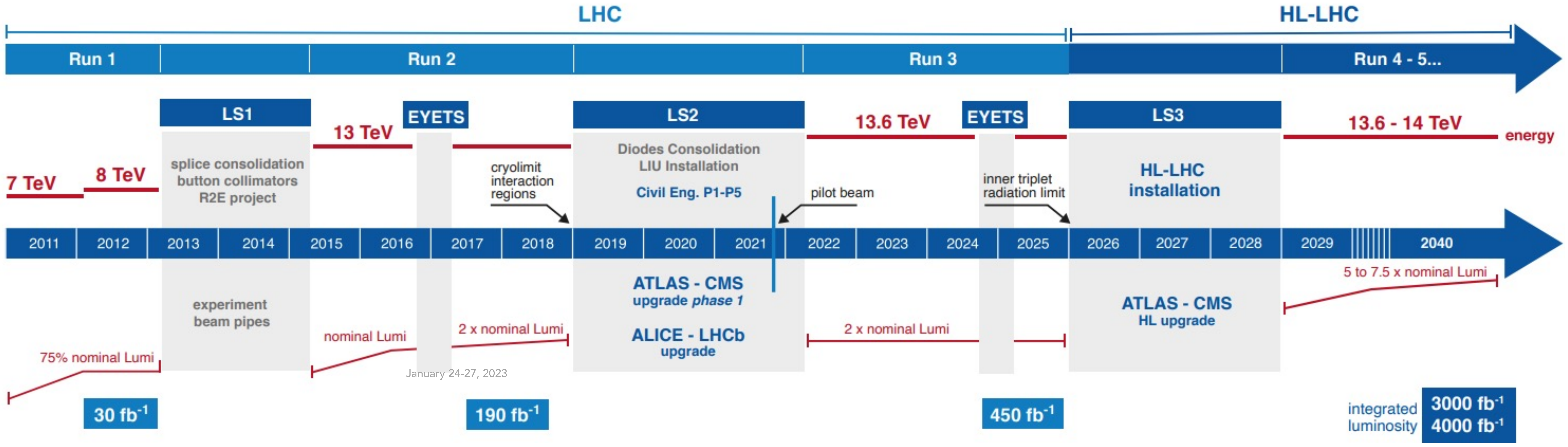
CMS Trigger Status

Jovan Mitrevski for CSAID CMS Developers

March 30, 2023

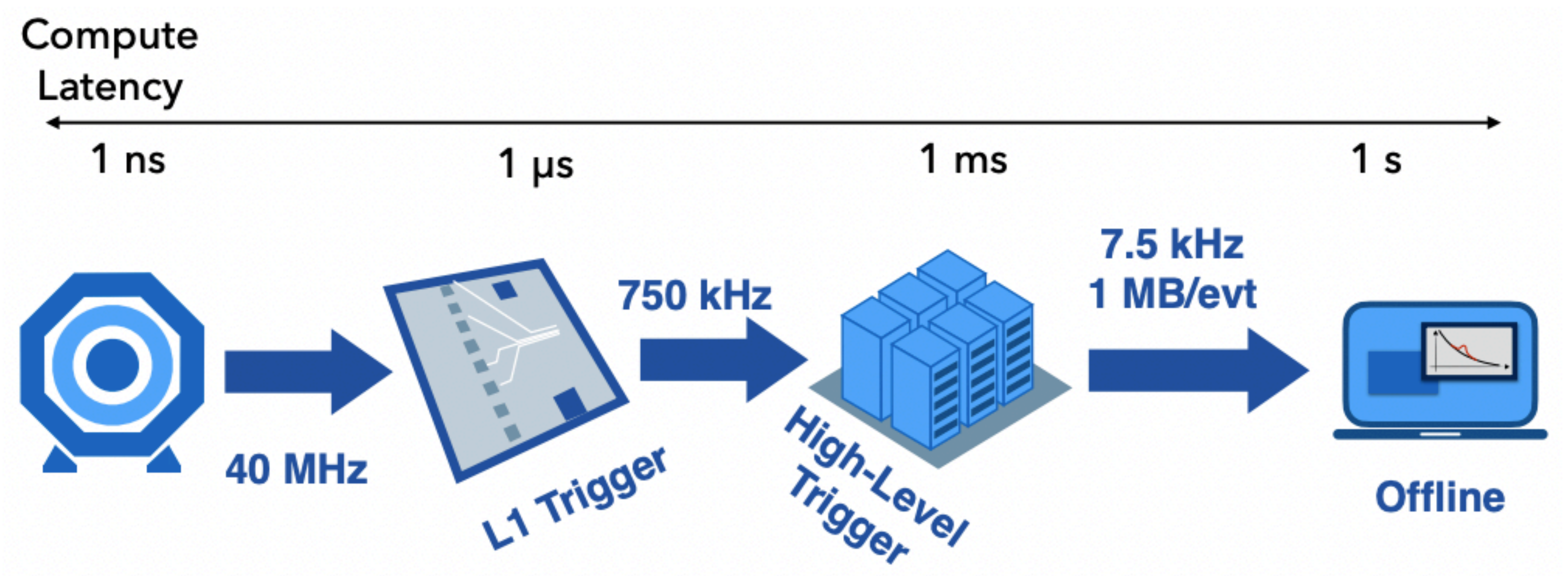
CSAID DAQ Meeting

The HL-LHC



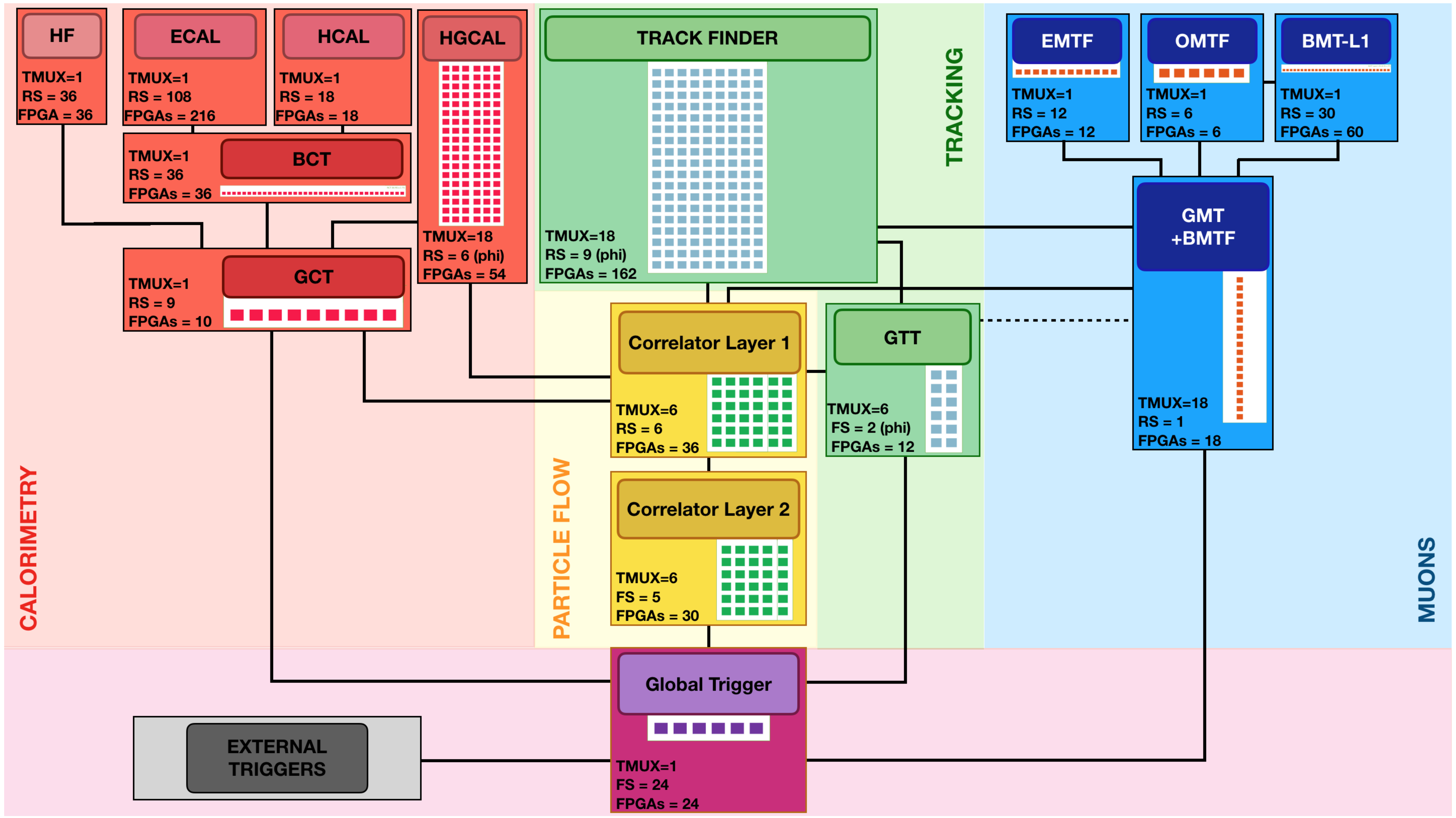
- The HL-LHC will target rare processes and precision measurements that can benefit from much higher statistics.
- The instantaneous luminosity will be high: up to 200 *p-p* collisions per bunch crossing.
- Triggering will be more challenging.

