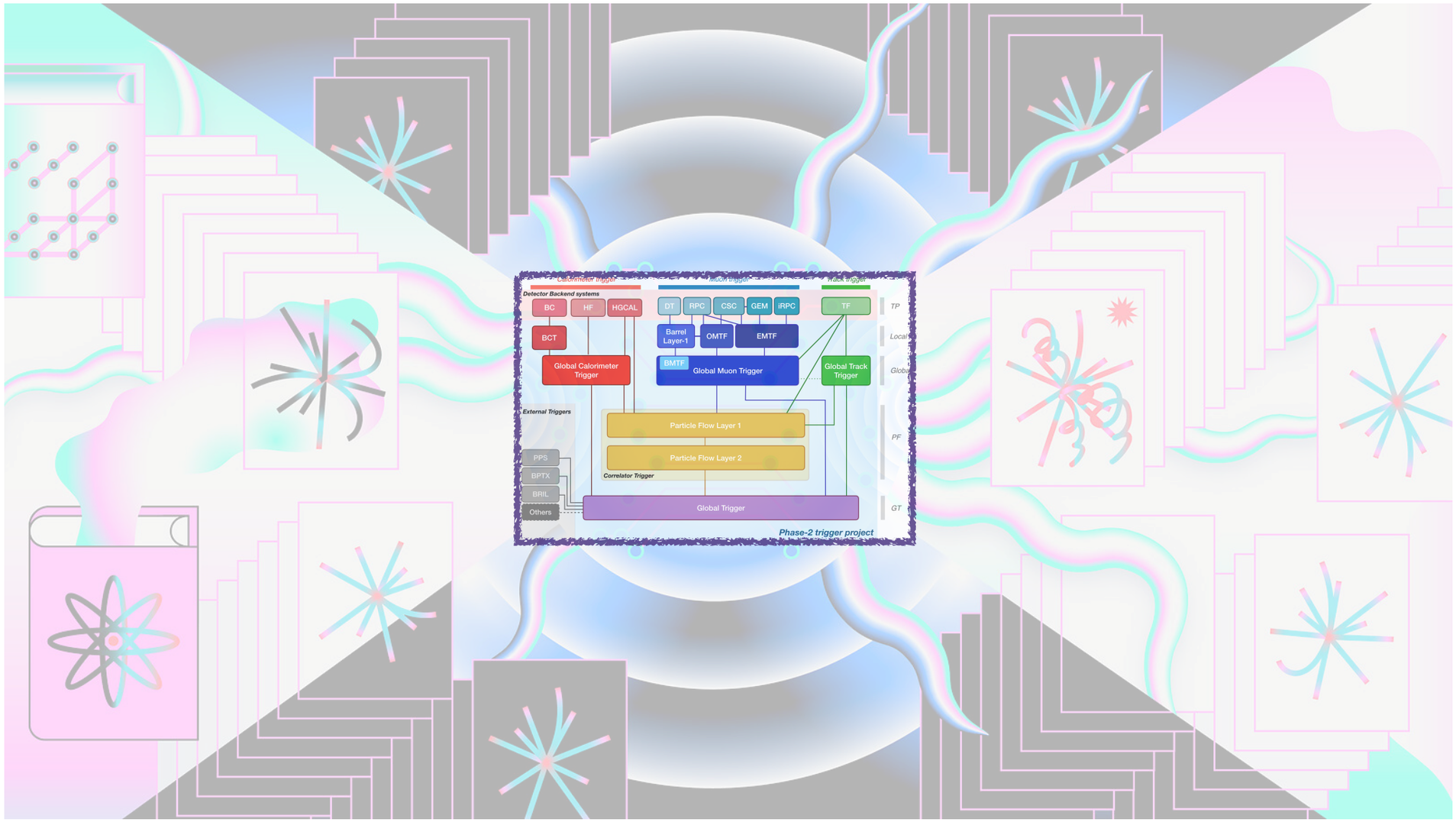
Implementing deep neural networks in CMS Level 1 Trigger



Duc Hoang, Aidan Chambers, Dylan Rankin, Philip Harris (MIT) on behalf of the CMS L1 Trigger Group

With significant help from Christian Herwig, Sergo Jindariani (Fermilab), Sioni Summers (CERN), Javier Duarte (UCSD).

Artwork

CMS TDR

Outline

1. Overview of the CMS L1 Trigger

2. Hls4ml workflow

3. Integration of b-tagging neural networks into CMS trigger.

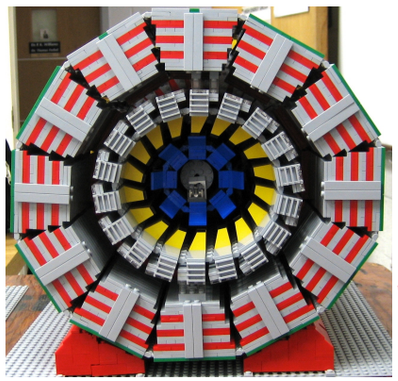
GIF Credit

At the LHC, there are 40 million proton-proton collisions per second.

