RNTuple – The Next-Generation TTree

Jakob Blomer, Philippe Canal, Javier Lopez Gomez
ROOT Workshop 2022, Fermilab

What is RNTuple?

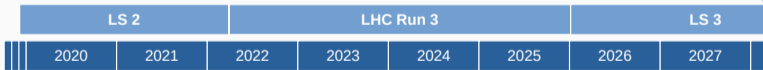Based on 25+ years of TTree experience, RNTuple is a redesigned I/O subsystem aiming at

- Less disk and CPU usage for the same data content
  - 25% smaller files, $\times$2–5 better single-core performance
  - 10 GB/s per box and 1 GB/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 (built for multi-threading and async I/O)
- Native support for object stores (see later)

LS 2		LHC Run 3				LS 3			Run 4 (HL-LHC)
2020	2021	2022	2023	2024	2025	2026	2027	2028	2029

RNTuple work in progress in ROOT::Experimental   **RNTuple goes production, adoption phase**

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

- Less disk and CPU usage for the same data content
  - 25% smaller files, $\times$2–5 better single-core performance
  - 10 GB/s per box and 1 GB/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 (built for multi-threading and async I/O)
- Native support for object stores (see later)

LS 2		LHC Run 3				LS 3	
2020	2021	2022	2023	2024	2025	2026	2027

RNTuple work in progress in ROOT::Experimental   **RNTuple goes production, adop**
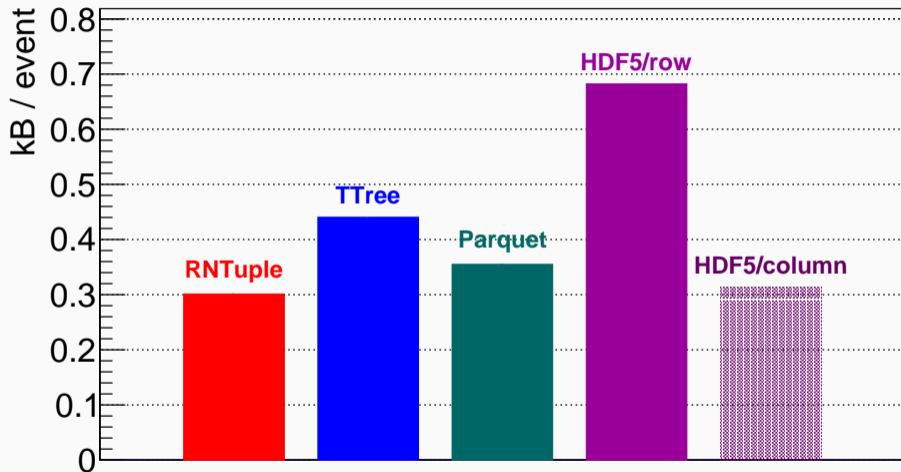
Note: TTree remains available in ROOT but the focus of attention will gradually shift to RNTuple

Size on disk, CMS Higgs4Leptons (84 branches)

CMS Higgs4Leptons (10/84 branches)
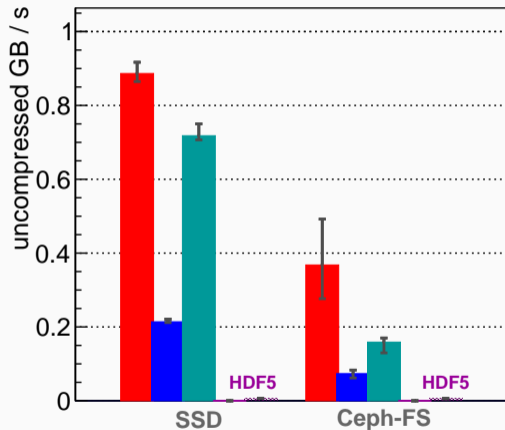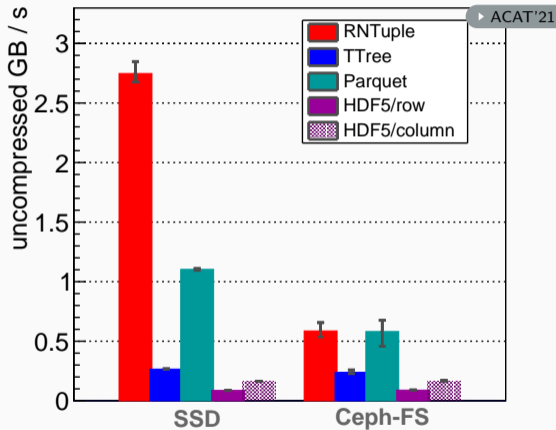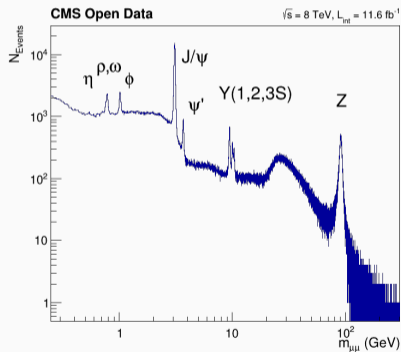
LHCb B2HHH (10/26 branches)

- 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 browser

Will I need to change my code?

