MET Algorithm Workflow for CMS Phase-II Level-1 Trigger

Junwon Oh

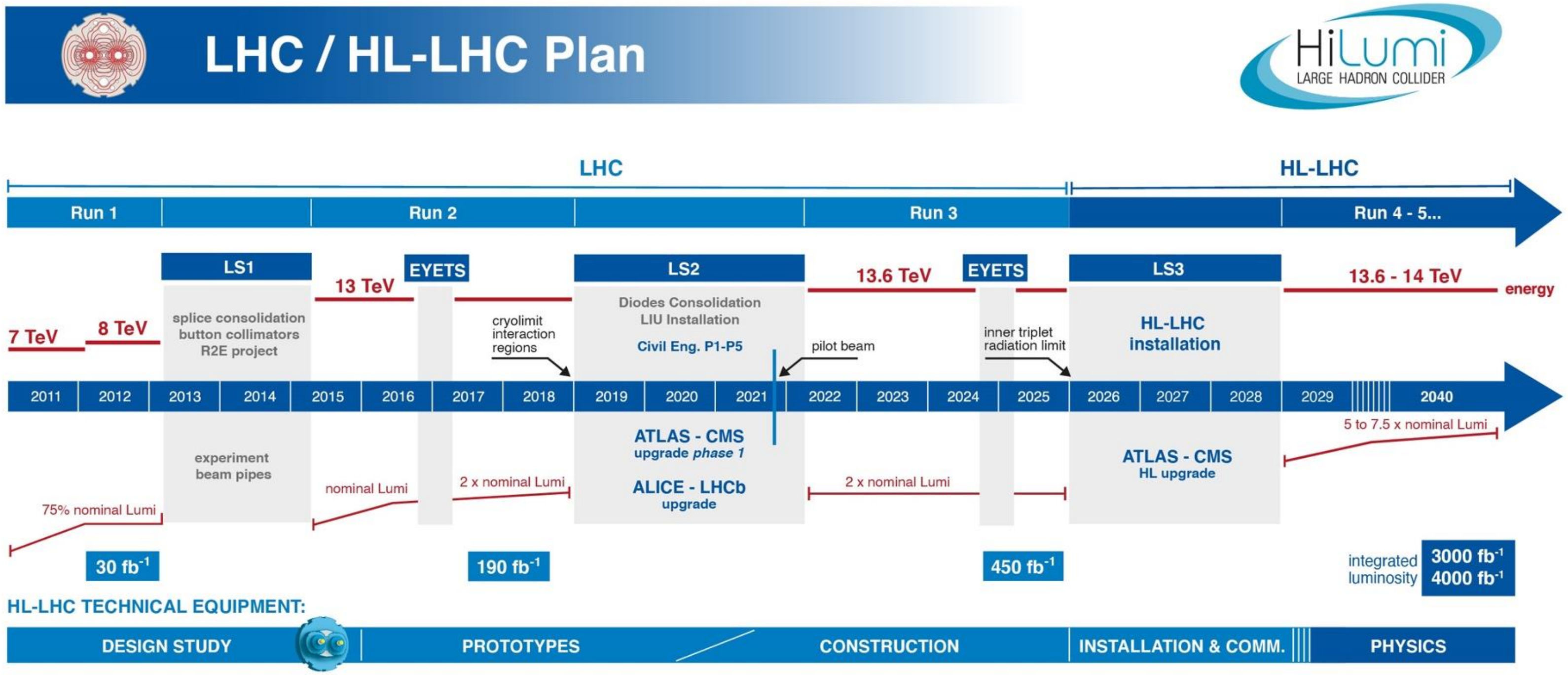
서울시립대학교
UNIVERSITY OF SEOUL

2025.08.25

Contents

- HL-LHC and CMS Phase-II Upgrade
- CMS Level-1 Trigger system
- HLS workflow
- Missing Transverse Energy Algorithm for Level-1 Trigger system
- Optimizations
- Implementations

HL-LHC and CMS Phase-II Upgrade



	Collision Energy	Interactions per bunch crossing	Instantaneous Luminosity	Ex) ggF Higgs Cross-section
Run3 (Current)	13.6TeV	$60\ collisions \times 40MHz$	$2 \times 10^{34} cm^{-2} \cdot s^{-1}$ $2 \times 10^{-5} fb^{-1} \cdot s^{-1}$	46.67 pb
HL-LHC (2026~)	14TeV	$200\ collisions \times 40MHz$	$5\sim7.5 \times 10^{34} cm^{-2} \cdot s^{-1}$ $5\sim7.5 \times 10^{-5} fb^{-1} \cdot s^{-1}$	50.35pb

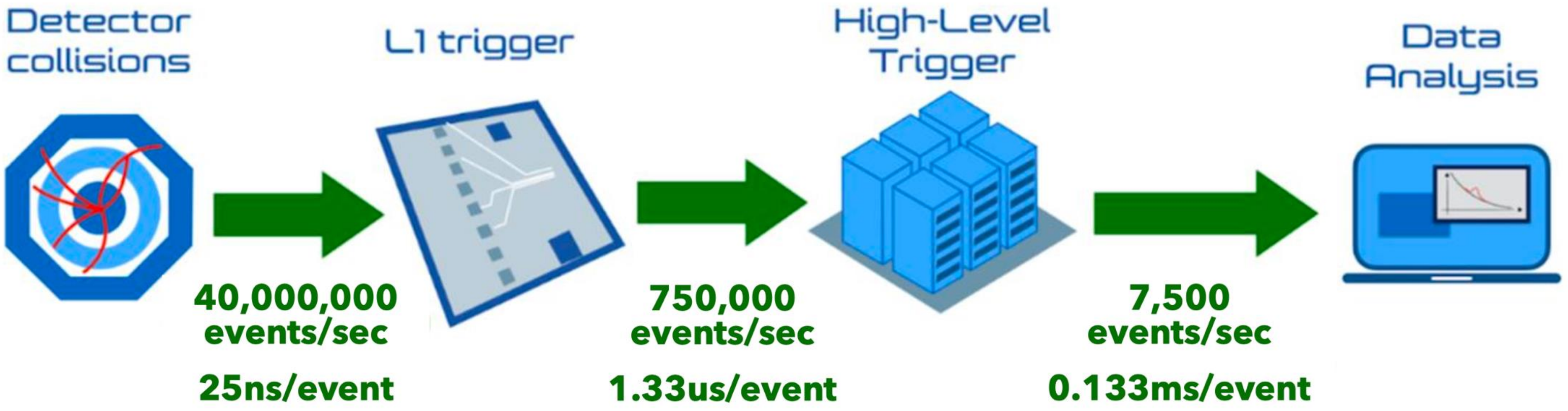
$1barn = 1b = 10^{-28} m^2$

$ggF\ Higgs\ Production$

$2 \times 10^{-5} fb^{-1} \cdot s^{-1} \times 46.67pb$

$= 0.9334\ per\ second$

CMS Level-1 Trigger system



- The Level-1 (L1) Trigger is the first stage in filtering data from collisions in CMS detector.
- L1 Trigger decisions are made within a few microseconds, operating under strict time constraints.
- Specialized hardware, such as FPGAs, is used to ensure fast and parallel data processing.
- The primary goal is to retain high-energy events while discarding low-relevance data.

CMS Level-1 Trigger system

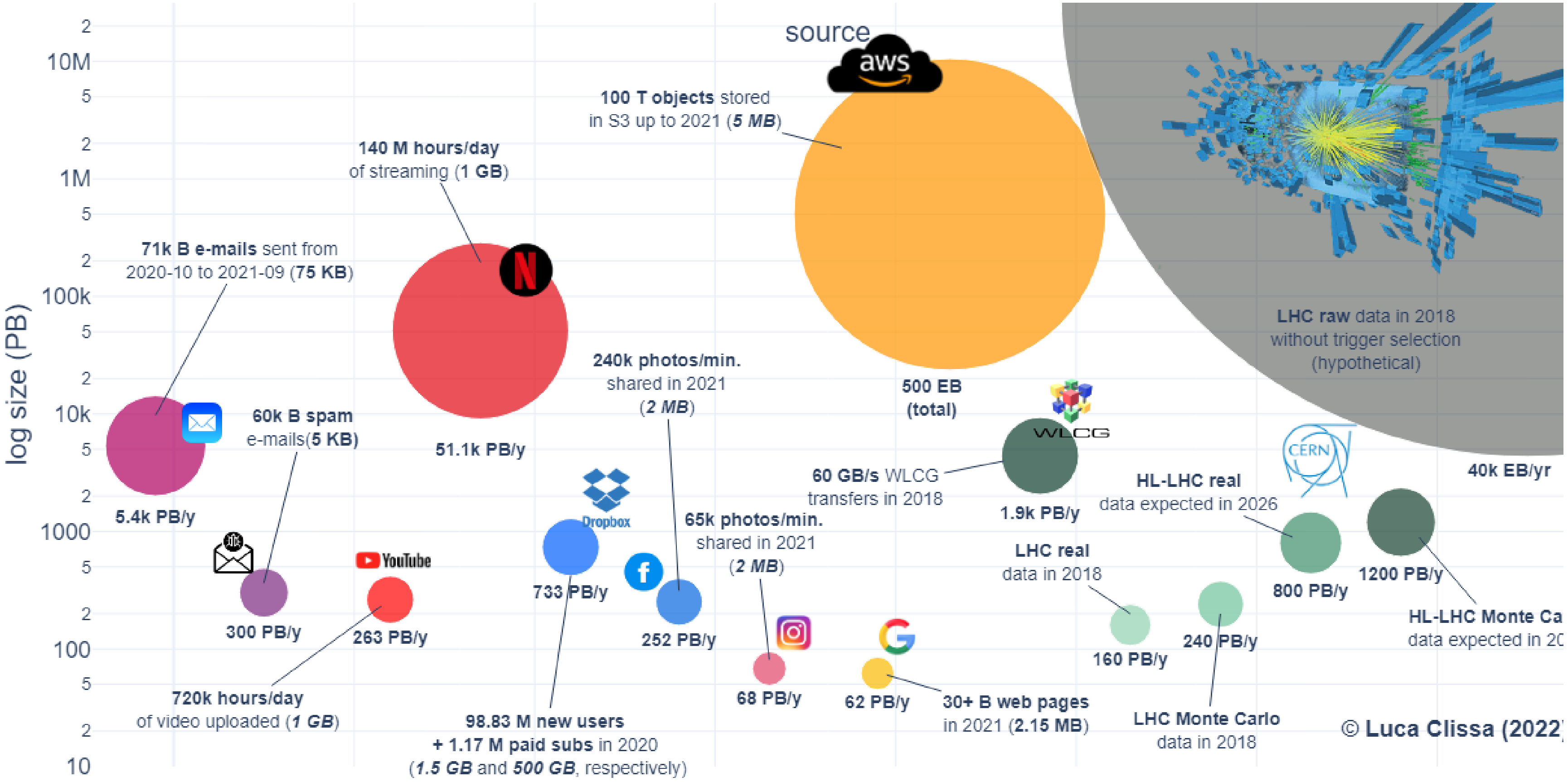
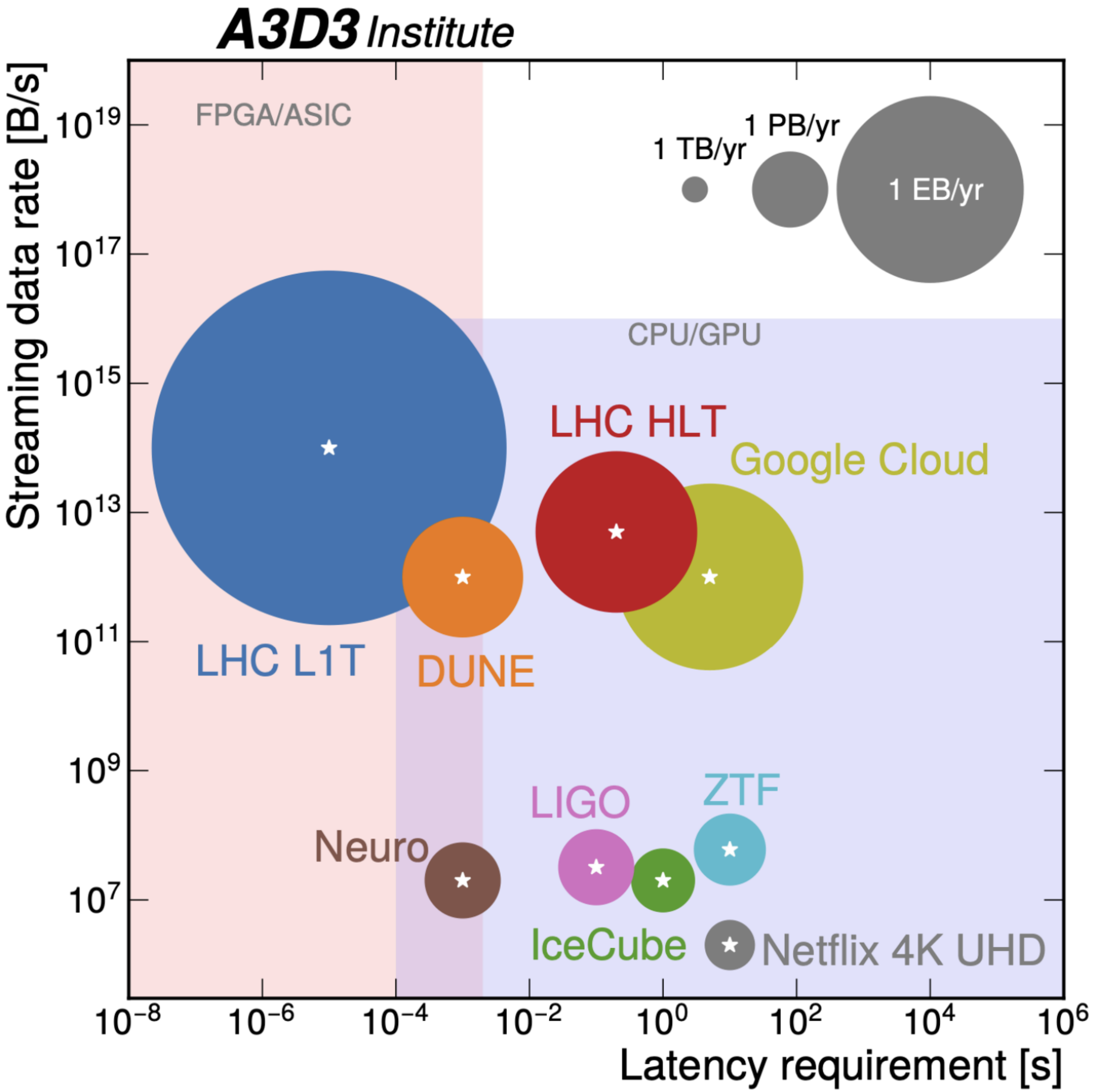
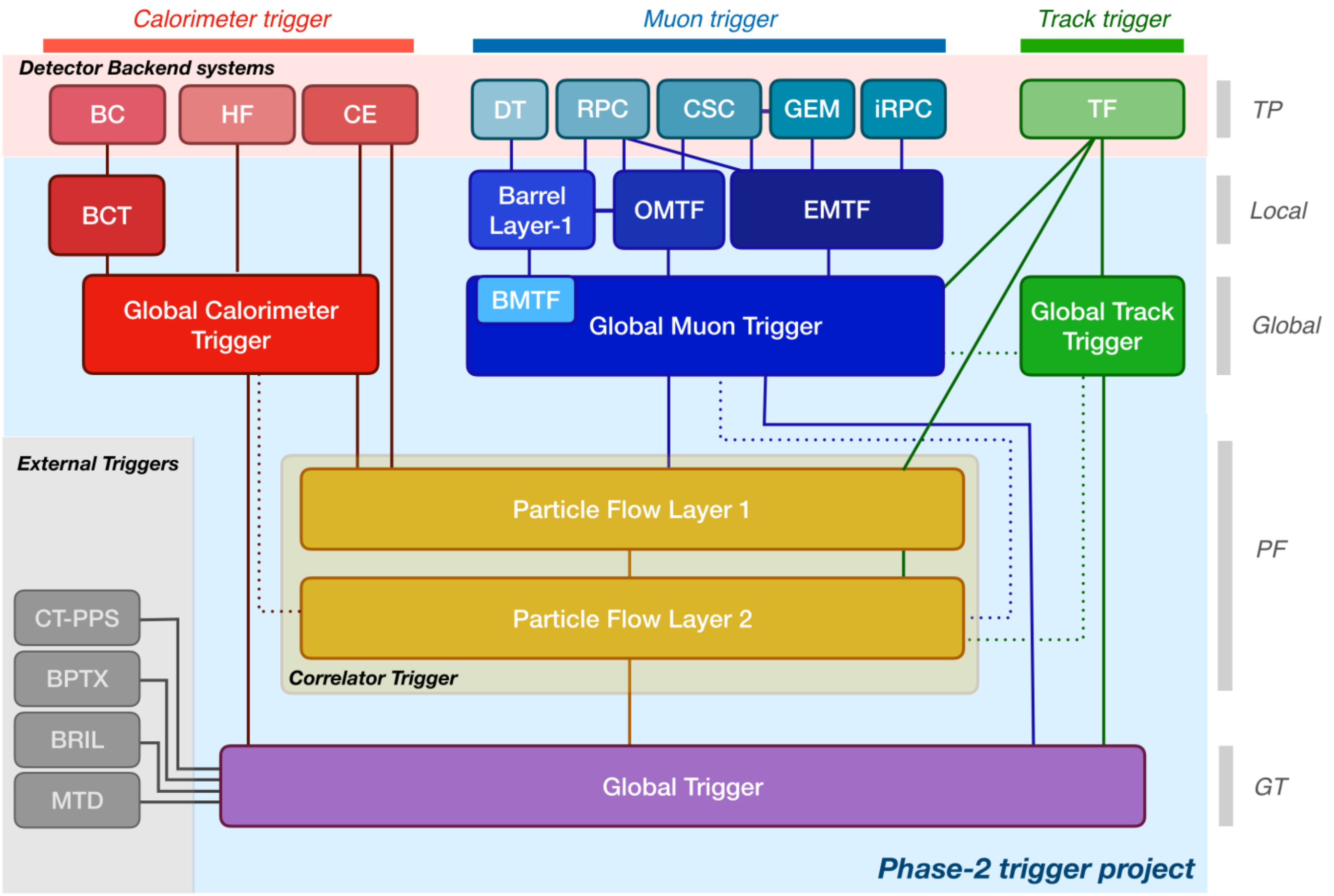


