Neural networks in FPGAs for trigger and DAQ

Jennifer Ngadiuba, Maurizio Pierini [CERN]
Javier Duarte, Sergo Jindariani, Ben Kreis, Ryan Rivera, ***Nhan Tran*** [Fermilab]
Edward Kreinar [Hawkeye 360]
Phil Harris [MIT]
Song Han [Stanford University/Google]
Zhenbin Wu [University of Illinois at Chicago]

Connecting The Dots
March 20, 2018

OUTLINE

motivation

hls4ml

case study: jet substructure
neural networks with HLS
results

status and outlook

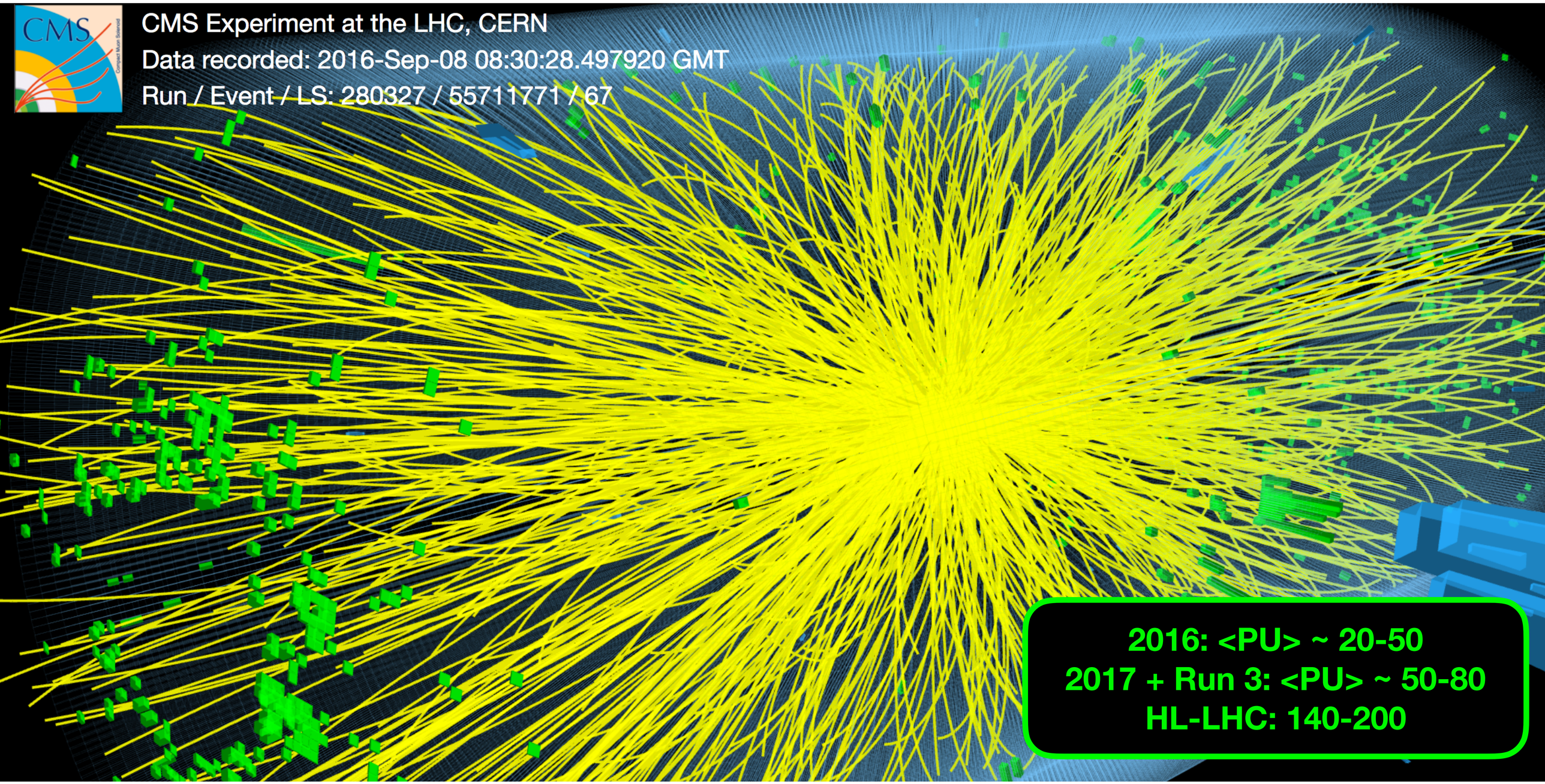
motivation

(HL-)LHC AND THE TRIGGER

A CHALLENGE TO MAINTAIN PHYSICS IN INCREASINGLY COMPLEX COLLISION ENVIRONMENTS

detector design, object performance, physics sensitivity,

lose untriggered events forever



(HL-)LHC AND THE TRIGGER

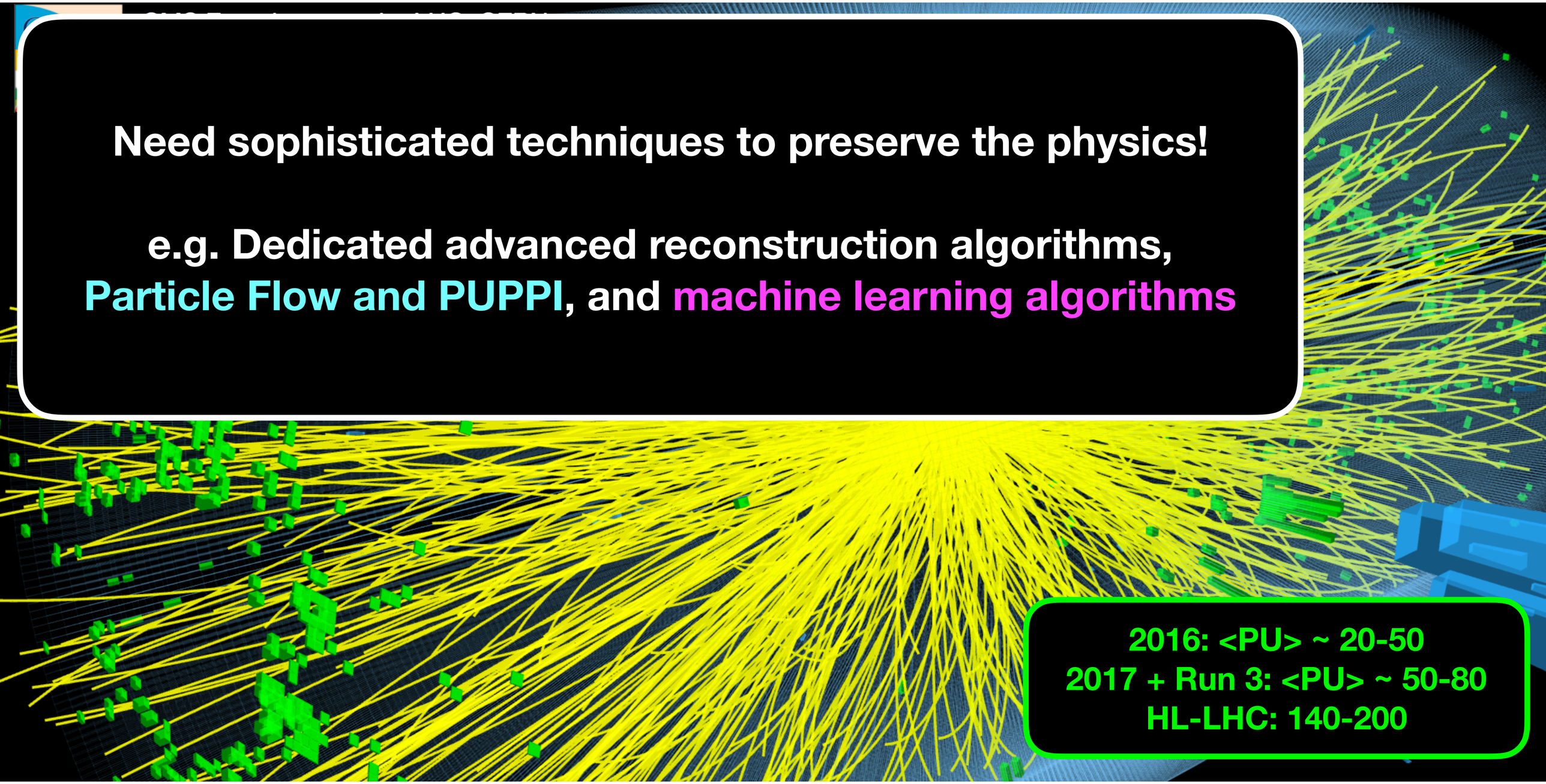
A CHALLENGE TO MAINTAIN PHYSICS IN INCREASINGLY COMPLEX COLLISION ENVIRONMENTS

detector design, object performance, physics sensitivity,

lose untriggered events forever

Need sophisticated techniques to preserve the physics!

e.g. Dedicated advanced reconstruction algorithms, Particle Flow and PUPPI, and machine learning algorithms



2016: <PU> ~ 20-50
2017 + Run 3: <PU> ~ 50-80
HL-LHC: 140-200

(HL-)LHC AND THE TRIGGER

A CHALLENGE TO MAINTAIN PHYSICS IN INCREASINGLY COMPLEX COLLISION ENVIRONMENTS

detector design, object performance, physics sensitivity,

lose untriggered events forever

Need sophisticated techniques to preserve the physics!

e.g. Dedicated advanced reconstruction algorithms, Particle Flow and PUPPI, and machine learning algorithms

See poster by Ben Kreis on PF + PUPPI in the trigger

this talk

2016: <PU> ~ 20-50
2017 + Run 3: <PU> ~ 50-80
HL-LHC: 140-200

THE TRIGGER @ ATLAS AND CMS

Goal:
reduce event rate from 40MHz to 500 Hz

How: multi-tier system

custom hardware ("L1")

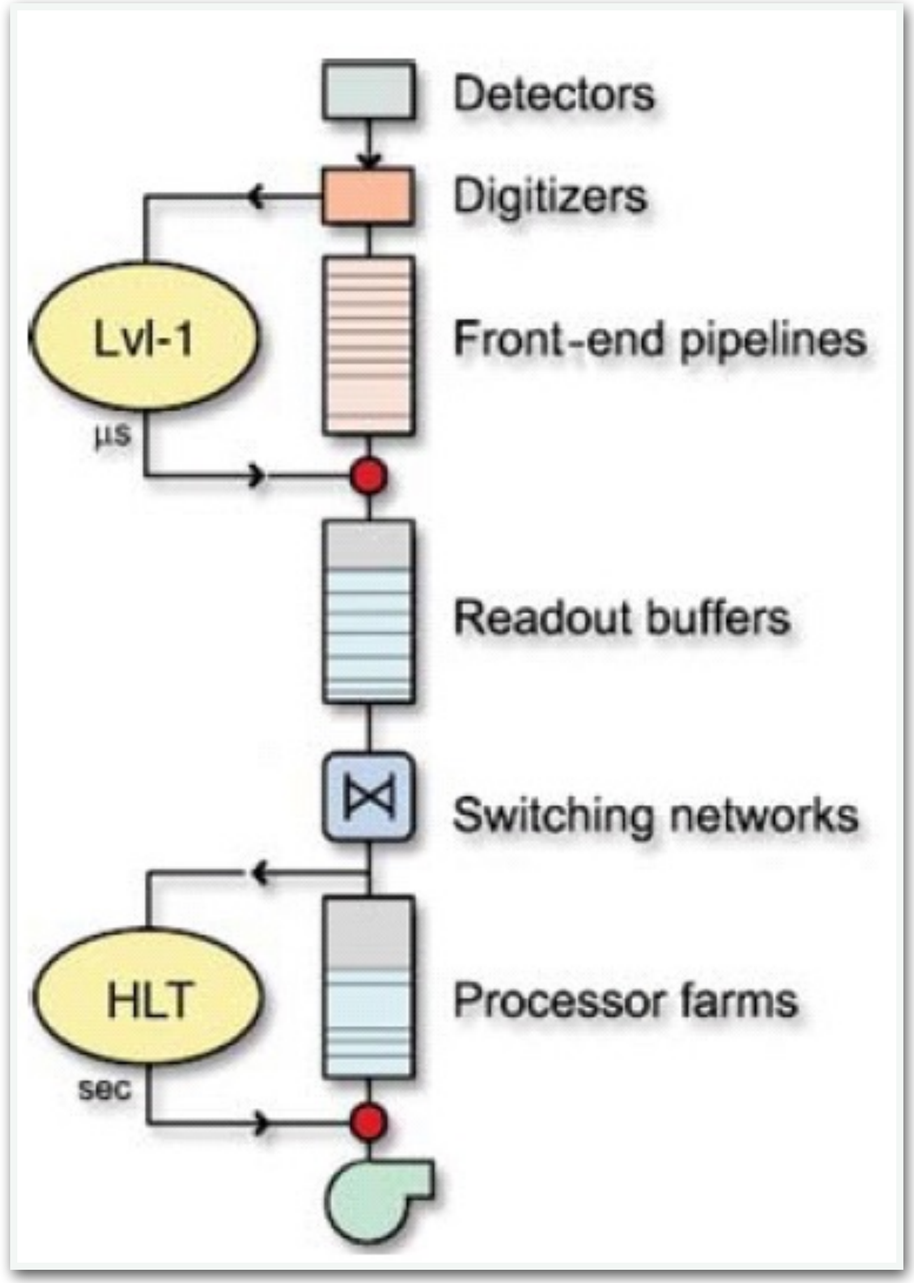
latency, O(μs)
rate in/out: 40 MHz / 100 KHz

computing farm ("HLT")

latency, O(100 ms)
rate in/out: 100 KHz / 500 Hz

n.b. all numbers approximate

For HL-LHC upgrade: latency and output rates go up ~5



THE TRIGGER @ ATLAS AND CMS

Goal:
reduce event rate from 40MHz to 500 Hz

How: multi-tier system

custom hardware ("L1")

