

RNTuple Workshop 2024 Summary

Jakob Blomer for the ROOT team
SFT Group Meeting
2024-12-16

ROOT

Data Analysis Framework

<https://root.cern>



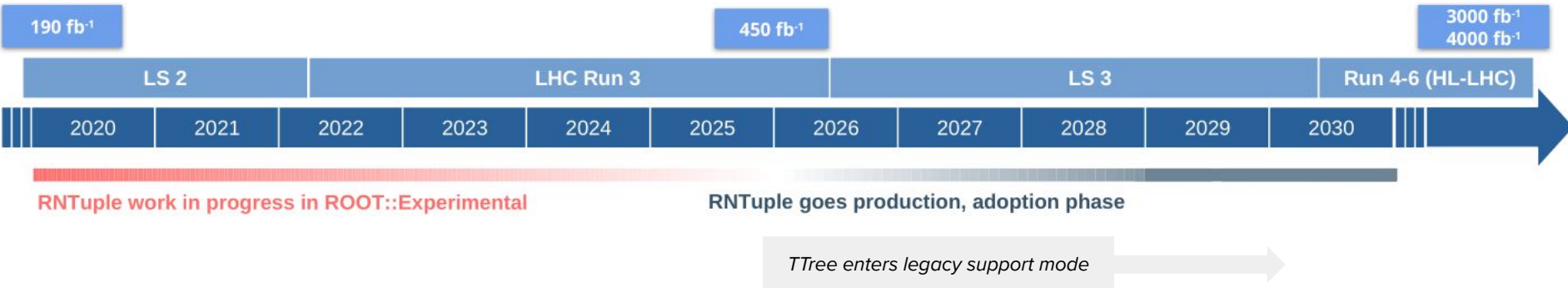
RNTuple Workshop 2024

- [Two afternoons](#)
- 22 Registered participants
 - ~15 in the room
- LHC experiment core software developers present
- [Last year's workshop](#) defined the path to "RNTuple 1.0"
- The RNTuple stable on-disk format was released with ROOT v6.34
 - On time wrt timeline announced to the LHCC and experiments
- This years workshop
 - Concluded the [HEP-CCE](#) API review
 - Discussed new ideas (design stage) and possible development approaches
 - Discuss priorities for 2025





Context: ROOT I/O Upgrade for HL-LHC



>2EB (now) → >10EB (end of HL-LHC)
~½ of the currently projected WLCG budget on storage

Major I/O upgrade of the event data file format and access API: TTree → RNTuple



RNTuple Main Results

- **Major I/O upgrade** of the event data file format and access API: **TTree** → **RNTuple**
 - Less disk and CPU usage for same data content
 - **10-50% smaller** files, **better single-core performance often by factors**
 - Give access to **novel and future storage technologies**
 - Native support for HPC and cloud object stores
 - Async and parallel I/O: fully exploits modern NVMe drives
 - Design prepared for accelerators (e.g., GPUs, compression offloading)
 - Systematic use of checksumming and exceptions to prevent silent I/O errors
- Initial support in **ATLAS, CMS, LHCb, ALICE software frameworks**
(for all data products currently stored in TTree [RECO, AOD, etc.])
- Large-scale testing with IT storage group
 - 70 nodes, 100GbE EOS connection, 100TB inflated AGC benchmark
- ROOT 6.34 (Nov 2024): RNTuple stable on-disk format (version 1.0) released
 - Future ROOT versions will read data written with 6.34
 - Planned optional and possibly forward-compatibility breaking changes foreseen
- ROOT v6.36 (planned for Q2/2025): first set of APIs move out of ROOT::Experimental
 - Taking into account the input received by the HEP-CCE review

