ROOT I/O and Foreign Languages

Jakob Blomer with material from Philippe Canal

ROOT
Data Analysis Framework
https://root.cern

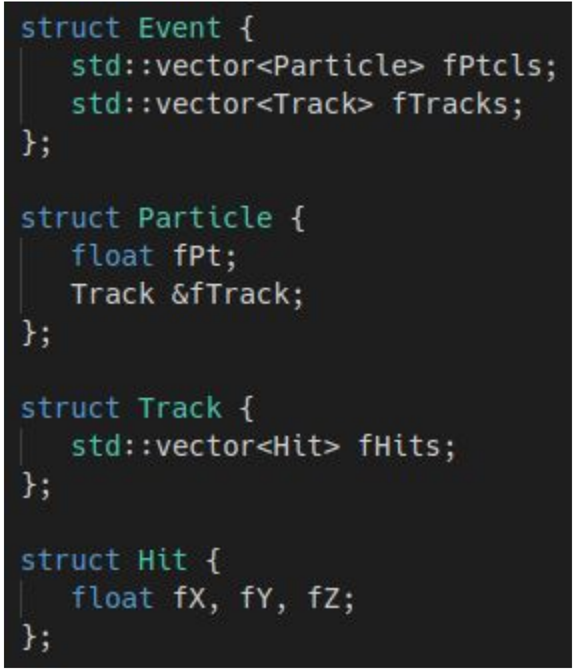
HEP Event Data I/O

Why invest in a **tailor-made I/O system**

TTree & RNTuple

- Capable of storing the **HEP event data model**: nested, inter-dependent collections of data points
- **Performance-tuned** for HEP analysis workflow (columnar binary layout, custom compression etc.)
- **Automatic schema** generation and evolution for C++ (via cling) and Python (via cling + PyROOT)
- Integration with **federated data management** tools (XRootD etc.)
- Long-term **maintenance** and support

Example EDM



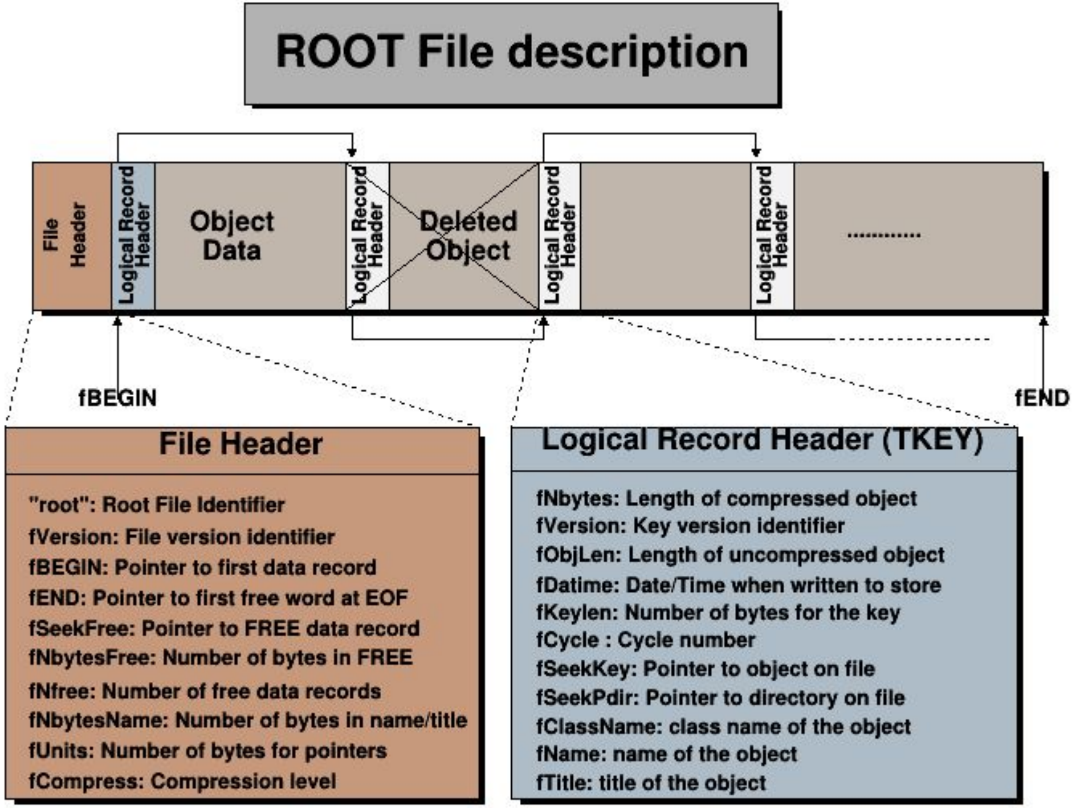
```
struct Event {
    std::vector<Particle> fPtcls;
    std::vector<Track> fTracks;
};

struct Particle {
    float fPt;
    Track &fTrack;
};

struct Track {
    std::vector<Hit> fHits;
};

struct Hit {
    float fX, fY, fZ;
};
```

