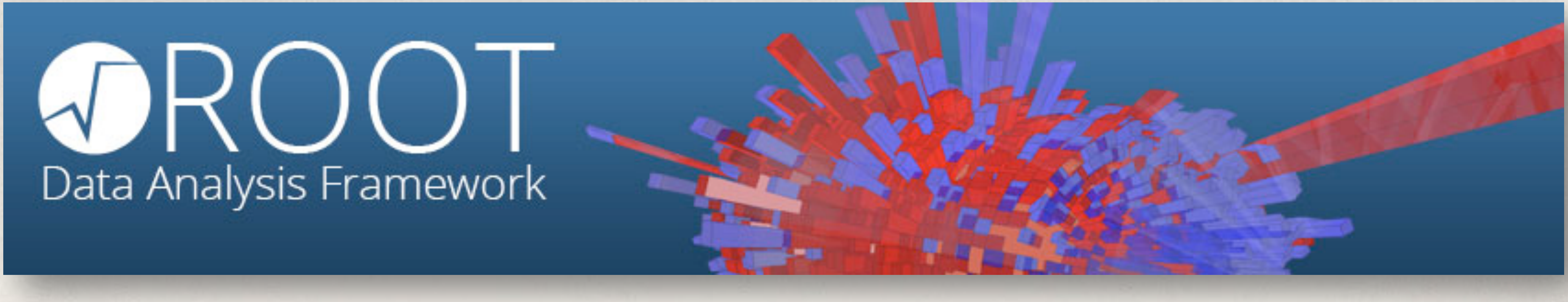


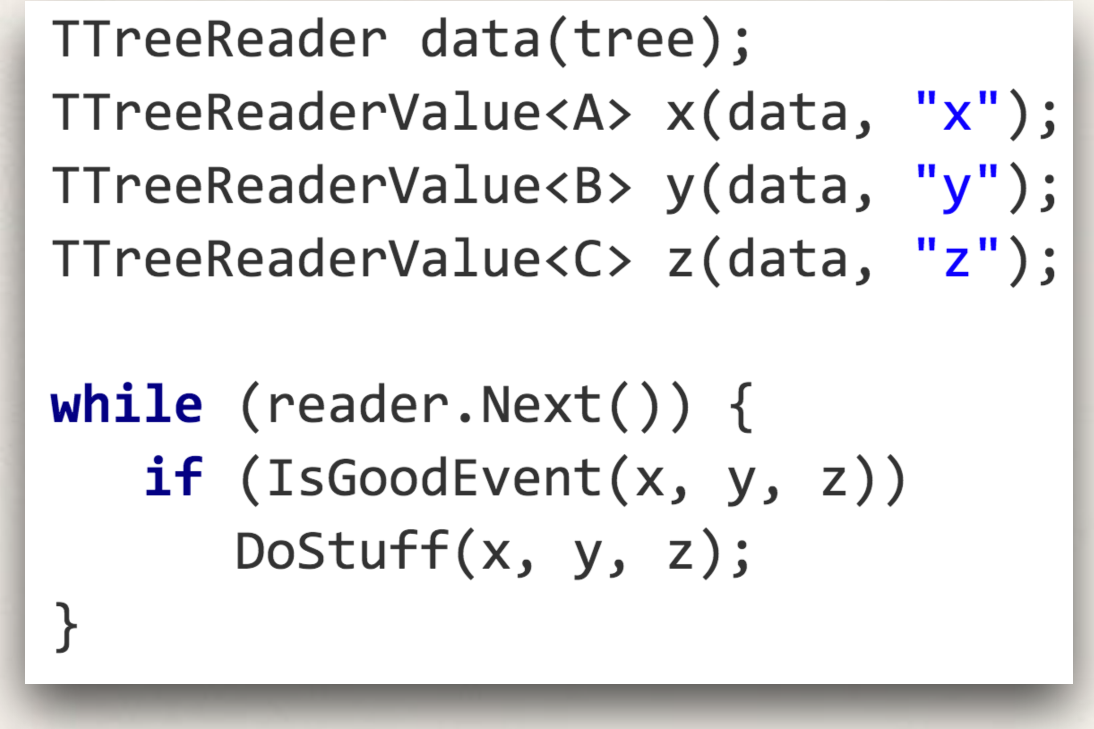
Parallel and Functional based Analysis in ROOT (TDataFrame)