Many results presented at CHEP'24

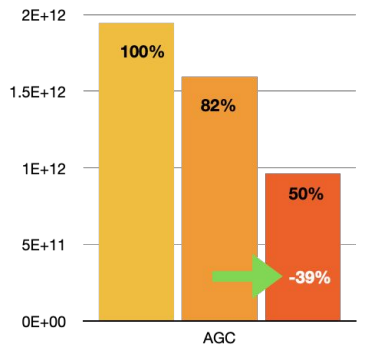
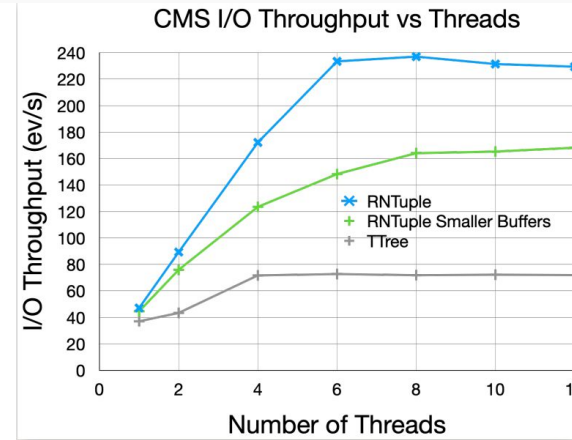
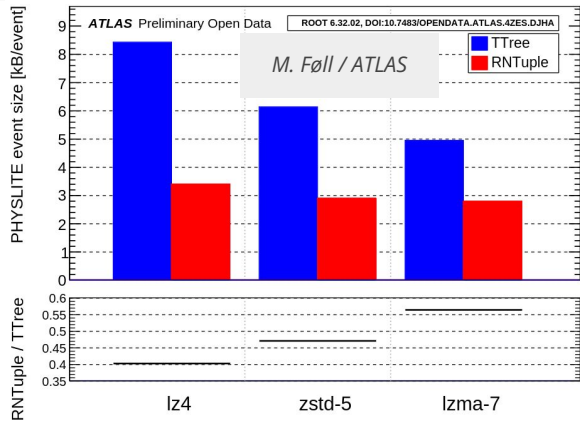


```
tree
├── dataframe
├── doc
├── ntuple
│   ├── inc
│   └── v7
│       ├── doc
│       │   ├── Architecture.md
│       │   ├── BinaryFormatSpecification.md
│       │   ├── README.md
│       │   ├── tuning.md
│       │   └── validation.md
│       ├── inc
│       ├── src
│       └── test
├── CMakeLists.txt
├── ntupleutil
│   ├── inc
│   └── v7
│       ├── inc
│       ├── src
│       └── test
└── CMakeLists.txt
```

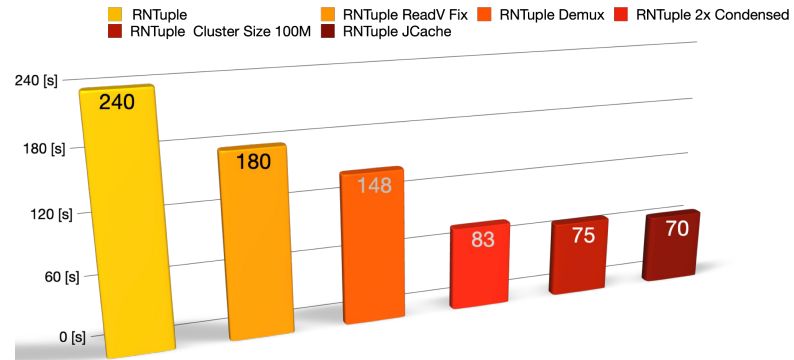
- Part of the ROOT sources
- 40-50k lines of code
 - out of which about 50% tests
- Plus tutorials, benchmarks, extra tools...
- Part of ROOT v7 (in ROOT::Experimental namespace)
- Extra documentation
 - [Architecture.md](#): code architecture documentation for developers and power users
 - [BinaryFormatSpecification.md](#): used as a reference and for 3rd party readers/writers; plan to publish as an independent CERN OPEN report