CMS Trigger for HL-LHC



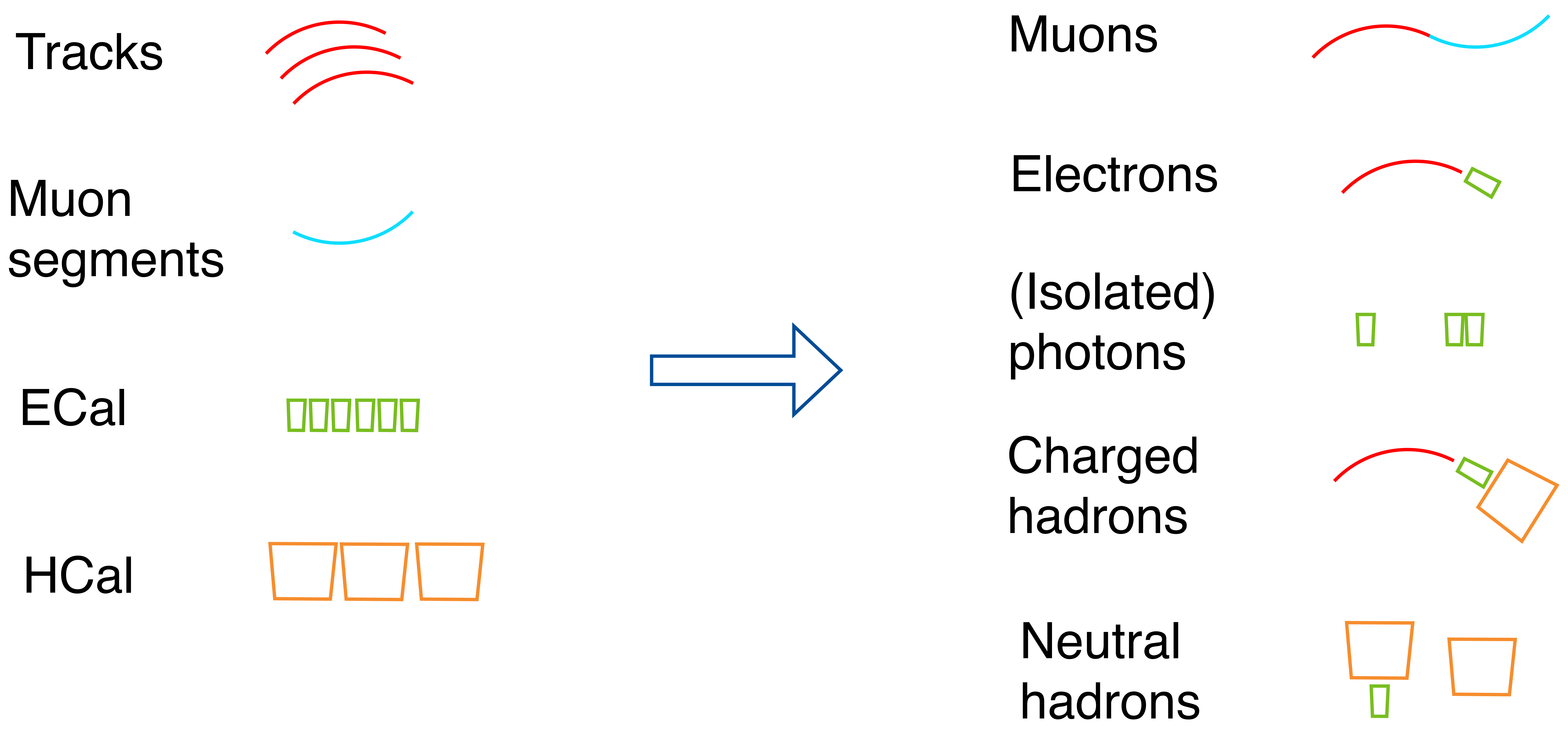
- The trigger system consists of a hardware-based L1 trigger followed by a software-based High-Level Trigger (HLT).
- L1 output bandwidth increased to 750 kHz (from 100 kHz).
- L1 latency increased to 12.5 μs (from 3.5 μs) to allow for more processing.
- Global tracking will be available in L1.