Pere Mato on behalf of the ROOT Team

TDataFrame

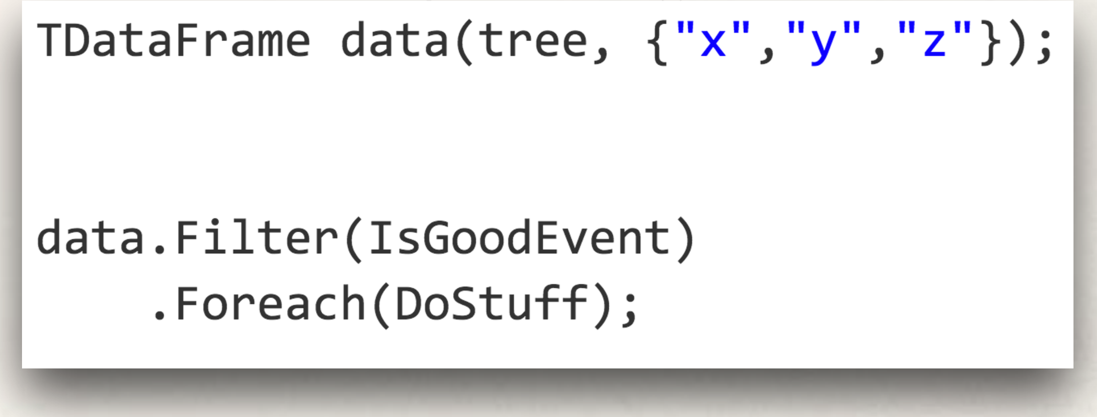
- New way to interact with ROOT columnar format
  - Inspiration taken from widely spread tools such as Pandas or Spark
  - Concept proposed earlier (e.g. LINQToROOT by G. Watts)
- Analysis expressed as a chain of **transformations** and **actions**
  - Transformation: filter, add a column, …
  - Actions: Fill an histo, a profile, count events, …
- The user specifies the **What** and ROOT chooses the **How**
  - Computation is only triggered at the end of the chain having the full knowledge of what the user wants to do
  - Great opportunity for optimizations (partitioning, caching, re-ordering, parallelization, etc.)

Controlling the Loop



```
TTreeReader data(tree);
TTreeReaderValue<A> x(data, "x");
TTreeReaderValue<B> y(data, "y");
TTreeReaderValue<C> z(data, "z");

while (reader.Next()) {
   if (IsGoodEvent(x, y, z))
      DoStuff(x, y, z);
}
```



```
TDataFrame data(tree, {"x","y","z"});


data.Filter(IsGoodEvent)
    .Foreach(DoStuff);
```

