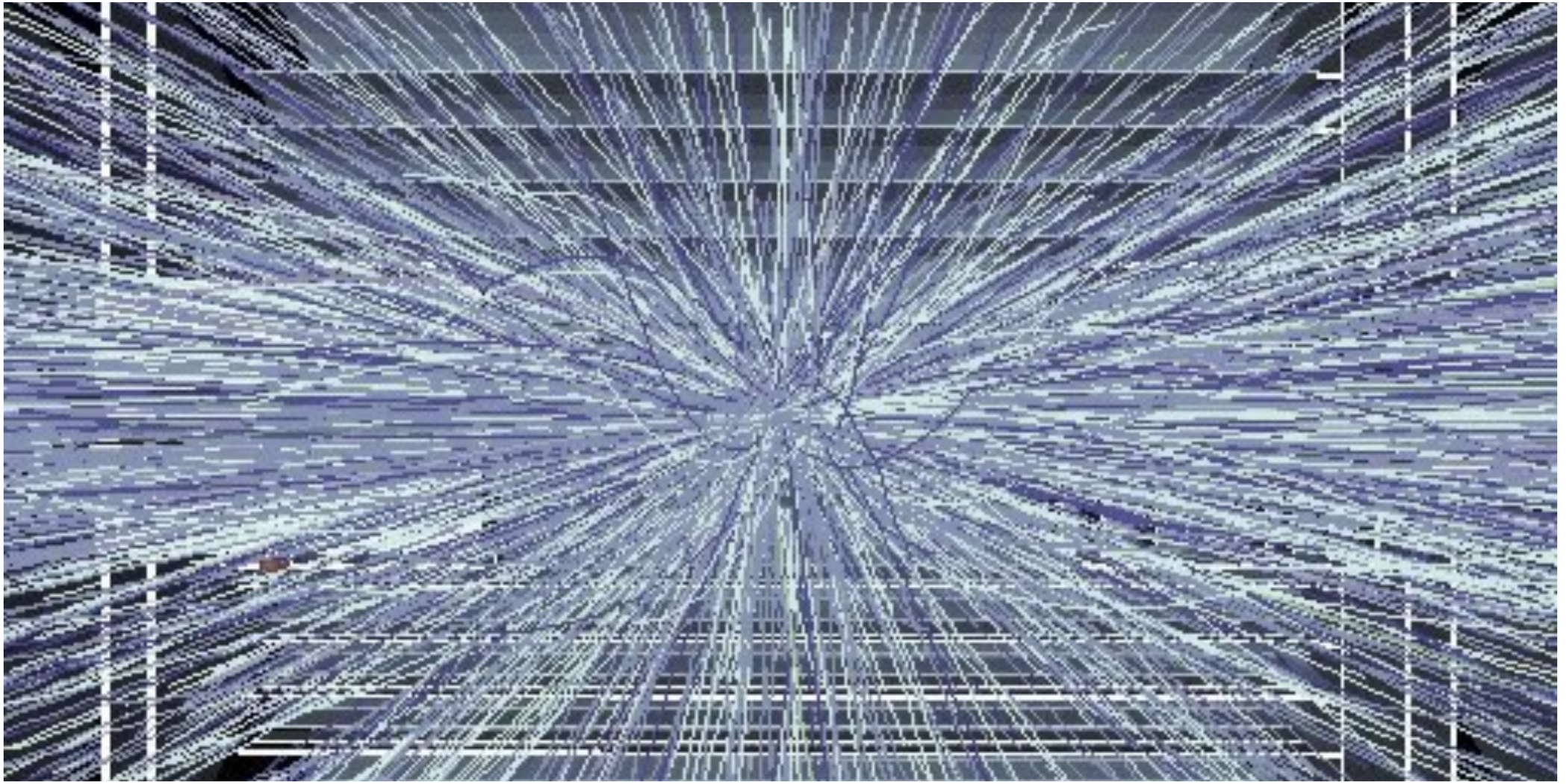
Deep Learning Applications for collider physics Lecture 3

Maurizio Pierini

Plan for these lectures

	Day1	Day2	Day3	Day4	Day5
1st hour	Introduction	ConvNN	LHC & fast Inference	Autoencoders	Graphs
2nd hour	Dense NNNs	GANs	RNNs	Anomaly Detection	Graphs
Tutorial	Dense NNNs	ConvNN	RNNs	AE	Graphs

European Research Council



Why Deep Learning for HEP?

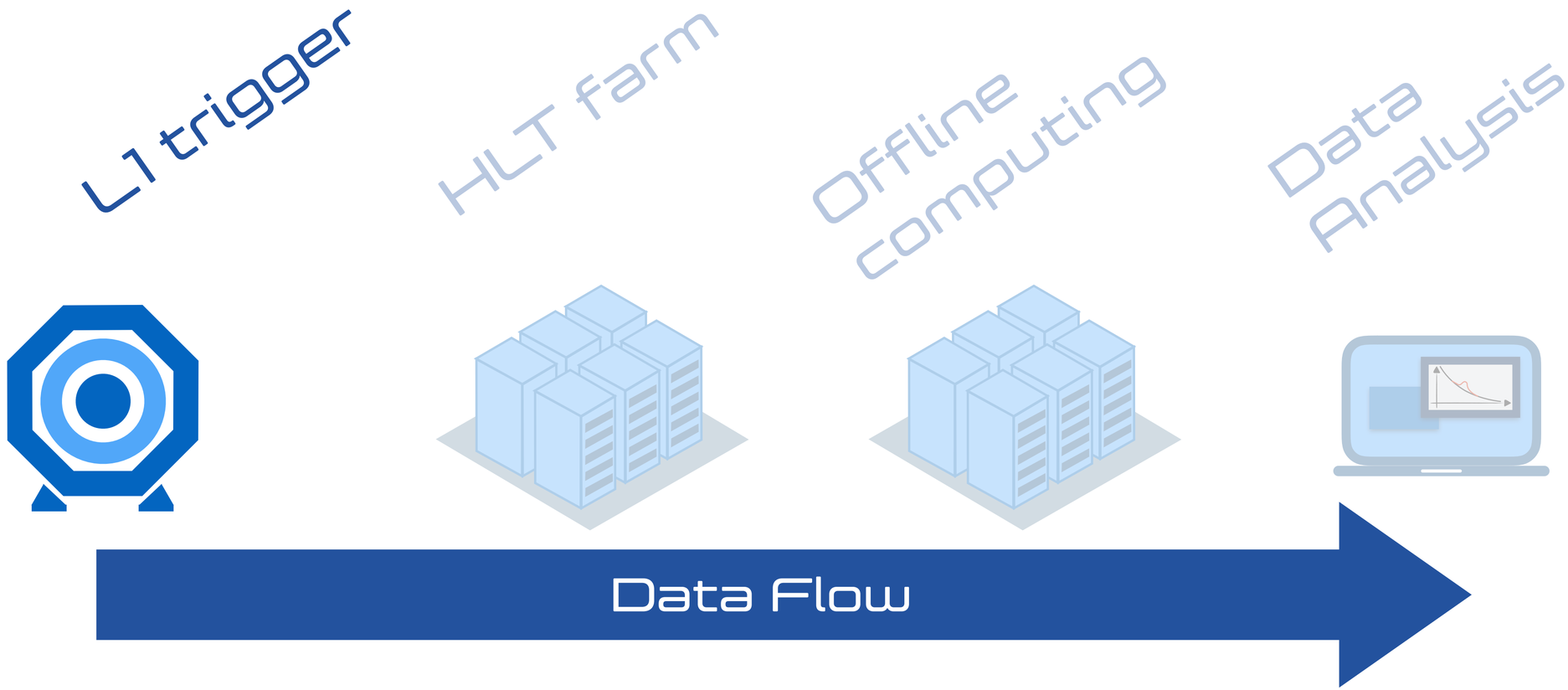
European Research Council

The LHC Big-data problem

- *The LHC generates 40 million collisions every second*
- *The technology to store all these data doesn't exist*
- *Keeping data costs money (for disk, tape, and CPU for processing)*
- *We need to select in real time what we want to keep*
- *Some event is more interesting than other (A BIG THEORETICAL BIAS that we have to pay)*

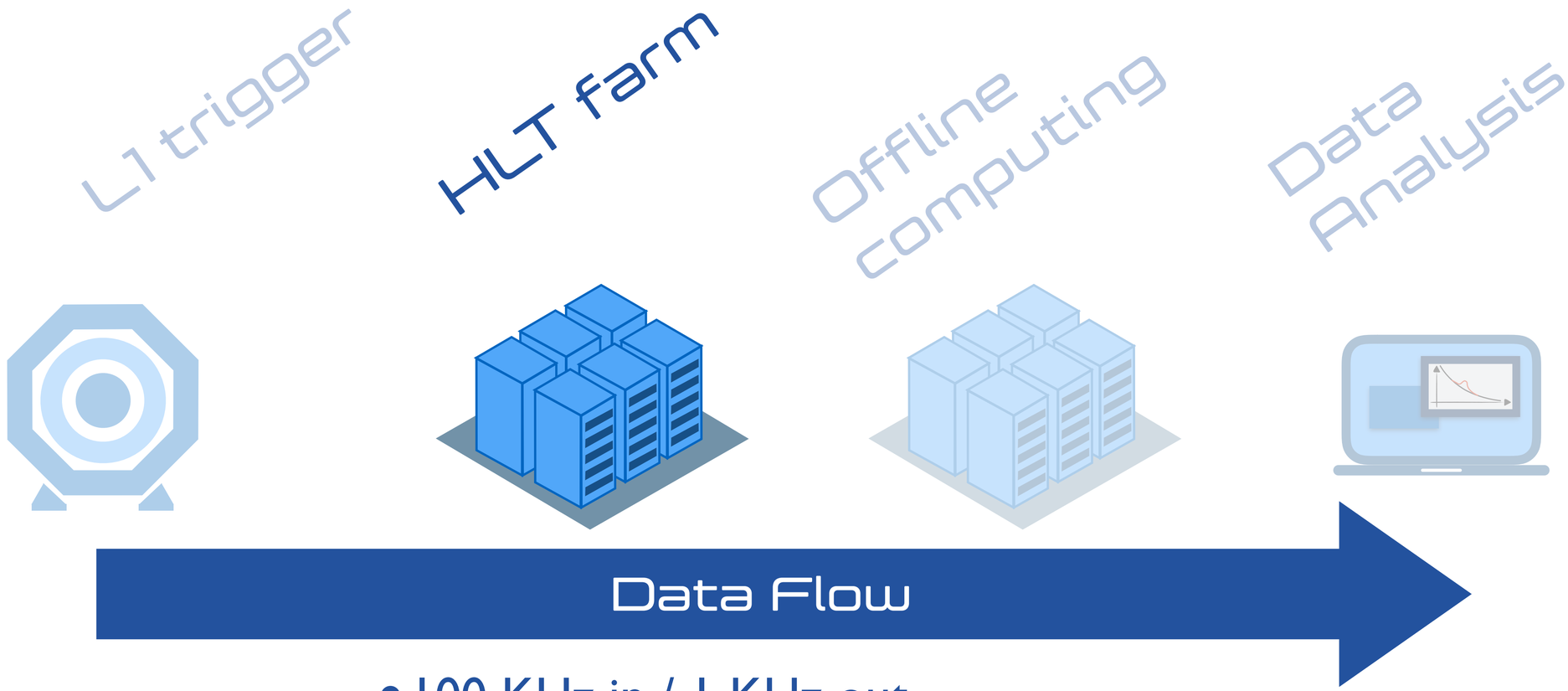
https://www.youtube.com/watch?v=jDC3-QSiLB4

The LHC Big Data problem



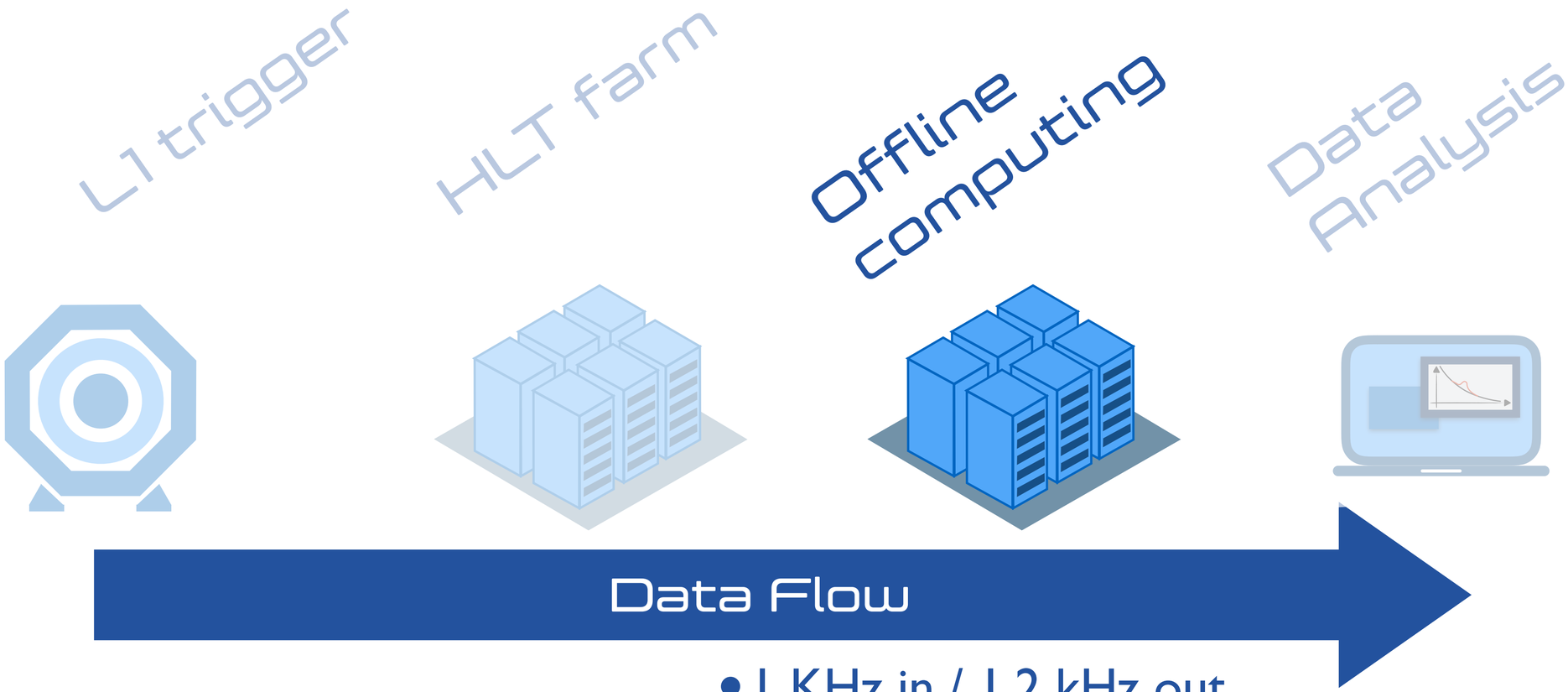
- 40 MHz in / 100 KHz out
- ~ 500 KB / event
- Processing time: ~10 µs
- Based on coarse local reconstructions
- FPGAs / Hardware implemented

The LHC Big Data problem



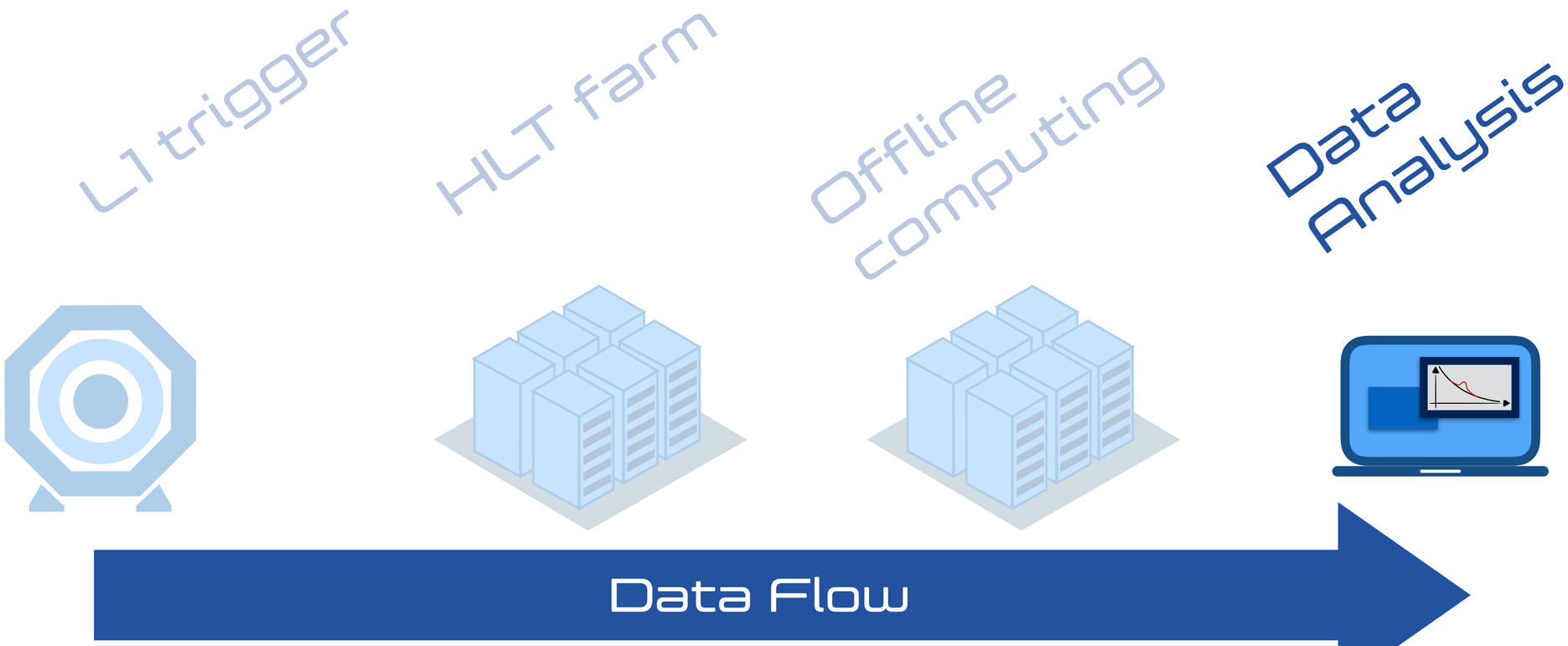
- 100 KHz in / 1 KHz out
- ~ 500 KB / event
- Processing time: ~100 ms
- Based on simplified global reconstructions
- Software implemented on CPUs

The LHC Big Data problem



- 1 KHz in / 1.2 kHz out
- ~ 1 MB / 200 kB / 30 kB per event
- Processing time: ~20 s
- Based on accurate global reconstructions
- Software implemented on CPUs

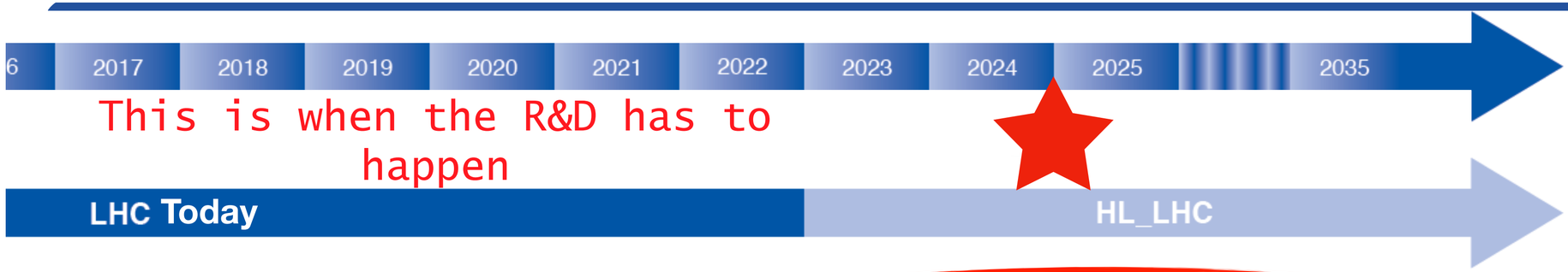
The LHC Big Data problem



- Up to ~ 500 Hz In / 100-1000 events out

- <30 KB per event
- Processing time irrelevant
- User-written code + centrally produced selection algorithms

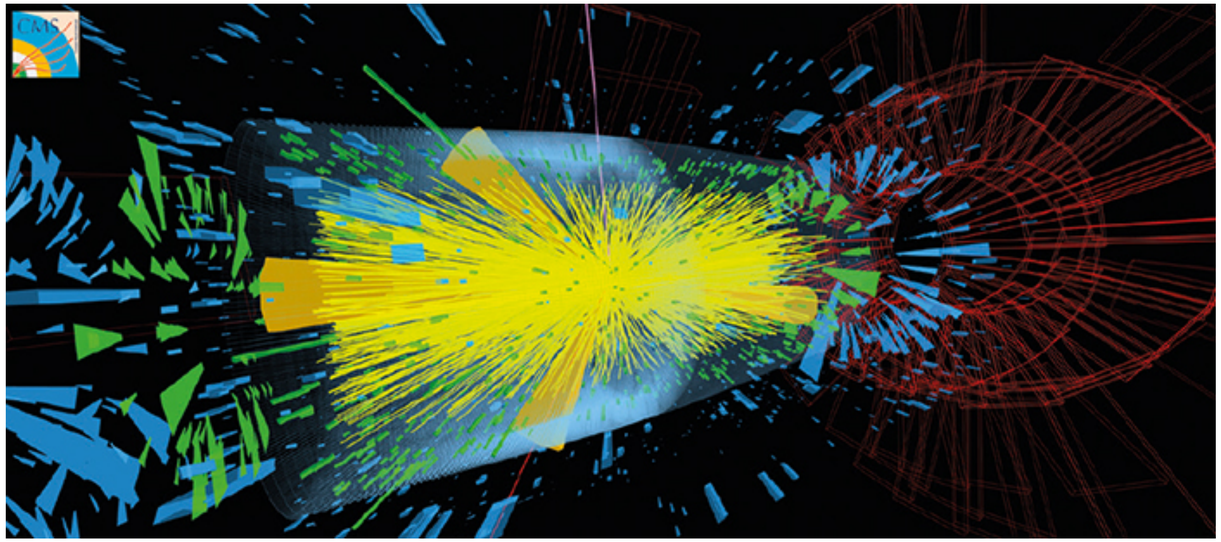
HL-LHC: elephant in the room



- ~40 collisions/event
- ~10 sec/event processing time
- (at best)Same computing resources as today

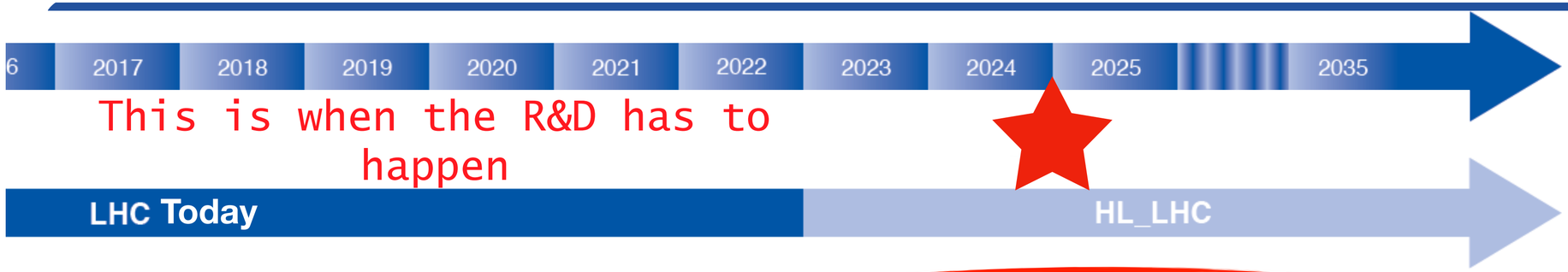
- ~200 collisions/event
- ~minute/event processing time(*)
- (at best)Same computing resources as today

- *Flat budget vs. more needs = **current rule-based reconstruction algorithms will not be sustainable***
- ***Adopted solution:*** *more granular and complex detectors → more computing resources needed → more problems*
- ***Modern Machine Learning might be the way out***



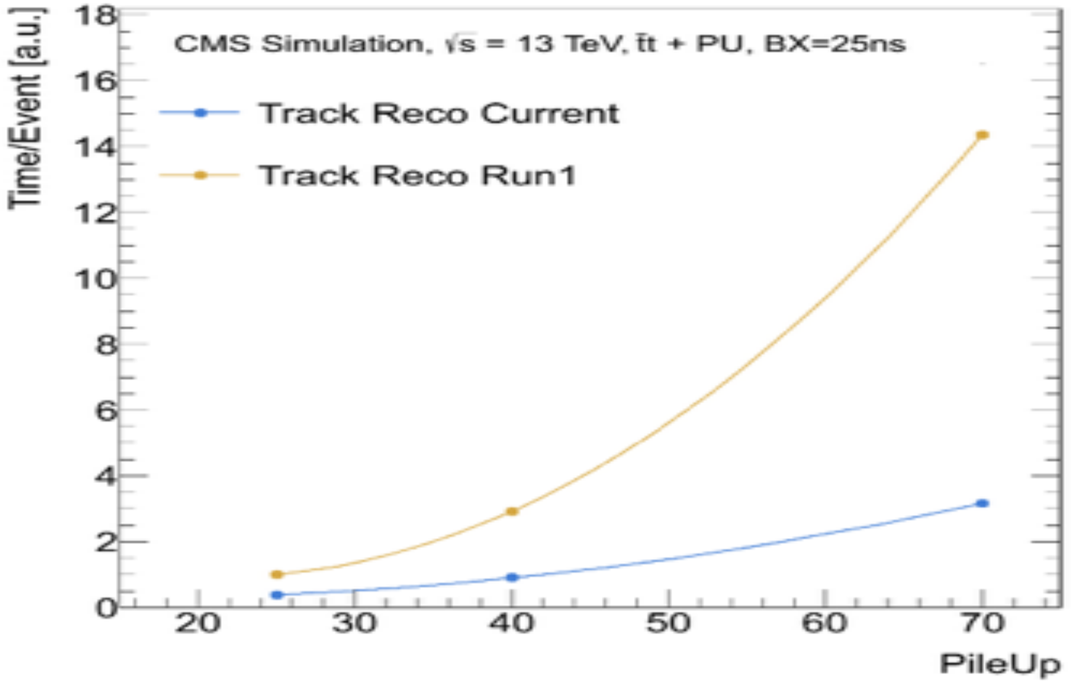
(*)With nowadays software development

HL-LHC: elephant in the room



- ~40 collisions/event
- ~10 sec/event processing time
- (at best)Same computing resources as today