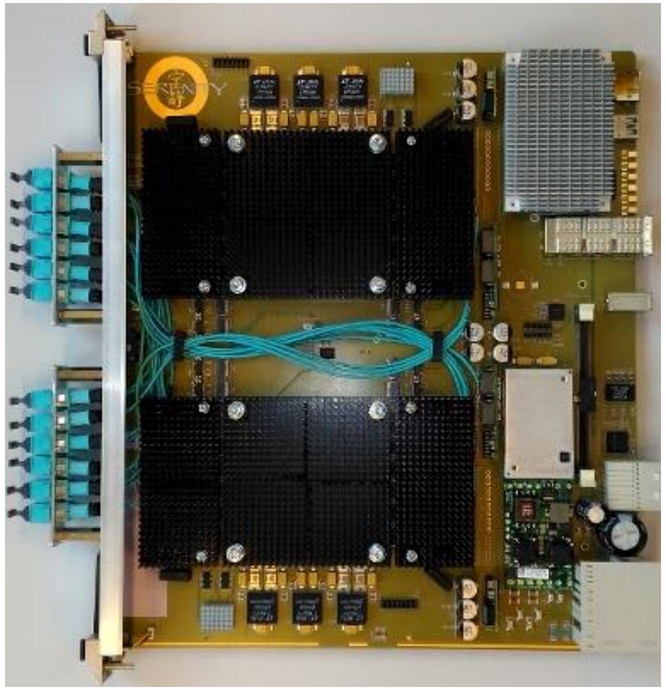
Figure 2.3: **Big Data sizes.** Bubble plot of the orders of magnitude of data produced by important big data players. The balloon areas illustrat
and the text annotations highlight the key factors considered in the estimates. Average per-unit sizes are reported in parentheses, where italic
reconstructed based on likely assumptions because no references were found.

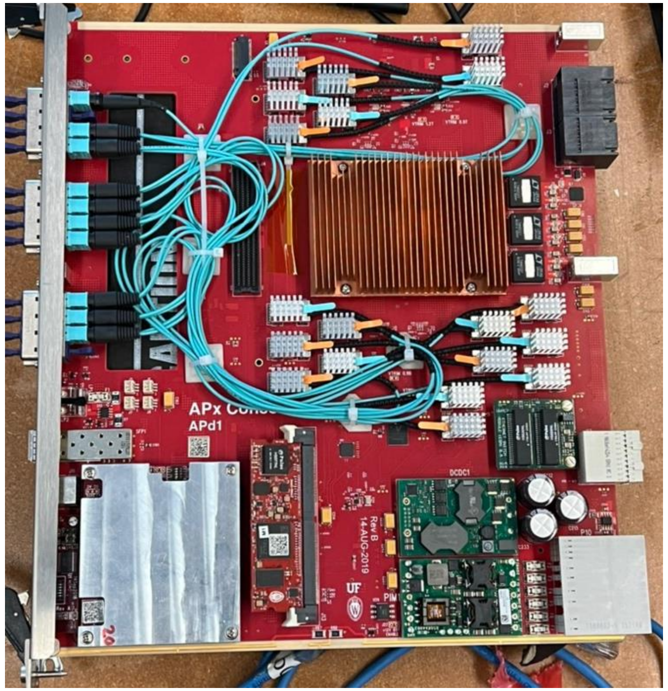
CMS Level-1 Trigger system



CMS Level-1 Trigger system



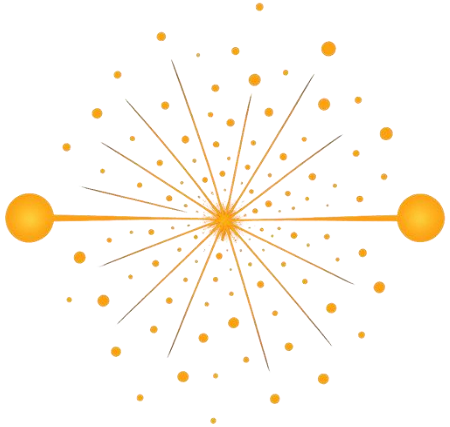




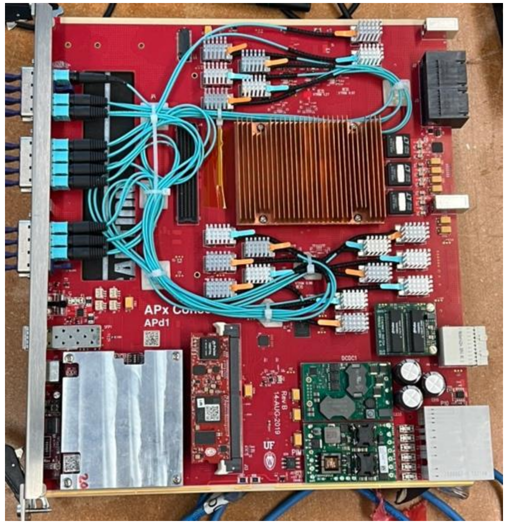
Correlator Trigger (CT) – Major part of L1 Trigger Upgrade

- Aggregating inputs from all upstream systems for L1 physics objects
- The CT will be capable of providing essential physics objects
  - Missing Transverse Momentum (MET),
  - Jets and their tagging information based on the PF (Particle Flow) or PUPPI (Pileup Per Particle Identification) candidates

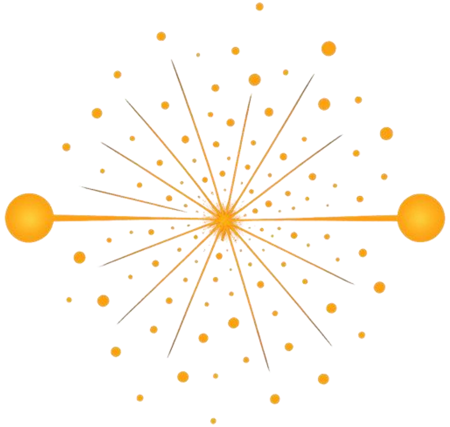
TMUX



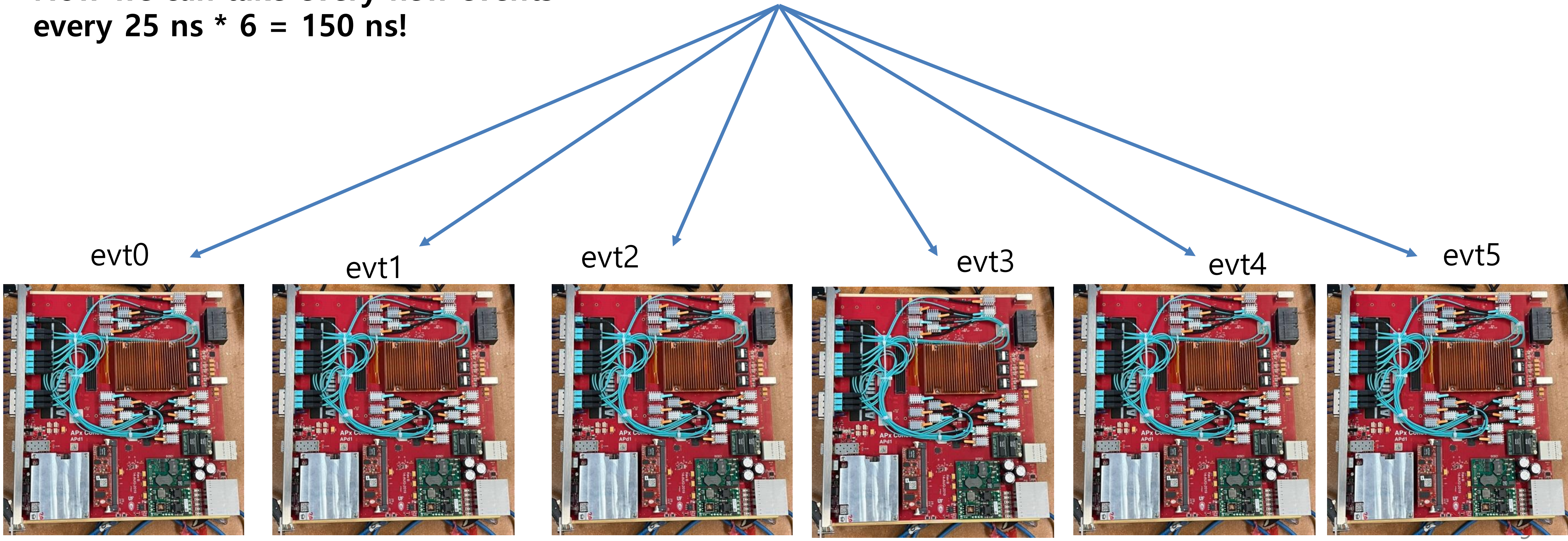
New Events every 25 ns (40 MHz)



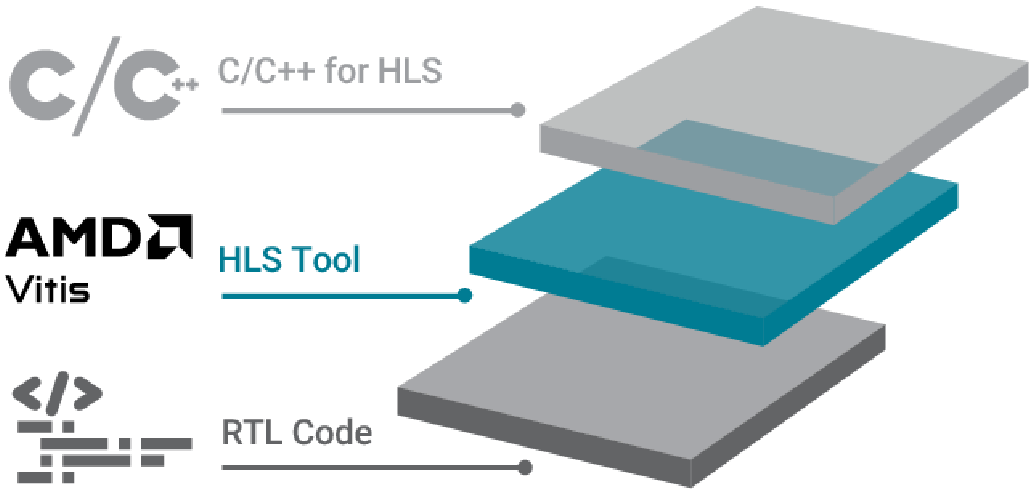
TMUX



Now we can take every new events every 25 ns * 6 = 150 ns!

