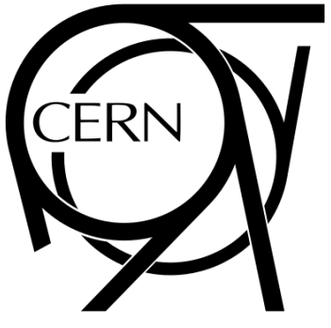
Real-time Deep Learning on FPGAs for L1 Trigger and Data Acquisition

CERN-Schneider Meeting

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



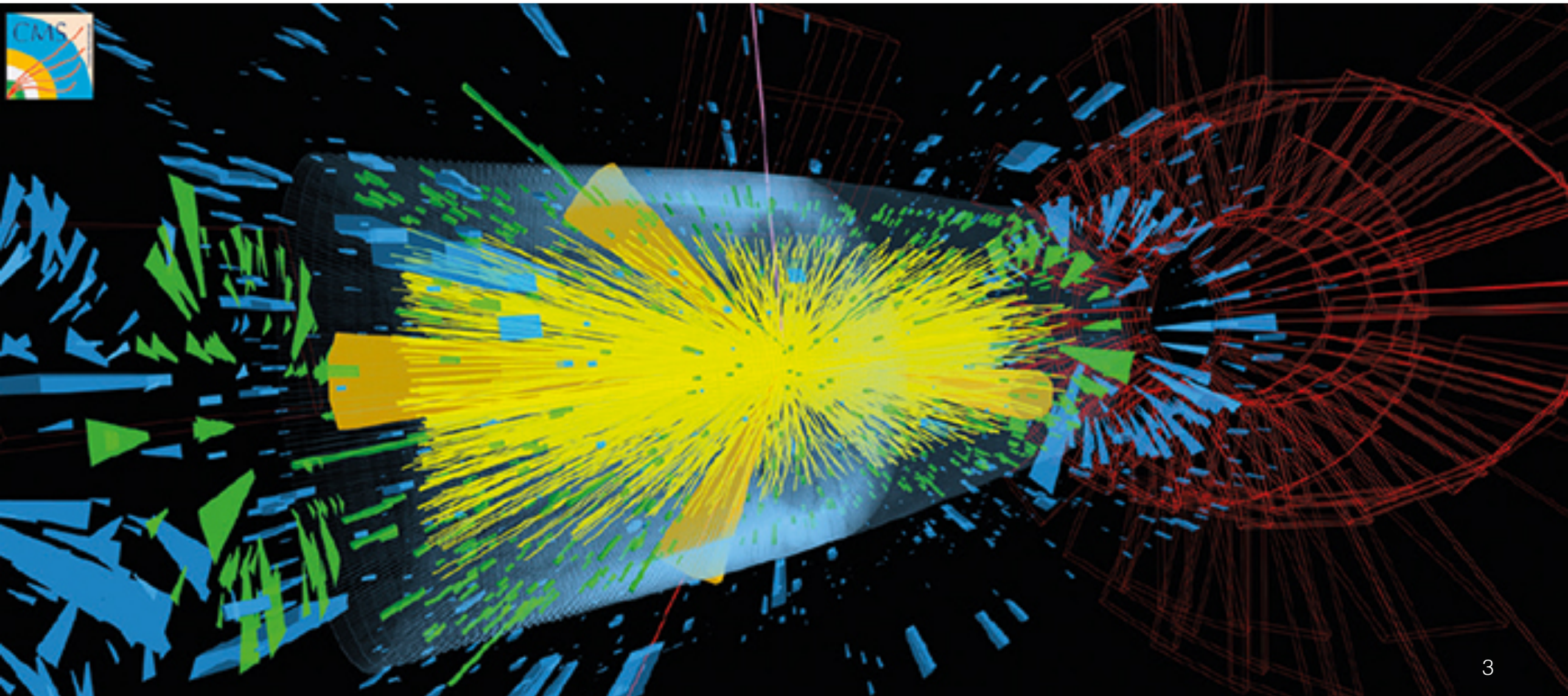
Contents

- Introduction - LHC ‘big data’ problem
- Neural Network to FPGA translation with hls4ml

The challenge: triggering at (HL-)LHC

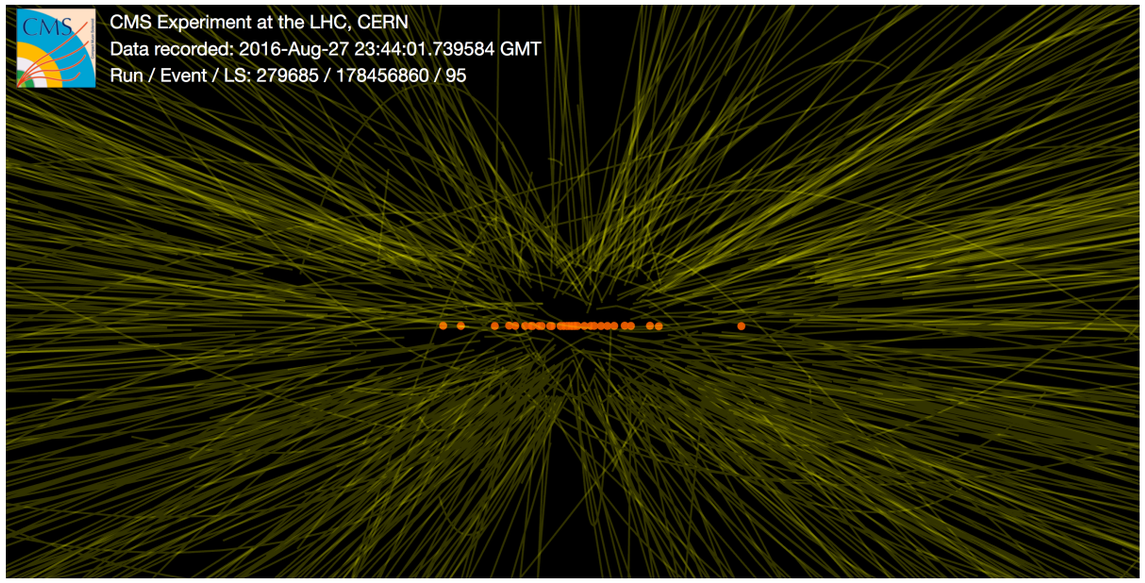
Extreme bunch crossing frequency of 40 MHz → extreme data rates O(100 TB/s)

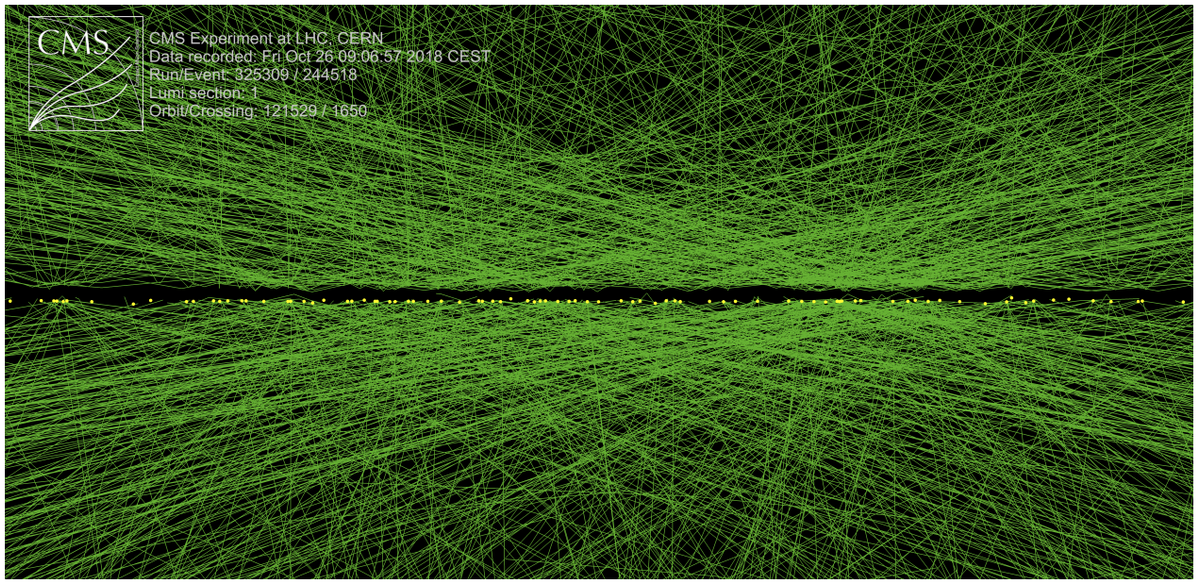
“Triggering” = filter events to reduce data rates to manageable levels

