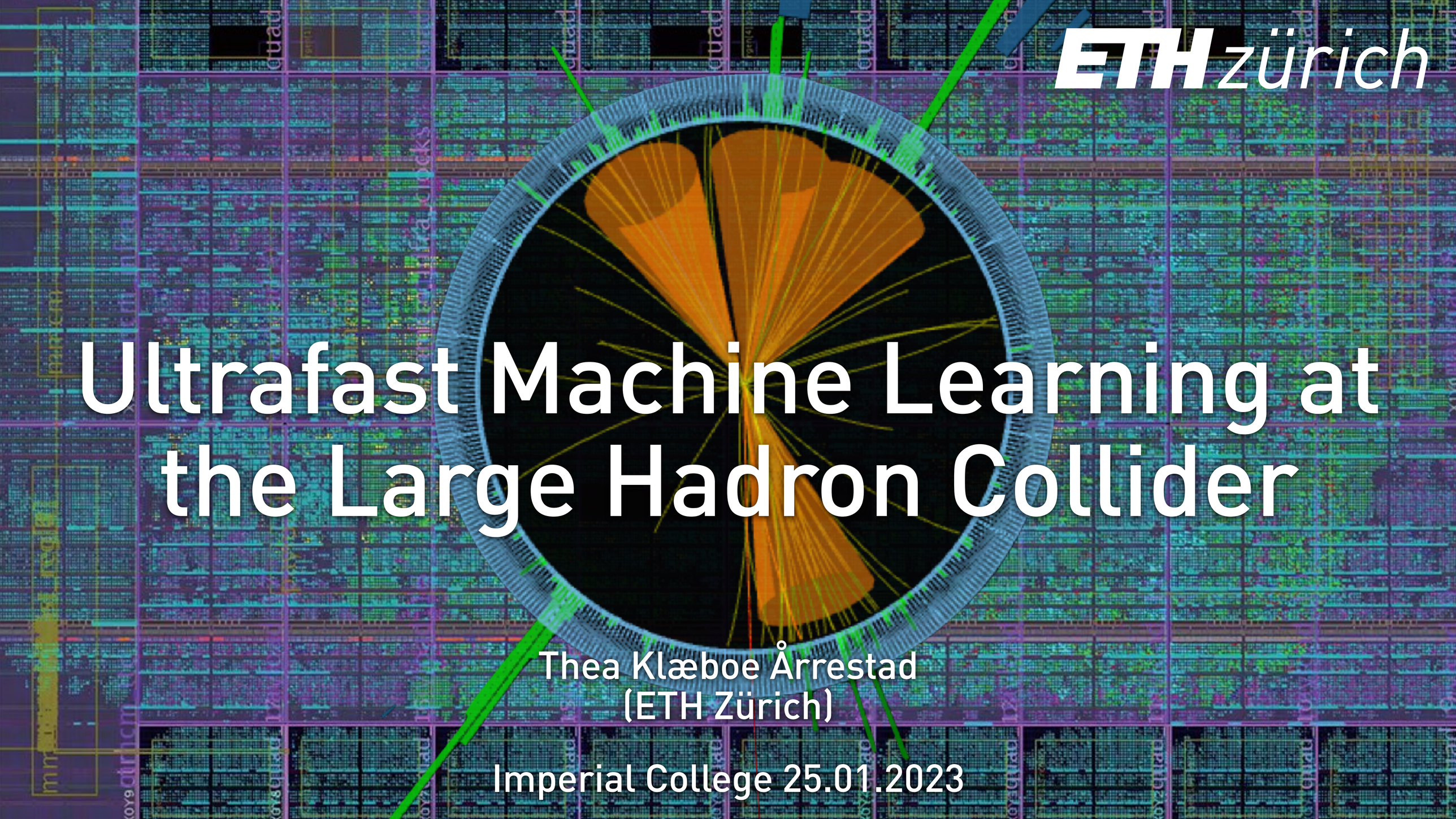
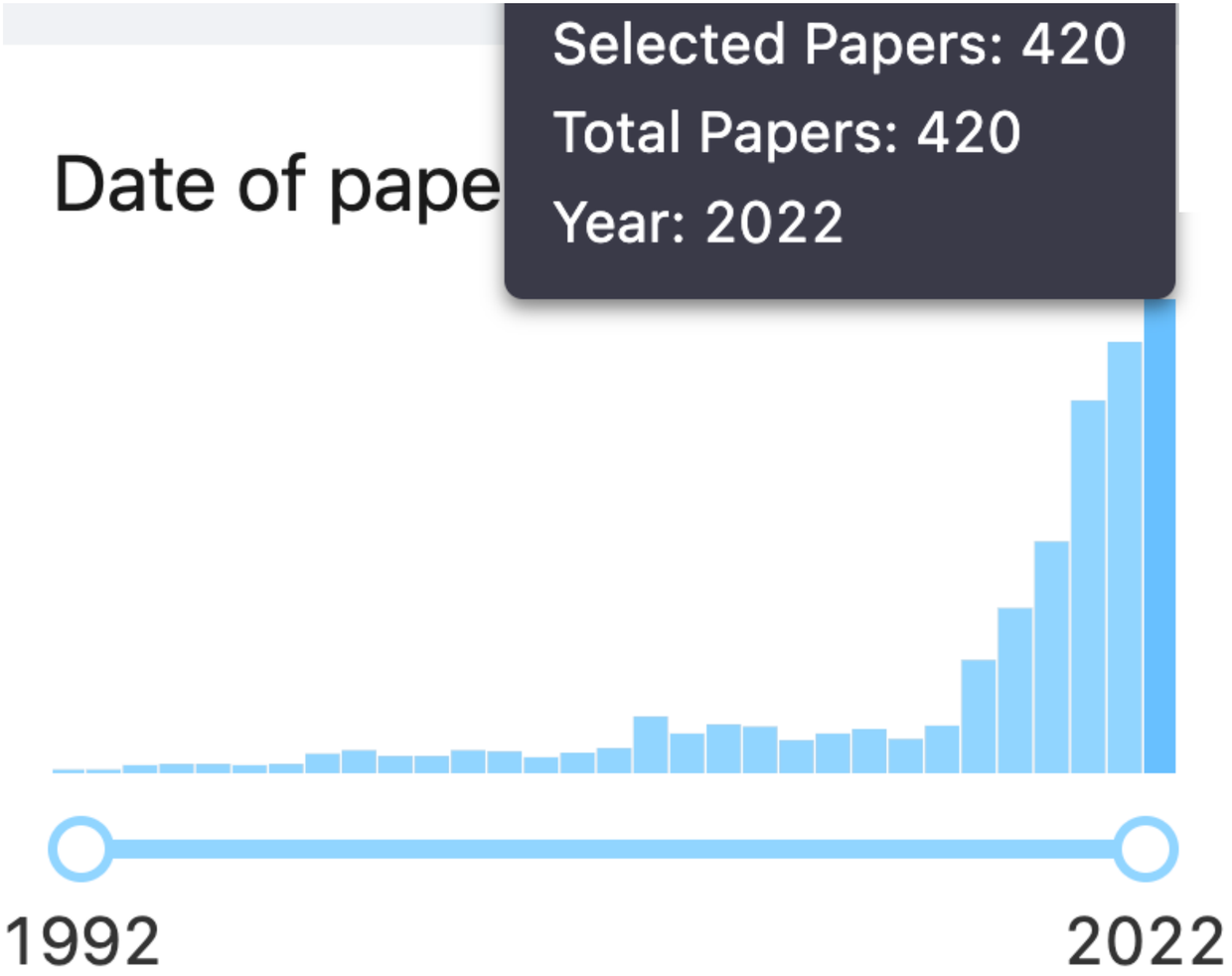


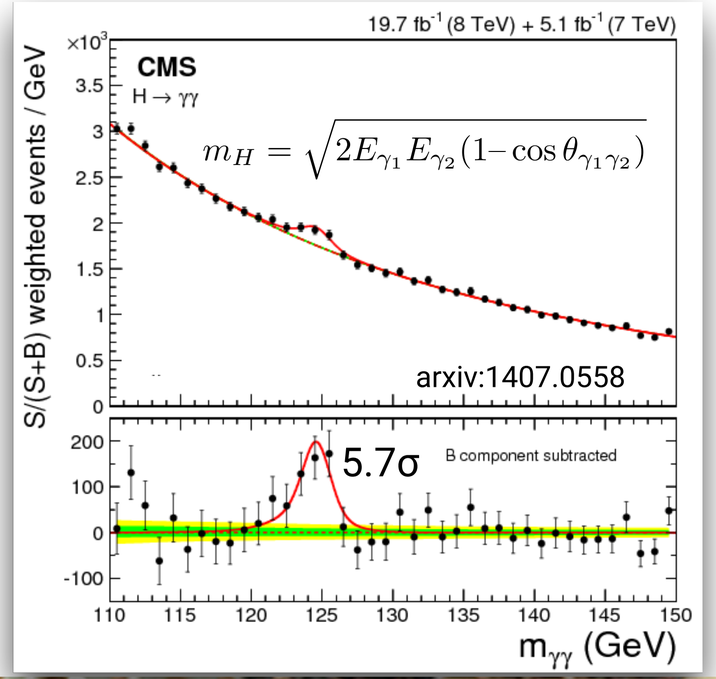
Ultrafast Machine Learning at the Large Hadron Collider

Thea Klæboe Årrestad
(ETH Zürich)

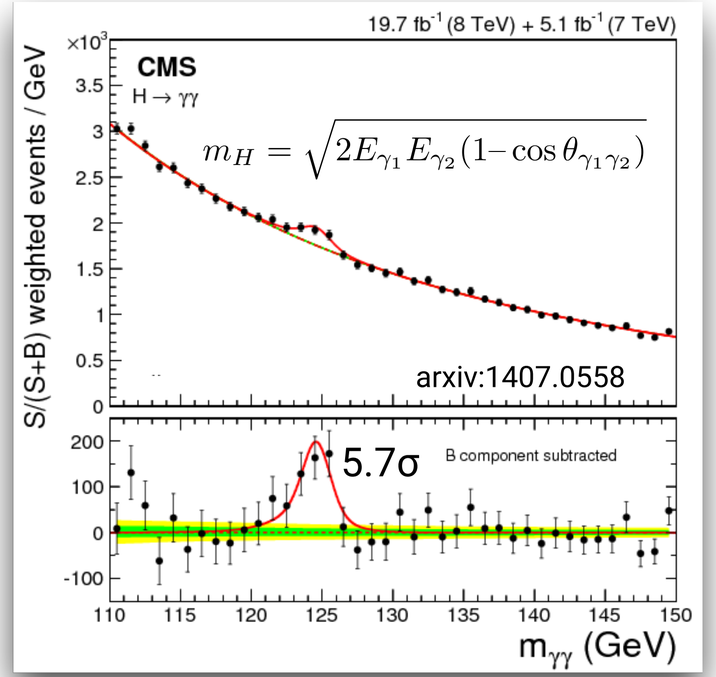
Imperial College 25.01.2023

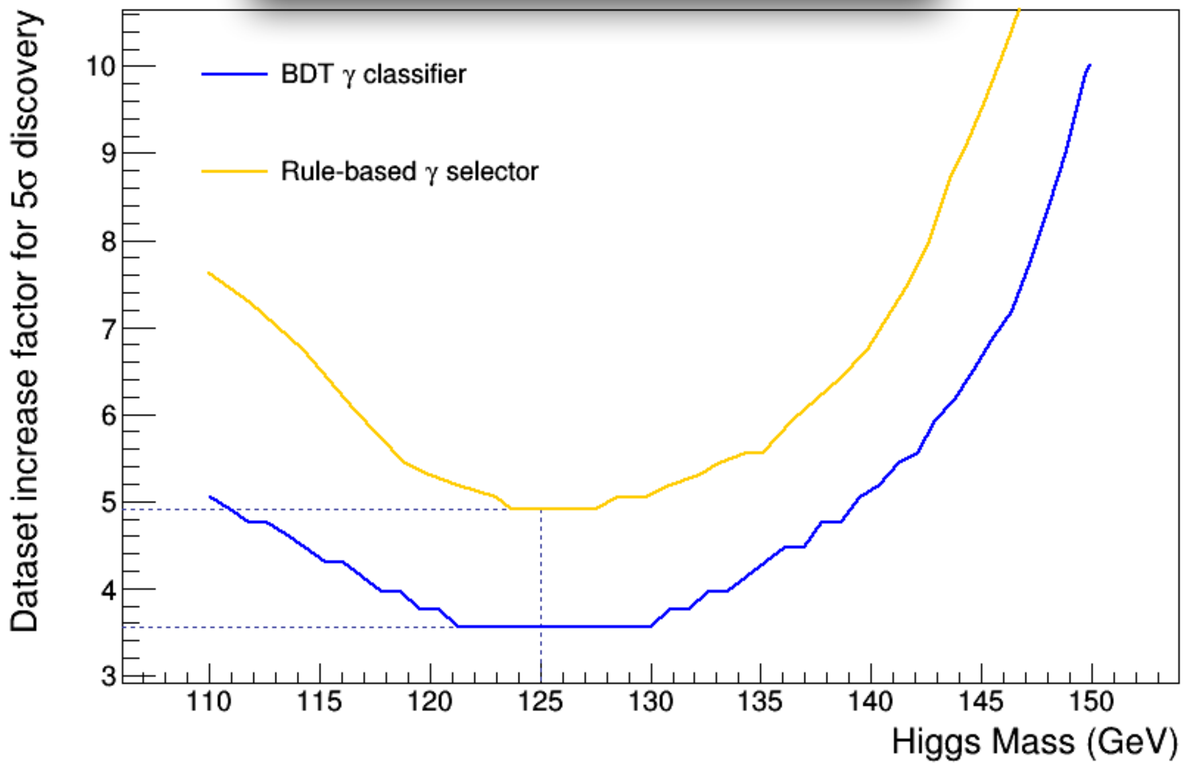
Inspire:
("machine learning" or "deep learning" or neural) and (hep-ex or hep-ph or hep-th)




19.7 fb-1 (8 TeV) + 5.1 fb-1 (7 TeV)
CMS
H → γγ
$m_H = \sqrt{2E_{\gamma_1}E_{\gamma_2}(1-\cos\theta_{\gamma_1\gamma_2})}$
arxiv:1407.0558
S/(S+B) weighted events / GeV
×10^3
5.7σ
B component subtracted
m_γγ (GeV)

Nature Review





Google

chat gpt

All | News | Images | Videos | Books | More | Tools

About 10,700 results (0.31 seconds)

The New York Times

Opinion | ChatGPT Has a Devastating Sense of Humor

The chat bot makes a lot of mistakes. But it's fun to talk to, and it knows its limitations.

3 weeks ago

The New York Times

Can ChatGPT Make This Podcast?

It's writing podcast scripts, finishing students' homework and correcting mistakes in computer code: ChatGPT, the A.I. chatbot from OpenAI,...

4 weeks ago

The New York Times

How to Use ChatGPT and Still Be a Good Person

It's a turning point for artificial intelligence, and we need to take advantage of these tools without causing harm to ourselves or others.

2 weeks ago

The New York Times

Did Artificial Intelligence Just Get Too Smart?

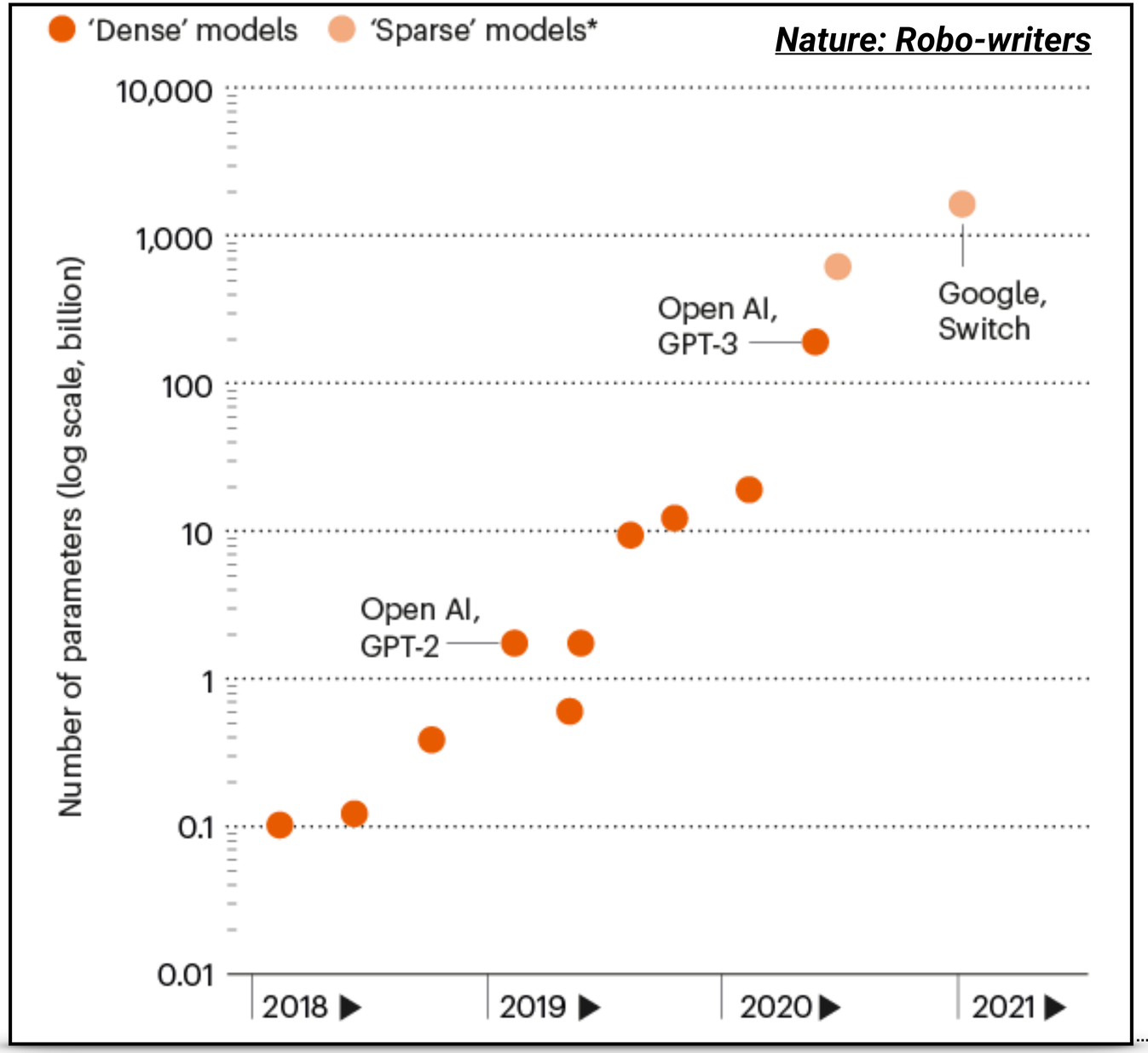
The power and potential of a technology called ChatGPT have led some to claim it heralds a new era in computing.

3 weeks ago

The New York Times

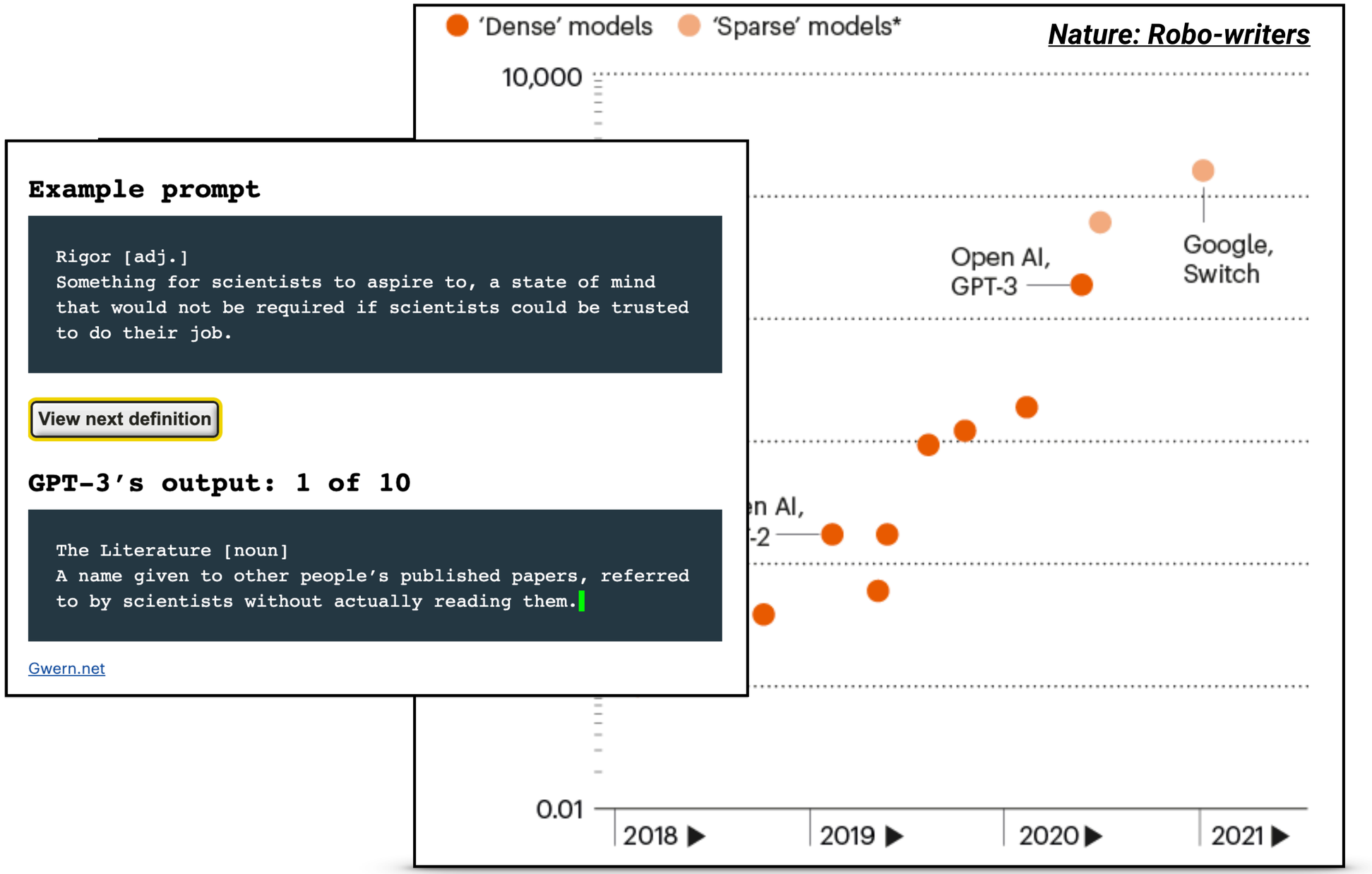
ChatGPT is Social Media's Newest Star

Social media's newest star is a robot: a program called ChatGPT that tries to answer questions like a person. Since its debut last week,...

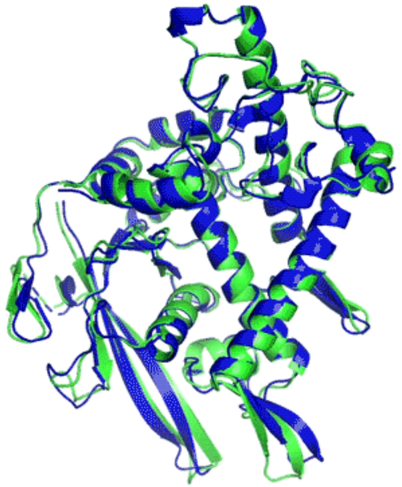


GPT-4: 100 trillion?!)

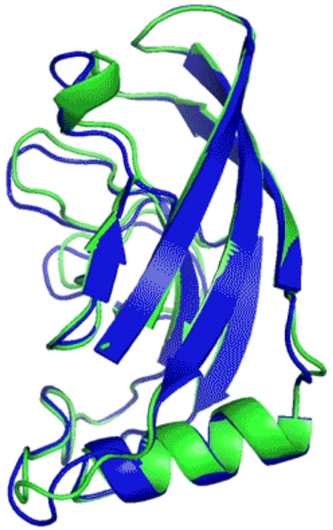
GPT-3: 175 billion parameters (0.16% of the human brain)



GPT-3: 175 billion parameters (0.16% of the human brain)

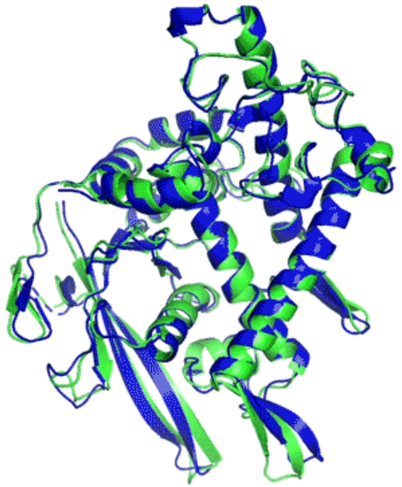


T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)

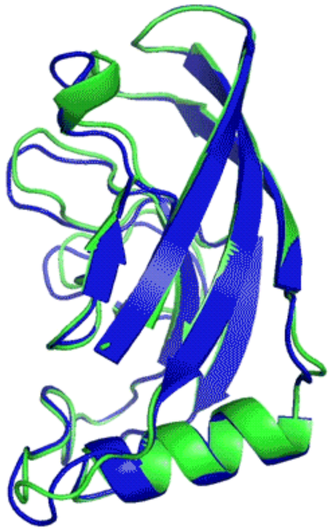


T1049 / 6y4f
93.3 GDT
(adhesin tip)

Experimental result
Computational prediction



T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)

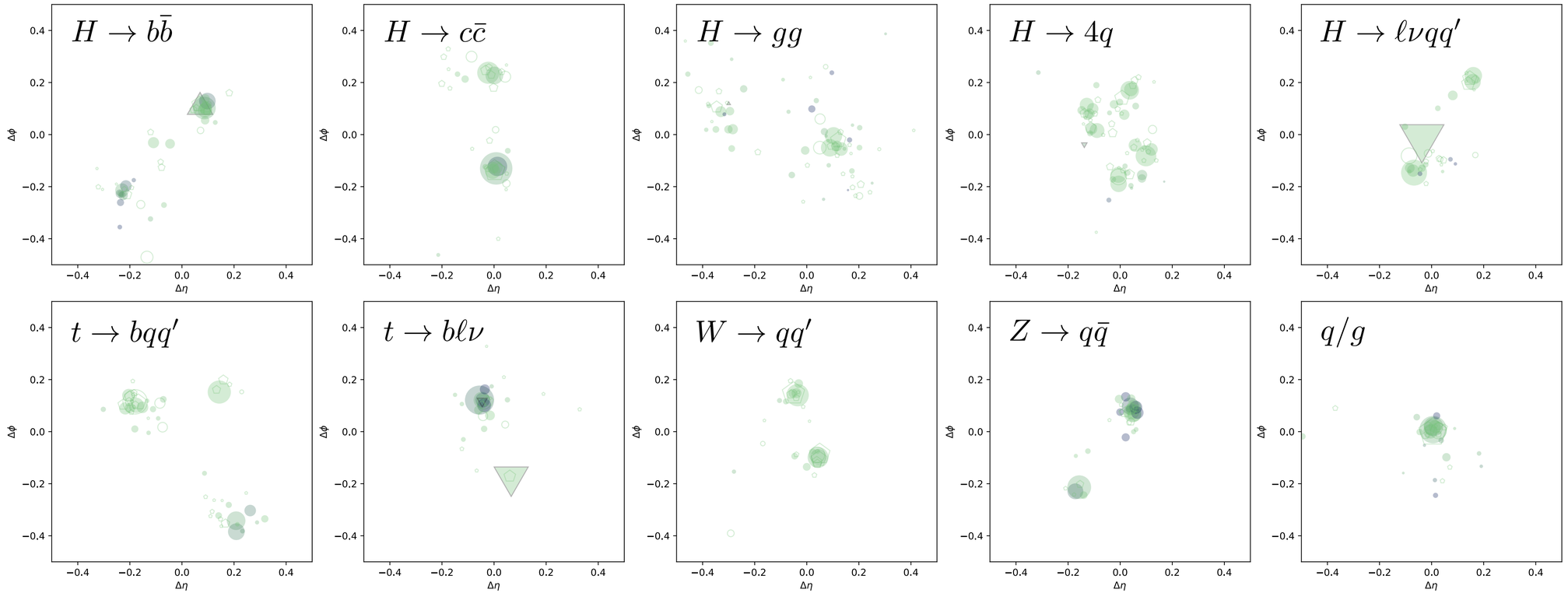


T1049 / 6y4f
93.3 GDT
(adhesin tip)

Experimental result
Computational prediction

100 million jets for training

	Accuracy	# params
PFN	0.772	86.1 k
P-CNN	0.809	354 k
ParticleNet	0.844	370 k
ParT	**0.861**	2.14 M
ParT (plain)	0.849	2.13 M

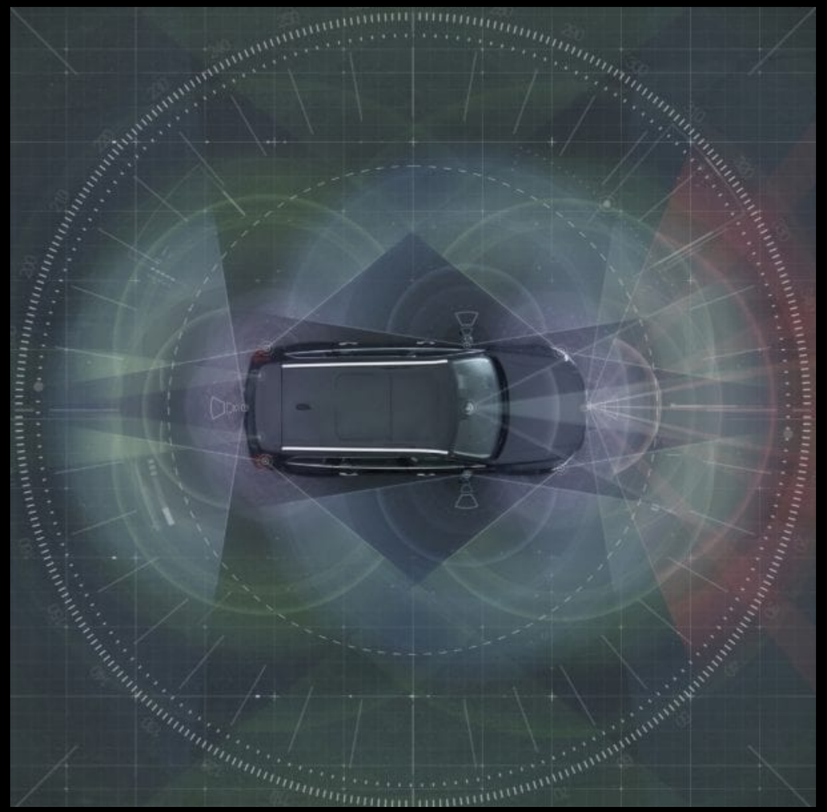


"Particle Transformer For Jet Tagging" H. Qu, C. Li, S. Qian

EDGE

EDGE

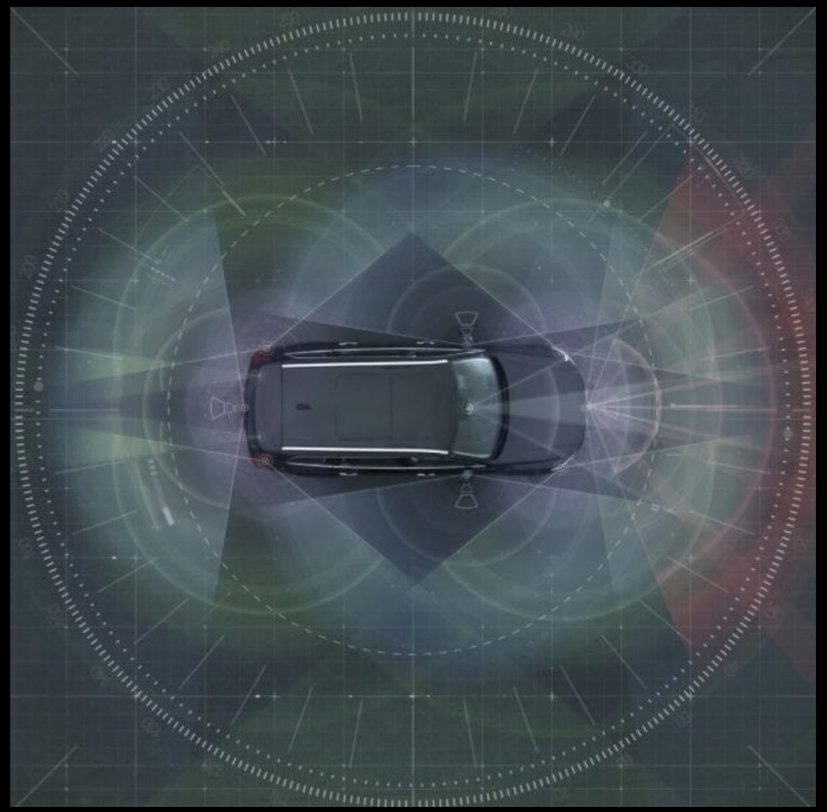
O(1) ms



ASIC/GPU

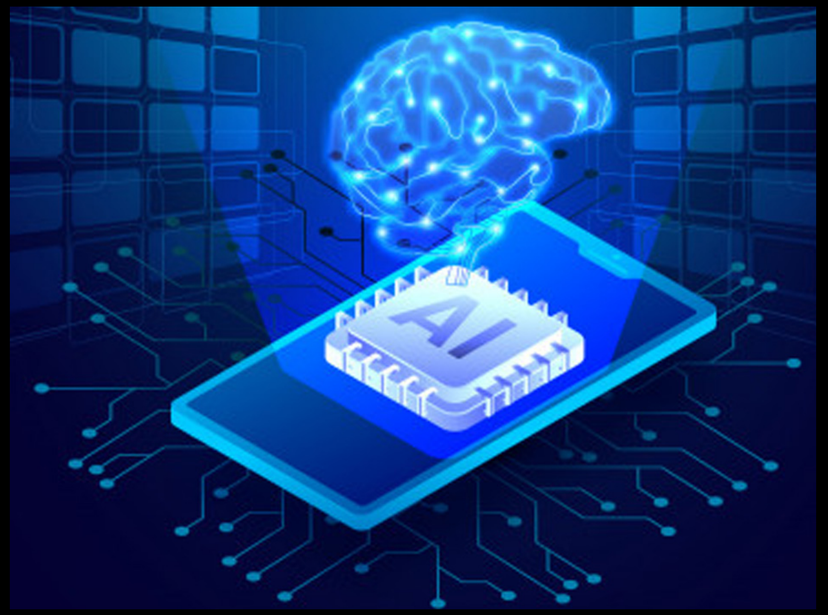
EDGE

O(1) ms



ASIC/GPU

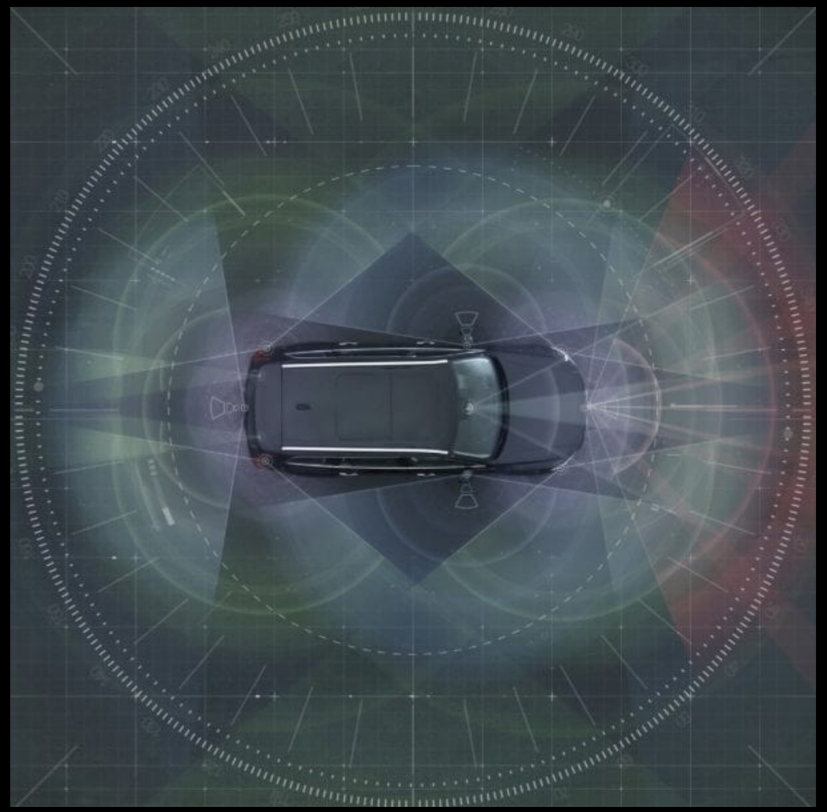
O(1) ms



ASIC

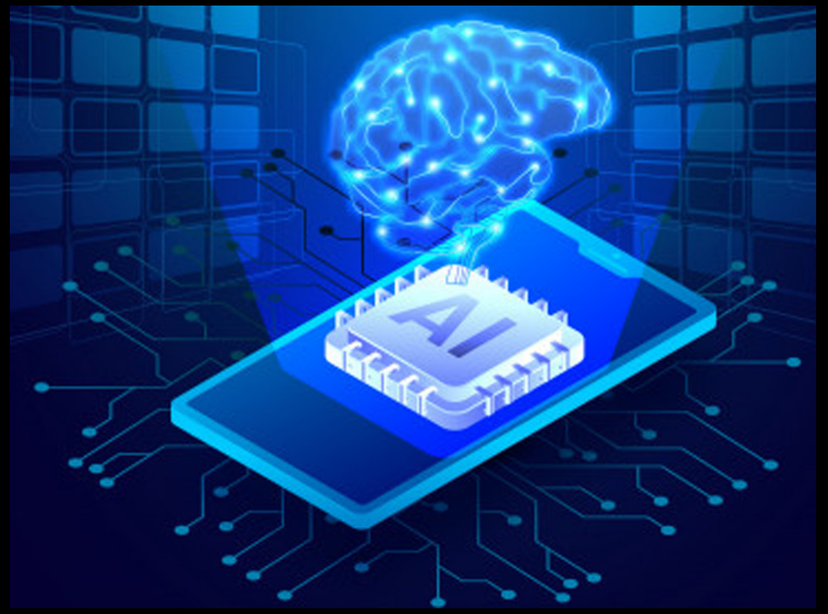
EDGE

O(1) ms



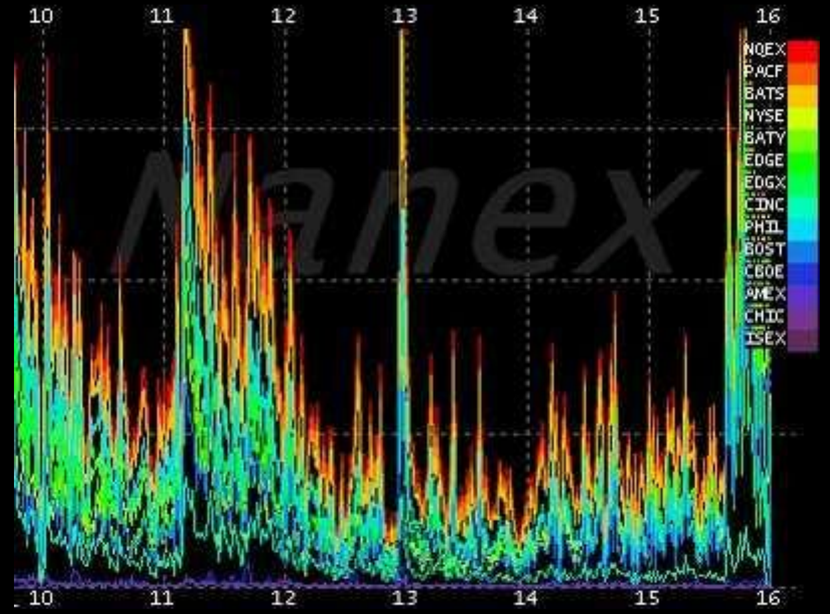
ASIC/GPU

O(1) ms



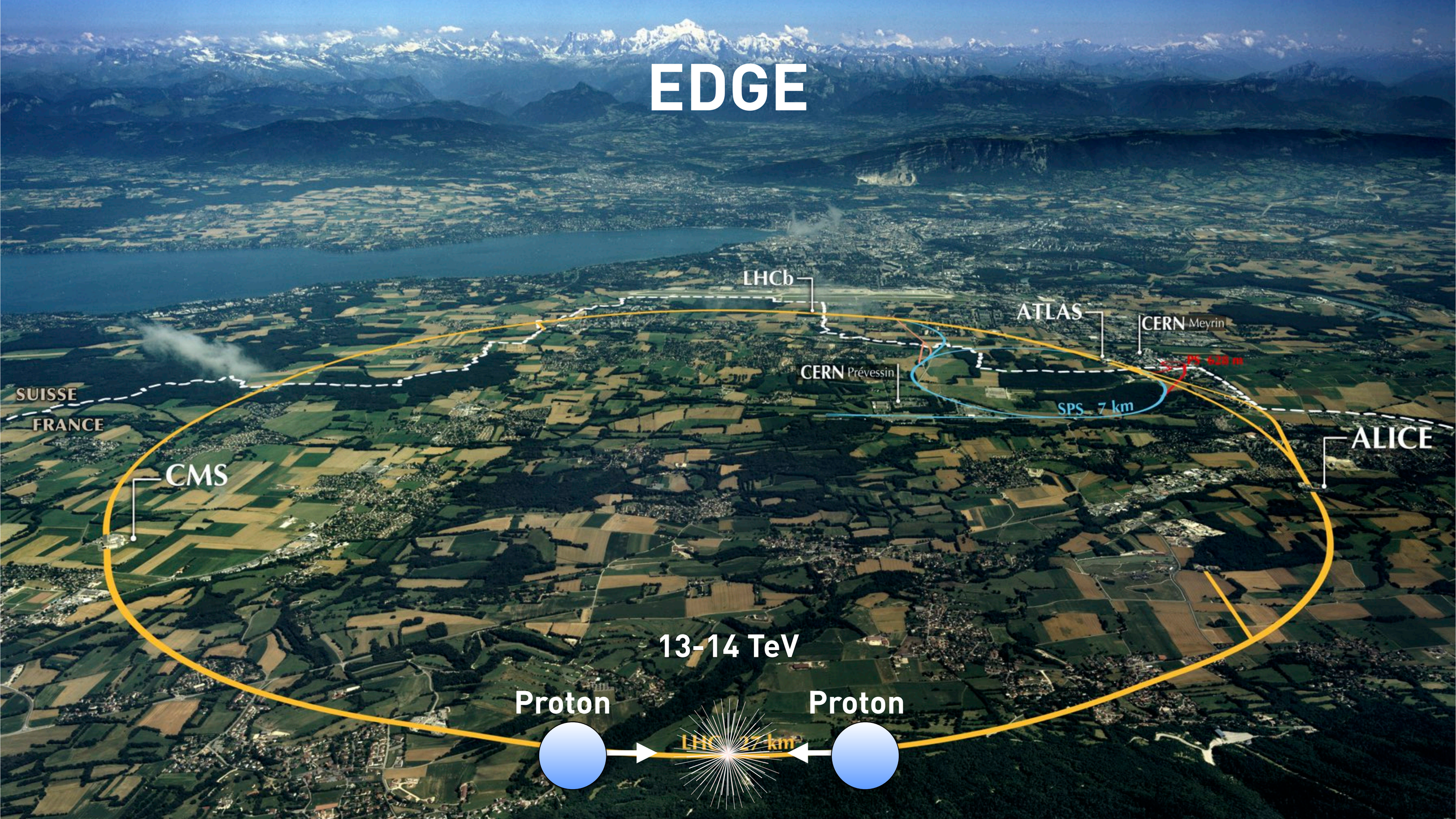
ASIC

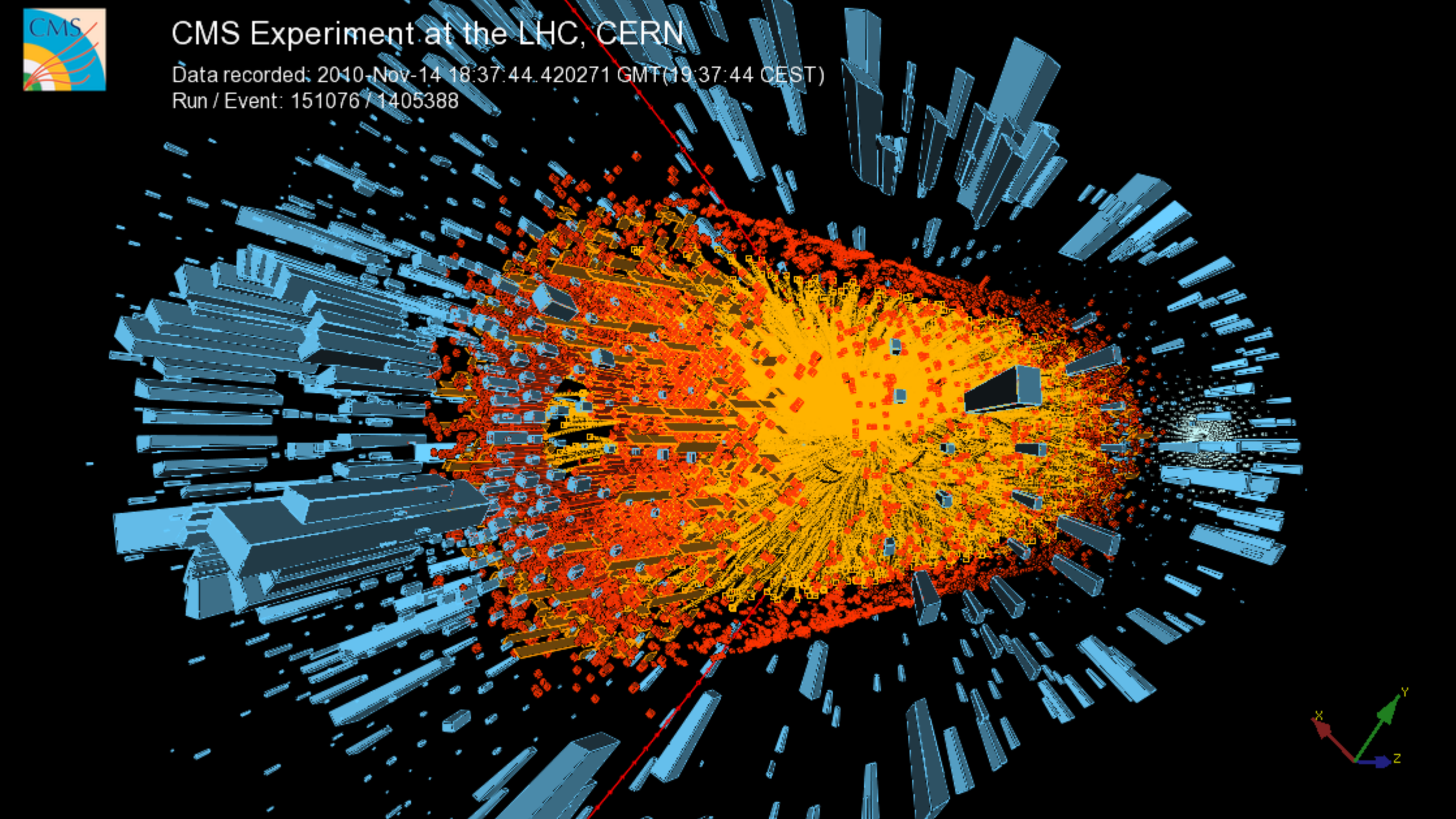
O(1) ns

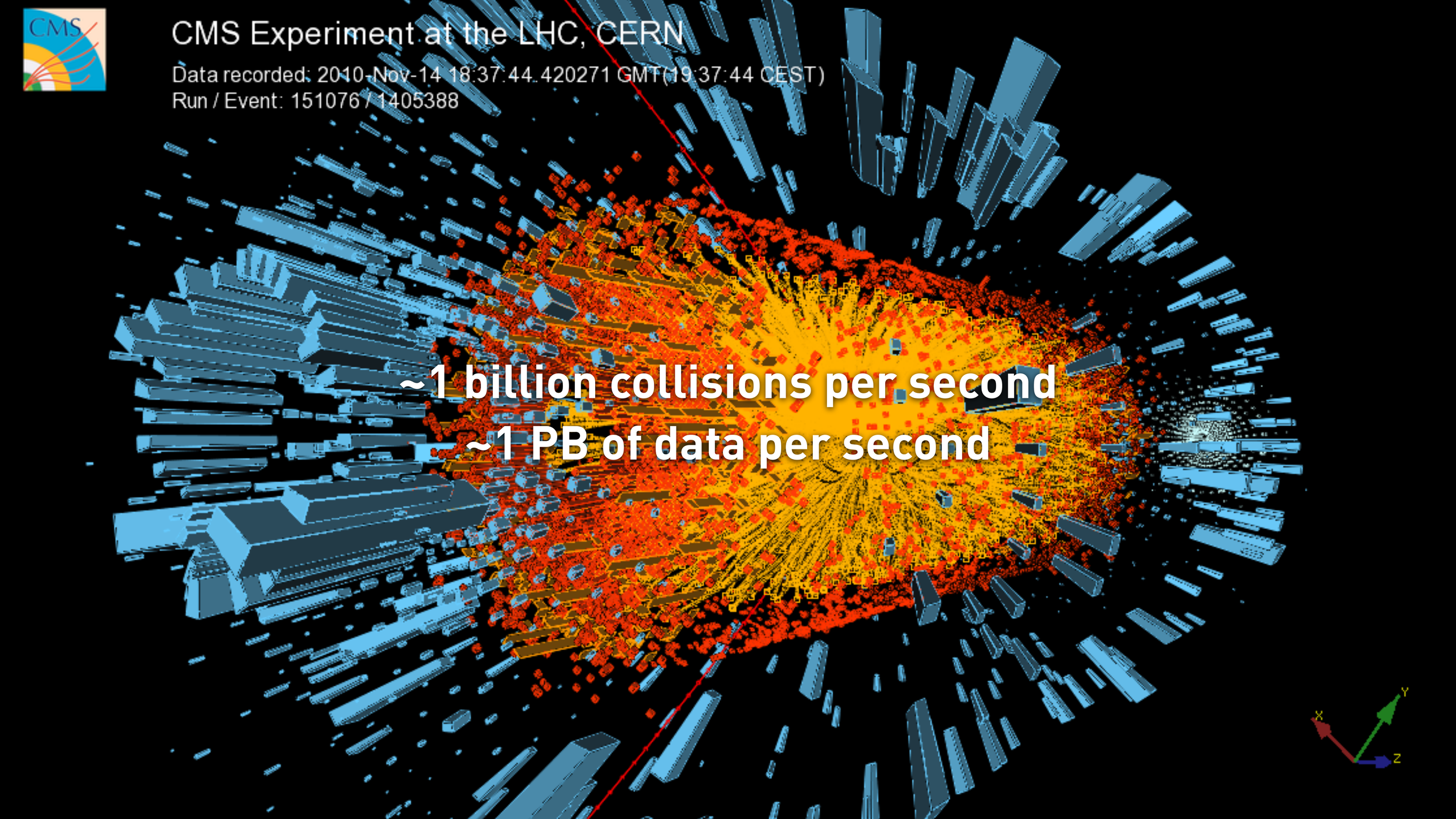


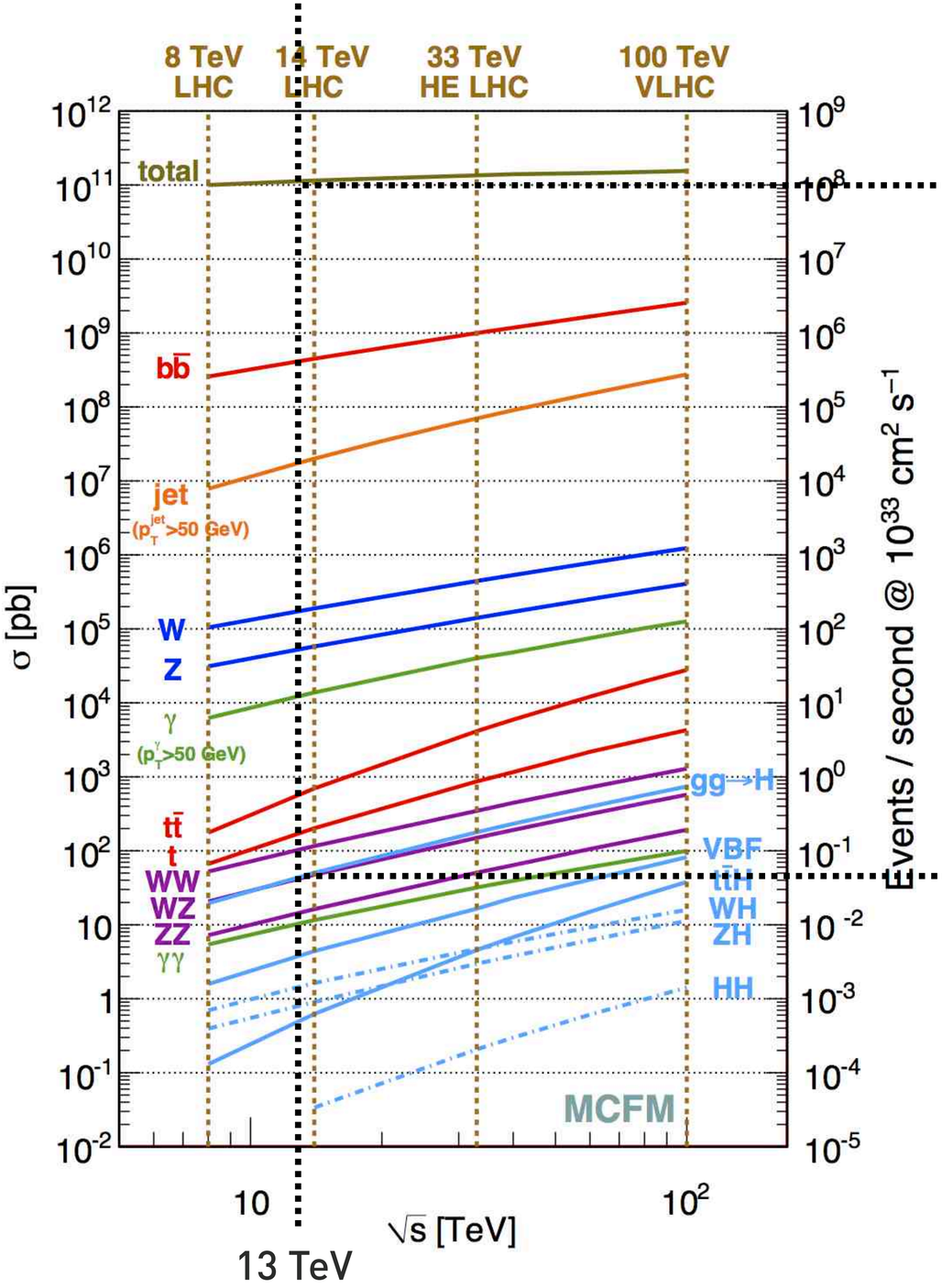
FPGA

EDGE


LHCb
ATLAS
CERN Meyrin
PS 628 m
CERN Prévessin
SPS 7 km
SUISSE
FRANCE
ALICE
CMS
13-14 TeV
Proton
Proton
LHC 27 km


CMS Experiment at the LHC, CERN
Data recorded: 2010-Nov-14 18:37:44.420271 GMT(19:37:44 CEST)
Run / Event: 151076 / 1405388
Y
X
Z


CMS Experiment at the LHC, CERN
Data recorded: 2010-Nov-14 18:37:44.420271 GMT(19:37:44 CEST)
Run / Event: 151076 / 1405388
~1 billion collisions per second
~1 PB of data per second
Y
X
Z