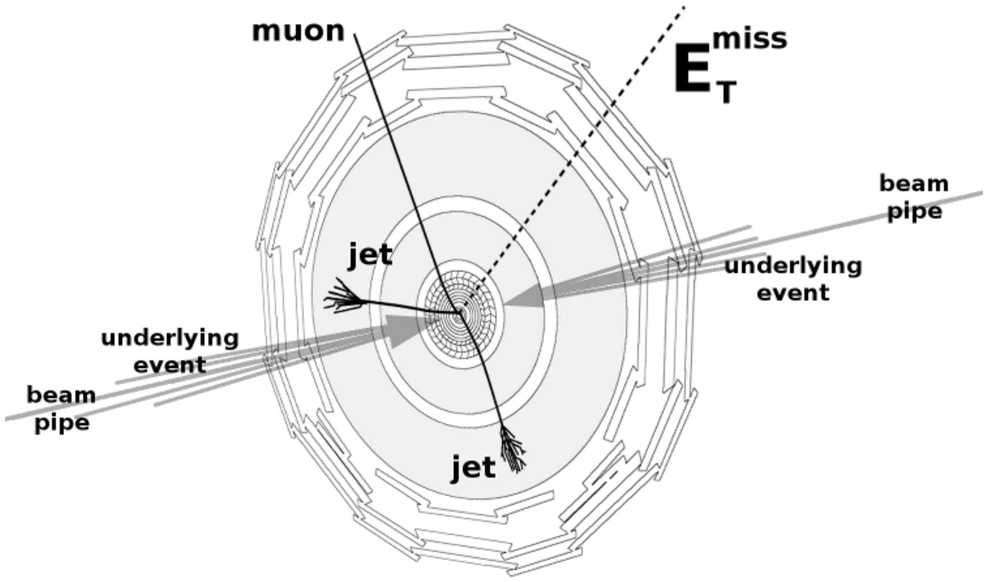


HLS workflow



- Start from C/C++ descriptions of the algorithm.
- High-Level Synthesis (HLS) automatically converts it into RTL (Verilog/VHDL).
- RTL can then be implemented on FPGA hardware.
- This approach avoids manual hardware coding, which is difficult for physicists.
- Widely used in our field to speed up development on FPGAs.

Missing Transverse Energy Algorithm for Level-1 Trigger system



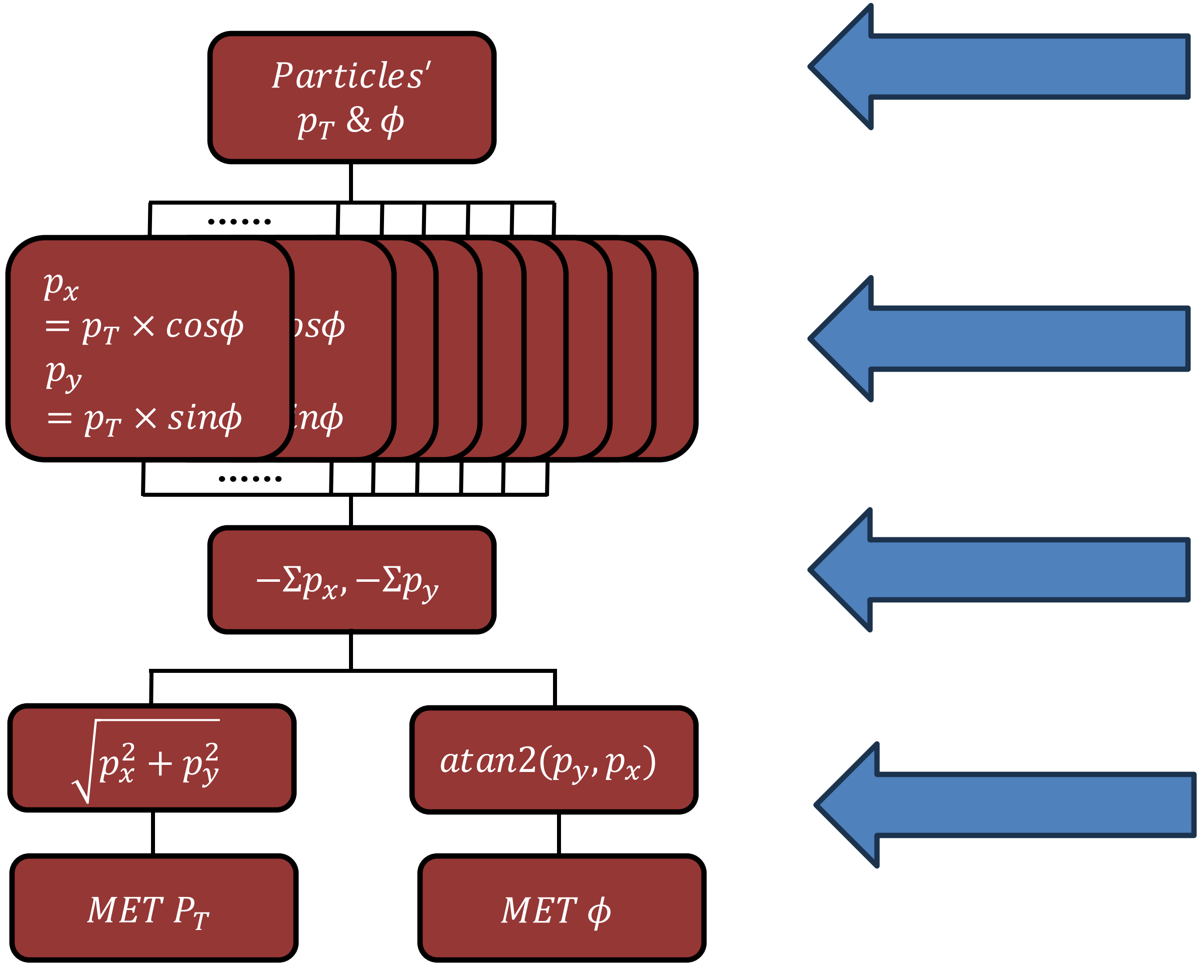
PUPPI MET

: Pileup Per Particle Identification MET

- PUPPI algorithm improves the identification of particles.
- MET serves as a proxy for invisible particles.
- Calculate the negative sum of particles' momenta

$$\boxed{\boldsymbol{p_T^{miss}} = -\boldsymbol{\Sigma_{particles}}\overrightarrow{\boldsymbol{p_T}}}$$

MET Algorithm for Level-1 Trigger system



$Particles'$
p_T & ϕ

$p_x = p_T \times cos\phi$
$p_y = p_T \times sin\phi$

$-\Sigma p_x, -\Sigma p_y$

$\sqrt{p_x^2 + p_y^2}$

$atan2(p_y, p_x)$

$MET\ P_T$

$MET\ \phi$

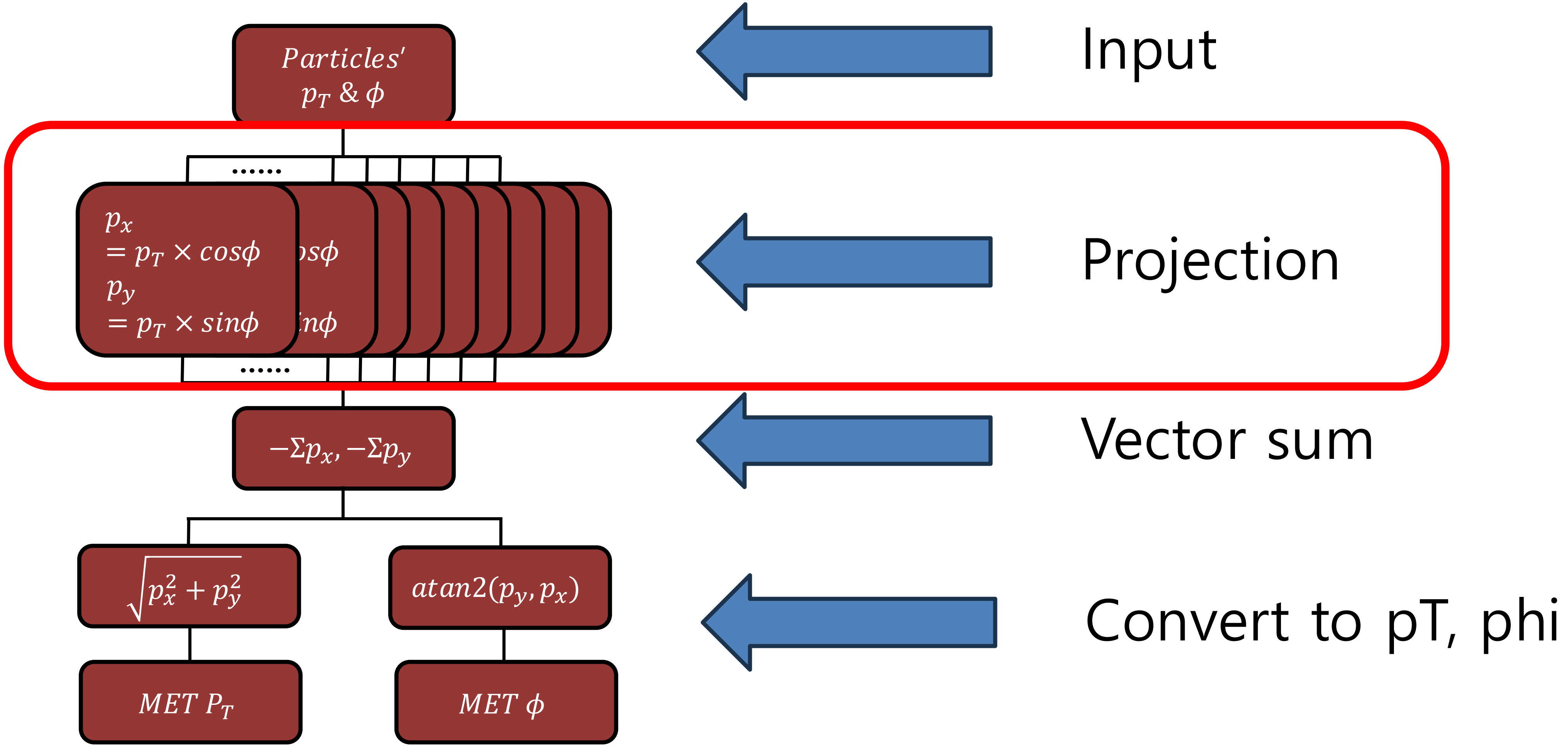
Input

Projection

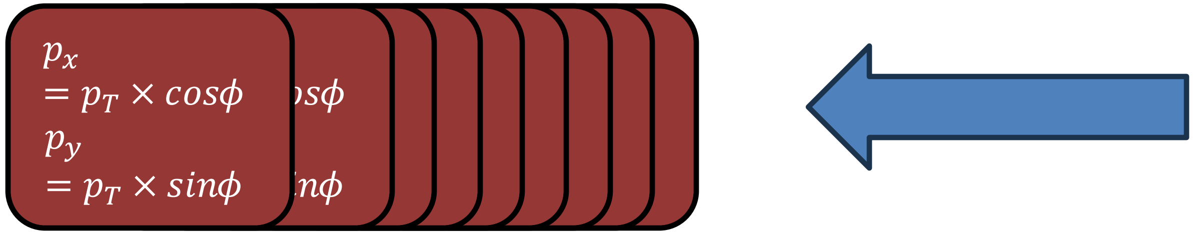
Vector sum

Convert to pT, phi

MET Algorithm for Level-1 Trigger system



Optimizations

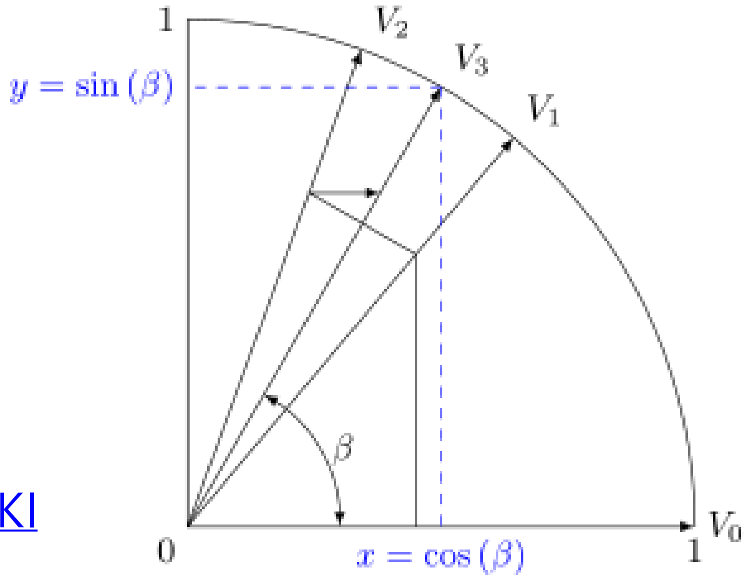


$p_x = p_T \times cos\phi$

$p_y = p_T \times sin\phi$

Projection

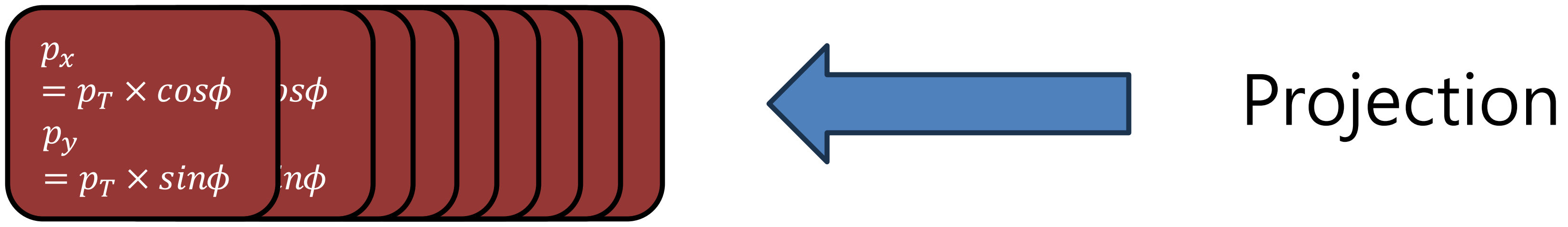
How can we implement trigonometric functions on FPGA?
Typically, the CORDIC algorithm is used. However…



WIKI

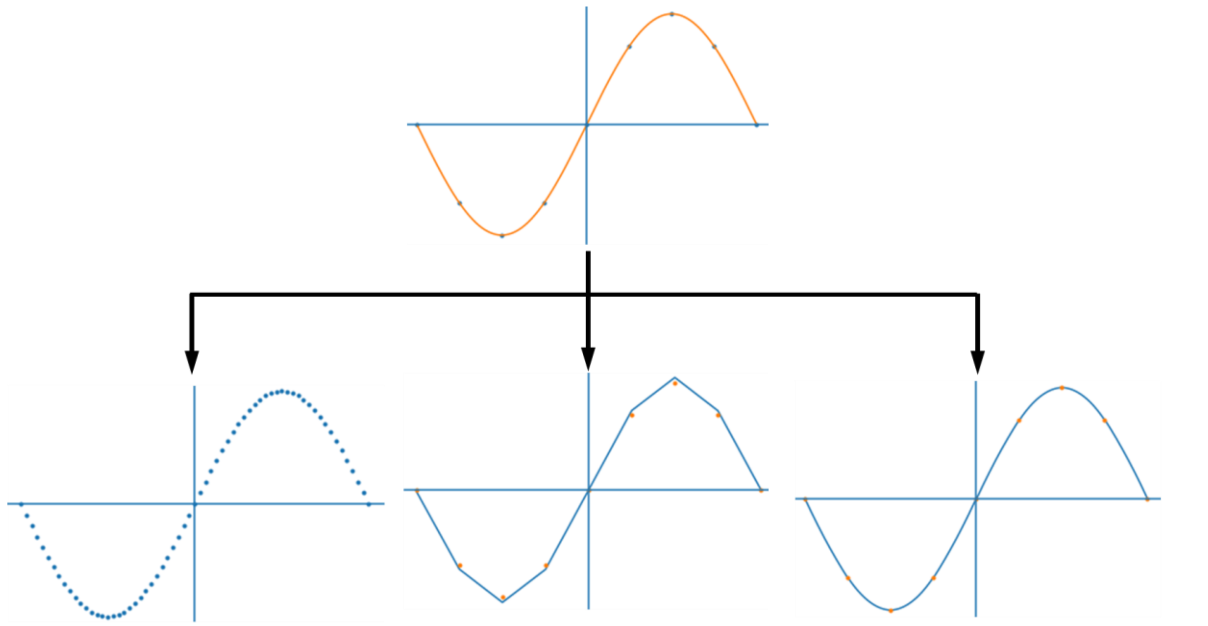
- Iteratively rotates a vector by predefined angles.
- Each iteration refines the approximation of sine and cosine.
- Requires many sequential loop steps for high precision.
- This sequential nature leads to longer latency on FPGA.

