WP 7.4 – Software, Efficient Analysis task

RNTuple and RDataFrame Status and Roadmap

Jakob Blomer (CERN), Javier López-Gómez (CERN), Vincenzo Padulano (CERN)
for the EP R&D 7.4 (Efficient Anaylsis) task

EP R&D Seminar Series, 2023-07-03

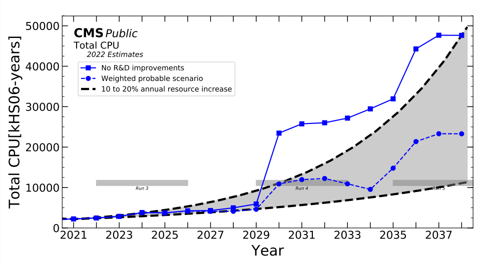
Contents

Introduction

R&D on Efficient Analysis

- Upcoming HEP experiments, e.g. at the HL-LHC, will lead to increased data complexity and $\sim 10\times$ increase in the data rate
- This is an unprecedented challenge in the data processing chain

- Answering how to analyze data at the HL-LHC and FCC scale is crucial for their success
- $> 1\,EB$ LHC data worldwide, estimated 50 M CHF / year spent in WLHCG on storage
  - Smaller file sizes have direct and significant impact on the computing budget



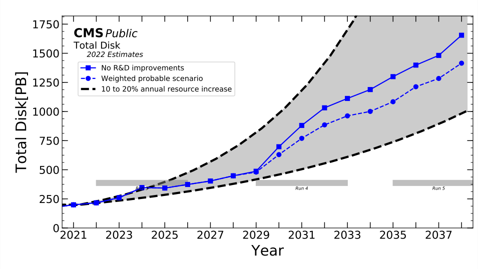


Figure 1: CMS HL-LHC resource projections (July 2022). Source:
`https://twiki.cern.ch/twiki/bin/view/CMSPublic/CMSOfflineComputingResults`

R&D on Efficient Analysis

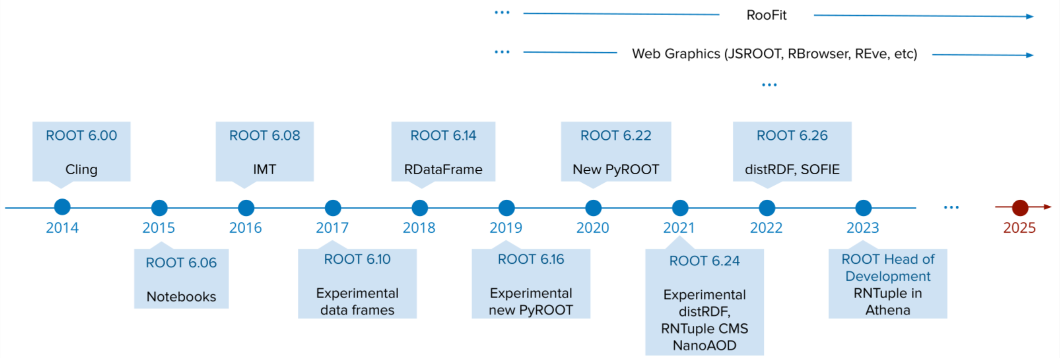
Specifically, for the WP7.4 task (Software for Efficient Analysis), this includes (but not limited to)

- An I/O layer that provides efficient storage and delivers the expected data rate
  - Supporting: **object stores**, **lossy compression**, caching, ...
  - and that leverages **heterogeneous architectures**, e.g. offloading of data (de-)compression to GPUs
- A common, easy-to-use, declarative interface for writing HEP analysis, allowing for
  - Using less time to write the analysis and leaving more time to understand the results
  - Single-core, multi-core, or distributed (multi-node) execution

The ROOT project

- Cornerstone of many software stacks used by HEP experiments at (but not only) CERN
- Foundational data processing libraries
- **Hub for R&D** activities: world-beating random numbers, efficient I/O, ...
- Integrated set of libraries for HENP data analyses
  - I/O, Machine Learning, histograms, fitting, ...
  - ...with RDataFrame being the entry point for writing an analysis
  - The cling C++ interpreter has an important role, and is required, e.g. for I/O
- Two first-class languages:
  - **Python**: Integration and Ergonomics
  - **C++**: high-performance core
- **Community tool: for us by us!**

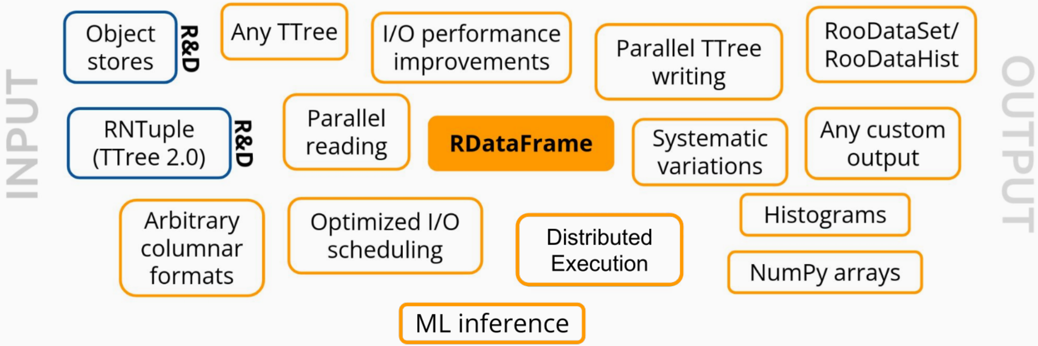
ROOT R&D, Selected Milestones



Plan for 2025: ROOT 7

- First RNTuple production version
- RHist (provided EP R&D funding)
- A more Pythonic ROOT
- Transparent use of GPUs

