TTreeFormula TTree::Draw, TTree::Scan

Philippe Canal, FNAL

TTree::Draw

- Designed for quick, convenient, exploratory analysis
  - Plots, cuts, weights
- Reverse Polish Engine
- Designed to minimize I/O
- Organically grown over (a long) time

- Can produce
  - 1,2,3,4 D histograms
  - Scatter plots up to 5D
  - Parallel coordinates plots, Candle sticks charts
  - Entry list

TTree::Scan

- Textual representation of the content of *TTree*.
- Allows printing several formulaes
- With cuts and weights.
- Both *TTree::Draw* and *TTree::Scan* uses *TTreeFormula* for evaluation.

TTreeFormula

Started simple with:

- Arithmetic operations on branch/leaf names.

  tree->***Draw***(“**event.fMass * event.fMass**”)

- With “*math.h*” function support

  tree->***Draw***(“**sqrt(event.fMass2)**”)

- And cuts

  tree->Draw(“**sqrt(event.fMass2)**”, “**fNvertex > 2**”);

  2nd arg is both a selection and a weight

Arrays

- Quickly extended to 1D arrays

```
tree->Draw(”track.pz",”track.pt>3");
```

- With an implicit ‘coordinated’ loop over the arrays.

Equivalent to:

```
foreach entry
    foreach track:
        if (track.pt>3)
            plot(“track.pz*track.pt>3");
```

- Coordinated
  - Heuristic/choices to handle uneven collections.

Multi-dimensional arrays

- Implicit loop over multiple dimension.
- Some control/fix of index.
- Coordination with other arrays.

- With fMatrix a 3 by 3 array and fResults a 5 by 2 array.

String passed	What is used for each entry of the tree
`fMatrix`	the 9 elements of fMatrix
`fMatrix[][]`	the 9 elements of fMatrix
`fMatrix[2][2]`	only the elements fMatrix[2][2]
`fMatrix[1]`	the 3 elements fMatrix[1][0], fMatrix[1][1] and fMatrix[1][2]
`fMatrix[1][]`	the 3 elements fMatrix[1][0], fMatrix[1][1] and fMatrix[1][2]
`fMatrix[][0]`	the 3 elements fMatrix[0][0], fMatrix[1][0] and fMatrix[2][0]

expression	element(s)	Loop
`fMatrix[2][1] - fResults[5][2]`	one	no loop
`fMatrix[2][] - fResults[5][2]`	three	on 2nd dim fMatrix
`fMatrix[2][] - fResults[5][]`	two	on both 2nd dimensions
`fMatrix[][2] - fResults[][1]`	three	on both 1st dimensions
`fMatrix[][2] - fResults[][]`	six	on both 1st and 2nd dimensions of fResults
`fMatrix[][2] - fResults[3][]`	two	on 1st dim of fMatrix and 2nd of fResults (at the same time)
`fMatrix[][] - fResults[][]`	six	on 1st dim then on 2nd dim
`fMatrix[][fResult[][]]`	30	on 1st dim of fMatrix then on both dimensions of fResults. The value if fResults[j][k] is used as the second index of fMatrix.

User functions

- Extended to allow the use of arbitrary user provided functions with significant caveats:
  - Only functions that takes and return simple numerical types.

tree->***Draw***(“**TMath::BreitWigner(fPx,3,2)**”)

Special functions

- This: pointer to the *TTree* itself:

```
tree->Draw("This->GetUserInfo()->At(0)->GetName()");
```

- Entry$: return the current entry number
- LocalEntry$: return the current entry number in the current tree of a chain
- Entries$: return the total number of entries
- LocalEntries$: return the total number of entries in the current tree of a chain
- Length$: return the total number of element of this formula for this entry
- Iteration$: return the current iteration over this formula for this entry (i.e. varies from 0 to Length$).
- Length$(formula): return the total number of element of the formula given as a parameter.

Special functions

- Sum$(formula)
  - sum of the value of the elements of the formula given as a parameter.
  - For example the mean for all the elements in one entry can be calculated with: Sum$(formula)/Length$(formula)
- Min$(formula)
  - minimun (within one TTree entry) of the value of the elements of the formula given as a parameter.
- Max$(formula)
  - maximum (within one TTree entry) of the value of the elements of the formula given as a parameter.
- MinIf$(formula,condition) MaxIf$(formula,condition)
  - minimum (maximum) (within one TTree entry) of the value of the elements of the formula given as a parameter if they match the condition.
- Alt$(primary, alternate)
  - return the value of "primary" if it is available for the current iteration otherwise return the value of "alternate".
  - For example, with arr1[3] and arr2[2] tree->***Draw***("**arr1+Alt$(arr2,0)**");
  - will draw **arr1[0]+arr2[0] ; arr1[1]+arr2[1] and arr1[2]+0**

Objects

- Support calling functions on objects (with same limitations as before)

  tree->***Draw***("**event.GetTriggerBits()**")

- Support for drilling through unsplit objects:

  tree->***Draw***("**event.fTrack.fVertex.fPosition.fX**")

  - Works independently of the split level.

