Research on NNVM Compiler

USTC Compiler Team 13

Zhu Yiming, Xie Yunting, Lin Zhiqi

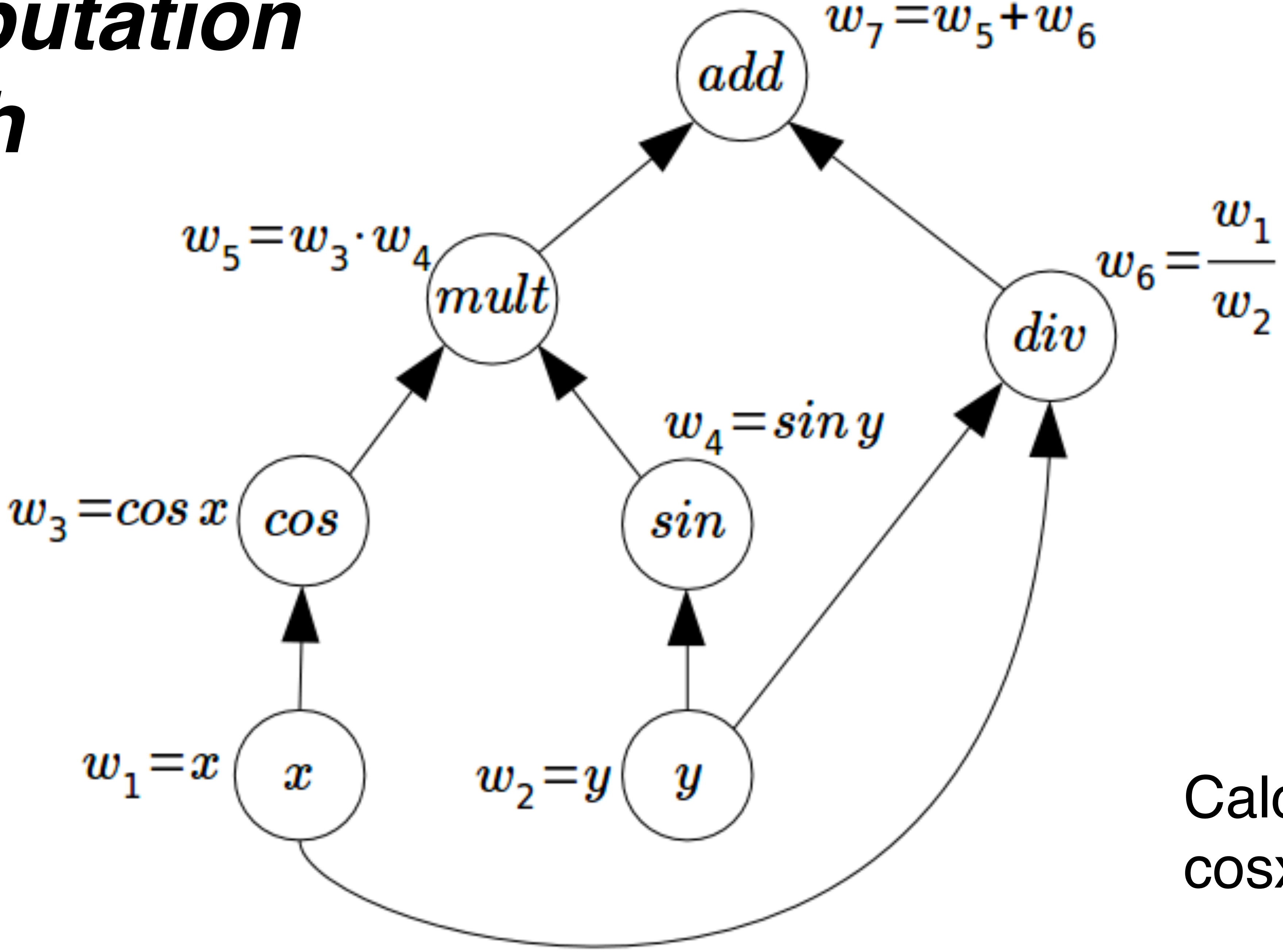
01

Part One

Background Introduce

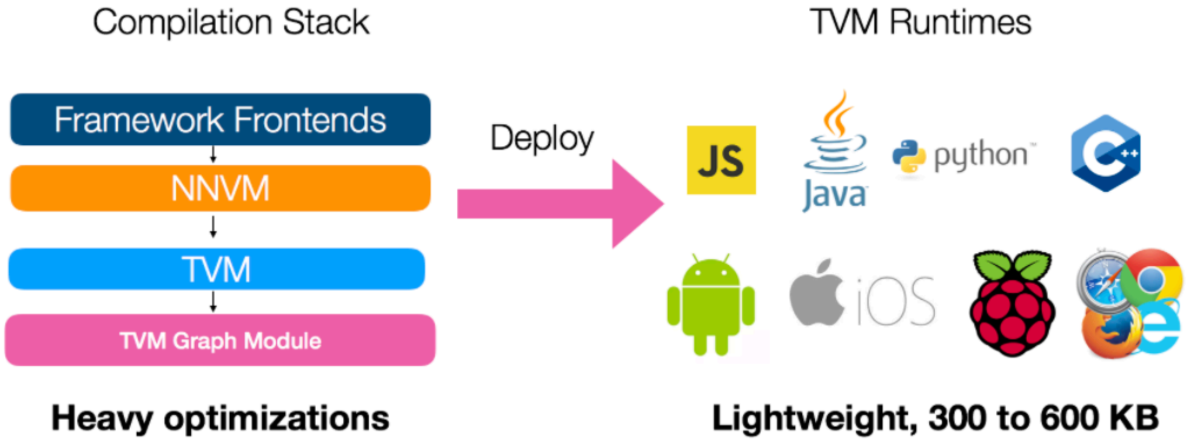
NNVM Compiler
nnvm
tvm

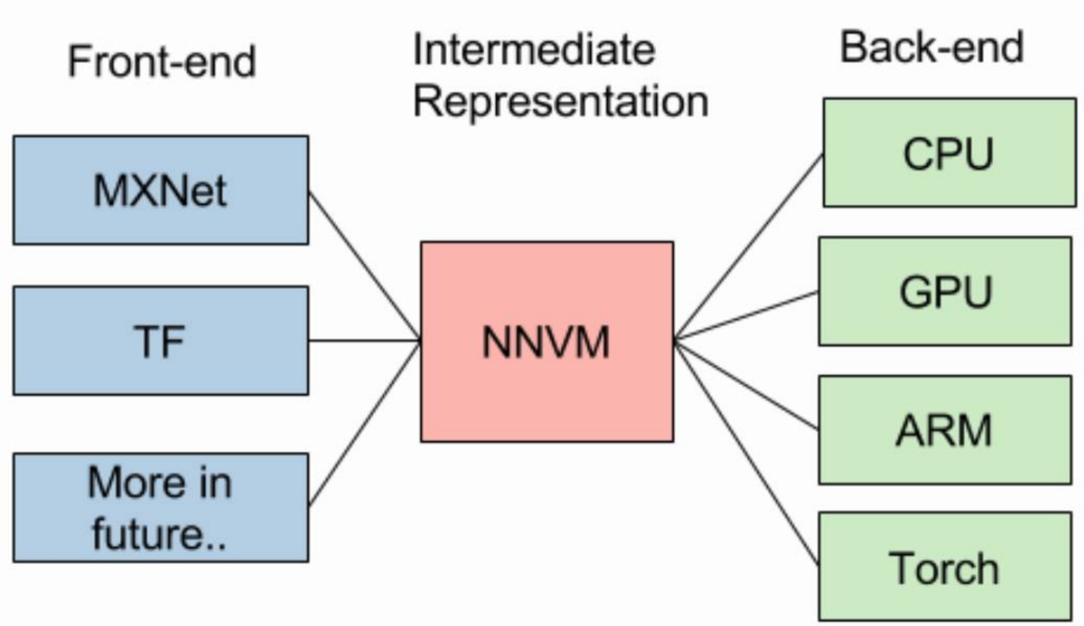
Computation Graph

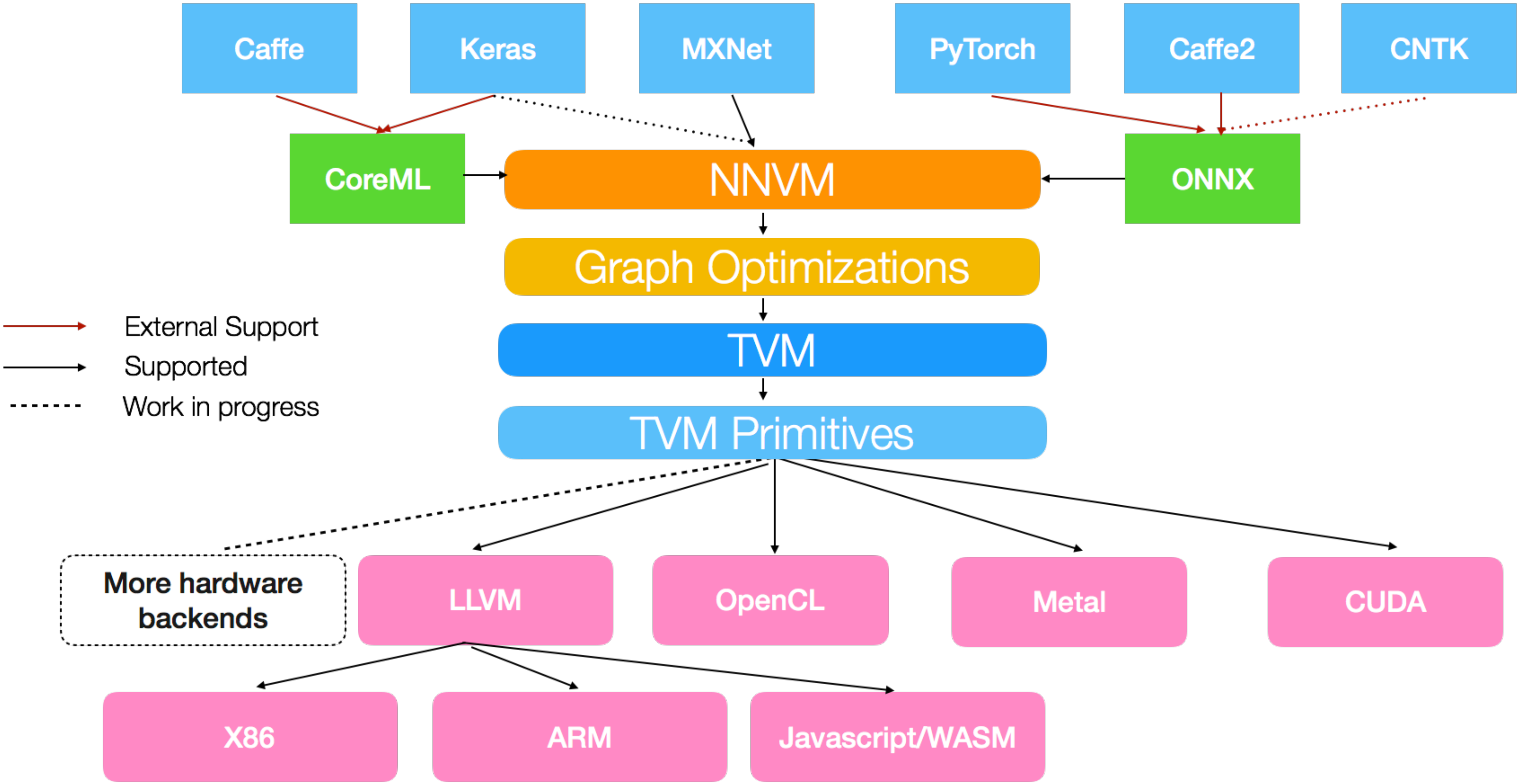


Calculate:
cosxsinx + x/y


NNVM Compiler
Neural Network Virtual Machine


Compilation Stack
Framework Frontends
NNVM
TVM
TVM Graph Module
Heavy optimizations
Deploy
TVM Runtimes
JS
Java
python
iOS
Lightweight, 300 to 600 KB


Front-end
Intermediate Representation
Back-end
MXNet
TF
More in future..
NNVM
CPU
GPU
ARM
Torch


Caffe
Keras
MXNet
PyTorch
Caffe2
CNTK
CoreML
NNVM
ONNX
Graph Optimizations
TVM
TVM Primitives
External Support
Supported
Work in progress
More hardware backends
LLVM
OpenCL
Metal
CUDA
X86
ARM
Javascript/WASM