Highlights from CHEP



■ TTree ZLIB ■ TTree ZSTD ■ RNTuple





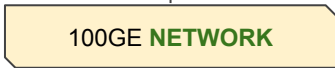
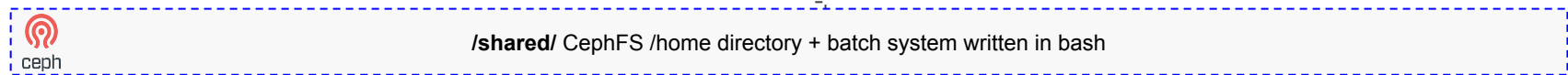
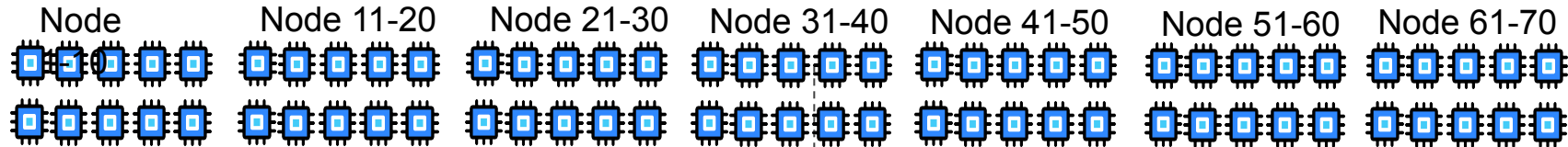
Progress since last year: type system

Type Class	Types	EDM Coverage	RNTuple Status
PoD	<code>bool, char, std::byte, (u)int[8,16,32,64]_t, float, double</code>	Flat n-tuple	Available
Records	Manually built structs of PoDs Type cast of PoDs		
(Nested) vectors	<code>std::vector, RVec, std::array,</code> C-style fixed-size arrays	Reduced AOD	Available
String	<code>std::string</code>	Full AOD / ESD / RECO	Available
User-defined classes	Non-cyclic classes with dictionaries		Available
User-defined enums	Scoped / unscoped enums with dictionaries		Available
User-defined collections	Non-associative collection proxy		Available
stdlib types	<code>std::pair, std::tuple, std::bitset,</code> <code>std::(unordered_) (multi)set,</code> <code>std::(unordered_) (multi)map</code>		Available
Alternating types	<code>std::variant, std::unique_ptr,</code> <code>std::optional</code>		Available
Streamer I/O	All ROOT streamable objects (stored as byte array)		Available
Low-precision floating points	<code>Double32_t, f16</code> Custom precision / range (bfloat16, TensorFloat-32, other AI formats)	Optimization benefitting all EDMs	Available




