




# Heterogenous Computing in High Energy Physics

(a CMS perspective)

Adriano Di Florio (CC-IN2P3)

FJPPL, 30-31 January 2024

# Outline and disclaimers

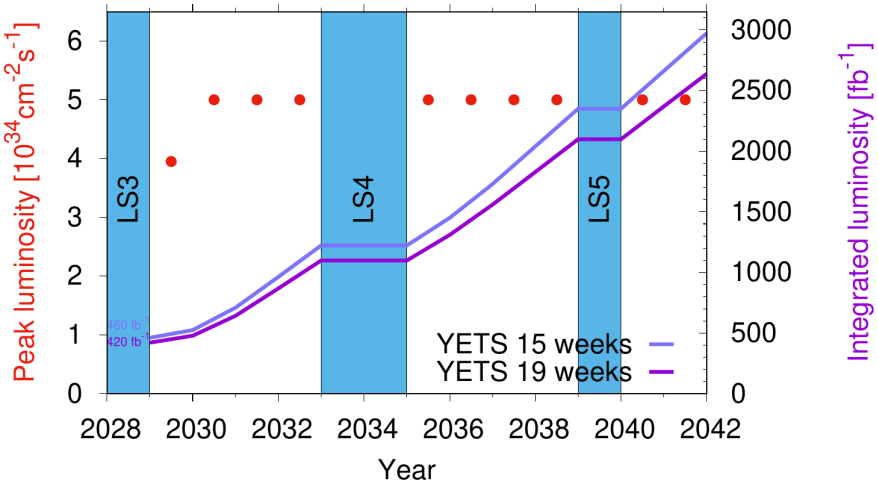
In these slides I will summarize what has happened, is happening and will happen in CMS on the heterogenous side, focussing on how we tackled some challenges.

**Disclaimer:** I tried to give a general picture, some technical details may be missing. Materials readapted from many sources.

# Why

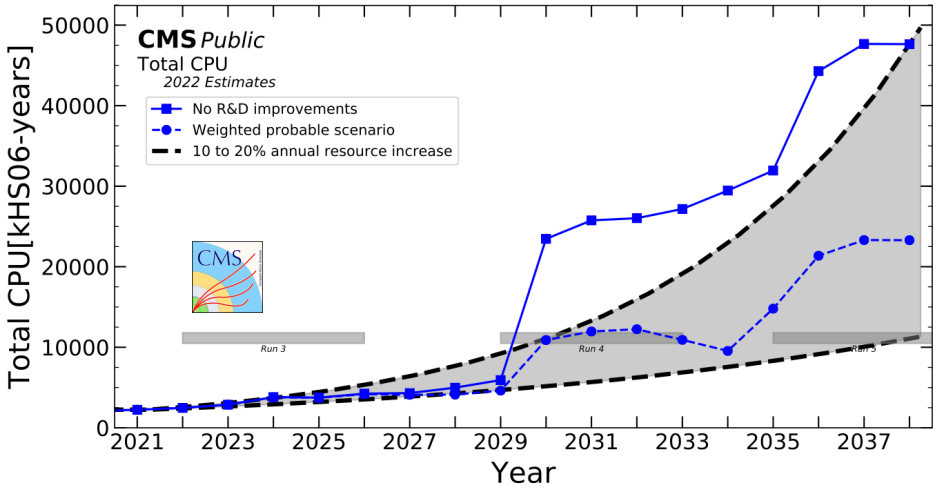
# An interlude, towards HL LHC (in a nutshell)

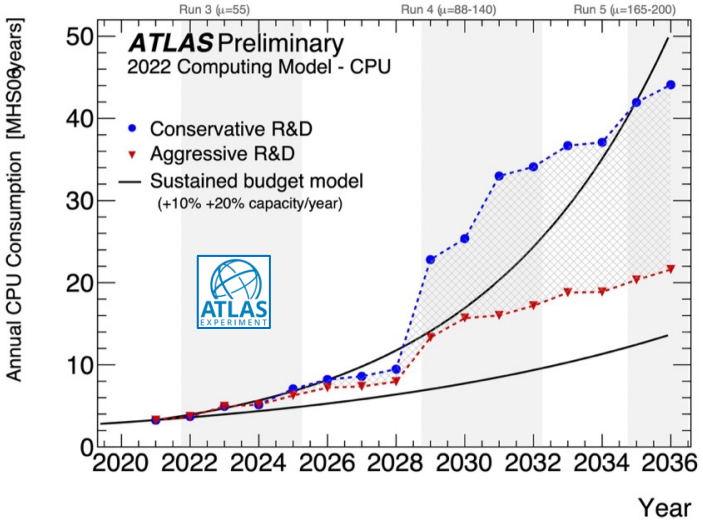
- Accelerator and detectors upgrade: **many channels, high granularity.**
- **More data**: double Run 1,2,3 integrated luminosity in one year.
- Mirror data (×10 statistic wrt today) with (many) simulated events.
- From ~50 to 140/200 parasitic collisions per bunch crossing.
- **4/5x event sizes** (stress on network+storage+tape infrastructure).
- Proton physics + Heavy Ions.



# An interlude, towards HL LHC: CPU needs

- Budget vs needs for ATLAS and CMS.

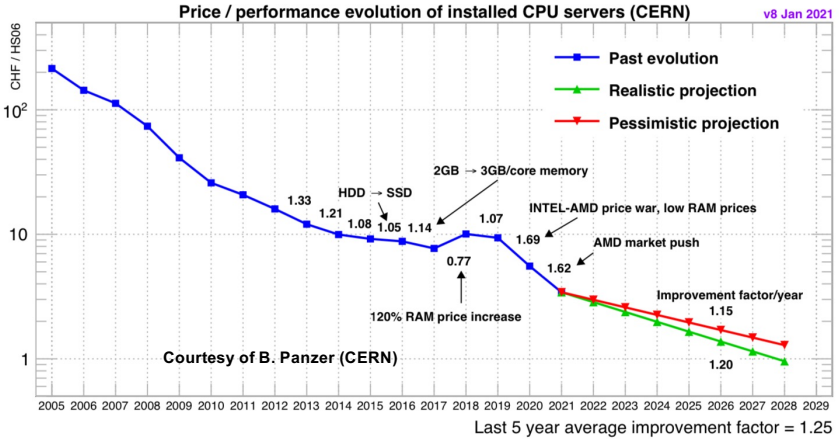


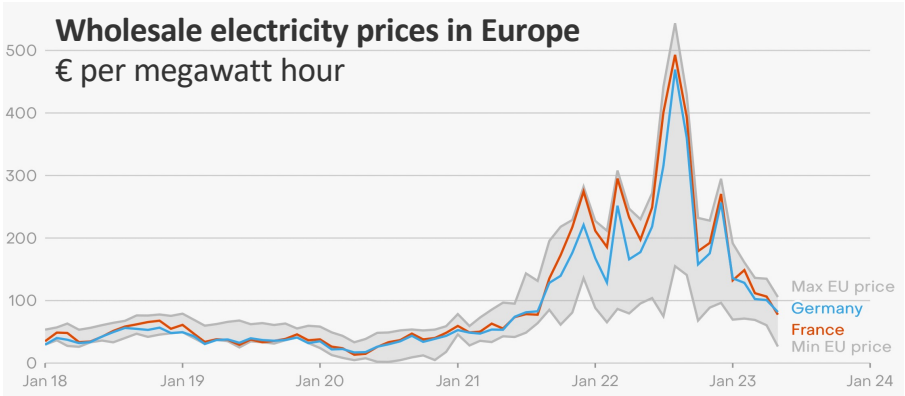




- On the bright side: things are improving (trends have headed down). But still.

# Heterogeneous Architectures

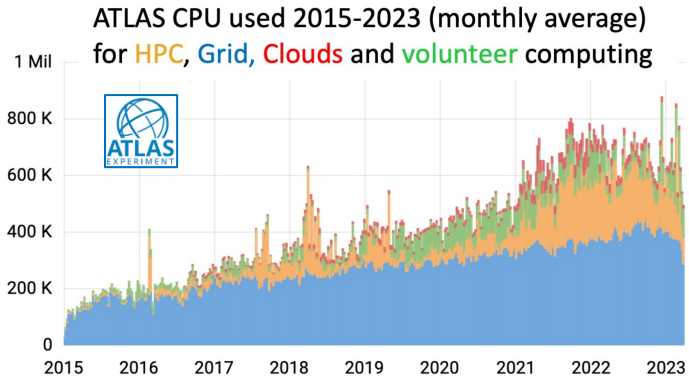
- Hardware cost dominated by market trends, which (our) science **cannot influence (anymore).**
- **Necessity to diversify (micro)architectures** (for many years to be essentially focusing on a single one): **CPU** (x86_64, Power, ARM, RISC) and **accelerators** (e.g. GPUs, departing from single vendor leadership).
- Heterogeneity is hitting the mass market.
- Power consumption becoming more important than ever (accelerators help).

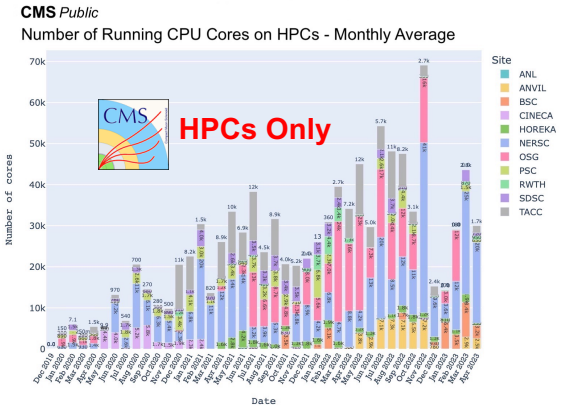


# High Performance Computing

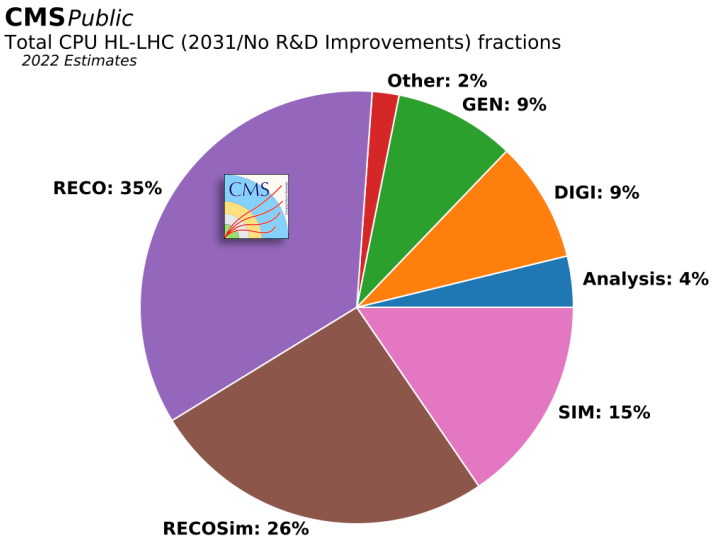
- HPCs more and more visible in the scientific computing infrastructure (massive investments from financing agencies).
- Several Exascale machines will be available during the next decades.
- HPCs used by all LHC experiments to some extent.
- HPCs for High Throughput Computing come at a price, e.g.:
  - **Data access (access, bandwidth, caches ...)**
  - Environment less open than Grid one (OS, access policies, ...)
  - Top performance comes **with exotic architectures**

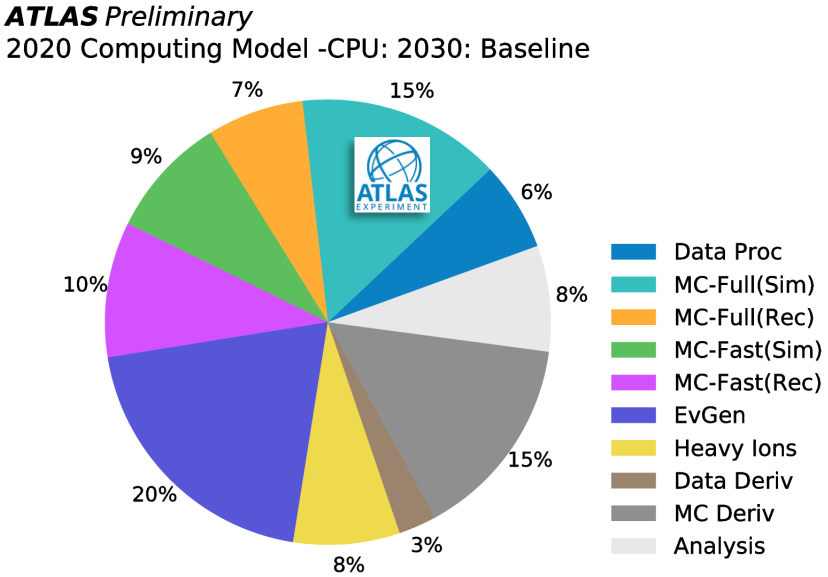


# An interlude, towards HL-LHC

- But where will we spend our HS06 in 2030?



