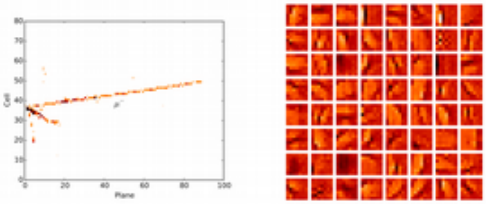
Using Machine Learning on FPGAs to Enhance Reconstruction Output

IRIS-HEP
Febraury 13th, 2019

Dylan Rankin [MIT]
On behalf of the hls4ml team

Introduction

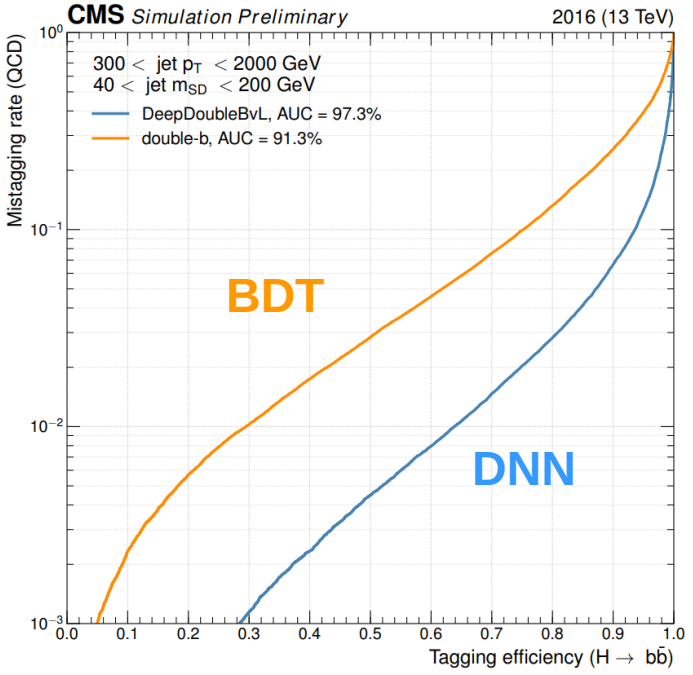
- Machine learning has become a common tool for broad spectrum of problems (industry & physics)
  - Particle/signal identification
  - Image/speech recognition

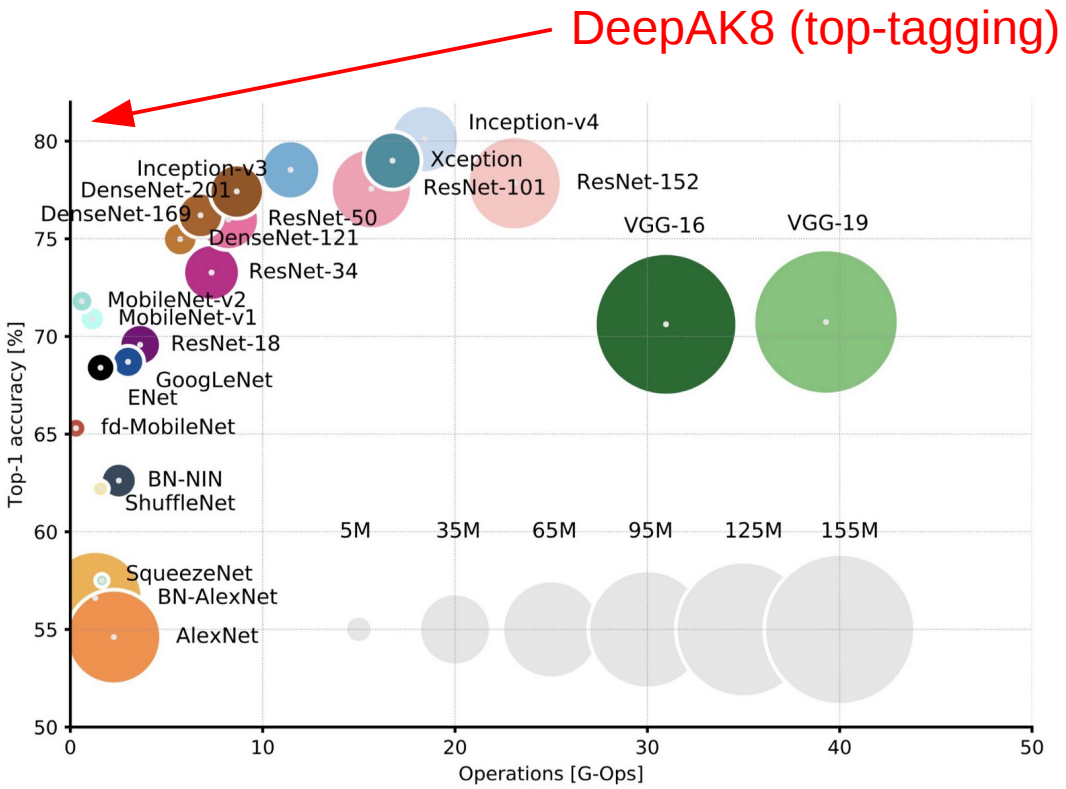


- Meanwhile, field-programmable gate arrays (FPGAs) have been used for decades to provide fast computing solutions
  - Development typically requires large initial investment (learning VHDL/Verilog, hardware cost)
  - Complex algorithms can be very difficult to implement
- ***hls4ml*** is a tool which facilitates implementing machine learning on FPGAs for fast inference [arXiv:1804.06913]
  - Provides possibility for highly customizable solutions to many HEP trigger problems

Machine Learning

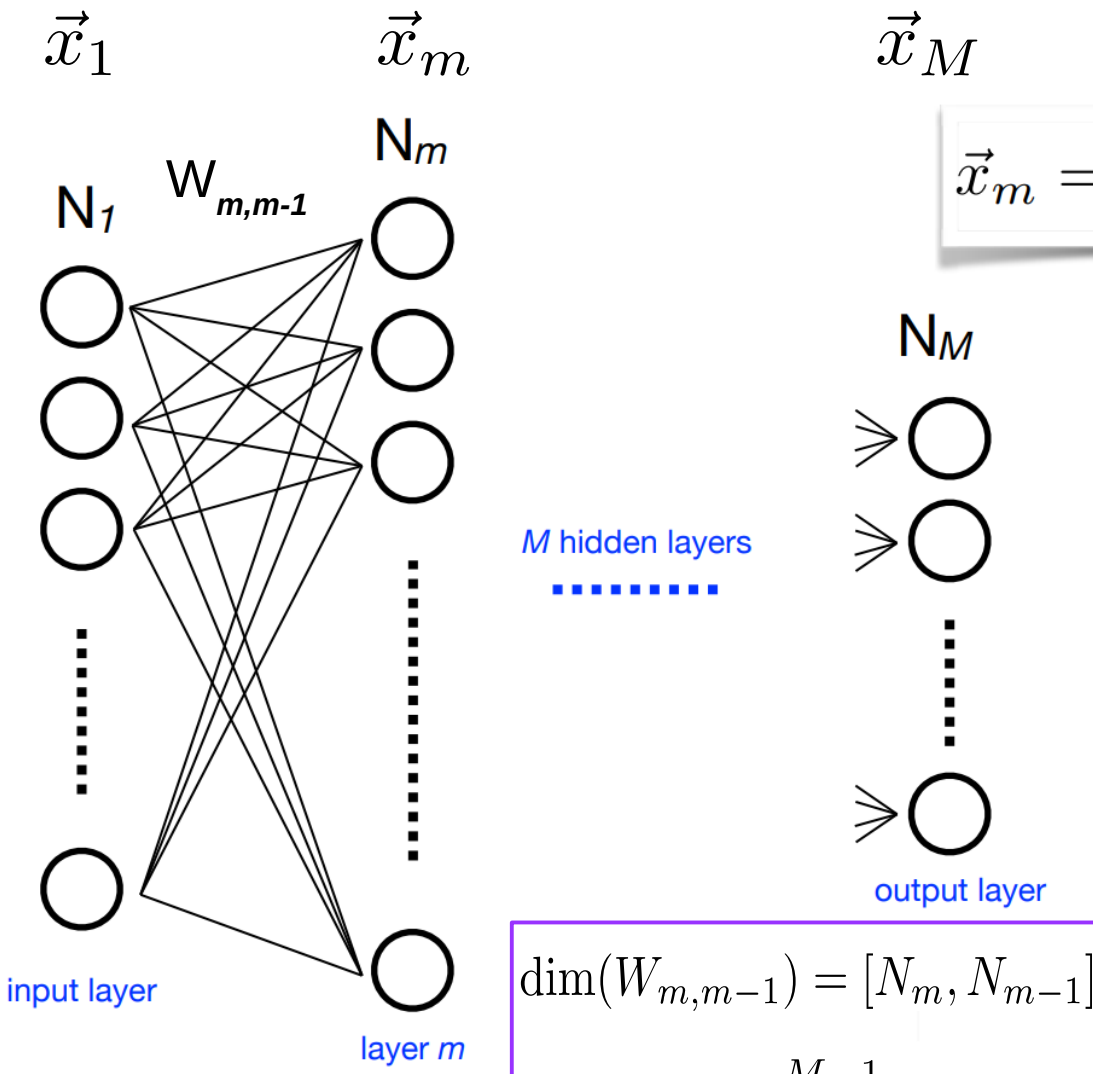
- Machine learning algorithms, especially deep neural networks, are becoming more and more common in HEP
  - Esp. LHC, neutrinos
- Provides capability to analyze very complex problems in straightforward way
- Very good performance even for difficult tasks
- Networks can become very large → long inference times





CMS-DP-2018-046

Neural Network



multiplication

$$\vec{x}_m = g_m\left(\mathrm{W}_{m,m-1}\vec{x}_{m-1} + \vec{b}_m\right)$$

activation function

addition

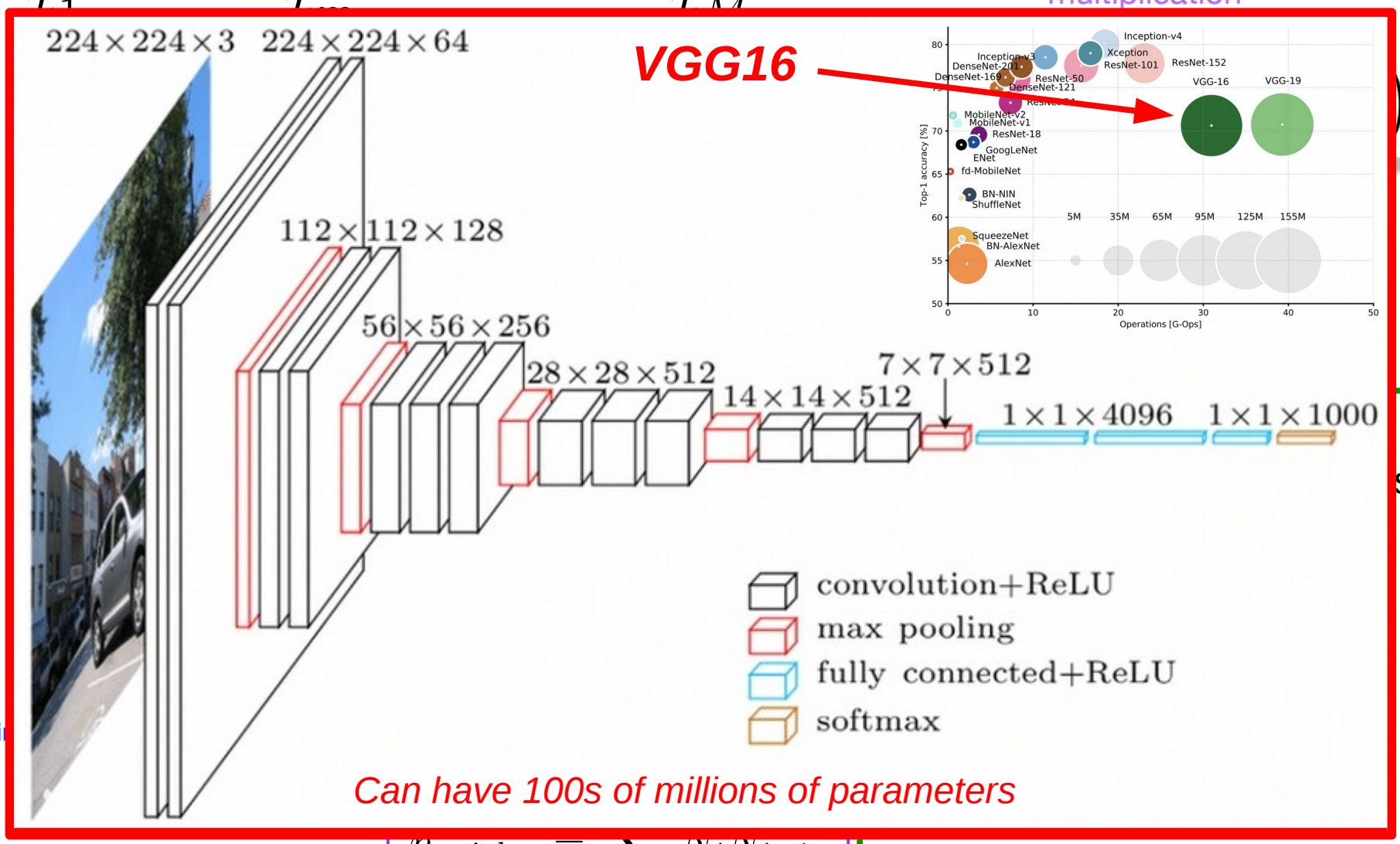
$$\dim(W_{m,m-1}) = [N_m, N_{m-1}]$$

$$n_{\text{weights}} = \sum_{i=0}^{M-1} N_i N_{i+1}$$

- Start with input vector (x_1)
- Using weight matrix (W), bias vector (b), and activation function (g), transform input vector to intermediate result vector (x_m)
  - Can be repeated many times
- Last layer provides output vector