Collections

- Same syntax for *STL* collection and *TClonesArray* as for arrays
  - Implicit loopings, multi-dimension control via [][number][..]

- Access to information about the collection via @ notation:

tree->***Draw***("**event.fTracks.size()**"); calls *Track::size*

tree->***Draw***("**event.@fTracks.size()**"); calls *std::vector::size*

Other features

- Control of the range of entries (start and number)
- Control the histogram's shape:

```
tree->Draw("sqrt(x):sin(y)>>hsqrt(100,10,60,50,.1,.5)")
```

- Aliases (string based)
- Fuzzy branch name 'prefix' match
- Ternary operation
- Boolean operation short-circuiting

Conclusions

- Simple, Convenient, Quick
- Easy to use for GUI (*TBrowser*, *TTreeViewer*)

- Significant but limited feature set
- Transition to ‘next’ framework challenging (when loops involved)
- All calculation in double precision or 64 bits longs
- Text based …

RDataFrame slides for TTReeFormula contribution

D. Piparo

ROOT

Data Analysis Framework

https://root.cern

These slides

RDataFrame is not a DSL but a tool to process and analyse columnar datasets

Structure analysis in terms of transformations, actions (inspired by Spark) and *side effects*:

- Transformations examples: filter, definition of new quantities
- Actions examples: create an histogram, a graph
- Side-effect examples: cache dataset in memory, dump dataset on disk

In a sense, a *narrative* about your analysis

A recipe for efficient HEP analyses

- ➔ strive for a **simple programming model**
- ➔ expose modern, elegant interfaces that are

  easy to use correctly and hard to use incorrectly
- ➔ allow to **transparently benefit from parallelism**

HEP is not alone in these challenges:
we can **learn from the data science industry**
and bring back what physicists need, in the form they need it

---

RDataFrame, officially part of ROOT since v6.14, tries to incarnate these ideas in the context of HEP analyses and HEP data manipulation

RDF user guide, cheat-sheet

DOI 10.5281/zenodo.260230

Classic interfaces: TTreeReader

```
TTreeReader reader("myDataset", myFile);
TTreeReaderValue<A> x(reader,"x");
TTreeReaderValue<B> y(reader,"y");
TTreeReaderValue<C> z(reader,"z");
while (reader.Next()) {
   if (IsGoodEntry(*x, *y, *z))
      h->Fill(*x);
}
```

- full control over the event loop
- requires some boilerplate
- users implement common tasks again and again
- parallelisation is not trivial

(cc) BY

Classic interfaces: TTreeDraw

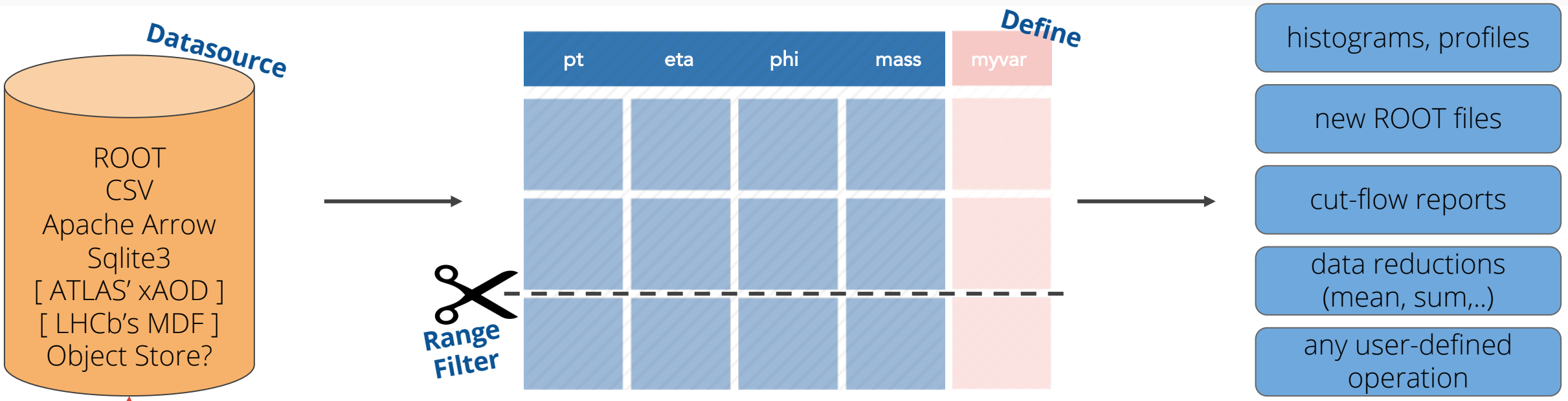
```
myDataset->Draw("pt", "eta > 2")

myDataset->Draw("Muon_pt","Sum$(Muon_pt*(Muon_eta > 1)) > 30")
```

- ad-hoc language allows to quickly specify queries
- can only produce histograms/graphs
- one event loop per histogram
- parallelisation is not possible
- relies on ROOT memory management of the histogram

can we address these limitations without losing expressivity?

