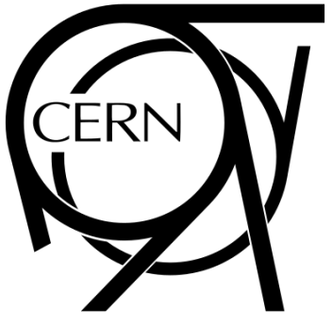
Real-time Deep Learning on FPGAs for L1 Trigger and Data Acquisition

CERN Openlab Workshop, January 23, 2019

Javier Duarte, Sergo Jindariani, Ben Kreis, Ryan Rivera, Nhan Tran (Fermilab)
Jennifer Ngadiuba, Maurizio Pierini, Vladimir Loncar (CERN)
Edward Kreinar (Hawkeye 360)
Phil Harris, Song Han, Dylan Rankin (MIT)
Zhenbin Wu (University of Illinois at Chicago)
Sioni Summer (Imperial College)



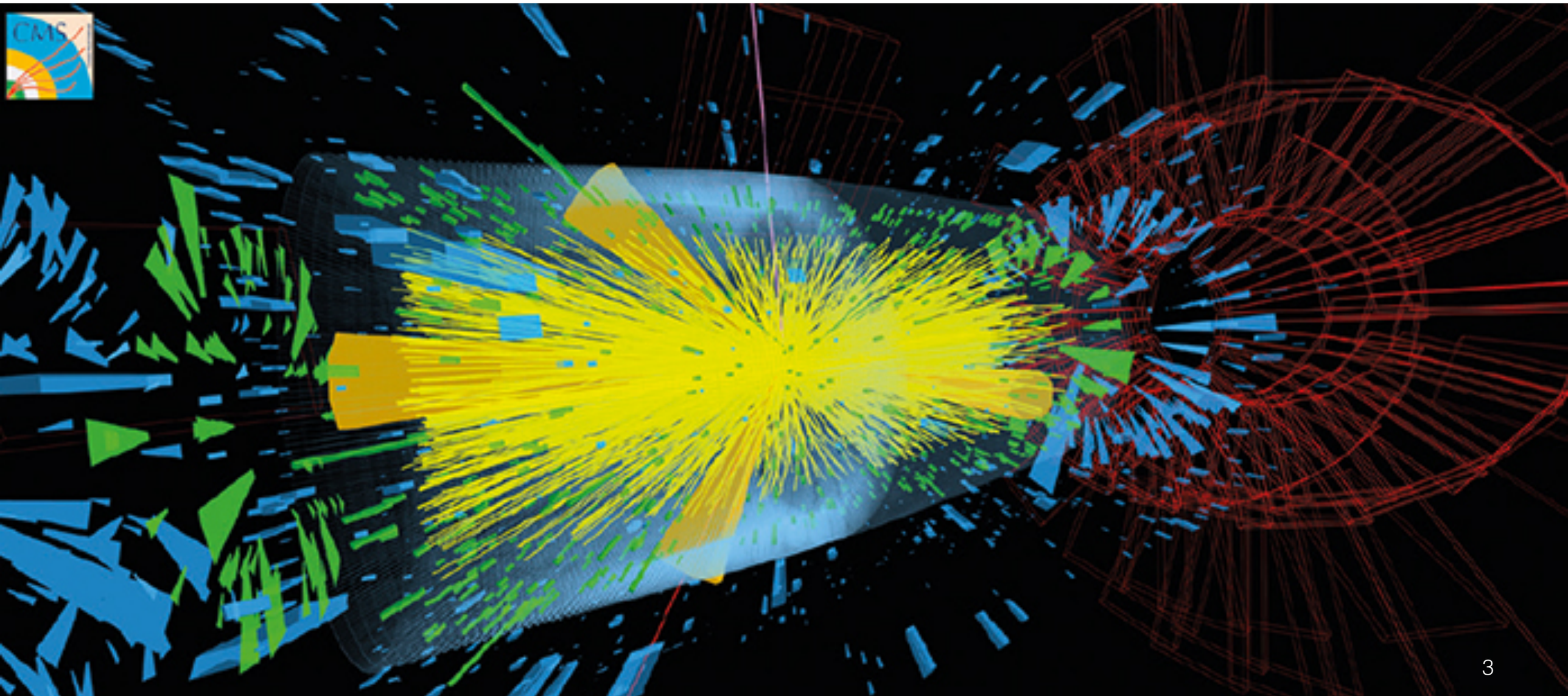
Motivation

The challenge: triggering at (HL-)LHC

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



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: <PU> ~ 20-50

2017 + Run 3: <PU> ~ 50-80

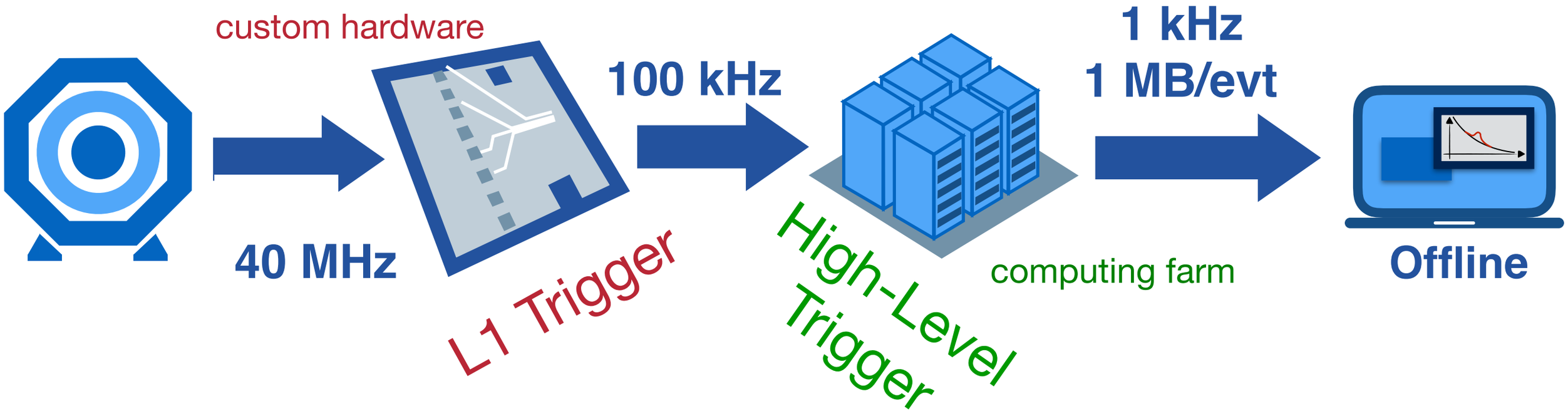
HL-LHC: 140-200

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(μs)

Latencies require all-FPGA design

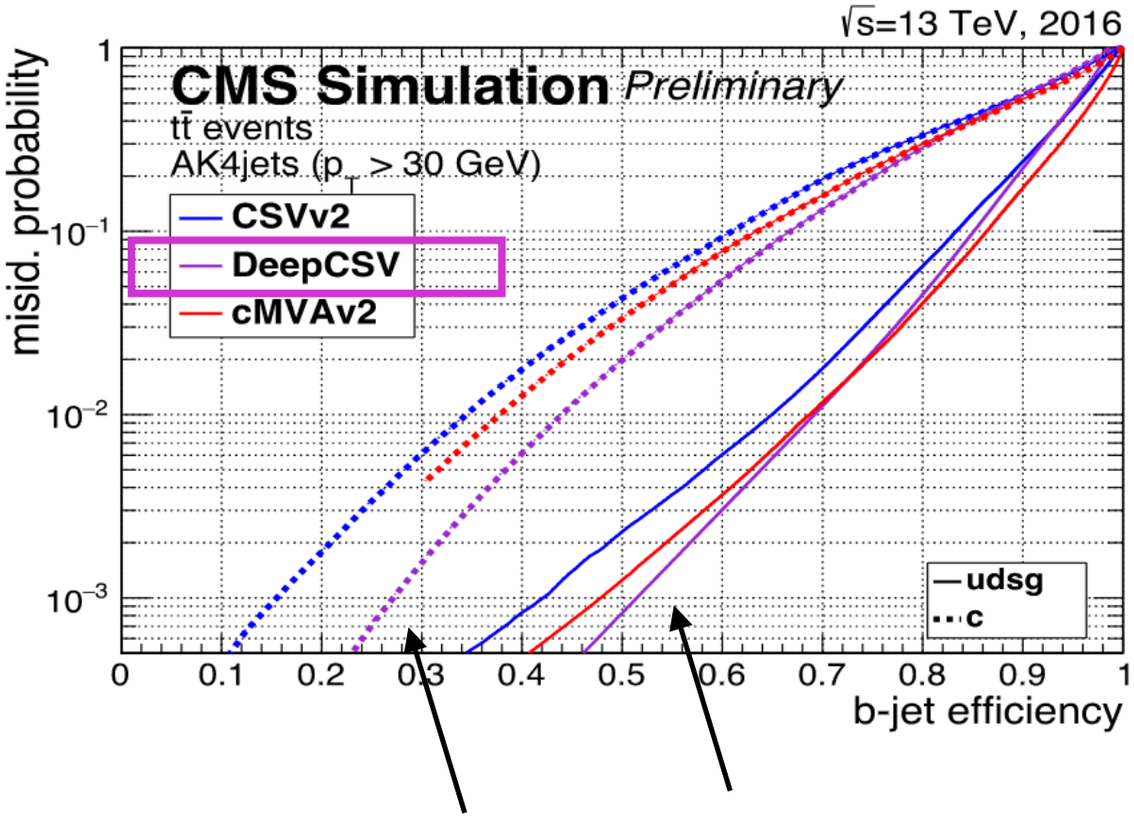
Computing farm for detailed analysis of the full event

