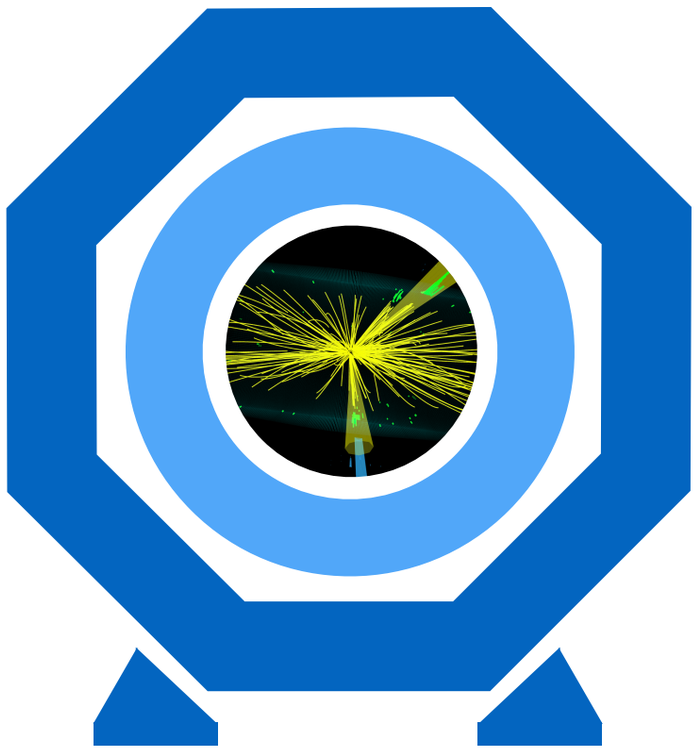
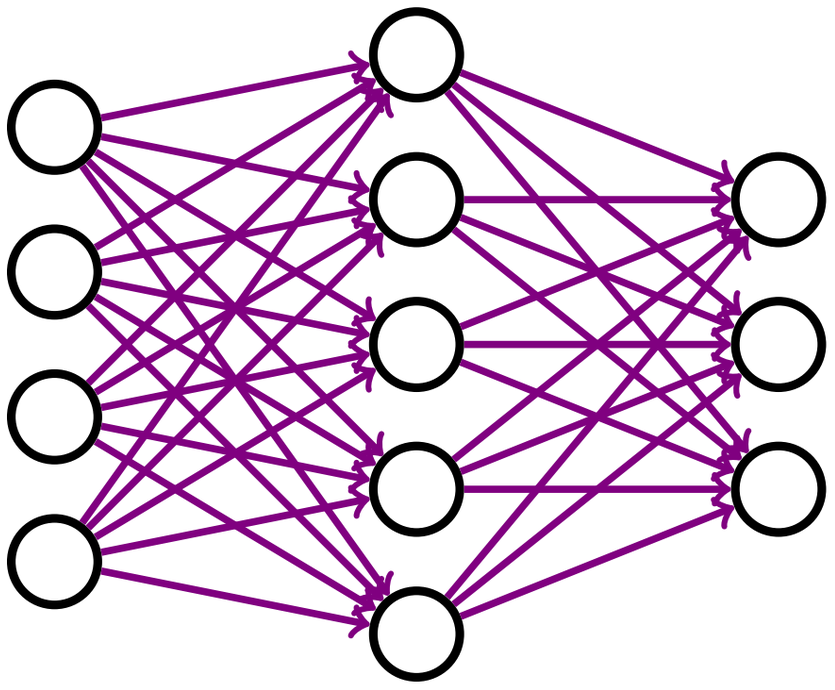
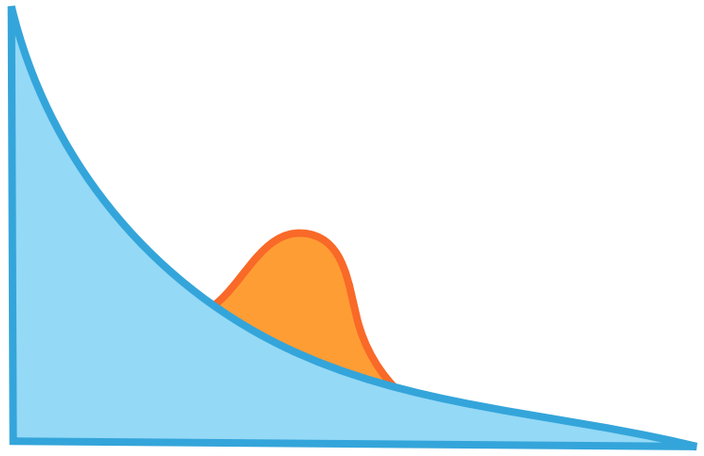






JAVIER DUARTE

JULY 16, 2019

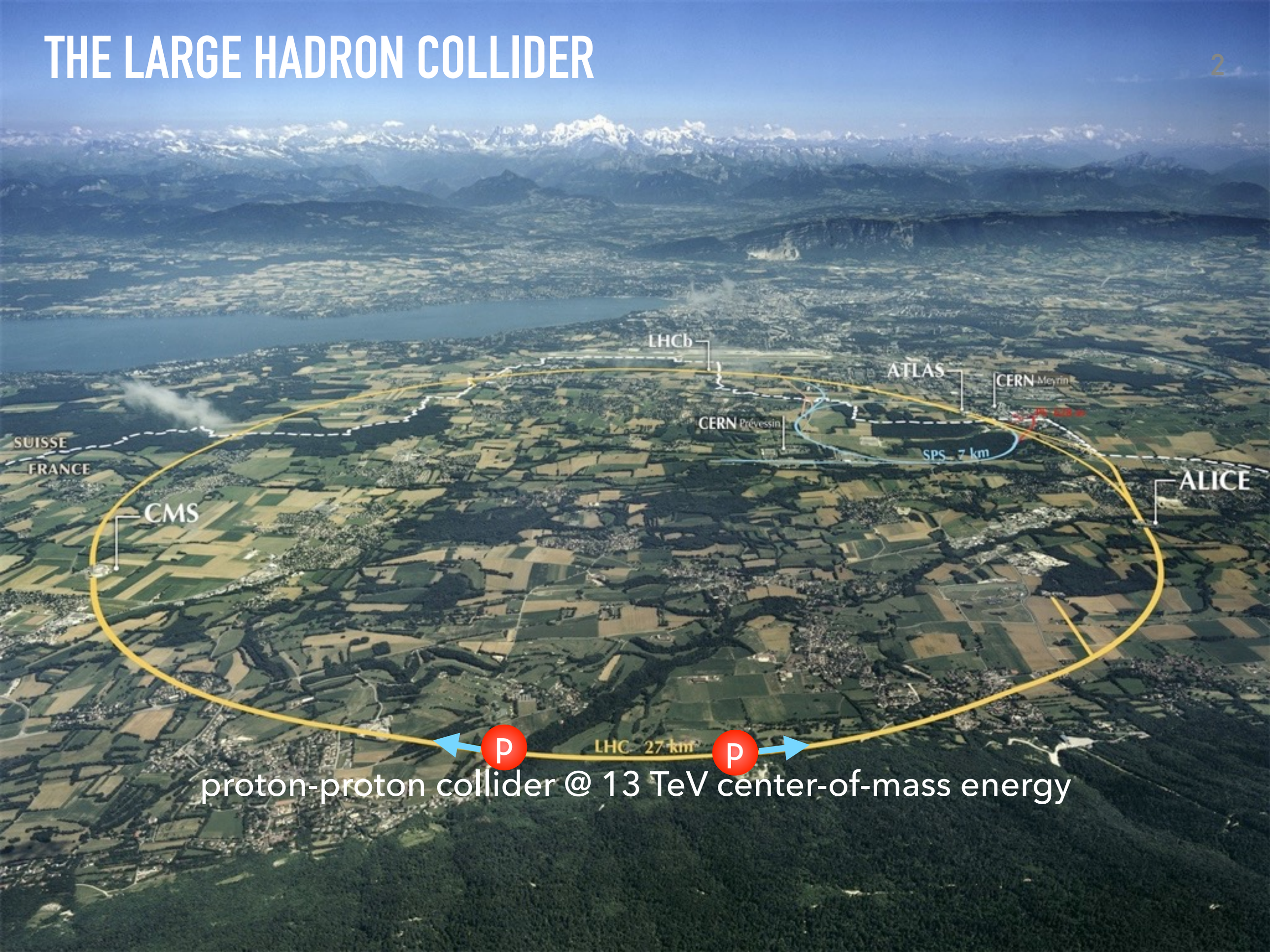
INSTITUT PASCAL

FAST INFERENCE OF DEEP NEURAL NETWORKS IN FPGAS FOR PARTICLE PHYSICS

(arXiv:1804.06913)

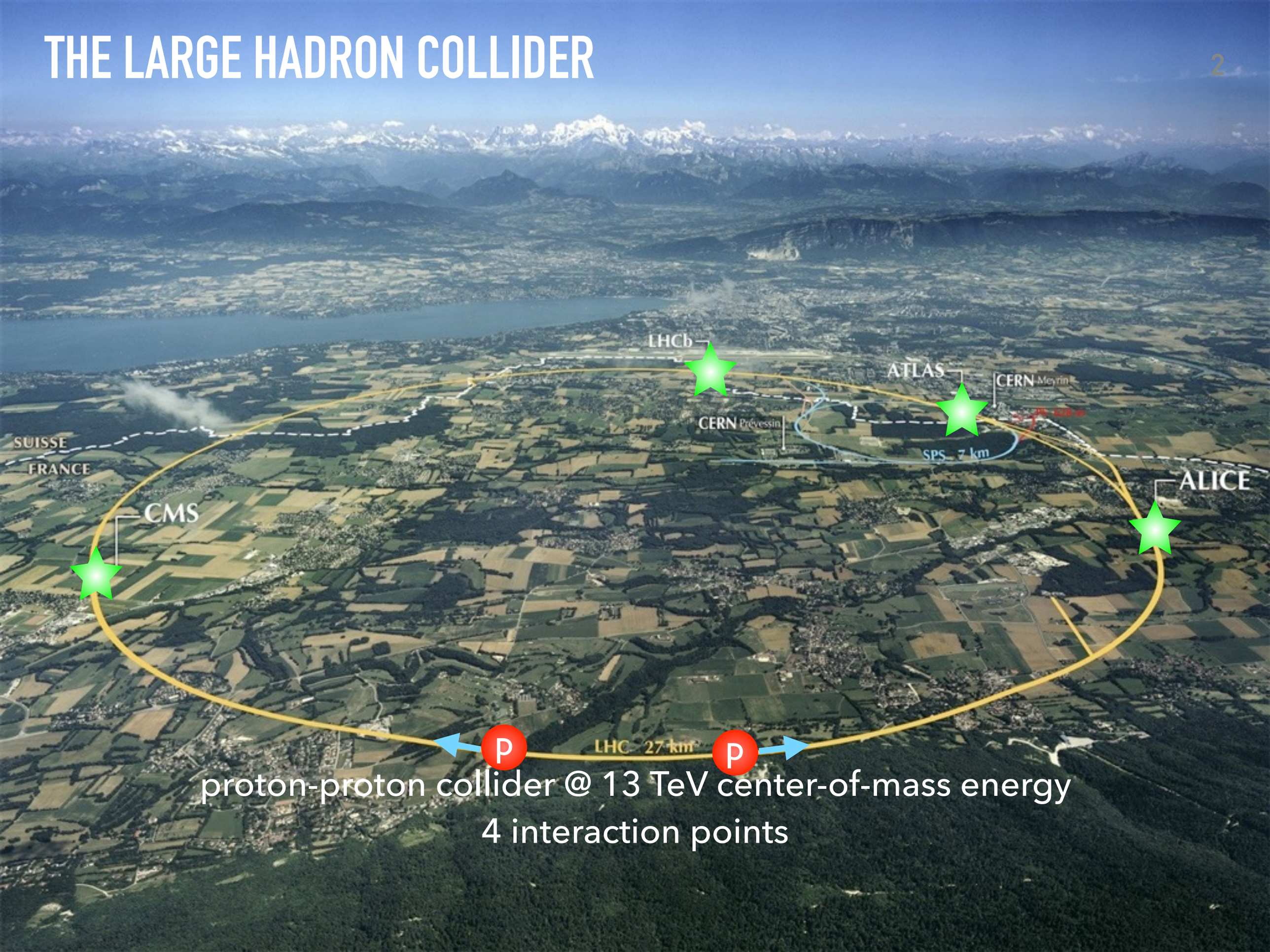
THE LARGE HADRON COLLIDER

THE LARGE HADRON COLLIDER



proton-proton collider @ 13 TeV center-of-mass energy

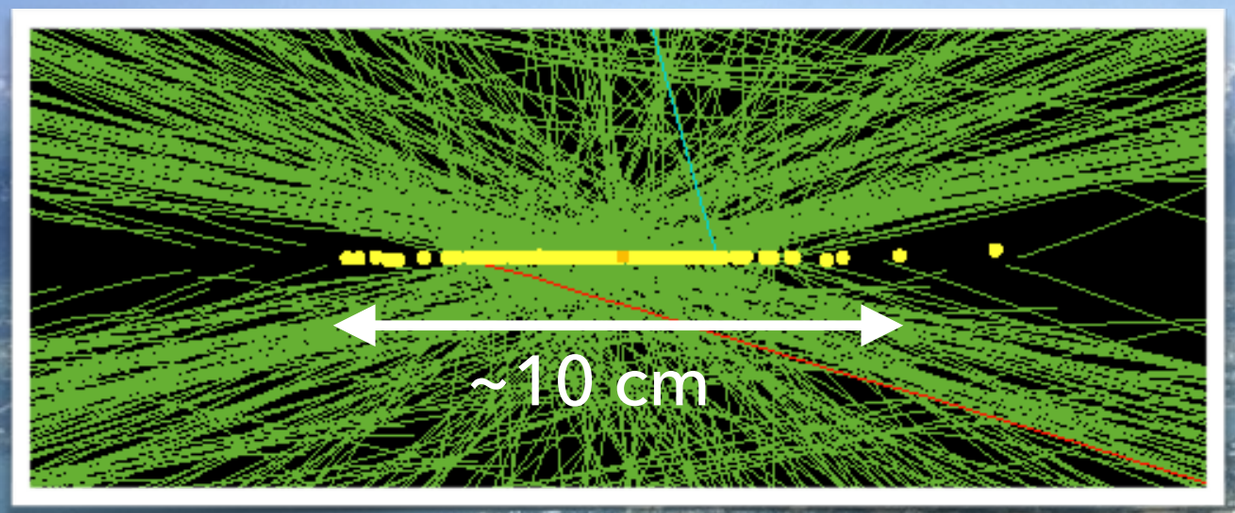
THE LARGE HADRON COLLIDER



proton-proton collider @ 13 TeV center-of-mass energy

4 interaction points

THE LARGE HADRON COLLIDER



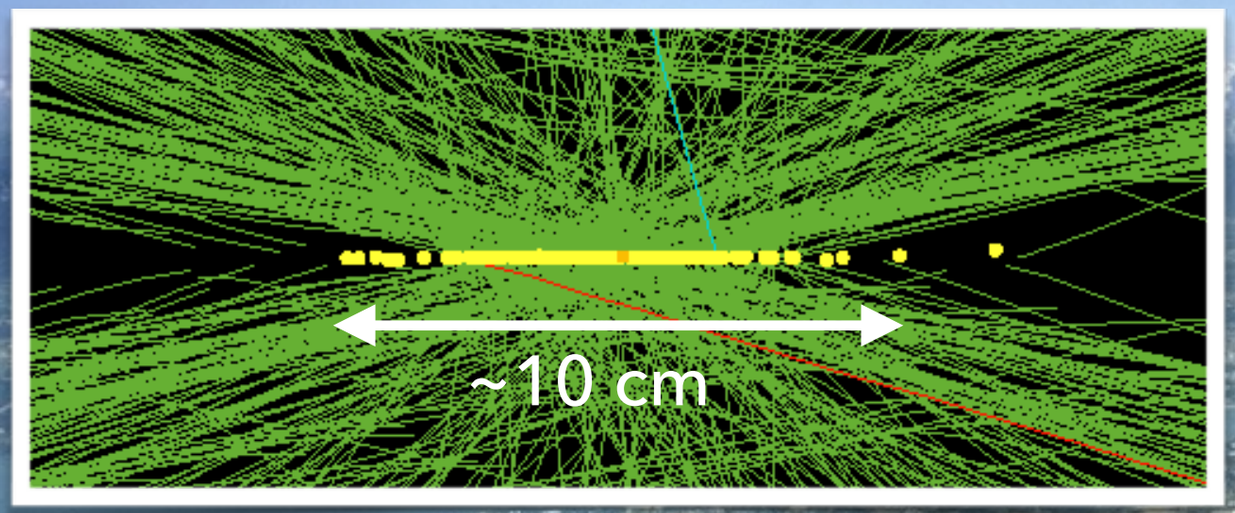
LHCb
ATLAS
CERN Meyrin
CERN Prévessin
SPS 7 km
SUISSE
FRANCE
CMS
ALICE
P
LHC 27 km
P

proton-proton collider @ 13 TeV center-of-mass energy

4 interaction points

40 million collisions / second

THE LARGE HADRON COLLIDER



LHCb
ATLAS
CERN Meyrin
CERN Prévessin
SPS 7 km
SUISSE
FRANCE
CMS
ALICE
P
LHC 27 km
P

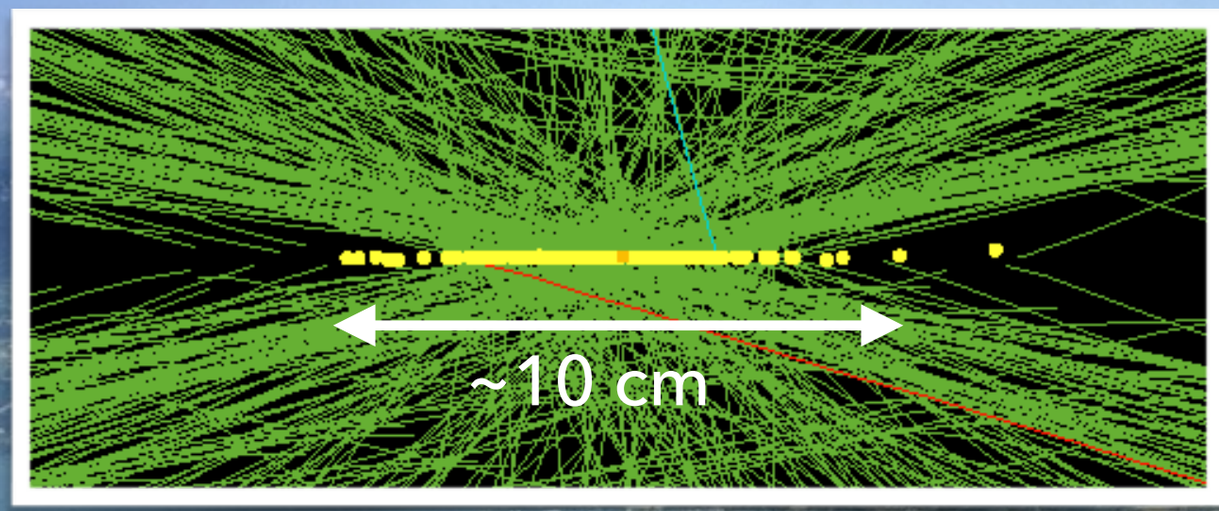
proton-proton collider @ 13 TeV center-of-mass energy

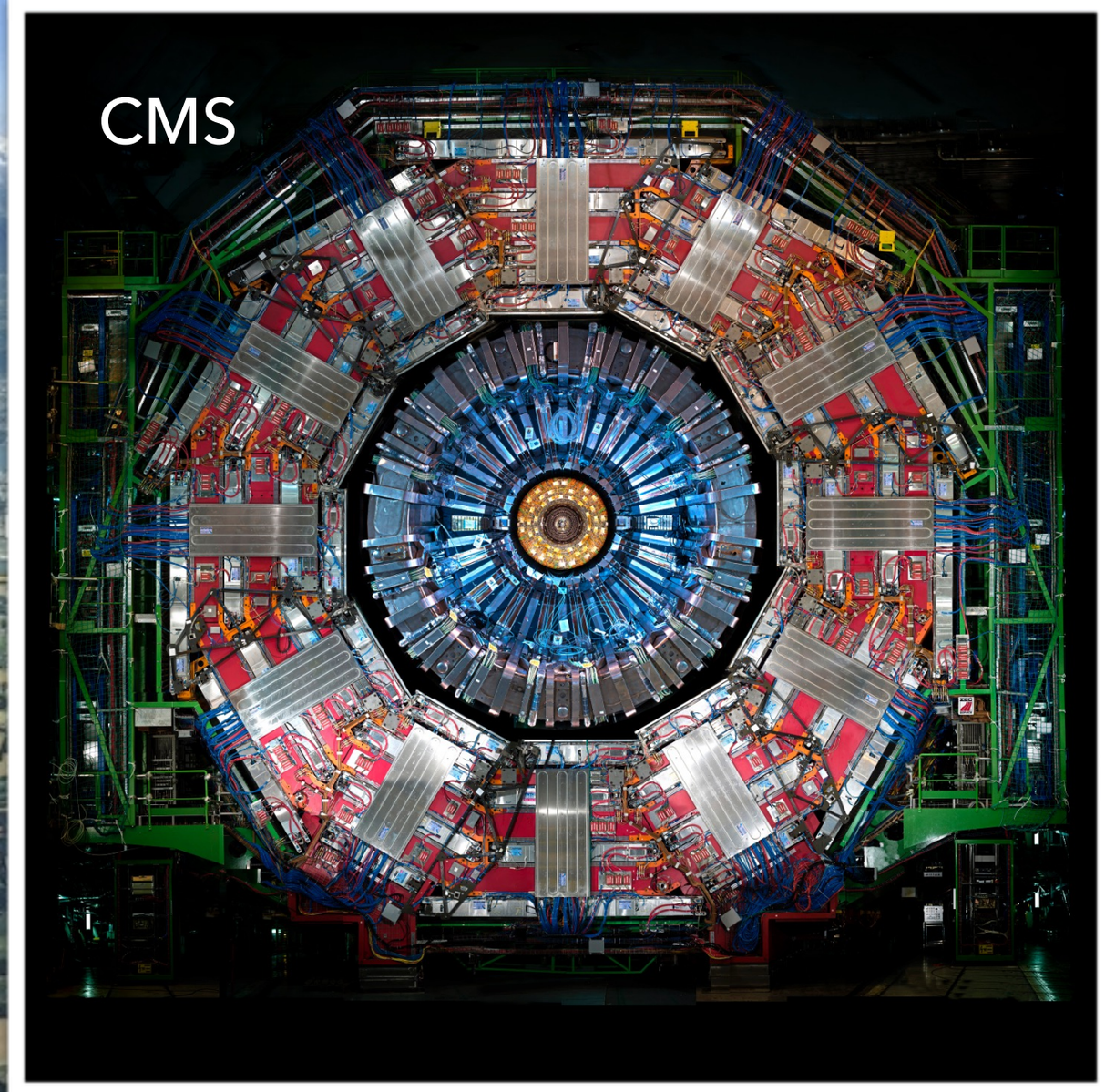
4 interaction points

40 million collisions / second

trigger selects ~1000 collisions / second

THE LARGE HADRON COLLIDER





SUISSE
FRANCE
CMS
P
LHC 27 km
P

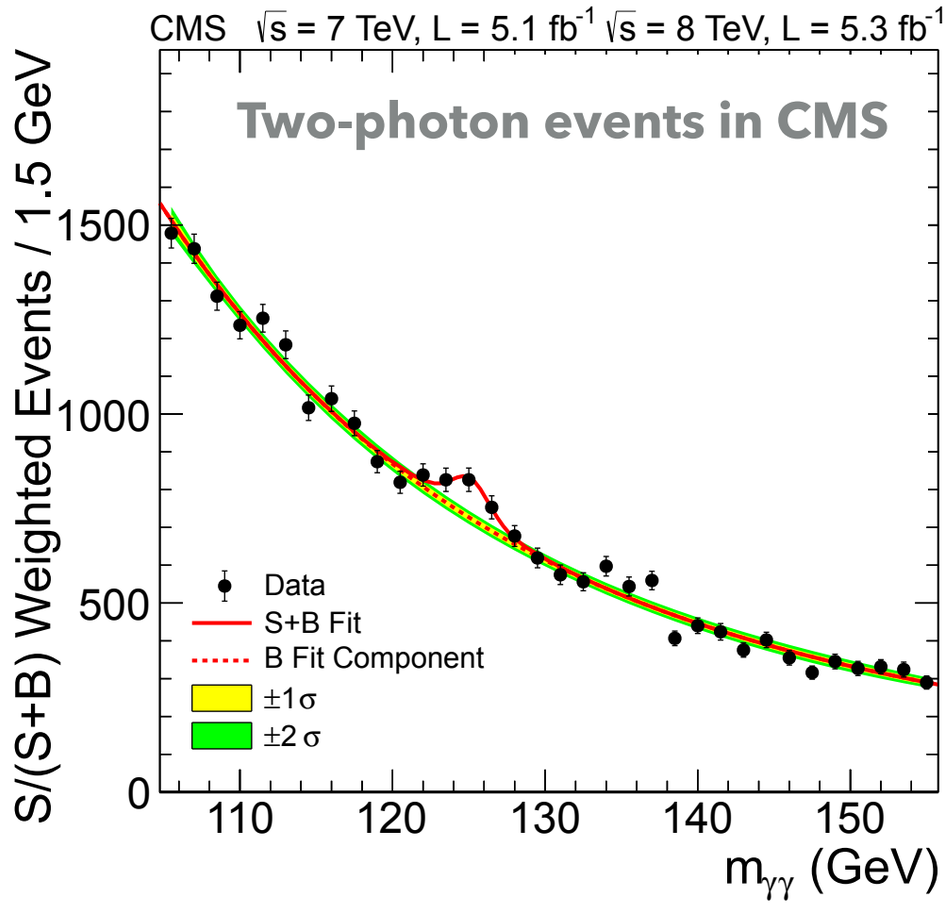
proton-proton collider @ 13 TeV center-of-mass energy

4 interaction points

40 million collisions / second

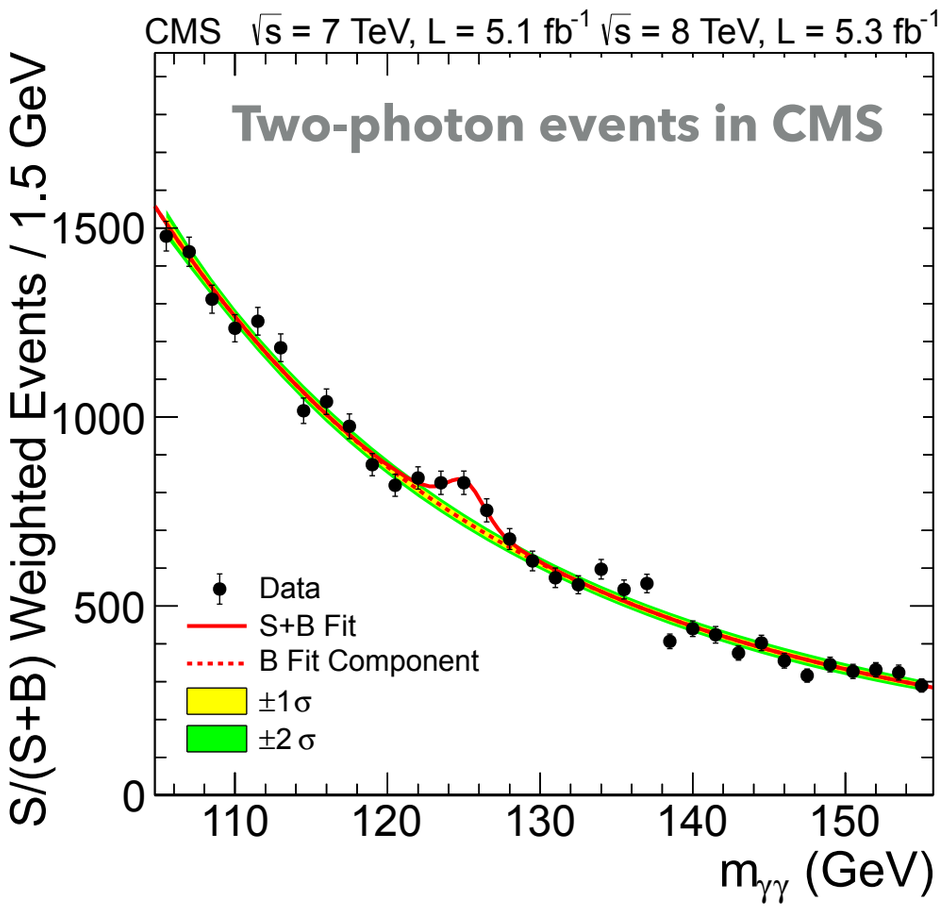
trigger selects ~1000 collisions / second

HIGH ENERGY PHYSICS + MACHINE LEARNING



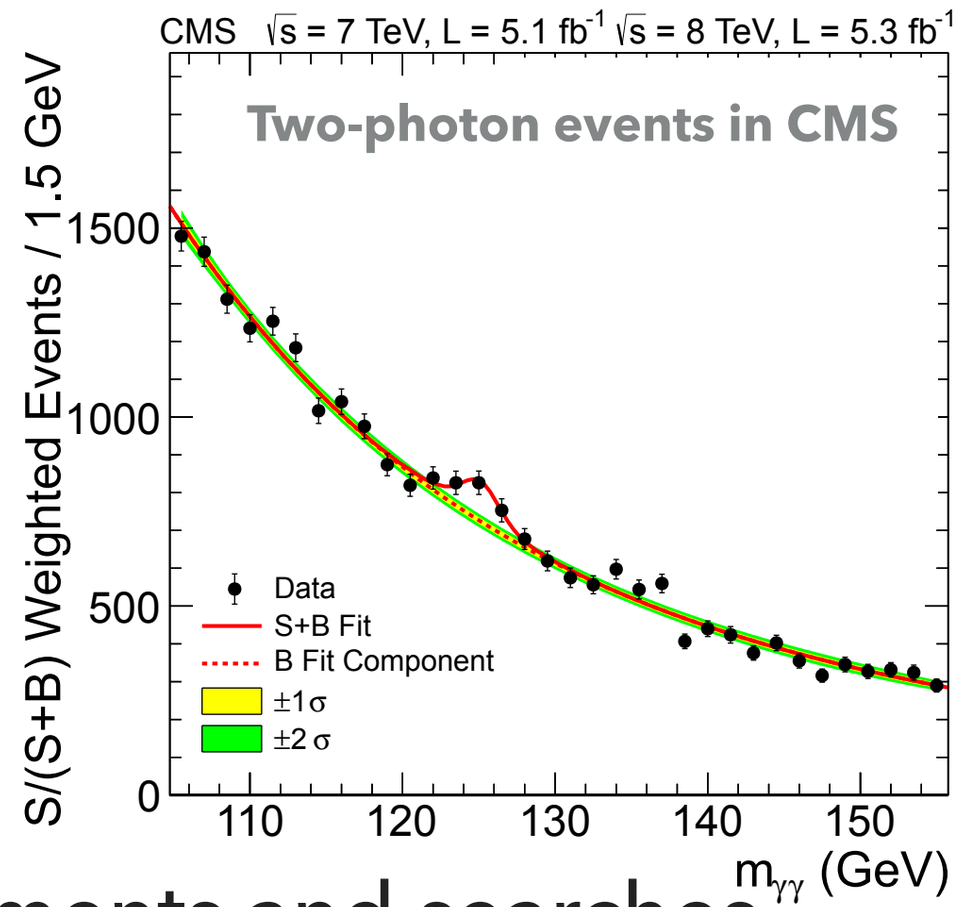
HIGH ENERGY PHYSICS + MACHINE LEARNING

- **Machine learning** was vital to make big discoveries like the Higgs boson on July 4, 2012



HIGH ENERGY PHYSICS + MACHINE LEARNING

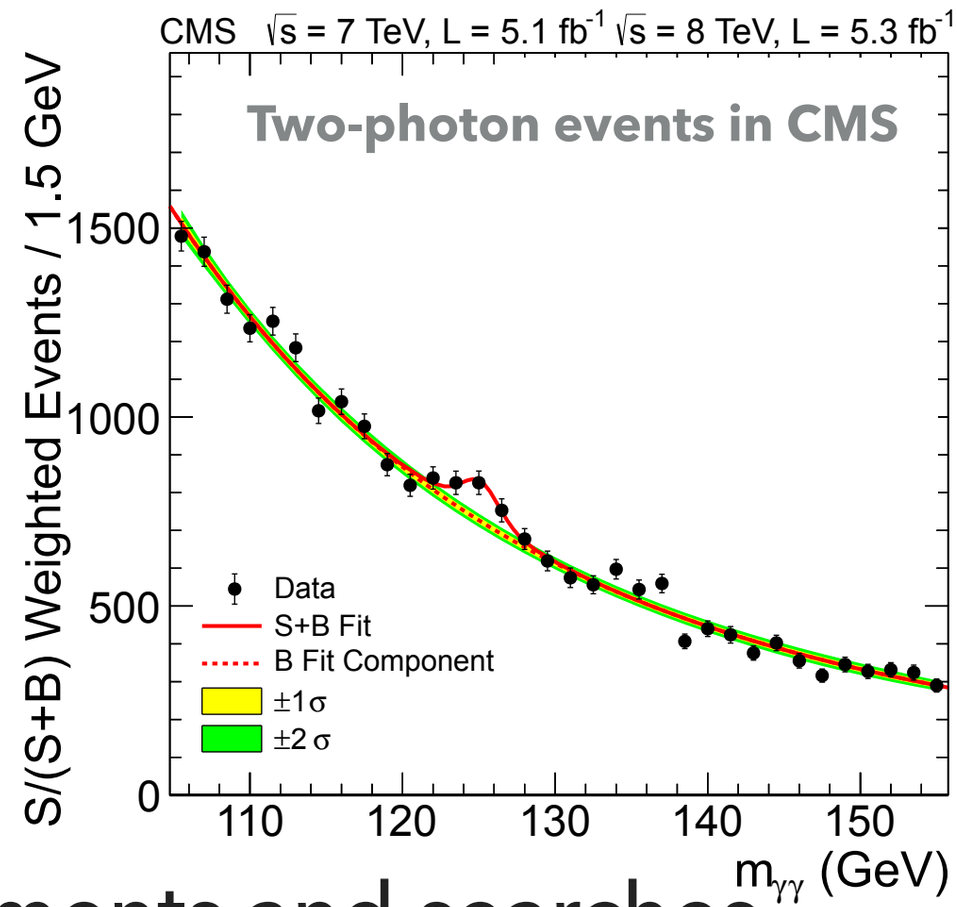
- **Machine learning** was vital to make big discoveries like the Higgs boson on July 4, 2012



- Today, ML is **enabling** new measurements and searches never thought possible at the LHC

HIGH ENERGY PHYSICS + MACHINE LEARNING

- **Machine learning** was vital to make big discoveries like the Higgs boson on July 4, 2012

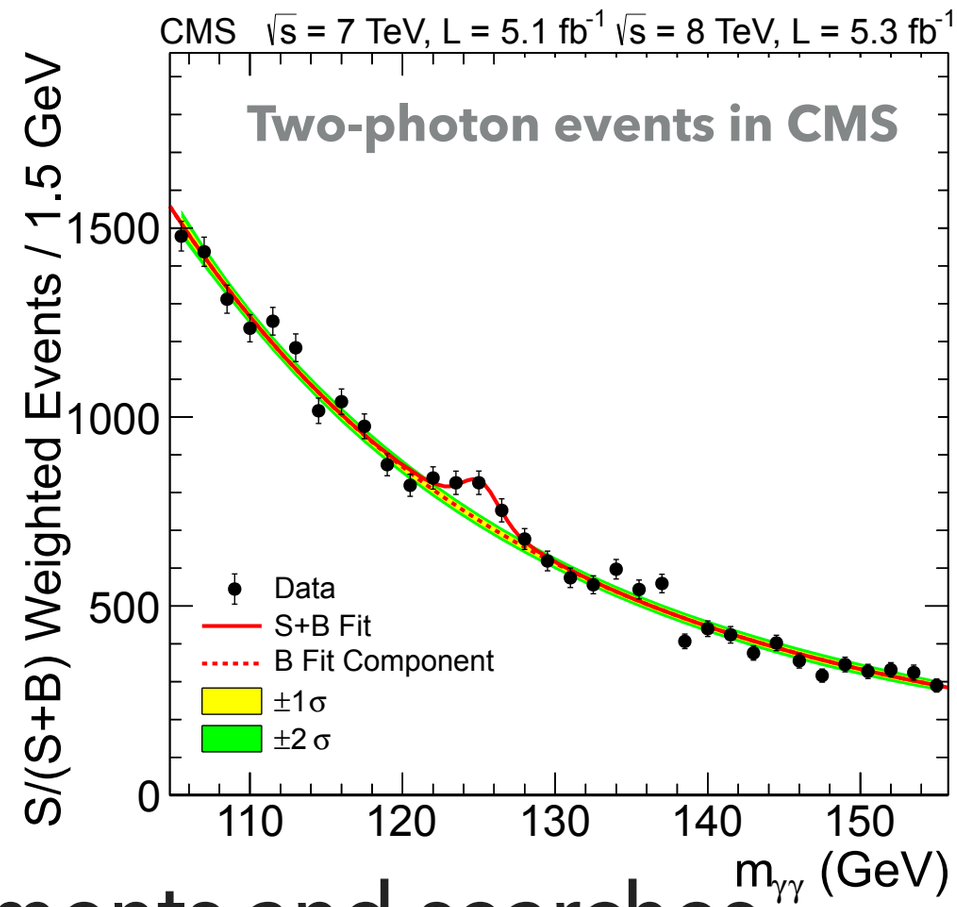


- Today, ML is **enabling** new measurements and searches never thought possible at the LHC

- At the same time, we must **plan** how we will overcome challenges in the **next generation of colliders**

HIGH ENERGY PHYSICS + MACHINE LEARNING

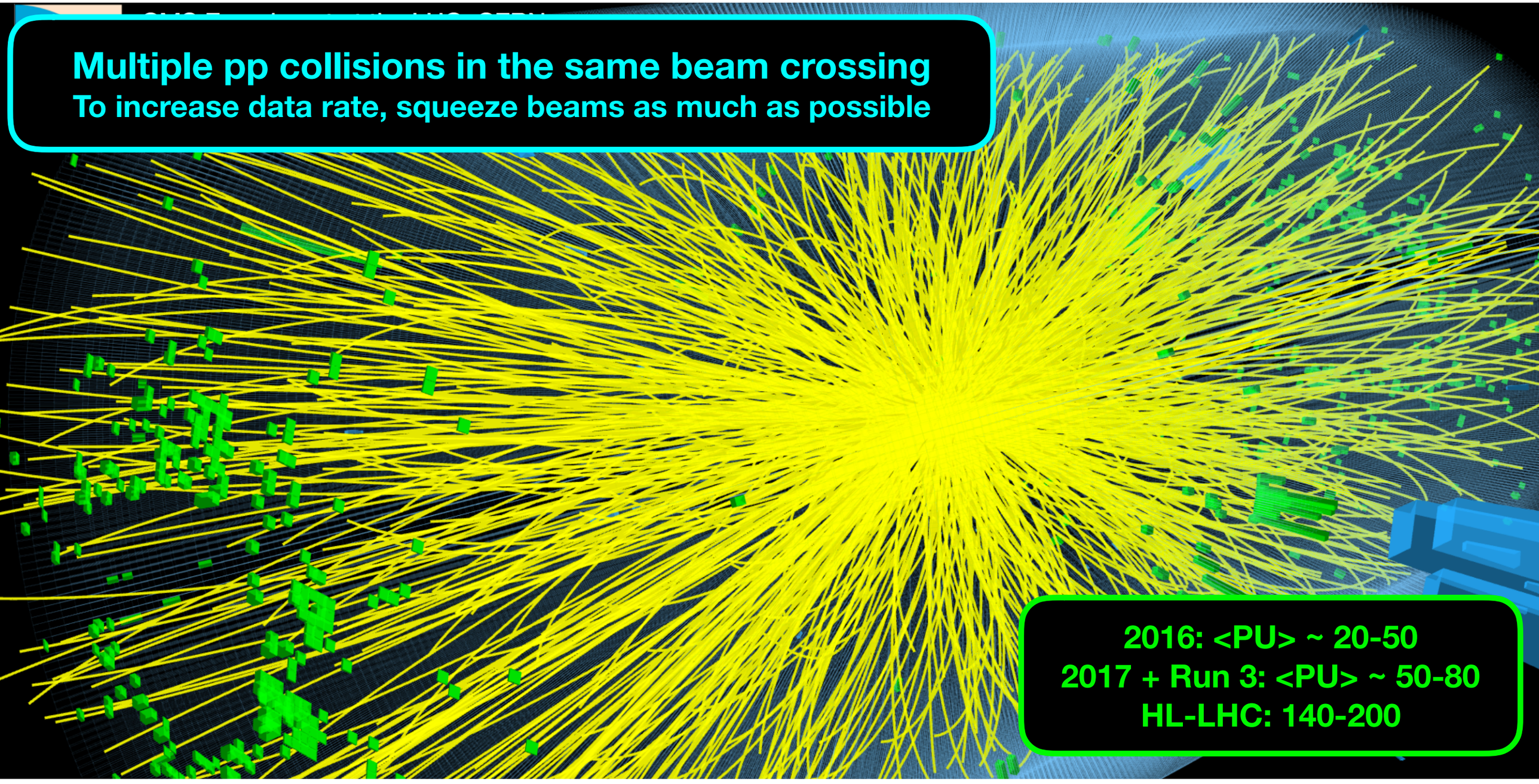
- **Machine learning** was vital to make big discoveries like the Higgs boson on July 4, 2012



- Today, ML is **enabling** new measurements and searches never thought possible at the LHC
- At the same time, we must **plan** how we will overcome challenges in the **next generation of colliders**
- ML may be a way out

CHALLENGE: PILEUP

Multiple pp collisions in the same beam crossing
To increase data rate, squeeze beams as much as possible

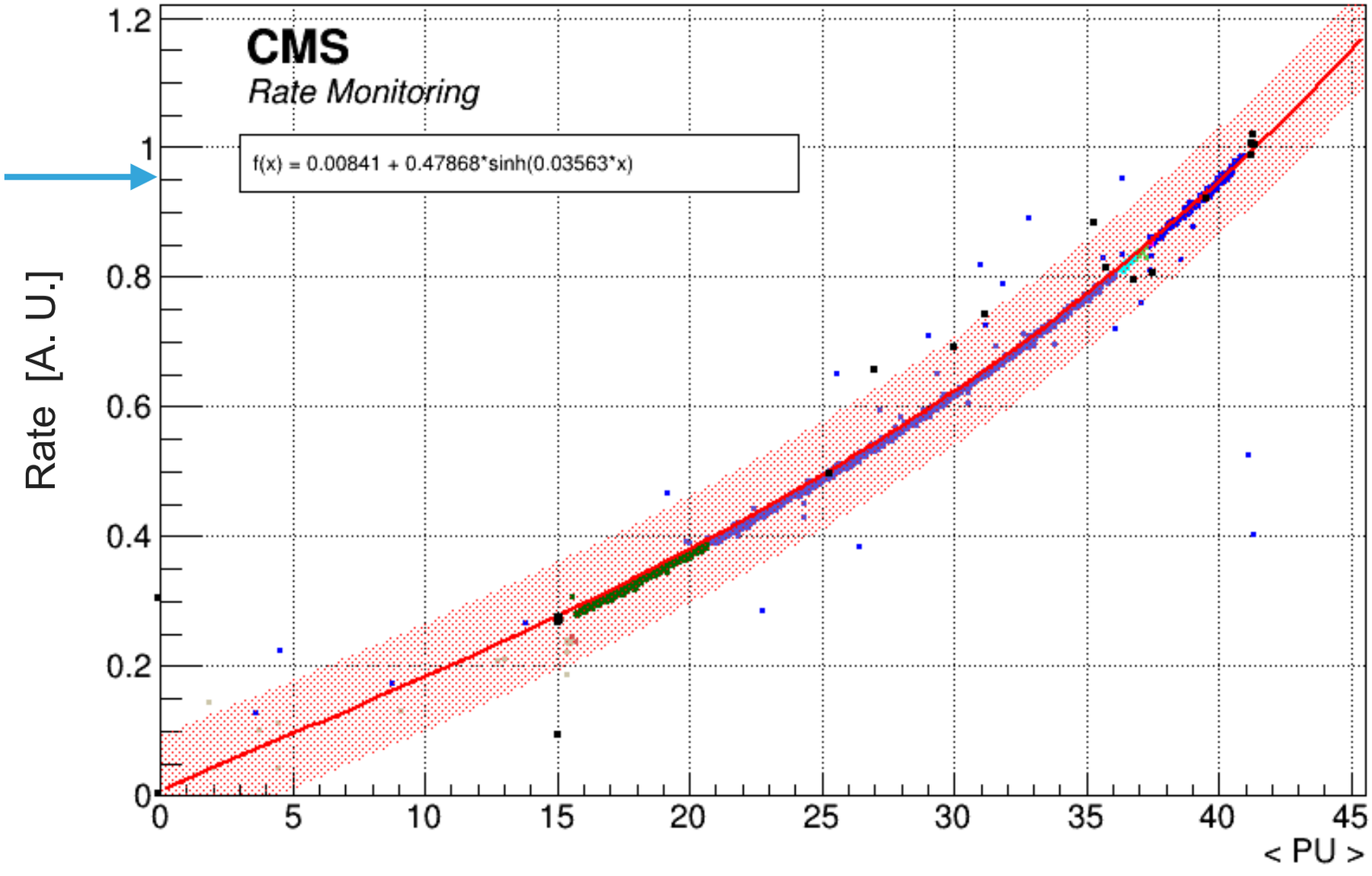


- At high luminosity, many collisions happen simultaneously (pileup)!
- Pileup makes our data more complex and noisy

TODAY'S TRIGGER IN 200 PILEUP

L1_HTT360er (used in gg→H→bb search!)