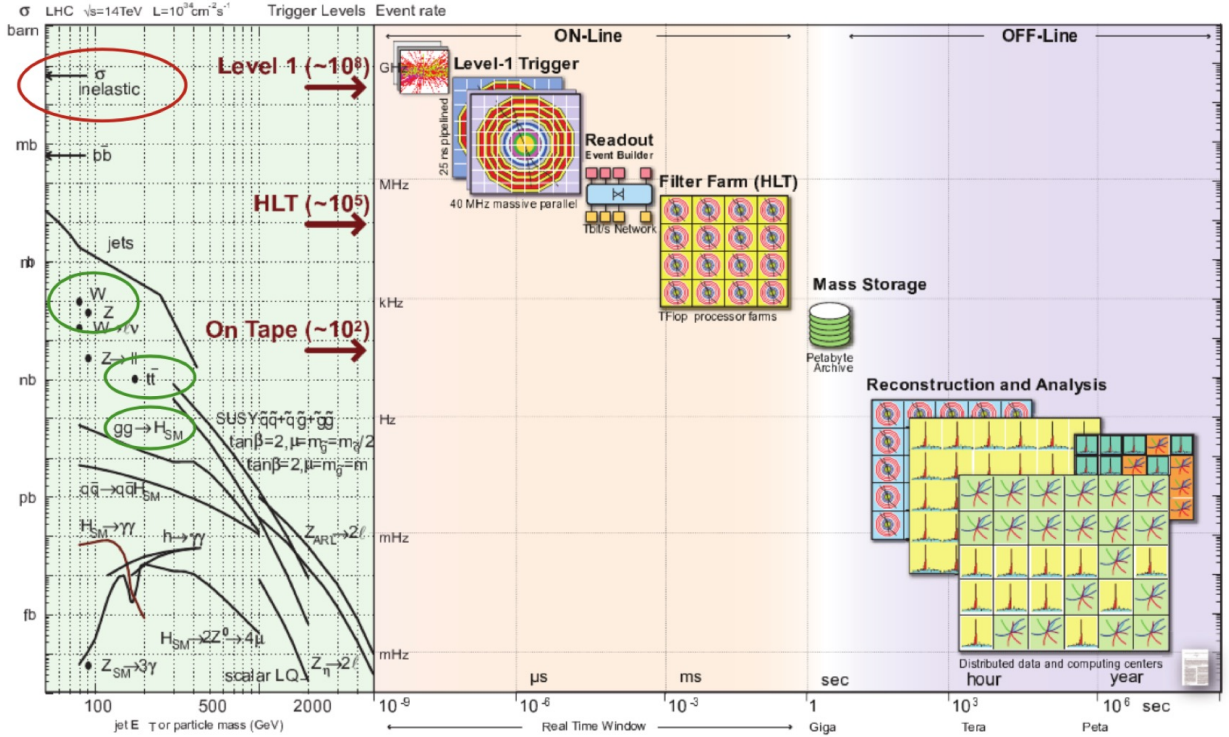




- Most of them go in event reconstruction.
- And most of it for simulations: ~no access to data (HPC friendly).

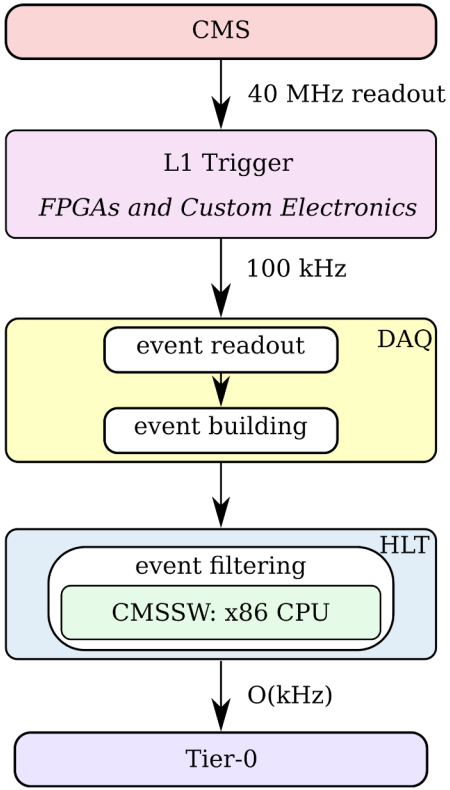
# What

# CMS Trigger(s) in a nutshell

- Collisions in the LHC happen at 40MHz, impossible to save all events
- **Level 1 trigger:** first filtering based on FPGAs and custom electonics reduces the rate to 100 kHz
- High Level Trigger (HLT): streamlined version of reconstruction software reduces the rate to about 1kHz for O(1GB/s) data readout



# CMS HLT Trigger towards Phase2




Since Run-3 (2022+), CMS operates an heterogeneous HLT farm:

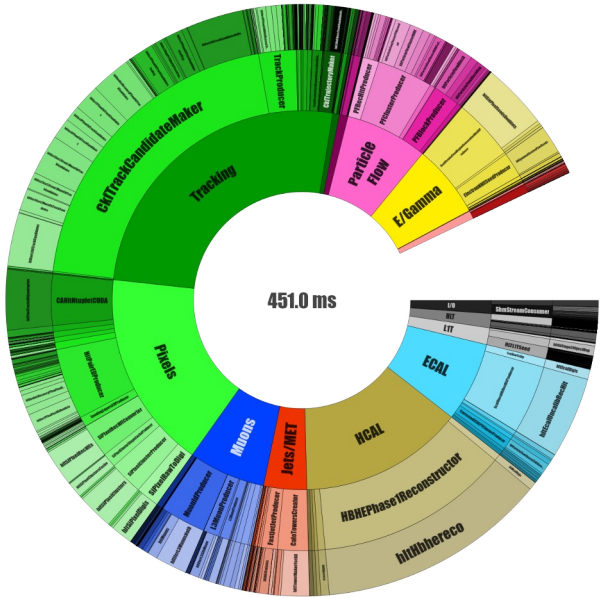
- CMS software, running on 200 nodes:
  - ~26k CPU cores AMD Milan 7763
  - 400 NVIDIA T4 GPUs
- Event size ~MB

CMS Phase-2 upgrade:

- L1T 100kHz -> 750kHz (7.5x)
- Pile-up 60 -> 200 (~3x)
- More complex detectors

Almost an order of magnitude higher performance required: CPU evolution is not able to cope with the increasing demand of performance.

# Where it started . . .




*Online reconstruction timing measured on pileup 50 events from Run2018D on a full **Run-2** HLT node (2x Intel Skylake Gold 6130) with HT enabled, running 16 jobs in parallel, with 4 threads each.*

While looking for opportunities for GPU offloading:

- No single hotspot
- Many instances of the same algorithm with different configurations (regional, global)
- Chains of algorithms containing enough parallelism
- Early in the reconstruction, no dependencies and then move downstream
- Enough expertise in the particular reconstruction algorithm and parallelism

Identified three targets:

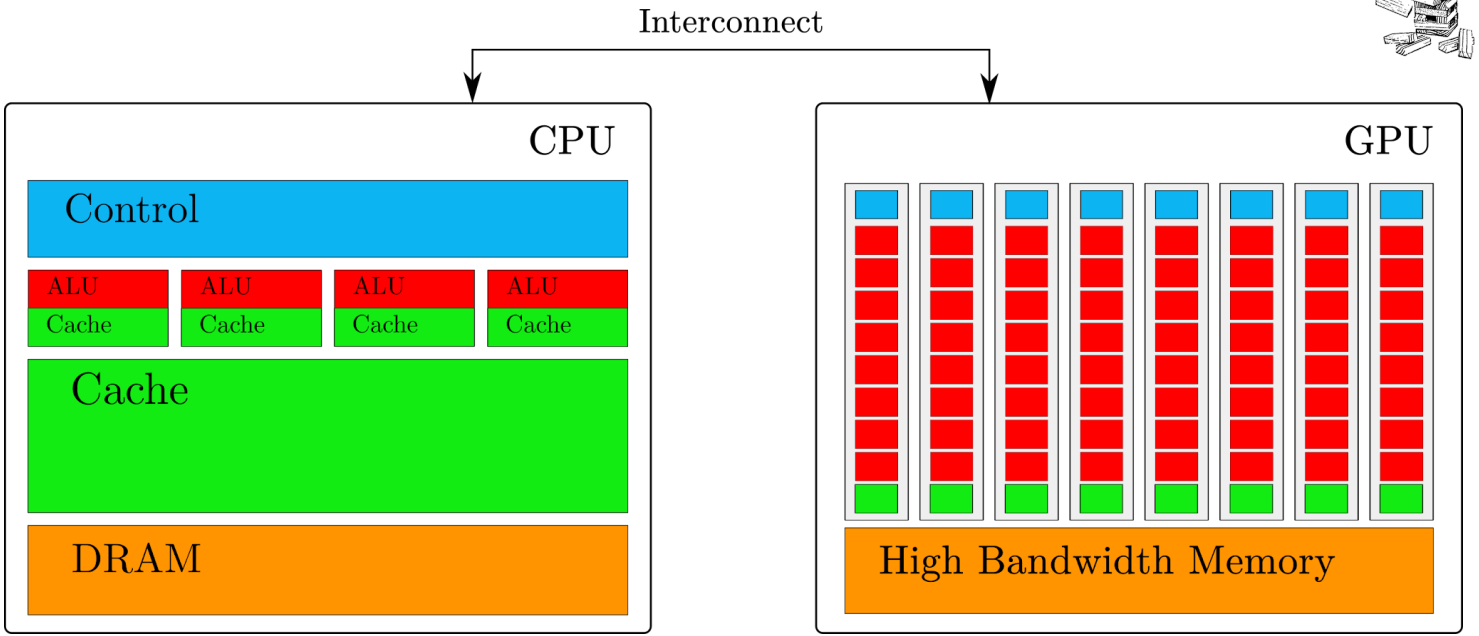
- Global pixel reconstruction chain, from RAW data down to pixel-only tracks and vertices
- ECAL local reconstruction
- HCAL local reconstruction

Make CPU and GPU codes to run together efficiently in the CMSSW framework

# . . . with a bunch of questions

- How do I make the code run faster?
- Should I change the data structures and memory management?
- What if the algorithm is not exposing enough parallelism?
- What if there is no single algorithm or detector that is a hotspot taking most of the reconstruction time?
- How do I integrate the code into the framework?
- How do I make the code portable?
- How do I make sure that the CPU threads are not idle waiting for the GPU?
- How do I balance the number and type of GPUs with the fraction of GPU code in the software?
- How do I match the GPU requirements of the job with the machines available on the grid?
- How do I engage the community to develop heterogeneous code?

