Enhancements to declarative analysis in ROOT

Massimo Tumolo

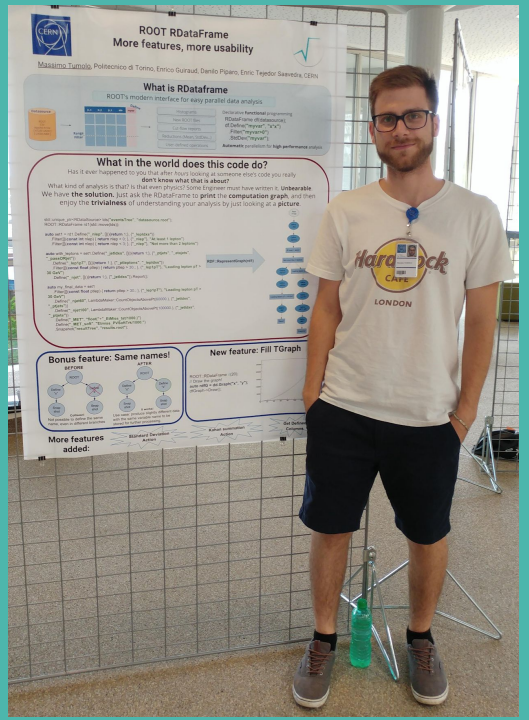
August 2018

About me

- MSc. in Computer Engineering
  - 39 days left!
- Summer Student for 12 weeks
  - Supervised by Danilo and Enric
  - ...Frequently bothering Enrico with tons of questions
- I **love** pizza

My activity at CERN

- ➔ Enhancements to declarative analysis in ROOT
  - ◆ Fancy name
- ➔ **Improving** the **RDataFrame** usability
  - ◆ By adding more features



Background: RDataFrame

RDataFrame

Declarative **functional** programming

Automatic parallelism for **high performance** analysis

```
RDataFrame df( /* ... */ );

auto d = df.Define("x", [ ]( ){ /* ... */ });

auto f1=Filter("x>0");
auto f2=Filter("x<0");

f1.Mean("x");
f2.Mean("x");
```

Chapter 1:
Drawing the computation graph

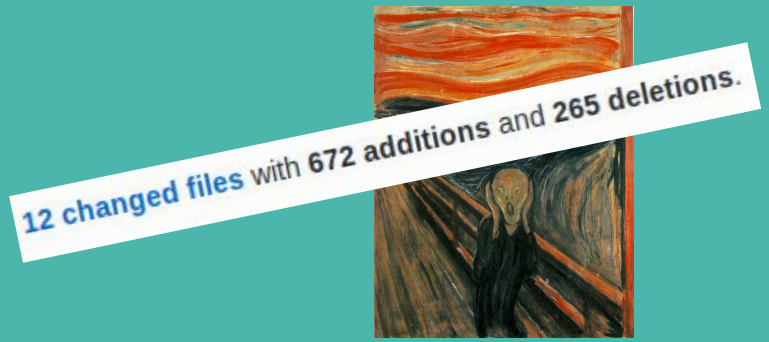
Drawing the computation graph

- Fairly **simple task...**
  - Just a warm up, the best is yet to come
- **...Undoable**!
  - *Define* nodes do not exists in the graph.
  - A *Define* is just a push_back in a vector of columns.
  - How to draw *Defines* if there are no *Defines*?

