The LHCb Heterogeneous Framework in Run 3 and Its Evolution Towards Run 4

Arthur Hennequin
arthur.hennequin@cern.ch





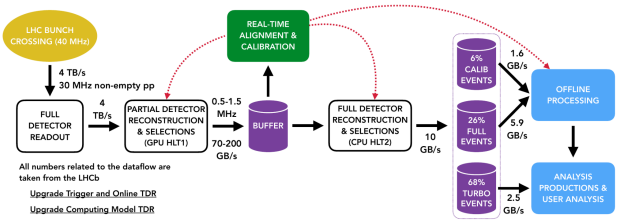
Outline

1. Run 3: the heterogeneous trigger
2. Lessons from two frameworks
3. A unified framework

What this talk covers

- How the Run 3 trigger works, and why it uses two frameworks.
- What operating it taught us about GPU/CPU heterogeneity.
- The scheduler that now unifies the two, and how data flow through it.
- Where we stand on the way to Run 4.

The Run 3 trigger: software at 30 MHz



- **No hardware trigger**: every bunch crossing (30 MHz) is read out.
- The event builder assembles complete events; the HLT farm receives ∼4 TB/s.
- **Two software stages**: HLT1 selects in real time, HLT2 reconstructs with offline quality.

A first for a collider experiment

The trigger is no longer a hardware decision — it is a **computing problem**, at the full bunch-crossing rate.

Two stages, two execution models



HLT1 — Allen (GPU)

- Real-time reconstruction: VELO, tracking, primary vertices, muon and ECAL.
- ∼500 RTX A5000 GPUs deliver the full 30 MHz.
- Output reduced to ∼100 GB/s for HLT2.

HLT2 — Gaudi (CPU)

- Full offline-quality reconstruction: tracking, RICH, calorimeters, muon, PID.
- Inclusive and exclusive selection lines.
- Output written to offline storage.

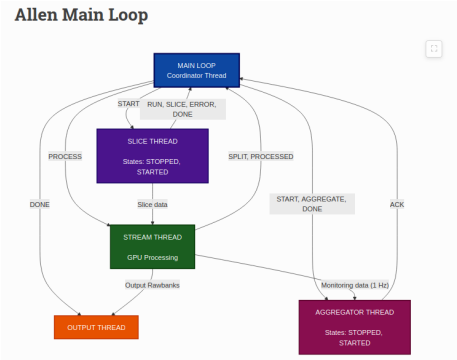
HLT1 is throughput- and efficiency-driven; HLT2 is offline quality.

HLT1: Allen — GPU-first by design

- The host only steers: raw data go straight to the GPU, decisions come back.
- Each CUDA stream processes a **slice of ∼1000 events**; the sequence is static and sorted at configuration time.
- Transient data stay on the device; memory is pre-allocated per stream.
- Backend-agnostic C++: the same source runs on GPU and CPU.

The key point

GPU acceleration came from designing the **whole framework** around the GPU.



Allen's multi-event execution model.

HLT2: Gaudi — physics quality first

- **Event-by-event** execution: algorithms read and write the transient event store (TES).
- The same reconstruction code as offline (Rec): tracking, RICH, calorimeters, muon, particle identification.
- The scheduler evaluates $\mathcal{O}(4000)$ physics lines over sequences of $\mathcal{O}(20000)$ algorithms; HLT1 has $\mathcal{O}(100)$ lines and $\mathcal{O}(300)$ algorithms.
- That combinatorial complexity — and the mature ecosystem around it — is why HLT2 is hard to replace.

A different world

The execution model is fundamentally **single-event** and **CPU-centric**. Running it faster means running it *differently*.

Why we cannot simply replace it

- Physics validation is defined against this code.
- Offline and HLT2 share the same algorithms.

What Run 3 proved — and where it hurts

Successes

- A fully software 30 MHz trigger is possible.
- GPU real-time reconstruction delivers the physics LHCb needs.
- HLT1 has been stable in production with ∼500 GPUs, fully integrated in the online system.
- The collaboration now has real GPU expertise and tooling.