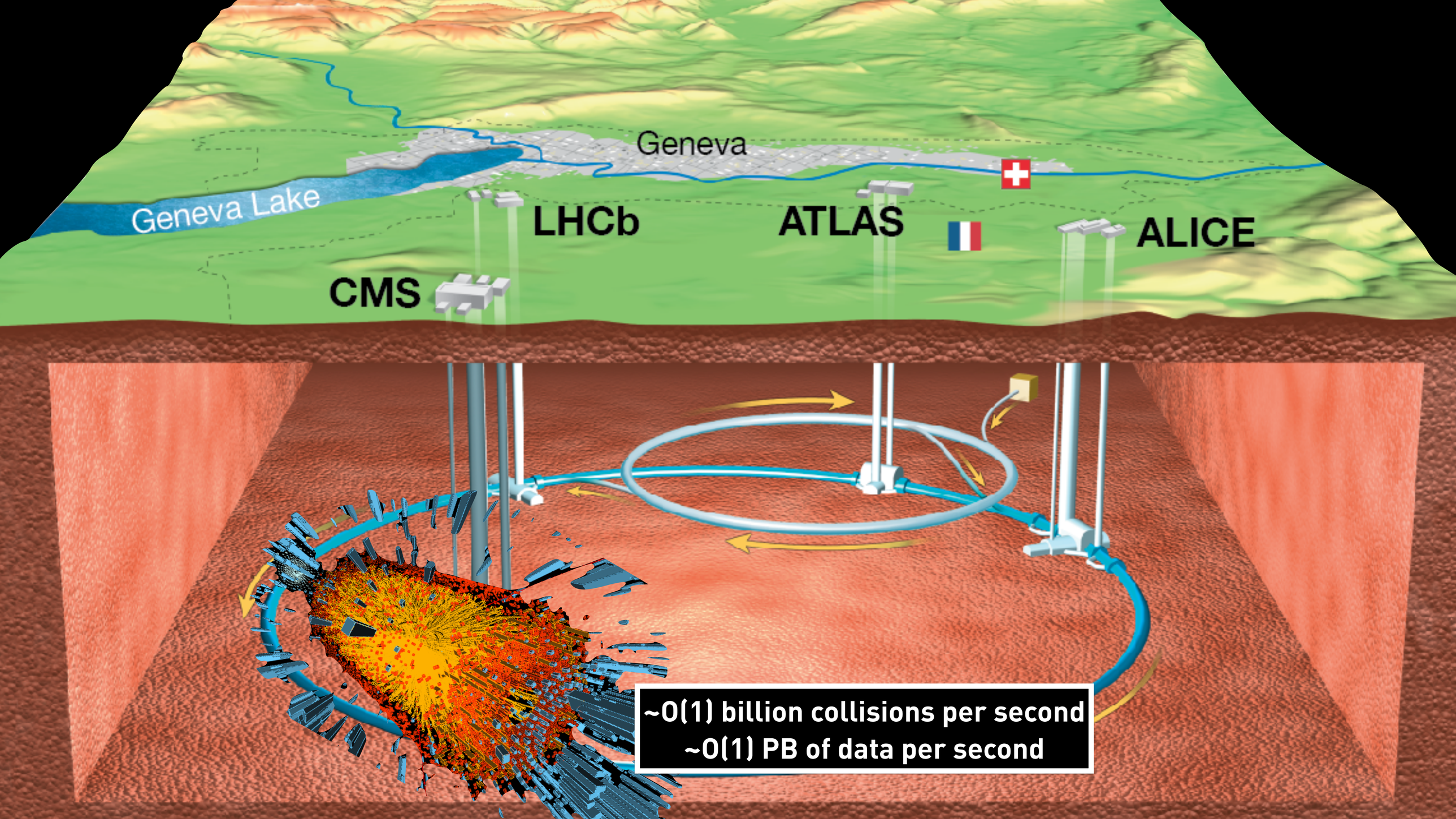


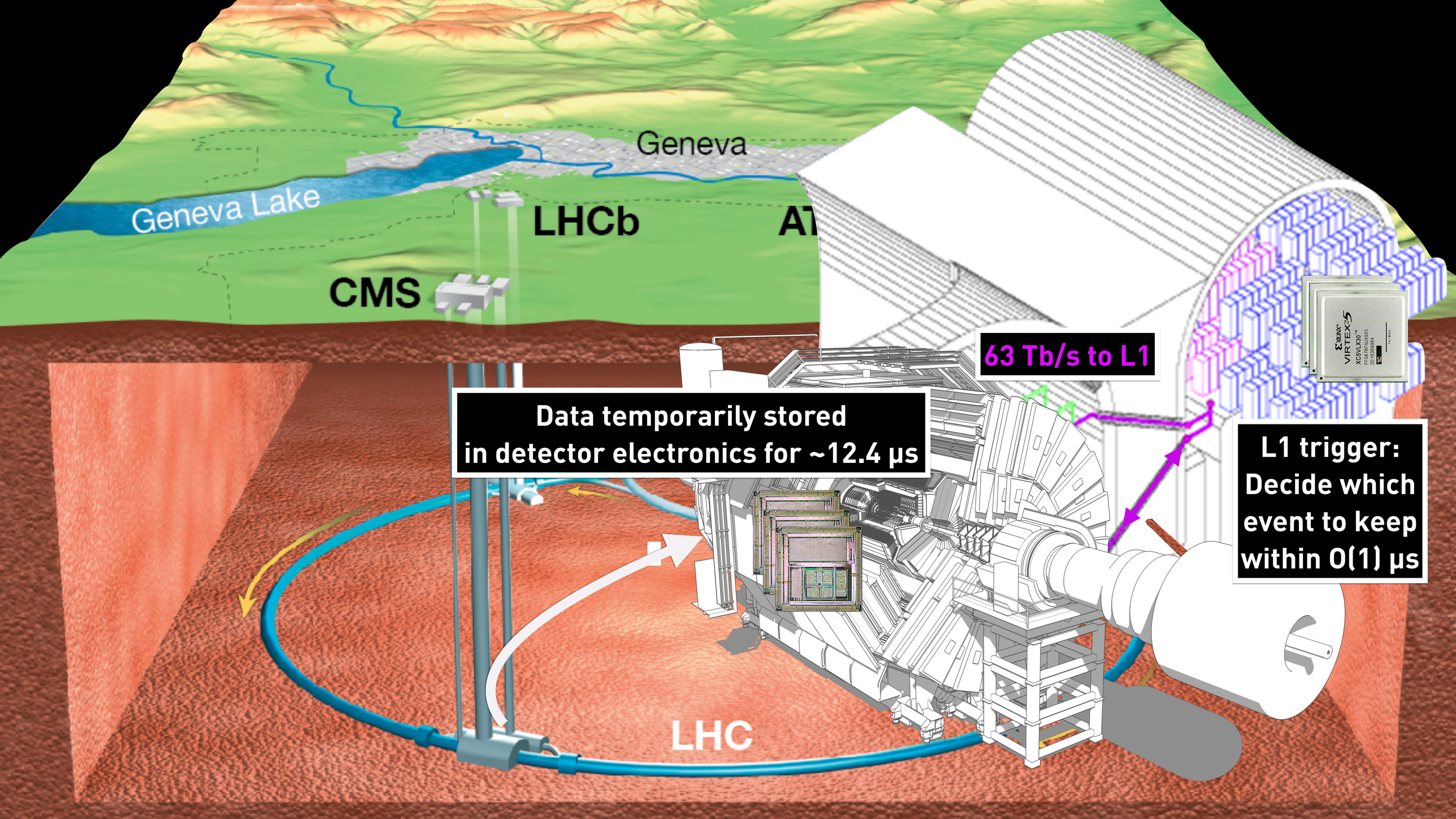
Total

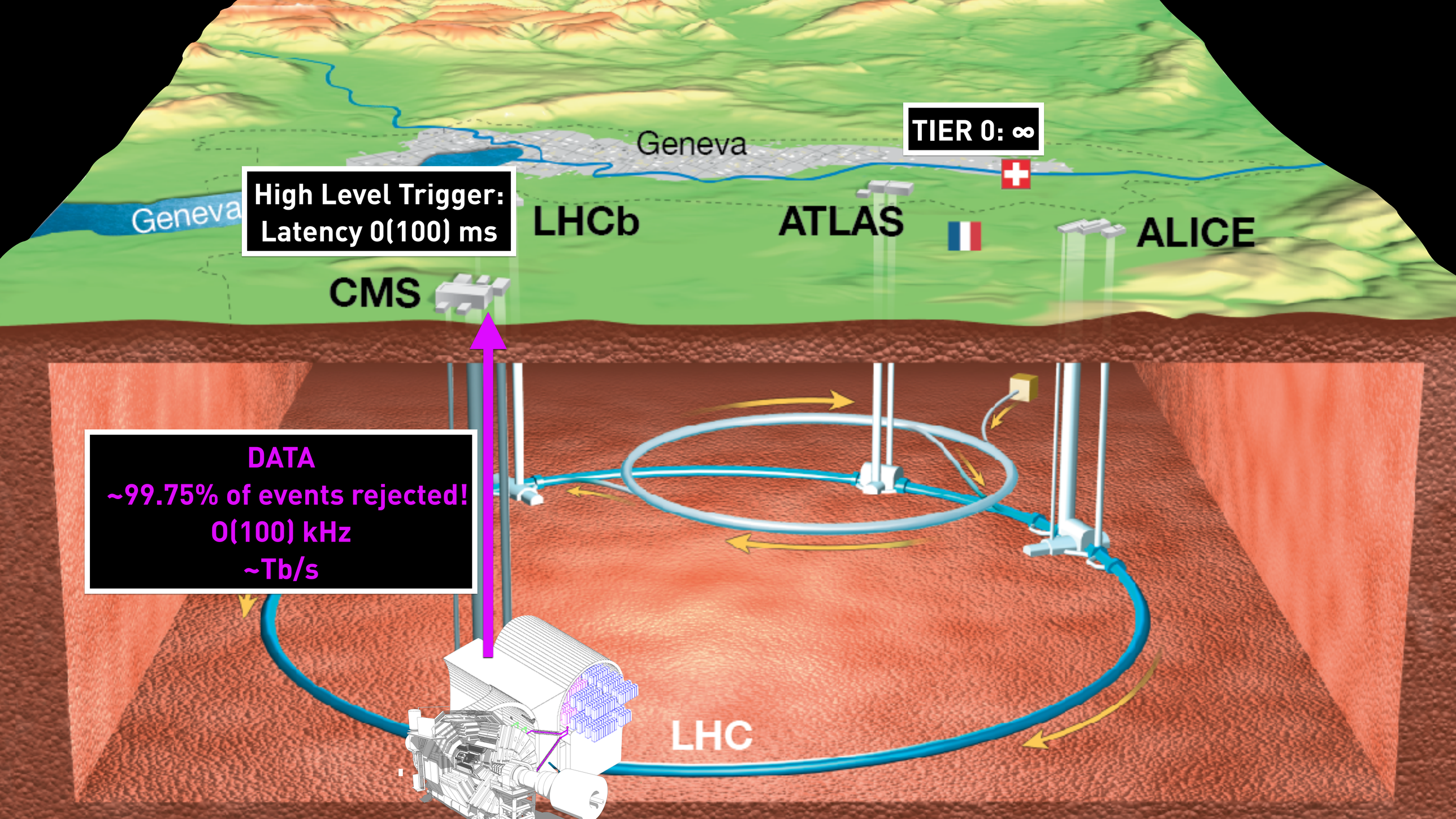
Higgs produced in
~1 out of 10^9 collisions

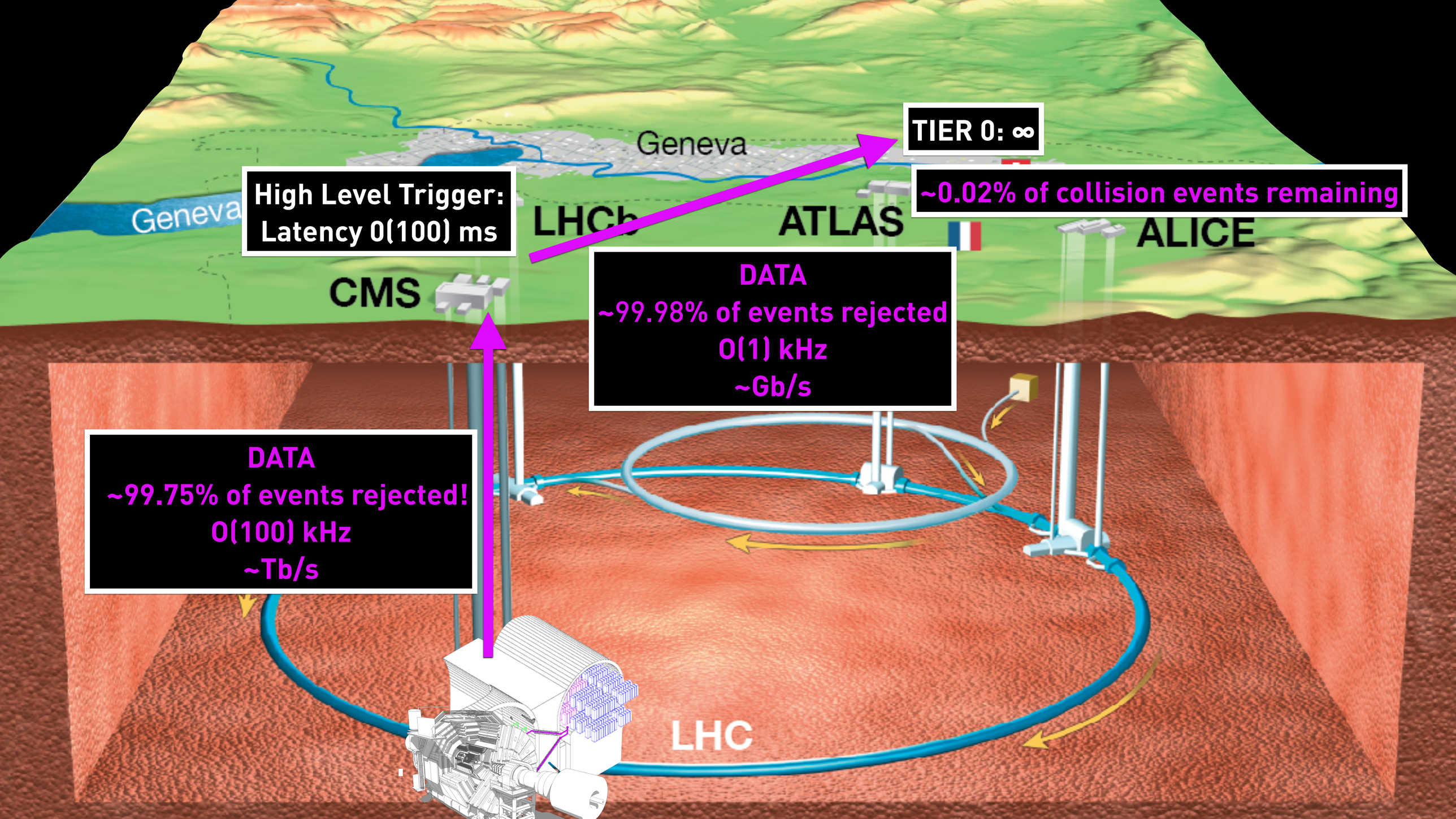
Saving all collisions not useful
(even if we could)!

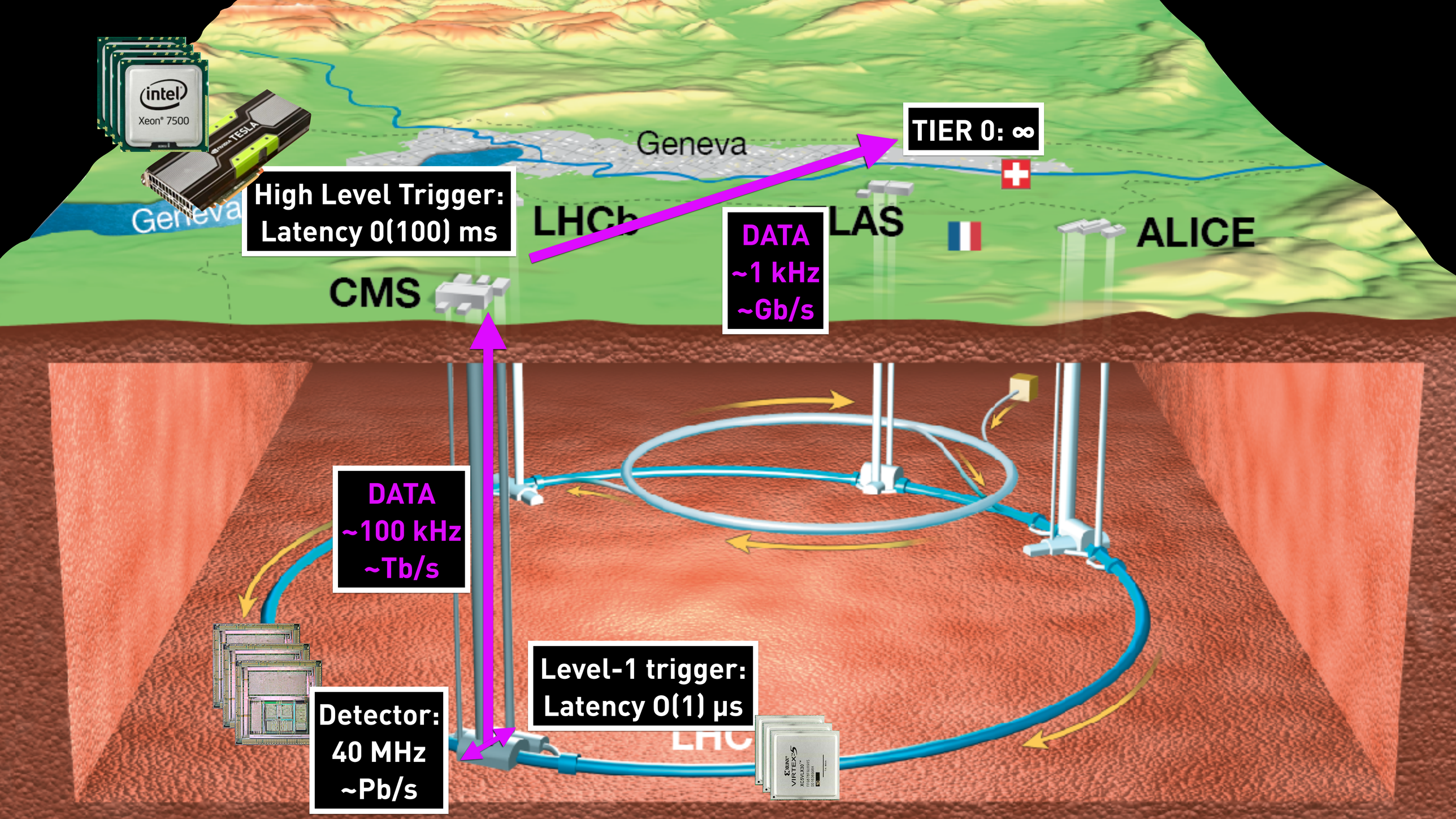
gg→H


Geneva
Geneva Lake
LHCb
ATLAS
ALICE
CMS
~O(1) billion collisions per second
~O(1) PB of data per second


Geneva
Geneva Lake
LHCb
CMS
63 Tb/s to L1
Data temporarily stored
in detector electronics for ~12.4 µs
L1 trigger:
Decide which
event to keep
within O(1) µs
LHC


Geneva
Geneva
TIER 0: ∞
High Level Trigger:
Latency O(100) ms
LHCb
ATLAS
ALICE
CMS
DATA
~99.75% of events rejected!
O(100) kHz
~Tb/s
LHC


Geneva
Geneva
High Level Trigger:
Latency O(100) ms
TIER 0: ∞
~0.02% of collision events remaining
LHCb
ATLAS
ALICE
CMS
DATA
~99.98% of events rejected
O(1) kHz
~Gb/s
DATA
~99.75% of events rejected!
O(100) kHz
~Tb/s
LHC


Xeon® 7500
TESLA
Geneva
High Level Trigger:
Latency 0(100) ms
TIER 0: ∞
Geneva
LHCb
LAS
ALICE
CMS
DATA
~1 kHz
~Gb/s
DATA
~100 kHz
~Tb/s
Level-1 trigger:
Latency 0(1) µs
Detector:
40 MHz
~Pb/s
LHC
VIRTEX-5
XC5VLX30

High Luminosity LHC

New Physics is produced 1 in 10^{12}

- Need more collisions to observe rare processes

High Luminosity LHC

New Physics is produced 1 in 10^{12}

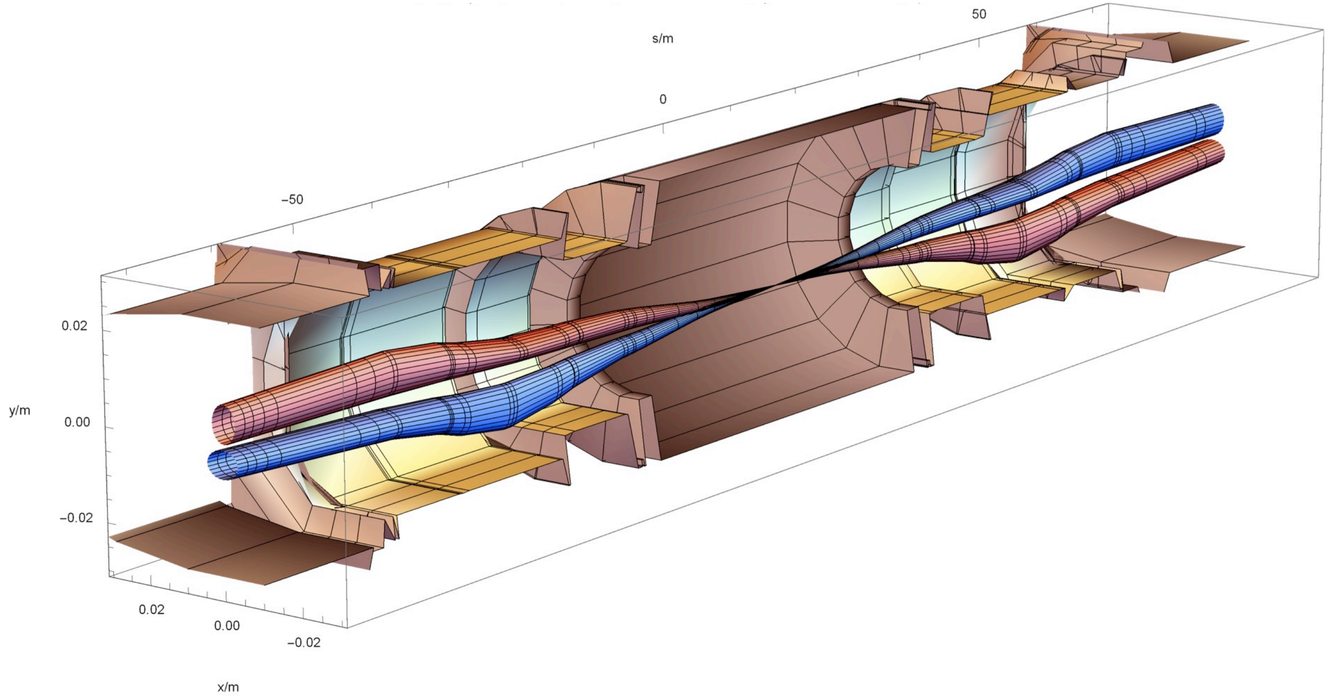
- Need more collisions to observe rare processes

High Luminosity HLC

- ×10 increase in data size
- ×3 collisions per second

How

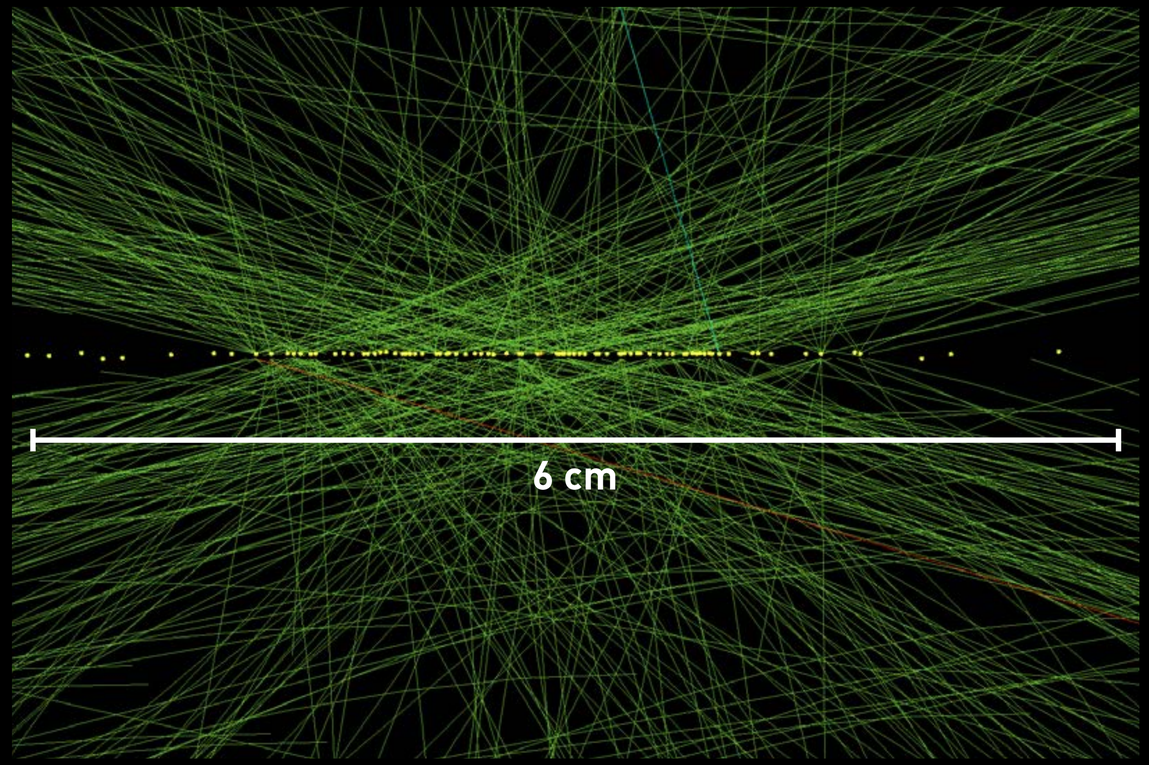
- ×2 protons per bunch
- Squeeze beam at interaction point (β*)



2022	2023	2024	2025	2026	2027	2028	2029	...	2037	2038
LHC				MAJOR UPGRADE			HL-LHC			
Run 3							Run 4+5			

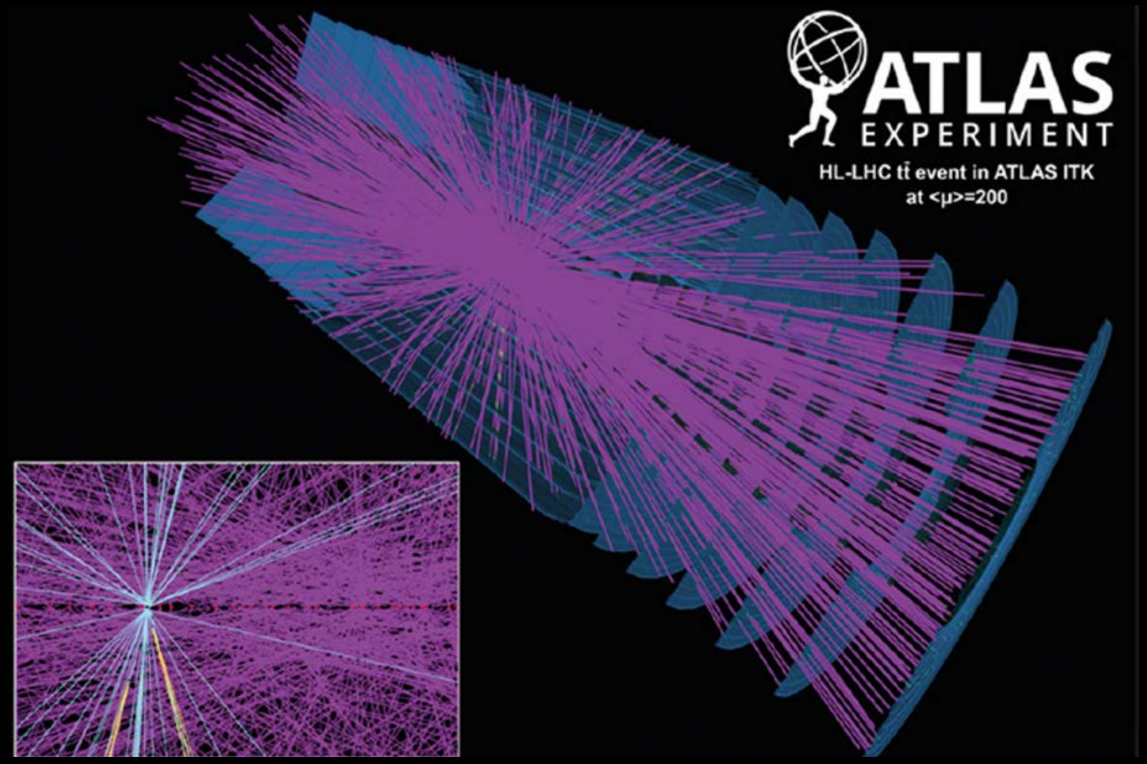
LHC

78 vertices
(average 60)



High Luminosity LHC

200 vertices
(average 140)



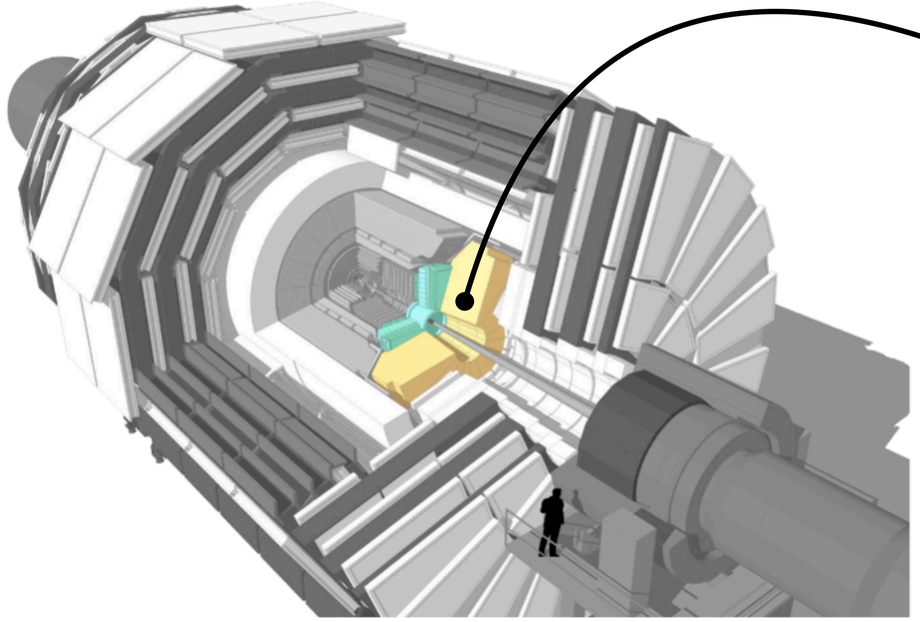
2022	2023	2024	2025	2026	2027	2028	2029	...	2037	2038
LHC				MAJOR UPGRADE			HL-LHC			
Run 3							Run 4+5			

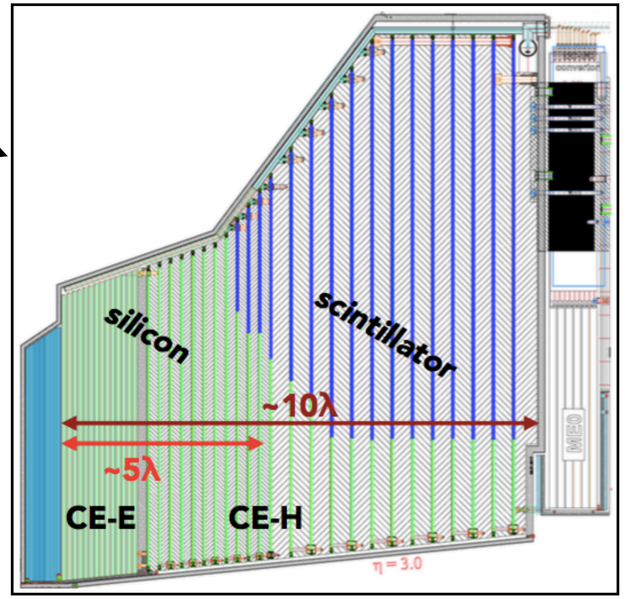
High Luminosity LHC

Must maintain physics acceptance → better detectors

CMS High Granularity (endcap) calorimeter
- 85K (today) → 6M (HL-LHC) readout channels

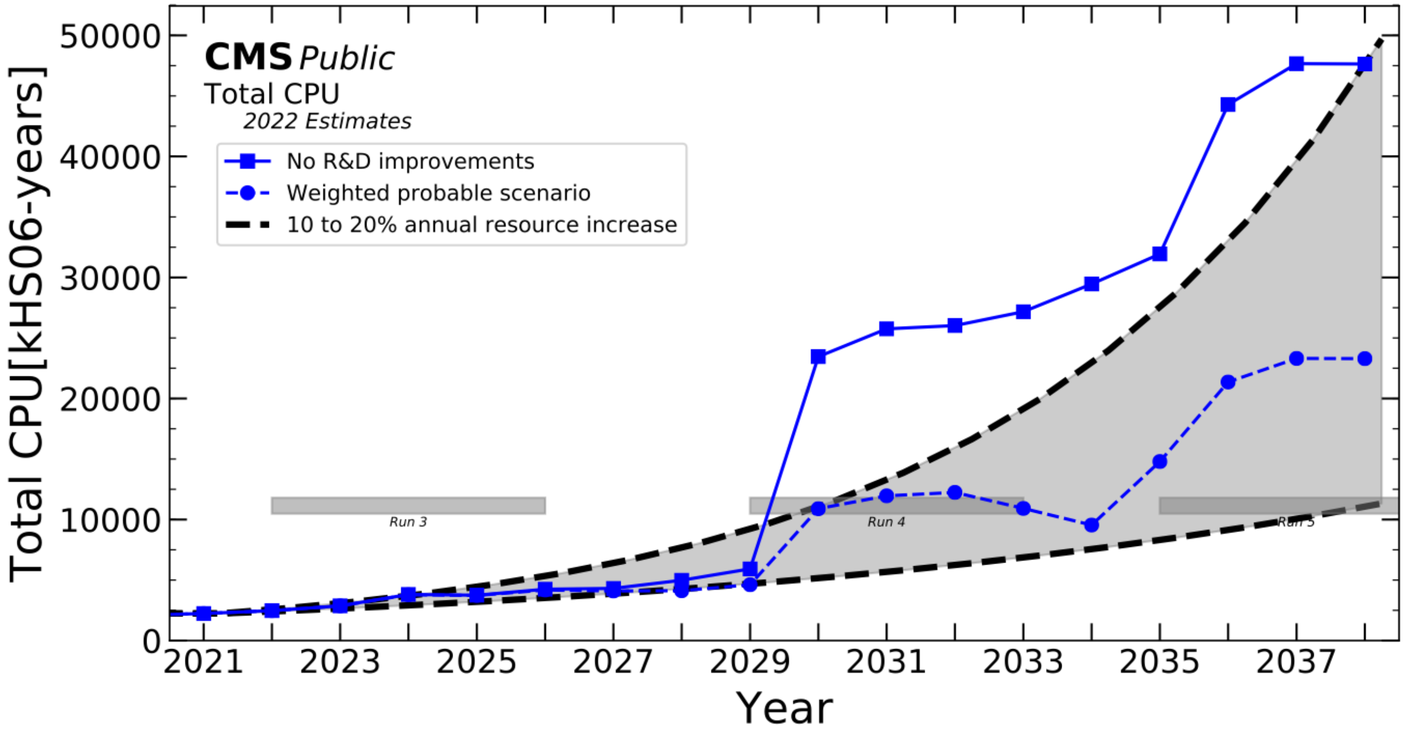
More collisions
More readout channels





CMS HGCAL TDR

High Luminosity LHC



... **flat** computing budget

Need innovation and new techniques to maintain physics reach
while staying within throughout requirements!

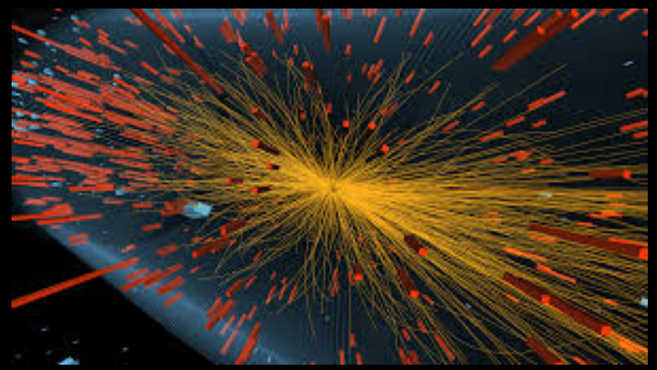
Todays algorithms will not be sustainable in HL-LHC!

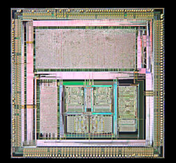
→ Utilise modern Machine Learning to become

faster
better
and do more

Fast ML on ASIC

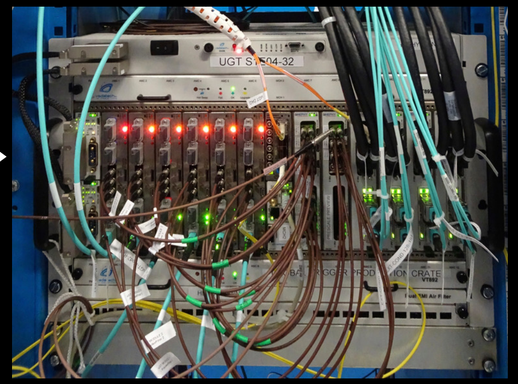
- Better, smaller data





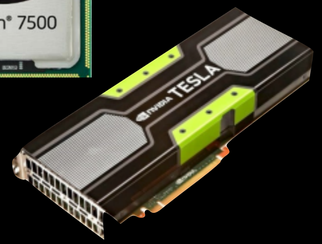
Fast ML on FPGA

- Faster and better decisions



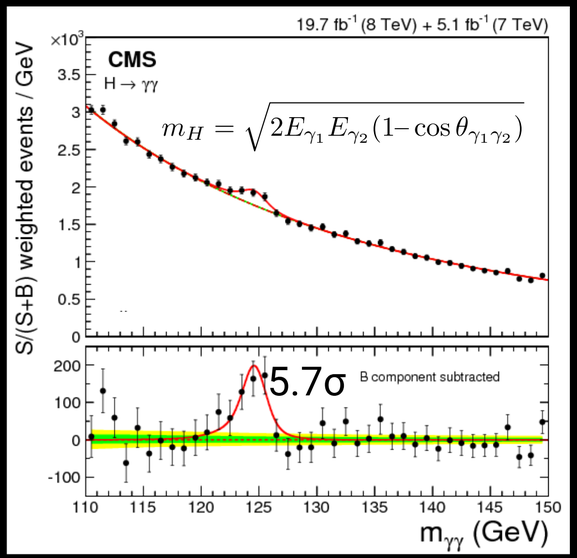
Fast ML on GPU

- Faster and better decisions

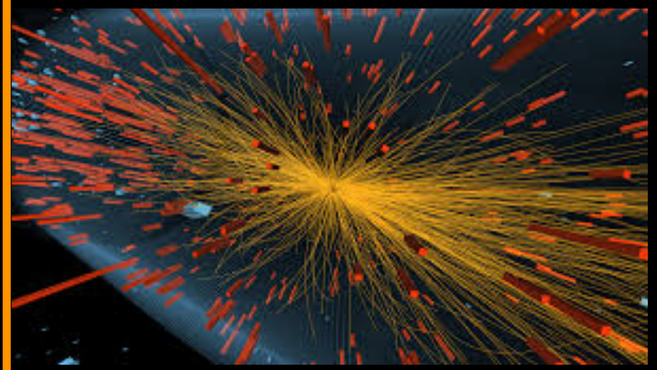


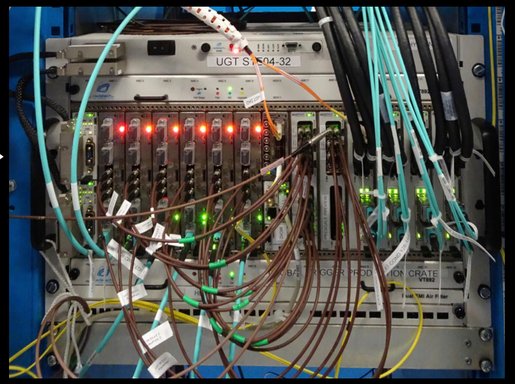
ML offline

- Improve analysis sensitivity

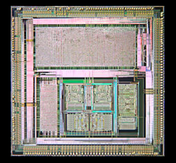


Nanosecond inference on specialised hardware

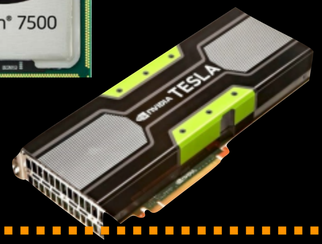


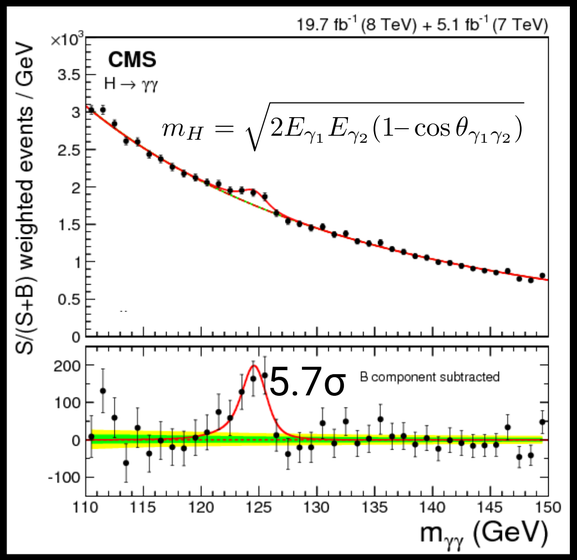


FPGA inference



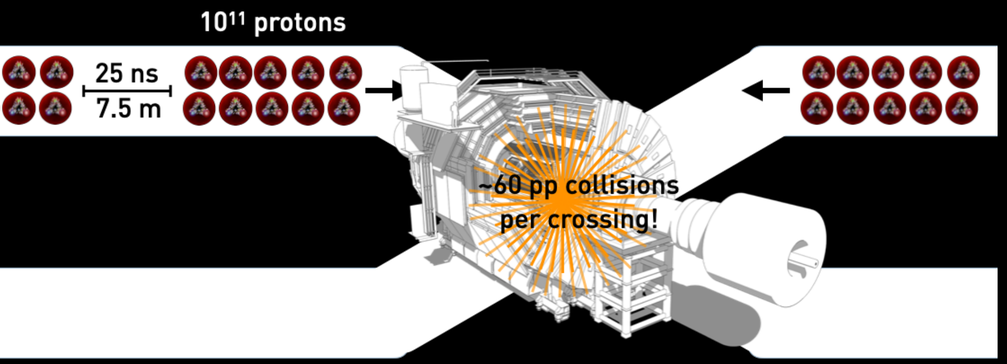
ASIC inference





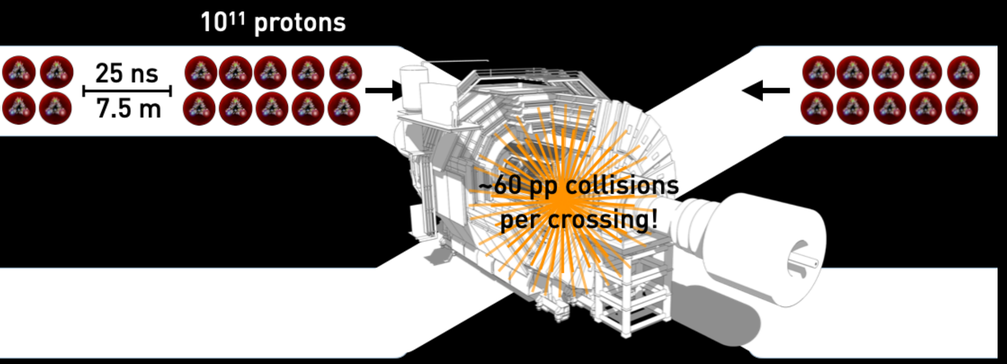
Low latency

- Strictly limited by collisions occurring every 25 ns



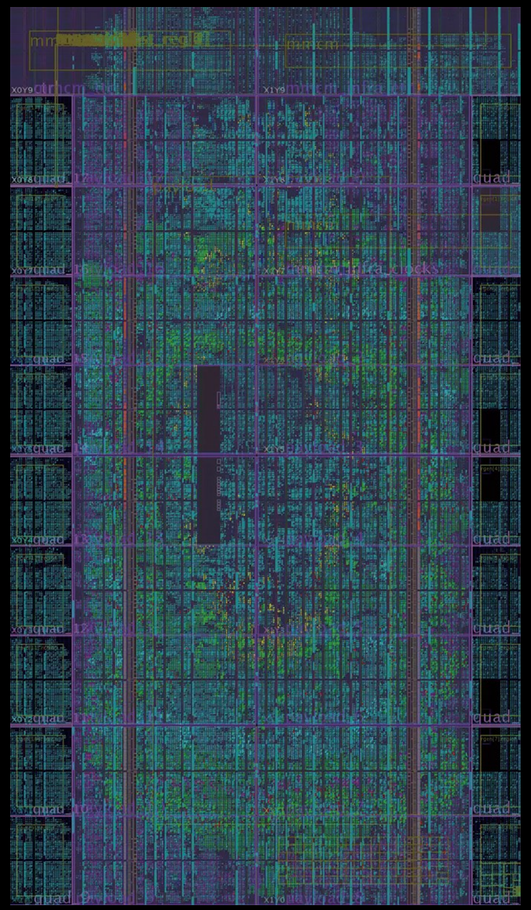
Low latency

- Strictly limited by collisions occurring every 25 ns



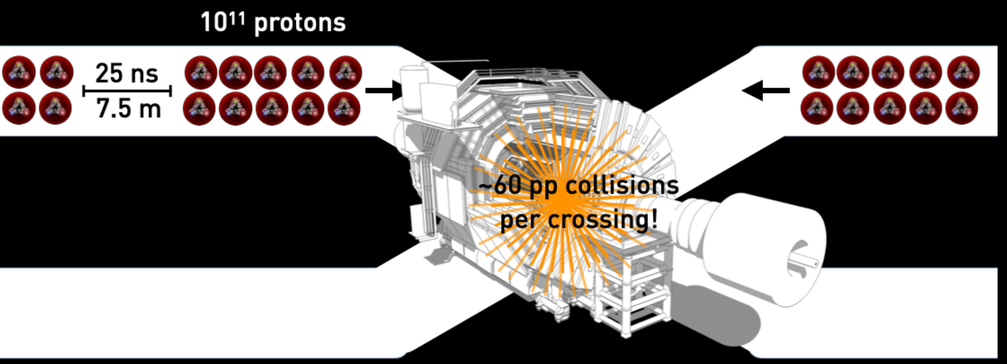
Low resource usage

- Several algorithms in parallel on single device



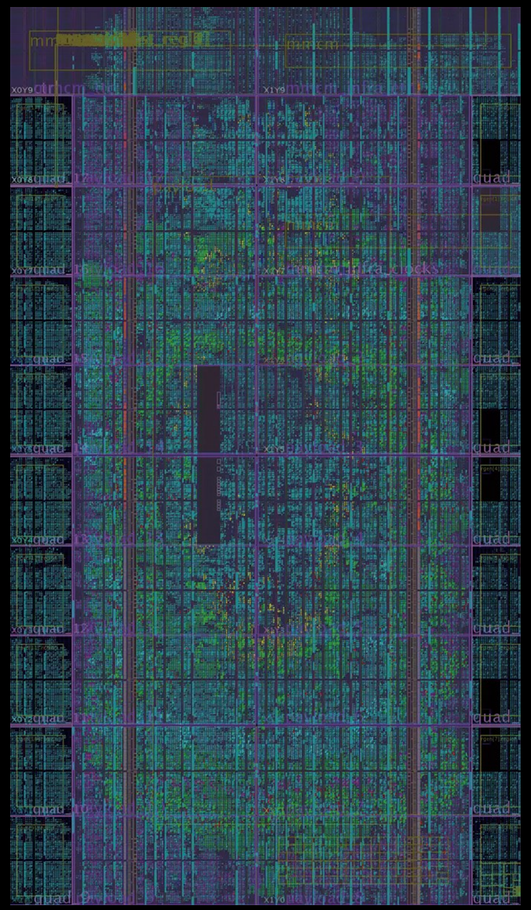
Low latency

- Strictly limited by collisions occurring every 25 ns



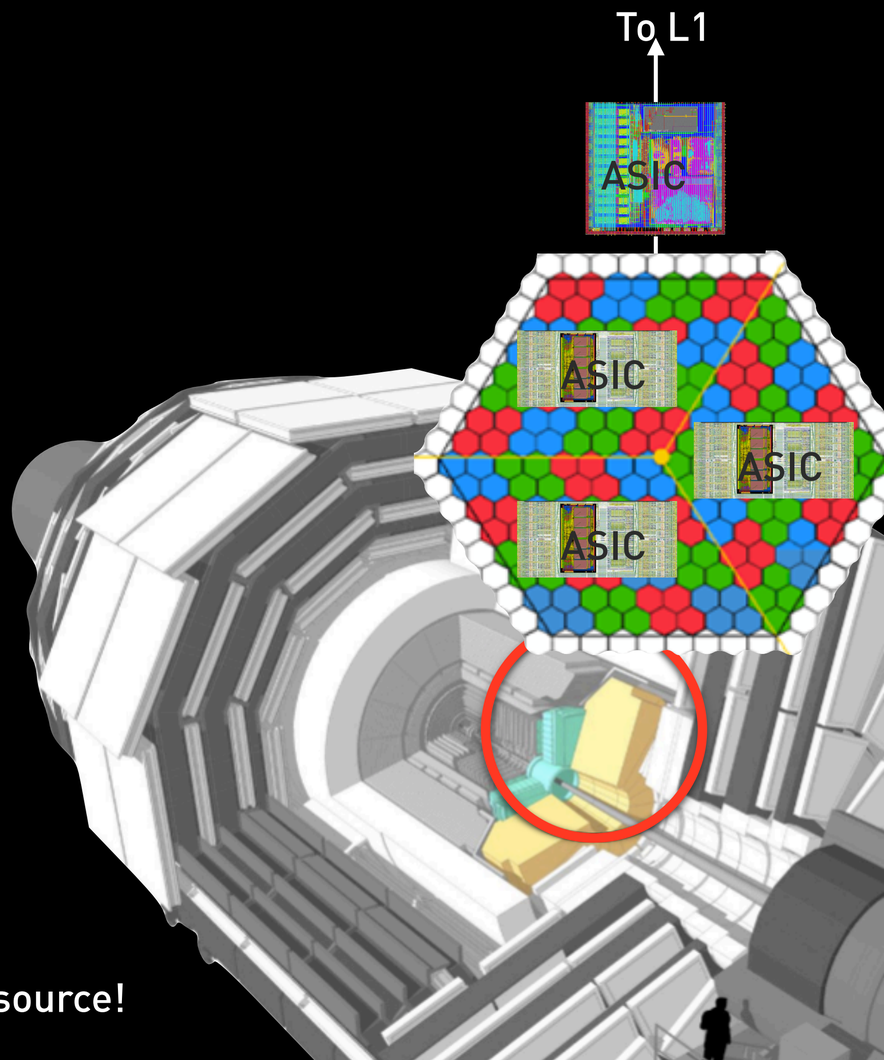
Low resource usage

- Several algorithms in parallel on single device

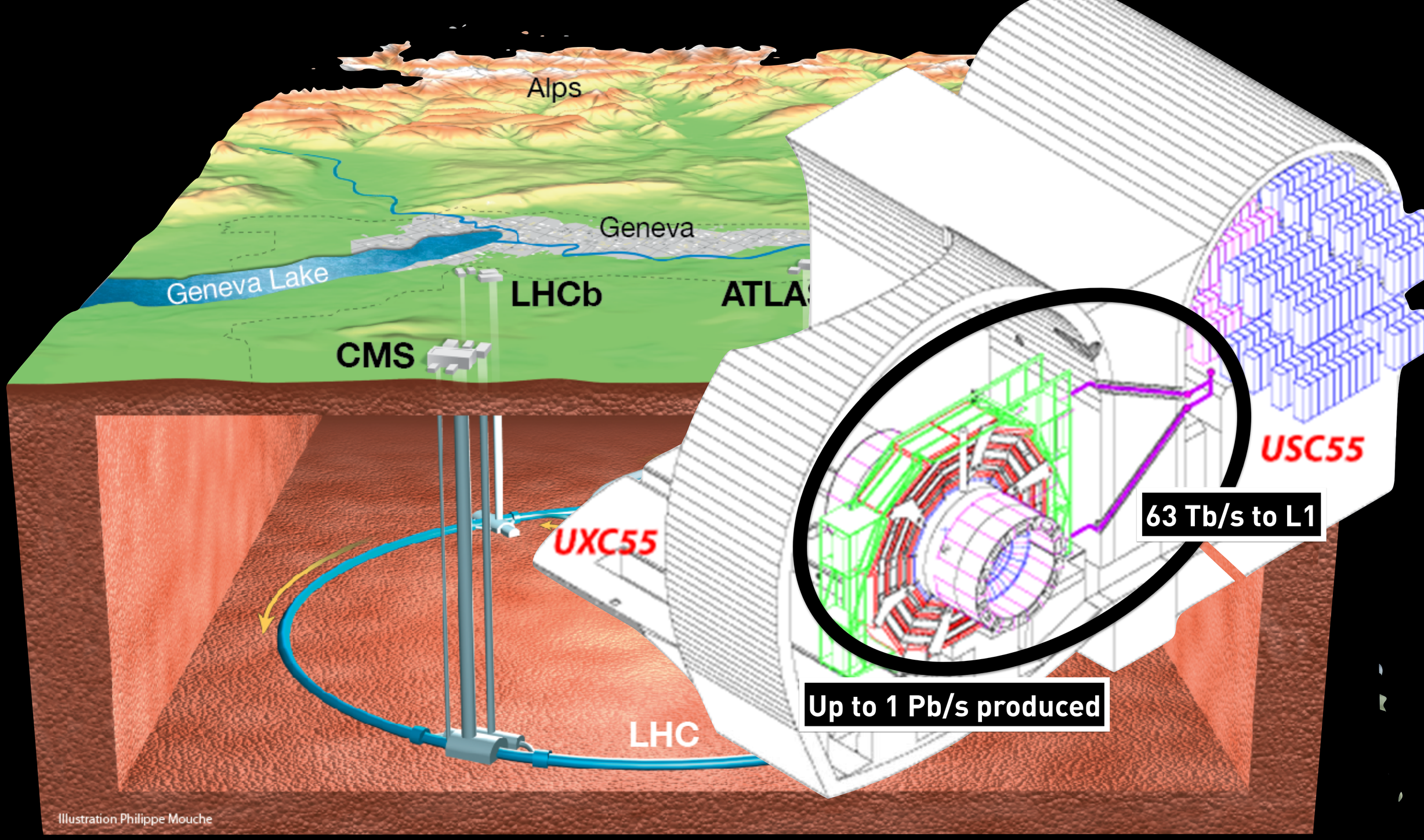


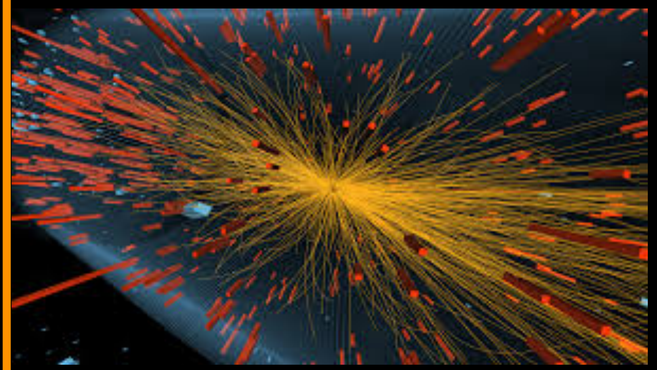
Power efficient

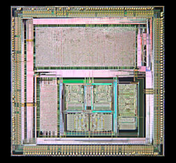
- On detector: limited to mW
- L1: Cooling key challenge



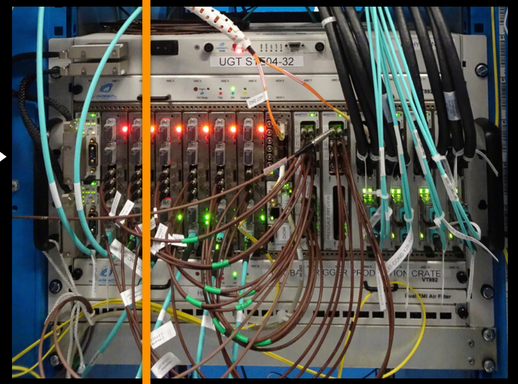
Extreme combination of low power, low latency, low resource!