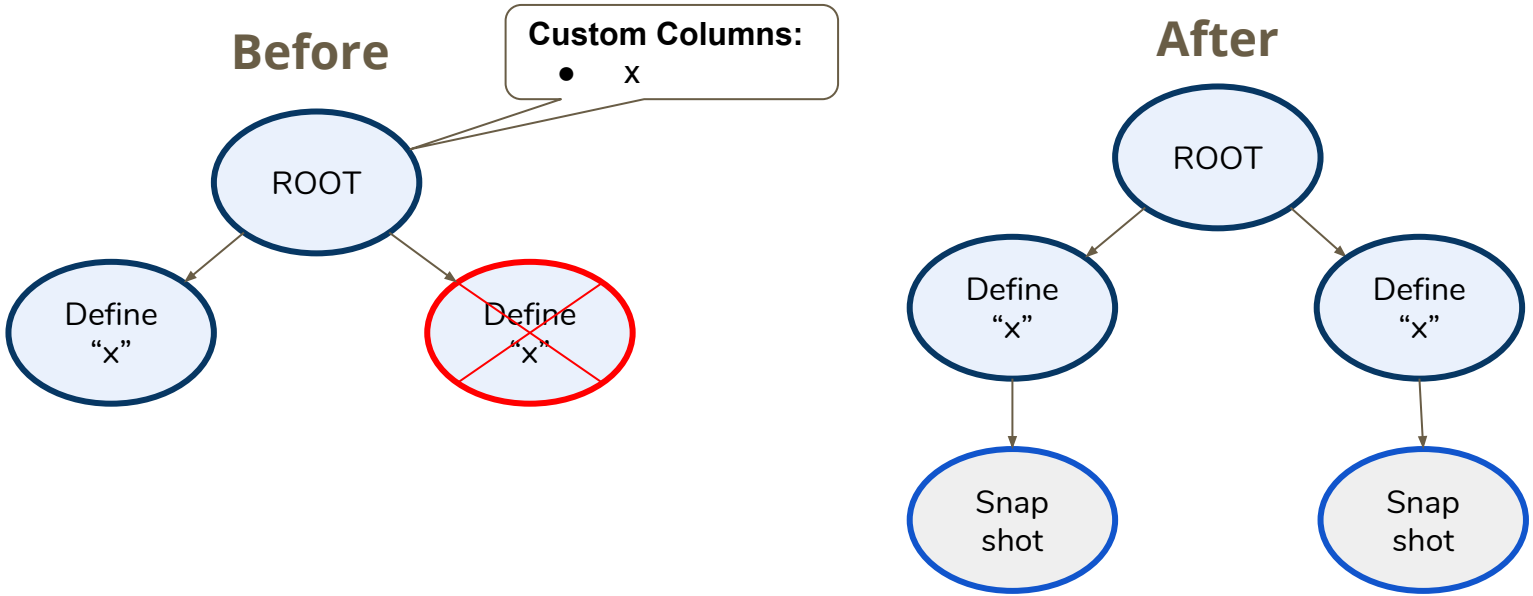
Drawing the computation graph

- But, behind the hood
  - The RDataframe lacked a feature.
  - The user **could not define the same custom columns** in multiple branches
    - Even if branches are independent!

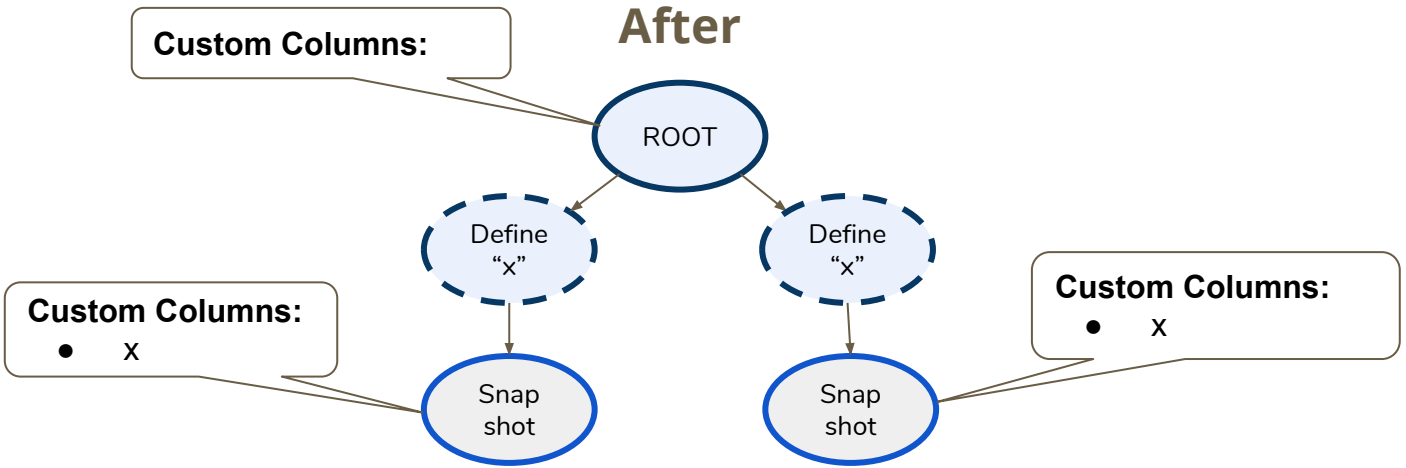
Chapter 0: Custom columns with the same name