Progress since last year: AGC Testing I

COMPUTE




Max read 40 GB/s


Price per Volume $\$ \frac{1}{x}$  Relative Price $\$ 4$

FST	FST	FST	FST	FST	FST	FST
FST	FST	FST	FST	FST	FST	FST

EOSPILOT
14 nodes **100GE** 1334x 18TB **HDDs**
24 PB - 20 PB usable




380 GB/s


$\$ \frac{1}{x}$  $\$ 35$

FS T	FS T	FS T	FS T	FS T	FS T	FS T
FS T	FS T	FS T	FS T	FS T	FS T	FS T

EOSALICEO²
125 nodes **100GE** 12000x **HDDs**
180 PB - 150 PB usable



22.5 GB/s

$\$ \frac{6}{12x}$  $\$ 0.8$

OSD	OSD	OSD
OSD	OSD	OSD

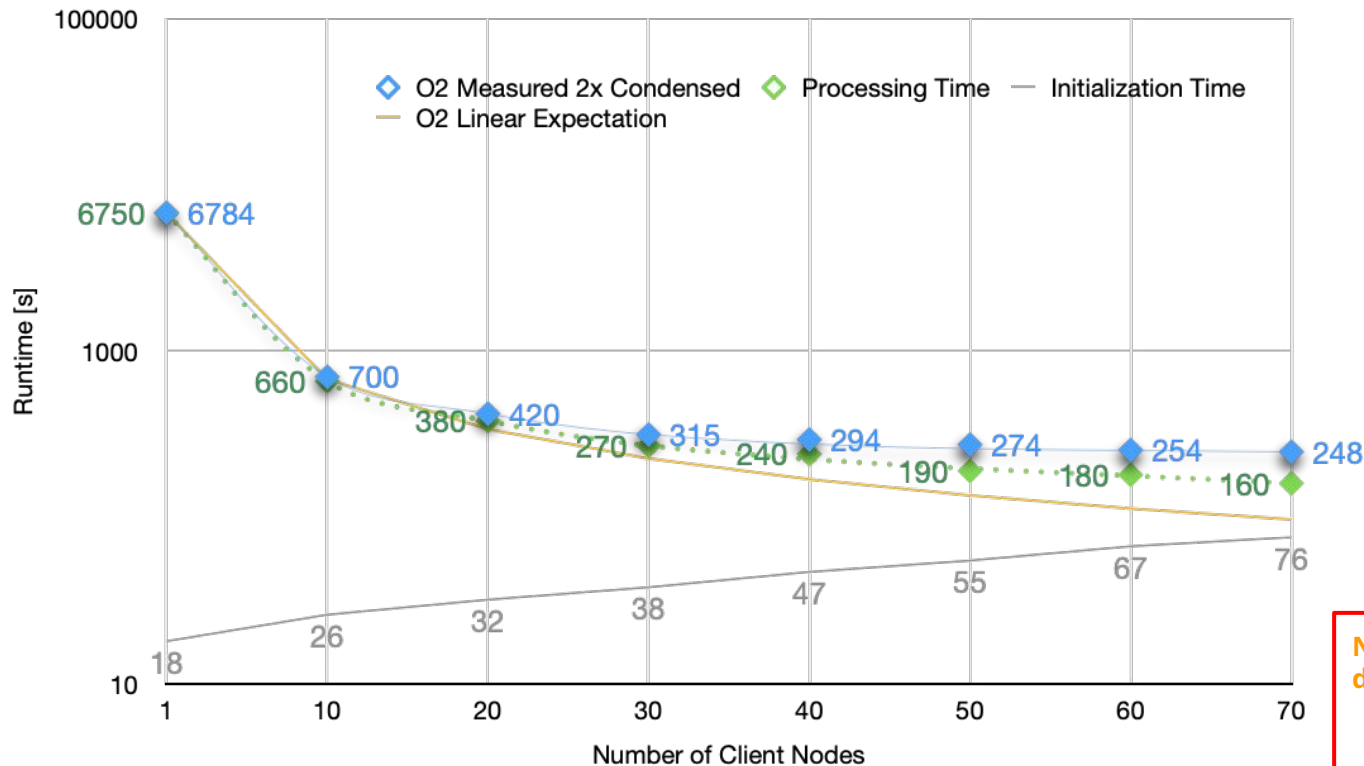
CephFS
8 nodes **25GE**
80 x 7.6 TB **NVMe**
568 TB - 284 TB usable

STORAGE



Progress since last year: AGC Testing II

- Introducing modified RNTuple format for **AGC²⁰⁰** with **EOSALICE²**



With a 100x inflated **AGC²⁰⁰** dataset we observe that as the number of client nodes increases, the initialization time gets close the processing time, resulting in a breakdown of scalability.

Single Analysis
 extremely sparse reaches avg. INGRES
 222 GBit/s during processing
345 GBit/s

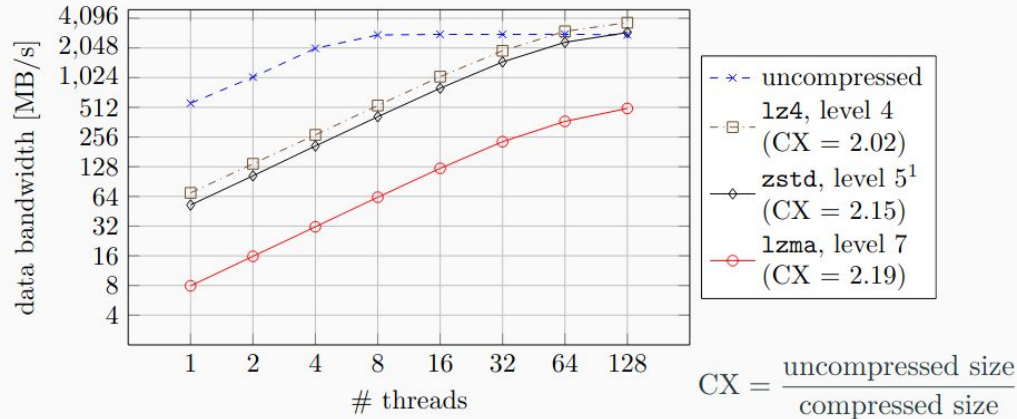
Next step: Reconstruction and/or data derivation benchmark(s)

