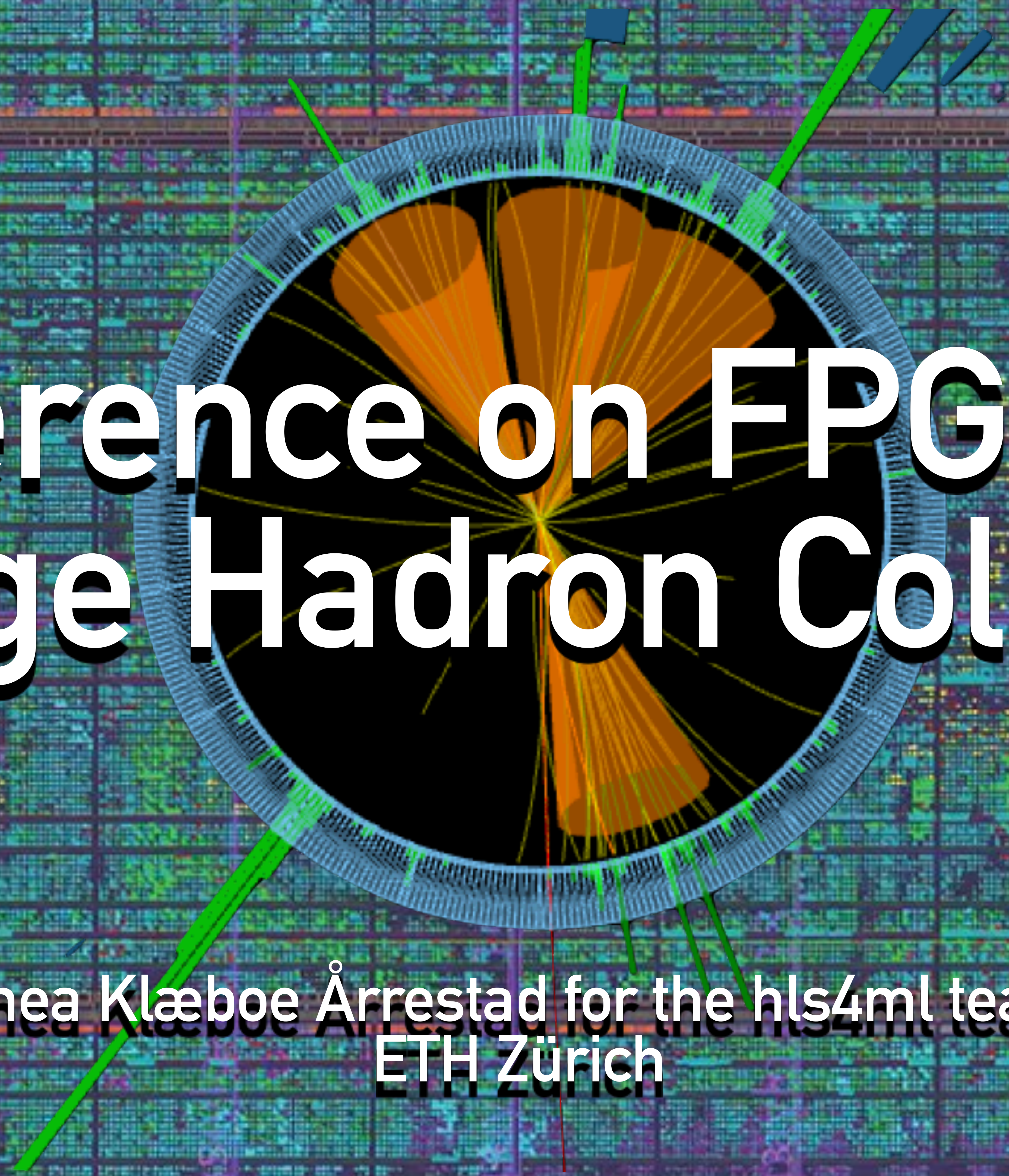
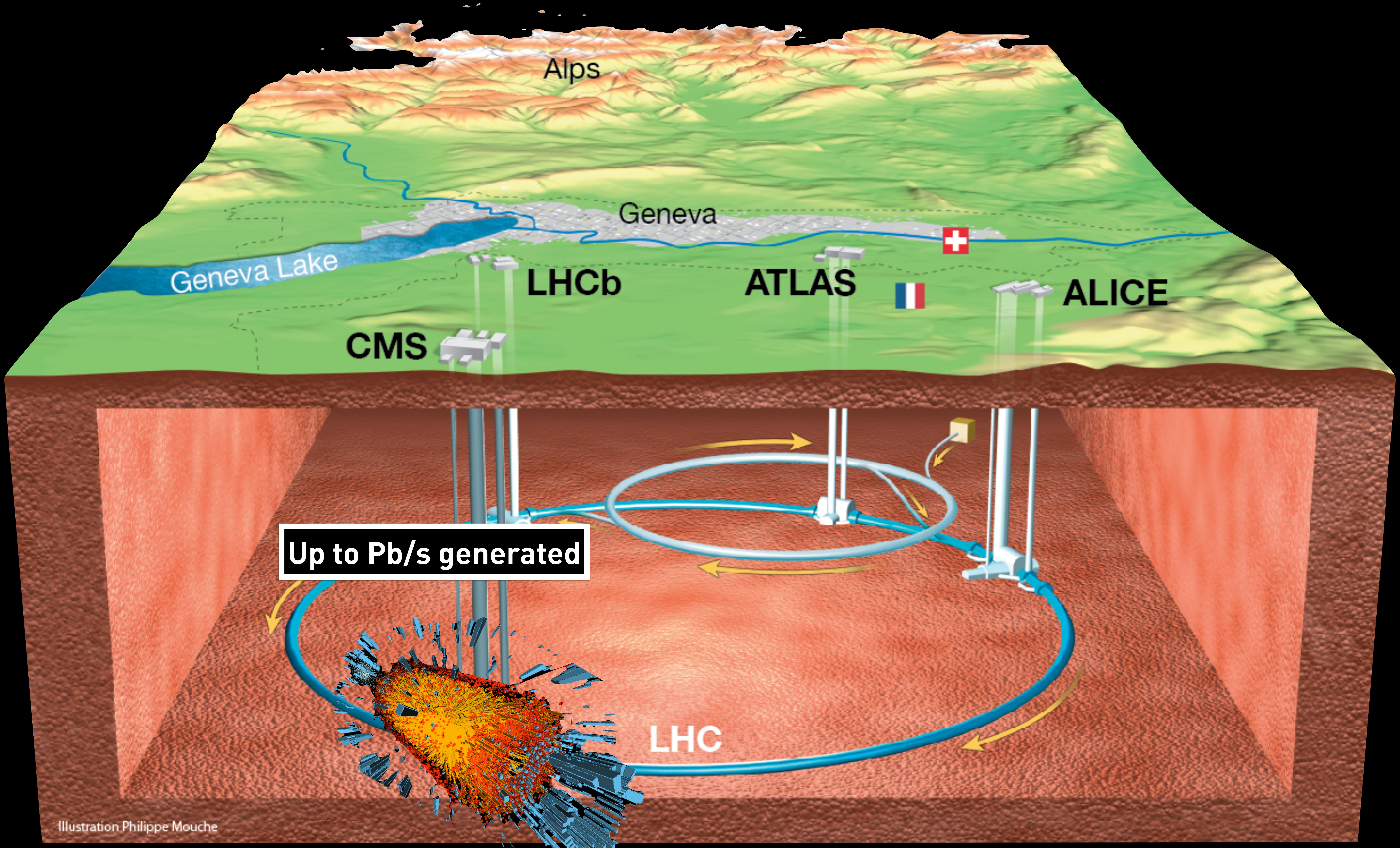
Fast inference on FPGAs at the Large Hadron Collider

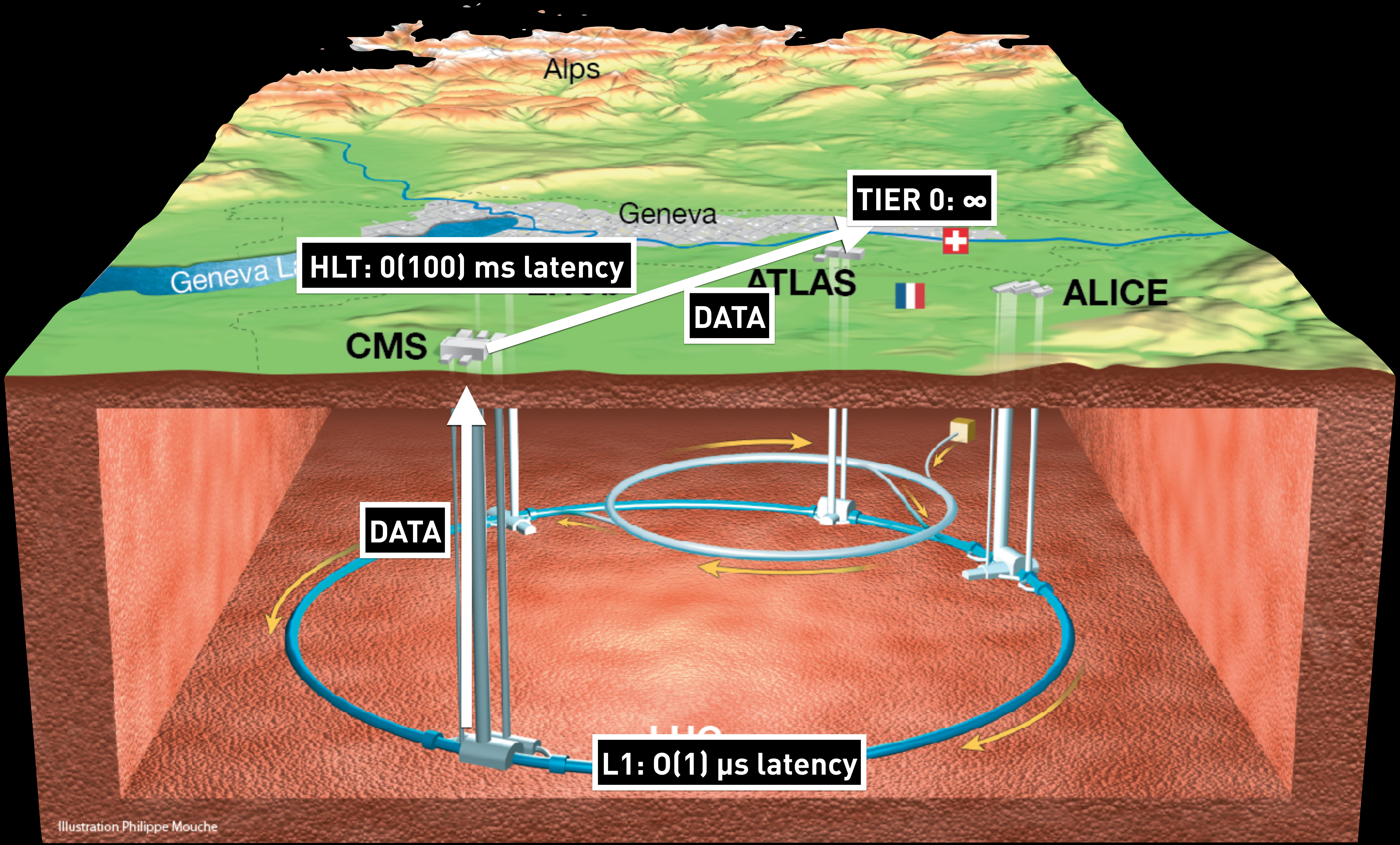


Thea Klæboe Årrestad for the hls4ml team
ETH Zürich

Bologna
November 2nd 2022


Alps
Geneva
Geneva Lake
LHCb
ATLAS
ALICE
CMS
Up to Pb/s generated
LHC
Illustration Philippe Mouche


Alps
Geneva
Geneva Lake
LHCb
ATLAS
CMS
LHC
63 Tb/s to L1
L1 trigger: O(1) μs latency
Data buffered on detector for O(1) μs
Illustration Philippe Mouche



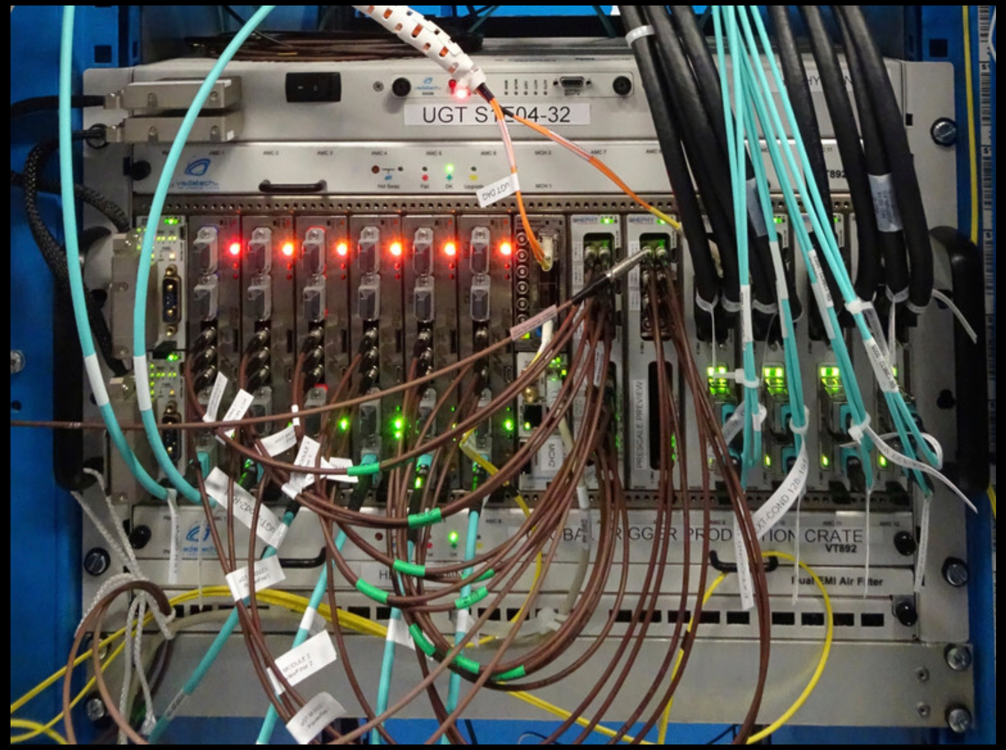
Alps
Geneva
TIER 0: ∞
HLT: O(100) ms latency
ATLAS
ALICE
DATA
CMS
DATA
L1: O(1) μs latency
Illustration Philippe Mouche

Alps
Geneva
Geneva Lake
LHCb
ATLAS
CMS
UXC55
USC55
LHC
Illustration Philippe Mouche

Level-1 hardware trigger

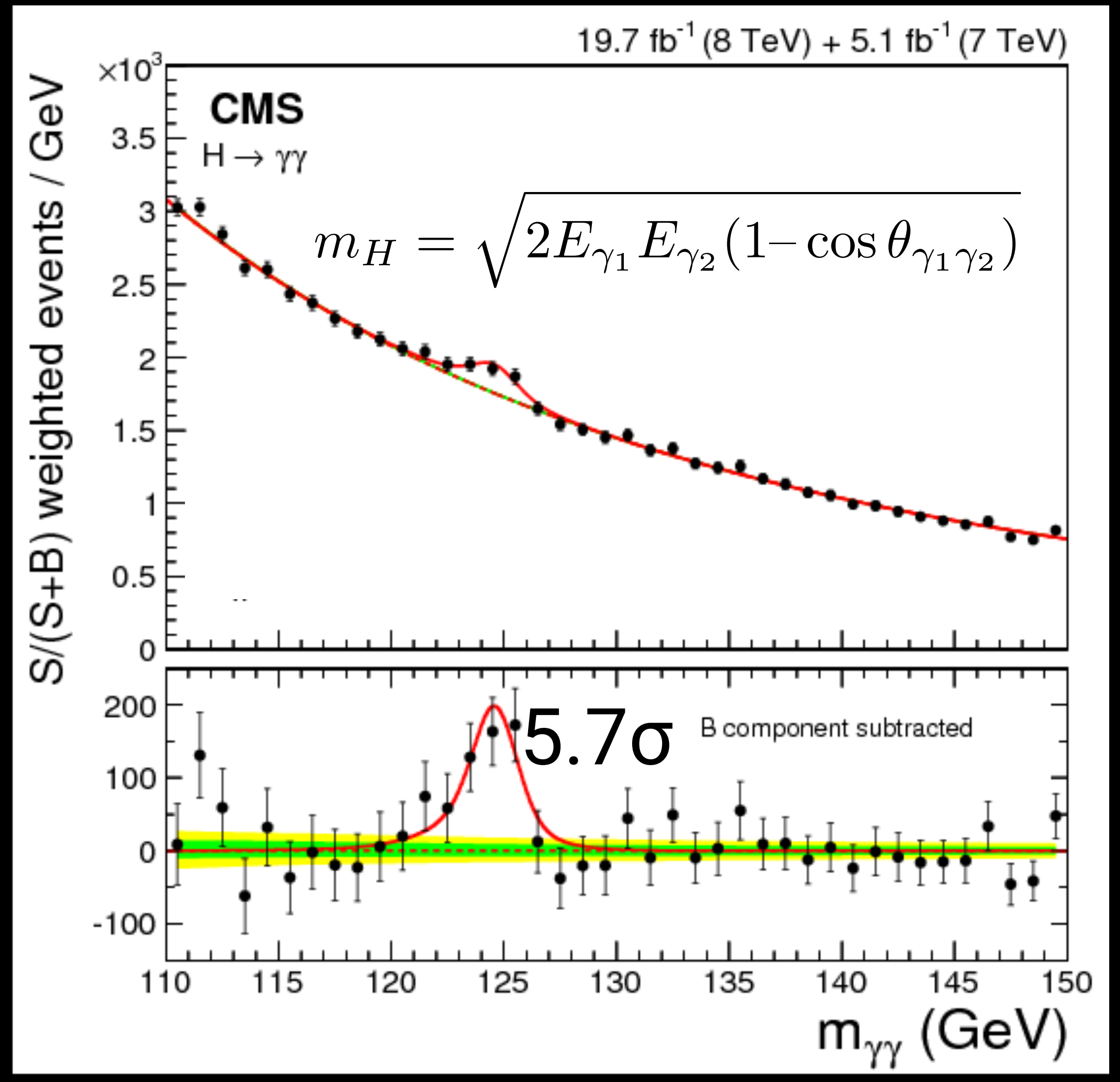
- **12.5 μs** to make decision
- Input data bandwidth **63 Tb/s**
- **1000 FPGAs** running thousands of algos

