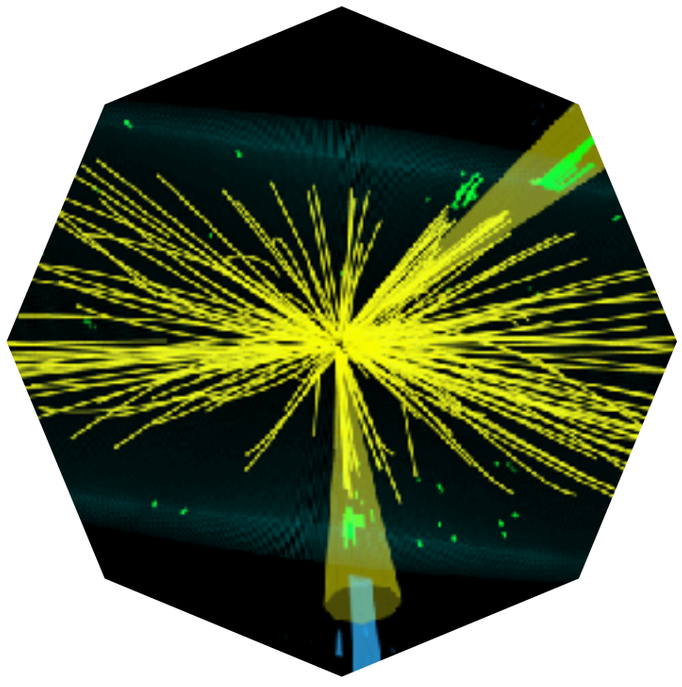
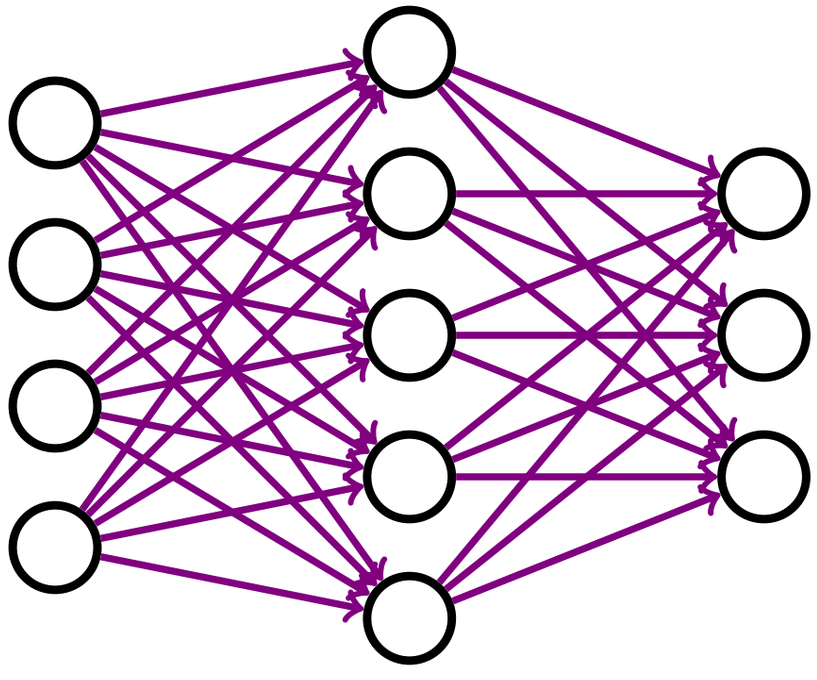
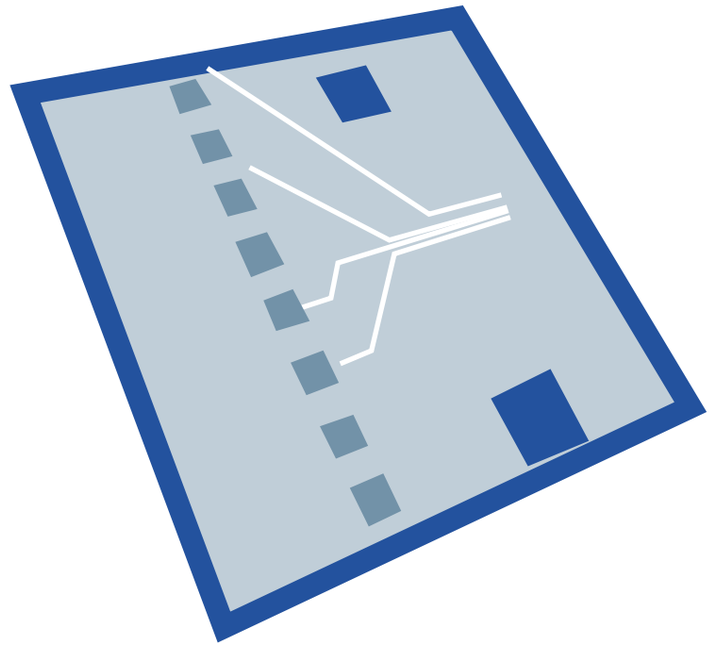
Fast Inference of Deep Neural Networks in FPGAs for Particle Physics (arXiv:1804.06913)







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

Research Techniques Seminar
Fermilab
4/24/2018

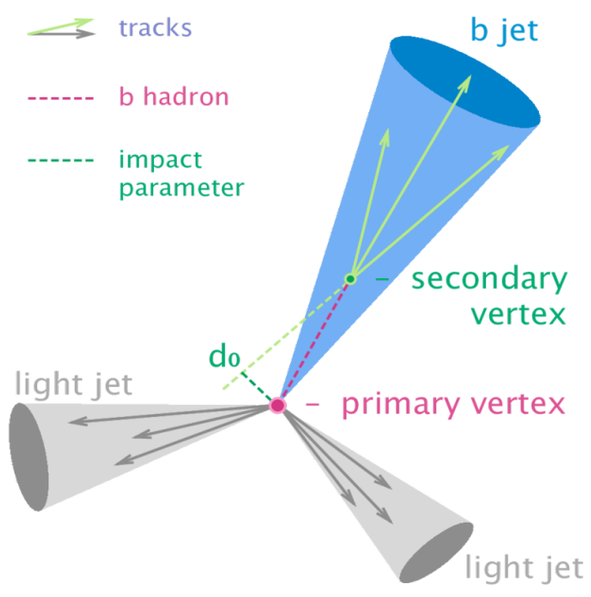
Outline

- Introduction and Motivation
  - Machine Learning in HEP
  - FPGAs and High-Level Synthesis (HLS)
  - Industry Trends
  - HEP Latency Landscape
    - `hls4ml`: HLS for Machine Learning
- Case Study and Design Exploration
- Summary and Outlook
  - Cloud-scale Acceleration

Introduction

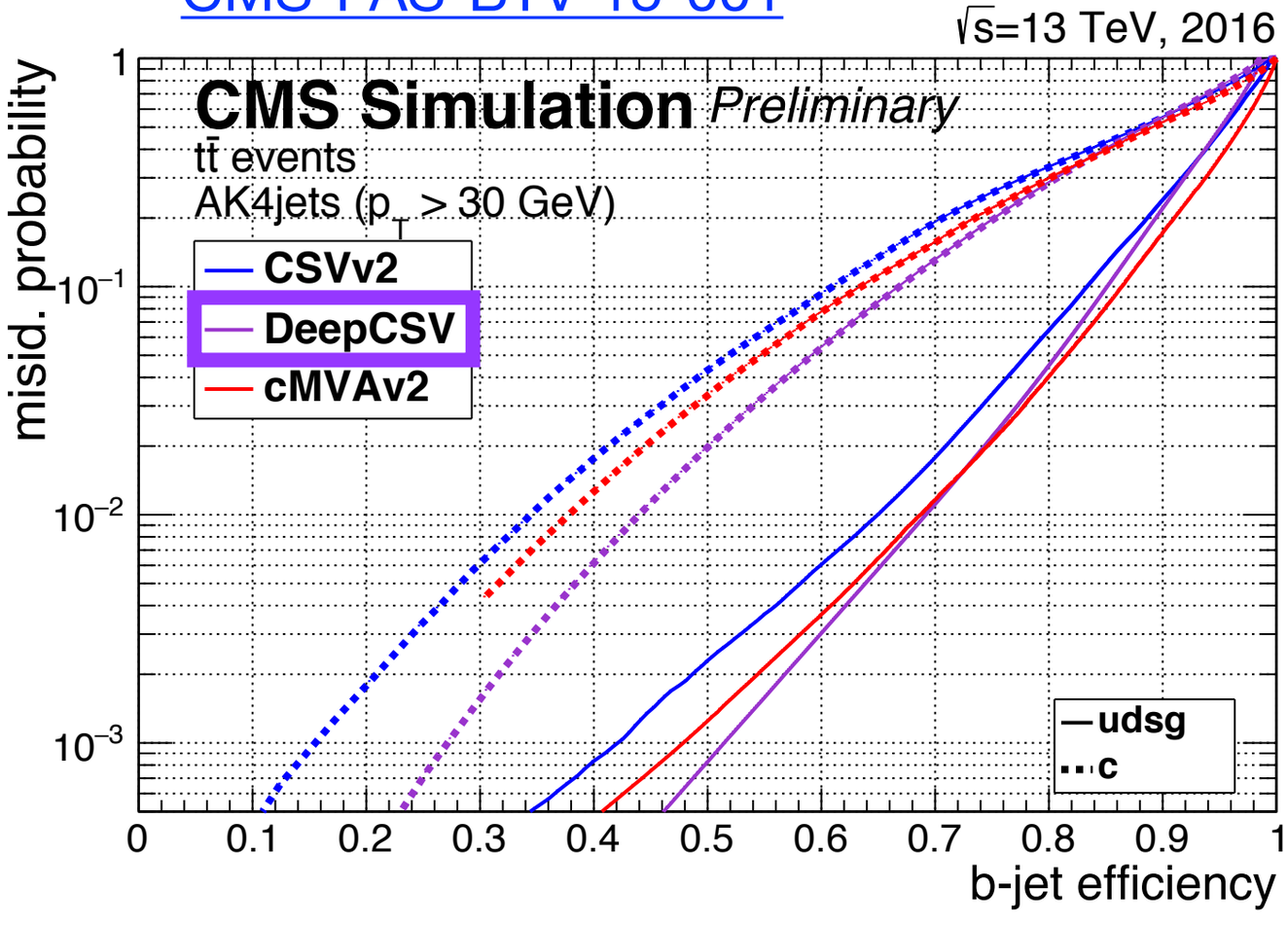
Machine Learning in HEP

- Learning optimized nonlinear functions of many inputs for performing difficult tasks from (real or simulated) data
- Many successes in HEP: **identification of b-quark jets**, Higgs candidates, particle energy regression, **analysis selection**, …



- **Neural network** based on high-level features

CMS-PAS-BTV-15-001

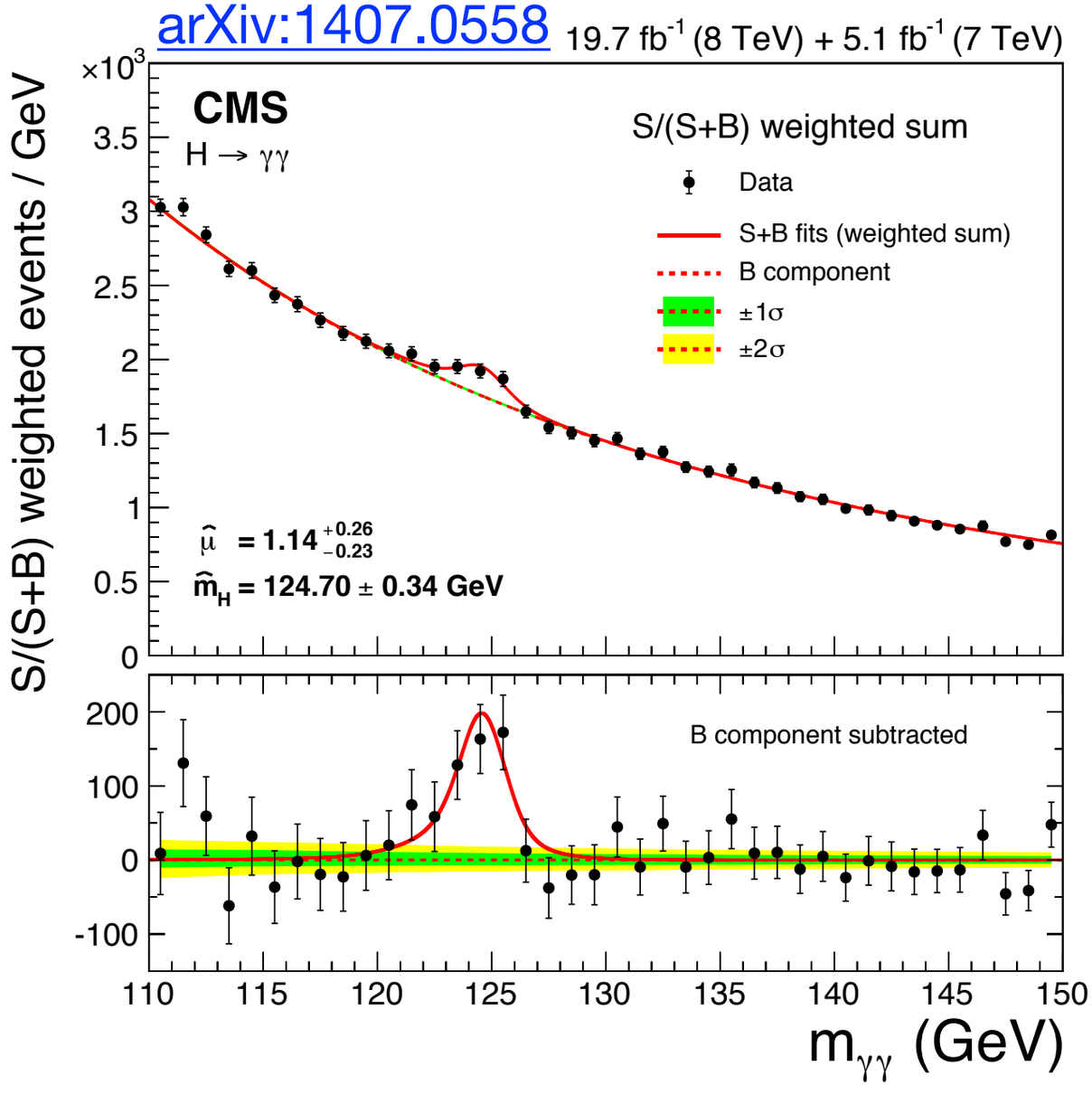


Machine Learning in HEP

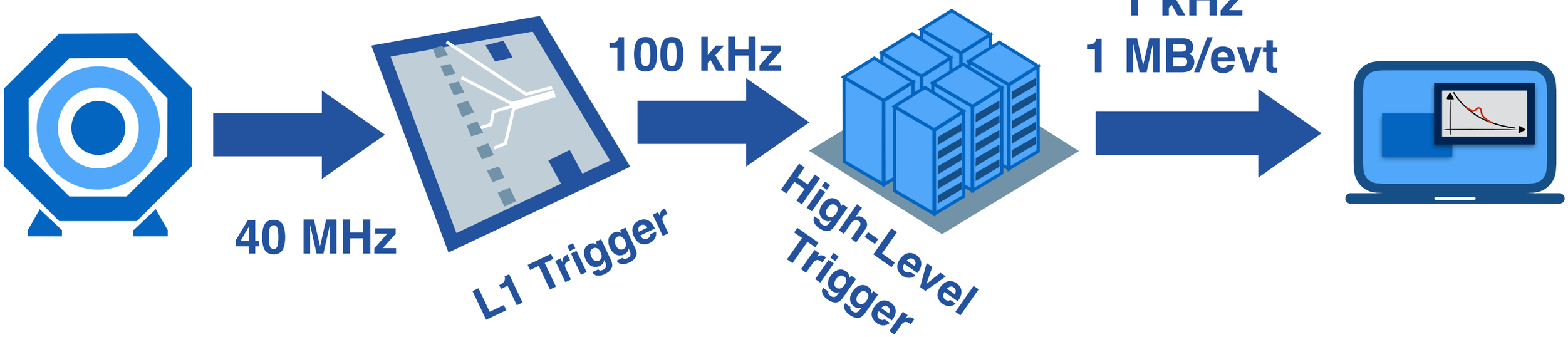
- Learning optimized nonlinear functions of many inputs for performing difficult tasks from (real or simulated) data
- Many successes in HEP: **identification of b-quark jets**, Higgs candidates, particle energy regression, **analysis selection**, …

ML algorithms used in every aspect of Higgs discovery:
energy regression
S/B discrimination,
…

Typically applied offline, not online (trigger-level)



CMS Trigger



- Level-1 Trigger (hardware)
- 99.75% rejected
- decision in ~4 μs

- High-Level Trigger (software)
- 99% rejected
- decision in ~100s ms

- After trigger, 99.99975% of events are gone forever

HEP Latency Landscape

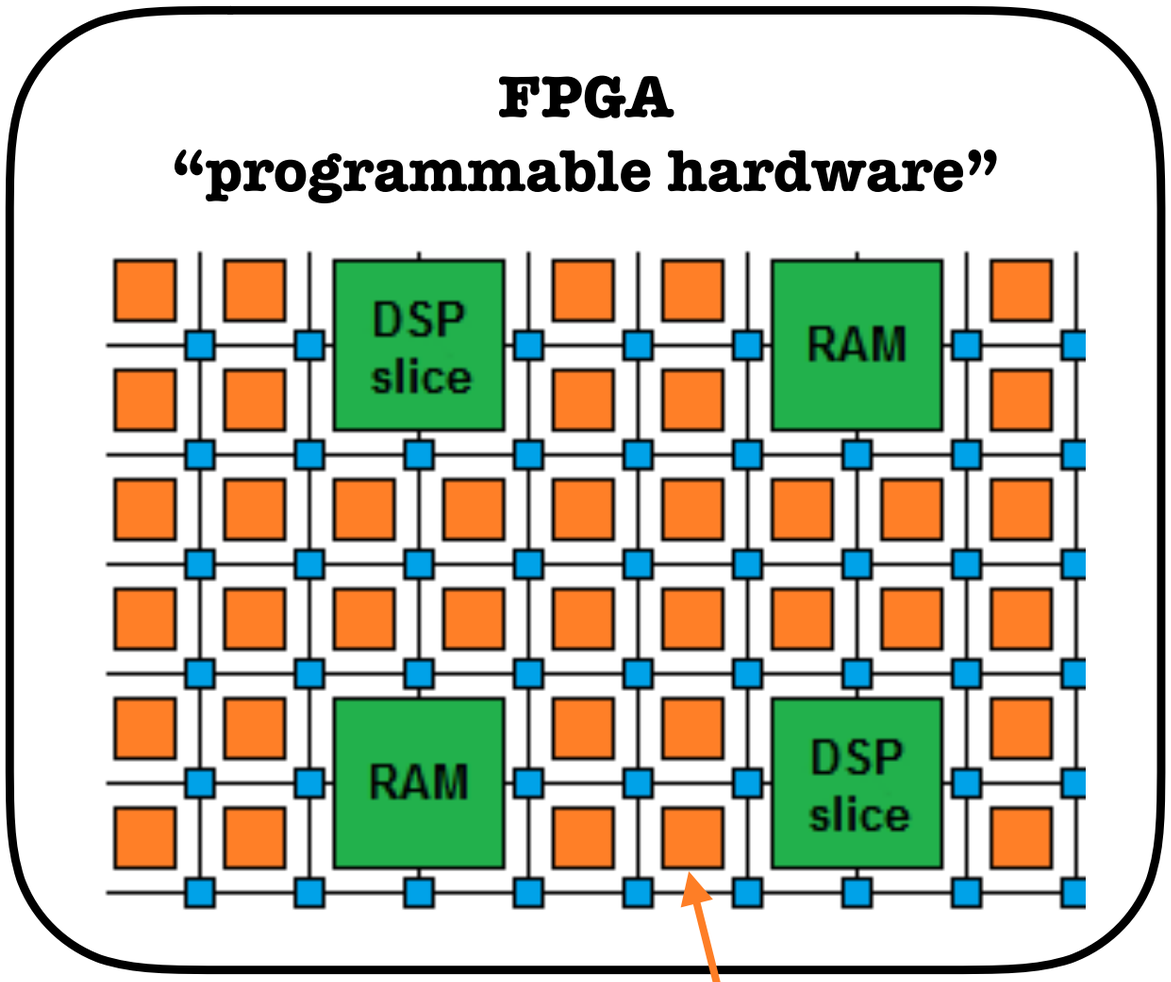
CMS: L1 Trigger HLT Offline

1 ns 1 us 1 ms 1 s

FPGAs CPUs

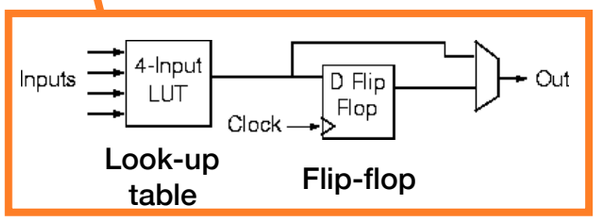
Field-Programmable Gate Array

- **Flexible:** re-programmable interconnects between **configurable logic blocks** and **embedded components**
  - LUTs (logic), Flip-Flops (registers), DSPs (arithmetic), Block RAMs (memory)
- **High Throughput:** O(100) optical transceivers running at O(15) Gbs
- **Massively parallel**
- **Low power** (relative to CPU/GPU)



Typical modern FPGA (Kintex UltraScale):
1.3 M FFs
700k LUTs
5500 DSPs
2200 BRAMs