CMS L1 Trigger for HL-LHC



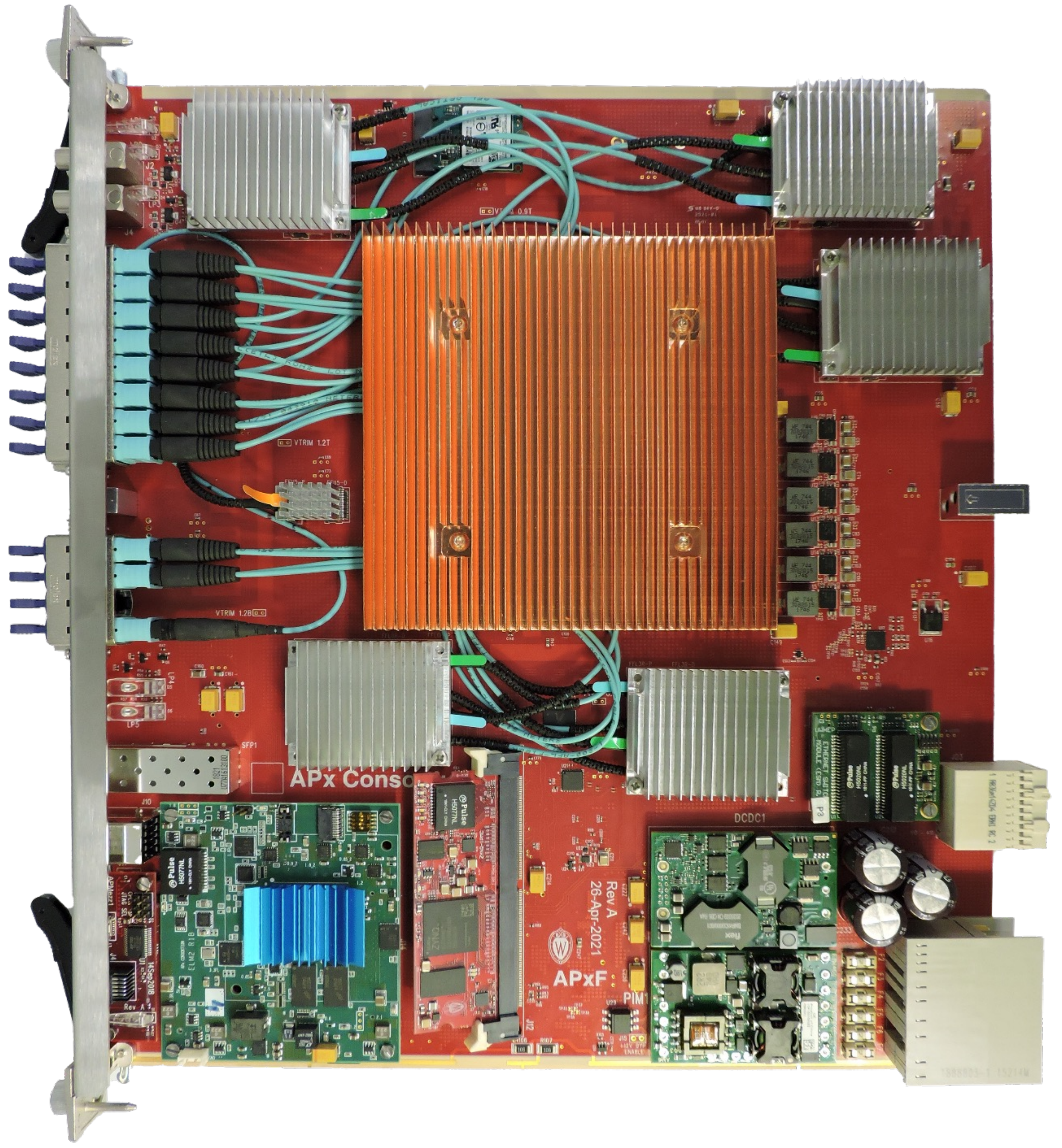
Particle Flow Reconstruction

- Combine information across all subdetectors to achieve the best possible resolution



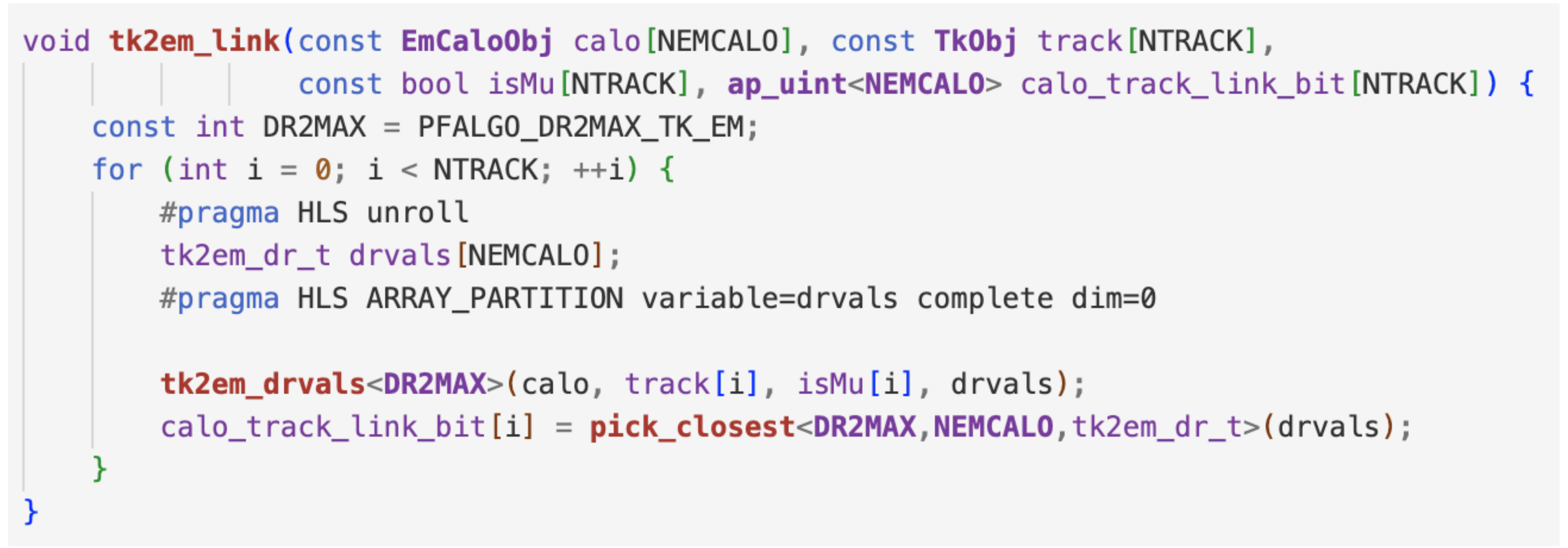
Correlator Trigger

- The correlator is divided into two parts:
  - Layer 1 (CL1): Create particle flow candidates and suppress pileup with PUPPI, create egamma objects
  - Layer 2 (CL2): Reconstruct objects from the Layer 1 inputs
- Implemented on “APx” and “Serenity” ATCA boards
  - use AMD/Xilinx Virtex Ultrascale+ FPGAs (VU13P–2 in production, VU13P–1 and VU9P–1/2 test boards)
  - Over 100 25 Gbps optical links
- CL1 uses 18 APx boards for the $|\eta| < 1.5$ barrel section and 18 Serenity boards for the endcap.
- Fermilab has an APx board with a VU9P–1 in Wilson Hall, 14th floor, in an ATCA shelf.



VHDL and HLS

- Board designers provide a framework shell (FS); developers add the algorithm.
- Top level algorithm is written in VHDL, but many algorithms are written in High Level Synthesis (HLS): C++ without dynamic memory, pragmas to direct conversion
  - Generally easier to code: do not explicitly use clock or time
  - Same code can be compiled for different clock speeds
  - Can validate logic by just compiling as C++ (used extensively in our CI/CD)
  - Can explore algorithm variations more quickly
- Downside of HLS is that you lose some control
  - VHDL can be very expressive, and some algorithms may map better to it directly
  - HDL generated from HLS is black box

