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

Jakob Blomer, Philippe Canal, Axel Naumann, Javier Lopez-Gomez, Giovanna Lazzari Miotto

CHEP 2023, Norfolk, U.S.
May 8, 2023

Introduction

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

- Less disk and CPU usage
  - Significantly smaller files
  - Significantly better 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, GPU data transfer
- Native support for object stores in addition to local and remote ROOT files
- Binary format defined in a dedicated specification

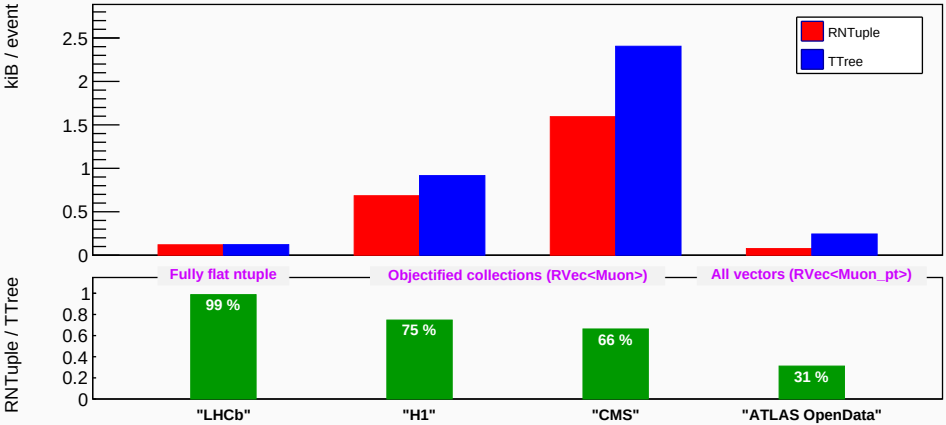
Introduction

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

- Less disk and CPU usage
  - Significantly smaller files
  - Significantly better 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, GPU data transfer
- Native support for object stores in addition to local and remote ROOT files
- Binary format defined in a dedicated ▸ specification

Performance: File Size

Size on disk, zstd compressed “final-stage” ntuples



(See backup slides for a description of the benchmarks)

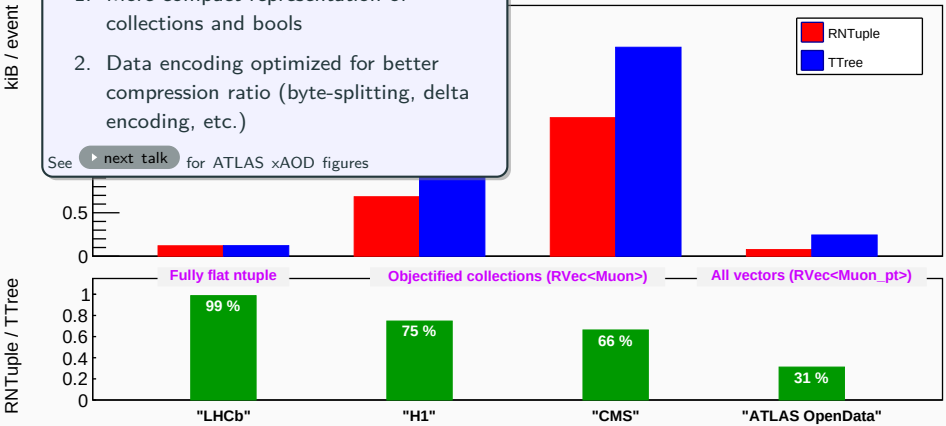
Performance: File Size

Size on disk of compressed "final-stage" ntuples

Main contributors to space savings:

1. More compact representation of collections and bools
2. Data encoding optimized for better compression ratio (byte-splitting, delta encoding, etc.)