40 MHz



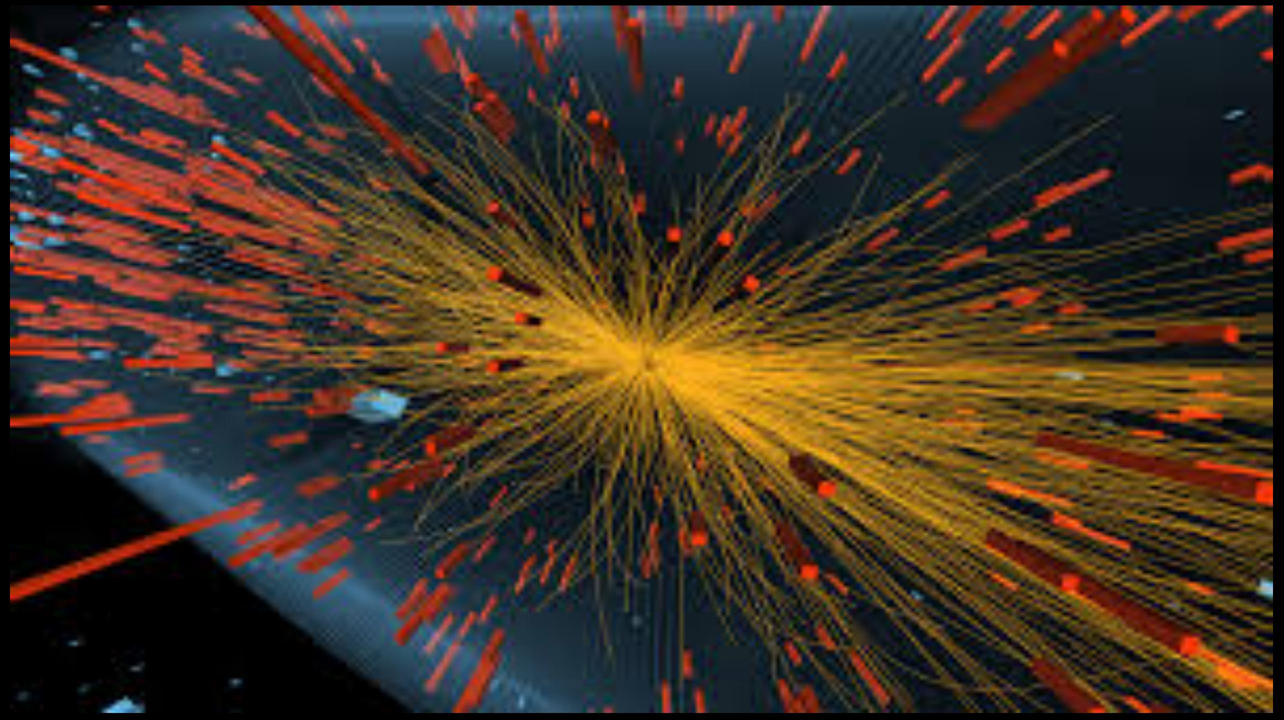
750 kHz

7.5 kHz

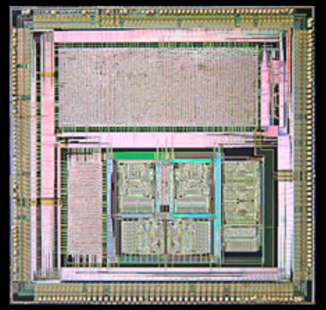


Detector

- Collisions every 25 ns
- Detector front-end **ASICs**

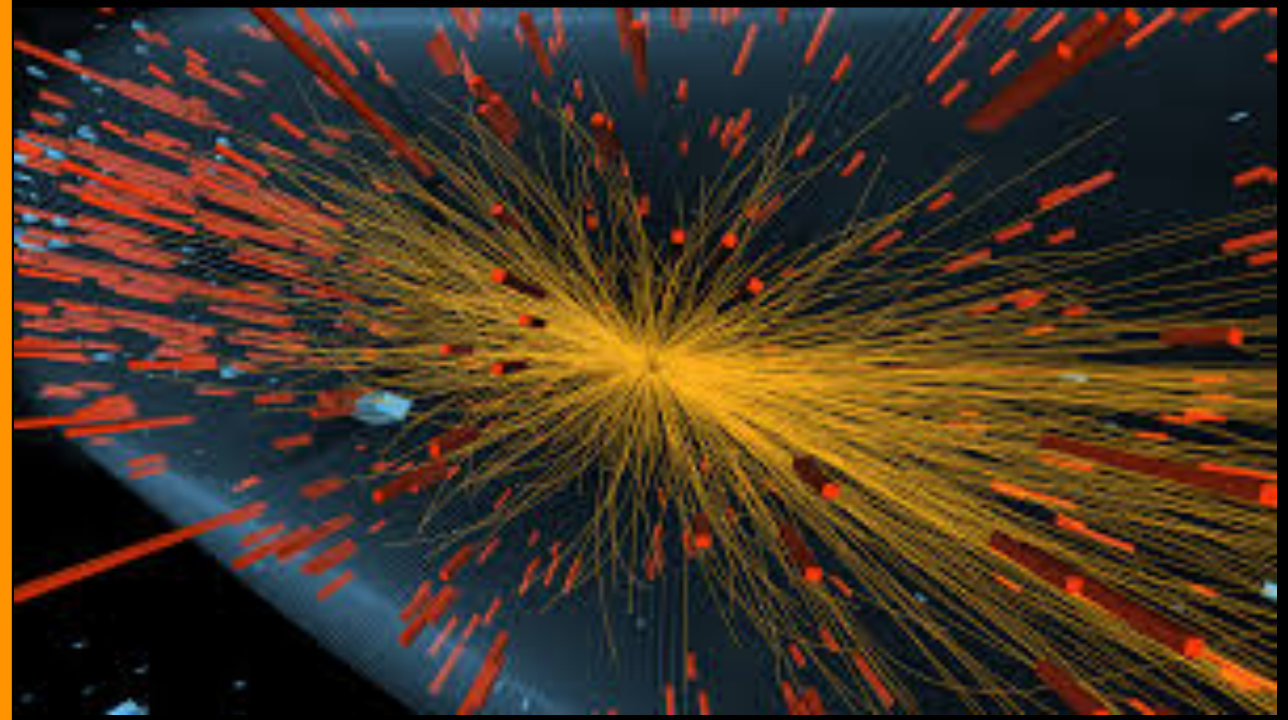


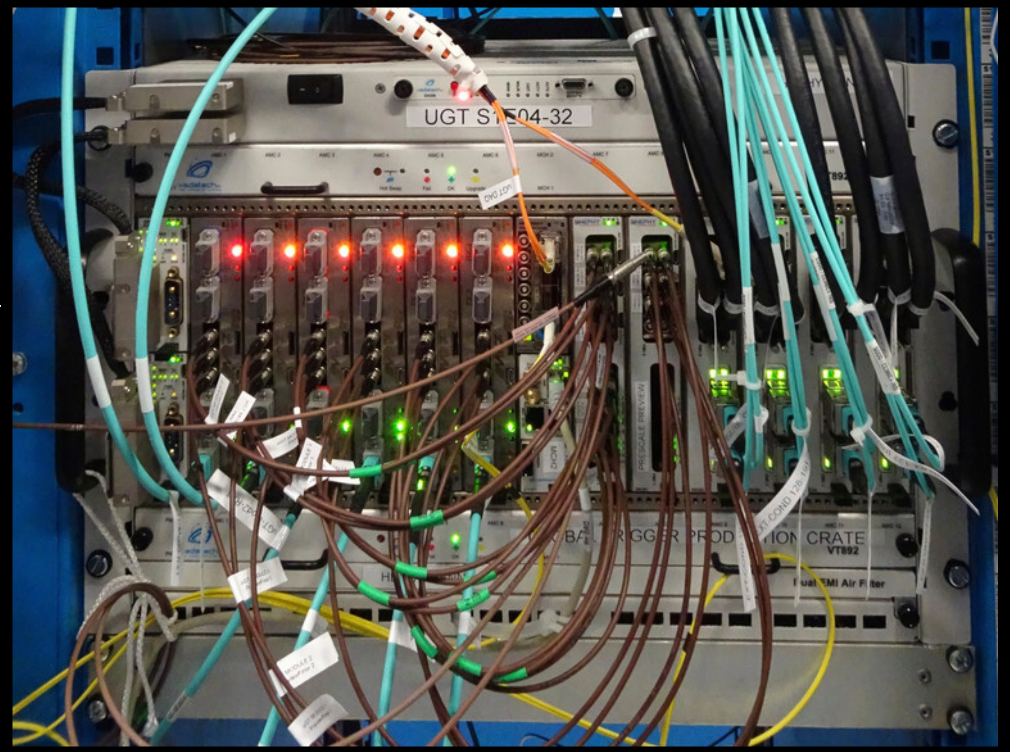
FPGA inference



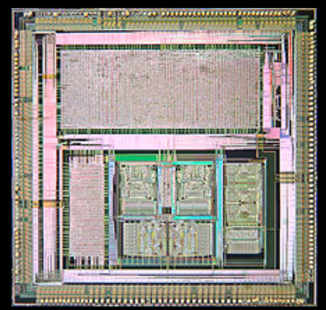
ASIC inference

Nanosecond inference on specialised hardware

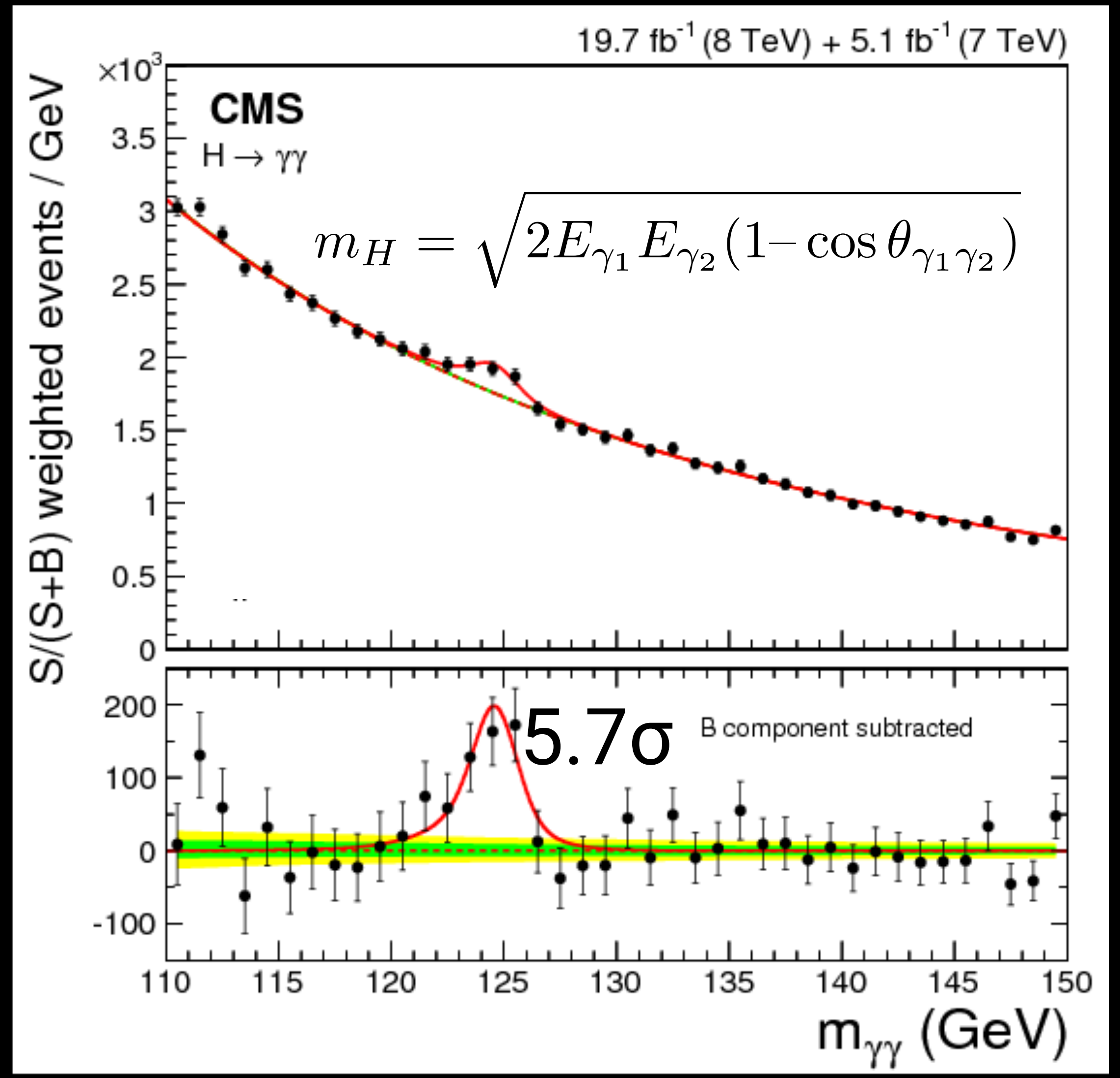




FPGA inference*



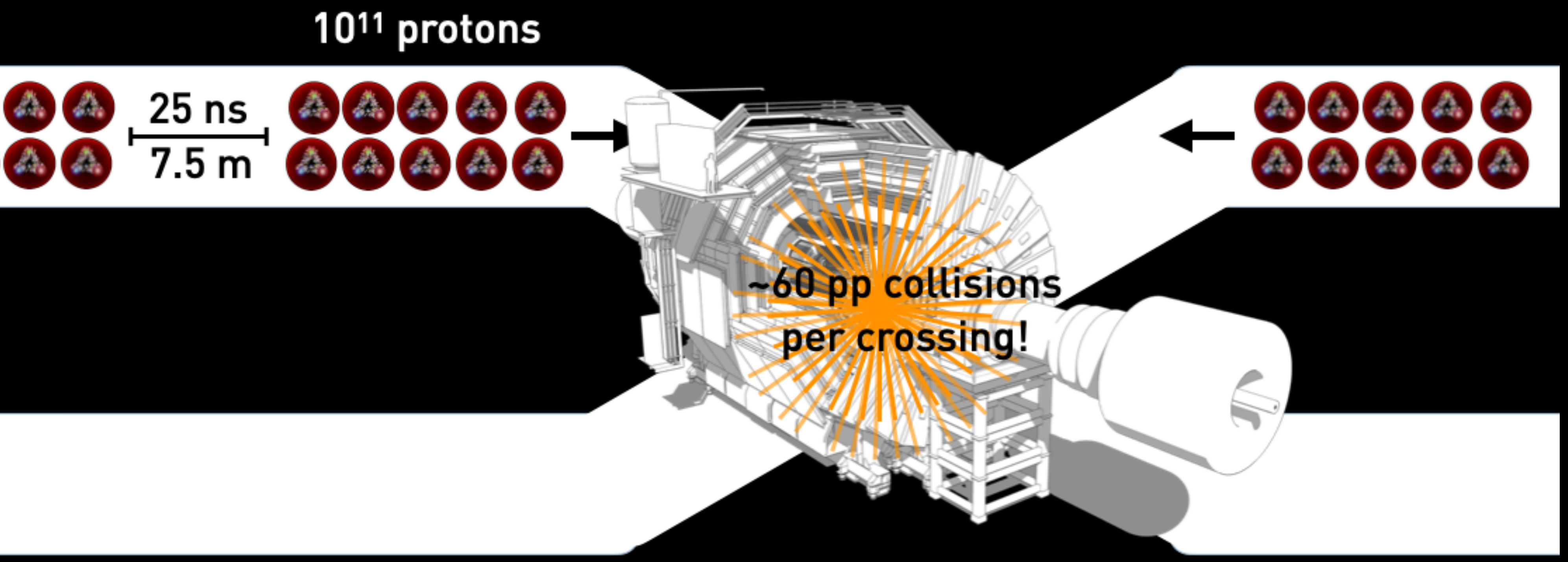
ASIC inference*



*examples in Jennifers talk

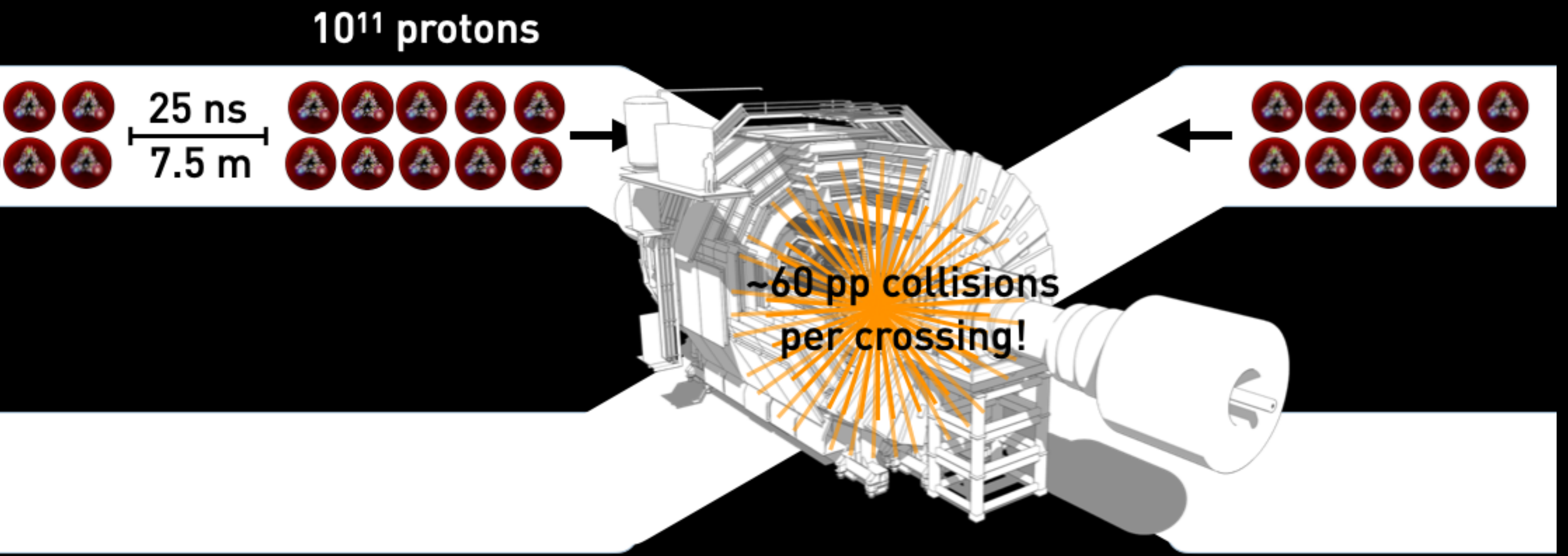
Low latency

- Strictly limited by collisions occurring every 25 ns



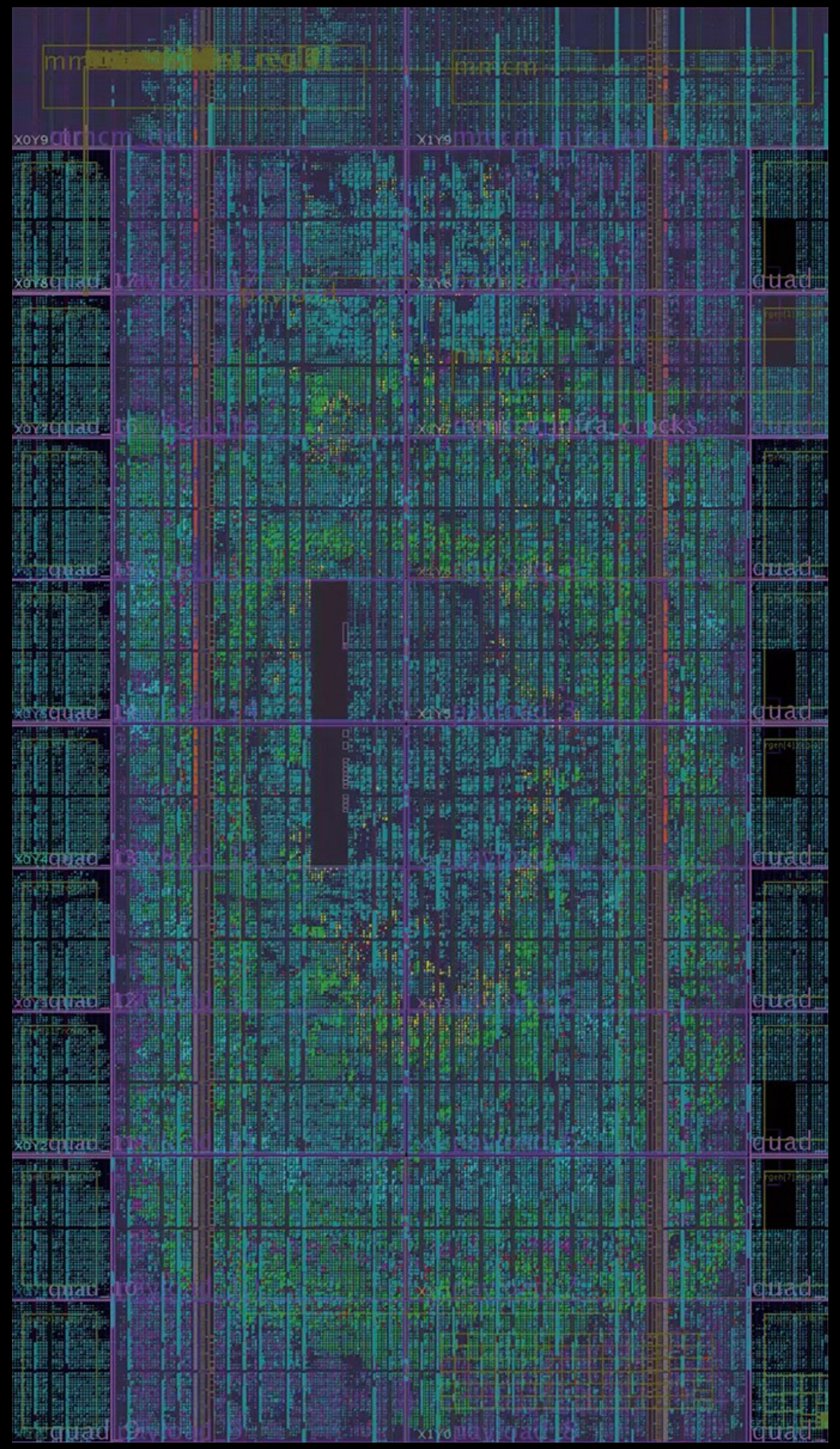
Low latency

- Strictly limited by collisions occurring every 25 ns



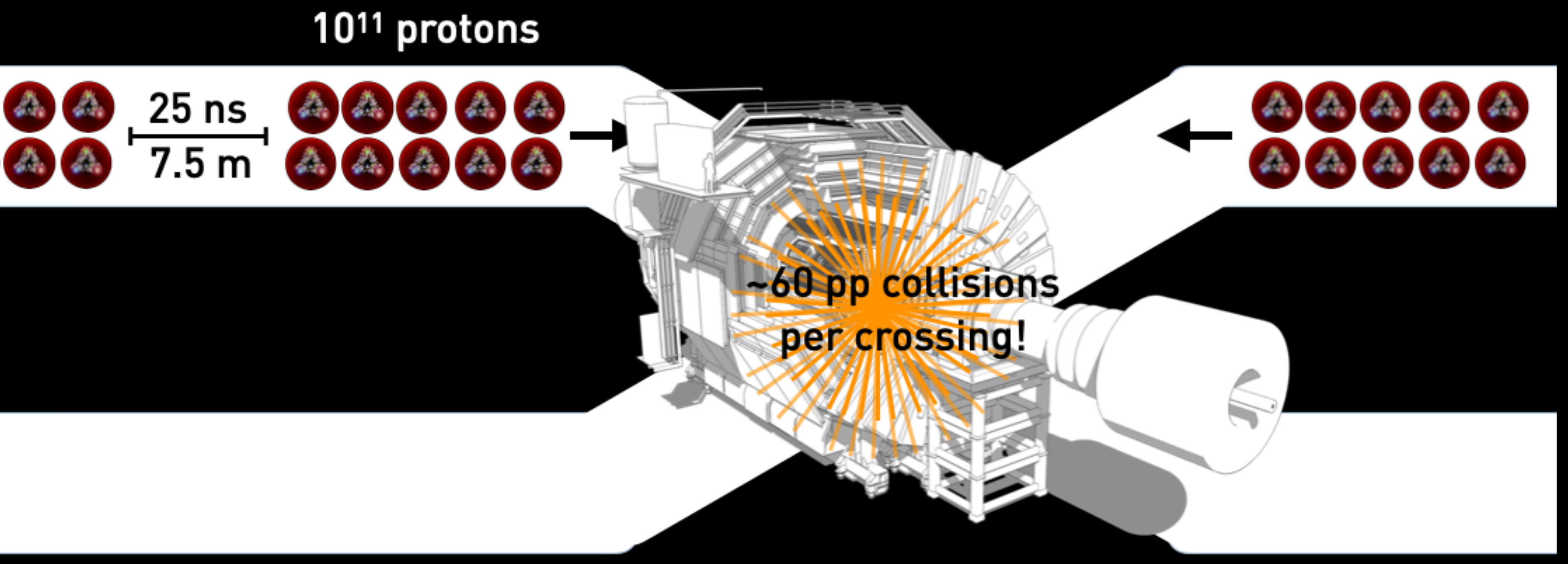
Low resource usage

- Several algos in parallel on single device



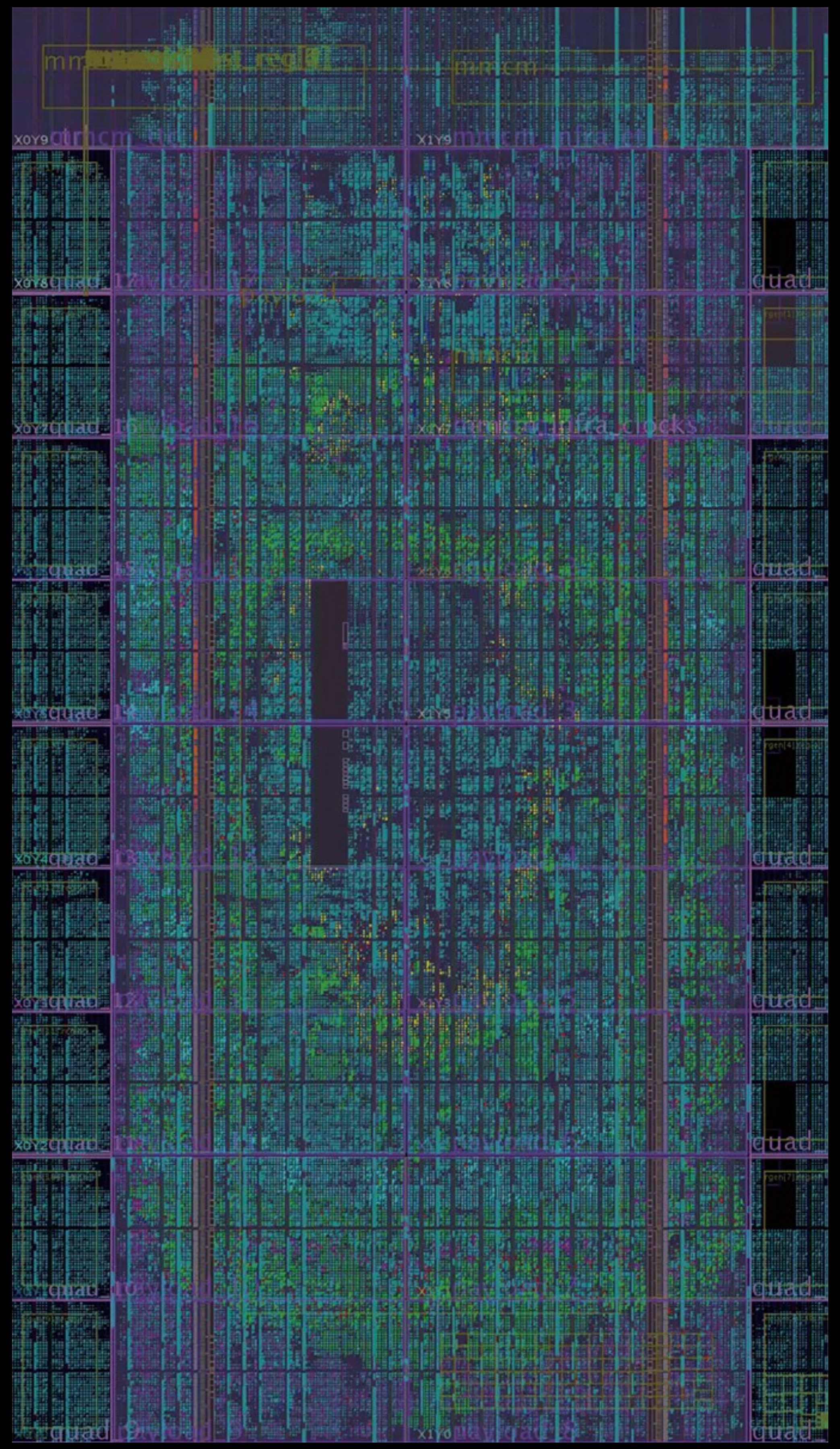
Low latency

- Strictly limited by collisions occurring every 25 ns



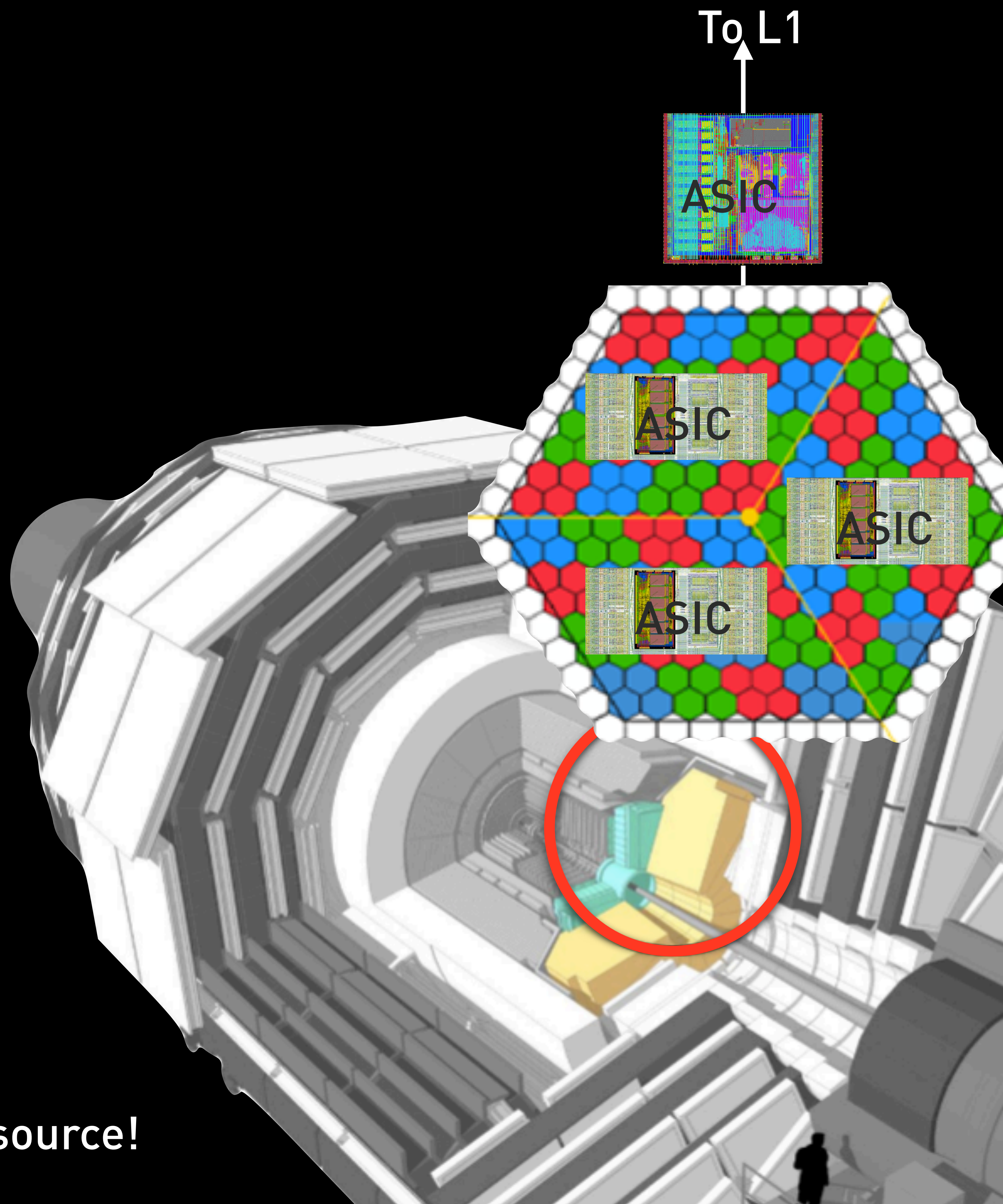
Low resource usage

- Several algos in parallel on single device



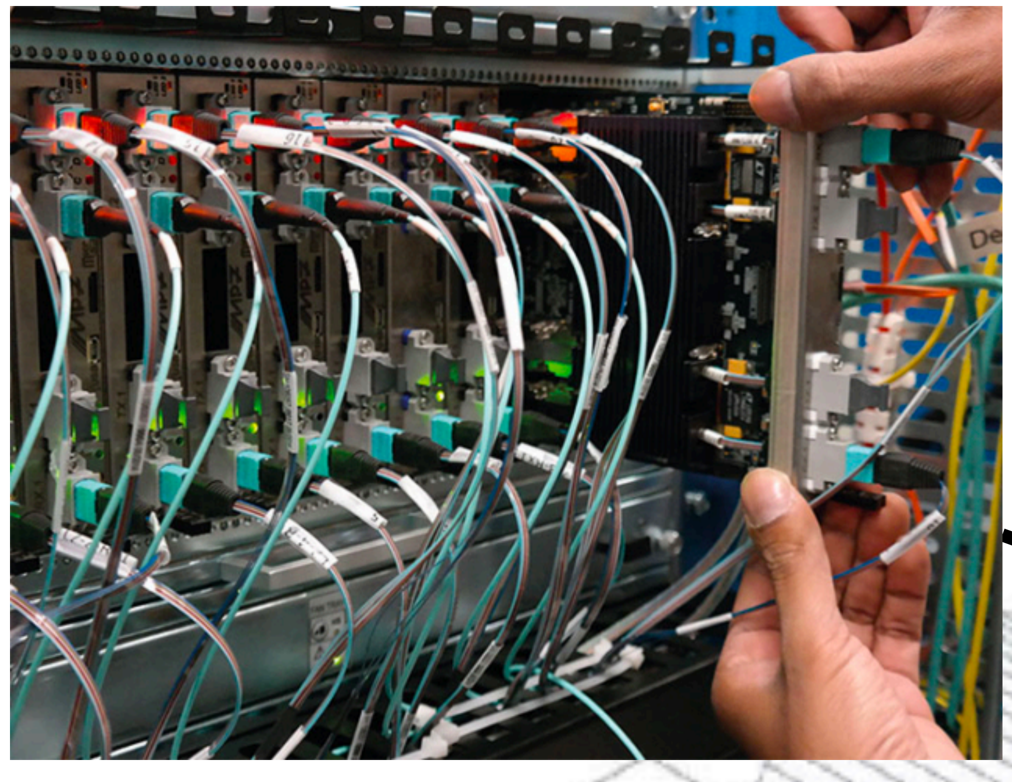
Low power

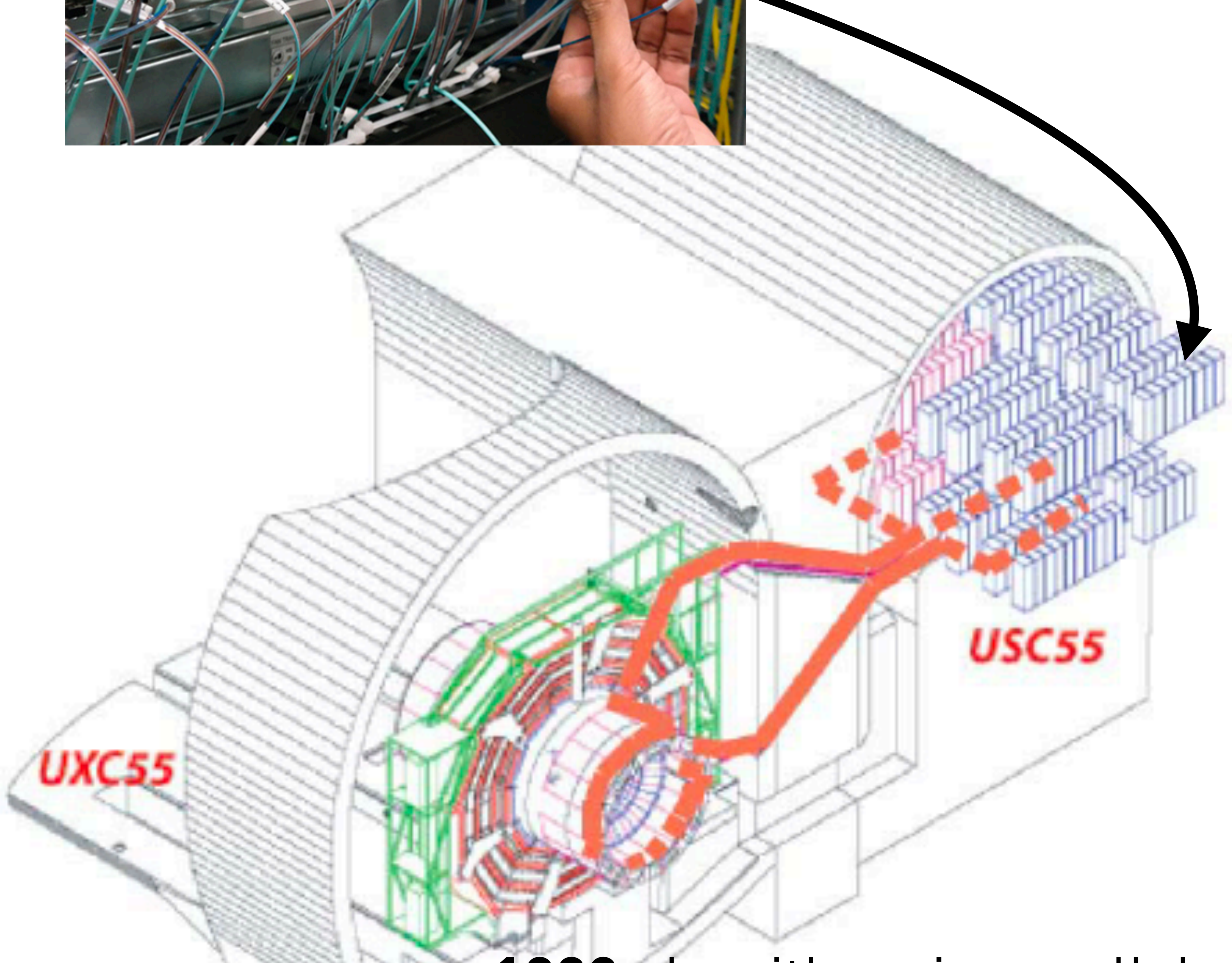
- On detector: cooled to -30°C
