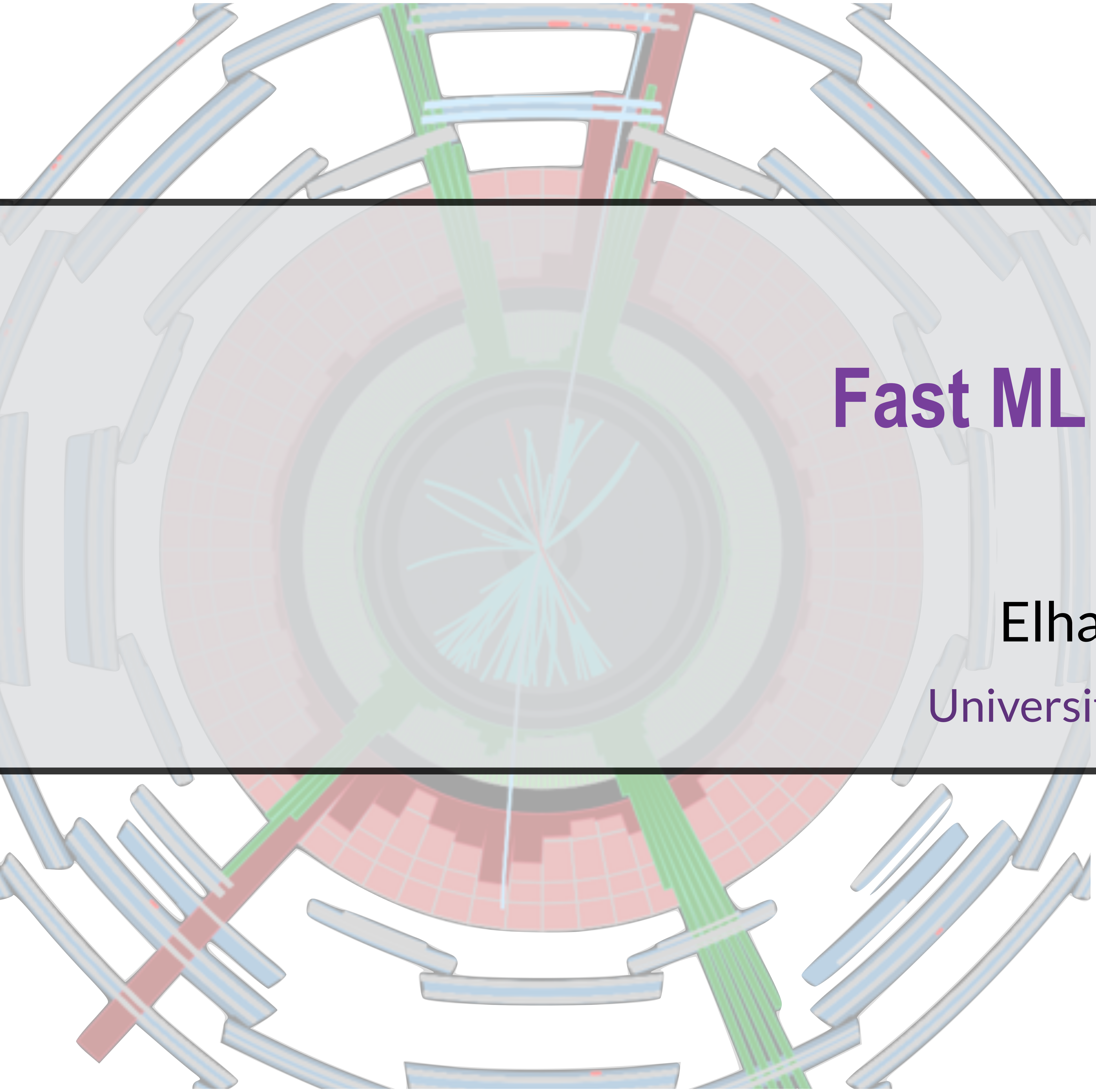
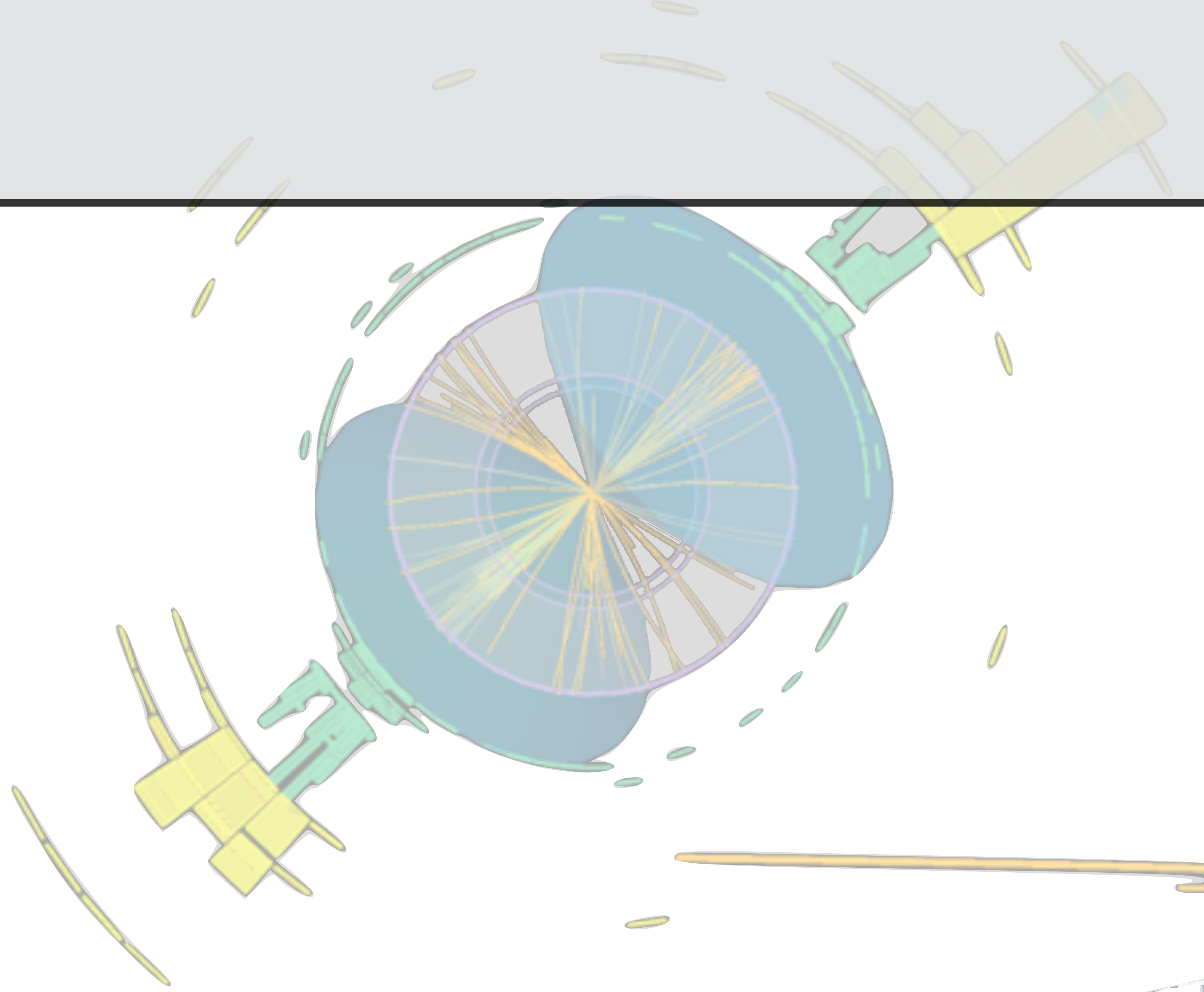


Fast ML Inference on FPGA

Elham E Khoda

University of Washington



US ATLAS ML Training 2023

July 28, 2023

UNIVERSITY *of* WASHINGTON

a3d3 Accelerated AI Algorithms for Data-Driven Discovery

Getting started!

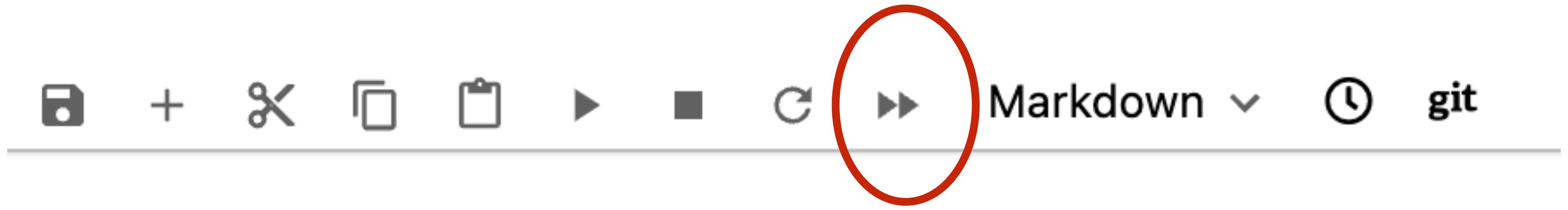
- Join hls4ml-tutorial GitHub Organization (check email for invite)

- Open https://jhub.35.192.180.88.nip.io in your web browser
- Authenticate with your GitHub account (login if necessary)

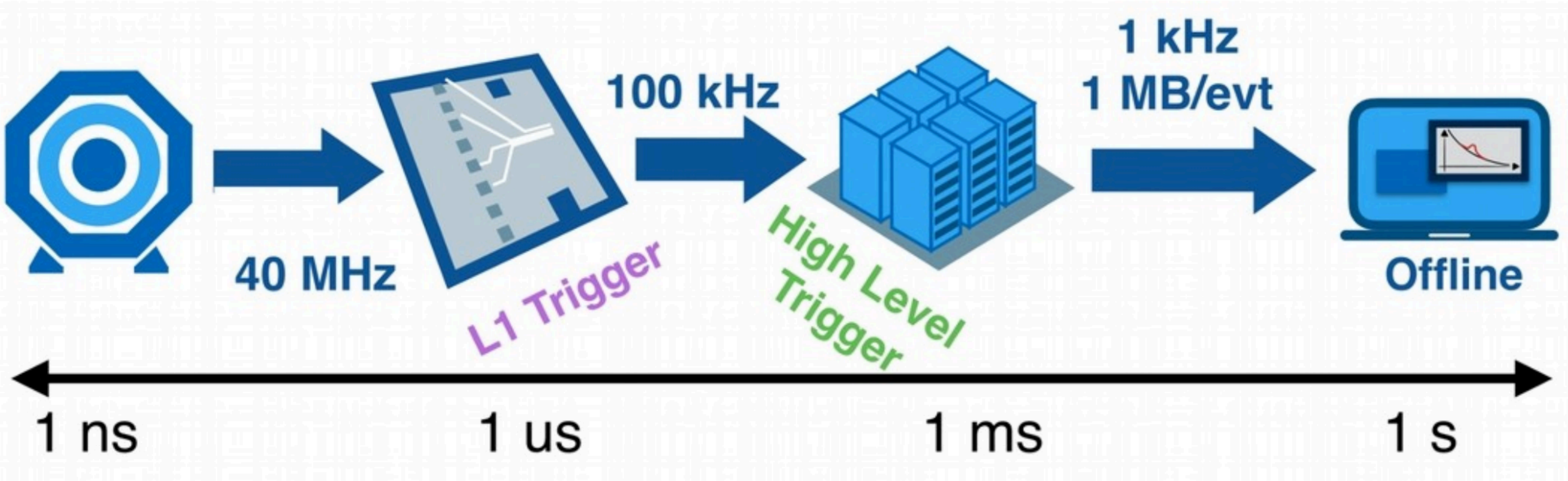
Open and start running through "part1_getting_started" !

Run all the cells

Hit this



ATLAS Run-3 Data Processing



L1 Trigger (hardware: FPGAs) – *O(μs) hard latency*

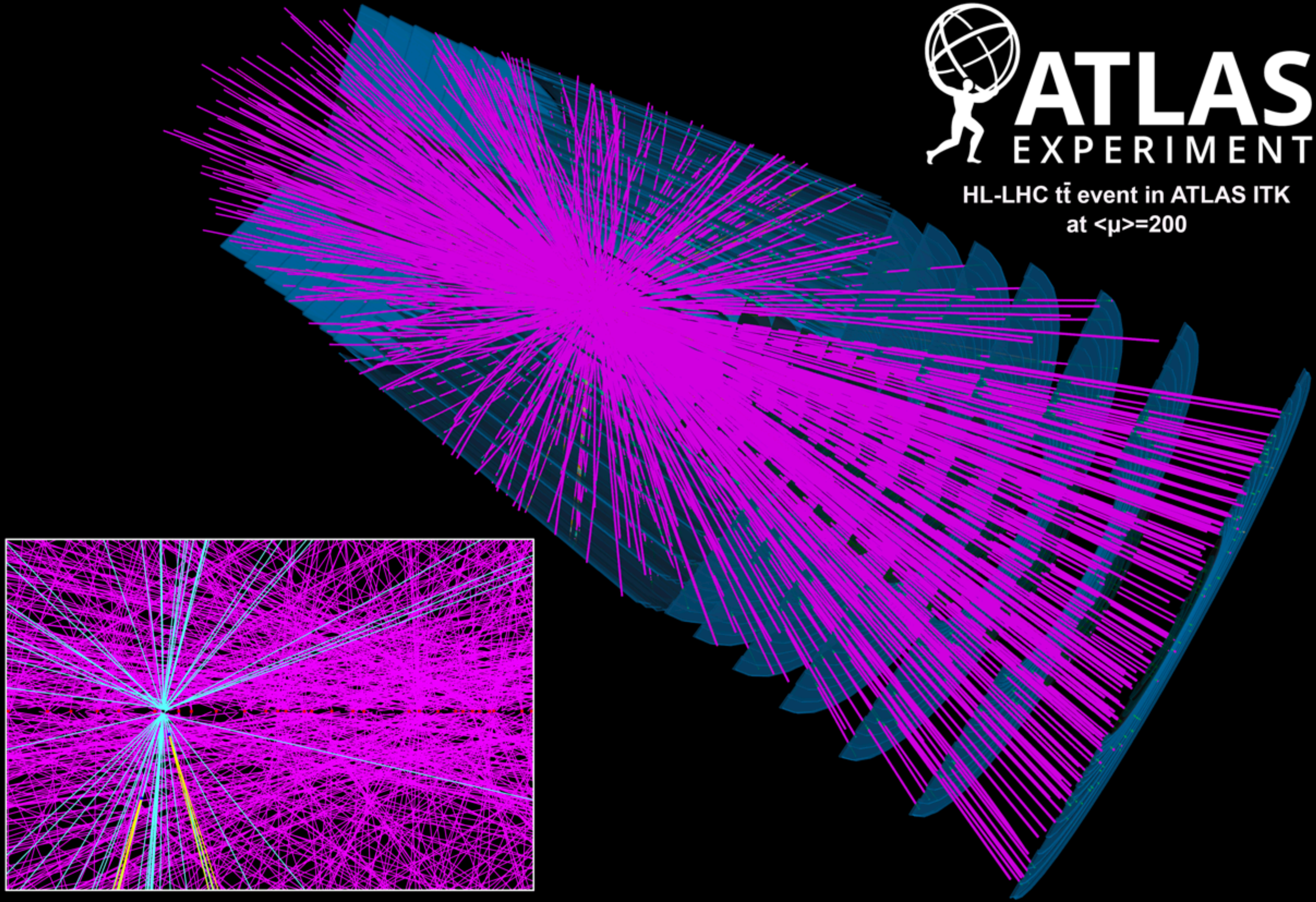
- Typically coarse selections are applied

High Level Trigger (software: CPUs) – *O(100 ms) soft latency*

- More complex algorithms (full detector information available), some BDTs and DNNs used

Offline (software: CPUs)

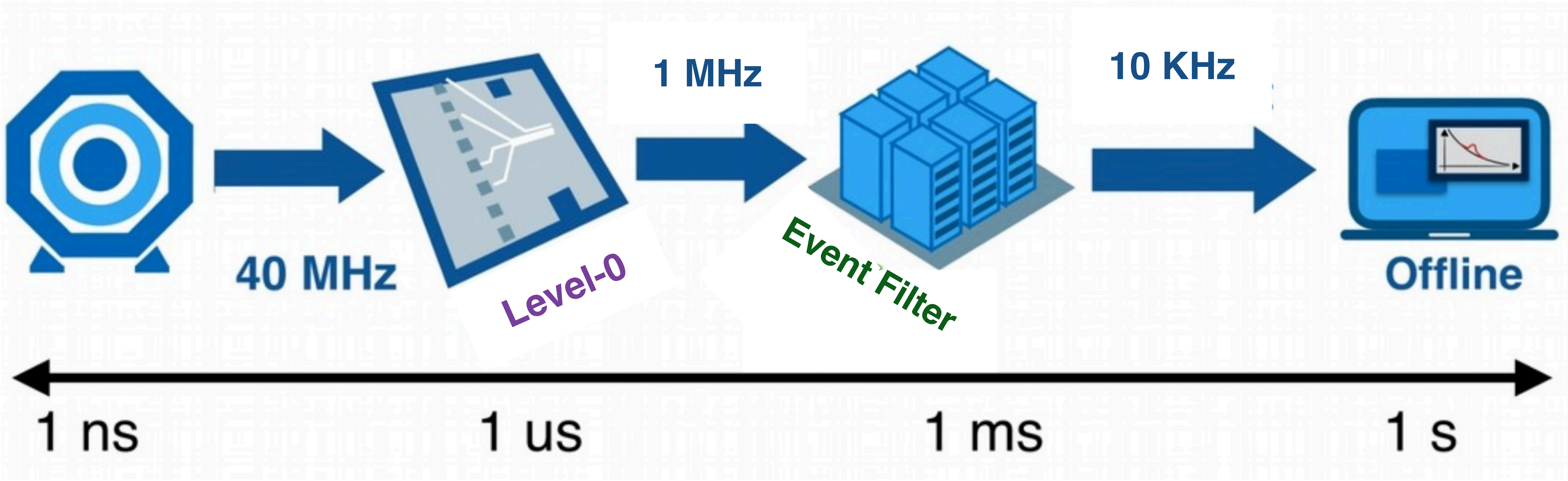
- Full event reconstruction, bulk of machine learning usage in ATLAS/CMS



ATLAS EXPERIMENT

HL-LHC $t\bar{t}$ event in ATLAS ITK
at $\langle\mu\rangle$=200

ATLAS Phase-II Data Processing



L0 Trigger (hardware: FPGAs) – *O(μs) hard latency*

- Typically coarse selections are applied

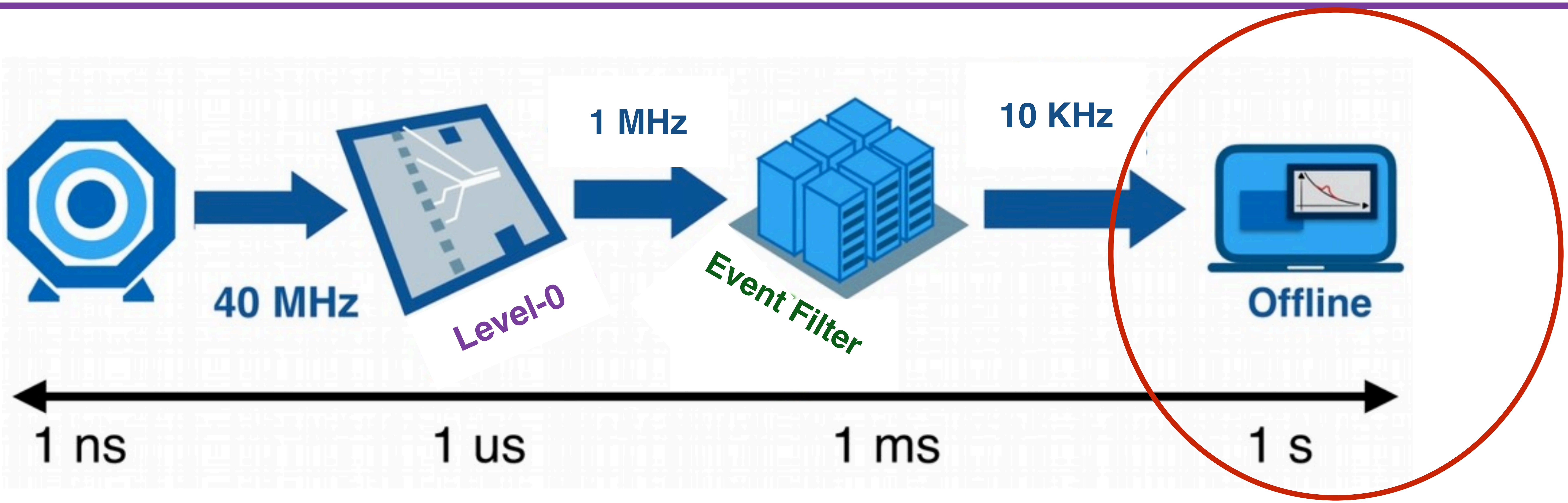
Event Filter (software: CPUs) – *O(100 ms) soft latency*

- More complex algorithms (full detector information available), some BDTs and DNNs used

Offline (software: CPUs)

- Full event reconstruction, bulk of machine learning usage in ATLAS/CMS

ATLAS Phase-II Data Processing

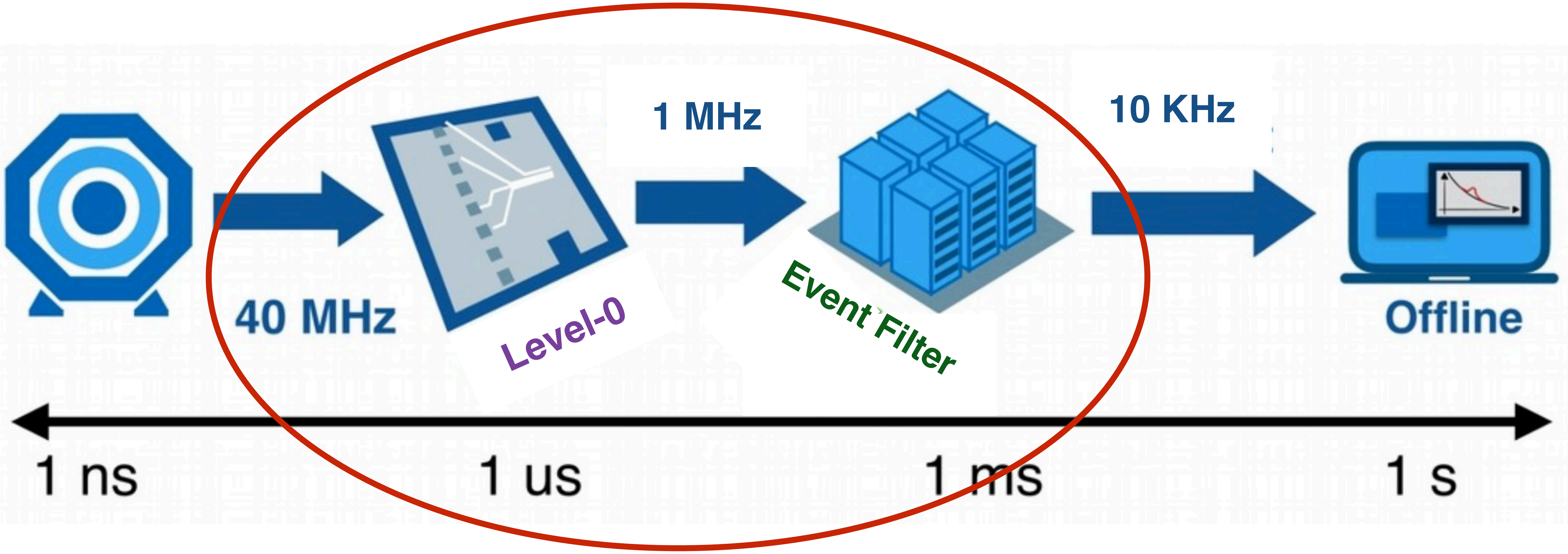


- Usage of ML is growing over time

- Active R&D
  Further improvements driven by more complicated algorithms

Usage of GPUs will be beneficial for the future LHC Runs (after 2026)