nnvm-tvm

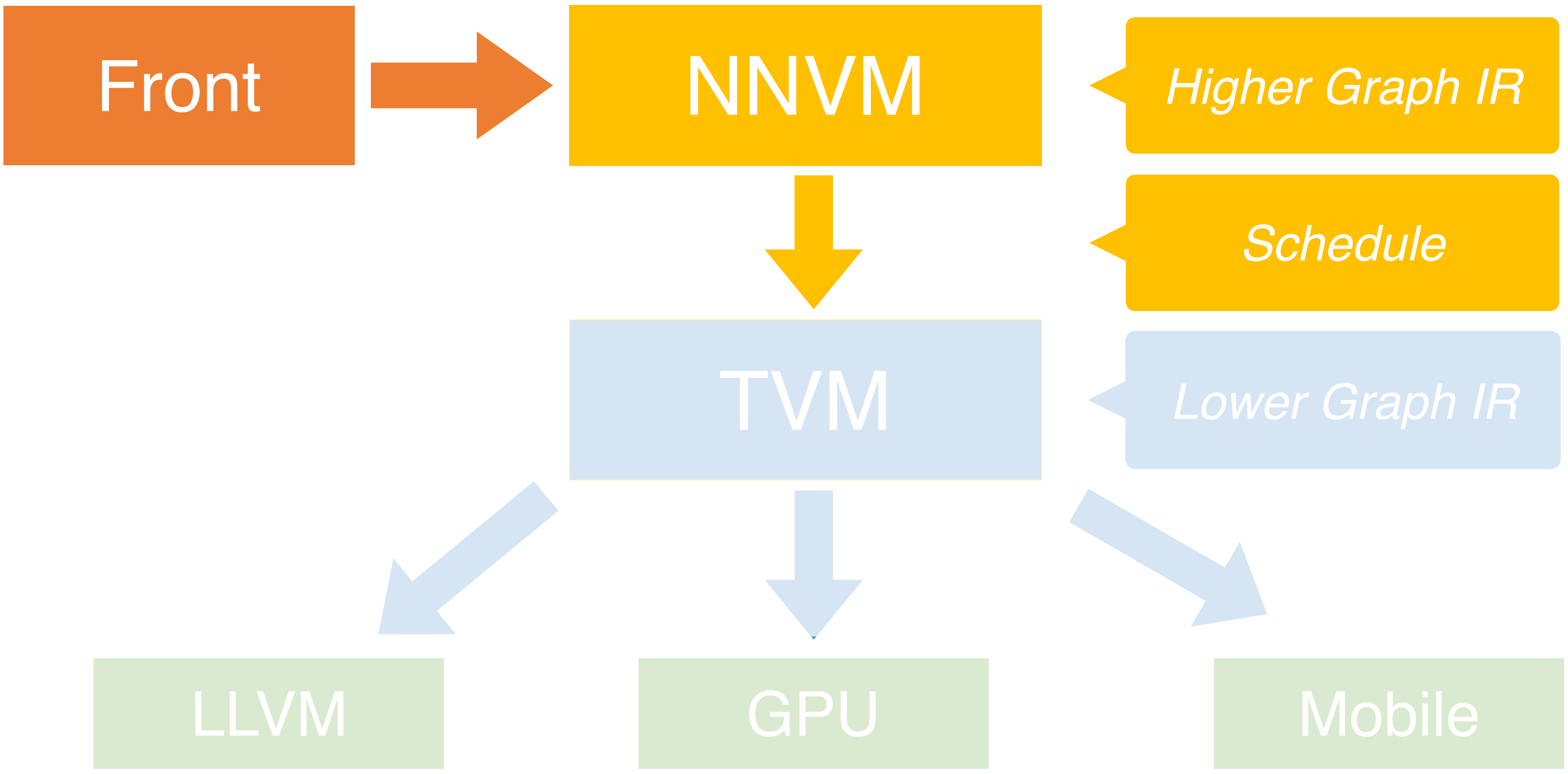
02

Part Two

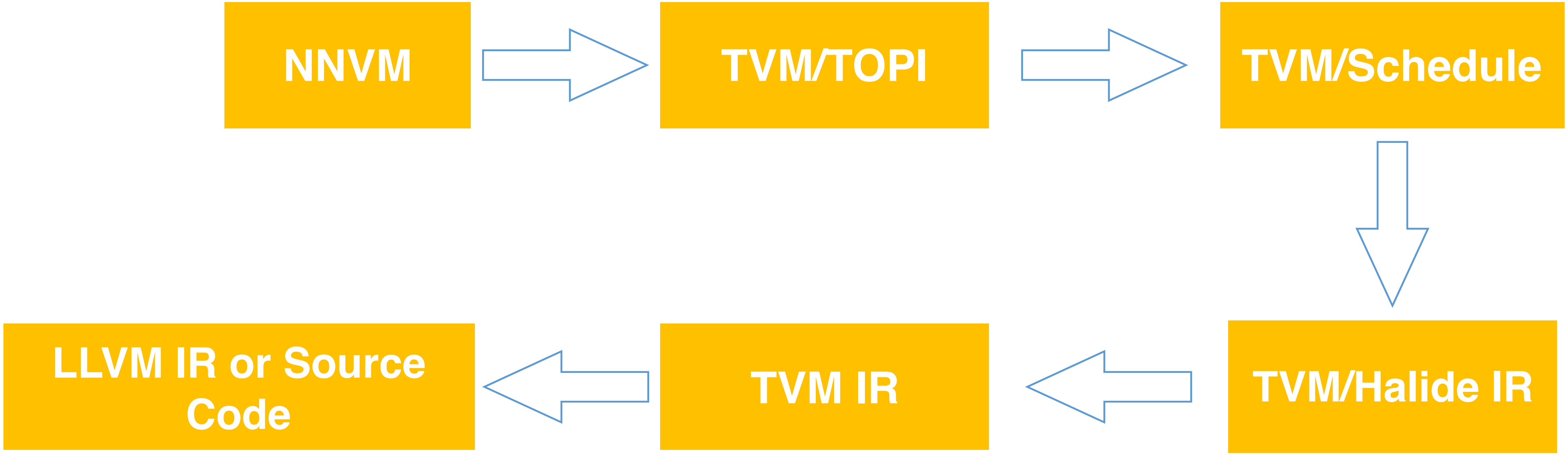
NNVM Introduction

Design Note
Optimization

NNVM Overview



NNVM Overview



TOPI (TVM Operator Inventory): TOPI is the operator collection library for TVM, to provide sugars for constructing compute declaration as well as optimized schedules.