RDataFrame as ROOT's entry point for analyses



RNTuple: The New Experimental ROOT I/O Subsystem

Reminder: HENP Event Data I/O

Why invest in tailor-made I/O sub system (TTree / RNTuple)

- Capable of storing the HENP event data model: **nested**, **inter-dependent**, **variable-sized** collections of data points
- Performance-tuned for HENP analysis workflow (columnar binary layout, custom compression, etc.)
- **Automatic schema generation and evolution** for C++ (via cling) and Python (via cling + PyROOT)
- Integration with federated data management tools (XRootD, etc.)
- Long-term maintenance and support

Reminder: HENP Event Data I/O

Why invest in tailor-made I/O sub system (TTree / RNTuple)

- Capable of storing the HENP event data model: **nested**, **inter-dependent**, **variable-sized** collections of data points
- Performance-tuned for HENP analysis workflow (columnar binary layout, custom compression, etc.)
- **Automatic schema generation and evolution** for C++ (via cling) and Python (via cling + PyROOT)
- Integration with federated data management tools (XRootD, etc.)
- Long-term maintenance and support

In general, needs not met by standard products in industry (see details in this backup slide).

RNTuple Goals

Based on the experience gained in 25+ years of TTree, RNTuple aims at...

- Less disk and CPU usage for same data content: **25% smaller files,** $> \times 2$ **better single-core performance**
- Systematic use of **data checksums** and **exceptions** to prevent silent I/O errors
- Efficient support for modern hardware:
  - Parallelism on all levels
  - Architectural heterogeneity
  - Direct data transfer to GPU memory
- Native support for **object stores** (targeting HPC)
- Optional: Lossy compression
- Well-defined format with dedicated specification ▶ Here

RNTuple vs. Others: Throughput

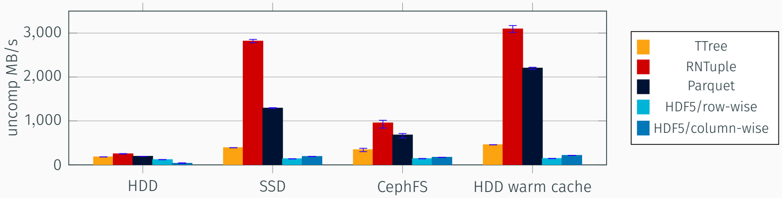


Figure 2: Throughput in uncompressed MB/s for LHCb run 1 OpenData B2HHH

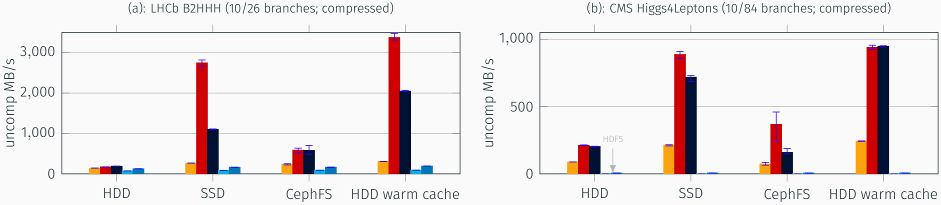
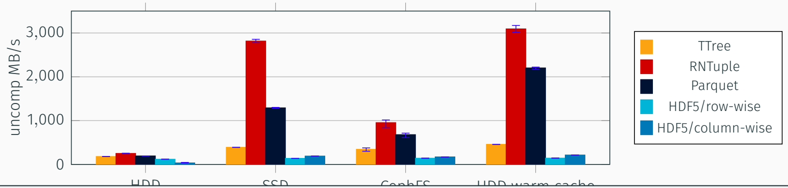


Figure 3: Throughput in uncompressed MB/s reading 10 br., for LHCb (left) and CMS (right) datasets

RNTuple vs. Others: Throughput



- RNTuple I/O delivers the highest read throughput across the board, in all tested scenarios, achieving also better results than well-known alternatives in the industry.

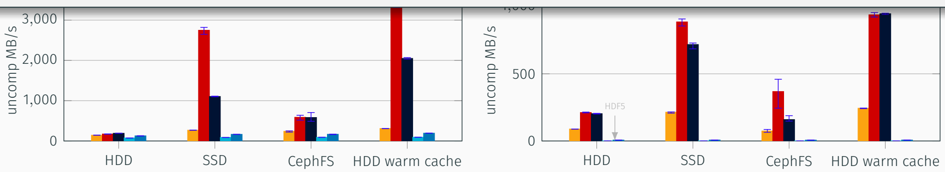
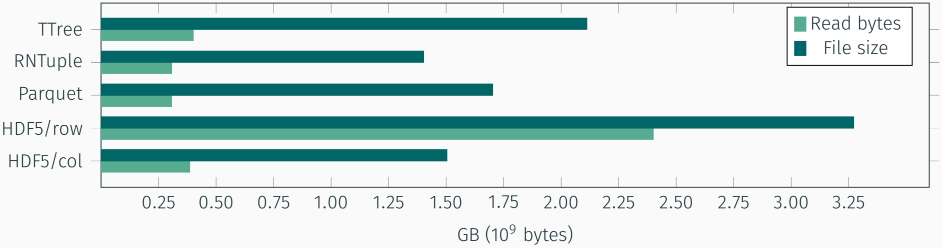