latency, O(μs)
rate in/out: 40 MHz / 100 KHz

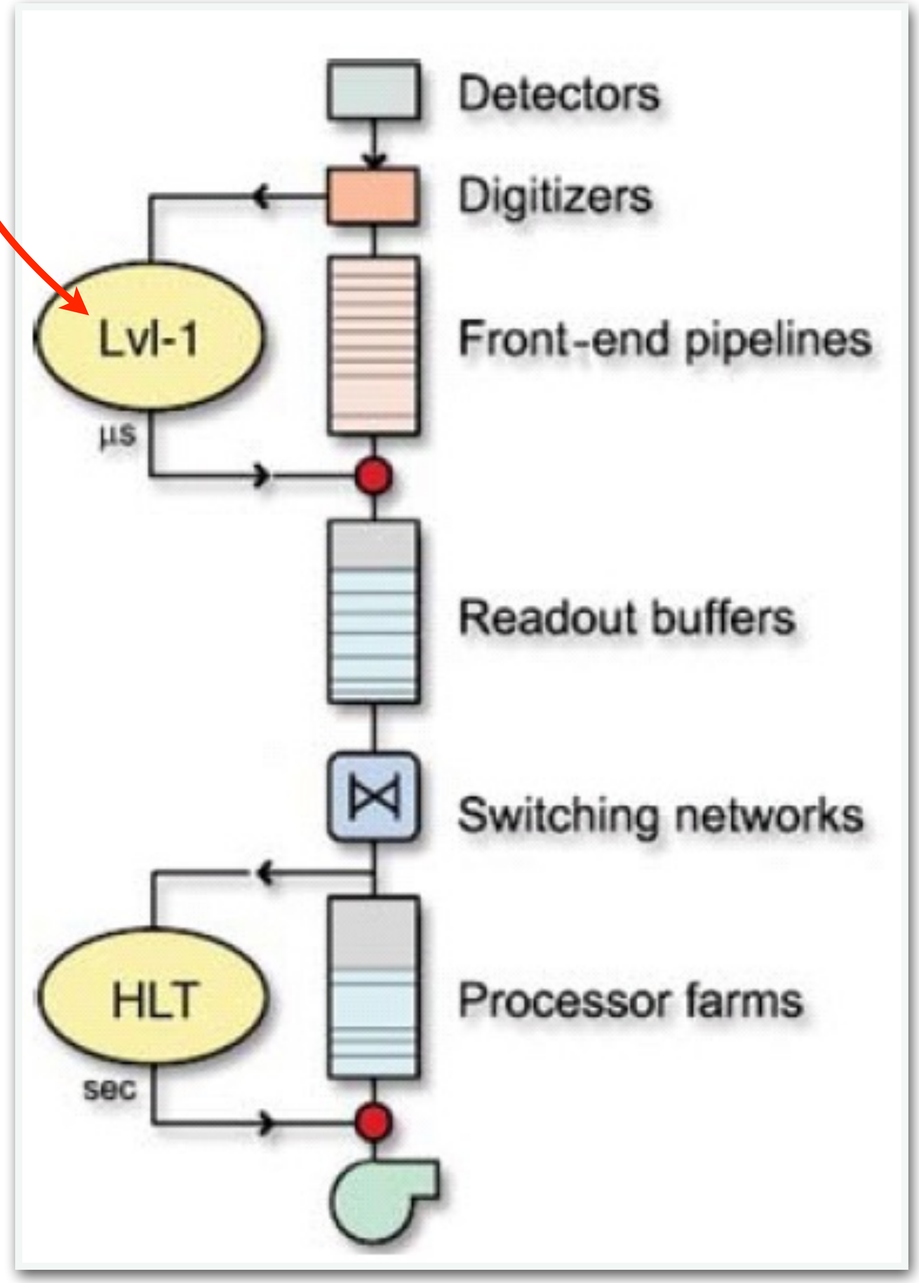
computing farm ("HLT")

latency, O(100 ms)
rate in/out: 100 KHz / 500 Hz

n.b. all numbers approximate

For HL-LHC upgrade: latency and output rates go up ~5

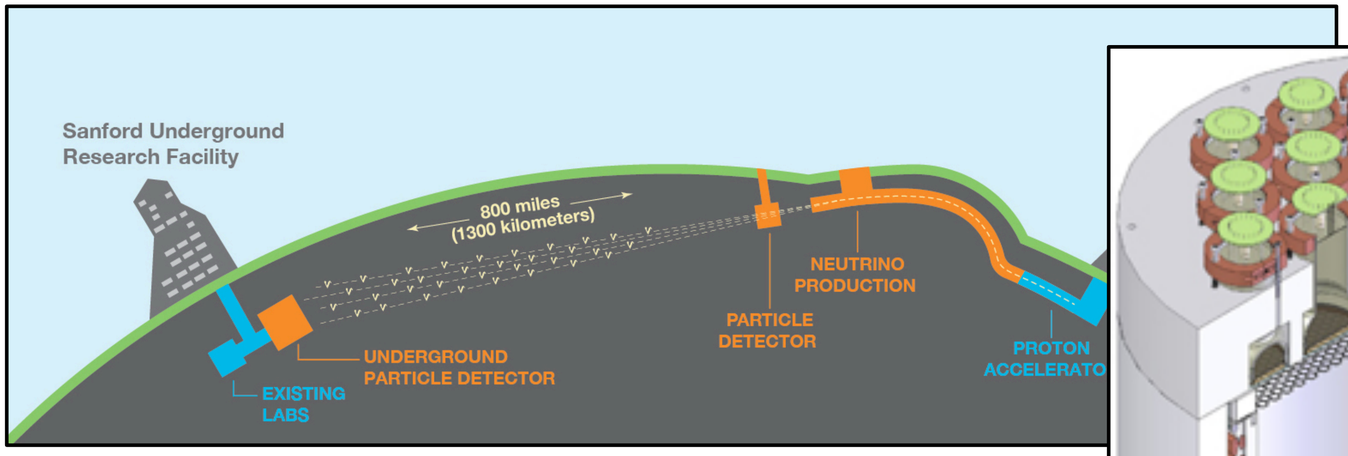
Latencies necessitate all-FPGA design!

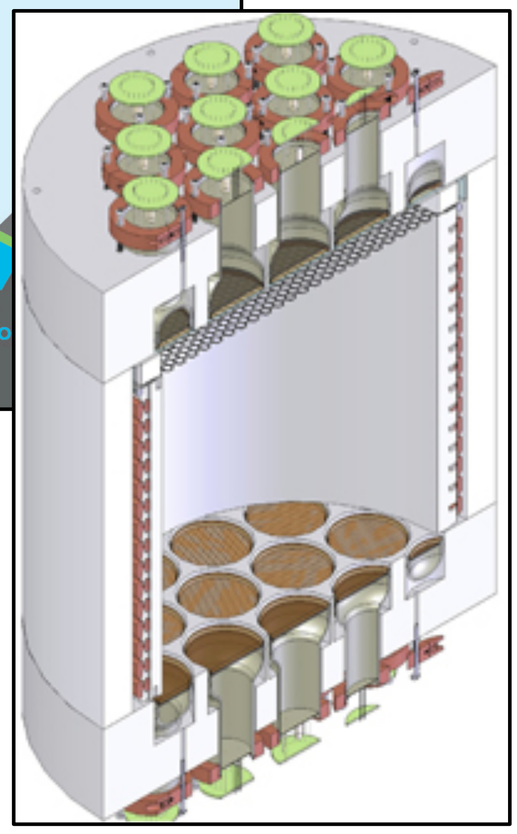


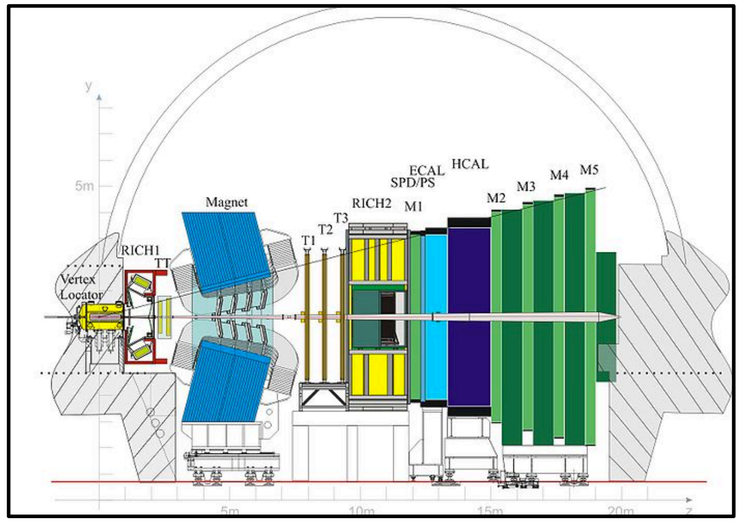
More opportunities

In the era of big science, more sophisticated triggers and DAQ systems are required

- Even in traditional “low” rate experiments
- Other LHC applications, like LHCb, and ATLAS/CMS HLT and cosmic and intensity frontier experiments







Machine learning & FPGAs

Machine learning algorithms are ubiquitous in HEP

FPGA usage broad across HEP experiments

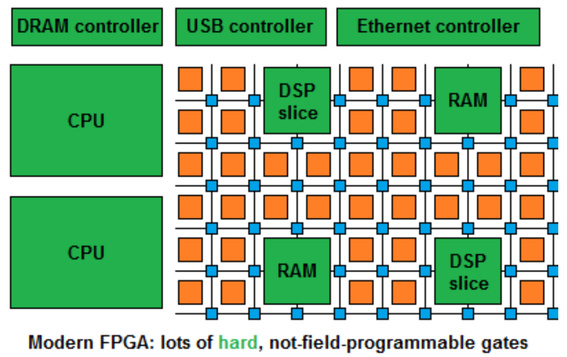
- Centered on DAQ and trigger development
- Some early adaptions of ML techniques in trigger [1]
- FPGA development becoming more accessible

High Level Synthesis, OpenCL

FPGA interest in industry is growing

Programmable hardware with structures that maps nicely onto ML architectures

FPGA
“programmable hardware”



Modern FPGA: lots of hard, not-field-programmable gates

DSPs (multiply-accumulate, etc.)
Flip Flops (registers/distributed memory)
LUTs (logic)
Block RAMs (memories)

[1] Carnes et al., https://indico.cern.ch/event/567550/contributions/2629686/

high level synthesis for machine learning

~~hlsfml~~

hls4ml

neural networks in HLS

case study: jet substructure

results

1st application:

Focus on L1 trigger

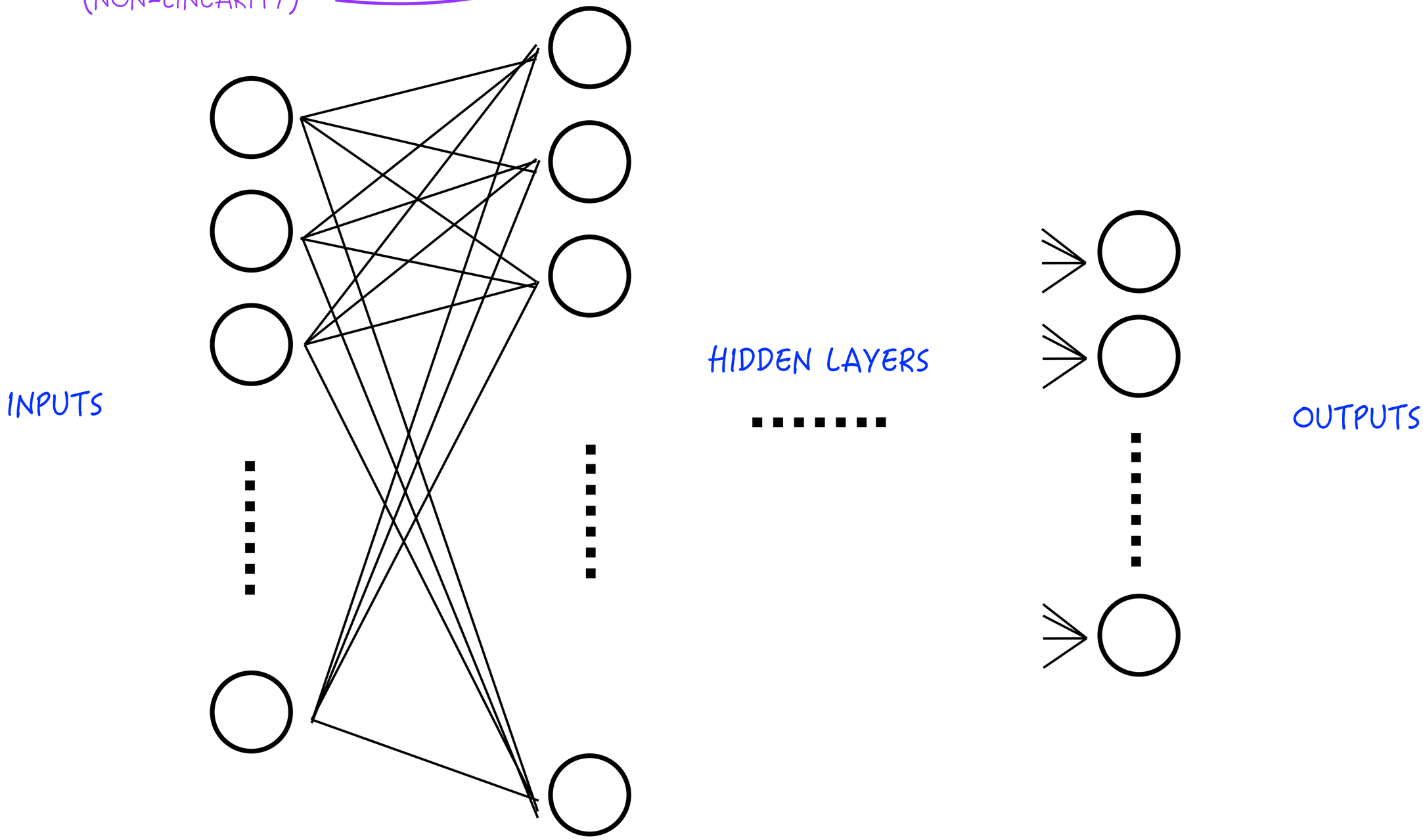
What can we do in < μs on one FPGA?