# Heterogenous Architecture




- Avg bandwidth between CPU and host memory
- Low core count/Powerful ALU
- Complex control unit
- Large caches

Latency

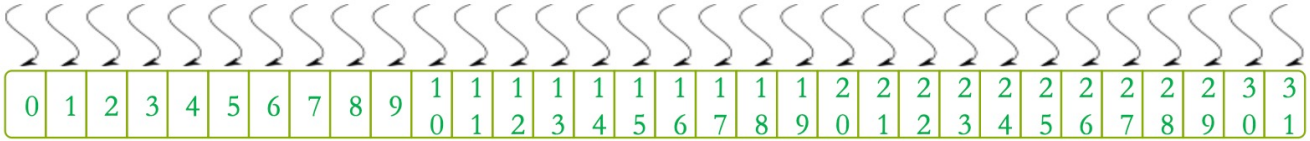
- High bandwidth between GPU cores and GPU memory
- High core count
- No complex control unit
- Small caches

Throughput

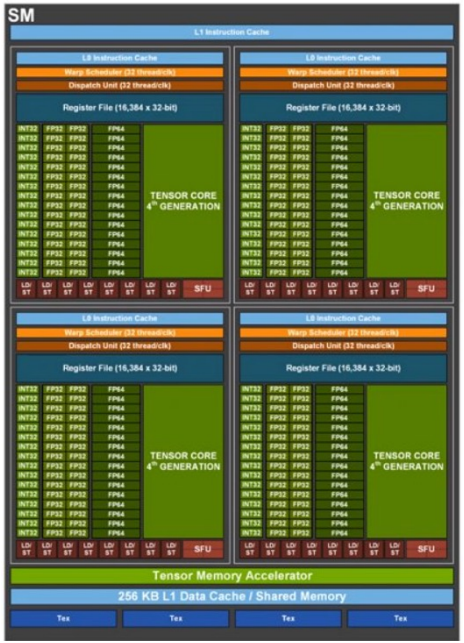
# GPU Programming Model

GPU code is executed by many "threads" in parallel
- Parallel functions, aka "kernels" spawn threads organized in a "grid" of blocks

Threads in the same block can:
- Communicate via fast on-chip shared memory
- Synchronize




Threads in the same block work in steplock in groups (aka warps) of 16, 32, 64 threads depending on the GPU brand.
- Preferred access to main device memory is coalesced
- Lose an order of magnitude in performance if cached access pattern used on GPU




*One of the 132 Streaming Multiprocessors of a NVIDIA H100*

# Where we are now (at HLT)

Fully integrated in the CMSSW framework

- can be reused in the offline reconstruction
- validated offline on GPU-equipped nodes on CMS Tier-1
- and Tier-2s commissioned and optimised over last year
- deployed in production since the beginning of LHC Run-3
- better physics pefromances

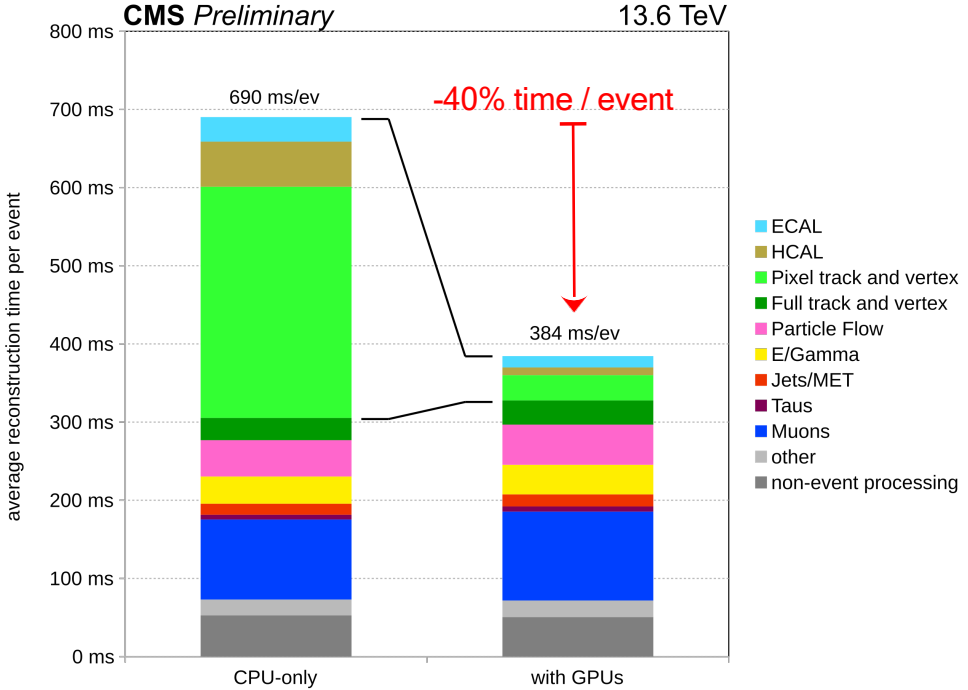
With the deployment of a GPU-equipped HLT farm:

- 70% better event processing throughput
- 50% better performance per kW
- 20% better performance per initial cost

Work is ongoing to rewrite more:

- particle flow clustering
- seeding of the electron reconstruction
- full primary vertex reconstruction

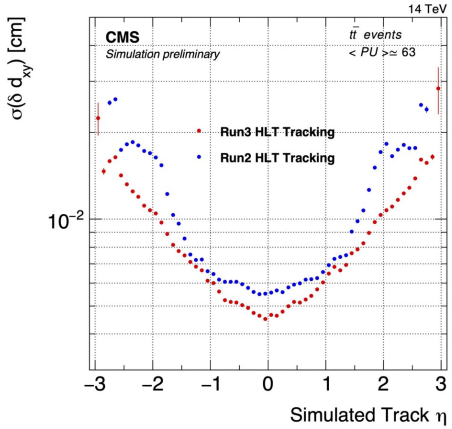
and more, targeting also the Phase-2 reconstruction

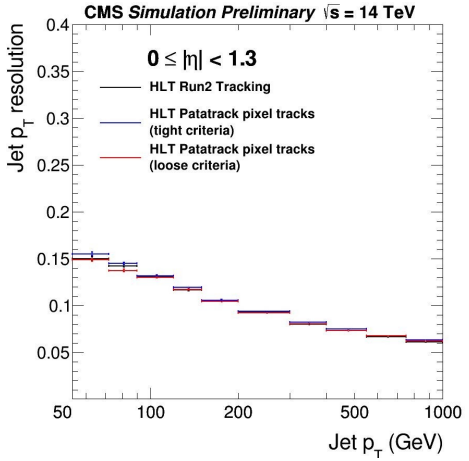



*2× AMD EPYC 7763*
*2× NVIDIA T4*

# Faster == Better

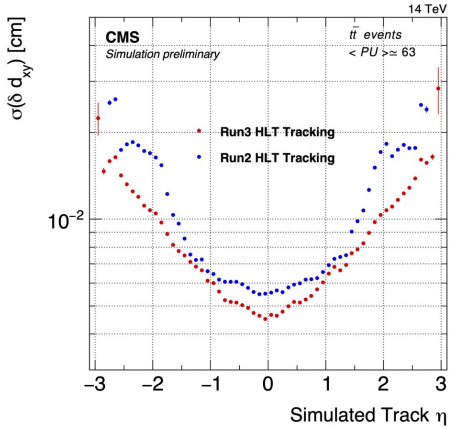
- Usage of a more accurate reconstruction
  - eg. migration to **single iteration tracking**
- Reduction of **intermediate filters**
  - eg. removal of b-tagging based on calo jets
- New timing consuming modules
  - eg. dedicated tracking for LLP, ParticleNet, **soft electron reco**
- Improve **scouting** in quality and rate:
  - muons, photons, electrons, PF jet/MET, tracks PF candidate for offline studies (eg. boosted obj.)
  - special version of **Particle Flow** based on pixel-track-based to minimize the impact on CPU
  - rate up to **30 kHz** (ie. 30% of L1 rate!)
- Necessary step for **Phase-2** HLT (PU 200)

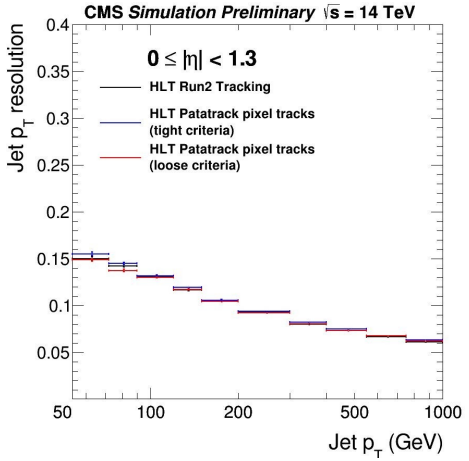


# Faster == Better

- Usage of a more accurate reconstruction
  - eg. migration to **single iteration tracking**
- Reduction of **intermediate filters**
  - eg. removal of b-tagging based on calo jets
- New timing consuming modules
  - eg. dedicated tracking for LLP, ParticleNet, **soft electron reco**
- Improve **scouting** in quality and rate:
  - muons, photons, electrons, PF jet/MET, tracks PF candidate for offline studies (eg. boosted obj.)
  - special version of **Particle Flow** based on pixel-track-based to minimize the impact on CPU
  - rate up to **30 kHz** (ie. 30% of L1 rate!)
- Necessary step for **Phase-2** HLT (PU 200)



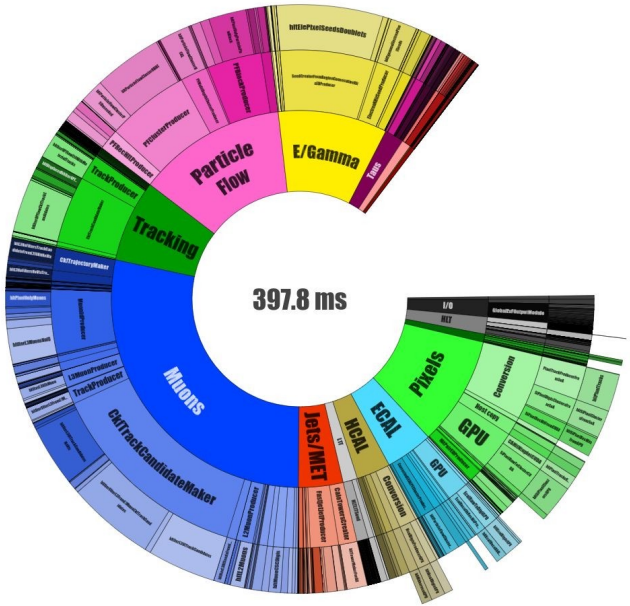



# How

# Steps to run


- The HLT menu has total of ~**4400 modules**

- GPU Offloaded parts
  - Pixel detector reconstruction: from RAW data unpacking up to tracks and vertices (11 modules)
  - ECAL local reconstruction (4 modules)
  - HCAL local reconstruction (3 modules)
- 57 unique kernels, ranging from 2 μs to 7 ms in these events