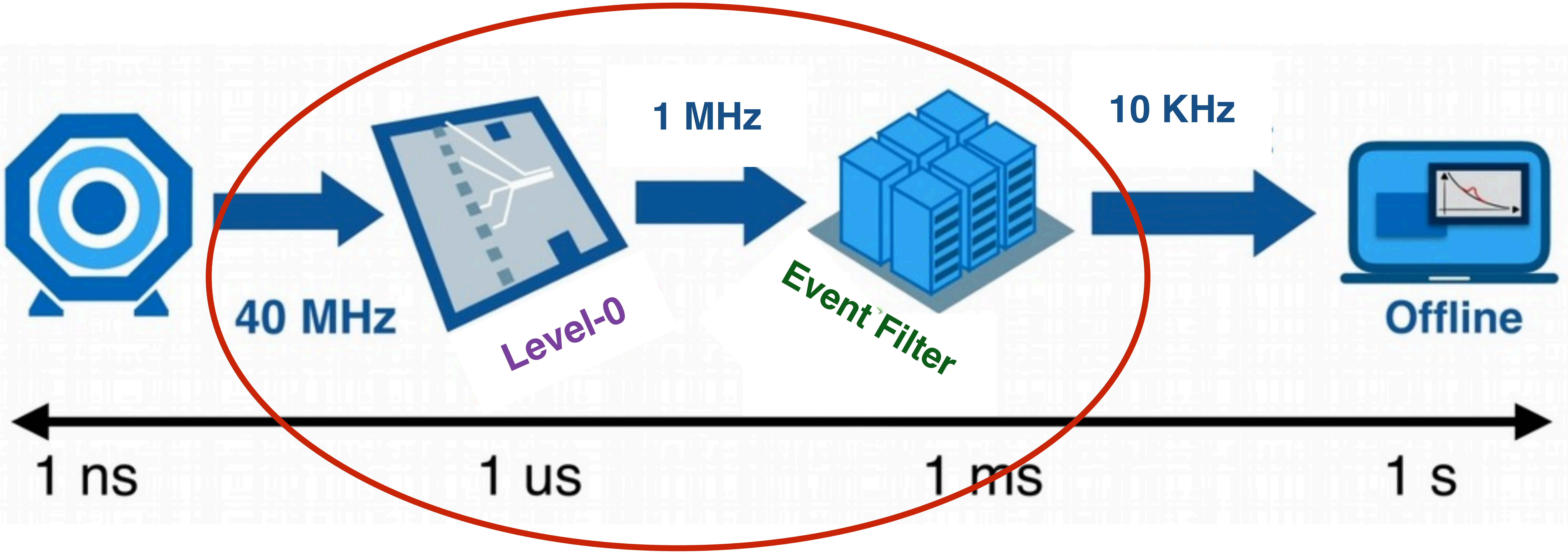
ATLAS Phase-II Data Processing



- ML has potential to improve physics performance in the trigger system

- **Strict latency requirements:** µs (ms) for Level-0 (Event Filter)
  For Level-0 trigger → we need to run ML on FPGAs

ATLAS Phase-II Data Processing



- ML has potential to improve physics performance in the trigger system

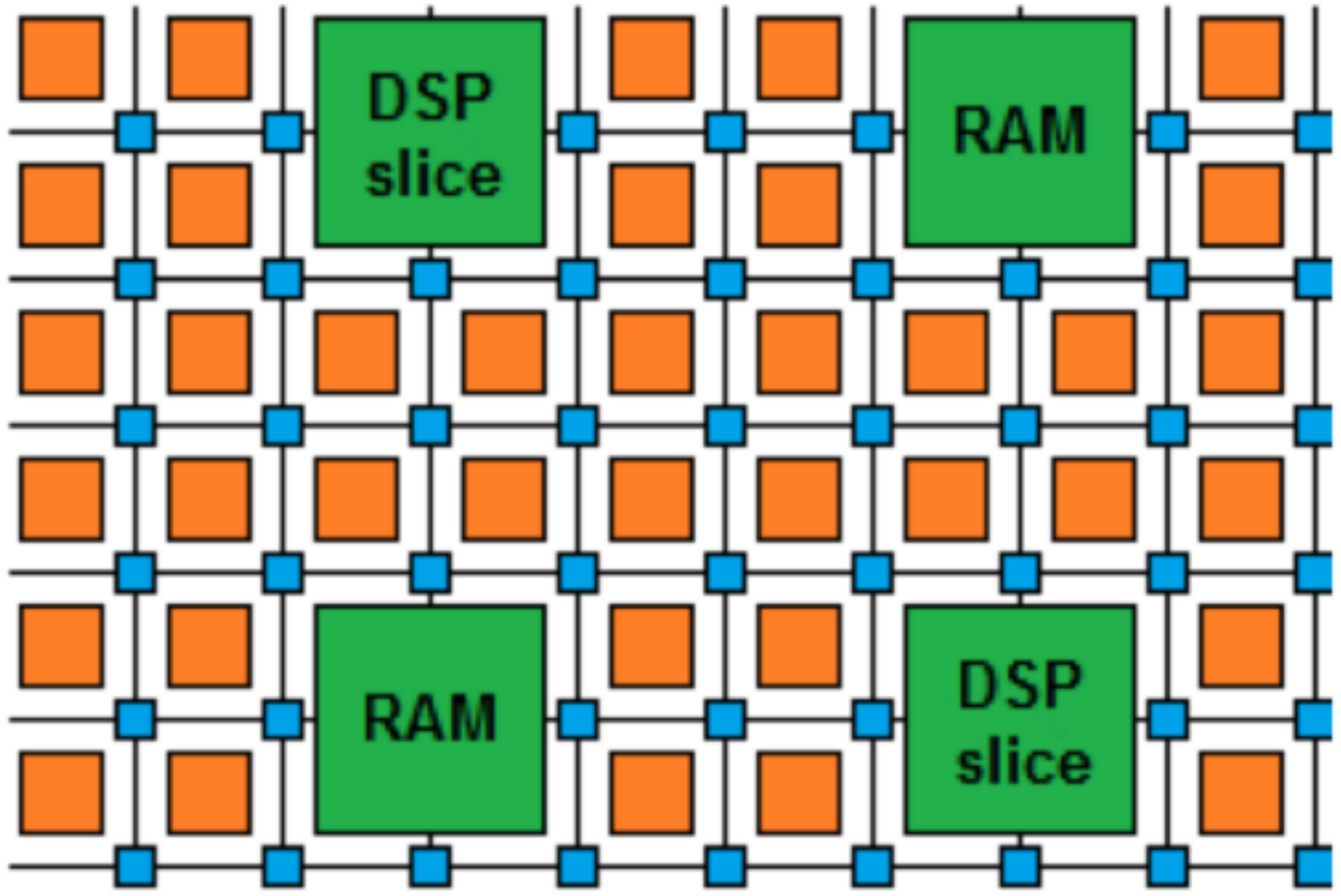
- **Strict latency requirements:** µs (ms) for **Level-0** (**Event Filter**)
  For **Level-0** trigger → we need to run ML on FPGAs

What is an FPGA?

Field **P**rogrammable **G**ate **A**rrays (FPGAs) are reprogrammable integrated circuits

- Contain many different building blocks (‘resources’) which are connected together as you desire

- Originally popular for prototyping ASICs, but now also for high performance computing



Building blocks:

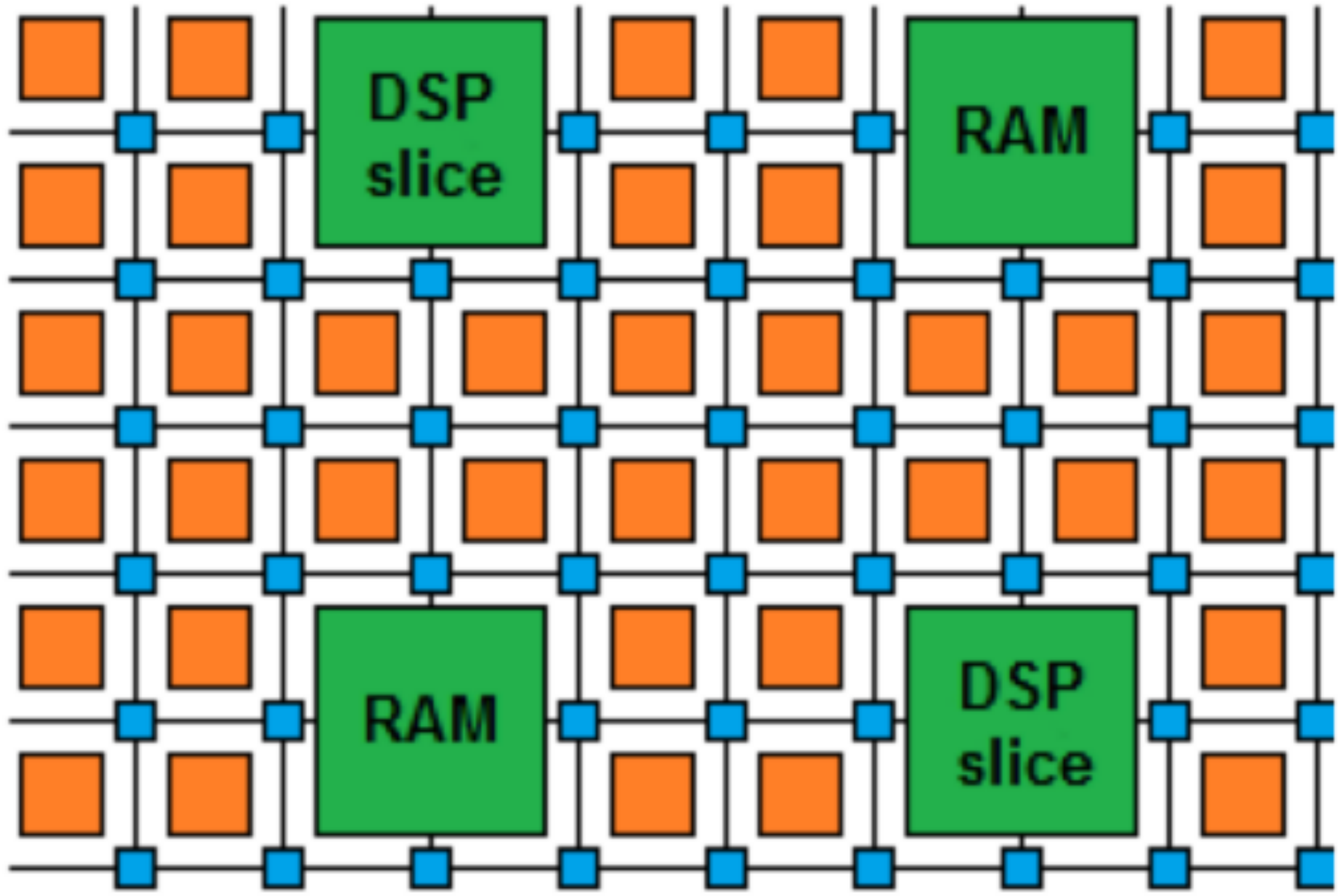
– **Multiplier units (DSPs)** [arithmetic]
– **Look Up Tables (LUTs)** [logic]
– **Flip-flops (FFs)** [registers]
– **Block RAMs (BRAMs)** [memory]

What is an FPGA?

- Run at high frequency - *O(100 MHz)*
  - Can compute outputs in O(ns)

- Low-level Hardware Description Language for programming
  Verilog/VHDL

- Possible to translate C/C++ → Verilog/VHDL using High **Level Synthesis (HLS)** tools

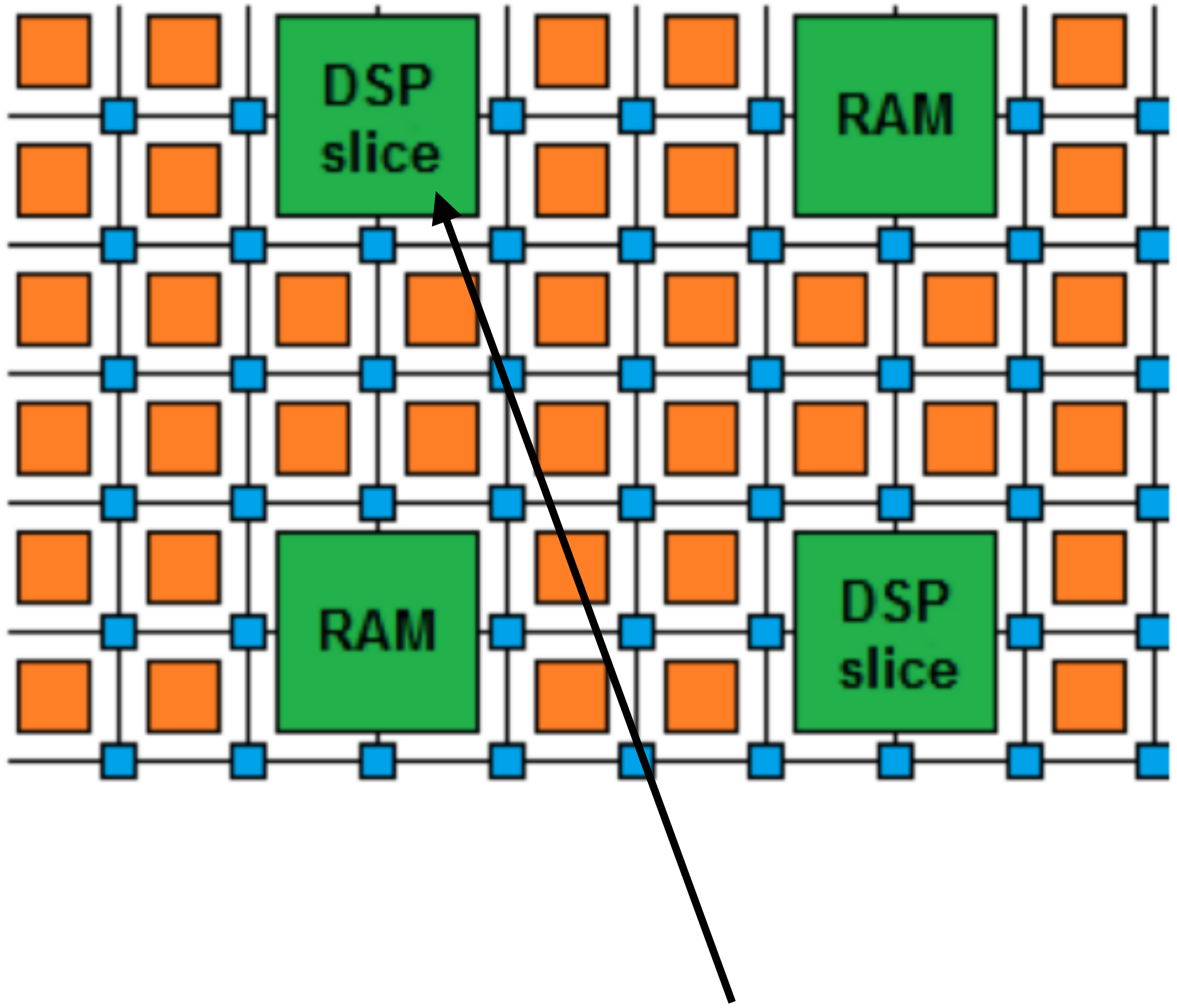


Building blocks:

– **Multiplier units (DSPs)** [arithmetic]
– **Look Up Tables (LUTs)** [logic]
– **Flip-flops (FFs)** [registers]
– **Block RAMs (BRAMs)** [memory]

What is an FPGA?

- **DSPs** (Digital Signal Processor) are specialized units for multiplication and arithmetic
- DSPs are often the most scarce for NNs
- Faster and more efficient than using LUTs for these types of operations



DSP
(multiplication)

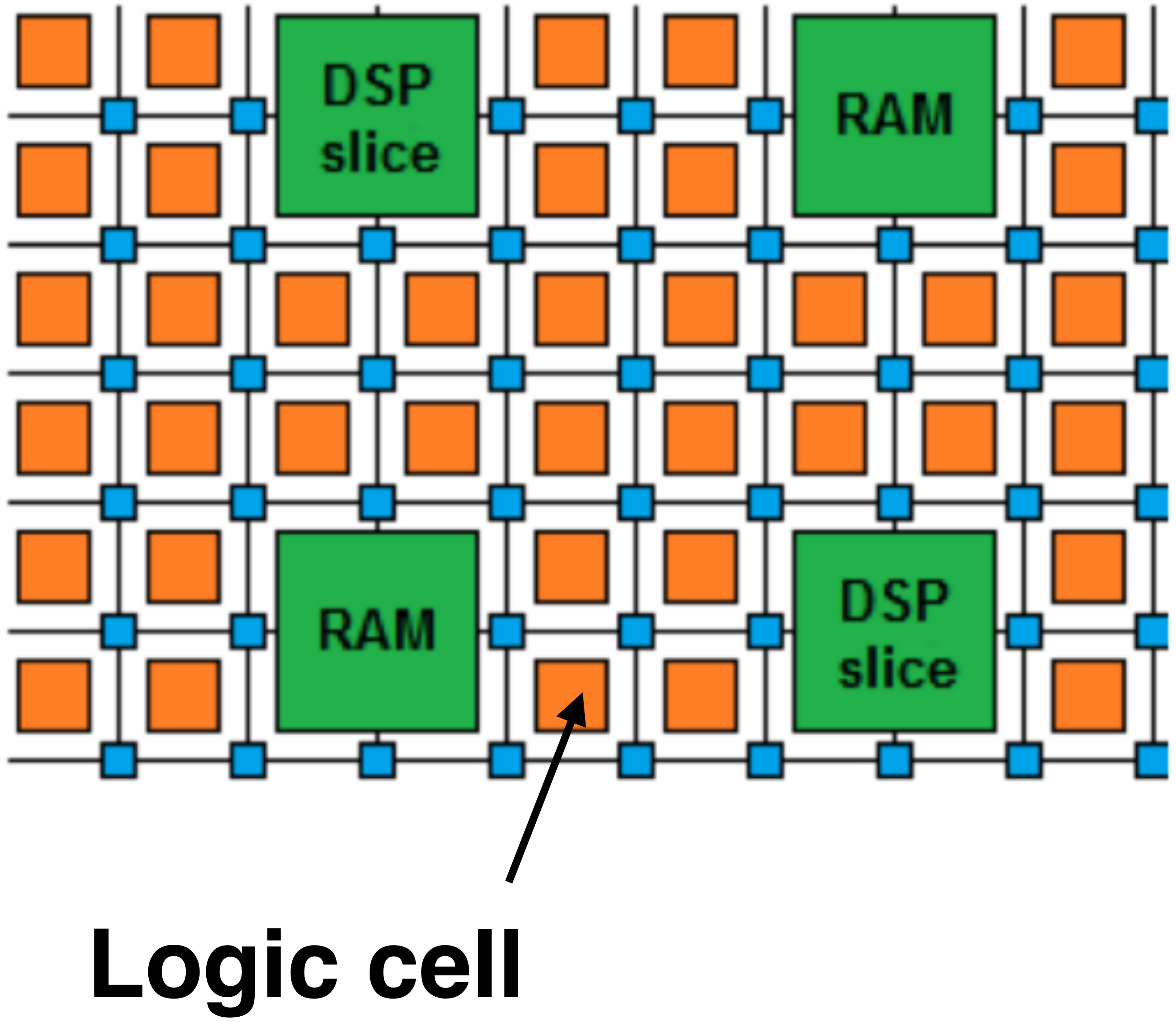
Building blocks:

– **Multiplier units (DSPs)** [arithmetic]
– **Look Up Tables (LUTs)** [logic]
– **Flip-flops (FFs)** [registers]
– **Block RAMs (BRAMs)** [memory]

What is an FPGA?

- **Logic cells / Look Up Tables** perform arbitrary functional operations on small bit-width inputs (2-6)
  - boolean, arithmetic
  - small memories

- **Flip-Flops** control the flow of data with the clock pulse



Building blocks:

– **Multiplier units (DSPs)** [arithmetic]
– **Look Up Tables (LUTs)** [logic]
– **Flip-flops (FFs)** [registers]
– **Block RAMs (BRAMs)** [memory]

Why FPGAs at LHC?

High parallelism ⇧ = **Low latency** ⇩

- Can work on different data simultaneously (pipelining)! High bandwidth

Why FPGAs at LHC?

High parallelism ⇧ = Low latency ⇩

- Can work on different data simultaneously (pipelining)! High bandwidth

Power efficient

- FPGAS ~x10 more power efficient than GPUs

Why FPGAs at LHC?

High parallelism ⇧ = Low latency ⇩

- Can work on different data simultaneously (pipelining)! High bandwidth

Power efficient

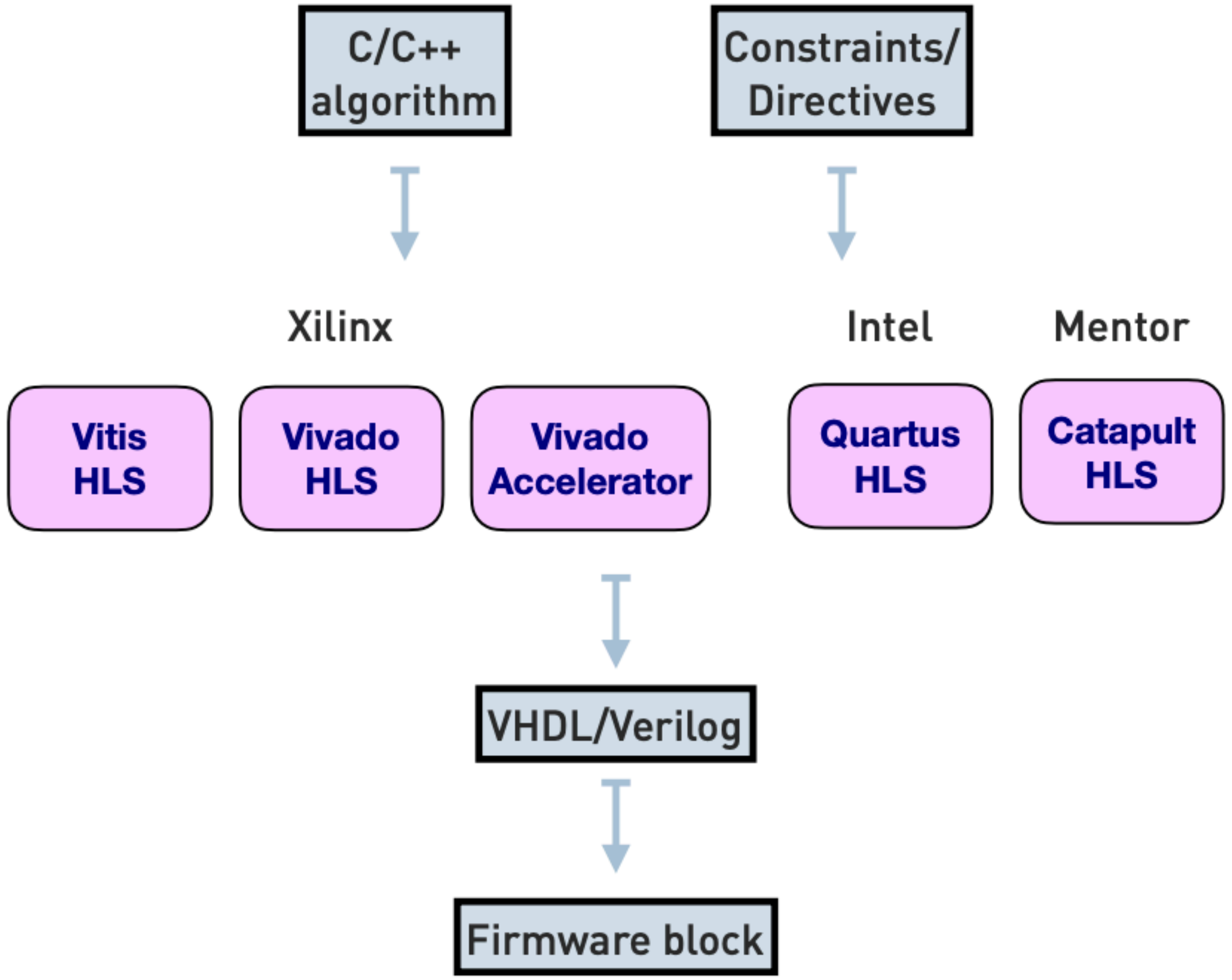
- FPGAS ~x10 more power efficient than GPUs

Latency deterministic

- FPGAs repeatable and predictable latency

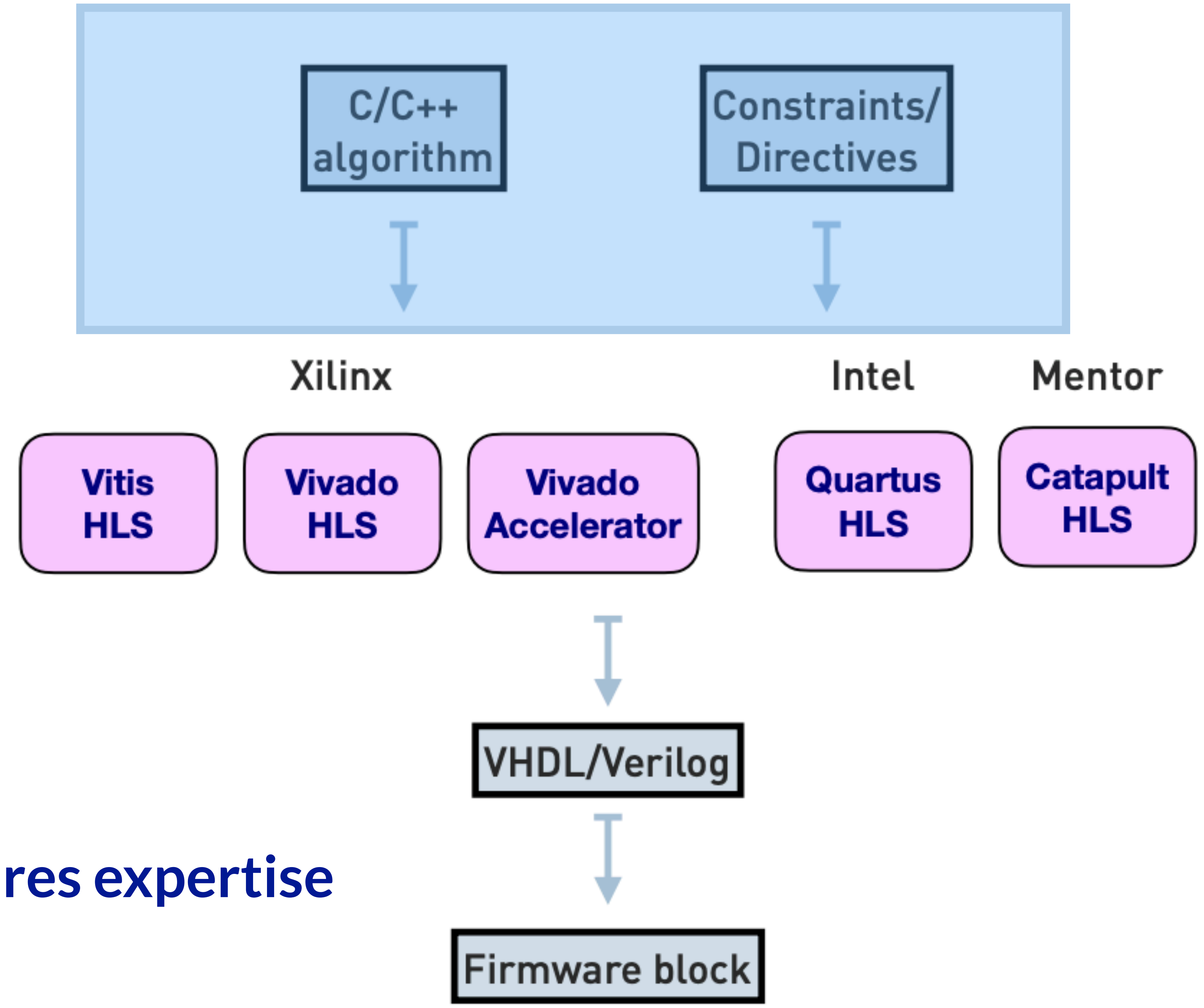
Latency is fixed by proton collisions occurring at 40 MHz, cannot tolerate slack