Limits

- **Two frameworks**, two execution models, two sets of infrastructure.
- GPU acceleration **stops at the HLT1/HLT2 boundary**.
- HLT2 is CPU-bound, and its cost grows with luminosity.
- Testing and production use **different code paths**.

Heterogeneity was the right answer for Run 3 — but it was implemented as **two separate systems**.

Run 4 and Run 5: the scenarios are still open

- Higher luminosity and pile-up: **larger and more complex events**, and more reconstruction per event.
- **Flat computing budget**: CPUs alone will not be enough.
- The LHCb physics programme benefits from more reconstruction in real time (Turbo, real-time analysis).
- HLT1 has already shown that GPU acceleration works at 30 MHz.

Several scenarios are under study

Different trigger splits, different output rates, different mixes of CPU and GPU. No single option has been chosen.

The common requirement

A framework in which GPU and CPU algorithms can be composed and tried, so the collaboration can evaluate the options.

The framework must give us the flexibility to try the scenarios — not commit to one today.

Requirements for a unified framework

- **GPU and CPU algorithms in one sequence**.
- **One data flow** between GPU buffers and per-event TES containers.
- **Test what you run**: same code path everywhere.
- **Static scheduling**: no runtime reordering overhead.
- Both **multi-event** (GPU) and **single-event** (CPU) semantics.
- **Any input, any output, any sequence**.

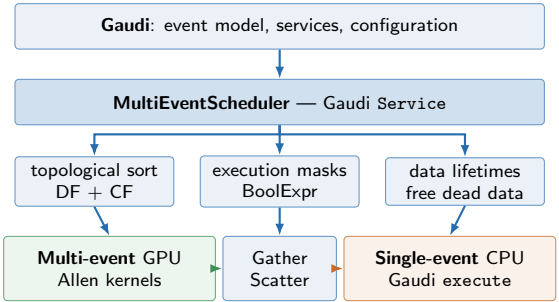
Not a rewrite

Build on **Gaudi** (event model, services, algorithms) and on **Allen** (GPU execution, multi-event scheduling). Converge, do not fork.

Key requirement

We aim to preserve HLT2 physics performance and offline-quality reconstruction.

The MultiEventScheduler: one scheduler for both worlds



- A Gaudi service that **replaces the default event loop**: the algorithms themselves are unchanged.
- Exposes the scheduler state to the algorithms that need it.
- Written in C++; the **whole schedule is computed once** at initialisation.
- The same scheduler runs HLT1 and HLT2 sequences.

Initialisation: the schedule is computed once

Initialisation (once)

1. Instantiate all algorithms.
2. Resolve **data-flow** dependencies from a producers map.
3. Add **control-flow** dependencies (ordered CF nodes).
4. **Topological sort** (Kahn, two ready-sets).
5. Build **execution masks**.
6. Compute **data lifetimes**.

No runtime reordering: the schedule is fixed before the first event.

Data-flow + control-flow

- Inputs determine the producer.
- Ordered CF nodes sequence their children.
- `both_ready` first, `df_only_ready` fallback.
- Cycles (DF–CF, CF–CF) are detected.

Execution masks

One `BoolExpr` tree per algorithm (AND, OR, NOT of other algorithms' masks), evaluated as **bitmask operations**: no TES round-trip.

Two kinds of algorithms in one sequence

Multi-event (`IsMultiEvent = true`)

- **One shared data store** per slice.
- Processes all events in the batch at once.
- Allen GPU algorithms, wrapped via `GaudiAllenAlgorithmWrapper`.

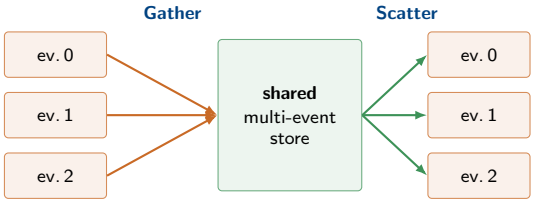
Single-event (default)

- **Per-event stores**, one per event in the slice.
- Traditional Gaudi `execute(ctx)`.
- Called once per *valid* event, following the mask.

