(Improving) HLT using SONIC

Maria Acosta, Yongbin Feng, Lindsey Gray, Burt Holzman, Kevin Pedro, Nhan Tran (Fermilab)

Phil Harris, Jeff Krupa, Patrick McCormack, Simon Rothman (MIT)

Mia Liu, Dmitry Kondratyev, Stefan Piperov, Yao Yao (Purdue)

Javier Duarte (UCSD)

Kelci Mohrman, Philip Chang (Florida)

CMS Machine Learning Forum

February 6th, 2024

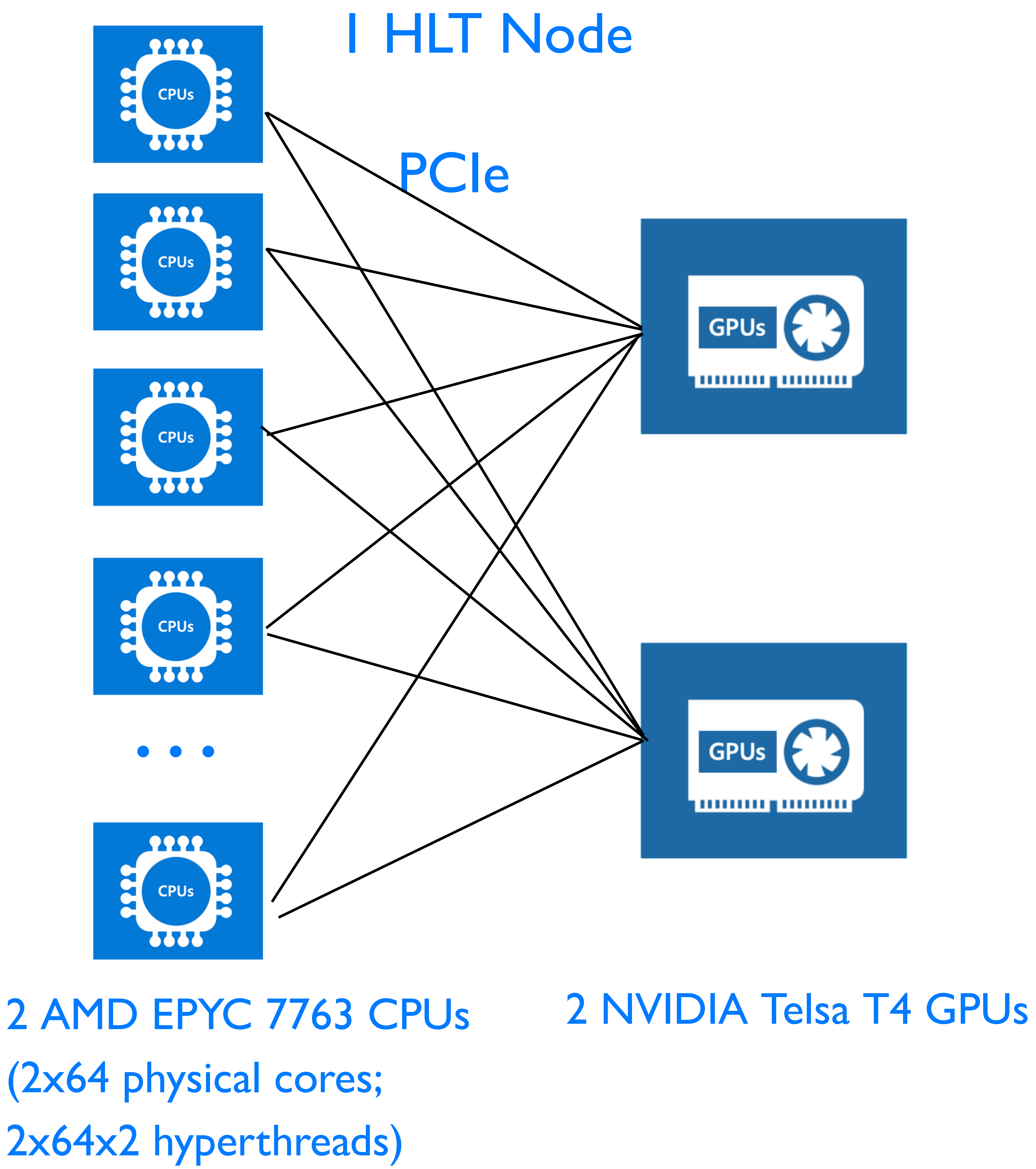
(Brief) History

- Most of the SONIC-HLT studies included here were done in 2021-2022, before the beginning of Run-3 data taking:
  - At that time we already had a demo of running the HLT workflow with Patatrack (and some ML algorithm) as a service on 2018 EphemeralHLTPhysics dataset, and we found this could increase the GPU utilization and potentially save GPU resources while getting consistent results as "direct inference"
  - Results reported and discussed in the S&C Blueprint meeting on March 9th, 2022: Indico, Slides
  - Then efforts on HLT were paused for a while, since the Run-3 configuration was supposed to be frozen. Also received suggestion to focus on the offline studies first
  - Main focus was on MLG-23-001 "Portable Acceleration of CMS Computing Workflows with Coprocessors as a Service", with studies on ML inference for offline acceleration demonstration. Paper expected to be submitted soon.
  - Plan to shift the focus back to HLT and continue the developments and tests

Goal

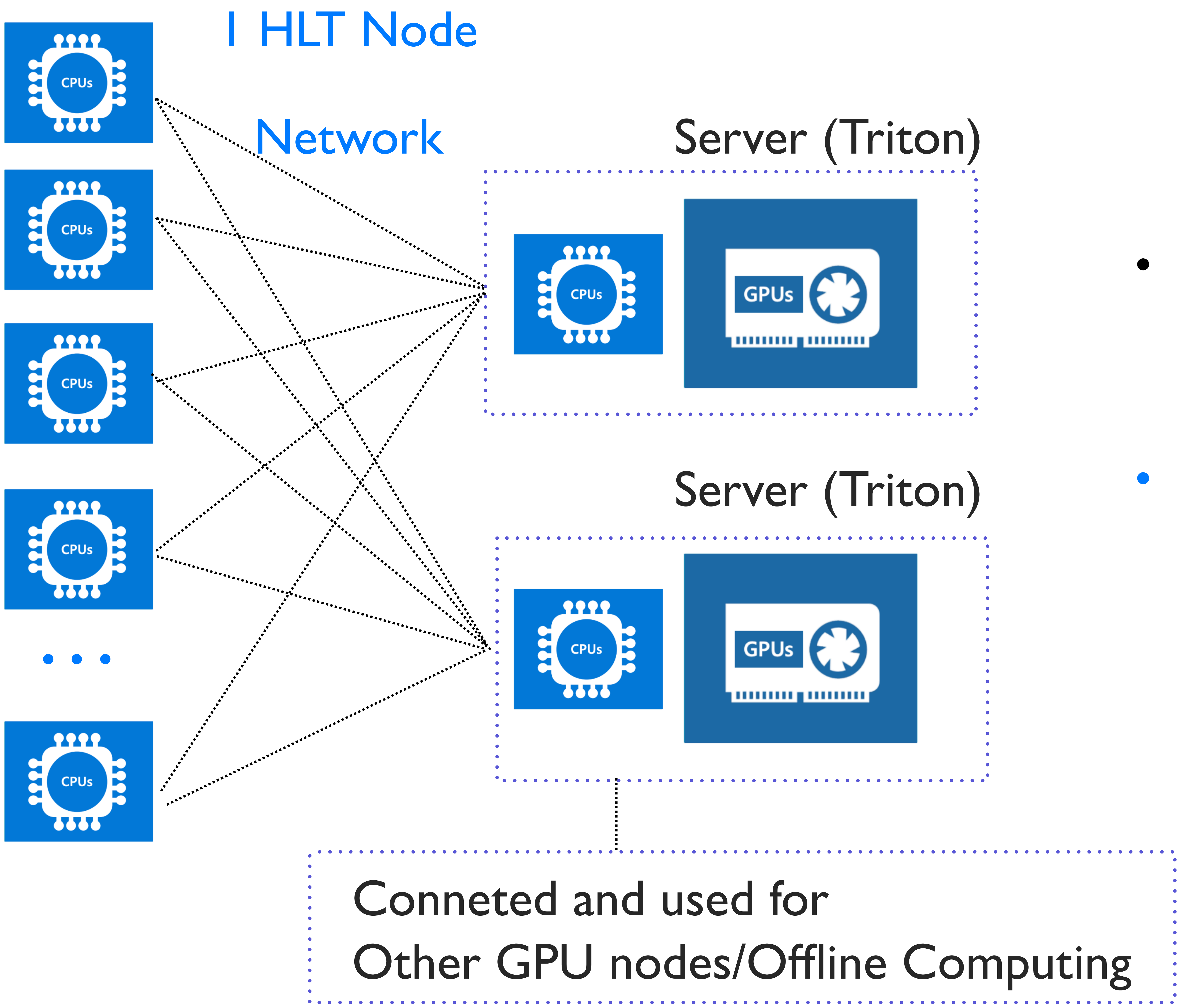
- Despite being primarily discussed in the Machine Learning Group, SONIC is general and can be applied to accelerate classical domain algorithms as well, for online/offline computing and different workflows
- We (SONIC team) would like to contribute to HLT developments, especially on the Next Generation Trigger study of “Towards a distributed HLT architecture”.
  - We have accumulated plenty of technologies and experience from the offline studies with machine learning algorithms
  - We have a workflow that can run the HLT with Patatrack as a service and have tested against other workflows
  - Link to the previous code and workflow: here. (Some of these are not synchronized with the latest developments, which is expected to be done soon.)
  - SONIC@HLT can be one candidate for the next MLG paper MLG-24-00X (online) as a continuation of MLG-23-001 (offline)
- Slide 4 - 13 are mostly recycling the material two years ago. Plan to continue the developments now.