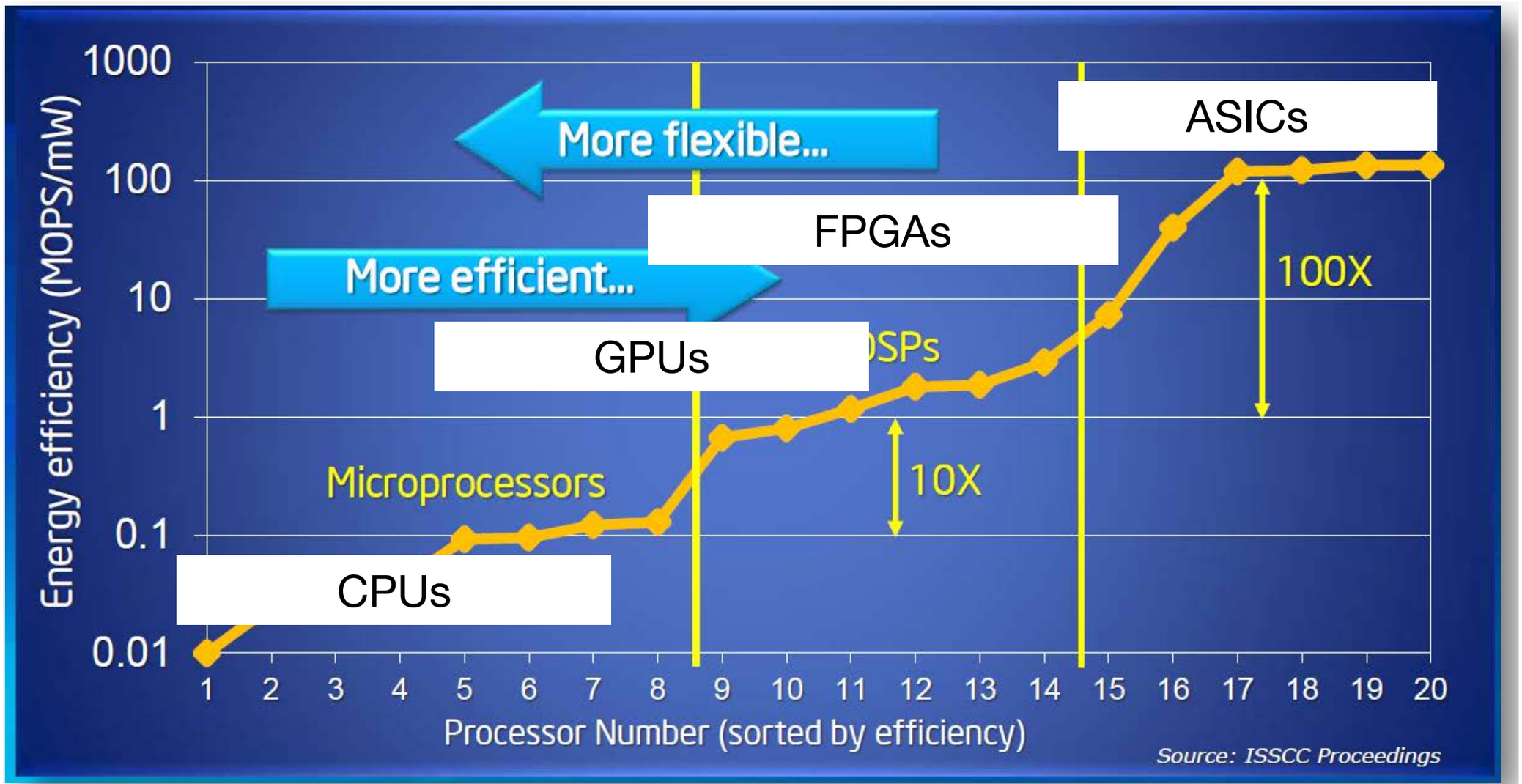
Logic cell



CPUs, GPUs, FPGAs, and ASICs

How are they different?

- FPGAs are the middle ground of latency, energy efficiency, and flexibility



Source: Bob Broderson, Berkeley Wireless group

High-Level Synthesis

- **FPGA** development is becoming more accessible, with tools like **High-Level Synthesis: untimed C code** with additional **directives** that is synthesized into **RTL Verilog/VHDL**

```
module dut(rst, clk, q);
  input rst;
  input clk;
  output q;
  reg [7:0] c;

  always @ (posedge clk)
  begin
    if (rst == 1b'1) begin
      c <= 8'b00000000;
  end
  else begin
    c <= c + 1;
  end

  assign q = c;
endmodule
```

RTL Verilog

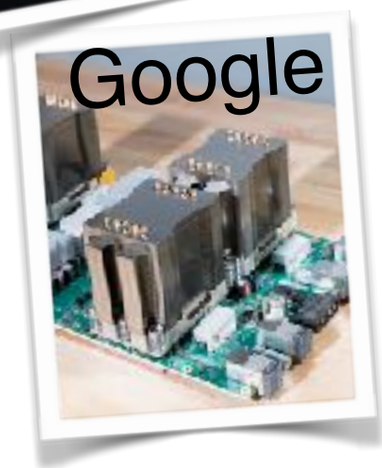
vs.

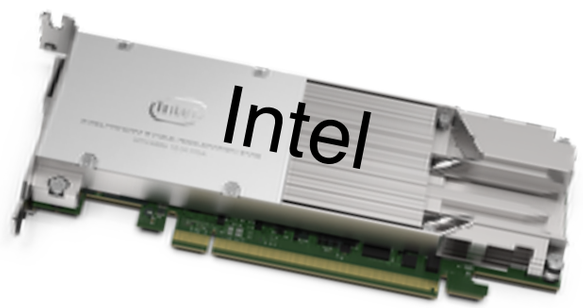
```
uint8 dut() {
  static uint8 c;
  c+=1;
}
```

Untimed C code

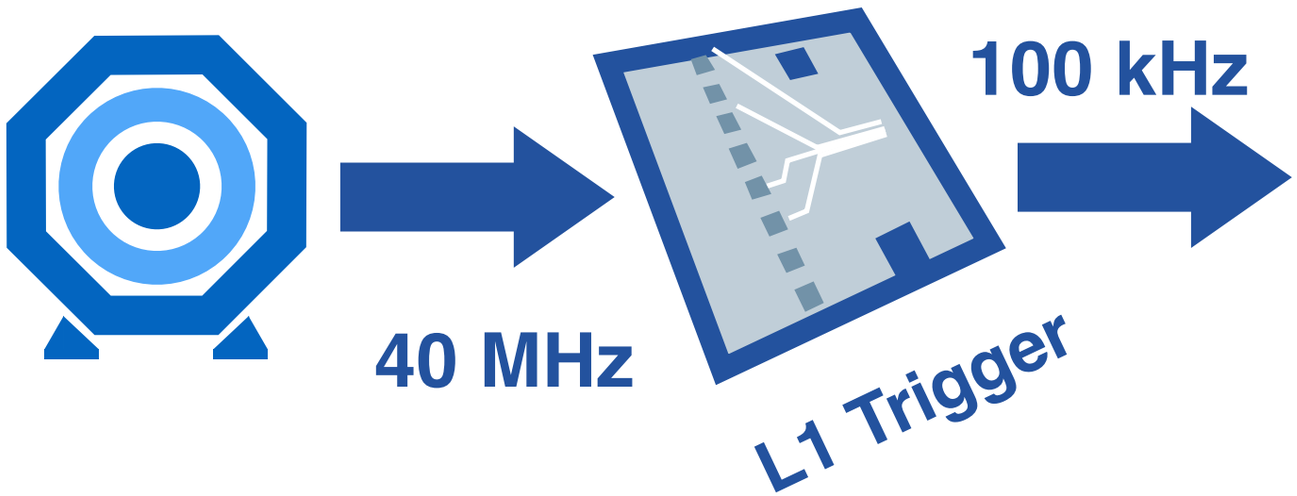
Industry Trends

- Industry is moving toward custom hardware and **FPGAs** to quickly apply ML algorithms
- Already being used as backend accelerators for **Bing** web searches, **Siri** queries, and more…





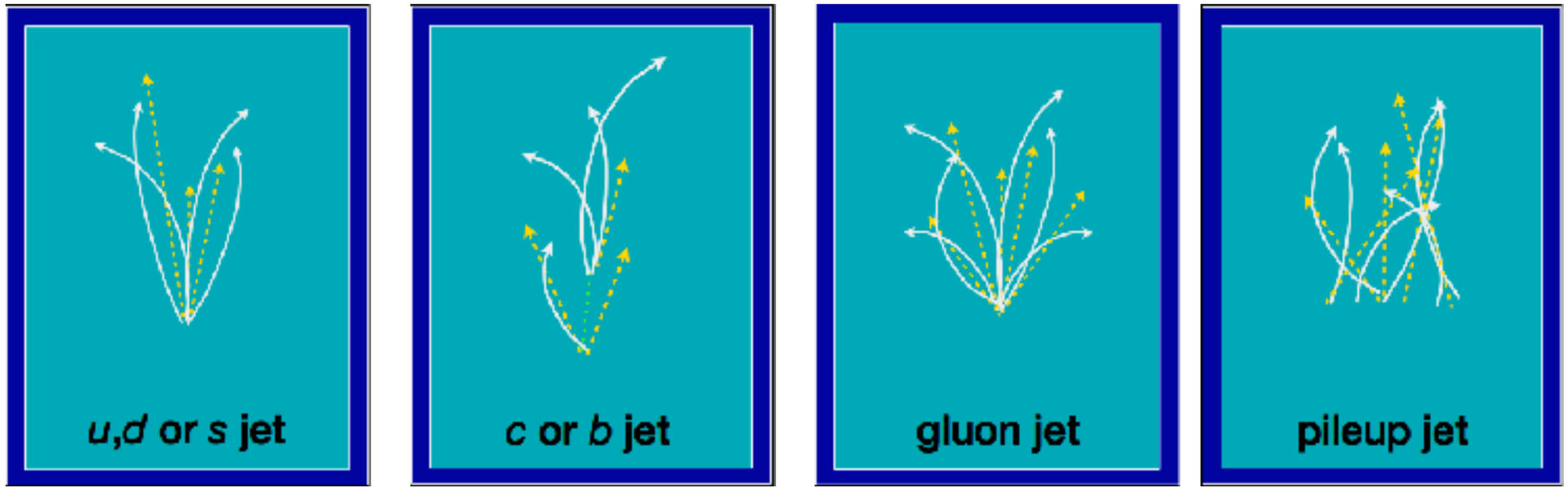
hls4ml



- hls4ml: neural network translation library for HLS
  - Support for common ML workflows and architectures
  - Tunable configuration for different use cases
- Focus on **L1 trigger** as **1st application**
  - What can we do in < μs on one FPGA?

Case Study and Design Exploration

Case Study: Jet Substructure



- Illustrative example: not necessarily the most realistic for L1 *today*, but lessons are generic

Jet Substructure Inputs

Observables

m_{mMDT}

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

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

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

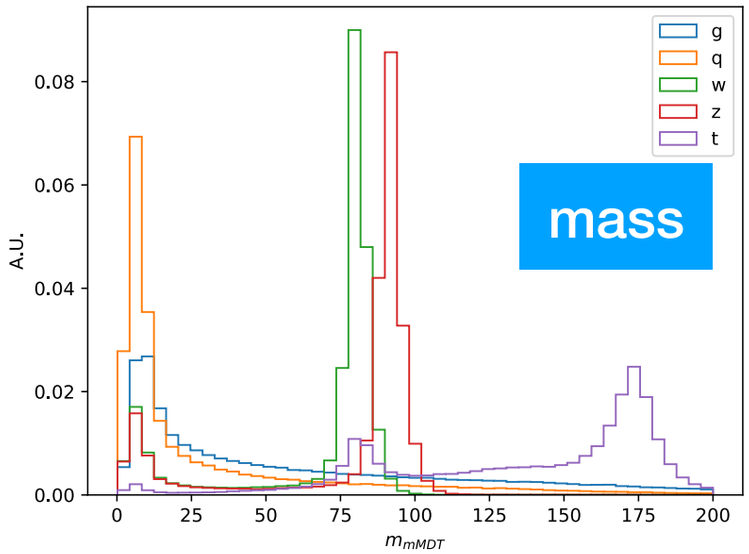
$C_2^{\beta=1,2}$

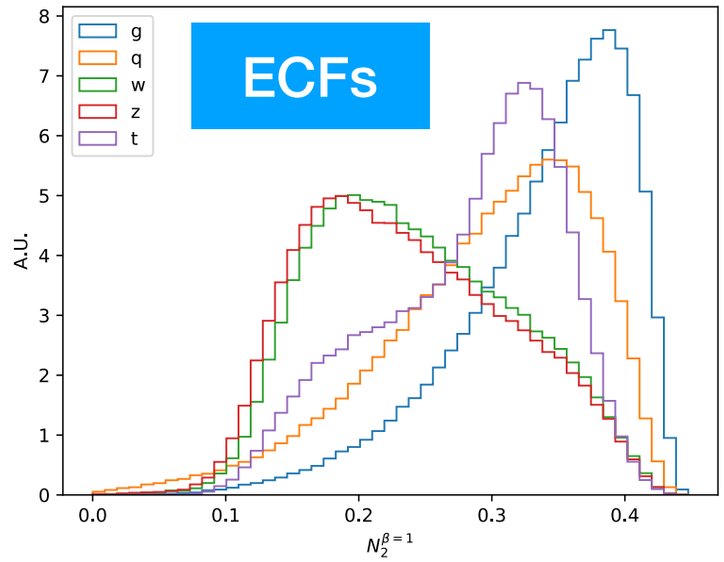
$D_2^{\beta=1,2}$

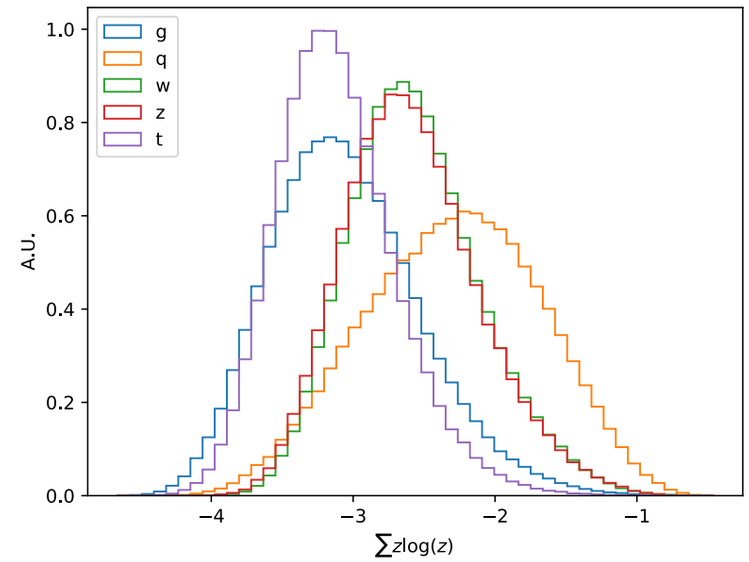
$D_2^{(\alpha,\beta)=(1,1),(1,2)}$

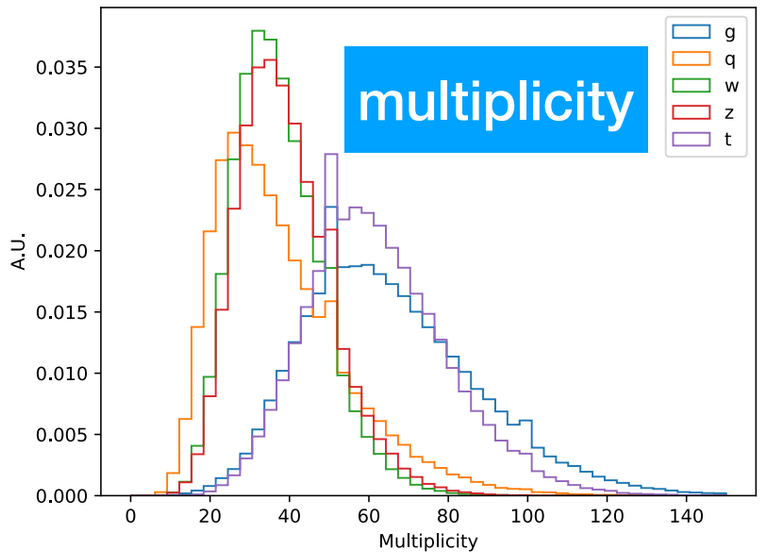
$\sum z \log z$

Multiplicity







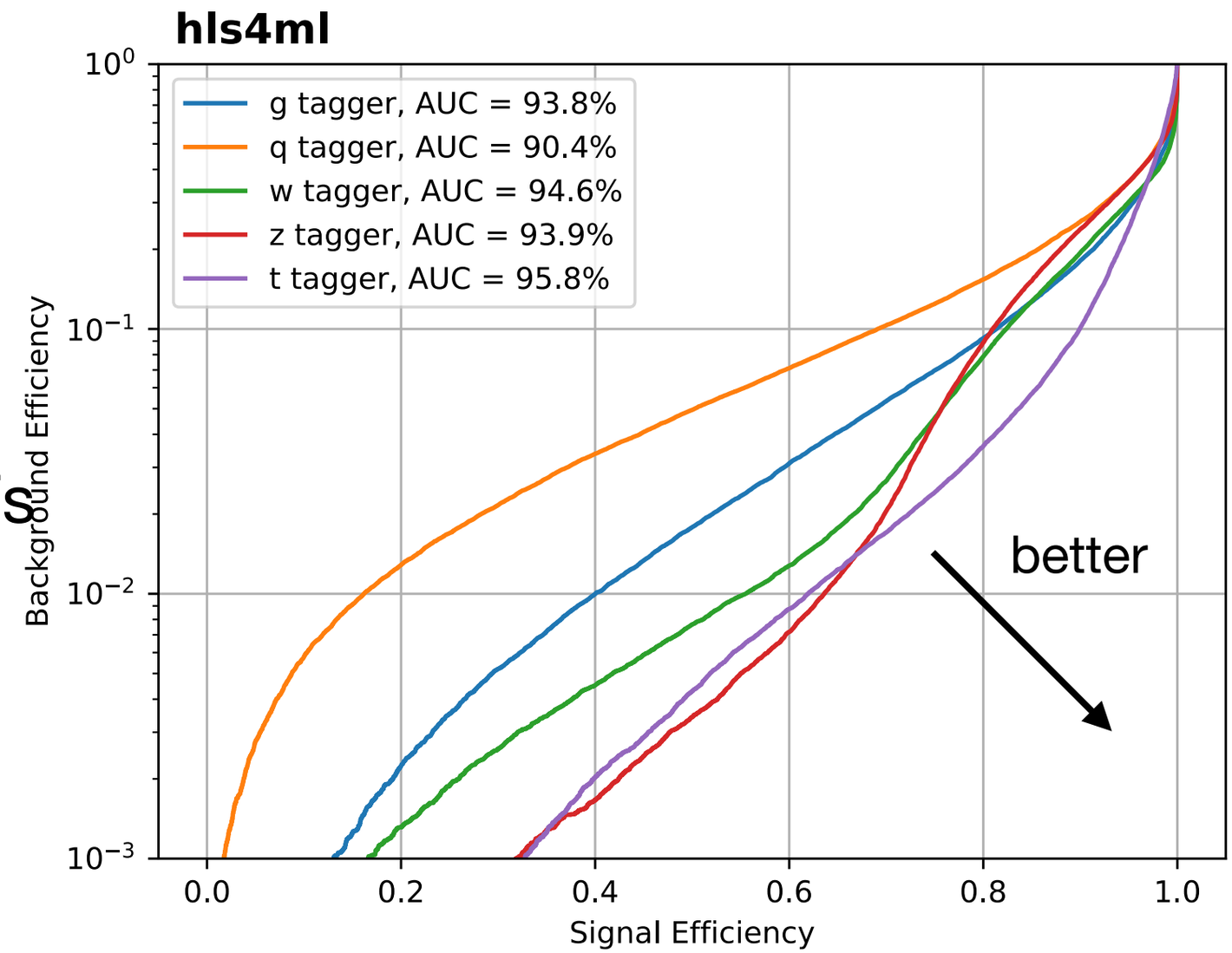


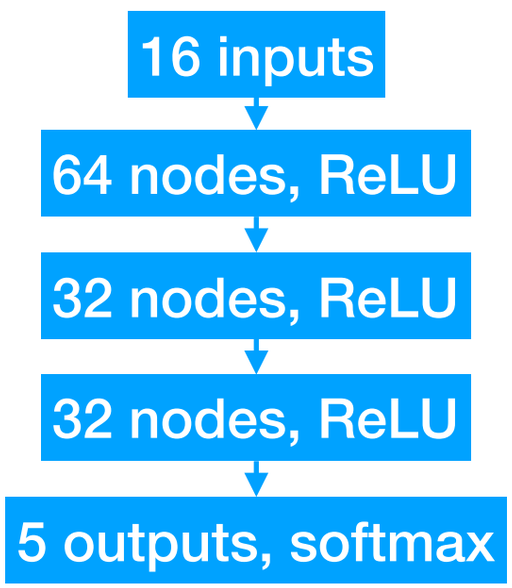
- **Groomed mass** separates top, W/Z, and quark/gluon
- top/gluon have greater **multiplicity** than W/Z/quark
- **ECF $N_2^{\beta=1}$** separates 2 and 3-prong jets (W/Z/top) from 1-prong jets (quark/gluon)

Case Study: Jet Substructure

- 5 output multi-classifier
  - Does a jet originate from a quark, gluon, W/Z boson, top quark?
- Fully connected network
- 16 expert inputs
  - jet mass, multiplicity, ECFs

