Performance Study of GPUDirect Storage for CMS Reconstruction Workflows

CERN openlab Summer Student Programme Lightning talks

AYAAN DEWANI

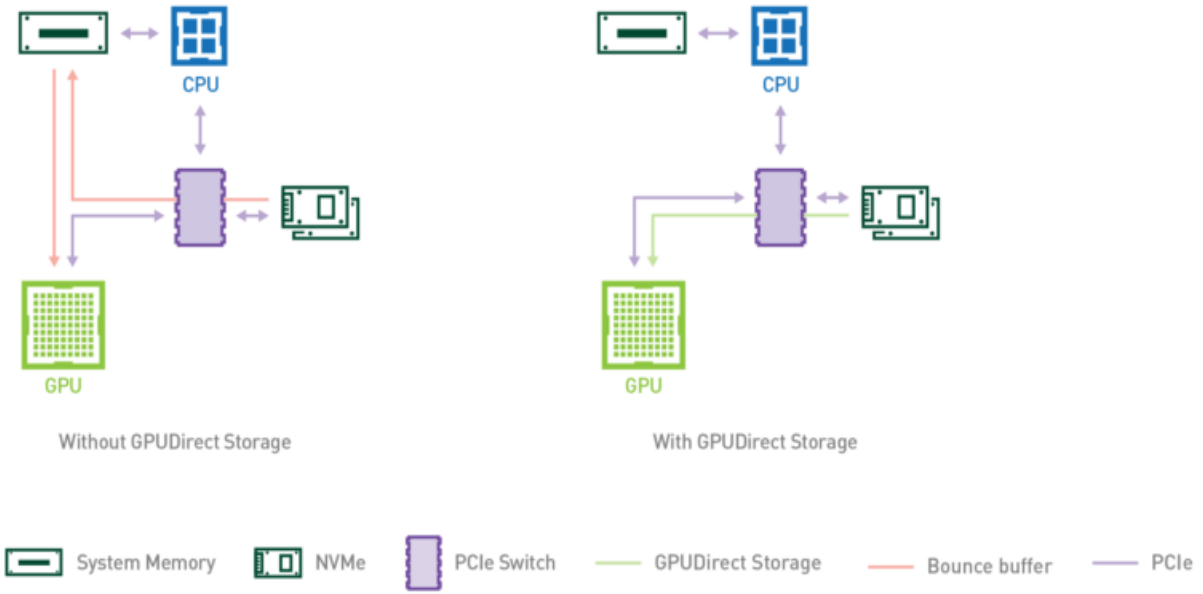
Supervisors: MARKUS HOLZER & FELICE PANTALEO

18/08/2026

Introduction to GPUDirect Storage (GDS)

What is GDS?

- Software library ecosystem by NVIDIA
- Enables Direct Path Between NVMe Storage and GPU Memory.
- Completely bypasses the host CPU and system memory, eliminating the "bounce buffer" I/O bottleneck.
- Utilized in modern deep learning and distributed machine learning training to prevent GPUs from starving while waiting for data.