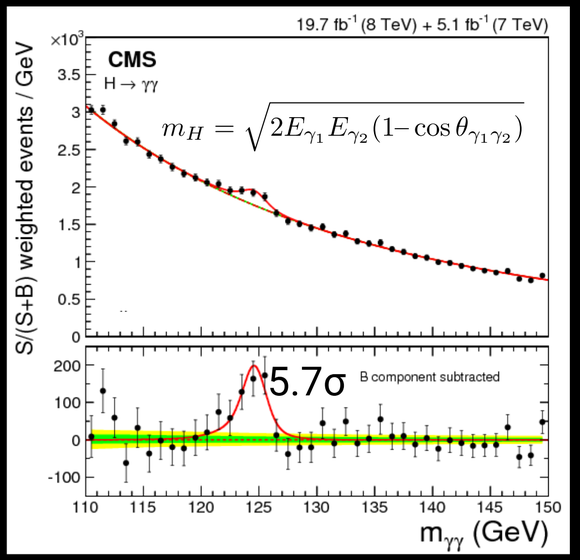

Alps
Geneva
Geneva Lake
LHCb
ATLAS
CMS
UXC55
USC55
63 Tb/s to L1
Up to 1 Pb/s produced
LHC
Illustration Philippe Mouche





ASIC inference

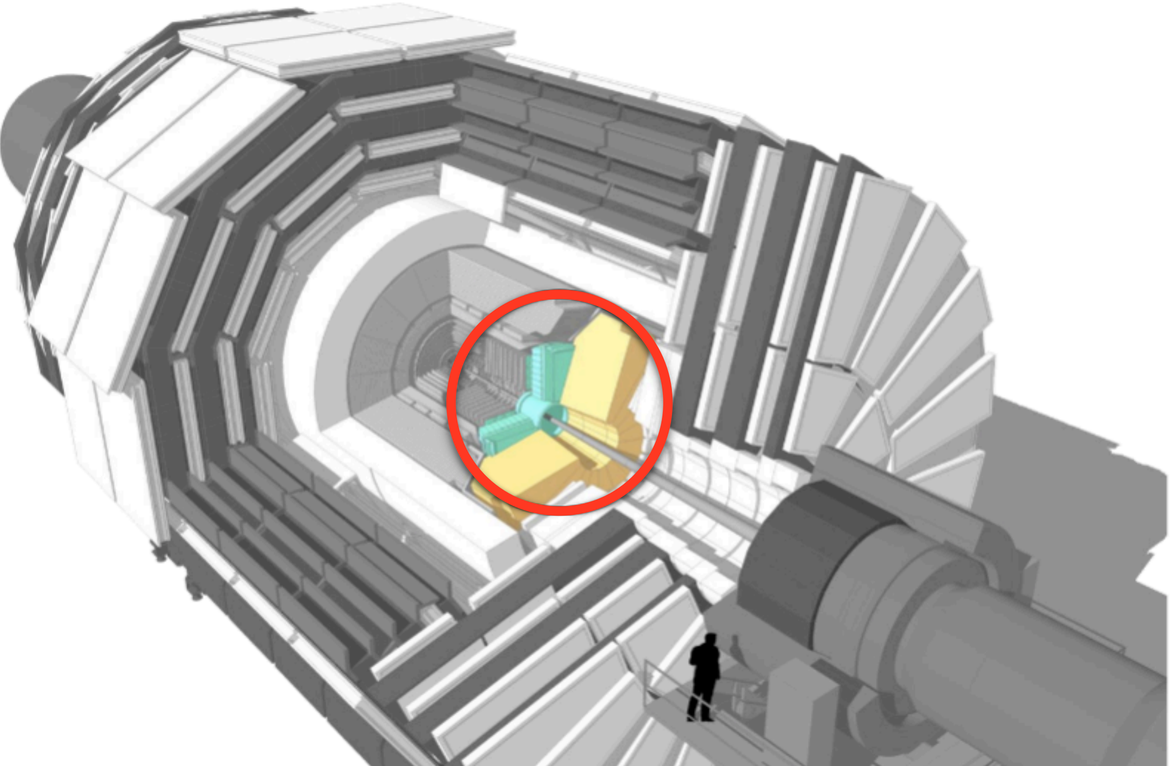


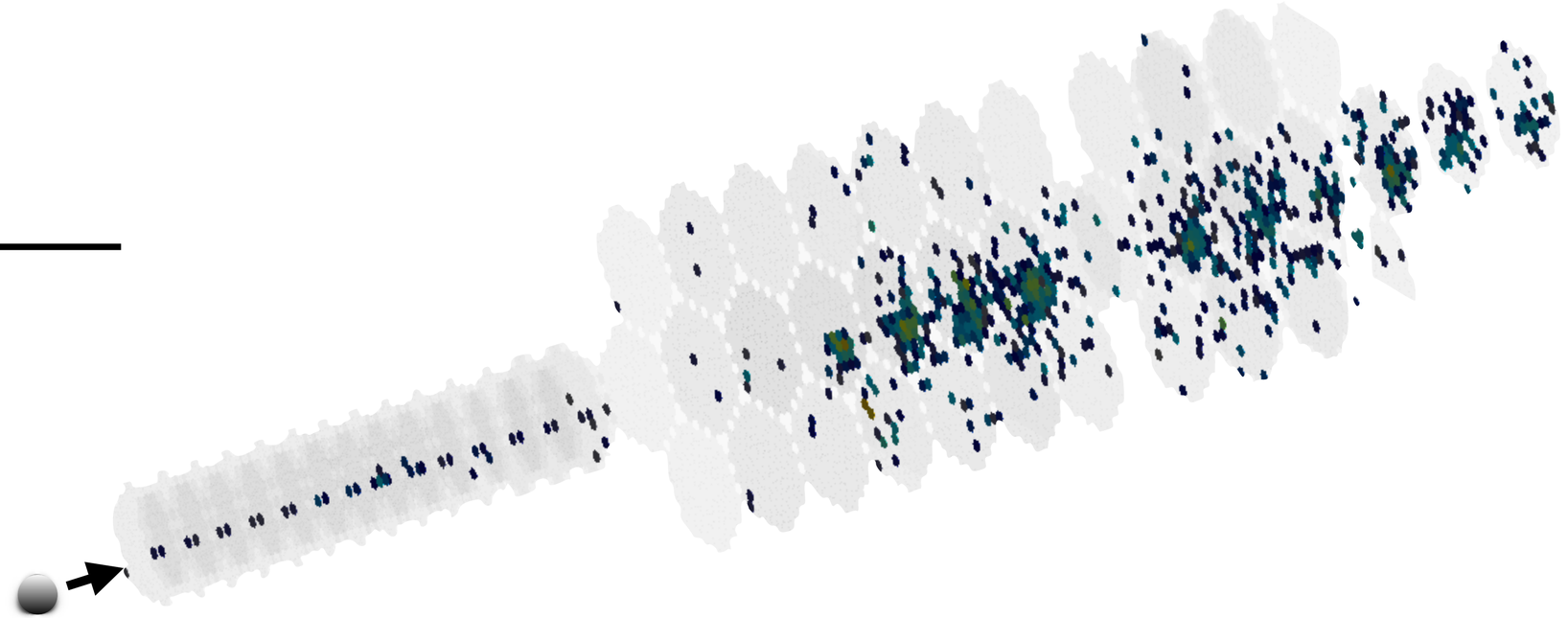


ML for compression

CMS High Granularity calorimeter

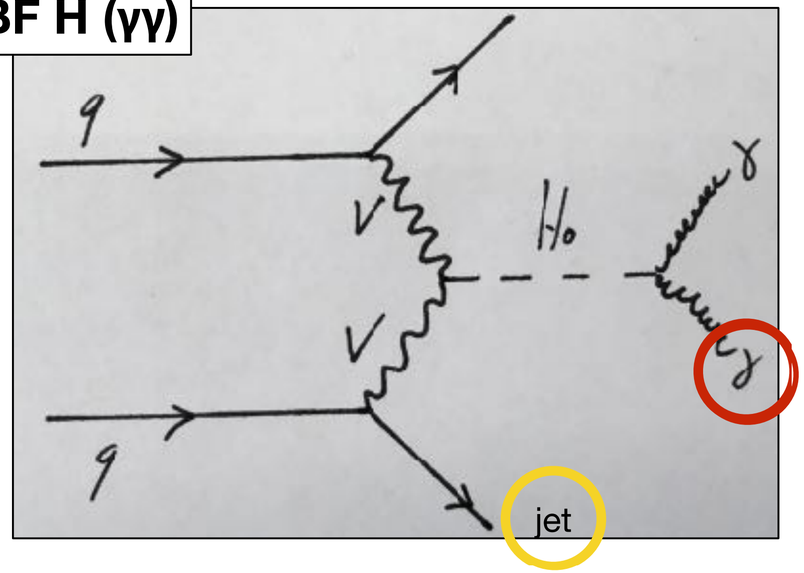
- 6.5 million readout channels, 50 layers



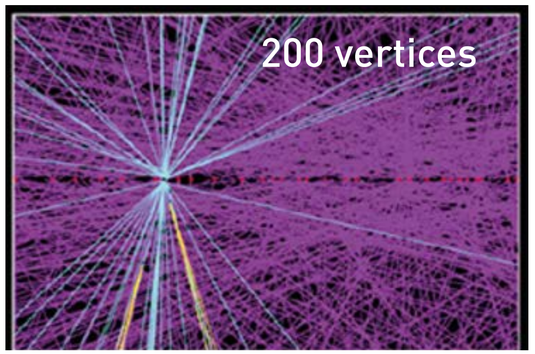


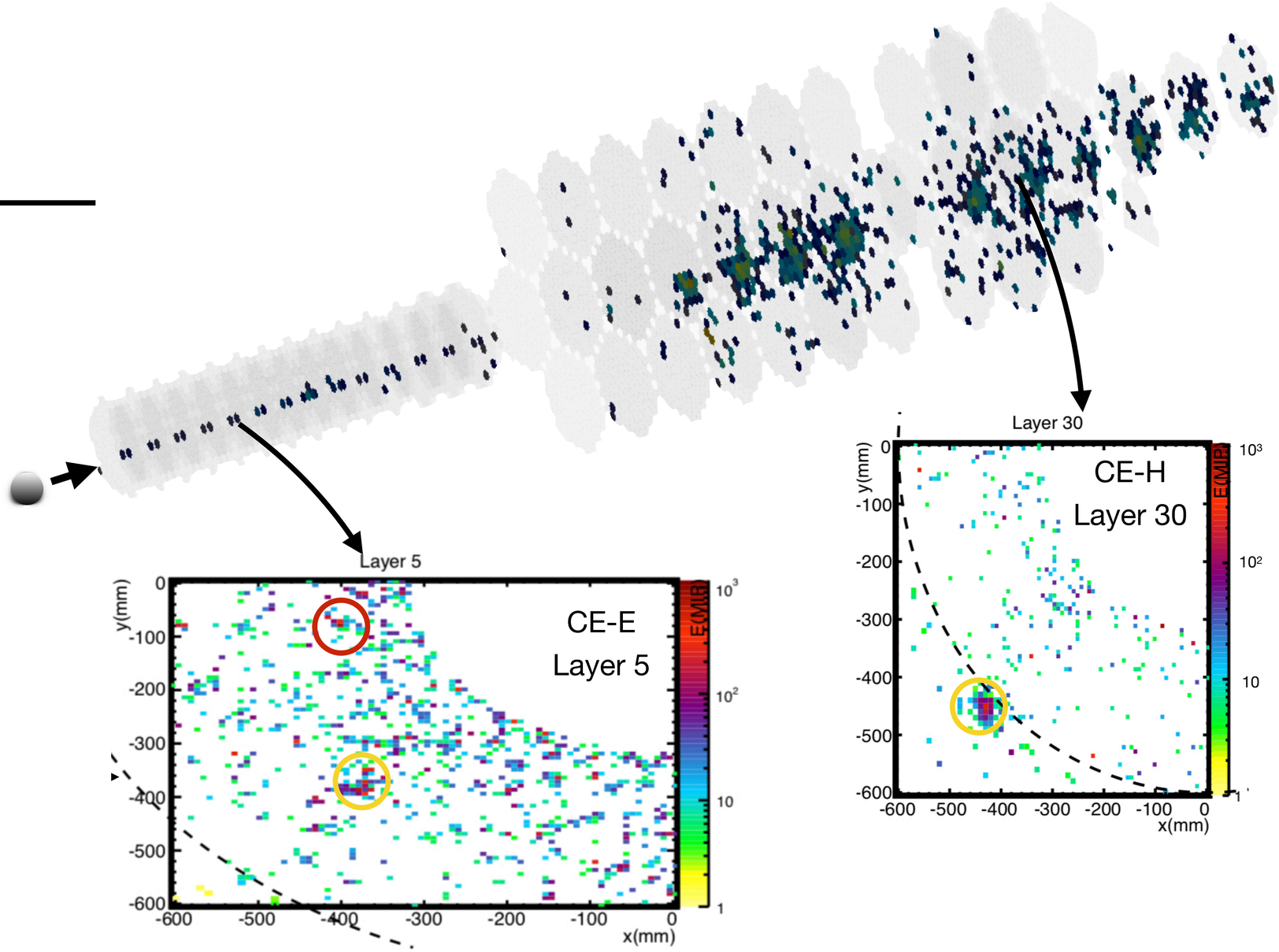
ML for compression

VBF H (γγ)

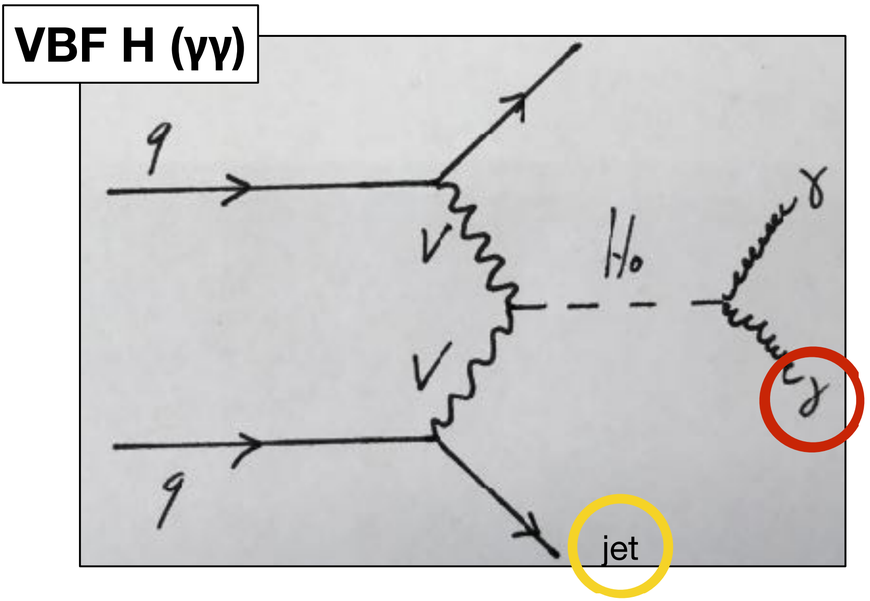


+

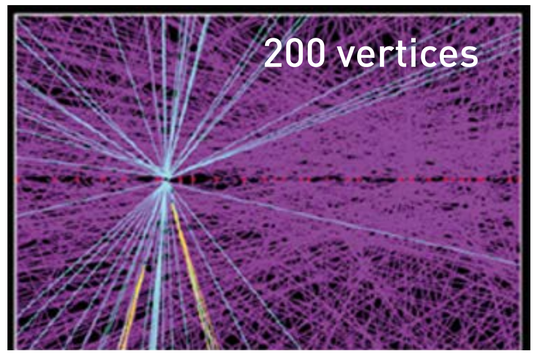


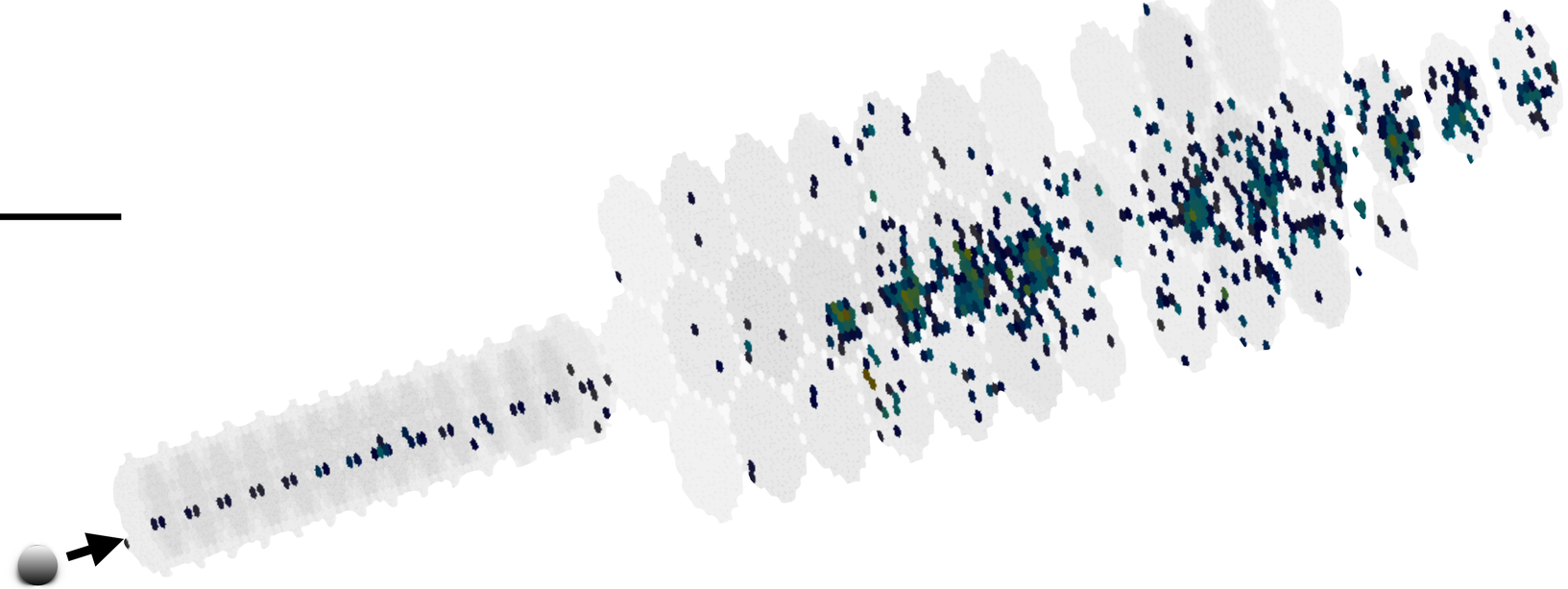


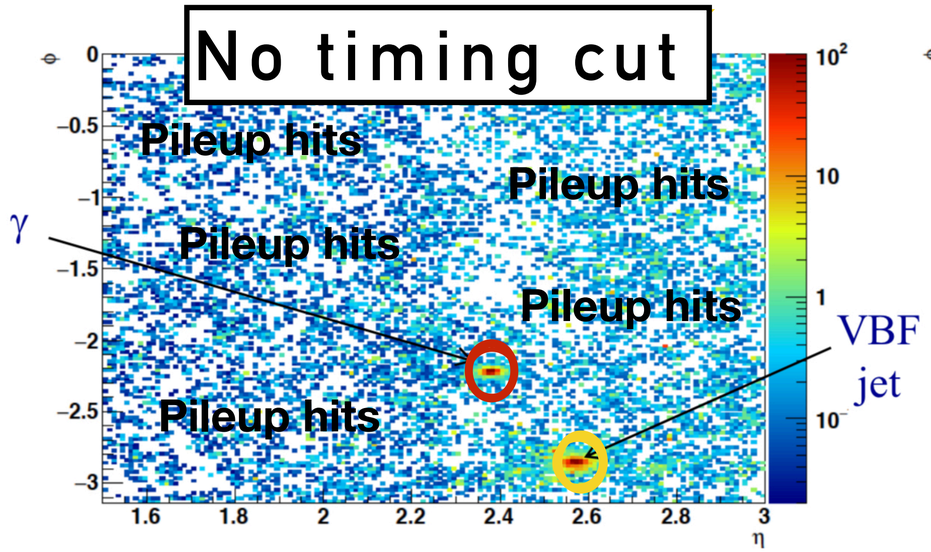
ML for compression

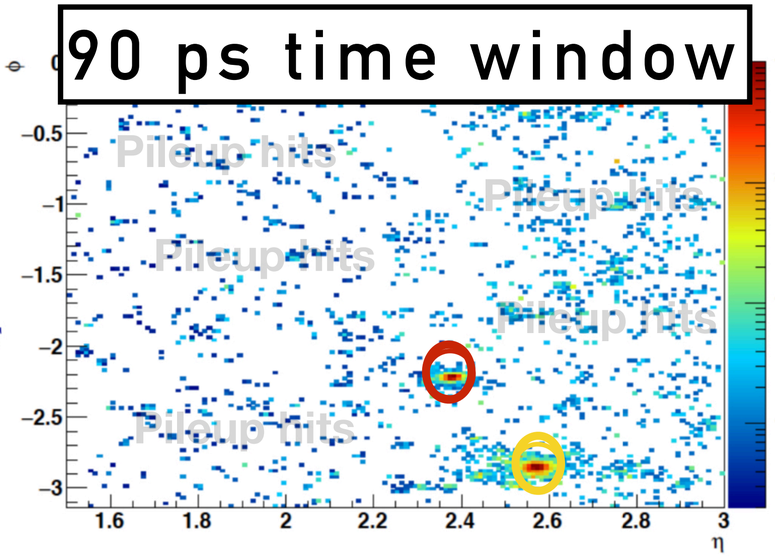


+

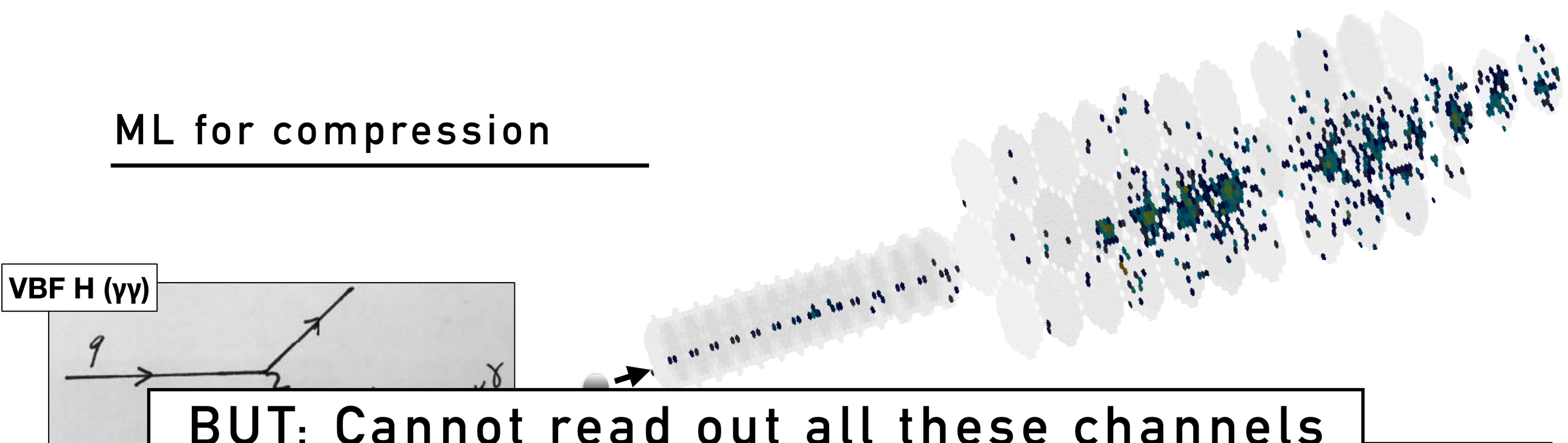




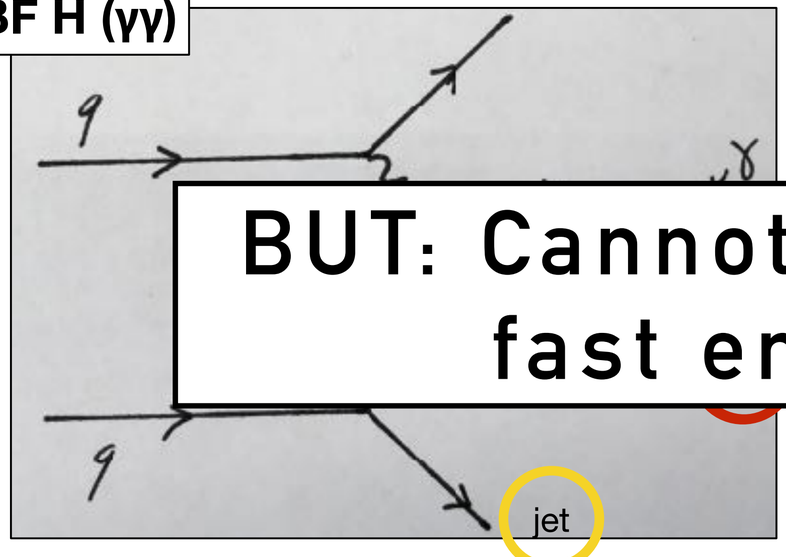




ML for compression



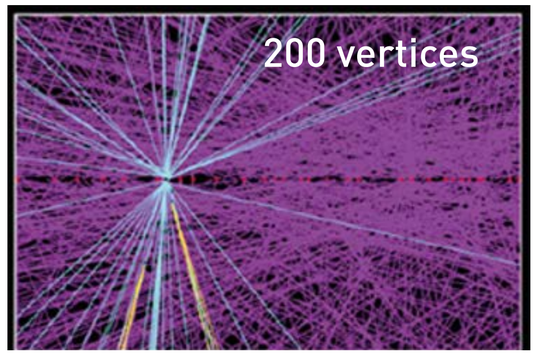
VBF H (γγ)

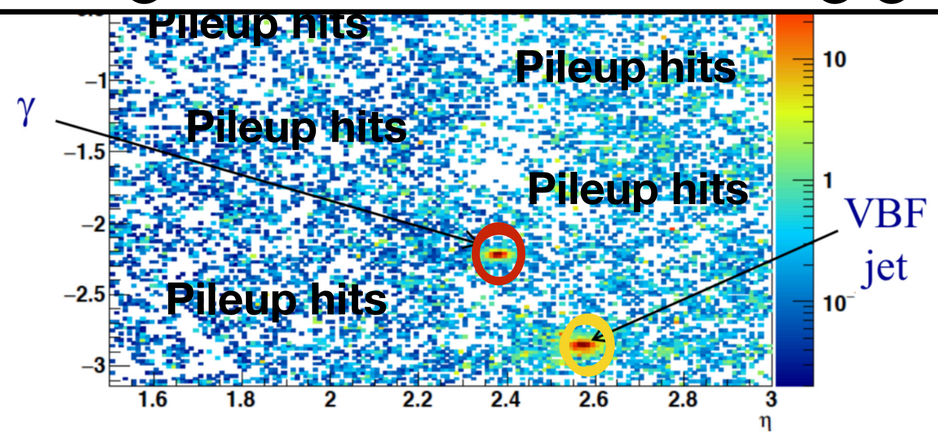


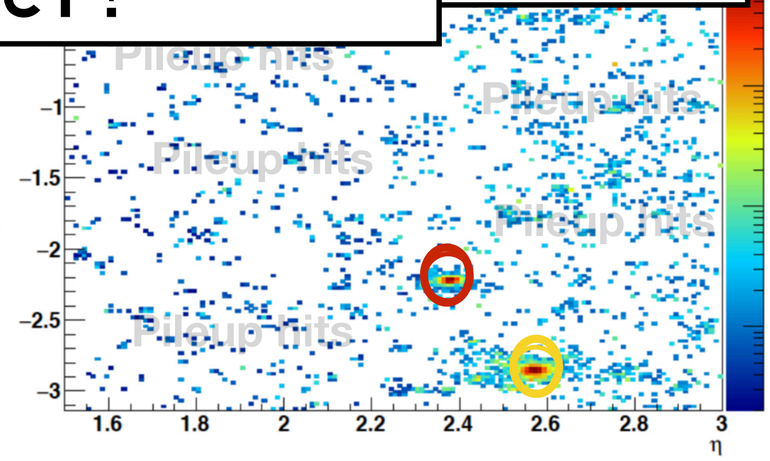
BUT: Cannot read out all these channels fast enough for L1 to trigger!

window

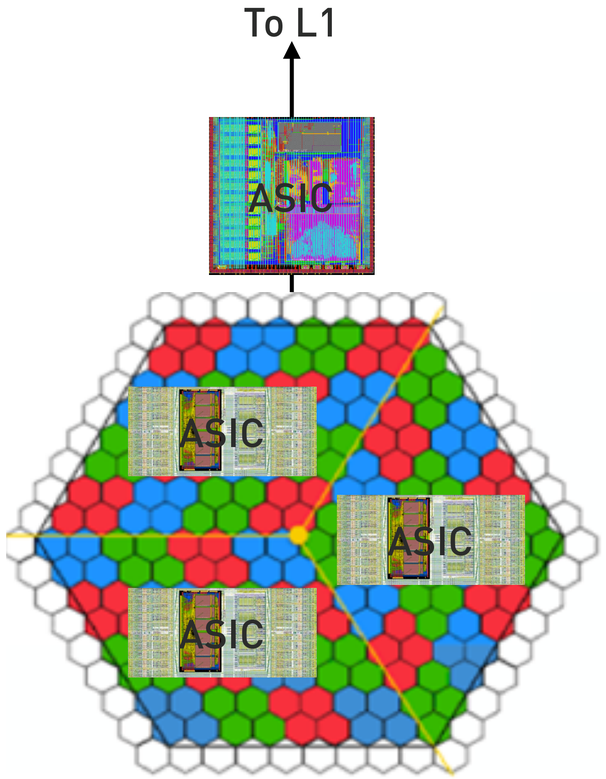
+

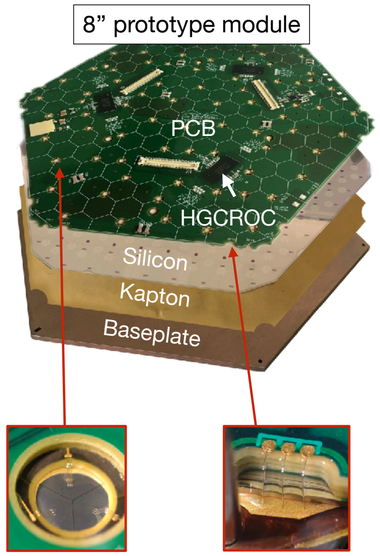






ML for compression

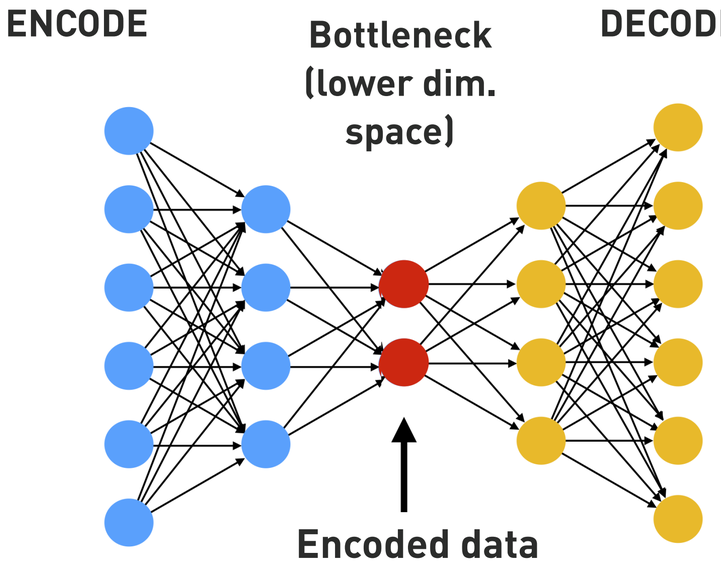




Must compress ON DETECTOR

- High radiation
- Cooled to -30 → low power
- 1.5 μs latency

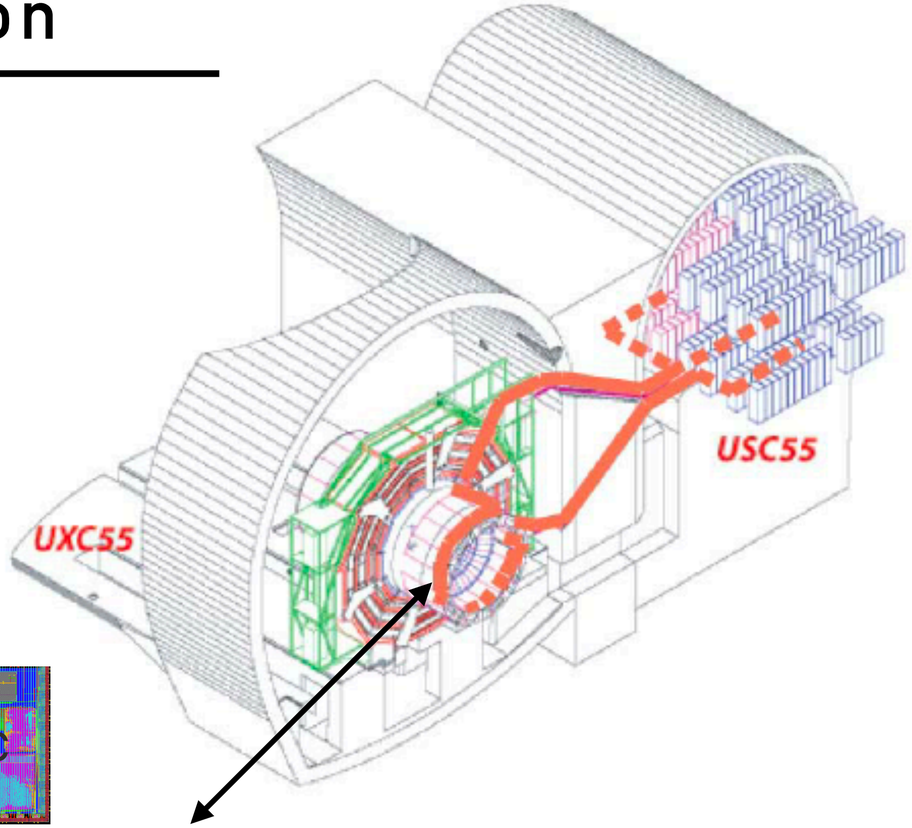
ML for compression

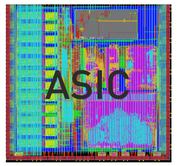


Variational Autoencoder

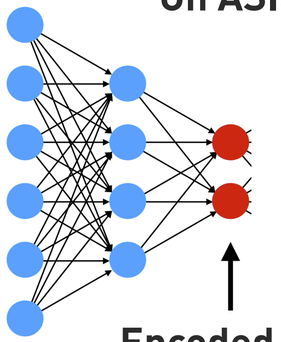
See more here

ML for compression

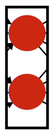




On ASIC



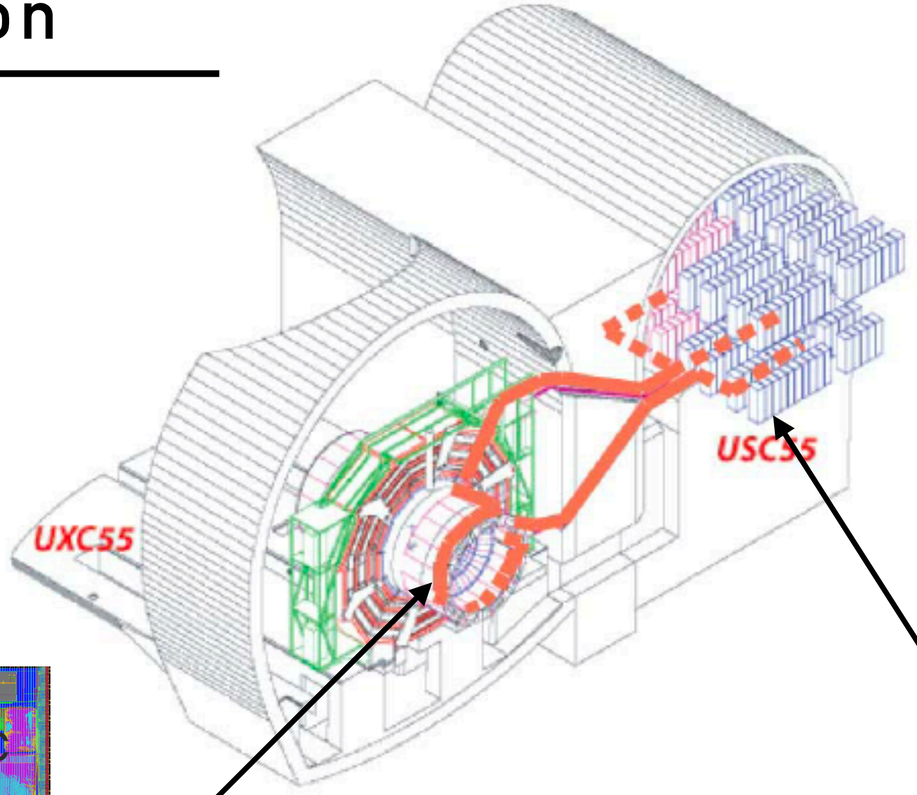
Encoded data

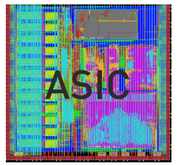


Transmit encoded data!

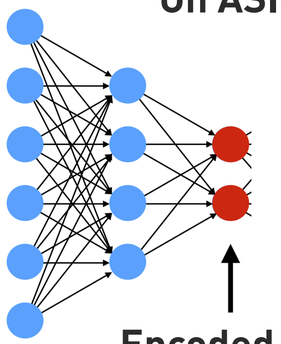
See more here

ML for compression

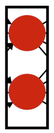




On ASIC

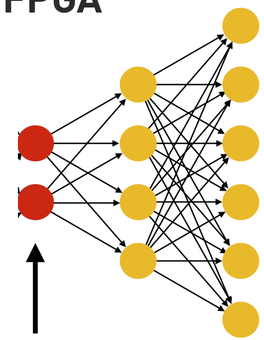


Encoded data

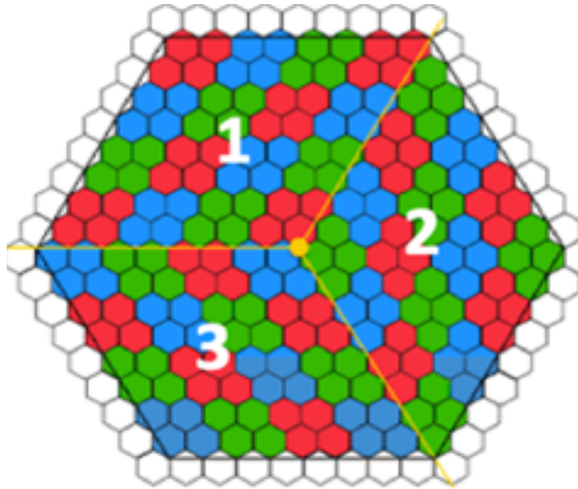


Transmit encoded data!

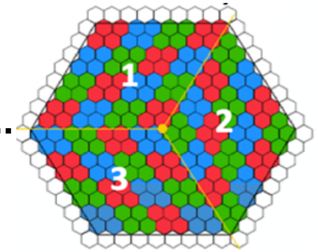
On FPGA



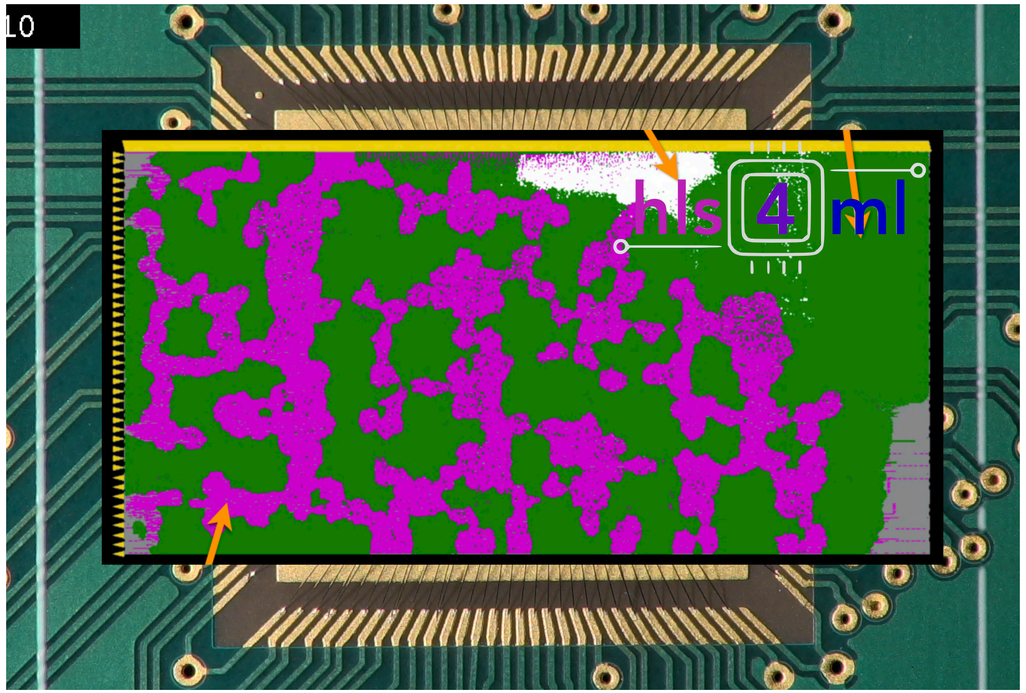
Encoded data

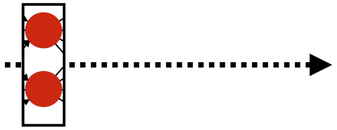


AI for compression

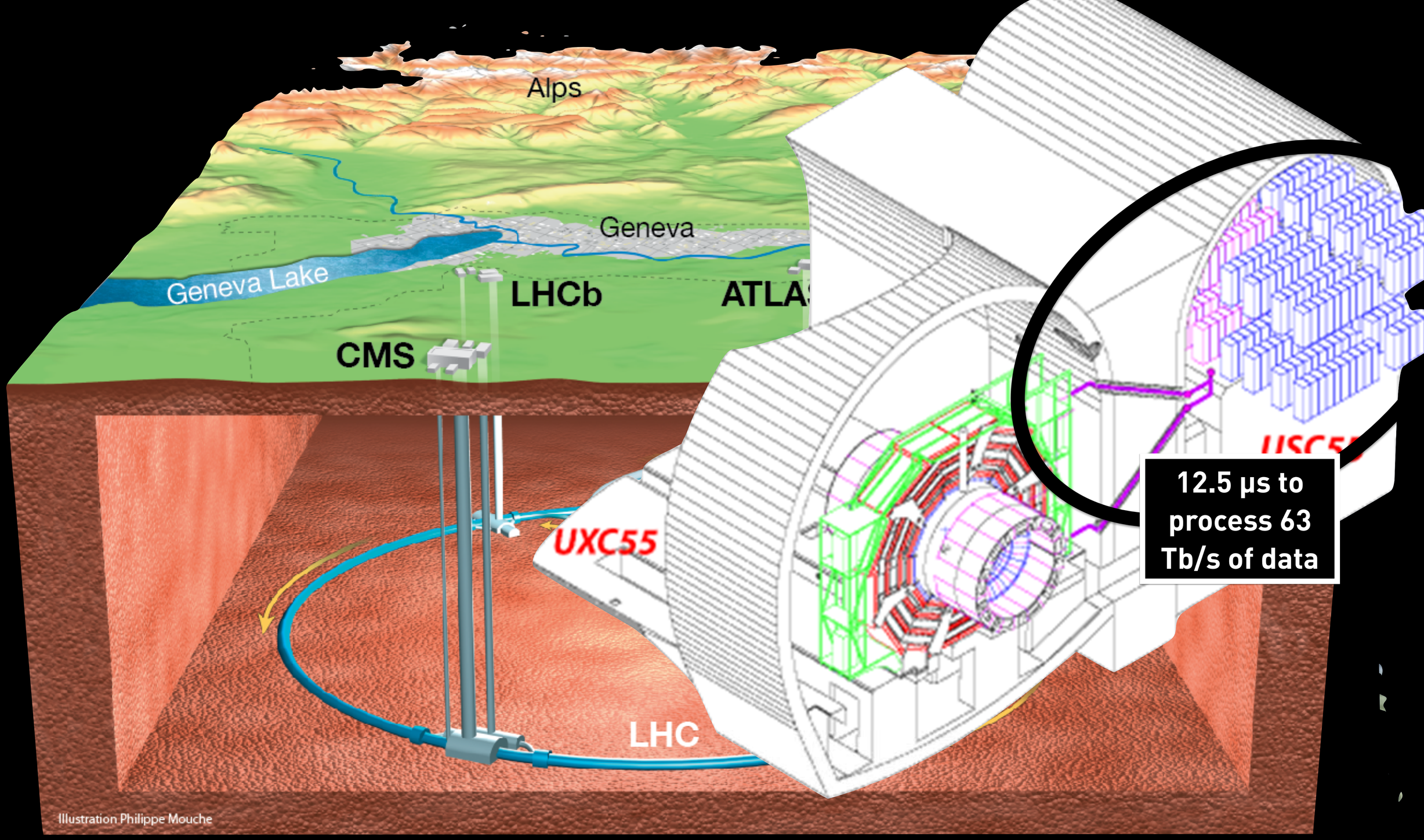


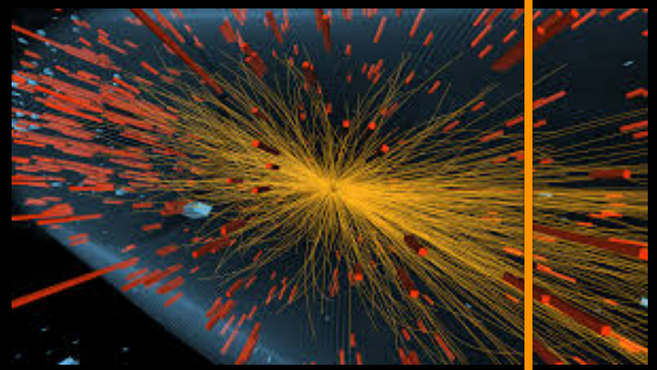
48 trigger cells

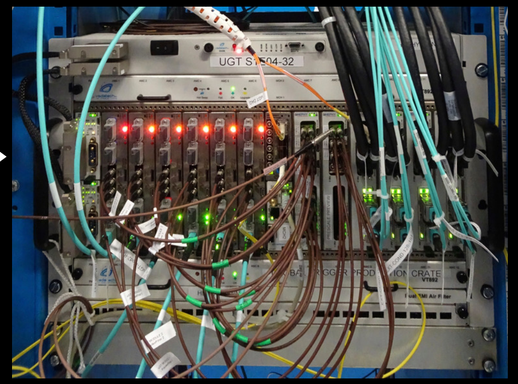


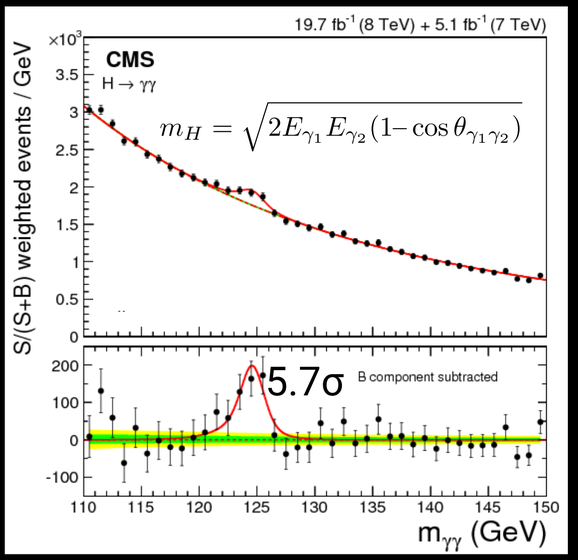


16 ReLU activated nodes


Alps
Geneva
Geneva Lake
LHCb
ATLAS
CMS
UXC55
LHC
12.5 µs to process 63 Tb/s of data
Illustration Philippe Mouche

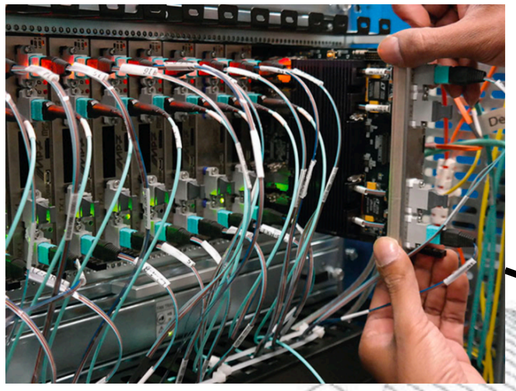


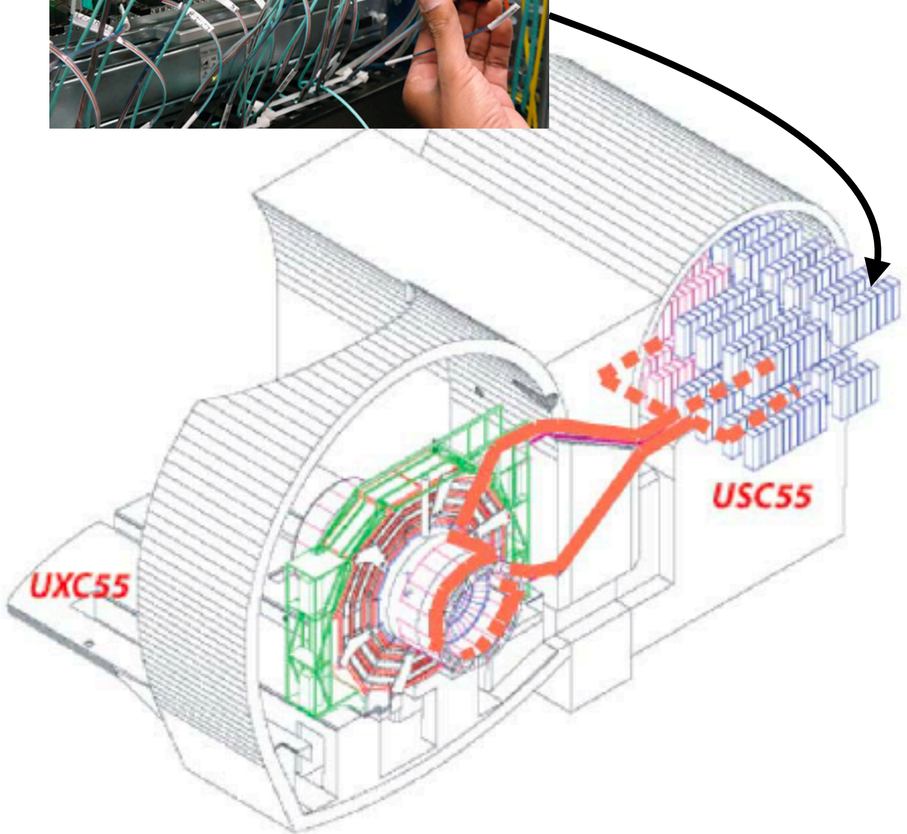


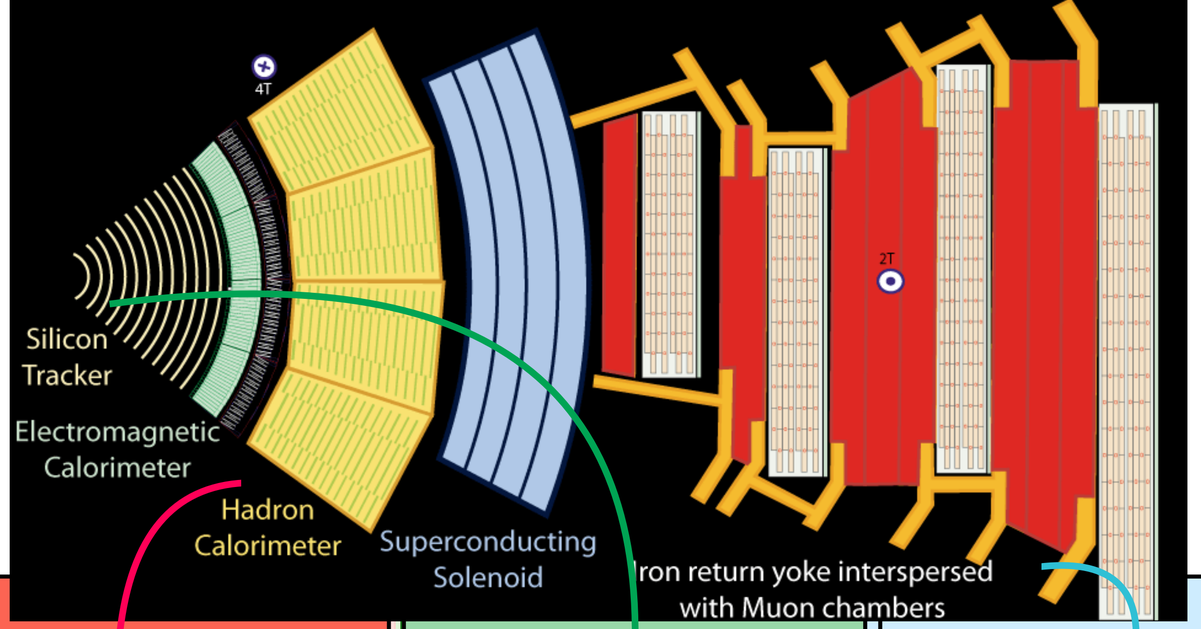


FPGA inference

The level-1 trigger







CALORIMETRY: 370 FPGAs

*54 for HGCAL only!

TRACK FINDER: 174 FPGAs

New for HL-LHC!

MUONS: 96 FPGAs

5 µs

COMBINE: 66 FPGAs

GLOBAL TRIGGER: 24 FPGAs

EXTERNAL TRIGGERS

CALORIMETRY

PARTICLE FLOW

MUONS

Trigger accept/reject

12.5 µs

The level-1 trigger

To cope with increased data complexity, new at L1:

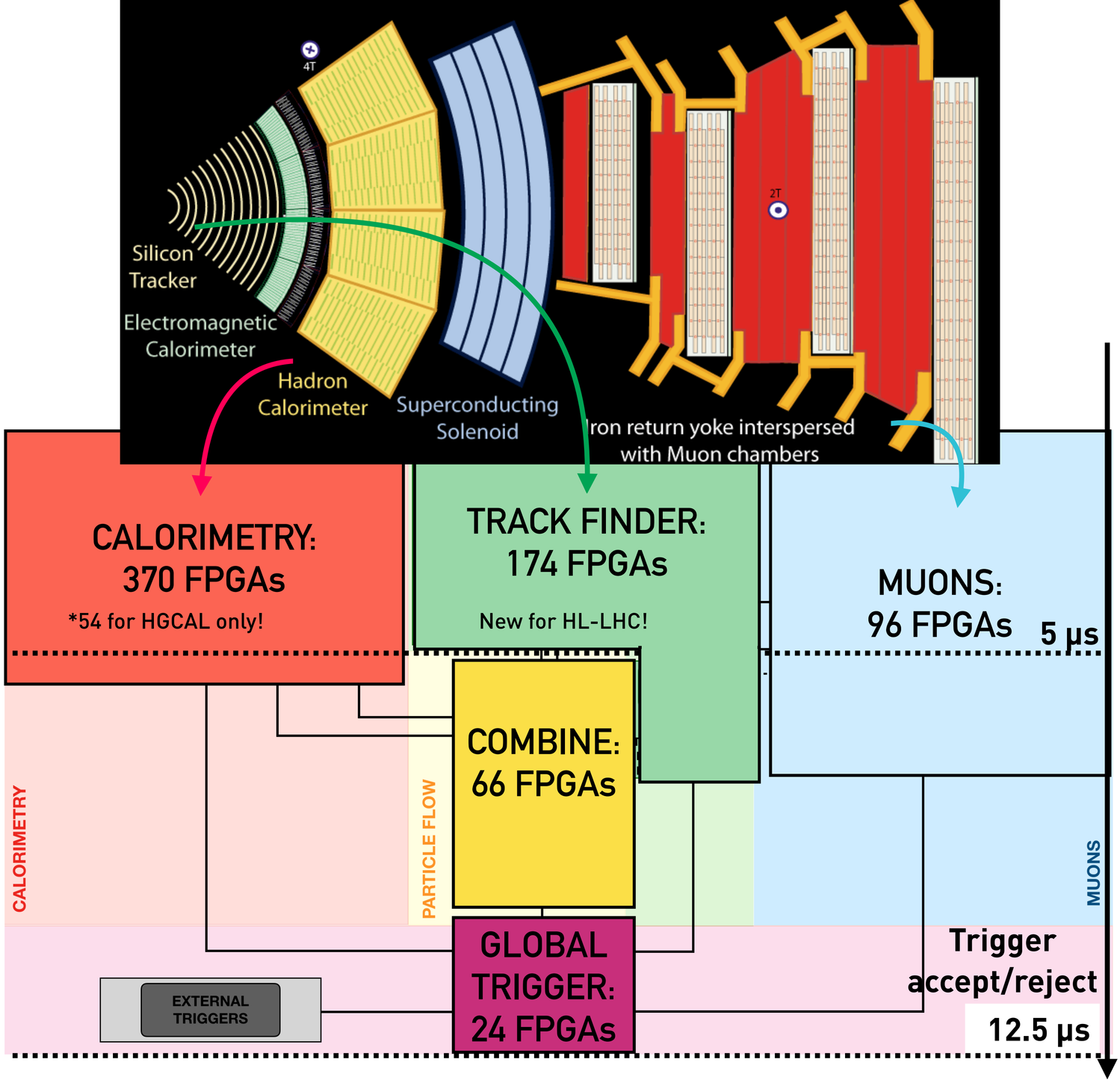
- Tracking
- Particle Flow
- O(1)M channel HGCal

Input data

- 2 Tb/s → 63 Tb/s

Latency

- 4 μs → 12 μs



Why FPGAs at LHC?

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
  (for Phase-2, FPGAs dissipate heat of ~7W/cm2 while processing 5% of total internet traffic!)

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
  (for Phase-2, FPGAs dissipate heat of ~7W/cm2 while processing 5% of total internet traffic!)

Latency deterministic

- CPU/GPU has processing randomness, FPGAs **repeatable and predictable latency**

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
  (for Phase-2, FPGAs dissipate heat of ~7W/cm2 while processing 5% of total internet traffic!)

Latency deterministic

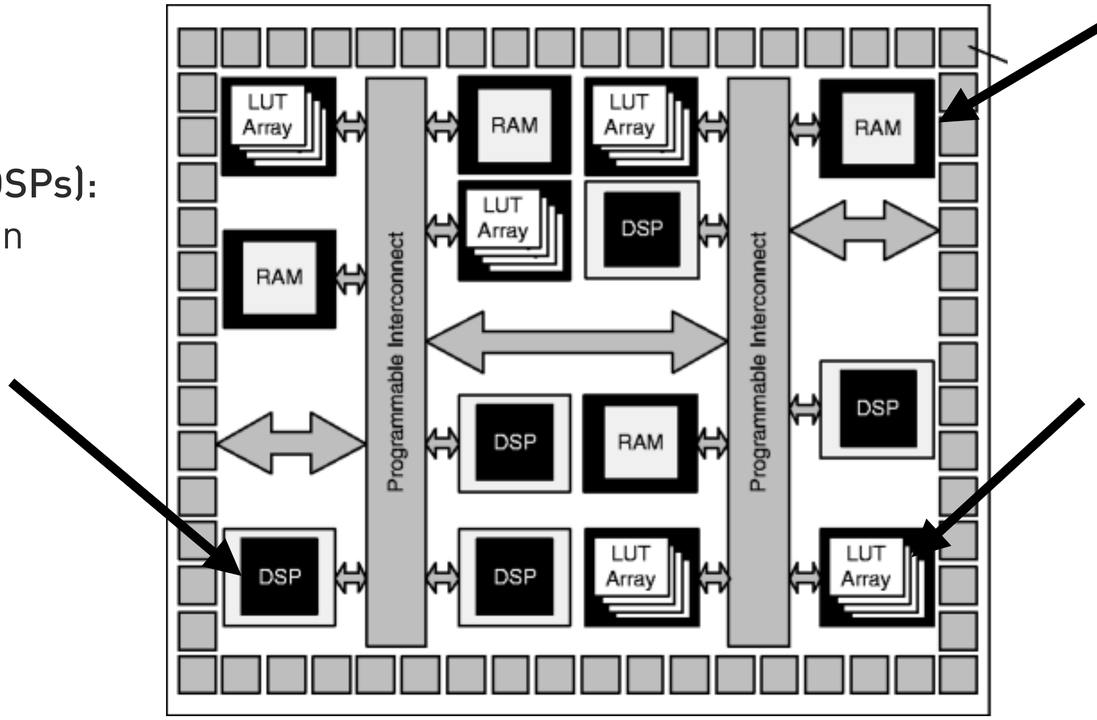
- CPU/GPU has processing randomness, FPGAs **repeatable and predictable latency**

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

What are FPGAs?

What are FPGAs?