- L1: Cooling, cooling, cooling

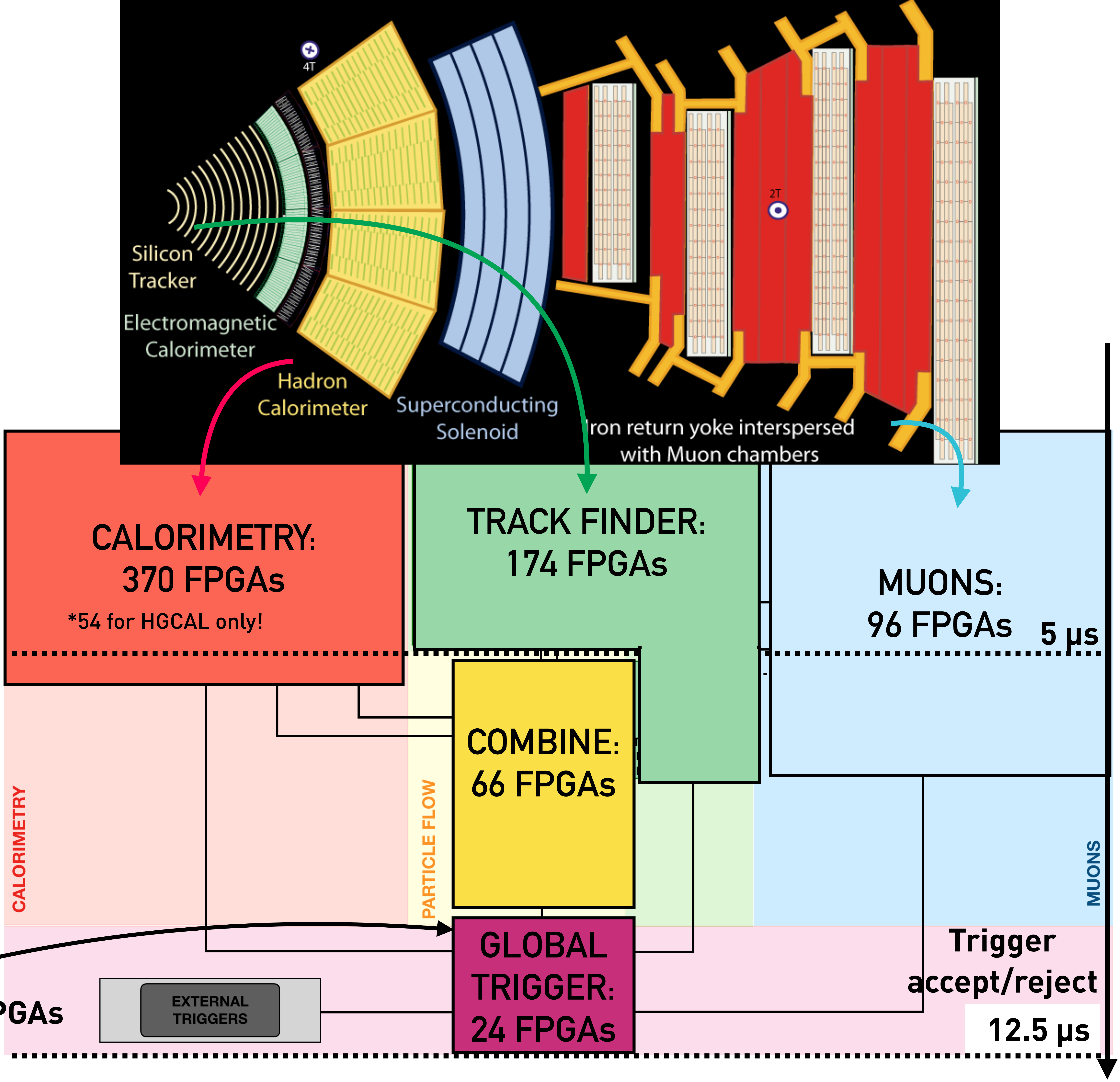


Extreme combination of low power, low latency, low resource!

The level-1 trigger







~1000 algorithms in parallel on **~10 FPGAs**

~100 ns latency per algorithm

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
  (our L1T FPGA processors pull currents of O(200)A at ~1V, dissipate **heat** of ~7W/cm^2
  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
  (our L1T FPGA processors pull currents of O(200)A at ~1V, dissipate **heat** of ~7W/cm^2
  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
  (our L1T FPGA processors pull currents of O(200)A at ~1V, dissipate **heat** of ~7W/cm^2 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?

Field Programmable Gate Arrays

- Different resources with programmable interconnects (reprogrammable)
- Originally ASIC prototyping, now also for high performance computing

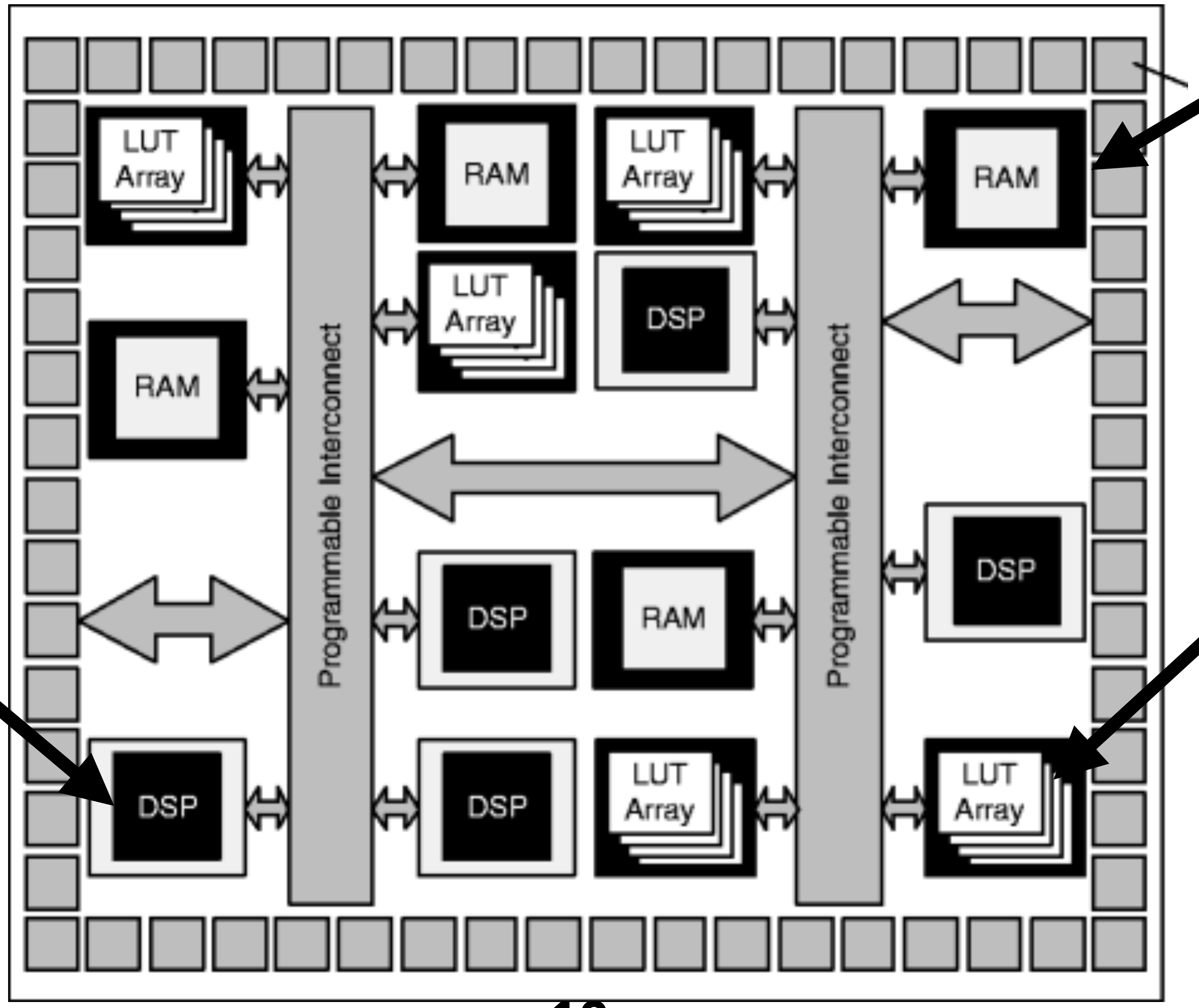
See Riccardo's talk!

What are FPGAs?

Field Programmable Gate Arrays

- Different resources with programmable interconnects (reprogrammable)
- Originally ASIC prototyping, now also for high performance computing

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

See Riccardo's talk!

