RNTuple: ROOT's Event Data I/O for HL-LHC

Jakob Blomer

XRootD and FTS Workshop 2023, Jožef Stefan Institute, Ljubljana

What is RNTuple?

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

- Less disk and CPU usage for typical analysis files and tasks
  - 10 % to 20 % smaller files, ×2–5 better single-core performance
  - 10 GB/s per box and 500 MB/s per core sustained end-to-end throughput (compressed data to histograms)
- Systematic use of exceptions to prevent silent I/O errors
- Efficient support of modern hardware (async, parallel I/O, many-core friendly, GPU data transfer)
- Native support for object stores (besides local and remote ROOT files)

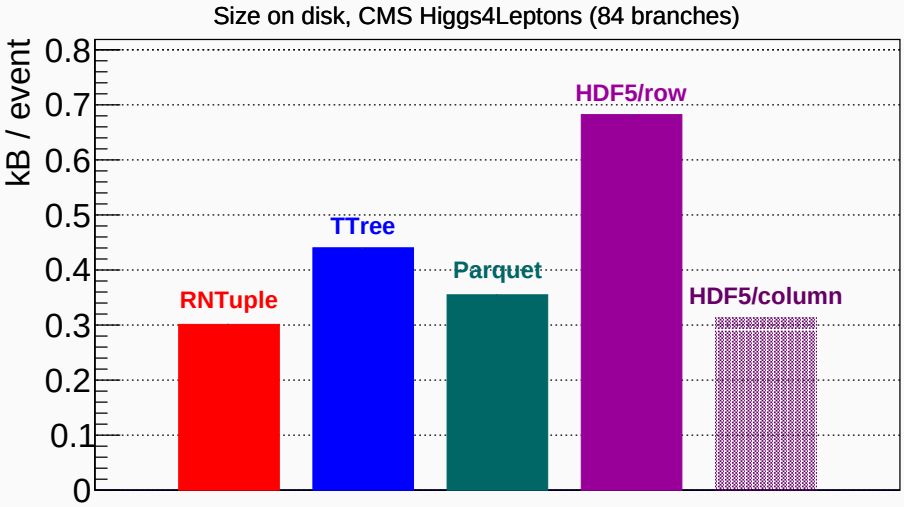
What is RNTuple?

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

- Less disk and CPU usage for typical analysis files and tasks
  - 10 % to 20 % smaller files, ×2–5 better single-core performance
  - 10 GB/s per box and 500 MB/s per core sustained end-to-end throughput (compressed data to histograms)
- Systematic use of exceptions to prevent silent I/O errors
- Efficient support of modern hardware (async, parallel I/O, many-core friendly, GPU data transfer)
- Native support for object stores (besides local and remote ROOT files)

Note: TTree remains available in ROOT but the focus of attention will gradually shift to RNTuple

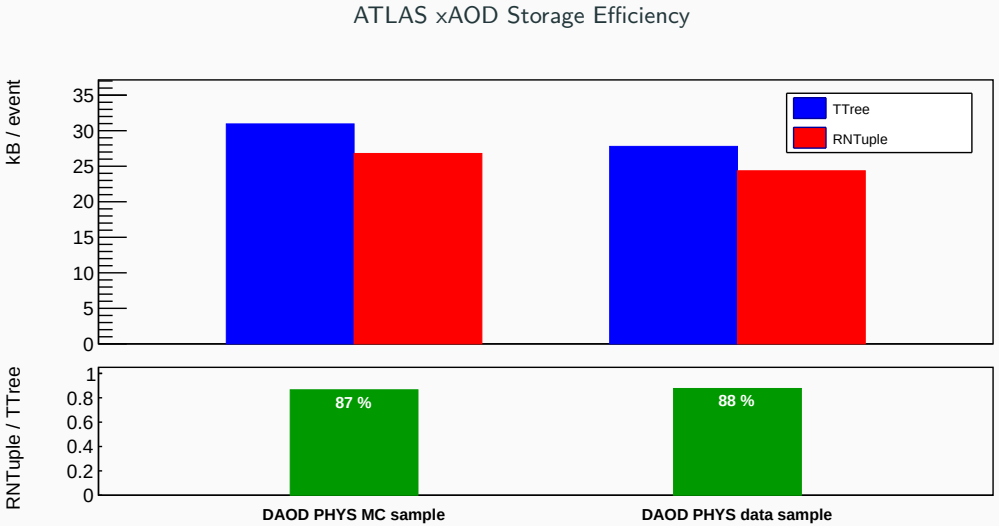
Performance plots (I)