The ROOT File

- In ROOT, objects are written in files* (“TFile”)
- TFiles are *binary* and have: a *header*, *records* and can be compressed (transparently for the user)
- TFiles have a logical “file system like” structure
  - e.g. directory hierarchy
- TFiles are self-descriptive:
  - Can be read without the code of the objects streamed into them
  - E.g. can be read from JavaScript

* this is an understatement - we’ll not go into the details.

ROOT File Description



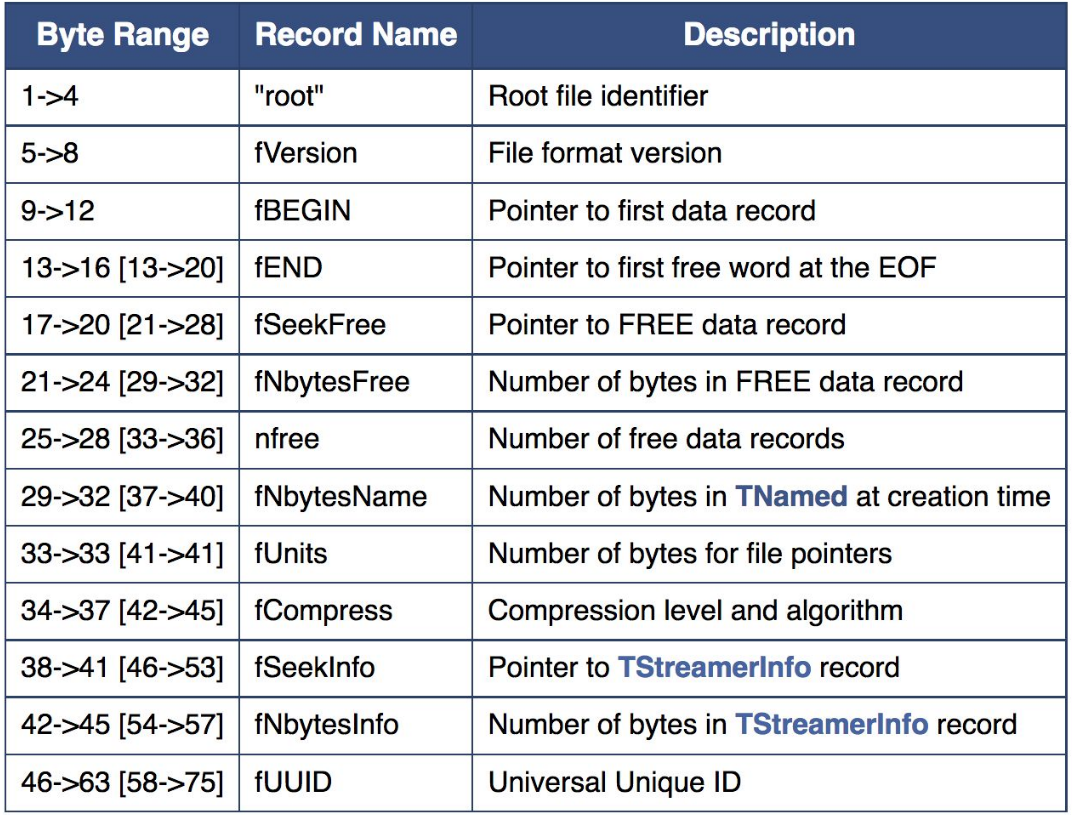
File Header

"root": Root File Identifier
fVersion: File version identifier
fBEGIN: Pointer to first data record
fEND: Pointer to first free word at EOF
fSeekFree: Pointer to FREE data record
fNbytesFree: Number of bytes in FREE
fNfree: Number of free data records
fNbytesName: Number of bytes in name/title
fUnits: Number of bytes for pointers
fCompress: Compression level