Custom columns with same name



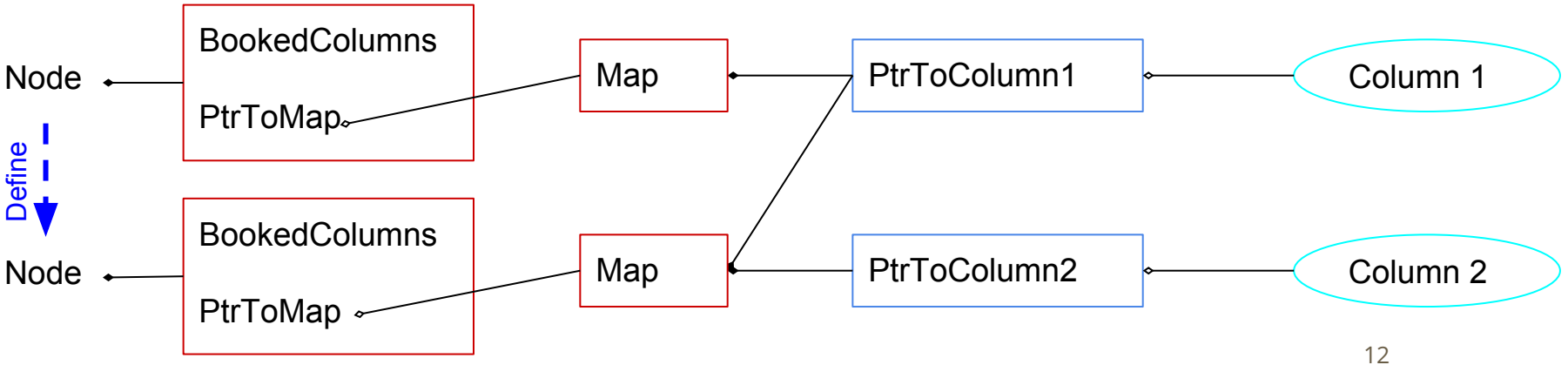
Custom columns with same name



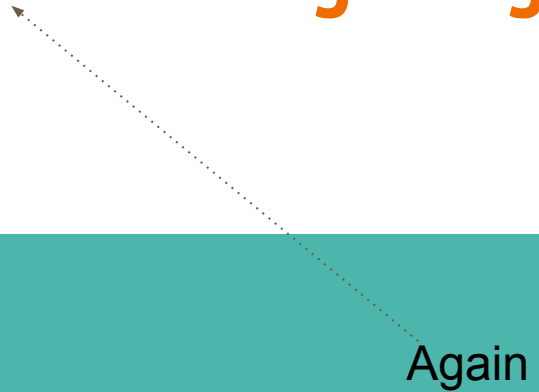
Custom columns with the same name

Question:
Nice feature! But how much copies are you doing and how memory are you wasting?

Answer:
Almost the same as before.



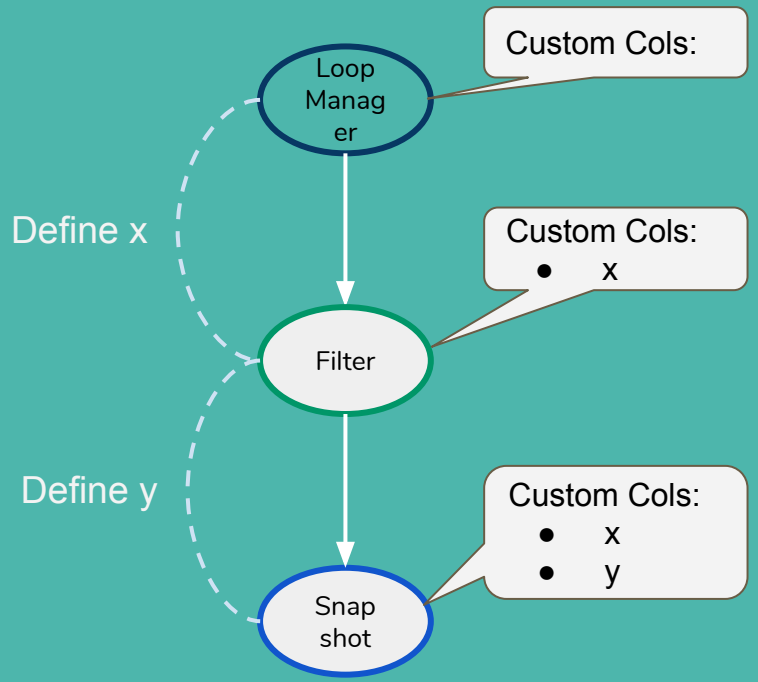
Chapter 1: Drawing the graph



Drawing the graph

- All information are available!
- Every node knows the columns defined up to it.
- If a node knows a column and the previous doesn't, a *Define* happened in between.