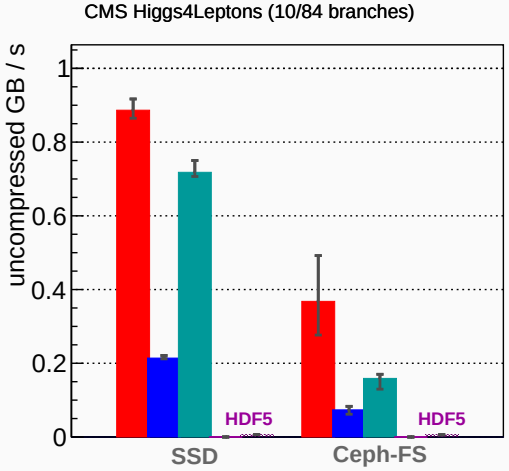
Code

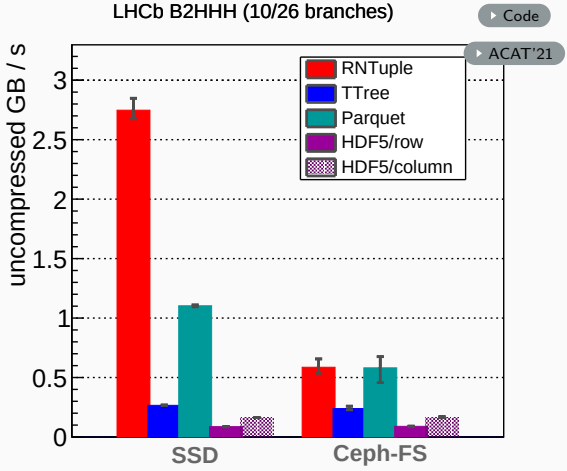
ACAT'21

Performance plots (II)

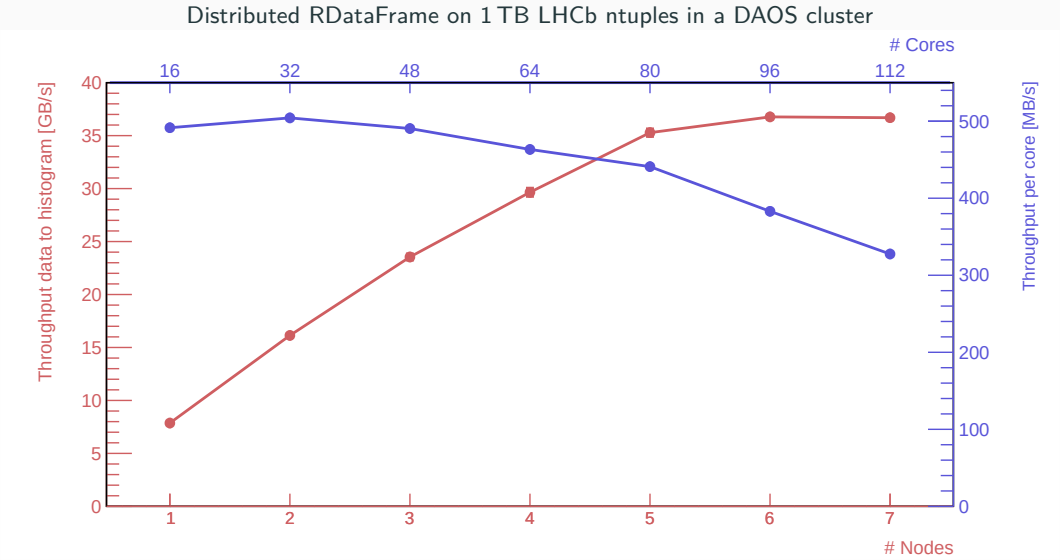


Performance plots (III)