Run-3 HLT Farm Setup



- Heterogenous system:
  - 2 x AMD EPYC 7763 (2x64 physical cores/2x64x2 hyperthreads) directly connected to 2 NVIDIA Tesla T4
- Offloads Patatrack + ECAL Multifits + HCAL MAHI Reco from CPU to GPU, reducing the HLT CPU processing time and increasing throughput by around 25-30%

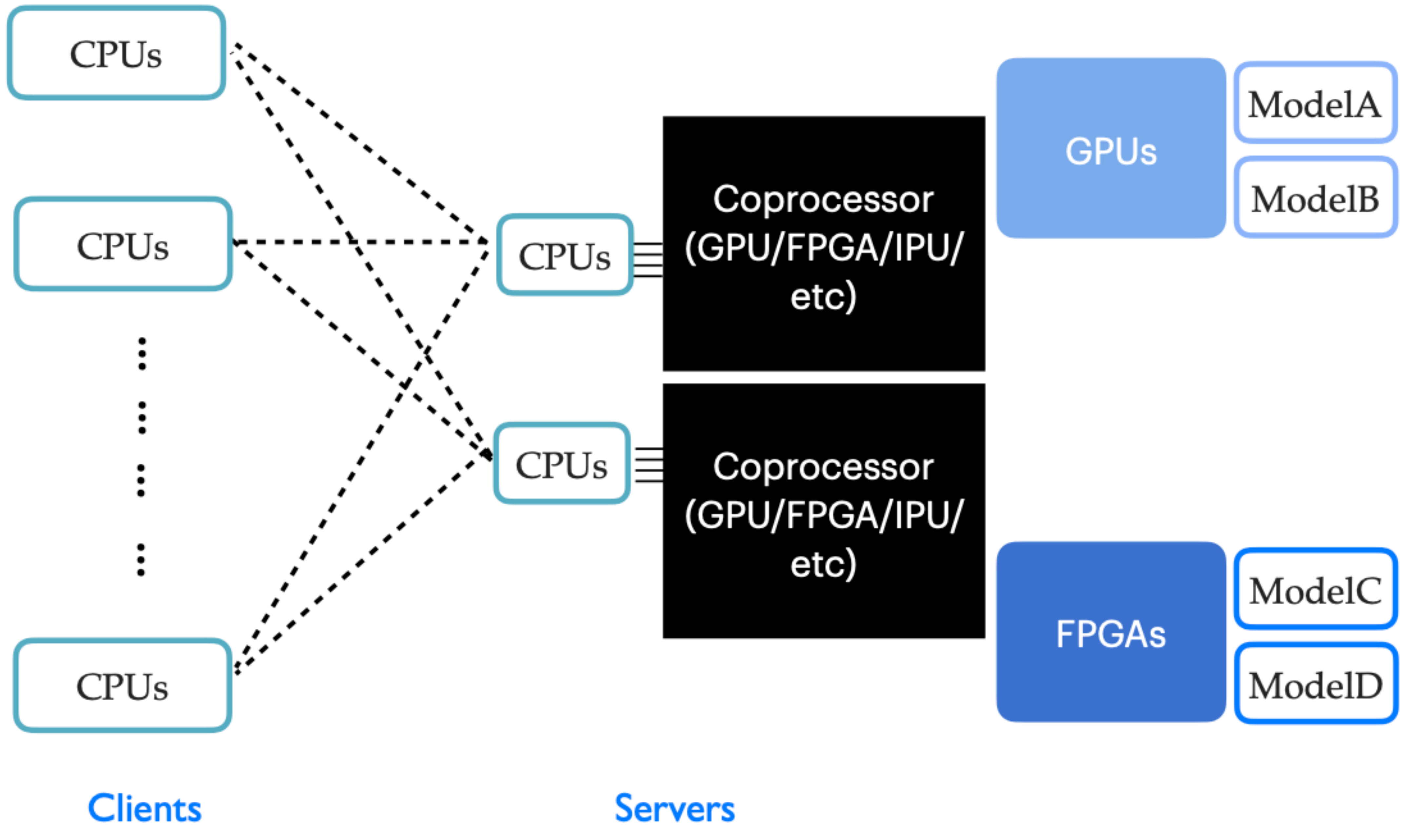
If with SONIC



- Build servers using the HLT GPUs and run Patrack + ECAL Multifits + HCAL MAHI Reco as a service on the HLT
- One GPU server can serve more than 1 AMD EPYC 7763 CPU; the extra saved GPUs can be used to serve other HLT nodes or used for e.g., Offline computing
  - More efficient utilizations of existing computing resources

Benefits with SONIC

- Many benefits of running inference aaS with SONIC, e.g.:
  - One coprocessor can serve many CPU clients; one CPU client can communicate with multiple coprocessors; easy to change - more degrees of freedom efficient and sufficient utilization of coprocessors
  - Factorize the ML and Coprocessor framework (TensorFlow, PyTorch, ONNX, Scikit-Learn, XGBoost, CUDA, Alpaka etc) out of clients (CMSSW), which only needs to handle the I/O conversions on the client side - easy support for different (ML) frameworks, models
  - Simple support for different coprocessors. No need to rewrite algorithms in coprocessor-specific languages - Easily Portable
  - Access to remote coprocessor resources.