Latency O(100 ms)

For HL-LHC upgrade: latency and output rates will increase by ~ 3 (ex: for CMS 3.8 → 12.5 μs @ L1)

The latency landscape @ LHC

ex, identification of b-quark jets

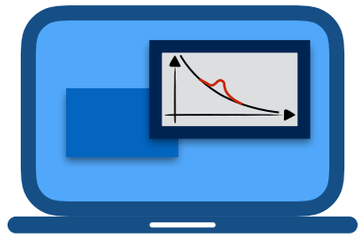


Deep neural network based on high-level features

both offline and @ HLT

High-Level Trigger

computing farm



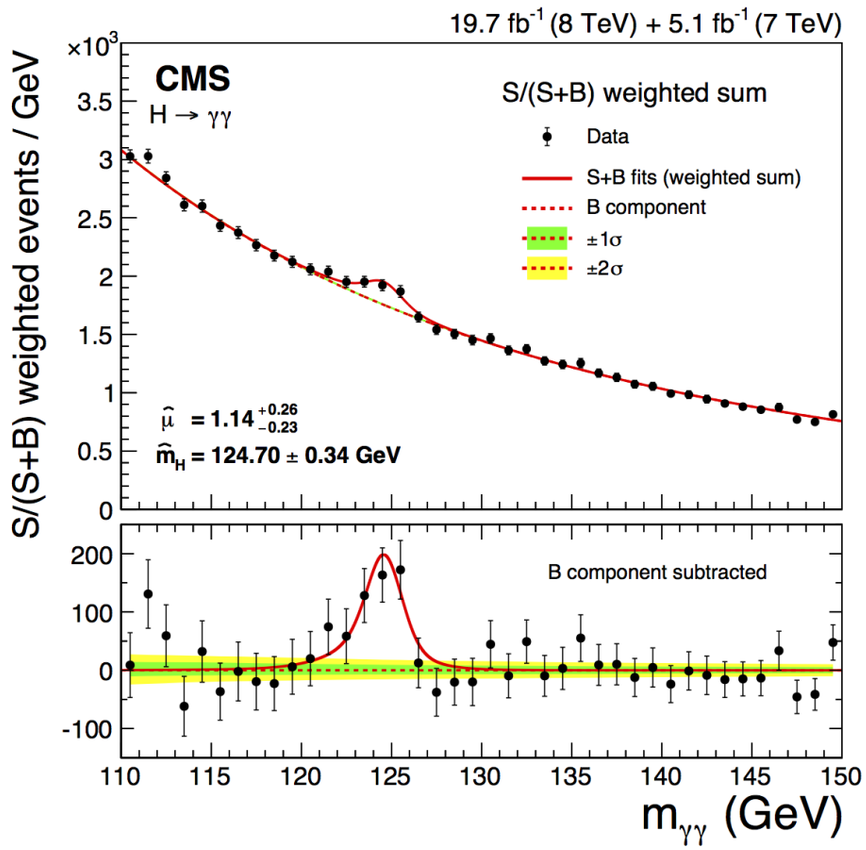
Offline

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, ….

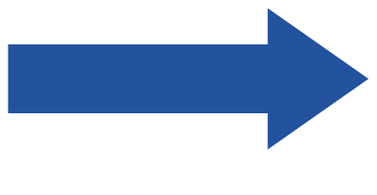
The latency landscape @ LHC

ex, Higgs discovery



High-Level Trigger

computing farm



Offline

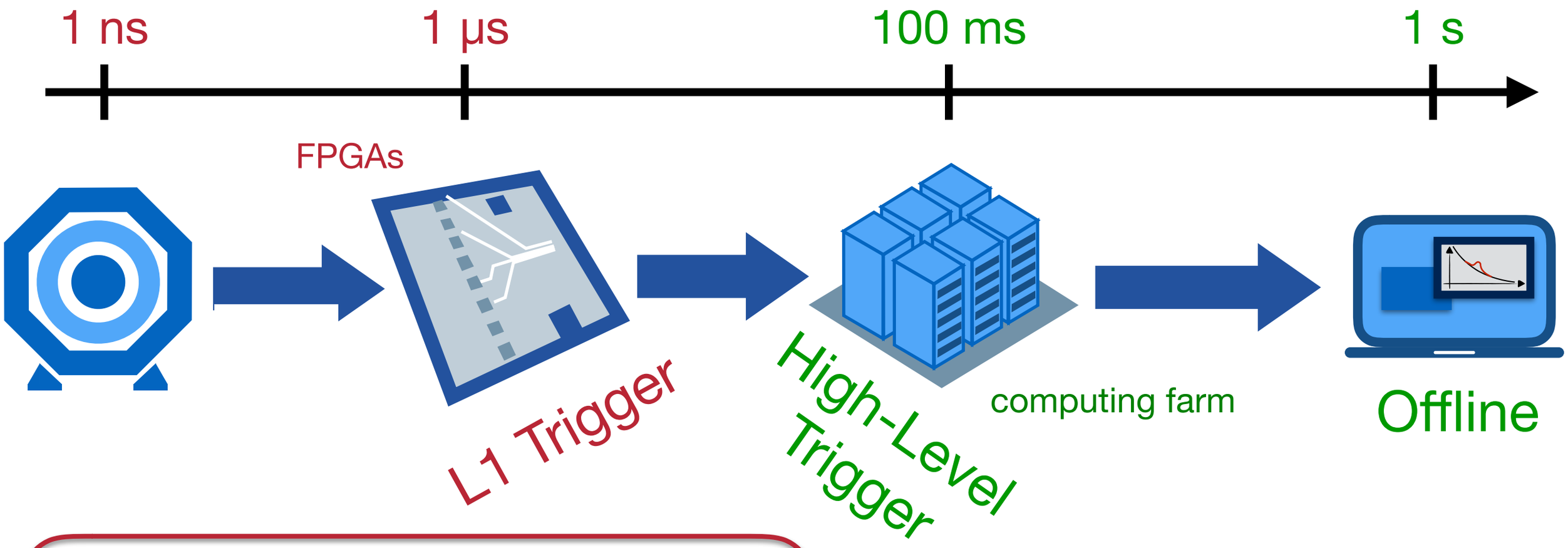
ML algorithms used offline for

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

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



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

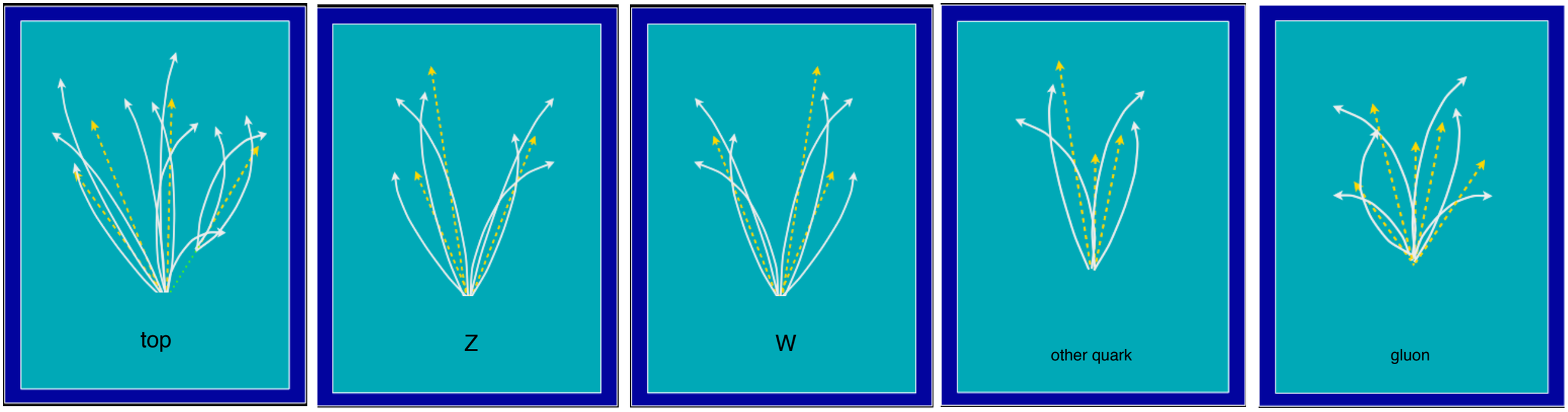
What can we do in ~ µs on one FPGA?

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, ….

