Usage of GPU for online data processing: The experience of ALICE and LHCb

David Rohr, Daniel Hugo Campora Perez
drohr@cern.ch, dcampora@cern.ch

CERN EP Software Seminar

11.12.2019

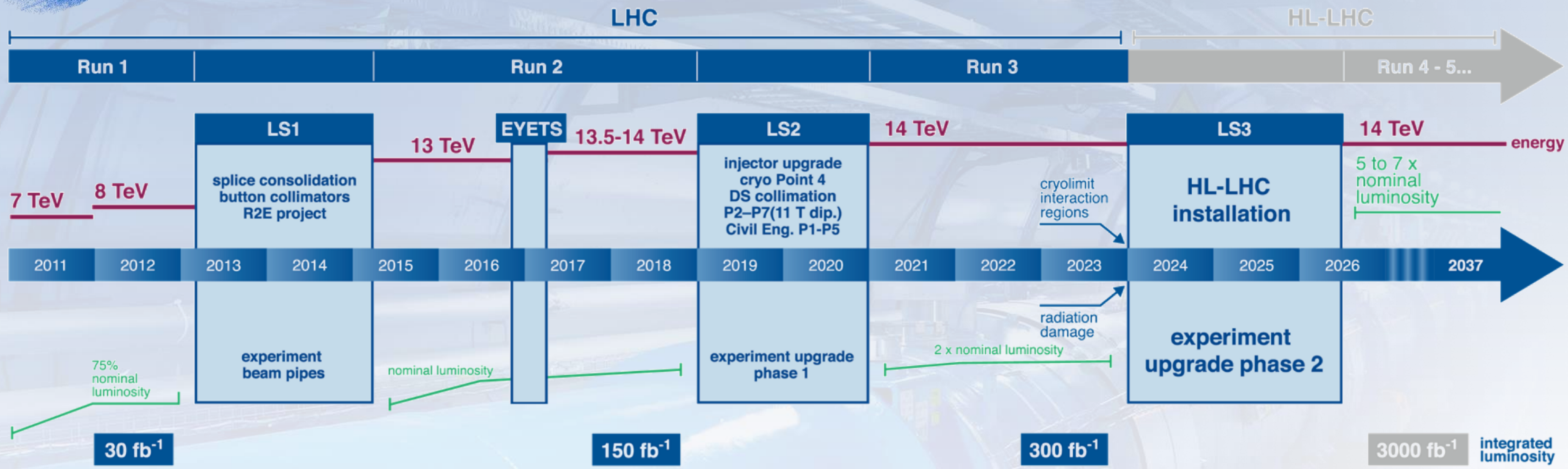
Outline

- **Overview of upgrades of LHC and the experiments:**
  - What are the upcoming challenges for ALICE and LHCb?
  - What do the online processing approaches from ALICE and LHCb have in common?
- **Short introduction to GPUs:**
  - Why should we use GPUs and what can we gain?
- **The experience of ALICE**
- **The experience of LHCb**
- **Conclusion**

COMPUTING FOR THE UPGRADES OF THE LHC EXPERIMENTS

David Rohr, drohr@cern.ch

LHC Upgrade Schedule



- LS2 LHC upgrade: heavy ion rate: >50 kHz in Run 3 (>10 kHz now), boost pp collision rate by small factor.
- LS3 LHC upgrade: HL-LHC era, boost pp collision rate by factor 5 – 7 in Run 4.
  - Highest pp luminosity only for ATLAS and CMS – their detectors are upgraded for Run 4 accordingly.
  - ALICE and LHCb perform a major upgrade for Run 3 now.
- Run 3 is adiabatic increase for ATLAS / CMS, with no increase for ALICE.

Online / Offline Computing in ALICE / LHCb in Run 3

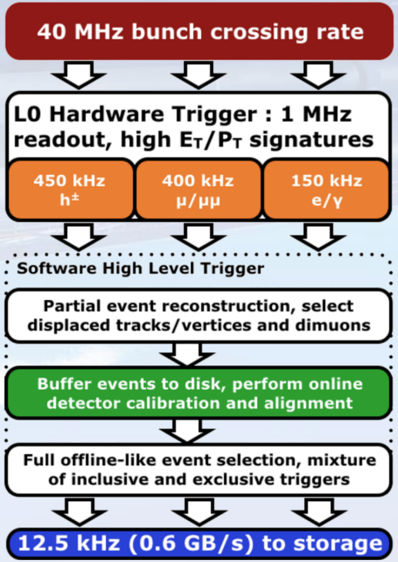
- **LHCb**
  - First phase of trigger (**HLT1**) during data taking.
  - Second phase of trigger (HLT2) when there is no beam.

Common strategy:

- 2 phase processing with disk buffer

- Full processing in software

Run 2



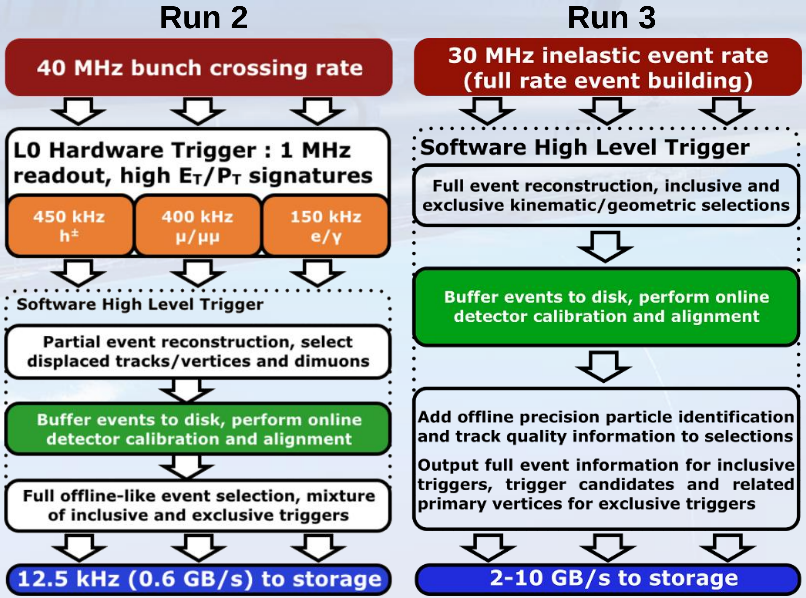
Online / Offline Computing in ALICE / LHCb in Run 3

- **LHCb**
  - First phase of trigger (**HLT1**) during data taking.
  - Second phase of trigger (HLT2) when there is no beam.

Common strategy:

- 2 phase processing with disk buffer

- Full processing in software



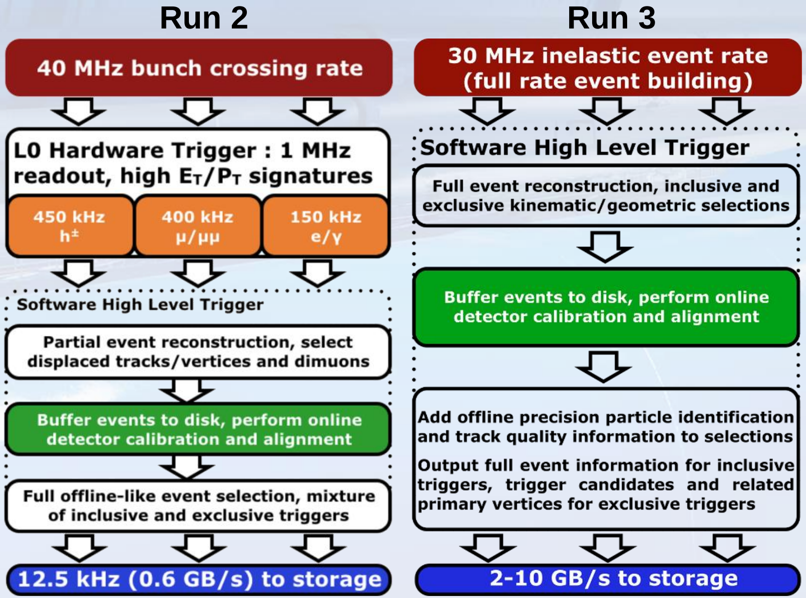
Online / Offline Computing in ALICE / LHCb in Run 3

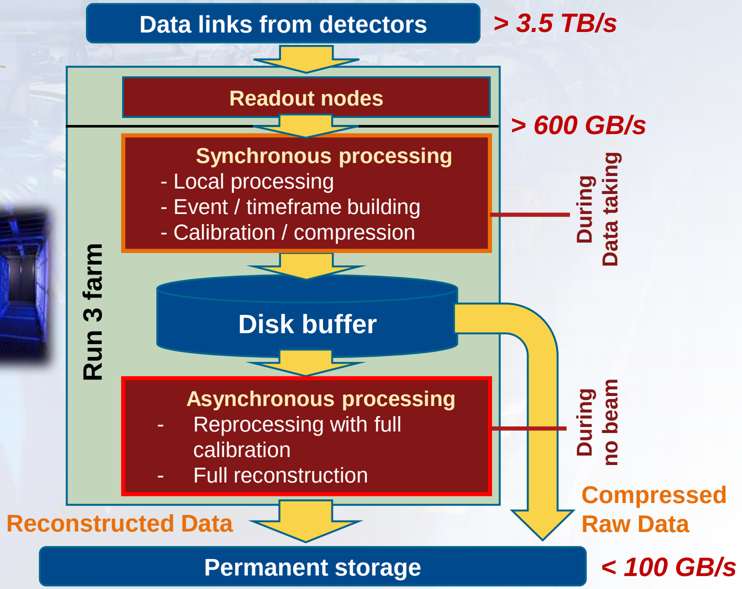
Common strategy:

- 2 phase processing with disk buffer
- Full processing in software

- **LHCb**
  - First phase of trigger (HLT1) during data taking.
  - Second phase of trigger (HLT2) when there is no beam.

- **ALICE**
  - Synchronous (online) processing for data compression and calibration.
  - When not taking data → Asynchronous (offline) processing with final reconstruction.





Comparison of processing, data rates and sizes

	ALICE (Pb-Pb)		LHCb		ATLAS		CMS	
	Run 2	Run 3	Run 2	Run 3	Run 2 / 3	Run 4	Run 2 / 3	Run 4 (PU 140/200)
Luminosity	~10 kHz	50 kHz	$4*10^{32}$	$2*10^{33}$	$2.14*10^{34}$	$5–7.5 *10^{34}$	$2.14*10^{34}$	$5–7.5 *10^{34}$
Hardware trigger	500 Hz – 2 kHz	50 kHz continuous	1 MHz	- / Full 30 MHz bunch crossing rate	95 kHz	1 MHz (can evolve to 4)	100 kHz	500 / 750 kHz
HLT Accept	No rejection	No HLT	12.5 kHz	>100 kHz	1 kHz (< 2)	10 kHz	1 kHz	5 / 7.5 kHz
Raw Data Rate into HLT	45 GB/s (w. ZS)	3 TB/s (w.o. ZS)	55 GB/s	4 TB/s (w. ZS)	29 GB/s (260 GB/s L1)	2.6 TB/s (5.2 TB/s L1)	1.6 TB/s (event network)	23 / 44 TB/s (event network)
Data stored	~10 GB/s	Up to 100 GB/s	0.6 GB/s	2-10 GB/s	2.4 GB/s	50 GB/s	5 GB/s	32 / 61 GB/s
Data Buffer	~1 PB DAQ buffer to Tier0	~60 PB (one year of compressed data), up to 100 GB/s	~12 PT	~100 PB (two weeks of HLT1 accepted raw data, 150 + 150 GB/s read/write.	1.5 TB events + 48 hours to Tier 0	36 PB, 48 hours + L1 to HLT	12 TB (RAM disk, events before HLT, 60s)	171 / 333 TB (events before HLT, 60s)

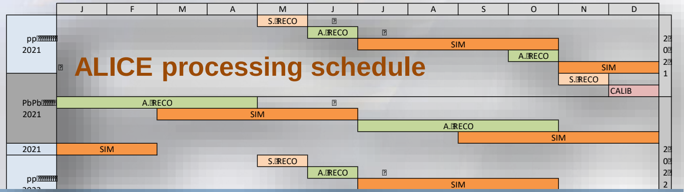
- **ALICE will take 100x more events, but only minimum bias, all other experiments collect 10x more statistics.**
  - ALICE investigates the option to run in a software triggered mode at higher rate during some time of a year.
- **ALICE and LHCb** process all data in software, use large disk buffers to hold large amount of (compressed raw data) for processing in the online farm when there is no beam.
- **ATLAS and CMS** have much higher luminosity, full readout of front-end at bunch-crossing rate and processing in software not feasible and not cost effective.
- **ALICE** features high data rate during Pb-Pb (due to TPC), collects large amount of data in only few weeks.

Comparison of ALICE and LHCb data processing in Run 3

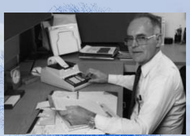
- **Both experiments do full two-stage online processing at bunch crossing rate, with disk buffer and offline quality output.**
  - Input data rates: >3TB/s for both – zero-suppressed for LHCb (factor 4) – raw for ALICE with zero-suppression in FPGA down to ~1TB/s.
  - Calibration: Full calibration available for Asynchronous stage / HLT2.
    - Online calibration with feedback loop tested in the ALICE HLT in Run 2, under study for Run 3, Velo alignment calibration in LHCb HLT1.
  - Event building: A set of input nodes (FLP / DAQ) receives the detector links via PCIe40 FPGA card.

