

# Future of ROOT I/O

**Florine Willemijn de Geus**<sup>1,2</sup>, Jakob Blomer<sup>1</sup>, Philippe Canal<sup>3</sup>,  
Jonas Hahnfeld<sup>1,4</sup>, Vincenzo Eduardo Padulano<sup>1</sup>, Giacomo Parolini<sup>1</sup>

<sup>1</sup>CERN <sup>2</sup>University of Twente <sup>3</sup>Fermilab <sup>4</sup>Goethe University Frankfurt

**ROOT Users Workshop 2025**

Valencia, Spain – 20-11-2025



## **RNTuple**: an excellent testbed for new I/O R&D!

→ Case in point, last presentation

The computing challenges of HL-LHC and beyond present opportunities to do more, both in terms of **performance** and **functionality**

This talk focuses on **ideas**, **designs** and **prototypes**

→ The items discussed here are not yet part of ROOT/RNTuple, but can become part of our future Program of Work. Your input is highly valued!



Handling ~~big~~ huge files

$O(10 \text{ GB})$

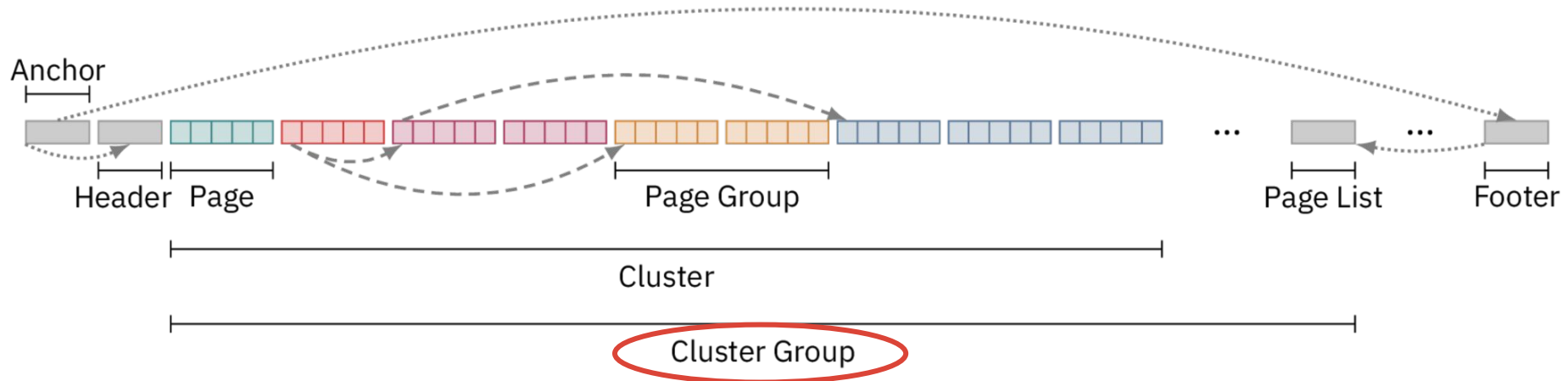
# Handling ~~big~~ huge files

$O(100 \text{ GB}) - O(1 \text{ TB})$



# Handling huge files in RNTuple

RNTuple – *by design* – is prepared for the future of HEP data storage



**Cluster groups** allow for loading metadata in segments, enabling iteration over huge files

N.B. exploitation of cluster groups is not yet implemented, but [architecture](#) and [specification](#) already anticipate it and will **not** break compatibility



With cluster groups, the [limits](#) for storing a data set in a **single file** are *virtually* endless:

Limit	Value	Reason / Comment
Volume	<b>1-10PB (theoretically more)</b>	Assuming 10k cluster groups of 10k clusters of 10-100MB each
Number of elements, entries	$2^{64}$	Using default (Split) Index64, otherwise $2^{32}$
Cluster & entry size	8TB (depends on pagination)	Assuming limit of 4 billion pages of 4kB each
Page size	2 billion elements, 256MB – 24GB	#elements × element size
Element size	8kB	16 bits for number of bits per element
Number of column types	64k	16 bits for column type
Envelope size	$2^{48}$ B (~280TB)	Envelope header encoding
Field / type version	4 billion	Field metadata encoding
Number of fields, columns	4 billion (expected: << 10 million)	32 bit column / field IDs, list frame limit
Number of clusters per group	4 billion (expected: < 10k)	List frame limits, cluster group summary encoding
Number of pages per cluster per column	4 billion	List frame limits