$\approx \mathcal{O}(100)$ Tbs/s

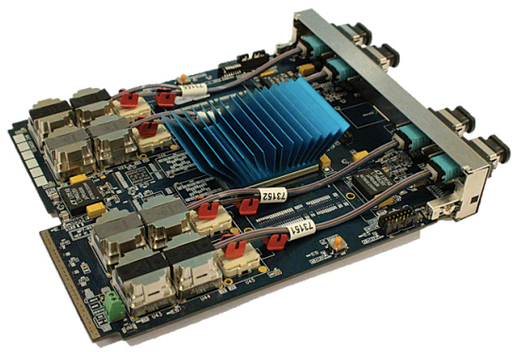
CMS Data Flow

Radiation hard ASICs



320 Tb/s

FPGA boards

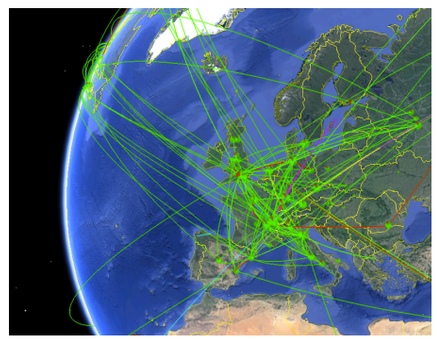


1 tb/s

Local CPU cluster

10 Gb/s

CPU grid



FAST

- 40 MHz collisions
- 12.5 μs window
- L1 Trigger

Intermediate

- 100 kHz collisions
- < 500 ms window
- High Level Trigger

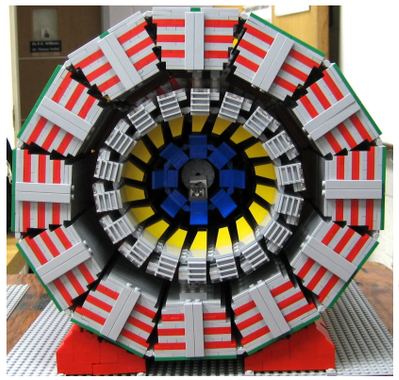
Slow

- 1kHz collisions
- 10s window
- Offline Cluster

Adapted from Phil Harris's IAIFI Colloquium slides

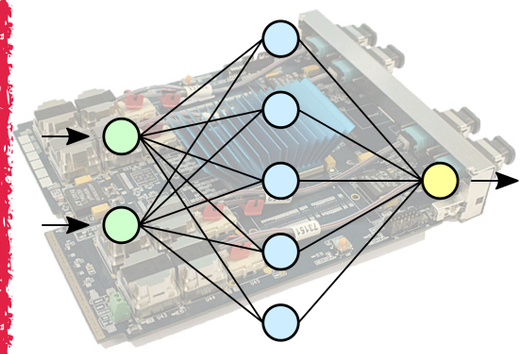
Focus of this talk

Radiation hard ASICs



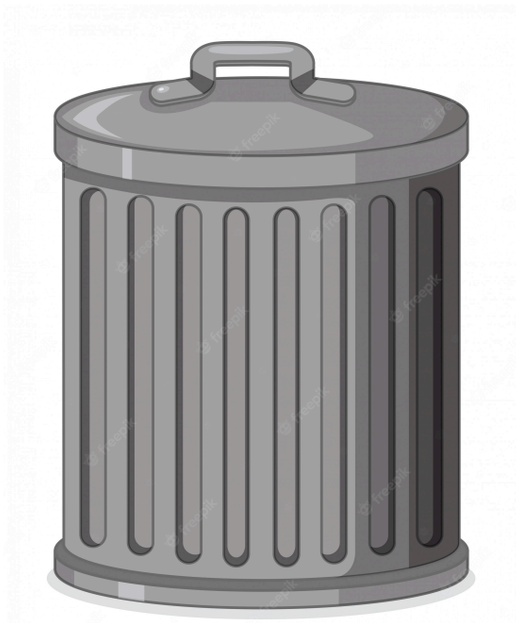
320 Tb/s

FPGA boards + Neural network



FAST

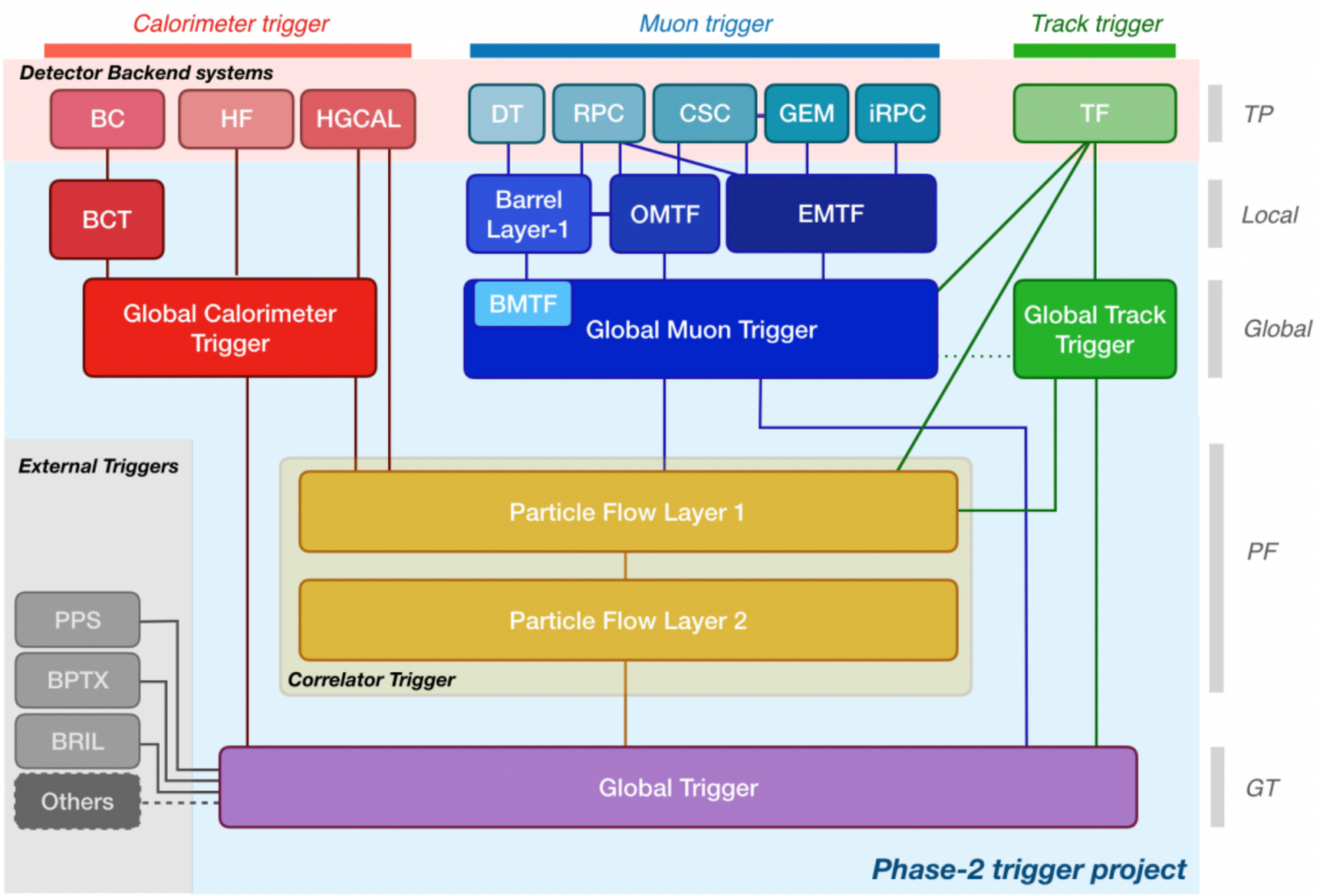
- 40 MHz collisions
- 12.5 μs window
- L1 Trigger



Select 1 event out of 400 events, the rest is thrown away forever!