Figure 3: Throughput in uncompressed MB/s reading 10 br., for LHCb (left) and CMS (right) datasets

RNTuple vs. Others: File Size



- Clear advantage of RNTuple over Apache-Parquet and HDF5, both in file size and throughput
- HDF5 results may vary depending on the effort put into adapting inherent tensor layout to columnar access
- Full comparison as of 2021 available at

▶ Lopez-Gomez, J., & Blomer, J. (2022). RNTuple performance: Status and Outlook

Performance: File Size

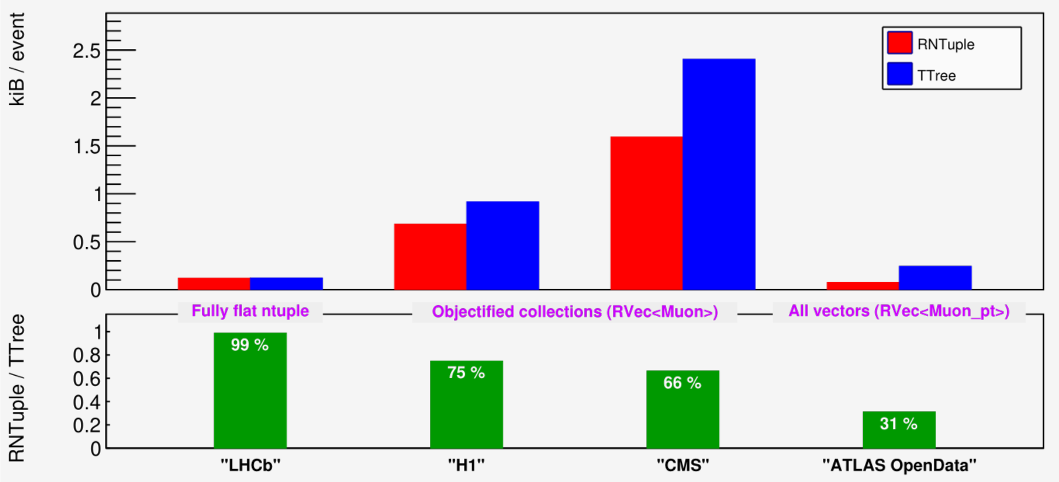
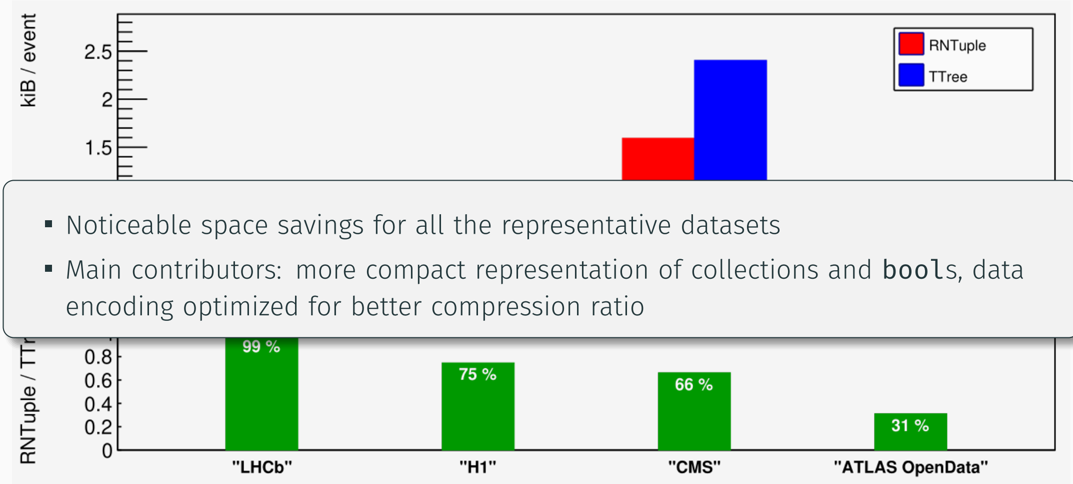


Figure 4: On-disk size, `zstd`-compressed “final stage” ntuples.

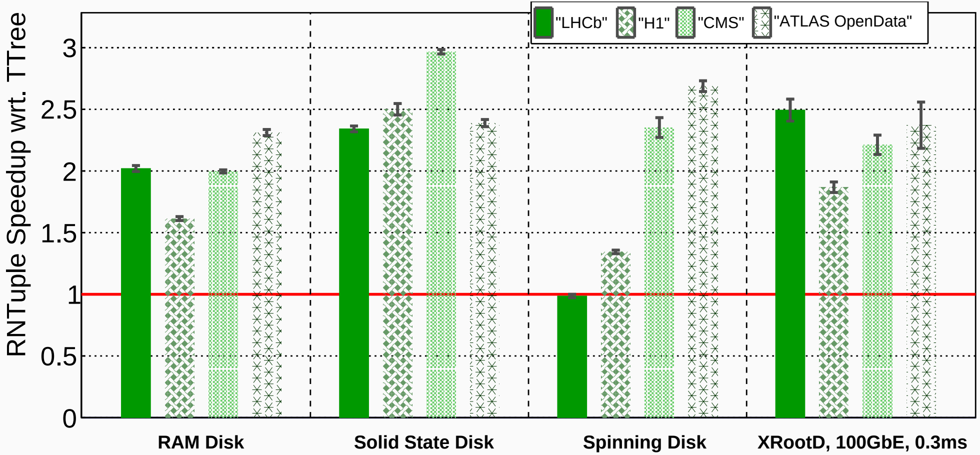
Performance: File Size



- Noticeable space savings for all the representative datasets
- Main contributors: more compact representation of collections and `bool`s, data encoding optimized for better compression ratio