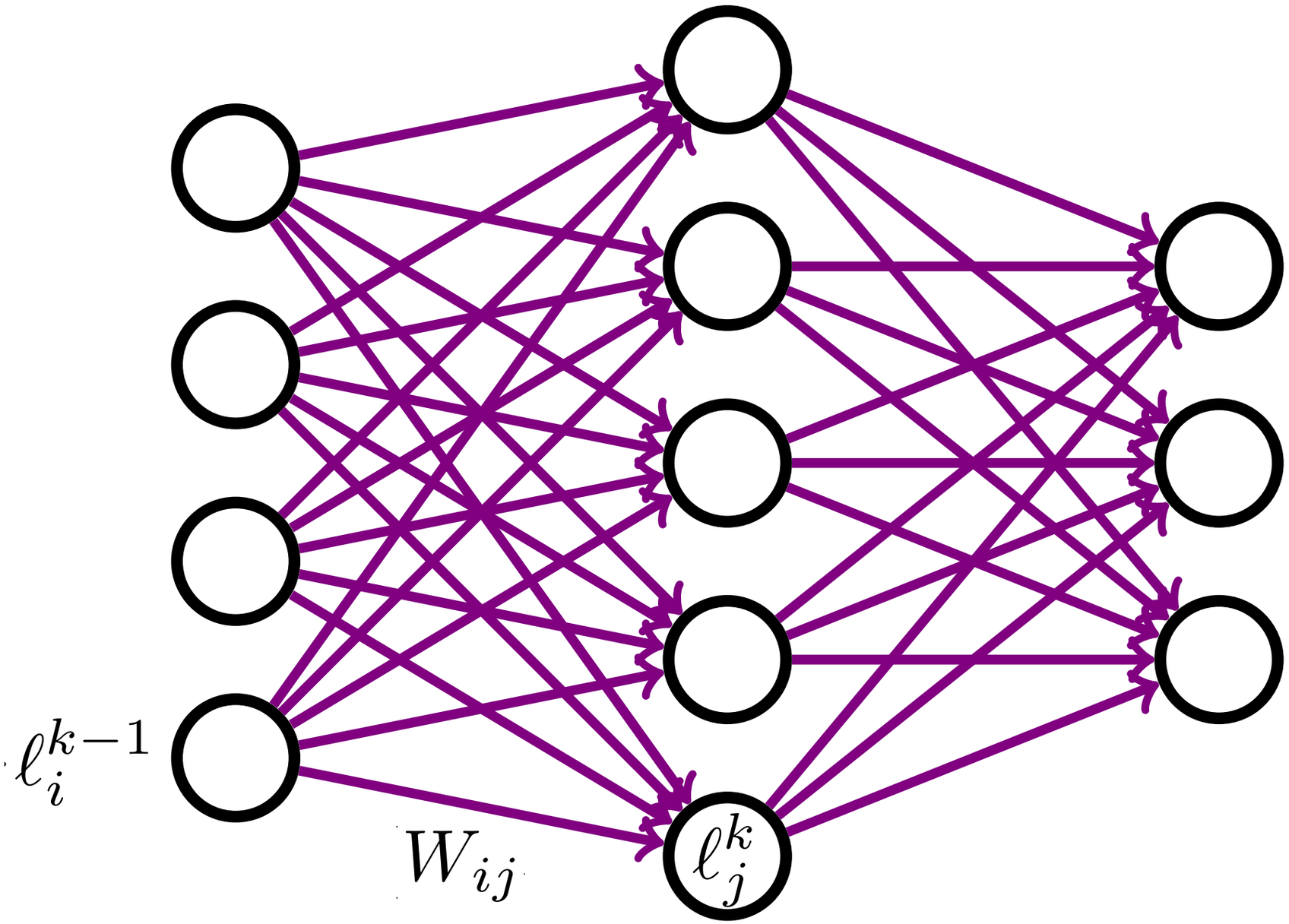
auc = area under ROC curve
(100% is perfect, 20% is random)





Neural Network

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



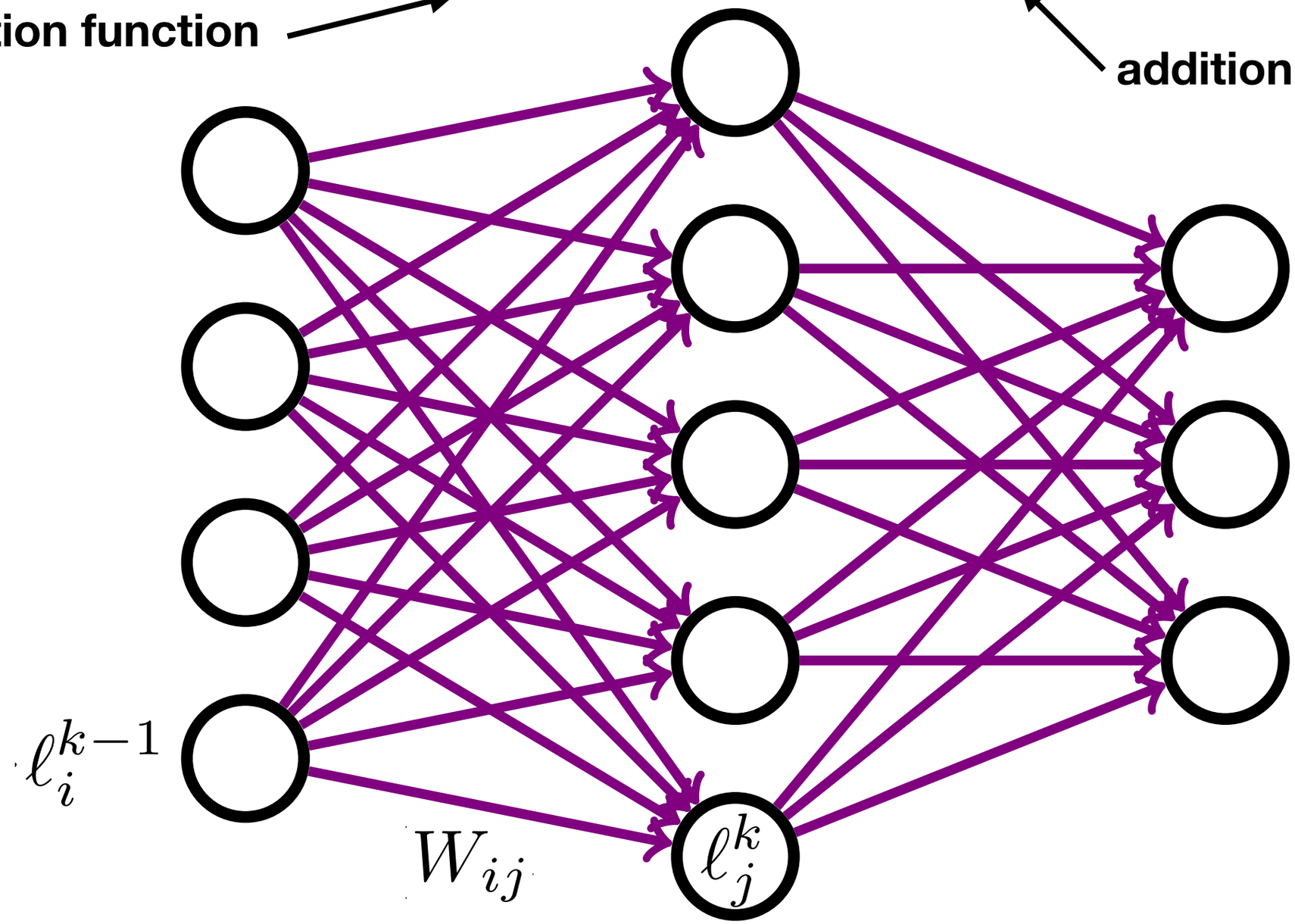
Neural Network

multiplication

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

activation function

addition



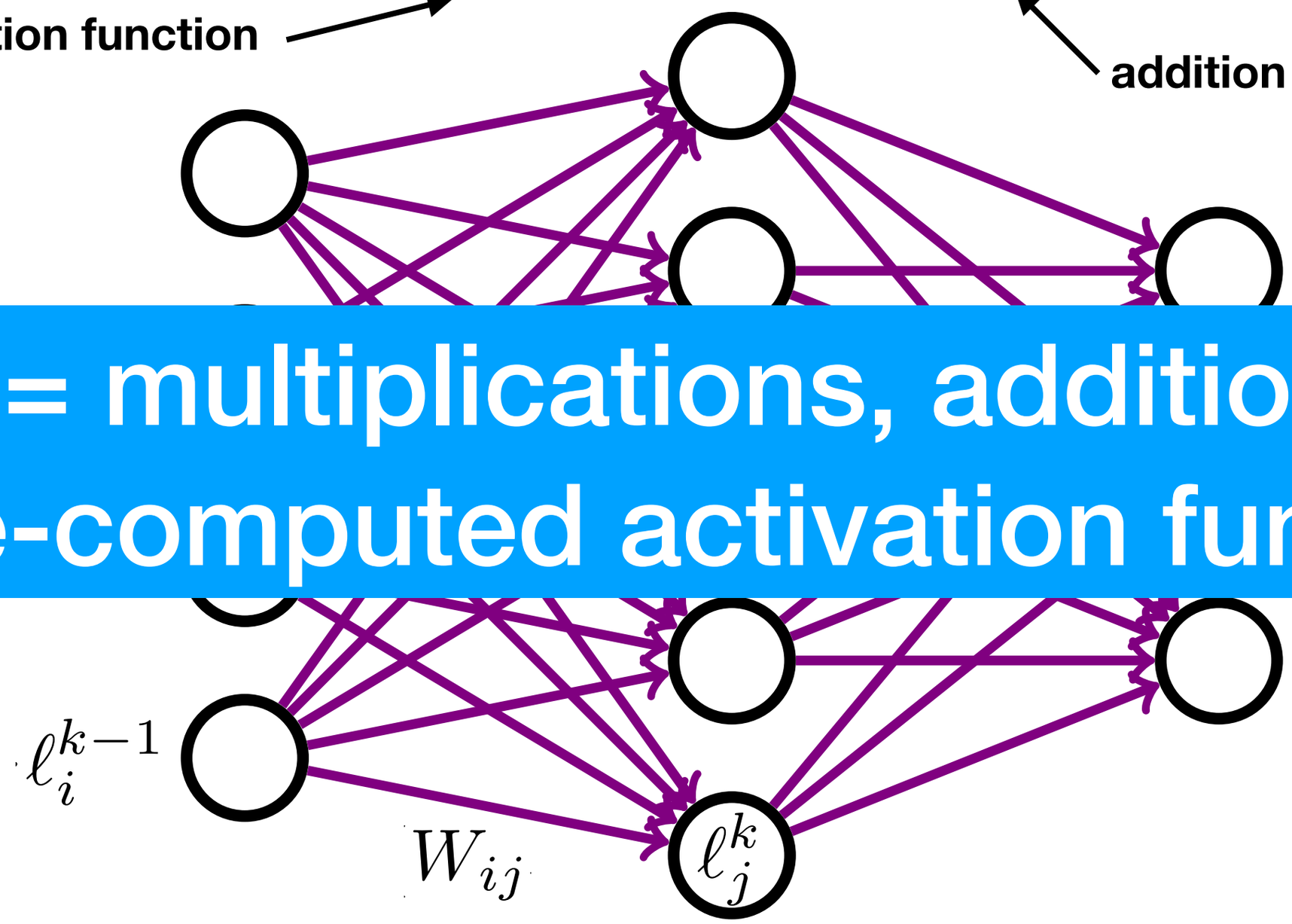
Neural Network

multiplication

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

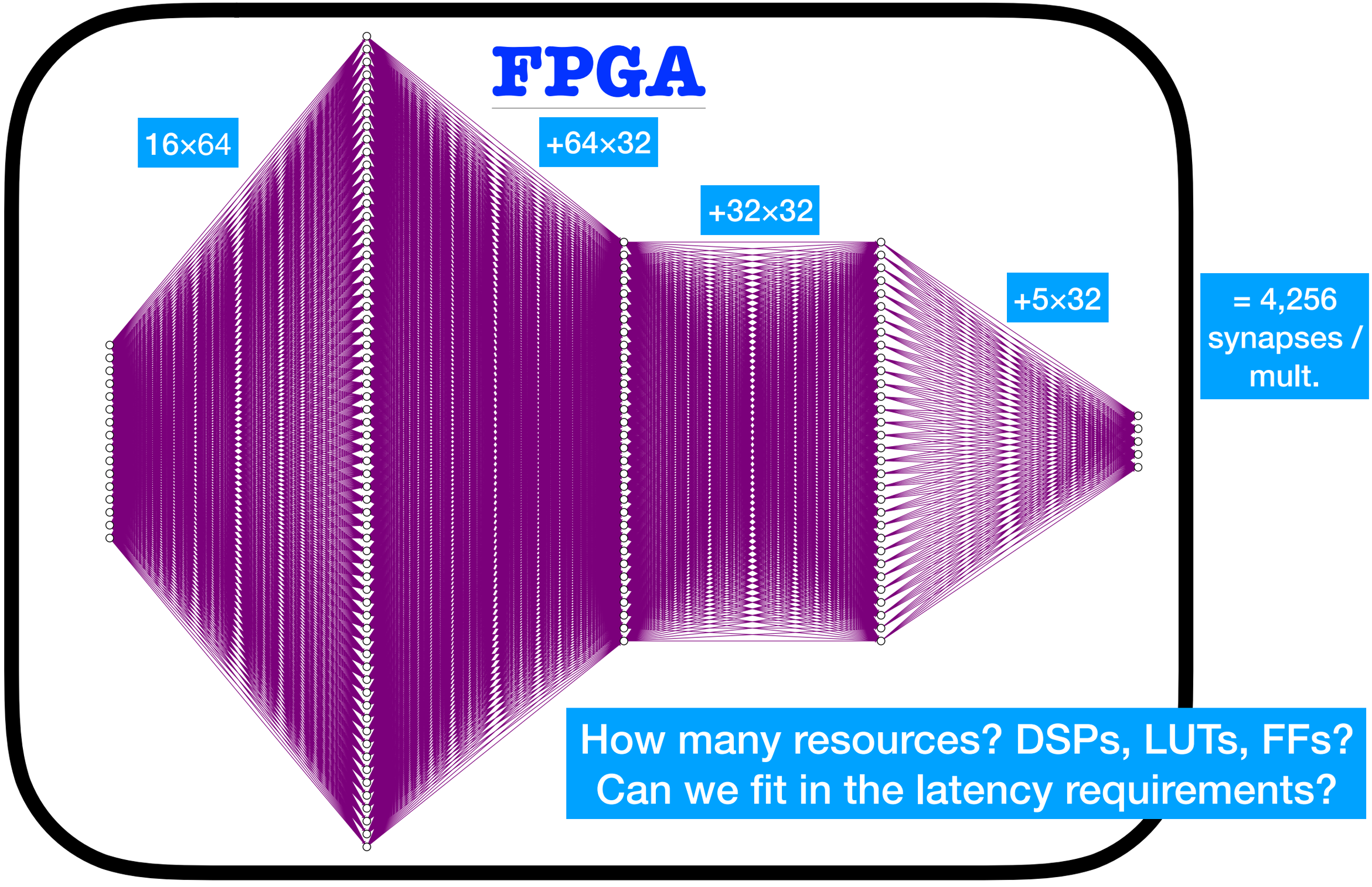
activation function

addition



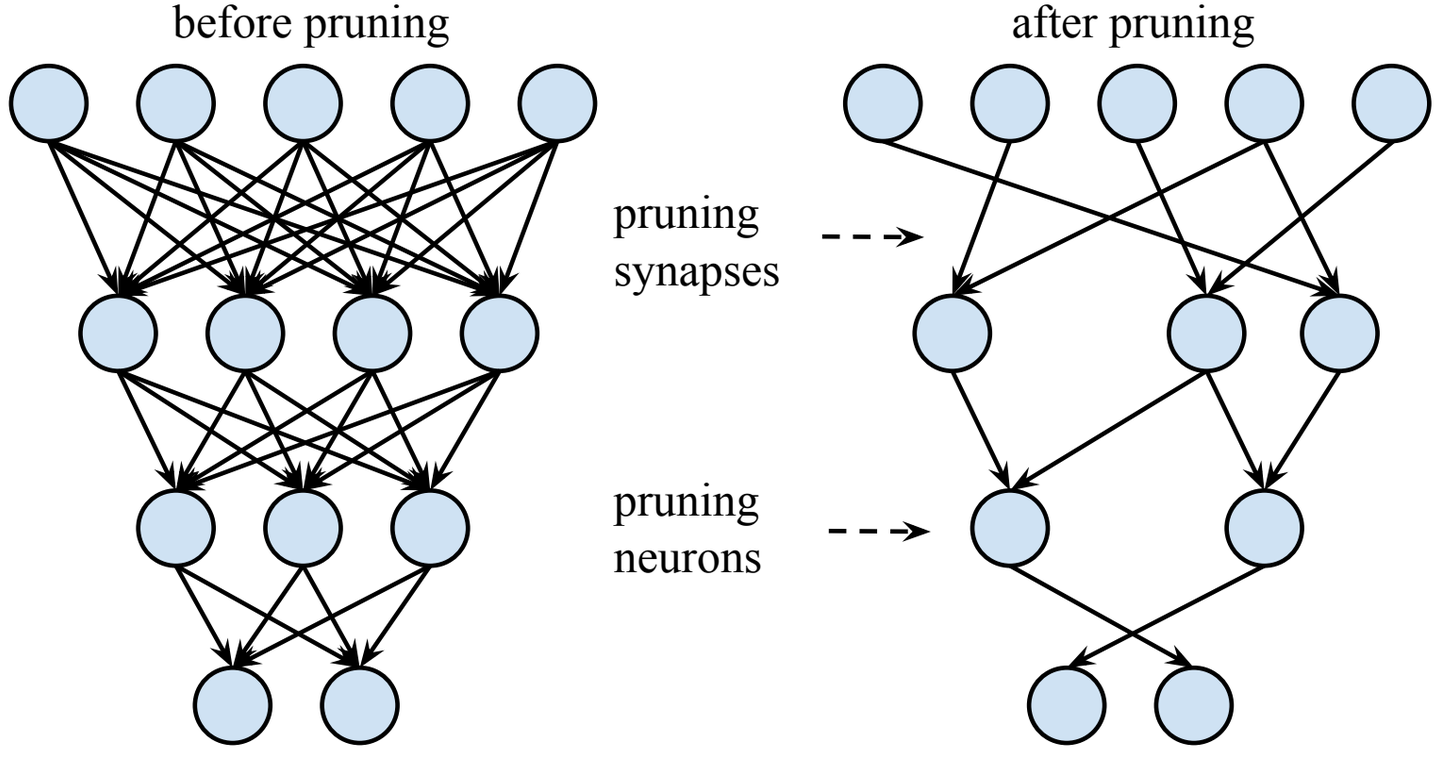
NN = multiplications, additions, and pre-computed activation functions

ML in FPGAs?


FPGA
16×64
+64×32
+32×32
+5×32
= 4,256 synapses / mult.
How many resources? DSPs, LUTs, FFs?
Can we fit in the latency requirements?

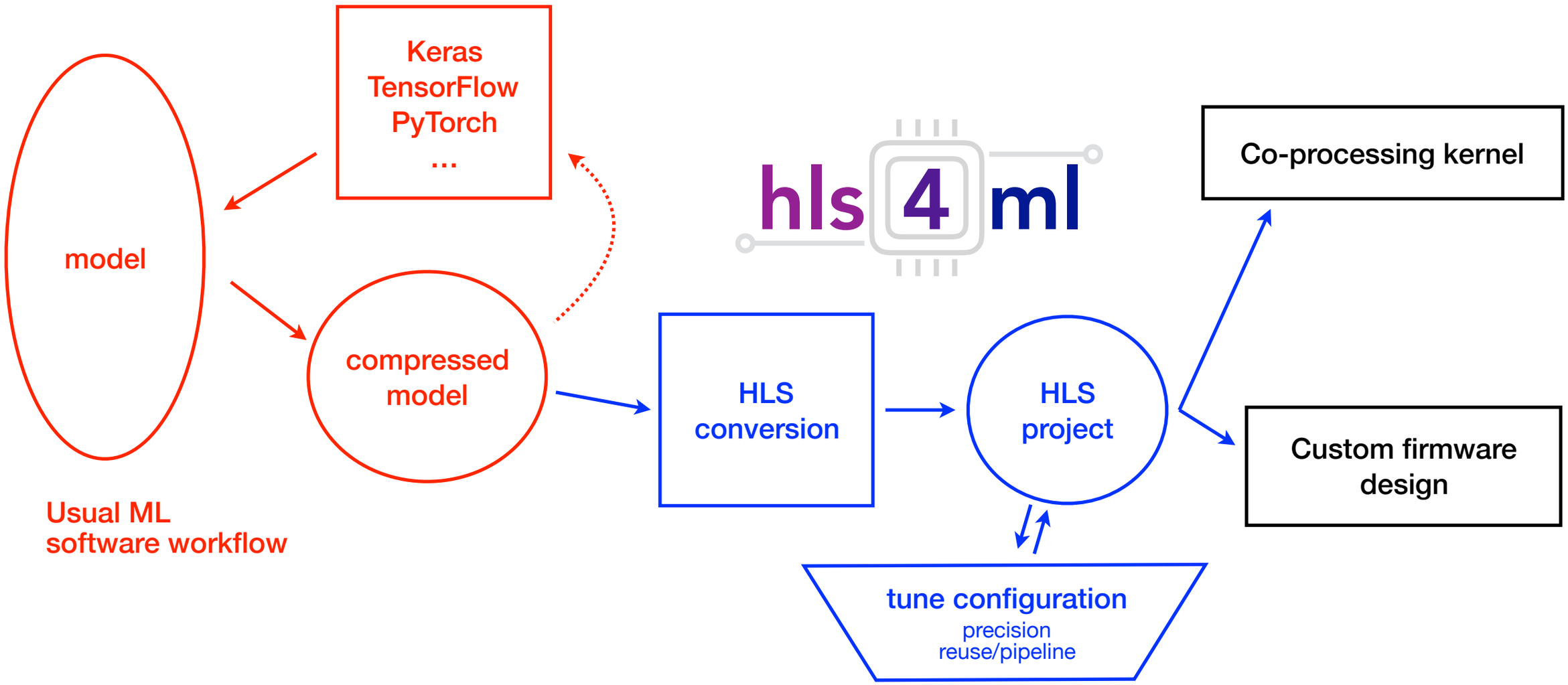
Efficient Neural Networks

- Compression
  - Maintain same performance while removing redundant synapses and neurons
- Quantization
  - 32-bit floating point math is overkill
  - 20-bit, 18-bit, 8-bit, …? fixed point, integers? binarized?