(cc) BY

ROOT Declarative Analysis: RDataFrame



Customisation point,
public interface!

Goals:

- ➔ Be the **fastest** way to manipulate HEP data
- ➔ Be the **go-to ROOT analysis interface** from laptop to cluster
- ➔ Consistent interfaces in **Python and C++**
- ➔ Top notch documentation and examples

An ergonomic, fast C++ dataframe

```
ROOT::EnableImplicitMT();
```
Run a parallel analysis

```
ROOT::RDataFrame df(dataset);
```
on this (ROOT, CSV, ...) dataset

```
auto df2 = df.Filter("x > 0")
```
only accept events for which **x > 0**

```
             .Define("r2", "x*x + y*y");
```
define **r2 = x² + y²**

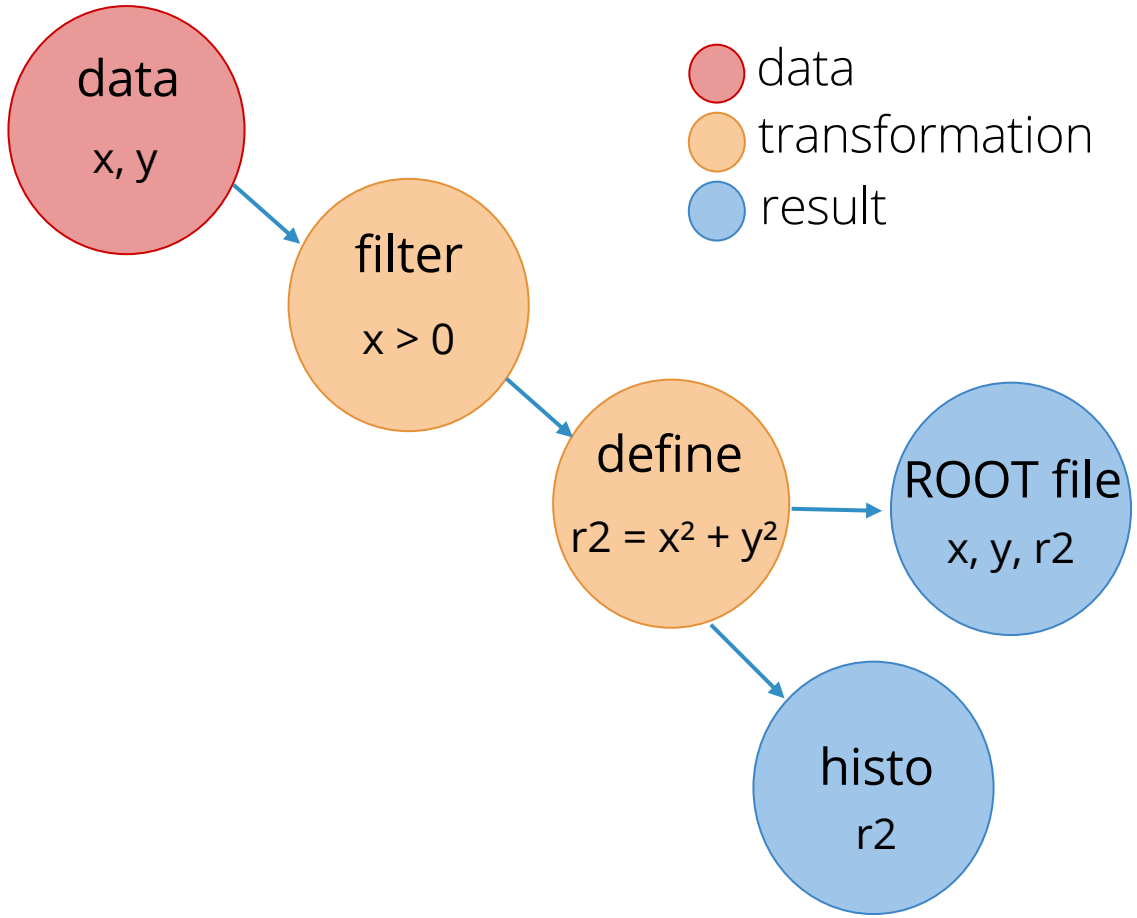
```
auto rHist = df2.Histo1D("r2");
```
plot **r2** for events that pass the cut

```
df2.Snapshot("newtree", "out.root");
```
write the skimmed data and **r2** to a new ROOT file

Lazy execution guarantees that all operations are performed in **one event loop**