Internals



Drawing the Graph

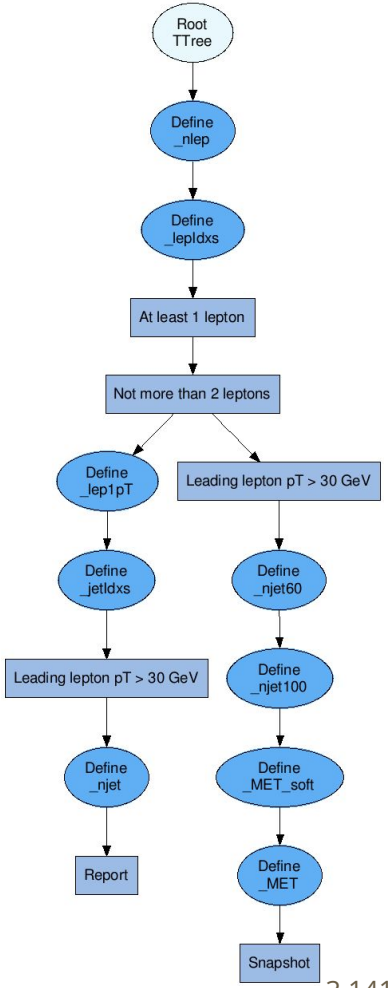
From raw hard-to-understand code...

```
std::unique_ptr<RDataSource> tds("eventsTree", "datasource.root");
ROOT::RDataFrame rd1(std::move(tds));

auto set1 = rd1.Define("_nlep", [](){return 1;}, {"_lepIdxs"})
    .Filter([](const int nlep) { return nlep > 0; }, {"_nlep"}, "At least 1 lepton")
    .Filter([](const int nlep) { return nlep < 3; }, {"_nlep"}, "Not more than 2
leptons")

auto with_leptons = set1.Define("_jetIdxs", [](){return 1;}, {"_ptjets",
"_etajets", "_passORjet"})
    .Define("_lep1pT", [](){return 1;}, {"_ptleptons", "_lepIdxs"})
    .Filter([](const float ptlep) { return ptlep > 30.; }, {"_lep1pT"}, "Leading
lepton pT > 30 GeV")
    .Define("_njet", [](){return 1;}, {"_jetIdxs"}).Report();

auto my_final_data = set1
    .Filter([](const float ptlep) { return ptlep > 30.; }, {"_lep1pT"}, "Leading
lepton pT > 30 GeV")
    .Define("_njet60", LambdaMaker::CountObjectsAbovePt(60000.),
{"_jetIdxs", "_ptjets"})
    .Define("_njet100", LambdaMaker::CountObjectsAbovePt(100000.),
{"_jetIdxs", "_ptjets"})
    .Define("_MET", "float("+"_EtMiss_tst/1000.)")
    .Define("_MET_soft", "Etmiss_PVSoftTrk/1000.")
    .Snapshot("resultTree", "results.root");
```

Drawing the Graph

… to a **visual** representation, easy to understand.

- If the root node is provided, the entire graph is printed.
- For any other node, only the branch they belong to.



Chapters 0 and 1:
Why not in the master?

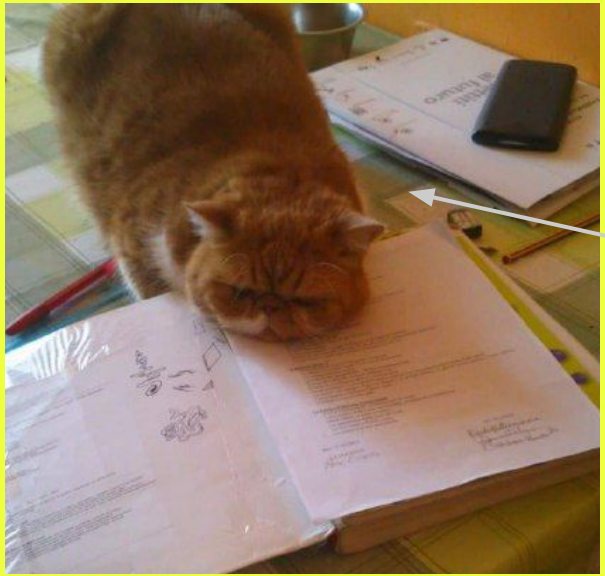
phsft-bot commented 21 hours ago — Member

Build failed on slc6-i686/gcc49.
See console output.

Failing tests:

- projectroot.tree.dataframe.test.gtest_tree_dataframe_test_dataframe_callbacks
- projectroot.tree.dataframe.test.gtest_tree_dataframe_test_dataframe_simple
- projectroot.roottest.root.dataframe.roottest_root_dataframe_test_inference

😕 1



Ned

Yuma

Still Following?

Chapter 2:
Standard Deviation

Standard Deviation

- Introduced the StdDev() action in the dataframe.
  - Based on the Welford's algorithm
  - Supports multithreaded execution
- Tested against the standard deviation canonical definition.
- In **master**!

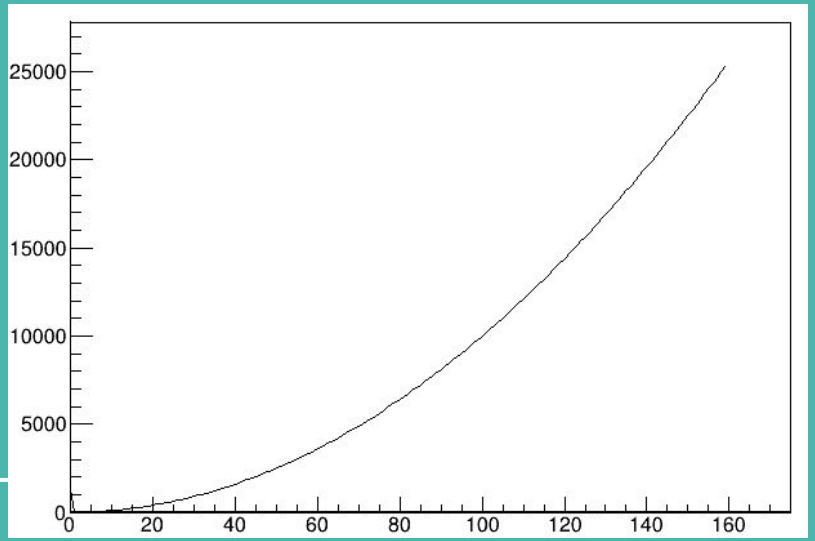
```
RDataFrame df( /* ... */ );
auto stdDev =
df.Define("x", [ ]( ){ /* ... */ })
.StdDev("x");
```