For further reading: [arXiv:1510.00149]

Design Exploration



Training

Training/Compression

Keras

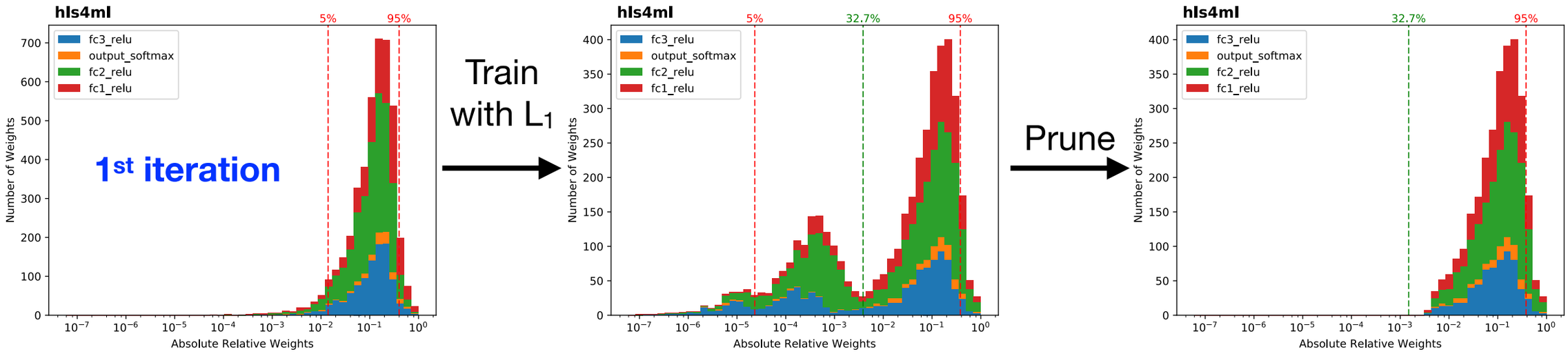
Example: https://github.com/hls-fpga-machine-learning/keras-training/

(Some useful stuff for compression too)

Export Model

```
json_string = keras_model.to_json()
keras_model.save_weights('my_model_weights.h5')
```

Training for Compression

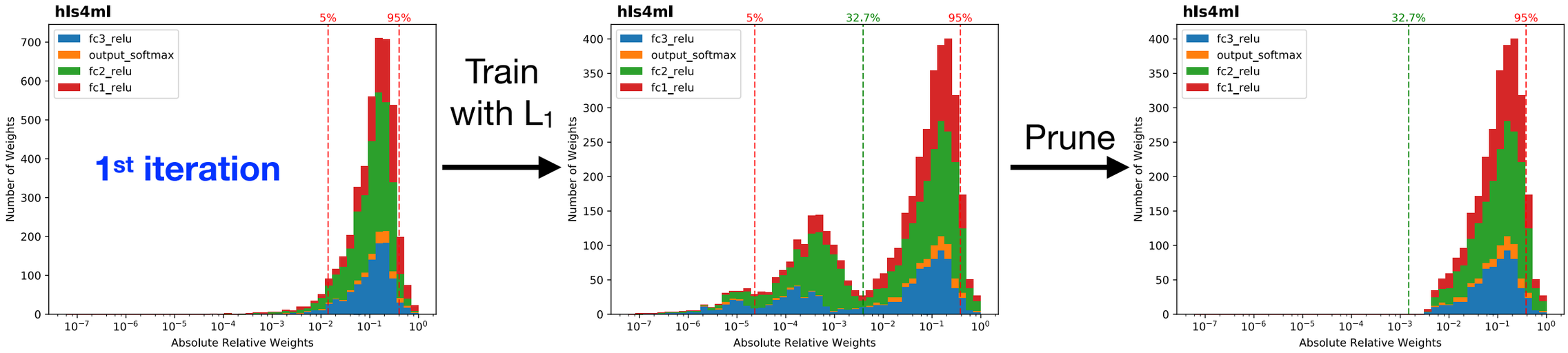


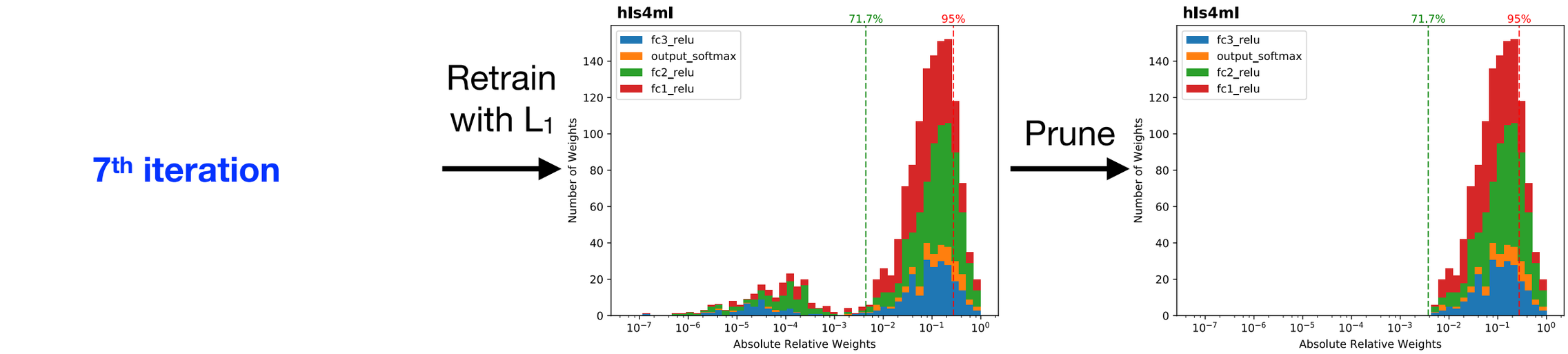
- Many possible schemes for compression
- Simple, iterative version:
  - Train with L_1 regularization (down-weights unimportant synapses)

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

  - Remove X% of weights and retrain
  - Repeat

Training for Compression





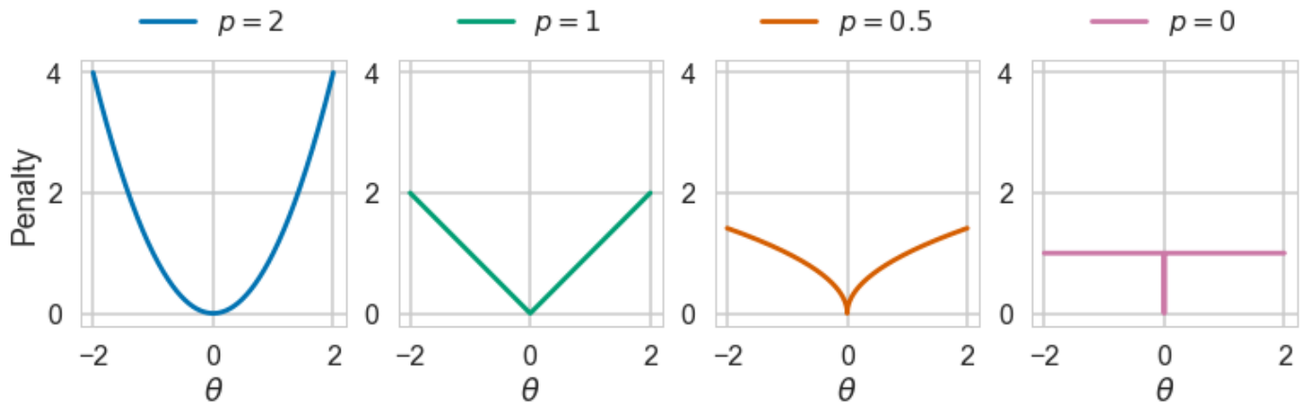
70% reduction with no loss in performance

Other Compression Schemes

- Train with L_p ($0 \leq p < 1$) regularization to promote sparsity (though difficult to optimize)
  - as $p \to 0$, $L_p \to L_0$

Louizos et al. 2017
arXiv:1712.01312

$$\|w\|_p = \left(\sum_i |w_i|^p\right)^{1/p}$$



- “Optimal brain damage:” use second derivatives of loss function to rank **parameter saliency** (rather than using parameter magnitude)

LeCun et al. 1989
NIPS 250

- Weight-sharing using k-means clustering to identify weights to share

Han et al. 2015
arXiv:1510.00149

- Huffman coding (optimal prefix)