Similar

  - ALICE sends time frames (TF) of 23ms from the FLP to Event Processing Nodes, where events are merged, build, and reconstructed.
    - LHCb first builds the events internally inside the DAQ via a fast network, then ships them to the Event Filter farm via a broad network.
      - Network transfers synchronized via software to avoid congestion.
      - Many events are coalesced (similar to TF).
      - Could switch to similar event building as ALICE if needed.
  - Cluster: Input: ~170 DAQ nodes @ LHCb, ~200 FEP nodes @ ALICE
    Processing: ~2000 Event Filter nodes @ LHCb, ~1500 GPUs in EPNs @ ALICE.
  - Disk buffer: LHCb buffers up to 3 weeks, exploits turnaround, TS, MD periods, runs MC at YETS. ALICE buffers 1 year, exploits also YETS.
    - ALICE compresses data in synchronous stage, compressed raw data stored to disk buffer and to tape is identical.
      - ALICE has highest data taking output rate of up to 100 GB/s, but only during beginning of Pb-Pb fill.
      - Pb-Pb data parked for processing on disk buffer for 1 year. Compute budget for 2 full reconstructions passes within the year.
      - pp data taking and asynchronous reconstruction in between / in parallel.
      - Part of asynchronous reconstruction can run on GRID.
      - Online farm must be capable to process this data in the synchronous stage.
    - LHCb stores raw data after HLT1 trigger to disk buffer, second trigger rejection in HLT2.
      - LHCb has higher data rate from HLT1 to disk buffer, but not for final storage.



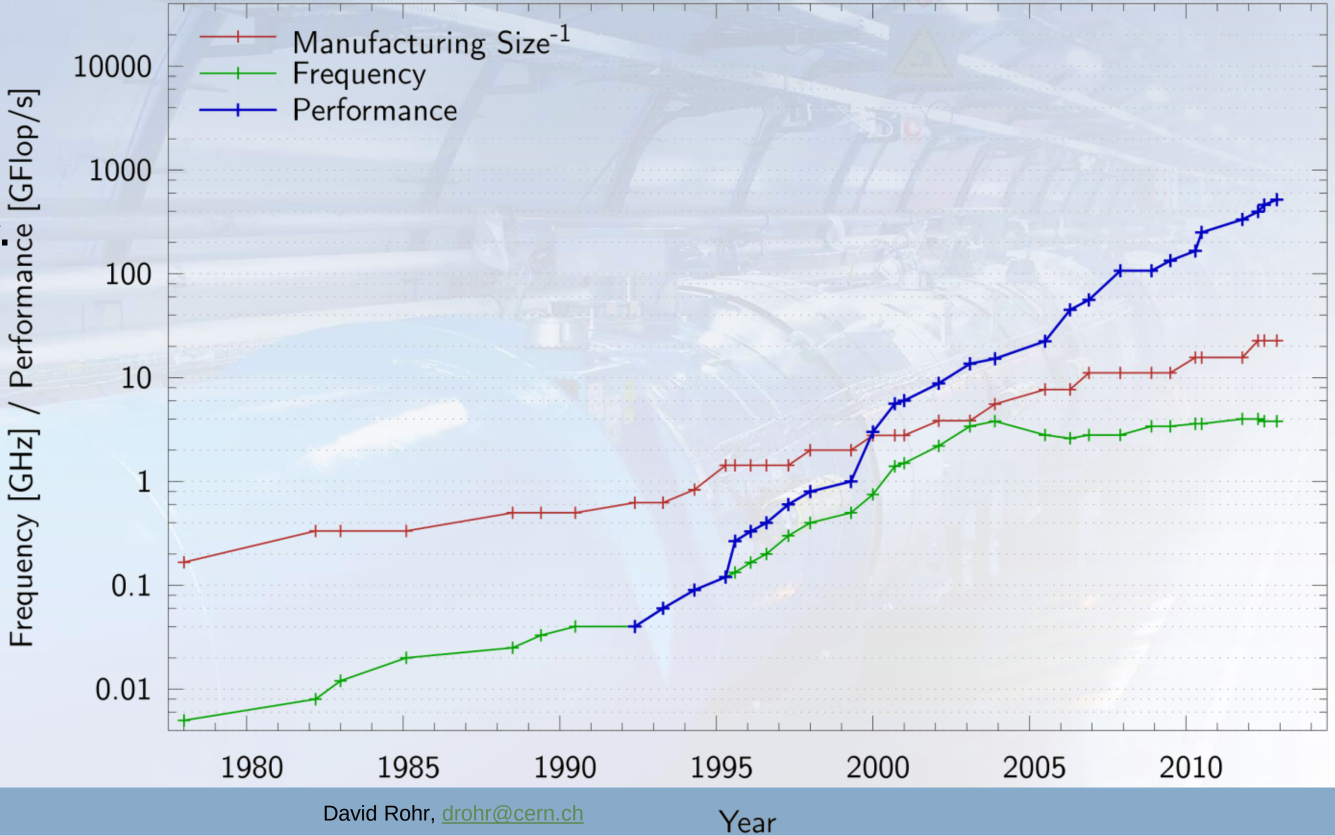
POTENTIAL OF GPUS



Introduction

- Moore‘s Law:
- Manufacturing
- size, frequency,
- and performance
- grow exponentially.

- Frequency began
- to stagnate 2003.

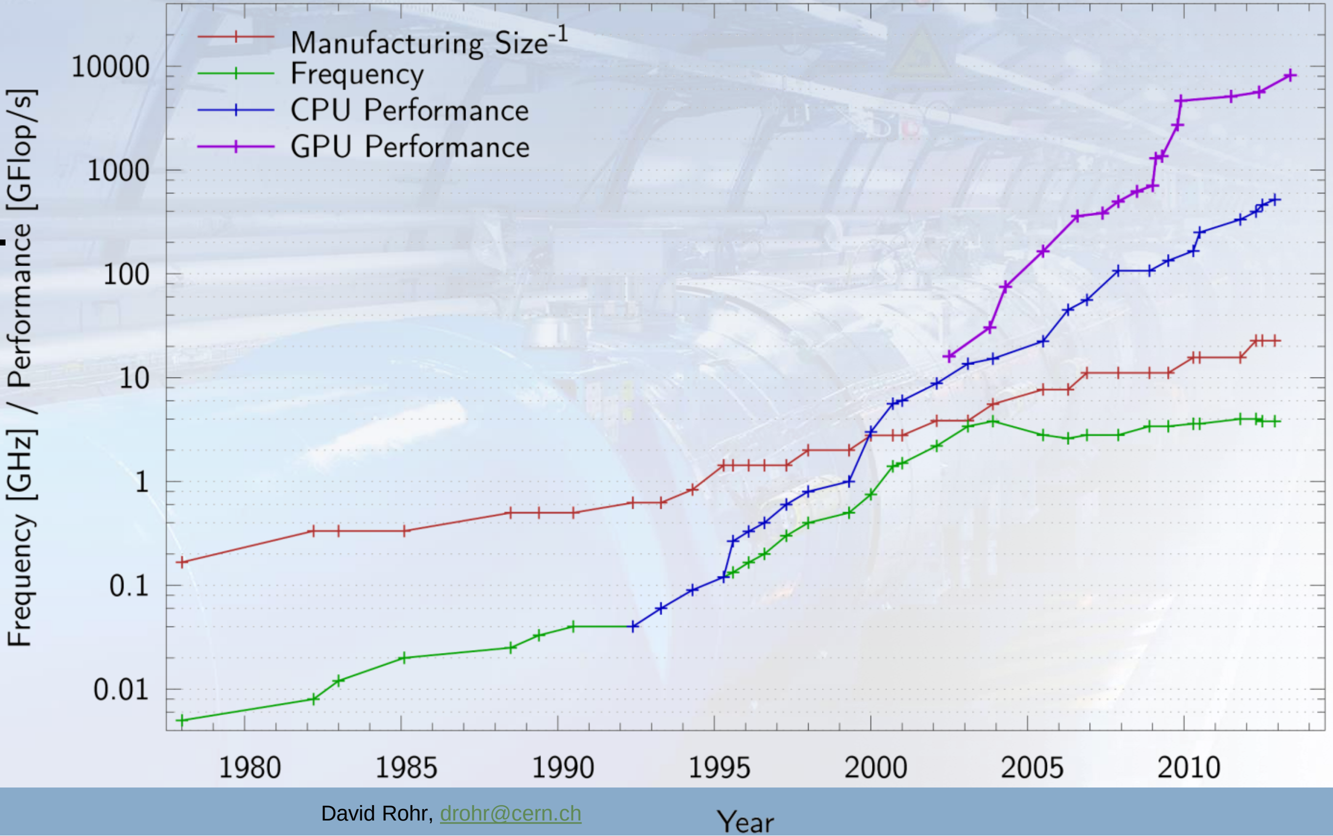


Introduction

- Moore's Law:
- Manufacturing
- size, frequency,
- and performance
- grow exponentially.

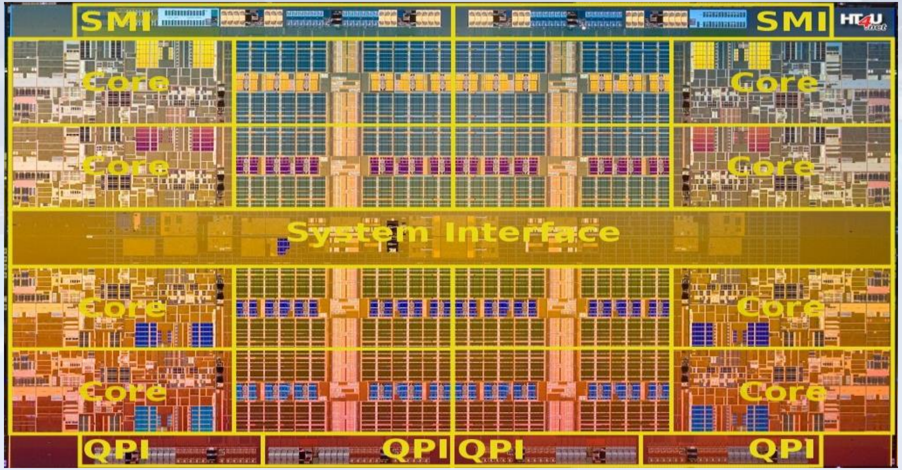
- Frequency began
- to stagnate 2003.

- GPUs are faster
- than CPUs.

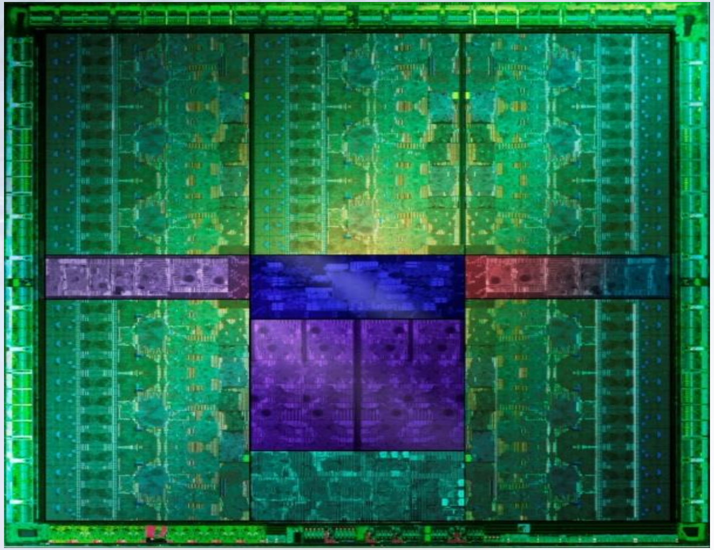


Why GPUs

- GPUs use their silicon for ALUs
- CPUs use their silicon mainly for caches, branch prediction, etc.



Intel Nehalem

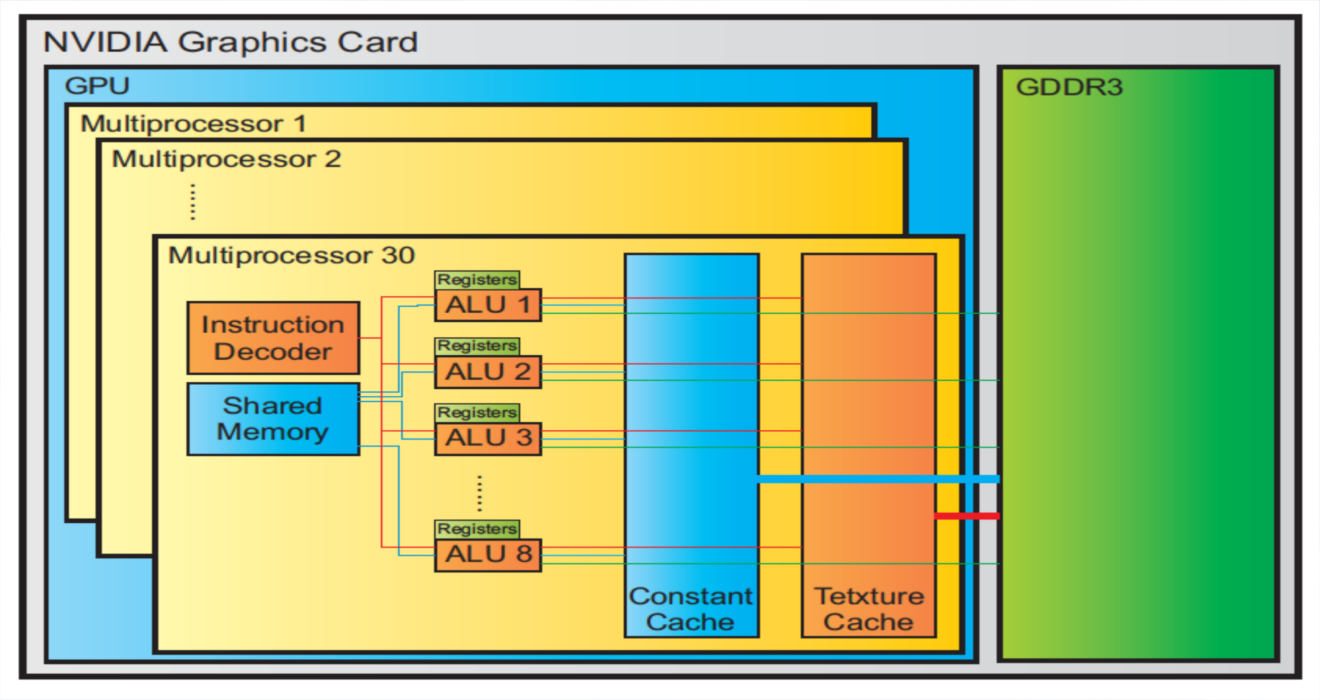


NVIDIA Kepler

Why GPUs

- **CPUs are designed for fast execution of serial programs.**
  - Clocks have reached a physical limit.
    → Vendors use parallelization to increase performance.
- **GPUs are designed for parallel execution in the first place.**
  - The „only“ limit for GPU performance is heat dissipation.
  - GPU clocks are usually lower than they could be.
    - This saves power
    - Hence more hardware can be powered in parallel
      → Better overall performance

