The Swiss Army Knife for Datasets Manipulation and Analysis

D. Piparo (CERN, EP-SFT) for the ROOT team

ROOT

Data Analysis Framework

https://root.cern

This talk

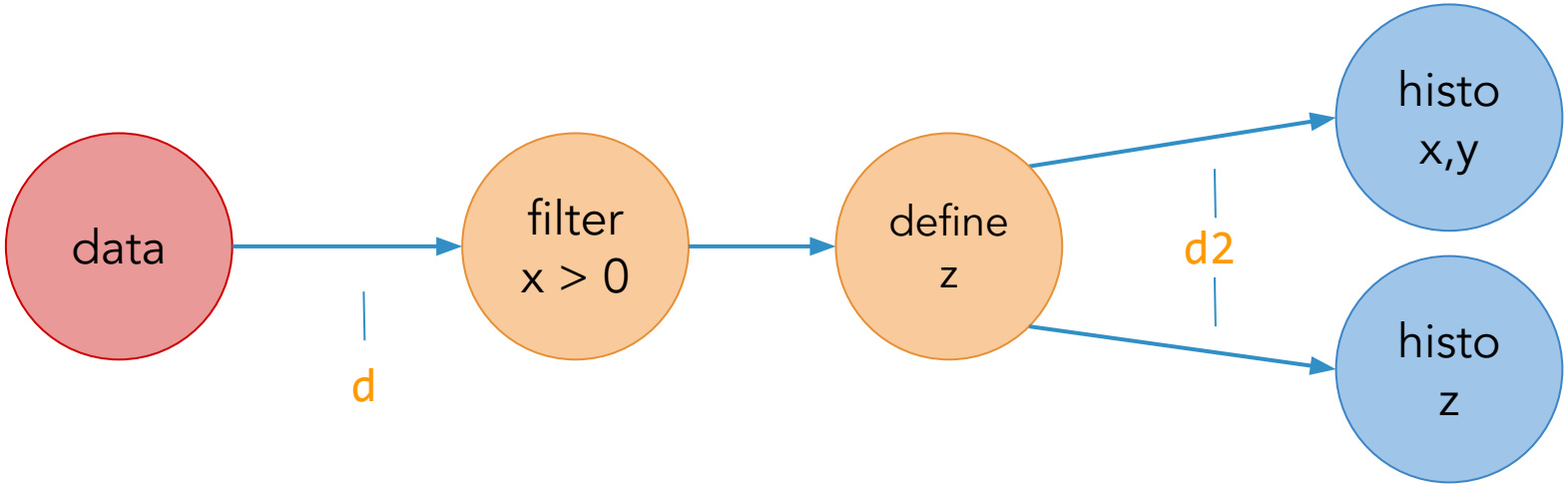
- Motivation for a ROOT Data Frame - RDataFrame (RDF)
- IO Features
  - Parallelism
  - Reading ROOT datasets
  - Reading from any kind of data source
  - Writing datasets
- The future

https://www.swiss-store.co.uk/

Why A Data Frame?

- Originally: allow to express analyses with functional chains
  - Productive programming model
  - Allow implicit parallelisation and transparent optimisation
    - Also hiding event loop
  - Check code sanity at compile time as much as possible
- Read ROOT columnar format
- Follow the trends in industry (Spark, Pandas)

Analysis as Data-Flow




```
// d2 is a new data-frame, a transformed version of d
auto d2 = d.Filter("x > 0")
           .Define("z", "x*x + y*y");
// make multiple histograms out of it
auto hz = d2.Histo1D("z");
auto hxy = d2.Histo2D("x","y");
```

… We did it in C++ and made it available in Python with PyROOT

Go Parallel!

```
ROOT::EnableImplcitMT()
```

A single statement to activate internal parallelisation (and thread safety)

See online doc for more details.