Programing an FPGA



1. Write C-style code of function
2. Pass to a high-level synthesis (HLS) tool
3. HLS translates to hardware-description language (HDL)
4. Build firmware

Programing an FPGA



1. Write C-style code of function

2. Pass to a high-level synthesis (HLS) tool

3. HLS translates to hardware-description language (HDL)

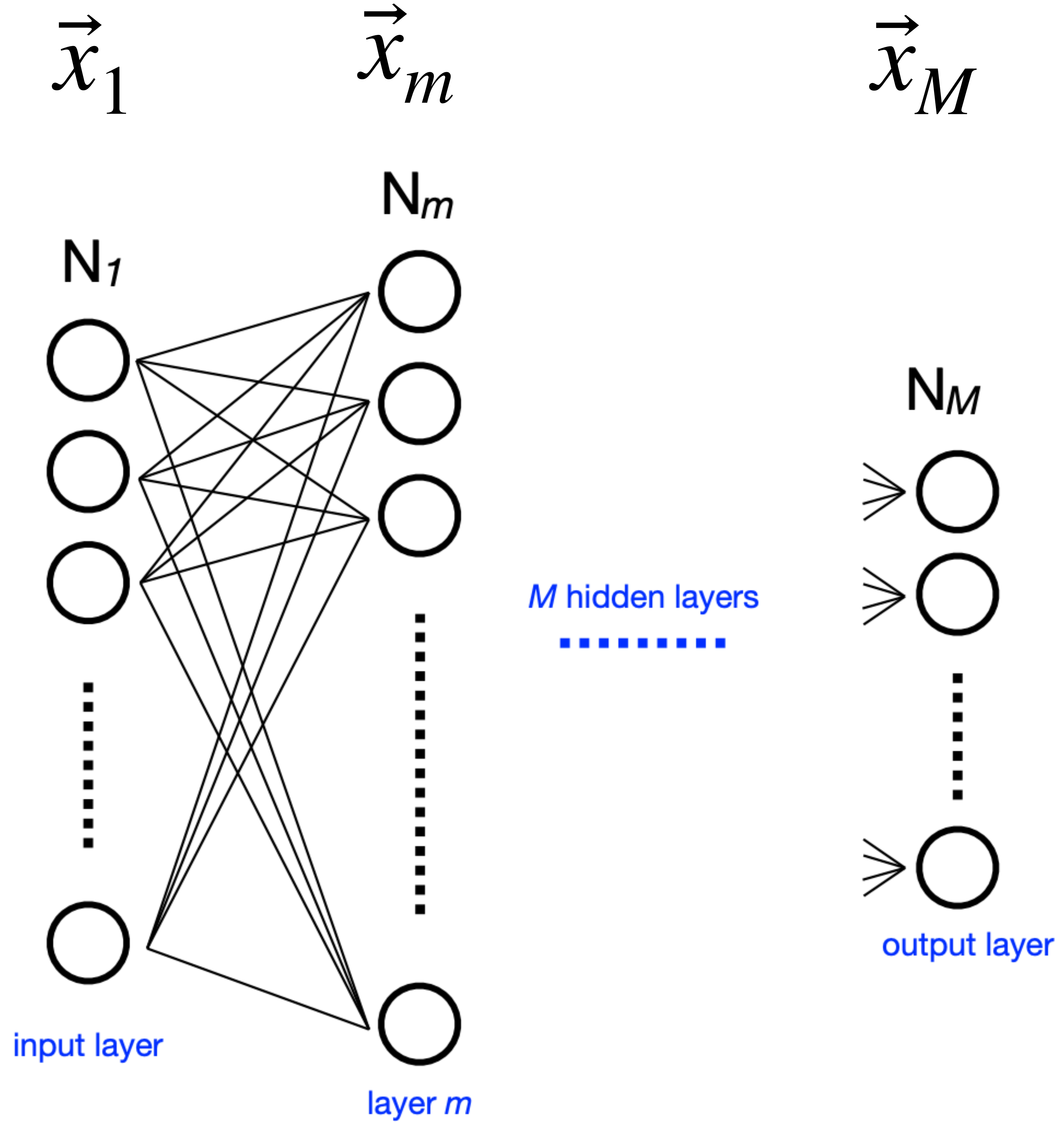
4. Build firmware

Efficient L1T firmware design requires expertise

- FPGA deployment in busy devices
- « 1μs latency target

Not well served by industry tools!

Inference on an FPGA



multiplication

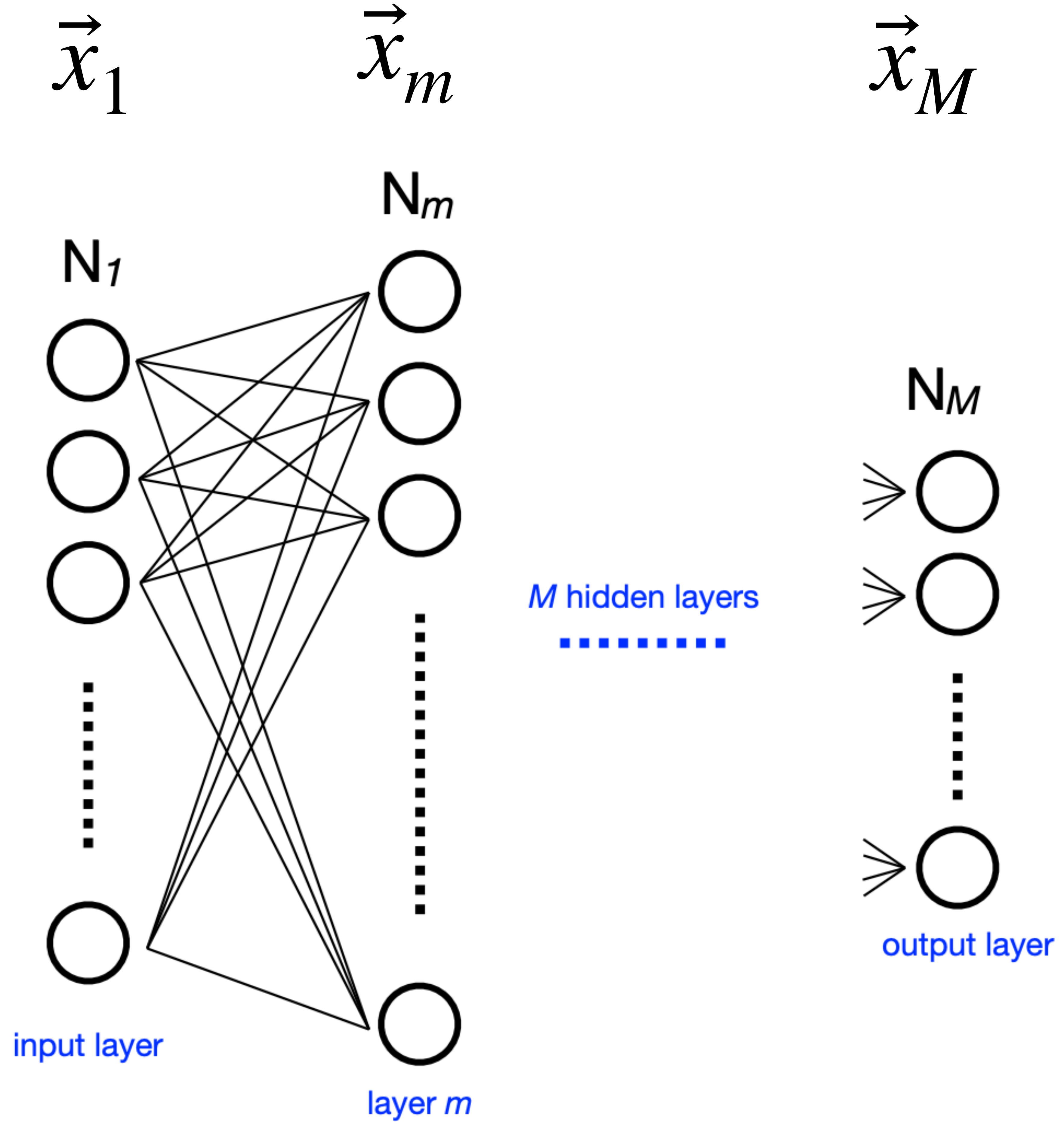
$$\vec{x}_m = g_m\left(W_{m,m-1}\vec{x}_{m-1} + \vec{b}_m\right)$$

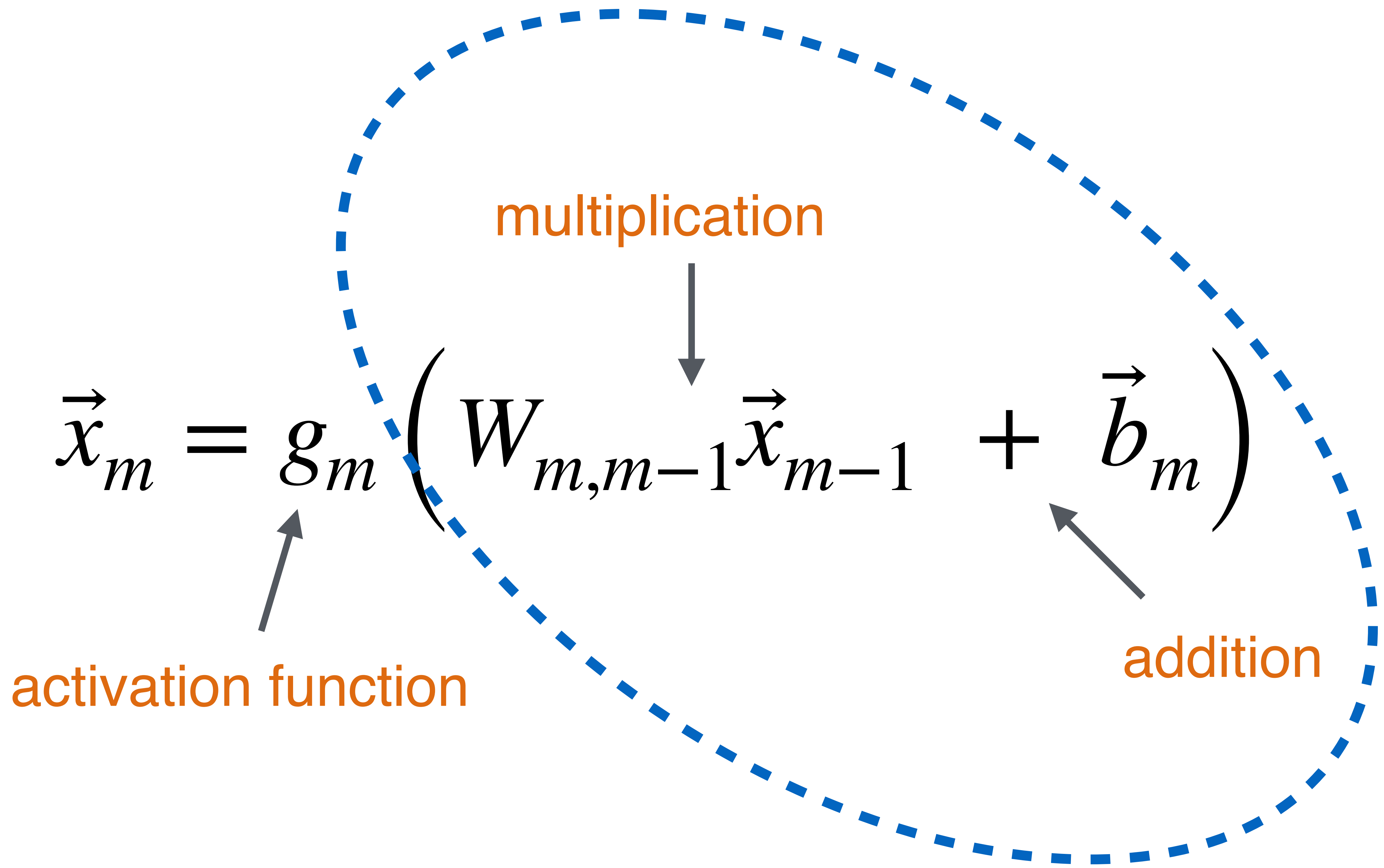
activation function

addition

Credit: Dylan Rankin

Inference on an FPGA





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

multiplication

activation function

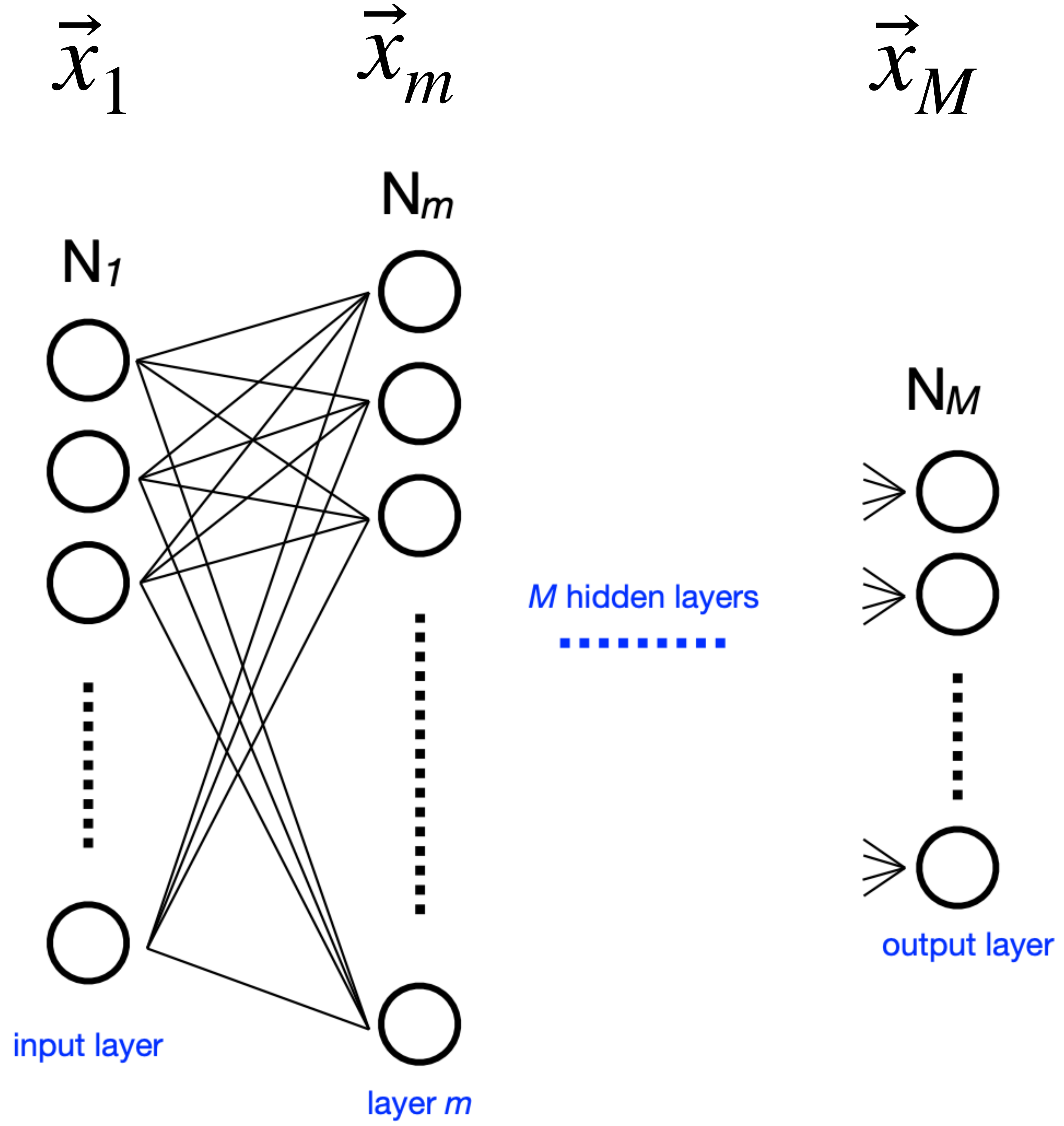
addition

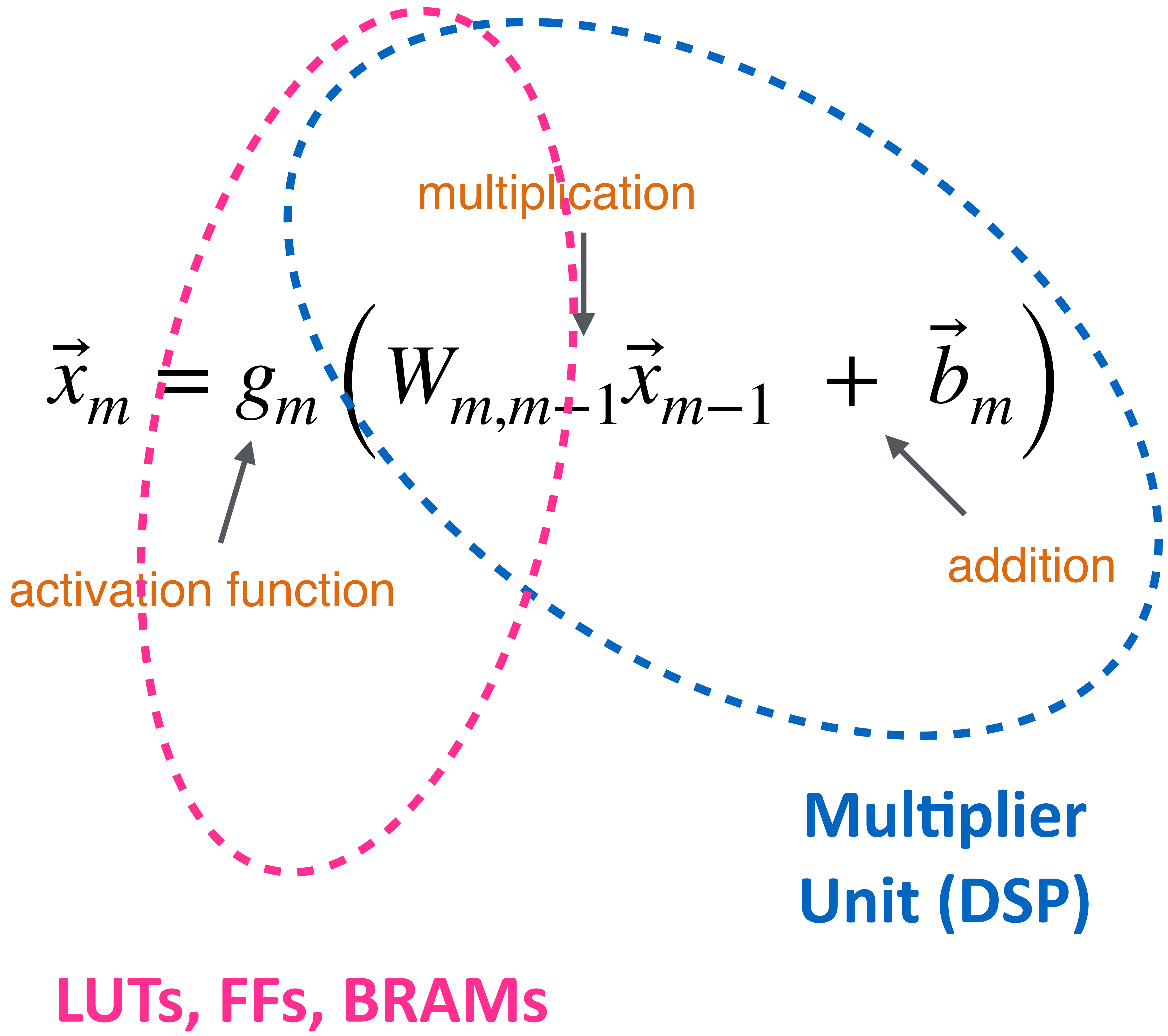
Multiplier Unit (DSP)

up to ~6k parallel operation (VU9P)

Credit: Dylan Rankin

Inference on an FPGA





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

Multiplier Unit (DSP)

LUTs, FFs, BRAMs

Credit: Dylan Rankin

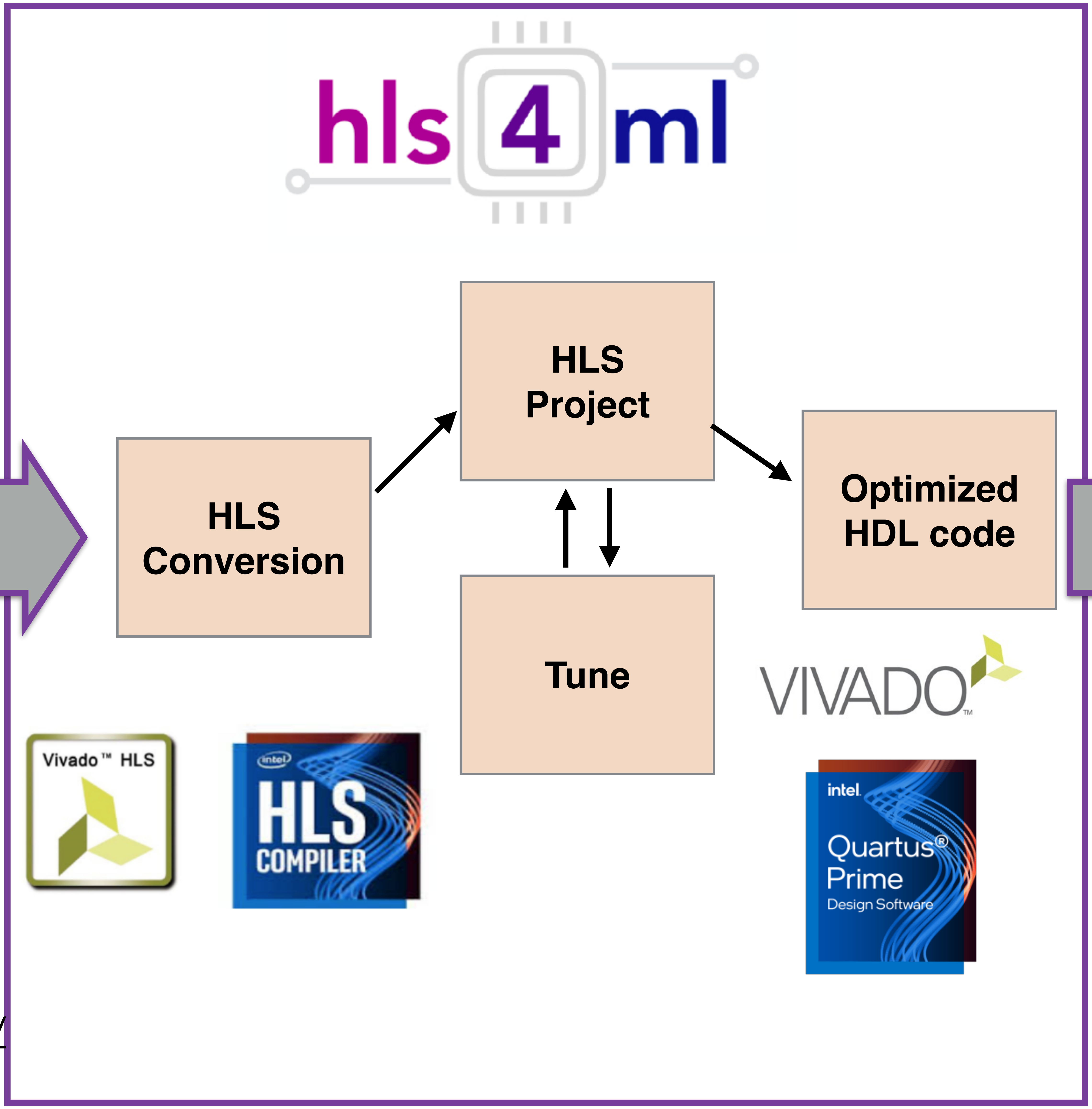
High Level Synthesis with Machine Learning (hls4ml)