What we've done & tested with HLT

- Ported domain algorithm Patatrack code into Triton Custom backend and can run Patatrack as-a-service
  - Start with pixeltrack-standalone and build the Patatrack custom backend. Recipe here. Synced with CMSSW_12_3_0_pre4. Outputs (e.g., tracks and vertices) are almost identical
  - Test at Purdue on the 2018 EphemeralHLTPhysics dataset: one GPU server can serve at least two AMD EPYC 7702 (2x64 physical cores) without any performance drop

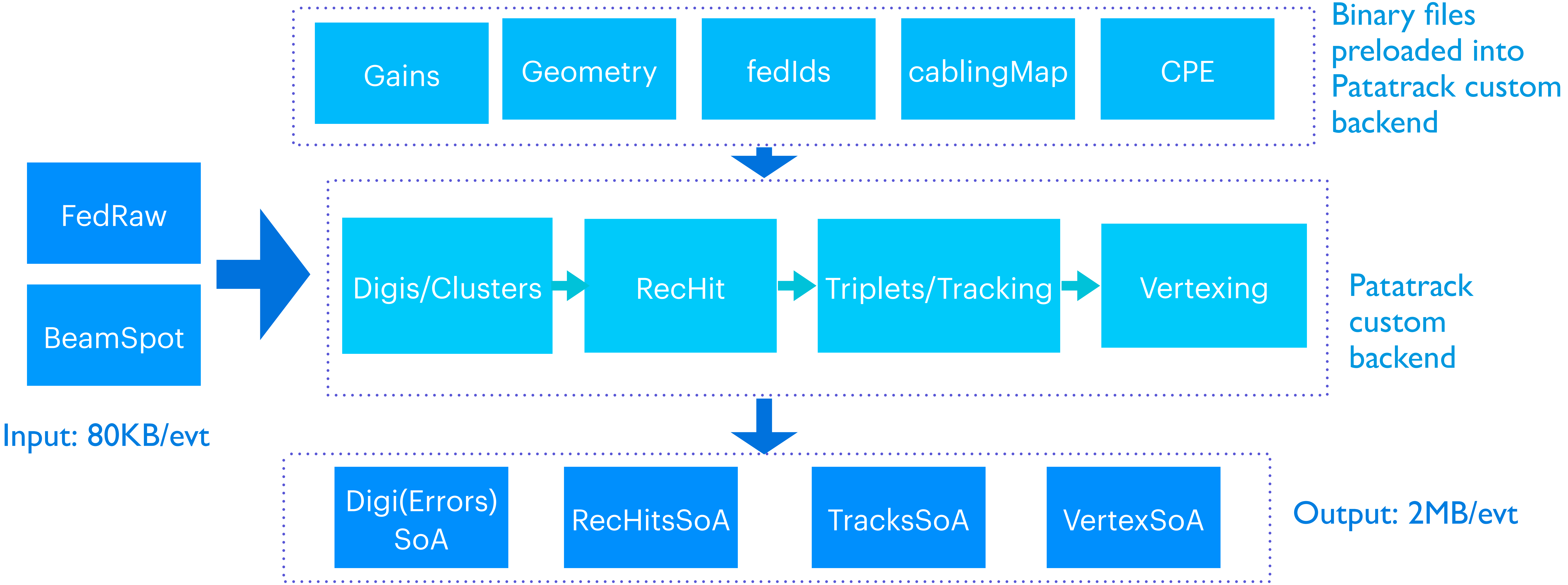
- Developed ML-based HCAL reconstruction algorithm FACILE and run it as-a-service.
  - A candidate replacement of HCAL MAHI reco
  - A small size Tensorflow model. Easy to train and deploy. Better or equivalent performance than MAHI
  - Test at Purdue on the Run-3 ttbar RelVar dataset: one GPU server can serve at least two AMD EPYC 7702 (2x64 physical cores) without any performance drop

Patatrack Performance

Latency [ms]	CPU	GPU	Reduction
Total	340	265	75
Patatrack	42	11.1	30
MAHI	37	1.6	35
ECAL	14	1.8	12

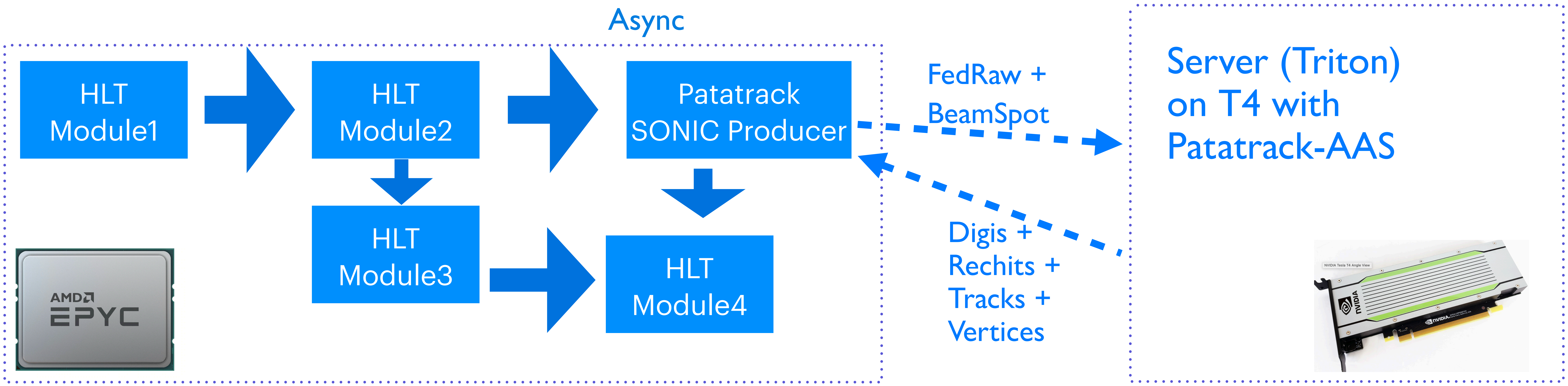
- Benchmark the Patatrack on GPU performance at Purdue (2xAMD7703 + 1 Tesla T4 GPU), using the 2018 EphemeralHLTPhysics dataset and the HLT config from the timing twiki
  - Patatrack + MAHI + ECAL directly on GPU together reduces the CPU latency by around 23%;
  - Patatrack alone reduces the latency by around 9%
- Running on 2018 EphemeralHLTPhysics dataset, directly running on GPU tend to require a lot of GPU memory (more than around 1GB per 4-thread job).
  - Can only do 16x4 jobs at the moment because of the limited memory on T4 (16GB)
  - GPU utilization is around 50-60%. Seems not sufficiently utilized

Patatrack Custom Backend



- Start from pixeltrack-standalone to avoid dealing with the enormous amount of libraries in CMSSW. Full Recipe here
  - Reuse more than 90% of the code in pixeltrack-standalone; plus some extra Triton backend code to control the IOs
  - For longer term, still an option to explore compiling together with CMSSW to be more maintainable

