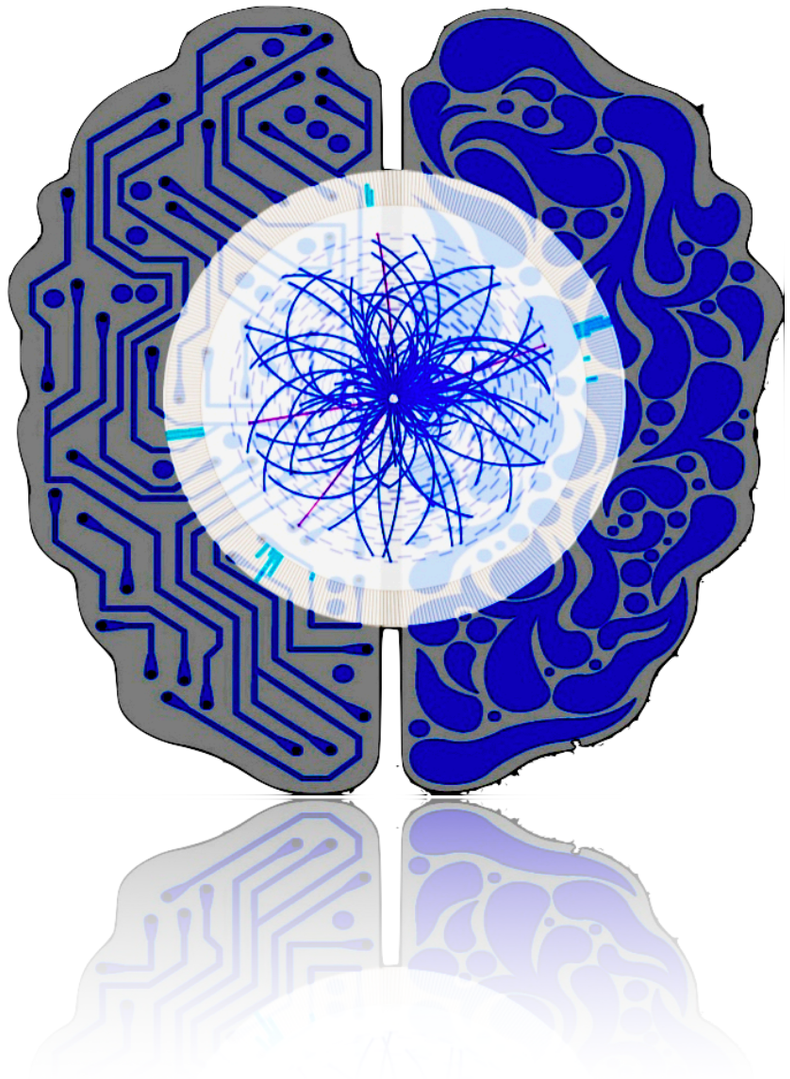
Real-time ML on FPGAs for particle physics

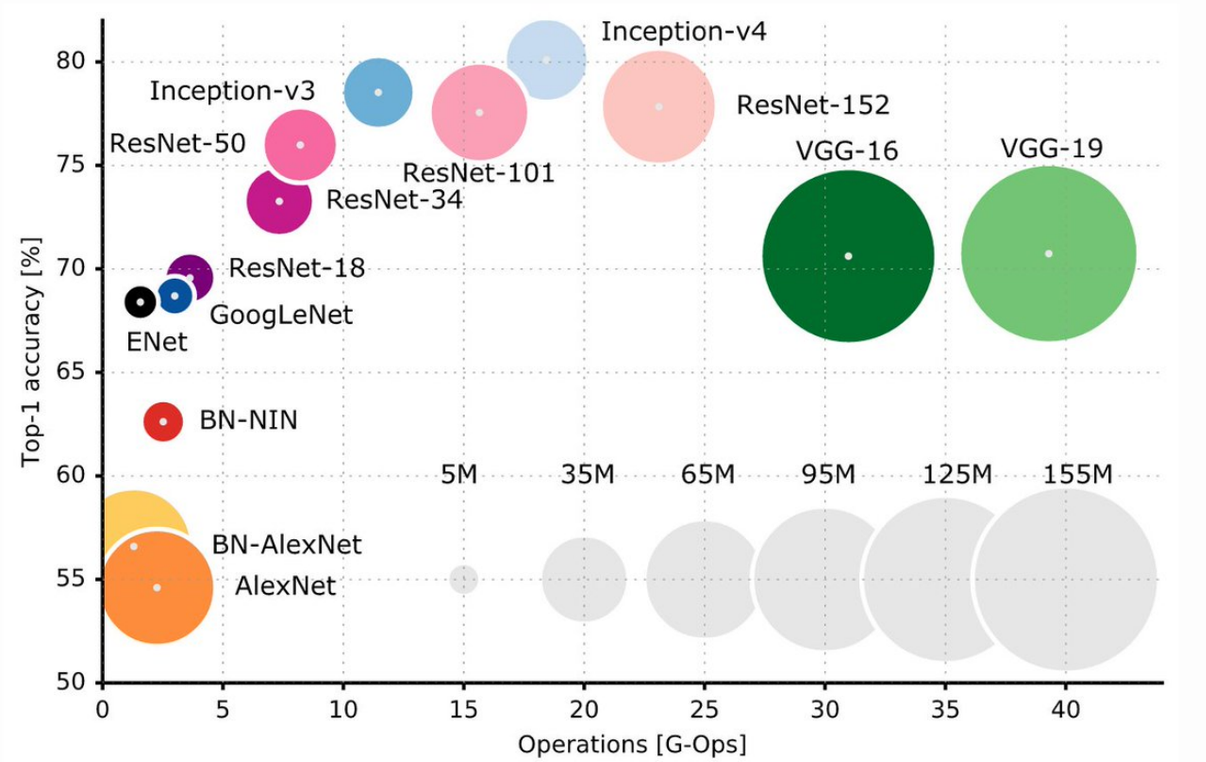


Jennifer Ngadiuba (Fermilab)
Sioni Summers (CERN)

2nd Terascale School of Machine Learning
10 March, 2021

Introduction

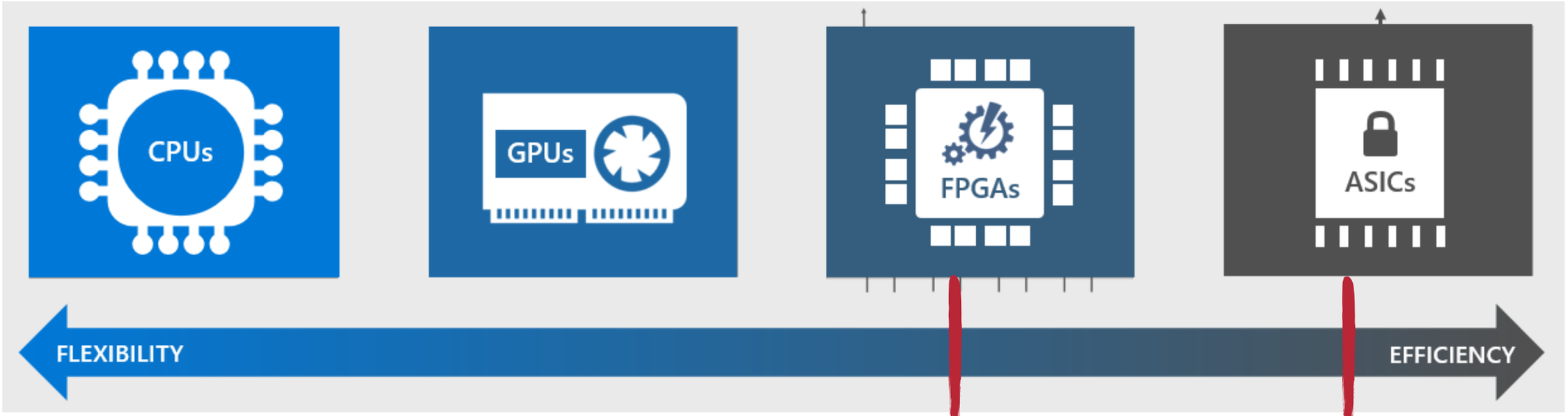
- In this school you have learned about several complex and large DL models (likely not the largest in the world but big enough…)
- You have probably also learned that you can **relatively efficiently and quickly** train or evaluate them on GPUs

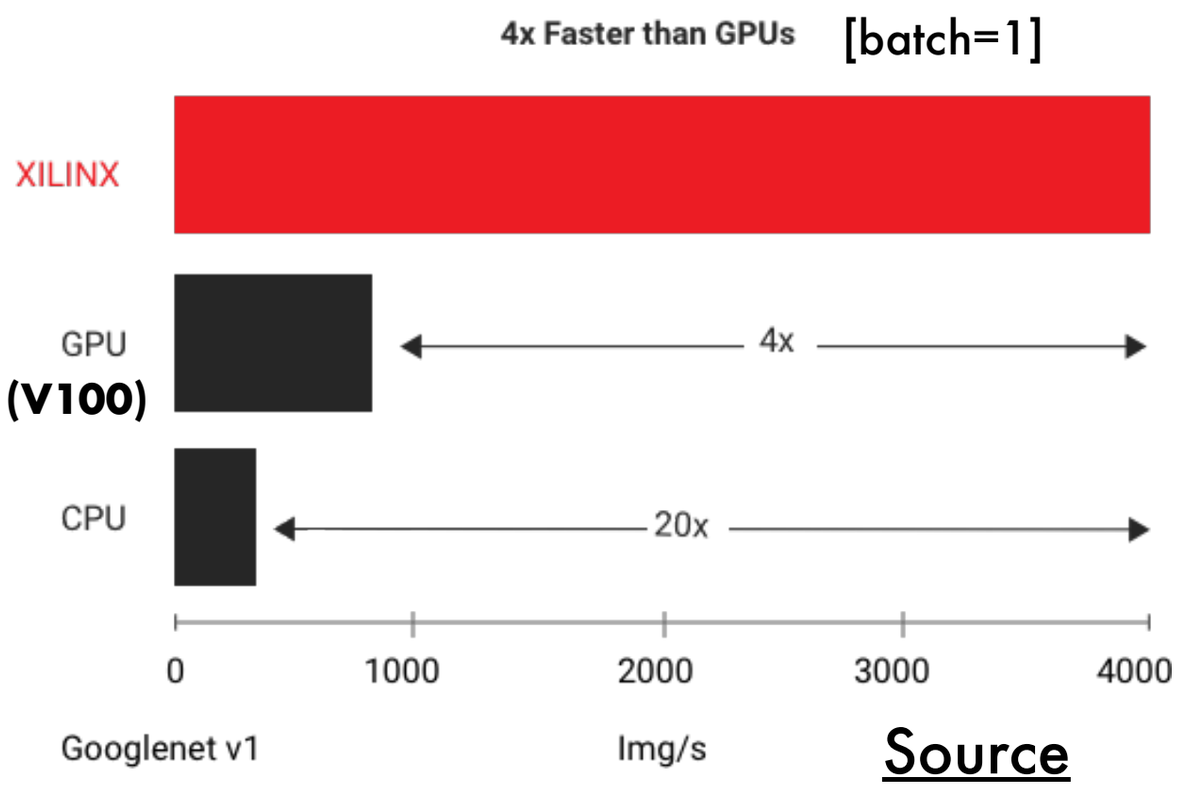


- **Relatively** means that it depends a lot on your application
- For some applications speed and efficiency are crucial and GPUs might not the best solution!

The rise of specialized hardware

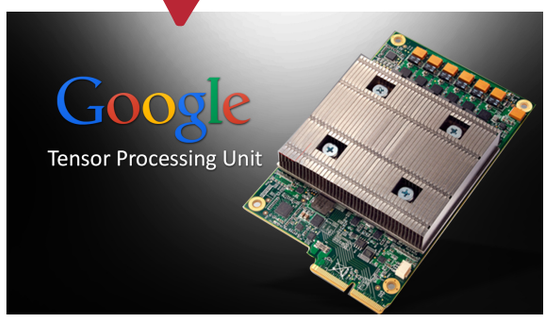
Recent industry trends towards developing new devices optimized for AI and speed up both training and inference





Source

Accelerating DNN with Xilinx FPGAs

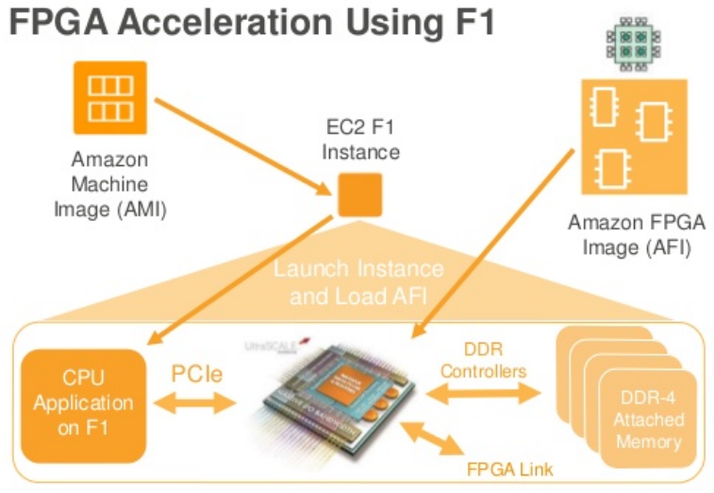


https://cloud.google.com/tpu/

The rise of specialized hardware

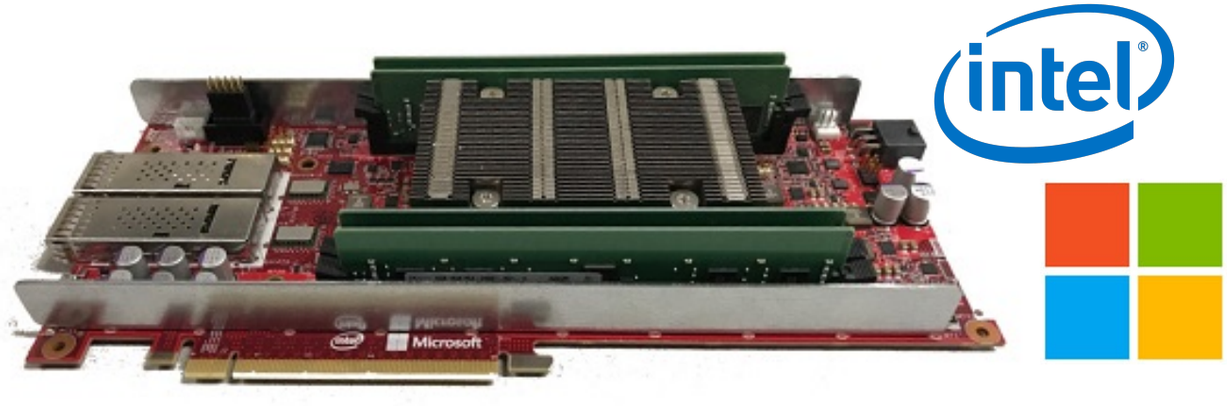
FPGAs and ASICs making their way into data centers as co-processors

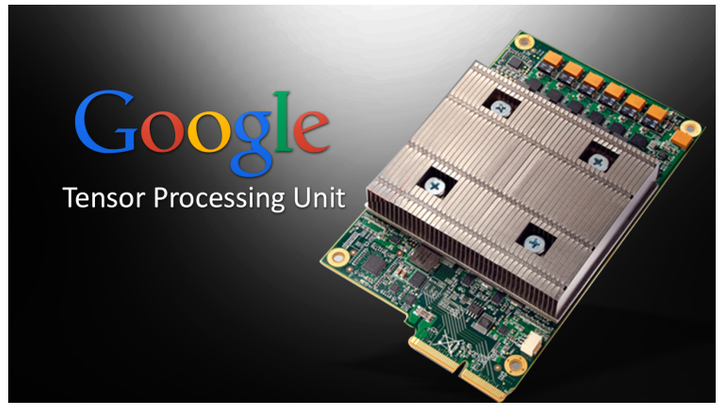
Xilinx & Amazon Web Service
Intel & Microsoft BrainWave
Google cloud TPUs
Xilinx & IBM cloud (CAPI)



Xilinx KUS FPGA
5k DSPs
1.4M logic cells
1.3M FF
80Mb BRAM

Companies also provide **toolkits to accelerate custom or standard DL models on FPGAs:**
Intel OpenVino, …
Xilinx ML Suite, …





high level synthesis for machine learning

- Today you will learn about **hls4ml**: a package for translating neural networks to FPGA firmware for inference with ultra low latency

  https://github.com/fastmachinelearning/hls4ml

  https://fastmachinelearning.org/hls4ml/

  ```
  pip install hls4ml
  ```

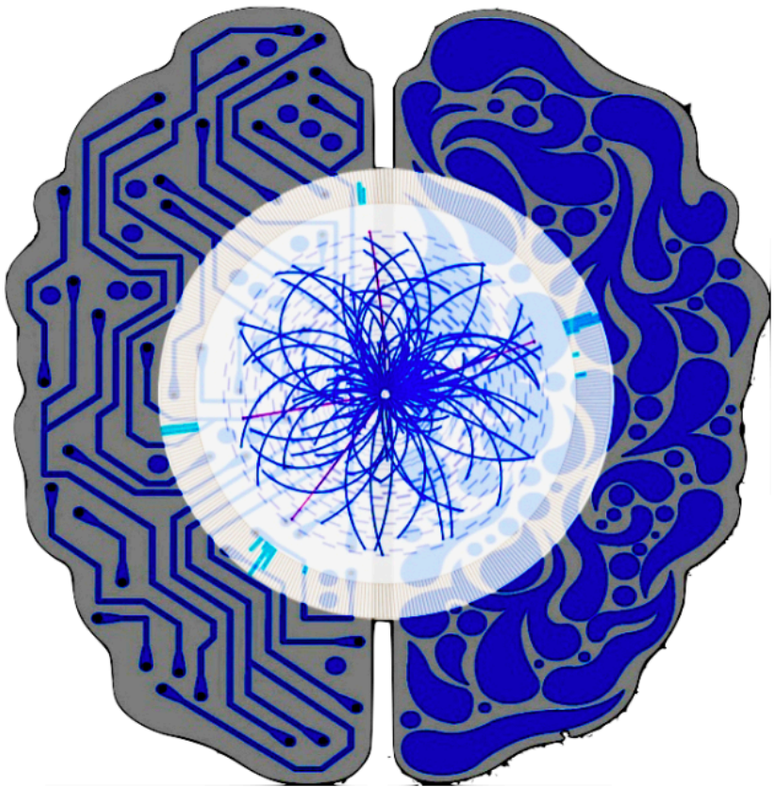
- **Objectives:**
  - Introduction on FPGA functionalities
  - Translate ML models into synthesizable FPGA code
  - Make your model inference computationally efficient and fast

FAST MACHINE LEARNING LAB

ABOUT THE FAST ML LAB

Real-time and accelerated ML for fundamental sciences

Fast ML Lab is a research collective of physicists, engineers, and computer scientists interested in deploying machine learning algorithms for unique and challenging scientific applications. Our projects range from real-time, on-detector and low latency machine learning applications to high-throughput heterogeneous computing big data challenges. We are interested in deploying sophisticated machine learning algorithms to advance the exploration of fundamental physics from the world's biggest colliders to the most intense particle beams to the cosmos.



The project kicked off 4 years ago …
~ 10 people, mostly physicists (with little expertise in electronic engineering)

https://fastmachinelearning.org/

Many more contributors and users now!

Caltech
Jennifer Ngadiuba (PhD, Physics);

CERN
Thea Årrestad (PhD, Physics); Vladimir Loncar (PhD, Computer Science); Maurizio Pierini (PhD, Physics); Sioni Summers (PhD, Physics);

Columbia University
Giuseppe Di Guglielmo (PhD, Computer Science);

Fermilab
Lindsey Gray (PhD,Physics); Christian Herwig (PhD,Physics); Duc Hoang (Undergraduate, Physics); Burt Holzman (PhD,Physics); Sergo Jindariani (PhD,Physics); Thomas Klijnsma (PhD,Physics); Ben Kreis (PhD,Physics); Kevin Pedro (PhD,Physics); Ryan Rivera (PhD,EE); Nhan Tran (PhD,Physics); Mike Wang (PhD,Physics); Tingjun Yang (PhD,Physics);

Hawkeye 360
EJ Kreinar (Computer Science)

Lehigh University
Joshua Agar (PhD, Material Science and Engineering);

MIT
Jack Dinsmore (Undergraduate, Physics); Song Han (PhD, EECS); Phil Harris (PhD, Physics); Jeffrey Krupa (Graduate, Physics); Sang Eon Park (Graduate, Physics); Dylan Rankin (PhD, Physics);

Northwestern University
Seda Memik-Ogrenci (Electrical and Computer Engineering); Farah Fahim (ECE, Adjunct); Nhan Tran (ECE, Adjunct)

Purdue University
Mia Liu (PhD,Physics);

UC San Diego
Javier Duarte (PhD, Physics); Vesal Razavimaleki (Undergraduate, Engineering Physics);

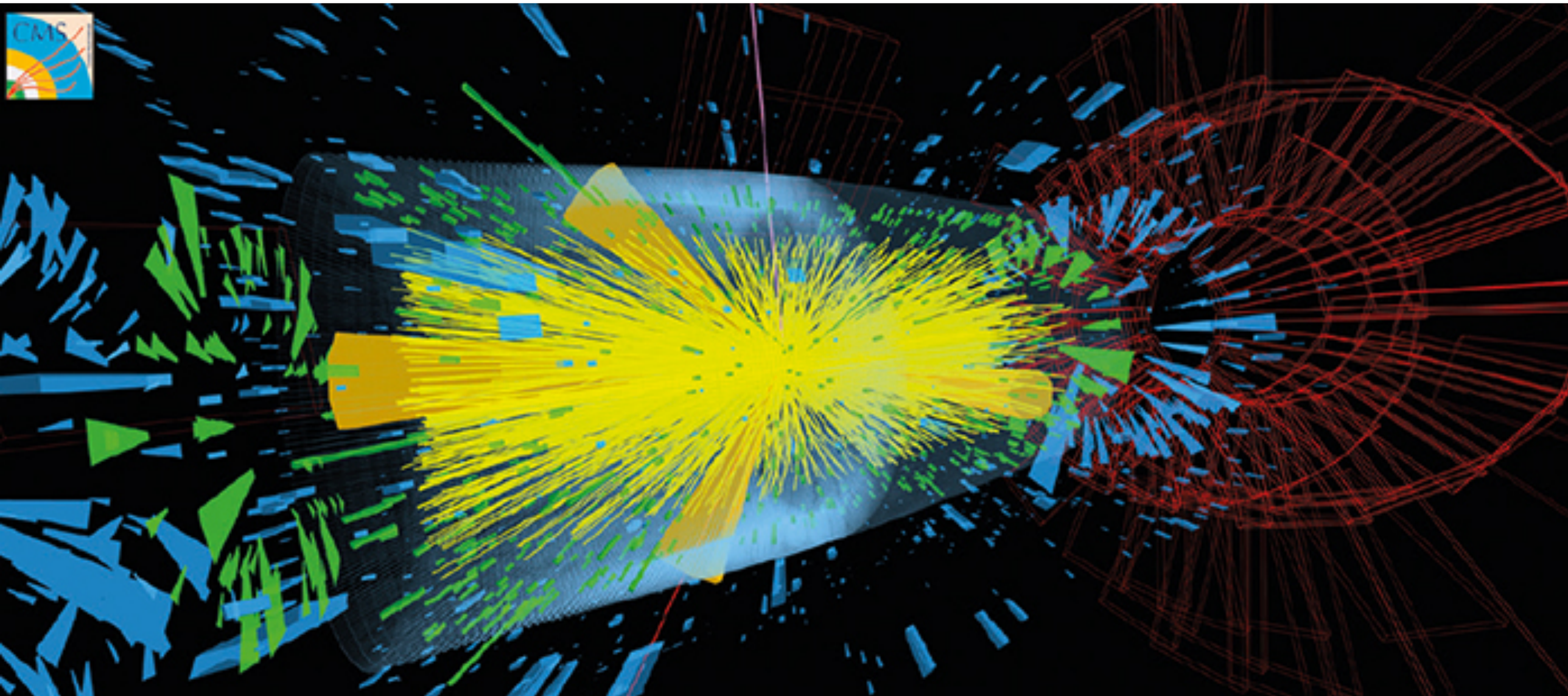
University of Illinois Chicago
Zhenbin Wu (PhD, Physics);

University of Illinois Urbana-Champaign
Markus Atkinson (PhD, Physics); Mark Neubauer (PhD, Physics);

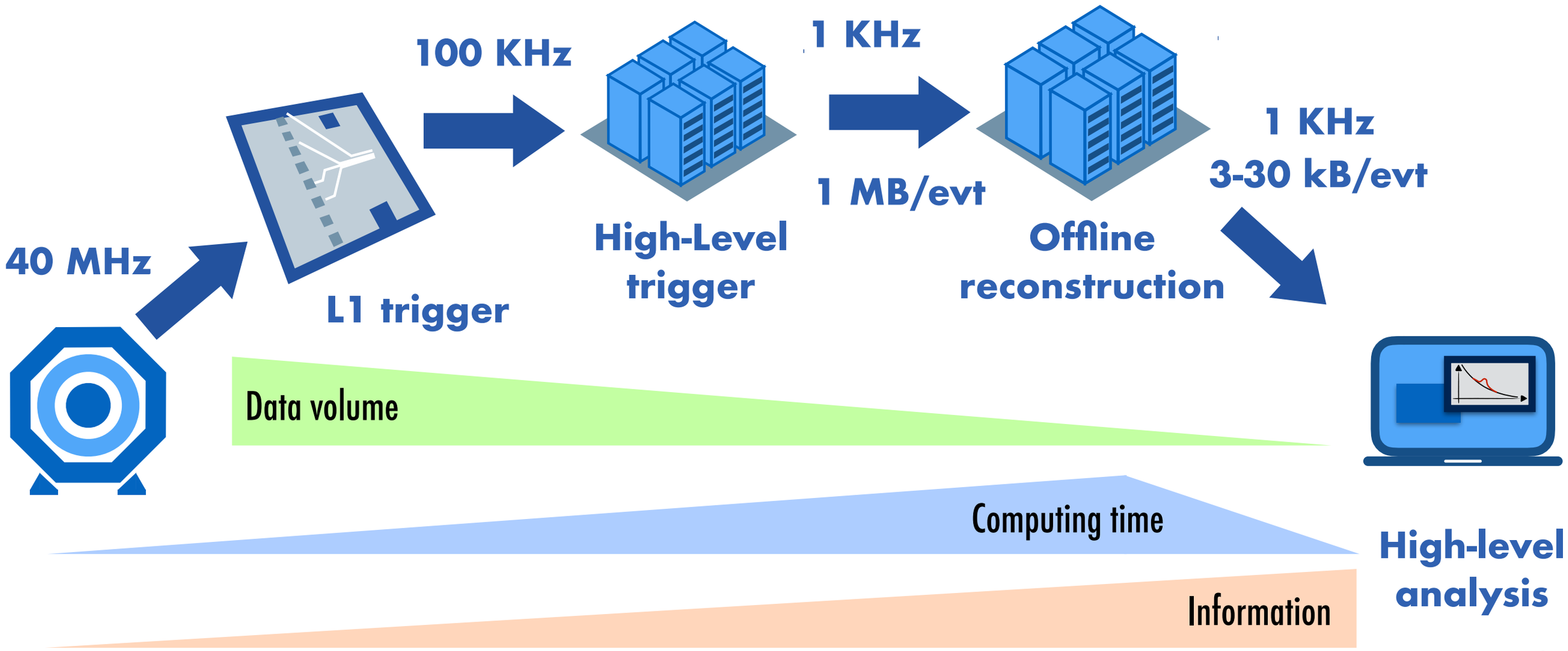
University of Washington
Scott Hauck (PhD, EECS); Shih-Chieh Hsu (PhD, Physics);

The origins: triggering @ (HL-) LHC

Extreme collision frequency of 40 MHz → extreme data rates O(100 Tb/s)
Most collision "events" don't produce interesting physics
"Triggering" = filter events to reduce data rates to manageable levels



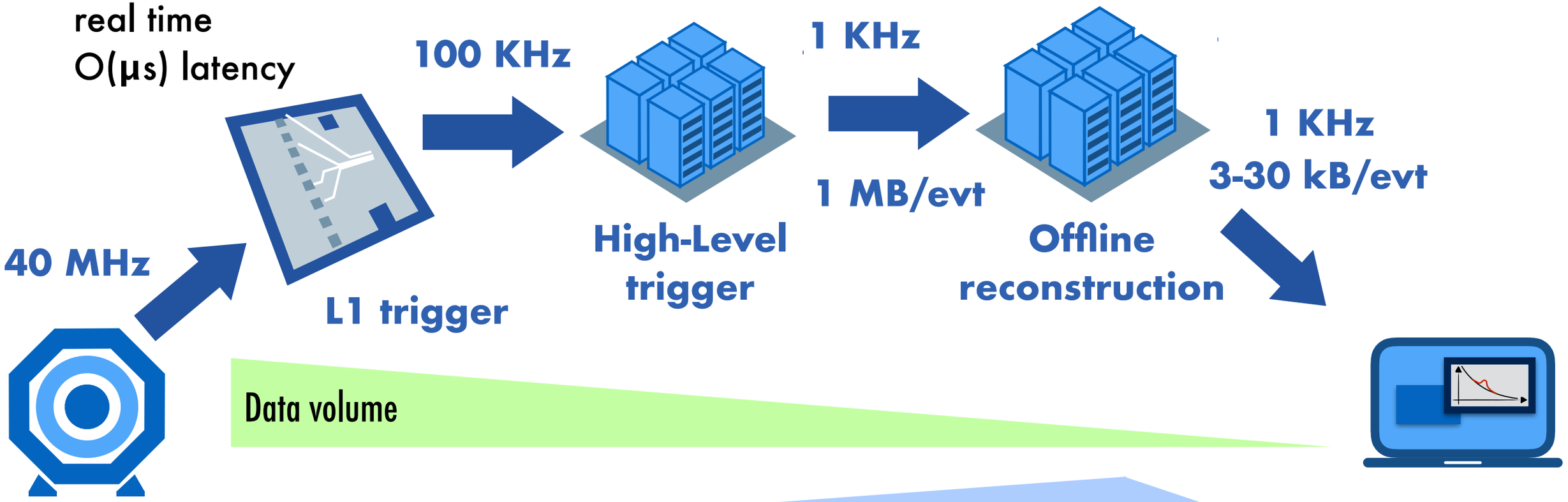
The typical LHC data flow



The typical LHC data flow

99.75% events rejected!

Hardware based
Runs on FPGAs in real time
O(µs) latency



The typical LHC data flow

99.75% events rejected!

Hardware based
Runs on FPGAs in real time
O(µs) latency

99% events rejected!

Software based
Runs on CPUs in real time
O(100 ms) latency

No data stored before this line

40 MHz

100 KHz

L1 trigger

High-Level trigger

1 KHz

1 MB/evt

Offline reconstruction

1 KHz
3-30 kB/evt

Data volume

Computing time

Information

High-level analysis

The typical LHC data flow

99.75% events rejected!