Neural Network

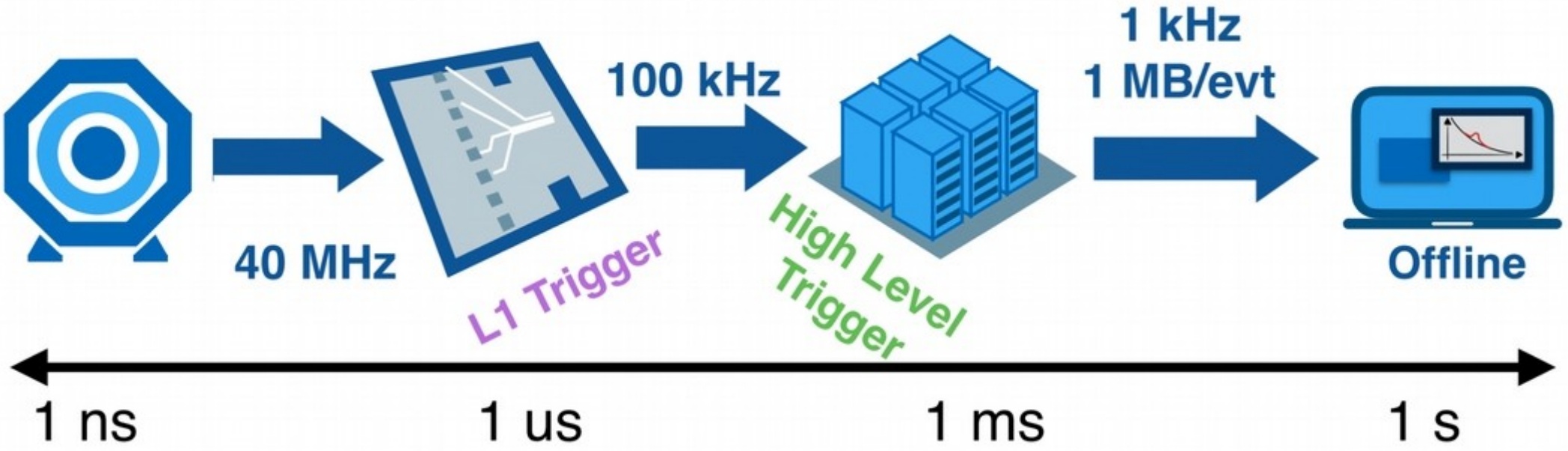
multiplication



Can have 100s of millions of parameters

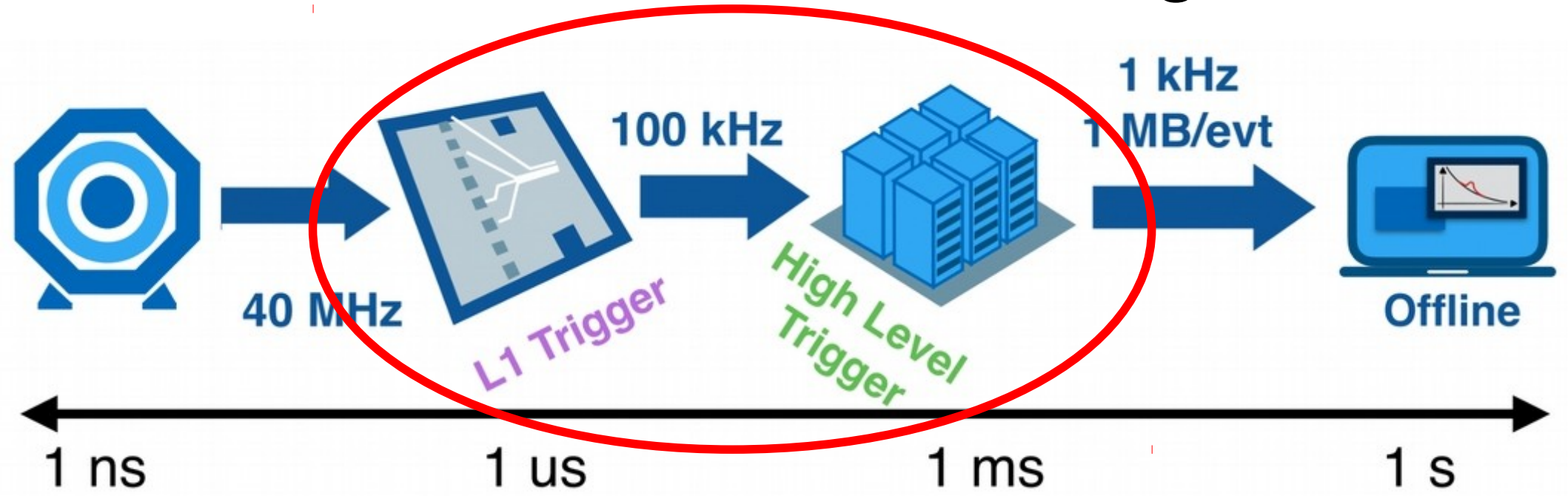
$$n_{\text{weights}} = \sum_{i=0} N_i N_{i+1}$$

LHC Data Processing



- **L1 Trigger** (hardware: FPGAs)
  - O(μs) hard latency. Typically coarse selection, BDT used for muon p_T assignment
- **HLT** (software: CPUs)
  - O(100 ms) soft latency. More complex algorithms (full detector information available), some BDTs and DNNs used
- **Offline** (software: CPUs)
  - > 1 s latencies. Full event reconstruction, bulk of machine learning usage in CMS

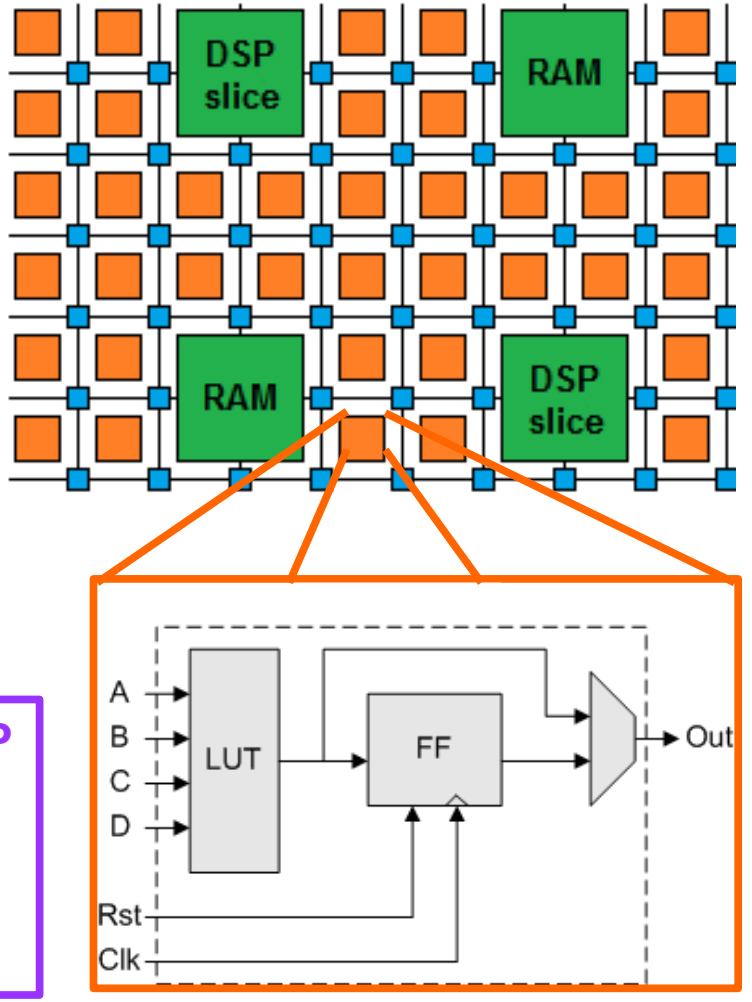
LHC Data Processing



- DNNs have the potential to greatly improve physics performance in the trigger system
- In order to implement an algorithm, need to ensure inference latencies of μs (ms) for L1 (HLT)
  - For L1, this means we *must* use FPGAs
- ***How can we run neural network inference quickly on an FPGA?***

FPGAs

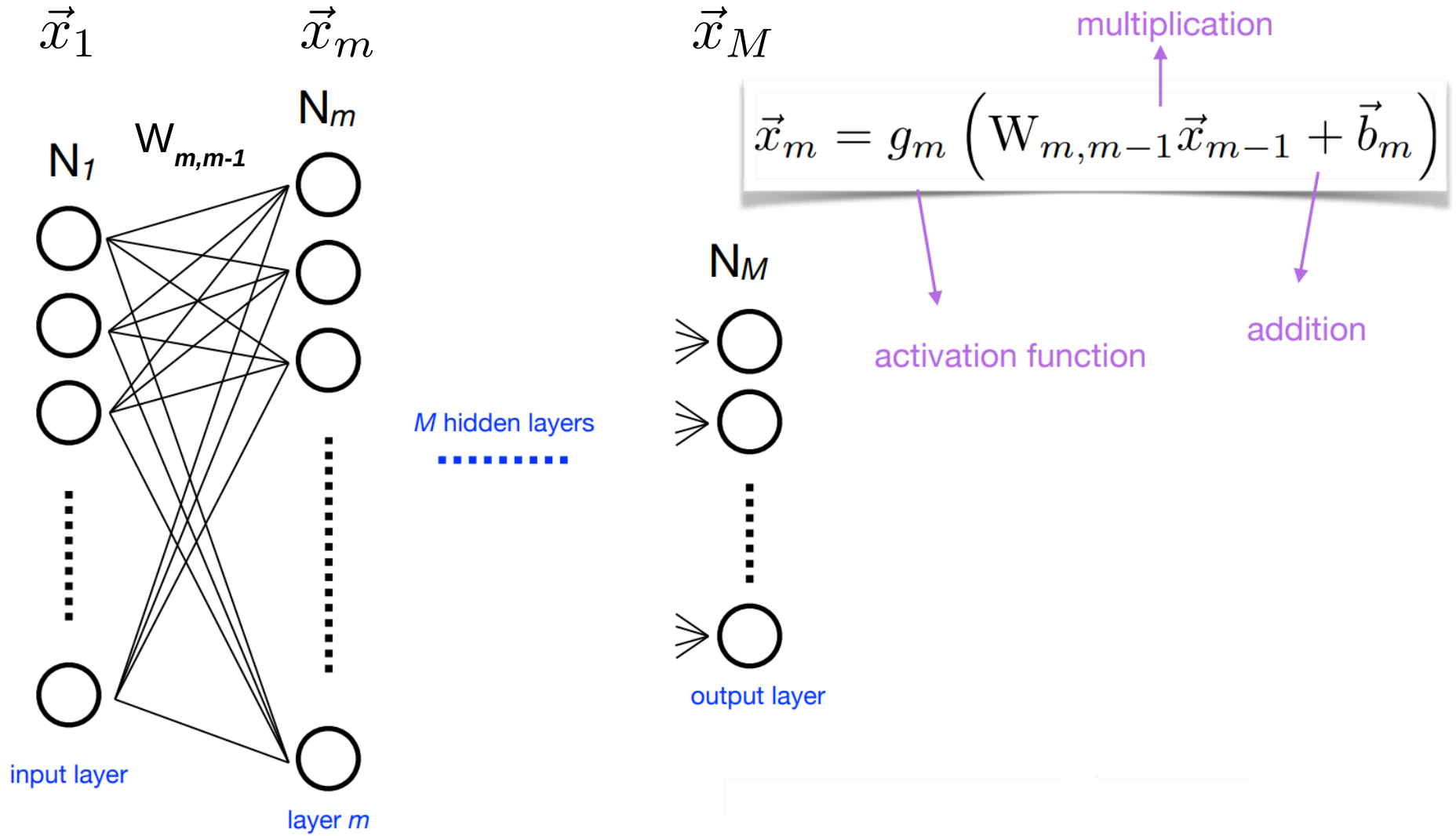
- Field-programmable gate arrays are a common solution for fast-computing
  - Ability to re-program for target needs is very appealing
- Building blocks:
  - **Multiplier units (DPSs) [arithmetic]**
  - **Look Up Tables (LUTs) [logic]**
  - **Flip-flops (FFs) [registers]**
  - **Block RAMs (BRAMs) [memory]**
- Algorithms are wired onto the chip
- Run at high frequency - *O(100 MHz)*
  - Can compute outputs in O(ns)
- Programming traditionally done in Verilog/VHDL
  - Low-level hardware languages
- Possible to translate C to Verilog/VHDL using High Level Synthesis (HLS) tools



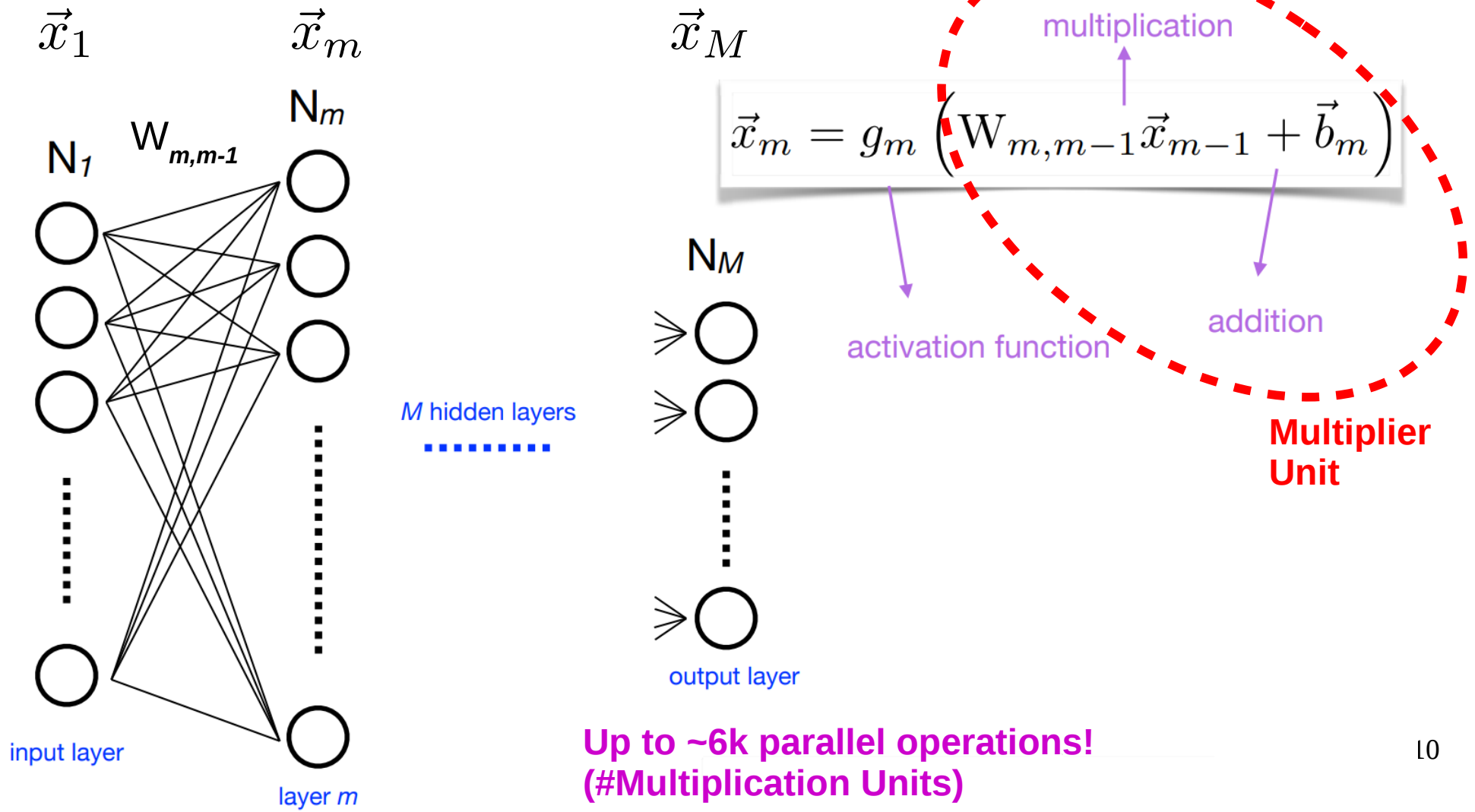
Virtex 7 XC7VX690T
3600 Multipliers
400K LUTs
800K FFs
10 Mb BRAM