Discussion on other applications at the end

NN INFERENCE IN A NUTSHELL

$$\vec{O_j} = \Phi(\vec{I_i} \times \overleftrightarrow{W_{ij}} + \vec{b_j})$$

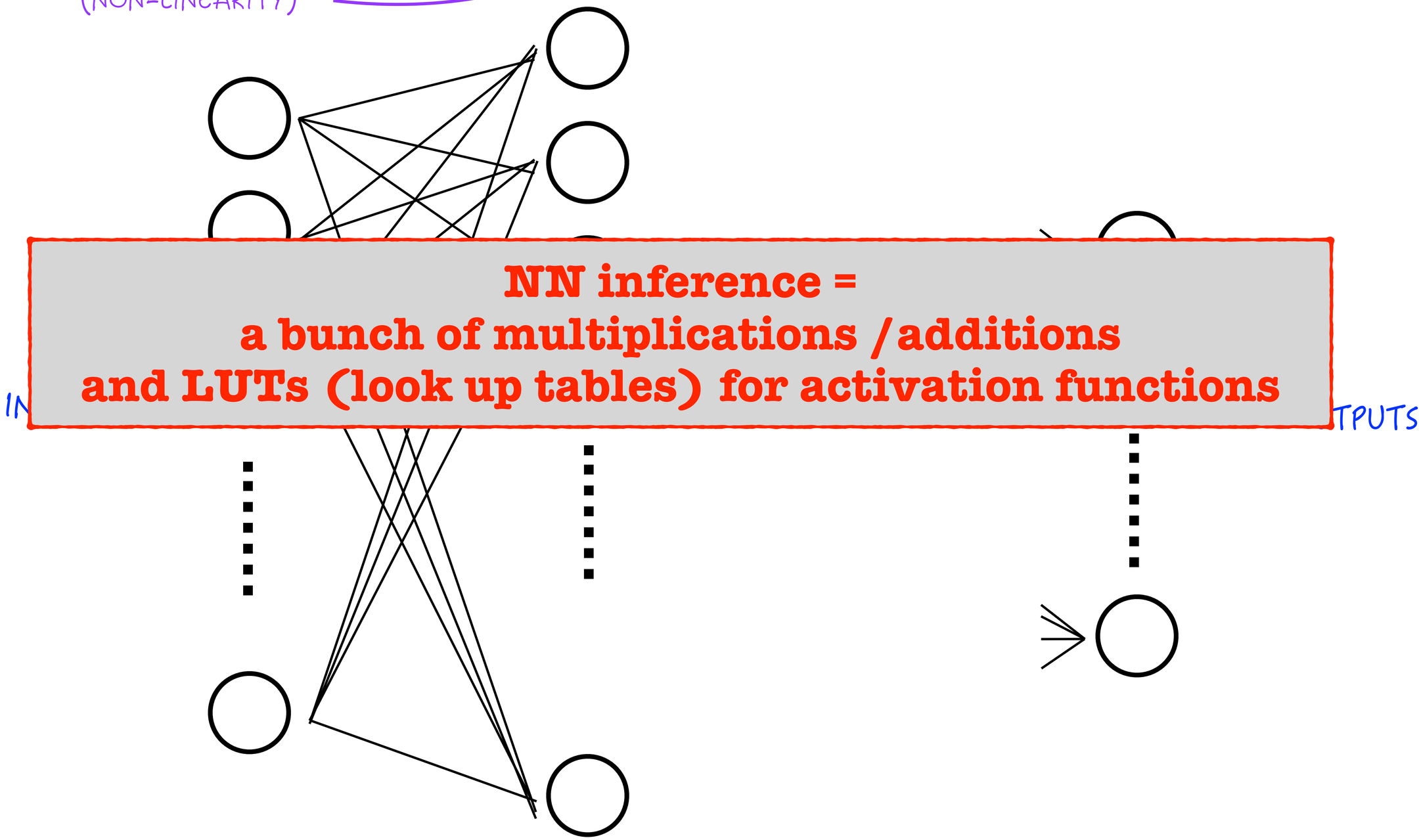
Φ = ACTIVATION FUNCTION (NON-LINEARITY)



NN INFERENCE IN A NUTSHELL

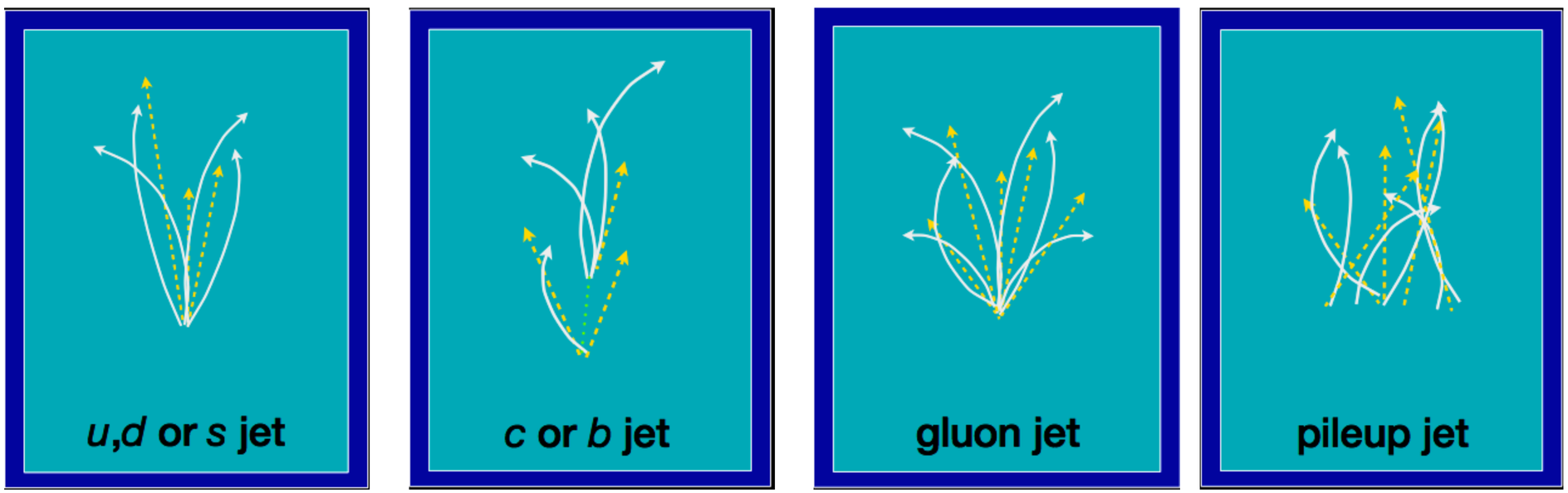
$$\vec{O_j} = \Phi(\vec{I_i} \times \overleftrightarrow{W_{ij}} + \vec{b_j})$$

Φ = ACTIVATION FUNCTION (NON-LINEARITY)



NN inference =
a bunch of multiplications /additions
and LUTs (look up tables) for activation functions

Case study: Jet substructure



Just an illustrative example, lessons are generic!

Might not be the best application, but a familiar one

ML in substructure is well-studied

CASE STUDY: JET SUBSTRUCTURE

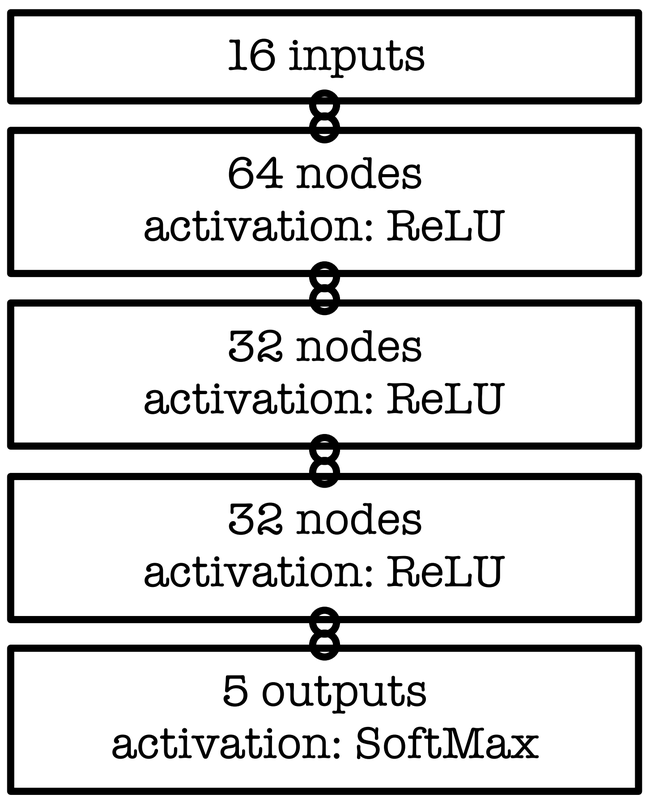
5 output multi-classifier:

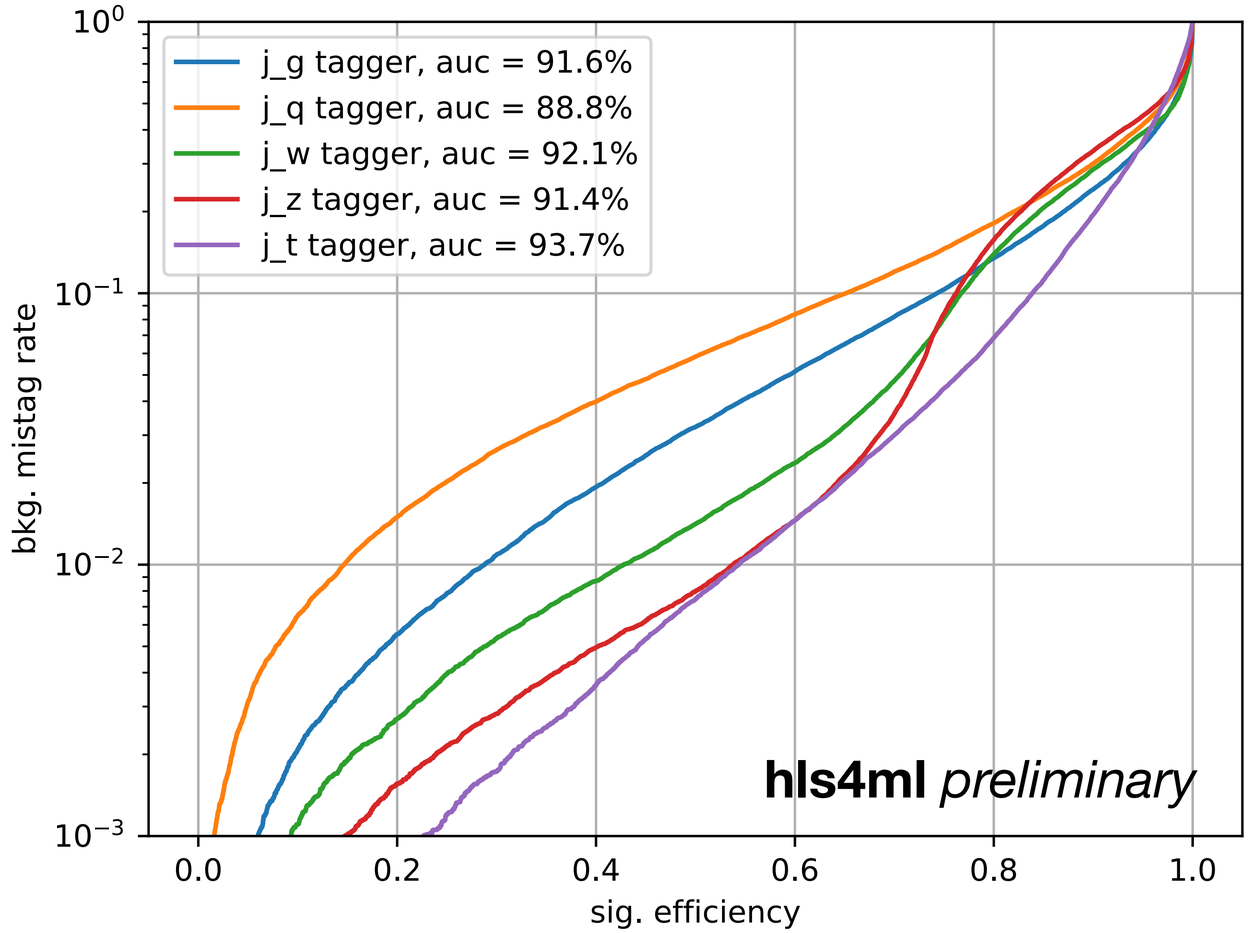
Does a jet originate from a quark, gluon, W/Z boson, top quark?

Network architecture

16 expert inputs

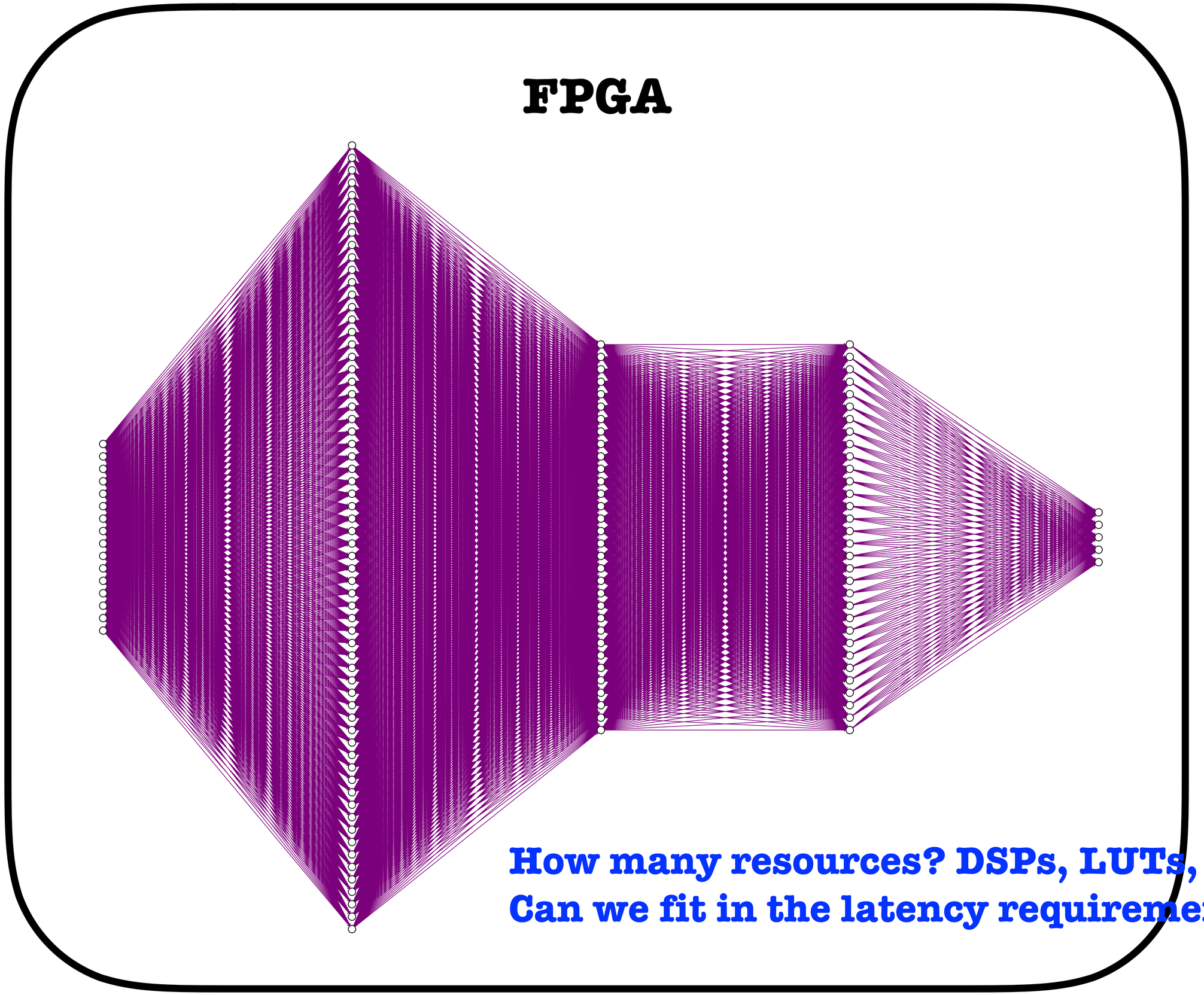
- jet masses, multiplicity
- ECFs (β=0,1,2)





Fully connected deep neural network

ML IN FPGAs?