Adapted from Phil Harris's IAIFI Colloquium slides

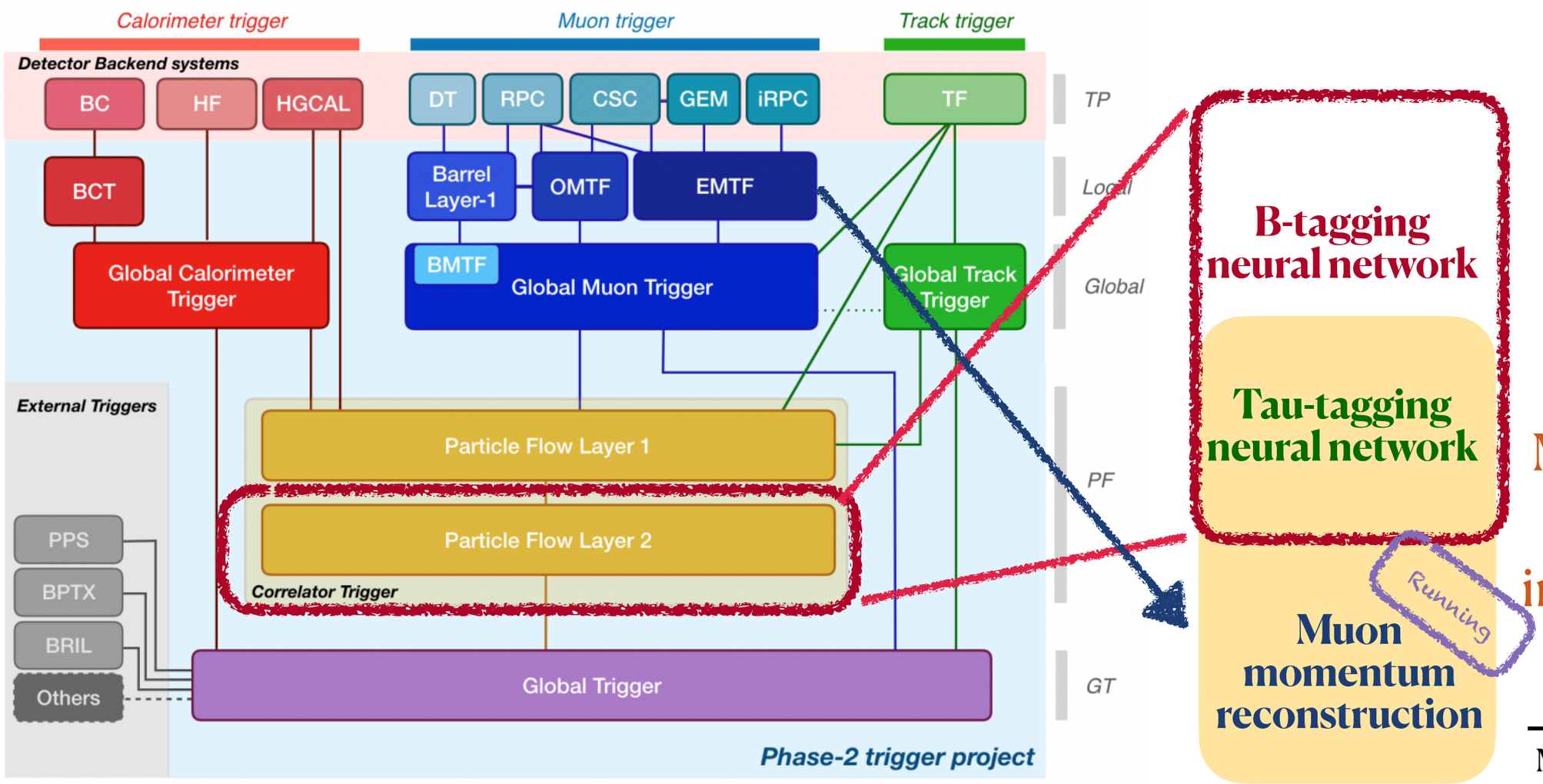
CMS L1 Phase-2 upgraded trigger design



Overall latency: 12.5 μs

CMS TDR

Where would the neural nets run?



CONV Architecture

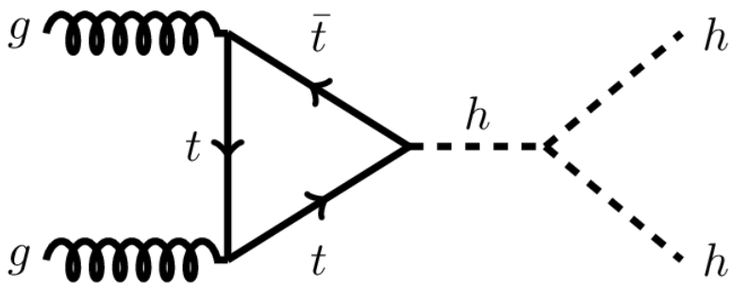
MLP architecture. Fully tested and integrated into the L1 trigger

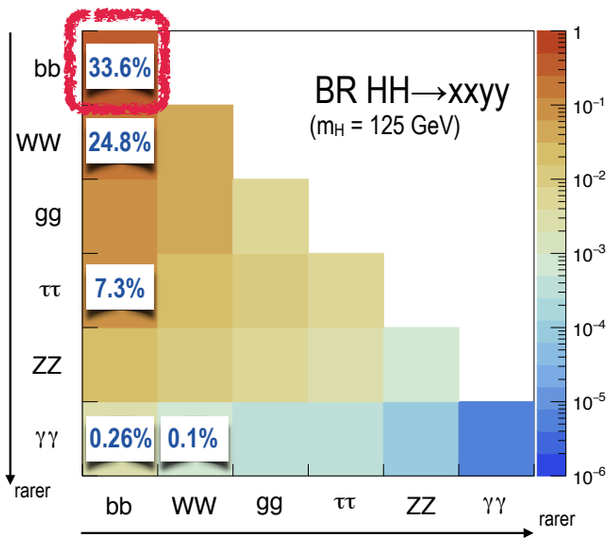
Muon is actually the current system.