Translation

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

Inputs →

Config

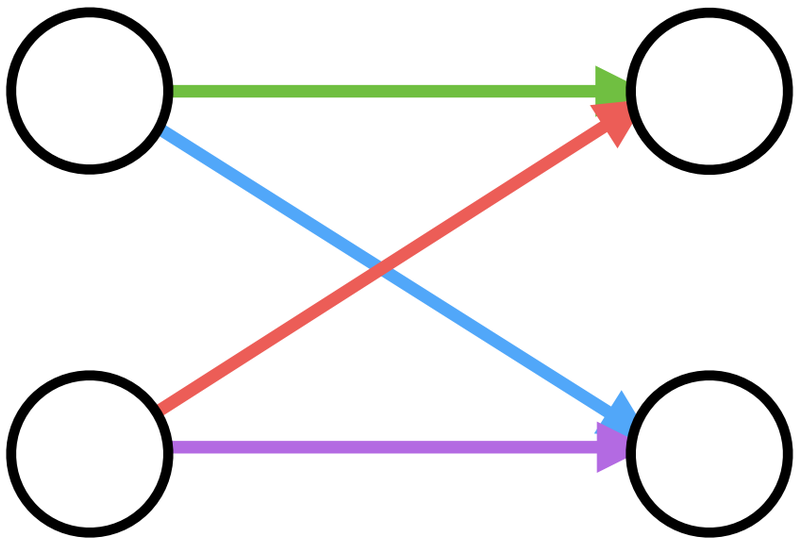
```
KerasJson: example-keras-model-files/KERAS_1layer.json
KerasH5:   example-keras-model-files/KERAS_1layer_weights.h5
OutputDir: my-hls-test
ProjectName: myproject
XilinxPart: xc7vx690tffg1927-2
ClockPeriod: 5

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

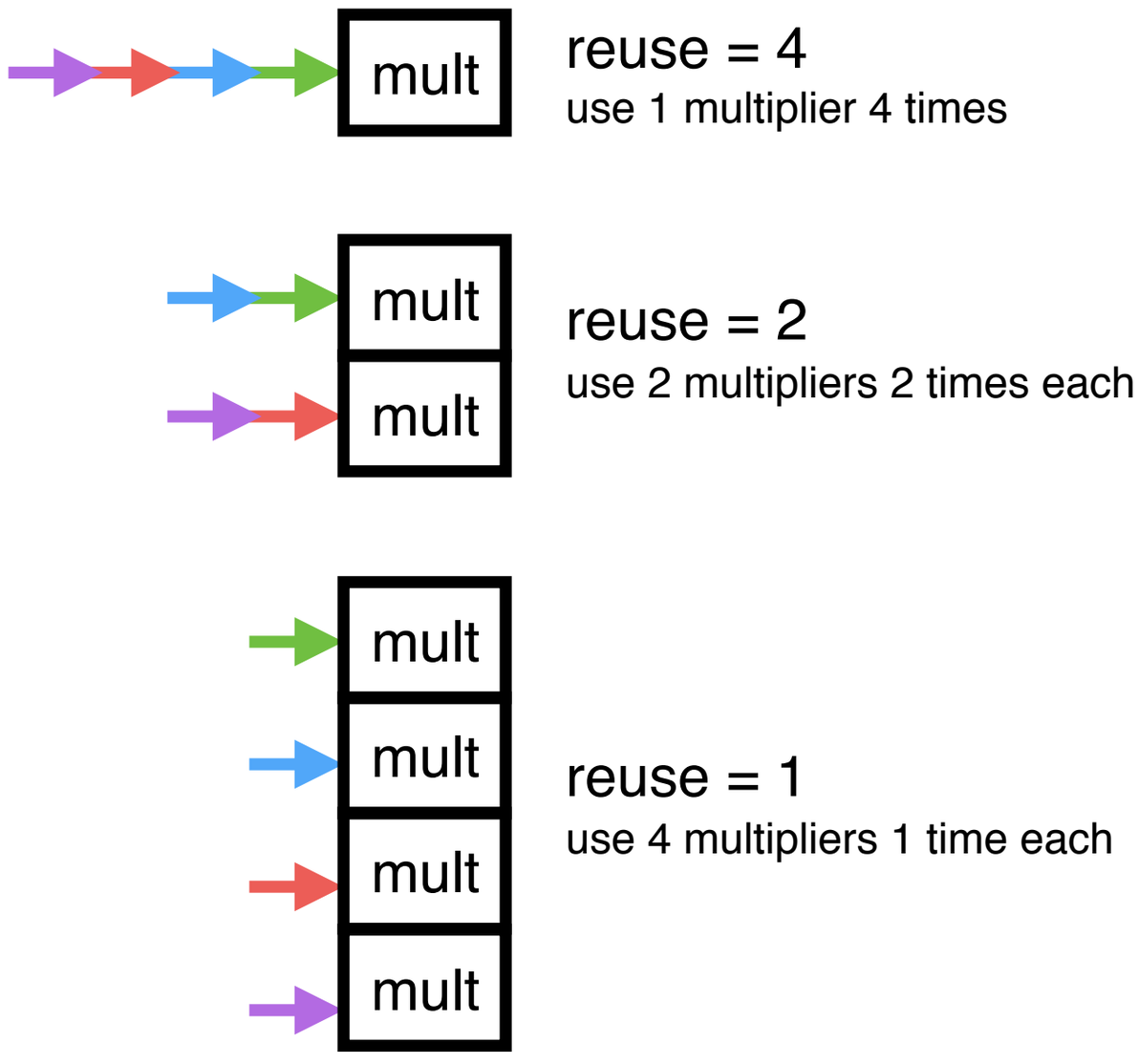
- `IOType`: parallelize or serialize
- `ReuseFactor`: how much to parallelize
- `DefaultPrecision`: inputs, weights, biases

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

Network Tuning: Parallelization

- `ReuseFactor`: how much to parallelize





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

HLS Project

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

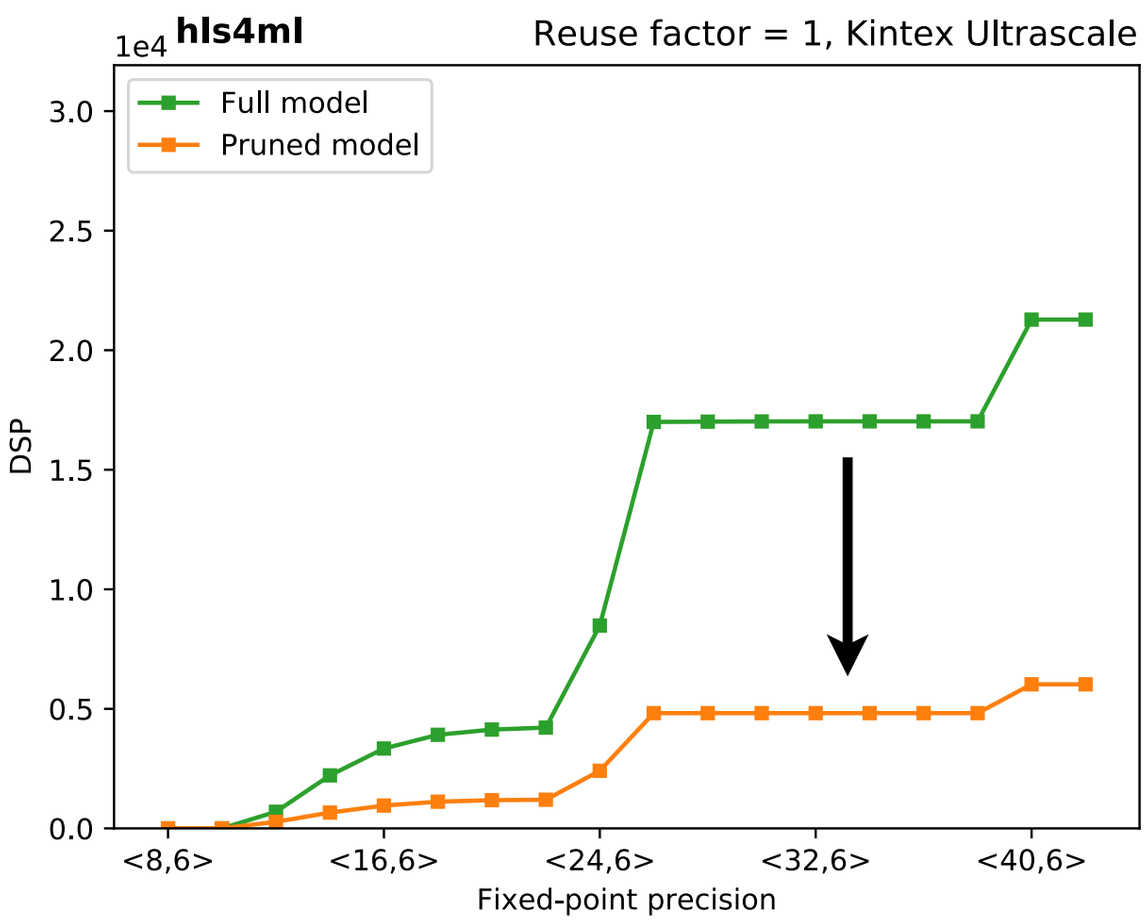
```
##################
#    HLS4ML      #
##################
open_project -reset myproject_prj
set_top myproject
add_files firmware/myproject.cpp -cflags "-I[file normalize ../../nnet_utils]"
add_files -tb myproject_test.cpp -cflags "-I[file normalize ../../nnet_utils]"
add_files -tb firmware/weights
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
```

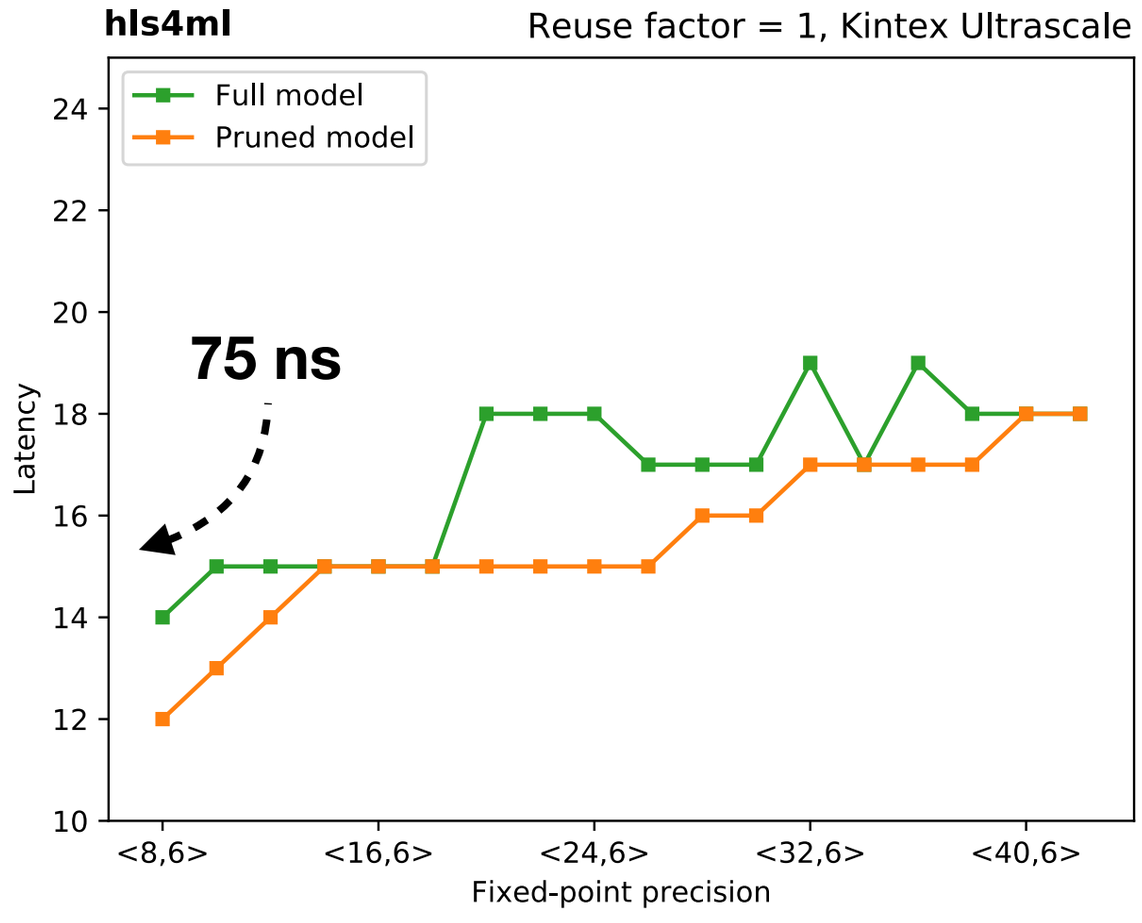
produces an “IP core” that can be dropped into a full firmware design

Study Results

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

Compression

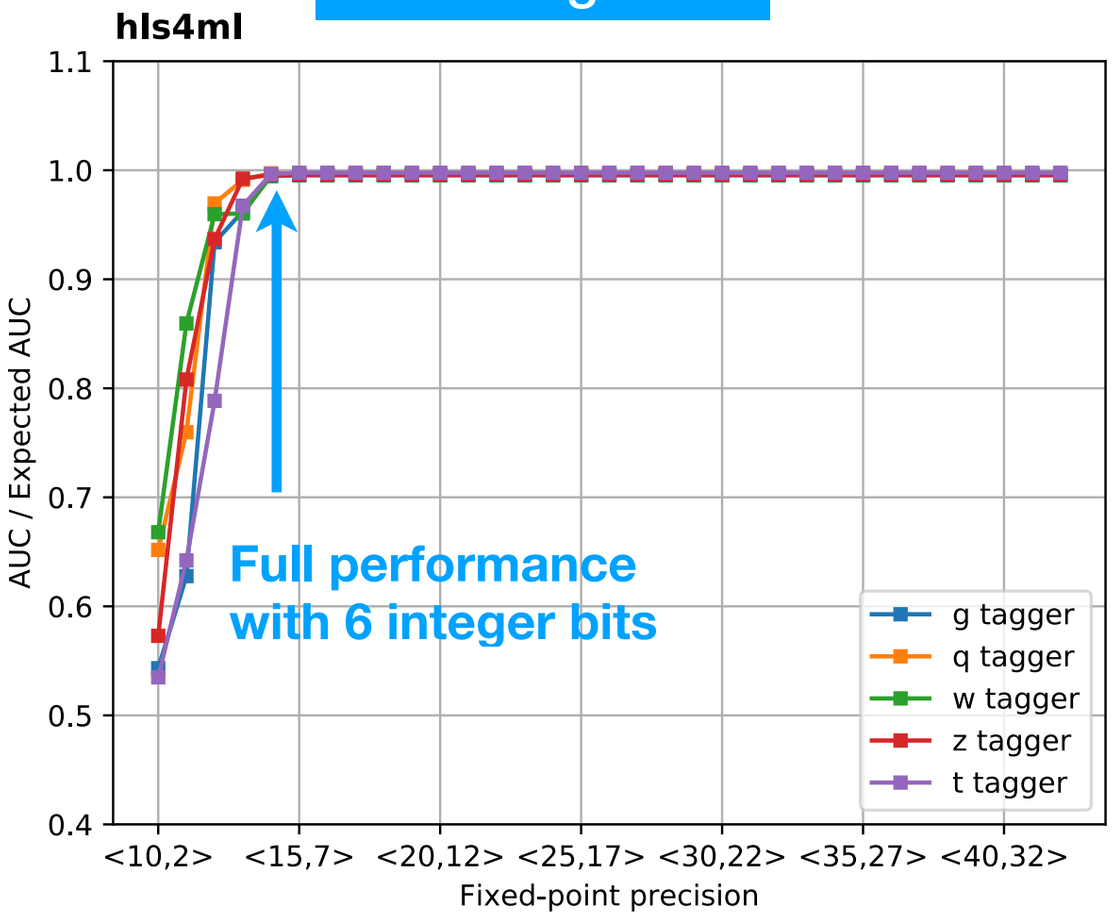




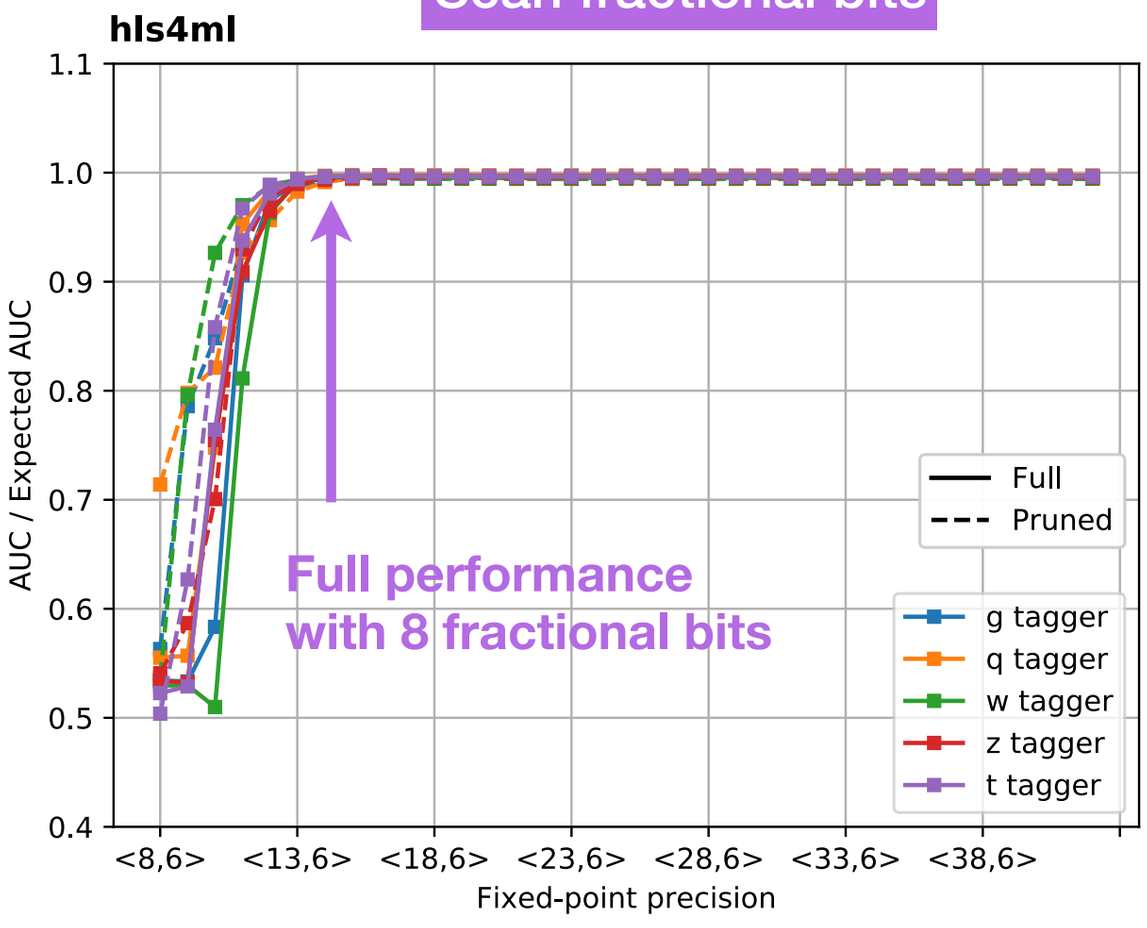
- Big reduction in DSP usage with pruned model!
- ~15 clocks @ 200 MHz = 75 ns inference

Quantization

Scan integer bits

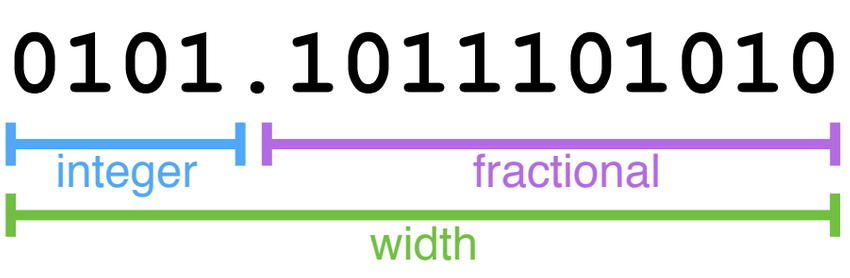


Scan fractional bits

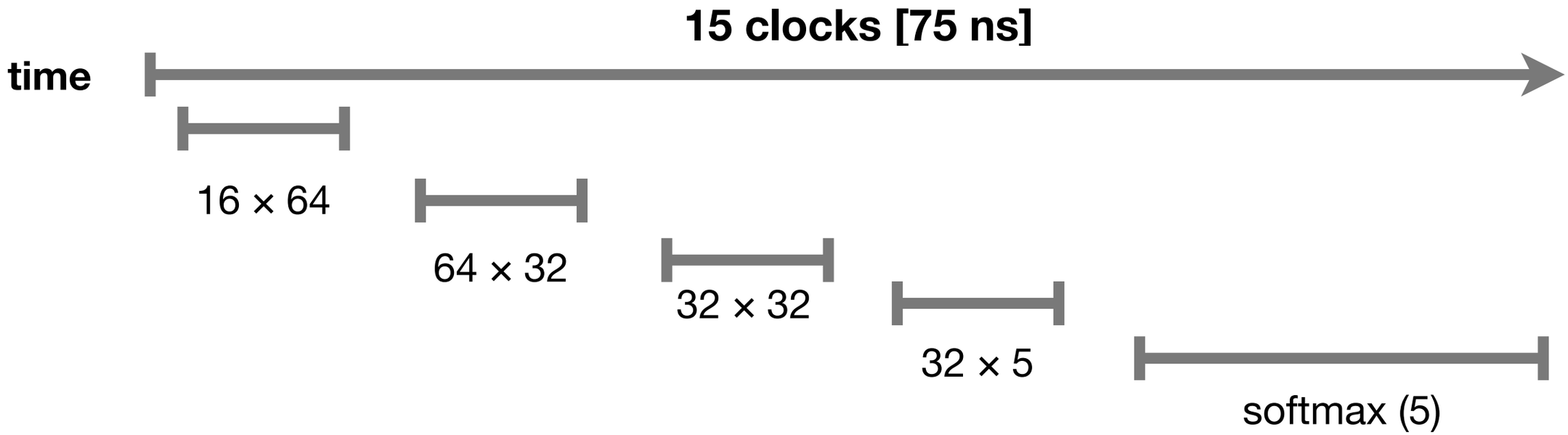


- General strategy: avoid overflows in integer bit then scan the decimal bit until reaching optimal performance

ap_fixed<width,integer>

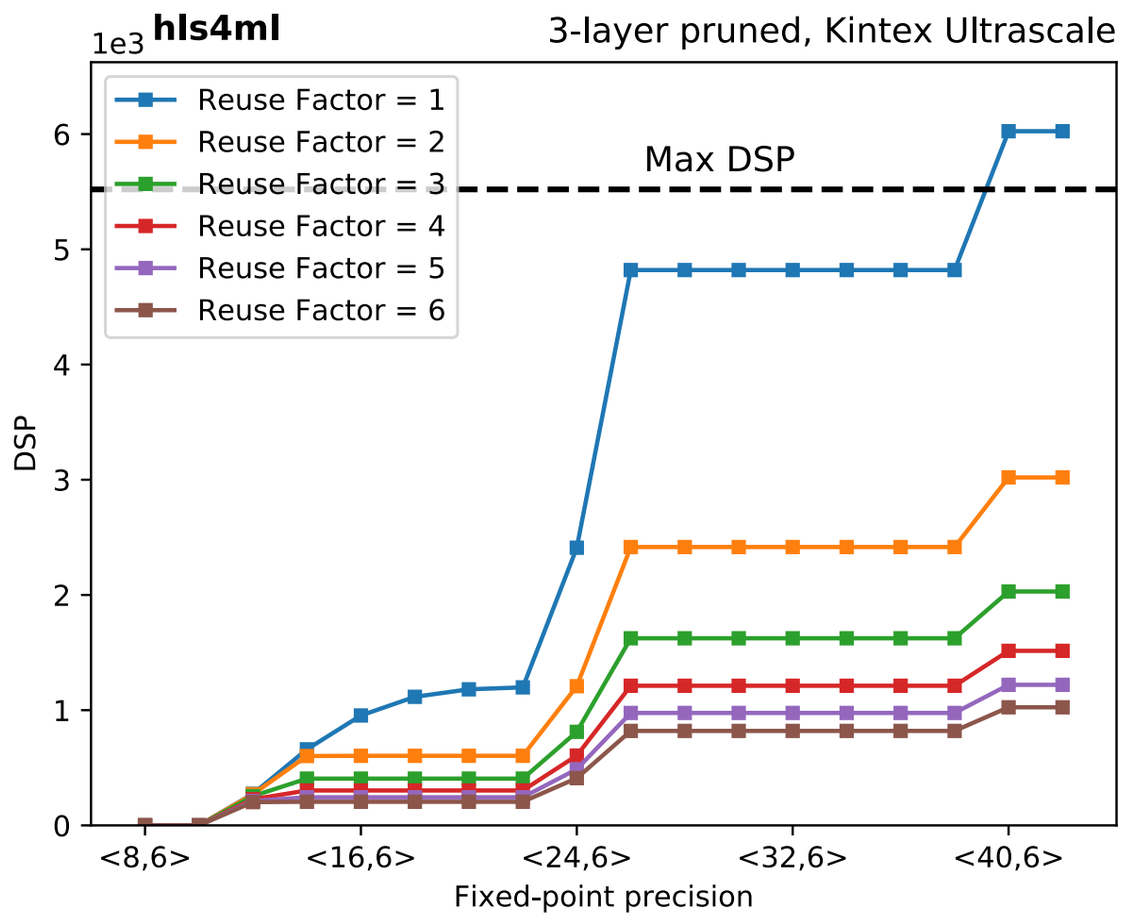


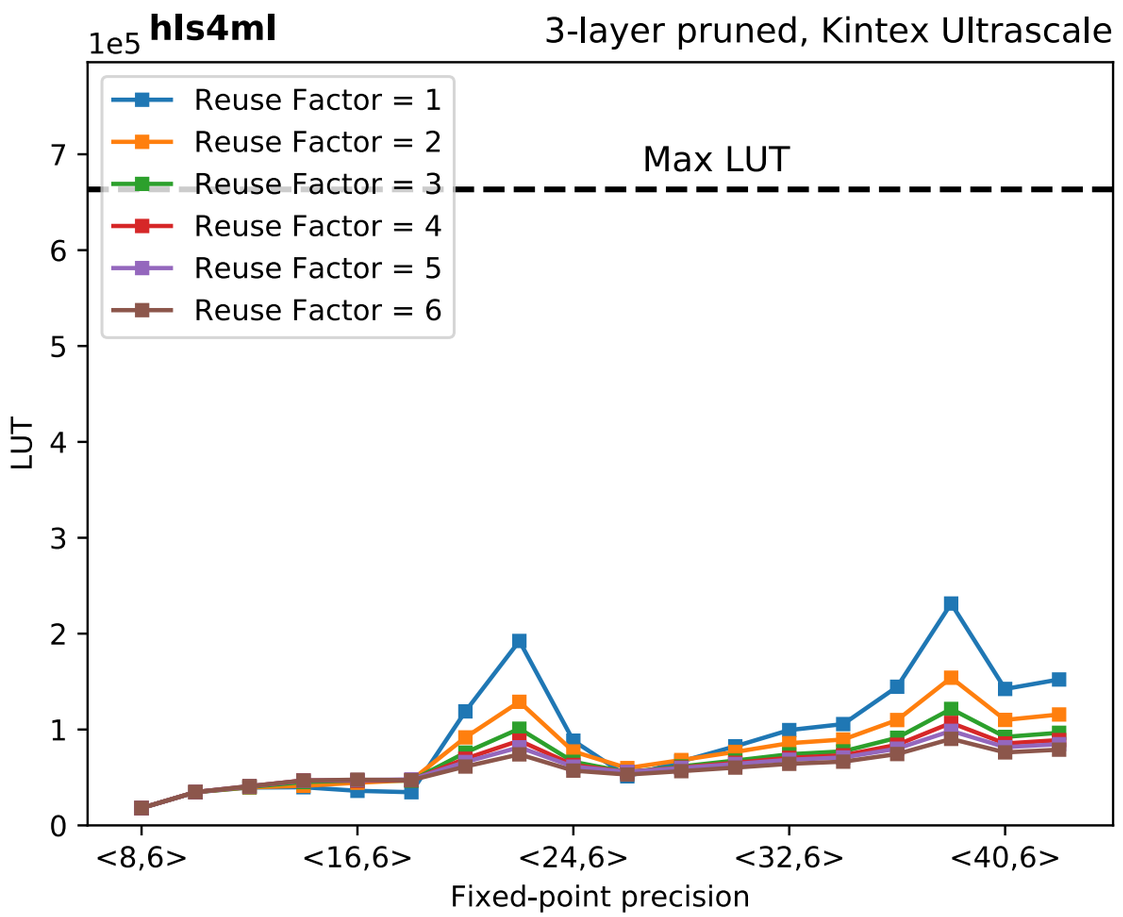
Resource Usage and Timing



reuse = 1 <16, 6> bits	BRAM	DSP	FF	LUT
Total	13	954	53k	36k
% Usage	~0%	17%	3%	5%

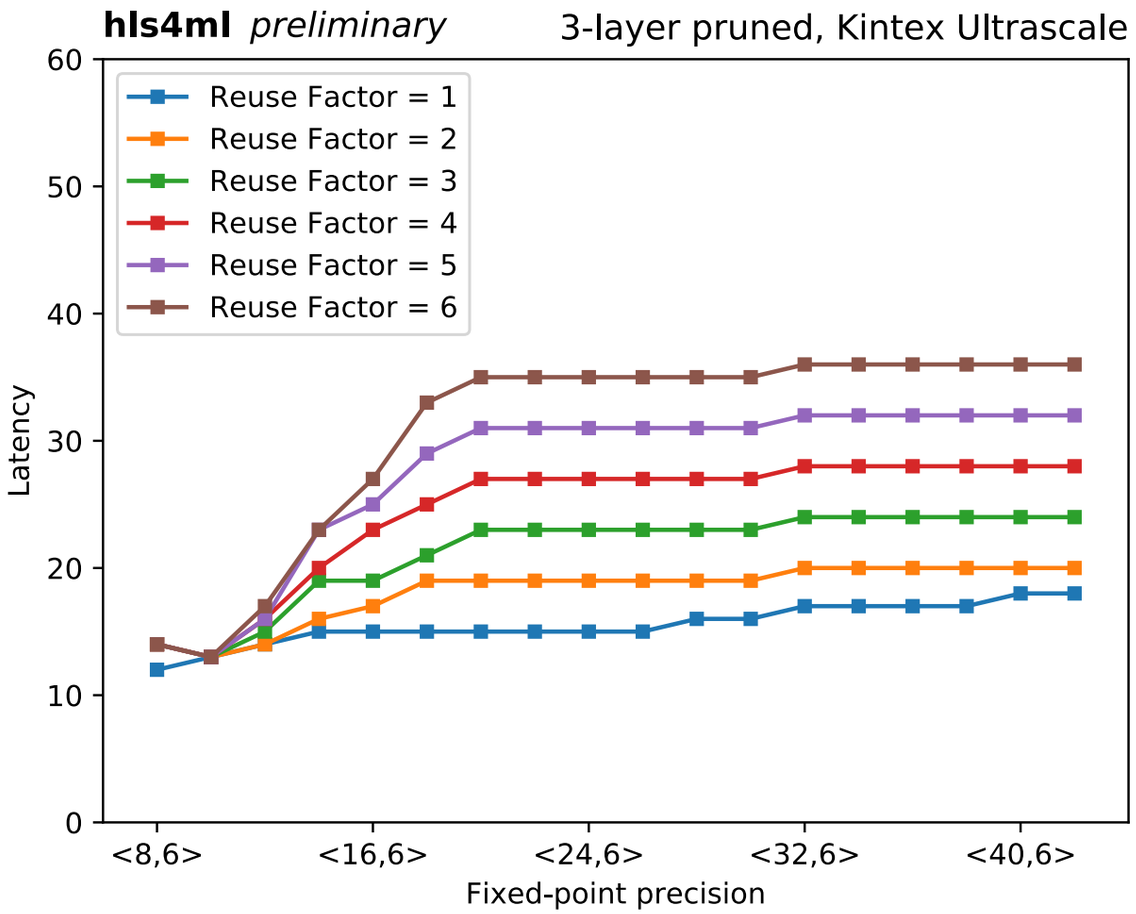
Resource Usage with Reuse

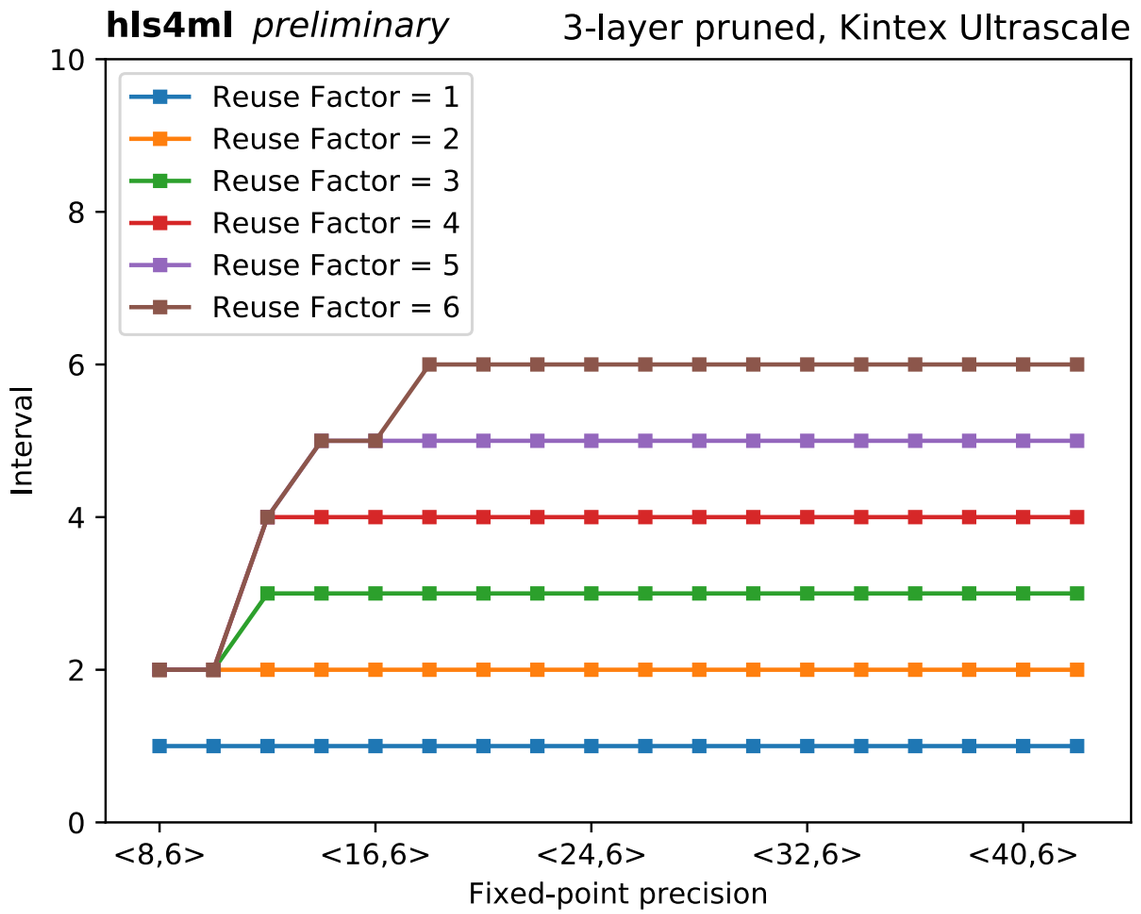




- Tuning the throughput with reuse factor reduces the DSP usage
- Steady increase of LUTs and FFs vs. bit precision
- Spikes in LUTs at the DSP precision transitions (not present in final implementation)

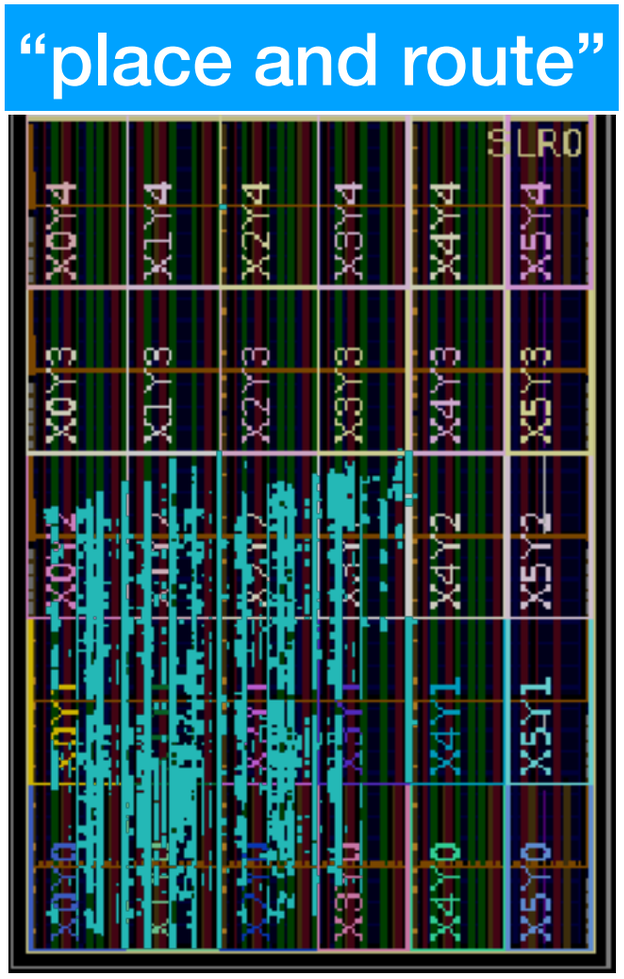
Timing with Reuse





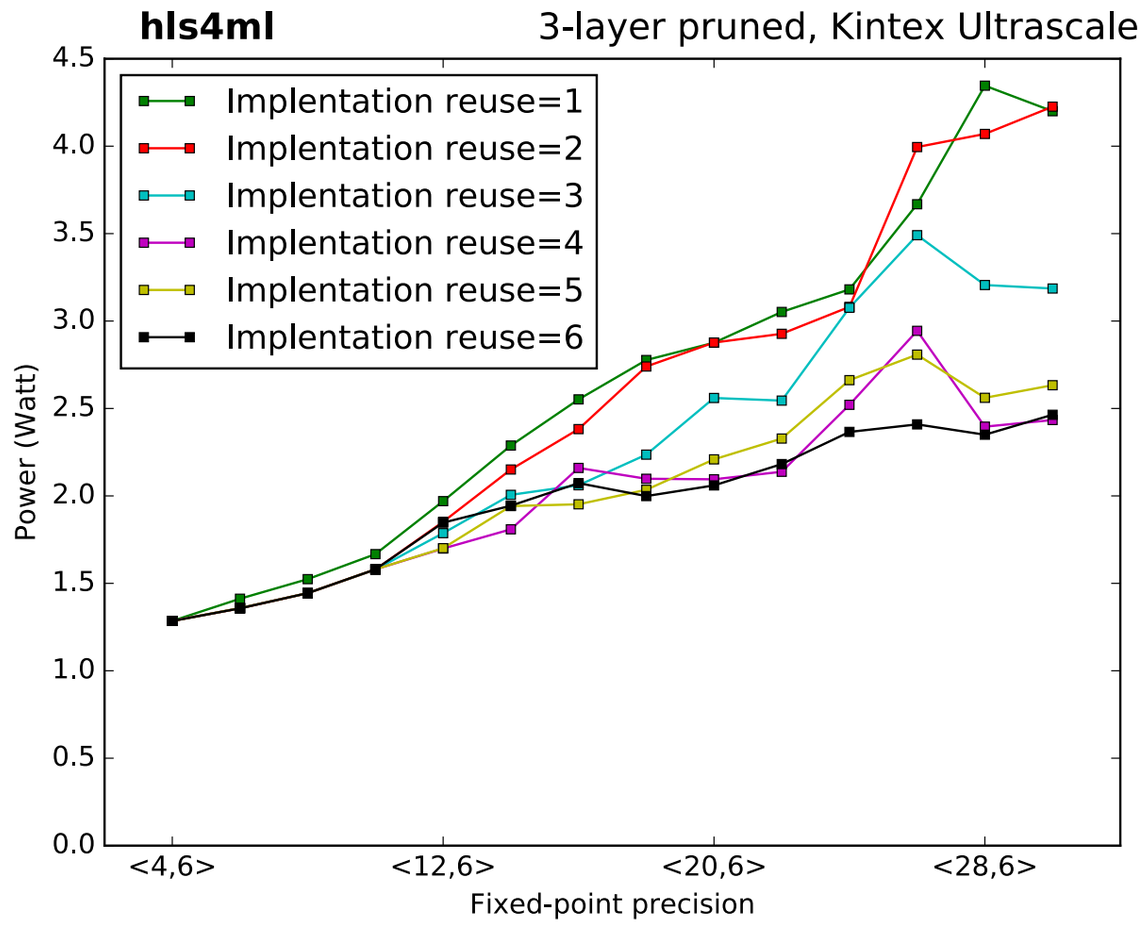
- Additional latency introduced by reusing the multipliers
- Initiation interval scales with the reuse factor

Implementing the HLS Design

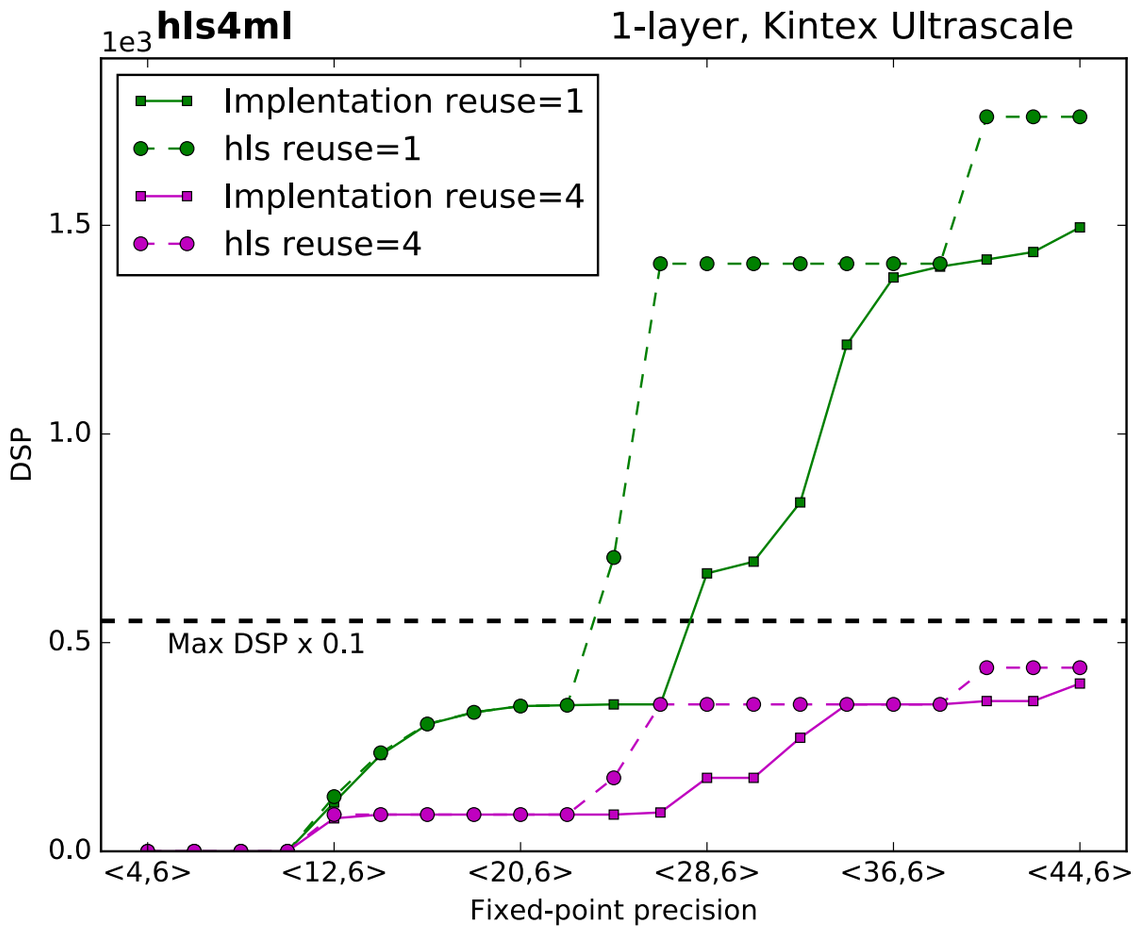


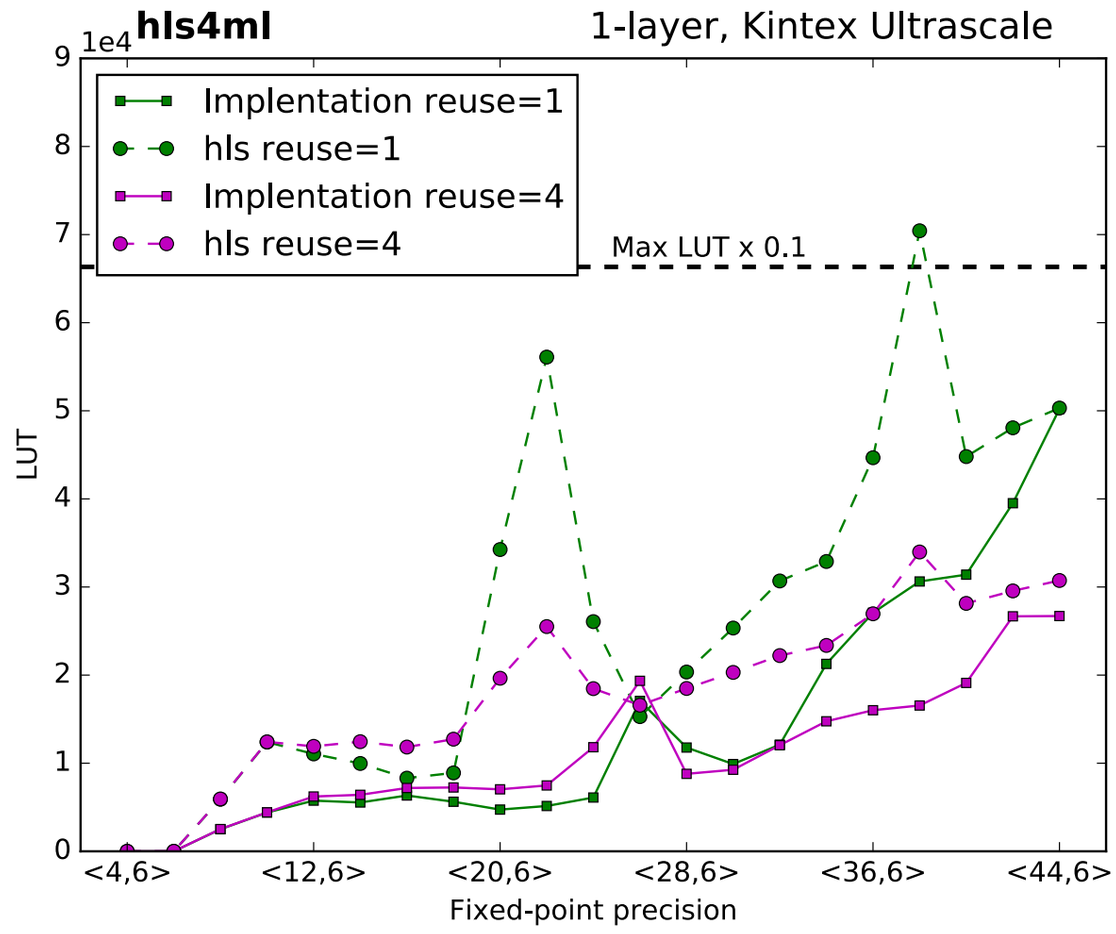
- How optimal is the HLS design (vs. RTL)?
  - For DSPs, HLS seems close to "back-of-the-envelope" optimal estimate
- HLS is good for quickly getting a **conservative** estimate of resources

- Power decreases as throughput is decreased (by increasing reuse factor)



HLS vs. Implementation





- HLS estimates are **conservative** compared to final implementation
- No spikes in LUTs at the DSP precision transitions in implementation

Note: different model

Summary and Outlook

hls4ml Status

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

- Beta version is live! arXiv:1804.06913
- Next steps:
  - **Applications?**
  - **Bigger networks?**

Fast inference of deep neural networks in FPGAs for particle physics