Digital signal processors (DSPs):
specialised for multiplication

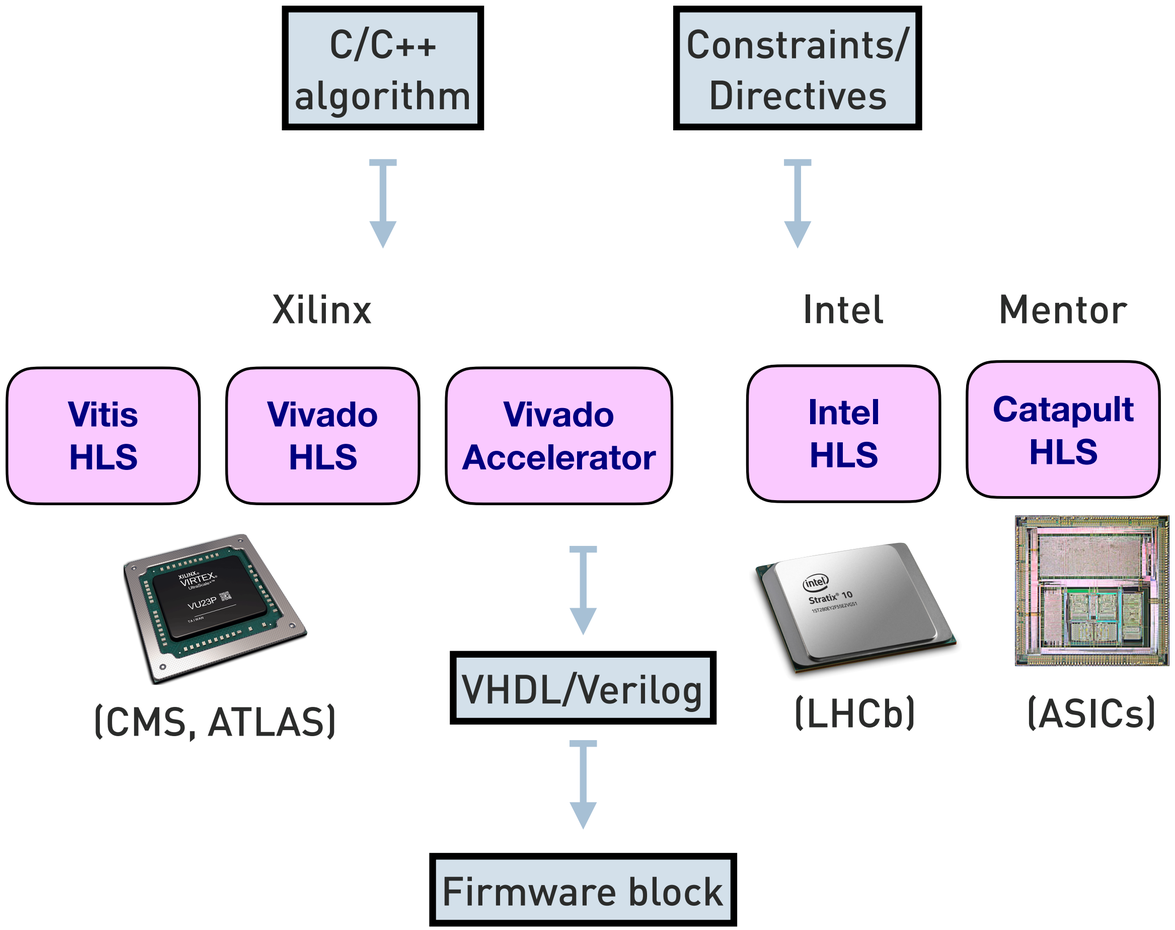


Memory (BRAM)

Logic cells/lookup tables (LUTs):
perform arbitrary functions

flip-flops (FF):
registers data in time with clock pulse

Programming an FPGA



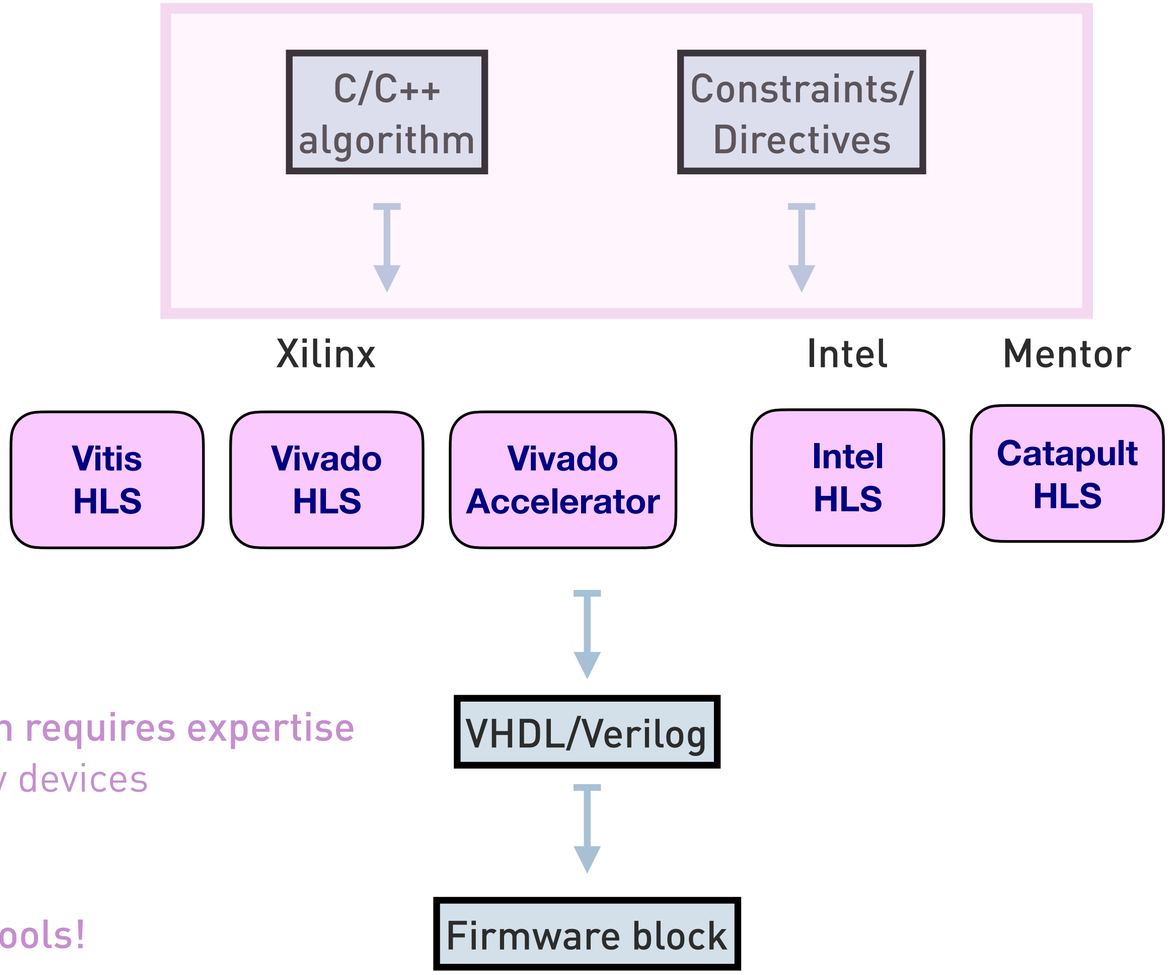
1. Write C-style code of function

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

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

4. Build firmware

Programming 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!

FPGA trigger code

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

use work.gtl_pkg.all;

entity invariant_mass is
    generic (
        upper_limit: real := 15.0;
        lower_limit: real := 10.0;
        pt1_width: positive := 12;
        pt2_width: positive := 12;
        cosh_cos_width: positive := 28;
        INV_MASS_PRECISION : positive := 1;
        INV_MASS_COSH_COS_PRECISION : positive := 3
    );
    port(
        pt1 : in std_logic_vector(pt1_width-1 downto 0);
        pt2 : in std_logic_vector(pt2_width-1 downto 0);
        cosh_deta : in std_logic_vector(cosh_cos_width-1 downto 0);  -- cosh of eta1 - eta2
        cos_dphi : in std_logic_vector(cosh_cos_width-1 downto 0);   -- cos of phi1 - phi2
        inv_mass_comp : out std_logic;
        sim_inv_mass_sq_div2 : out std_logic_vector(pt1_width+pt2_width+cosh_cos_width-1 downto 0)
    );
end invariant_mass;

architecture rtl of invariant_mass is

  constant INV_MASS_VECTOR_WIDTH : positive := pt1_width+pt2_width+cosh_cos_width;
  constant INV_MASS_PRECISION_FACTOR : real := real(10**INV_MASS_PRECISION);.pkg.
  constant FACTOR_4_VECTOR : std_logic_vector((INV_MASS_COSH_COS_PRECISION+1)*4-1 downto 0) := conv_std_logic_vector(10**(INV_MASS_COSH_COS_PRECISION+1),(INV_MAS

  signal inv_mass_sq_div2   : std_logic_vector(INV_MASS_VECTOR_WIDTH-1 downto 0);
  signal upper_limit_vector : std_logic_vector(INV_MASS_VECTOR_WIDTH-1 downto 0);
  signal lower_limit_vector : std_logic_vector(INV_MASS_VECTOR_WIDTH-1 downto 0);

  begin

    -- Converting the boundary value for the comparison
    upper_limit_vector <= conv_std_logic_vector((integer(upper_limit*INV_MASS_PRECISION_FACTOR)),INV_MASS_VECTOR_WIDTH-FACTOR_4_VECTOR'length)*FACTOR_4_VECTOR;
    lower_limit_vector <= conv_std_logic_vector((integer(lower_limit*INV_MASS_PRECISION_FACTOR)),INV_MASS_VECTOR_WIDTH-FACTOR_4_VECTOR'length)*FACTOR_4_VECTOR;

    -- Calculation of invariant mass with the formula: M**2/2 = pt1*pt2 * (cosh(eta1 - eta2) - cos(phi1 - phi2))
    inv_mass_sq_div2 <= pt1 * pt2 * (cosh_deta - cos_dphi);
    sim_inv_mass_sq_div2 <= inv_mass_sq_div2;

    -- Comparison with boundary values
    inv_mass_comp <= '1' when (inv_mass_sq_div2 >= lower_limit_vector and inv_mass_sq_div2 <= upper_limit_vector) else '0';

end architecture rtl;
```

M. Jeitler

FPGA trigger code

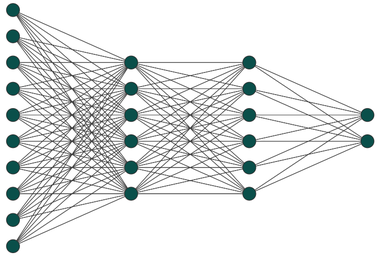
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

use work.gtl_pkg.all;

entity invariant_mass is
    generic (
        upper_limit: real := 15.0;
        lower_limit: real := 10.0;
        pt1_width: positive := 12;
        pt2_width: positive := 12;
        cosh_cos_width: positive := 28;
        INV_MASS_PRECISION : positive := 1;
        INV_MASS_COSH_COS_PRECISION : positive := 3
    );
    port(
        pt1 : in std_logic_vector(pt1_width-1 downto 0);
        pt2 : in std_logic_vector(pt2_width-1 downto 0);
        cosh_deta : in std_logic_vector(cosh_cos_width-1 downto 0);  -- cosh of eta1 - eta2
        cos_dphi : in std_logic_vector(cosh_cos_width-1 downto 0);   -- cos of phi1 - phi2
        inv_mass_comp : out std_logic;
        sim_inv_mass_sq_div2 : out std_logic_vector(pt1_width+pt2_width+cosh_cos_width-1 downto 0)
    );
end invariant_mass;

architecture rtl of invariant_mass is

  constant INV_MASS_VECTOR_WIDTH : positive := pt1_width+pt2_width+cosh_cos_width;
  constant INV_MASS_PRECISION_FACTOR : real := real(10**INV_MASS_PRECISION);.pkg.
  constant FACTOR_4_VECTOR : std_logic_vector((INV_MASS_COSH_COS_PRECISION+1)*4-1 downto 0) := conv_std_logic_vector(10**(INV_MASS_COSH_COS_PRECISION+1),(INV_MAS

  signal inv_mass_sq_div2   : std_logic_vector(INV_MASS_VECTOR_WIDTH-1 downto 0);
  signal upper_limit_vector : std_logic_vector(INV_MASS_VECTOR_WIDTH-1 downto 0);
  signal lower_limit_vector : std_logic_vector(INV_MASS_VECTOR_WIDTH-1 downto 0);

  begin

    -- Converting the boundary value for the comparison
    upper_limit_vector <= conv_std_logic_vector((integer(upper_limit*INV_MASS_PRECISION_FACTOR)),INV_MASS_VECTOR_WIDTH-FACTOR_4_VECTOR'length)*FACTOR_4_VECTOR;
    lower_limit_vector <= conv_std_logic_vector((integer(lower_limit*INV_MASS_PRECISION_FACTOR)),INV_MASS_VECTOR_WIDTH-FACTOR_4_VECTOR'length)*FACTOR_4_VECTOR;

    -- Calculation of invariant mass with the formula: M**2/2 = pt1*pt2 * (cosh(eta1 - eta2) - cos(phi1 - phi2))
    inv_mass_sq_div2 <= pt1 * pt2 * (cosh_deta - cos_dphi);
    sim_inv_mass_sq_div2 <= inv_mass_sq_div2;

    -- Comparison with boundary values
    inv_mass_comp <= '1' when (inv_mass_sq_div2 >= lower_limit_vector and inv_mass_sq_div2 <= upper_limit_vector) else '0';

end architecture rtl;
```

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

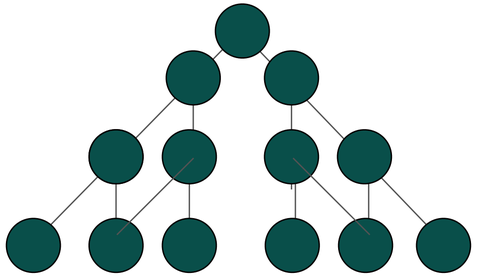
Generic HLS implementations for DNN inference

M. Jeitler

TensorFlow / TF Keras / PyTorch / ONNX



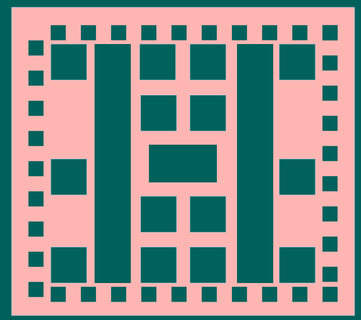
scikit-learn / XGBoost / TMVA



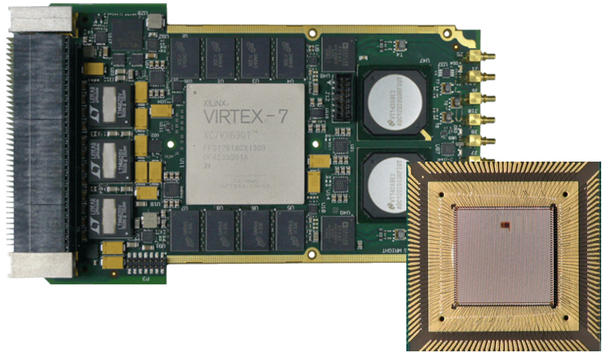
hls4ml

Conifer

HLS project:
Xilinx Vitis HLS, Intel Quartus HLS,
Mentor Catapult HLS



```
pip install hls4ml
pip install conifer
```



Conv2D

ReLU

MaxPool2D

Conv2D

ReLU

MaxPool2D

Flatten

Dense

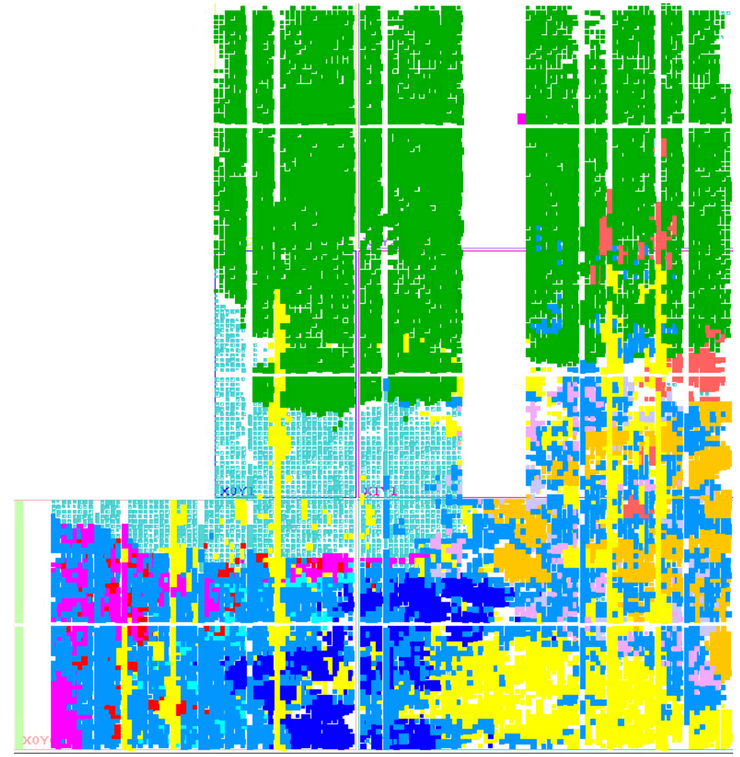
Softmax

Prediction

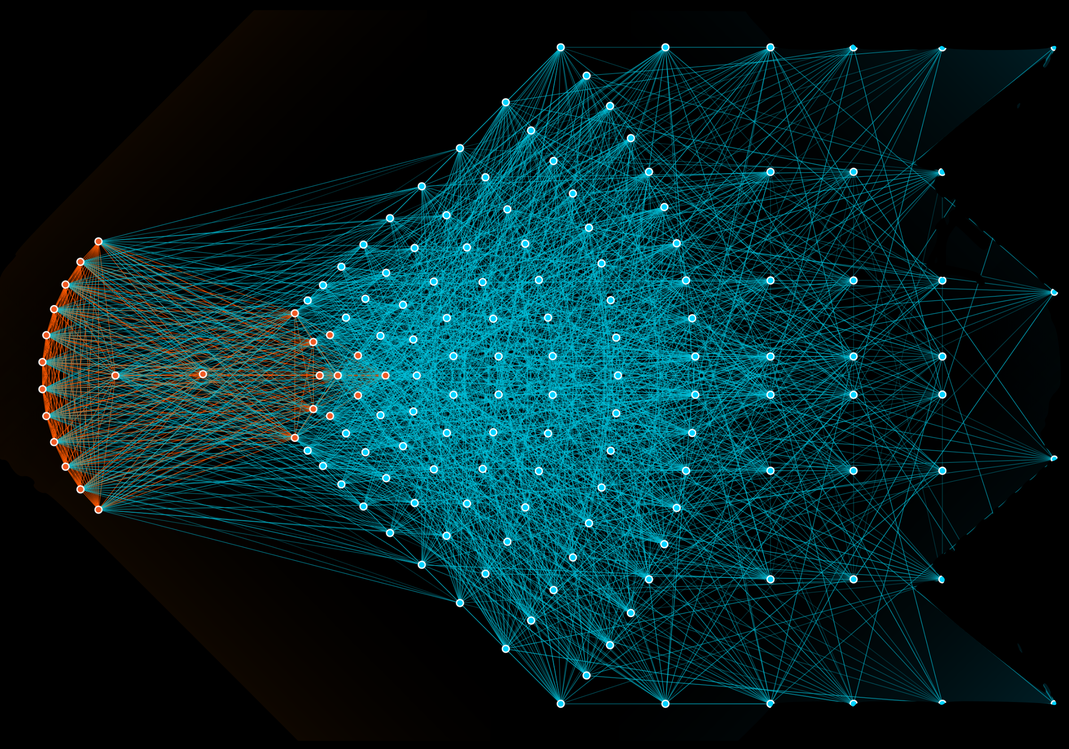
hls4ml

```
from hls4ml import …
import tensorflow as tf

# train or load a model
model = … # e.g. tf.keras.models.load_model(…)

# make a config template
cfg = config_from_keras_model(model,
granularity='name')

# tune the config
cfg['LayerName']['layer2']['ReuseFactor'] = 4

# do the conversion
hmodel = convert_from_keras_model(model, cfg)

# write and compile the HLS
hmodel.compile()

# run bit accurate emulation
y_tf = model.predict(x)
y_hls = hmodel.predict(x)

# do some validation
np.testing.assert_allclose(y_tf, y_hls)

# run HLS synthesis
hmodel.build()
```

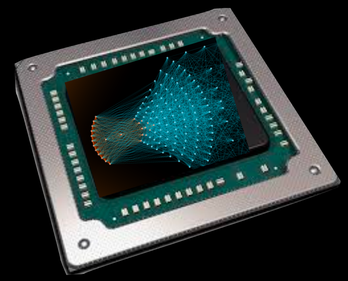
pynq-z2 floorplan



(from Sioni S Summers)



Ideally



Reality

Efficient NN design for edge compute

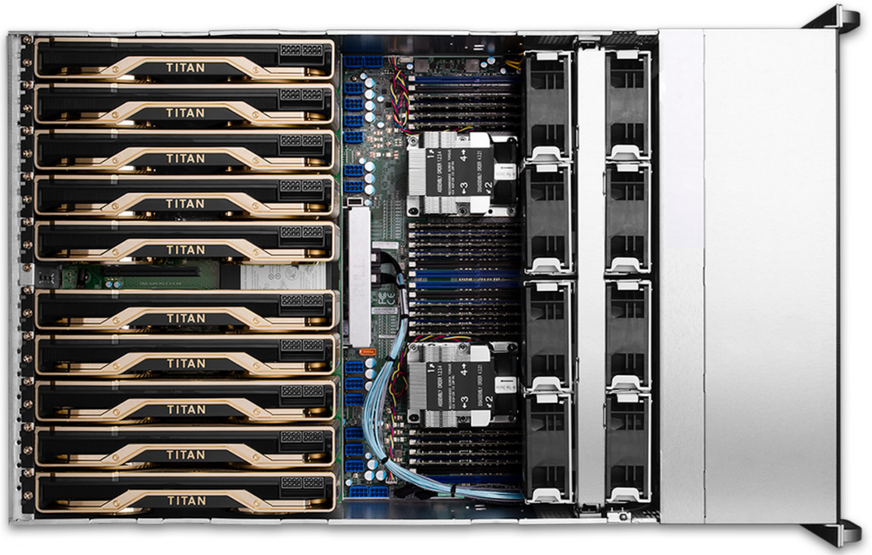
During training

- Quantization: do you really need 32-bit FP precision?
- Pruning: removal insignificant synapses
- Knowledge distillation

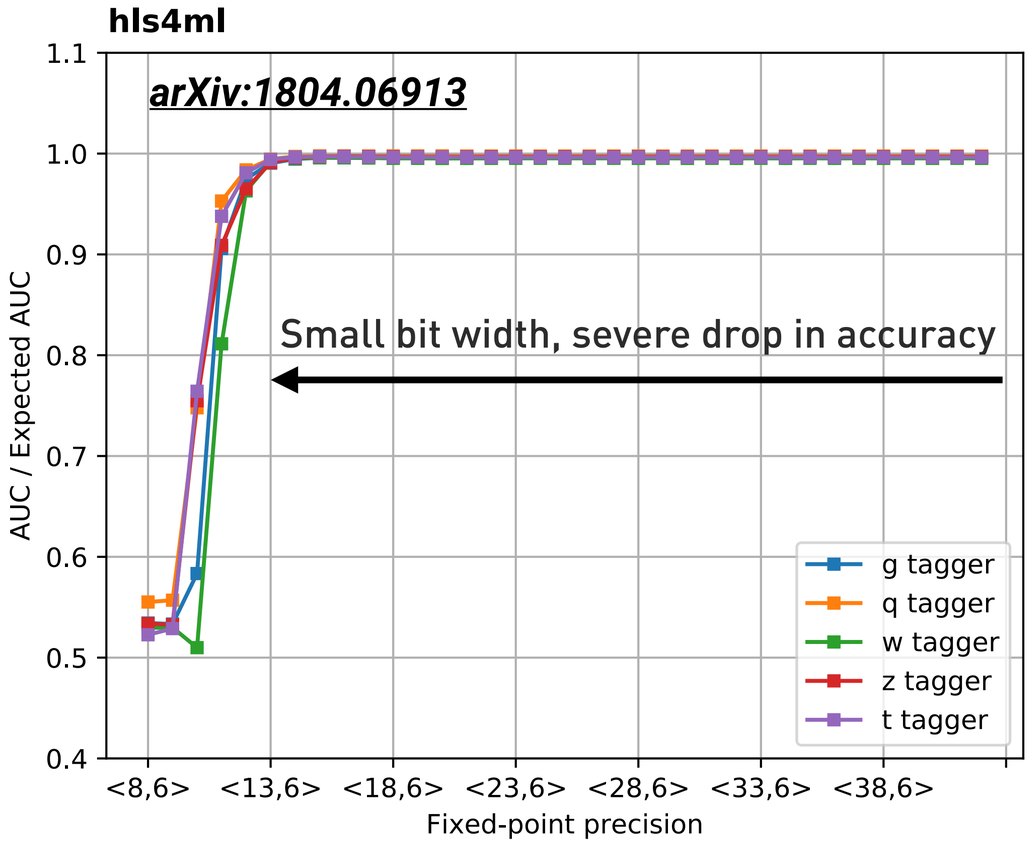
Post-training

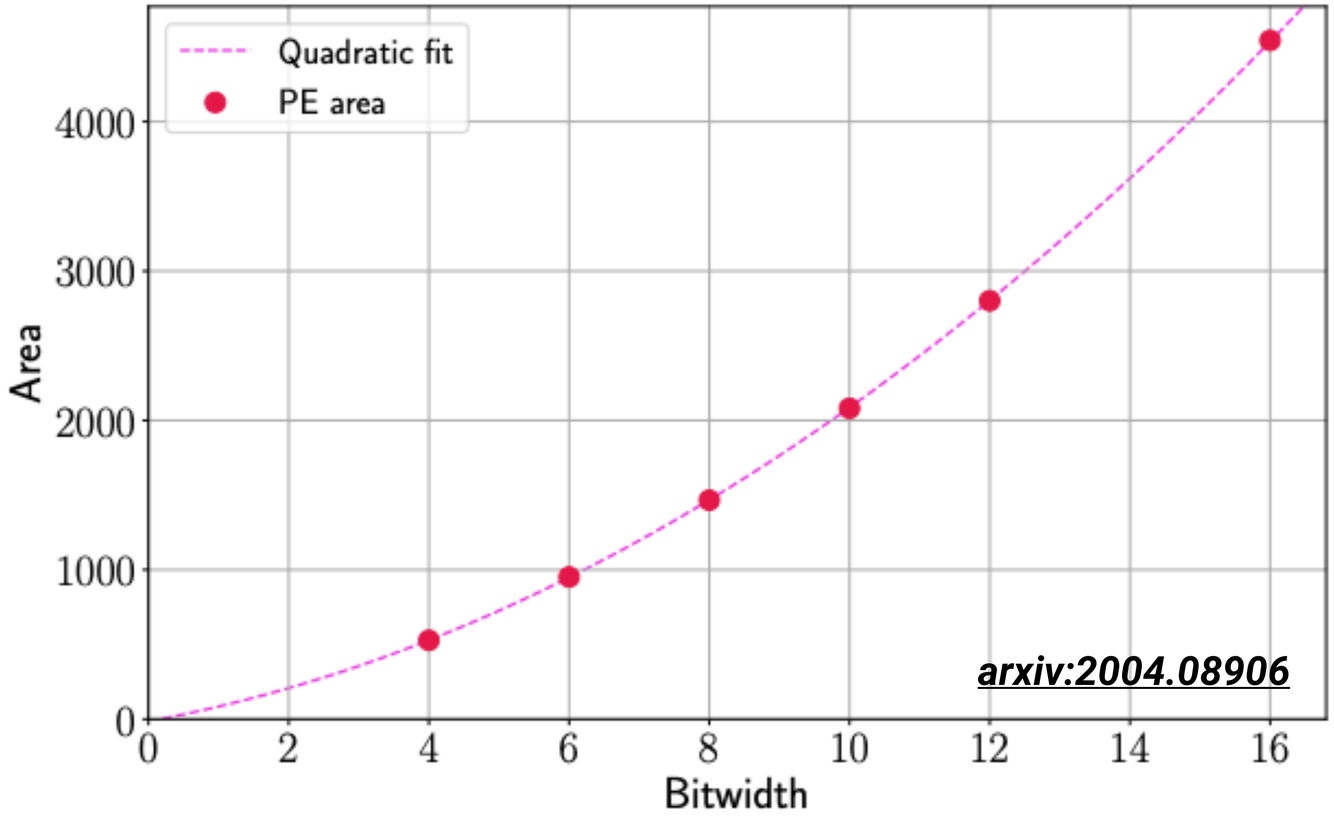
- Parallelisation (lower latency ↔ more resources)

From 8 GPU server to tiny FPGA!



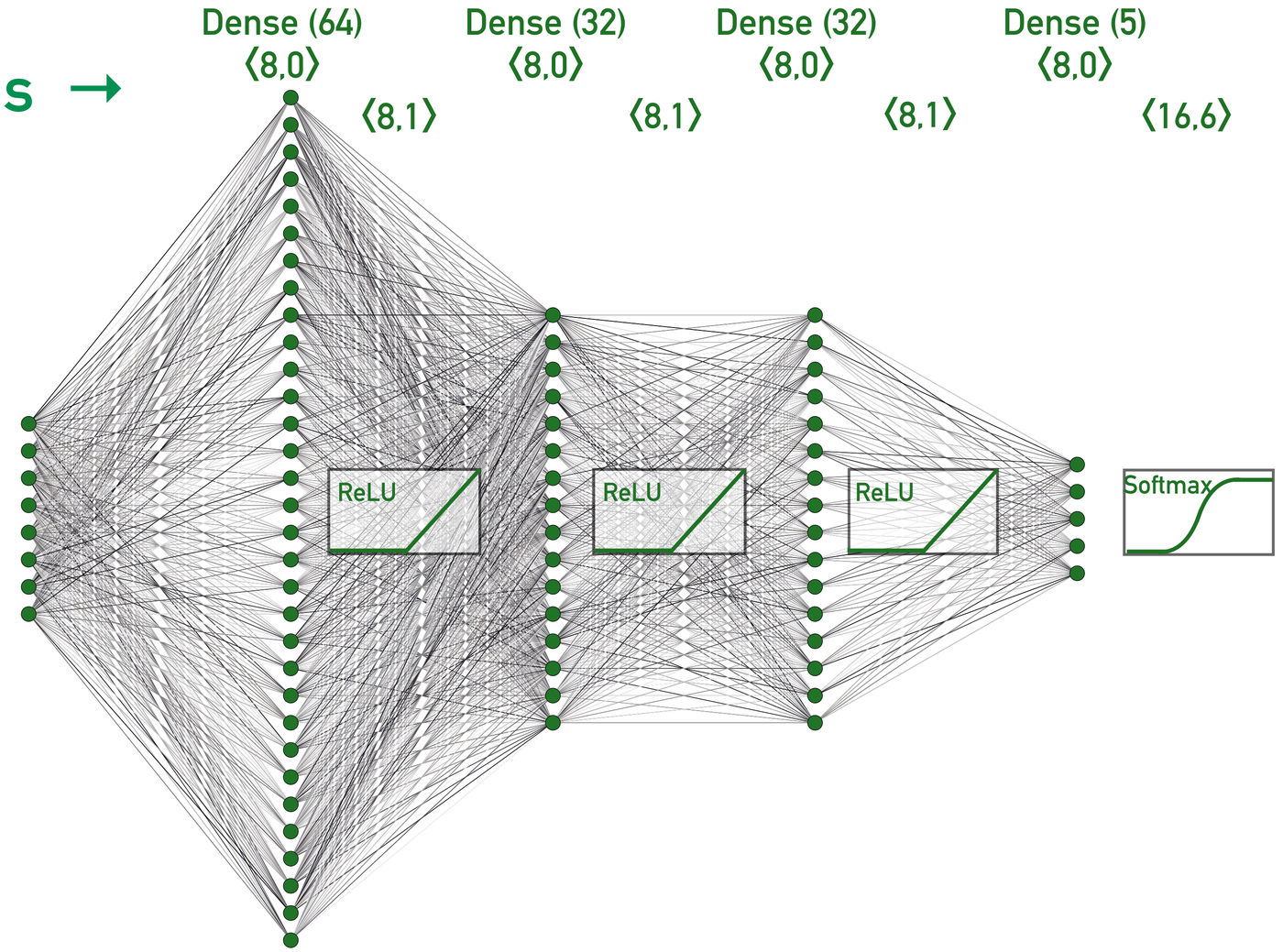
Quantization

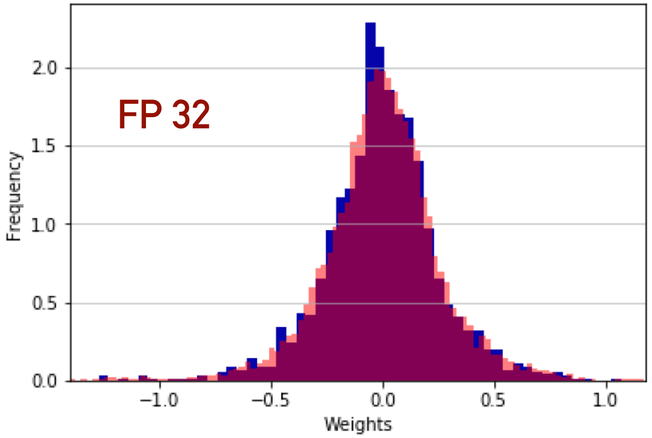


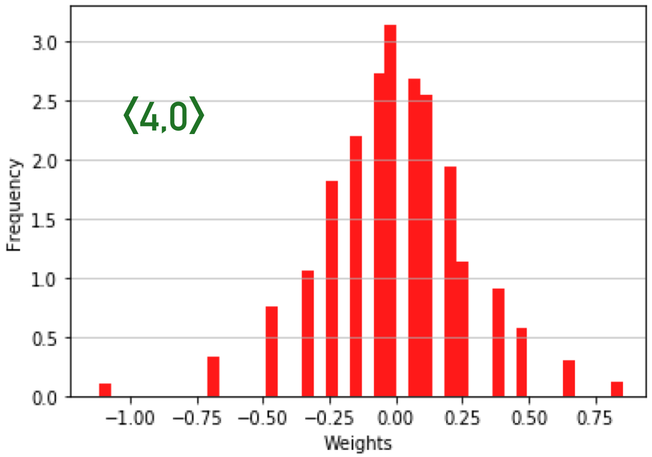


Quantization-aware training

Forward pass →

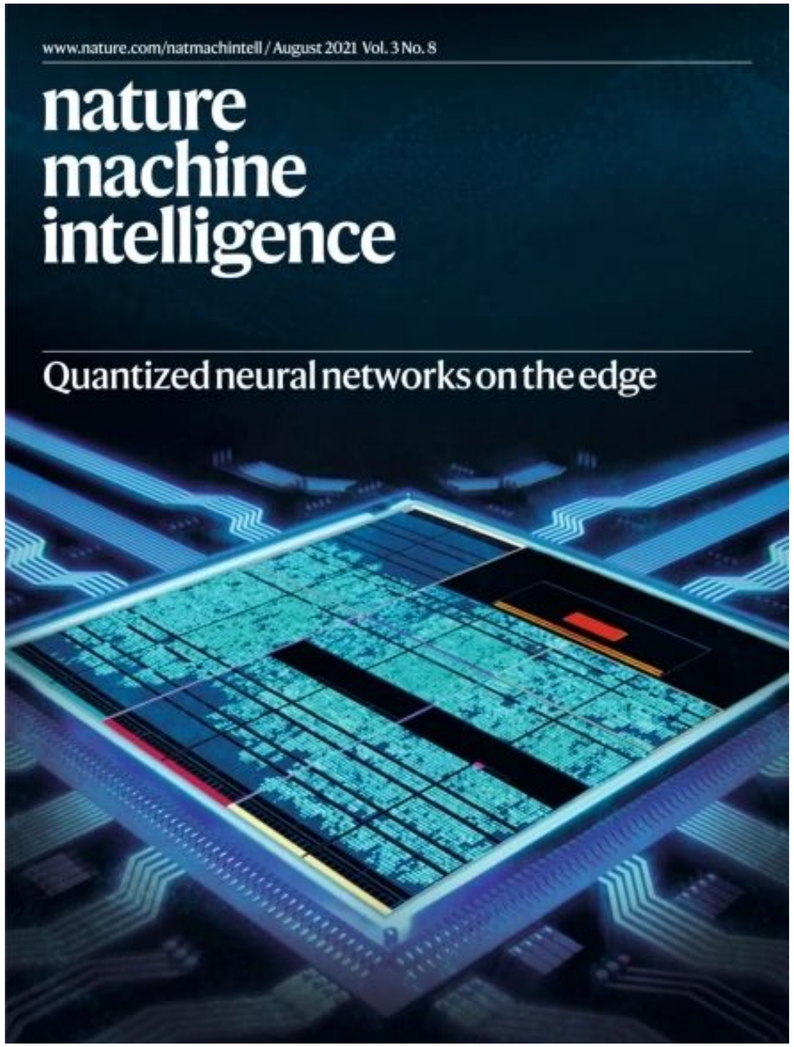


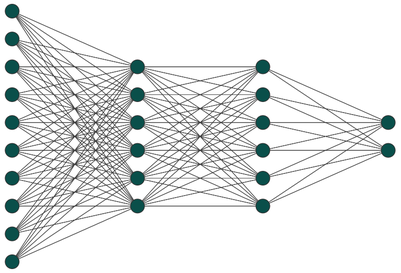




← Back propagation

Nature Machine Intelligence 3 (2021)





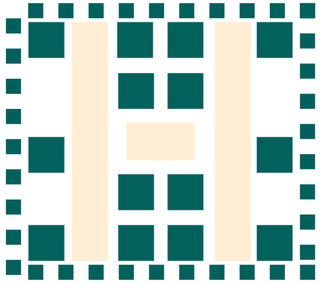
QKeras
Quantization-aware training

Google AI

QKeras
model

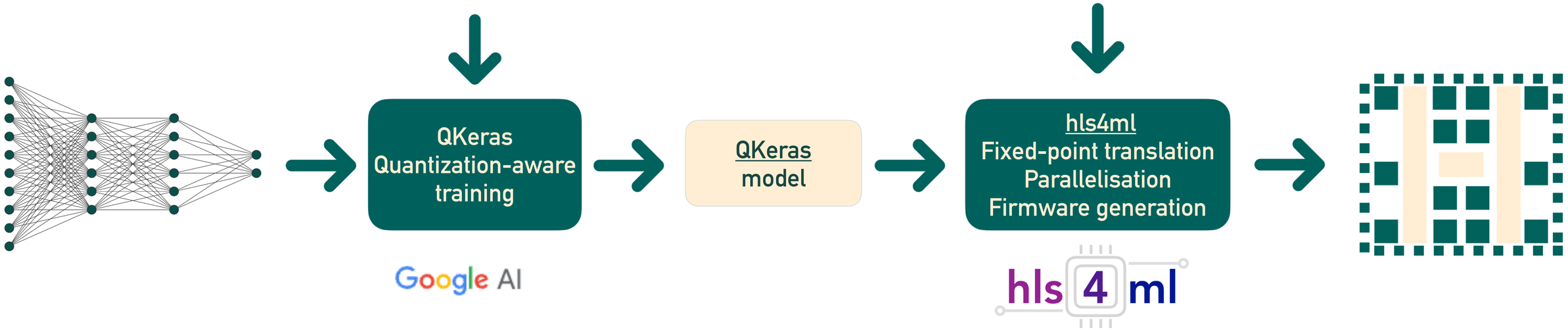
hls4ml
Fixed-point translation
Parallelisation
Firmware generation

hls 4 ml

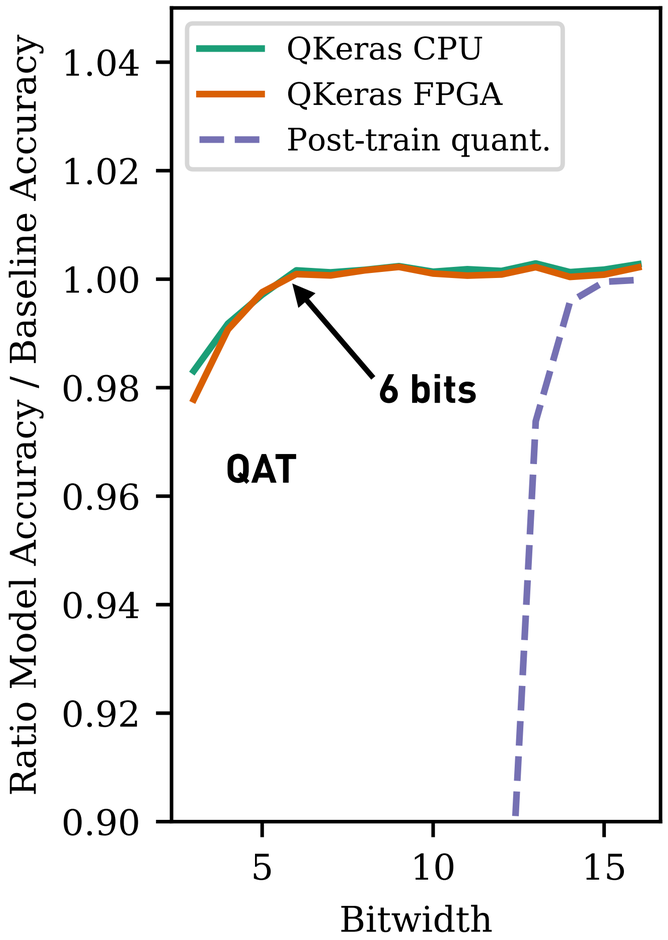


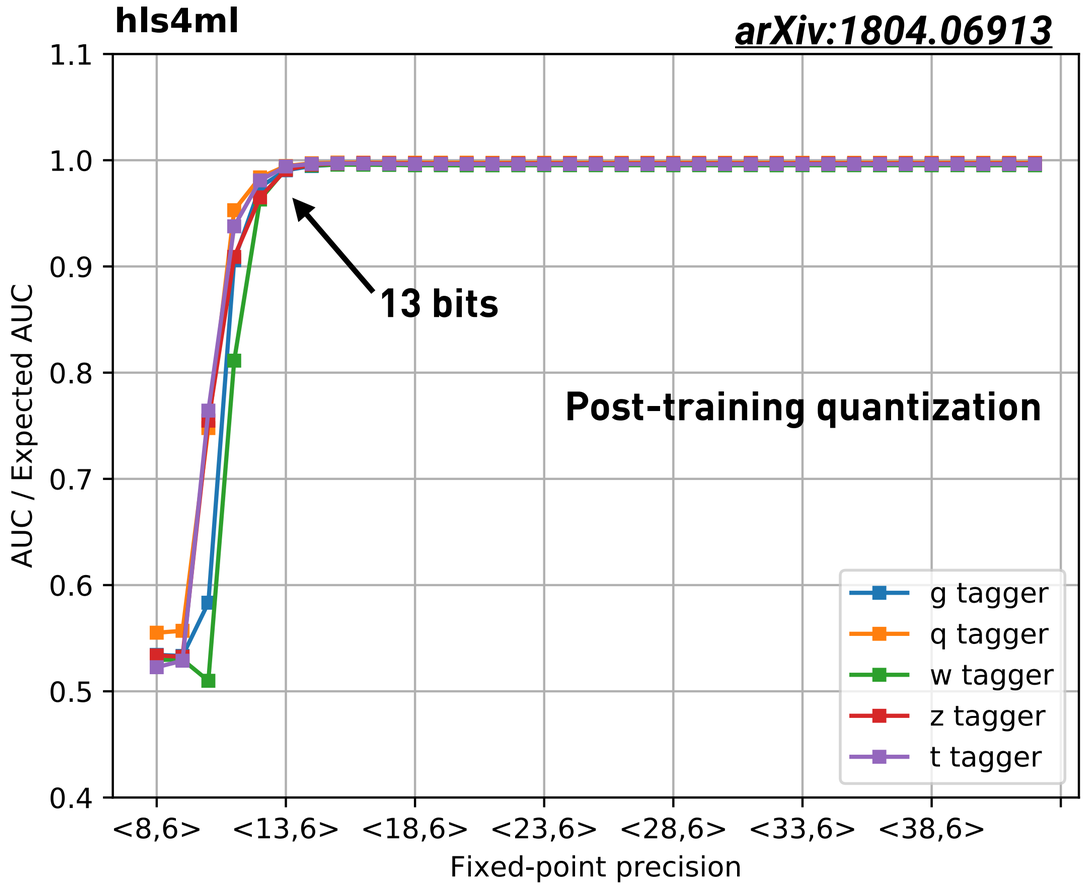
```
from tensorflow.keras.layers import Input, Activation
from qkeras import quantized_bits
from qkeras import QDense, QActivation
from qkeras import QBatchNormalization

x = Input((16))
x = QDense(64,
    kernel_quantizer = quantized_bits(6,0,alpha=1),
    bias_quantizer   = quantized_bits(6,0,alpha=1))(x)
x = QBatchNormalization()(x)
x = QActivation('quantized_relu(6,0)')(x)
x = QDense(32,
    kernel_quantizer = quantized_bits(6,0,alpha=1),
    bias_quantizer   = quantized_bits(6,0,alpha=1))(x)
x = QBatchNormalization()(x)
x = QActivation('quantized_relu(6,0)')(x)
x = QDense(32,
    kernel_quantizer = quantized_bits(6,0,alpha=1),
    bias_quantizer   = quantized_bits(6,0,alpha=1))(x)
x = QBatchNormalization()(x)
x = QActivation('quantized_relu(6,0)')(x)
x = QDense(5,
    kernel_quantizer = quantized_bits(6,0,alpha=1),
    bias_quantizer   = quantized_bits(6,0,alpha=1))(x)
x = Activation('softmax')(x)
```

```
from hls4ml import …
import tensorflow as tf

# train or load a model
model = tf.keras.models.load_model(…)

# make a config
cfg = config_from_keras_model(model,
granularity='name')

# do the conversion
hmodel = convert_from_keras_model(model, cfg)

# write and compile the HLS
hmodel.compile()

# run HLS synthesis
hmodel.build()
```



FPGA performance





Nature Machine Intelligence 3 (2021)

Why do tree-based models still outperform deep learning on typical tabular data?

Leo Grinsztajn, Edouard Oyallon, Gael Varoquaux

06 Jun 2022 (modified: 16 Jan 2023) NeurIPS 2022 Datasets and Benchmarks Readers: Everyone Show Bibtex Show Revisions

Abstract: While deep learning has enabled tremendous progress on text and image datasets, its superiority on tabular data is not clear. We contribute extensive benchmarks of standard and novel deep learning methods as well as tree-based models such as XGBoost and Random Forests, across a large number of datasets and hyperparameter combinations. We define a standard set of 45 datasets from varied domains with clear characteristics of tabular data and a benchmarking methodology accounting for both fitting models and finding good hyperparameters. Results show that tree-based models remain state-of-

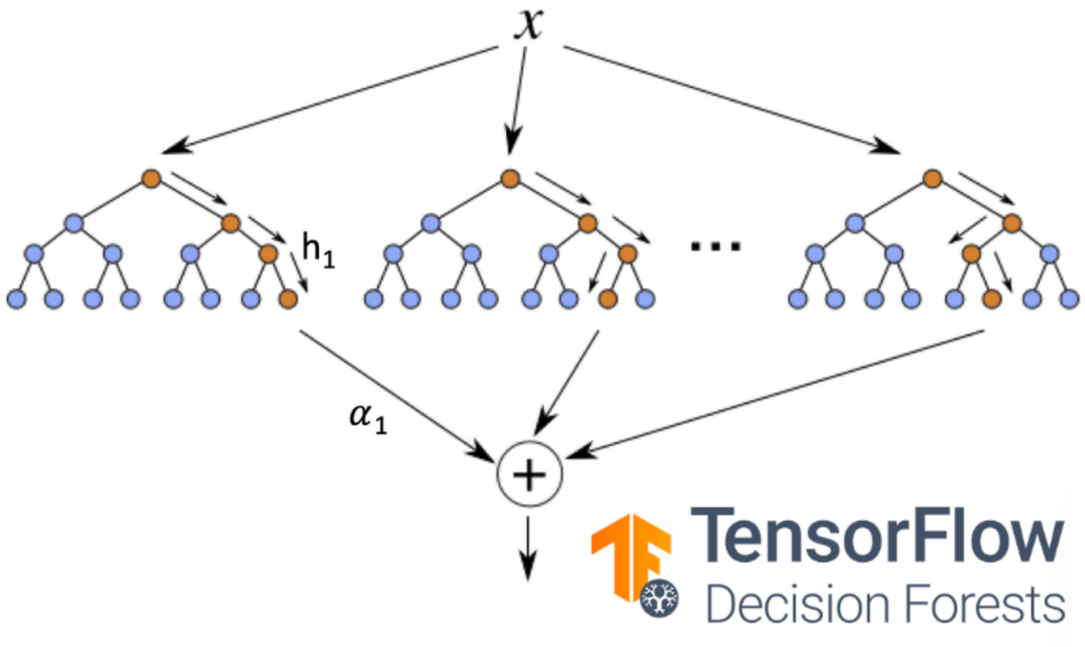
BDTs

Often the best way to compress is: Just use BDTs!

Conifer = hls4ml for BDTs

If resource/latency constrained, BDT might be solution

- Depending on data, can be as accurate as a DNN
- Usually significantly faster and more resource efficient



%VU9P	Accuracy	Latency	DSP	LUT
QKeras 6b	75.6%	40 ns	22 (~0%)	1%
sklearn + conifer	74.9%	5 ns	-	0.5%

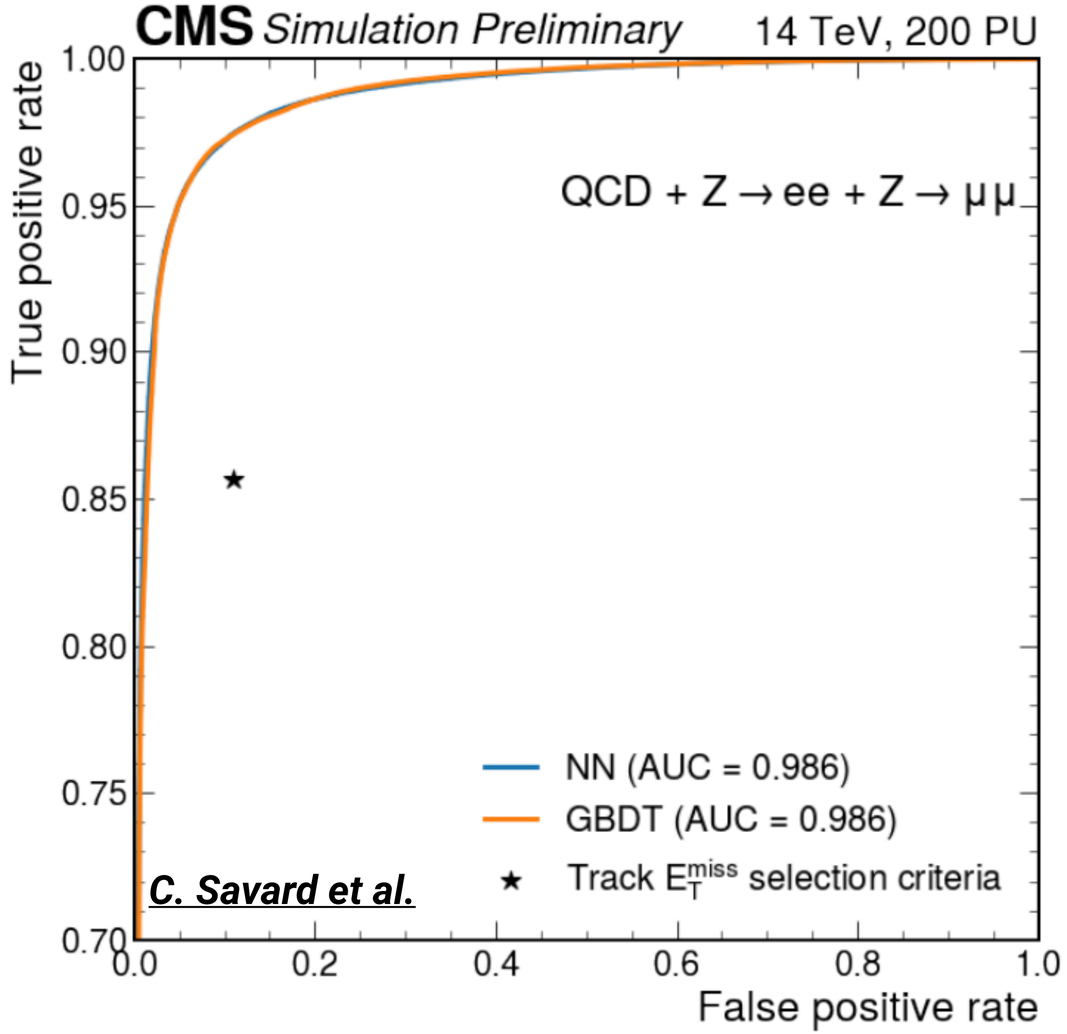
BDTs

Often the best way to compress is: Just use BDTs!