CMS TDR

Btagging

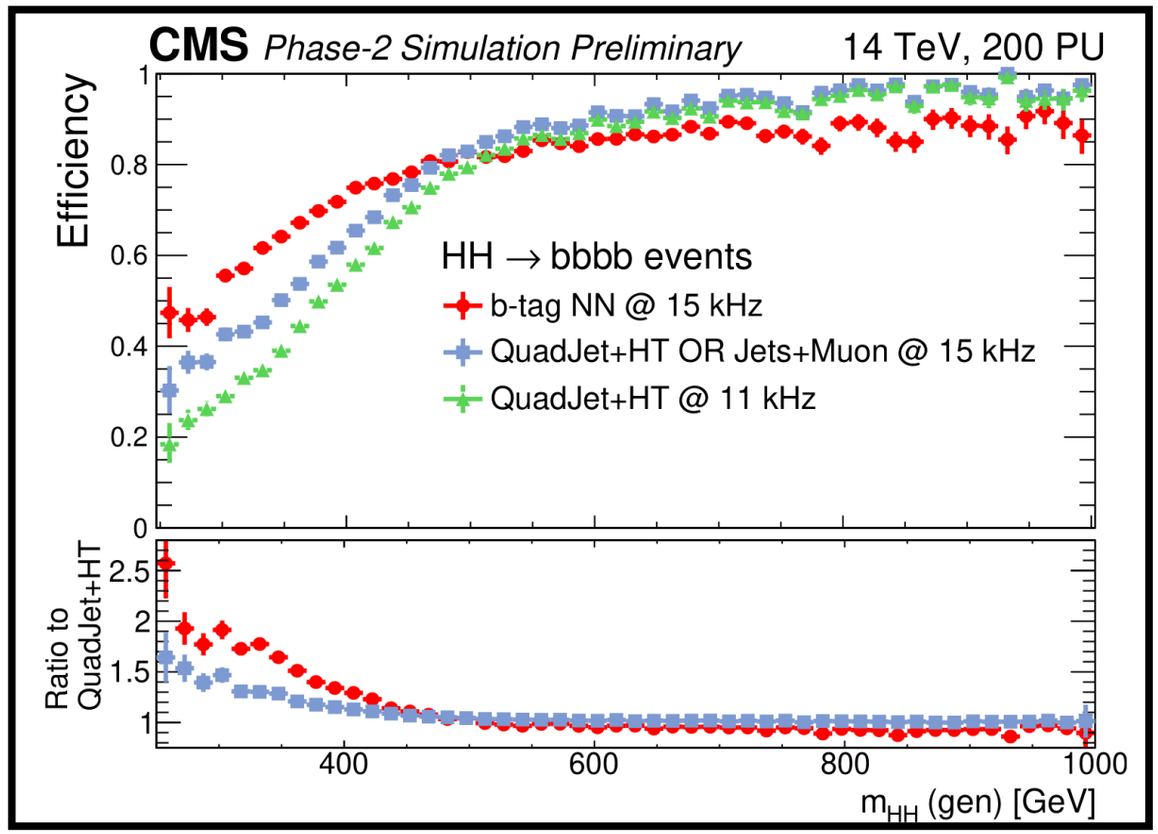
Physics study — Aidan Chambers





$HH \to b\bar{b}b\bar{b}$ has the largest branching ratio!

B-tagging in Level 1 trigger at CMS is of great physics interest



Analysis done by A. Chambers has shown significant improvement compared to previously used algorithms.

Btagging model architecture

Each particle is a 64bit word

0x0123456789abcdef

Each letter is a 4 bit number: 0b0000

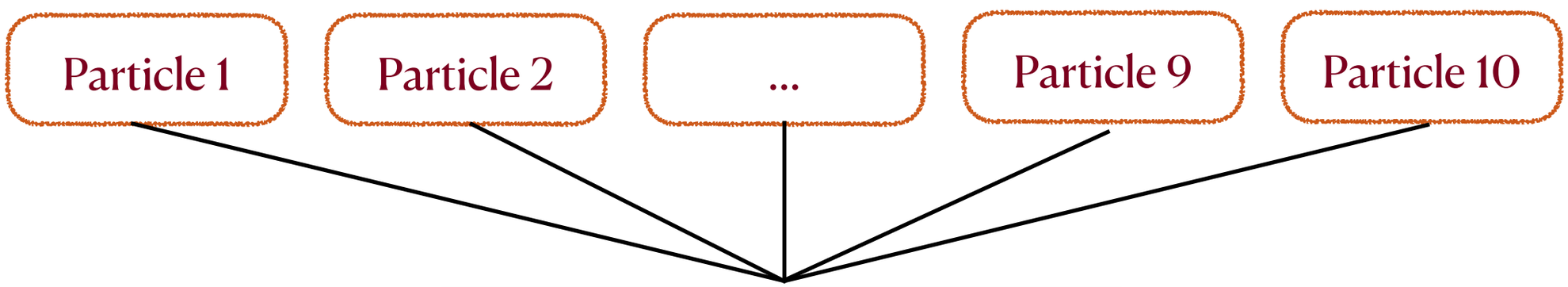
0x[Valid bit][other bits][Particle ID][phi][eta][pt]

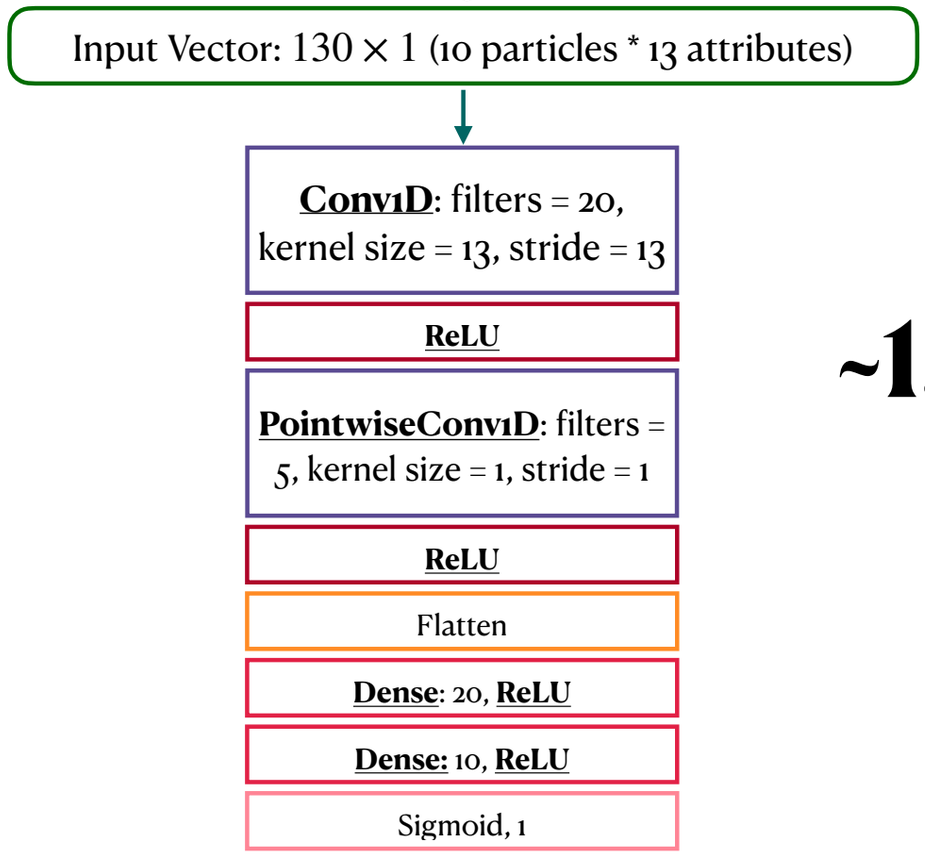
Most significant bit (MSB) used for validity