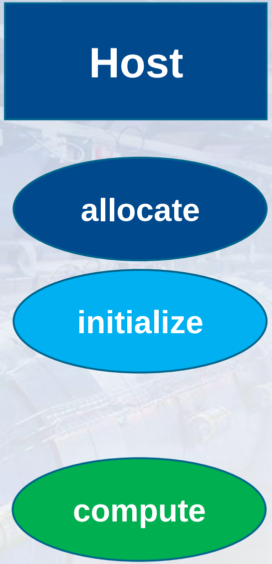
NVIDIA GTX280 GPU



GPU Programming example (stupid addition of 2 vectors)

```
#include <vector>
#define SIZE 1024
int main(int, char**) {
        std::vector<float> hostArray1(SIZE), hostArray2(SIZE);
        for (int i = 0;i < SIZE;i++) { hostArray1[i] = 2 + 3 * i; hostArray2[i] = 4 + 5 * i; }
        for (int i = 0;i < SIZE;i++) { //Computation
                hostArray2[i] += hostArray1[i];
        }
        return 0;
}
```

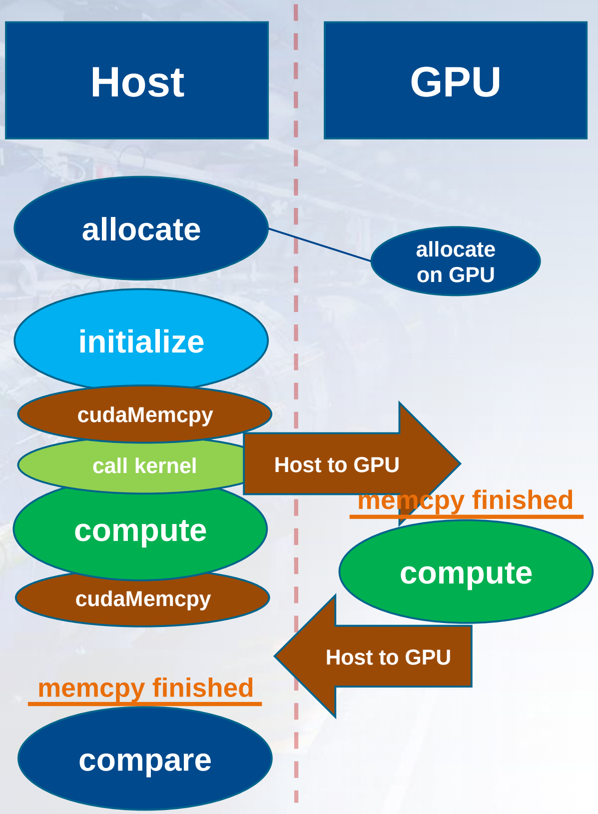
On the CPU



GPU Programming example (stupid addition of 2 vectors)

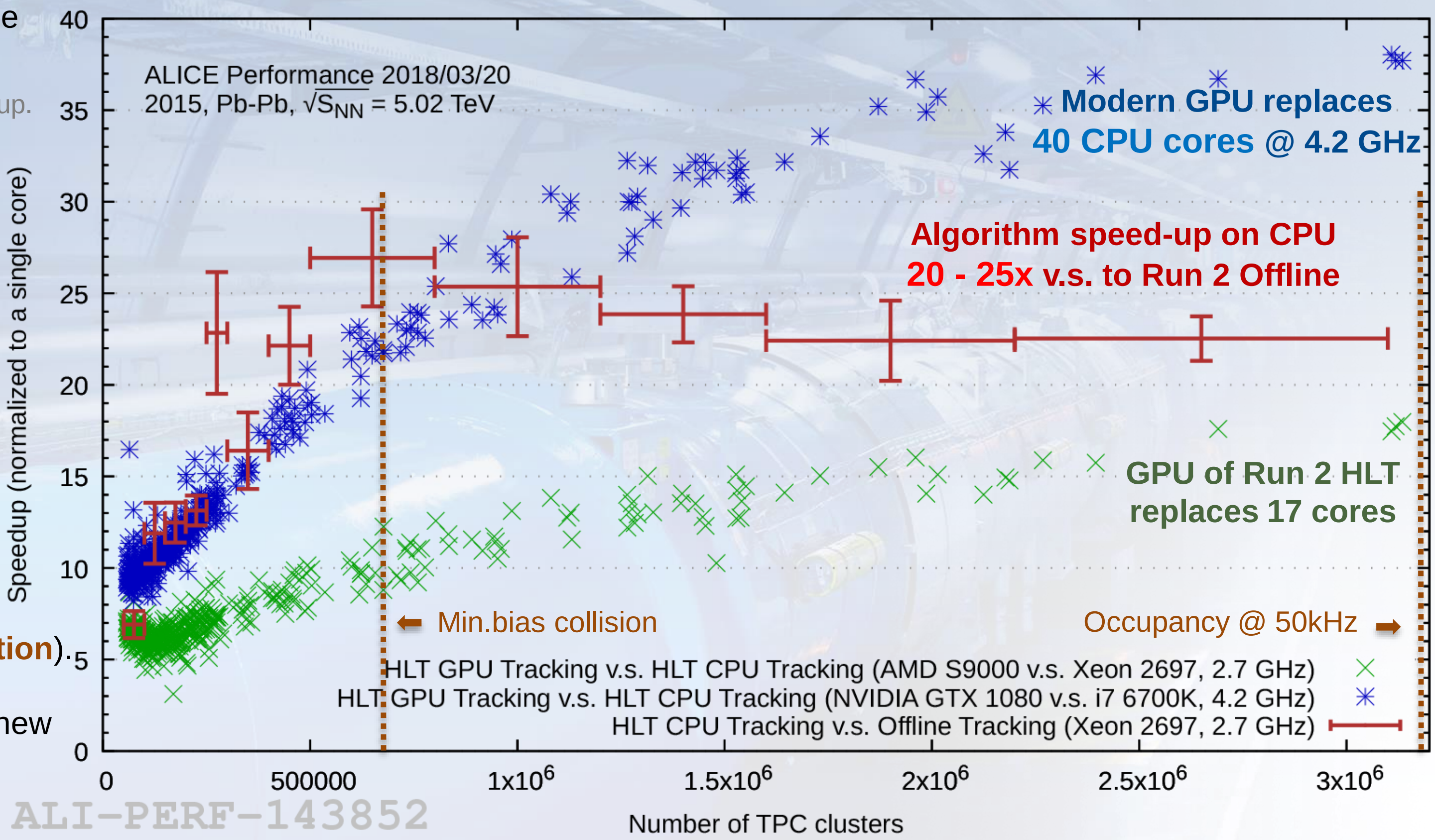
On the GPU

```
#include <vector>
#define SIZE 1024
__global__ void testKernel(float* vec1, const float* vec2, int size) {
        int myId = threadIdx.x + blockIdx.x * blockDim.x;
        if (myId >= size) return;
        for (int i = myId; i < size; i += blockDim.x * gridDim.x) {
                vec1[i] += vec2[i];
        }
}
int main(int, char**) {
        std::vector<float> hostArray1(SIZE), hostArray2(SIZE);
        float *devicePtr1, *devicePtr2;
        for (int i = 0;i < SIZE;i++) { hostArray1[i] = 2 + 3 * i; hostArray2[i] = 4 + 5 * i; }
        cudaMalloc(&devicePtr1, hostArray1.size() * sizeof(hostArray1[0])); //Allocate memory on the device
        cudaMalloc(&devicePtr2, hostArray2.size() * sizeof(hostArray2[0]));
        cudaMemcpy(devicePtr1, hostArray1.data(), hostArray1.size() * sizeof(hostArray1[0]), cudaMemcpyHostToDevice); //Copy buffers to the device
        cudaMemcpy(devicePtr2, hostArray2.data(), hostArray2.size() * sizeof(hostArray2[0]), cudaMemcpyHostToDevice);
        testKernel<<<(SIZE + 127) / 128, 128>>>(devicePtr1, devicePtr2, SIZE); //Launch kernel to add the vectors (add vector 2 onto vector 1)
        for (int i = 0;i < SIZE;i++) { //We do the same computation on the host
                hostArray2[i] += hostArray1[i];
        }
        //We copy back the first vector, which contains the result (cudaMemcpy implies synchronization)
        cudaMemcpy(hostArray1.data(), devicePtr1, hostArray1.size() * sizeof(hostArray1[0]), cudaMemcpyDeviceToHost);
        cudaFree(devicePtr1); //Free CUDA memory
        cudaFree(devicePtr2);
        bool ok = true; //Compare results
        for (int i = 0;i < SIZE;i++) {
                if (hostArray1[i] != hostArray2[i]) {
                        printf("Error at position %d: %f != %f\n", i, hostArray1[i], hostArray2[i]);
                        ok = false;
                }
        }
        if (ok) printf("Result OK!\n");
        return 0;
}
```



TPC Tracking performance

- Speed-up normalized to single CPU core.
  - Red curve: algorithm speed-up.
  - Other curves: GPU v.s. CPU speed-up corrected for CPU resources.
    - How many cores does the GPU replace.
- Significant gain with newer GPU (blue v.s. green).
- GPU with Run 3 algorithm replaces **> 800 CPU cores** Running Run 2 algorithm. (**blue * red**). (**at same efficiency / resolution**).
- We see ~**30%** speedup with new GPU generation (RTX 2080 v.s. GTX 1080)

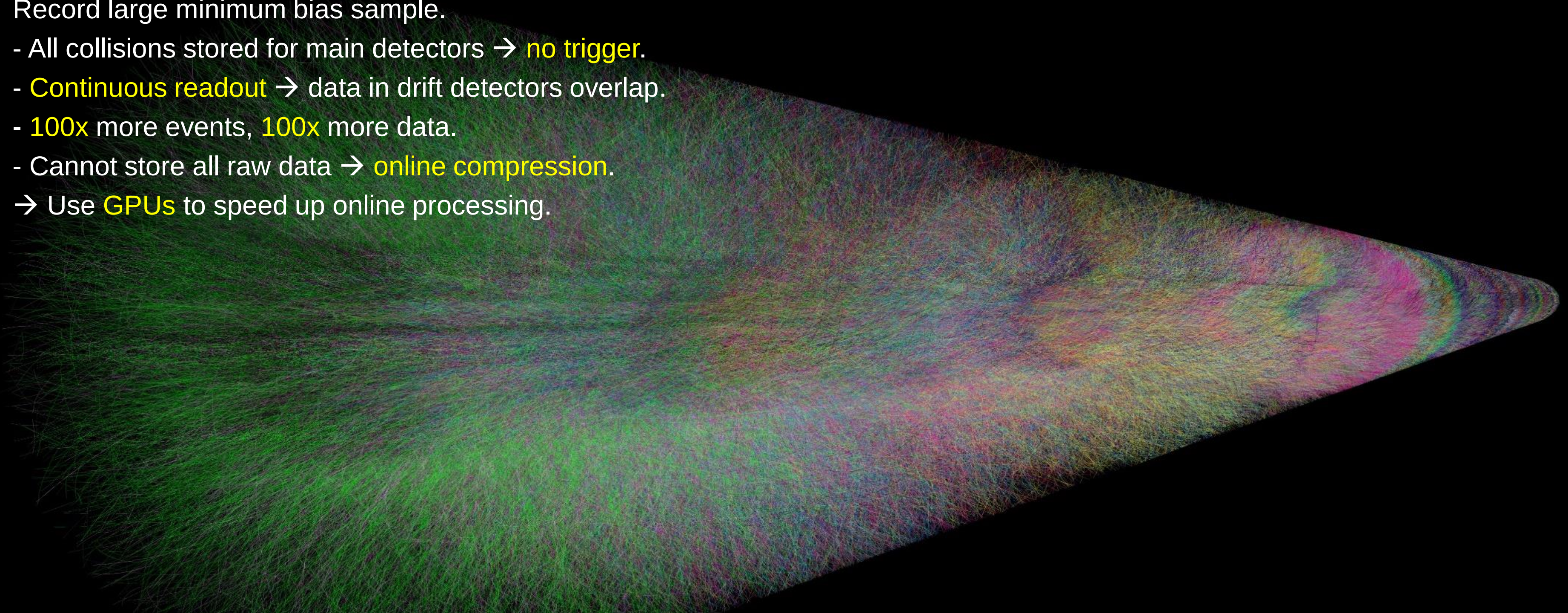


EXPERIENCE OF THE ALICE EXPERIMENT

ALICE in Run 3: 50 kHz Pb-Pb

Record large minimum bias sample.
- All collisions stored for main detectors → no trigger.
- Continuous readout → data in drift detectors overlap.
- 100x more events, 100x more data.
- Cannot store all raw data → online compression.

→ Use GPUs to speed up online processing.