FPGA
How many resources? DSPs, LUTs, FFs?
Can we fit in the latency requirements?

(Energy) Efficient Neural Networks

Emergent engineering field, efficient implementation of NN architecture

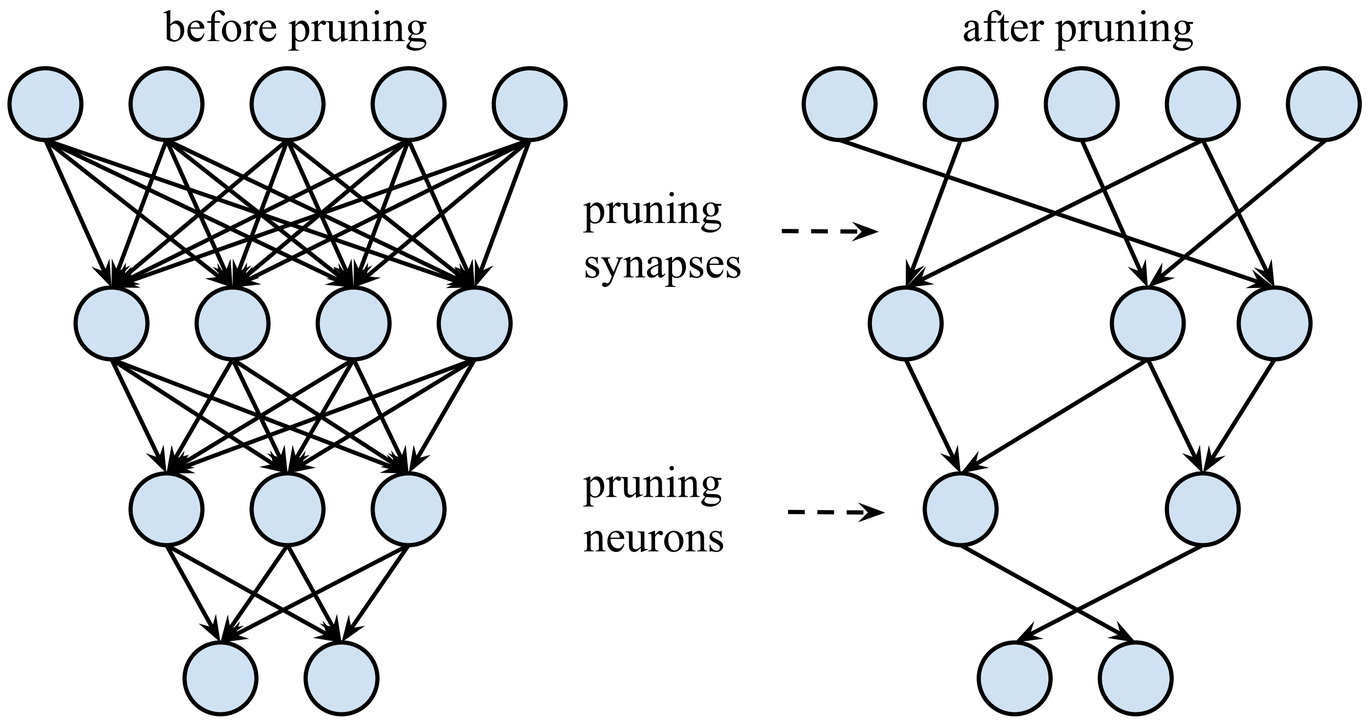
Compression:

maintain the same performance while removing low weight synapses and neurons (many schemes)

Quantization:

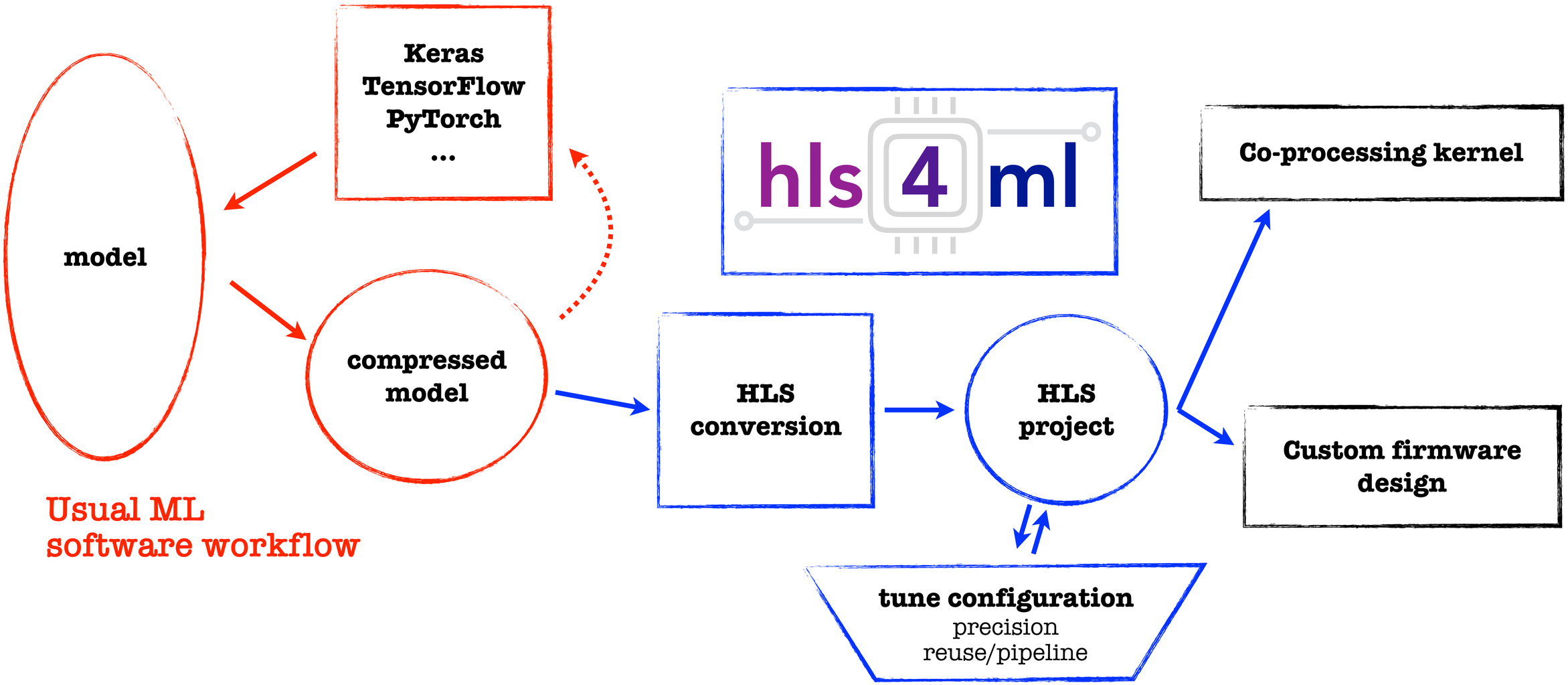
32-bit floating point math is overkill

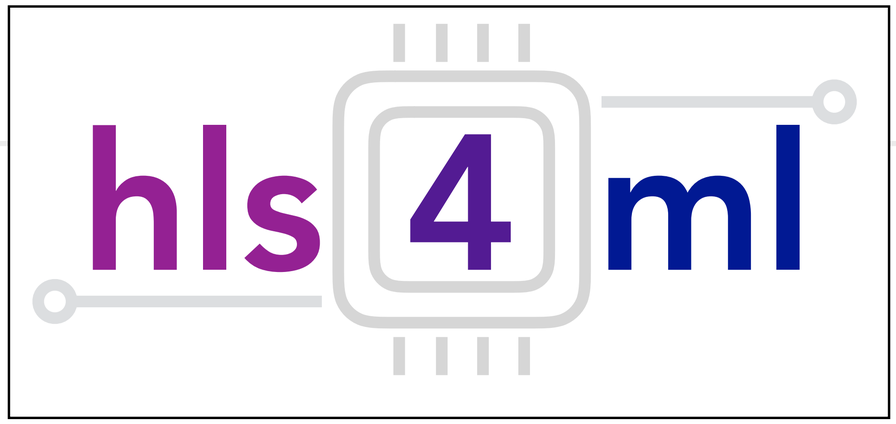
20-bit, 18-bit, …? fixed point, integers? binarized NNs?



For further reading, start here: https://arxiv.org/pdf/1510.00149v5.pdf

PROJECT OVERVIEW





Translation in one line!

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

```
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>
```

} Keras inputs

IOType: parallelize or serialize

ReuseFactor: how much to parallelize

DefaultPrecision: inputs, weights, biases

hls4ml

Translation in one line!

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

```
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>
```

} Keras inputs

IOType: parallelize or serialize

ReuseFactor: how much to par

DefaultPrecision: inputs, weig

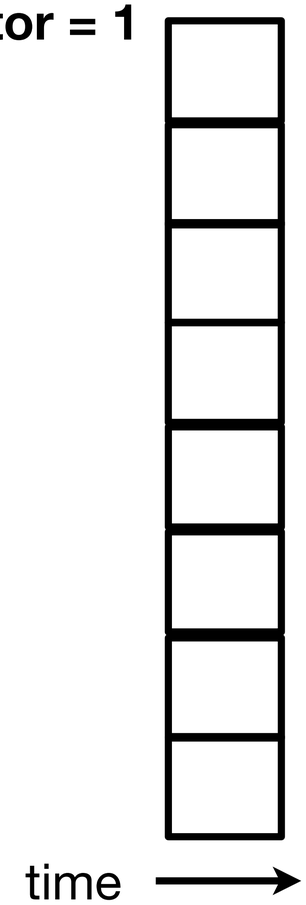
FIRMWARE IN MINUTES,
FAST DEVELOPMENT CYCLES!

TUNING YOUR NETWORK: PARALLELIZATION

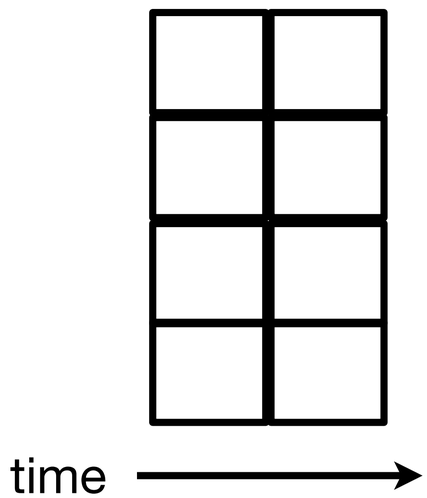
ReuseFactor: how much to parallelize operations a hidden layer

# of multiplications per clock (DSPs usage)

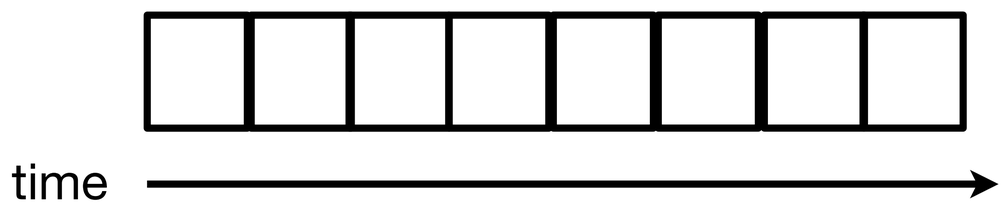
parallel
8 multipliers in 1 clock
reuse factor = 1



parallel
4 multipliers in 2 clocks
reuse factor = 2



serial
1 multiplier in 8 clocks



less resources/less throughput

defines the Pipeline Interval = when new inputs are introduced to algo

Study details

Xilinx Vivado 2017.2

Clock frequency: 200 MHz

FPGA: Xilinx Kintex Ultrascale (XCKU115-FLVB2104)

[We'll come back to these later]

A note on inputs:

We assume network inputs have already been computed

Not a good assumption in the jet substructure case

We ignore this here, but the choice to use "raw" or "expert" features will depend on the problem