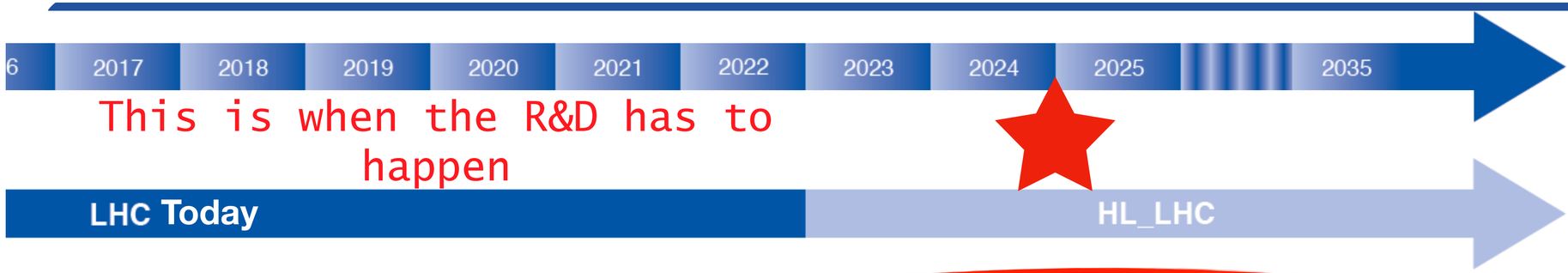
- ~200 collisions/event
- ~minute/event processing time(*)
- (at best)Same computing resources as



- *Flat budget vs. more needs = **current rule-based reconstruction algorithms will not be sustainable***
- ***Adopted solution:*** *more granular and complex detectors → more computing resources needed → more problems*
- ***Modern Machine Learning might be the way out***

(*)With nowadays software development

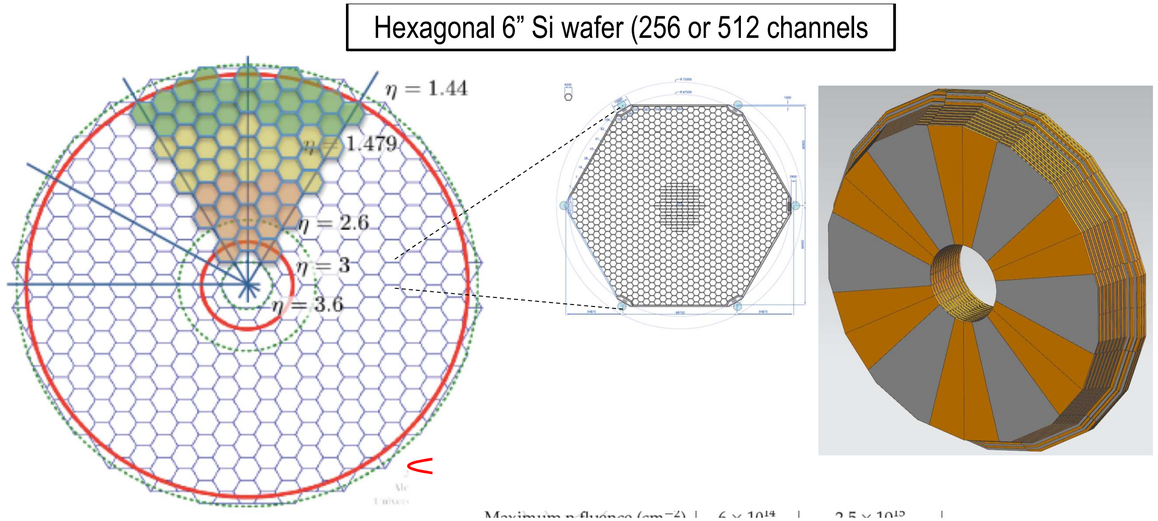
HL-LHC: elephant in the room



- ~40 collisions/event
- ~10 sec/event processing time
- (at best)Same computing resources as today

- ~200 collisions/event
- ~minute/event processing time(*)
- (at best)Same computing resources as today

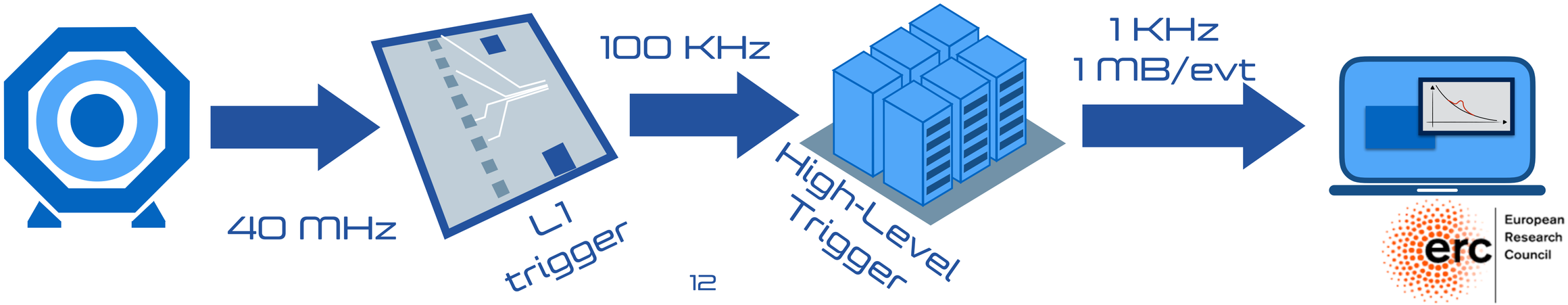
- *Flat budget vs. more needs =* ***current rule-based reconstruction algorithms will not be sustainable***
- ***Adopted solution:*** *more granular and complex detectors → more computing resources needed → more problems*
- ***Modern Machine Learning might be the way out***



(*)With nowadays software development

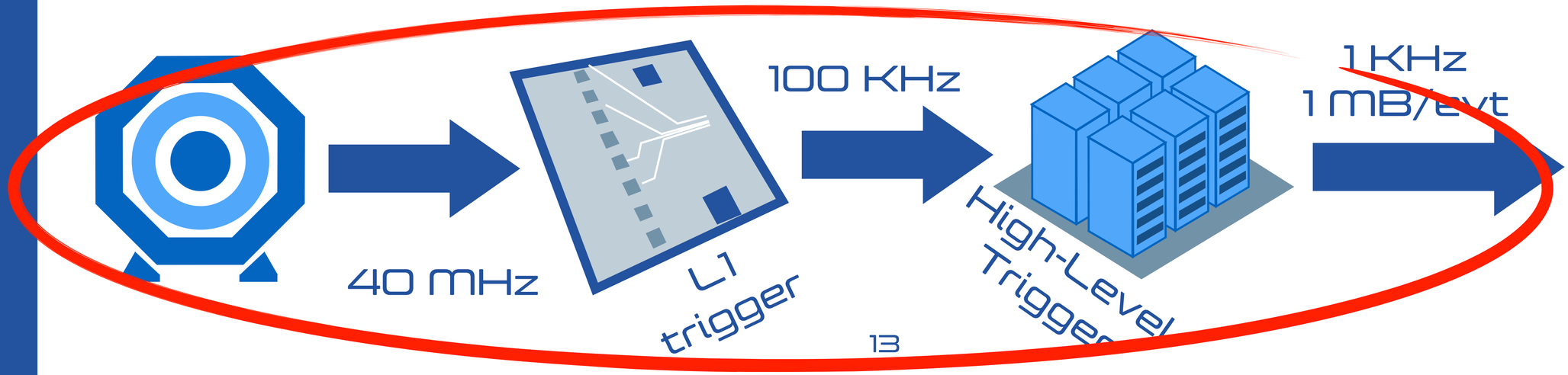
The LHC Big Data Problem

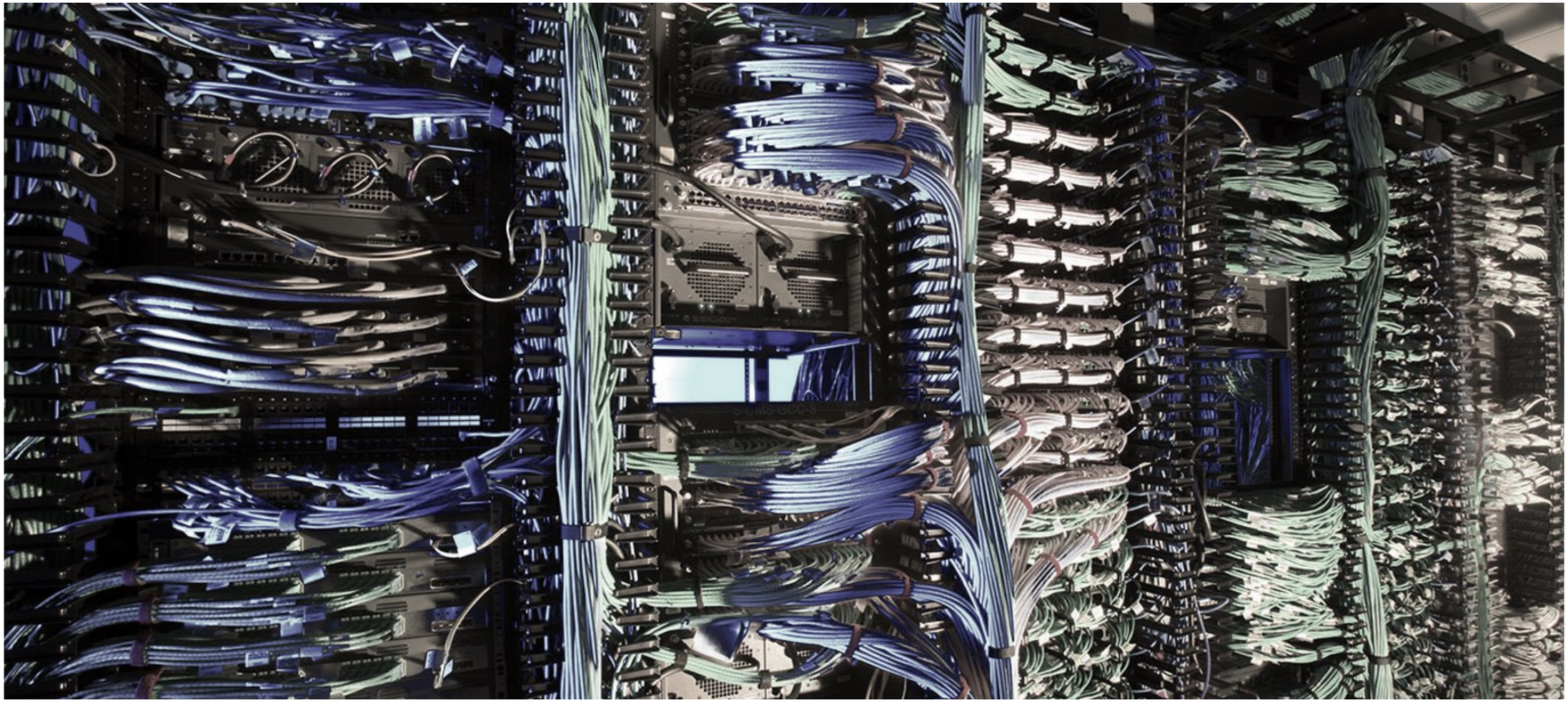
- *Too many data, too large data -> need to filter online*
- *Filters based on theoretical bias: **we might be loosing good events***
  - *L1 trigger: local, hardware based, on FPGA, @experiment site*
  - *HLT: local/global, software based, on CPU, @experiment site*
  - *Offline: global, software based, on CPU, @CERN T0*
  - *Analysis: user-specific applications running on the grid*



The LHC Big Data Problem

- *We are not seeing new physics: just re-doing what we do today with x10 more data WILL NOT be enough.*
- *The solution to the HL-LHC problem: modern Machine Learning as a fast shortcut between the data and the right answer (the outcome of our traditional & slow algorithms)*
- *This ML deployment need to happen in between collisions and data analysis (trigger, reconstruction, …), where freeing resources will make a difference*

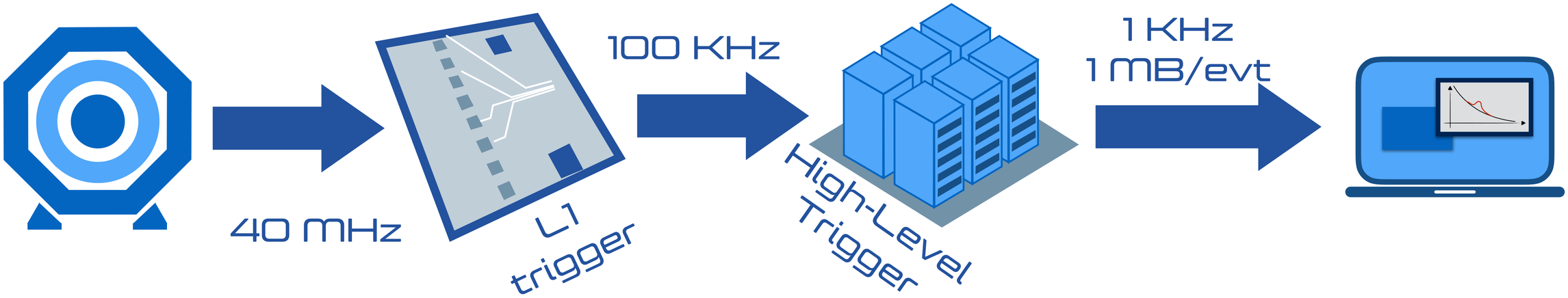




Fast Decision Taking

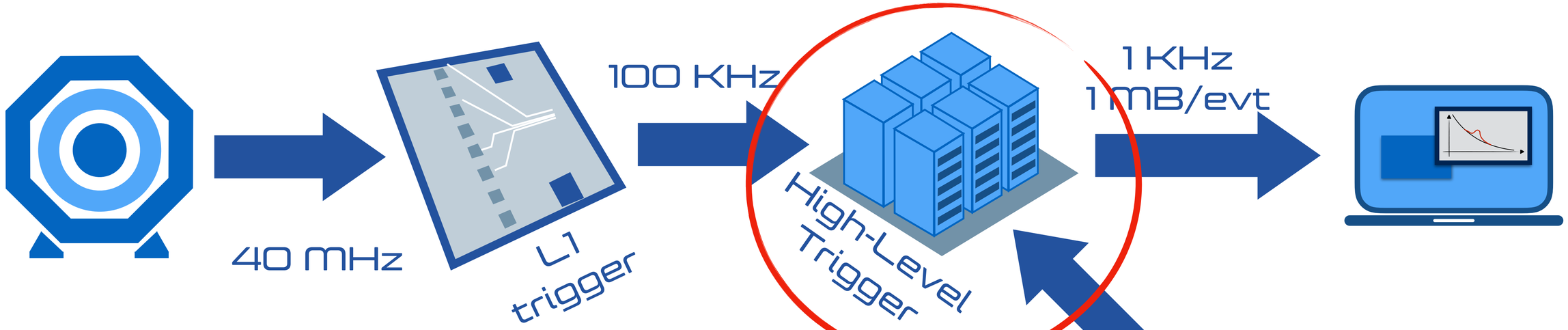
European Research Council

Future Architecture

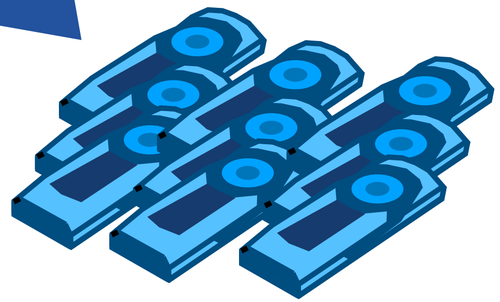


- *The more we use Machine Learning in our centralised reconstruction, the more we need to reconsider the design of our architecture*
  - *new performance/latency balance*
- *(to some extent) Not an issue after the event is written on disk (latency long enough that inference on CPU is OK*
- *This is not true for the trigger*

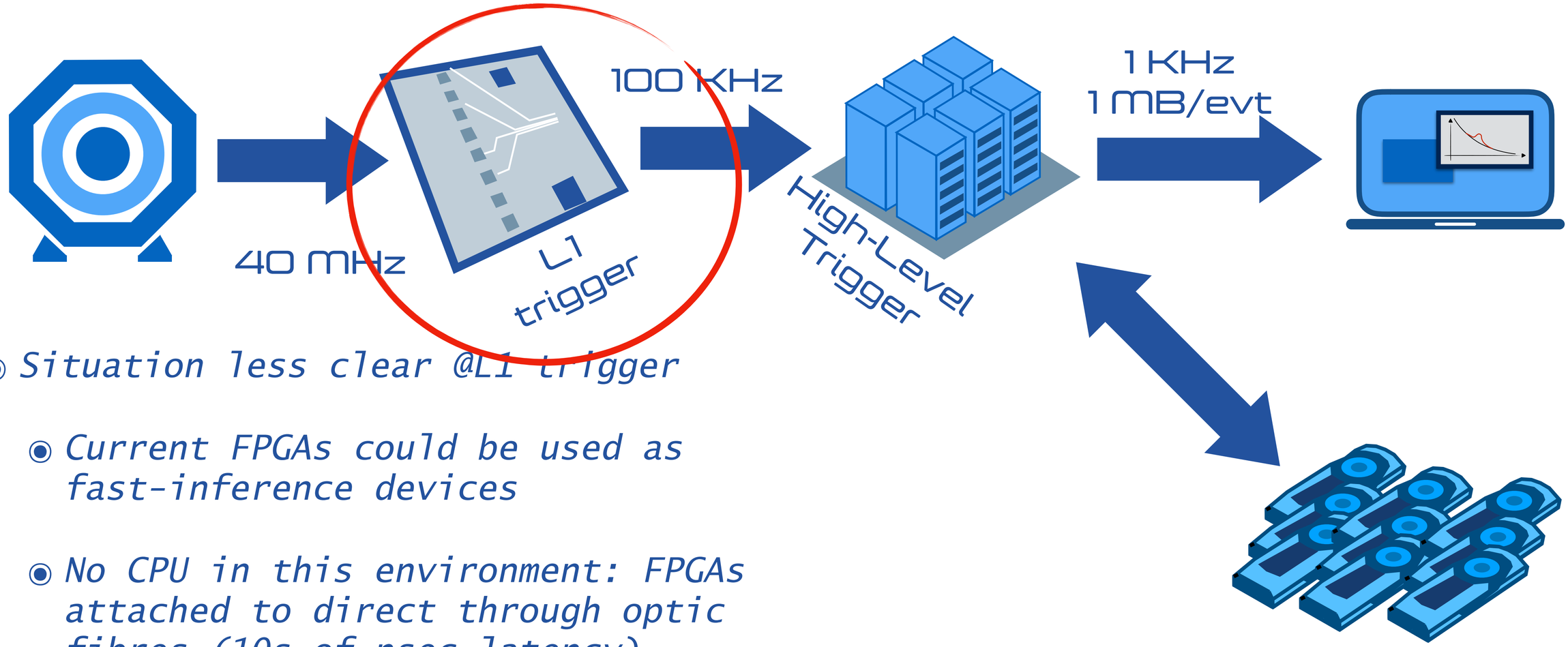
Not just a matter of training



- *Example: the seed-selection for tracking I showed you before*
  - *1 µsec to know if a seed is good or not*
  - *1M seeds/event -> 1sec to process an event serially*
- *This is why we are studying GPU clusters for our HLT farms*
  - *ML inference, tracking, etc delegated to GPUs as a service*

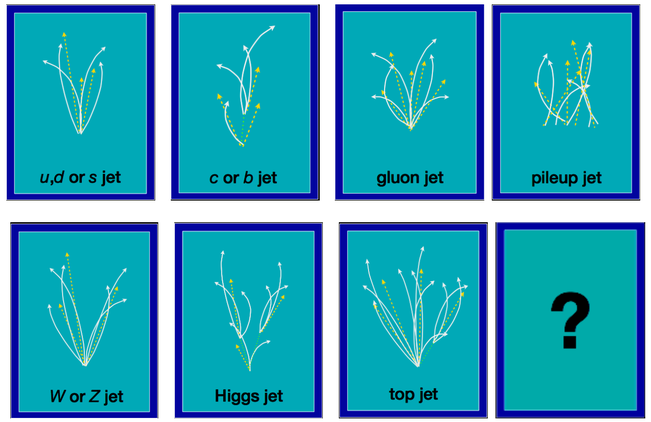


Not just a matter of training



- *Situation less clear @L1 trigger*
  - *Current FPGAs could be used as fast-inference devices*
  - *No CPU in this environment: FPGAs attached to direct through optic fibres (10s of nsec latency)*