Optimizations



p_x
$= p_T \times cos\phi$
p_y
$= p_T \times sin\phi$

Projection



[Example of making LUT Array]

[Example of 1st Order Polynomial Interpolation]

[Example of 2nd Order Polynomial Interpolation]

Look Up Table (LUT)

- LUT is one of the easiest and efficient way to express any functions by saving values as numerical tables in the hardware
- We set **512 points** for the sin & cos functions
- Lower latency and smaller resource usage than the HLS math library in trade-off of numerical precision

Polynomial Interpolation

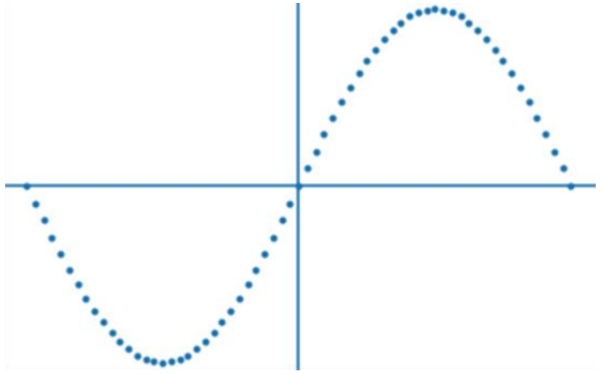
- Trigonometric functions can be approximated with piecewise polynomial
- We used 1st & 2nd order polynomial interpolation algorithms and set 16 points on sin & cos functions
- The resource usage and latency are better than the HLS math library
- The precision is significantly improved

Optimizations

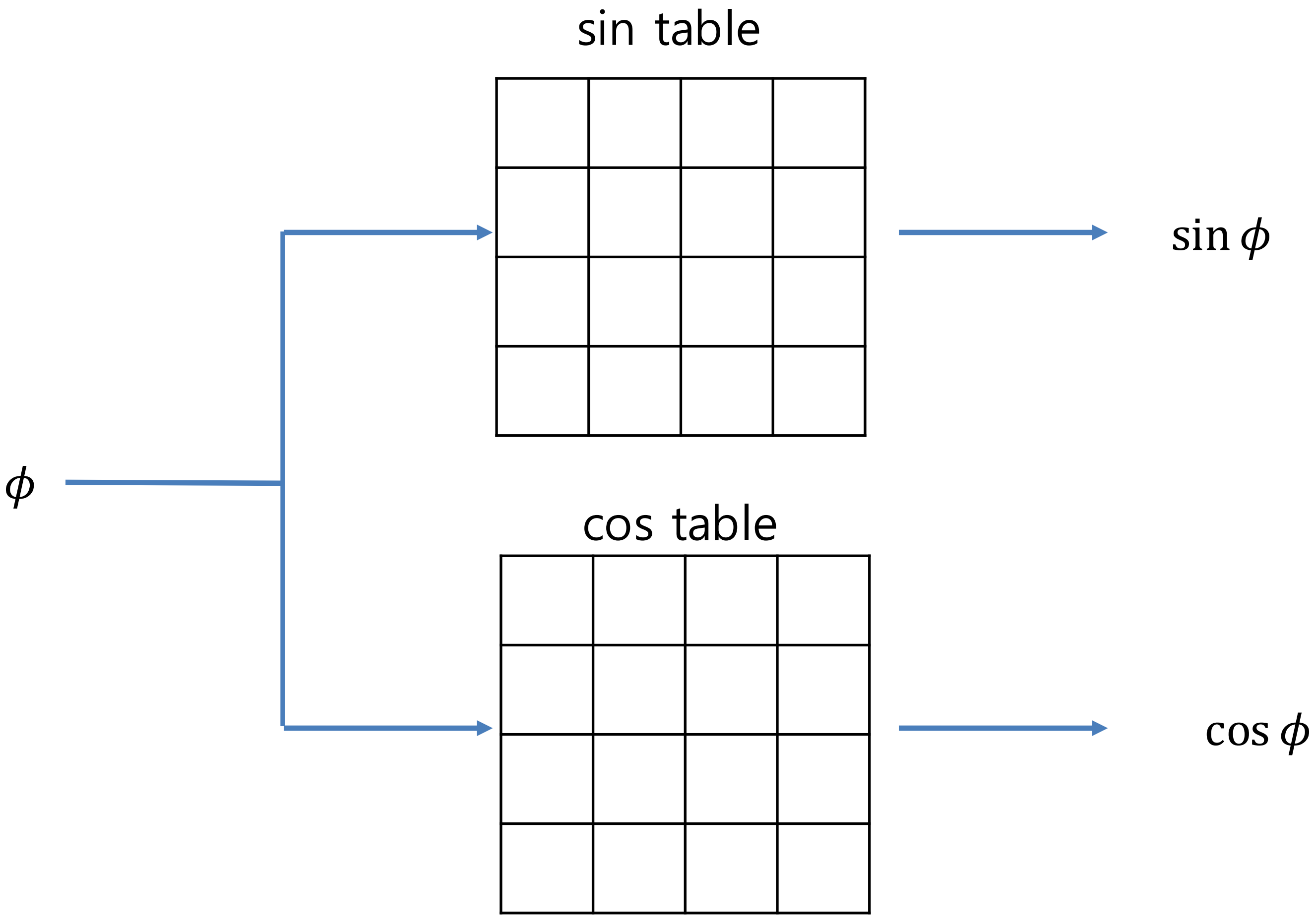
$p_x = p_T \times cos\phi$

$p_y = p_T \times sin\phi$

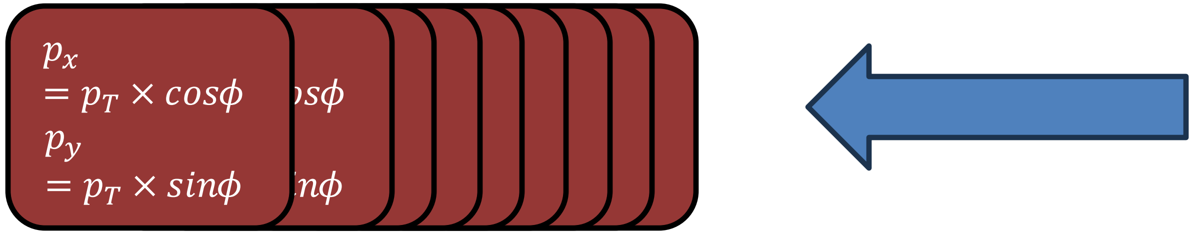
Projection



[Example of making LUT Array]



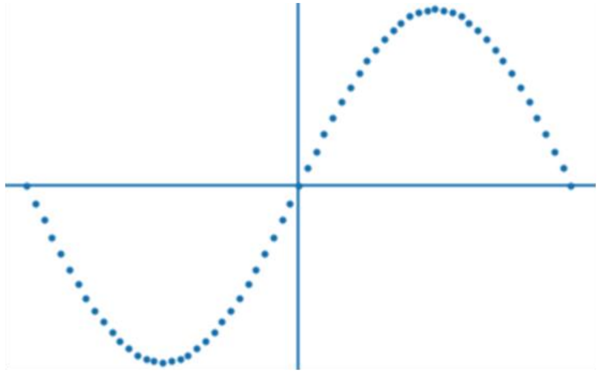
Optimizations



$p_x = p_T \times cos\phi$

$p_y = p_T \times sin\phi$

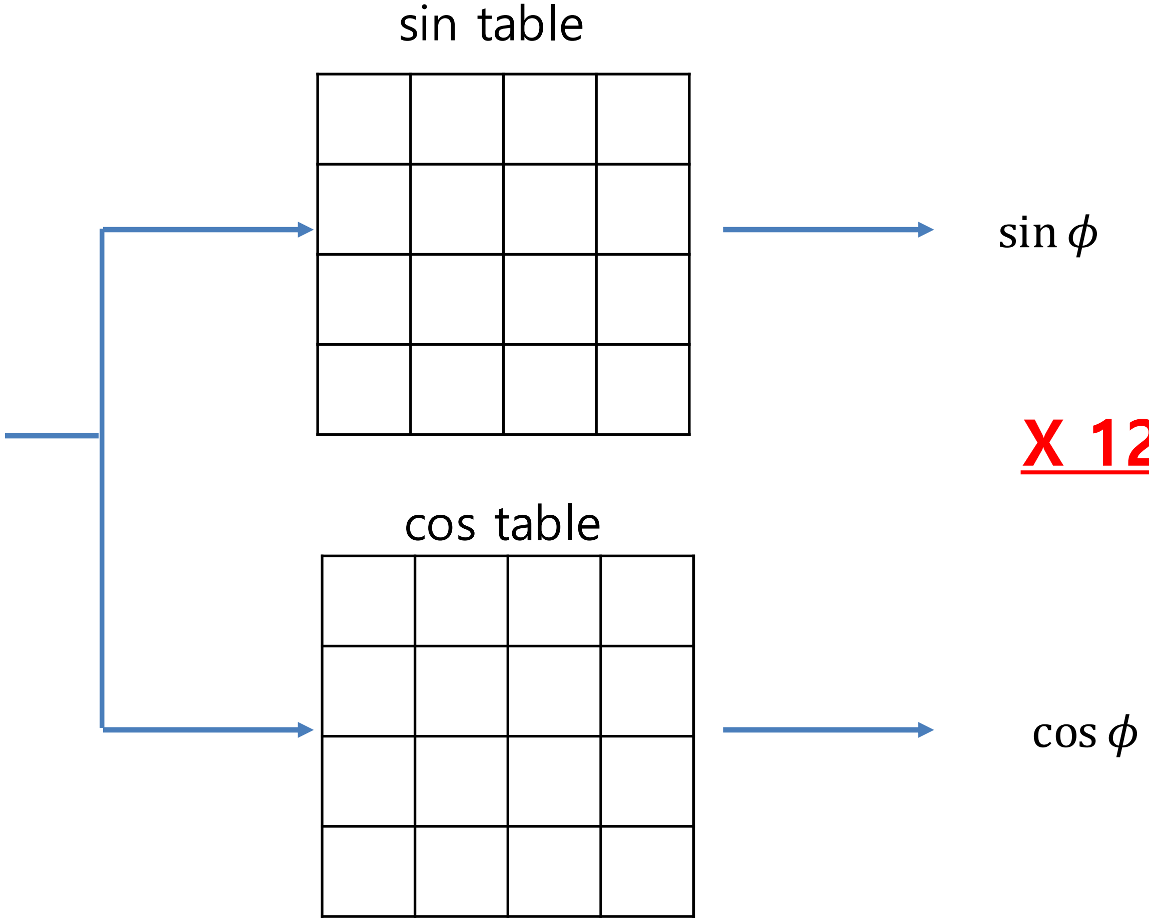
Projection



[Example of making LUT Array]