Virtex Ultrascale+ VU9P
6800 Multipliers
1M LUTs
2M FFs
75 Mb BRAM

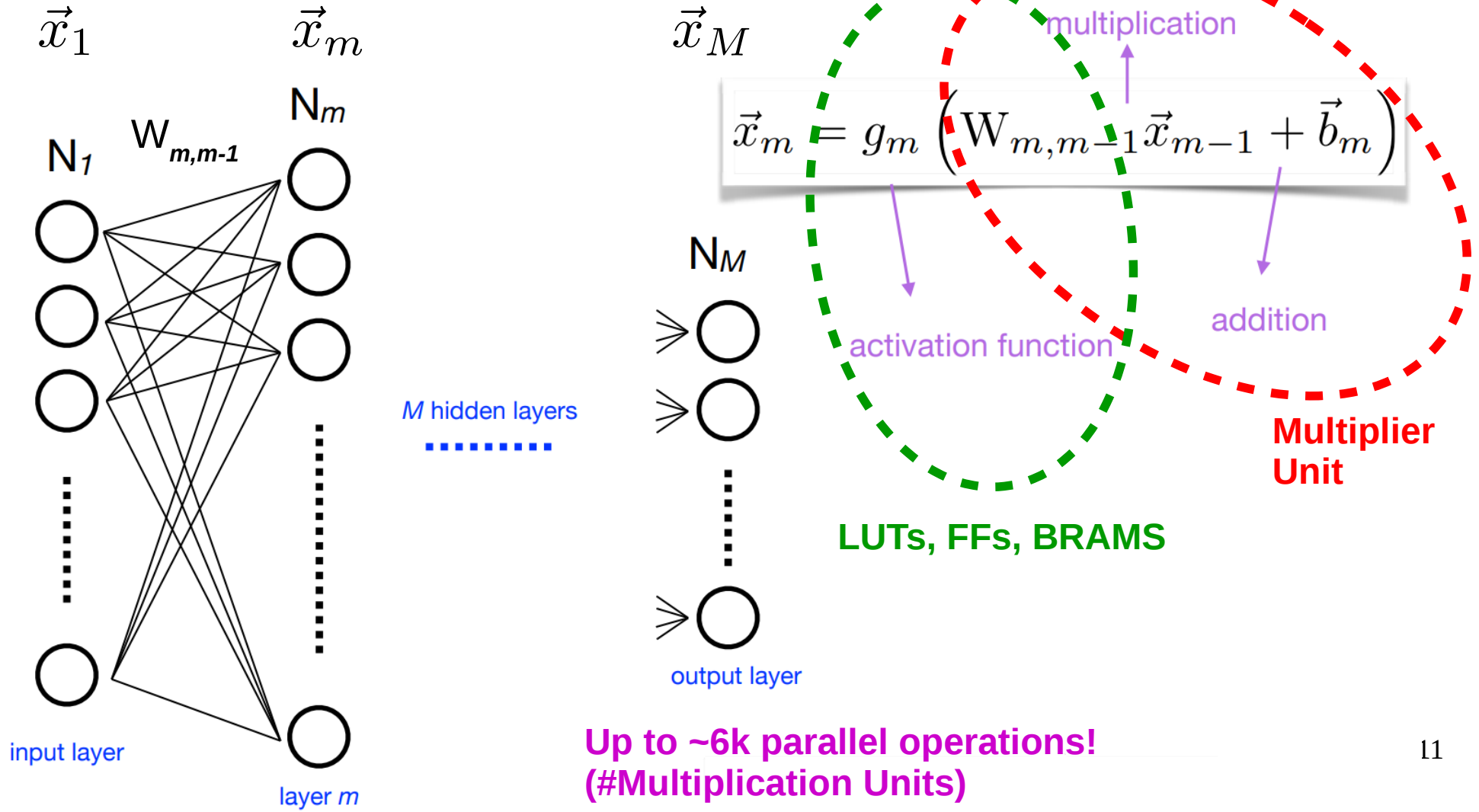
Inference on an FPGA



Inference on an FPGA

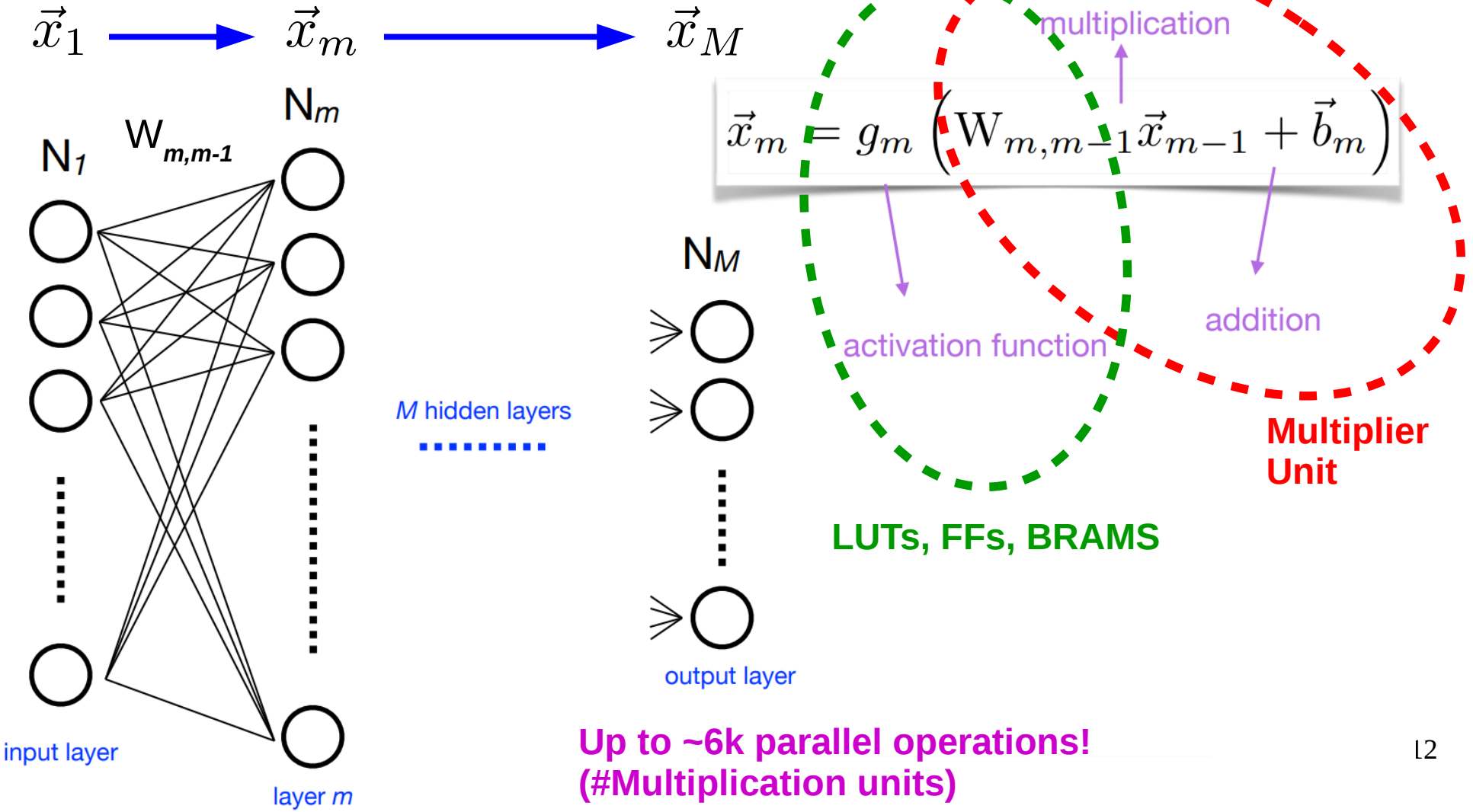


Inference on an FPGA



Inference on an FPGA

Every clock cycle
(all layer operations can be performed simultaneously)



Up to ~6k parallel operations!
(#Multiplication units)

- ***hls4ml*** is a software package for creating HLS implementations of neural networks
  - https://hls-fpga-machine-learning.github.io/hls4ml/
- Supports common layer architectures and model software
- Highly customizable output for different latency and size needs
- Simple workflow to allow quick translation to HLS

Project Configuration (Keras)

keras-config.yml

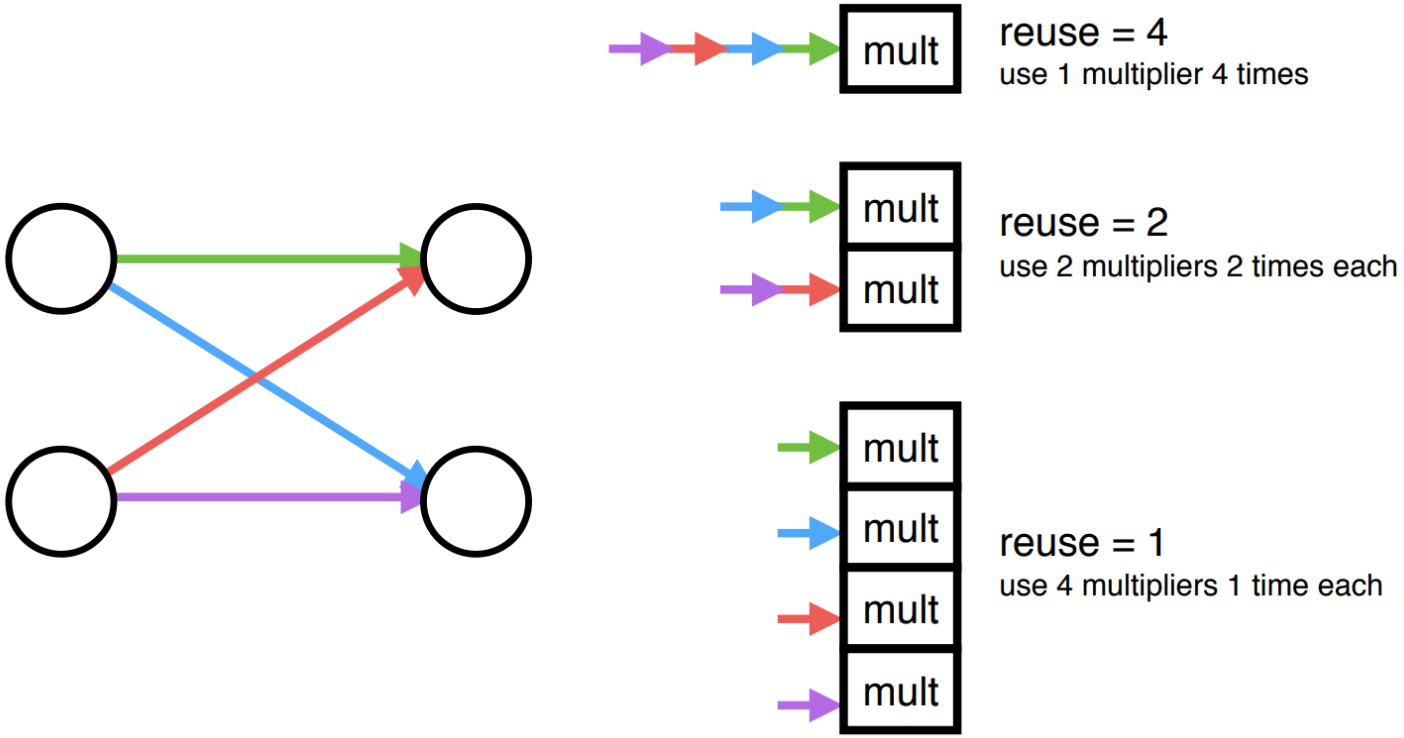
```
KerasJson: example-keras-model-files/KERAS_1layer.json
KerasH5: example-keras-model-files/KERAS_1layer_weights.h5
OutputDir: my-hls-test
ProjectName: myproject
XilinxPart: xcku115-flvb2104-2-i
ClockPeriod: 5

IOType: io_parallel # options: io_serial/io_parallel
ReuseFactor: 1
DefaultPrecision: ap_fixed<16,6>
```

- Configuration file takes model architecture and weights files as input
- Main customization options:
  - *ReuseFactor:* calculations per multiplier per layer (parallelization)
  - *DefaultPrecision:* used for inputs, weights, biases

```
python keras-to-hls.py -c keras-config.yml
```

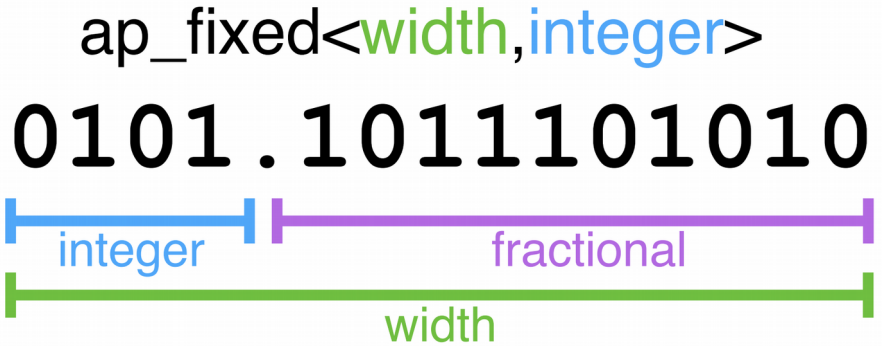
Customization: Reuse



- For lowest latency, compute all multiplications for a given layer at once
  - Reuse = 1 (fully parallel) → latency ≈ # layers
- Larger reuse implies more serialization
  - Reuse = # weights (fully serialized) → latency = (# weights) x (# layers)
- Allows trading higher latency for lower resource usage