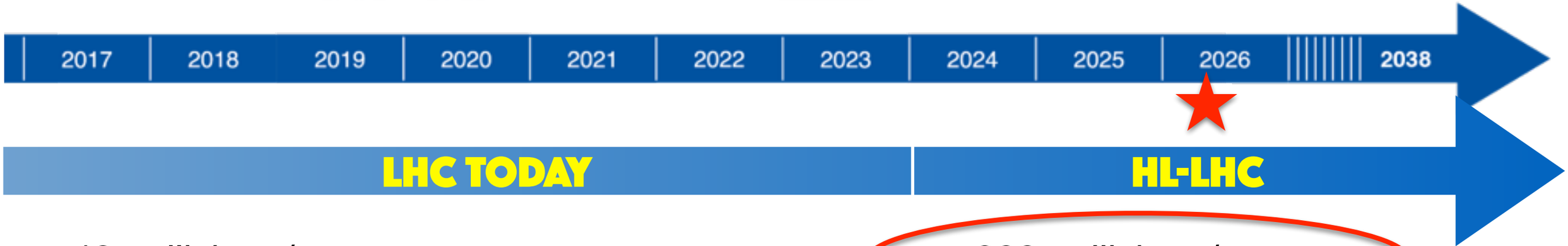


Future challenges @ LHC

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







- ~ 40 collisions/event
- ~ 10 sec/event processing time

- ~ 200 collisions/event
- more granular detector
- ~ minutes/event processing time
- flat budget for computing resources

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

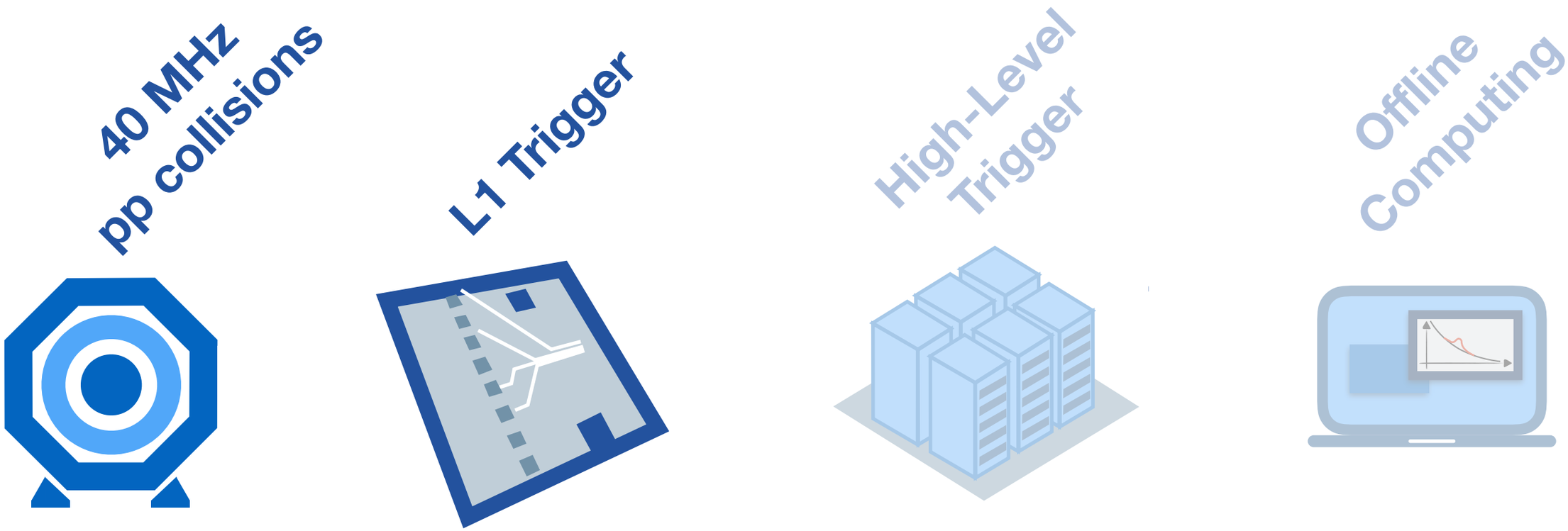
2017 + Run 3: <PU> ~ 50-80

HL-LHC: 140-200

CHALLENGE: maintain physics in increasingly complex collision environment
→ untriggered events lost forever!

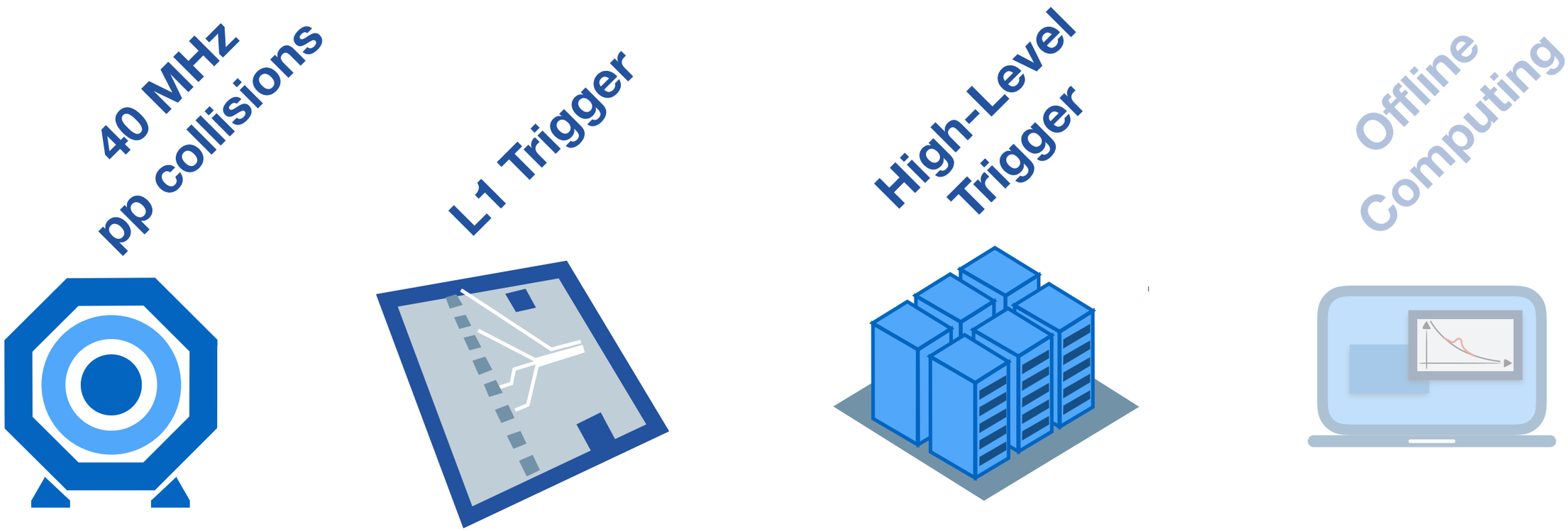
Sophisticated techniques needed to preserve the physics!

The LHC big data problem



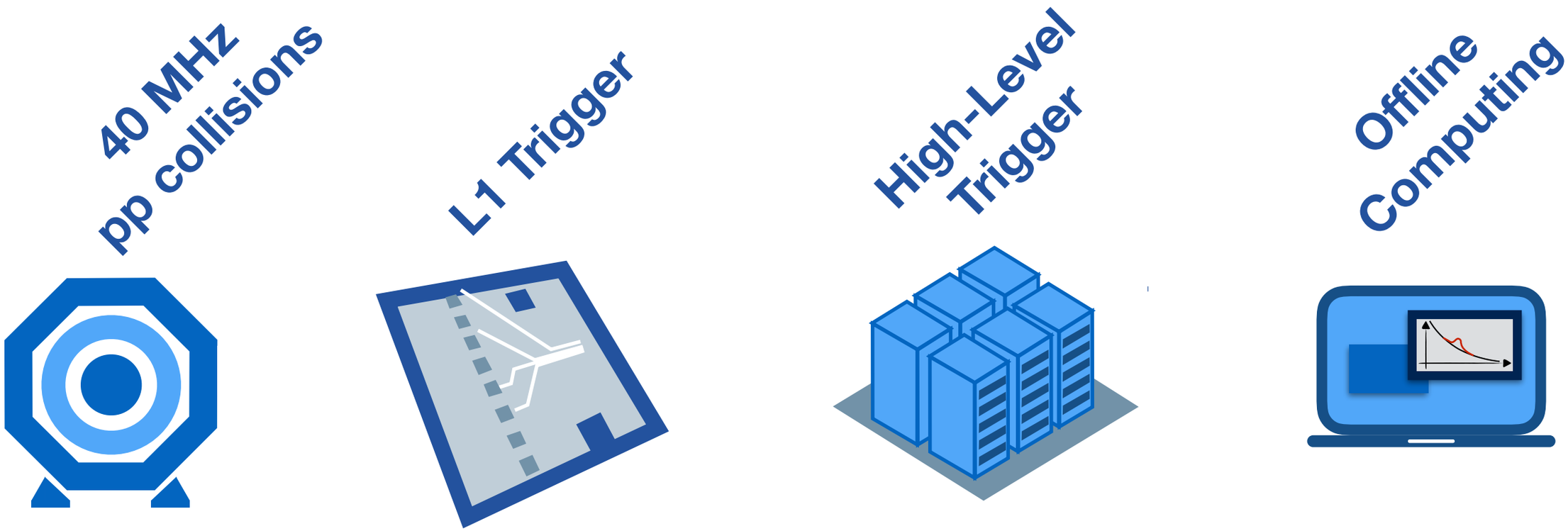
- 40 MHz in / 100 KHz out
- Absorbs 100s TB/s
- Trigger decision to be made in **~ 10 µs**
- Coarse local reconstruction
- FPGAs / Hardware implemented

