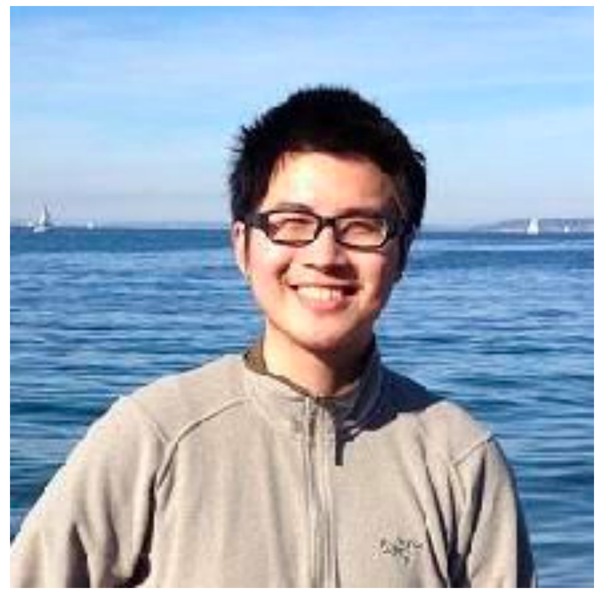
PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

An End to End IR Stack for Deep Learning Systems

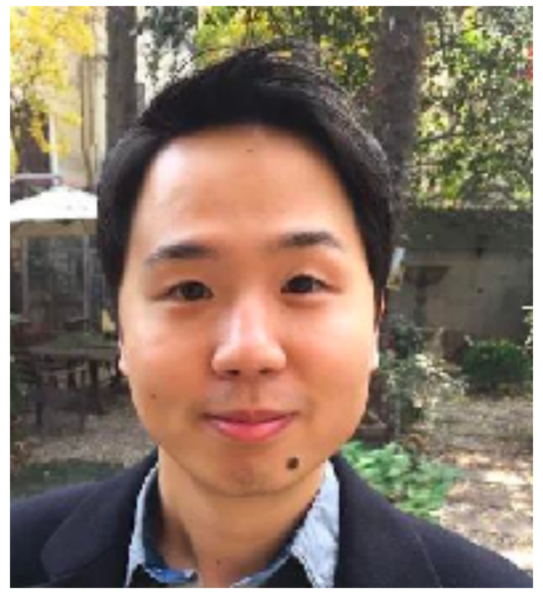
Presenter: Tianqi Chen
Carlos Guestrin, Luis Ceze, Arvind Krishnamurthy

Collaborators

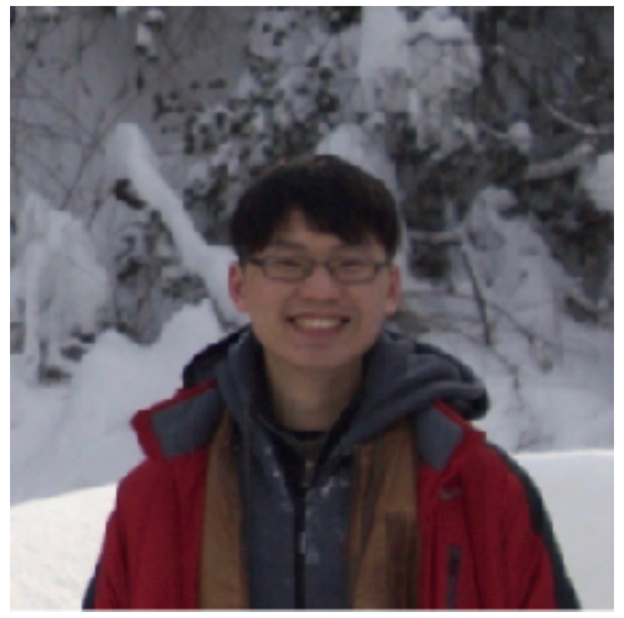


Tianqi Chen

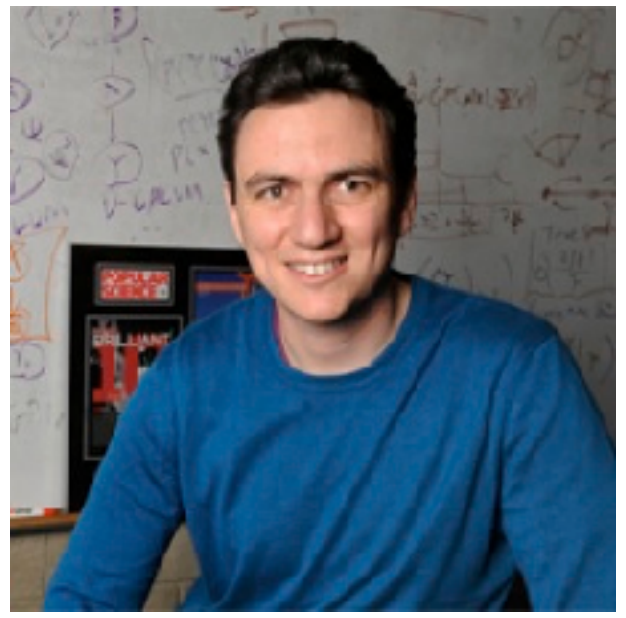
Thierry Moreau



Haichen Shen



Ziheng Jiang



Carlos Guestrin

Luis Ceze

Arvind Krishnamurthy

and many contributors in DMLC community

Apache MXNet: Mixed Approach to Deep Learning

Imperative API

```
>>> import mxnet as mx
>>> a = mx.nd.zeros((100, 50))
>>> a.shape
(100L, 50L)
>>> b = mx.nd.ones((100, 50))
>>> c = a + b
>>> b += c
```

10k github stars,
Adopted by AWS

Declarative API

```
>>> import mxnet as mx
>>> net = mx.symbol.Variable('data')
>>> net = mx.symbol.FullyConnected(data=net, num_hidden=128)
>>> net = mx.symbol.SoftmaxOutput(data=net)
>>> type(net)
<class 'mxnet.symbol.Symbol'>
>>> texec = net.simple_bind(data=data_shape)
```