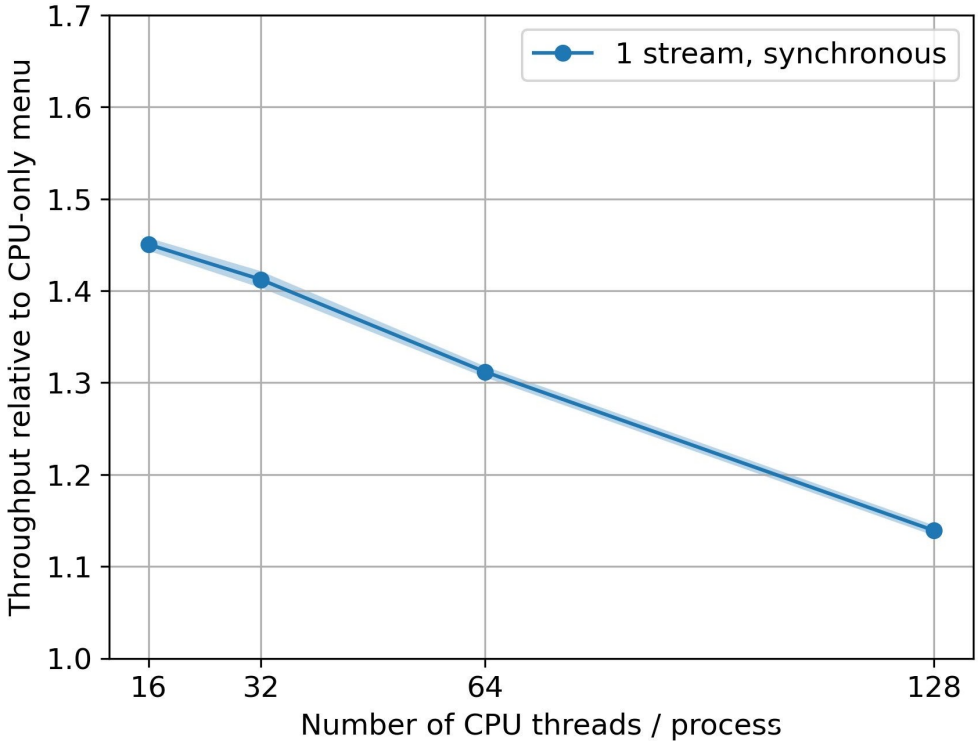




- Memory pool to amortize cost of raw memory allocations and provide asynchronous allocation interface in CUDA stream order
- All offloaded modules have CPU versions that are used for reference measurement

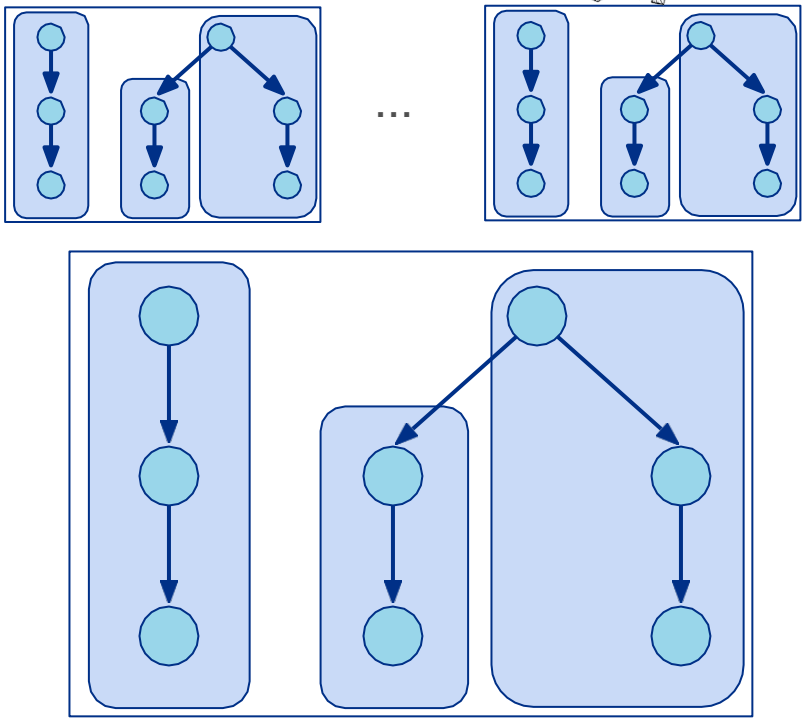
# Let's Start Simple

- Each CUDA-using module launches their CUDA work by directly interacting with the CUDA API
- All these modules launch their work into the same CUDA stream
  - Mimics the behavior of the default CUDA stream
- Every CUDA-using module does a blocking synchronization
  - cudaStreamSynchronize()
- 15-45 % improvement compared to CPU-only



# Add CUDA Streams


- Each non-branching chain of modules within an event uses a separate CUDA stream.
- Each concurrent event has its own chains.

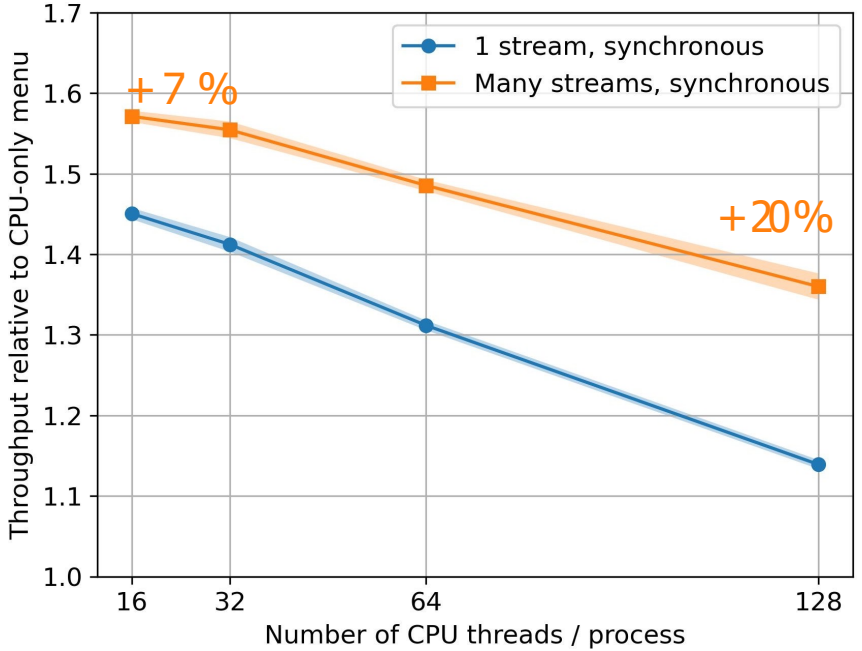



Example module chains where 3 CUDA streams are used.

- Every CUDA-using module still does a blocking synchronization
  - Tested cudaDeviceSchedule{Auto, Spin, Yield, BlockingSync}, all gave practically the same performance
    - Reporting cudaDeviceScheduleAuto

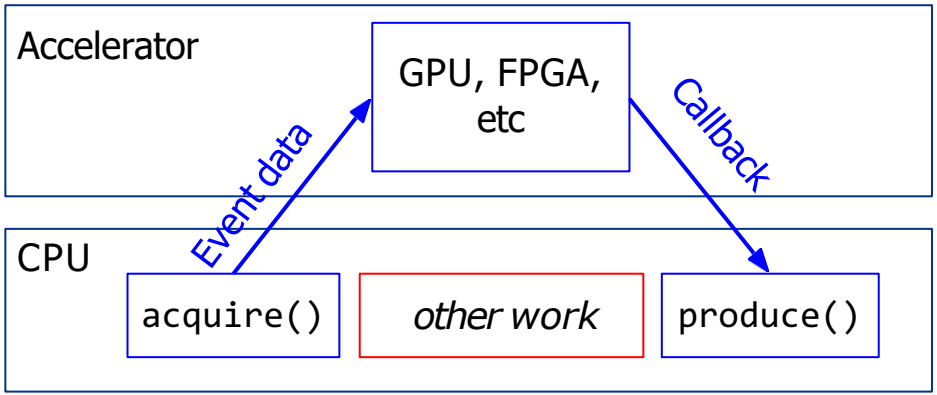
# Add Multiple CUDA Stream

- Each non-branching chain of modules within an event uses a separate CUDA stream
  - Each concurrent event has its own chains
- Every CUDA-using module still does a blocking synchronization
  - Tested cudaDeviceSchedule{Auto, Spin, Yield, BlockingSync}, all gave practically the same performance
    - Reporting cudaDeviceScheduleAuto



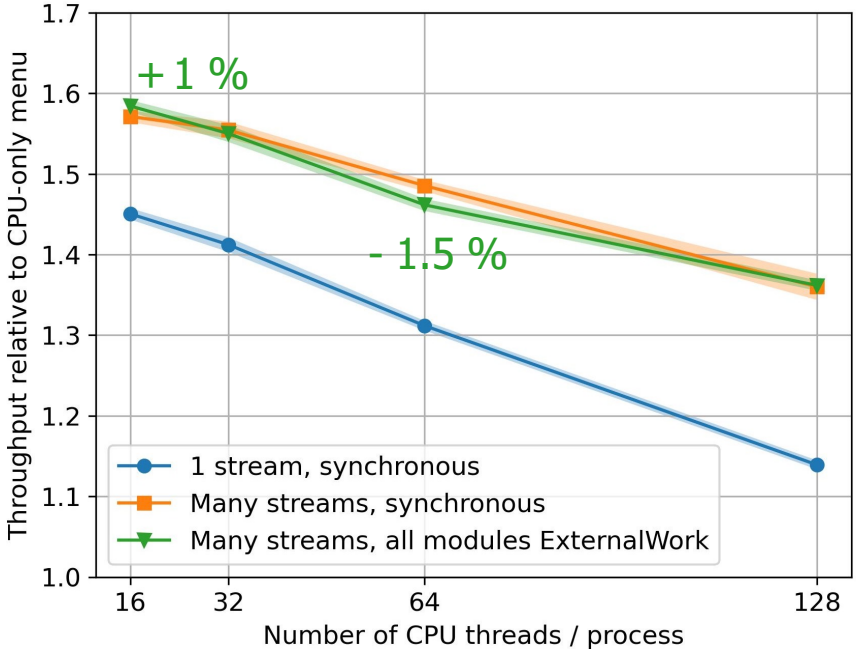
# External Worker Mechanism

- Replace blocking waits with a callback-style solution
- Traditionally the algorithms have one function called by the framework, produce()
- That function is split into two stages
  - acquire(): Called first, launches the asynchronous work
  - produce(): Called after the asynchronous work has finished
- acquire()is given a reference-counted smart pointer to the task that calls produce()
  - Decrease reference count when asynchronous work has finished
  - Capable of delivering exceptions