Logical Record Header (TKEY)

fNbytes: Length of compressed object
fVersion: Key version identifier
fObjLen: Length of uncompressed object
fDatime: Date/Time when written to store
fKeylen: Number of bytes for the key
fCycle : Cycle number
fSeekKey: Pointer to object on file
fSeekPdir: Pointer to directory on file
fClassName: class name of the object
fName: name of the object
fTitle: title of the object

ROOT File Specification

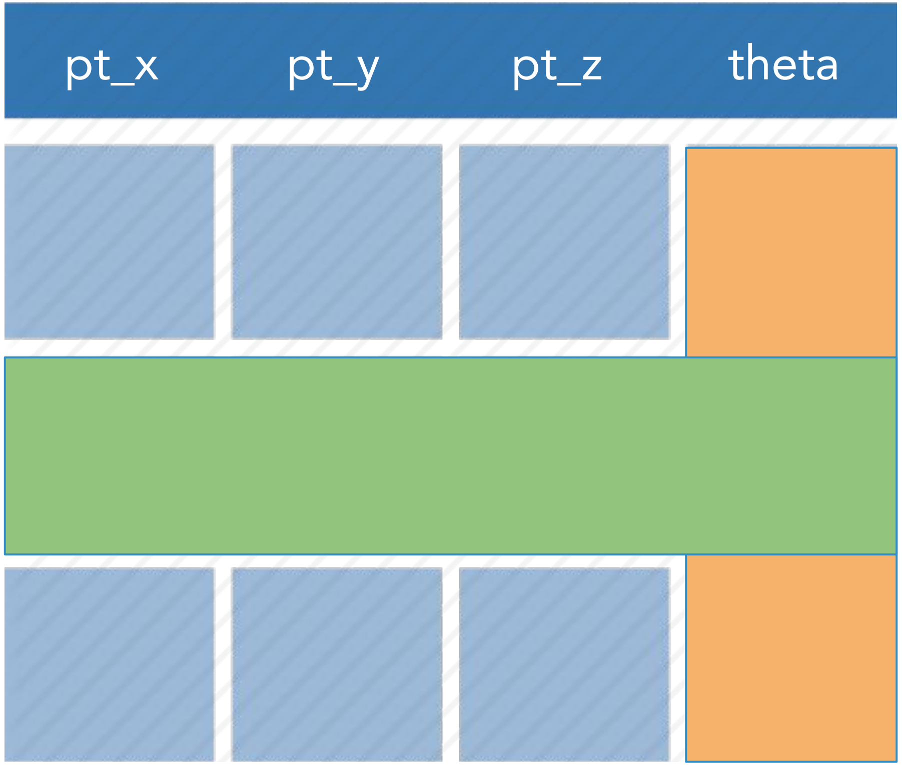


Byte Range	Record Name	Description
1->4	"root"	Root file identifier
5->8	fVersion	File format version
9->12	fBEGIN	Pointer to first data record
13->16 [13->20]	fEND	Pointer to first free word at the EOF
17->20 [21->28]	fSeekFree	Pointer to FREE data record
21->24 [29->32]	fNbytesFree	Number of bytes in FREE data record
25->28 [33->36]	nfree	Number of free data records
29->32 [37->40]	fNbytesName	Number of bytes in **TNamed** at creation time
33->33 [41->41]	fUnits	Number of bytes for file pointers
34->37 [42->45]	fCompress	Compression level and algorithm
38->41 [46->53]	fSeekInfo	Pointer to **TStreamerInfo** record
42->45 [54->57]	fNbytesInfo	Number of bytes in **TStreamerInfo** record
46->63 [58->75]	fUUID	Universal Unique ID

Event Data and ROOT Files

- A ROOT file can be seen as a hierarchically organized container of objects
  - E.g. a file can contain directories with histograms
- In addition, ROOT files can also contain event data
  - E.g., a series of `TEvent` objects for a user-defined `TEvent` class
- Event data stored in a TTree (or RNTuple, see later) is usually written as a set of many objects
- TTree and RNTuple have a custom, internal serialization format (columnar layout)
- A binary format within the TFile binary format

