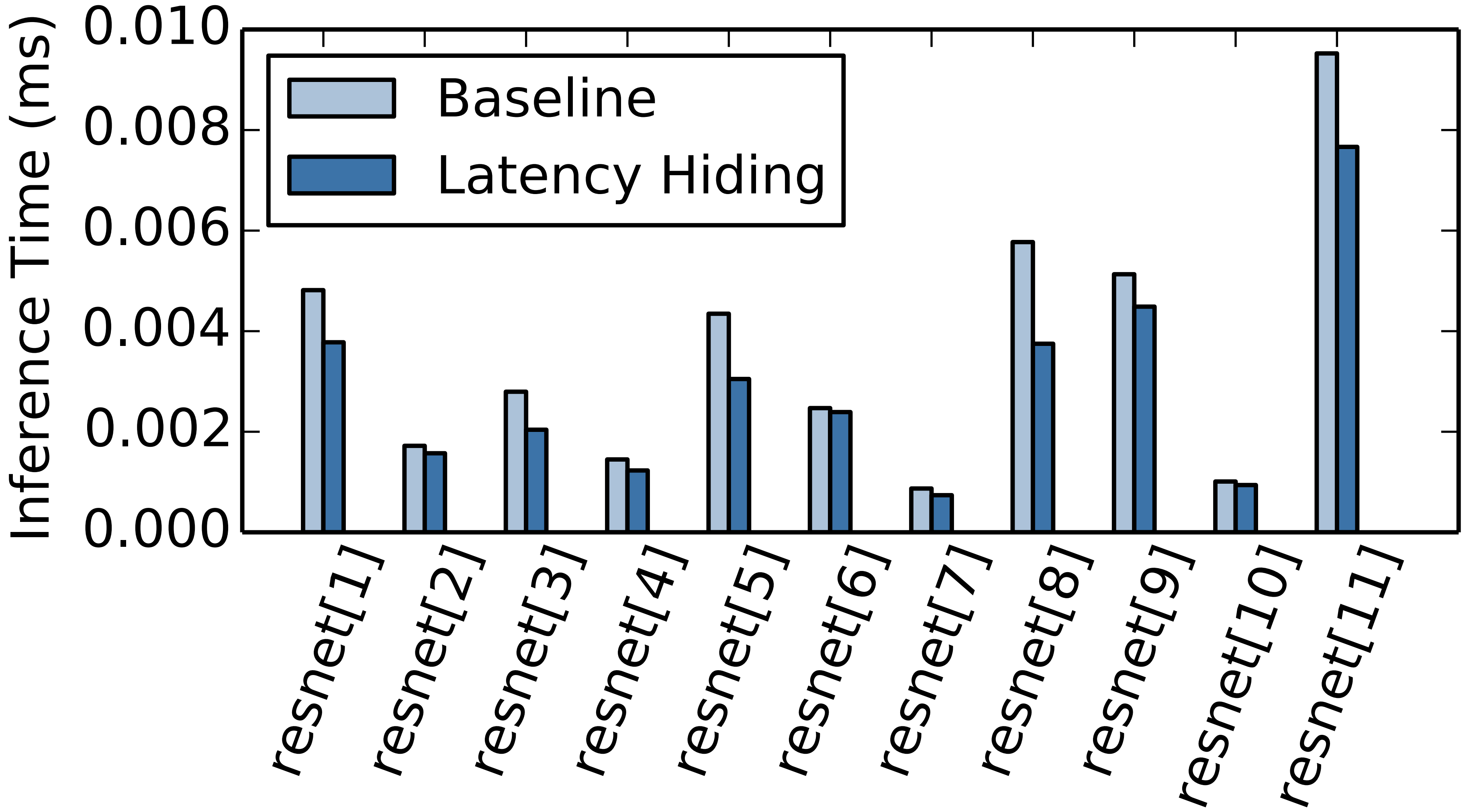


Credit: Lianmin Zheng(SJTU)

VTA: Support New Accelerators







TVM Stack + MXNet

Already supported:

MXNet TVMBridge: Customized operator specification

NNVM Compiler integration for hardware backends

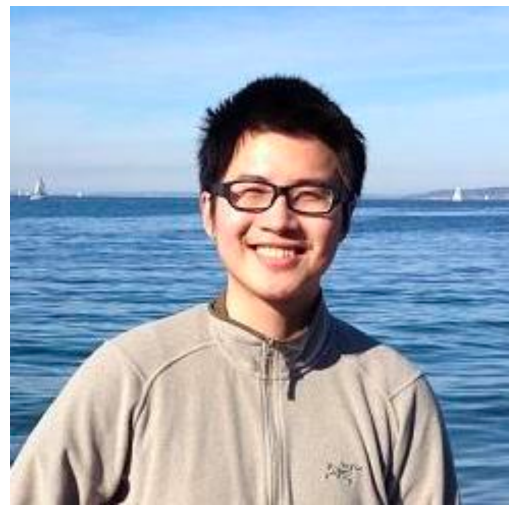
Soon:

More end to end performance improvements

VTA opensource

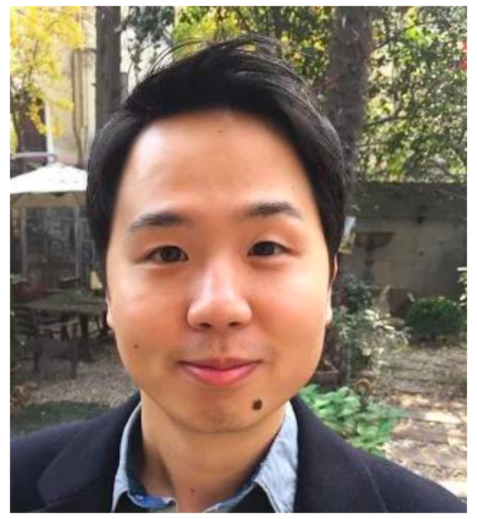
TVM Stack Collaborators

University of Washington

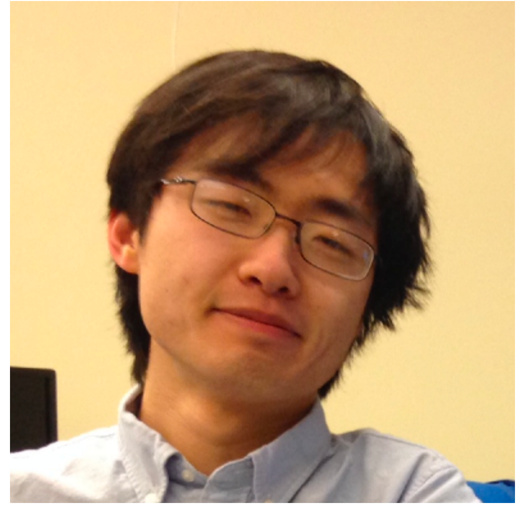


Tianqi Chen

Thierry Moreau

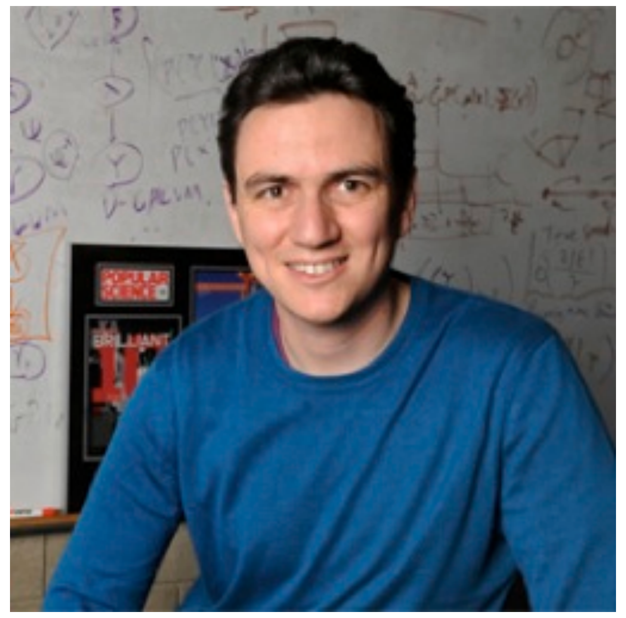


Haichen Shen



Eddie Yan

Meghan Cowan

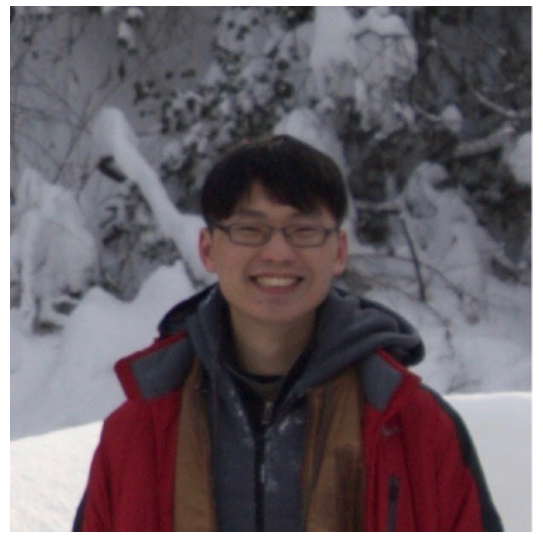


Carlos Guestrin

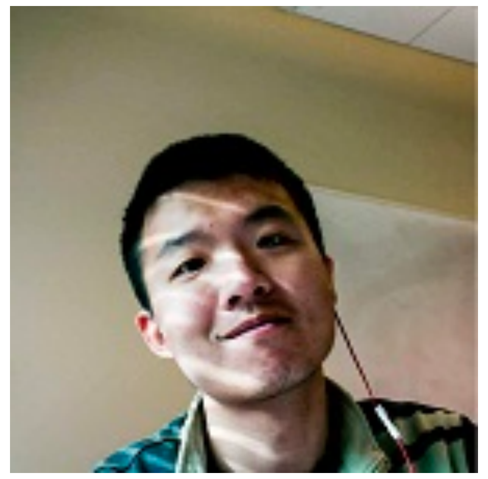
Luis Ceze

Arvind Krishnamurthy

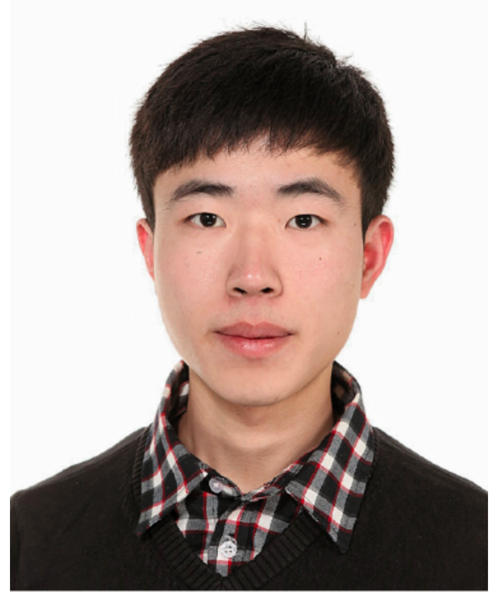
External Collaborators



Ziheng Jiang



Liang Luo



Lianmin Zheng

and many more contributors in the TVM open source community