On the coming plots: resource usage comes from HLS estimates

discussion on differences w.r.t. implementation later

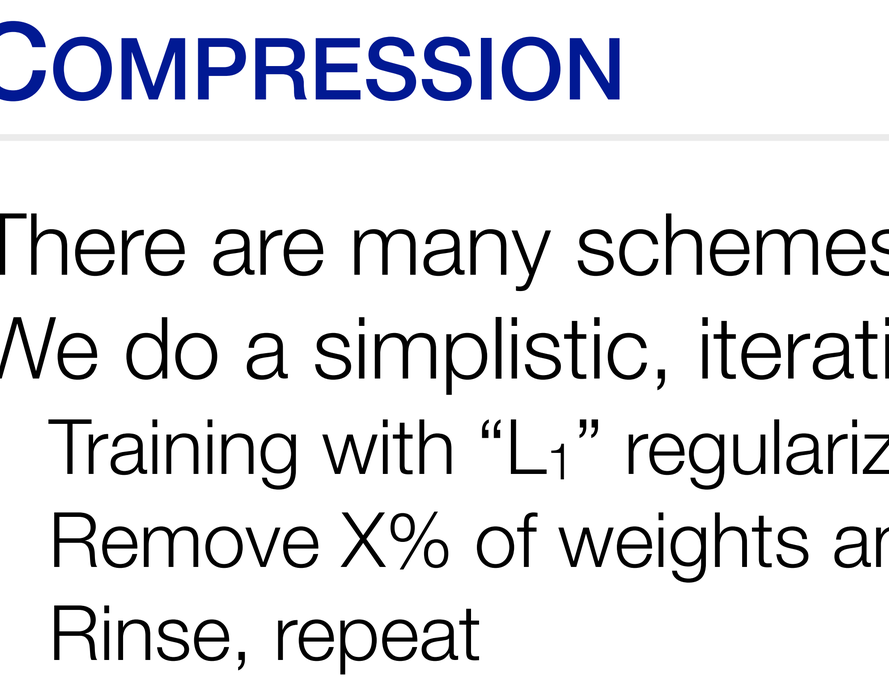
COMPRESSION

There are many schemes for compression

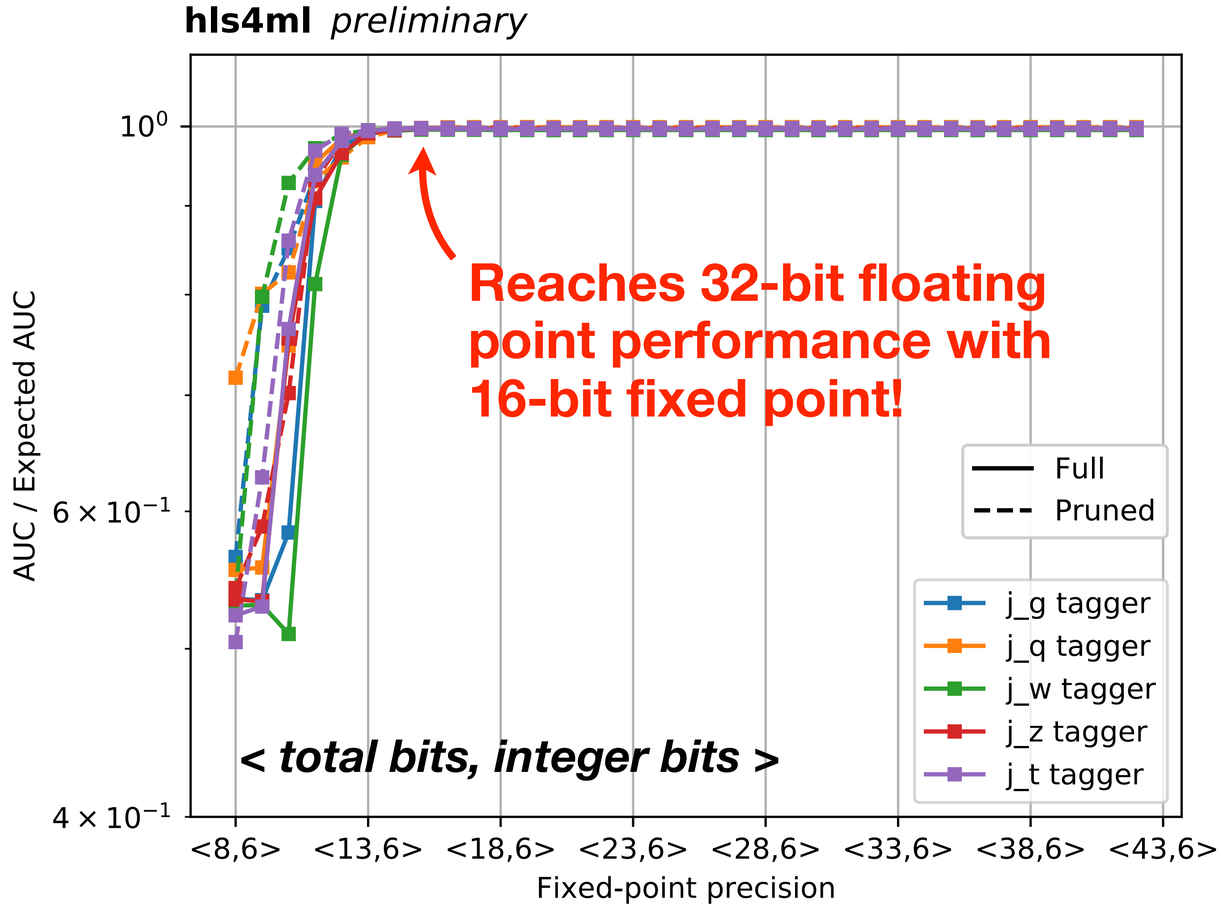
We do a simplistic, iterative version

- Training with "L_1" regularization, down-weight unimportant synapses
- Remove X% of weights and retrain
- Rinse, repeat

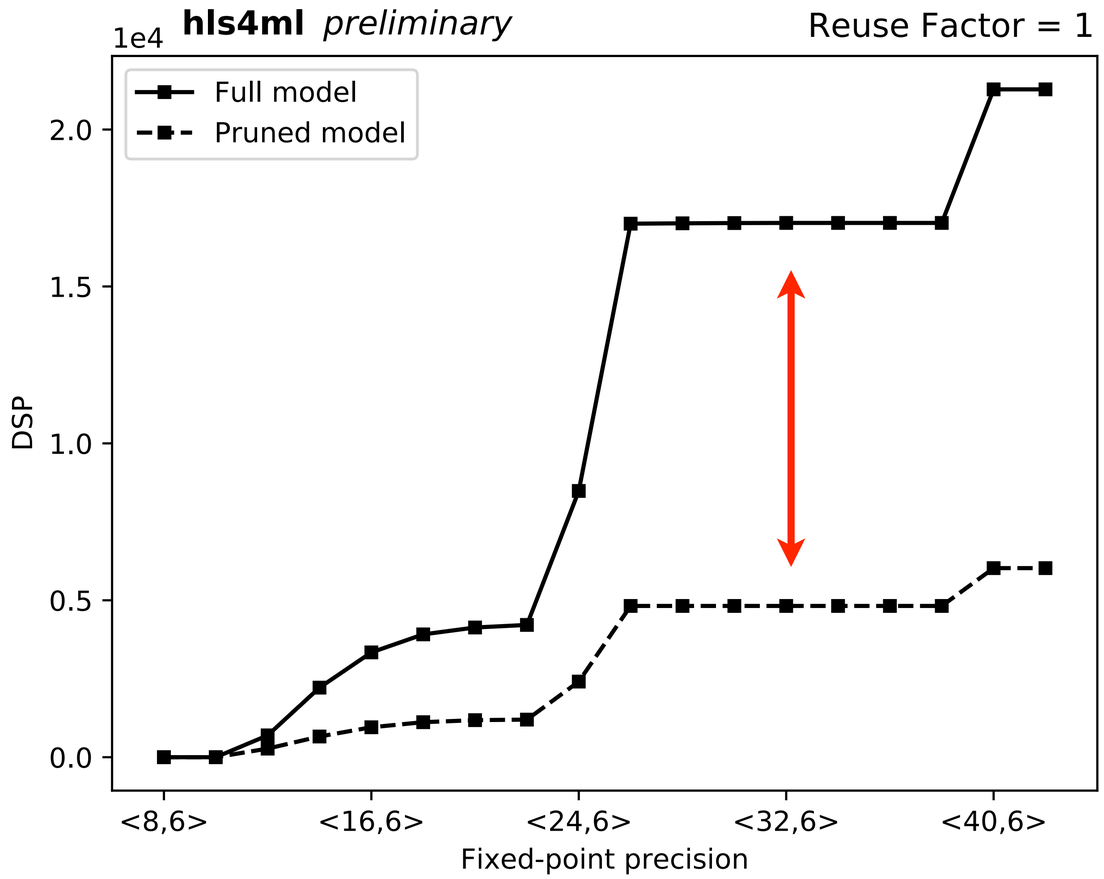
<u>Our case study: 70% weight reduction with no performance loss</u>



Distribution of weights in NN with L_1 regularization



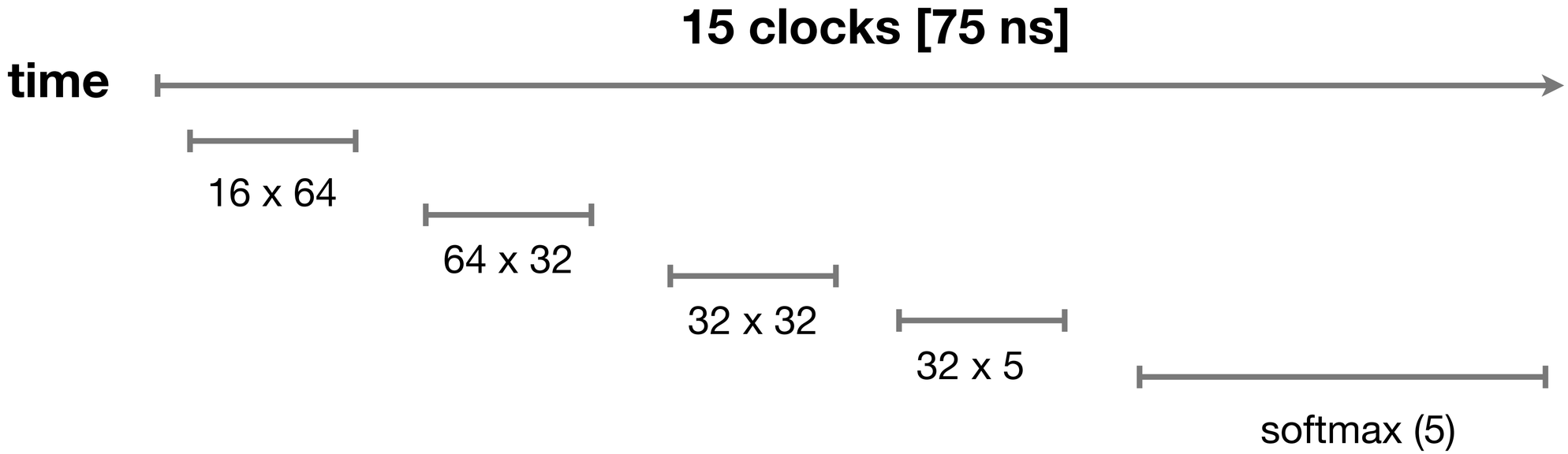
COMPRESSION



Big reduction in DSP resources

3-LAYER PRUNED MODEL - UNDER THE HOOD

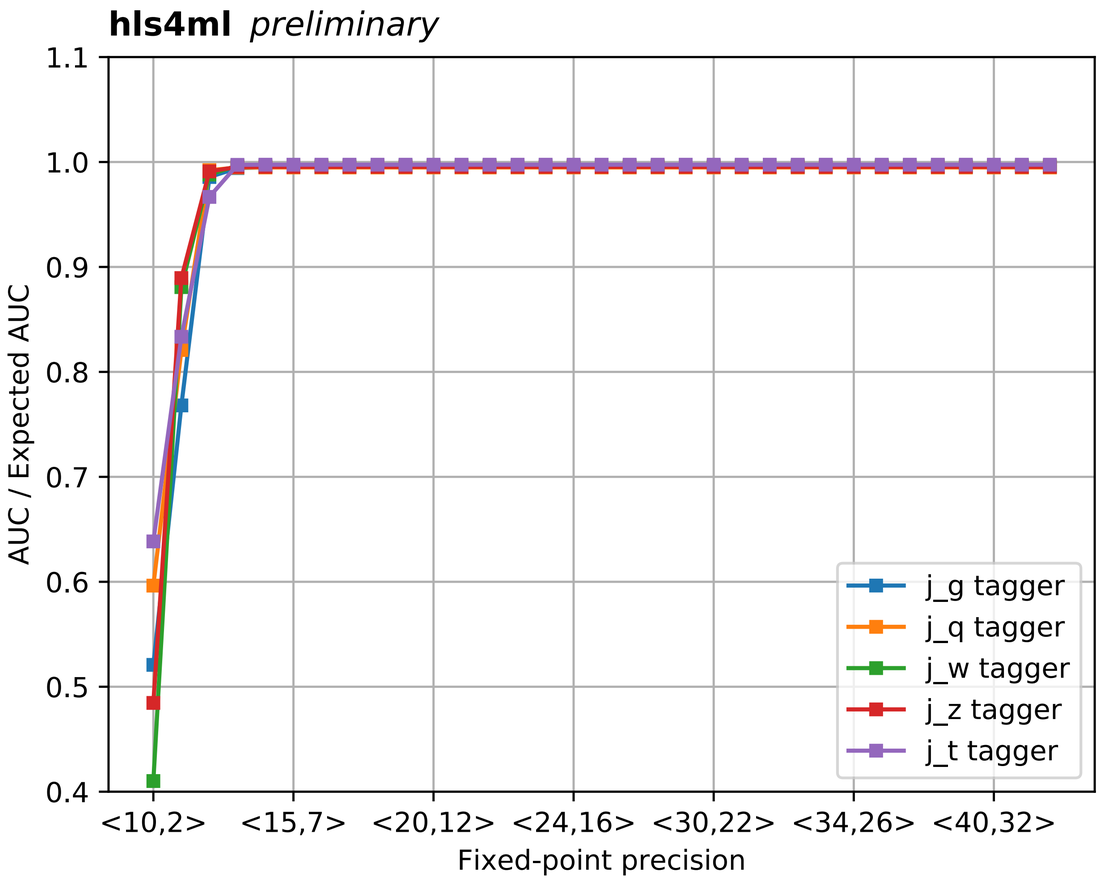
Let's take our **3-layer pruned neural network** as a benchmark and study it in more detail