Columnar Representation

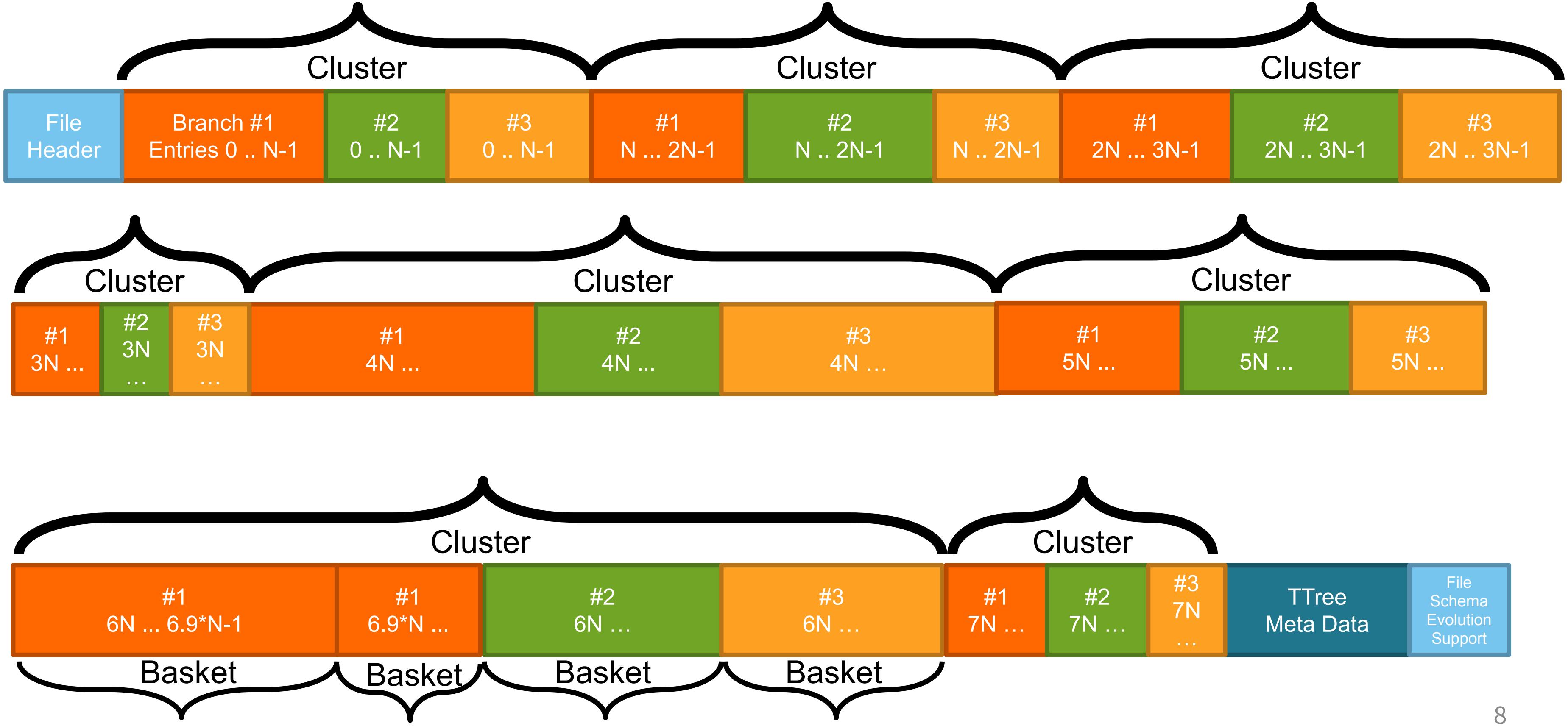


entries
or events
or rows

columns
or “branches”

can contain complex
c++ objects

Anatomy of a Tree



Cluster
Cluster
Cluster

File Header
Branch #1 Entries 0 .. N-1
#2 0 .. N-1
#3 0 .. N-1
#1 N ... 2N-1
#2 N .. 2N-1
#3 N .. 2N-1
#1 2N ... 3N-1
#2 2N .. 3N-1
#3 2N .. 3N-1

Cluster
Cluster
Cluster

#1 3N ...
#2 3N …
#3 3N …
#1 4N ...
#2 4N ...
#3 4N …
#1 5N ...
#2 5N ...
#3 5N ...

Cluster
Cluster

#1 6N ... 6.9*N-1
#1 6.9*N ...
#2 6N …
#3 6N …
#1 7N …
#2 7N …
#3 7N …
TTree Meta Data
File Schema Evolution Support

Basket
Basket
Basket
Basket

ROOT Data Access Options

- ROOT can read, write, and represent data in C++
- ROOT can read, write, and represent data in Python through pyROOT (dynamic binding between C++ and Python)
  - Can also export ROOT trees to numpy arrays
- ROOT can read and represent trees and the most common classes (histograms, graphs, etc.) in JavaScript with JSROOT
  - Can also export objects in JSON