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

activation function

precomputed and
stored in BRAMs

multiplication

DSPs

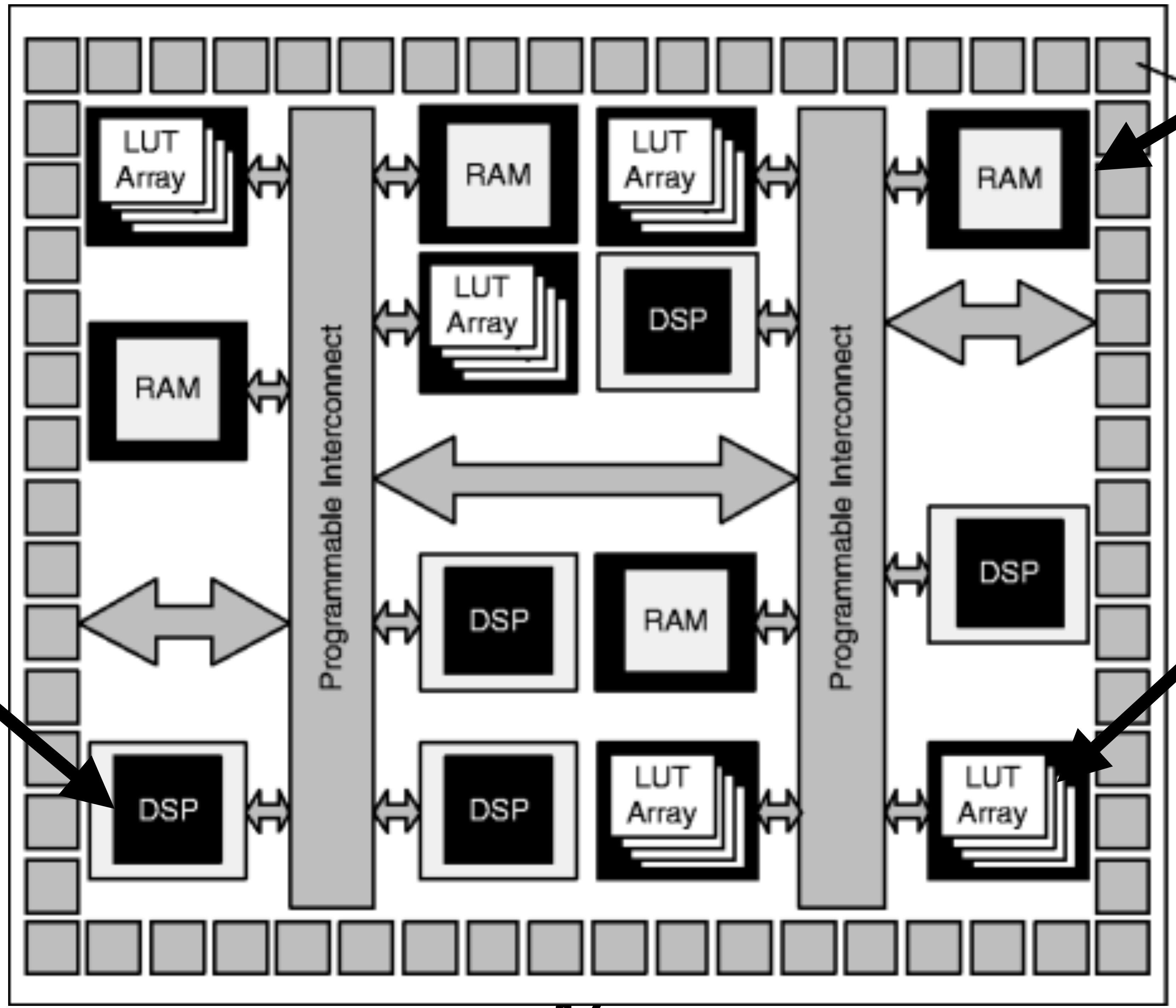
addition

logic cells

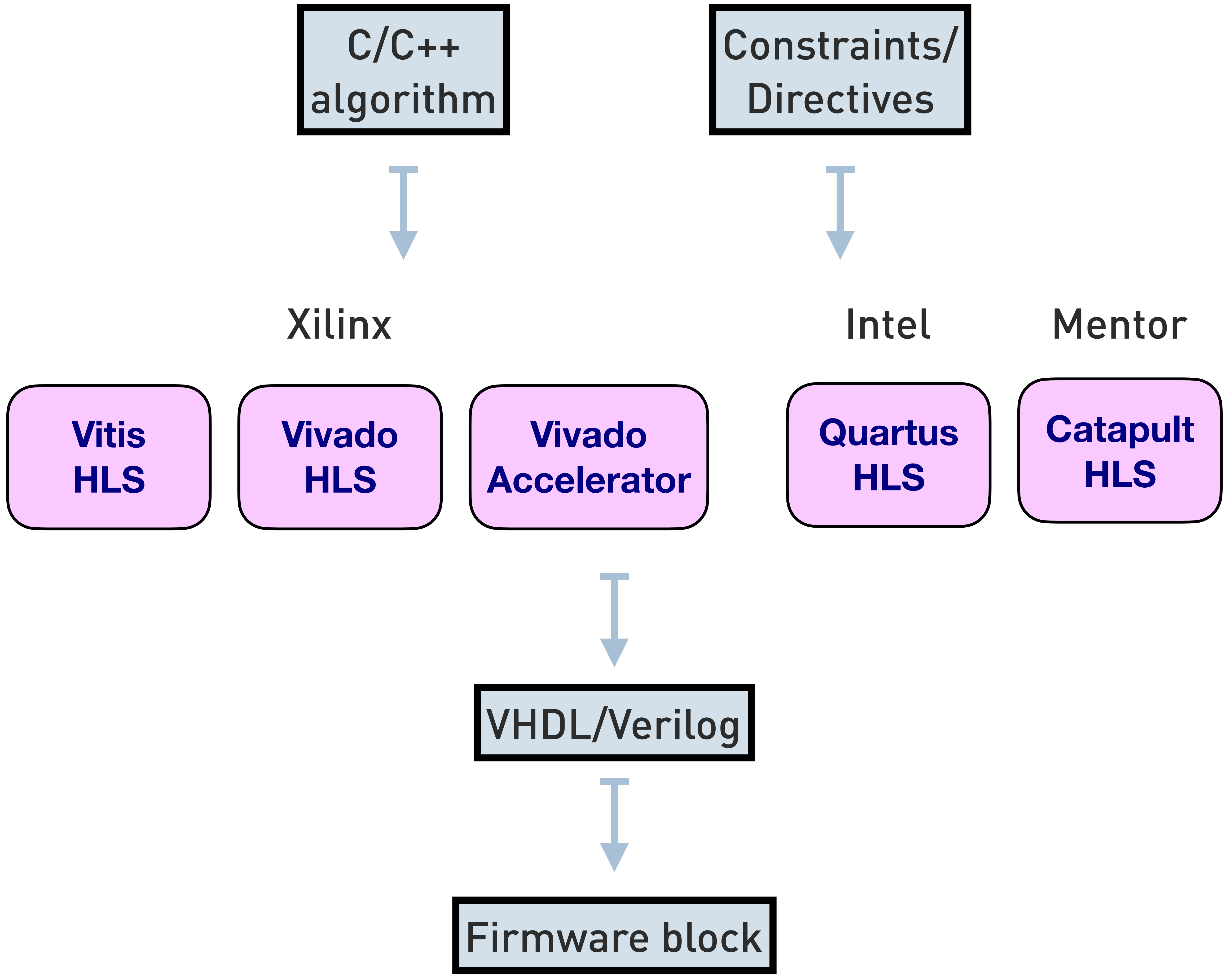
Memory (BRAM)

Digital signal processors (DSPs)

Logic cells/lookup tables (LUTs)

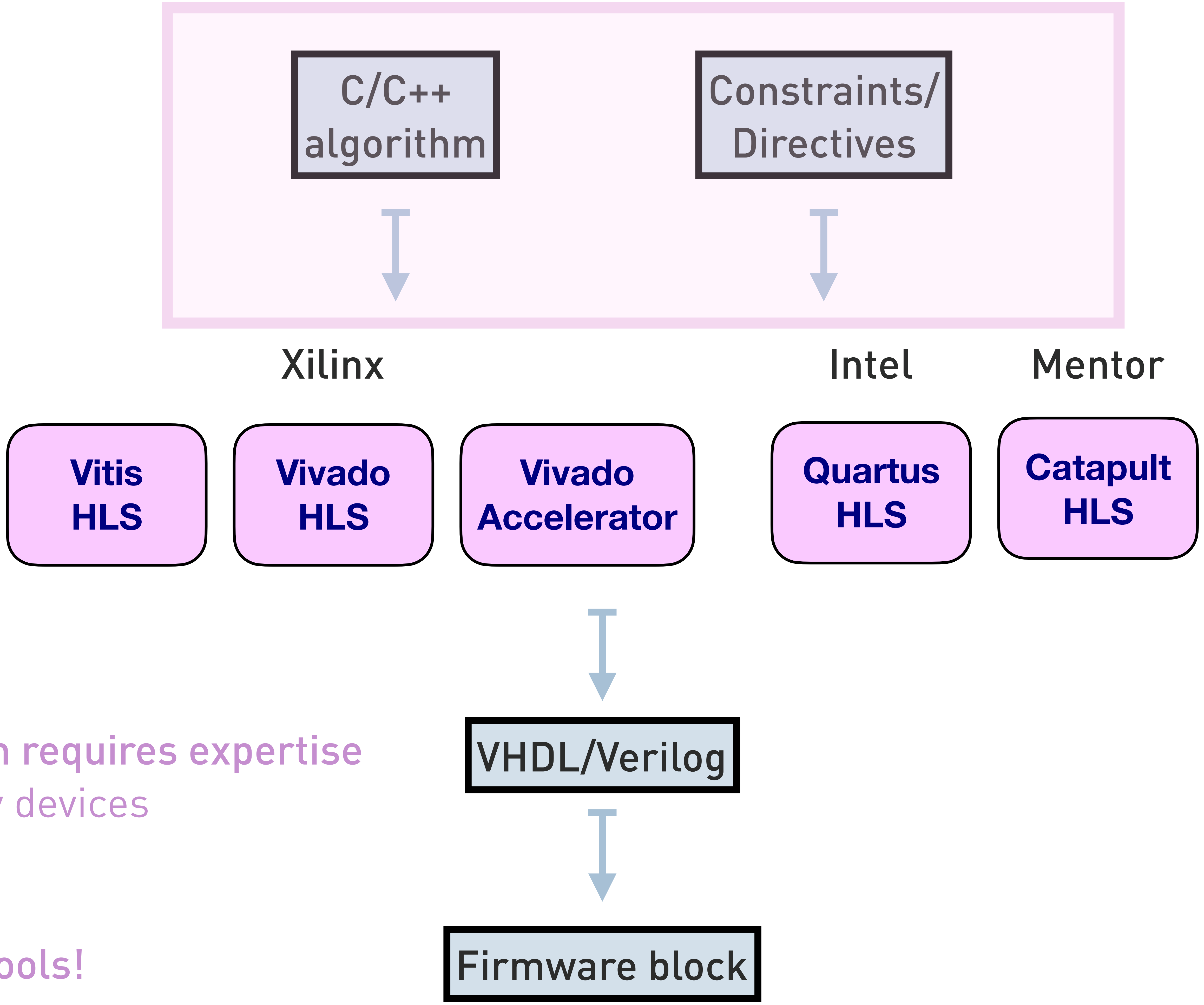


Programming an FPGA



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



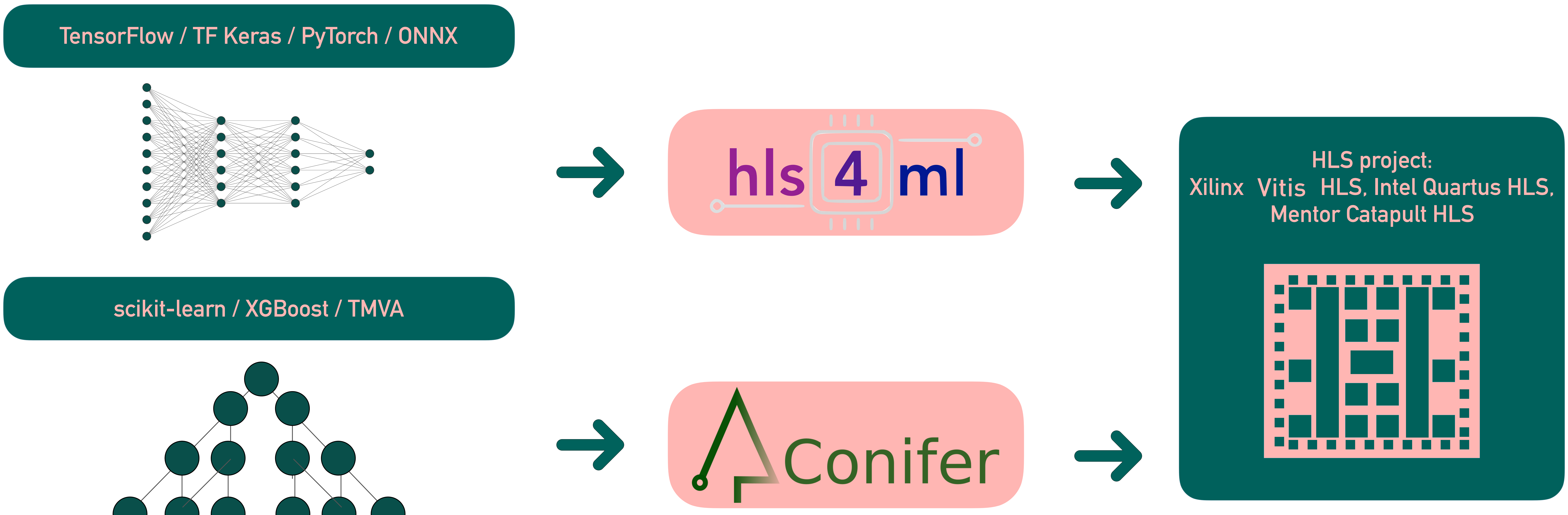
1. Write C-style code of function
2. Pass to a high-level synthesis (HLS) tool
3. HLS translates to hardware-description language (HDL)
4. Build firmware

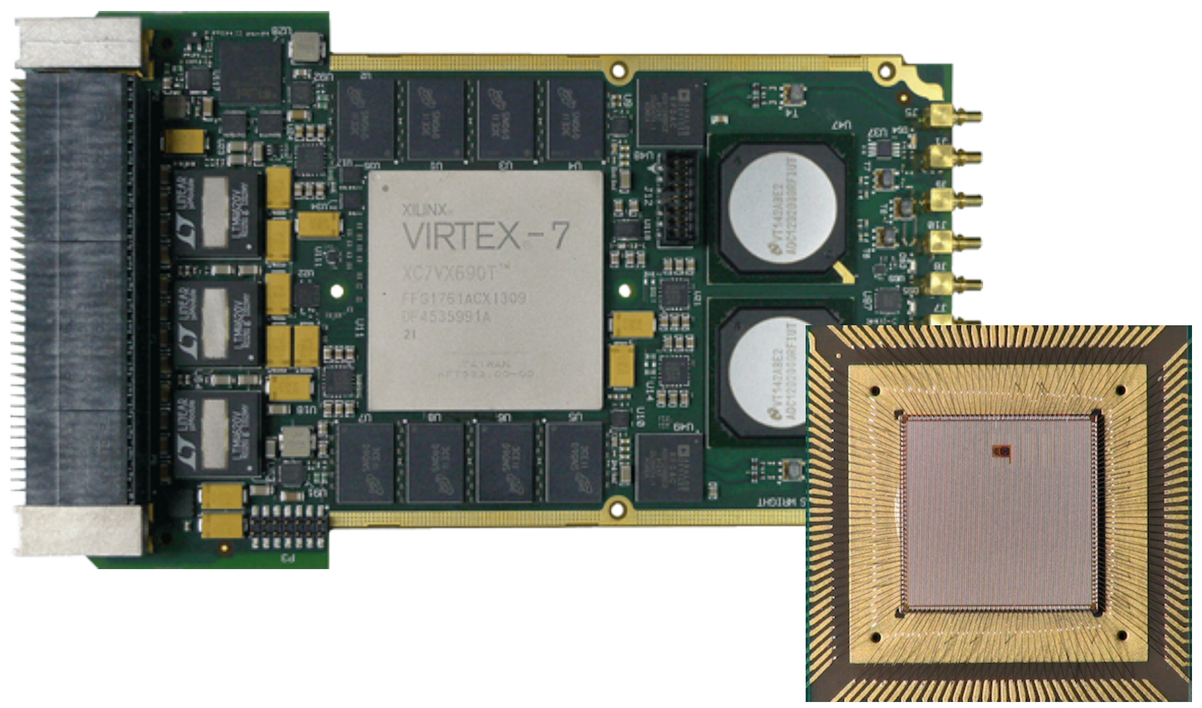
Efficient L1T firmware design requires expertise

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

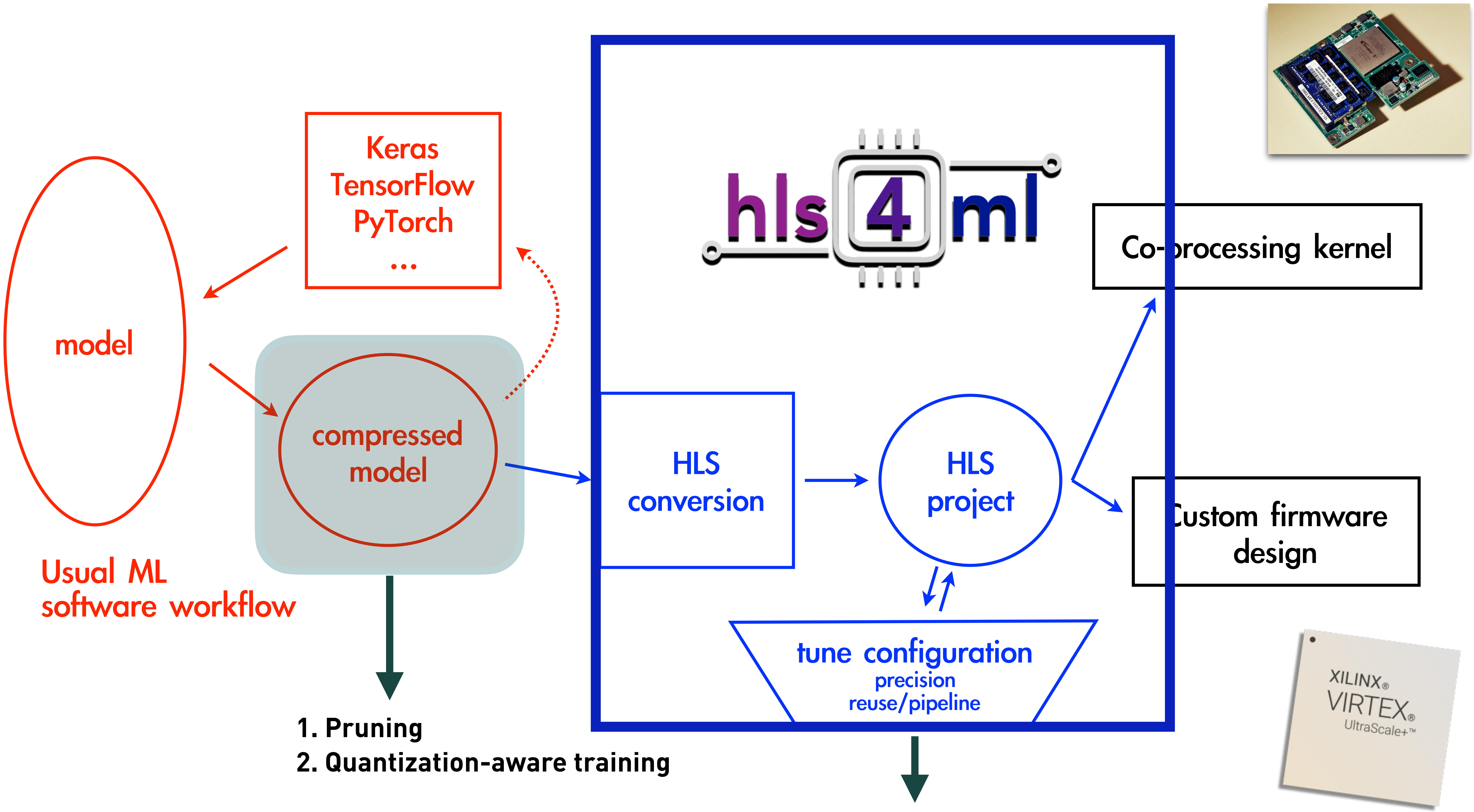
Not well served by industry tools!

See Riccardo's talk!





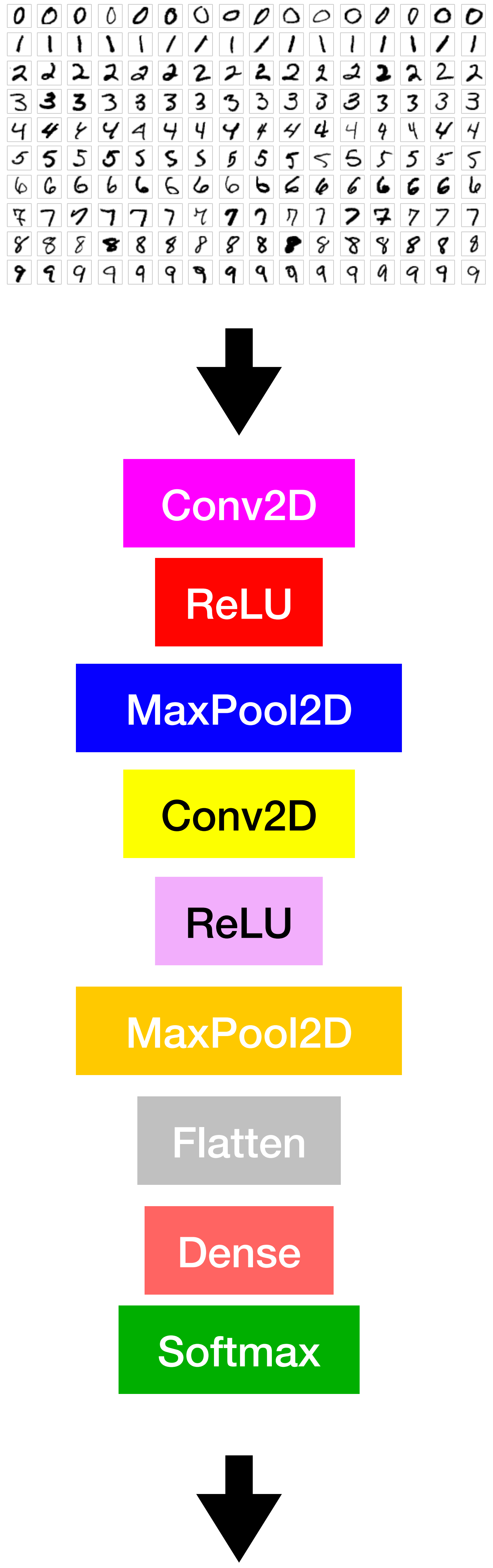

```
pip install hls4ml
pip install conifer
```