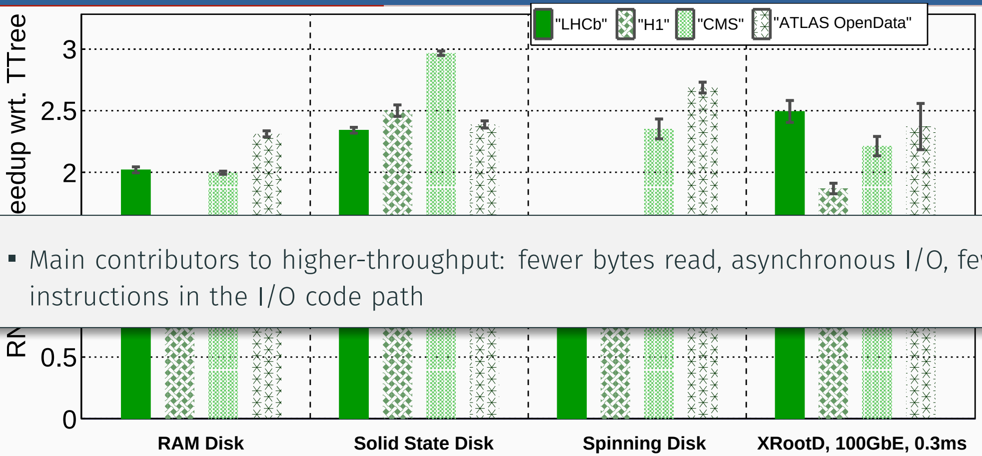
Figure 4: On-disk size, `zstd`-compressed "final stage" ntuples.

Performance: Time-to-Plot



▶ CHEP 2023: ROOT's RNTuple I/O Subsystem: The Path to Production

Performance: Time-to-Plot



- Main contributors to higher-throughput: fewer bytes read, asynchronous I/O, fewer instructions in the I/O code path

▶ CHEP 2023: ROOT's RNTuple I/O Subsystem: The Path to Production

Progress towards full ATLAS and CMS data models

RNTuple supports arbitrary combinations of a well-defined set of C++ types.

<table>
<tr><th>Type</th><th>Examples</th><th colspan="3">EDM Coverage</th><th>RNTuple Status</th></tr>
<tr><td>PoD</td><td>bool, int, float</td><td rowspan="2">Flat n-tuple</td><td rowspan="5">Reduced AOD</td><td rowspan="9">Full AOD / RECO</td><td>Available</td></tr>
<tr><td>Vector<PoD></td><td>RVec<float></td><td>Available</td></tr>
<tr><td>String</td><td>std::string</td><td></td><td>Available</td></tr>
<tr><td>Nested vector</td><td>RVec<RVec<float>></td><td></td><td>Available</td></tr>
<tr><td>User-defined classes</td><td>"TEvent"</td><td></td><td>Available</td></tr>
<tr><td>User-defined collections</td><td>"TCudaVector"</td><td></td><td></td><td>Available</td></tr>
<tr><td>stdlib collections</td><td>std::map, std::tuple</td><td></td><td></td><td>Avail. / Testing</td></tr>
<tr><td>Variadic types</td><td>std::variant, std::unique_ptr</td><td></td><td></td><td>Avail. / Testing</td></tr>
<tr><td>Intra-event references</td><td>"&Electrons[7]"</td><td></td><td></td><td>In design</td></tr>
<tr><td rowspan="3">Low-precision floating points</td><td>Float16_t, Double32_t</td><td colspan="3" rowspan="3">Optimization benefitting all EDMs</td><td>Testing</td></tr>
<tr><td>Custom precision and range</td><td>In design</td></tr>
<tr><td>Precision cascades</td><td>In design</td></tr>
</table>

Format Transition and Compatibility

For maximum optimization opportunities, RNTuple breaks backward compatibility with TTree. At the same time, RNTuple aims at smooth integration with the well-established ROOT/HEP ecosystem.

- For users' code using **`RDataFrame`**: no change[1]!
- Consistent tooling:
  - **`RBrowser`** support
  - Automatic conversion of TTree $\rightarrow$ RNTuple
  - **hadd** support under construction
- RNTuple data stored typically in ROOT files, accessed the usual way. Additionally: transparent object store access (DAOS, S3).
- RNTuple adopts TTree's **I/O customization rules and schema evolution** (WIP)

[1] Coming soon: auto-detection of input format (TTree / RNTuple)

Transition to RNTuple: Collaboration with Experiments

- Frameworks still require their code to be adapted to the new RNTuple interfaces
- Regular meetings with I/O experts from **ATLAS** and **CMS** to...
  - Add support for missing core features, where required
  - Help in adapting experiments' code base to RNTuple I/O
  - Testing and bug fixing

Status on Writing and Deriving Datasets

Entry-by-entry writing

- Available, including multi-threaded writing
- Support for "**late model extension**": allow for accommodating frameworks' on-demand schema definition
- Planned: direct RNTuple output from RDataFrame
- Reducing contention of highly-parallel writes

R&D: Data reshaping: dataset derivation without decompression / deserialization

- Fast merging of files (possibly zero-copy), merging of clusters, discarding columns...
- Under construction!