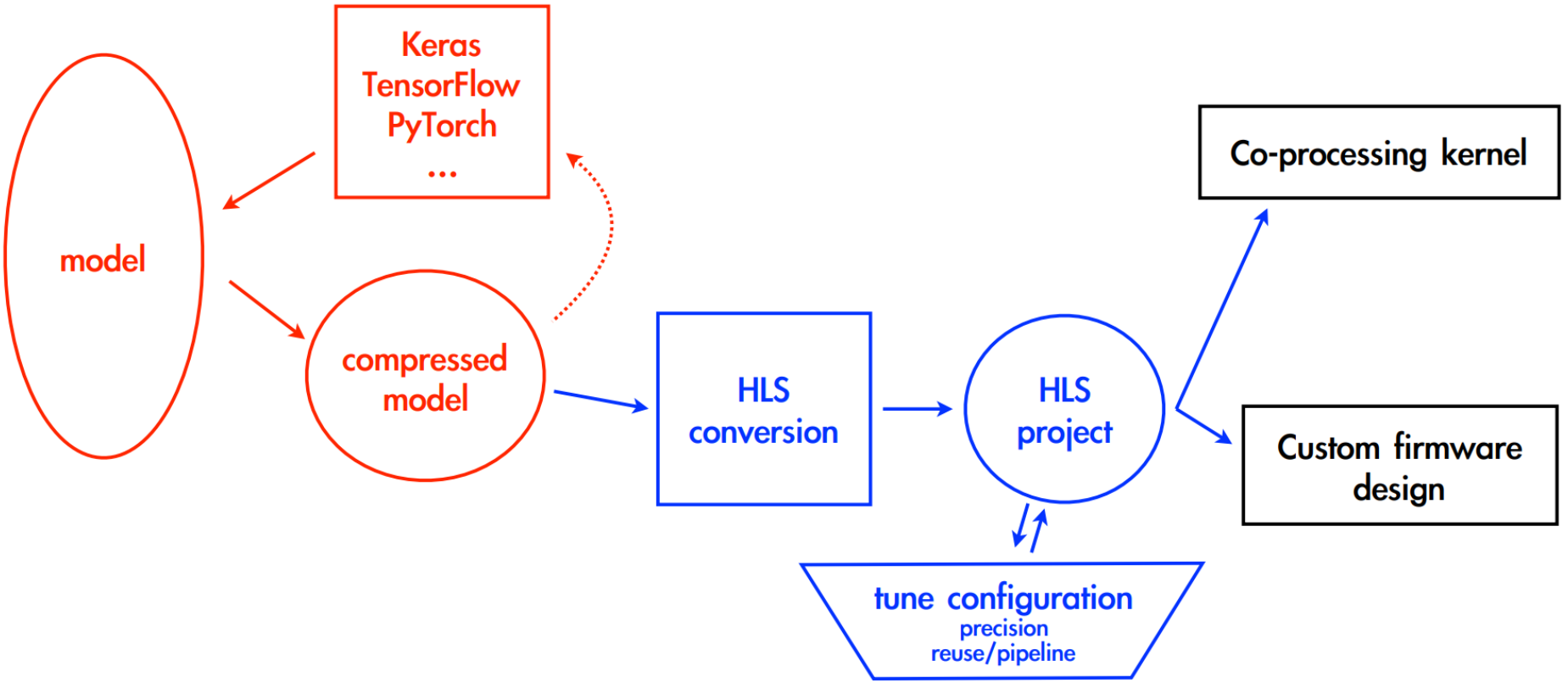
Customization: Precision

- `hls4ml` uses fixed point classes for all computations



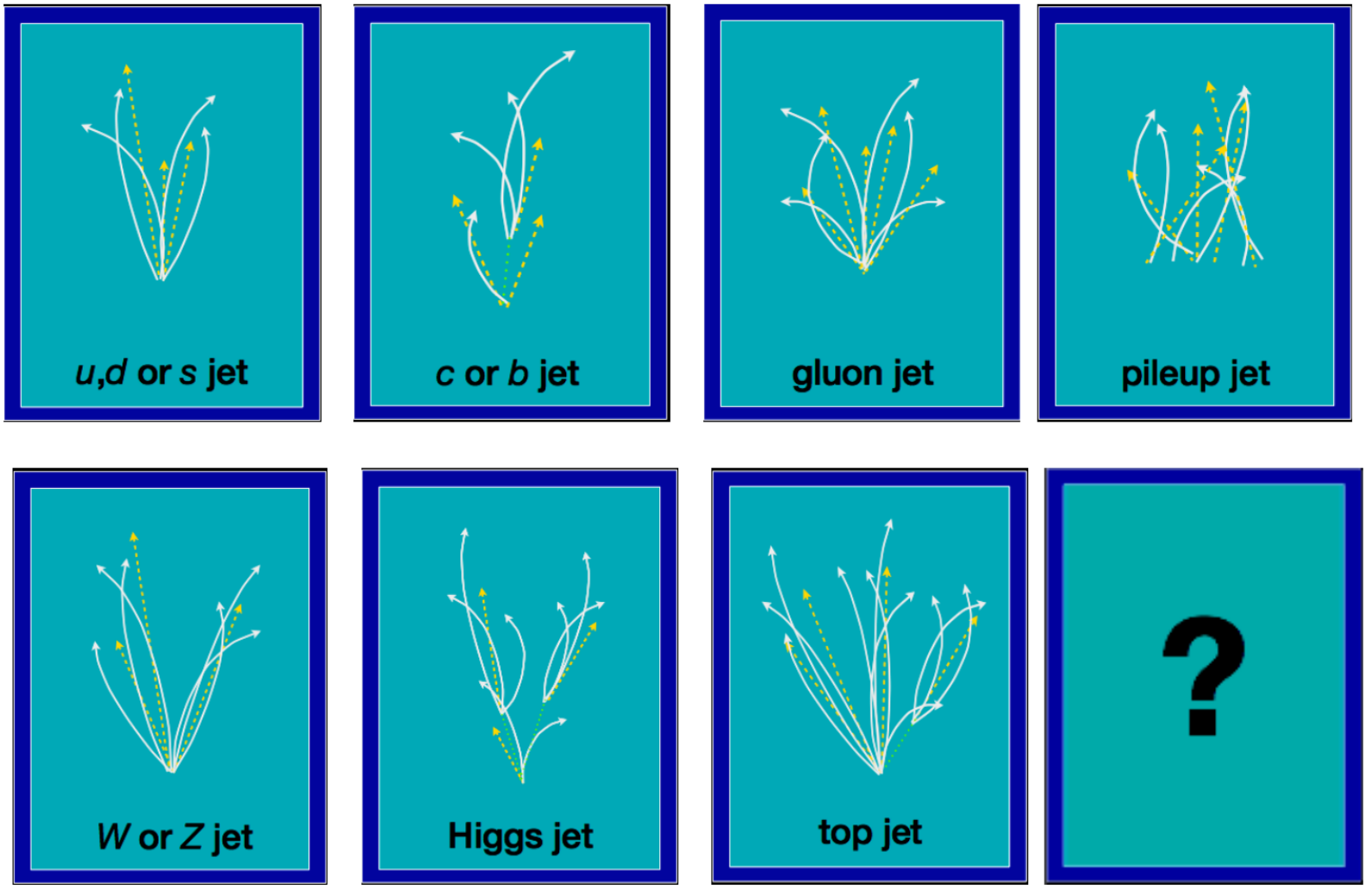
- Precision can be adjusted as needed for desired accuracy, performance
  - Also impacts resource usage
- Default behavior is to use same precision for all layers

Design Workflow



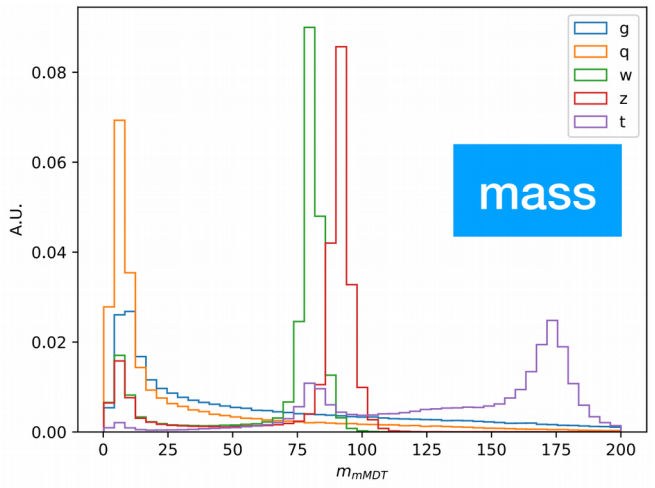
- Design model with standard software tools (Keras, Tensorflow, PyTorch)
- Pass network architecture and weights/biases along with configuration parameters to `hls4ml` (creates HLS project)
- Interface HLS code with desired project

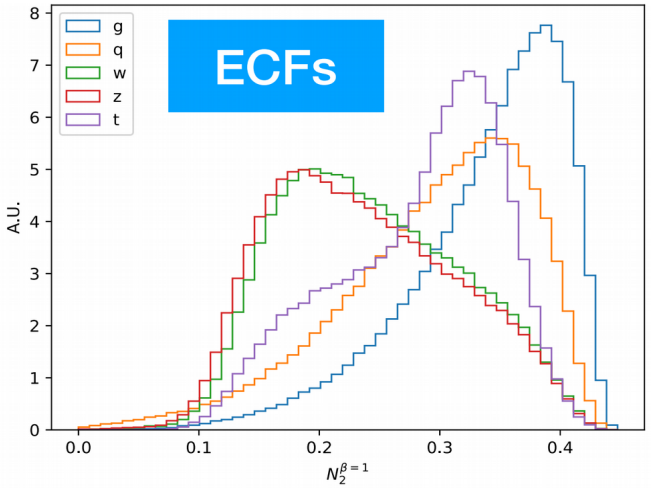
Jet Classification Example

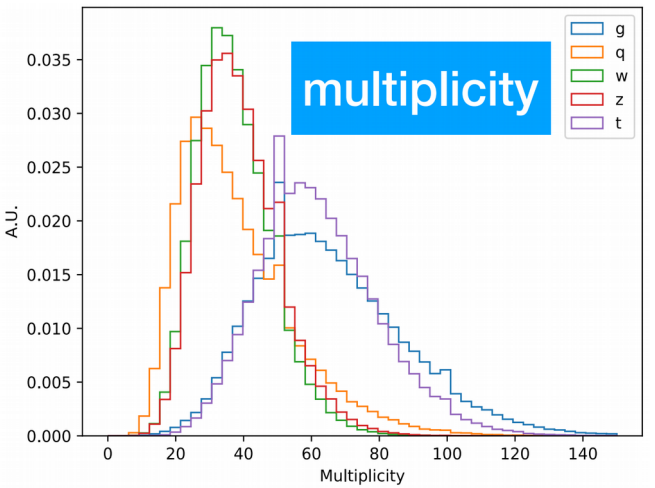


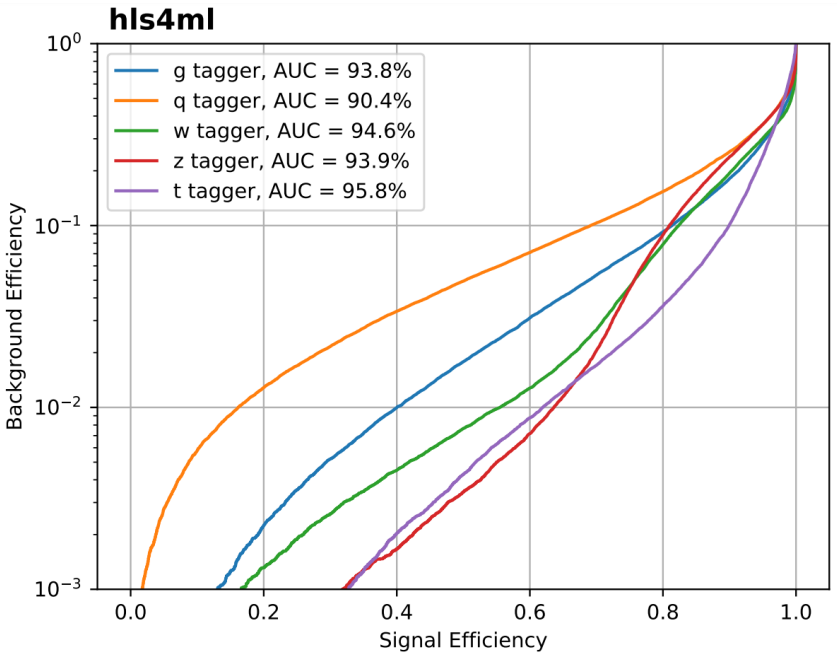
- Perhaps an unrealistic example for L1 trigger, lessons are useful
- Problem certainly a clear candidate for ML usage

Example Network



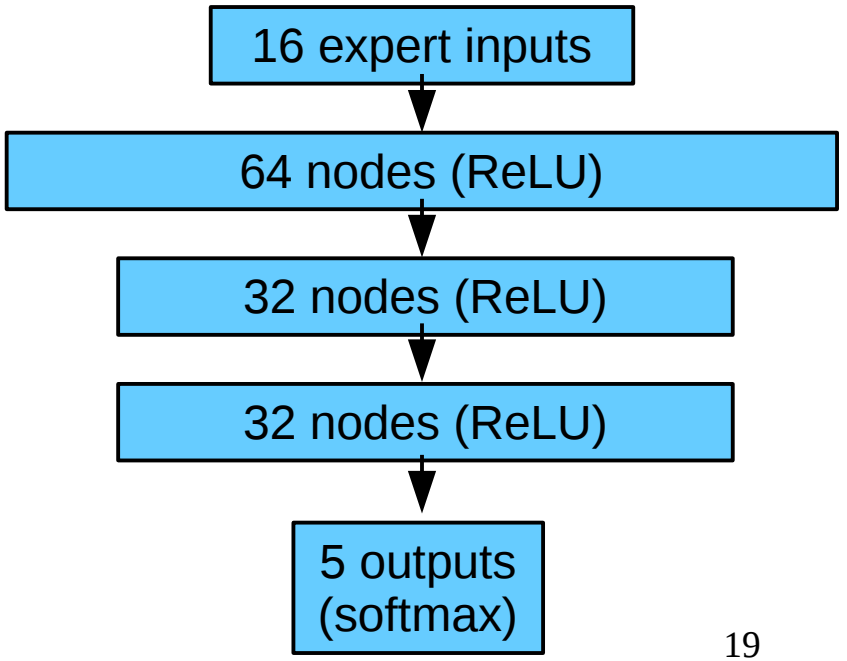






Observables

m_{mMDT}
$N_2^{\beta=1,2}$
$M_2^{\beta=1,2}$
$C_1^{\beta=0,1,2}$
$C_2^{\beta=1,2}$
$D_2^{\beta=1,2}$
$D_2^{(\alpha,\beta)=(1,1),(1,2)}$
$\sum z \log z$
Multiplicity

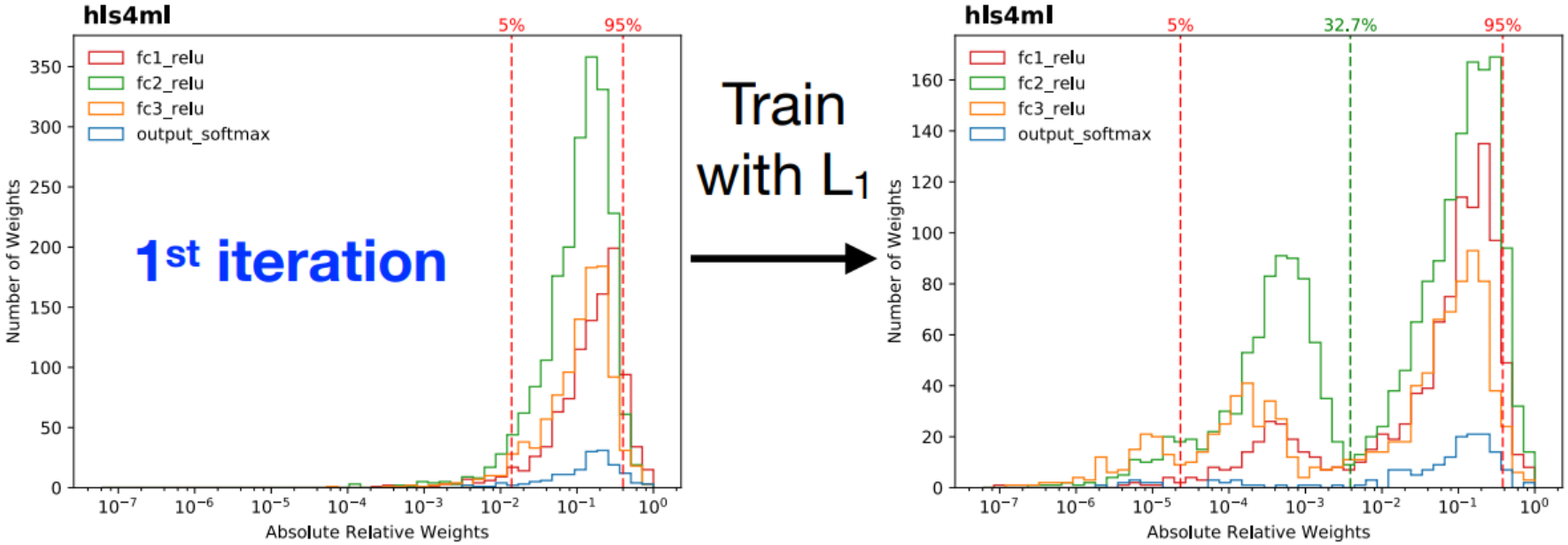


5k weights → 5k multipliers

Reducing Network Size: Compression

- *Compression*
  - Removing nodes or connections from network
- To identify redundant connections, we use a method of successive retraining and weight minimization (pruning)
  - Use L1 regularization, modify loss function with penalty term for large weights
  - Remove smallest weights
  - Repeat
- HLS automatically removes multiplications by 0!

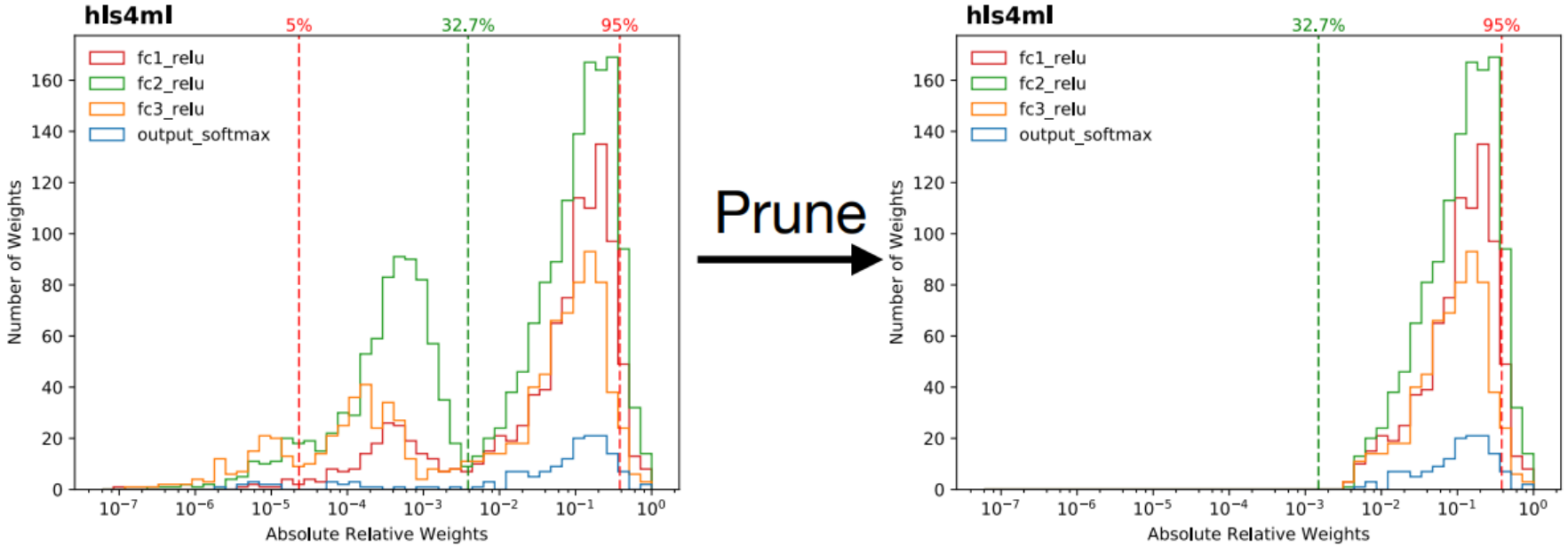
$$L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \lambda\|\mathbf{w}\|$$



Reducing Network Size: Compression

- *Compression*
  - Removing nodes or connections from network
- To identify redundant connections, we use a method of successive retraining and weight minimization (pruning)
  - Use L1 regularization, modify loss function with penalty term for large weights
  - Remove smallest weights
  - Repeat
- HLS automatically removes multiplications by 0!

