



Deep learning on FPGAs for L1 trigger and Data Acquisition

DESY Instrumentation Seminar, October 12, 2018

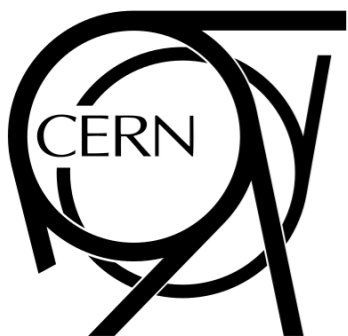
Javier Duarte, Sergo Jindariani, Ben Kreis, Ryan Rivera, Nhan Tran (Fermilab)

Jennifer Ngadiuba, Maurizio Pierini (CERN)

Edward Kreinar (Hawkeye 360)

Phil Harris, Song Han (MIT)

Zhenbin Wu (University of Illinois at Chicago)





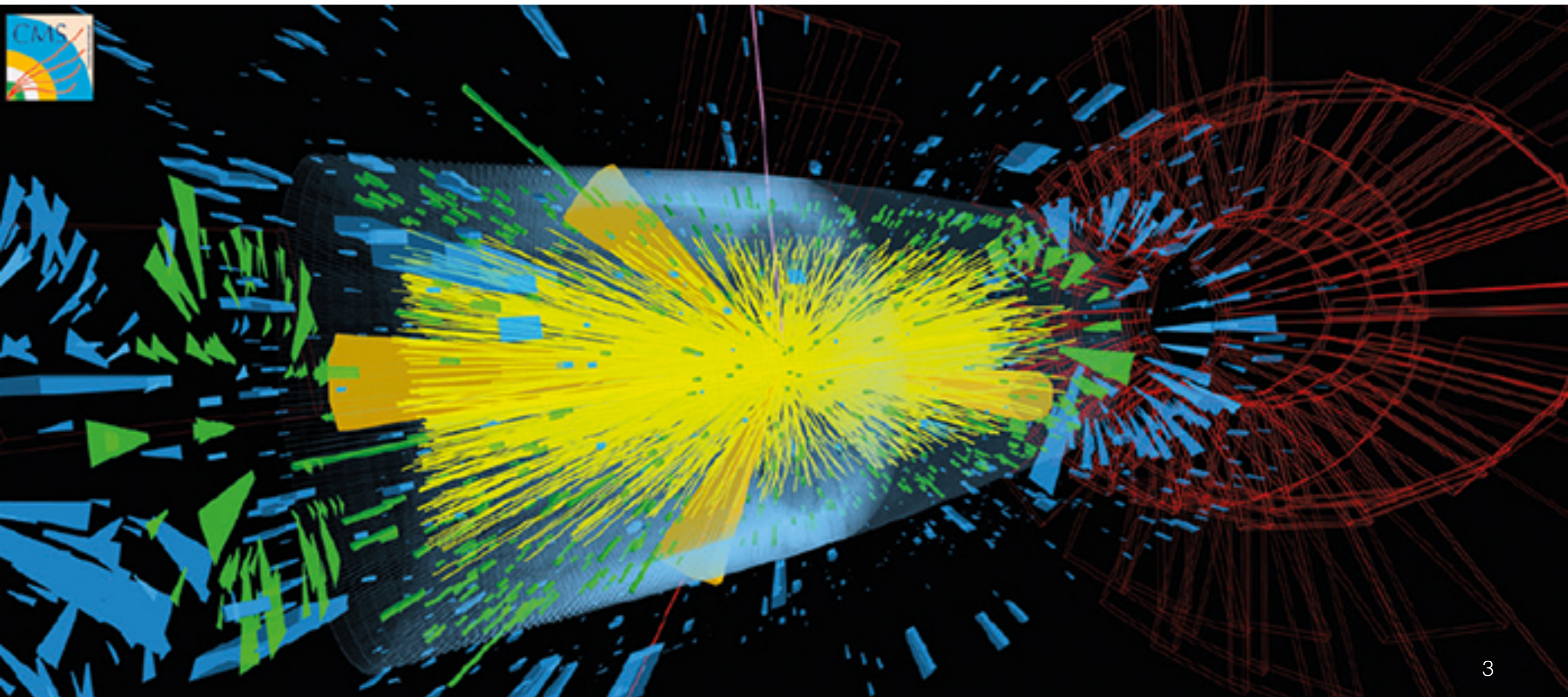
Motivation

The challenge: triggering at (HL-)LHC

The challenge: triggering at (HL-)LHC

Extreme bunch crossing frequency of 40 MHz \rightarrow extreme data rates $O(100 \text{ TB/s})$

“**Triggering**” = filter events to reduce data rates to manageable levels



The challenge: triggering at (HL-)LHC

Extreme bunch crossing frequency of 40 MHz → extreme data rates O(100 TB/s)

“**Triggering**” = filter events to reduce data rates to manageable levels

Squeeze the beams to increase data rates
→ multiple pp collisions per bunch crossing (pileup)

2016: $\langle \text{PU} \rangle \sim 20\text{-}50$

2017 + Run 3: $\langle \text{PU} \rangle \sim 50\text{-}80$

HL-LHC: 140-200

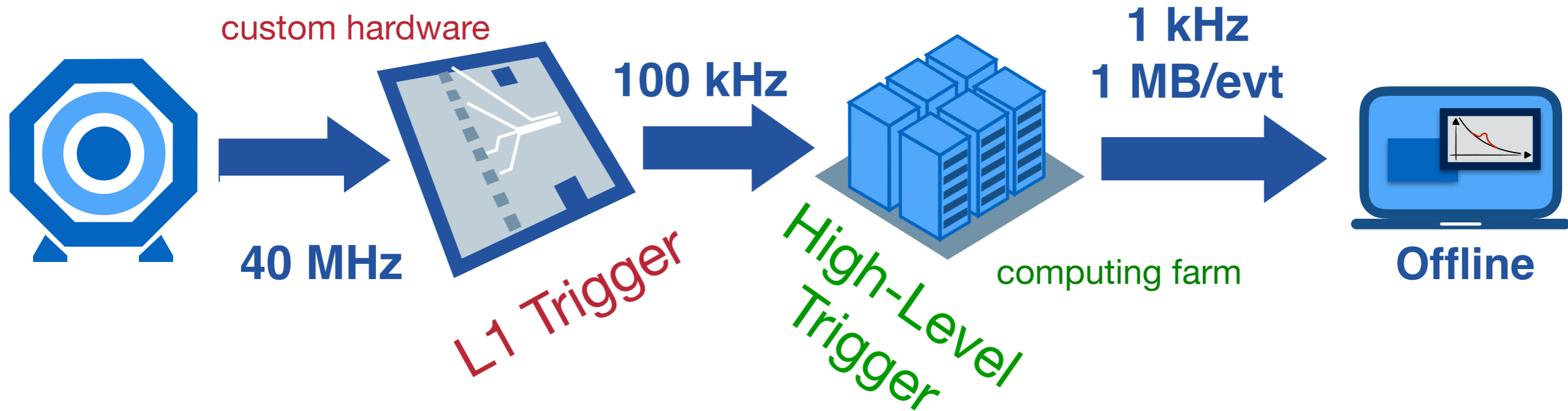
CHALLENGE: maintain physics in increasingly complex collision environment

→ untriggered events lost forever!

Sophisticated techniques needed to preserve the physics!

A typical trigger system

Triggering typically performed in multiple stages @ ATLAS and CMS



Absorbs 100s TB/s

Trigger decision to be made in $O(\mu\text{s})$

Latencies require all-FPGA design

Computing farm for detailed analysis of the full event

Latency $O(100\text{ ms})$

For HL-LHC upgrade: latency and output rates will increase by ~ 3 (ex: for CMS $3.8 \rightarrow 12.5\ \mu\text{s}$ @ L1)

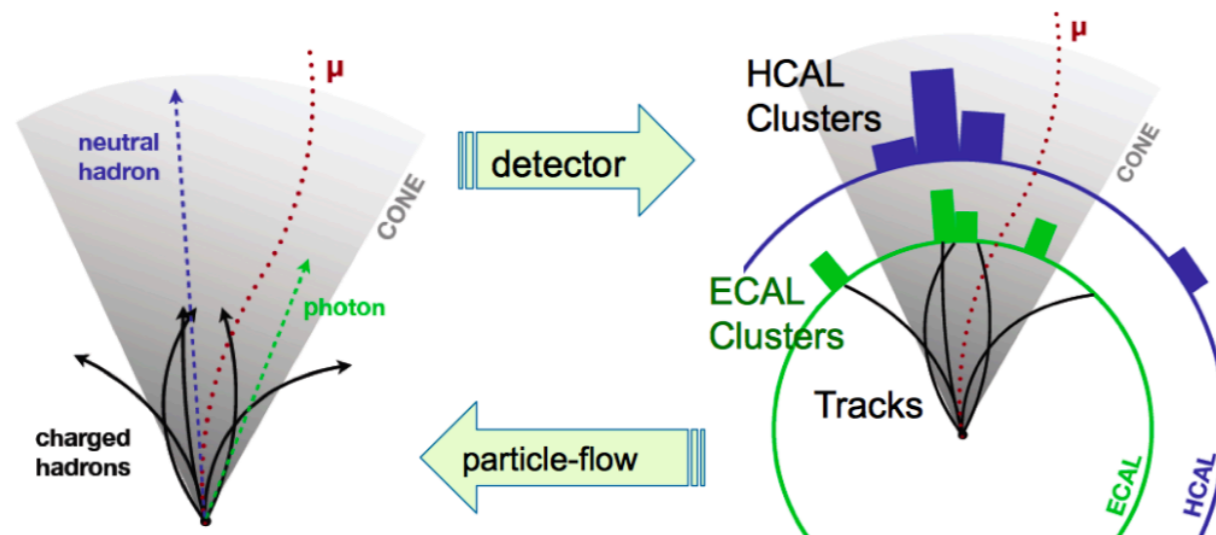
New trigger algorithms

The detector and its trigger system



Output: trigger primitives
(calo energy clusters,
muons, tracks)

Particle-flow algorithm @ L1



Output:
particle candidates

Trigger decision

[CMS-TDR-017, JINST 12 \(2017\) P10003](#)

New trigger algorithms

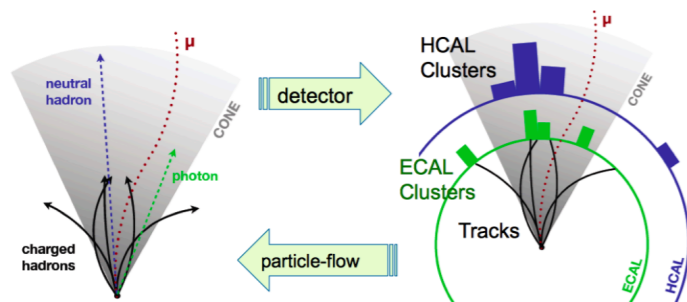
The detector and its trigger system



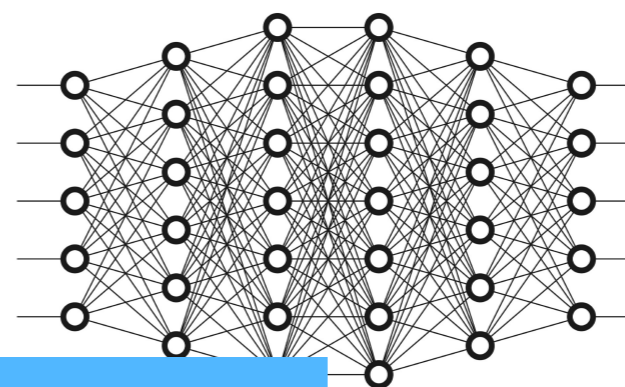
Output: trigger primitives
(calo energy clusters, muons, tracks)

Particle-flow algorithm @ L1

Output:
particle candidates



Machine learning



THIS TALK!

Trigger decision

[CMS-TDR-017, JINST 12 \(2017\) P10003](#)

New trigger algorithms

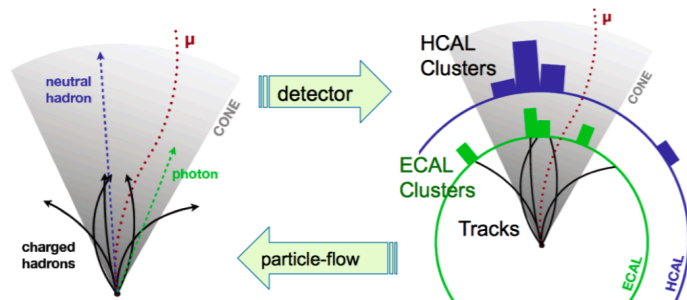
The detector and its trigger system



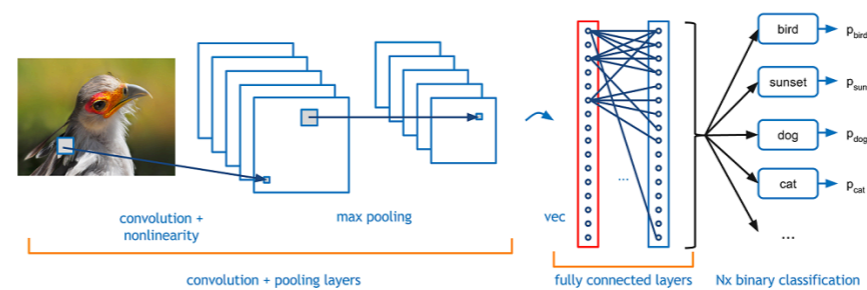
Output: trigger primitives
(calo energy clusters,
muons, tracks)

Particle-flow algorithm @ L1

Output:
particle candidates



Machine learning

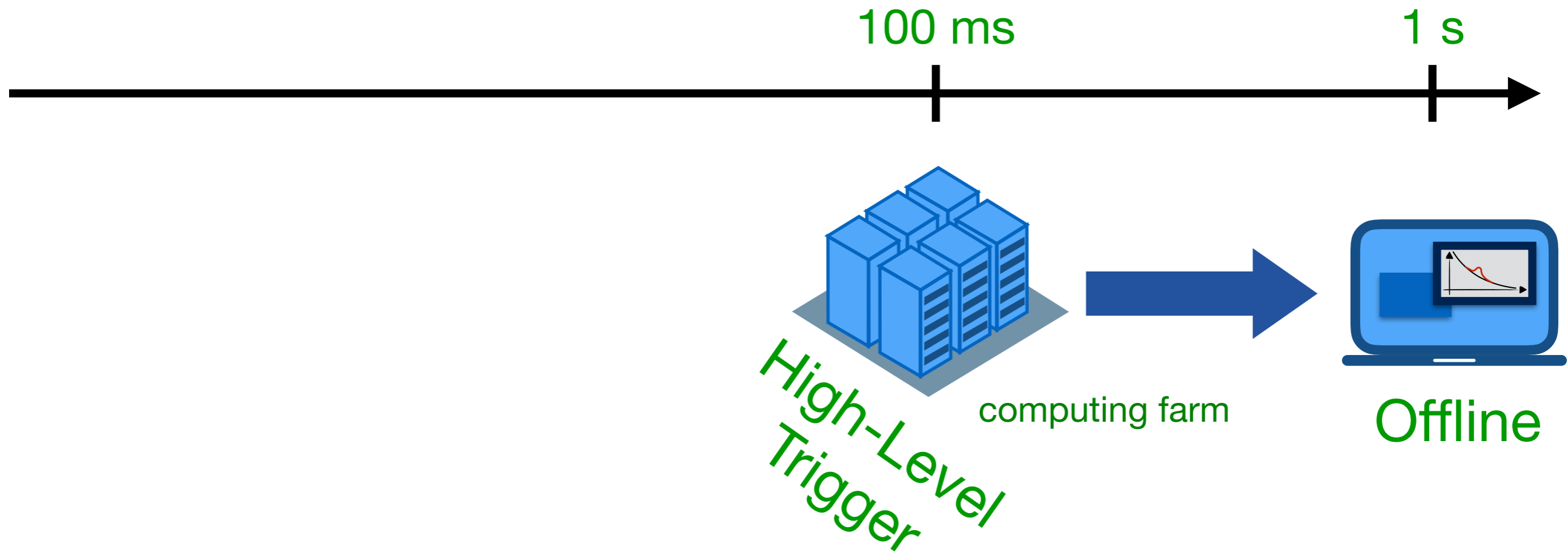


THIS TALK!

Trigger decision

[CMS-TDR-017, JINST 12 \(2017\) P10003](#)

The latency landscape @ LHC



ML methods typically employed in offline analysis or longer latency trigger tasks

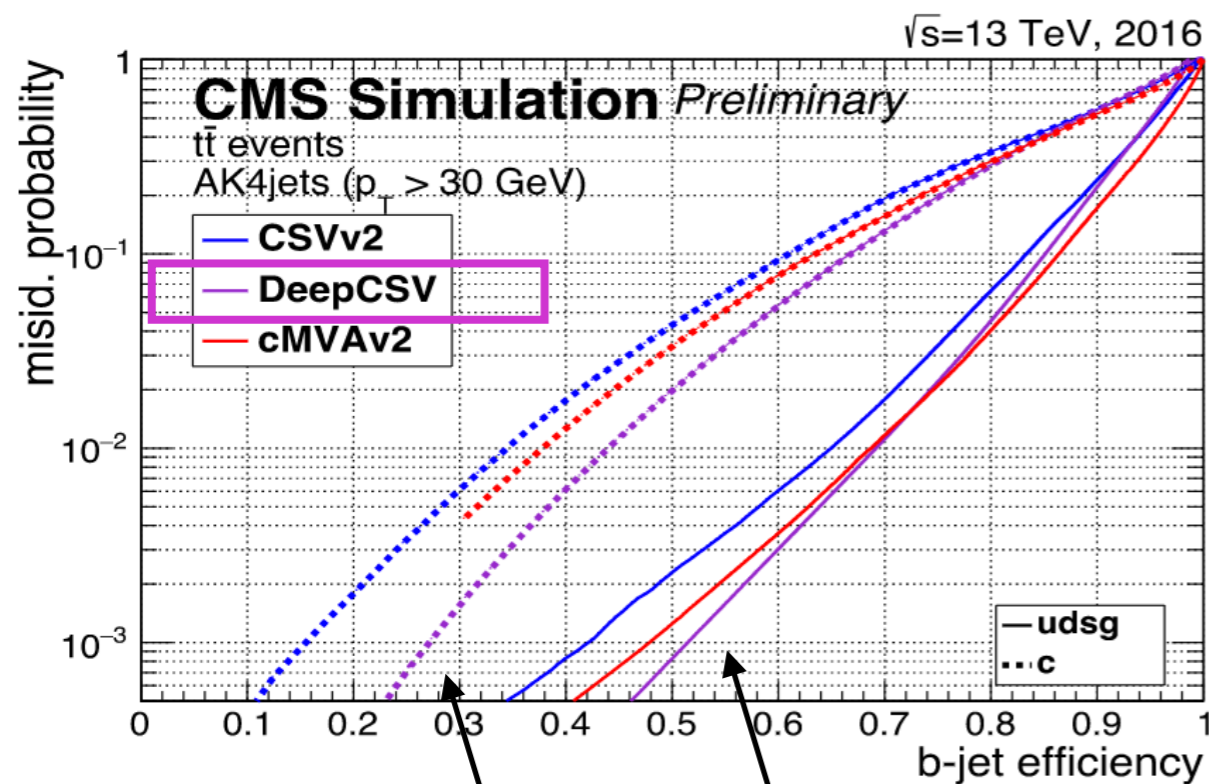
Many successes in HEP: identification of b-quark jets, Higgs candidates, particle energy regression, analysis selections,

The latency landscape @ LHC

100 ms

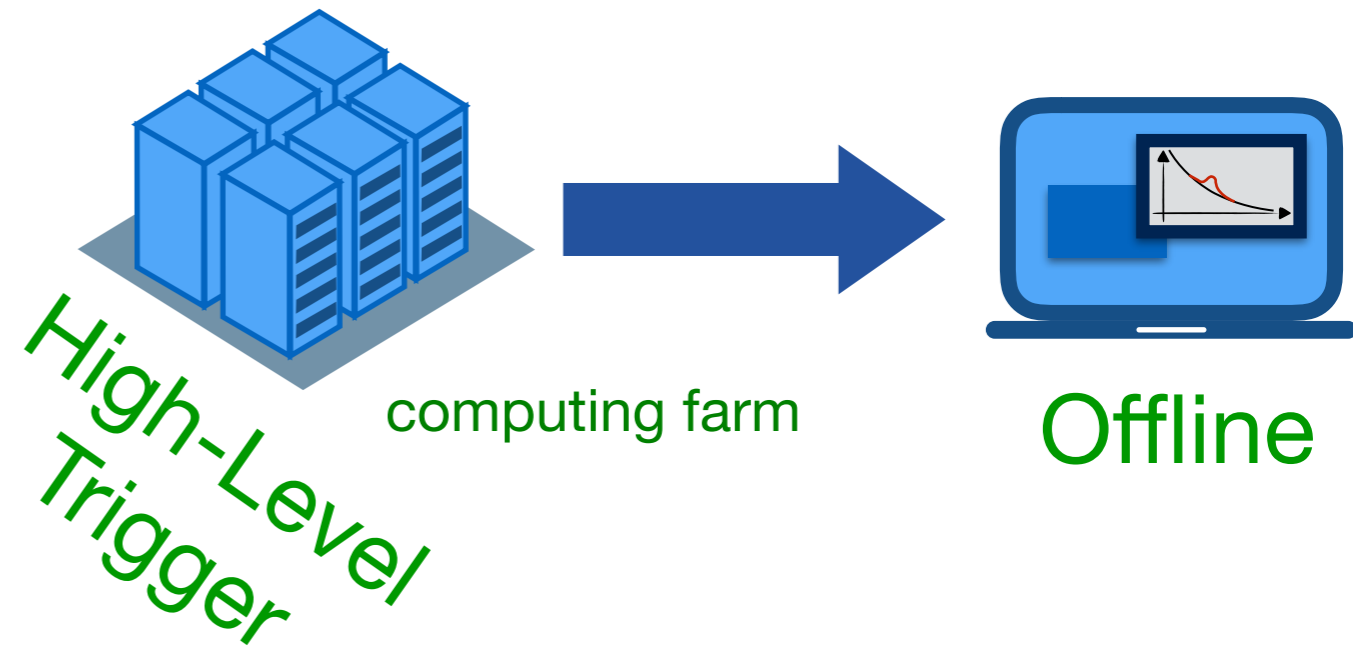
1 s

ex, identification of b-quark jets



Deep neural network based on high-level features

both offline and @ HLT



ML methods typically employed in offline analysis or longer latency trigger tasks