"Frame valdid"

UnInt<**3**> Int<**11**> int<**12**> ufixed<**14**,12>

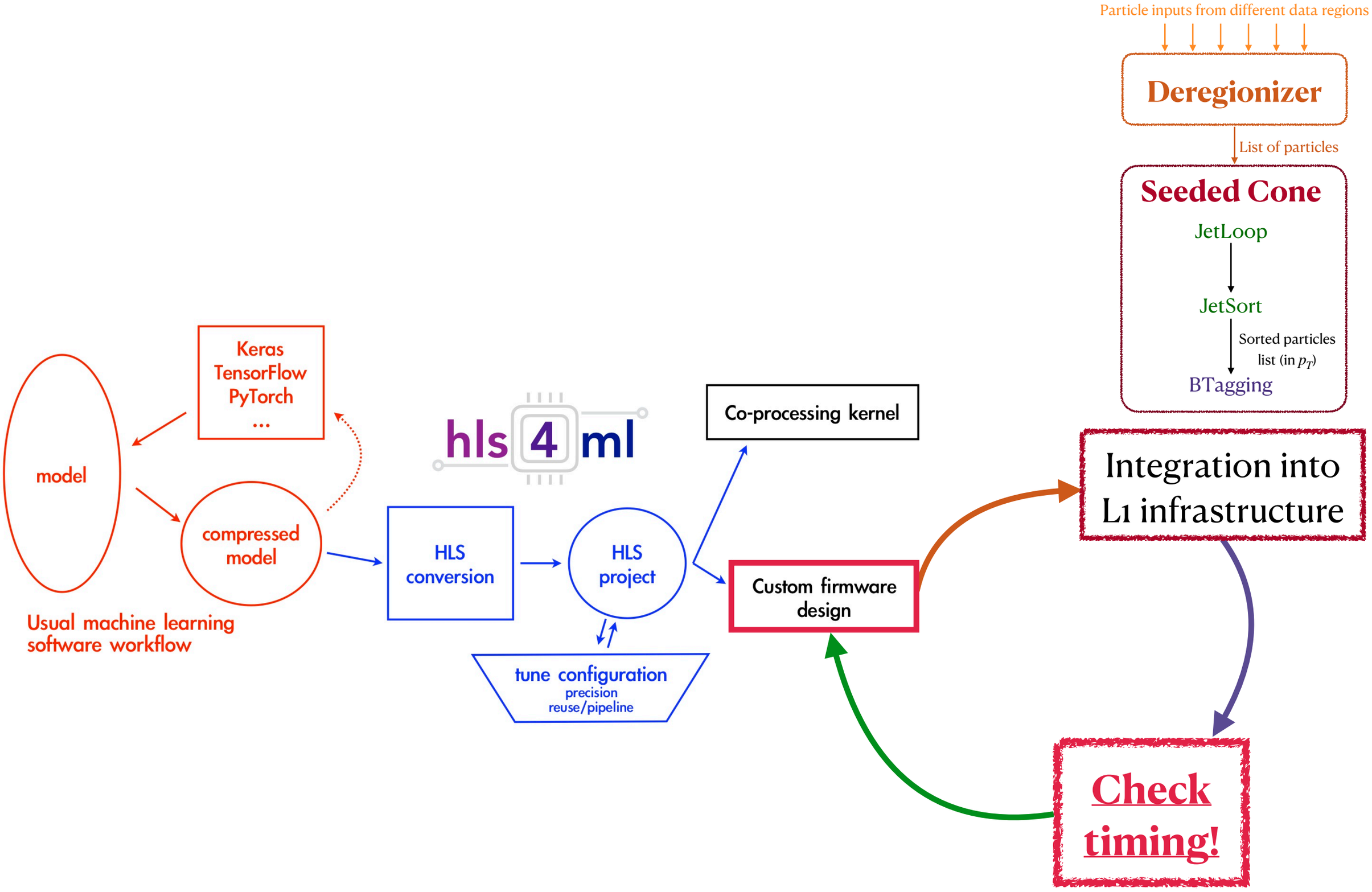
Data valid just checks for non-zero pt.