Hardware based
Runs on FPGAs in real time
O(µs) latency

99% events rejected!

Software based
Runs on CPUs in real time
O(100 ms) latency

No data stored before this line

Software based
Run on data centers (LHC grid)
O(s) computing time

40 MHz

100 KHz

L1 trigger

High-Level trigger

1 KHz

1 MB/evt

Offline reconstruction

1 KHz
3-30 kB/evt

Data volume

Computing time

Information

High-level analysis

The typical LHC data flow

99.75% events rejected!

Hardware based
Runs on FPGAs in real time
O(µs) latency

99% events rejected!

Software based
Runs on CPUs in real time
O(100 ms) latency

No data stored before this line

Software based
Run on data centers (LHC grid)
O(s) computing time

40 MHz

100 KHz

L1 trigger

High-Level trigger

1 KHz

1 MB/evt

Offline reconstruction

1 KHz
3-30 kB/evt

Run by individuals on the grid

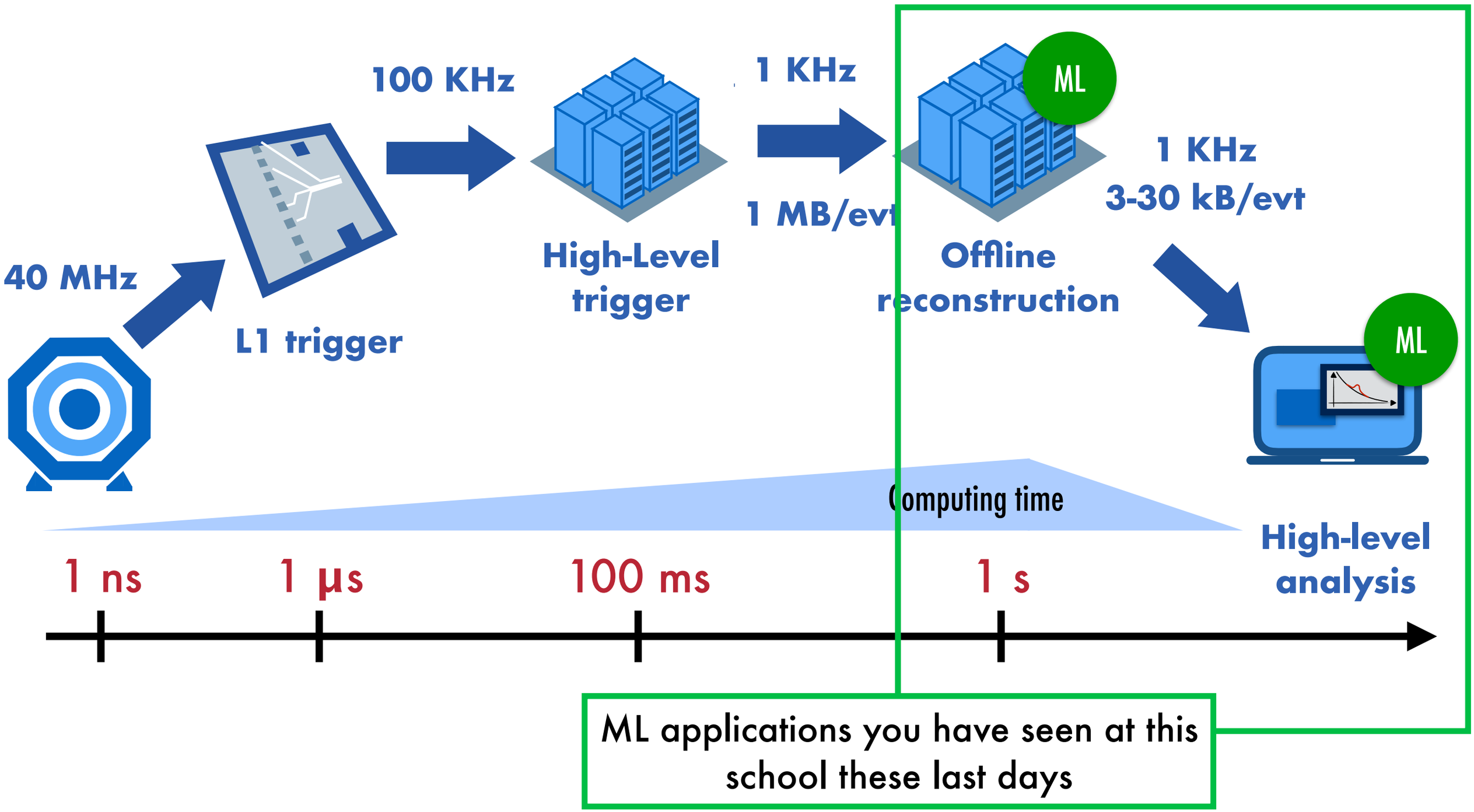
Data volume

Computing time

Information

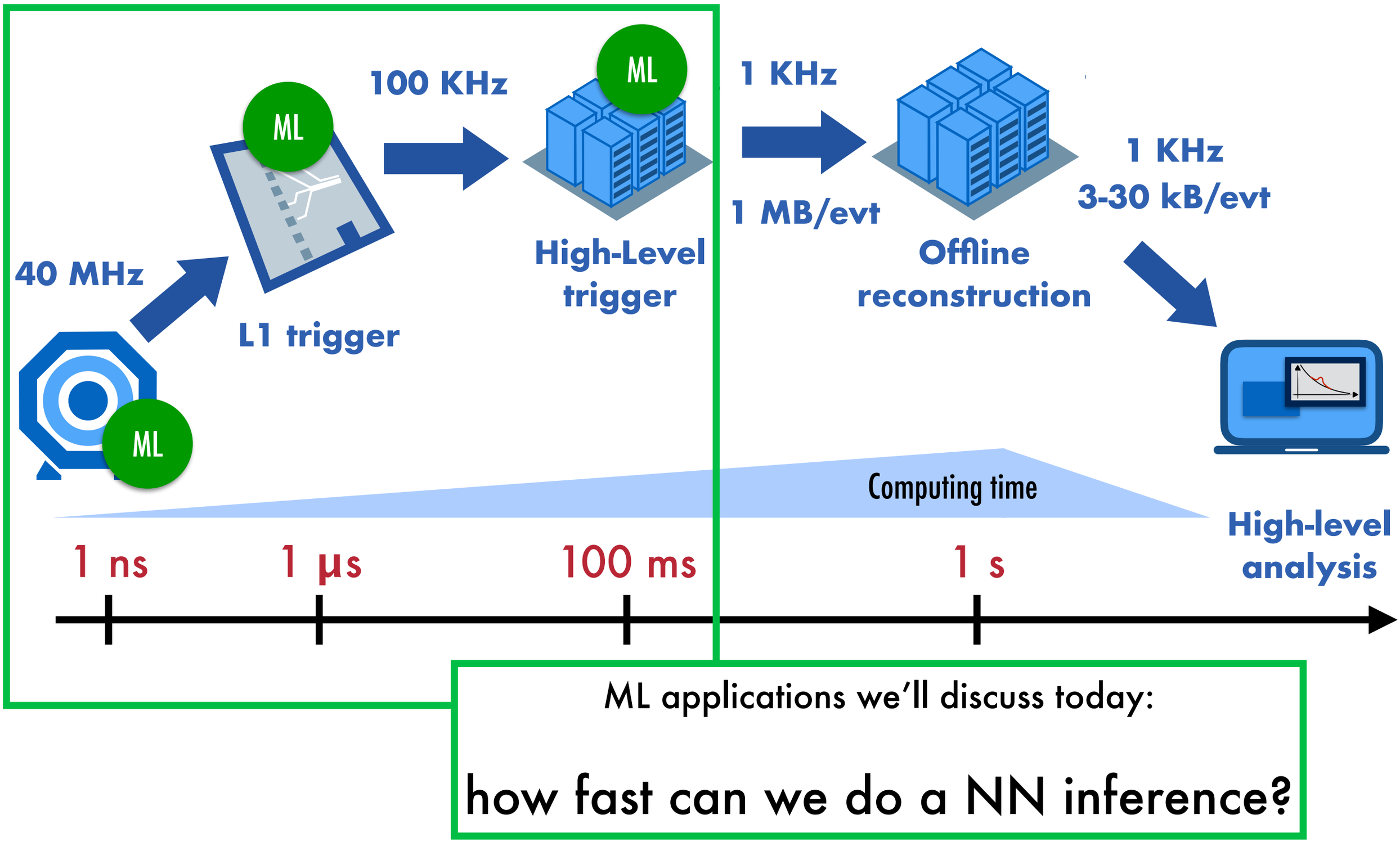
High-level analysis

Ultra fast ML for triggering



We know it works here: jet identification, anomaly detection, objects or event reconstruction and identification, …

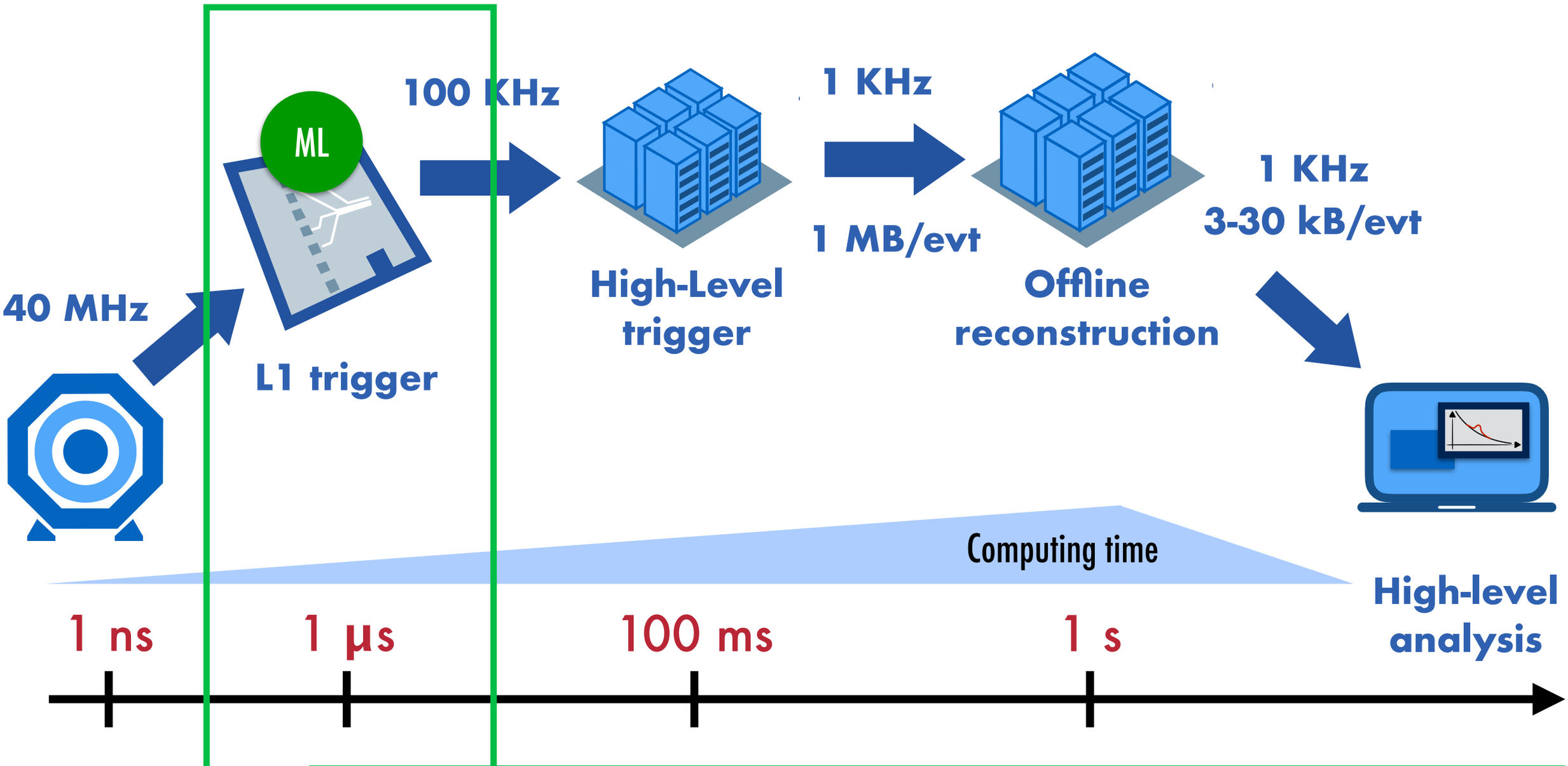
Ultra fast ML for triggering



ML applications we'll discuss today:

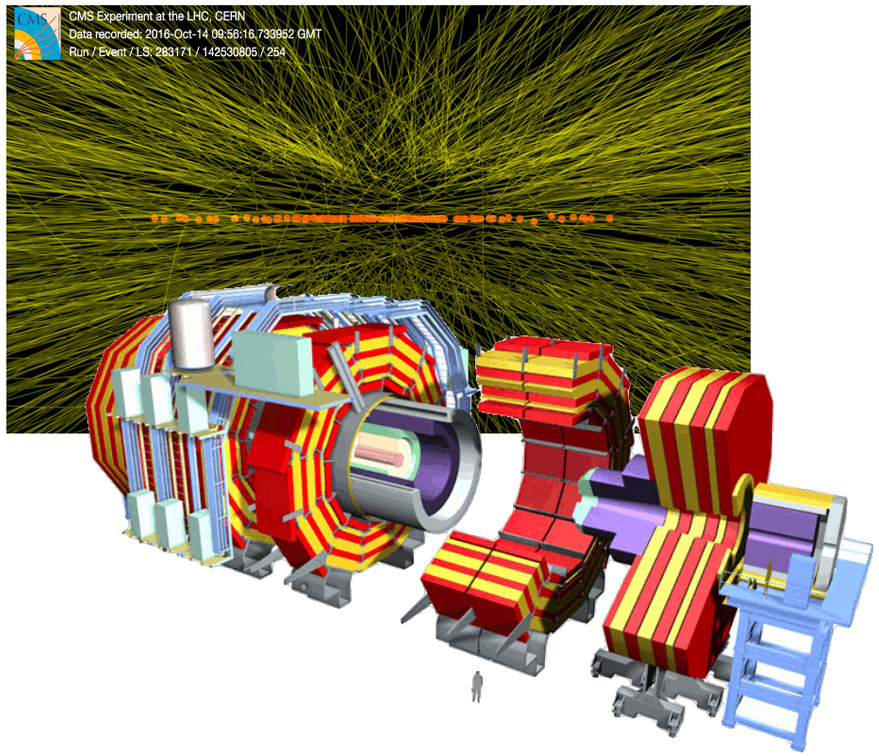
how fast can we do a NN inference?

Ultra fast ML for triggering



Main focus: deploy ML very early in the game
Remember: events not surviving this stage won't get a chance later on!
Goal: improve our algorithms here!

The L1 trigger system

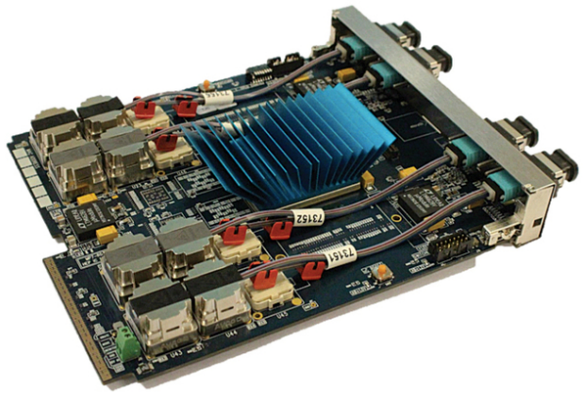


detector front end electronics

detector data

optical links

O(100) Xilinx FPGAs



in: 40 MHz

out: 100 KHz

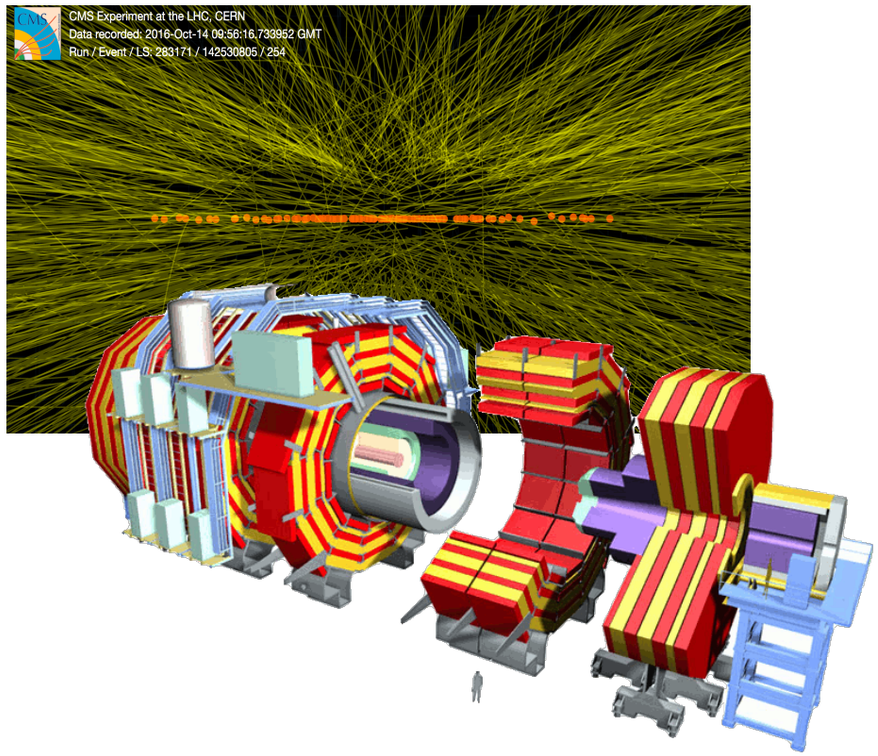
AMCs w/ MicroTCA backplane

Challenge: ultra low latency and scarse resources

<u>Most of it allocated by standard algorithms:</u>

- receive, calibrate, and sort calorimeter energy deposits over the whole detector
- aggregate them to make physics objects (jets, electrons, taus, energy sums)
- run track finders combining hits in muon stations

The L1 trigger system

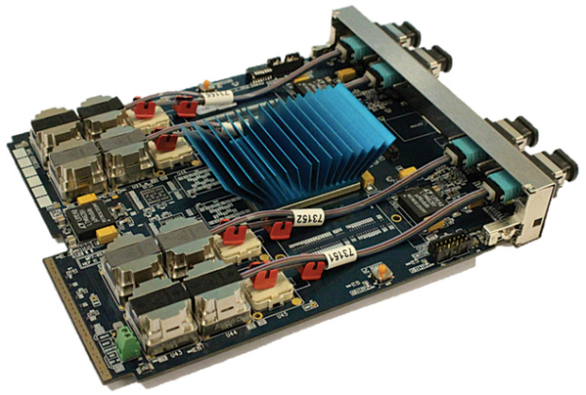


detector front end electronics

detector data

optical links

O(100) Xilinx FPGAs



in: 40 MHz

out: 100 KHz

AMCs w/ MicroTCA backplane

Challenge: ultra low latency and scarse resources

Most of it allocated

- receive, calibrate … the whole detector
- aggregate them … , energy sums)
- run track finders combining hits in muon stations

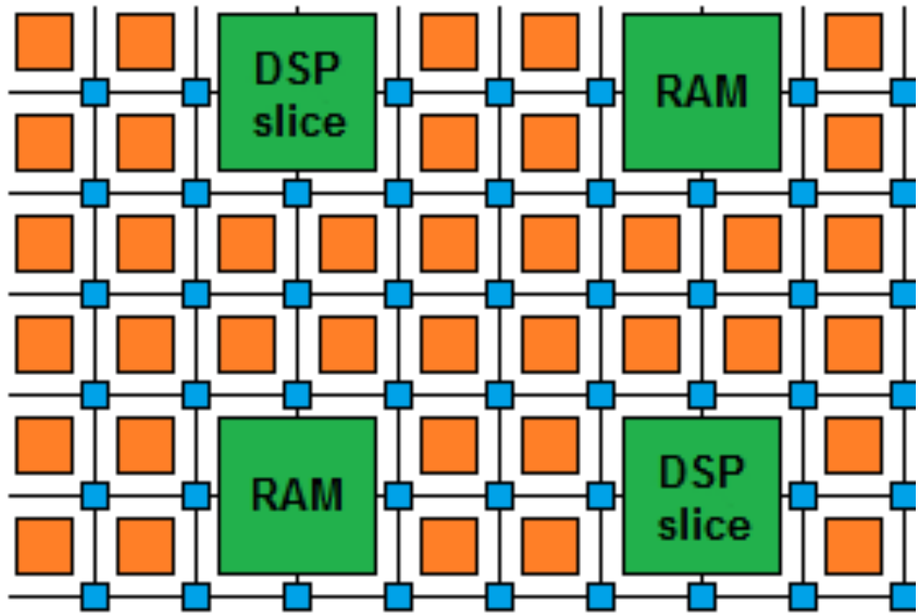
how to fit ML models here?

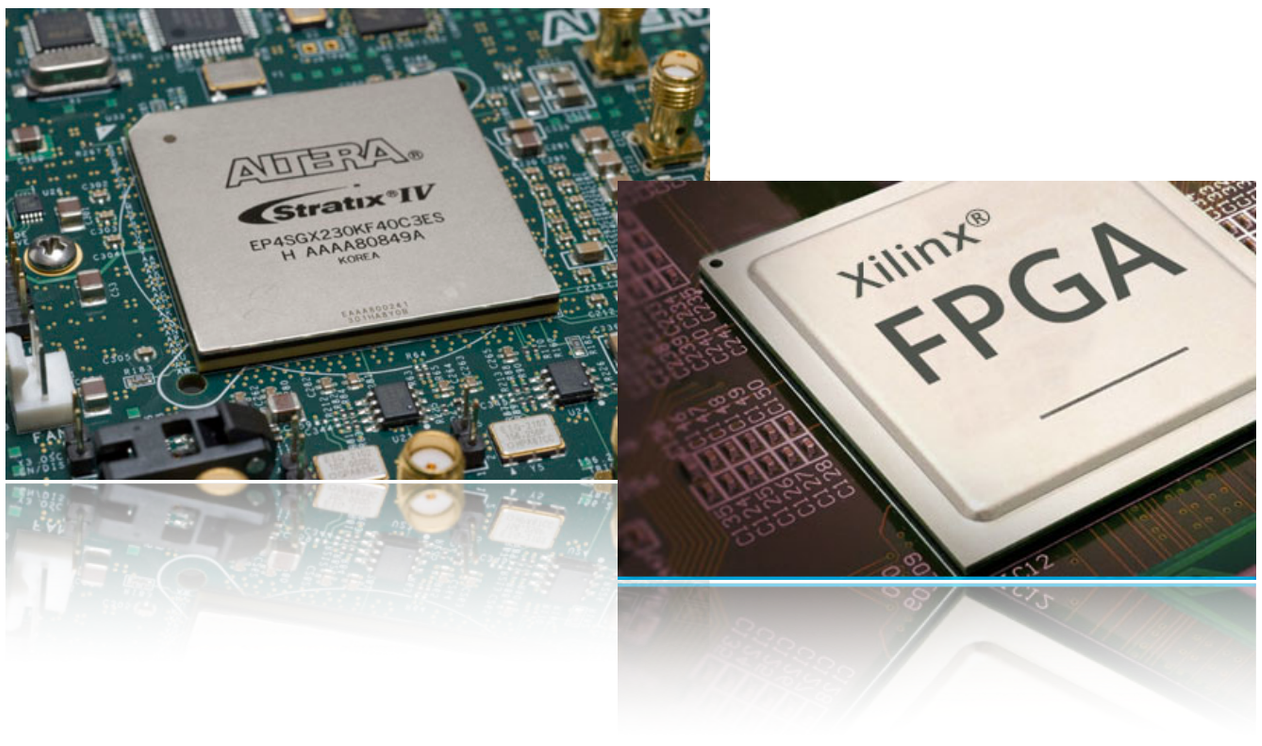
What are FPGAs?

Field Programmable Gate Arrays
are reprogrammable integrated circuits

Contain many different building blocks ('resources') which are connected together as you desire

FPGA diagram



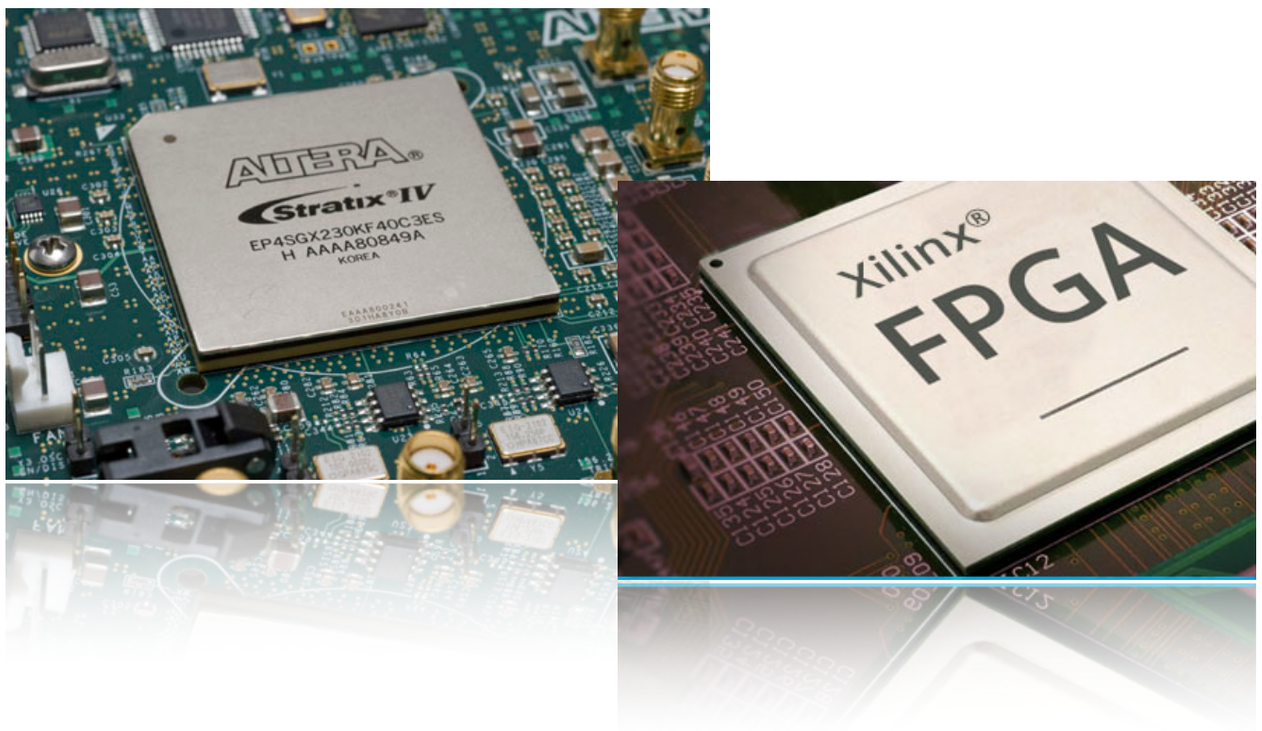


What are FPGAs?