02

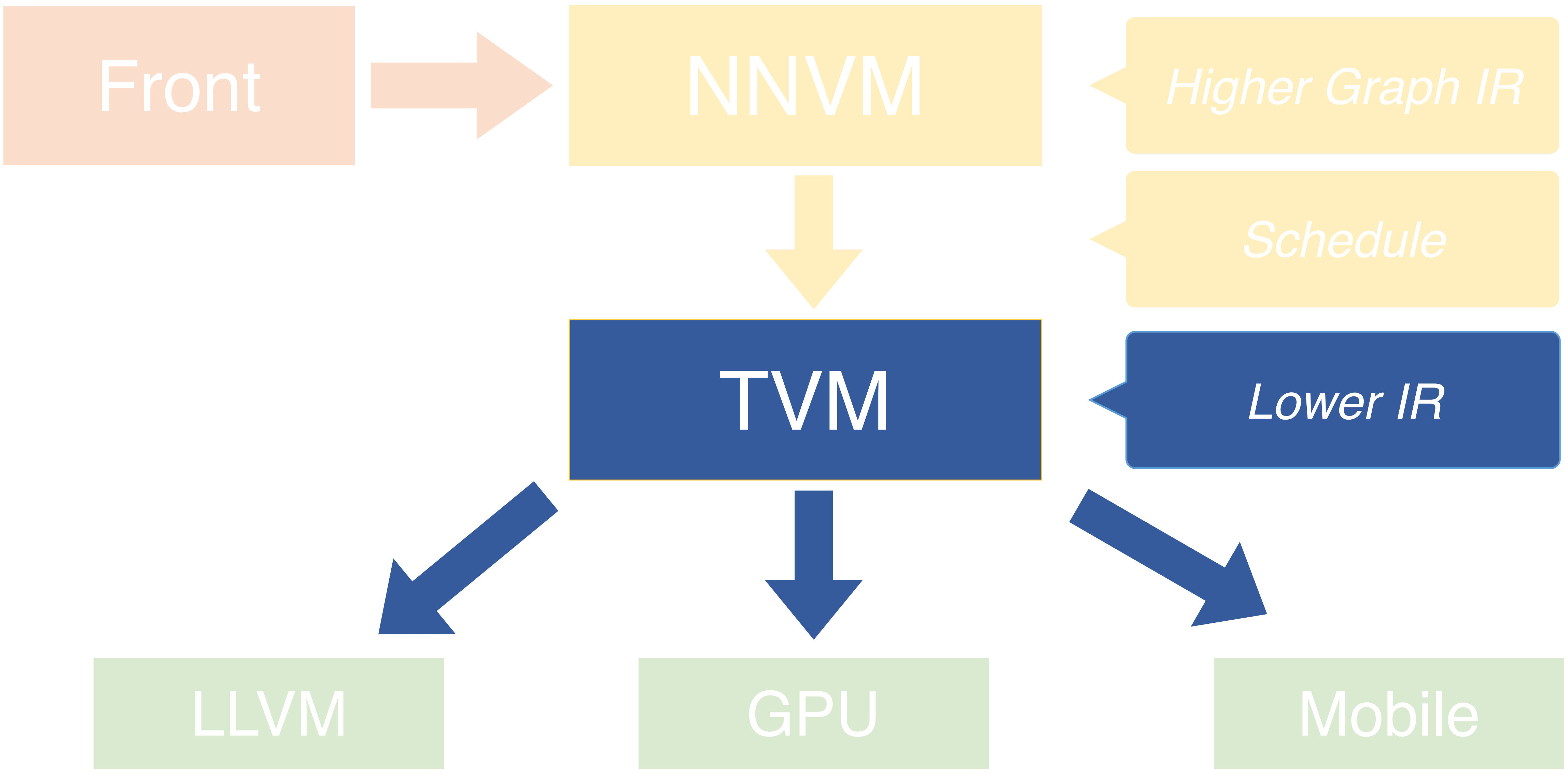
03

Part Three

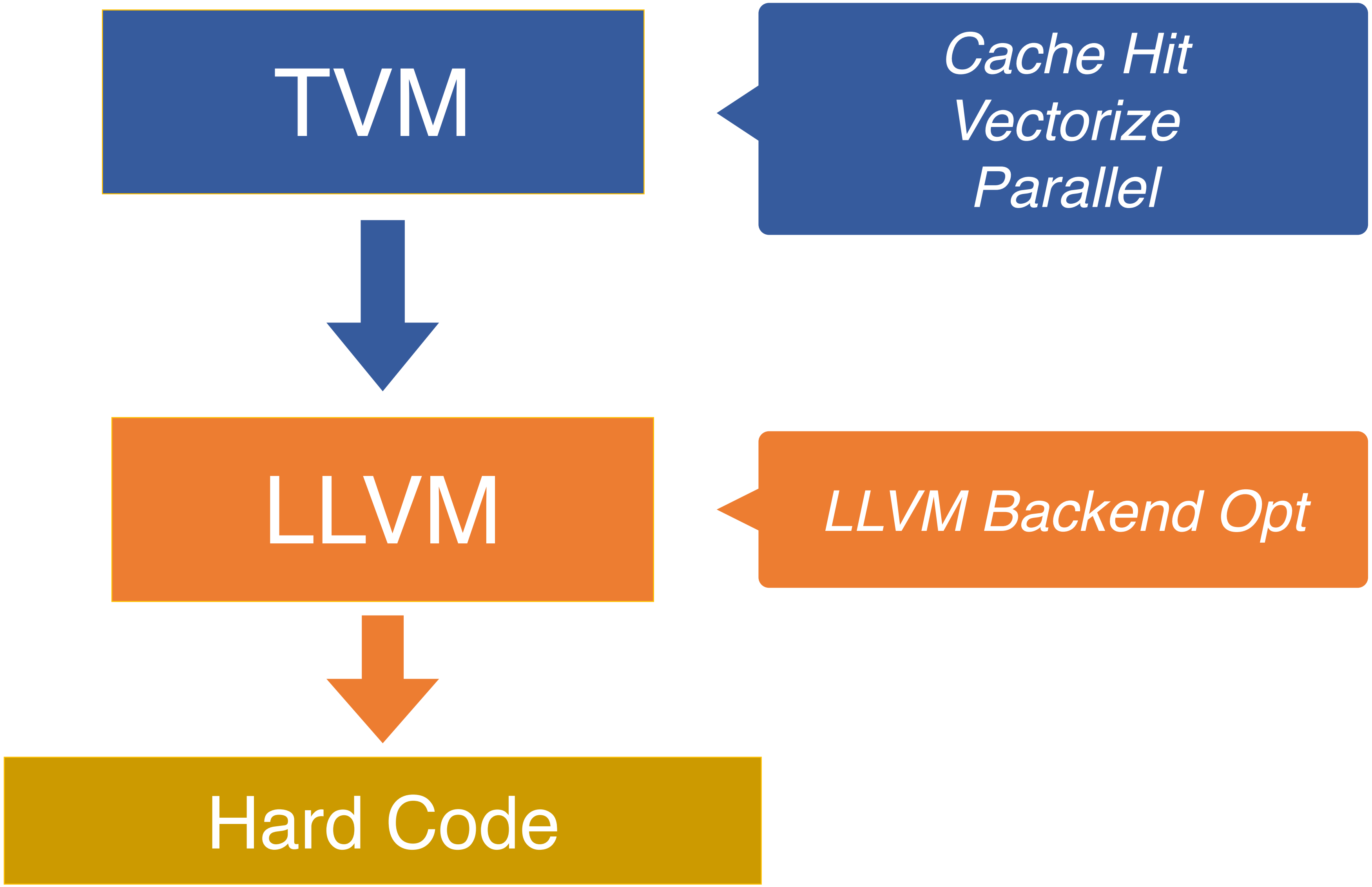
TVM Stack

Lower IR
Optimization

TVM Overview



TVM Optimization on CPU



TVM Optimization on CPU - Matrix Multi

```python
k = tvm.reduce_axis((0, N), 'k')
A = tvm.placeholder((N, N), name = 'A')
B = tvm.placeholder((N, N), name = 'B')
C = tvm.compute(
           A.shape,
           lambda x, y: tvm.sum(A[x, k] * B[k, y], axis = k),
           name = 'C')
```