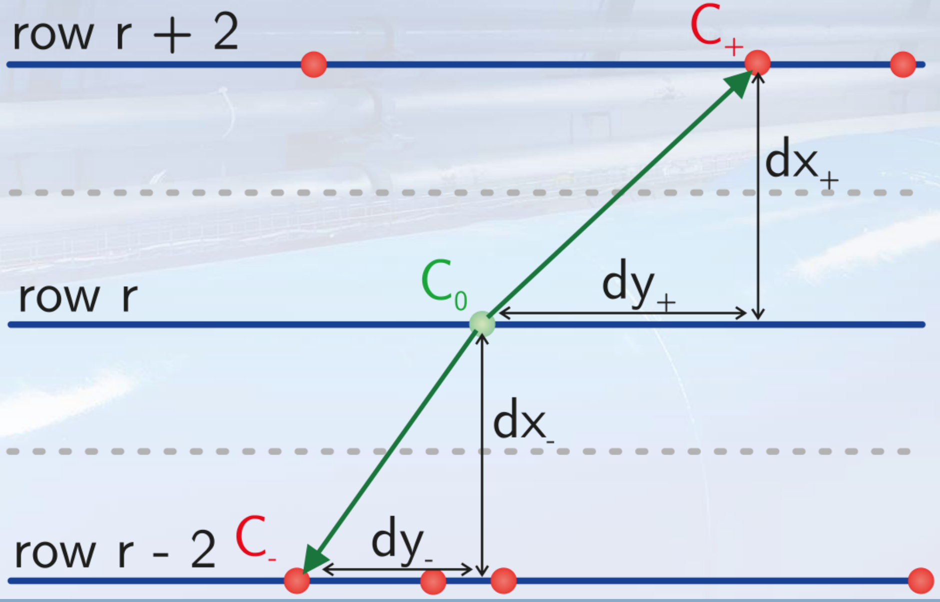


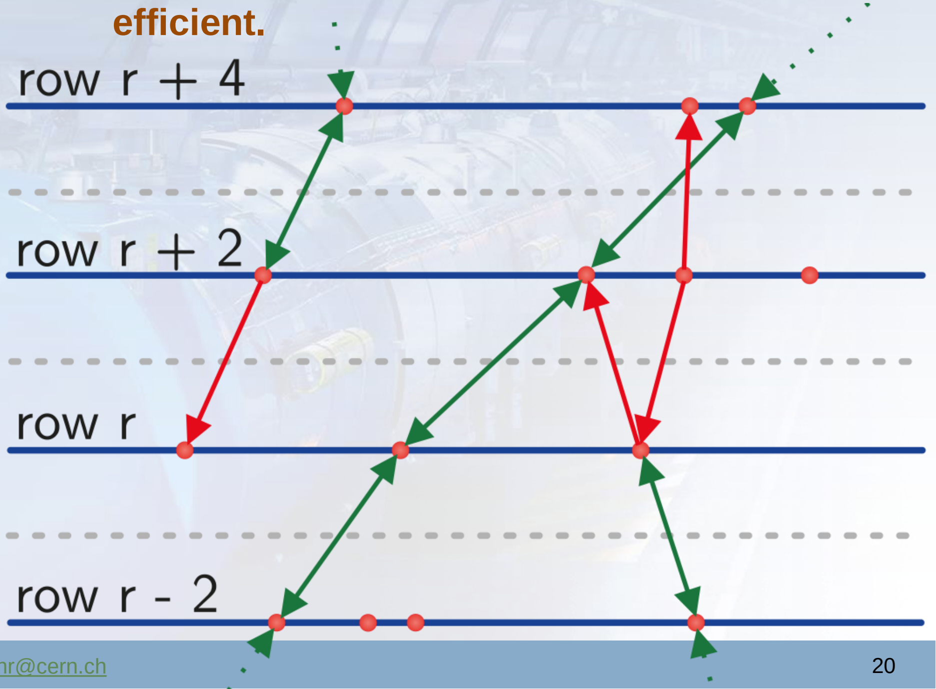
- Overlapping events in TPC with realistic bunch structure @ 50 kHz Pb-Pb.
- Timeframe of 2 ms shown (will be 10 – 20 ms in production).
- Tracks of different collisions shown in different colors.

Step 1 (Seeding)

- **Step 1: Combinatorial seeding**
- **Searches for three clusters composing straight line**
- **Concatenates straight lines**
- **Only step with non-linear runtime.**

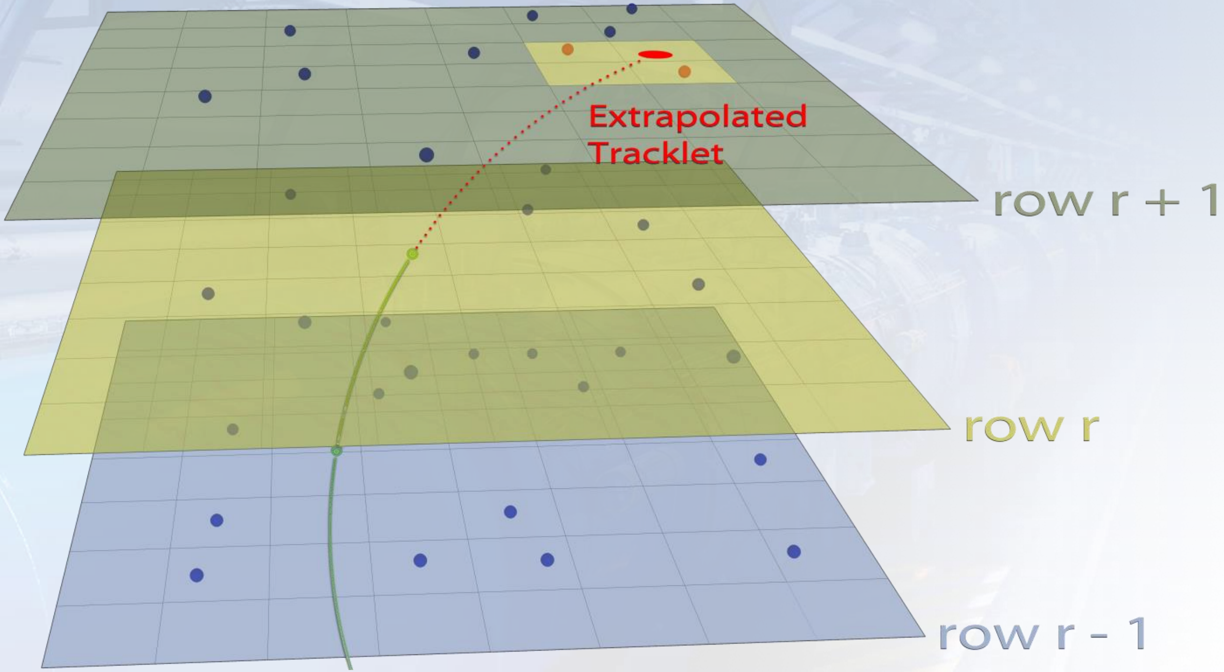
- **Strategy: deal with the combinatorics as early as possible.**
- **Seed everywhere, each track has at least some seedable part, no need to be 100% efficient.**





Step 2 (Track Following)

- **Step 2 (Simplified Kalman Fit):**
- **Track parameters are fit to the seed.**
- **Trajectory is extrapola-ted to adjacent TPC row.**
- **Cluster closest to extrapolated position is found.**
- **Fit is improved with new cluster.**



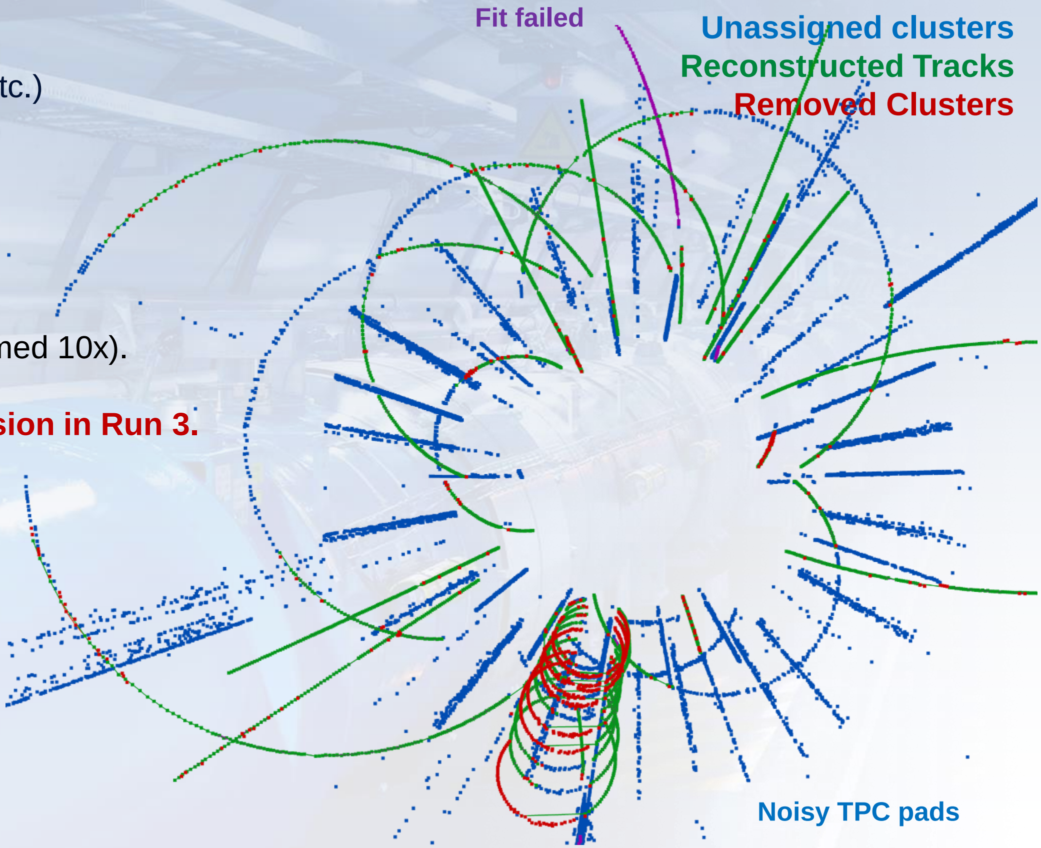
TPC Data Compression

- TPC Data compression involves 3 steps:
  1. Entropy reduction (Track model, variable precision, etc.)
  2. Entropy encoding (Huffman, Arithmetic, ANS)
  3. Removal of tracks not used for physics.

- Steps 1 + 2 implemented for Run 2.
  - Current compression factor **8.3x**.
  - Prototype for Run 3 achieves factor **9.1x** (TDR assumed 10x).

- **Step 3 must close the gap to the required compression in Run 3.**
  - Remove clusters from background / looping tracks.
    - Adjacent to low-p_T track < 50 MeV.
    - Adjacent to secondary leg of low-p_T track < 200 MeV.
    - Adjacent to any track with φ > 70° in the fit.
  - Protect clusters of physics tracks.
    - Not Adjacent to any physics-track (except φ > 70°).

- In addition:
  - Use reconstructed track quantities to reduce entropy.



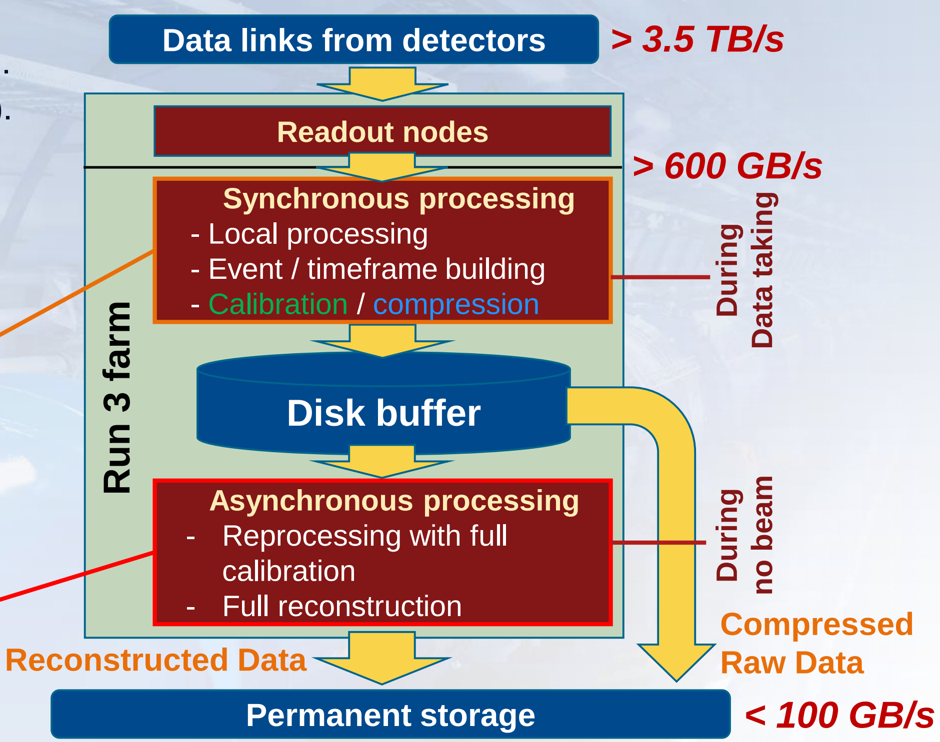
Online / Offline Computing in ALICE in Run 3

- **ALICE computing strategy for Run 3**
  - On-site server farm for **synchronous** (online) processing.
  - When not taking data → used for **asynchronous** (offline).