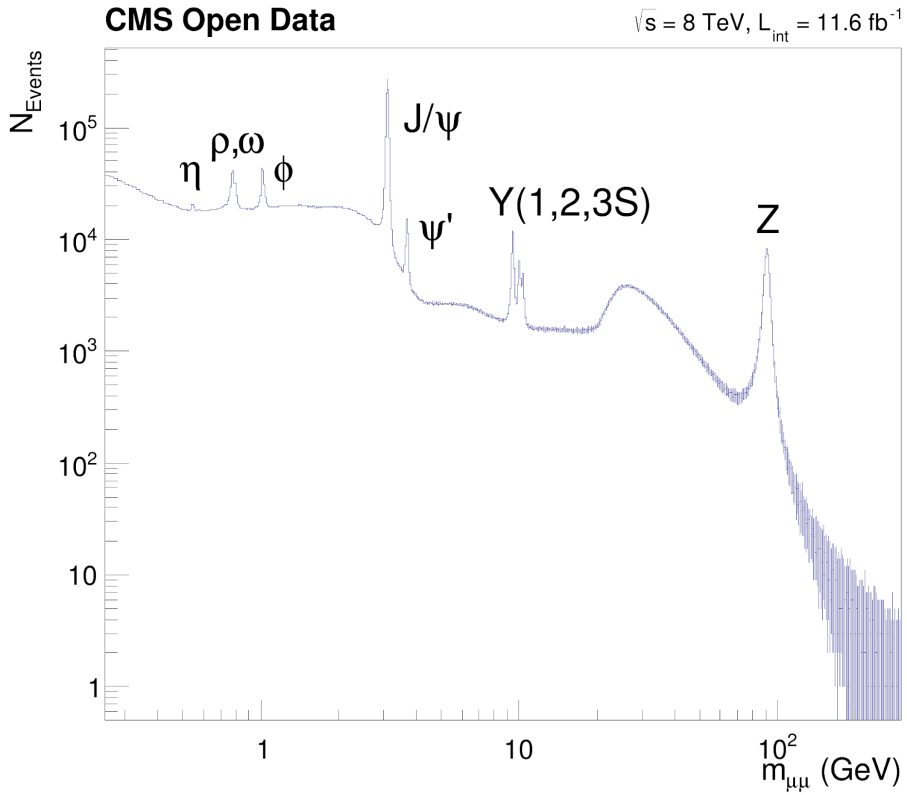
Analyses as computation graphs

```
ROOT::RDataFrame df(dataset);
auto df2 = df.Filter("x > 0")
             .Define("r2", "x*x + y*y");
auto rHist = df2.Histo1D("r2");
df2.Snapshot("newtree", "newfile.root");
```

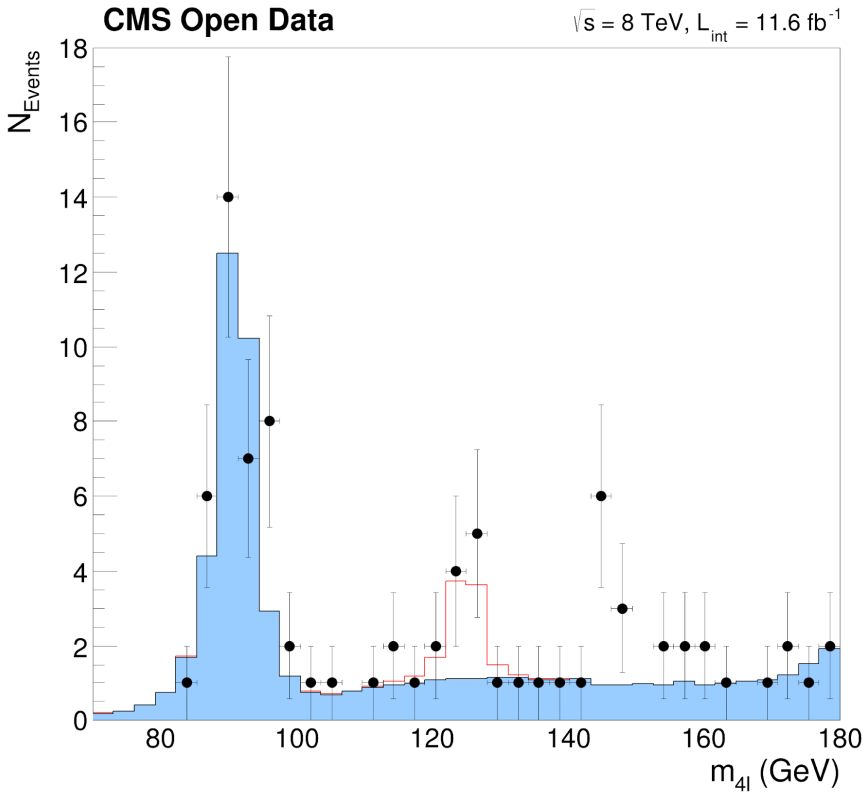


Write datasets to disk, also in parallel.

From CMS Open Data



Full Example



Full Example

Fully runnable examples with data and code

More realistic analysis examples in the pipeline!