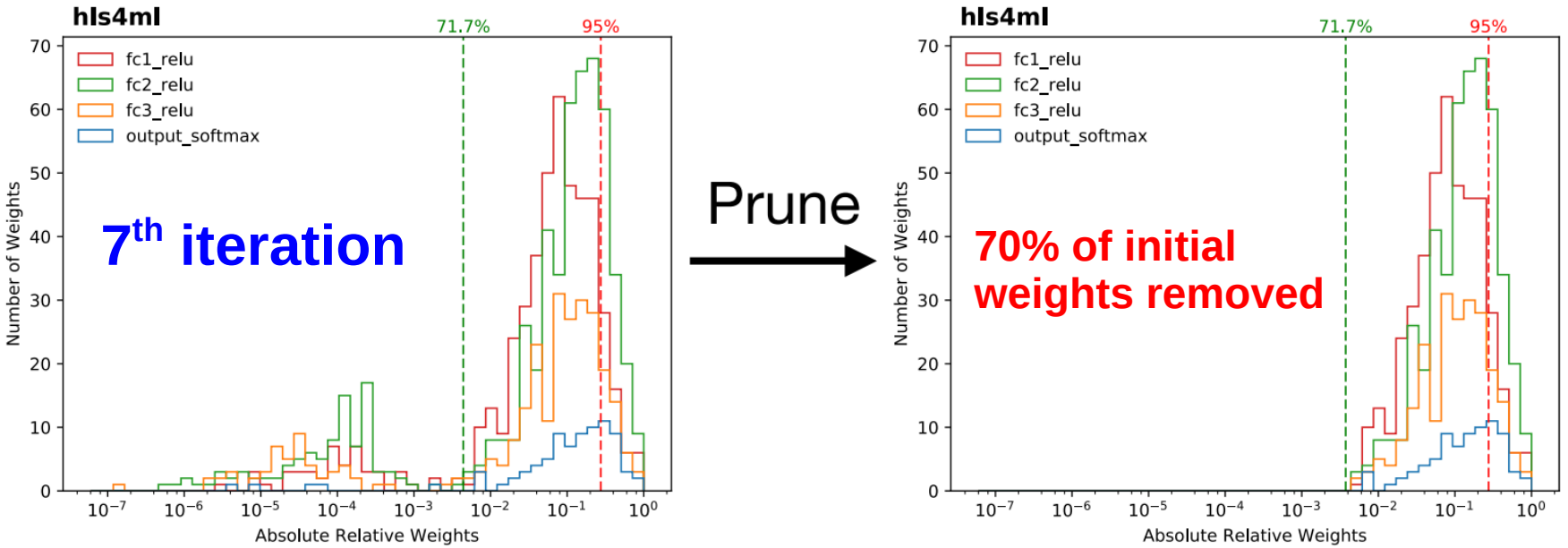
$$L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \lambda\|\mathbf{w}\|$$



Reducing Network Size: Compression

- *Compression*
  - Removing nodes or connections from network
- To identify redundant connections, we use a method of successive retraining and weight minimization (pruning)
  - Use L1 regularization, modify loss function with penalty term for large weights
  - Remove smallest weights
  - Repeat
- HLS automatically removes multiplications by 0!

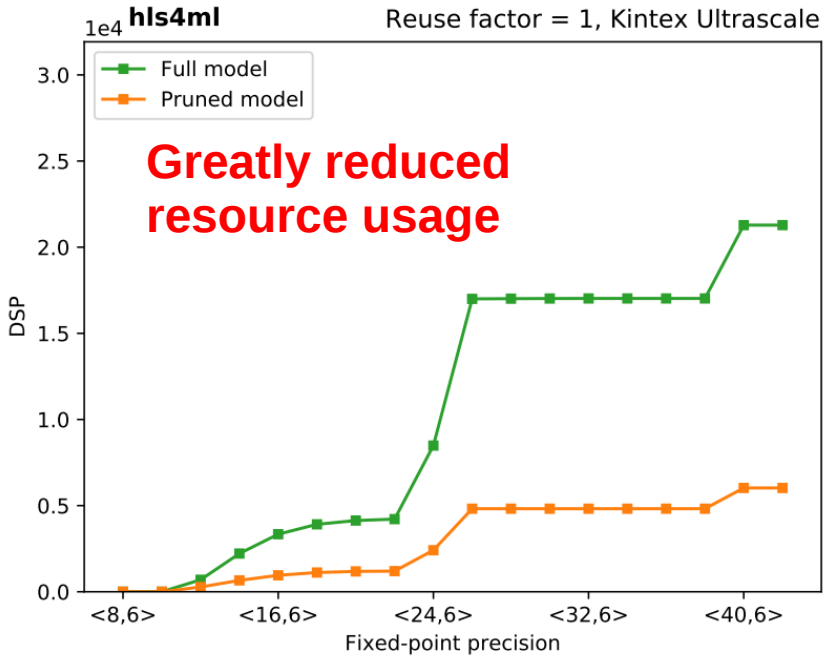
$$L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \lambda\|\mathbf{w}\|$$

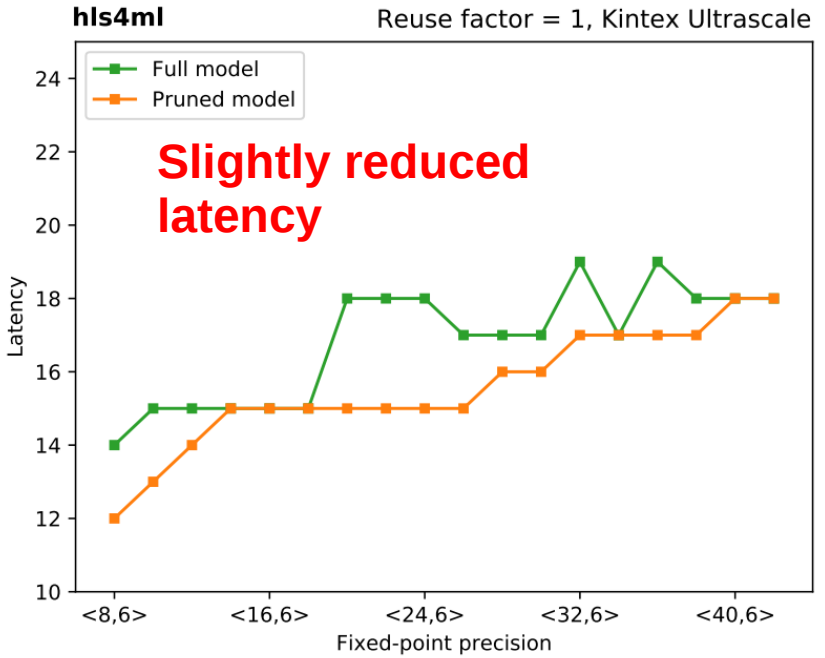


Reducing Network Size: Compression

- *Compression*
  - Removing nodes or connections from network
- To identify redundant connections, we use a method of successive retraining and weight minimization (pruning)
  - Use L1 regularization, modify loss function with penalty term for large weights
  - Remove smallest weights
  - Repeat
- HLS automatically removes multiplications by 0!

$$L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \lambda\|\mathbf{w}\|$$





Reducing Network Size: Quantization

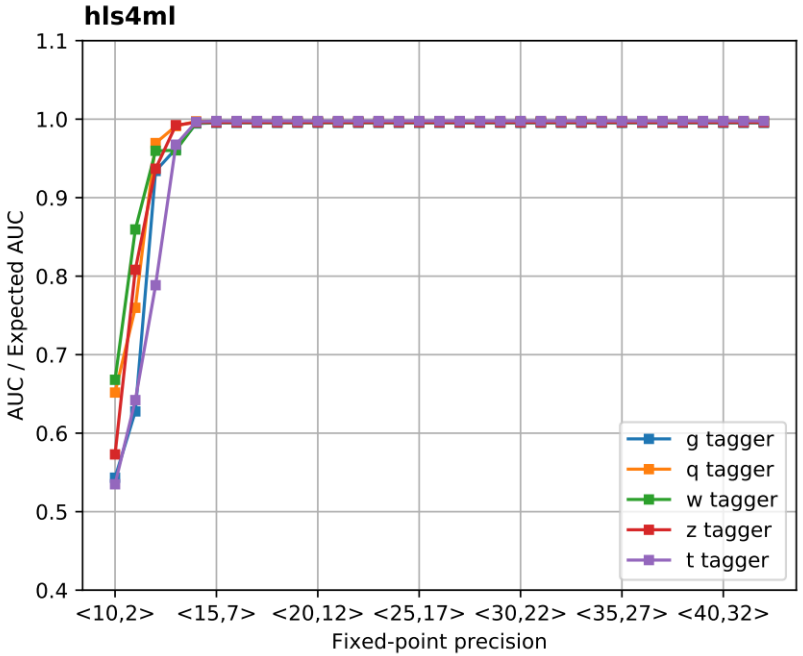
- *Quantization*
  - Reducing the bit precision used for NN arithmetic
- Software assumes all computations performed with floating point arithmetic
  - Not always necessary for desired performance
- Reduction of precision automatically zeros very small weights ($w < 2^{-fractional}$)
  - Also reduces resources needs to compute/store multiplications and intermediate layers
- Full performance at 8 integer bits, 8 fractional bits