3rd Party Implementations of ROOT I/O

- There are several projects that re-implement parts of the ROOT file format
    - Julia: unroot
    - Python: uproot
    - Go: hep/groot
    - Java/Scala: FreeHEP rootio
    - Rust: alice-rs/root-io

- Typically supported features: reading of simple objects (histograms) and trees with a simple structure (numerical types and vectors thereof)

Facets of a full I/O system

In addition to reading the most common file contents, the full I/O system has many more aspects, such as

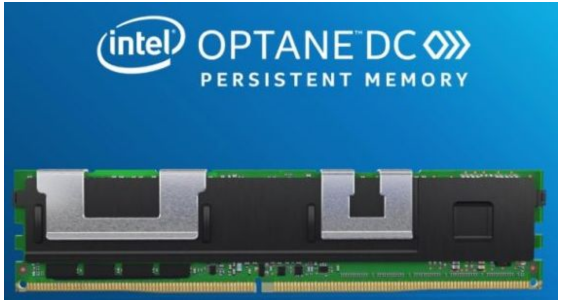
- Parallel and distributed reading & writing
- I/O scheduling (read-ahead, request coalescing, etc)
- Beyond file system I/O: HTTP, XRootD, object stores
- Schema evolution
- Data set combinations: chains, friends, indexes, merging
- Complex object hierarchies (e.g. for ESD EDMs)
- User customizations
  - E.g. skip “transient data members”
  - I/O customization rule (transformation of data)

Motivation for RNTuple

1. HL-LHC challenge: major milestone on the way towards future accelerators and detectors
    - From 300fb^{-1} in run 1-3 to 3000fb^{-1} in run 4-6
    - 10B events/year to 100B events/year
    - Real analysis challenge depends on several factors: number of events, analysis complexity, number of reruns, etc.
        - **As a starting point, preparing for ten times the current demand**

2. Full exploitation of modern storage hardware
    - Ultra fast networks and SSDs: 10GB/s per device reachable (HDD: 250MB/s)
    - Flash storage is inherently parallel ➔ asynchronous, parallel I/O 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

At 10GB/s, we have ~3µs to process a 32kB block
➔ Suggests an updated software design

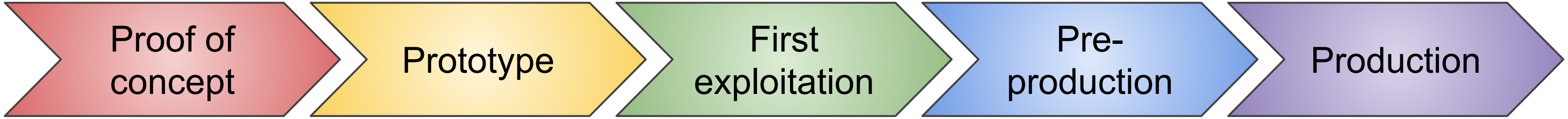


RNTuple Goals

Based on 25+ years of TTree experience, we redesign the I/O subsystem for

- Less disk and CPU usage for same data content
    - 25% smaller files, x2-5 better single-core performance
    - 10GB/s per box and 1GB/s per core sustained end-to-end throughput (compressed data to histograms)
- Native support for object stores (targeting HPC)
- Lossy compression
- Systematic use of exceptions to prevent silent I/O errors

RNTuple Development Plan



~2018-19	~2019-20	~2021-22	~2022-23	~2023-24
Proof of concept	Prototype	First exploitation	Pre-production	Production
• Class design • File format R&D	• Interplay with other ROOT classes • Performance validation	• Interplay with experiment frameworks • Schema evolution	• PB scale test cases • Production tests with last-stage ntuples	• Ready to use for new runs