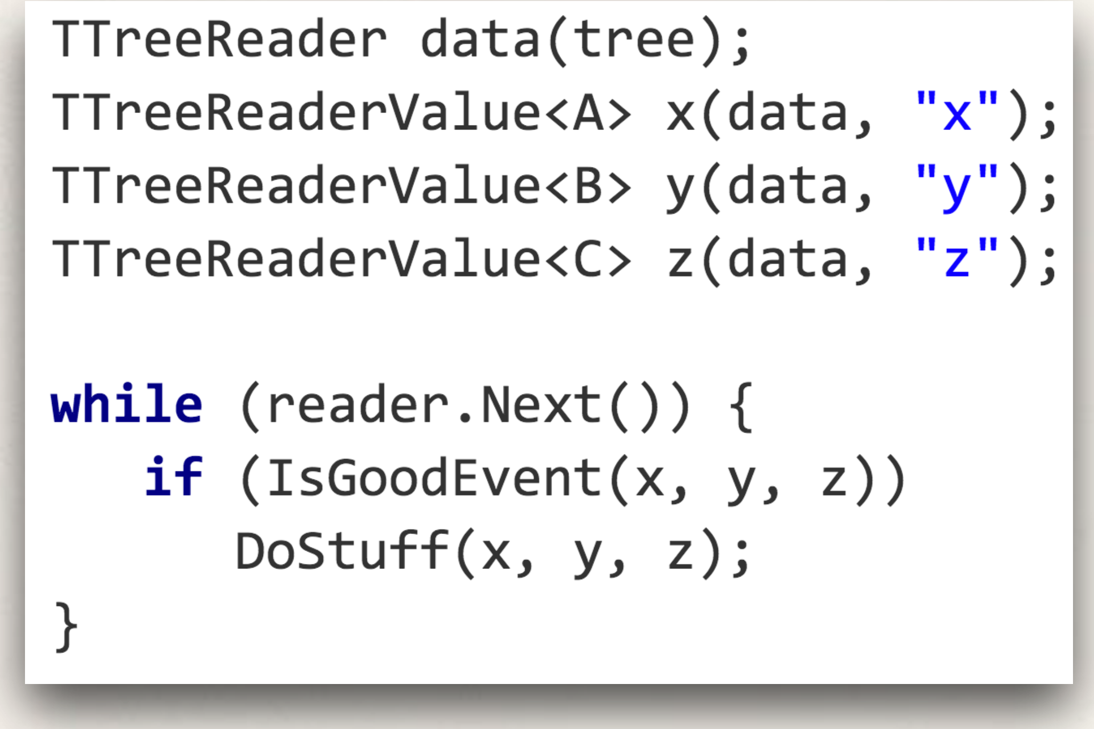
(+) The current interface the user has full control of the event-loop

(-) needs some boilerplate

(-) running the event-loop in parallel is not trivial

(-) users implement trivial operations again and again

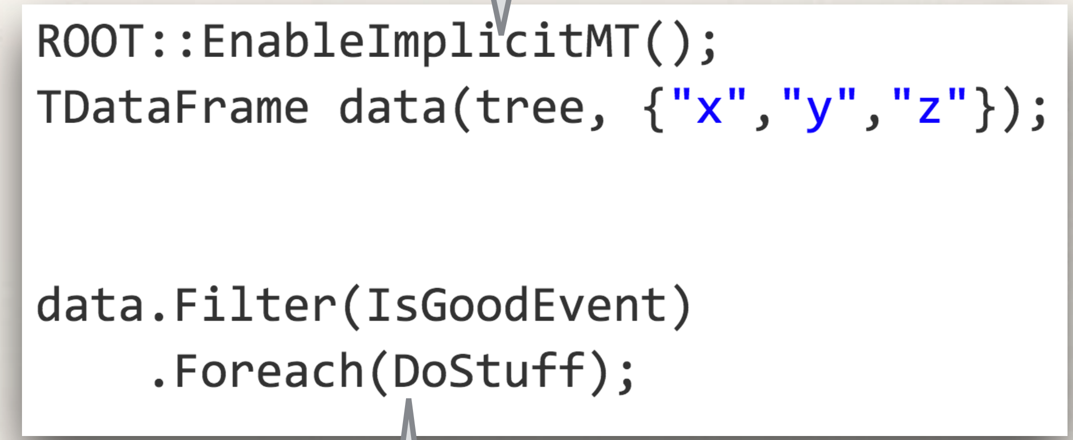
Controlling the Loop



```
TTreeReader data(tree);
TTreeReaderValue<A> x(data, "x");
TTreeReaderValue<B> y(data, "y");
TTreeReaderValue<C> z(data, "z");

while (reader.Next()) {
   if (IsGoodEvent(x, y, z))
      DoStuff(x, y, z);
}
```

ROOT will parallelize the operations.



```
ROOT::EnableImplicitMT();
TDataFrame data(tree, {"x","y","z"});


data.Filter(IsGoodEvent)
    .Foreach(DoStuff);
```