See next talk for ATLAS xAOD figures

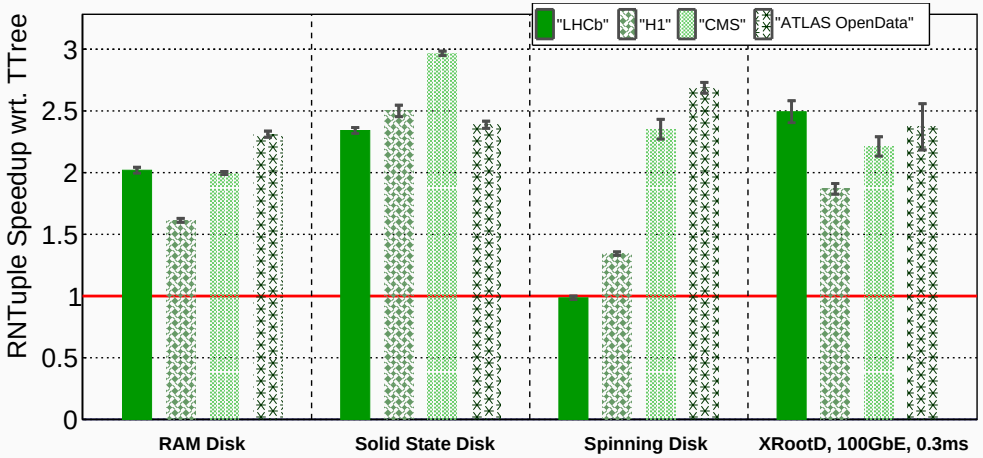


(See backup slides for a description of the benchmarks)

Performance: Time-to-Plot

Single-core analysis throughput using RDataFrame

Code



Performance: Time-to-Plot

Single-core analysis throughput using RDataFrame

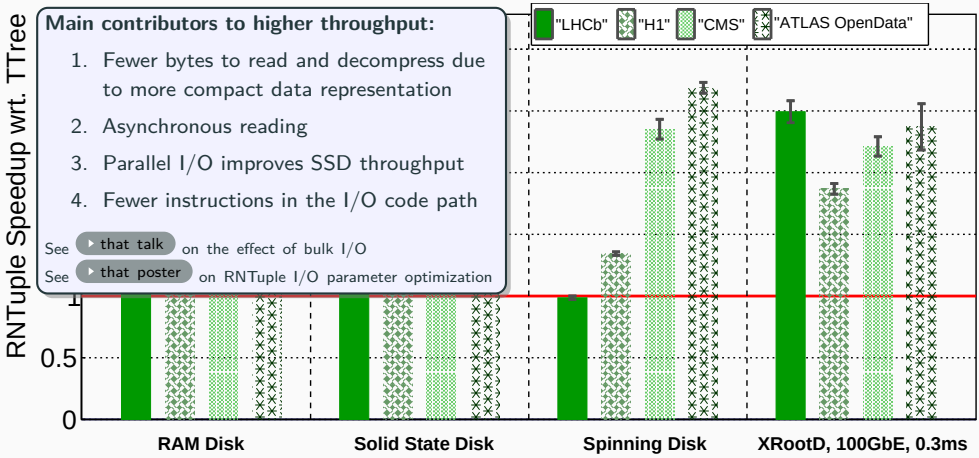
Code

Main contributors to higher throughput:

1. Fewer bytes to read and decompress due to more compact data representation
2. Asynchronous reading
3. Parallel I/O improves SSD throughput
4. Fewer instructions in the I/O code path

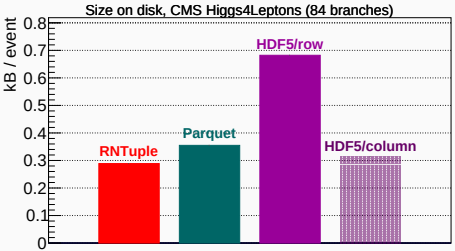
See that talk on the effect of bulk I/O

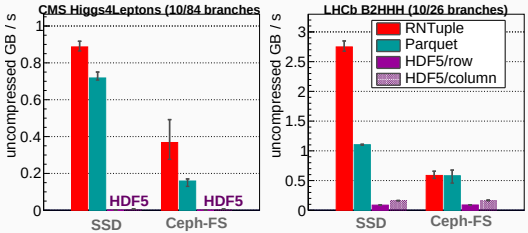
See that poster on RNTuple I/O parameter optimization