Not immediately urgent for HL-LHC, but indications for relevance are here already → e.g. [DUNE supernova events](#)



# AoS and SoA



We (usually) reason about our data as arrays of structs...

```
struct Particle {  
    float fPt;  
    float fEnergy;  
};  
  
using ParticleAoS = ROOT::RVec<Particle>;
```

VS.

...but sometimes representing them as structs of arrays gives better performance

```
struct ParticleSoA {  
    using Record_t = Particle;  
  
    ROOT::RVec<float> fPt;  
    ROOT::RVec<float> fEnergy;  
};
```

RNTuple's columnar layout already stores data like this on disk, but the core API is largely object-wise

SoA reading is already possible **today** [with RNTuple views](#), but comes with caveats:

- Not the most convenient ergonomics
- Not fully composable
- No solution for writing



We provide two ways of storing SoA collections in RNTuple:

- 1. As a “real SoA type”**
  - Must correspond to its record type
  - Marked as such in the ROOT dictionaries
- 2. As an `RNTupleSoA<RecordT>` wrapper type**
  - No explicit SoA type required
  - Provides runtime access to set the address of SoA columns

Both options enable reading from on-disk `Vec<RecordT>` into `SoA<T>`

Both options support class hierarchies and **automatic\*** schema evolution

(\*manual schema evolution will break all SoA performance benefits)



## We want to:

1. Facilitate reading and writing to/from SoA memory blocks
2. Be safe by default through ROOT-managed memory
3. Provide the flexibility to use user-defined buffers
4. Preserve RNTuple's type system composability
  - It should be possible to use SoA collections as subfields of "regular" fields



## We don't plan to:

1. Support complex shapes
  - These can be composed with structs
  - SoA column sizes are implicitly given in RNTuple, no extra scalars needed
2. Provide an AoS ↔ SoA adaptation layer
  - RNTuple will only take care of efficient data moving

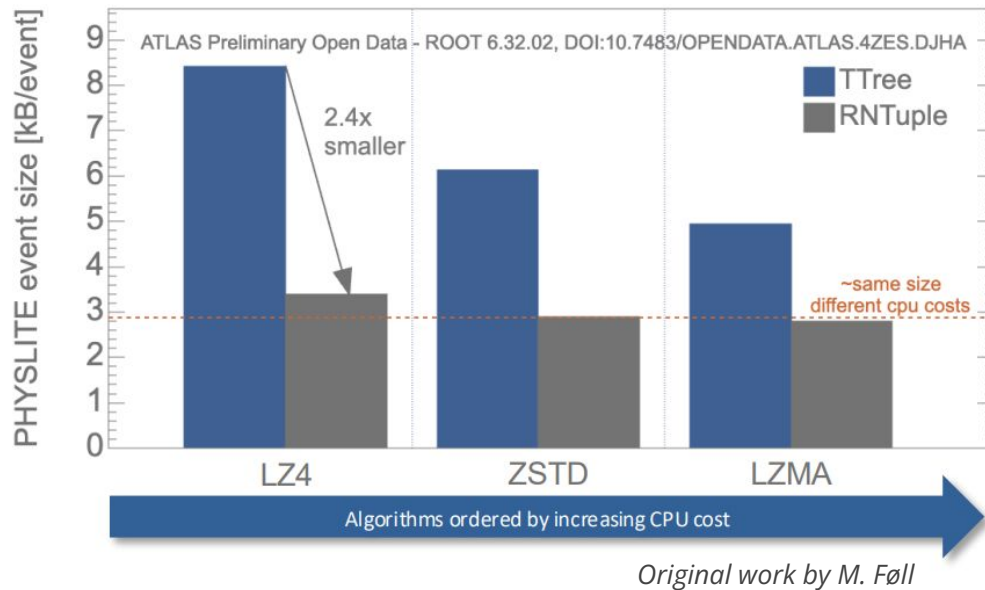
```
struct Event {  
    RNTupleSoA<Track> fTracks;  
    RNTupleSoA<Jet> fJets;  
};
```