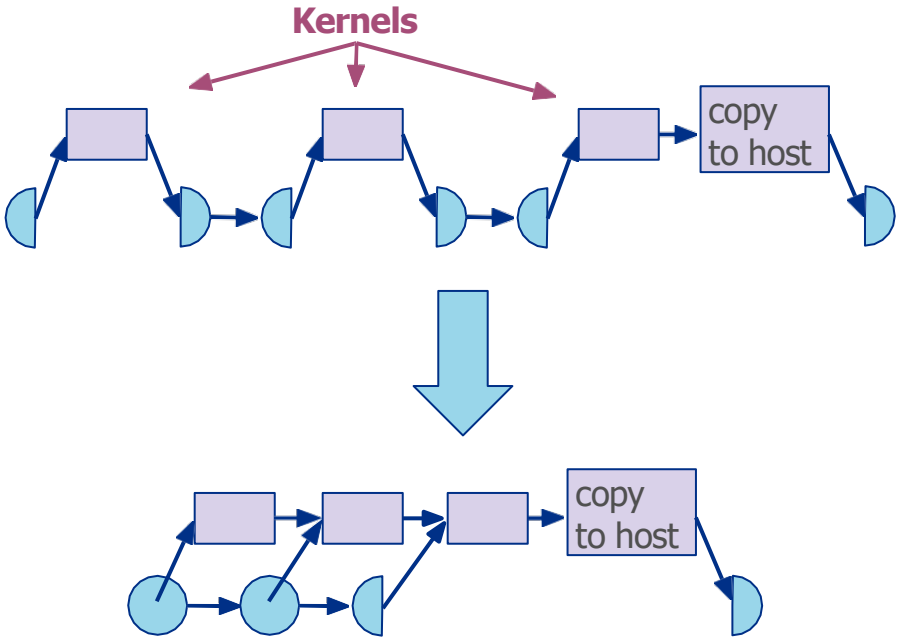
# External Worker Mechanism

- Use of CUDA streams stays the same
- Every CUDA module does a non-blocking synchronization
  - It follows that the modules depending on the data of the CUDA-using module are scheduled to be run only after the GPU work has finished
  - We use cudaStreamAddCallback() to queue a host-side callback function that notifies the CMSSW framework of the completion of the GPU work
    - cudaStreamAddCallback() is deprecated, cudaLaunchHostFunc() gave same performance



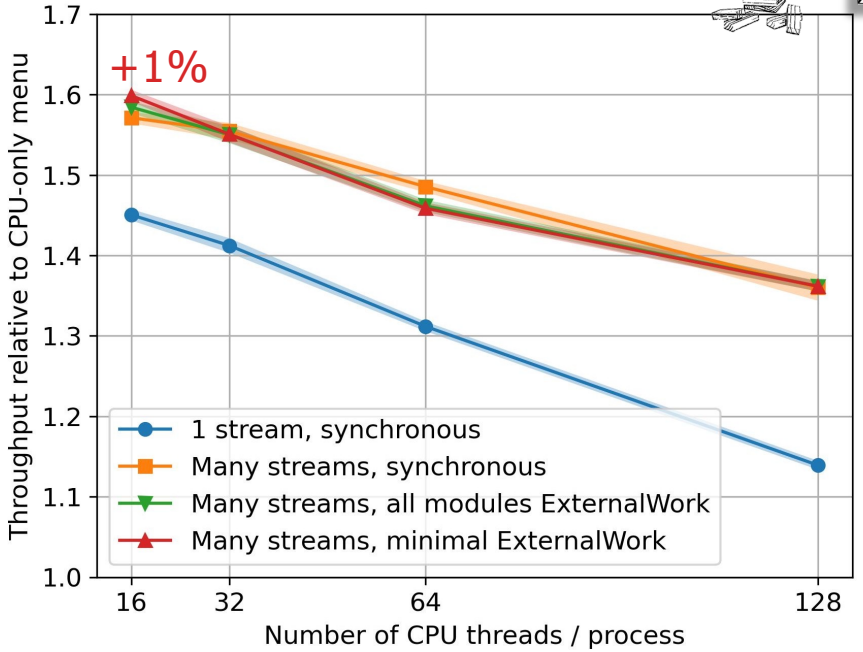
# Minimize the external worker use

- Use of CUDA streams stays the same
- Modules that produce only "device-side" information do not really need synchronization with host
  - Instead we make the consuming module to call cudaStreamWaitEvent() in case it would use a different stream
  - Now framework can schedule the consuming modules without waiting their GPU work to finish
- This is the setup used in CMSSW



# Minimize the external worker use


- Use of CUDA streams stays the same
- Modules that produce only "device-side" information do not really need synchronization
  - Instead we make the consuming module to call cudaStreamWaitEvent() in case it would use a different stream
  - Now framework can schedule the consuming modules without waiting their GPU work to finish
- This is the setup used in CMSSW




Further improvements with our own pool of threads waiting on cudaEventSynchronize(): ~2 % better on top of cudaStreamAddCallback().

# Where

# Perfromance Portability

- New code written using the native CUDA API, targetting NVIDIA GPUs
- But most offline sites (CERN, WLCG) do not use GPUs...
- Two «by-hand» approaches:

maintainance issues!

  - common kernels, different wrappers;
  - ad hoc compatibility layer, with a lot of **#ifdefs**

code duplication!

  - two implementations: legacy (CPU-only) and parallel (GPU-only)
  - duplication of development, maintenance and validation efforts

- Adoption of GPUs from other vendors in HPCs is increasing
  - **LUMI-G**, in Finland, and **Frontier**, at Oak Ridge, use AMD MI250X GPUs
  - **Aurora**, at Argonne National Laboratory, will use Intel Xe GPUs
- Can we target different CPUs and GPUs with a single code base ? Yes. No free lunch.

how do we run there ?

and how do we run here?

# A vast zoology


OpenCL™

OpenMP®

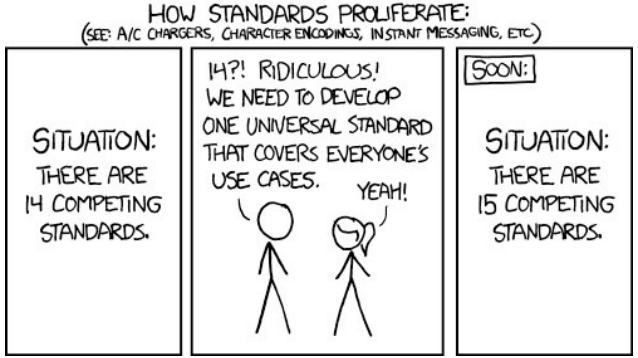
std::par

SYCL™

OpenACC

okkos

alpaka



# Performance Portability: Solutions

- Performance portability layers: aim to provide performance portability across accelerators through the abstraction of the underlying levels of parallelism. Rapidly changing ~O(months) portability solutions.

Software →

Hardware ↓

| | CUDA | Kokkos | SYCL | HIP | OpenMP | alpaka | std::par |
|---|---|---|---|---|---|---|---|
| NVIDIA GPU | | | intel/llvm compute-cpp | hipcc | nvc++ LLVM, Cray GCC, XL | | nvc++ |
| AMD GPU | | | openSYCL intel/llvm | hipcc | AOMP LLVM Cray | | |
| Intel GPU | | | oneAPI intel/llvm | CHIP-SPV: early prototype | Intel OneAPI compiler | prototype | oneapi::dpl |
| x86 CPU | | | oneAPI intel/llvm computecpp | via HIP-CPU Runtime | nvc++ LLVM, CCE, GCC, XL | | |
| FPGA | | | | via Xilinx Runtime | prototype compilers (OpenArc, Intel, etc.) | prototype via SYCL | |

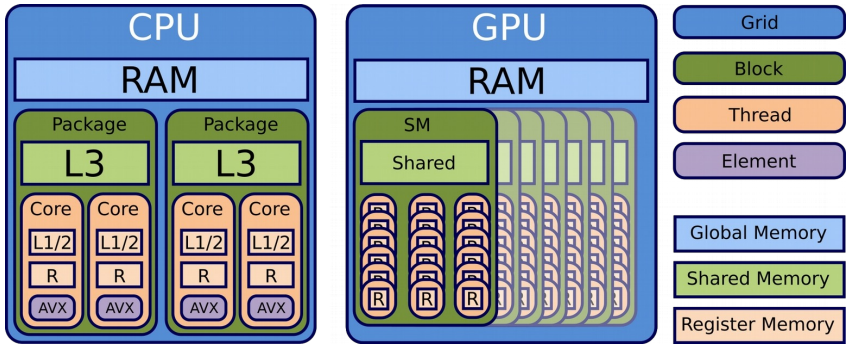
- HEP-CCE cross-experiment initiative has

| | Kokkos | SYCL | OpenMP | Alpaka | std::par |
|---|---|---|---|---|---|
| Patatrack | Done | Done* | WIP | Done* | Done compiler bugs |
| Wirecell | Done | Done | Done | no | Done |
| FastCaloSim | Done | Done | Done | Done | Done |
| P2R | done | Done | OpenACC | Done | Done |

- After a thorough investigation and discussion Alpaka
- N.B.: this is not cast in stone. CMS is still continuously exploring the rapidly changing ~O(years) portability solution landscape.
- With a portable SoA model (see later).

# alpaka


- alpaka is a header-only C++17 abstraction library for heterogeneous software development
  - it aims to provide *performance portability* across accelerators through the abstraction of the underlying levels of parallelism
  - *may* expose the underlying details when necessary
  - (almost) *native* performance on different hardware

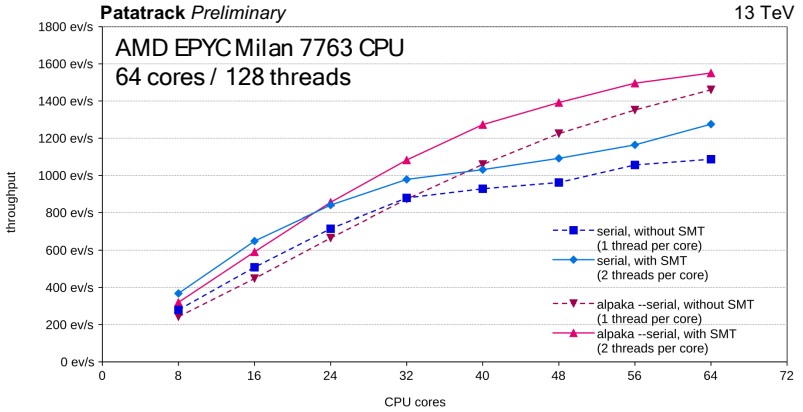


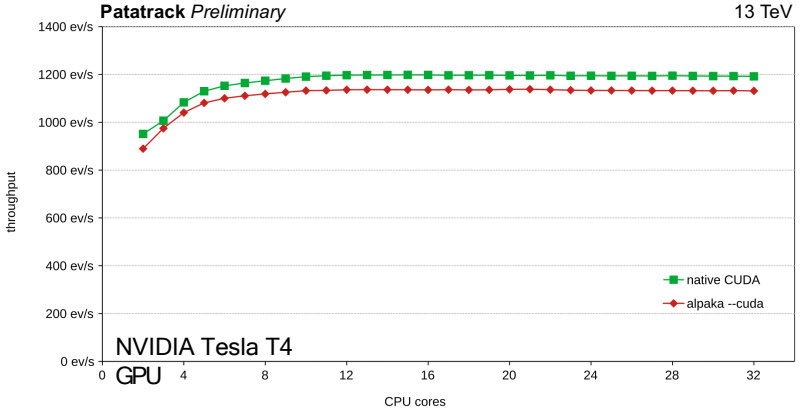

- supports all platforms of interest to CMS
  - x86, ARM and Power CPUs
    - with serial and parallel execution
  - NVIDIA and AMD GPUs
    - with CUDA and ROCm backends
  - support for Intel GPUs and FPGAs is *under development*, based on SYCL and oneAPI