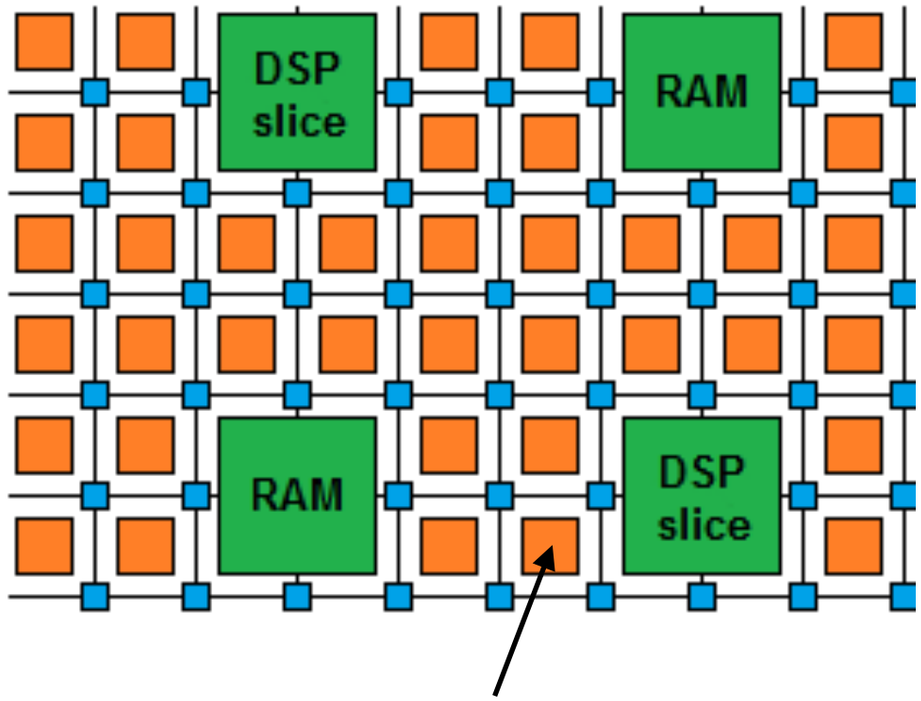
Field Programmable Gate Arrays
are reprogrammable integrated circuits

Look Up Tables (LUTs) perform arbitrary functions on small bitwidth inputs (2-6 bits)
→ used for boolean operations, arithmetics, memory

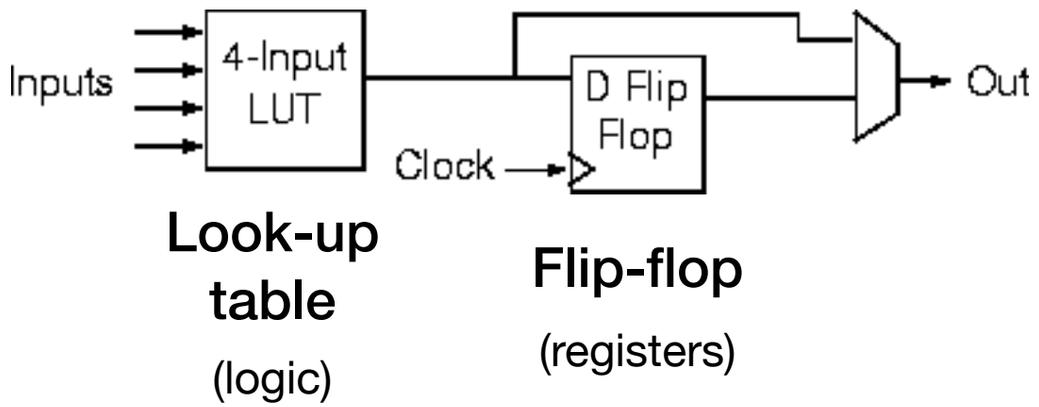
Flip-flops register data in time with the clock pulse



FPGA diagram



Logic cell



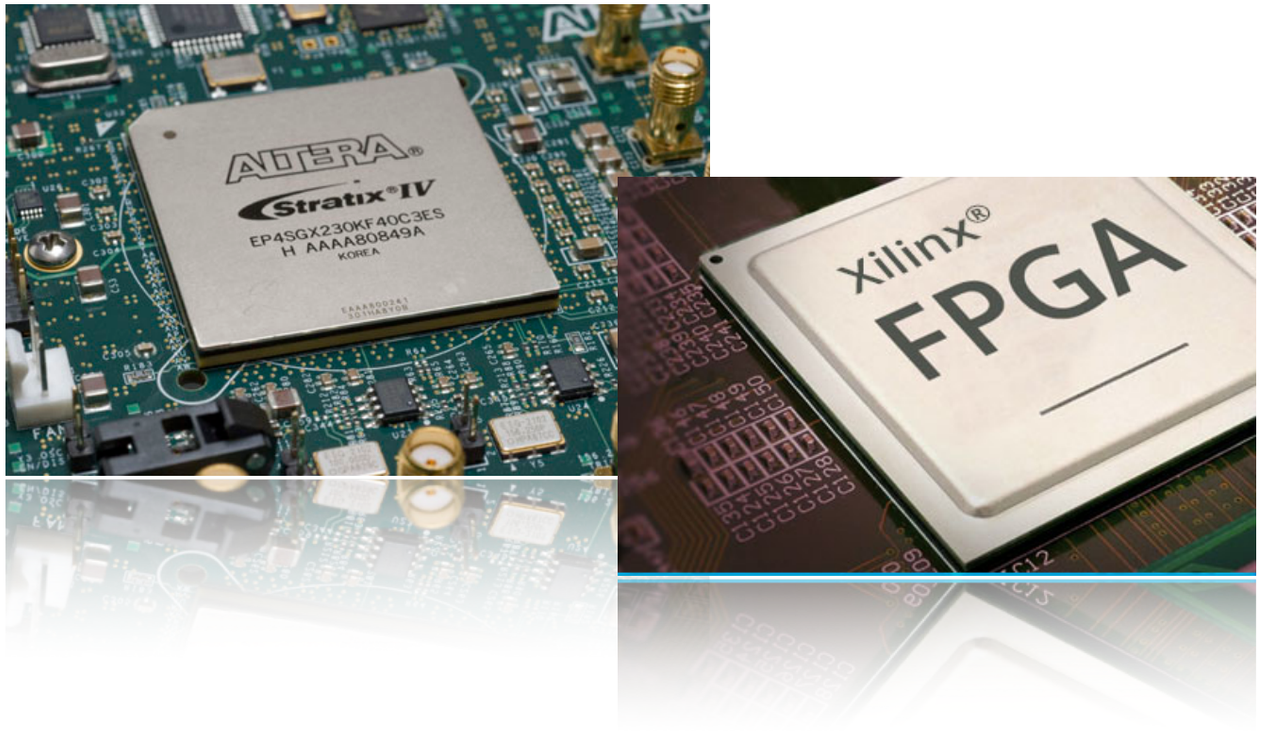
Look-up table
(logic)

Flip-flop
(registers)

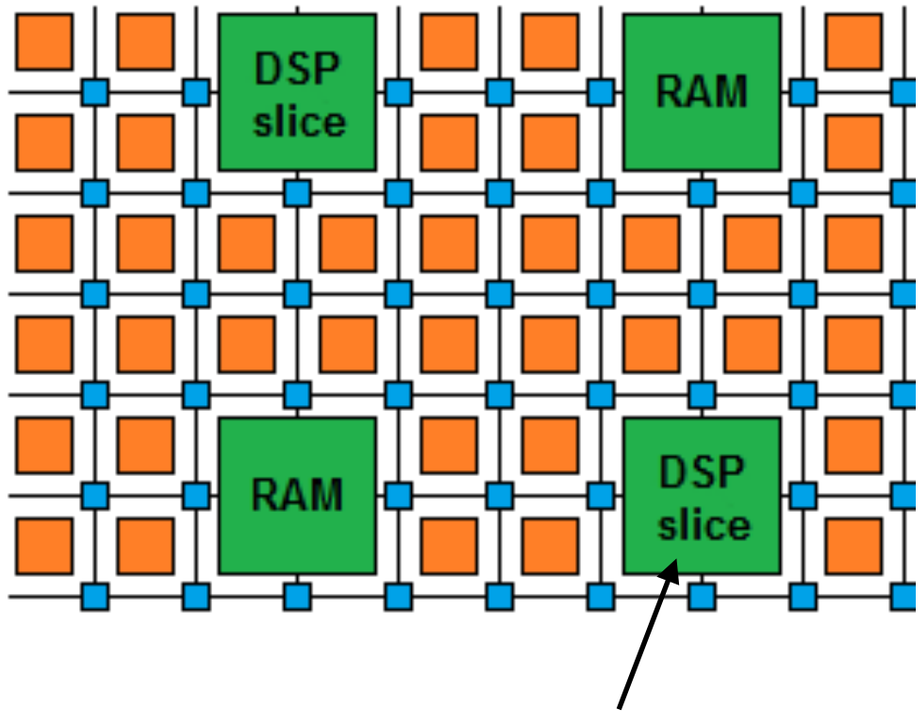
What are FPGAs?

Field Programmable Gate Arrays
are reprogrammable integrated circuits

DSPs are specialized units for multiplication and arithmetic
→ faster and more efficient than LUTs for these type of operations
→ for deep learning, they are often the most precious resource



FPGA diagram



Also contain embedded components:

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

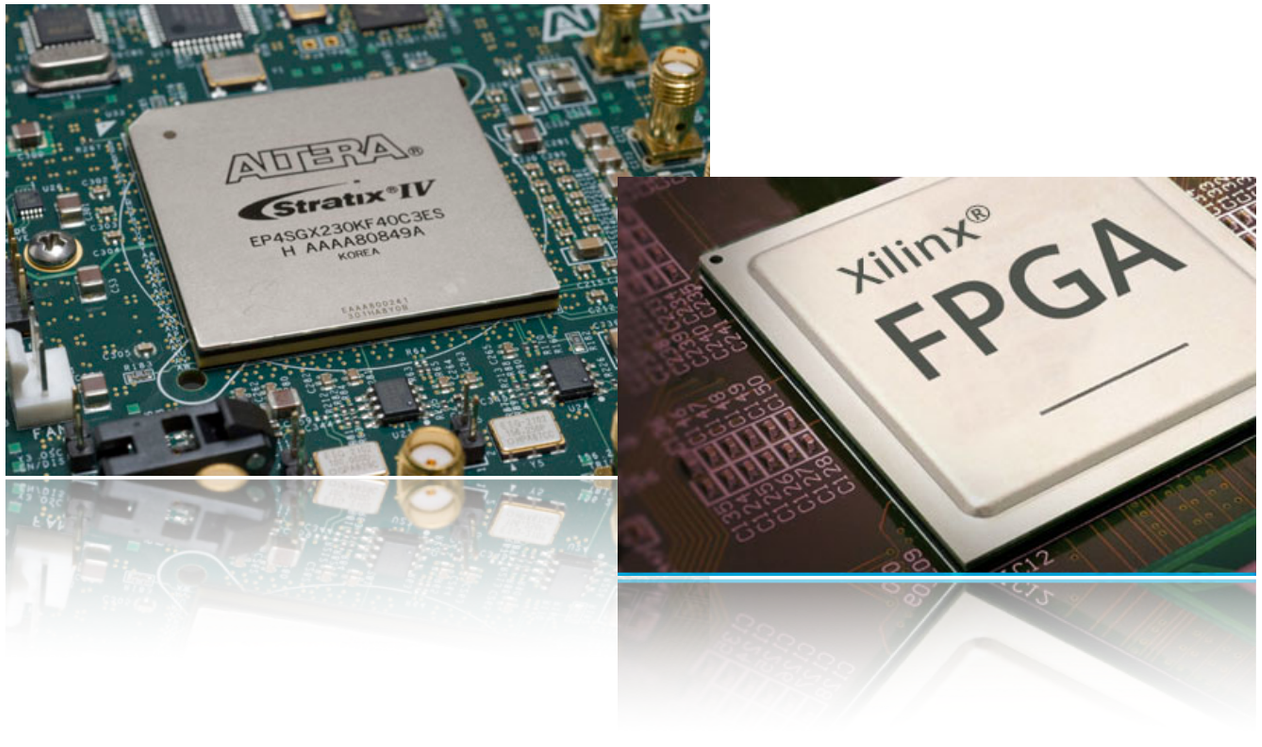
What are FPGAs?

Field Programmable Gate Arrays
are reprogrammable integrated circuits

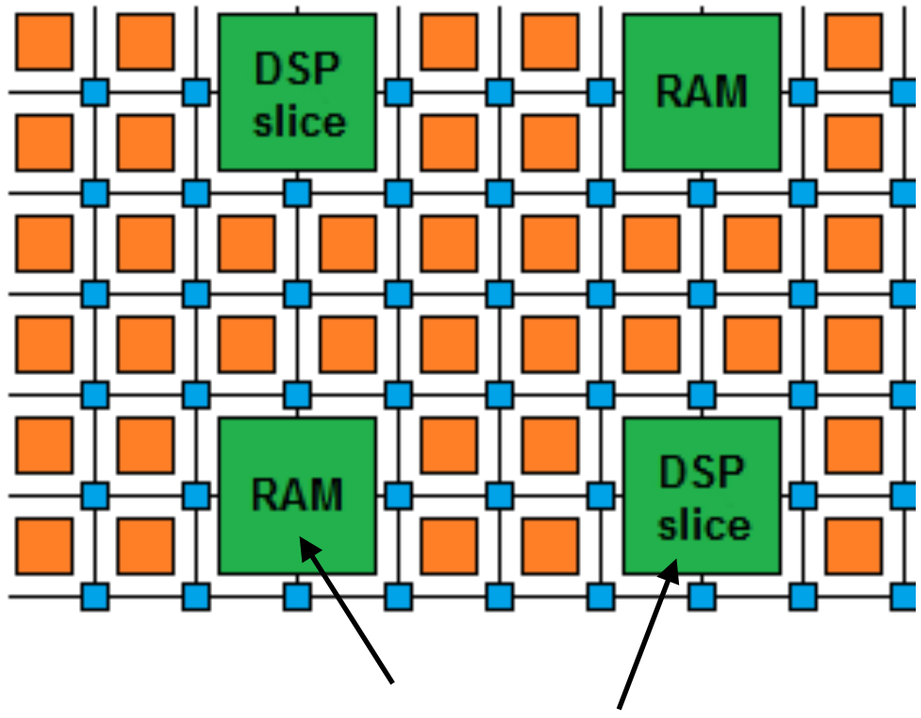
BRAMs are small, fast memories (ex, 18 Kb each)

→ more efficient than LUTs when large memory is required

Modern FPGAs have ~100 Mb of BRAMs, chained together as needed



FPGA diagram



Also contain embedded components:

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

Random-access memories (RAMs):
embedded memory elements

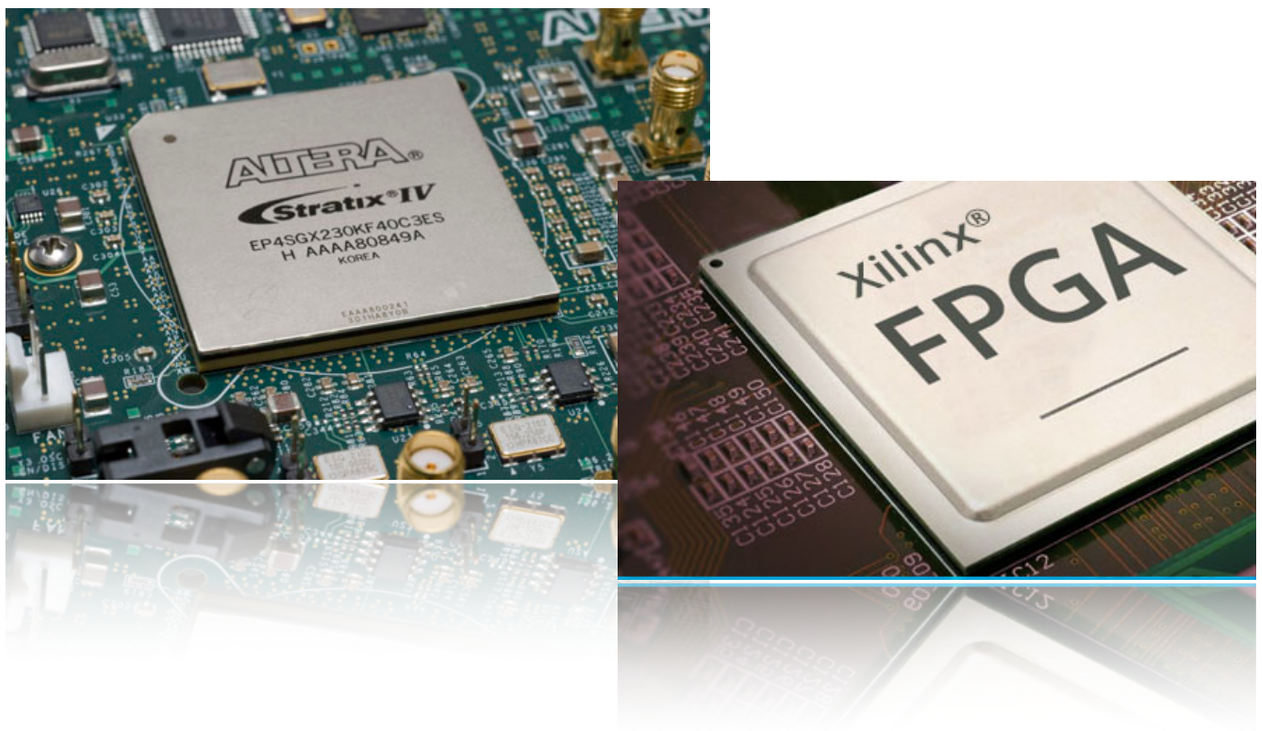
What are FPGAs?

Field Programmable Gate Arrays
are reprogrammable integrated circuits

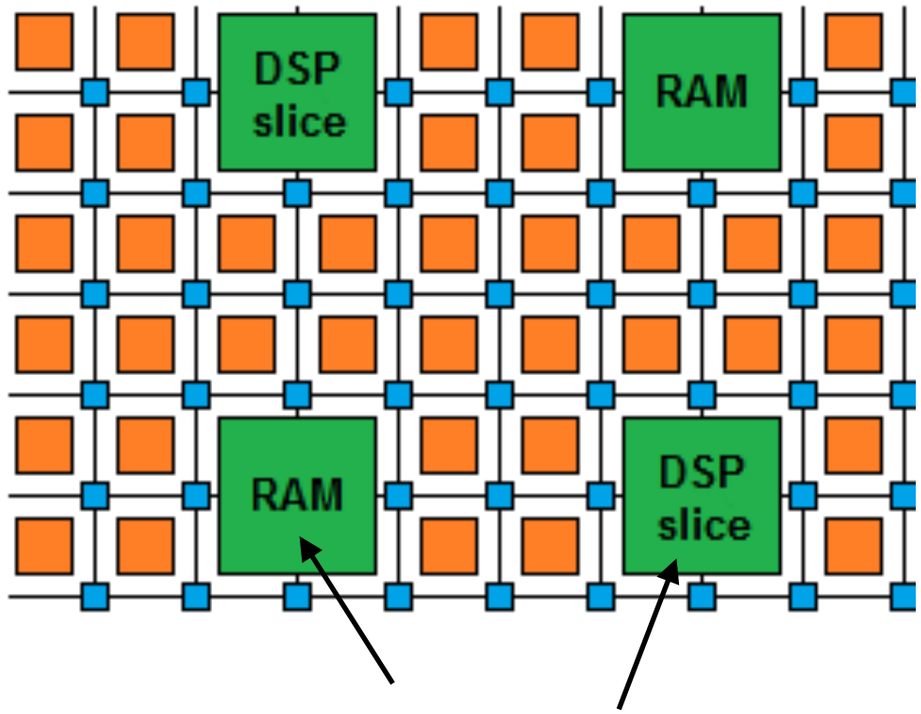
Contain array of **logic cells** embedded with **DSPs**, **BRAMs**, etc.

Support **highly parallel** algorithm implementation

Low power per Op (relative to CPU/GPU)



FPGA diagram



Also contain embedded components:

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

Random-access memories (RAMs):
embedded memory elements

Why are FPGAs fast?

- Fine-grained / resource parallelism
  - use the many resources to work on different parts of the problem simultaneously
  - allows us to achieve **low latency**
- Most problems have at least some sequential aspect, limiting how low latency we can go
  - but we can still take advantage of it with…
- **Pipeline parallelism**
  - instruct the FPGA to work on different data simultaneously
  - allows us to achieve **high throughput**

Like a production line for data…

How are FPGAs programmed?

Hardware Description Languages

HDLs are programming languages which describe electronic circuits

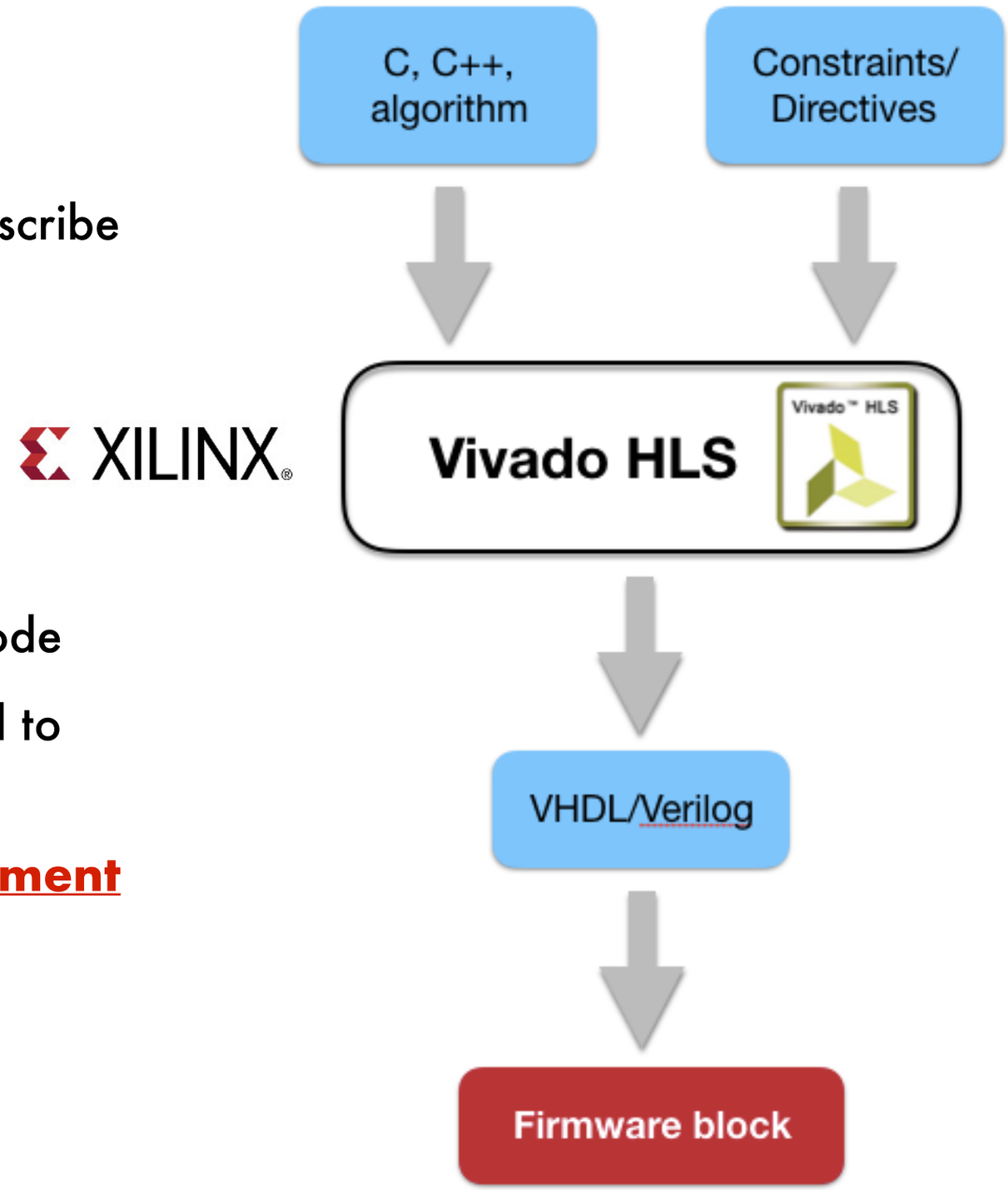
High Level Synthesis

generate HDL from more common C/C++ code

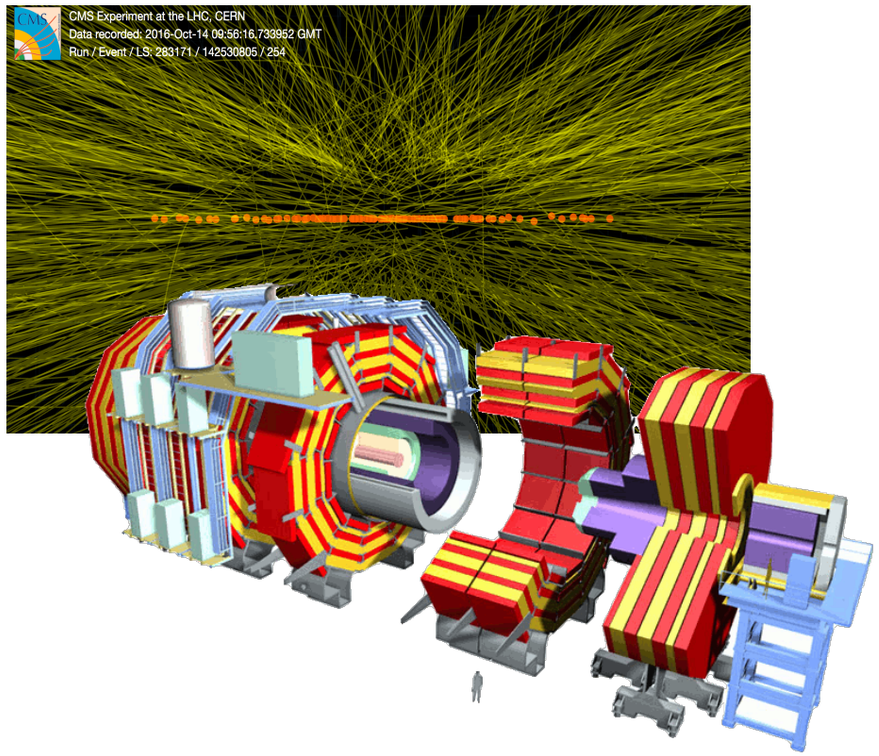
pre-processor directives and constraints used to optimize the timing

drastic decrease in firmware development time!

See Xilinx Vivado HLS, Intel HLS, Catapult HLS



The L1 trigger system

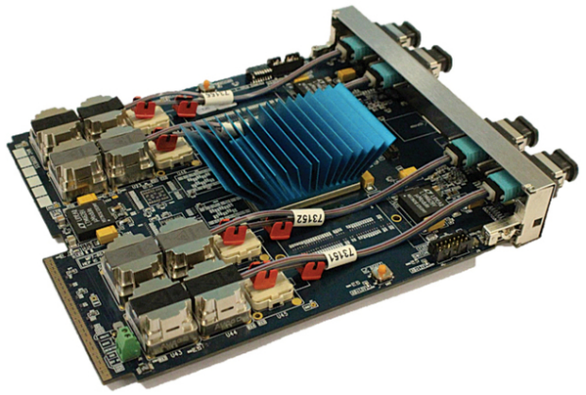


detector front end electronics

detector data

optical links

O(100) Xilinx FPGAs



in: 40 MHz

out: 100 KHz

AMCs w/ MicroTCA backplane

Challenge: ultra low latency and scarse resources

Most of it allocated

- receive, calibrate … the whole detector
- aggregate them … , energy sums)
- run track finders combining hits in muon stations

how to fit ML models here?

Bring DL to FPGA for L1 trigger with

high level synthesis for machine learning

PYTORCH

Keras
TensorFlow
PyTorch
...

model

compressed model

Usual ML
software workflow

Vivado™ HLS

hls4ml

Co-processing kernel

HLS conversion

HLS project

Custom firmware design

tune configuration
precision
reuse/pipeline

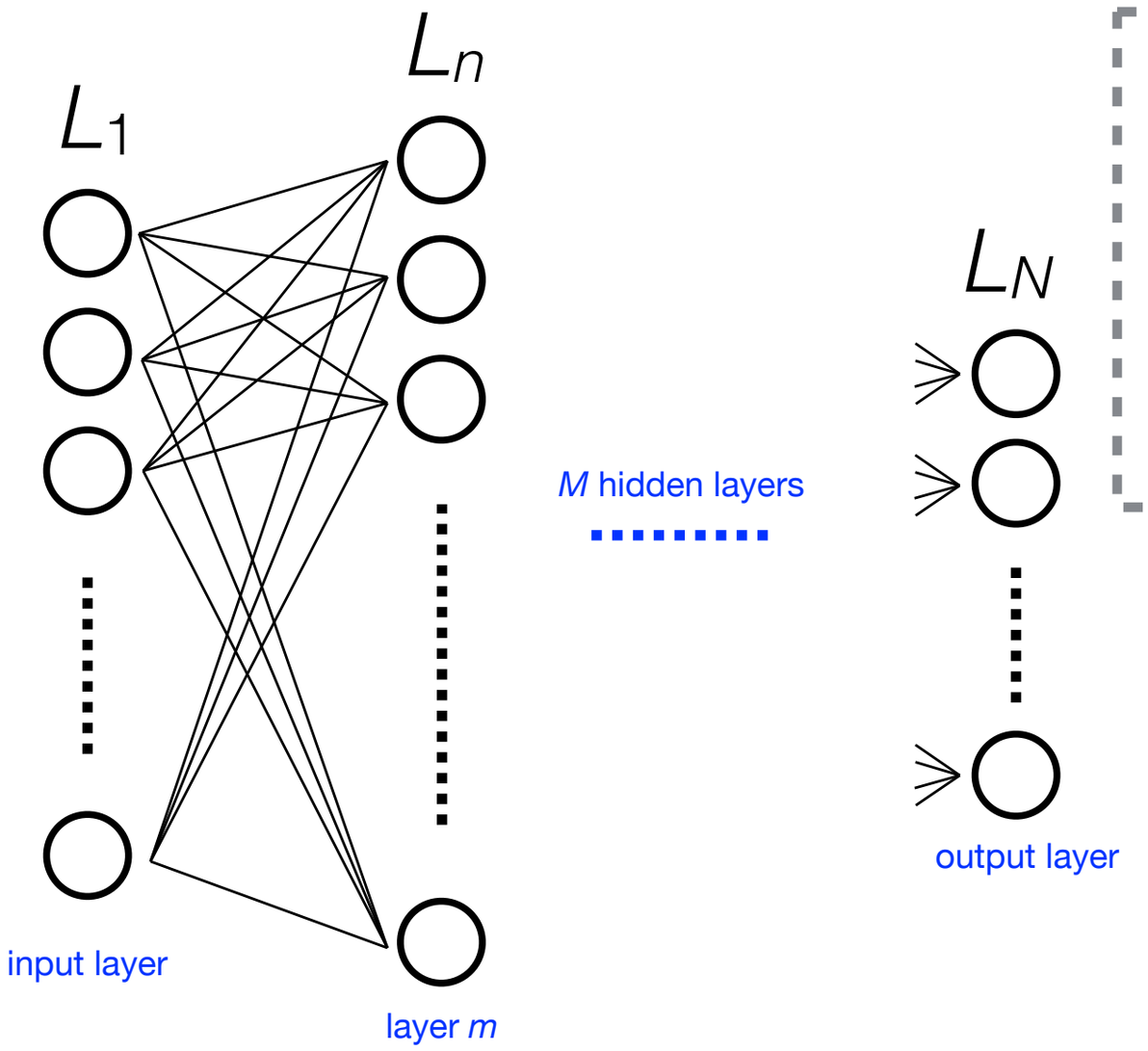
K + TensorFlow

ONNX

arxiv.1804.06913

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

Neural network inference



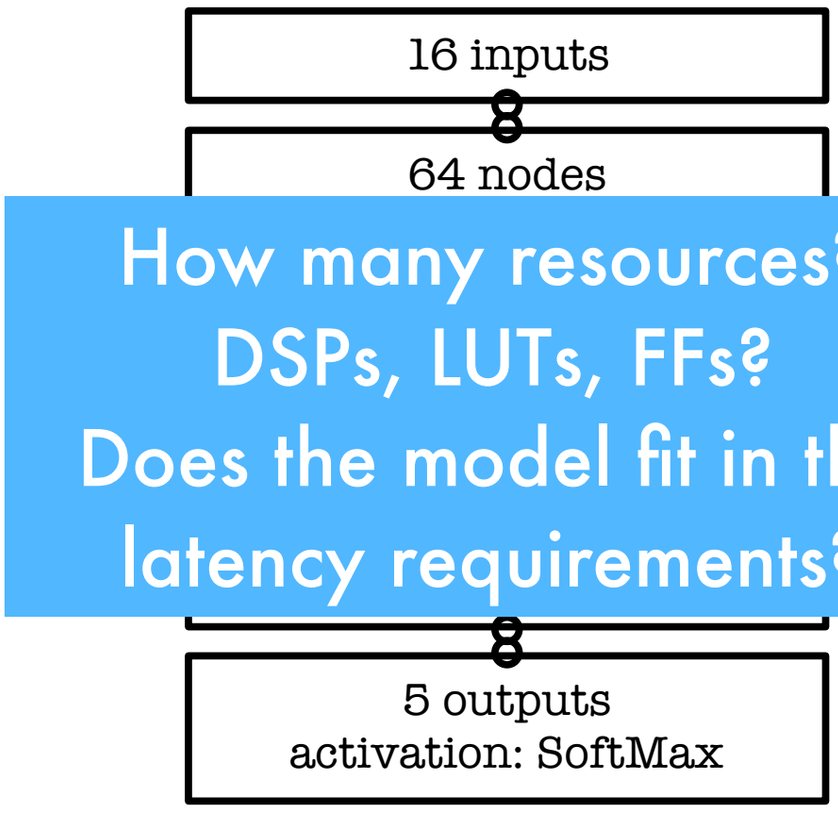
$$\mathbf{x}_n = g_n(\mathbf{W}_{n,n-1}\mathbf{x}_{n-1} + \mathbf{b}_n)$$

activation function
precomputed and stored in BRAMs

multiplication
DSPs

addition
logic cells

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



How many resources?
DSPs, LUTs, FFs?
Does the model fit in the latency requirements?

How to fit ML on one FPGA?