R&D: Data combinatorics: virtual datasets

- Friends (horizontal combination): Available!
- Chains (vertical combination): Under construction
- Pending EP R&D phase 2 funding: more advanced uses cases, e.g. stored filters, indexed joins, etc.

RNTuple's Path to Production

Goal by the end of 2024: First stable release of RNTuple on-disk format, i.e. promise to keep backwards compatibility.

Requires:

- Comprehensive understanding of LHC experiment requirements for all stages (ESD, AOD)
- Large-scale validation

Early adoption

- **ATLAS:** Experimental support for writing and reading xAOD (PHYS/PHYSLITE) files
- **CMS:** Experimental support for writing nanoAOD files
- **uproot:** Independent implementation of the RNTuple format; validated the ▶ Specification

RNTuple's Path to Production

RNTuple State and Level of Maturity

- Support for **LHCb** analysis ntuples, **CMS RAW** & **nanoAOD**, and **ATLAS DAOD PHYS/PHYSLITE** data models
- Able to handle more demanding / different access patterns wrt. TTree, e.g. leveraging asynchronous I/O scheduling
- Well understood on "lab scale": benchmarks up to 1TB dataset size and 7-node HPC-grade cluster

Goals for the next 18 months

- Evaluate impact of large-scale, distributed RNTuple analysis tasks
- Complete support for full AODs, ESDs
- Complete implementation of data derivation and combination mechanisms (fast merge & clone, chains)
- Integration of lossy data compression
- Schema evolution and validation of forward compatibility
- Input to experiments of data schema and I/O tuning parameters

RNTuple and Object Stores: Support for HPC and Cloud Storage

Why Supporting Object Stores?

- There are known limits to the scalability in parallel filesystems[2]
- Object stores, on the other hand, are based on a flat namespace that contains uniquely-identified objects[3], e.g.
  - DAOS: high-throughput, low-latency, HPC exascale storage system
  - S3: de-facto standard for cloud storage

Thus no hierarchical structure, but instead...

- Massive scalability, optimized for Write-Once-Read-Many
- Reduced complexity
- Resiliency: per-object replication/redundancy
- Cost efficiency

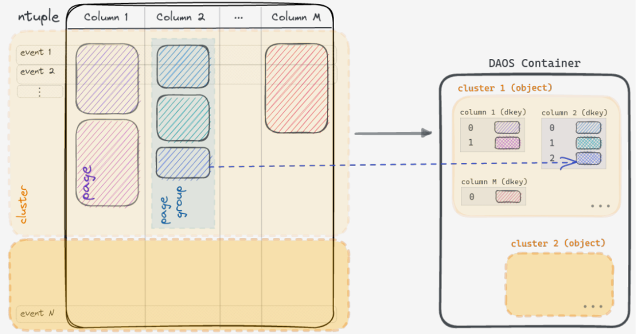
Downside: more logic needs to move into the application layer (ROOT)

[2] e.g. POSIX I/O strong consistency model

[3] See more information here.

RNTuple and Object Stores

- Considerable R&D effort for supporting Object Stores as a first-class storage system
- Currently supported: DAOS (HPC), S3 (Cloud)
- RNTuple's design allow for backends to **conveniently** and **efficiently** map its data structures to objects



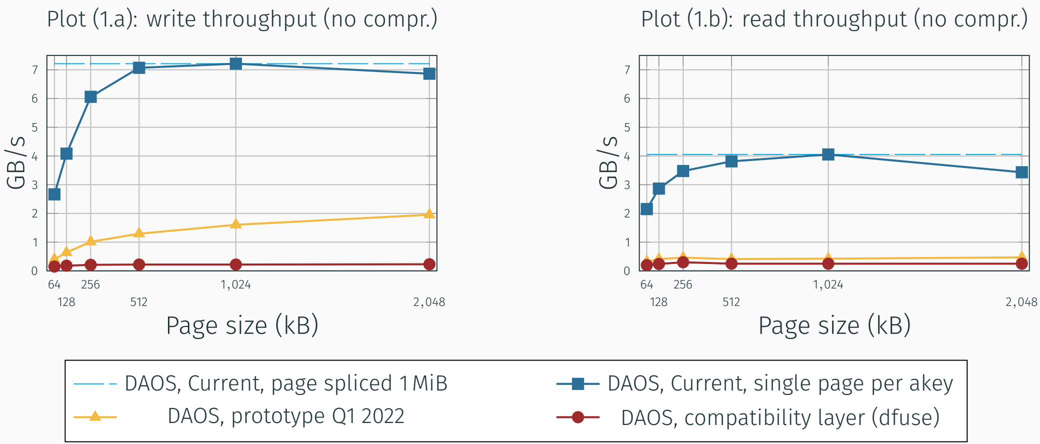
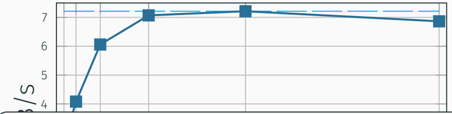
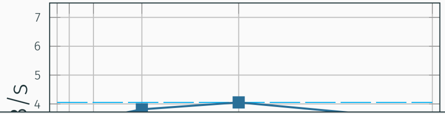


Figure 5: RNTuple + DAOS, LHCb run 1 OpenData B2HHH, single-node. See also: ▸ ACAT 2022

Plot (1.a): write throughput (no compr.)