The LHC big data problem



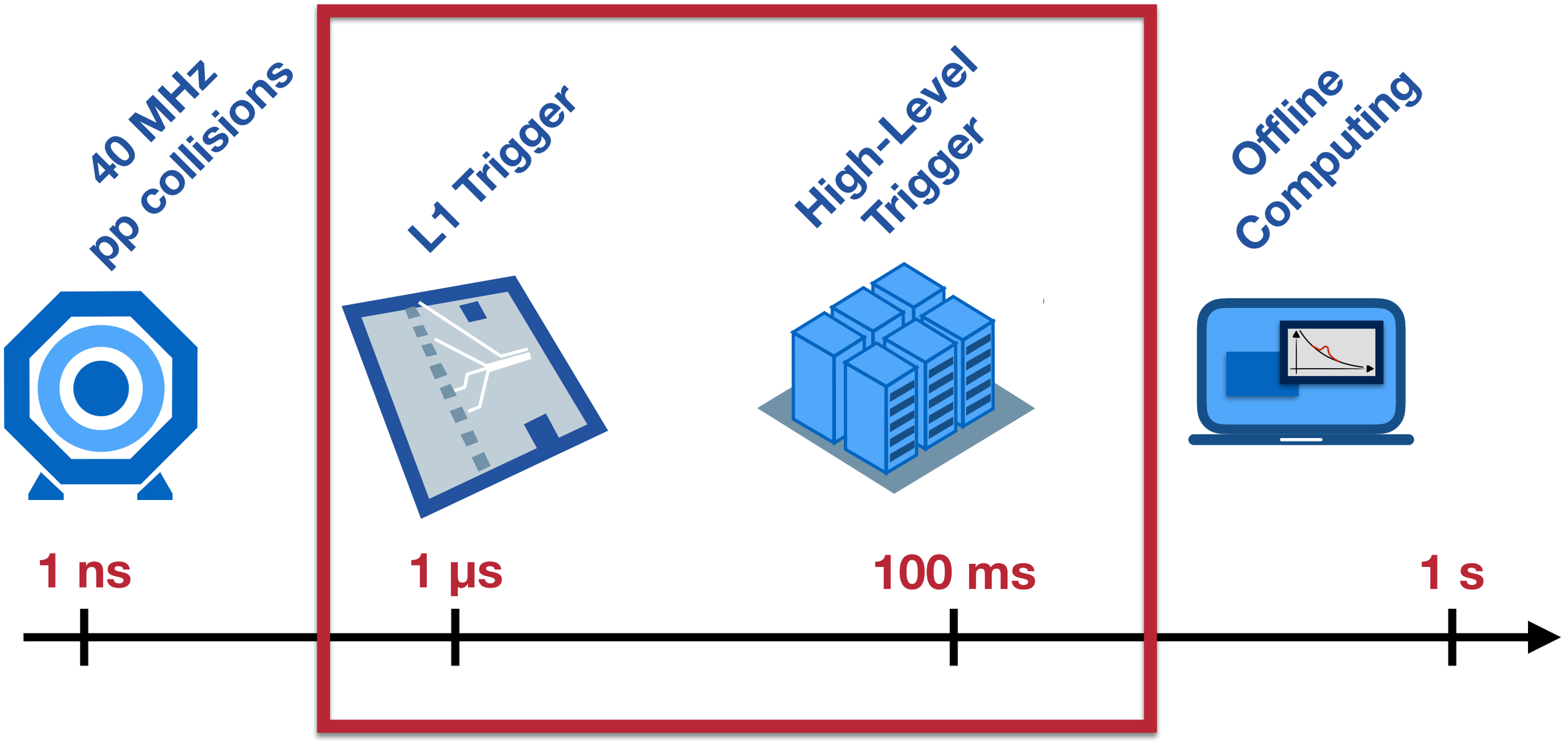
- 100 KHz in / 1 KHz out
- Output: ~ 500 KB/event
- Processing time **~ 300 ms**
- Simplified global reconstruction
- Software implemented on CPUs

The LHC big data problem



- Output: max. 1 MB/event
- Processing time **~ 20 s**
- Accurate global reconstruction
- Software implemented on CPUs

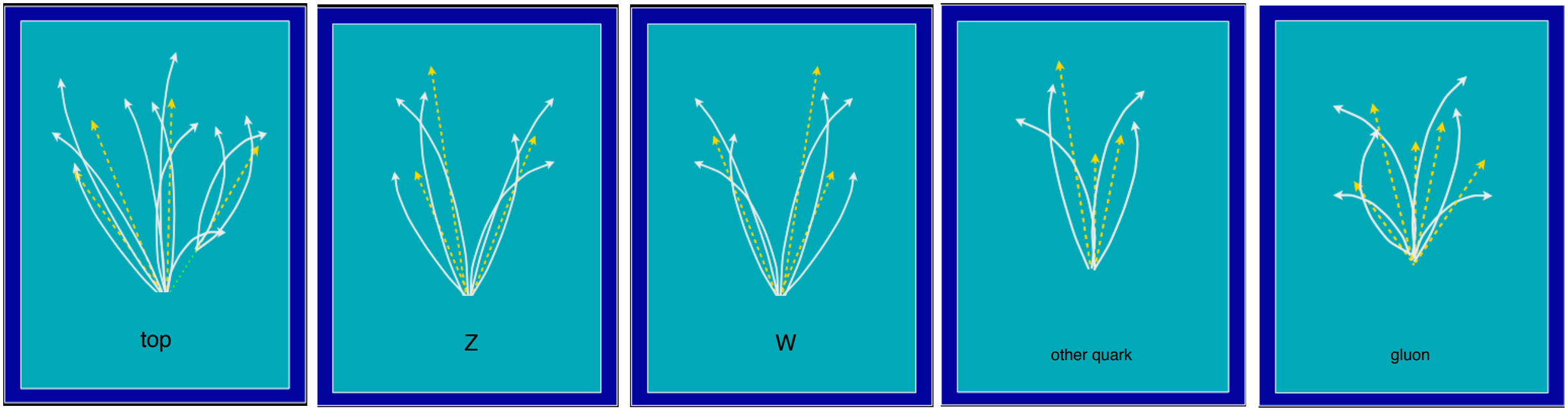
The LHC big data problem



Deploy ML algorithms very early in the game
Challenge: strict latency constraints!

Physics case: jet tagging

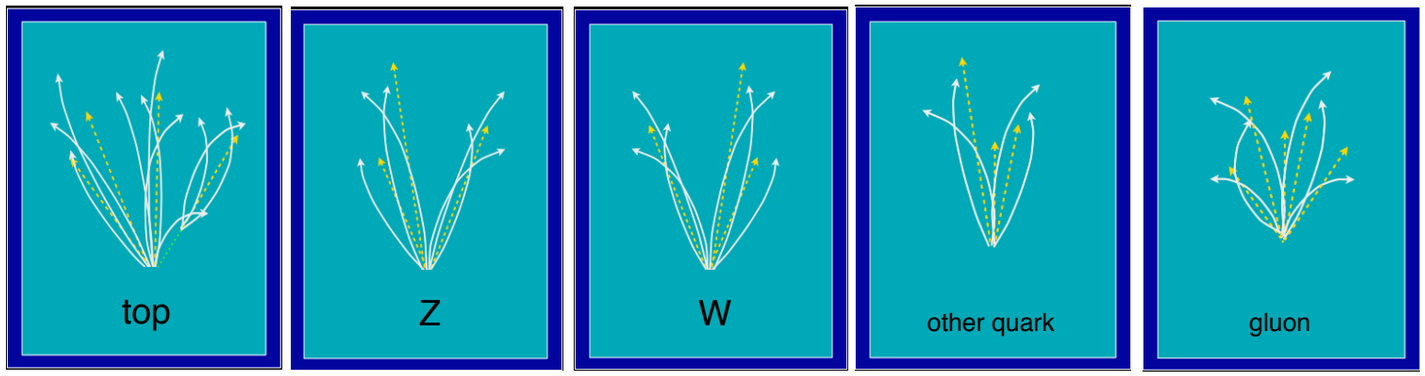
Study a **multi-classification task to be implemented on FPGA:** discrimination between highly energetic (boosted) ***q, g, W, Z, t*** initiated jets

