Development Status of RNTuple: the future HEP Columnar Storage Software Technology

Danilo Piparo for the ROOT team (CERN, EP-SFT)

15-05-2024

*For more information about ROOT, and not only RNTuple, please see **this talk at the WLCG/HSF Workshop 2024**!*

What is RNTuple?

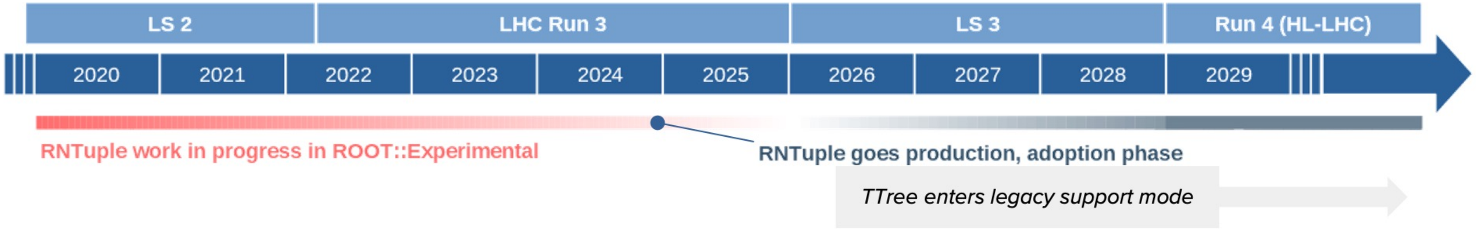
- RNTuple is the **successor of TTree**, ROOT's columnar storage technology
- RNTuple is **part of current ROOT releases, approaching production quality** (e.g. fixed file format, stable interfaces)
    - **By the end of 2024 the file format on disk will be frozen (RNTuple 1.0)**: read back what was written in RNTuple. The C++ interfaces will continue to evolve.
- RNTuple stores fundamental types, arrays thereof **but also sophisticated data models* that characterise our science**
- RNTuples are **stored in ROOT files**
- For SW frameworks, RNTuple requires code migration from TTree interfaces; it is a **drop-in replacement for RDataFrame based analyses** (no user code changes)

* Yes, this still needs reflection

Why RNTuple?

Based on 25+ years of TTree experience, RNTuple is a redesigned columnar I/O subsystem aiming at:

- **Less storage, compute and network usage**
  - Significantly smaller files and higher throughput, often by factors
- Systematic use of **data checksums** and runtime **exceptions** to prevent silent I/O errors
- Efficient support of **modern hardware**
  - Asynchronous & parallel I/O + many-core friendly + direct data transfer to GPU memory
  - **Native support for object stores** in addition to local and remote ROOT files, but not all of the TTree features
- **Binary format defined in a dedicated specification**



ROOT's Strategic Goals

Provide a unified software package for the storage, processing, visualisation and analysis of scientific data that is reliable, performant and supported, that is easy to use and obtain, and that *minimises the computing resources needed to achieve scientific results.*

RNTuple also fits well our strategic goals.

Context of RNTuple

QoS	ALICE	ATLAS	CMS	LHCb	*Total*
Disk [PB]	199	406	304	93	1002
Tape [PB]	283	666	673	250	1875

`24 Pledges: source [CRIC](#)

2024 Disk + tape pledges at T{0,1s,2s}: ~2.9 EB

- Modulo notable exceptions (e.g. ATLAS/LHCb raw), **that space is (going to be) used for data in ROOT format, mainly columnar (TTree): ROOT DOES SCALE.**
- **Storage pledged at Grid sites is not the only one used for ROOT files**: e.g.T3s, university clusters, personal laptops, analysis facilities, cloud...

RNTuple: the great responsibility and opportunity to build on 25+ years of success and experience of TTree

4. The ROOT I/O system

One of the basic pillars of the ROOT system is its hierachical object database. The database is designed to be particularly efficient for objects frequently manipulated by physicists: histograms, ntuples, trees and events.

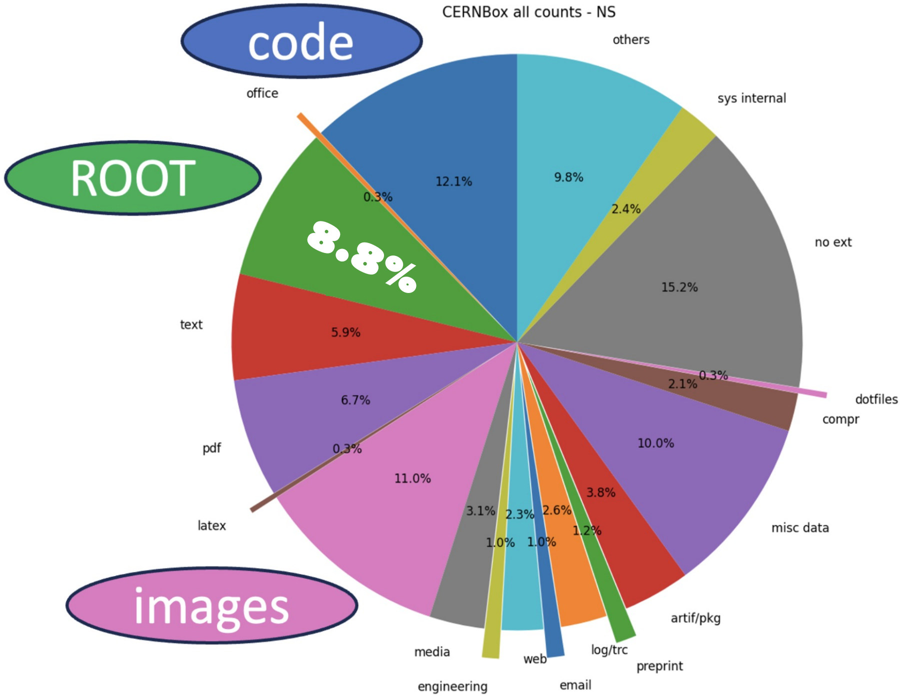
One could argue that this functionality can also be provided by a full fledged commercial Object Oriented Data Base Management System (OODBMS). We consider OODBMSs as potential candidates for the replacement of tools like HEPDB [4] or FATMEN [5], i.e. when locking and concurrent writing is required. But we do not believe that they provide a solution for the types of objects mentioned above. Why?