- Dense reading and (parallel) writing



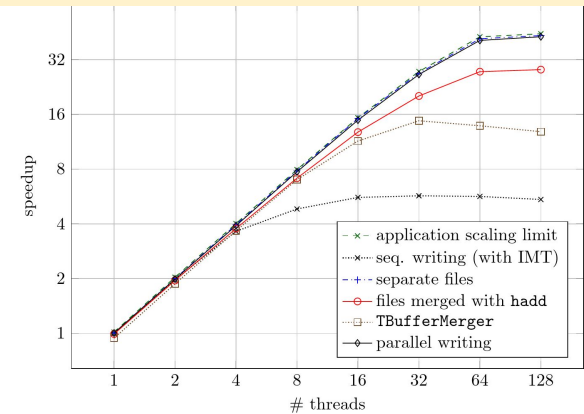
Progress since last year: parallel writing & direct I/O

Reconsider trade-off between write speed and file size



→ <https://indico.cern.ch/event/1338689/contributions/6010002>

Writing in parallel to one file is as fast as writing 128 files



→ <https://arxiv.org/abs/2410.14239>

- Truly parallel writing; prototype support for multi-process and MPI support
- Capable of fully exploiting NVMe drives
- Reaching throughput values that allow for meaningful contribution to processing workflow of DUNE supernova event candidates



Progress since last year

- RNTupleProcessor: friends & chains with solid underpinnings
 - <https://indico.cern.ch/event/1338689/contributions/6016196>
 - See talk by Florine later today:
<https://indico.cern.ch/event/1468611/#3-rntuple-processor-joins>
- Connect RNTuple type description to TFile streamer info (enabling, e.g., MakeProject and manual schema evolution)
- Late model extension in RNTupleMerger (TFileMerger) as well as incremental merging
- Removal of 1GB TFile limit for RNTuple data (exception: streamer field)
- Tested limits: 100k columns, 100k clusters, 600M elements per page
 - Some factor of 10 larger than largest examples we encounter today (e.g., ~15k columns in CMS AOD)

Tooling: RNTupleViewer

RNTupleViewer: <https://codeberg.org/silverweed/rntviewer>

The screenshot displays the RNTupleViewer interface. On the left, a hex dump shows the raw data of a ROOT file, with memory addresses on the left and hex values on the right. On the right, the internal structure of the ROOT file is visualized as a tree. The tree shows the following components:

- TFfile Header: 108 B
- TFfile Object: 74 B
- TFfile Streamer Info: 333 B
- TFfile FreeList: 10 B
- Tkey List: 58 B
- RNTuple Anchor: 78 B
- RNTuple Header: 332 B
- RNTuple Footer: 148 B
- Page Start
- Tkey Header
- Page: 0 (range: - to + 755 B)
- Checksum
- Page List: 0 (range: - to + 244 B)

Below the tree, the ROOT version is 6.35.1, and the TFfile compression is 0. The number of pages is 4, the number of elements is 415, and the number of entries is 22. A list of contributors is shown, including JakobPhilippeAxelDaniloSimon, BertrandMaxJavierEnricoSeraquGio, VannoJerruFlorens WilhelmBernh, ard ManfredVincenzo EduardIoIollu, Alaettin SerhanJonasMaciejGiacom, oGriqorisSpecial thanks! o 4.

Further internal tooling:

- RNTupleInspector (presented last year)
- RNTupleImporter (presented last year)
- RNTupleExporter (dumps pages)

We can imagine a set of power tools, maintained outside the ROOT source tree.