Matrix dot:

A[1024,1024], B[1024,1024]

Intel(R) Xeon(R)
CPU E5-2660
16 logical cores

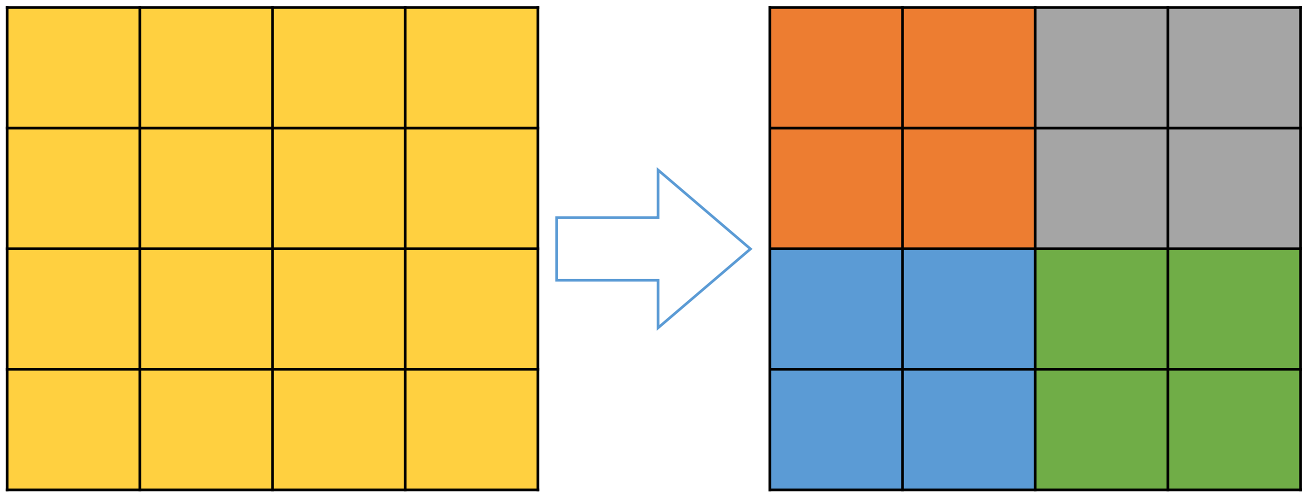
$$C = A \cdot B$$

T = 2.11167s

```
produce C {
  for (x, 0, 1024) {
    for (y, 0, 1024) {
      C[((x*1024) + y)] = 0.000000f
      for (k, 0, 1024) {
        C[((x*1024) + y)] = (C[((x*1024) + y)] + (A[((x*1024) + k)]*B[(y + (k*1024))]))
      }
    }
  }
}
```

TVM Optimization on CPU - Tiling/Blocking

Goal: Increase cache hit rate



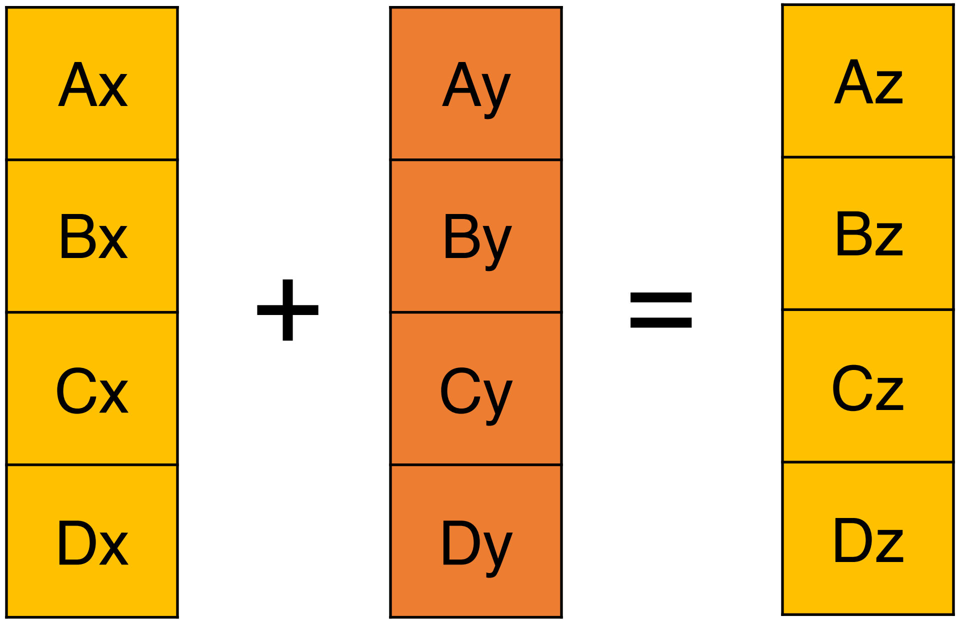
```
bn = 32
s = tvm.create_schedule(C.op)
# Blocking by loop tiling
xo, yo, xi, yi = s[C].tile(C.op.axis[0], C.op.axis[1], bn, bn)
```

T = 2.11167s -> 0.897558s

```
produce C {
  for (x.outer, 0, 32) {
    for (y.outer, 0, 32) {
      for (x.inner.init, 0, 32) {
        for (y.inner.init, 0, 32) {
          C[(((((x.outer*1024) + y.outer) + (x.inner.init*32)
        }
      }
      for (k, 0, 1024) {
        for (x.inner, 0, 32) {
          for (y.inner, 0, 32) {
            C[(((((x.outer*1024) + y.outer) + (x.inner*32))*3
          }
        }
      }
    }
  }
}
```

TVM Optimization on CPU - Vectorize

SIMD (Single Instruction Multiple Data)




```
s = tvm.create_schedule(C.op)
xo, yo, xi, yi = s[C].tile(C.op.axis[0], C.op.axis[1], bn, bn)
s[C].reorder(xo, yo, k, xi, yi)

# Vectorization
s[C].vectorize(yi)
func = tvm.build(s, [A, B, C], name = 'mmult')
```