The hard part

Different data layouts: **structure-of-arrays device buffers spanning many events** vs. **per-event TES containers**.
Mixed sequences need an explicit bridge.

Gather and scatter: bridging the two worlds



The two bridges that enable mixed HLT1 + HLT2 sequences.

`GatherEventMultiTransformer`

- Collects per-event inputs into `span<const T*>`; output goes to the shared store.
- HLT2 per-event data → GPU algorithm.

`ScatterEventMultiTransformer`

- Returns one vector per output; the framework dispatches each element to its event store.
- GPU output → HLT2 per-event algorithms.

One I/O layer, one entry point

Input: `IInputProviderSvc`

- `MDFProvider` — MDF files
- `MEPProvider` — MEP buffers (online)
- ROOT provider — event branches

Output: `IOutputWriter`

- `FileOutputWriter` — ROOT or MDF
- `ZMQOutputWriter` — network
- MBM output — MooreOnline

Entry point: `lbexec`

- One way to run any sequence
- Same path in test and production
- No separate legacy workflows

Any input $\times$ any output $\times$ any sequence.

Status: migration done, restructuring ahead

Done

- Migration to the MultiEventScheduler complete.
- The old standalone executable is retired: the scheduler is the only path.
- HLT1 and HLT2 can run in one sequence.

Next: internal restructuring

- Separate LHCb-specific code from the framework-independent part.
- Move the framework-independent part to its own project, **AllenCore**.
- AllenCore is based on Gaudi and adds the multi-event and GPU layer.

The physics code stays in LHCb; the framework becomes reusable beyond LHCb.

From consolidation to Run 4 preparation

Framework

- Consolidation is mostly functional.
- Teams already use it to prepare Run 4.

Physics

- New algorithms are being ported from CPU to GPU.
- The target is HLT2 physics performance.

Reproducibility

- Campaign to reach bit-identical CPU/GPU results.
- Differences mostly from FMA contraction and math function implementations.
- Run 3 Allen was already very close, but not bit-reproducible.

Run 4 scenarios are still being defined: the framework is the tool to evaluate them.

Lessons learned

Technical

- Framework design determines what algorithms can do.
- Porting algorithms without unifying data flow does not scale.
- Static scheduling and event slices make GPUs efficient.
- Keep a CPU fallback for simulation and validation.

Organisational

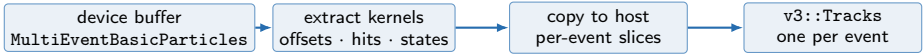
- Test what you run: one code path.
- Consolidate early — technical debt compounds.
- GPU work needs expertise and profiling.
- Physics validation comes first.

Summary

1. **Run 3** delivered the first fully software 30 MHz trigger, with HLT1 accelerated on ∼500 GPUs.
2. **Heterogeneity works**: the physics performance is there, and so is the operational experience.
3. But **two frameworks are the bottleneck**: duplicated infrastructure, test ≠ production, GPU acceleration stopping at HLT1.
4. The **MultiEventScheduler** unifies them: one static scheduler, mixed GPU/CPU sequences, gather/scatter bridges, any input and output.
5. The **MultiEventScheduler is portable HEP-wide**: not LHCb-specific, and AllenCore targets other experiments.
6. **Run 4**: the scenarios are open; the framework must let us try them.

Questions?

Backup: GPU tracks become Gaudi tracks



`ConvertAllenLongToV3Tracks` — a concrete scatter transformer:

- Input: a device-side view of *all* particles in the slice.
- CUDA kernels extract track offsets, LHCbIDs, hit counts and Kalman states.
- Everything is copied to the host once.
- Per event, a `v3::Tracks` container is built with VELO, UT, FT and muon hits, Kalman states and track IDs.
- Output: `std::vector<OutTracks>`, scattered to the per-event stores.

Why this matters

After this step, GPU output is **ordinary Gaudi data**. HLT2 algorithms cannot tell where it came from, and no physics code has to change.