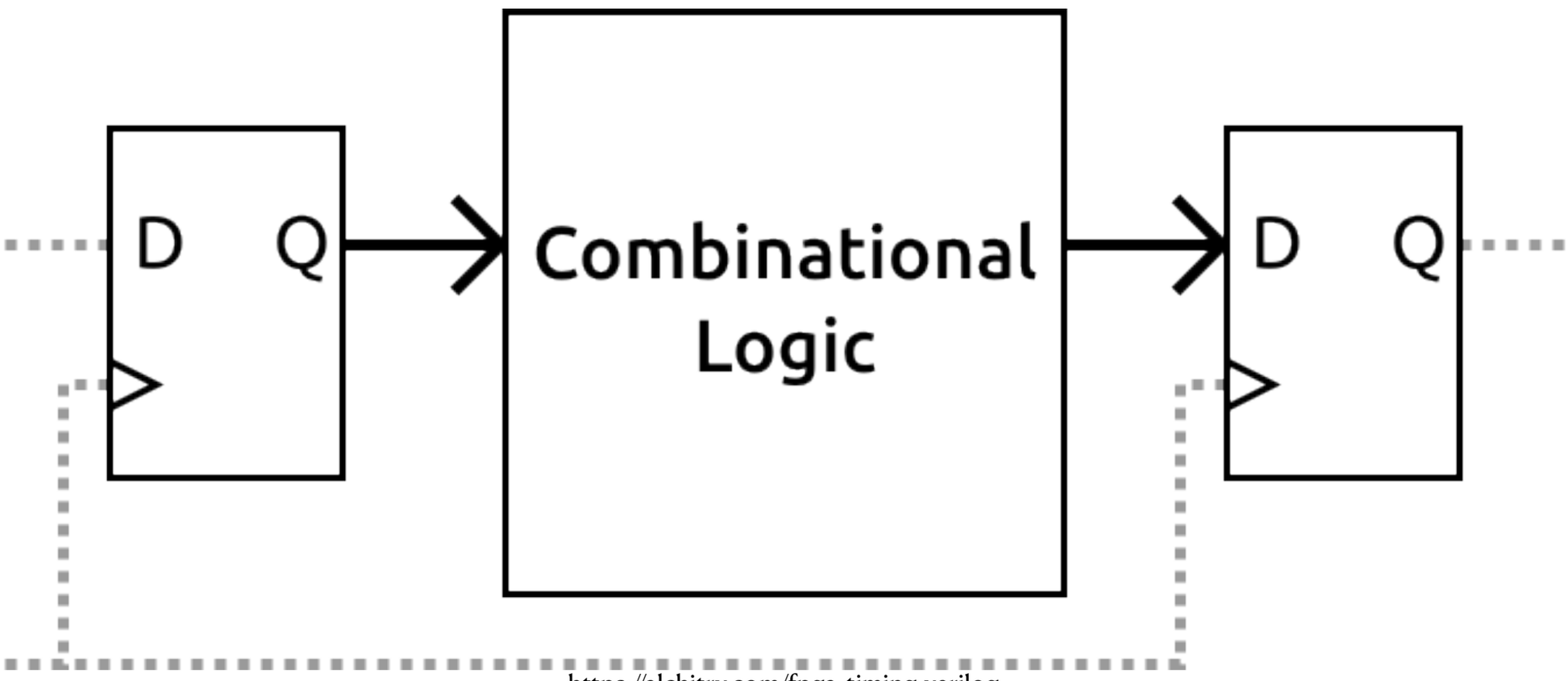
~1.6k parameters

Hls4ml (extended) workflow



FPGA Timing

Not often reflected in hls reports



https://alchitry.com/fpga-timing-verilog

Time takes from one flip-flop, through some combinational logic, to propagate to another flip-flop

hls4ml models would not always meet timing!

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	6	-
FIFO	-	-	-	-	-
Instance	1	1947	13212	144467	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	66	-
Register	-	-	10907	-	-
Total	1	1947	24119	144539	0
Available SLR	1440	2280	788160	394080	320
Utilization SLR (%)	~0	85	3	36	0
Available	4320	6840	2364480	1182240	960
Utilization (%)	~0	28	1	12	0

Latency: 6ons, II=1

Reasonable resource usage

This model has **worst negative slack (WNS) of *-5ns***

#1 trick: Convolution to fully-connected layers

Input Vector: 150×1 (10 particles * 13 attributes + **padding**)

Conv1D

10 copies

Dense resource 15×20 — Dense resource 15×20

Pointwise Conv1D

10 copies

Dense resource 20×5 — Dense resource 20×5

Dense ...

WNS: -5ns → -0.6ns

Oftentimes you need to customize your model

2 trick: Area contraint

No constraints

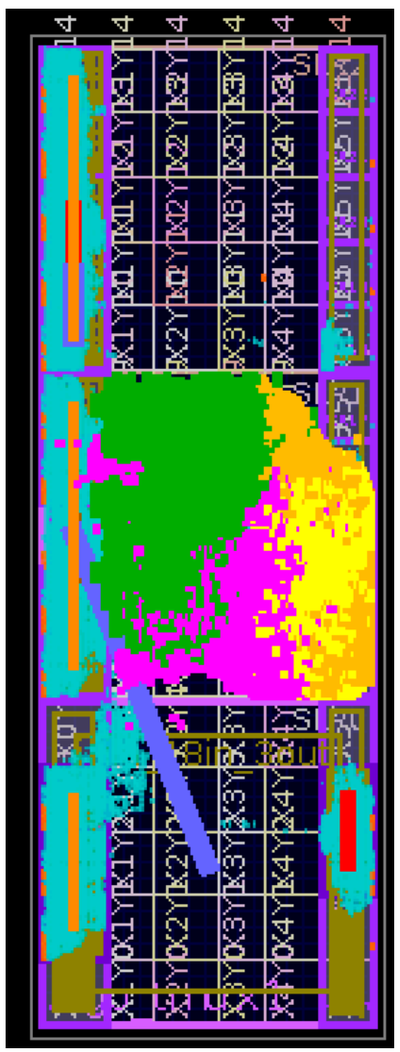
Move btag model to a separate SLR and closer to output port

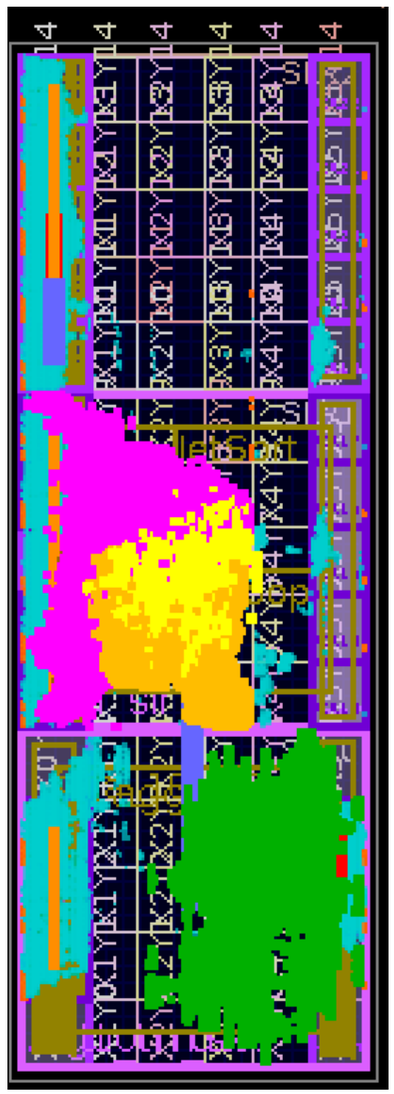
Deregionizer

JetLoop

Jet Sort

Btag NN





WNS: -0.6ns

WNS: -0.5ns

#3 Trick: be explicit!