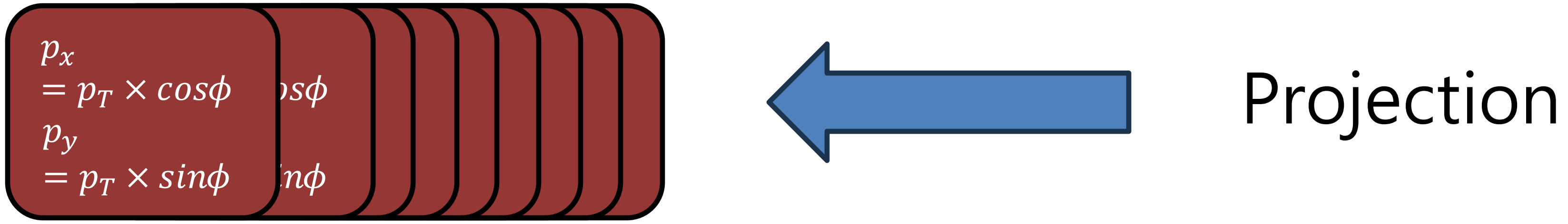
ϕ

11 bits



X 128 particles

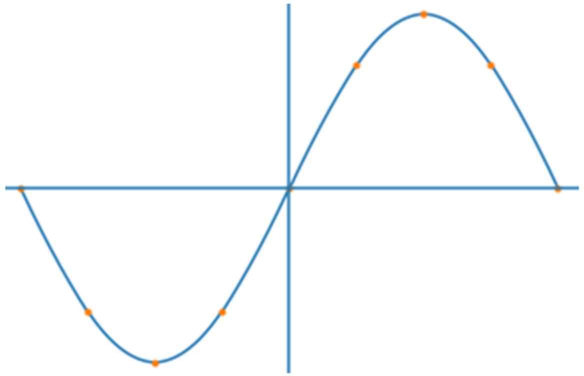
Optimizations



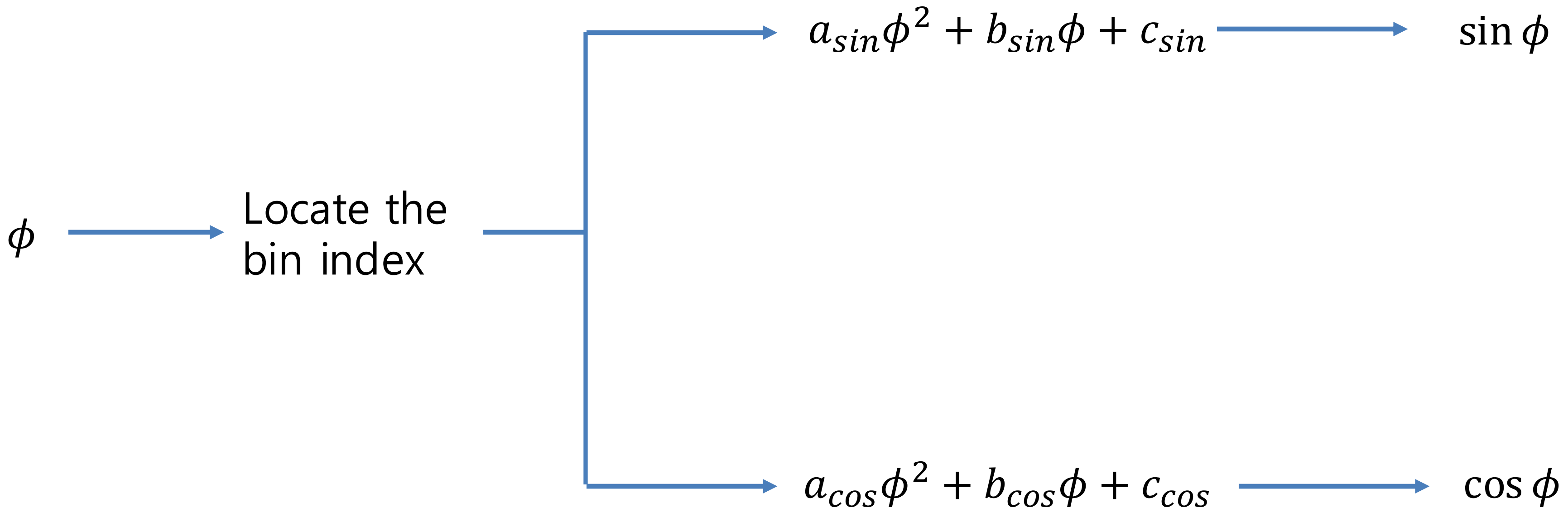
$p_x = p_T \times cos\phi$

$p_y = p_T \times sin\phi$

Projection



[Example of 2nd Order Polynomial Interpolation]



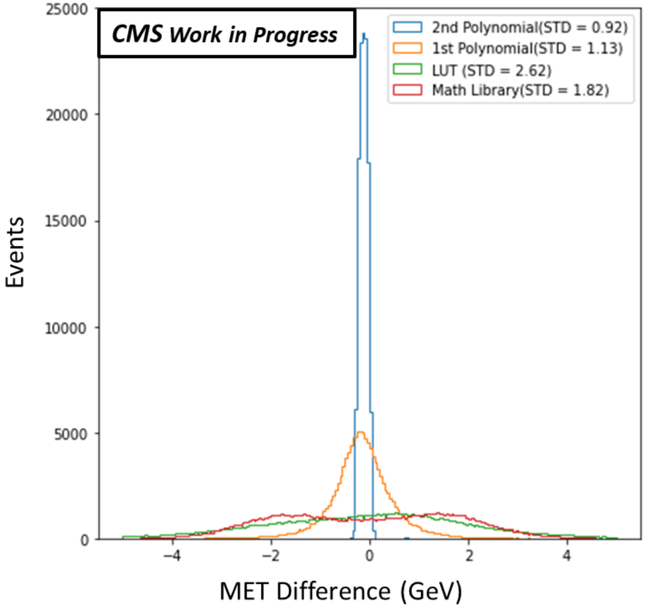
ϕ

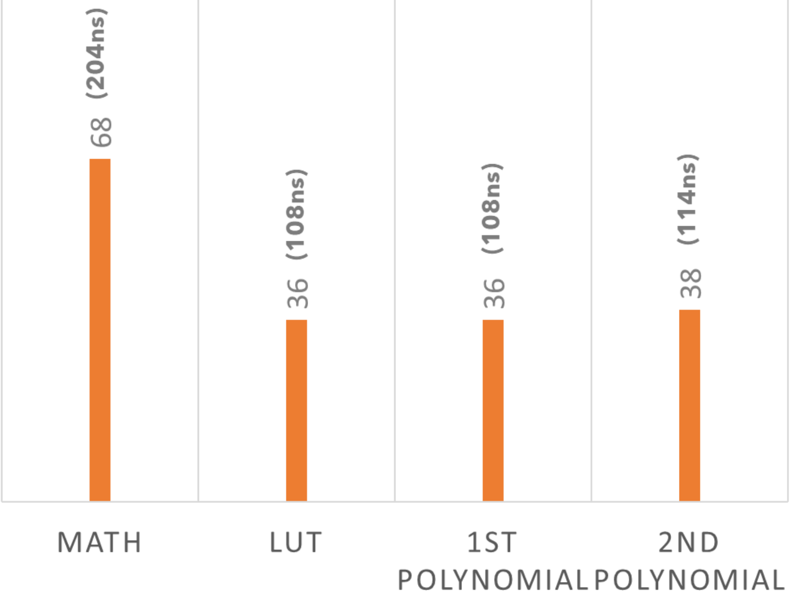
Locate the bin index

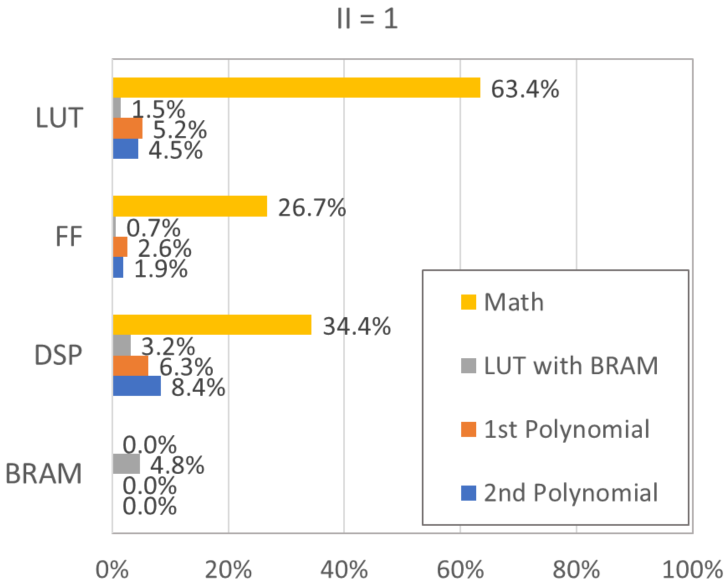
$a_{sin}\phi^2 + b_{sin}\phi + c_{sin}$ → $\sin\phi$