- Interactive computing is towards commodity desktop and notebook devices. They will be heavily used for histogram manipulation and data presentation. This should not require a special connection to a central data base or a license server (think of home computing).
- OODBMSs, by definition, are designed to store complete objects. Data clustering is organized around objects and containers of objects. They are not designed to access only a subset of the object attributes. We have demonstrated with the PAW column-wise Ntuples the usefulness of having access to single attributes. The ROOT Tree functionality cannot be provided in an efficient way by the current OODBMSs.
- OO data bases do not support on the fly data compression. We are designing experiments that will generate massive amounts of data. The cost of direct access devices for tens of terabytes may be a dominant factor in the cost of computing.
- Attribute range specification is not supported. A 4 byte integer cannot be saved as a single byte.
- The data bases companies are small and fragile. Will they survive after a few years? The technology is not yet mature and compatibility between vendors is not guaranteed.

IIc. INTERACTIVE ANALYSIS

See here for [more info](#)

2.9 EB is an Underestimate: CERNBox

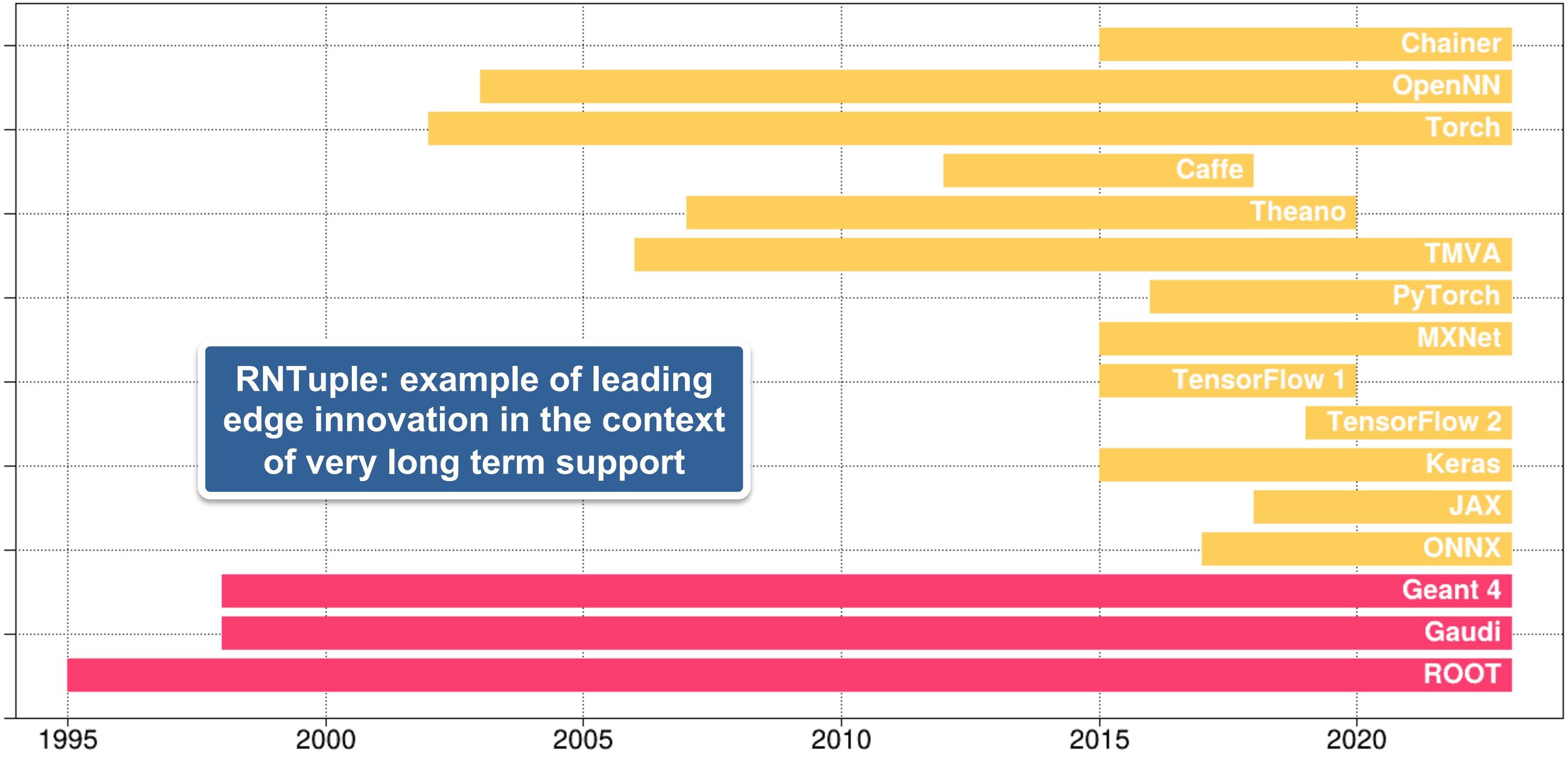


Courtesy of G. Lo Presti and D. Castro (CERN IT)