1. Pruning
2. Quantization-aware training

1. Parallelisation
2. Post-training quantization

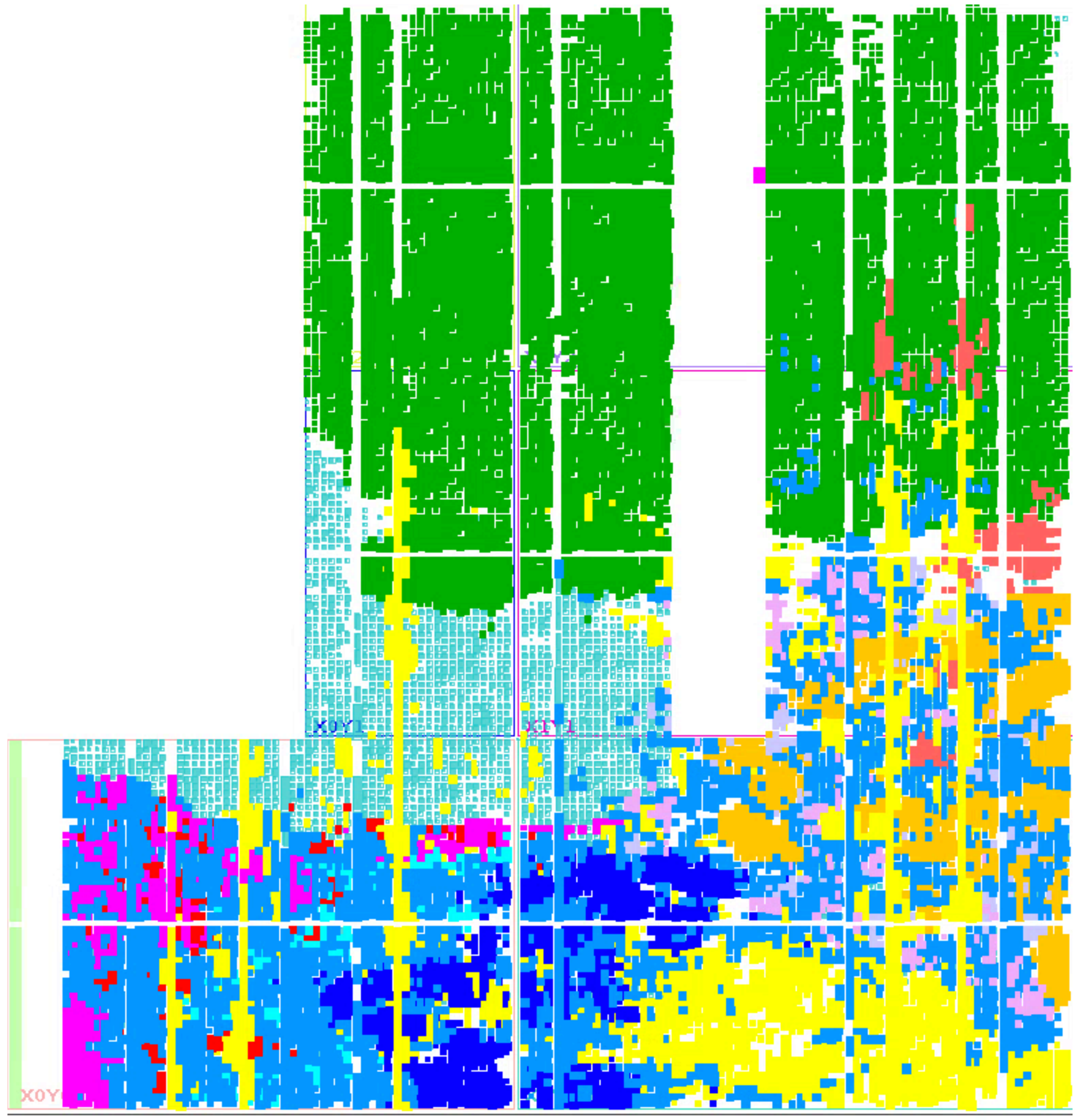
hls4ml



Prediction

```
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()
```

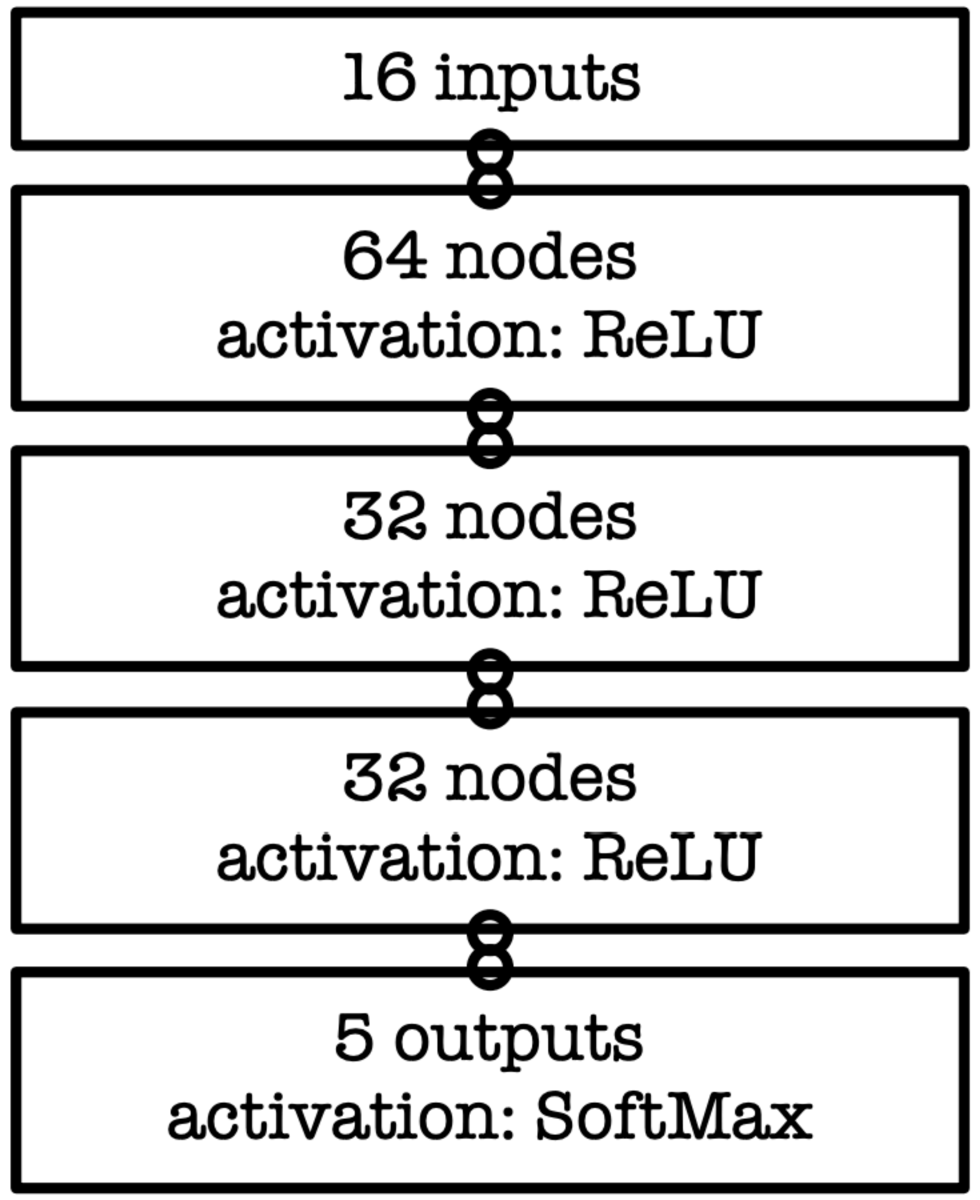
pynq-z2 floorplan

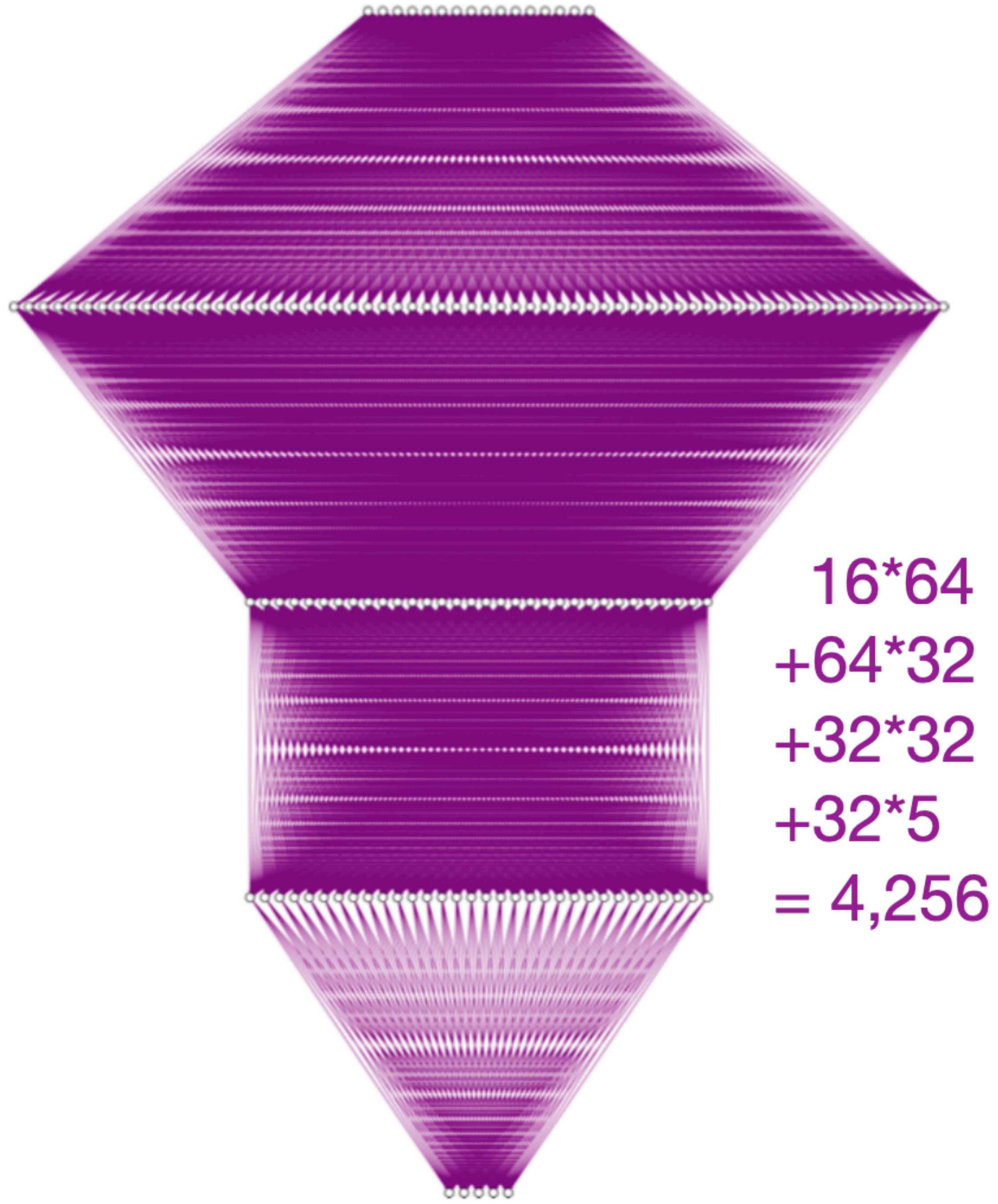


Learn how to use hls4ml in tomorrows tutorial by Sioni!

(from Sioni S Summers)

Compression





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

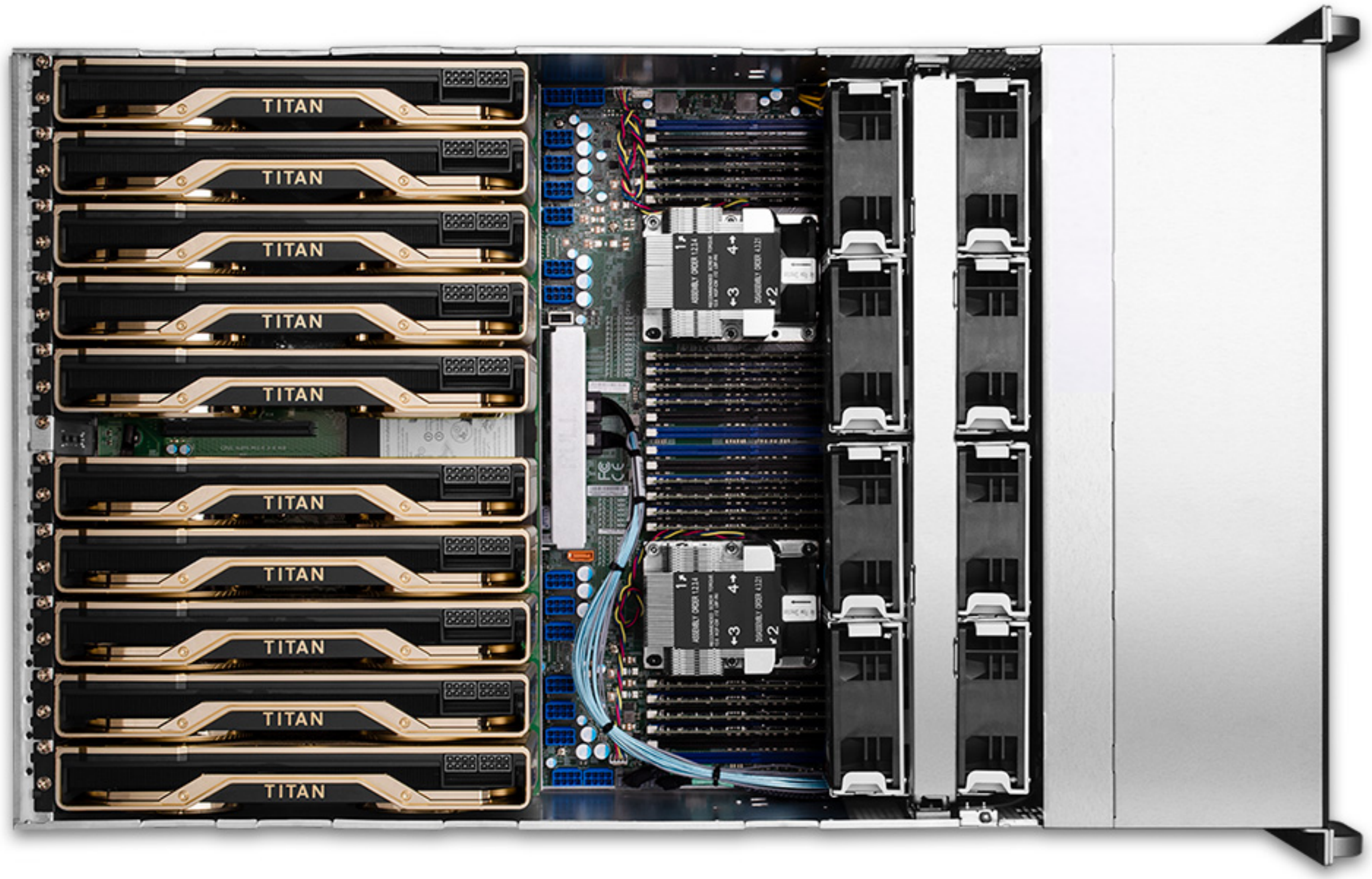
addition

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!

During training
- **Quantization**
- Pruning

Post-training
- Parallelisation (lower latency ↔ more resources)

From 8 GPU server to tiny FPGA!

See RIccardos talk and learn more in tomorrows tutorial by Sioni!

Quantization

Fixed point post-training quantization

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

Decimal: 3.25

During training: $-1^{S} \cdot 2^{E-127} \cdot (1.M)$

01000000010100000000000000000000

S Exponent Mantissa

On hardware: ap_fixed (W,I)

⟨00011.01⟩

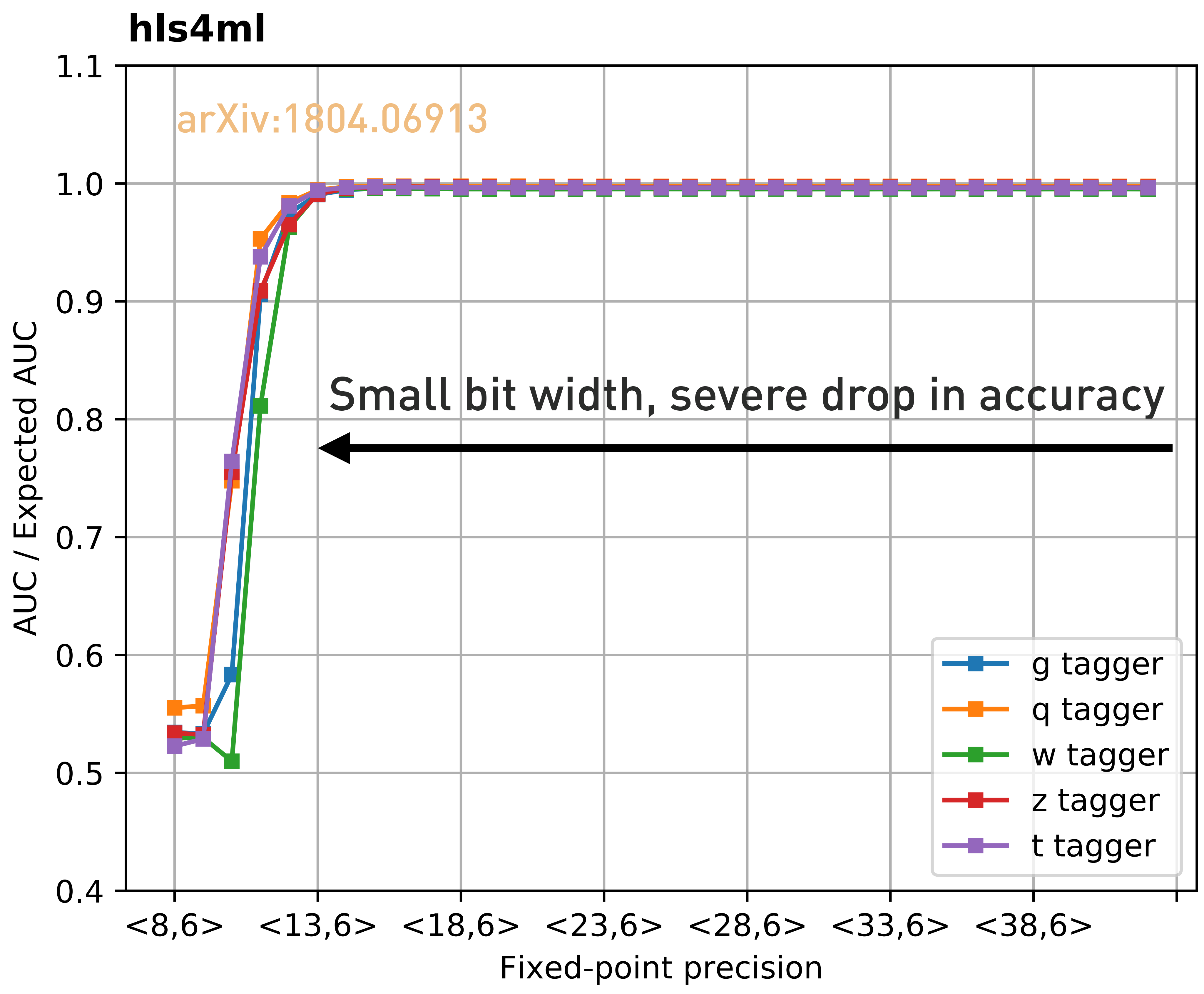
I

W

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

Can we do better? Yes!

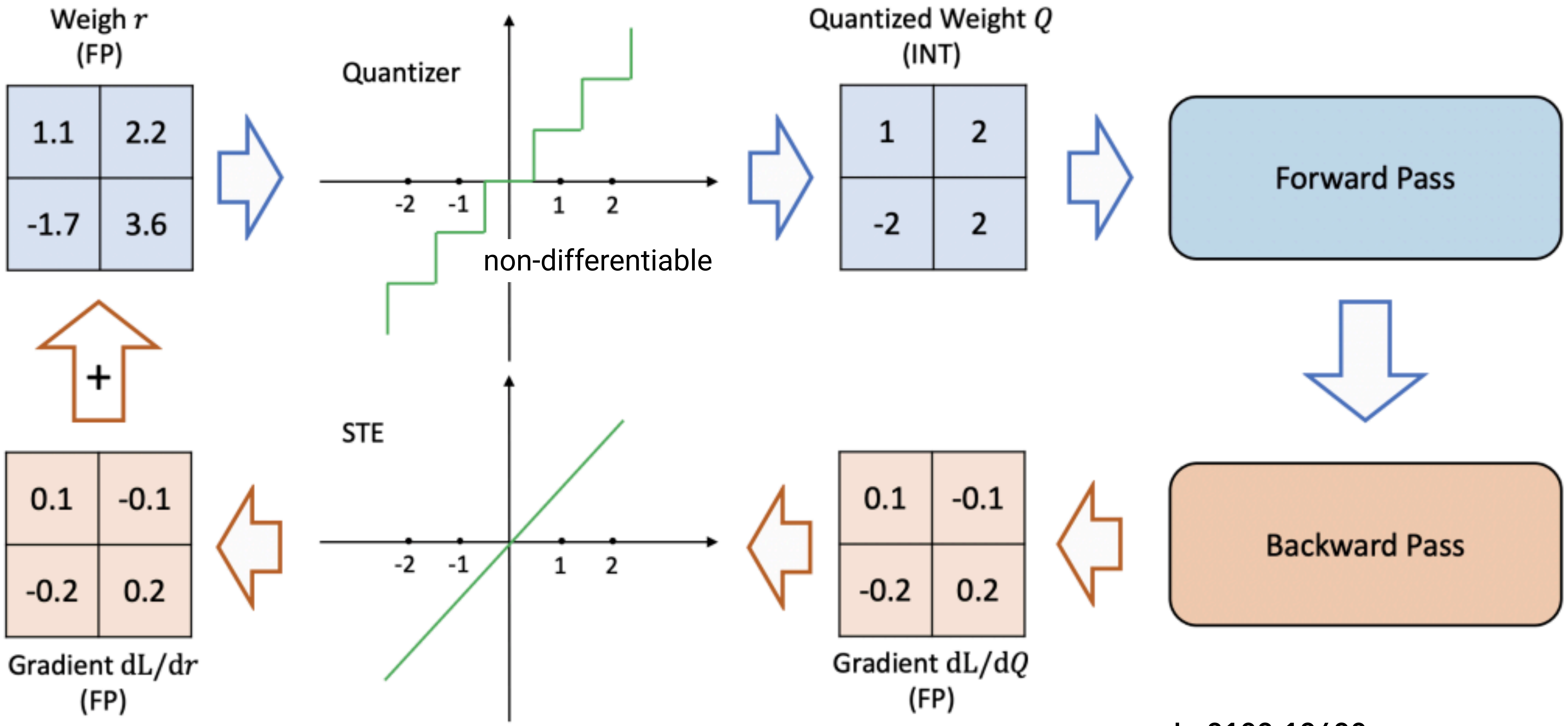
- Quantization-aware training (QAT)



See Riccardo's talk!

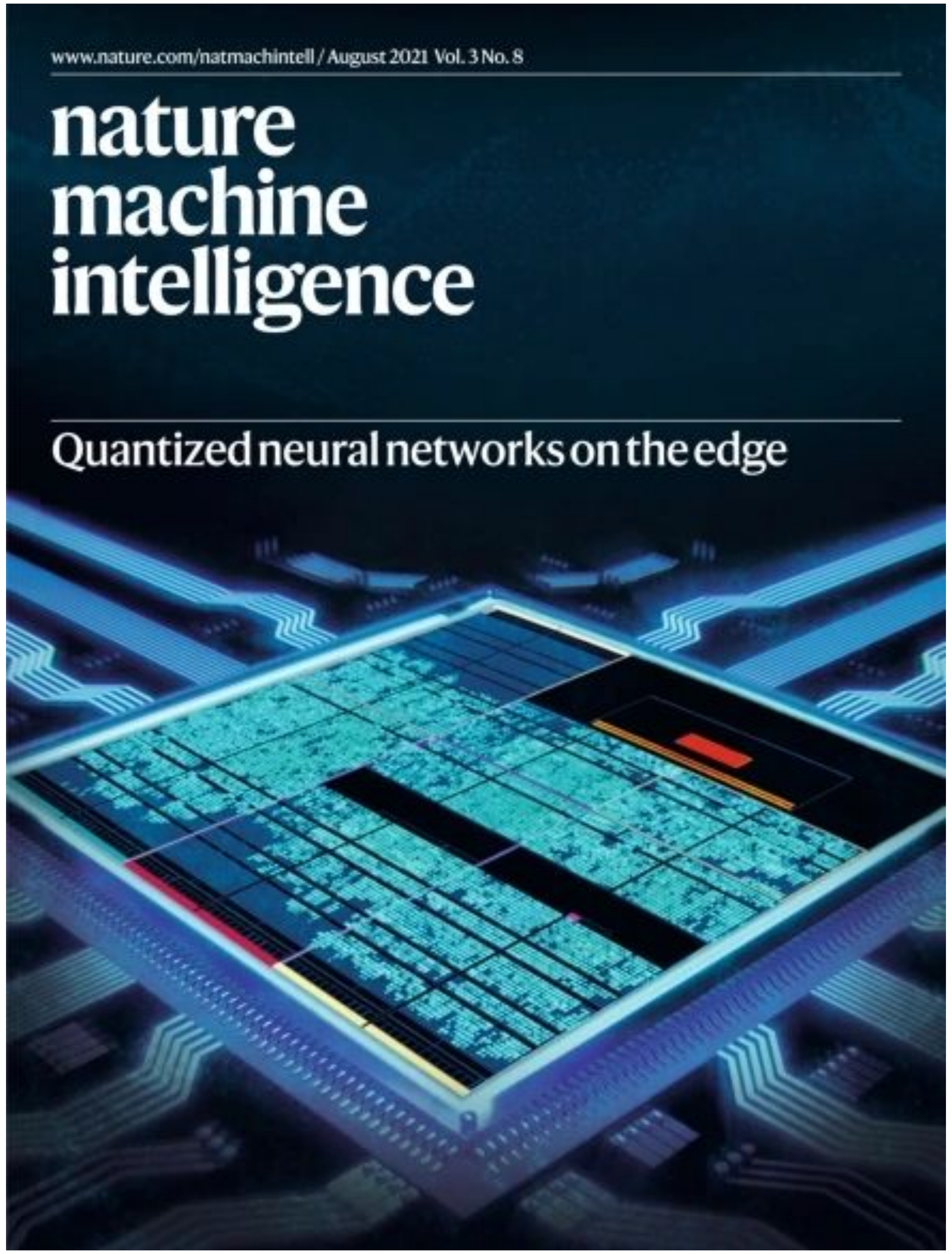
Quantization-aware training

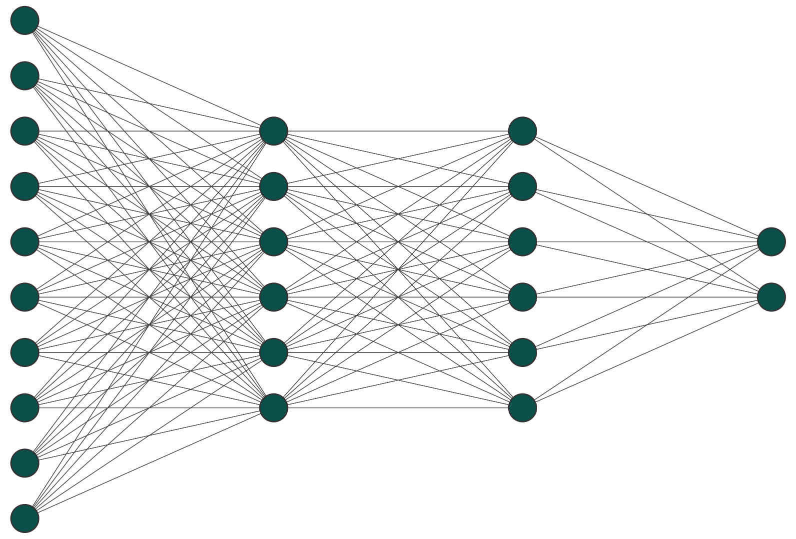
Lossless quantization for deep neural networks!