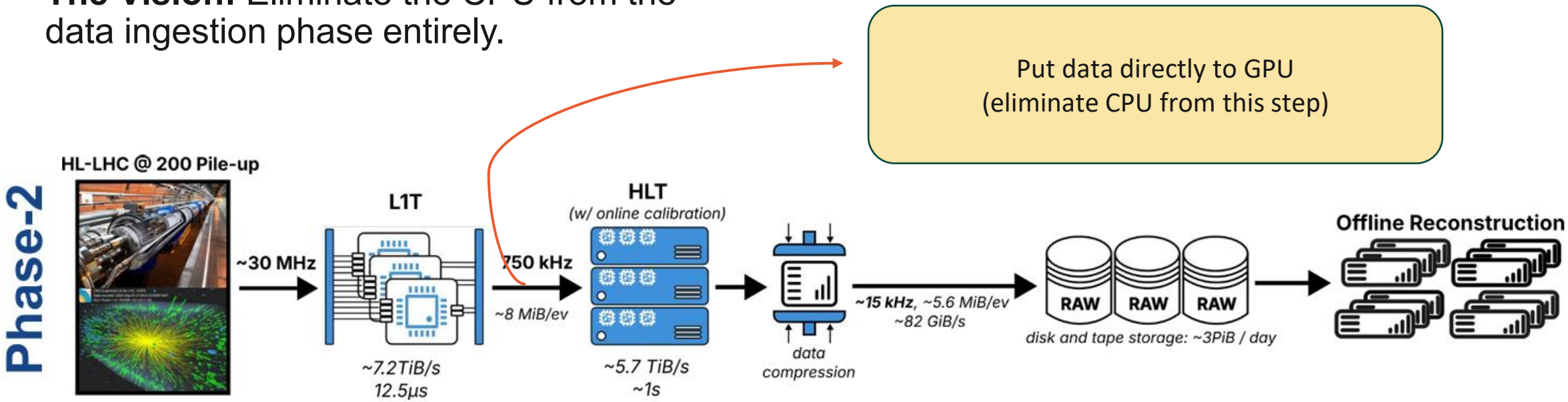


GDS in CMSSW

The Problem: High-Energy Physics (HEP) detectors generate massive binary payloads. Processing this data on GPUs currently requires bouncing the raw bytes through the CPU and system memory first, creating an I/O bottleneck.

The Vision: Eliminate the CPU from the data ingestion phase entirely.

The Goal: We want to see if we can integrate GDS into the existing CMSSW pipelines. Implement a "zero-copy" ingestion pipeline for the CMS Pixel Clusterizer that reads, parses, and unpacks raw detector data entirely on the GPU.



Benchmarking the Bottleneck (GDS vs. POSIX)

Before rewriting the CMSSW pipeline, we needed to prove GDS outperforms standard POSIX reads.

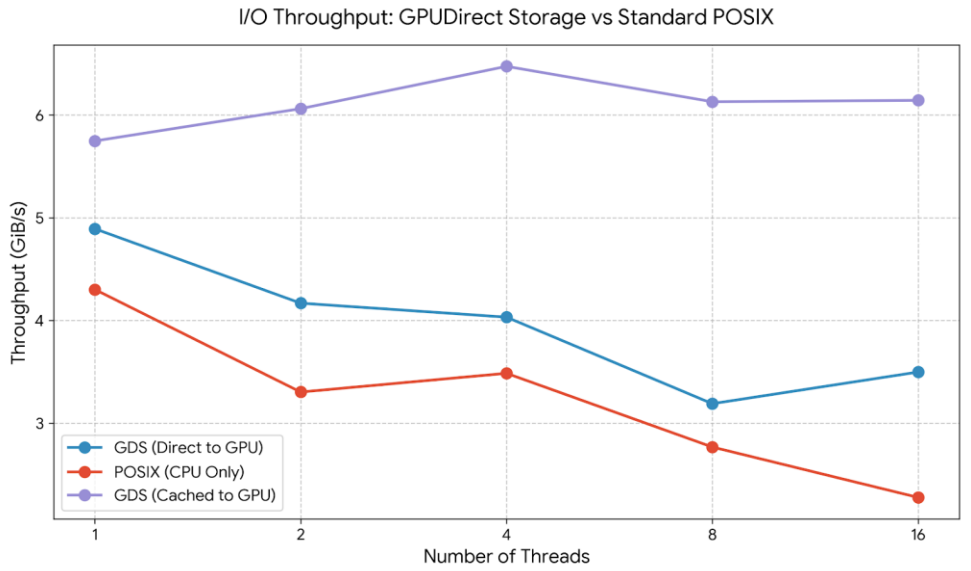
Benchmark Methodology: Compared standard fio against gdsio to simulate heavy I/O loads.