Just one example: ROOT files on CERNBox, CERN's Sync'n'Share storage

- 8.8% of the files are ROOT files
- ROOT files are 40% of the volume (~4.4 PB out of 11PB)

HEP Software Support Timeline



Plot inspired by [M. Mazurek]

Technical Insights, Programming Model and Performance

HW Aware SW Development

Next-Generation Experiments: HL-LHC, DUNE, EIC

- From 300fb-1 in run 1-3 to 3000fb-1 in HL-LHC run 4-6
- **Single events in the multi-gigabyte range for DUNE**
- **As a starting point, preparing for ten times the current demand**

Full exploitation of modern storage hardware

- Fast networks and SSDs: 10GB/s per device reachable (HDD: 250MB/s)
- Flash storage is inherently parallel → asynchronous, parallel I/O is key
- Heterogeneous computing hardware → GPU should be able to load data directly from SSD, e.g. to feed ML pipeline
- Distributed storage systems move from POSIX to object stores

This has consequences for our software, for example: **at 10GB/s, we have ~3μs to process a 32kB block → CPU optimizations deep into I/O stack**

RNTuple Compatibility Overview

Two principles:

1. A **new event data format** and a **new API:** maximise opportunities for optimisations
2. At the same time, RNTuple aims at **smooth integration with the ROOT/HEP ecosystem**.

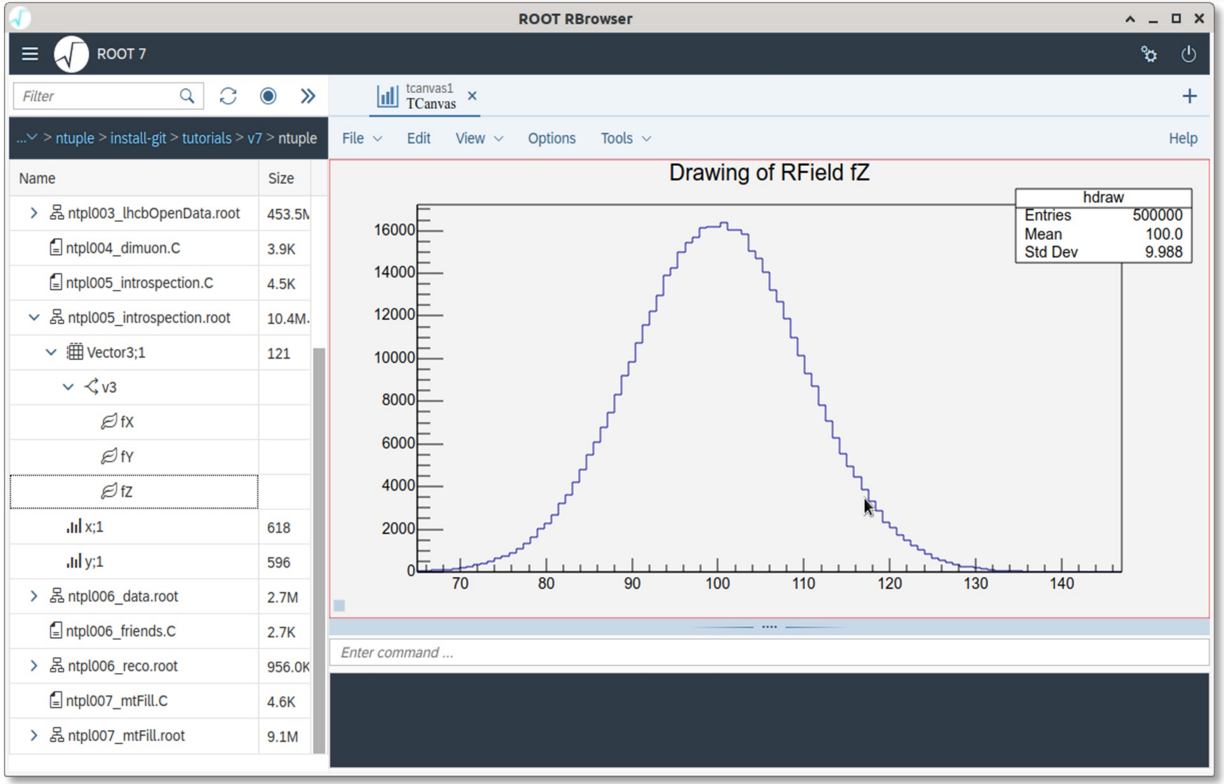
- RNTuple API for writing and reading, targeting frameworks: **follows modern C++ core guidelines**
- For **RDataFrame code: no change required**
- Consistent tooling
  - **RBrowser** support
  - **Disk-to-disk importer** TTree → RNTuple [1] [2]
  - **hadd** support

```
root [1] .ls
TFile**         /data/gg_data.root
 TFile*         /data/gg_data.root
  KEY: TTree    mini;55 mini [current cycle]
  KEY: TTree    mini;54 mini [backup cycle]
  KEY: ROOT::Experimental::RNTuple      mini_imported;
```

A TTree and an RNTuple in the same ROOT file. In this example, the RNTuple data has been converted from the tree using the `RNTupleImporter`.