Conifer = hls4ml for BDTs

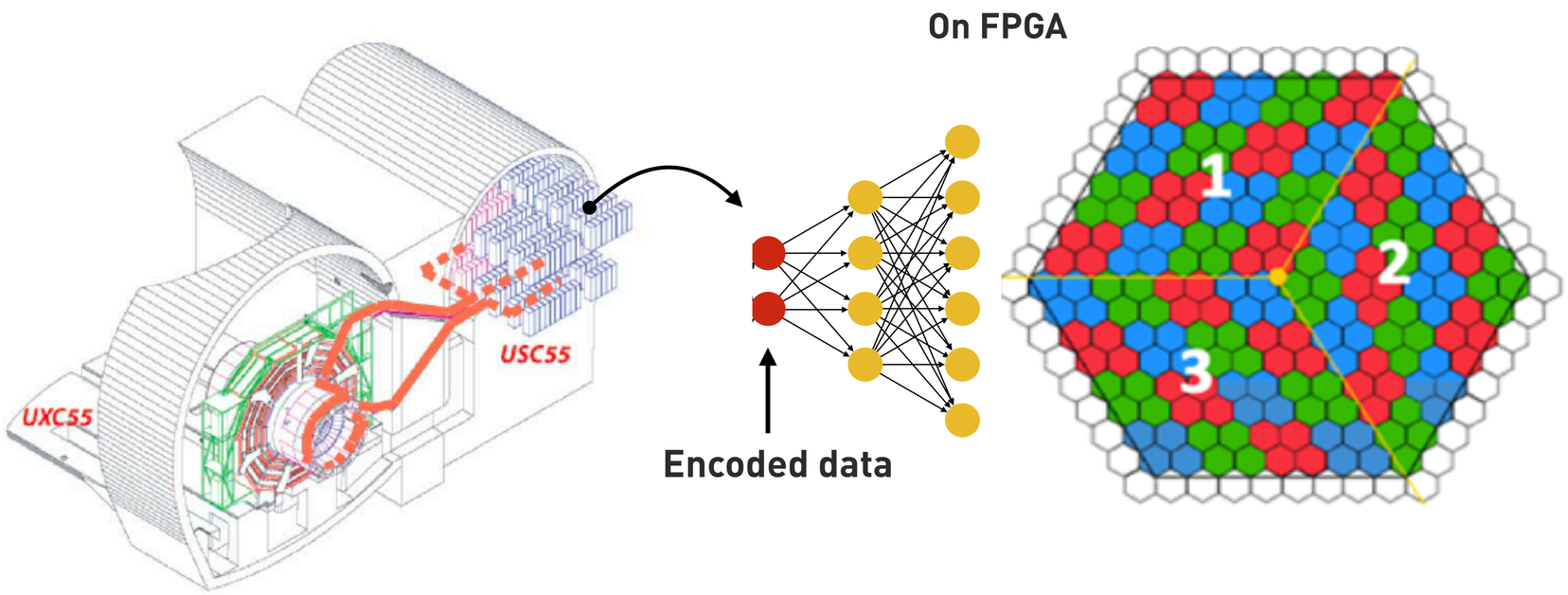
If resource/latency constrained, BDT might be solution

- Depending on data, can be as accurate as a DNN
- Usually significantly faster and more resource efficient



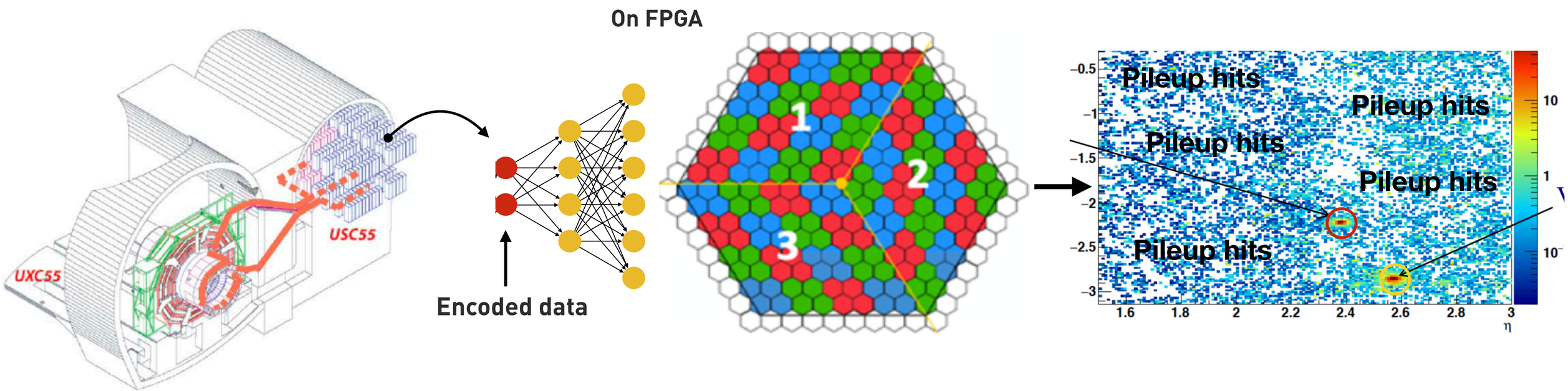
Model	Python AUC	HLS AUC	Latency (clk)	LUT %	FF %	DSP %
NN	0.985	0.982	8	0.104	0.029	0.292
GBDT	0.986	0.981	3	0.140	0.027	0.0

ML for reconstruction



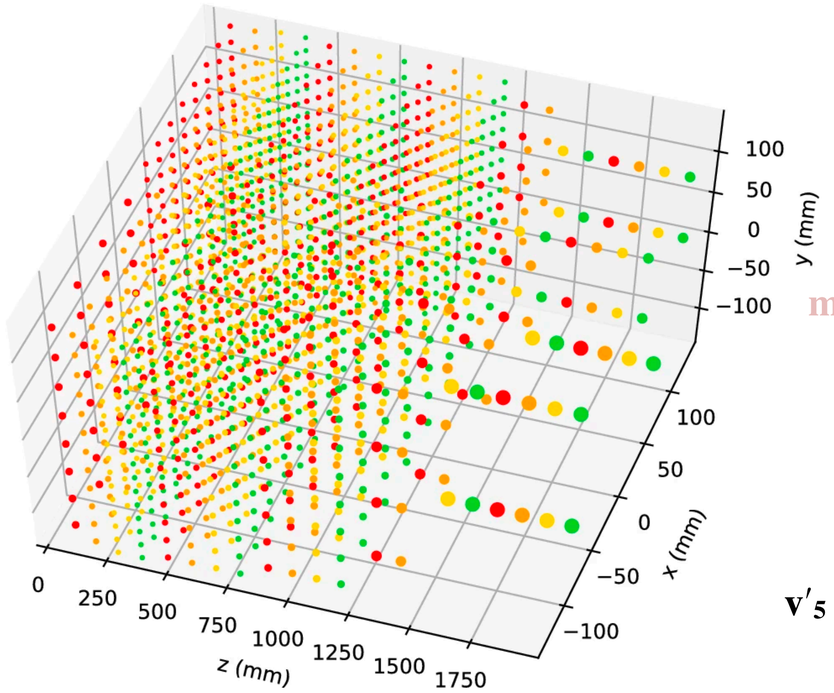
On FPGA: 3.5 μs to cluster energy deposits

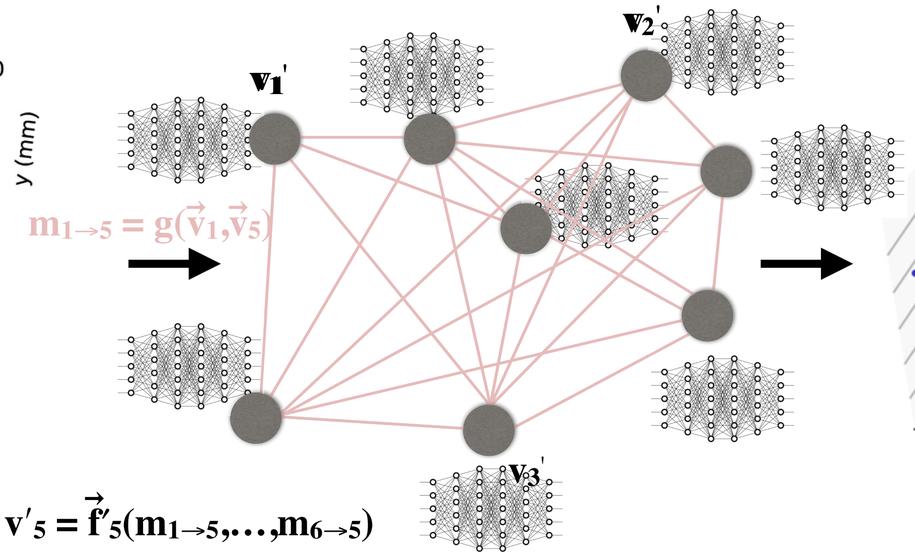
ML for reconstruction

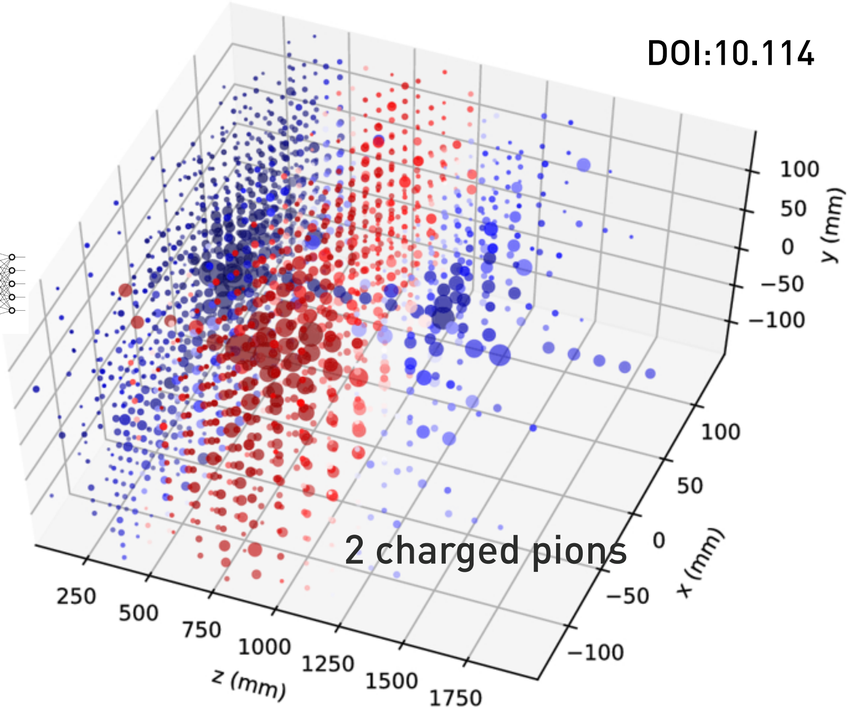


On FPGA: 3.5 μs to cluster energy deposits

ML for reconstruction







DOI:10.114

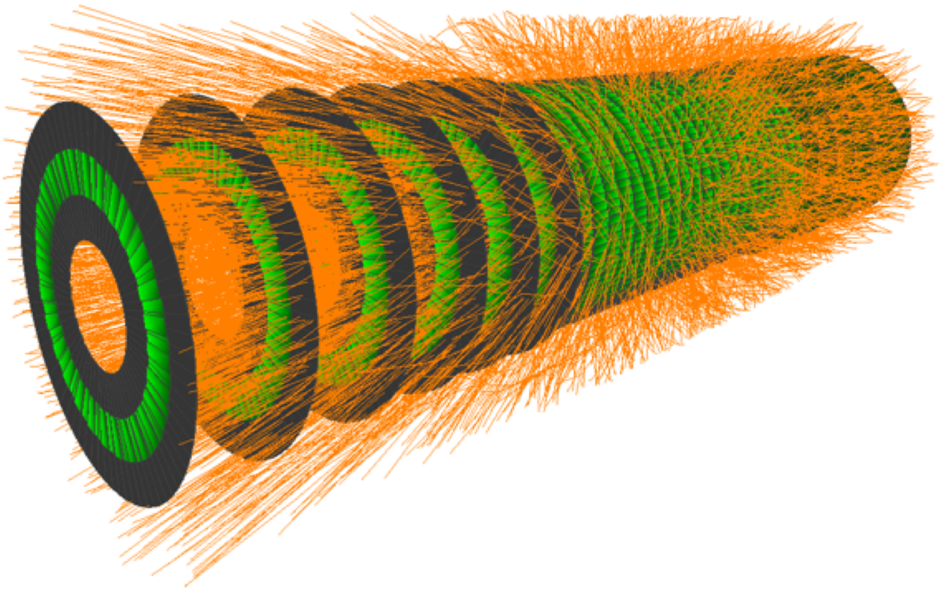
On FPGA: 3.5 μs to cluster energy deposits

- Graph Neural Networks for fast clustering of irregular geometry detectors

Work done on speeding this up from Imperial!

DOI:10.3389/fdata.2022.828666

ML for tracking



In HL-LHC, will need to do track finding at L1

- O(1000) hits, O(100) tracks, 40 MHz rate, ~5 μs latency

Graph Neural Networks for fast charged particle tracking

"Throughput-optimised" for L1 applications,
"resource-optimised" for co-processing

Design	(n_{nodes}, n_{edges})	RF	Precision	Latency [cycles]	II [cycles]	DSP [%]	LUT [%]	FF [%]	BRAM [%]
Throughput-opt.	(28, 56)	1	`ap_fixed<14,7>`	59	1	99.9	66.0	11.7	0.7
Resource-opt.	(28, 56)	1	`ap_fixed<14,7>`	79	28	56.6	17.6	3.9	13.1

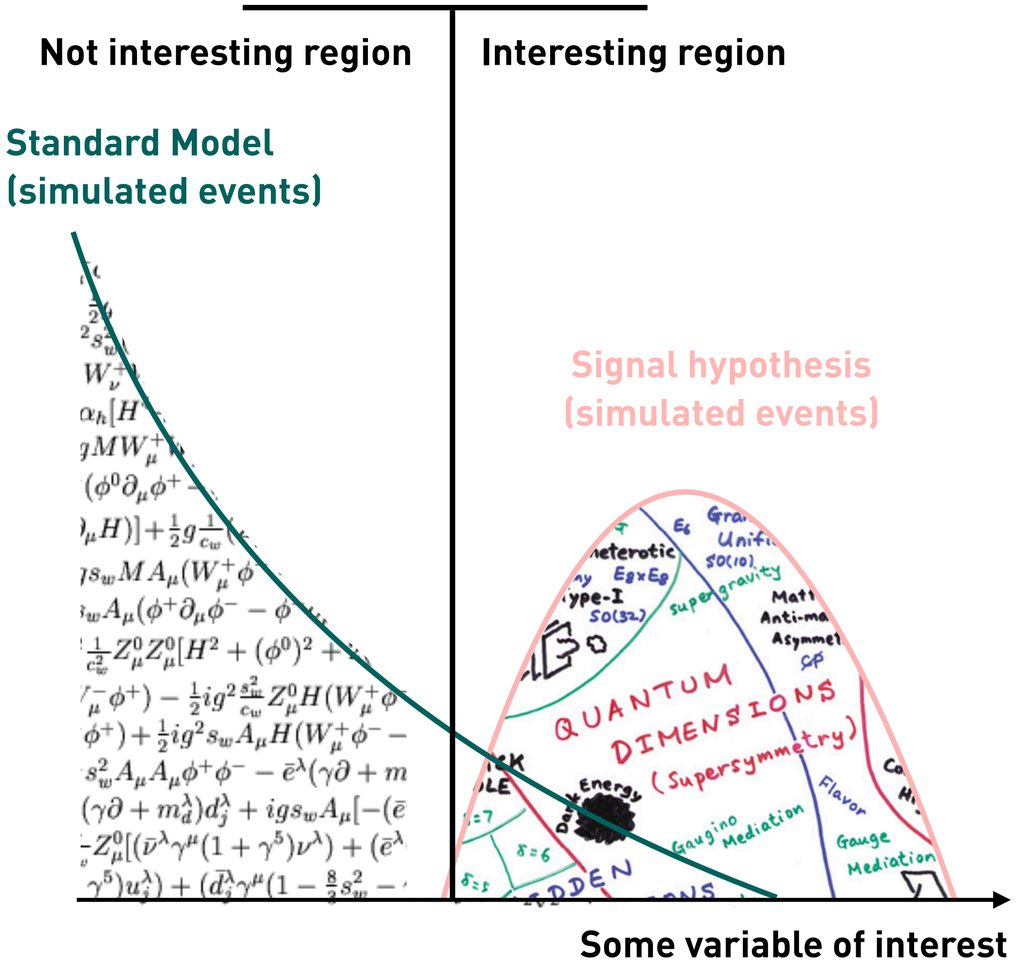
Which ML algorithms are we currently exploring to

do things
completely different
?

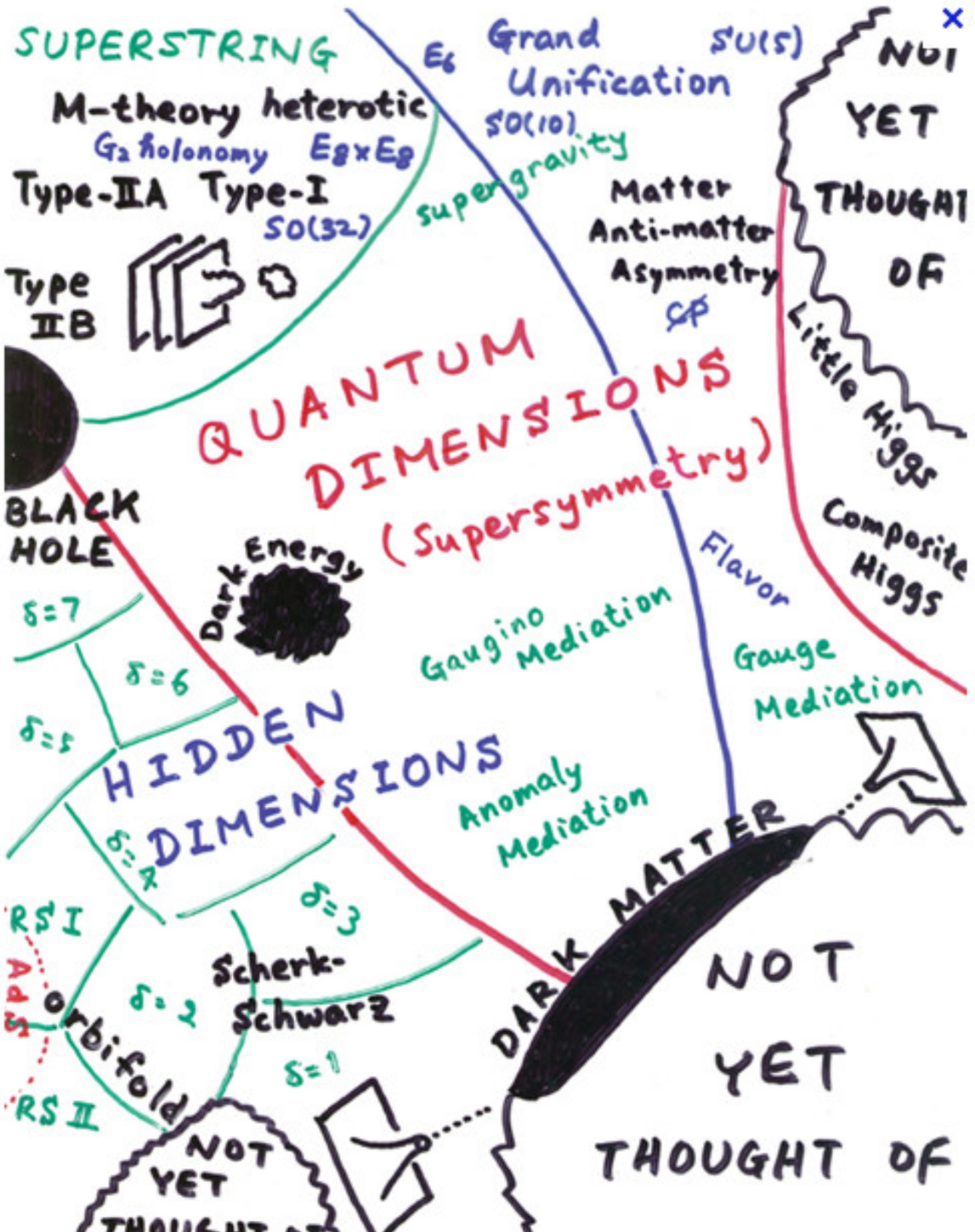
Searches for new particles at LHC

Searches for new particles at LHC

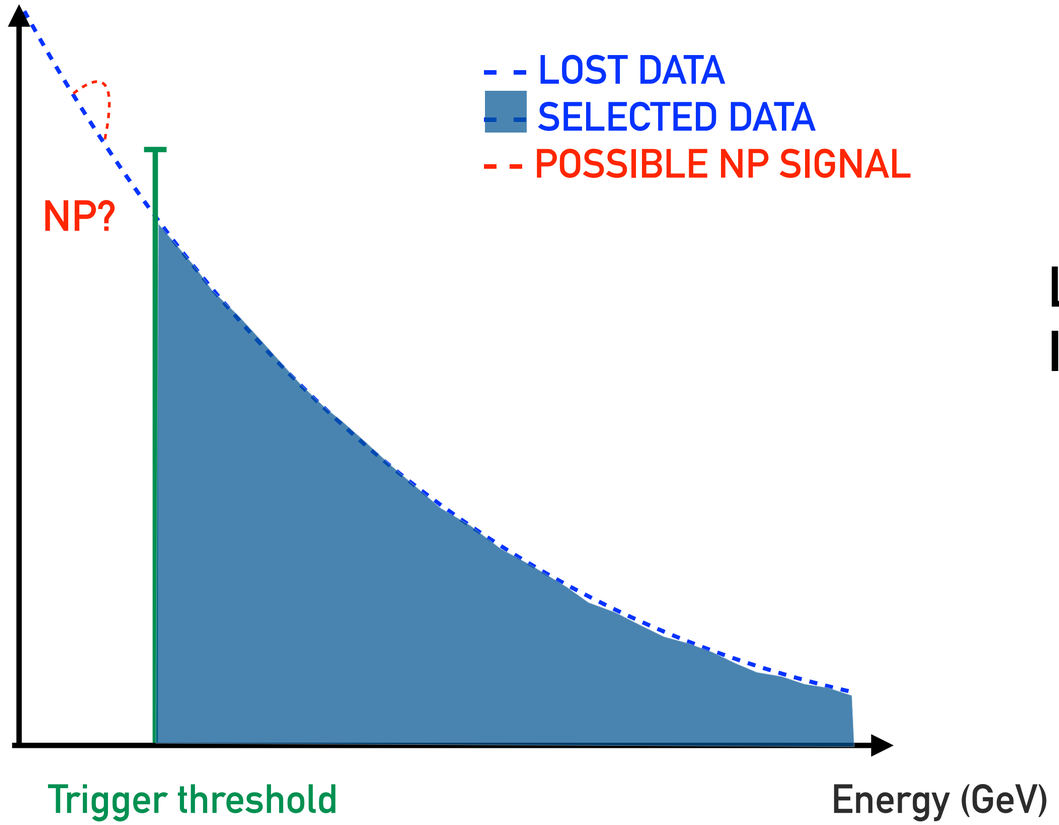
Bias in particle physics



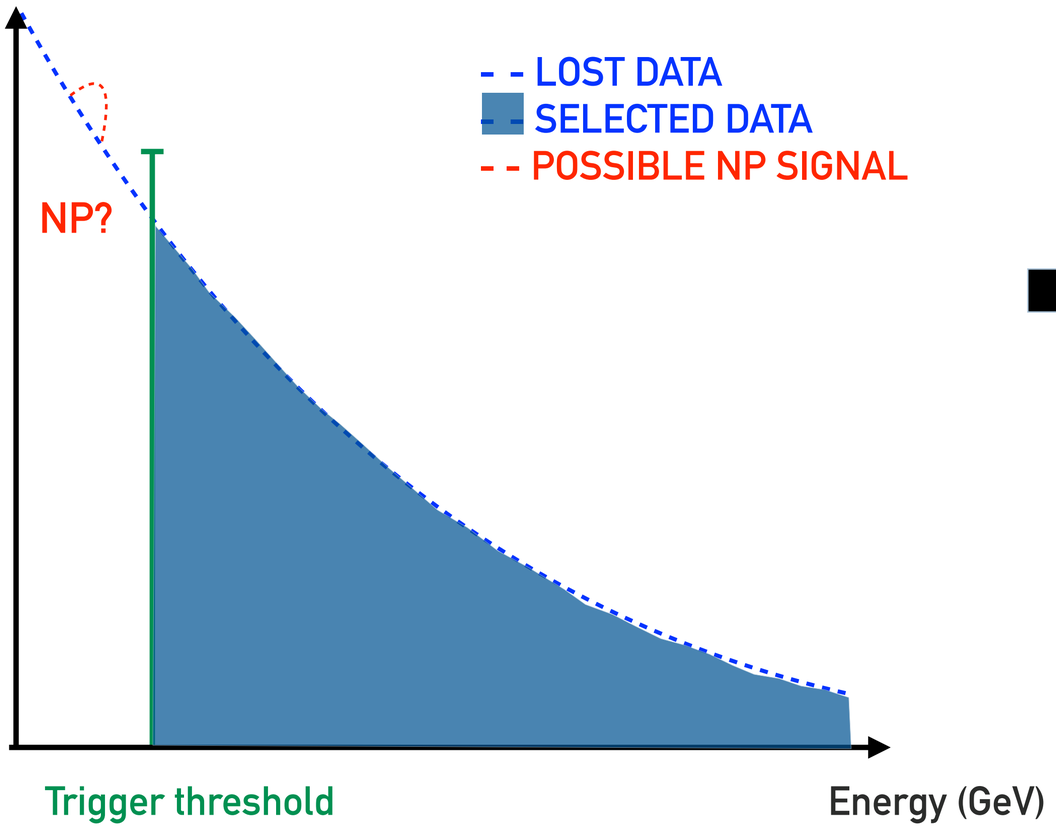
Need to exploit the full capabilities of the LHC and be more generic!



Limitations of current trigger

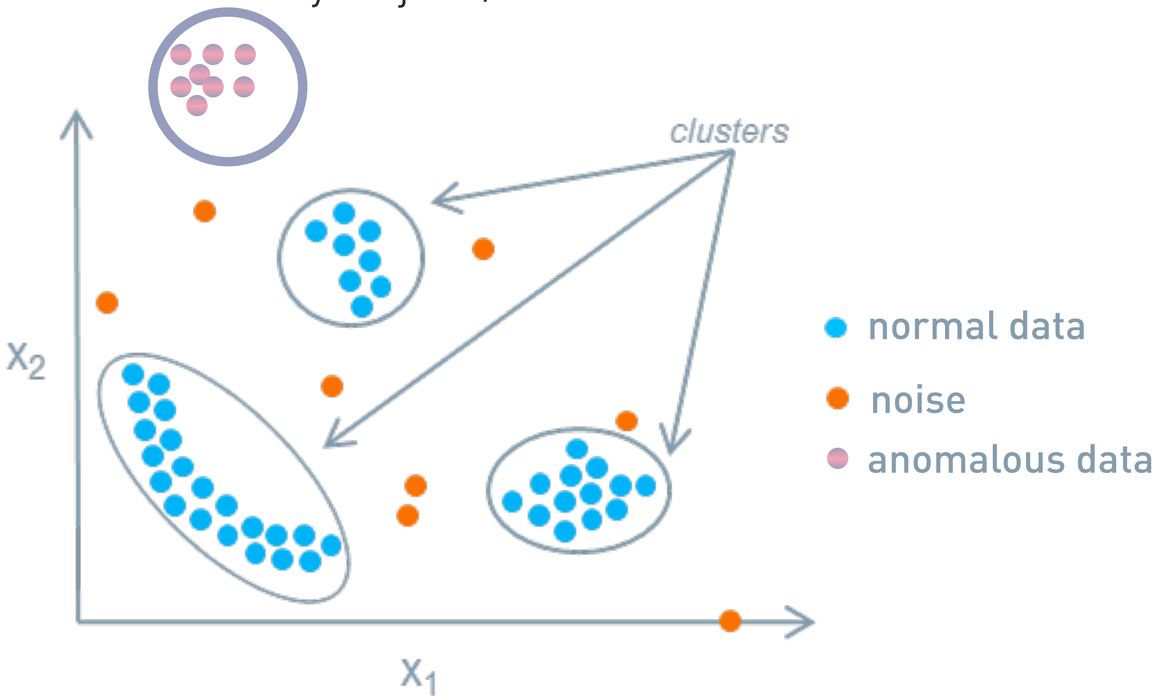


Level-1 rejects >99% of events!
Is there a smarter way to select?



Look at data rather than defining signal hypothesis a priori

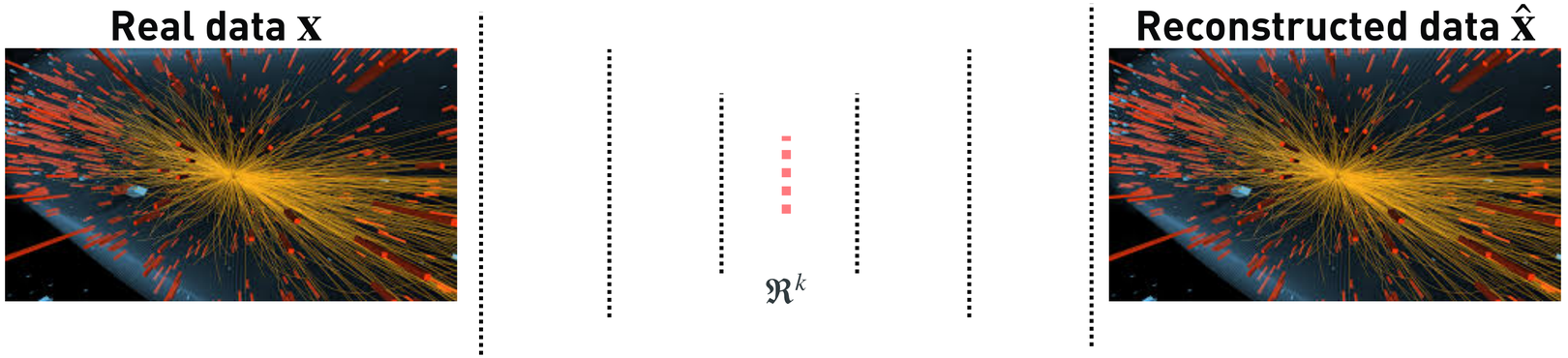
- Can we "classify" objects/events?



ML for anomaly detection

Autoencoders: Learns from data

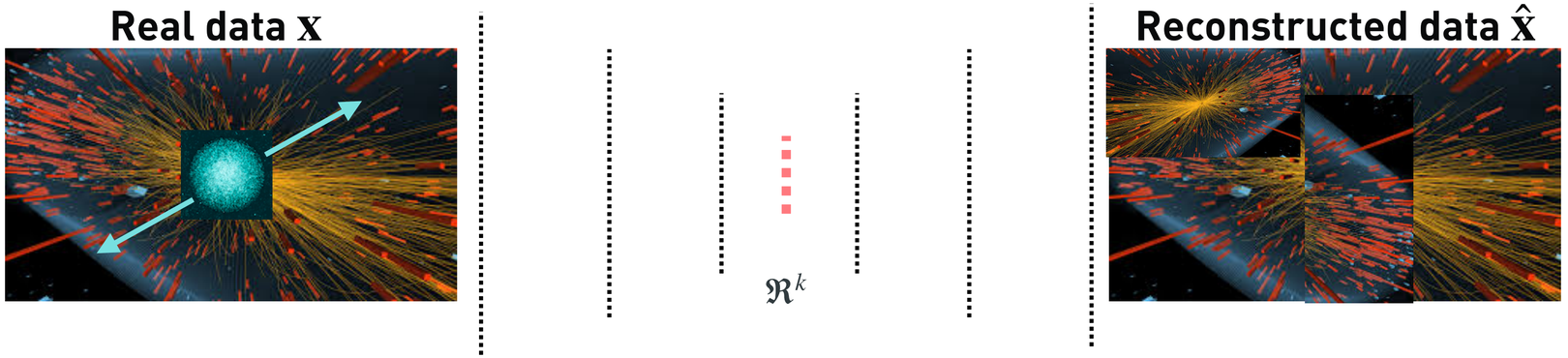
- Trains unsupervised
- Learns to compress, then reconstruct data
- Often used for financial fraud detection
  - Low rate of anomalous events versus high rate “background”



ML for anomaly detection

Autoencoders: Learns from data

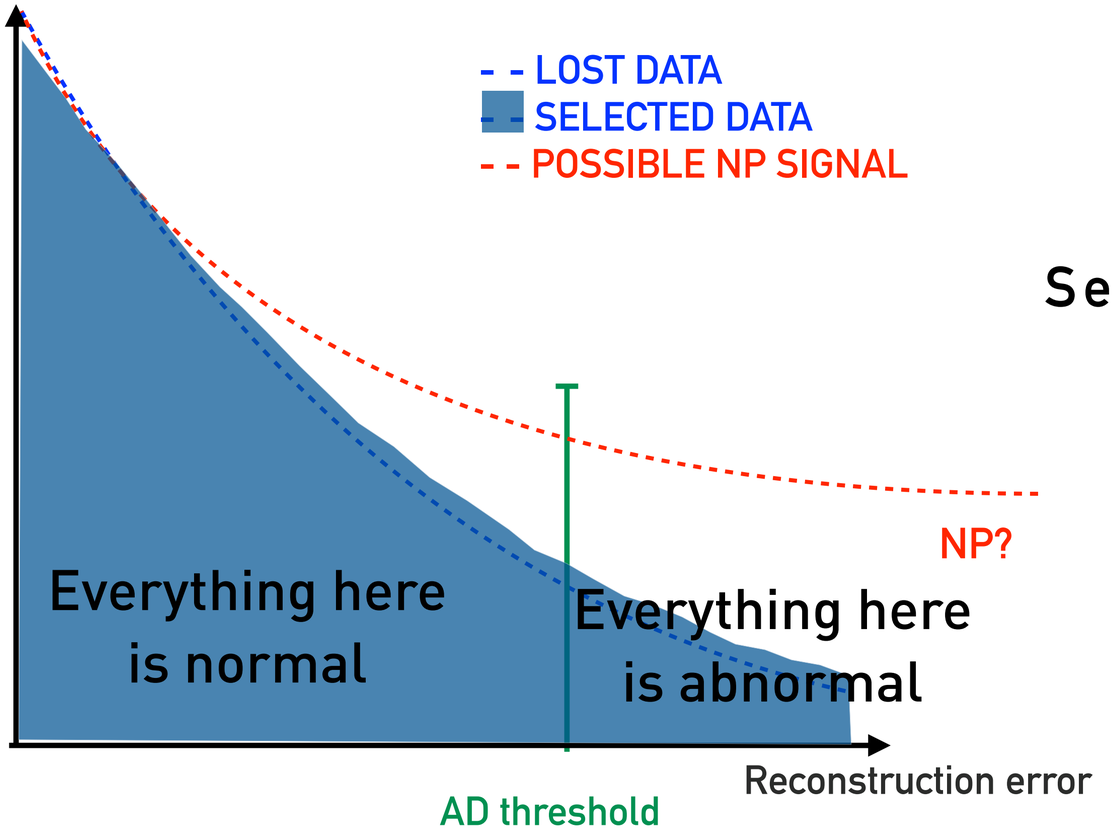
- Trains unsupervised
- Learns to compress, then reconstruct data
- Often used for financial fraud detection
  - Low rate of anomalous events versus high rate “background”



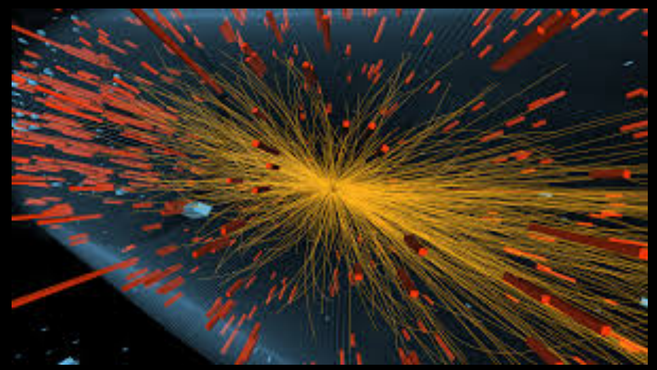
- Difference $\mathbf{x}$-$\hat{\mathbf{x}}$ defines "degree of abnormality”

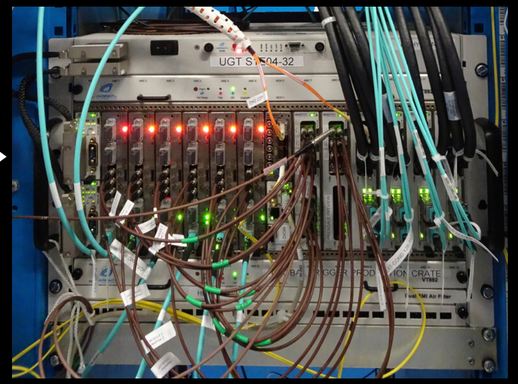
Nature Machine Intelligence 4, 154 (2022)

ML for anomaly detection

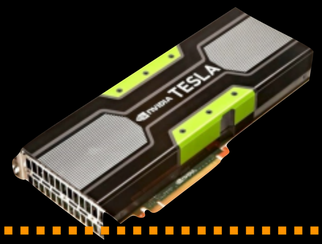


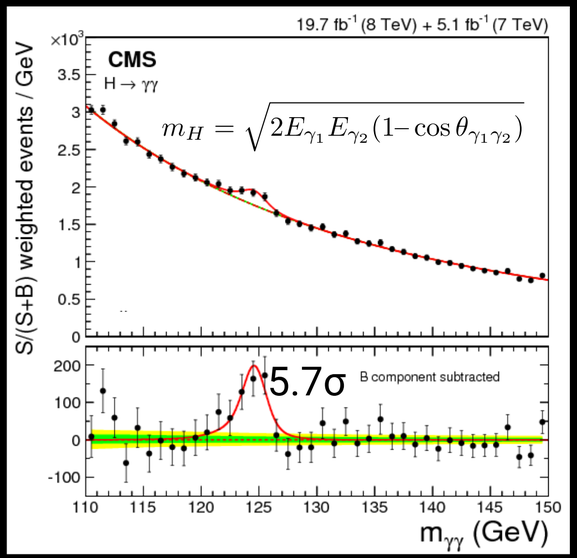
Select based on degree of abnormality!

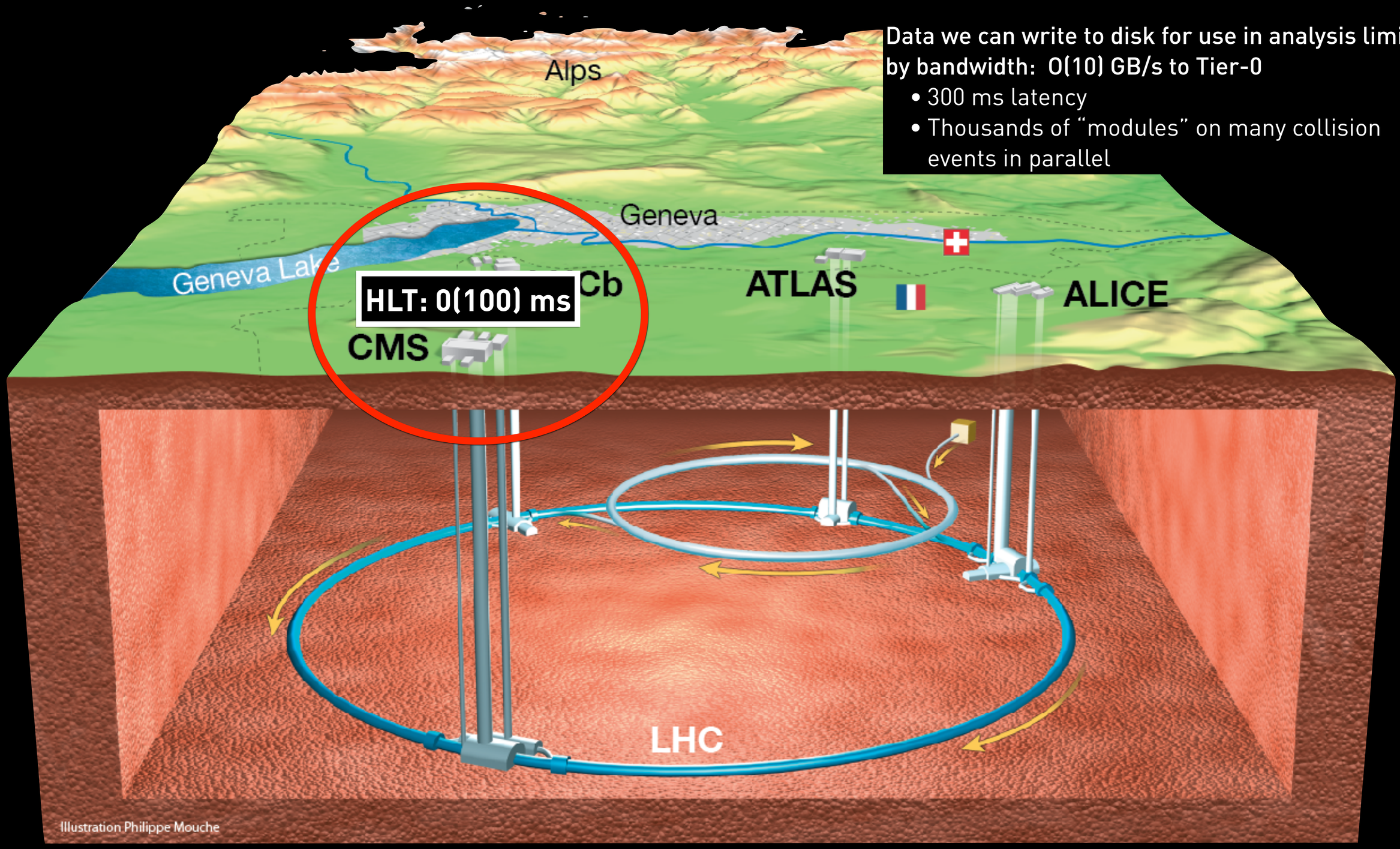




GPU inference







Data we can write to disk for use in analysis limited by bandwidth: O(10) GB/s to Tier-0

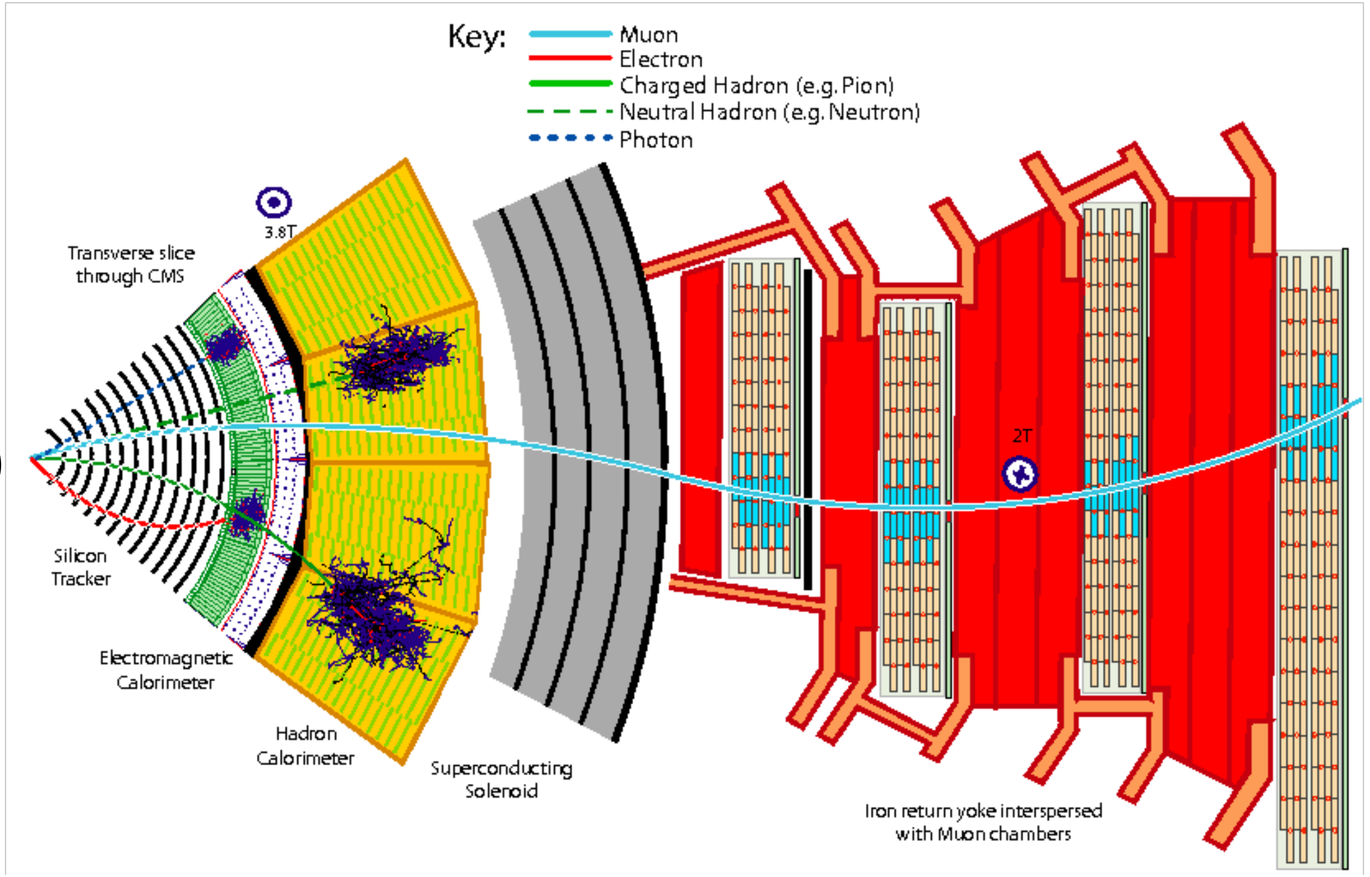
- 300 ms latency
- Thousands of "modules" on many collision events in parallel

HLT: More Particle Flow

Particle Flow: Best reconstruction at HLT

- Slow, cannot run on all events (currently 17%)

Bandwidth (kB/s) = Event rate (kHz) x Event size



AI for fast reconstruction

CPU node (16/20 cores)

Events from L1 @ 750 kHz

30k cores, single-threaded
→ ~300 ms available per event

CPU tasks

Raw data

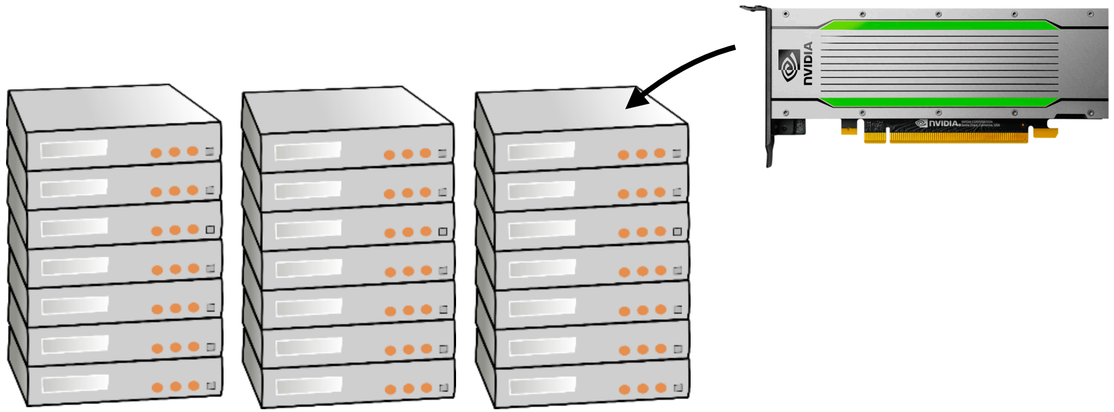
Pixel tracks
~ 48 ms

Pixel tracks

Particle Flow

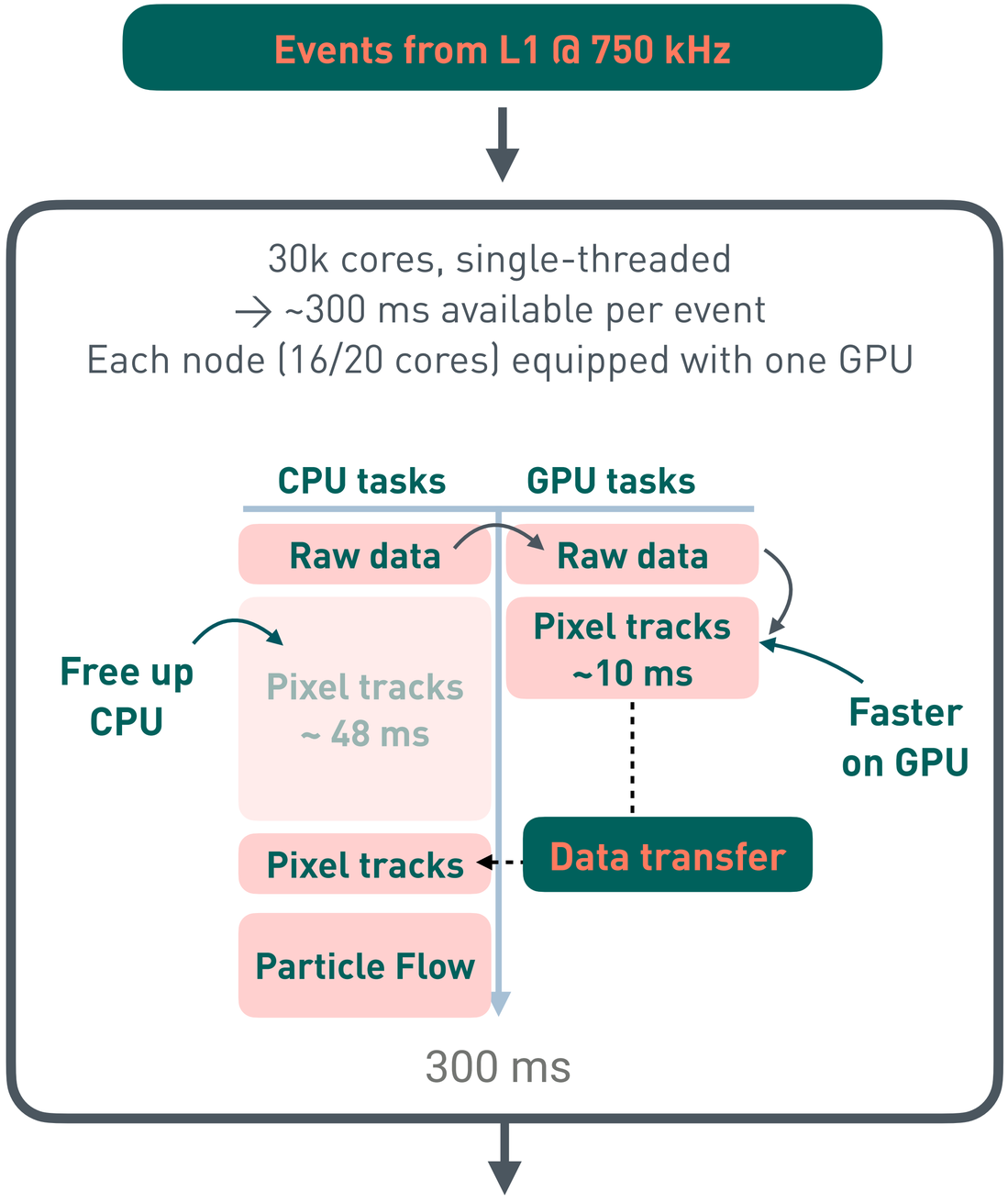
300 ms

AI for fast reconstruction

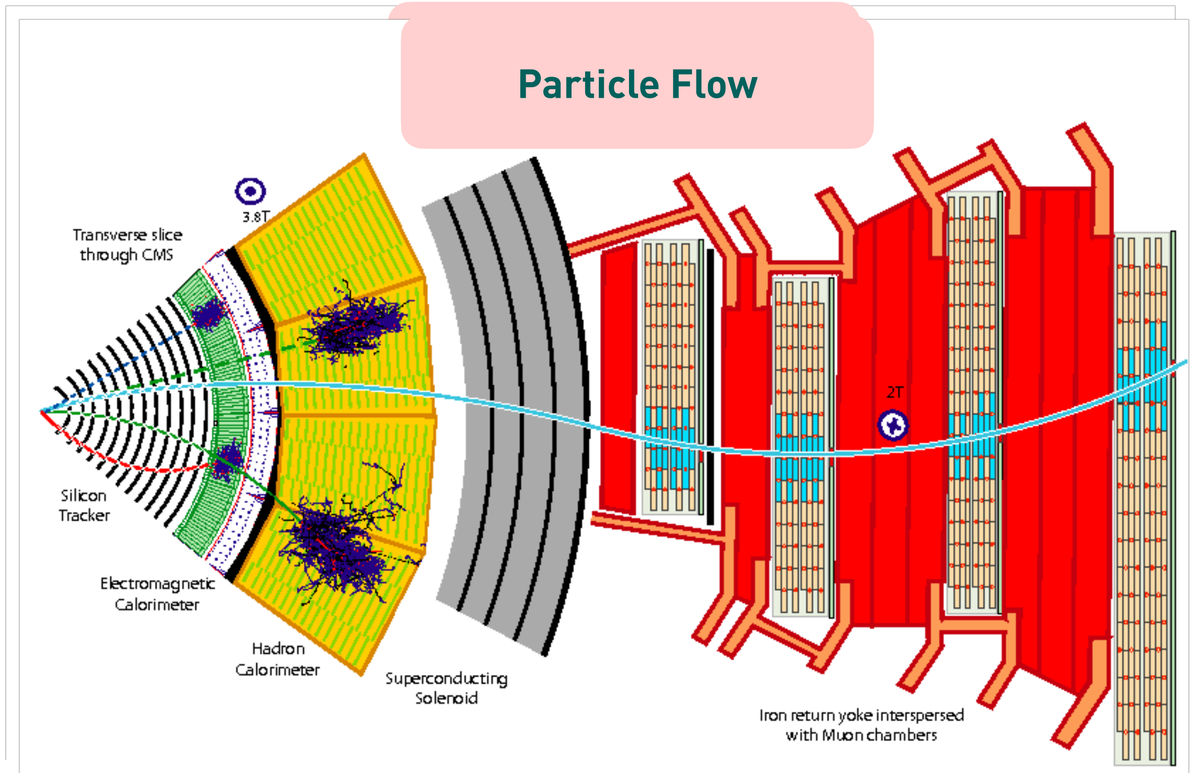


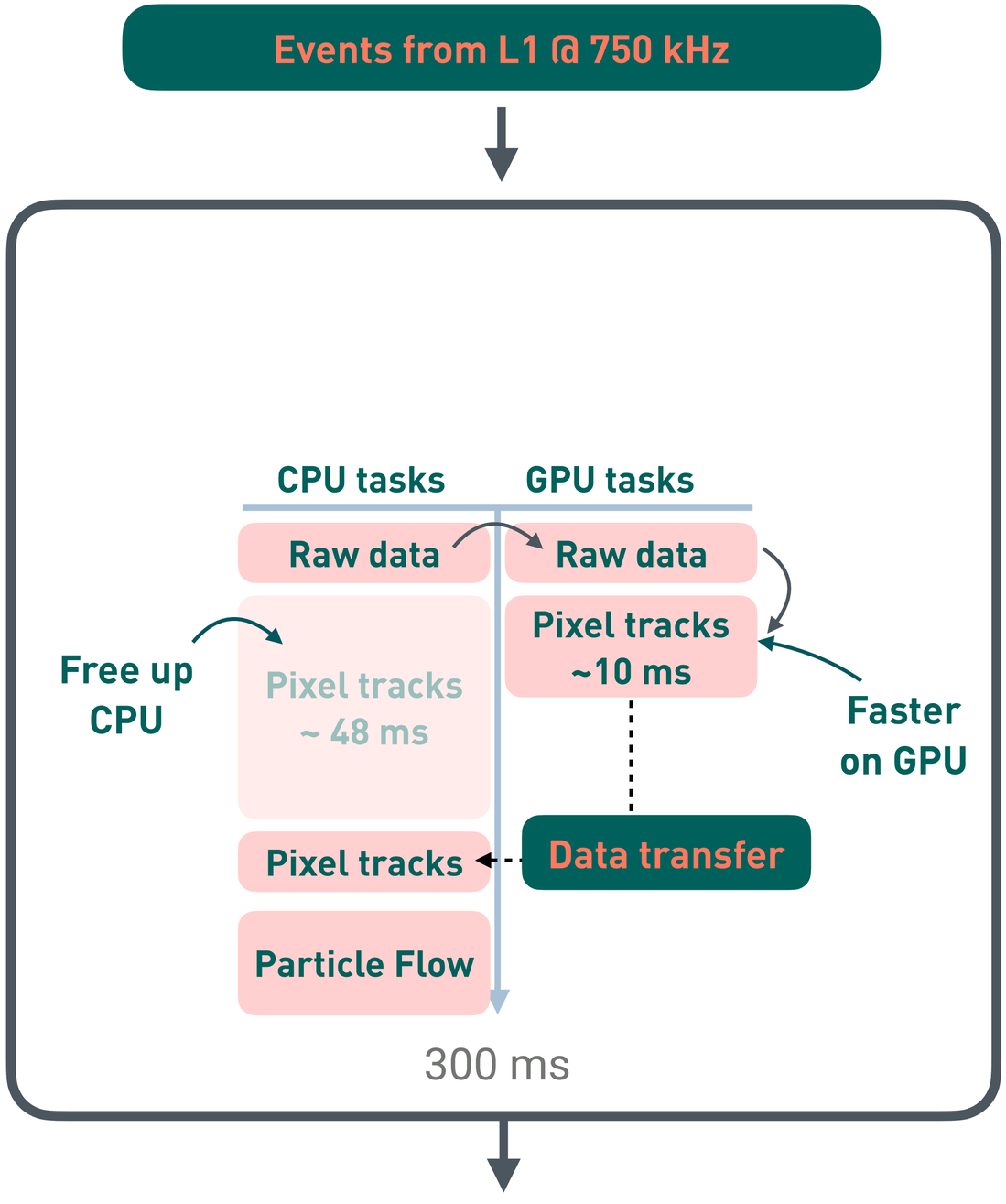
To handle HL-LHC data rates

- Offload resource-intensive computations to GPU



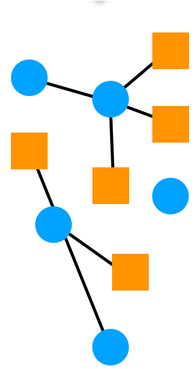
AI for fast reconstruction



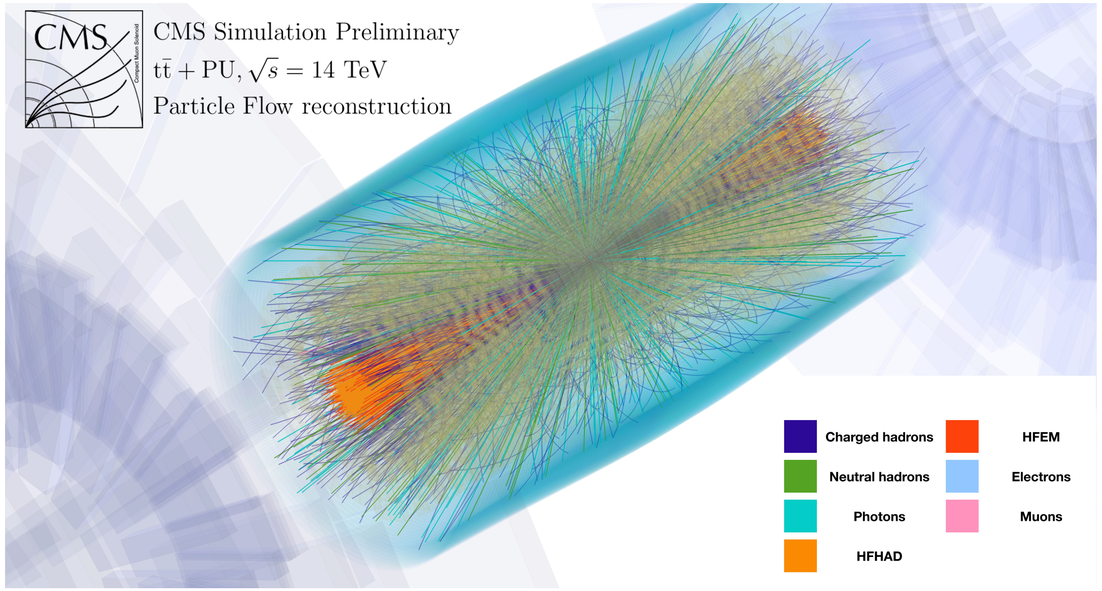


AI for fast reconstruction

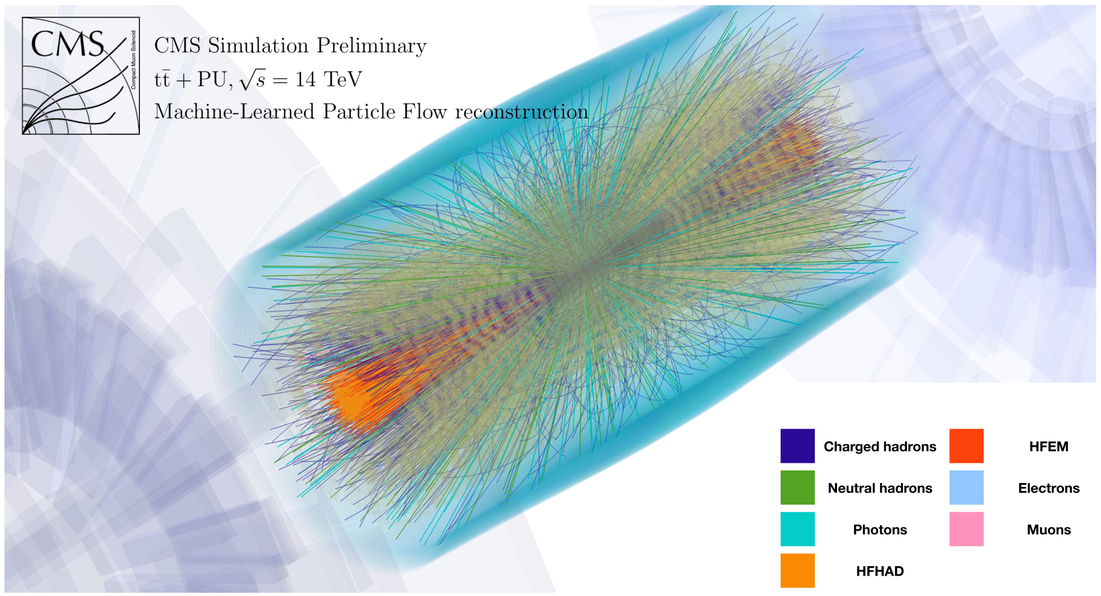
Graph Deep Neural Networks:“fast” approximations of ParticleFlow