FPGAs provide huge flexibility

Performance depends on how well you take advantage of this

Constraints:
Input bandwidth
FPGA resources
Latency

<u>Today you will learn how to optimize your project through:</u>

- **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

Today's hls4ml hands on

- **Part 1:** get started with hls4ml and train a basic model and run the conversion, simulation & c-synthesis steps

  `notebook: part1_getting_started.ipynb`

- **Part 2:** learn how to tune inference performance with quantization and reuse factor

  `notebook: part2_advanced_config.ipynb`

- **Part 3:** perform model compression and observe its effect on the FPGA resources/ latency

  `notebook: part3_compression.ipynb`

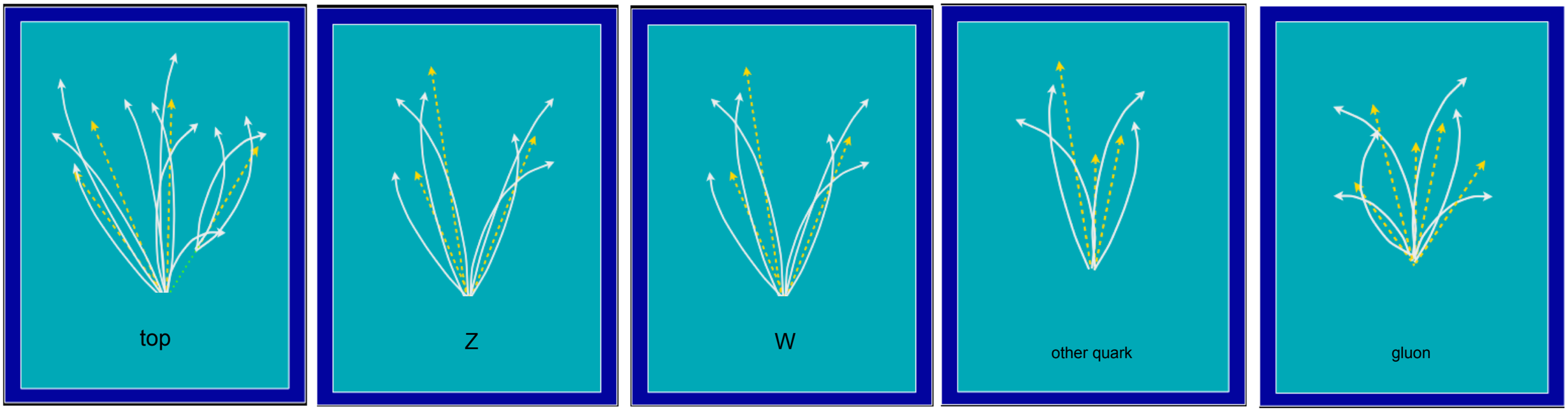
- **Part 4:** train using QKeras "quantization aware training" and study impact on FPGA metrics

  `notebook: part4_quantization.ipynb`

Part 1: model conversion

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 — 3-prong jet

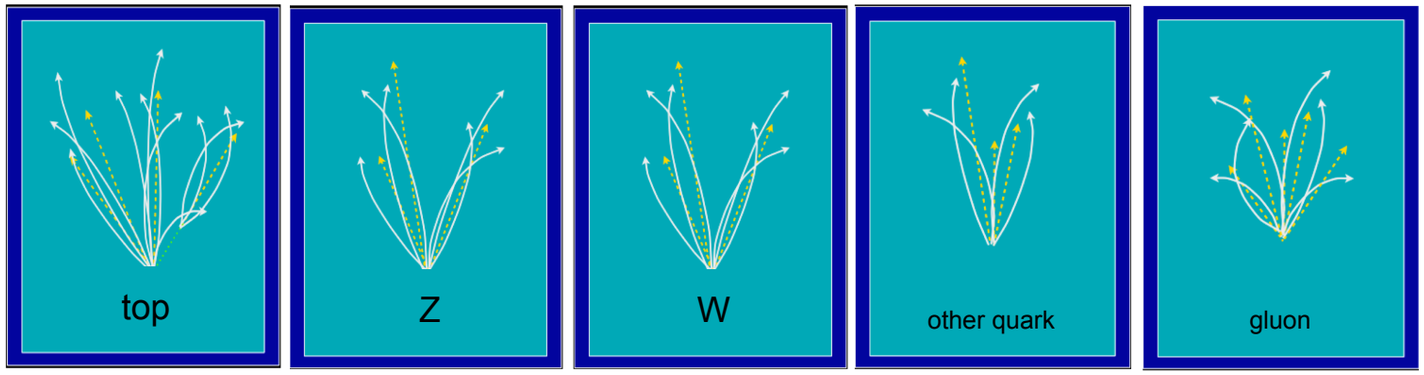
Z→qq — 2-prong jet

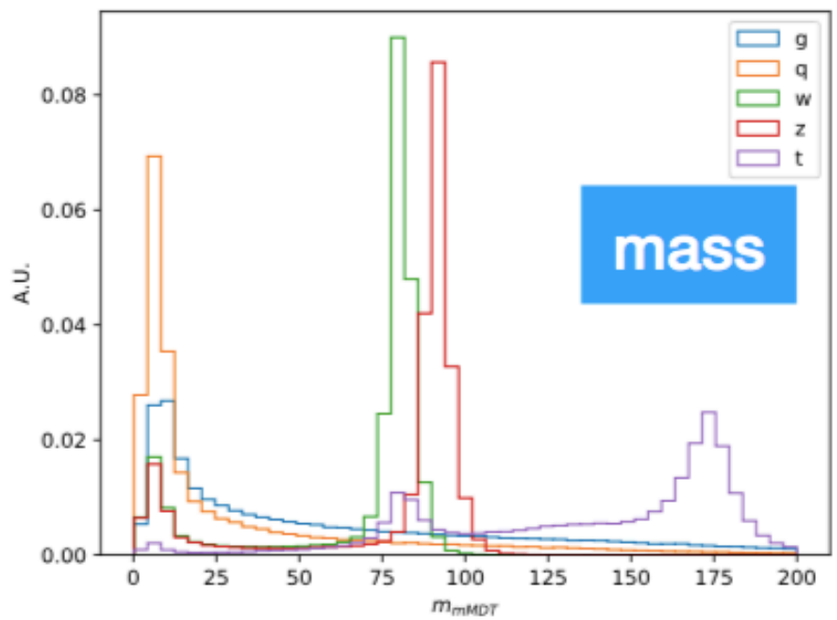
W→qq — 2-prong jet

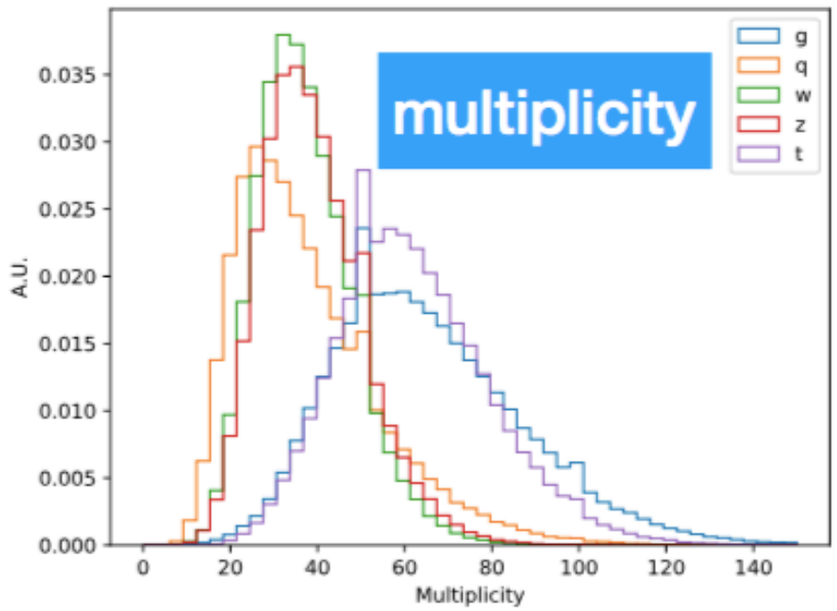
q/g background — no substructure and/or mass ~ 0

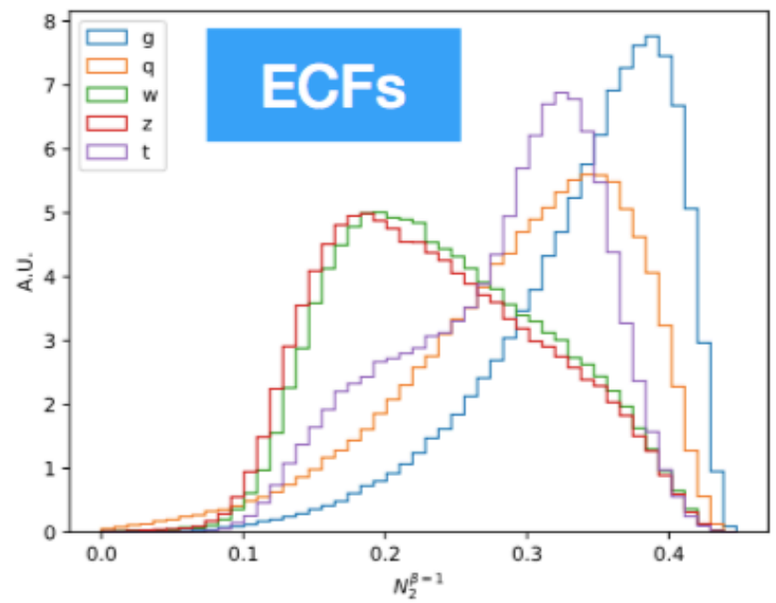
Reconstructed as one massive jet with substructure

Physics case: jet tagging









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

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

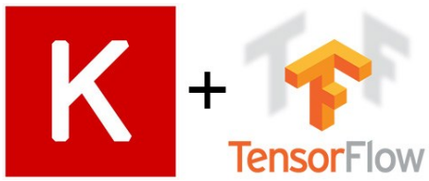
[*] D. Guest at al. PhysRevD.94.112002, G. Kasieczka et al. JHEP05(2017)006, J. M. Butterworth et al. PhysRevLett.100.242001, etc..

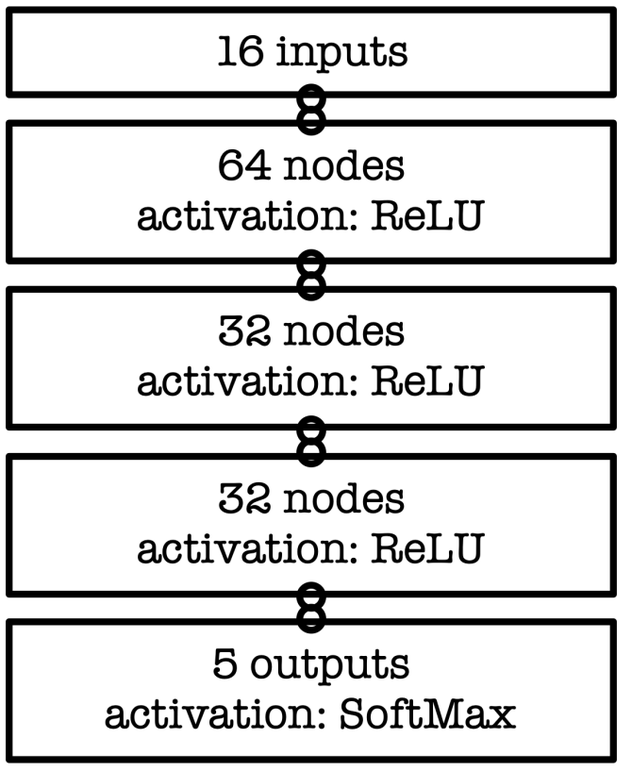
Physics case: jet tagging

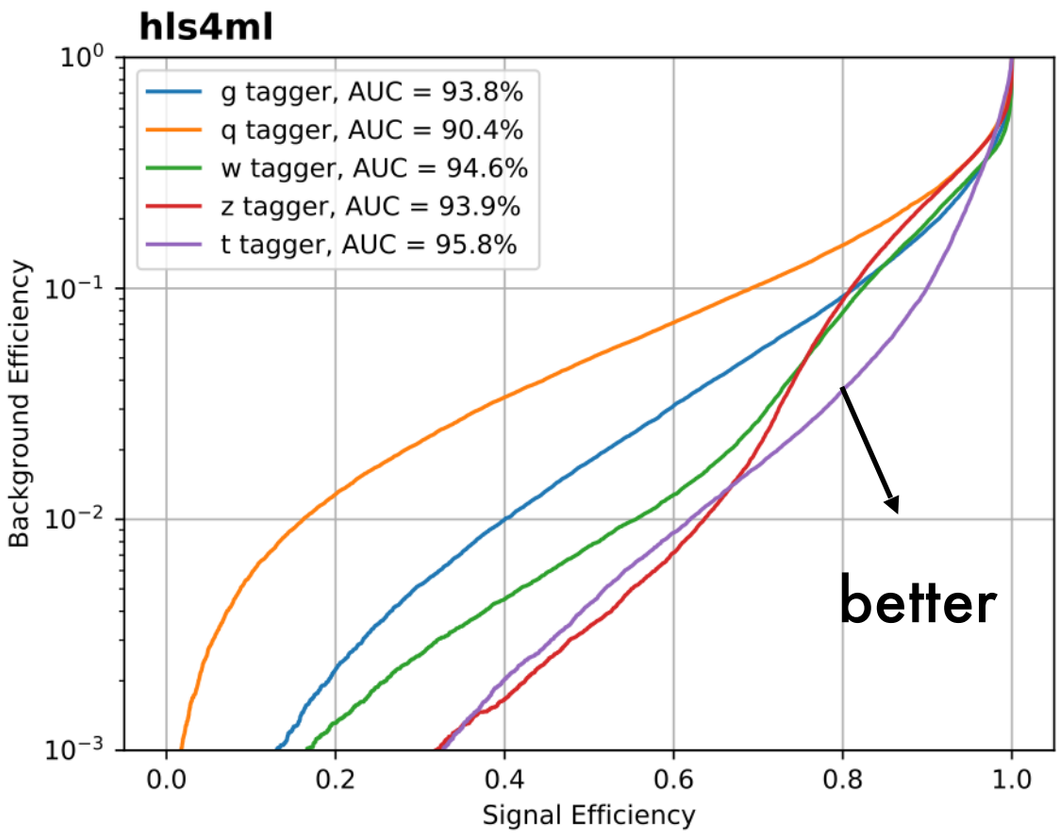
- We'll train the five class multi-classifier on a sample of ~1M events with two boosted WW/ZZ/tt/qq/gg anti-k_T jets

  [doi:10.5281/zenodo.3602254, OpenML]

- Fully connected neural network with 16 expert-level inputs:
  - Relu activation function for intermediate layers
  - Softmax activation function for output layer







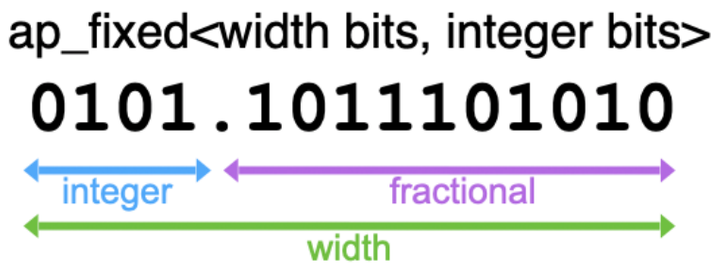
AUC = area under ROC curve
(100% is perfect, 20% is random)

Setup

- The interactive part is served with Python notebooks
- Open https://cern.ch/ssummers/hls4ml-tutorial in your web browser
- Authenticate with your Github account (login if necessary)
- Open and start running through `part1_getting_started.ipynb`
- If you're new to Jupyter notebooks, select a cell and hit "shift + enter" to execute the code
- If you have Vivado installed, you might prefer to work locally, see 'conda' section at: https://github.com/fastmachinelearning/hls4ml-tutorial

Part 2: advanced configuration

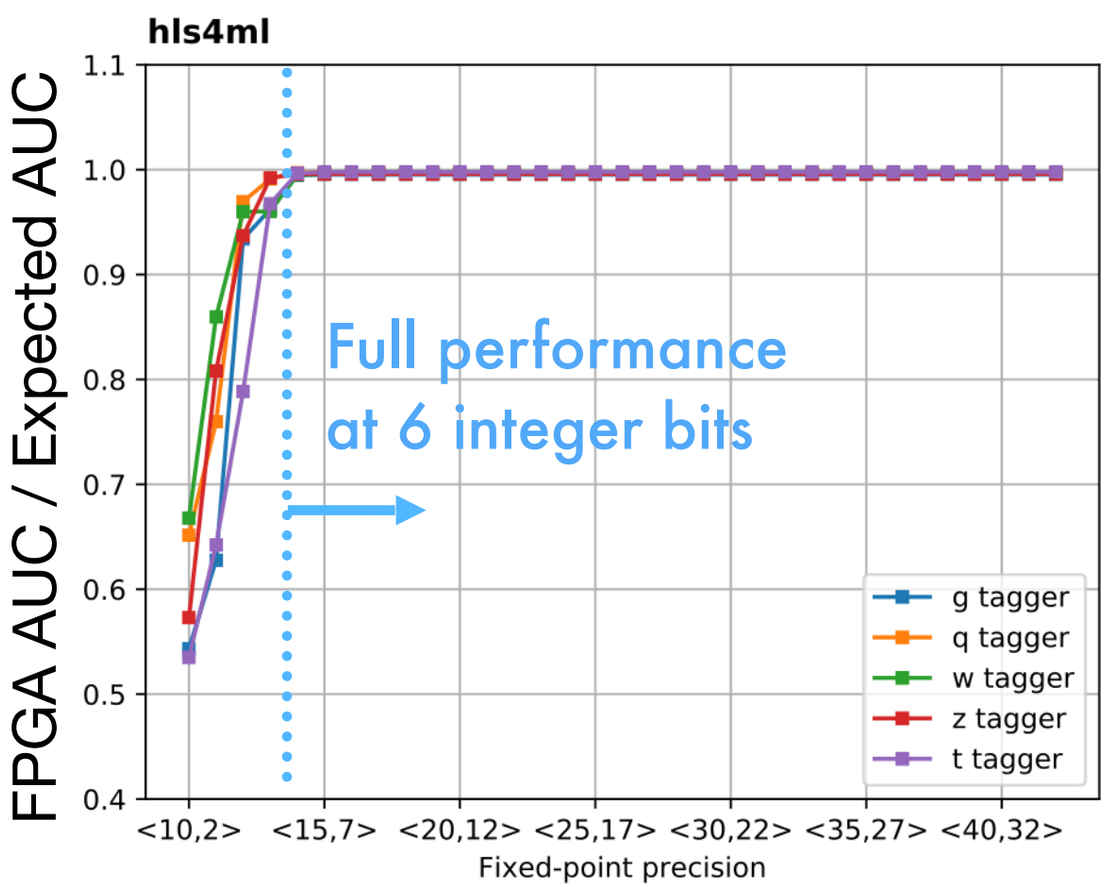
Efficient NN design: quantization



- In the FPGA we use fixed point representation
  - operations are integer ops, but we can represent fractional values
- But we have to make sure we've used the correct data types!

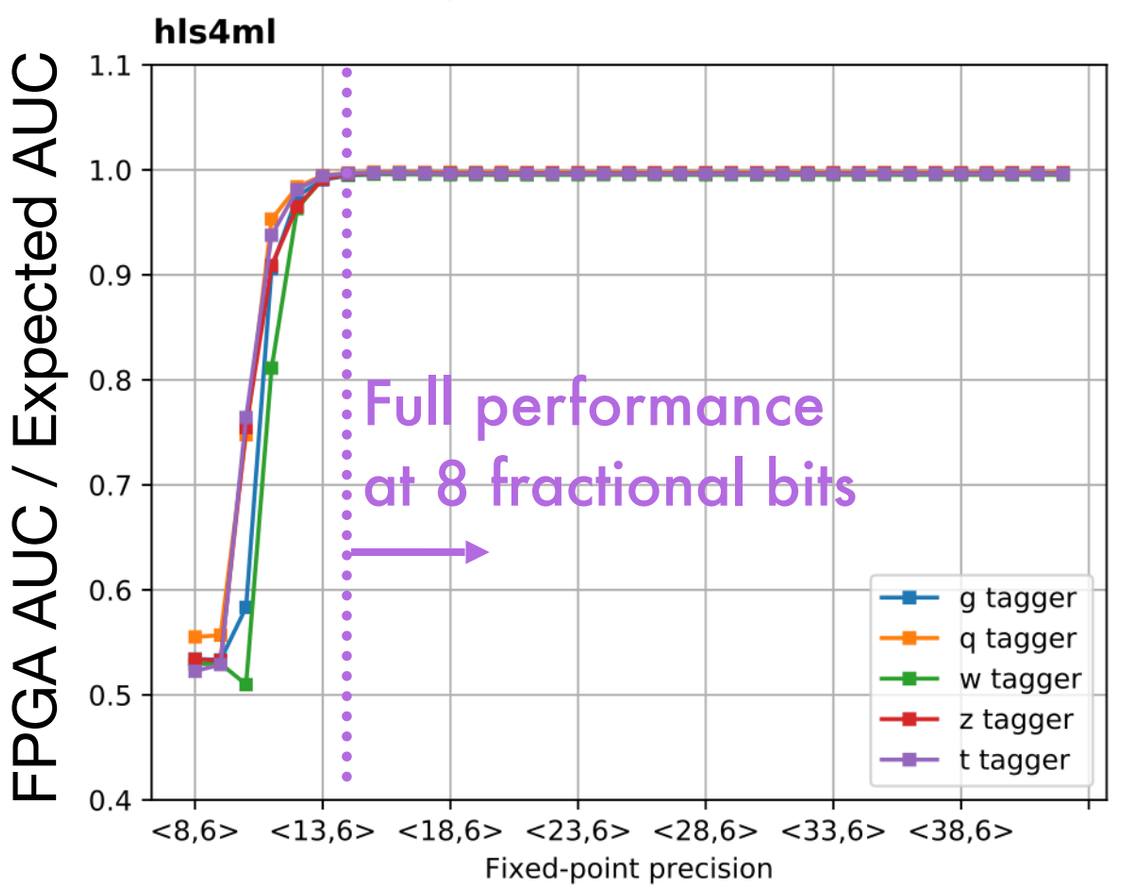
Scan integer bits

Fractional bits fixed to 8



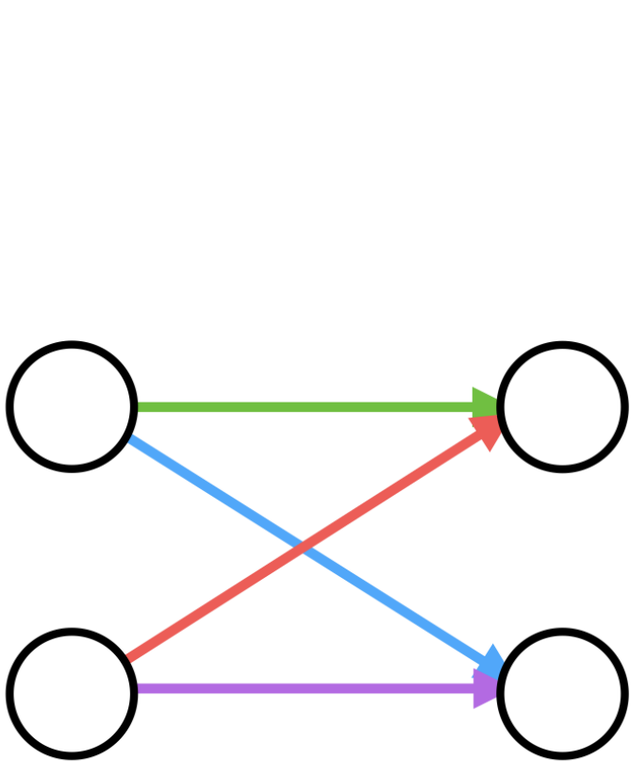
Scan fractional bits

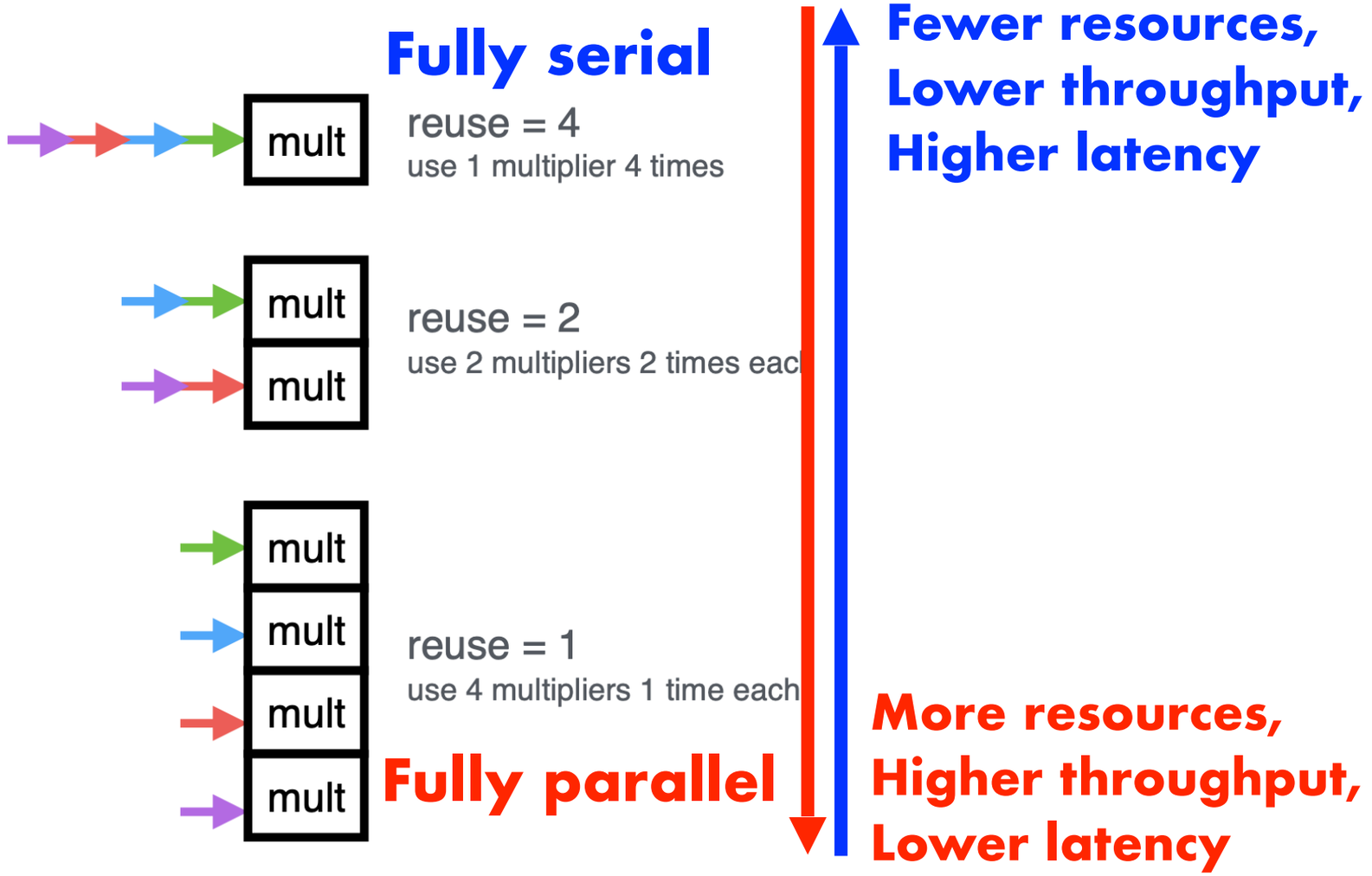
Integer bits fixed to 6



Efficient NN design: parallelization

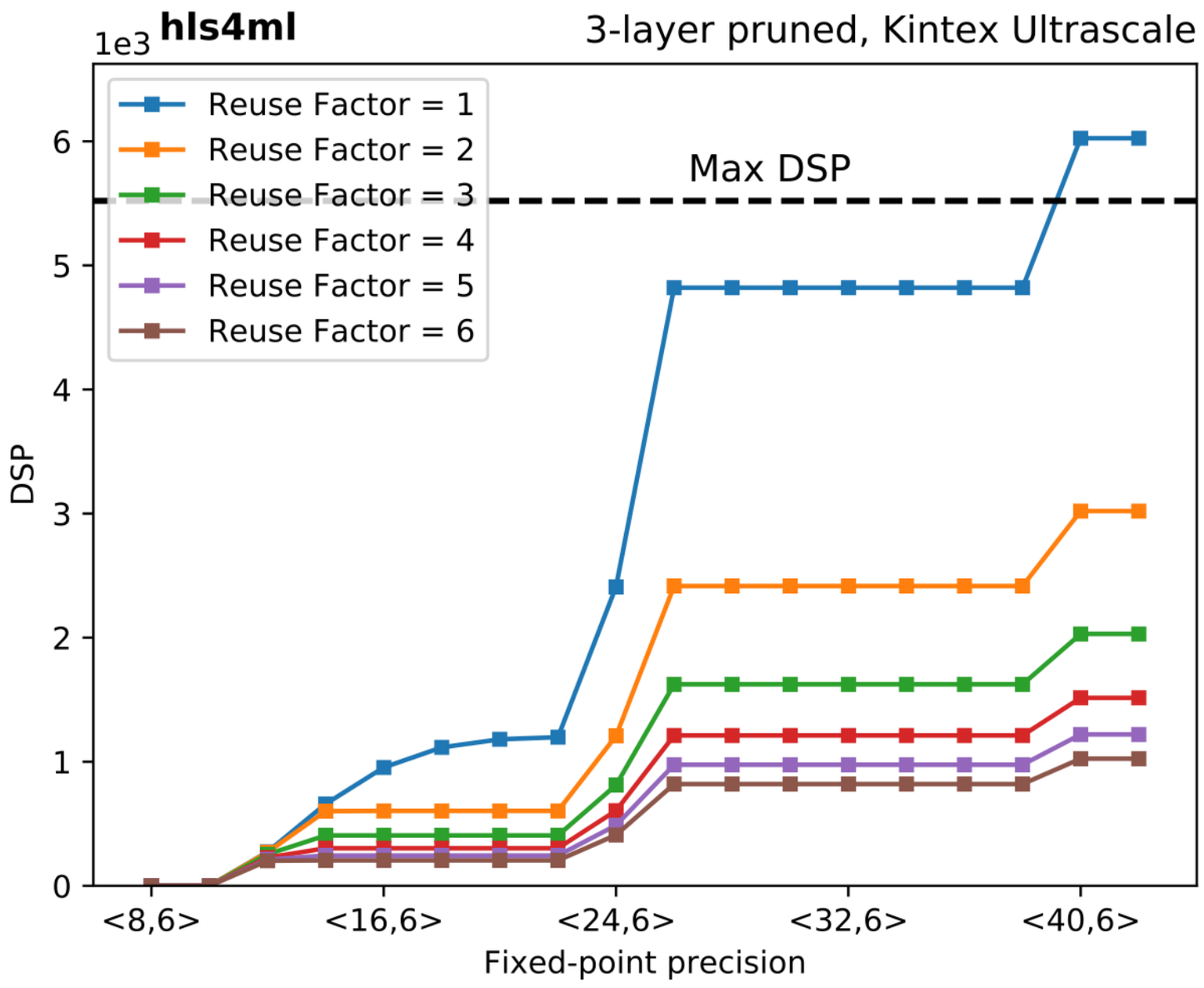
- Trade-off between latency and FPGA resource usage determined by the parallelization of the calculations in each layer
- Configure the "`reuse factor`" = number of times a multiplier is used to do a computation