Physics case: jet tagging

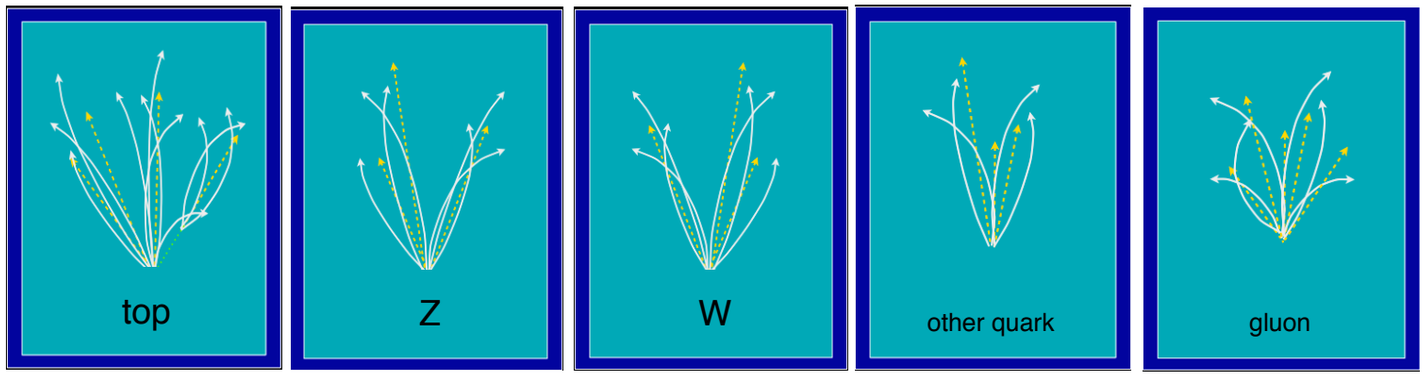
Study a **multi-classification task to be implemented on FPGA:** discrimination between highly energetic (boosted) ***q, g, W, Z, t*** initiated jets

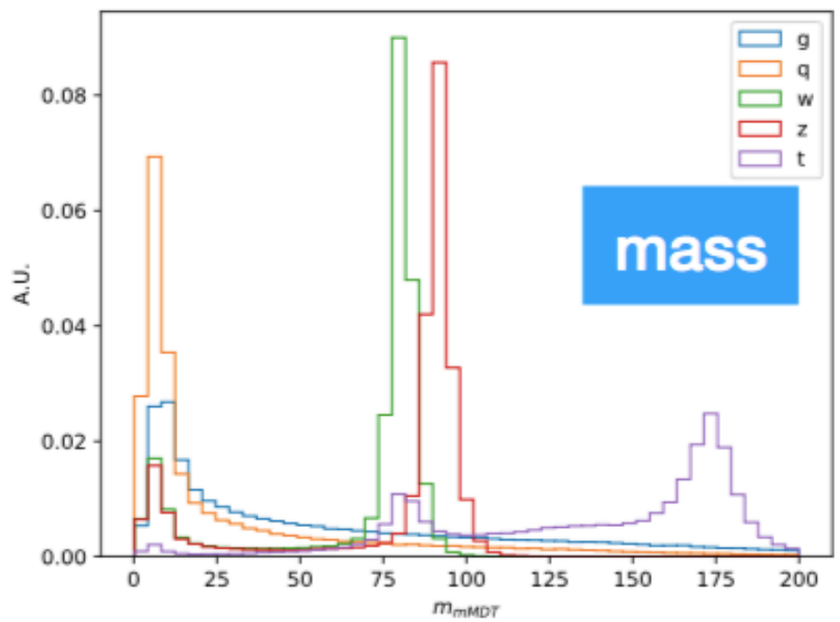


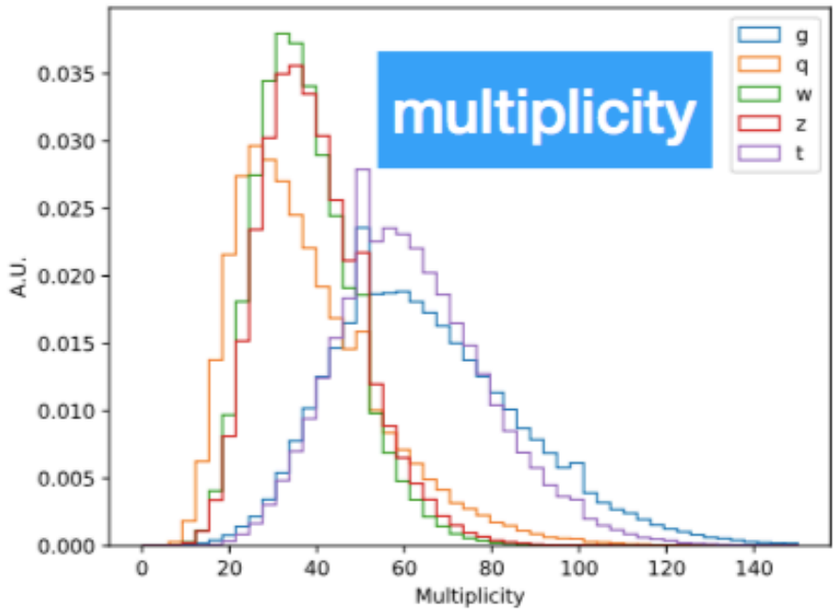
t→bW→bqq	**Z→qq**	**W→qq**	**q/g background**
3-prong jet	2-prong jet	2-prong jet	no substructure and/or mass ~ 0

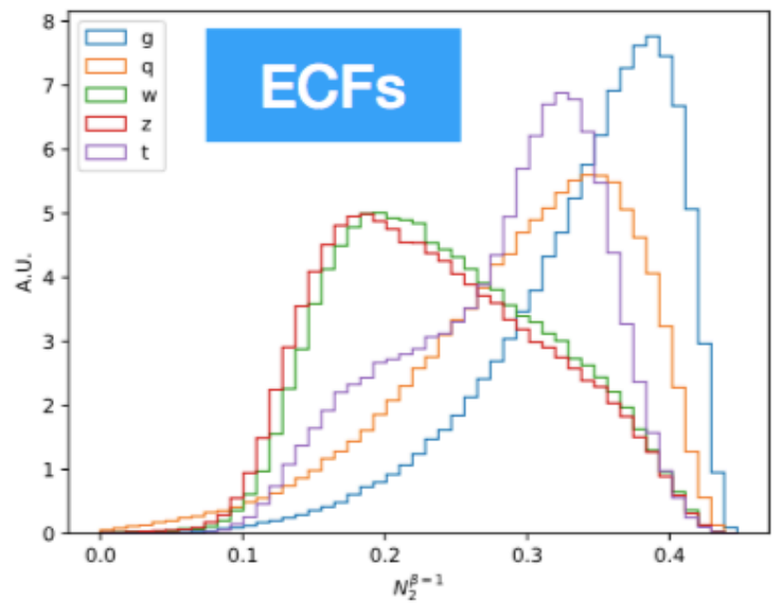
Reconstructed as one massive jet with substructure

Physics case: jet tagging









Input variables: several obervables known to have high discrimination power from offline data analyses and published studies [*]