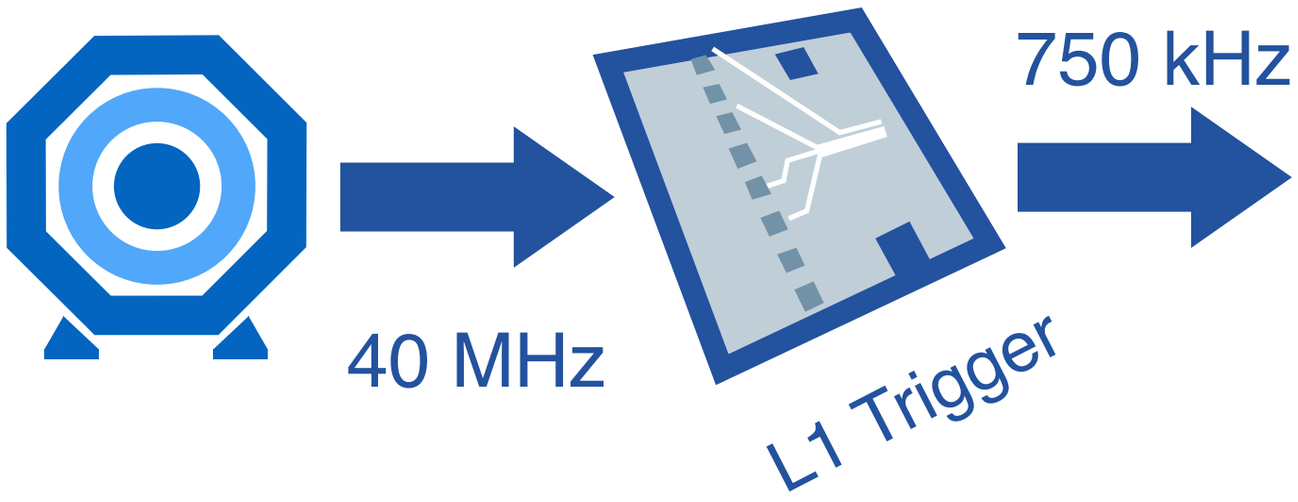
~2.4 kHz



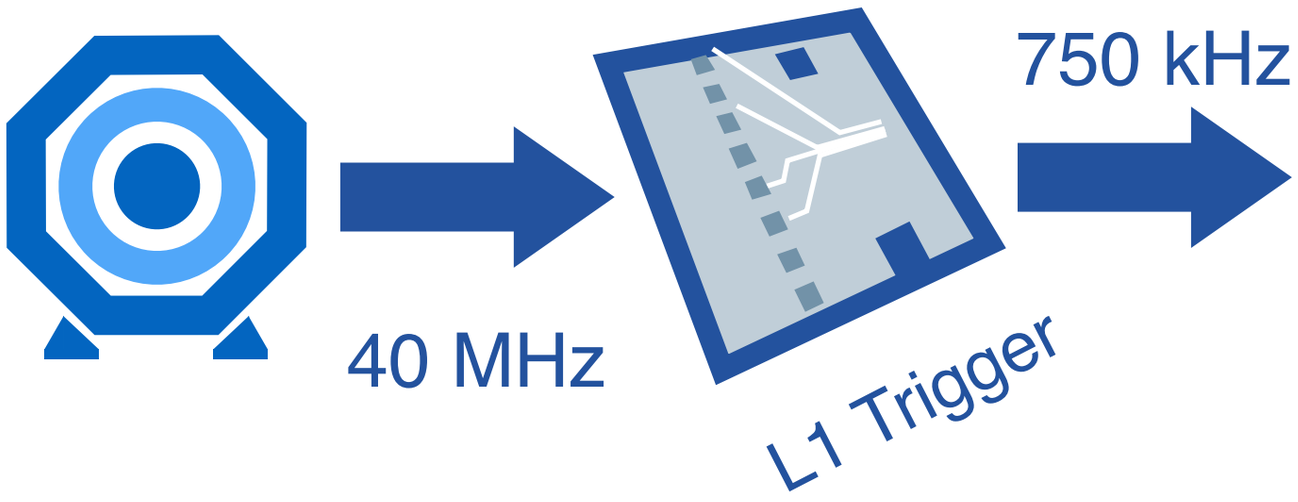
- Jet activity trigger @ 40 pileup: ~2.4 kHz (2.4% of budget)
  @ 200 pileup: ~750 kHz (100% of budget)

- Deep learning can help!

UPGRADED LEVEL-1 TRIGGER



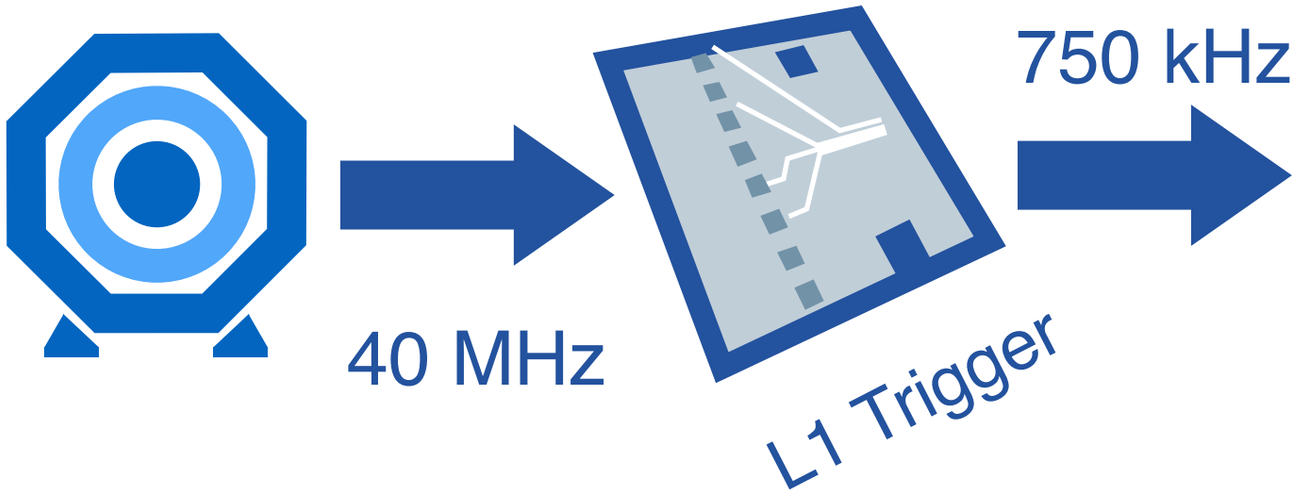
UPGRADED LEVEL-1 TRIGGER

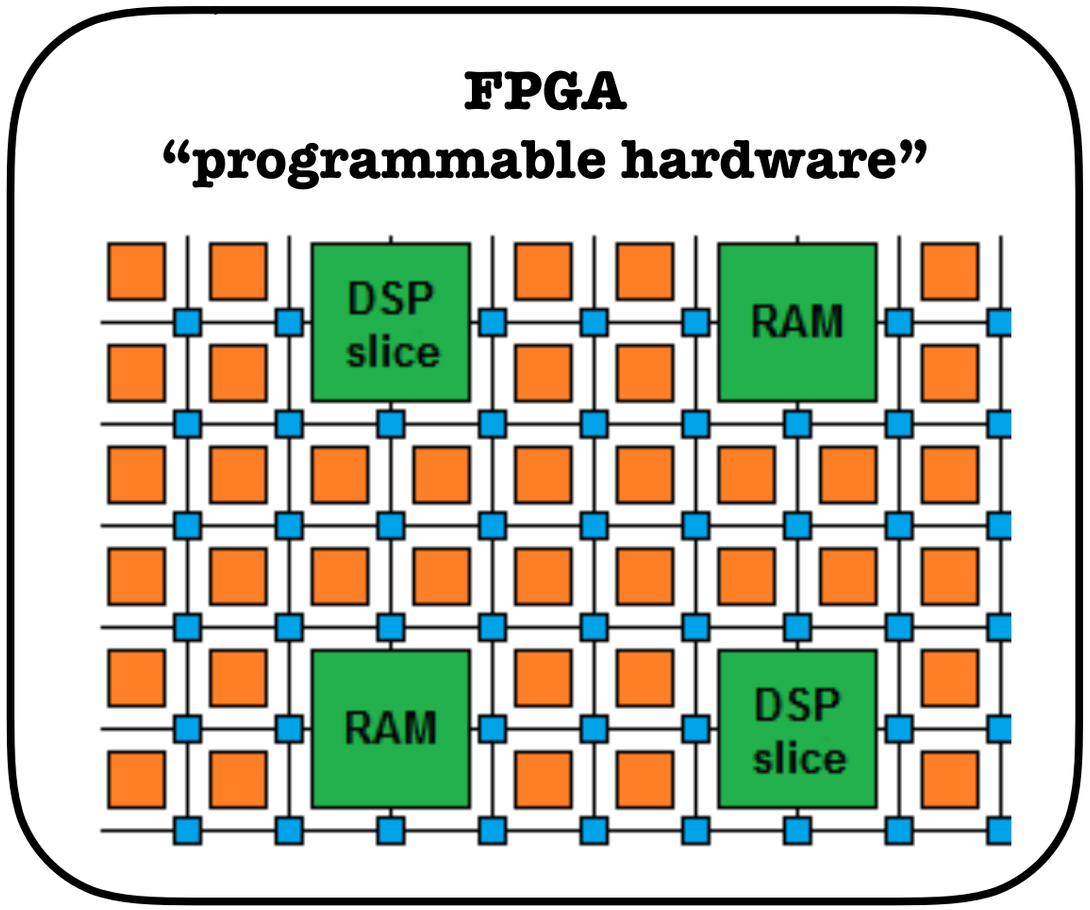


- Level-1 Trigger:
  40 MHz → 750 kHz

- Reconstruct and filter
  2% of events in ~12 μs

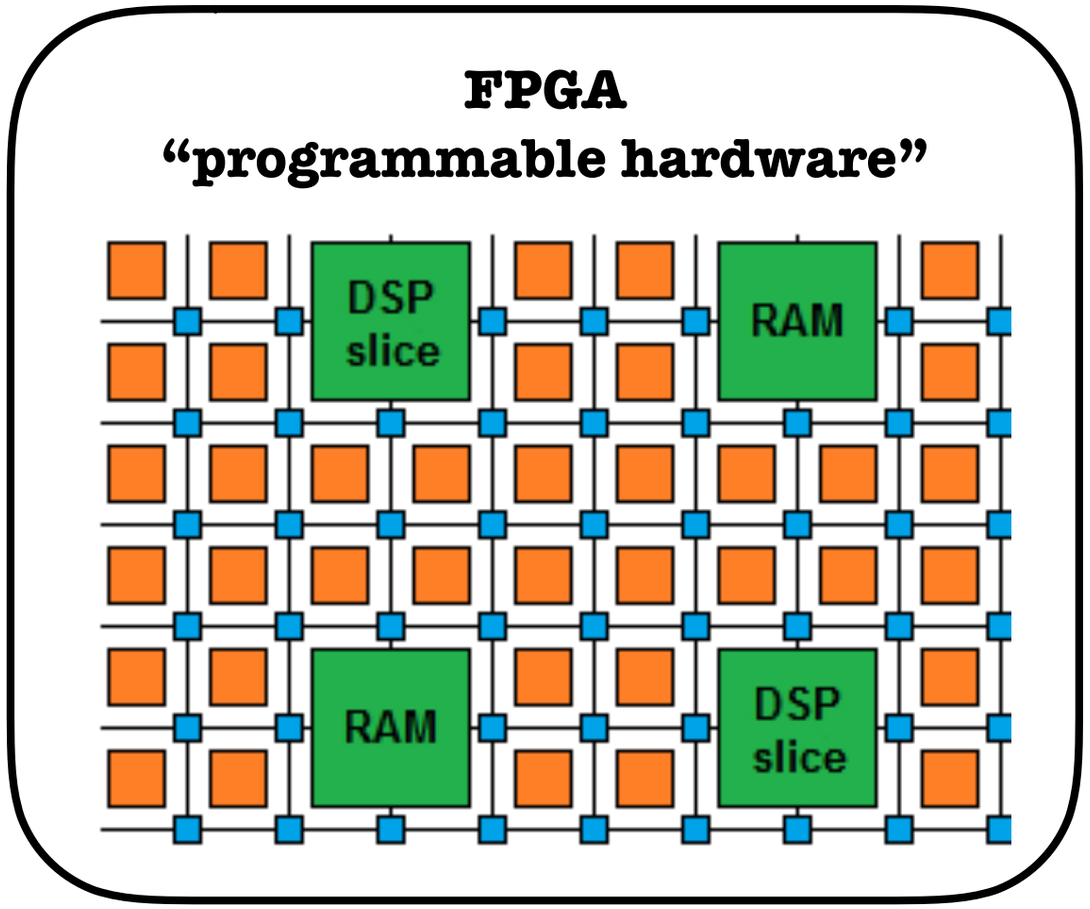
UPGRADED LEVEL-1 TRIGGER





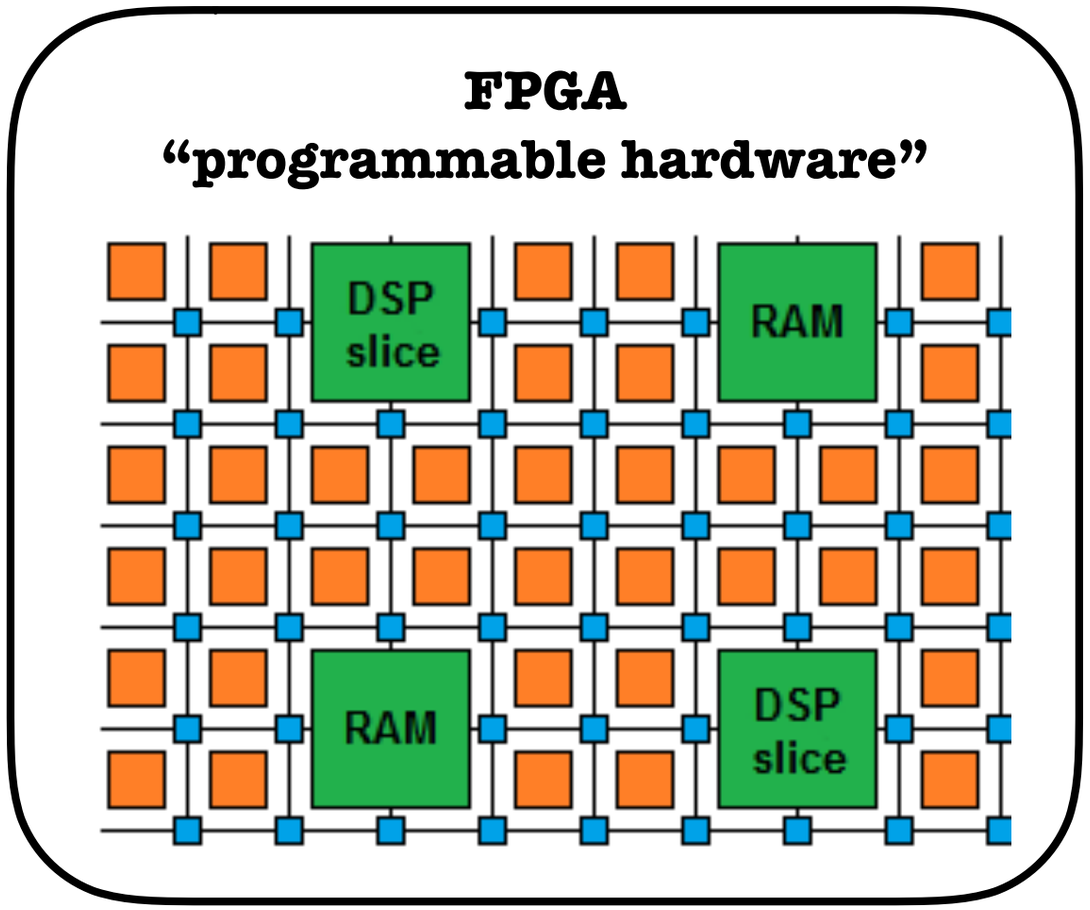
- Level-1 Trigger:
  40 MHz → 750 kHz
- Reconstruct and filter
  2% of events in ~12 μs
- Latency necessitates all **FPGA** design

FPGAS: REPROGRAMMABLE HARDWARE



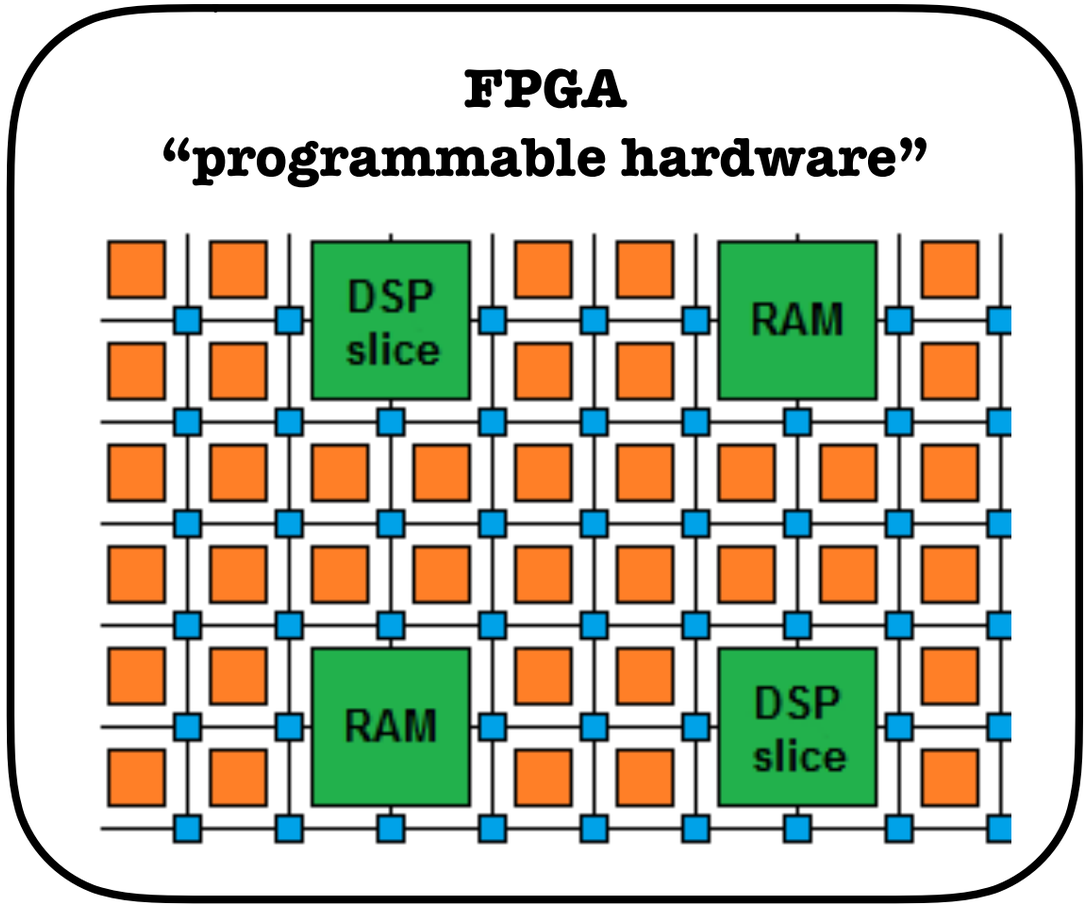
FPGAS: REPROGRAMMABLE HARDWARE

- **Pros:**



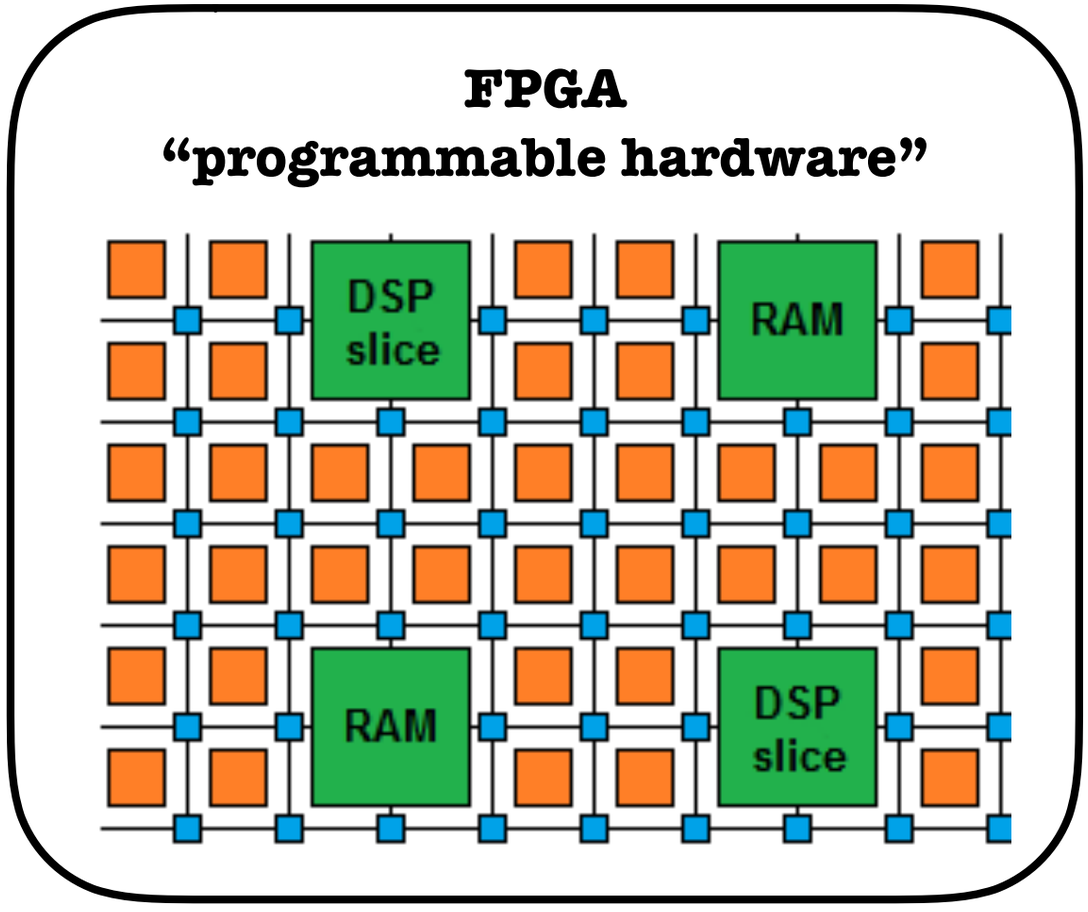
FPGAS: REPROGRAMMABLE HARDWARE

- **Pros:**
  - **Reprogrammable** interconnects between **embedded components** that perform multiplication (**DSPs**), apply logical functions (**LUTs**), or store memory (**BRAM**)



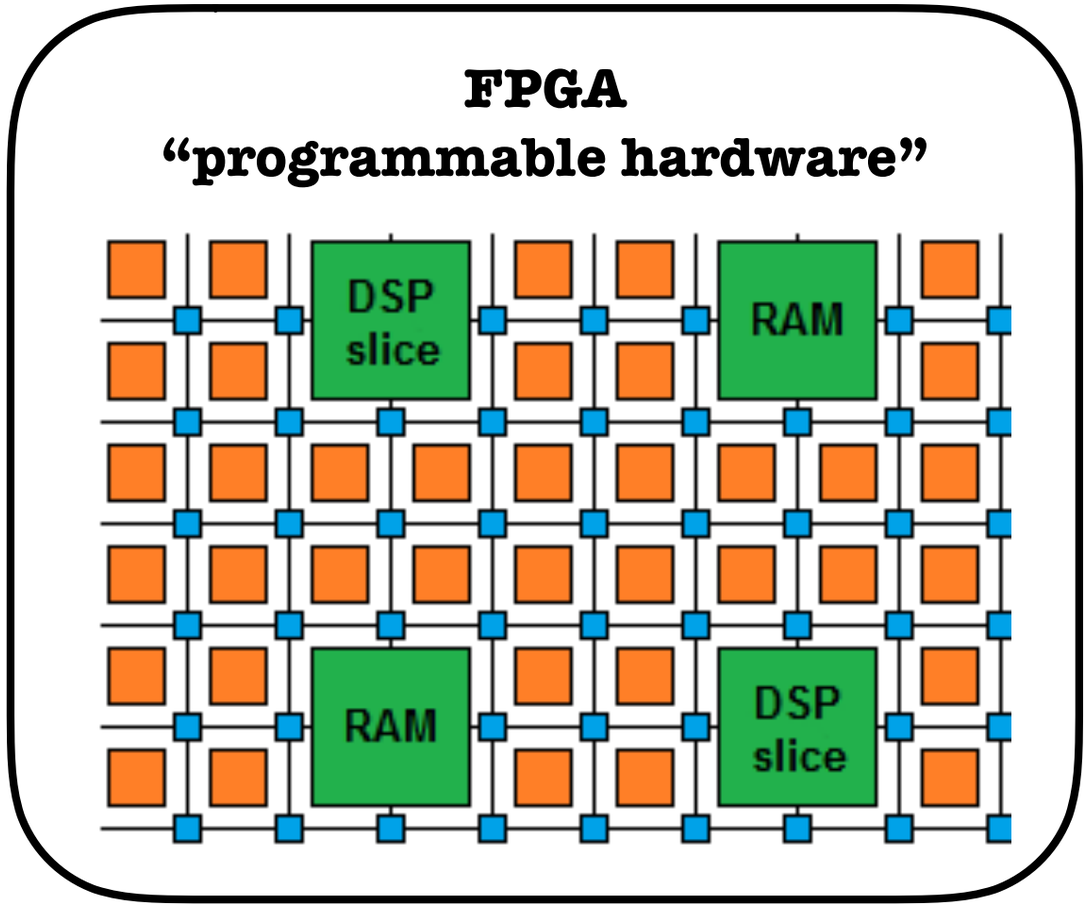
FPGAS: REPROGRAMMABLE HARDWARE

- **Pros:**
  - **Reprogrammable** interconnects between **embedded components** that perform multiplication (**DSPs**), apply logical functions (**LUTs**), or store memory (**BRAM**)
  - **High throughput:** O(100) optical transceivers running at O(15) Gbs



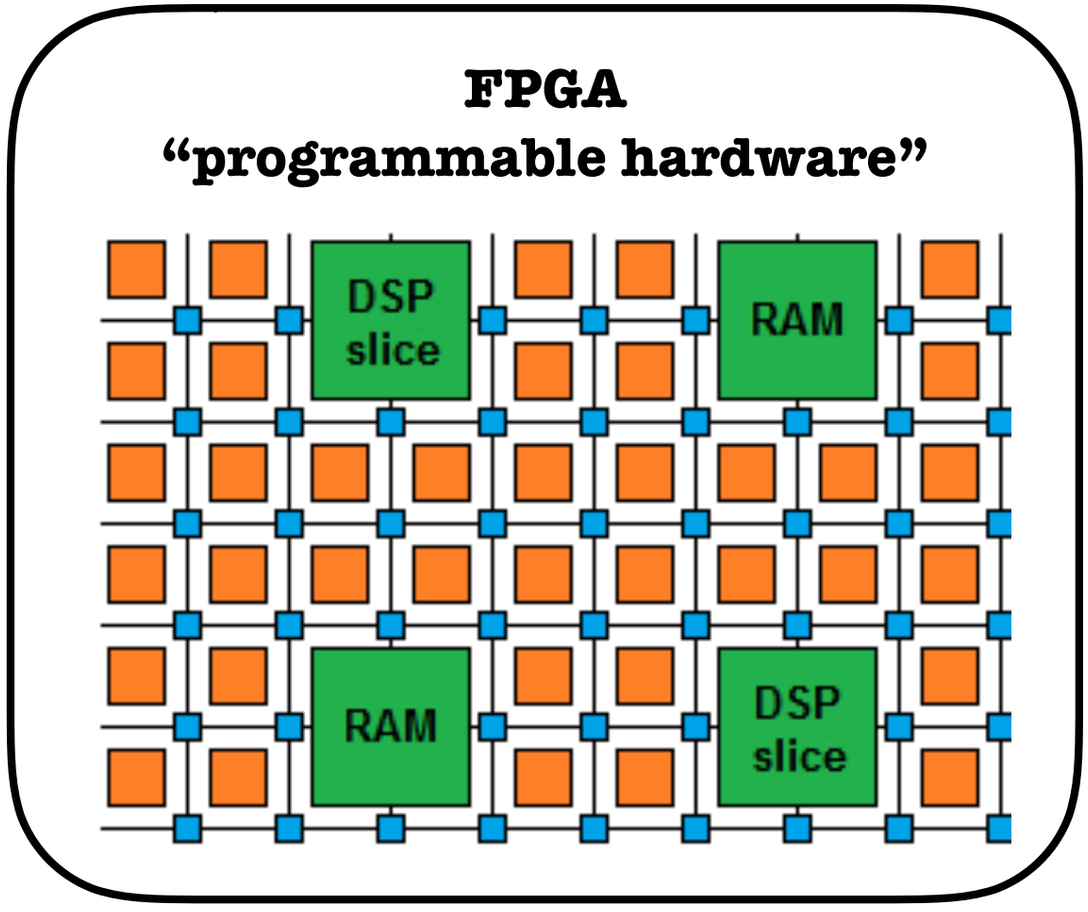
FPGAS: REPROGRAMMABLE HARDWARE

- **Pros:**
  - **Reprogrammable** interconnects between **embedded components** that perform multiplication (**DSPs**), apply logical functions (**LUTs**), or store memory (**BRAM**)
  - **High throughput:** O(100) optical transceivers running at O(15) Gbs
  - **Massively parallel**



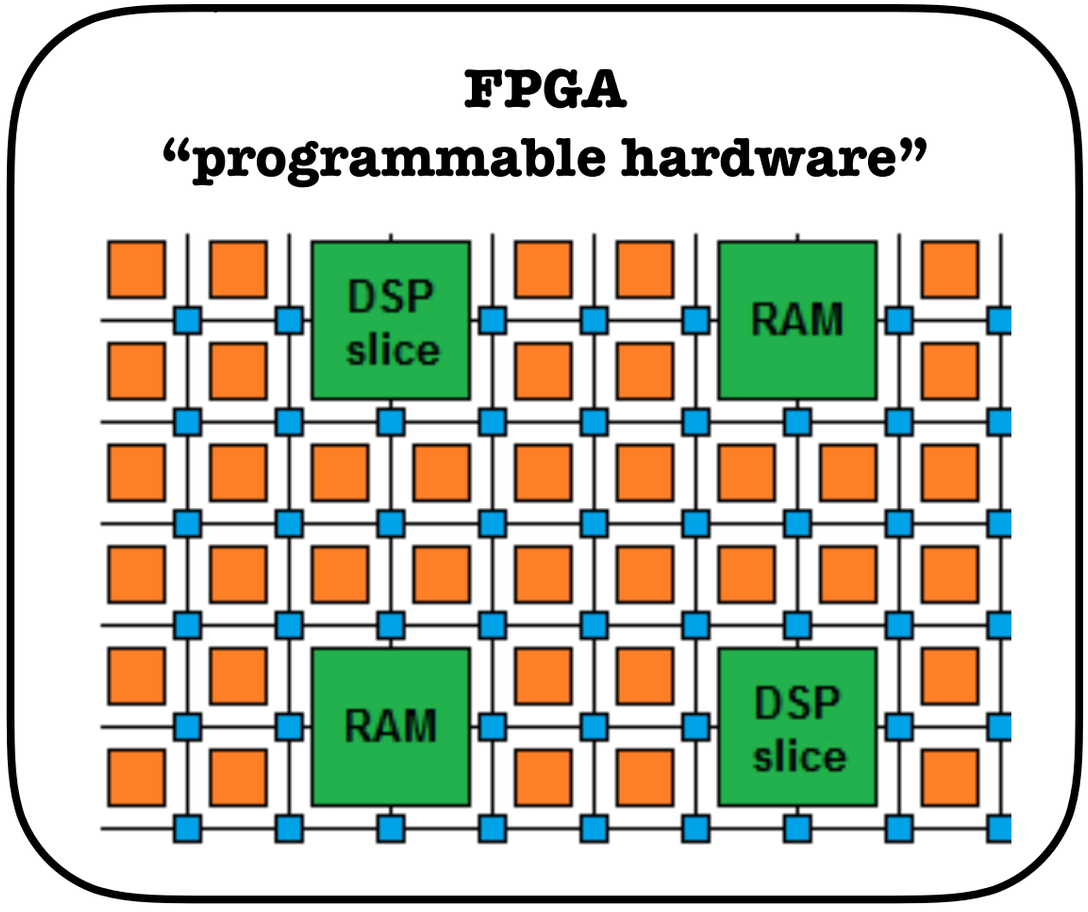
FPGAS: REPROGRAMMABLE HARDWARE

- **Pros:**
  - **Reprogrammable** interconnects between **embedded components** that perform multiplication (**DSPs**), apply logical functions (**LUTs**), or store memory (**BRAM**)
  - **High throughput:** O(100) optical transceivers running at O(15) Gbs
  - **Massively parallel**
  - **Low power**



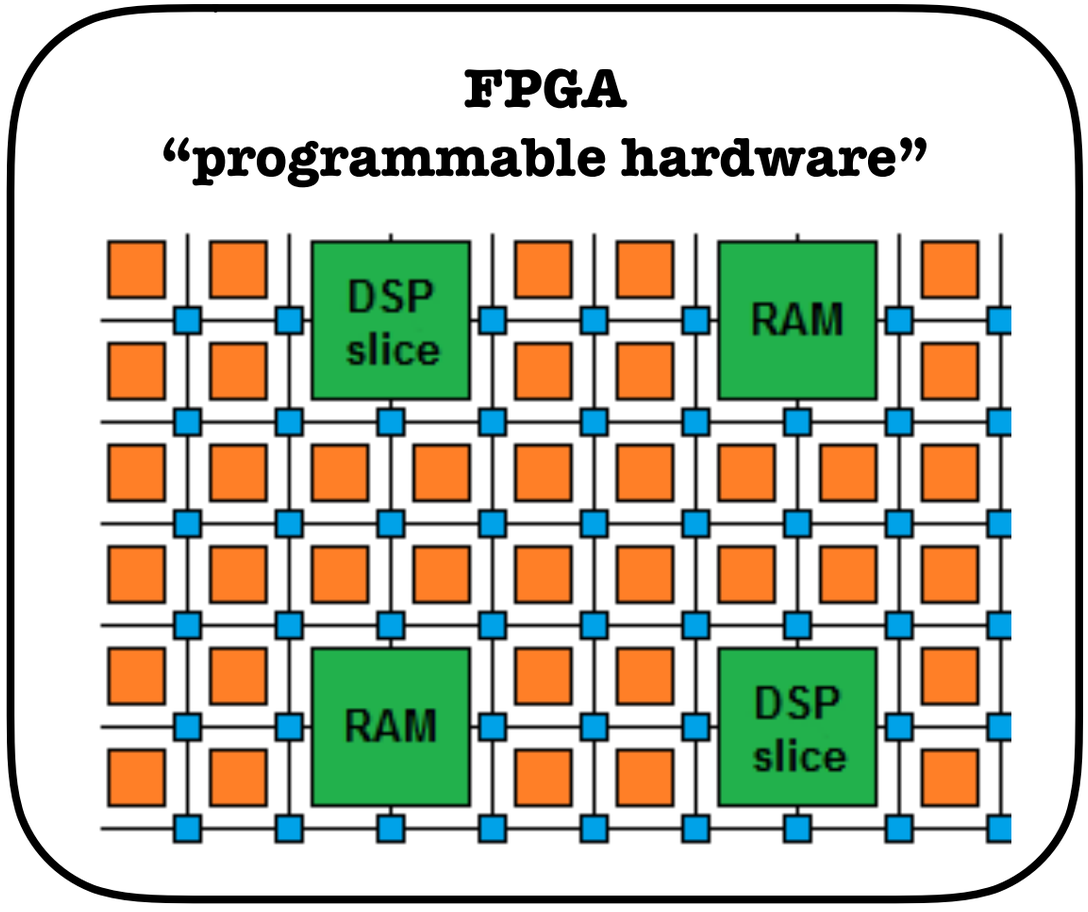
FPGAS: REPROGRAMMABLE HARDWARE

- **Pros:**
  - **Reprogrammable** interconnects between **embedded components** that perform multiplication (**DSPs**), apply logical functions (**LUTs**), or store memory (**BRAM**)
    - **High throughput:** O(100) optical transceivers running at O(15) Gbs
    - **Massively parallel**
    - **Low power**
- **Cons:**



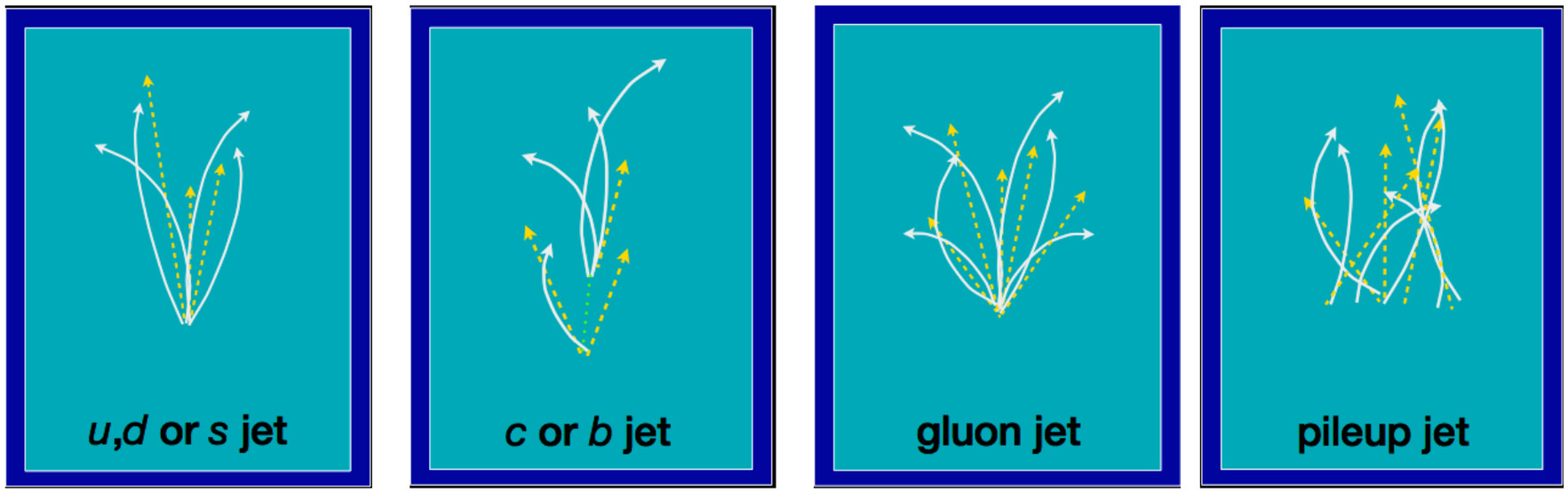
FPGAS: REPROGRAMMABLE HARDWARE

- **Pros:**
  - **Reprogrammable** interconnects between **embedded components** that perform multiplication (**DSPs**), apply logical functions (**LUTs**), or store memory (**BRAM**)
    - **High throughput:** O(100) optical transceivers running at O(15) Gbs
  - **Massively parallel**
  - **Low power**



- **Cons:**
  - **Requires domain knowledge to program** (using VHDL/Verilog)