- **Partial ITS + TPC + TRD tracking for TPC calibration**
  - reduced statistics sufficient
  (TPC calibration based on matching of TPC / ITS / TRD tracks)
- *Other detectors without significant CPU load*

- **Full TPC tracking for TPC compression**
  - cluster to track residuals → better entropy coding
  - removal of tracks not used for physics
- *Entropy coding for other detectors*

Final reconstruction pass with final calibration



Tracking in ALICE in Run 3

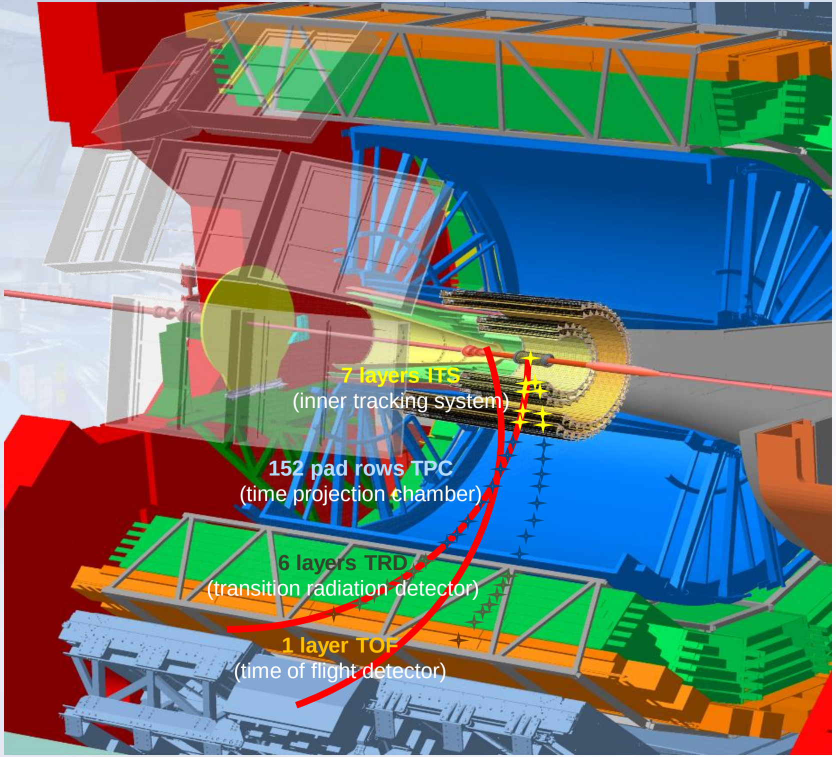
- **Bulk of computing workload:**

Synchronous

- >90% TPC tracking / compression
- Low load for other detectors

Asynchronous

- TPC among largest contributors
- Other detectors also significant



Tracking in ALICE in Run 3

- **Bulk of computing workload:**

Synchronous
- >90% TPC tracking / compression
- Low load for other detectors

Asynchronous
- TPC among largest contributors
- Other detectors also significant

- **ALICE GPU processing strategy**

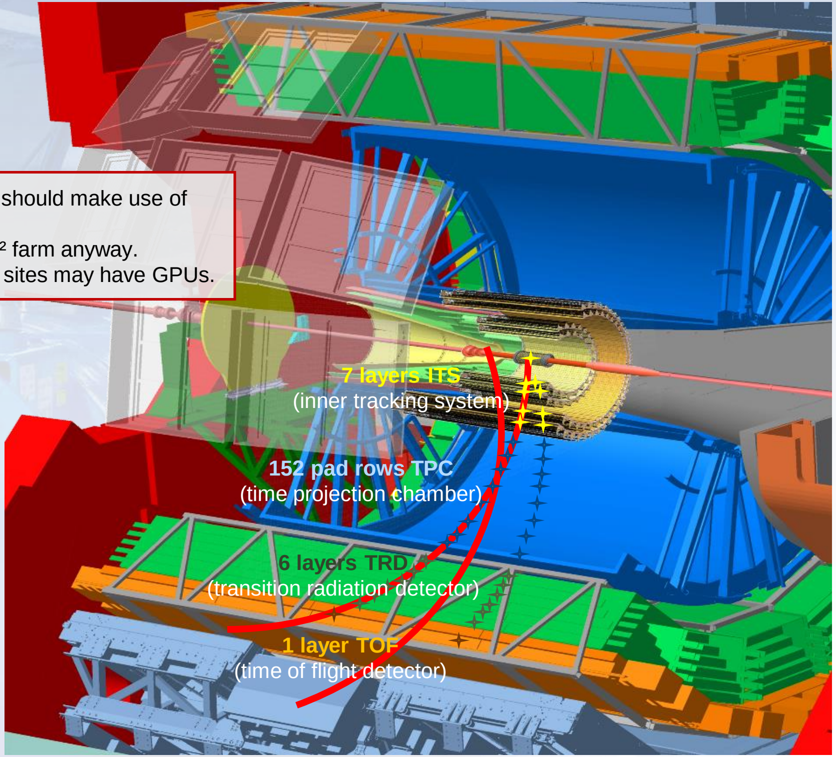
Baseline solution
(almost available today):
TPC + part of ITS tracking on GPU

- **Mandatory** solution to keep up with the data rate online.
- **Defines** number of servers / **GPUs**.

Optimistic solution
(what could we do in the ideal case):
Run most of tracking + X on GPU.

- Extension of baseline solution to make best use of GPUs.
  - Ideally, **full barrel tracking** without ever leaving the GPU.
  - In the end, we will probably be somewhere in between.

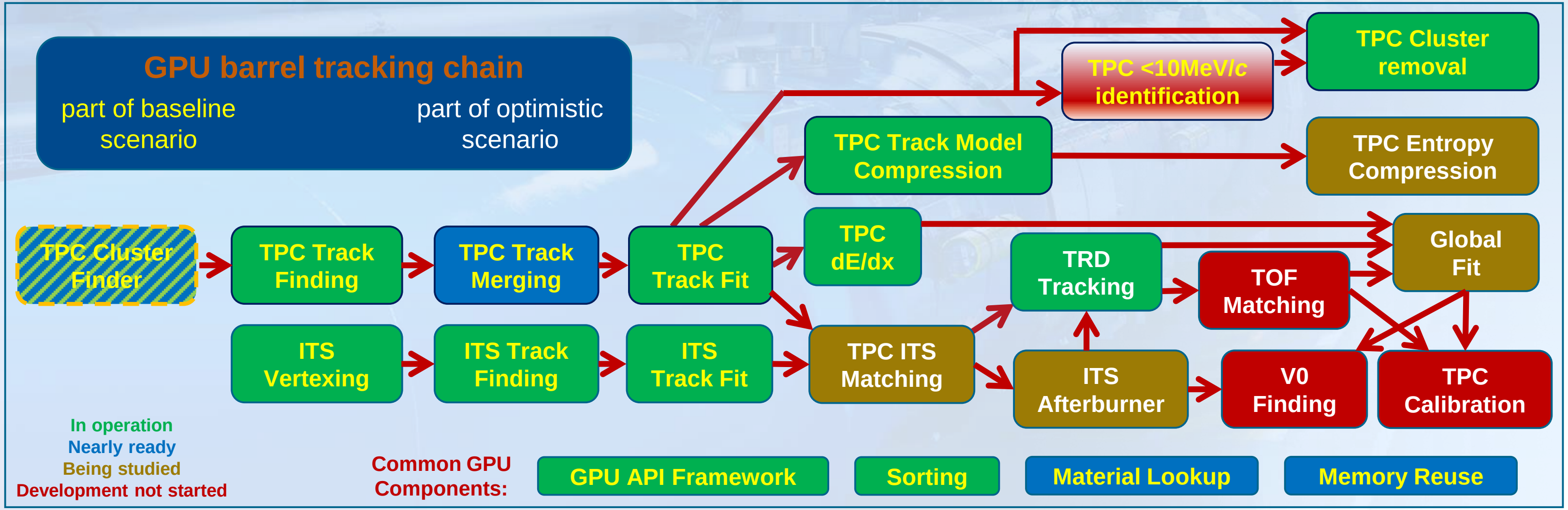
Asynchronous phase should make use of the available GPUs.
- Available in the O² farm anyway.
- Future HPC / grid sites may have GPUs.



Reconstruction steps on GPU (Barrel Tracking)

- **Status of reconstruction steps on GPU:**
  - All TPC steps during synchronous reconstruction are **required** on the GPU.
  - Synchronous ITS tracking and TPC dE/*dx* in good shape, thus considered **baseline** on the GPU.
  - Remaining steps in tracking chain part of **optimistic scenario**, being ported step by step to GPU.
    - Porting order follows topology of chain, to avoid unnecessary data transfer for ported steps – current blocker is **TPC ITS matching**.

TRD tracking / TPC calibration: see poster of Ole Schmidt
Space point calibration of the ALICE TPC with track residuals

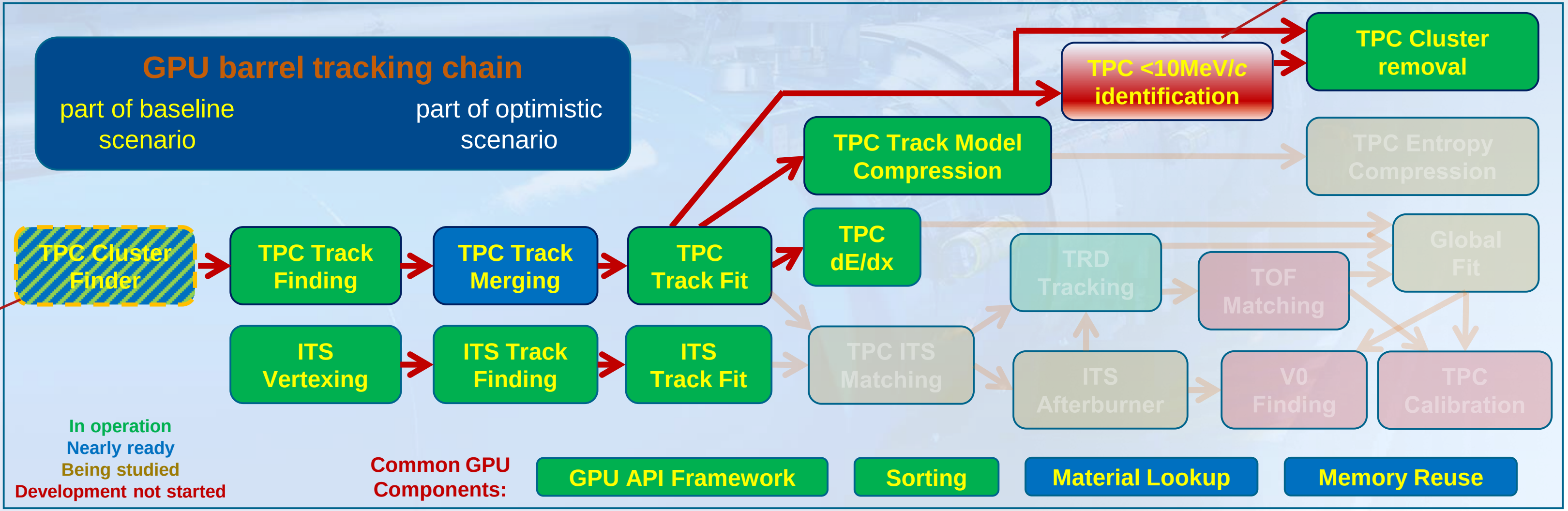


Reconstruction steps on GPU (Barrel Tracking)

- **Status of reconstruction steps on GPU:**
  - Baseline scenario: all steps almost ready

TRD tracking / TPC calibration: see poster of Ole Schmidt
Space point calibration of the ALICE TPC with track residuals

Not clear if necessary on GPU, depends on rejection strategy (see later)

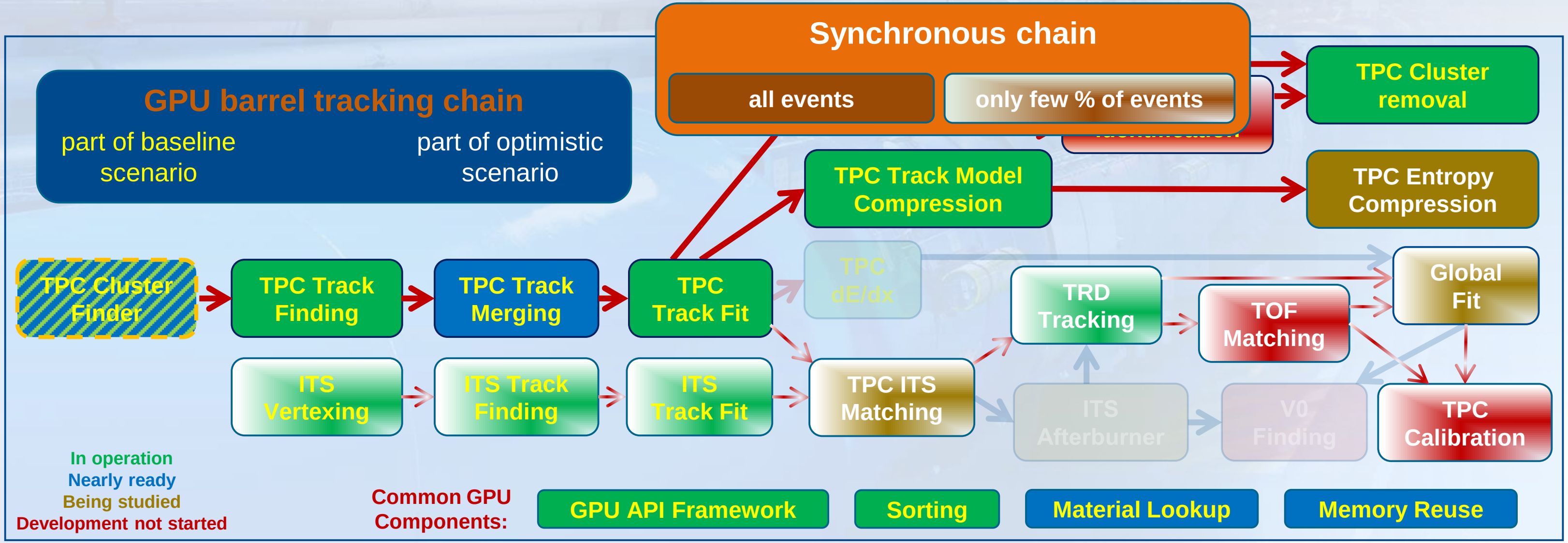


New requirement arose few months ago, since clusterizer does not fit in FPGA.

Reconstruction steps on GPU (Barrel Tracking)

- **Status of reconstruction steps on GPU:**
  - Different reconstruction steps enabled in synchronous and asynchronous reconstruction.