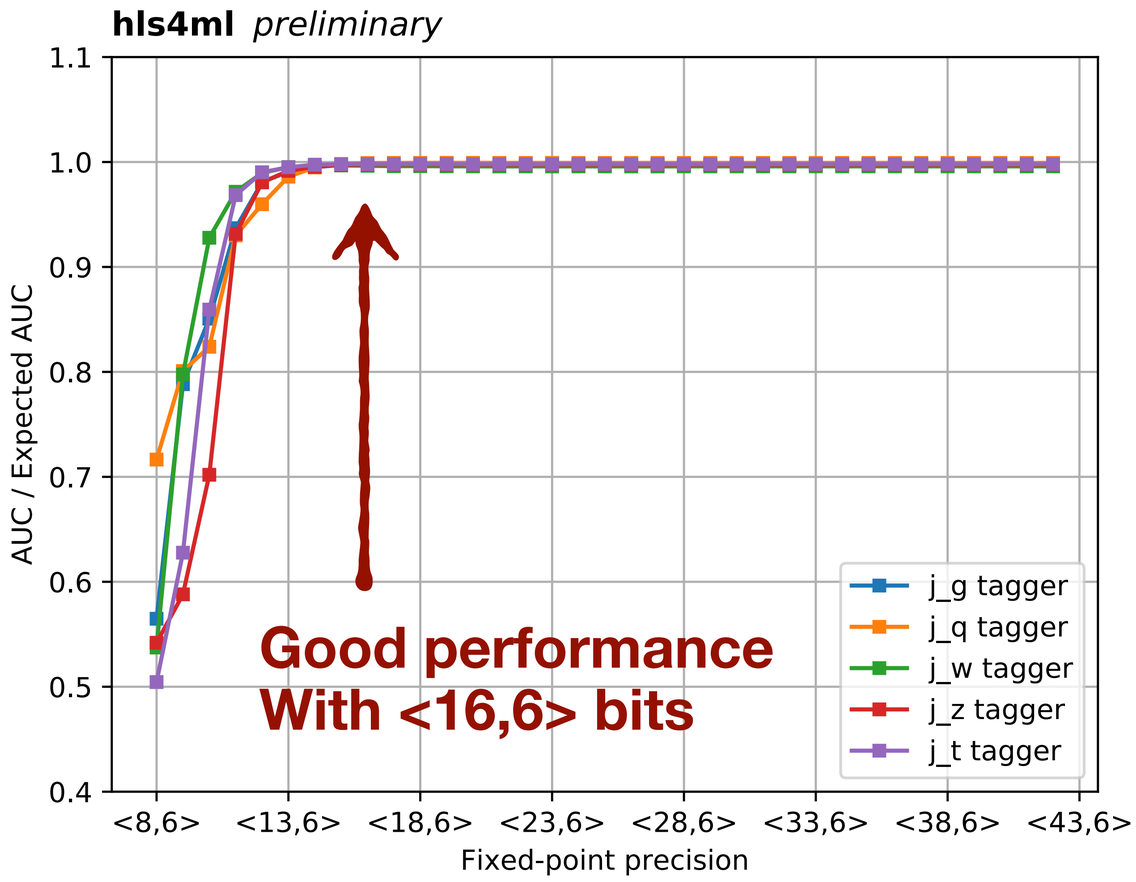
Reuse = 1	BRAM	DSP	FF	LUT
Total	13	1116	47k	35k
% Usage	~0%	20%	3%	5%

PERFORMANCE VERSUS PRECISION

Scan integer bit

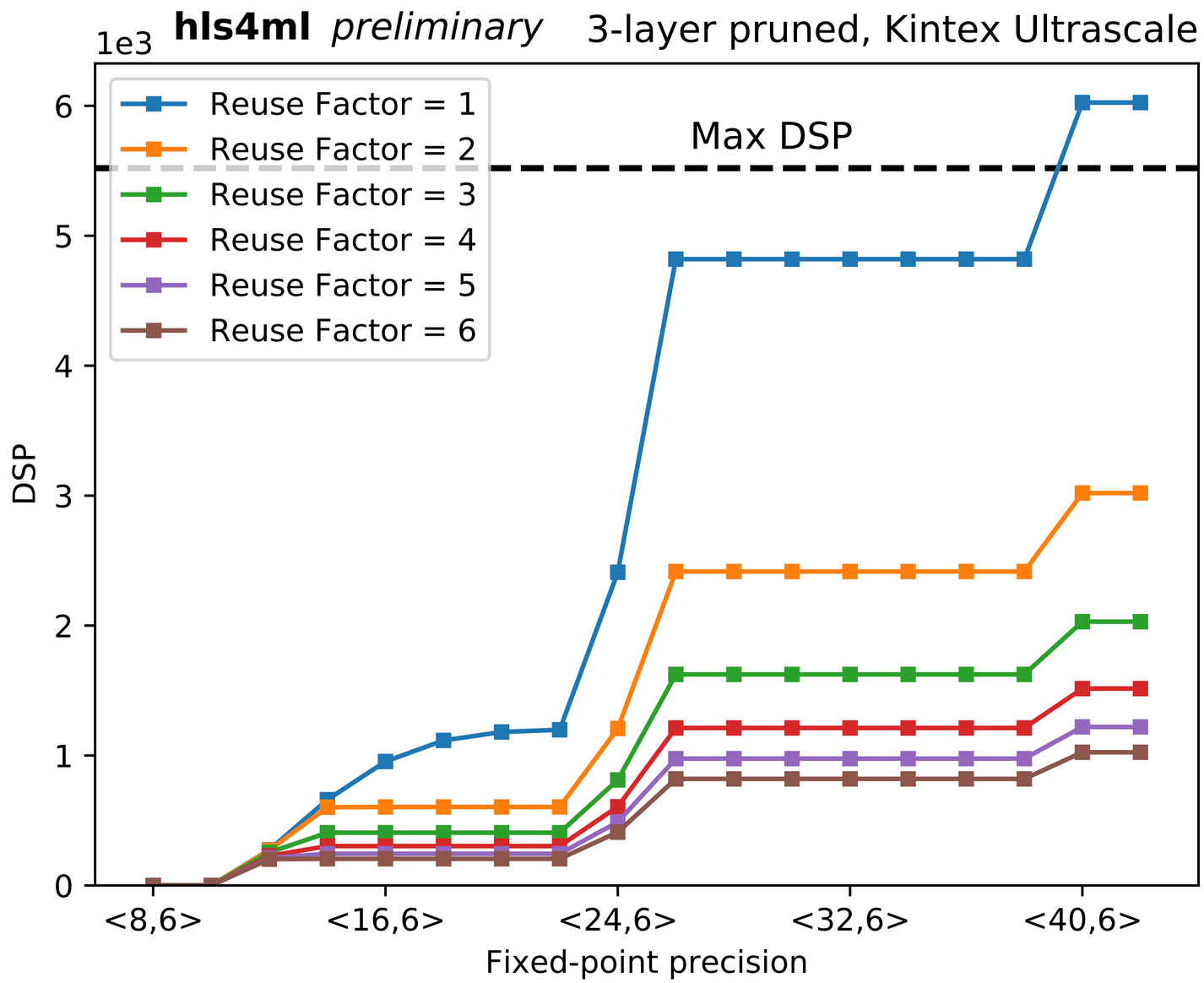


Scan decimal bit



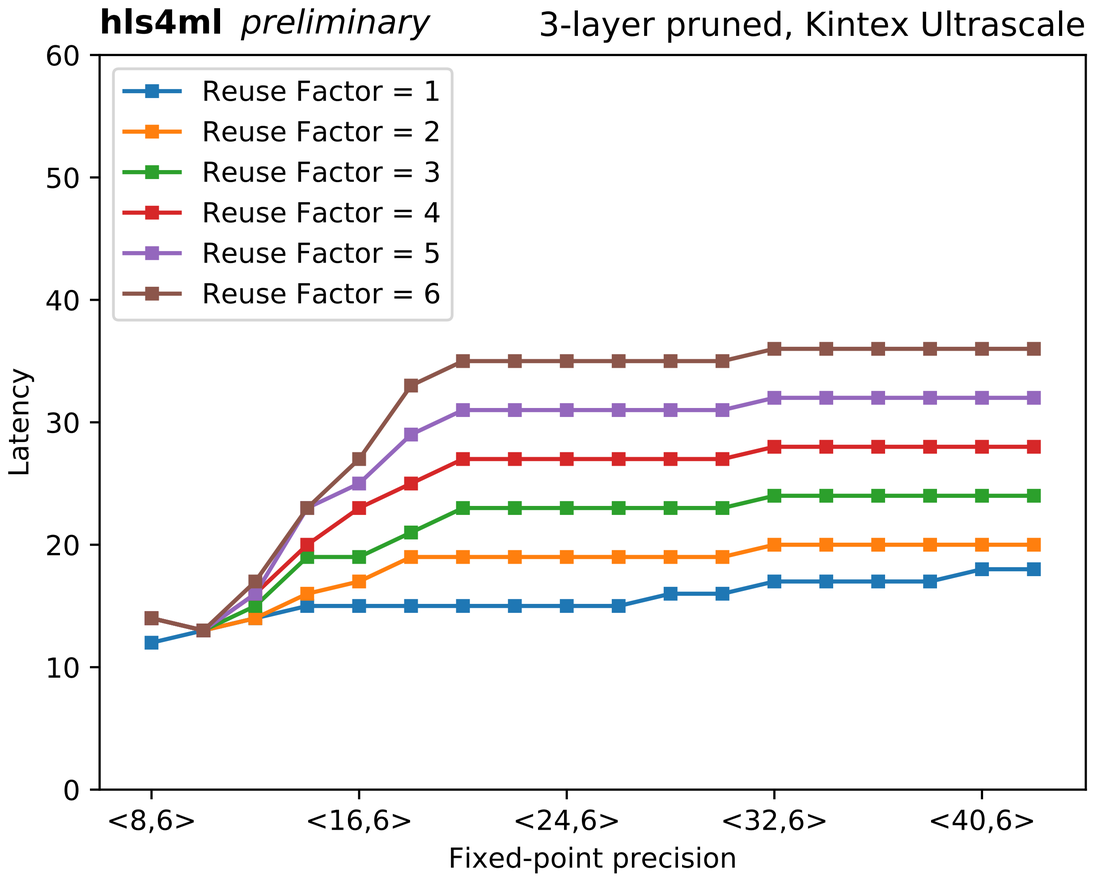
General strategy, avoid overflows in integer bit;
then scan the decimal bit until reaching optimal performance