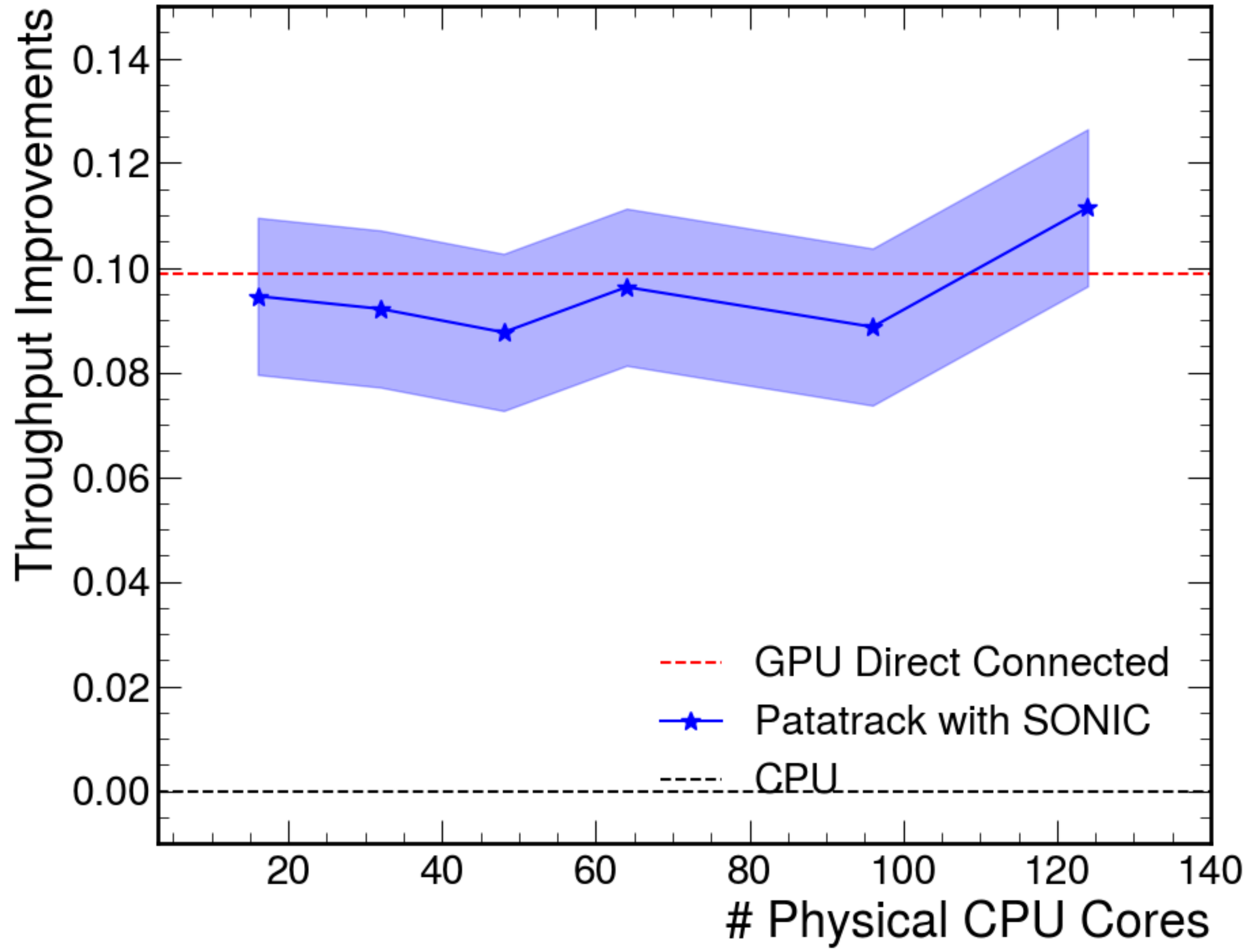
Patatrack as-a-Service



- A Patatrack SONIC Producer to handle the IOs on the client (CMSSW) side. Running asynchronously.
- Inputs are FedRaw data and beamspot information (<100KB/evt) .
- Outputs are Digis + Rechits + Tracks + Vertices. All zero-suppressed to reduce the output size. About 2MB/evt after zero suppression. (Without zero-suppreson the output would be 8MB/s)
- Comparing the AAS with directly running Patatrack in CMSSW - results (tracks and vertices) and output trigger flags are almost identical

Patatrack as-a-Service Performance

Throughput [evts/s]	1 thread	10 threads
Patatrack	660	930
+ CPU/GPU Transfer	440	870
+ zero suppression	400	820



- Table shows the results of testing Patatrack standalone
- The throughput with PatatrackAAS from Triton Perf Client is around 400-500 evts/s
- Running exclusiving Patatrack-aaS on one server. The throughout improvements with PatatrackAAS is expected - around 10%. One GPU running PatatrackAAS can serve at least 124 physical CPU cores
  - Need about 3 CPU cores for the server; GPU utilization is around 60% with 124 CPU cores.
- One Patatrack-AAS server with one T4 GPU can probably serve up to ~180-200 CPU cores. The remaining GPUs can be saved and used for other purposes

FACILE Performance On Run3 RelVar

Latency	CPU	GPU	Reduction	AAS (Patatrack + Facile) + ECalGPU	Reduction
Total	1560	1450	120		
Patatrack	80	32	50	31	50
MAHI	54	2.0	52	11	43
ECAL	27	3.0	24	3.0	24

- With ttbar relval samples, the Run-3 workflow. Running 31 4-thread jobs at Purdue (124 physical cores) with Patatrack + FACILE aaS and ECALGPU directly connected.
- The latency of these three processes are pretty consistent. For ttbar sample, one GPU serving Patatrack + Facile as-a-Service can serve at least 124 physical cores

Summary

- Ported Patatrack code into Triton Custom backend and can run Patatrack as-a-service
  - ✤ Test at Purdue on the 2018 EphemeralHLTPhysics dataset: one GPU server running PatatrackAAS can serve at least two AMD EPYC 7702 (2x64 physical cores) without any performance decrease.
  - ✤ Can port MAHI and ECAL GPU algorithms into AAS as well if needed
- Developed DNN-based HCAL reconstruction algorithm FACILE and run it as-a-service.
  - ✤ Test at Purdue on the Run-3 ttbar RelVar dataset: one GPU server can serve at least two AMD EPYC 7702 (2x64 physical cores) without any performance drop
- Running HLT as-a-Service can make more efficient usage of the HLT resources and allow us to explore wider usage

Plans

- Plan to continue the HLT studies with SONIC for optimizing resources at HLT. Work on e.g.:
  - ✤ Sync the code with the latest setup; compile Alpaka into the backend, etc
  - ✤ Work on a Centos/Alma-9 based server and explore using code and libraries directly from CMSSW
  - ✤ Automate the custom backend server creation for any GPU algorithm in CMSSW
- SONIC is not limited to ML. We would like to contribute to the HLT developments (and the Next Generation Trigger project)