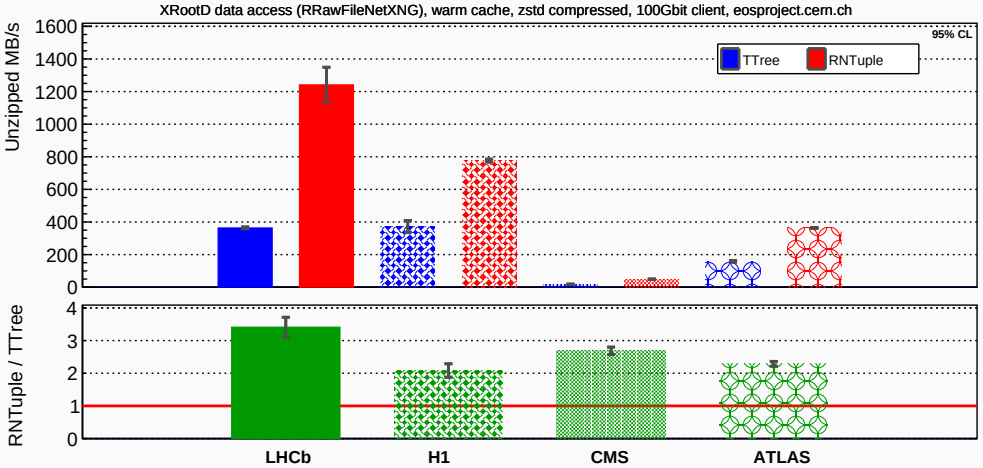


Performance plots (IV)



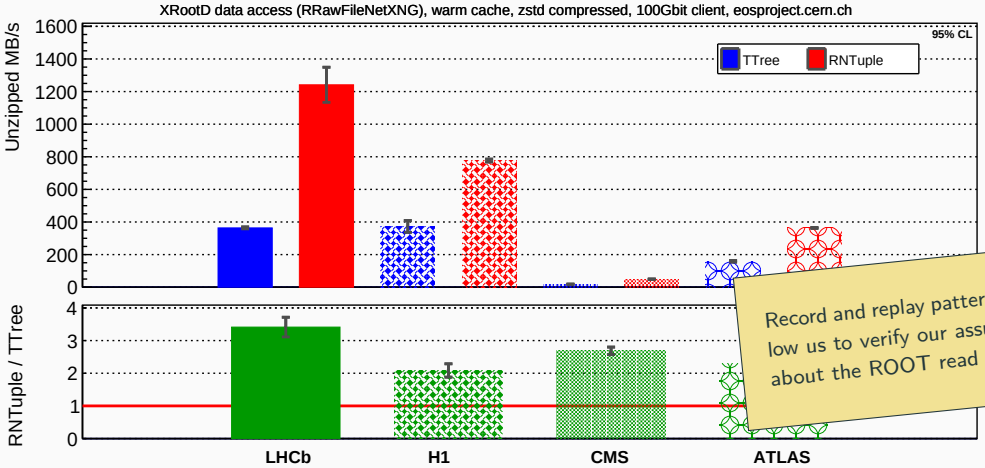
Performance plots (V)

Single-core end-to-end throughput (compressed data to histogram), eosproject.cern.ch



Performance plots (V)

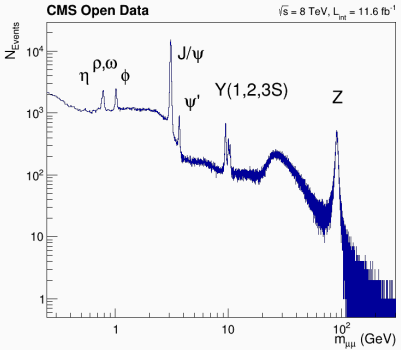
Single-core end-to-end throughput (compressed data to histogram), eosproject.cern.ch