Performance Comparison with Parquet and HDF5

Code ACAT'21



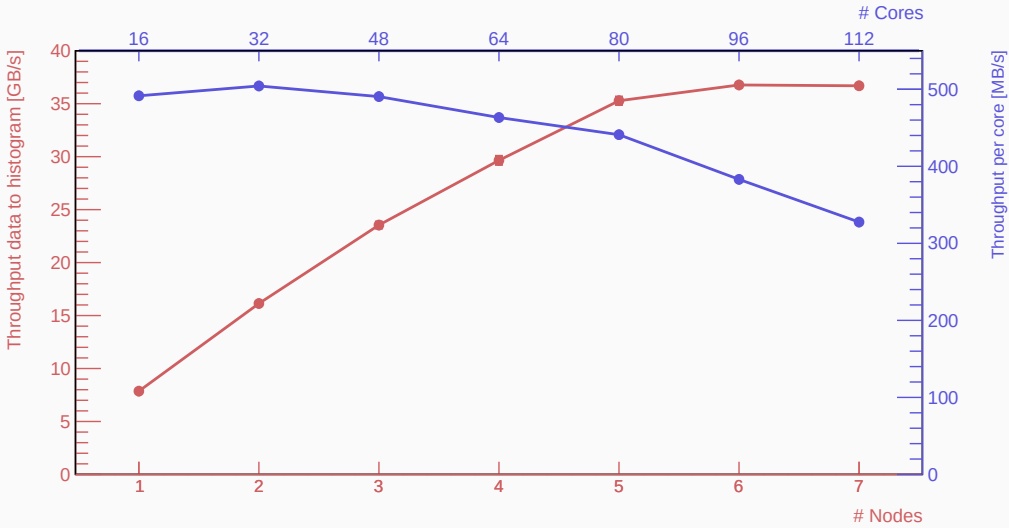


- Clear advantage of RNTuple over Parquet and HDF5, both in file size and throughput
- HDF5 results may vary depending on the effort put into adapting inherent tensor layout to columnar access

First Scale-Out Results

Distributed RDataFrame on 1 TB LHCb ntuples in a DAOS object store cluster, 100 Gbit/s network

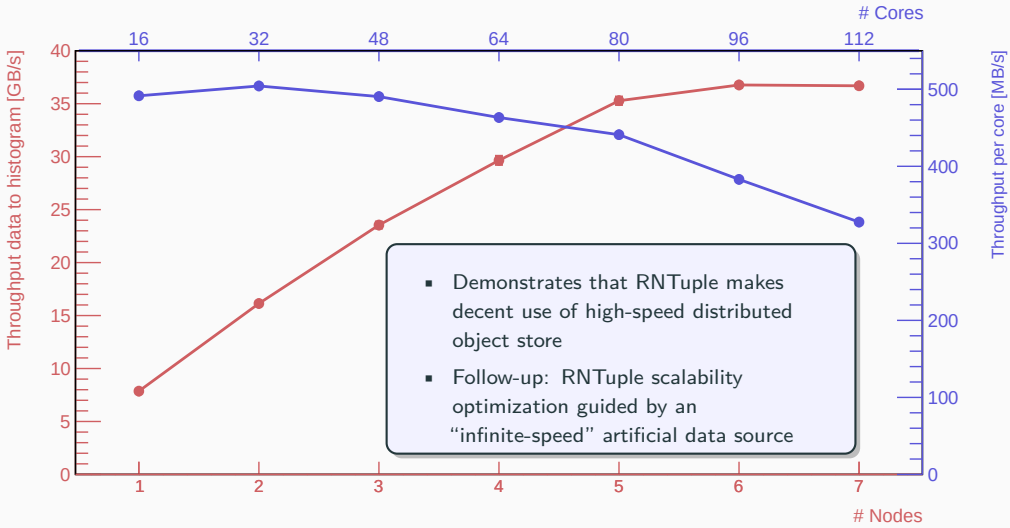
Paper



First Scale-Out Results

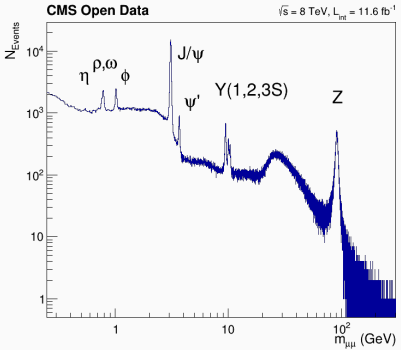
Distributed RDataFrame on 1 TB LHCb ntuples in a DAOS object store cluster, 100 Gbit/s network

Paper



How to try it out

- Take a ROOT package built with C++17 for access to the experimental classes
- Start with tutorials in `tutorials/v7/ntuple`, e. g. `ntpl004_dimuon.C`:



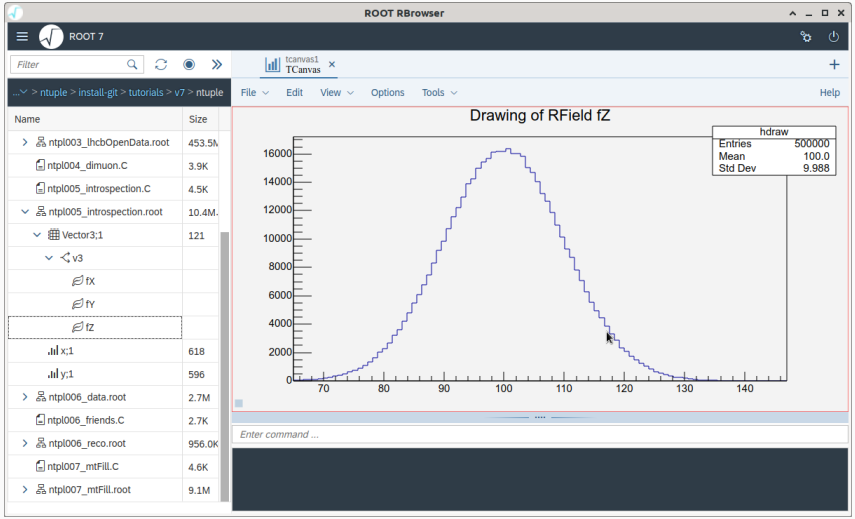
Format Transition and Compatibility

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

- For **RDataFrame** analysis code: no change required[1]
- Consistent tooling:
  - **RBrowser** support
  - **Disk-to-disk converter** TTree → RNTuple
  - `hadd` support under construction
- RNTuple data are stored in ROOT files and can be accessed the usual way locally and remotely through **XRootD** and **HTTP**
  - **New:** transparent access of RNTuple data in object stores (**DAOS**, **S3**) See that talk
- Native RNTuple API for writing and reading, targeting frameworks:
  new API following modern C++ core guidelines
- RNTuple adopts **TTree's I/O customization rules and schema evolution** system (under construction)
- `TTree::Draw` replacement under discussion

[1] Soon, RDataFrame will auto-detect input format TTree vs RNTuple.

RNTuple Data in the RBrowser



Early adoption

Framework integration

CMSSW RNTuple NanoAOD output module since 2021

Athena support for writing and reading ATLAS xAOD (PHYS & PHYSLITE) files since 2023

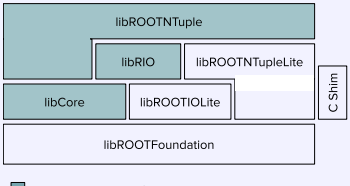
See also next talk

Continuous effort on EDM support and framework integration. RNTuple development & required feature set guided by early adoption; onboarding one-by-one to match development bandwidth.

3rd party

uproot Independent implementation based on the format specification

libRNTupleLite Low-level C API to support languages other than C++ and Python (**planned**)