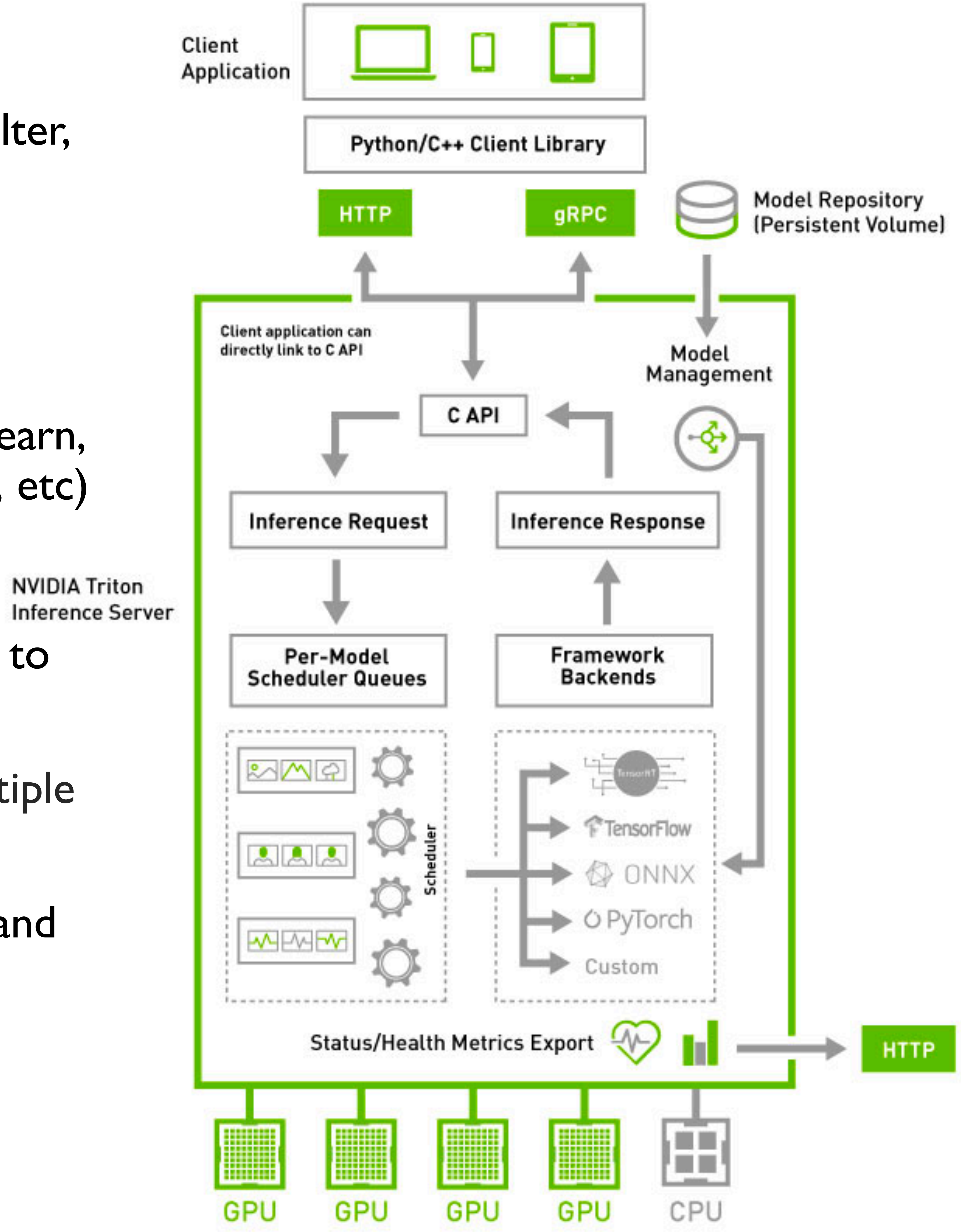
3.2 Evolving the HLT into a distributed application

- goal: support independent scaling of CPU and GPU resources at HLT
- **extend CMSSW to a fully distributed application**
  - distribute **any kind of modules** across multiple jobs, without rewriting them
    - support for different CPU and GPU architectures
    - support for arbitrary network topologies
  - leverage high-speed interconnect (IB, RoCE) and shared memory
    - evaluate different approaches: **client-server**, **microservices**, …
- **plans for 2024**
  - finalise the technical details of the projects
  - hire a **doctoral student** during the second part of the year
  - work towards the first milestone: implement a client-server test application

Back Up

Overview: SONIC in CMSSW

- Client side: CMSSW
  - SonicCore (repo) + SonicTriton (repo): includes different modules (EDProducer, EDFilter, EDAnalyzer); provides synchronous and asynchronous modes for clients
- Server side: NVIDIA Triton Inference Server (webpage, repo)
  - It supports numerous ML backends (TensorFlow, TensorRT, PyTorch, ONNX, Scikit-Learn, XGBoost, etc) and custom backends for e.g., non-ML algorithms (python, cpp, CUDA, etc)
  - Many attractive features including:
    - Dynamic batching: accumulate requests from multiple events and process together to increase inference throughputs; transparent to clients
    - Concurrent model execution + Multi-GPU load balancing: one GPU can serve multiple models; one model can be served on multiple GPUs with load balancing
    - Model pipelines: model ensembles form a pipeline of some models, connect input and output tensors in between



Results on 2018 Data

Latency [ms]	CPU	GPU	Reduction
Total	340	265	75
Patatrack	42	11.1	30
MAHI	37	1.6	35
ECAL	14	1.8	12

Latency [ms]	CPU	GPU	Reduction
Total	570	440	130
Patatrack	73	17	56
HCAL (MAHI)	61	2.8	58
ECAL	23.5	2.8	20

- Run the HLT workflow with CMSSW_12_3_0_pre4 and the 2018 raw data, following the similar steps as th HLTTimingReport.
  - Left is running without hyperthreading
  - Right is running with hyperthreading
- Trigger results are basically the same between these two, with very minor differences. Latency reduction about 25%.
  - Results in the HLT node tests were 614ms and 466ms respectively.

Results on ttbar RelVal Sample

Latency	CPU	GPU	Reduction	AAS (Patatrack + Facile) + ECalGPU	Reduction
Total	1560	1450	120	1550	10(?)
Patatrack	80	32	50	31	50
MAHI	54	2.0	52	11	43
ECAL	27	3.0	24	3.0	24

- With ttbar relval samples, the Run-3 workflow, and CMSSW_12_0_1. Running 31 4-thread jobs at Purdue (128cores with hyperthreading disabled).
- Patatrack CPU and GPU produces some different trigger flags; the latency of the other modules are changed, causing the total timing to be different. But the latency of these three processes are pretty consistent. Can further check and validate this in CMSSW_12_3_0_pre4