The inverse direction

`GatherEventMultiTransformer` does the opposite: HLT2 data are packed into a shared store and handed to a GPU algorithm.

Backup: runtime — slices, workers, memory

Main thread — `nextEvent()`

```
while (true) {
  auto [ok, eof, _, idx, n, odin]
      = inputProvider->get_slice(1000);

  if (n > 0)
    workers->submit({idx, 0, n});
  else
    break;
}
workers->wait_for_completion();
```

- Each worker thread has its own CUDA context and stream, host/device memory managers, masks and per-event store slots.
- Slices are distributed through a multi-producer/multi-consumer queue.
- On `MemoryException`, the slice is **halved** and both halves are resubmitted.
- Per algorithm: evaluate mask → execute (once or per event) → free data no longer needed.

Throughput comes from large slices; safety comes from graceful degradation.

Backup: the four Allen entry points

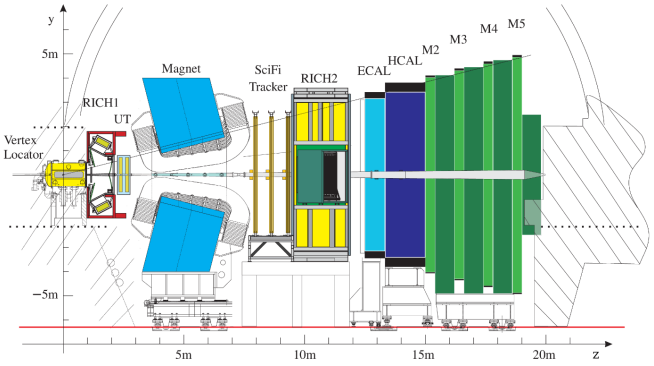
Path	What it was good for	What it could not do
Allen in Moore	CPU testing and simulation; could run with HLT2 algorithms and their checkers	No GPU
Allen next to Moore	A production-like environment for local testing	Not the real production environment
Allen next to MooreOnline	The production environment, on the GPU	Cannot mix with HLT2 algorithms; hard to run locally
Allen standalone	Fast local GPU development	No direct detector geometry (binary dumps); its own checkers; code duplication

The key point

Each path covered a different need, so all four were needed: none of them offered GPU, the production environment, HLT2 algorithms and easy local running at the same time.

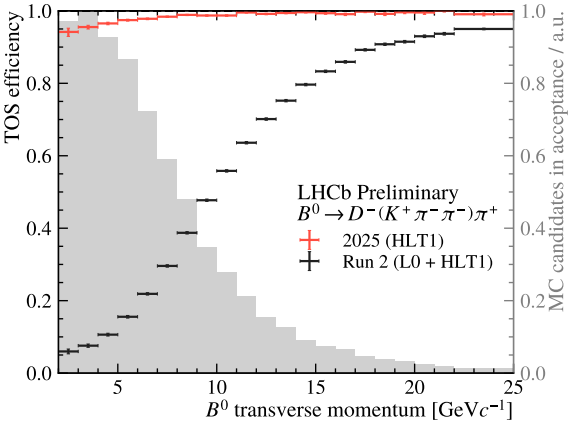
Today: Allen standalone is retired; the MultiEventScheduler is the only path.

Backup: the LHCb detector



- A **forward spectrometer** ($2 < \eta < 5$) for b- and c-hadron physics.
- **VELO**: vertices and impact parameter.
- **RICH1/2**: $\pi/K/p$ separation.
- **UT + FT + magnet**: track momentum.
- **ECAL/HCAL + muon**: photons, electrons, hadrons, muons.
- Run 3: 30 MHz readout, no hardware trigger.

Backup: physics benefit of real-time reconstruction



$B^0 \to D\pi$: 2025 data vs Run 2

Comparison of TOS efficiencies between 2025 data (red) and Run 2 (black). The 2025 efficiencies are of the combined OneTwoTrackMVA response; the Run 2 efficiencies are the product of the combined OneTwoTrackMVA efficiencies with the respective L0 efficiency. The right axis shows the MC candidates in acceptance (arbitrary units).