$a_{cos}\phi^2 + b_{cos}\phi + c_{cos}$ → $\cos\phi$

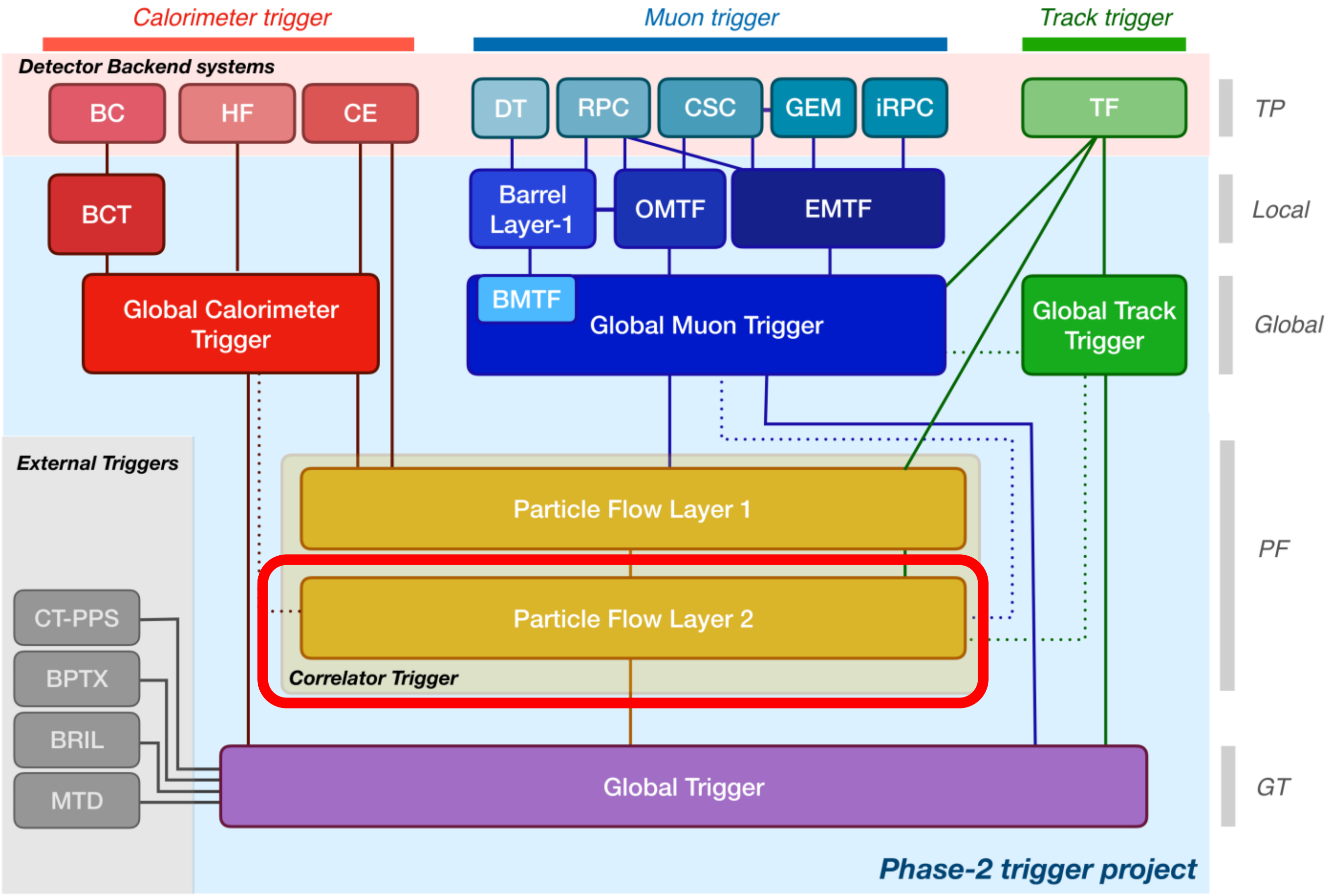
Optimizations

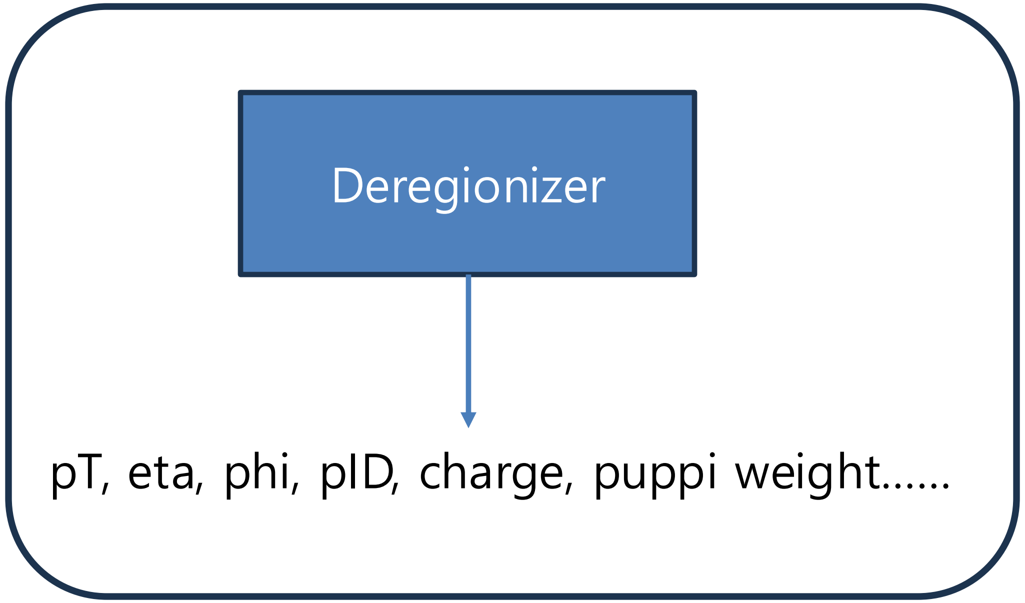






Implementation

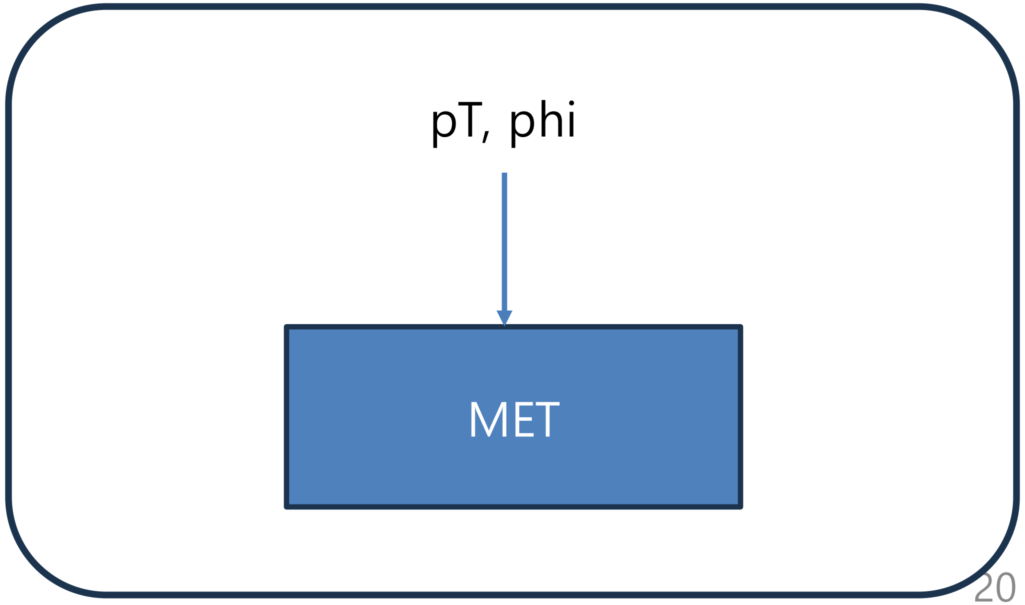




Deregionizer

pT, eta, phi, pID, charge, puppi weight……

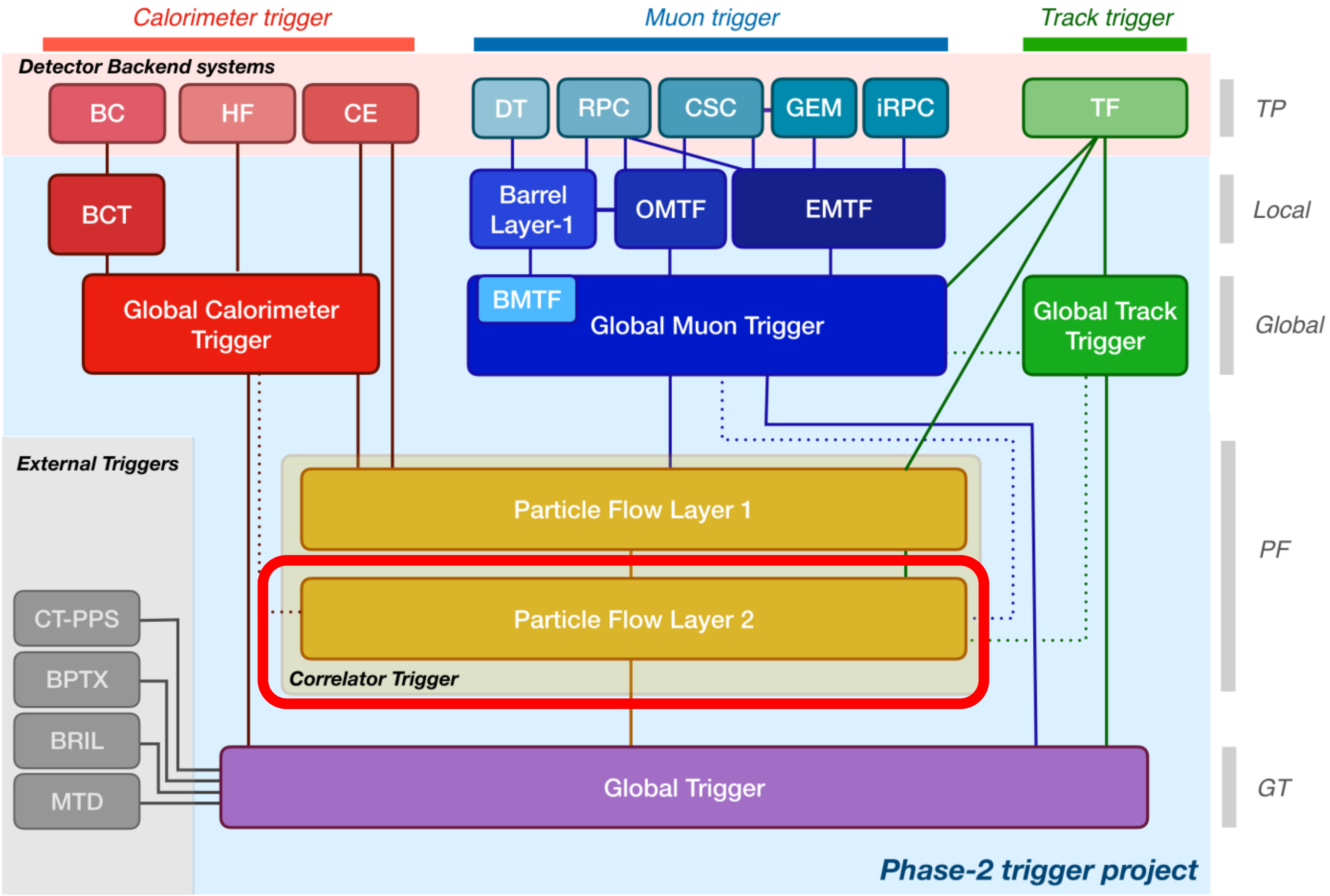
How can we connect them?

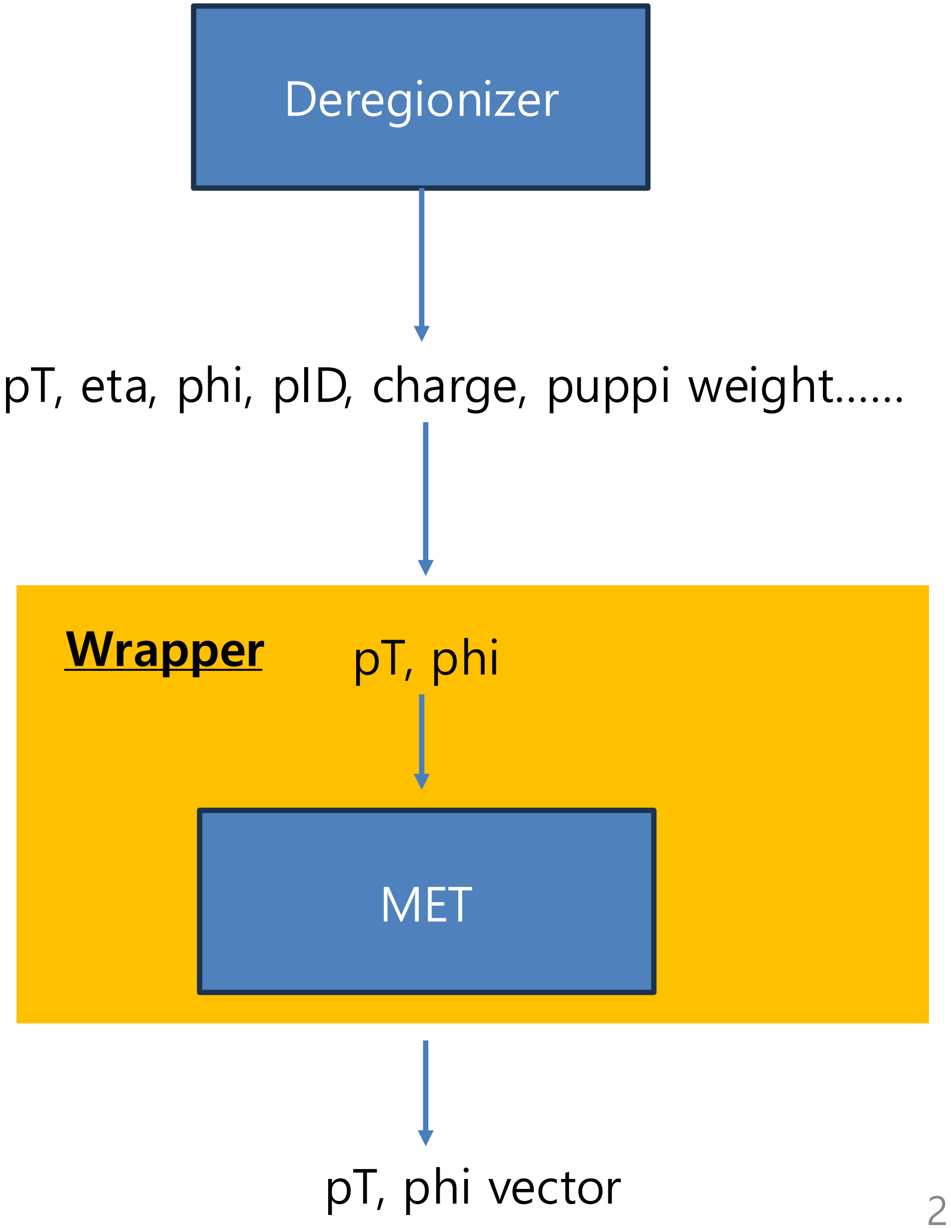


pT, phi

MET

Implementation





Deregionizer

pT, eta, phi, pID, charge, puppi weight……

Wrapper

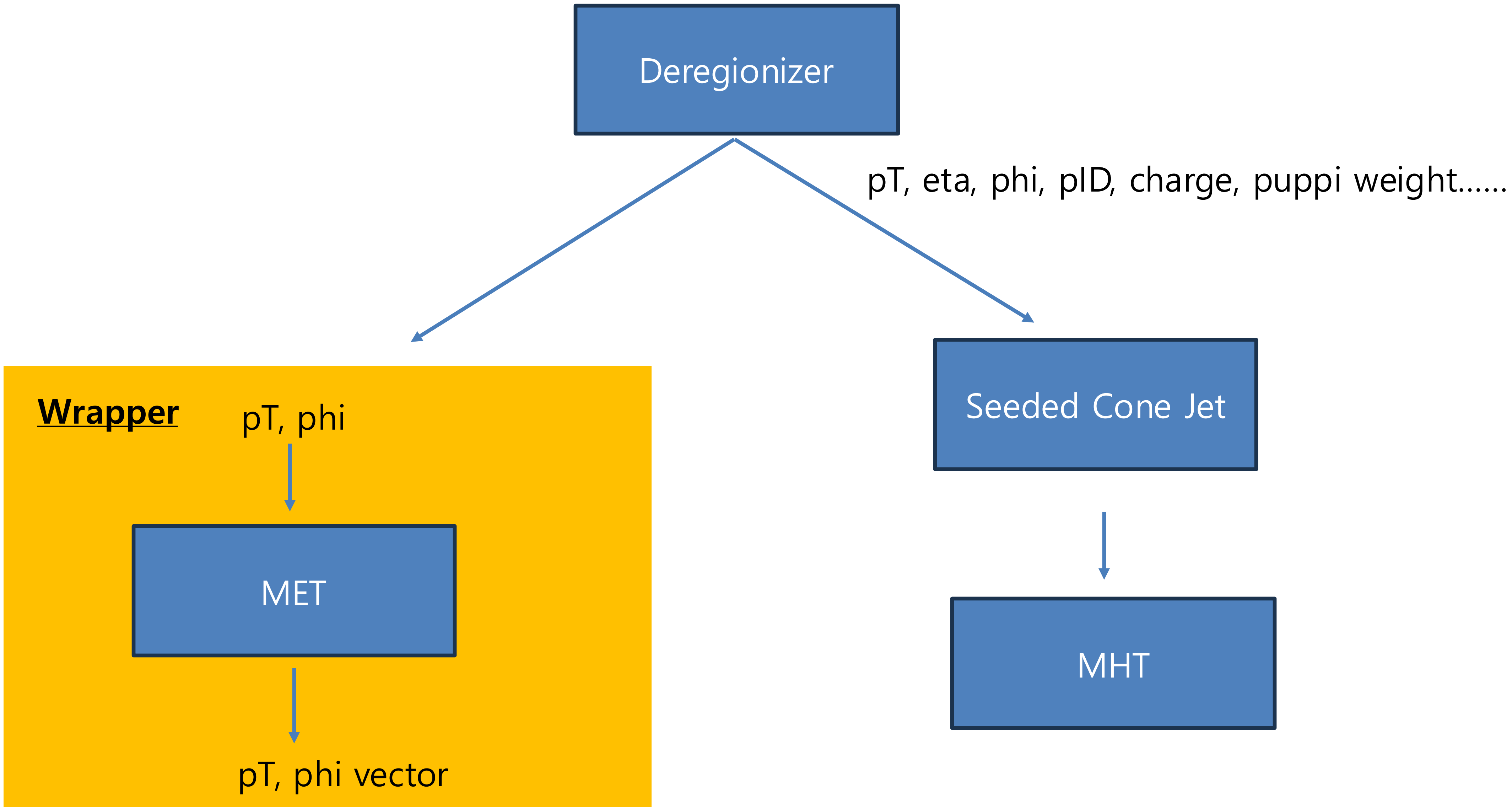
pT, phi

MET

pT, phi vector

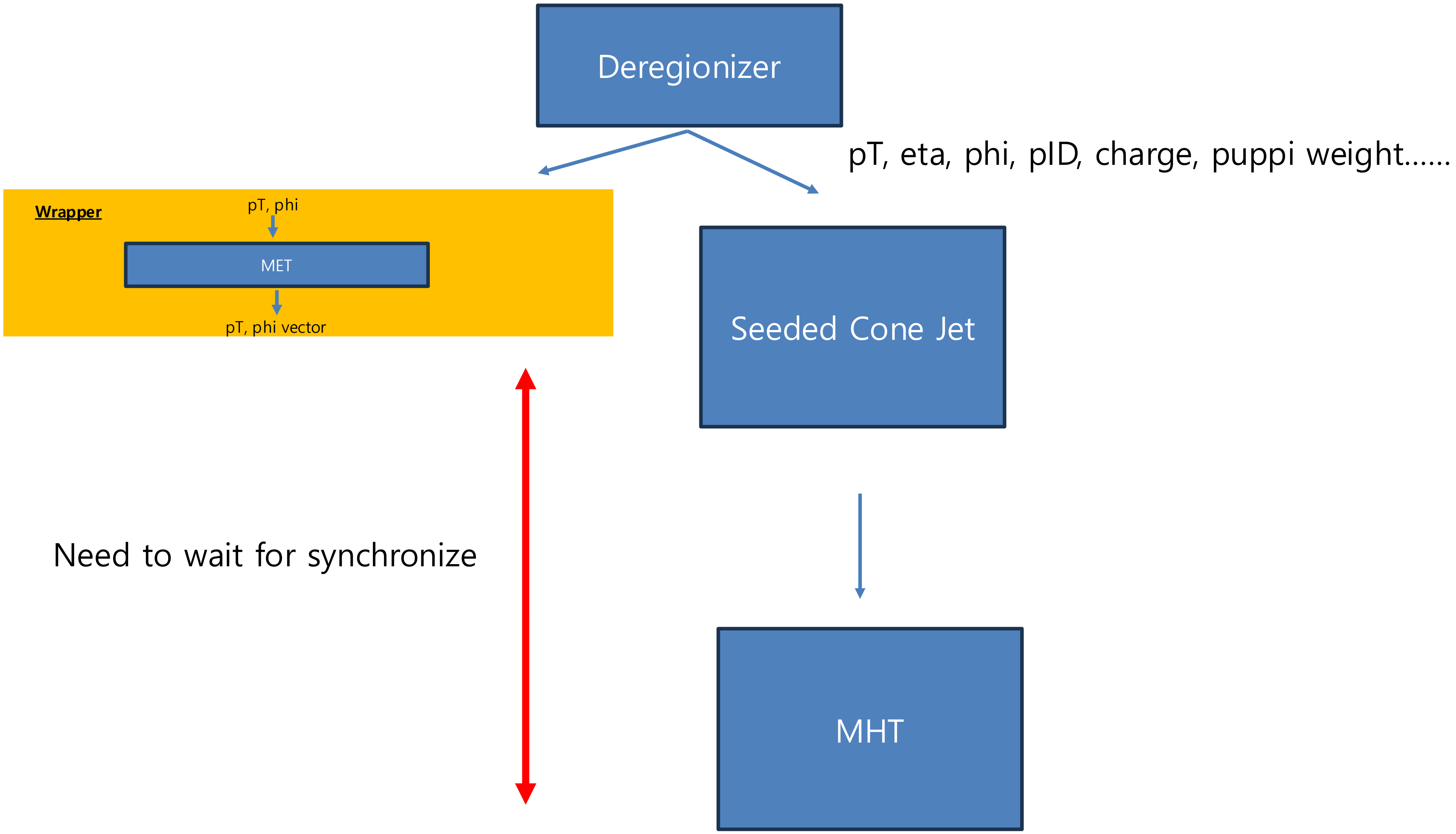
Implementation

What if we want to put other algorithms together?