Classical Particle Flow

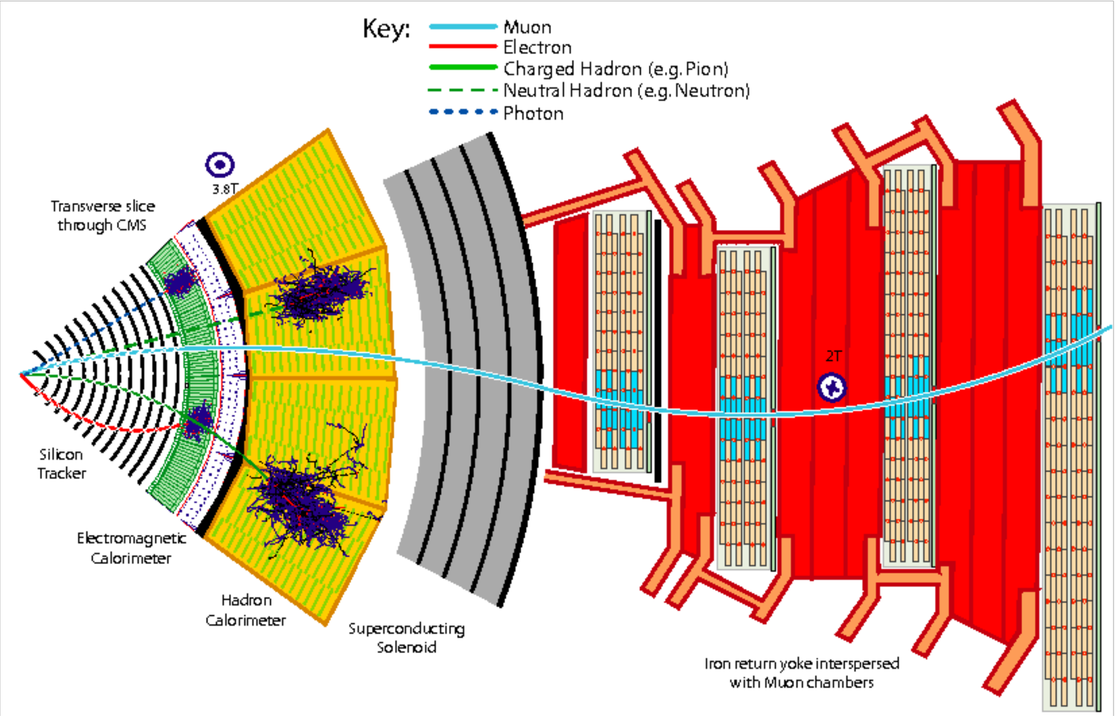


Graph Neural Network

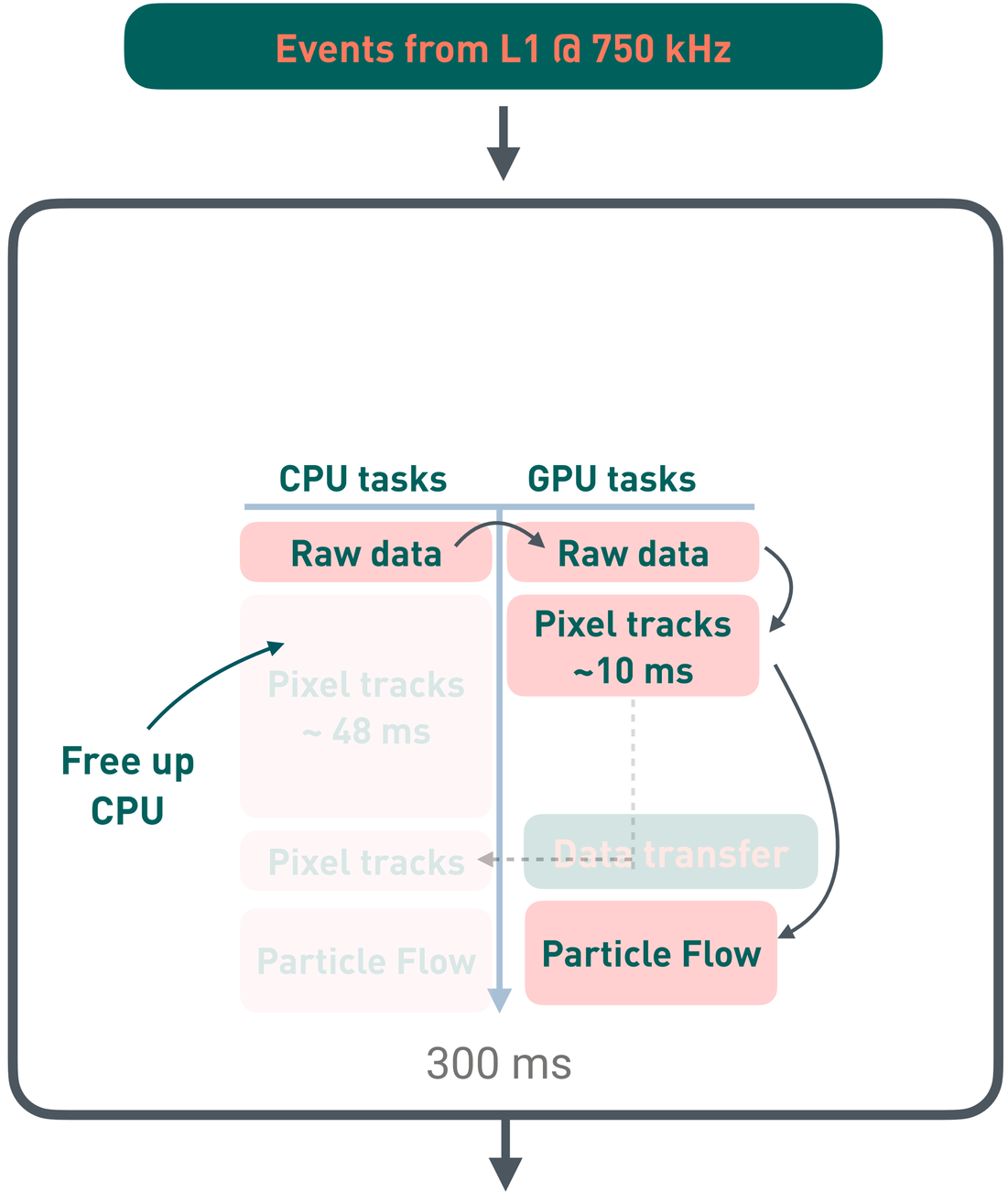


arXiv:2101.08578

AI for fast reconstruction

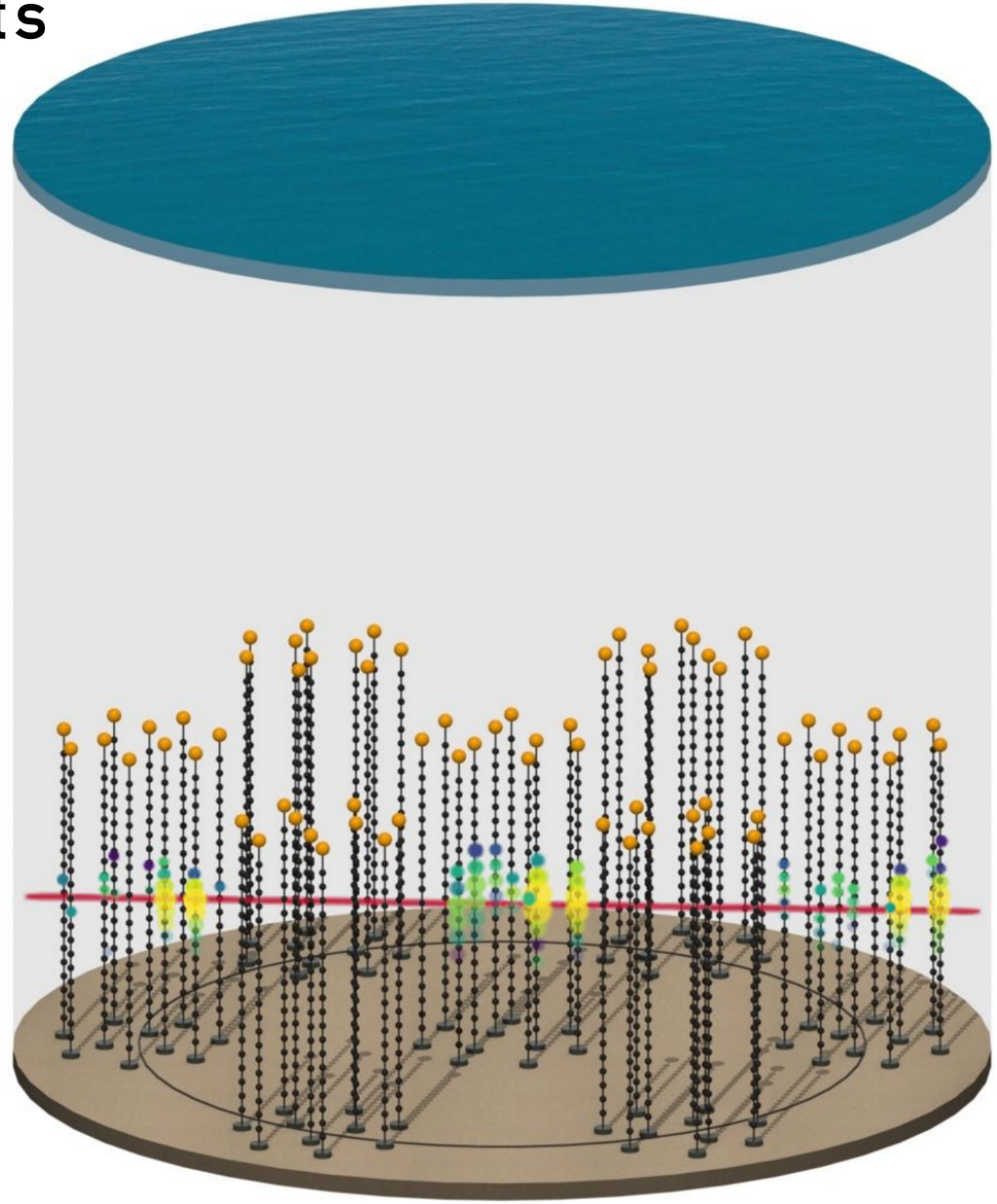


Bandwidth (kB/s) = Event rate (kHz) x Event size



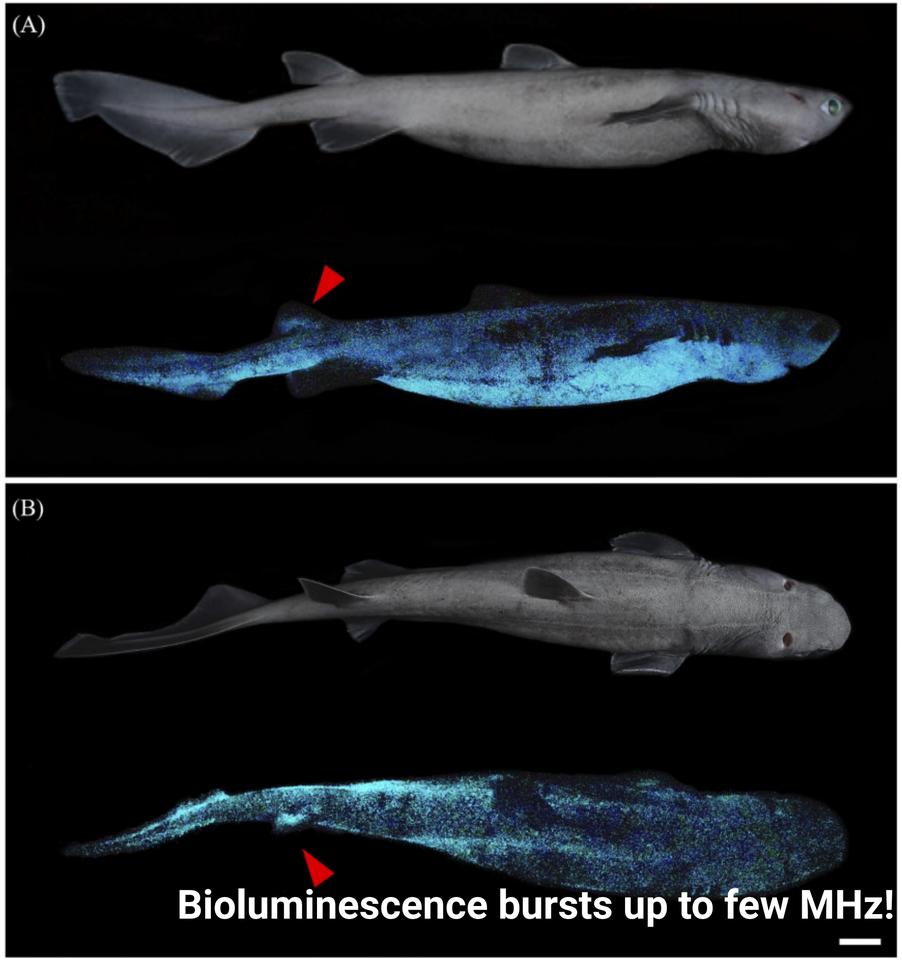
F. Capel et al.

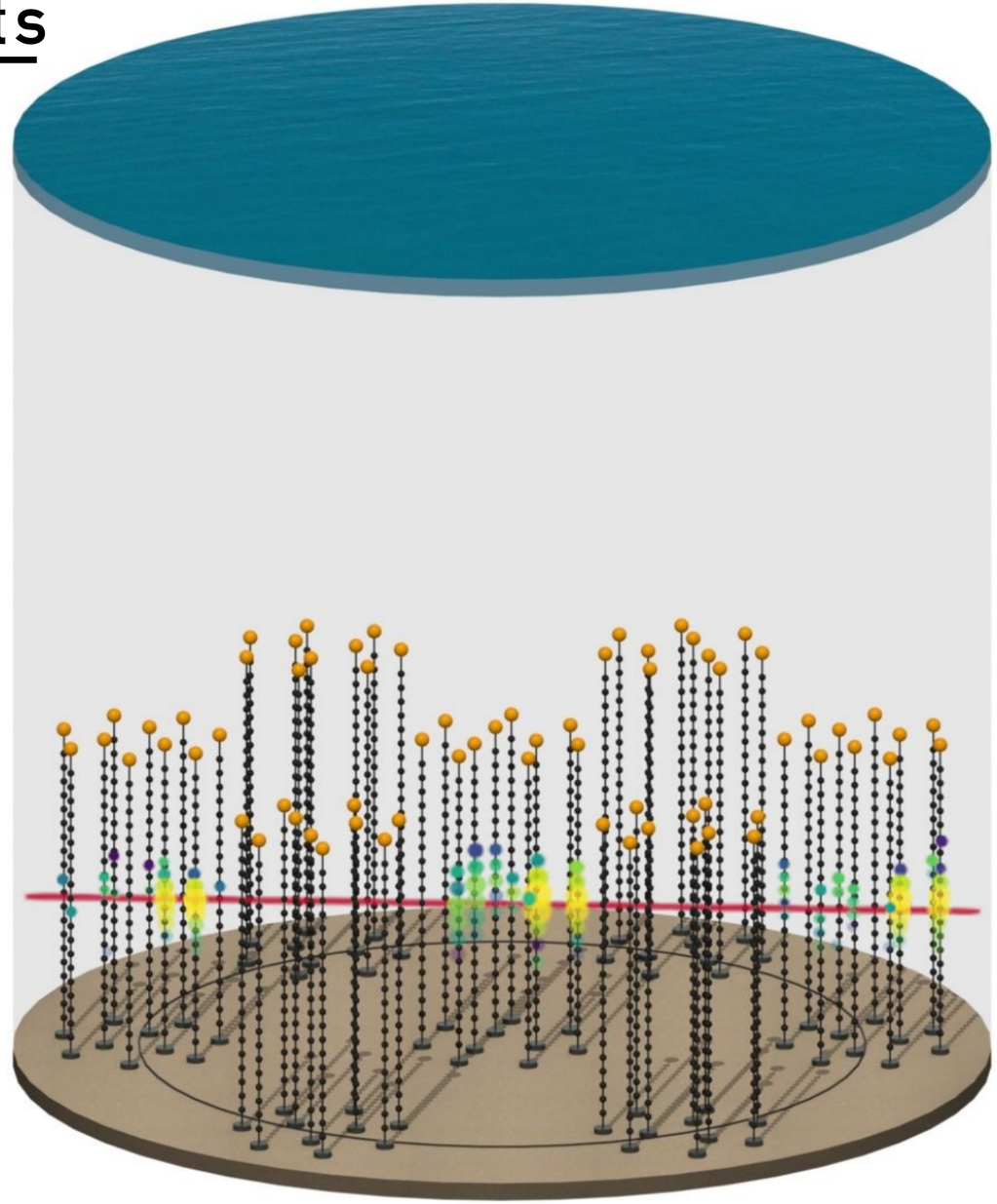
Real-time ML in other experiments



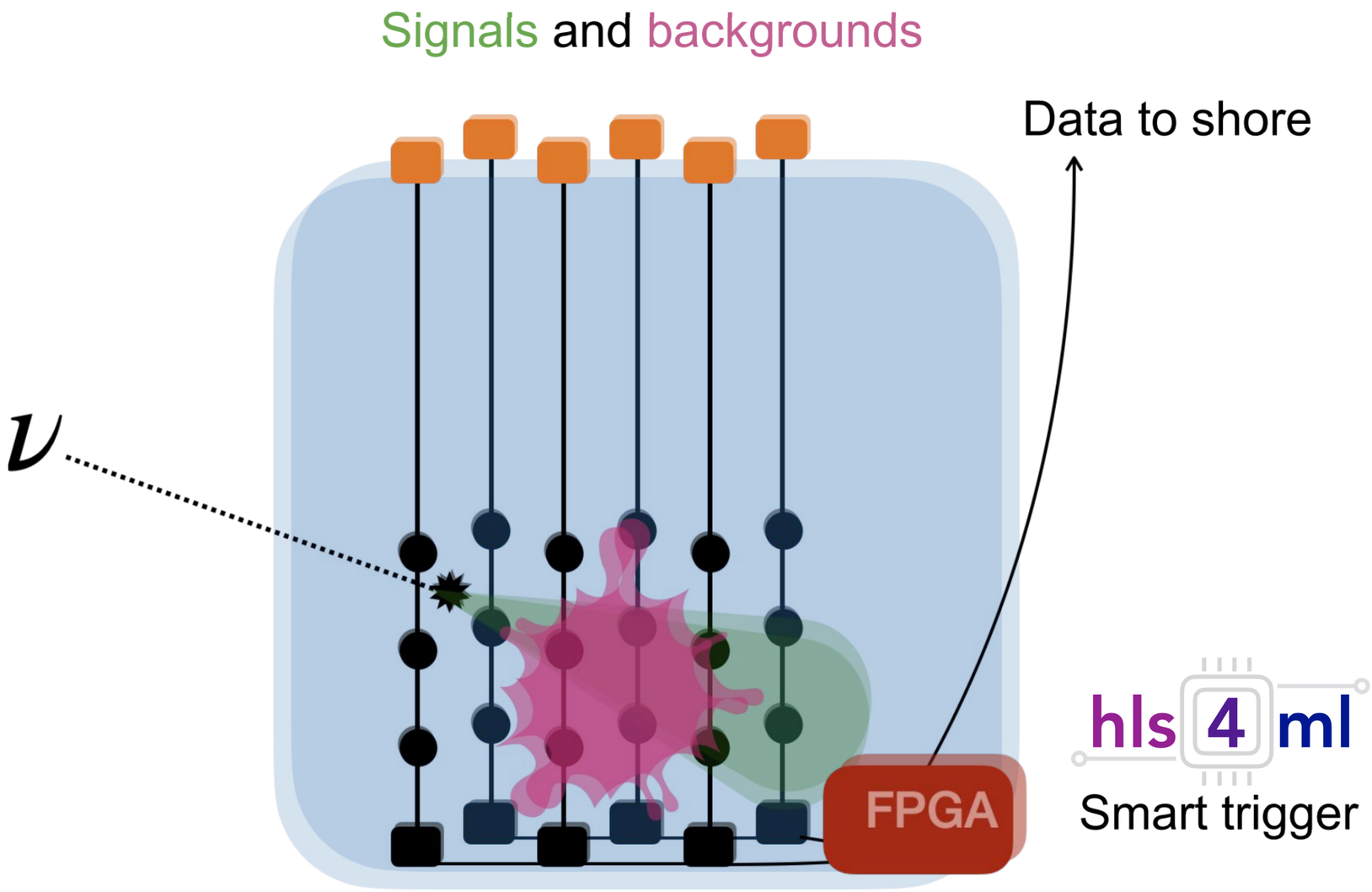
F. Capel et al.

Real-time ML in other experiments





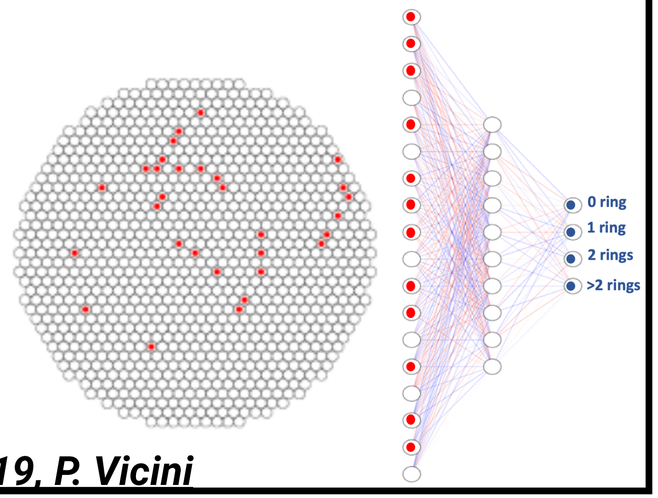
Real-time ML in other experiments



hls4ml in other CERN experiments

NA62: Measuring ultra rare kaon decays

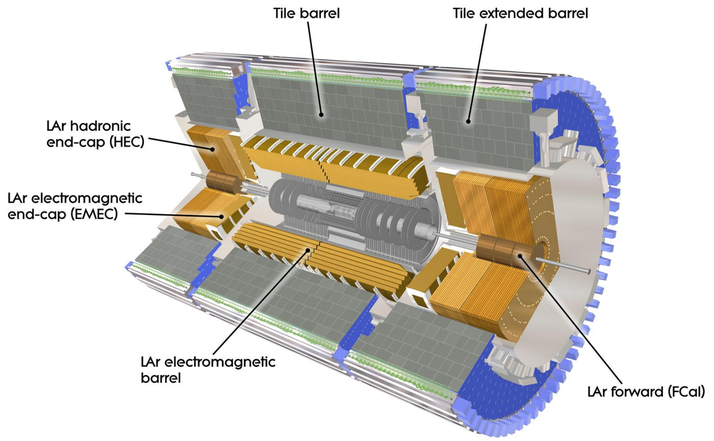
- DNN for RICH rings identification
- Latency: 40 cycles at 100 MHz



CHEP 2019, P. Vicini

ATLAS Liquid Argon Calorimeter

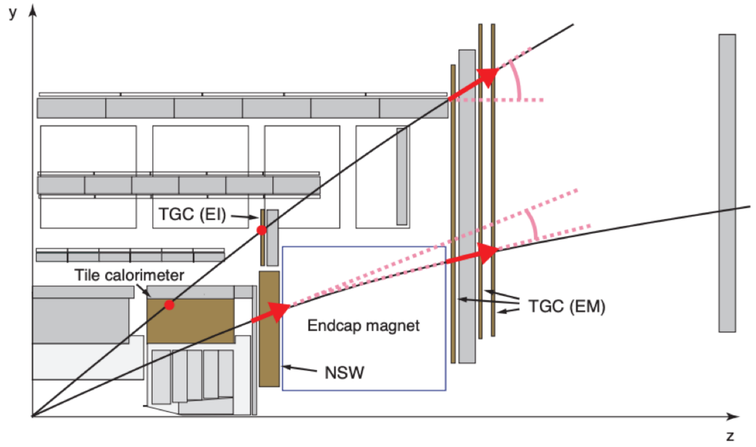
- RNN for real-time energy reconstructions
- Latency ~200 ns



DOI:10.1007/s41781-021-00066-y

Muon segment finding and reconstruction

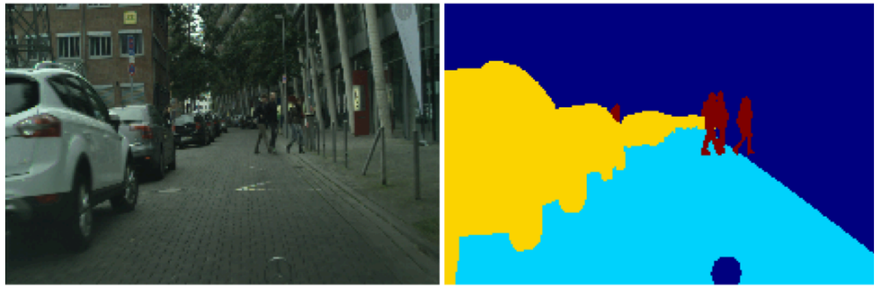
- Regression of muon position and angle
- 400 ns budget



R. Teixeira de Lima, R Rojas Caballero et al.

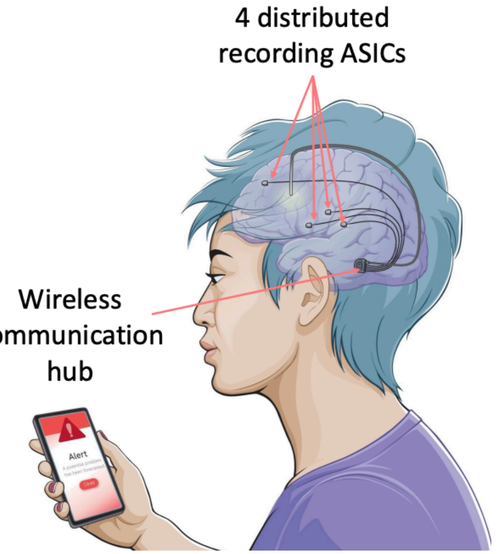
...and outside of HEP

Semantic segmentation for autonomous vehicles



N. Ghielmetti et al.

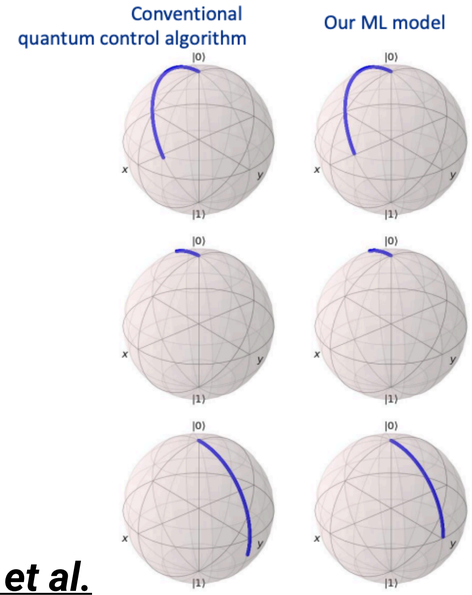
Seizure Predicting Brain Implant



W. Lemaire et al.

NN accelerator for quantum control

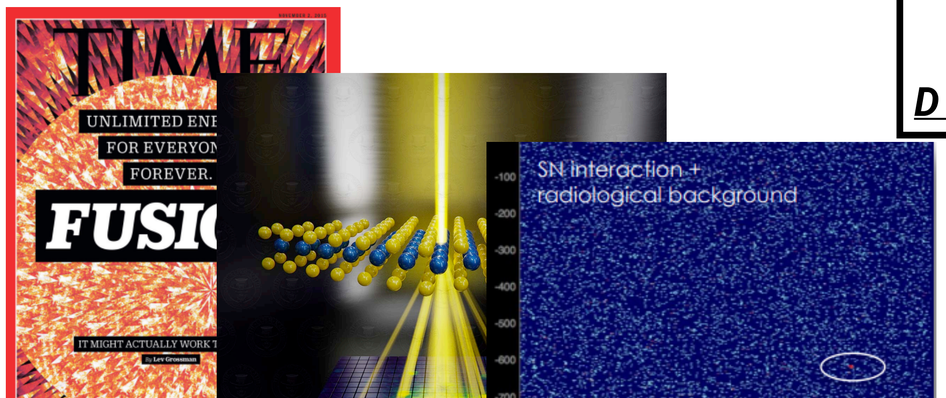
- Putting control in cryostat
  (e.g optimal pulse parameters)

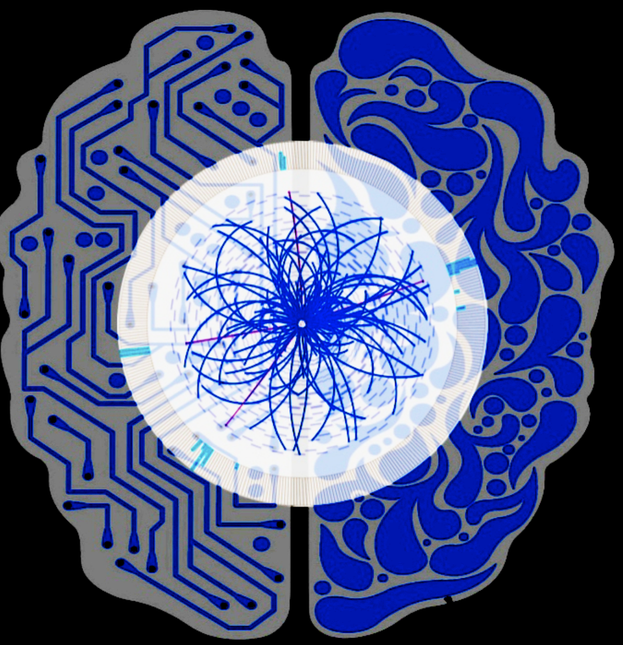


D Xu et al.

Other examples

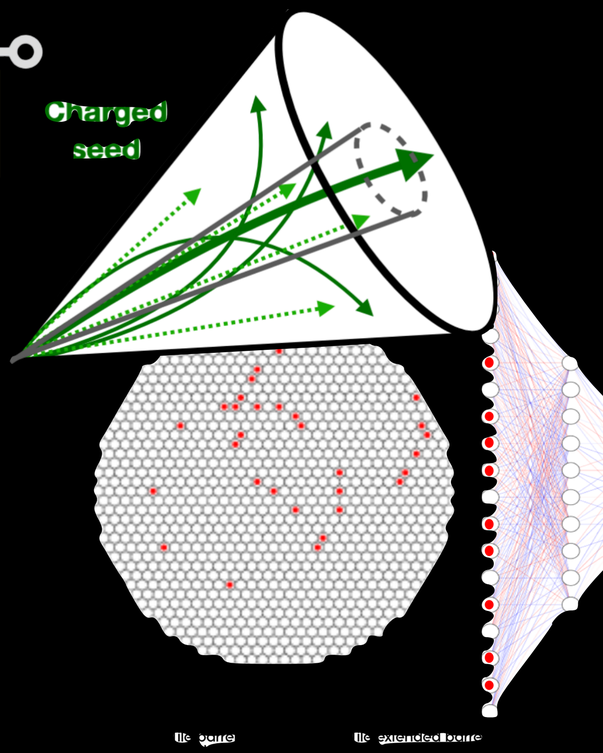
- ***For fusion science phase/mode monitoring***
- ***Crystal structure detection***
- ***Triggering in DUNE***
- ***Accelerator control***
- ***Magnet Quench Detection***
- ***MLPerf tinyML benchmarking***
- ***Food contamination detection***
- etc....



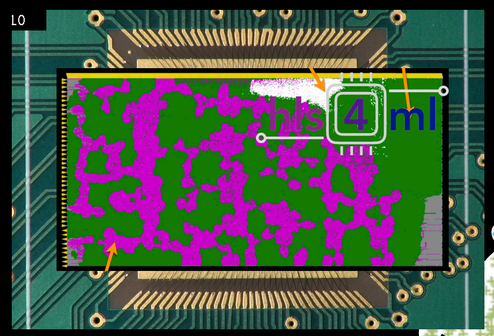


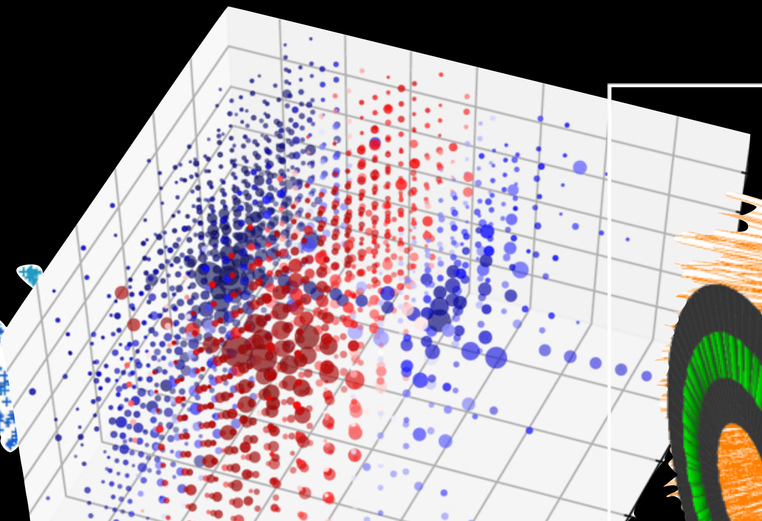
Conifer

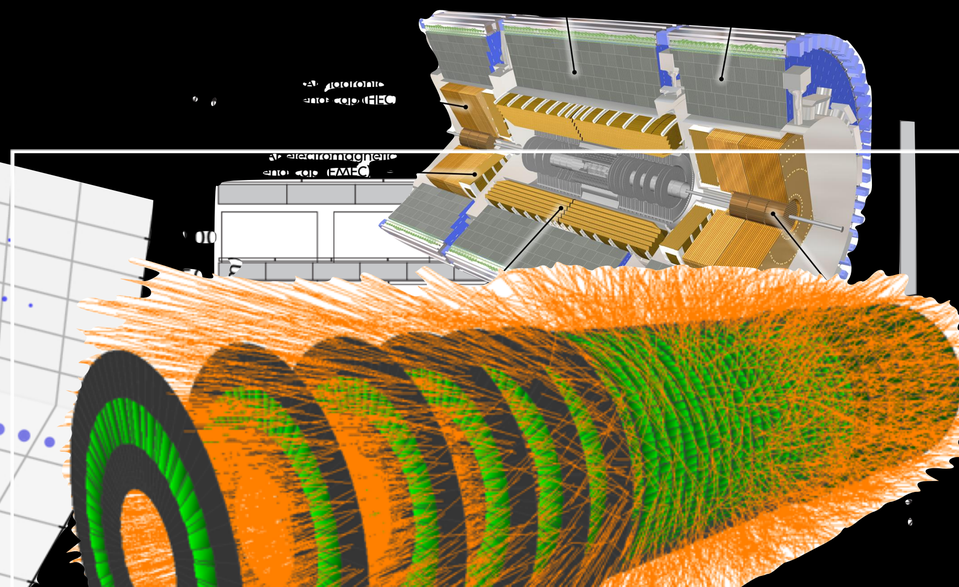
hls4ml



Join the community:
fastmachinelearning.org
Sign up to the hls-fml group







Backup

ADC 2021

Data challenge on real-time anomaly detection

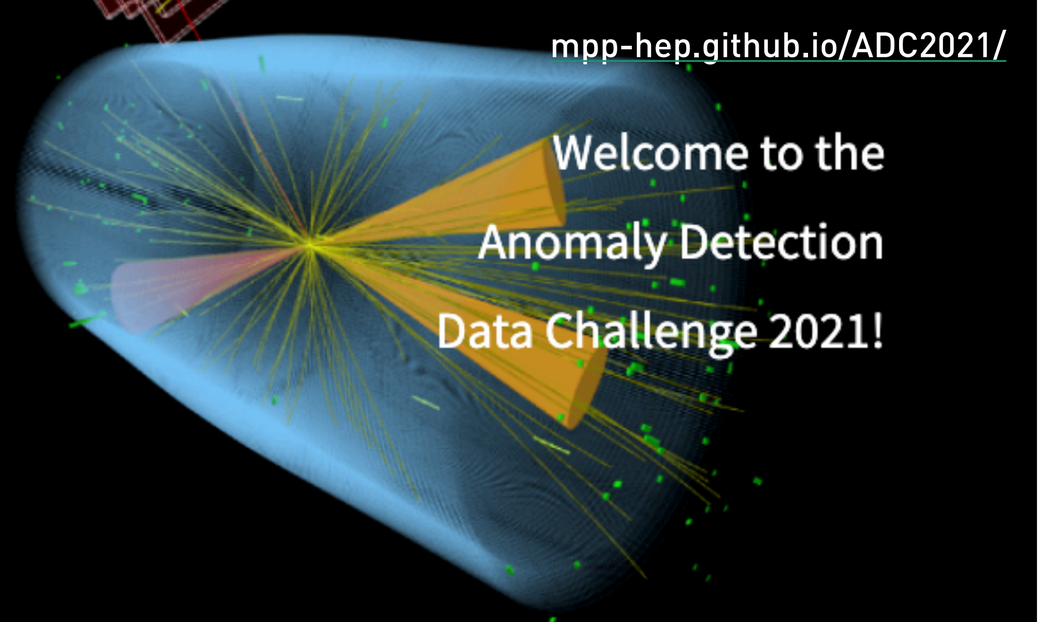
- Dataset: Nature Scientific Data (2022) 9:118
- Deadline: November 2022

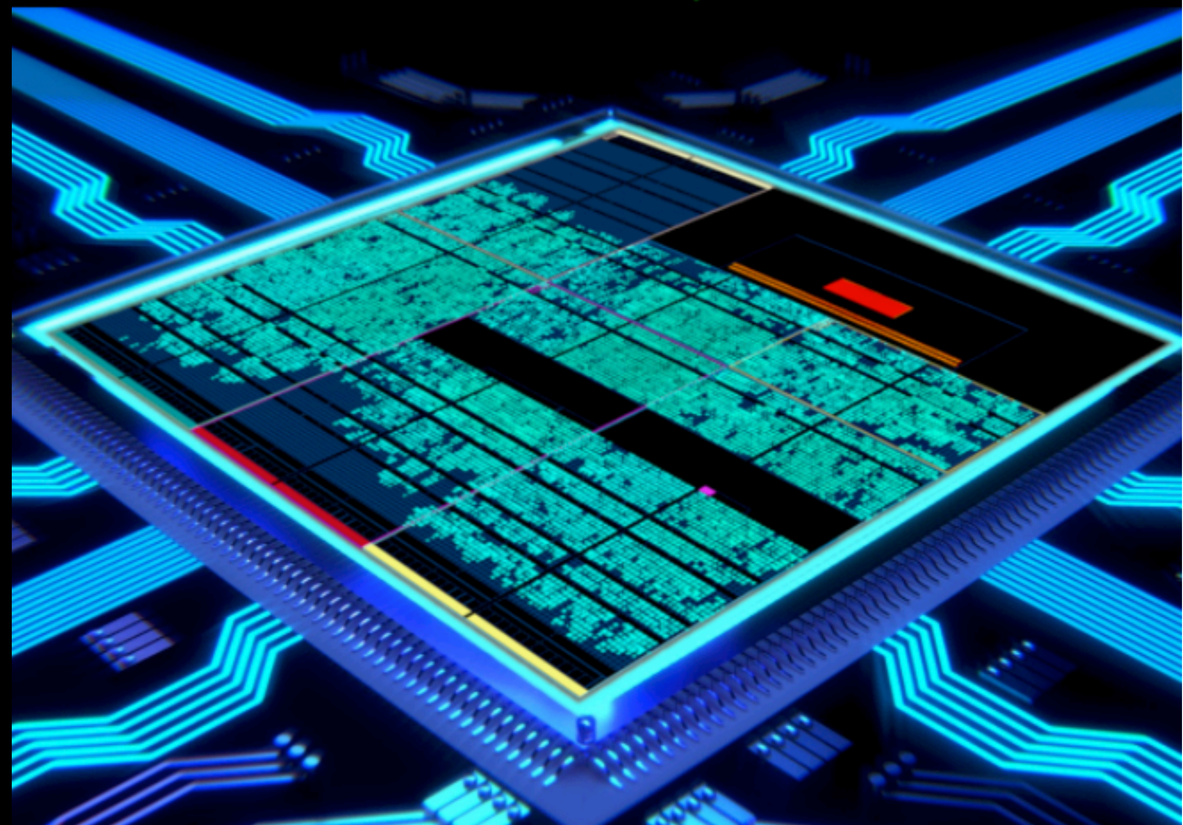
Tutorial: Anomaly detection on FPGA with hls4ml

- github:thaarres/quantumUniverse_pynqZ2

Help us find new physics!

mpp-hep.github.io/ADC2021/





K + TensorFlow

ONNX

PYTORCH

Model (quantized/pruned)

Quantized:

QONNX

QKERAS

hls4ml

Convert model to internal representation

Write HLS project targeting specified backend (configurable parallelization/ quantization)

Run emulation

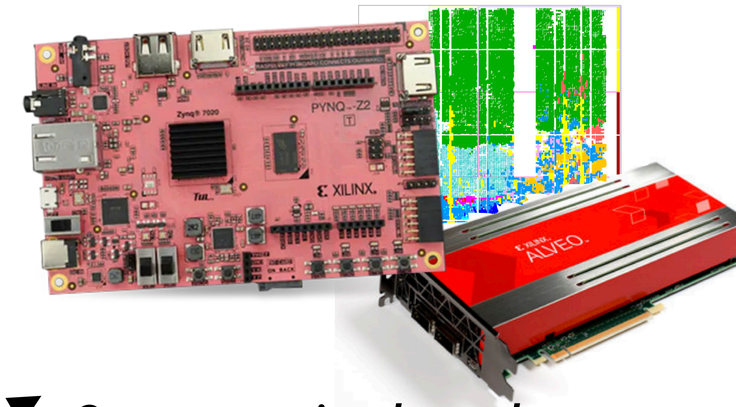
Run synthesis

Vivado™ HLS

Mentor® A Siemens Business

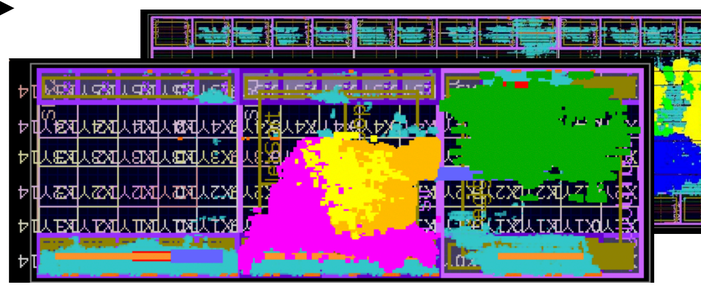
Catapult (soon!)

intel HLS COMPILER

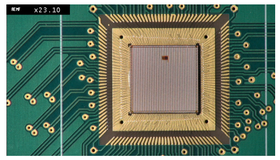


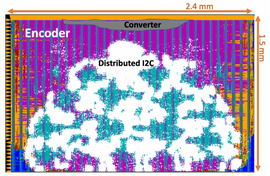
Co-processing kernel (Xilinx accelerators/SoCs)

FPGA custom designs (eg trigger algorithms)



ASICs

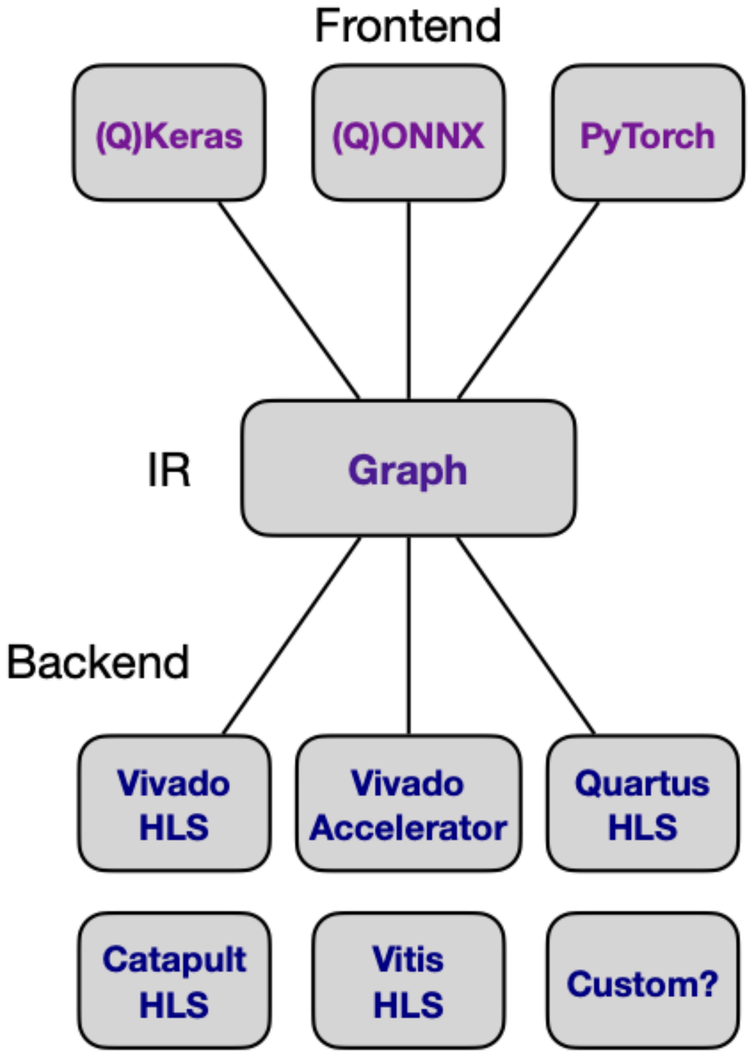




hls4ml:

Very broad frontend (ML library) and backend (HLS compiler) to cover most use cases!

Should be a library all experiments can use!



Conv2D

ReLU

MaxPool2D

Conv2D

ReLU

MaxPool2D

Flatten

Dense

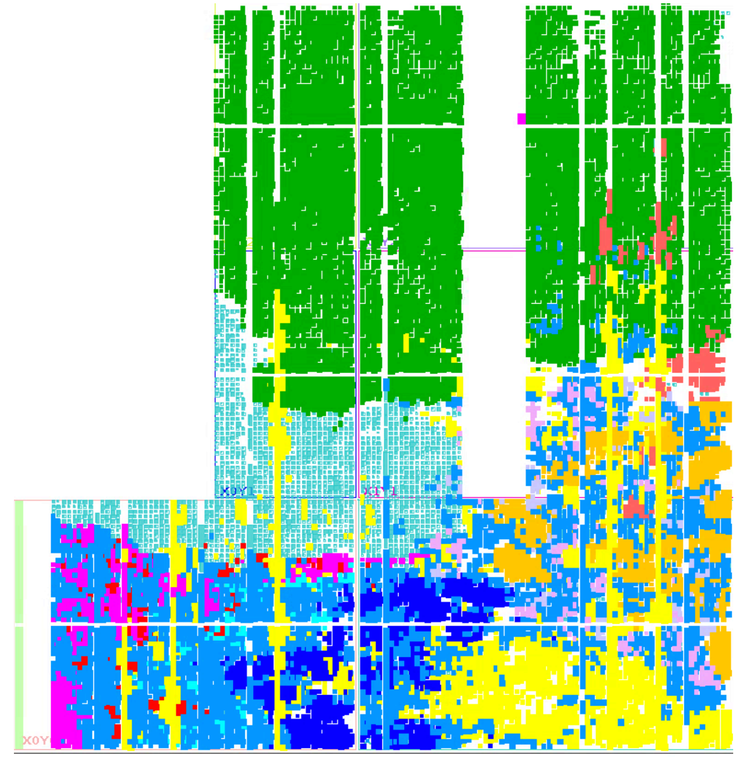
Softmax

Prediction

hls4ml

```
from hls4ml import …
import tensorflow as tf

# train or load a model
model = … # e.g. tf.keras.models.load_model(…)

# make a config template
cfg = config_from_keras_model(model,
granularity='name')

# tune the config
cfg['LayerName']['layer2']['ReuseFactor'] = 4

# do the conversion
hmodel = convert_from_keras_model(model, cfg)

# write and compile the HLS
hmodel.compile()

# run bit accurate emulation
y_tf = model.predict(x)
y_hls = hmodel.predict(x)

# do some validation
np.testing.assert_allclose(y_tf, y_hls)

# run HLS synthesis
hmodel.build()
```

pynq-z2 floorplan



(from Sioni S Summers)

Compression

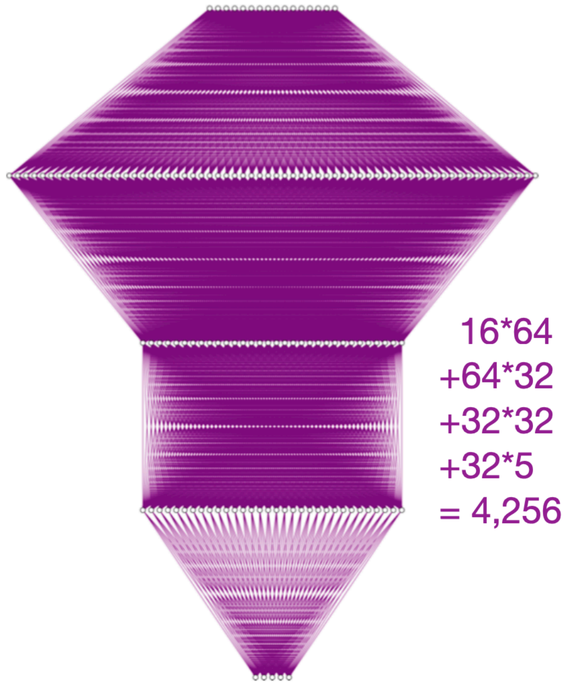
16 inputs

64 nodes
activation: ReLU

32 nodes
activation: ReLU

32 nodes
activation: ReLU

5 outputs
activation: SoftMax



16*64
+64*32
+32*32
+32*5
= 4,256

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

activation function — precomputed and stored in BRAMs

multiplication — DSPs

addition — logic cells

Network size limited by N multiplications

- E.g, simple dense network, total multiplications: 4256!
- A typical FPGA at LHC usually has 4-6000 DSPs
- Can your network fit within the resources?

Efficient NN design for FPGAs (and other edge compute)

Before deploying any DNN on chip (CMS trigger, iPhone), must make it efficient!

- Big engineering field in its own right

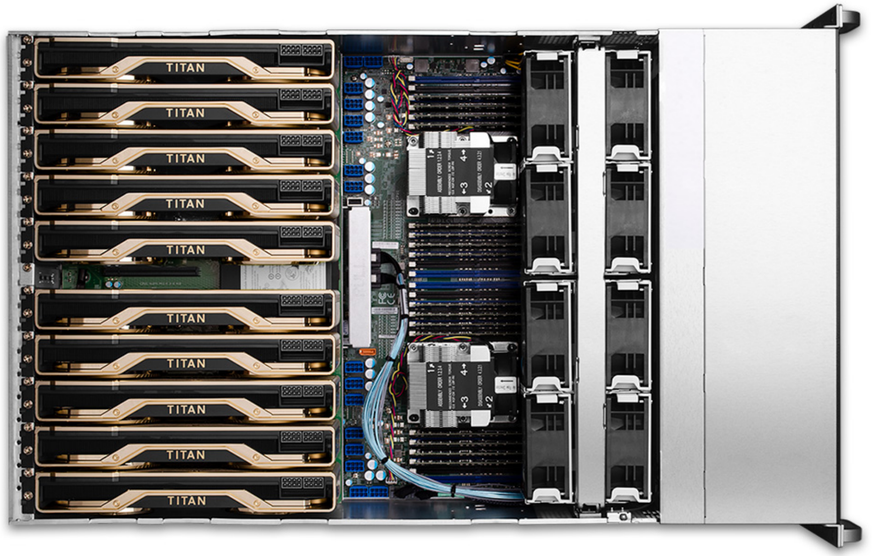
During training

- Quantization: do you really need 32-bit FP precision?
- Pruning: removal insignificant synapses

Post-training

- Parallelisation (lower latency ↔ more resources)

From 8 GPU server to tiny FPGA!



Quantization

Fixed point post-training quantization

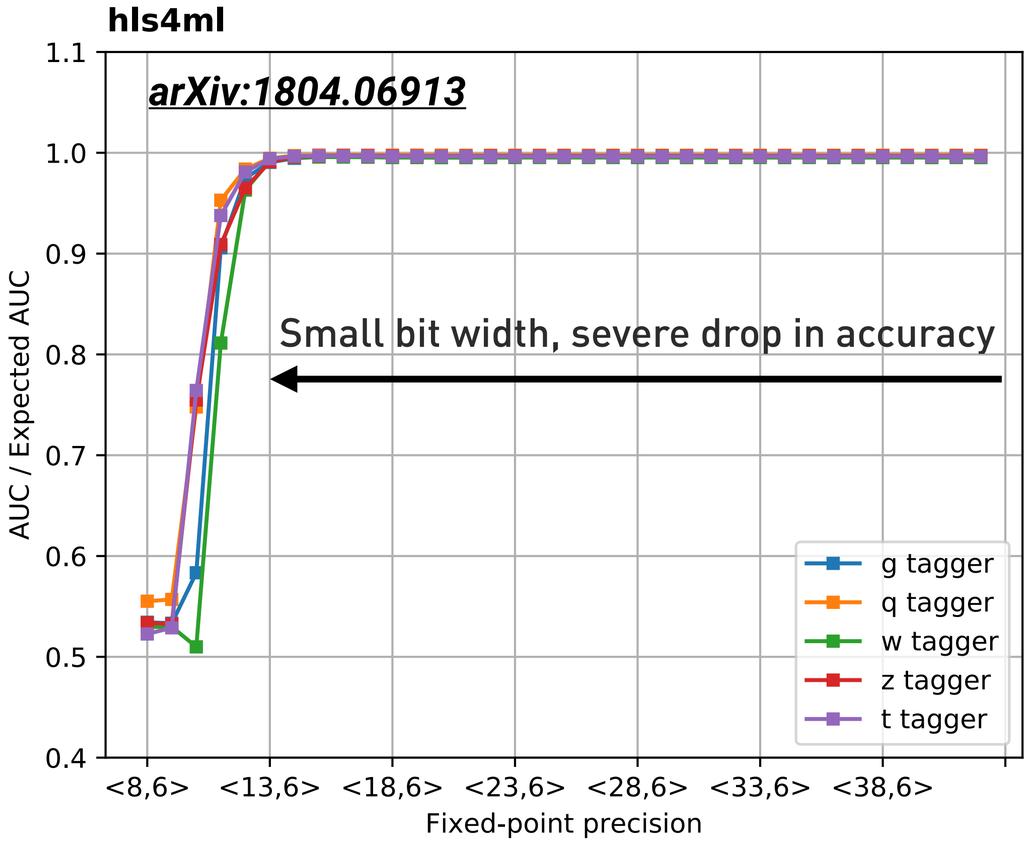
- FP 32 arithmetic ~ **x3-5** more resources,
  x2 higher latency than fixed-point → convert to fixed-point

On hardware: ap_fixed (W,I)

⟨00011.01⟩

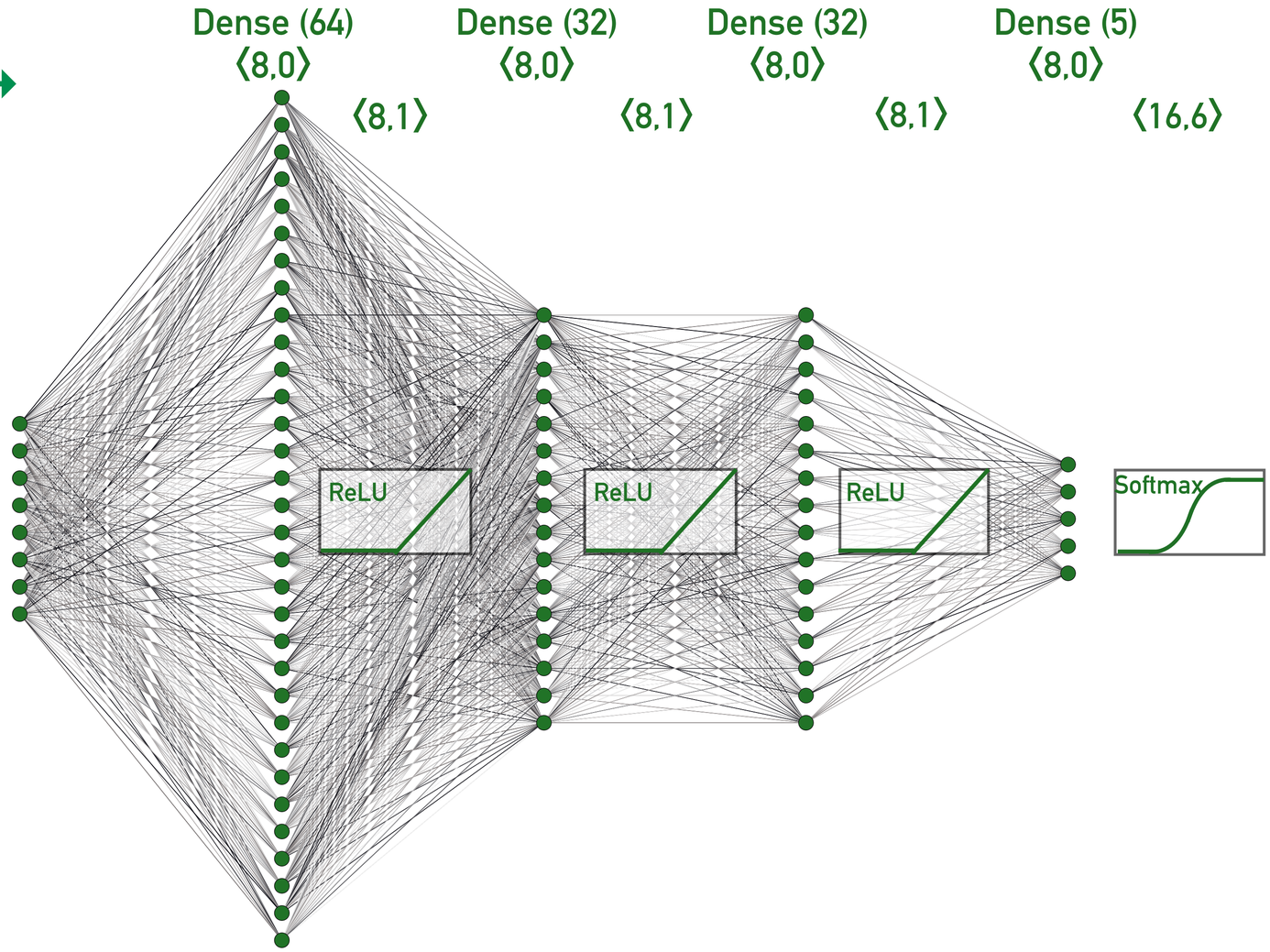
By definition lossy, precision must be tuned carefully (weights usually don't need large dynamic range. But, worse 'resolution')

Can we do better?