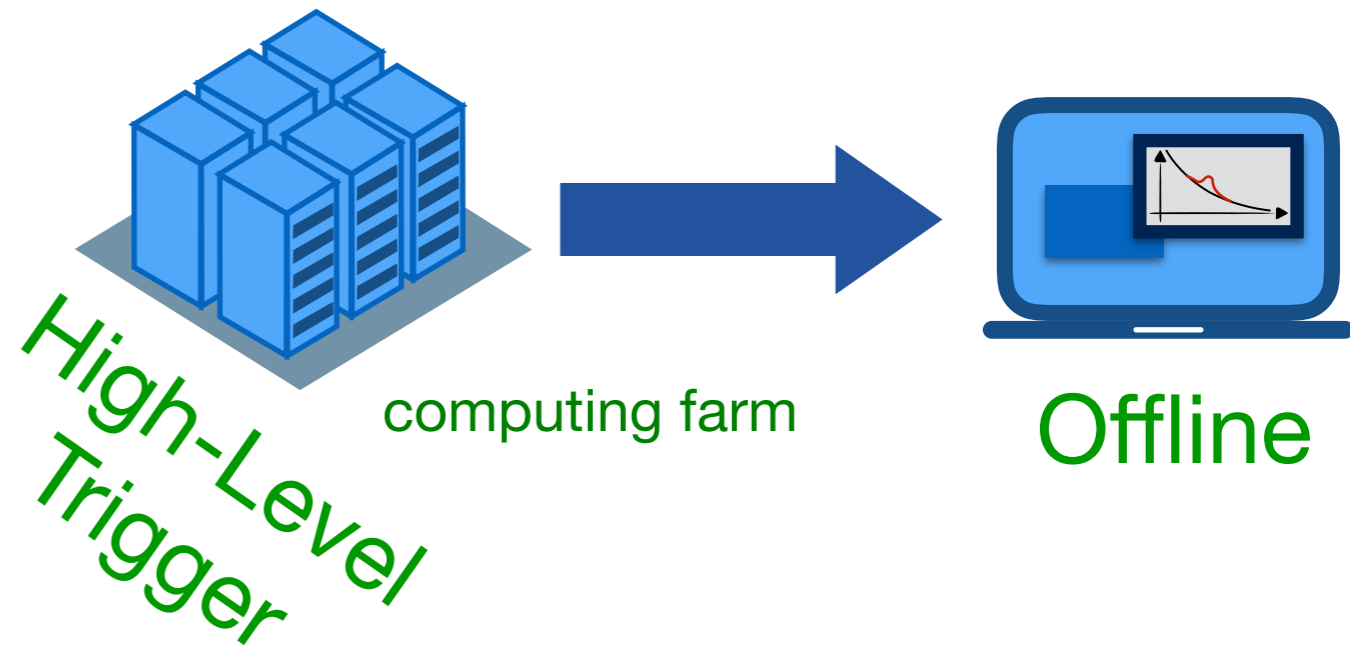
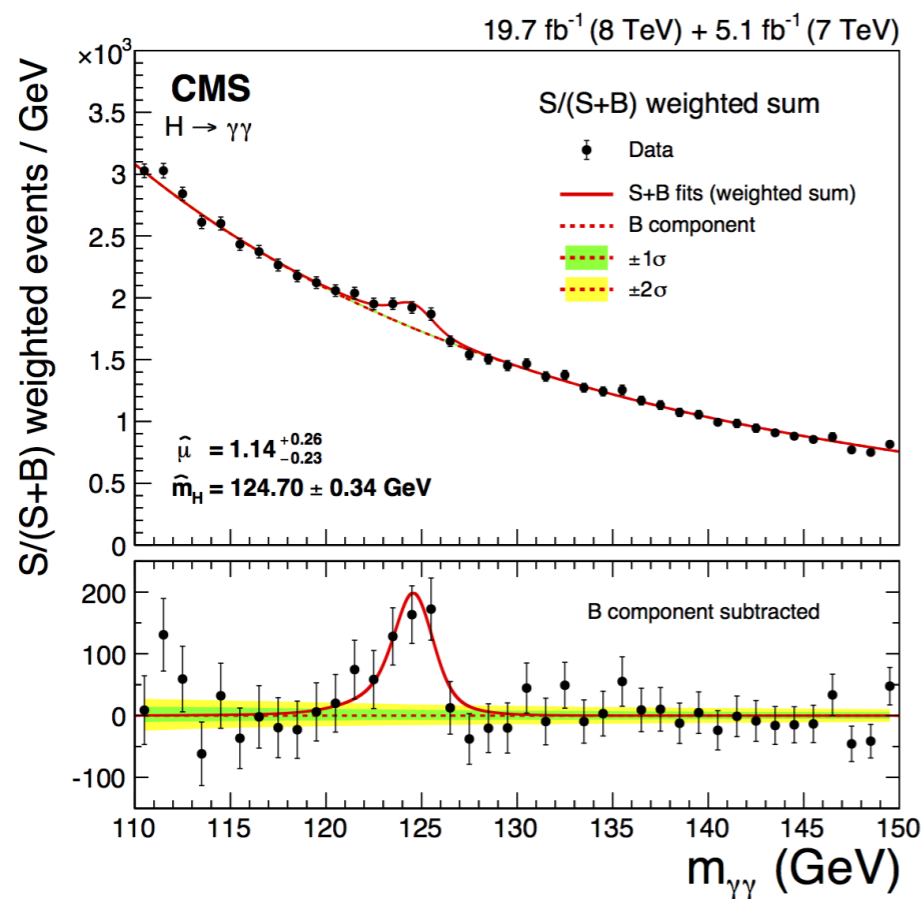
Many successes in HEP: identification of b-quark jets, Higgs candidates, particle energy regression, analysis selections,

The latency landscape @ LHC

100 ms

1 s

ex, Higgs discovery



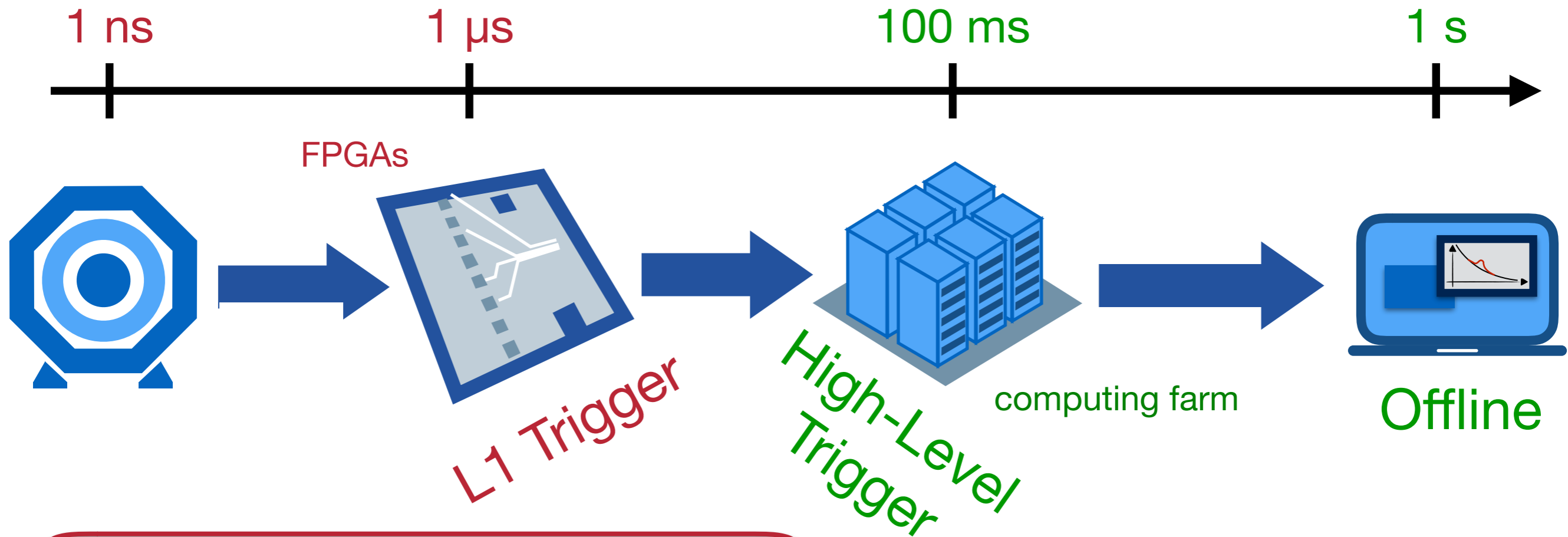
ML methods typically employed in offline analysis or longer latency trigger tasks

Many successes in HEP: identification of b-quark jets, Higgs candidates, particle energy regression, analysis selections,

ML algorithms used offline for

- * improving Higgs mass resolution with particle energy regression
- * enhancing signal/background discrimination

The latency landscape @ LHC



Exploration of ML algorithms in low-latency, real-time processing has just begun!

What can we do in $< \mu$ s on one FPGA?

ML methods typically employed in offline analysis or longer latency trigger tasks

Many successes in HEP: identification of b-quark jets, Higgs candidates, particle energy regression, analysis selections,

Muon reconstruction @ L1

First implementation of a ML algo for CMS L1 trigger on FPGAs [*]

A BDT is used to improve the momentum of muons in the forward region of the detector

based on curvature angles in the magnetic fields ($\Delta\phi, \Delta\theta$) and few other variables

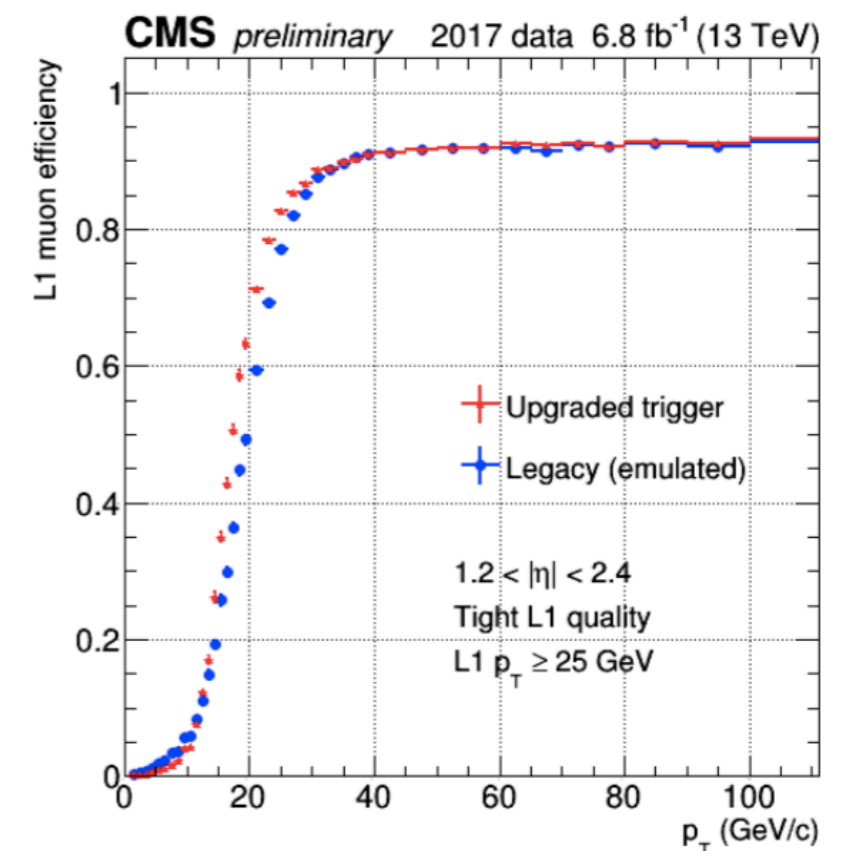
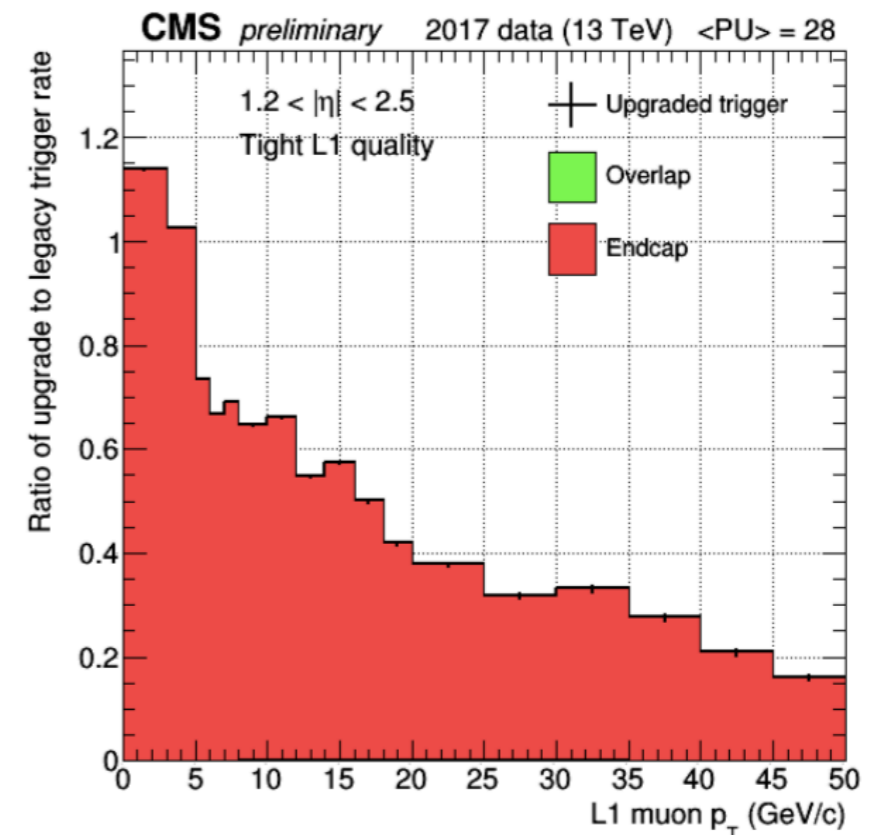
Prediction of BDT for every possible input stored into pre-computed 1.2 GB Look-Up Table (LUT) on FPGA

Achieved reduction of background rates by factor 3 w/o efficiency losses

Usage of LUTs does not scale nicely with ML algo complexity → quickly use all resources

Can we improve this approach?

[*] http://cds.cern.ch/record/2290188/files/CR2017_357.pdf?version=1



The rise of specialized hardware for ML



GPUs excel at parallel processing

Good for complex NN training of huge amount of data!

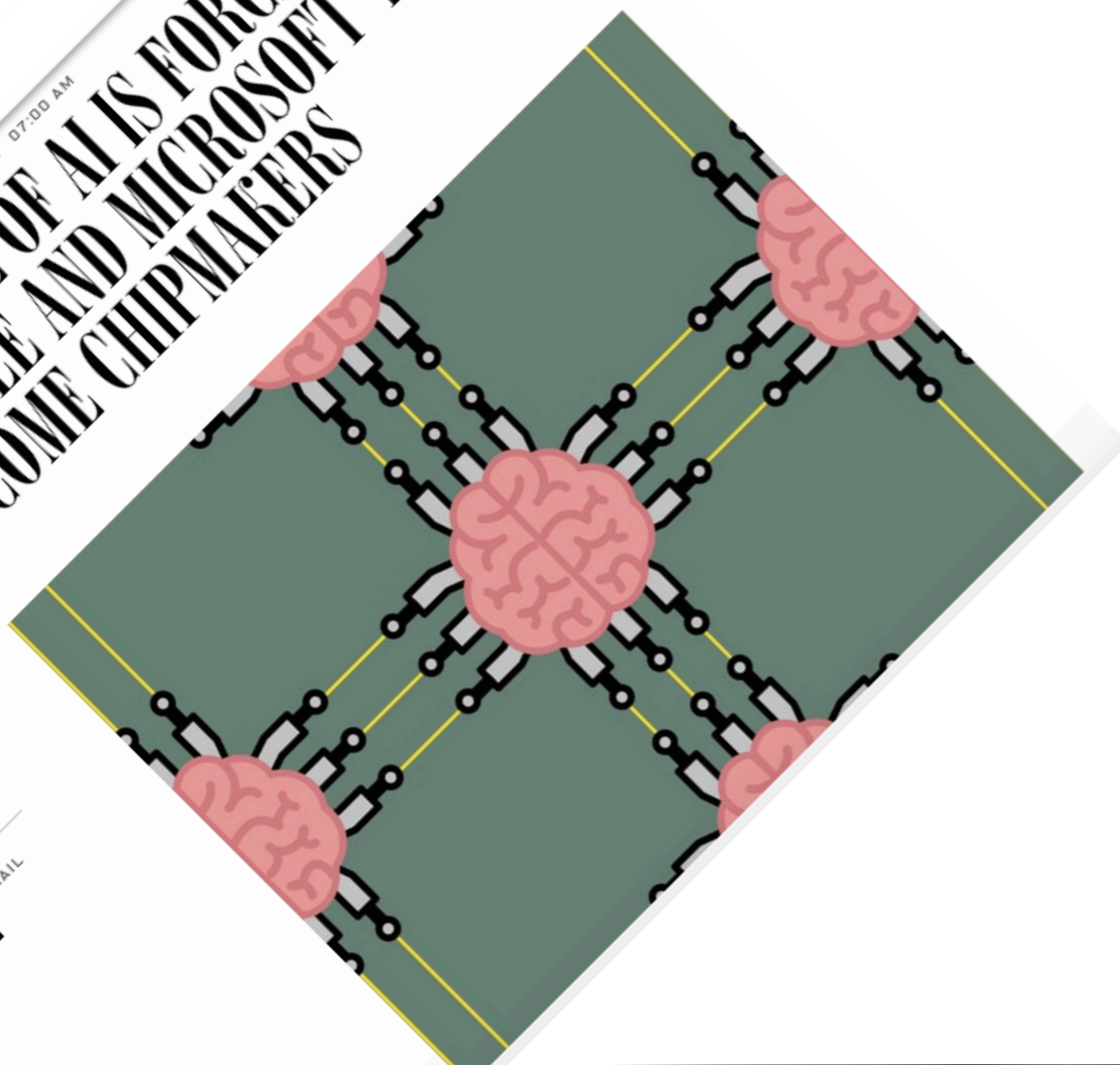
Notoriously power-hungry

Sub-optimal for fast and simple NN inference

Optimize resources utilization for less intensive tasks

The rise of specialized hardware for ML

SIMONITE BUSINESS 07.25.17 07:00 AM
**THE RISE OF AI IS FORCING
GOOGLE AND MICROSOFT TO
BECOME CHIPMAKERS**



GPUs excel at parallel processing
*Good for complex NN training of huge
amount of data!*
Notoriously power-hungry

Sub-optimal for fast and simple NN
inference

*Optimize resources utilization for less
intensive tasks*

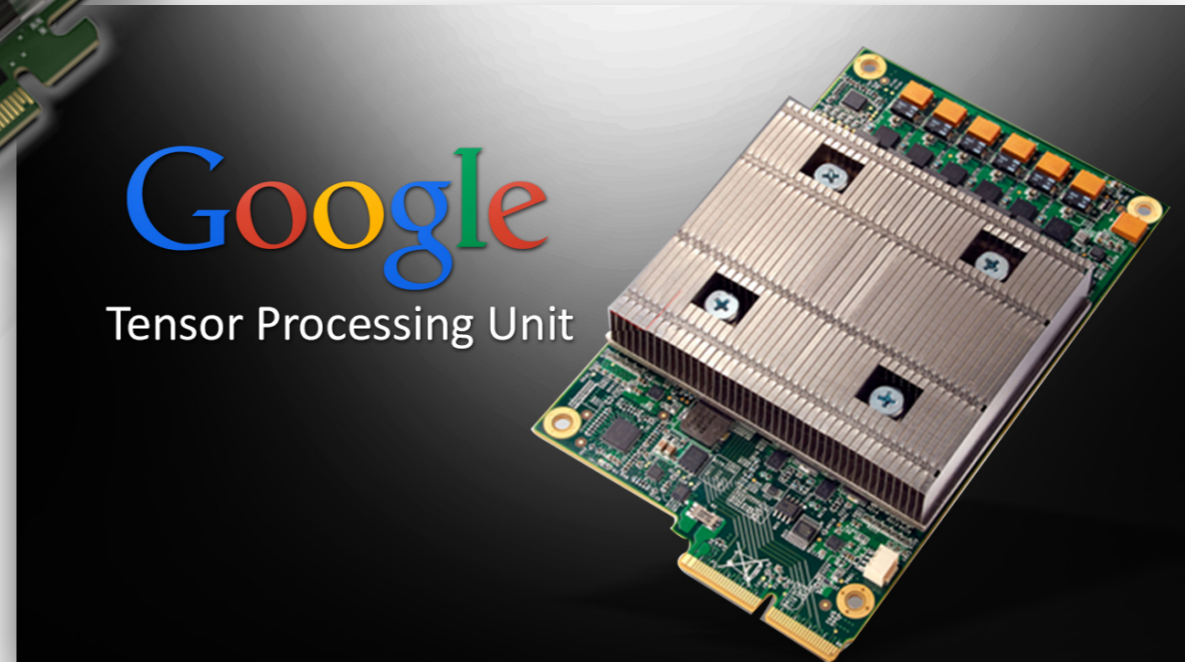
New developments in
FPGAs and ASICs making
#RealTimeAI possible!

The rise of specialized hardware for ML

SIMONITE BUSINESS 07.25.17 07:00 AM
**THE RISE OF AI IS FORCING
GOOGLE AND MICROSOFT TO
BECOME CHIPMAKERS**

Custom AI hardware for
Google on the cloud

Intel Arria 10 already at
cloud scale for Microsoft
Bing, Azure, etc..



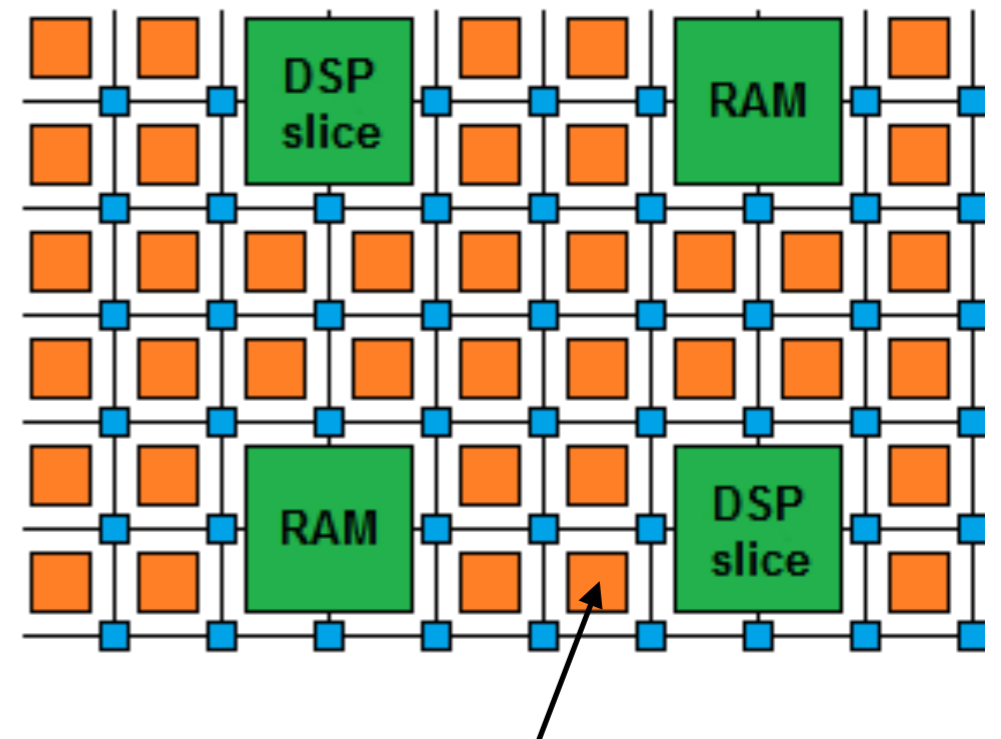
New developments in
FPGAs and ASICs making
#RealTimeAI possible!

What are FPGAs?