T = 0.897558s -> 0.282522s

TVM Optimization on CPU - Parallel

Whenever parallelism is possible, just use parallel() give a hint to tvm

```
s = tvm.create_schedule(C.op)
xo, yo, xi, yi = s[C].tile(C.op.axis[0], C.op.axis[1], bn, bn)
s[C].reorder(xo, yo, xi, k, yi)
# vectorize
s[C].vectorize(yi)

# parallel
s[C].parallel(xo)
```

T = 0.028522s -> 0.015810s

Speedup

Optimization on CPU

Tiling

235%

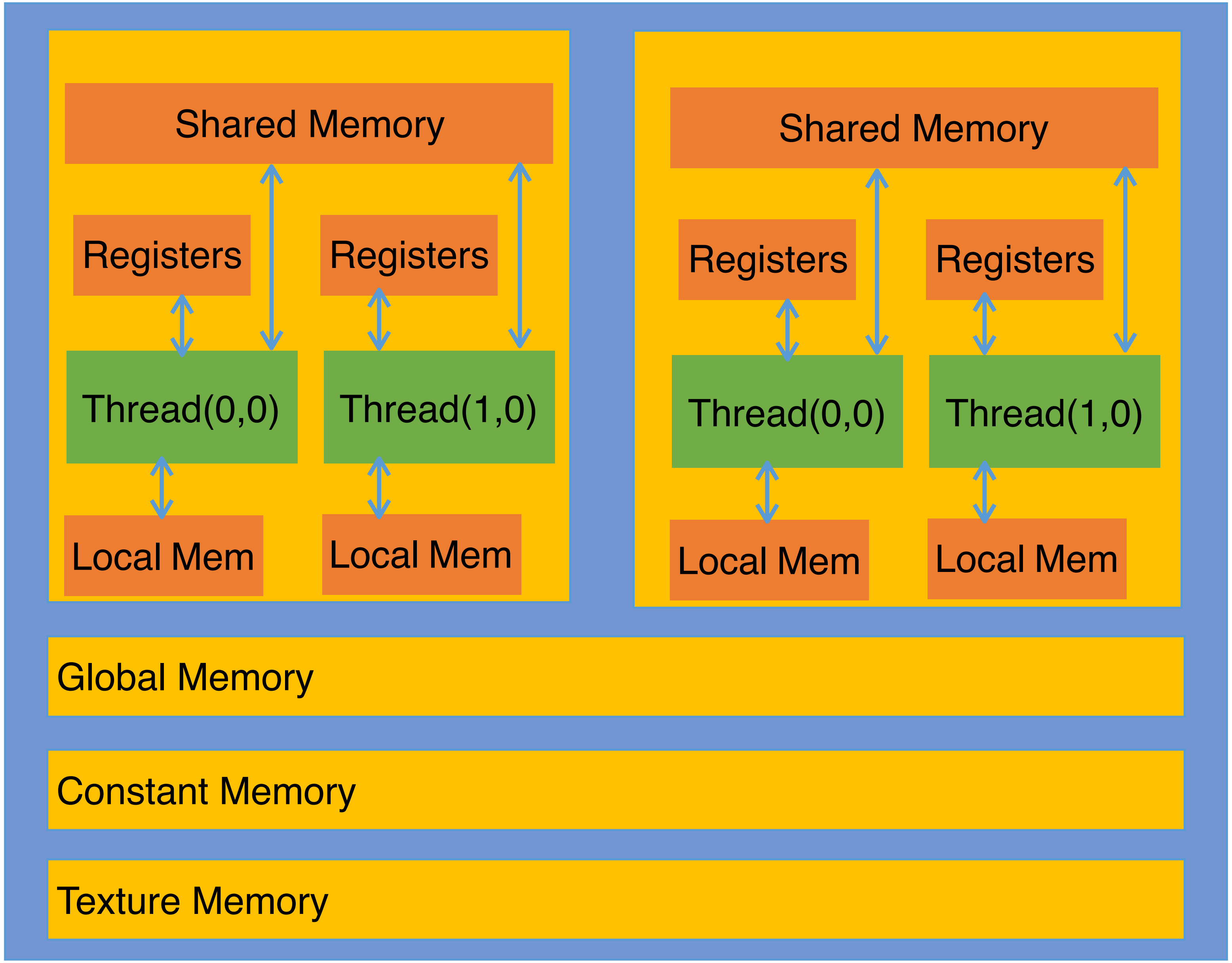
Tiling + Vectorize

747%

Tiling + Vectorize+ Parallel

1336%

TVM Optimization on GPU



TVM Optimization on GPU - Convolution

Convolution

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

*

1	0	-1
1	0	-1
1	0	-1

=

-5	-4	0	8