RESOURCE USAGE

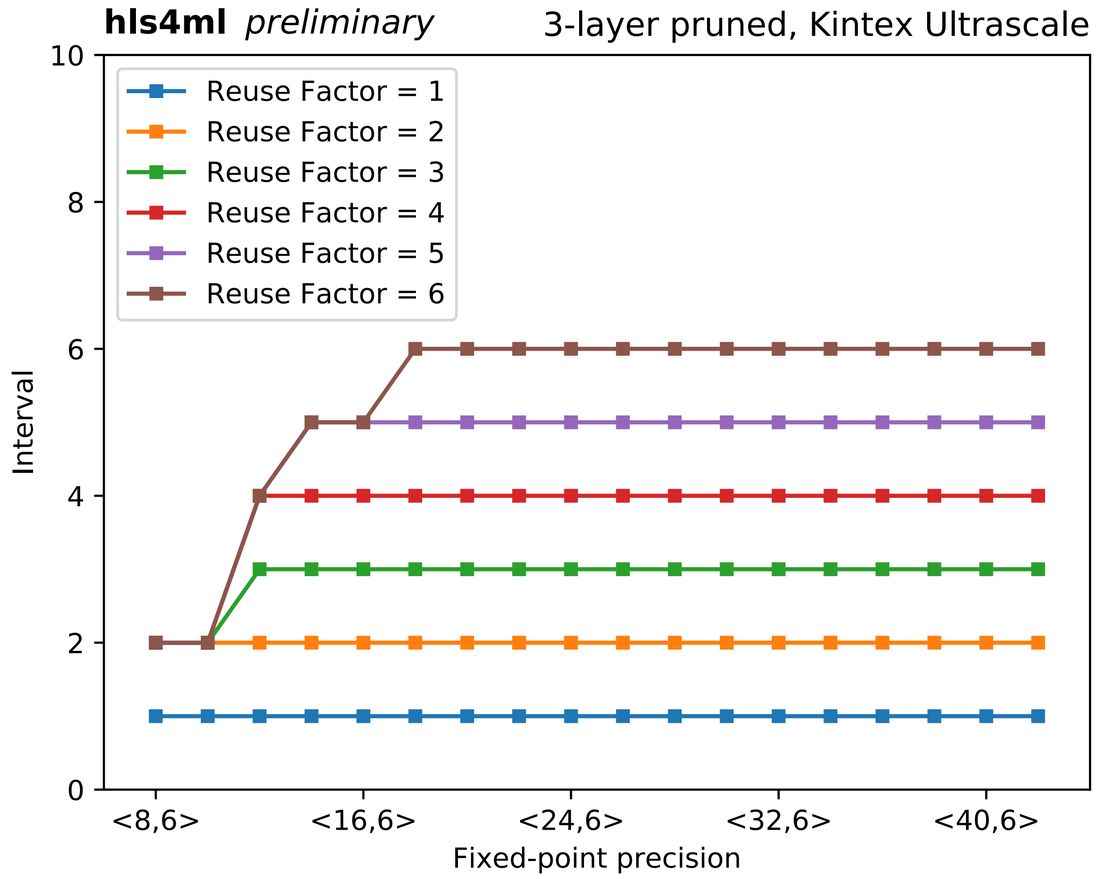


Tuning the throughput with reuse factor will reduce the DSP usage

TIMING

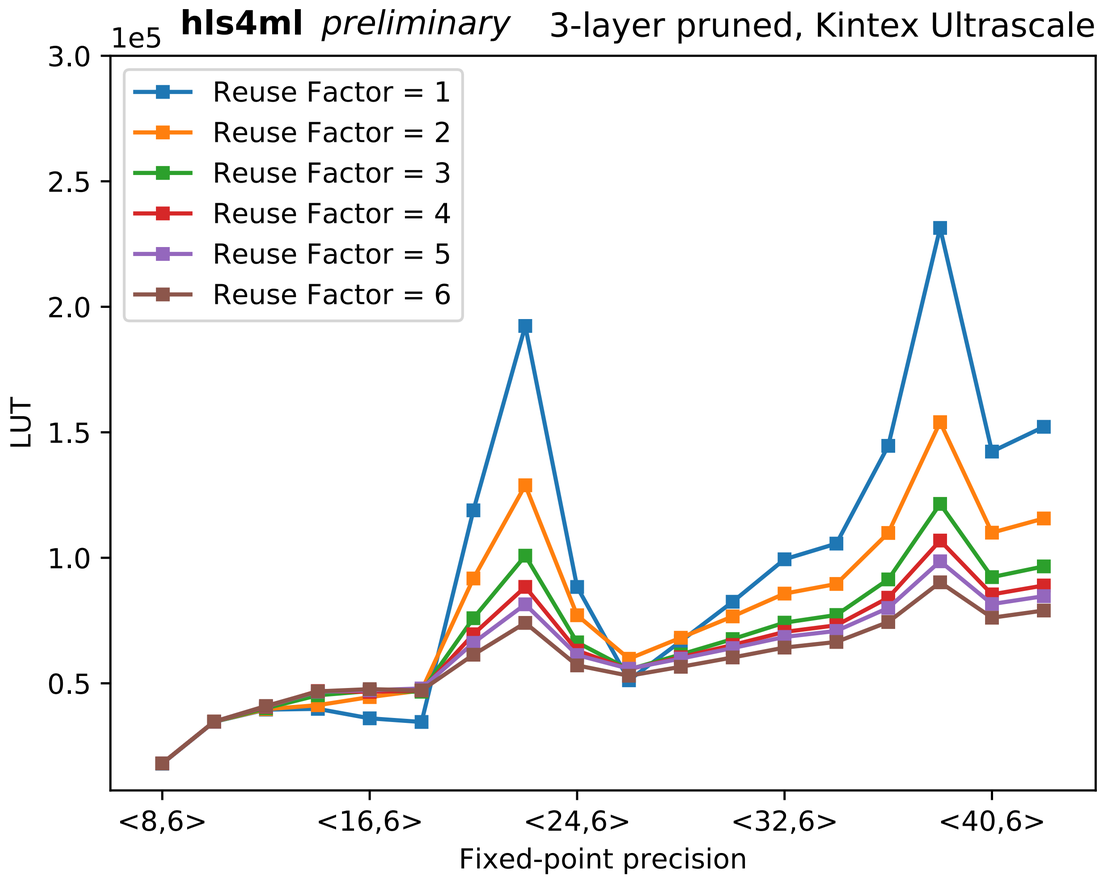


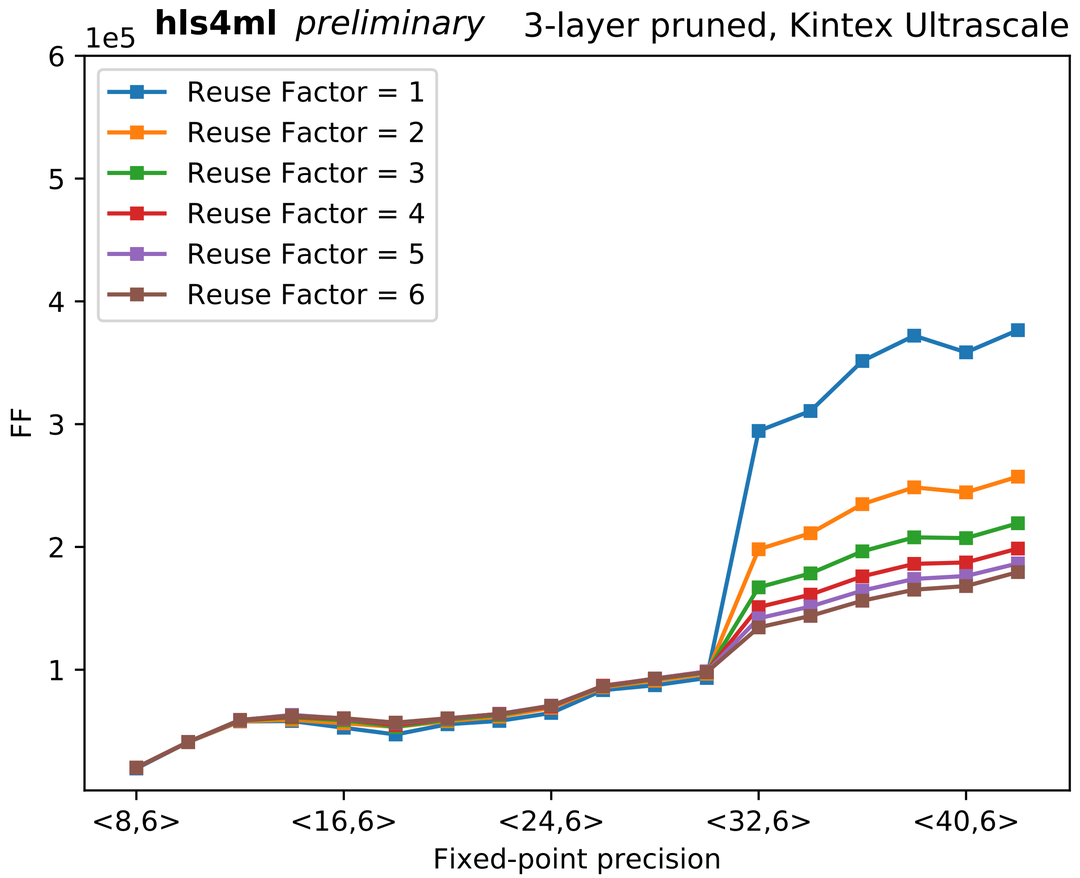
Additional latency introduced by reusing the multipliers



Behavior of pipeline interval controlled well by the reuse factor

RESOURCE USAGE





Steady increase of LUTs and FFs vs. bit precision

Spikes in LUTs at the DSP precision transitions

Will come back to these estimates later in final implementation

Study details

Xilinx Vivado 2017.2

Results are slightly different in other versions of Vivado

e.g. 2016.4 optimization is less performant for Xilinx ultrascale FPGAs

Clock frequency: 200 MHz

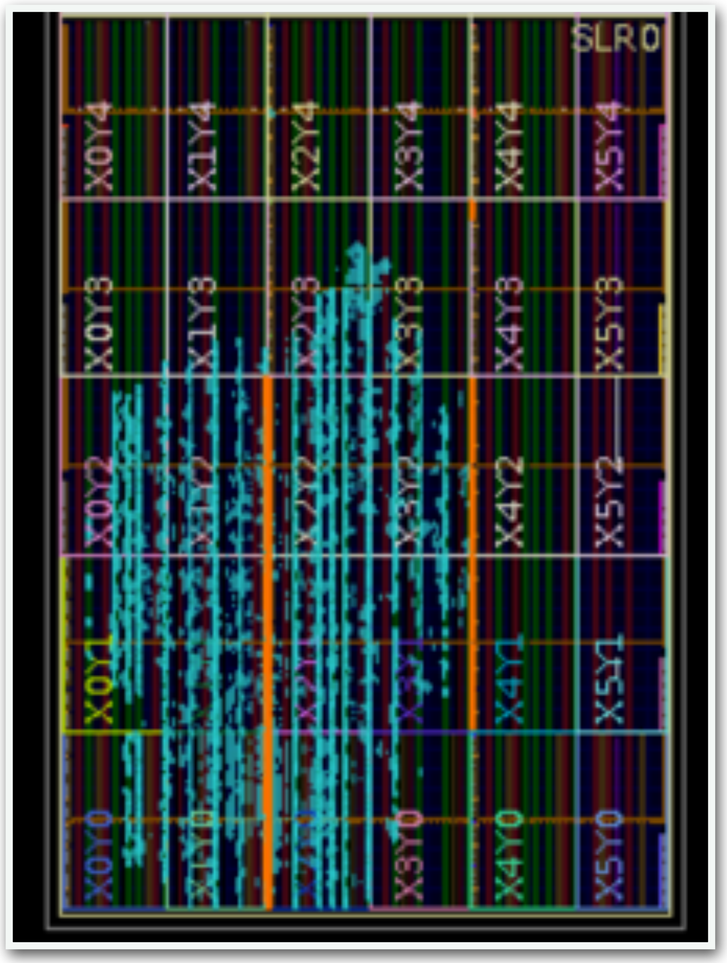
Latency results can vary (~10%) with different clock choices

FPGA: Xilinx Kintex Ultrascale (XCKU115-FLVB2104)

Results are slightly different in other FPGAs

e.g. Virtex-7 FPGAs are slightly differently optimized

Implementation Efficiency

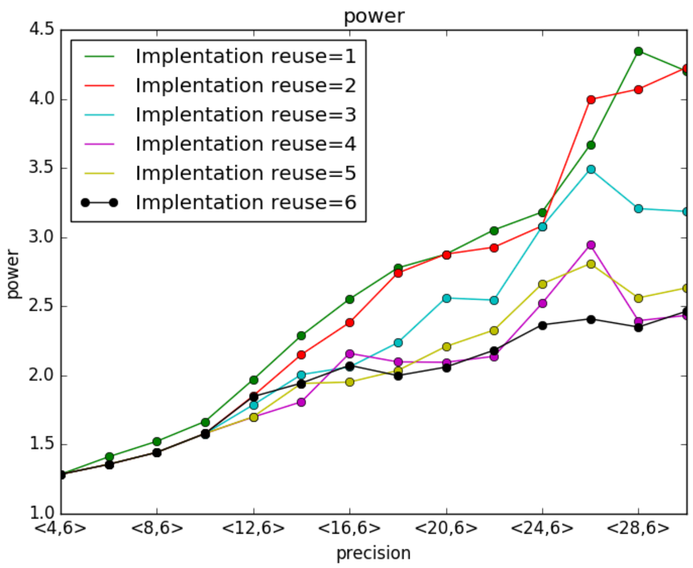


How optimal is the HLS design?

For DSPs, one can get an optimal “back-of-the-envelope” estimate

DSPs are the dominant resource usage

HLS resource estimates are great for quickly getting an estimate of resources



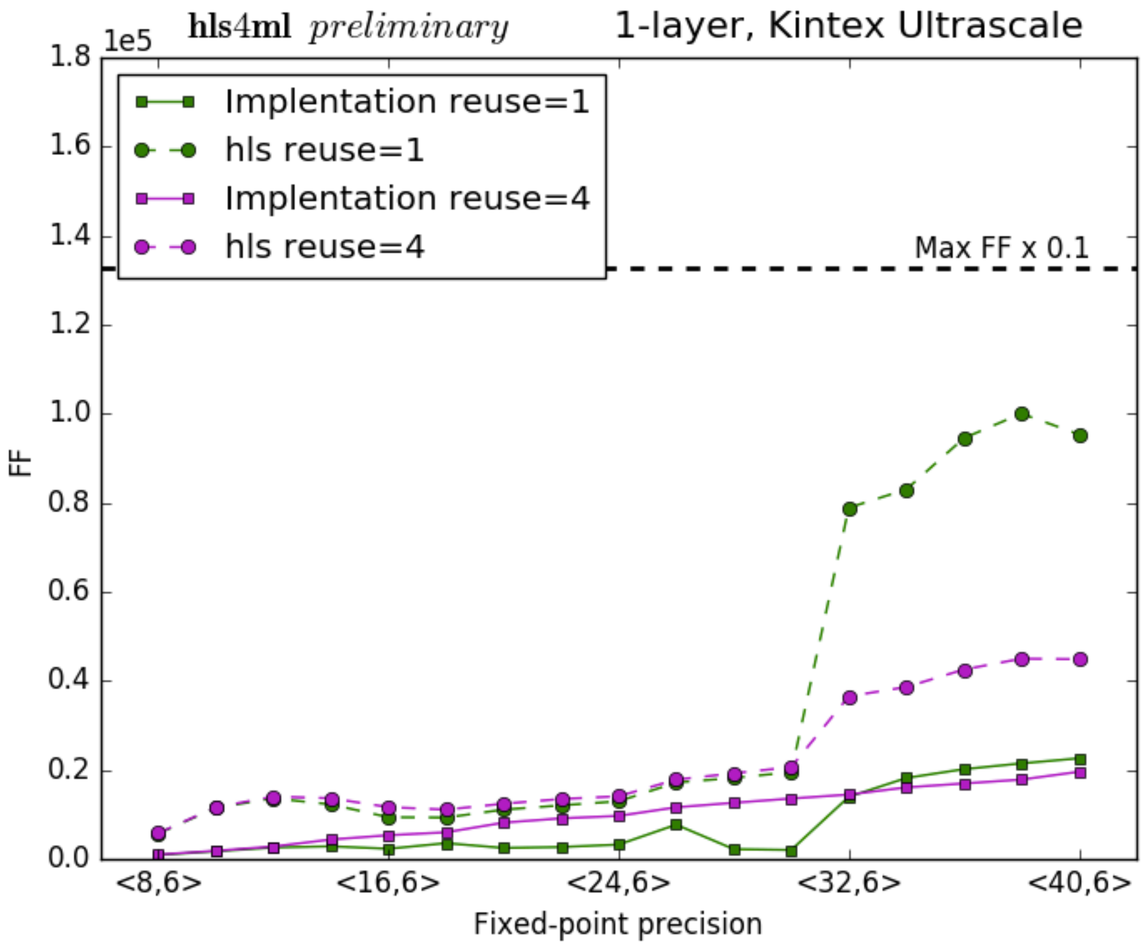
Final firmware implementation gives actual resource usage