Chapter 3:
Filling a TGraph

Filling a TGraph

- Introduced the Graph() action in the RDataframe.
- Given two columns, a TGraph is built.
- If filled by multiple threads, the order of points can’t be foreseen.
- In **master!**

```
ROOT::RDataFrame d(20);
// Draw the graph!
auto rdfG = dd.Graph("x", "y");
dfGraph->Draw();
```



Chapter 4:
Kahan Summation

Kahan Summation

- Introduced a Kahan summation
  - To avoid losing precision while using floating point arithmetic.
- Not among the default actions, but provided as a tutorial to who might need it.
- In **master!**

```
KahanSum<double> kahan();

auto kahanResult =
dd.Book<double>(std::move(kaha
n), {"x1"});

std::cout << "Kahan: "
          << *kahanResult
          << " Classical: "
          << standardSum;

// Outputs: Kahan:
1000000000.000000011920929
Classical: 1000000000
```

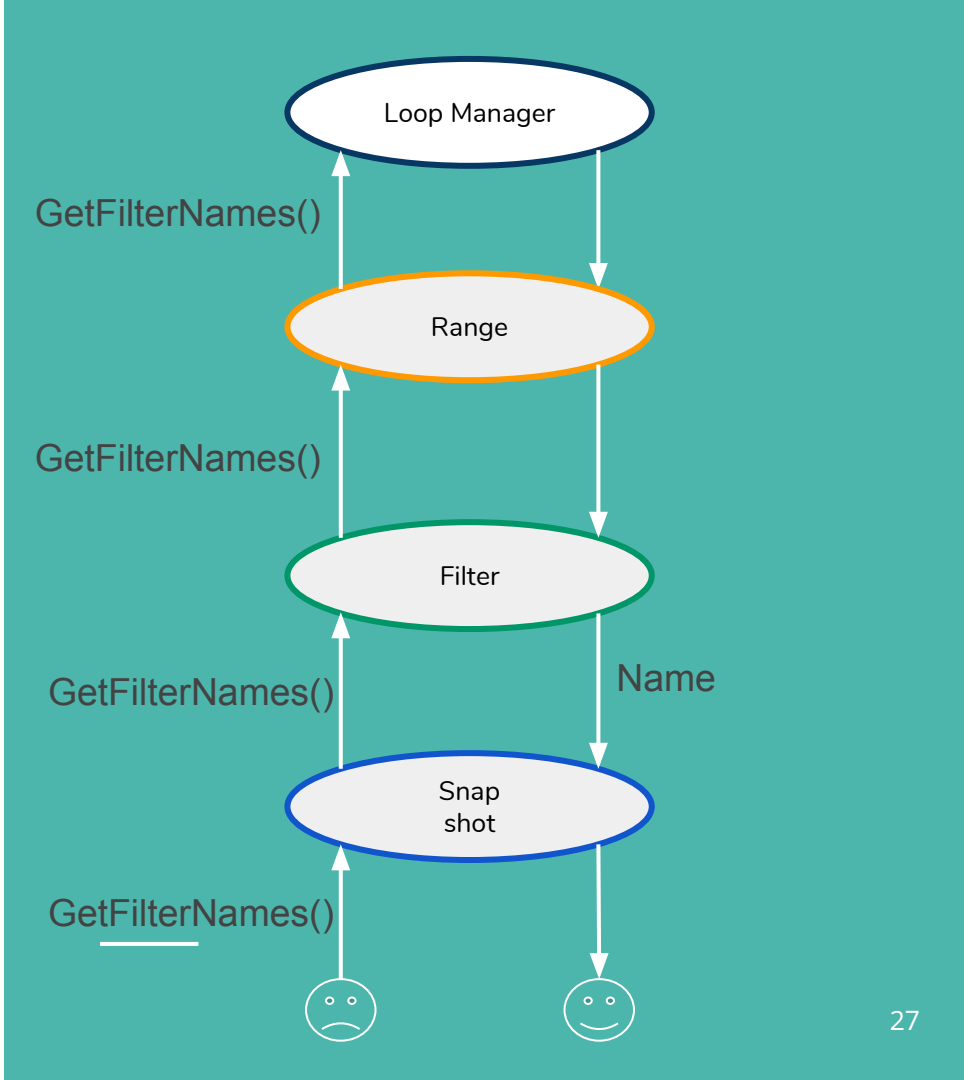
Chapter 5:
Get Defined Columns and Get Filters

Get Defined Columns and Get Filters

- Given any node, the user can retrieve the filter or column defined up to it.
- Given a root node, the user can retrieve any filter defined.
- Utility offered to the user, no real computation performed.