Quantization-aware training

Forward pass →

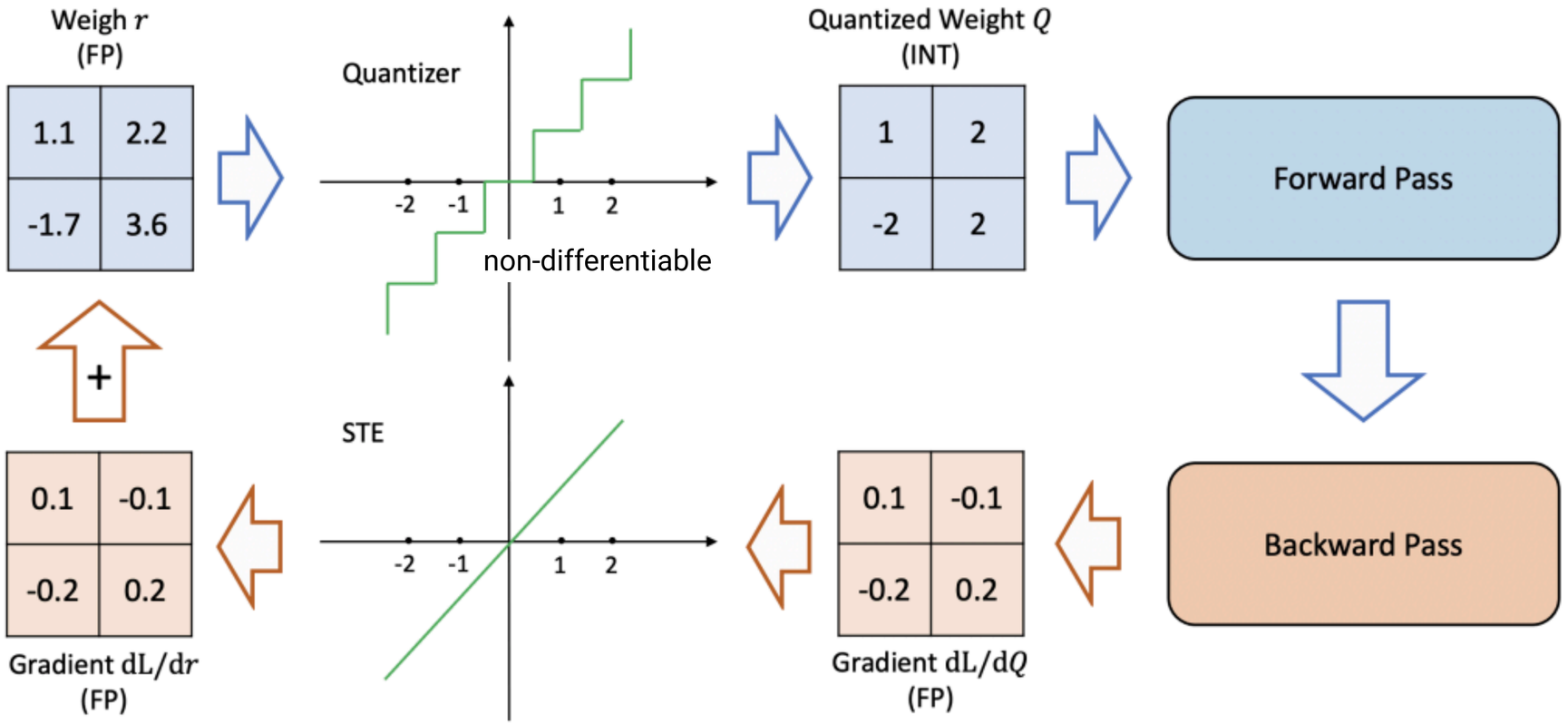


← Back propagation

FP 32 FP 32 FP 32 FP 32 FP 32 FP 32 FP 32 FP 32

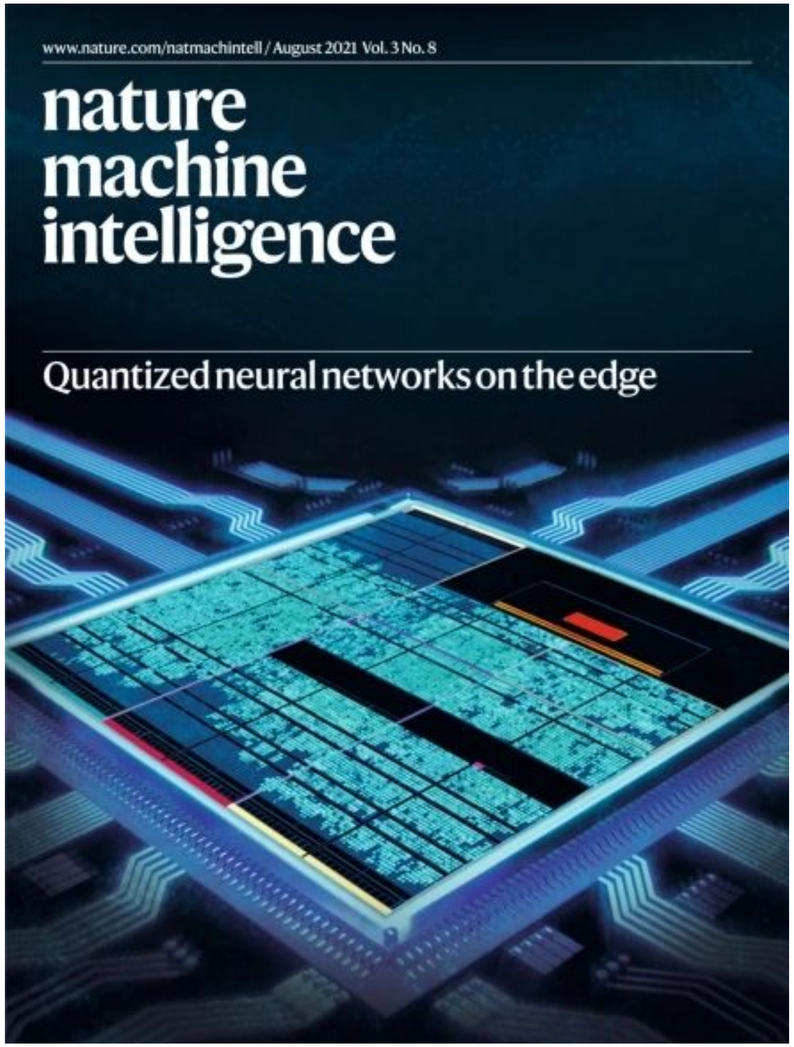
Quantization-aware training

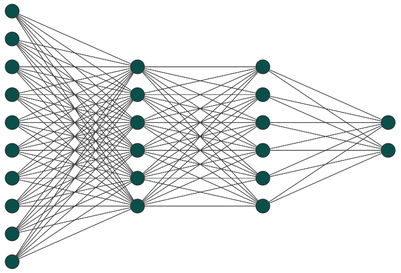
Lossless quantization for deep neural networks!



arxiv:2103.13630

Nature Machine Intelligence 3 (2021)





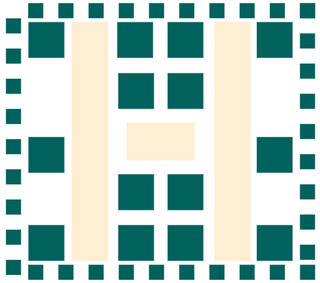
QKeras
Quantization-aware training

Google AI

QKeras
model

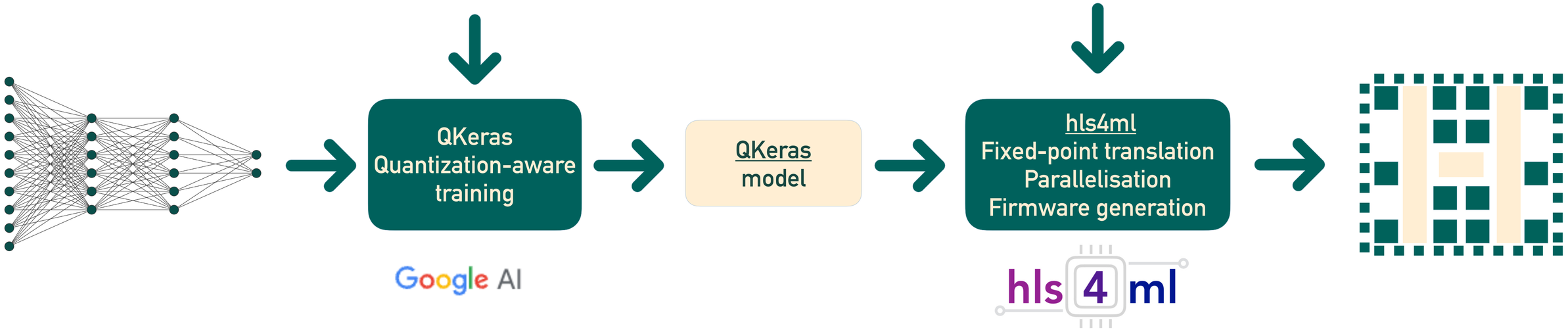
hls4ml
Fixed-point translation
Parallelisation
Firmware generation

hls 4 ml

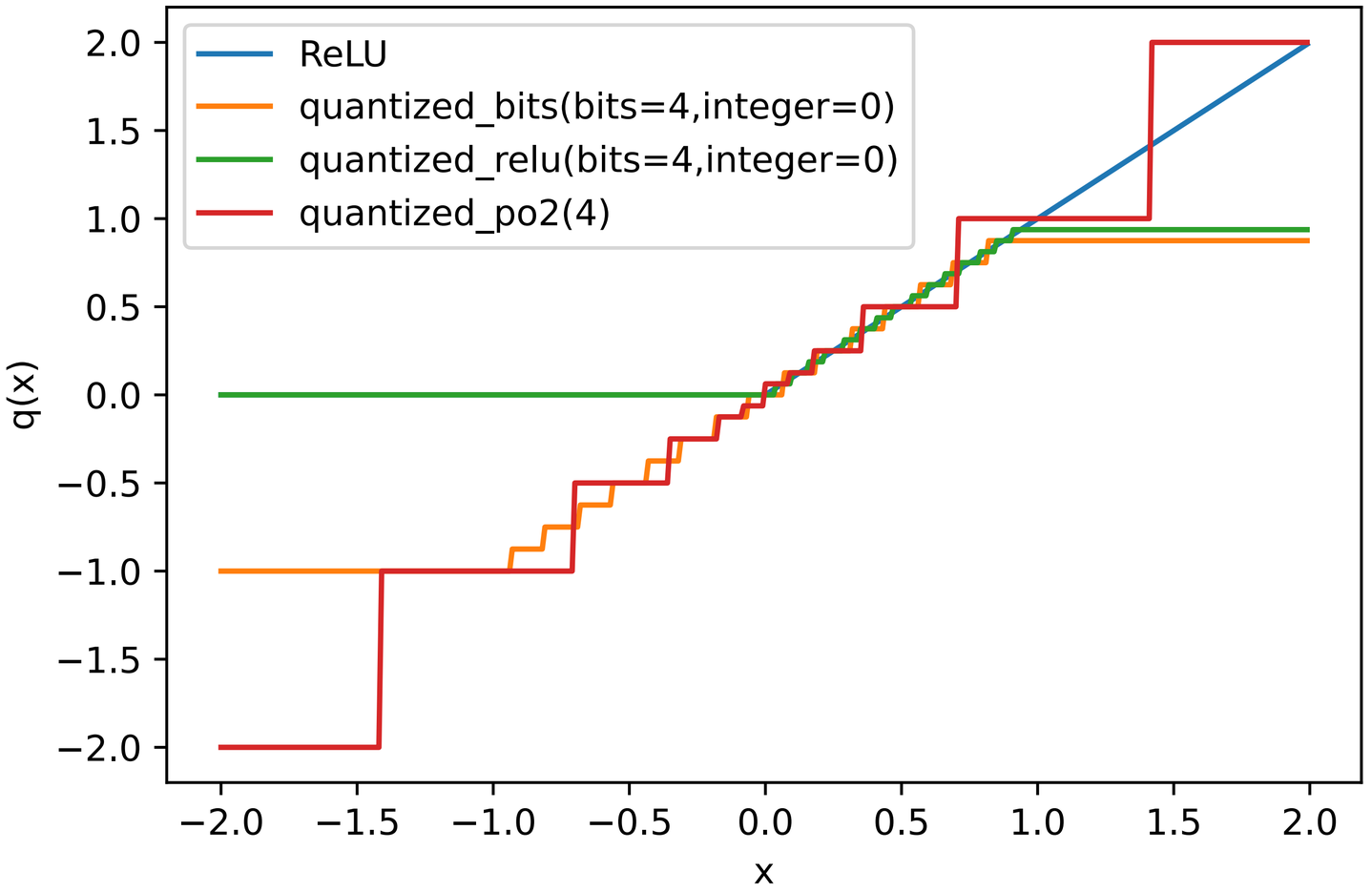


```
from tensorflow.keras.layers import Input, Activation
from qkeras import quantized_bits
from qkeras import QDense, QActivation
from qkeras import QBatchNormalization

x = Input((16))
x = QDense(64,
    kernel_quantizer = quantized_bits(6,0,alpha=1),
    bias_quantizer   = quantized_bits(6,0,alpha=1))(x)
x = QBatchNormalization()(x)
x = QActivation('quantized_relu(6,0)')(x)
x = QDense(32,
    kernel_quantizer = quantized_bits(6,0,alpha=1),
    bias_quantizer   = quantized_bits(6,0,alpha=1))(x)
x = QBatchNormalization()(x)
x = QActivation('quantized_relu(6,0)')(x)
x = QDense(32,
    kernel_quantizer = quantized_bits(6,0,alpha=1),
    bias_quantizer   = quantized_bits(6,0,alpha=1))(x)
x = QBatchNormalization()(x)
x = QActivation('quantized_relu(6,0)')(x)
x = QDense(5,
    kernel_quantizer = quantized_bits(6,0,alpha=1),
    bias_quantizer   = quantized_bits(6,0,alpha=1))(x)
x = Activation('softmax')(x)
```

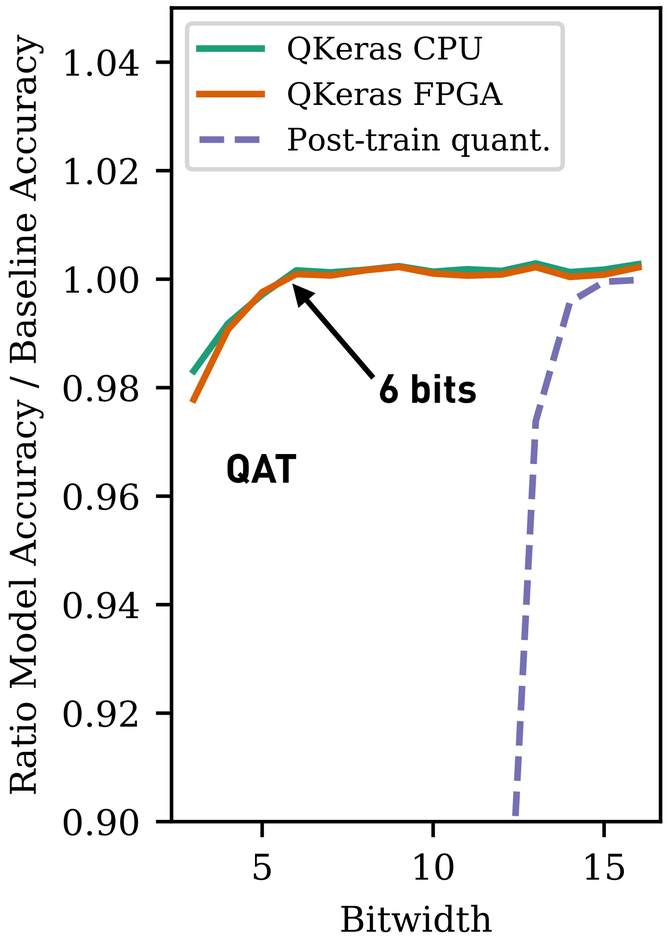
```
from hls4ml import …
import tensorflow as tf

# train or load a model
model = tf.keras.models.load_model(…)

# make a config
cfg = config_from_keras_model(model,
granularity='name')

# do the conversion
hmodel = convert_from_keras_model(model, cfg)

# write and compile the HLS
hmodel.compile()

# run HLS synthesis
hmodel.build()
```

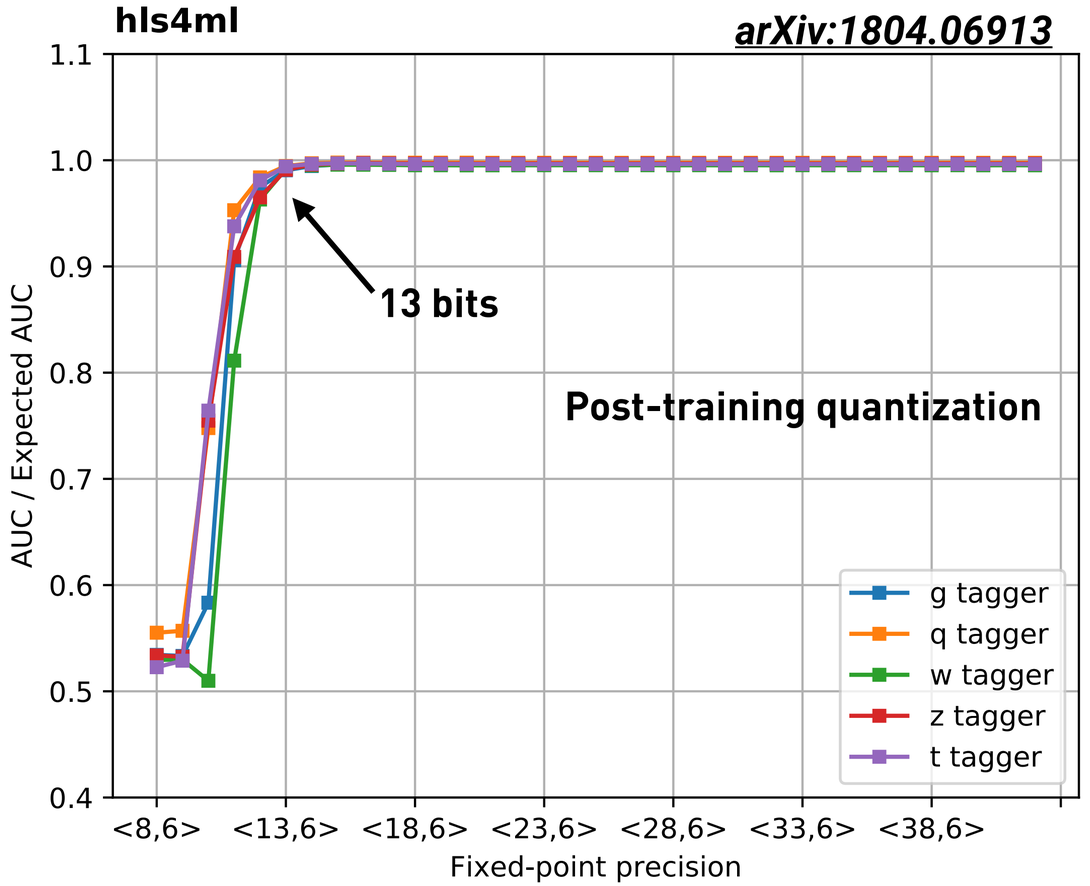


QKeras quantisers



FPGA performance

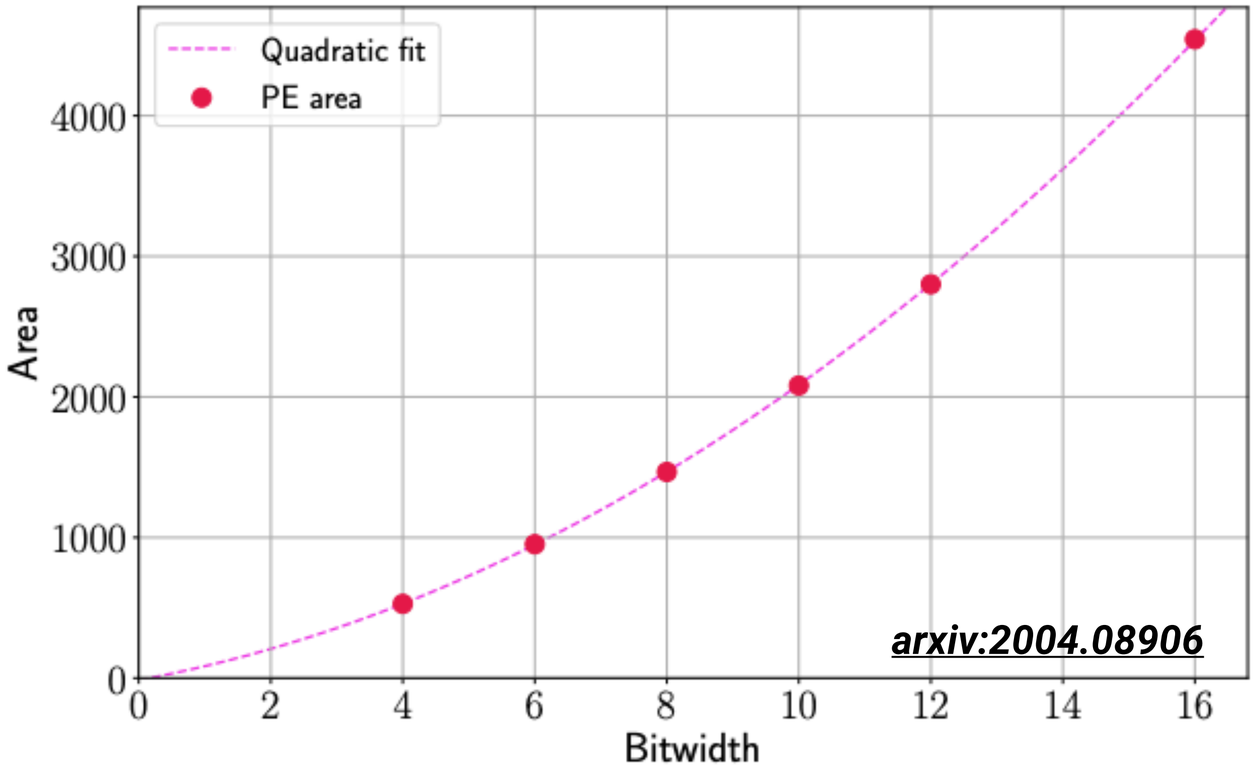




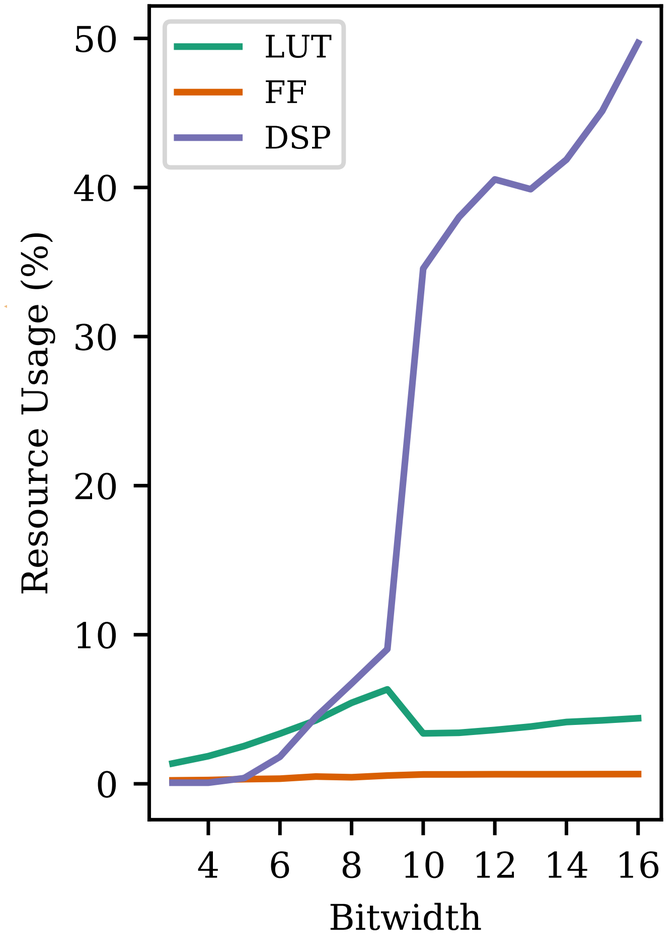
Nature Machine Intelligence 3 (2021)

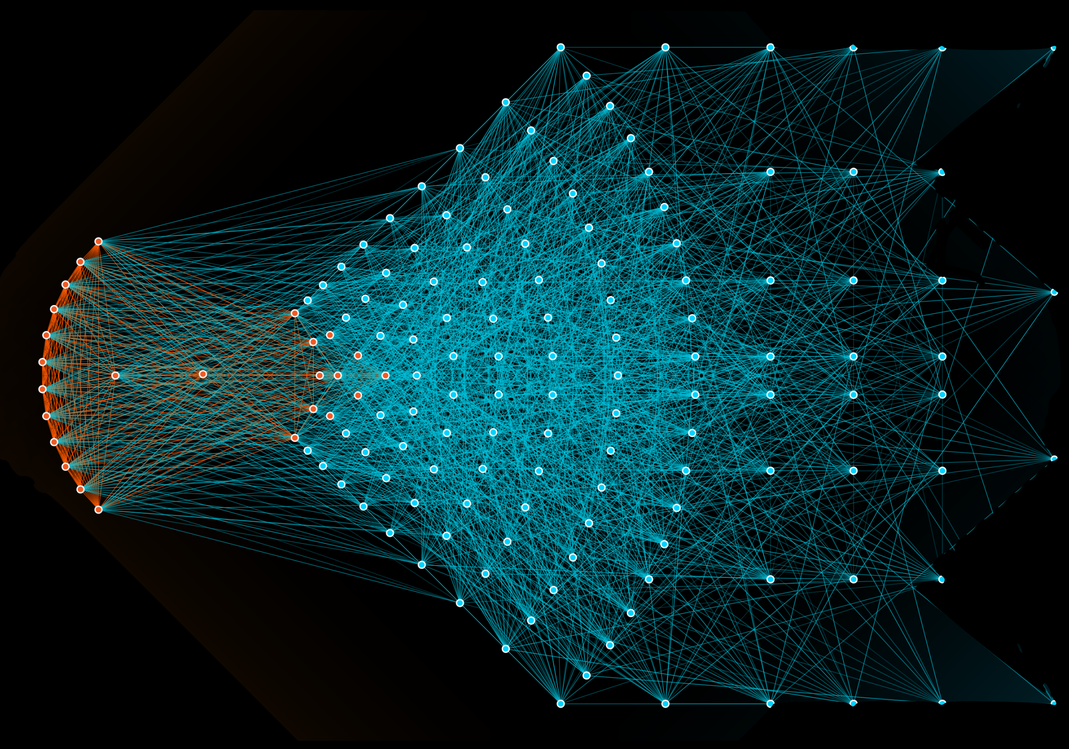
FPGA performance

Area/power scales quadratically with bitwidth!

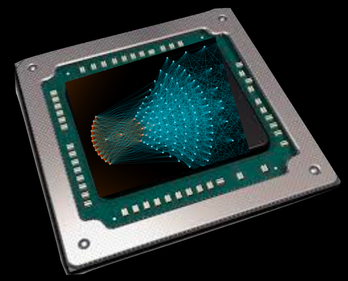


Nature Machine Intelligence 3 (2021)





Ideally



Reality

Estimating energy and size

Some layers more accommodating for aggressive quantization, others require expensive arithmetic

- heterogeneous quantization

Estimating energy and size

Some layers more accommodating for aggressive quantization, others require expensive arithmetic

- heterogeneous quantization

For edge inference, need best possible quantization configuration for

- Highest accuracy ↑…
- … and lowest resource consumption ↓

→ **hyper-parameter scan over quantizers which considers energy and accuracy simultaneously**

Estimating energy and size

Some layers more accommodating for aggressive quantization, others require expensive arithmetic

- heterogeneous quantization

For edge inference, need best possible quantization configuration for

- Highest accuracy ↑...
- ... and lowest resource consumption ↓

→ hyper-parameter scan over quantizers which considers energy and accuracy simultaneously

QTools: Estimate QKeras model bit and energy consumption, assuming 45 nm Horowitz process

- Relative model size in bits
- Relative energy consumption in Watts

Model	Accuracy [%]	Per-layer energy consumption [pJ]								Total energy [μJ]	Total bits
		Dense	ReLU	Dense	ReLU	Dense	ReLU	Dense	Softmax		
BF	74.4	1735	53	3240	27	1630	27	281	11	0.00700	61446
Q6	74.8	794	23	1120	11	562	11	99	11	0.00263	26334

$$\text{Forgiving Factor} = 1 + \Delta_{accuracy} \times \log_{rate}(S \times \frac{Cost_{ref}}{Cost_{trial}})$$

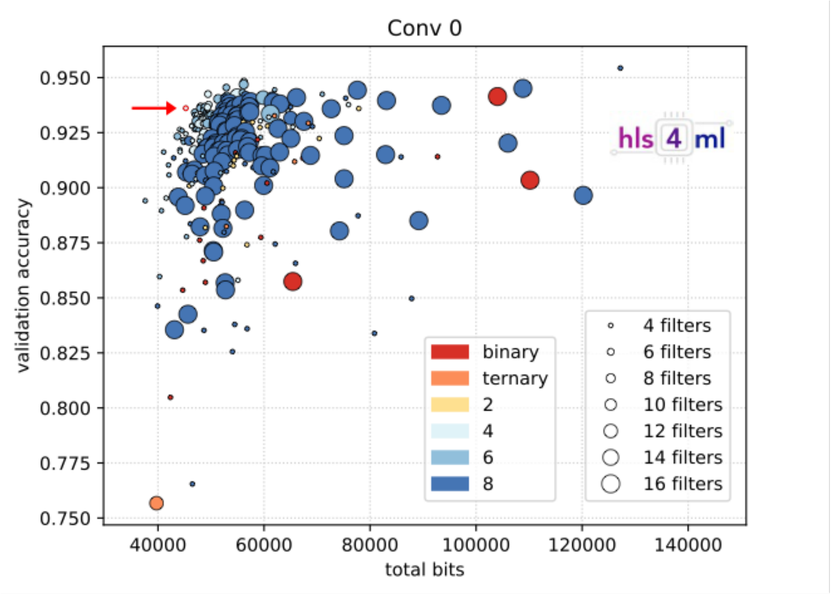
Maximize accuracy + minimizing cost in hyper parameter scan over quantizers: AutoQKeras

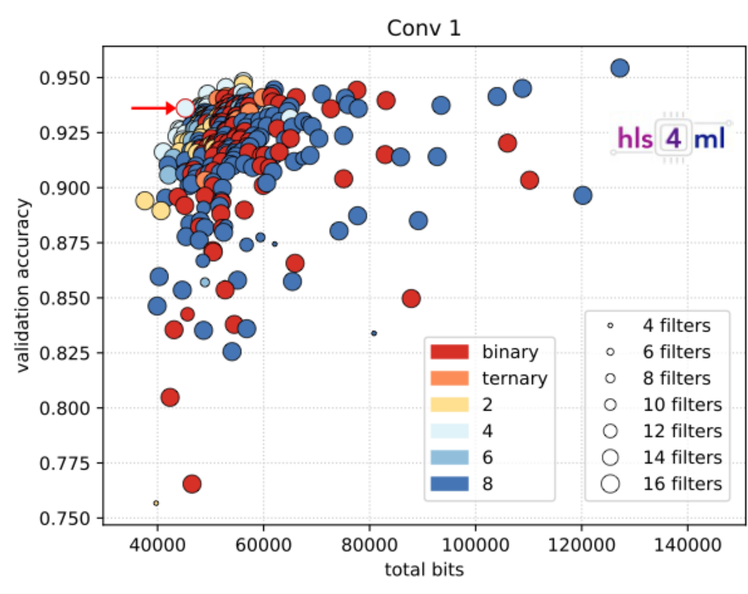
DOI 10.1088/2632-2153/ac0ea1

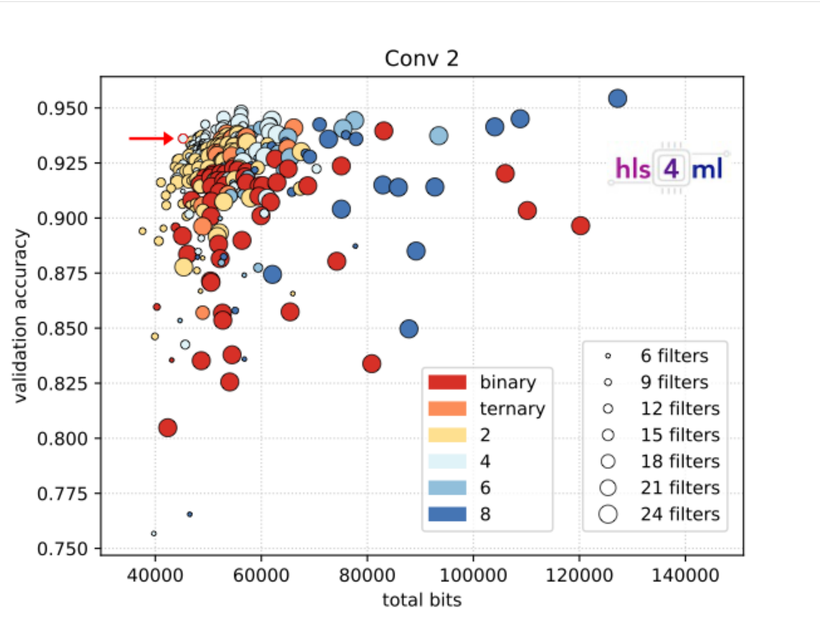
AutoQKeras

AutoQ Bayesian optimization at work!

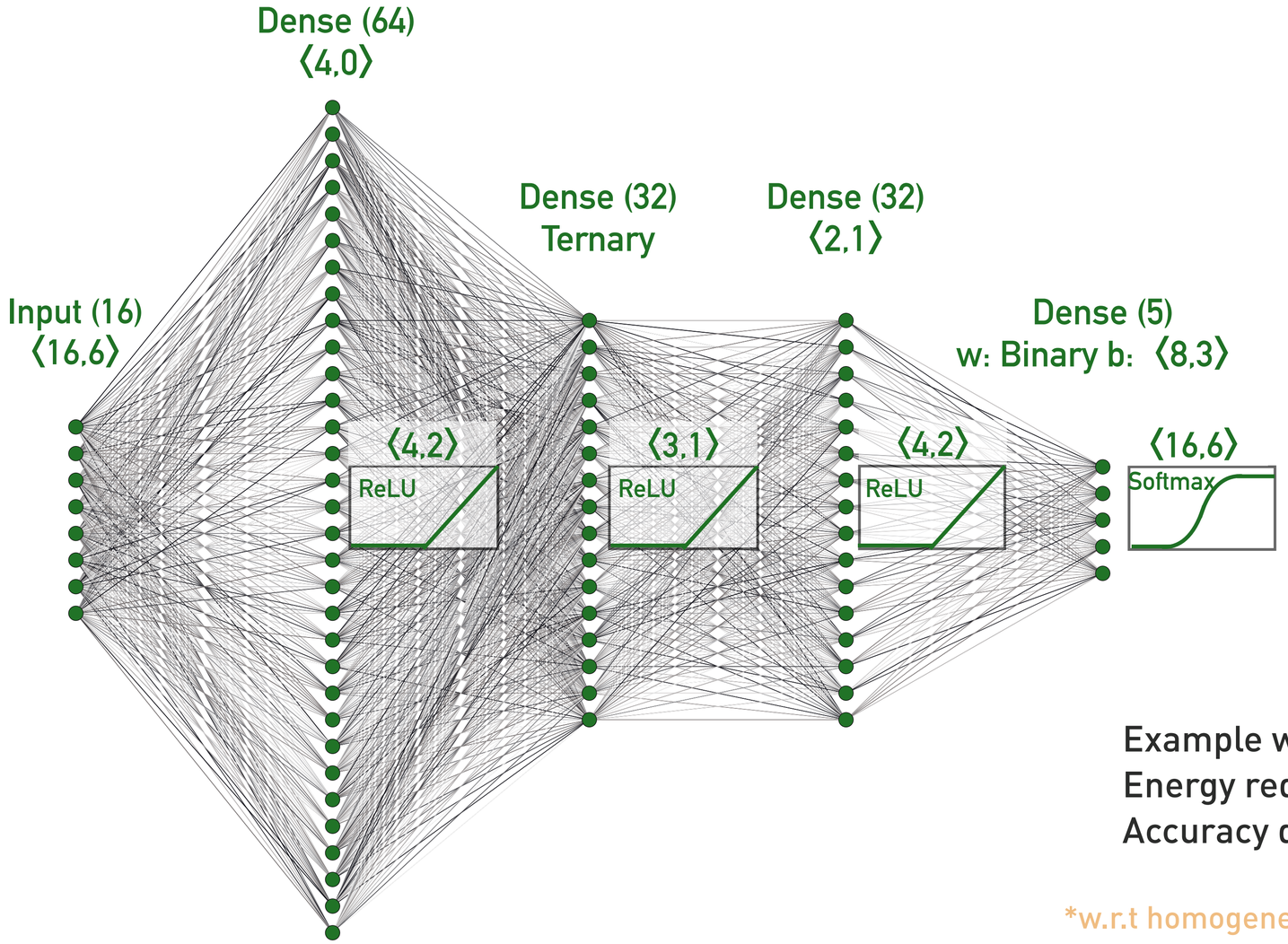
- Simultaneously scan quantizers and N filters/neurons (often less/more filters/neurons needed when quantizing)







Nature Machine Intelligence 3 (2021)



Example with target:
Energy reduction x4
Accuracy degradation max 5%

*w.r.t homogeneously quantized 6 bit model

Model	Accuracy (%)	Precision								$\frac{E}{E_{Q6}}$	$\frac{\text{Bits}}{\text{Bits}_{Q6}}$
		Dense	ReLU	Dense	ReLU	Dense	ReLU	Dense	Softmax		
QE	72.3	⟨4, 0⟩	⟨4, 2⟩	Ternary	⟨3, 1⟩	⟨2, 1⟩	⟨4, 2⟩	w: Stoc. bin. b: ⟨8, 3⟩	⟨16, 6⟩	0.27	0.18*

QPYTORCH ?

https://github.com/Xilinx/brevitas

```
import brevitas.nn as qnn
qnn.
```

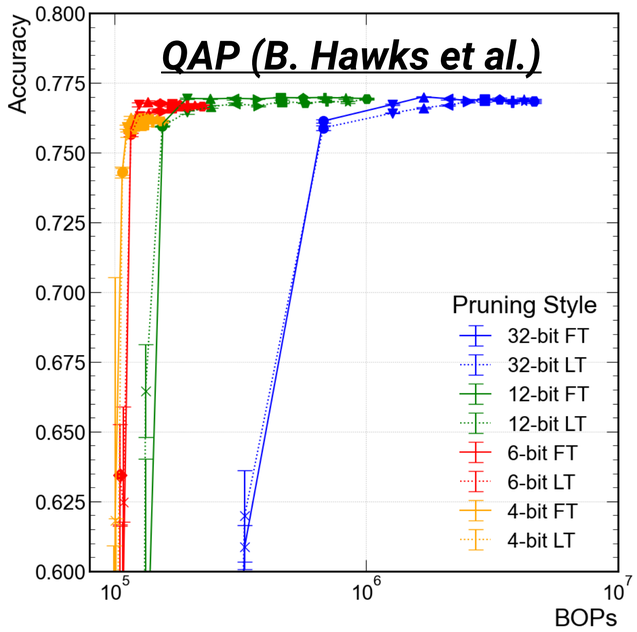
```
quant_bn (brevitas.nn)
QuantCat                          brevitas.nn.quant_eltwise
QuantTanh                         brevitas.nn.quant_activation
ScaleBias                         brevitas.nn.quant_scale_bias
quant_conv (brevitas.nn)
hadamard_classifier (brevitas.nn)
quant_accumulator (brevitas.nn)
quant_activation (brevitas.nn)
quant_avg_pool (brevitas.nn)
quant_convtranspose (brevitas.nn)
quant_dropout (brevitas.nn)
quant_eltwise (brevitas.nn)
quant_linear (brevitas.nn)
quant_max_pool (brevitas.nn)
quant_scale_bias (brevitas.nn)
quant_upsample (brevitas.nn)
BatchNorm1dToQuantScaleBias       brevitas.nn.quant_bn
BatchNorm2dToQuantScaleBias       brevitas.nn.quant_bn
ClampQuantAccumulator             brevitas.nn.quant_accumulator
```

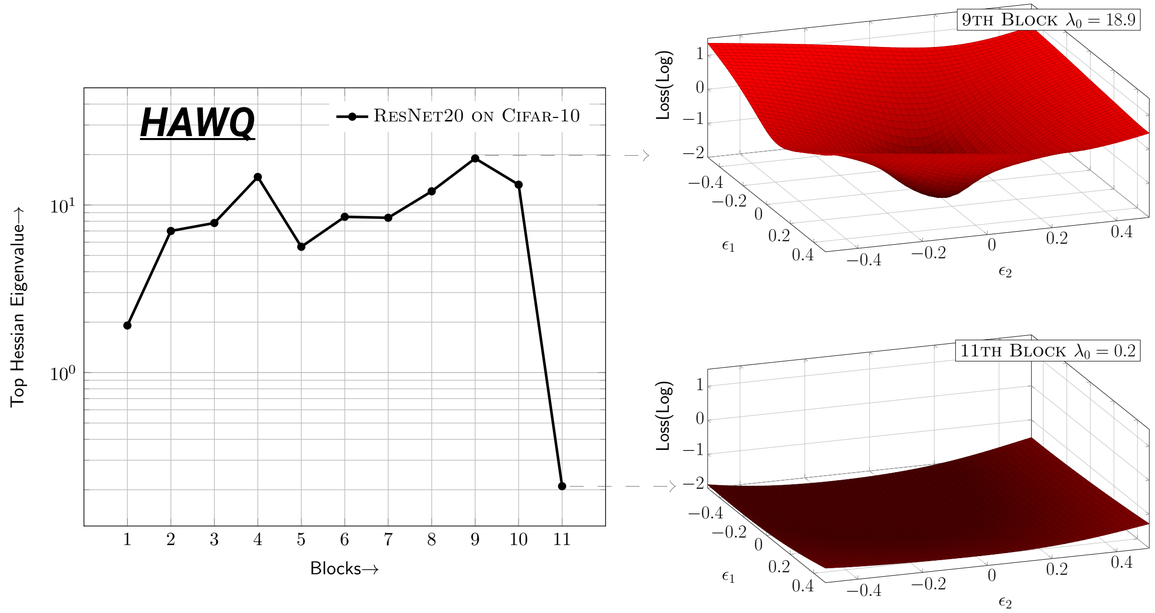
Brevitas like QKeras, but for PyTorch

- QAT library
- Support most common layers and activation functions

Other quantization techniques:

- ***HAWQ: Hessian AWare Quantization***
- ***Quantization Aware Pruning (B. Hawks et al.)***





QPYTORCH ?

Brevitas like QKeras, but for PyTorch

- QAT library
- Support most common layers and activation functions

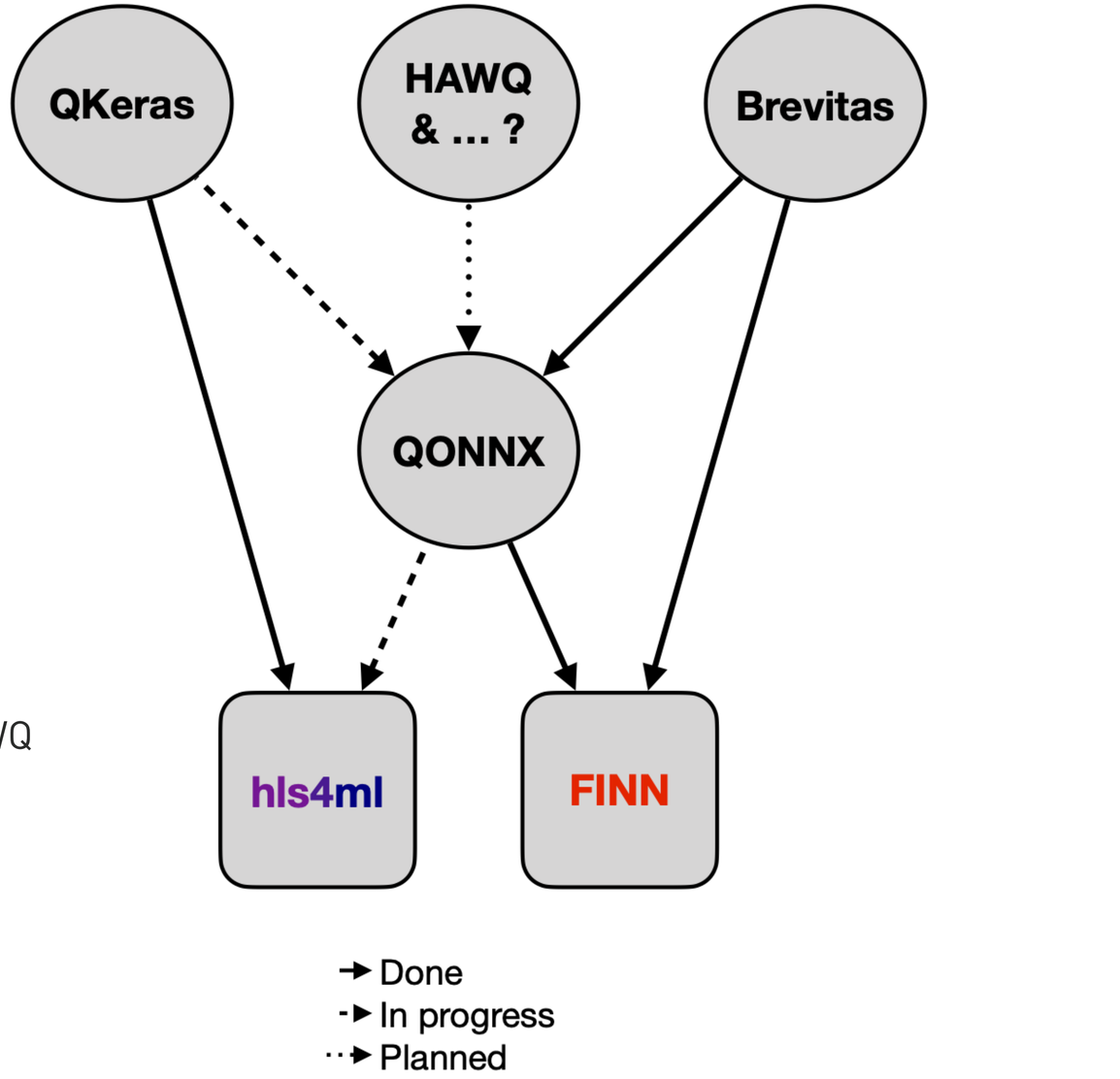
Other quantization techniques:

- ***HAWQ: Hessian AWare Quantization***
- ***Quantization Aware Pruning (B. Hawks et al.)***

hls4ml collaborate with Xilinx Research Labs to develop QOONX

- Introducing 'Quant' node to ONNX graph
- Brevitas (PyTorch) and QKeras (Keras) can export QONNX (HAWQ export in progress): then hls4ml and FINN can import QONNX

Quantized ONNX (QONNX), J. Mitrevski et. al



BDTs

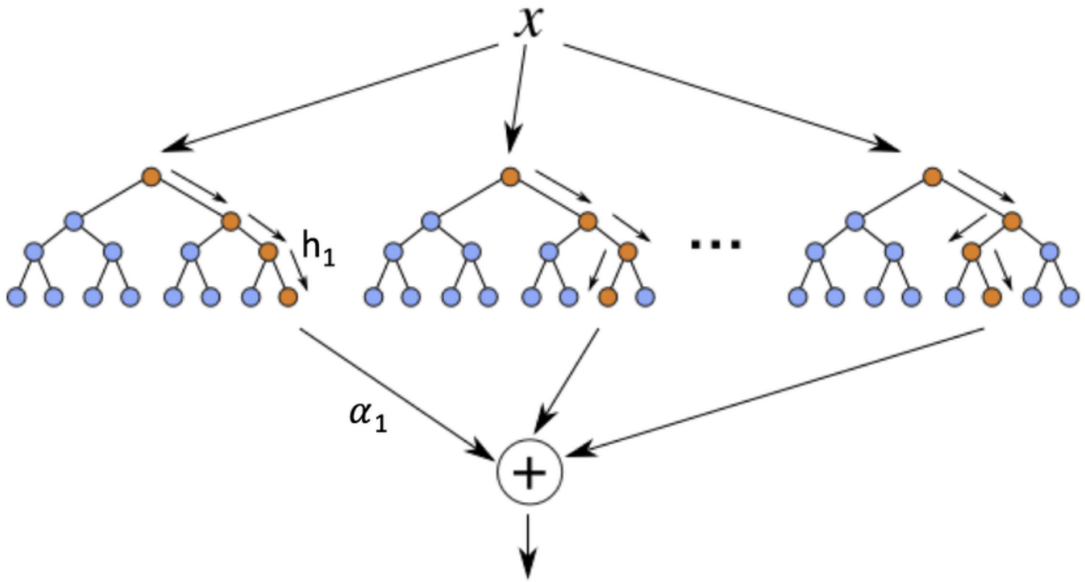
Often the best way to compress is: Just use BDTs!

Conifer is hls4ml for Boosted decision trees (scikit-learn, XGBoost)

If resource/latency constrained, BDT might be the way to go

- Depending on your data, might be as accurate as a DNN
- Usually significantly faster and more resource efficient

%VU9P	Accuracy	Latency	DSP	LUT
QKeras 6b	75.6%	40 ns	22 (~0%)	1%
sklearn + conifer	74.9%	5 ns	-	0.5%



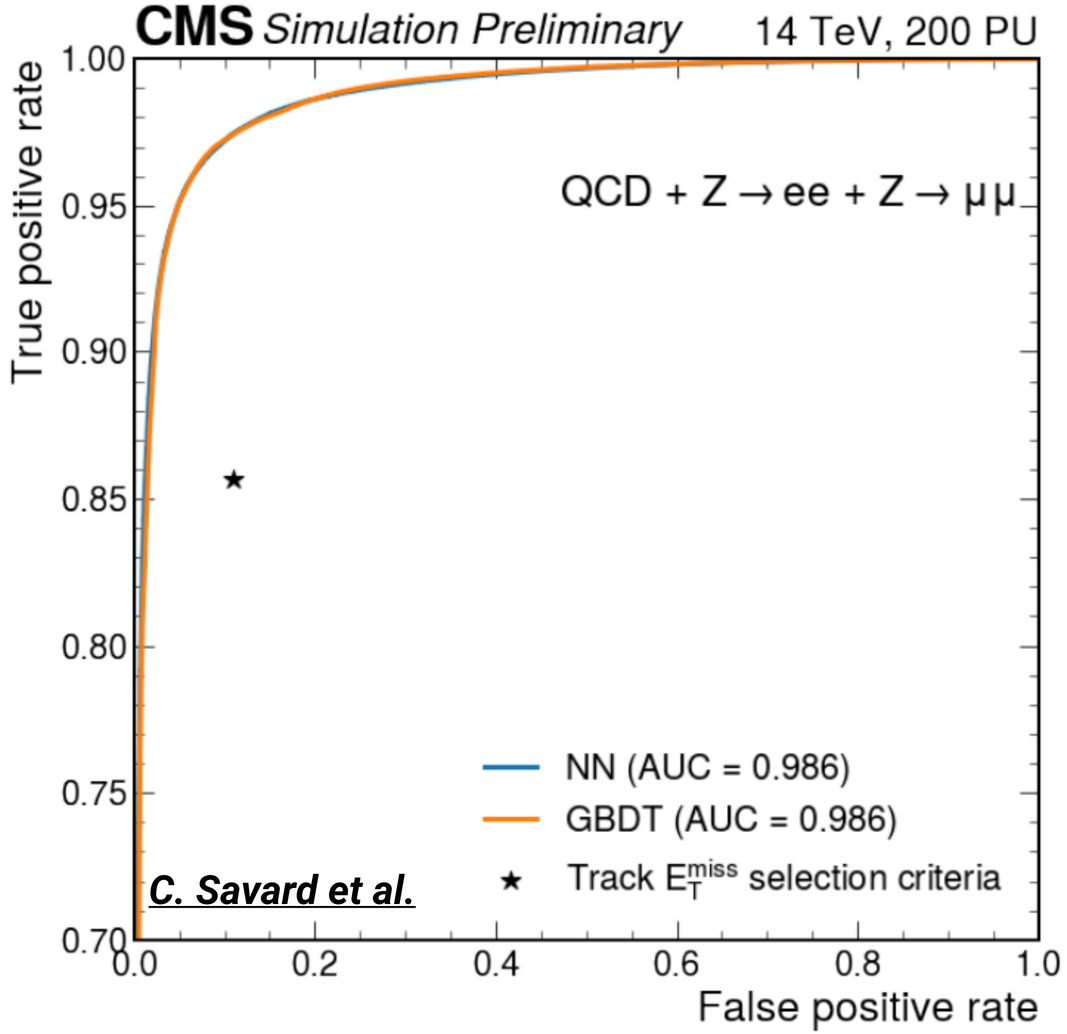
BDTs

Often the best way to compress is: Just use BDTs!

Conifer is hls4ml for Boosted decision trees (scikit-learn, XGBoost)

If resource/latency constrained, BDT might be the way to go

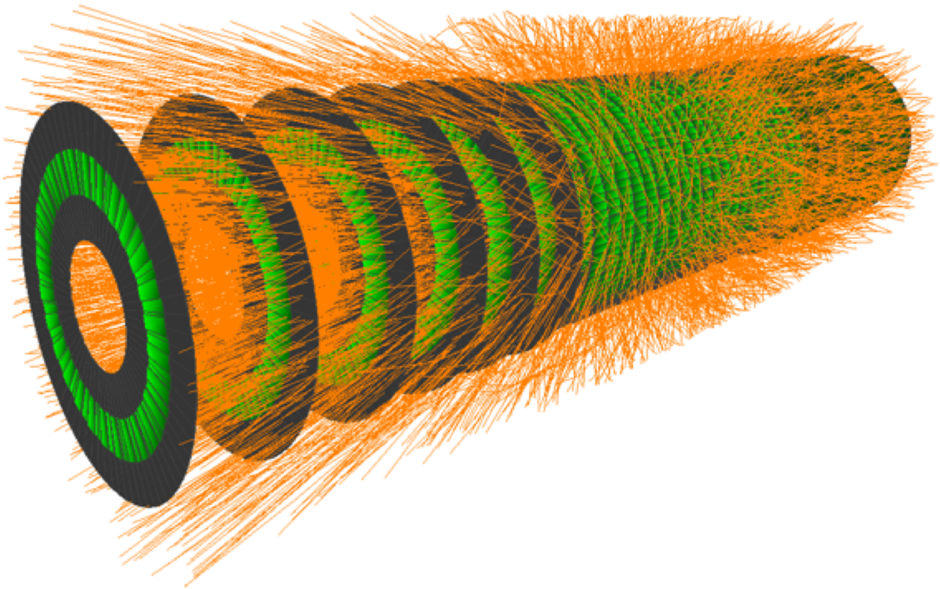
- Depending on your data, might be as accurate as a DNN
- Usually significantly faster and more resource efficient



Model	Python AUC	HLS AUC	Latency (clk)	LUT %	FF %	DSP %
NN	0.985	0.982	8	0.104	0.029	0.292
GBDT	0.986	0.981	3	0.140	0.027	0.0

DOI:10.3389/fdata.2022.828666

ML for tracking



In HL-LHC, will need to do track finding at L1

- O(1000) hits, O(100) tracks, 40 MHz rate, ~5 μs latency

Graph Neural Networks for fast charged particle tracking

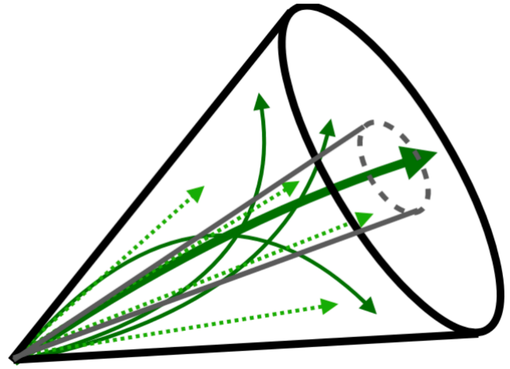
- Custom converter for PyTorch Geometric integrated in hls4ml

Throughput-optimized for L1 applications, resource-optimised for co-processing

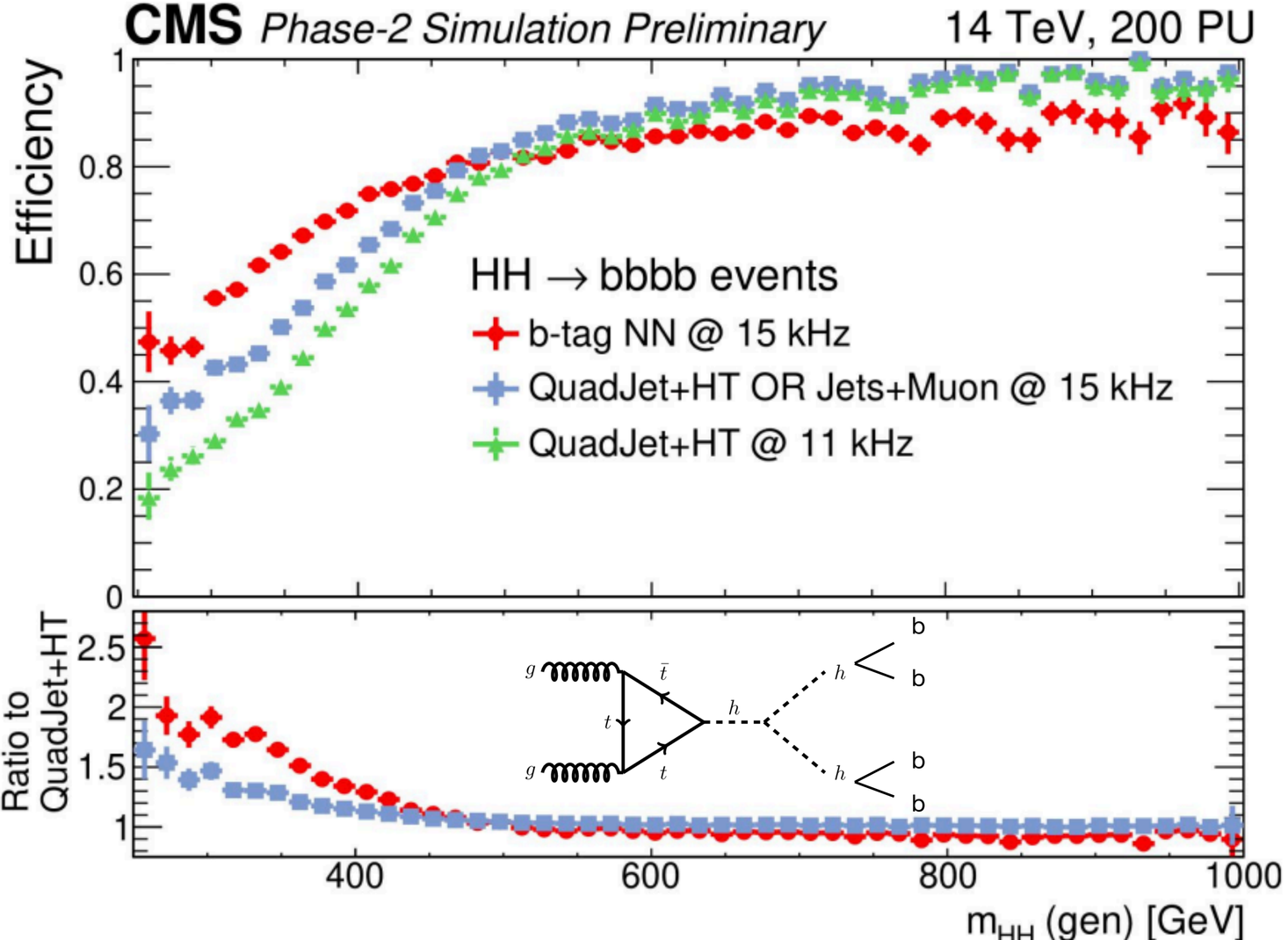
Design	(n_{nodes}, n_{edges})	RF	Precision	Latency [cycles]	II [cycles]	DSP [%]	LUT [%]	FF [%]	BRAM [%]
Throughput-opt.	(28, 56)	1	`ap_fixed<14,7>`	59	1	99.9	66.0	11.7	0.7
Throughput-opt.	(28, 56)	8	`ap_fixed<14,7>`	75	8	21.9	23.8	4.7	0.7
Resource-opt.	(28, 56)	1	`ap_fixed<14,7>`	79	28	56.6	17.6	3.9	13.1
Resource-opt.	(448, 896)	1	`ap_fixed<14,7>`	470	174	56.6	25.0	7.4	16.5
Resource-opt.	(448, 896)	8	`ap_fixed<14,7>`	1590	520	5.6	25.0	7.4	16.3

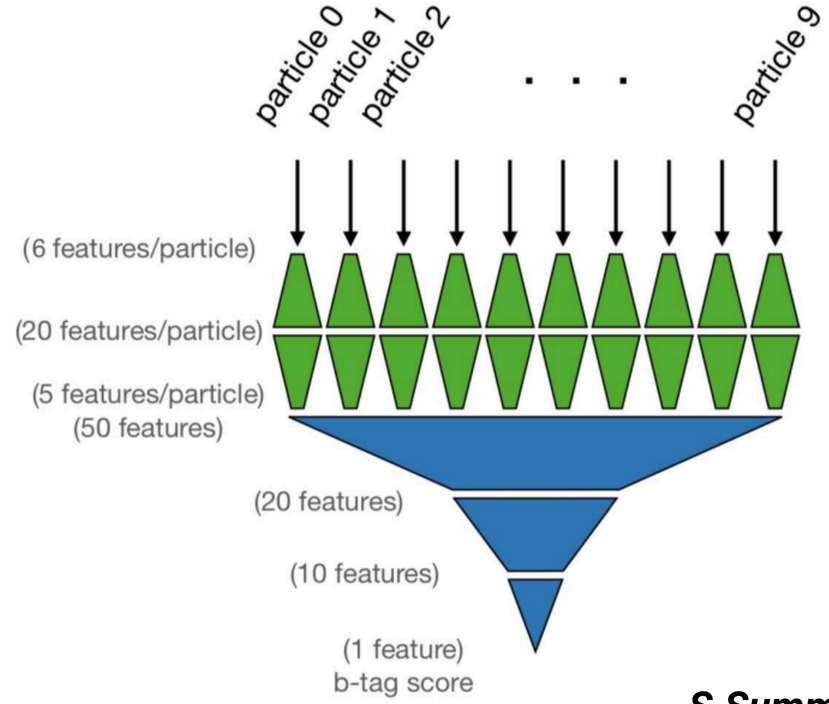
The target FPGA is a Xilinx Virtex UltraScale+ VU9P FPGA (part number xcvu9p-flga2104-2L-e), which has 6,840 DSPs, 1,182,240 LUTs, 2,364,480 FFs, and 75.9 Mb of BRAM (Xilinx, Inc., 2021). A 5 ns clock period is used.