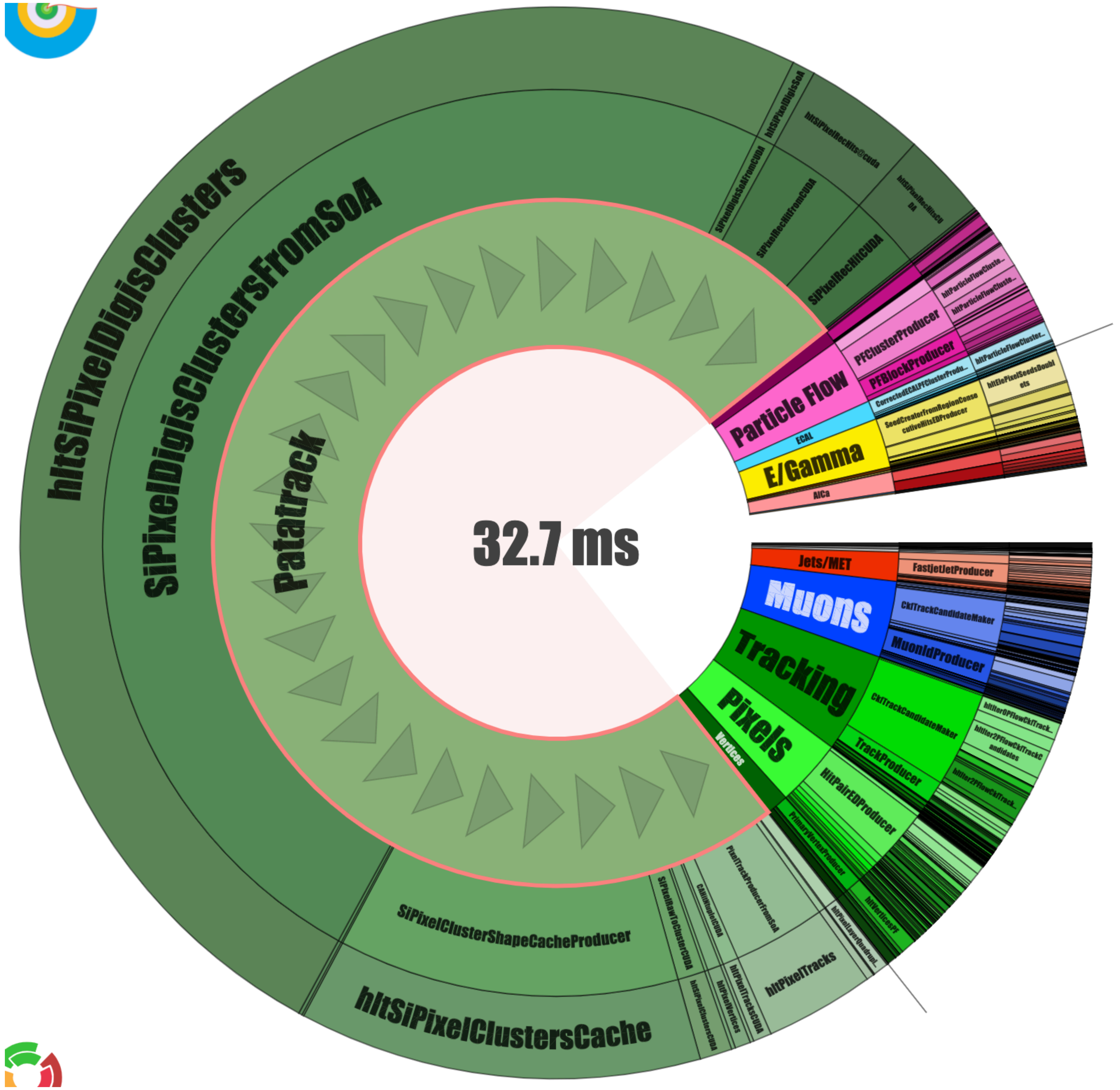
Results on ttbar RelVal Sample

GPU utilization	16 4-thread jobs	31 4-thread jobs
Patatrack	5-10%	10-20%
Patatrack + MAHI	20%	30-40%
Patatrack + MAHI + ECAL	30-40%	60%

Server CPU utilization	16 4-thread jobs	31 4-thread jobs
Patatrack	30%	60%
Patatrack + Facile	80%	160%

- With ttbar RelVar sample, running 31 4-thread jobs with Patatrack + MAHI + ECALGPU offloaded the GPU, the GPU utilization is around 60%
- The server CPU utilization is around 160% running 31 4-threaded Patatrack + FACILE jobs
- Server supports 31 4-thread jobs well, with Patatrack + Facile aaS and ECAL running on local GPU
- Study and optimize the server side CPU usage

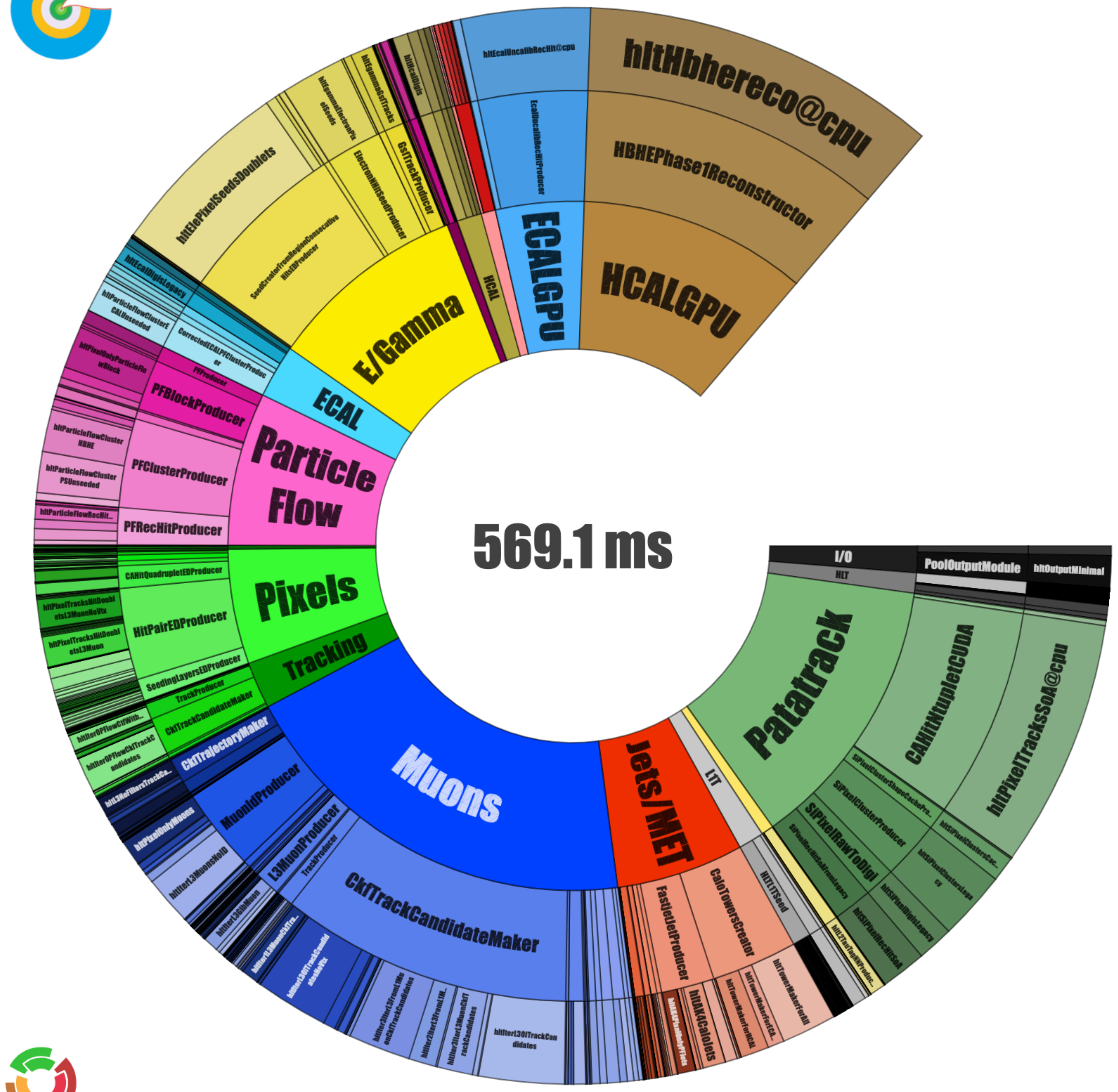
PatatrackGPU



- Out of the 33ms total latency:
  - SiPixelDigisClustersFromSoA: 21.5ms. (Convert the pixel digis and clusters to legacy format.
  - SiPixelClusterShapeCacheProducer: 5ms (also exists in the legacy CPU workflow)
  - Data transfer from GPU to host takes about 2ms in total