- **RNTuple adopts TTree's I/O customization and schema evolution system** (work in progress)
- TTree::Draw will not be replicated "as-is" in RNTuple; a possible replacement on top of RDataFrame is under discussion

RNTuple in the RBrowser



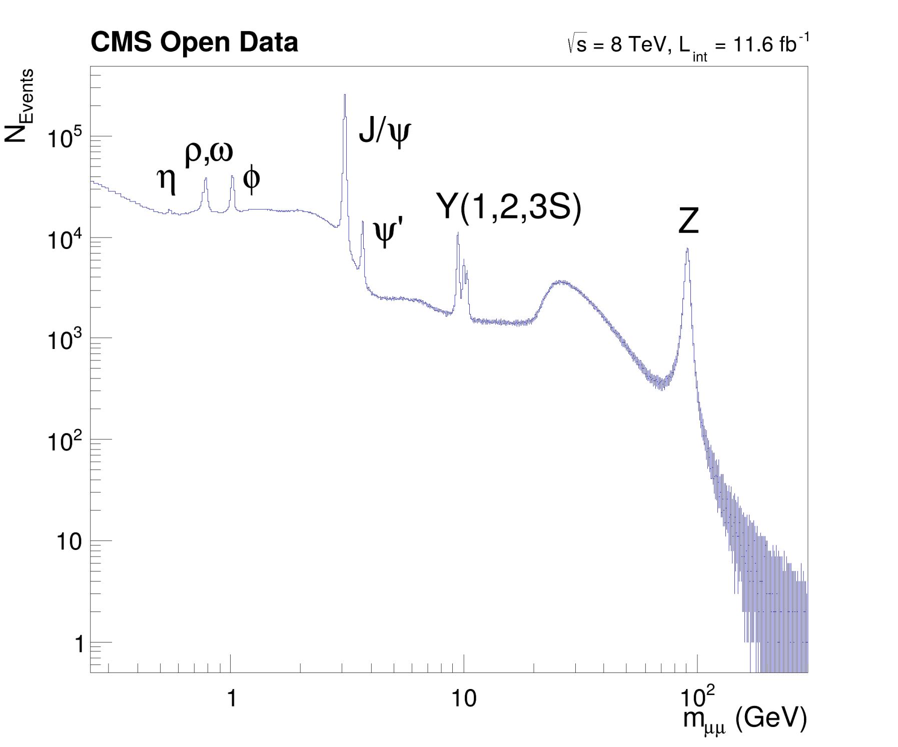
Yes, we are modernising ROOT's GUI and graphics system, too. Will be part of ROOT 6.32.00: a simple switch to activate it.
`$ root --web` *or, programmatically,* `gROOT->SetWebDisplay()`. **As usual, happy to get feedback and improve!**

RDataFrame Analysis with TTree

```python
import ROOT

# Enable multi-threading
ROOT.ROOT.EnableImplicitMT()

# Create dataframe from NanoAOD files
df = ROOT.RDataFrame("Events",
    "root://eospublic.cern.ch//eos/opendata/cms/derived-data/AOD2NanoAODOutreachTool
        /Run2012BC_DoubleMuParked_Muons.root")

# For simplicity, select only events with exactly two muons and require opposite charge
df_2mu = df.Filter("nMuon == 2", "Events with exactly two muons")
df_os = df_2mu.Filter("Muon_charge[0] != Muon_charge[1]", "Muons with opposite charge")

# Compute invariant mass of the dimuon system
df_mass = df_os.Define(
    "Dimuon_mass", "InvariantMass(Muon_pt, Muon_eta, Muon_phi, Muon_mass)"
)

# Make histogram of dimuon mass spectrum. Note how we can set titles and axis labels in one go.
h = df_mass.Histo1D(("Dimuon_mass", "", 30000, 0.25, 300), "Dimuon_mass")

# Request cut-flow report
report = df_mass.Report()

# Produce plot
ROOT.gStyle.SetOptStat(0)
ROOT.gStyle.SetTextFont(42)
c = ROOT.TCanvas("c", "", 800, 700)
c.SetLogx()
c.SetLogy()

h.SetTitle("")
h.GetXaxis().SetTitleSize(0.04)
h.GetYaxis().SetTitleSize(0.04)
h.Draw()

label = ROOT.TLatex()
label.SetNDC(True)
label.DrawLatex(0.175, 0.740, "#eta"); label.DrawLatex(0.205, 0.775, "#rho,#omega")
label.DrawLatex(0.270, 0.740, "#phi"); label.DrawLatex(0.400, 0.800, "J/#psi")
label.DrawLatex(0.415, 0.670, "#psi'"); label.DrawLatex(0.485, 0.700, "Y(1,2,3S)")
label.DrawLatex(0.755, 0.680, "Z")
label.SetTextSize(0.040); label.DrawLatex(0.100, 0.920, "#bf{CMS Open Data}")
label.SetTextSize(0.030); label.DrawLatex(0.630, 0.920, "#sqrt{s} = 8 TeV, L_{int} = 11.6 fb^{-1}")
```