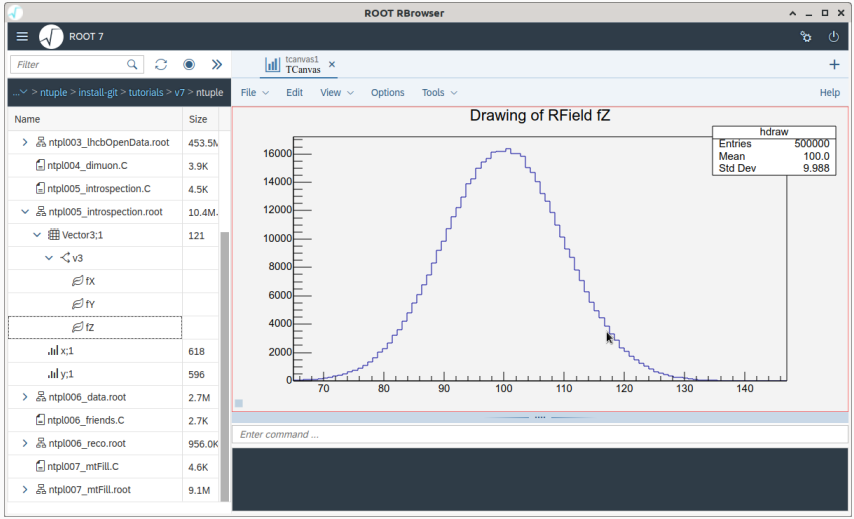
Record and replay pattern may allow us to verify our assumptions about the ROOT read pattern

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`:



RNTuple data in the RBrowser



How to try it out

- For RDF analyses: **one line**

```
auto rdf = ROOT::RDF::Experimental::FromRNTuple("Events", "data.root");
```

  Aiming for "zero line" change

- Tooling:
  - Disk-to-disk converter TTree → RNTuple: **available**
  - `hadd` support: **coming this year**
- "Native" RNTuple writing and reading (frameworks):
  new API following modern C++ core guidelines

RNTuple type system

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

- `float`, `double`
- `int`, `unsigned int`: 1, 2, and 4 bytes long
- `bool`, `std::bitset`
- `std::string`
- `std::vector`, `ROOT::RVec`
- `std::array`, C style arrays
- `std::pair`, `std::tuple`
- `std::variant`
- User-defined classes and collection proxies (w/ multiple inheritance but w/o polymorphism)
- Coming: `enums`, `std::map`, `std::unique_ptr`, `std::optional`, `Double32`, intra-event references

Supports ATLAS xAOD, CMS nanoAOD, LHCb final-stage ntuples. Support for more event data models coming.

RNTuple type system

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

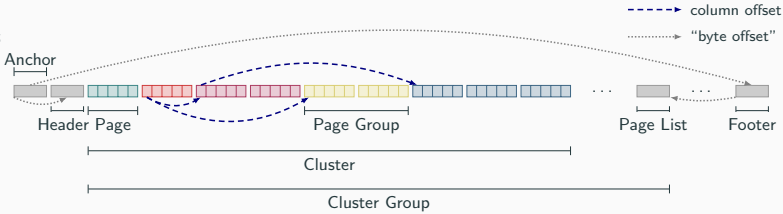
- `float`, `double`
- `int`, `unsigned int`: 1, 2, and 4 bytes long
- `bool`, `std::bitset`
- `std::string`
- `std::vector`, `ROOT::RVec`
- `std::array`, C style arrays
- `std::pair`, `std::tuple`
- `std::variant`
- User-defined classes and collection proxies (w/ multiple inheritance but w/o polymorphism)
- Coming: `enums`, `std::map`, `std::unique_ptr`, `std::optional`, `Double32`, intra-event references

For standard library classes, their "semantics" is stored on disk.

Supports ATLAS xAOD, CMS nanoAOD, LHCb final-stage ntuples. Support for more event data models coming.