- it is production-ready today !
  - open source project, easy to contribute to: https://github.com/alpaka-group/alpaka/

# Performances in alpaka

- evaluated on Run-3 algorithms within the [Patatrack pixel-only standalone reconstruction]
- good performance on current hardware
  - running on an AMD EPYC "Milan" 7763 CPU (64 cores / 128 threads SMT)
  - running on an NVIDIA Tesla T4 GPU
- header-only library, easy to integrate in the CMS framework
  - support multithreading in the host application
  - support multiple targets in a single build
    - GPUs from different vendors and different generations
    - CPUs with different execution modes, *e.g.* parallel execution using TBB
- low-level approach, very close to CUDA
  - easy to port code from CUDA, and to teach to students

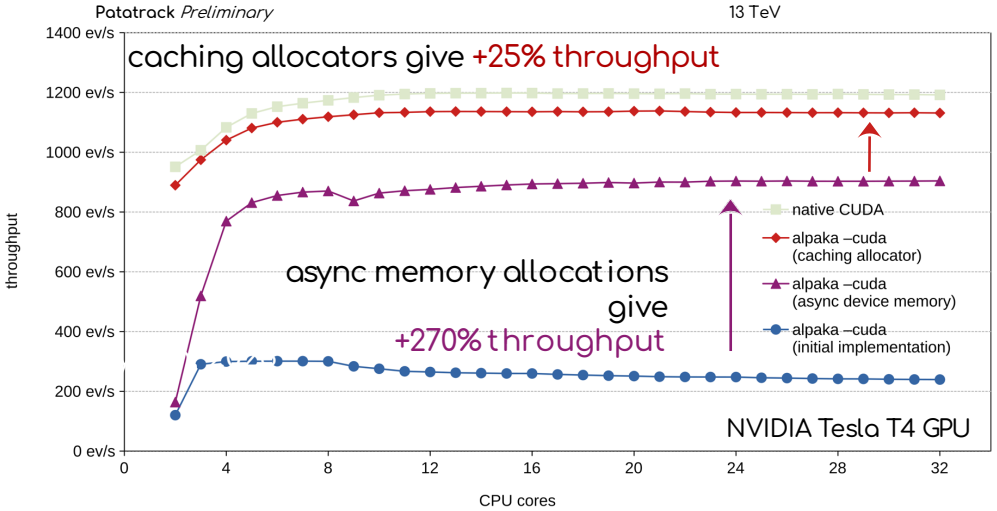


# CMS contributions in alpaka


fruitful collaboration with the Alpaka development team

- improve efficiency for CMS workflows
- contribute support for new features and architectures students' projects !




- more flexible support for CUDA and HIP
  - support for CUDA and HIP APIs in the host
  - compiler support for CUDA and HIP targets in a single build
- asynchronous memory allocations, on backends that support them
  - `cudaMallocAsync` / `cudaFreeAsync()`
  - CUDA ≥ 11.2, ROCm ≥ 5.4, CPUs
- caching of GPU resources
  - streams and events
  - device and host memory buffers
- contribute to the SYCL implementation
  - support for USM memory model in oneAPI
- more efficient atomic operations
- improved memory buffer and kernel
- syntax bug fixes, improvements to the tests, *etc.*

in progress

# Structures

# Structures of Arrays

SoA use widespread in CMSSW with many ad-hoc implementations.

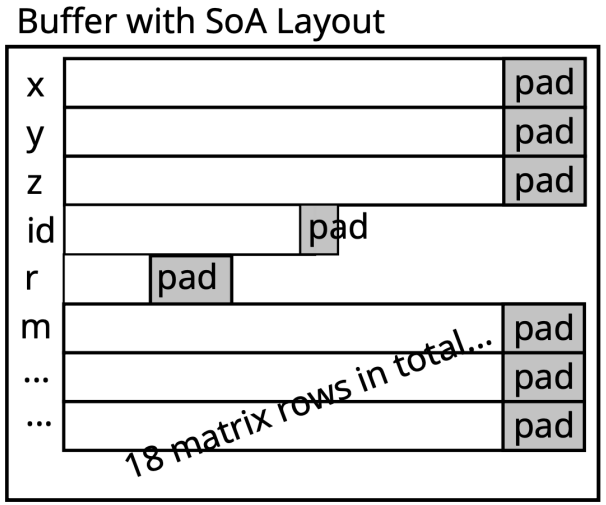
A Generic SoA [1] has been developed to automate definition and implementation of runtime sized SoAs:

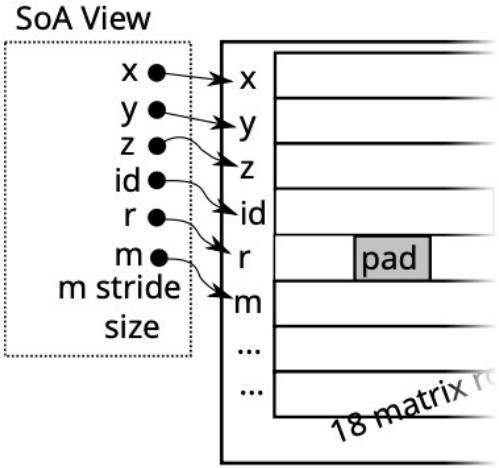
- **Layout** divides a buffer into runtime sized columns
- **View** is the interface to the data. Lightweight, this is the structure passed to kernels
  - Just pointers to columns, size
- **Buffers** host memory, pinned host memory or device memory, allocated from the framework (CUDA or Alpaka)

**PortableCollection** wraps the Layout and View

- Manages the buffer allocation
- Provides an interface for memory transfers
- Manages serialization to ROOT files

# Buffers, Layouts and Views




- Layout helper function computes necessary buffer size from the number of elements
- Layout computes column addresses at construction time
- Add padding at end of columns for cache line alignment
- Also computes auxiliary strides and sizes for Eigen and serialization
- Cache line size configurable at compile time

- Minimal memory footprint to minimize kernel launch parameters size: one pointer per column; size (number of elements); stride size for Eigen elements.
- View is the data access interface
- Const variant of the class
- Constructor variant taking per-column bare pointers
  - Map existing structure to View interface
  - Used during porting to SoA
- Layout classes provide automatically corresponding trivial View

# Structure definition:

- C++ statically typed: code generation before compile time with macros
- Based on `Boost::PP`

```
namespace portabletest {
  // the typedef is needed because commas confuse macros
  using Matrix = Eigen::Matrix<double, 3, 6>;

  // SoA layout with x, y, z, id, m fields
  GENERATE_SOA_LAYOUT(TestSoALayout,
                      // columns: one value per element
                      SOA_COLUMN(double, x),
                      SOA_COLUMN(double, y),
                      SOA_COLUMN(double, z),
                      SOA_COLUMN(int32_t, id),
                      // scalars: one value for the whole structure
                      SOA_SCALAR(double, r),
                      // Eigen columns
                      SOA_EIGEN_COLUMN(Matrix, m))
  using TestSoA = TestSoALayout<>;
}  // namespace portabletest
```

# Portable collections

**PortableCollection** wraps the Layout and View:

```
using TestDeviceCollection = cms::cuda::PortableDeviceCollection<portabletest::TestSoA>;

TestDeviceCollection deviceProduct(size_, ctx.stream()};

testAlgoKernel(deviceProduct.view(), deviceProduct->metadata().size());

cudatest::TestHostCollection hostProduct{size_, ctx.stream()};

cms::cuda::copyAsync(hostProduct.buffer(), deviceProduct.const_buffer(),
  deviceProduct.bufferSize(), ctx.stream());
```

- Available for Alpaka and CUDA.
- Manages the buffer allocation.
- Host and Device side.
- Provides an interface for memory transfers.
- Manages serialization to ROOT files.

# Validation

# caveat emptor

parallel algorithms have some additional problems with respect to serial ones

- more complicated to design and implement efficiently

  *e.g.* divergences in the parallel execution may lead to suboptimal performance

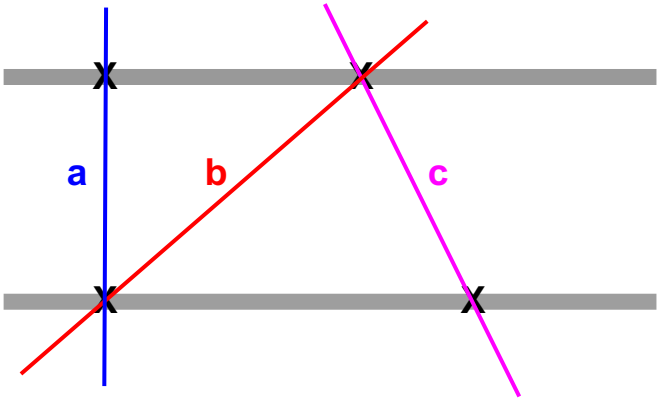
- undefined order of execution may produce results that are not fully reproducible

  *e.g.* in combinatorial algorithms and reductions

# Intrinsic differences: an example

- Tracks sharing one or more hits are considered **ambiguous**: only one is good while the others are fake.

- The GPU tasks work in **parallel**. The tasks are completed in a **random order**. The resolution is order dependent and therefore the result is **not deterministic**.

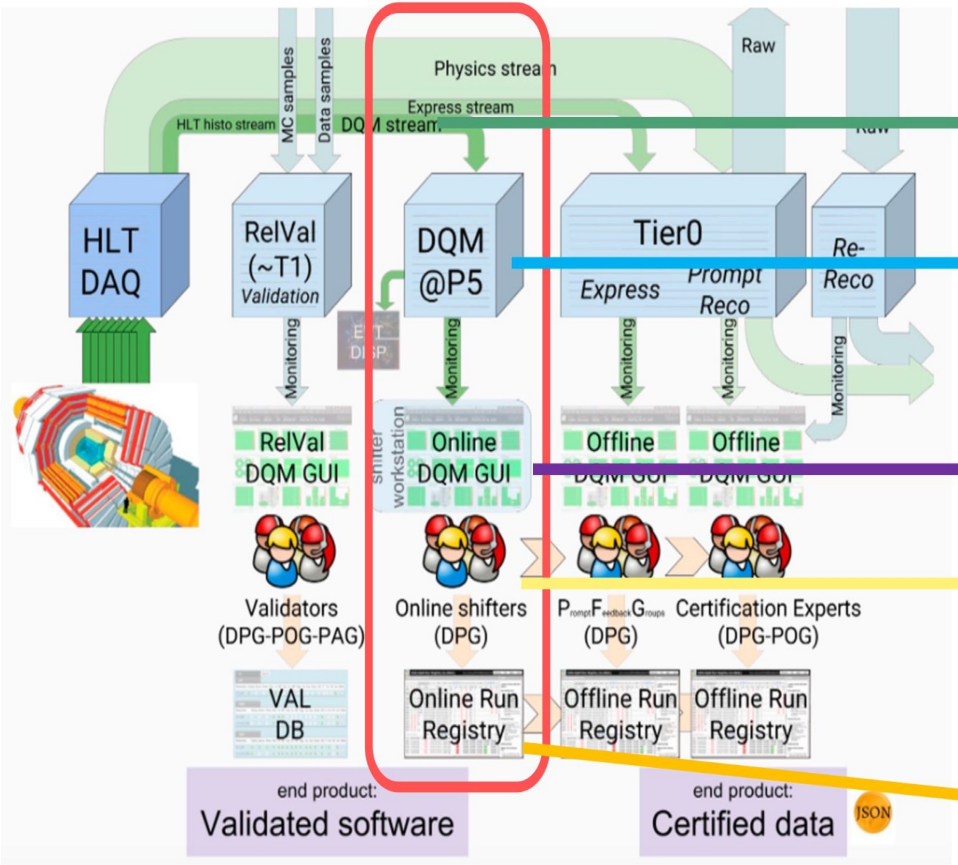



Assuming we have 3 candidate tracks with quality q(c)>q(b)>q(a).
We can have different results depending on the order :

1) b vs c → b removed.
Both a and c survive (no shared hits)

2) a vs b → a removed; then c vs b → b removed.
Only c survives.