Back to our first example



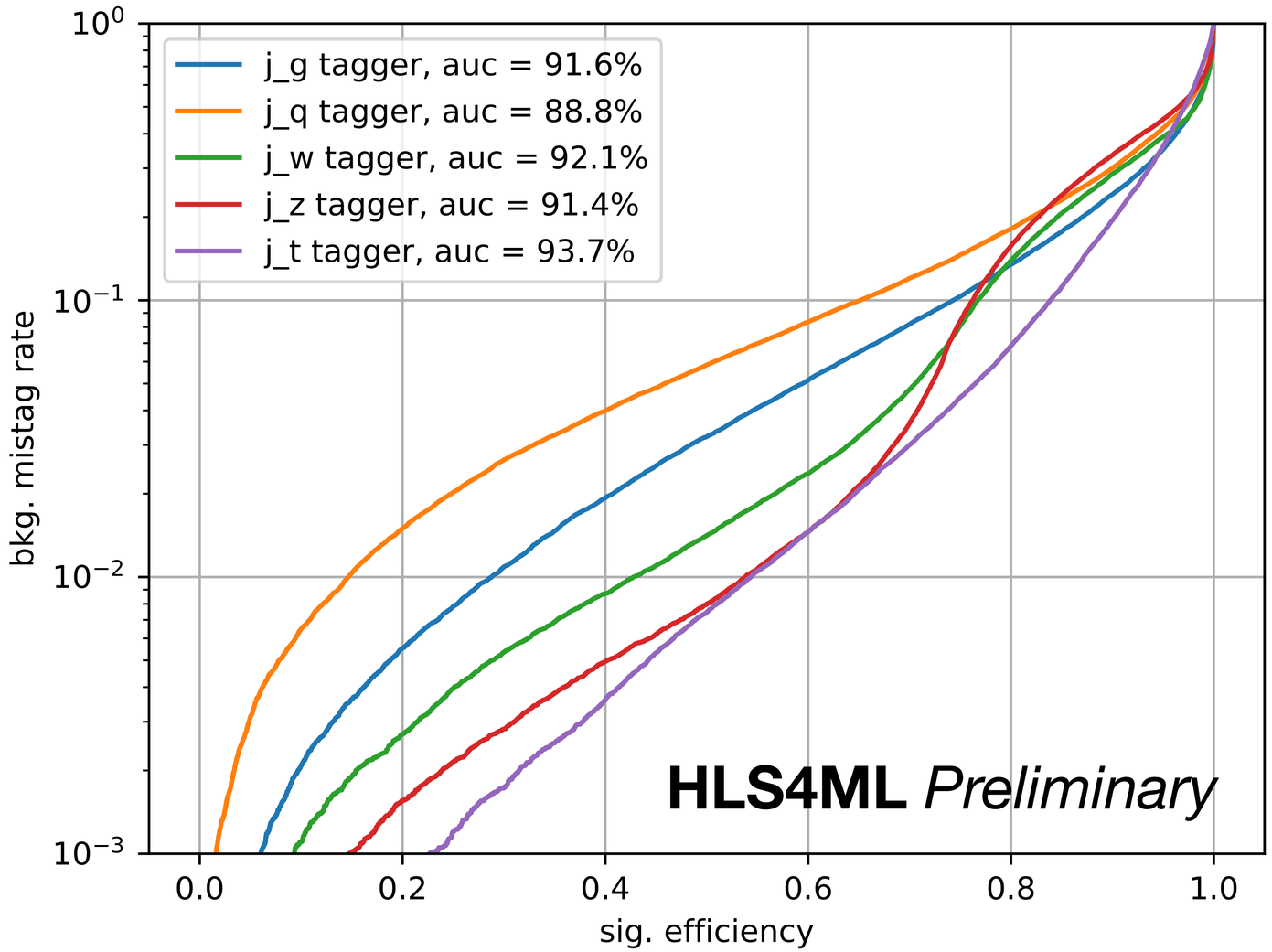
16 inputs

64 nodes
activation: ReLU

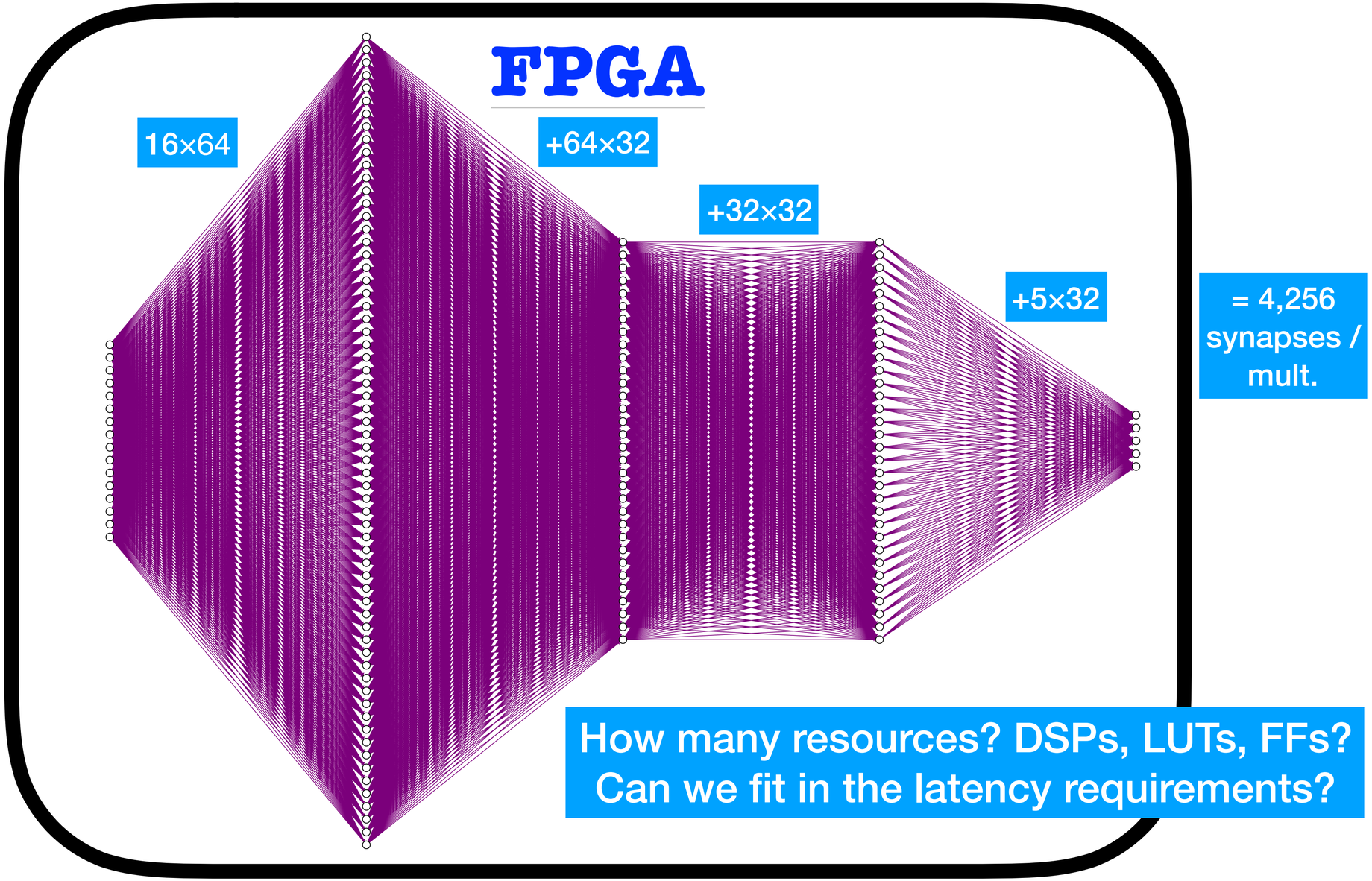
32 nodes
activation: ReLU

32 nodes
activation: ReLU

5 outputs
activation: SoftMax

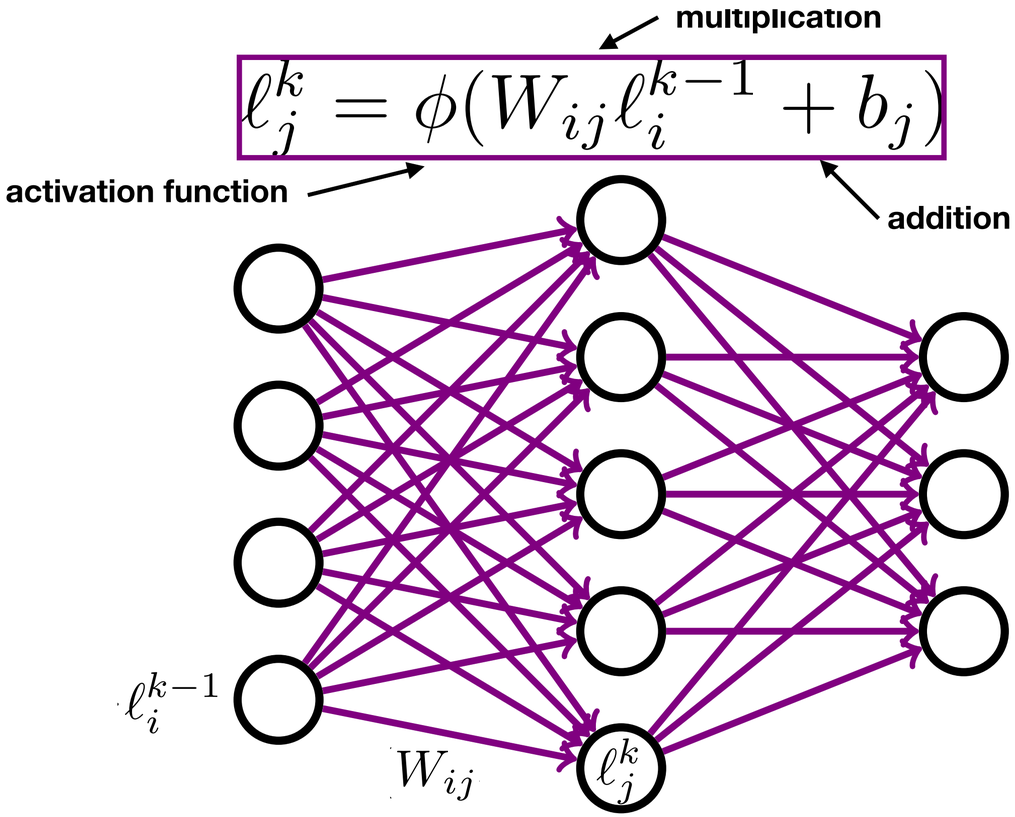


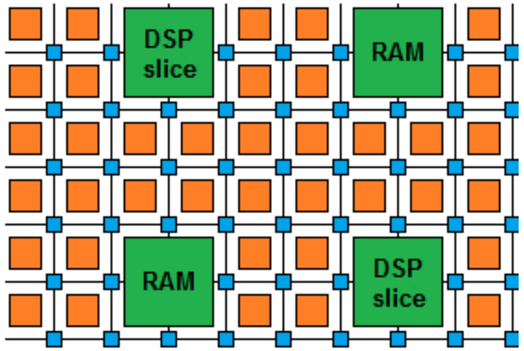
The full model



Network Operations

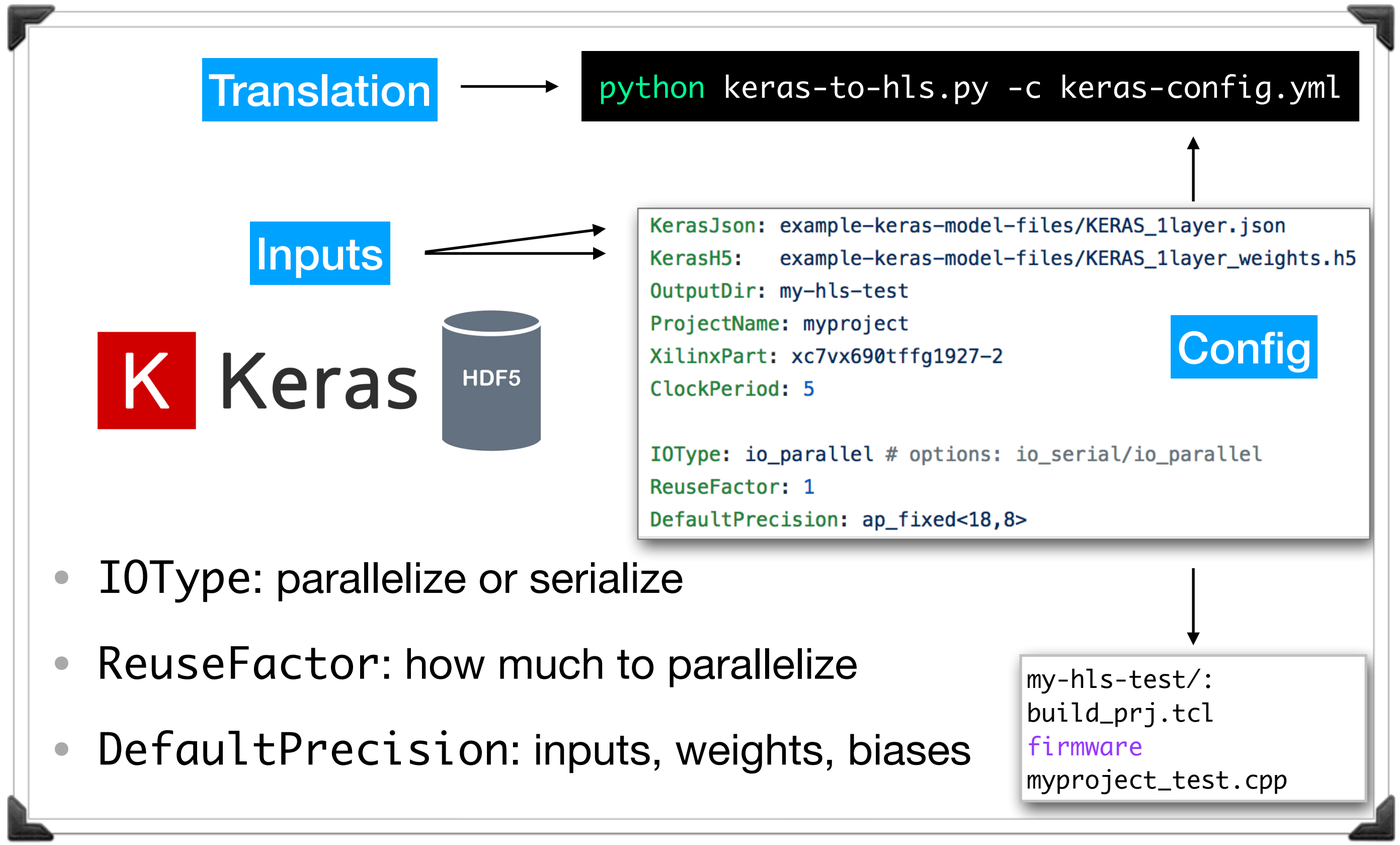
- A classic Dense NN manipulates the inputs in three ways
  - multiplying by weights
  - adding biases
  - applying activation functions
- All these operations map nicely into an FPGA
  - high IO, DSPs, LUTs, tunable precision





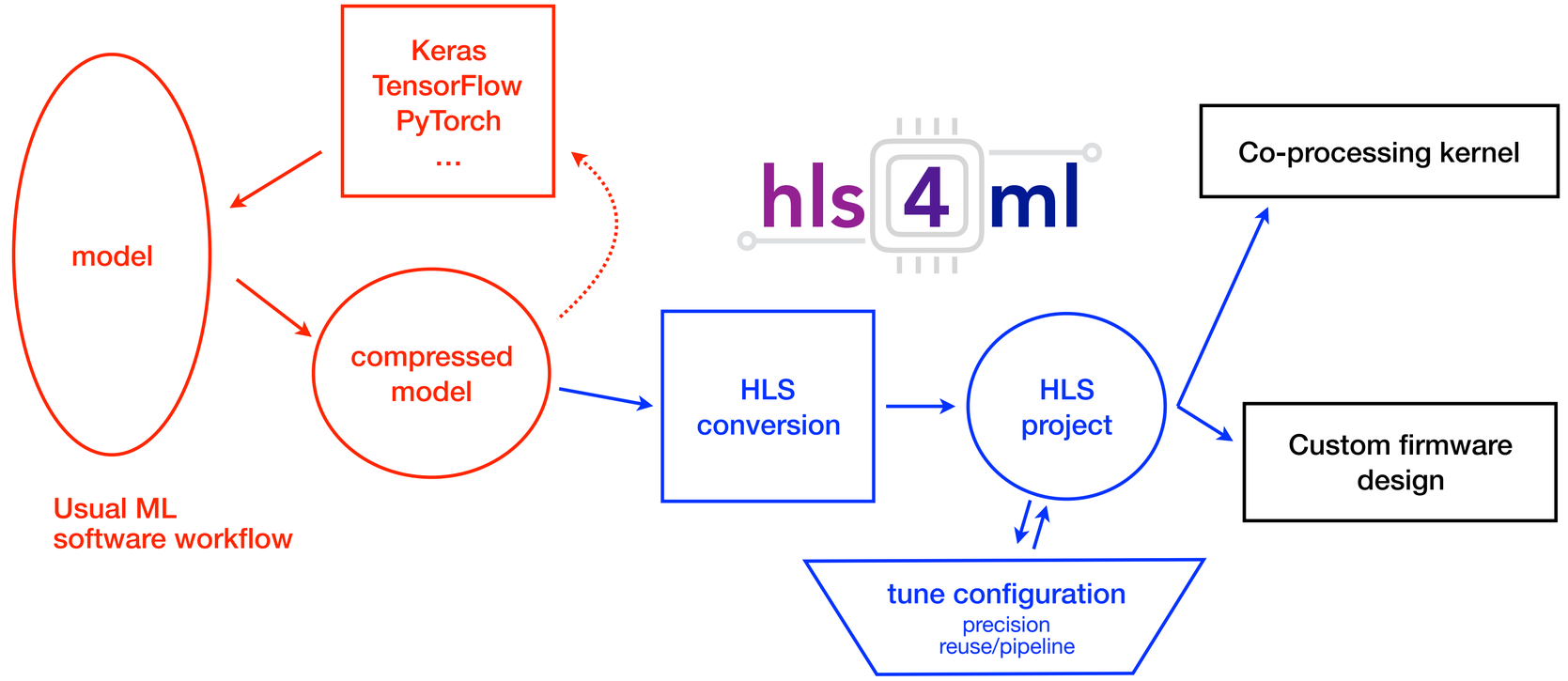
Bring the model to FPGA

- *How this works in practice*
  - *A python based library that takes inputs via a yam file*
  - *Model architecture with supported format*
  - *FPGA configuration parameters (reuse factor, FPGA model, Clock period, etc)*
- *The library provides inputs for Vivado HLS*



HLS4ML

- *HLS4ML aims to be this automatic tool*
  - *reads as input models trained on standard DeepLearning libraries*
  - *comes with implementation of common ingredients (layers, activation functions, etc)*
  - *Uses HLS softwares to provide a firmware implementation of a given network*
  - *Could also be used to create co-processing kernels for HLT environments*

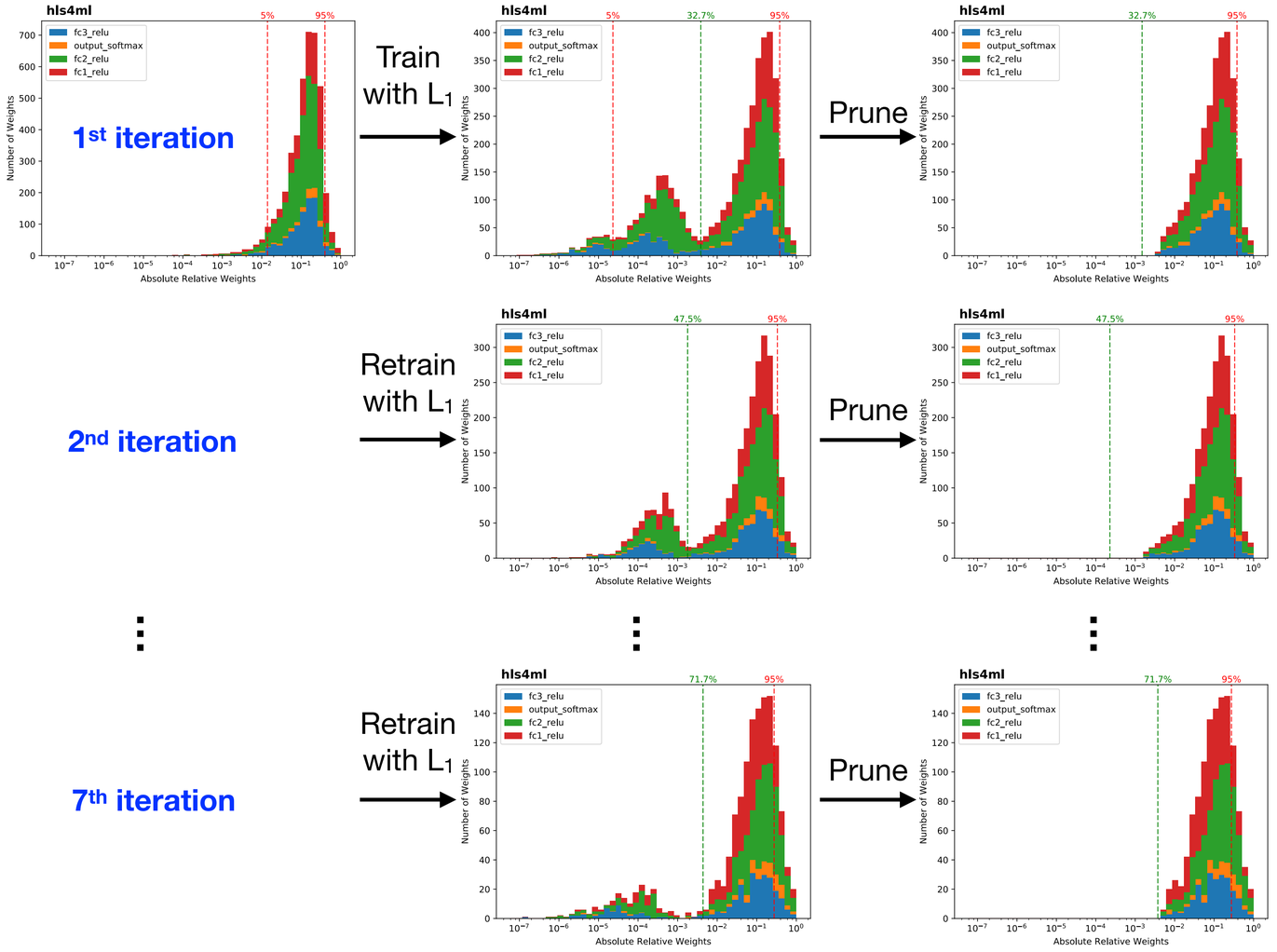


Make the model cheaper

- ***Pruning:*** *remove parameters that don't really contribute to performances*
  - *force parameters to be as small as possible (regularization)*

$$L_\lambda(\vec{w}) = L(\vec{w}) + \lambda||\vec{w}_1||$$

  - *Remove the small parameters*
  - *Retrain*

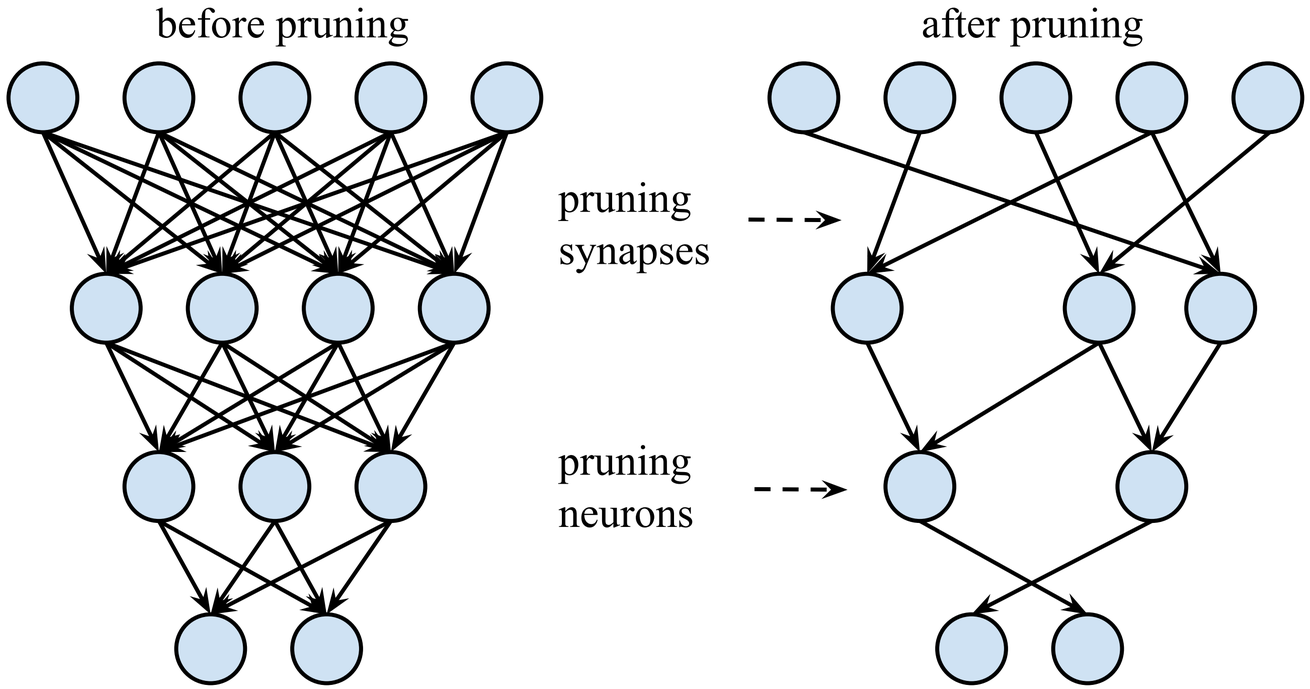


Make the model cheaper

- ***Pruning:*** *remove parameters that don't really contribute to performances*
  - *force parameters to be as small as possible (regularization)*

$$L_\lambda(\vec{w}) = L(\vec{w}) + \lambda||\vec{w}_1||$$

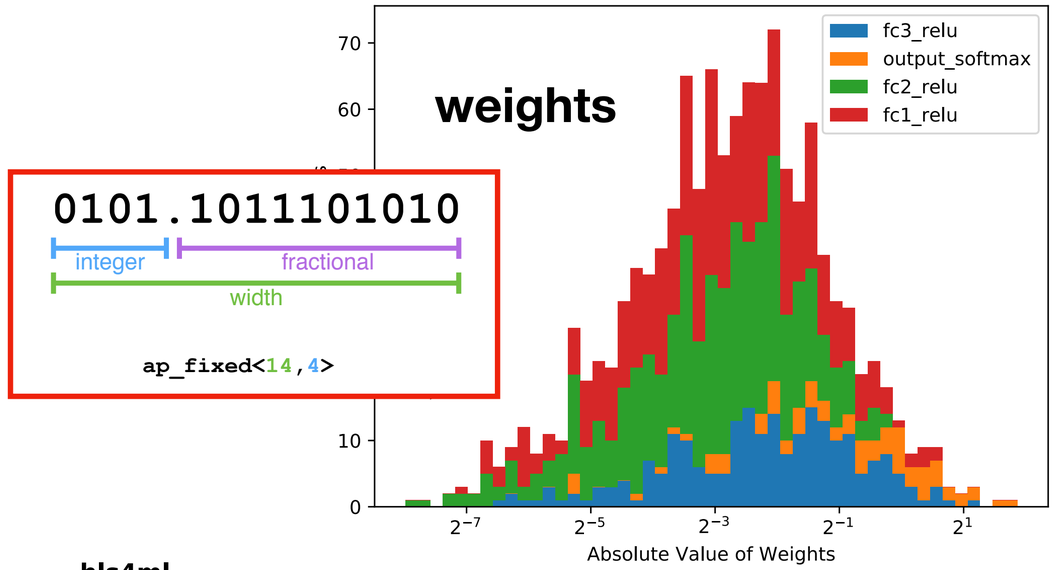
  - *Remove the small parameters*
  - *Retrain*