Usual ML software workflow

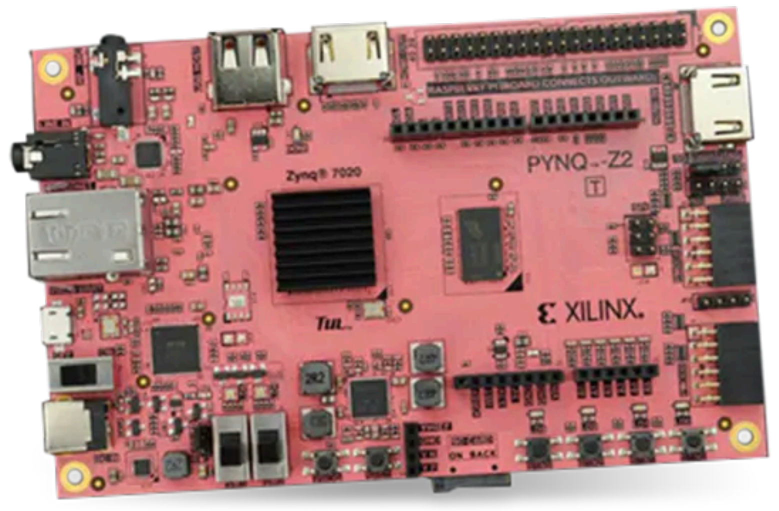
(Q) K + TensorFlow

(Q) ONNX

PYTORCH



Bit Files for FPGA



https://fastmachinelearning.org/hls4ml/
arXiv:2103.05579

High Level Synthesis with Machine Learning (hls4ml)

https://fastmachinelearning.org/hls4ml/
arXiv:2103.05579

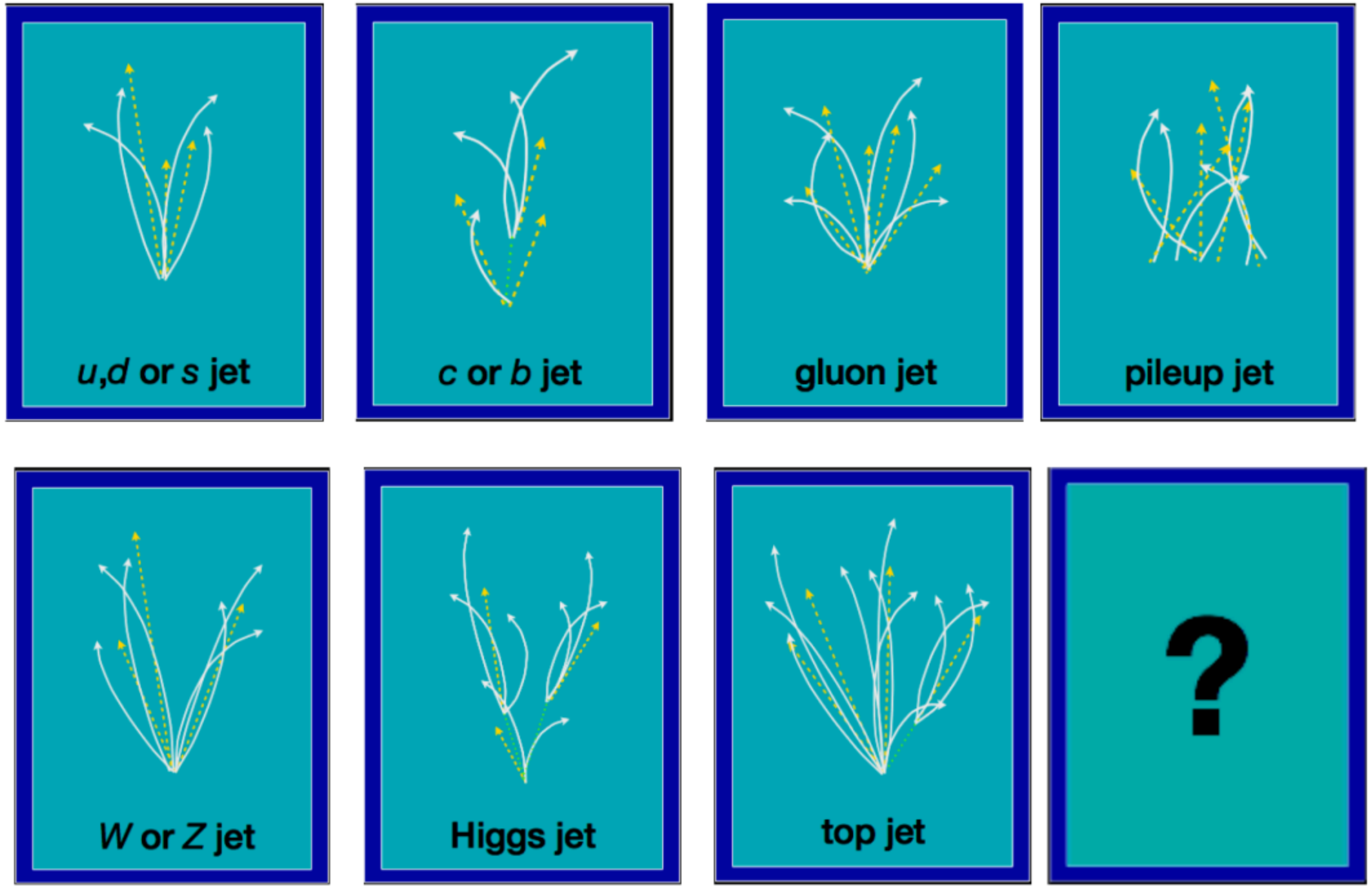
A software interface for implementing Neural Networks on an FPAG

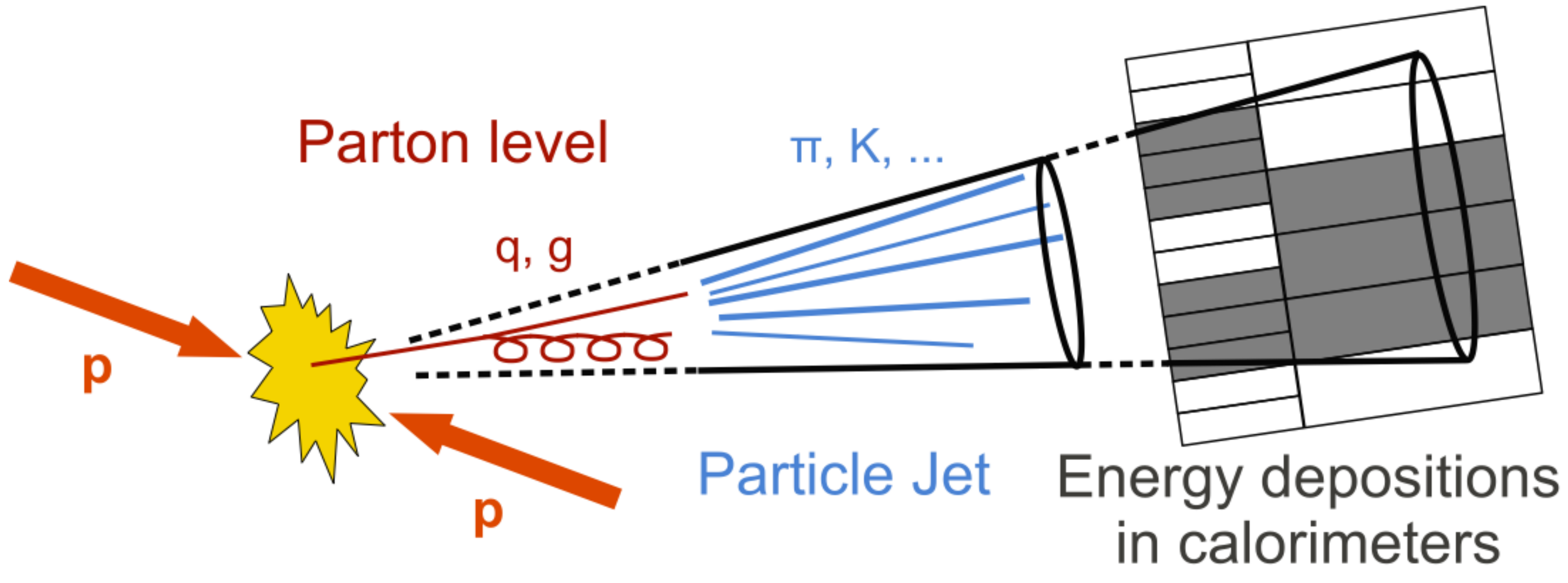
- Supports many common layer like DNN, CNN, etc
- Recursive Neural Networks were not implemented until late 2022

RNN-based algorithms could be deployed at the Level-0 trigger
Example:

- Tau-particle identification
- Missing Transverse Energy reconstruction

Example: Jet Classification



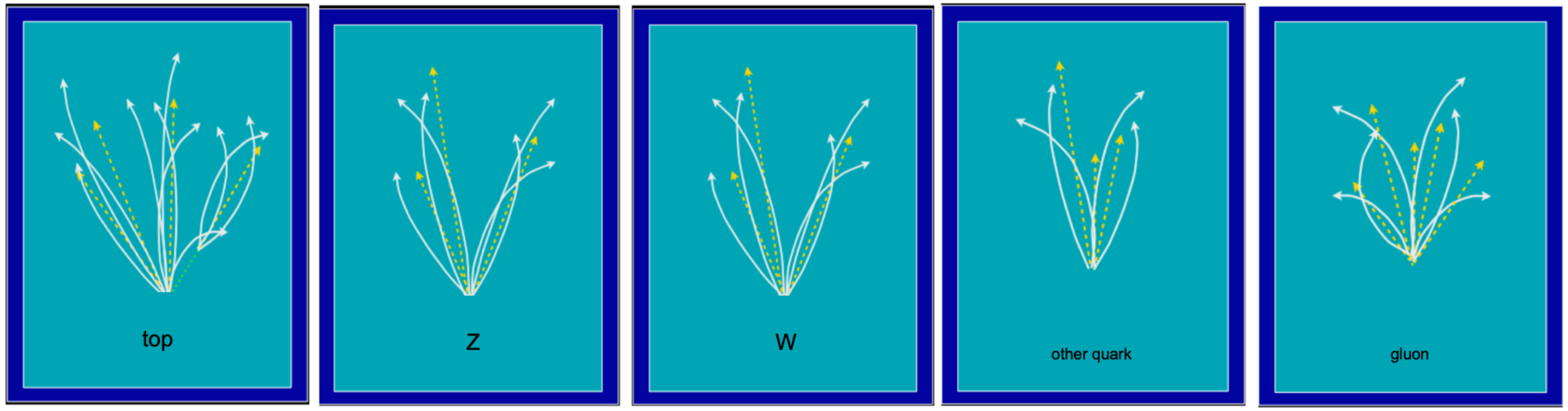


Perhaps an unrealistic example for L1 trigger, but lessons are useful

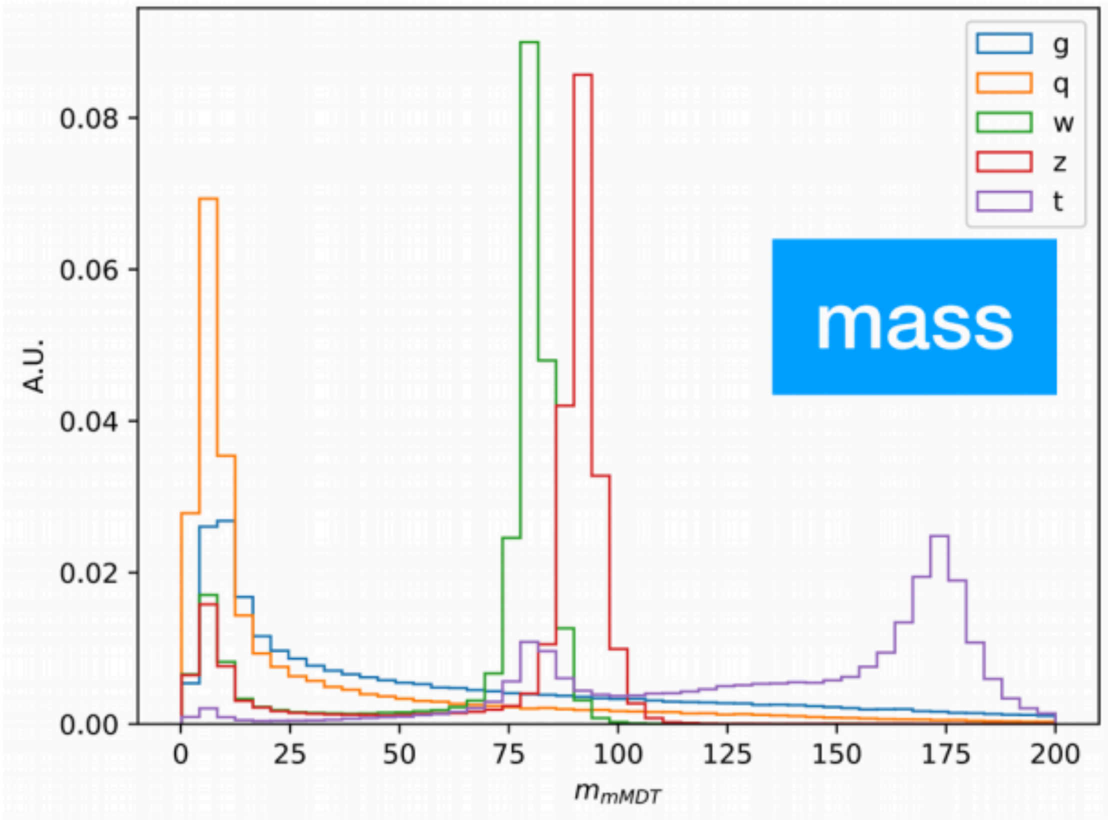
Jet Classification: 5-class classifier

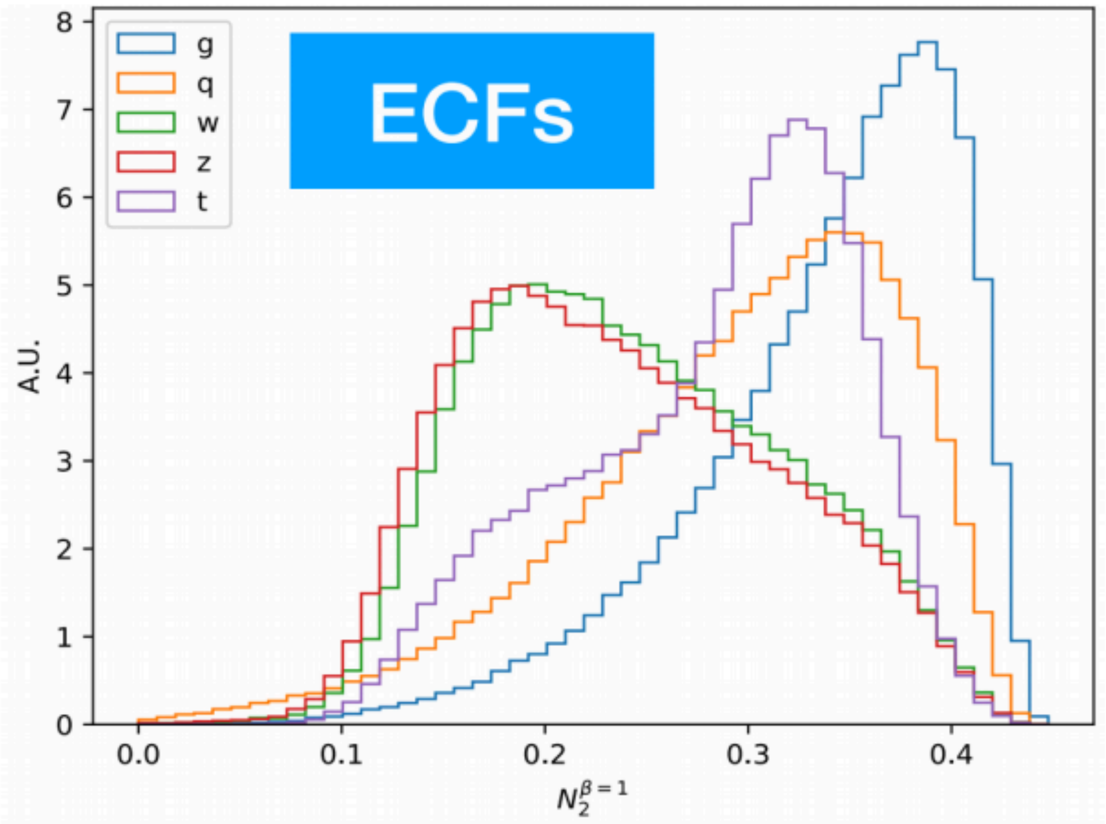
Five class classifier

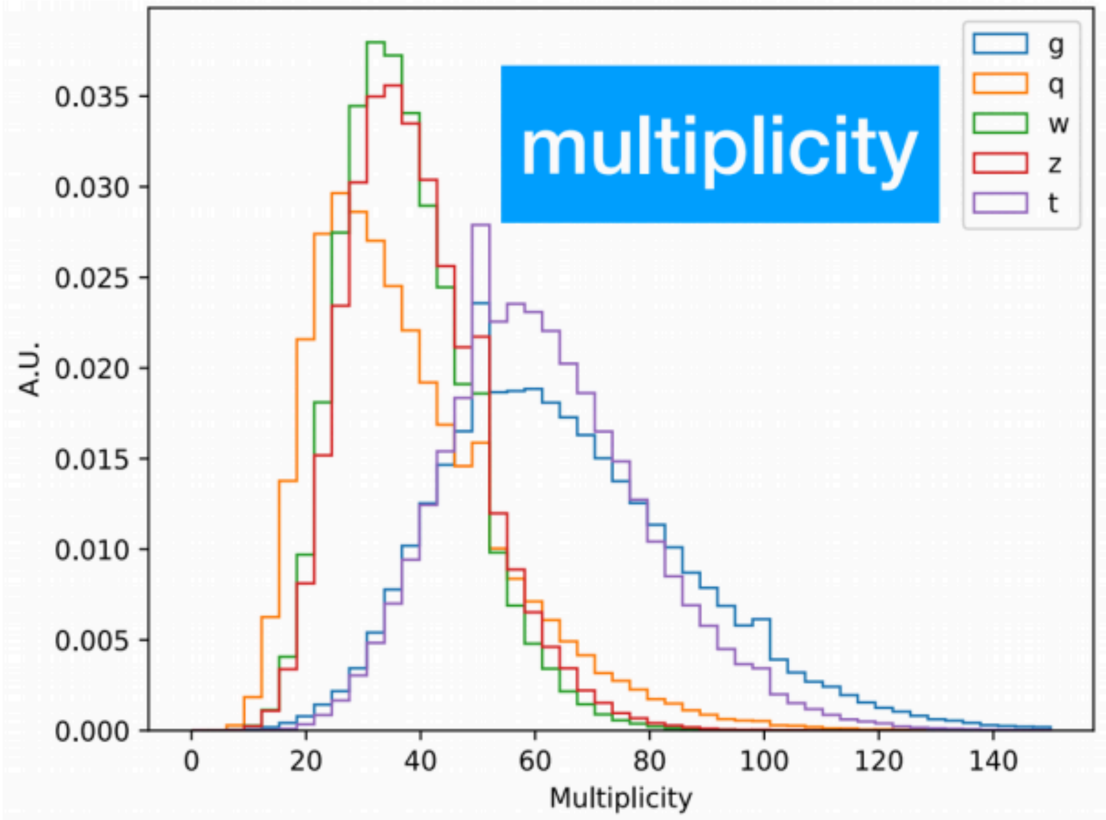
Sample: ~ 1M events with two boosted WW/ZZ/tt/qq/gg anti-kT jets



t→bW→bqq Z→qq W→qq q/g background







Observables

m_{mMDT}

$N_2^{\beta=1,2}$

$M_2^{\beta=1,2}$

$C_1^{\beta=0,1,2}$

$C_2^{\beta=1,2}$

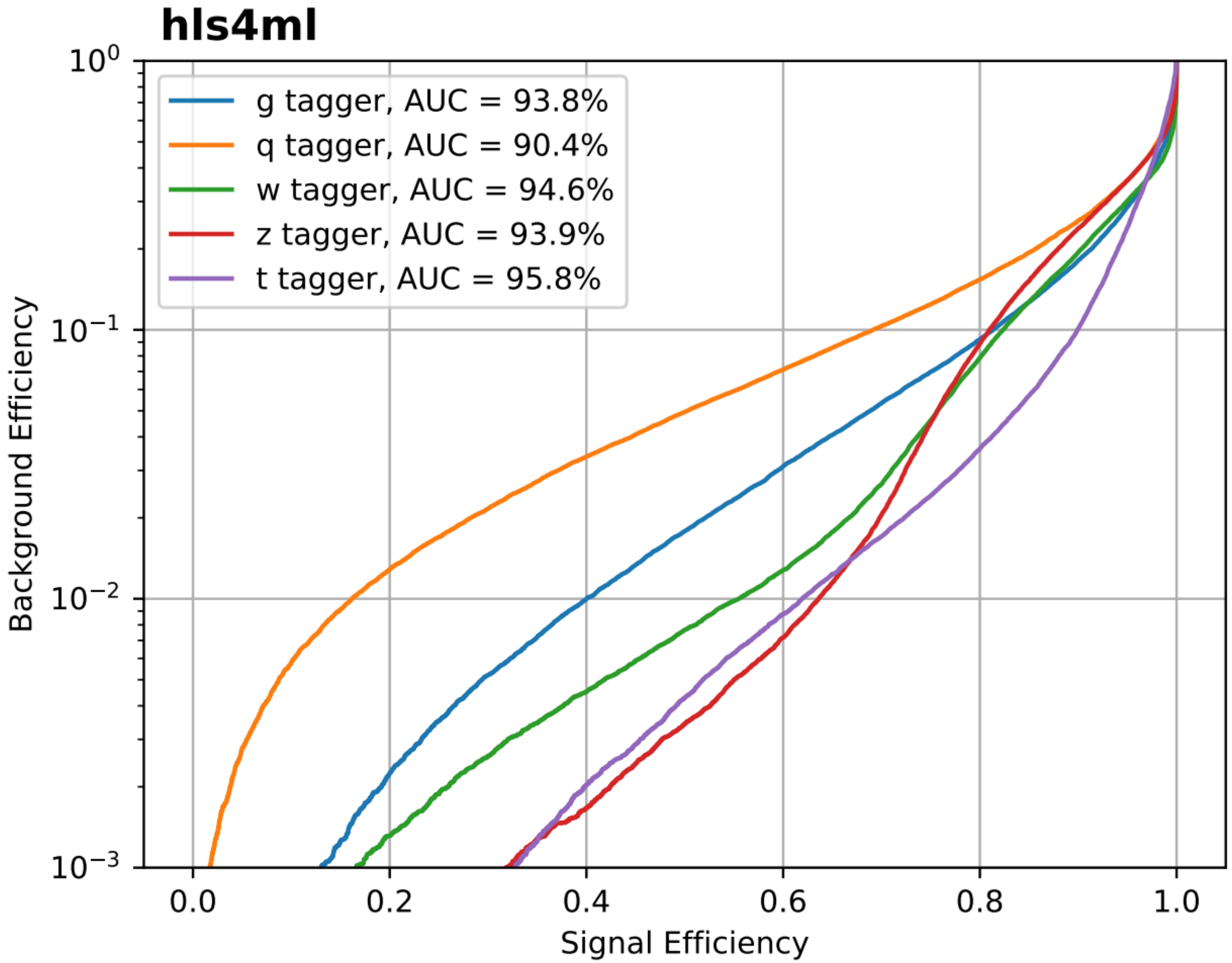
$D_2^{\beta=1,2}$

$D_2^{(\alpha,\beta)=(1,1),(1,2)}$

$\sum z \log z$

Multiplicity

Jet-tagging ROC



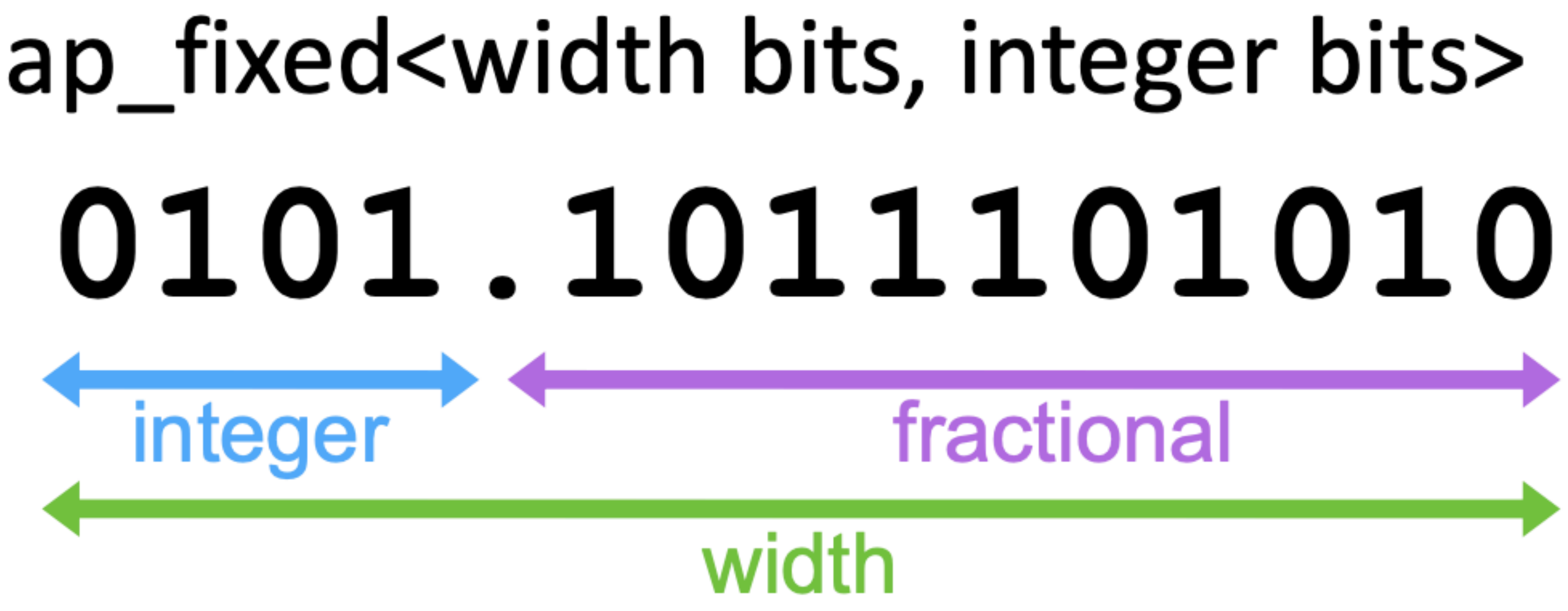
Quantization

Quantization – Reducing the bit precision used for NN arithmetic

Why this is necessary?