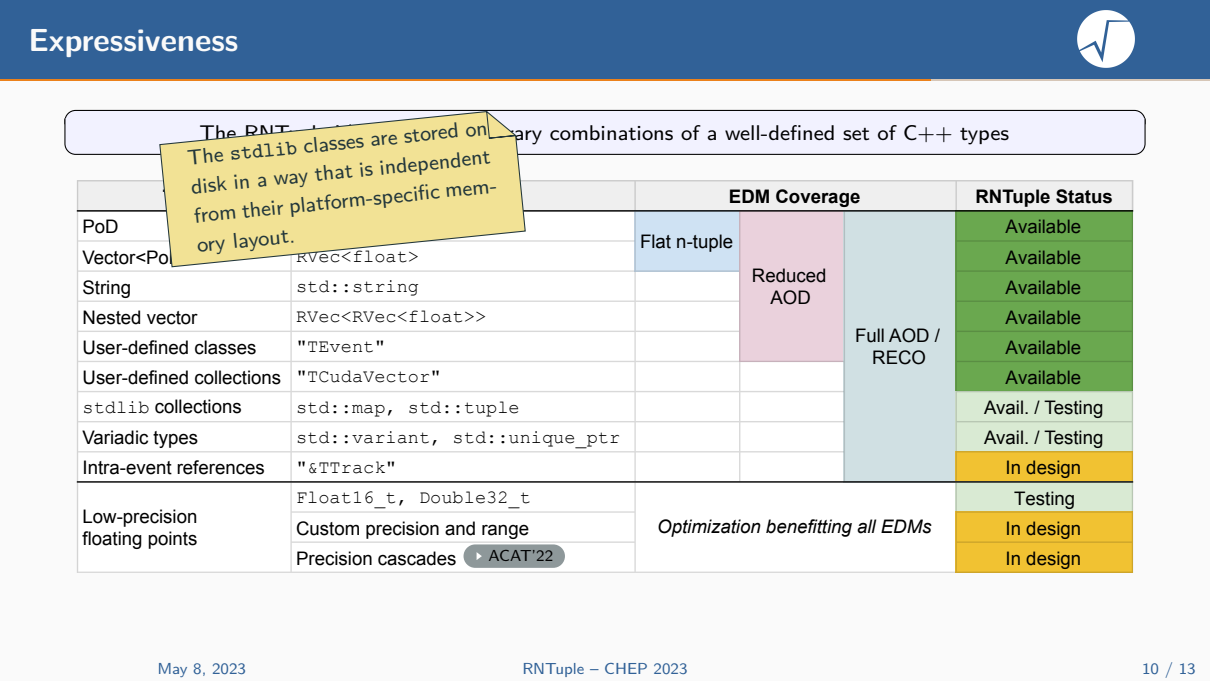


Expressiveness

The RNTuple I/O supports arbitrary combinations of a well-defined set of C++ types

Type	Examples	EDM Coverage			RNTuple Status
PoD	`bool, int, float`	Flat n-tuple	Reduced AOD	Full AOD / RECO	Available
Vector<PoD>	`RVec<float>`				Available
String	`std::string`				Available
Nested vector	`RVec<RVec<float>>`				Available
User-defined classes	`"TEvent"`				Available
User-defined collections	`"TCudaVector"`				Available
`stdlib` collections	`std::map, std::tuple`				Avail. / Testing
Variadic types	`std::variant, std::unique_ptr`				Avail. / Testing
Intra-event references	`"&TTrack"`				In design
Low-precision floating points	`Float16_t, Double32_t`	*Optimization benefitting all EDMs*			Testing
	Custom precision and range				In design
	Precision cascades ACAT'22				In design

Expressiveness



The RNT… …ary combinations of a well-defined set of C++ types

The stdlib classes are stored on disk in a way that is independent from their platform-specific memory layout.

		EDM Coverage			RNTuple Status
PoD		Flat n-tuple	Reduced AOD	Full AOD / RECO	Available
Vector<Po…	`RVec<float>`				Available
String	`std::string`				Available
Nested vector	`RVec<RVec<float>>`				Available
User-defined classes	`"TEvent"`				Available
User-defined collections	`"TCudaVector"`				Available
`stdlib` collections	`std::map, std::tuple`				Avail. / Testing
Variadic types	`std::variant, std::unique_ptr`				Avail. / Testing
Intra-event references	`"&TTrack"`				In design
Low-precision floating points	`Float16_t, Double32_t`	*Optimization benefitting all EDMs*			Testing
	Custom precision and range				In design
	Precision cascades ▸ ACAT'22				In design

Expressiveness

The RNT… …ary combinations of a well-defined set of C++ types

The stdlib classes are stored on disk in a way that is independent from their platform-specific memory layout.

RNTuple supports the most common and performance-critical stdlib types, such as std::vector, natively (without dictionaries).

		EDM Coverage			RNTuple Status
PoD		Flat n-tuple	Reduced AOD	Full AOD / RECO	Available
Vector<Po…	RVec<float…				Available
String					Available
Nested ve…					Available
User-defin…					Available
User-defin…					Available
stdlib co…	…, std::tuple				Avail. / Testing
Variadic types	std::variant, std::unique_ptr				Avail. / Testing
Intra-event references	"&TTrack"				In design
Low-precision floating points	Float16_t, Double32_t	*Optimization benefitting all EDMs*			Testing
	Custom precision and range				In design
	Precision cascades ▸ ACAT'22				In design

Expressiveness

The RNT… …ary combinations of a well-defined set of C++ types

The stdlib classes are stored on disk in a way that is independent from their platform-specific memory layout.

RNTuple supports the most common and performance-critical stdlib types, such as std::vector, natively (without dictionaries).

RNTuple does not support polymorphism.