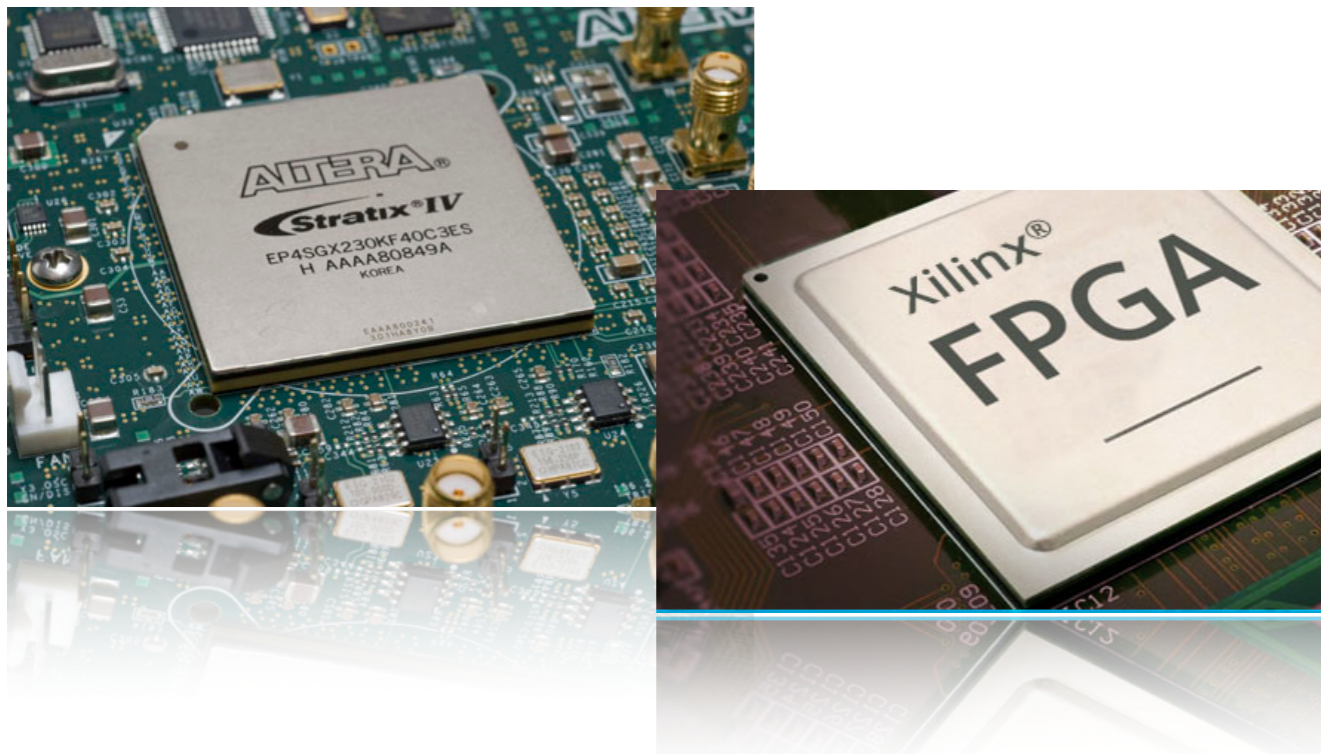
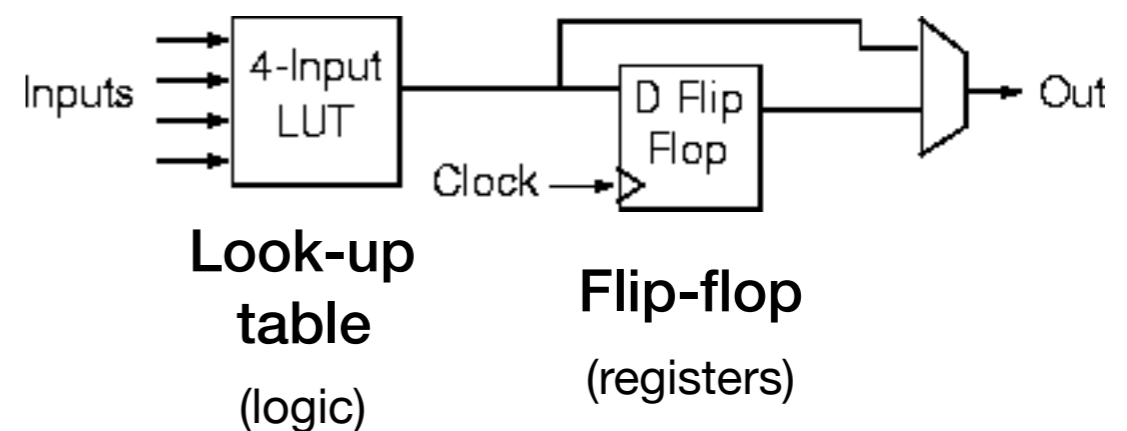
Field Programmable Gate Arrays are reprogrammable integrated circuits

Contain array of **logic cells** used to configure low level operations (bit masking, shifting, addition)

FPGA diagram



Logic cell

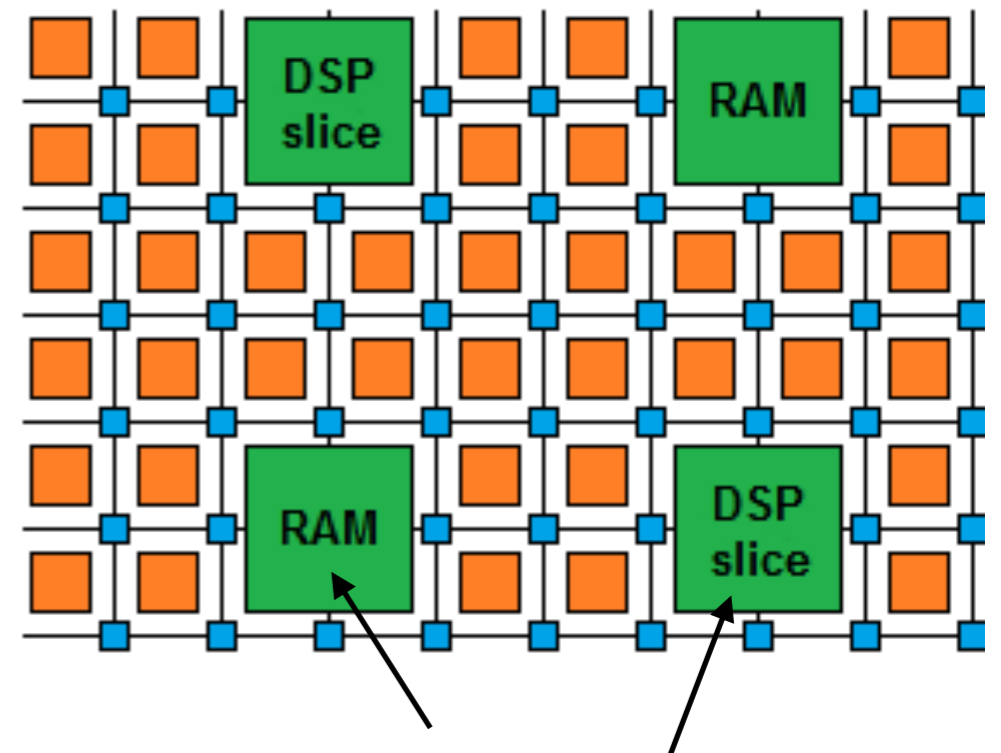


What are FPGAs?

Field Programmable Gate Arrays are reprogrammable integrated circuits

Contain array of **logic cells** used to configure low level operations (bit masking, shifting, addition)

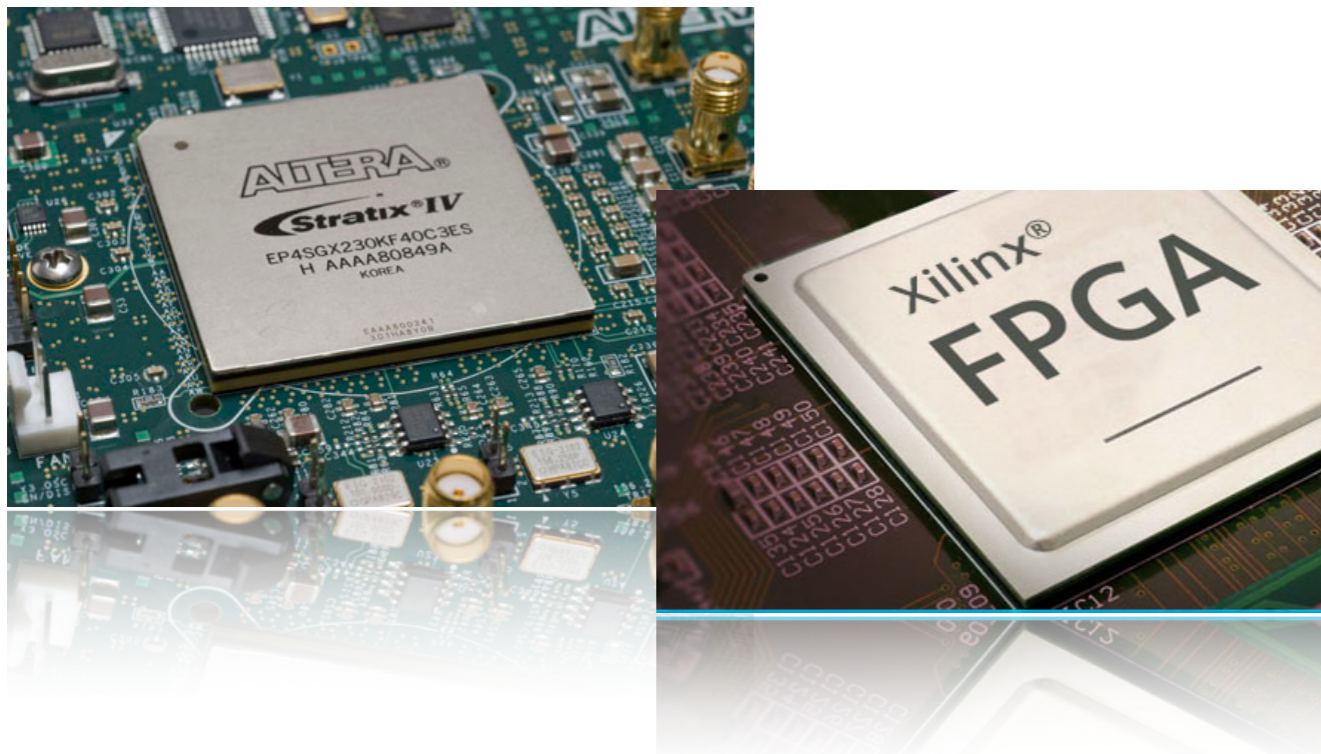
FPGA diagram



Also contain embedded components:

Digital Signal Processors (DSPs):
logic units used for multiplications

Random-access memories (RAMs):
embedded memory elements



What are FPGAs?

Field Programmable Gate Arrays are reprogrammable integrated circuits

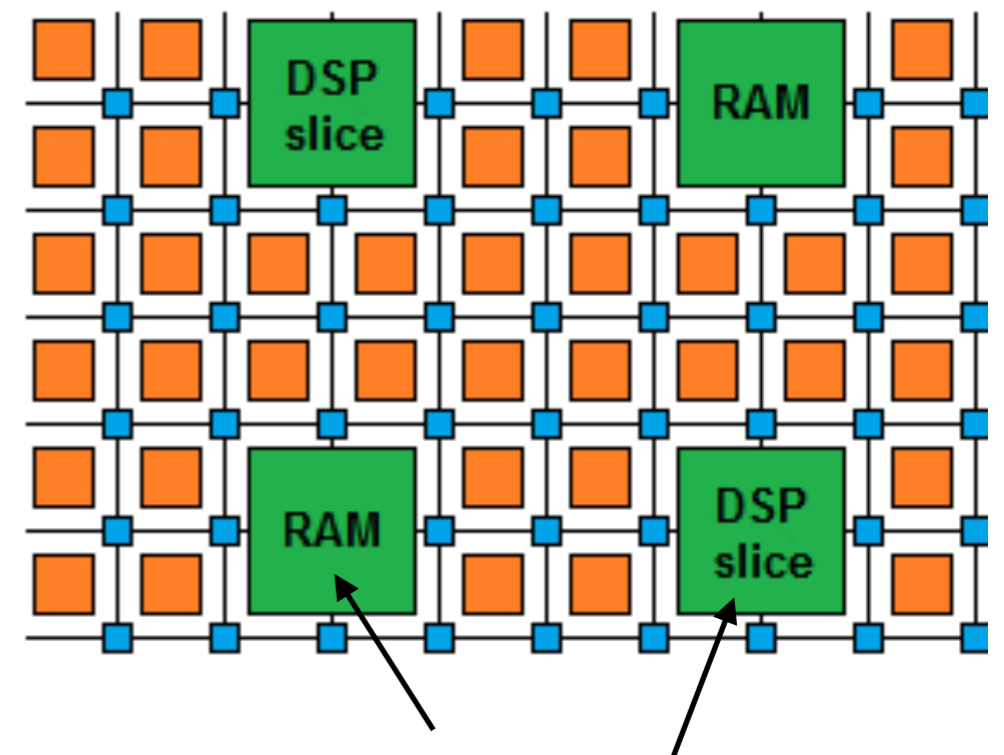
Contain array of **logic cells** embedded with **DSPs**, **BRAMs**, etc.

High speed input/output to handle the large bandwidth

Support highly parallel algorithm implementations

Low power (relative to CPU/GPU)

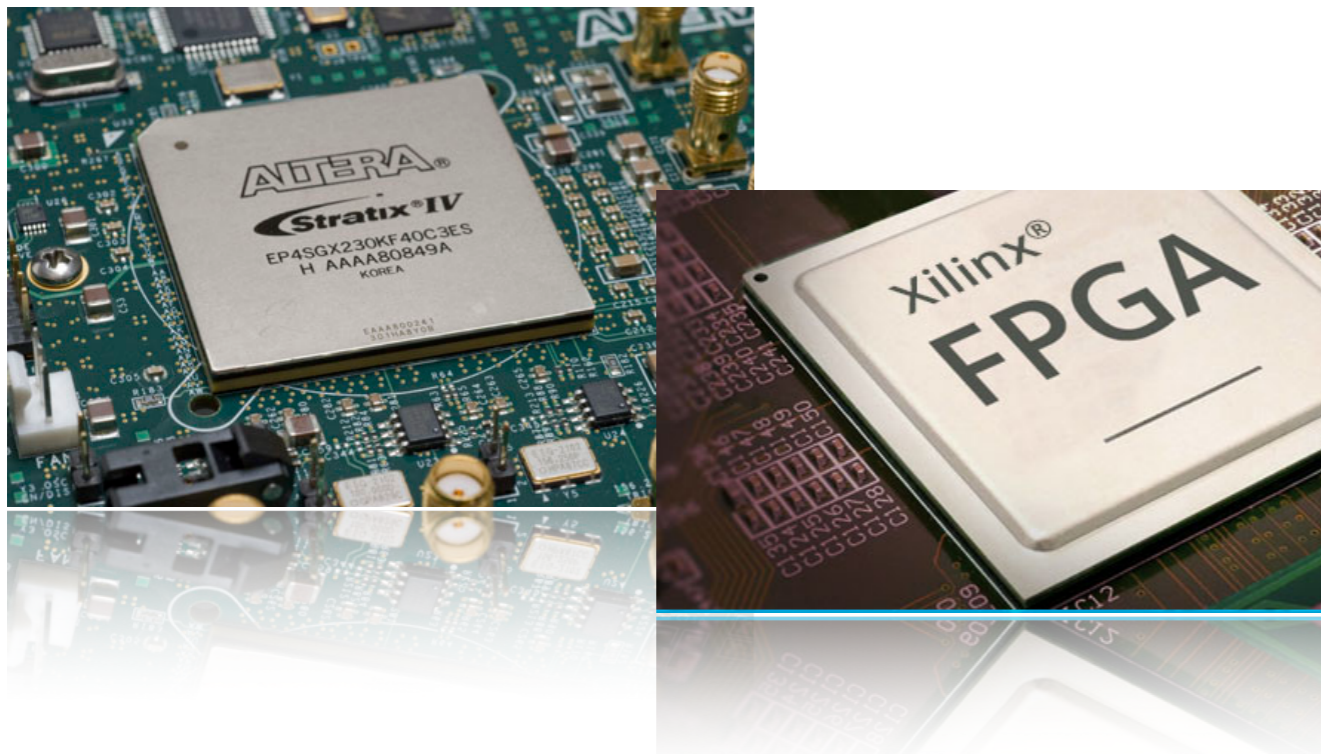
FPGA diagram



Digital Signal Processors (DSPs):
logic units used for multiplications

Random-access memories (RAMs):
embedded memory elements

Flip-flops (FF) and look up tables (LUTs) for additions



How are FPGAs programmed?

Hardware Description Languages

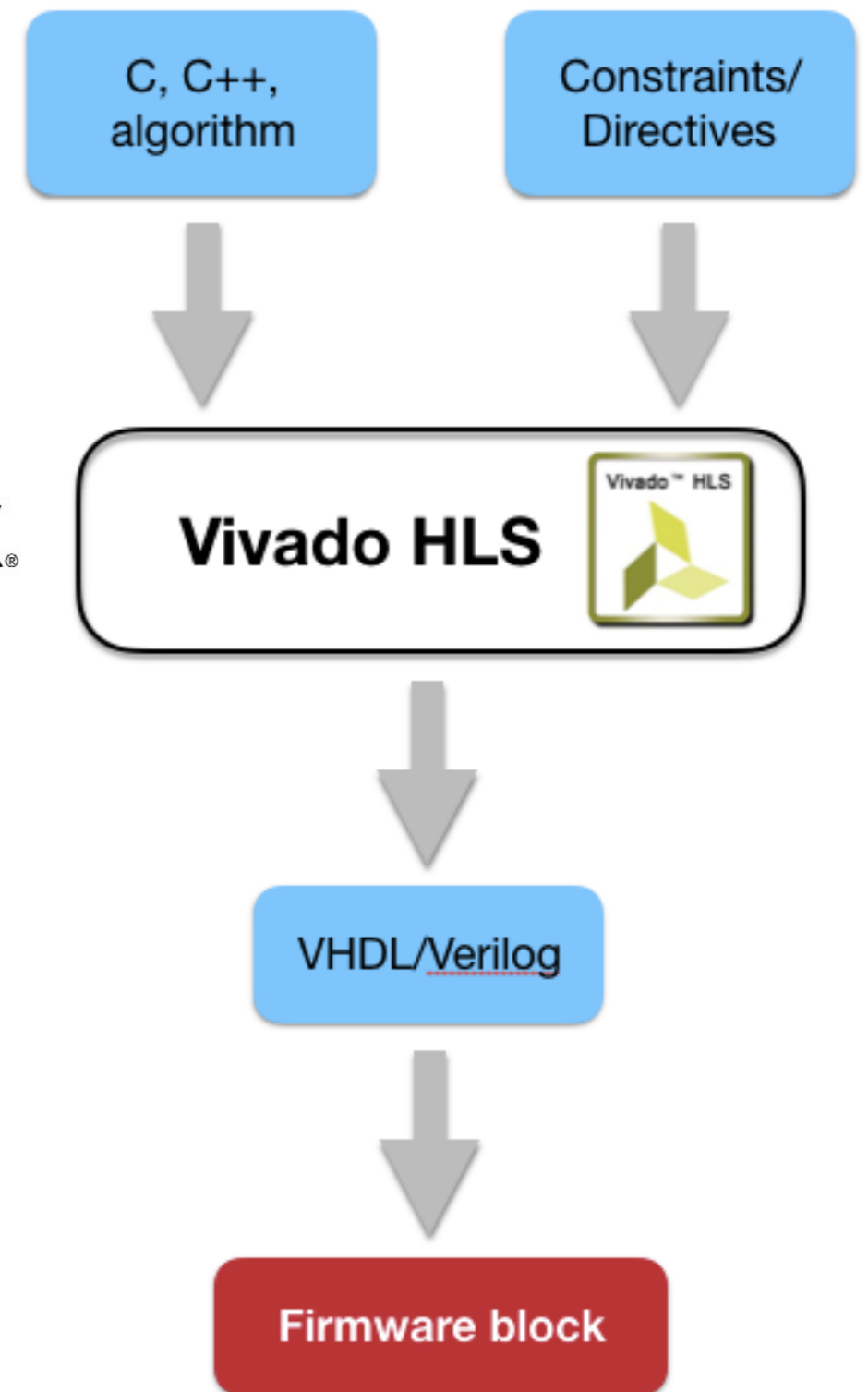
HDLs are programming languages which describe electronic circuits

High Level Synthesis

generate HDL from more common C/C++ code
pre-processor directives and constraints used to optimize the timing

drastic decrease in firmware development time!

We use here **Xilinx Vivado HLS** [*], but plan also Intel/Altera Quartus HLS



[*] https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug902-vivado-high-level-synthesis.pdf

Machine learning & FPGAs

FPGAs used broadly across particle physics experiments for DAQ and trigger development

becoming more accessible thanks to use of HLS

the hardware structures maps nicely onto ML architectures

early adoption of ML algorithms for L1 trigger uses BDT

Extensive literature on deep learning in FPGAs

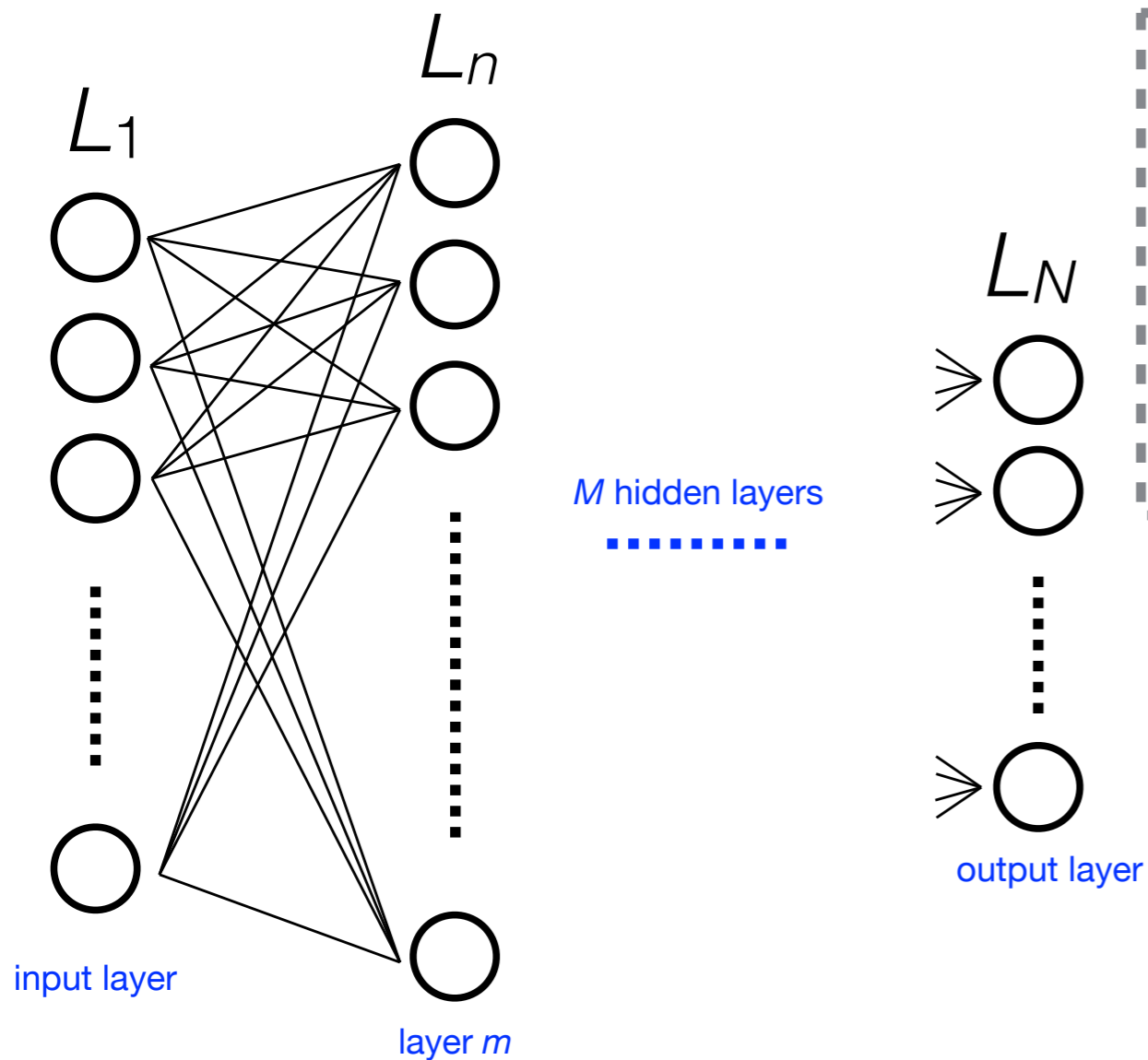
mainly targeting acceleration of large networks, relaxed latency constraints

support of ML architectures in Keras/TensorFlow, Caffe, Torch

This is the first dedicated study on inference of deep neural networks in FPGAs for low-latency application

[arxiv.1804.06913](https://arxiv.org/abs/1804.06913)