- Floating-point operations (32 bit numbers) on an FPGA consumes large resources
- Not necessary to do it for desired performance

- **hls4ml uses fixed-point representation for all computations**
  - Operations are integer ops, but we can represent fractional values



ap_fixed<width bits, integer bits>

Quantization Strategies

Usual ML software workflow

(Q) K + TensorFlow

(Q) ONNX

PYTORCH

Post Training Quantization

Use hls4ml package to find optimal representation

Initial Model

Usual ML software workflow

(Q) K + TensorFlow

(Q) ONNX

PYTORCH

Quantization-Aware Training

- QKeras
- PyTorch (limited options)
- TensorFlow (limited options)
- QONNX (in development)

Let's try it out!

Lets start the Jupyterhub following the instructions:

https://github.com/usatlas-ml-training/lbnl-2023/tree/main/hls4ml_tutorial

Start your Jupyterhub

Note it is a different jupyterhub compared to the other days

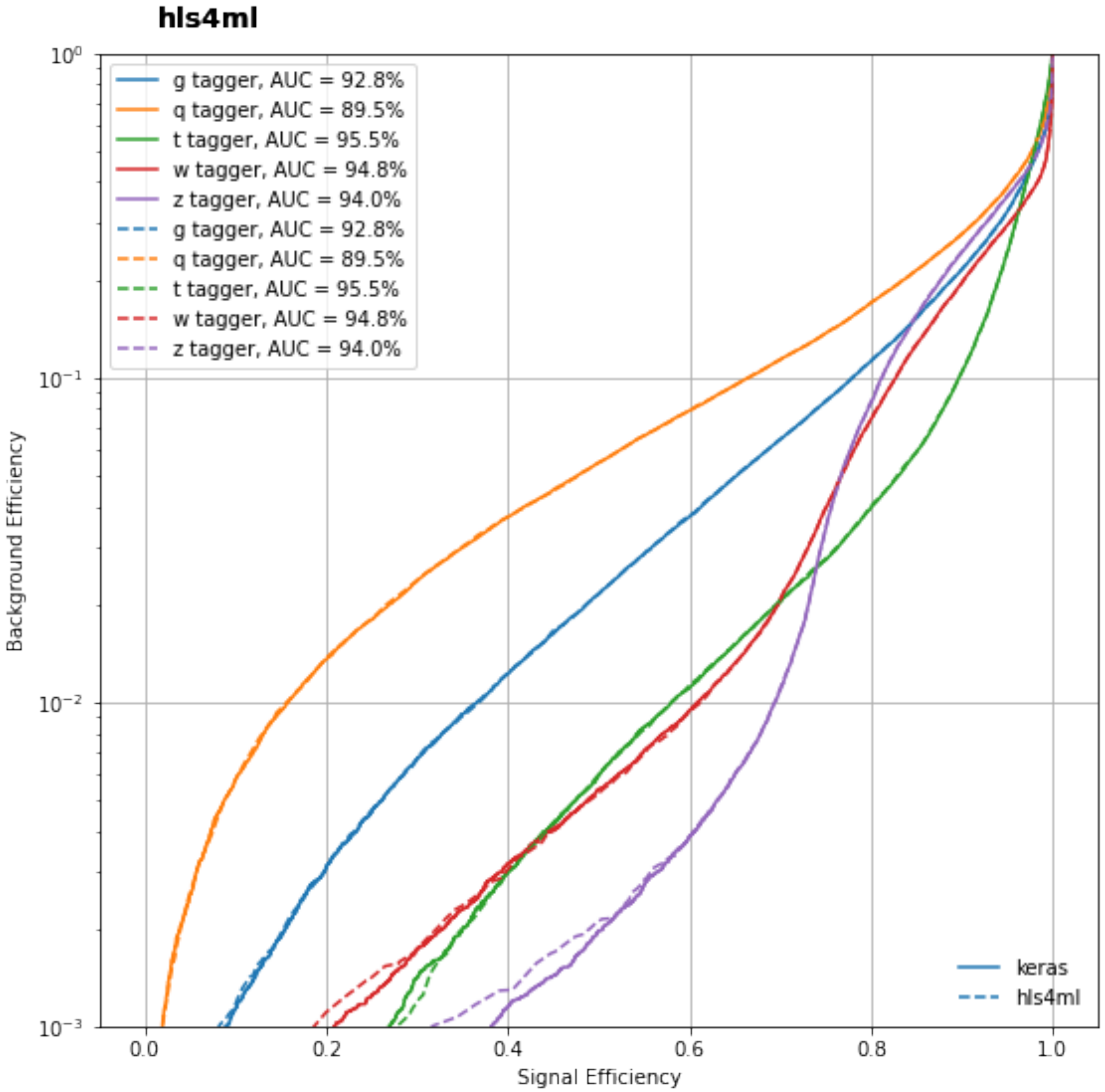
Checkout the tutorial repo: https://github.com/usatlas-ml-training/lbnl-2023.git

Quantization Demo

https://github.com/usatlas-ml-training/lbnl-2023/blob/main/hls4ml_tutorial/PTQ_demo.ipynb