		EDM Coverage			RNTuple Status
PoD		Flat n-tuple	Reduced AOD	Full AOD / RECO	Available
Vector<Po…	RVec<float…				Available
String					Available
Nested ve…					Available
User-defin…					Available
User-defin…					Available
stdlib co…	…std::tuple				Avail. / Testing
Variadic types	std… …unique_ptr				Avail. / Testing
Intra-ever…					In design
Low-precis… floating points	…t	*Optimization benefitting all EDMs*			Testing
	Custom precision and range				In design
	Precision cascades ▸ ACAT'22				In design

Writing and Deriving Data

Entry-by-entry writing

- Available, including multi-threaded writing
- Includes **"late model extensions"** to accommodate for frameworks' on-demand schema definition
- Planned: RNTuple output from `RDataFrame::Snapshot`
- R&D: **reducing contention** of highly parallel writes

Reshaping data: dataset derivation without decompressing / deserialization

- Fast merging of files, merging of clusters, discarding columns (fast "`CloneTree`")
- **Under construction**

Data combinatorics: virtual data sets

- **Friends** (available), **chains** (under construction)
- R&D program in approval on more advanced use cases, such as stored filters, indexed joins, and provenance meta-data; this is considered a potential extension after the first production release

Demonstrator: Zero-Copy Merge

Code

RNTuple proof-of-concept exploitation of modern file systems' block sharing support.

```
[root@phsft-cvm01 test7]# xfs_bmap -vp ntpl1.root
ntpl1.root:
 EXT: FILE-OFFSET       BLOCK-RANGE          AG AG-OFFSET         TOTAL FLAGS
   0: [0..7]:           105009056..105009063  2 (151456..151463)      8 000000
   1: [8..300007]:      105009064..105309063  2 (151464..451463) 300000 100000
   2: [300008..300095]: 105309064..105309151  2 (451464..451551)     88 000000
[root@phsft-cvm01 test7]# xfs_bmap -vp ntpl2.root
ntpl2.root:
 EXT: FILE-OFFSET       BLOCK-RANGE          AG AG-OFFSET         TOTAL FLAGS
   0: [0..7]:           105309152..105309159  2 (451552..451559)      8 000000
   1: [8..480007]:      105309160..105789159  2 (451560..931559) 480000 100000
   2: [480008..480135]: 105789160..105789287  2 (931560..931687)    128 000000
[root@phsft-cvm01 test7]# xfs_bmap -vp ntplmerged.root
ntplmerged.root:
 EXT: FILE-OFFSET       BLOCK-RANGE          AG AG-OFFSET             TOTAL FLAGS
   0: [0..7]:           157286488..157286495  3 (88..95)                  8 000000
   1: [8..300007]:      105009064..105309063  2 (151464..451463)     300000 100000
   2: [300008..300087]: 171841608..171841687  3 (14555208..14555287)     80 000000
   3: [300088..780087]: 105309160..105789159  2 (451560..931559)     480000 100000
   4: [780088..780215]: 171841688..171841815  3 (14555288..14555415)    128 000000
```

RNTuple 1

RNTuple 2

Merged RNTuple

Summary & Outlook

ROOT RNTuple is a leap in data throughput and storage efficiency

- Significantly smaller files and faster reads compared to TTree
- Efficient use of modern devices and storage systems such as SSDs, object stores, accelerators
- Work in progress with first successful integration efforts:
  CMS & ATLAS frameworks, RDataFrame, RBrowser, XRootD, TTree data importer

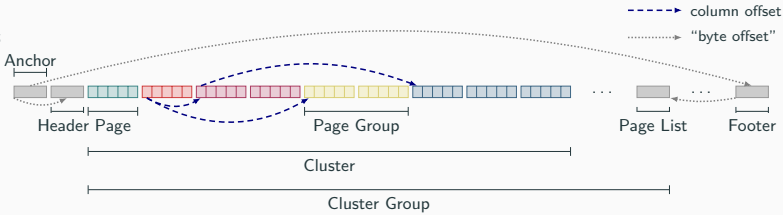
Roadmap to production use

- Stable binary format by the end of 2024
  - Backwards compatibility guarantee as of this point
  - Timeframe for a minimum viable product
- For HL-LHC, we expect RNTuple to cover the TTree use cases
  - We expect LHC Run 1–3 data remain in TTree format and new data being written in RNTuple format
- Next milestones:
  - Validation: RDataFrame version of the Analysis Grand Challenge with RNTuple data (see ▸ that talk)
  - Scale-out tests on big storage sites
  - Onboarding of full AOD/RECO formats