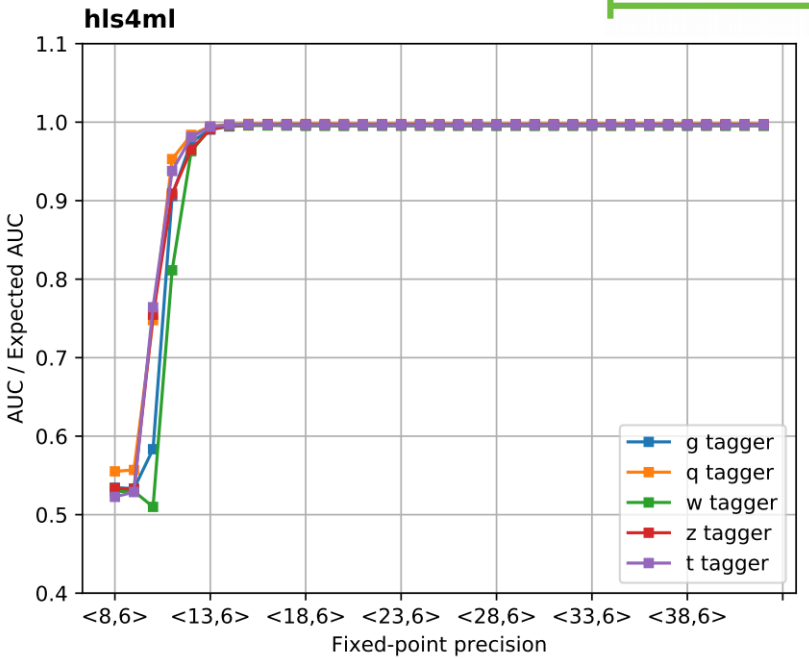
ap_fixed<width,integer>

0101.1011101010

integer | fractional

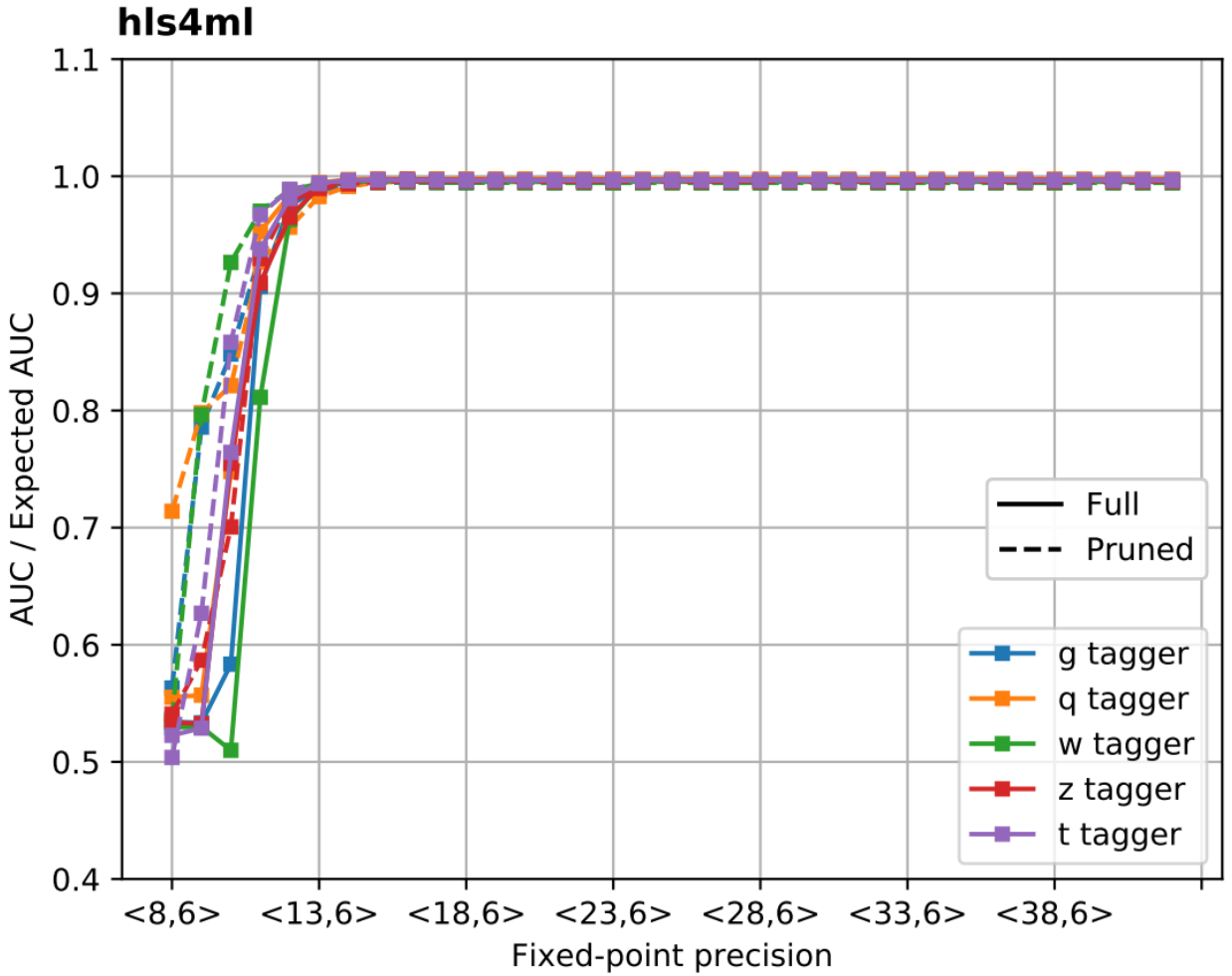
width



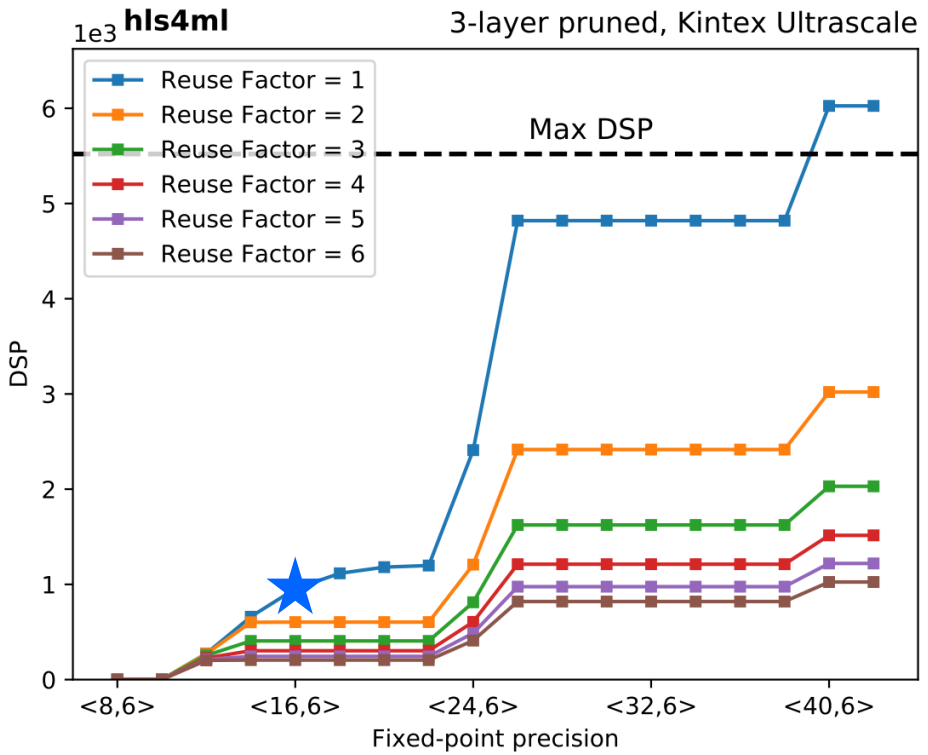


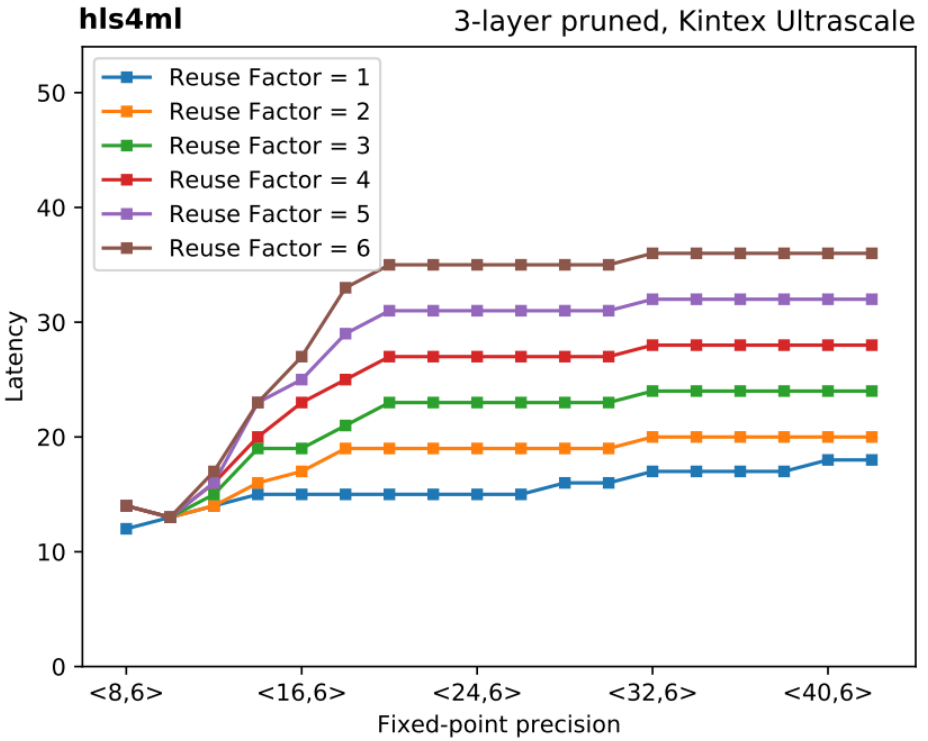
Network Tuning

- Compression & quantization can be used together, maintain full performance

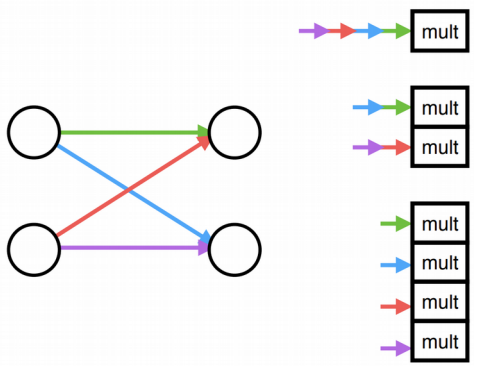


Network Tuning: Reuse

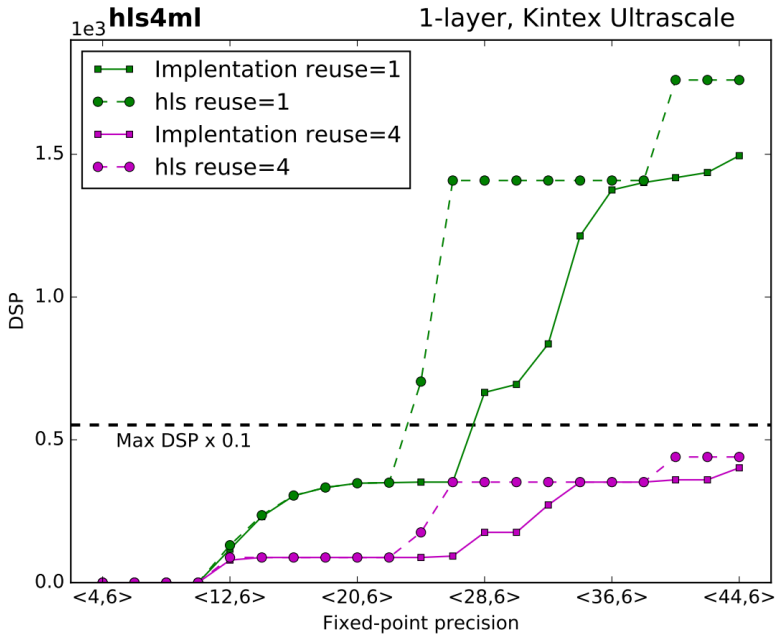


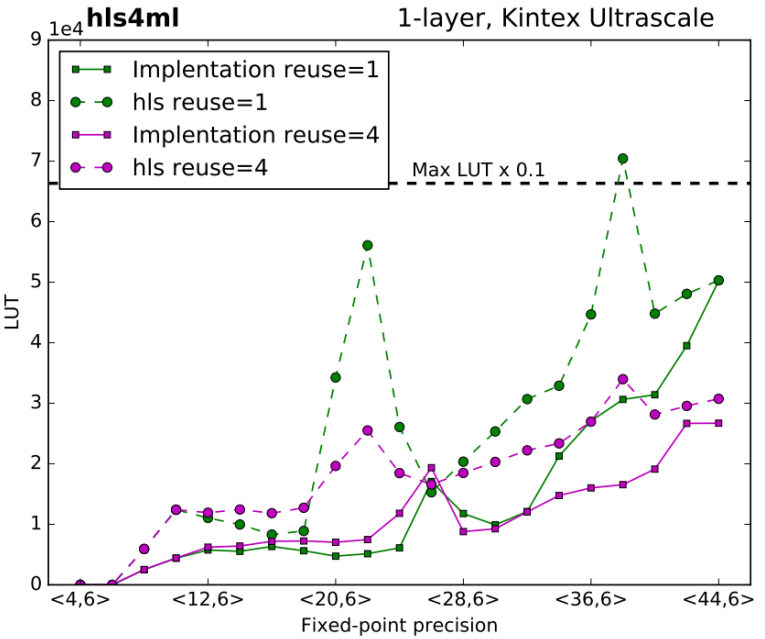


- Reduces multiplier usage at the cost of increasing latency (and initiation interval)
  - Scales as expected
- Minimal effect of reuse on LUTs and FFs
- **For reuse = 1, <16,6> precision**, find total resource usage well below available resources for target FPGA (KU115)



Synthesis vs. Implementation

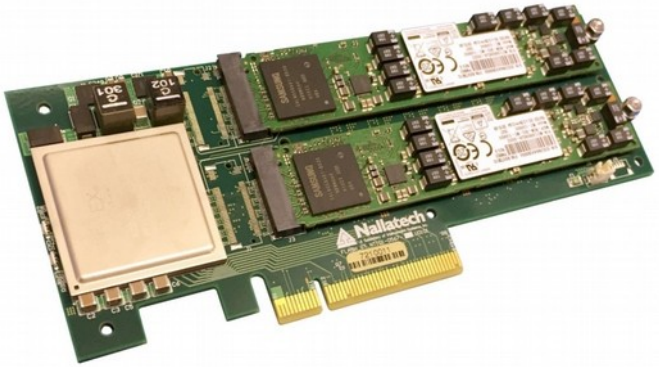




- All previous results come from HLS synthesis estimates
- Known differences between HLS estimates and final implementation
- For slightly smaller model (this slide):
  - FF/LUT – overestimated in most cases
  - Multipliers – accurate below max width of multiplier input, overestimated above
- Also able to meet timing constraints

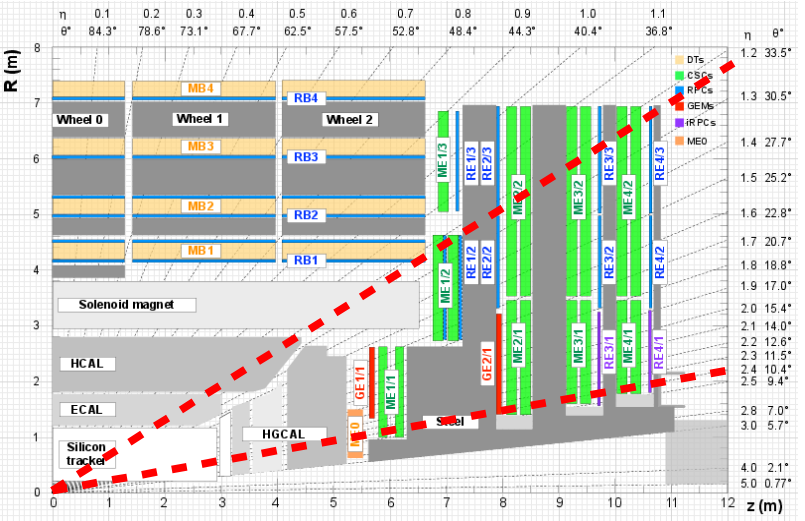
Under Development

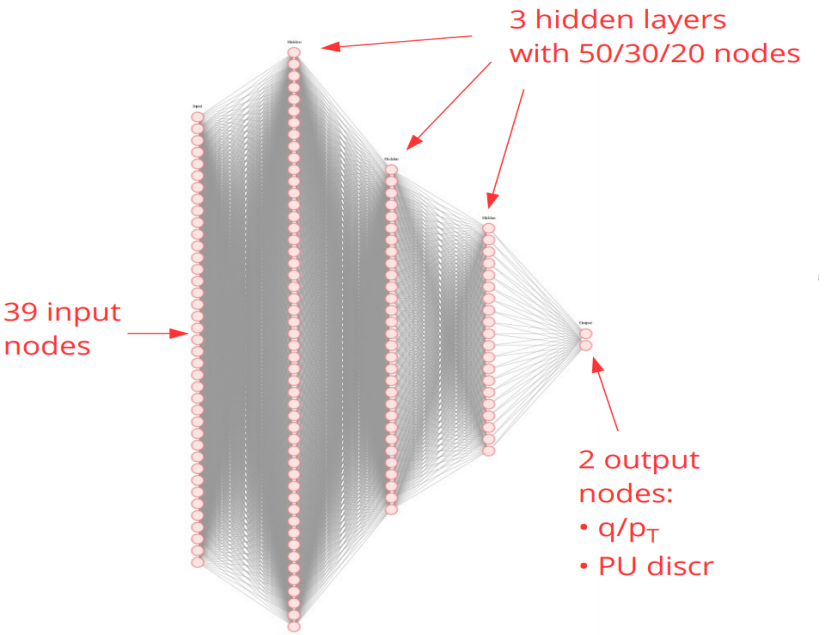
- Large amount of ongoing development with `hls4ml`
- Expanding tool capabilities
  - Working on adding full support for:
    - Conv1D and Conv2D layers (partial support)
    - LSTM/GRU (testing)
    - Graph neural networks (prototyping)
    - Binary/ternary dense networks (partial support)
    - Pooling (prototyping)
    - Boosted decision trees (testing)
  - Working on ability to handle larger networks
  - Stay tuned for updates!
- Multiple potential use cases for LHC trigger systems



ML in the CMS Trigger

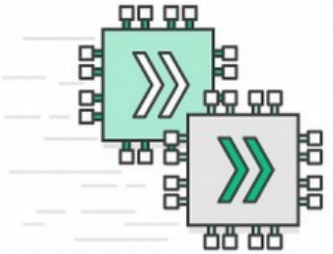
- CMS endcap muon trigger uses BDT for muon p_T assignment
  - Large DRAM bank on-board to implement LUT
- Work ongoing to investigate replacing BDT with DNN
  - Developing with hls4ml
  - CPAD 2018 Talk
- Can use two output nodes to simultaneously do p_T assignment and PU discrimination
- Implementation fits comfortably in a VU9P
  - Algorithm latency is 72 ns
  - 41% DSP usage
- Also work done for CMS trigger upgrade using hls4ml for tau lepton ID
  - Range of additional applications for particle ID





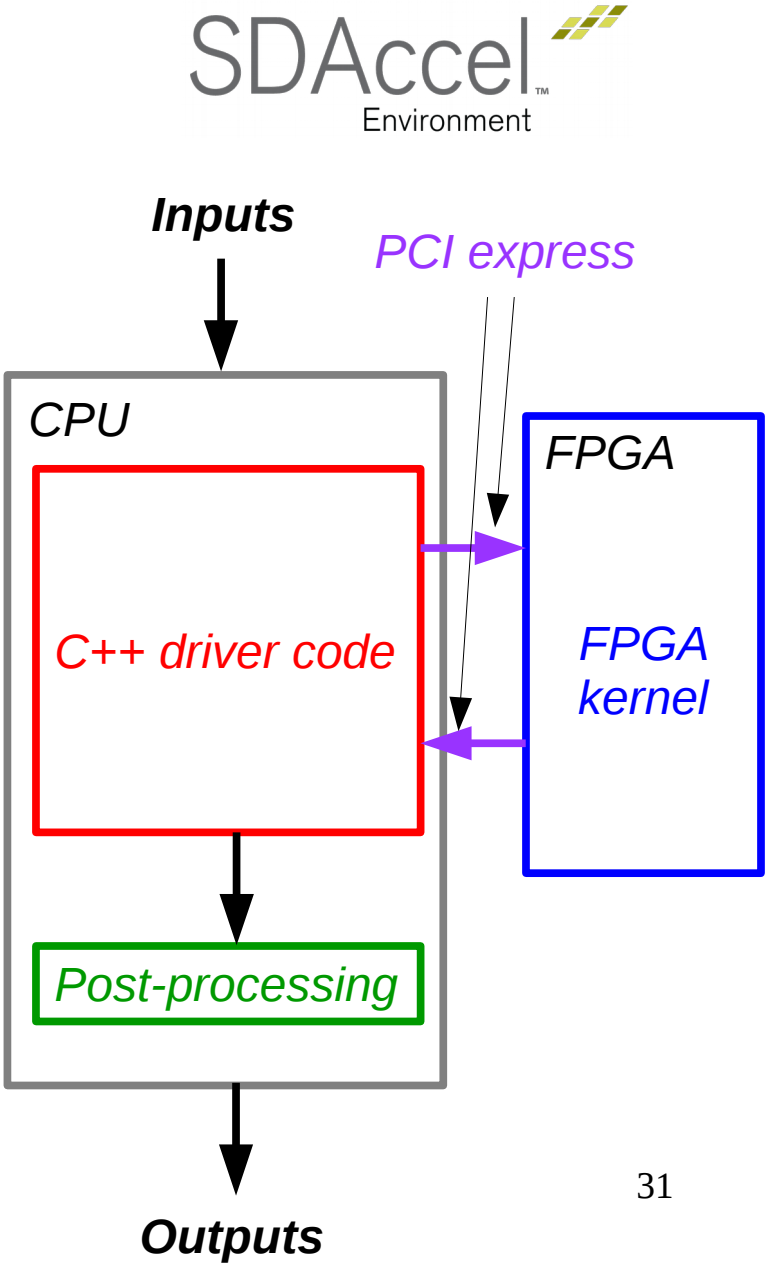
Co-processors

- Increasing popularity of co-processor systems
  - CPU connected to a FPGA/GPU/TPU
  - Common setup for FPGA connects to CPU through PCI-express
- Allows algorithms to run on the most optimized hardware (“acceleration”)
- FPGA-CPU co-processor machines are available as an offering on Amazon Web Services (AWS)
  - F1 instances (connected to a Virtex Ultrascale+ VU9P) can be used to explore possibilities for accelerated inference