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

Parallelization: DSPs



More resources

Fully parallel
Each mult. used 1x

Each mult. used 2x

Each mult. used 3x

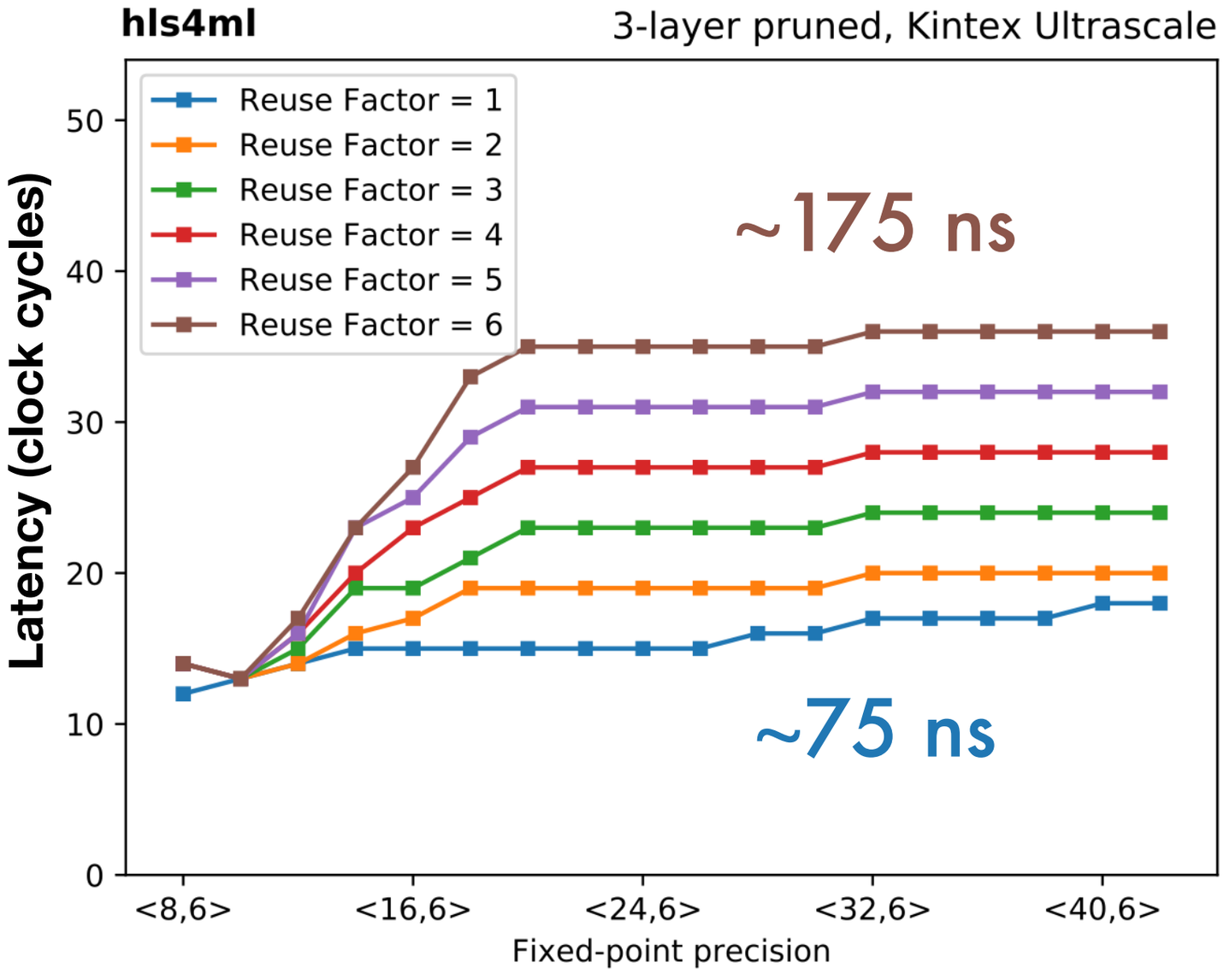
⋮

Longer latency

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

Large fully-connected NN

- `'Strategy: Resource'` for larger networks and higher reuse factor
- Uses a slightly different HLS implementation of the dense layer to compile faster and better for large layers
- Here, we use a different partitioning on the first layer for the best partitioning of arrays

```
IOType: io_parallel
HLSConfig:
  Model:
    Precision: ap_fixed<16,6>
    ReuseFactor: 128
  Strategy: Resource
LayerName:
   dense1:
     ReuseFactor: 112
```

This config is for a model trained on the MNIST digits classification dataset
Architecture (fully connected): 784 → 128 → 128 → 128 → 10
Model accuracy:~97%
We can work out how many DSPs this should use…

Large fully-connected NN

- It takes a while to synthesise, so here's one I made earlier…
- The DSPs should be: (784 x 128) / 112 + (2 x 128 x 128 + 128 x 10) / 128 = 1162 🤞

```
================================
================================
+ Timing (ns):
    * Summary:
    +--------+-------+----------+------------+
    |  Clock | Target| Estimated| Uncertainty|
    +--------+-------+----------+------------+
    |ap_clk  |   5.00|     4.375|        0.62|
    +--------+-------+----------+------------+

+ Latency (clock cycles):
    * Summary:
    +-----+-----+-----+-----+----------+
    |  Latency  |  Interval | Pipeline |
    | min | max | min | max |   Type   |
    +-----+-----+-----+-----+----------+
    |  518|  522|  128|  128| dataflow |
    +-----+-----+-----+-----+----------+
```

II determined by the largest reuse factor

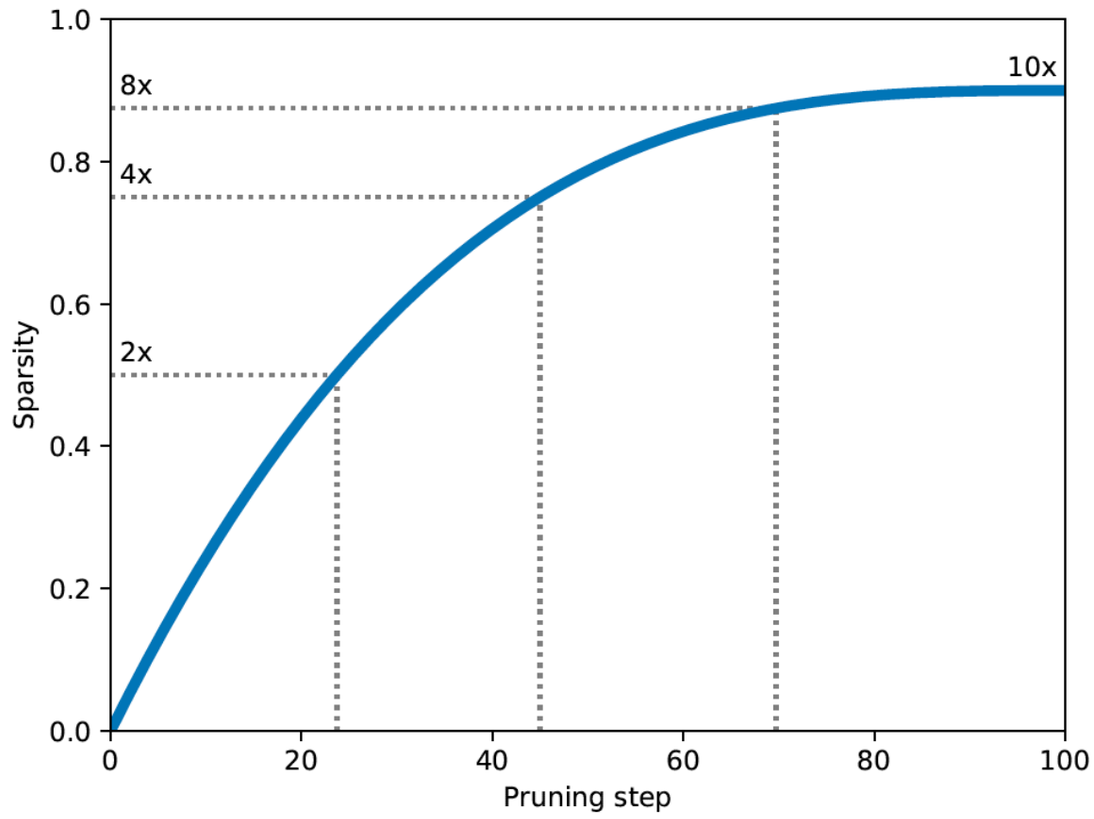
```
=======================================
== Utilization Estimates
=======================================
+--------------------+---------+-------+---------+--------+
|        Name        | BRAM_18K| DSP48E|    FF   |   LUT  |
+--------------------+---------+-------+---------+--------+
...
+--------------------+---------+-------+---------+--------+
|Total               |     1962|   1162|   169979|  222623|
+--------------------+---------+-------+---------+--------+
|Available SLR       |     2160|   2760|   663360|  331680|
+--------------------+---------+-------+---------+--------+
|Utilization SLR (%) |       90|     42|       25|      67|
+--------------------+---------+-------+---------+--------+
|Available           |     4320|   5520|  1326720|  663360|
+--------------------+---------+-------+---------+--------+
|Utilization (%)     |       45|     21|       12|      33|
+--------------------+---------+-------+---------+--------+
```

Part 3: compression

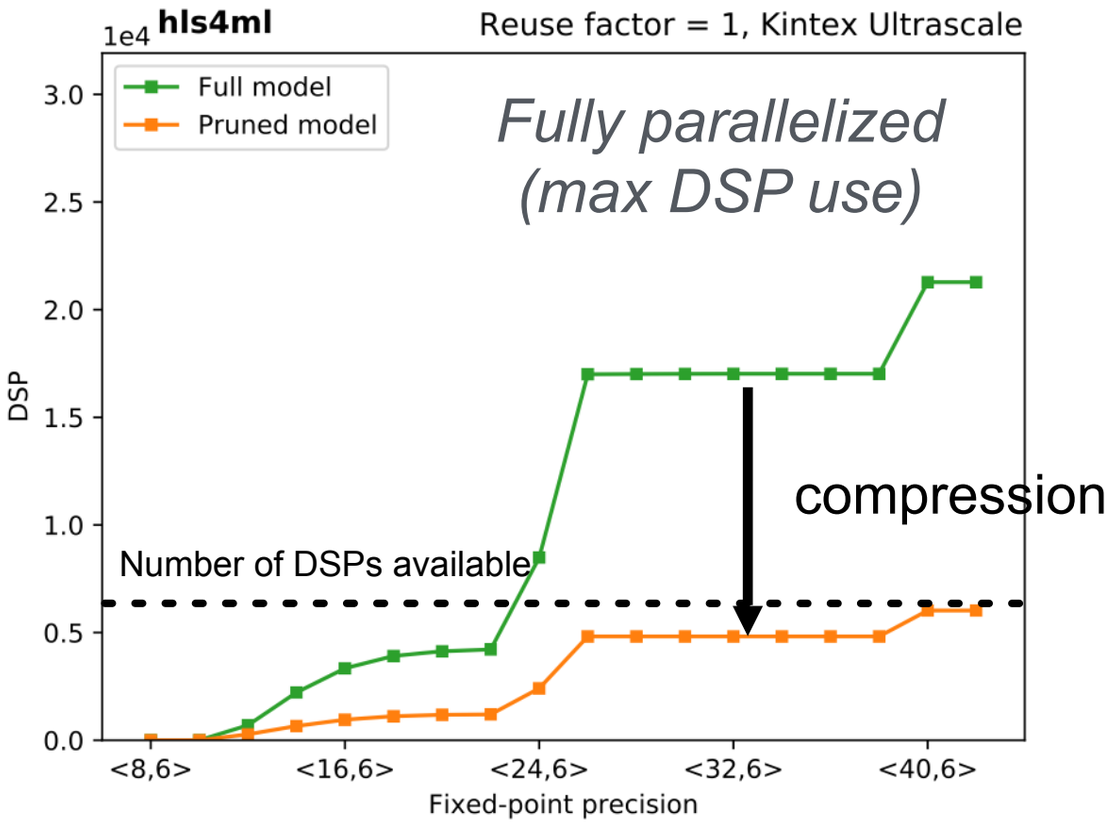
Efficient NN design: compression

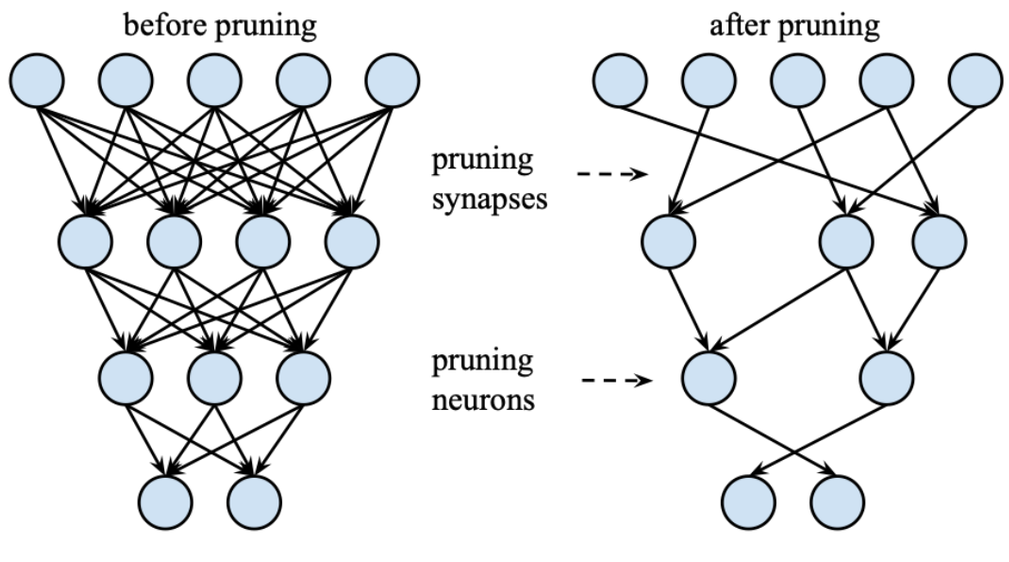
- Neural Network compression is a widespread technique to reduce the size, energy consumption, and overtraining of deep neural networks
- Several approaches in literature [arxiv.1510.00149, arxiv.1712.01312, arxiv.1405.3866, arxiv.1602.07576, doi:10.1145/1150402.1150464]
- Today we will test the tensorflow model sparsity toolkit
  - https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html

Main idea:
iteratively remove low magnitude weights, starting with 0 sparsity, smoothly increasing up to the set target as training proceeds



Efficient NN design: compression



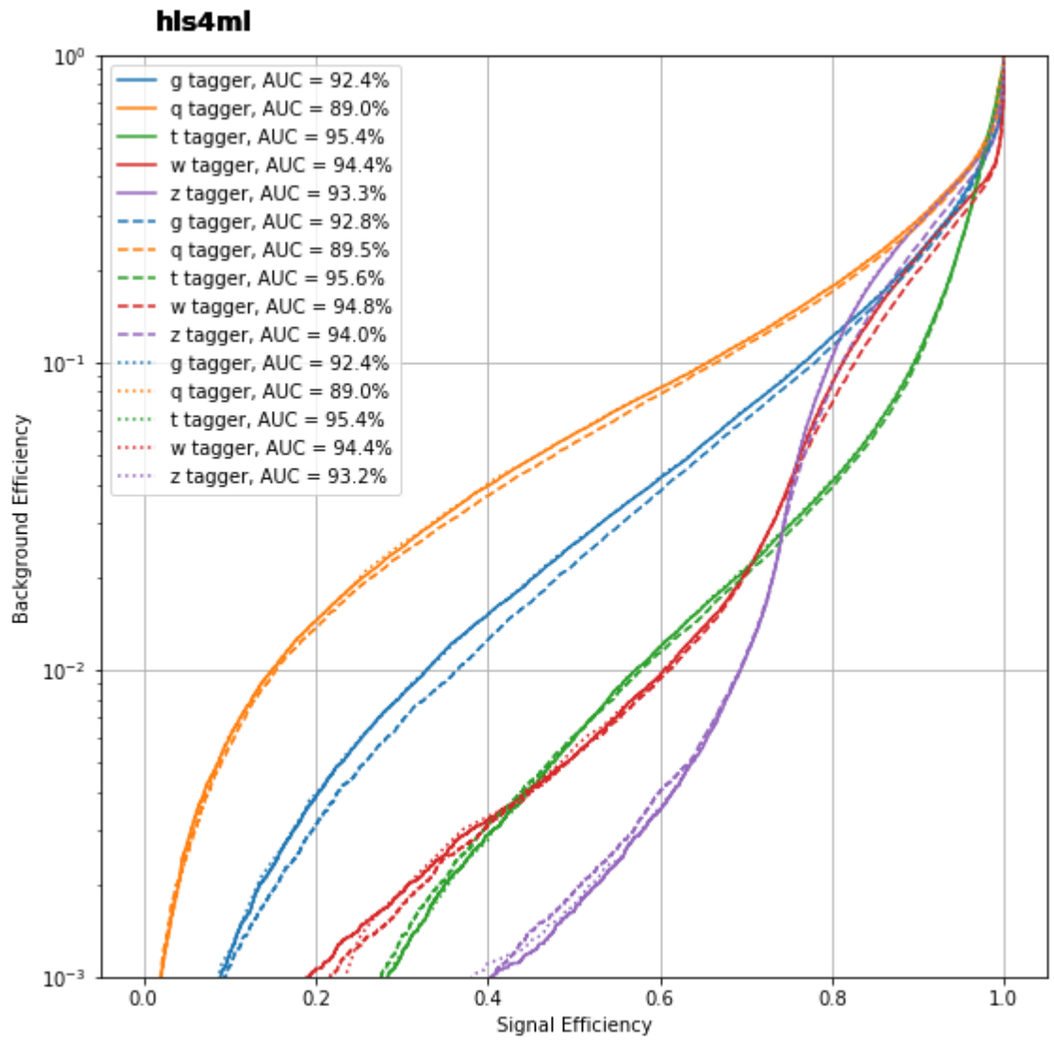


- DSPs (used for multiplication) are often limiting resource
  - maximum use when fully parallelized
  - DSPs have a max size for input (e.g. 27x18 bits), so number of DSPs per multiplication changes with precision

Part 4:
quantization-aware training

Efficient NN design: quantization

- hls4ml allows you to use different data types everywhere, we saw how to tune that in part 2
- In this part we will also try quantization-aware training with QKeras [arxiv.2006.10159]
- With quantization-aware we can even go down to just 1 or 2 bits [Mach. Learn.: Sci. Technol. 2, 015001 (2020)]



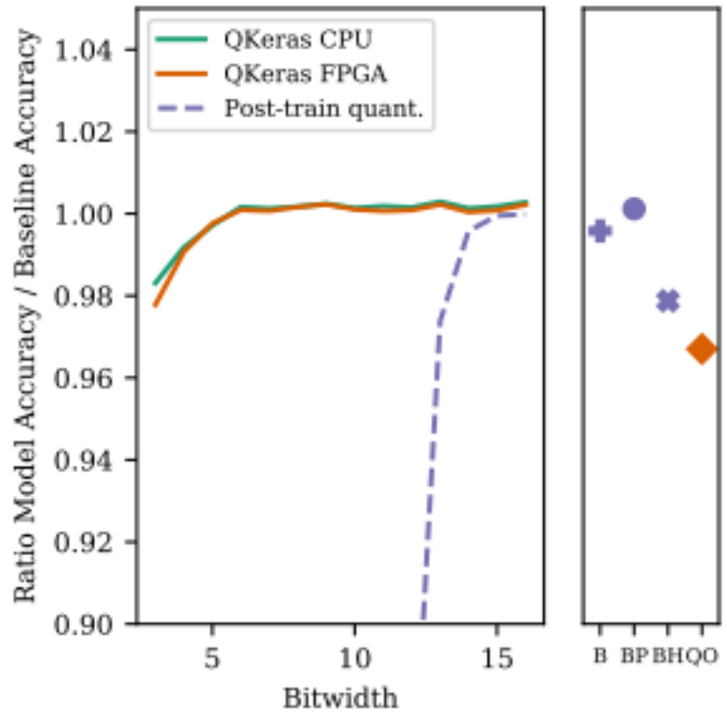
Efficient NN design with QKeras

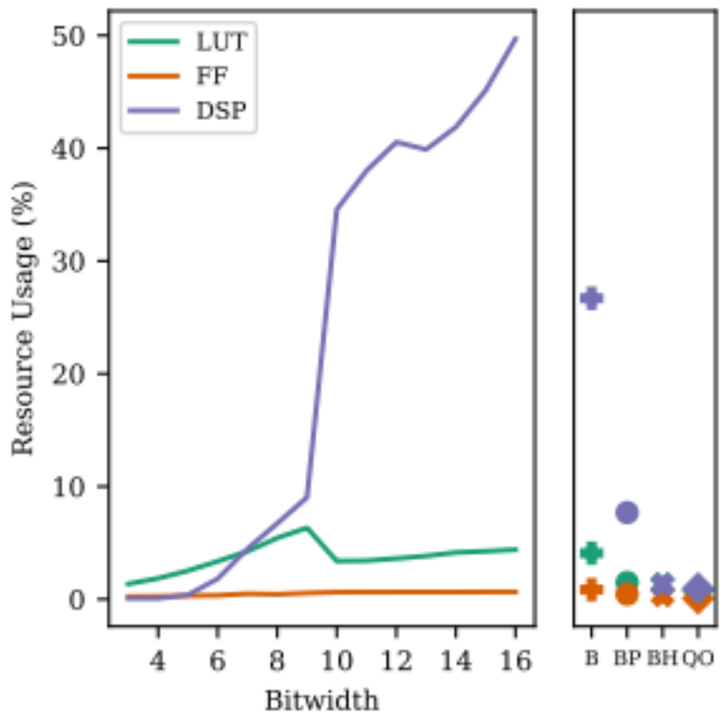
- QKeras is a library developed and maintained by Google to train models with quantization in the training

- Easy to use, drop-in replacements for Keras layers
  - **e.g.** `Dense` → `QDense` **OR** `Conv2D` → `QConv2D`
  - **use** `quantizers` **to specify how many bits to use where with same kind of granularity as hls4ml**

- Can achieve good performance with very few bits

- We've recently added support for QKeras-trained models to hls4ml
  - the number of bits used in training is also used in inference
  - the intermediate model is adjusted to capture all optimizations possible with QKeras



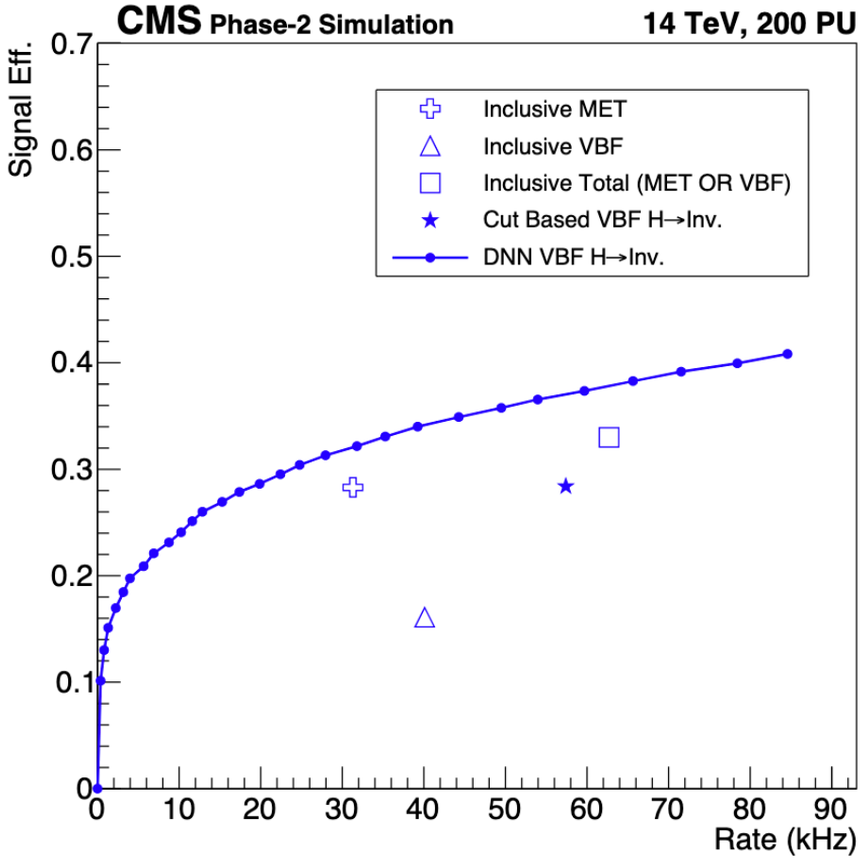


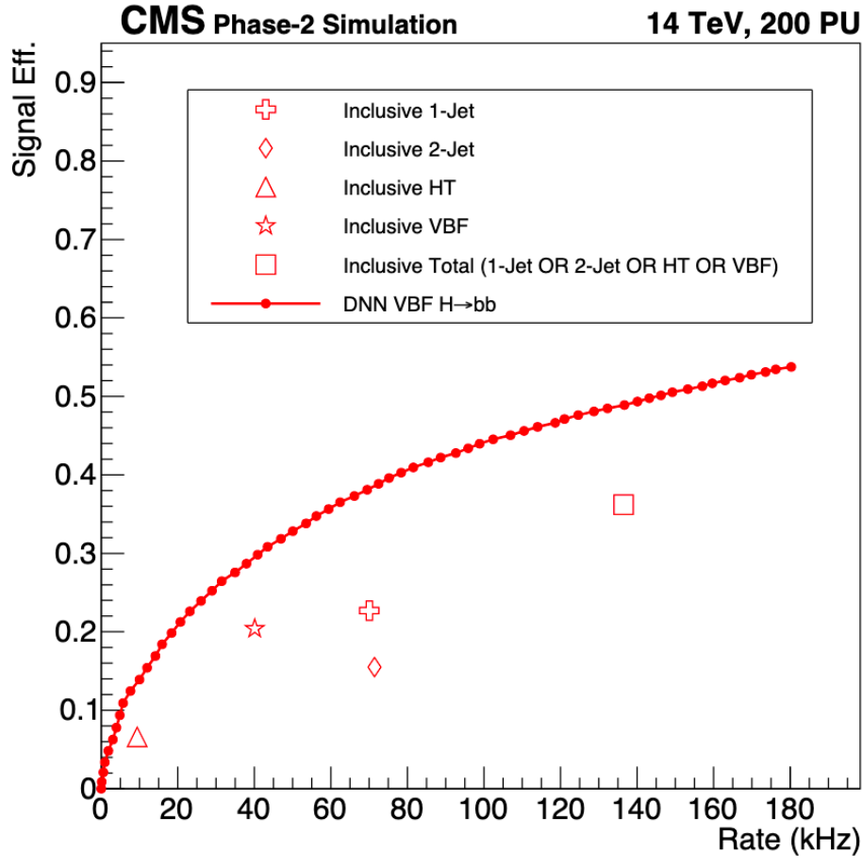
Bonus: L1 trigger applications

hls4ml for triggering @ 40 MHz

- hls4ml enabled large development of new trigger algorithms with large gain for physics
  - replace standard cut-based algorithms

CMS Phase-2 L1 trigger upgrade TDR





NN VBF H→bb

	Usage	Percentage
Latency	24 clk @ 200MHz	
II	5	
DSP48E	484	8%
FF	32634	2%
LUT	62358	9%

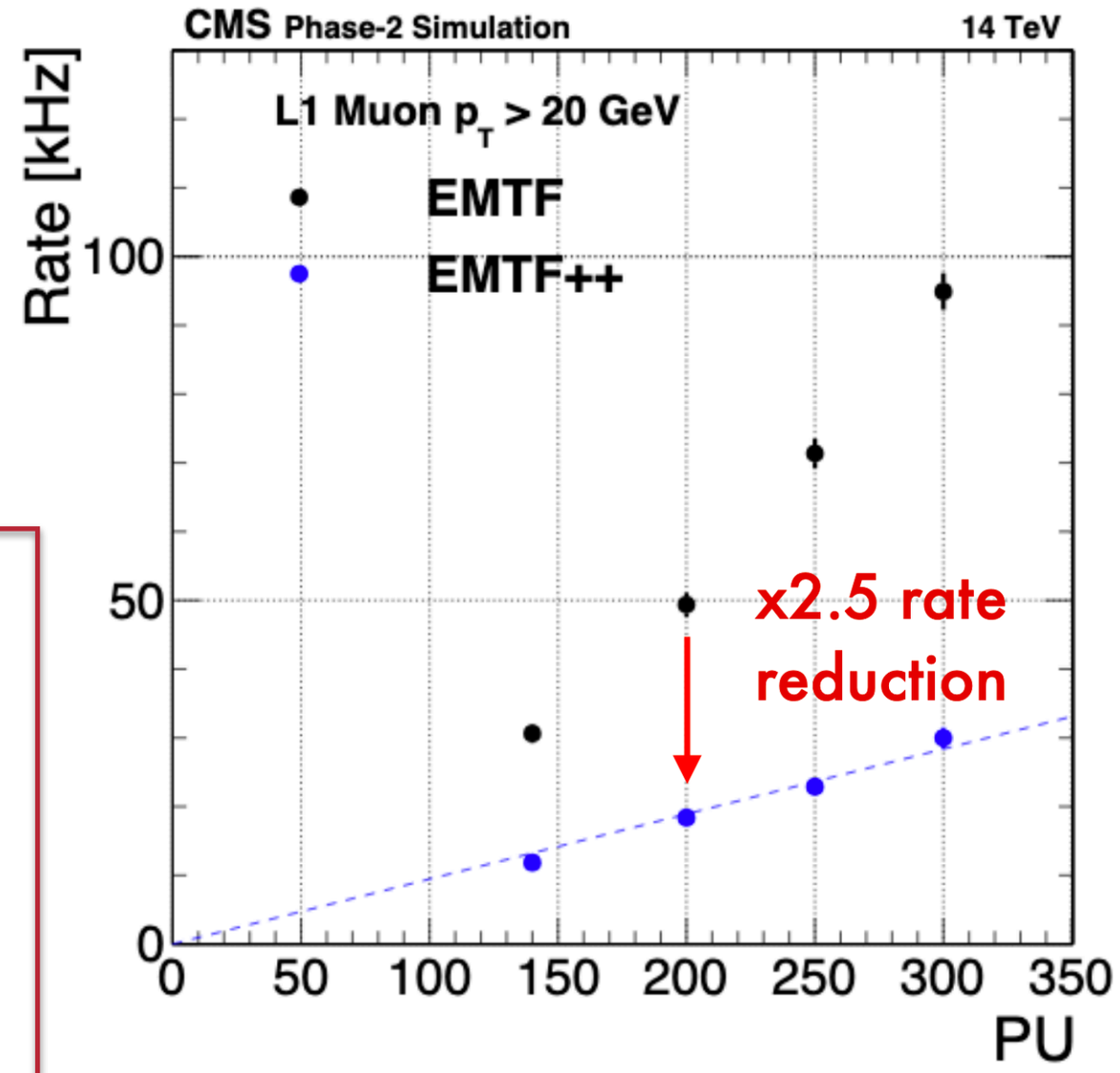
hls4ml for triggering @ 40 MHz

- hls4ml enabled large development of new trigger algorithms with large gain for physics