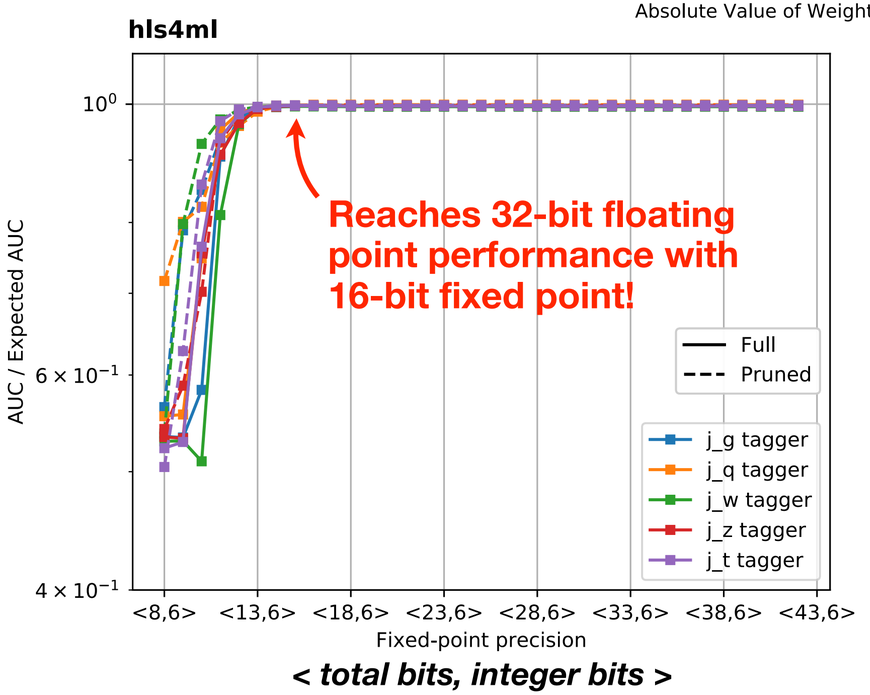


→ 70% reduction of weights and multiplications w/o performance loss

Make the model cheaper

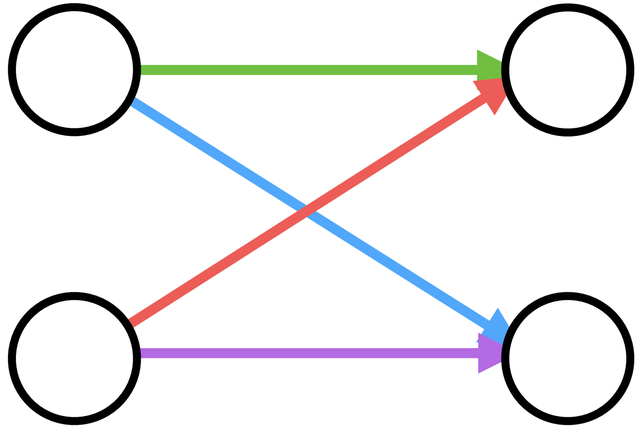
- ***Quantization***: *reduce the number of bits used to represent numbers (i.e., reduce used memory)*
  - *models are usually trained at 64 or 32 bits*
  - *this is not necessarily needed in real life*
- *In our case, we could reduce to 16 bits w/o loosing precision*
- *Beyond that, one would have to accept some performance loss*

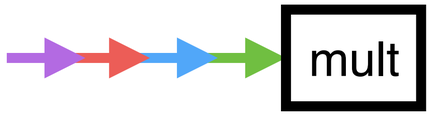




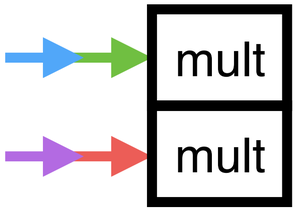
Parallelisation

- `ReuseFactor`: how much to parallelize

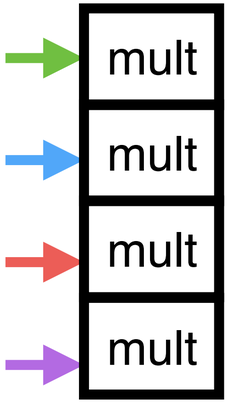




reuse = 4
use 1 multiplier 4 times



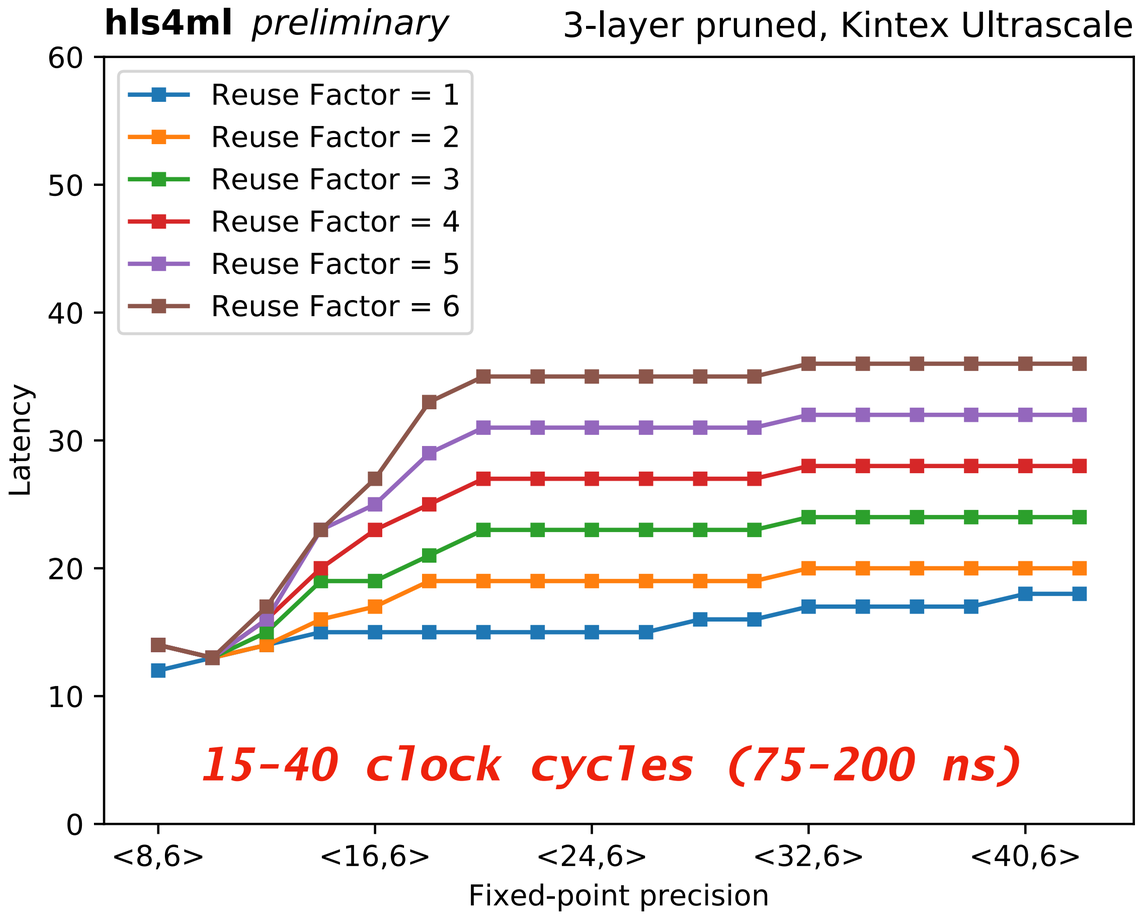
reuse = 2
use 2 multipliers 2 times each

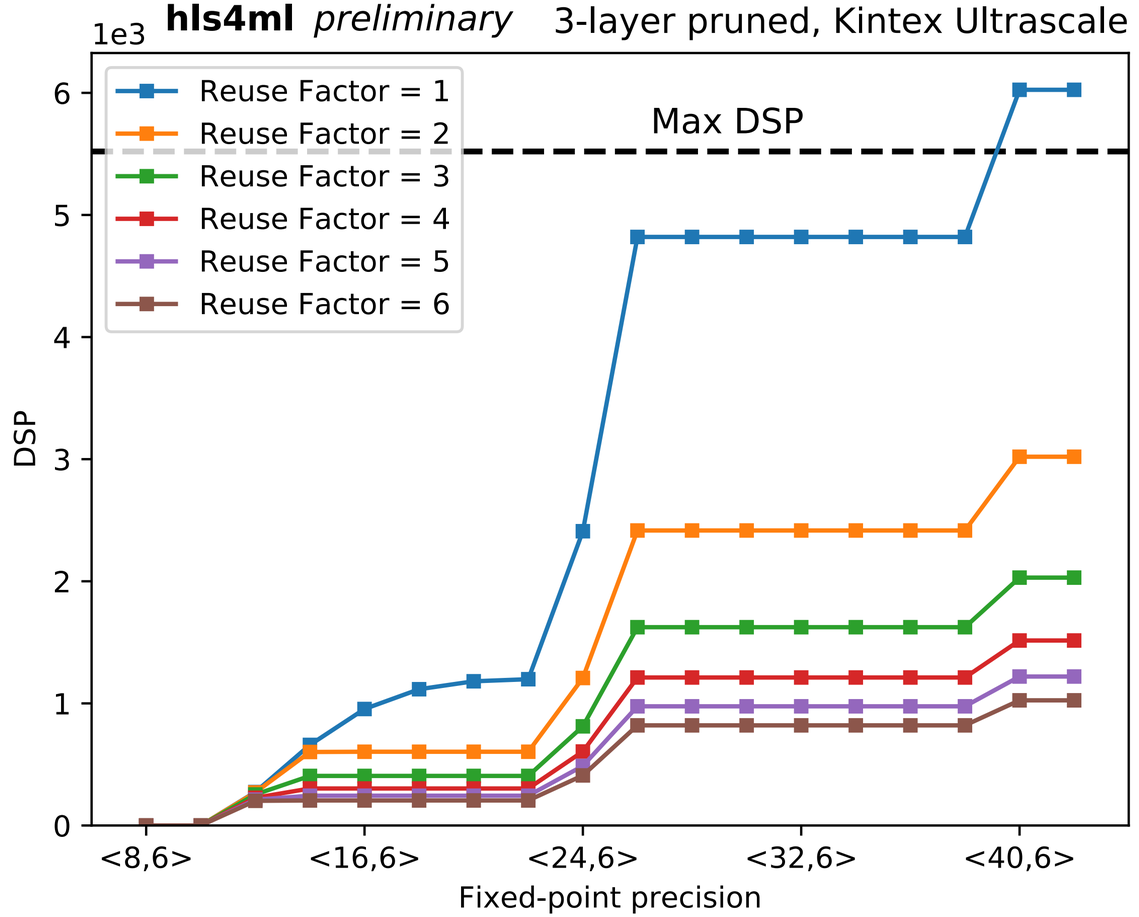


reuse = 1
use 4 multipliers 1 time each

related to the Initiation Interval = when new inputs are introduced to the algo.

Fast-inference





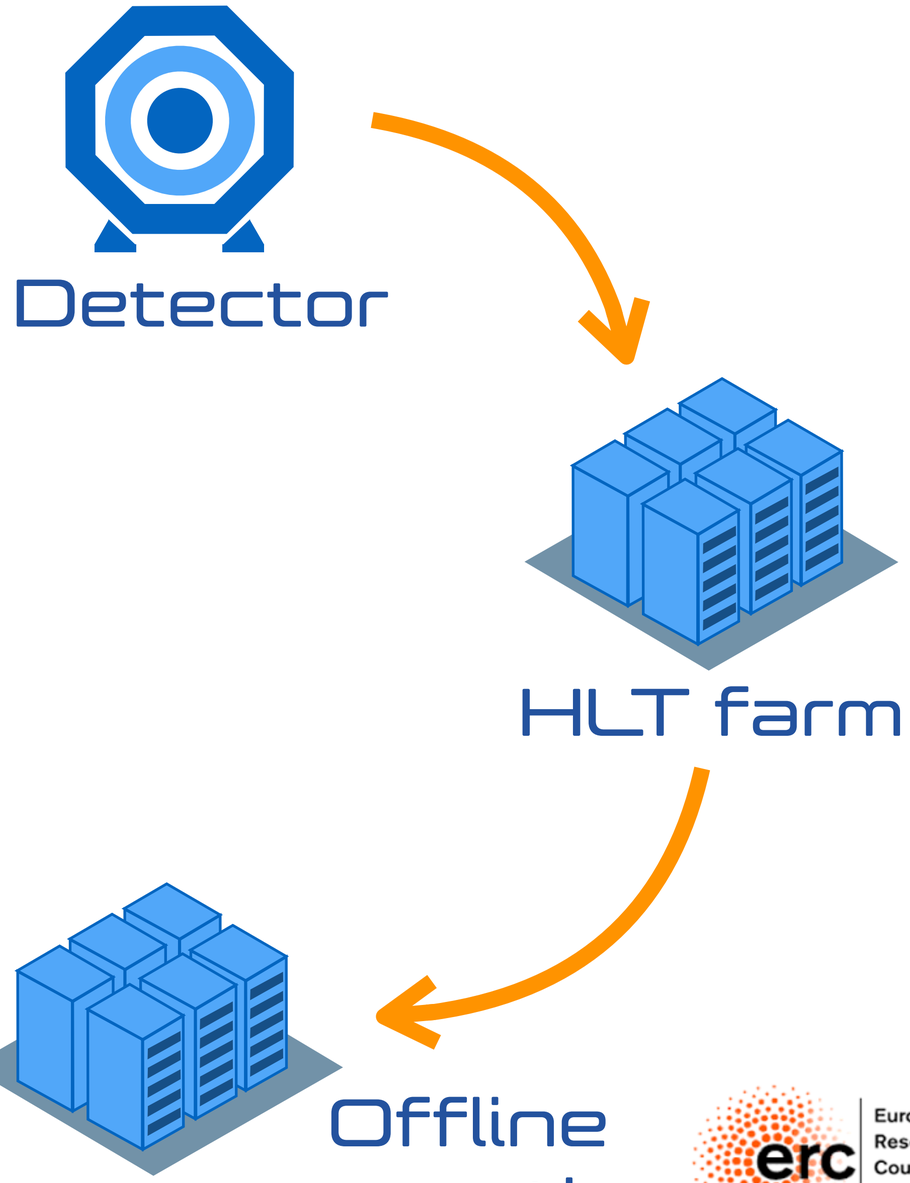
Foreseen architecture (FPGAs) will handle these networks
Inference-optimized GPUs could break the current paradigm
Looking forward to R&D projects with nVidia & E4 on this

Deep Learning on the Cloud

European Research Council

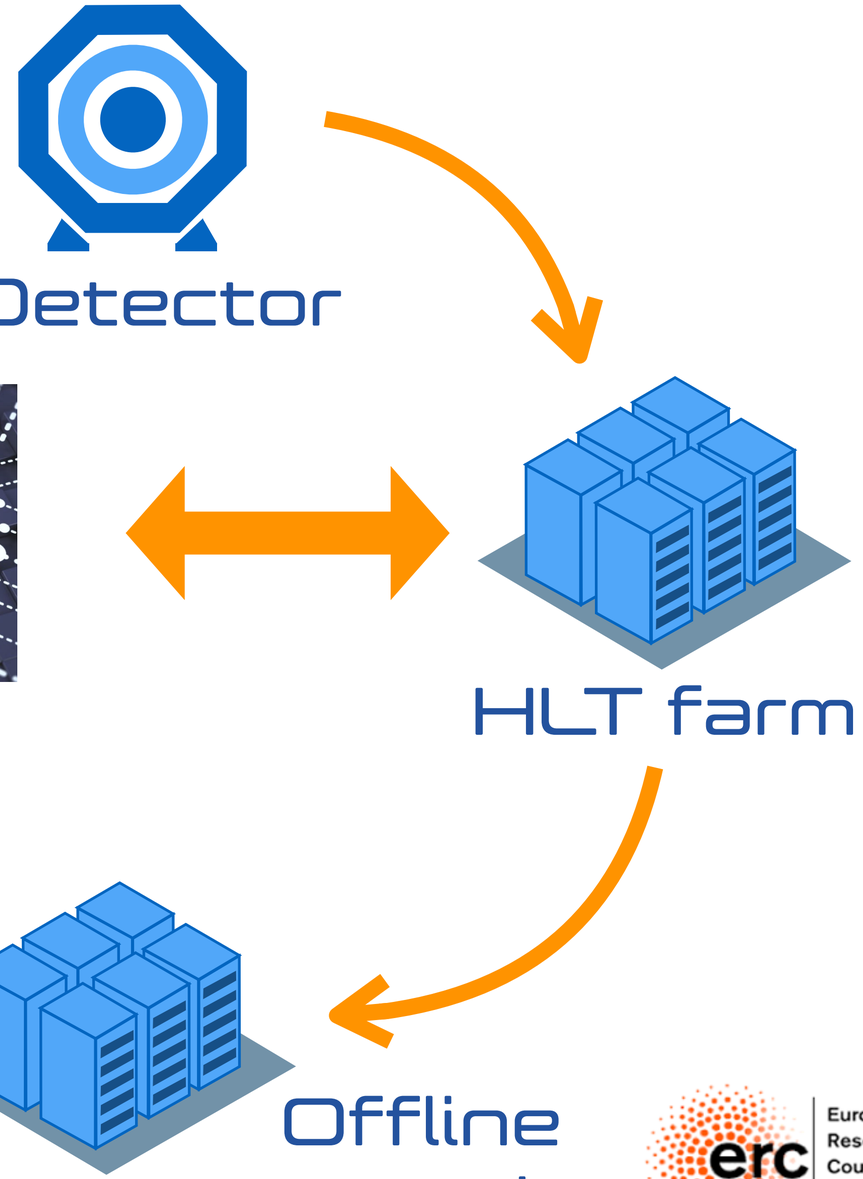
Heterogenous DAQ

- *In the (near) future, DAQ/HLT farms will be based on edge heterogenous computing*
  - *CPU+GPU / CPU+FPGA*
  - *Mainly to accelerate slow algorithms (e.g., tracking) through parallelisation*
  - *Also useful for ML inference*
- *Technology improving quickly*
  - *typically, our systems built ahead and old when operating (e.g., CMS FPGAs)*
  - *can we change the paradigm to take advantage of the improving technology?*



Heterogenous DAQ

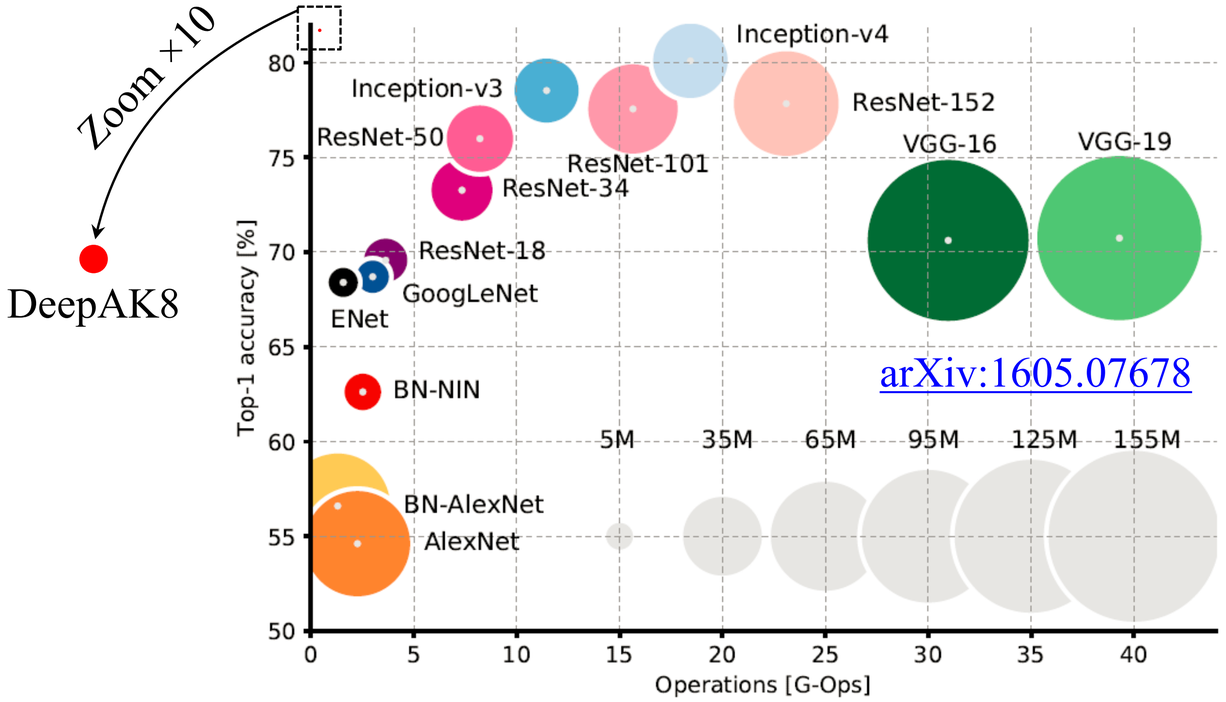
- *One could instead use opportunistically cloud resources as part of the online processing system*
  - *price decreasing (could be competitive in the future)*
  - *always the best hardware at hand*



European Research Council

HEP & Modern Deep Learning

- *Deep Learning is becoming ubiquitous*
  - *used in many applications, mainly image recognition*
  - *large models deployed, reaching large accuracy on multiple tasks*
  - *optimized implementations of these models are deployed on commercial clouds*
- *Can we take advantage of that?*



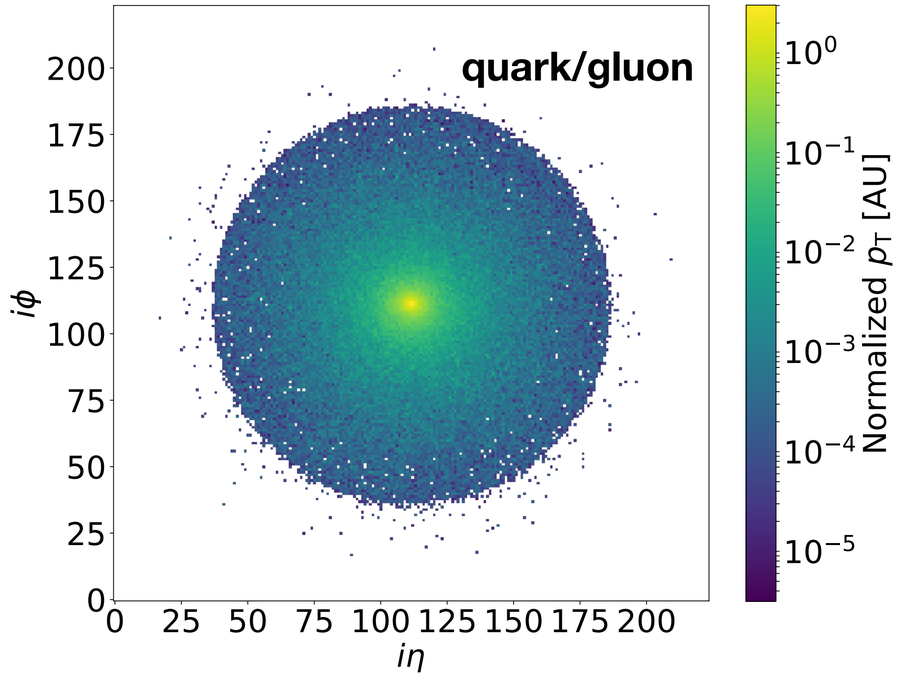
arXiv:1605.07678

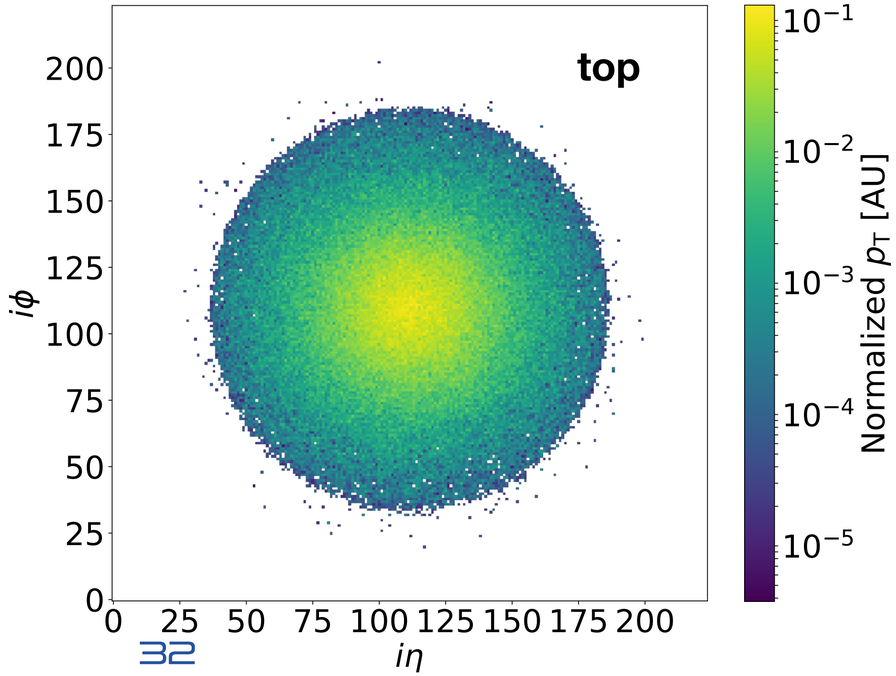
- ResNet50: 25M parameters, 7B operations
- Examples of large networks used in CMS:
  - DeepAK8, 500K parameters, 15M operations
  - DeepDoubleB, 40K parameters, 700K operations

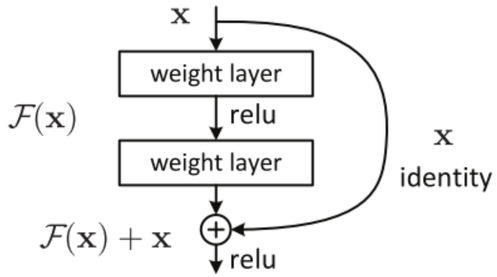
European Research Council

Jet Tagging with ResNet on Cloud

- *To exploit cloud services, we need to turn our problems in something similar to what is done there*
  - *mainly image identification*
  - *computing vision successfully exploited for jet tagging*
- *We investigated the use of a standard network (ResNet) for top tagging*