(!) A thread safe DoStuff needed

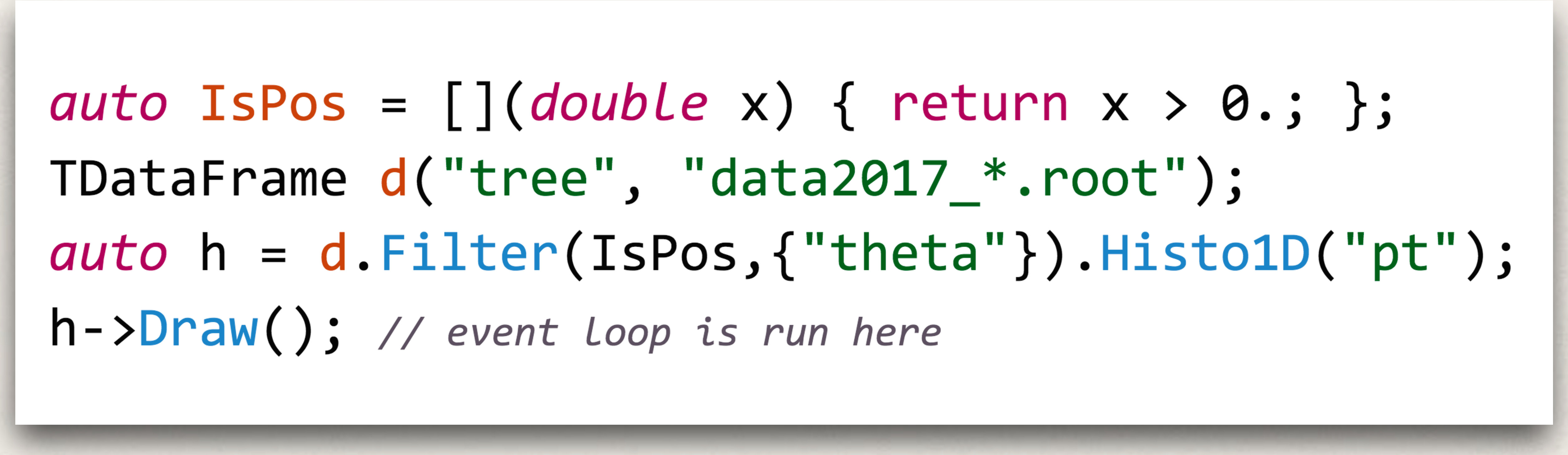
(+) The current interface the user has full control of the event-loop

(-) needs some boilerplate

(-) running the event-loop in parallel is not trivial

(-) users implement trivial operations again and again

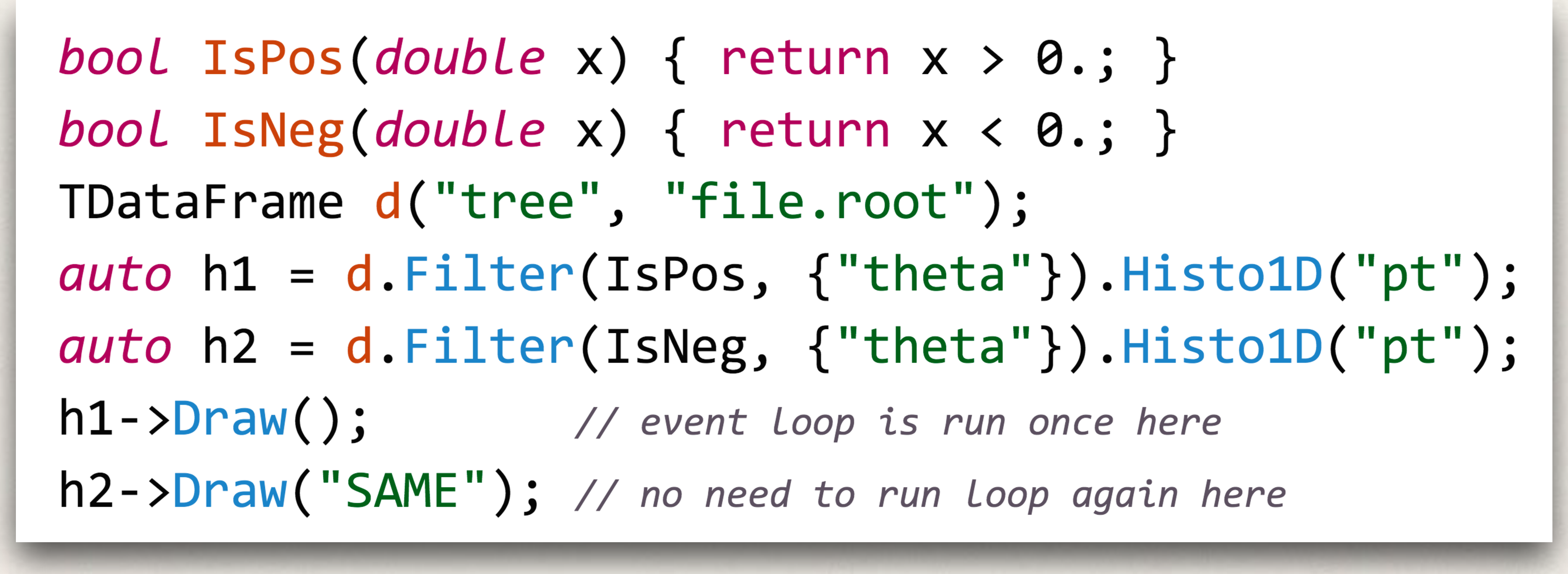
Example: Cut and Fill



```
auto IsPos = [](double x) { return x > 0.; };
TDataFrame d("tree", "data2017_*.root");
auto h = d.Filter(IsPos,{"theta"}).Histo1D("pt");
h->Draw(); // event loop is run here
```