Plot (1.b): read throughput (no compr.)



- Native access to object stores provides important improvements on read/write throughput over the compatibility layer

Figure 5: RNTuple + DAOS, LHCb run 1 OpenData B2HHH, single-node. See also: ▸ ACAT 2022

Summary and Next Steps

Summary

- Native, mature DAOS backend with 8+ GB/s writes, 4+ GB/s reads single-node
- Native, experimental S3 backend
- Efficient data migration across storage systems, i.e. data reshaping (WIP)

Next steps

- Optimize S3 backend based on first results
- Finalize reshaping of data during the “data move” phase (from grid storage to object stores)

RDataFrame and Distributed Execution

Introduction to RDataFrame

```
df = ROOT.RDataFrame(dataset)
df = df.Filter("x > 0")
        .Define("r2", "x*x + y*y")
rHist = df.Histo1D("r2")
df.Snapshot("newntuple", "out.root")
```

- On this dataset
- Make a cut ($x > 0$)
- Define $r2 = x^2 + y^2$
- Plot $r2$ for events that pass the cut
- Write skimmed data to file

From One Core to Many Cores

```
ROOT.EnableImplicitMT()

# 8<---- Untouched code below
df = ROOT.RDataFrame(dataset)
df = df.Filter("x > 0")
        .Define("r2", "x*x + y*y")
rHist = df.Histo1D("r2")
df.Snapshot("newntuple", "out.root")
```

From One Node to Many Nodes (Dask)

```python
cluster = dask_jobqueue.HTCondorCluster(
    n_workers=64, cores=32)
df = ROOT.RDataFrame(dataset, daskclient=Client(cluster))

# 8<---- Untouched code below
df = df.Filter("x > 0")
        .Define("r2", "x*x + y*y")
rHist = df.Histo1D("r2")
df.Snapshot("newntuple", "out.root")
```

Latest Developments in RDataFrame

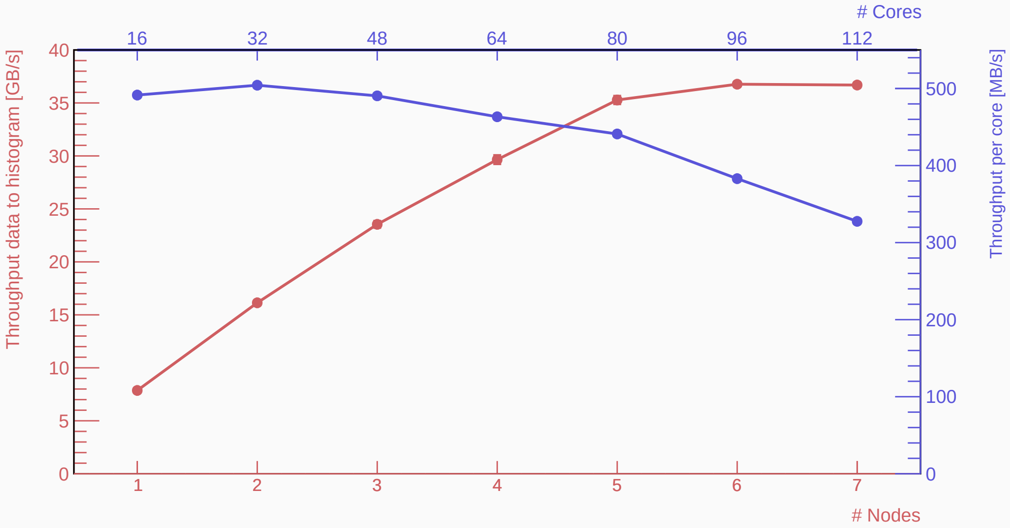
Current State

- Central, high-level entry point for HENP analysis tasks
- R&D on ergonomics: system variations, dataset specs, Pythonic interfaces
- Interoperability with other ROOT components + external software
- Transparent optimizations: bulk I/O, implicit multi-threading, distributed execution

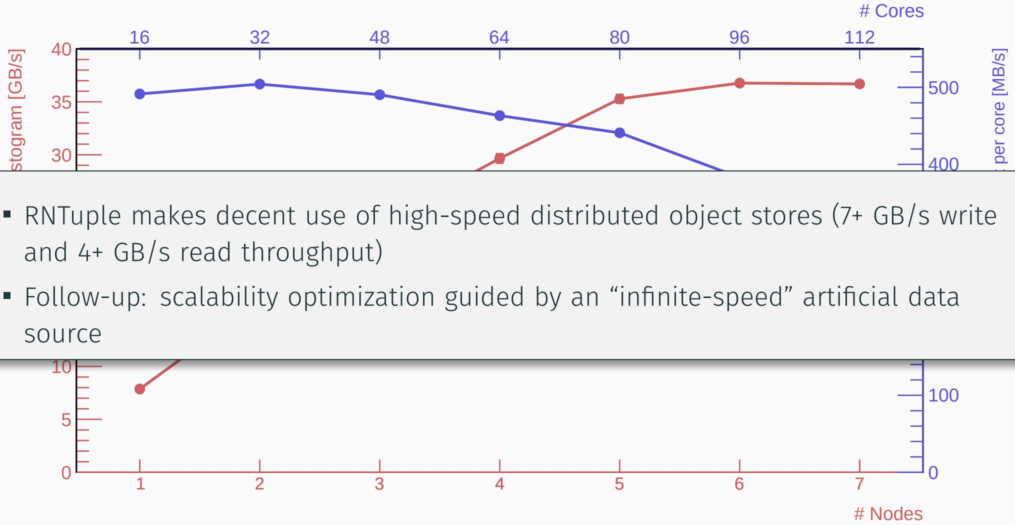
Future Plans

- Column aggregation into "physics objects"
- Python interface improvements, e.g. using Numba instead of passing strings with C++ code
- Transparent use of GPU kernels, e.g. for ML inference in SOFIE