mxnet

Challenges for Deep Learning System

Framework

Framework

Hardware
Back-Ends

Challenges for Deep Learning System

Framework

Framework

Hardware
Back-Ends

Challenges for Deep Learning System

Framework

Framework

Hardware
Back-Ends

Challenges for Deep Learning System

Framework

Framework

Hardware
Back-Ends

Challenges for Deep Learning System

Framework

Framework

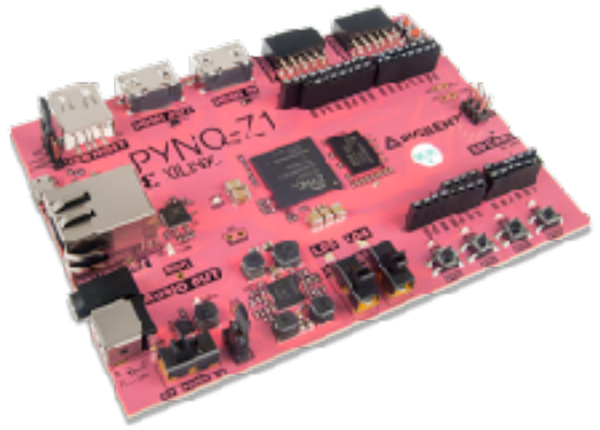
Hardware
Back-Ends

Challenges for Deep Learning System

Framework

Framework





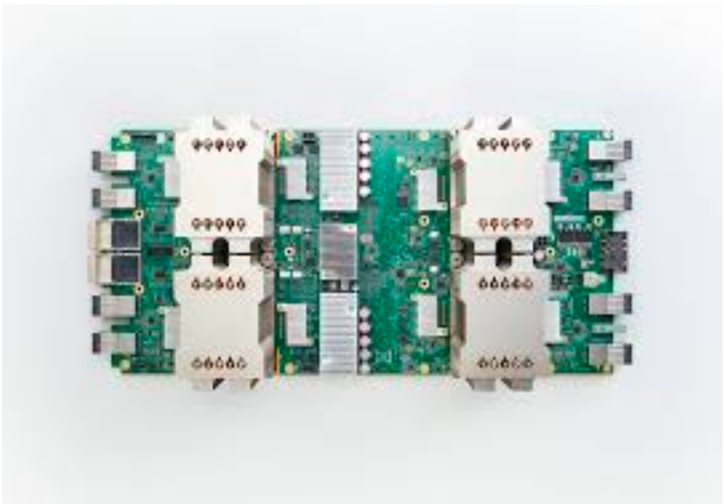
Hardware
Back-Ends

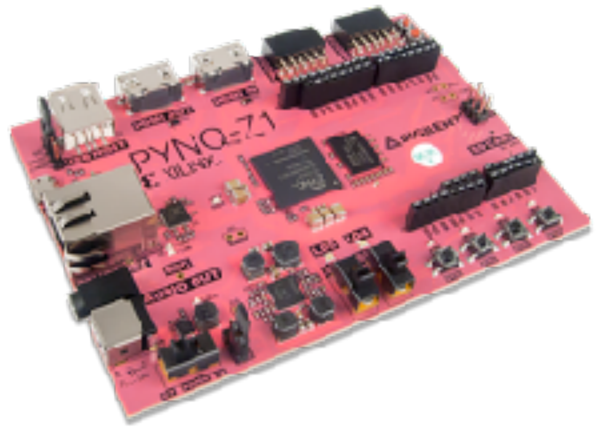