TTree Version

For more details see the di-muon analysis tutorial



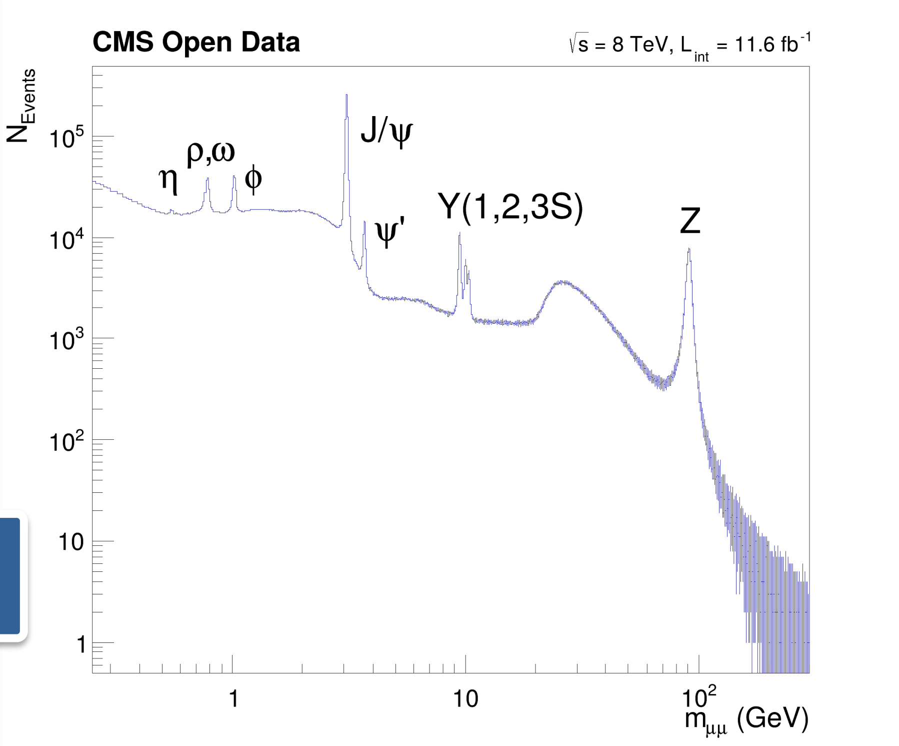
RDataFrame Analysis with TTree

```python
import ROOT

# Enable multi-threading
ROOT.ROOT.EnableImplicitMT()

# Create dataframe from NanoAOD files
df = ROOT.RDataFrame( "Events", "http://root.cern/files/tutorials/ntpl004_dimuon_v1rc2.root")

# For simplicity, select only events with exactly two muons and require opposite charge
df_2mu = df.Filter("nMuon == 2", "Events with exactly two muons")
df_os = df_2mu.Filter("Muon_charge[0] != Muon_charge[1]", "Muons with opposite charge")

# Compute invariant mass of the dimuon system
df_mass = df_os.Define(
    "Dimuon_mass", "InvariantMass(Muon_pt, Muon_eta, Muon_phi, Muon_mass)"
)

# Make histogram of dimuon mass spectrum. Note how we can set titles and axis labels in one go.
h = df_mass.Histo1D(("Dimuon_mass", "", 30000, 0.25, 300), "Dimuon_mass")

# Request cut-flow report
report = df_mass.Report()

# Produce plot
ROOT.gStyle.SetOptStat(0)
ROOT.gStyle.SetTextFont(42)
c = ROOT.TCanvas("c", "", 800, 700)
c.SetLogx()
c.SetLogy()

h.SetTitle("")
h.GetXaxis().SetTitleSize(0.04)
h.GetYaxis().SetTitleSize(0.04)
h.Draw()

label = ROOT.TLatex()
label.SetNDC(True)
label.DrawLatex(0.175, 0.740, "#eta"); label.DrawLatex(0.205, 0.775, "#rho,#omega")
label.DrawLatex(0.270, 0.740, "#phi"); label.DrawLatex(0.400, 0.800, "J/#psi")
label.DrawLatex(0.415, 0.670, "#psi'"); label.DrawLatex(0.485, 0.700, "Y(1,2,3S)")
label.DrawLatex(0.755, 0.680, "Z")
label.SetTextSize(0.040); label.DrawLatex(0.100, 0.920, "#bf{CMS Open Data}")
label.SetTextSize(0.030); label.DrawLatex(0.630, 0.920, "#sqrt{s} = 8 TeV, L_{int} = 11.6 fb^{-1}")
```

RNTuple Version

Identical code. Just change the input files: like before, but better.

For more details see the di-muon analysis tutorial



RNTuple Binary Format

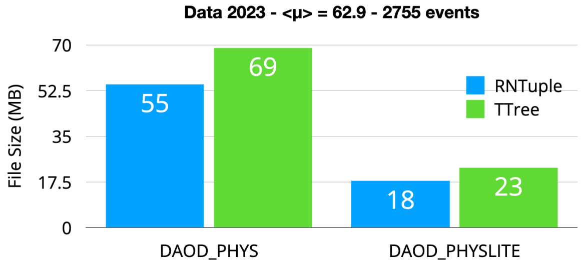
Benefits of a new binary format:

- **More efficient storage of collections** and boolean values
- Addition of **new basic types, e.g. f16**
- **Little-endian** numbers: memory mappable on most contemporary platforms
- **Type-based encoding**: e.g. zig-zag for signed ints, bit packing for bools, etc.
- **Split storage for arbitrarily nested collections**
- More scalable meta-data, **better memory control**
- **New default compression**: zstd
- Format independent of TFile

***Technical Note:** RNTuple has its own schema encoding, independent of the streamer info*

```
root [1] .ls
TFile**         basic2.root
 TFile*         basic2.root
  KEY: TTree    ntuple;1        data from ascii file
  KEY: ROOT::Experimental::RNTuple      imported;1      object title
root [2] _file0->Map()
20231028/012556  At:100     N=118       TFile
20231028/012556  At:218     N=3824      TBasket        CX =  1.06
20231028/012556  At:4042    N=3826      TBasket        CX =  1.06
20231028/012556  At:7868    N=3754      TBasket        CX =  1.08
20231028/012556  At:11622   N=511       TTree          CX =  3.55
20231028/013026  At:12133   N=65        FreeSegments
Address = 12198 Nbytes = -4750  =====G A P===========
20231028/013026  At:16948   N=176       RBlob          CX =  1.66
20231028/013026  At:17124   N=3745      RBlob          CX =  1.08
20231028/013026  At:20869   N=3728      RBlob          CX =  1.08
20231028/013026  At:24597   N=3517      RBlob          CX =  1.15
20231028/013026  At:28114   N=126       RBlob          CX =  1.32
20231028/013026  At:28240   N=128       RBlob          CX =  1.30
20231028/013026  At:28368   N=134       ROOT::Experimental::RNTuple
20231028/013026  At:28502   N=185       KeysList
20231028/013026  At:28687   N=4909      StreamerInfo   CX =  3.11
20231028/013026  At:33596   N=1         END
root [3]
```

ATLAS and RNTuple



Conclusions and Outlook

- **A Rough RNTuple timeline from ATLAS' perspective:**

LHC Run 3			Long Shutdown 3			LHC Run 4 (HL-LHC)
... 2023	2024	2025	2026	2027	2028	2029 onward
Experimental Development			Adoption, Testing, Validation			Production

★ We are here!