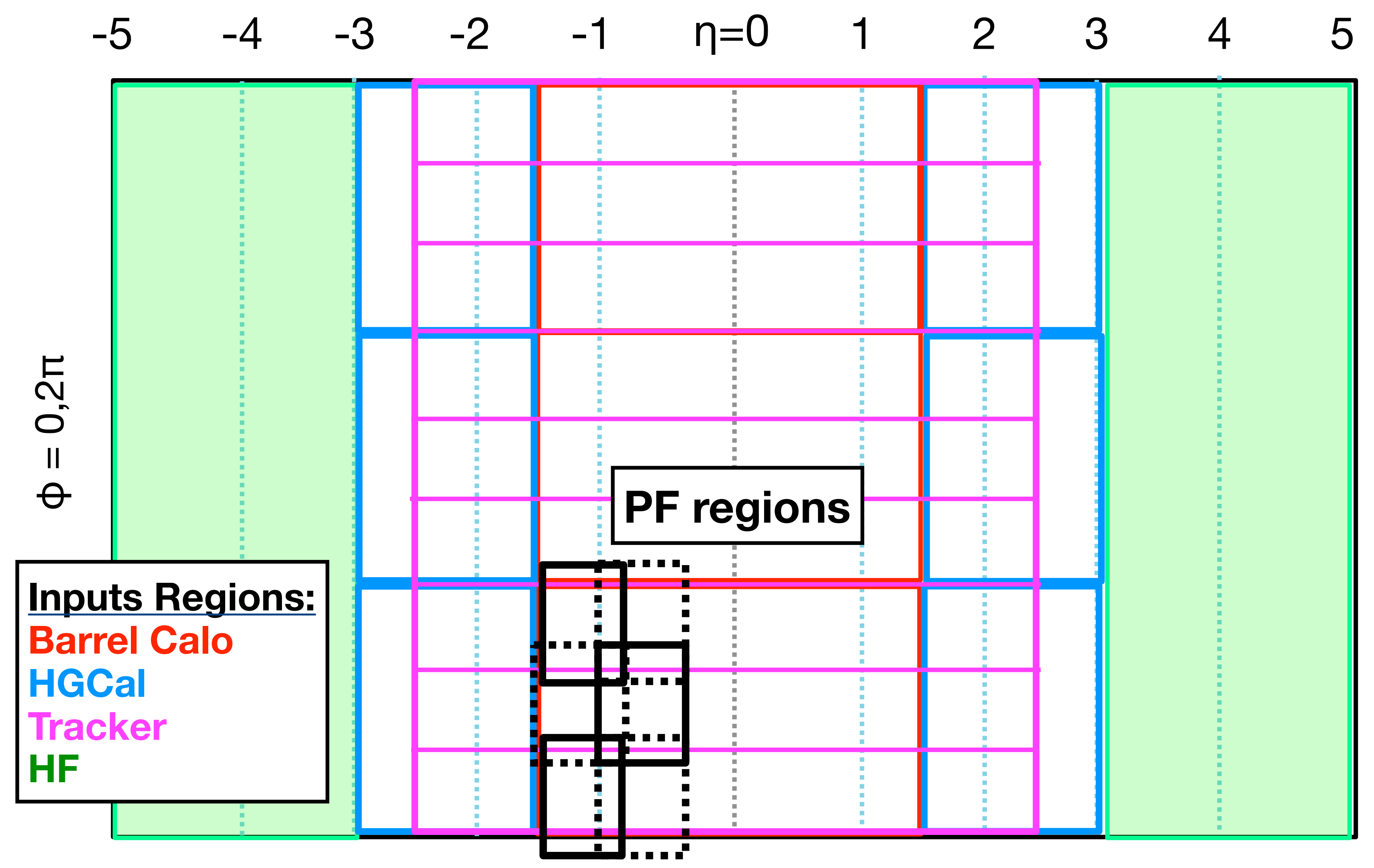


```
void tk2em_link(const EmCaloObj calo[NEMCALO], const TkObj track[NTRACK],
                const bool isMu[NTRACK], ap_uint<NEMCALO> calo_track_link_bit[NTRACK]) {
    const int DR2MAX = PFALGO_DR2MAX_TK_EM;
    for (int i = 0; i < NTRACK; ++i) {
        #pragma HLS unroll
        tk2em_dr_t drvals[NEMCALO];
        #pragma HLS ARRAY_PARTITION variable=drvals complete dim=0

        tk2em_drvals<DR2MAX>(calo, track[i], isMu[i], drvals);
        calo_track_link_bit[i] = pick_closest<DR2MAX,NEMCALO,tk2em_dr_t>(drvals);
    }
}
```

Correlator Layer 1 (Barrel)

- APx development board uses a VU9P–2 FPGA
- System is time-multiplexed 6 times (TM6): there are 6 copies of each board, and each board gets 1/6th of the events. Inputs are TM6 or TM18
- The CL1 algorithm is made up of:
  - Regionizer (VHDL): Divide the inputs into small regions (with overlaps). Subsequent algorithms run in small regions
  - Particle Flow (HLS): Run particle flow algorithm matching tracks and calorimeter clusters
  - PUPPI (HLS): determine object pile-up probability
  - Final sort (VHDL): sort the PUPPI output
  - egamma track matching (not yet implemented)