Challenges for Deep Learning System

Framework

Framework

Hardware
Back-Ends





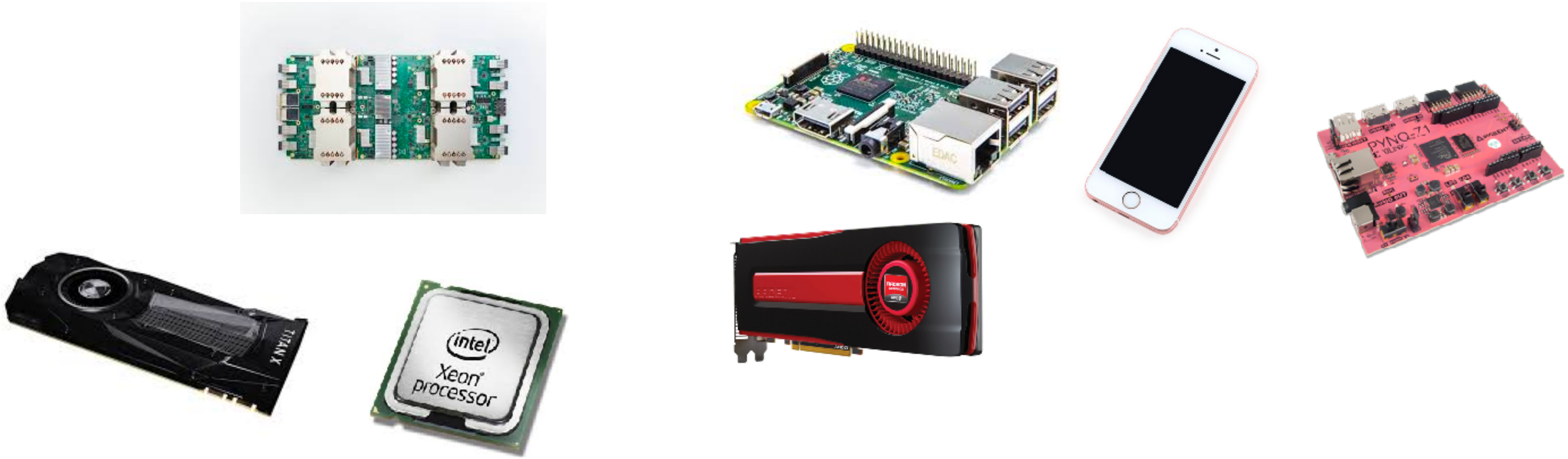
Challenges for Deep Learning System

Framework

Framework

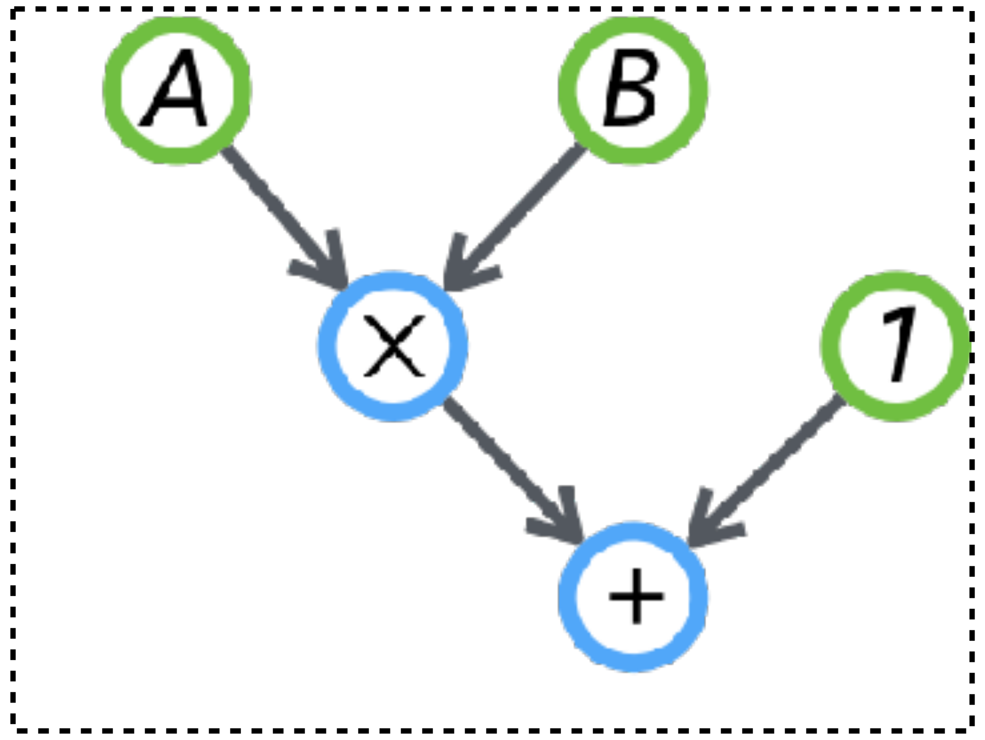
Intermediate representation

Hardware
Back-Ends



NNVM: Graph as High Level IR

Frontends



NNVM Graph

Auto Differentiation

Memory Plan

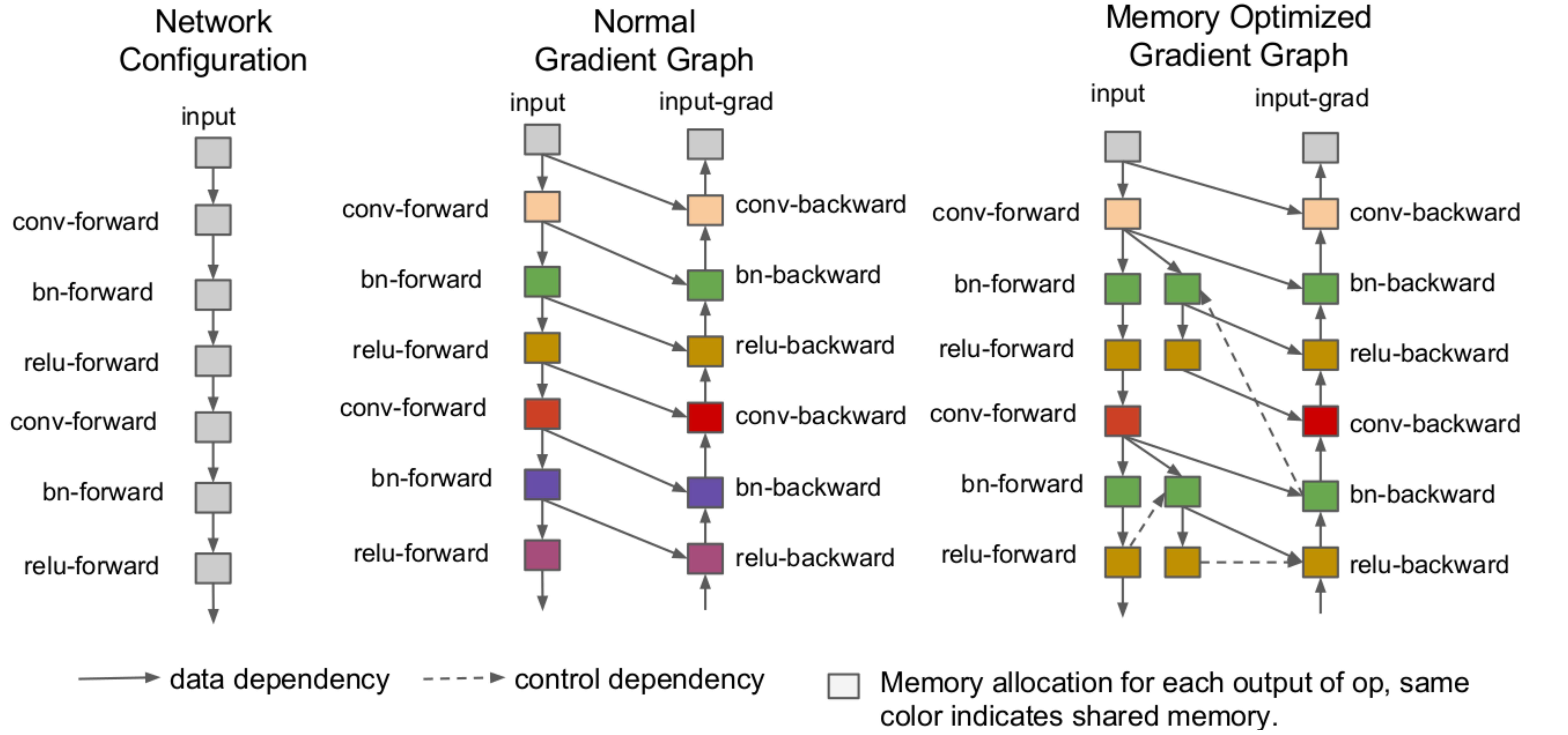
Operator Fusion

...

Backends

Memory Optimization via Gradient Graph Rewriting

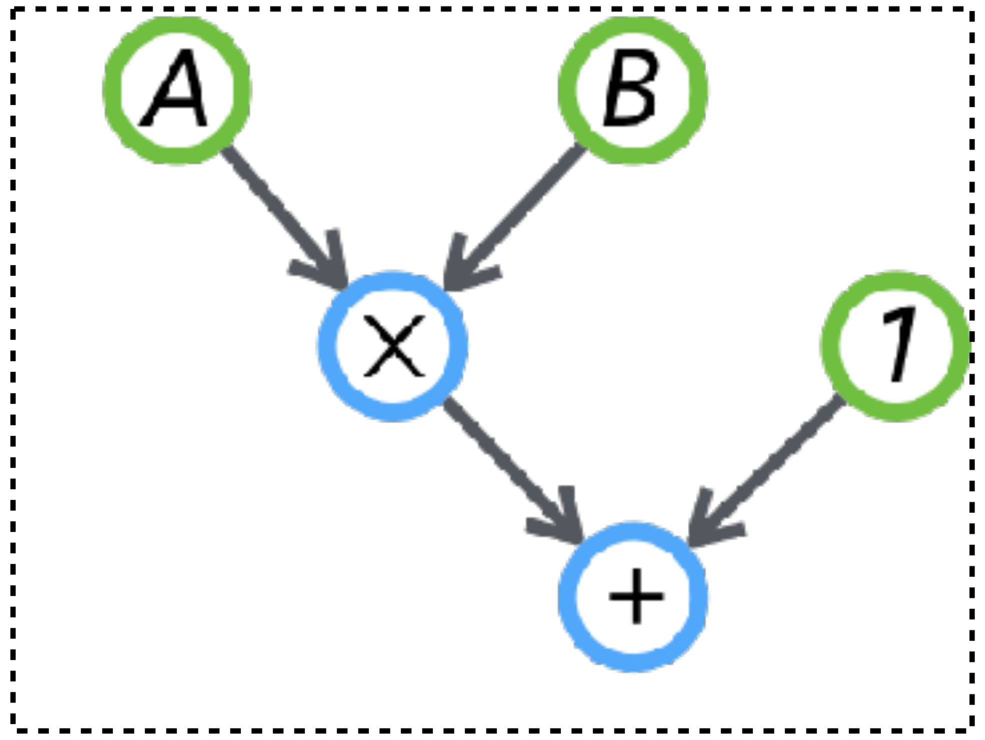
Can now training 1000 layer ResNet on ImageNet with a Titan X