Implementation

What if we want to put other algorithms together?



Deregionizer

pT, eta, phi, pID, charge, puppi weight……

Wrapper

pT, phi

MET

pT, phi vector

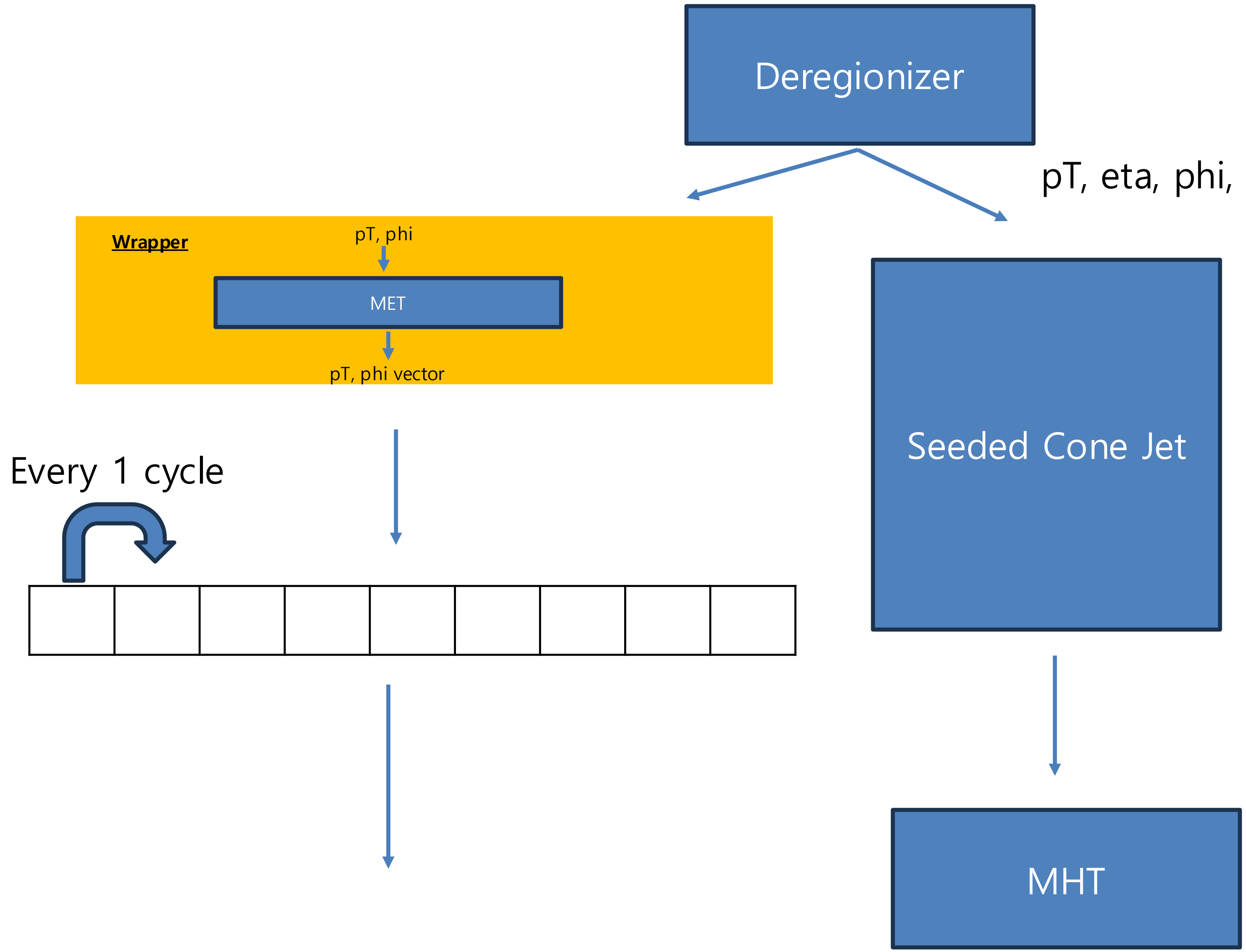
Seeded Cone Jet

MHT

Need to wait for synchronize

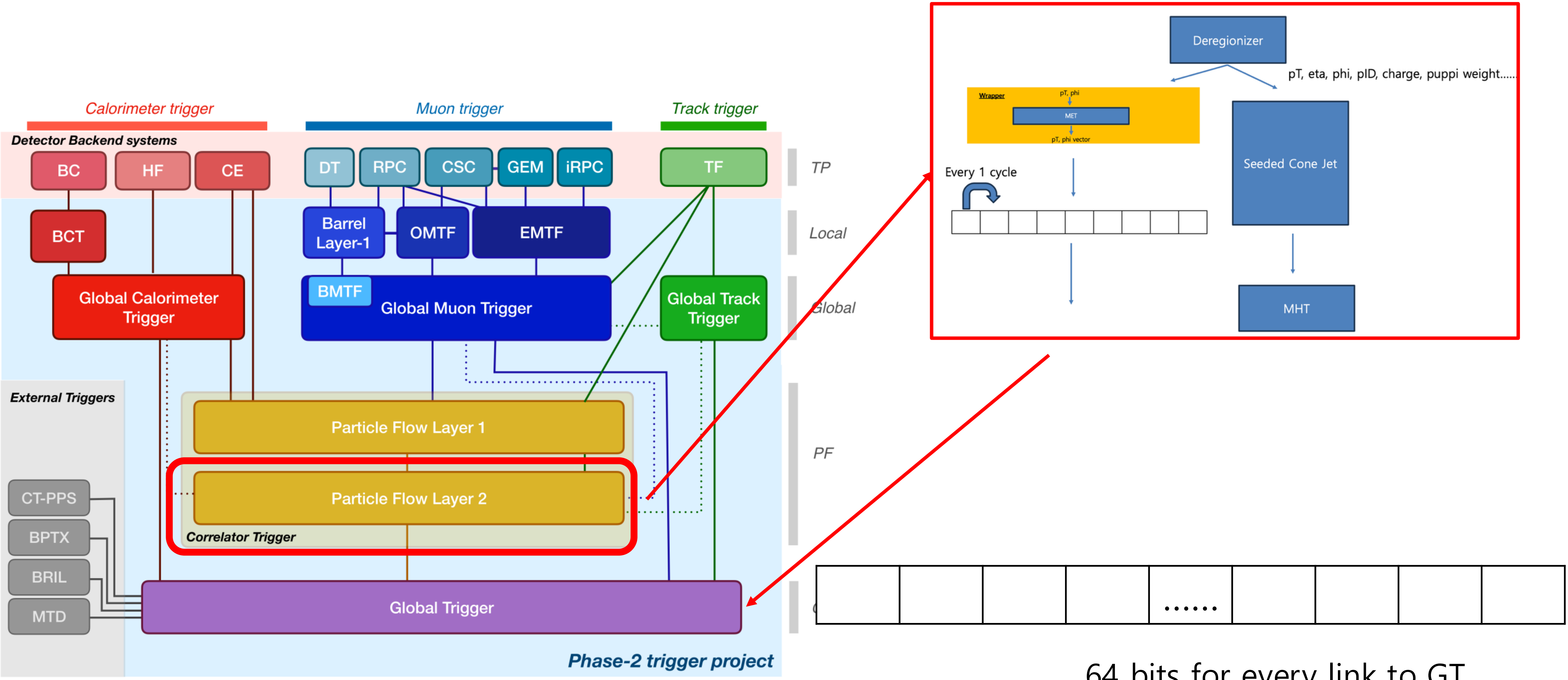
Implementation

What if we want to put other algorithms together?