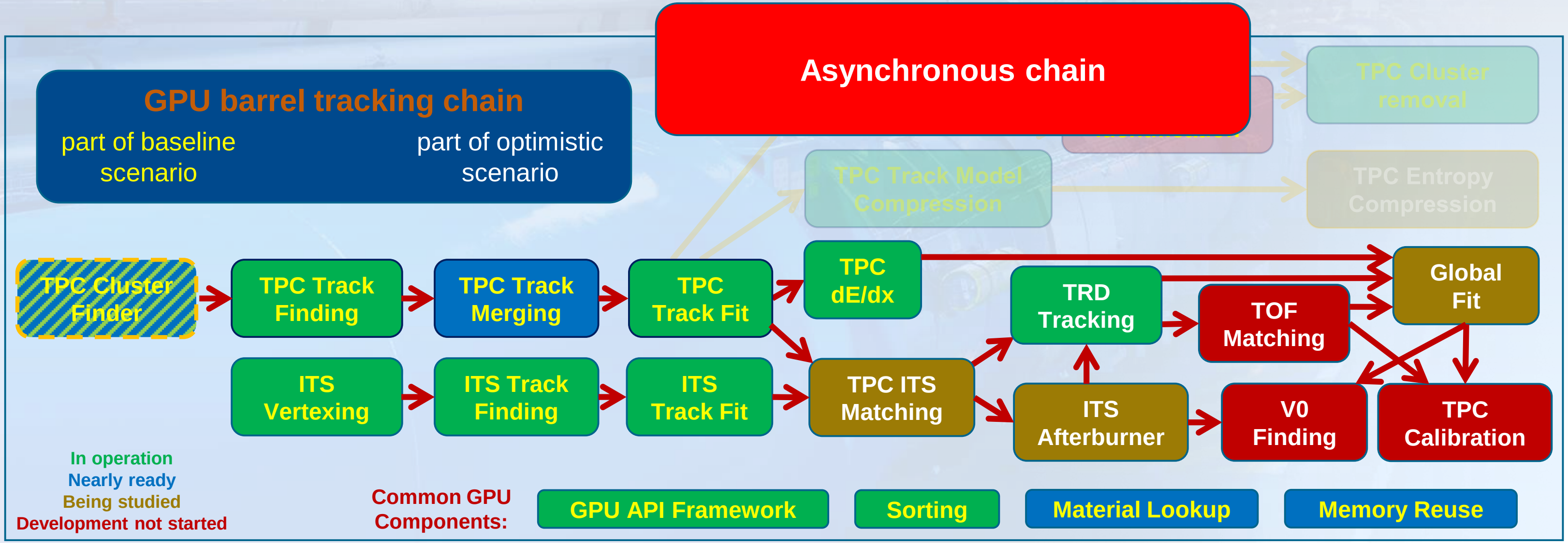
TRD tracking / TPC calibration: see poster of Ole Schmidt
Space point calibration of the ALICE TPC with track residuals



Reconstruction steps on GPU (Barrel Tracking)

- **Status of reconstruction steps on GPU:**
  - Different reconstruction steps enabled in synchronous and asynchronous reconstruction.

TRD tracking / TPC calibration: see poster of Ole Schmidt
Space point calibration of the ALICE TPC with track residuals



Reconstruction steps on GPU (Barrel Tracking)

- **Status of reconstruction steps on GPU:**

Strategy:

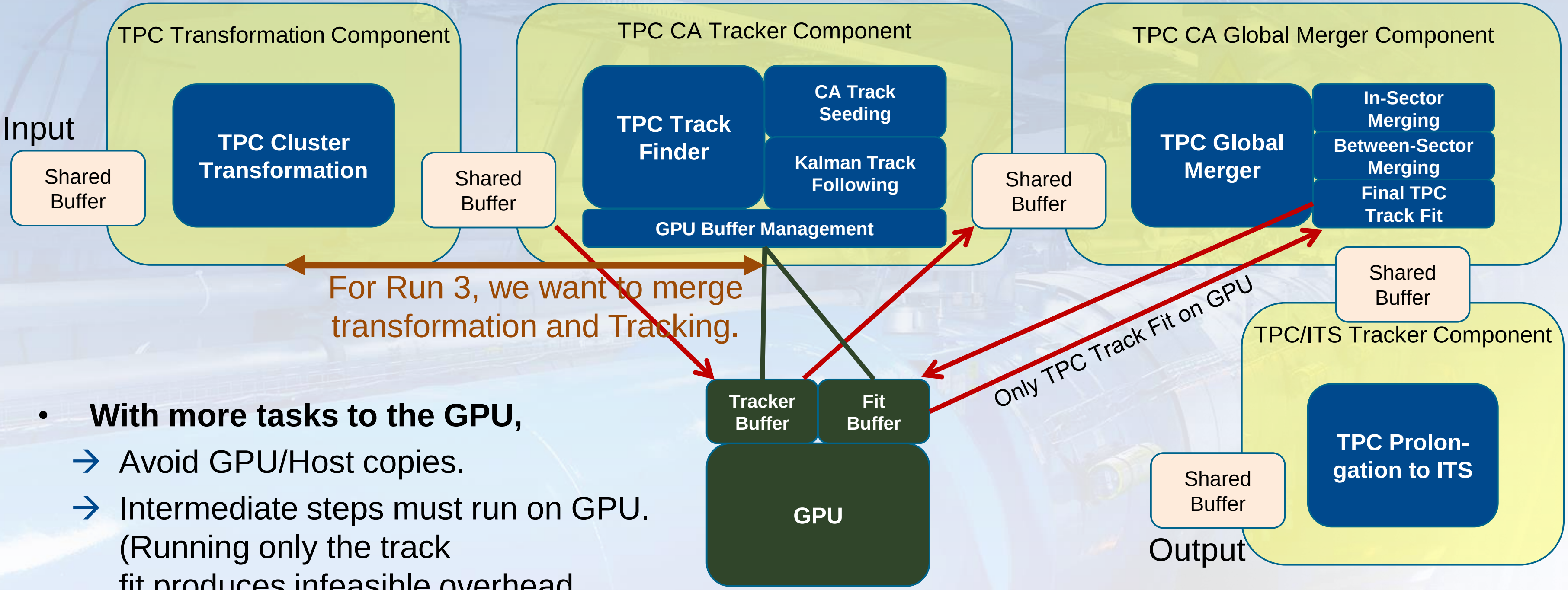
- Start with standalone TPC and ITS tracking.
  - Standalone ITS tracking needed since TPC tracks lack absolute time.
  - ITS tracking uses vertexer as first step.
  - TPC tracking has no vertex constraint, starts with segment tracking in individual TPC sectors, than merges the segments and refits.
- ITS and TPC tracks are matched, fixing the time for the TPC.
- The afterburner propagates unmatched TPC tracks into the ITS and tries to find matching hits of short tracks not found in ITS standalone tracking.
- Tracks are extrapolated outwards into the TRD, once the time is fixed.
  - TRD standalone tracking and matching (like for ITS) is less efficient due to many fake TRD tracklets.
- Optionally, after TRD tracks can be extrapolated to TOF.
- Global refit uses the information from all detectors.
- V0 finding
- In the synchronous phase, the TPC compression chain starts after the TPC standalone tracking in parallel:
  - Clusters not used in physics are removed, depending on the strategy (see later) this might require extra steps for identification and rejection of very low p_T clusters below 10 MeV/c.
  - Track model (and other steps) reduce the entropy for the final entropy encoding.
  - Final entropy encoding using ANS. Not clear yet whether this will run on GPU efficiently. Alternatively, transport entropy-reduced clusters to host and run entropy encoder there.

TRD tracking / TPC calibration: see poster of Ole Schmidt
Space point calibration of the ALICE TPC with track residuals

Development not started

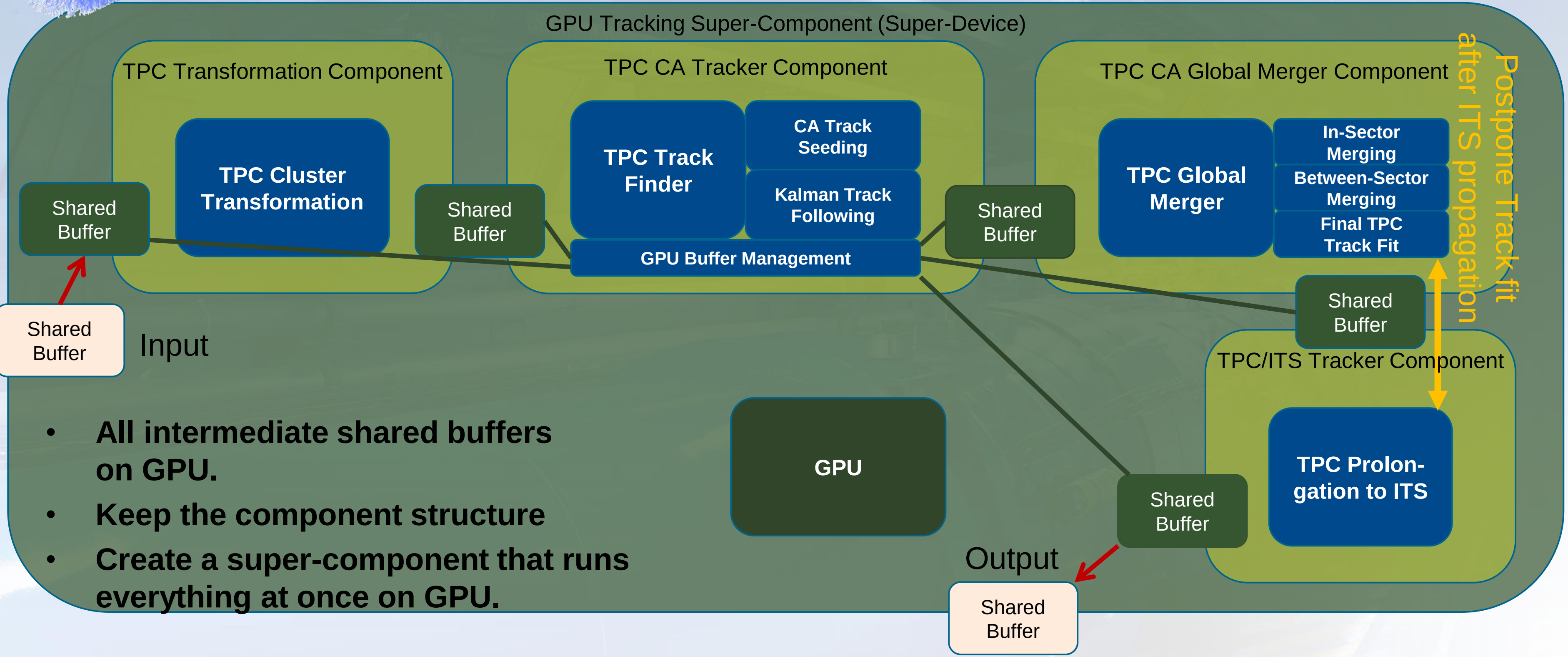
Components: GPU API Framework | Sorting | Material Lookup | Memory Reuse

Approach if Run 2 HLT TPC / ITS Tracking Components



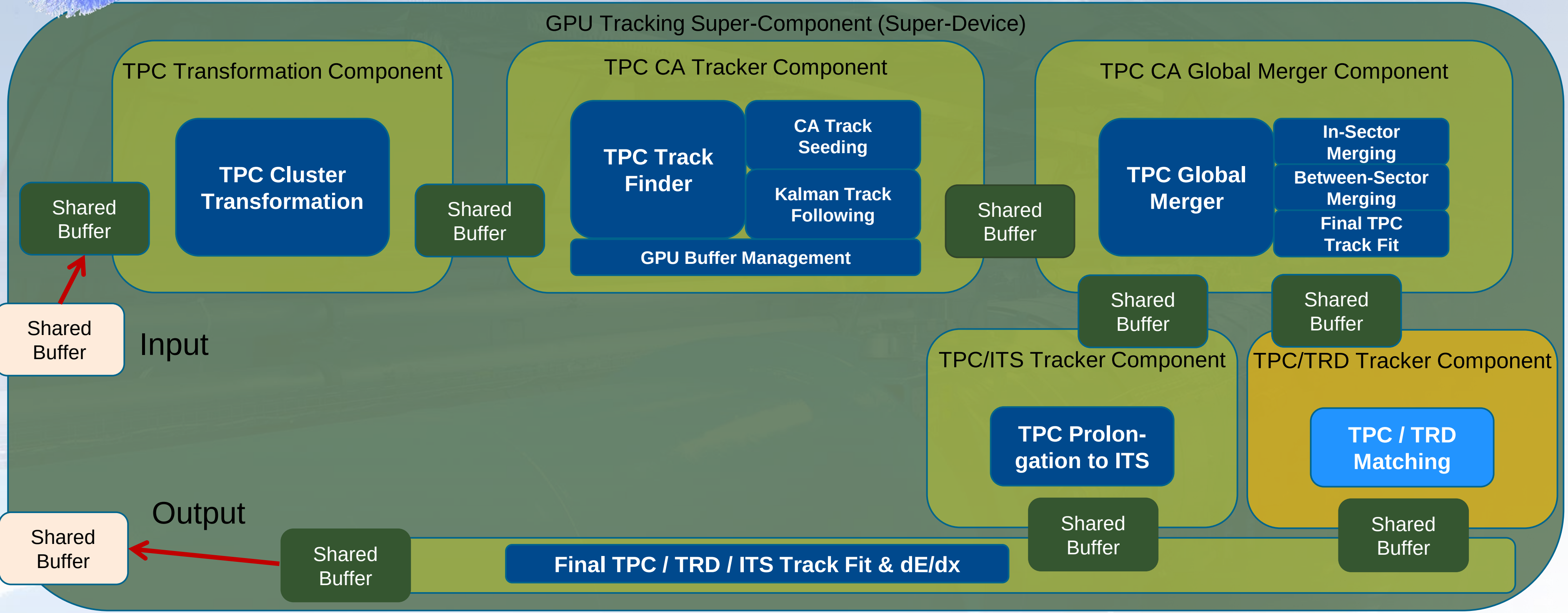
- **With more tasks to the GPU,**
  - → Avoid GPU/Host copies.
  - → Intermediate steps must run on GPU. (Running only the track fit produces infeasible overhead.