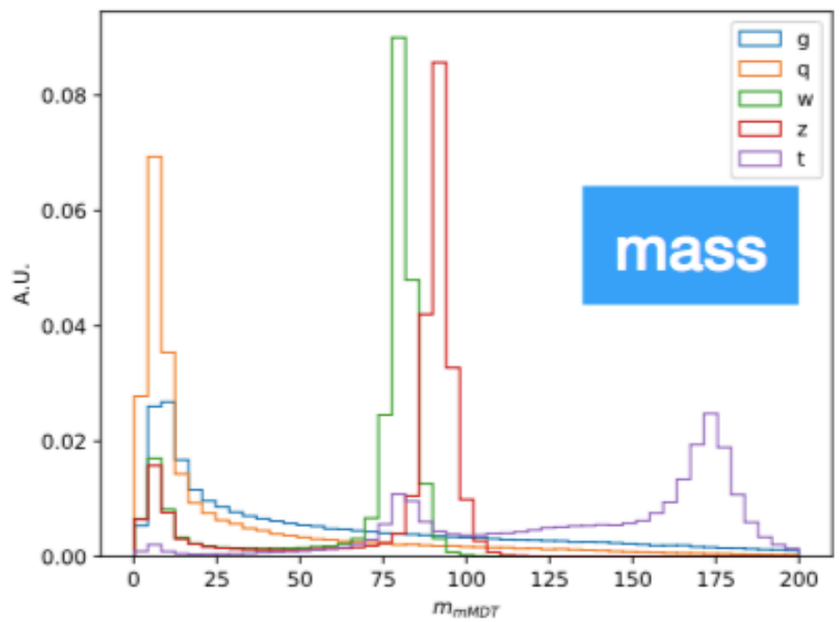


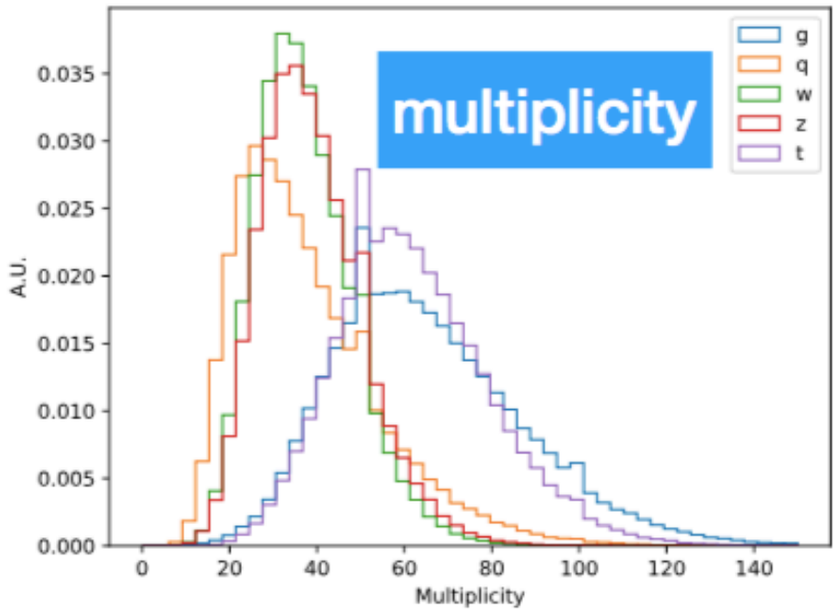
t→bW→bqq	**Z→qq**	**W→qq**	**q/g background**
3-prong jet	2-prong jet	2-prong jet	no substructure and/or mass ~ 0

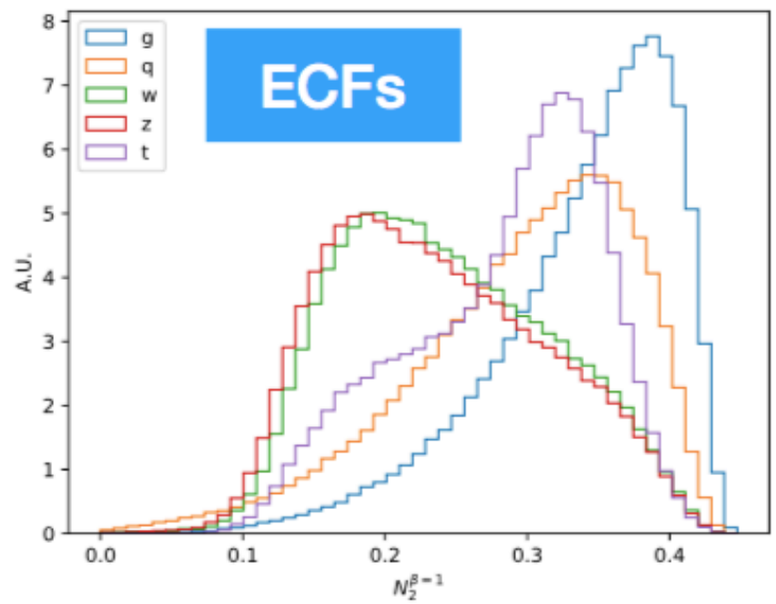
Reconstructed as one massive jet with substructure

Physics case: jet tagging









Input variables: several observables 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..

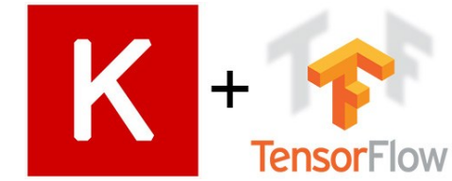
Physics case: jet tagging

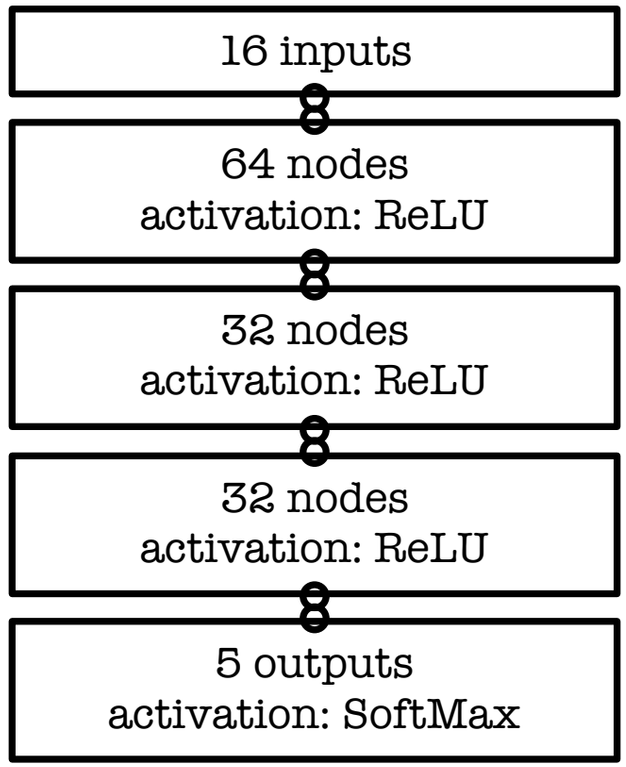
- We train (on GPU) the **five output multi-classifier** on a sample of ~ 1M events with two boosted WW/ZZ/tt/qq/gg anti-k_T jets

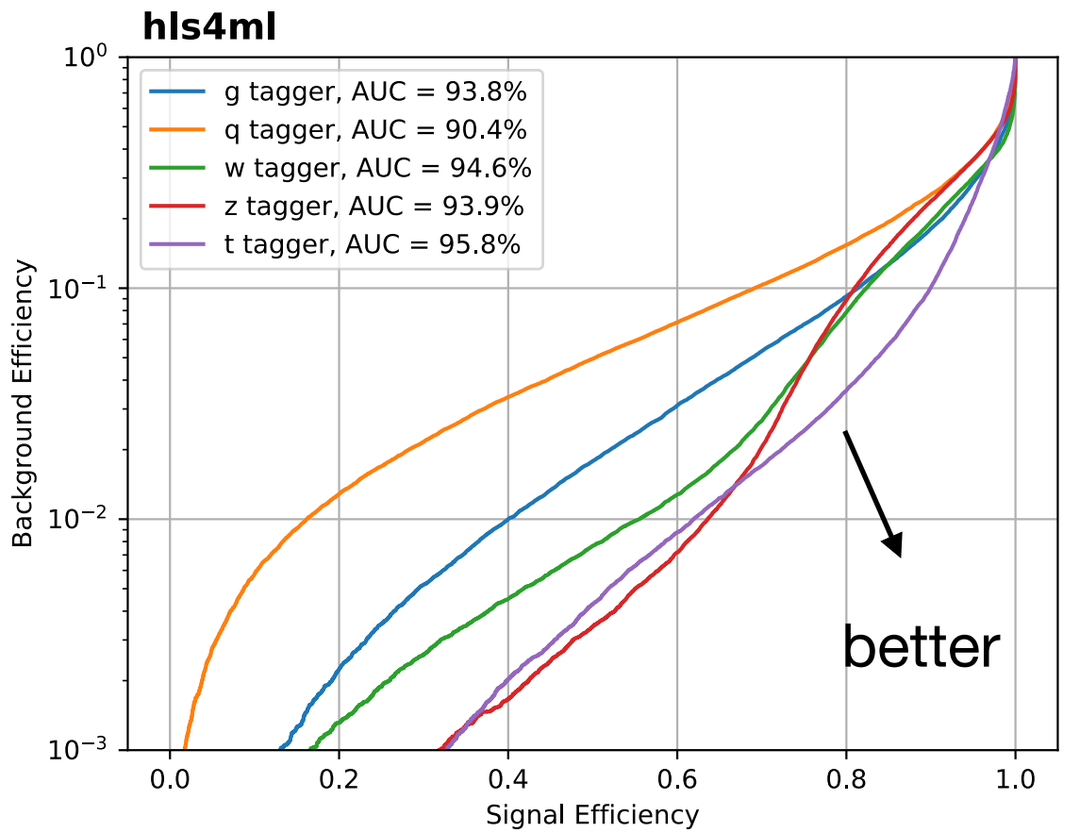
- Fully connected neural network with **16 expert-level inputs**:

  - Relu activation function for intermediate layers

  - Softmax activation function for output layer







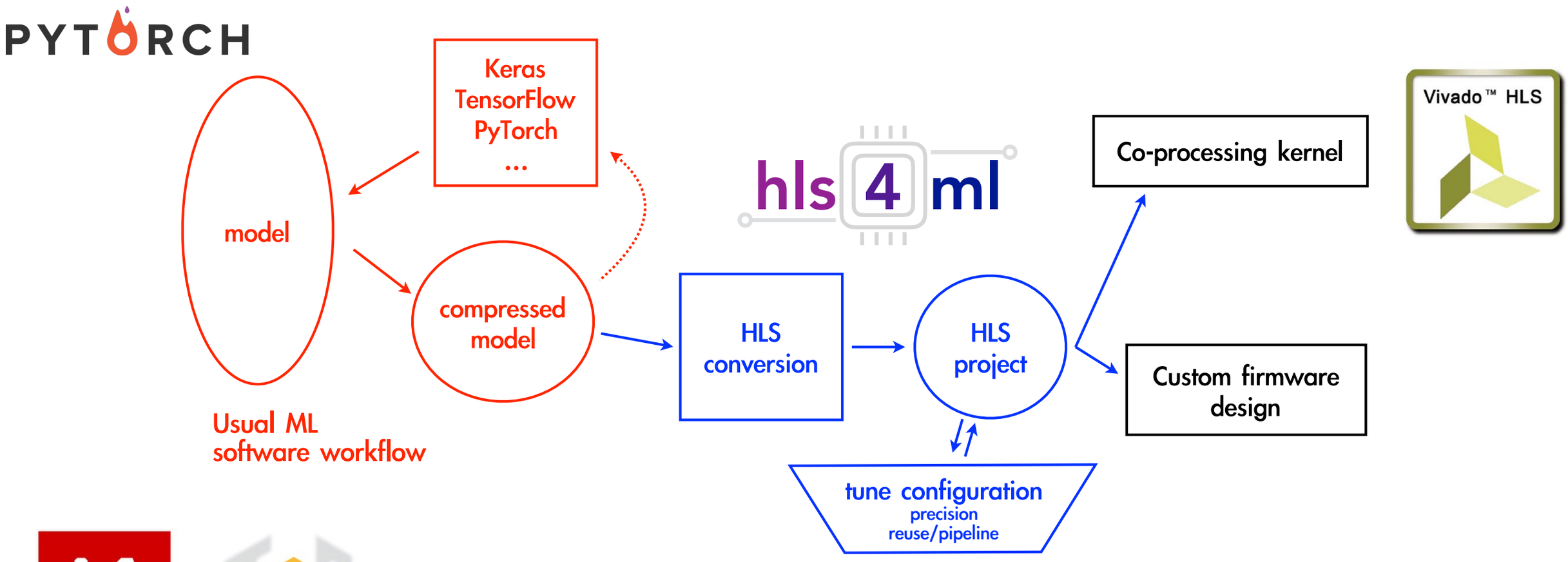
AUC = area under ROC curve (100% is perfect, 20% is random)

Neural Network to FPGA translation with hls4ml

high level synthesis for machine learning

Implemented a 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 …)



K + TensorFlow

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

https://arxiv.org/abs/1804.06913

hls4ml : unique aspects

- Due to extreme constrains from trigger system we remove some complexity from run-time, and move it to compile-time
  - Weights are ‘baked into’ FPGA firmware - not reconfigurable without reprogramming device
    - For longer latency applications, weights storage in on-chip block memory is possible
    - No loading weights from peripherals via - e.g. DDR, PCIe
- Allow many different: layer types, activation functions, kernel sizes, input/output dimensions at compile-time while keeping hardware fully utilised at run-time
- Keeps the latency and resource usage low: inference is optimized for the model
- User controllable trade-off between resource usage and latency/throughput
- Thanks to HLS: easy to target different devices, clock frequency

Efficient NN design for FPGAs

FPGAs provide huge flexibility

Performance depends on how well you take advantage of this

Constraints:

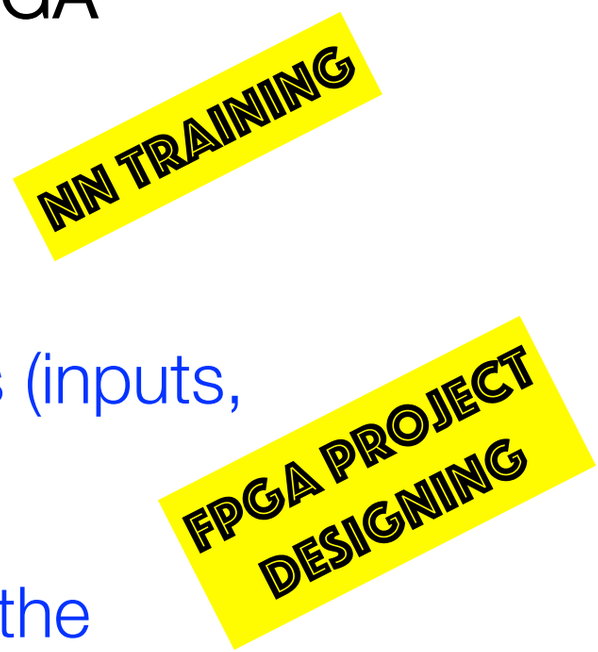
Input bandwidth

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



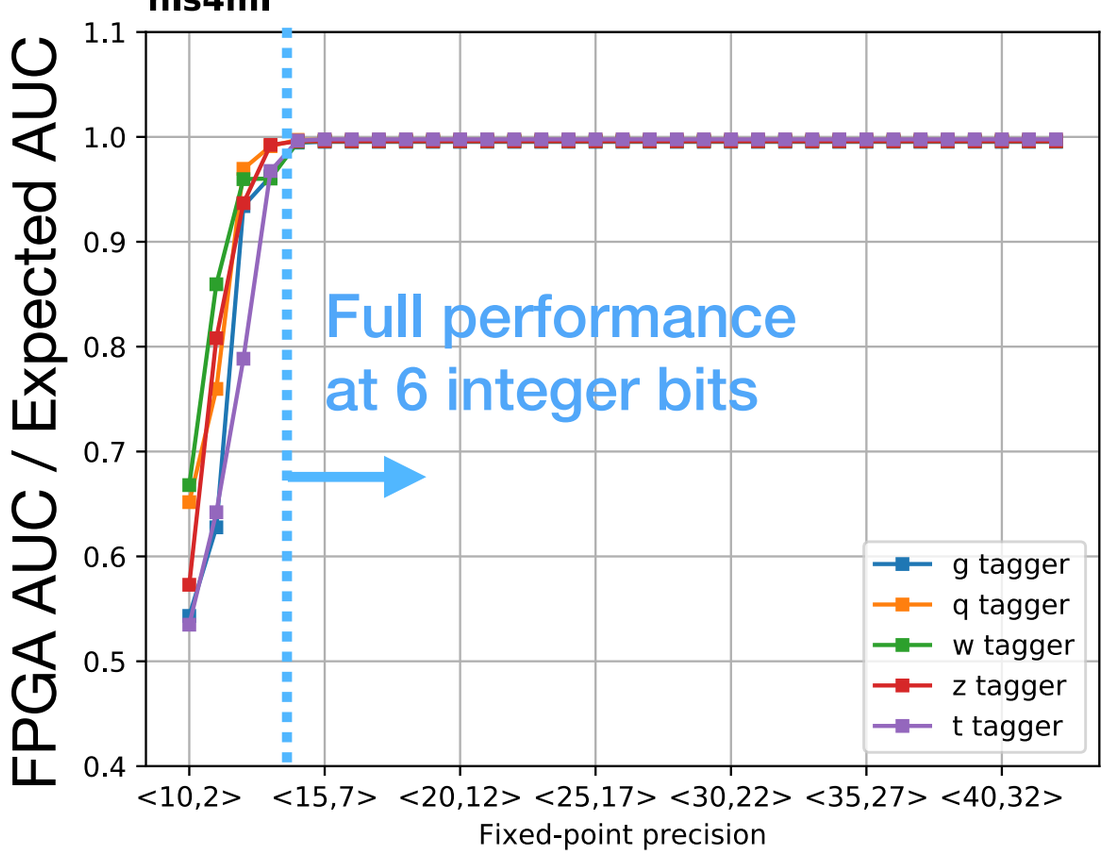
Efficient NN design: quantization

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

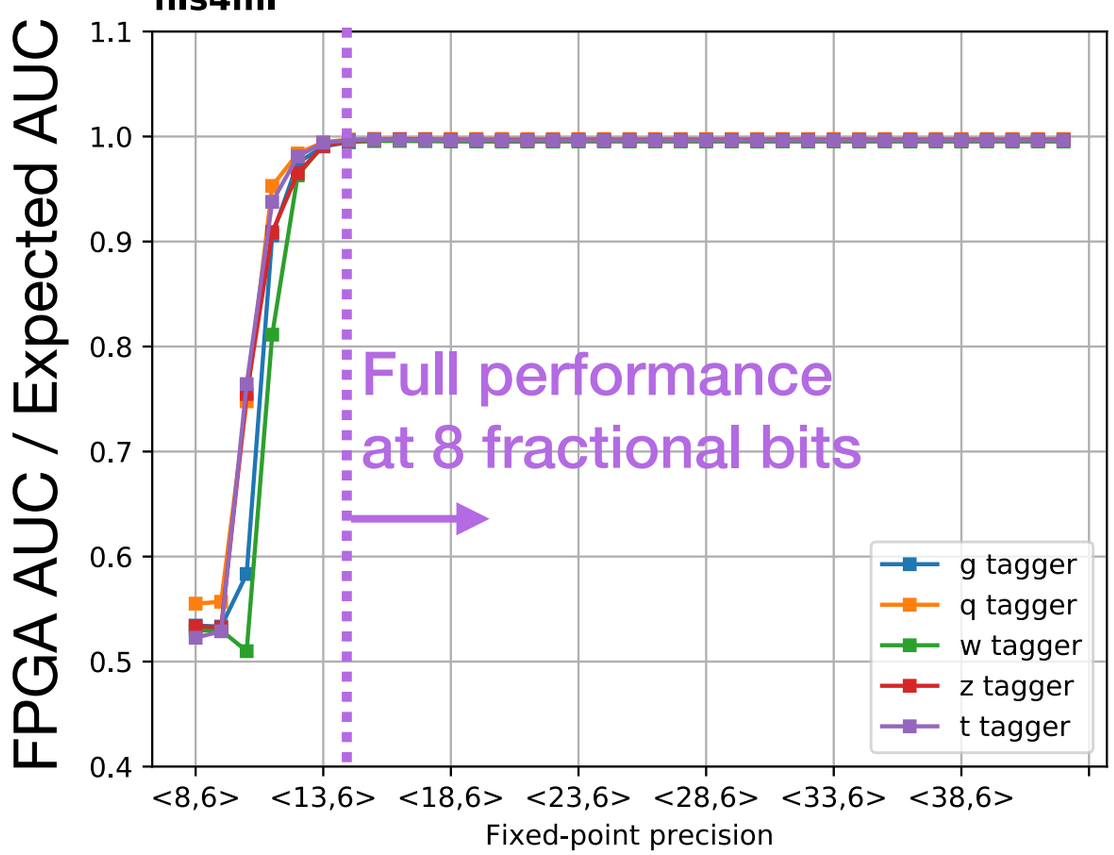
Scan integer bits

Fractional bits fixed to 8

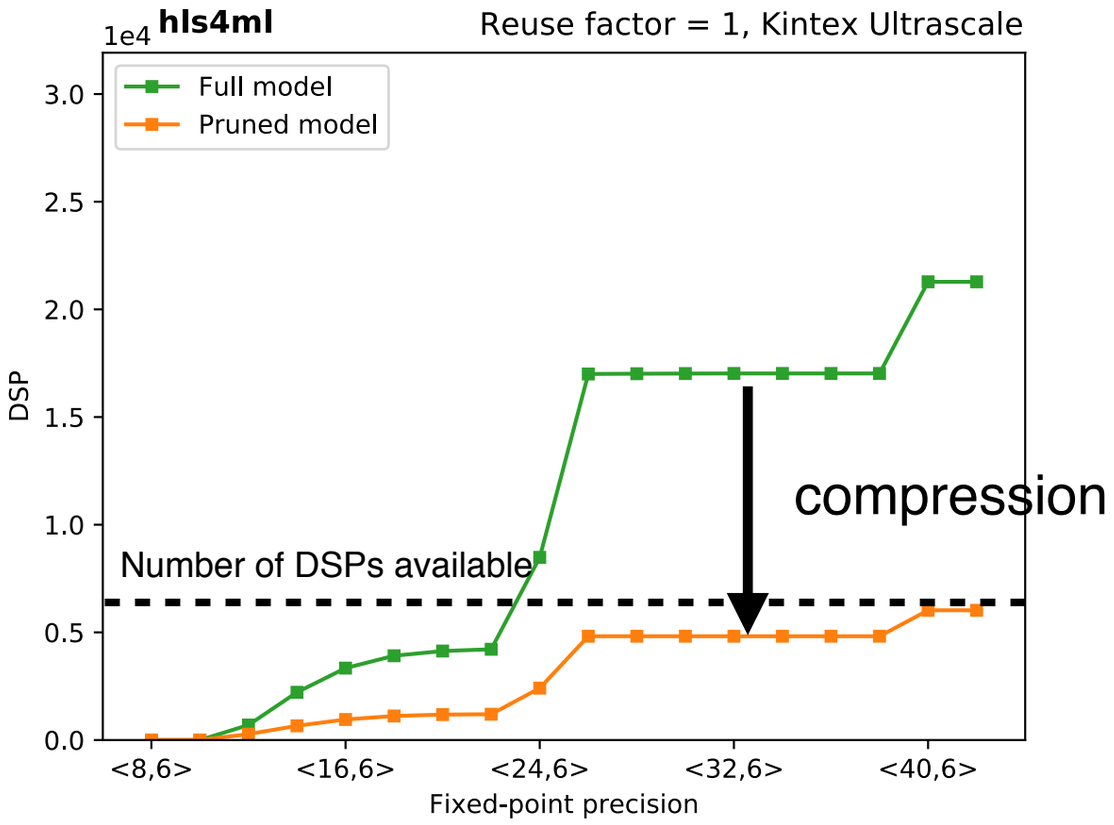


Scan fractional bits

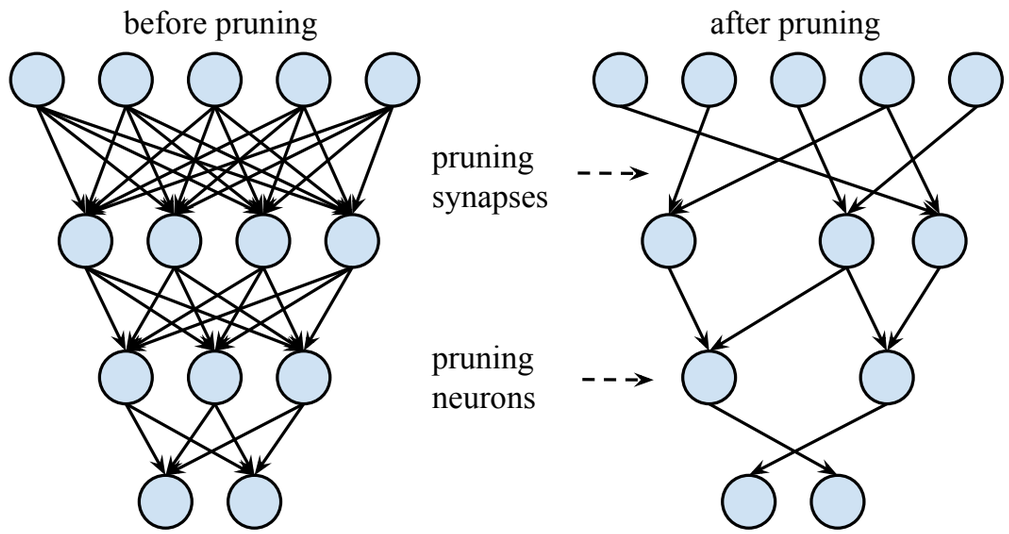
Integer bits fixed to 6



Efficient NN design: compression

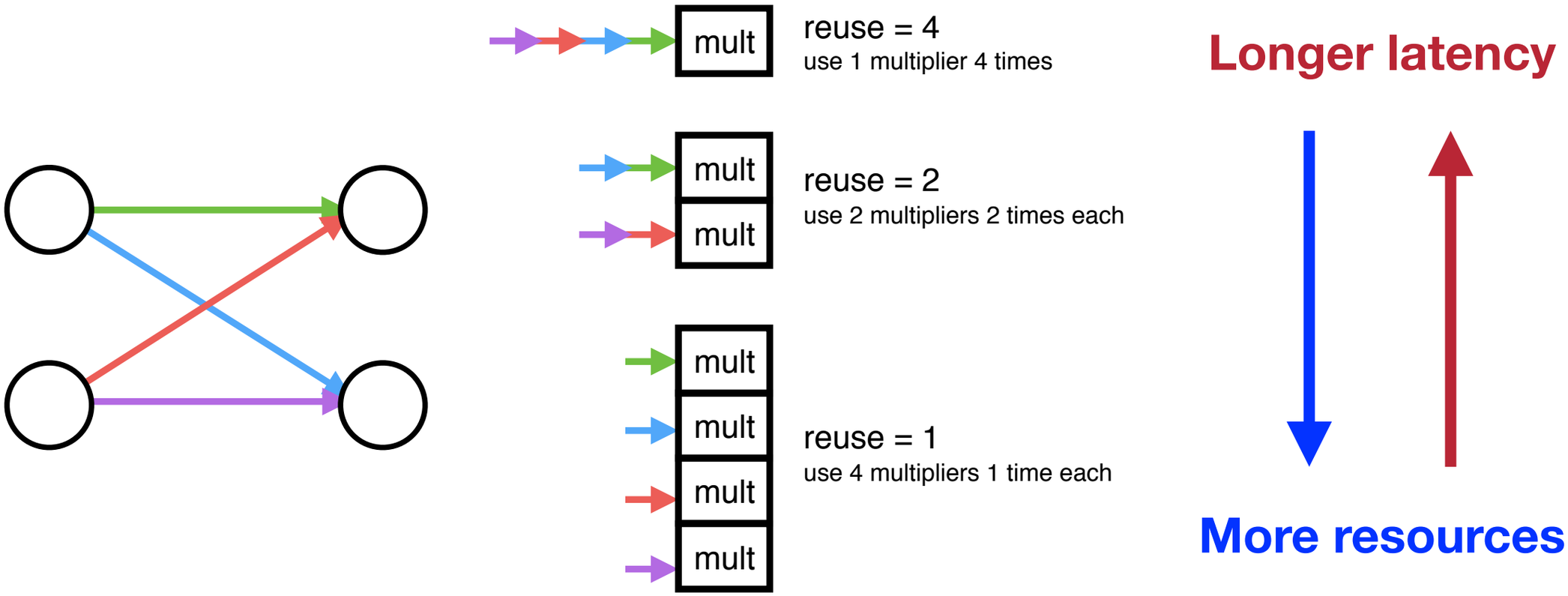


70% compression ~ 70% fewer DSPs



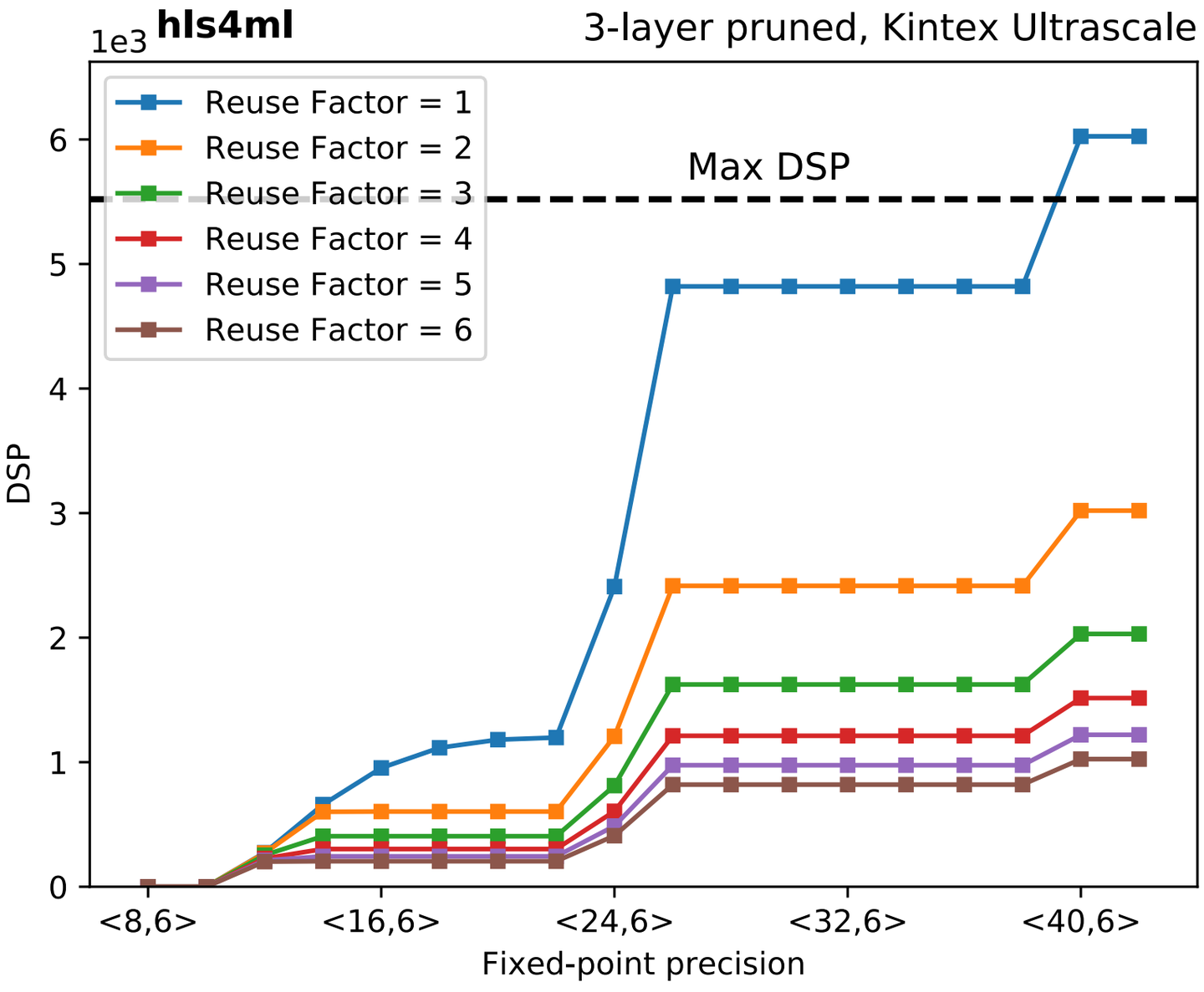
- DSPs (used for multiplication) are often limiting resource