Javier Duarte[a] , Song Han[b] , Philip Harris[b] , Sergo Jindariani[a] , Edward Kreinar[c] , Benjamin Kreis[a] , Jennifer Ngadiuba[d] , Maurizio Pierini[d] , Ryan Rivera[a] , Nhan Tran[a] , Zhenbin Wu[e]

[a] *Fermi National Accelerator Laboratory, Batavia, IL 60510, USA*
[b] *Massachusetts Institute of Technology, Cambridge, MA 02139, USA*
[c] *HawkEye360, Herndon, VA 20170, USA*
[d] *CERN, CH-1211 Geneva 23, Switzerland*
[e] *University of Illinois at Chicago, Chicago, IL 60607, USA*

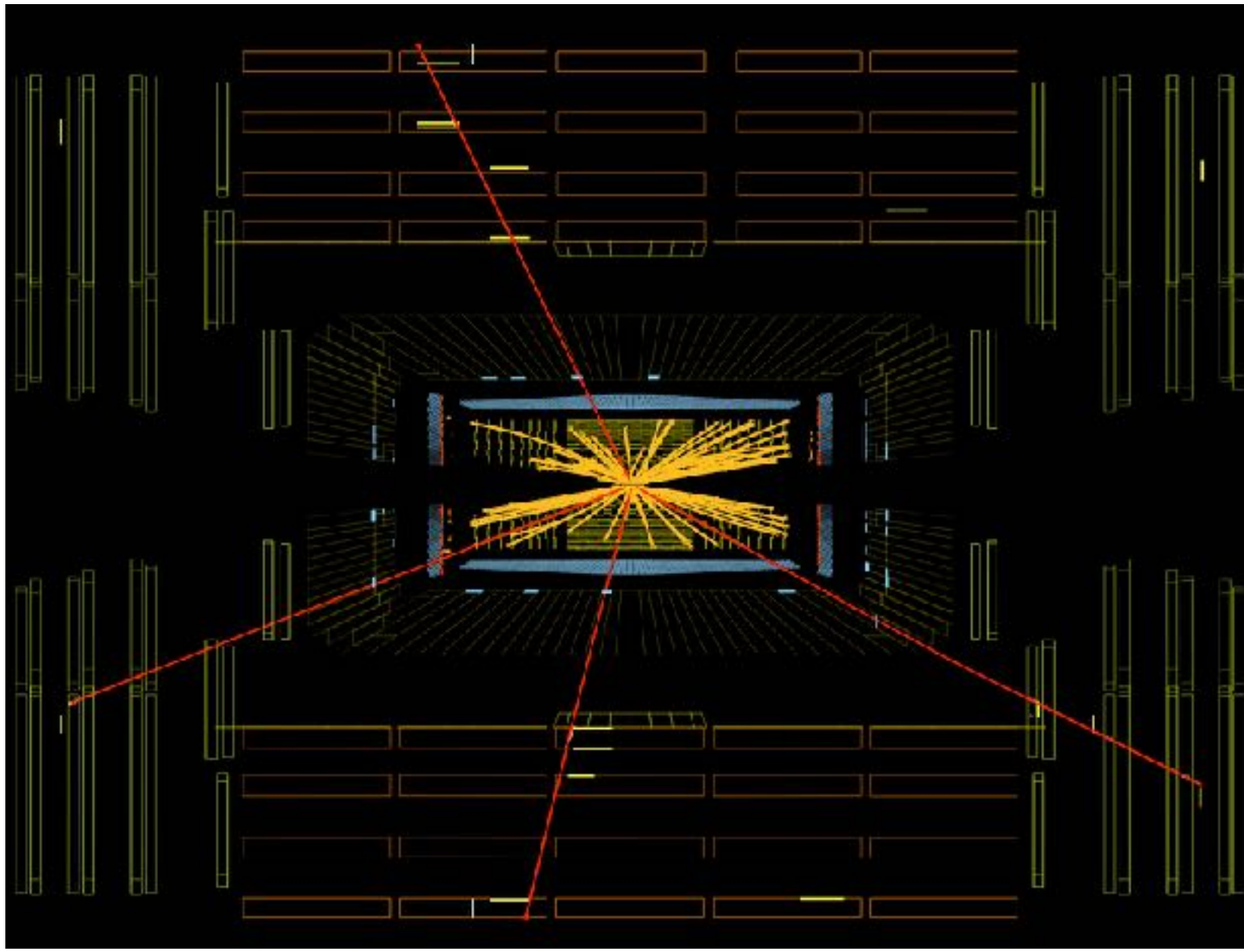
E-mail: hls4ml.help@gmail.com

Abstract: Recent results at the Large Hadron Collider (LHC) have pointed to enhanced physics capabilities through the improvement of the real-time event processing techniques. Machine learning methods are ubiquitous and have proven to be very powerful in LHC physics, and particle physics as a whole. However, exploration of the use of such techniques in low-latency, low-power FPGA hardware has only just begun. FPGA-based trigger and data acquisition (DAQ) systems have extremely low, sub-microsecond latency requirements that are unique to particle physics. We present a case study for neural network inference in FPGAs focusing on a classifier for jet substructure which would enable, among many other physics scenarios, searches for new dark sector particles and novel measurements of the Higgs boson. While we focus on a specific example, the lessons are far-reaching. We develop a package based on High-Level Synthesis (HLS) called hls4ml to build machine learning models in FPGAs. The use of HLS increases accessibility across a broad user community and allows for a drastic decrease in firmware development time. We map out FPGA resource usage and latency versus neural network hyperparameters to identify the problems in particle physics that would benefit from performing neural network inference with FPGAs. For our example jet substructure model, we fit well within the available resources of modern FPGAs with a latency on the scale of 100 ns.

L1 Muon Trigger Application

ACAT 2017 BDT L1T

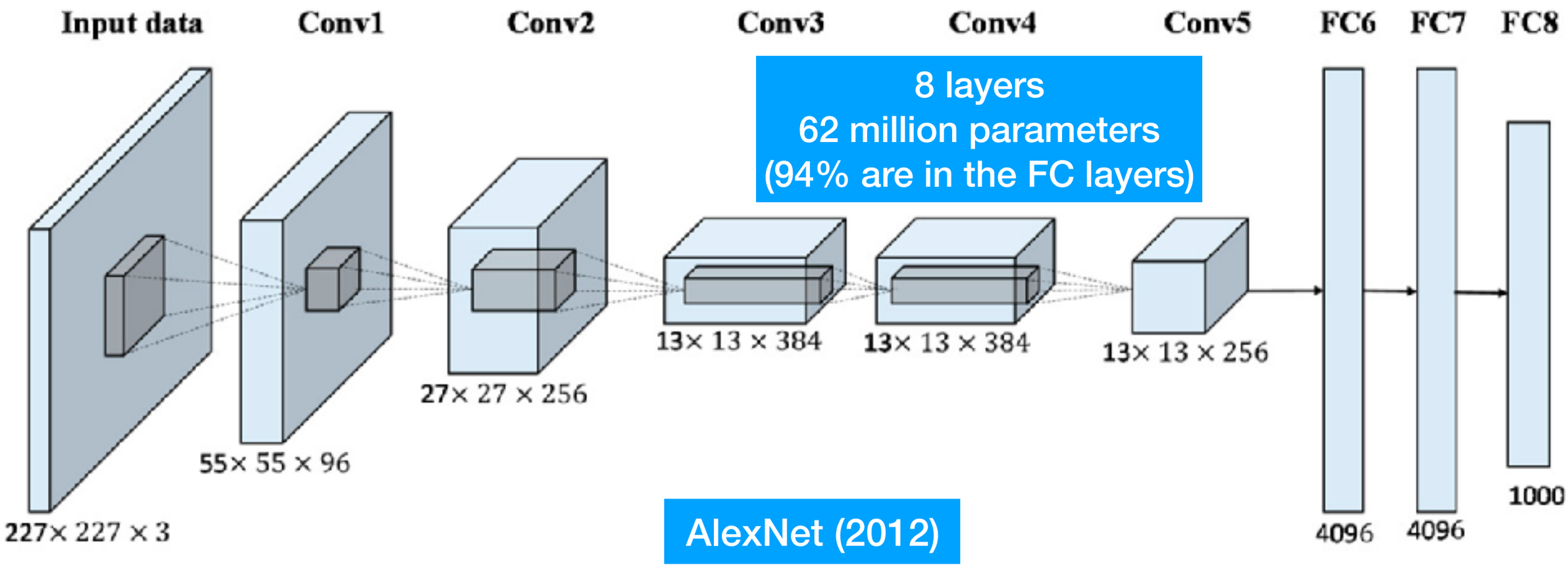
- Current **BDT** for CMS Level-1 muon p_T assignment based on defection angles (ΔΦ, Δθ) and other variables
  - Implemented using a 1 GB pre-computed LUT that stores the BDT output for every the possible input (compressed to 30-bits)
- CMS studying NN implemented with `hls4ml`, which allows additional inputs (e.g. 68 variables with 18 bits each = 1228 bits!) and improved p_T resolution



Big Convolutional Neural Networks

- Main task is computer vision/image recognition
- Control the number of parameters by baking in assumptions like locality and translation invariance to **share weights** within a layer

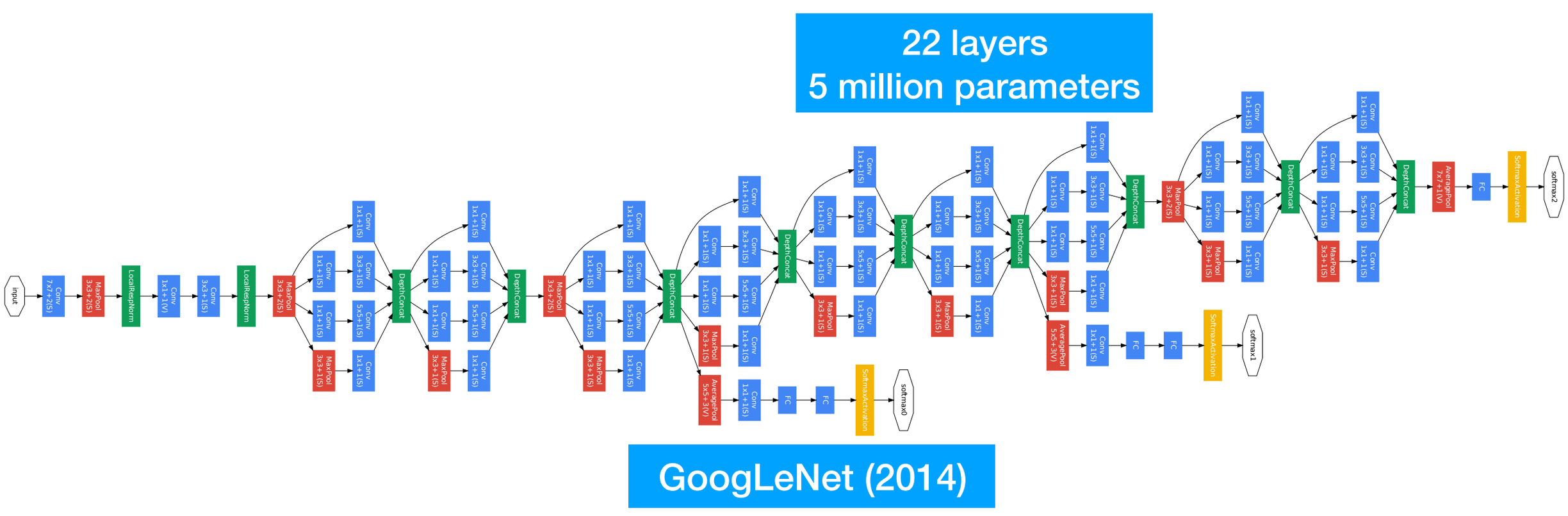
Krizhevsky, et al.
NIPS 4824



Big Convolutional Neural Networks

- Main task is computer vision/image recognition
- Control the number of parameters by baking in assumptions like locality and translation invariance to **share weights** within a layer

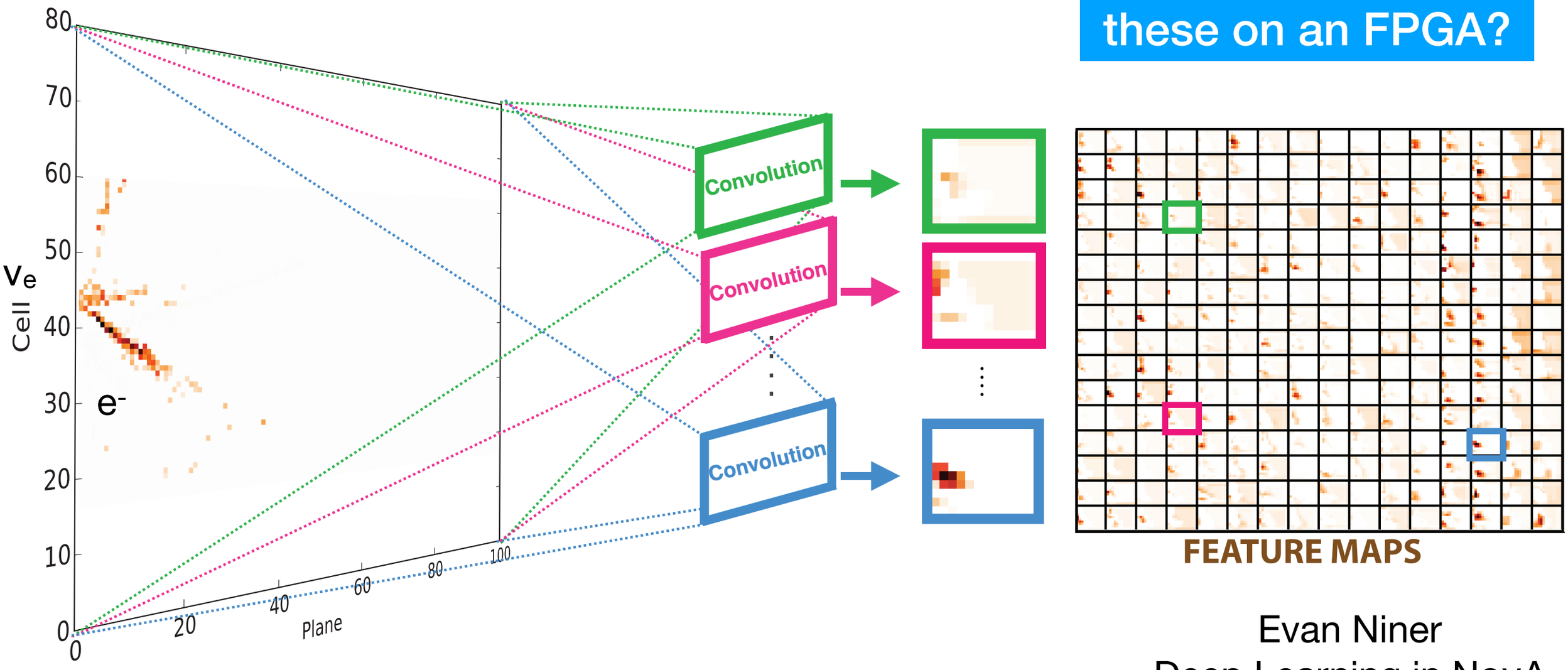
Szegedy et al.
arXiv:1409.4842



CNNs in Neutrino Experiments

- Readout detector as a (multidimensional) image
- Shown to be effective at classifying NovA neutrino events
- Adapted from GoogLeNet

Can we implement these on an FPGA?



Evan Niner
Deep Learning in NovA
arXiv:1604.01444

HEP Latency Landscape

Tools to infer big networks:
- Use BRAM to store weights
- Increase reuse factor ×1000