# Compression and encoding



With RNTuple, the relative difference between compression algorithms fades:



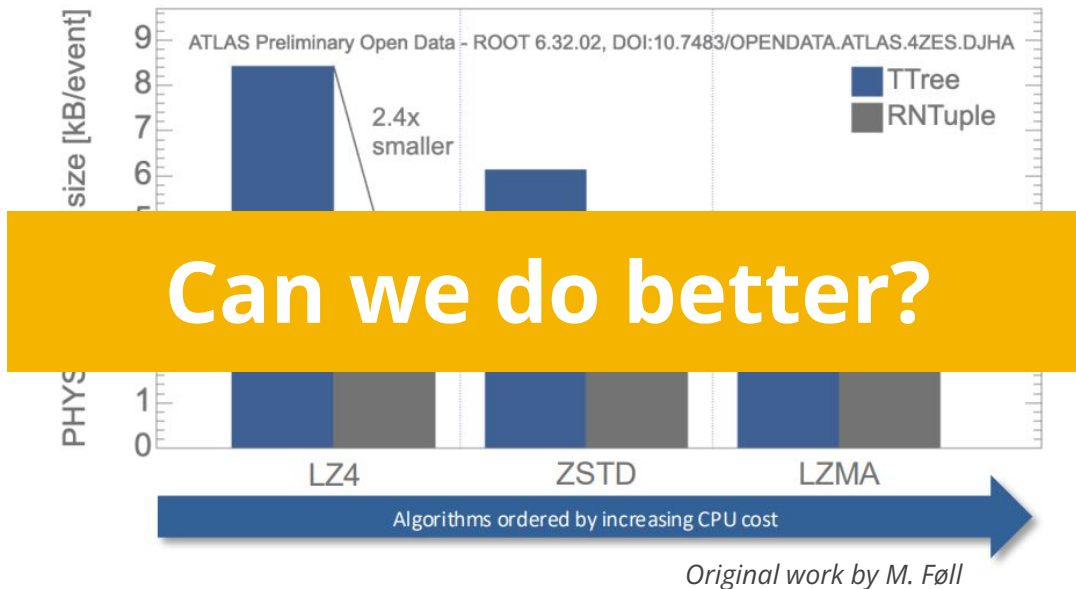
Type-dependent encodings

This is largely thanks to **lightweight compression schemes**





With RNTuple, the relative difference between compression algorithms fades:



Type-dependent encodings

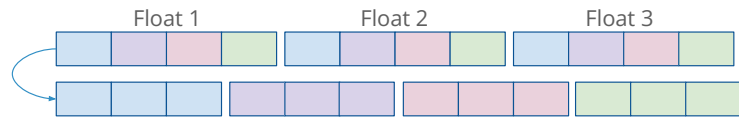
This is largely thanks to **lightweight compression schemes**



# Compression: new encodings

## Encodings currently in RNTuple:

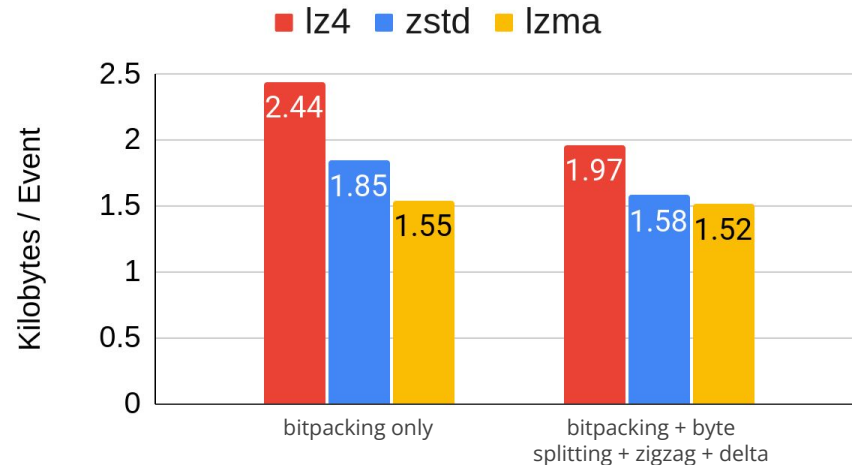
- Bitpacking (bools)
- Byte splitting (floats, ints, offsets)
- Zigzag (signed ints)
- Delta (offsets)