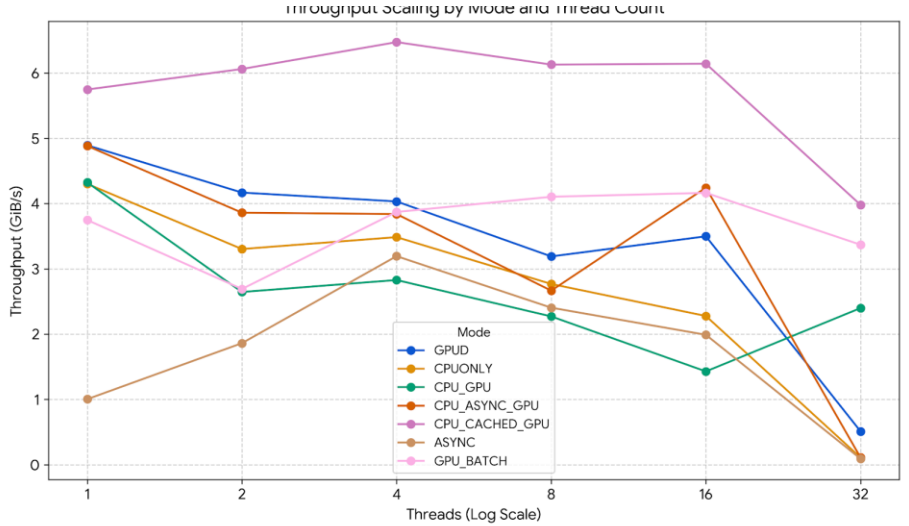
The Results:

Standard POSIX (fio): A single thread achieved ~3.29 GiB/s. It required spinning up 16 CPU threads just to reach ~4.70 GiB/s.

GDS (gdsio): A single thread completely bypassed the CPU and achieved ~4.89 GiB/s immediately, maxing out at over 6.4 GiB/s with caching.

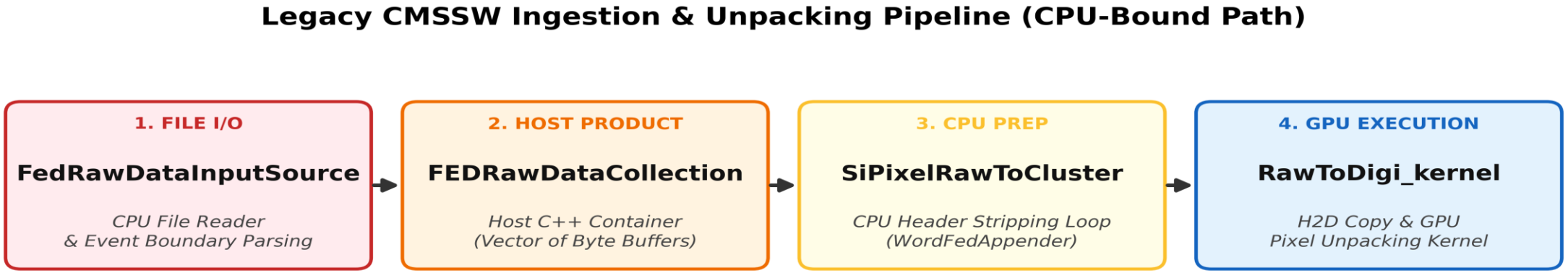
Takeaway: GDS is a promising technology for our purpose.





The Original Pipeline & FedRawDataInputSource

How does CMSSW currently deal with the raw data?



The CPU Source (`FedRawDataInputSource.cc`): Reads the file chunk by chunk, identifies event boundaries using the CPU, and allocates host memory.

The Product (`FEDRawDataCollection`): A host-side container. It owns vectors of bytes representing separated hardware fragments.