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 Class Layering

Event iteration
Reading and writing in event loops
`RDataFrame`, `RNTupleReader`, `RNTupleView`, `RNTupleWriter`

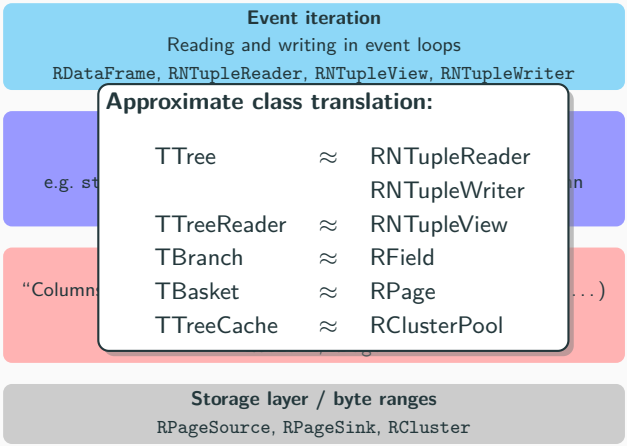
Logical layer / C++ objects
Mapping of C++ types onto columns
e.g. `std::vector<float>` ↦ index column and a value column
`RField`, `RNTupleModel`, `REntry`

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

Storage layer / byte ranges
`RPageSource`, `RPageSink`, `RCluster`

- Storage access
  - File backend: local or remote using new `RRawFile`. Remote file access through Davix and XRootD
  - Thanks to Michal for the `RRawFileNetXNG` implementation!
  - 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



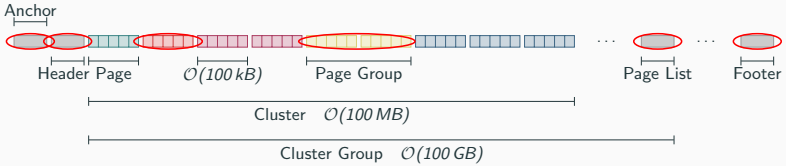
- Storage access
  - File backend: local or remote using new `RRawFile`. Remote file access through Davix and XRootD
  - Thanks to Michal for the `RRawFileNetXNG` implementation!
  - Object store: stores page groups directly in objects, implementation for Intel DAOS, S3 upcoming
  - Virtual: "friend"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

ROOT I/O: beyond serialization

Feature	Connection to XRootD
Architecture-independent encoding	no
C++ and Python support	no
Transparent compression	no
Lossy compression (low-precision floats)	no
Data checksums	no
Columnar access	**read pattern**
Horizontal data combinations (friends)	no
Vertical data combinations (chains)	**number of file opens**
Merging without decompressing data	**possibly: see later**
RDataFrame integration	no
RBrowser support	no
Remote access: HTTP and XRootD support	**RRawFile, TFile plugins**
Multi-threaded writes	no
Schema evolution	no
Late model extension	no
Support for application-defined metadata	no

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

Optimisation Opportunities?

- Don't coalesce for remote reads
- Multi-threaded behavior: share `XrdCl::File`
- Move footer to the front
- Information exchange: sparse/dense read, block sizes, ...

Zero-Copy Merge

Modern file systems support block sharing across files through `copy_file_range()`.
If possible through XRootD across nodes, this may have a huge impact on distributed workflows.

```
[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

ROOT RNTuple aims at a **leap in data throughput**

- Smaller files and significantly faster reads compared to TTree
- Efficient use of modern devices and storage systems
  (such as SSD, object stores, many cores, direct transfer to GPUs)
- Work in progress in `ROOT::Experimental`; the on-disk format is still subject to small changes

We are planning large-scale XRootD validation and optimisation in 2023 and 2024.
Perfect moment to optimise the HL-LHC event data format and the storage & transport layer in conjunction.

Backup Slides

RNTuple compile-time type-safe API write example

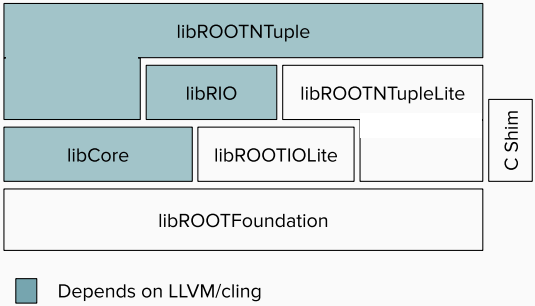
```
// Unique pointer to a new data schema
auto model = RNTupleModel::Create();
// Shared pointer to an std::vector<float>
auto fieldVpx = model->MakeField<std::vector<float>>("vpx");

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)