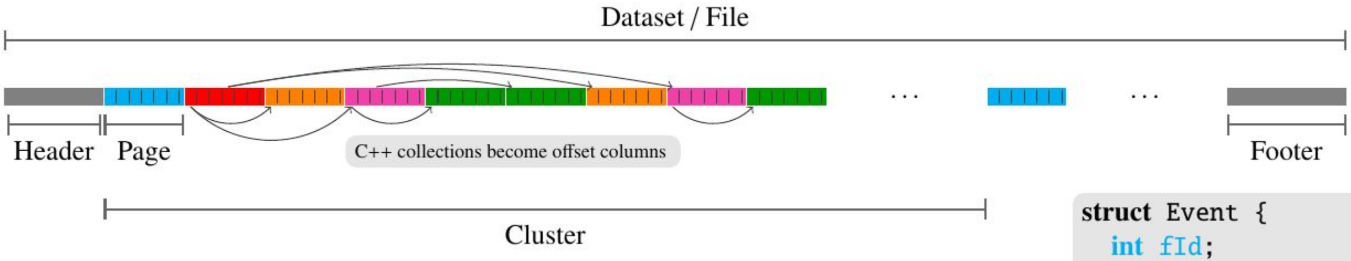
RNTupleLight C API

Jakob Blomer

2020-10-28

RNTuple On-Disk Format



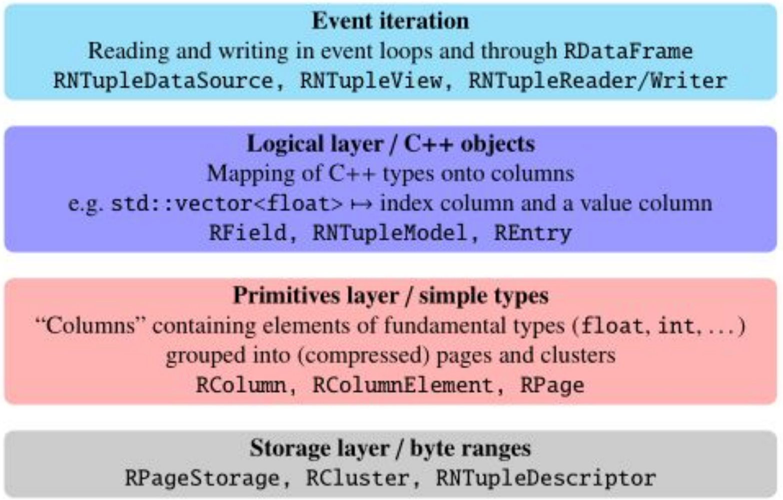
Approximate translation between TTree and RNTuple concepts:

Basket	≈	Page
Leaf	≈	Column
Cluster	≈	Cluster

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

- Header, Footer: RNTuple (non-ROOT), extensible serialization (Implementation, specification stub)
  - Header: schema information
  - Footer: location of pages and clusters
- Pages: ROOT compression envelope
  - uncompressed content: little-endian fundamental types (possibly *packed*, e.g. bitfields)
- Container format:
  - ROOT TFile (anchor TKey + header, footer, pages anonymous TKeys each)
  - *Bare file* (for internal purposes)
  - Planned for this year: DAOS object store (pages, header, footer in individual objects)

RNTuple Class Design

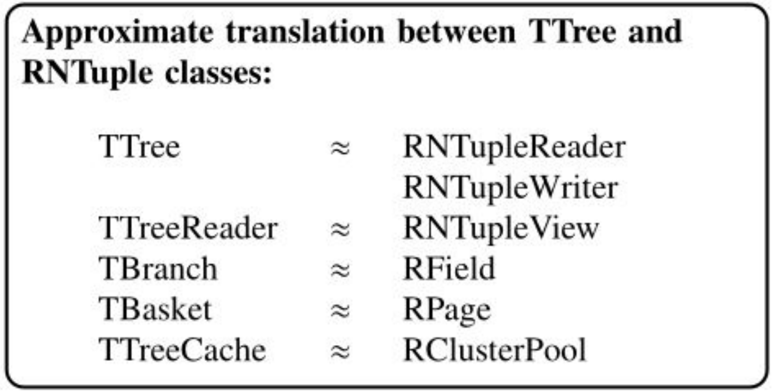


Event iteration
Reading and writing in event loops and through `RDataFrame`
`RNTupleDataSource, RNTupleView, RNTupleReader/Writer`