This depends on what else is in the rest of the firmware design

We implement the network in a “barebones” design and compare

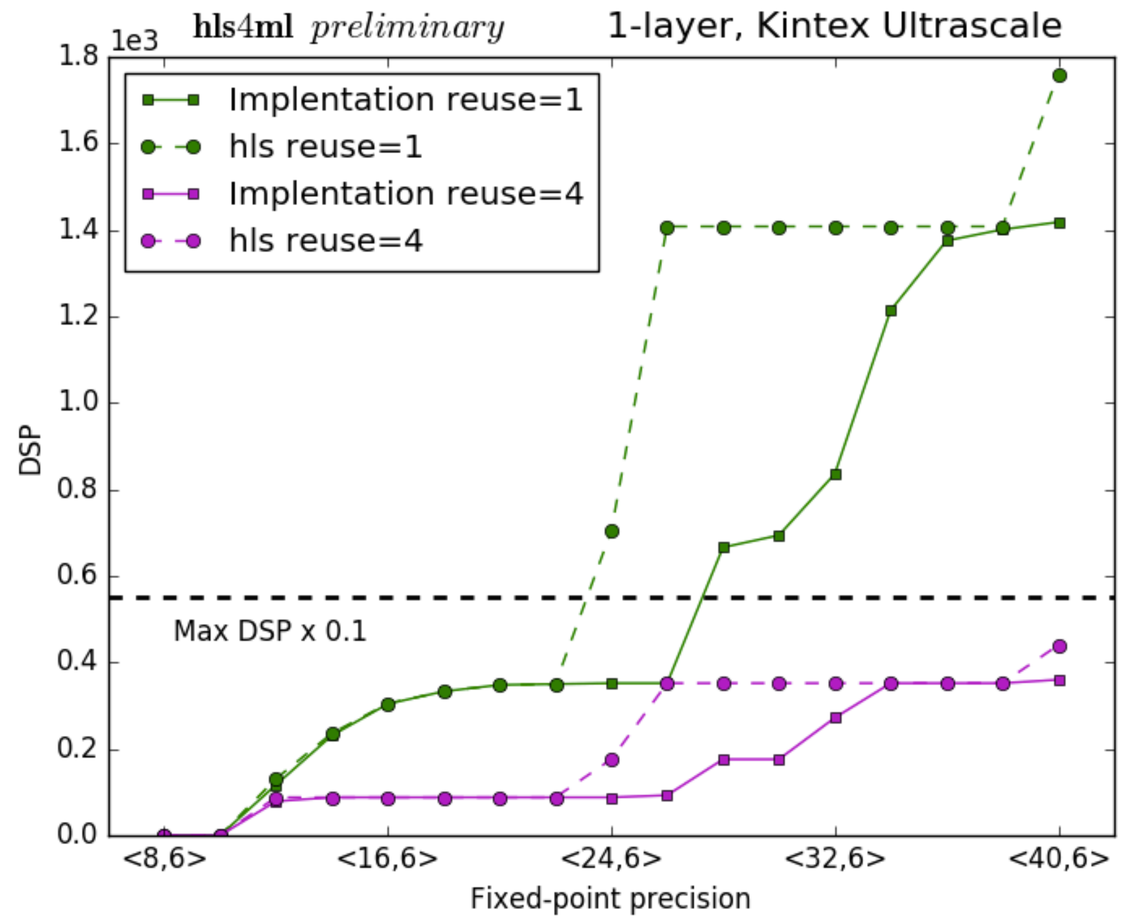
Does not always meet timing (see bonus)

HLS VS. SYNTHESIS RESOURCES



DSPs are estimated fairly well

Around DSP precision transition, Vivado does further optimizations



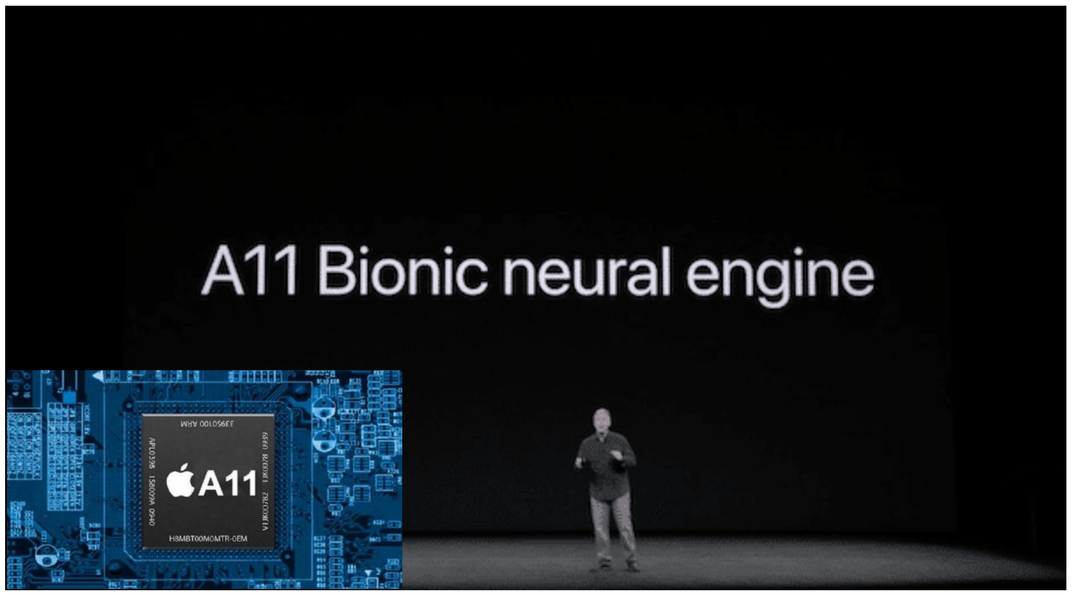
FFs are overestimated by a factor of 2-4
LUTs are overestimated by a factor of ~2

n.b. small hidden 1-layer network for testing implementation

status and outlook

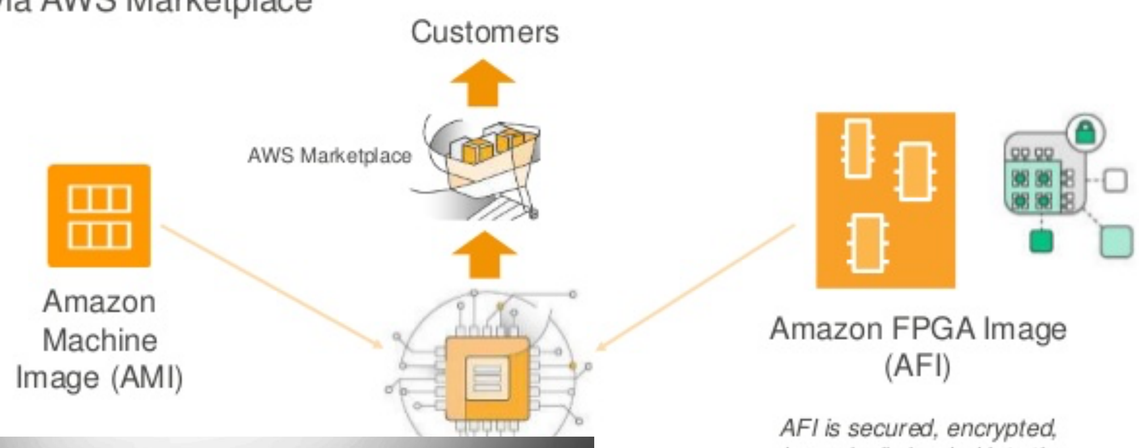
#TRENDING

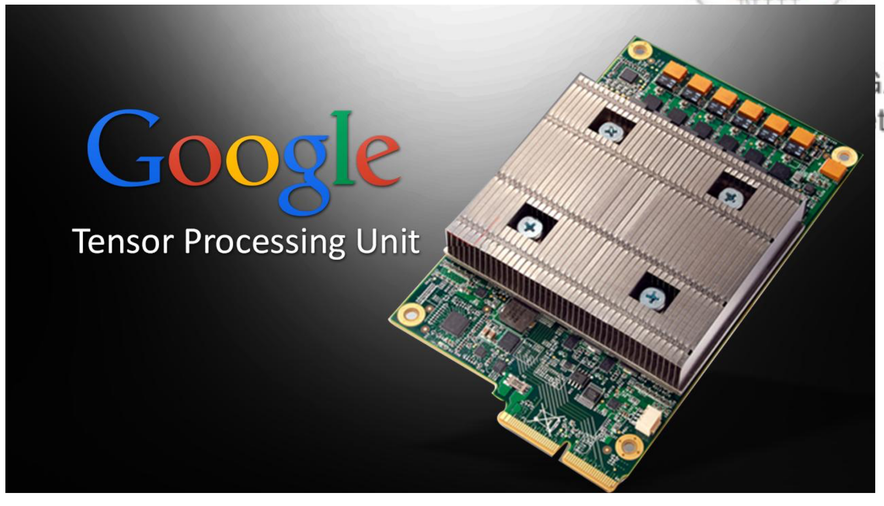
Specialized co-processor hardware for machine learning inference



Microsoft Catapult/Brainwave

Delivering FPGA Partner Solutions on AWS
via AWS Marketplace





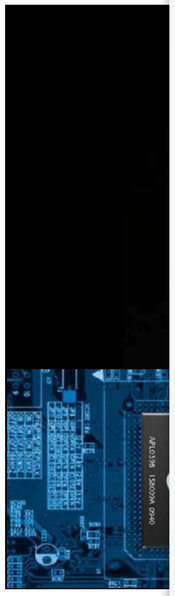
INTEL® FPGA ACCELERATION HUB

The Intel® Xeon® Acceleration Stack for FPGAs is a robust framework enabling data center applications to leverage an FPGA's potential to increase

#TRENDING

Specialized co-processor hardware for machine learning inference

Microsoft Catapult/Brainwave



The same NN libraries can be adapted for other applications
From µs to ms timescales and small to large networks

hls4ml

INTEL® FPGA ACCELERATION HUB

The Intel® Xeon® Acceleration Stack for FPGAs is a robust framework enabling data center applications to leverage an FPGA's potential to increase

Status

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

A package for machine learning inference in FPGAs. We create firmware implementations of machine learning algorithms using high level synthesis language (HLS). We translate traditional open-source machine learning package models into HLS that can be configured for your use-case!

The project is currently in development, so please let us know if you are interested, your experiences with the package, and if you would like new features to be added.

contact: hls4ml.help@gmail.com

Project status

in beta version (March 17, 2018), **v0.1.2**

BETA VERSION IS LIVE!

Watch for arXiv preprint in a few weeks

Outlook

More network architectures

- Library for **convolutional layers** are implemented, ***in testing***
  - Similar to deep neural networks, more parameters
- Prototype libraries exist for **recurrent layers and LSTM**
  - Similar to DNN for in multiplication, different data flow paradigm

More ML codes

- Support currently for **Keras** and **TensorFlow**
- **PyTorch** has a first implementation, in testing
- Others?

More FPGAs

- Support planned for **Altera Quartus HLS**

SUMMARY

We introduce a software/firmware package, `hls4ml`

Automated translation of neural networks into firmware using HLS

Case study present with jet substructure in L1 trigger

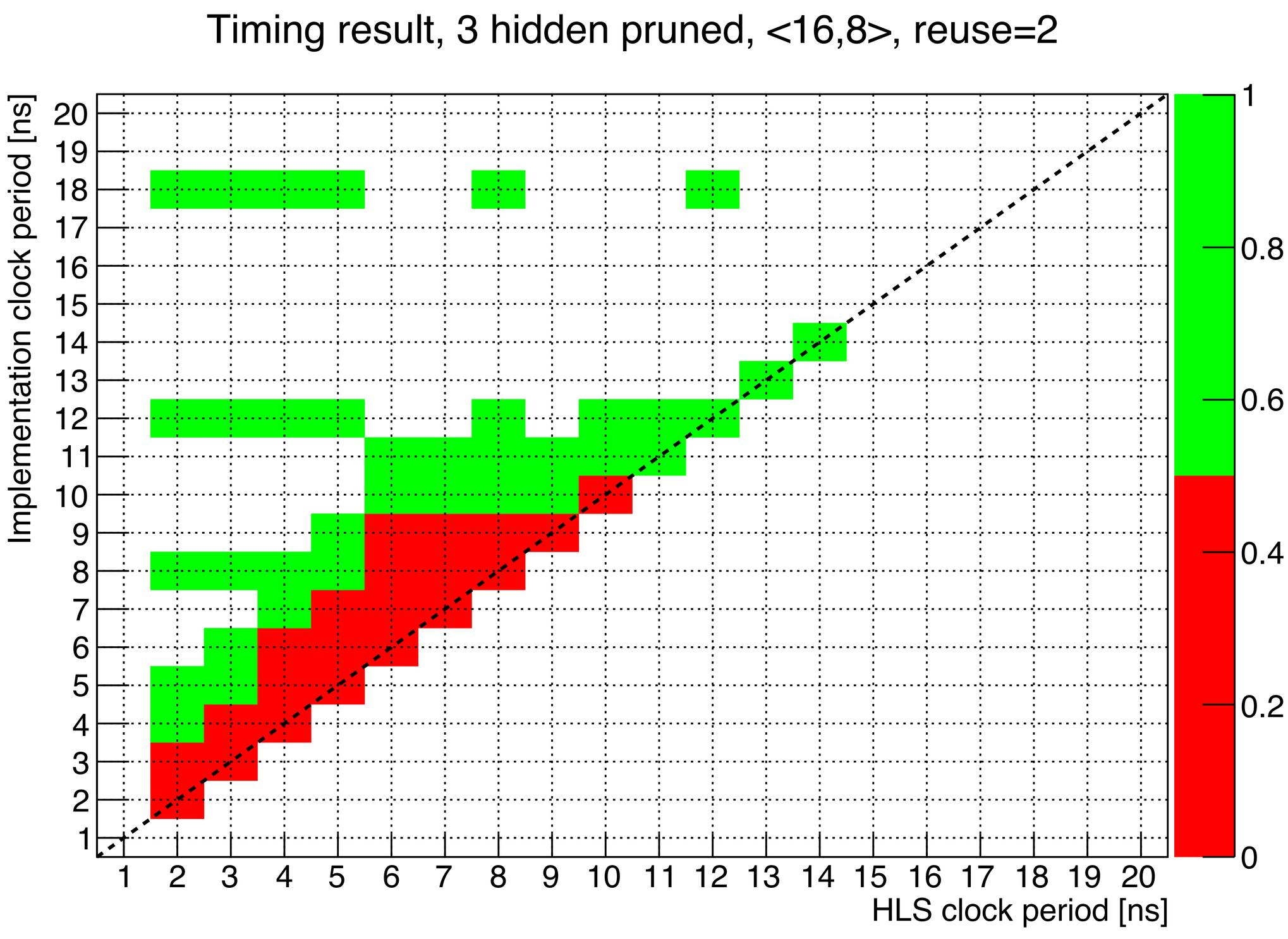
Tunable configuration for a broad range use cases

More info here:

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

Bonus

MEETING TIMING



LUTs in implementation