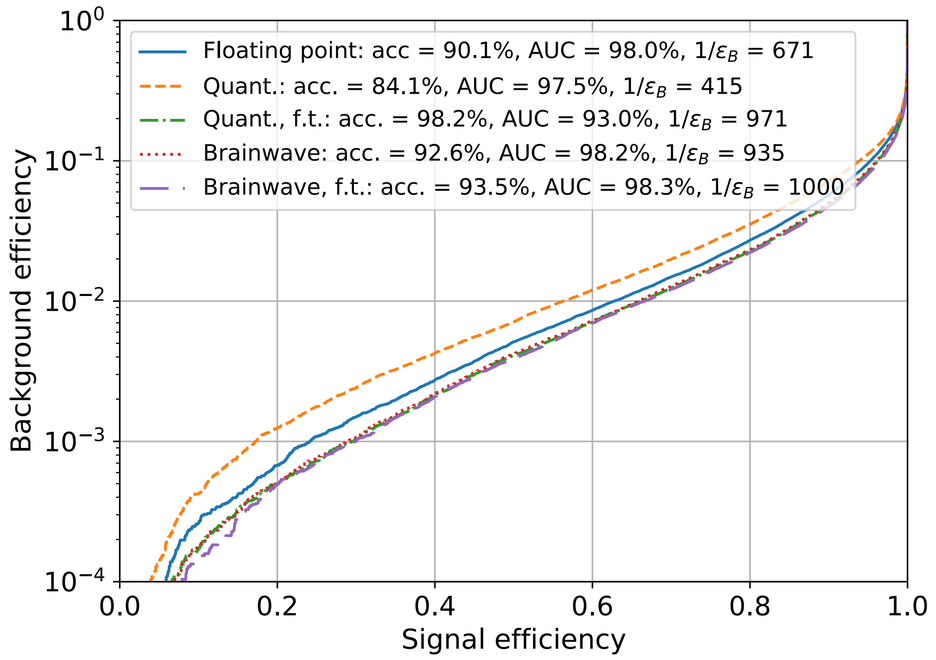




Jet Tagging with ResNet on Cloud

- *To exploit cloud services, we need to turn our problems in something similar to what is done there*
  - *mainly image identification*
  - *computing vision successfully exploited for jet tagging*
- *We investigated the use of a standard network (ResNet) for top tagging*

Model	Accuracy	AUC	$1/\varepsilon_B(\varepsilon_S = 30\%)$
Floating point	0.9009	0.9797	670.8
Quant.	0.8413	0.9754	414.6
Quant., f.t.	0.9296	0.9825	970.7
Brainwave	0.9257	0.9821	934.8
Brainwave, f.t.	0.9348	0.9830	999.6



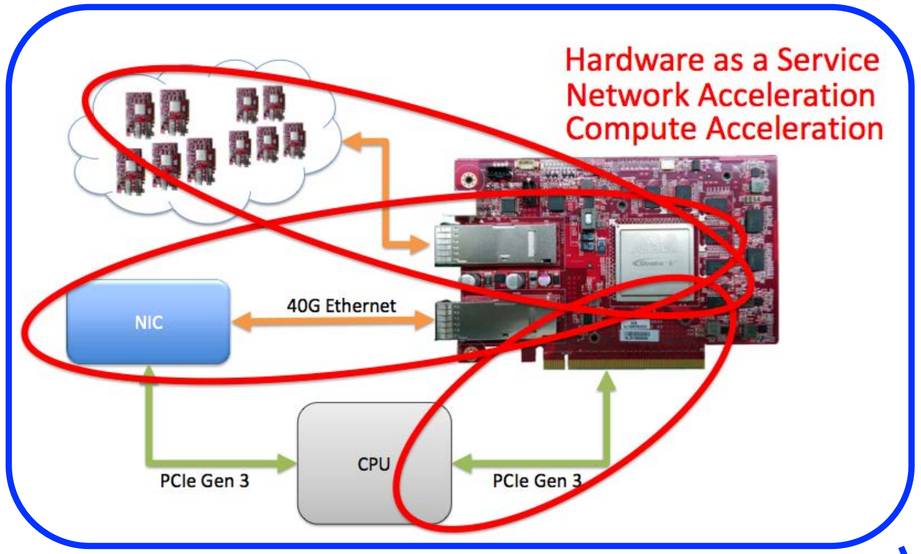
European Research Council

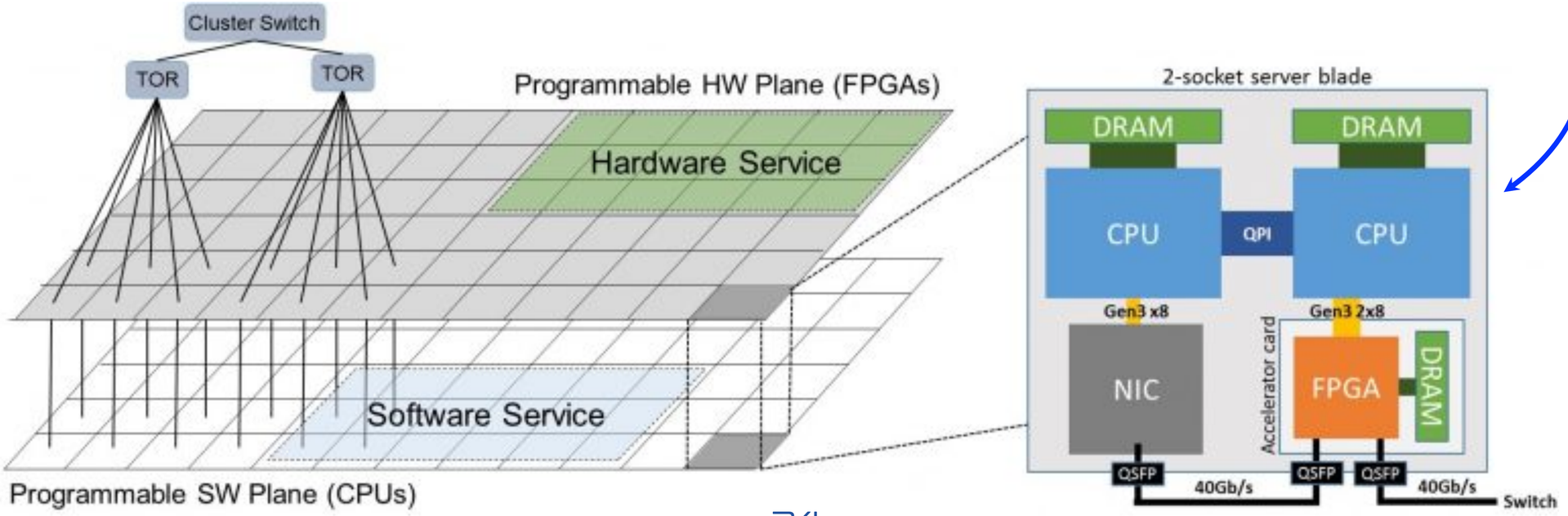
Brainwave at scale

- Provides a full service at scale (more than just a single co-processor)
- Multi-FPGA/CPU fabric, accelerating both compute and network

➢ Demonstrated large improvements in processing time for Bing searches

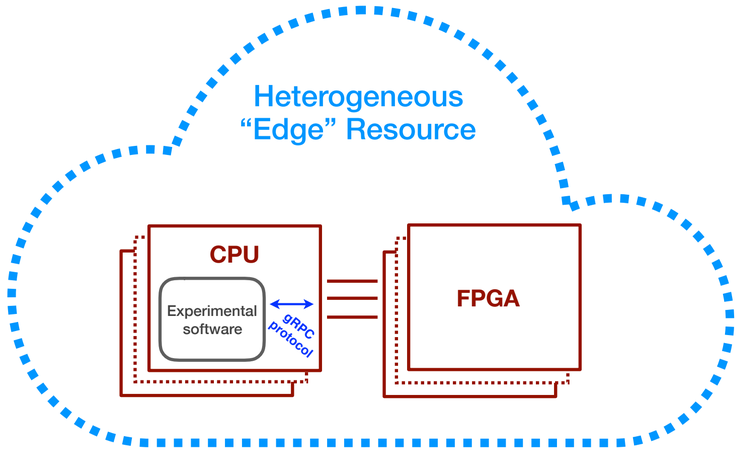
- Caveat: only selected pre-trained DNN models currently available

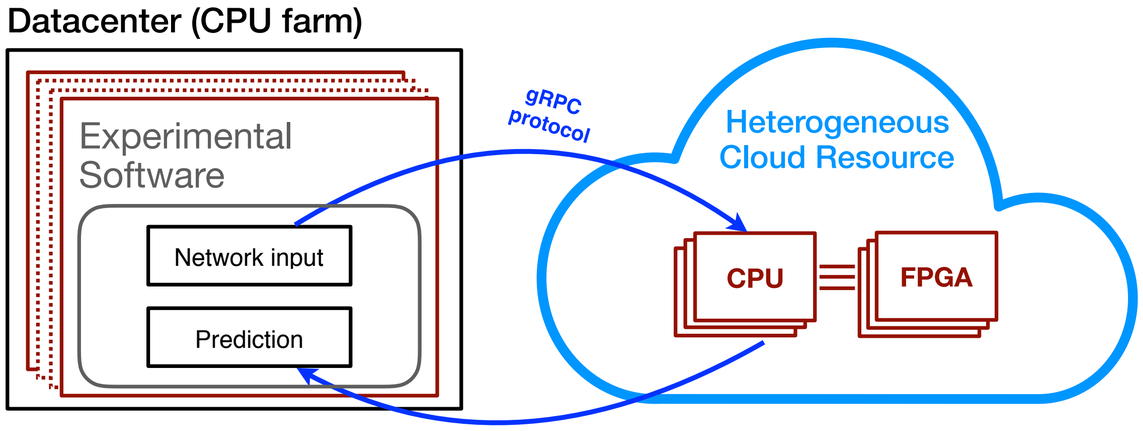




Edge vs Cloud

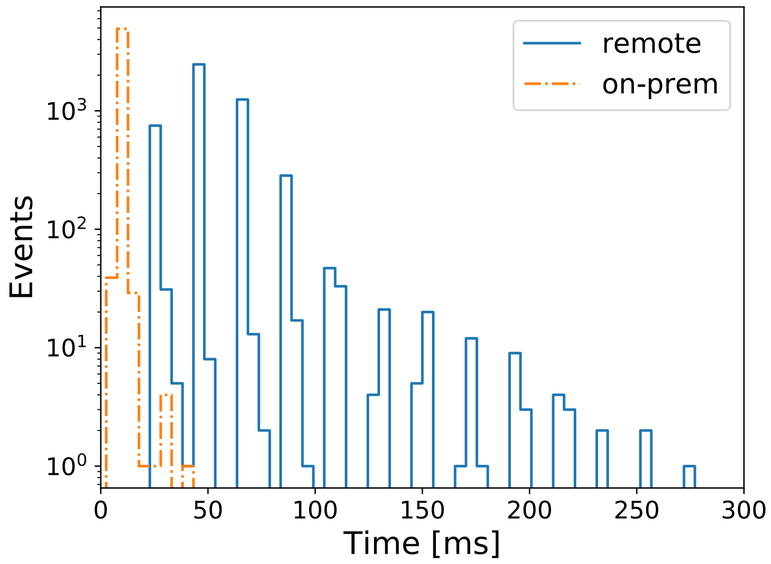
- *On Brainwave, we emulated the edge and the cloud options on the same hardware*
  - ***edge (on prem):*** *there, we emulated a CPU+FPGA infrastructure*
  - ***cloud (remote):*** *at FNAL, we emulated a CPU-only infrastructure, communicated to Brainwave (in Virginia)*
  - *communication via gRPC protocol (driven by Microsoft infrastructure boundaries)*
- *We then compared these two solutions against alternative options (e.g., GPU acceleration)*

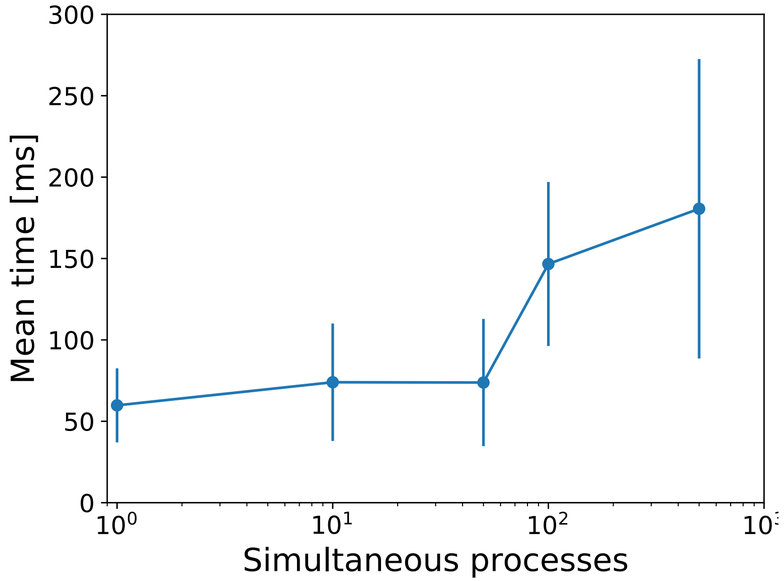


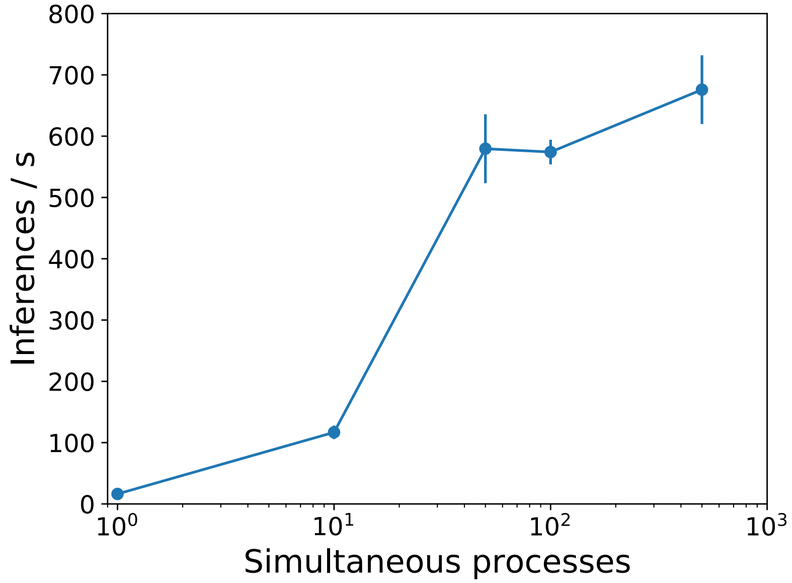


European Research Council

Performance of Remote Solution



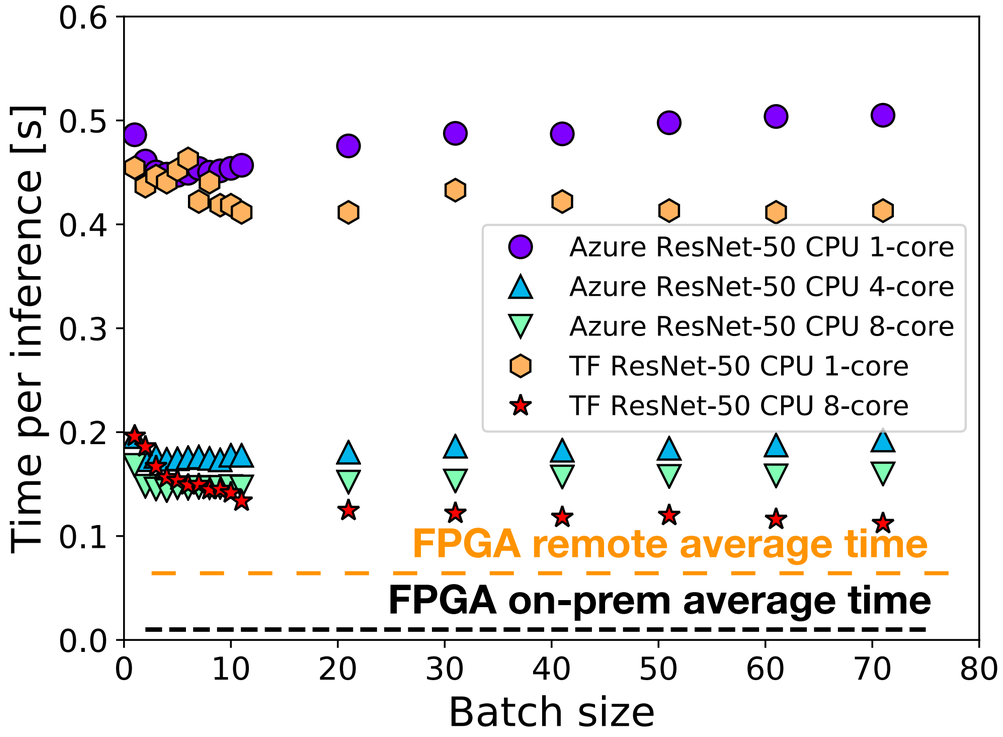


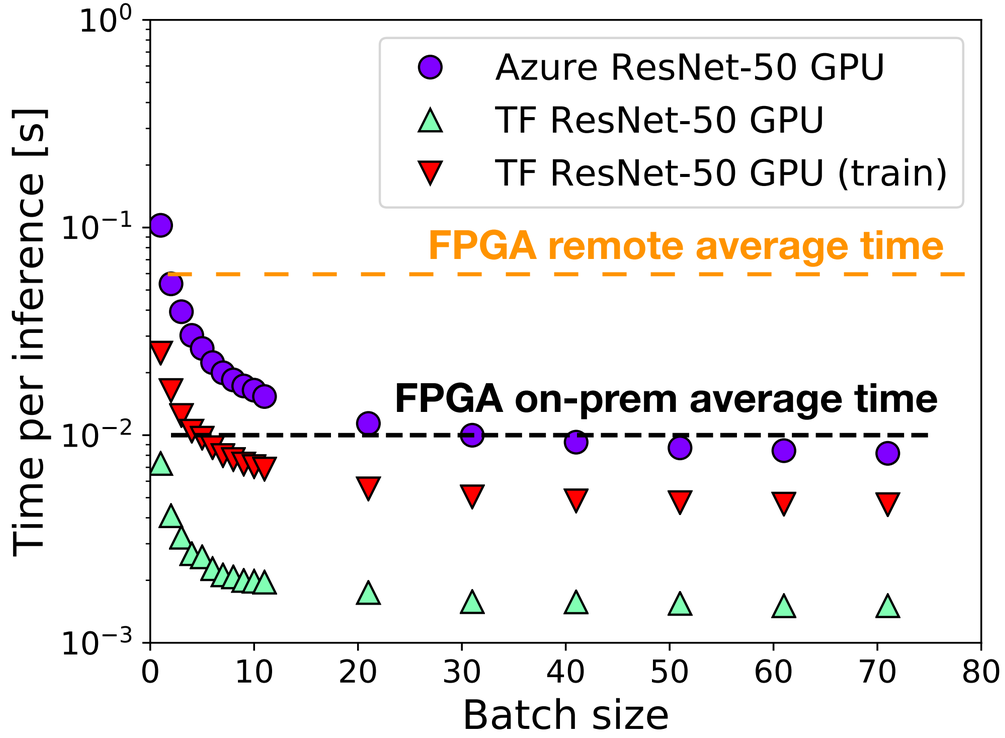


- Good performance in initial tests
  - “remote”: cmslpc @ FNAL to Azure (VA), ‹time› = 56 ms
  - “onprem”: run CMSSW on Azure VM, ‹time› = 10 ms
    (~2 ms on FPGA, rest is classifying and I/O)
  - CPU (cmslpc): 1.75 sec
    (6 min to load ResNet50 session)

➢ More than order of magnitude improvement!

Comparison to CPU/GPU





- *Worse performance with CPUs, partially recovered when increasing #cores*
- *GPUs become competitive when running simultaneously on batches (GPU more efficiently occupied)*
- *Optimal solution depends on use case (batches vs no batches)*

Comparison to CPU/GPU

Type	Hardware	⟨Inference time⟩	Max throughput	Setup
CPU	Xeon 2.6 GHz, 1 core	1.75 seconds	0.6 img/s	`CMSSW`, `TF v1.06`
CPU	i7 3.6 GHz, 1 core	500 ms	2 img/s	python, `TF v1.10`
CPU	i7 3.6 GHz, 8 core	200 ms	5 img/s	python, `TF v1.10`
GPU (batch=1)	NVidia GTX 1080	100 ms	10 img/s	python, `TF v1.10`
GPU (batch=32)	NVidia GTX 1080	9 ms	111 img/s	python, `TF v1.10`
GPU (batch=1)	NVidia GTX 1080	7 ms	143 img/s	TF internal, `TF v1.10`
GPU (batch=32)	NVidia GTX 1080	1.5 ms	667 img/s	TF internal, `TF v1.10`
Brainwave	Altera Artix	10 ms	660 img/s	`CMSSW`, *on-prem*
Brainwave	Altera Artix	60 ms	660 img/s	`CMSSW`, *remote*

- *Worse performance with CPUs, partially recovered when increasing #cores*
- *GPUs become competitive when running simultaneously on batches (GPU more efficiently occupied)*
- *Optimal solution depends on use case (batches vs no batches)*

European Research Council

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

https://github.com/hls-fpga-machine-learning/SonicCMS

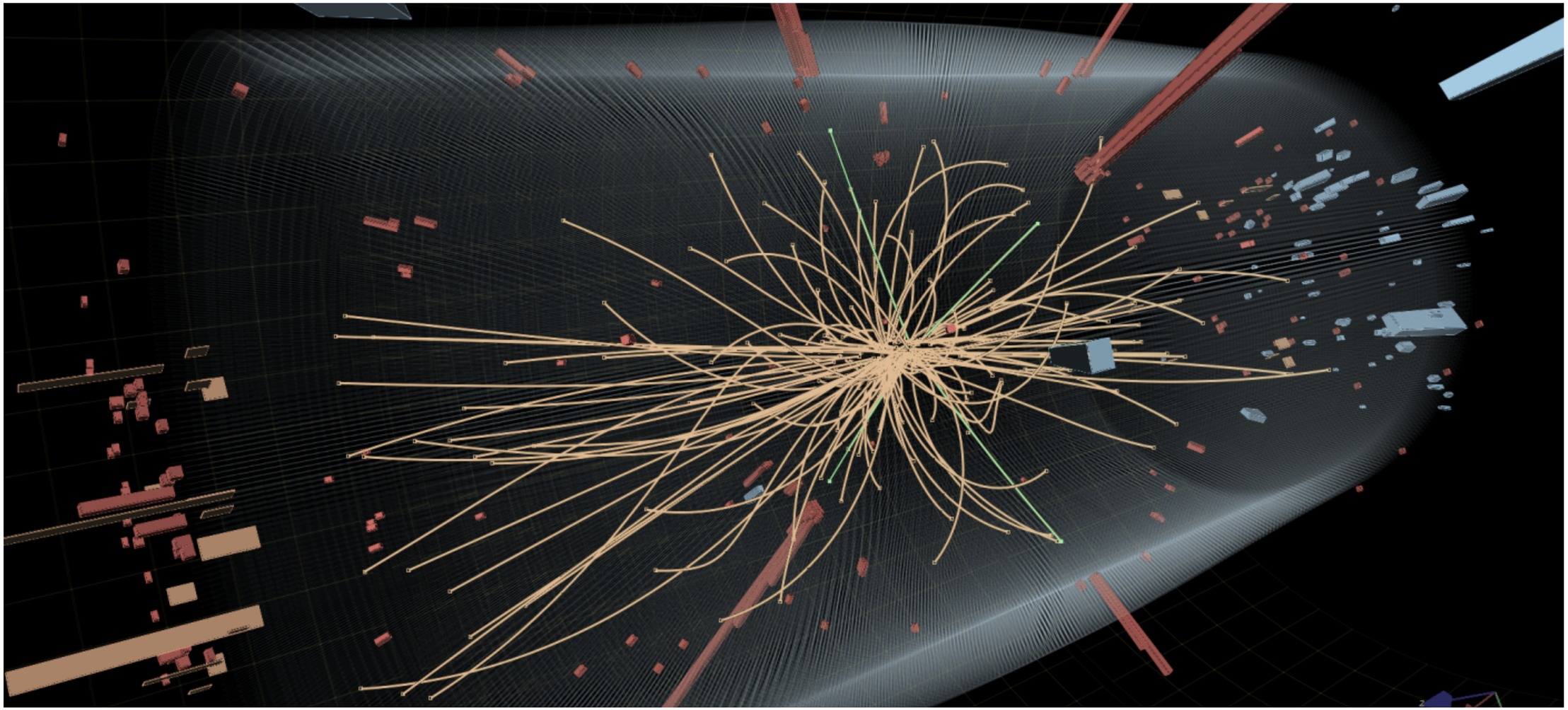
https://hls-fpga-machine-learning.github.io/hls4ml/

arXiv.org

https://arxiv.org/pdf/1804.06913.pdf

https://arxiv.org/pdf/1904.08986.pdf

European Research Council

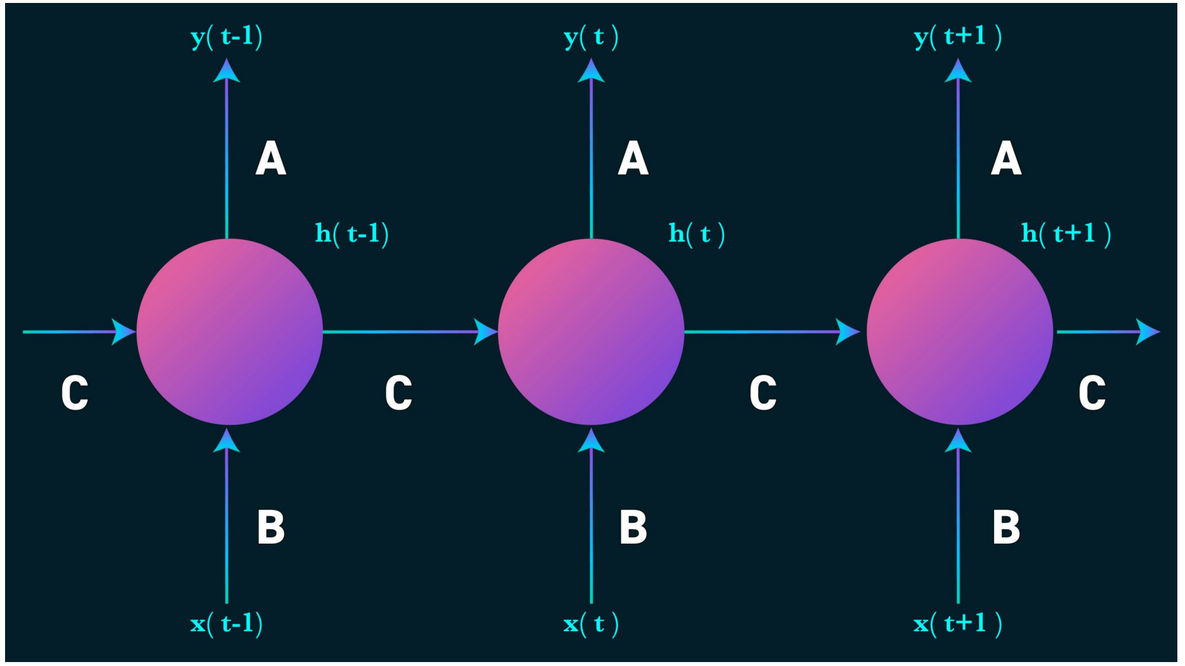


Language processing for particle physics

European Research Council

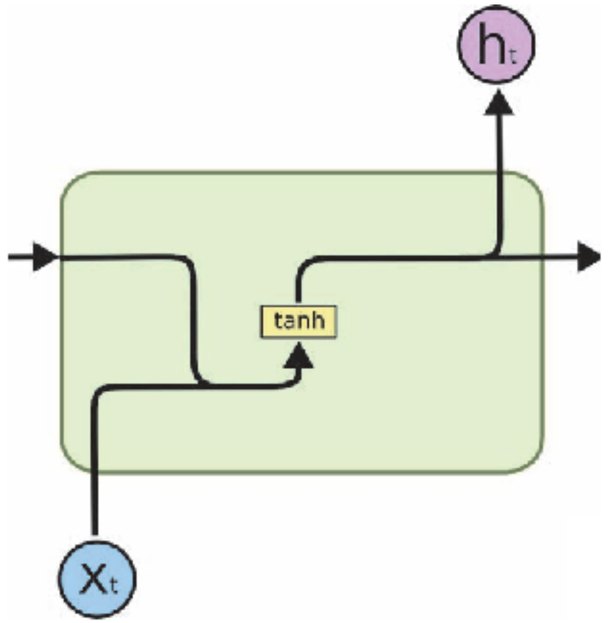
Recurrent betworks

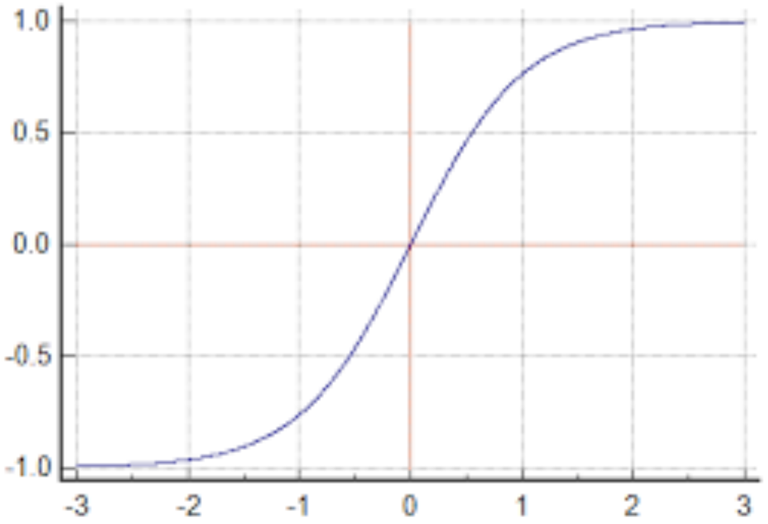
- *Recurrent architectures are designed to process sequences of data*
- *Then idea is to have information flowing in the network while the sequence is sequentially processed*
- *Through this idea, recurrent networks mimic memory persistence*
- *Advantages*
  - *the input is not fixed-sized*