For loop

```
DenseConv: for(int ii = 0; ii < CONFIG_T::out_width; ii++){

    data_T dense_data[CONFIG_T::filt_width];
    res_T  dense_res[CONFIG_T::n_filt];

    #pragma HLS ARRAY_PARTITION variable=dense_data complete dim=0
    #pragma HLS ARRAY_PARTITION variable=dense_res complete dim=0

    CopyDenseData: for(int i=0; i < CONFIG_T::filt_width; i++){
        dense_data[i] = data[ii*CONFIG_T::filt_width + i];
    }

    // Fill dense data
    dense_latency<data_T, res_T, CONFIG_T>(dense_data, dense_res, weights, biases);

    //Copy to res
    for(int jj=0; jj < CONFIG_T::n_filt; jj++){
        dense_res_all[ii * CONFIG_T::n_filt + jj] = dense_res[jj];
    }
```

WNS: -0.5ns

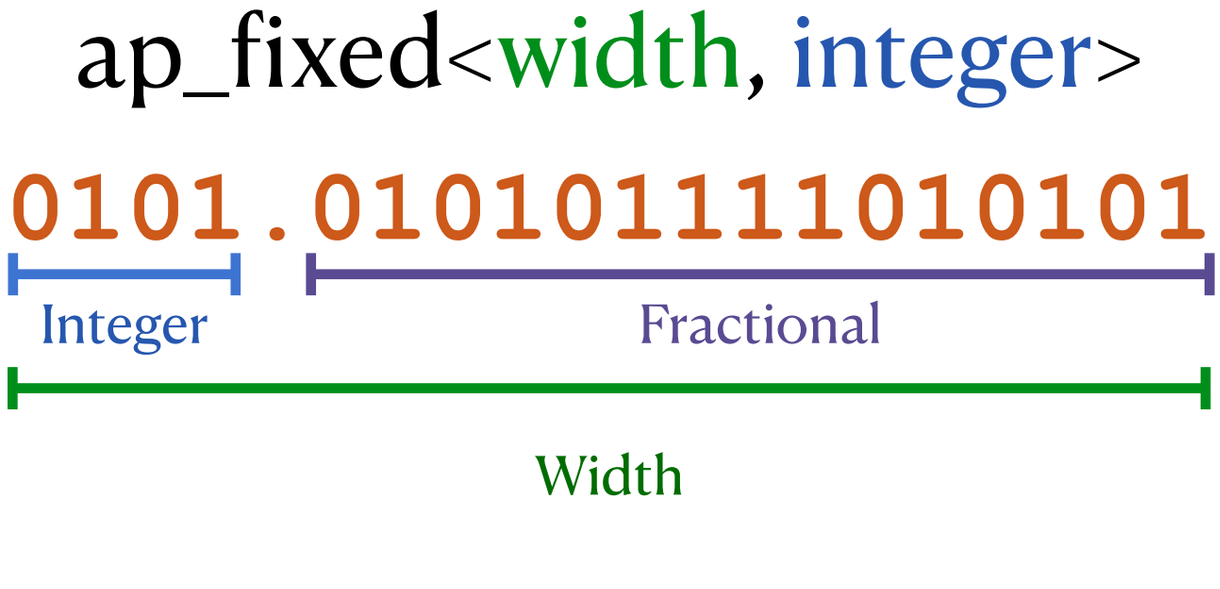
Write down every instantiations

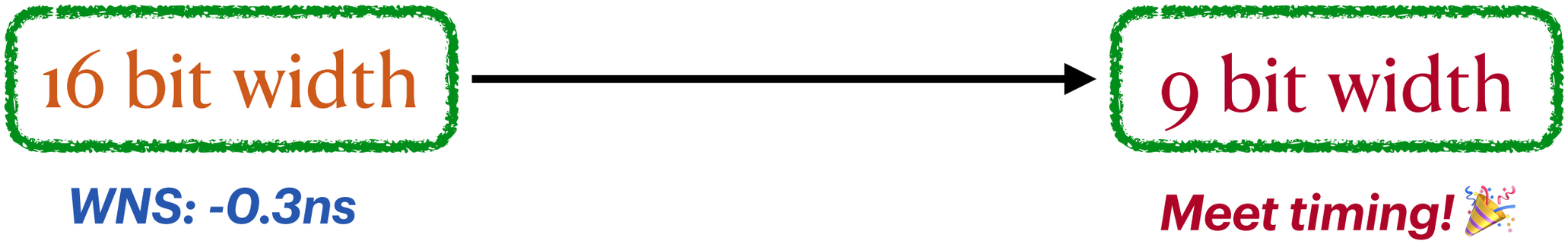
```
//Get res
dense_latency<data_T, res_T, CONFIG_T>(dense_data_1, dense_res_1, weights, biases);
dense_latency<data_T, res_T, CONFIG_T>(dense_data_2, dense_res_2, weights, biases);
dense_latency<data_T, res_T, CONFIG_T>(dense_data_3, dense_res_3, weights, biases);
dense_latency<data_T, res_T, CONFIG_T>(dense_data_4, dense_res_4, weights, biases);
dense_latency<data_T, res_T, CONFIG_T>(dense_data_5, dense_res_5, weights, biases);
dense_latency<data_T, res_T, CONFIG_T>(dense_data_6, dense_res_6, weights, biases);
dense_latency<data_T, res_T, CONFIG_T>(dense_data_7, dense_res_7, weights, biases);
dense_latency<data_T, res_T, CONFIG_T>(dense_data_8, dense_res_8, weights, biases);
dense_latency<data_T, res_T, CONFIG_T>(dense_data_9, dense_res_9, weights, biases);
dense_latency<data_T, res_T, CONFIG_T>(dense_data_10, dense_res_10, weights, biases);
```

WNS: -0.3ns

vivado_hls tends to like more explicit instructions

#4 trick: Quantization!