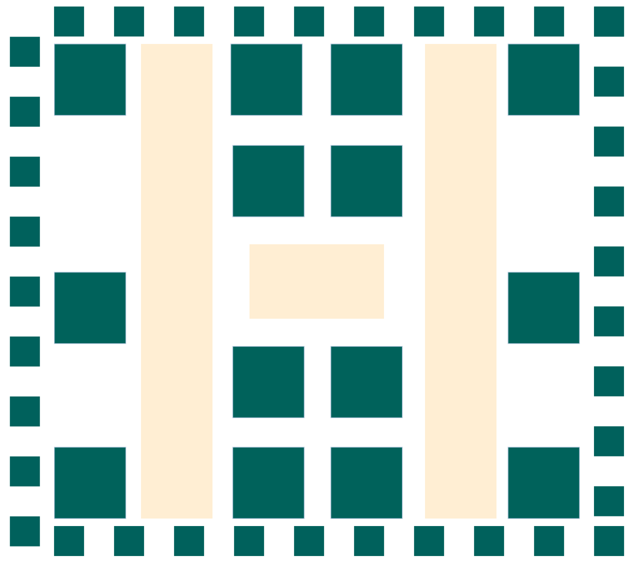
arxiv:2103.13630

Nature Machine Intelligence 3 (2021)





→ QKeras Quantization-aware training → QKeras model → hls4ml Fixed-point translation Parallelisation Firmware generation →

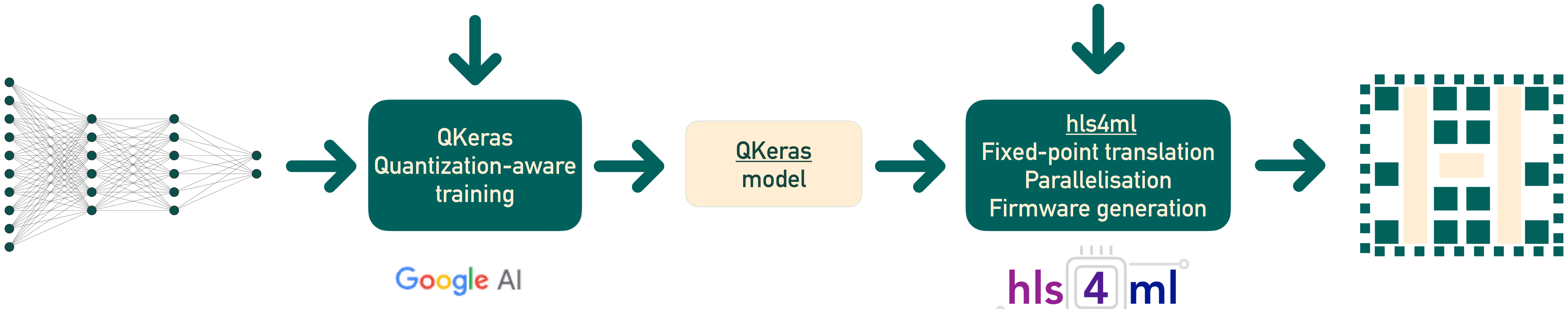


Google AI

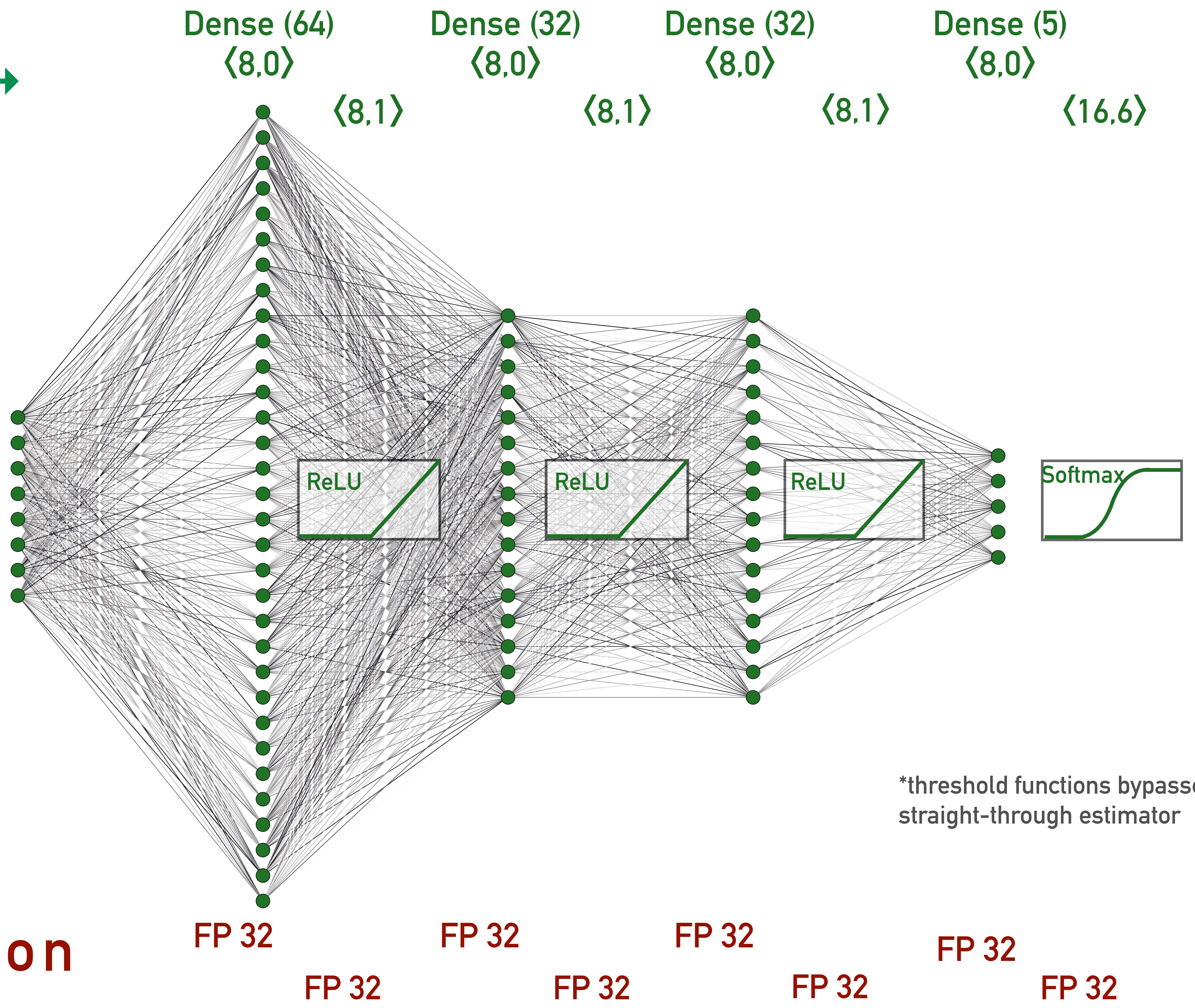
hls4ml

```python
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)
```

```python
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()
```



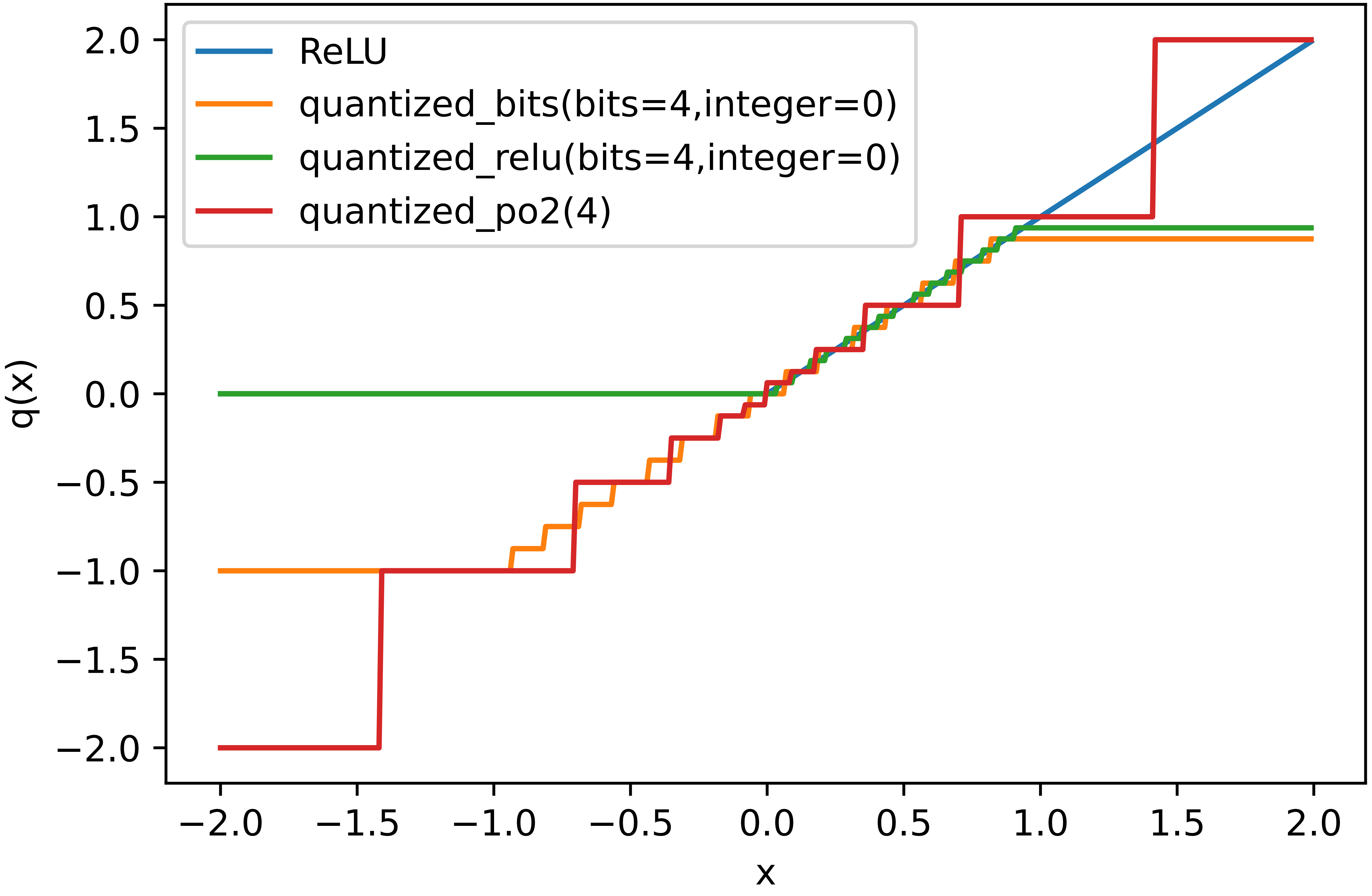
Forward pass →



*threshold functions bypassed in backward pass, straight-through estimator

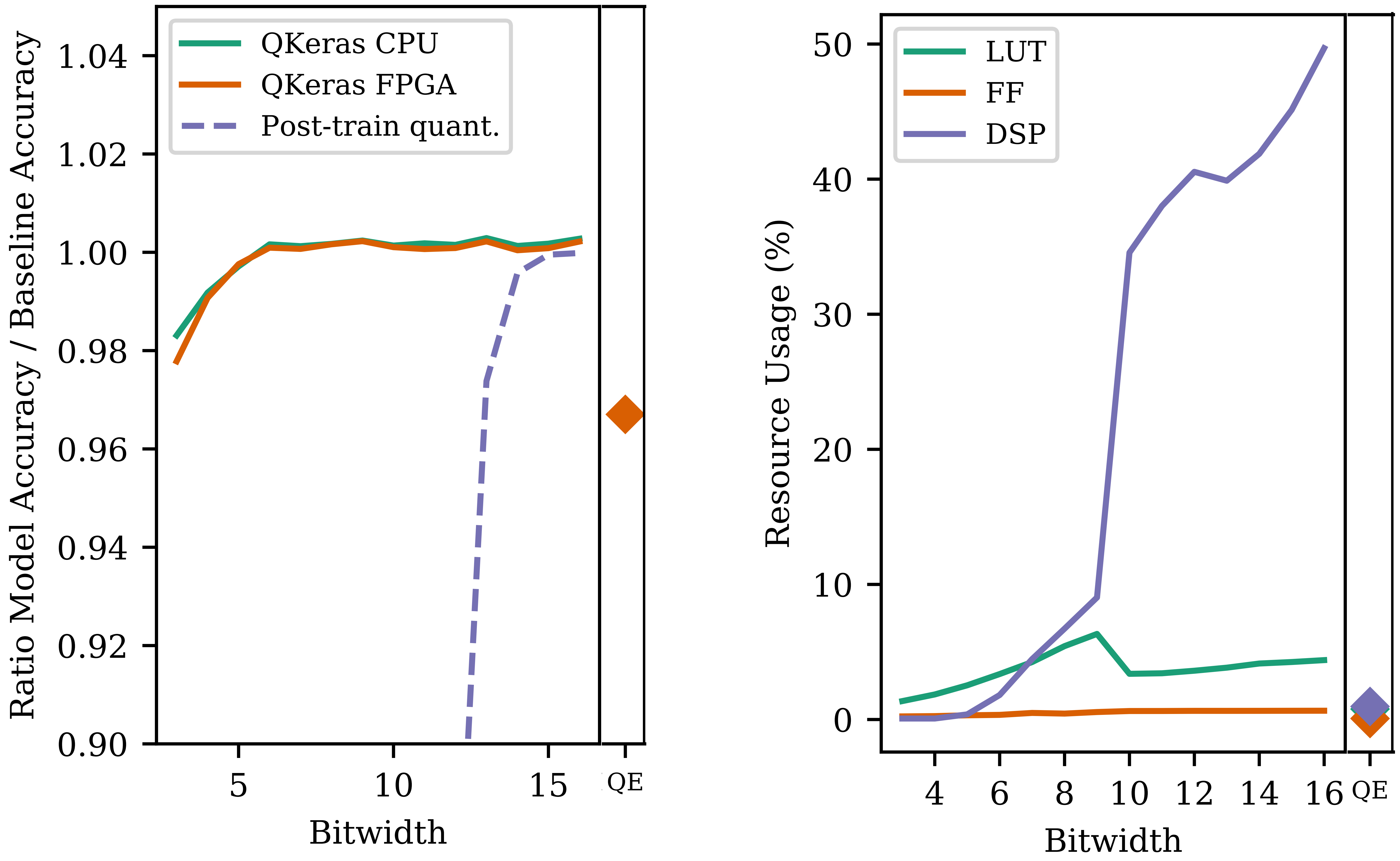
← Back propagation

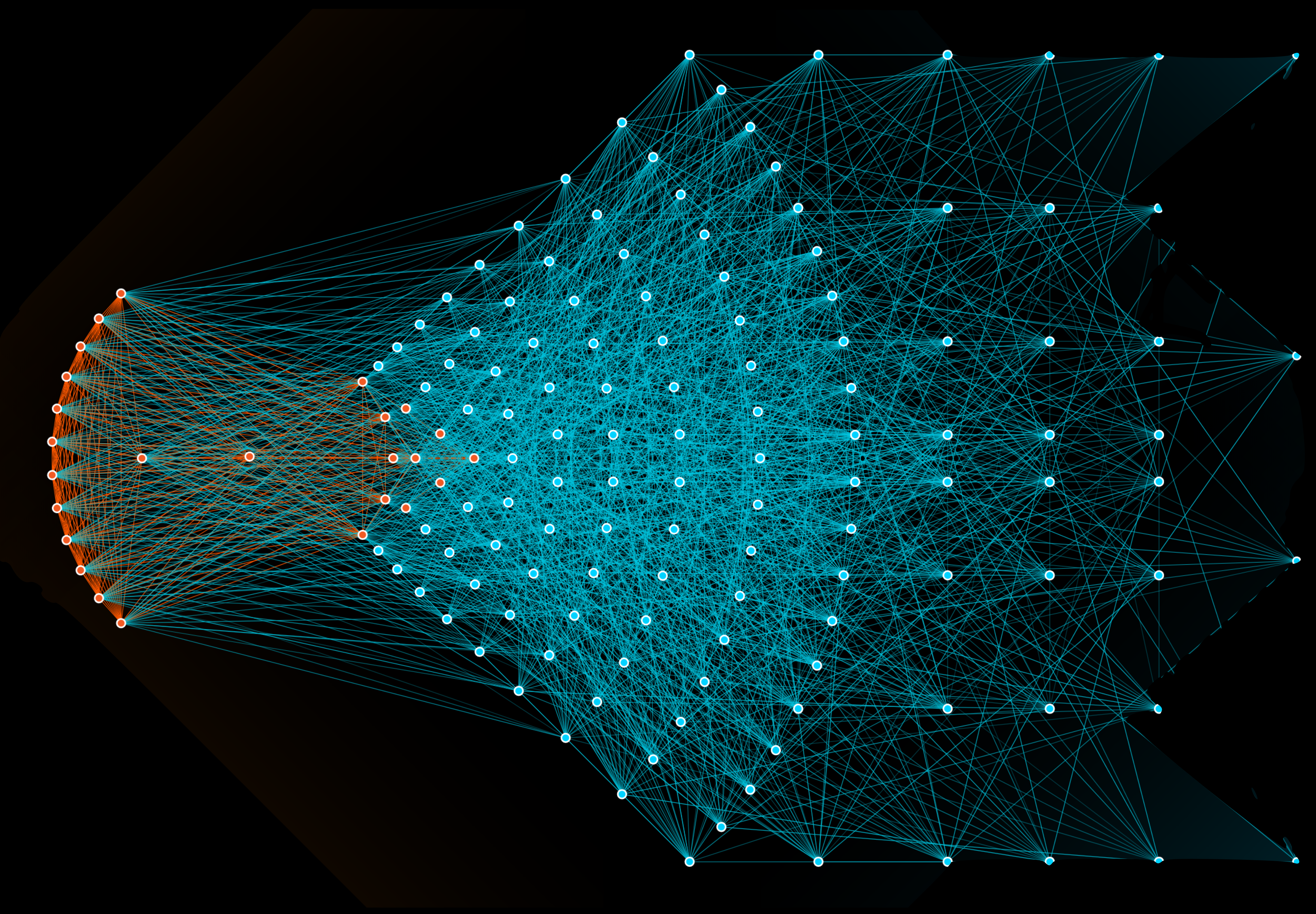
Quantization-aware training



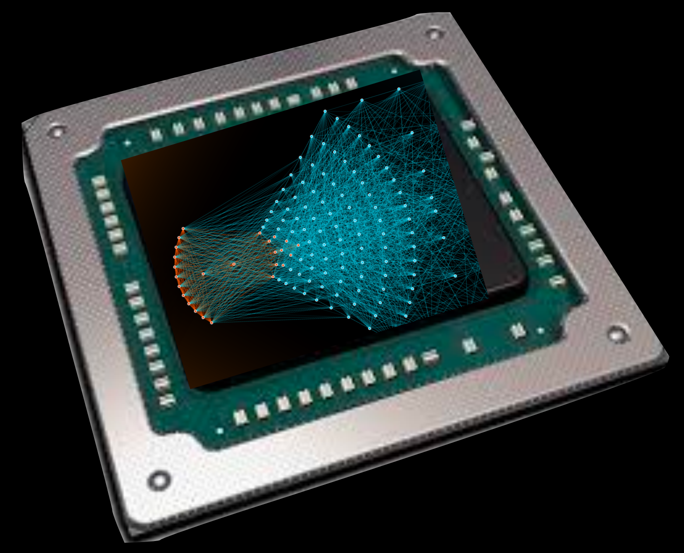
FPGA performance

Multiplications move to LUTs at bit width <10.





Ideally



Reality

QTools energy estimate

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

- heterogeneous quantization

QTools energy estimate

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 energy estimate

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

- Model size in bits
- 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

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

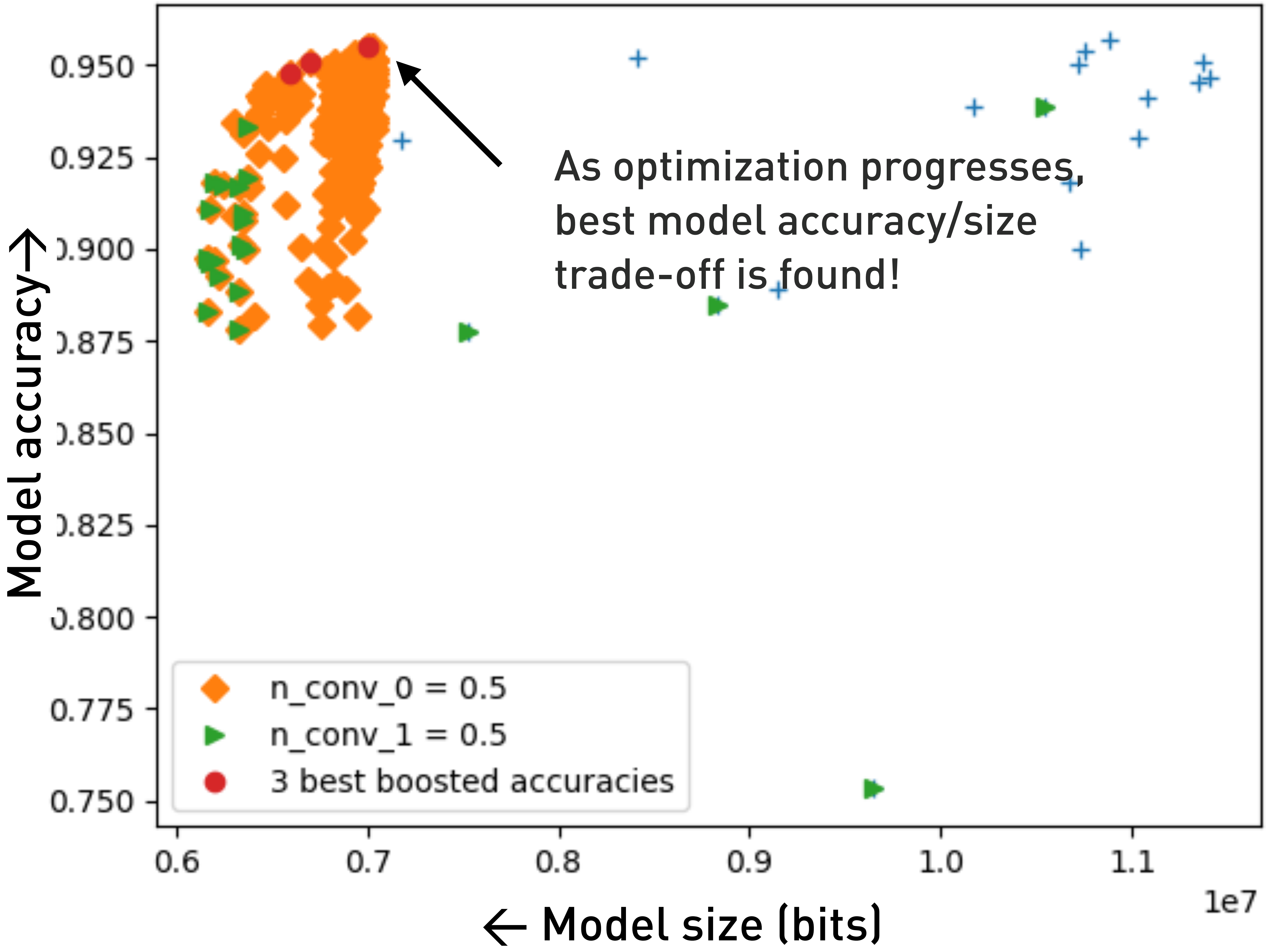
Maximize accuracy + minimizing cost in hyper parameter scan over quantizers:

AutoQKeras

AutoQKeras

AutoQ Bayesian optimization at work!