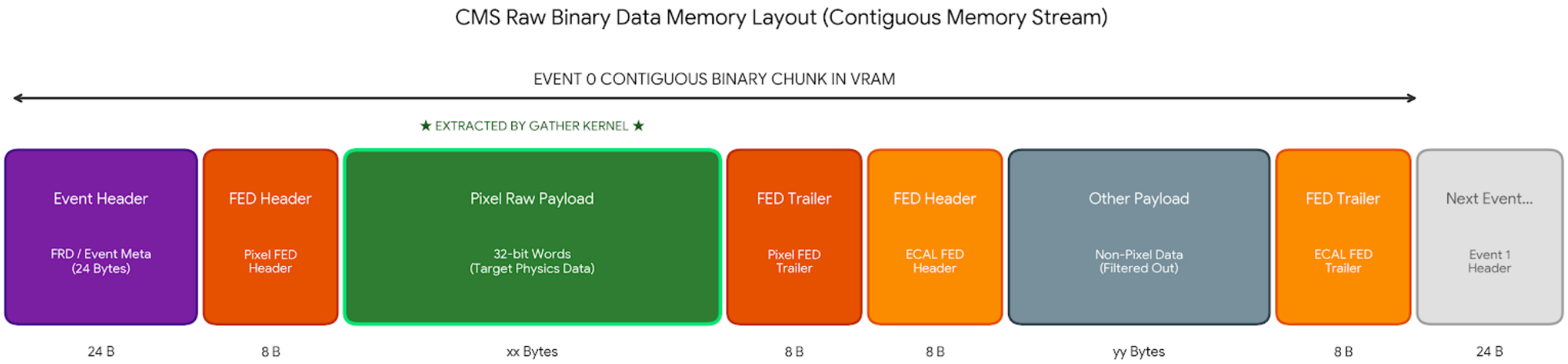
The Handoff: The downstream unpacker (`SiPixelRawToCluster.cc`) uses a CPU loop (`WordFedAppender`) to extract data from this collection, strip headers, and pack flat arrays.

The Issue: Only after all that CPU processing is finished does the framework issue the `cudaMemcpy` (Host-to-Device) to send the flat buffer across the PCIe bus to the GPU.

CMS Raw Data Format

To bypass the CPU, we had to reverse-engineer how the raw binary chunks are structured so the GPU could parse them natively.

The Contiguous Layout:

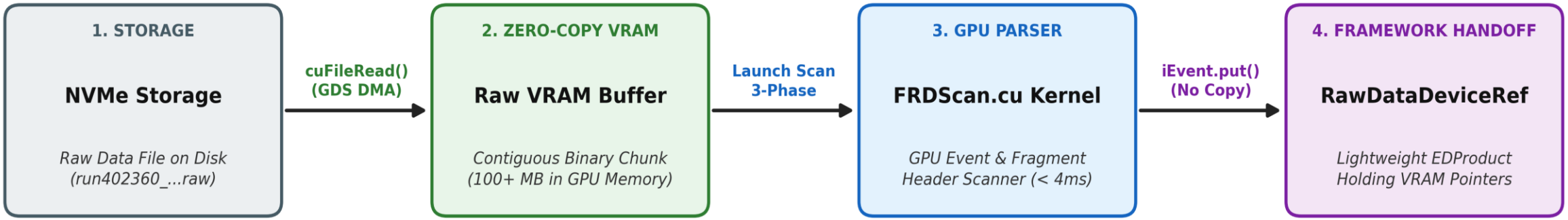


The Challenge: The size of the raw payload (xx or yy bytes) is highly dynamic based on collision occupancy.

Zero-Copy Ingestion (GDSRawFileReaderPure.cc)

- **Custom Source Module (`GDSRawFileReaderPure.cc`):**
- **Direct-to-VRAM Storage Transfer**
- **On-Device Event Indexing (`FRDScan.cu`: device-side metadata)**
- **Device-Native Handoff (`RawDataDeviceRef`)**

Zero-Copy Ingestion Architecture (GDSRawFileReaderPure.cc)



The Result: Downstream modules receive raw memory pointers and FED index offsets that are already resident in VRAM, enabling 100% host-bypass for subsequent unpacking.

The Gather Step

The downstream physics decoder doesn't want an index. It wants a flat, perfectly concatenated array of pure physics words.

The `GDSPixelGather` Kernel: Replaces the host-side **`WordFedAppender`**.

Filtering: The kernel looks at the device-resident index. An event might have 754 total fragments. The kernel checks the numerical fedId, rejects the calorimeters/muon chambers, and isolates only the ~108 fragments belonging to the Pixel Tracker*.

Stripping & Packing: It mathematically skips the 8-byte FED header and trailer, dynamically concatenating only the xx/yy bytes of pure physics payloads into a final device array.