ML for jet tagging



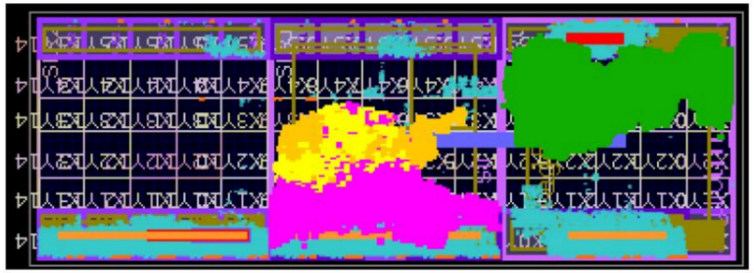
cds.cern.ch/record/2814728/





S.Summers

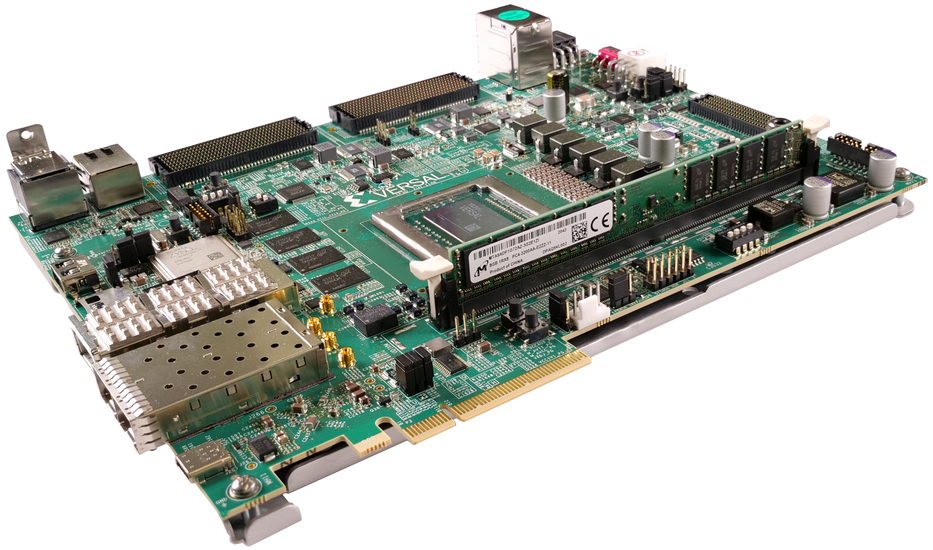
Jet Finding
b tag NN



AI engines

More and more dedicated AI processors on the market

- Can we utilise highly specialised ML hardware at CERN?



AI engines

GNNs with Versal AI, P. Schwaebig

More and more dedicated AI processors on the market

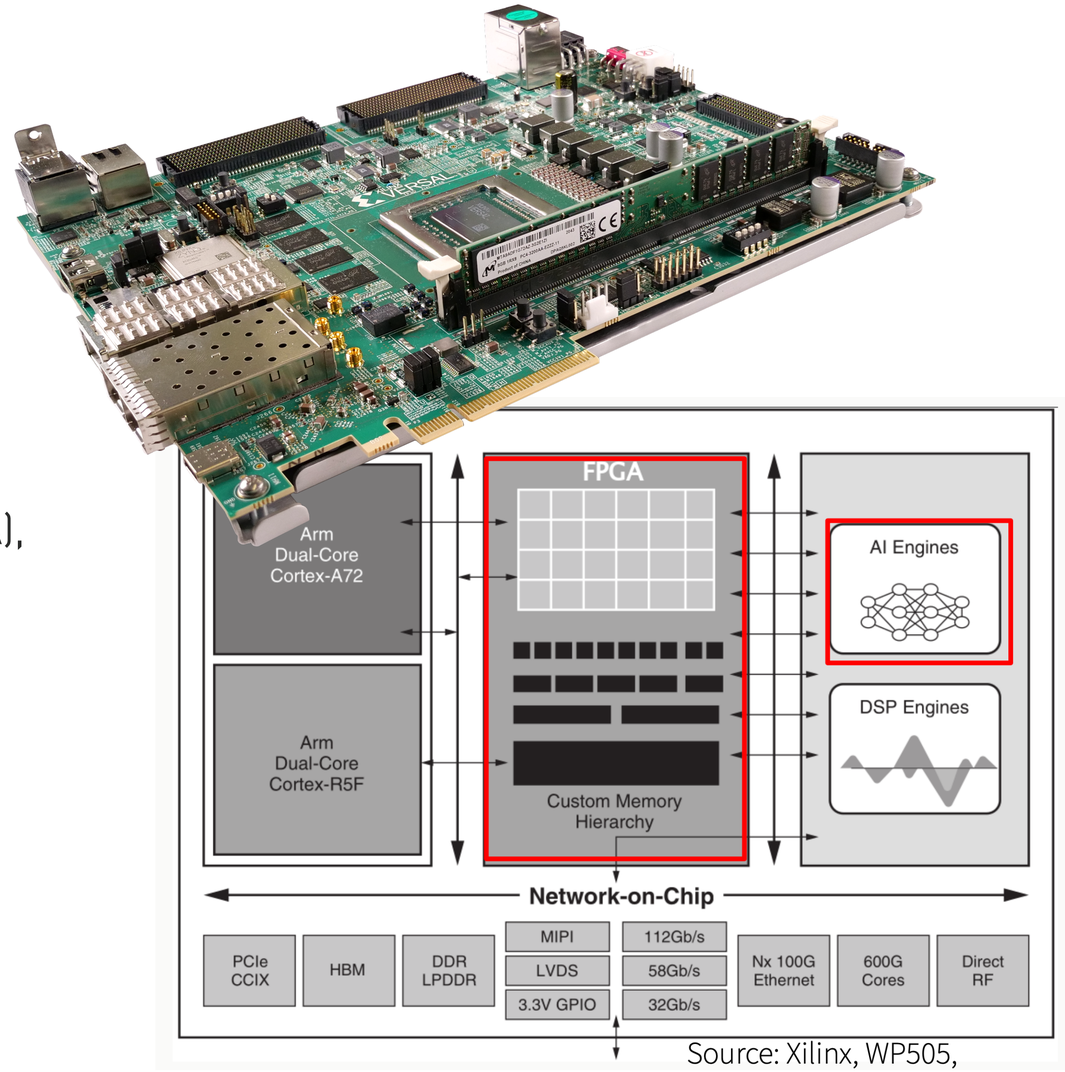
- Can we utilise highly specialised ML hardware at CERN?

Xilinx Versal AI processors

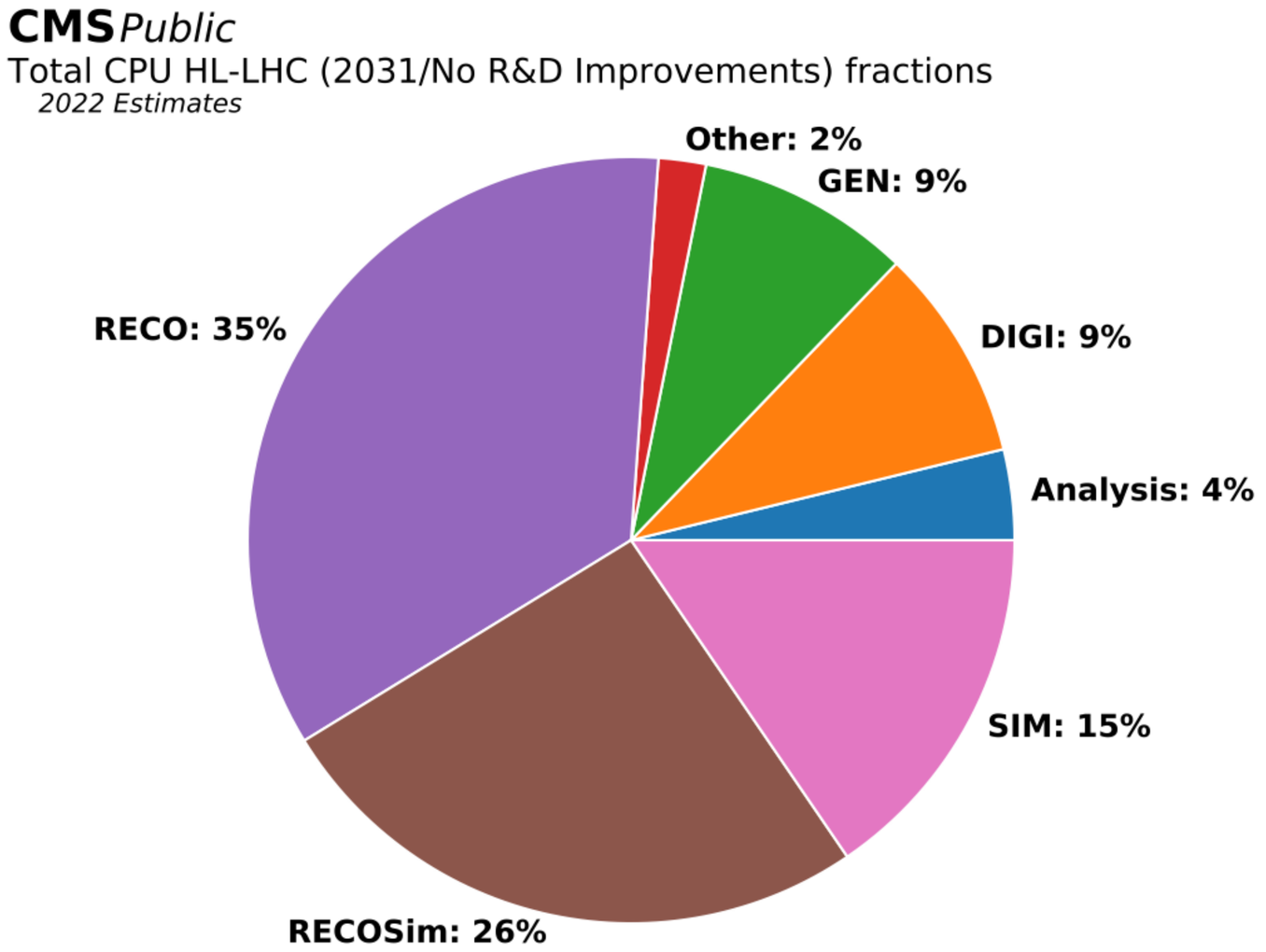
- Programmed in C/C++
- Running at 1 GHz
- Example Xilinx ACAP board: 400 AI processors, ~2M logic cells (FPGA), 2k DSPs, Arm CPU, Arm RPU
- Data can move back and forth between AI Engines and FPGA

Currently explored for real-time tracking in trigger application

- Interaction Network for pattern recognition (similar to **DeZoort et al**)
- Deployed on Xilinx Versal VC1902 ACAP



Source: Xilinx, WP505,


CMS Public
Total CPU HL-LHC (2031/No R&D Improvements) fractions
2022 Estimates
Other: 2%
GEN: 9%
DIGI: 9%
Analysis: 4%
SIM: 15%
RECOSim: 26%
RECO: 35%

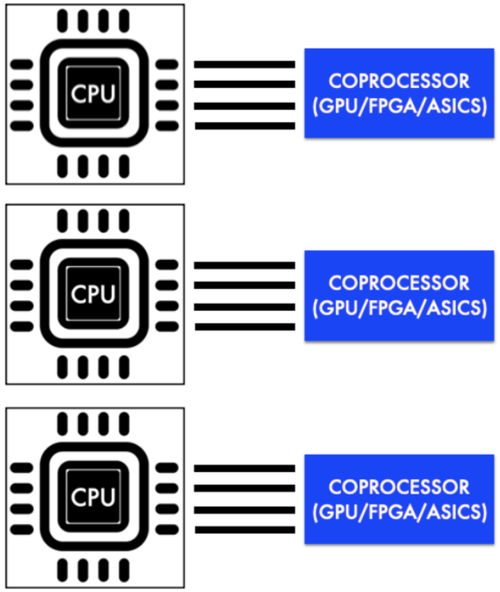
FPGAs as accelerators

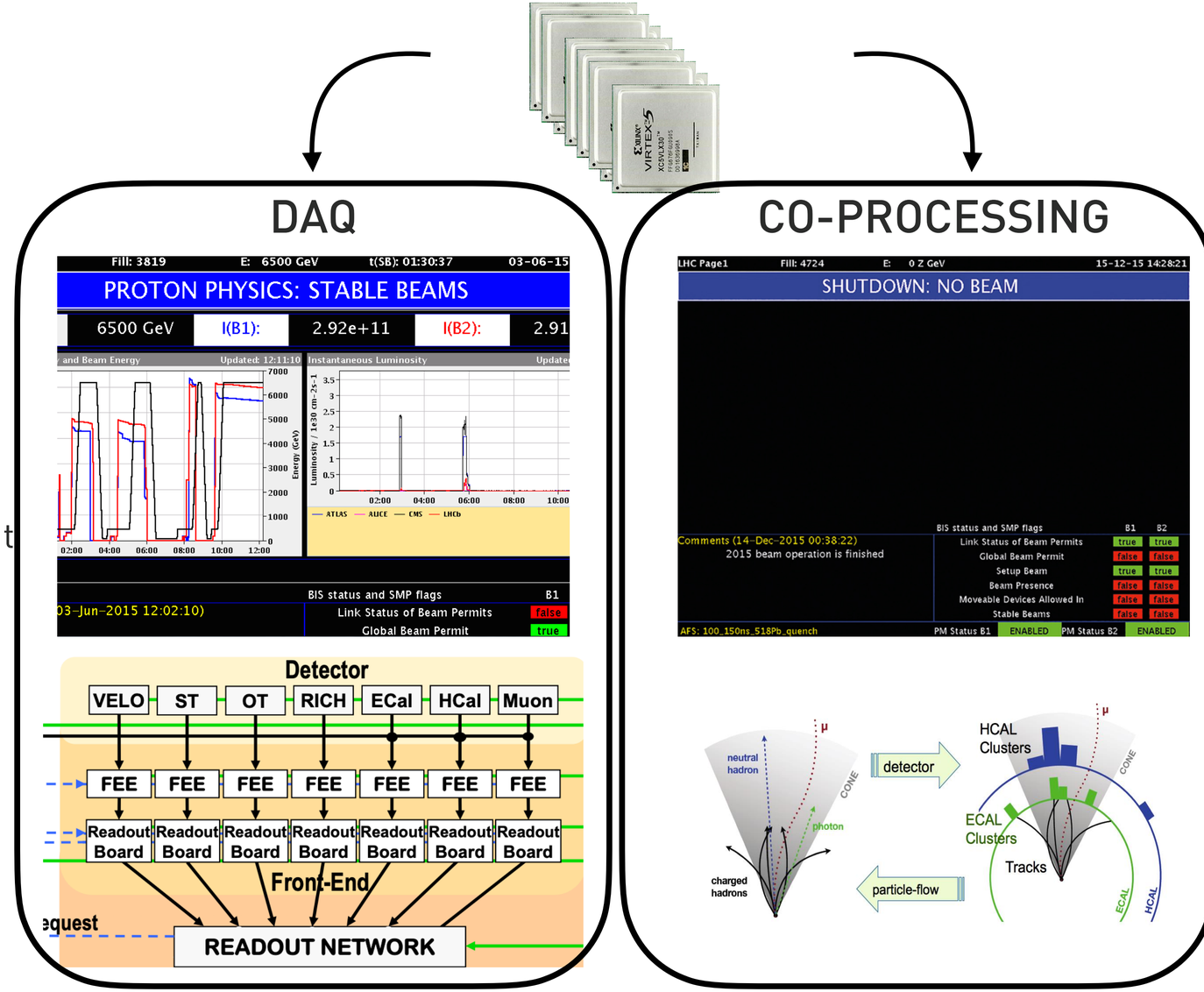
Our DAQ FPGAs are idle ~50% of the time (no collisions)

- Could these be utilised for co-processing?

Heterogeneous compute on-site with FPGA co-processors

- E.g LHCb: No hardware trigger, but ~200 FPGA read-out boards receiving data from sub detectors
- Repurpose for data processing when LHC is off?





C. Beteta, I. Bezshyiko, N. Serra

FPGAs as accelerators

Our DAQ FPGAs are idle ~50% of the time (no collisions)

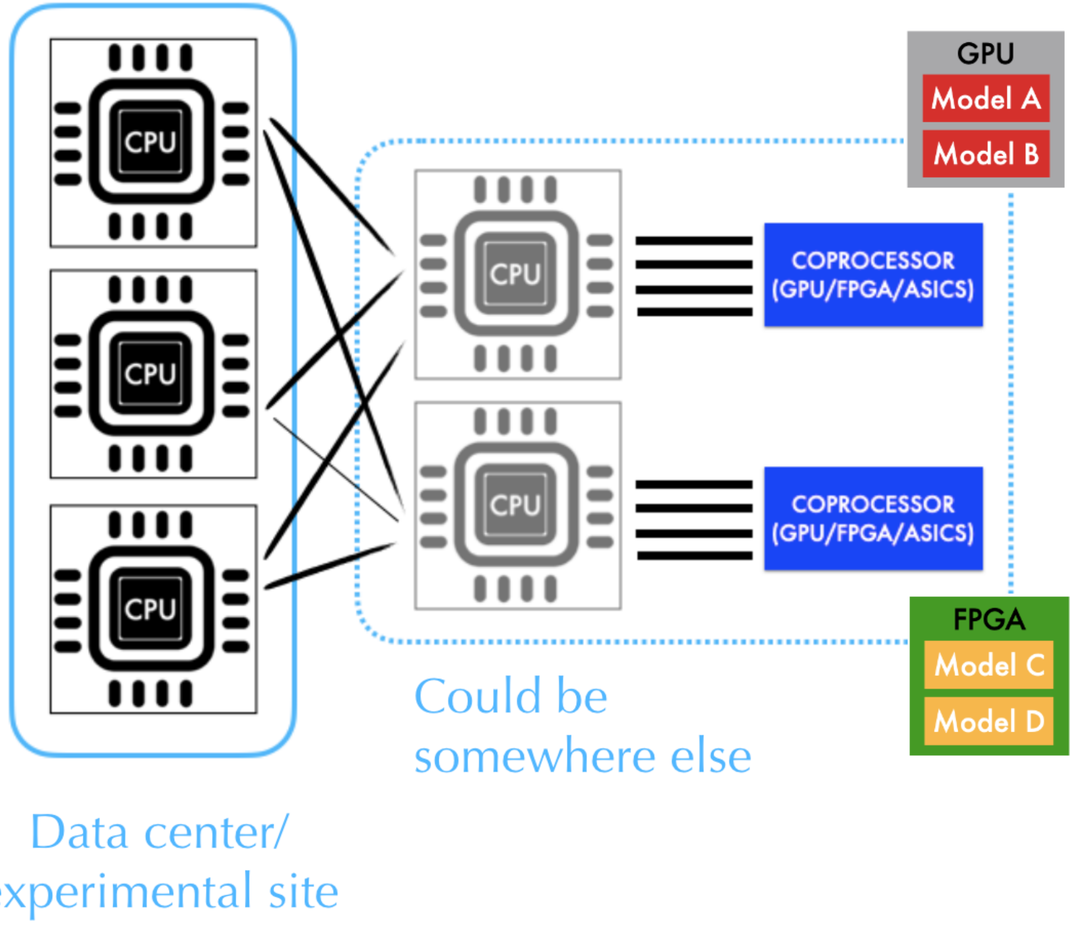
- Could these be utilised for co-processing?

Heterogeneous compute on-site with FPGA co-processors

- E.g LHCb: No hardware trigger, but ~200 FPGA read-out boards receiving data from sub detectors
- Repurpose for data processing when LHC is off?

Alternative: FPGA-as-a-Service toolkit for Cloud inference

- Using hls4ml to deploy large models on FPGA, run inference in the cloud



Algorithm	Platform	Number of Devices	Batch Size	Inf./s [Hz]	Bandwidth [Gbps]
FACILE	AWS EC2 F1	1	16,000	36 M	23
FACILE	Alveo U250	1	16,000	86 M	55
FACILE	T4 GPU	1	16,000	8 M	5.1

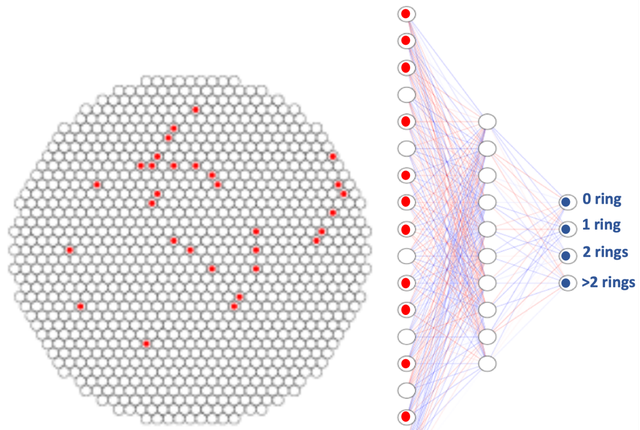
FaaST, D. Rankin et. al

hls4ml in other CERN experiments

NA62: Measuring ultra rare kaon decays

BR($K^+ \rightarrow \pi^+ \nu\bar{\nu}$) = (8.4±1.0) x 10^{-11}

- Hardware FPGA trigger 800 MHz→1 MHz
- DNN for RICH rings identification
- Latency: 40 cycles at 100 MHz



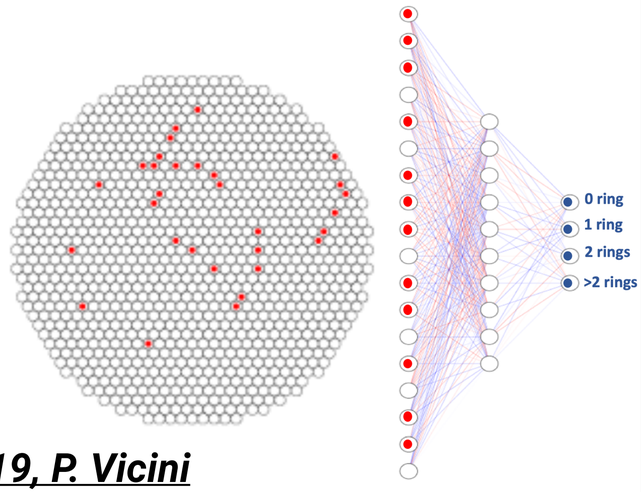
CHEP 2019, P. Vicini

hls4ml in other CERN experiments

NA62: Measuring ultra rare kaon decays

BR($K^+ \rightarrow \pi^+ \nu\bar{\nu}$) = (8.4±1.0) x 10^{-11}

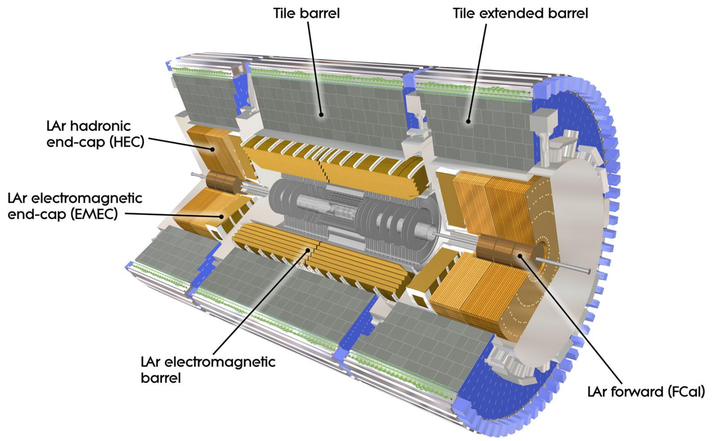
- Hardware FPGA trigger 800 MHz→1 MHz
- DNN for RICH rings identification
- Latency: 40 cycles at 100 MHz



CHEP 2019, P. Vicini

ATLAS Liquid Argon Calorimeter

- Recurrent Neural Networks (RNN) for real-time energy reconstructions
- Latency ~200 ns on Intel Stratix-10 FPGA



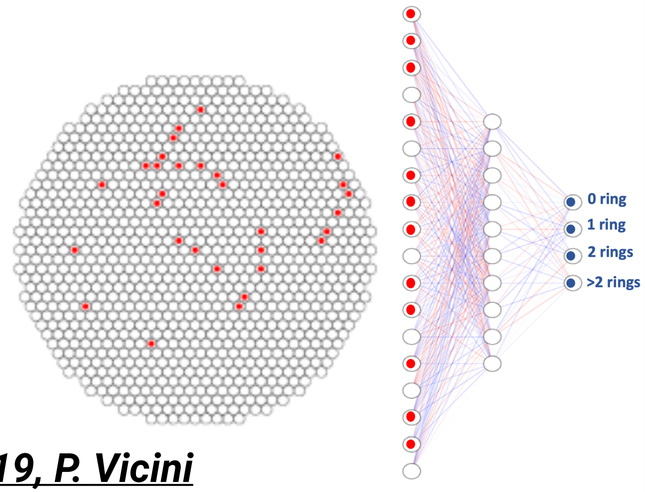
DOI:10.1007/s41781-021-00066-y

hls4ml in other CERN experiments

NA62: Measuring ultra rare kaon decays

BR($K^+ \rightarrow \pi^+ \nu\bar{\nu}$) = (8.4±1.0) x 10^{-11}

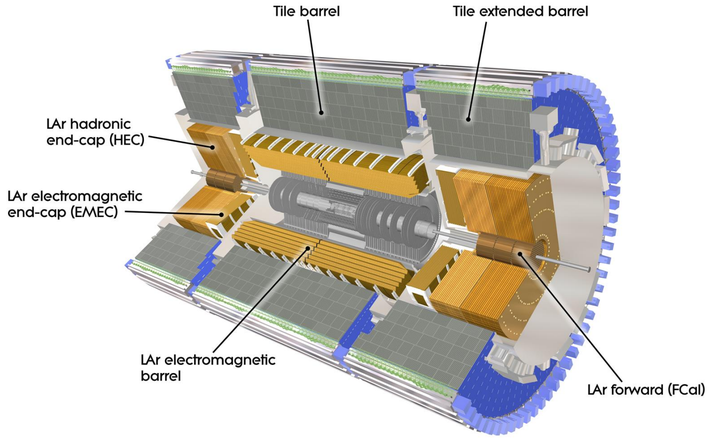
- Hardware FPGA trigger 800 MHz→1 MHz
- DNN for RICH rings identification
- Latency: 40 cycles at 100 MHz



CHEP 2019, P. Vicini

ATLAS Liquid Argon Calorimeter

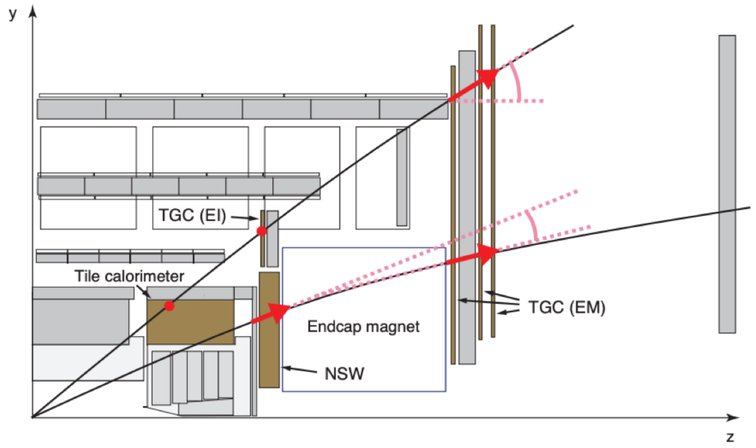
- Recurrent Neural Networks (RNN) for real-time energy reconstructions
- Latency ~200 ns on Intel Stratix-10 FPGA



DOI:10.1007/s41781-021-00066-y

Muon segment finding and reconstruction

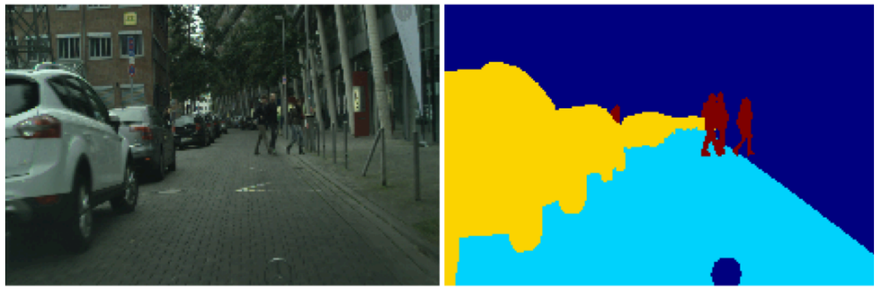
- Regression of muon position and angle
- Feasible within 400 ns budget



R. Teixeira de Lima, R Rojas Caballero et al.

...and outside of HEP

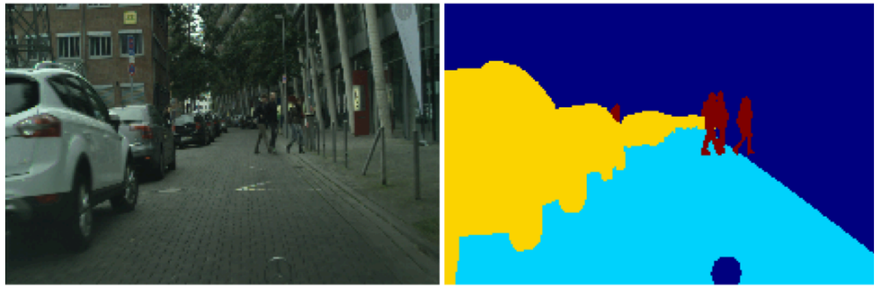
Semantic segmentation for autonomous vehicles



N. Ghielmetti et al.

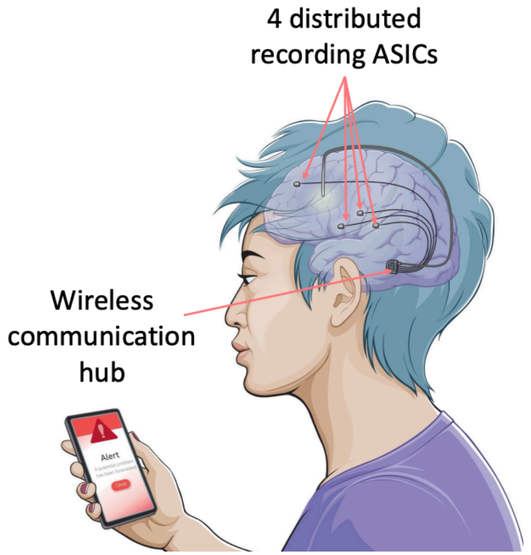
...and outside of HEP

Semantic segmentation for autonomous vehicles



N. Ghielmetti et al.

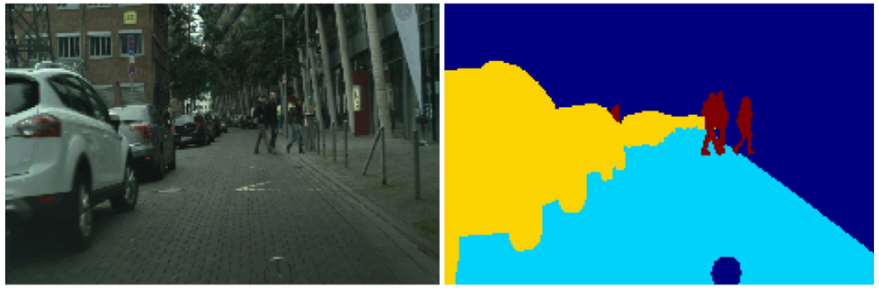
Seizure Predicting Brain Implant



W. Lemaire et al.

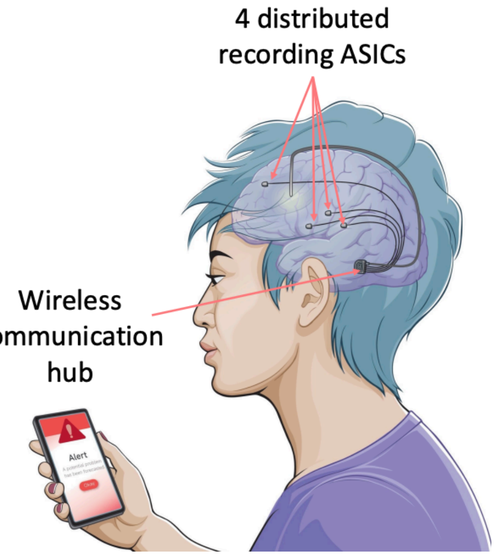
...and outside of HEP

Semantic segmentation for autonomous vehicles



N. Ghielmetti et al.

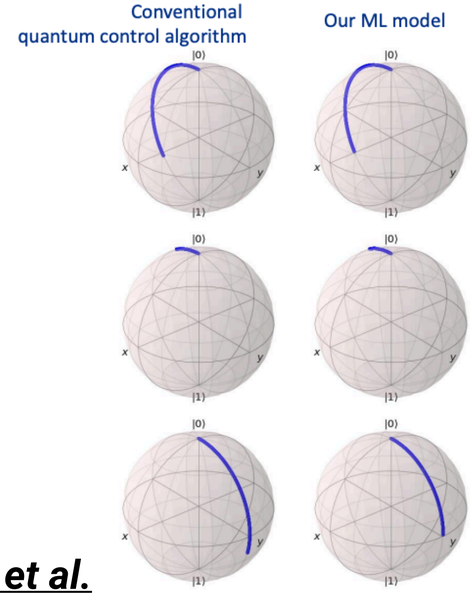
Seizure Predicting Brain Implant



W. Lemaire et al.

NN accelerator for quantum control

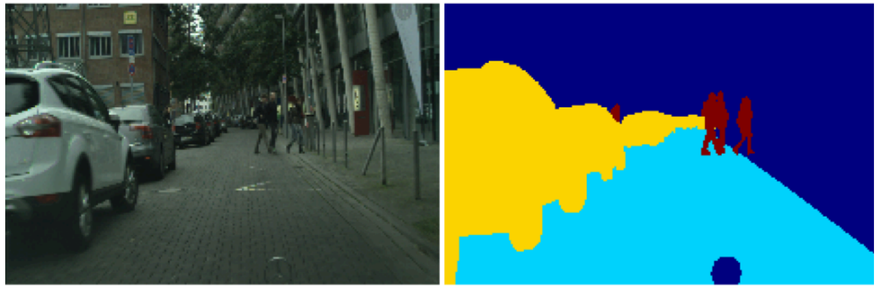
- Putting control in cryostat
  (e.g optimal pulse parameters)



D Xu et al.

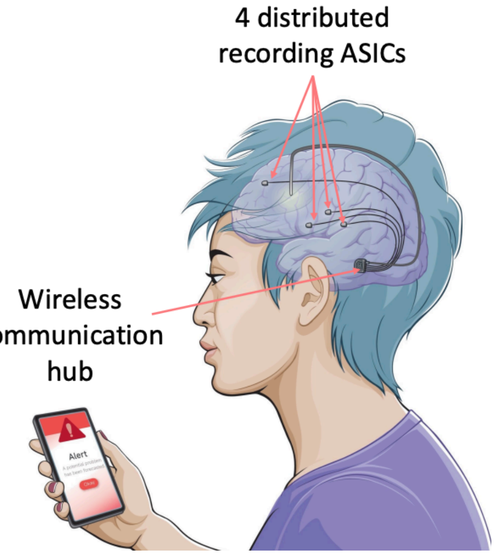
...and outside of HEP

Semantic segmentation for autonomous vehicles



N. Ghielmetti et al.

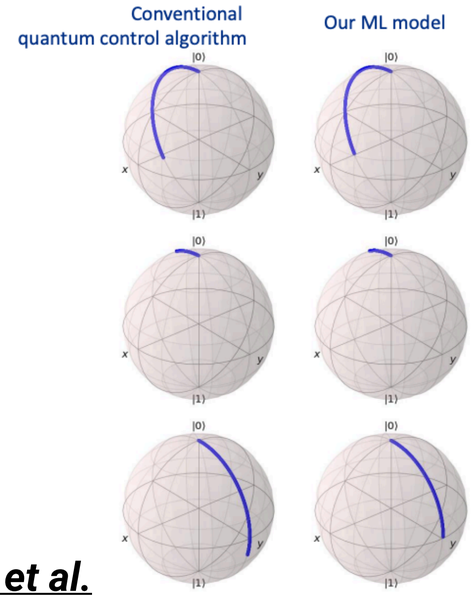
Seizure Predicting Brain Implant



W. Lemaire et al.

NN accelerator for quantum control

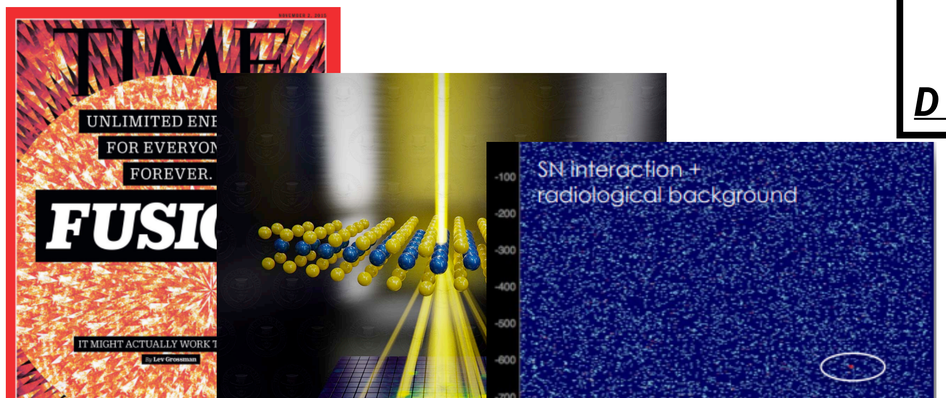
- Putting control in cryostat
  (e.g optimal pulse parameters)



D Xu et al.

Other examples

- ***For fusion science phase/mode monitoring***
- ***Crystal structure detection***
- ***Triggering in DUNE***
- ***Accelerator control***
- ***Magnet Quench Detection***
- ***MLPerf tinyML benchmarking***
- ***Food contamination detection***
- etc....



Benchmarking

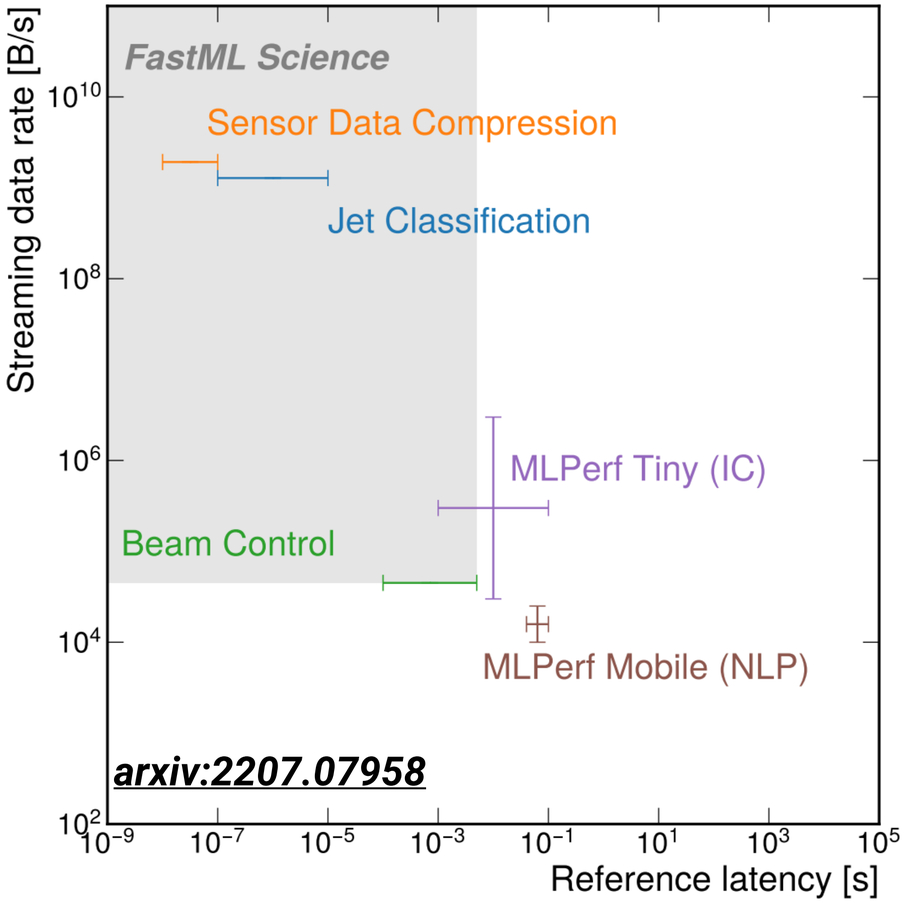
Datasets: Common FastML Science Benchmarking datasets

- guide design of edge ML hardware and software for sub-microsecond inference!

Algorithms: hls4ml-FINN benchmarked in MLPerf™

- how fast systems can process inputs and produce results
- demonstrate efficient and low-latency solutions on FPGAs with hls4ml and FINN

Consistently competitive (QKeras+hls4ml, semantic segmentation, MLPerf)



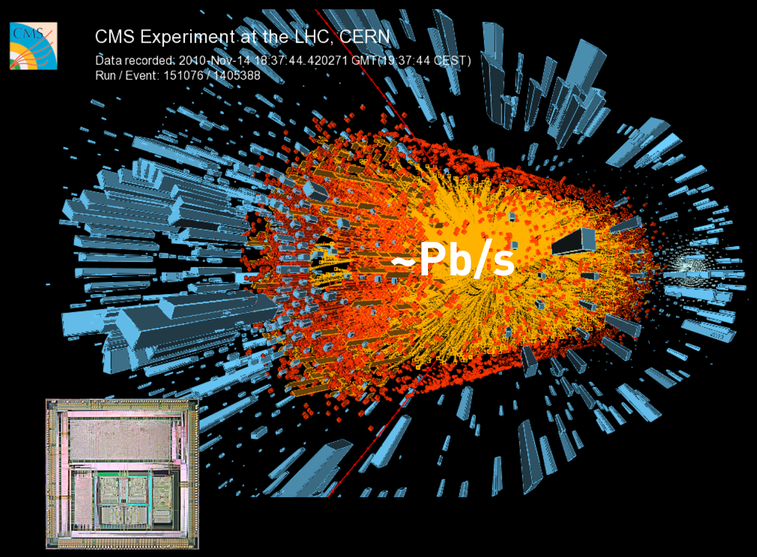
https://mlcommons.org/en/inference-tiny-07/

Model	LUT		LUTRAM		FF		BRAM [36 kb]		DSP		Latency [ms]	Energy/inf. [μJ]
Pynq-Z2												
Available	53 200		17 400		106 400		140		220		–	–
IC (hls4ml)	28 544	53.7%	3 756	21.6%	49 215	46.3%	42	30.0%	4	1.8%	27.3	44 330
IC (FINN)	24 502	46.1%	2 086	12.0%	34 354	32.3%	100	71.4%	0	0.0%	1.5	2 535
AD	40 658	76.4%	3 659	21.0%	51 879	48.8%	14.5	10.4%	205	93.2%	0.019	30.1
KWS	33 732	63.4%	1 033	5.9%	34 405	32.3%	37	26.4%	1	0.5%	0.017	30.9

arxiv:2103.05579

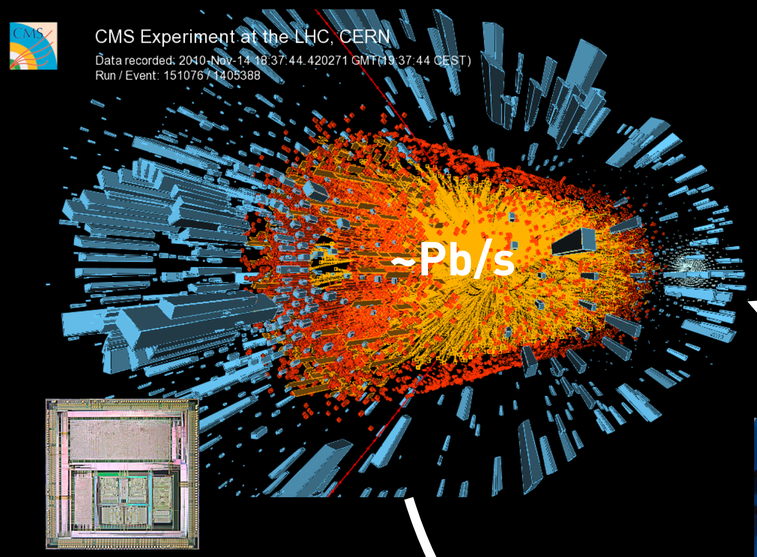
Detector

- 100% of events
- Latency: 25 ns



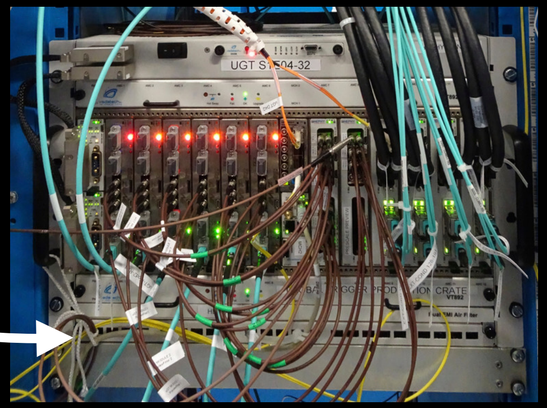
Detector

- 100% of events
- Latency: 25 ns



accept/reject

40 MHz
~Tb/s

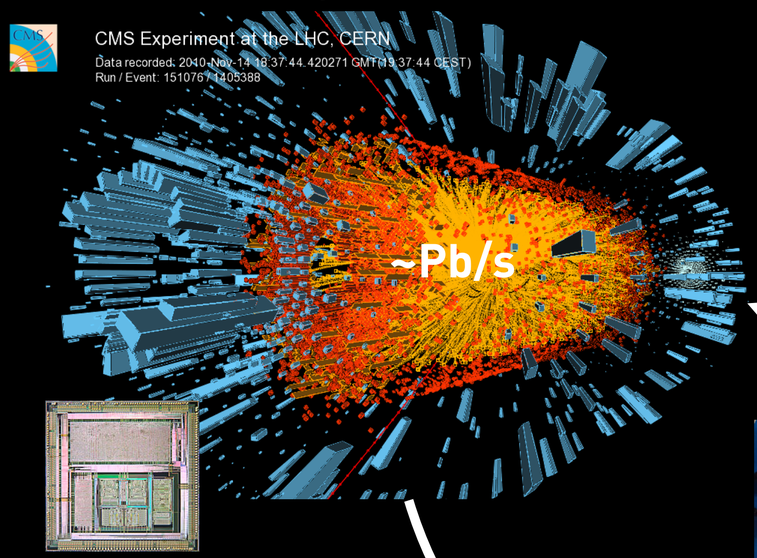


Level-1 hardware trigger

- Reject 99.75% of events
- Latency: O(1) **µs**

Detector

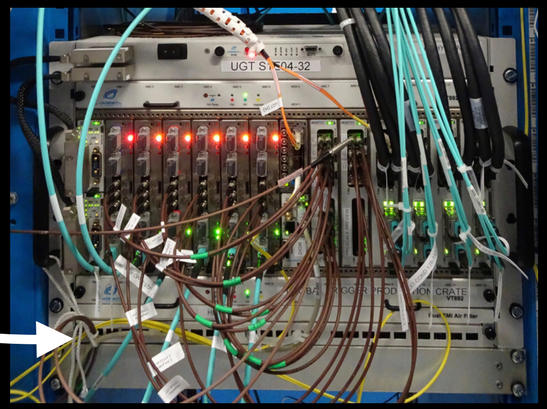
- 100% of events
- Latency: 25 ns



750 kHz
~Tb/s

accept/reject

40 MHz
~Tb/s



Level-1 hardware trigger

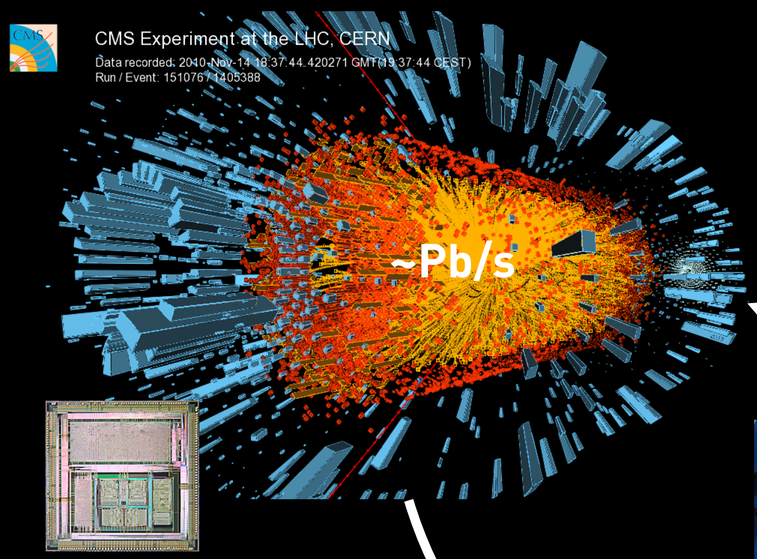
- Reject 99.75% of events
- Latency: O(1) **µs**

High Level Trigger

- Reject 99.982% of events
- Latency: O(100) ms

Detector

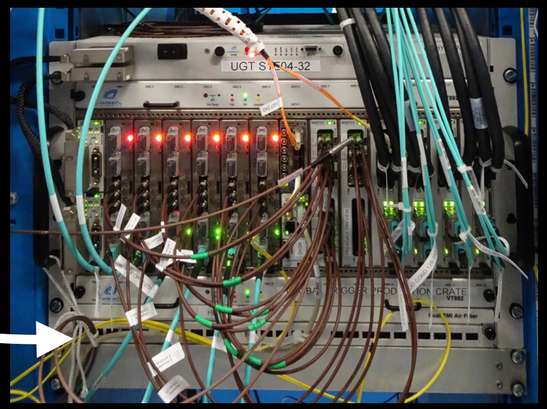
- 100% of events
- Latency: 25 ns



750 kHz
~Tb/s

accept/reject

40 MHz
~Tb/s



Level-1 hardware trigger

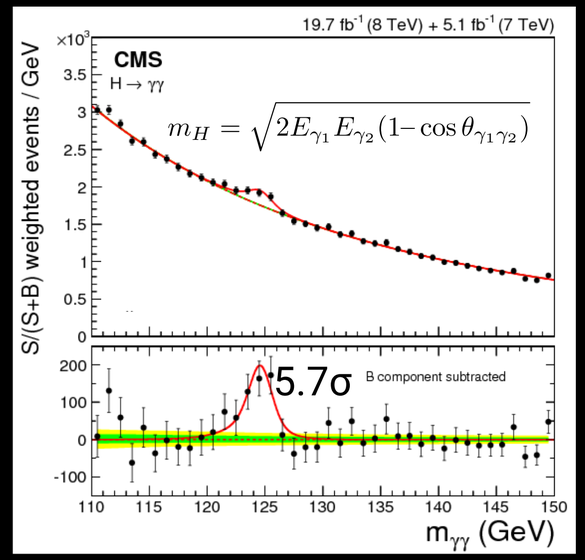
- Reject 99.75% of events
- Latency: O(1) **µs**

High Level Trigger

- Reject 99.982% of events
- Latency: O(100) ms

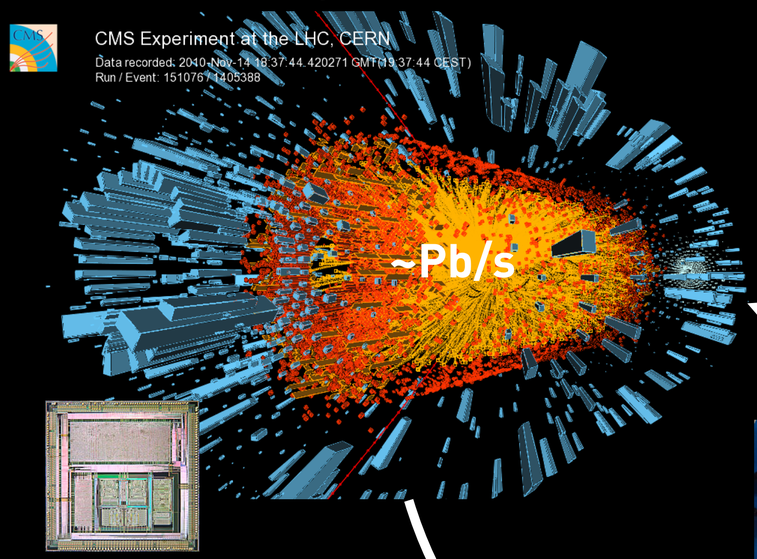
~1 kHz
~Gb/s

Offline reconstruction and storage



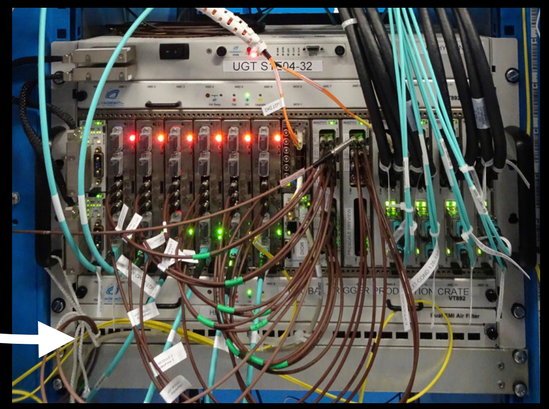
Do physics with 0.018% of collision events, the rest is discarded!
Throughput is main limitation

Detector
- 100% of events
- Latency: 25 ns



750 kHz
~Tb/s

accept/reject



40 MHz
~Tb/s

Level-1 hardware trigger
- Reject 99.75% of events
- Latency: O(1) **µs**

High Level Trigger
- Reject 99.982% of events
- Latency: O(100) ms

~1 kHz
~Gb/s

Offline reconstruction and storage