Backup Slides

Breakdown of the RNTuple On-Disk Format

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



Cluster

- Block of consecutive complete events
- Defaults to 50 MB compressed

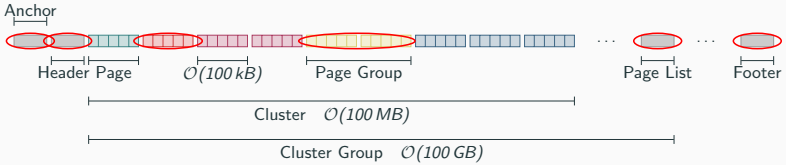
Format specification

Page

- Unit of (de-)compression
- Defaults to 64 kB uncompressed
- Not necessarily aligned on event boundary

RNTuple Read Pattern for Analysis Tasks

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



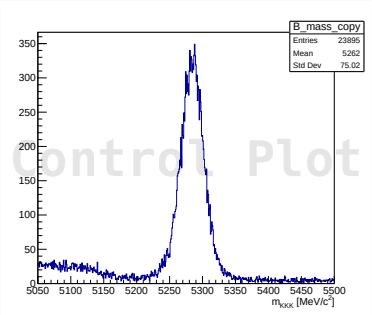
1. File open: read anchor, header, footer (once)
2. Read page list (one per cluster group)
3. Background thread: read-ahead page groups for the next k clusters in vector reads, close-by byte ranges get coalesced

Benchmark Descriptions I/II

Data Code

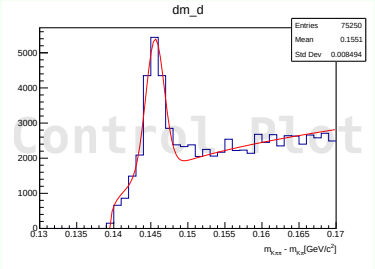
LHCb run 1 open data B2HHH

- Dense reading ($> 75\,\%$): 18/26 branches
- Fully flat data model
- 8.5 million events
- 24 k selected events



H1 micro dst [$\times 10$]

- Medium dense reading ($\sim 10\,\%$): 16/152 branches
- Event substructure: vector of jets etc.
- 2.8 million events
- 75 k selected events

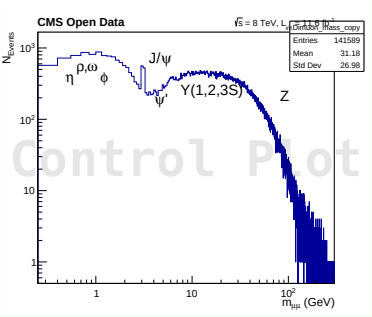


Benchmark Descriptions II/II

Data | Code

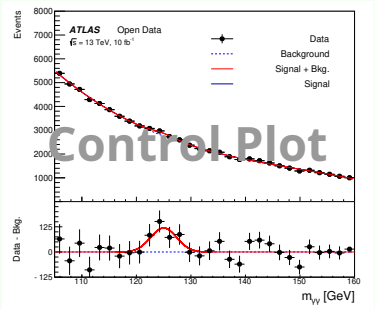
CMS nanoAOD June 2019

- Sparse reading ($< 1\,\%$): 6/1479 branches
- Event substructure: vector of jets etc.
- 1.6 million events
- 141 k selected events



ATLAS OpenData

- Medium dense reading ($\sim 15\,\%$): 13/81 branches
- Only vectors: vector or muon pt, eta, etc.
- 7.8 million events
- 76 k selected events



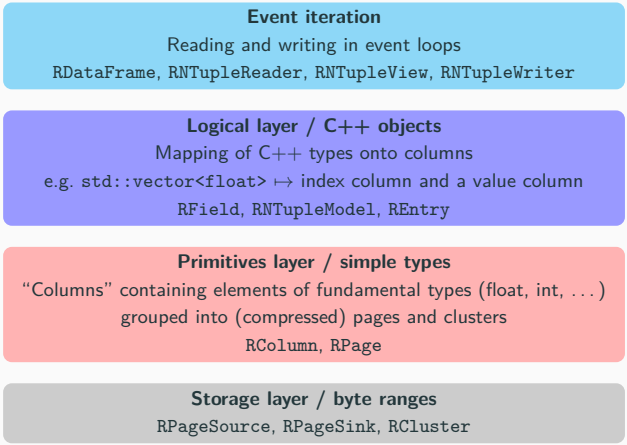
Benchmark Hardware and Software

CPU	AMD EPYC 7702P
Memory	DDR4 RDIMM 3200 MHz
SSD (flash)	SAMSUNG MZWLJ3T8HBLS-00007
HDD (spinning)	TOSHIBA MG07ACA14TE SATA 7200 RPM
Network	100 GbE

XRootD benchmarks used the projects.cern.ch EOS instance (same datacenter).