CASE STUDY FOR THE TRIGGER: JET TAGGING



- Use machine learning to classify jets in the trigger

CASE STUDY: JET TAGGING INPUTS

Observables

m_{mMDT}

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

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

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

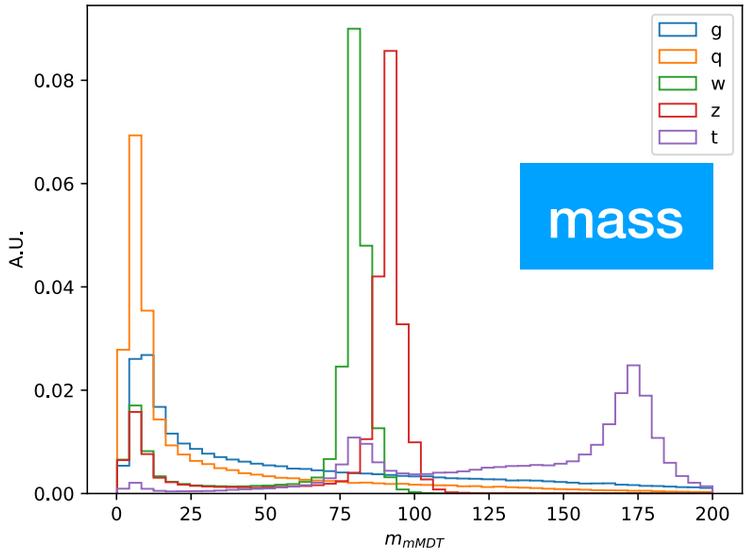
$C_2^{\beta=1,2}$

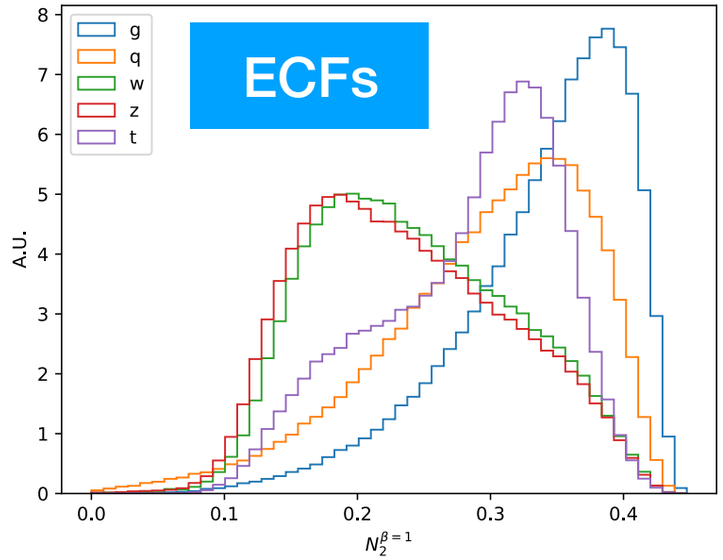
$D_2^{\beta=1,2}$

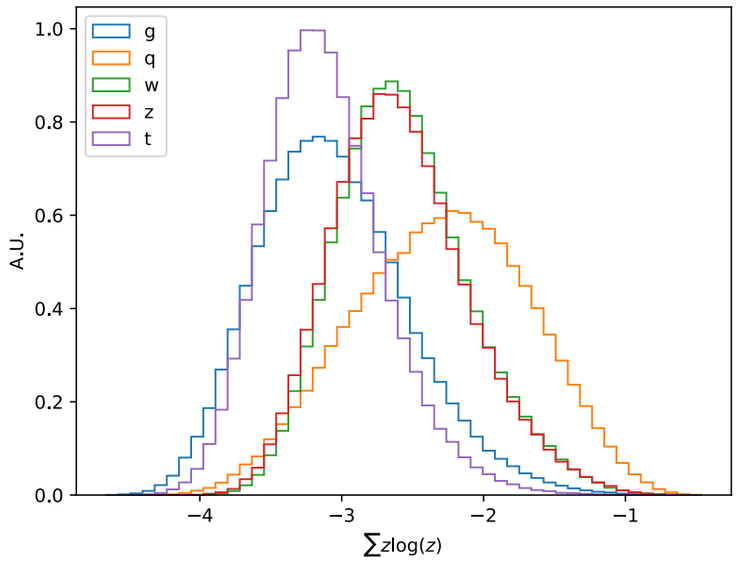
$D_2^{(\alpha,\beta)=(1,1),(1,2)}$

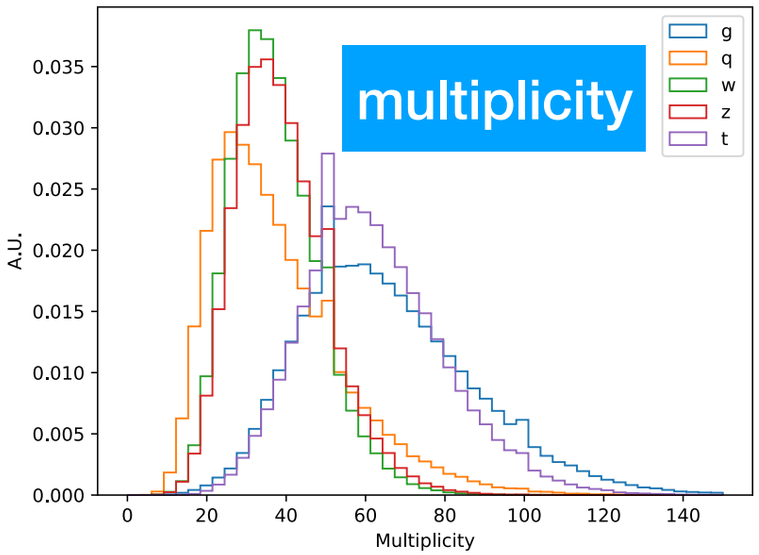
$\sum z \log z$

Multiplicity





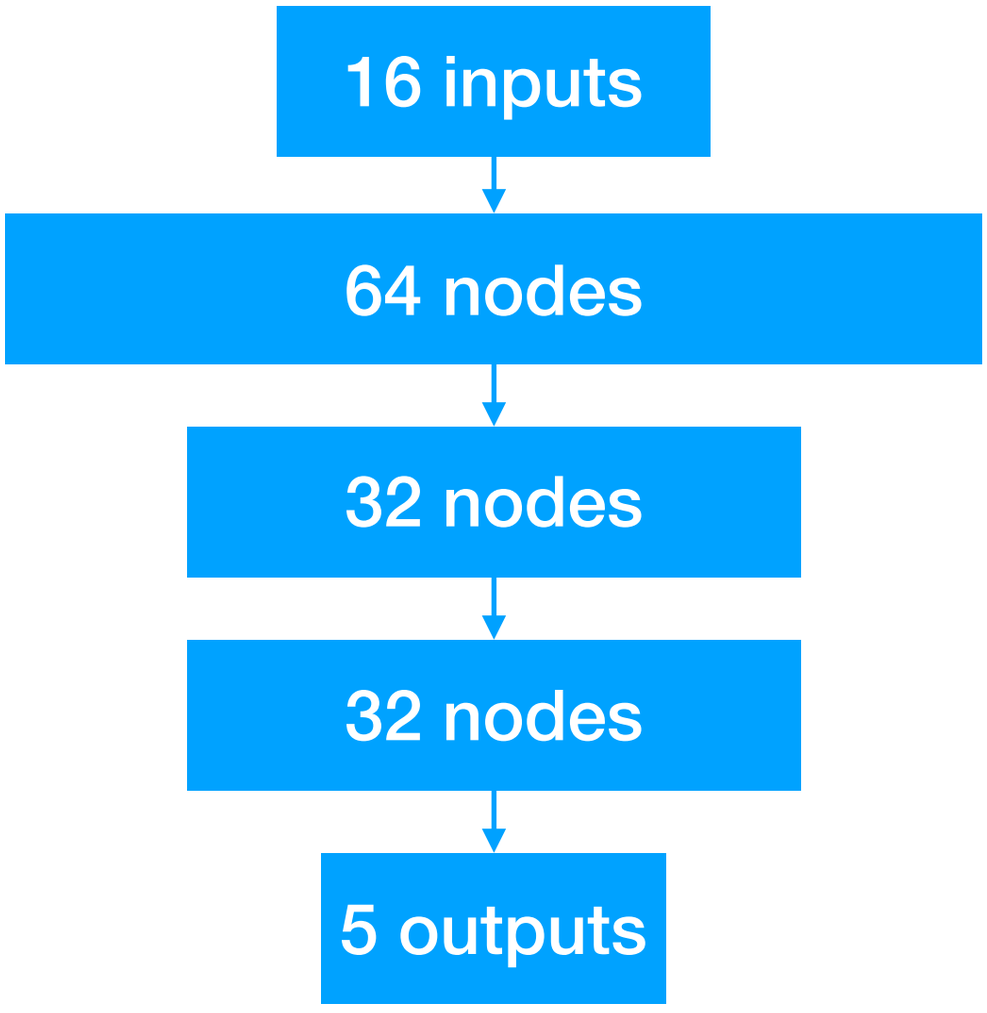


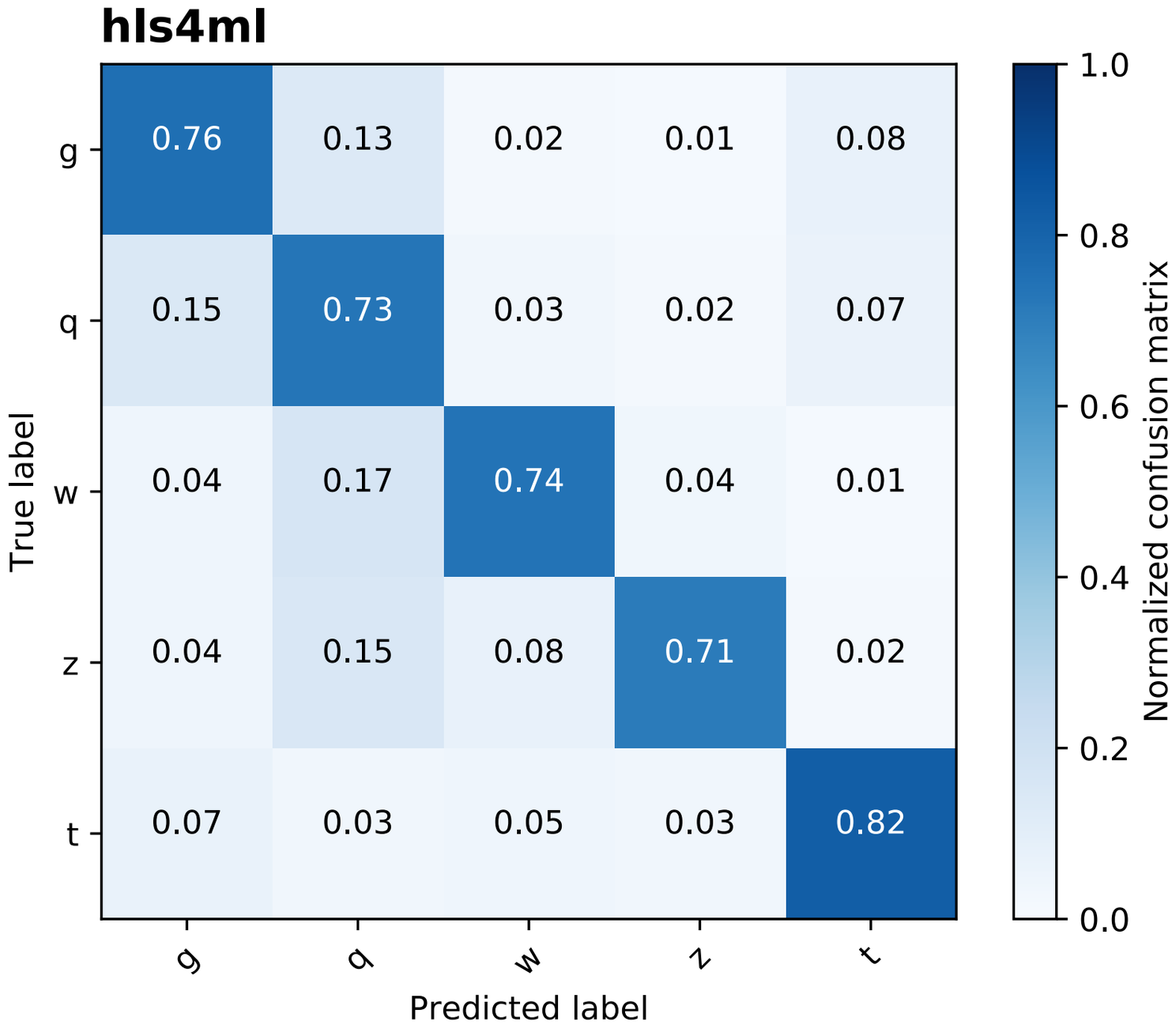


- 16 expert observables provide separation between top, W/Z, and quark/gluon

CASE STUDY: JET TAGGING PERFORMANCE

Moderate NN correctly identifies jets 70-80% of the time

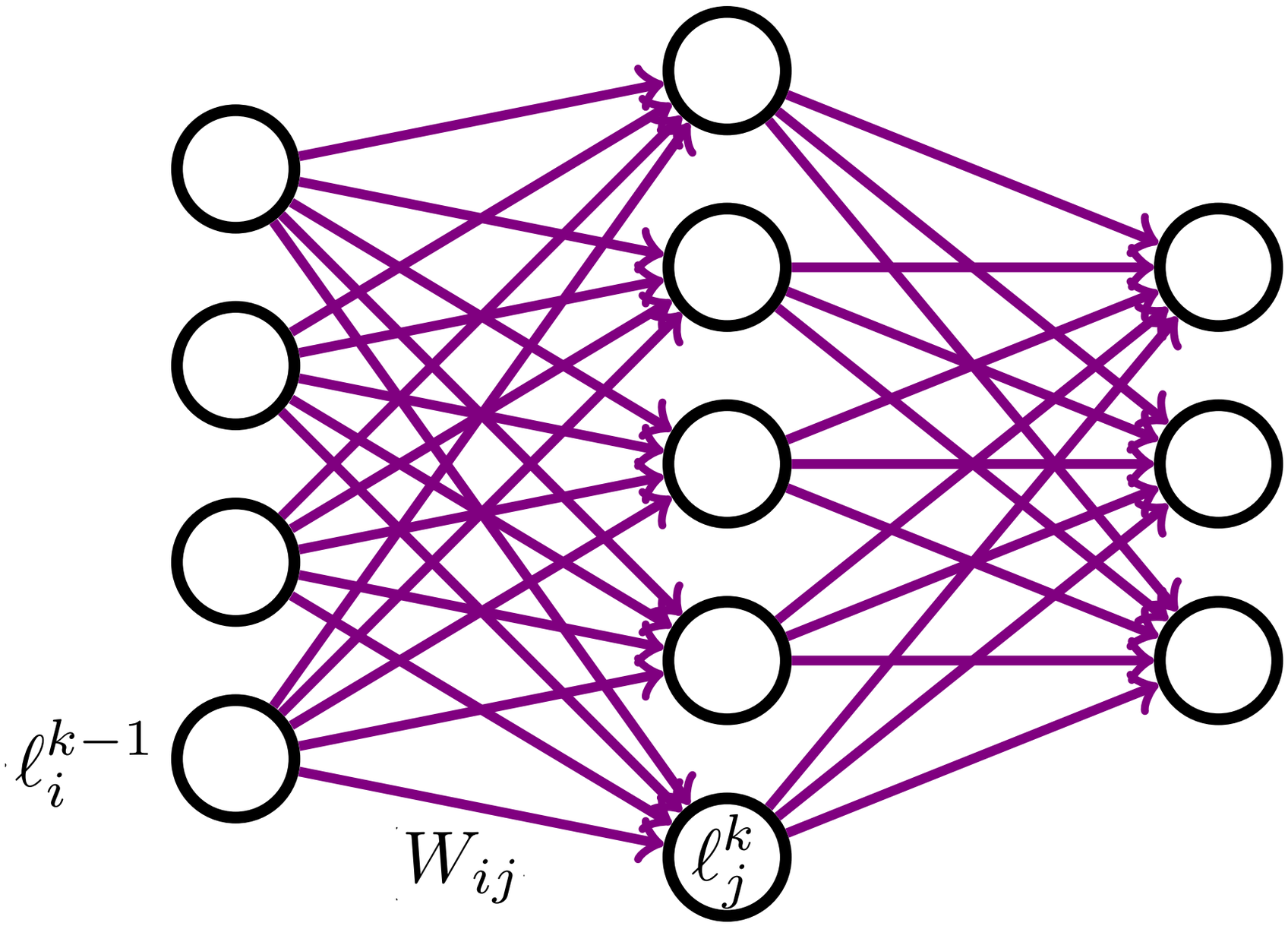




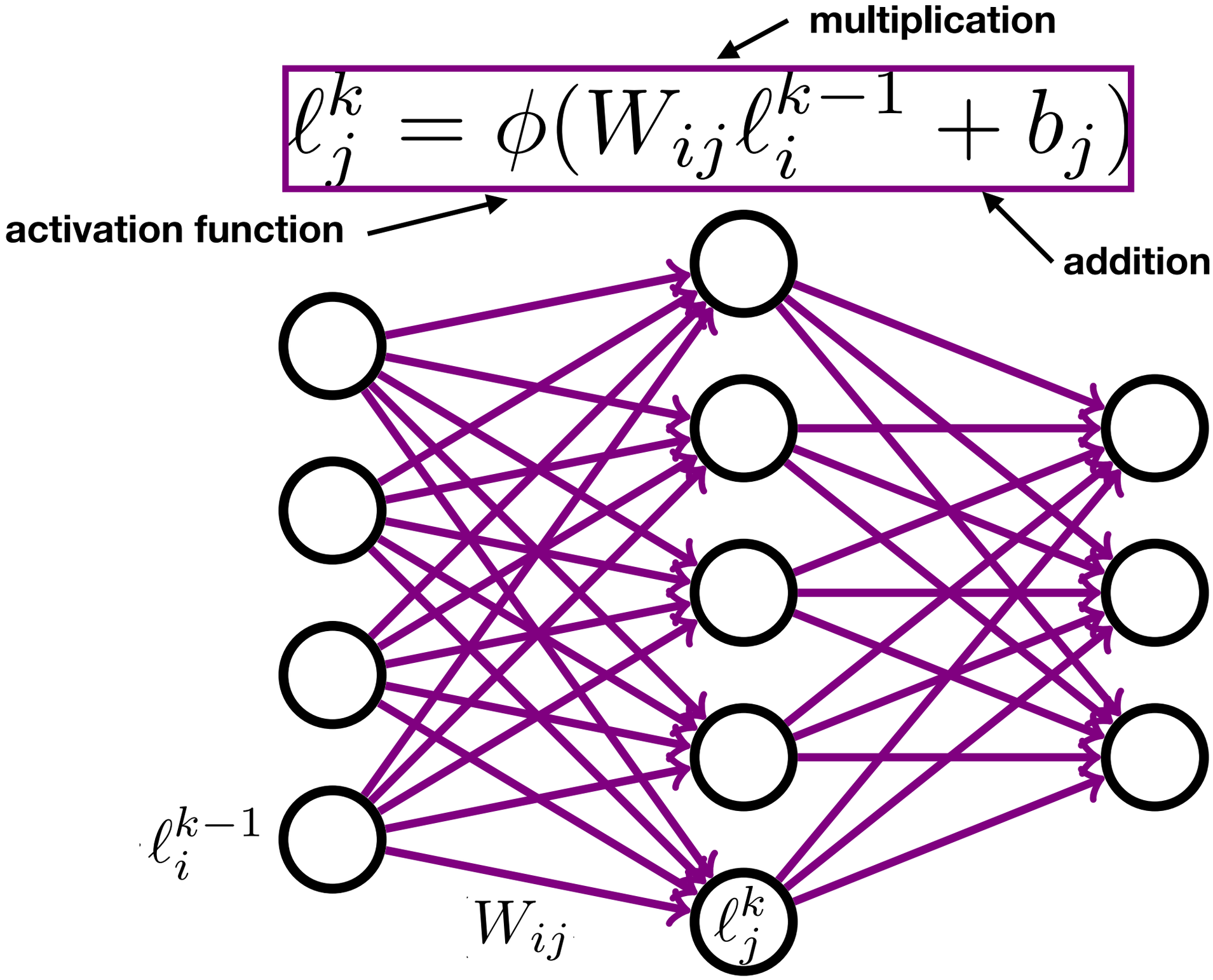
- Use machine learning to classify jets in the trigger

NEURAL NETWORK OPERATIONS

$$\ell_j^k = \phi(W_{ij}\ell_i^{k-1} + b_j)$$



NEURAL NETWORK OPERATIONS



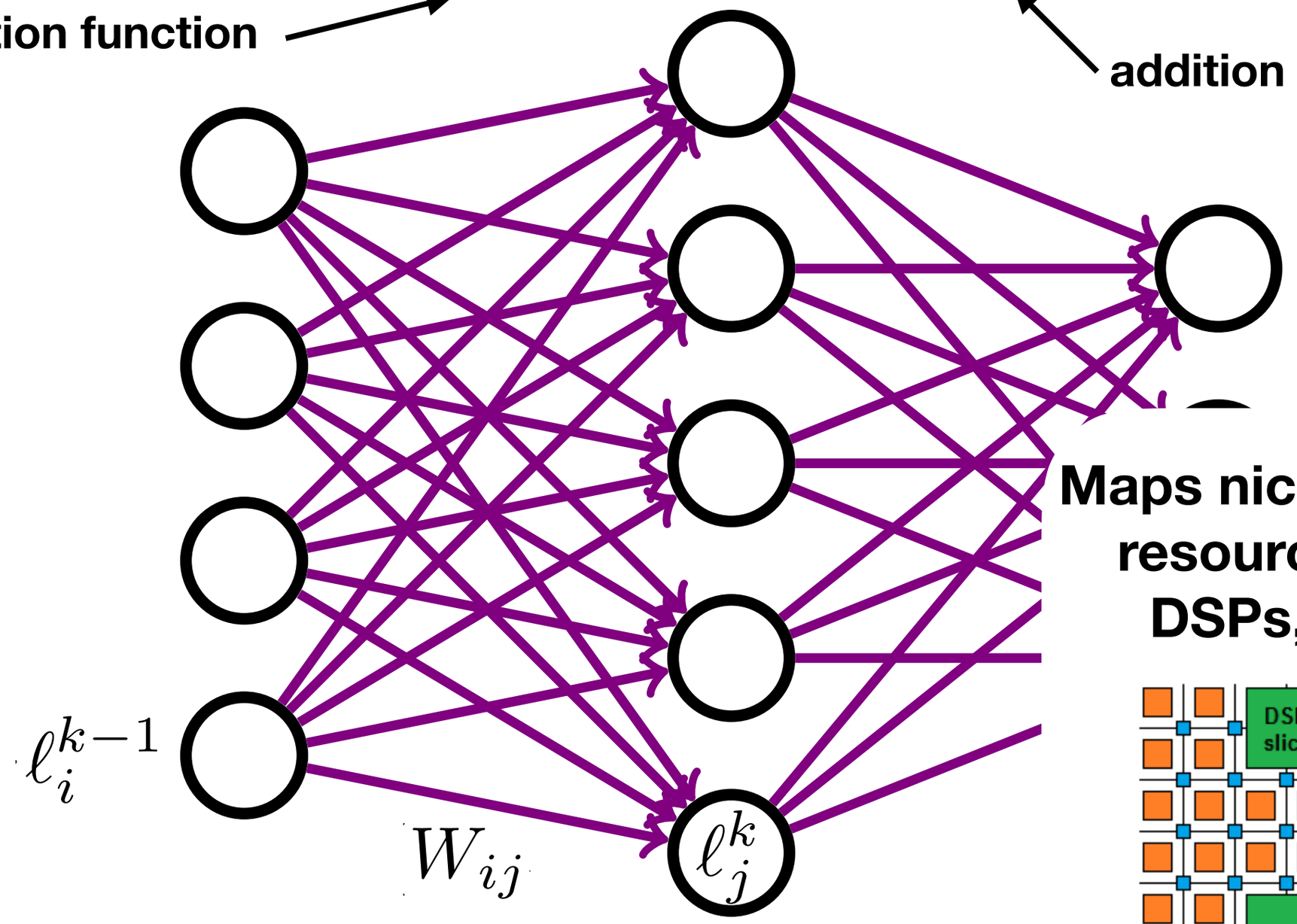
NEURAL NETWORK OPERATIONS

multiplication

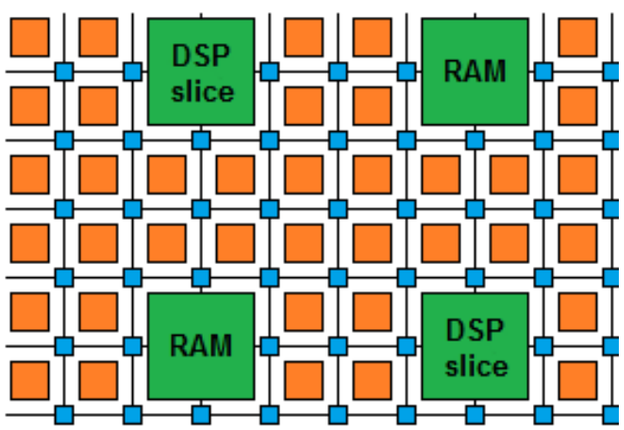
$$\ell_j^k = \phi(W_{ij}\ell_i^{k-1} + b_j)$$

activation function

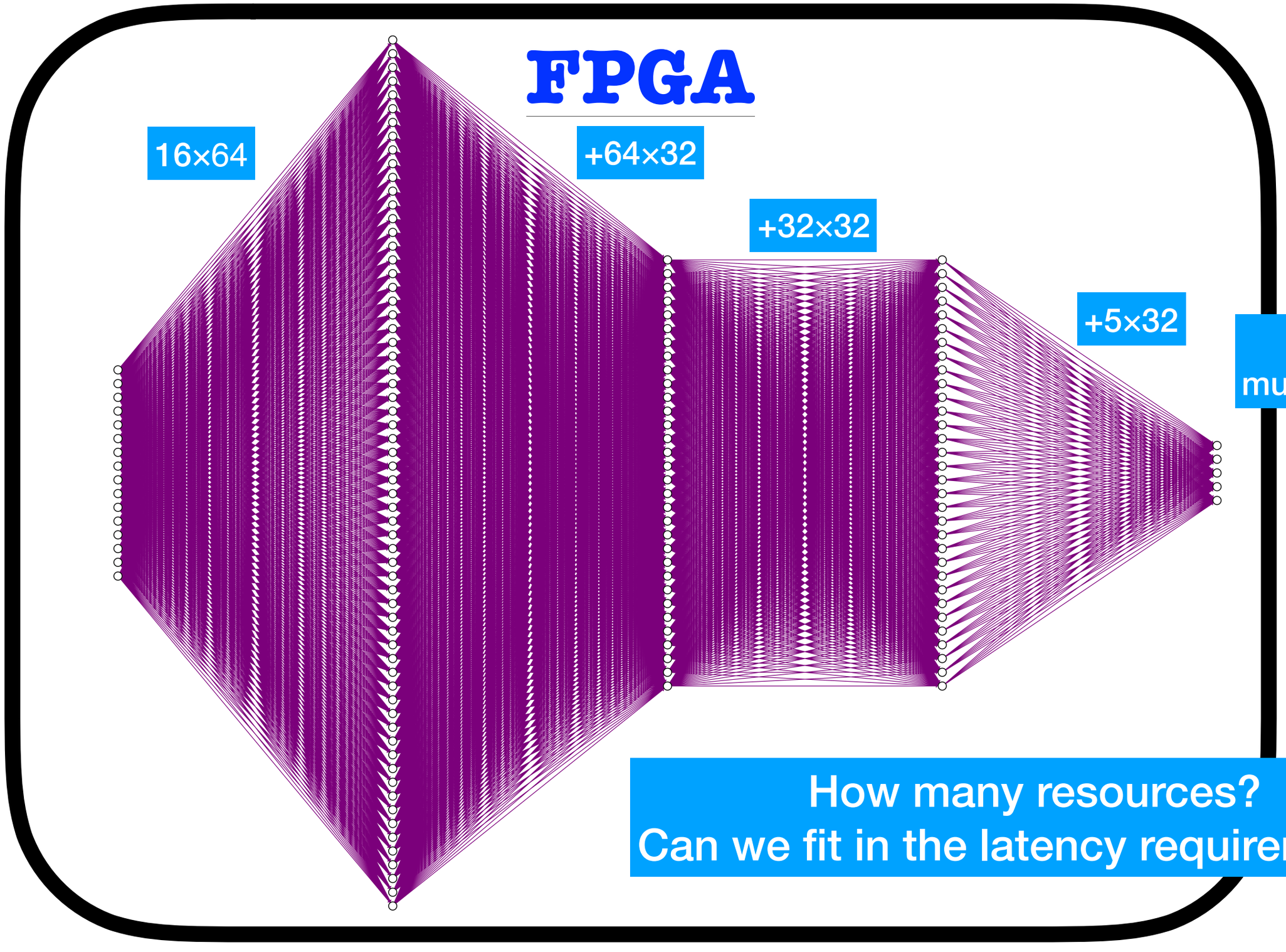
addition



Maps nicely onto FPGA resources: high IO, DSPs, LUTs, etc.



MACHINE LEARNING IN FPGAS?



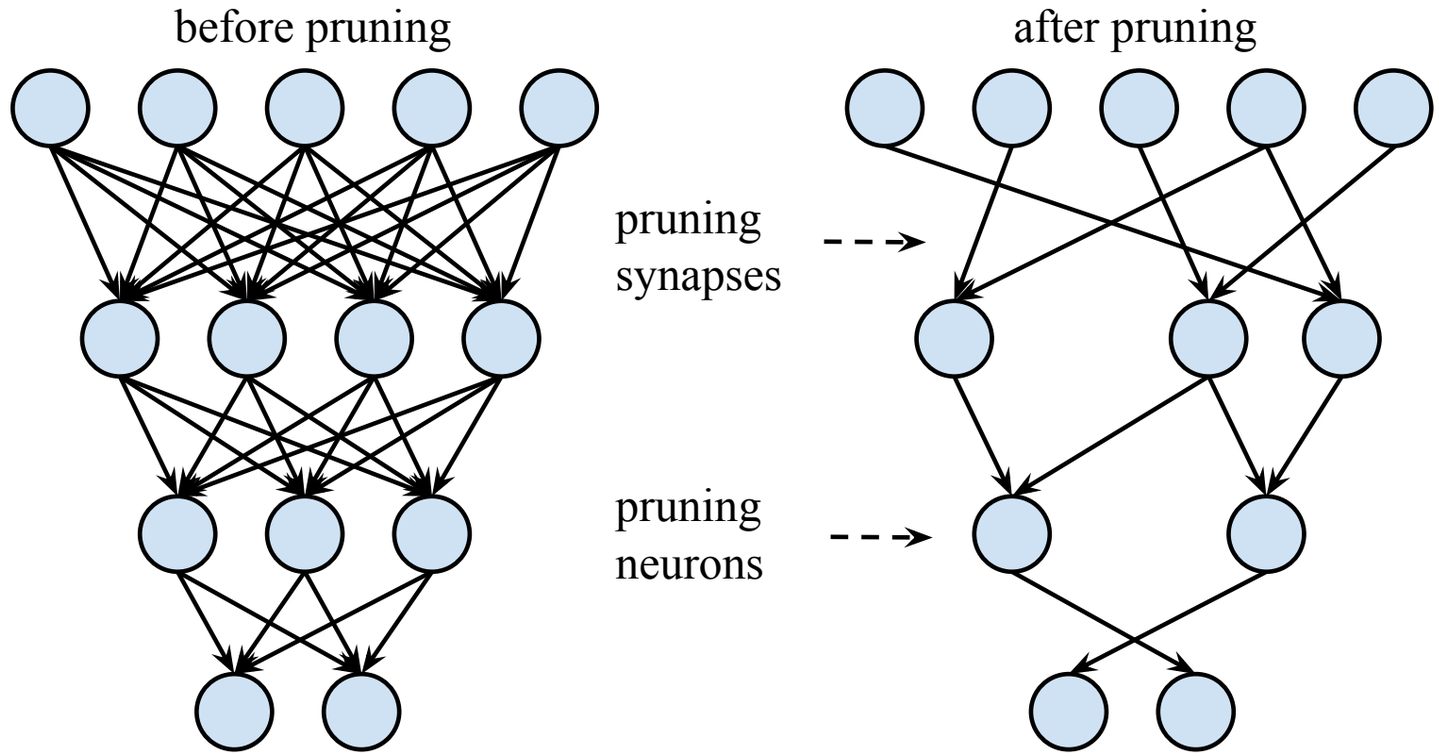
(ENERGY) EFFICIENT NEURAL NETWORKS

Further reading: arXiv:1510.00149

(ENERGY) EFFICIENT NEURAL NETWORKS

1. Compression

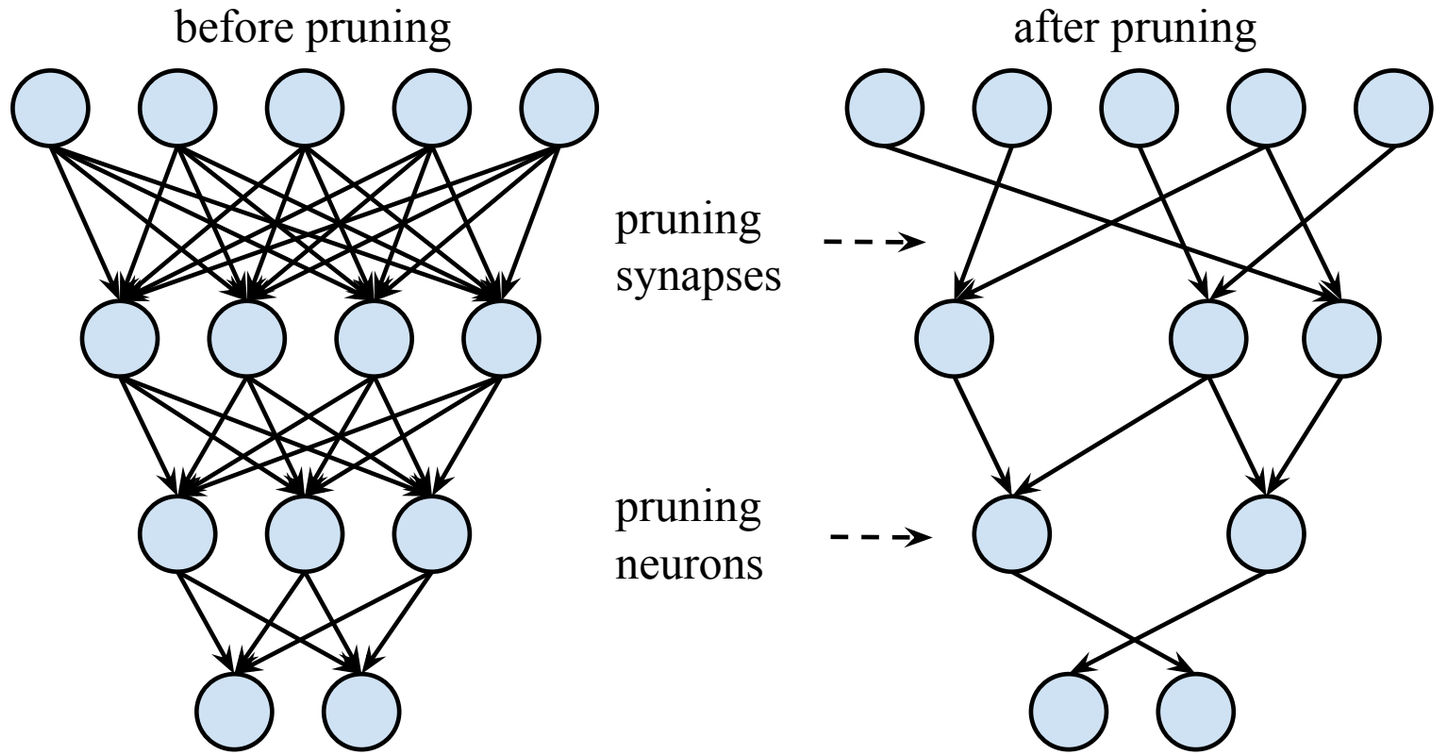
   - Maintain high performance while removing redundant synapses and neurons



Further reading: arXiv:1510.00149

(ENERGY) EFFICIENT NEURAL NETWORKS

1. Compression
   - Maintain high performance while removing redundant synapses and neurons

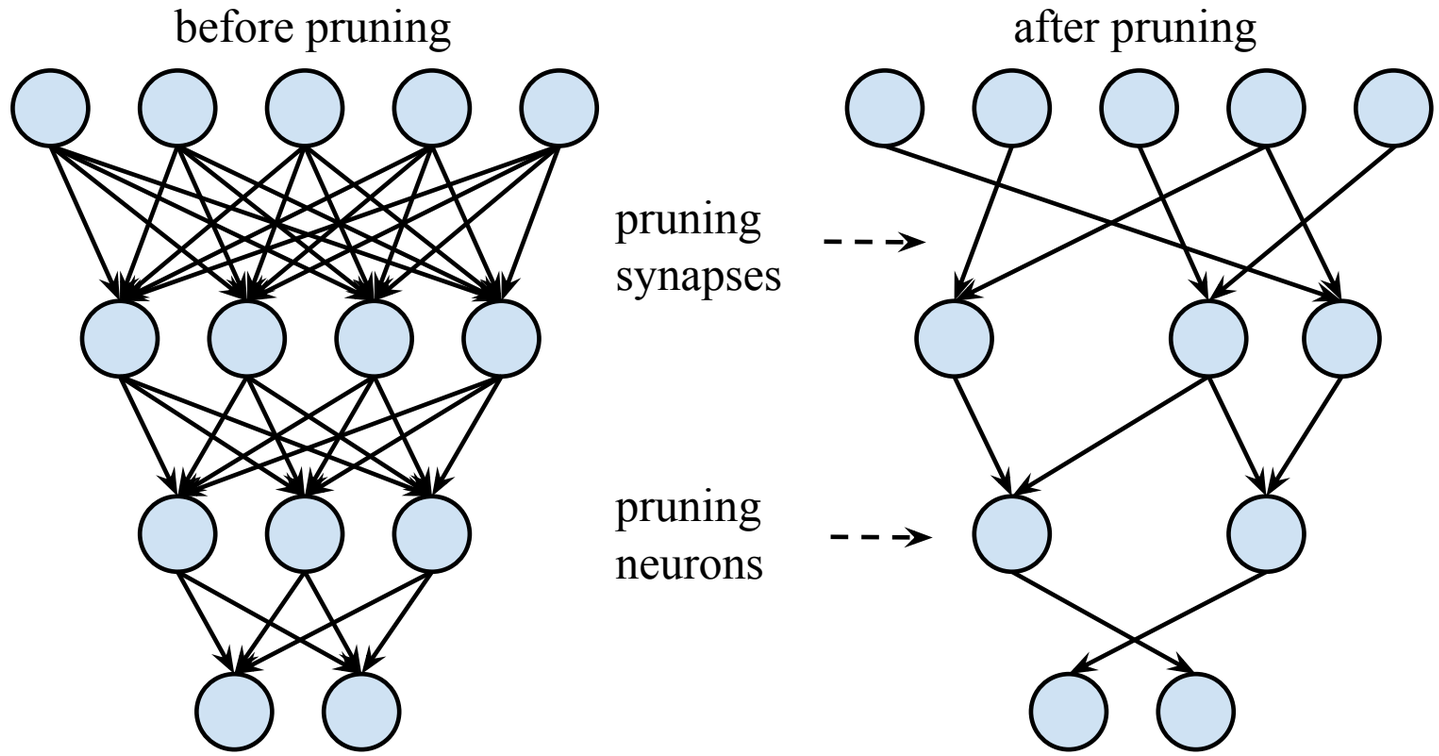


2. Quantization
   - Reduce precision from 32-bit floating point to 20-bit, 8-bit, …

Further reading: [arXiv:1510.00149]

(ENERGY) EFFICIENT NEURAL NETWORKS

1. Compression
   - Maintain high performance while removing redundant synapses and neurons

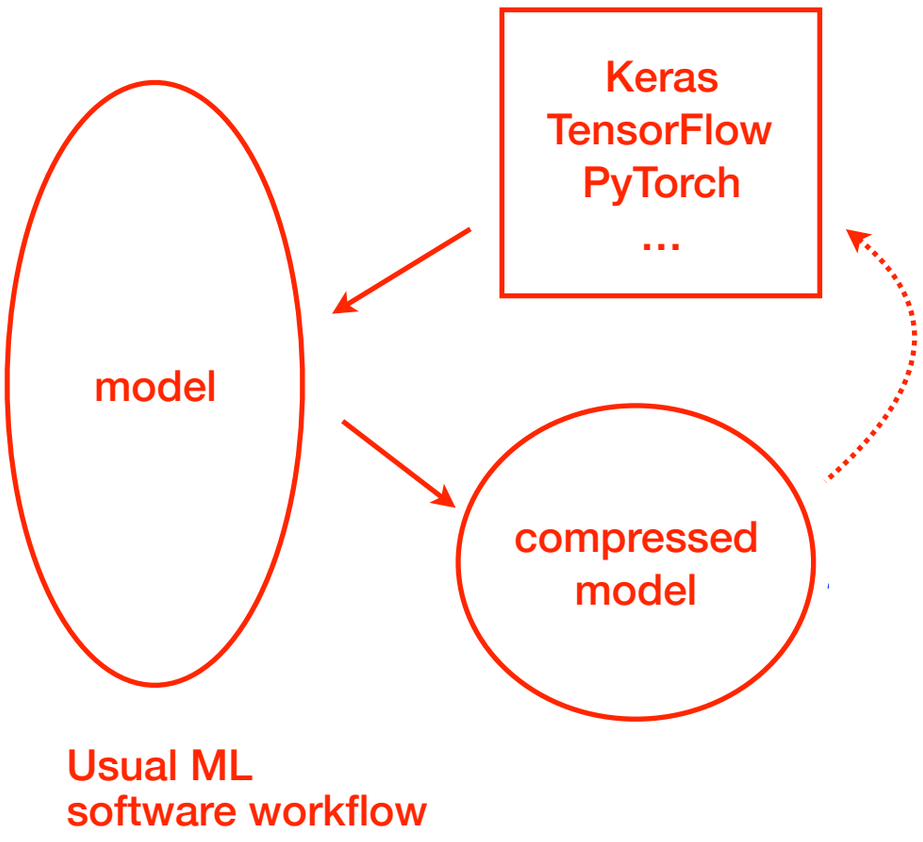


2. Quantization
   - Reduce precision from 32-bit floating point to 20-bit, 8-bit, …
3. Parallelization/Reuse
   - Balance parallelization (how fast) with resources needed (how costly)

Further reading: arXiv:1510.00149

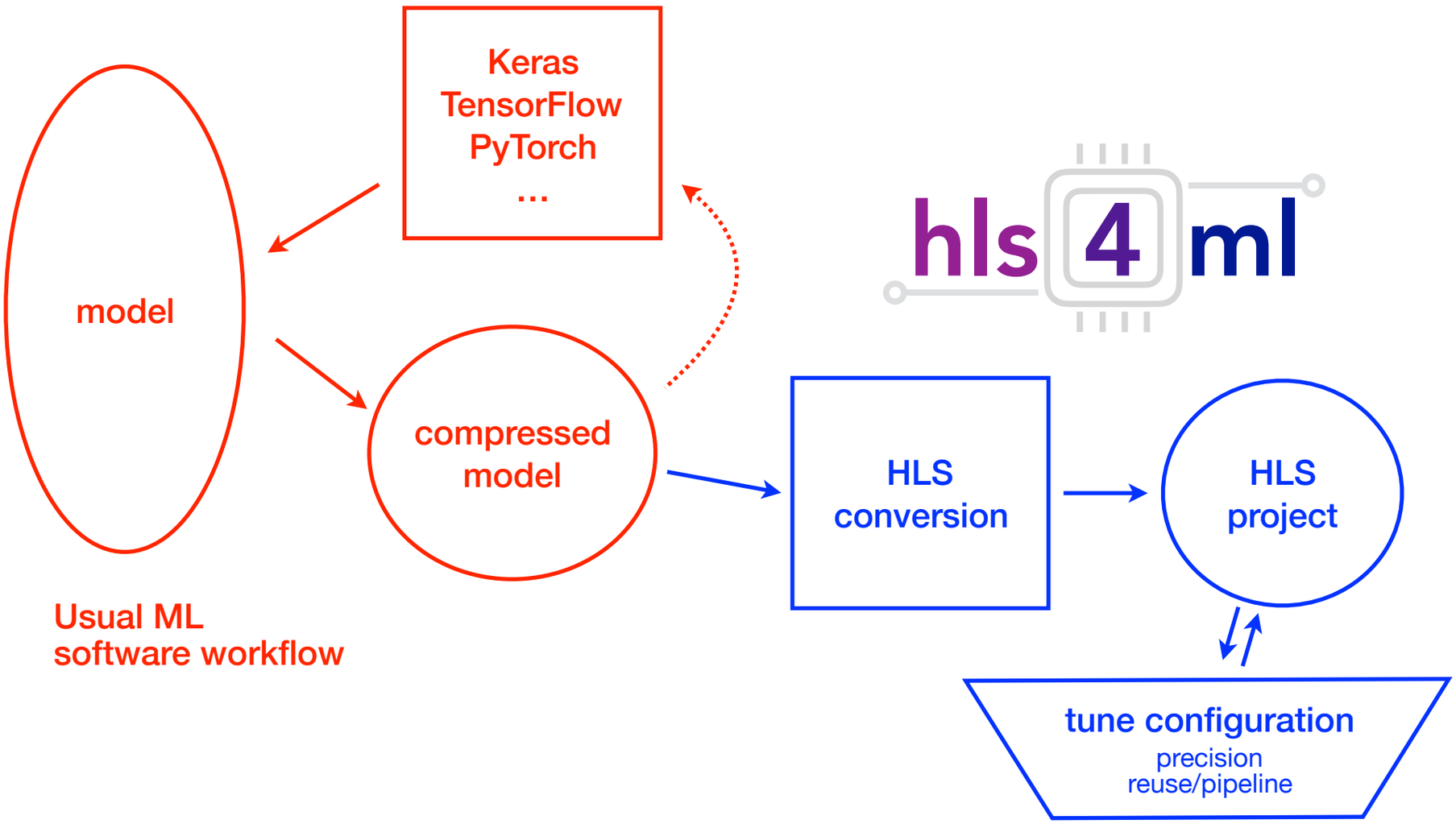
DESIGN EXPLORATION WITH HLS4ML

- **hls4ml** for physicists or ML experts to translate **ML algorithms** into **FPGA firmware**



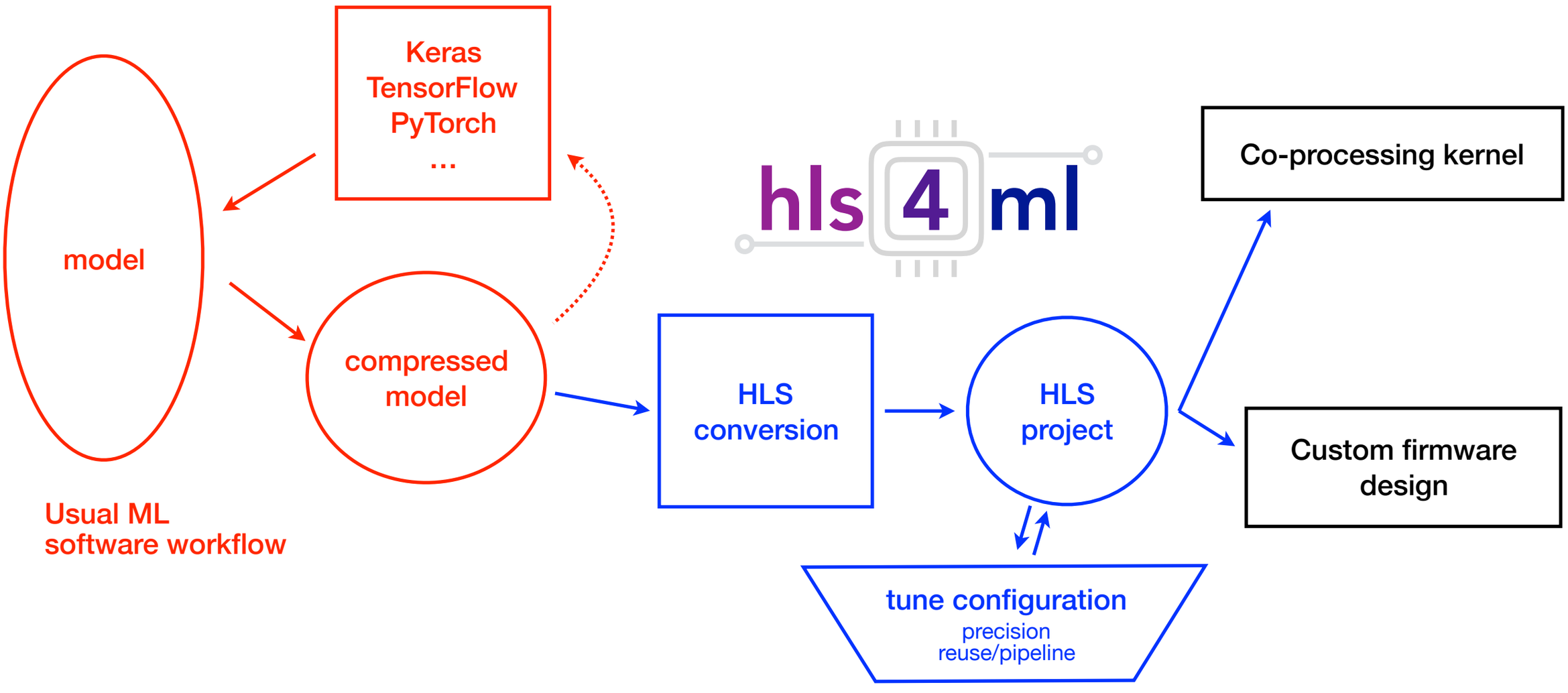
DESIGN EXPLORATION WITH HLS4ML

- **hls4ml** for physicists or ML experts to translate **ML algorithms** into **FPGA firmware**



DESIGN EXPLORATION WITH HLS4ML

- **hls4ml** for physicists or ML experts to translate **ML algorithms** into **FPGA firmware**



TRANSLATION OF ML MODELS

Translation → `python keras-to-hls.py -c keras-config.yml`

Inputs →

Keras

HDF5

```
KerasJson: example-keras-model-files/KERAS_1layer.json
KerasH5:   example-keras-model-files/KERAS_1layer_weights.h5
OutputDir: my-hls-test
ProjectName: myproject
XilinxPart: xc7vx690tffg1927-2
ClockPeriod: 5

IOType: io_parallel # options: io_serial/io_parallel
ReuseFactor: 1
DefaultPrecision: ap_fixed<18,8>
```

Config

- IOType: parallelize or serialize
- ReuseFactor: how much to parallelize
- DefaultPrecision: inputs, weights, biases

```
my-hls-test/:
build_prj.tcl
firmware
myproject_test.cpp
```

HLS PROJECT

Build HLS project → `vivado_hls -f build_prj.tcl`

```
##################
#    HLS4ML      #
##################
open_project -reset myproject_prj
set_top myproject
add_files firmware/myproject.cpp -cflags "-I[file normalize ../../nnet_utils]"
add_files -tb myproject_test.cpp -cflags "-I[file normalize ../../nnet_utils]"
add_files -tb firmware/weights
open_solution -reset "solution1"
set_part {xcku115-flvf1924-2-i}
create_clock -period 5 -name default
csim_design
csynth_design
cosim_design -trace_level all
export_design -format ip_catalog
exit
```

produces an “IP core” that can be dropped into a full firmware design

NETWORK TUNING: QUANTIZATION

Xilinx Vivado 2017.2
Clock frequency: 200 MHz
FPGA: Xilinx Kintex Ultrascale
(XCKU115-FLVB2104)