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

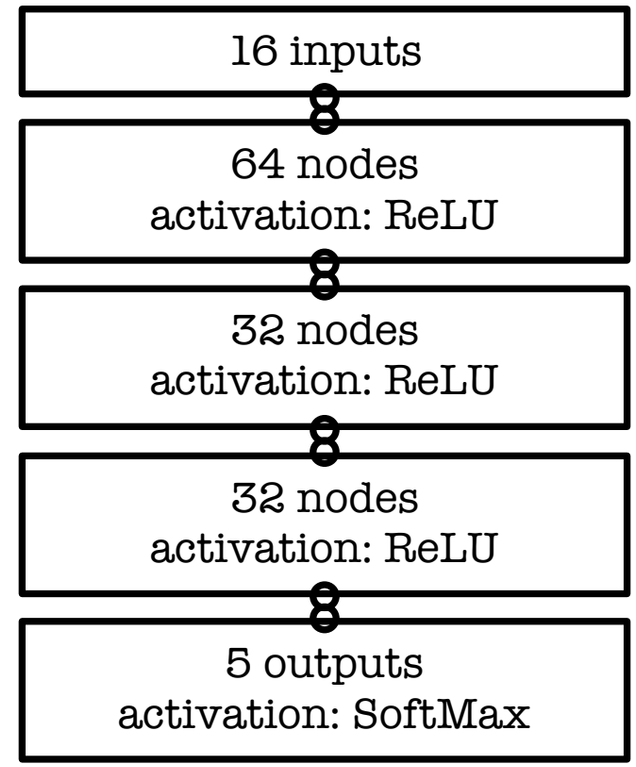
Physics case: jet tagging

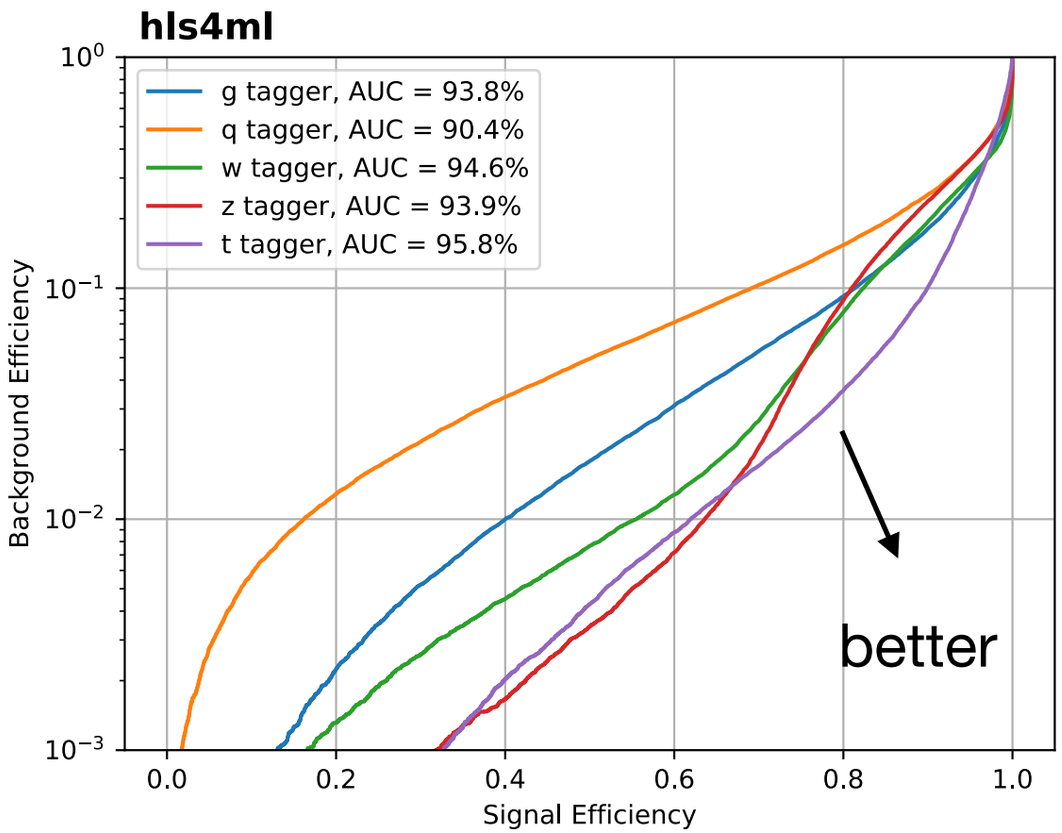
- We train (on GPU) the **five output multi-classifier** on a sample of ~ 1M events with two boosted WW/ZZ/tt/qq/gg anti-k_T jets

- Fully connected neural network with **16 expert-level inputs**:

  - Relu activation function for intermediate layers

  - Softmax activation function for output layer



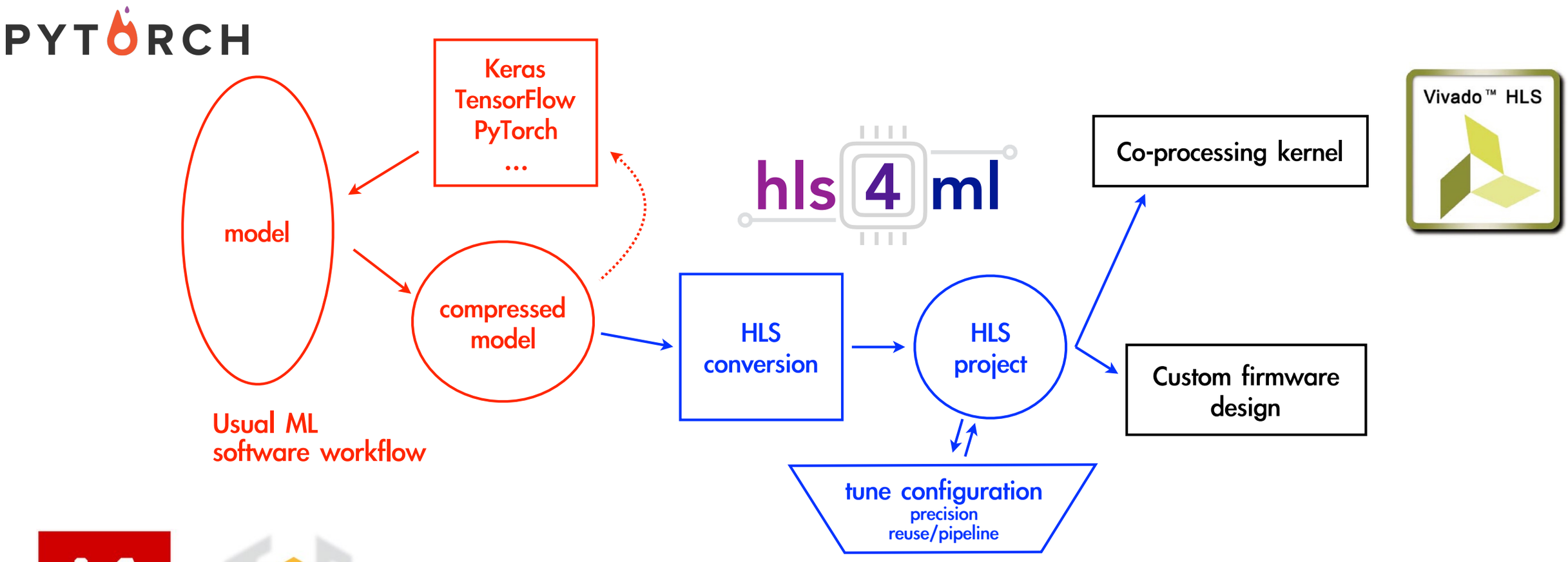


AUC = area under ROC curve (100% is perfect, 20% is random)

high level synthesis for machine learning

Implemented an user-friendly and automatic tool to develop and optimize FPGA firmware design for DL inference:

- reads as input models trained with standard DL libraries
- uses Xilinx HLS software (accessible to non-expert, engineers resource not common in HEP)
- comes with implementation of common ingredients (layers, activation functions, binary NN …)



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

https://arxiv.org/abs/1804.06913

Efficient NN design for FPGAs

FPGAs provide huge flexibility

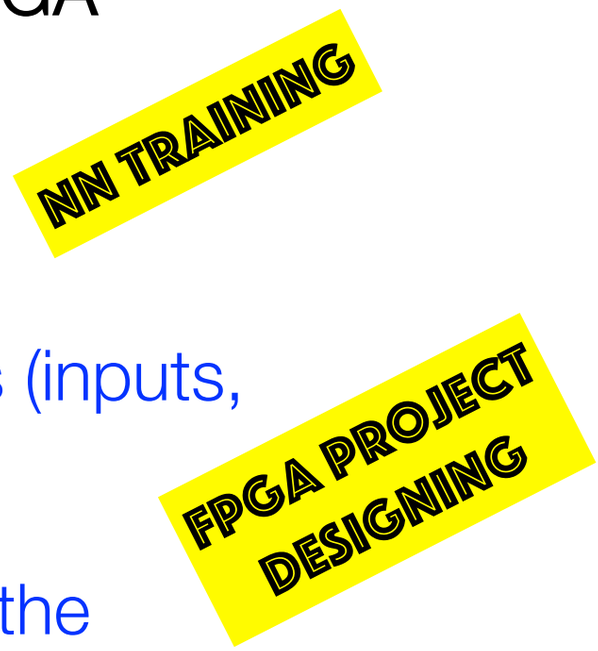
Performance depends on how well you take advantage of this

Constraints:

- Input bandwidth
- FPGA resources
- Latency

With `hls4ml` package we have studied/optimized the FPGA design through:

- **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



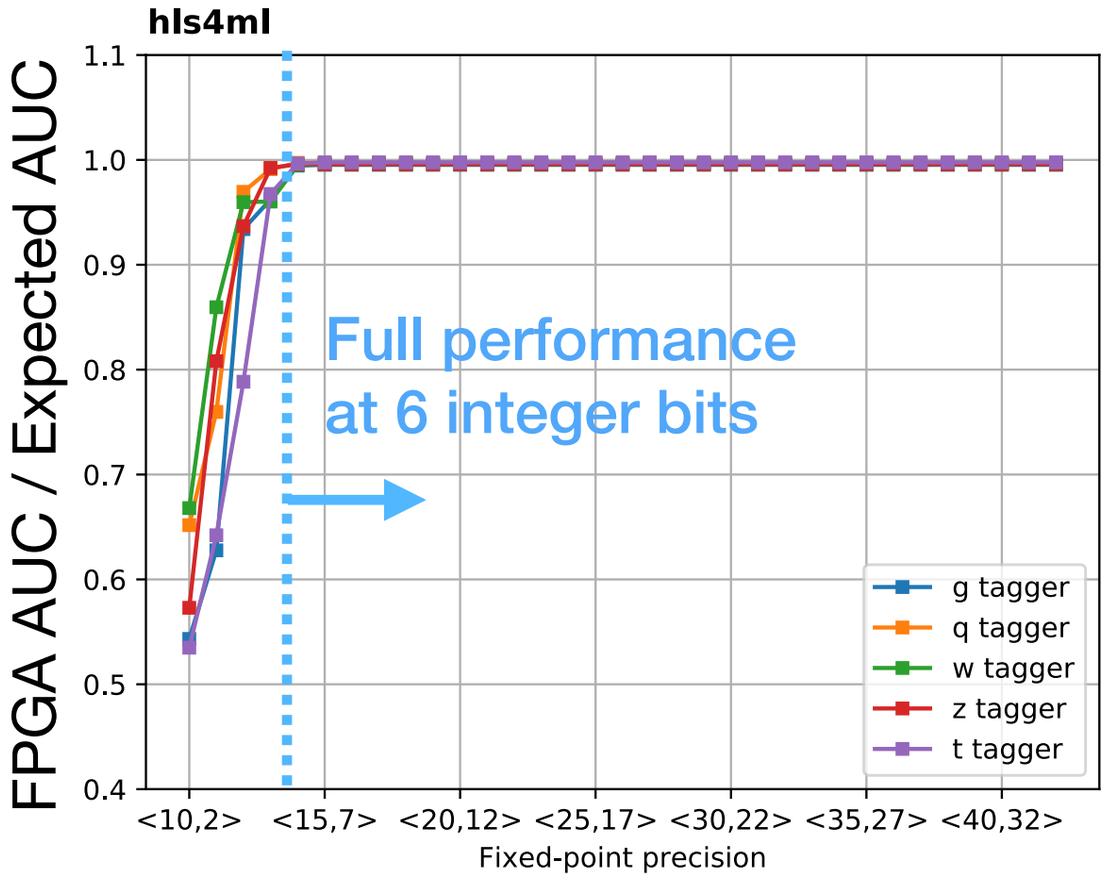
Efficient NN design: quantization

ap_fixed<width,integer>

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

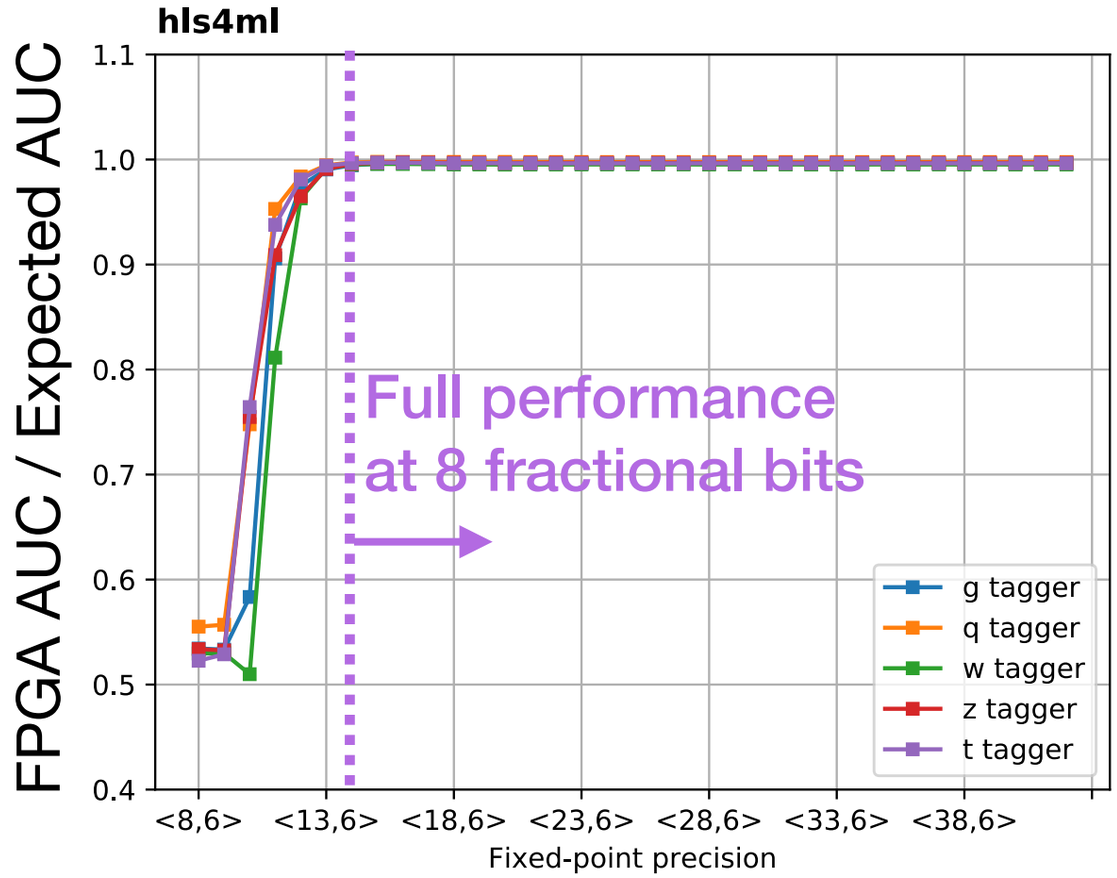
Scan integer bits

Fractional bits fixed to 8

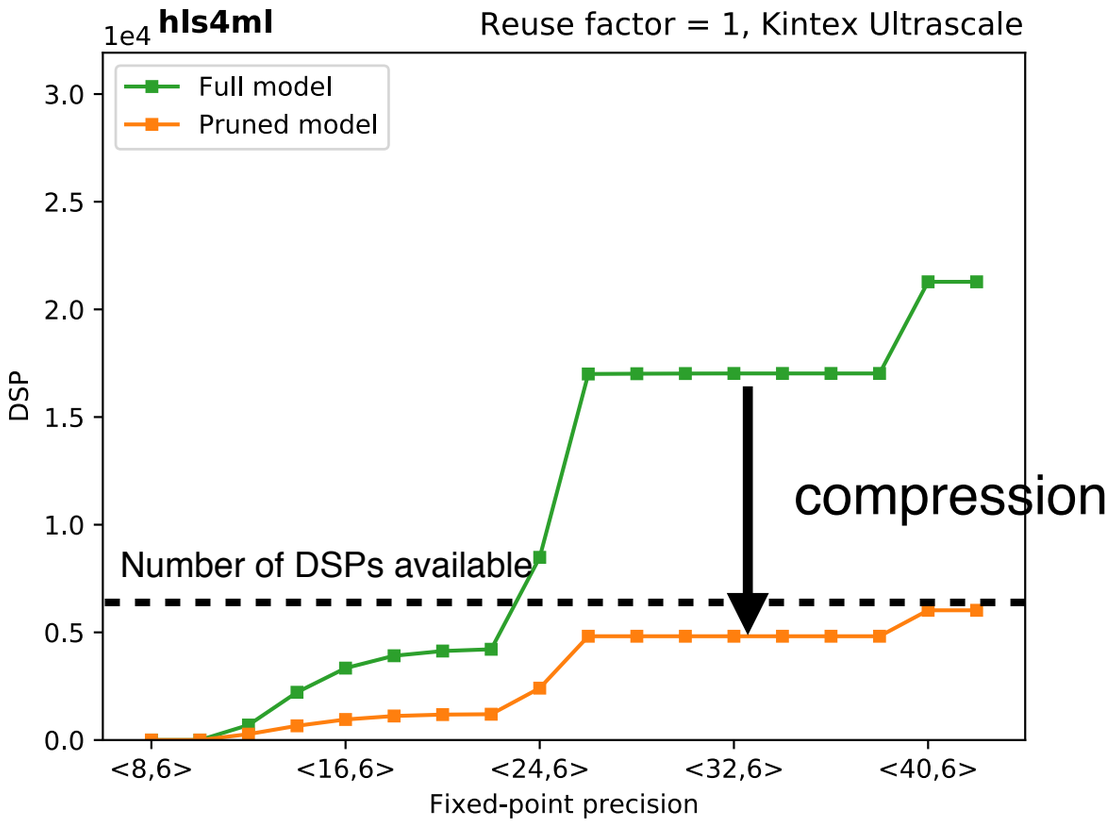


Scan fractional bits

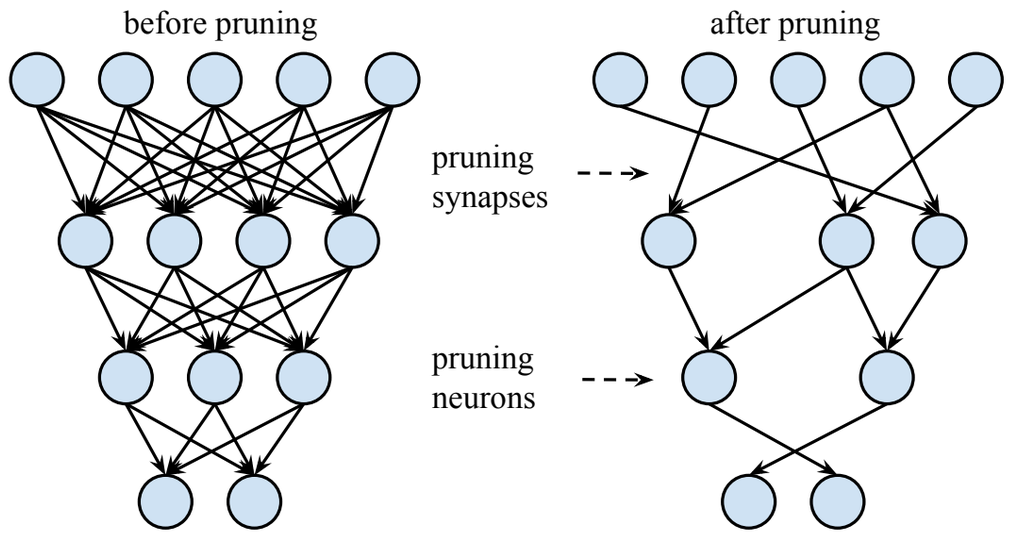
Integer bits fixed to 6



Efficient NN design: compression

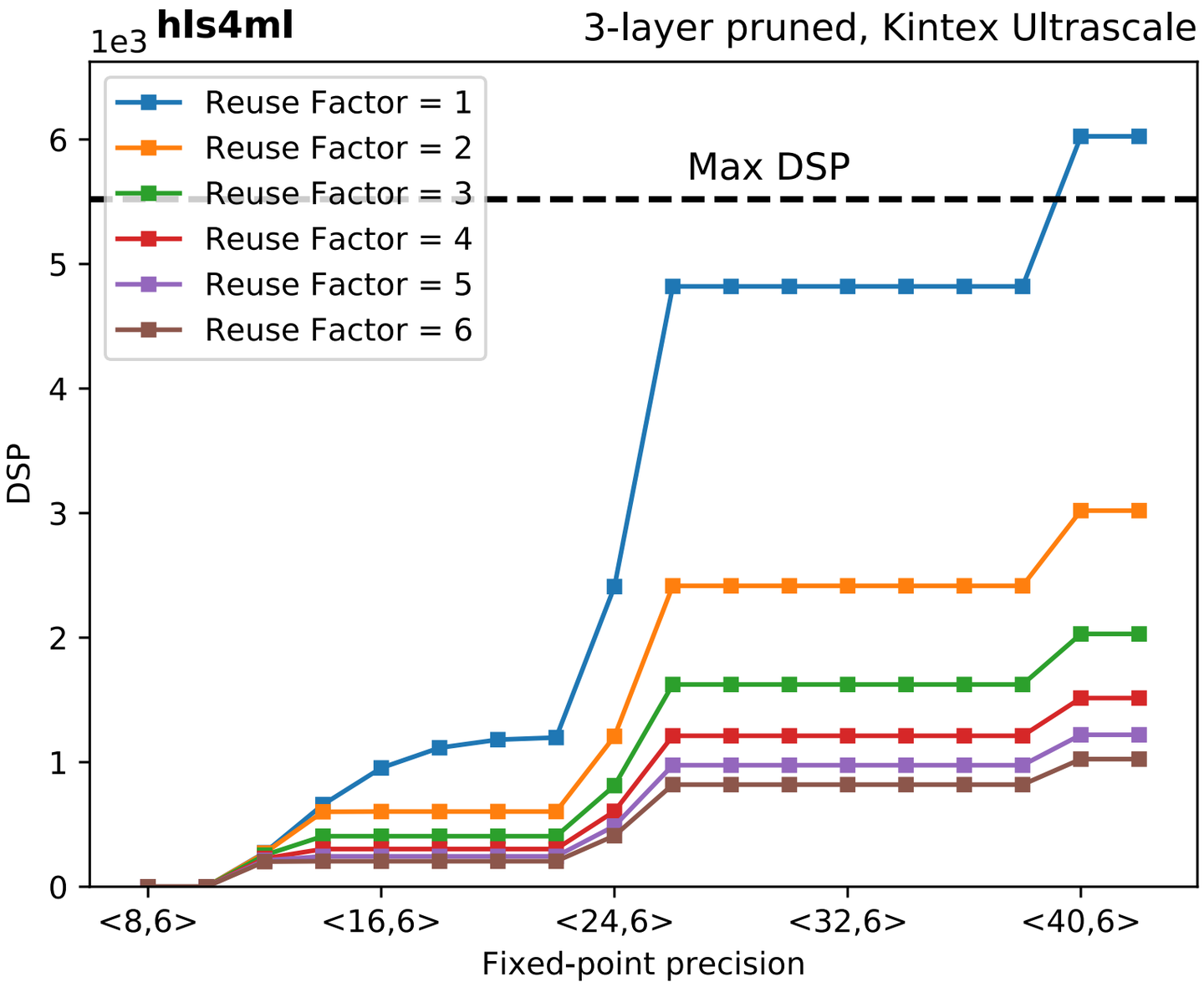


70% compression ~ 70% fewer DSPs



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

Parallelization: DSPs usage



Fully parallel
Each mult. used 1x

Each mult. used 2x