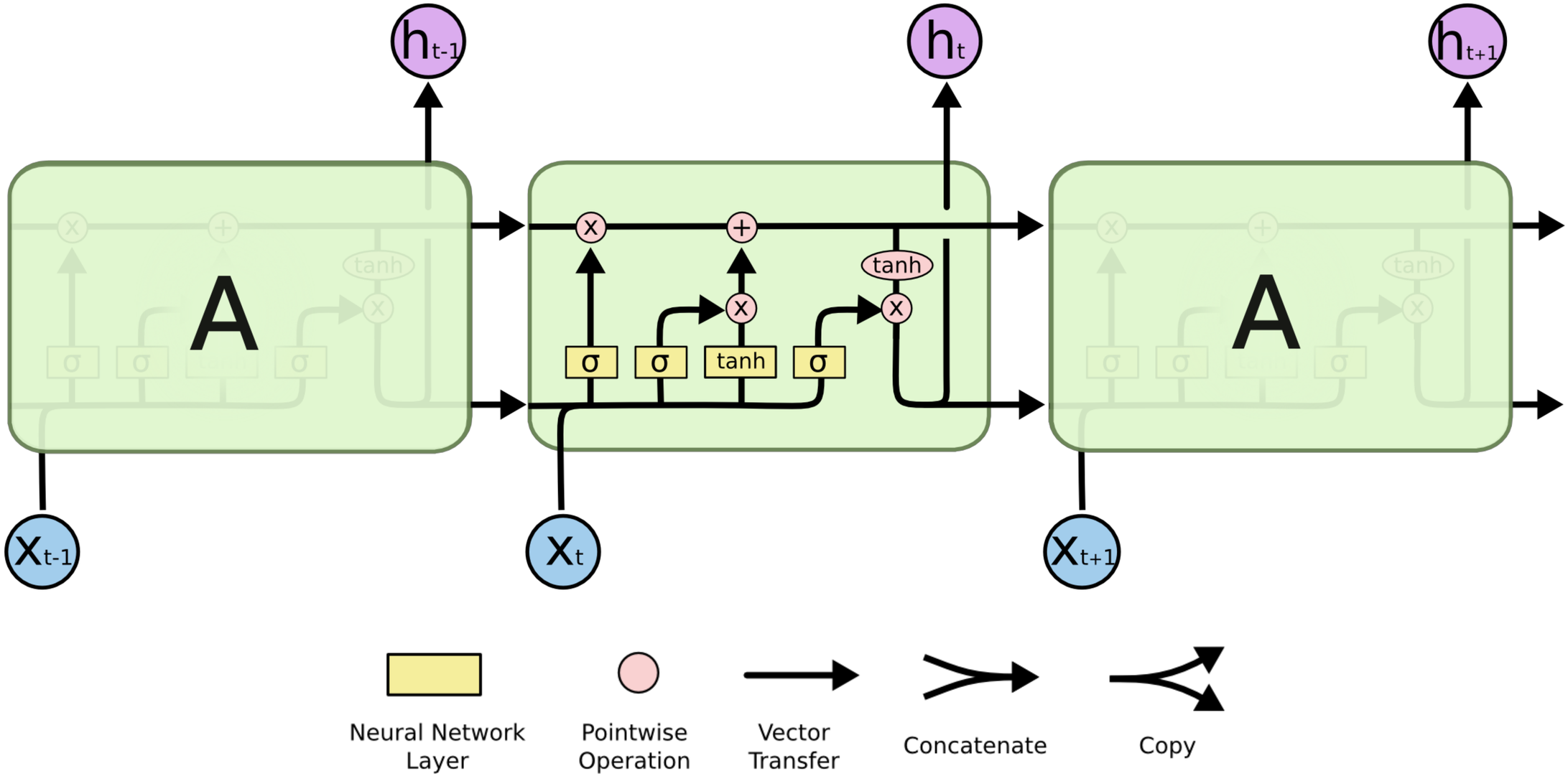
RNN

- *The simplest recurrent architecture*
  - *The processing unit receives an input x and the output of the previous-particle processing h (the context)*
  - *The product of the two is activated by a tank function*
  - *the result can be used as it is and/or passed to the next processing*

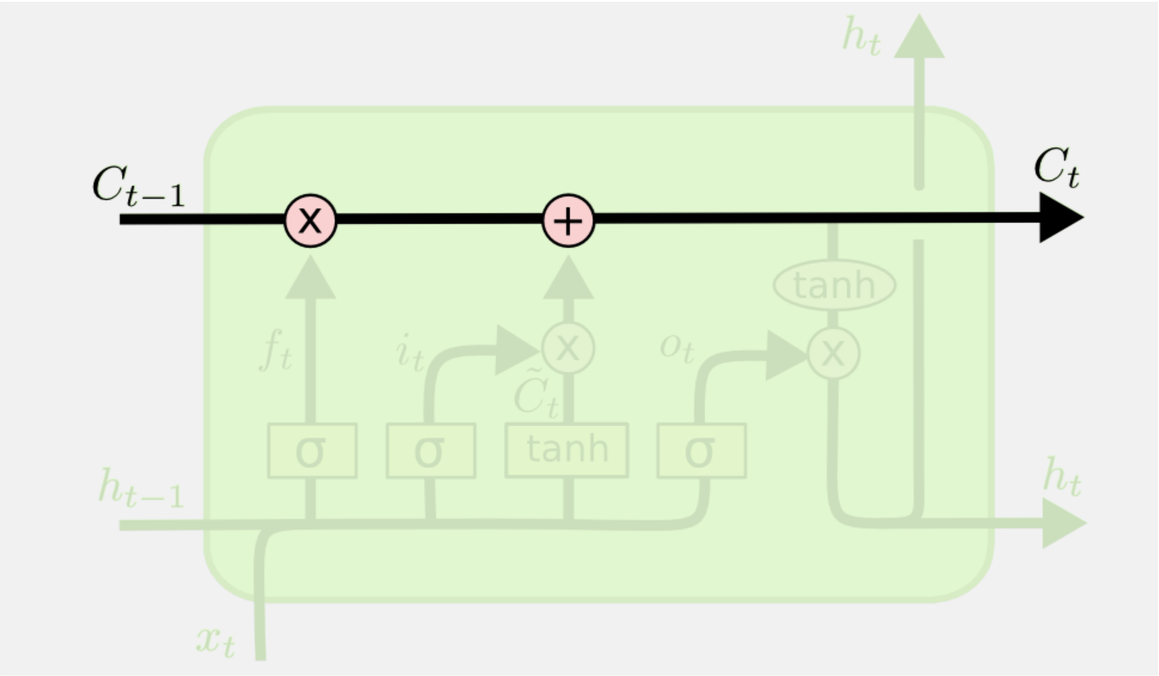


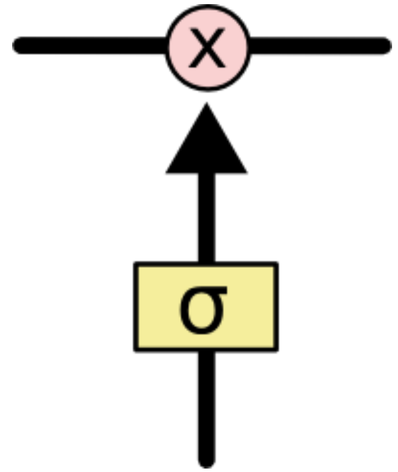


LSTM


h_{t-1}
h_t
h_{t+1}
A
A
tanh
σ
σ
tanh
σ
tanh
X_{t-1}
X_t
X_{t+1}
Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy

LSTM's memory cell

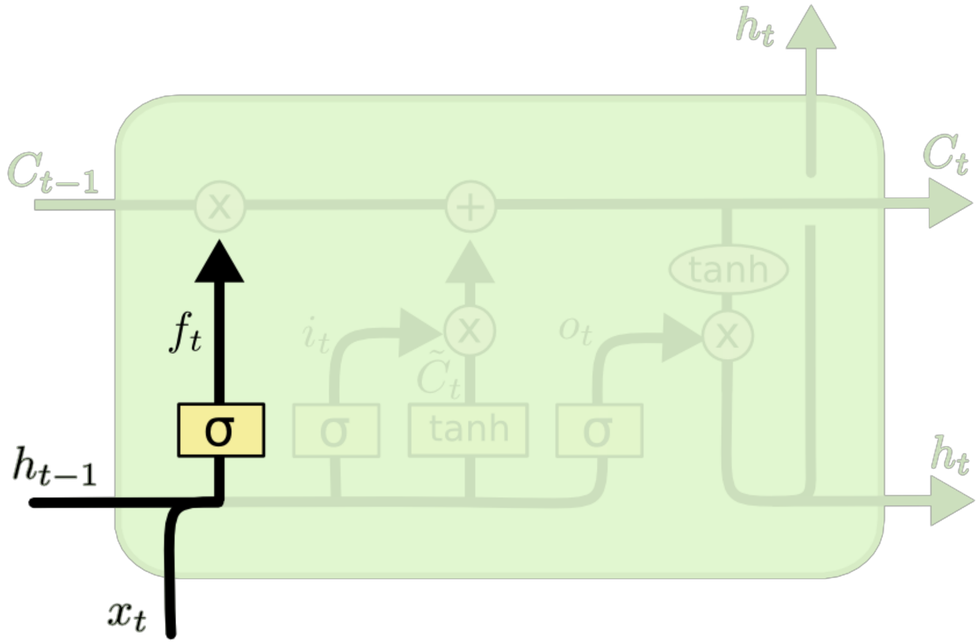




- *Information in the network flows through the memory cell*
  - *simple operations on it (to make the flow easy)*
  - *capability to reset the information, whit the next input acting as a gate*

LSTM processing

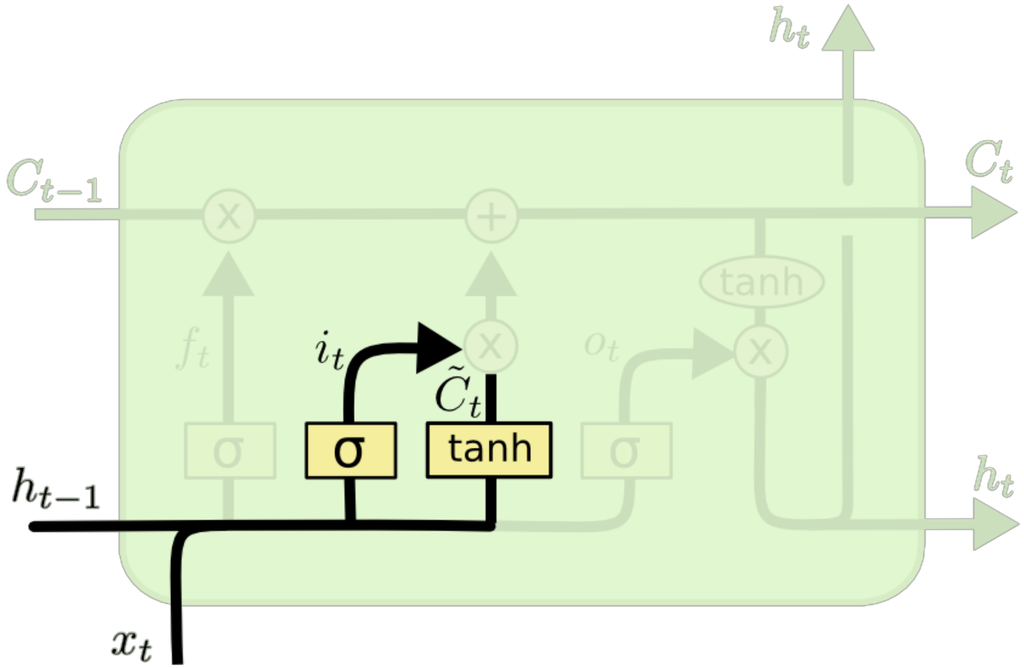
- ***Layer1:*** *decide if the context stored in the memory cell is to be forgotten or not*
  - *yes/no question -> sigmoid function*
  - *decision taken based on context and input*



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

LSTM processing

- ***Layer2:*** *two networks act to update the memory cell*
  - *a sigmoid (as in layer 1) decides which values to pass through (0/1)*
  - *a tanh function assigns a weight in [-1,1] to it*

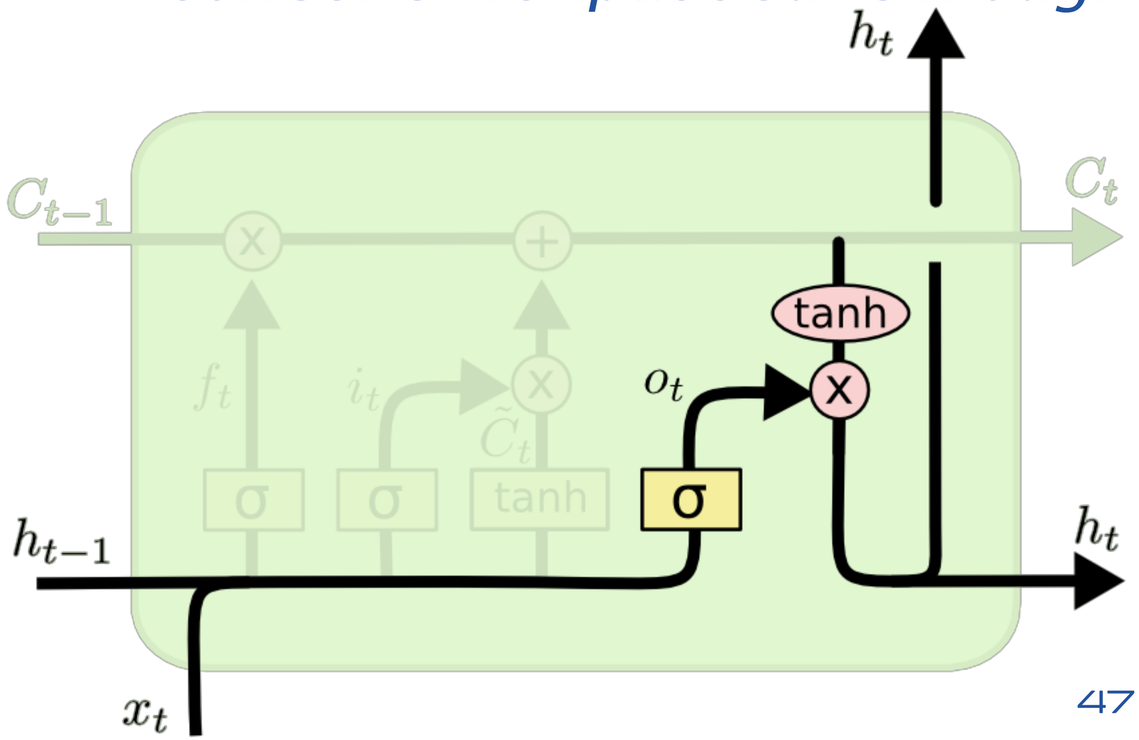


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

LSTM processing

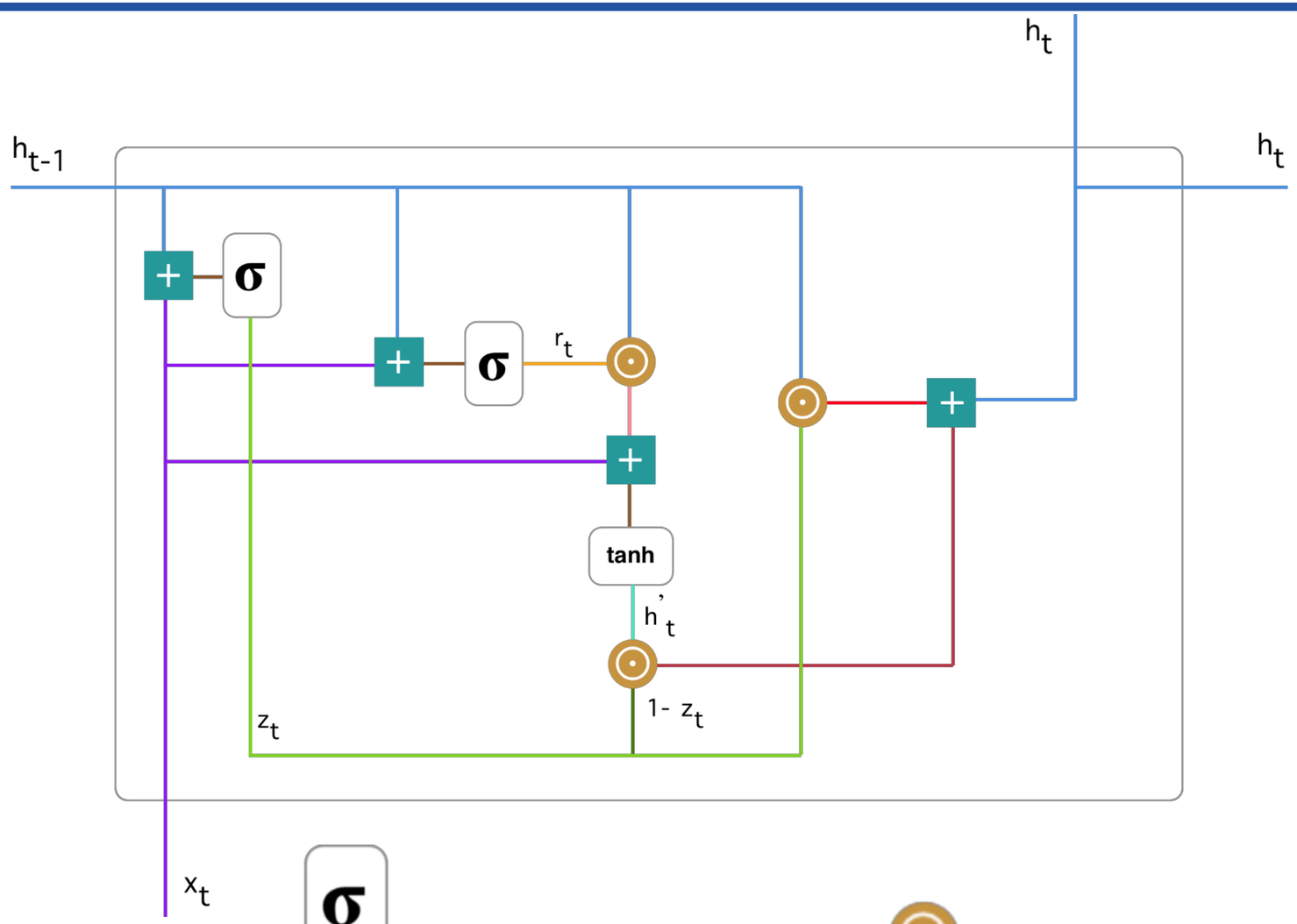
- ***Layer3:*** *fixes the output value to be passed next*
  - *the cell content is pushed in [-1,1] by a tan*
  - *a gate function (sigmoid) sets which fraction of the cell content is passed through*