That array is then passed to the device-side unpacker which works on it.

Results

```
GDSRawFileReaderPure: 1 input files, largest 139240672 bytes, 3 device slots of 139243520 bytes (398.379 MiB device)
GDSRawFileReaderPure: [1/1] run402360_ls0214_index000000.raw  139240672 B  read 24.8319 ms (5.60733 GB/s)  scan 2.03451 ms  events 100  fragments 75400
Begin processing the 1st record. Run 402360, Event 483392604, LumiSection 214 on stream 0 at 12-Aug-2026 15:34:18.548 CEST
    digi 0: fed 1338  link 1  roc 2  row 73  col 19  adc 80
    digi 1: fed 1338  link 1  roc 2  row 70  col 19  adc 142
    digi 2: fed 1338  link 1  roc 2  row 69  col 19  adc 50
    digi 3: fed 1338  link 1  roc 2  row 69  col 20  adc 51
    digi 4: fed 1338  link 1  roc 5  row 74  col 30  adc 156
```

GDSPipeline SUMMARY
  events 10
  pixel words 600136
  valid digis 538177 (89.6758%)
  gather (sum) 1.20952 ms 0.120952 ms/event
  decode (sum) 1.4064 ms 0.14064 ms/event
  gather+decode (sum) 2.61592 ms
  event-loop wall 0.00435786 s -> 2294.7 events/s (3.2x faster)
GDSRawFileReaderPure SUMMARY: files 1 bytes 139240672 events 10 fragments 75400
  GDS read 20.5418 ms (6.31452 GB/s aggregate) device scan 2.03451 ms

Hardware Specs:
➔ 4 NVIDIA H100s
➔ INTEL(R) XEON(R) GOLD 6530
➔ 1 stream
➔ Same config: CPU: 714.9 ± 6.6 ev/s

Successfully implemented GDS pipeline. Promising results; consistently outperforms the CPU pipeline.

Conclusion

- ✓ **Successfully implemented the full GDS pipeline**
- ✓ **Complete raw data preprocessing on GPU using CUDA**
- ✓ **Promising speedup (3.2x) compared to the original CPU pipeline**

What's Next?

- ➔ Optimize the phase 1 scanning kernel.
- ➔ Parallelize the GDS Read
- ➔ Make it multi-streamed.
- ➔ Lower GPU memory footprint by making sure all the fragments have a corresponding unpacker kernel.
- ➔ Implement host-side equivalent data structures on the device.

References

https://developer.nvidia.com/blog/gpudirect-storage/

https://indico.cern.ch/event/1700724/contributions/7157200/attachments/3301317/5905627/NGT_CMS_ThirdAllHands.pdf

https://github.com/cms-sw/cmssw

CERN openlab

Thank you

AYAAN A DEWANI

18/08/2026

CERN openlab

Extra

AYAAN A DEWANI

18/08/2026

Zero-Copy Ingestion (GDSRawFileReaderPure.cc)

Custom Source Module (GDSRawFileReaderPure.cc): Designed from scratch to replace CMSSW's CPU FedRawDataInputSource, bridging GPUDirect Storage directly into the framework's Event processing model.

Direct-to-VRAM Storage Transfer: Uses the NVIDIA cuFile API (cuFileRead) to issue direct DMA reads from NVMe storage straight into allocated GPU VRAM. Zero host bounce buffers, zero CPU memory allocations.

On-Device Event Indexing: Immediately launches the GPU scanning kernel (FRDScan.cu) over the raw VRAM buffer. The GPU parses event headers, counts fragments, and populates a device-side metadata array (frdscan::FedEntry) in under 4 milliseconds.

Device-Native Handoff (RawDataDeviceRef): Instead of producing a heavy host FEDRawDataCollection, GDSRawFileReaderPure puts a lightweight gdsraw::RawDataDeviceRef into the CMSSW Event.

The Result: Downstream modules receive raw memory pointers and FED index offsets that are already resident in VRAM, enabling 100% host-bypass for subsequent unpacking.