Each mult. used 3x

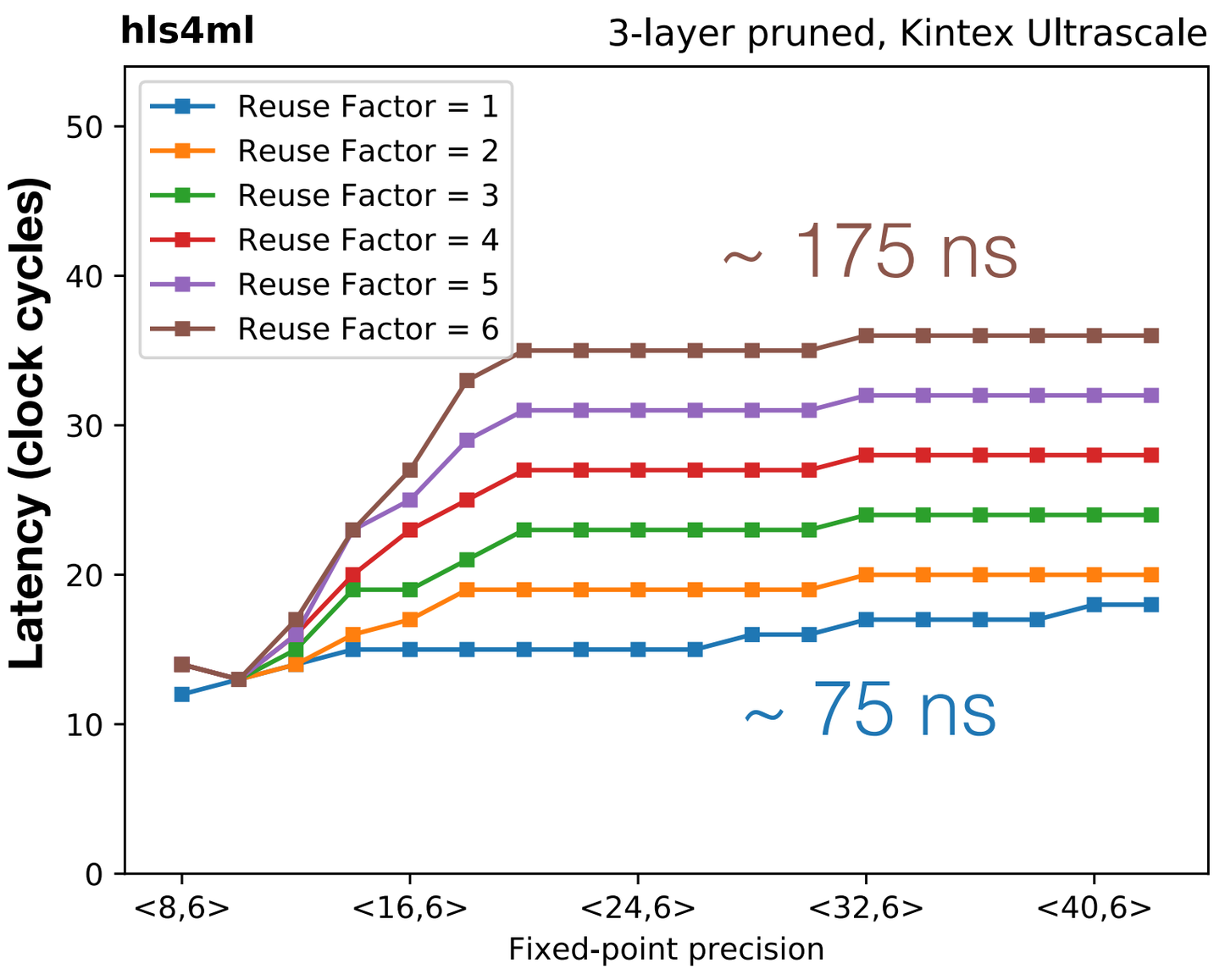
⋮

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

Parallelization: Timing

Latency of layer m

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



Longer latency

Each mult. used 6x

Each mult. used 3x

Fully parallel
Each mult. used 1x

More resources

The hls4ml package

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

Inputs

Keras

HDF5

```
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>
```

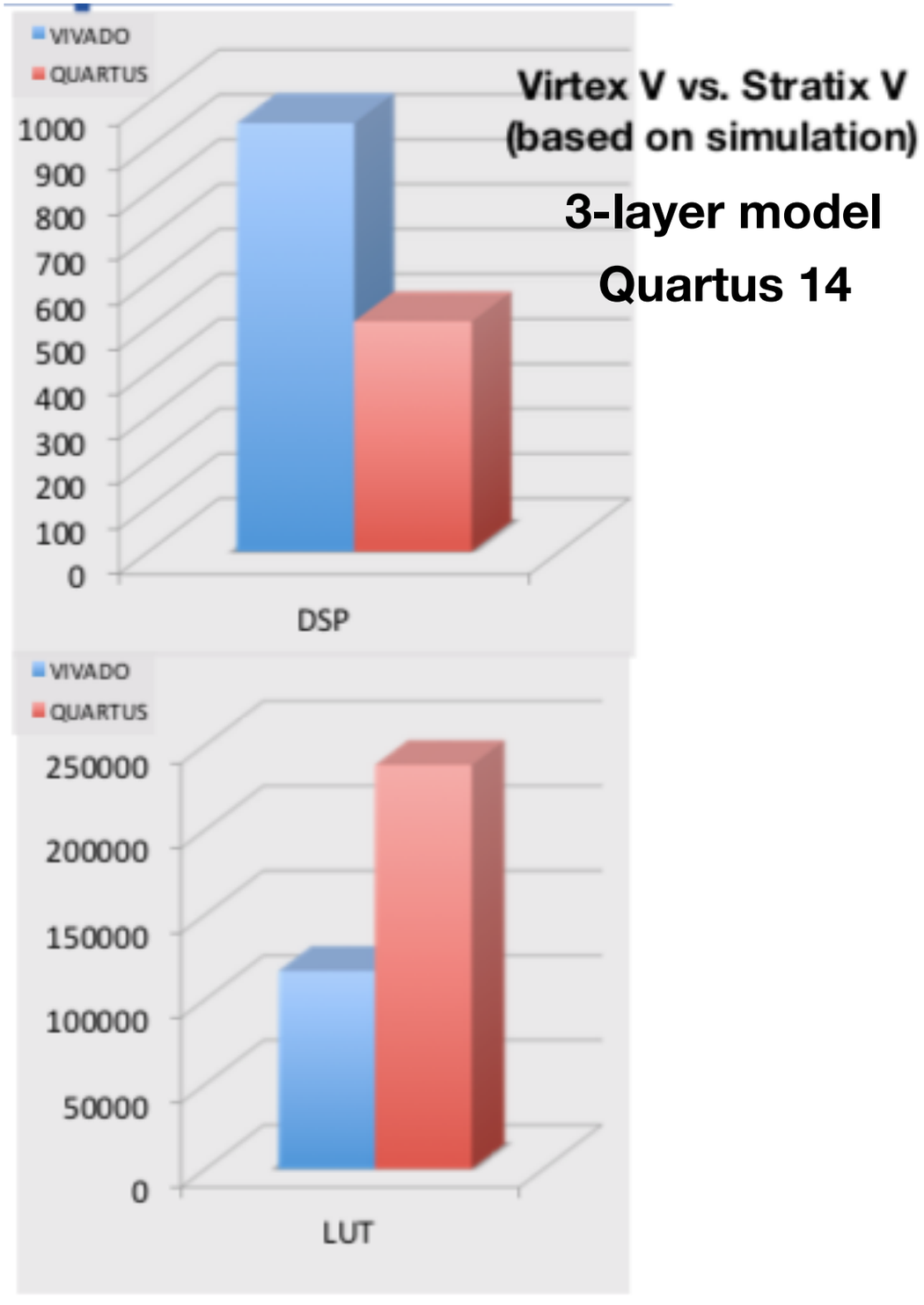
Config

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

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

Extend hls4ml with Intel support