- Additional compression: weight sharing, Huffman coding
- Multi-pumping DSPs

Leads to O(ms) latencies

CMS: L1 Trigger HLT Offline

1 ns 1 us 1 ms 1 s

DUNE: DAQ Offline

Pure FPGAs

HEP Latency Landscape

Tools to infer big networks:
- Use BRAM to store weights
- Increase reuse factor ×1000
- Additional compression: weight sharing, Huffman coding
- Multi-pumping DSPs

Leads to O(ms) latencies

Natural application:
accelerate offline workflows

CMS: L1 Trigger — HLT — Offline

1 ns — 1 us — 1 ms — 1 s

DUNE: DAQ — Offline

Pure FPGAs

Acceleration with FPGAs

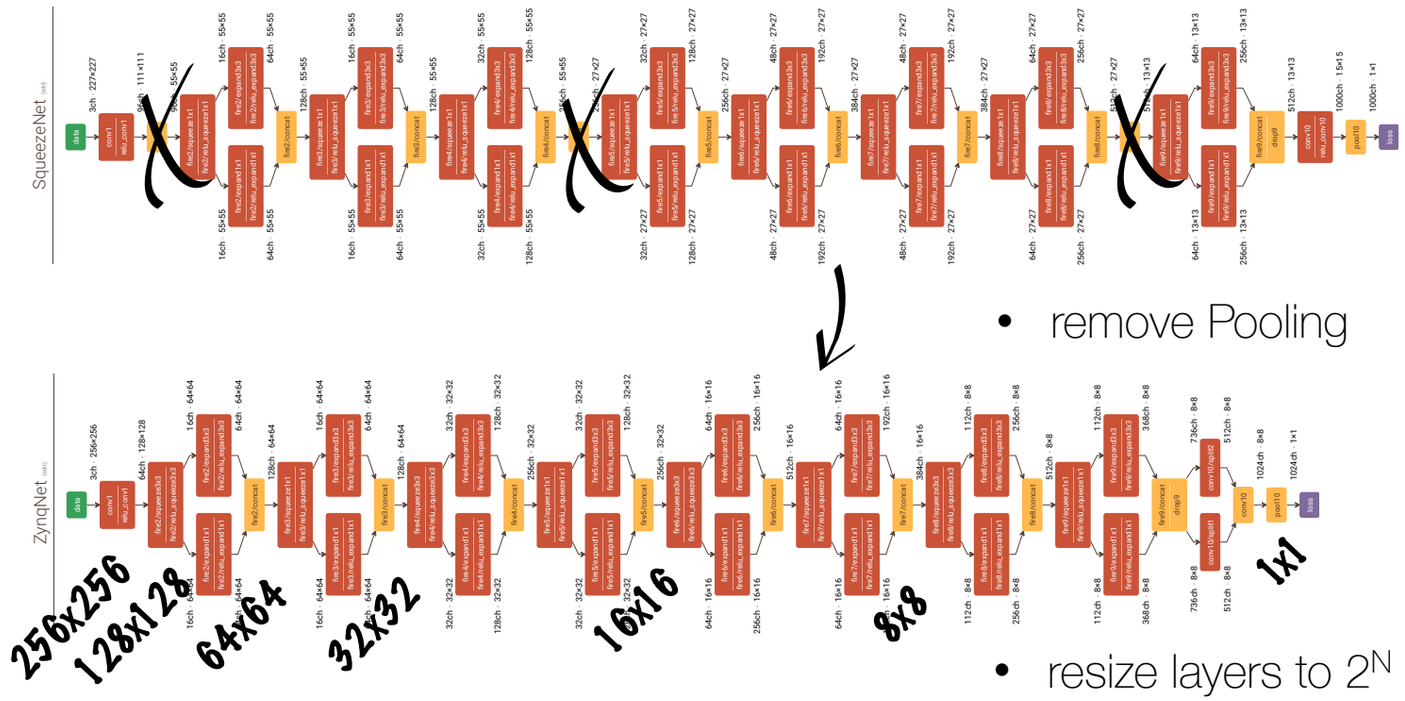
SqueezeNet

Han et al. 2016
arXiv:1602.07360
Gschwend 2016
ZynqNet

- 6-bit SqueezeNet smaller than 32-bit AlexNet by a factor of 500 and achieves the same accuracy (Han et al. 2016)
- Fits on one FPGA with on board memory (Gschwend 2016)
- Others have also demonstrated CNNs on FPGAs

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	**510x**	57.5%	80.3%

Additional optimization for FPGA



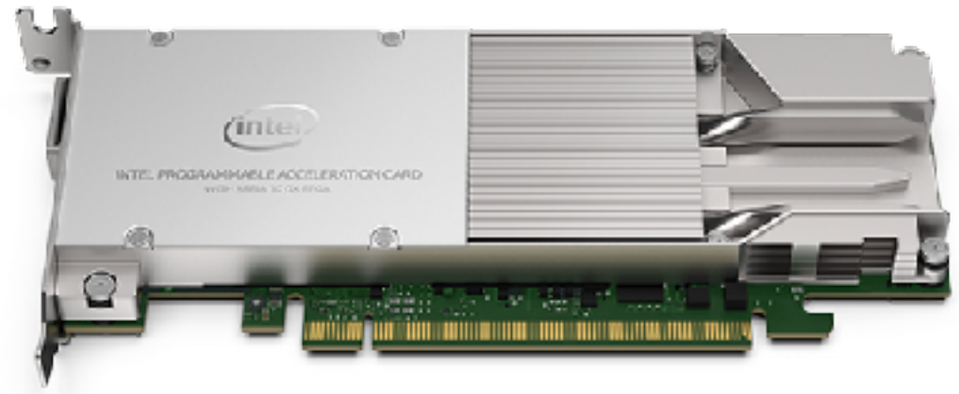
- remove Pooling
- resize layers to 2^N

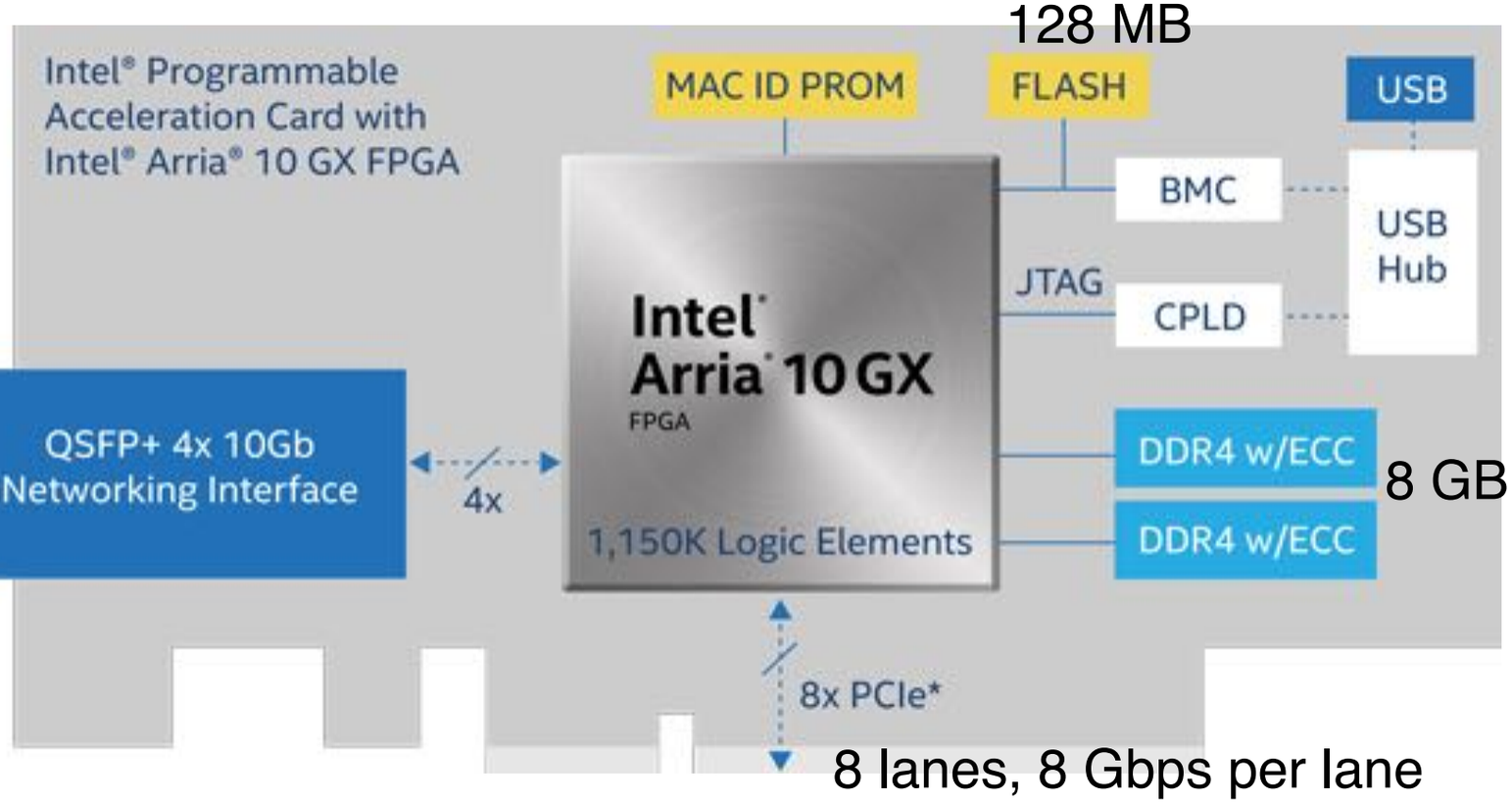
FPGA resources

resource	Block RAM	DSP Slices	FF	LUT
used	996	739	137k	154k
available	1090	900	437k	218k
utilization	91 %	82 %	31 %	70 %

FPGA Co-Processor Acceleration Card

Leverage recent advances/trends in industry





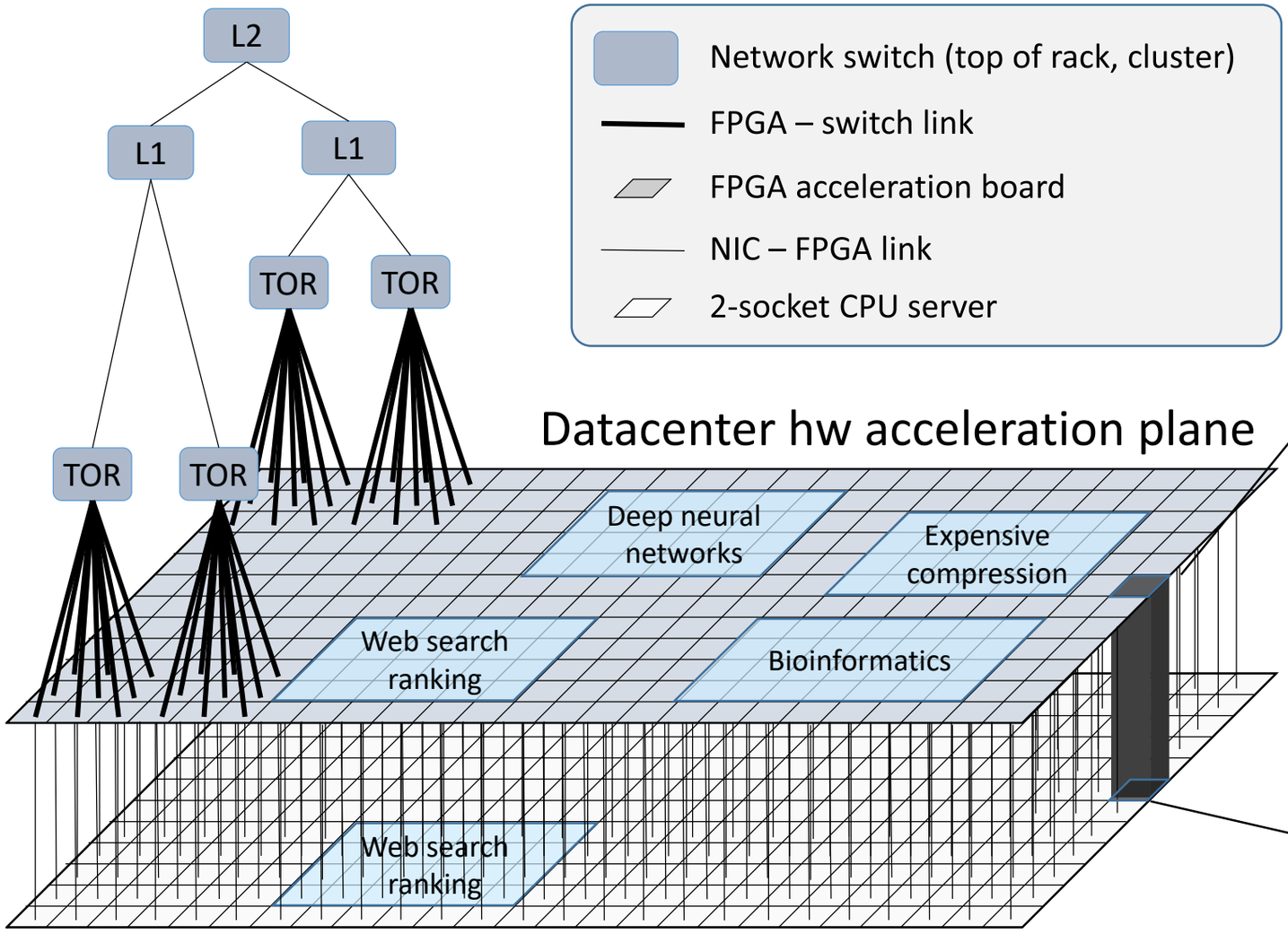
Targeted Workloads

- Big data analytics
- Artificial intelligence
- Video transcoding
- Cyber security
- High-performance computing
- Financial technology, or FinTech

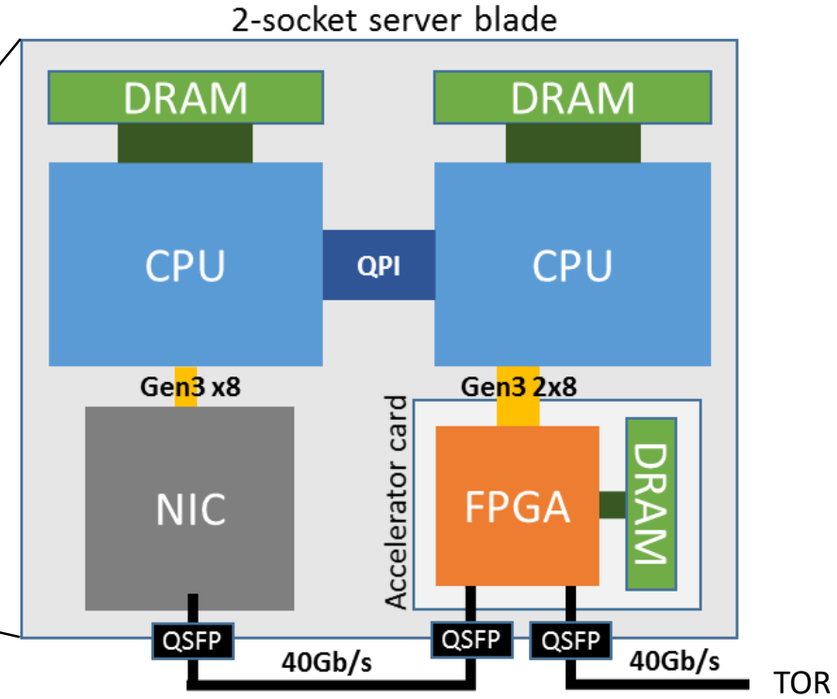
Cloud Scale

Machine Learning Forum
Andrew Putnam
(Microsoft Research)
May 14 @ 1pm

>100 k FPGAs can communicate to each other



Microsoft Catapult



A. Caulfield, et al., Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (2016).
https://www.microsoft.com/en-us/research/publication/configurable-cloud-acceleration/

Summary and Outlook

- We introduce an HLS-based software/firmware compiler for ultra low-latency applications
  - Case study with jet substructure in Level-1 trigger
  - Tunable configuration for a broad range of use cases
  - Upcoming features:
    - More network architectures (CNN, RNN, etc.)
    - Support for Altera/Intel Quartus HLS
- FPGAs (with support from industry/cloud computing) may help to accelerate HEP computing workflows
- **Exciting times** ahead at the intersection of machine learning, custom hardware, and high energy physics

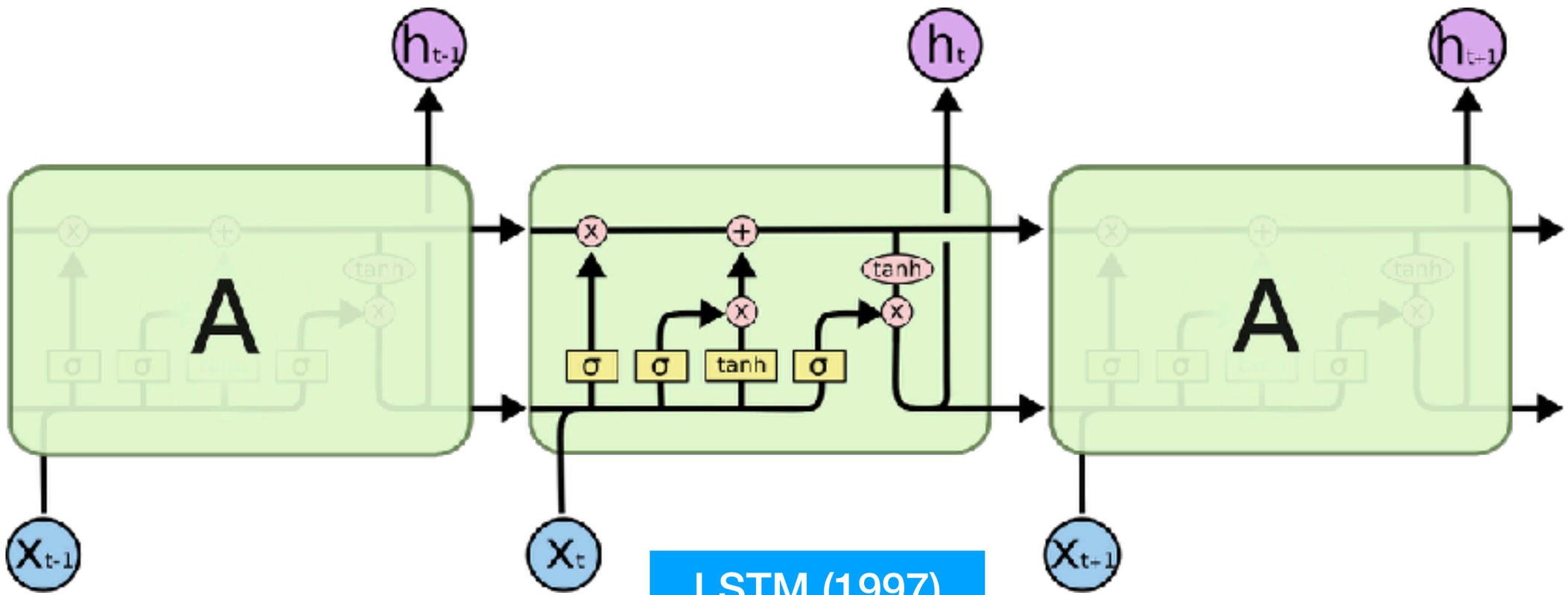
Backup

CNNs on FPGAs

- NIPS 2017 Demo: https://docs.google.com/presentation/d/1mTqsm5TronnB8MFD6yyq3CSPCV4bfck_aQ-ericM9cQ/edit#slide=id.p3
- Snowflake: arXiv:1708.02579
- DNNWeaver: http://act-lab.org/artifacts/dnnweaver/
- fpgaConvNet: http://cas.ee.ic.ac.uk/people/sv1310/fpgaConvNet.html
- Caffeine

Recurrent Neural Network

- Main task is language processing, time sequence prediction
- **LSTM layers** allow learned information to persist; network can learn long-term dependences in sequences



LSTM (1997)

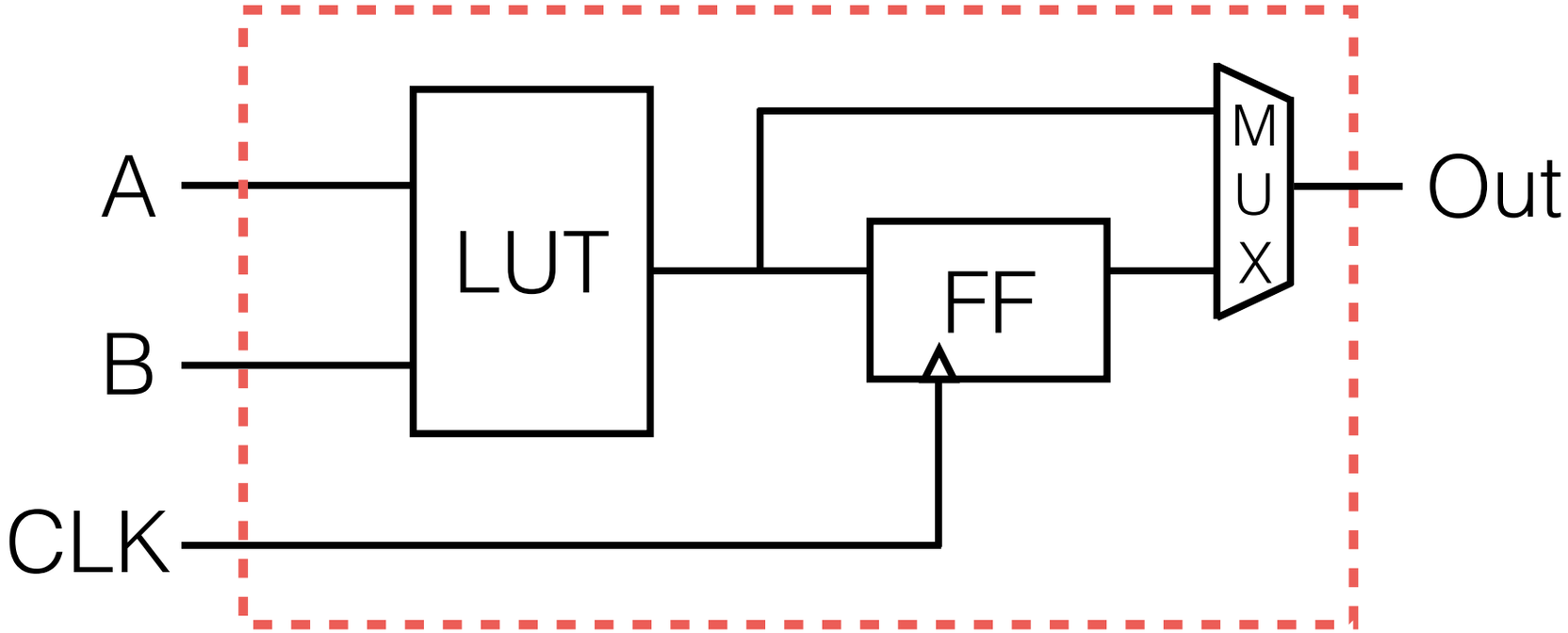
HLS Study Details

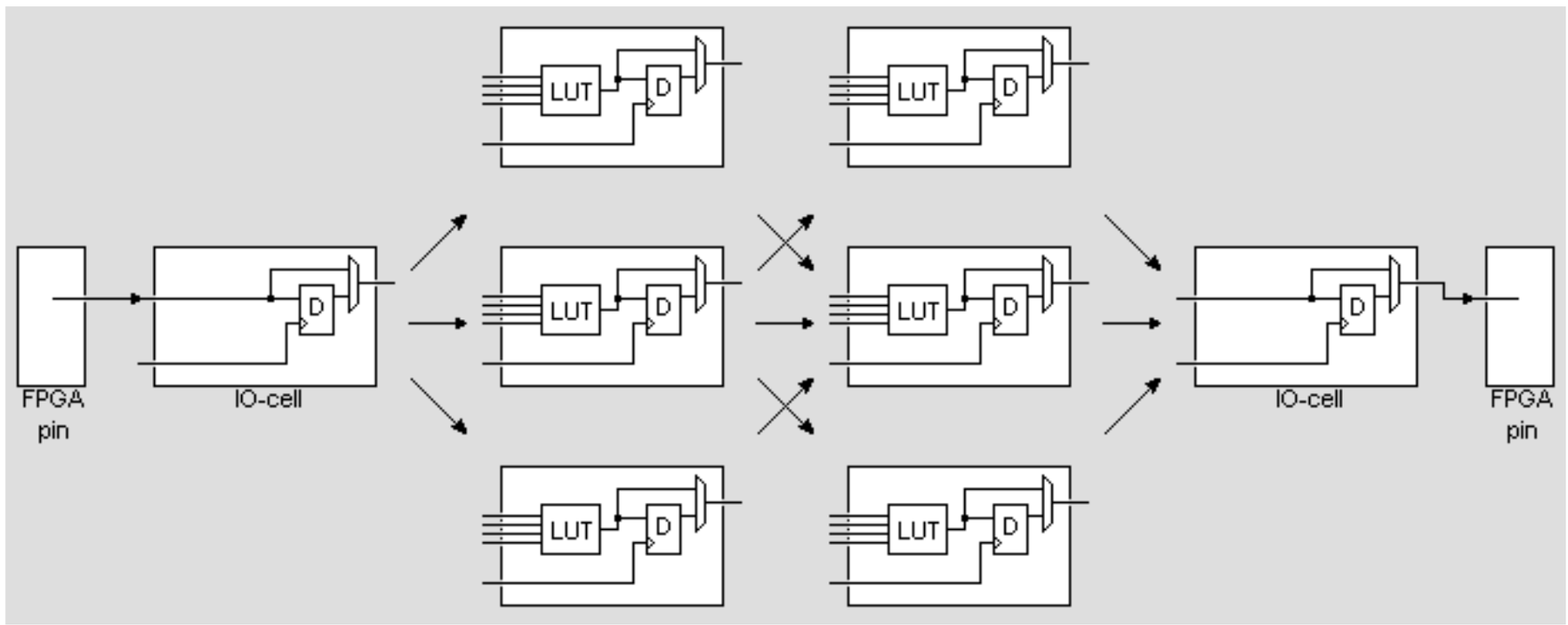
- Xilinx Vivado 2017.2
- Clock frequency: 200 MHz
- FPGA: Xilinx Kintex Ultrascale (XCKU115-FLVB2104)

- A note on inputs:
  - We assume network inputs have already been computed
    - Not a good assumption in the jet substructure case with “expert” features