  - DSPs have a max size for input (e.g. 27x18 bits), so number of DSPs per multiplication changes with precision

Efficient NN design: reuse



- Key feature of hls4ml: a handle to trade resource usage and latency/throughput
- Reuse = 1: fully unroll everything onto different resources
  - Fastest, most resource intensive
- Reuse > 1: one resource used sequentially for several operations
  - Slower, but save resources

Parallelization: DSPs usage



Fully parallel
Each mult. used 1x

Each mult. used 2x

Each mult. used 3x

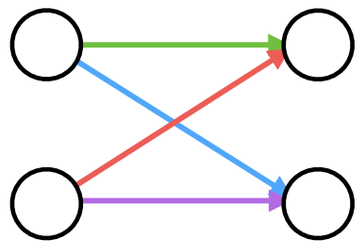
⋮

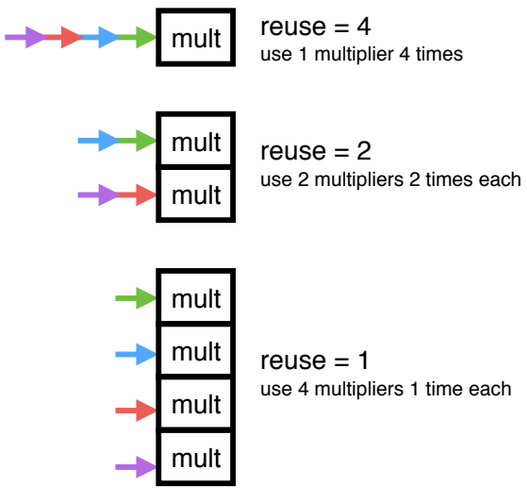
Reuse factor: how much to parallelize operations in a hidden layer