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Gated Recurrent Unit



"plus" operation

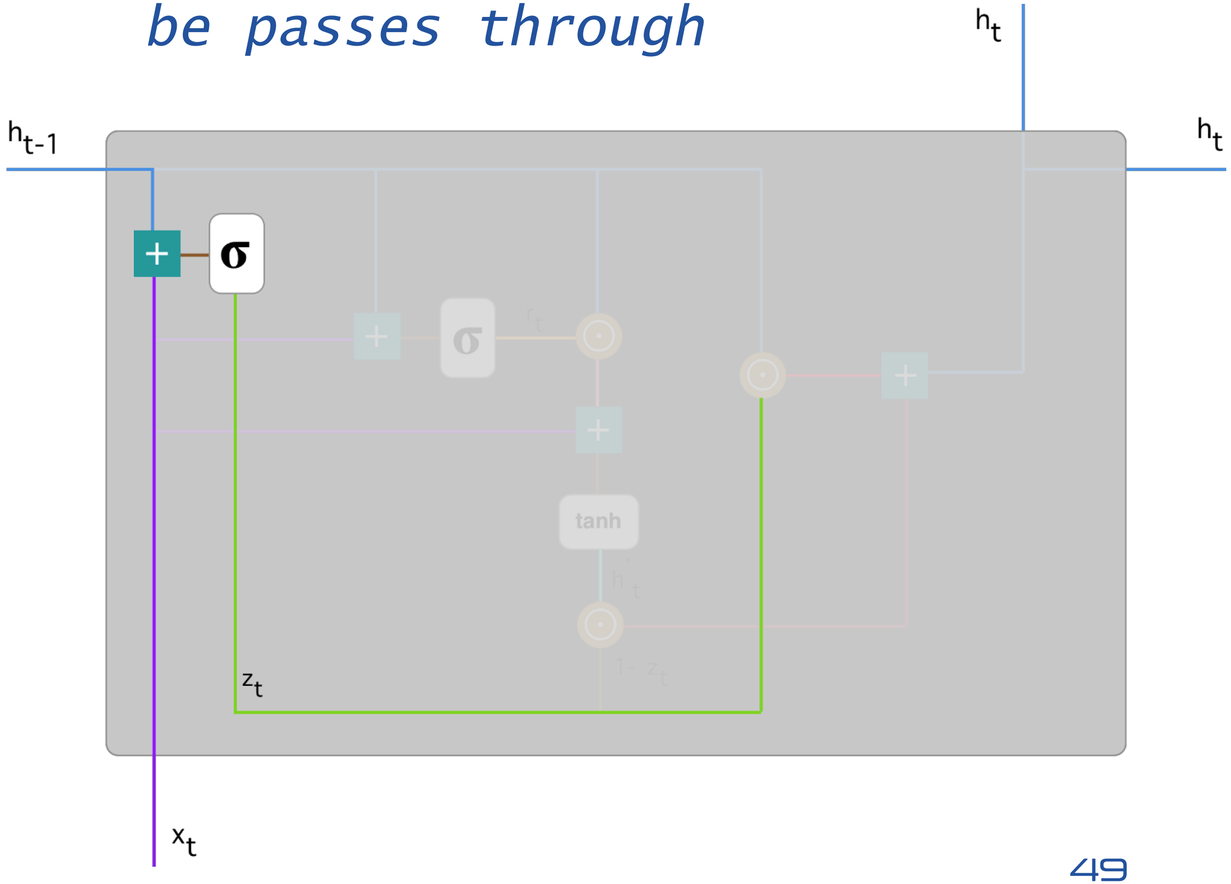
"sigmoid" function

"Hadamard product" operation

"tanh" function

GRU processing

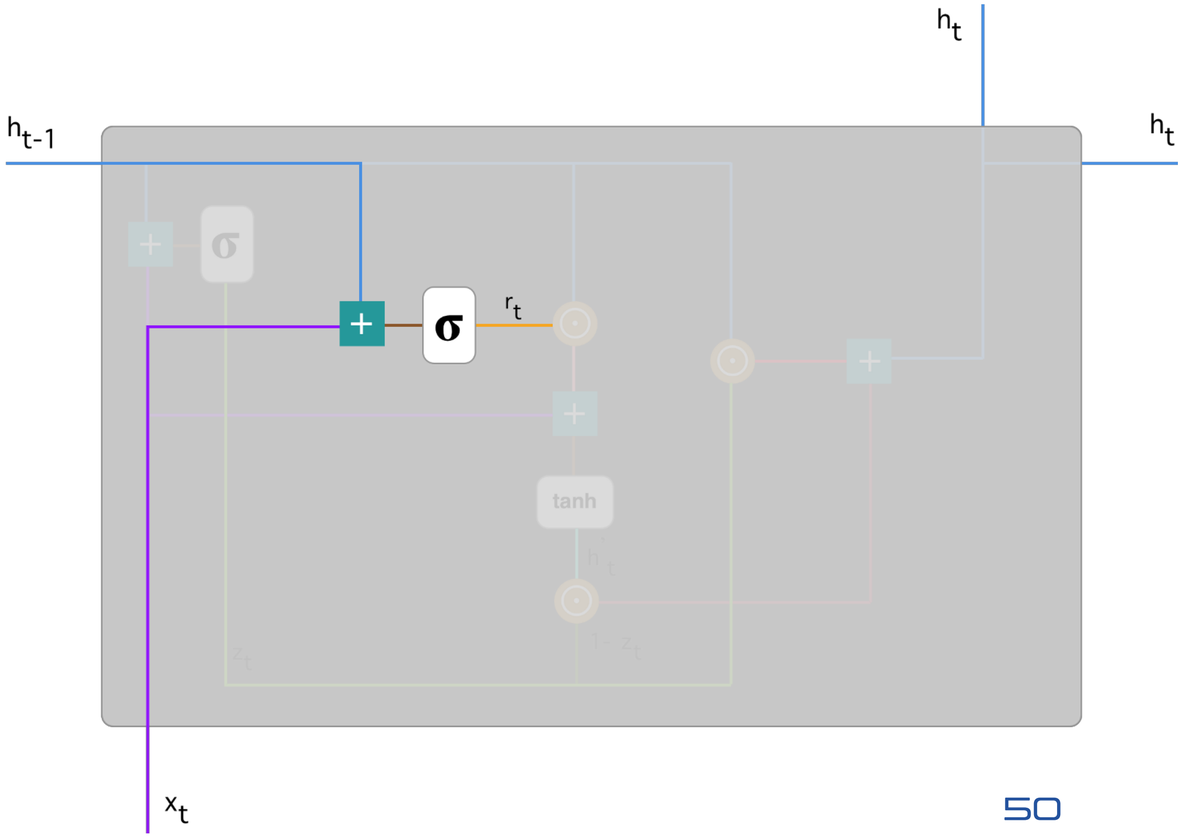
- ***Layer1:*** *update gate*
  - *multiply input and context by update weights*
  - *based on that, decide how much of the previous information should be passes through*



$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

GRU processing

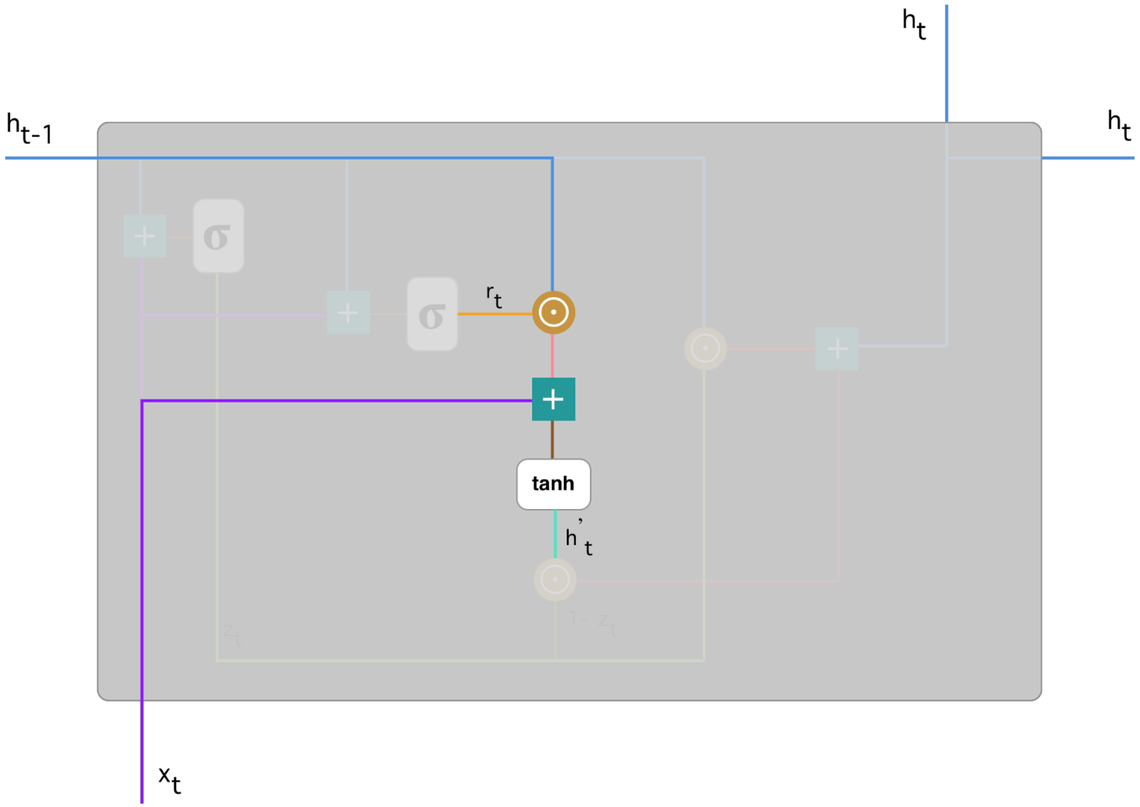
- ***Layer2:*** *reset gate*
  - *multiply input and context by reset weights*
  - *decide how much of the previous information is to be forgotten*



$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

GRU processing

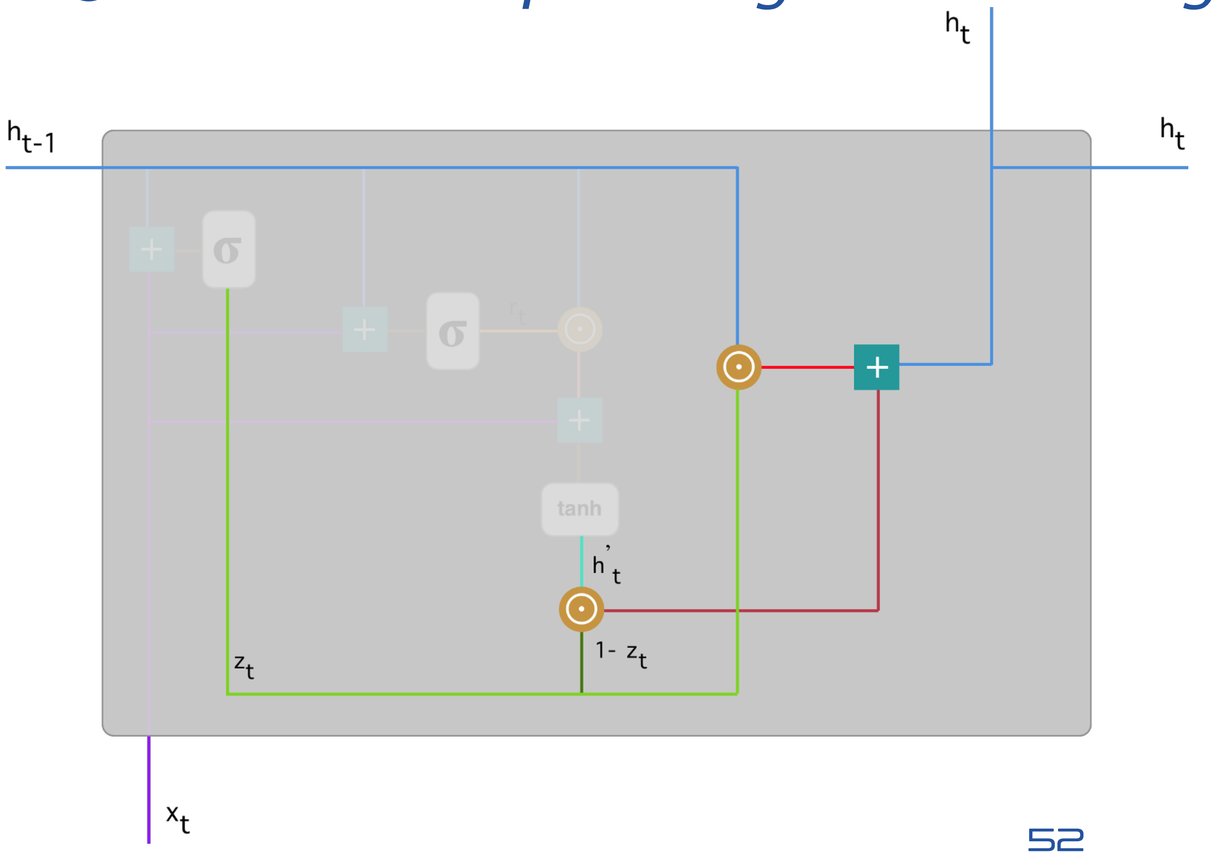
- ***Layer3:*** *determine the current memory content*
  - *the context information is partially removed to an element-by-element product with the reset gate*
  - *the result is summed to the (weighted) input and passed through a tank activation function*



$$h_t' = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

GRU processing

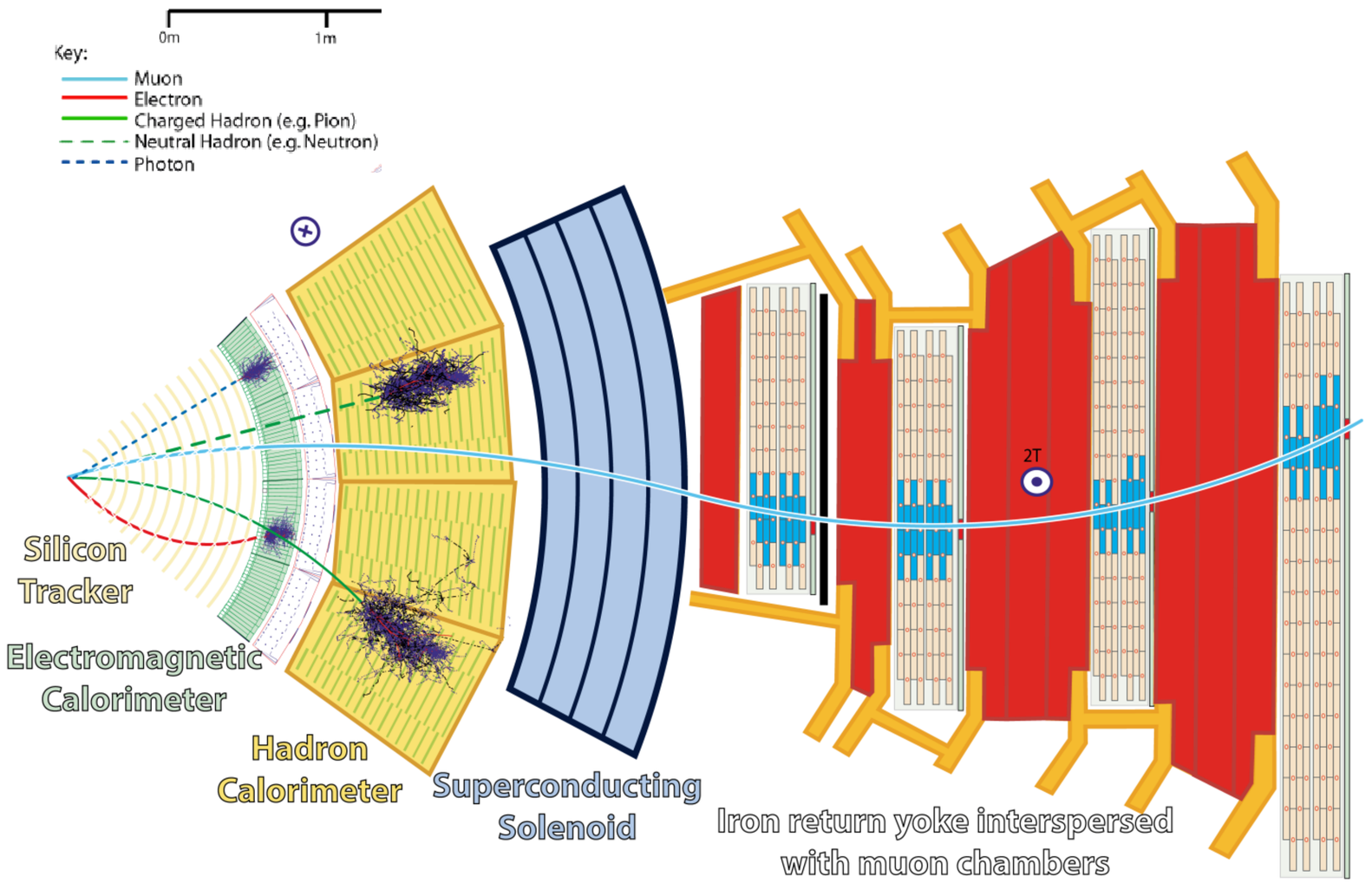
- ***Layer4:*** *compute the new memory state, to pass through*
  - *mixes the current memory state with the input one*
  - *uses the update gate to weight the two contributions*



$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t$$

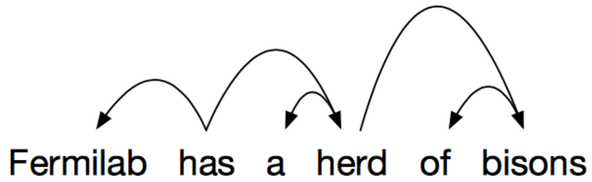
RNNs for particle physics

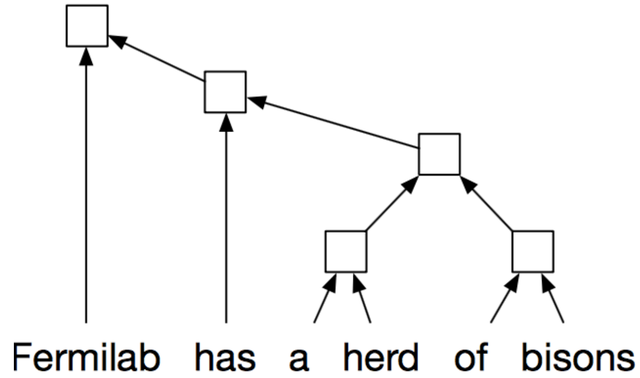
- *RNN can be used to deal with list of reconstructed particles*
- *Their architecture better fits our needs*
  - *No underlying assumption on the detector geometry*
- *This comes at two costs*
  - *Some ordering principle needs to be specified*
  - *Inference is sequential (not ideal for L1 real time)*

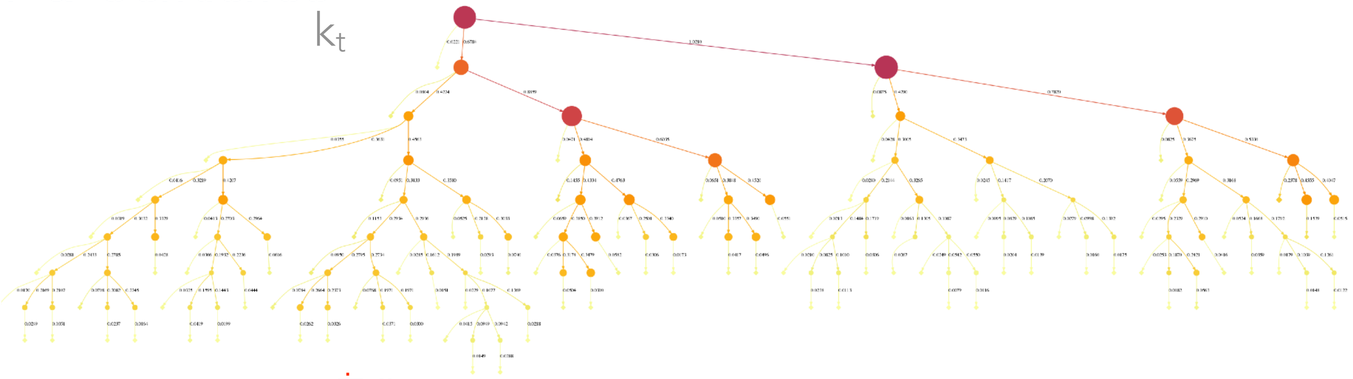


LHC events & language processing

- *PF reco is not the best match for computing vision techniques (e.g., convolutional neural networks) don't work*
  - *one would have to convert the particles to a pixelated images, loosing resolution*
- *Instead, list of particles can be processed by Deep Learning architectures designed for natural language processing (RNN, LSTMs, GRUs, …)*
  - *particles as words in a sentence*
  - *QCD is the grammar*

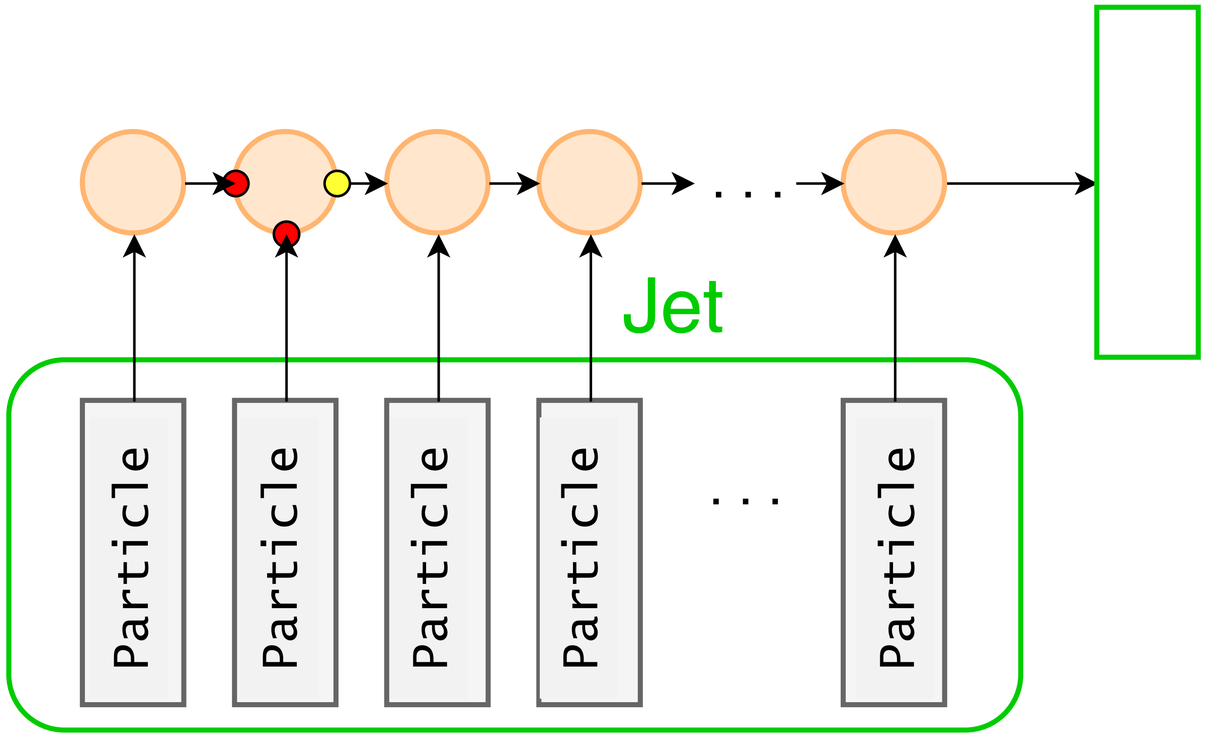






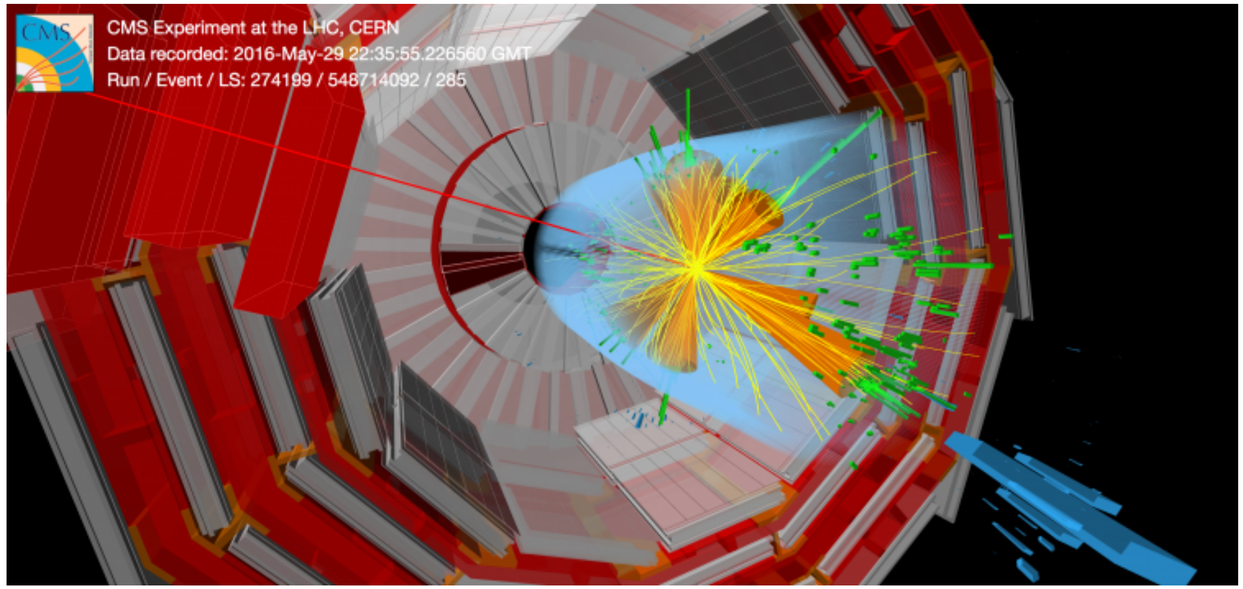
Recurrent Neural Networks

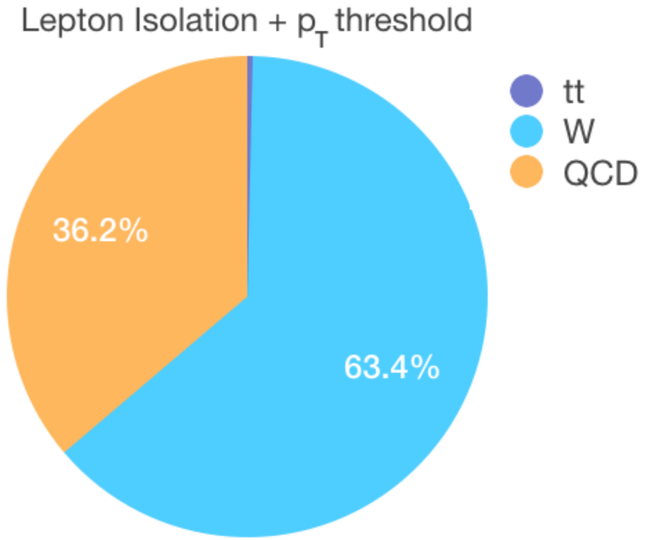
- A network architecture suitable to process an ordered sequence of inputs
  - words in text processing
  - a time series
  - particles in a list
- Could be used for a single jet or the full event
- Next step: graph networks (active research direction)