Barrel Correlator L1

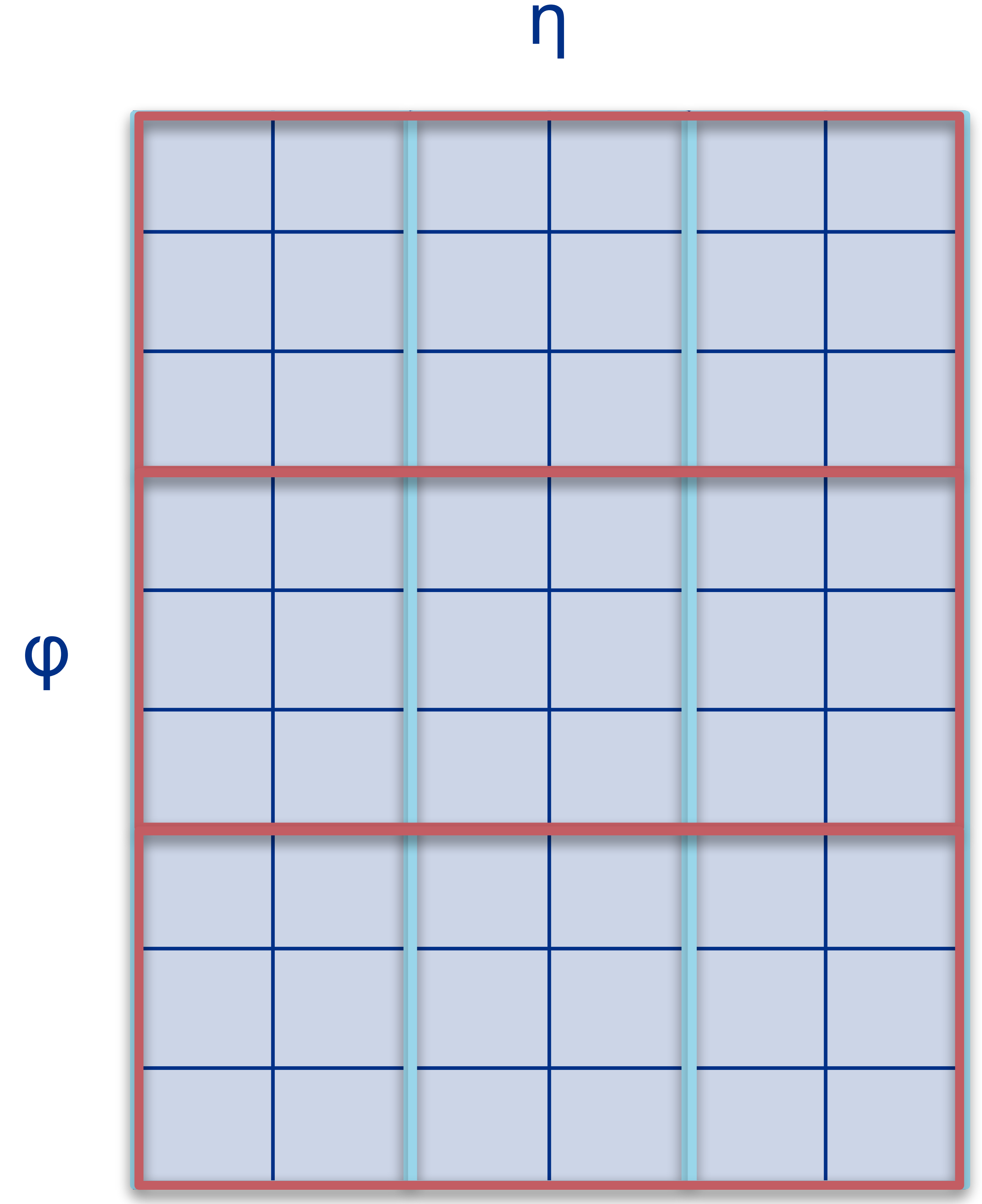
360 MHz
240 MHz
360 MHz
No registers in between; add?
2 reg*
64to96*
format conv*
clock domain FIFO*
L1 regionizer*: SR LUT lookup (no reg on input), data latched 2 reg; fill pipes
Regionizer L1 to L2 pipes†
L2 regionizer†: URAM buffers (1 reg before)
track
...
other
track
...
other
track
...
other
vtx
big region end ctrl signal
sync
Registers
pflow
shift registers
neutral puppi
No registers in between; add?
bitonic sort
clock domain FIFO
out
charged puppi and fifo
track bypass fifo
Instantiated in:
regionizer_wrapper
level1_fifo_only_buffer
level2_*
algo_wrapper
output links