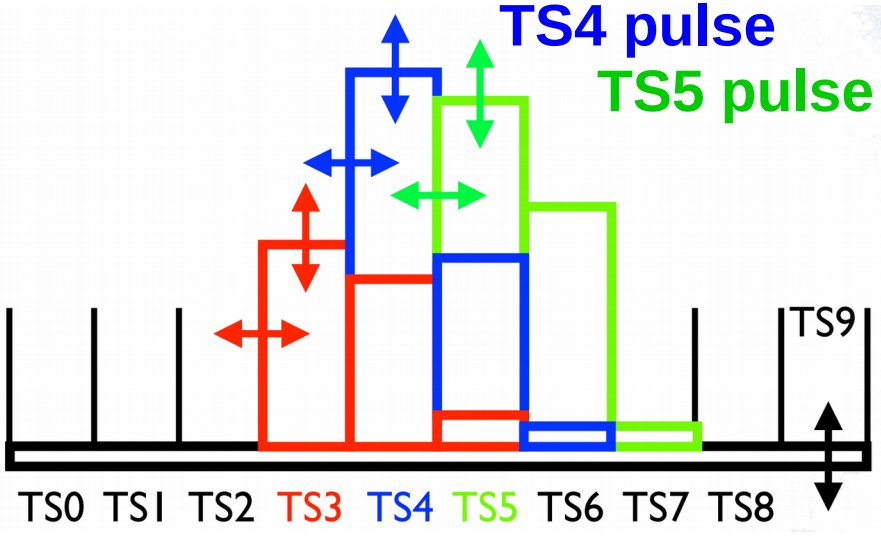
Acceleration with AWS

- Development for FPGA kernel and CPU host code is done with SDAccel environment
  - Invokes Vivado HLS under the hood, produces traditional synthesis reports etc.
- Run host code on CPU, manages data transfer and FPGA kernel execution
- `hls4ml` project only needs to be wrapped to provide specific inputs/outputs for SDAccel to interface properly
  - Can be done generically
  - Have accelerated variety of hls4ml projects on AWS F1
- Limited in speed by I/O bandwidth

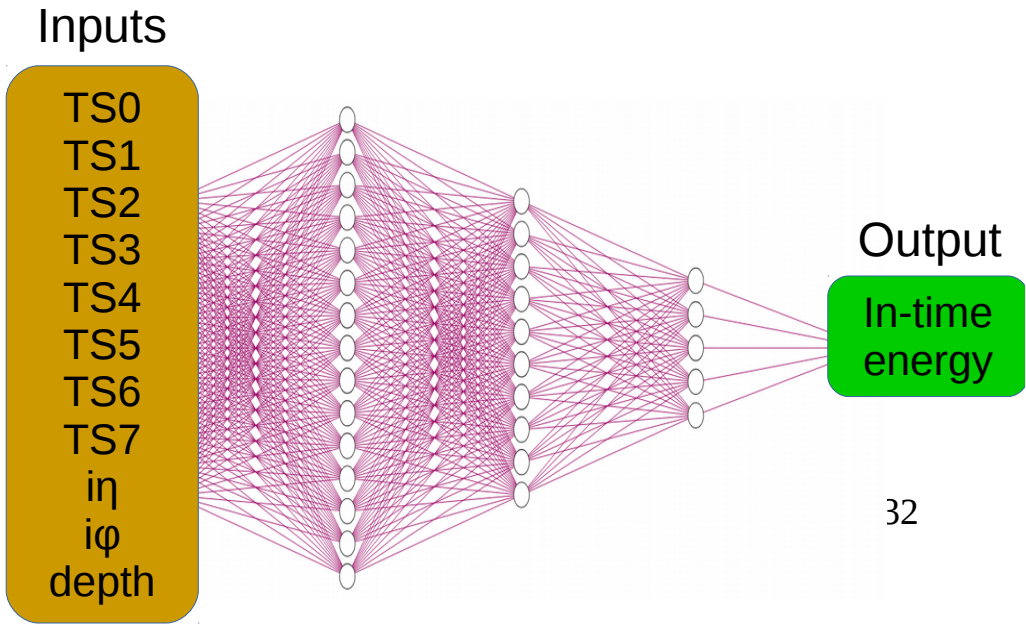


An Acceleration Case Study (1)

- HCAL reconstruction at CMS currently uses an iterative fit to extract in-time energy
  - 15% of HLT processing time
- Similar procedure used for ECAL reconstruction
  - 10% of HLT processing time
- Situation expected to worsen in more intense conditions
- We have begun investigating machine learning alternatives
  - Similar inputs to current algorithm (energies + η/φ/depth)
  - Comparable performance with simple network (3 layers, 15/10/5 nodes)



vs



An Acceleration Case Study (2)

- Have successfully implemented/run the network inference on AWS using hls4ml/SDAccel
- https://github.com/drankincms/AccelFPGA
- Including data transfer to/from CPU, whole *FPGA inference process takes **2 ms*** for all 16k HCAL channels
  - Inference alone takes 80 us (70 ns for one inference)
- Has been tested inside standard CMS software code environment, using high-level trigger job
  - Every event sends input features to FPGA, waits for callback
- *Iterative fit procedure takes **50 ms*** for same inputs
- FPGA inference is a fixed-latency procedure, iterative fit is not
- Inference on CPU or GPU also significantly faster than iterative fit
  - FPGA inference fastest

Algorithm	***Architecture***	***Time/event (ms)***
Iterative fit	***CPU***	**50**
NN Inference	***CPU***	**15**
NN Inference	***GPU***	**12**
NN Inference	***FPGA***	**2**

hls4ml/SDAccel Workshop

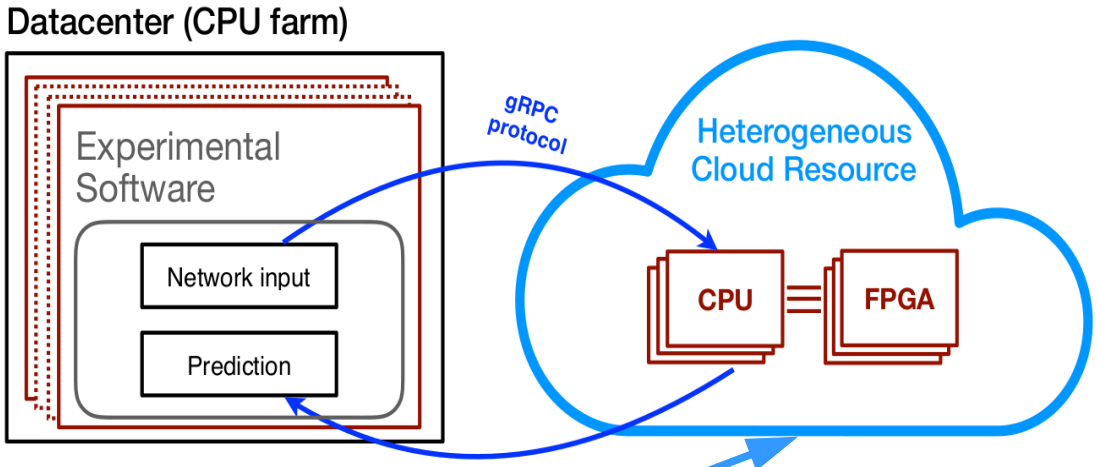
- Organized workshop for hls4ml and acceleration last week
  - *How to do ultrafast DNN inference on FPGAs*
  - https://indico.cern.ch/event/769727/
- Lots of interest in across many HEP experiments, industry
  - 90 participants
- By the end of the course all participants were able to actually run inference on FPGAs
  - Used AWS to provide machines for development/acceleration
- Interested in replicating workshop at other locations

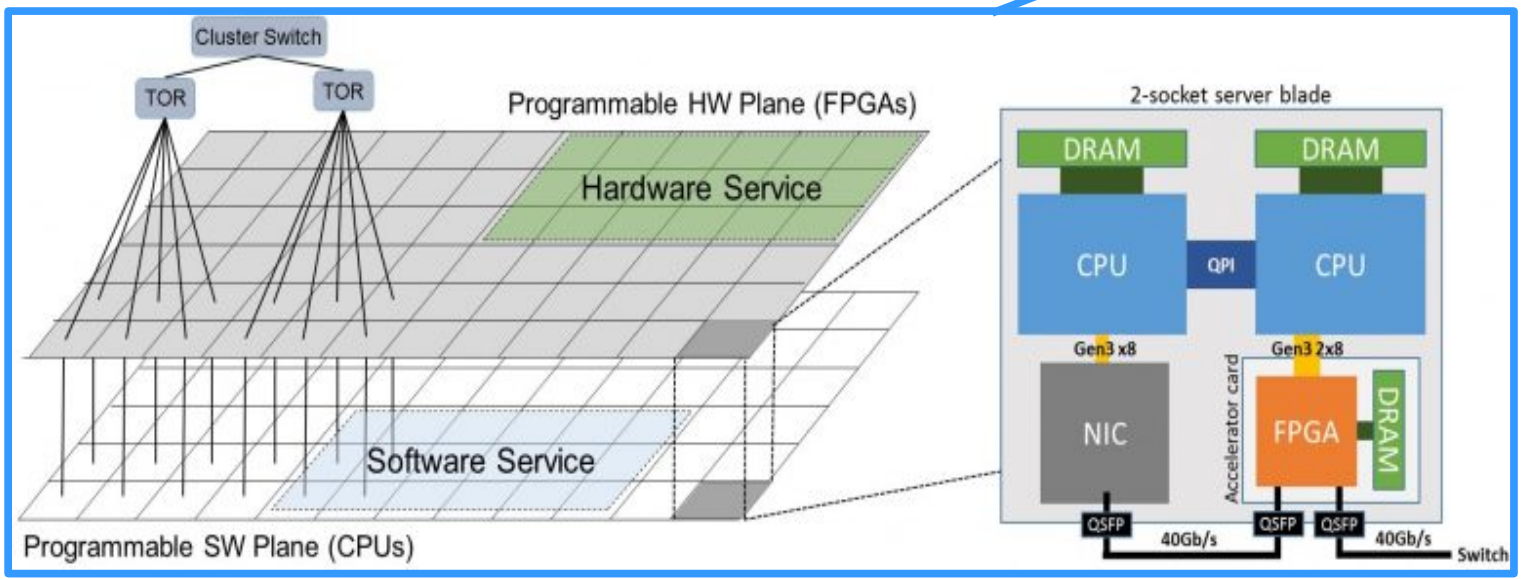
Workshop Program

- First part:
  - Understand the hls4ml package, its functionalities and design synthesis by running with one of the provided trained NN
  - Learn how to read out an estimate of FPGA resources and latency for a NN after synthesis
  - Learn how to optimize the design with quantization and parallelization
- Second part:
  - Learn how to export the HLS design to firmware with SDAccel
- Third part:
  - Learn how to do model compression and its effect on the FPGA resources/latency
- Fourth part:
  - Learn how to accelerate NN inference firmware on a real FPGA (provided on Amazon cloud) with SDAccel
  - Timing and resources studies after running on real FPGA

Microsoft Brainwave

- Microsoft Brainwave is a CPU-FPGA server farm
- ML offered “as a service”
  - Send a preprocessed image to Brainwave, get prediction back from ResNet50
- Extremely interesting option for longer latency (>10 ms) systems
- Stay tuned for another talk in the future





Outlook (1)

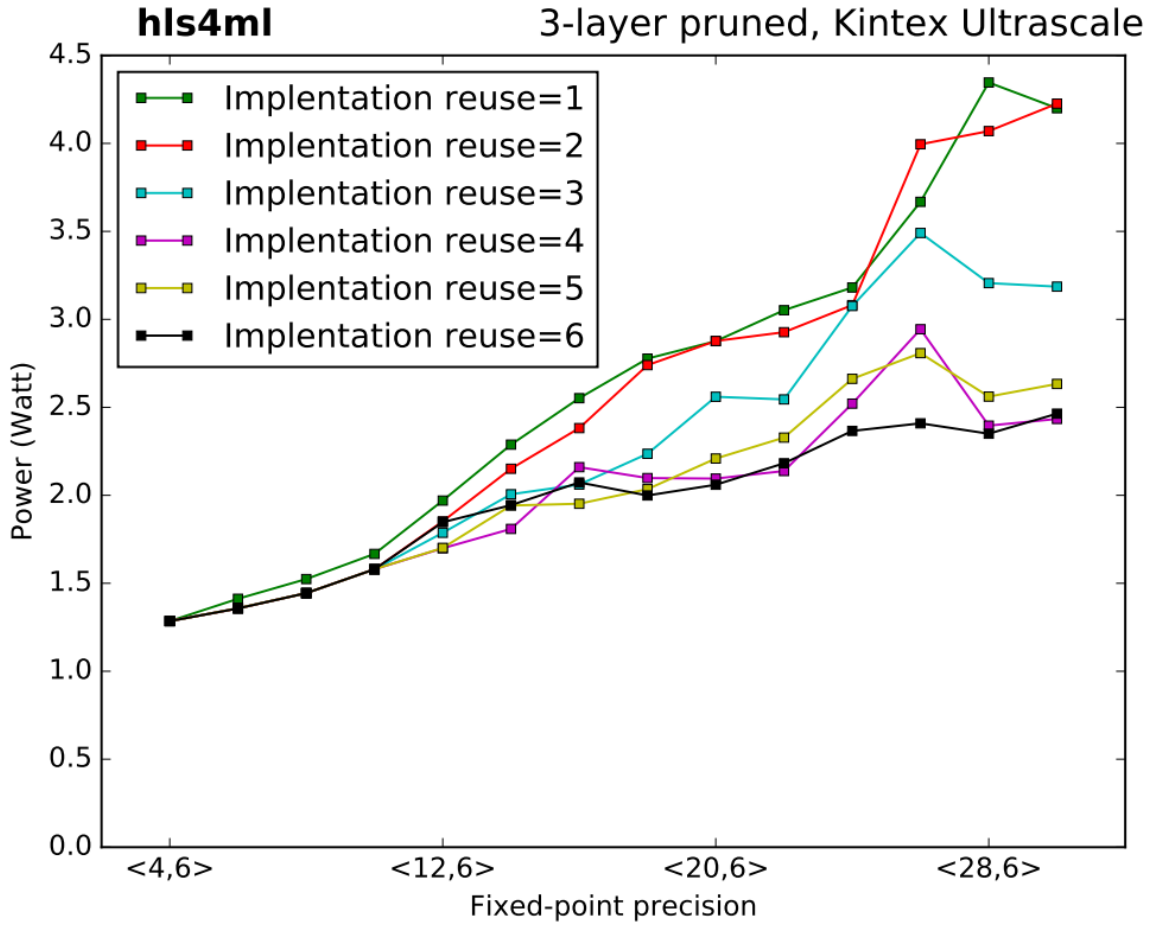
- Machine learning is becoming increasingly widely used for complex problems
  - Not only in physics but also industry
- Some of the most challenging problems in HEP exist in the trigger
  - Even with a machine learning solution, still need to be able to perform inference very fast → FPGAs
- **`hls4ml` provides the ability to implement very low latency machine learning solutions on FPGAs with a high degree of customization**
  - Can adjust configuration of implementation for desired resource usage, latency, and accuracy

Outlook (2)