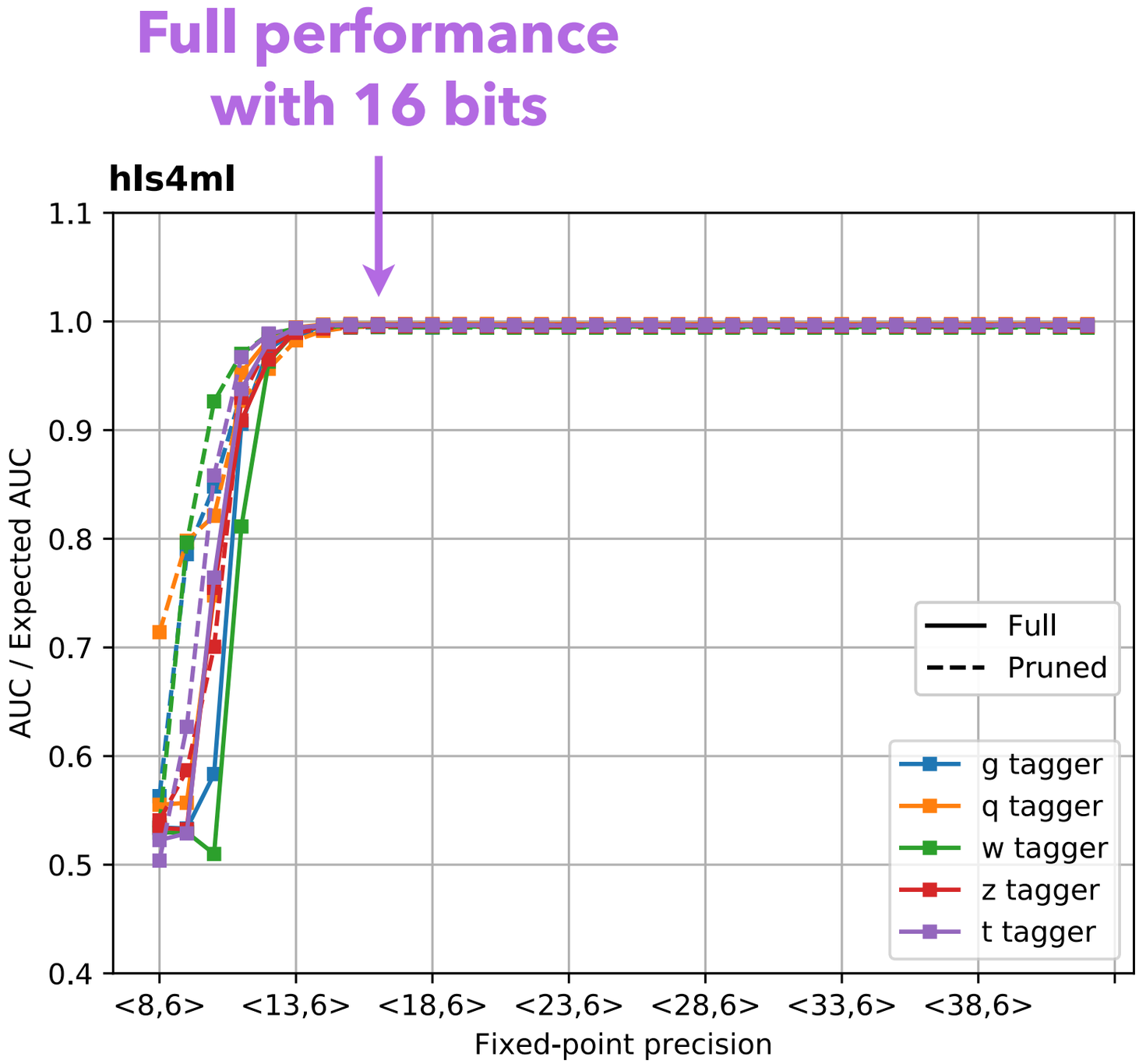
- Scan the bit width until you reach optimal performance

ap_fixed<width,integer>

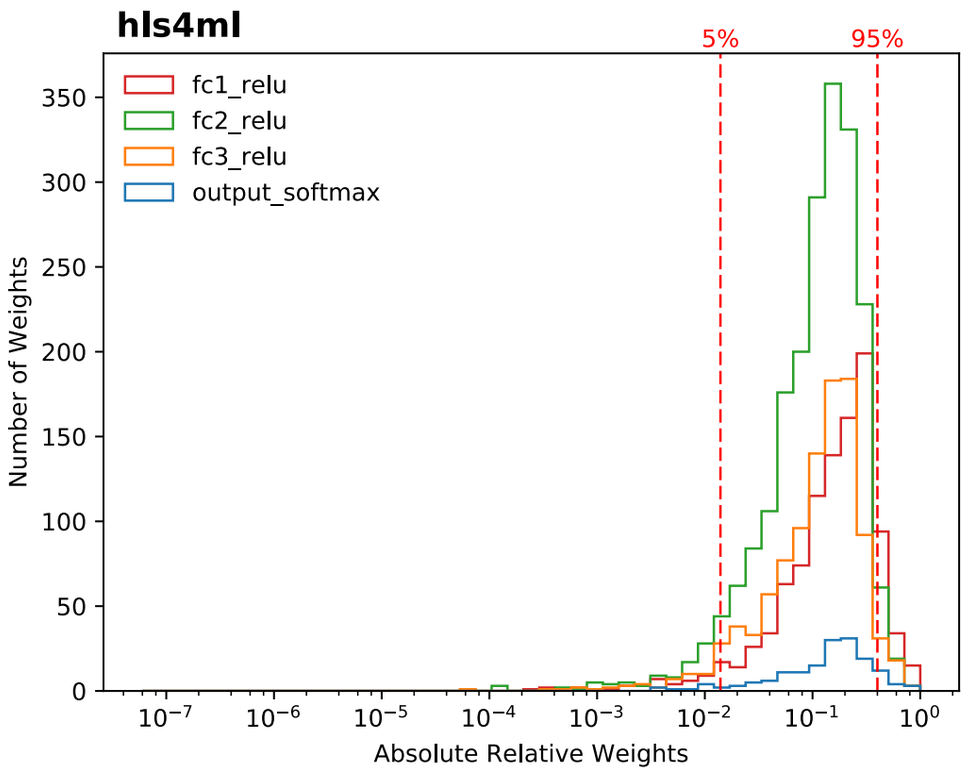
0101.1011101010

integer | fractional

width



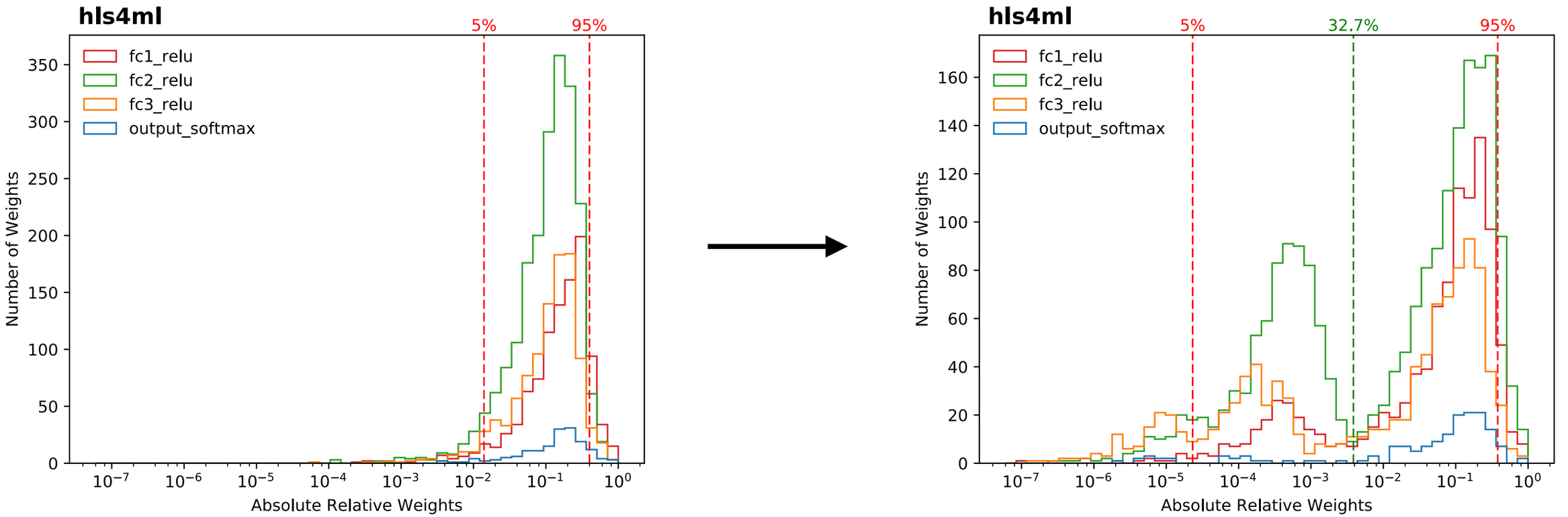
NETWORK TUNING: COMPRESSION



NETWORK TUNING: COMPRESSION

- Train with **L_1 regularization** (down-weights unimportant synapses)

$$L_\lambda(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1 \qquad \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

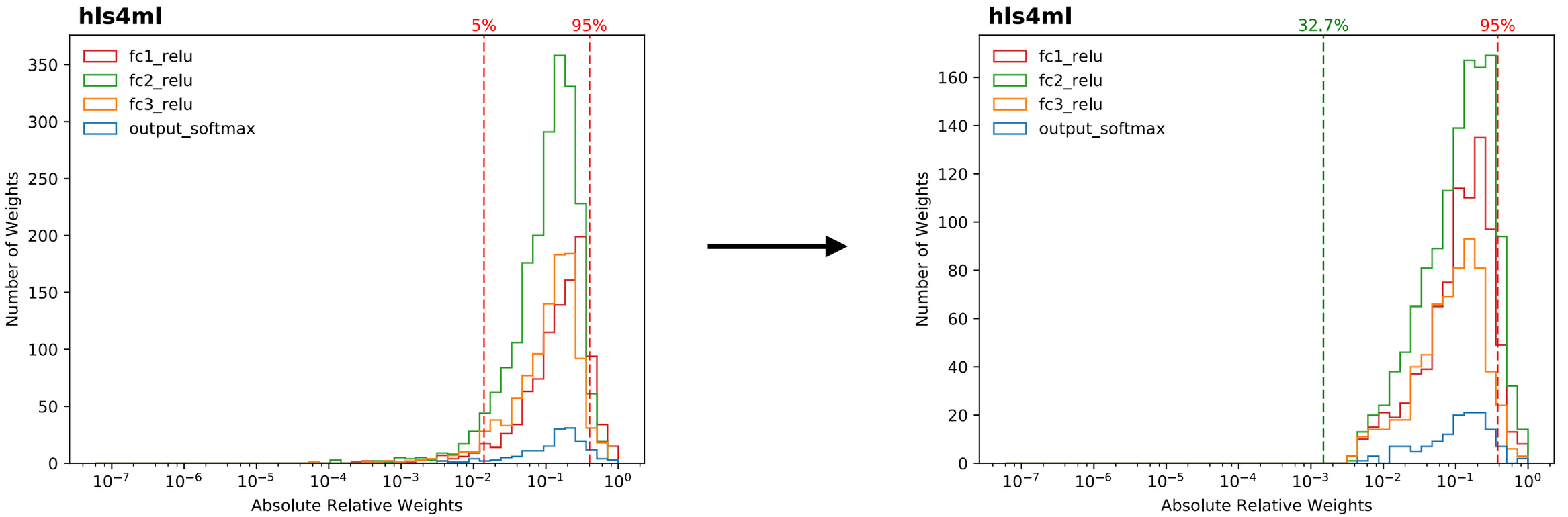


NETWORK TUNING: COMPRESSION

- Train with **L_1 regularization** (down-weights unimportant synapses)

$$L_\lambda(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1 \qquad \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

- Remove **smallest** weights



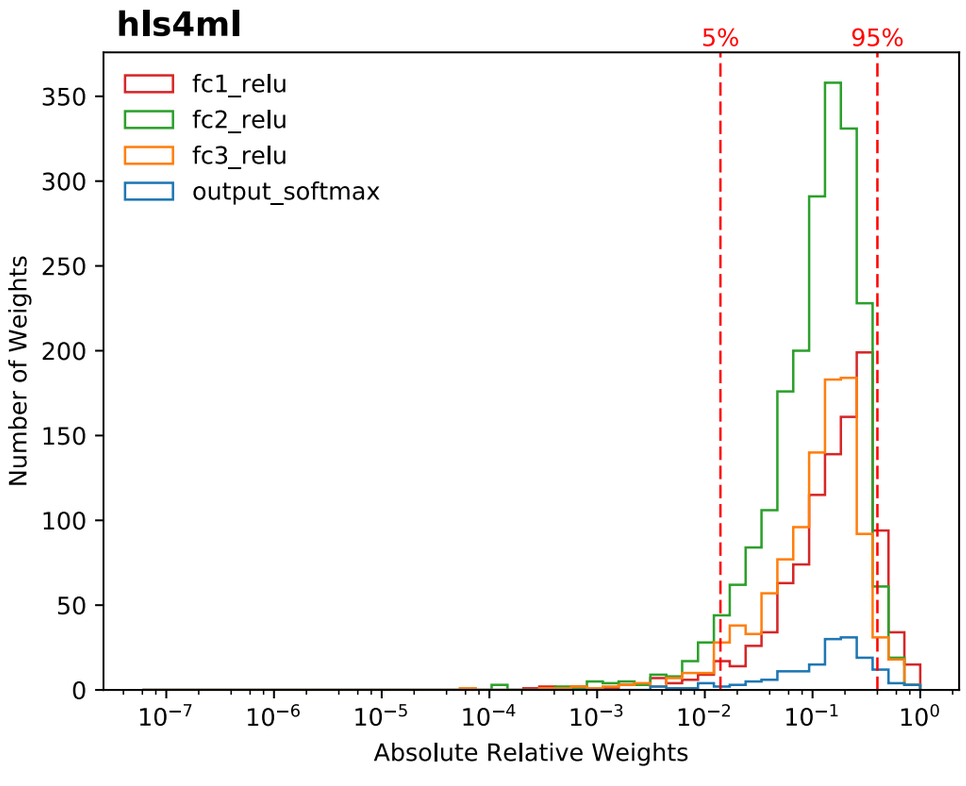
NETWORK TUNING: COMPRESSION

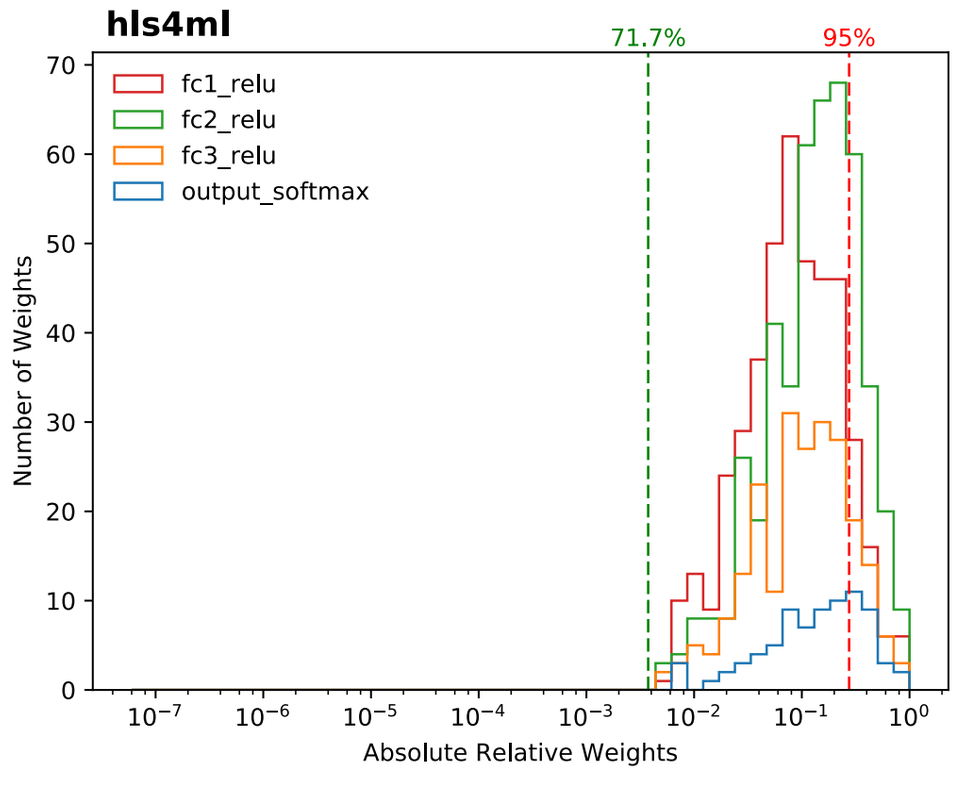
- Train with **L_1 regularization** (down-weights unimportant synapses)

$$L_\lambda(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1 \qquad \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

- Remove **smallest** weights
- Iterate

70% REDUCTION OF WEIGHTS WITH NO LOSS IN PERF.





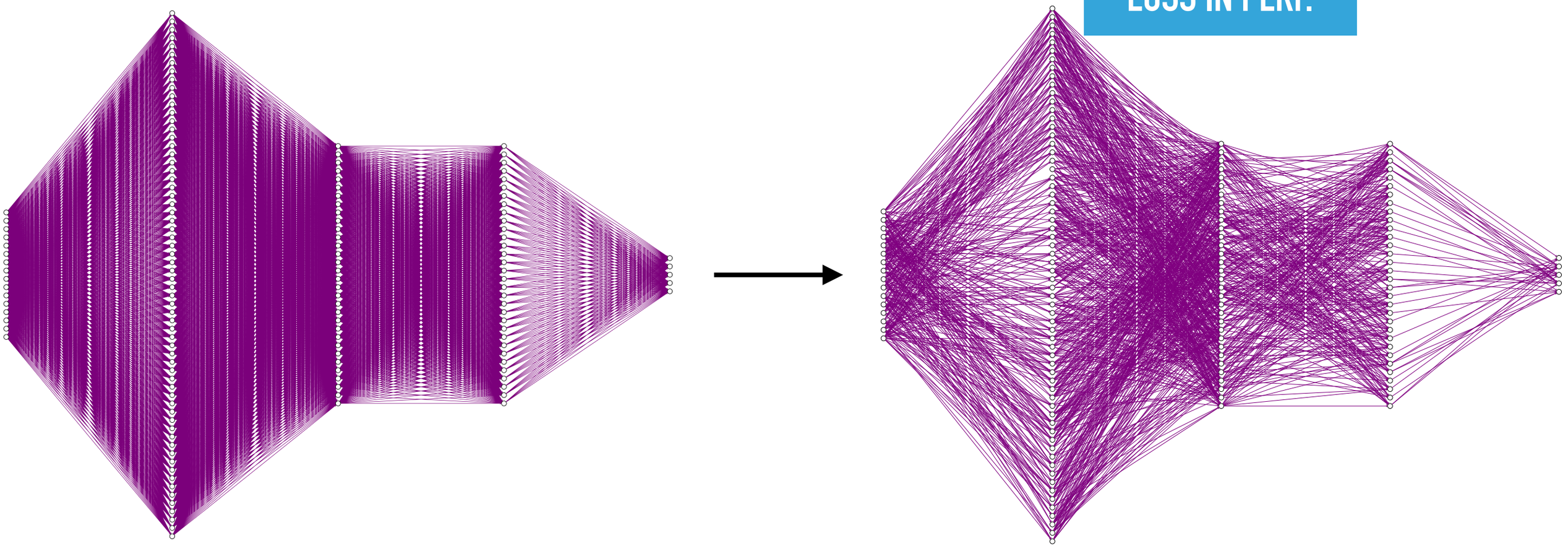
NETWORK TUNING: COMPRESSION

- Train with **L_1 regularization** (down-weights unimportant synapses)

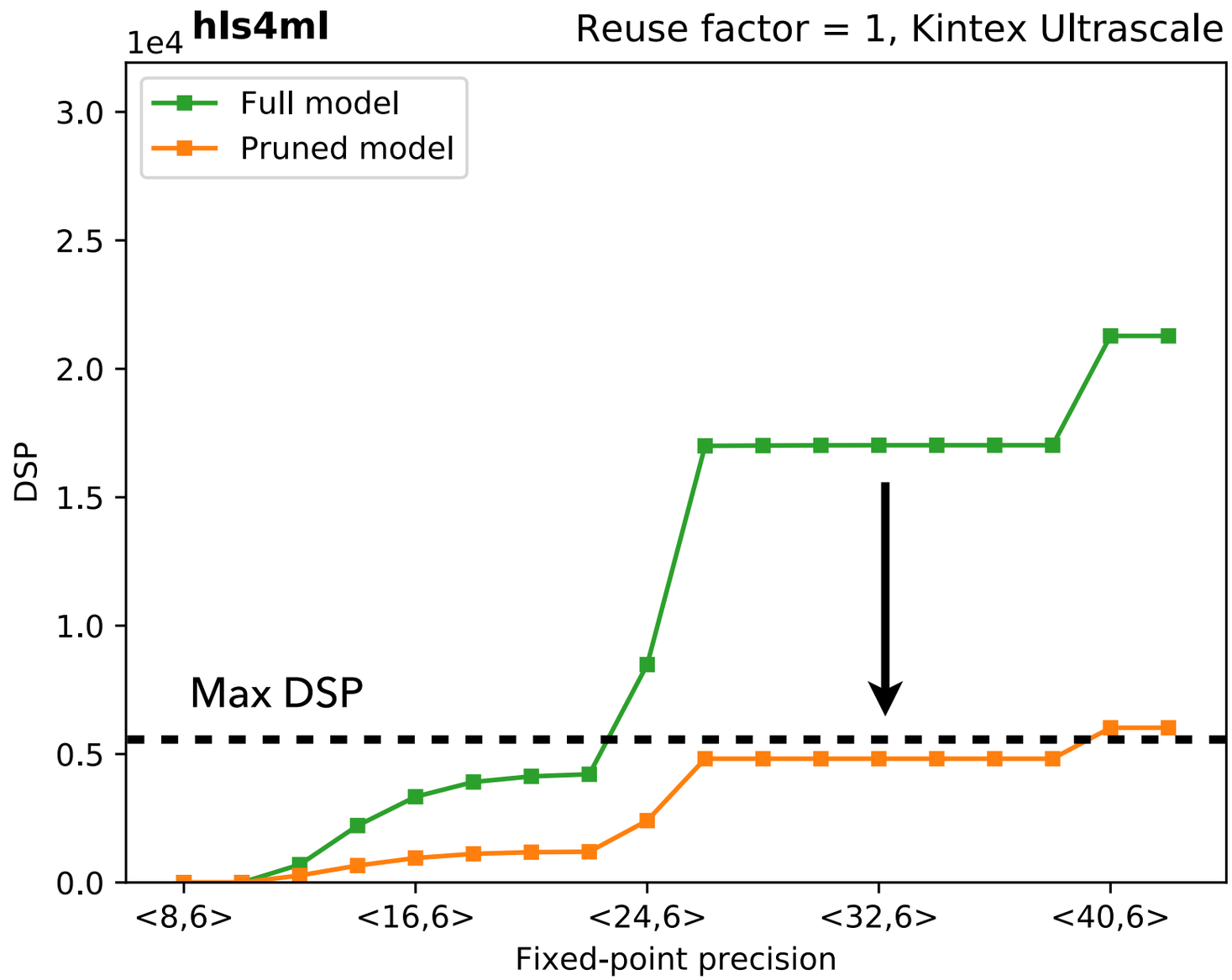
$$L_\lambda(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1 \qquad \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

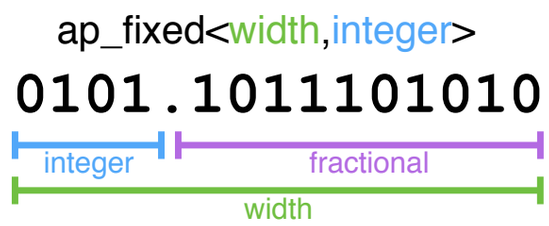
- Remove **smallest** weights
- Iterate

70% REDUCTION OF WEIGHTS WITH NO LOSS IN PERF.

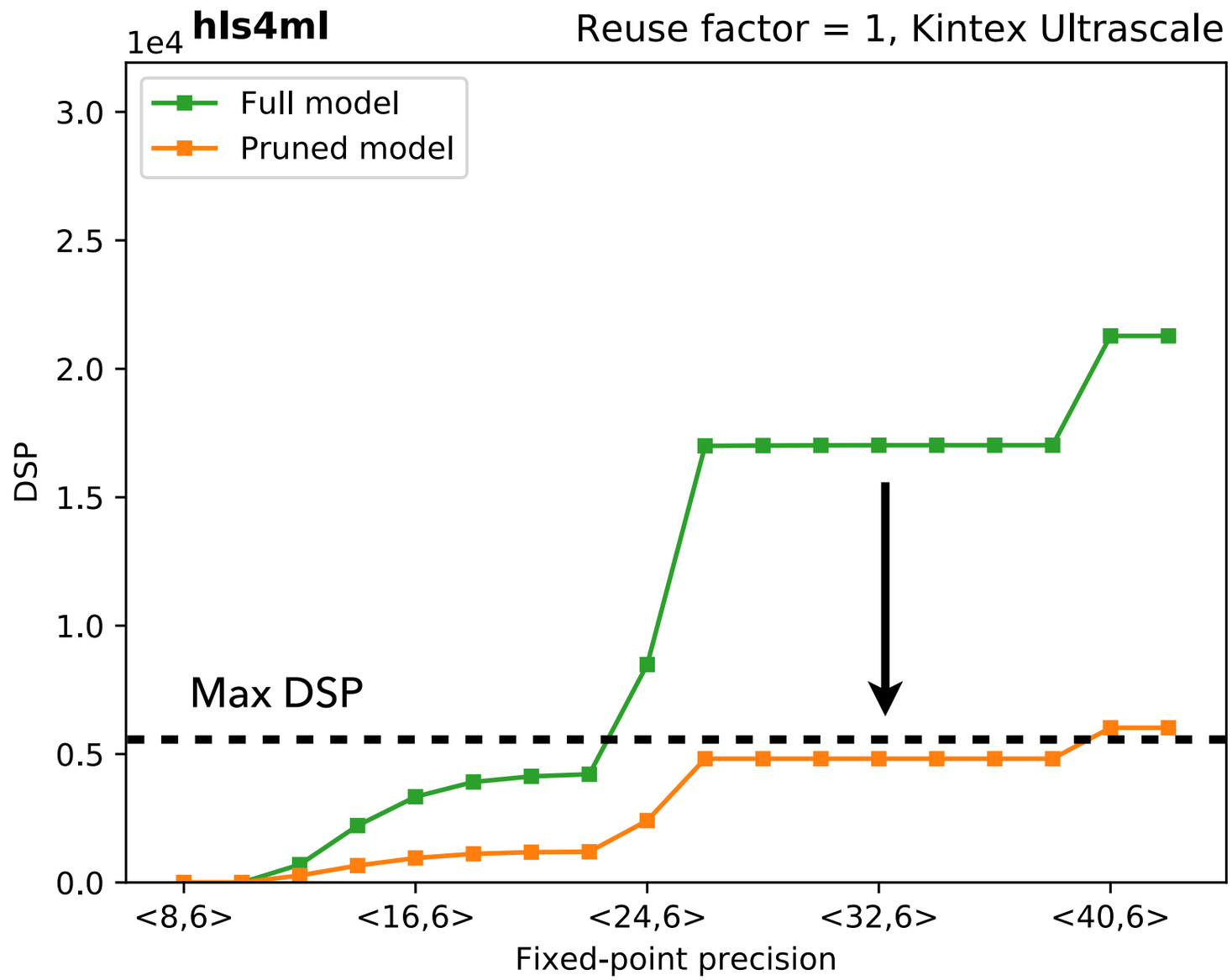


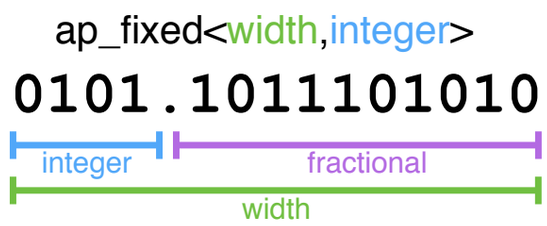
NETWORK TUNING: COMPRESSION & RESOURCES





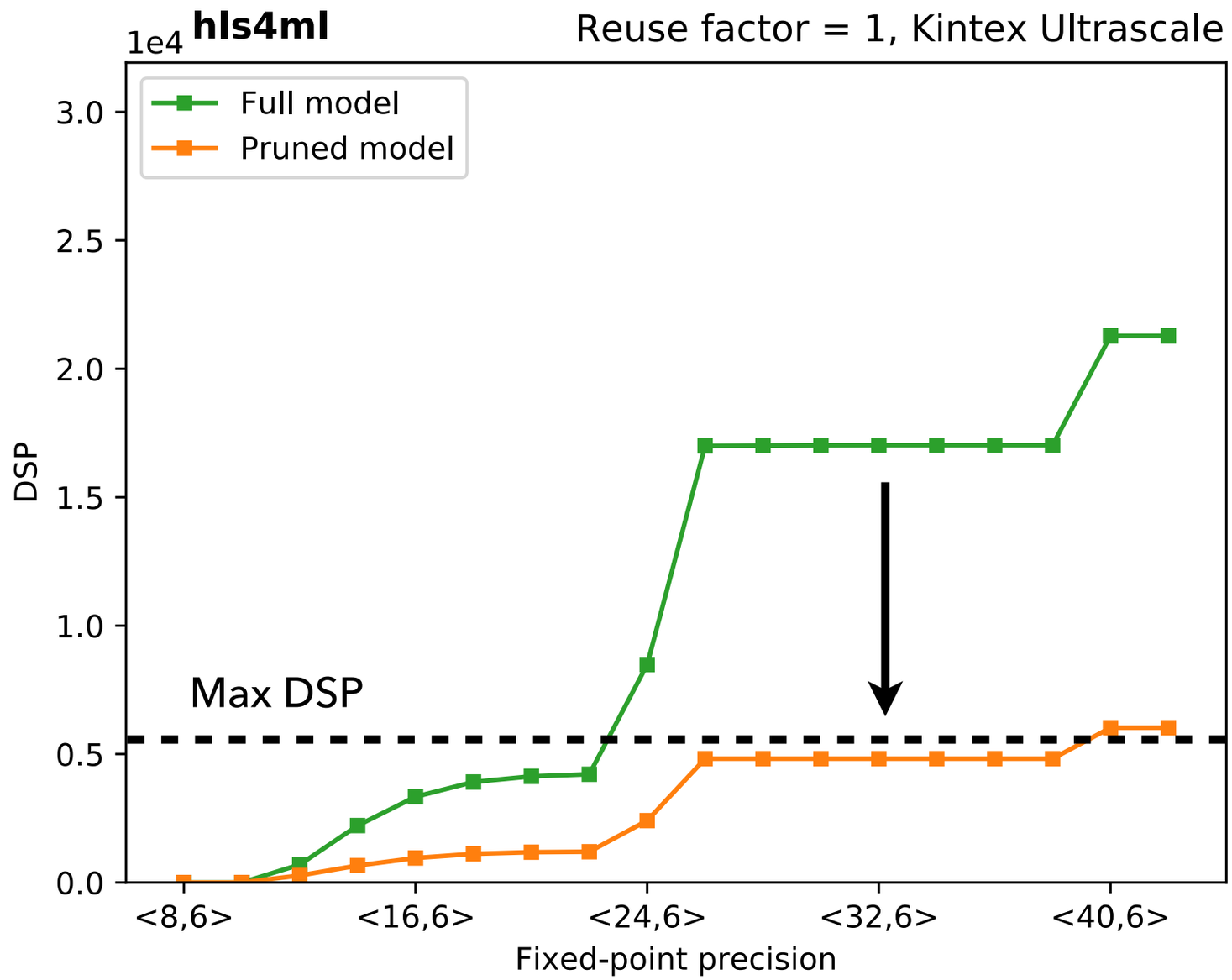
NETWORK TUNING: COMPRESSION & RESOURCES

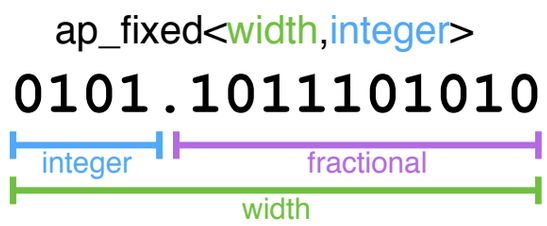




- Big reduction in DSPs (multipliers) with compression

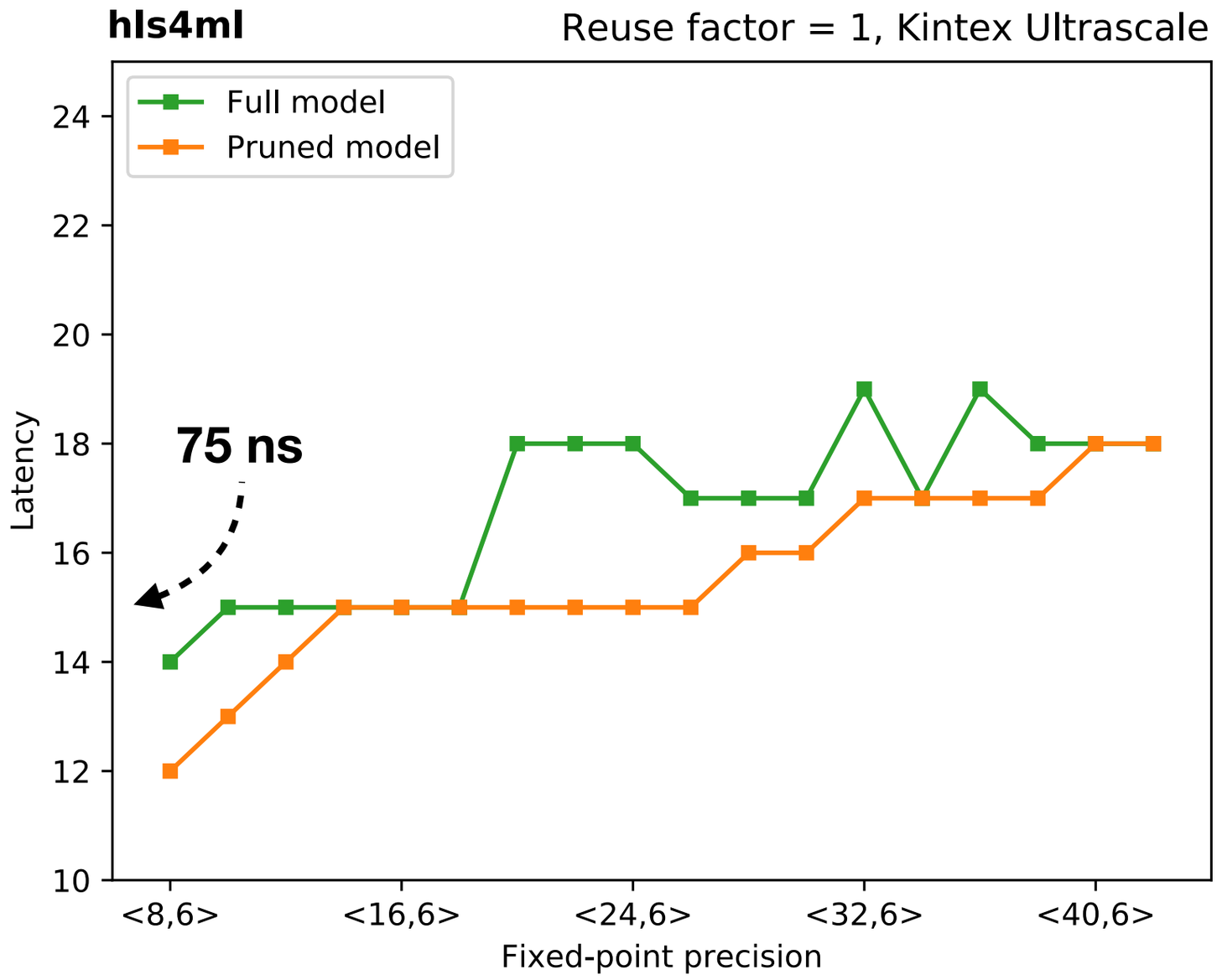
NETWORK TUNING: COMPRESSION & RESOURCES

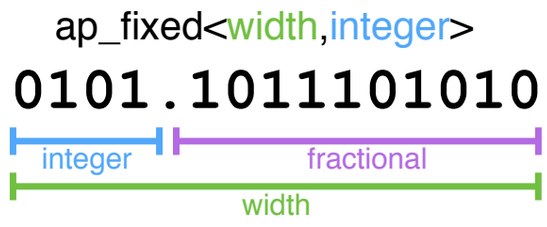




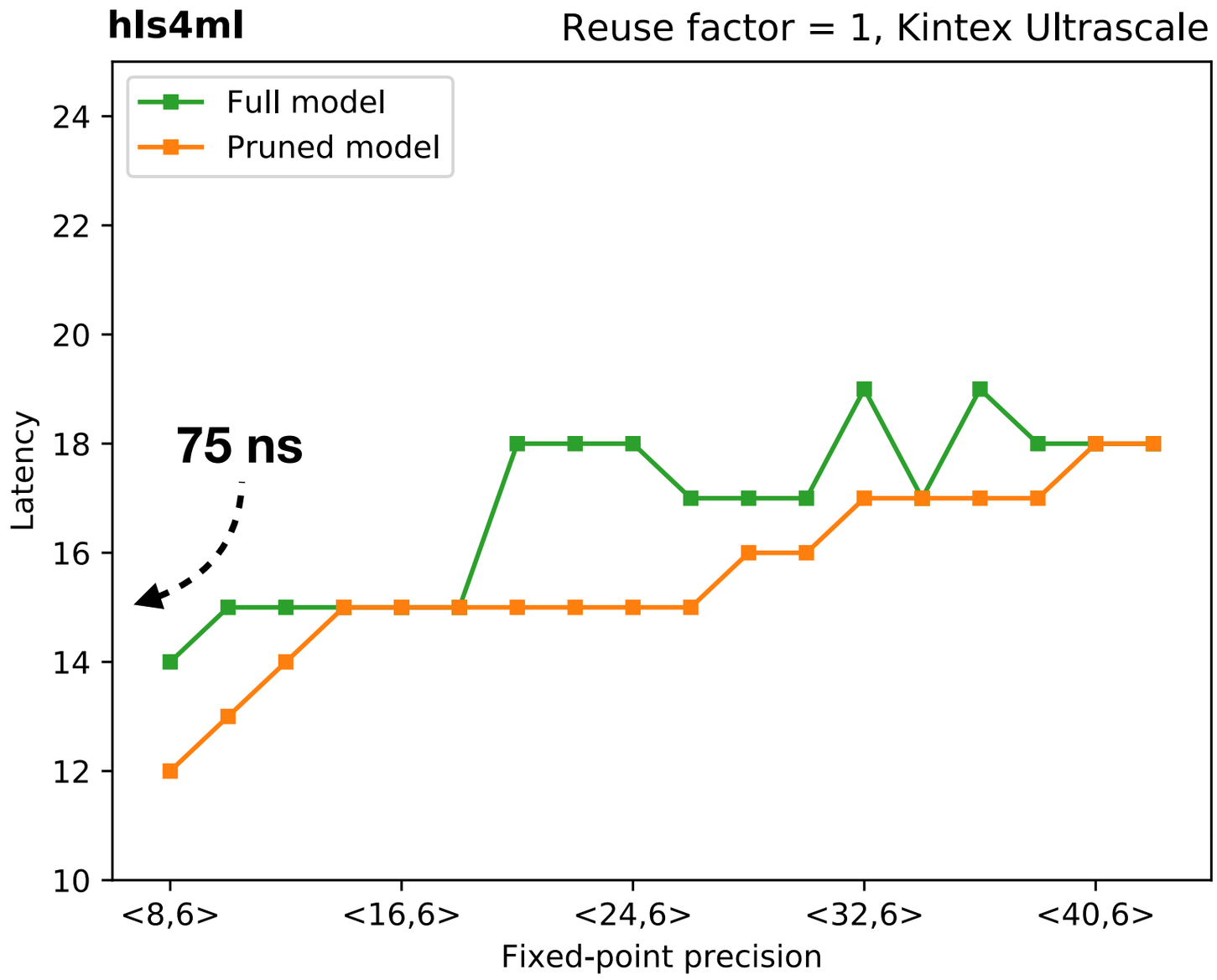
- Big reduction in DSPs (multipliers) with compression
- Easily fits on 1 FPGA **after compression**

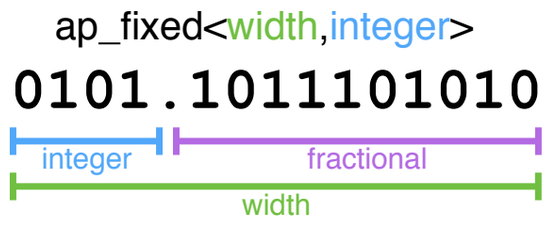
NETWORK TUNING: COMPRESSION & LATENCY





NETWORK TUNING: COMPRESSION & LATENCY

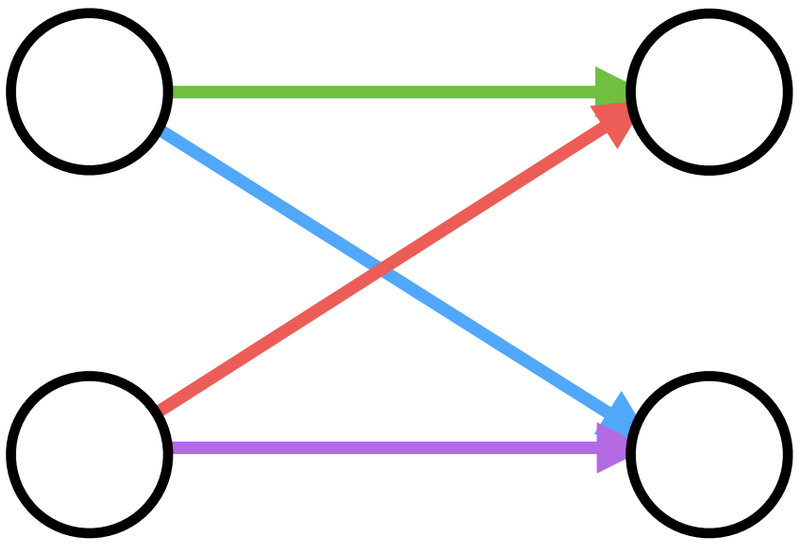


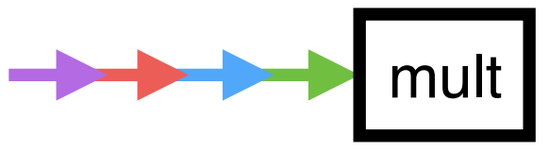


- ~15 clocks @ 200 MHz = **75 ns** inference!

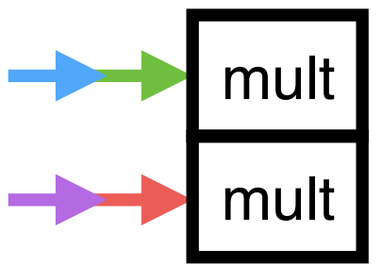
NETWORK TUNING: PARALLELIZATION

- Reuse Factor: how much to parallelize

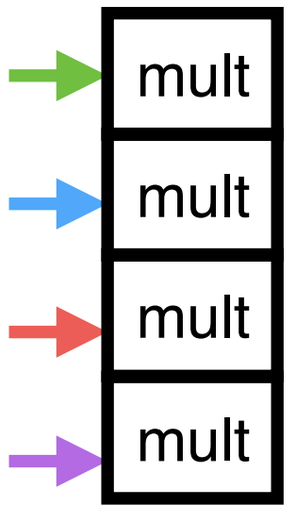




reuse = 4
use 1 multiplier 4 times



reuse = 2
use 2 multipliers 2 times each

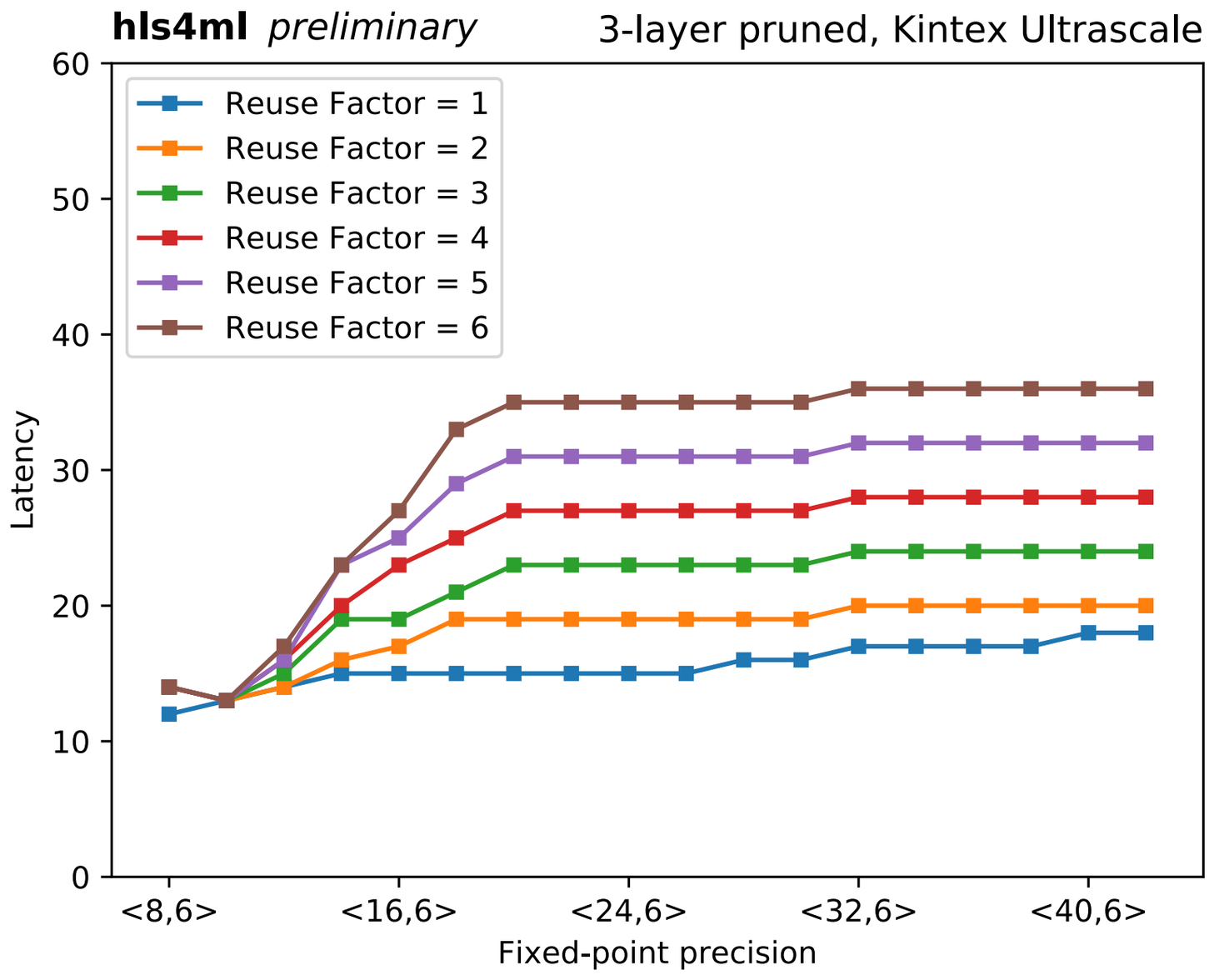


reuse = 1
use 4 multipliers 1 time each

Related to the Initiation Interval = when new inputs can be introduced to the algo.