- `hls4ml` supports Xilinx FPGAs and software from beginning
- Currently working to extend the package to work with Intel/Altera Quartus HLS
  - work in progress: technical complications slowed us down (software licenses and installation, Quartus HLS version @ CERN, …)
- First results encouraging (based on emulation and to be confirmed with actual deployment on card)

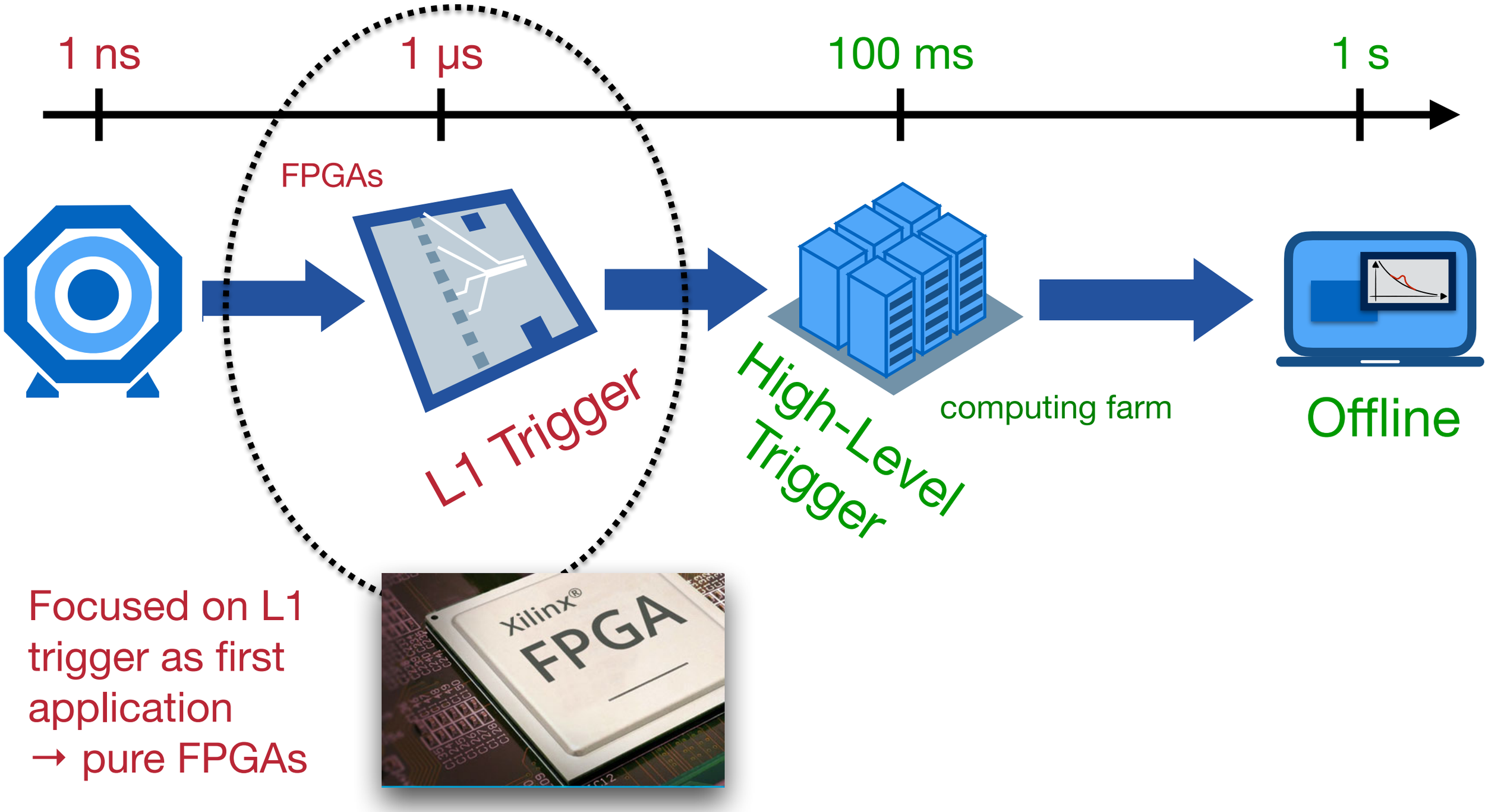


Virtex V vs. Stratix V (based on simulation)

3-layer model

Quartus 14

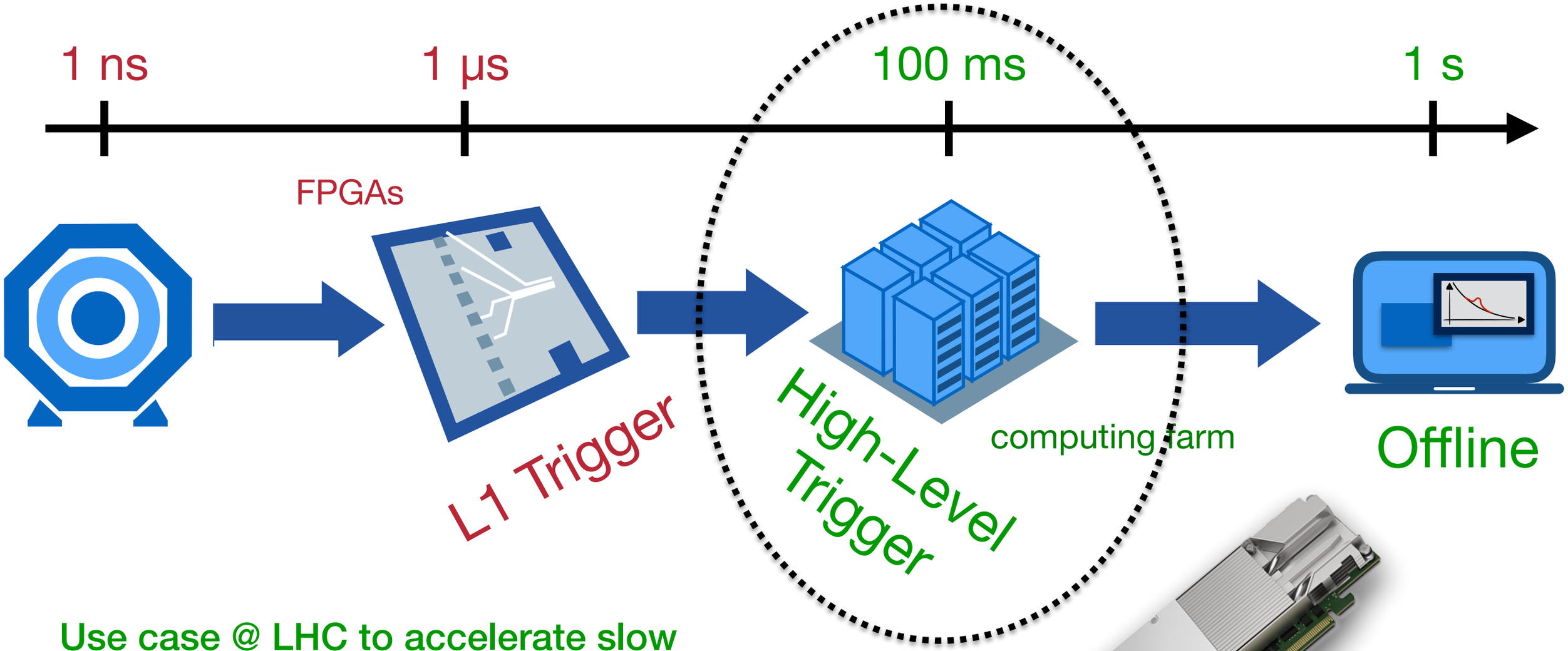
The latency landscape @ LHC



Focused on L1 trigger as first application
→ pure FPGAs

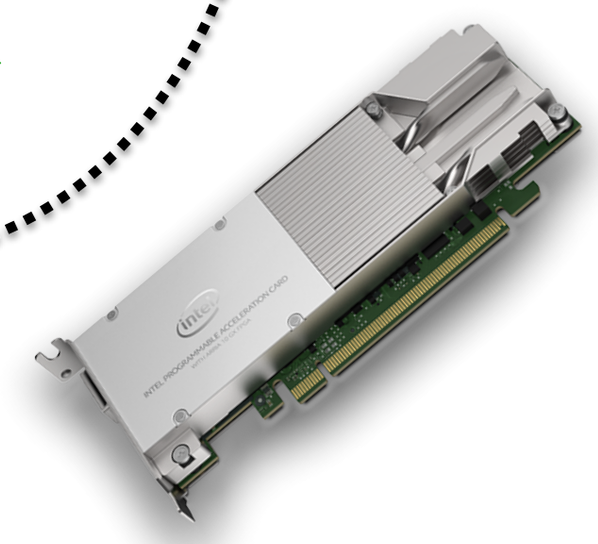
What can we do in < us on one FPGA?