  - replace standard cut-based algorithms

  - improve physics objects reconstruction (muons, taus, jets)

36 INPUT FEATURES:
ϕ,θ of track segments in muon stations
track segment quality
track segment curvature
⬇
3 HIDDEN LAYERS (30x25x20)
⬇
1 OUTPUT: muon p_T



CMS Phase-2 L1 trigger upgrade TDR

hls4ml for triggering @ 40 MHz

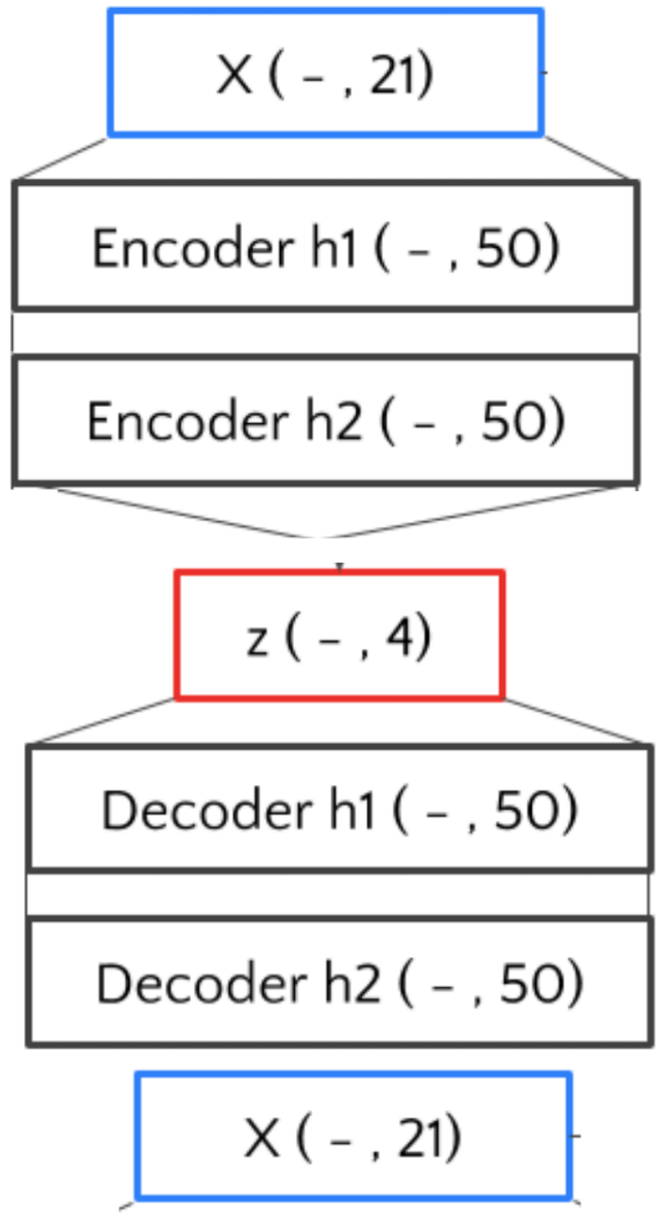
- hls4ml enabled large development of new trigger algorithms with large gain for physics

  - replace standard cut-based algorithms

  - improve physics objects reconstruction (muons, taus, jets)

  - develop new strategies like anomaly detection with autoencoders for signal-agnostic triggering

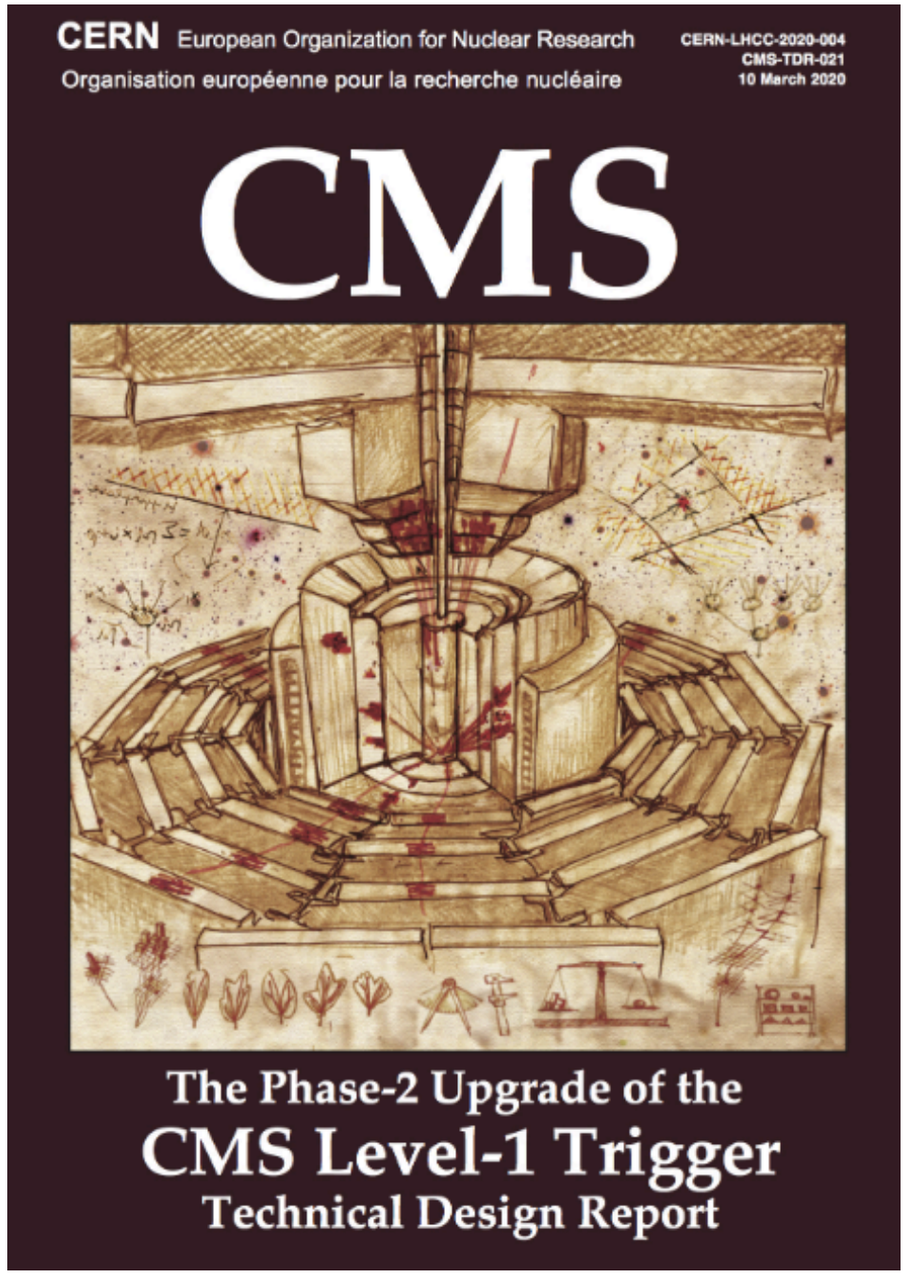
21 inputs: $p_T/\eta/\Phi$ of 4 e/γ, 4 μ, 10 jets, and MET



K. Govorkova @ Fast Machine Learning workshop 20

hls4ml for triggering @ 40 MHz

- hls4ml enabled large development of new trigger algorithms with large gain for physics
  - replace standard cut-based algorithms
  - improve physics objects reconstruction (muons, taus, jets)
  - develop new strategies like anomaly detection with autoencoders for signal-agnostic triggering
- Expected to see further developments thanks to the latest QKeras and AutoQ support
  - make the model small and accurate!



CMS Phase-2 L1 trigger upgrade TDR

Bonus:
further developments and features

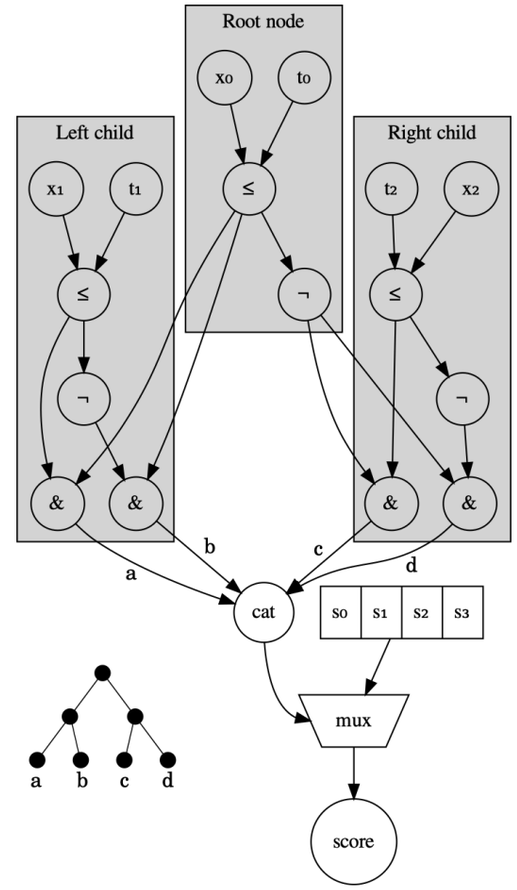
hls4ml bonuses

- We have discussed the original motivations behind hls4ml: extreme low latency, high throughput domain as for LHC first-stage triggers
- Since then, we have been expanding!
  - longer latency domains, larger models, resource constrained
  - different FPGA vendors (Xilinx, Intel, Mentor)
  - new applications, new architectures
- While maintaining core characteristics:
  - HLS-based fully on-chip implementation
  - extremely configurable: precision, resource vs latency/throughput tradeoff
  - research project, application- and user-driven
  - accessible, easy to use

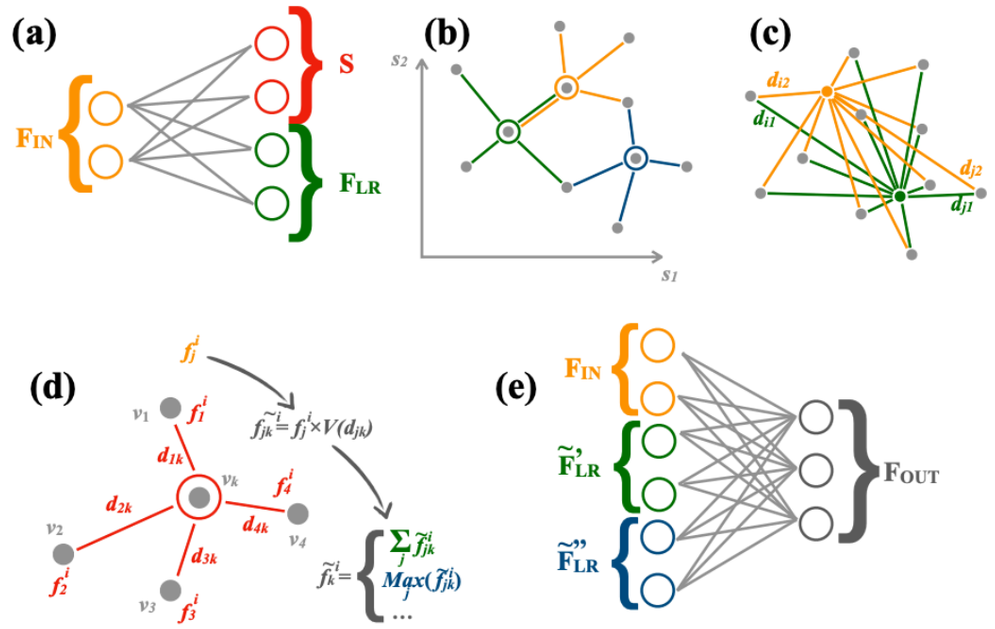
hls4ml bonuses

hls4ml community is very active!

- **Boosted Decision Trees** [JINST 15 P05026 (2020)]



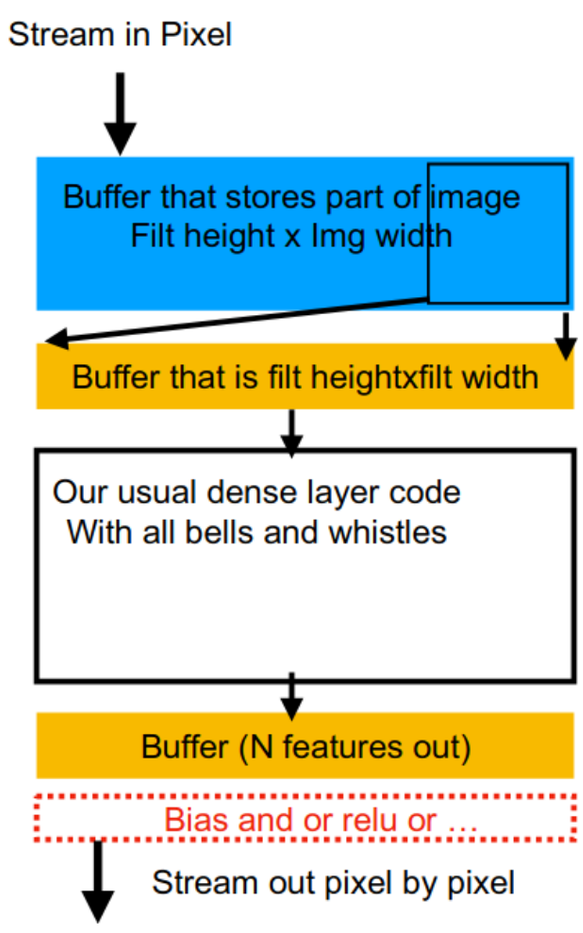
- **Custom graph neural networks:**
  - GarNet/GravNet for calorimeter reconstruction [arXiv: 2008.03601]
  - Interaction networks for tracking [arxiv.2012.01563]

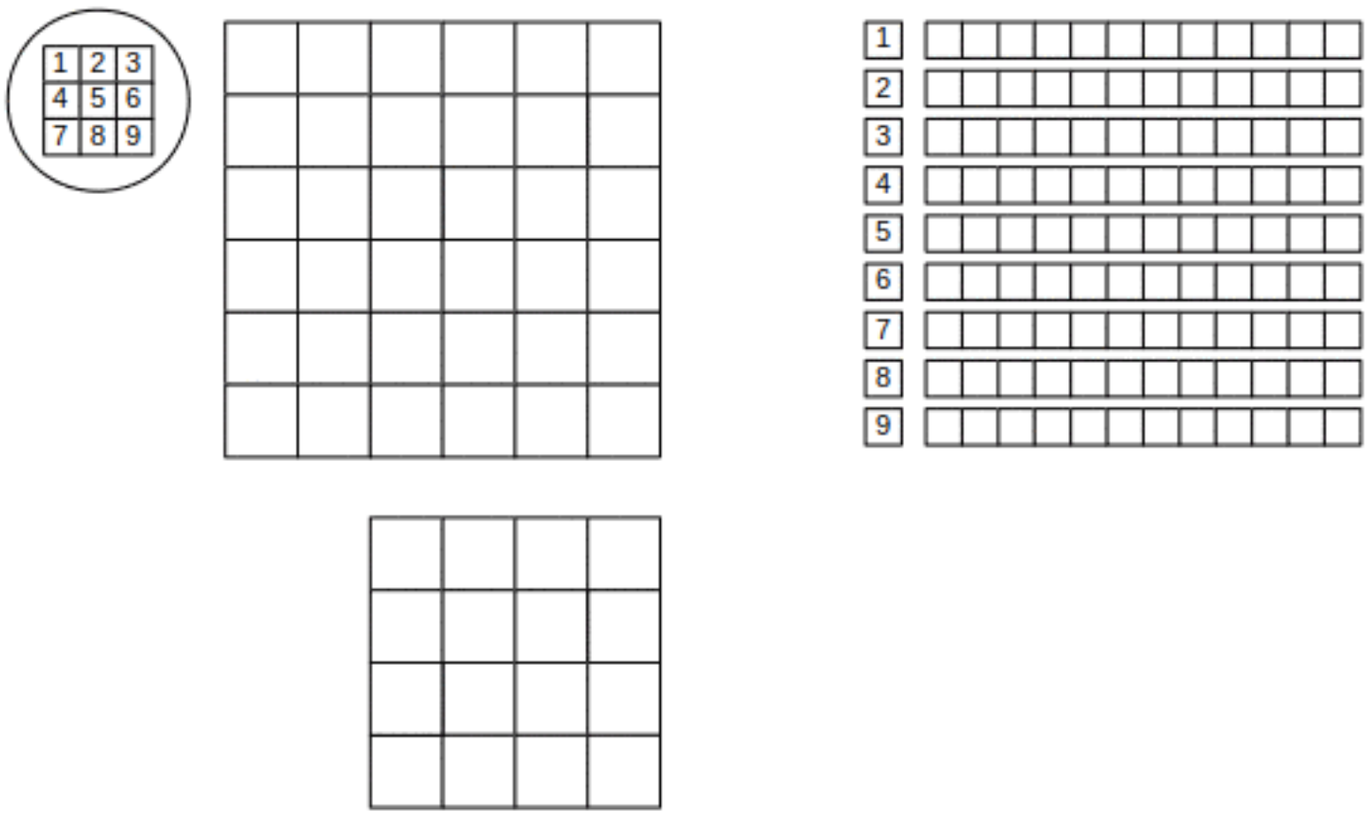


- **Large convolutional neural networks** [arxiv.2101.05108]

Fast convolutional neural networks

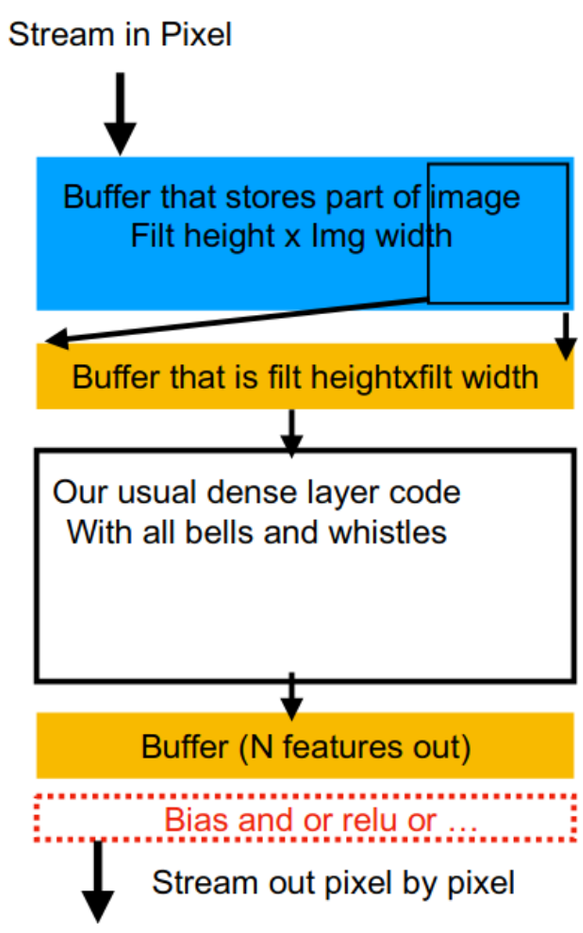
- New implementation based on streaming `hls::stream<T>`
  - collect data from input pixels until we can compute one output (FIFOs)
  - compute the value of output pixel with a single call to matrix-vector multiplication
  - can reuse existing matrix-vector multiplication used for fully connected layers

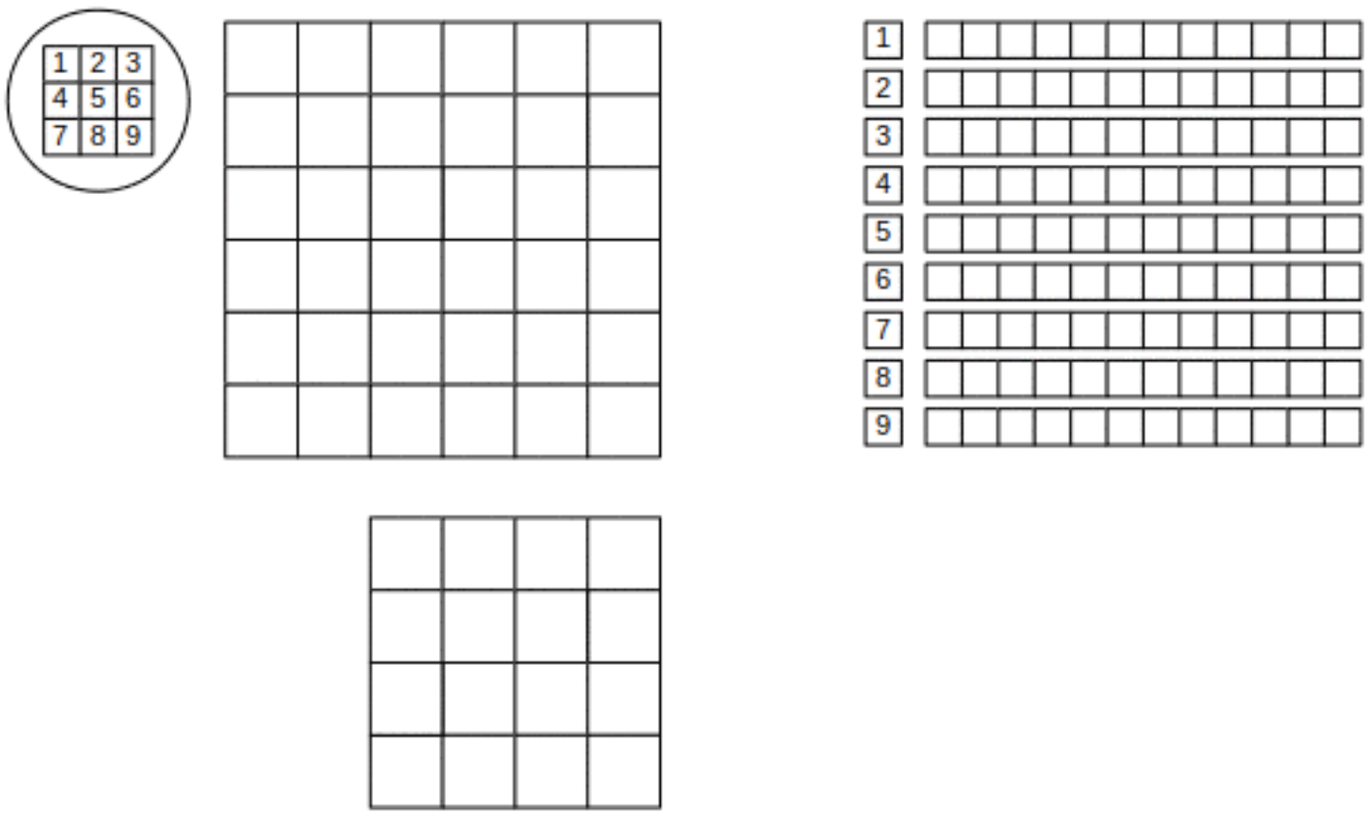




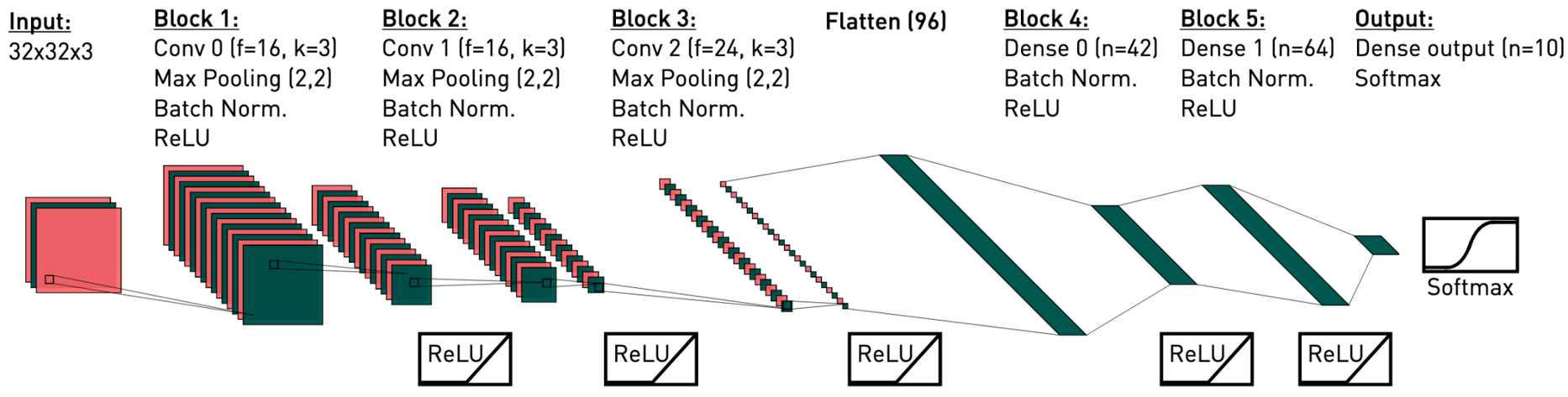
Fast convolutional neural networks

- New implementation based on streaming `hls::stream<T>`
  - collect data from input pixels until we can compute one output (FIFOs)
  - compute the value of output pixel with a single call to matrix-vector multiplication
  - can reuse existing matrix-vector multiplication used for fully connected layers

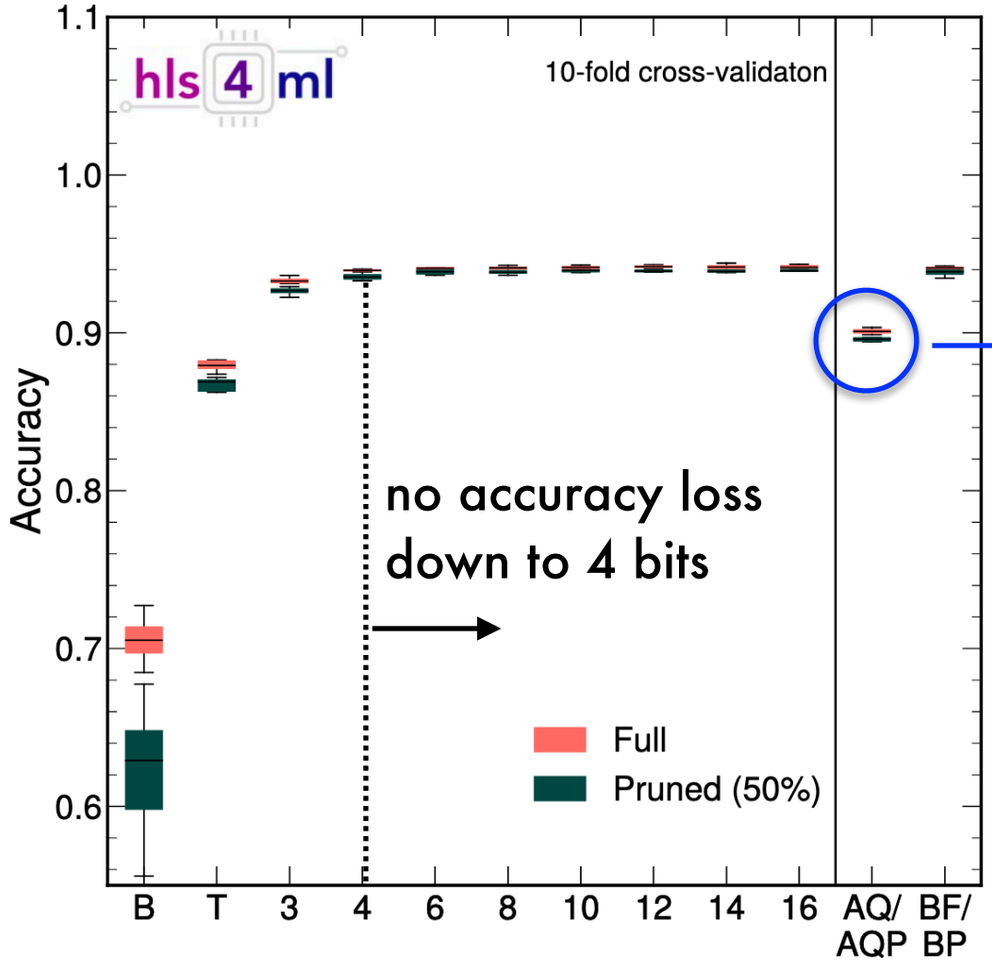




Fast convolutional neural networks



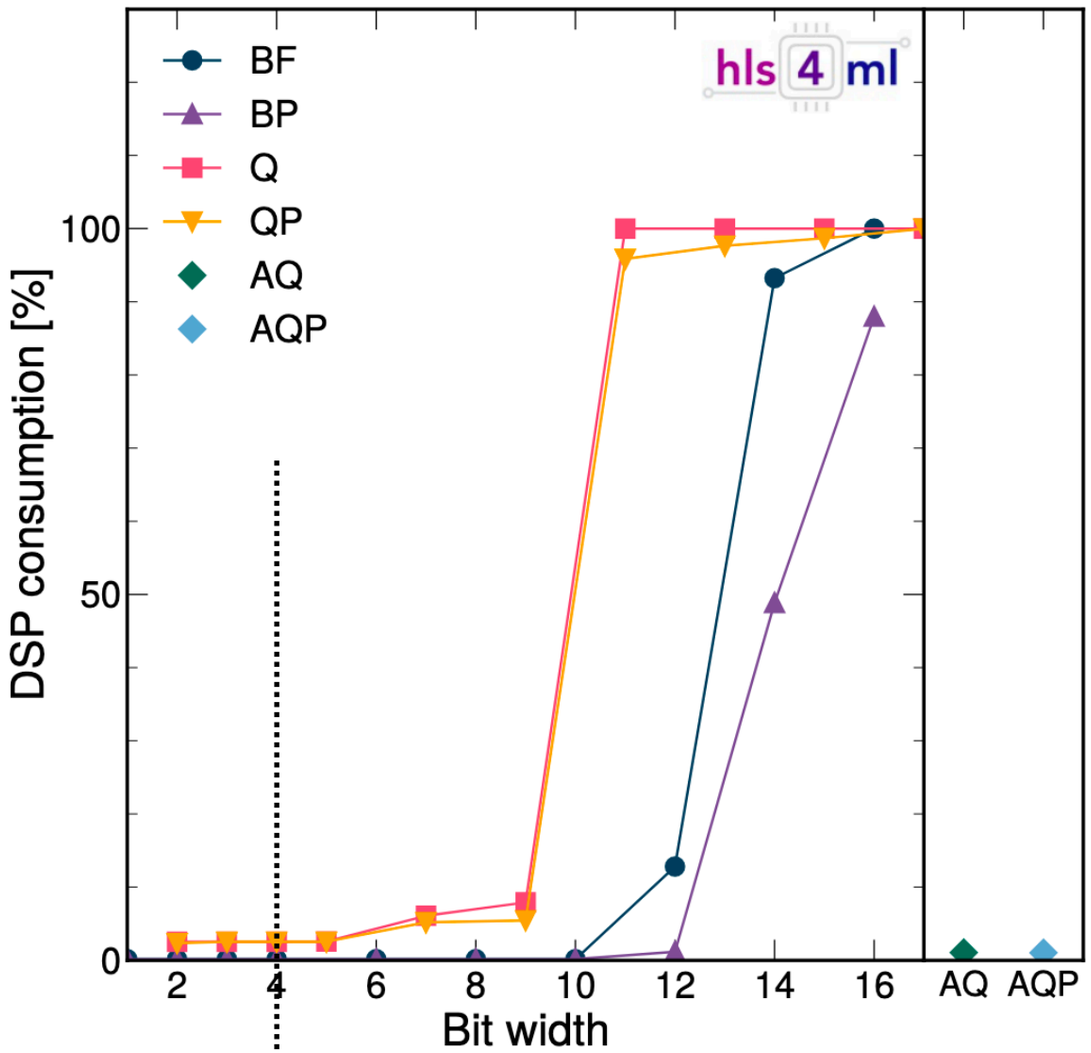
Evaluate performance on street-view house numbers dataset (32x32x3)