Results on 2018 Data: CPU

- Running on 2018 data with hyperthreading enabled



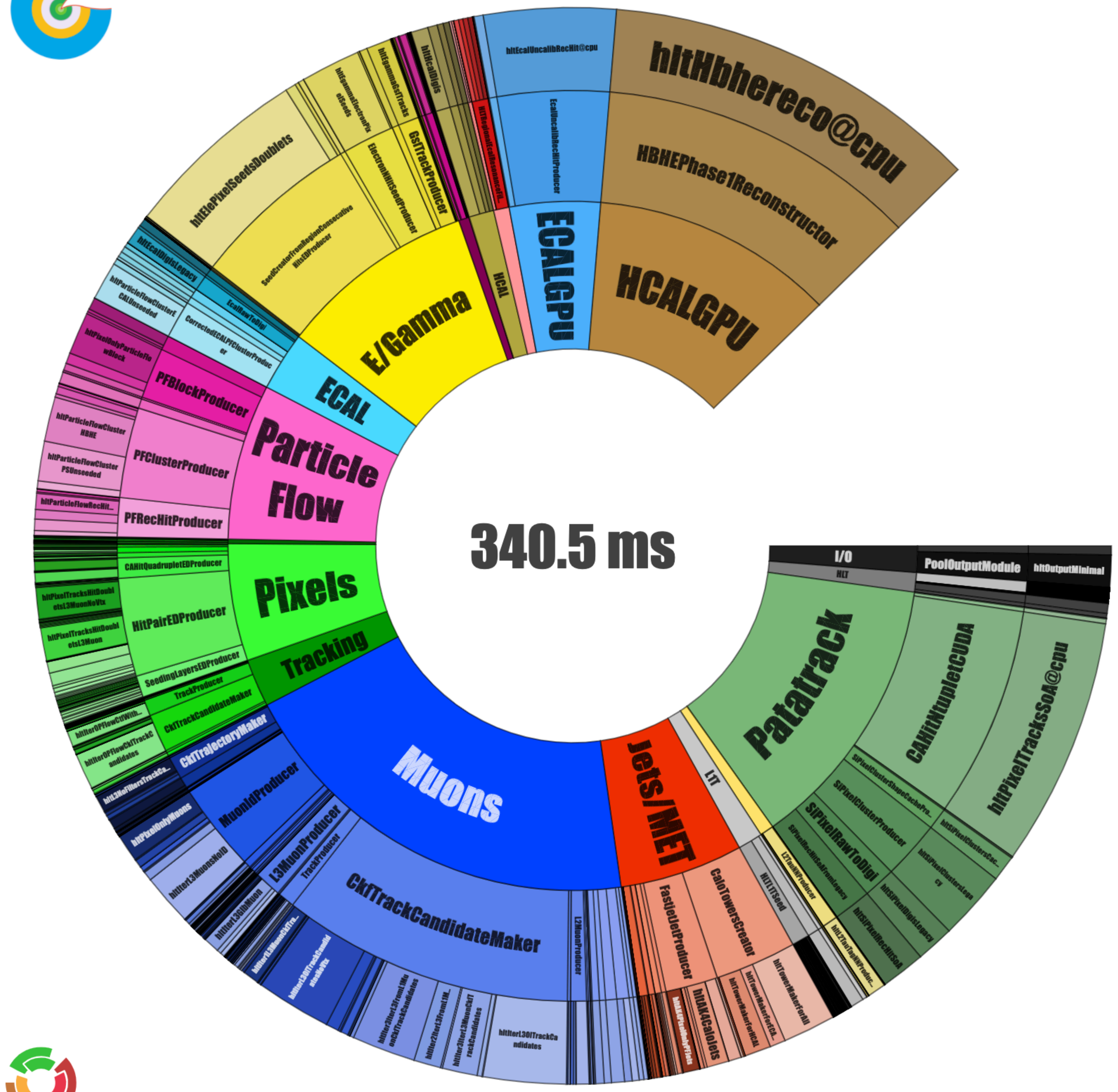
Dataset: Select the data to visualise
or upload a file: Choose File HT_CPU_63jobs.json
Metric: real time | Groups: hlt_new_12_3_0 | Colour style: hlt_new3

Element	Time	Fraction
Muons	110.2 ms	19.4 %
other	78.4 ms	13.8 %
Patatrack	73.8 ms	13.0 %
HCALGPU	60.9 ms	10.7 %
E/Gamma	53.3 ms	9.4 %
Particle Flow	40.9 ms	7.2 %
Jets/MET	34.0 ms	6.0 %
Pixels	30.6 ms	5.4 %
ECALGPU	23.5 ms	4.1 %
ECAL	14.2 ms	2.5 %
Tracking	12.5 ms	2.2 %
I/O	7.1 ms	1.3 %
L1T	6.8 ms	1.2 %
HCAL	6.5 ms	1.2 %
HLT	5.4 ms	0.9 %
AlCa	3.9 ms	0.7 %
Unassigned	3.3 ms	0.6 %
Taus	2.7 ms	0.5 %
Vertices	1.1 ms	0.2 %
Framework	0.0 ms	0.0 %
B tagging	0.1 ms	0.0 %
CTPPS	0.0 ms	0.0 %
total	*569.1 ms*	*100.0 %*

Circles

Results on 2018 Data: CPU

- Running on 2018 data with hyperthreading disabled



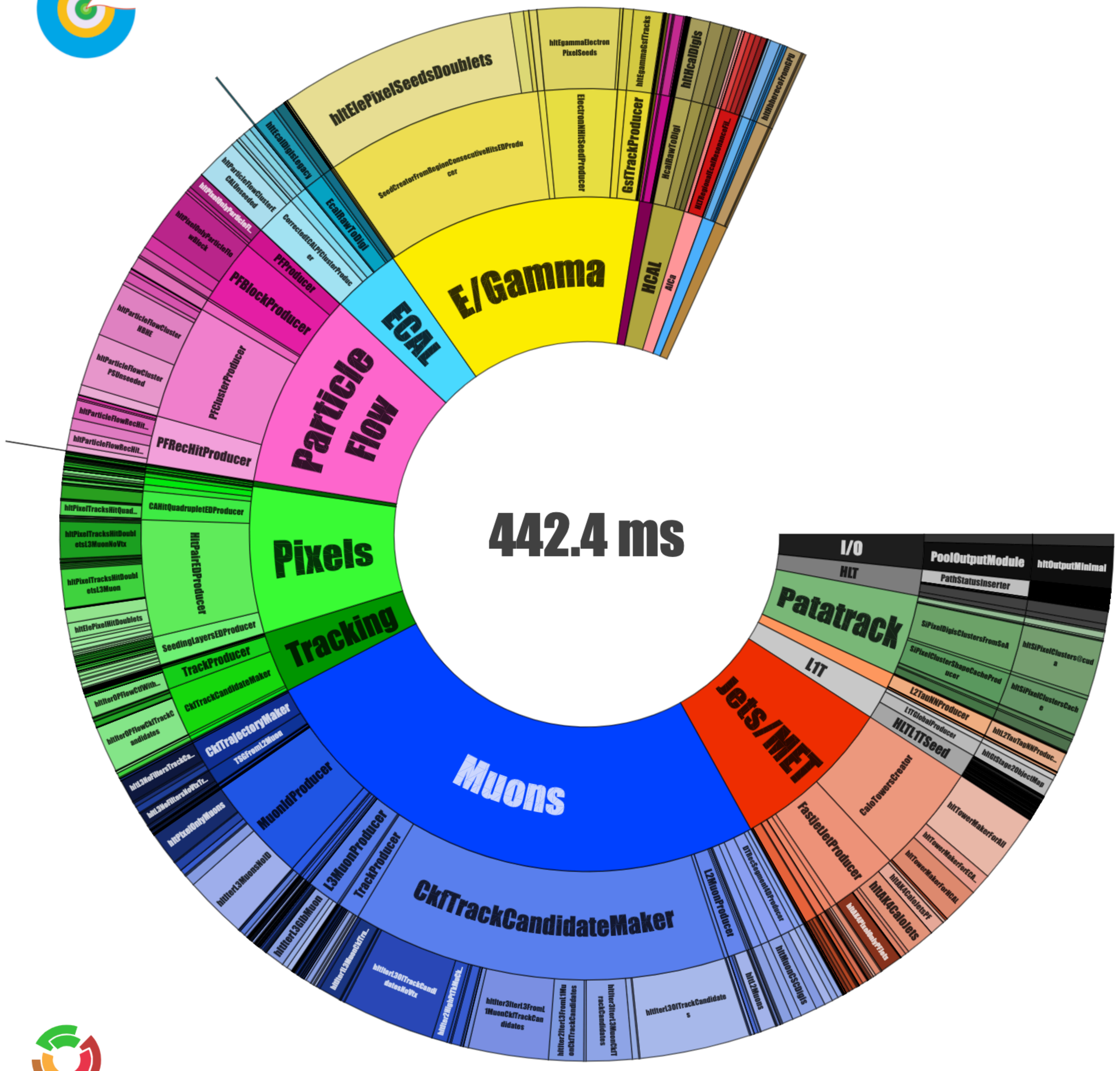
Circles

Dataset Select the data to visualise
or upload a file Choose File CPU_Patatr…_23jobs.json
Metric real time Groups hlt_new_12_3_0 Colour style hlt_new3