NETWORK TUNING: PARALLELIZATION & TIMING

- **Increasing** reuse factor, **increases** latency



~35 clocks
@ 200 MHz
= 175 ns

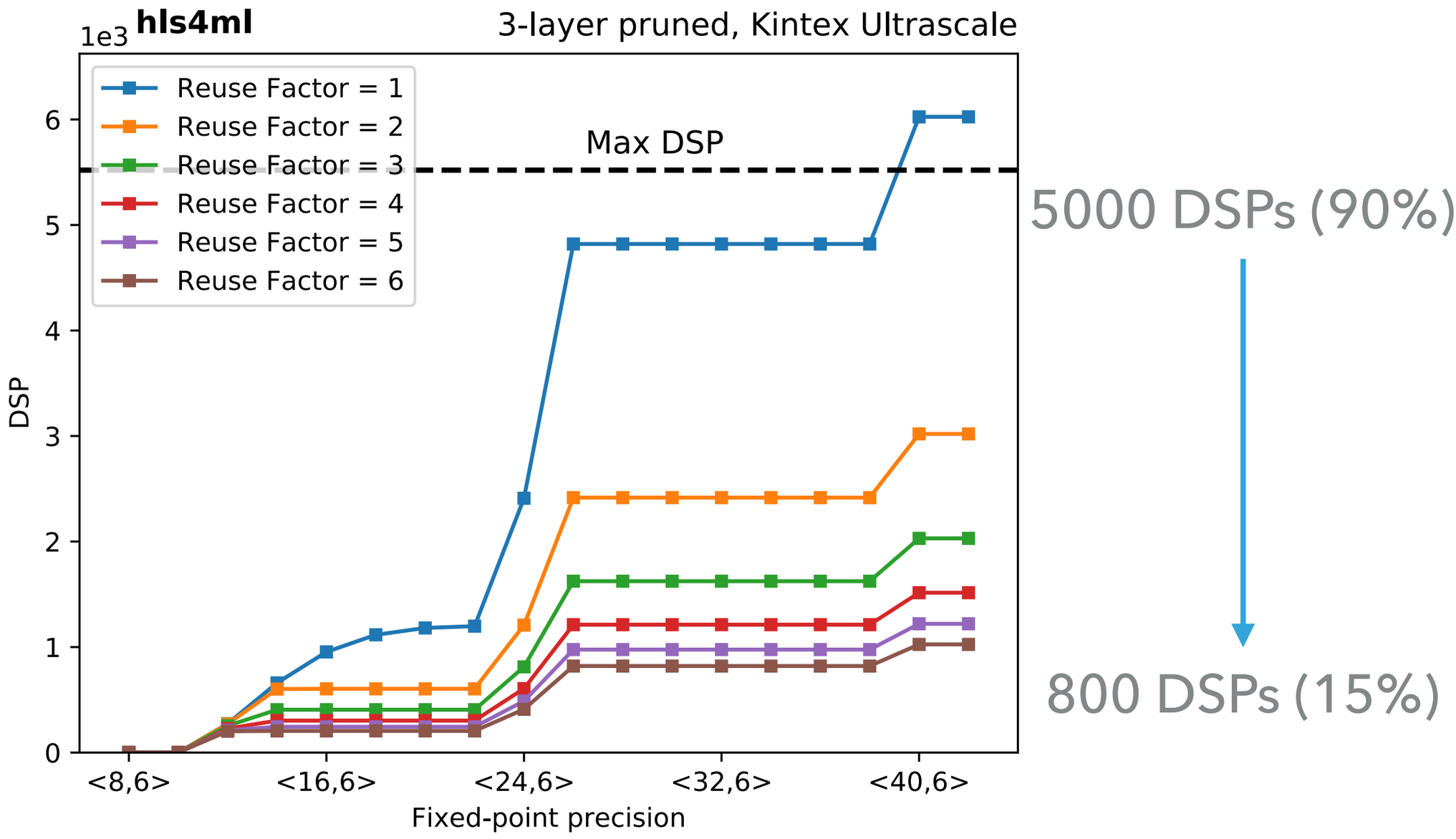
~15 clocks
@ 200 MHz
= 75 ns

For low-latency, small reuse factor, inference in O(100 ns)!

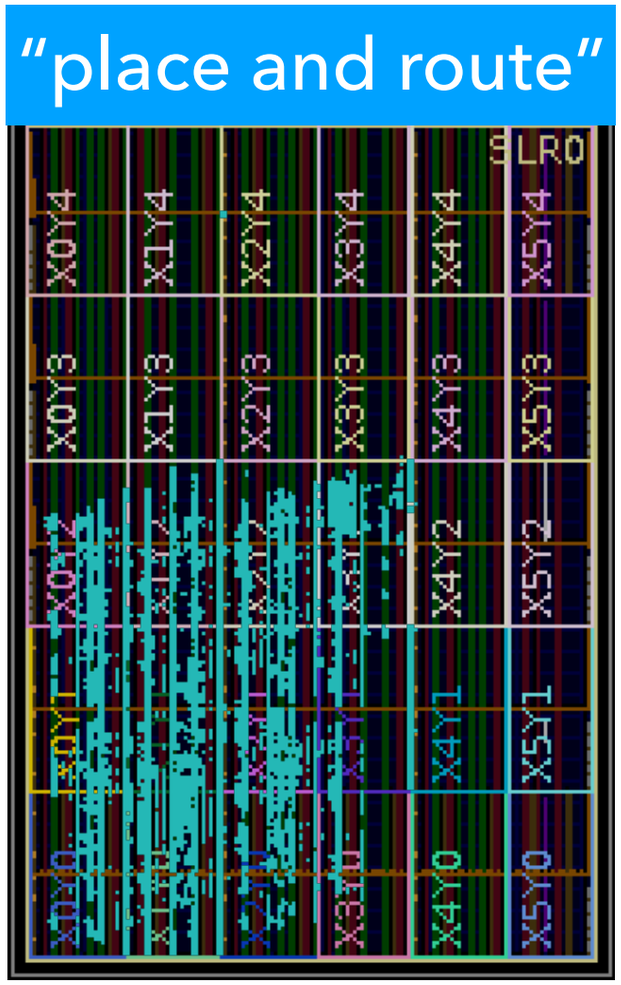
What if we have O(ms)? Can go to **bigger networks!**

NETWORK TUNING: PARALLELIZATION & RESOURCES

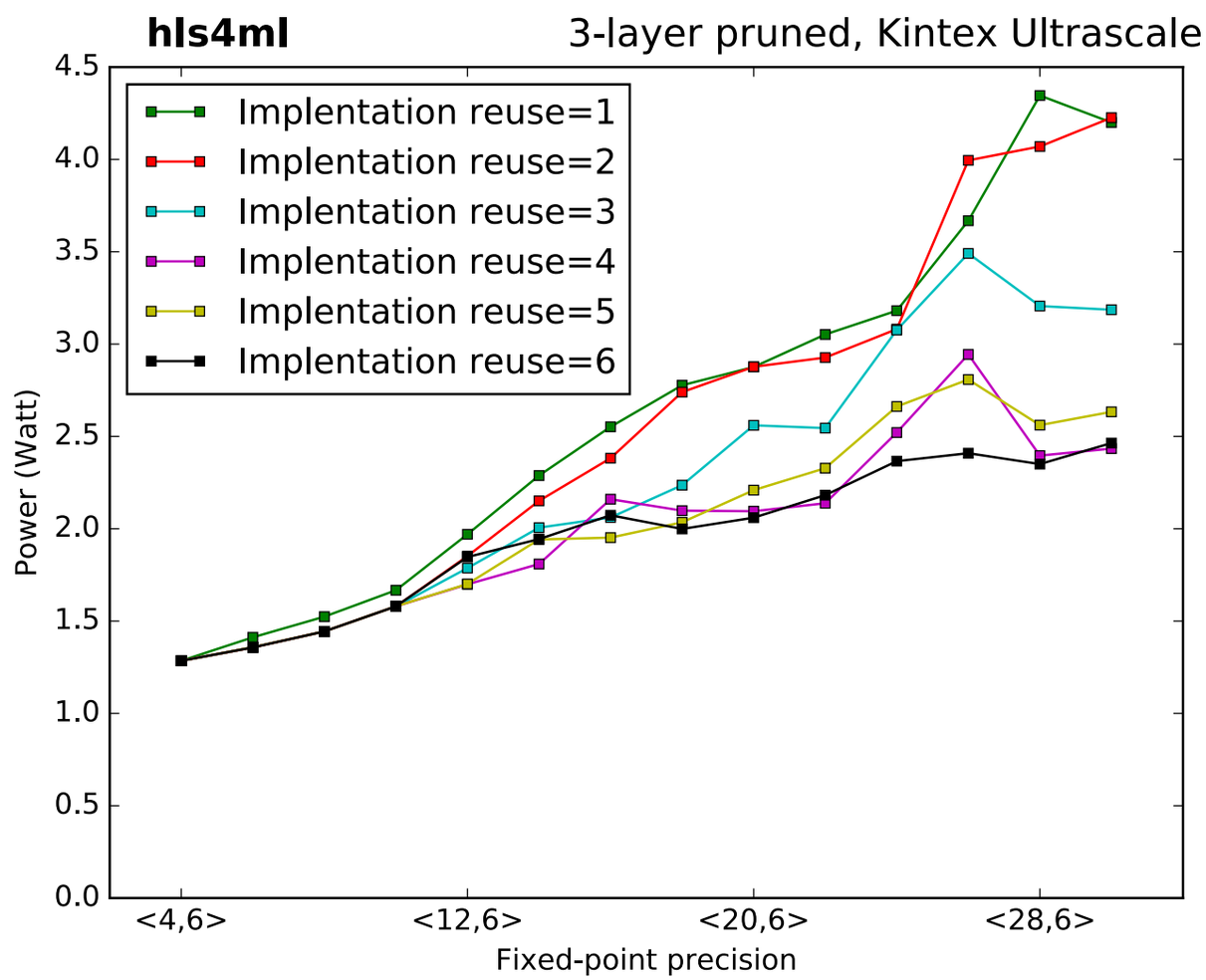
- **Increasing** reuse factor, **decreases** resources



IMPLEMENTING THE DESIGN



- How optimal is the HLS design (vs. RTL)?
- For DSPs, HLS seems close to "back-of-the-envelope" optimal estimate
- HLS is good for quickly getting a conservative estimate of resources
- Power decreases as throughput is decreased (by increasing reuse factor)



MACHINE LEARNING IN FPGAS

MACHINE LEARNING IN FPGAS

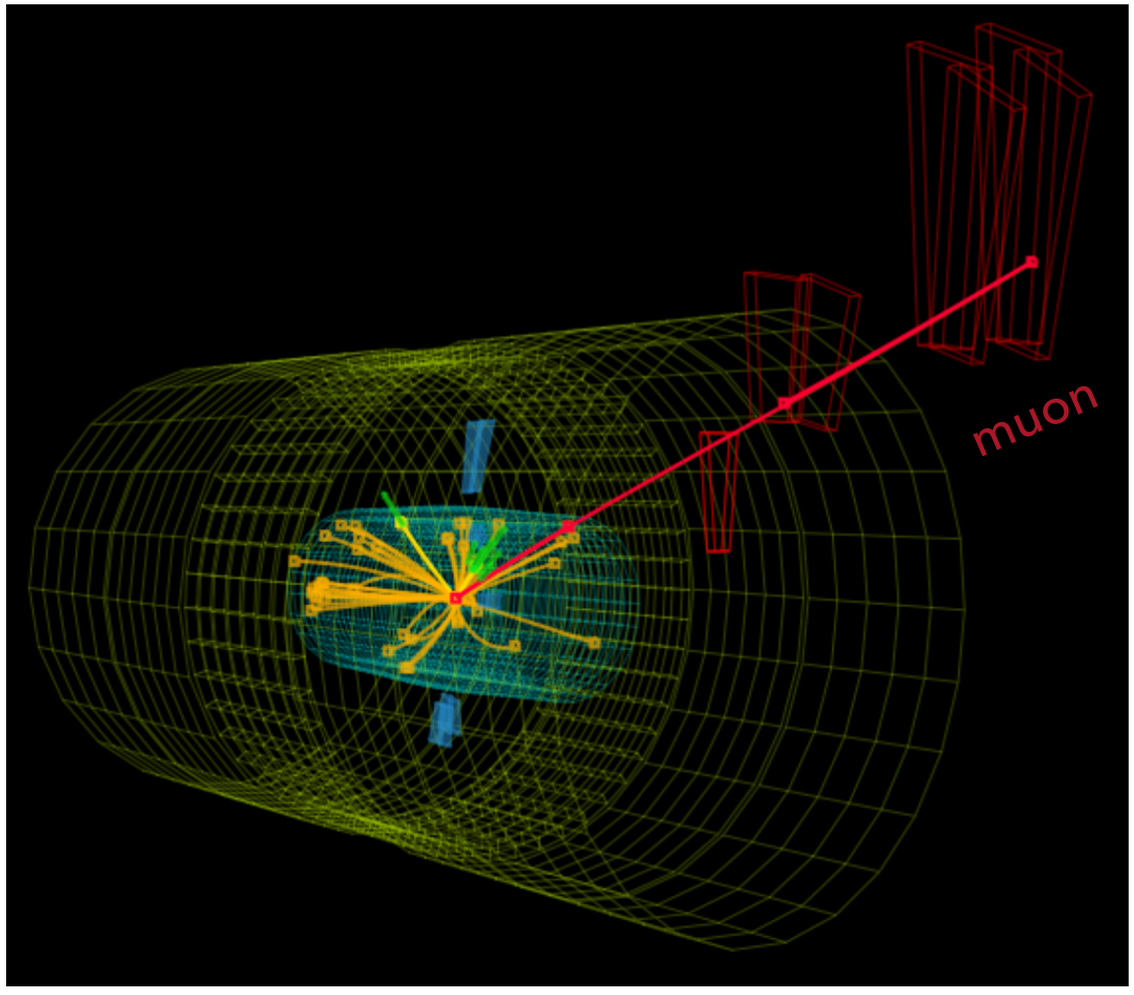
- Inference of ML algorithms possible in **O(100 ns)** on **1 FPGA** with **hls4ml**!

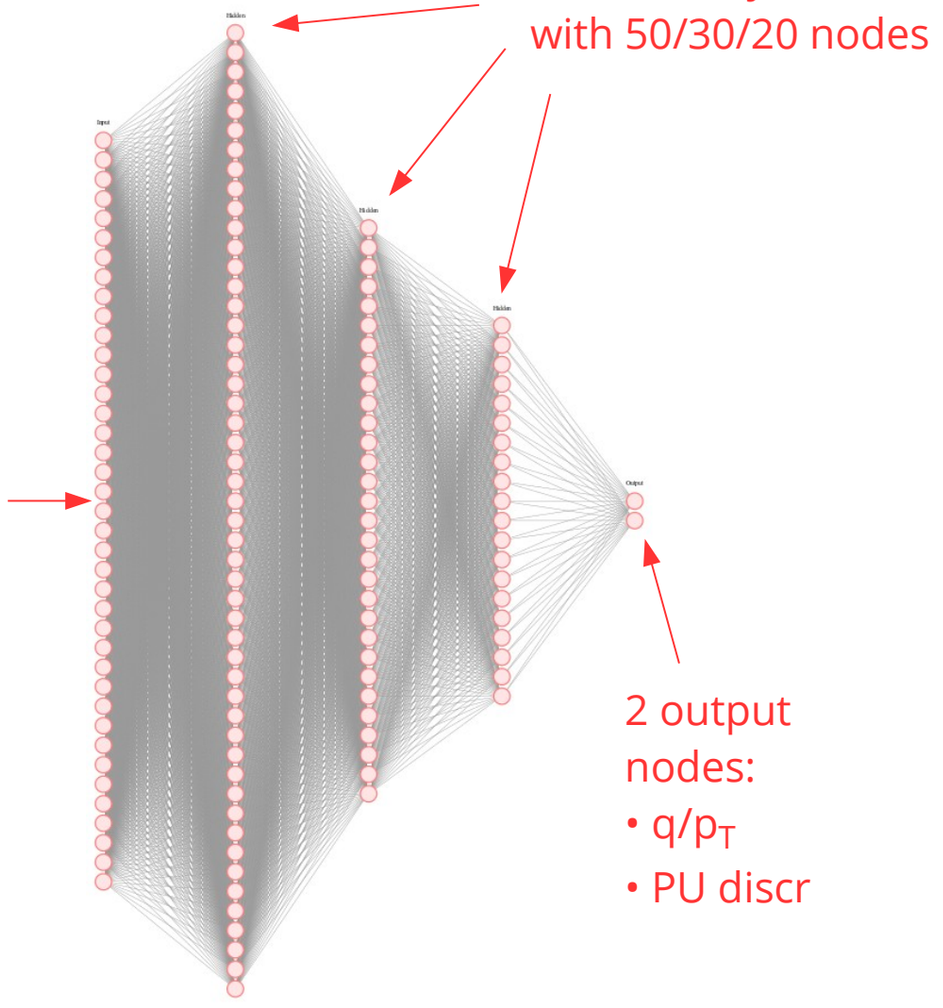
MACHINE LEARNING IN FPGAS

- Inference of ML algorithms possible in **O(100 ns)** on **1 FPGA** with **hls4ml**!
  - Applications across CMS, ATLAS, DUNE, and accelerator controls

MACHINE LEARNING IN FPGAS

- Inference of ML algorithms possible in **O(100 ns)** on **1 FPGA** with **hls4ml**!
  - Applications across CMS, ATLAS, DUNE, and accelerator controls
  - E.g. muon p_T determination in the CMS endcap with a DNN: **runs in 160 ns** on an FPGA and **reduces** the **fake muon rate** by **up to 80%**





SUMMARY AND OUTLOOK

SUMMARY AND OUTLOOK

- Deep learning algorithms have proven to be better than traditional algorithms in HEP for **jet tagging** and much more

SUMMARY AND OUTLOOK

- Deep learning algorithms have proven to be better than traditional algorithms in HEP for **jet tagging** and much more
- With FPGAs, machine learning can be done **quickly** and **efficiently**

SUMMARY AND OUTLOOK

- Deep learning algorithms have proven to be better than traditional algorithms in HEP for **jet tagging** and much more
- With FPGAs, machine learning can be done **quickly** and **efficiently**
- Applications of ML in FPGAs are far-reaching across HEP!
  - **Deep learning** for the **trigger**
  - ML-based **reconstruction** aided by **co-processors**

SUMMARY AND OUTLOOK

- Deep learning algorithms have proven to be better than traditional algorithms in HEP for **jet tagging** and much more
- With FPGAs, machine learning can be done **quickly** and **efficiently**
- Applications of ML in FPGAs are far-reaching across HEP!
  - **Deep learning** for the **trigger**
  - ML-based **reconstruction** aided by **co-processors**
- Deep learning + specialized hardware will allow us to **do more** with **less resources** to enable **new discoveries** at the energy and other frontiers

Beyond LHC

Ex: self-driving cars

A single self-driving test vehicle can produce **~ 30 TB/day**

There are over **250 million cars** on the road in the US alone

If **< 1% replaced by autonomous vehicles** by 2020

→ HUGE amount of data generated, not manageable by central servers!

Need **edge computing architectures**, low power and small in size to run powerful data analytics programs oboard

NB: latency matters! even a few milliseconds of delay can result in an accident!

The stakes are too high to wait the answer from a distant cloud server.

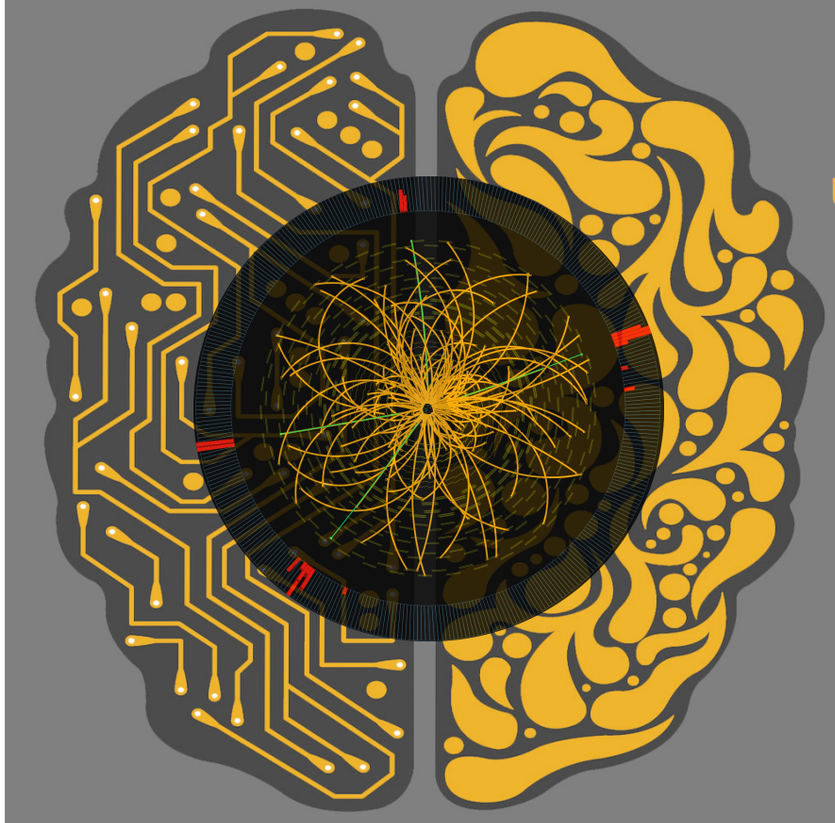
~~FML~~ FAST ML WORKSHOP

Fast Machine Learning

September 10-13, 2019 at Fermilab

Sept. 10-11
IRIS-HEP Blueprint Meeting

Sept. 12-13
Developer Bootcamp



Accelerating ML in science:

Ultrafast on-detector inference and real-time systems

Acceleration as-a-service

Hardware platforms

Coprocessor technologies (CPU/GPU/TPU/FPGAs)

Distributed learning

Local Organization:
Gabriele Benelli (Brown U.)
Javier Duarte (Fermilab)
Lindsey Gray (Fermilab)
Mia Liu (Fermilab)
Kevin Pedro (Fermilab)
Alexx Perloff (CU Boulder)
Zhenbin Wu (U. Illinois Chicago)

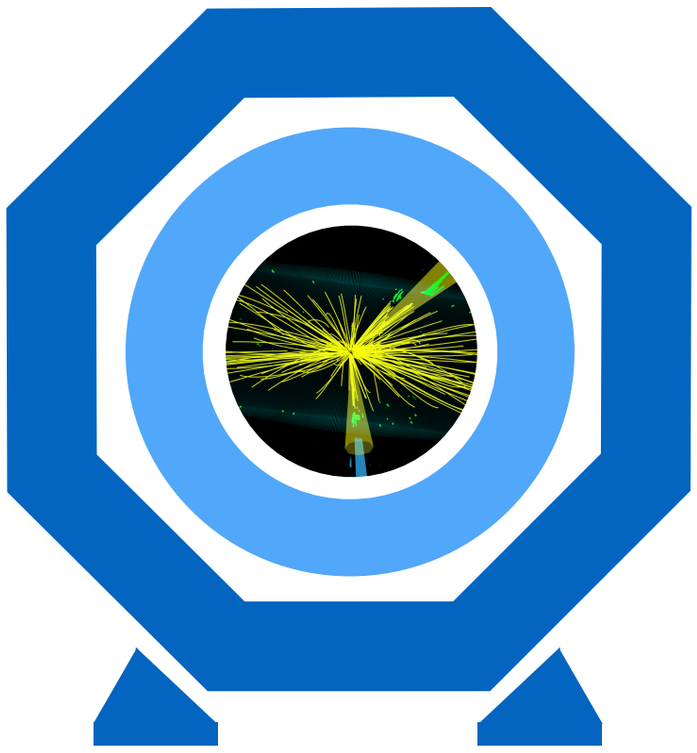
Scientific Organization:
Phil Harris (MIT)
Burt Holzman (Fermilab)
Shih-Chieh Hsu (U. Washington)
Sergo Jindariani (Fermilab)
Maurizio Pierini (CERN)
Mark Neubauer (U. Illinois Urbana-Champaign)
Nhan Tran (Fermilab)

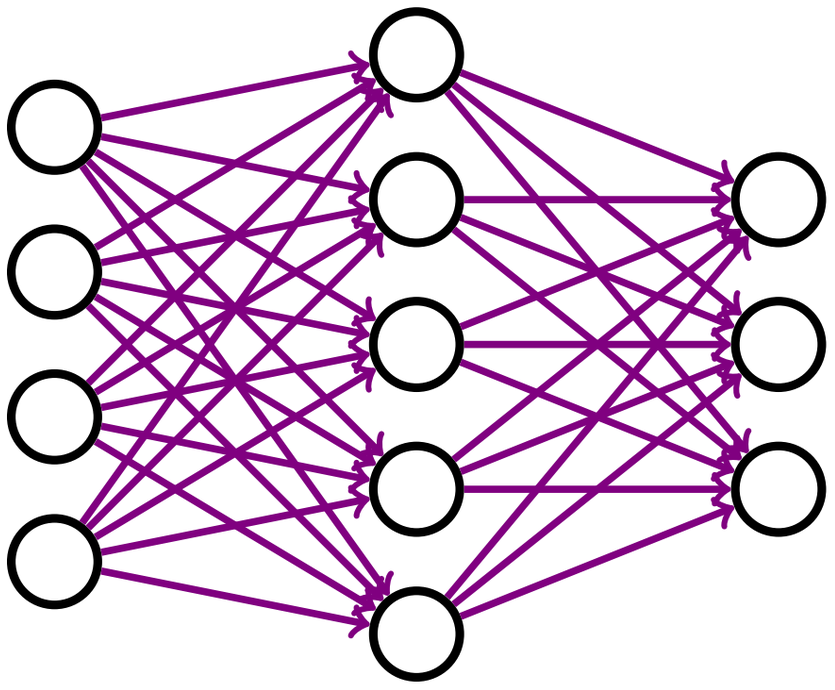
https://indico.cern.ch/e/FML

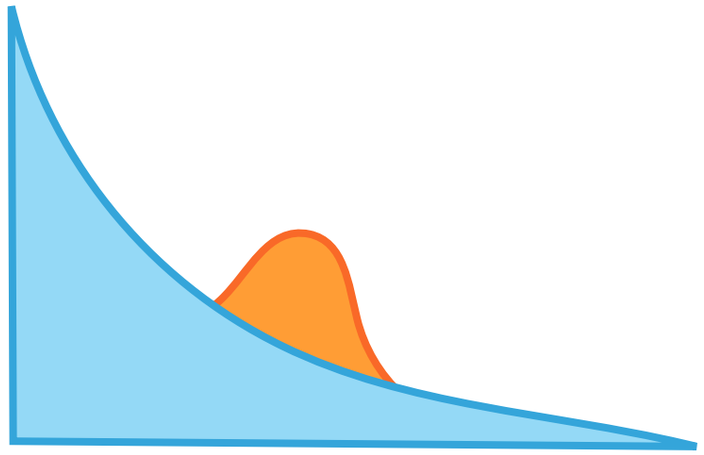
TUTORIALS AND LINKS

TUTORIALS AND LINKS

- hls4ml tool: https://github.com/FPGA4HEP/hls4ml/tree/paris
- SDAccel project: https://github.com/FPGA4HEP/hls4ml_c/tree/paris
- Course material: https://github.com/FPGA4HEP/course_material/tree/paris
  - Part 1: https://github.com/FPGA4HEP/course_material/blob/paris/part1_hls4ml_intro.md
    - Map out resources on FPGA vs reuse factor and precision
  - Part 2: https://github.com/FPGA4HEP/course_material/blob/paris/part2_aws_sdaccel.md
    - Run on a real (cloud) FPGA through AWS (f1.2xlarge instance)
  - (Optional) Part 3: https://github.com/FPGA4HEP/course_material/blob/paris/part3_model_compression.md
    - Compress a neural network by L1 regularization + pruning







JAVIER DUARTE

JULY 16, 2019

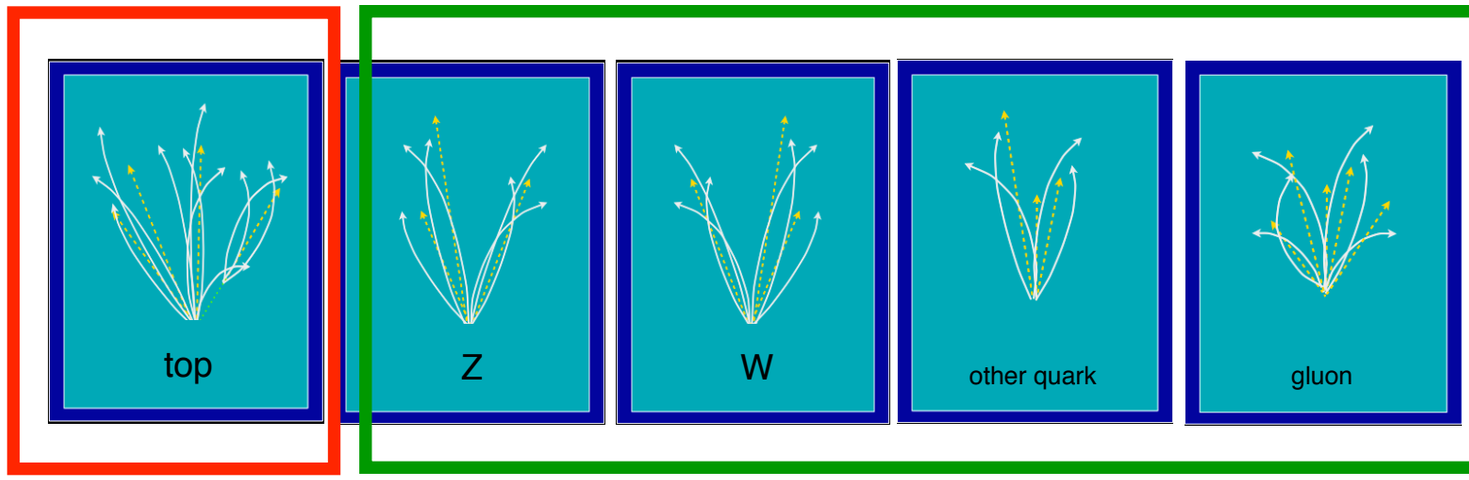
INSTITUT PASCAL

TUTORIAL PART 1: HLS4ML & FPGA RESOURCES

1-layer Dense NN

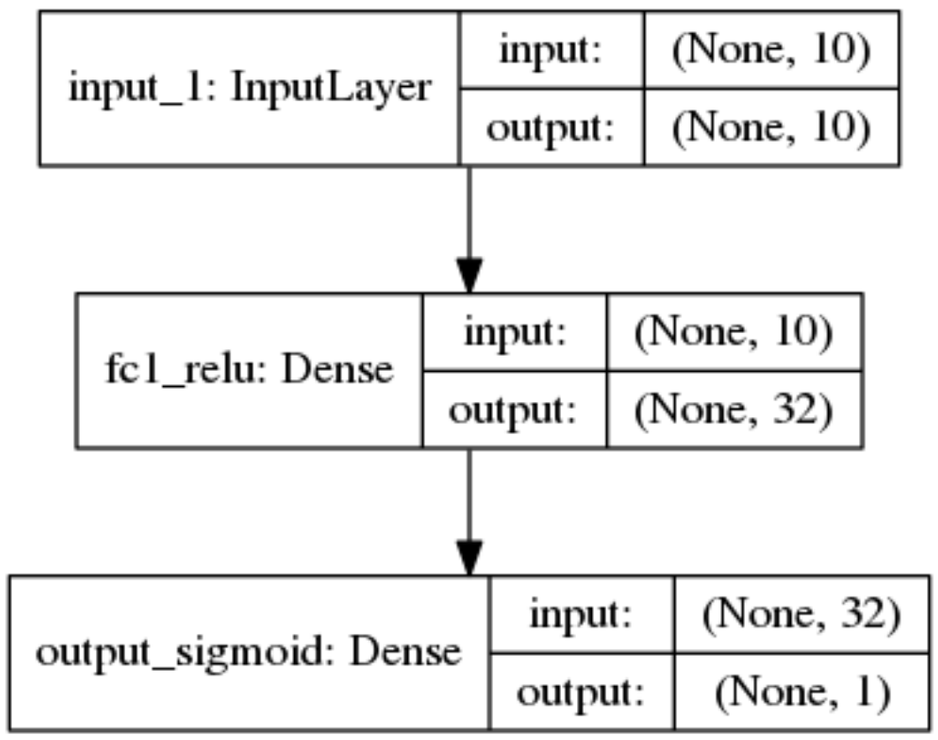
Simple classification task: discriminate top-quark jets from the others

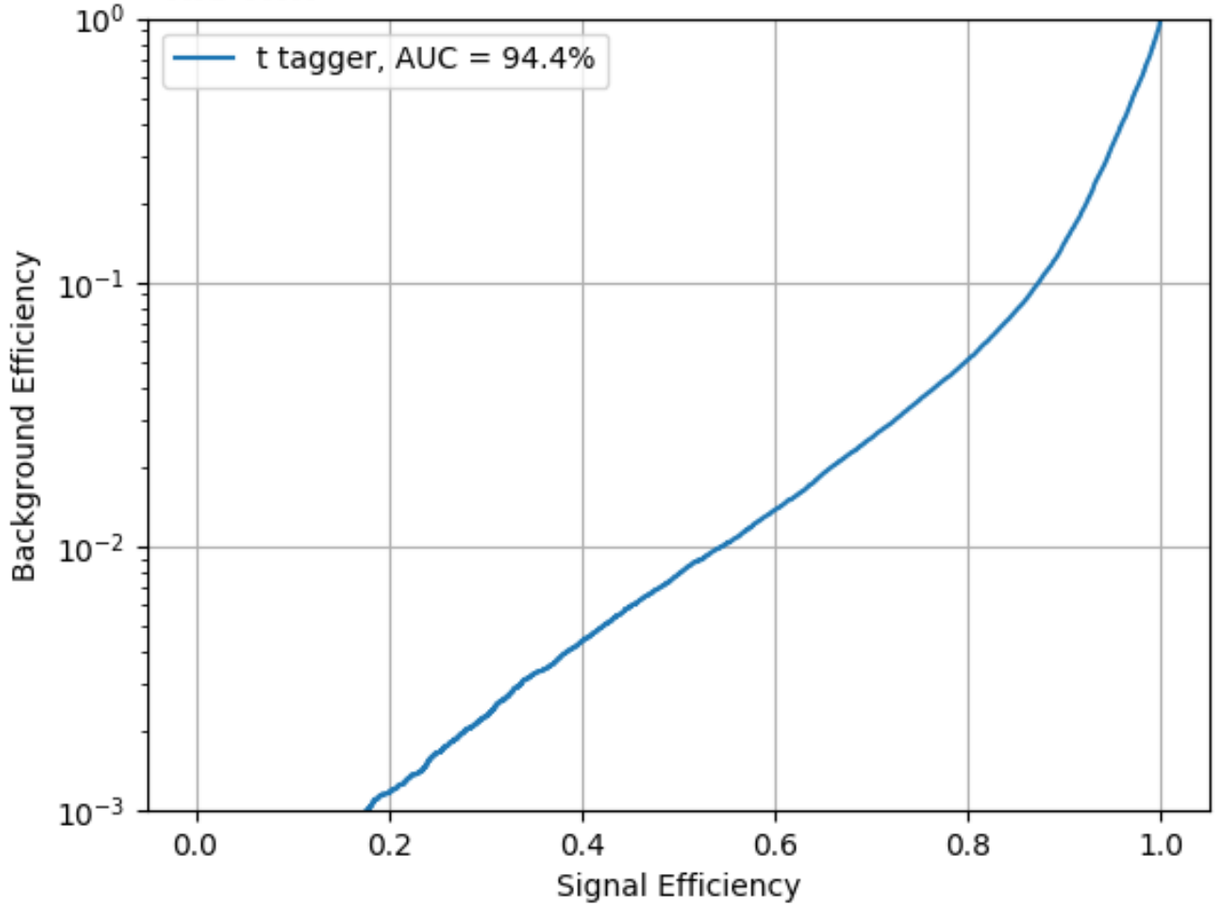
signal:
top-quark jets



background:
all other jets

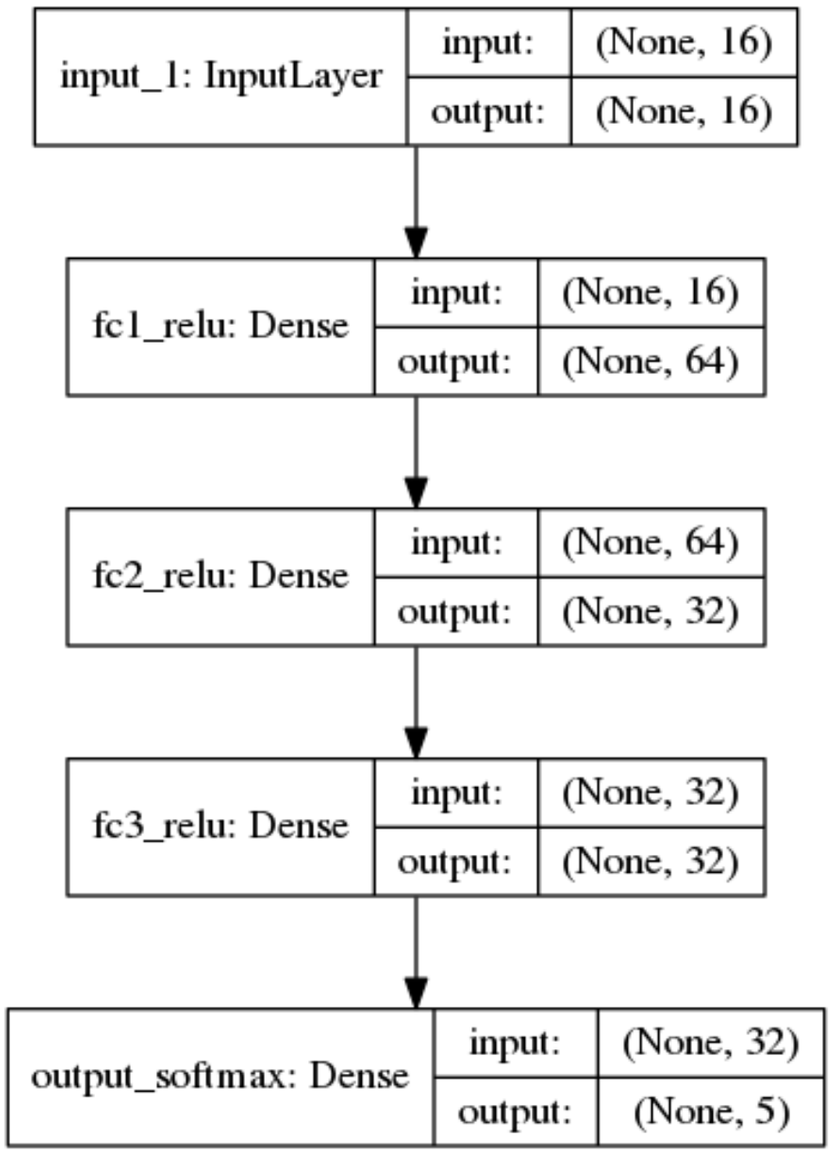
hls4ml

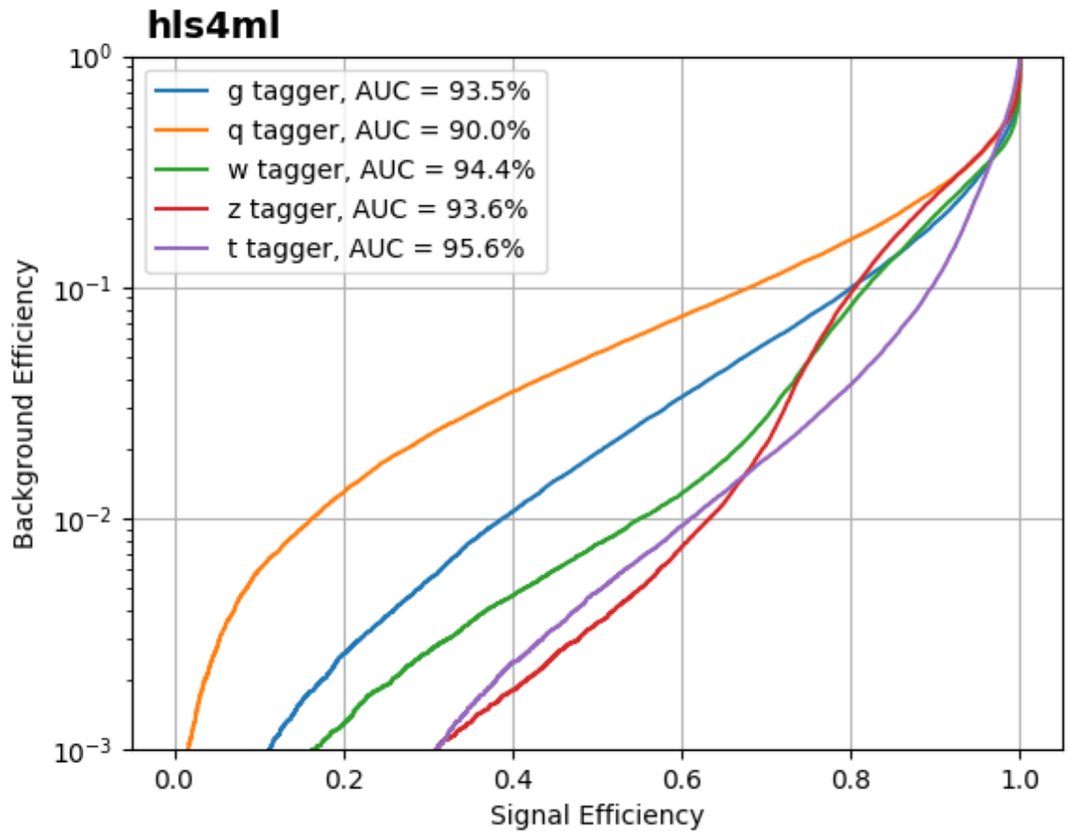


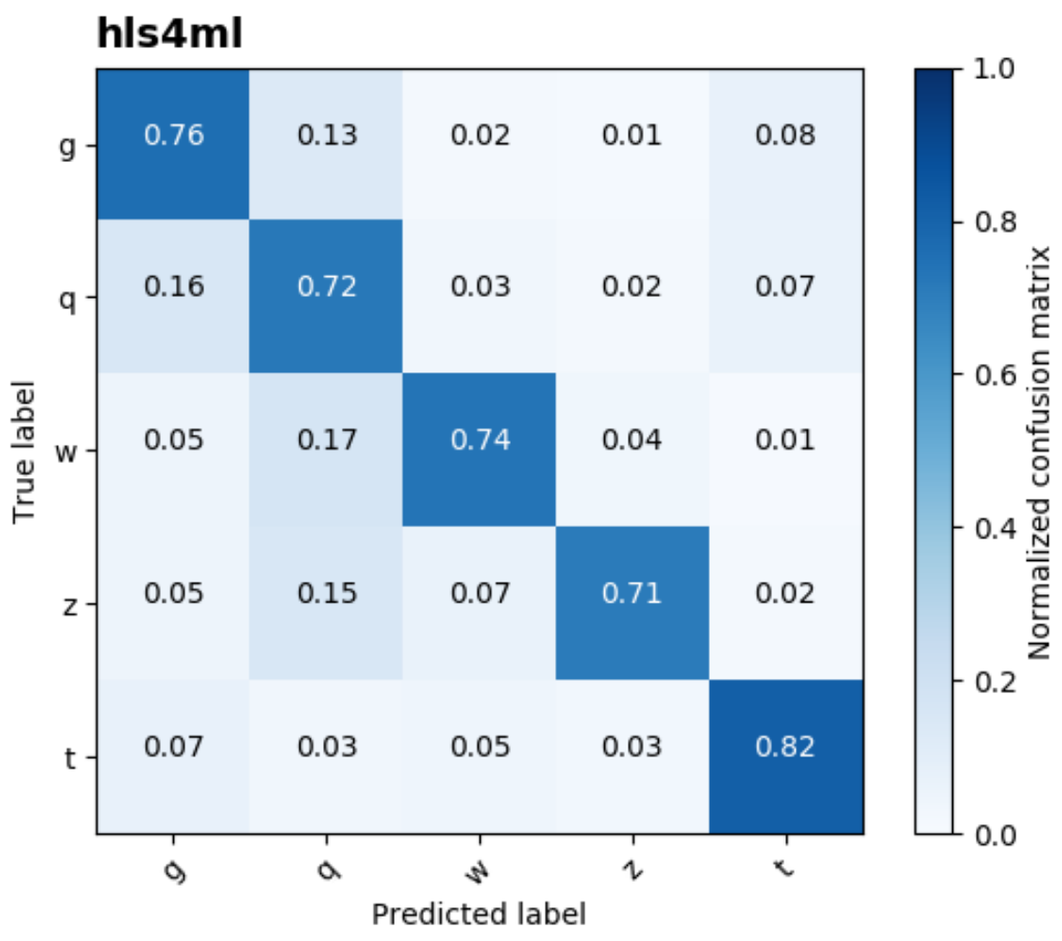


3-layers Dense NN (1)

Multi classification task: discrimination between the 5 classes of jets (t,W,Z,q,g)







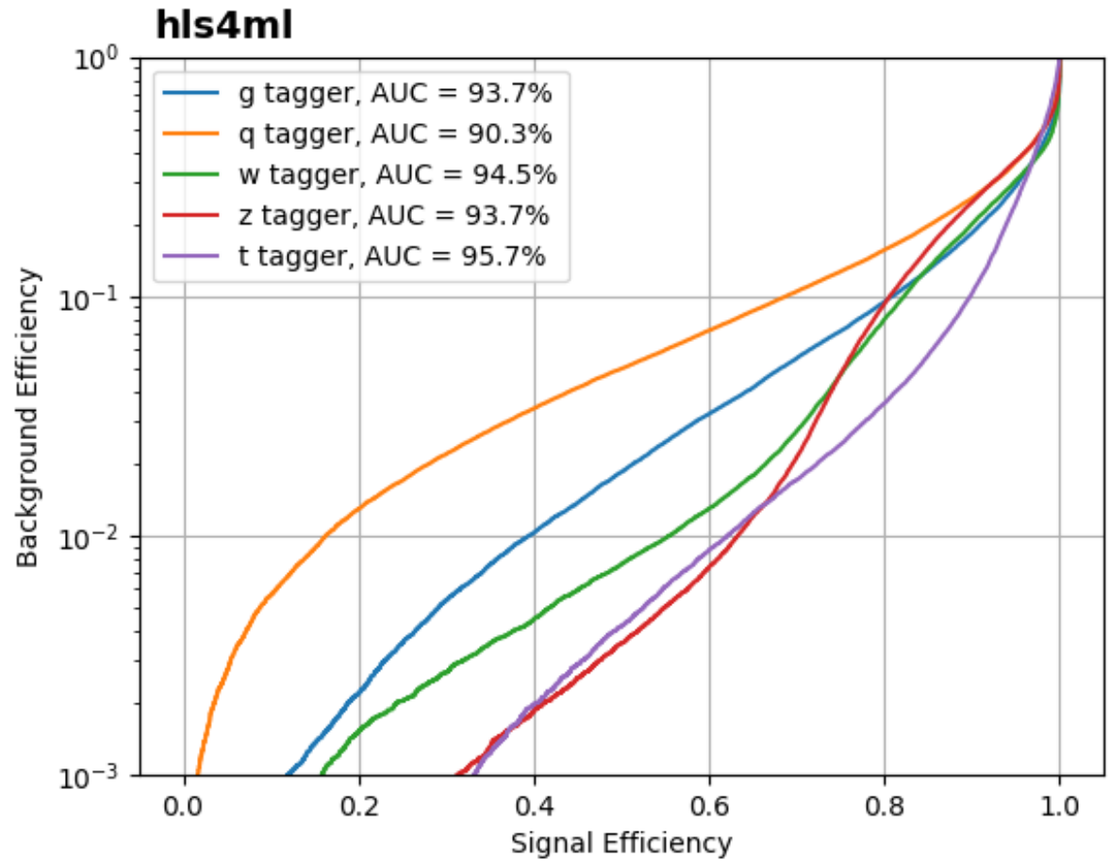
3-layers Dense NN (2)

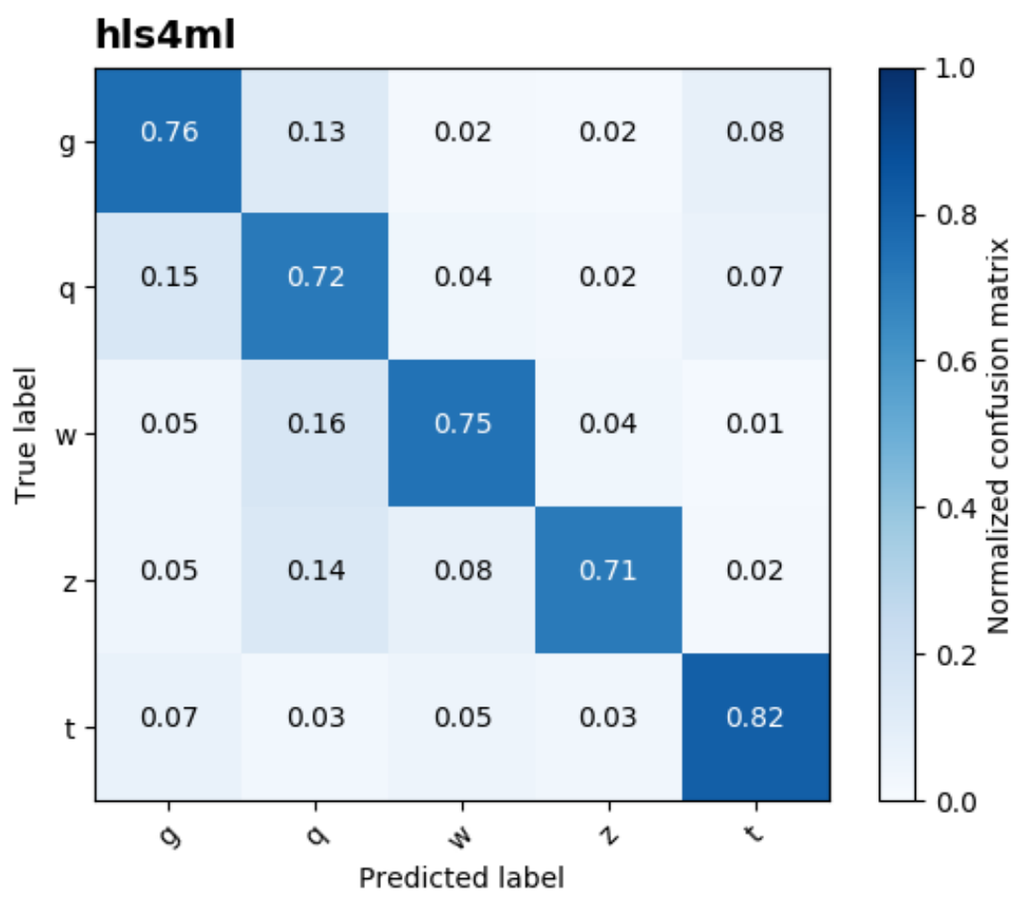
As previous slide but adding batch normalization layers [*]

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 16)	0
fc1_relu (Dense)	(None, 64)	1088
bn1 (BatchNormalization)	**(None, 64)**	**256**
relu1 (Activation)	(None, 64)	0
fc2_relu (Dense)	(None, 32)	2080
bn2 (BatchNormalization)	**(None, 32)**	**128**
relu2 (Activation)	(None, 32)	0
fc3_relu (Dense)	(None, 32)	1056
bn3 (BatchNormalization)	**(None, 32)**	**128**
relu3 (Activation)	(None, 32)	0
output_softmax (Dense)	(None, 5)	165
bn4 (BatchNormalization)	**(None, 5)**	**20**
softmax (Activation)	(None, 5)	0

Total params: 4,921
Trainable params: 4,655
Non-trainable params: 266

[*] nb, we choose here to do batch normalization before the activation but there are different ways to do this.This results in one more Dense layer (output_softmax).