* means that it is replicated per link, not just one as in the diagram

† per detector per group; each small region pair has a pipe/buffer

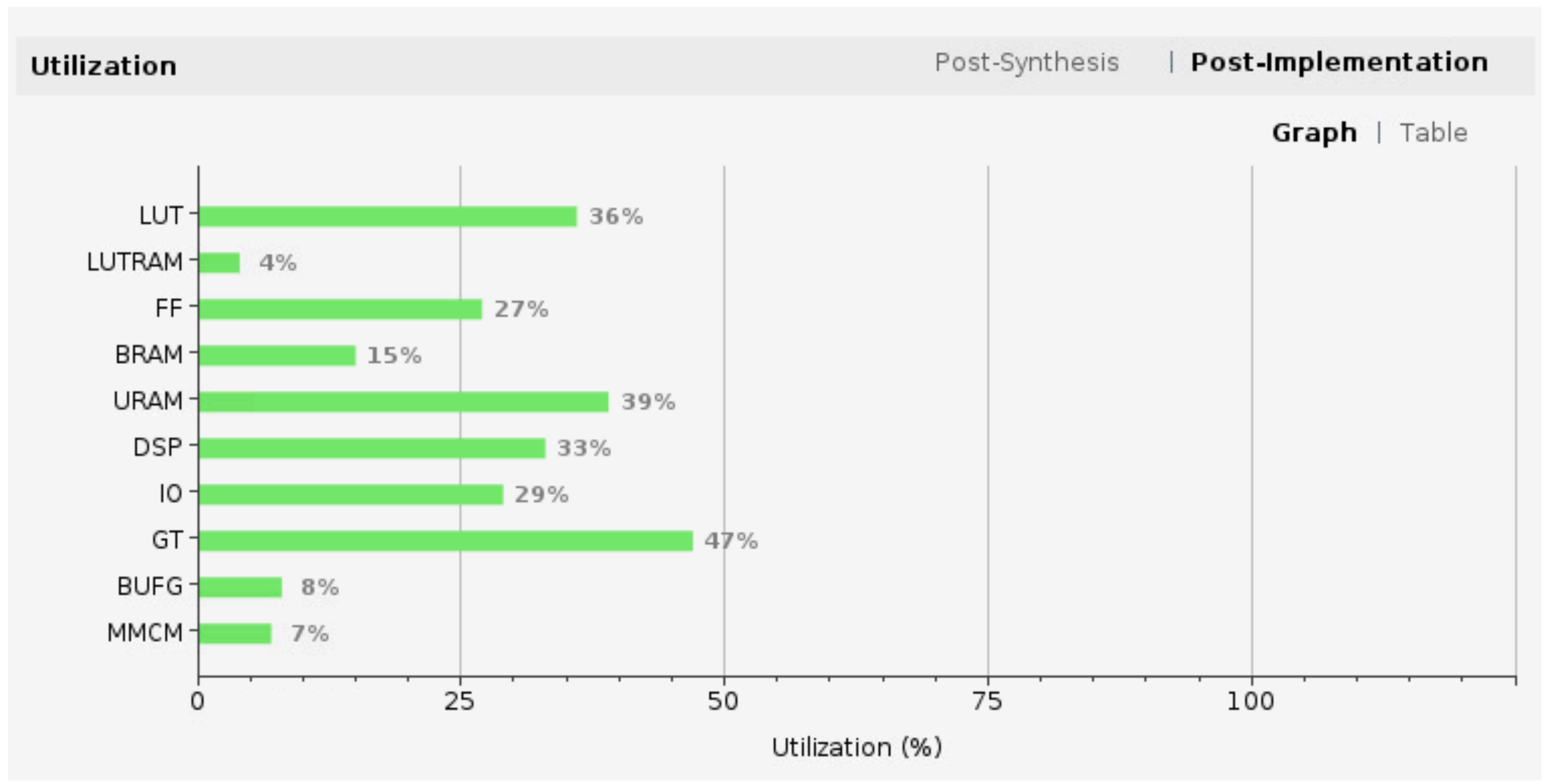
CL1 Barrel Design Evolution

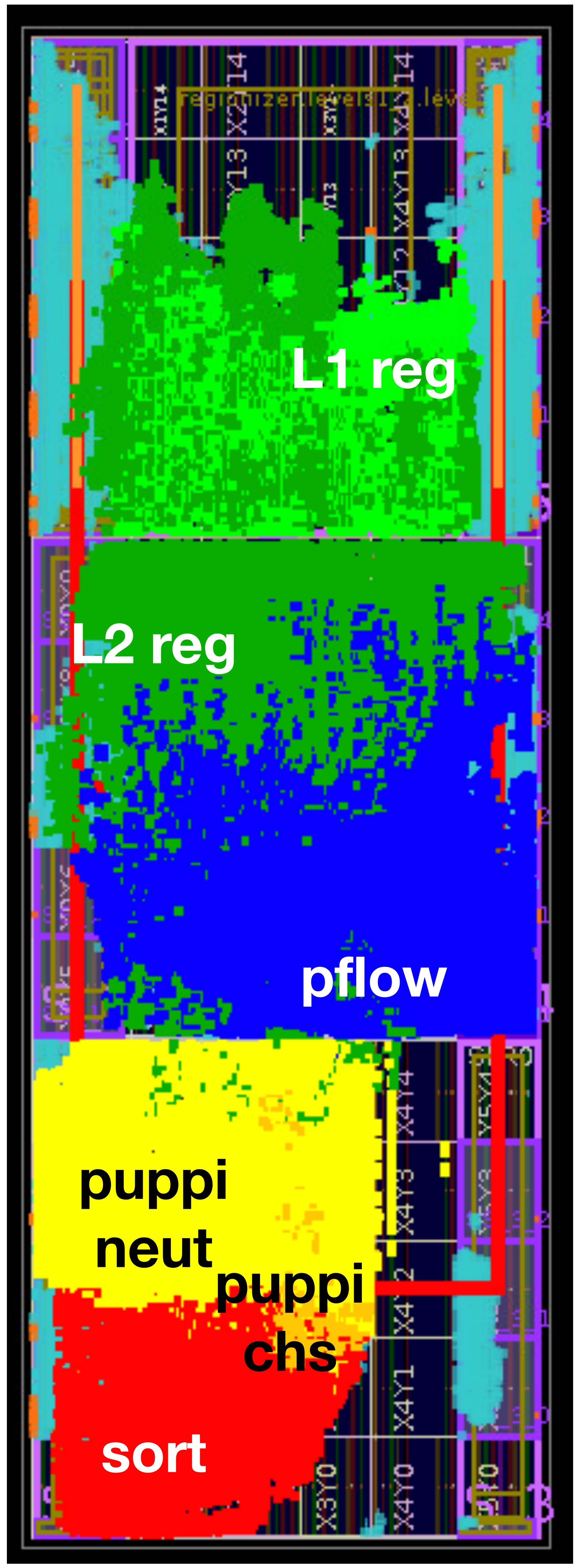
- Originally had trouble meeting timing
  - Decided to focus on –2 FPGA boards
  - Moved to using Vitis HLS
  - Wrote a bitonic sort in VHDL with initiation interval of 2
  - Inserted shift registers where needed to ease routing
  - Tried to control the layout and flow of the data across the Super-Logic Regions (SLRs) with link assignments and a bit of floorplanning
- The barrel is divided into 3 big regions, each assigned to a board (replicated 6 times)
  - Originally, the big regions were segmented in η (blue)
  - To reduce the number of input fibers, we recently moved to big regions segmented in ϕ (red)