- **Training Deep Nets with Sublinear Memory Cost** Chen.et.al arXiv 1604.06174
- **Memory-Efficient Backpropagation Through Time** Gruslys.et.al arXiv:1606.03401

Data Layout and Precision Optimization

Computation Graph to Code: The Remaining Gap



NNVM Graph

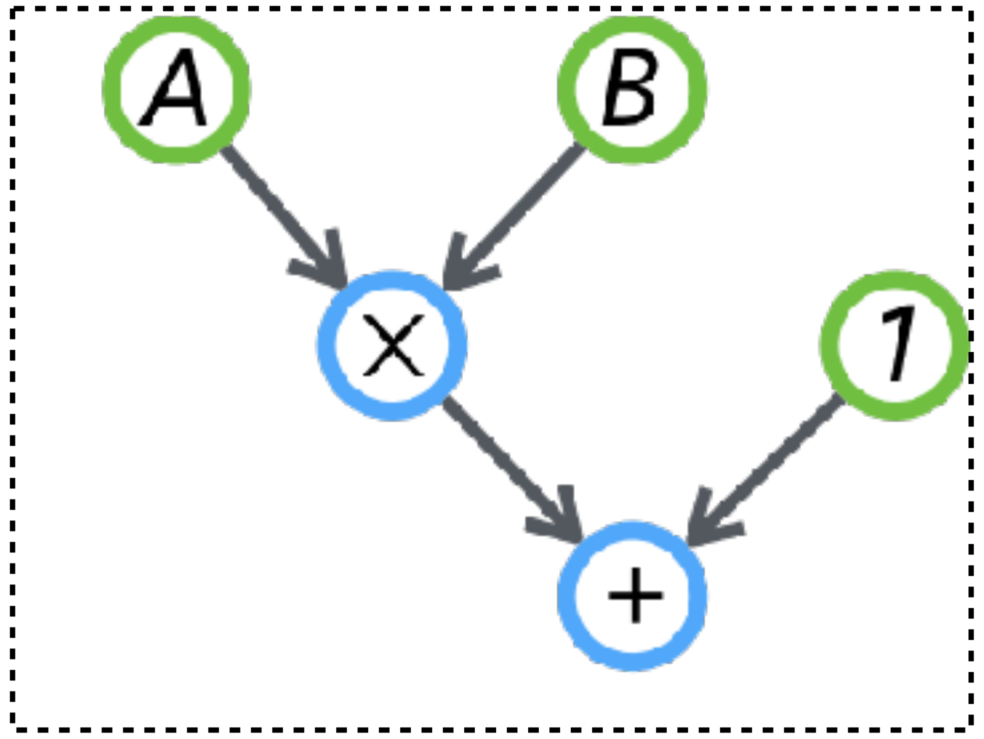
Auto Differentiation

Memory Plan

Operator Fusion

Backends

Computation Graph to Code: The Remaining Gap



NNVM Graph

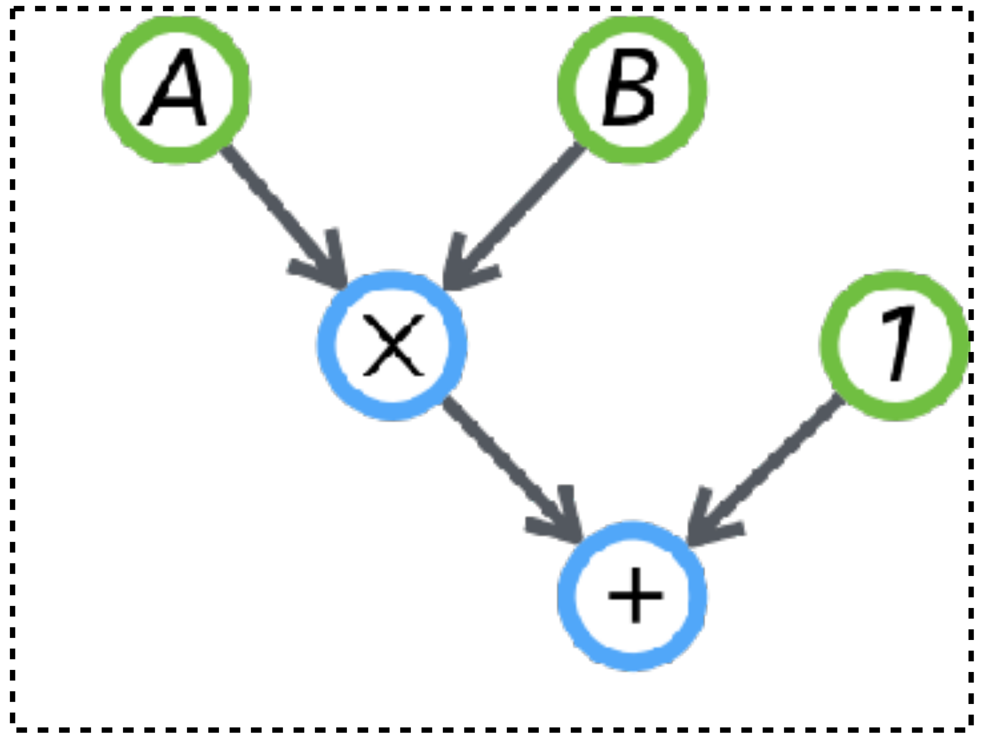
Auto Differentiation

Memory Plan

Operator Fusion

Backends

Computation Graph to Code: The Remaining Gap



NNVM Graph

Auto Differentiation

Memory Plan

Operator Fusion

too many possible choices:

precision, layout, fused pattern, device, threading …

Need a low level IR to express them explicitly

Backends

A Wishlist of Low Level IR

A Wishlist of Low Level IR

- Compact and expressive

A Wishlist of Low Level IR

- Compact and expressive
- Explicit control over memory layout of the data

A Wishlist of Low Level IR

- Compact and expressive
- Explicit control over memory layout of the data
- Explicit control over data locality and data movement