3-layers Binary Dense

3-layers with batch normalization as in previous slides but:

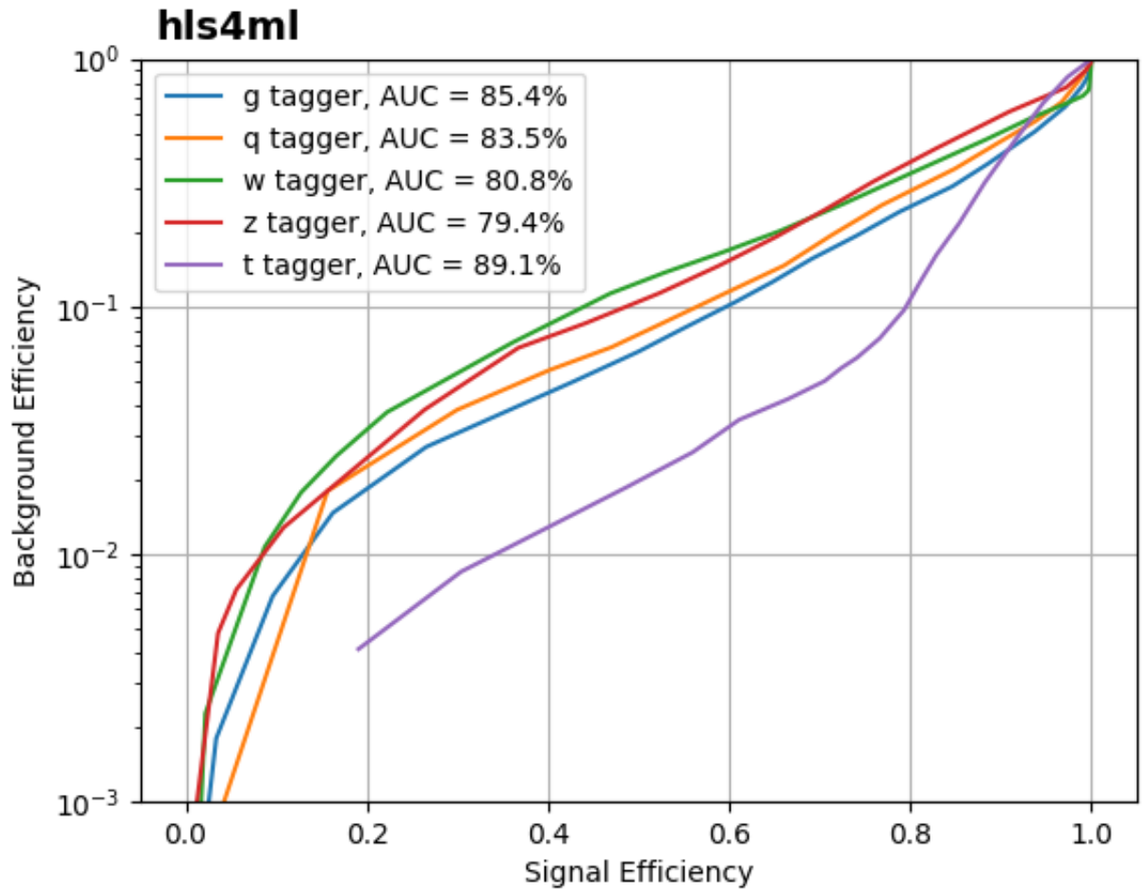
weights = +/- 1

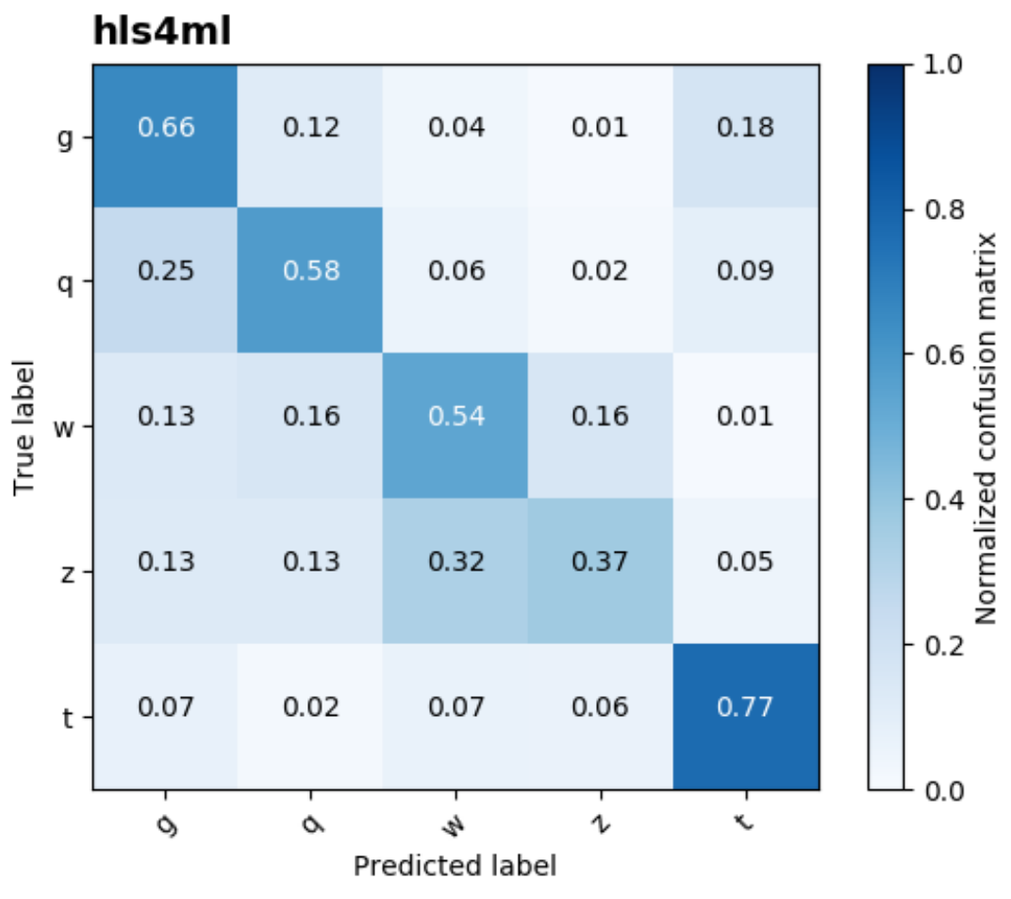
binary activation function

no output softmax

Layer (type)	Output Shape	Param #
fc1 (BinaryDense)	**(None, 64)**	**1024**
bn1 (BatchNormalization)	(None, 64)	256
act1 (Activation)	(None, 64)	0
fc2 (BinaryDense)	**(None, 32)**	**2048**
bn2 (BatchNormalization)	(None, 32)	128
act2 (Activation)	(None, 32)	0
fc3 (BinaryDense)	**(None, 32)**	**1024**
bn3 (BatchNormalization)	(None, 32)	128
act3 (Activation)	(None, 32)	0
output (BinaryDense)	**(None, 5)**	**160**
bn (BatchNormalization)	(None, 5)	20

Total params: 4,788
Trainable params: 4,522
Non-trainable params: 266





3-layers Ternary Dense

3-layers with batch normalization as in previous slides but:

weights = +/- 1 or 0

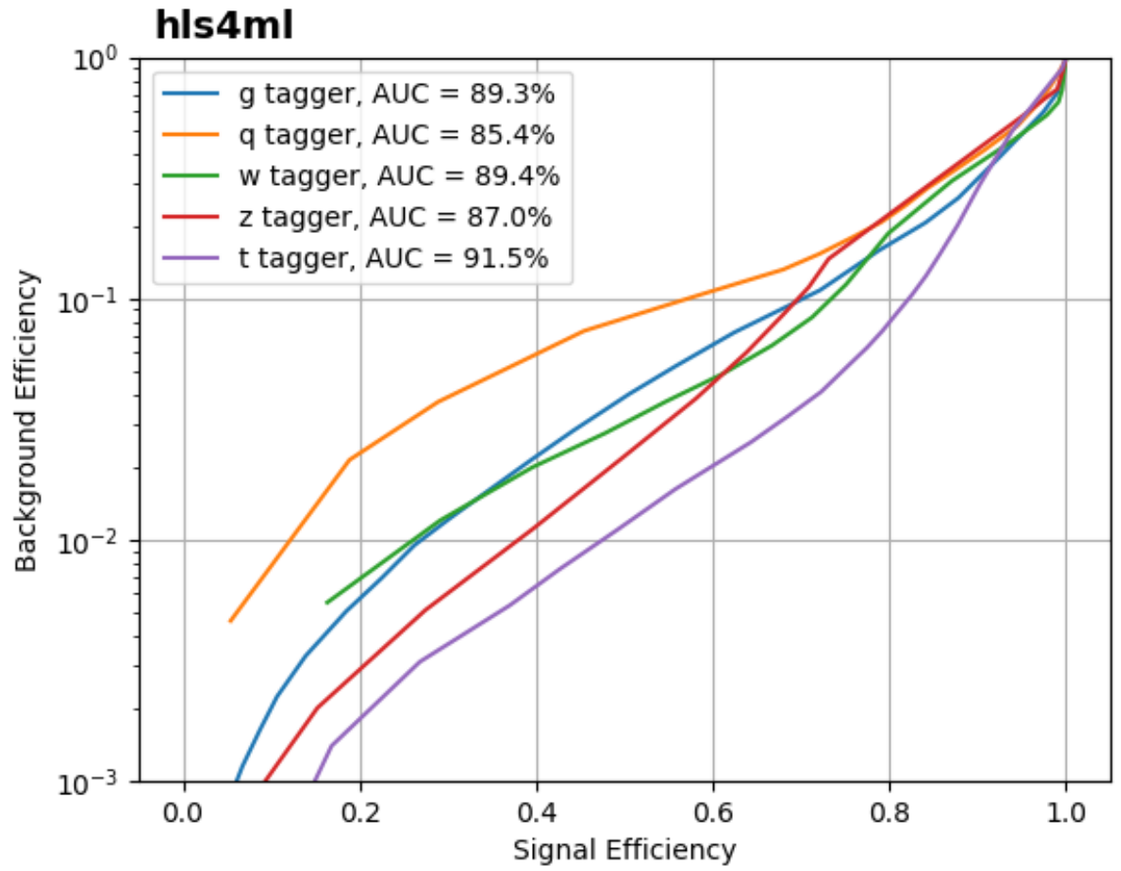
ternary activation function

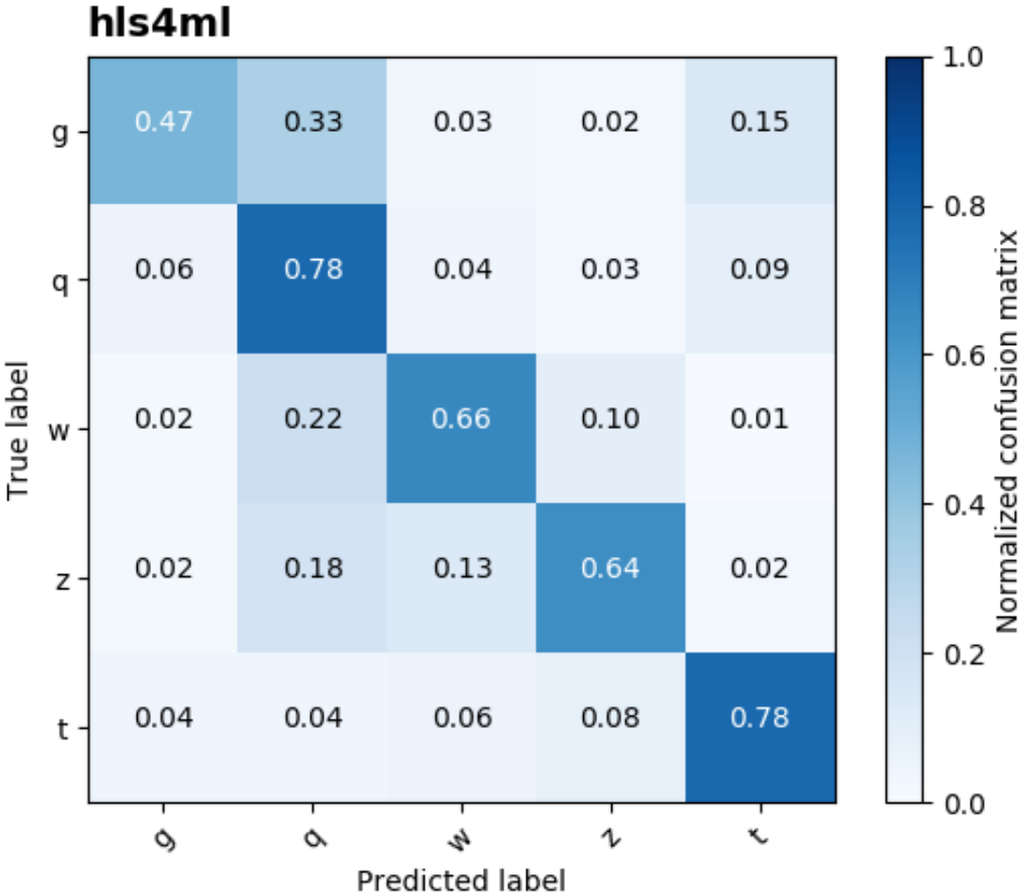
no output softmax

Layer (type)	Output Shape	Param #
fc1 (TernaryDense)	**(None, 64)**	**1024**
bn1 (BatchNormalization)	(None, 64)	256
act1 (Activation)	(None, 64)	0
fc2 (TernaryDense)	**(None, 32)**	**2048**
bn2 (BatchNormalization)	(None, 32)	128
act2 (Activation)	(None, 32)	0
fc3 (TernaryDense)	**(None, 32)**	**1024**
bn3 (BatchNormalization)	(None, 32)	128
act3 (Activation)	(None, 32)	0
output (TernaryDense)	**(None, 5)**	**160**
bn (BatchNormalization)	(None, 5)	20

Total params: 4,788
Trainable params: 4,522
Non-trainable params: 266

[*] more info: https://arxiv.org/abs/1605.04711





List of available models

- Find the weigths (.h5 file) and model architecture (.json file) under the `example-keras-model-files`

  - **1-layer Dense NN:**
    `KERAS_1layer.json, KERAS_1layer_weights.h5`

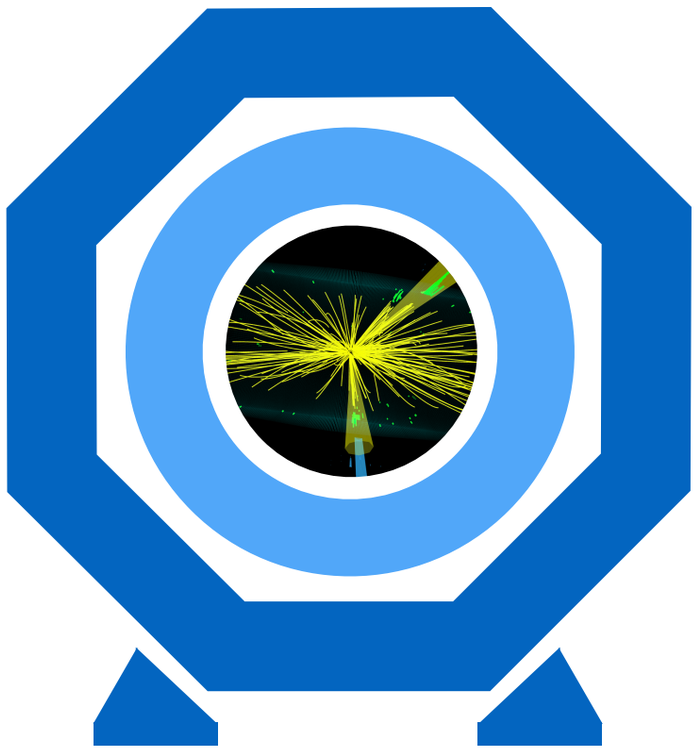
  - **3-layers Dense NN:**
    `KERAS_3layer.json, KERAS_3layer_weights.h5`

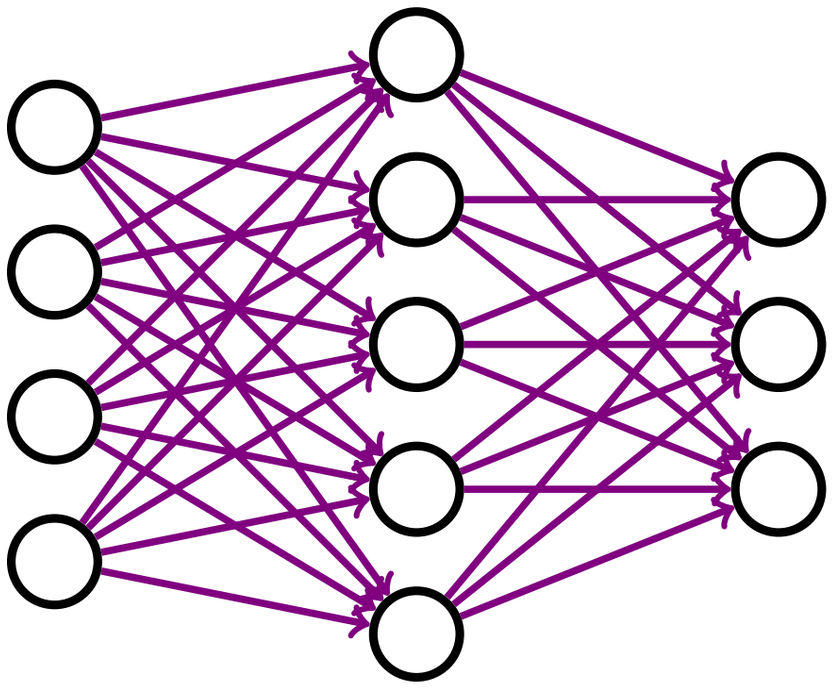
  - **3-layers Dense NN + BatchNormalization:**
    `KERAS_3layer_batch_norm.json, KERAS_3layer_batch_norm_weights.h5`

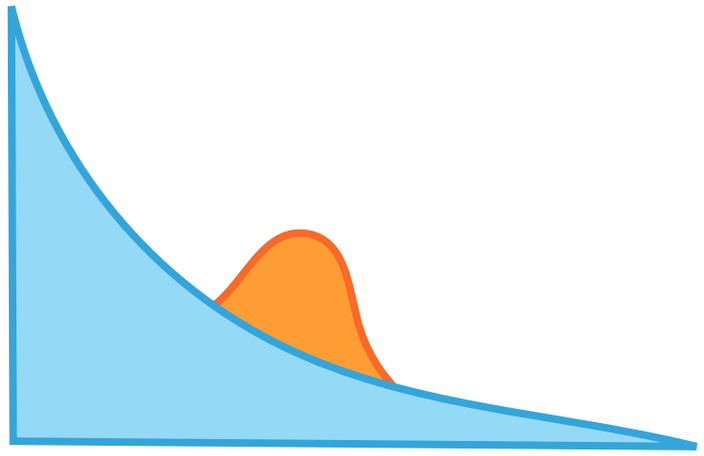
  - **3-layers Binary Dense NN:**
    `KERAS_3layer_binary.json, KERAS_3layer_binary_weights.h5`

  - **3-layers Ternary Dense NN:**
    `KERAS_3layer_ternary.json, KERAS_3layer_ternary_weights.h5`

Pick your favourite one to implement it on FPGA!







JAVIER DUARTE

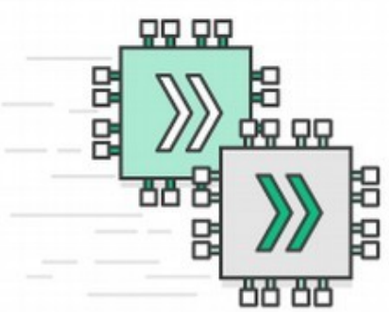
JULY 16, 2019

INSTITUT PASCAL

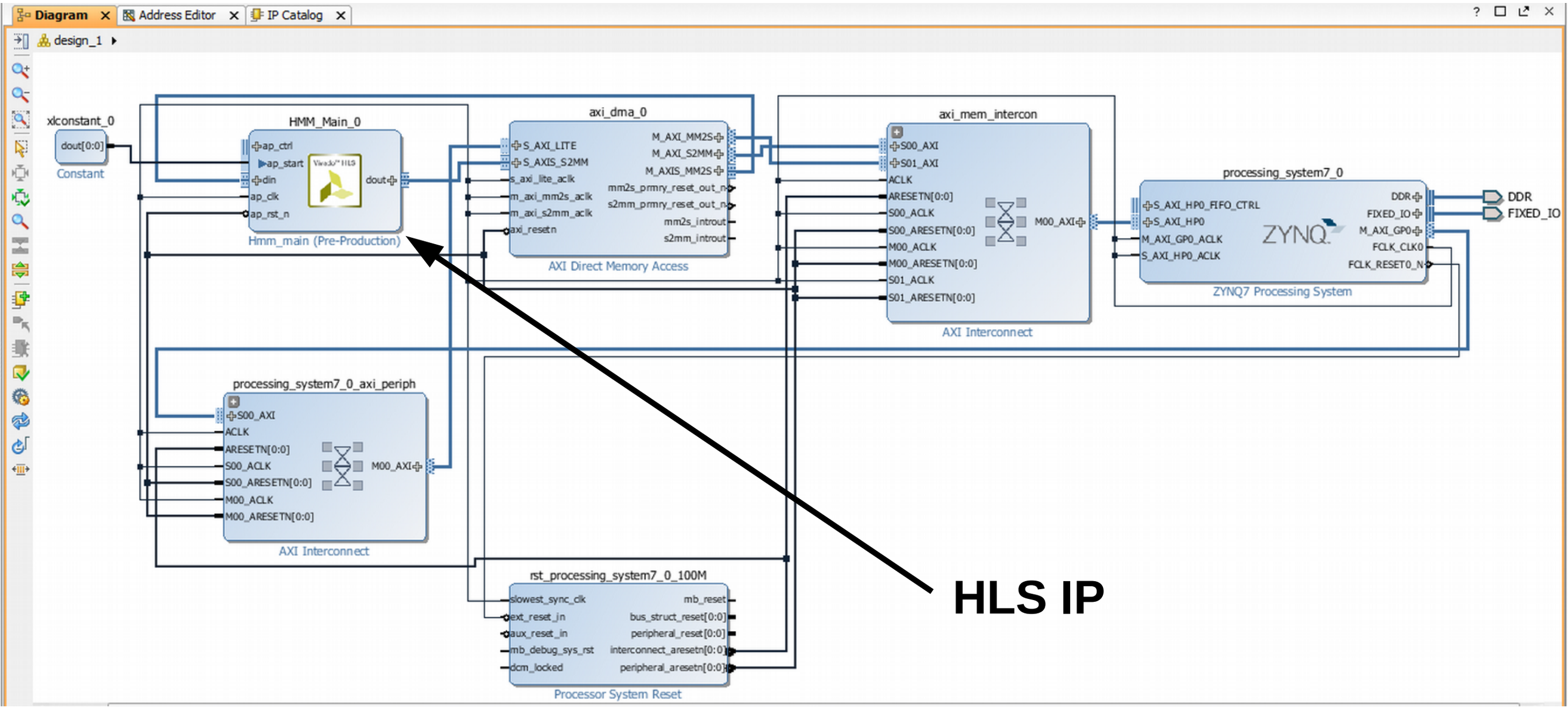
TUTORIAL PART 2: HLS4ML & SDACCEL ON AWS F1 FPGA

Introduction

- HLS project is just one part of actually running an application on an FPGA
  - Need to handle data in to/out of FPGA, how to actually route signals through FPGA, etc.
- SDAccel is a tool that helps in development/implementing designs on FPGAs specifically for acceleration (CPU ↔ FPGA)
- Will show today how to use SDAccel to accelerate an hls4ml project

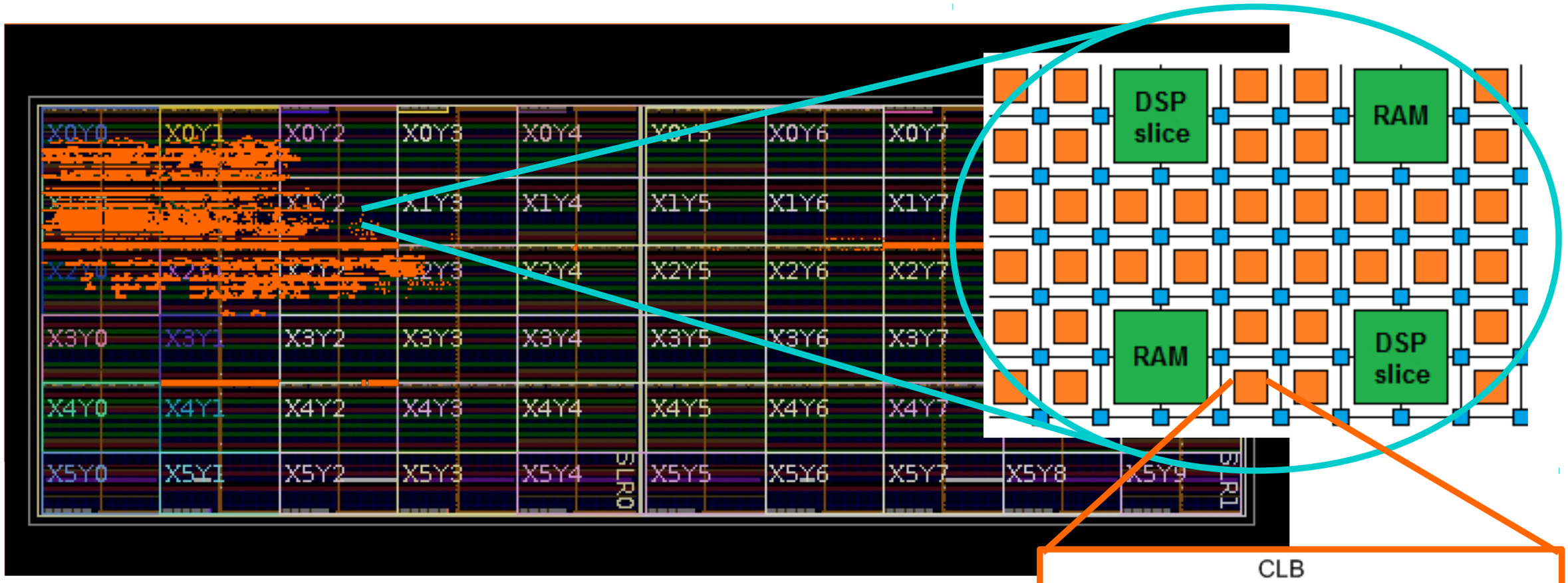


Programming an FPGA



- Typically need to specify how signals will be sent through FPGA

Programming an FPGA



3 layer, reuse = 1, KU 115, pruned

- With block design, can then attempt to route signals on physical FPGA
- Relies on knowledge of specific device, layout of components

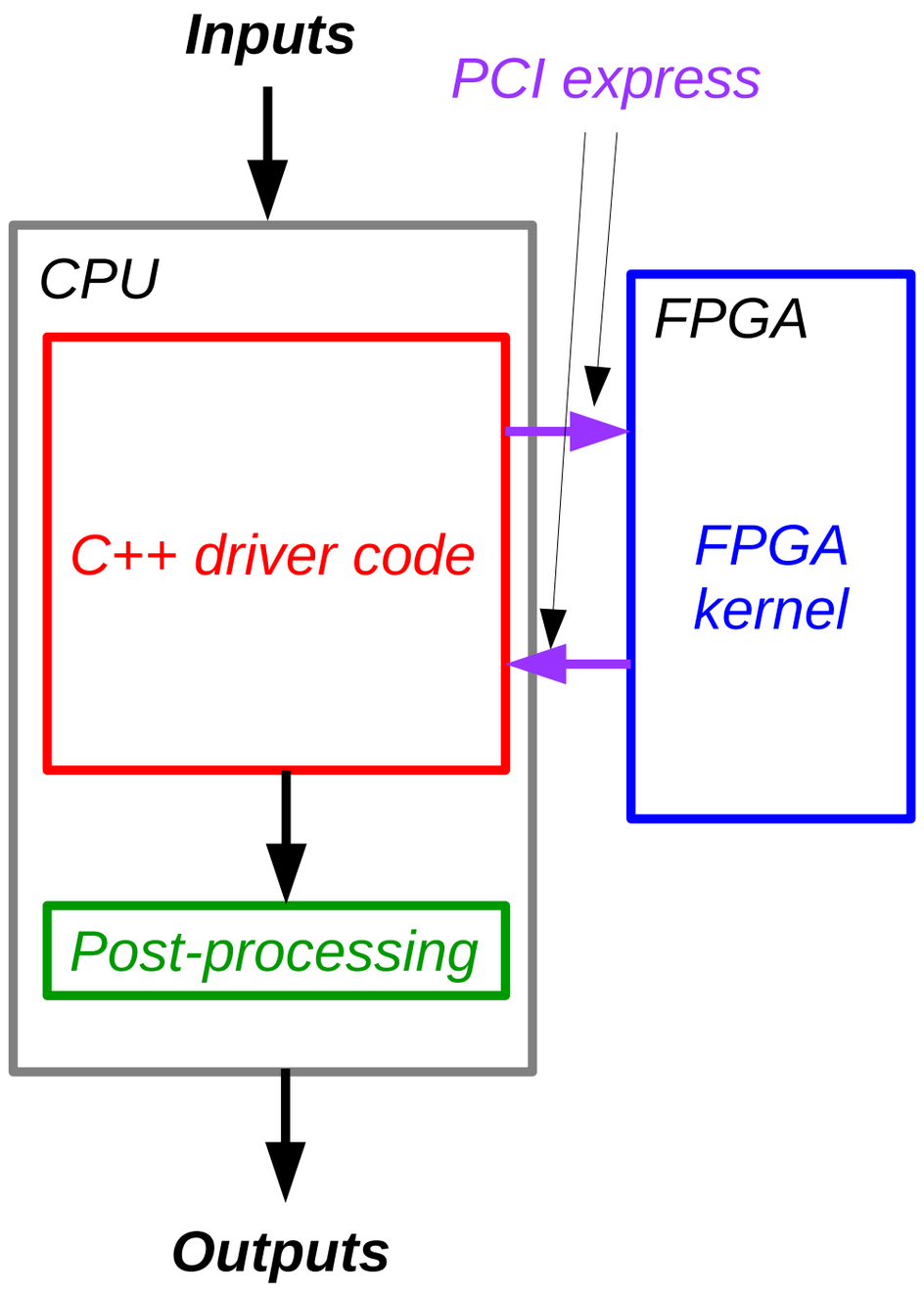
aws Overview

- AWS F1 instances are machines connected directly to a Xilinx Virtex UltraScale+ FPGA (VU9P) using PCI-express
- Good to use cheaper general computing AWS instances (T2) to develop applications
  - F1 cost: $1.65/hr
  - T2 cost: $0.37/hr
- General application development on AWS done using SDAccel

Virtex Ultrascale+ VU9P
6800 DSPs
1M LUTs
2M FFs
75 Mb BRAM

SDAccel™ Environment

- SDAccel Development Environment allows the development/running of connected FPGA kernels and CPU processes
- Define FPGA kernel using HLS, OpenCL, or VHDL/Verilog
  - Need to meet certain design constraints regarding control, input/output protocol
- Any FPGA application defined in one of these languages can be easily accelerated using SDAccel
- Once FPGA kernel is available, write host code to run on CPU and manage data transfer, FPGA execution
  - Examples: https://github.com/Xilinx/SDAccel_Examples

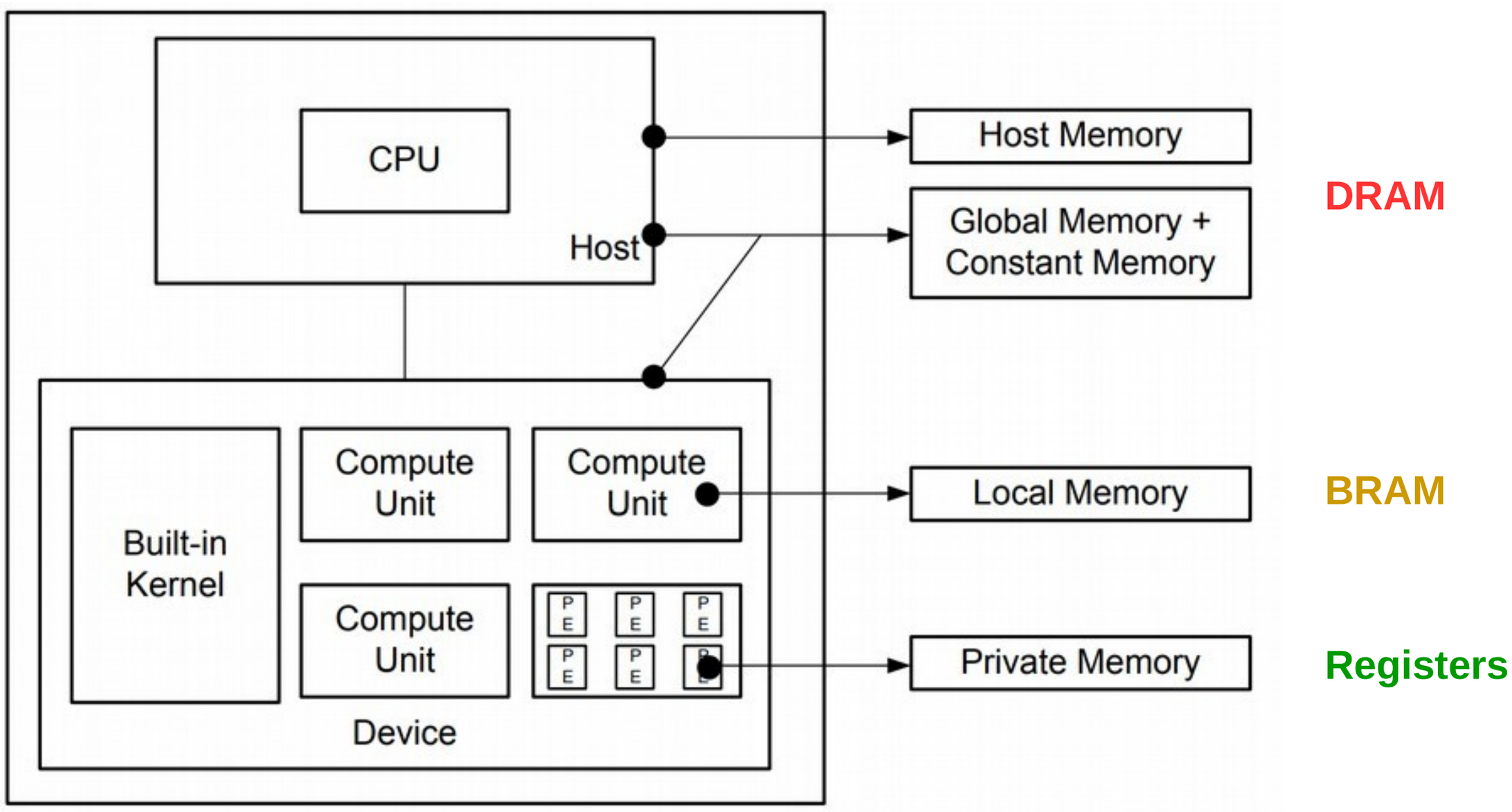


FPGA Kernel

- SDAccel kernels require inputs and outputs to be passed in certain manner
  - Must be AXI-stream
  - Must be mapped to global memory
  - Specific control

```
extern "C" {
void aws_hls4ml(
        const data_t *in, // Read-Only Vector
        data_t *out       // Output Result
        )
{
#pragma HLS INTERFACE m_axi port=in  offset=slave bundle=gmem
#pragma HLS INTERFACE m_axi port=out offset=slave bundle=gmem
#pragma HLS INTERFACE s_axilite port=in   bundle=control
#pragma HLS INTERFACE s_axilite port=out  bundle=control
#pragma HLS INTERFACE s_axilite port=return bundle=control
```

SDAccel Data Management



- Transfer between CPU and FPGA *must* use DRAM

FPGA Kernel

```
    unsigned short insize, outsize; //necessary for hls4ml kernel, not used

    input_t in_buf[STREAMSIZE][N_INPUTS];
    result_t out_buf[STREAMSIZE][N_OUTPUTS]; //these will get partitioned properly in
the hls4ml code

//getting data from axi stream and formatting properly
    for (int i = 0; i < STREAMSIZE; i++) {
#pragma HLS LOOP UNROLL
        for (int j = 0; j < N_INPUTS; j++) {
#pragma HLS LOOP UNROLL
            in_buf[i][j] = (input_t)in[i*N_INPUTS+j];
        }
    }

//run inference
    for (int i = 0; i < STREAMSIZE; i++) {
#pragma HLS dataflow
        Hls4ml: myproject(in_buf[i],out_buf[i],insize,outsize);
    }

//place output into axi stream output
    for (int i = 0; i < STREAMSIZE; i++) {
#pragma HLS LOOP UNROLL
        for (int j = 0; j < N_OUTPUTS; j++) {
#pragma HLS LOOP UNROLL
            out[i*N_OUTPUTS+j] = (data_t)out_buf[i][j];
        }
    }
```

Host Code

- Need to allocate block in memory for data
- Pass information to device with OpenCL buffer objects
  - `cl::Buffer`

```
size_t vector_size_in_bytes = sizeof(data_t) * DATA_SIZE_IN * STREAMSIZE;
size_t vector_size_out_bytes = sizeof(data_t) * DATA_SIZE_OUT * STREAMSIZE;
std::vector<data_t,aligned_allocator<data_t>> source_in(DATA_SIZE_IN*STREAMSIZE);
std::vector<data_t,aligned_allocator<data_t>> source_hw_results(DATA_SIZE_OUT*STREAMSIZE);

// OPENCL HOST CODE AREA START
// get_xil_devices() is a utility API which will find the xilinx
// platforms and will return list of devices connected to Xilinx platform
std::vector<cl::Device> devices = xcl::get_xil_devices();
cl::Device device = devices[0];
cl::Context context(device);
cl::CommandQueue q(context, device, CL_QUEUE_PROFILING_ENABLE);

// Allocate Buffer in Global Memory
// Buffers are allocated using CL_MEM_USE_HOST_PTR for efficient memory and
// Device-to-host communication
cl::Buffer buffer_in   (context,CL_MEM_USE_HOST_PTR | CL_MEM_READ_ONLY,
        vector_size_in_bytes, source_in.data());
cl::Buffer buffer_output(context,CL_MEM_USE_HOST_PTR | CL_MEM_WRITE_ONLY,
        vector_size_out_bytes, source_hw_results.data());
```

Host Code

- Specify binary file to use for programming FPGA
- Connect global memory buffers with kernel arguments

```
// find_binary_file() is a utility API which will search the xclbin file for
// targeted mode (sw_emu/hw_emu/hw) and for targeted platforms.
std::string binaryFile = xcl::find_binary_file(device_name,"aws_hls4ml");

// import_binary_file() ia a utility API which will load the binaryFile
// and will return Binaries.
cl::Program::Binaries bins = xcl::import_binary_file(binaryFile);
devices.resize(1);
cl::Program program(context, devices, bins);

std::vector<cl::Memory> inBufVec, outBufVec;
inBufVec.push_back(buffer_in);
outBufVec.push_back(buffer_output);

cl::Kernel krnl_aws_hls4ml(program,"aws_hls4ml");

int narg = 0;
krnl_aws_hls4ml.setArg(narg++, buffer_in);
krnl_aws_hls4ml.setArg(narg++, buffer_output);
```

Host Code

- To run:
- 1) Place inputs in allocated global memory
- 2) Launch kernel
- 3) Move outputs back from global memory when available

```
    // Copy input data to device global memory
    q.enqueueMigrateMemObjects(inBufVec,0/* 0 means from host*/);
    // Launch the Kernel
    // For HLS kernels global and local size is always (1,1,1). So, it
is recommended
    // to always use enqueueTask() for invoking HLS kernel
    q.enqueueTask(krnl_aws_hls4ml);
    // Copy Result from Device Global Memory to Host Local Memory
    q.enqueueMigrateMemObjects(outBufVec,CL_MIGRATE_MEM_OBJECT_HOST);
    // Check for any errors from the command queue
    q.finish();
```

Vivado HLS Synthesis Reports

- Can access standard Vivado HLS reports after `hw_emu/hw`
  - Exact location changes with SDAccel/AMI version
  - *FPGA HDK 1.5.0:* `_x/aws_hls4ml.hw_emu.xilinx_aws-vu9p-f1-04261818_dynamic_5_0/aws_hls4ml/aws_hls4ml/solution/syn/report/`**`myproject_csynth.rpt`**

```
================================================================
== Performance Estimates
================================================================
+ Timing (ns):
    * Summary:
    +--------+-------+----------+------------+
    |  Clock | Target| Estimated| Uncertainty|
    +--------+-------+----------+------------+
    |ap_clk  |   4.00|     2.920|        1.08|
    +--------+-------+----------+------------+

+ Latency (clock cycles):
    * Summary:
    +-----+-----+-----+-----+----------+
    |  Latency  |  Interval | Pipeline |
    | min | max | min | max |   Type   |
    +-----+-----+-----+-----+----------+
    |   24|   24|    1|    1| function |
    +-----+-----+-----+-----+----------+
```

Vivado HLS Synthesis Reports

- Can access standard Vivado HLS reports after `hw_emu/hw`
  - Exact location changes with SDAccel/AMI version
  - *FPGA HDK 1.5.0:* `_x/aws_hls4ml.hw_emu.xilinx_aws-vu9p-f1-04261818_dynamic_5_0/aws_hls4ml/aws_hls4ml/solution/syn/report/`**`myproject_csynth.rpt`**