Scaling in Distributed RDataFrame + RNTuple/DAOS



Scaling in Distributed RDataFrame + RNTuple/DAOS



- RNTuple makes decent use of high-speed distributed object stores (7+ GB/s write and 4+ GB/s read throughput)
- Follow-up: scalability optimization guided by an "infinite-speed" artificial data source

Conclusion

Conclusion

Successful R&D in software for Efficient Analysis, **matching the expectations of HL-LHC**, covering:

1. Efficient HENP data storage layer (size- and throughput-wise)
   - **RNTuple is a leap** in storage efficiency, throughput, and usability
   - Successful R&D on support for object stores, which also opened collaboration with third parties: Intel and HPE
2. Common, declarative interface for analysis tasks, capable of
   - Efficient **single-core, multi-core, or multi-node** execution of analyses
   - described using an easy-to-use, storage-agnostic interface

Next steps

- Ongoing work to finalize support for missing features required by main experiments
- More advanced use cases to be covered in 2^{nd} phase of EP R&D programme

Thanks!

References

[1] Vincenzo Eduardo Padulano, Enric Tejedor Saavedra, Pedro Alonso-Jordà, Javier López Gómez, and Jakob Blomer. *A caching mechanism to exploit object store speed in High Energy Physics analysis.* DOI: 10.1007/s10586-022-03757-2.

[2] Giovanna Lazzari Miotto, and Javier Lopez-Gomez. *RNTuple: Towards First-Class Support for HPC data centers* In ACAT'22.

[3] Axel Naumann, Philippe Canal, Enric Tejedor, Enrico Guiraud, Lorenzo Moneta, Bertrand Bellenot, Olivier Couet, Alja Mrak Tadel, Matevz Tadel, Sergey Linev, Javier Lopez-Gomez, Jonas Rembser, Vincenzo Eduardo Padulano, Jakob Blomer, Jonas Hahnfeld, Bernhard Manfred Gruber, Vassil Vassilev. *ROOT for the HL-LHC: data format.* DOI: 10.48550/ARXIV.2204.04557.

[4] Javier Lopez-Gomez, and Jakob Blomer. *RNTuple performance: Status and Outlook.* DOI: 10.1088/1742-6596/2438/1/012118.

References

[5] Jakob Blomer, Philippe Canal, Axel Naumann, Javier López Gómez, and Giovanna Lazzari Miotto. *ROOT's RNTuple I/O Subsystem: The Path to Production*. In CHEP'23.

[6] Florine De Geus, Javier Lopez-Gomez, Jakob Blomer, Marcin Nowak, Peter van Gemmeren. *Integration of RNTuple in ATLASS Athena*. In CHEP'23.

[7] Giovanna Lazzari Miotto, and Javier Lopez-Gomez. *Storing LHC Data in Amazon S3 and Intel DAOS through RNTuple*. In CHEP'23.

[8] Enrico Guiraud, Vincenzo Eduardo Padulano, Enric Tejedor Saavedra, Ivan Kabadzhov, Pawan Pawan. *RDataFrame: A flexible and scalable analysis experience*. In ACAT'22.

[9] RNTuple Reference Specifications. `https://github.com/root-project/root/blob/master/tree/ntuple/v7/doc/specifications.md`.

Backup Slides

ROOT5, ROOT6, and ROOT(7)

- ROOT5 → ROOT6: swapped CINT by the cling C++ interpreter (backwards-compatible)
  - Successful multi-year R&D effort
  - Core component for modern C++ support, PyROOT, I/O, automatic differentiation...
- As of ROOT6, `ROOT::Experimental` namespace for new, backwards-incompatible R&D areas
  - Allows for **immediate preview of new developments**
  - Allows for making new developments **available in production without delay**
  - R&D areas: RDataFrame, web graphics, RNTuple, SOFIE (ML inference), RHist
  - cppyy (and cling) at the core of new PyROOT
  - Many improvements in RooFit; no backwards-incompatible change foreseen
- ROOT 7:
  - All the already delivered R&D components + RNTuple + RHist (pending EP R&D funding)
  - RNTuple on-disk format to be fixed by end of 2024, i.e. targeting 2025 for a ROOT7 release
  - **R&D continuing after the release!**

RNTuple vs. Others: Features

	HDF5	Parquet	TTree	RNTuple
Transparent compression	(1)	●	●	●
Columnar access	(2)	●	●	●
Merging without uncompressing data			●	○
Vertical/horizontal data combinations		/	●	○
C++ and Python support	/	●	●	●
Support for structs/nested collections	?	●	●	●
Architecture-independent encoding	●	●	●	●
Schema evolution			●	○
Support for application-defined metadata	●			○
Fully checksummed	●	●		○
Multi-threading friendly	○	●	●	●
Native object-store support	●	●		●
XRootD support			●	●
Automatic schema creation from C++ classes			●	●
On-demand schema extension (backfilling)			●	●
Split encoding / delta encoding		●		●
Variable-length floats (min, max, bit size)			●	○