Element	Time	Fraction
Muons	66.7 ms	19.6 %
Patatrack	44.3 ms	13.0 %
other	41.9 ms	12.3 %
HCALGPU	38.8 ms	11.4 %
E/Gamma	31.5 ms	9.3 %
Particle Flow	24.9 ms	7.3 %
Jets/MET	19.7 ms	5.8 %
Pixels	18.7 ms	5.5 %
ECALGPU	14.5 ms	4.2 %
ECAL	9.4 ms	2.8 %
Tracking	7.6 ms	2.2 %
I/O	4.2 ms	1.2 %
L1T	4.0 ms	1.2 %
HCAL	4.0 ms	1.2 %
HLT	3.1 ms	0.9 %
Unassigned	2.2 ms	0.7 %
AlCa	2.5 ms	0.7 %
Taus	1.7 ms	0.5 %
Vertices	0.7 ms	0.2 %
Framework	0.0 ms	0.0 %
B tagging	0.1 ms	0.0 %
CTPPS	0.0 ms	0.0 %
total	*340.5 ms*	*100.0 %*

Results on 2018 Data: GPU

- Running on 2018 data with hyperthreading enabled

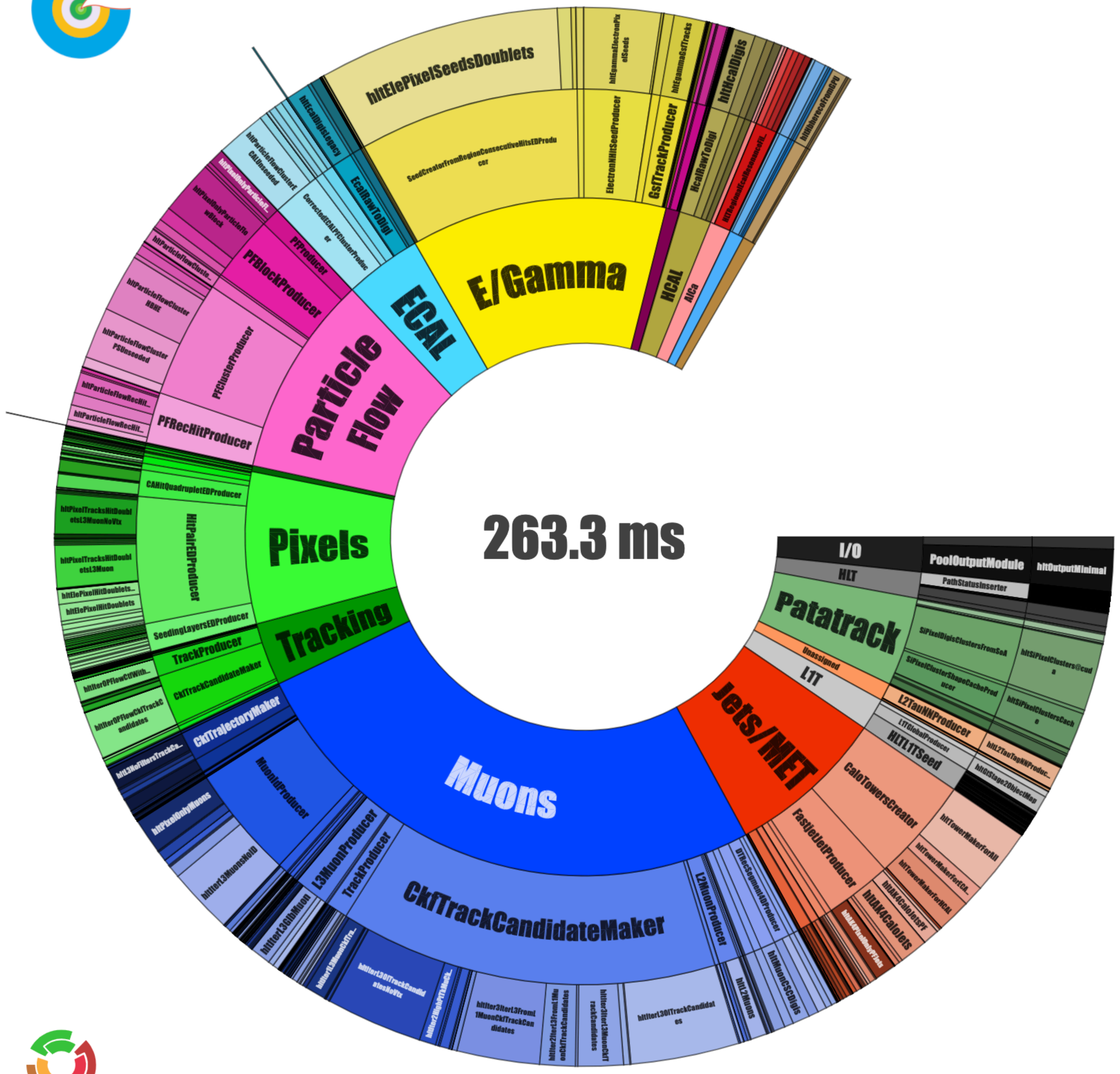


Dataset Select the data to visualise
or upload a file Choose File HT_GPU_All_16jobs.json
Metric real time Groups hlt_new_12_3_0 Colour style hlt_new3

Element	Time	Fraction
Muons	112.4 ms	25.4 %
other	80.4 ms	18.2 %
E/Gamma	54.2 ms	12.2 %
Particle Flow	41.7 ms	9.4 %
Jets/MET	34.5 ms	7.8 %
Pixels	31.1 ms	7.0 %
Patatrack	17.2 ms	3.9 %
ECAL	14.6 ms	3.3 %
Tracking	12.9 ms	2.9 %
I/O	7.3 ms	1.7 %
L1T	6.9 ms	1.6 %
HCAL	6.7 ms	1.5 %
HLT	5.6 ms	1.3 %
AlCa	4.1 ms	0.9 %
Unassigned	3.4 ms	0.8 %
Taus	2.8 ms	0.6 %
ECALGPU	2.8 ms	0.6 %
HCALGPU	2.7 ms	0.6 %
Vertices	1.1 ms	0.3 %
Framework	0.0 ms	0.0 %
B tagging	0.1 ms	0.0 %
CTPPS	0.0 ms	0.0 %
total	*442.4 ms*	*100.0 %*

Results on 2018 Data: GPU

- Running on 2018 data with hyperthreading disenabled



Dataset Select the data to visualise
or upload a file Choose File GPU_8jobs.json
Metric real time Groups hlt_new_12_3_0 Colour style hlt_new3

Element	Time	Fraction
Muons	67.8 ms	25.8 %
other	43.7 ms	16.6 %
E/Gamma	32.1 ms	12.2 %
Particle Flow	25.4 ms	9.6 %
Jets/MET	19.9 ms	7.6 %
Pixels	19.0 ms	7.2 %
Patatrack	11.2 ms	4.2 %
ECAL	9.7 ms	3.7 %
Tracking	7.9 ms	3.0 %
I/O	4.4 ms	1.7 %
L1T	4.1 ms	1.5 %
HCAL	4.0 ms	1.5 %
HLT	3.3 ms	1.2 %
AlCa	2.6 ms	1.0 %
Unassigned	2.2 ms	0.8 %
ECALGPU	1.8 ms	0.7 %
Taus	1.7 ms	0.6 %
HCALGPU	1.6 ms	0.6 %
Vertices	0.7 ms	0.3 %
Framework	0.0 ms	0.0 %
B tagging	0.1 ms	0.0 %
CTPPS	0.0 ms	0.0 %
total	*263.3 ms*	*100.0 %*

Circles