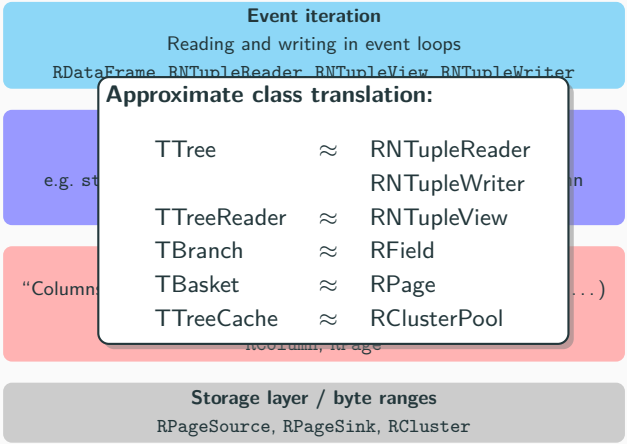
Library	**Version**
ROOT	▸ github tag
Benchmarks	▸ github tag
Linux	AlmaLinux 9.1 with Linux kernel 6.3 from ELrepo (uring enabled)

RNTuple Class Layering



- Storage access
  - File backend: local or remote using new `RRawFile`. Remote file access through Davix and XRootD
  - Object store: stores page groups directly in objects, implementation for Intel DAOS, S3 upcoming
  - Virtual: "friend" and "chain", buffered writes
- Utility classes: `RNTupleImporter`, `RNTupleInspector`, ...

RNTuple Class Layering



- Storage access
  - File backend: local or remote using new `RRawFile`. Remote file access through Davix and XRootD
  - Object store: stores page groups directly in objects, implementation for Intel DAOS, S3 upcoming
  - Virtual: "friend" and "chain", buffered writes
- Utility classes: `RNTupleImporter`, `RNTupleInspector`, ...

RNTuple Compile-Time Type-Safe API: Write Example

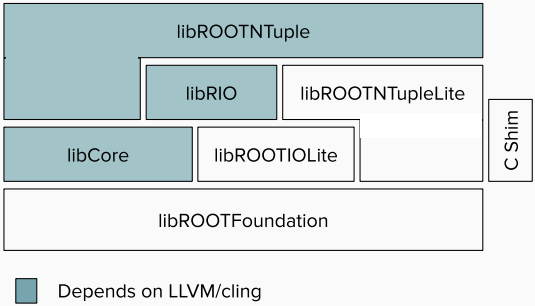
```
// Unique pointer to a new data schema
auto model = RNTupleModel::Create();
// Shared pointer to an std::vector<float>
auto fieldVpx = model->MakeField<std::vector<float>>("vpx");

auto ntplWriter = RNTupleWriter::Recreate(std::move(model), "Events", "data.root");

for (int i = 0; i < 1000; i++) {
   int npx = gRandom->Integer(15);
   fieldVpx->clear();
   for (int j = 0; j < npx; ++j)
      fieldVpx->emplace_back(gRandom->Gaus(0, 1));
   ntplWriter->Fill();
}

// Auto-save and close when ntplWriter goes out of scope
```

RNTuple Compile-Time Type-Safe API: Write Example

```cpp
// Unique pointer to a new data schema
auto model = RNTupleModel::Create();
// Shared pointer to an std::vector<float>
auto fieldVpx = model->MakeField<std::vector<float>>("vpx");

auto ntplWriter = RNTupleWriter::Recreate(std::move(model), "Events", "data.root");

for (int i = 0; i < 1000; i++) {
   int npx = gRandom->Integer(15);
   fieldVpx->clear();
   for (int j = 0; j < npx; ++j)
      fieldVpx->emplace_back(gRandom->Gaus(0, 1));
   ntplWriter->Fill();
}

// Auto-save and close when ntplWriter goes out of scope
```

For use in frameworks, a `void *` API exists as well, where types are passed as strings

libRNTupleLite



- The lite libraries are built just like any other ROOT libraries in ROOT proper (including modules, dictionaries etc)
- The lite libraries do not use any infrastructure from libCore but only from libROOTFoundation
- Contents of the lite libraries:
  - RIOLite: RRawFile without support for plugins, i. e. only local files
  - ROOTNTupleLite: RPageSource, RNTupleDescriptor (read-only)