10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

TVM Optimization on GPU - Convolution

Convolution — Padding — Stride

3	0	1	2	7
1	5	8	9	3
2	7	2	5	1
0	1	3	1	7
4	2	1	6	2

Stride = 2

*

1	0	-1
1	0	-1
1	0	-1

=

-5	0
0	-4

TVM Optimization on GPU - Convolution

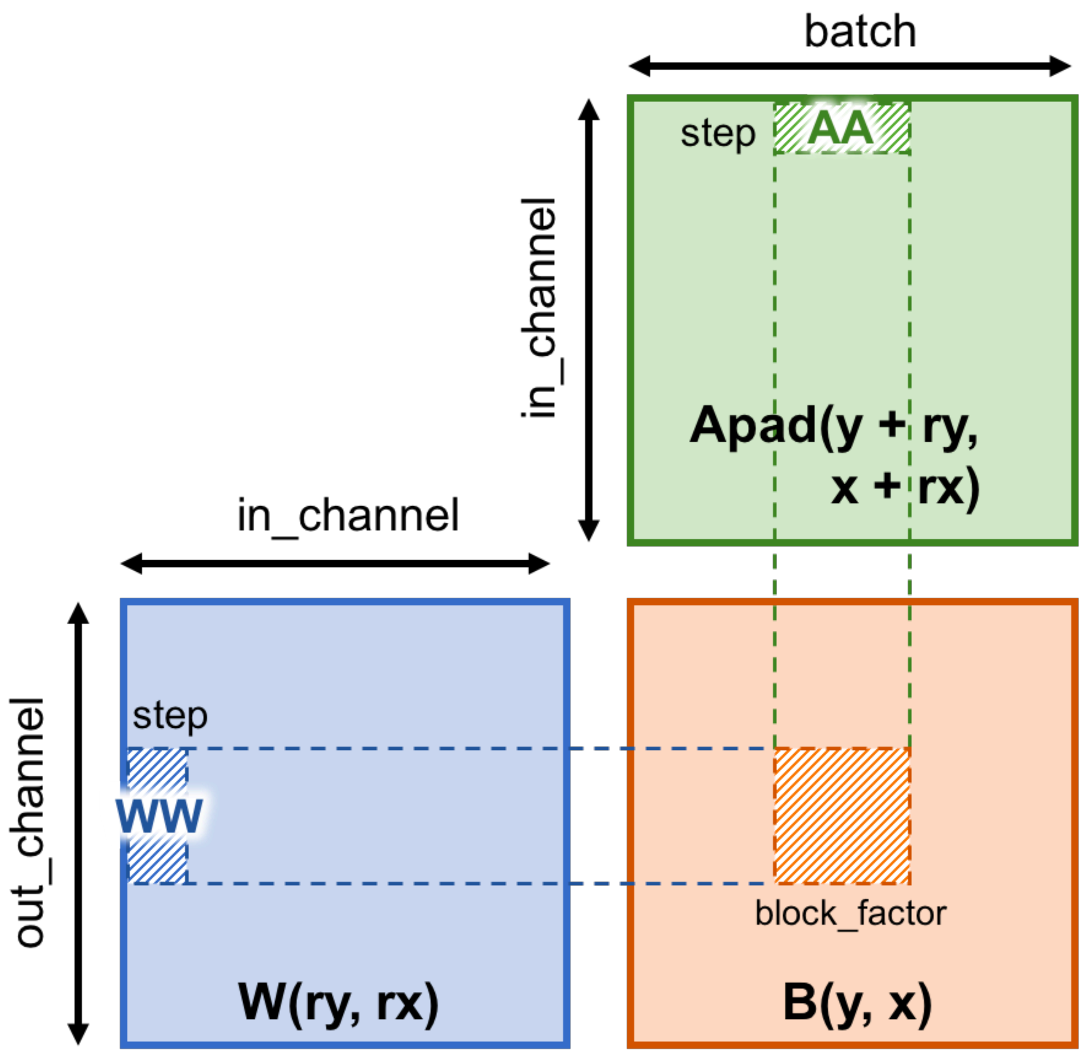
Optimize1: Blocking

AA/WW: Buffer

Apad: lhs in convolution

W: rhs in convolution

B: convolution result

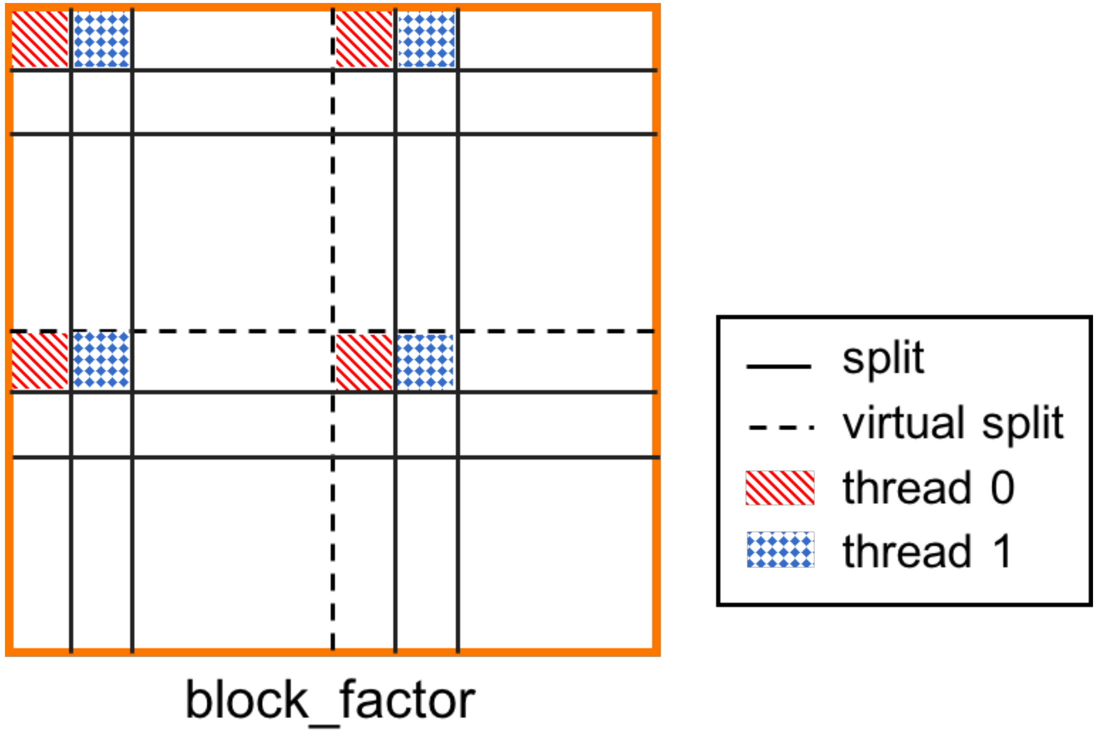


nnvm-tvm

TVM Optimization on GPU - Convolution

Optimize2:

Virtual Thread Split



Optimize3:

Cooperative Fetching

To reduce memory transfer per thread, make threads in the same thread block cooperatively fetch independent data from global memory