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

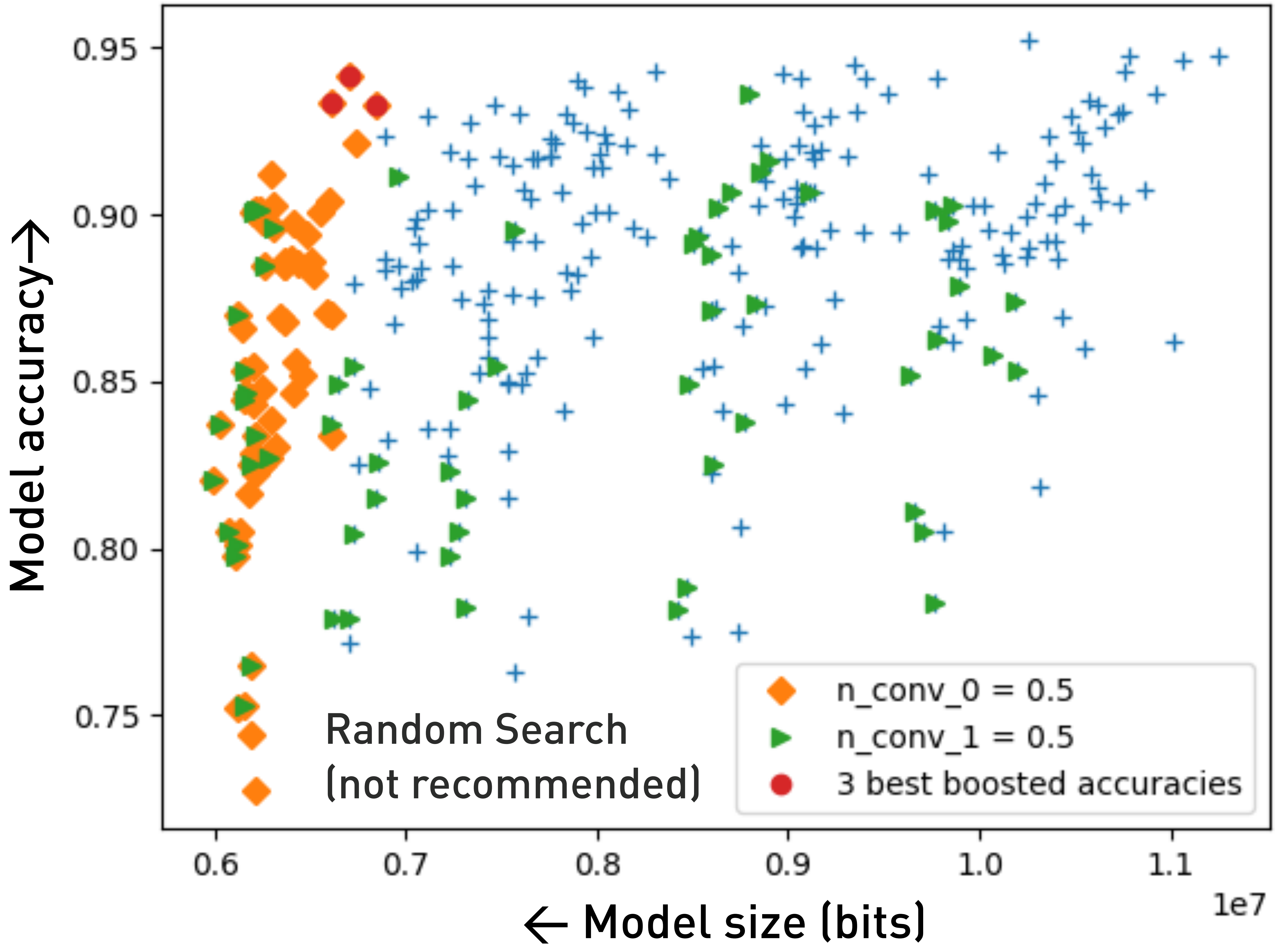


H. Linander, C. Petersson (Zenseact)

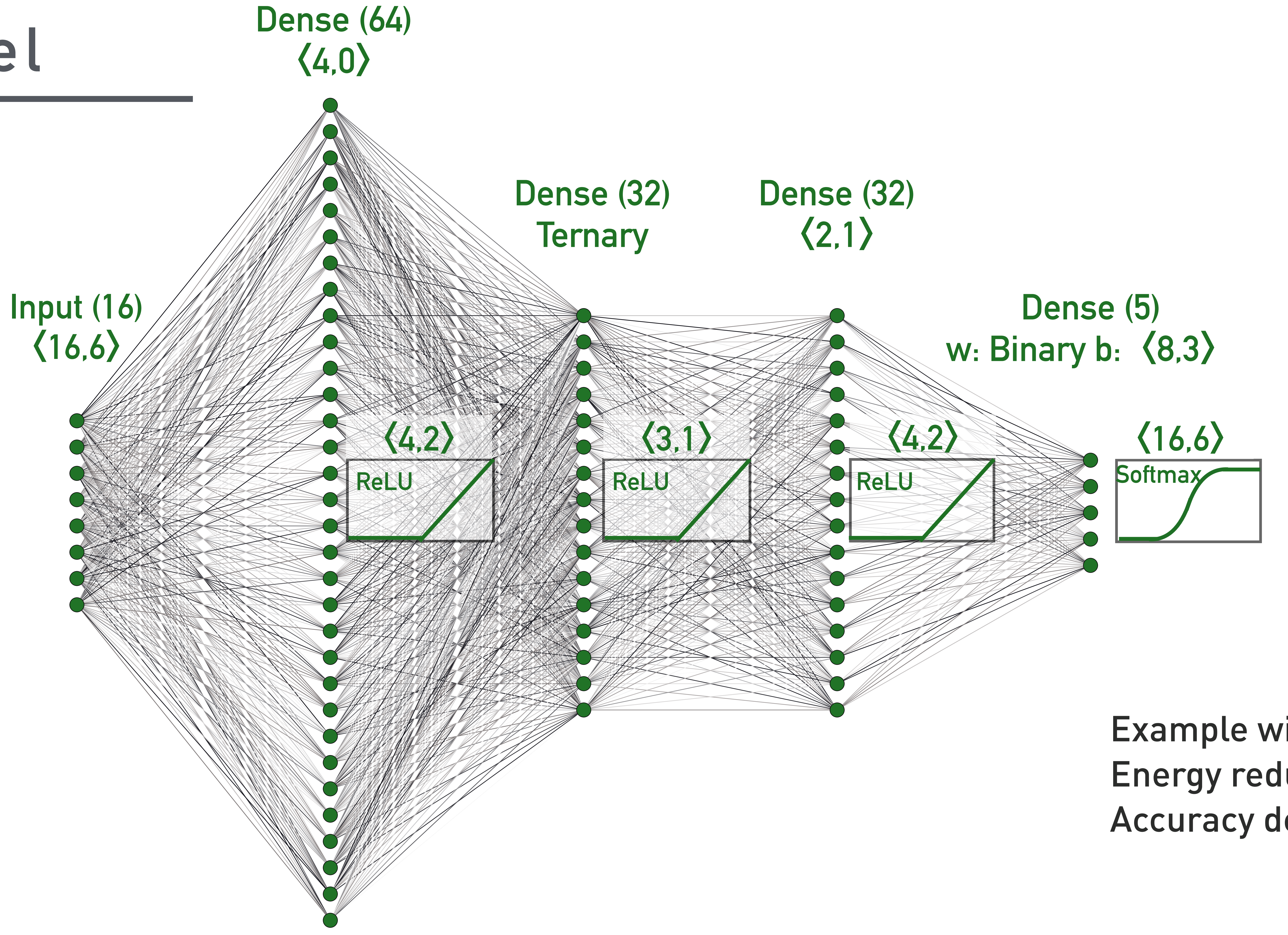
AutoQKeras

AutoQ Bayesian optimization at work!

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



AutoQ model

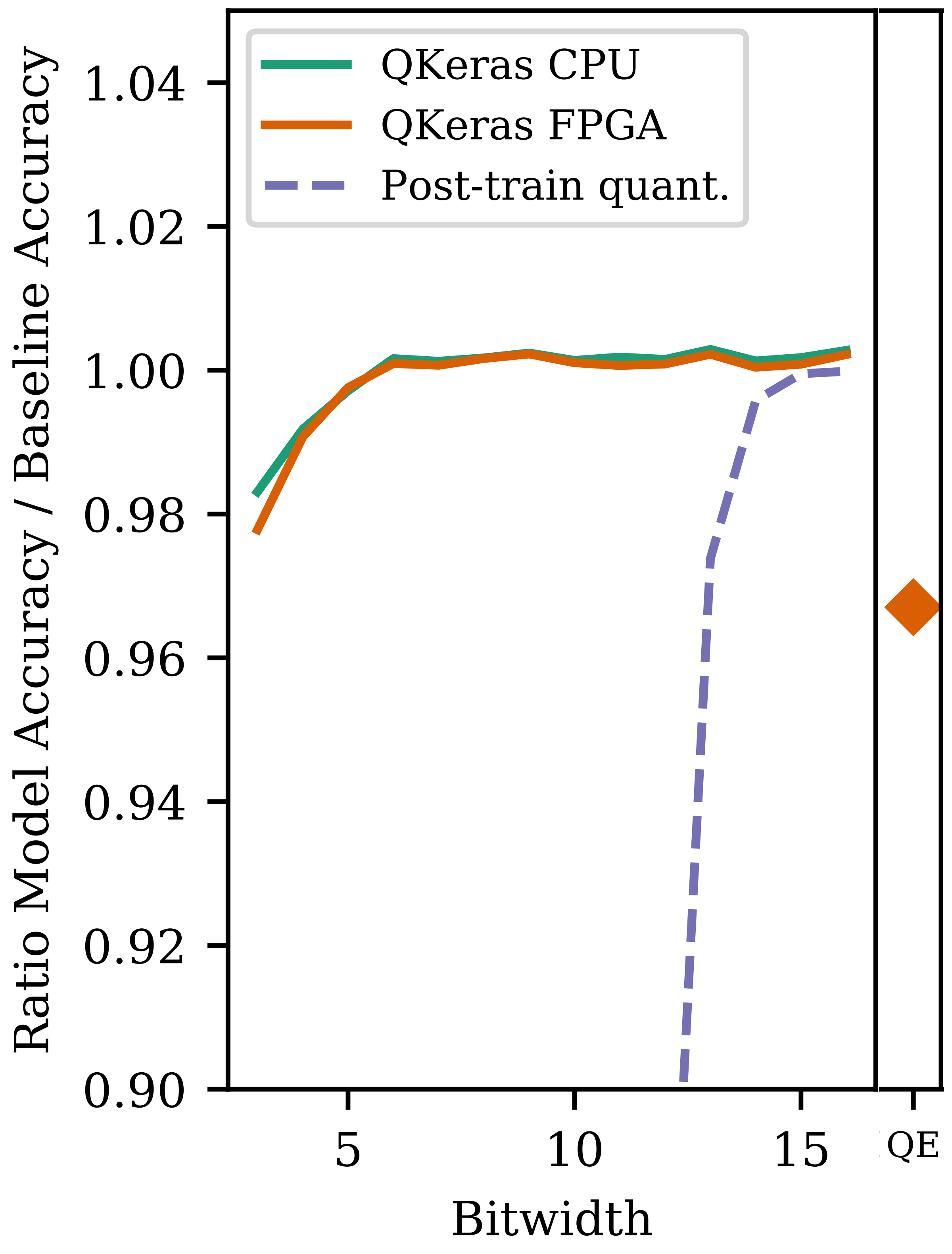


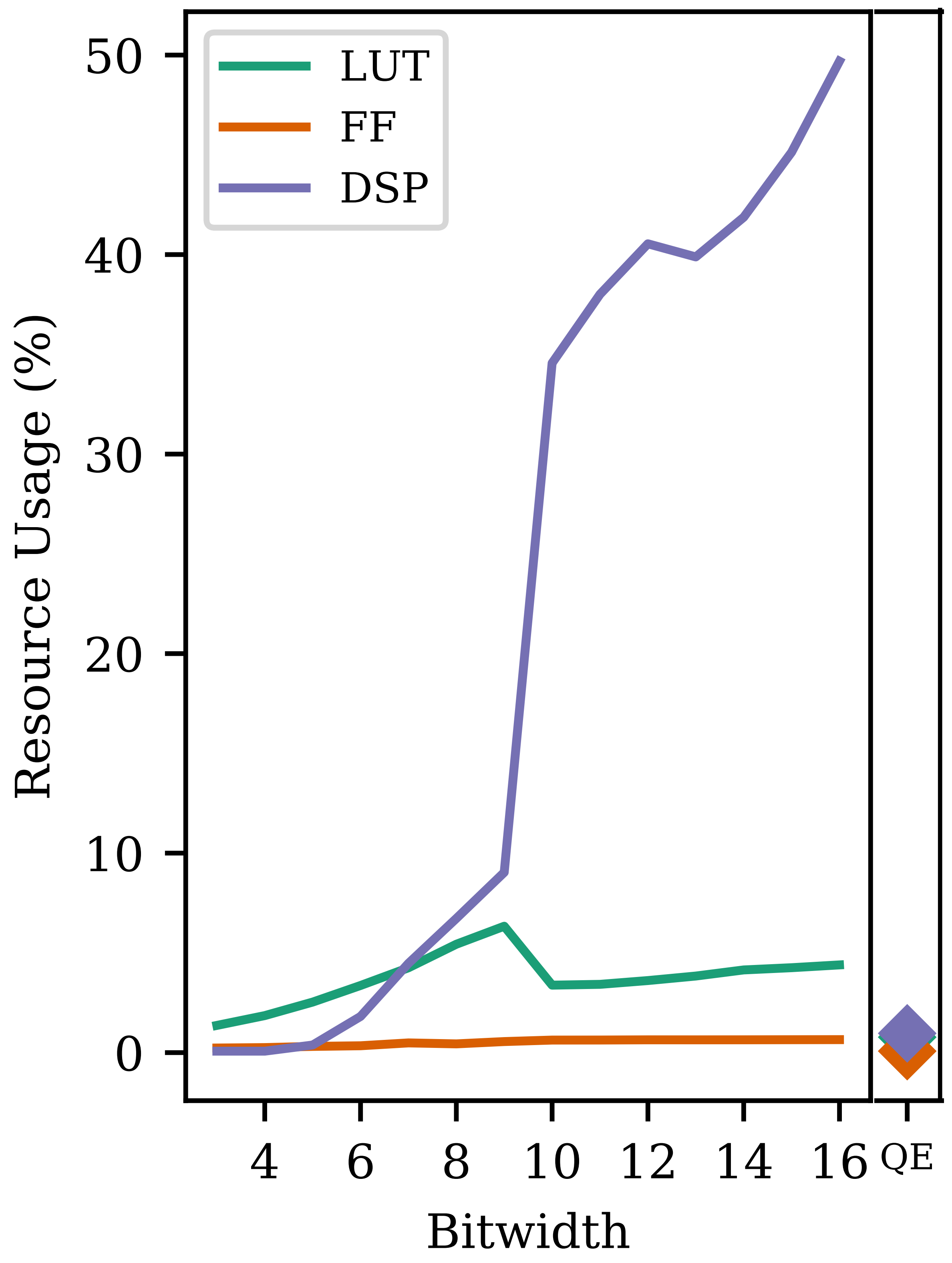
Example with target:
Energy reduction x4
Accuracy degradation max 5%

Model	Acc. [%]	Precision								Tot. energy [μJ]	Tot. bits
		Dense	ReLU	Dense	ReLU	Dense	ReLU	Dense	Softmax		
QE	72.3	$\langle 4, 0\rangle$	$\langle 4, 2\rangle$	Ternary	$\langle 3, 1\rangle$	$\langle 2, 1\rangle$	$\langle 4, 2\rangle$	w: Stoc. Bin. b: $\langle 8, 3\rangle$	$\langle 16, 6\rangle$	0.00095	4728

FPGA performance

Multiplications move to LUTs at bit width <10.



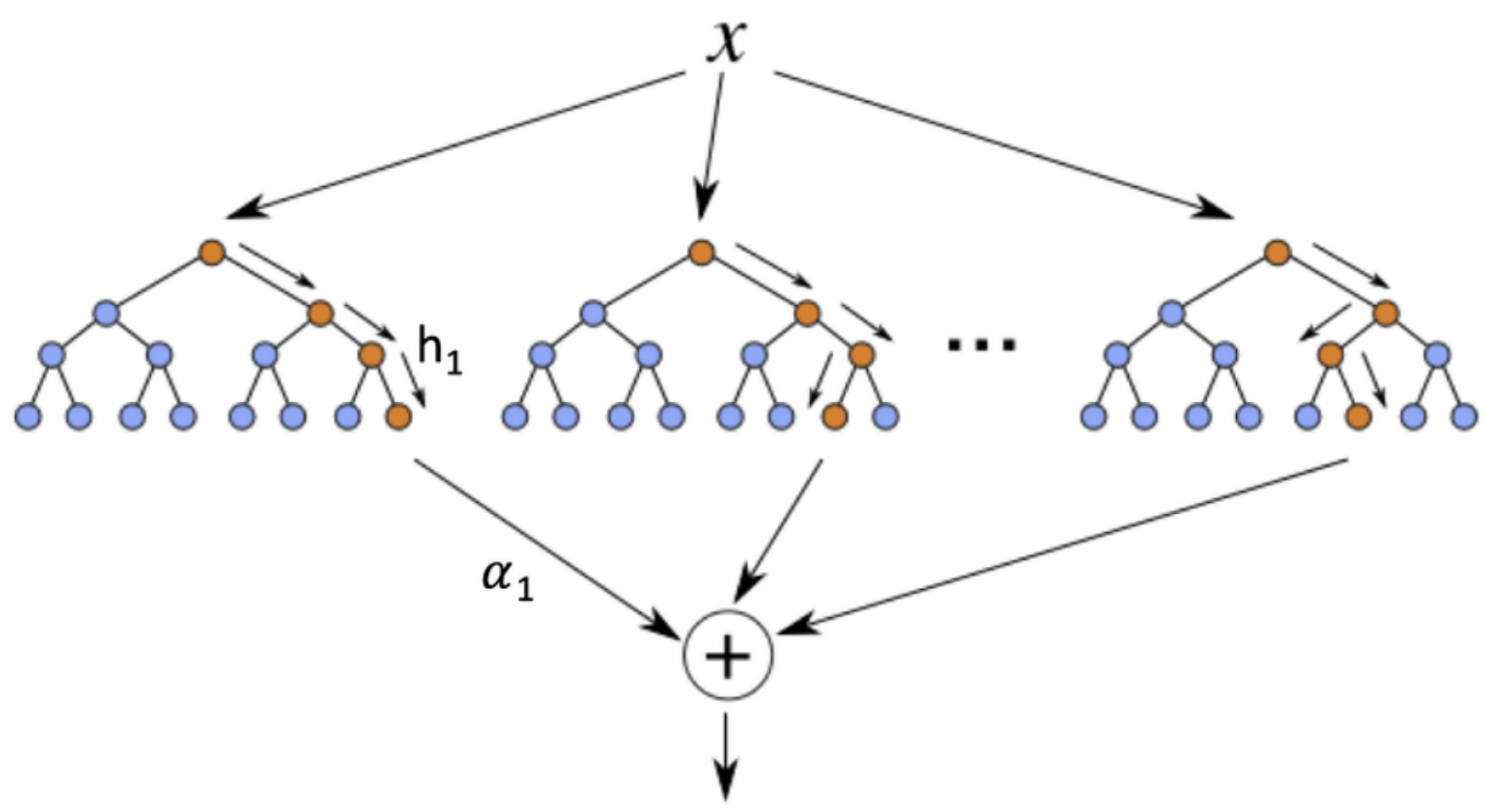


Model	Accuracy [%]	Latency [ns]	Latency [clock cycles]	DSP [%]	LUT [%]	FF [%]
BF	74.4	45	9	56.0 (1826)	5.2 (48321)	0.8 (20132)
Q6	74.8	55	11	1.8 (124)	3.4 (39782)	0.3 (8128)
QE	72.3	55	11	**1.0 (66)**	**0.8 (9149)**	0.1 (1781)

Same as hls4ml but for Boosted decision trees (scikit-learn, XGBoost)

If resource/latency constainted, 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%

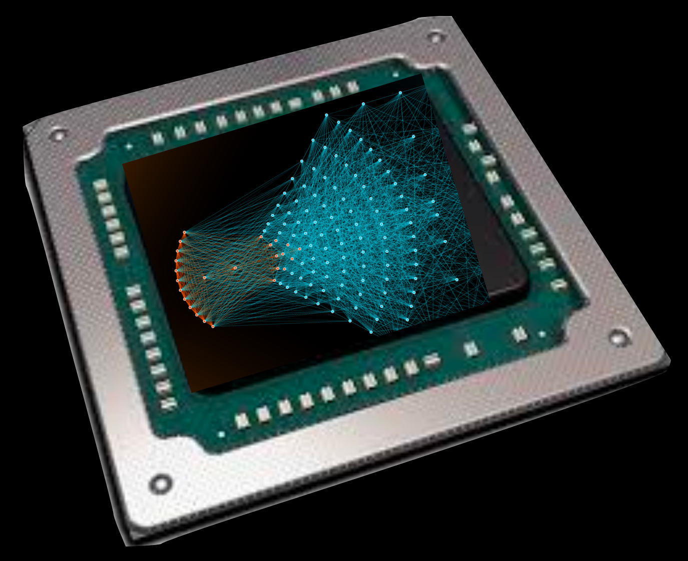

hls 4 ml
Conifer
ATLAS
EXPERIMENT
HL-LHC tt̄ event in ATLAS ITK
at <μ>=200
Join the community:
fastmachinelearning.org