The latency landscape @ LHC



Use case @ LHC to accelerate slow algorithms (ex: tracking) and ML inference for HLT and offline analysis

Ongoing R&D on heterogeneous computing on-site (@CERN) and on commercial clouds (Microsoft Brainwave, Amazon Web Services, Google TPU cloud)



→ CPU+FPGA co-processor

goes to the cloud

amazon web services™

SDAccel™ Environment

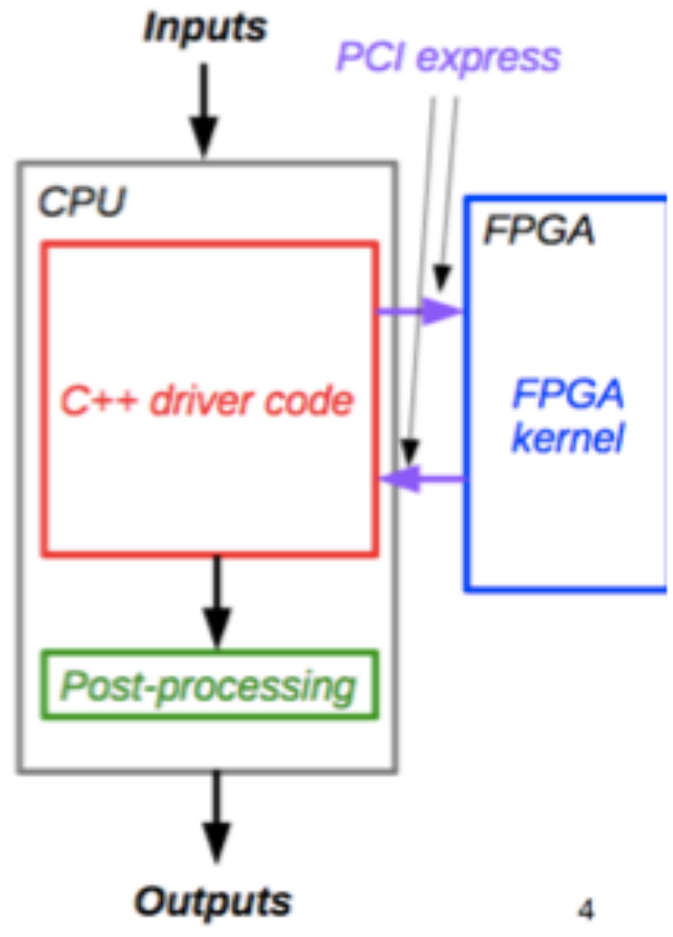
- **Amazon Web Service** provides co-processor CPU+FPGA systems with Xilinx Virtex Ultrascale+ VU9P

- Xilinx SDAccel development environment allows the development/running of connected FPGA kernels and CPU processes

  - any FPGA application defined in HLS, OpenCL, or VHDL/Verilog can be accelerated

- `hls4ml` project only needs to be wrapped to provide specific I/O ports configuration for SDAccel to interface properly

- **Succesfully accelerated 1D CNN example project on AWS F1:** 10 four-channel inputs , 3 convolutional layers, 2 dense layers, 5 outputs → *latency 116 ns*



DL acceleration on Intel FPGAs @ CERN

- `hls4ml` developed from start to target very low latency

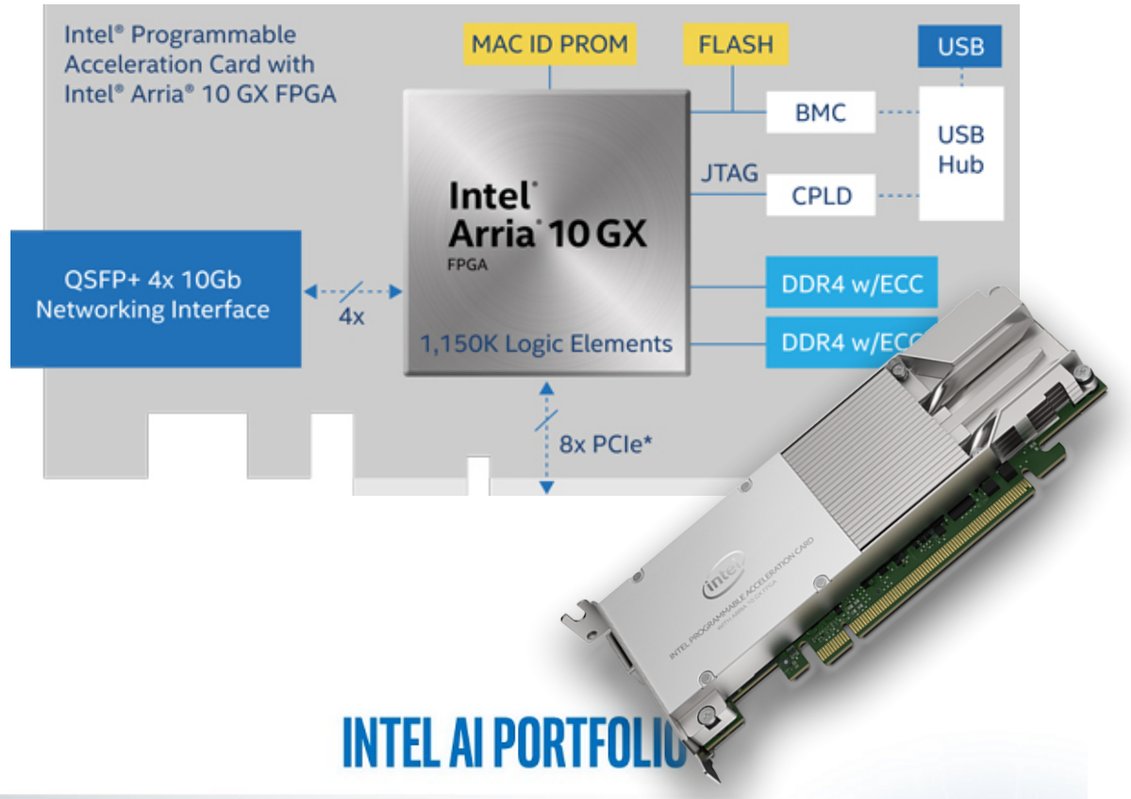
- Latencies at HLT less strict allowing inference of much bigger networks
  → different firmware project design wrt L1 trigger application

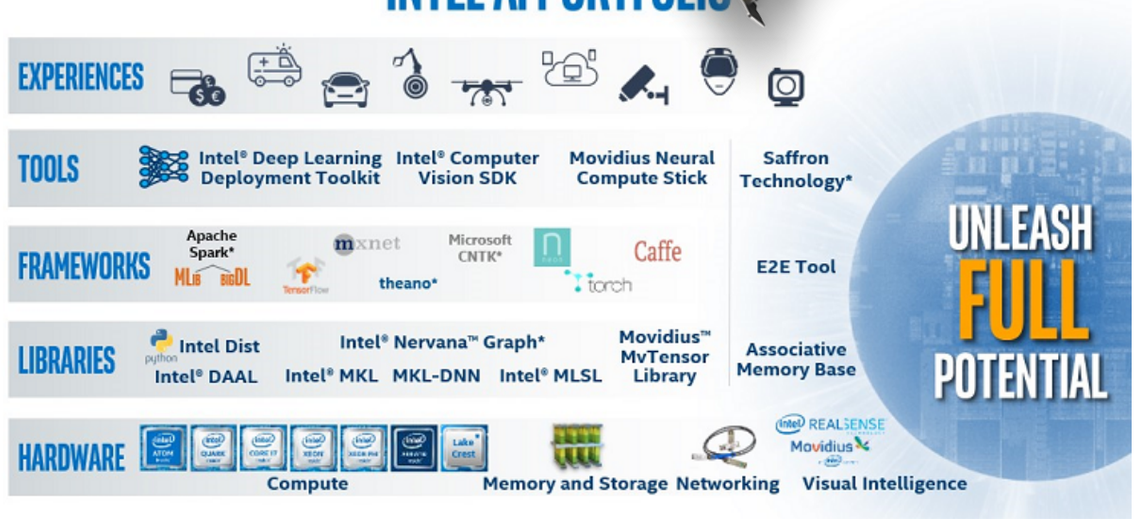
- No need to reinvent the wheel…

- **Work in progress:** accelerate DL inference of predefined big networks through Intel softwares such as **OpenVino** to benchmark a specific physics case

  - perform studies on-site with Intel Arria 10 GX FPGA available at CERN

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





Summary

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

Supports Xilinx HLS but will be extended for Intel support with Quartus HLS

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