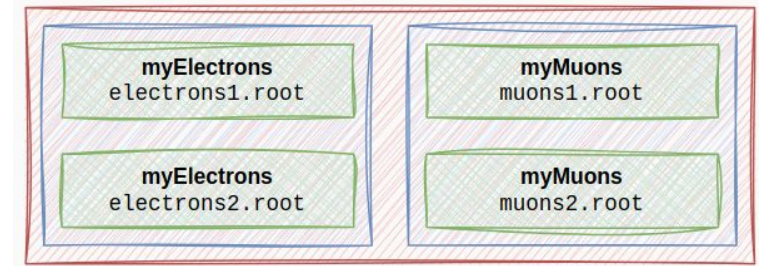
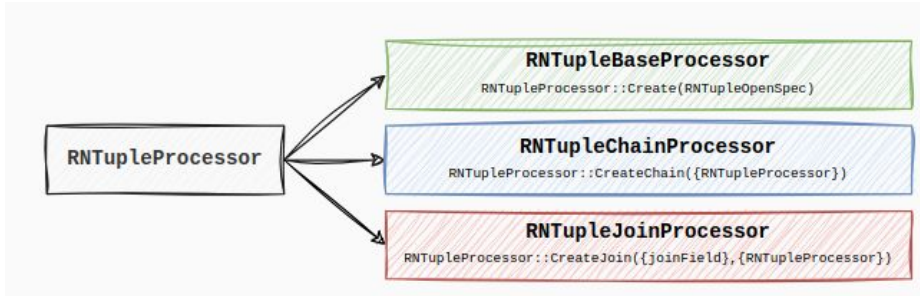
E.g., manual RNTuple descriptor manipulation.



- As part of the last year's ROOT RNTuple Format and Feature Assessment, CCE was asked to host an RNTuple API Review (focusing on framework use)
- Open to all everyone
- Start: March 2024, Reviewers:
 - ATLAS: Marcin Nowak, Serhan Mete, Peter van Gemmeren
 - CMS: Chris Jones, Matti Kortelainen, Dan Riley
 - CAF: Amit Bashyal
 - DUNE: Barnali Chowdhury
 - CCE: Saba Sehrish, Philippe Canal, and several experts from Computer Science
- Produced mid-term report that was submitted to the ROOT team in September 2024
- **Finding that RNTuple API is sufficient for adoption by experiment's production frameworks**
- Parallel to the API Review, experts in CCE shared studies about RNTuple functionality and performance

- → RNTuple API Review was a **very useful exercise**; substantially improved API when it is still cheap to change, **facilitated collaboration** with framework developers





- Presented latest state of the RNTupleProcessor (chains & friends iteration)
- Positive feedback from experiments, addresses current use cases
- Questions and suggestions on index creation and how to persistify the index



```
import ROOT

RNTupleWriter = ROOT.Experimental.RNTupleWriter
RNTupleModel = ROOT.Experimental.RNTupleModel

model = RNTupleModel.Create()
model.MakeField["int"]("f")

with RNTupleWriter.Recreate(model, "ntpl", "ntuple_example.root") as writer:
    entry = writer.CreateEntry()
    entry["f"] = 42
    writer.Fill(entry)
```

- New result from the hackathon: Python accessibility of the RNTuple native API
 - NB: array-oriented programs best served by RNTuple support in RDF
- Side result: implemented passing of unique pointer in PyROOT
- LHCb raised the question of a C API for language bindings
 - This is not a priority of the ROOT team for 2025 but the specification in principle allows 3rd parties to go ahead



Topics: Schema Evolution

- Reading on-disk classes into a changed in-memory shape or changed member semantics

```
class Event {  
    std::string fProperties;  
    float fTemperature;  
    float fX;  
    float fY;  
  
    ClassDef(Event, 2)  
};
```



```
class Event {  
    // no implicit conversion from string to int  
    int fProperties;  
    // Unit change from Kelvin to Celsius  
    float fTemperature;  
    // Change from Euclidian to Polar coordinates  
    float fR;  
    float fPhi;  
  
    ClassDef(Event, 3)  
};
```