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`

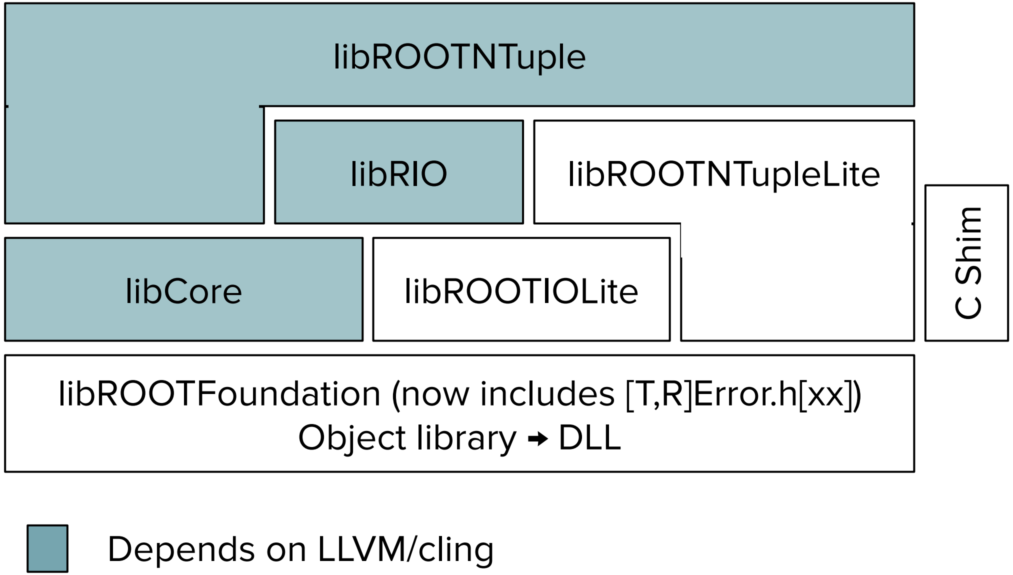


Approximate translation between TTree and RNTuple classes:

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

- Storage layer: access to the schema, the pages, and the footer (= location of pages)
- Main classes:
  - RPageSourceFile, RPageSinkFile, RRawFile (for file based access, local or remote)
  - RPageSourceDaos, RPageSinkDaos (DAOS object store)
  - …

XyzLite Library Layering



libROOTNTuple

libRIO

libROOTNTupleLite

C Shim

libCore

libROOTIOLite

libROOTFoundation (now includes [T,R]Error.h[xx])
Object library ➔ DLL

■ Depends on LLVM/cling

- The libXyzLite libraries are built just like any other ROOT libraries in ROOT proper (including modules, dictionaries etc)

- The libXyzLite libraries must not use any infrastructure from libCore but only from libROOTFoundation

- Current contents:
  - RIOLite: RRawFile without support for plugins, i.e. only local files
  - ROOTNTupleLite: RPageSink, RPageSource

RNTupleLight C API

- C API header and dynamic library, e.g., libROOTNTupleLite.so
    - Header files would be in
        - **io/iolite/inc/ROOT/IOLite.h**
        - **tree/ntuplelite/inc/ROOT/NTupleLite.h**
- Provides a C front to the C++ libROOTRNTupleLite.so
- Minimal usable subset from RNTuple Light:
    - 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
    - Unsure about cluster pool support (async parallel page loading and decompression):
        - Option 1: Unsupported with the C library
        - Option 2: C library uses threads internally (might create problems)
        - Option 3: C library provides means to let the user provide a thread scheduler
- Deliverable:
    - C test program (bails out on compilation if CXX is defined) that uses dlopen to load libROOTNTupleLite.so (no name mangling) and reads some data from an RNTuple
    - To be added to ROOTTest