Get Filters

- Every node has a GetFilterNames() method
- Only the *Filters* do something with the method



Chapter 6:
Visitor pattern

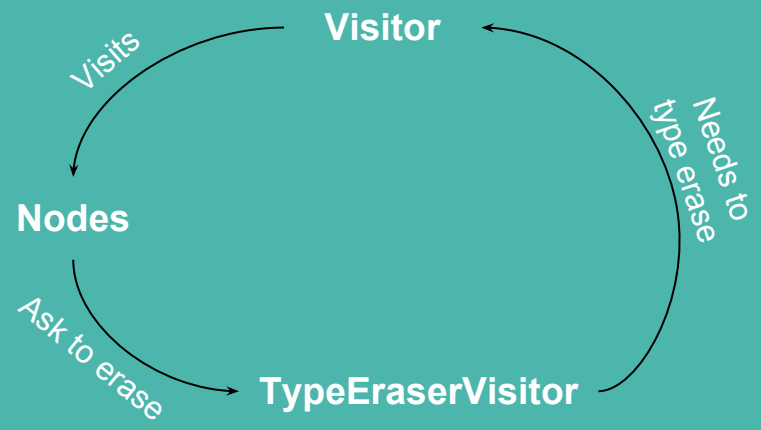
Or.. Let's rewrite chapter 5!