04

Part Four

Compare & Evaluate

XLA
Darkroom
Performance

NNVM vs. XLA

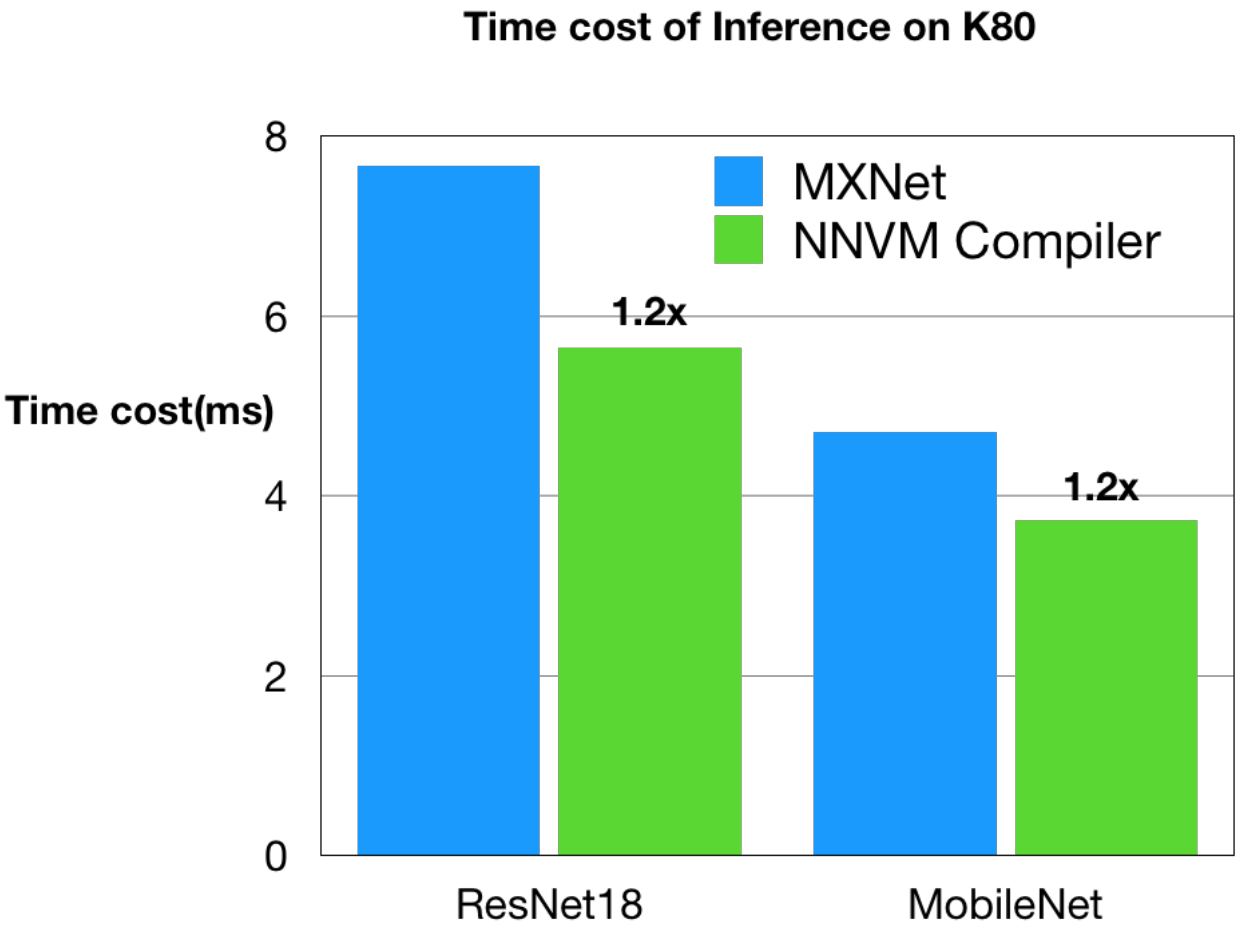
- Different strategy for code generation
- Different strategy for optimization methods
  - XLA: provide a unified optimization method for all hardware resources
  - NNVM: every hardware resource have relevant method
- Similar Optimization on high-level graphs (fusions, layout…)

NNVM vs. Darkroom/Halide

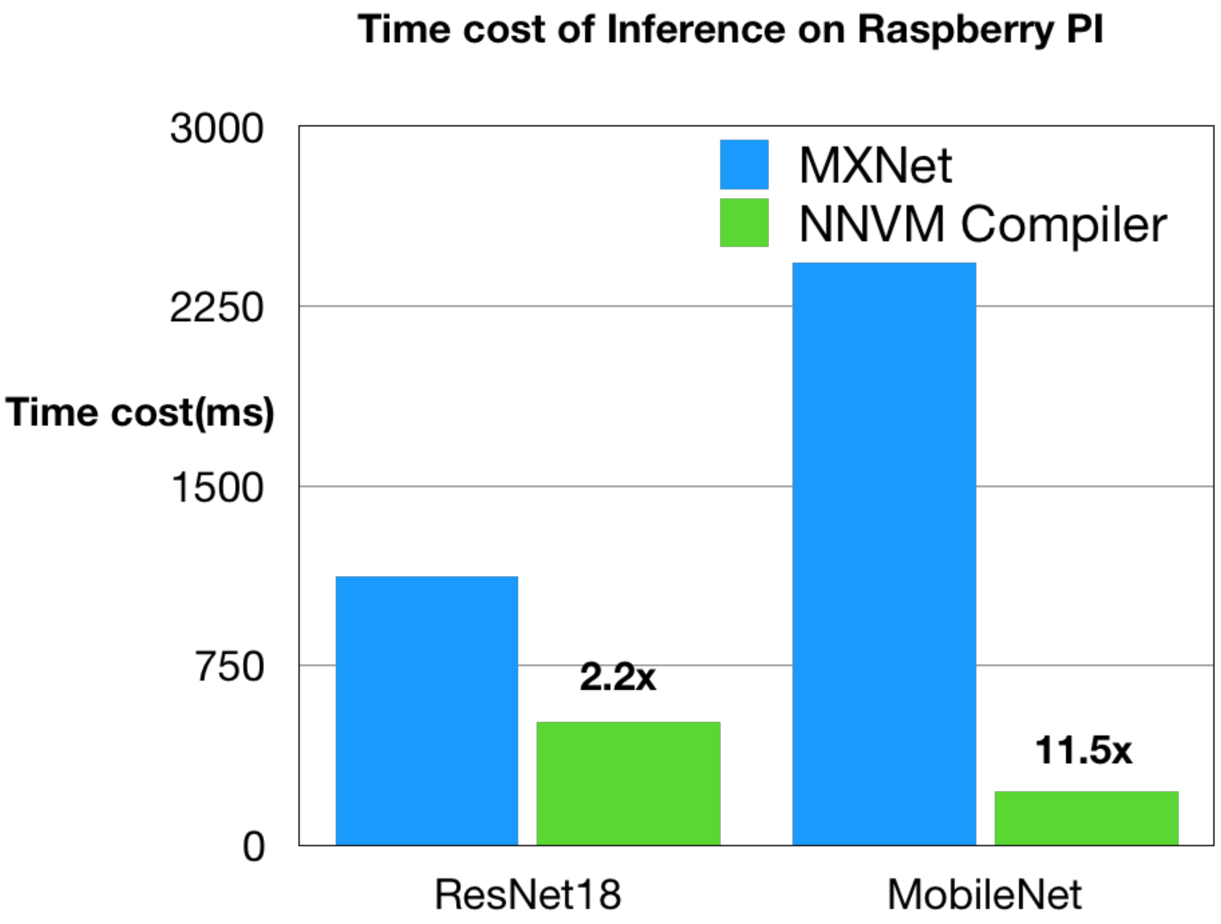
- Similar Ideas between Halide and TVM
  - Both separate algorithms and schedules
- Similar strategy for optimization methods
  - Both have similar optimization on Hardware-independent code
- Different levels of optimization
  - NNVM has more rules and more efficient ways on optimization

NNVM Performance

INVIDIA GPU



Raspberry PI



Thanks

USTC Compiler Team 13 nnvm-tvm
Zhu Yiming, Xie Yunting, Lin Zhiqi