- For RDF analyses: **one line**

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

- ROOT's tooling for ROOT files
  - RBrowser integration: **available**
  - `hadd` support: **coming this year**
  - Disk to disk converter TTree → RNTuple: **coming this year**
- Writing and RNTuple native reading: **new API following modern C++ core guidelines**

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
```

```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

- Native object store support
    - Intel DAOS HPC object store: **available**
    - S3 cloud storage: **coming**
- Zero-copy merging on modern file systems: **R&D**
- Direct data transfer SSD $\rightarrow$ GPU & GPU accelerated decompression: **R&D**

Issues with traditional storage stack. . .

- Designed for spinning disks (few IOPS): I/O coalescing, buffering, etc., became less relevant for modern devices → **overhead**

- **Limitations in scalability for parallel filesystems**

Issues with traditional storage stack...

- Designed for spinning disks (few IOPS): I/O coalescing, buffering, etc., became less relevant for modern devices → **overhead**

- **Limitations in scalability for parallel filesystems**

Hmmm... This situation can be improved!

What is DAOS?

- **Modern fault-tolerant object store** optimized for high bandwidth, low latency, and high IOPS. Foundation of the Intel exascale storage stack

- Optimal use of storage-class memory and NVMe SSDs
- I/O of Argonne's Aurora[1] supercomputer will be based on it

- Experience acquired supporting this in RNTuple can be reused for other object stores, e.g. Amazon S3

While DAOS has a compatibility layer, e.g. a FUSE filesystem, RNTuple provides a backend that delivers high performance.

---

[1] https://alcf.anl.gov/aurora

- Next-generation datacenters may not use a filesystem to store to-be-processed / processed data
- In HEP, object stores will not probably be the permanent way of storing data; instead, we see them as a temporary storage, e.g. for **high-throughput distributed analysis**

- Thus, quick population of object store with data is also important

DAOS pool   DAOS container   DAOS object
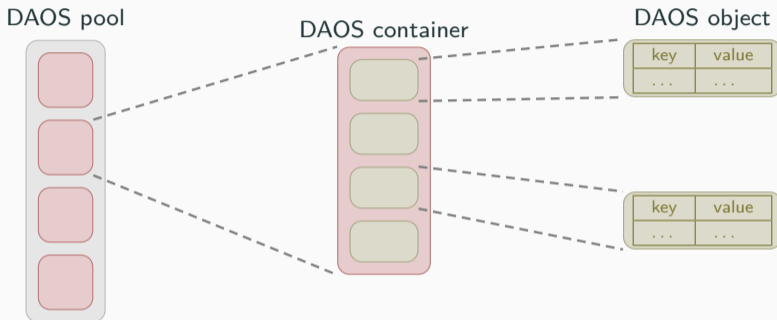
key	value
...	...

key	value
...	...

- **Object:** to put it short, a Key–Value store with locality
- **Object class:** determines redundancy, e.g. replication/erasure code
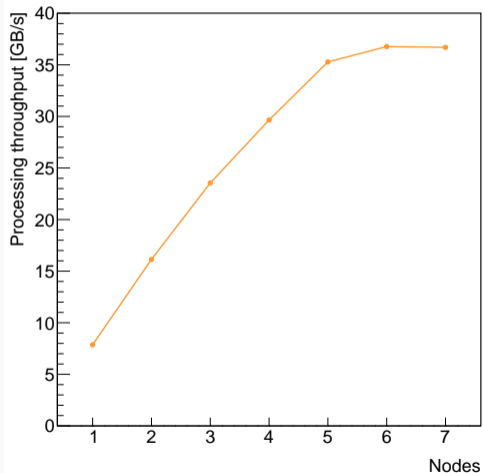
Only requires the replacement of the file path

```cpp
auto ntuple = RNTupleReader::Open("DecayTree",
                                  "./B2HHH~zstd.ntuple");
```

to a daos:// URI

```cpp
auto ntuple = RNTupleReader::Open("DecayTree",
                                  "daos://<POOL UUID>/<CONTAINER UUID>");
```

- 800 GB dataset based on LHCB opendata B2HHH

- Processed using distributed RDataFrame + RNTuple DAOS backend

- 70% of the nominal bandwidth (48 GB/s) of the cluster achieved

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

- Expect smaller files and significantly faster reads compared to TTree

- Modern and robust API

- Capable of making efficient use of modern devices and storage systems
  (such as SSD, object stores, many cores)

> RNTuple is work in progress in `ROOT::Experimental`.
> (The on-disk format is still subject to small changes!)
> We are happy to get your feedback!