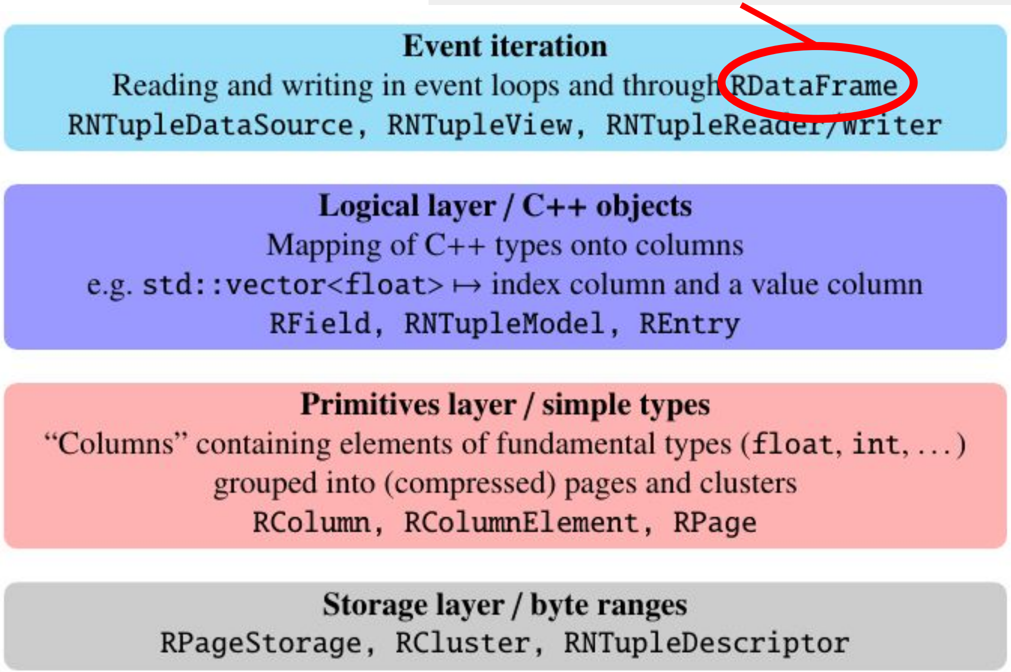
We see RNTuple as a Run 4 technology

Available now in `ROOT::Experimental`

Note: TTree technology will remain available for the 1EB+ existing data sets

RNTuple Class Design

Seamless transition from TTree to RNTuple



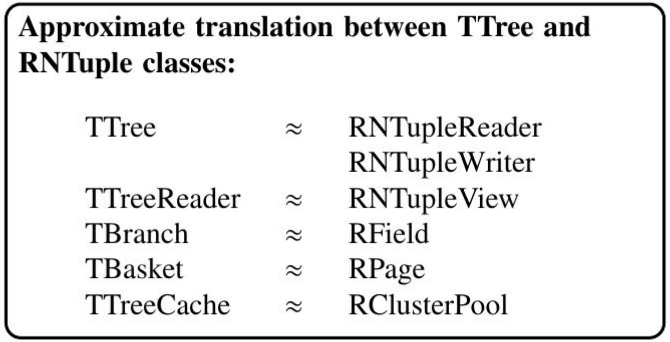
Event iteration
Reading and writing in event loops and through `RDataFrame`
`RNTupleDataSource, RNTupleView, RNTupleReader/Writer`

Logical layer / C++ objects
Mapping of C++ types onto columns
e.g. `std::vector<float>` ↦ 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`

Modular storage layer that supports files as data containers but also file-less systems (object stores)



Approximate translation between TTree and RNTuple classes:

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

→ RNTuple v1 Format Specification

RNTuple Format Evolution

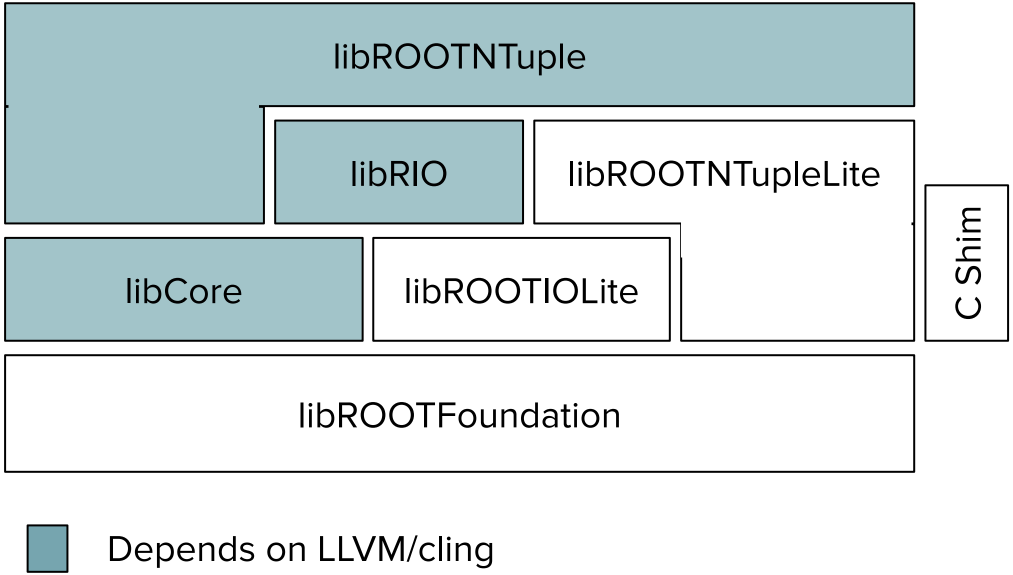
- Key binary layout changes wrt. TTree
  - More efficient nested collections
  - More efficient boolean values (bitfield), interesting for trigger bits
  - experimenting with “split floats”
  - Little-endian values (allows for mmap())