Visitor pattern

Expectations

```
AnyVisitor visitor;

AnyNode.subscribe(visitor);
```

Reality

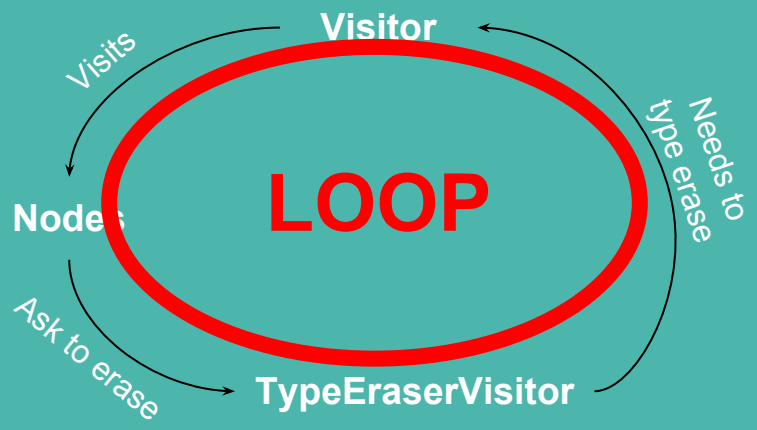


Visitor pattern

Expectations

```
AnyVisitor visitor;

AnyNode.subscribe(visitor);
```

Reality

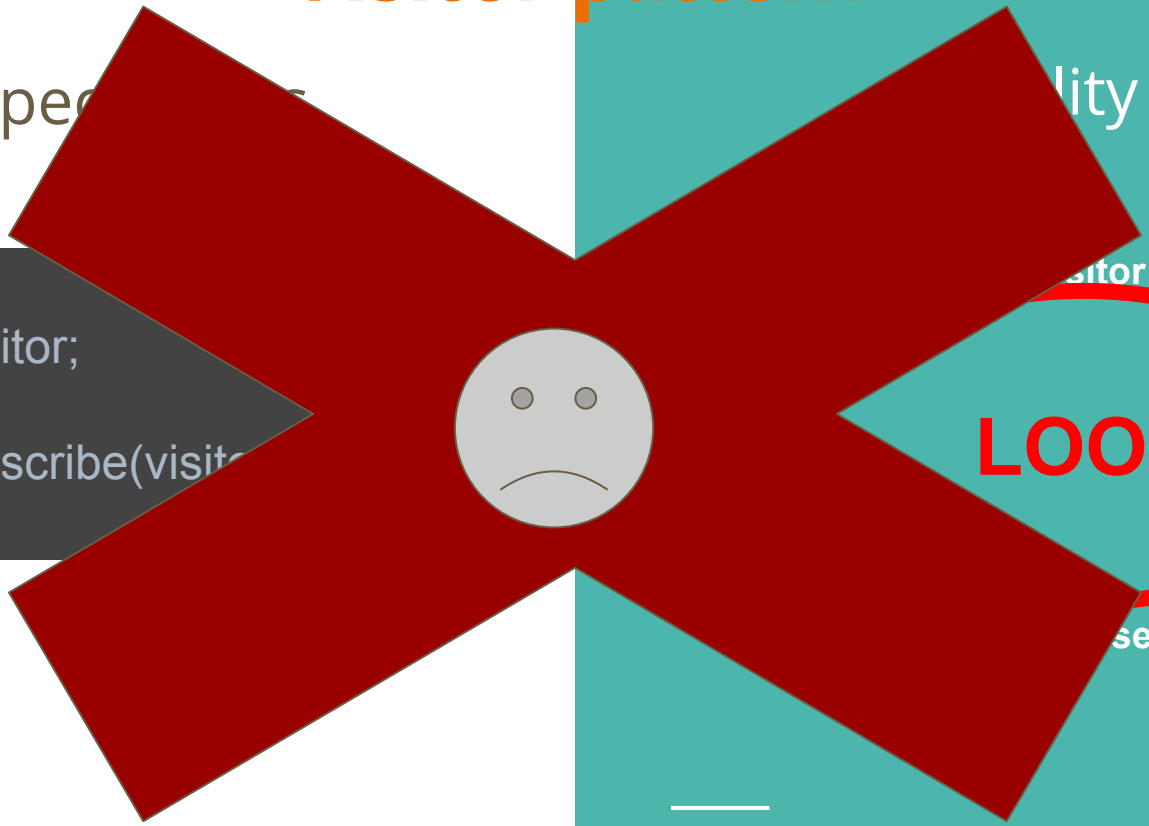


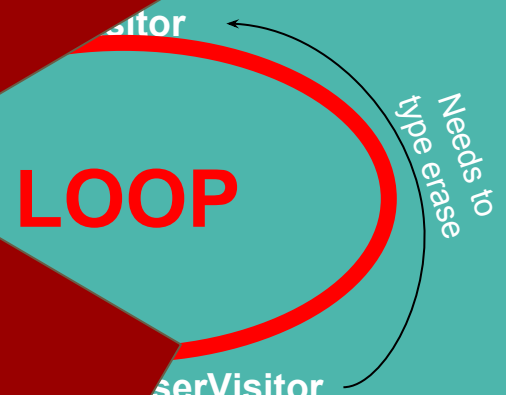
Visitor pattern

Expec

lity

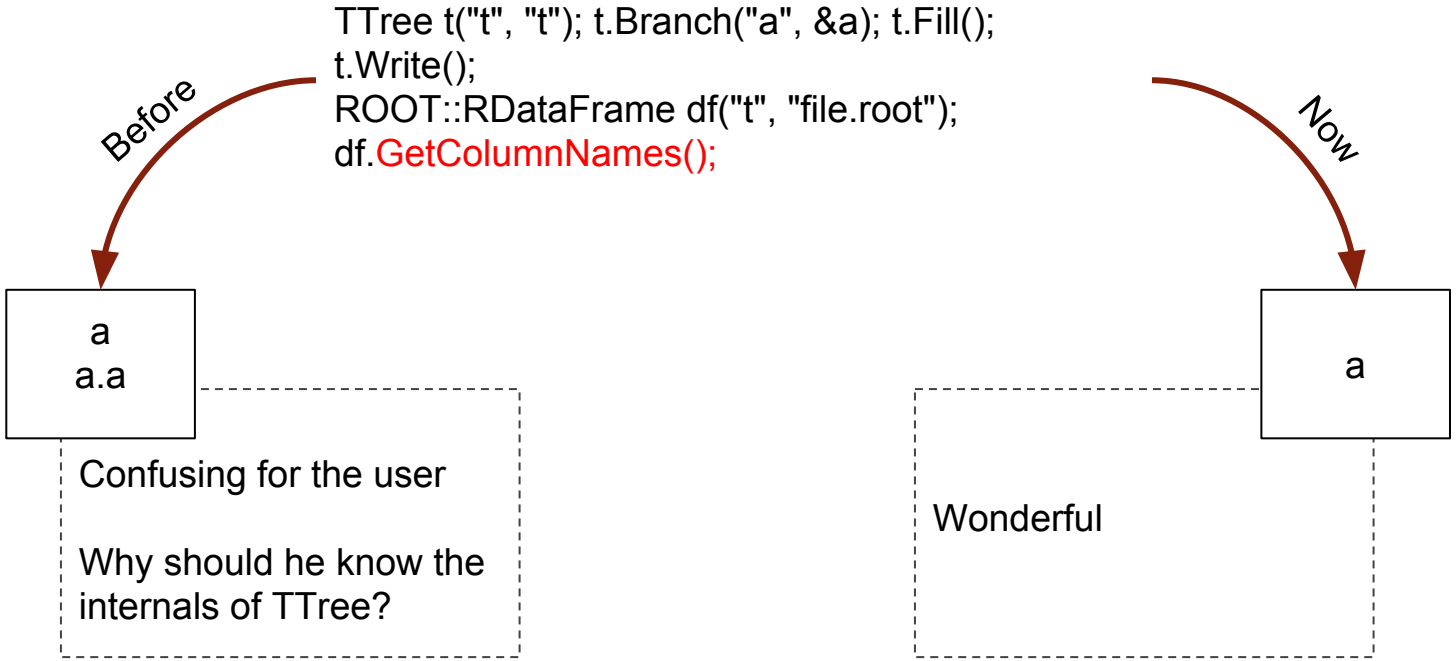
```
AnyVisitor visitor;

AnyNode.subscribe(visit
```