Neural network inference

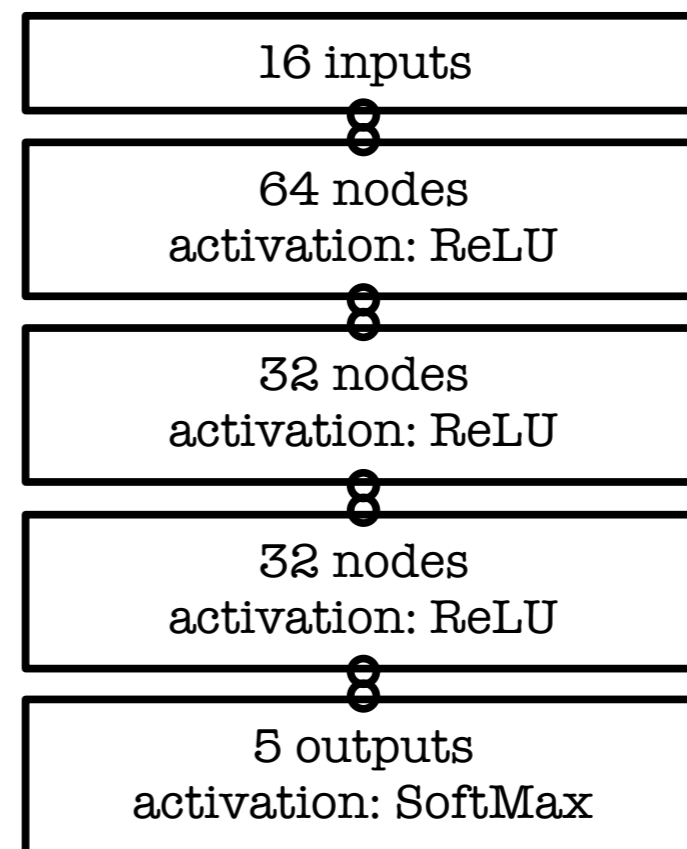


$$\mathbf{x}_n = g_n(\mathbf{W}_{n,n-1}\mathbf{x}_{n-1} + \mathbf{b}_n)$$

↗ activation function
precomputed and stored in BRAMs

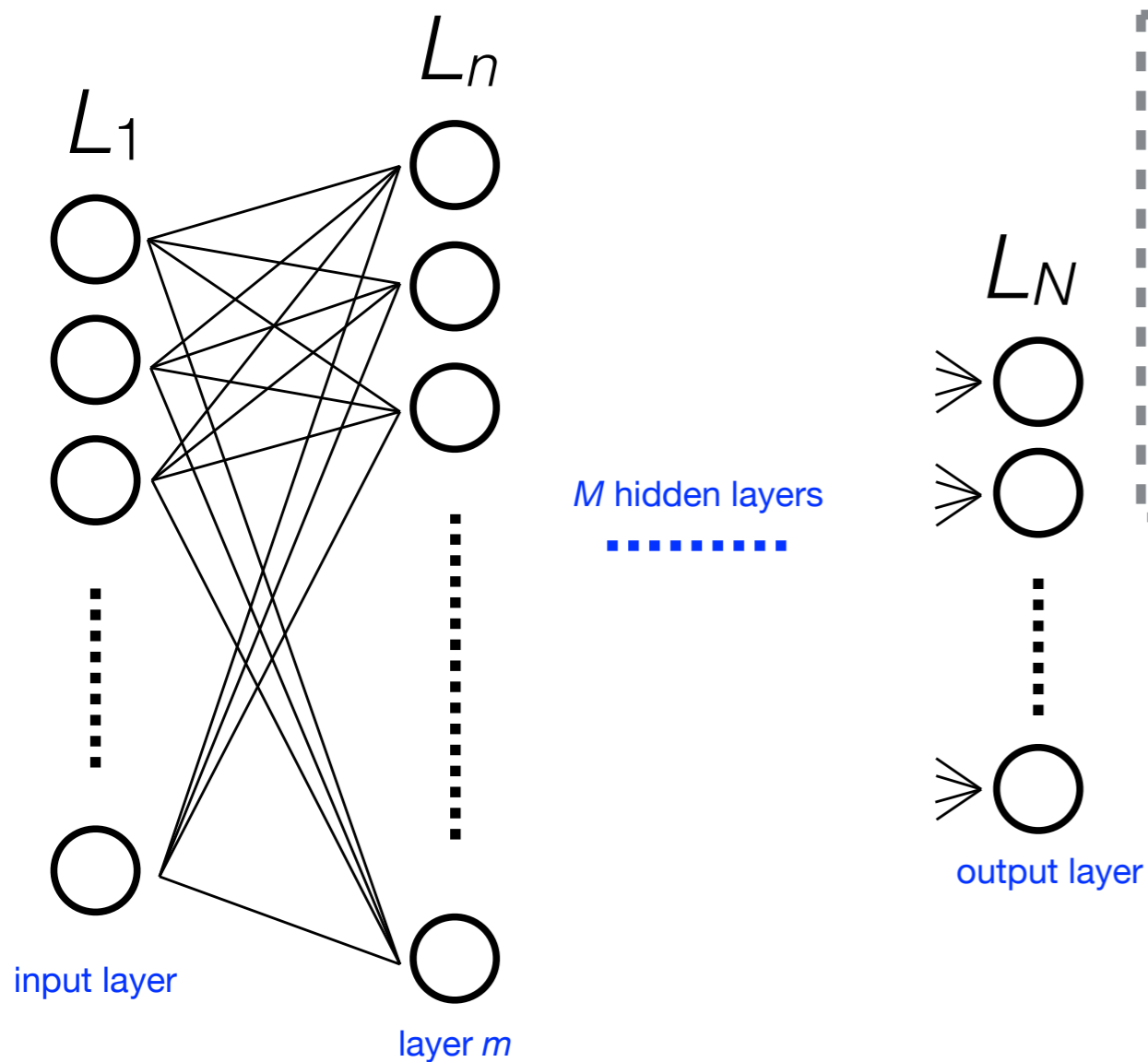
↑ multiplication
DSPs

↖ addition
logic cells



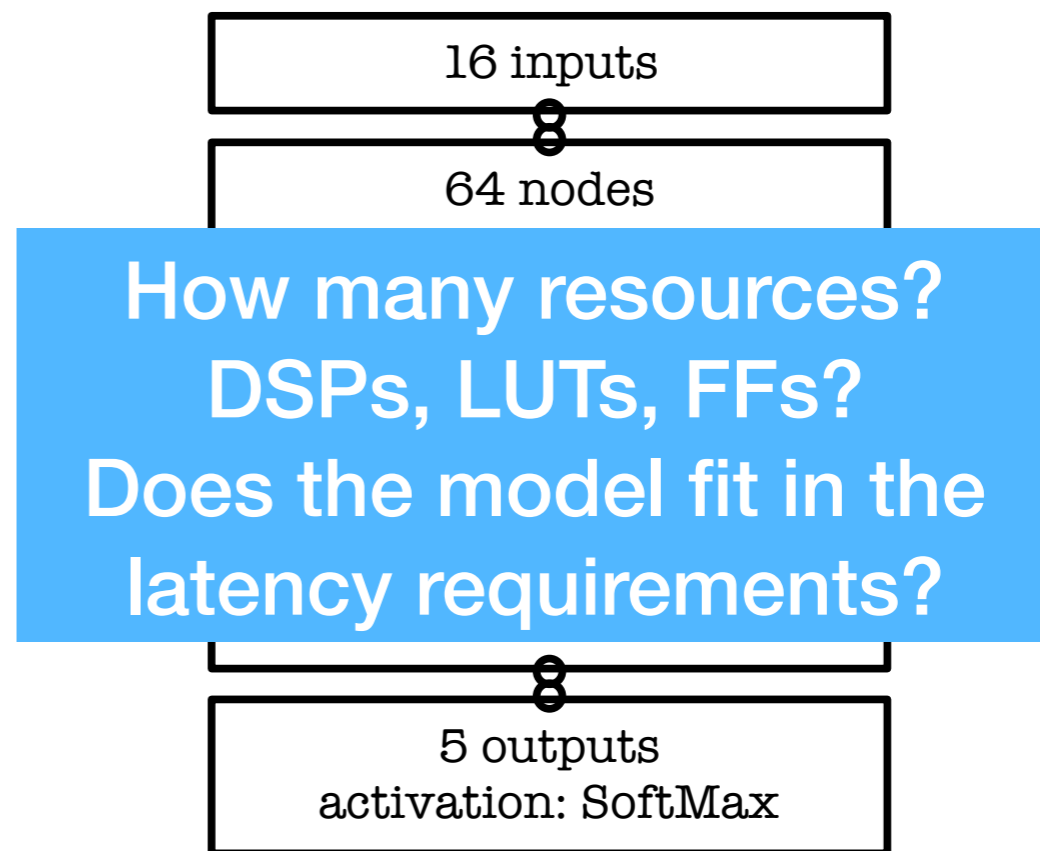
$$N_{\text{multiplications}} = \sum_{n=2}^N L_{n-1} \times L_n$$

Neural network inference



$$\mathbf{x}_n = g_n(\mathbf{W}_{n,n-1}\mathbf{x}_{n-1} + \mathbf{b}_n)$$

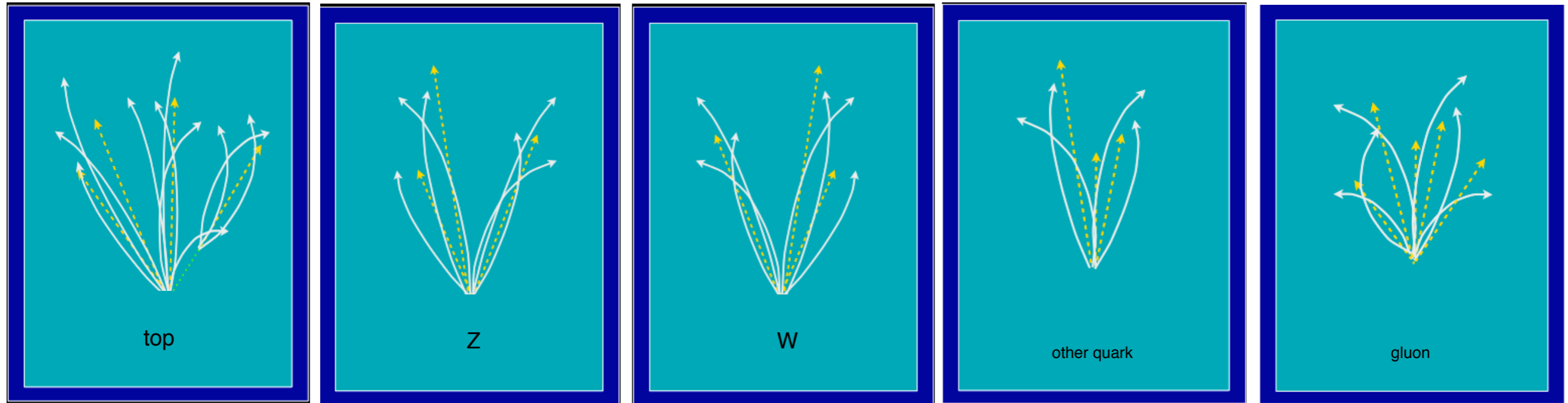
↗ activation function ↗ multiplication ↖ addition
precomputed and stored in BRAMs DSPs logic cells



$$N_{\text{multiplications}} = \sum_{n=2}^N L_{n-1} \times L_n$$

Case study: jet tagging

Study a multi-classification task: discrimination between highly energetic (boosted) q, g, W, Z, t initiated jets



$t \rightarrow bW \rightarrow bqq$

$Z \rightarrow qq$

$W \rightarrow qq$

q/g background

3-prong jet

2-prong jet

2-prong jet

no substructure
and/or mass ~ 0

Signal: reconstructed as one massive jet with substructure

Jet substructure observables used to distinguish signal vs background [*]

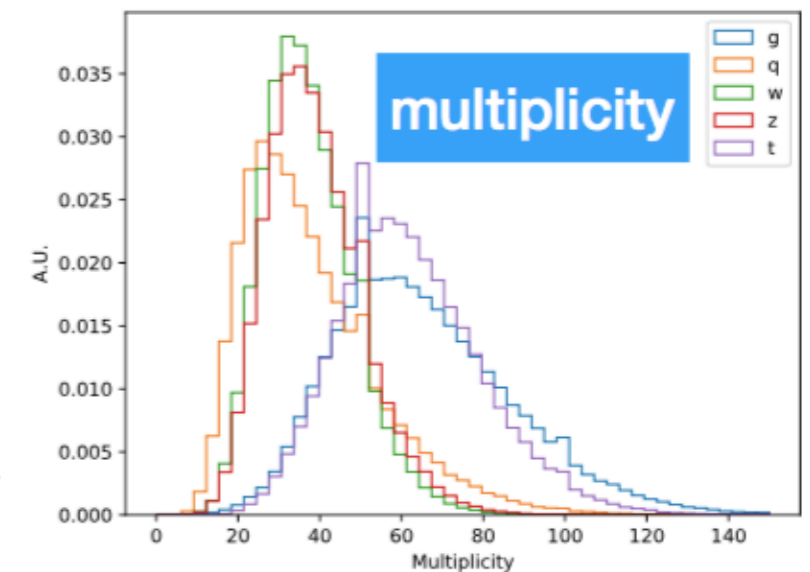
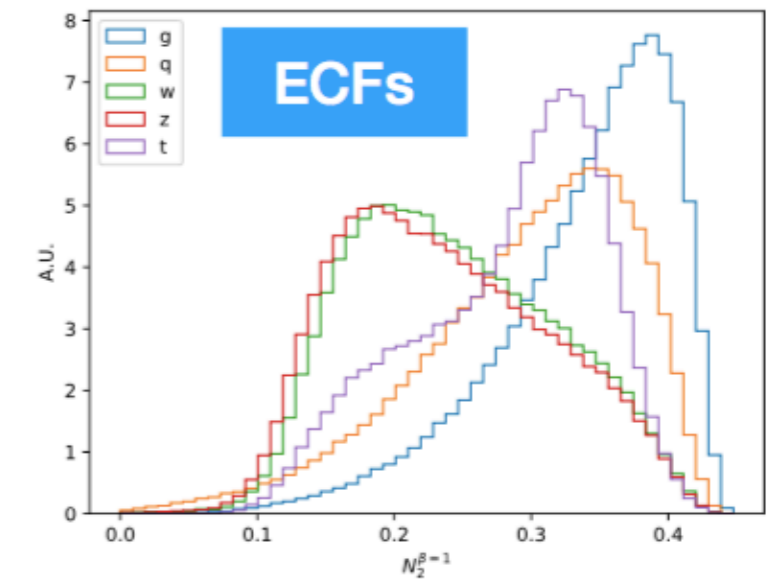
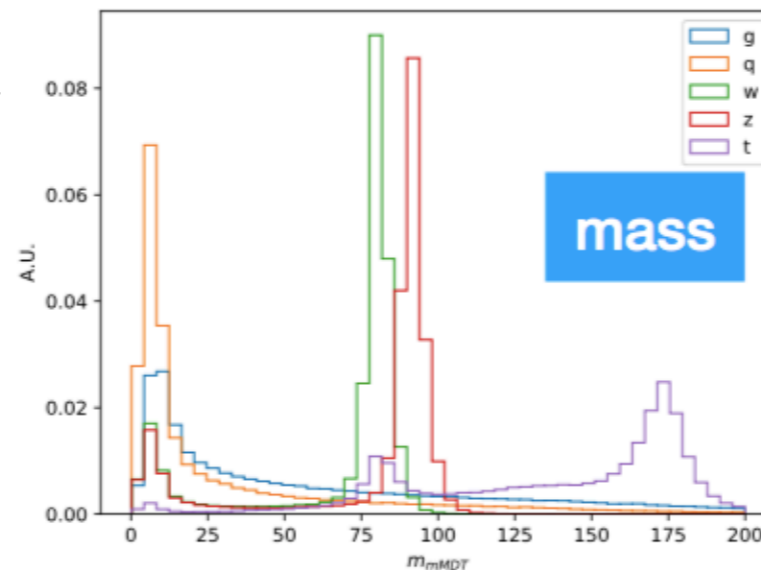
[*] D. Guest et al. [PhysRevD.94.112002](#), G. Kasieczka et al. [JHEP05\(2017\)006](#), J. M. Butterworth et al. [PhysRevLett.100.242001](#), etc..

Jet substructure features

Jet substructure observables provide large discrimination power between these types of jets

mass, multiplicity, energy correlation functions, ...
(computed with *FastJet* [*])

[*] E. Coleman et al. [JINST13\(2018\) T01003](#),
M. Cacciari et al, [Eur. Phys. J.C72\(2012\)1896](#)



These are expert-level features

Not necessarily realistic for L1 trigger

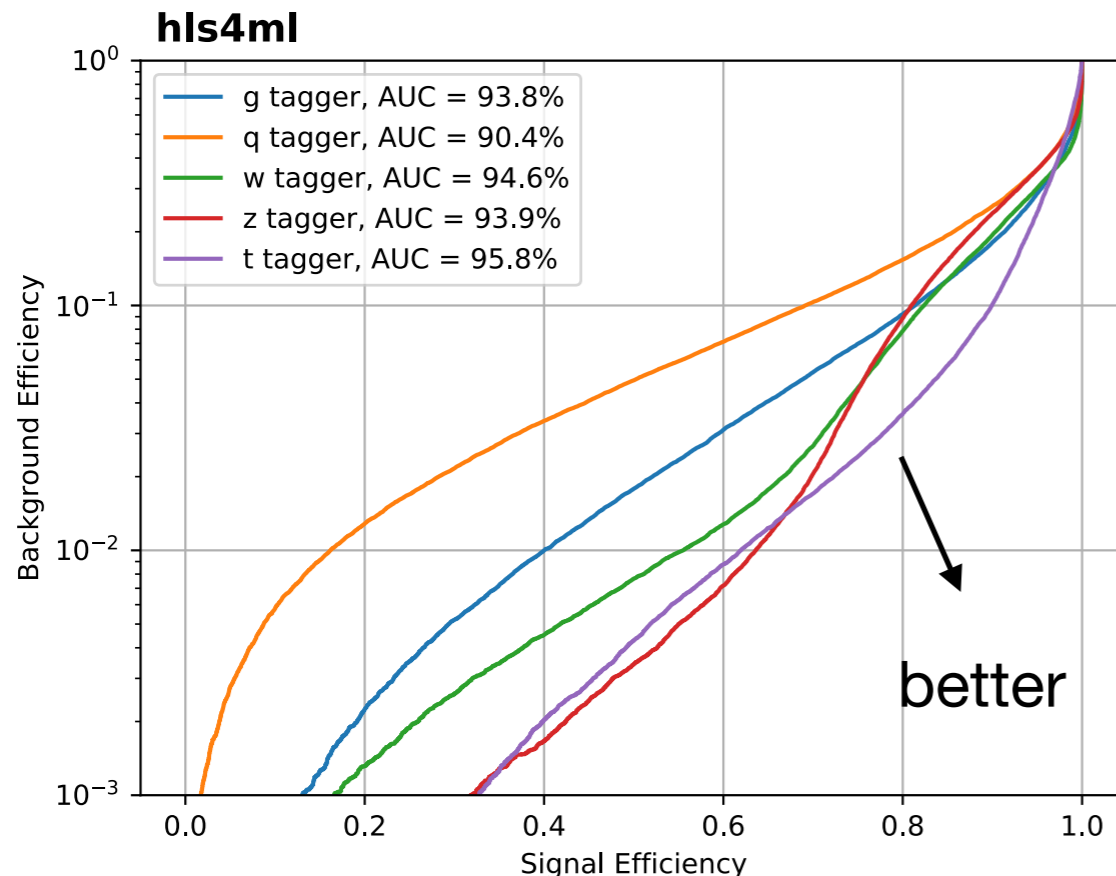
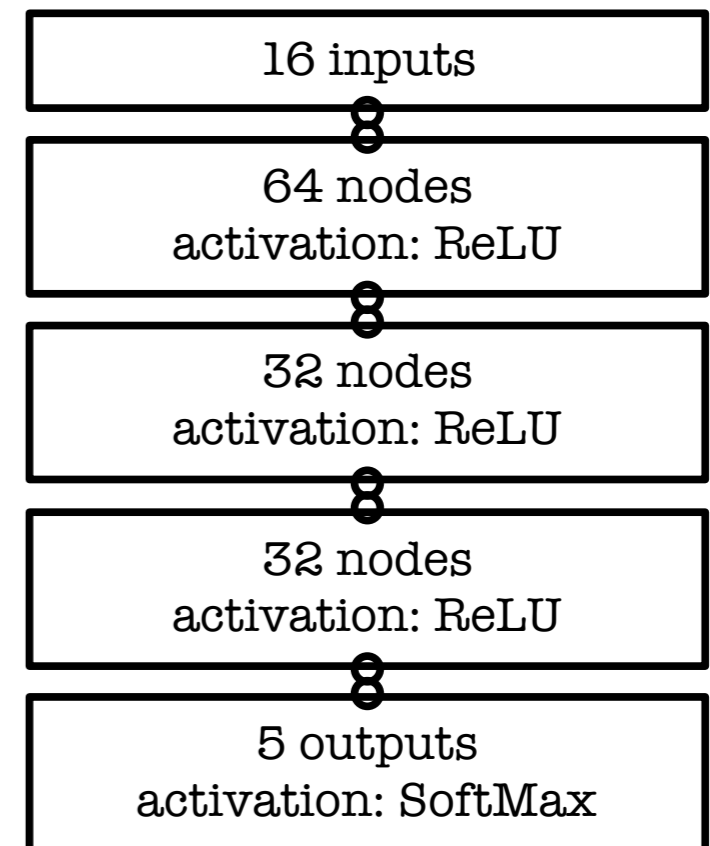
“Raw” particle candidates more suitable (*to be studied next*)

But lessons here are generic

One more case: $H \rightarrow bb$ discrimination vs $W/Z \rightarrow qq$ requires more “raw” inputs for b-tagging information

Case study: jet tagging

- We train (on GPU) a **five output multi-classifier**: sample of events with two boosted WW/ZZ/tt/qq/gg anti- k_T jets, generated with Madgraph and showered with Pythia8
- Fully connected neural network with **16 expert inputs**:
 - Relu activation function for intermediate layers
 - Softmax activation function for output layer



**AUC = area under ROC curve
(100% is perfect, 20% is random)**

Efficient NN design for FPGAs

FPGAs provide huge flexibility

Performance depends on how well you take advantage of this

Constraints:

Input bandwidth

FPGA resources

Latency

We have three handles:

- **compression:** reduce number of synapses or neurons
- **quantization:** reduces the precision of the calculations (inputs, weights, biases)
- **parallelization:** tune how much to parallelize to make the inference faster/slower versus FPGA resources

Efficient NN design for FPGAs

FPGAs provide huge flexibility

Performance depends on how well you take advantage of this

Constraints:

Input bandwidth

FPGA resources

Latency

We have three handles:

- **compression:** reduce number of synapses or neurons
- **quantization:** reduces the precision of the calculations (inputs, weights, biases)
- **parallelization:** tune how much to parallelize to make the inference faster/slower versus FPGA resources