From CMS Open Data

CMS Open Data $\sqrt{s}$ = 8 TeV, L_{int} = 11.6 fb^{-1}

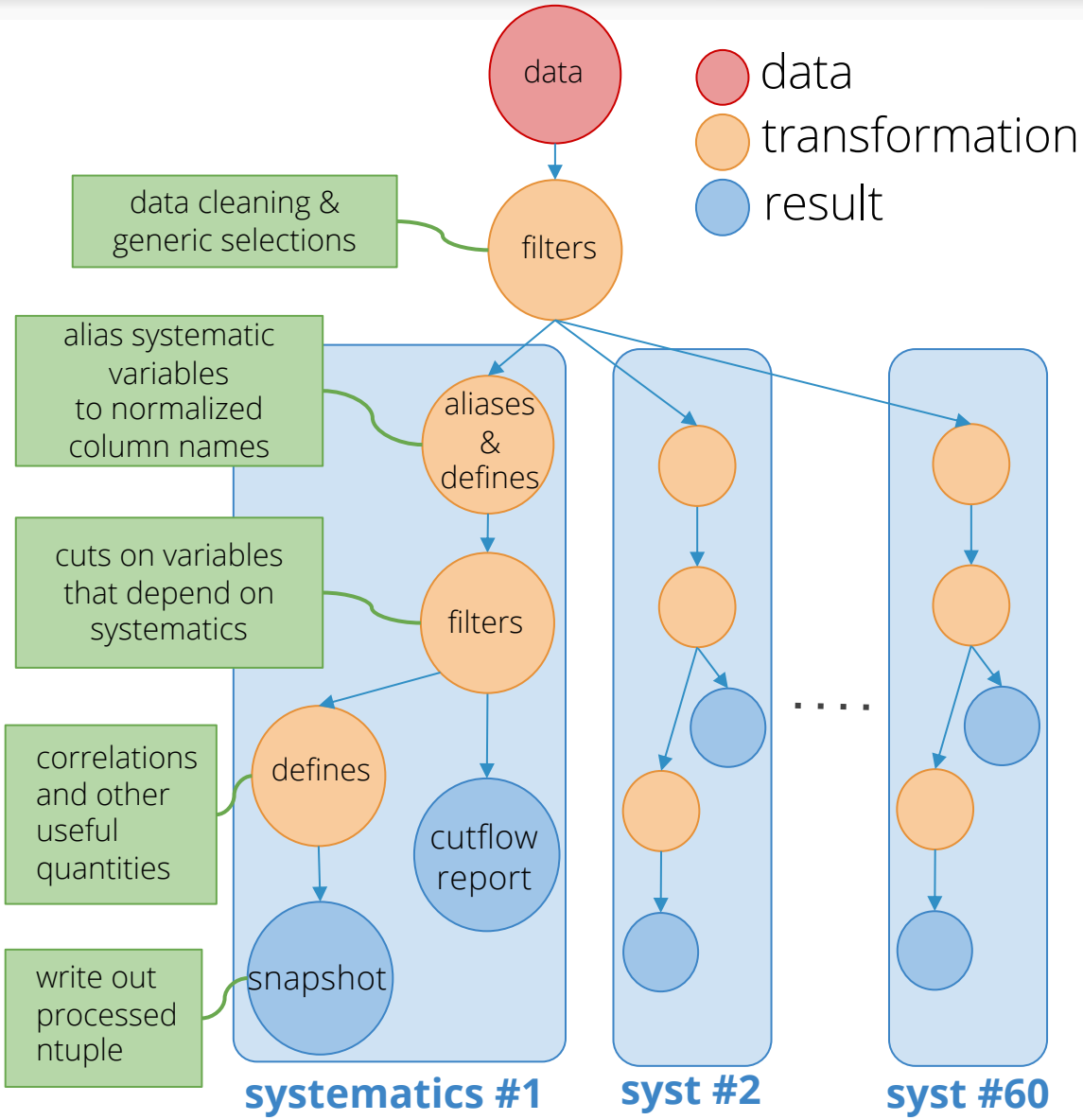
CMS Open Data $\sqrt{s}$ = 8 TeV, L_{int} = 11.6 fb^{-1}

```
// Enable multi-threading
ROOT::EnableImplicitMT();

// Create dataframe from NanoAOD files
ROOT::RDataFrame df("Events",
                    {"root://eospublic.cern.ch//eos/root-eos/cms_opendata_2012_nanoaod
    /Run2012B_DoubleMuParked.root",
                     "root://eospublic.cern.ch//eos/root-eos/cms_opendata_2012_nanoaod
    /Run2012C_DoubleMuParked.root"});

// For simplicity, select only events with exactly two muons and require opposite charge
auto df_2mu = df.Filter("nMuon == 2", "Events with exactly two muons");
auto df_os = df_2mu.Filter("Muon_charge[0] != Muon_charge[1]", "Muons with opposite
    charge");

// Compute invariant mass of the dimuon system
auto df_mass = df_os.Define("Dimuon_mass", InvariantMass<float>, {"Muon_pt", "Muon_eta",
    "Muon_phi", "Muon_mass"});

// Make histogram of dimuon mass spectrum
auto h = df_mass.Histo1D({"Dimuon_mass", "Dimuon_mass", 30000, 0.25, 300},
    "Dimuon_mass");
```

Fully runnable examples with data and code

More realistic analysis examples in the pipeline!