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-------+---------+---------+-----+
|       Name      | BRAM_18K| DSP48E|    FF   |   LUT   | URAM|
+-----------------+---------+-------+---------+---------+-----+
|DSP              |        -|      -|        -|        -|    -|
|Expression       |        -|      -|        0|        6|    -|
|FIFO             |        -|      -|        -|        -|    -|
|Instance         |        -|    123|    14829|    65502|    -|
|Memory           |        -|      -|        -|        -|    -|
|Multiplexer      |        -|      -|        -|        9|    -|
|Register         |        -|      -|     6249|        -|    -|
+-----------------+---------+-------+---------+---------+-----+
|Total            |        0|    123|    21078|    65517|    0|
+-----------------+---------+-------+---------+---------+-----+
|Available        |     4320|   6840|  2364480|  1182240|  960|
+-----------------+---------+-------+---------+---------+-----+
|Utilization (%)  |        0|      1|    ~0   |        5|    0|
+-----------------+---------+-------+---------+---------+-----+
```

Vivado HLS Synthesis Reports

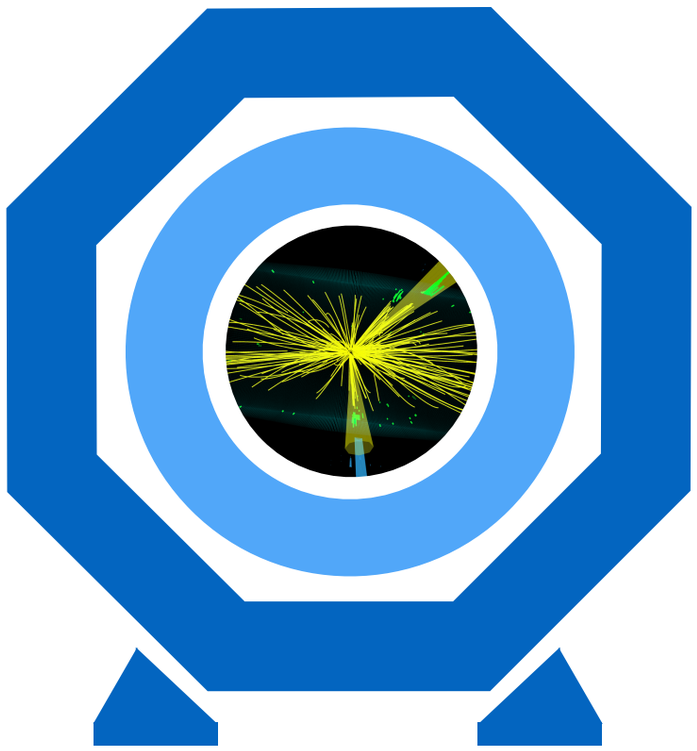
- Can access standard Vivado HLS reports after `hw_emu/hw`
  - Exact location changes with SDAccel/AMI version
  - *FPGA HDK 1.5.0:* `_x/aws_hls4ml.hw_emu.xilinx_aws-vu9p-f1-04261818_dynamic_5_0/aws_hls4ml/aws_hls4ml/solution/syn/report/`**`aws_hls4ml_csynth.rpt`**

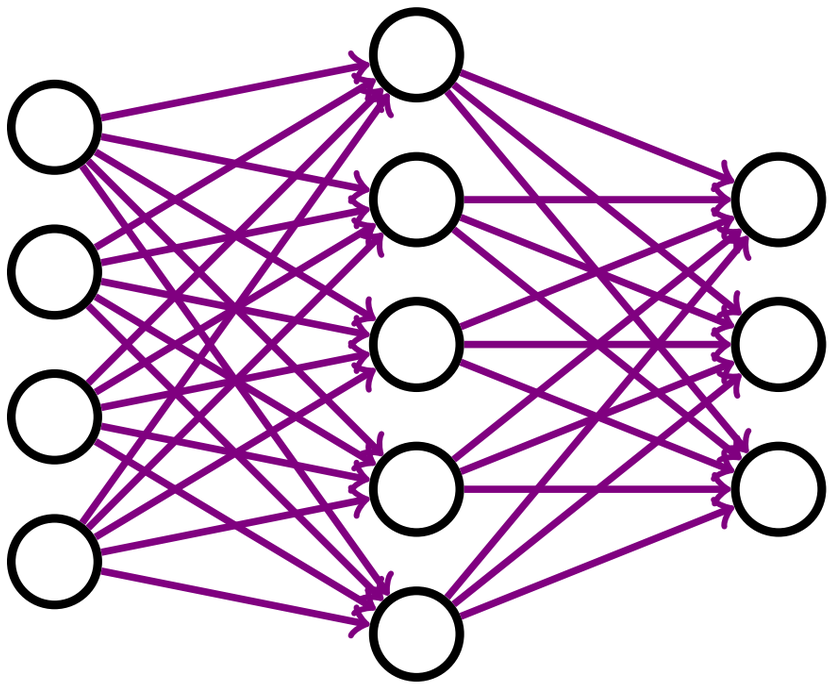
```
================================================================
== Performance Estimates
================================================================
+ Timing (ns):
    * Summary:
    +--------+-------+----------+------------+
    |  Clock | Target| Estimated| Uncertainty|
    +--------+-------+----------+------------+
    |ap_clk  |   4.00|     2.920|        1.08|
    +--------+-------+----------+------------+

+ Latency (clock cycles):
    * Summary:
    +-----+-----+-----+-----+---------+
    |  Latency  |  Interval | Pipeline|
    | min | max | min | max |   Type  |
    +-----+-----+-----+-----+---------+
    |   58|   58|   58|   58|   none  |
    +-----+-----+-----+-----+---------+
```

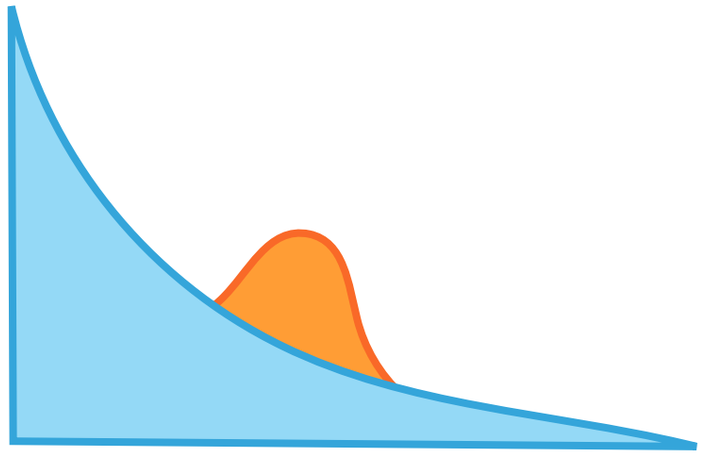
Vivado HLS Synthesis Reports

- Can access standard Vivado HLS reports after `hw_emu/hw`
  - Exact location changes with SDAccel/AMI version
  - *FPGA HDK 1.5.0:* `_x/aws_hls4ml.hw_emu.xilinx_aws-vu9p-f1-04261818_dynamic_5_0/aws_hls4ml/aws_hls4ml/solution/syn/report/`**`aws_hls4ml_csynth.rpt`**

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-------+---------+---------+-----+
|       Name      | BRAM_18K| DSP48E|    FF   |   LUT   | URAM|
+-----------------+---------+-------+---------+---------+-----+
|DSP              |        -|      -|        -|        -|    -|
|Expression       |        -|      -|        -|        -|    -|
|FIFO             |        -|      -|        -|        -|    -|
|Instance         |        2|    123|    21791|    66490|    -|
|Memory           |        -|      -|        -|        -|    -|
|Multiplexer      |        -|      -|        -|      362|    -|
|Register         |        -|      -|      768|        -|    -|
+-----------------+---------+-------+---------+---------+-----+
|Total            |        2|    123|    22559|    66852|    0|
+-----------------+---------+-------+---------+---------+-----+
|Available        |     4320|   6840|  2364480|  1182240|  960|
+-----------------+---------+-------+---------+---------+-----+
|Utilization (%)  |    ~0   |      1|    ~0   |        5|    0|
+-----------------+---------+-------+---------+---------+-----+
```







JAVIER DUARTE

JULY 16, 2019

INSTITUT PASCAL

SUMMARY

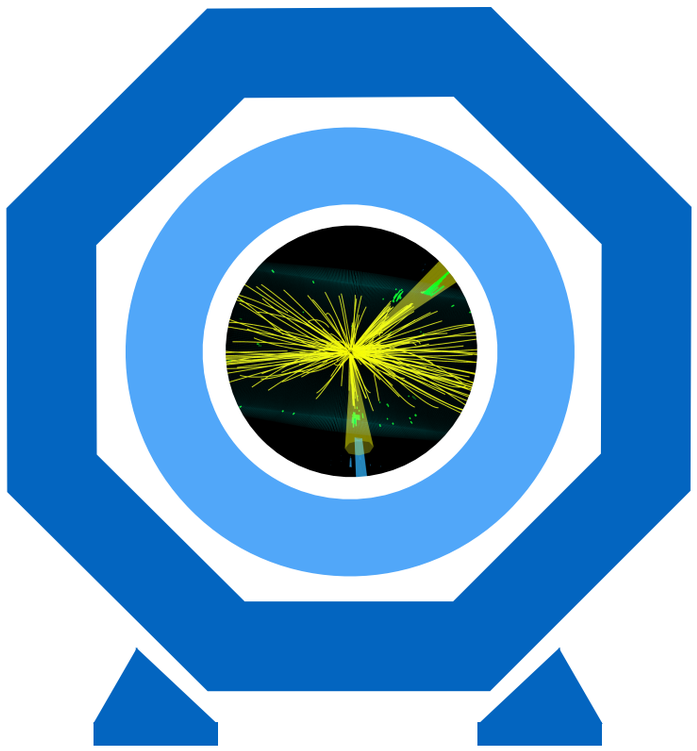
LOGISTICS ARE HARD

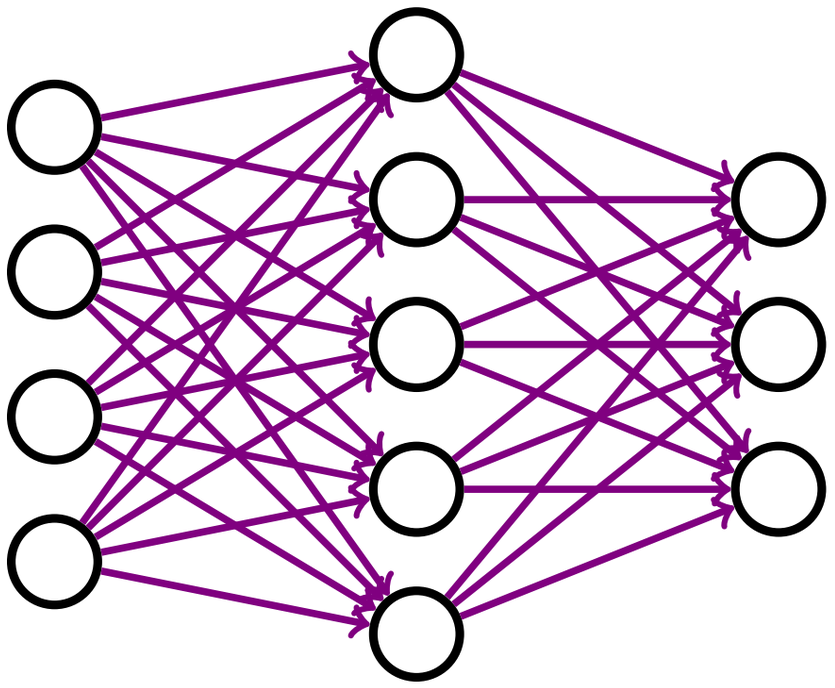
- Set up tutorial through FNAL resources (but ssh restrictions!)
- 15 CERN→FNAL ssh tunnels through had to be set up!
- AWS f1.2xlarge instances cost $1.65 / hour: total cost ~$200
- Compiling firmware takes a long time! Even running it during lunch (~12:30pm) it didn't finish until ~4pm
- Demonstrated neural network compression in the meantime

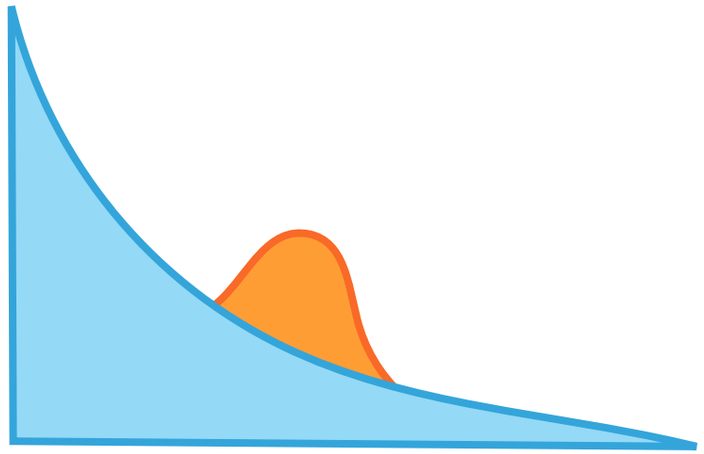
```
                StateIP         Owner   IDtype         AMI  AZ Launch_time (UTC)    Lifetime

running  54.189.158.71   fpga4hep   i-0f41ad4d5b9ca6c45f1.2xlargeFPGA HDK 1.6.0us-west-2a2019-07-16 08:53:46  1 day, 0:00:00
running    52.37.39.63   fpga4hep   i-054a4095b516d1771f1.2xlargeFPGA HDK 1.6.0us-west-2a2019-07-16 08:53:47  1 day, 0:00:00
running   52.43.22.213   fpga4hep   i-0ed42805a4868bb31f1.2xlargeFPGA HDK 1.6.0us-west-2a2019-07-16 08:53:47  1 day, 0:00:00
running 18.236.152.145   fpga4hep   i-04a9db8d1ff53237cf1.2xlargeFPGA HDK 1.6.0us-west-2a2019-07-16 08:53:47  1 day, 0:00:00
running   34.217.24.26   fpga4hep   i-0ab8694d391111838f1.2xlargeFPGA HDK 1.6.0us-west-2a2019-07-16 08:53:48  1 day, 0:00:00
running  54.202.68.101   fpga4hep   i-05af6e63c99e5cc1af1.2xlargeFPGA HDK 1.6.0us-west-2a2019-07-16 08:53:48  1 day, 0:00:00
running    52.43.5.172   fpga4hep   i-0d636124f9576d191f1.2xlargeFPGA HDK 1.6.0us-west-2a2019-07-16 08:53:48  1 day, 0:00:00
running  54.149.231.83   fpga4hep   i-07dbc173c5a3c0dd6f1.2xlargeFPGA HDK 1.6.0us-west-2a2019-07-16 08:53:49  1 day, 0:00:00
running 34.216.113.101   fpga4hep   i-0aa4fe8b02d282c0ff1.2xlargeFPGA HDK 1.6.0us-west-2a2019-07-16 08:53:49  1 day, 0:00:00
running 18.236.161.220   fpga4hep   i-094ef25463ec5d003f1.2xlargeFPGA HDK 1.6.0us-west-2a2019-07-16 08:53:49  1 day, 0:00:00
running  54.203.20.100   fpga4hep   i-08d9b77239ac65ef4f1.2xlargeFPGA HDK 1.6.0us-west-2a2019-07-16 08:53:49  1 day, 0:00:00
running  54.188.37.195   fpga4hep   i-09667447091b50067f1.2xlargeFPGA HDK 1.6.0us-west-2a2019-07-16 08:53:50  1 day, 0:00:00
running 34.222.132.102   fpga4hep   i-0e87640b0d8873e86f1.2xlargeFPGA HDK 1.6.0us-west-2a2019-07-16 08:53:50  1 day, 0:00:00
running 34.220.228.122   fpga4hep   i-0024653f016e3b03ef1.2xlargeFPGA HDK 1.6.0us-west-2a2019-07-16 08:53:50  1 day, 0:00:00
running   52.89.37.141   fpga4hep   i-0a758b565a92ea611f1.2xlargeFPGA HDK 1.6.0us-west-2a2019-07-16 08:53:51  1 day, 0:00:00
```

SOME DISCUSSION

- SDAccel is nice for making programming FPGAs more accessible to programmers/industry (lots of infrastructure), but is it the way forward for physicists?
  - Definitely not for Level-1 trigger (PCIe is not fast enough!)
  - May be good for high-level trigger or offline (application for HCAL reconstruction)
- Other layer types (convolutional, recurrent, graph) in active development







JAVIER DUARTE

JULY 16, 2019

INSTITUT PASCAL

BACKUP

FPGA DATA SHEET

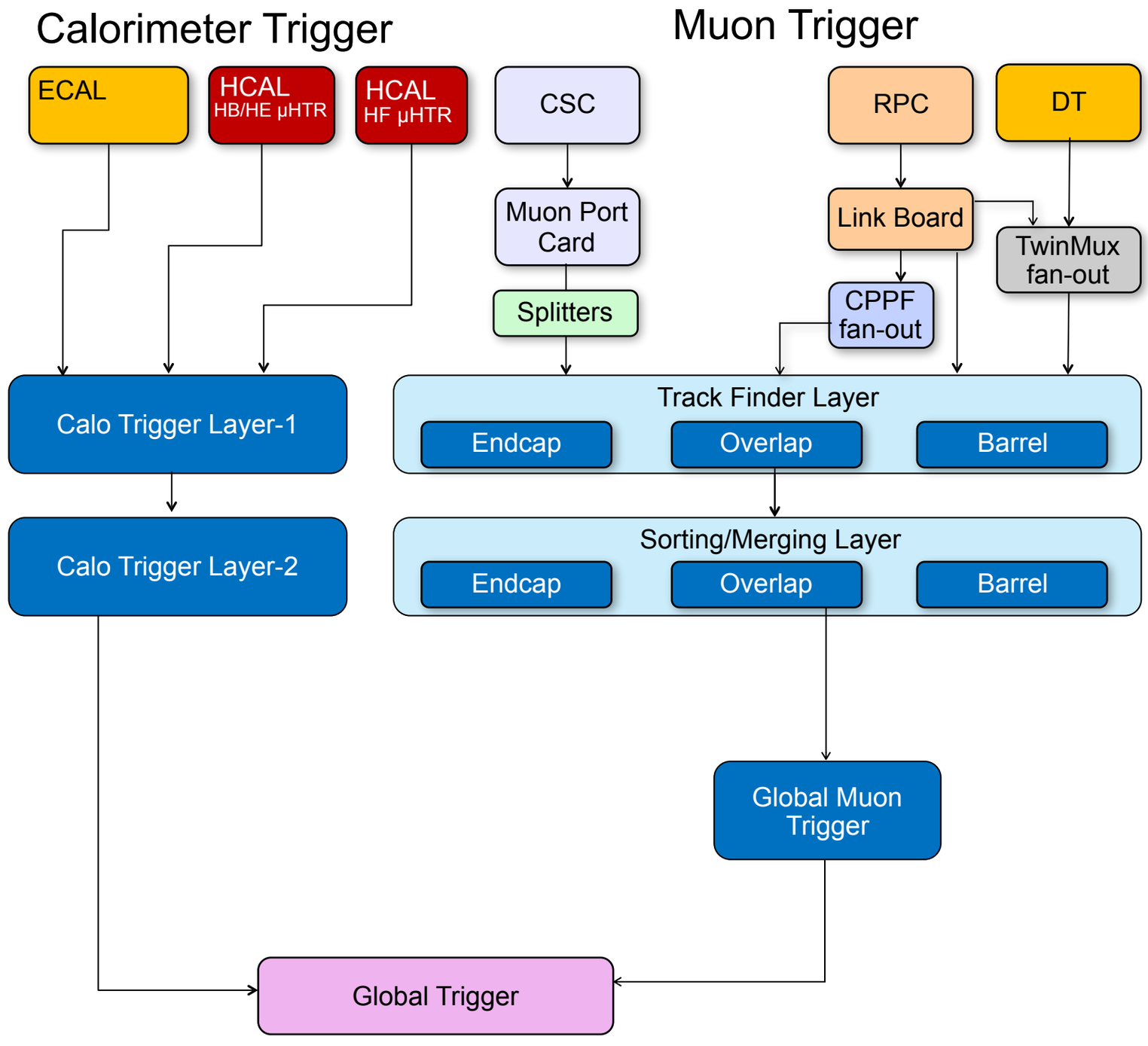
			Foundation					58G PAM4	
Device Name			VU3P	VU5P	VU7P	VU9P	VU11P	VU13P	VU27P	VU29P
System Logic Cells (K)			862	1,314	1,724	2,586	2,835	3,780	2,835	3,780
CLB Flip-Flops (K)			788	1,201	1,576	2,364	2,592	3,456	2,592	3,456
CLB LUTs (K)			394	601	788	1,182	1,296	1,728	1,296	1,728
Max. Dist. RAM (Mb)			12.0	18.3	24.1	36.1	36.2	48.3	36.2	48.3
Total Block RAM (Mb)			25.3	36.0	50.6	75.9	70.9	94.5	70.9	94.5
UltraRAM (Mb)			90.0	132.2	180.0	270.0	270.0	360.0	270.0	360.0
HBM DRAM (GB)			–	–	–	–	–	–	–	–
HBM AXI Interfaces			–	–	–	–	–	–	–	–
Clock Mgmt Tiles (CMTs)			10	20	20	30	12	16	16	16
DSP Slices			2,280	3,474	4,560	6,840	9,216	12,288	9,216	12,288
Peak INT8 DSP (TOP/s)			7.1	10.8	14.2	21.3	28.7	38.3	28.7	38.3
PCIe® Gen3 x16			2	4	4	6	3	4	1	1
PCIe Gen3 x16/Gen4 x8 / CCIX[1]			–	–	–	–	–	–	–	–
150G Interlaken			3	4	6	9	6	8	8	8
100G Ethernet w/ KR4 RS-FEC			3	4	6	9	9	12	15	15
Max. Single-Ended HP I/Os			520	832	832	832	624	832	676	676
GTY 32.75Gb/s Transceivers			40	80	80	120	96	128	32	32
GTM 58Gb/s PAM4 Transceivers									48	48
100G / 50G KP4 FEC									24 / 48	24 / 48
Extended[2]			-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3
Industrial			-1 -2	-1 -2	-1 -2	-1 -2	-1 -2	-1 -2	-1 -2	-1 -2
Footprint[3,4,5]		Dim. (mm)	HP I/O, GTY						HP I/O, GTY, GTM	
Footprint compatible with 20nm UltraScale Devices with same footprint identifier	C1517	40x40	520, 40							
	F1924[6]	45x45					624, 64			
	A2104	47.5x47.5		832, 52	832, 52	832, 52				
		52.5x52.5[7]						832, 52		
	B2104	47.5x47.5		702, 76	702, 76	702, 76	572, 76			
		52.5x52.5[7]						702, 76		
	C2104	47.5x47.5		416, 80	416, 80	416, 104	416, 96			
		52.5x52.5[7]						416, 104		
	D2104	47.5x47.5				676, 76	572, 76			
		52.5x52.5[7]						676, 76	676, 16, 30	676, 16, 30
	A2577	52.5x52.5				448, 120	448, 96	448, 128	448, 32, 48	448, 32, 48

Virtex® UltraScale+™ FPGAs

1. This block operates in compatibility mode for 16.0GT/s (Gen4) operation. See PG213.
2. -2LE (Tj = 0°C to 110°C). See Ordering Information in DS890.
3. For full part number details, see DS890, *UltraScale Architecture and Product Overview*.
4. All packages are 1.0mm ball pitch.
5. Consult UG583, *UltraScale Architecture PCB Design User Guide* for specific migration details.
6. The GTY transceiver line rate in the F1924 footprint is package limited to 16.3Gb/s. Refer to data sheet for details.
7. These 52.5x52.5mm packages have the same PCB ball footprint as the 47.5x47.5mm packages and are footprint compatible.

UPGRADING THE LEVEL-1 TRIGGER (BEFORE)

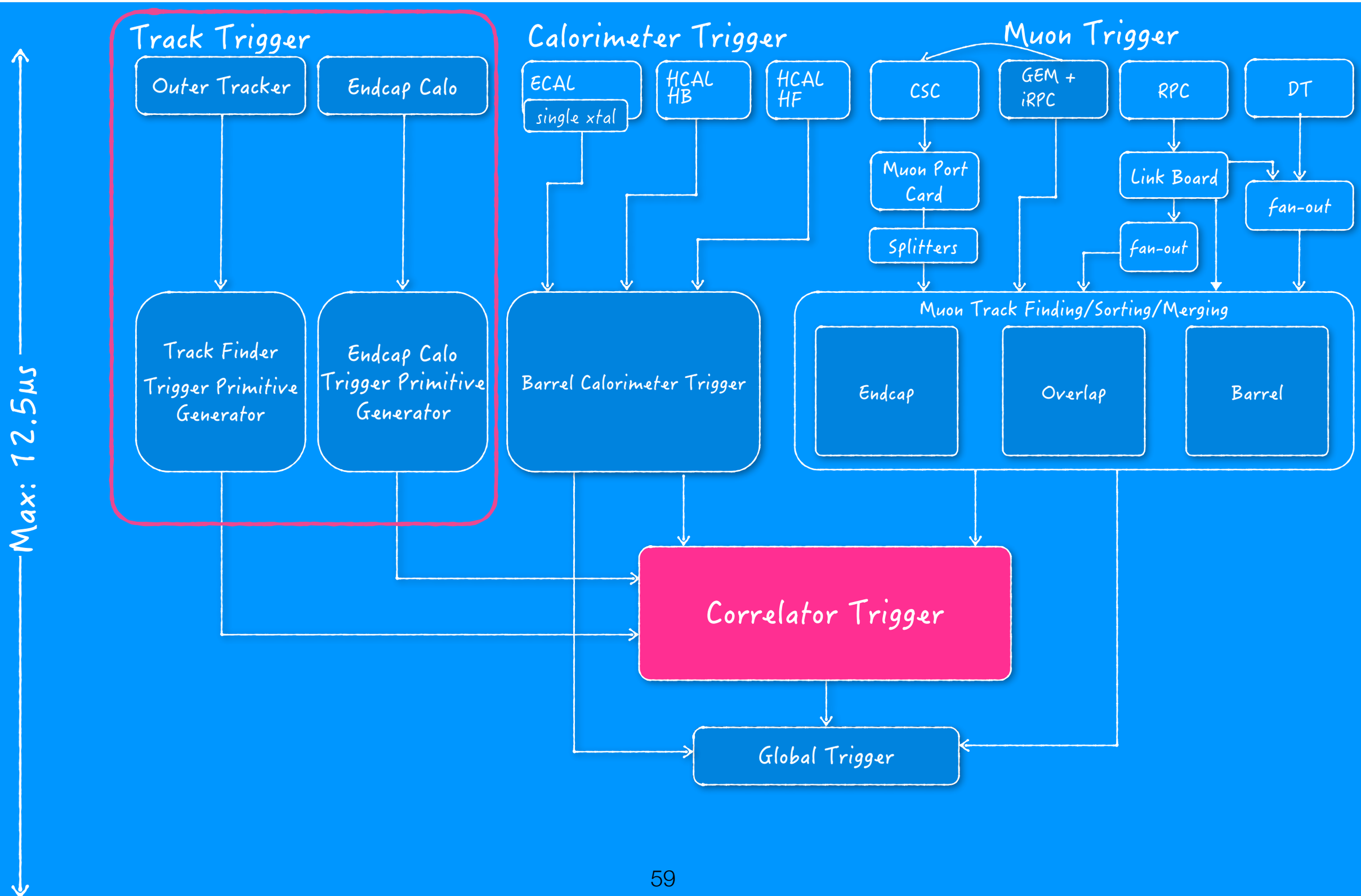
~4μs



UPGRADING THE LEVEL-1 TRIGGER (AFTER)

More and better information available in the Level-1 trigger!

What can we do with it?



CASE STUDY: JET TAGGING INPUTS

Observables

m_{mMDT}

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

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

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

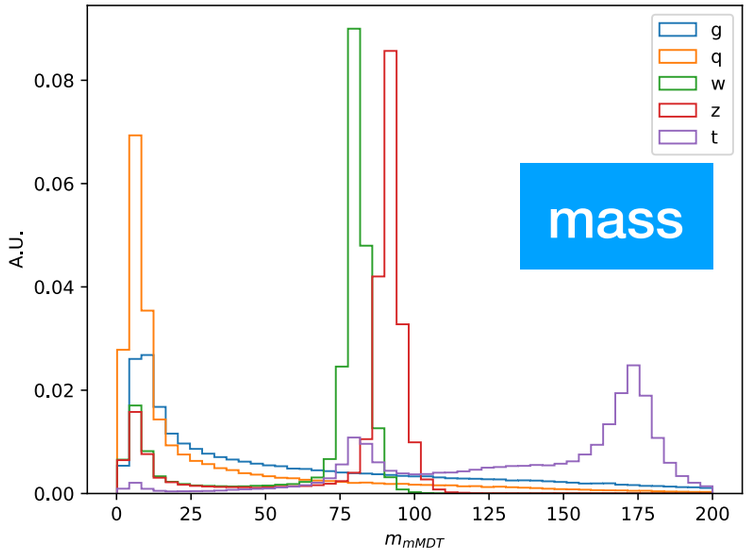
$C_2^{\beta=1,2}$

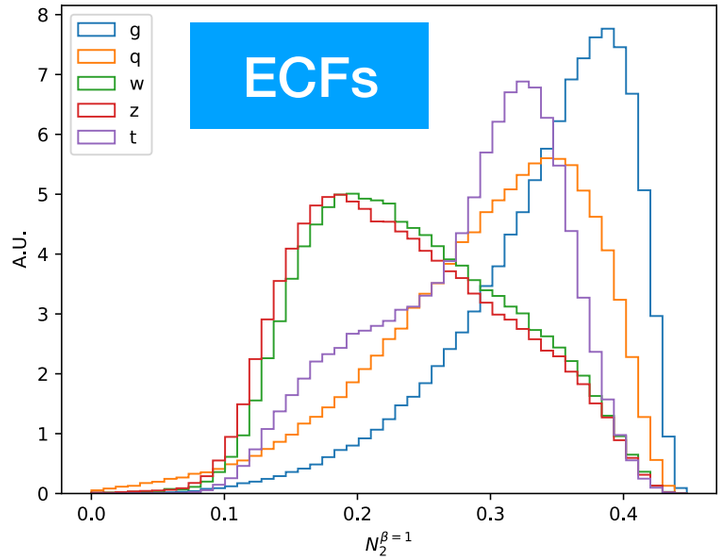
$D_2^{\beta=1,2}$

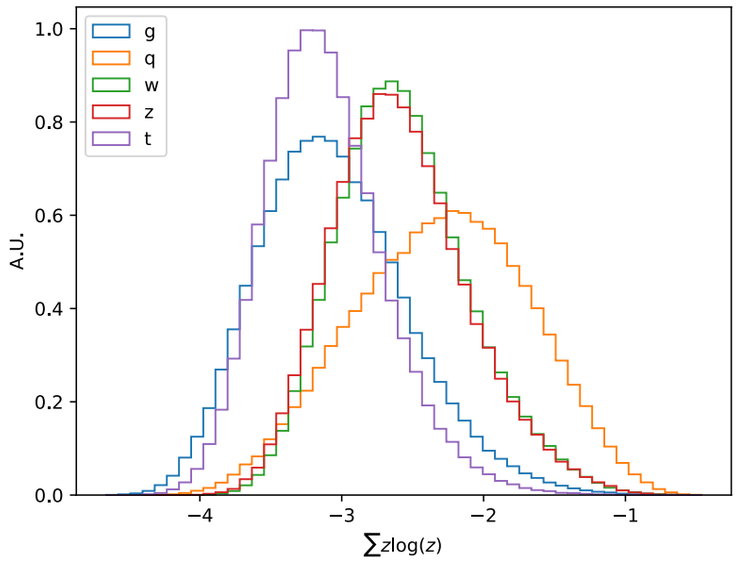
$D_2^{(\alpha,\beta)=(1,1),(1,2)}$

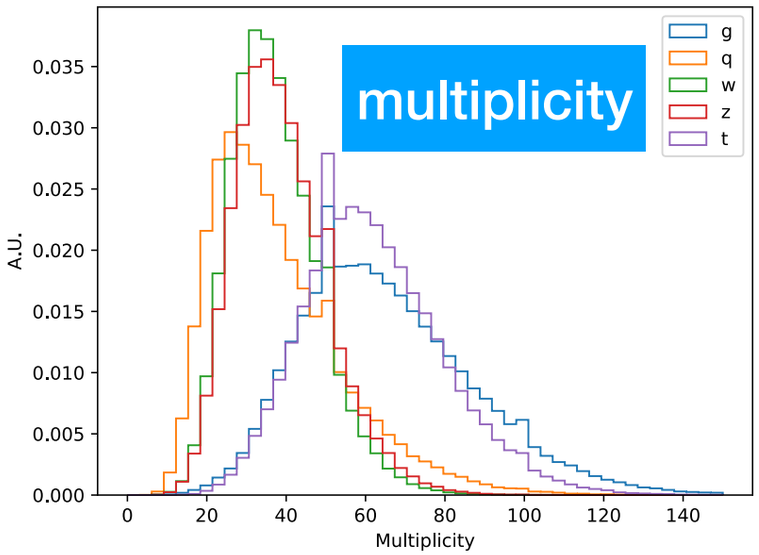
$\sum z \log z$

Multiplicity



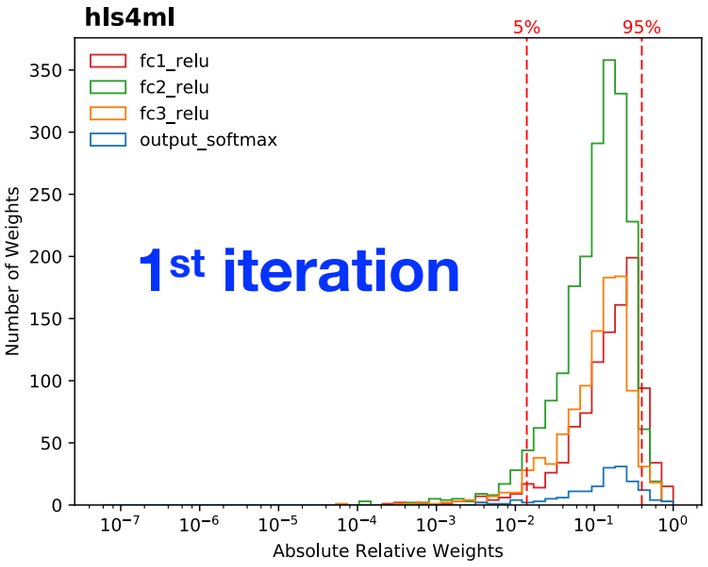




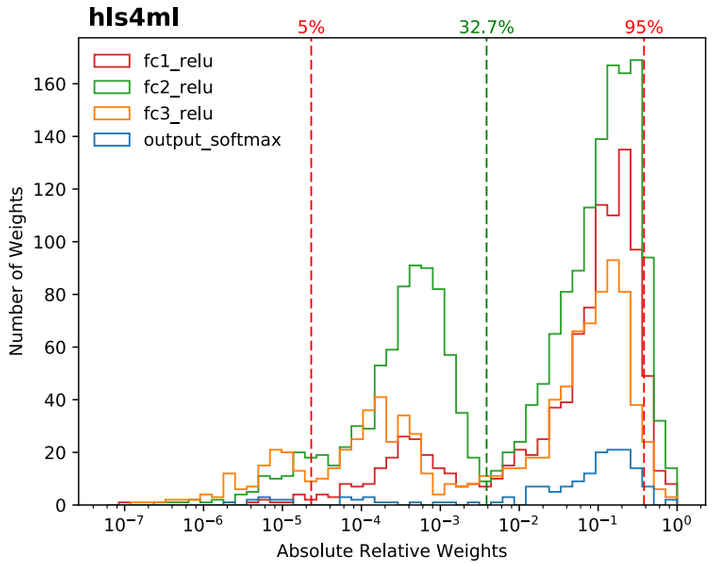


- 16 expert observables provide separation between top, W/Z, and quark/gluon

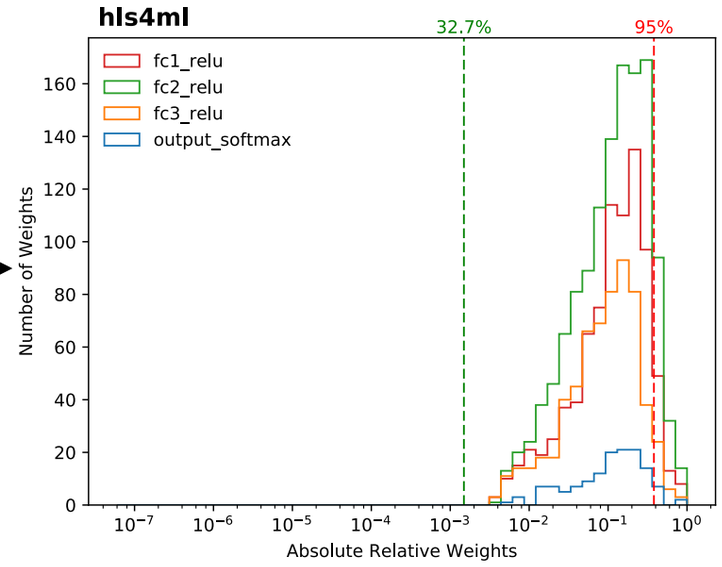
1st iteration



Train with L_1

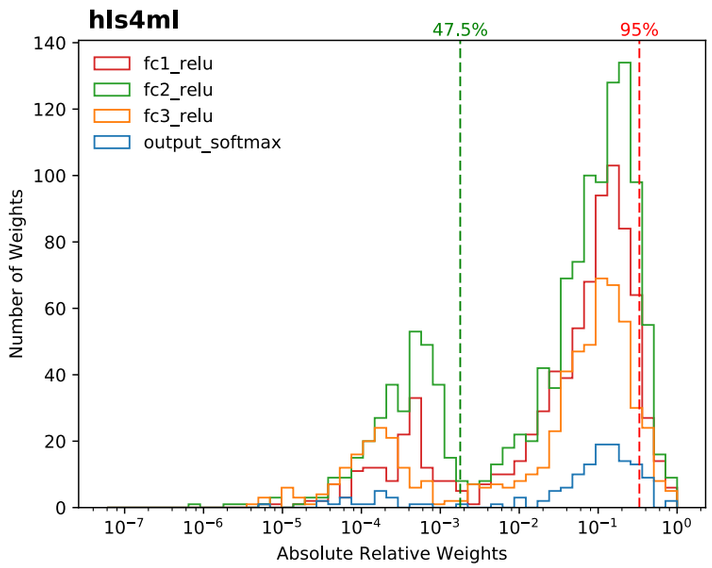


Prune

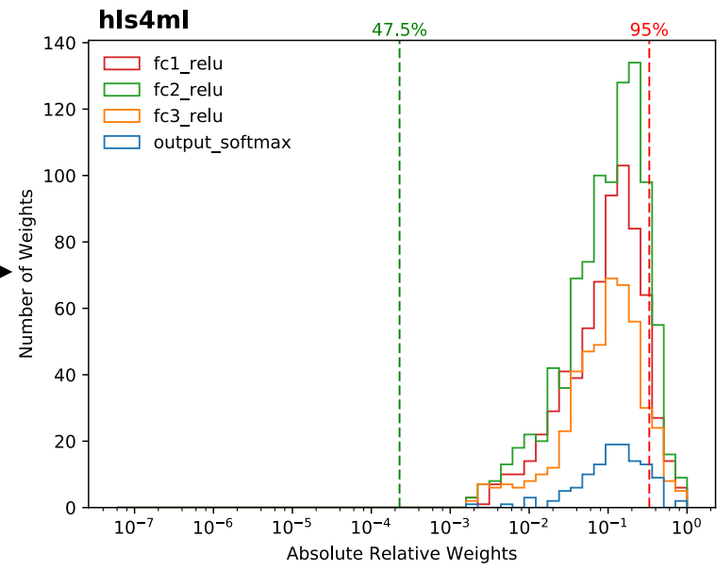


2nd iteration

Retrain with L_1



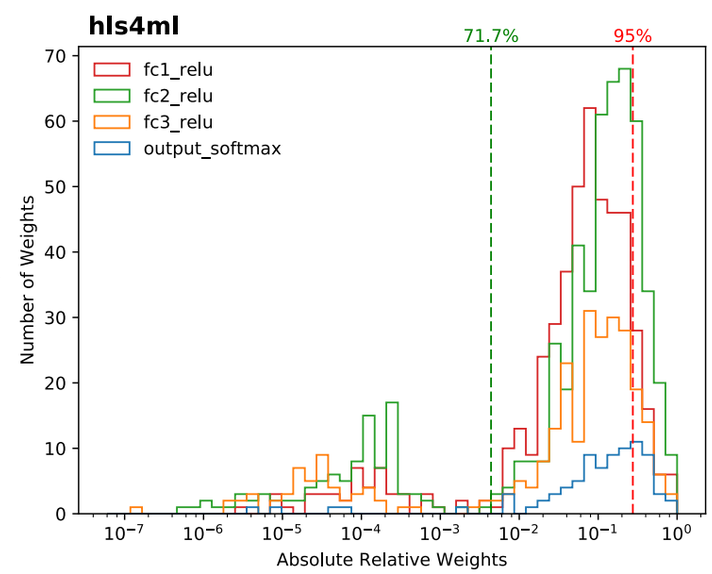
Prune



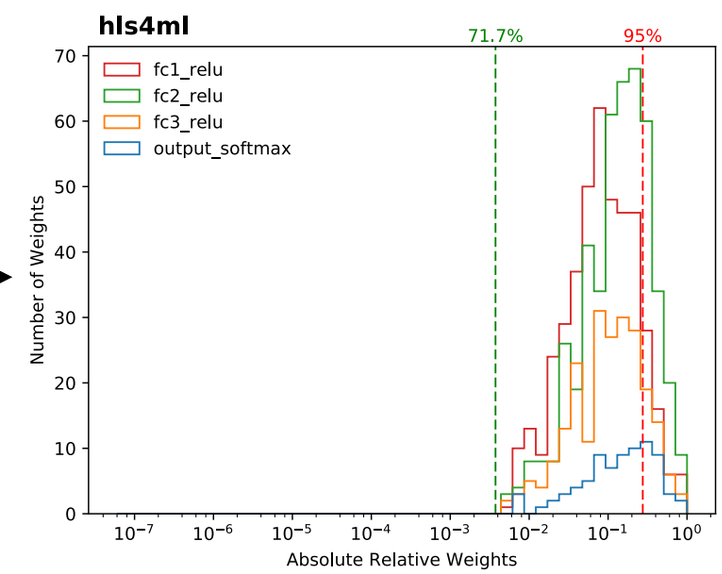
⋮

7th iteration

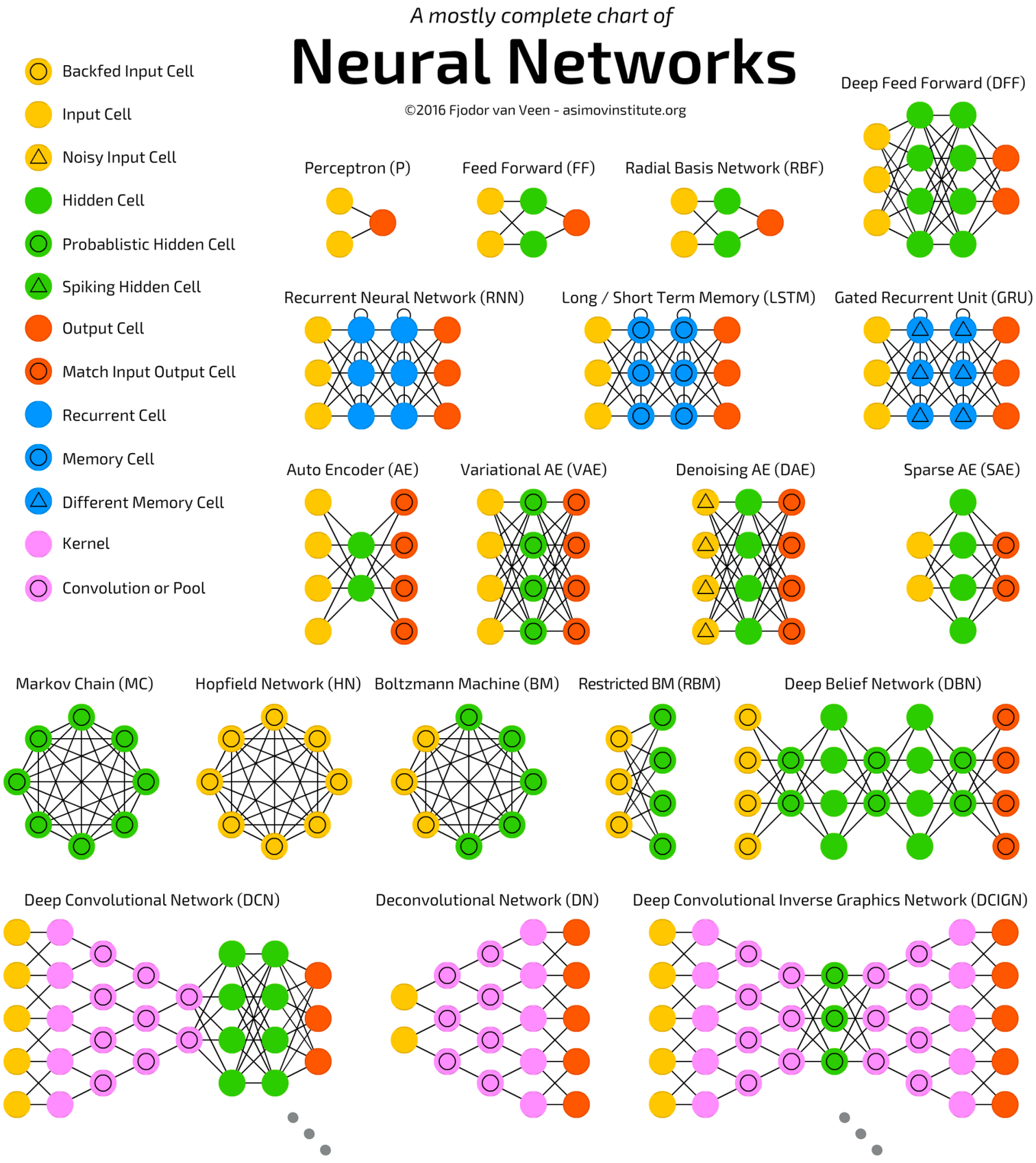
Retrain with L_1



Prune

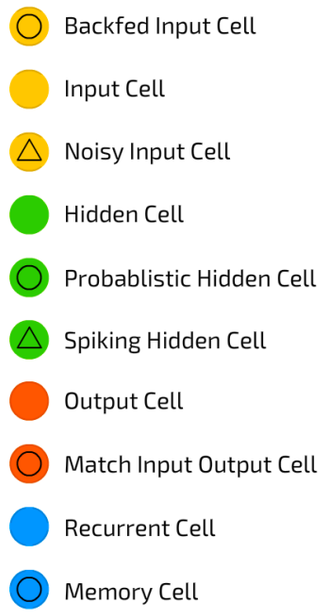


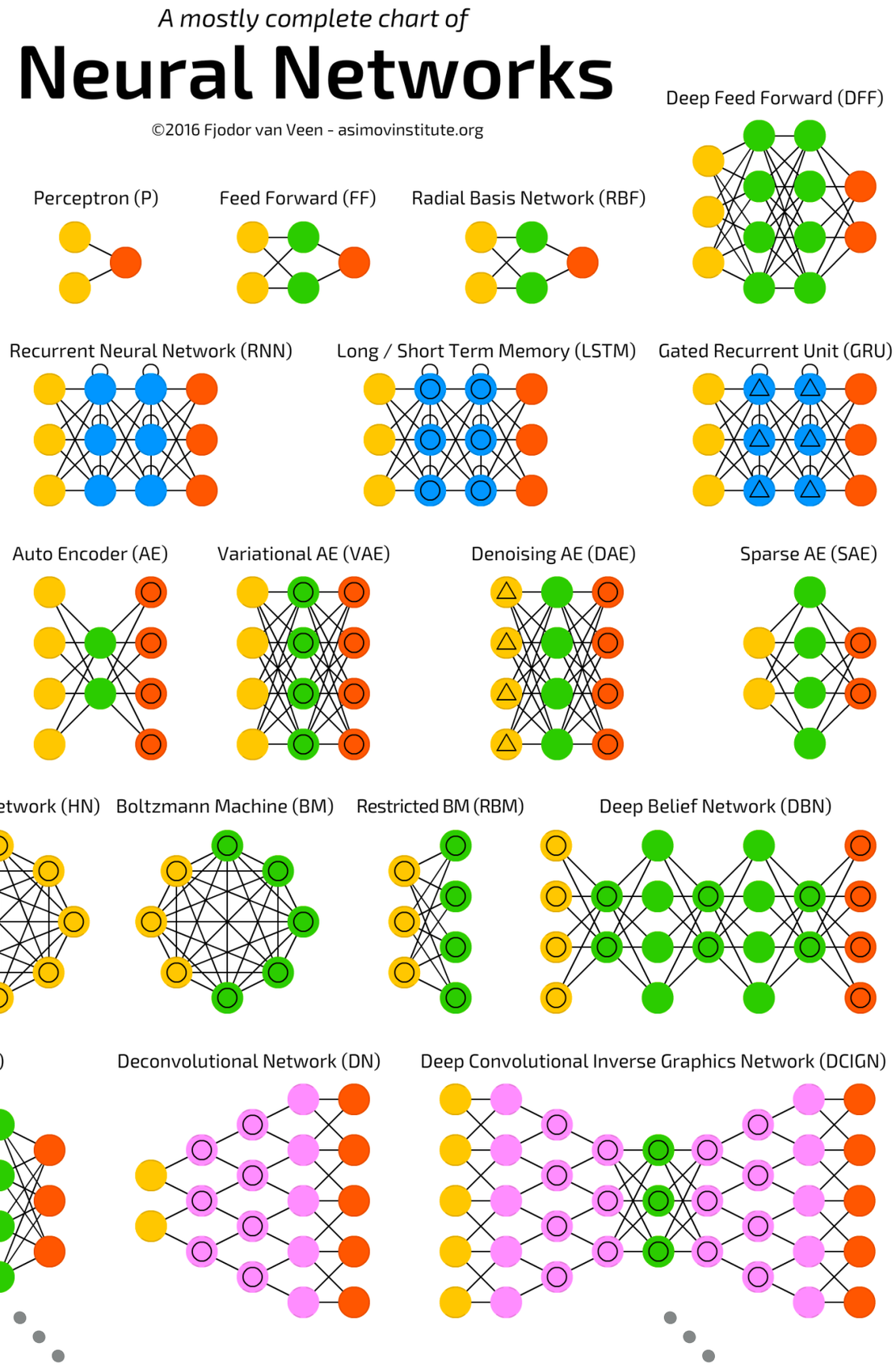
NEURAL NETWORK ZOO



NEURAL NETWORK ZOO

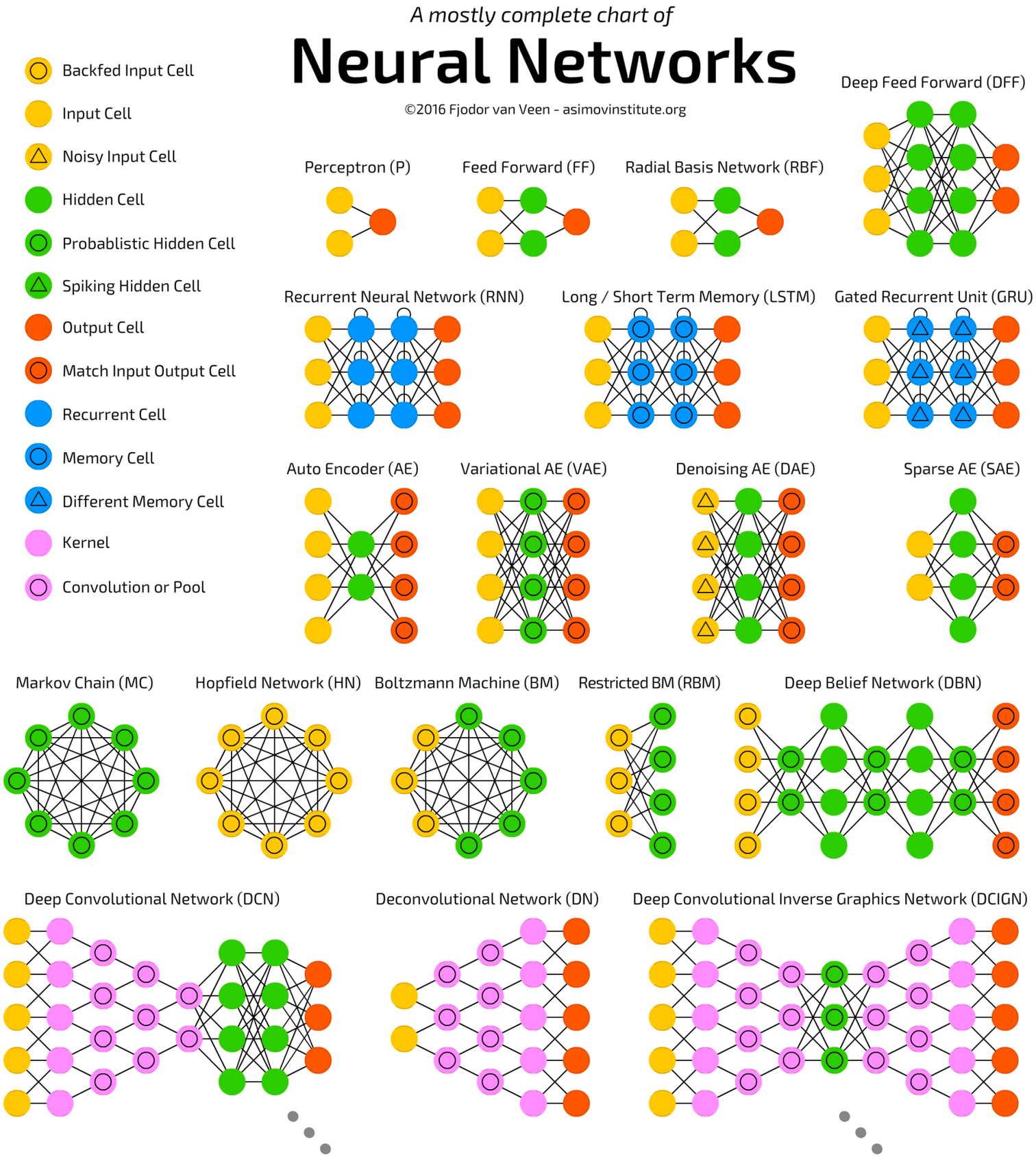
- You have a task to accomplish, which can be represented as a smooth function from your inputs to the answer you want