A Wishlist of Low Level IR

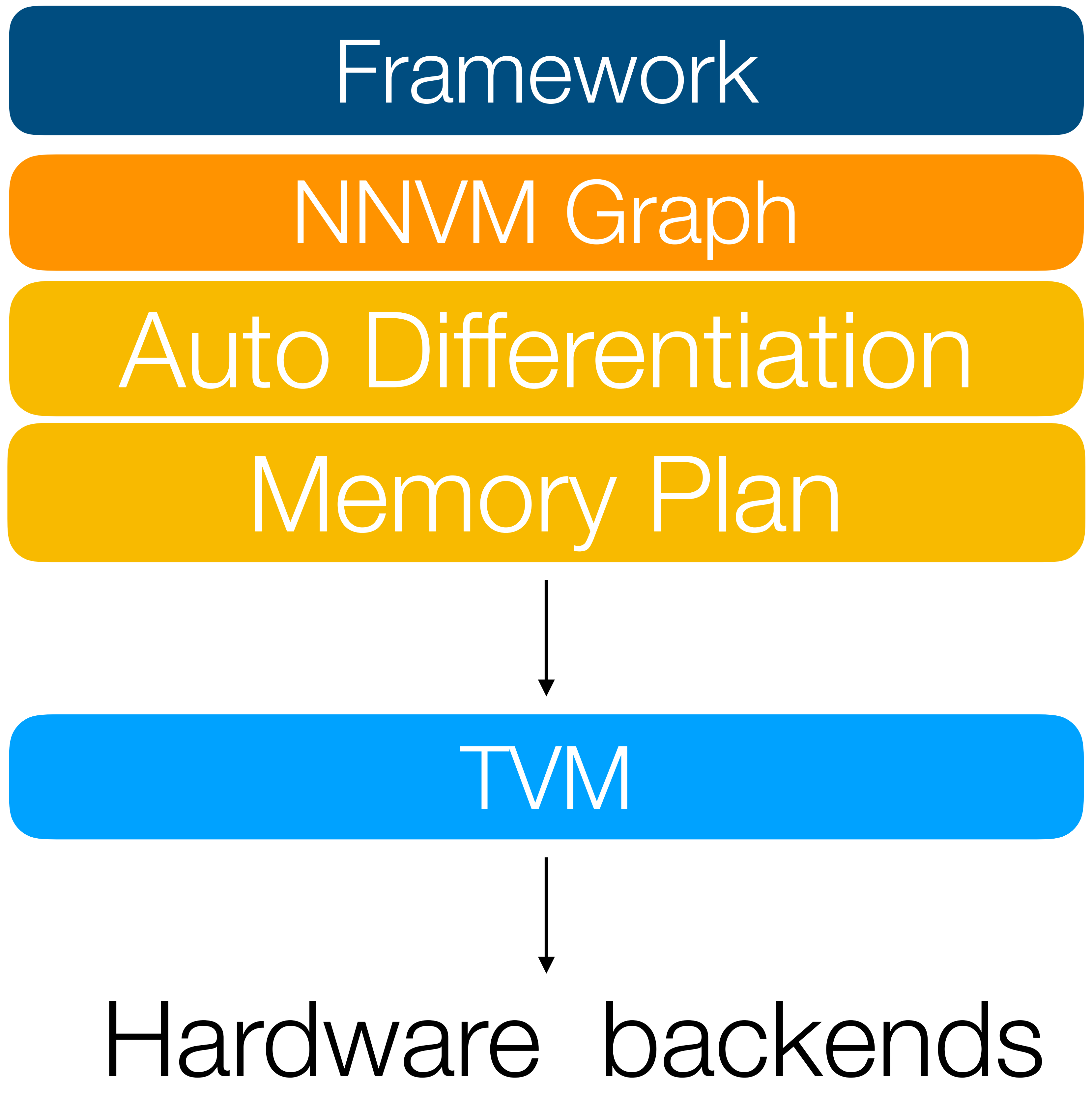
- Compact and expressive
- Explicit control over memory layout of the data
- Explicit control over data locality and data movement
- Support for tensorization

A Wishlist of Low Level IR

- Compact and expressive
- Explicit control over memory layout of the data
- Explicit control over data locality and data movement
- Support for tensorization
- Support for arbitrary fixed-point/floating precision

TVM: Low Level IR

- Concise and compact description
- Explicit control on codegen
- Ease of deployment
- Support accelerators



Index based IR

Compute C = dot(A, B.T)

```
import tvm

m, n, h = tvm.var('m'), tvm.var('n'), tvm.var('h')
A = tvm.placeholder((m, h), name='A')
B = tvm.placeholder((n, h), name=‘B')


k = tvm.reduce_axis((0, h), name=‘k')
C = tvm.compute((m, n), lambda i, j: tvm.sum(A[i, k] * B[j, k], axis=k))
```

Inputs

Shape of C

Computation Rule

Expressive Compute Primitive

Affine Transformation

```
out = tvm.compute((n, m), lambda i, j: tvm.sum(data[i, k] * w[j, k], k))
out = tvm.compute((n, m), lambda i, j: out[i, j] + bias[i])
```

Convolution

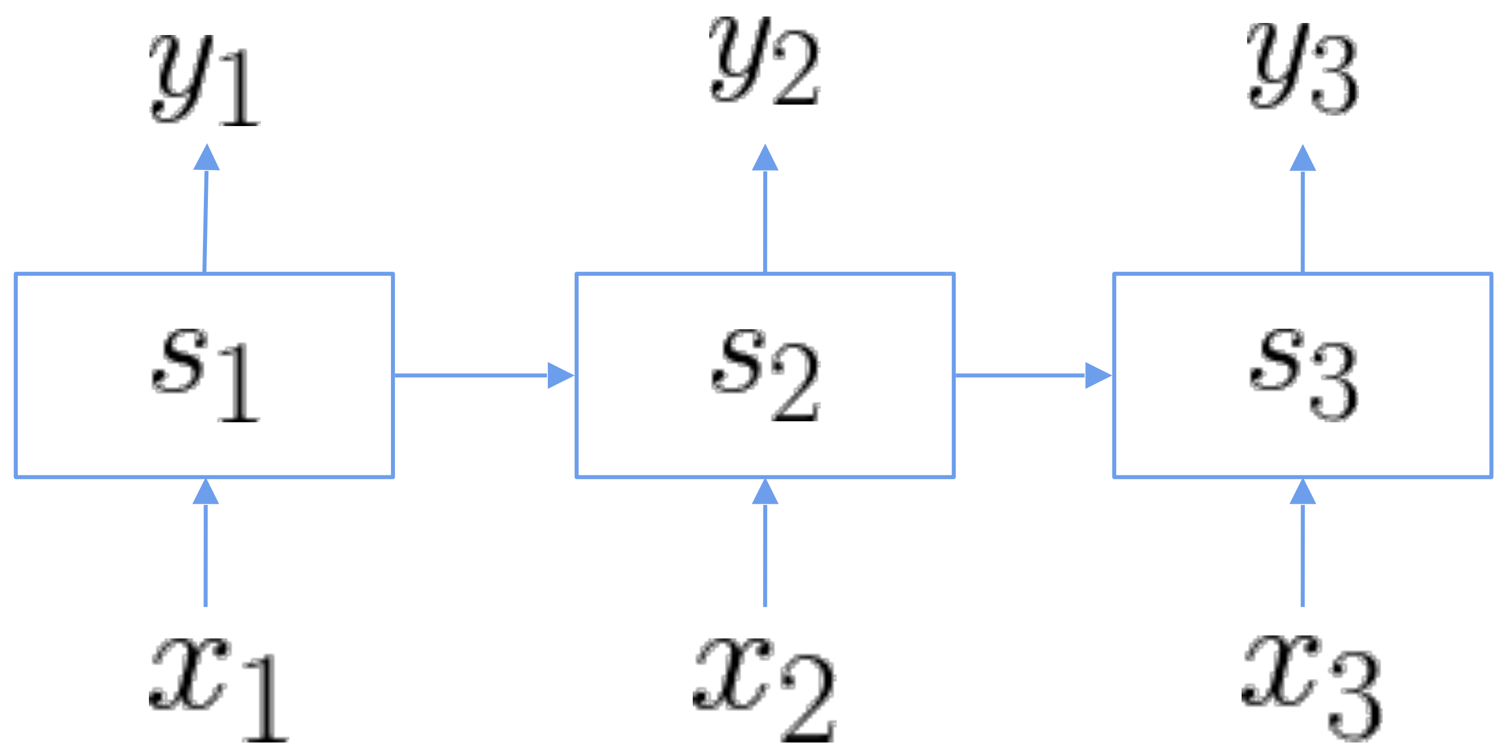
```
out = tvm.compute((c, h, w),
   lambda i, x, y: tvm.sum(data[kc,x+kx,y+ky] * w[i,kx,ky], [kx,ky,kc]))
```