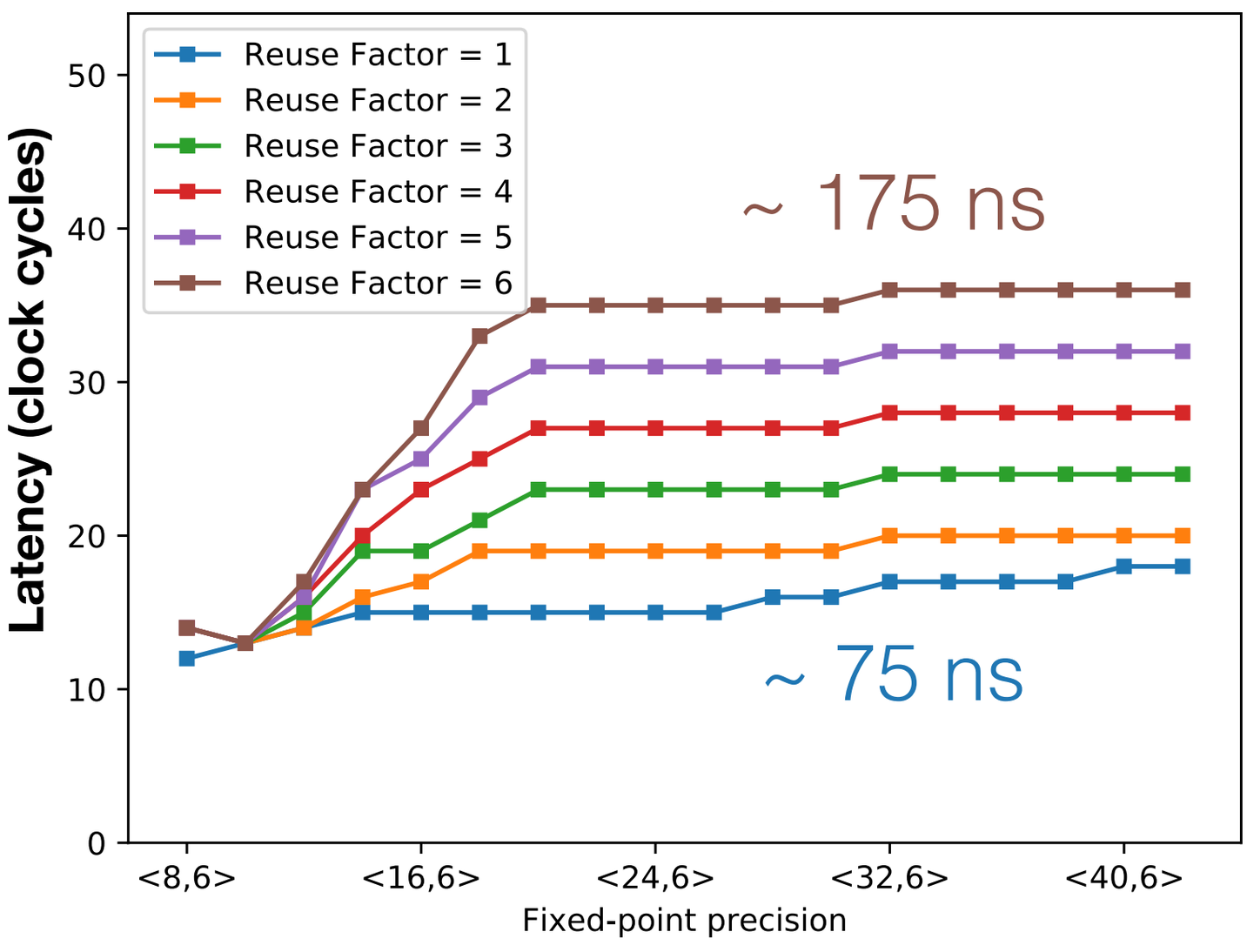
Parallelization: Timing

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

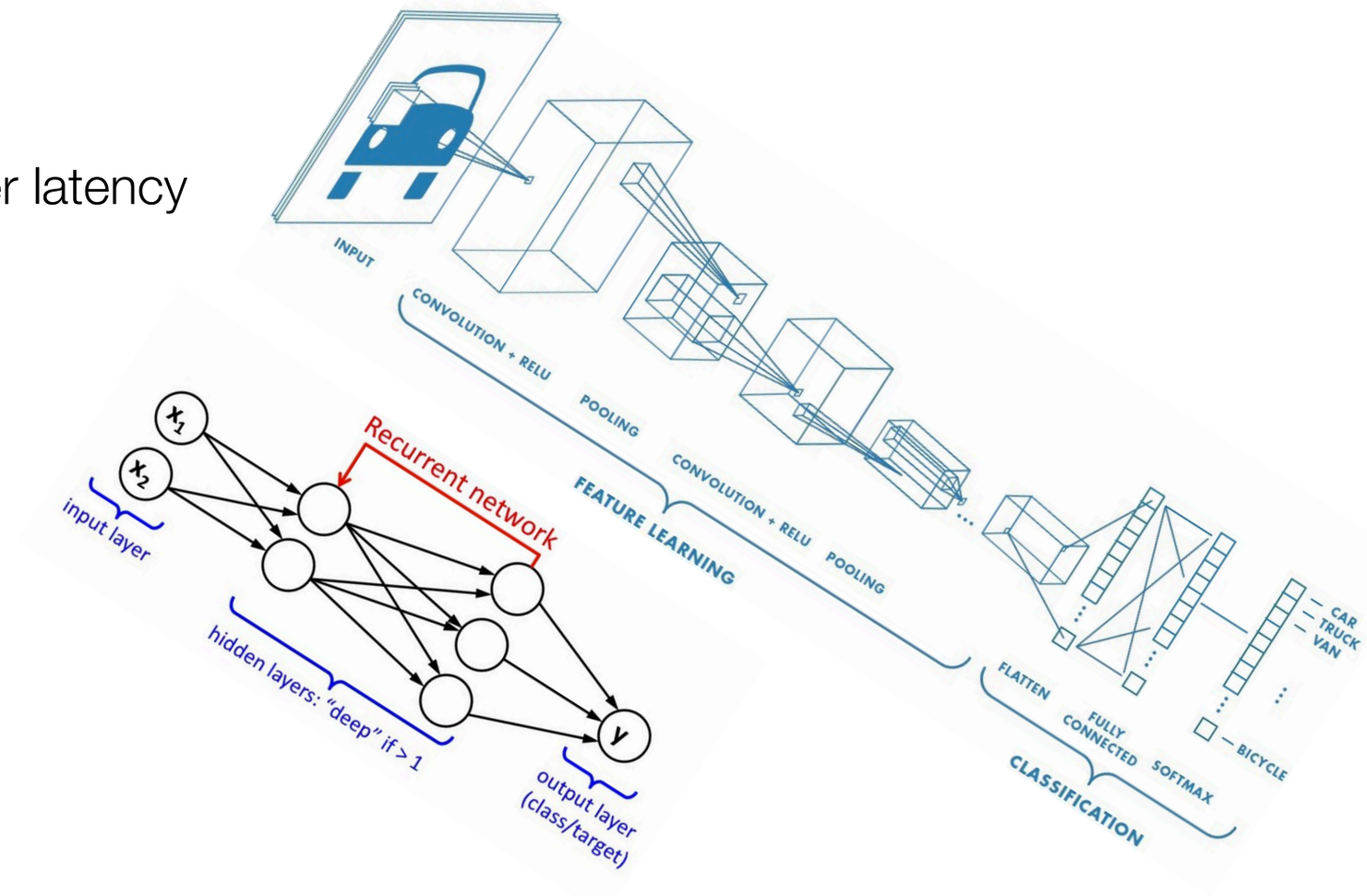
⋮

Fully parallel
Each mult. used 1x

More resources

Features under development

- ‘Large’ neural networks
  - Very high ‘reuse’ factor -> longer latency
- Convolutional Neural Networks
  - Popular for image processing
- Recurrent neural networks
  - For time series/list processing
- Binary / Ternary neural networks
  - Very low precision weights, use FPGA resources efficiently
- Boosted Decision Trees
  - Not neural networks, but can be effective and efficient



Conclusion

- hls4ml software package translates trained neural networks into synthesizable FPGA firmware
- User can tune resource usage vs. latency/throughput
- Initially targeting Level 1 Trigger - big FPGAs, O(1 μs) latency
- Also working on larger networks with longer latency for other applications e.g. neutrino, astronomical experiments, industrial applications
- Website: https://hls-fpga-machine-learning.github.io/hls4ml/
- Paper: https://iopscience.iop.org/article/10.1088/1748-0221/13/07/P07027
- Code: https://github.com/hls-fpga-machine-learning/HLS4ML
- Fast ML Workshop: https://indico.cern.ch/event/822126/