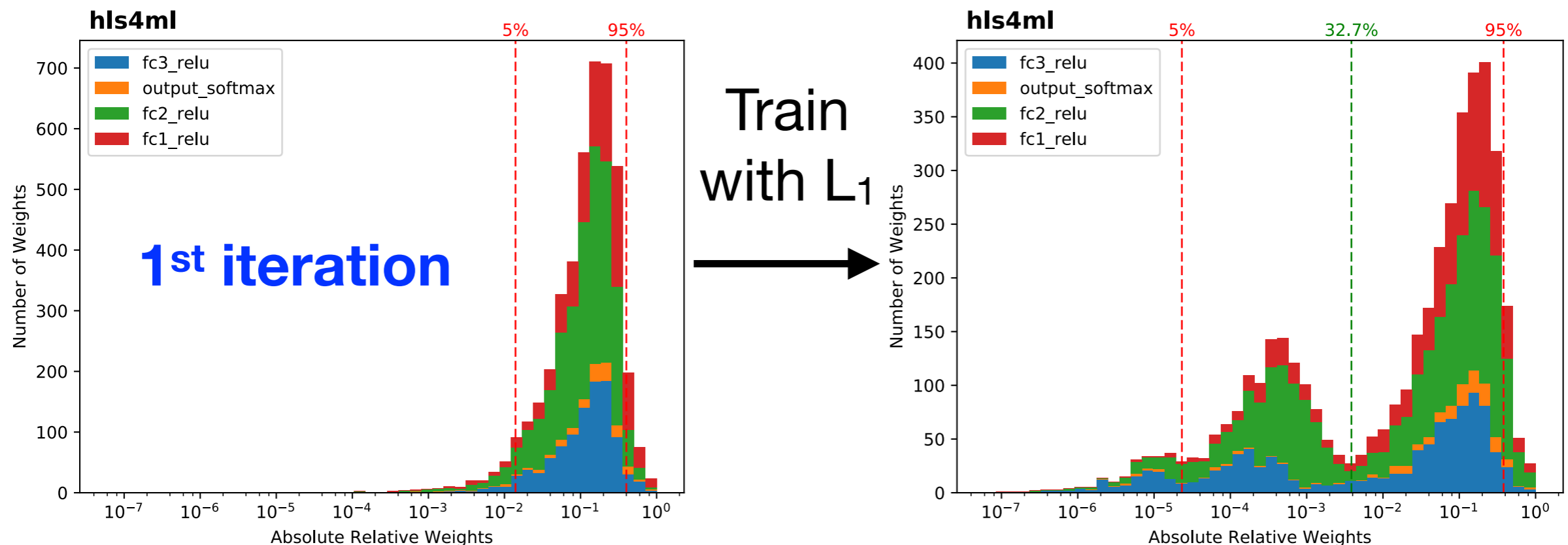
Efficient NN design: compression

- Iterative approach:

- train with **L1 regularization** (loss function augmented with penalty term):

$$L_{\lambda}(\vec{w}) = L(\vec{w}) + \lambda ||\vec{w}_1||$$

- sort the weights based on the value relative to the max value of the weights in that layer



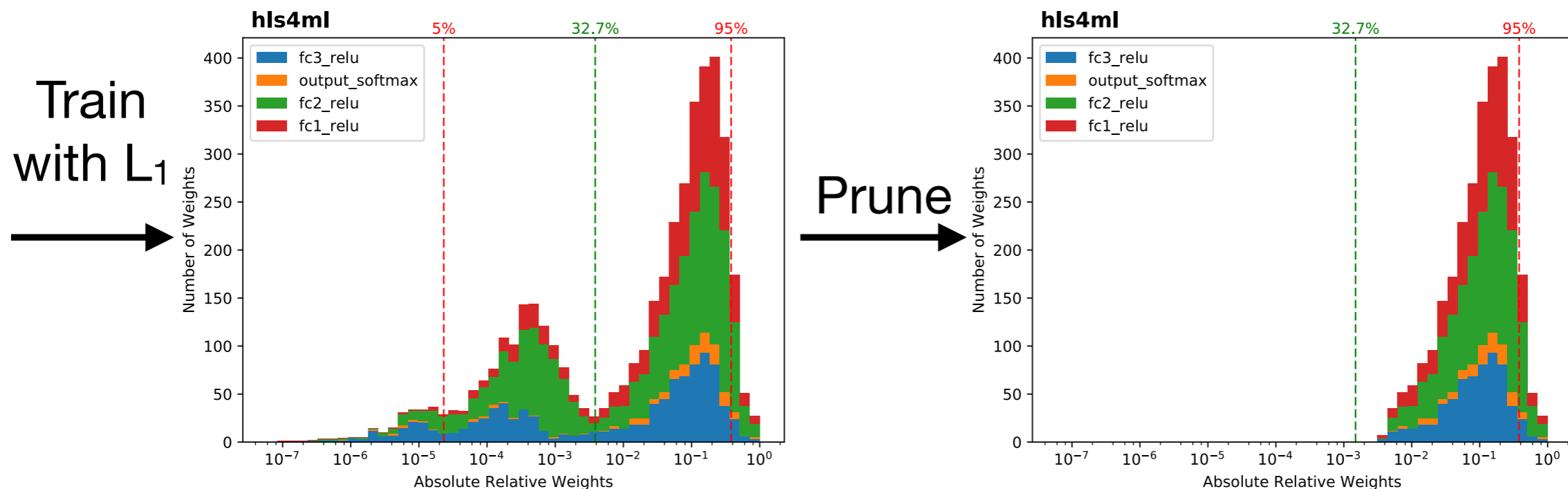
Efficient NN design: compression

- Iterative approach:

- train with **L1 regularization** (loss function augmented with penalty term):

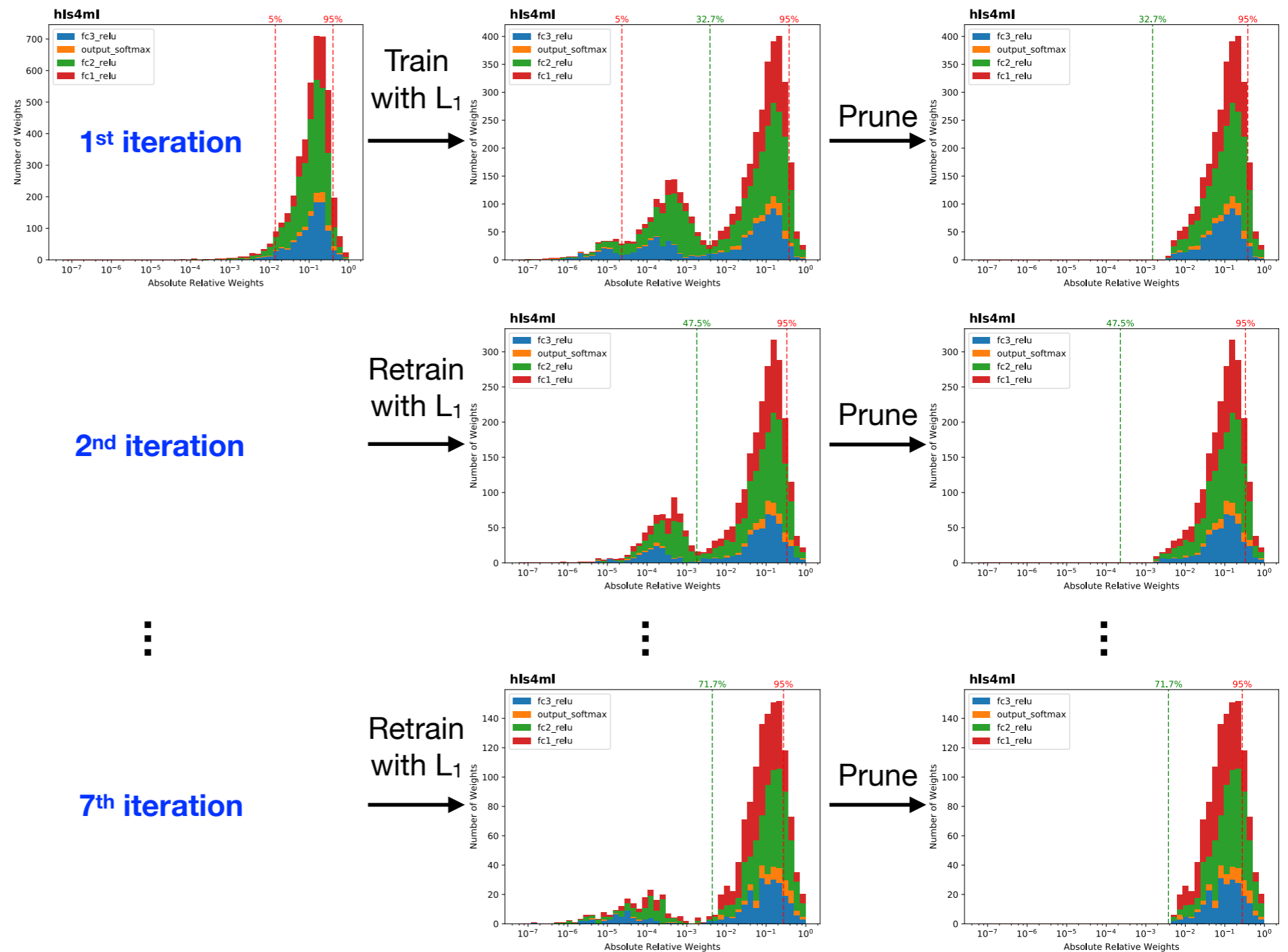
$$L_{\lambda}(\vec{w}) = L(\vec{w}) + \lambda ||\vec{w}_1||$$

- sort the weights based on the value relative to the max value of the weights in that layer
- prune weights falling below a certain percentile and retrain



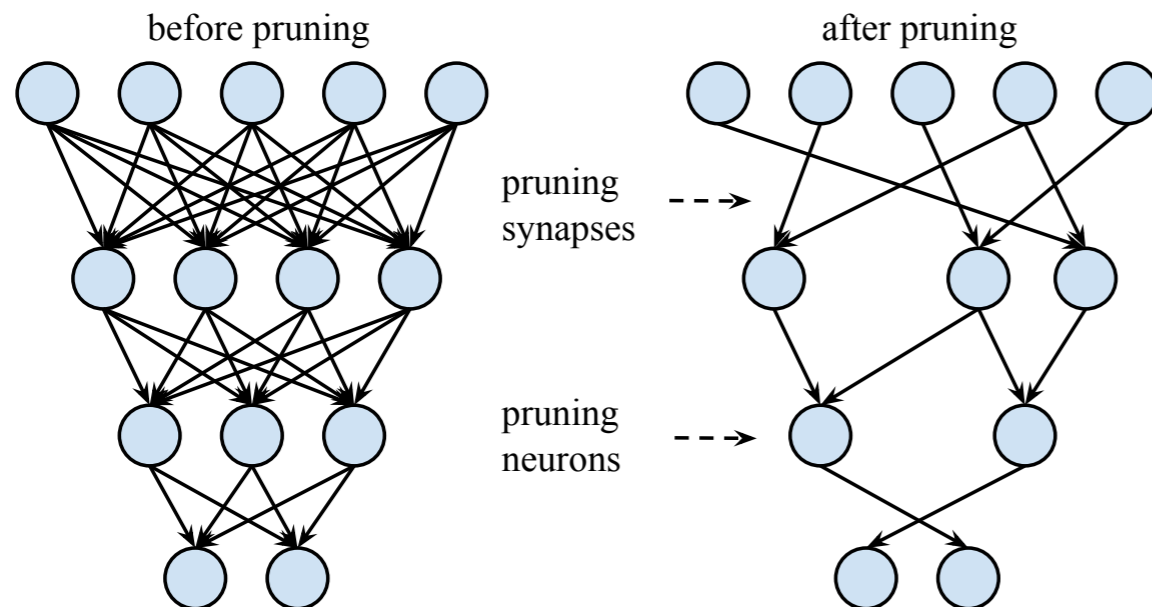
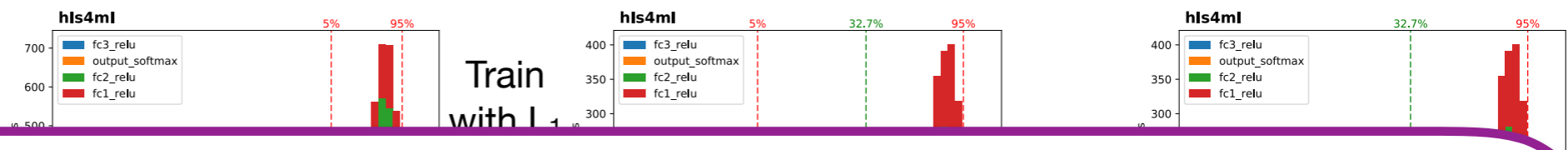
Efficient NN design: compression

Prune and repeat the train for 7 iterations



Efficient NN design: compression

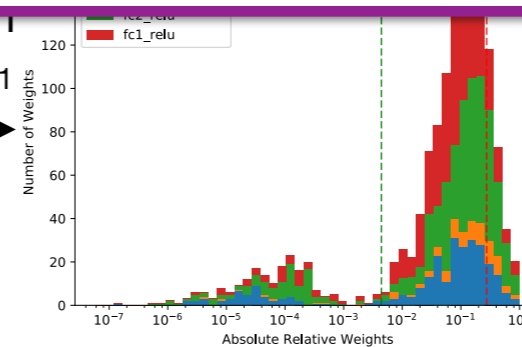
Prune and repeat the train for 7 iterations



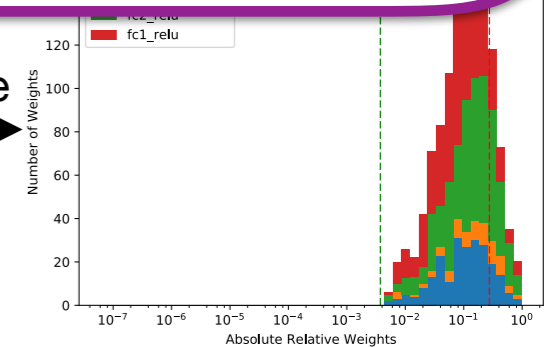
→ 70% reduction of weights and multiplications w/o performance loss

7th iteration

retrain with L1



Prune



Efficient NN design for FPGAs

FPGAs provide huge flexibility

Performance depends on how well you take advantage of this

Constraints:

Input bandwidth

FPGA resources

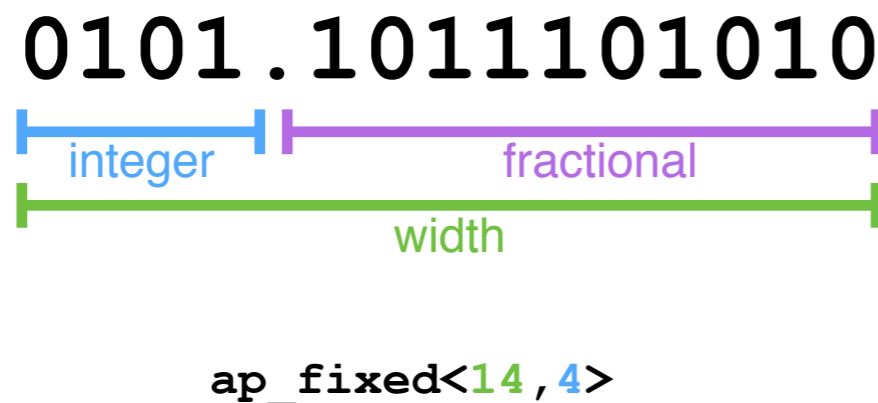
Latency

We have three handles:

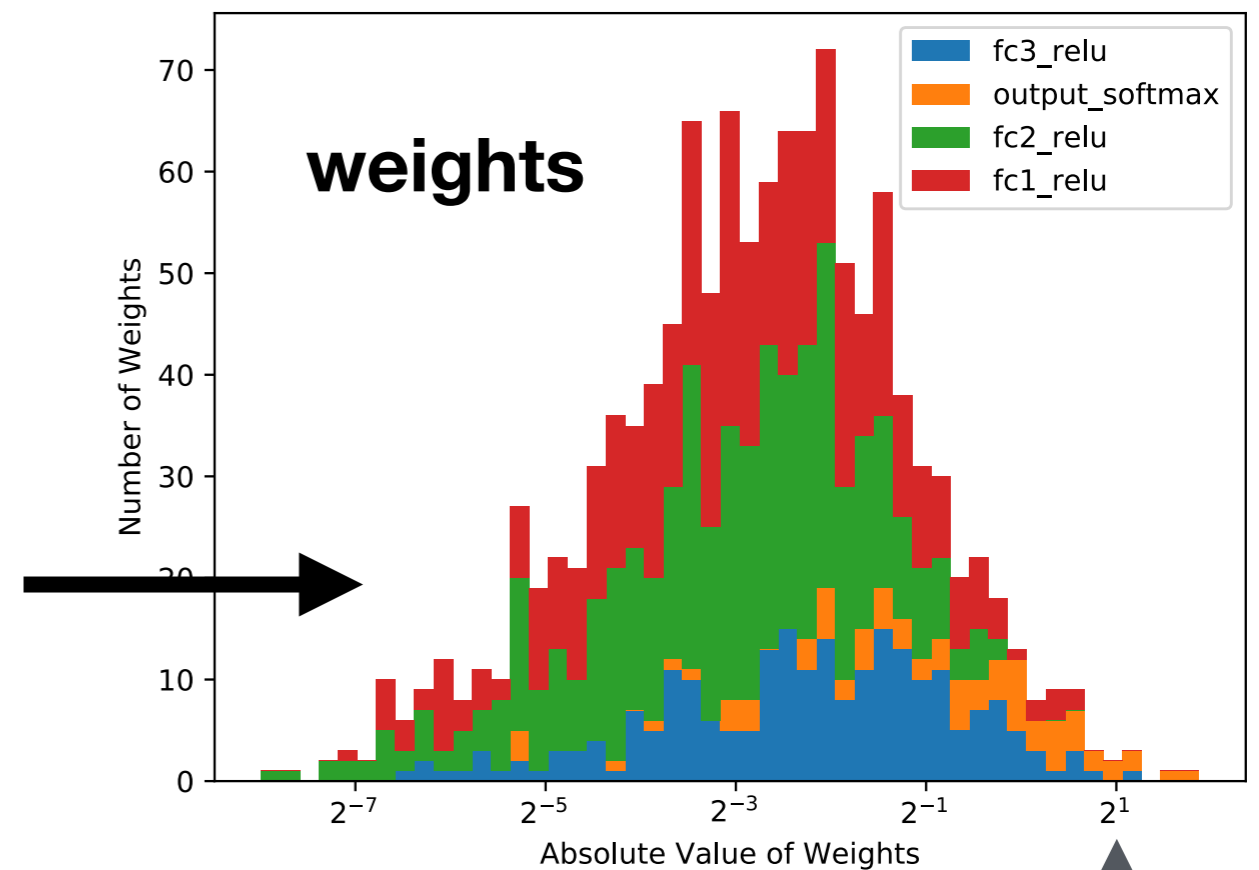
- **compression:** reduce number of synapses or neurons
- **quantization:** reduces the precision of the calculations (inputs, weights, biases)
- **parallelization:** tune how much to parallelize to make the inference faster/slower versus FPGA resources

Efficient NN design: quantization

- In FPGAs use **fixed point data types** → less resources and latency than 32-bit floating point
- NN inputs, weights, biases, outputs represented as `ap_fixed<width,integer>`



- To avoid overflow/underflow of weights at least 3 bits needed
- But need more bits for neurons as computed with multiplications and sums → we perform a **scan of physics performance versus bit precision**



integer bits = 2 + 1 for sign
(need more for neurons)

Efficient NN design for FPGAs

FPGAs provide huge flexibility

Performance depends on how well you take advantage of this

Constraints:

Input bandwidth

FPGA resources

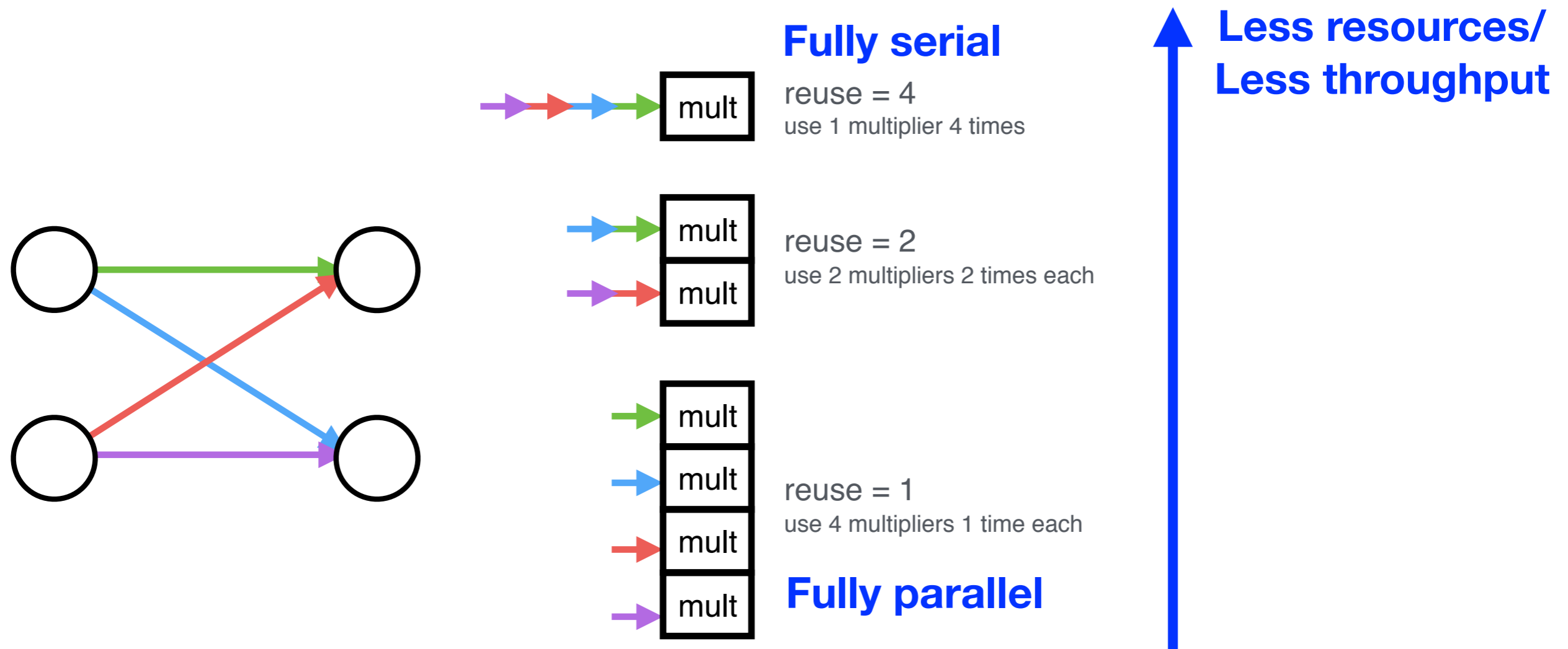
Latency

We have three handles:

- **compression:** reduce number of synapses or neurons
- **quantization:** reduces the precision of the calculations (inputs, weights, biases)
- **parallelization:** tune how much to parallelize to make the inference faster/slower versus FPGA resources