Fused with Activation

```
out =tvm.compute(shape, lambda *i: tvm.max(0, out(*i))
```

With Recurrent Support

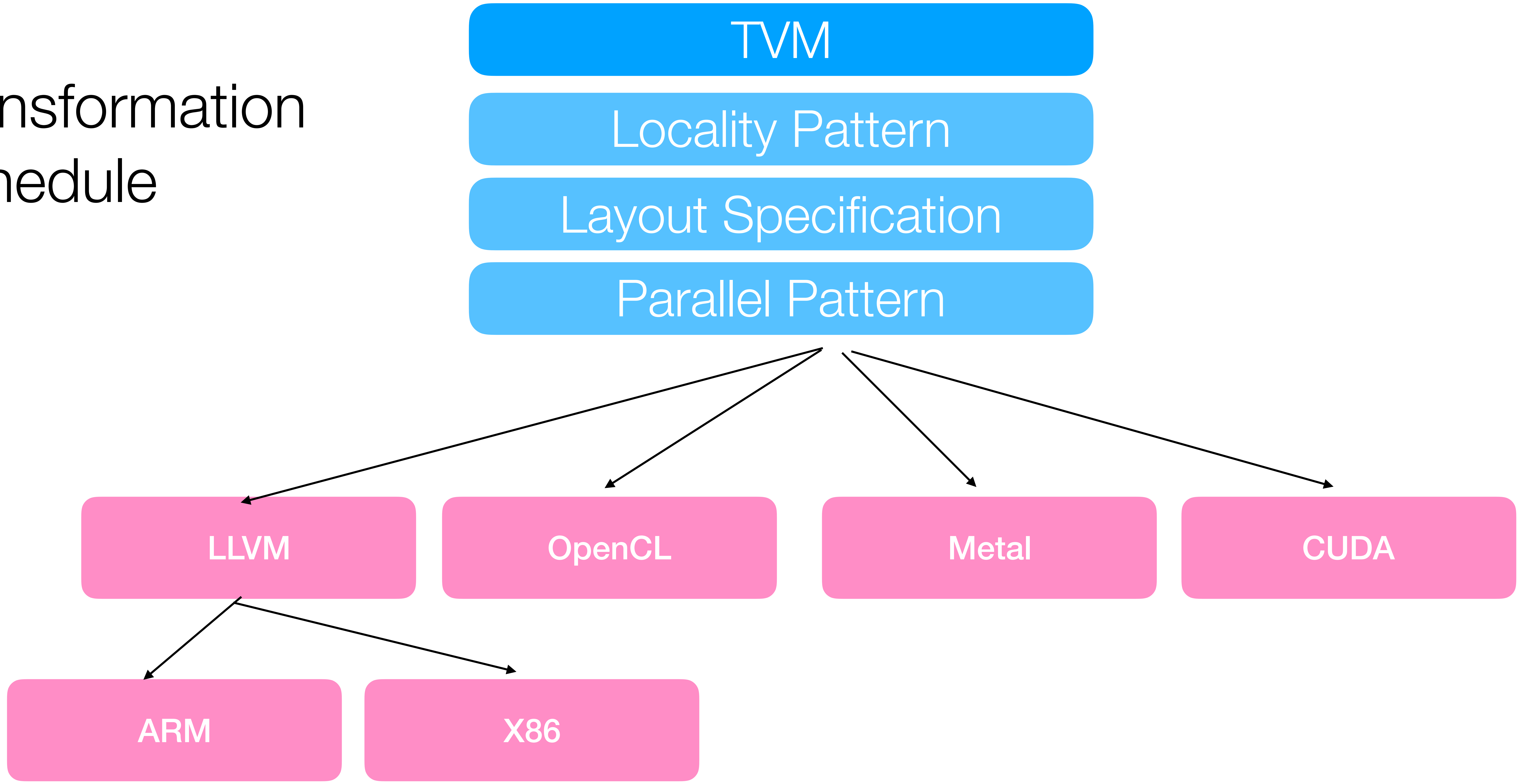
Compute Y = cumsum(X)