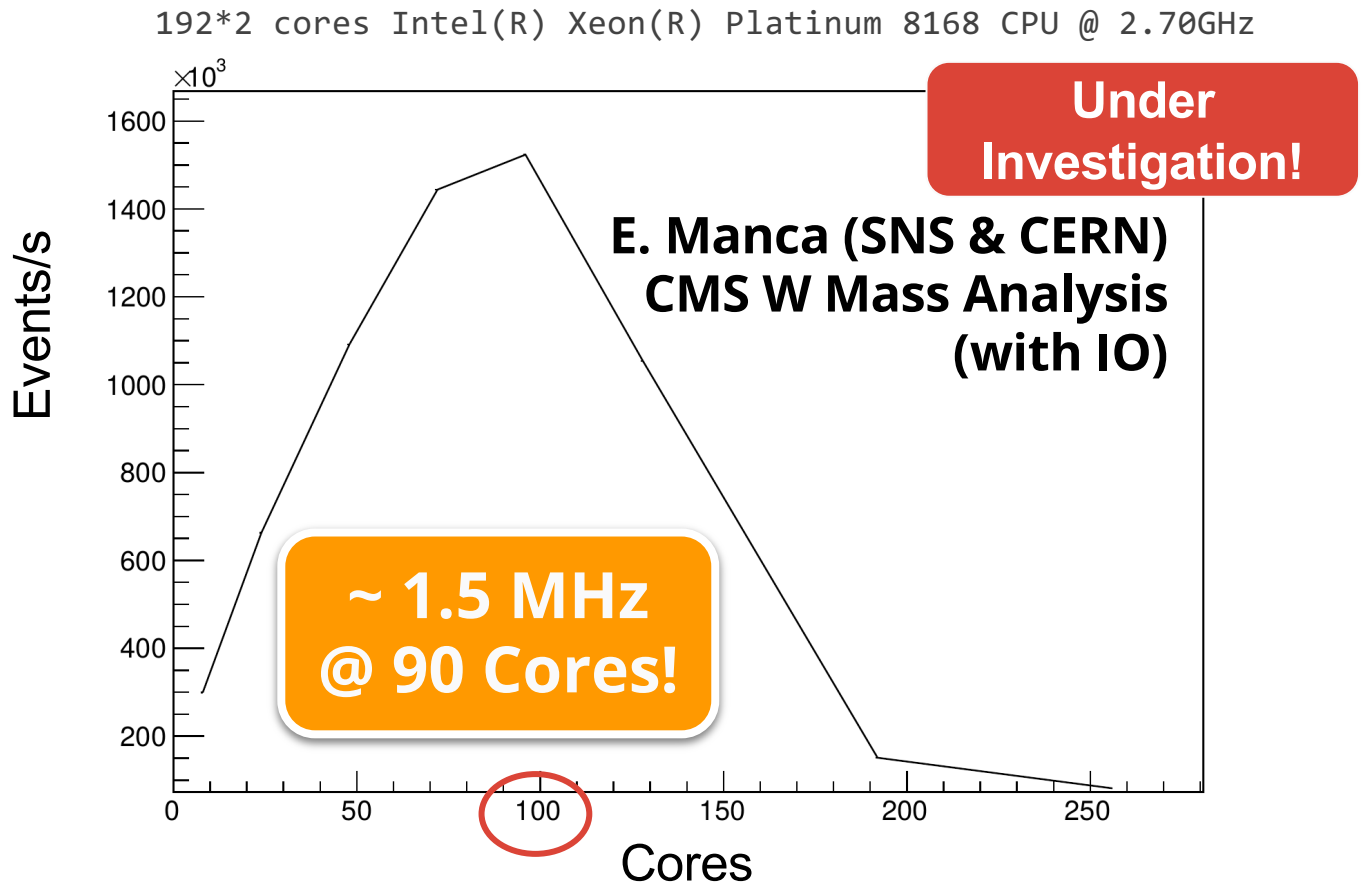
Case study: ATLAS SUSY ntuple → ntuple



Local ntuple → ntuple processing, MC data is processed to add quantities relevant for publication

- ➔ program's **main** reads similarly to this graph
- ➔ the large blue boxes represent one single function that applies the same operations to an RDF variable and is re-used for all different systematics
- ➔ cuts, calculations and writing of the 60 output trees all happen in the same multi-thread event loop

Does All This Scale?