# Heterogenous DQM



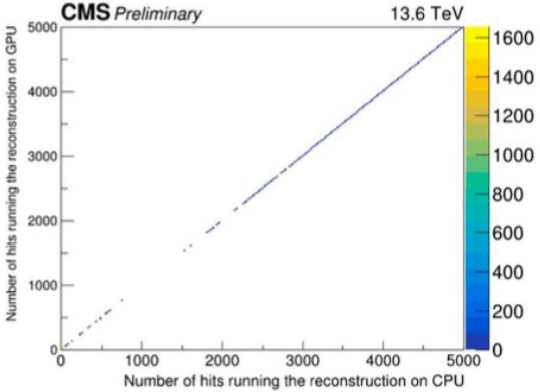

**DQM online** is a system that allows to monitor the CMS sub-detector in nearly real-time during data taking

- Data processed by DAQ and HLT (DQM streams)
  - Delivered on our **P5 machines**
- Data processing by a suite of CMSSW application (clients)
  - On-the-fly reconstruction from raw data.
  - Production of plots of physics **observables** sensitive to detector functioning and operations uploaded in the DQM Online GUI.
- Plots are inspected by DQM shifters with the support of the DQM DOCs 24/7
  - Issues spotted are reported to subsystem experts and SL
- Results of the procedure are recorded in the Run Registry

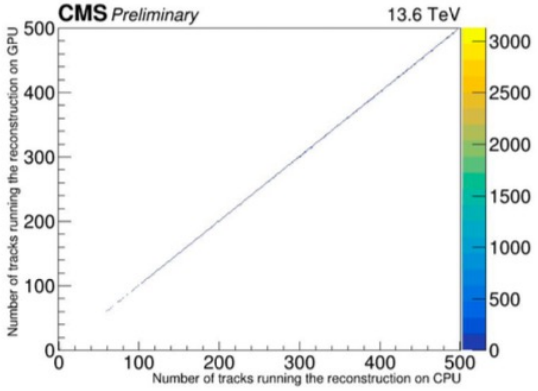
Specific streams added to monitor CPUvsGPU on an event by event basis.

# Event by event comparisons

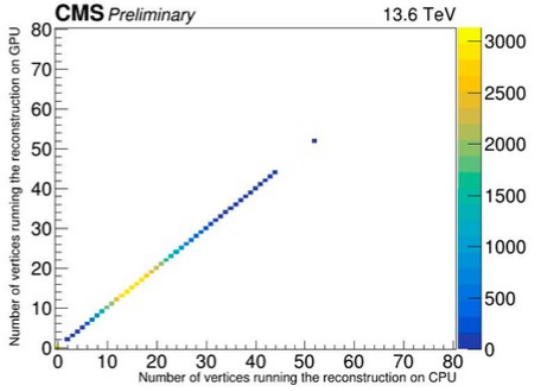
- Big effort in reducing the CPU / GPU difference.
- The current differences between CPU/GPU objects are very small
- Online DQM system allow to monitor the differences in real time!




*Comparison of the number of reconstructed hits (rechits) in the pixel detector per event*




*Comparison of number of pixel tracks per event*



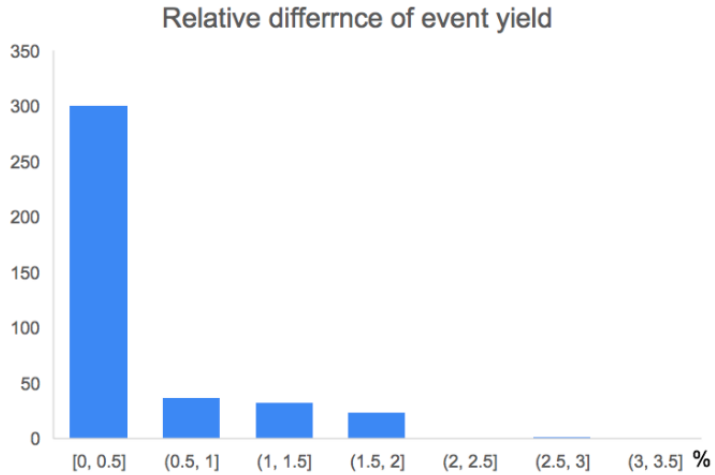

*Comparison of number of pixel vertices per event*

# Still ...

- HLT menu made of ~700 HLT paths. Each HLT path is made of sequences and filters.

- Even though the difference in the objects are very small, we saw differences of ~1% in the trigger results of few HLT paths.

- Considering the HLT with a rate > 5 Hz (~400 paths):
  - 185 paths have no difference at all
  - 150 paths have difference on <1% of events
  - 60 paths have difference on >1% of events
  - 4 paths have difference on >2% of events




Bottom line: we (as CMS) are discussing on how to deal with this. The path seems we will need to take most of these in account as irreducible and quote them.

# Offline

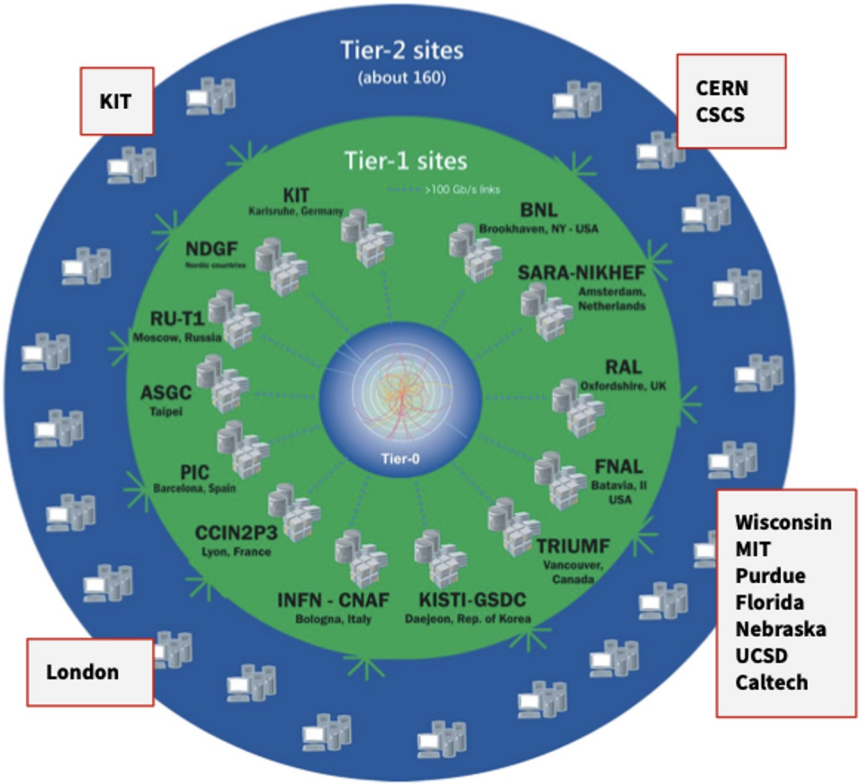
# Heterogenous Offline

Overlap between heterogeneous algorithms running online with their offline counterparts

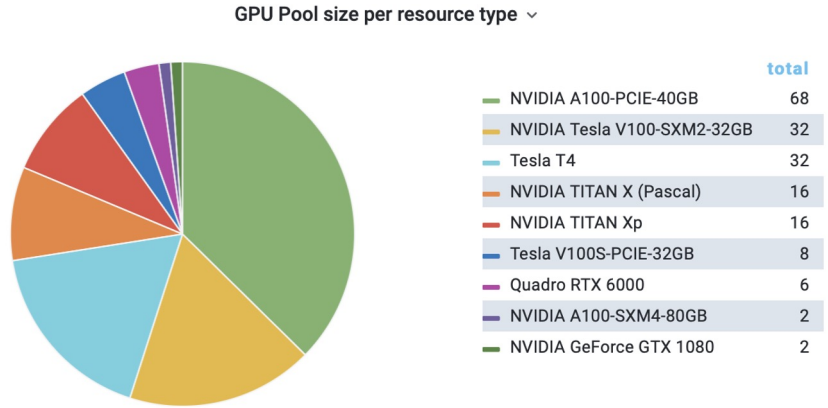
- Pixel Tracks and vertices are not directly used offline, but they provide seeds for the Kalman Filter
  - Phase-2 pixel reconstruction will be replaced by Patatrack pixel reconstruction in 2023
- For Phase-2 HGCAL replaces the ECAL-HCAL endcap calorimeters
  - HGCAL Reconstruction by the TICL framework will be completely ported to Alpaka in 2023.
- Developments ongoing for heterogeneous tracking in the Outer Tracker.
- Overall aiming towards a 10% offload of Phase-2 offline reconstruction, and common modules for Run 3 in the next months.


*CPU time for offline Phase-2 reconstruction in ttbar events with an average of 200 overlapping proton-proton collisions as of CMSSW_13_1_0_pre3*

# CMS GPU Resources

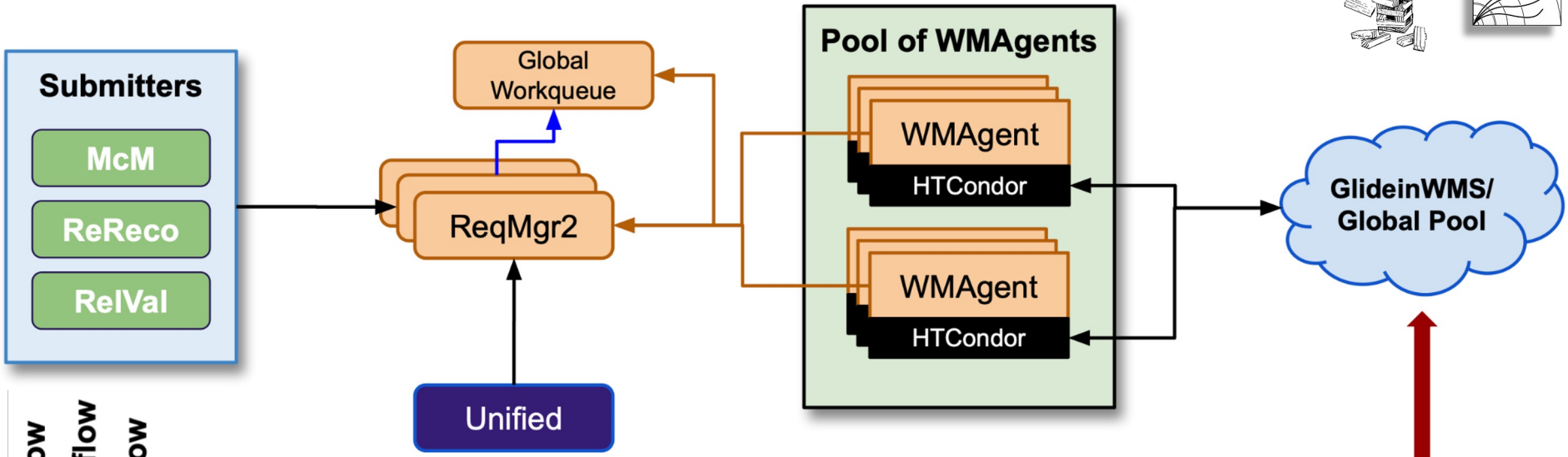



- Many Tier-1 and Tier-2 sites are already equipped with GPUs.
- Most are opportunistic but a few are dedicated to CMS.