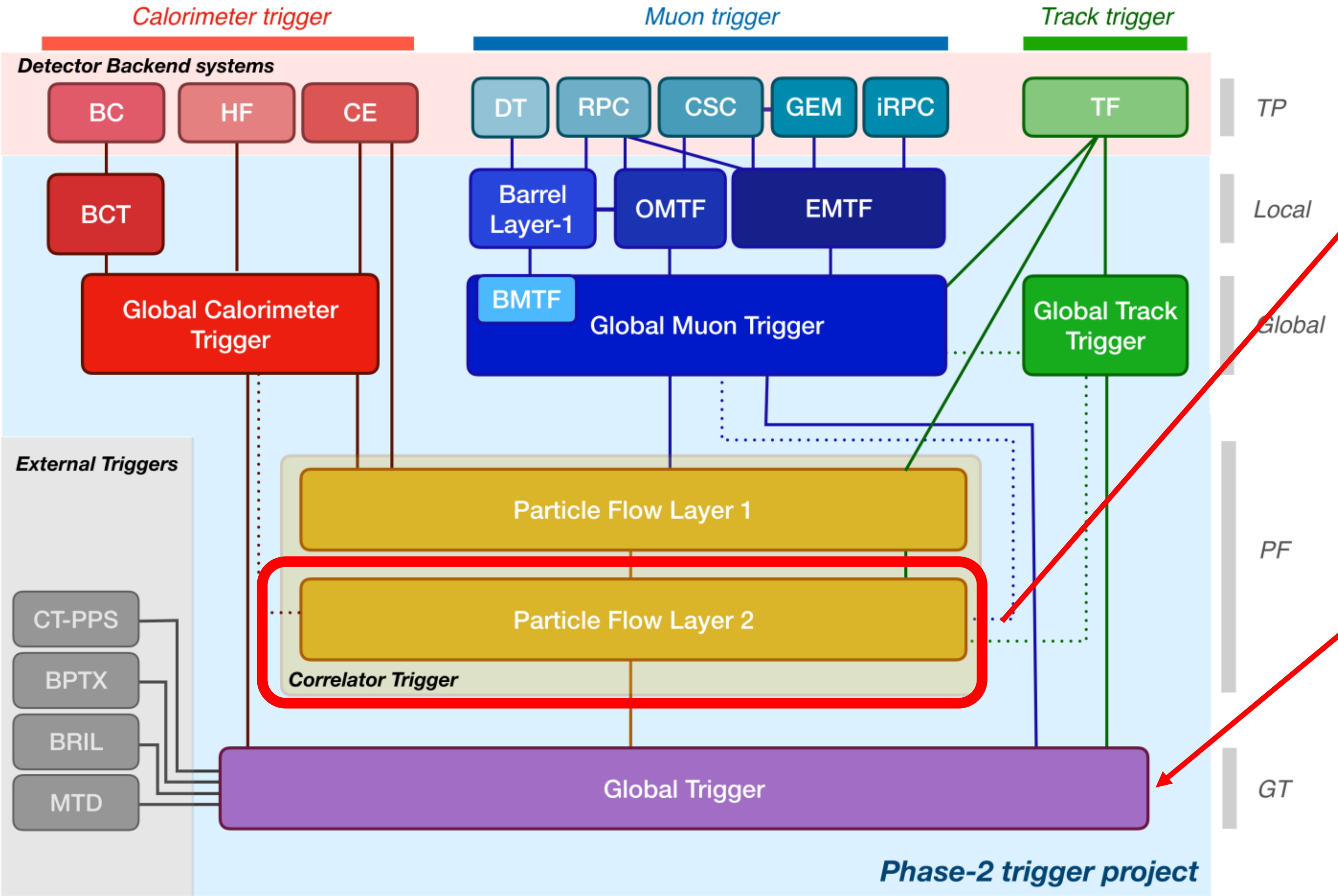
Implementation

CTL 2

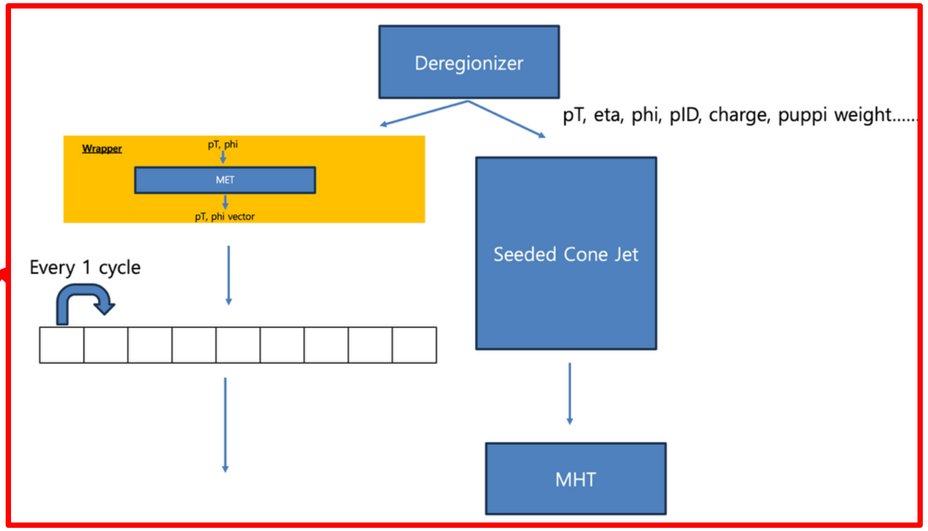


64 bits for every link to GT, sent for 54 cycles

Implementation



CTL 2

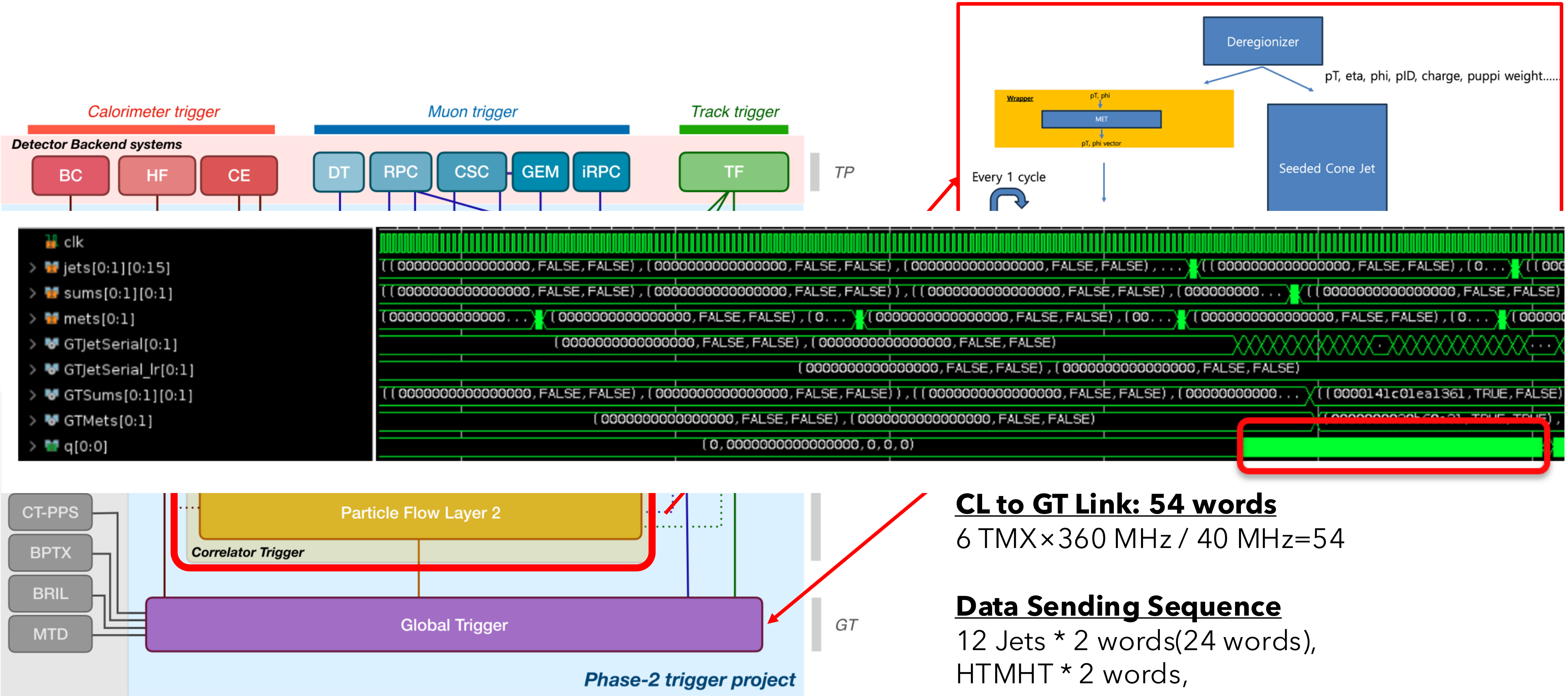


CL to GT Link: 54 words
6 TMX×360 MHz / 40 MHz=54

Data Sending Sequence
12 Jets * 2 words(24 words),
HTMHT * 2 words,
12 Jets with Large radius * 2 words(24 words),
HTMHT with Large radius * 2 words,
MET * 2 words

Implementation

CTL 2



```
clk
jets[0:1][0:15]        ((0000000000000000,FALSE,FALSE),(0000000000000000,FALSE,FALSE),(0000000000000000,FALSE,FALSE),...  ((0000000000000000,FALSE,FALSE),(0...  ((000
sums[0:1][0:1]         ((0000000000000000,FALSE,FALSE),(0000000000000000,FALSE,FALSE)),((0000000000000000,FALSE,FALSE),(000000000...  ((0000000000000000,FALSE,FALSE)
mets[0:1]              (0000000000000000...  (0000000000000000,FALSE,FALSE),(0...  (0000000000000000,FALSE,FALSE),(00...  (0000000000000000,FALSE,FALSE),(0...  (00000
GTJetSerial[0:1]       (0000000000000000,FALSE,FALSE),(0000000000000000,FALSE,FALSE)
GTJetSerial_lr[0:1]    (0000000000000000,FALSE,FALSE),(0000000000000000,FALSE,FALSE)
GTSums[0:1][0:1]       ((0000000000000000,FALSE,FALSE),(0000000000000000,FALSE,FALSE)),((0000000000000000,FALSE,FALSE),(00000000000...  ((0000141c01ea1361,TRUE,FALSE)
GTMets[0:1]            (0000000000000000,FALSE,FALSE),(0000000000000000,FALSE,FALSE)
q[0:0]                 (0,0000000000000000,0,0,0)
```

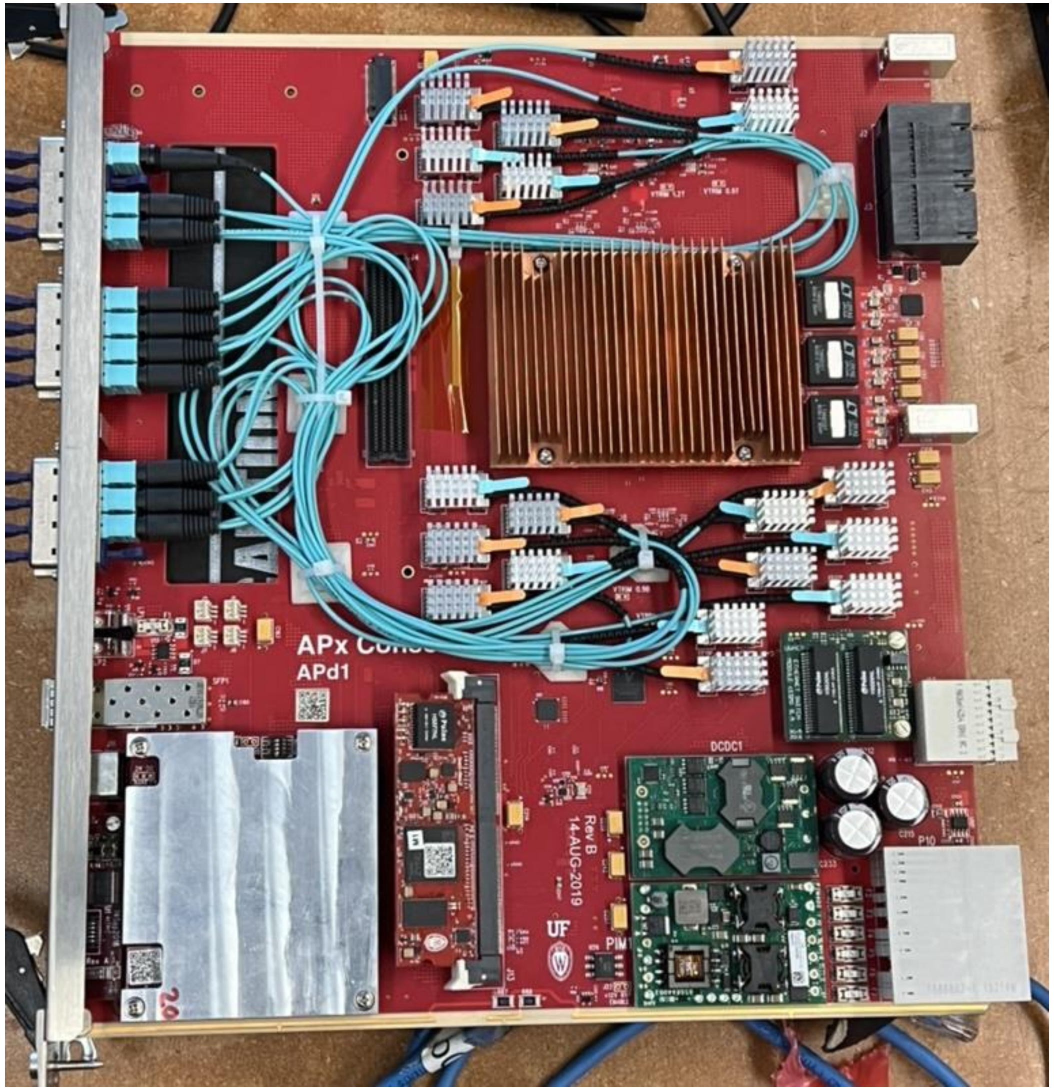
CL to GT Link: 54 words

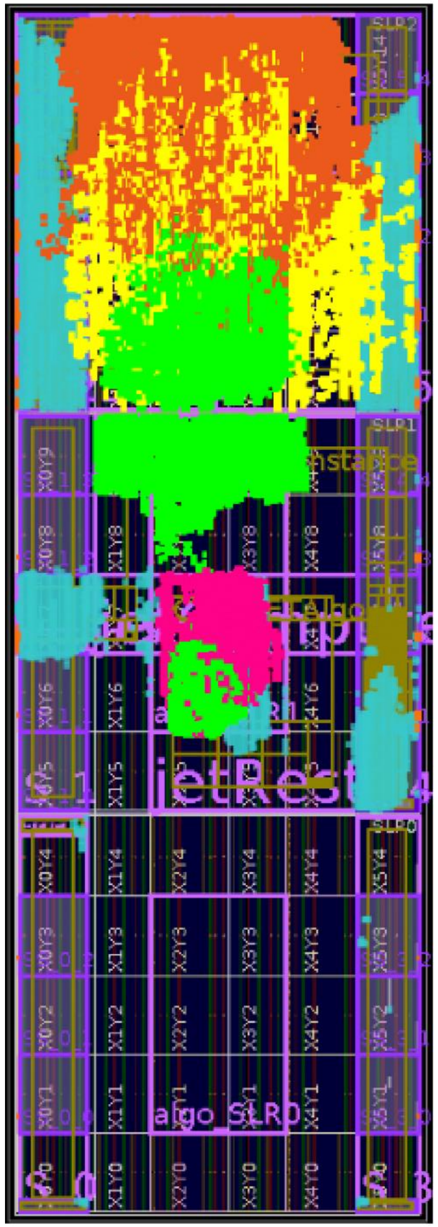
6 TMX×360 MHz / 40 MHz=54

Data Sending Sequence

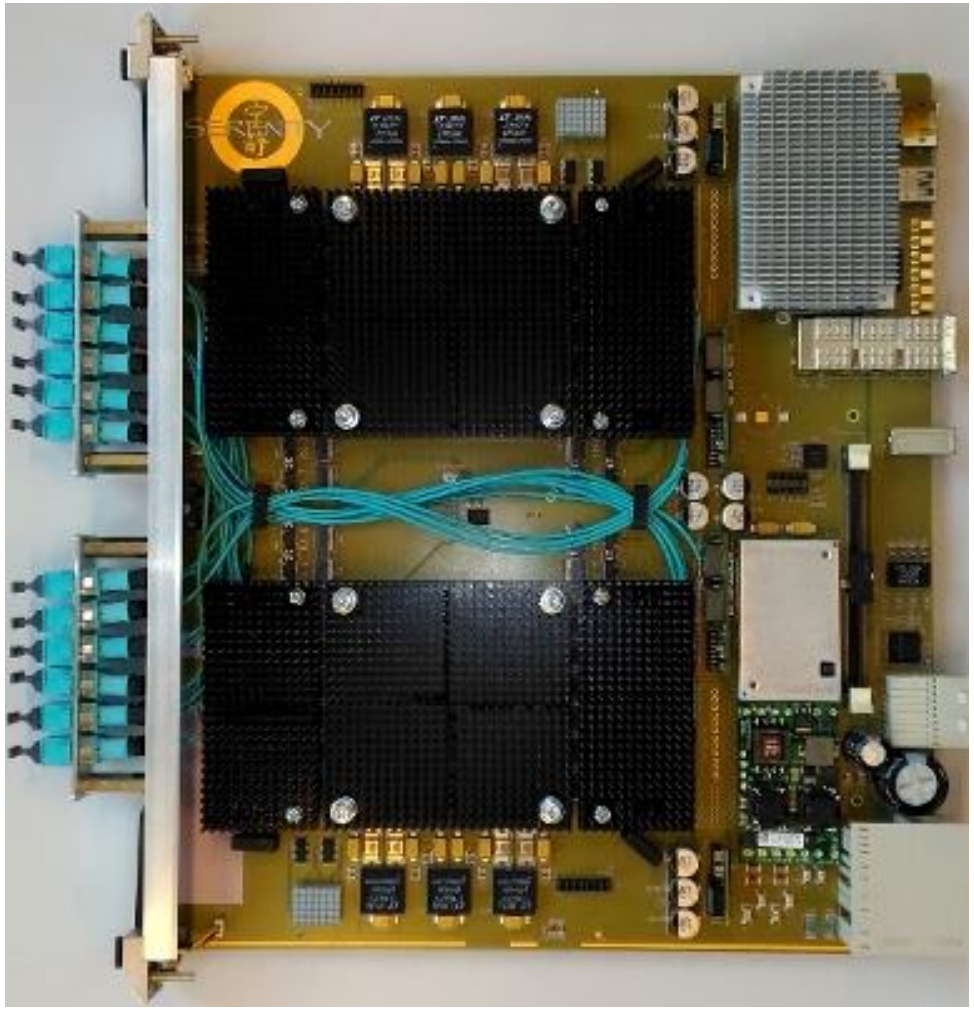
12 Jets * 2 words(24 words),
HTMHT * 2 words,
12 Jets with Large radius * 2 words(24 words),
HTMHT with Large radius * 2 words,
MET * 2 words

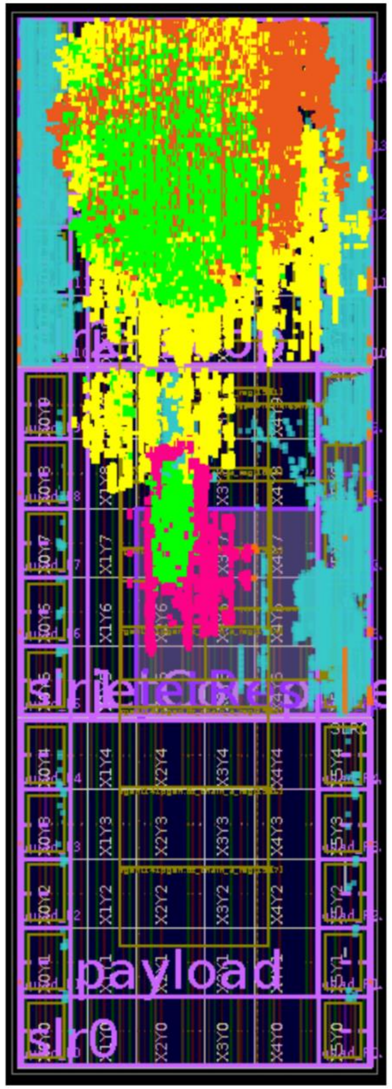
Implementation





Implementation





Thank You!

Useful Links

- https://khu.dcollection.net/srch/srchDetail/200000847818
- https://cds.cern.ch/record/2932976?ln=en
- https://indico.cern.ch/event/1498722/