There are more encodings: run-length, dictionary, frame-of-reference...

**Would these be able to improve compression for our data?**

Effect of compression and pre-filters  
(CMS NanoAOD ttjet\_13tev\_2019)



Smart encoding choices can significantly improve storage efficiency, but have to be decided a-priori and manually

**Can we do this automatically?**



# Compression: better algorithms

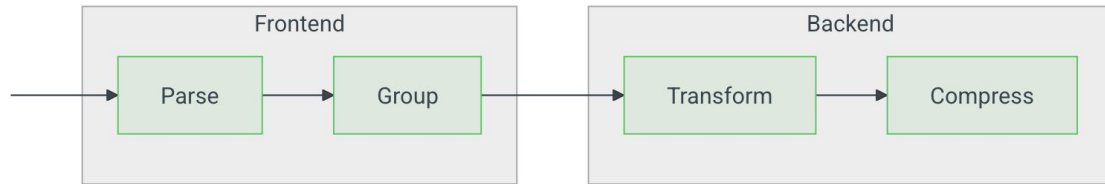
## OpenZL

From the developers of **Zstandard**

Not a new algorithm, but a new approach to compression

Leveraging the underlying data structure through a **graph model**

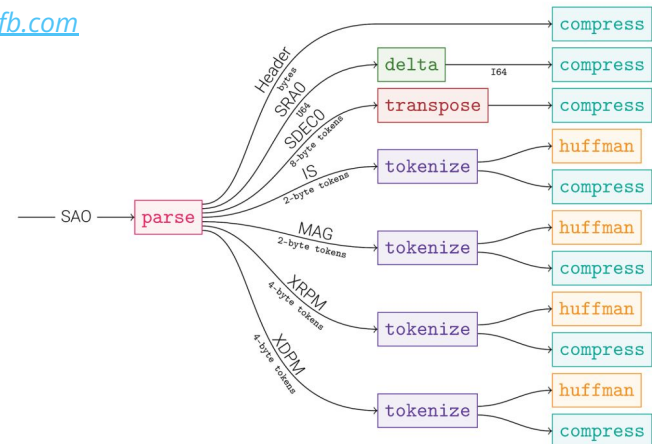
With our existing data layouts, how much would we still gain from the OpenZL approach?



Compression results for saio (part of *Silesia Corpus*; text data)

Compressor	zstd -3	lzma -9	OpenZL
Compressed Size	5,531,935 B	4,414,351 B	3,516,649 B
Compression Ratio	x1.31	x1.64	x2.06
Compression Speed	220 MB/s	3.5 MB/s	340 MB/s
Decompression Speed	850 MB/s	45 MB/s	1200 MB/s

From [engineering.fb.com](http://engineering.fb.com)



From [openzl.org](http://openzl.org)



# Compression: better algorithms

## BZip3

New lossless compression algorithm, based on the **Burrows-Wheeler Transform**

Potential successor and/or alternative to LZMA

Official benchmarks on (mostly) **text data**

**Silesia Corpus (211.9MB)**  
Runtime (mm:ss), result size (MB), memory usage (MB)

bzip3	ZStd -19	lz4 -9	lzma -9
00:17	01:23	00:07	01:17
47.2 MB	53.0 MB	78.0 MB	48.7 MB
98M	237M	9M	675M

*compression speed*

From [github.com/iczelia/bzip3](https://github.com/iczelia/bzip3)