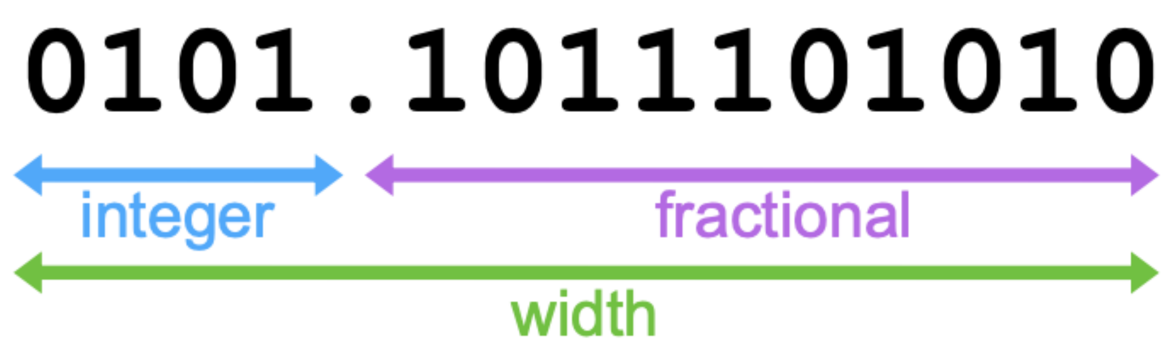
Jet-tagging ROC: Post Quantization

Used precision: <16,6>
Integet bits: 6
fractional bits: 10



Scan to find optimal precision

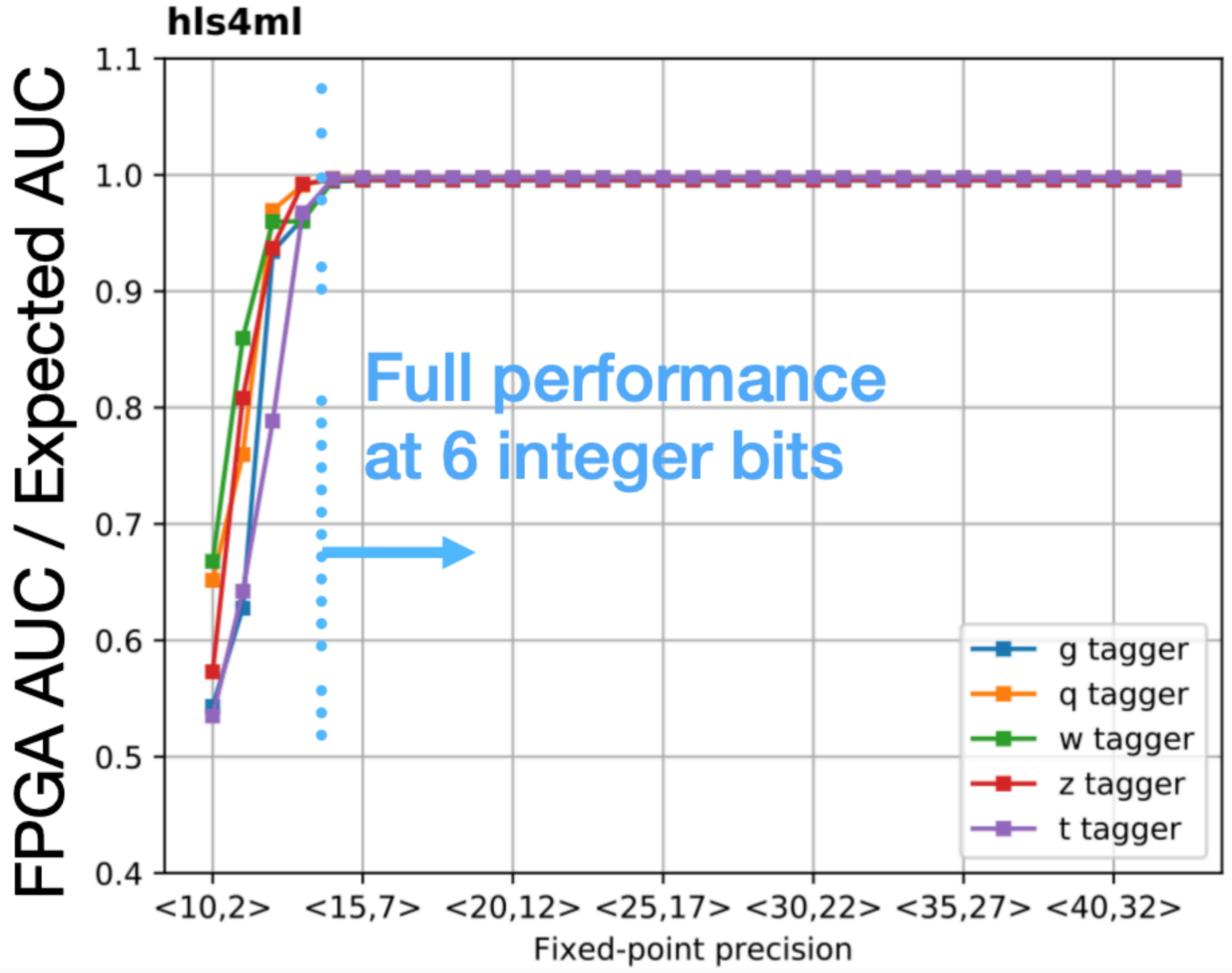
ap_fixed<width bits, integer bits>



0101.1011101010

Scan integer bits

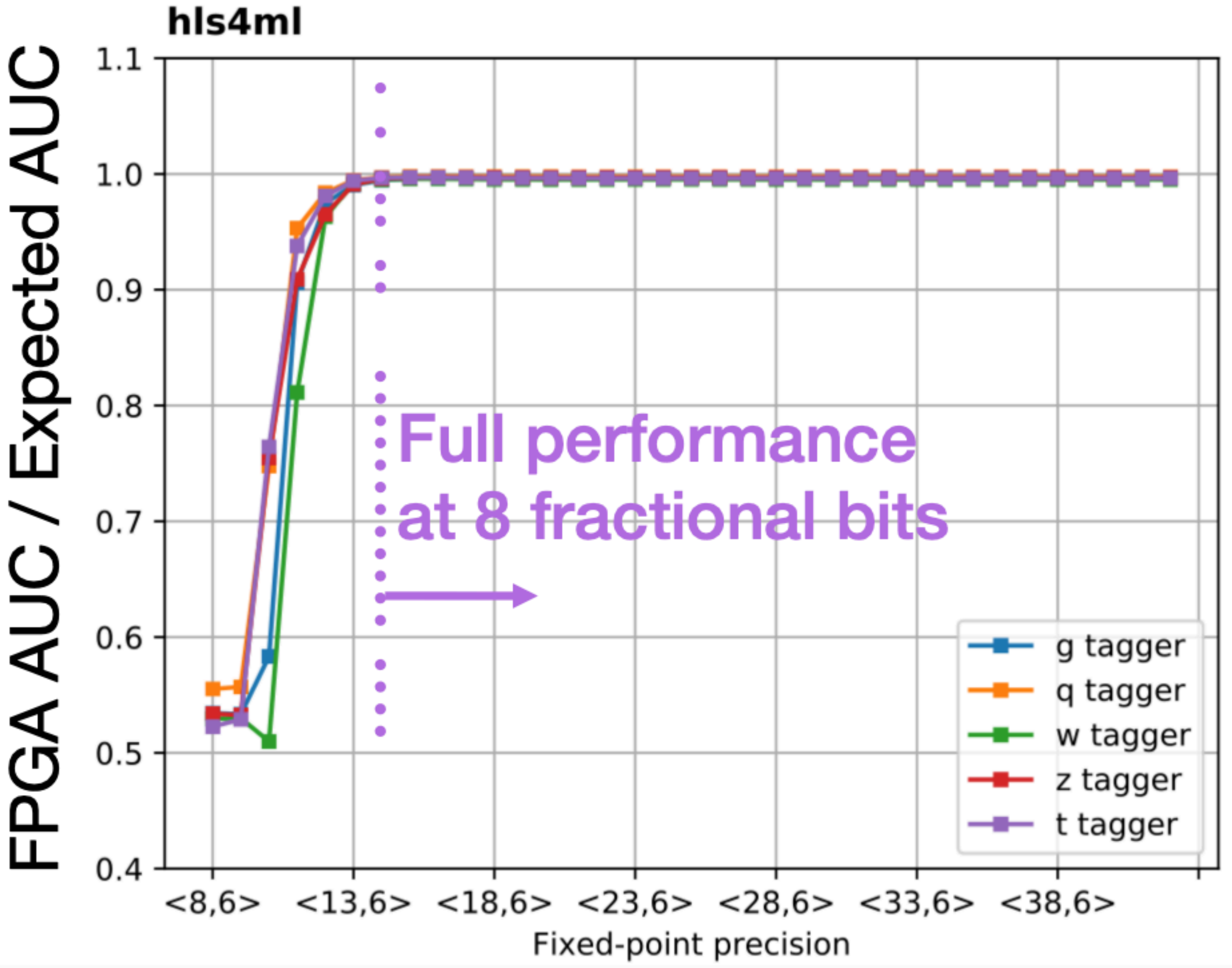
Fractional bits fixed to 8



Full performance at 6 integer bits

Scan fractional bits

Integer bits fixed to 6

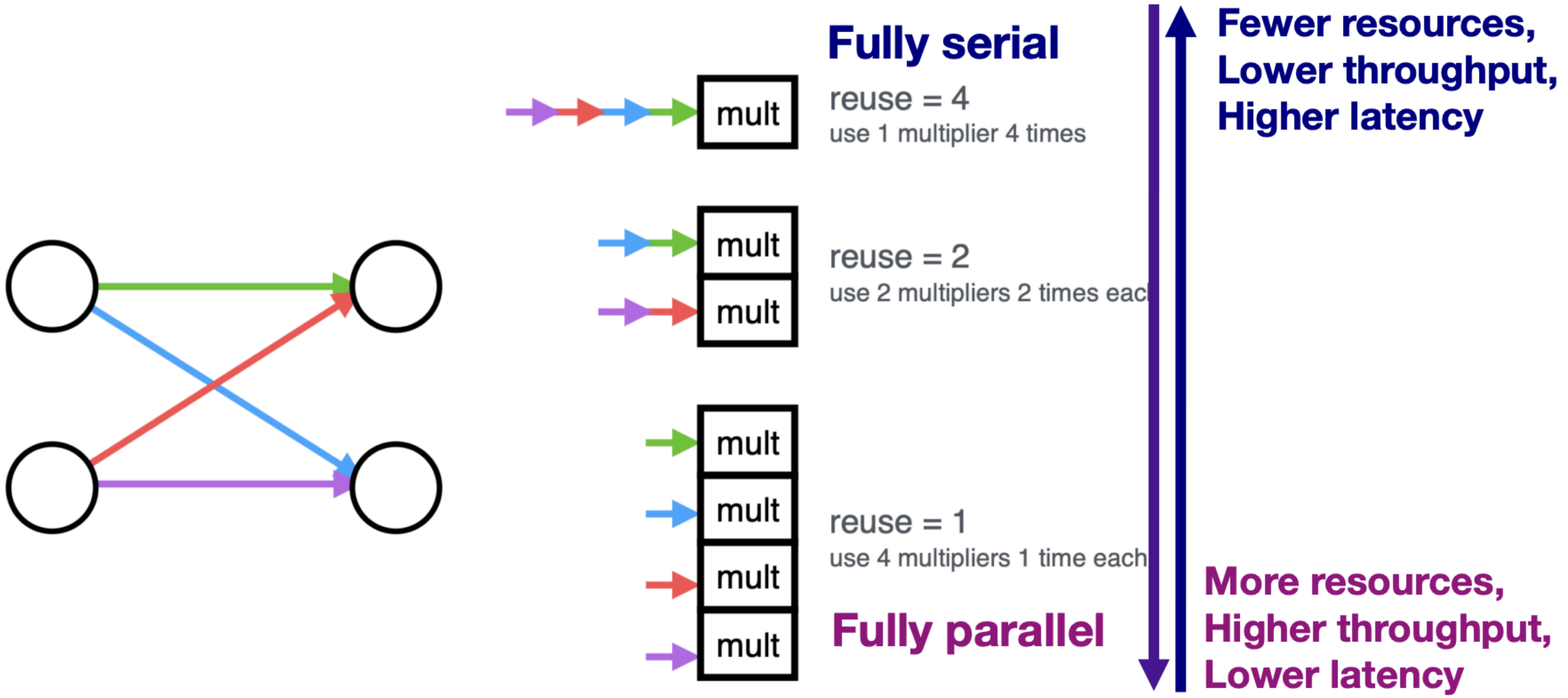


Full performance at 8 fractional bits

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



Fully serial

reuse = 4
use 1 multiplier 4 times

reuse = 2
use 2 multipliers 2 times each

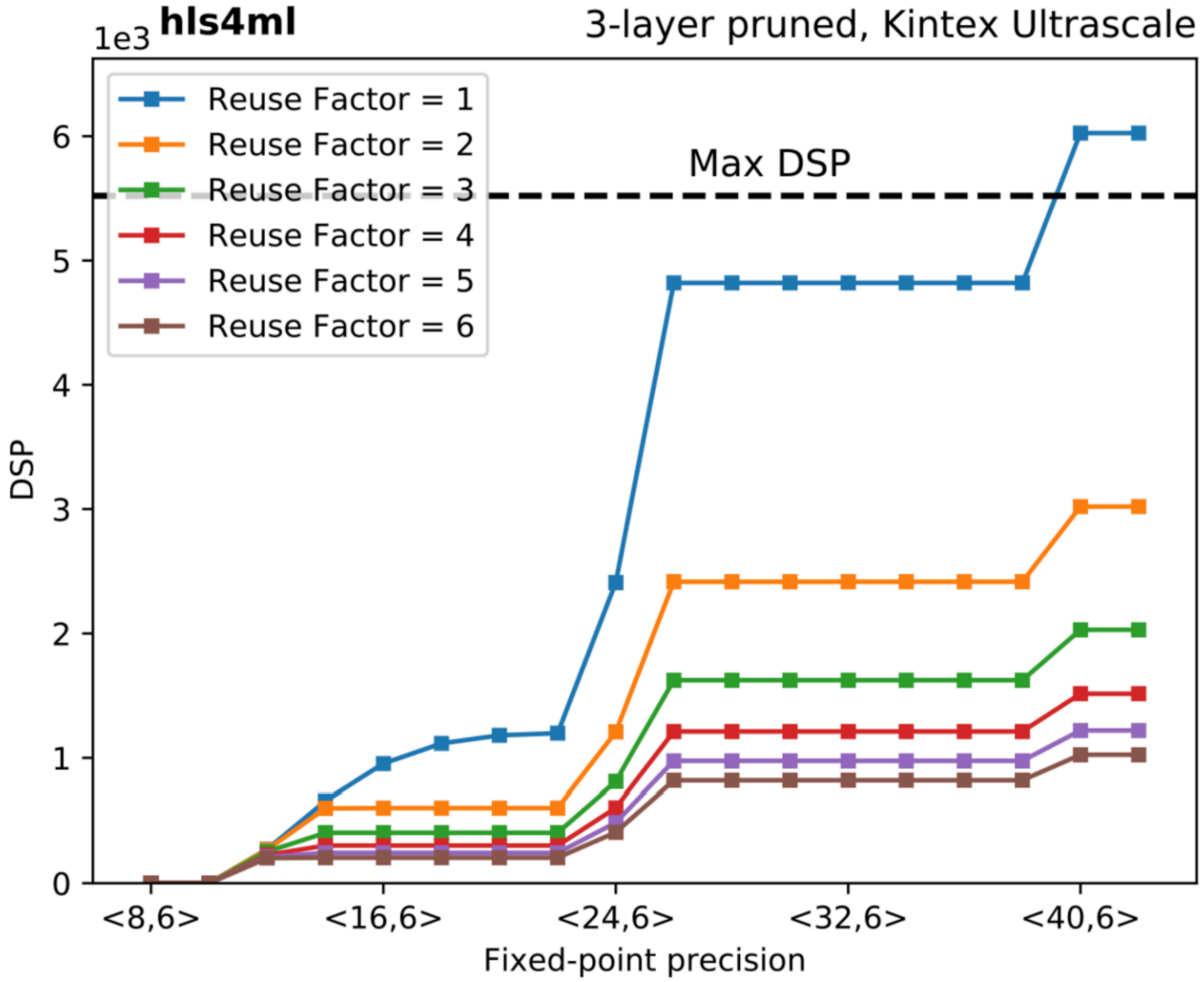
reuse = 1
use 4 multipliers 1 time each

Fully parallel

Fewer resources,
Lower throughput,
Higher latency

More resources,
Higher throughput,
Lower latency

(Example) 5-class jet-tagging: DSP usage



Fully parallel
Each mult. used 1x

Each mult. used 2x

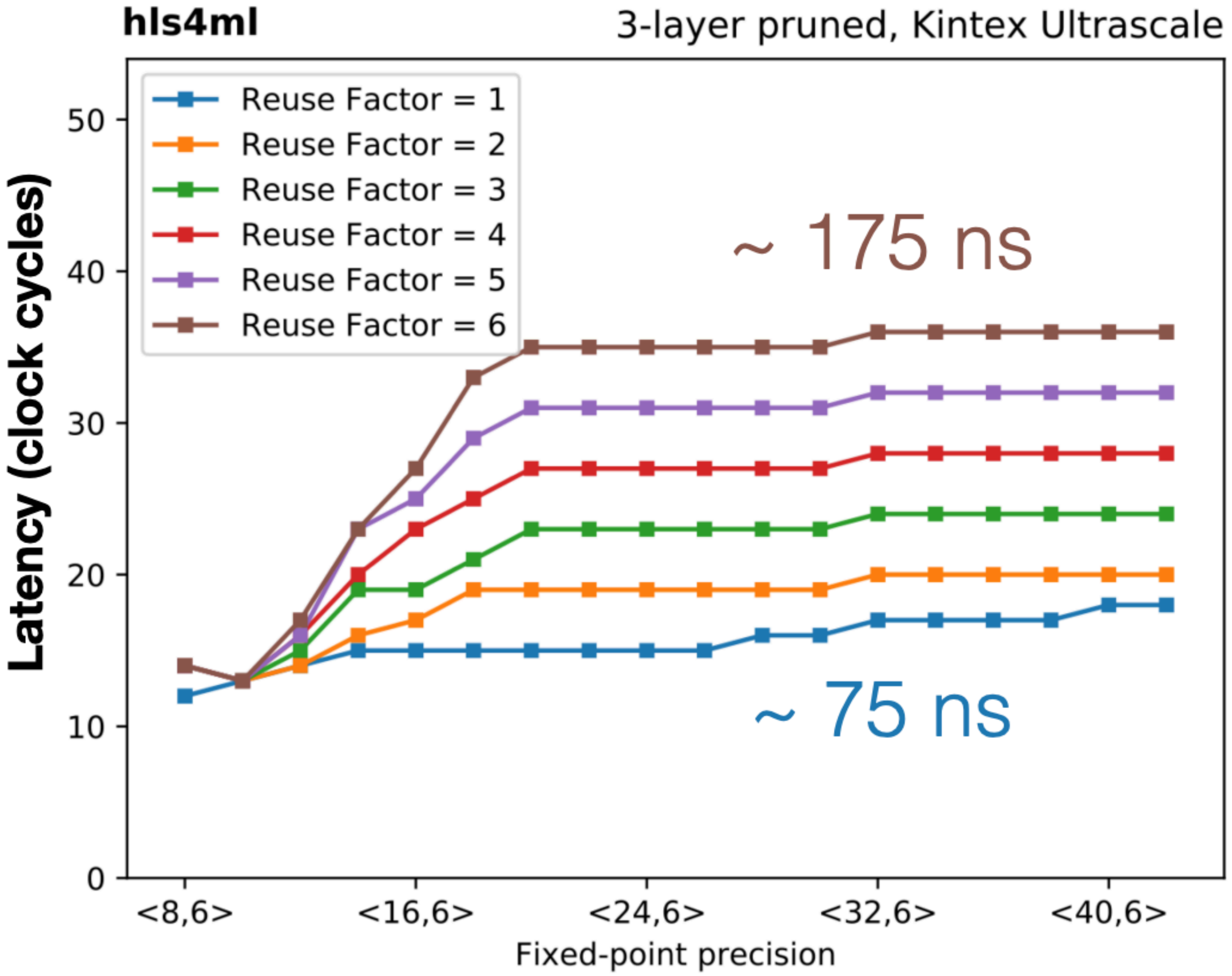
Each mult. used 3x

⋮

More resources

Longer latency

(Example) 5-class jet-tagging: Timing



Longer latency

Each mult. used 6x

⋮

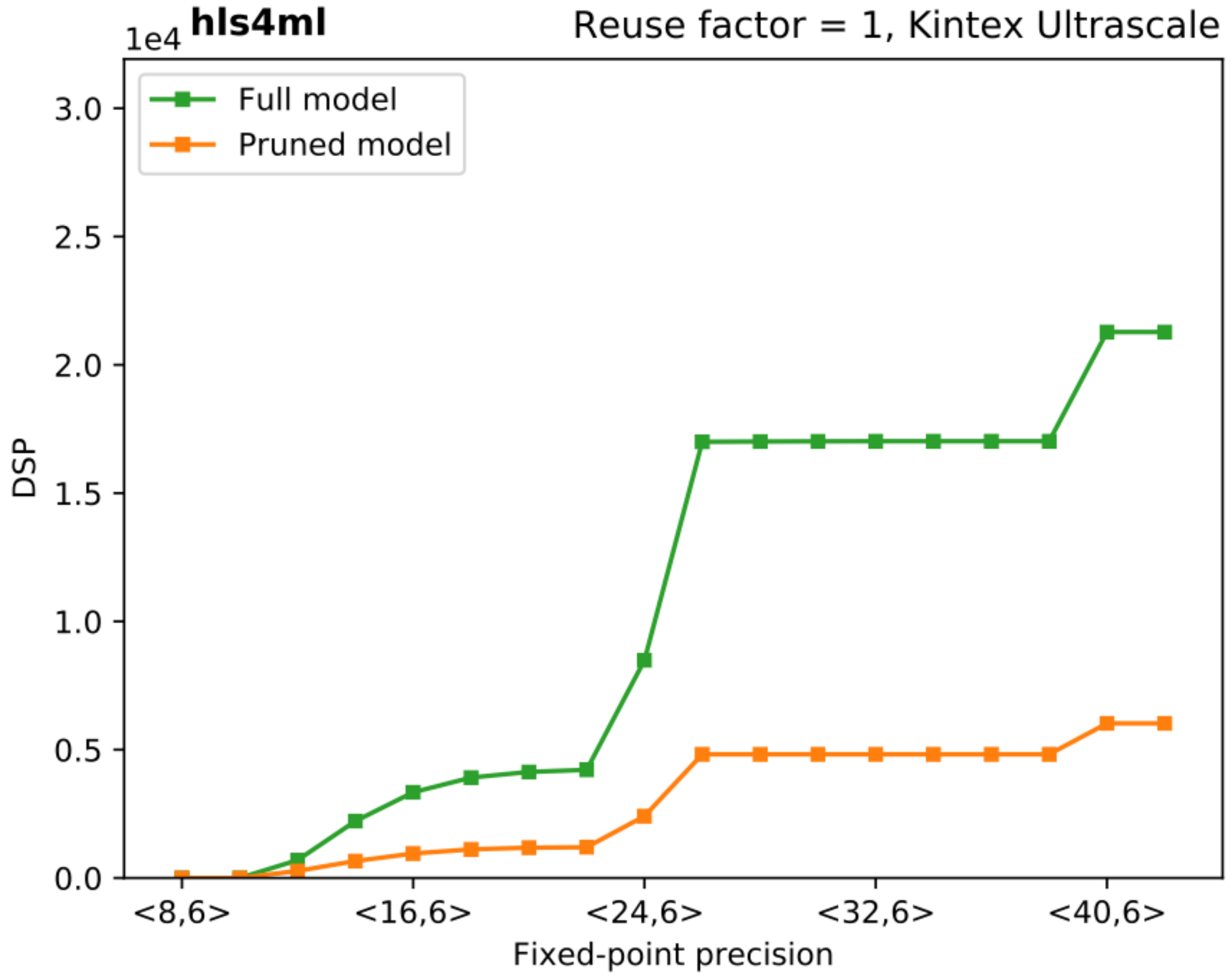
Each mult. used 3x

⋮

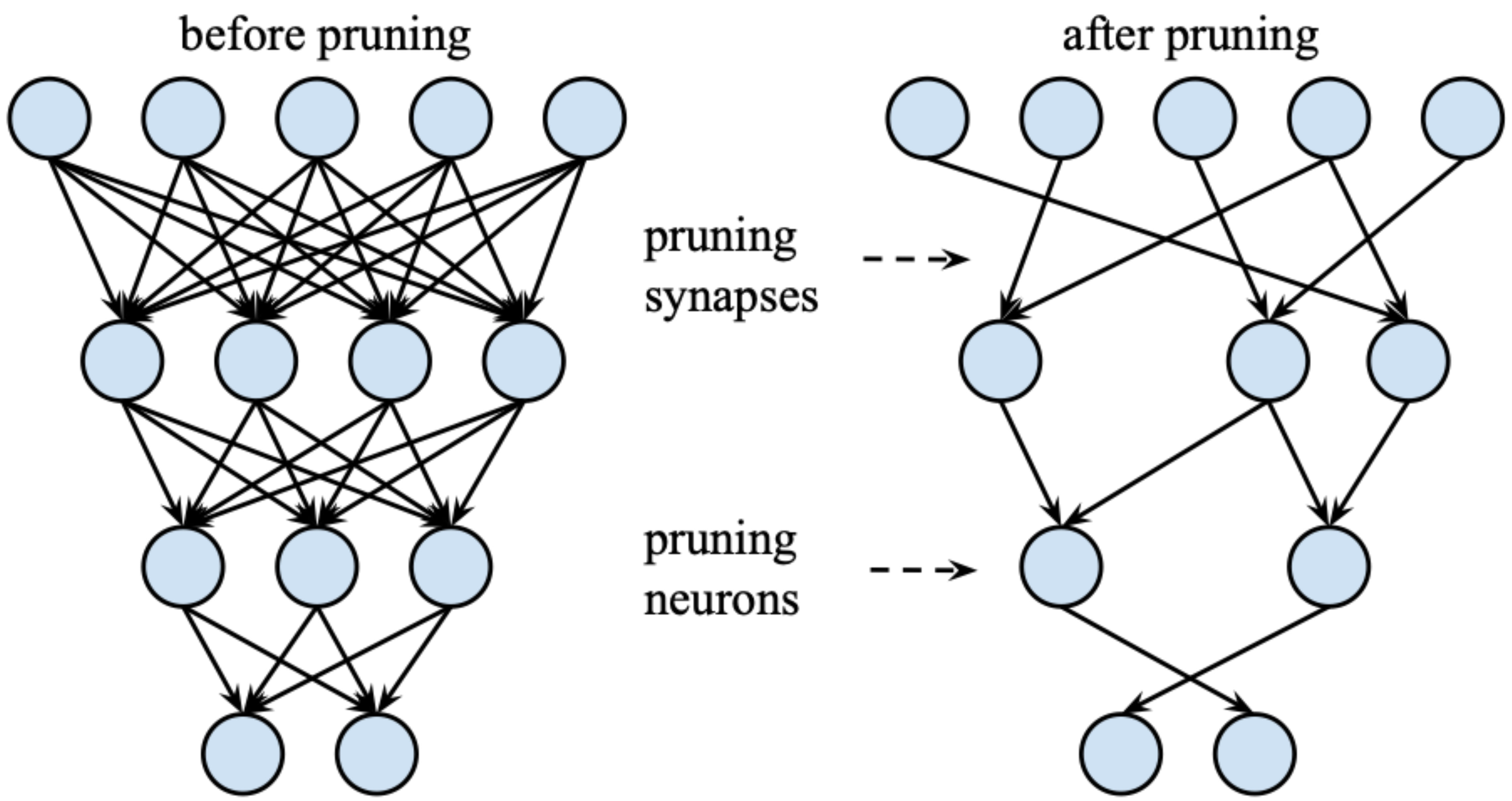
Fully parallel
Each mult. used 1x

More resources

Model Compression via Pruning



70% compression ~ 70% fewer DSPs

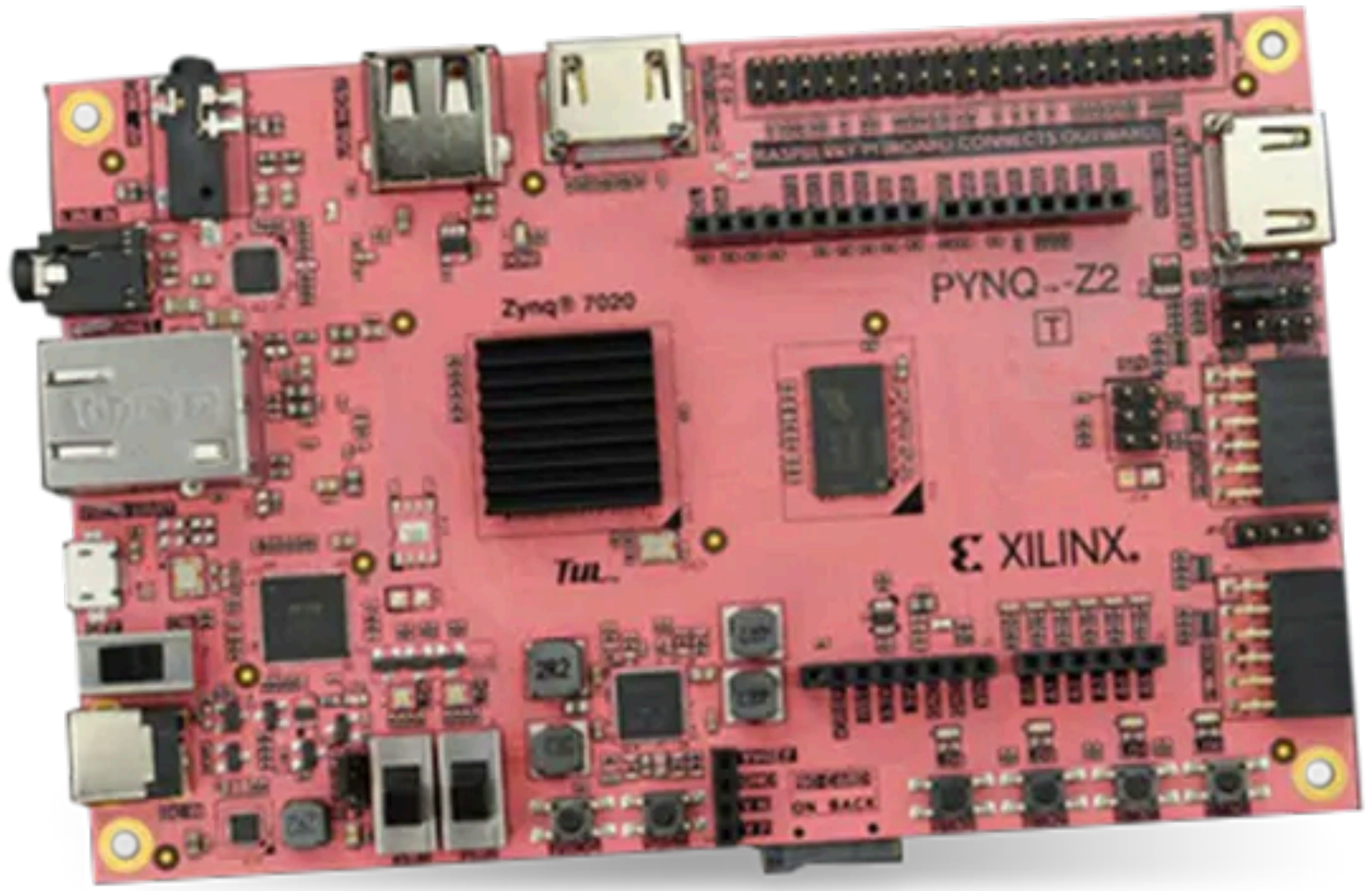


- DSPs (used for multiplication) are often limiting resource
  - maximum use when fully parallelized
  - DSPs have a max size for input (e.g. 27x18 bits), so number of DSPs per multiplication changes with precision

Summary

- Some of the most challenging problems in HEP exist in the trigger
  - Fast ML inference on FPGAs

- hls4ml user-friendly interface for Pyton model → HSL conversion



Thank You!

Extra Slides

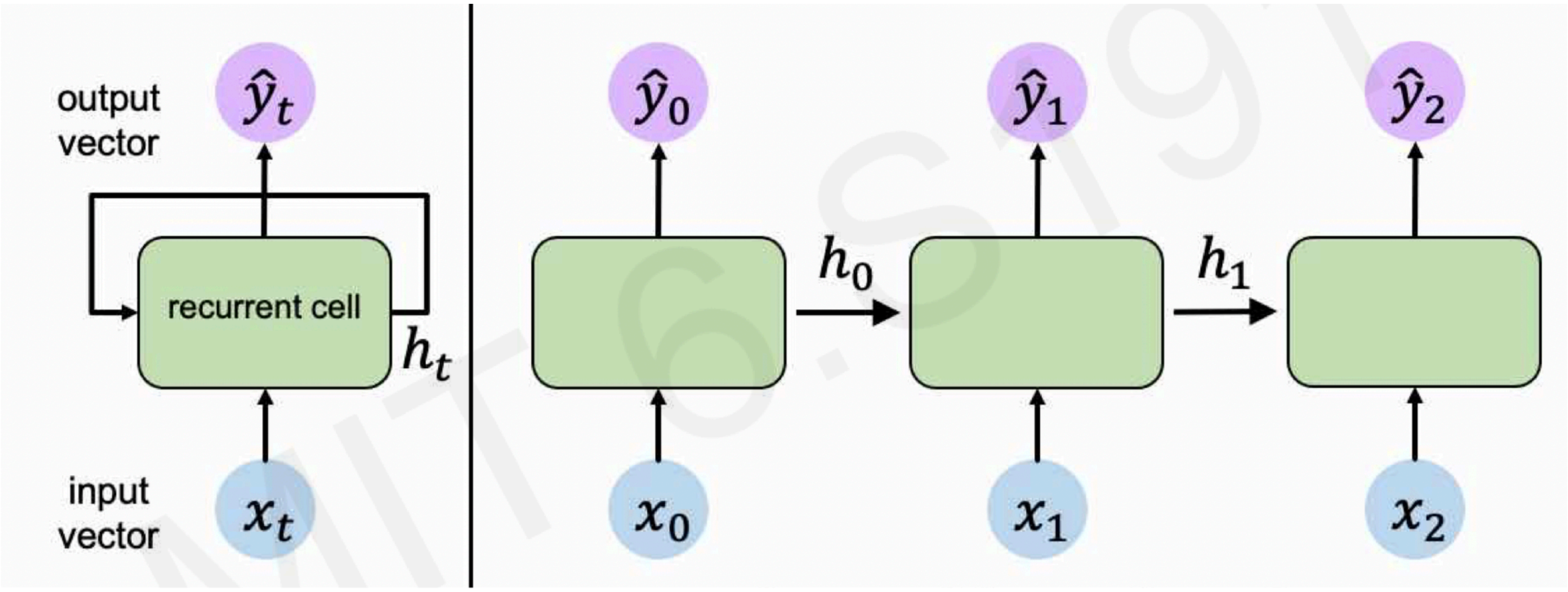
Recurrent Neural Network (RNN)

Recurrent Neural Networks

- Designed to work with sequential data
  - Text, audio, video, strokes, etc

- RNNs have a state, h_t, that is updated at each time step as the sequence is processed

- Recurrence relation at every time step



$$\hat{y} = f(x_t, h_{t-1})$$

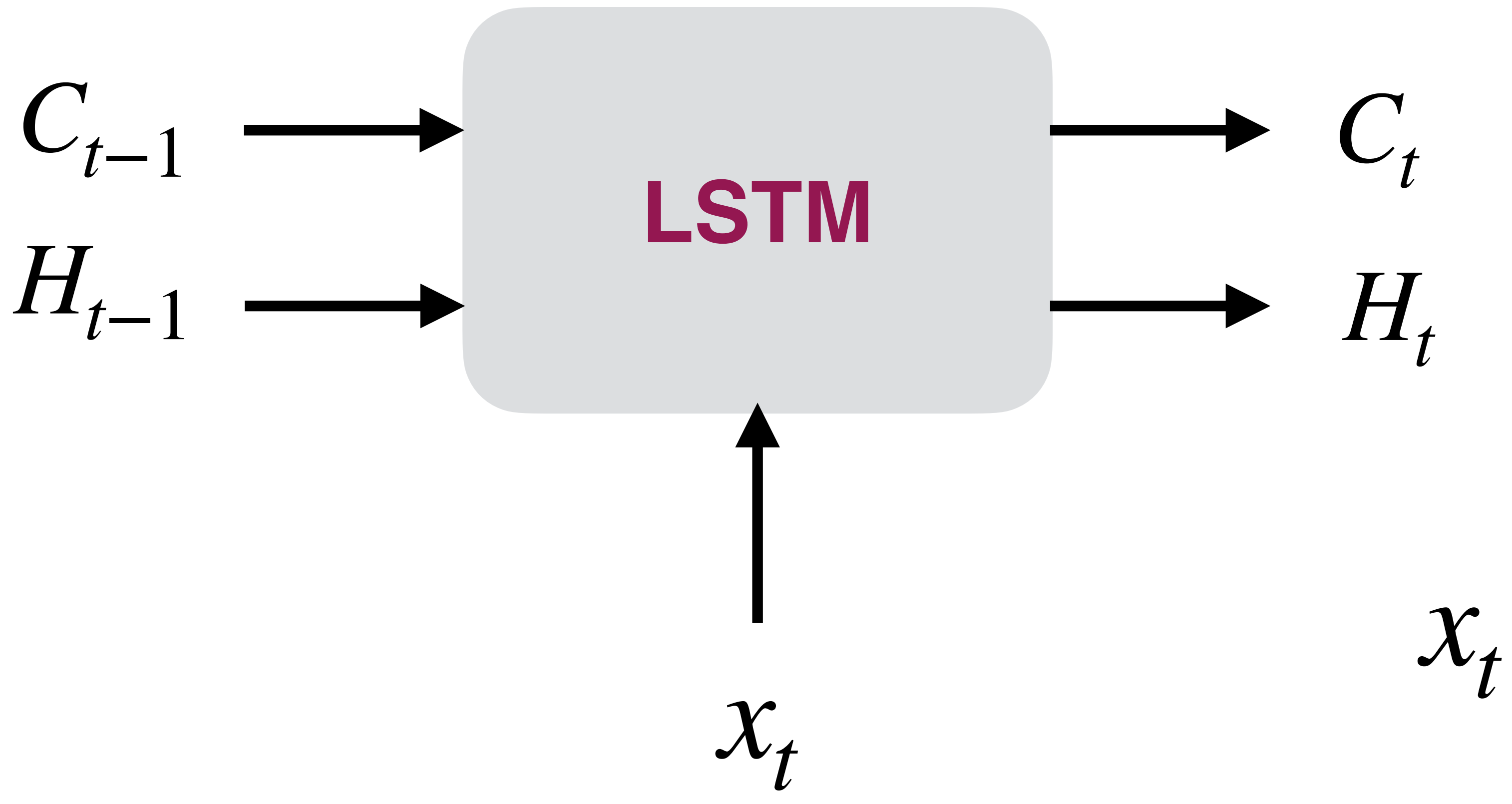
Output Input past memory

$$\underset{\text{cell state}}{h_t} = \underset{\text{Function with weights W}}{\boxed{f_W}}(\overset{\text{Input}}{\boxed{x_t}}, \overset{\text{old state}}{\boxed{h_{t-1}}})$$

Implementation of RNN models:

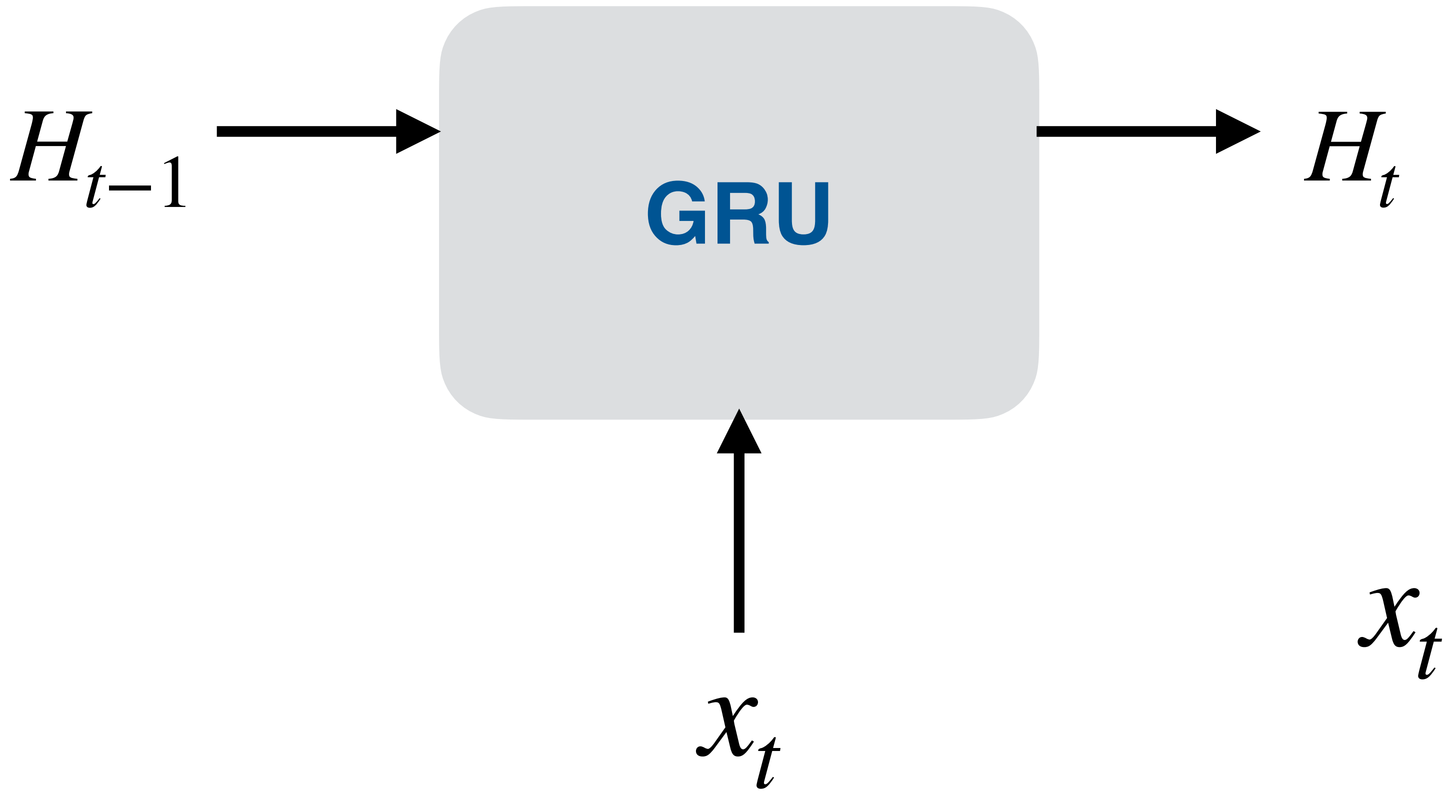
- LSTM (Long Short-Term Memory)
- GRU (Gated Recurrent Unit)

LSTM vs GRU



x_t = input

- **3 gates:** Input, Output, Forget
- **2 States:** Cell state (C_t) and Hidden state (H_t)