Approach for Run 3



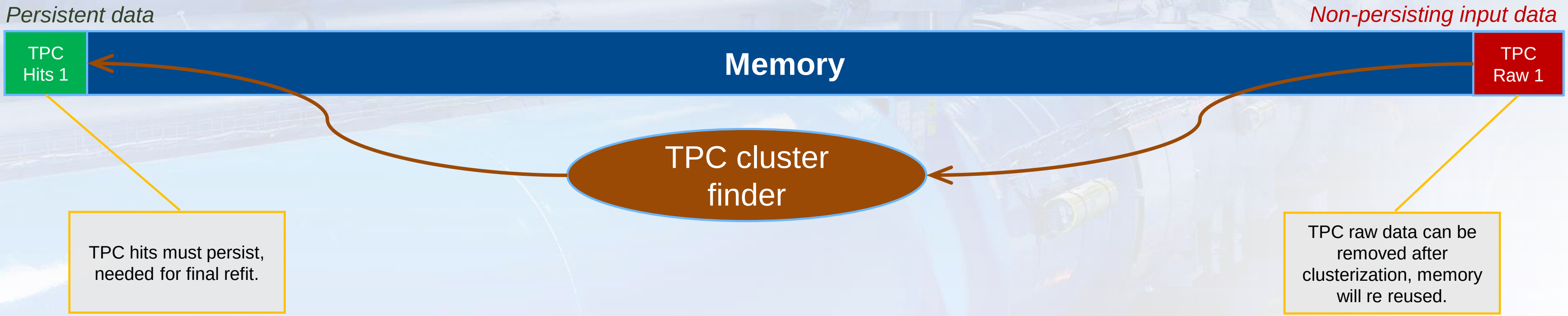
- **All intermediate shared buffers on GPU.**
- **Keep the component structure**
- **Create a super-component that runs everything at once on GPU.**

Approach for Run 3



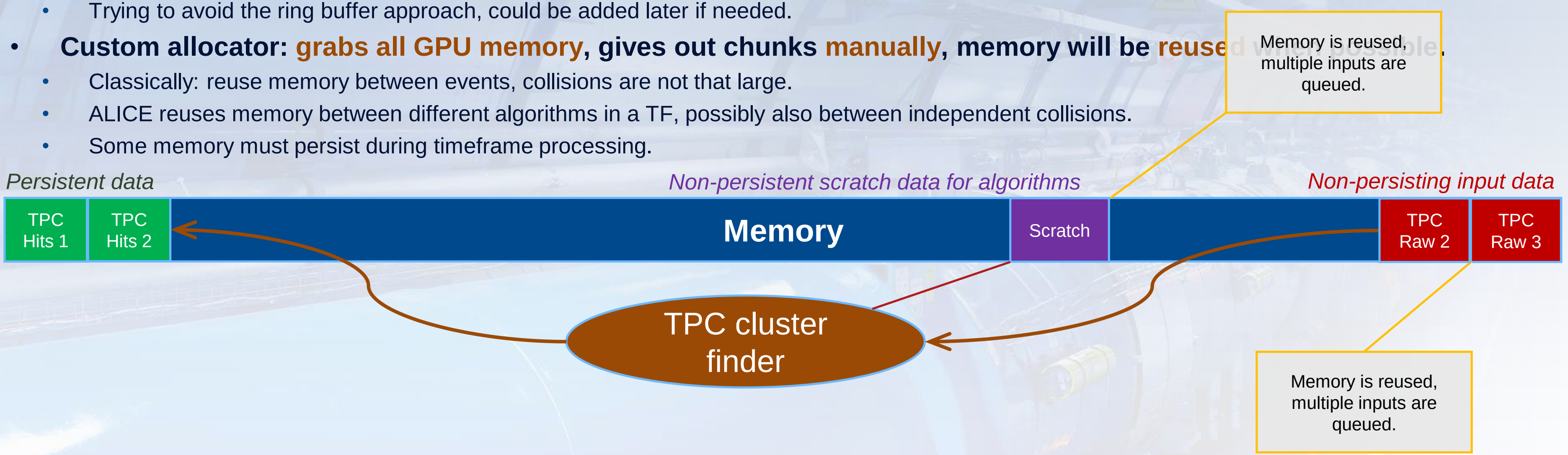
Memory requirements

- **ALICE reconstructs timeframes (TF) independently (~10 – ~20 ms; 128 – 256 orbits; ~500 – ~1000 collisions).**
  - One TPC drift time of data not reconstructible at TF border (~ 90 us) → < 1 % of statistics lost (< 0.5 % for 20 ms).
  - Timeframe should fit in GPU memory. If not, could use kind of ring buffer, or reduce TF length to 128 orbits.
  - Trying to avoid the ring buffer approach, could be added later if needed.
- **Custom allocator: grabs all GPU memory, gives out chunks manually, memory will be reused when possible.**
  - Classically: reuse memory between events, collisions are not that large.
  - ALICE reuses memory between different algorithms in a TF, possibly also between independent collisions.
  - Some memory must persist during timeframe processing.



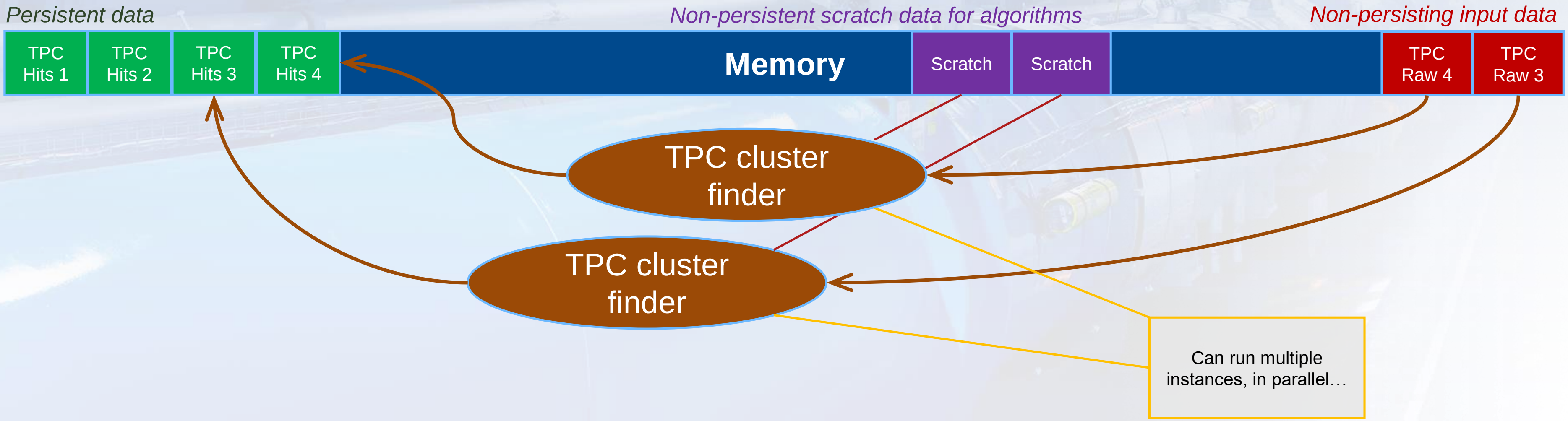
Memory requirements

- **ALICE reconstructs timeframes (TF) independently (~10 – ~20 ms; 128 – 256 orbits; ~500 – ~1000 collisions).**
  - One TPC drift time of data not reconstructible at TF border (~ 90 us) → < 1 % of statistics lost (< 0.5 % for 20 ms).
  - Timeframe should fit in GPU memory. If not, could use kind of ring buffer, or reduce TF length to 128 orbits.
  - Trying to avoid the ring buffer approach, could be added later if needed.
- **Custom allocator: grabs all GPU memory, gives out chunks manually, memory will be reused when possible.**
  - Classically: reuse memory between events, collisions are not that large.
  - ALICE reuses memory between different algorithms in a TF, possibly also between independent collisions.
  - Some memory must persist during timeframe processing.



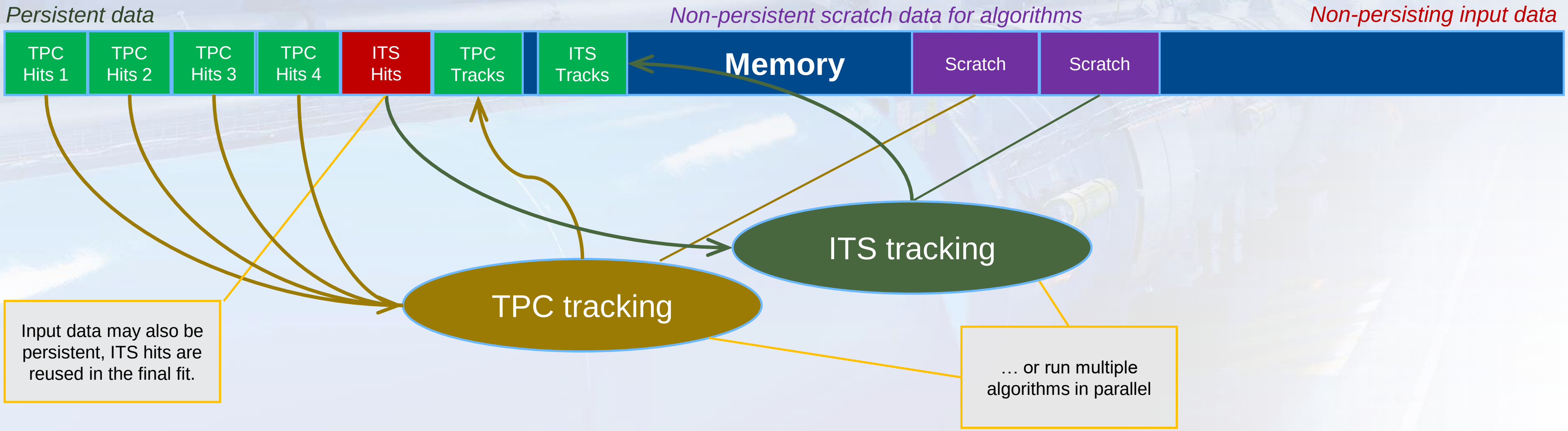
Memory requirements

- **ALICE reconstructs timeframes (TF) independently (~10 – ~20 ms; 128 – 256 orbits; ~500 – ~1000 collisions).**
  - One TPC drift time of data not reconstructible at TF border (~ 90 us) → < 1 % of statistics lost (< 0.5 % for 20 ms).
  - Timeframe should fit in GPU memory. If not, could use kind of ring buffer, or reduce TF length to 128 orbits.
  - Trying to avoid the ring buffer approach, could be added later if needed.
- **Custom allocator: grabs all GPU memory, gives out chunks manually, memory will be reused when possible.**
  - Classically: reuse memory between events, collisions are not that large.
  - ALICE reuses memory between different algorithms in a TF, possibly also between independent collisions.
  - Some memory must persist during timeframe processing.



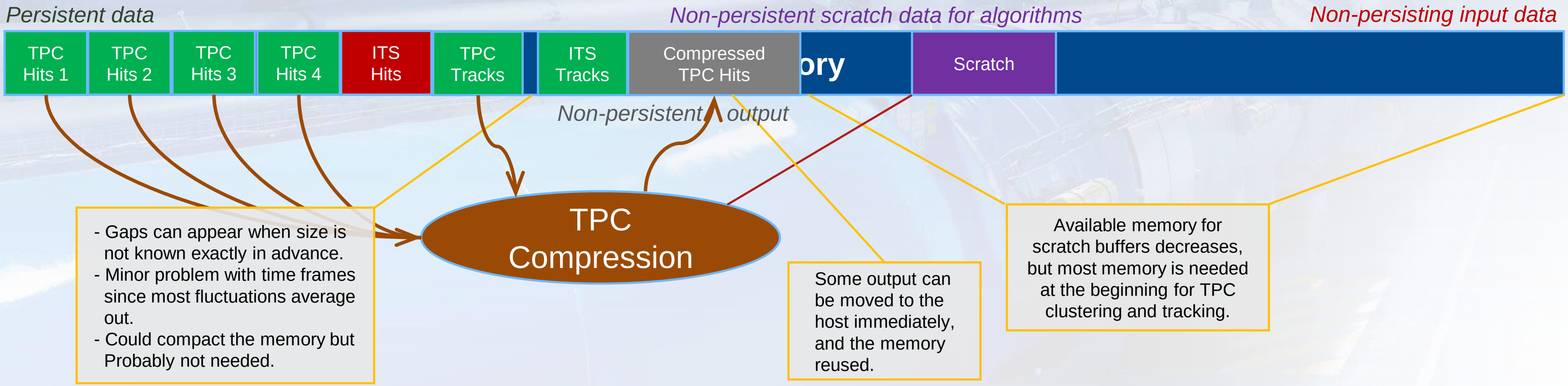
Memory requirements

- **ALICE reconstructs timeframes (TF) independently (~10 – ~20 ms; 128 – 256 orbits; ~500 – ~1000 collisions).**
  - One TPC drift time of data not reconstructible at TF border (~ 90 us) → < 1 % of statistics lost (< 0.5 % for 20 ms).
  - Timeframe should fit in GPU memory. If not, could use kind of ring buffer, or reduce TF length to 128 orbits.
  - Trying to avoid the ring buffer approach, could be added later if needed.
- **Custom allocator: grabs all GPU memory, gives out chunks manually, memory will be reused when possible.**
  - Classically: reuse memory between events, collisions are not that large.
  - ALICE reuses memory between different algorithms in a TF, possibly also between independent collisions.
  - Some memory must persist during timeframe processing.