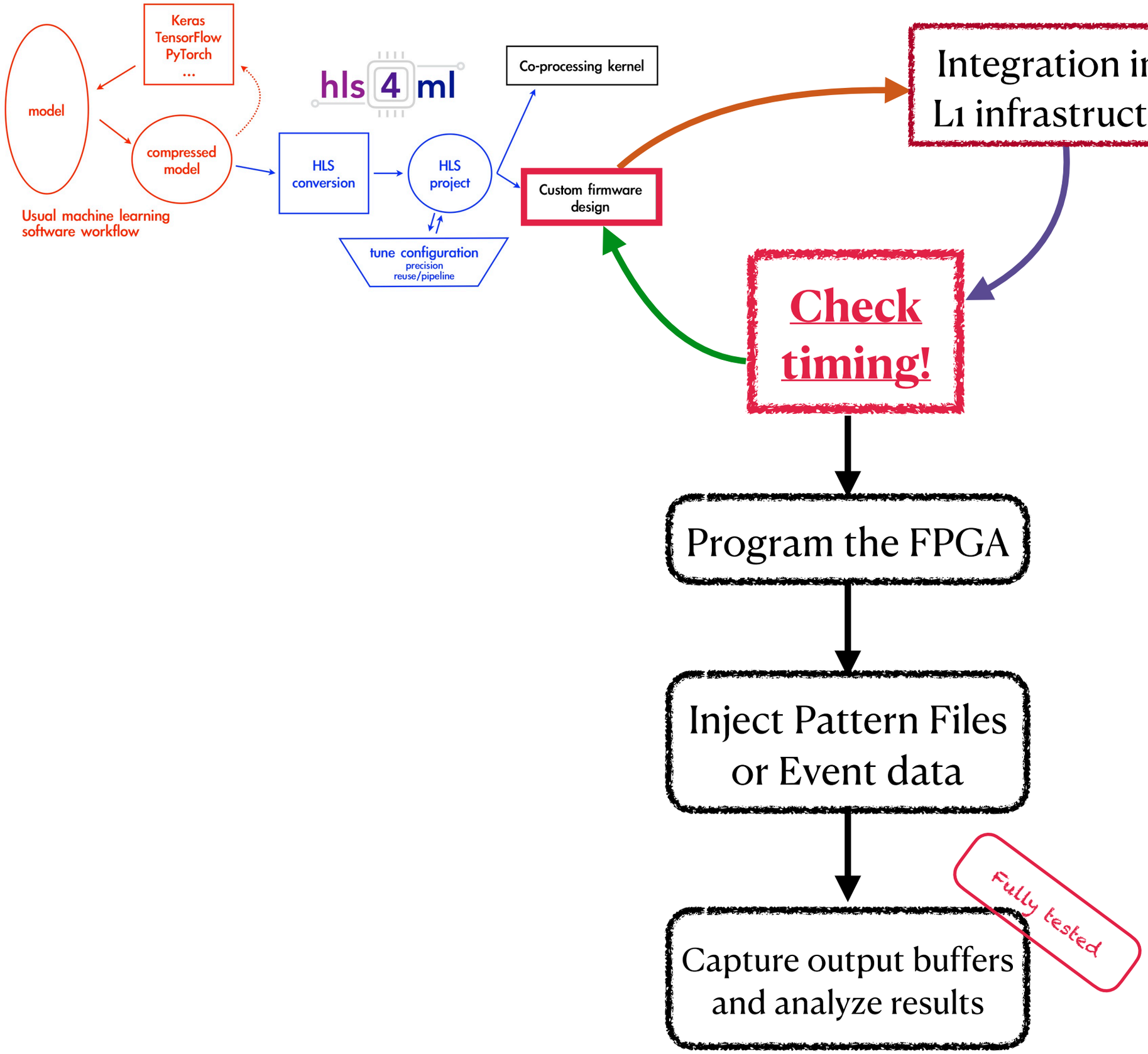




Vivado switches from **DSPs** to **LUTs** if the multiplication is lower than 9 bits. There are more **LUTs** available in the FPGA.

Illustration adapted from Javier Duarte's talk

Summary


Keras
TensorFlow
PyTorch
...
model
compressed model
Usual machine learning software workflow
hls 4 ml
HLS conversion
HLS project
tune configuration
precision
reuse/pipeline
Co-processing kernel
Custom firmware design
Integration into L1 infrastructure
Check timing!
Program the FPGA
Inject Pattern Files or Event data
Capture output buffers and analyze results
Fully tested

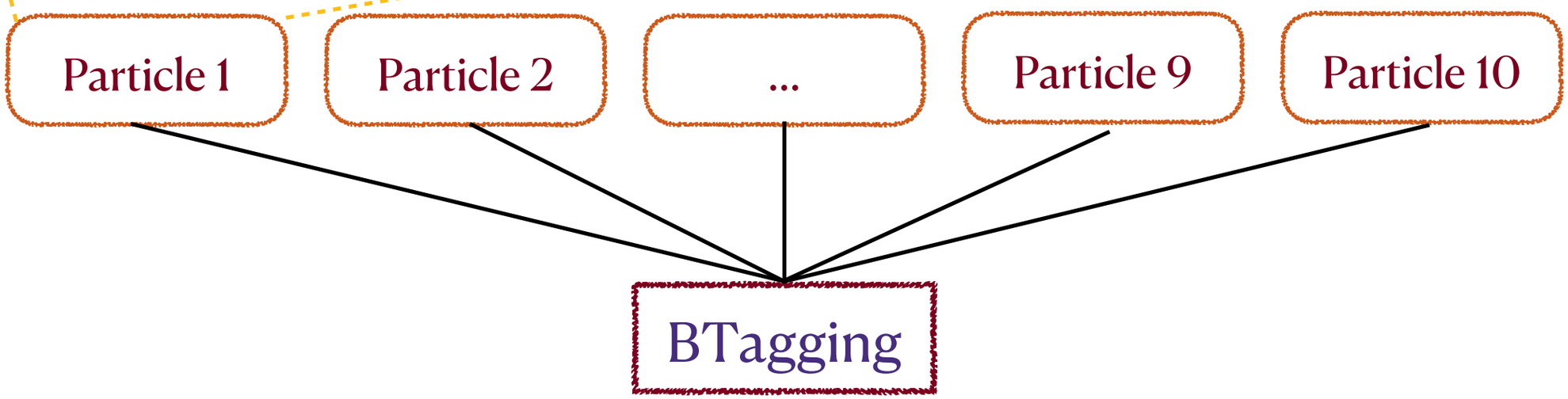
Back up

Btagging inputs

Set to 0 for neutral particles

Particle: 64 bit word

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Notation	$e-$	$e+$	$u-$	$u+$	γ	K_L	$\pi+$	$\pi-$	z_0	dxy	p_T	η	ϕ



Documentation of btagging model from his -> vhdl level

Reproducibility

- Btagging model training/hls synthesis script is here.
  - Training data: (currently on submit) at `/home/submit/aidandc/L1BTag/`
    - `trainingDataTT_PUP_Pad150.h5`
    - `testingDataTT_PUP_Pad150.h5`
    - `sampleDataTT_PUP_Pad150.h5`
    - `jetDataTT_PUP_Pad150.h5`
- Tested with a custom `hls4ml` implementation to make sure hls and python get the same results.
- **The “official" hls and vhdl implementation including preprocessing:**
- All are compiled with vivado v2019.2 on correlator2.