- **The current plan is to adopt RNTuple for (at least) the Event Data for Run 4**
  - Discussions on how to handle in-file Meta Data is currently ongoing
- **All in all we're in a very good position but there is much work ahead of us!**
  - All aspects need to be rigorously tested and validated well in advance of Run 4
    - Multi-process/thread Athena jobs, complementary tools, benchmarking, and optimizations
- **We're looking forward to all of the fun ahead!**

13

Towards Getting Production Ready

- **Being able to read/write our data in RNTuple is a great start!**
  - **ATLAS can read/write all data formats, i.e., HITS, RDO, ESD, AOD, and DAOD in RNTuple!**
- **However, there are many other features that are needed for production**
  - Fast merging of RNTuple objects on-the-fly and custom entry/event indexing
    - These are primarily needed for the DAOD production workflows
    - These jobs run in multi-process Athena where a dedicated process merges worker outputs on-the-fly
  - Having various utilities/tools to peek into, compare, validate, … RNTuples
    - These are needed for job configuration, input/output validation etc.
  - Relational RNTuples, a.k.a. *friendship*
    - This allows us to use event sample augmentation
- **In addition, detailed optimizations/stress-testing studies need to be done**
  - We need to make sure RNTuple works reliably/efficient in all official ATLAS workflows
  - We also need to make sure that the data products and the jobs meet production limitations

12

RNTuple adoption for other experiments

CMS, LHCb and ALICE are also making substantial progress thanks to the tireless efforts, also in collaboration with the ROOT team.

We are here to support the transition to RNTuple of all experiments

Material from S. Mete, ACAT 2024, some highlighting added

Tooling and Some Code

Convert your existing TTree to RNTuple:

```
#include <ROOT/RNTupleImporter.hxx>
using ROOT::Experimental::RNTupleImporter;

auto importer = RNTupleImporter::Create(
   "Events",
   "myNanoAOD.ttree.root",
   "myNanoAOD.rntuple.root");

// Optional
importer->SetNTupleName("EventsNTuple");

auto writeOptions = importer->GetWriteOptions();
// Optional, default is zstd level 5
auto algo = RCompressionSetting::EAlgorithm::kLZMA;
writeOptions.SetCompression(algo, 7);
importer->SetWriteOptions(writeOptions);

importer->Import();
```

RNTupleImporter docs and tutorial

Get detailed storage information for your RNTuple:

```
#include <ROOT/RNTupleInspector.hxx>
using ROOT::Experimental::RNTupleInspector;

auto inspector = RNTupleInspector::Create(
   "EventsNTuple", "myNanoAOD.rntuple.root");

std::cout << "My NanoAOD is compressed using "
          << inspector->GetCompressionSettingsAsString()
          << std::endl;
inspector->PrintColumnTypeInfo();
```

```
my NanoAOD is compressed using lzma (level 7)
 column type | count | # elems    | compr. bytes | uncompr. bytes
-------------|-- ----|------------|--------------|----------------
SplitIndex64 | 5     | 267230990  | 84109056     | 2137847920
 SplitReal32 | 45    | 3856668029 | 11402474398  | 15426672116
  SplitInt32 | 15    | 1436663181 | 147427186    | 5746652724
```

RNTupleInspector docs

RNTuple Metrics

```
auto tree = file->Get<TTree>("tree");
TTreePerfStats *ps = new TTreePerfStats("ioperf", tree);
// …
ps->Print();
```