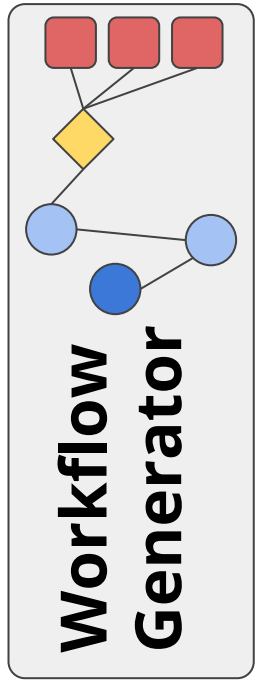
RDataFrame Scales on Many Cores

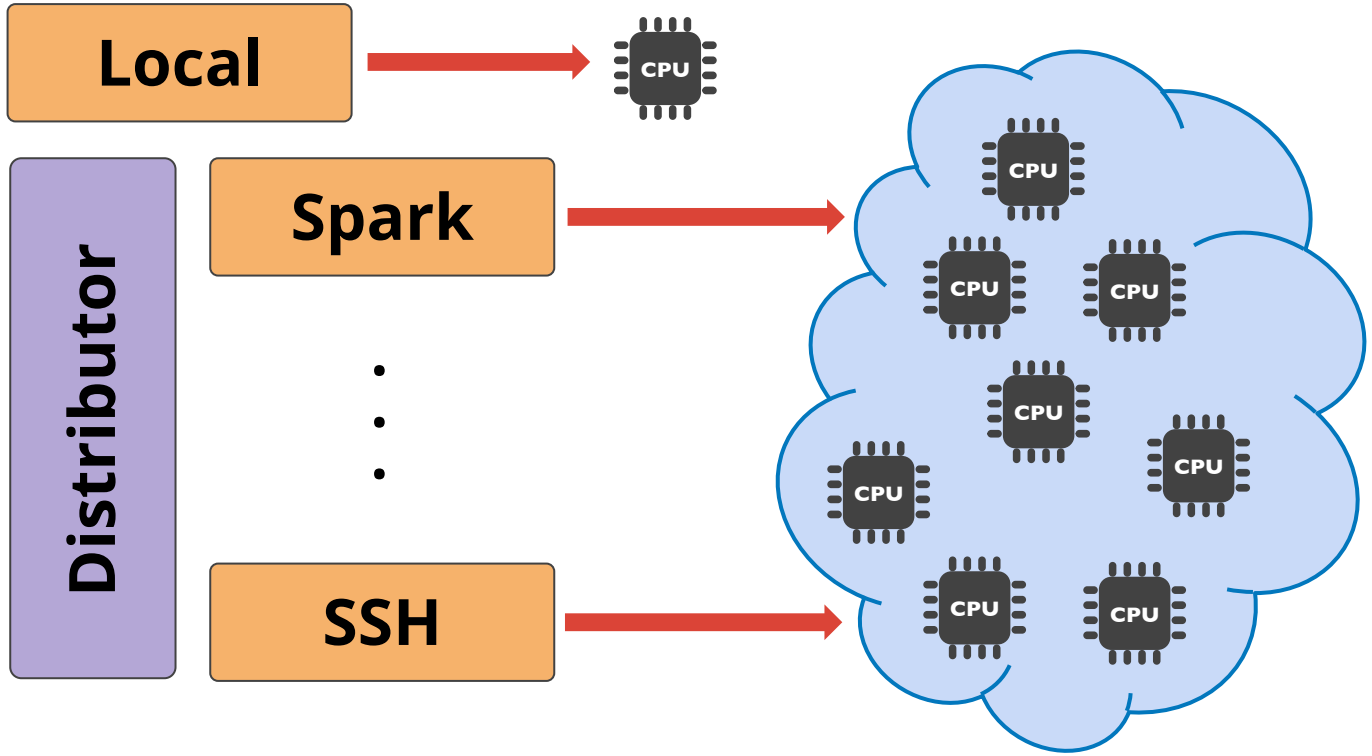
Investigate and prototype a complement to PROOF

Parallelism on many nodes
Transparent distribution
Support several different backends

Re-use RDF interface: Minimal/No change in analysis code

```
d = RDataFrame (“t”, dataset)
f = d.Define(...)
      .Define(...)
      .Filter(...)

h1 = f.Histo1D(...)
h2 = f.Histo1D(...)
h3 = f.Histo1D(...)
```





JavierCVilla/PyRDF

Not in 6.18
Working prototype available!

Array Syntax and its Support

Many of the quantities in HEP are collections

- e.g. vector<particle>

Ergonomic interfaces for treating collections: a must

ROOT::RVec class:

- std::vector like interface
- Array operations are vectorised
- Math functions supported
- Can adopt memory

RDataFrame relies on RVec for treating collections

- Zero copy with adoption
- SBO makes functional approach performant

ROOT::RVec<T> In Action

```
RVec<double> mus_pt {15., 12., 10.6, 2.3, 4., 3.};
RVec<double> mus_eta {1.2, -0.2, 4.2, -5.3, 0.4, -2.};
RVec<double> good_mus_pt = mus_pt[mus_pt > 10 && abs(mus_eta) < 2.1];
```

Already integrated with RDataFrame

```
RVec<float> vals = {2.f, 5.5f, -2.f};
RVec<float> sin_vals = sin(vals);
```

```
ROOT::EnableImplicitMT();
RDataFrame f(treename, filename);
f.Define("good_pt", "sqrt(px*px + py*py)[E>100]")
 .Histo1D({"pt", "pt", 16, -.5, 3.5}, "good_pt")->Draw();
```

py is a collection, not a scalar

No templates: C++ → JIT → Python

C++

```
d.Filter([](double t) { return t > 0.; }, {"theta"})
 .Snapshot<vector<float>>("mytree","f.root",{"pt_x"});
```