Chapter 7:
Improving GetColumnNames

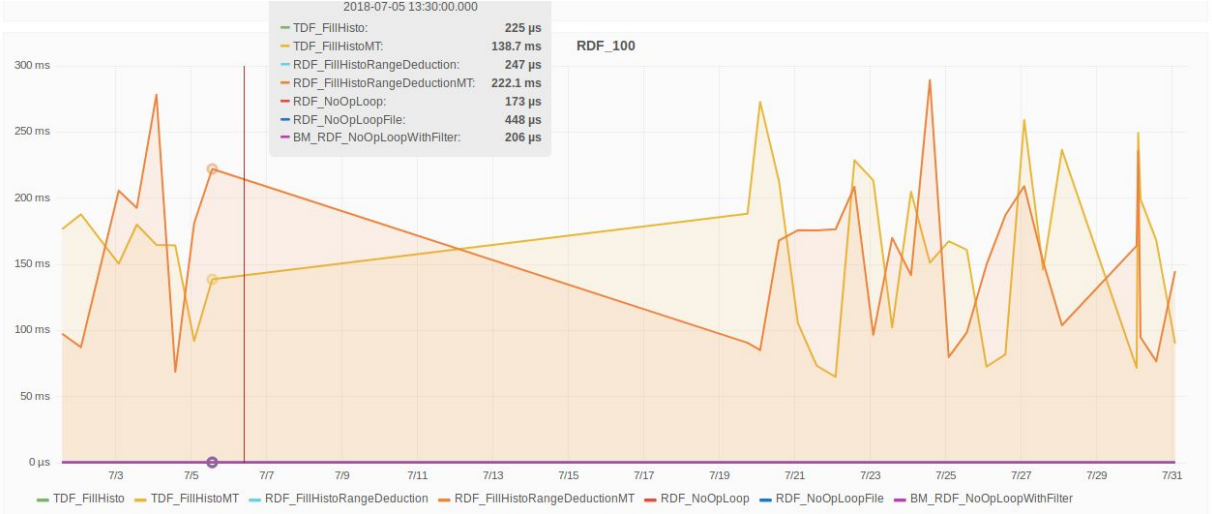
Improving GetColumnNames



Chapter 8:
Upgrade of the Graphana Board

Upgrade of the Graphana Board

- Fix broken benchmarks
- Added graphs to compare the operations performance with a fixed amount of events



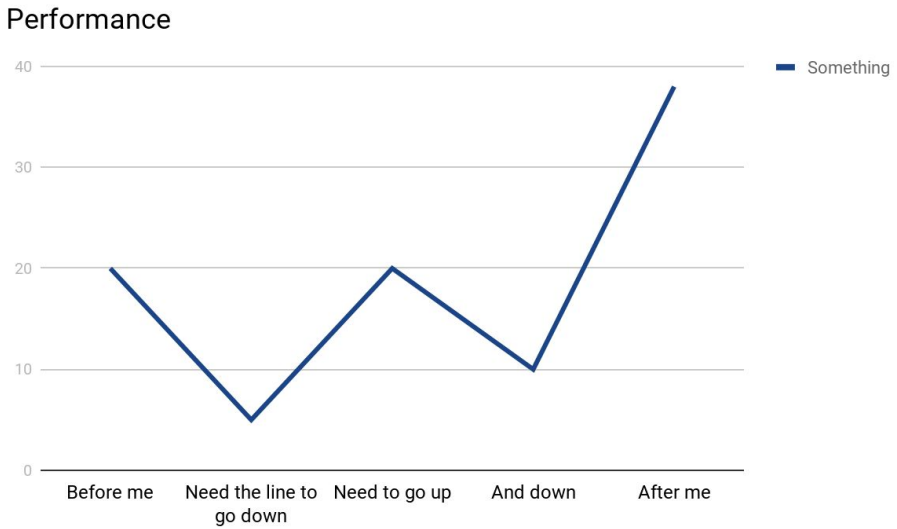
Annex A:
What's next?

Display

- Display the content of the columns.
- Still choosing the interface offered to the user
  - Display and Head action?
  - Multithreaded allowed?
- PoC available in my fork

```
RDataFrame rdf(datasource);
*rdf.Display();

// Outputs:
X |   Y   | Z
1 |  200 | 3.1
1 |  2     | 4.2
```

Benchmarks



... I have not started, yet

Thank you for your attention

And thank you to Danilo, Enric, and Enrico for giving me this opportunity and for teaching me an insane amount of things.