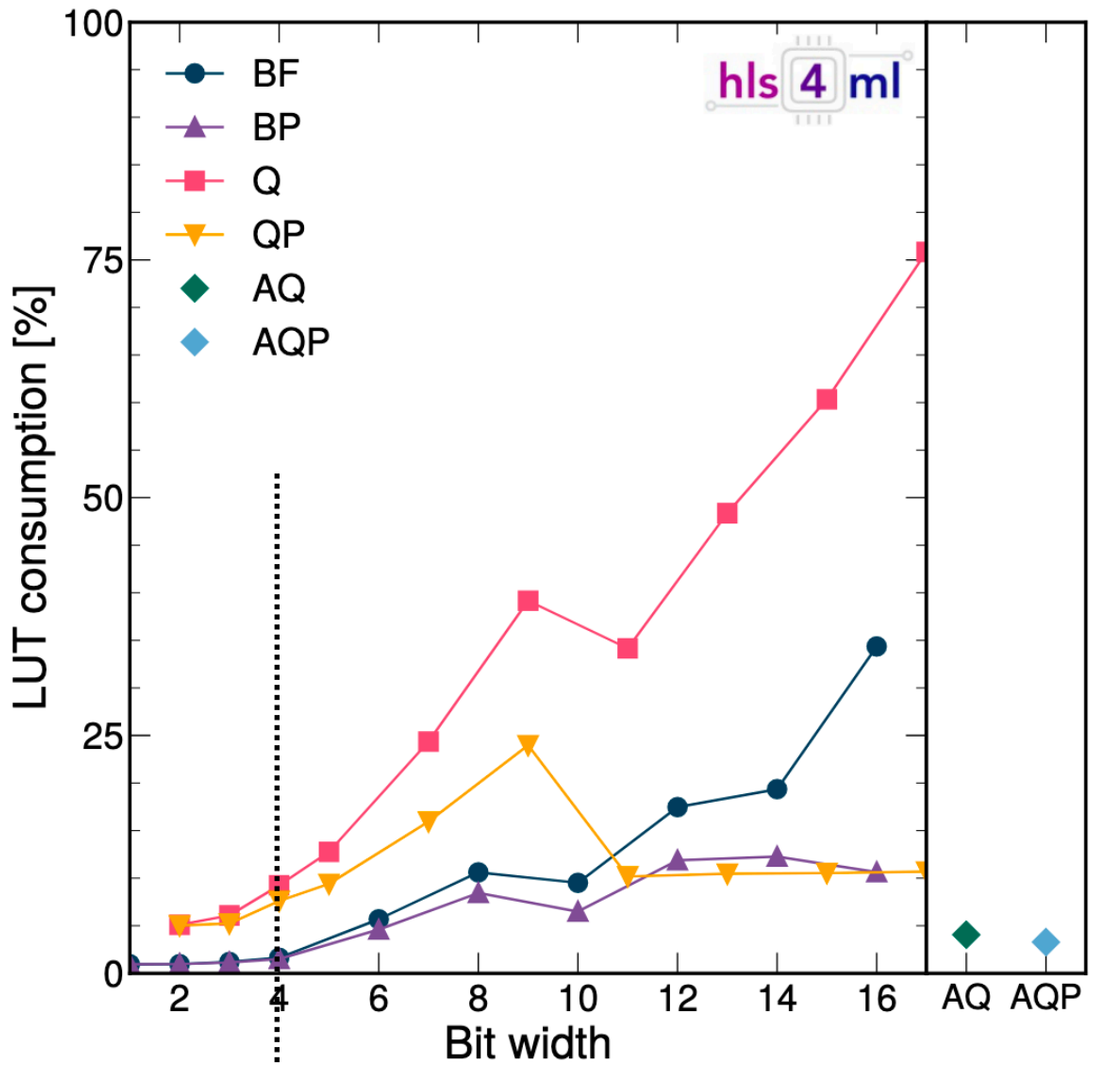
heterogeneously quantized model through bayesian optimization

Fast convolutional neural networks

Max parallelization, i.e. reuse factor = 1



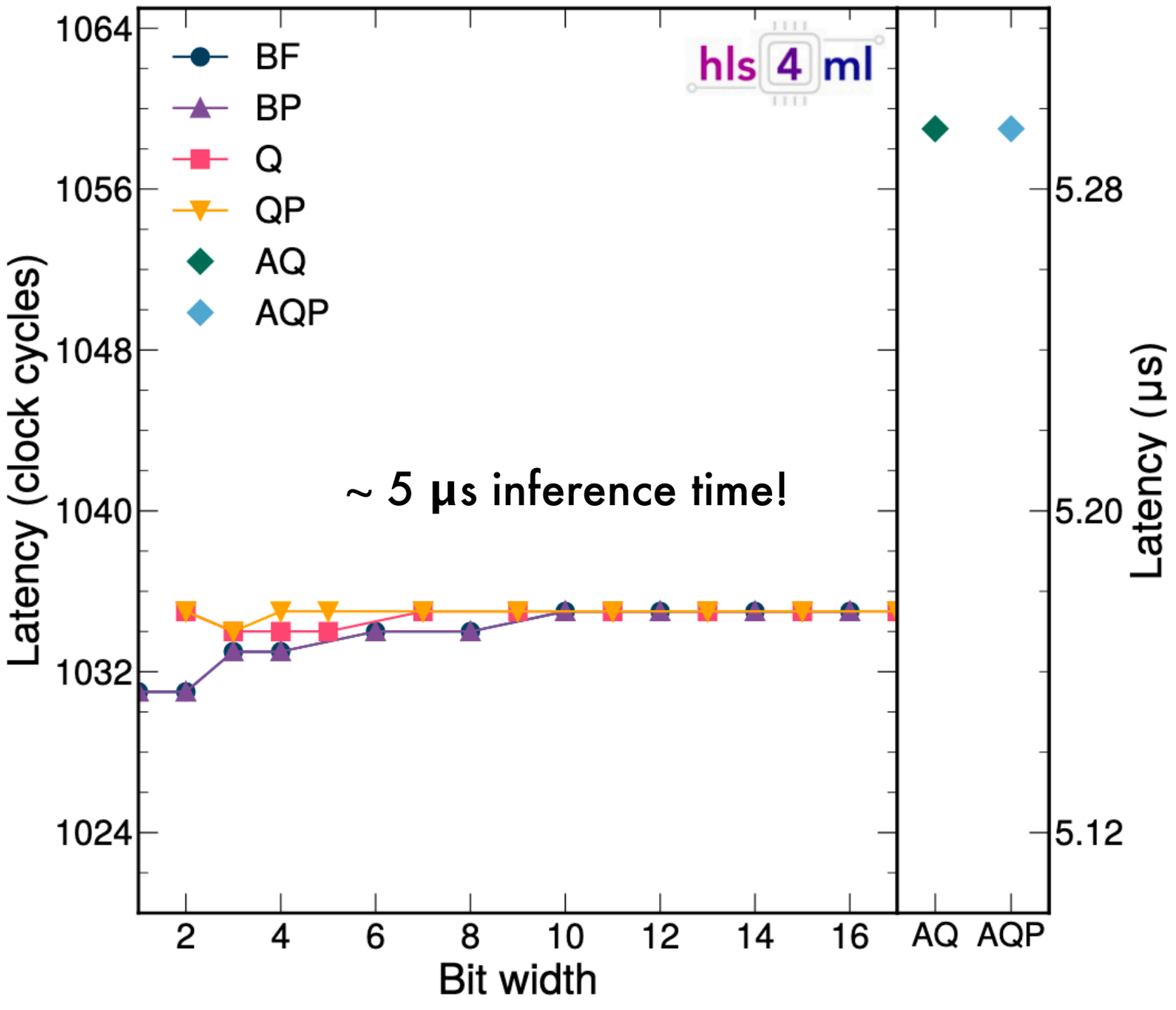
no accuracy loss
down to 4 bits for
Q/QP models

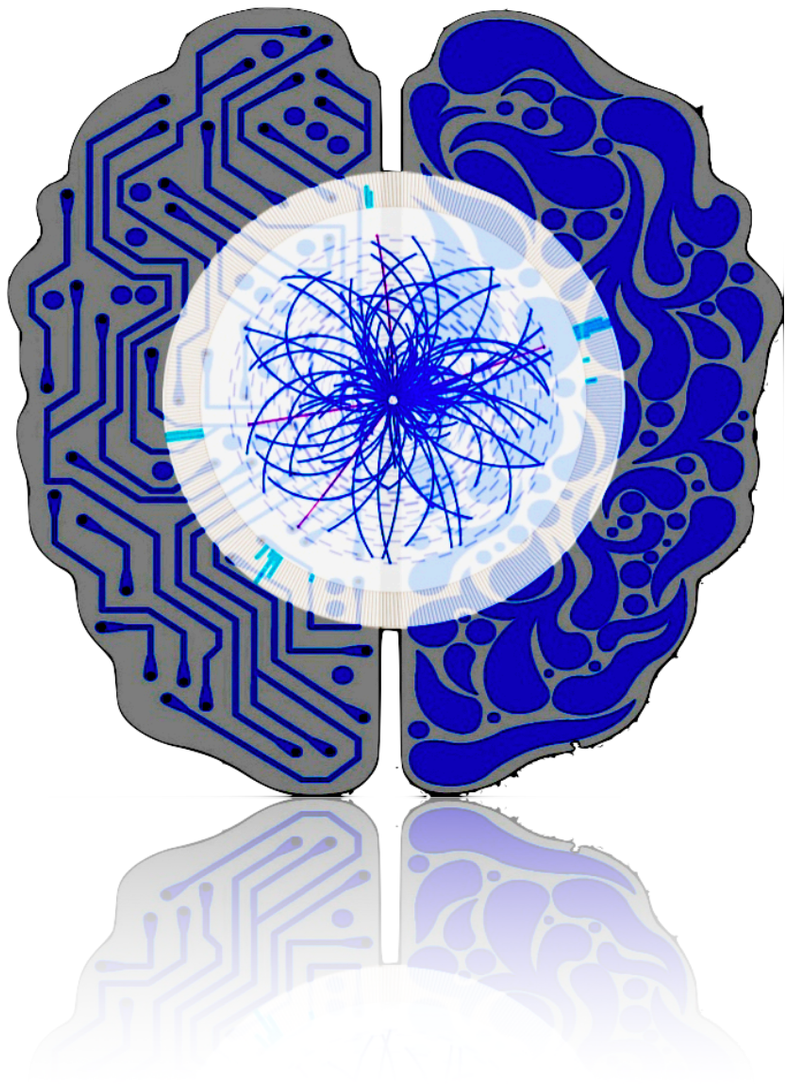


no accuracy loss
down to 4 bits for
Q/QP models

Fast convolutional neural networks

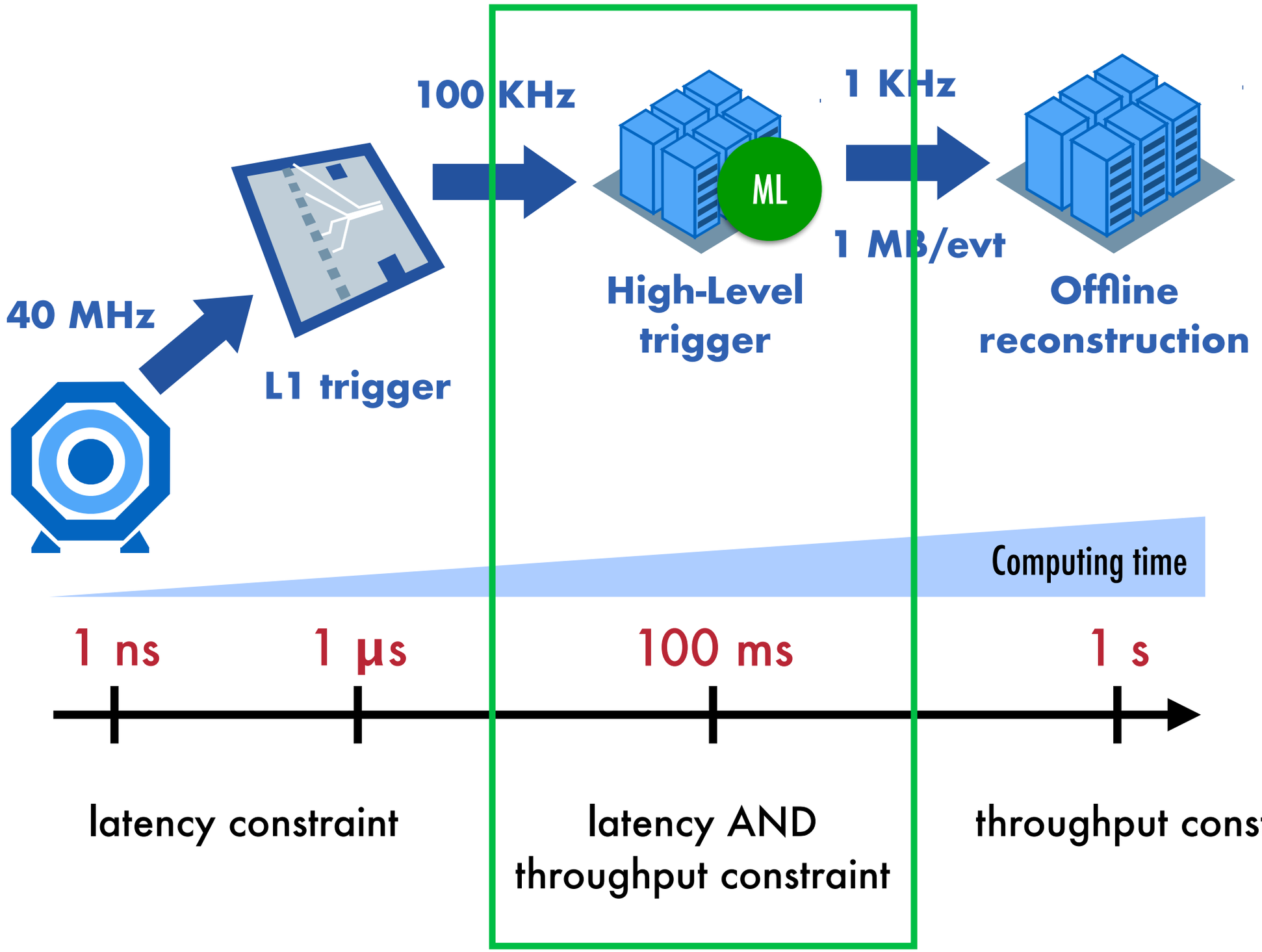
Max parallelization, i.e. reuse factor = 1





Bonus:
Fast machine learning beyond L1 trigger

The need for **fast ML**

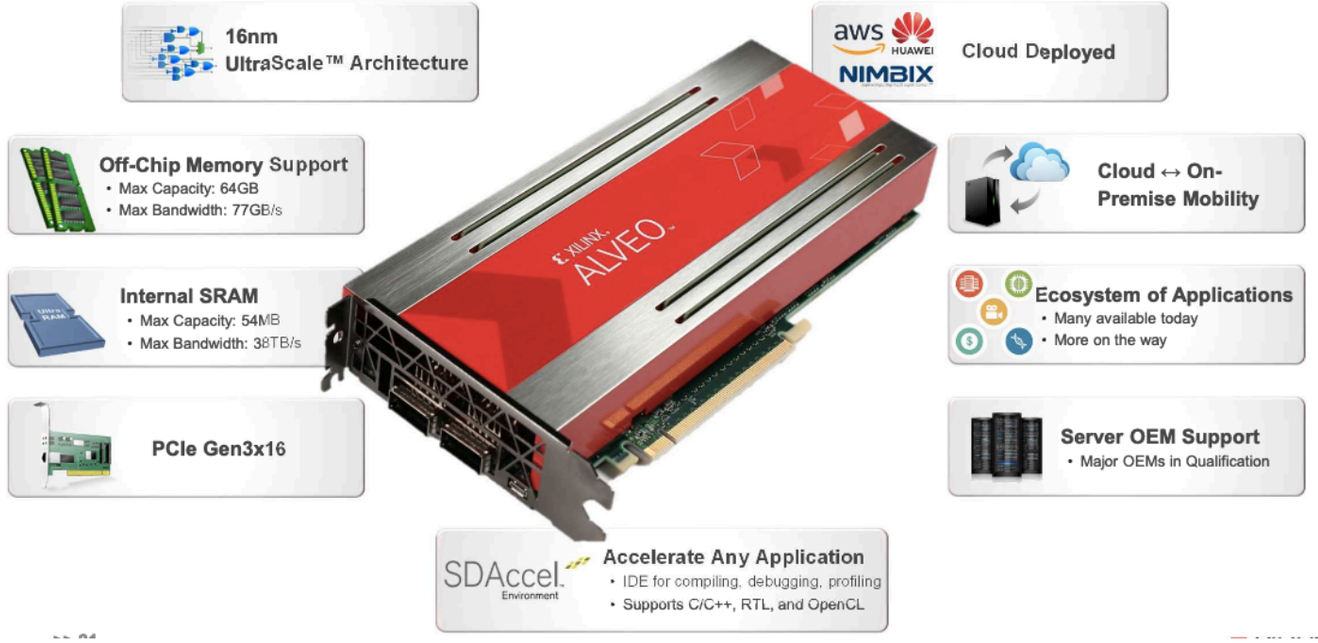


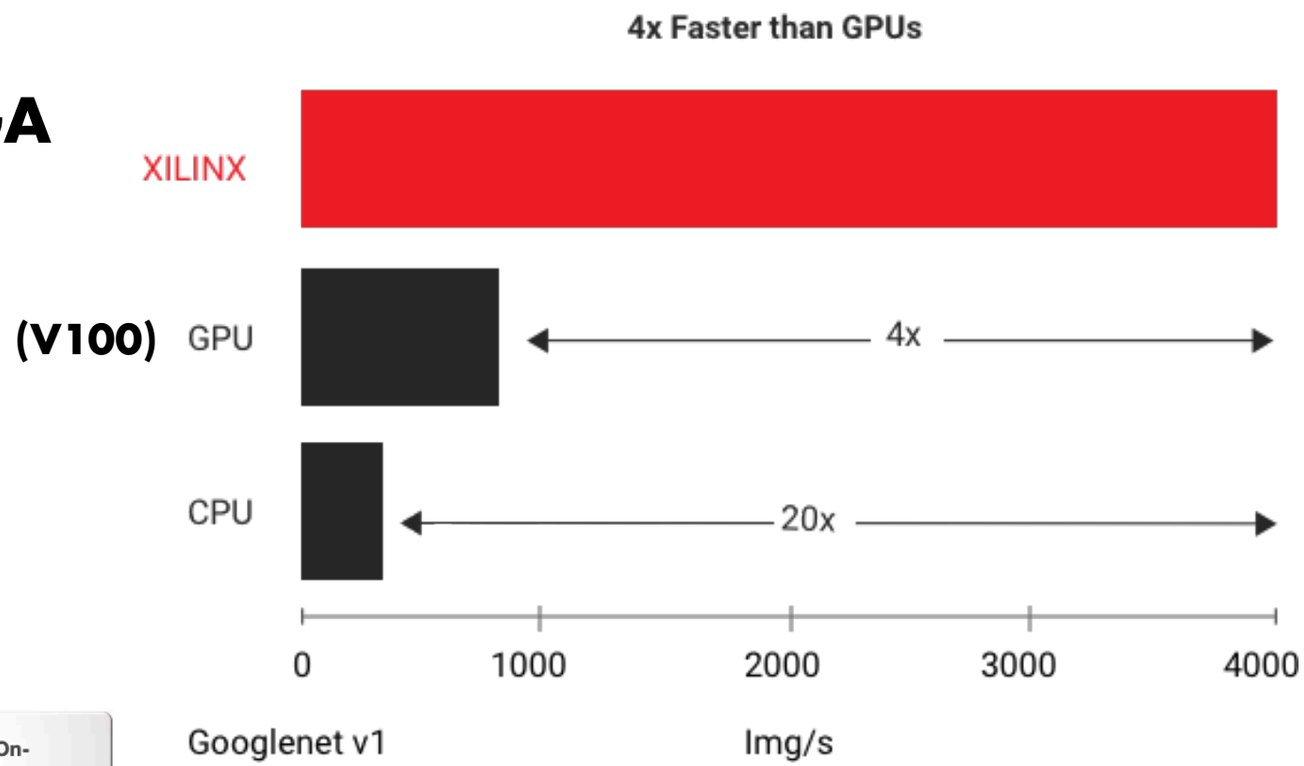
Longer time → bigger models → coprocessors

Heterogenous computing

These platforms based on CPU+FPGA co-processor system: **offload a CPU from the computational heavy parts to a FPGA "accelerator"**

Common setup for FPGA connects to CPU through PCI-express





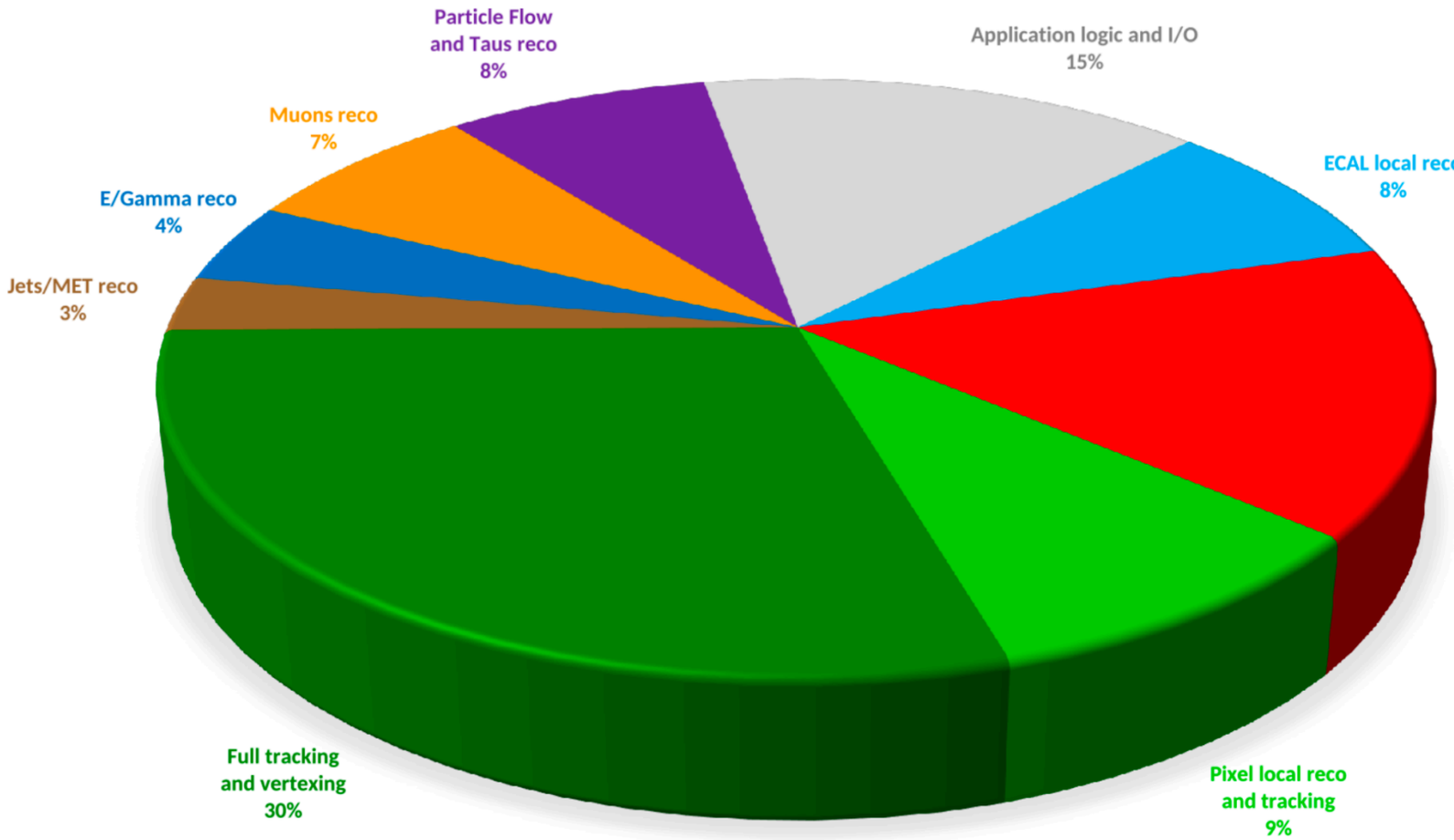
Increased computational speed of 10x-100x
Reduced system size of 10x
Reduced power consumption of 10x-100x

https://www.xilinx.com/support/documentation/white_papers/wp504-accel-dnns.pdf
https://www.xilinx.com/publications/events/machine-learning-live/colorado/xDNN_ML_Suite.pdf
https://www.xilinx.com/applications/megatrends/machine-learning.html

Online reconstruction

Example: CMS online reconstruction

1000 servers w/ 32 cores processing
100K events per second
(750K @ HL-LHC)



ECAL reconstruction 8%

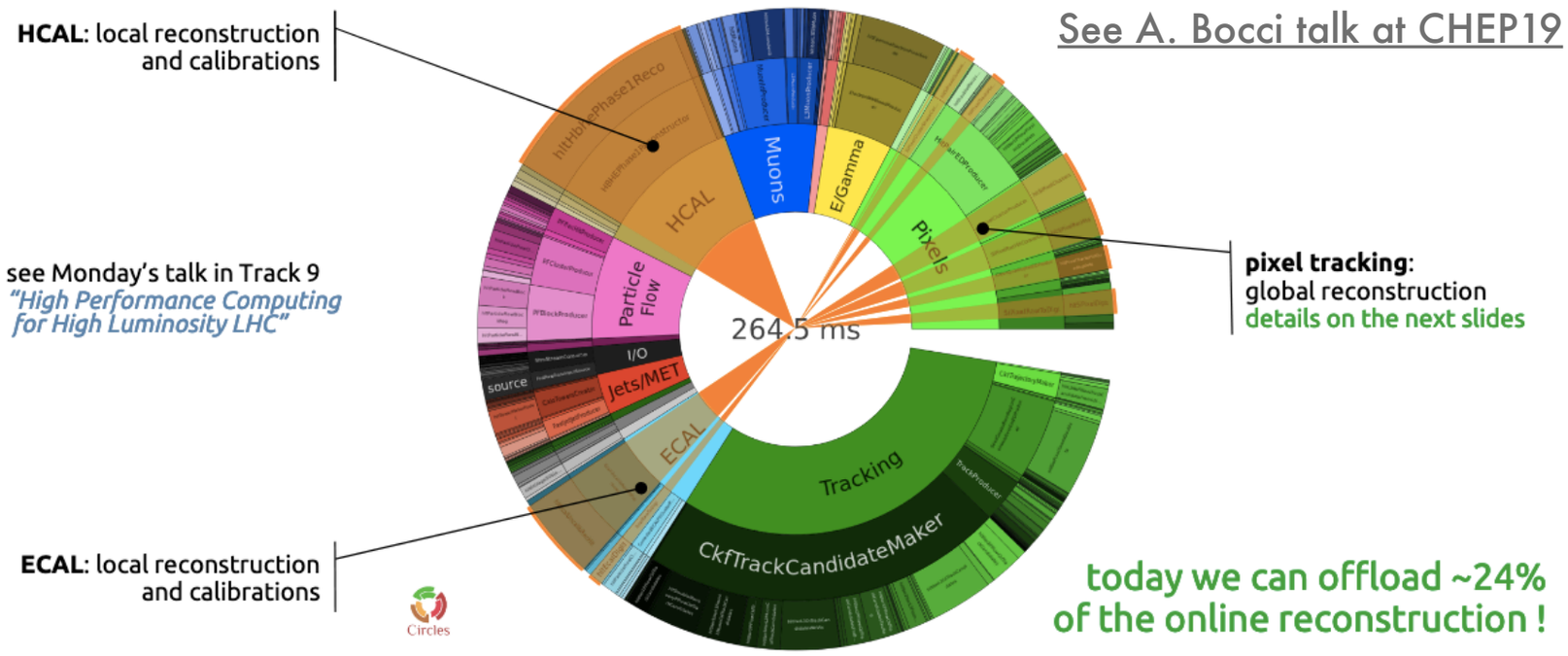
HCAL reconstruction 16%

Full tracking and vertexing 30%

Speeding up tracking and calorimeter reconstruction crucial to increase throughput