- Have utilized hls4ml already to enable fast machine learning solutions in CMS trigger
  - Ex. muon p_T assignment, tau identification, others
- Improvements in fast inference need not be limited to traditionally FPGA-based systems
  - Ex. large potential improvement in processing time for HCAL
  - Could envision using accelerator cards during offline processing or HLT
- Usage not restricted only to CMS trigger, many other possibilities for use of fast ML
- Growing list of collaborators:
  - Jennifer Ngadiuba, Vladimir Loncar, Maurizio Pierini [CERN] Giuseppe Di Guglielmo [Columbia University] Javier Duarte, Burt Holzman, Sergo Jindariani, Ben Kreis, Mia Liu, Kevin Pedro, Ryan Rivera, Nhan Tran, Aristeidis Tsaris [Fermilab] Edward Kreinar [HawkEye 360] Sioni Summers [Imperial College London] Song Han, Phil Harris, Dylan Rankin [MIT] Zhenbin Wu [UIC] Scott Hauk, Shih-Chieh Hsu, Dustin Warren, Risha Rao [UW] Mark Neubauer, Markus Atkinson [UIUC]

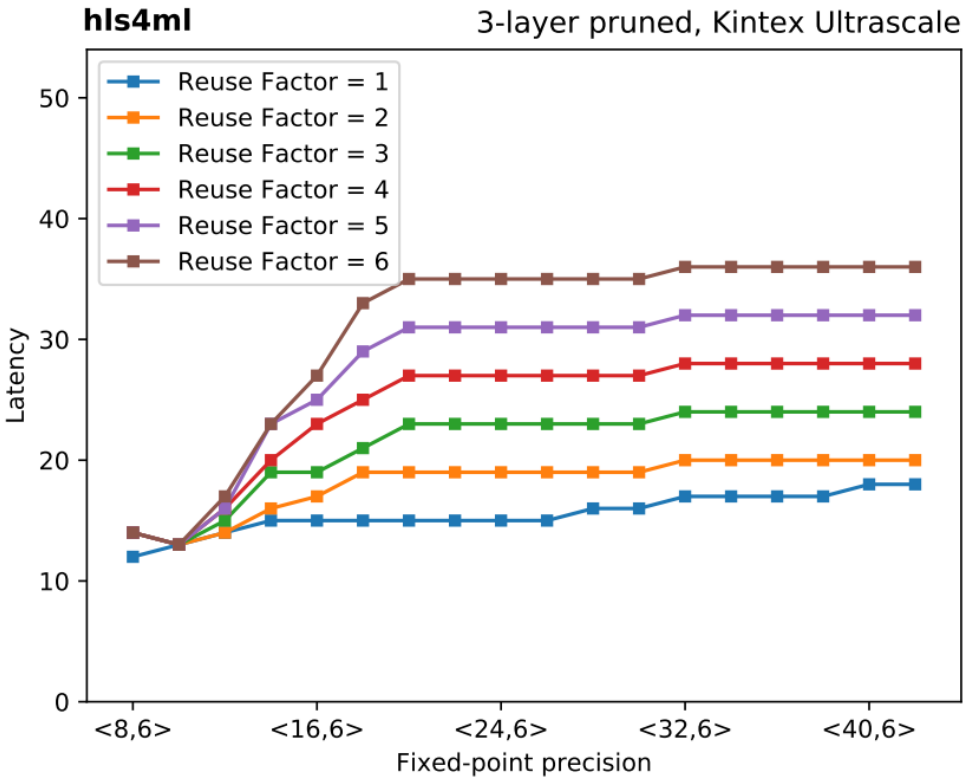
BACKUP

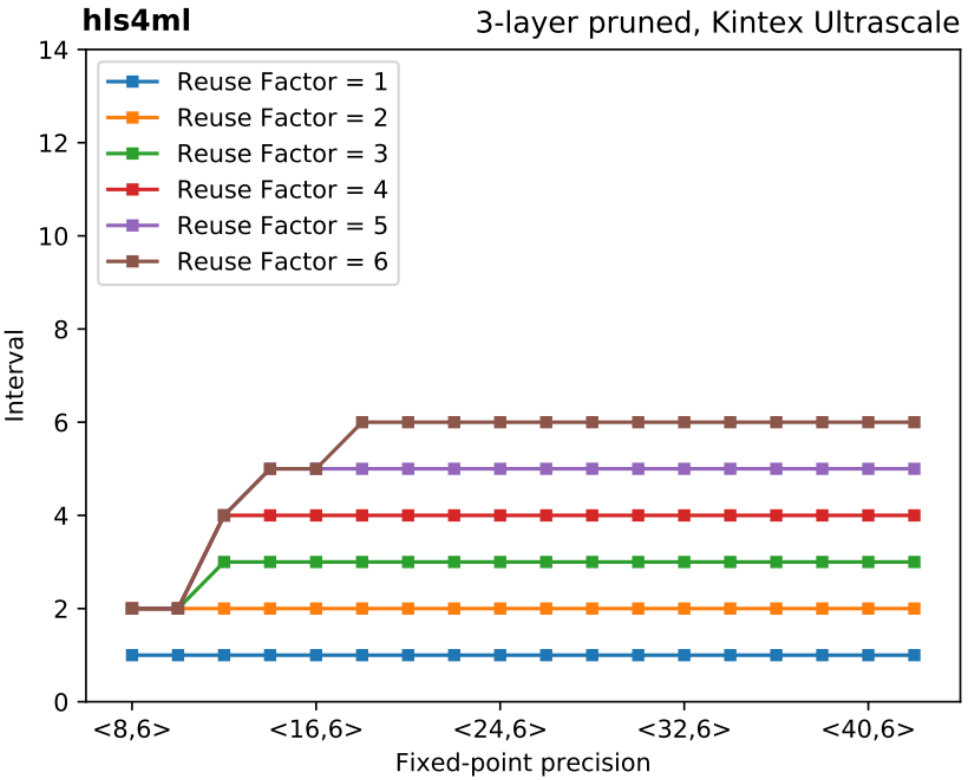
Power Usage



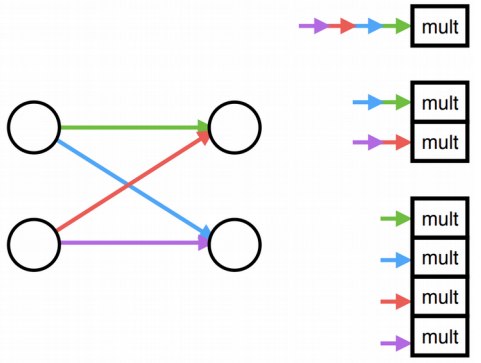
- Reuse also improves power consumption

Example Tuning: Reuse

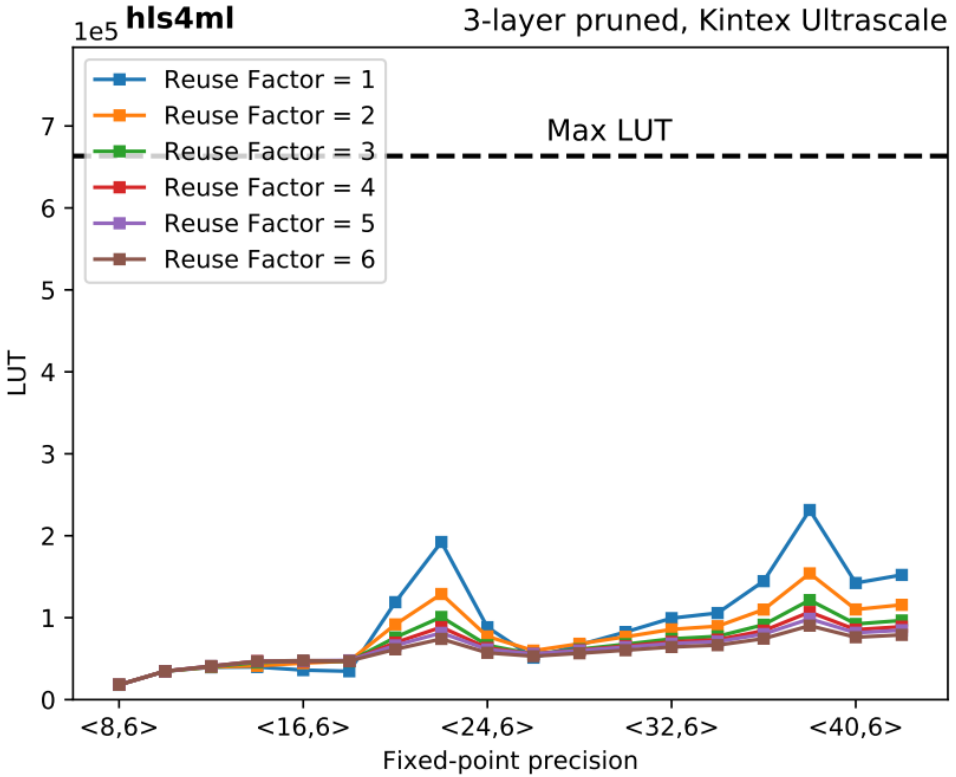


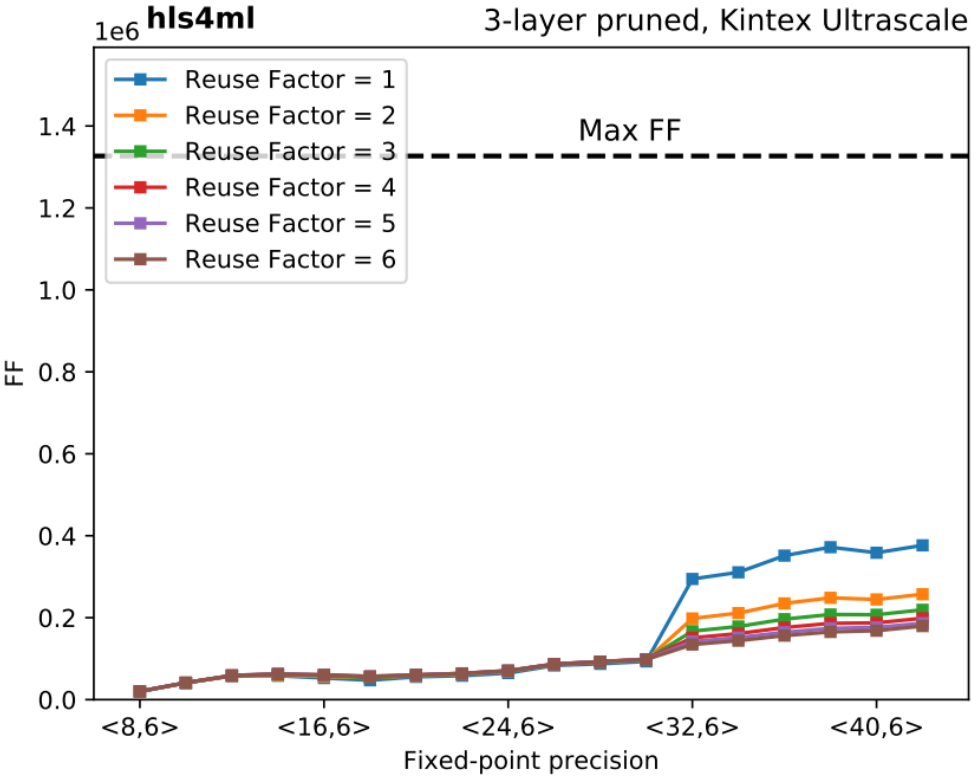


- Can tune reuse factor in hls4ml configuration

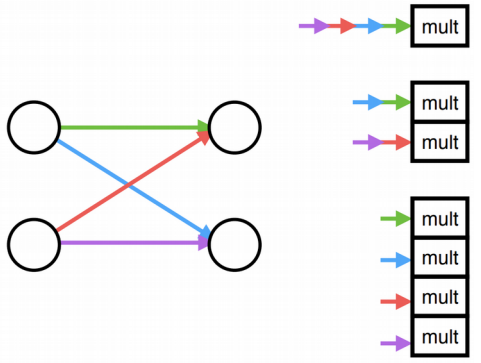


Example Tuning: Reuse

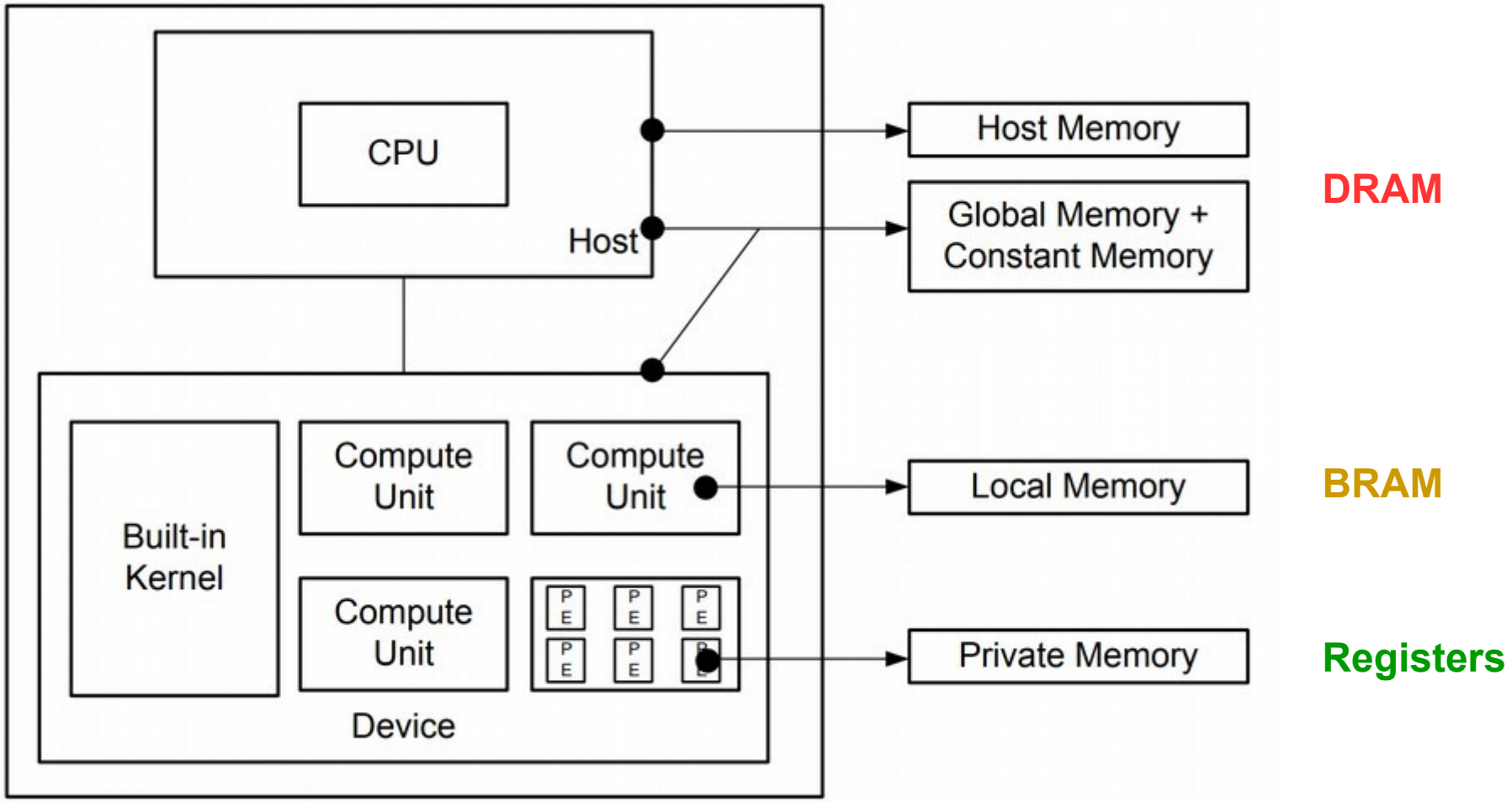




- Can tune reuse factor in hls4ml configuration



SDAccel Dataflow



- Multiple different memory management options

Acceleration with Xilinx xDNN

- Have also investigated Xilinx xDNN package for acceleration of large convolutional networks
  - Connection between CPU and FPGA with PCI-express as before
  - Major latency comes in xDNN setup (loading weights)
  - Can batch inputs: allows reuse of loaded weights, only costs additional few ms per image
- Some similarities with Microsoft Brainwave
- Major difference is that xDNN/AWS lacks an “as a service” offering