- Event-loop is run lazily, upon first access to the results

Example: Cut and Fill

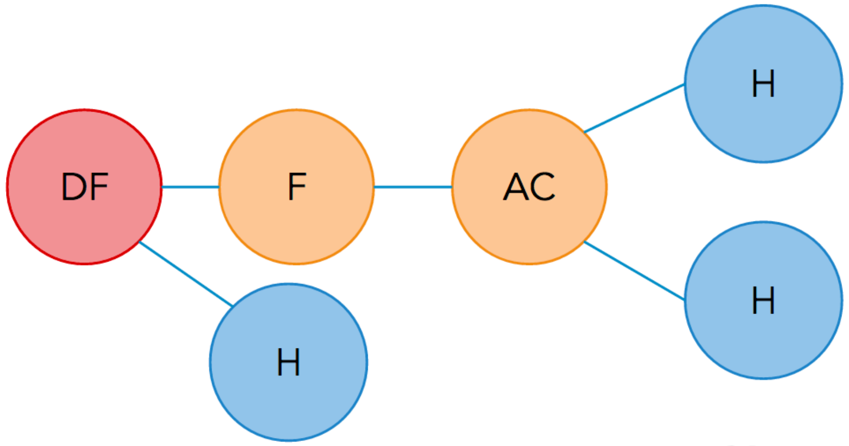


```
bool IsPos(double x) { return x > 0.; }
bool IsNeg(double x) { return x < 0.; }
TDataFrame d("tree", "file.root");
auto h1 = d.Filter(IsPos, {"theta"}).Histo1D("pt");
auto h2 = d.Filter(IsNeg, {"theta"}).Histo1D("pt");
h1->Draw();         // event loop is run once here
h2->Draw("SAME"); // no need to run loop again here
```