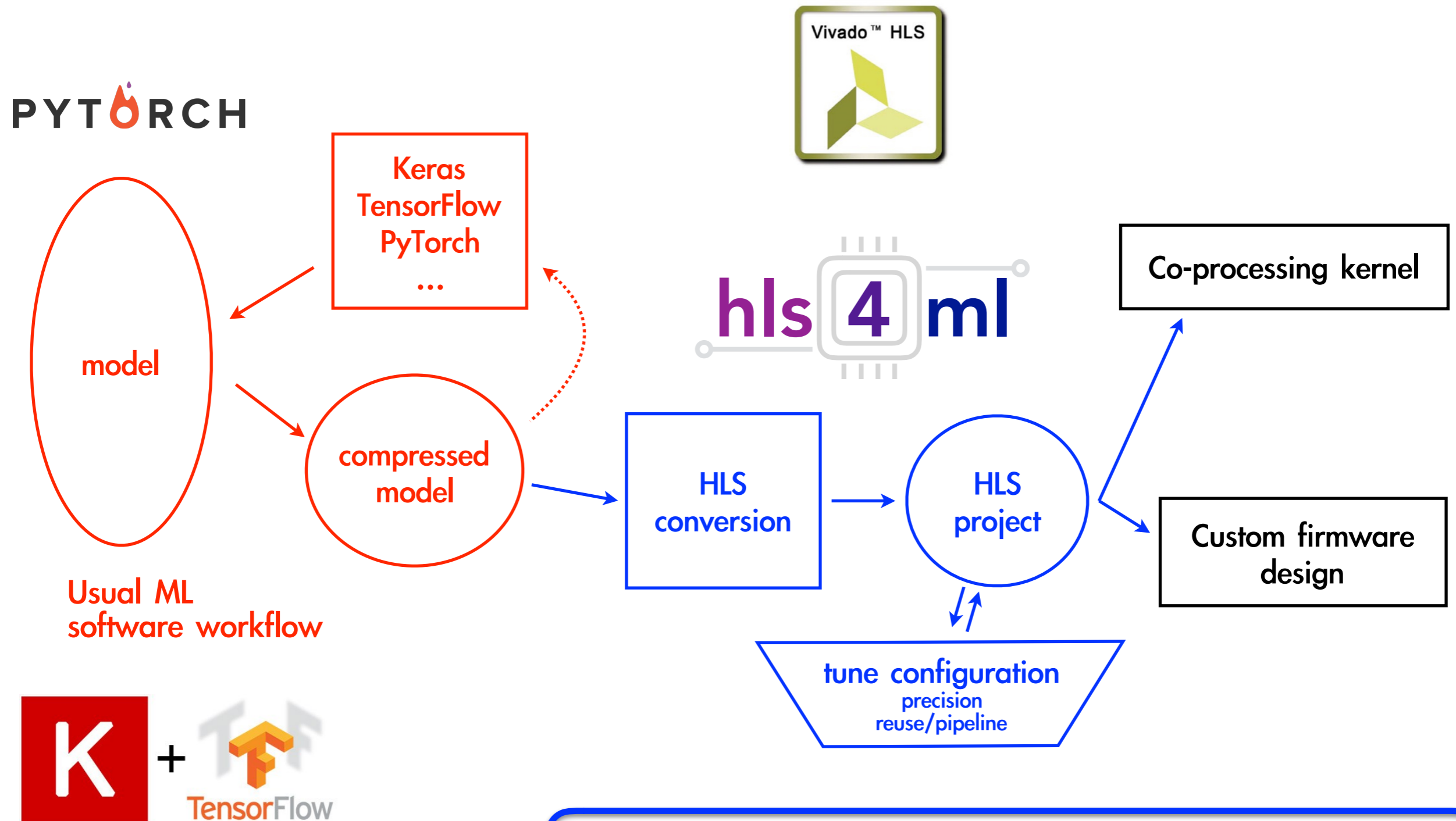
Efficient NN design: parallelization

- Trade-off between latency and FPGA resource usage determined by the parallelization of the calculations in each layer
- Configure the “reuse factor” = number of times a multiplier is used to do a computation



Reuse factor: how much to parallelize operations in a hidden layer

high level synthesis for machine learning



<https://hls-fpga-machine-learning.github.io/hls4ml/>

The package

Translation



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

Inputs



Keras



```
KerasJson: example-keras-model-files/KERAS_1layer.json
KerasH5:   example-keras-model-files/KERAS_1layer_weights.h5
OutputDir: my-hls-test
ProjectName: myproject
XilinxPart: xc7vx690tffg1927-2
ClockPeriod: 5

IOType: io_parallel # options: io_serial/io_parallel
ReuseFactor: 1
DefaultPrecision: ap_fixed<18,8>
```

Config

- IOType: parallelize or serialize
- ReuseFactor: how much to parallelize
- DefaultPrecision: inputs, weights, biases

```
my-hls-test/:
build_prj.tcl
firmware
myproject_test.cpp
```

The package

Build HLS project



`vivado_hls -f build_prj.tcl`



```
#####  
#   HLS4ML   #  
#####  
open_project -reset myproject_prj  
set_top myproject  
add_files firmware/myproject.cpp -cflags "-I[file normalize ../../nnet_utils]"  
add_files -tb myproject_test.cpp -cflags "-I[file normalize ../../nnet_utils]"  
add_files -tb firmware/weights  
open_solution -reset "solution1"  
set_part {xc7k115-f1vf1924-2-i}  
create_clock -period 5 -name default  
csim_design  
csynth_design  
cosim_design -trace_level all  
export_design -format ip_catalog  
exit
```

Produce a firmware block in ~ minutes!

Study details

GOAL

Map out FPGA performance, resource usage and latency versus compression, quantization, and parallelization hyperparameters

SETUP

Xilinx Vivado 2017.2

HLS target clock frequency: 200 MHz (5 clocks/BX)

Kintex Ultrascale, xcku115-flvb2104-2-i

- 1.4M logic cells, 5,520 DSPs, 1.3M FFs, 700k LUTs, 2200 BRAMs

RESULTS

First examine resource usage coming from HLS estimate

Then discuss the exact resources given by the final implementation

Efficient NN design: quantization

ap_fixed<width,integer>

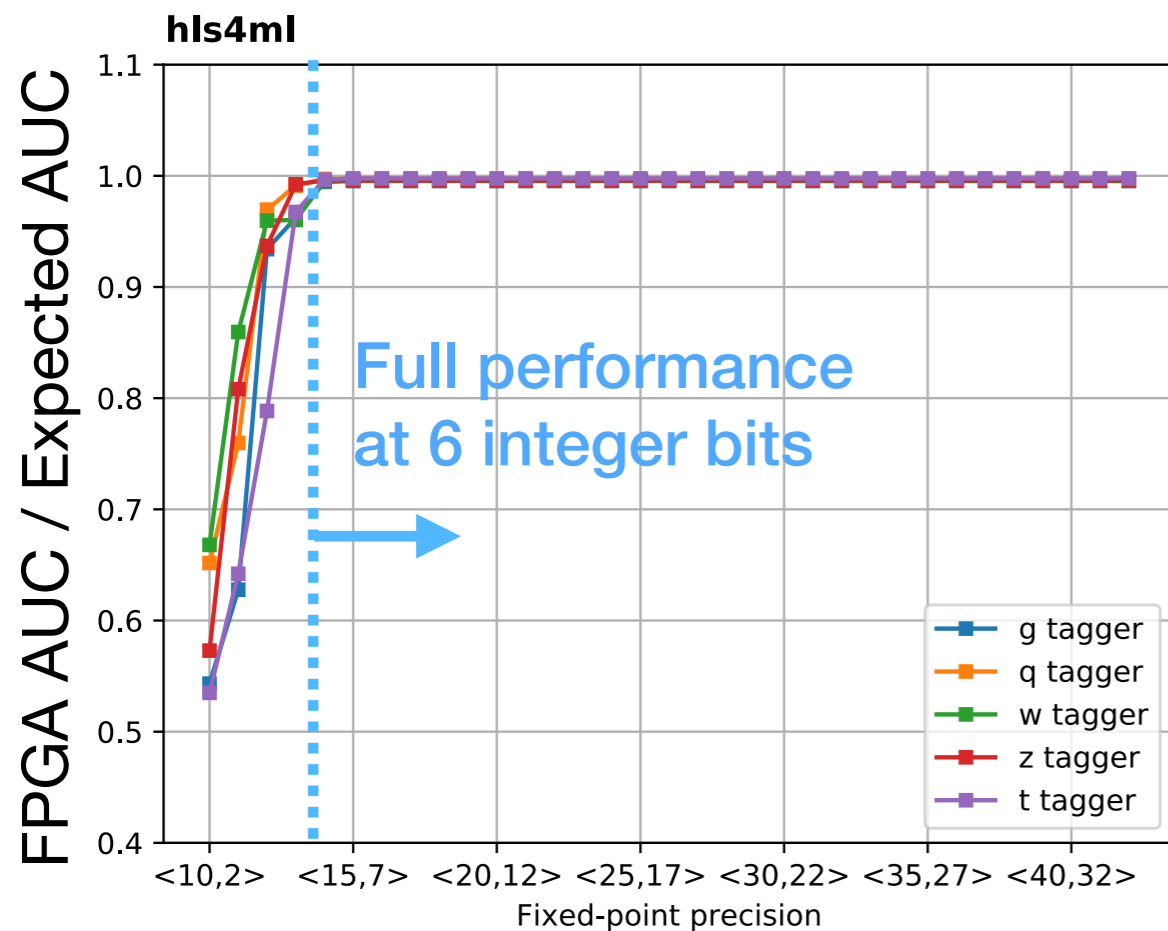
0101.1011101010



- Quantify the performance of the classifier with the AUC
- Expected AUC = AUC achieved by 32-bit floating point inference of the neural network