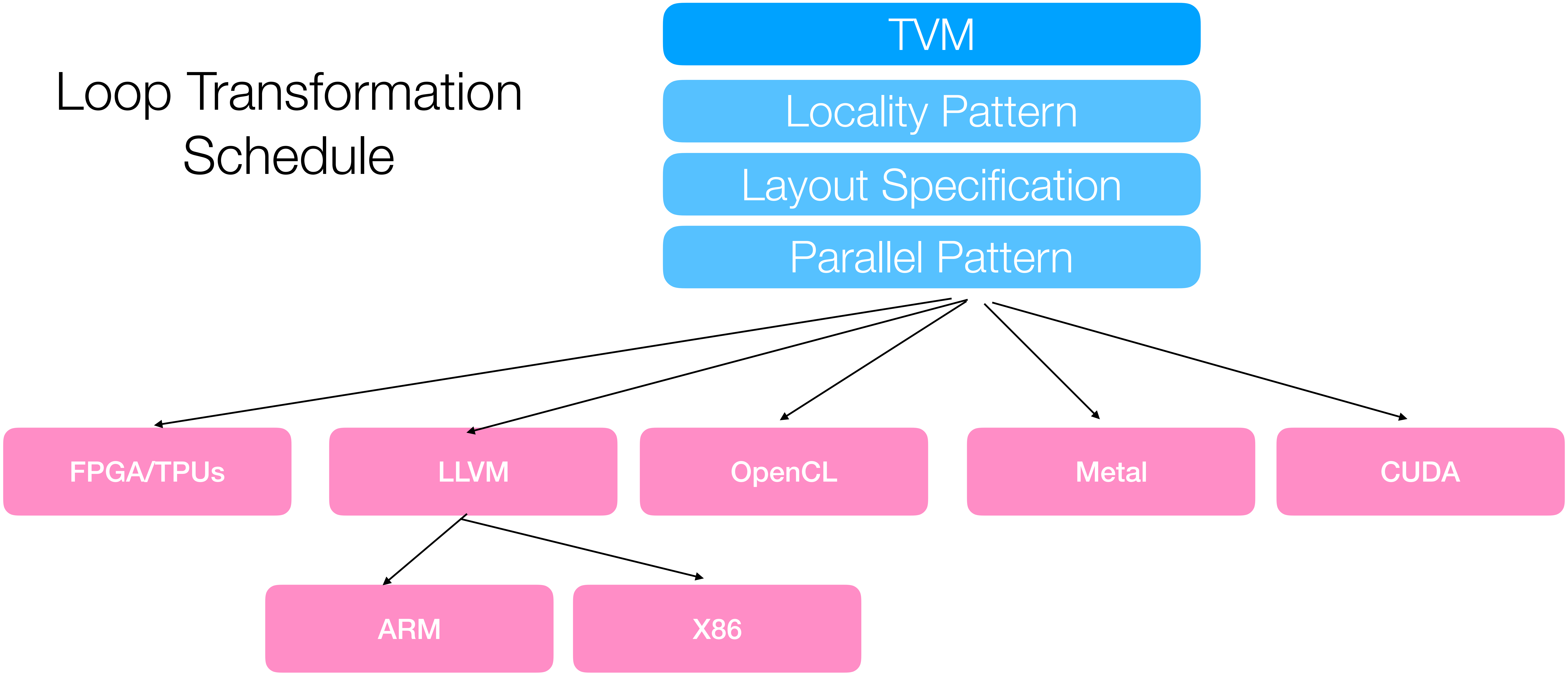

```
import tvm

m = tvm.var("m")
n = tvm.var("n")
X = tvm.placeholder((m, n), name="X")

s_state = tvm.placeholder((m, n))
s_init = tvm.compute((1, n), lambda _, i: X[0, i])
s_update = tvm.compute((m, n), lambda t, i: s_state[t-1, i] + X[t, i])

Y = tvm.scan(s_init, s_update, s_state, inputs=[X])
```