- Schema evolution governed by *rules*
 - Implicit (automatic) rules and
 - I/O customization rules / (manual) read rules
 - Extend the capabilities of automatic rules
- RNTuple implements the well-established ROOT I/O schema evolution mechanisms (partially done)
 - Experiment support for cutting non-obvious cases from the automatic rules
 - We discovered that manual rules are lacking expressiveness for some cases (eg, some class hierarchy changes)
 - Manual I/O rules will need an extension of the current mechanism
 - Opportunity to improve documentation and ability to reason about the system



Topics: Schema Evolution, Automatic Rules

	Class Layout Change	RNTuple Support	Comment
Class Members	Reorder members, add member, remove members	Available	
Base Classes (not intermediate)	Add new base class, remove base class	Available	
	Read derived in-memory class from base on-disk class	→ Manual rules	
	Reorder base classes	Tbd (simple)	
Types with identical on-disk representation	<code>std::pair ↔ std::tuple</code>	Available	Recursive evolution, e.g. <code>vector<int32_t> → RVec<int64_t></code>
	<code>std::unique_ptr ↔ std::optional</code>	Available	
	<code>std::vector ↔ ROOT::RVec ↔ collection proxy ↔ std::*set</code>	Available	
	Between <code>std::[unordered_]map</code>	Available	unique constraint check tbd
	Between <code>std::[unordered_]map</code> and sequential collection of <code>std::pair</code>	Available	
PoD transformations (column-level transformation)	Between <code>bool</code> and integral types (except <code>std::byte</code>)	Available	
	Between integral types with bounds checking (except <code>std::byte</code>)	Available	
	Between floating point types	Available	safety check for FP class tbd
Field-level transformations	<code>enum ↔ integral type</code>	Tbd (simple)	
	<code>std::atomic<T> ↔ T</code>	Tbd (simple)	
	<code>std::unique_ptr<T>, std::optional<T> ↔ T</code>	Tbd (intricate)	uni-directional only?
	fixed-sized array ↔ sequential collection	Tbd (intricate)	currently available for RVec only
Class hierarchy changes	Move members between base and derived class	→ Manual rules	Prefer to move to manual schema evolution, if feasible
	Insert or drop intermediate classes	→ Manual rules	



- A future version of RNTuple will include some (internal) metadata
 - Eg, UUID, ROOT provenance information
- Experiments store meta-data (e.g., job configuration) in independent extra trees
 - Does not automatically copy or merge
- Proposal to add RNTuple "attribute sets"
 - Technically each attribute set is its own RNTuple, hard-wired to the main RNTuple
 - Attributes (entries of the attribute set) are attached to data row ranges
 - Merges canonically, but should provide means for user-provided squash function
- First assessment shows some overlap between ROOT internal meta-data case and experiment meta-data
 - But also some differences
 - Needs a second pass to determine if a common denominator can be found



Topics: SoA Data Structures

- Important for data products computed on GPUs
- AoS most efficient on-disk description
 - Single size column
 - Note that the columnar on-disk layout internally performs an AoS → SoA transformation; but SoA layout not easily exposed from TTree
- CMS: AoS → SoA transformation currently done with elaborated I/O read rules
- RNTuple "View" API became flexible enough to read AoS as SoA
- Further work needed on a safe API to write SoA records