Correlator Layer 1 (Barrel)

- Recently performed multiboard tests
  - Vertex information from the Global Track Trigger (GTT)
  - Other data is read from playback buffers
- Identical results with alternate configuration, reading everything from playback buffers
- Vivado 2022.2 was used to build the barrel firmware
  - (Used 2021.2 Vitis HLS)
- Latency:
  - 0.86 µs (last-in, last out)
  - 1.16 µs (first-in, first out)



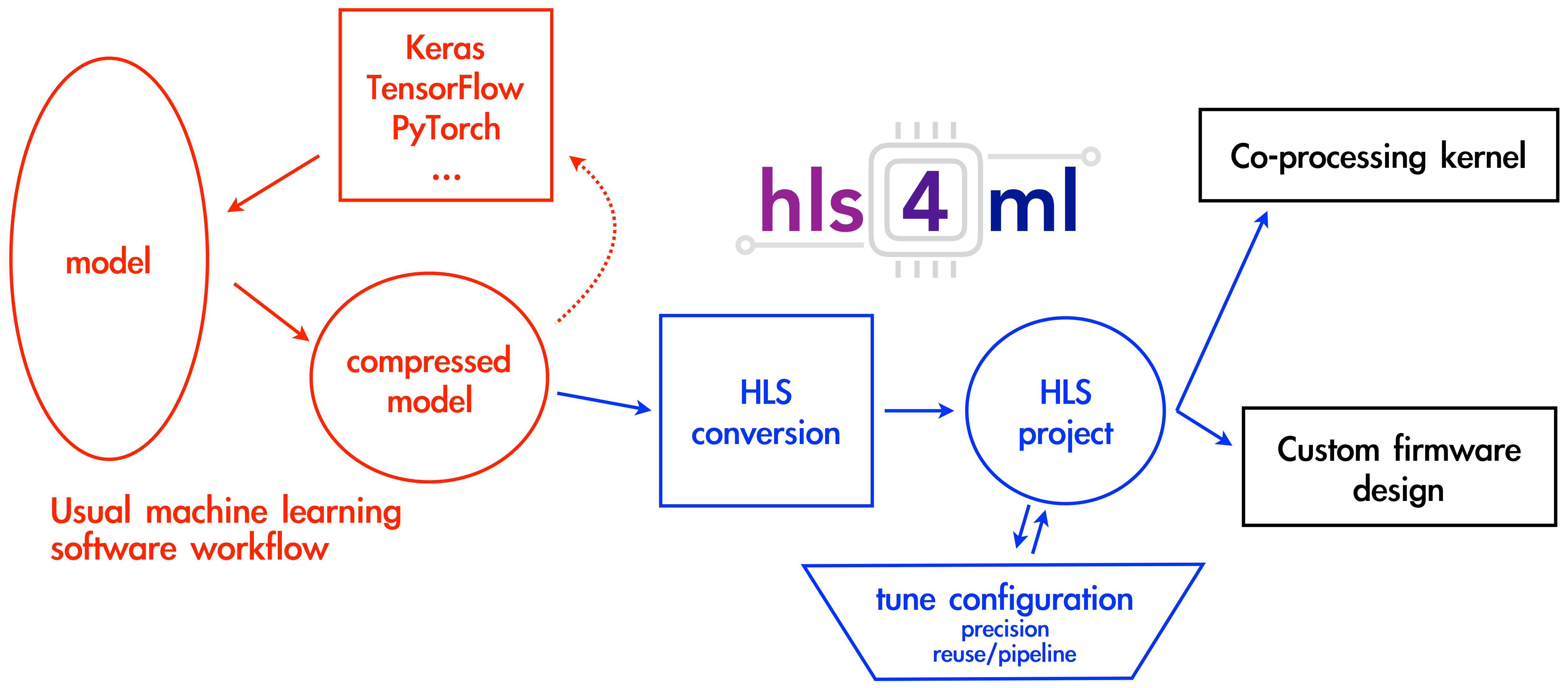


CL1 Barrel Near Term plans

- When we had the CL1 segmented in η, the firmware simulation and emulation software predictions matched
- With the switch to ϕ segmentation, the firmware got ahead of the emulator
- We have recently updated the emulator and better integrated it into CMSSW.
- Working to add an emulation-simulation test in our CI/CD suite
  - estimated time: 1-2 weeks
- Add egamma matching algorithm (goal: May)
- Build firmware for VU13P–2 (goal: end of April)
- CL1 to CL2 APx multi-board test
- Multi-board tests with non-APx boards

Converting NNs to HLS: hls4ml

- hls4ml is a compiler taking Keras, pytorch, or ONNX as input and producing HLS
- Was originally developed for CMS trigger applications, and is being used in HL-LHC trigger development, including some CL2 algorithms
- CSAID plays a major role in the development for hls4ml


Keras
TensorFlow
PyTorch
...
model
compressed model
Usual machine learning software workflow
hls4ml
HLS conversion
HLS project
tune configuration
precision
reuse/pipeline
Co-processing kernel
Custom firmware design