Multiple Backends

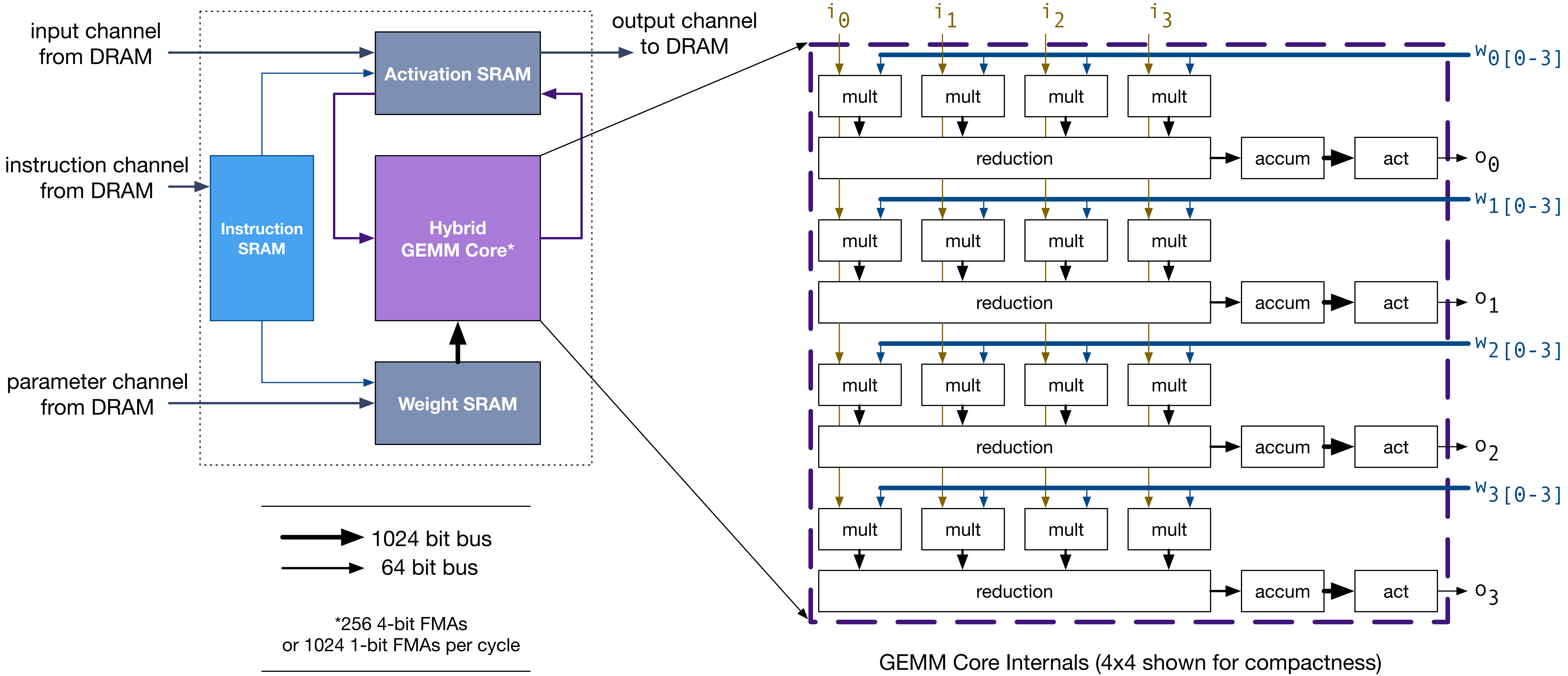
Loop Transformation
Schedule



Multiple Backends



Prototype FPGA-based GEMM Accelerator



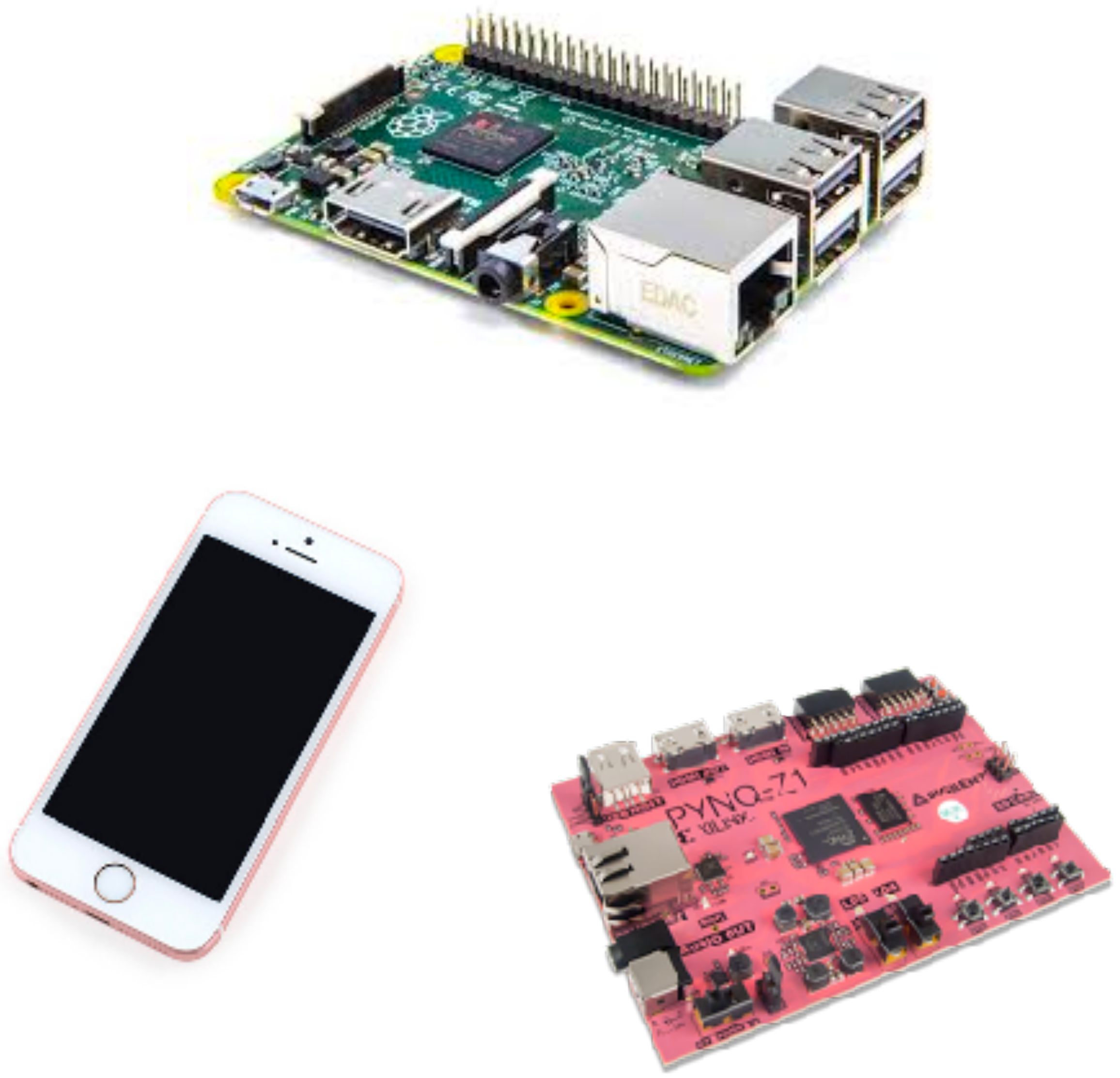
GEMM Core Internals (4x4 shown for compactness)

Remote Execution

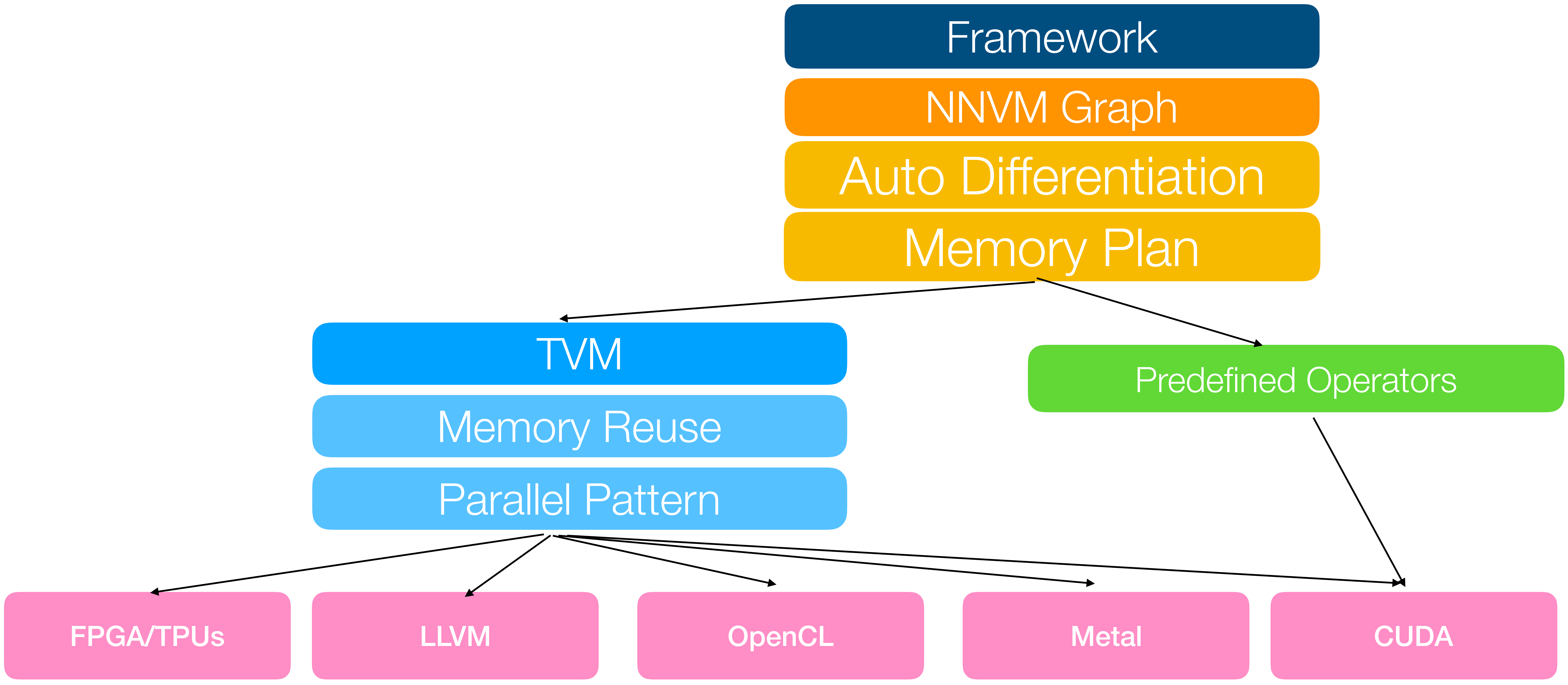
Server with TVM Compiler

TVM RPC

Devices with TVM Runtime



An End-to-End Stack



Thank you!

- MxNet Github Link: https://github.com/dmlc/mxnet
- UW Deep learning system course： http://dlsys.cs.washington.edu/
- TVM: to be open-sourced in summer
- Email contact: tqchen@cs.washington.edu, moreau@cs.washington.edu