● Supported ○ Planned / Under development / Partial / Incomplete ? Unclear
(1) Only for chunked datasets (2) Via emulated columnar

RNTuple Architecture Overview

Event iteration
Looping over events for reading/writing

Logical layer / C++ objects
Mapping of C++ types onto columns, e.g.
`std::vector<float>` $\mapsto$ index column and a value column

Primitives layer / simple types
"Columns" containing elements of fundamental types (`float`, `int`, ...) grouped into (compressed) pages and clusters

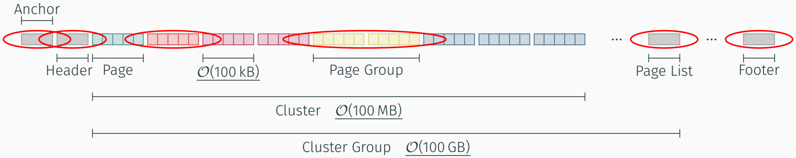
Storage layer / byte ranges
(`RPageSourceXxx`, `RPageSinkXxx`)
POSIX files, **object stores**, ...

Approximate equivalent of TTree and RNTuple classes:

TTree	≈	RNTupleReader
		RNTupleWriter
TTreeReader	≈	RNTupleView
TBranch	≈	RField
TBasket	≈	RPage
TTreeCache	≈	RClusterPool

RNTuple On-disk File Format

```
struct Event {
  int fId;
  vector<Particle> fPtcls;
};
struct Particle {
  float fE;
  vector<int> fIds;
};
```



Page: Array of values of a fundamental datatype[4]

Cluster: All the pages that contain data for a specific row range, e.g. 1–1000

Page group: Pages with data for a specific column within a given cluster

Header / Footer: Information about the schema and location of pages/clusters

[4] Size in the order of a few kibibytes; defaults to 64 KiB.

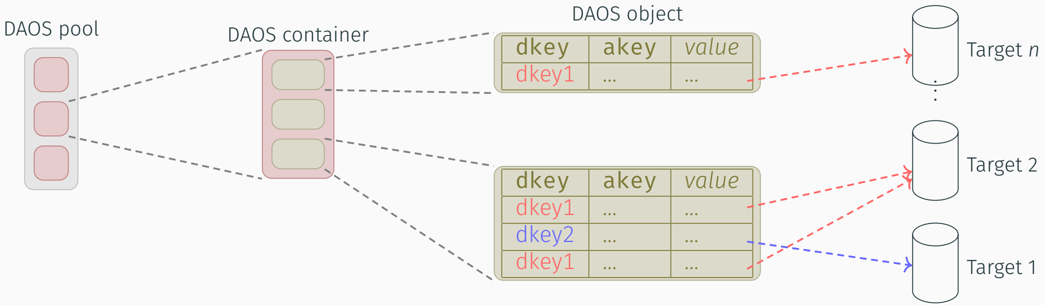
Latest RNTuple Developments

- MERGED Improvements to the type system, e.g. support for `std::bitset`, etc.
- MERGED Storage of user-defined classes behaving as a collection
- MERGED `RField` post-read callbacks, i.e. support for custom ROOT I/O rules
- MERGED Late model extensions
- MERGED Alternative column encodings
- MERGED Projected fields
- MERGED Basic support for Double32 (`double` in memory; `float` on disk)

Coming soon

- DRAFT Support for other STL collections (e.g. `std::set` and `std::map`)
- DRAFT Performance tuning, bulk I/O, etc.

Simplified DAOS top-level organization



Evaluation: RNTuple on S3

E2E analysis on client-server (MinIO) pair over CERN network

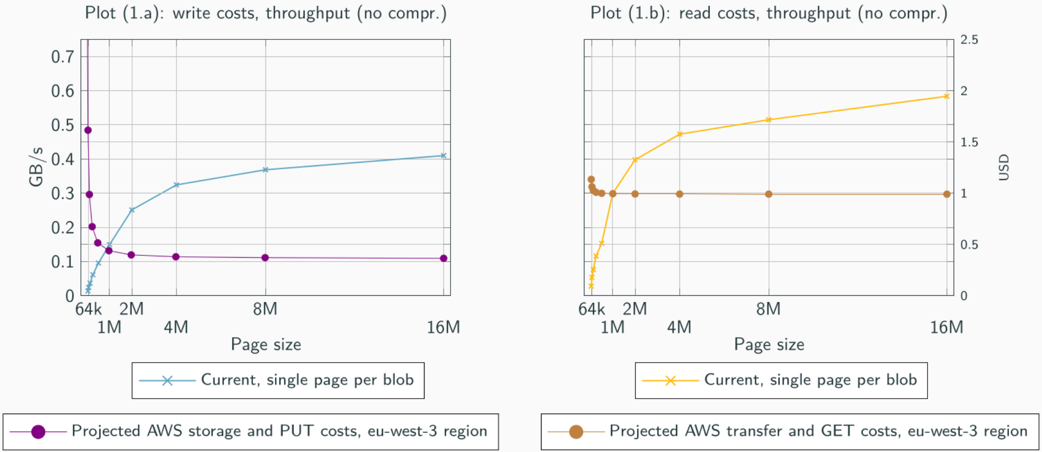


Figure 6: RNTuple + S3, LHCb run 1 OpenData B2HHH, single-node. See also: ▶ CHEP 2023