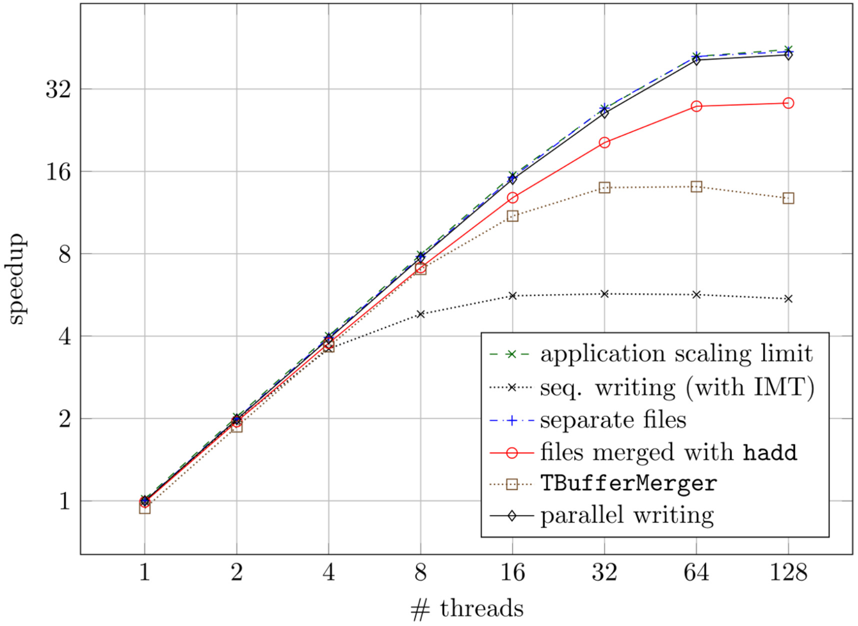
```
auto anchor = file->Get<RNTuple>("ntpl");
auto reader = RNTupleReader::Open(anchor);
reader->EnableMetrics();
// …
reader->PrintInfo(ENTupleInfo::kMetrics);
```

```
TreeCache = 30 MBytes
N leaves  = 26
ReadTotal = 749.412 MBytes
ReadUnZip = 1137.82 MBytes
ReadCalls = 524
ReadSize  = 1430.176 KBytes/read
Readahead = 256 KBytes
Readextra =  0.00 per cent
Real Time =   2.090 seconds
CPU  Time =   1.550 seconds
Disk Time =   0.724 seconds
Disk IO   = 1034.508 MBytes/s
ReadUZRT  = 544.310 MBytes/s
ReadUZCP  = 734.076 MBytes/s
ReadRT    = 358.504 MBytes/s
ReadCP    = 483.492 MBytes/s
```

```
RNTupleReader.RPageSourceFile.nReadV||number of vector read requests|21
RNTupleReader.RPageSourceFile.nRead||number of byte ranges read|834
RNTupleReader.RPageSourceFile.szReadPayload|B|volume read from storage (required)|731470154
RNTupleReader.RPageSourceFile.szReadOverhead|B|volume read from storage (overhead)|180996722
RNTupleReader.RPageSourceFile.szUnzip|B|volume after unzipping|1129407576
RNTupleReader.RPageSourceFile.nClusterLoaded||number of partial clusters preloaded from storage|21
RNTupleReader.RPageSourceFile.nPageLoaded||number of pages loaded from storage|17175
RNTupleReader.RPageSourceFile.nPagePopulated||number of populated pages|17175
RNTupleReader.RPageSourceFile.timeWallRead|ns|wall clock time spent reading|337259128
RNTupleReader.RPageSourceFile.timeWallUnzip|ns|wall clock time spent decompressing|527901157
RNTupleReader.RPageSourceFile.timeCpuRead|ns|CPU time spent reading|1355967000
RNTupleReader.RPageSourceFile.timeCpuUnzip|ns|CPU time spent decompressing|1373490000
RNTupleReader.RPageSourceFile.bwRead|MB/s|bandwidth compressed bytes read per second|2705.536486
RNTupleReader.RPageSourceFile.bwReadUnzip|MB/s|bandwidth uncompressed bytes read per second|3348.782827
RNTupleReader.RPageSourceFile.bwUnzip|MB/s|decompression bandwidth of uncompressed bytes per second|2139.430007
RNTupleReader.RPageSourceFile.rtReadEfficiency||ratio of payload over all bytes read|0.801640
RNTupleReader.RPageSourceFile.rtCompression||ratio of compressed bytes / uncompressed bytes|0.647658
```

Parallel Writing: Application Benchmark

- Skimming of the "Analysis Grand Challenge" (AGC) dataset
  - Drop unused columns
  - Filter events based on coarse cuts and entries in nested collections
- Compare multiple implementations of parallel writing
  - Using ROOT’s implicit multithreading (IMT)
  - Separate files + merging with hadd
  - TBufferMerger (in-memory merging)
  - Parallel RNTuple writing
- Parallel RNTuple writing as fast as independent writing into separate files
  - **Reaches 330 MB/s, below hardware limit: parallel writing is not the bottleneck!**

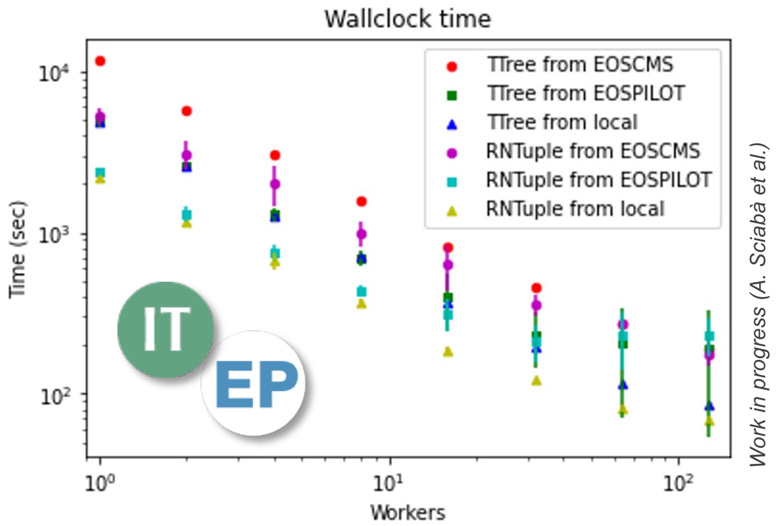


RNTuple makes leading edge R&D possible

RNTuple Validation and Limit Testing

Joint effort between IT-SD and EP-SFT on large-scale testing

- IT provided testbed: 80 nodes, 20PB storage, 100GbE
- Quick cycles of benchmarking and software improvements on ROOT and XRootD
  - First numbers with the "Analysis Grand Challenge" (a community standard benchmark) confirm the speed improvements of RNTuple when reading from EOS with high core counts (see plot)
  - workload variations
- Next steps during summer months
  - Tests with experiment-provided tasks
- Target for final results: CHEP; contribution submitted



RNTuple Interface Review

- Conducted by US High Energy Physics Center for Computational Excellence (HEP-CCE)
- Including Experiment experts form ATLAS, CMS & DUNE
- Final report expected in Q3/2024

Thanks!!