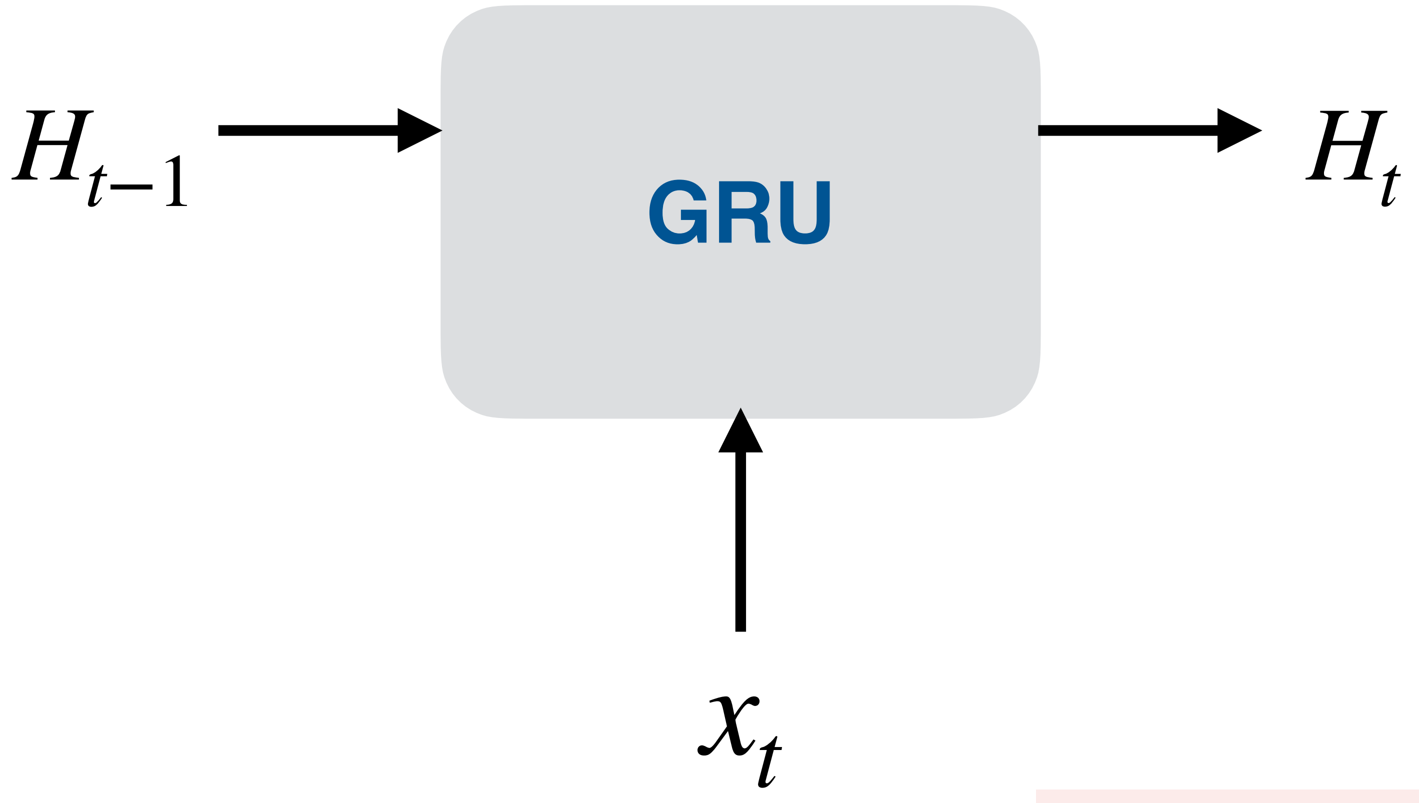


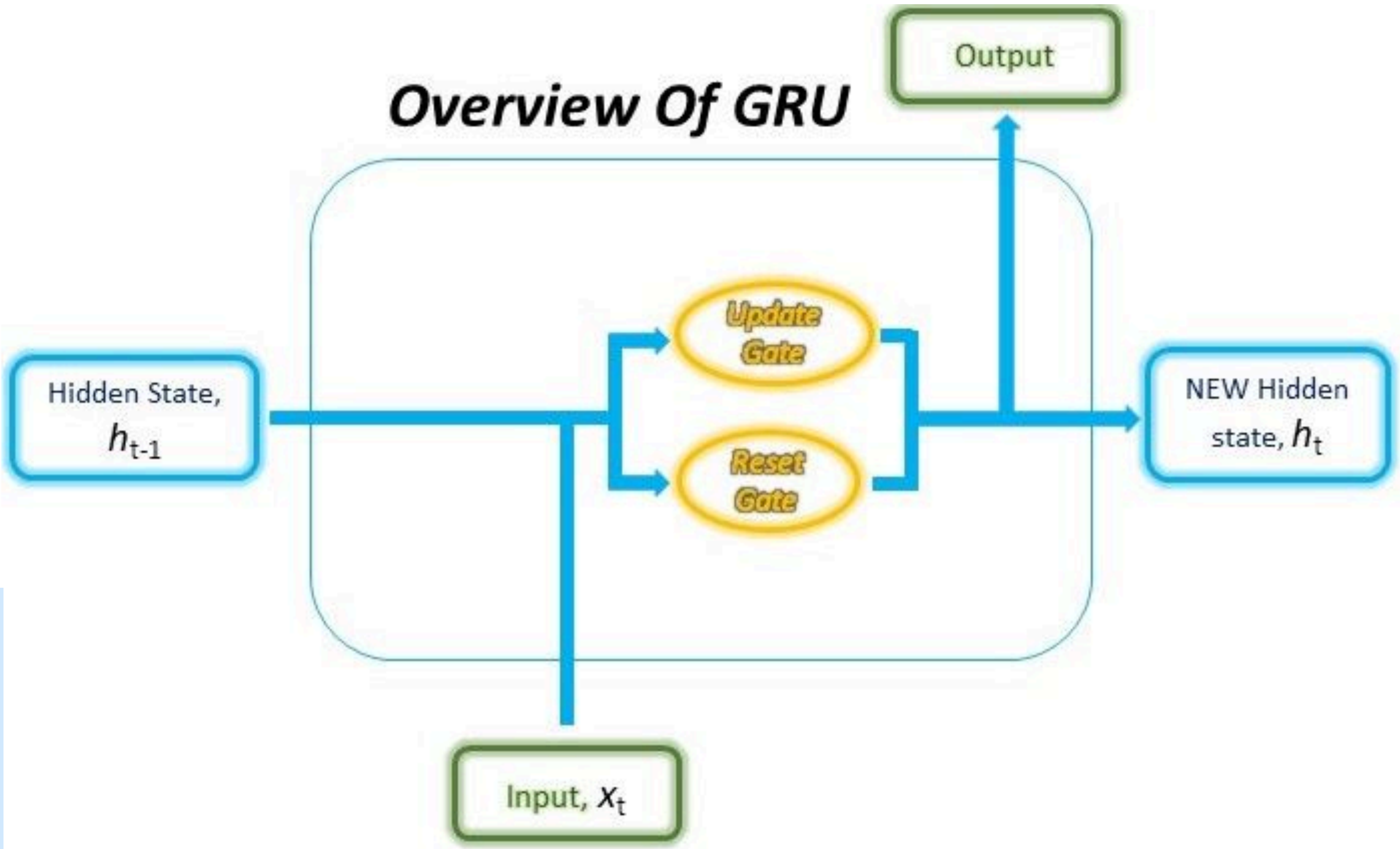
x_t = input

- **2 gates:** Update and Reset
- **Single** Hidden state (H_t)

- Less number of matrix multiplications
- Faster to train

Gated Recurrent Unit (GRU)





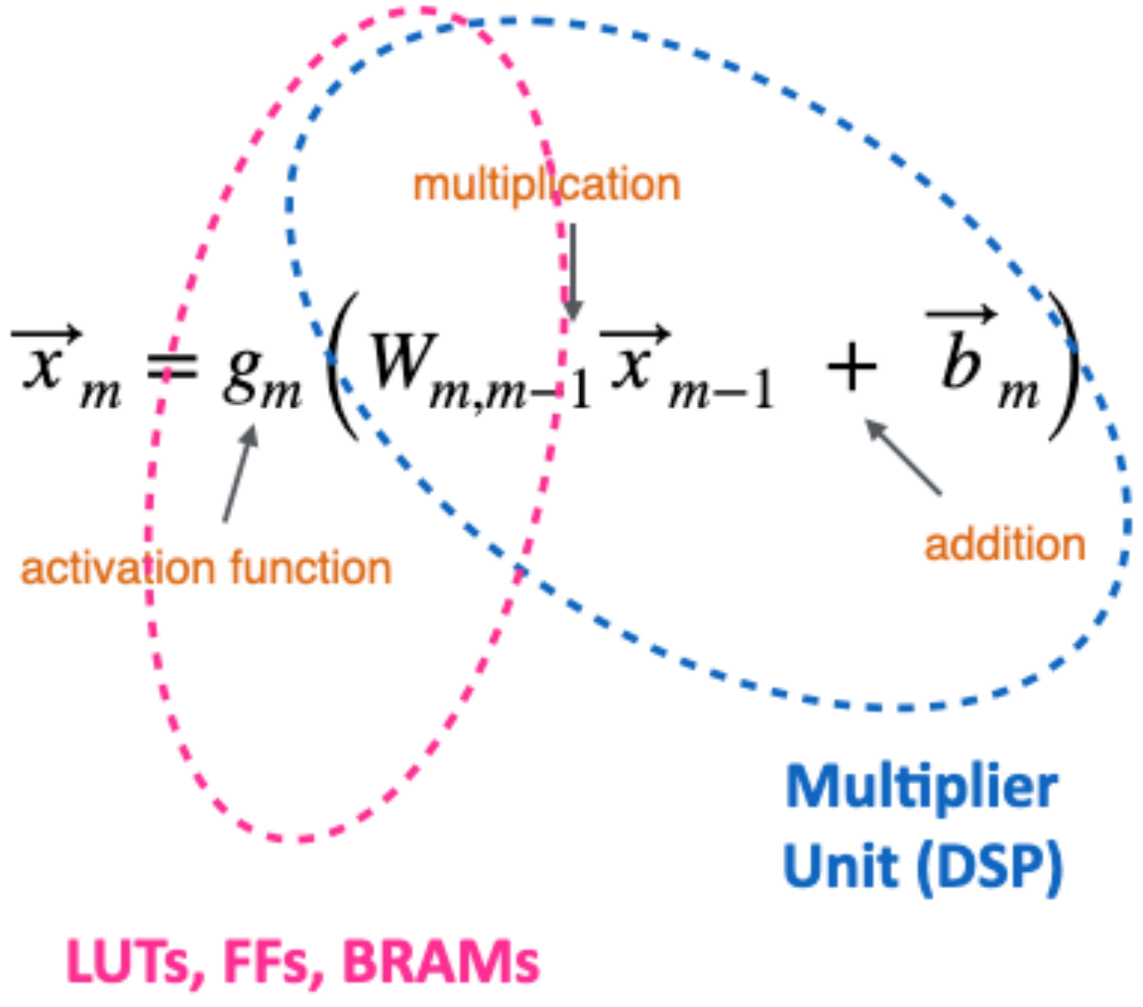
Dense | Dense

Reset: $r_t = \sigma\left(W_{xr} \cdot x_t + b_r + W_{hr} \cdot h_{t-1}\right)$

Update: $u_t = \sigma\left(W_{xu} \cdot x_t + b_u + W_{hu} \cdot h_{t-1}\right)$

Candidate hidden state: $\tilde{h}_t = \tanh\left(W_{xh} \cdot x_t + b_h + (r_t \odot h_{t-1}) \cdot W_{hh}\right)$

Output hidden state: $h_t = u_t \odot h_{t-1} + (1 - u_t) \cdot \tilde{h}_t$



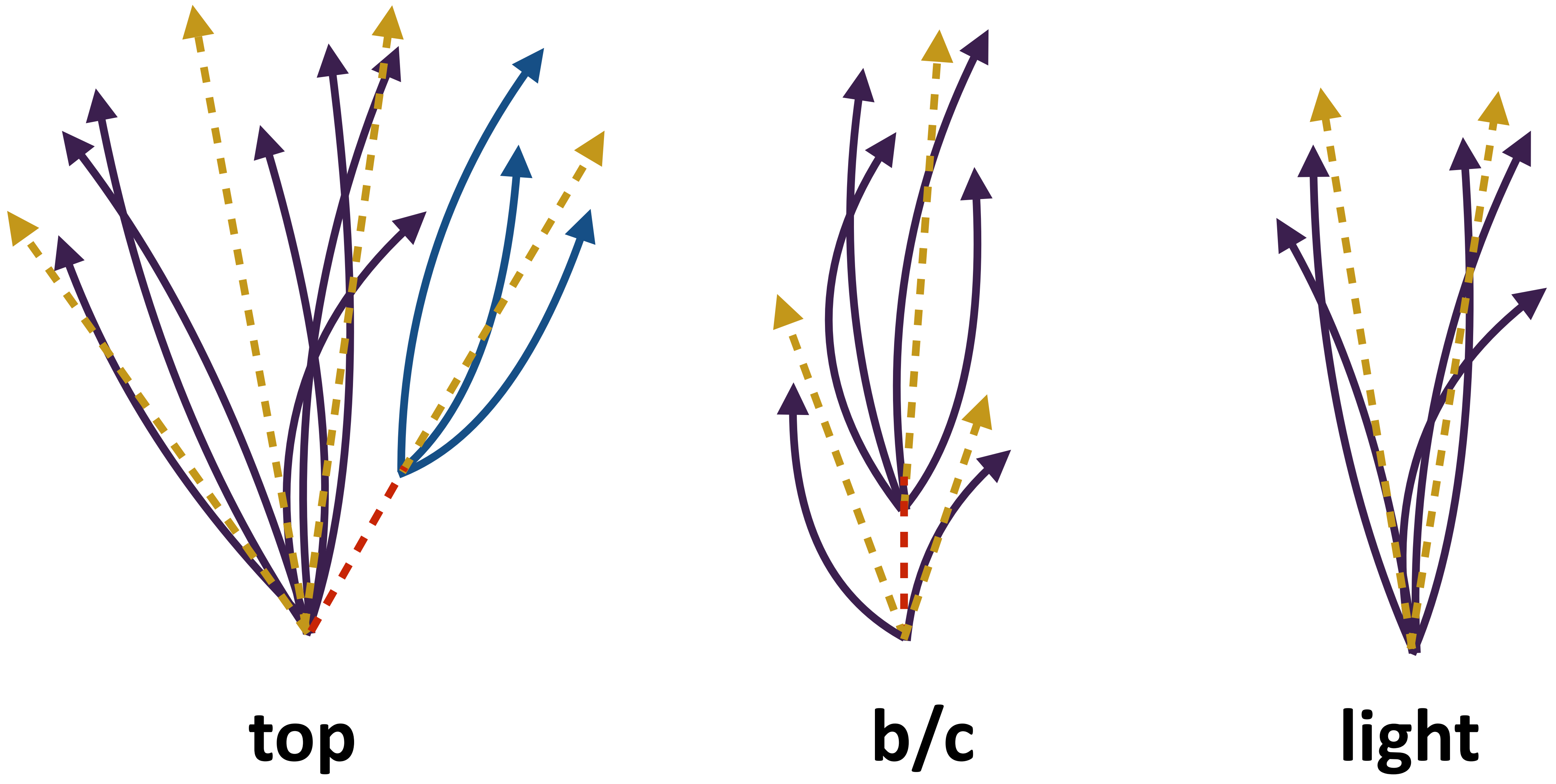
Benchmark Examples

Three benchmark cases

1. Binary classifier: ~4k parameters
   Identify top-quarks

2. 3-class classifier: ~60k parameters
   Classify b / c / light jets

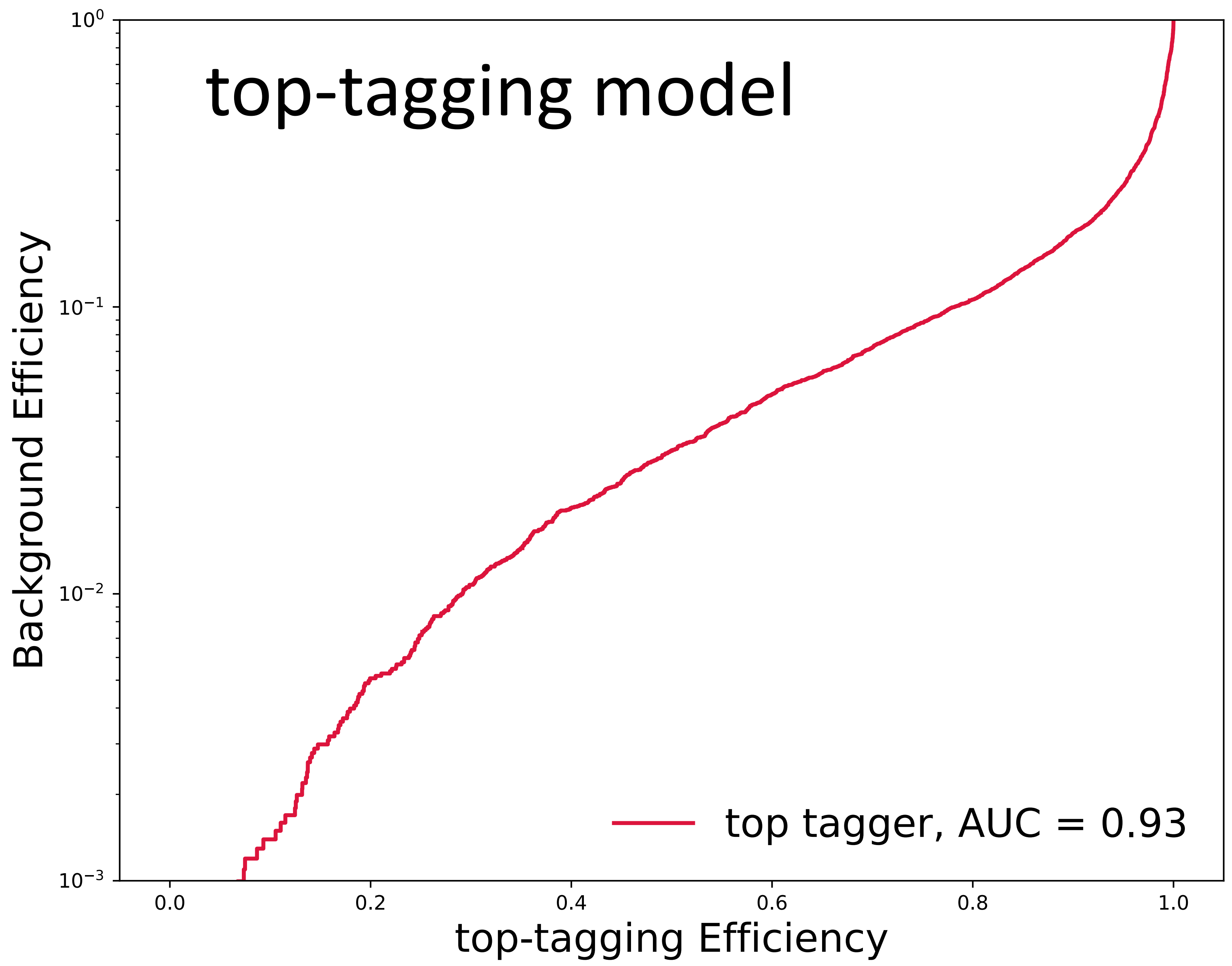
3. 5-class classifier: ~130k parameters
   QuickDraw dataset: differentiate between Bees, Butterflies, Mosquitos, Snails, Ants

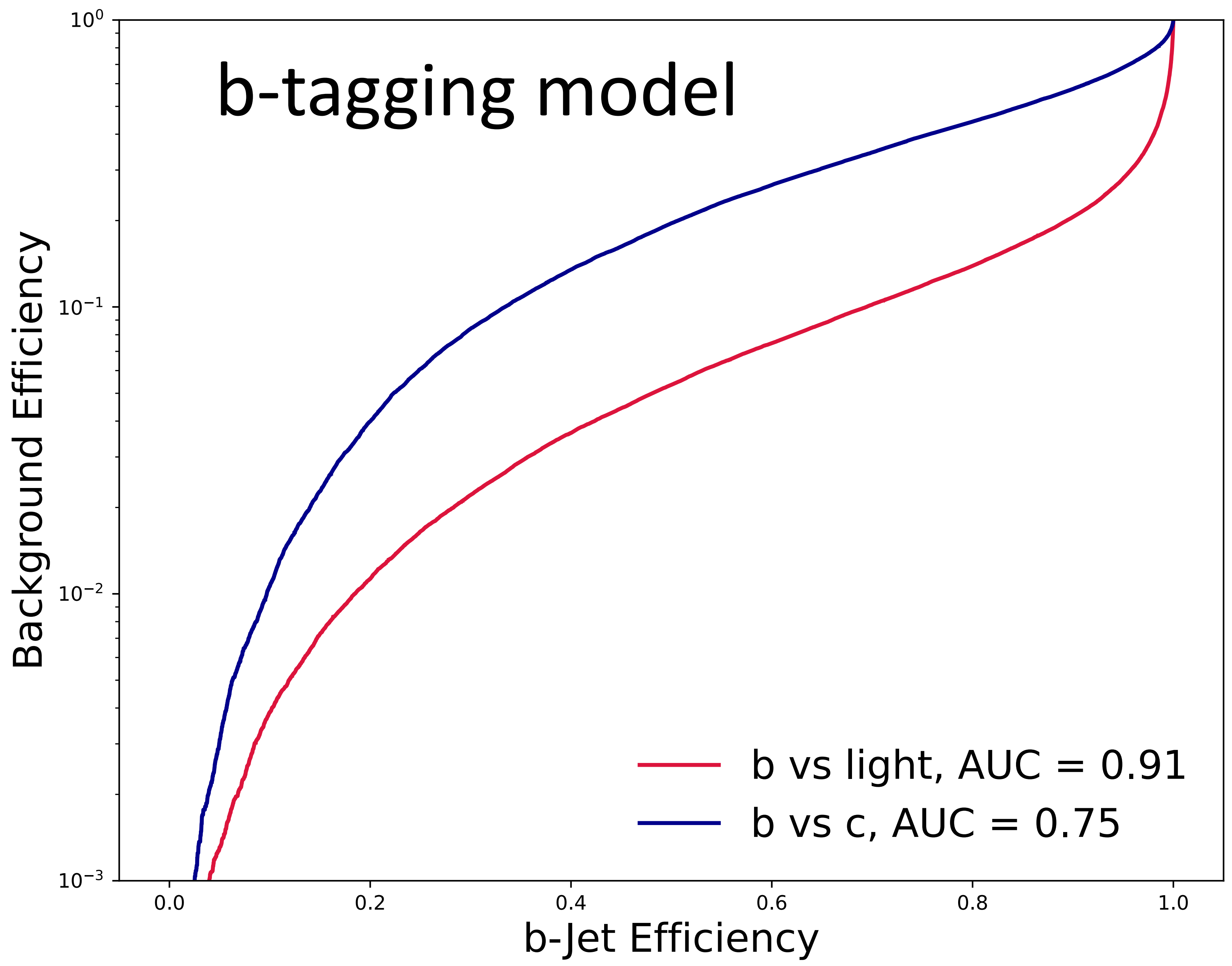


QuickDraw dataset

Model Performance: ROC

- All the benchmark models are trained using **Keras + TensorFlow**
- Weights and biases are represented by 32 bit floating point numbers



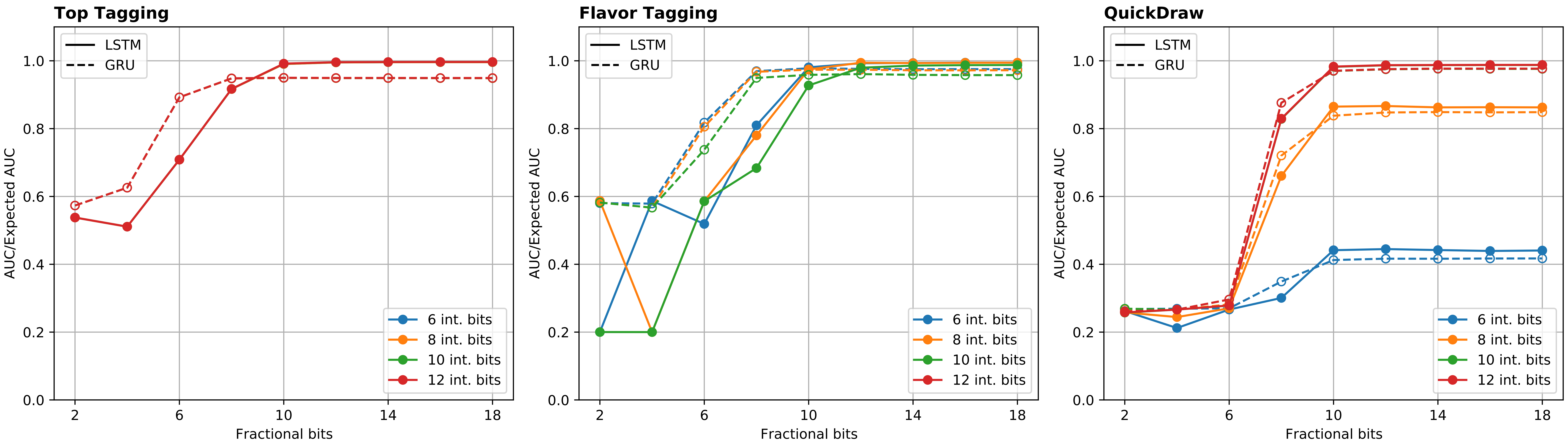


AUC after HLS conversion

Mach. Learn.: Sci. Tec. 4 025004

$$\text{Relative AUC} = \frac{\text{AUC}_{\text{HLS}}}{\text{AUC}_{\text{Keras}}}$$

- Post-training quantized **LSTM models** (with optimal precision) performs similar to the floating-point models
- Small performance degradation (< 5%) in the **GRU models** after quantization