Backup Slides

Breakdown of the RNTuple on-disk format

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


Header Page   Page List   Footer
                              Cluster
                         Cluster Group
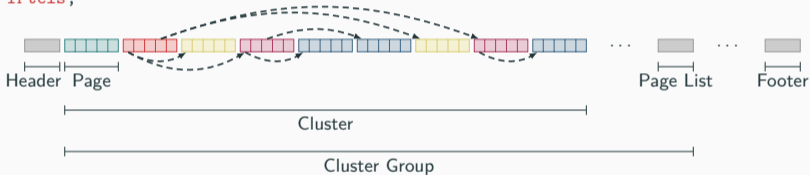
Cluster

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

Page

- Unit of (de-)compression and (un-)packing
- Defaults to 64 kB uncompressed

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>` $\mapsto$ 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, RPage`

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

- Storage access
  - Physical: ROOT file container, raw file, object store
  - Virtual: "friend" and "chain", buffered writes

- Serialization of simple types and STL collections built-in – can be read without libCore

RNTuple Class Layering

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

e.g. st...   ...n

"Column...   ...)
RColumn, RPage

Storage layer / byte ranges
RPageSource, RPageSink, RCluster

Approximate class translation:

TTree	$\approx$	RNTupleReader
		RNTupleWriter
TTreeReader	$\approx$	RNTupleView
TBranch	$\approx$	RField
TBasket	$\approx$	RPage
TTreeCache	$\approx$	RClusterPool

- Storage access
  - Physical: ROOT file container, raw file, object store
  - Virtual: "friend" and "chain", buffered writes
- Serialization of simple types and STL collections built-in – can be read without libCore

libRNTupleLite

libROOTNTuple

libRIO   libROOTNTupleLite

C Shim

libCore   libROOTIOLite

libROOTFoundation
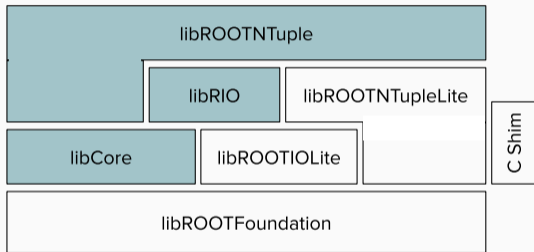
Depends on LLVM/cling

- 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)

RNTuple type system

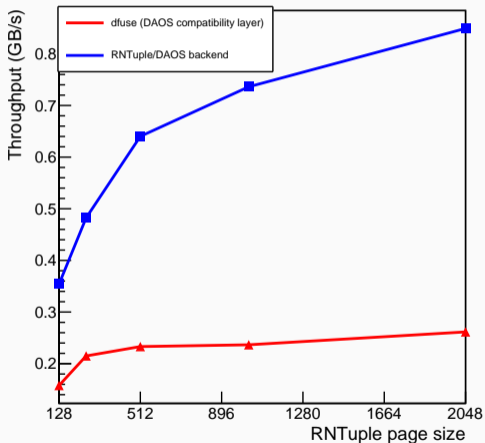> The RNTuple I/O supports arbitrary combinations of a well-defined set of C++ types

- `float`, `double`
- `int`, `unsigned int`: 1, 2, and 4 bytes long
- `std::string`
- `bool`
- `std::vector`, `ROOT::RVec`
- `std::array`
- `std::variant`
- Classes with dictionaries incl. (multiple) inheritance but w/o polymorphism
- Coming: enums, `std::pair`, `std::set`, intra-event references

I/O features: the essentials

Feature	Status
Architecture-independent encoding	available
C++ and Python support	available (w/o pythonizations)
Transparent compression	available
Fully checksummed	coming soon
Columnar access	available
Horizontal data combinations (friends)	available (aligned only)
Vertical data combinations (chains)	coming soon
Merging without uncompressing data	coming soon
RDataFrame integration	available
RBrowser support	available
Remote access: HTTP and XRootD support	available
Async reading, parallel decompression	available
Multi-threaded writes	available (only compression parallelized)
Schema evolution	coming soon
On-demand schema extension (backfilling)	coming soon
Support for application-defined metadata	coming soon

DAOS single-node throughput (HPE)

Write throughput, LHCB run 1 opendata B2HHH

- dfuse (DAOS compatibility layer)
- RNTuple/DAOS backend

Throughput (GB/s) vs RNTuple page size

Read throughput, LHCB run 1 opendata B2HHH

Throughput (GB/s) vs RNTuple page size