Planning and Schedule

RNTuple Type Support

<table>
<tr><th>Type Class</th><th>Types</th><th colspan="3">EDM Coverage</th><th>RNTuple Status</th></tr>
<tr><td>PoD</td><td><code>bool, (unsigned) char, std::byte, (u)int[8,16,32,64]_t, float, double</code></td><td rowspan="2">Flat n-tuple</td><td rowspan="4">Reduced AOD</td><td rowspan="9">Full AOD / ESD / RECO</td><td>Available</td></tr>
<tr><td>(Nested) vectors</td><td><code>std::vector, RVec, std::array,</code> C-style fixed-size arrays</td><td>Available</td></tr>
<tr><td>String</td><td><code>std::string</code></td><td></td><td>Available</td></tr>
<tr><td>User-defined classes</td><td>Non-cyclic classes with dictionaries</td><td></td><td>Available</td></tr>
<tr><td>User-defined enums</td><td>Scoped / unscoped enums with dictionaries</td><td></td><td></td><td>Available</td></tr>
<tr><td>User-defined collections</td><td>Non-associative collection proxy</td><td></td><td></td><td>Available</td></tr>
<tr><td><code>stdlib</code> types</td><td><code>std::atomic, std::pair, std::tuple, std::bitset, std::(unordered)set, std::(unordered)map</code></td><td></td><td></td><td>Available</td></tr>
<tr><td>Alternating types</td><td><code>std::variant, std::unique_ptr, std::optional</code></td><td></td><td></td><td>Available</td></tr>
<tr><td>Unsplit</td><td>All ROOT streamable objects (stored as byte array)</td><td></td><td></td><td>Available</td></tr>
<tr><td>Intra-event links</td><td><code>"&Electrons[7]"</code></td><td></td><td></td><td></td><td>post version 1.0</td></tr>
<tr><td rowspan="3">Low-precision floating points</td><td><code>Double32_t, Float16_t, (b)float16</code></td><td colspan="3" rowspan="3">Optimization benefitting all EDMs</td><td>Available</td></tr>
<tr><td>Custom precision and range</td><td>ongoing work / v1.0</td></tr>
<tr><td>Precision cascades</td><td>post version 1.0</td></tr>
</table>

Schedule Presented to the LHCC, Updated

~2018-19	~2019-20	~2021-22	~2022-23	~2023-24
Proof of concept	Prototype	First exploitation	Pre-production	Production

~2018-19 – Proof of concept

- ✅ Architecture
- ✅ Review on state-of-the-art
- ✅ First prototypes

~2019-20 – Prototype

- ✅ Adoption in ROOT::Experimental
- ✅ I/O scheduler for local and remote access
- ✅ Performance validation

~2021-22 – First exploitation

- ⛅ Object store support
  - ✅ DAOS (HPC)
  - ⛅ S3 (Cloud)
- ✅ RNTuple version 1 spec
- ✏ RNTupleLite
- ⛅ Schema evolution
- ✅ Disk-to-disk conversion
- ⛅ Virtual data sets for skims and selections
- ✅ First exposure to frameworks:
  - ✅ CMSSW nanoAOD output module
  - ✅ Prototyping by ATLAS, CMS, LHCb I/O experts

~2022-23 – Pre-production

- ✅ RDataFrame bulk processing
- ⛅ Debugging and inspection tools
- ✏ Metadata API
- ✅ Special use case support: e.g. backfill, in-memory adapters
- ✅ XRootD support
- ✅ Validation of feature coverage
- ⛅ Training experiments' core developers
- ⛅ Large-scale experiment benchmarks

~2023-24 – Production

- ⛅ PB scale tests
- ✏ Automatic optimization features
- ⛅ Low-precision floats
- ✏ ML Training: direct GPU transfer
- ✏ End-user training
- ⛅ Training and support for code and data migration

✅ = available
⛅ = under development
✏ = programme of work
— = in collaboration with users/experiments

Work items defined: Nov 2021
Development state: May 2024

Potential Future Directions

The RNTuple design opens the door to new functionality, which can be worked on after the initial production release.

For example:

- **Horizontal fast merge** ("persistified friends")
- **Zero-copy merge** on copy-on-write file systems
- Better **metadata support** (e.g. scale factors, varied columns)
- **Layout optimizer** that rewrites a file for strictly linear reads

Conclusions

Conclusions

- **TTree has been extremely successful in the last 25y**: it remains available & supported
- Building on top of the experience accumulated so far, **RNTuple is being developed to scale into the HL-LHC era (and beyond)**
  - A drop-in replacement for analysis, not for fwks. Many benefits to balance this downside
- **Golden opportunity for leading edge R&D**, e.g. parallel writing, object store support, low level hw optimisations
- **A solid plan ahead, agreed with experiments, monitored continuously, results and features are being delivered**
- v1.0 due at the **end of 2024: fix file format on disk, still evolving interfaces**
- **Adoption by experiments progressing quickly**, thanks to the effort of the core software teams, fully supported by the ROOT team

Backup

RNTuple Classes Design

Event iteration
Reading and writing in event loops and through `RDataFrame`
`RNTupleDataSource, RNTupleView, RNTupleReader/Writer`

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

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

Storage layer / byte ranges
`RPageStorage, RCluster, RNTupleDescriptor`

Approximate translation between TTree and RNTuple classes:

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

General coding guidelines

- Following C++ core guidelines
- Use of exceptions (RException)
- Conditionally thread-safe
- Compile-time type-safe interfaces, runtime type-safe interfaces and void* interfaces
- Shared pointers for values to be (de-)serialized
  - With option to pass raw pointes