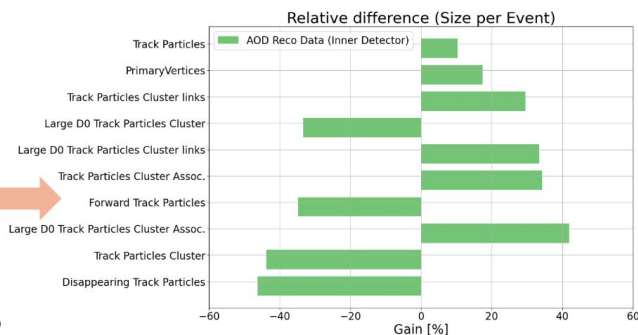
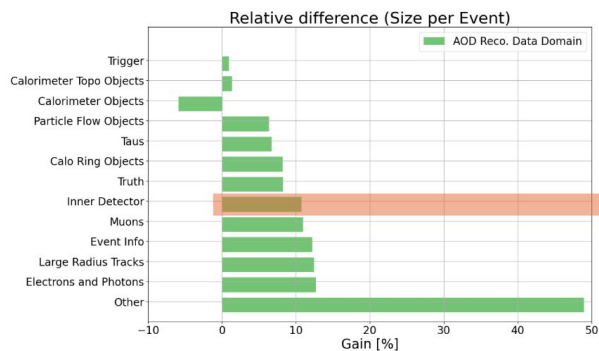
```
struct Point {
    int x;
    int y;
};

using PointsAoS = ROOT::RVec<Point>;

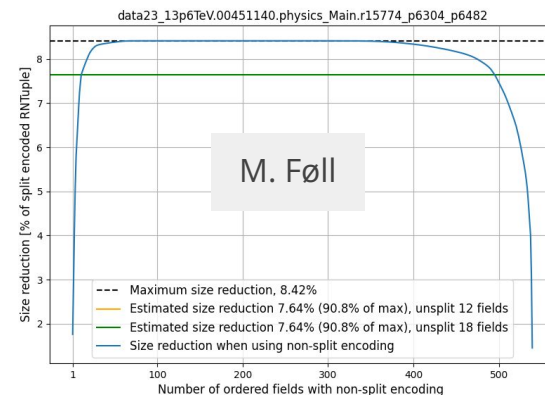
struct PointsSoA {
    ROOT::RVec<int> x;
    ROOT::RVec<int> y;
};
```

```
PointsSoA soa;
auto points_x =
    reader->GetView<decltype(PointsSoA::x)>("points_x", &soa.x);
auto points_y =
    reader->GetView<decltype(PointsSoA::y)>("points_y", &soa.y);
```

- Tuning (auto tuning?) of column encoding



→ <https://indico.cern.ch/event/1338689/contributions/6010824/>



- Investigation of MiniAOD space savings (~7.5 %, would ideally be > 10% [somewhat arbitrary])
- Framework support: profile & improve writing and reading from frameworks
- Support for vectors with custom allocators (ATLAS)
- Bulk reading optimizations (ALICE)
- Validation suite for 3rd party readers



- Define the first set of APIs to move out of ROOT::Experimental
 - Planned for ROOT v6.36, i.e. likely May 2025
 - More or less the classes subject to the HEP-CCE review
 - We can extend the APIs later (e.g. additional ClusterPool tuning), but once in production it will be costly to change existing APIs
 - Not all RNTuple APIs will move out at the same time
- Fully functional schema evolution (basic functionality working for v6.36, full set possibly post v6.36)
- RNTupleProcessor: capability to arbitrarily combine friends and chains
- RNTuple attribute extension prototype, likely leading to v1.1 ondisk format
- Testing and validation on IT testbed with data derivation and/or reconstruction benchmark(s)
- Tuning, support, bug fixes, training: with the transition to production, the support effort begins
- *Lower priority:* S3 backend, intra-event links, checkpoints during writing, C API, sharded clusters and horizontal merge



- 2024 was a critical year for RNTuple
 - Came from not being able to store any CMS data products except nano to being able to store all of them
 - Many more important challenges solved, in important areas for multiple experiments
 - → All experiments can store all their central data products in RNTuple format
 - → Released the first stable on-disk format in time!
- We had many open questions at the end of last year...
 - ... and we have a full plan of work (even beyond 2025) by the end of this year
 - Solved enough problems to have confidence that we can solve the remaining ones, too
- We are entering a new phase in the RNTuple life cycle
 - Transition to production comes with maintenance responsibility: support, training, bug fixes
 - Expect slower pace regarding new functionality
- RNTuple (I/O) workshops are important events to solicit input from the I/O experts of *multiple* experiments
 - Reality check for new designs
 - Align development priorities with experiment needs