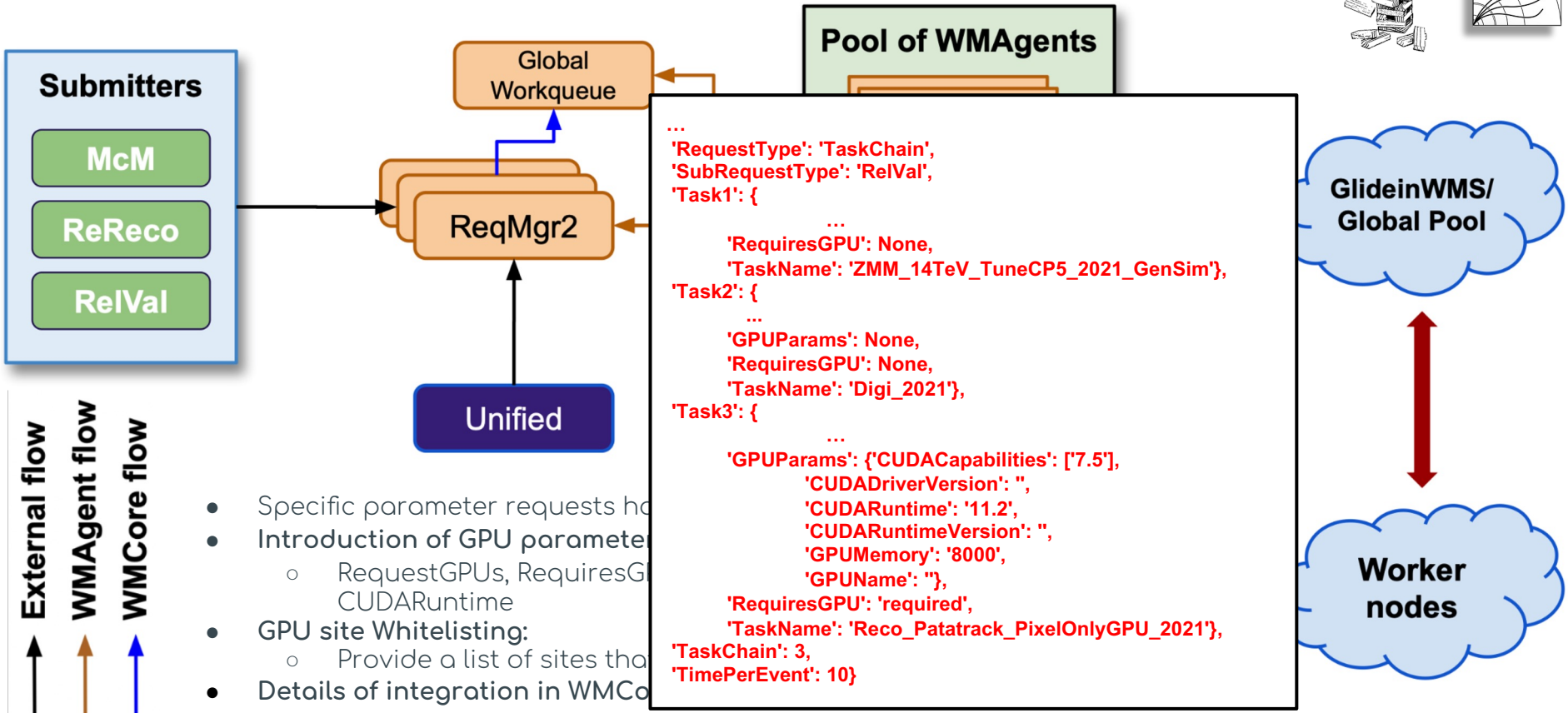

from the dashboard

# WM Infrastructure




- Specific parameter requests have been added for workflows that require GPUs
- **Introduction of GPU parameters :**
  - RequestGPUs, RequiresGPU, GPUMemoryMB, CUDACapability, CUDARuntime
- **GPU site Whitelisting:**
  - Provide a list of sites that are equipped with GPUs (see the dashboard)
- **Details of integration in WMCore** Twiki and summary in this CHEP talk
  - Additional work for improving GPU support is on-HgToCinogndor

# WM Infrastructure




- Specific parameter requests ha
- Introduction of GPU parameter
  - RequestGPUs, RequiresGl
    CUDARuntime
- GPU site Whitelisting:
  - Provide a list of sites tha
- Details of integration in WMCo
  - Additional work for improving GPU support is on-HgToCinogndor

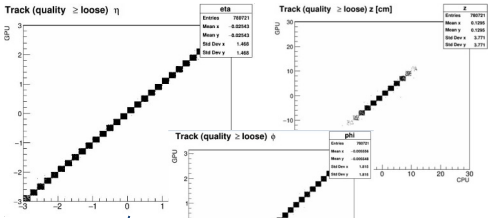
# CMS GPUs Test in Production (for Validation)

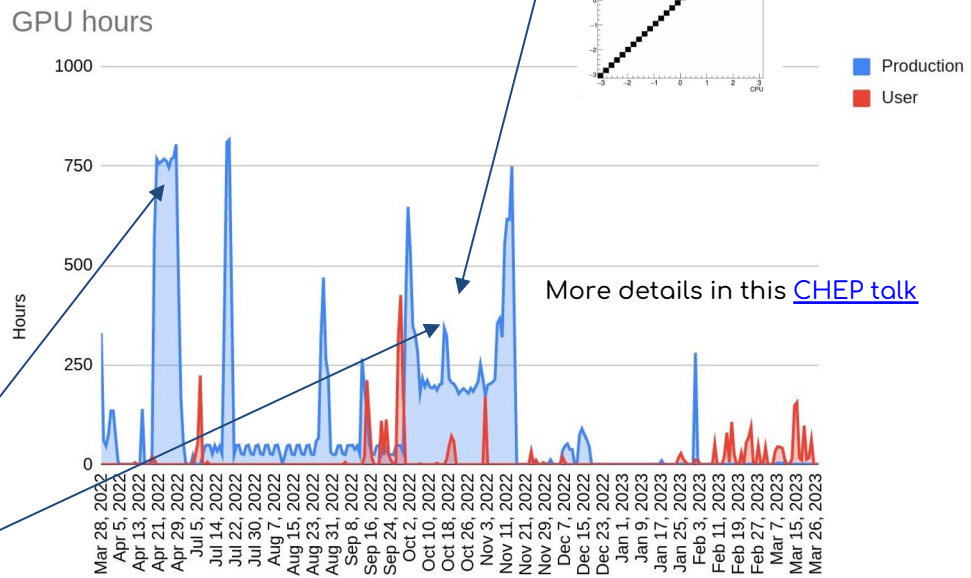
- GPU workflows are already running in central production
  - The HLT configuration was utilized
  - On GridKa TOpAS cluster (24x A100, 24x V100S, 8x V100, expandable to KIT HPC cluster) and Wisconsin (35xT4)

- Example of the usage of the GPU cluster at the Wisconsin Tier-2 site
  - Have run both Data and MC wfs

Large scale validation of CMS reconstruction code before it was deployed and used for data-taking

Release Validation workflows to validate updates in GPU reconstruction code



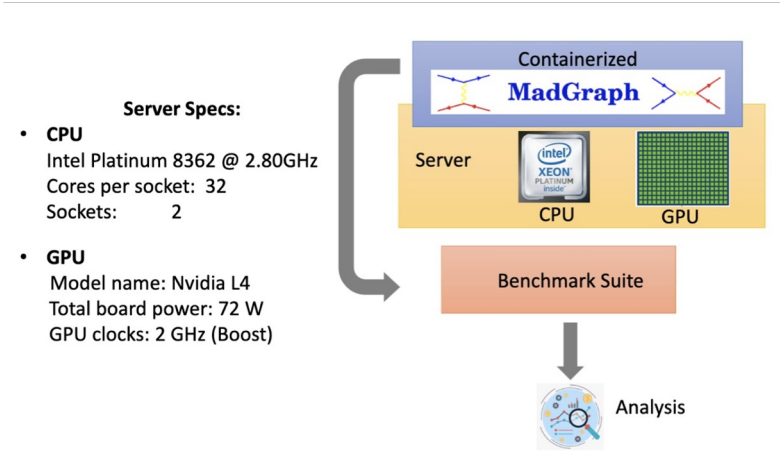




More details in this CHEP talk

# Open questions and future challenges - I

1. **Use of GPUs in CMS MC production multi-step workflows?**
   - Multi-step jobs (multiple cmsRun executions with different settings per job), not all of them capable of making use of the GPUs. Implemented in WM, need some adjustments to work for the SI.
   - See developments in [dmwm/WMCore#11689]

2. **Benchmarking of GPUs**
   - A requirement for pledge definition and resource acquisition.
   - Progress has been done recently using a MadGraph CPU+GPU workflow. Benchmarking including also power consumption.
   - Full [HEPScore23] team report.




by D. Giordano and K.Tuteja

See [Brokering to heterogeneous resources in the CMS Computing Grid] by A. Pérez-Calero Yzquierdo

# Open questions and future challenges - II

3. **Predictable workflow runtimes**, a key parameter for an efficient matchmaking of jobs to slots:
   - Requires benchmarking.
   - Hard because of the high diversity among GPU types.

4. **GPU usage accounting:**
   - Also requires GPU resource benchmarking
   - Could use HTCondor's GPUsAverageUsage as proxy
     i. Cron job uses the NVIDIA driver and tools libraries to query statistics on all of the GPUs
     ii. Generate usage report back to the slot info and payload job classad

See Brokering to heterogeneous resources in the CMS Computing Grid by A. Pérez-Calero Yzquierdo

# One more thing

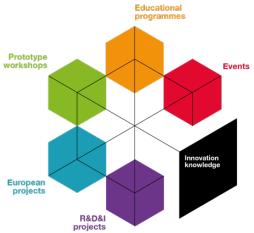
# Engaging the community and partners

- The «sociological» factor here is of key importance. It will be difficult to find the sweet spot. **It goes under the risk column.**

- A working example/history of engagement and training comes from the Patatrack R&D project:

  - Continuous support throughout the development
  - Thanks to experts and the support of partners like CERN openlab, Flatiron Institute, Intel, NVIDIA, E4
  - Three/four times per year CMS Patatrack Team organize the Patatrack Hackathon. Main themes: algorithms and implementations, performance portability, heterogeneous computing, machine learning, software engineering.
  - Huge Boosts of productivity, perfect for newcomers.
  - Building links and a community with common interests.

➢ Extend beyond experiment boundaries?



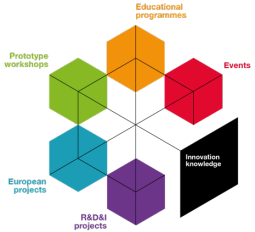
# Engaging the community and partners

- The «sociological» factor here is of key importance. It will be difficult to find the sweet spot. **It goes under the risk column.**
- A working exan
  from the Patatr
  - Con

14 Hackathons organized so far with 30 participants
Foster a familiar environment: about 200kg of pasta cooked in total....

Flatiron Institute, Intel, NVIDIA, E4

openlab




NVIDIA


**Questions?**

# Back up 💾