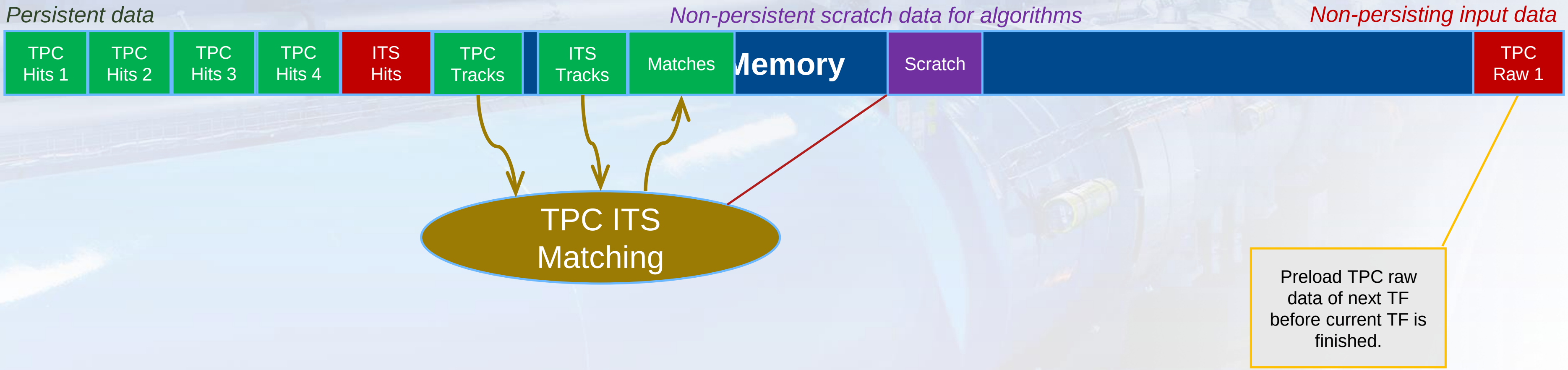
Memory requirements

- **ALICE reconstructs timeframes (TF) independently (~10 – ~20 ms; 128 – 256 orbits; ~500 – ~1000 collisions).**
  - One TPC drift time of data not reconstructible at TF border (~ 90 us) → < 1 % of statistics lost (< 0.5 % for 20 ms).
  - Timeframe should fit in GPU memory. If not, could use kind of ring buffer, or reduce TF length to 128 orbits.
  - Trying to avoid the ring buffer approach, could be added later if needed.
- **Custom allocator: grabs all GPU memory, gives out chunks manually, memory will be reused when possible.**
  - Classically: reuse memory between events, collisions are not that large.
  - ALICE reuses memory between different algorithms in a TF, possibly also between independent collisions.
  - Some memory must persist during timeframe processing.



Memory requirements

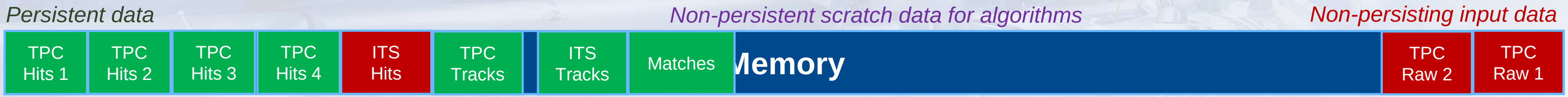
- **ALICE reconstructs timeframes (TF) independently (~10 – ~20 ms; 128 – 256 orbits; ~500 – ~1000 collisions).**
  - One TPC drift time of data not reconstructible at TF border (~ 90 us) → < 1 % of statistics lost (< 0.5 % for 20 ms).
  - Timeframe should fit in GPU memory. If not, could use kind of ring buffer, or reduce TF length to 128 orbits.
  - Trying to avoid the ring buffer approach, could be added later if needed.
- **Custom allocator: grabs all GPU memory, gives out chunks manually, memory will be reused when possible.**
  - Classically: reuse memory between events, collisions are not that large.
  - ALICE reuses memory between different algorithms in a TF, possibly also between independent collisions.
  - Some memory must persist during timeframe processing.



Memory requirements

Work in Progress

- **ALICE reconstructs timeframes (TF) independently (~10 – ~20 ms; 128 – 256 orbits; ~500 – ~1000 collisions).**
  - One TPC drift time of data not reconstructible at TF border (~ 90 us) → < 1 % of statistics lost (< 0.5 % for 20 ms).
  - Timeframe should fit in GPU memory. If not, could use kind of ring buffer, or reduce TF length to 128 orbits.
  - Trying to avoid the ring buffer approach, could be added later if needed.
- **Custom allocator: grabs all GPU memory, gives out chunks manually, memory will be reused when possible.**
  - Classically: reuse memory between events, collisions are not that large.
  - ALICE reuses memory between different algorithms in a TF, possibly also between independent collisions.
  - Some memory must persist during timeframe processing.



- **Estimated maximum memory needed during important for 10 ms TF (*2 for 20 ms):**
  - TPC Cluster finder: ~ 3 GB (+ input / scratch data, which is pipelined)
  - TPC Transformation: 12.1 GB
  - TPC Sector tracker: ~ 14.6 GB (including persistent memory from previous steps)
  - TPC Merger / track fit: 14.1 GB
  - TPC Compression: 12.9 GB
    - Later steps do not scale their scratch memory with TPC input → less memory intensive.

→ **16 GB GPU will suffice for 10 ms TF (unclear for 12 GB after optimizations).**

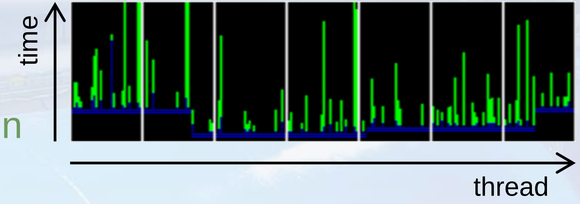
- **8 GB** insufficient for **10 ms TF**, **20 ms TF** needs **32 GB**, alternatively **ring buffer**.

Performance Tuning

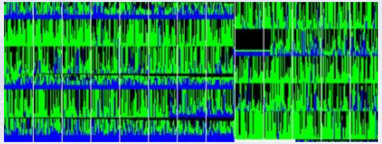
- **GPUs of different vendor's / generation's might favor different tuning.**
  - Many algorithms have tunable parameters (for processing speed).
  - We implemented most features such, that they can be switched off.
    - Worst case, at compile time via preprocessor definition.
- **One example: Distribution of tracks among GPU threads during track following:**
  - Illustration of active GPU threads over time (time on y-axis).

- Black : Idle
- Blue : Track Fit
- Green : Track Extrapolation

No dynymic rescheduling



With dynamic rescheduling

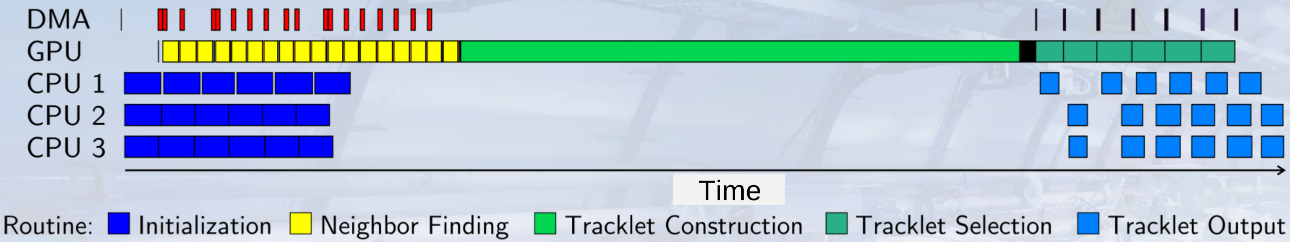


- Number of average idle threads reduced by factor ~3, but large overhead for rescheduling.
- Yields ~50% speedup on some GPUs, but becomes even slower on others.

- **For new GPUs:**
  - Run a benchmark with a parameter range scan to find best settings.
  - After 3 iterations (GPU generations), we got good results out of the box.

Performance Tuning

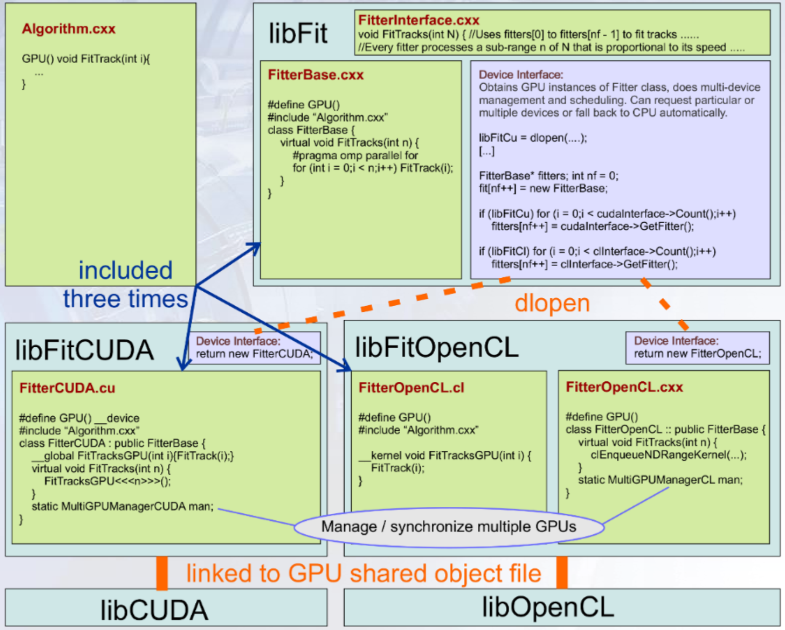
- **Handling of asynchronous computation / data transfers**
  - **1st iteration (Run 1 HLT)**: Split event in chunks, to pipeline CPU processing, GPU processing, and PCIe transfer.



  - **2nd iteration (Run 2 HLT)**: Processing of two events in parallel on the GPU concurrently.
    - ~20% faster than first version – GPUs have become wider and this exploits the parallelism better.
    - Not possible during Run 1 due to GPU limitations at that time.
    - We still kept the pipeline-scheme within each event, to maximize performance.
  - **3rd iteration (Run 3)**: Go back to the old scheme from Run 1 – with time frames instead of events.
    - Time frames are large → avoid keeping multiple in memory.
    - Enough parallelism inside one time frame.

Compatibility with several GPU frameworks

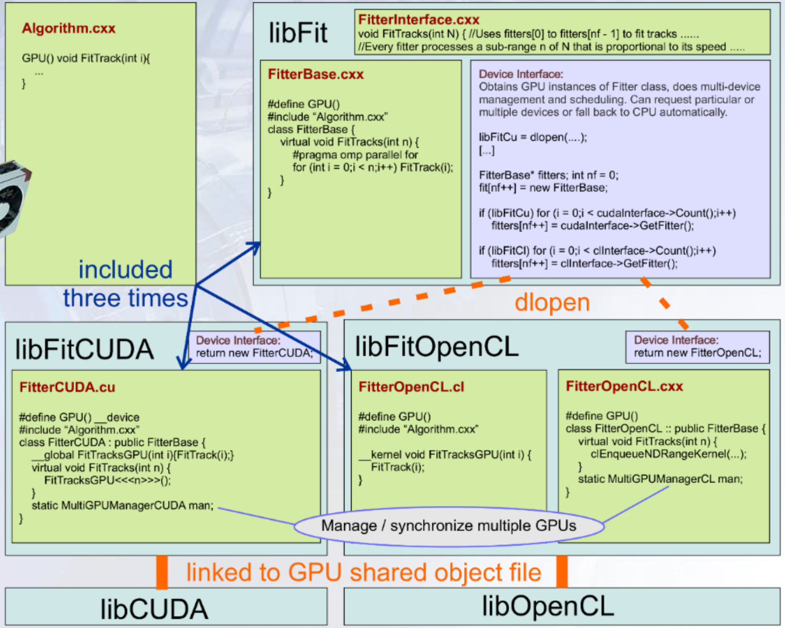
- **Generic common C++ Code compatible to CUDA, OpenCL, HIP, and CPU (with pure C++, OpenMP, or OpenCL).**
  - OpenCL needs clang compiler (ARM or AMD ROCm) or AMD extensions (TPC track finding only on Run 2 GPUs and CPU for testing).
  - Certain worthwhile algorithms have a vectorized code branch for CPU using the Vc library.
  - All GPU code swapped out in dedicated libraries, same software binaries run on GPU-enabled and CPU servers.



Compatibility with several GPU frameworks

- **Generic common C++ Code compatible to CUDA, OpenCL, HIP, and CPU (with pure C++, OpenMP, or OpenCL).**
  - OpenCL needs clang compiler (ARM or AMD ROCm) or AMD extensions (TPC track finding only on Run 2 GPUs and CPU for testing).
  - Certain worthwhile algorithms have a vectorized code branch for CPU using the Vc library.
  - All GPU code swapped out in dedicated libraries, same software binaries run on GPU-enabled and CPU servers.

- **Screening different platforms for best price / performance.**
  (including some non-competitive platforms for cross-checks and validation.)
  - **CPUs** (**AMD Zen**, **Intel Skylake**)
    C++ backend with **OpenMP**, AMD **OCL**
  - **AMD GPUs**
    (**S9000** with **OpenCL 1.2**, **MI50** / **Radeon 7** / **Navi** with **HIP** / **OCL 2.x**)
  - **NVIDIA GPUs**
    (**RTX 2080** / **RTX 2080 Ti** / **Tesla T4**
    with **CUDA**)
  - **ARM Mali GPU** with OCL 2.x
    (Tested on dev-board with Mali G52)



- **Optimize TCO (faster GPUs → less latency → smaller buffers).**

EXPERIENCE OF THE LHCB EXPERIMENT