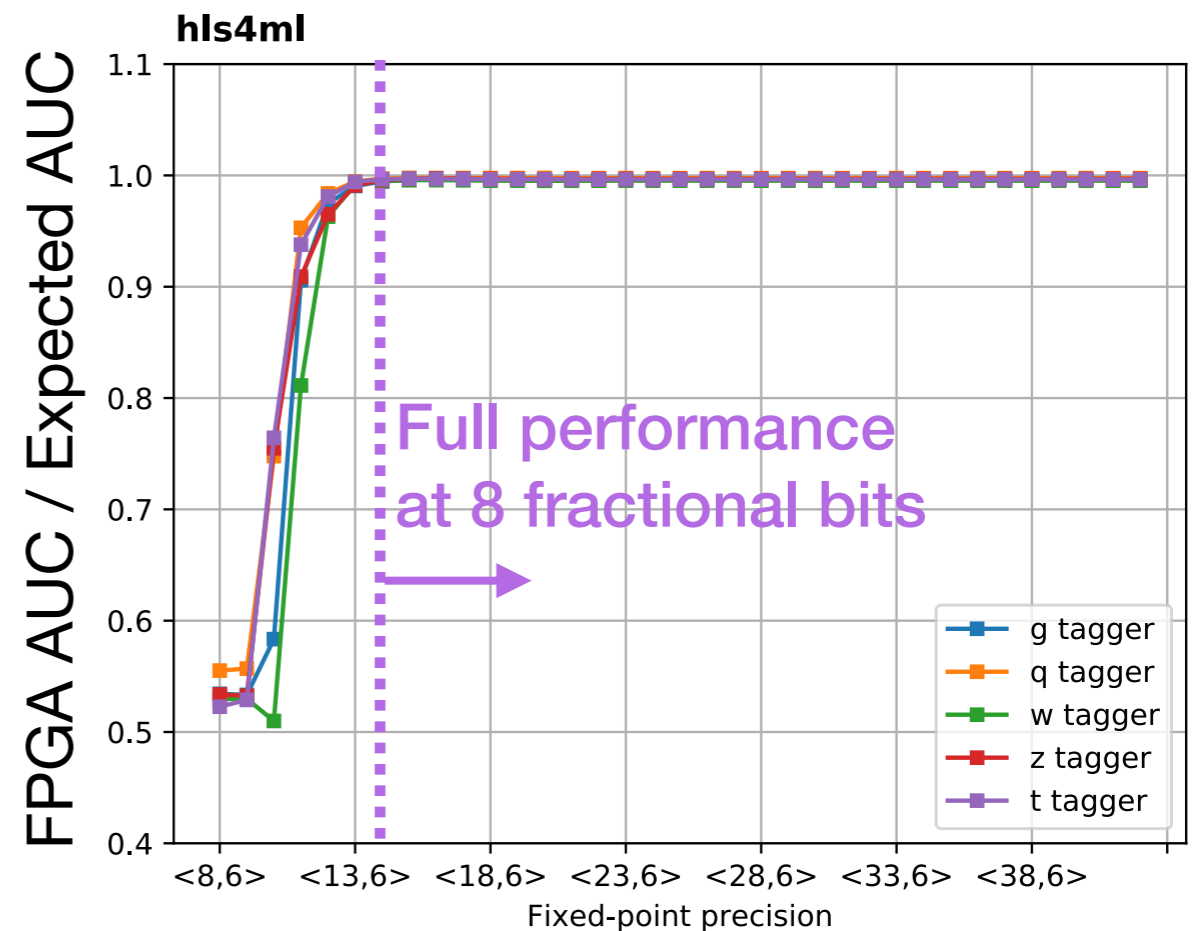
Scan integer bits

Fractional bits fixed to 8

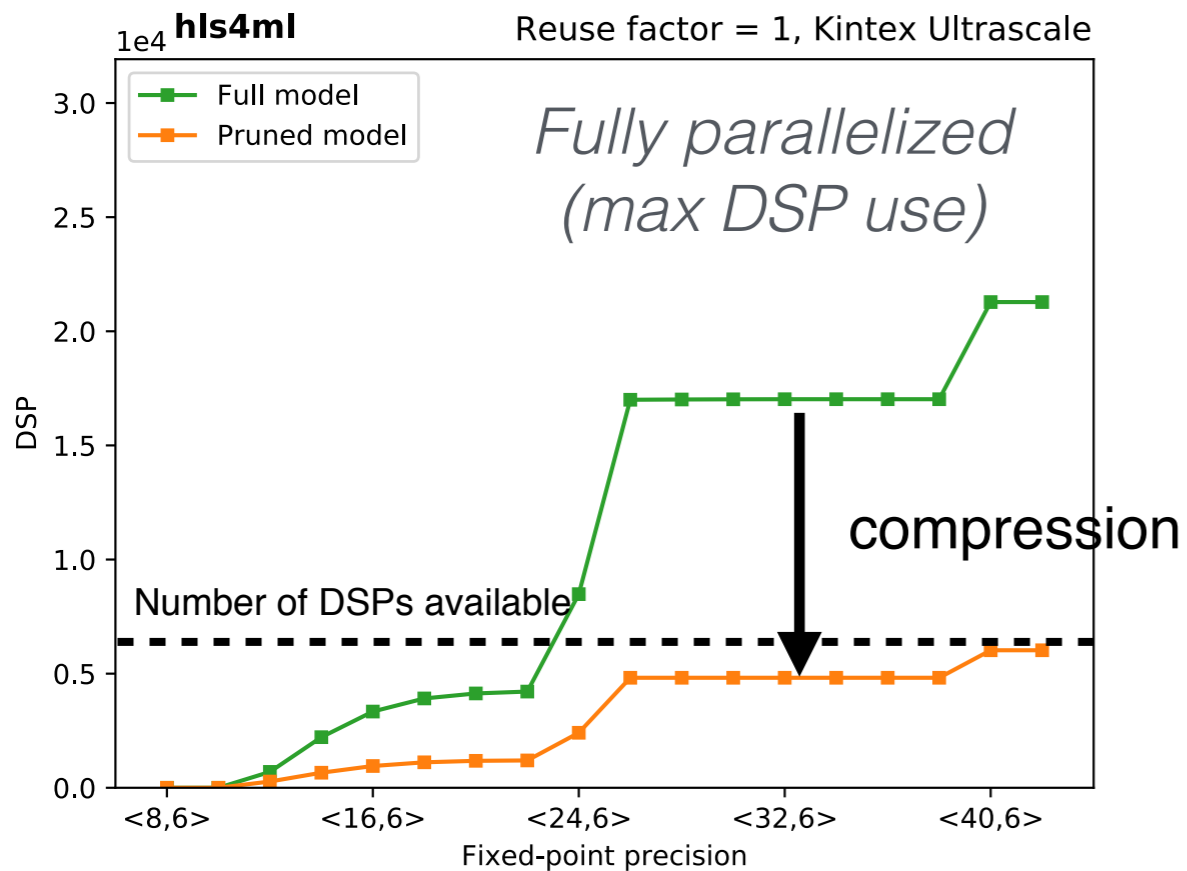


Scan fractional bits

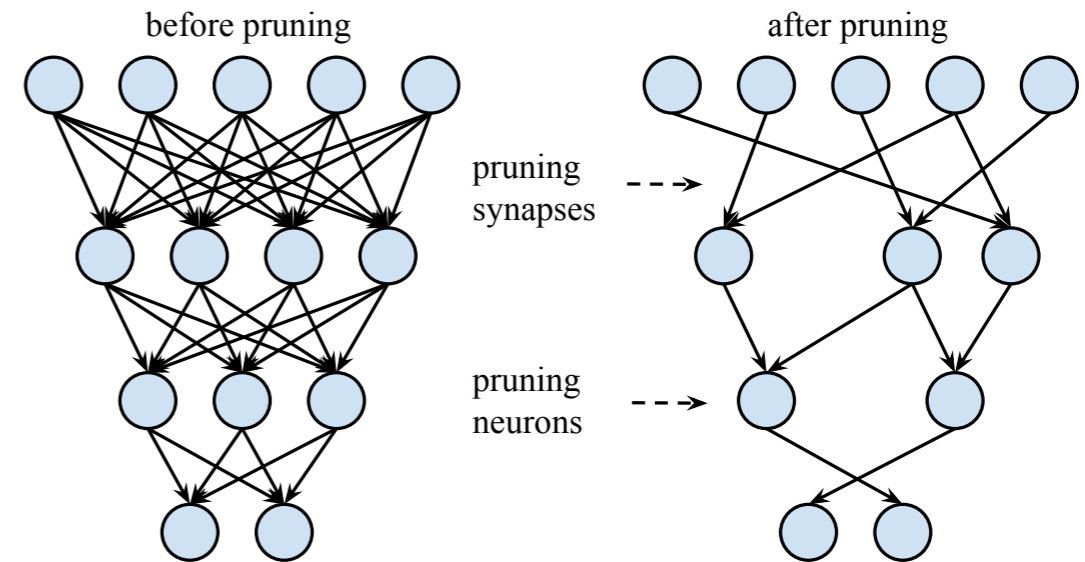
Integer bits fixed to 6



Efficient NN design: **compression**

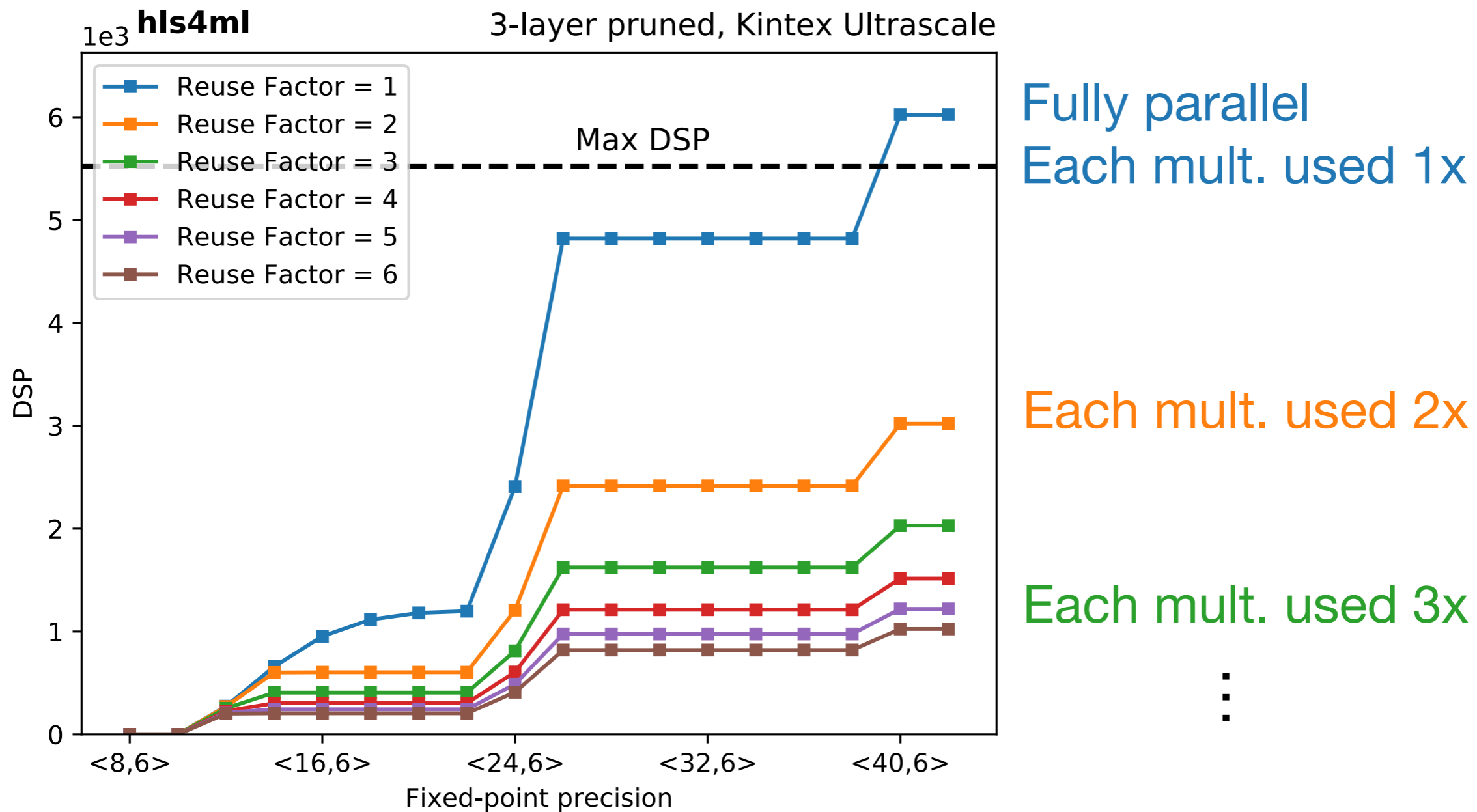


70% compression ~ 70% fewer DSPs

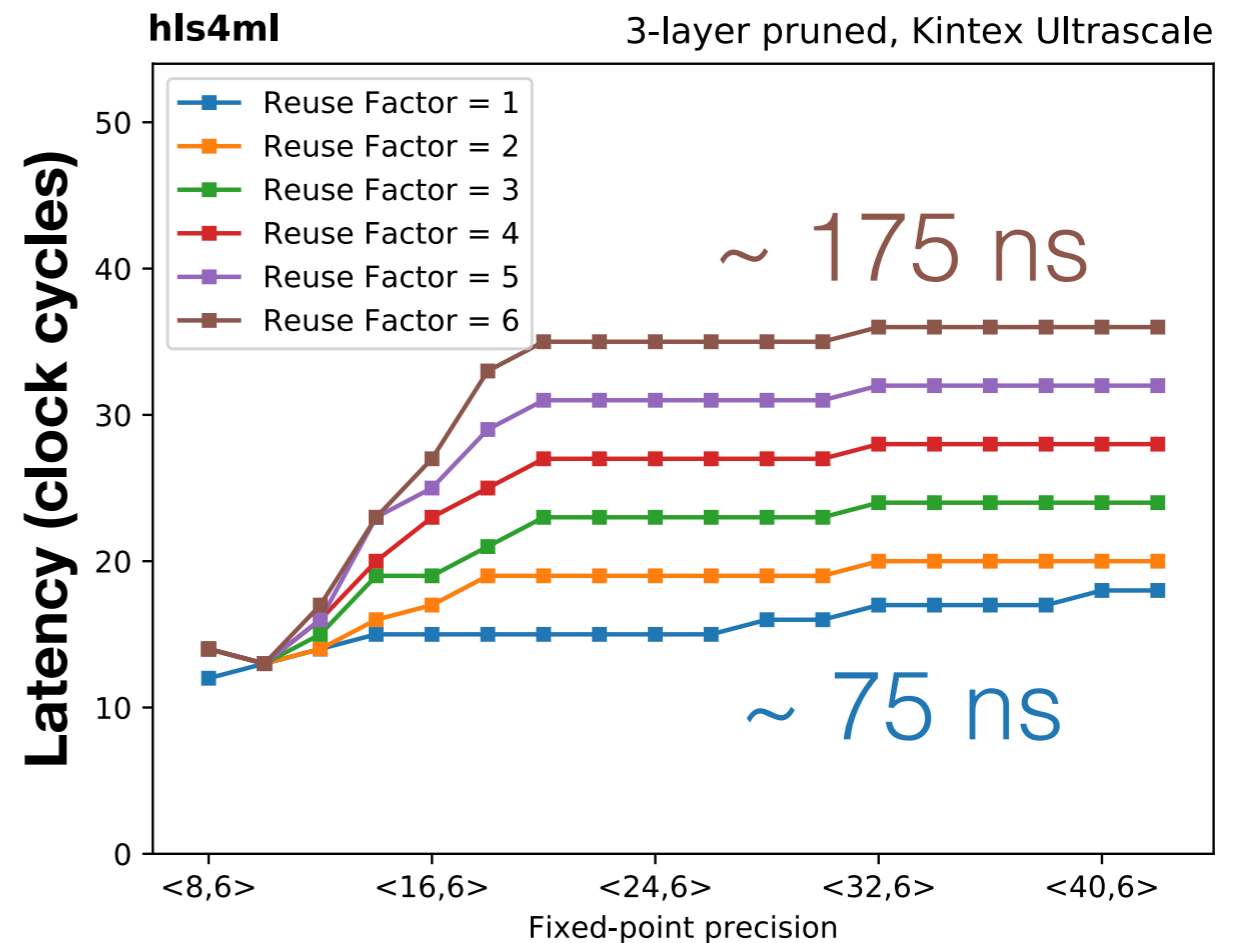
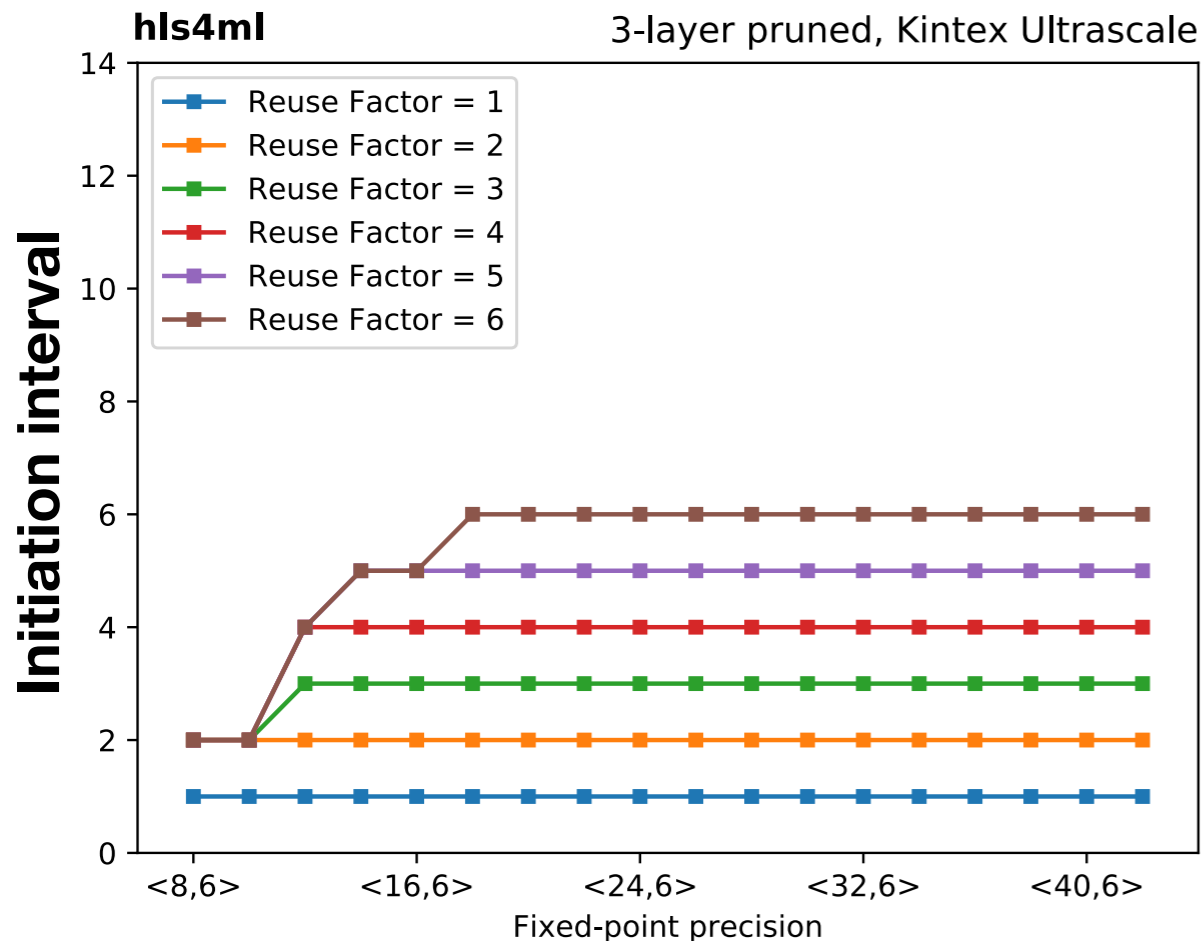


- DSPs (used for multiplication) are often limiting resource
 - maximum use when fully parallelized
 - DSPs have a max size for input (e.g. 27x18 bits), so number of DSPs per multiplication changes with precision

Parallelization: DSPs usage



Parallelization: Timing



- **Initiation interval:** number of clocks before accepting new inference inputs

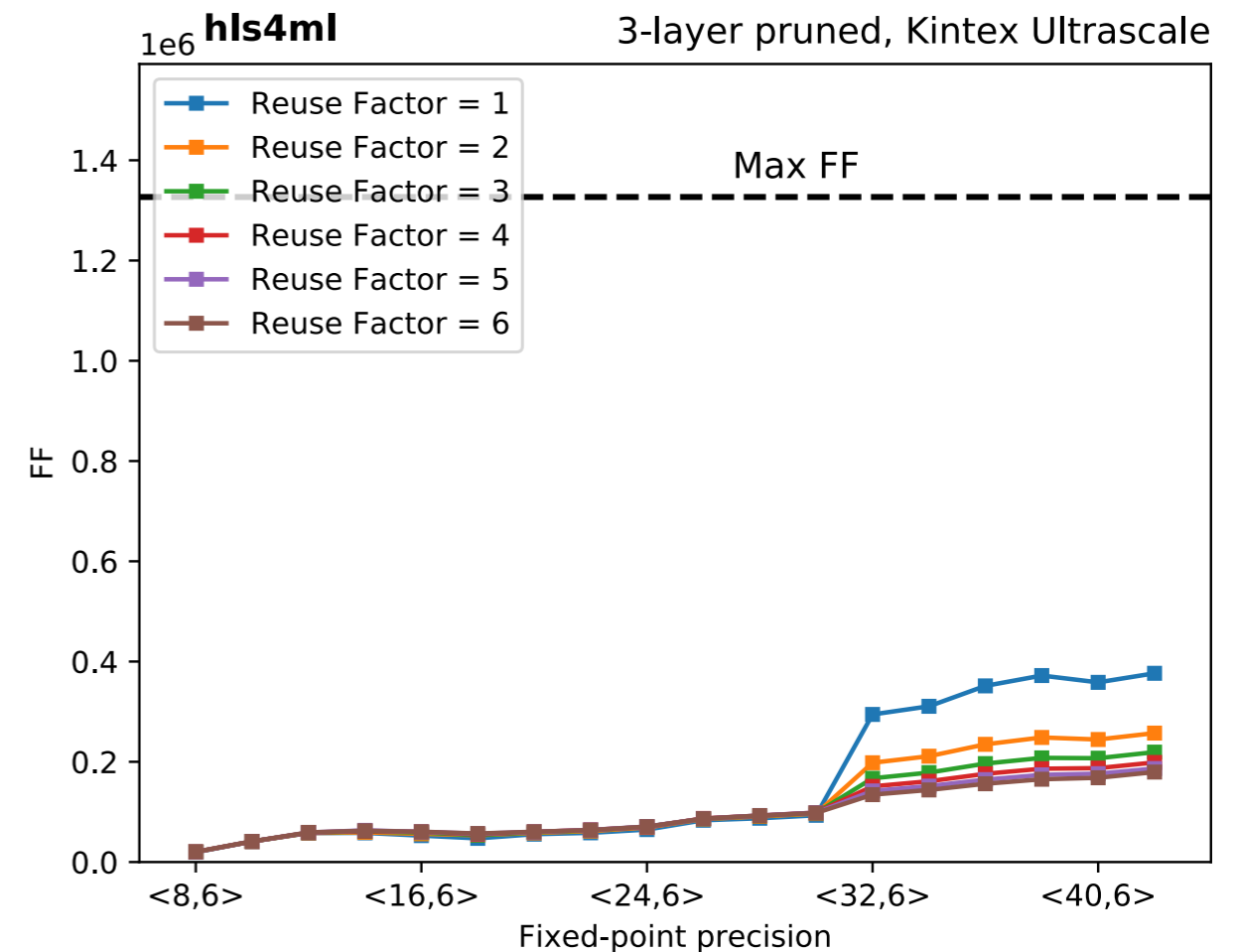
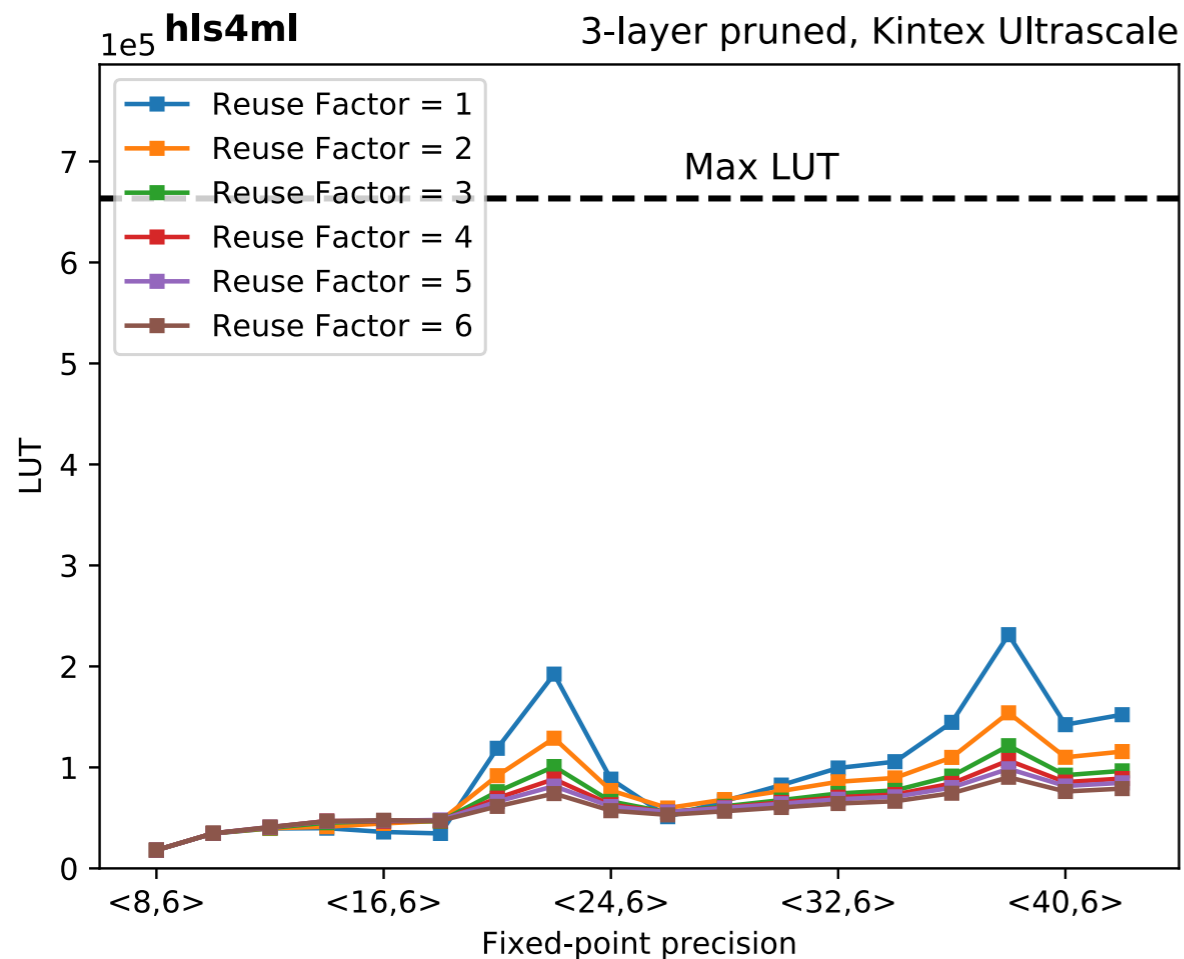
- scales with reuse factor
- for very low data precisions multiplications implemented through FFs and LUTs

- Additional **latency** introduced by reusing the multiplier

Latency of layer m

$$L_m = L_{\text{mult}} + (R - 1) \times II_{\text{mult}} + L_{\text{activ}}$$

Other resources: FFs and LUTs



Fairly linear increase with precision

Small percentage of total available

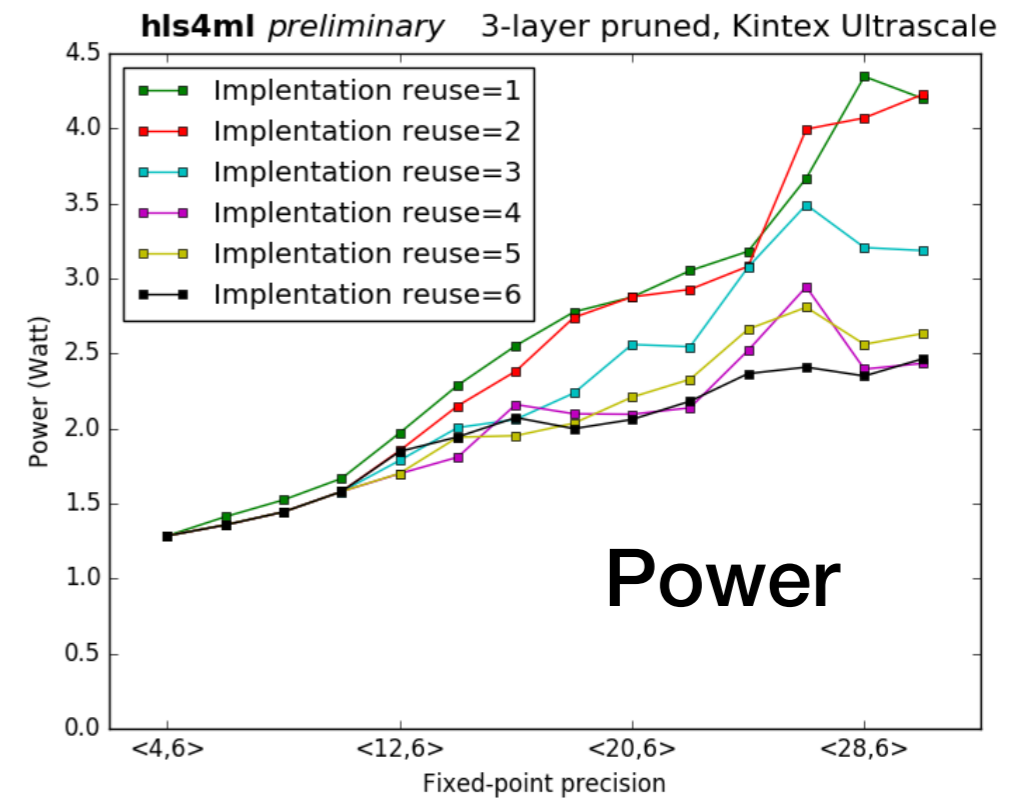
Spikes present at steep transitions in LUTs usage as artifacts of HLS synthesis

Not observed in implementation

Found also dependence on Vivado HLS version

Firmware implementation

- Final implementation gives actual resource usage and timing estimate
 - how optimal is the HLS design?
- Power usage increases with precision, it goes down for less throughput (higher reuse factor)



Firmware implementation

- Final implementation gives actual resource usage and timing estimate

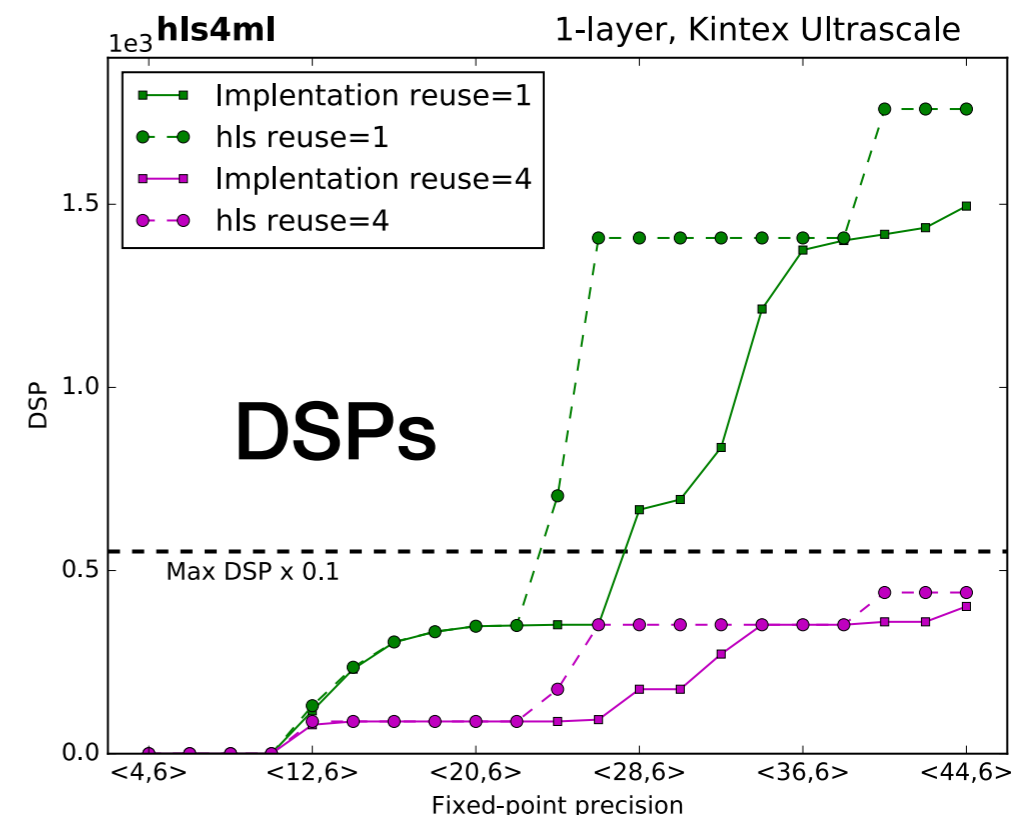
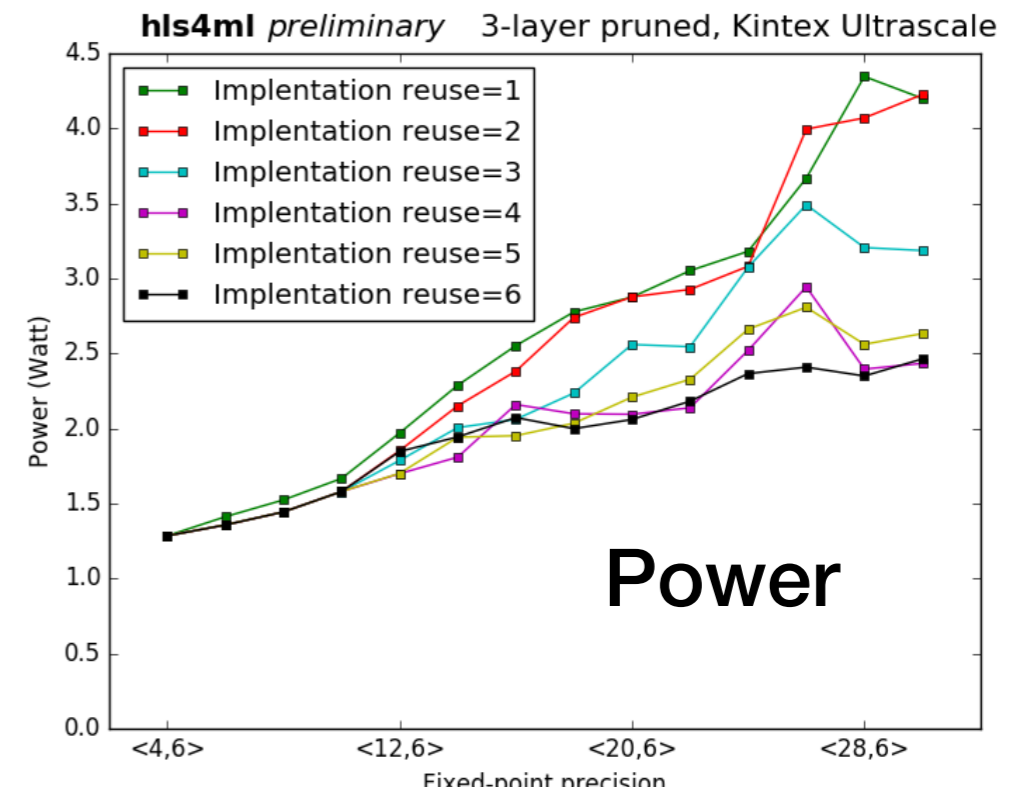
- how optimal is the HLS design?

- Power usage increases with precision, it goes down for less throughput (higher reuse factor)

- Implement a 1-layer NN, simply routing all firmware block's inputs and outputs to FPGA available pins

- HLS estimate on resource usage are conservative

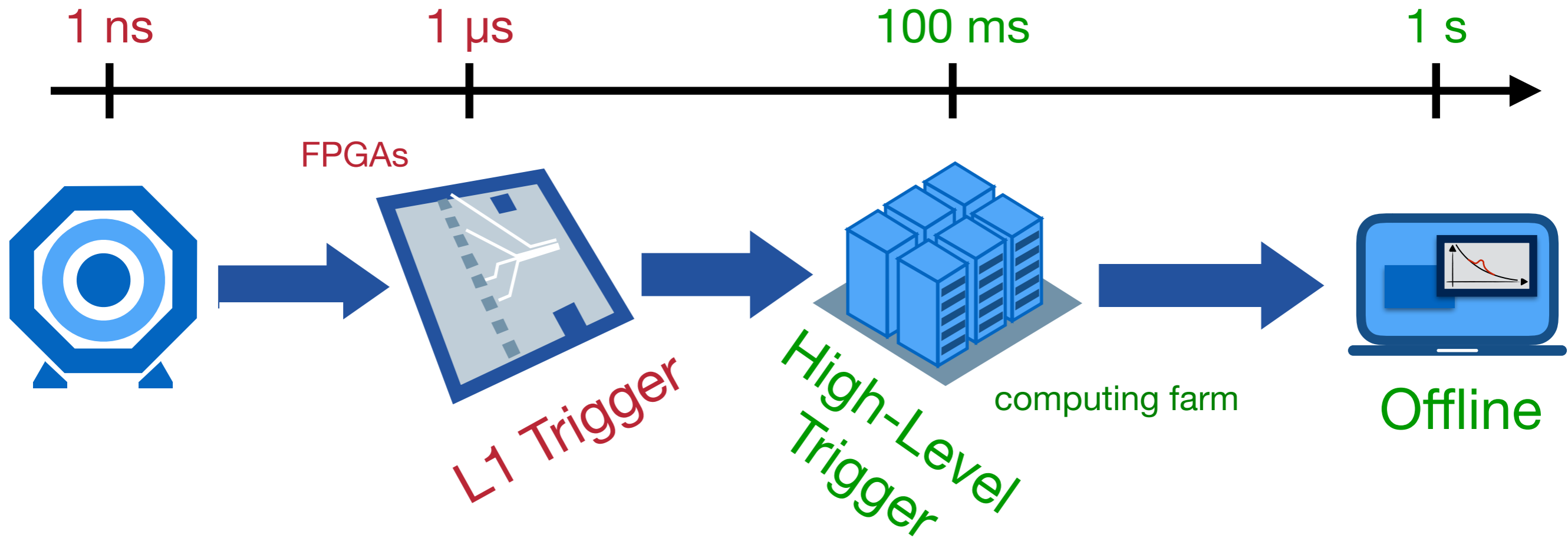
- DSPs usage agree well below DSP precision transition (27 bit), implementation does further optimization
- FFs and LUTs overestimated by a factor 2-4





Summary and outlook

The latency landscape @ LHC

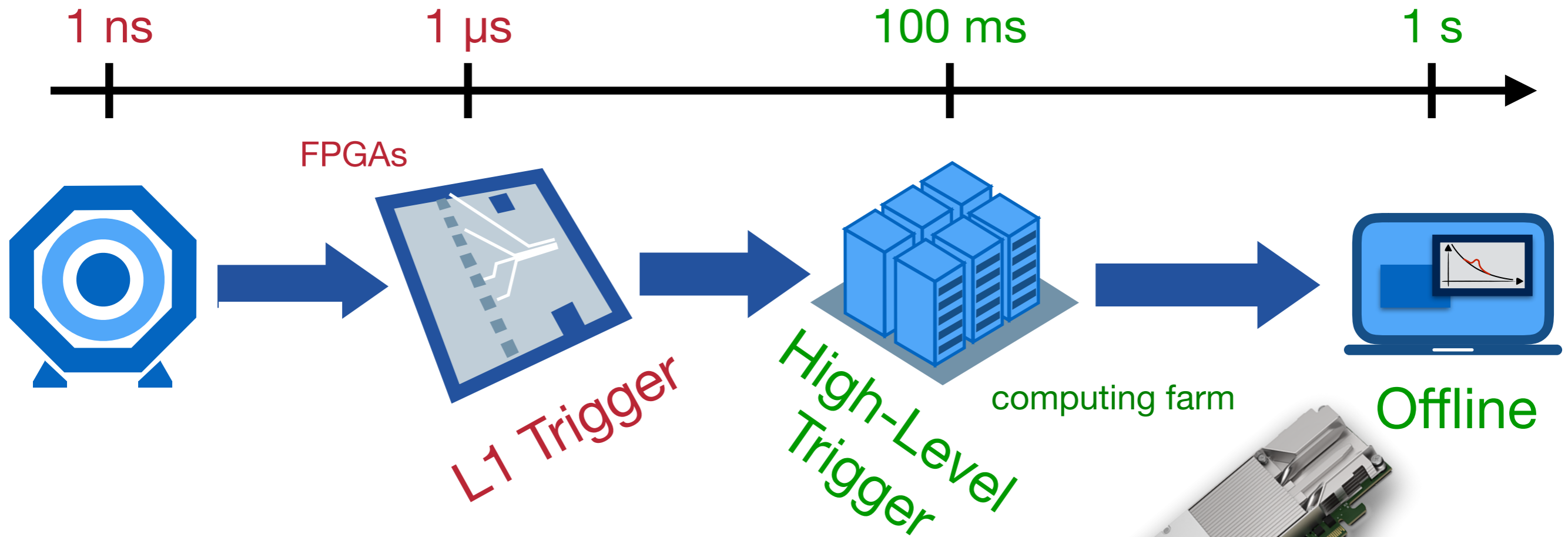


Focused on L1 trigger as first application
→ pure FPGAs



What can we do in $< \mu$ s on one FPGA?