*Preliminary, but promising results for **HEP data***

Sample	LZMA (lvl. 7)	BZip3	Ratio
B2HHH	985.15 MB	991.57 MB	100.7%
h1dst	1.80 GB	1.83 GB	102.1%
ttjet	2.32 GB	2.20 GB	95.1%
gg_data	579.56 MB	571.71 MB	98.6%



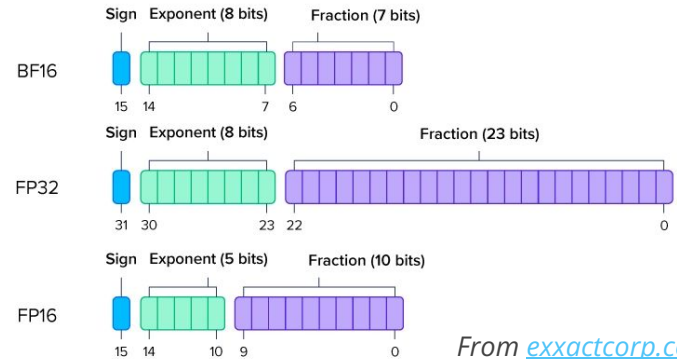
# Compression: low-precision float data

Compared to TTree, RNTuple offers more flexibility to store lossy float data:

- **Decoupling of lossiness** in data *type* and *schema*
- Making it possible to represent **lossiness inside collections**

As well as more ways to express lossiness:

- **IEEE-754 half-precision floats**
- IEEE-754 single precision float with **truncated mantissa**
- **Quantized integer representation** of real values contained in a specified range



**With this, all means for storing low-precision floating point numbers are available**



# Merging and copying



# The future of merging

x	y	x	y
x0	y0	x2	y2
x1	y1	x3	y3



x	y
x0	y0
x1	y1
x2	y2
x3	y3

When possible, RNTuple uses **fast merging** to avoid data recompression and reserialization

A potential next step is **zero-copy merging**

```
[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..931657) 80 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
```

Blocks are shared between source RNTuples and merged RNTuple

RNTuple 1

RNTuple 2

Merged RNTuple

Initial prototype by [E. Marinelli](#)

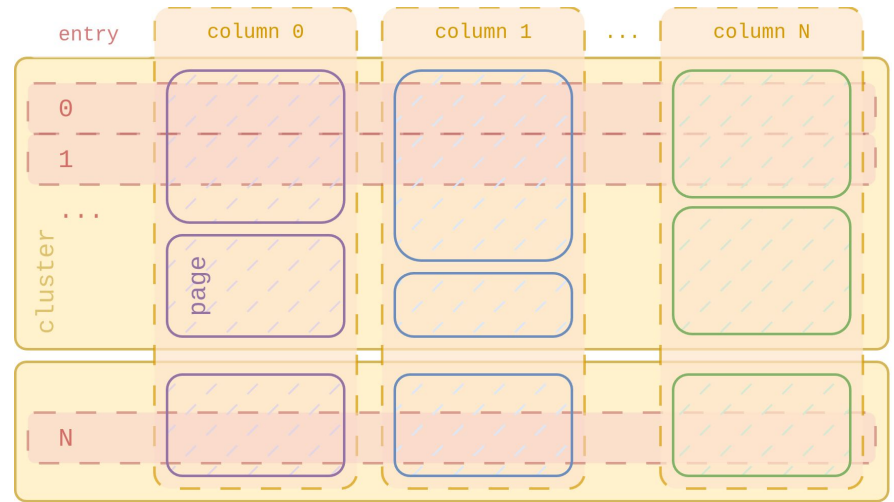


# Recluster-on-merge(-or-copy)

**The idea:** reoptimize RNTuple cluster sizes during merging

*Is this also relevant for “just” copying?*

- Different storage systems may benefit from different RNTuple cluster configurations
- We could potentially optimize the clustering for the target system upon copy and/or merge



Final clusters in a dataset may not be optimally filled

**Would this be useful, could this make a (big) difference?**



# Summary and outlook



## RNTuple enables us to take ROOT's I/O to the next level

These are our ideas! We would like to know:

- What are your priorities for the future of I/O?
- Is there anything we did not mention?

As always: **thank you!**