- All actions are executed in the same event-loop

Example: Branching

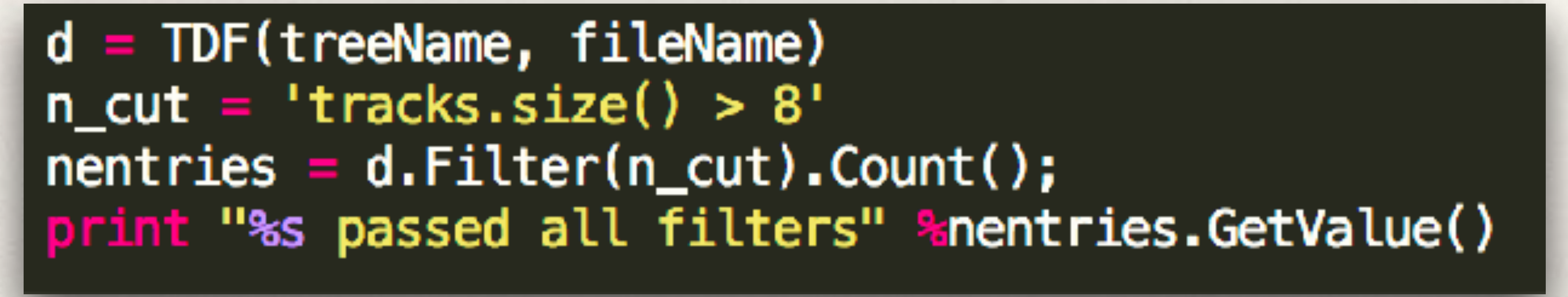
```
TDataFrame d("tree", "file.root", {"x"});
// store a filtered data-frame with a new column
auto f = d.Filter([](double a) { return a > 0.})
          .AddColumn("z", Sum, {"x","y"});
// make multiple histograms out of it
auto hz = f.Histo1D("z");
auto hxy = f.Histo2D("x","y");
auto hy = d.Profile1D("x","y");
```



- Not just functional chains but functional *graphs*

Implementation Status

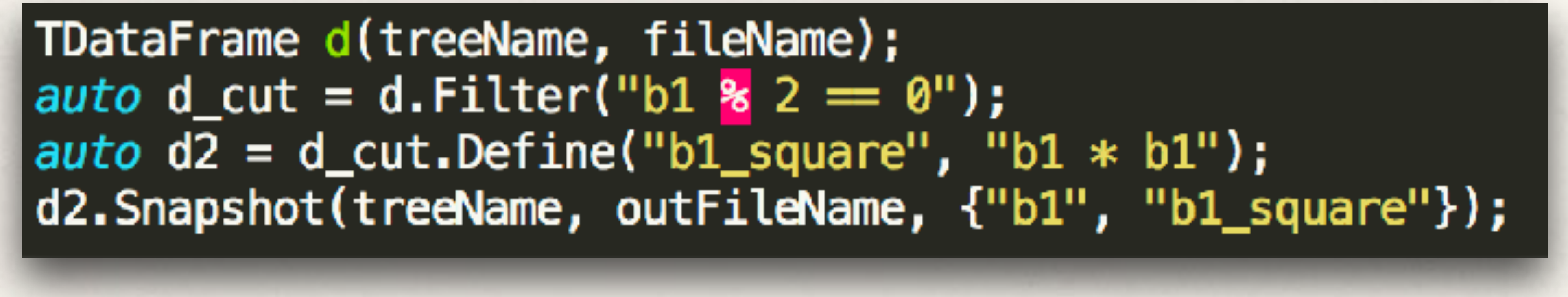
- A type safe approach with everything templated
  - E.g. usage of TTreeReader
- Good performance
  - 0 copies
  - read-decompress-deserialise only what is needed
  - as many actions as the user specifies in a single event loop
- Sophisticated usage of Jitting behind the scenes possible
  - Slimmer programming model accessible: user must not specify column types, cling can jit the right template arguments
  - Write expressions for new columns or filters as strings written in C++



```
d = TDF(treeName, fileName)
n_cut = 'tracks.size() > 8'
nentries = d.Filter(n_cut).Count();
print "%s passed all filters" %nentries.GetValue()
```

Implementation Status (2)

- Python interface via PyROOT with jitted C++
- Write out **transformed** dataset in ROOT format, implicit parallelism can be activated here too
    - Write a tree from different threads (no one file per thread only!)



```
TDataFrame d(treeName, fileName);
auto d_cut = d.Filter("b1 % 2 == 0");
auto d2 = d_cut.Define("b1_square", "b1 * b1");
d2.Snapshot(treeName, outFileName, {"b1", "b1_square"});
```

- Plan to read other formats too (Parquet, CSV, SQL …)
- Benchmarking of TDataFrame and its SnapShot capabilities being performed now on desktops, Xeon bleeding edge servers and KNL
- All this will be in ROOT 6.10