    - Convolutional and recurrent networks which more naturally operate on “raw” features are in development

- Note: resource usage comes from HLS estimates
  - Discussion on differences w.r.t. implementation later

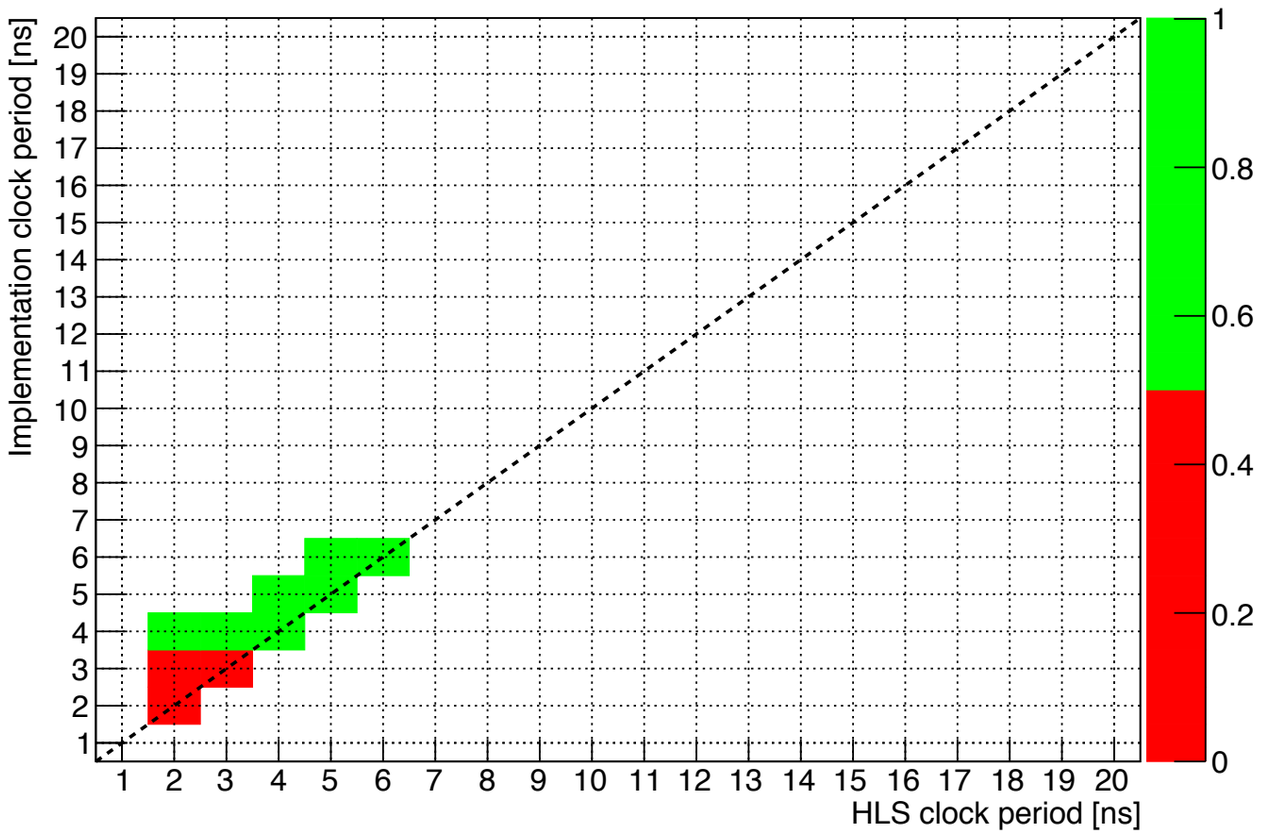
Logic Cell



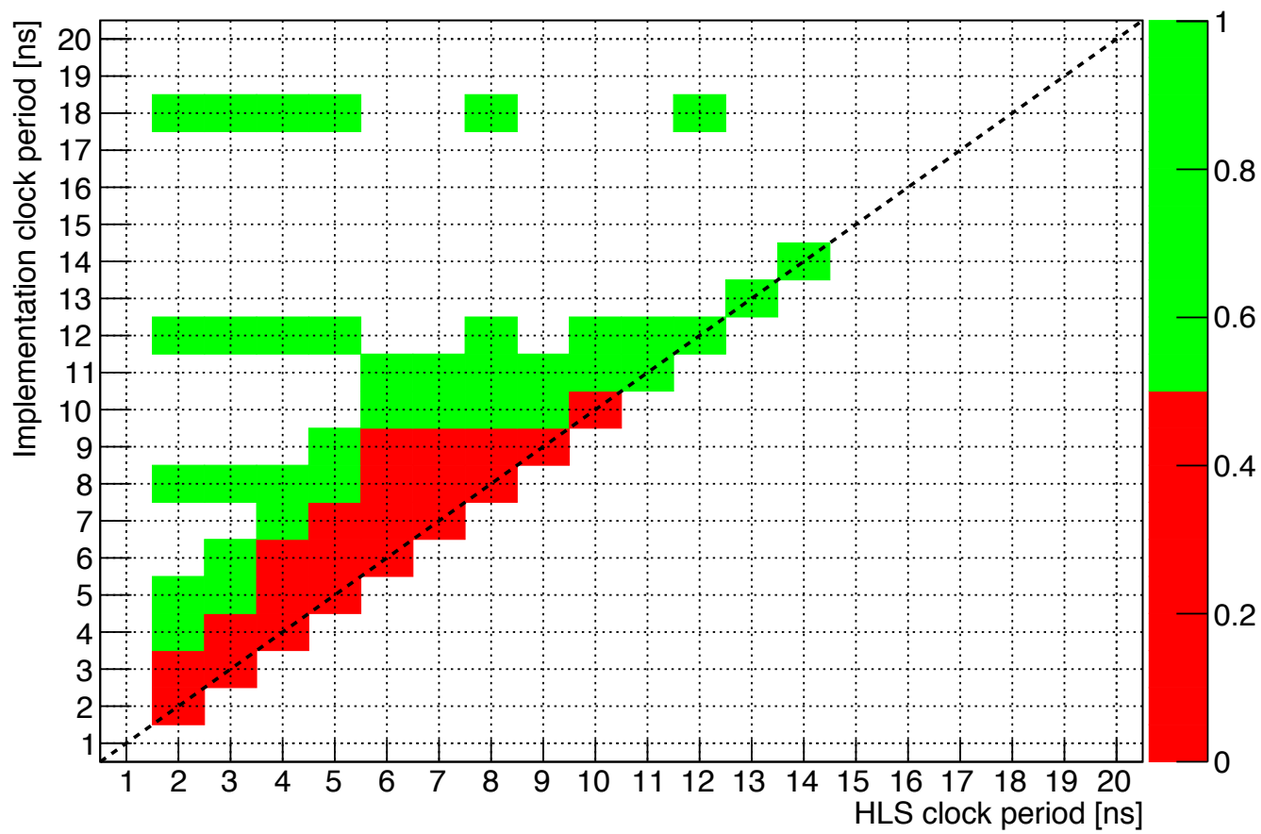


Meeting HLS Target Timing

v2016.4



v2017.2



- Implementation timing meets HLS target in v2017.2 for clock periods ≥ 11 ns
- Implementation timing meets HLS target in v2016.4 for clock periods ≥ 4 ns

Kintex® UltraScale™ FPGAs

	Device Name	KU025[1]	KU035	KU040	KU060	KU085	KU095	KU115
Logic Resources	System Logic Cells (K)	318	444	530	726	1,088	1,176	1,451
	CLB Flip-Flops	290,880	406,256	484,800	663,360	995,040	1,075,200	1,326,720
	CLB LUTs	145,440	203,128	242,400	331,680	497,520	537,600	663,360
Memory Resources	Maximum Distributed RAM (Kb)	4,230	5,908	7,050	9,180	13,770	4,800	18,360
	Block RAM/FIFO w/ECC (36Kb each)	360	540	600	1,080	1,620	1,680	2,160
	Block RAM/FIFO (18Kb each)	720	1,080	1,200	2,160	3,240	3,360	4,320
	Total Block RAM (Mb)	12.7	19.0	21.1	38.0	56.9	59.1	75.9
Clock Resources	CMT (1 MMCM, 2 PLLs)	6	10	10	12	22	16	24
	I/O DLL	24	40	40	48	56	64	64
I/O Resources	Maximum Single-Ended HP I/Os	208	416	416	520	572	650	676
	Maximum Differential HP I/O Pairs	96	192	192	240	264	288	312
	Maximum Single-Ended HR I/Os	104	104	104	104	104	52	156
	Maximum Differential HR I/O Pairs	48	48	48	48	56	24	72
Integrated IP Resources	DSP Slices	1,152	1,700	1,920	2,760	4,100	768	5,520
	System Monitor	1	1	1	1	2	1	2
	PCIe® Gen1/2/3	1	2	3	3	4	4	6
	Interlaken	0	0	0	0	0	2	0
	100G Ethernet	0	0	0	0	0	2	0
	16.3Gb/s Transceivers (GTH/GTY)	12	16	20	32	56	64[2]	64
Speed Grades	Commercial	-1	-1	-1	-1	-1	-1	-1
	Extended	-2	-2 -3	-2 -3	-2 -3	-2 -3	-2	-2 -3
	Industrial	-1 -2	-1 -1L -2	-1 -1L -2	-1 -1L -2	-1 -1L -2	-1 -2	-1 -1L -2

	Package Footprint[3, 4, 5, 6]	Package Dimensions (mm)	HR I/O, HP I/O, GTH/GTY						
Footprint Compatible with Virtex® UltraScale Devices	A784[7]	23x23[8]		104, 364, 8	104, 364, 8				
	A676[7]	27x27		104, 208, 16	104, 208, 16				
	A900[7]	31x31		104, 364, 16	104, 364, 16				
	A1156	35x35	104, 208, 12	104, 416, 16	104, 416, 20	104, 416, 28		52, 468, 28	
	A1517	40x40				104, 520, 32	104, 520, 48		104, 520, 48
	C1517	40x40						52, 468, 40	
	D1517	40x40							104, 234, 64
	B1760	42.5x42.5					104, 572, 44	52, 650, 48	104, 598, 52
	A2104	47.5x47.5							156, 676, 52
	B2104	47.5x47.5						52, 650, 64	104, 598, 64
	D1924	45x45							156, 676, 52
	F1924	45x45					104, 520, 56		104, 624, 64

Notes:
1. Certain advanced configuration features are not supported in the KU025. Refer to the Configuring FPGAs section in DS890, *UltraScale Architecture and Product* Overview.
2. GTY transceivers in KU095 devices support data rates up to 16.3Gb/s.
3. Packages with the same package footprint designator, e.g., A2104, are footprint compatible with all other UltraScale devices with the same sequence. See the migration table for details on inter-family migration.
4. Maximum achievable performance is device and package dependent; consult the associated data sheet for details.
5. For full part number details, see the Ordering Information section in DS890, *UltraScale Architecture and Product* Overview.
6. See UG575, *UltraScale Architecture Packaging and Pinouts User Guide* for more information.
7. GTH transceivers in A784, A676, and A900 packages support data rates up to 12.5Gb/s.
8. 0.8mm ball pitch. All other packages listed 1mm ball pitch.