Example: topology classification @LHC

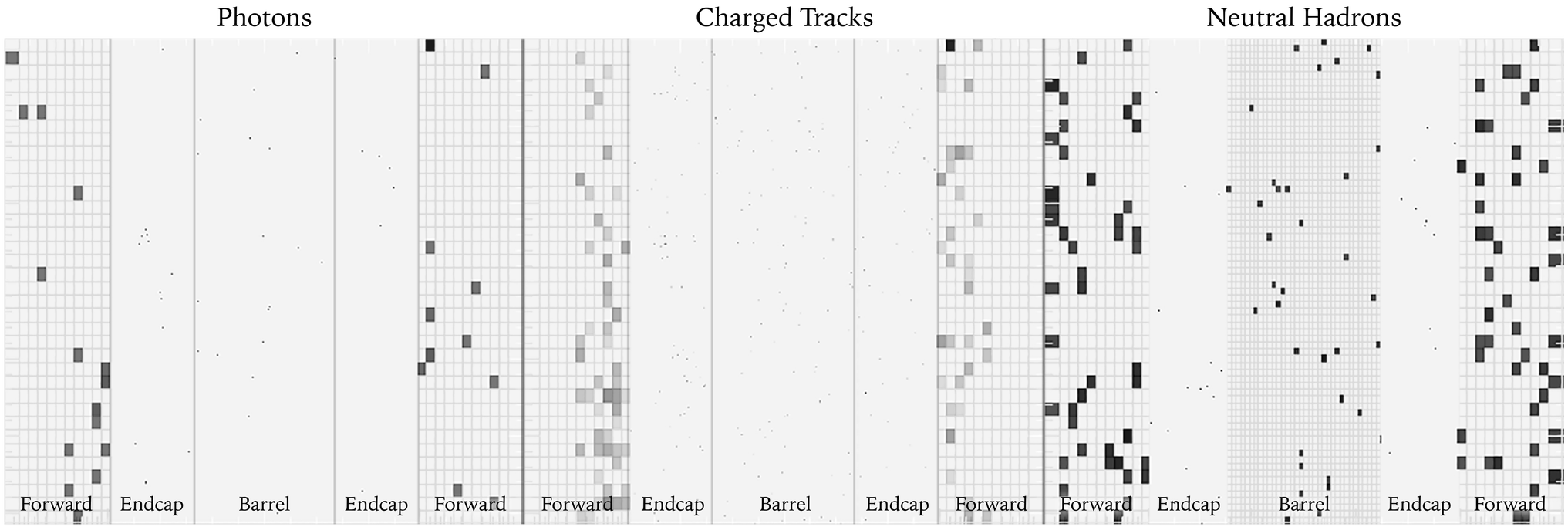
- *Finding which process generated an event is a typical task @LHC experiments*
- *Let's take as input the events with one energetic electron or muon (lepton)*
  - *After some loose selection, mainly three processes left: jet production (QCD, the background) + W or tt productions (the signals)*
  - *We want to separate the three components*



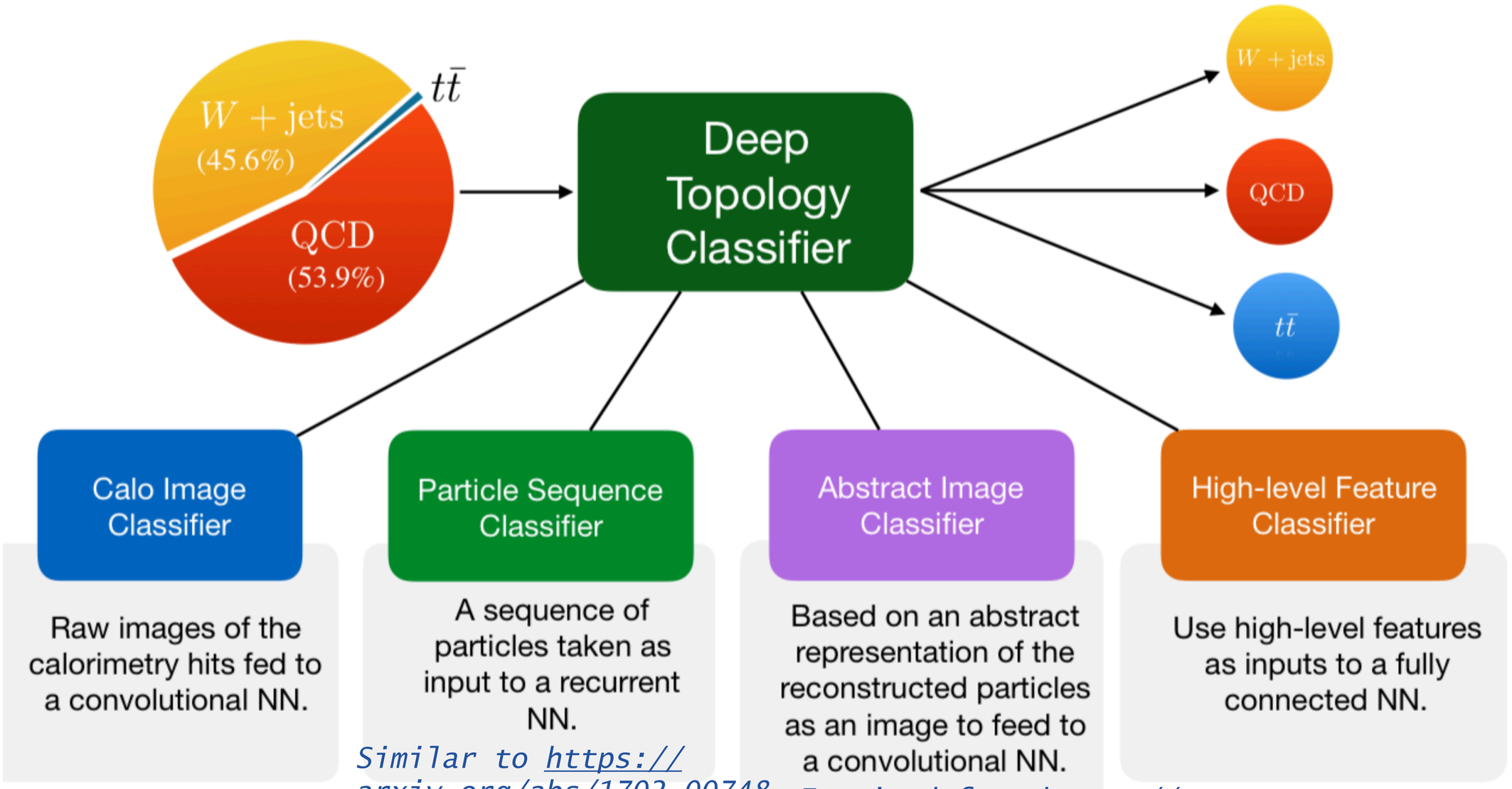


What the event looks like

- *sparse image with many pixels*
  - *not the kind of image that CNNs usually deal with*
  - *still, reasonable performances (AUC~90%) can be obtained*



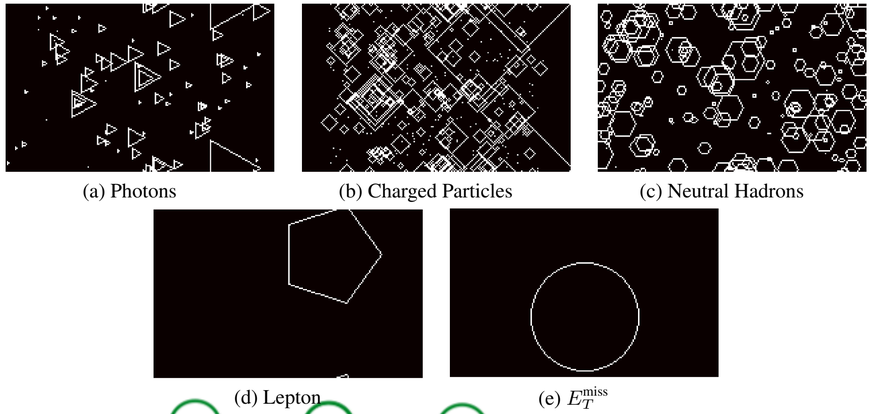
Event Representations

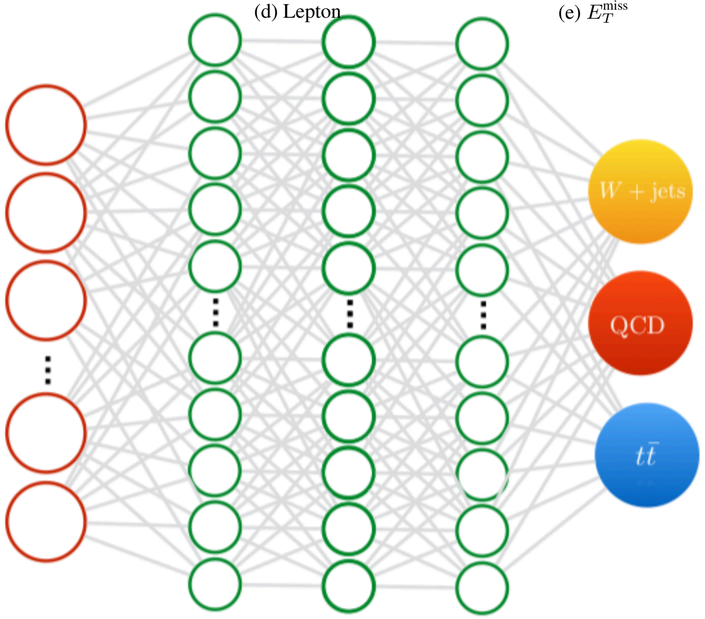


Similar to https://arxiv.org/abs/1702.00748

Inspired from https://arxiv.org/abs/1708.07034

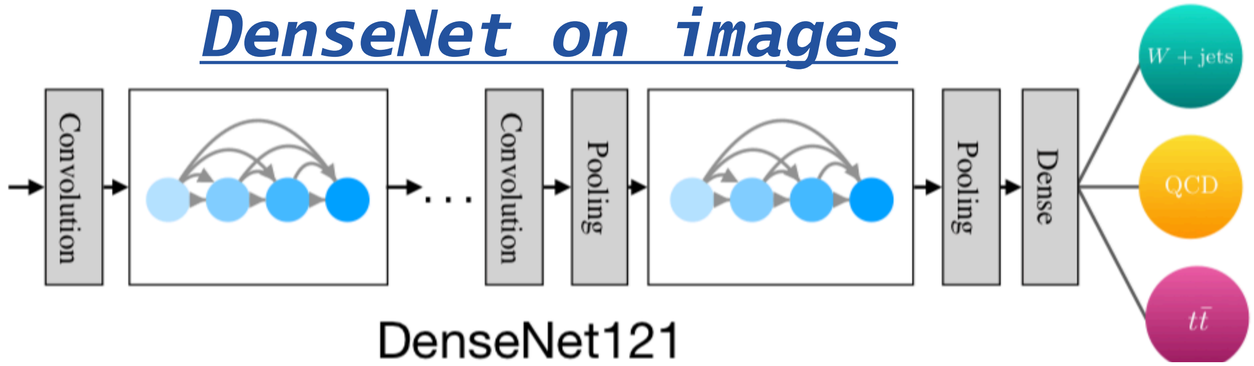
Event Representations



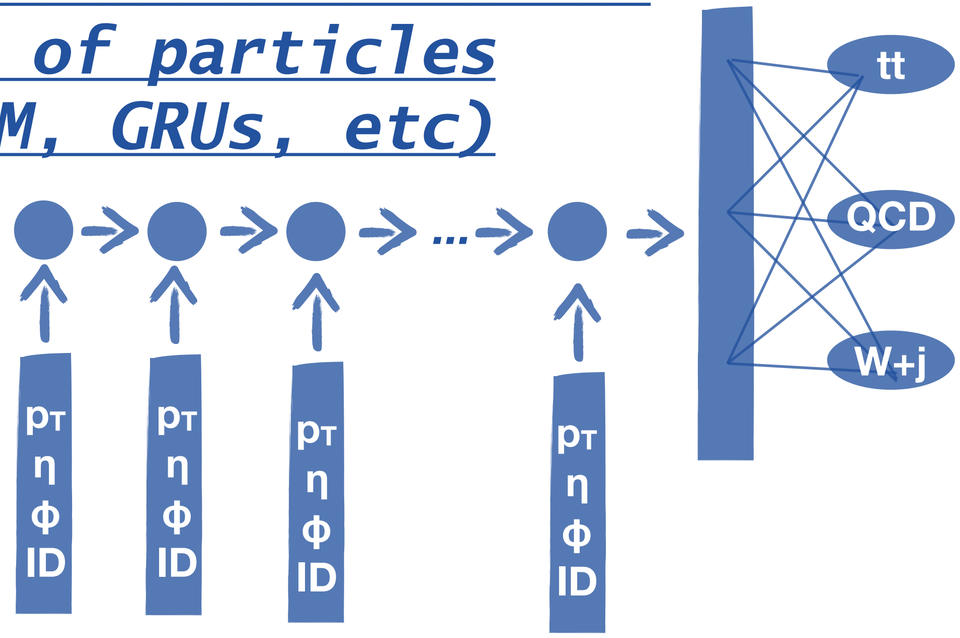


Fully-Connected classifier on physics-motivated features

DenseNet on images

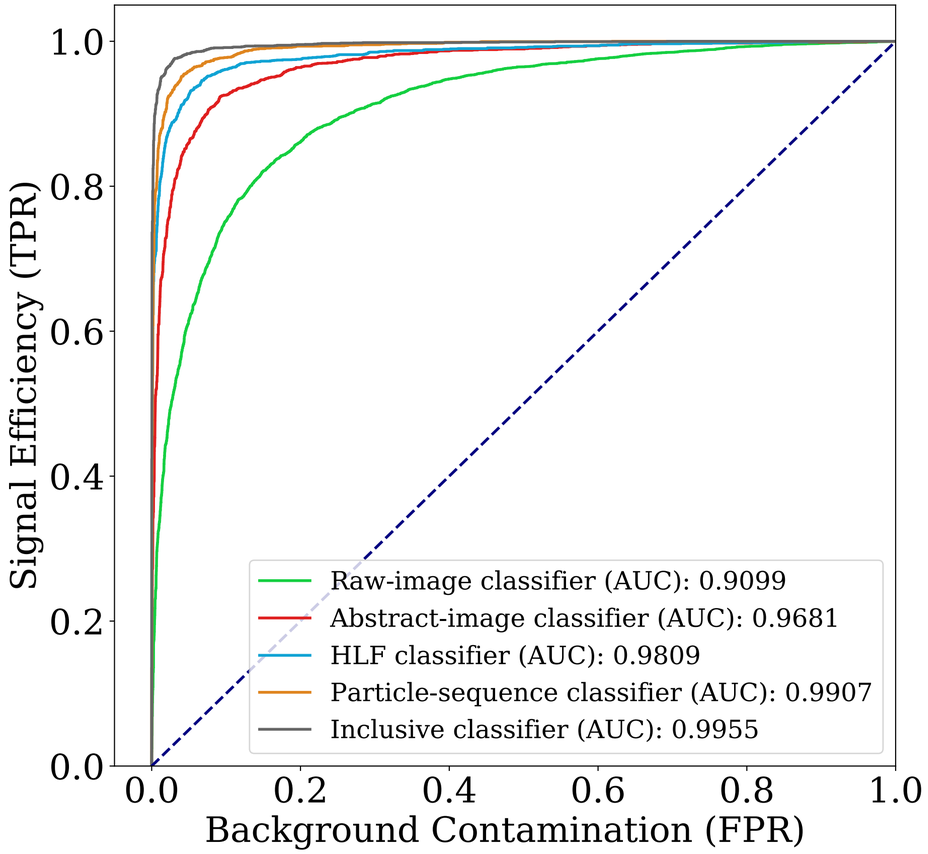


Recurrent nets on the list of particles (LSTM, GRUs, etc)

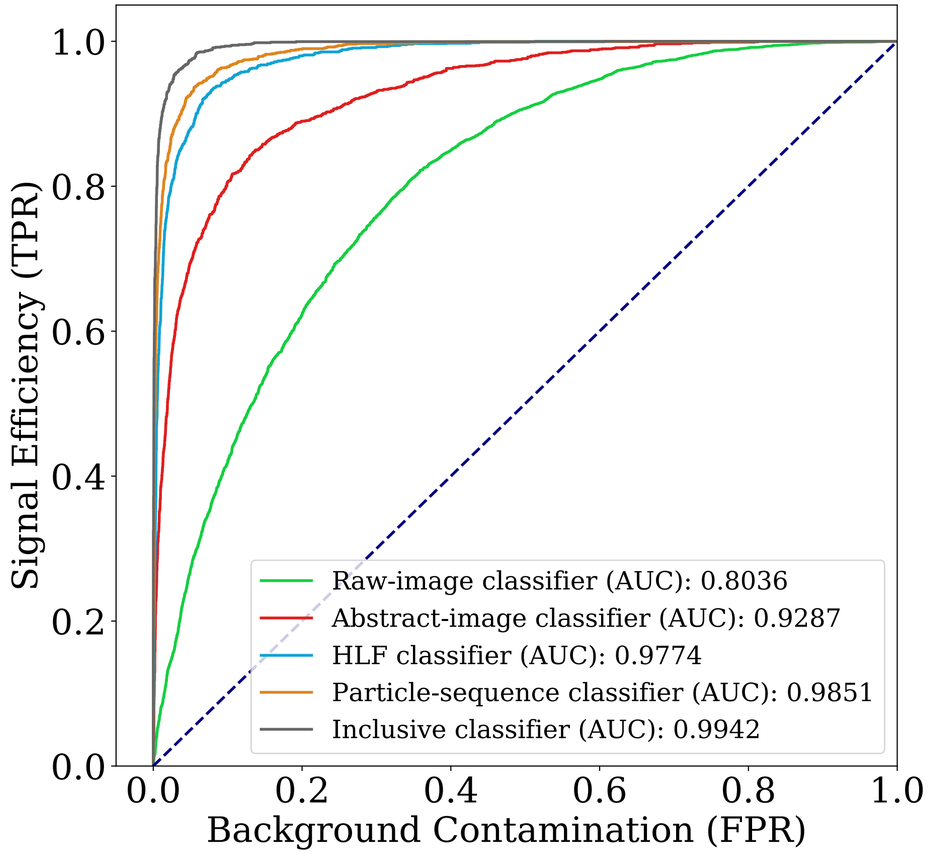


Performance

- *The GRU provides the best discrimination power*
- *The HLFs come second*
- *The combination of the two further improve*

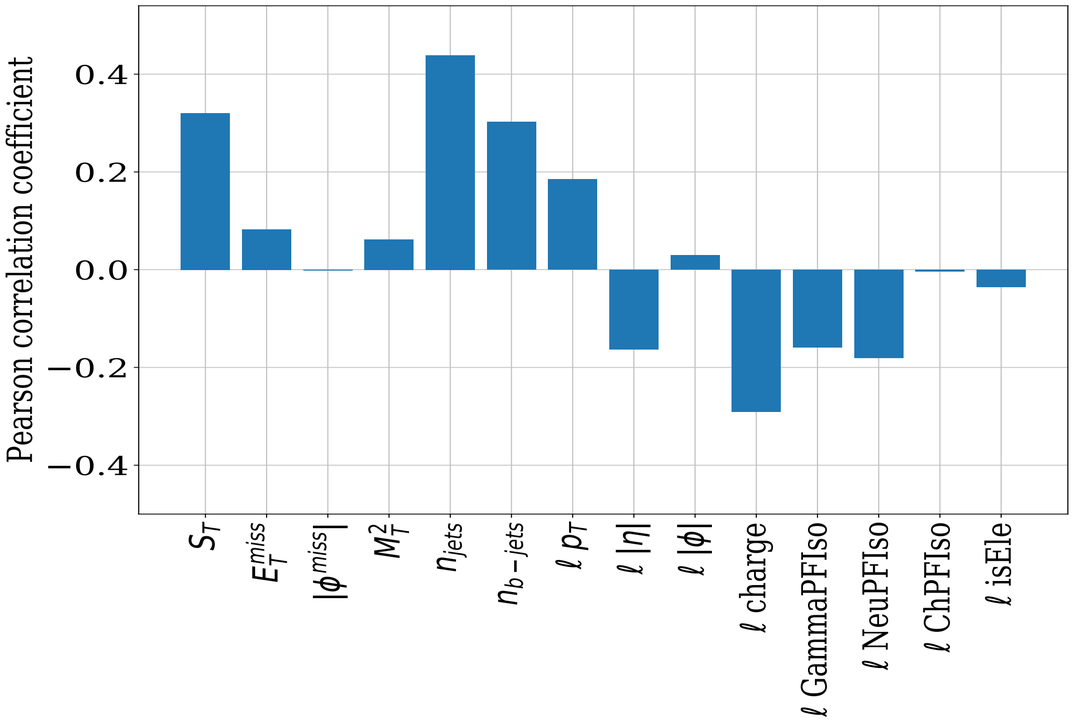


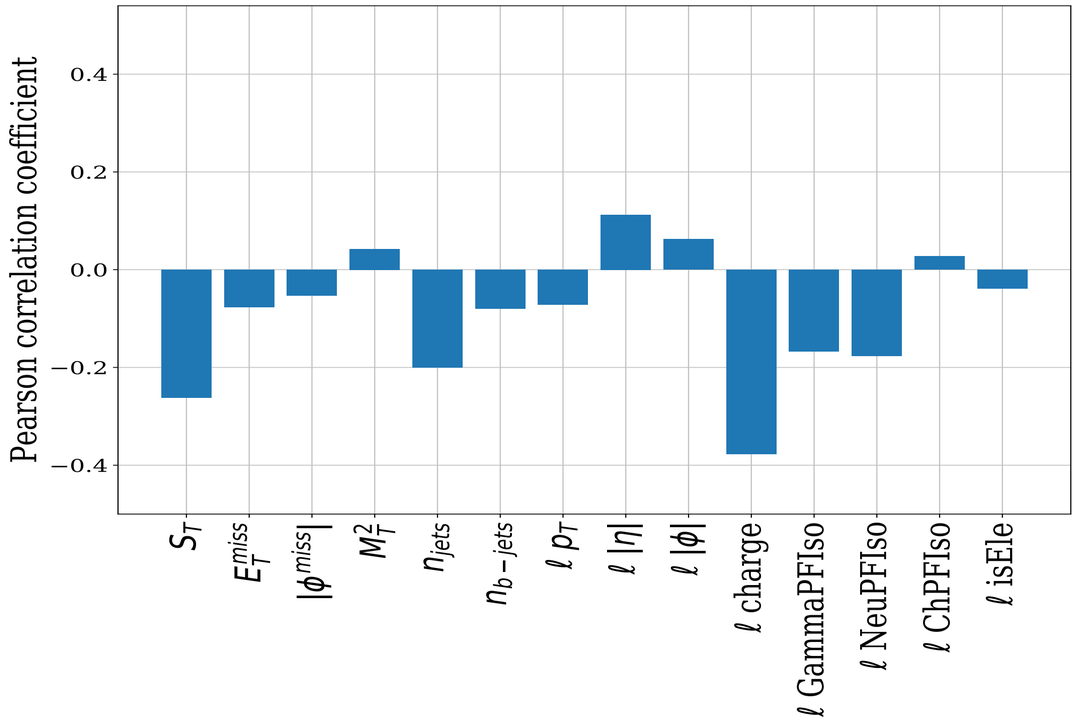
(a) $t\bar{t}$ selector



(b) W selector

Selection performances





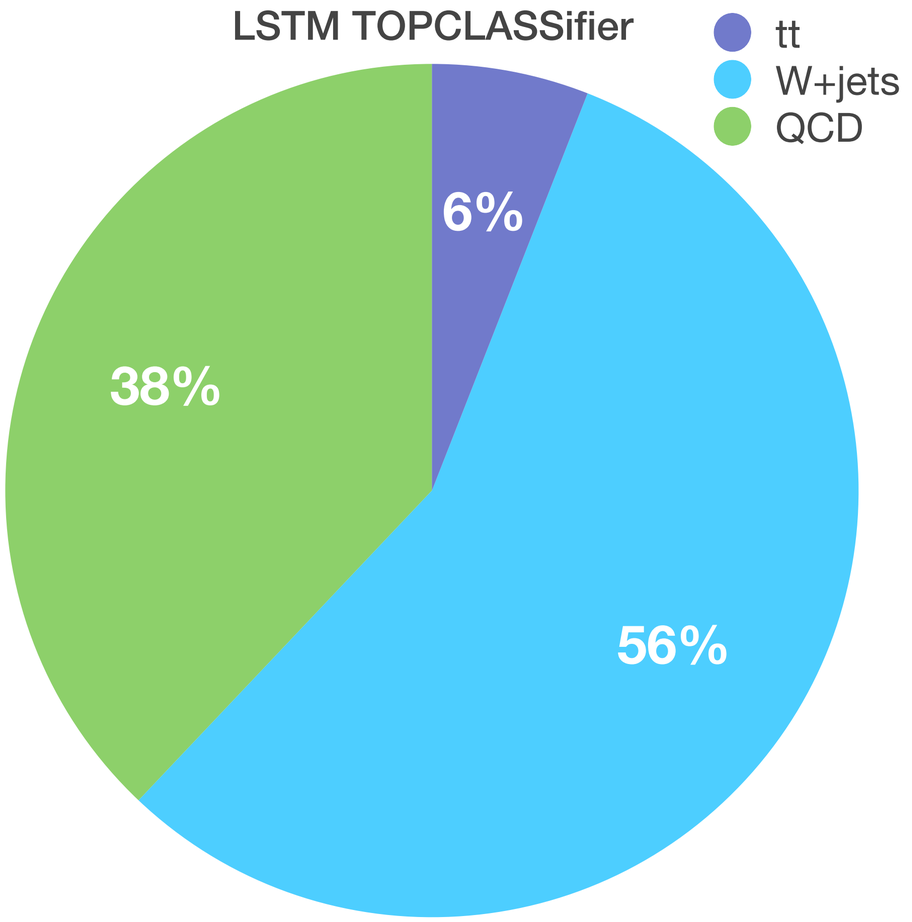
What is the network learning?

- tt events are more crowded that W events
- leptons in W and tt events are isolated from other particles

Cleaning up selected sample

A typical example: leptonic triggers

- *at the LHC, producing an isolated electron or muon is very rare. Typical smoking gun that something interesting happened (Z,W,top,H production)*
- *Triggers like those are very central to ATLAS/CMS physics*
- *The sample selected is enriched in interesting events, but still contaminated by non-interesting ones*
- *Contamination can be reduced with a DL classifier that rejects obvious false positives looking at the full event, not just at the lepton*



Summary

- *We discussed the importance of fast inference for future LHC upgrade*
- *We saw how NNs could be ported on FPGAs (translation to electronic circuit)*
- *We saw how cloud solution could be explored for on-line big-data processing*
- *We discuss recurrent NNs and their potential usage to save processing resources @LHC*

European Research Council