Results: Foreseen and Unexpected

Results and Side Effects

- ROOT 6.10: `Experimental::TDataFrame`, 6.14 `ROOT::RDataFrame`
  - Tens of users, 185 posts on the forum
  - Multithreaded analysis accessible to the masses
  - Everything works in sequential and IMT, same programming model

- *Several* improvements (doc, performance, scaling) and bug fixes in all corners of ROOT
- Prominent I/O related features
  - Read the same tree in parallel
  - Snapshot write RDF content on disk: same TTree written in parallel
  - Read non ROOT datasets

Read The Same Dataset in Parallel

- Partially supported by ROOT already
  - One `TFile` opened per thread
  - Needed to adapt to task based model!
    - E.g. one cluster of events per task (avoid duplicated decompression, deserialisation)
  - Build on solid lower level interfaces, e.g. `TTreeProcessorMT`
- Some pieces were missing, e.g. optimisation for MT usage of `TTreeReader{value,array}`
- Reduction of critical sections' size, e.g. at file opening
  - Check of streamer info record w/o interaction with type system

Write out Trees, also in Parallel

- Now possible to snapshot the columns of a data frame in a ROOT dataset
  - Multiple threads writing the same `TTree`
  - Low level interface used: `TBufferMerger`
- Easiest way to produce a ROOT dataset (see next slide)
  - It is also typesafe!

The content of a RDataFrame can be written to ROOT files as TTree, also in parallel

Easy Creation of Datasets

https://root.cern/doc/master/df007__snapshot_8C.html

```
auto produceVec = [](float x) {
   std::vector<float> v; v.reserve(3);
   for (int i = 0; i < 3; i++) v.emplace_back(x * i);
   return v;}
ROOT::RDataFrame d(tName, fName); // contains b1 and b2
auto d2 = d.Define("b1_square", "b1 * b1")
           .Define("b2_v", produceVec, {"b2"});
d2.Snapshot<float, float, std::vector<float>>(tName, fName, {"b1", "b1_square", "b2_v"});
// or with jitting
d2.Snapshot(treeName, outFileName, {"b1", "b1_square", "b2_v"});
```

Easy Creation of Datasets In Parallel

https://root.cern/doc/master/df007__snapshot_8C.html

```
ROOT::EnableImplicitMT();

auto produceVec = [](float x) {
   std::vector<float> v; v.reserve(3);
   for (int i = 0; i < 3; i++) v.emplace_back(x * i);
   return v;}
ROOT::RDataFrame d(tName, fName); // contains b1 and b2
auto d2 = d.Define("b1_square", "b1 * b1")
           .Define("b2_v", produceVec, {"b2"});
d2.Snapshot<float, float, std::vector<float>>(tName, fName, {"b1", "b1_square", "b2_v"});
// or with jitting
d2.Snapshot(treeName, outFileName, {"b1", "b1_square", "b2_v"});
```

See Guilherme's talk!

Go parallel with a single line

Data Sources - RDataSource

- `RDataSource`: exposes input data to `RDataFrame`
- In release: RCsvDS, RArrowDS
  - Others available RMDFDS
  - More coming: RXAodDS !
- The analysis code is decoupled from the type of source
- `RDataSource` is an interface: anyone can implement her own RDS!
- Easy to convert to ROOT datasets with Snapshot!

ROOT is now a framework to analyse potentially any columnar dataset, not only TTrees

The Future

Thoughts About the Future - 1

Achieving parallel reading, writing and processing was not easy

- ROOT's global lock, automatic registration of objects into global book-keeping
- It will be harder and harder to further parallelise

To support Run3 analysis and beyond, (even) faster reading is required

- Reading the value of a column for several entries in one go may be beneficial
  - A "Fast-Path"
- `RDataSource` to shield users from any hurdle in the programming model
- Leverage in-memory caching (landscape will change, we'll not have just RAM)

Thoughts About the Future - 2

Presently the event loop is entry based

- A dataset with N rows can be reduced to a dataset with N - K rows, K>=0
- There might be the need for overcoming this constraint (associations of entities such as vertices and tracks in a continuous readout system)
- Can the `RDataSource` be of help if we allow the `RDataFrame` to alter its state?

Thoughts About the Future - 3

RDataSource: entry point for a columnar dataset

- Interface intended for RDataFrame, not users BUT
- a complete and simple low level (void ptrs! Ptr to ptr!) reader of columns
- Can we leverage this as a bridge for new IO prototypes?
  - New IO, same code for the analysis
  - Testable, benchmarkable