Implementation uses templates to slash memory copies and virtual function calls in common I/O paths

- Supported types
  - Boolean
  - Integers, floating point
  - std::string
  - std::vector, std::array
  - std::variant
  - User-defined classes
  - More classes planned (e.g. std::chrono timepoints)

Fully composable (including aggregation, inheritance) within the supported type system

libRNTupleLite (under development)



- The libRNTupleLite library is built just like any other ROOT libraries in ROOT proper (including modules, dictionaries etc)
- The libRNTupleLite does not use any infrastructure from libCore but only from libROOTFoundation
- Functionality:
  - RIOLite: RRawFile without support for plugins, i.e. only local files
  - ROOTNTupleLite: Provide access to meta-data (schema etc.) and data pages

libRNTupleLite C API

- C API header and dynamic library libROOTNTupleLite.so
  - Header files will be in
    - io/iolite/inc/ROOT/IOLite.h
    - tree/ntuplelite/inc/ROOT/NTupleLite.h

- Provides a C wrapper to the C++ libROOTRNTupleLite.so

- Provided functionality:
  - Open an RNTuple that is stored in a local ROOT file
  - Read the schema: fields, columns, pages, and their relationships
  - Read pages into void * memory areas given column id and page id
    - Takes care of decompressing and unpacking pages along the way

- Aims at being a building block for 3rd party tool builders

ROOT I/O: Support

Full support by the ROOT Team:

- I/O through the ROOT C++ library
- pyROOT
- Conversion of simple structures to numpy arrays
- JSROOT
- JSON serialization of objects
- In the future: C API provided by RNTupleLite

Indirect support ("support the maintainers")

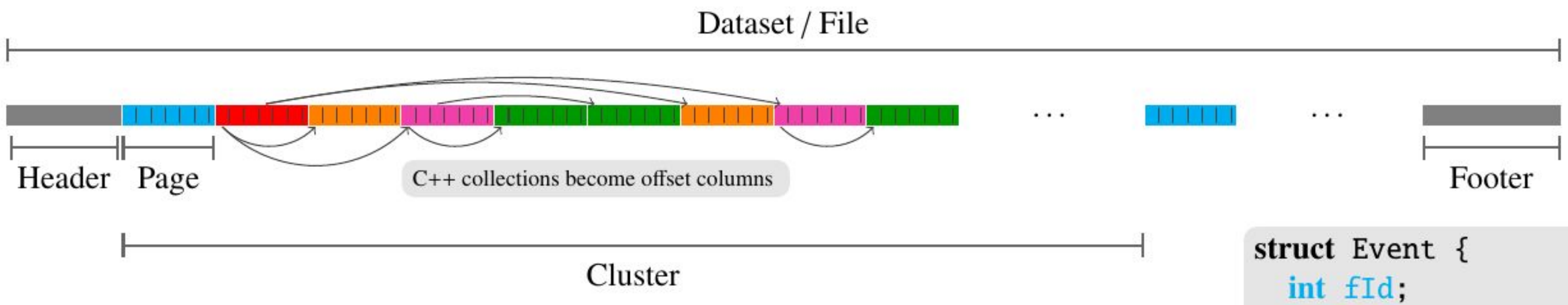
- Third-party implementation of the binary format (uproot, unroot, Java, Go, ...)

Backup slides

Resources

- ROOT Website: https://root.cern
- Introduction material: https://root.cern/getting-started
- Reference Guide: https://root.cern/doc/master/index.html
- Training material: https://github.com/root-project/training
- Forum: https://root-forum.cern.ch

RNTuple Format Breakdown



Approximate translation between TTree and RNTuple concepts:

Basket	≈	Page
Leaf	≈	Column
Cluster	≈	Cluster

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

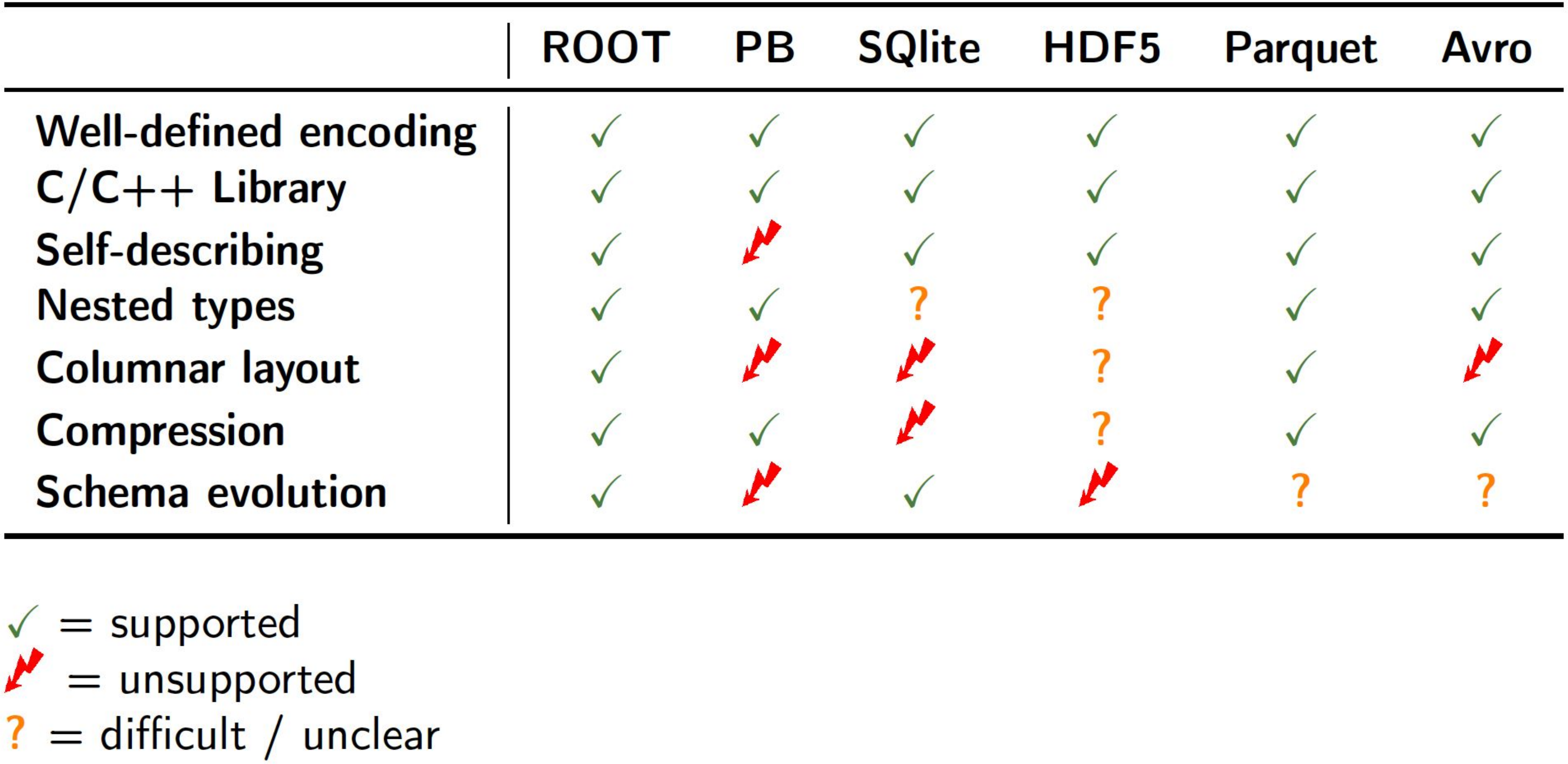
Cluster:

- Block of consecutive complete events
- Unit of thread parallelization (read & write)
- Typically tens of megabytes

Page/Basket:

- Unit of memory mapping or (de)compression
- Typically tens of kilobytes

Comparison With Other I/O Systems



	ROOT	PB	SQlite	HDF5	Parquet	Avro
Well-defined encoding	✓	✓	✓	✓	✓	✓
C/C++ Library	✓	✓	✓	✓	✓	✓
Self-describing	✓	↯	✓	✓	✓	✓
Nested types	✓	✓	?	?	✓	✓
Columnar layout	✓	↯	↯	?	✓	↯
Compression	✓	✓	↯	?	✓	✓
Schema evolution	✓	↯	✓	↯	?	?

✓ = supported
↯ = unsupported
? = difficult / unclear

J. Blomer, [A quantitative review of data formats for HEP analyses](#) ACAT 2017