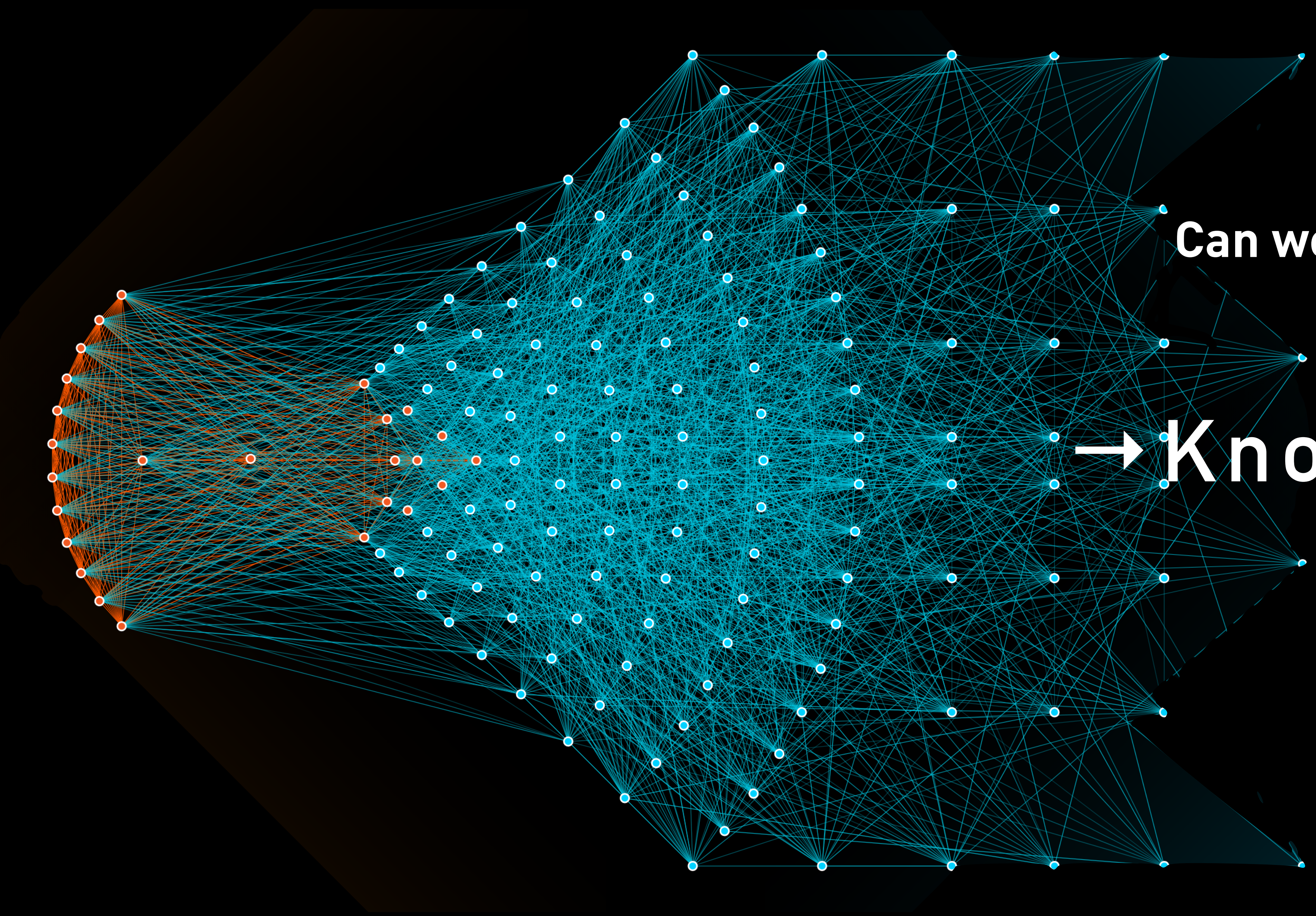
Backup


We have infinite time and resources to train huge networks

We have infinite time and resources to train huge networks

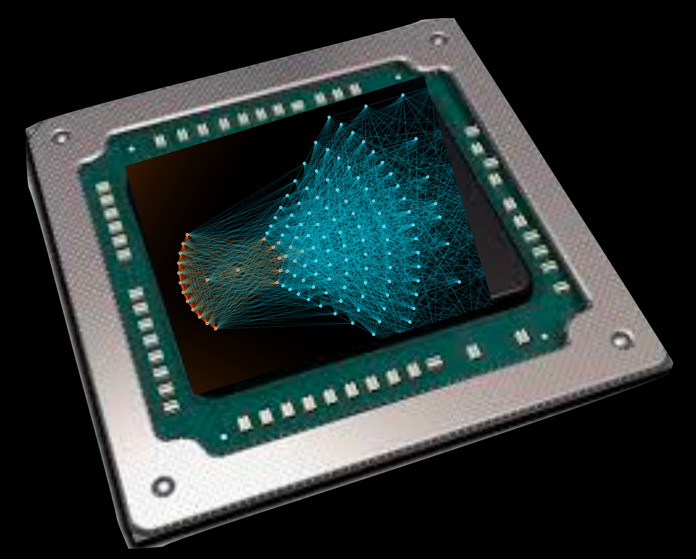
but very little for inference





Can we have the best of both worlds?

→Knowledge Distillation

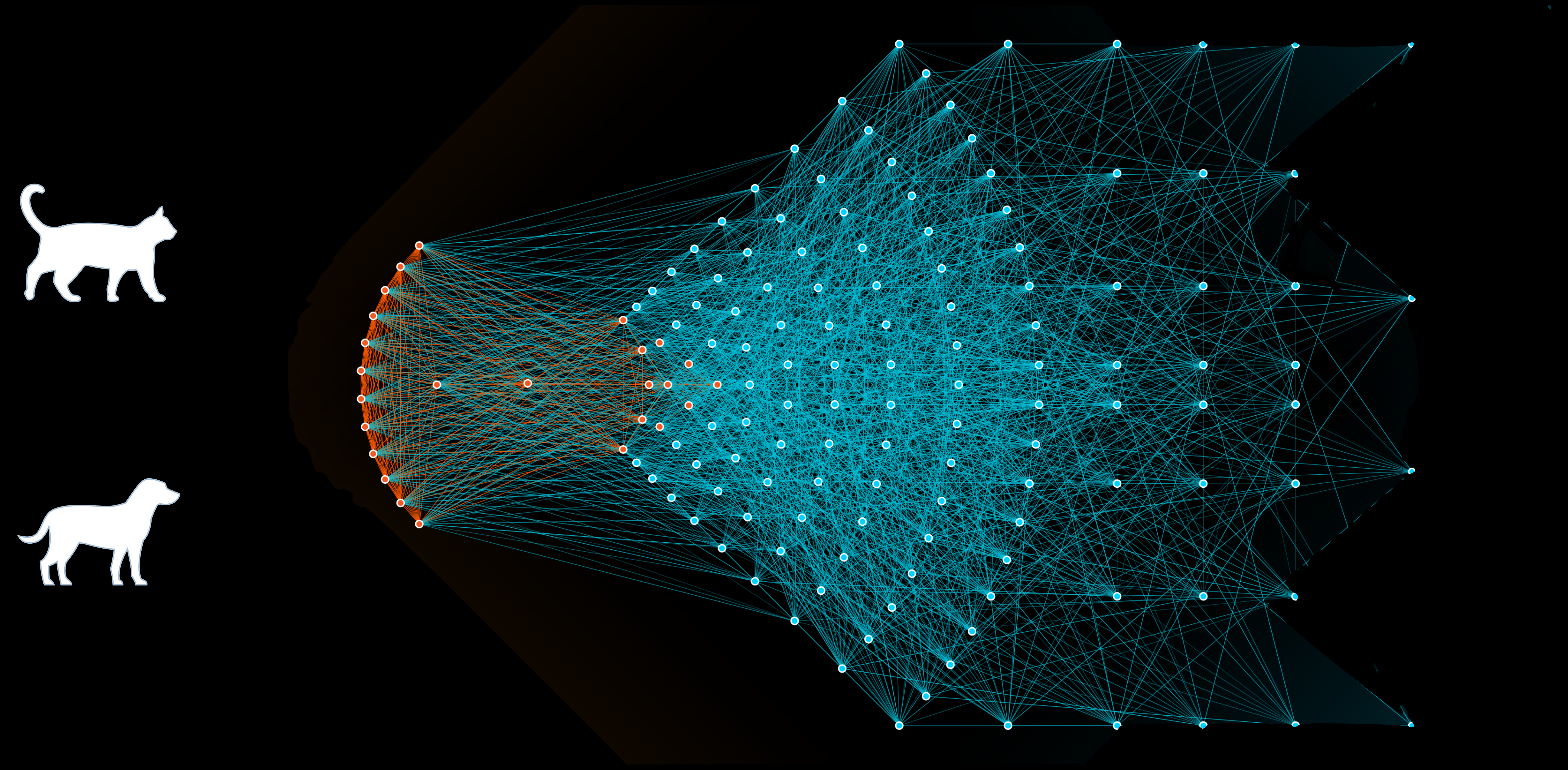


Train

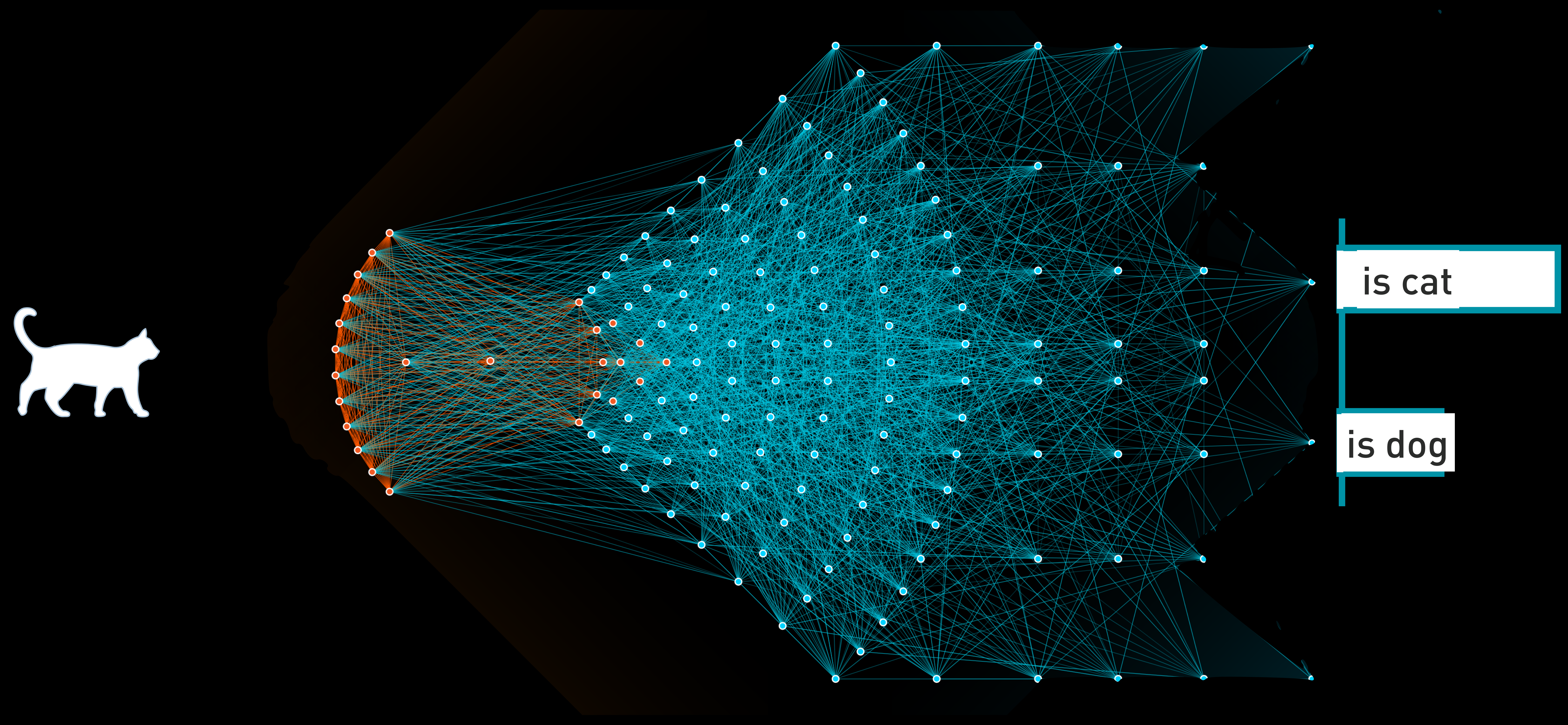
Inference

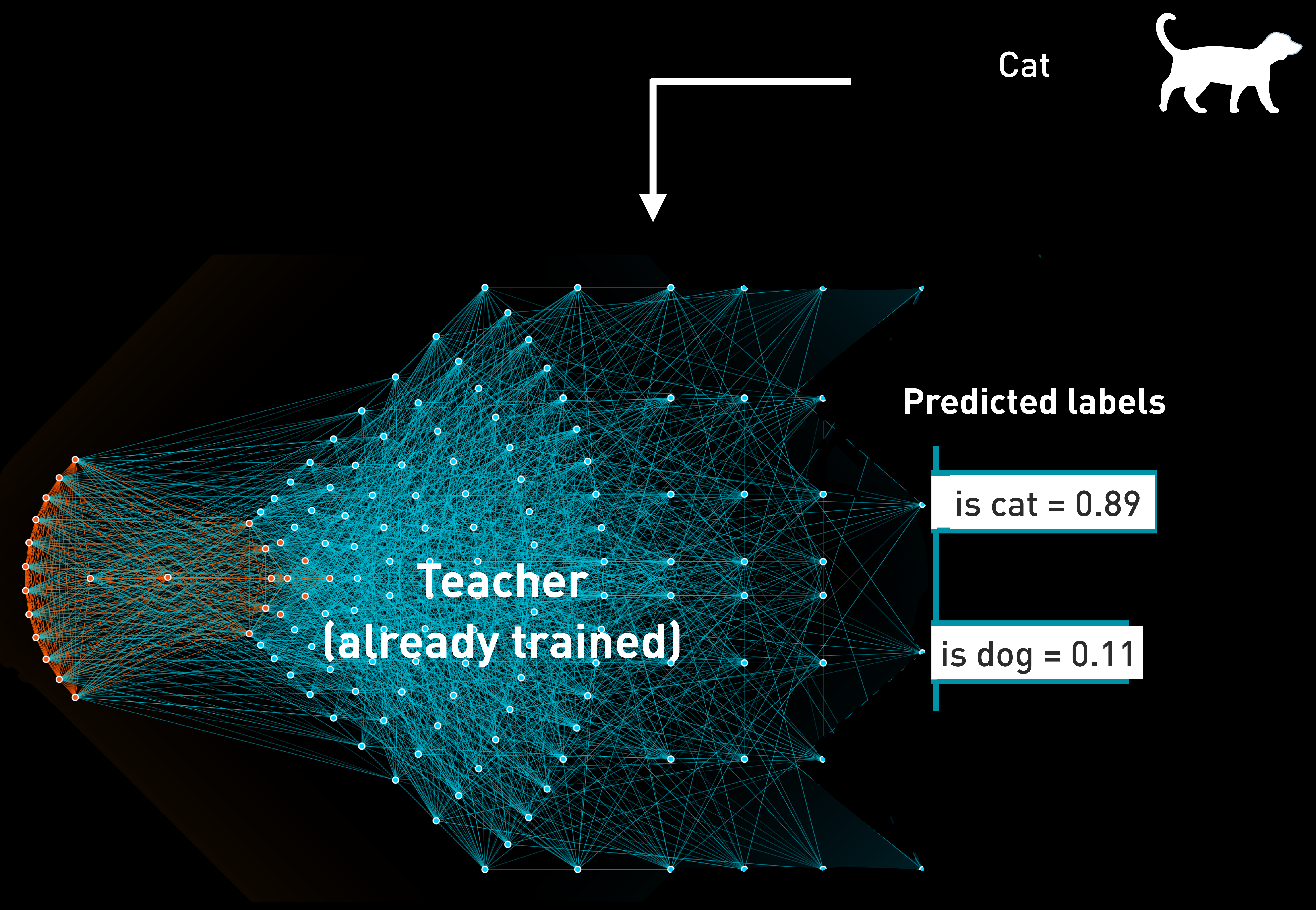
Cat

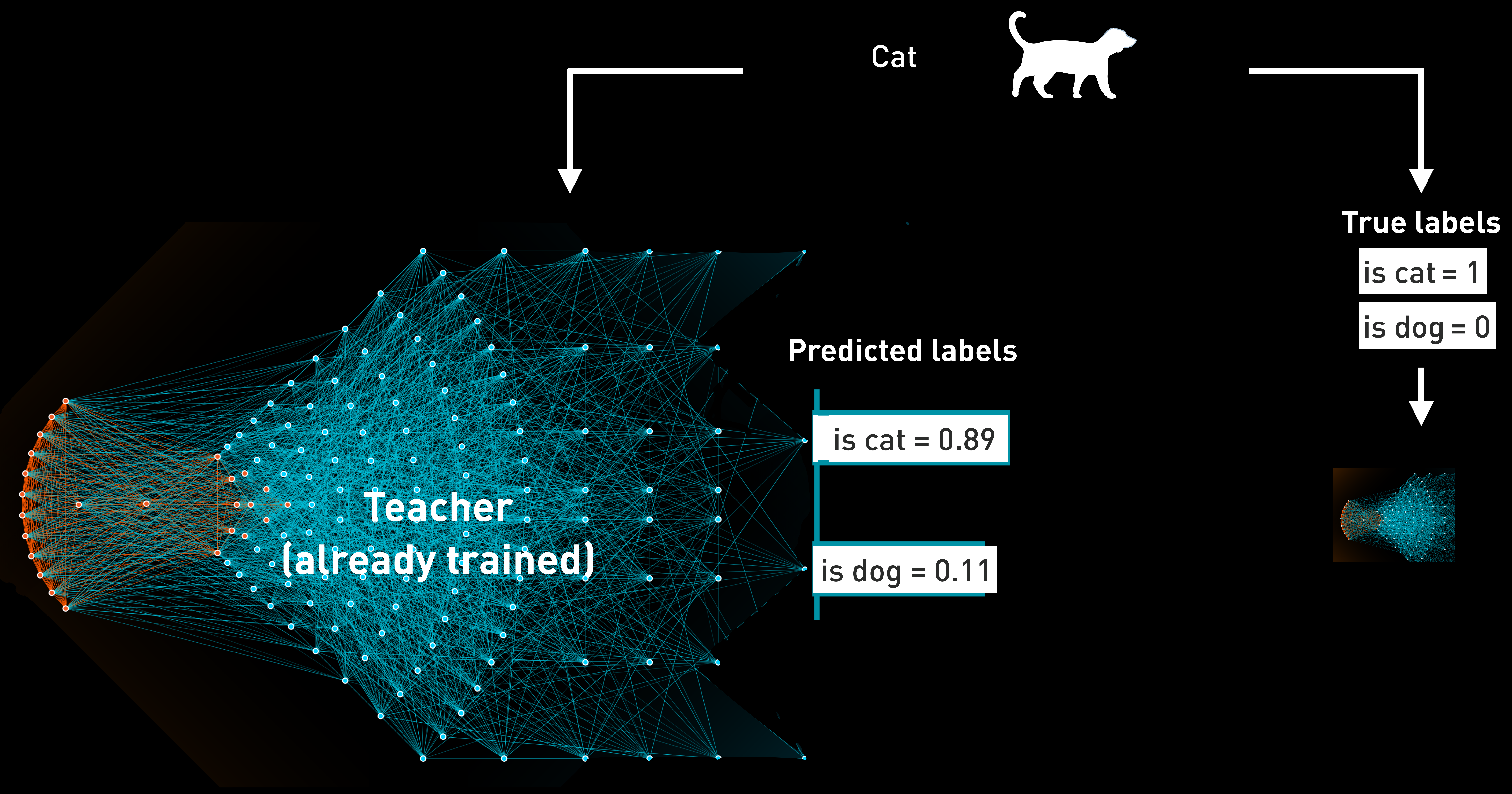
Dog

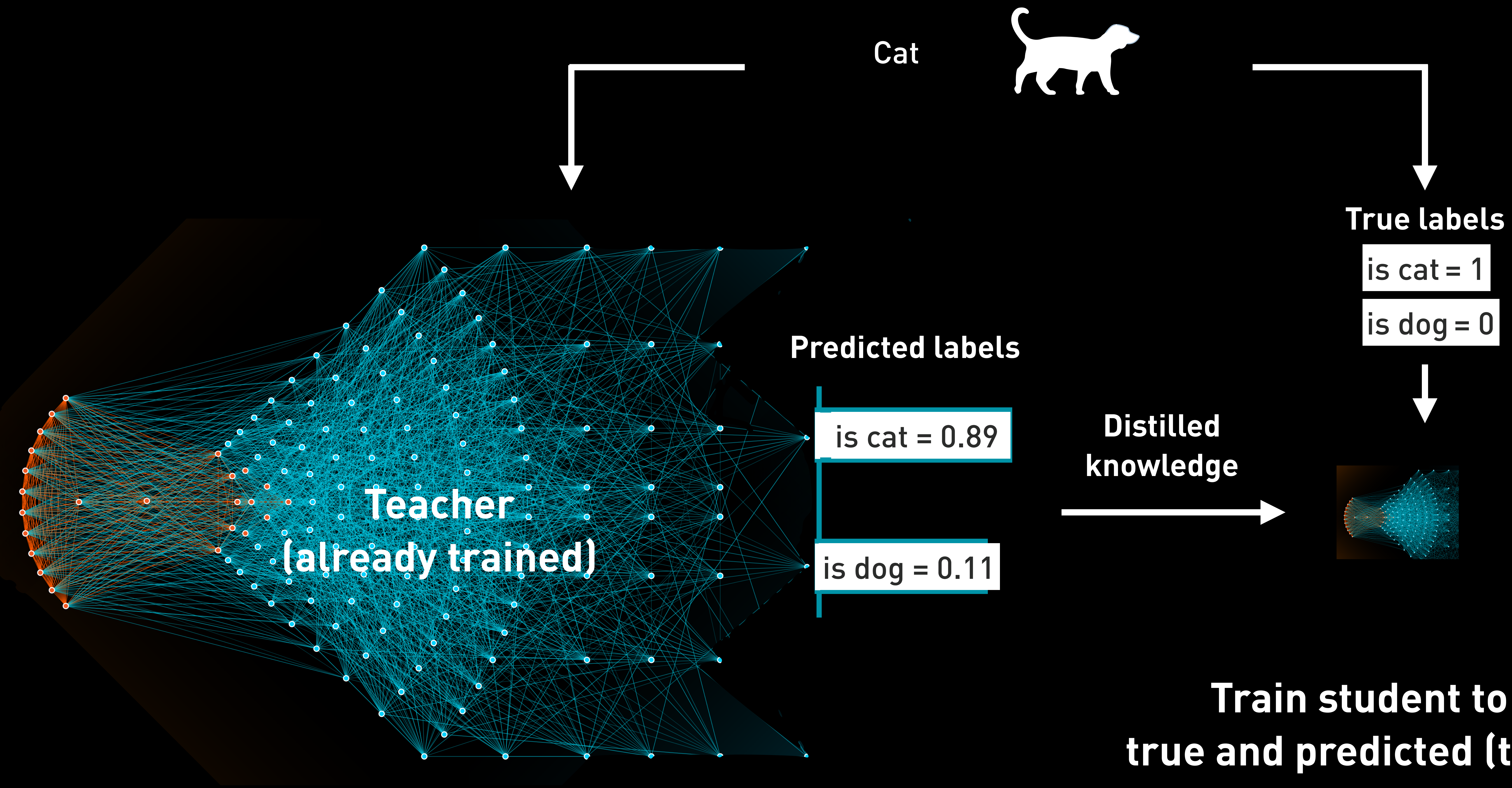


Cat




Cat
Teacher
(already trained)
Predicted labels
is cat = 0.89
is dog = 0.11


Cat
True labels
is cat = 1
is dog = 0
Predicted labels
is cat = 0.89
is dog = 0.11
Teacher
(already trained)



Train student to learn both true and predicted (teacher) labels!

$$L_{total} = \beta \times L_{Distillation} + \alpha \times L_{student}$$

Student learns subtle learned features from teacher!