C++ with cling's just-in-time compilation

```
d.Filter("theta > 0").Snapshot("mytree","f.root","pt_x");
```

PyROOT, automatically generated Python bindings

```
d.Filter("theta > 0").Snapshot("mytree","f.root","pt_x")
```

RDataFrame to numpy and pandas

```
# Run input pipeline with C++ performance that can process TBs of data, reads from remote, ...
import ROOT
df = ROOT.RDataFrame("tree", "file.root")
       .Filter("HLT_Mu22_v42", "Trigger requirement")
       .Filter("All(tight_iso)", "Quality cut")
       .Define("r", "sqrt(eta*eta + phi*phi)")

# Extract selection w/ defined variables as numpy arrays
col_dict = df.AsNumpy(["r", "eta", "phi"])

# Wrap data with pandas
import pandas
p = pandas.DataFrame(col_dict)
print(p)
```

```
   r     eta   phi
0  0.26  0.1  -0.5
1  1.0  -1.0   0.0
2  4.45  2.1   0.2
...
```

All the power of RDF + possibility to convert to NumPy

Keywords, Actions and Transformations

Transformations allow to modify the dataset

Transformation	**Description*
Define	Creates a new column in the dataset.
DefineSlot	Same as `Define`, but the user-defined function must take an extra `unsigned int slot` as its first parameter. `slot` will take a different value, `0` to `nThreads - 1`, for each thread of execution. This is meant as a helper in writing thread-safe `Define` transformation when using **`RDataFrame`** after **`ROOT::EnableImplicitMT()`**. `DefineSlot` works just as well with single-thread execution: in that case `slot` will always be `0`.
DefineSlotEntry	Same as `DefineSlot`, but the entry number is passed in addition to the slot number. This is meant as a helper in case some dependency on the entry number needs to be honoured.
Filter	Filter the rows of the dataset.
Range	Creates a node that filters entries based on range of entries

Lazy Actions 1/3

Lazy actions do not trigger the event loop

Lazy action	Description
Aggregate	Execute a user-defined accumulation operation on the processed column values.
Book	Book execution of a custom action using a user-defined helper object.
Cache	Caches in contiguous memory columns' entries. Custom columns can be cached as well, filtered entries are not cached. Users can specify which columns to save (default is all).
Count	Return the number of events processed.
Display	Obtains the events in the dataset for the requested columns. The method returns a RDisplay instance which can be queried to get a compressed tabular representation on the standard output or a complete representation as a string.
Fill	Fill a user-defined object with the values of the specified branches, as if by calling `Obj.Fill(branch1, branch2, ...).
Graph	Fills a **TGraph** with the two columns provided. If Multithread is enabled, the order of the points may not be the one expected, it is therefore suggested to sort if before drawing.
Histo{1D,2D,3D}	Fill a {one,two,three}-dimensional histogram with the processed branch values.

Lazy Actions 2/3

Max	Return the maximum of processed branch values. If the type of the column is inferred, the return type is `double`, the type of the column otherwise.
Mean	Return the mean of processed branch values.
Min	Return the minimum of processed branch values. If the type of the column is inferred, the return type is `double`, the type of the column otherwise.
Profile{1D,2D}	Fill a {one,two}-dimensional profile with the branch values that passed all filters.
Reduce	Reduce (e.g. sum, merge) entries using the function (lambda, functor...) passed as argument. The function must have signature `T(T,T)` where T is the type of the branch. Return the final result of the reduction operation. An optional parameter allows initialization of the result object to non-default values.
Report	Obtains statistics on how many entries have been accepted and rejected by the filters. See the section on named filters for a more detailed explanation. The method returns a RCutFlowReport instance which can be queried programmatically to get information about the effects of the individual cuts.
StdDev	Return the unbiased standard deviation of the processed branch values.

Lazy Actions 3/3

Sum	Return the sum of the values in the column. If the type of the column is inferred, the return type is `double`, the type of the column otherwise.
Take	Extract a column from the dataset as a collection of values. If the type of the column is a C-style array, the type stored in the return container is a **`ROOT::VecOps::RVec`**<T> to guarantee the lifetime of the data involved.

Instant actions do trigger the event loop

Instant action	Description
Foreach	Execute a user-defined function on each entry. Users are responsible for the thread-safety of this lambda when executing with implicit multi-threading enabled.
ForeachSlot	Same as `Foreach`, but the user-defined function must take an extra `unsigned int slot` as its first parameter. `slot` will take a different value, `0` to `nThreads - 1`, for each thread of execution. This is meant as a helper in writing thread-safe `Foreach` actions when using **`RDataFrame`** after **`ROOT::EnableImplicitMT()`**. `ForeachSlot` works just as well with single-thread execution: in that case `slot` will always be `0`.
Snapshot	Writes processed data-set to disk, in a new **`TTree`** and **`TFile`**. Custom columns can be saved as well, filtered entries are not saved. Users can specify which columns to save (default is all). Snapshot, by default, overwrites the output file if it already exists. `Snapshot` can be made *lazy* setting the appropriate flage in the snapshot options.