The latency landscape @ LHC



Focused on L1 trigger as first application
→ pure FPGAs



What can we do in $< \mu$ s on one FPGA?



More time means
More resource reuse (x1000)
Bigger networks
→ acceleration with FPGAs

FPGA Co-Processor Acceleration Card

Offload a CPU from the computational heavy parts to a FPGA “accelerator”

- Increased computational speed of 10x-100x
- Reduced system size of 10x
- Reduced power consumption of 10x-100x

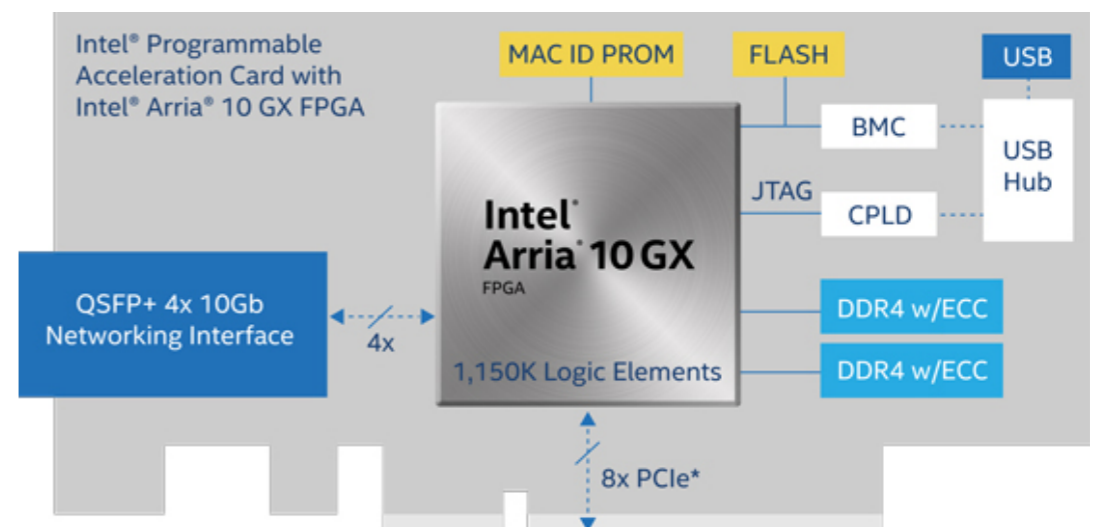
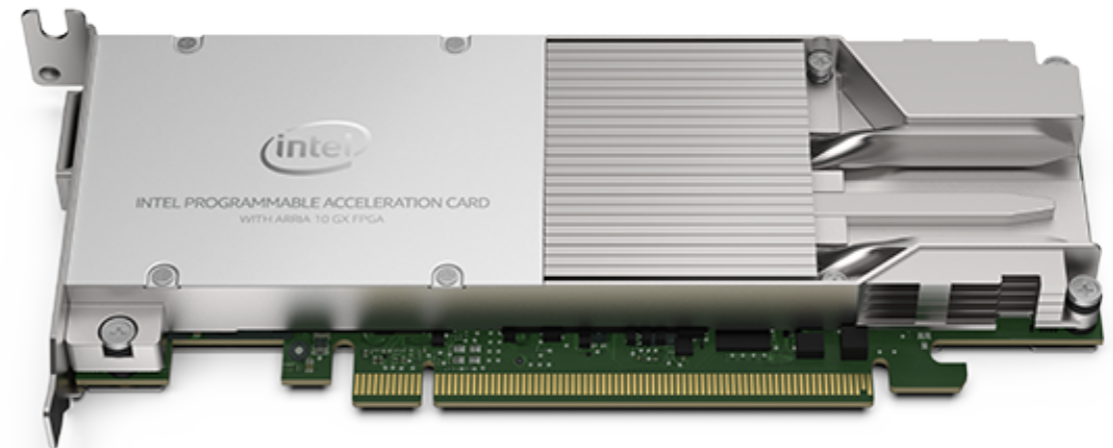
General use cases

- Autonomous vehicles
- Edge computing (phones)
- Big data analytics
- Medical applications (health monitoring, medical imaging)

Use case @ LHC

- FPGA accelerators on-site for HLT
- FPGA accelerators for offline computing resources (Cloud: Microsoft, Amazon, etc.)

Intel® Programmable Acceleration Card with Intel Arria® 10 GX FPGA





goes to the cloud



- **Amazon Web Service** provides cloud based system consisting of CPU/GPU/FPGAs
 - AWS F1 instances include up to 8 Xilinx Virtex Ultrascale+
- Used hls4ml through SDAccel to create a firmware implementation of 1D CNN
 - 5 layers with 10 four-channel inputs, latency of 116 ns
 - successfully run on an AWS F1 instance

New possibilities for HEP

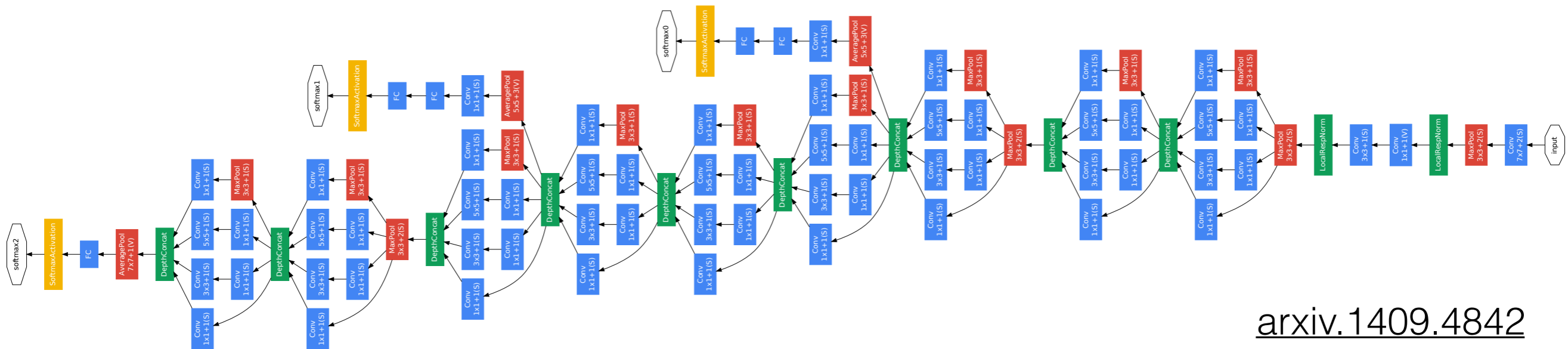
Convolutional neural networks

CNNs primarily developed for computer vision/image recognition

Designed to recognize visual patterns from pixel images with minimal preprocessing
Exploit strong spatially local correlation present in natural images
Share weights within a layer thanks to locality and translation invariance

ex, GoogLeNet (2014)

22 layers
5 million parameters



[arxiv.1409.4842](https://arxiv.org/abs/1409.4842)

New possibilities for HEP

Adaptation of GoogLeNet used offline by NOvA experiment:

Readout detector as a (multidimensional) image
Identify neutrino interactions based on their topology from “raw” data

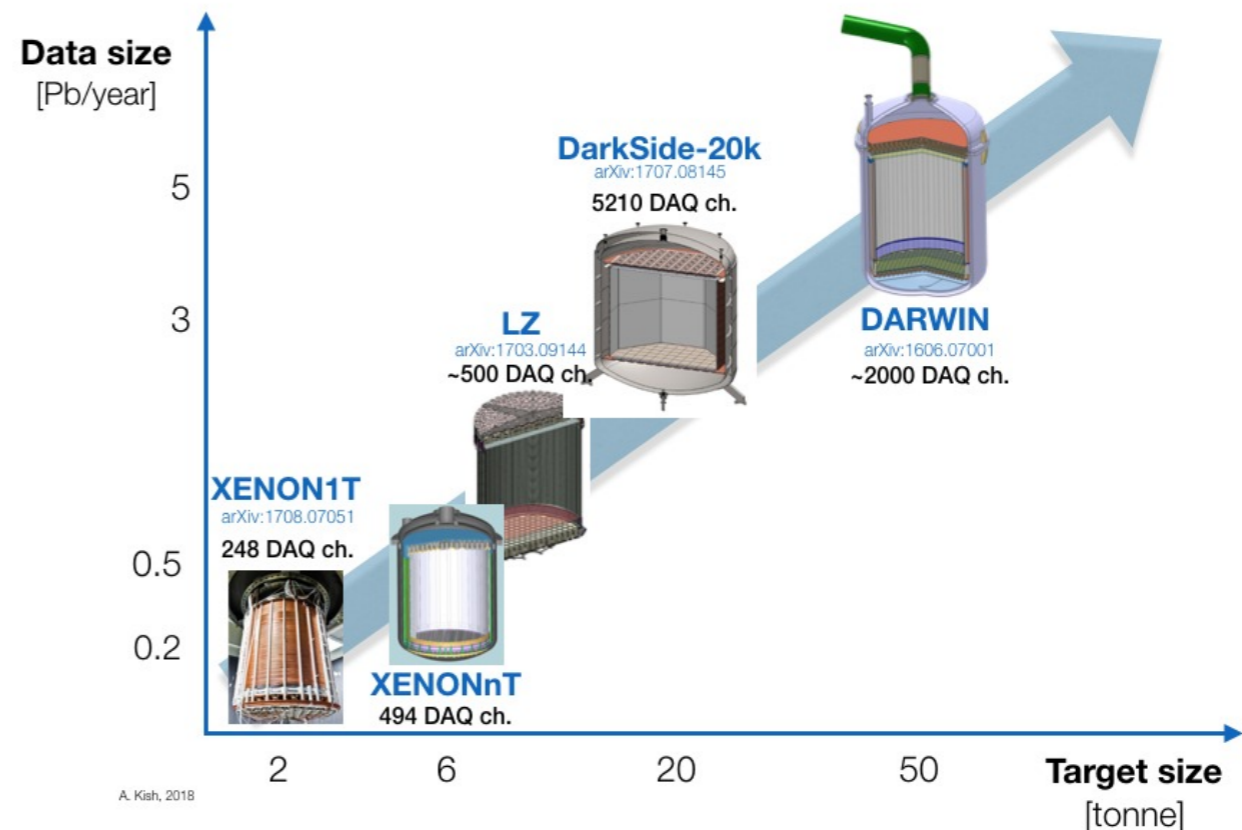
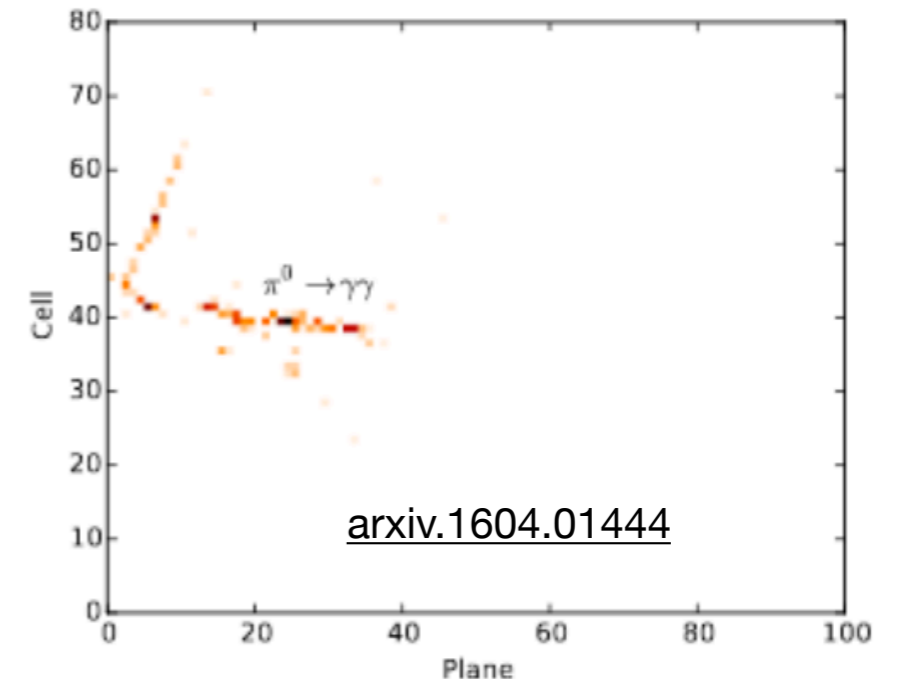
→ Can we fit this on a FPGA?

Benefits in large reduction of recorded data

Beyond LHC...

Noble gas TPCs for dark matter and neutrino physics increasing target size and facing increased data rates

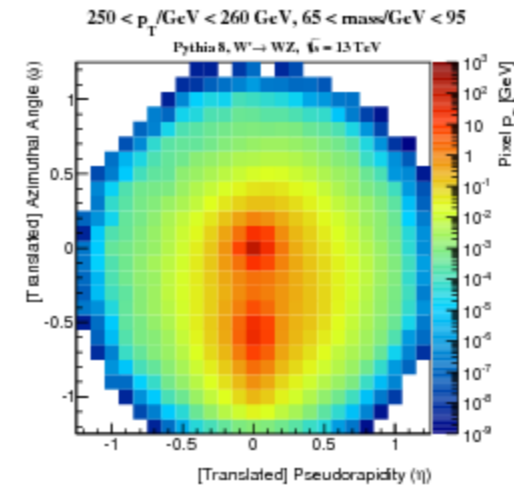
ex, CNN image input



hls4ml: other NN architectures

- **Convolutional Neural Networks**

- active implementation of small Conv1D and Conv2D with hls4ml
- resources reuse and compression supported
- work is ongoing to ensure large scale networks

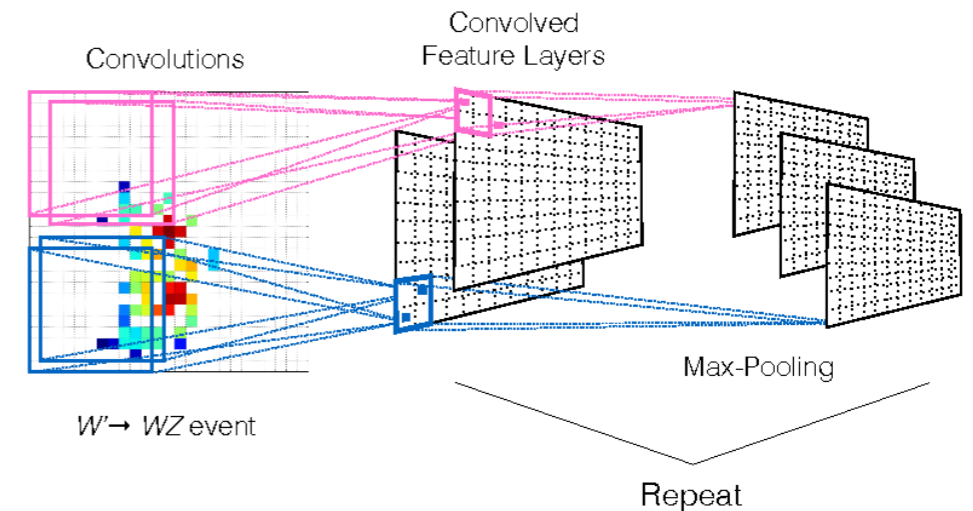


Use case of CNN
 @ LHC for jet
 image

[JHEP 07 \(2016\) 069](#)

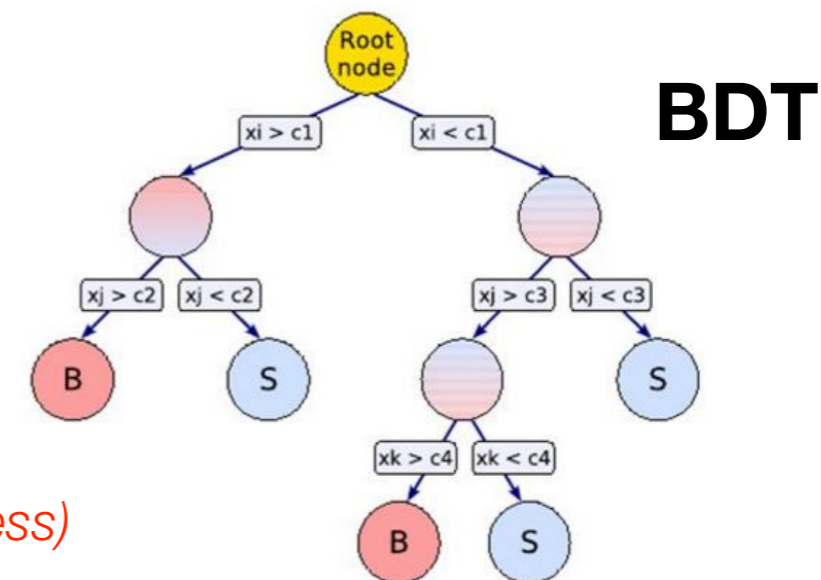
- **Boosted Decision Tree** (*work in progress*)

- each node in decision tree compares element against a threshold \rightarrow boolean logic, thresholds in LUT, suitable for FPGA
- each tree is independent \rightarrow high parallelization



- **Binary/Ternary Neural Networks** (*work in progress*)

- weights are binary/ternary in the inference = $\pm 1, 0$
- ternary NN does not need pruning/compression
- similar performance and latency with 0% DSPs used



- **Recurrent NN and LSTM** under testing (*work in progress*)

Summary



We introduced a new software/firmware package **hls4ml**

Automated translation of everyday machine learning inference into firmware in ~ minutes

Tunable configuration for optimization of your use case

First application is single FPGA, <1 us latency for L1 trigger or DAQ

Explore also applications for acceleration with CPU-FPGA co-processors for long latency trigger tasks

For more info

<https://hls-fpga-machine-learning.github.io/hls4ml/>

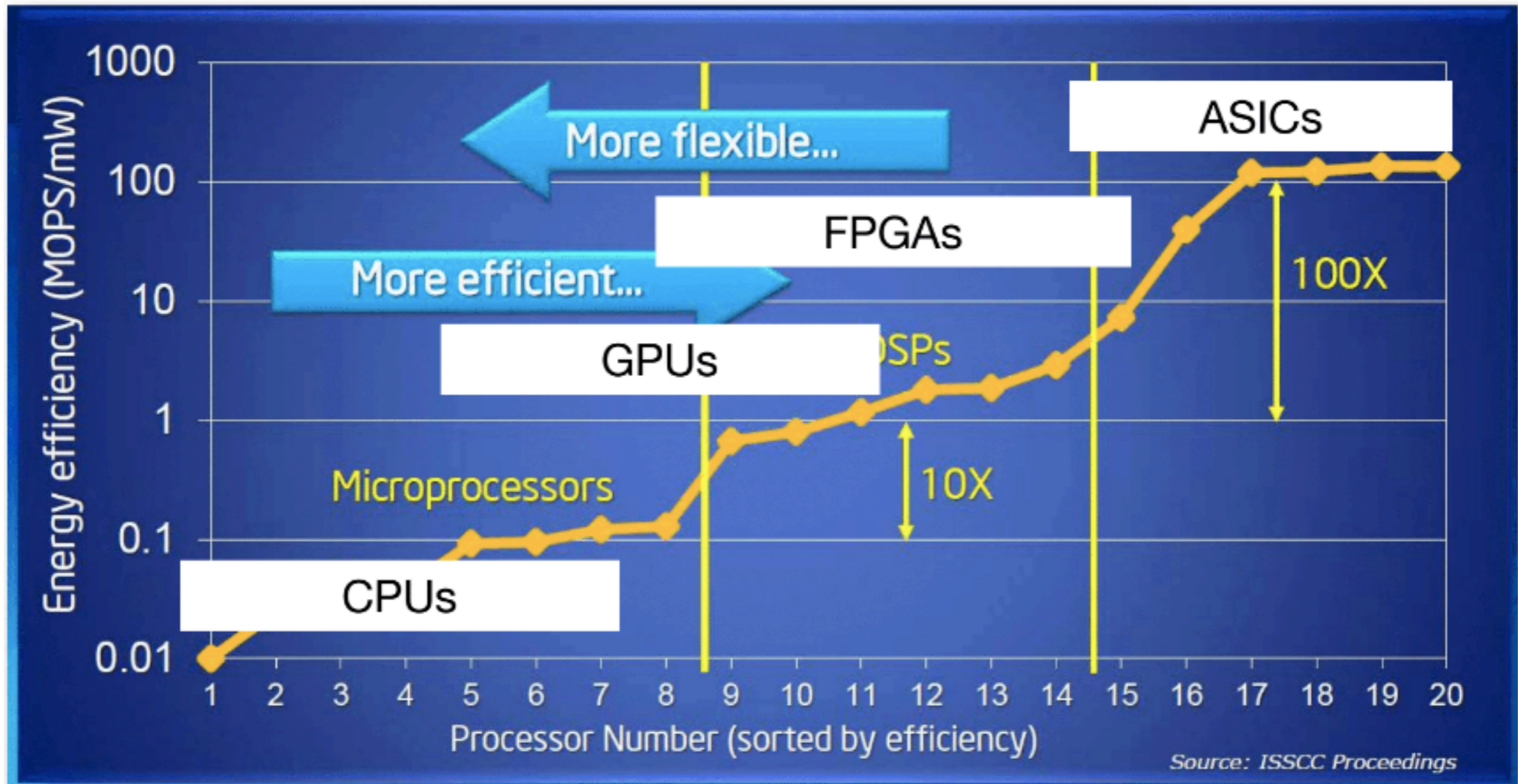
<https://arxiv.org/abs/1804.06913>



Backup

CPU, GPU, FPGA, and ASIC

FPGAs are the middle ground of latency, energy efficiency and flexibility



Source: Bob Broderson, Berkeley Wireless group