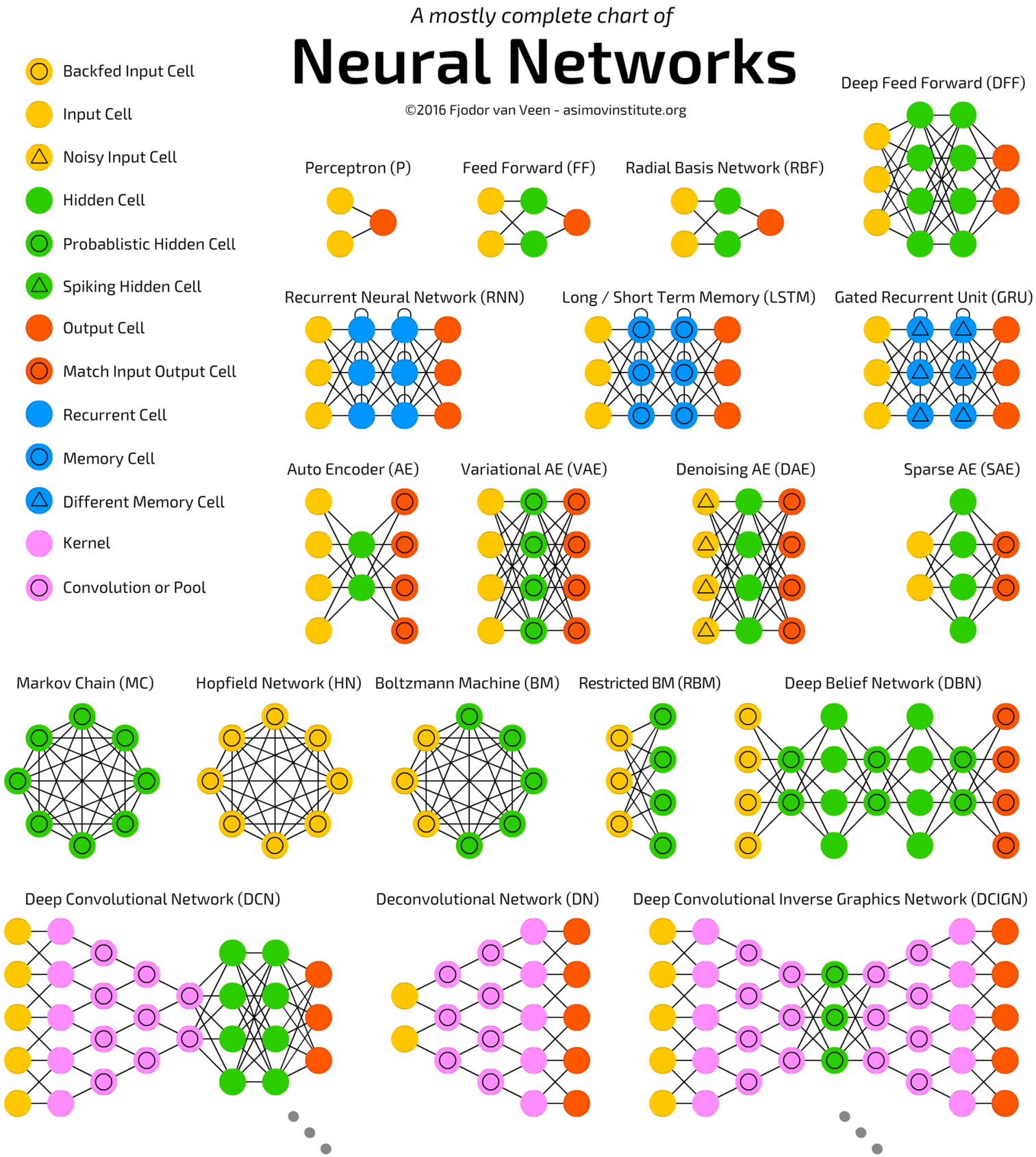
NEURAL NETWORK ZOO

- You have a task to accomplish, which can be represented as a smooth function from your inputs to the answer you want
  - Train an algorithm to learn an approximation of the optimal solution function (Machine Learning)



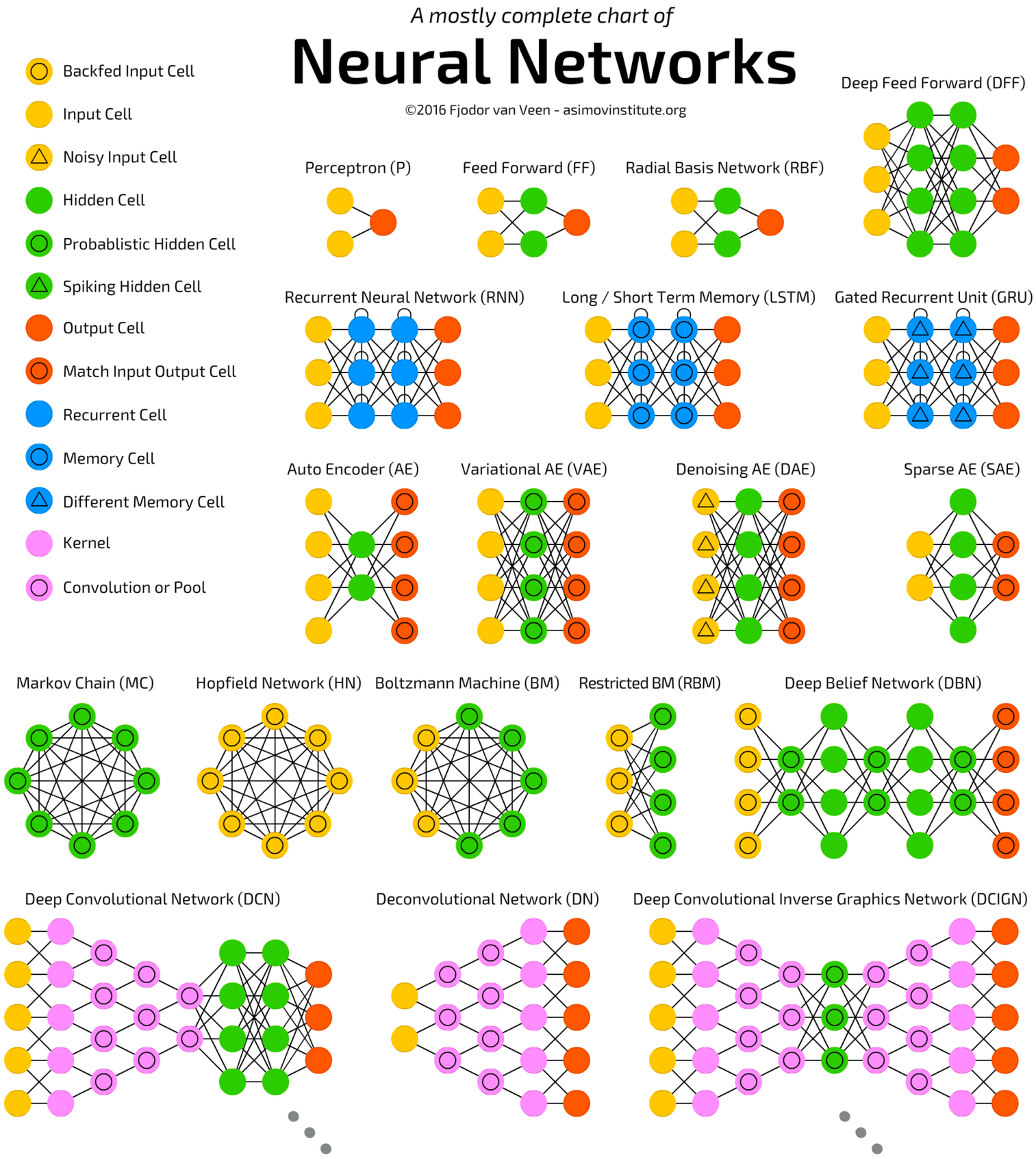
NEURAL NETWORK ZOO

- You have a task to accomplish, which can be represented as a smooth function from your inputs to the answer you want
  - Train an algorithm to learn an approximation of the optimal solution function (Machine Learning)
- NNs are the best ML solution on the market *today*



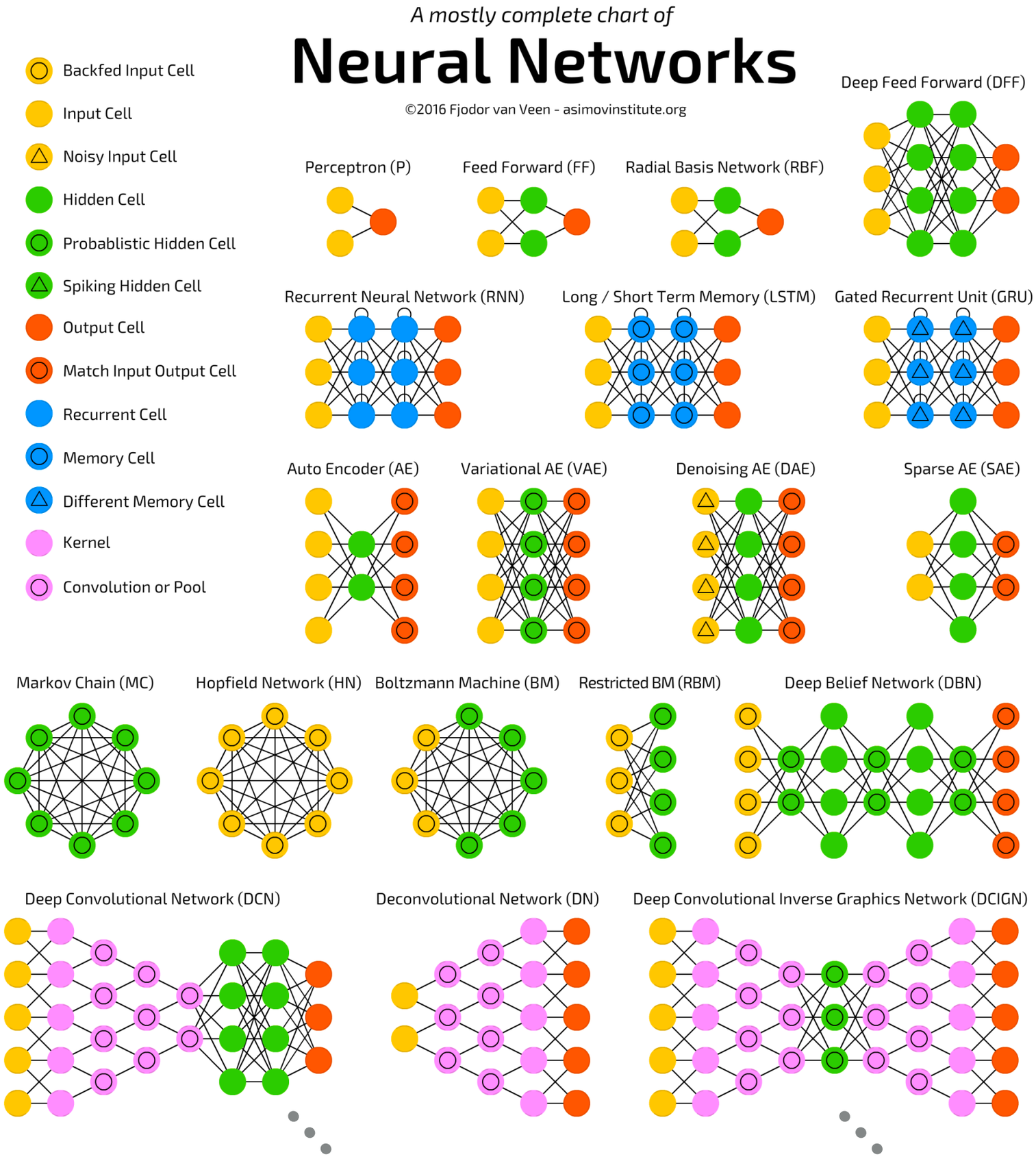
NEURAL NETWORK ZOO

- You have a task to accomplish, which can be represented as a smooth function from your inputs to the answer you want
  - Train an algorithm to learn an approximation of the optimal solution function (Machine Learning)
- NNs are the best ML solution on the market *today*
  - Each node performs a math operation on the input



NEURAL NETWORK ZOO

- You have a task to accomplish, which can be represented as a smooth function from your inputs to the answer you want
  - Train an algorithm to learn an approximation of the optimal solution function (Machine Learning)
- NNs are the best ML solution on the market *today*
  - Each node performs a math operation on the input
  - Edges represent the flow of nodes' inputs & outputs

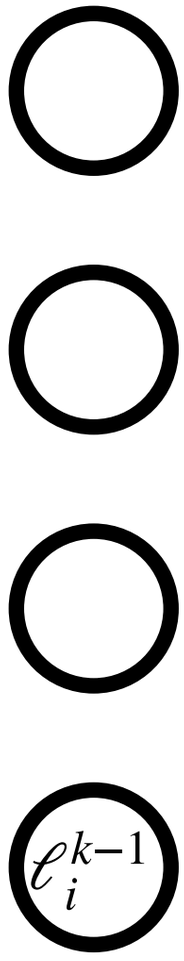


$$\ell_i^{k-1}$$

$$\ell_i^{k-1}$$

NEURAL NETWORK

- Classic feed forward architecture with some modifications responsible for revolutions in computer vision, language processing, etc.

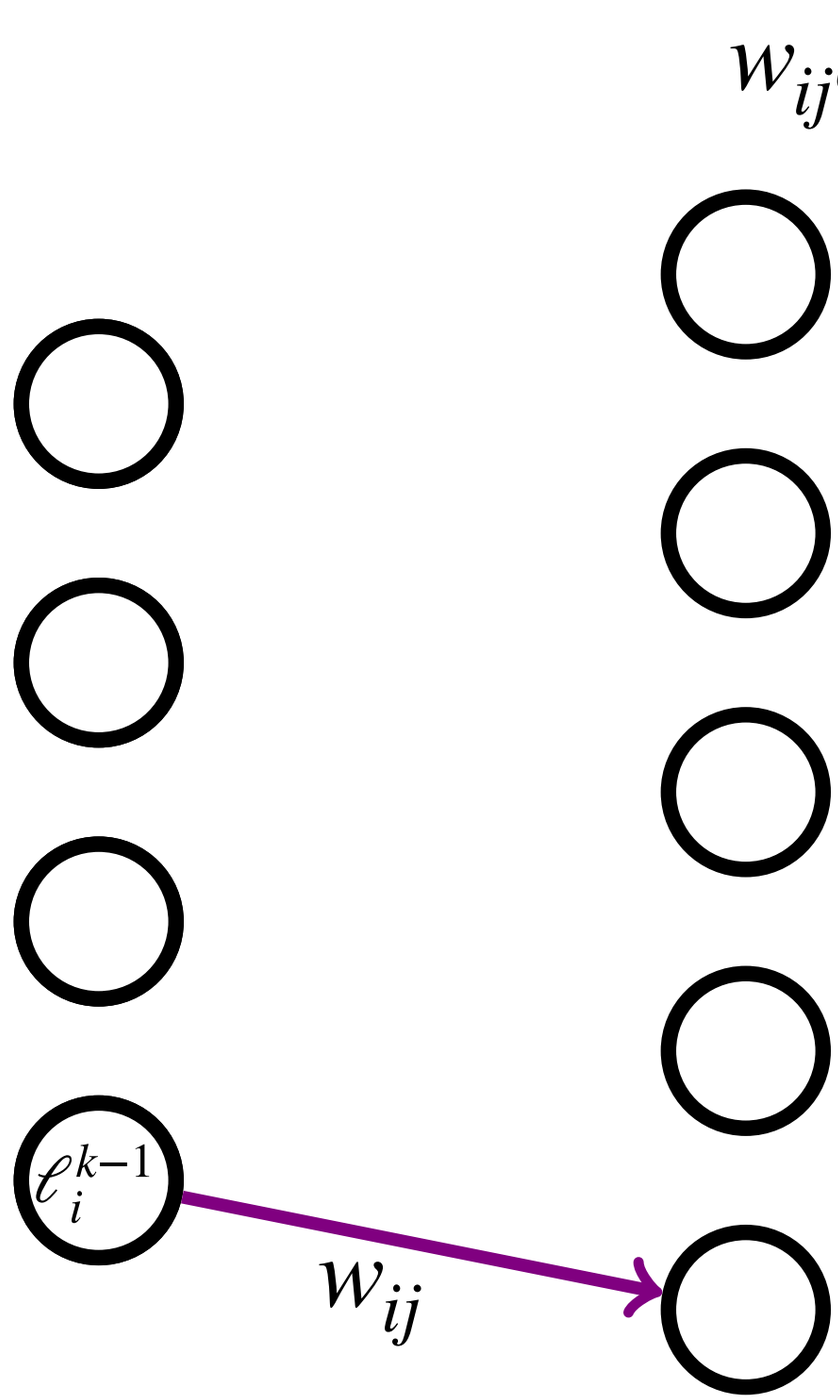


ℓ_i^{k-1}

NEURAL NETWORK

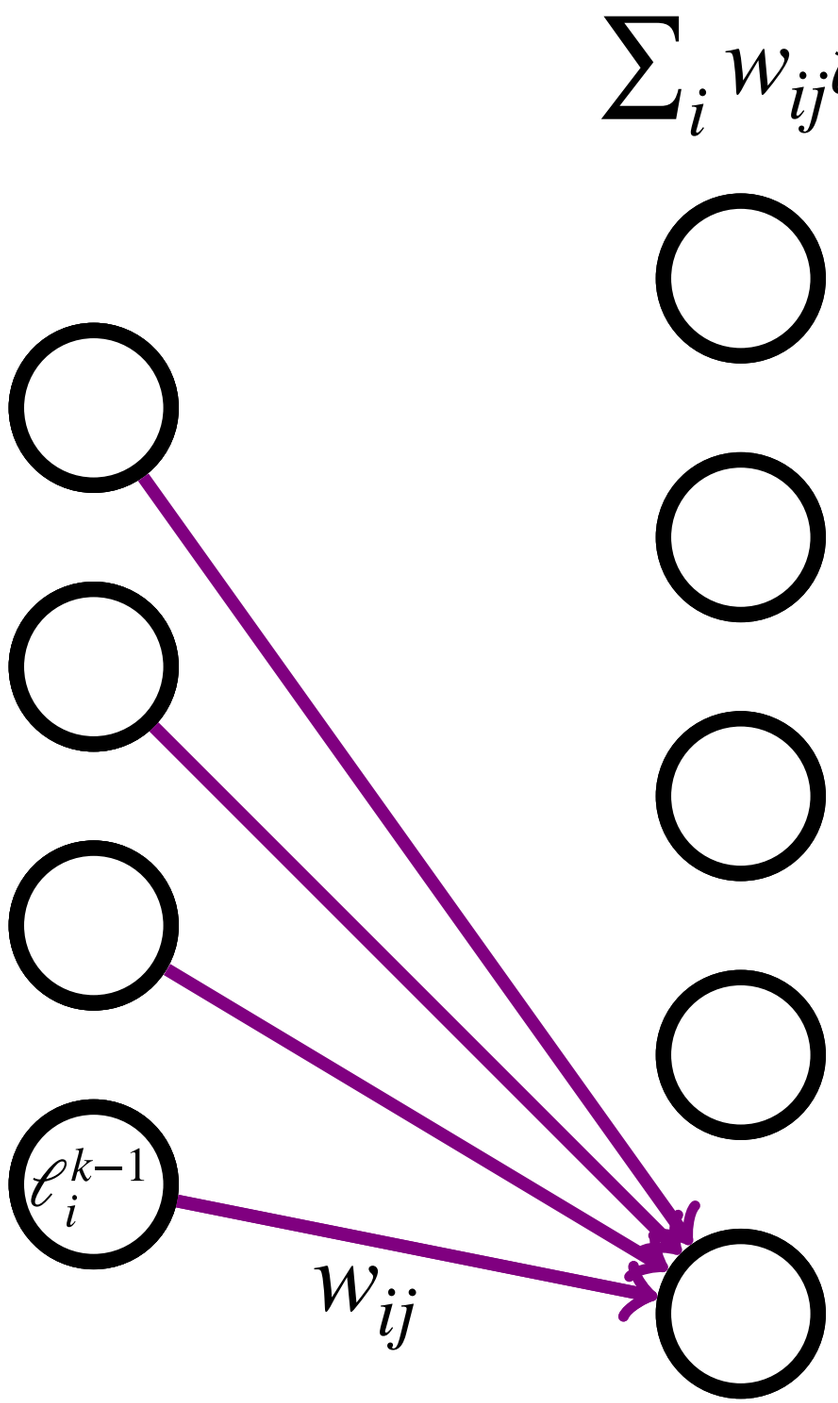
- Classic feed forward architecture with some modifications responsible for revolutions in computer vision, language processing, etc.

- Each input multiplied by a weight



NEURAL NETWORK

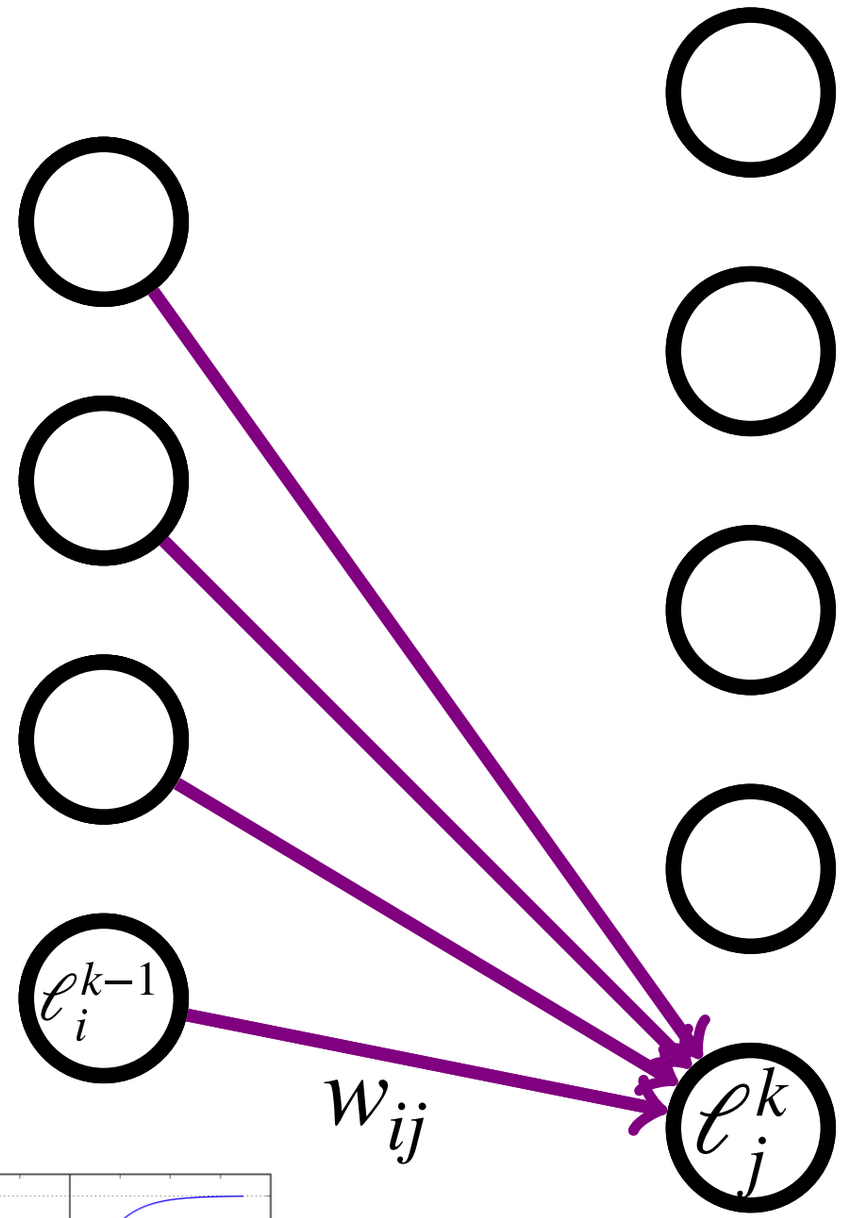
- Classic feed forward architecture with some modifications responsible for revolutions in computer vision, language processing, etc.
- Each input multiplied by a weight
- Weighted values are summed, bias is added



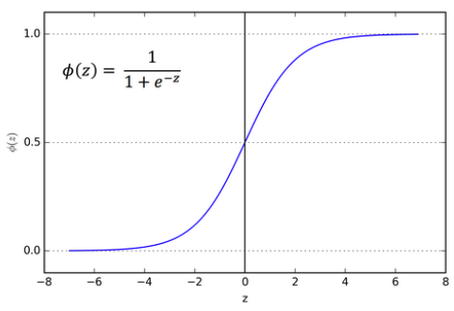
NEURAL NETWORK

- Classic feed forward architecture with some modifications responsible for revolutions in computer vision, language processing, etc.

$$\ell_j^k = \phi\left(\sum_i w_{ij}\ell_i^{k-1} + b_j\right)$$



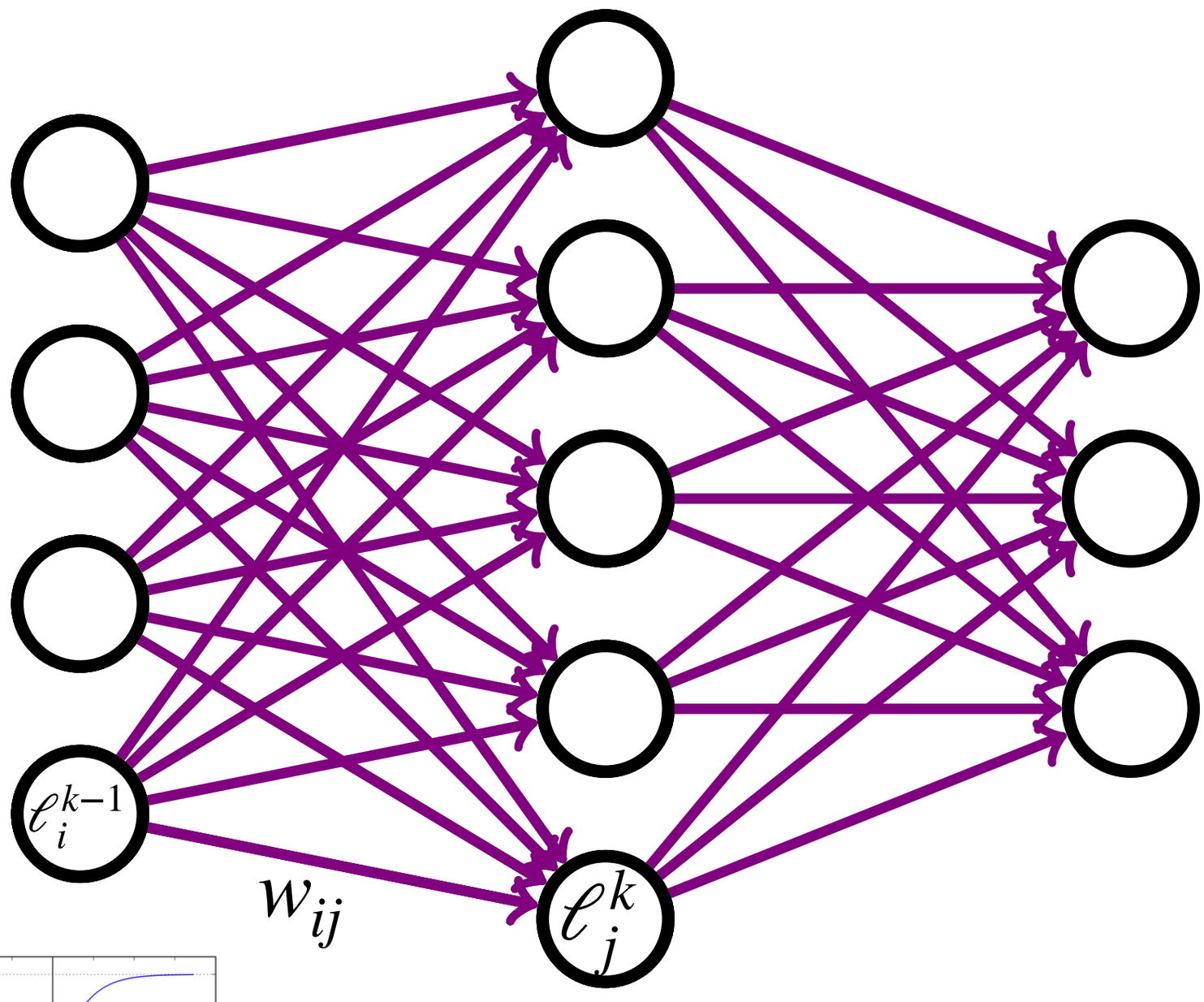
- Each input multiplied by a weight
- Weighted values are summed, bias is added
- Nonlinear activation function is applied



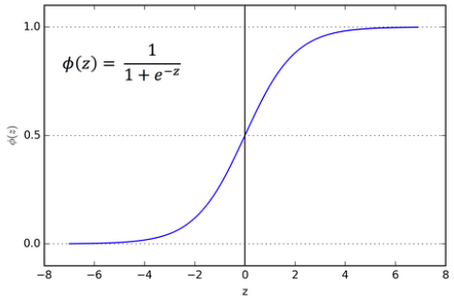
NEURAL NETWORK

- Classic feed forward architecture with some modifications responsible for revolutions in computer vision, language processing, etc.

$$\ell_j^k = \phi\left(\sum_i w_{ij}\ell_i^{k-1} + b_j\right)$$

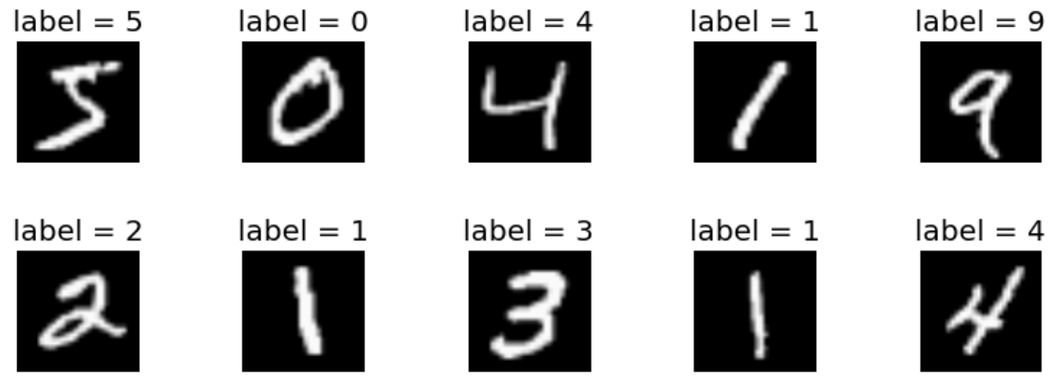


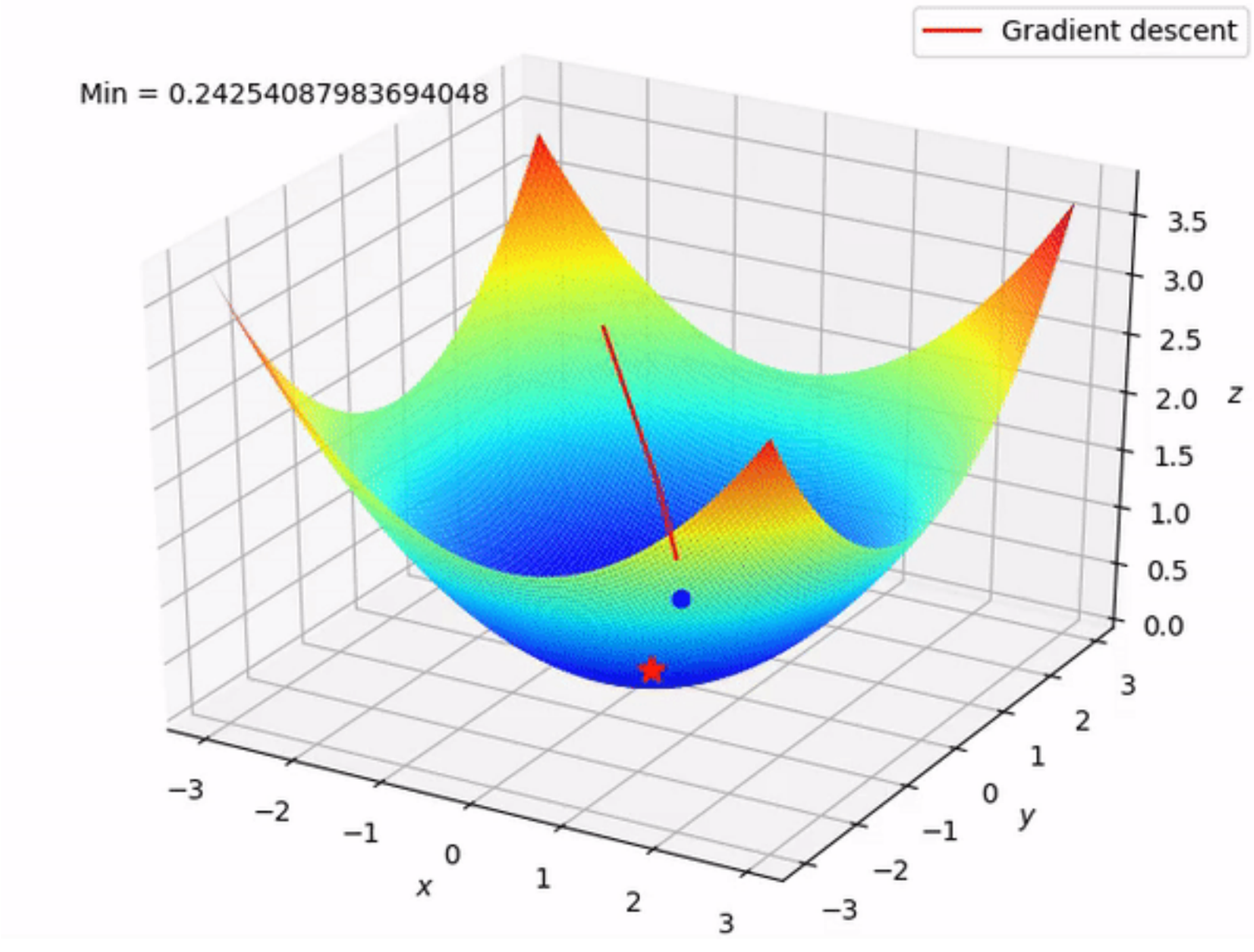
- Each input multiplied by a weight
- Weighted values are summed, bias is added
- Nonlinear activation function is applied



A sufficiently "wide" neural network can approximate any function!

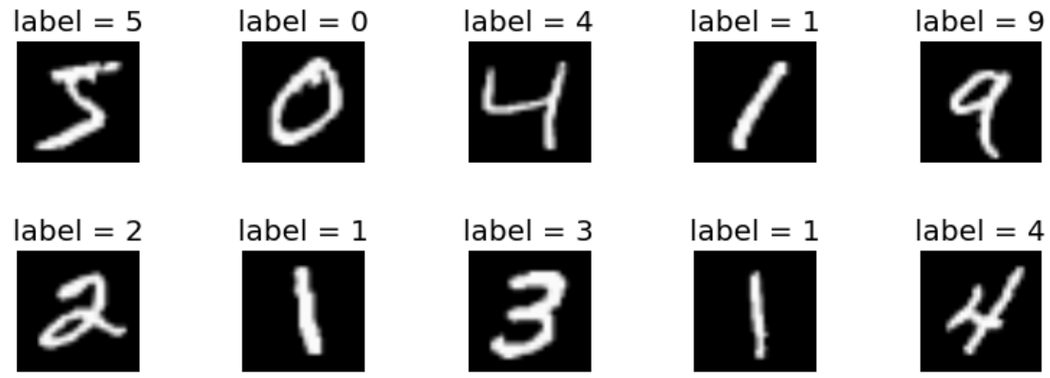
TRAINING

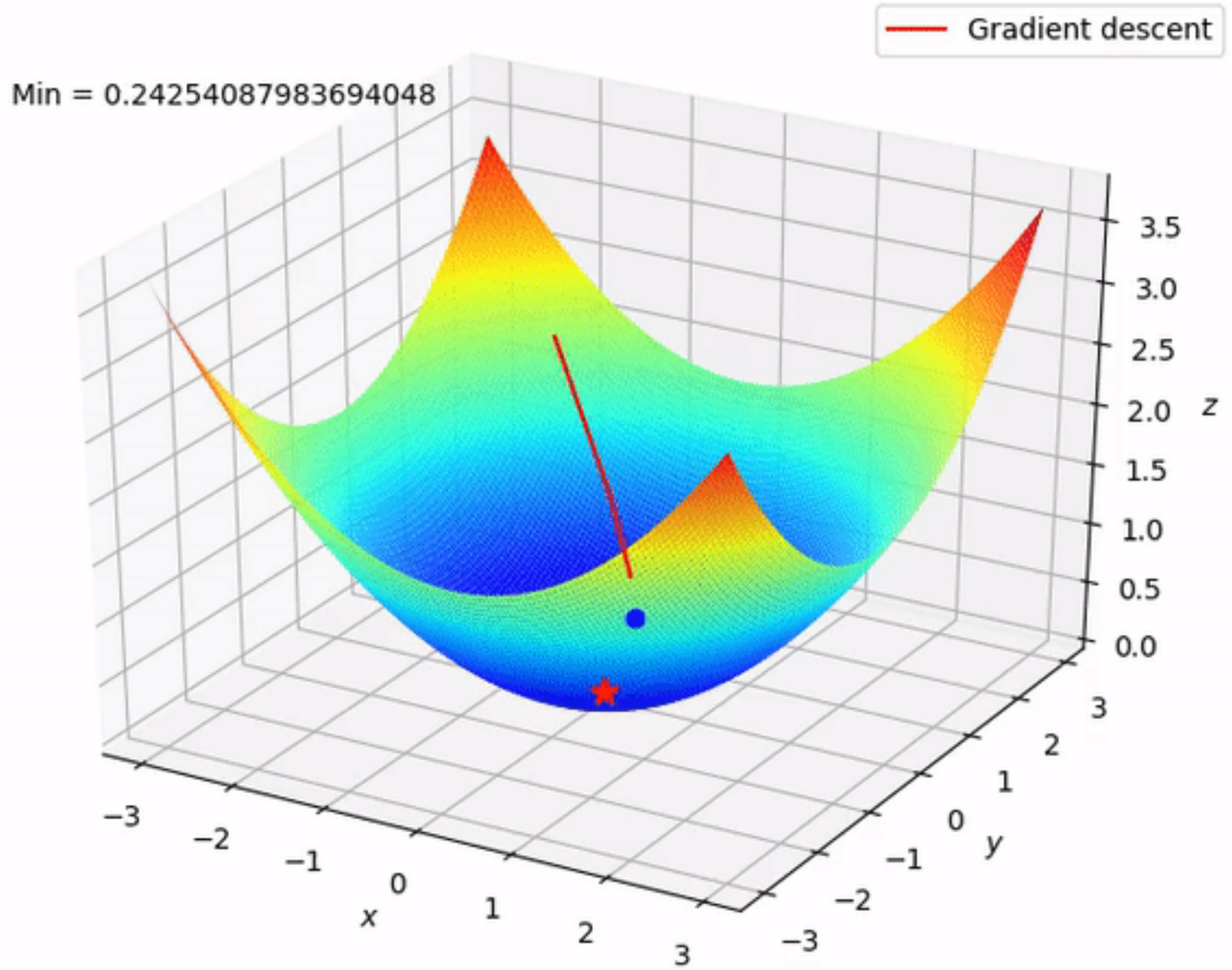

label = 5
label = 0
label = 4
label = 1
label = 9
label = 2
label = 1
label = 3
label = 1
label = 4


Gradient descent
Min = 0.24254087983694048
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
z
−3
−2
−1
0
1
2
3
x
y

TRAINING

- A network is trained by specifying inputs, targets, and a loss function
  - Target is what the network should learn for that input, can be a “truth” label (supervised) or the input itself (unsupervised)
  - Loss function quantifies how many mistakes the network makes
- Training is the minimization of the loss function by varying the network parameters





TRAINING

- A network is trained by specifying inputs, targets, and a loss function
  - Target is what the network should learn for that input, can be a “truth” label (supervised) or the input itself (unsupervised)
  - Loss function quantifies how many mistakes the network makes
- Training is the minimization of the loss function by varying the network parameters