Top Tagging

Flavor Tagging

QuickDraw

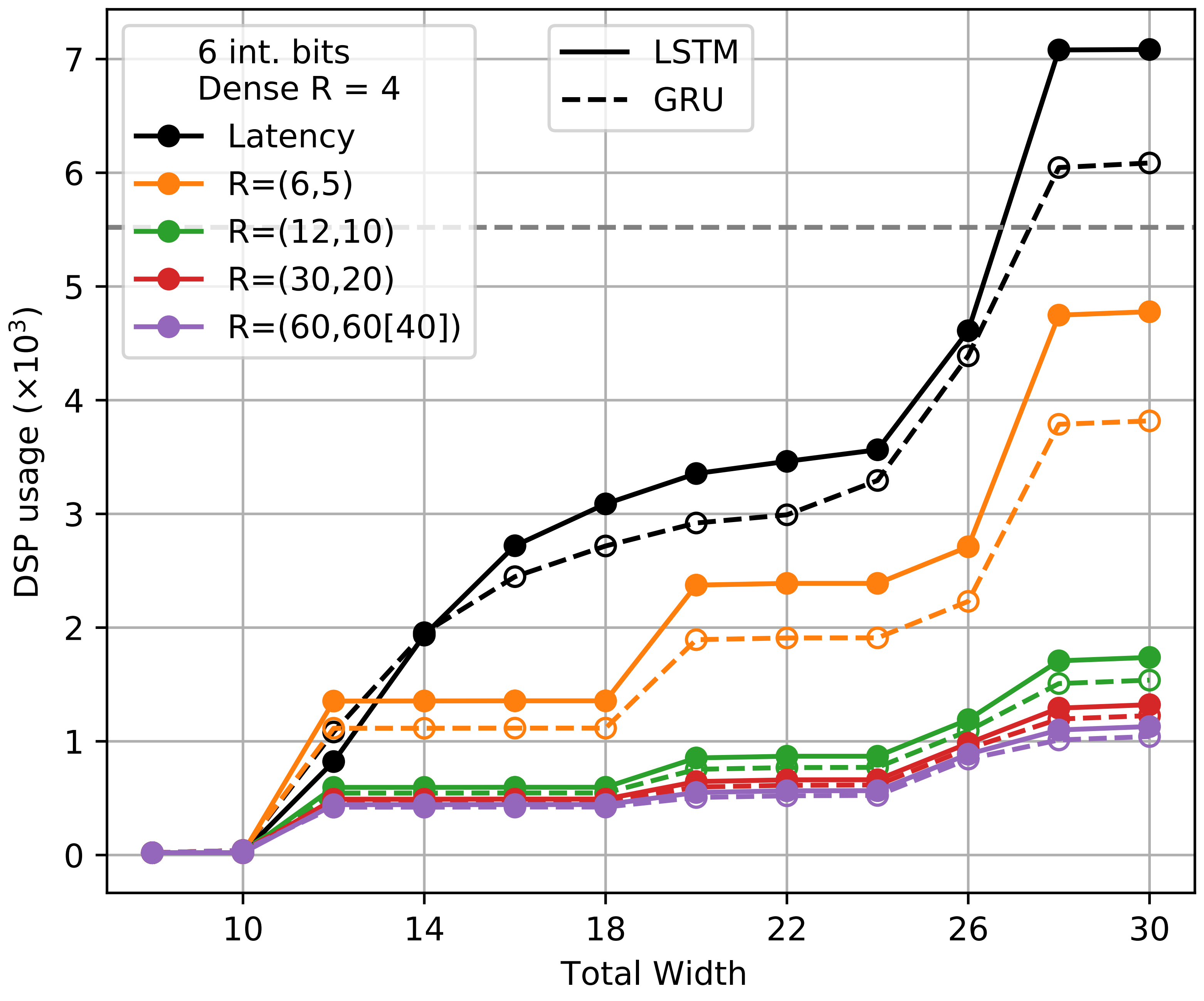
HLS Synthesis (RNN): DSP Usage

Mach. Learn.: Sci. Tec. 4 025004

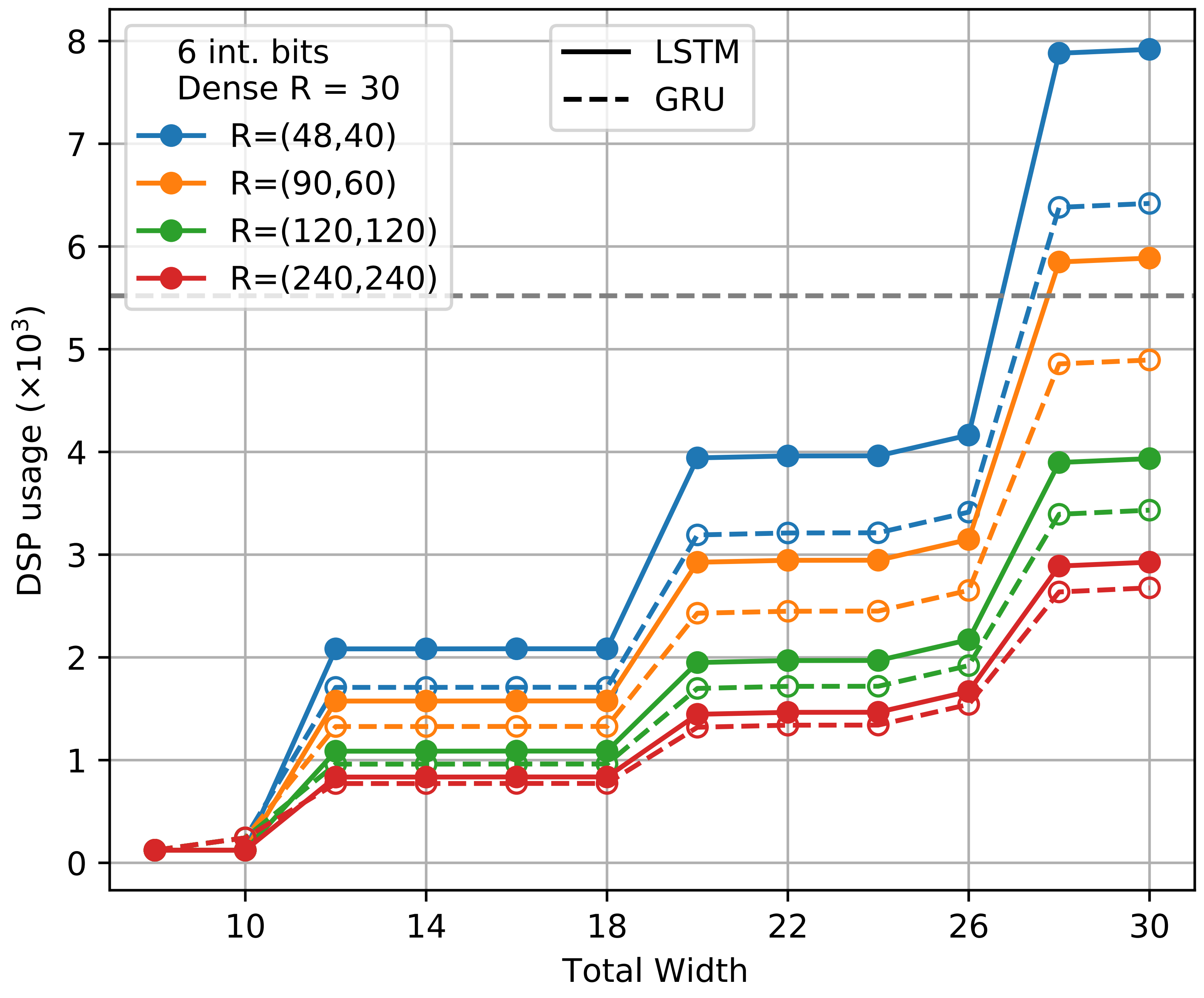
- **DSP usage** as a function of **Total bit width** after HLS synthesis
- The Jumps correspond to DSP input width

Synthesized using Xilinx Kintex UltraScale FPGA
FPGA part: xcku115-flvb2104-2-i

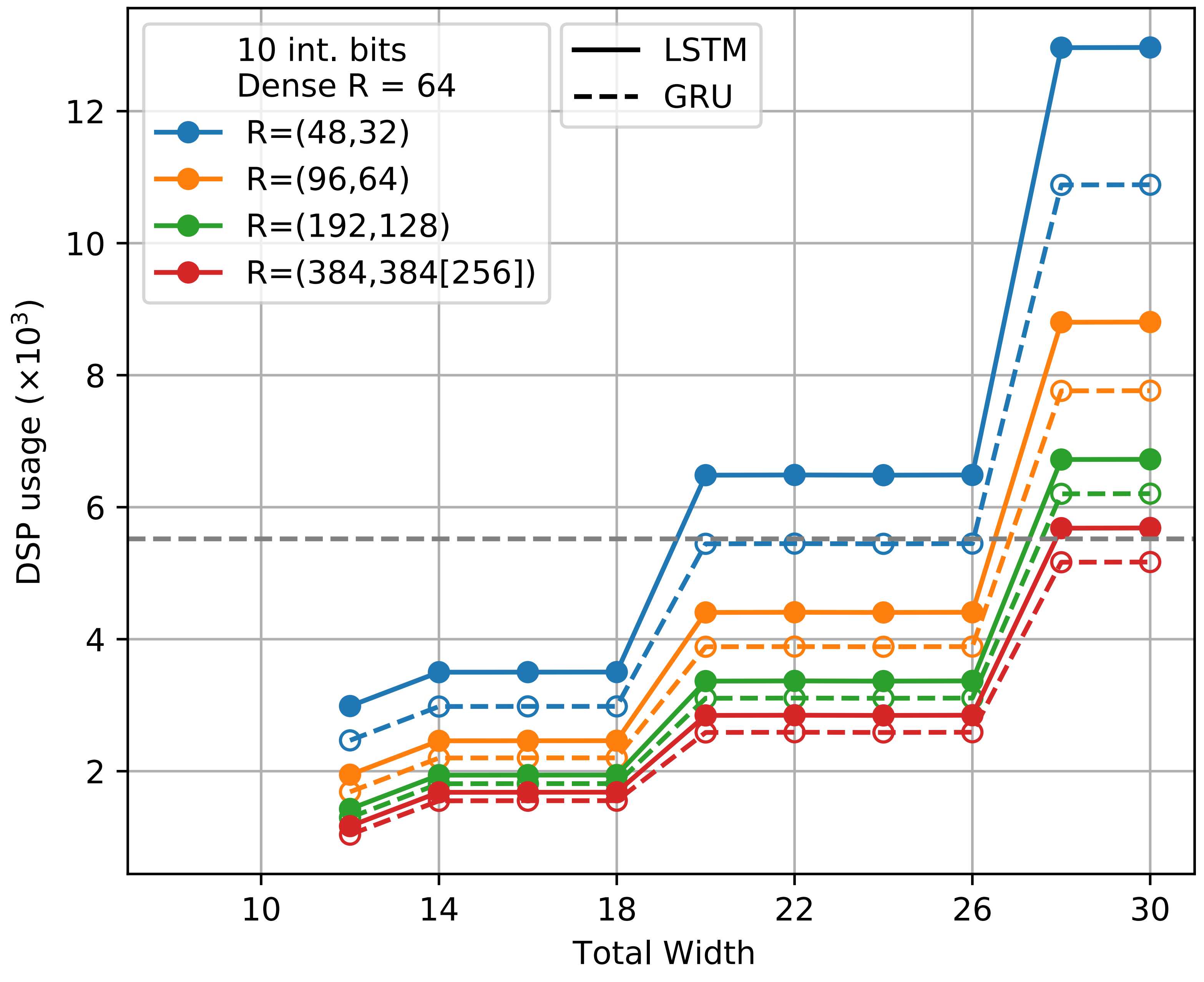
hls4ml: Top Tagging



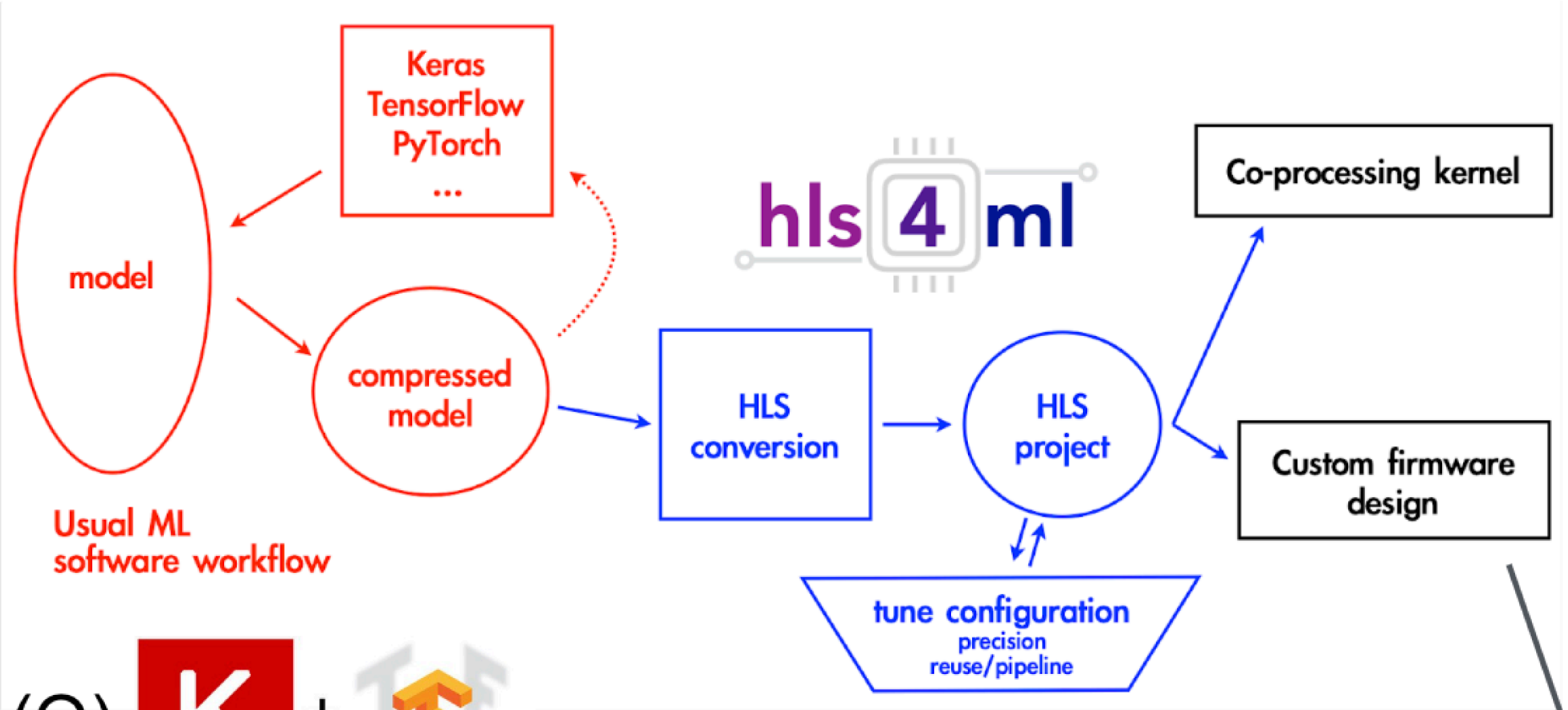
hls4ml: Flavor Tagging



hls4ml: QuickDraw



High Level Synthesis with Machine Learning (hls4ml)



(Q) K + TensorFlow

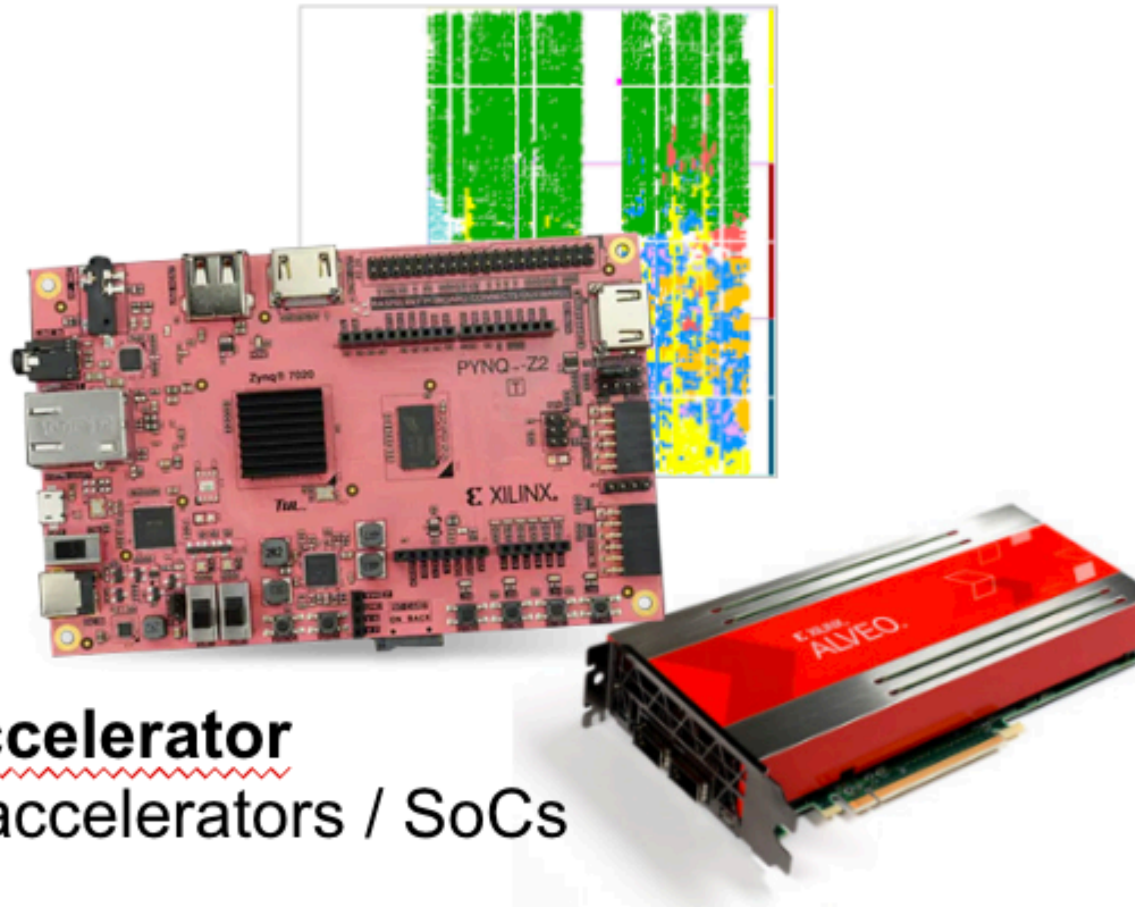
(Q) ONNX

PYTORCH

Vivado™ HLS

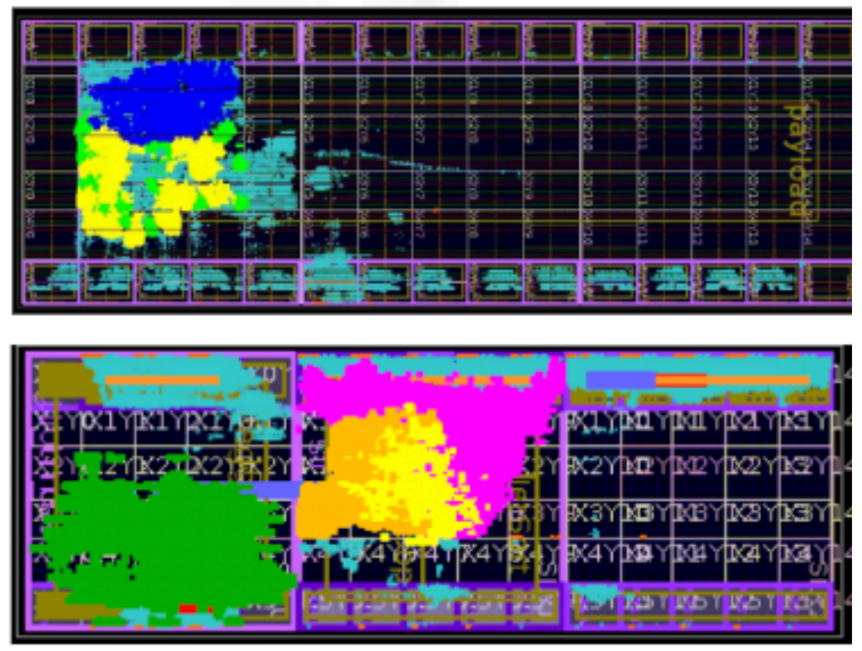
intel HLS COMPILER

Mentor®
Catapult
Coming soon

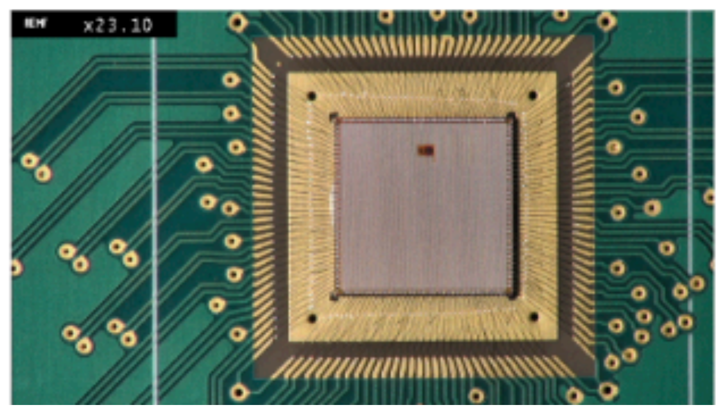


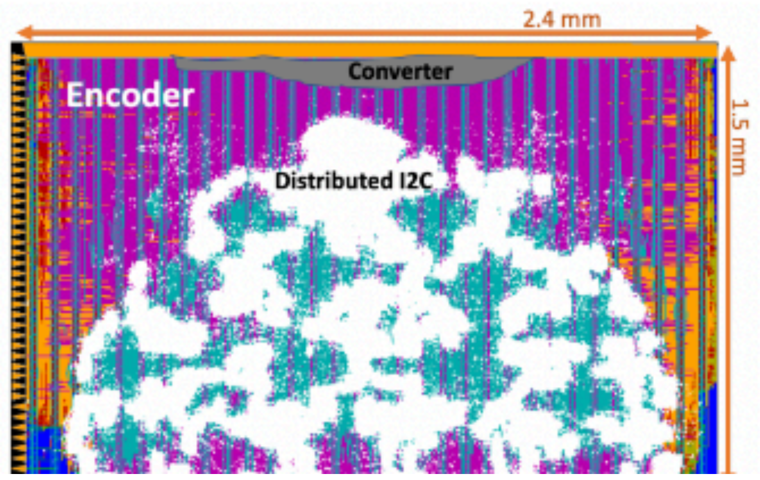
VivadoAccelerator
for Xilinx accelerators / SoCs

Custom design
(talk 1, 2)

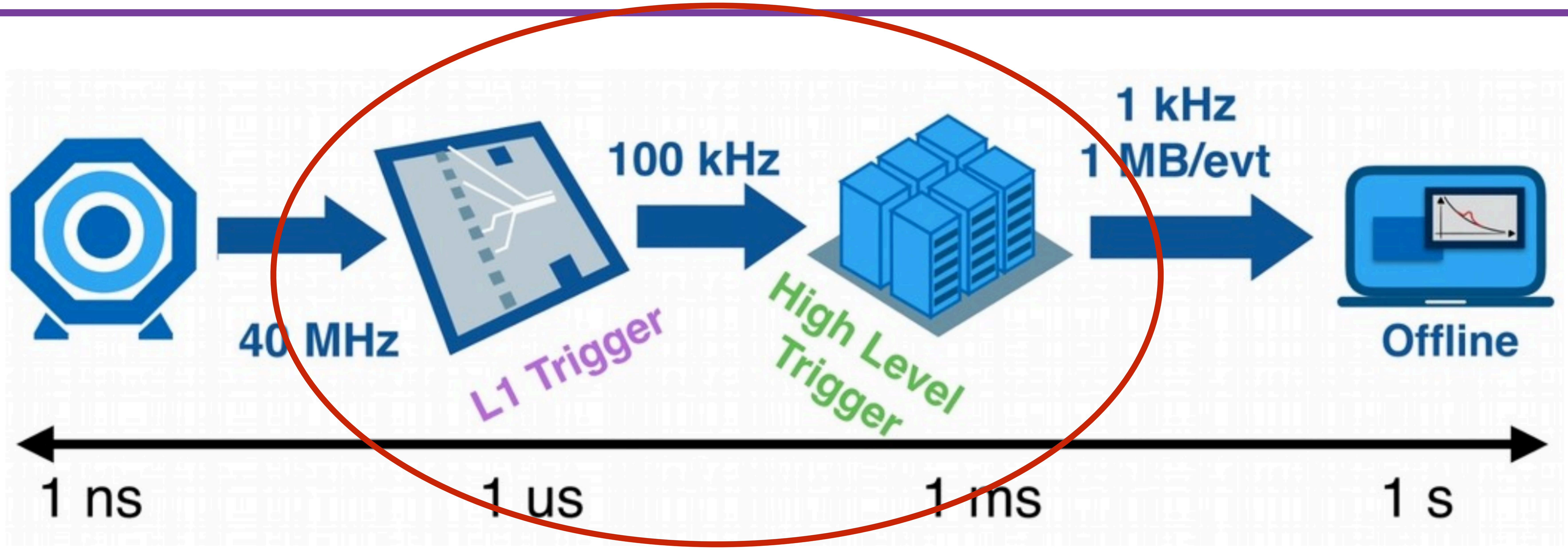


ASICs (talk 1, 2)





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?