Online reconstruction

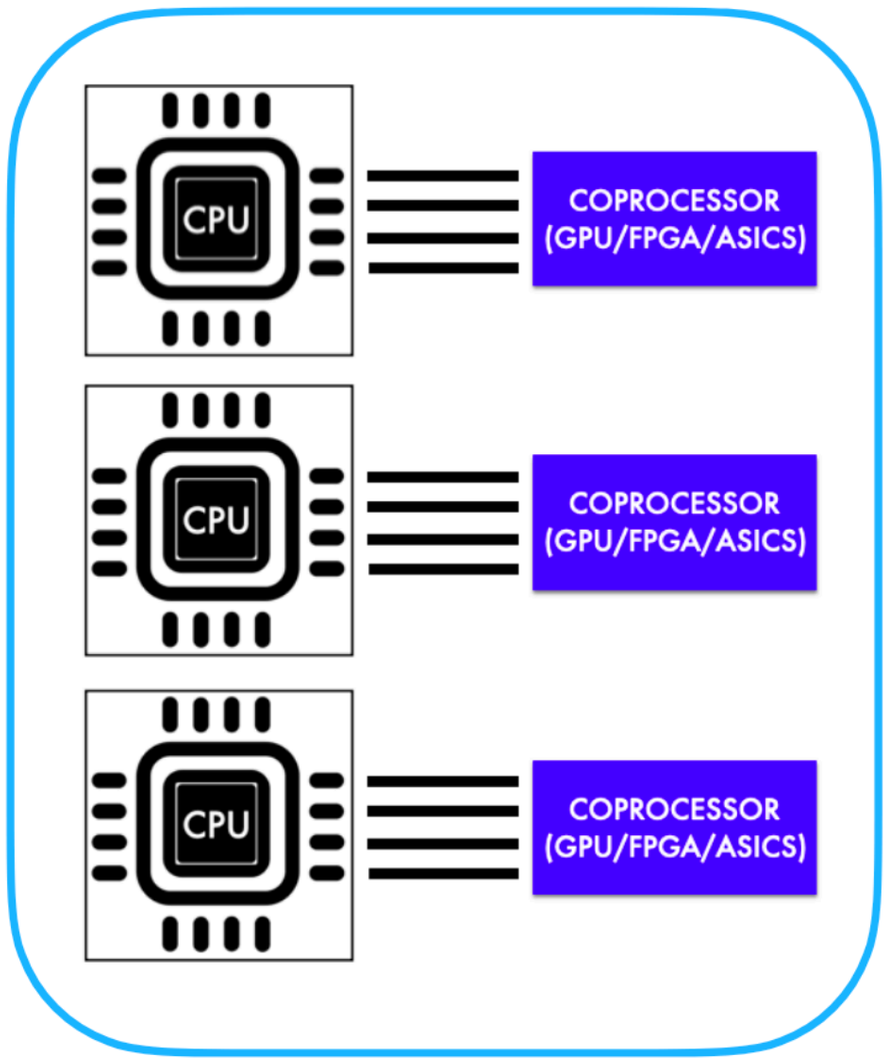
- Large effort in the past years to rewrite parts of the reconstruction in CUDA for Nvidia GPUs
  - example for CMS: can offload 24% of the online reconstruction achieving up to x10 higher throughput



- Parallel effort to replace parts of the reconstruction with ML
  - minimize need to learn new processor-specific code → decrease effort, increase maintainability
  - must exploit co-processors to achieve highest throughput

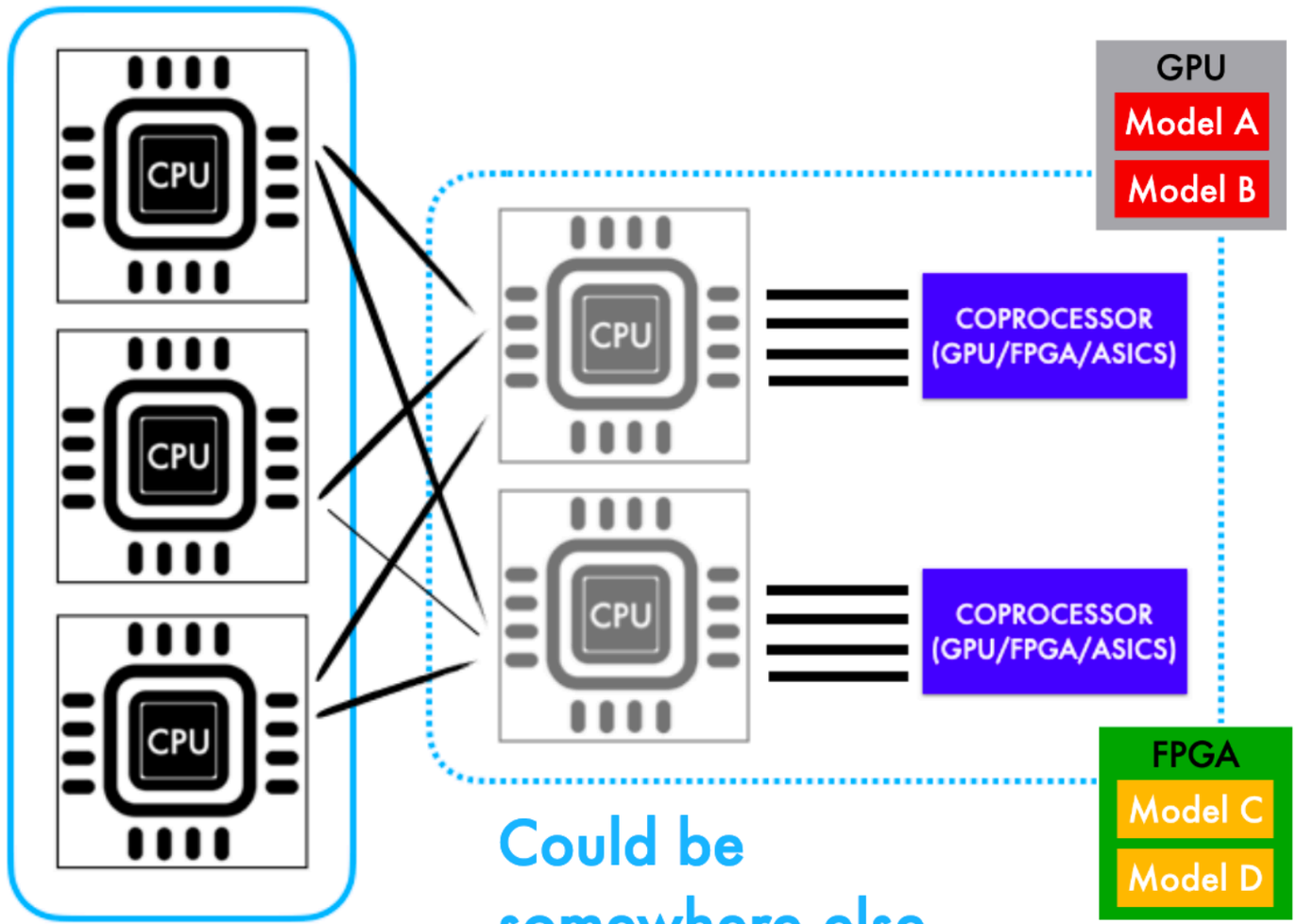
Heterogenous computing @ LHC

Option 1: direct



Data center/ experimental site

Option 2: as a service

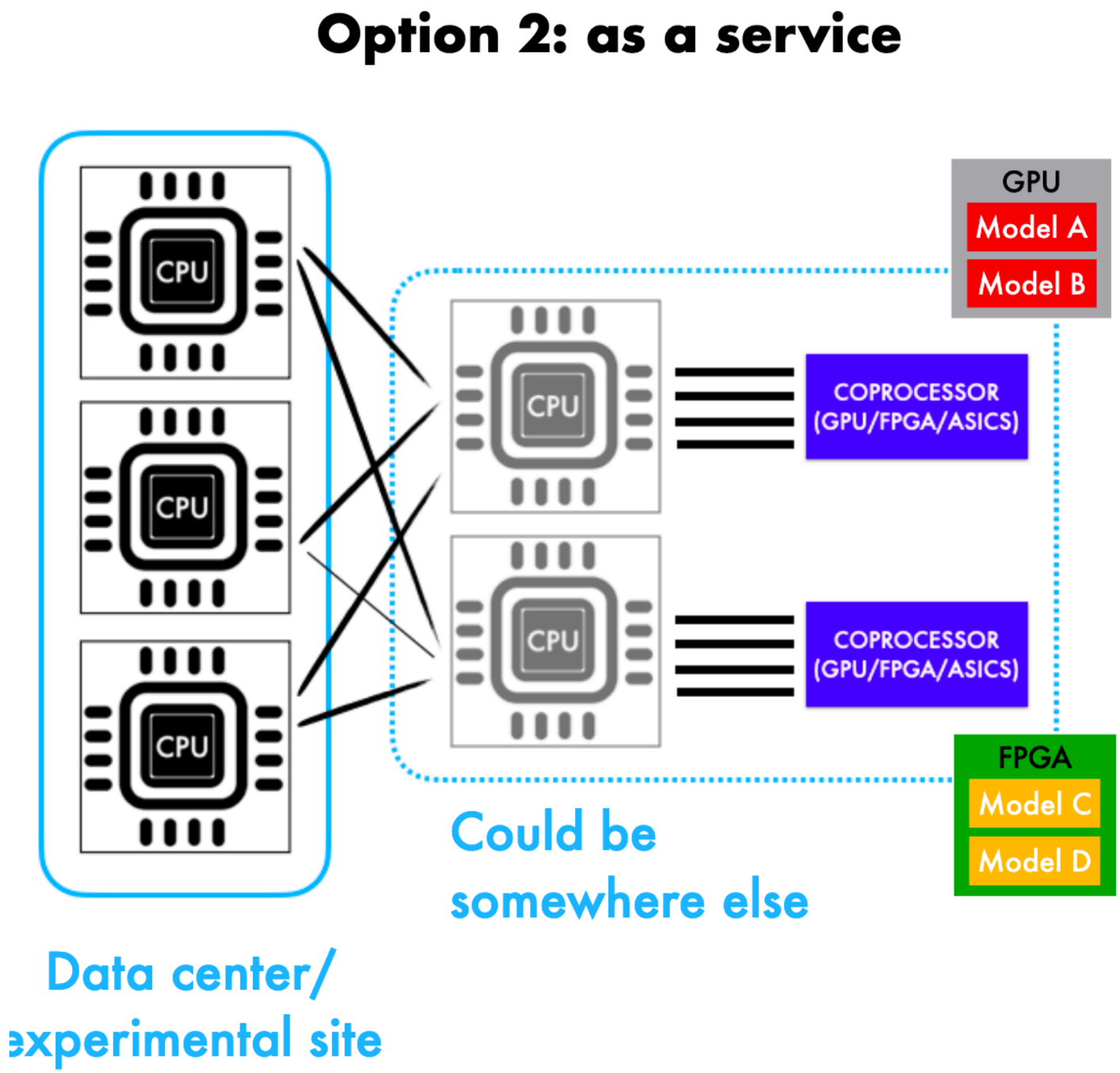


Data center/ experimental site

Could be somewhere else

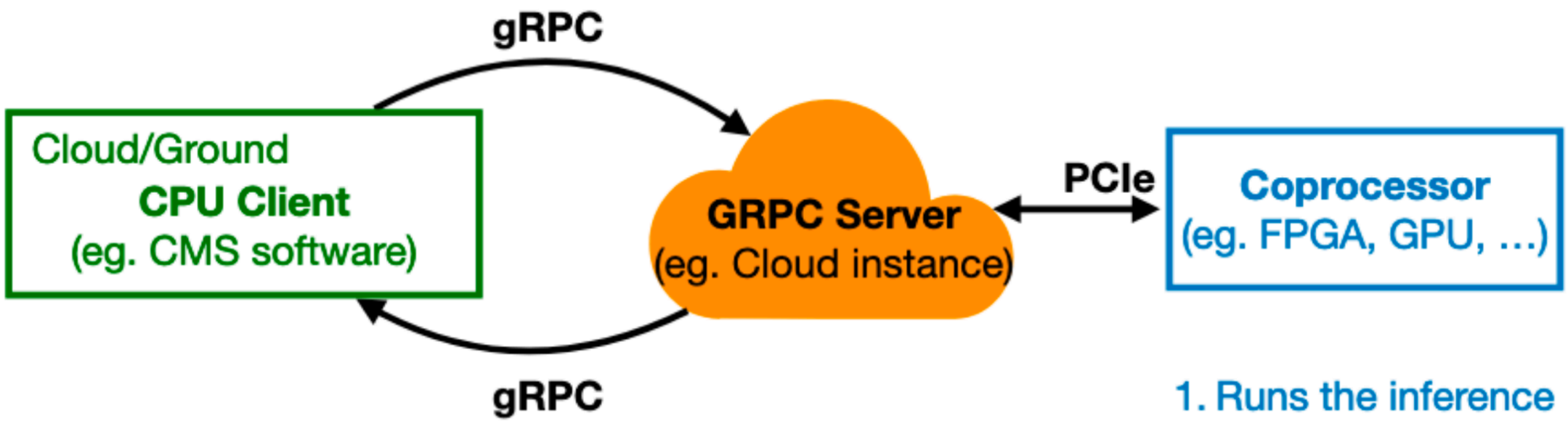
Heterogenous computing @ LHC

- One coprocessor can serve many CPUs → reduce cost and increase scalability
- Increase heterogeneity: choose best device for each job
- Deploy GPUs, FPGAs, … simultaneously
- Model optimization for the processor could be obtained with available tools (ex, Intel oneAPI [*])



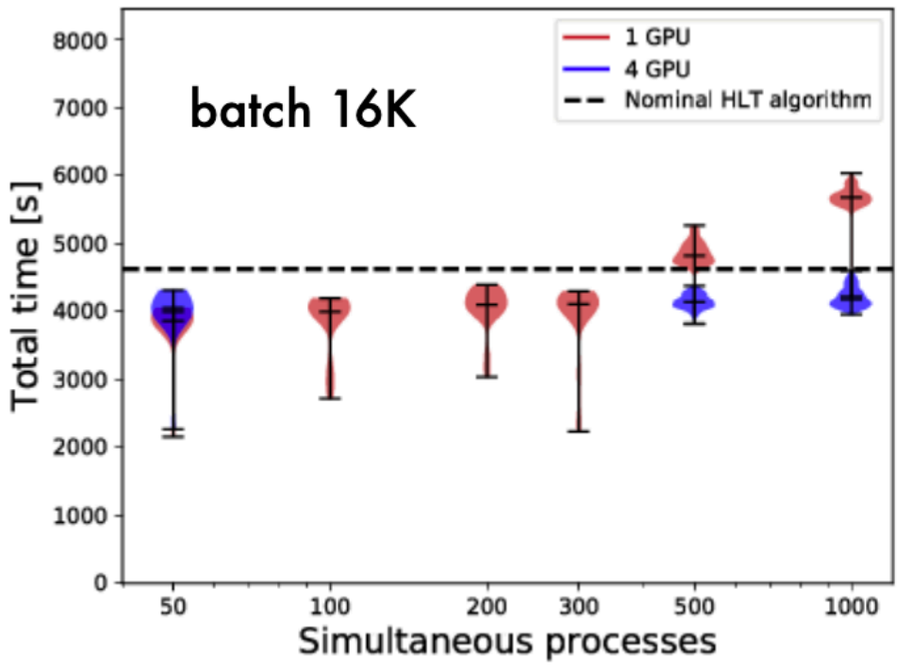
MLaaS with SONIC

- **Services for Optimized Network Inference on Coprocessors** (SONIC) enables inference as a service in experiment software frameworks
  - experiment software (C++) only has to handle converting inputs and outputs between event data format and inference server format
- Uses industry tools as gRPC communication and Nvidia Triton inference servers
- Interacts with cloud services: Azure, AWS, GCP

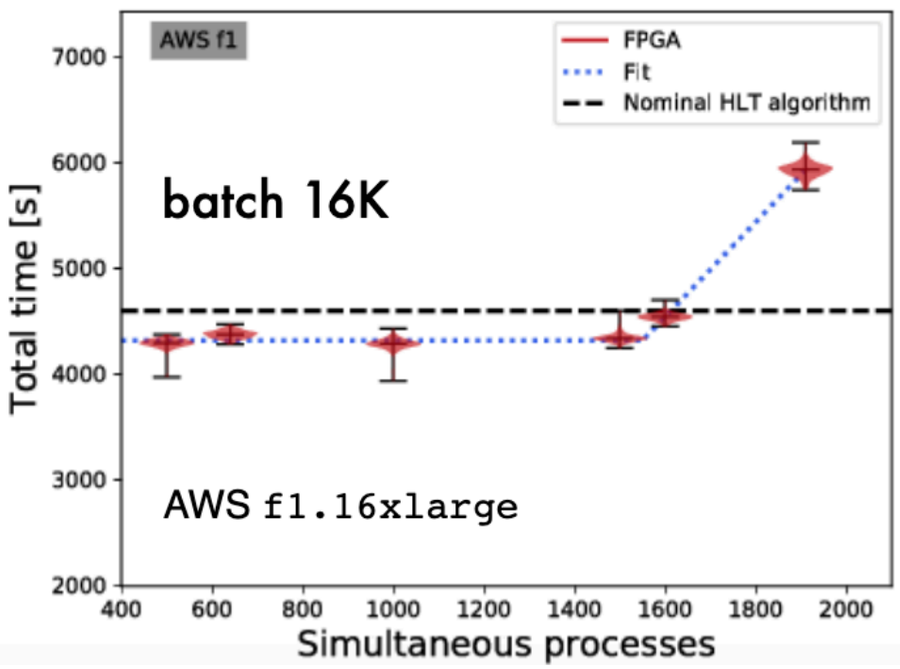


MLaaS with SONIC

Replace hadronic calorimeter reconstruction with ML (2k parameters dense NN here) and enable the model inference in the CMS software with SONIC



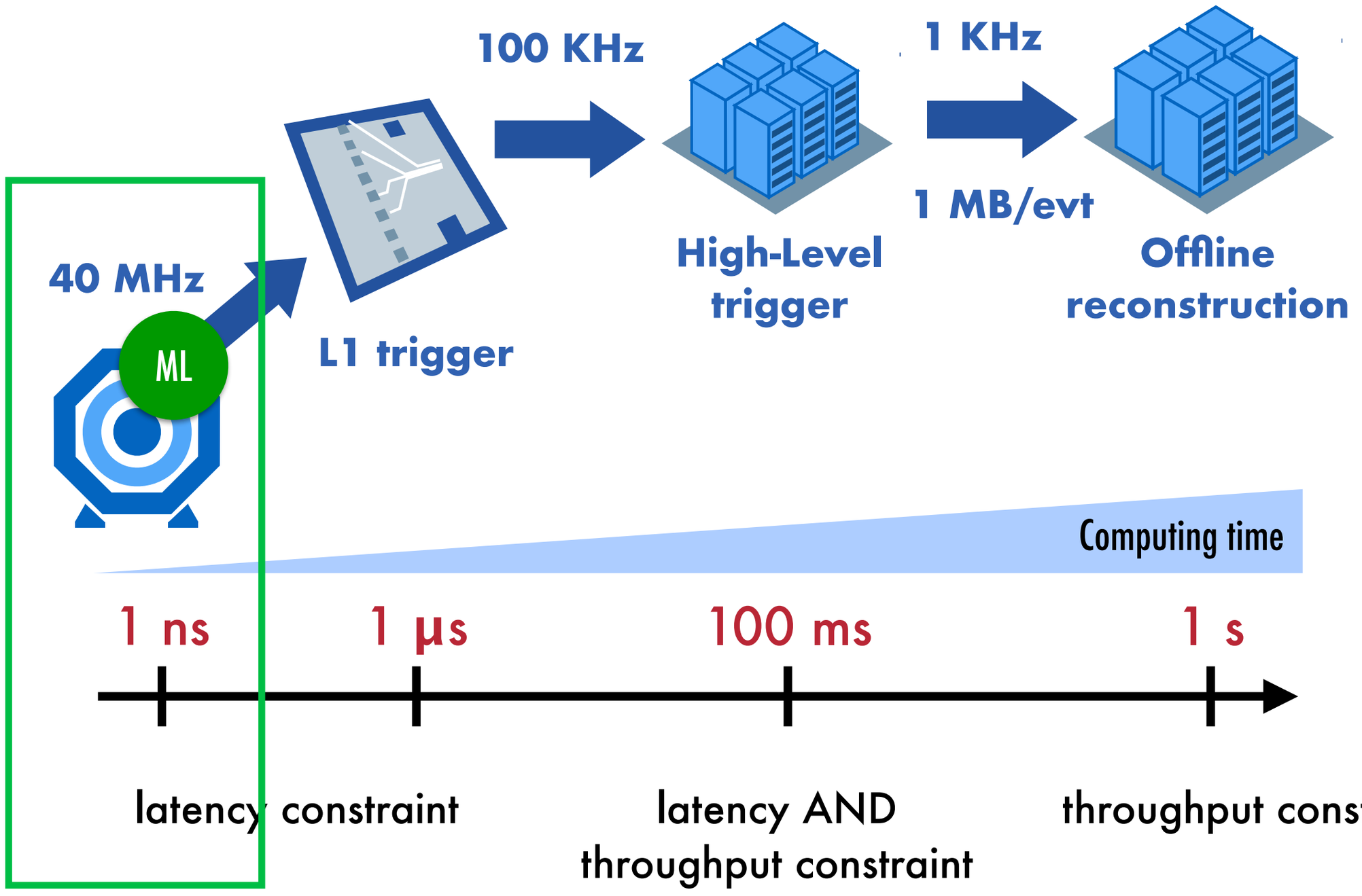
GPU as a service [arxiv.2007.10359]
Each client is given 7,000 events
A single GPU can serve up to 500 HLT nodes with 10% increase in throughput



FPGA as a service [arxiv.2010.08556]
A single service server capable of serving 1500 simultaneous clients while preserving throughput
25Gbps network bandwidth limit hit above 1500

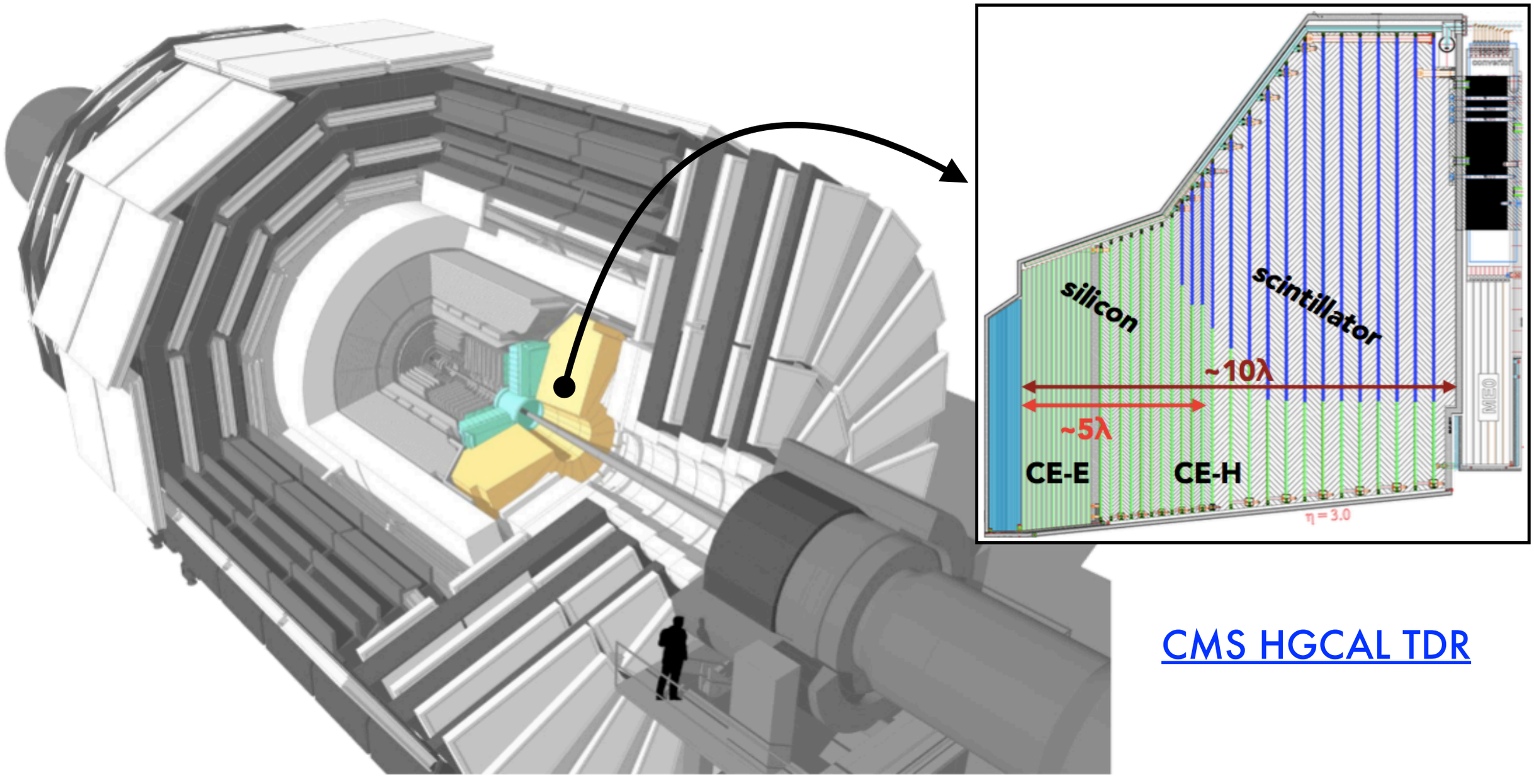
The need for **fast ML**

ASICs typically used at the front end for sensors read out:
directly embed ML in here to allow intelligent data compression before transmission



Example:
High-granularity calorimeter @ HL-LHC

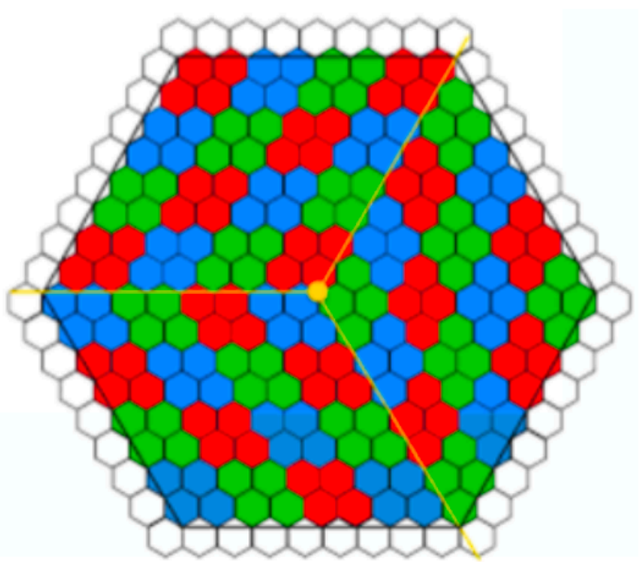
Novel technology for CMS endcap calorimeter:
52 layers with unprecedented number of readout channels!



[CMS HGCAL TDR]

Example: CMS HG calorimeter

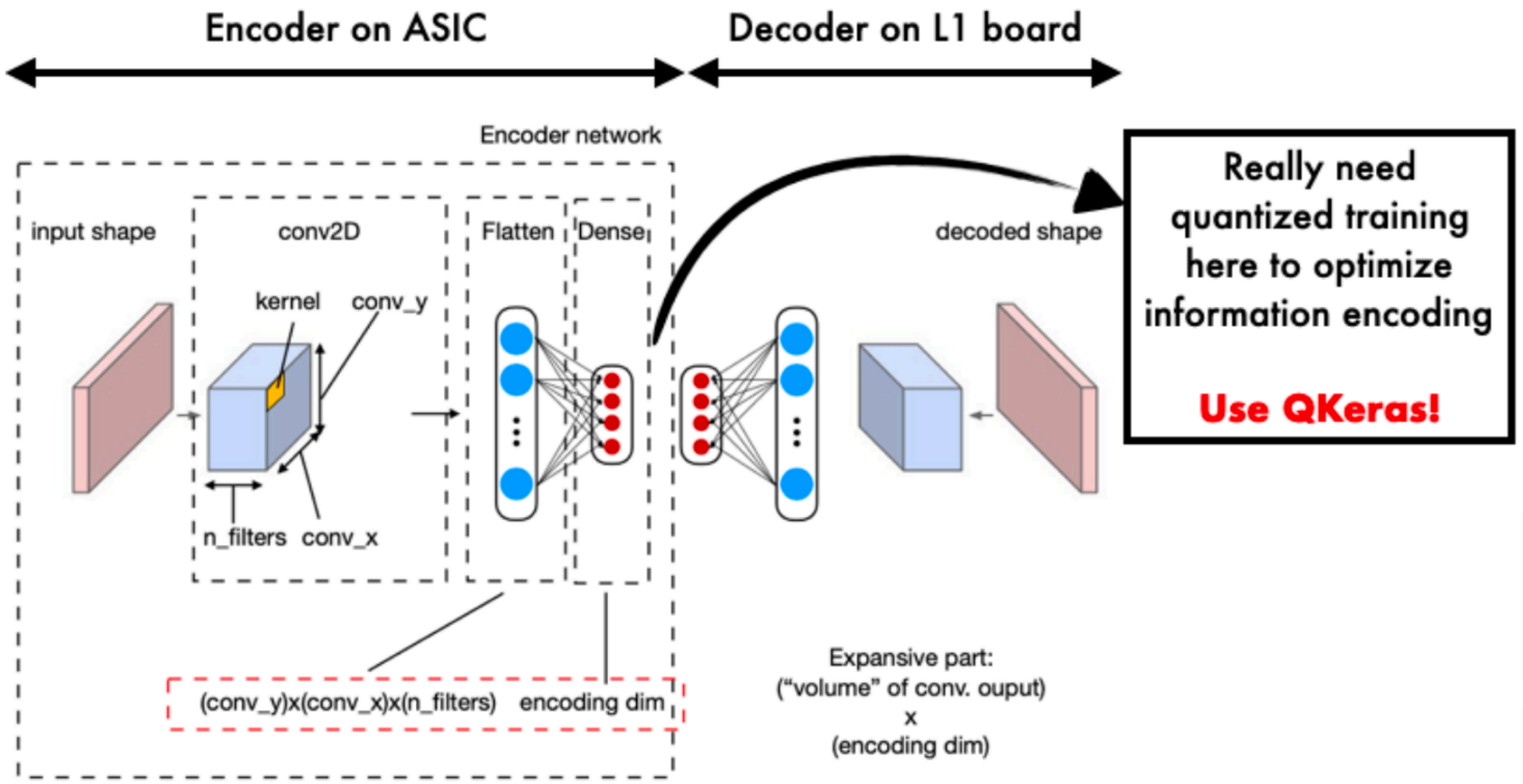
Input
48 "trigger cells"
7b floating point
(336b total)



ASIC

Output:
"Super trigger cell" algo
3[16 TC sum] x 13b = 39b

Can we do a better job of encoding the info in those bits w/o so much loss in granularity?



Really need quantized training here to optimize information encoding

Use QKeras!

What you have learned today

- Machine learning models are intrinsically parallelizable and can be executed efficiently on suitable hardware
- Could replace our standard physics-inspired algorithms which are instead typically sequential
- To gain from this potential down to ultra-low latency the hls4ml library was developed to translate your favourite ML model to an efficient FPGA implementation
- We hope you have gained some experience with hls4ml
  - tutorial always available at https://cern.ch/ssummers/hls4ml-tutorial
  - or if you want to run locally https://github.com/fastmachinelearning/hls4ml-tutorial (need Vivado installation)
- Stay tune for all new features at https://github.com/fastmachinelearning/hls4ml
- And for fast machine learning updates beyond hls4ml check https://fastmachinelearning.org/projects.html