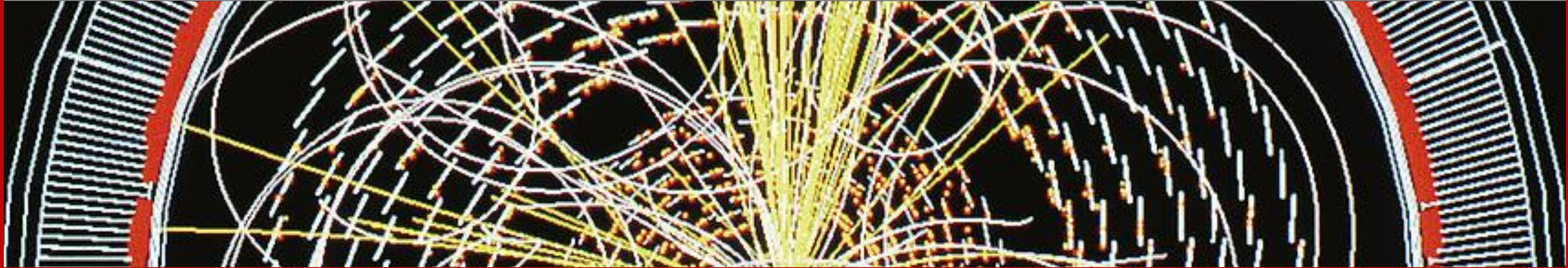
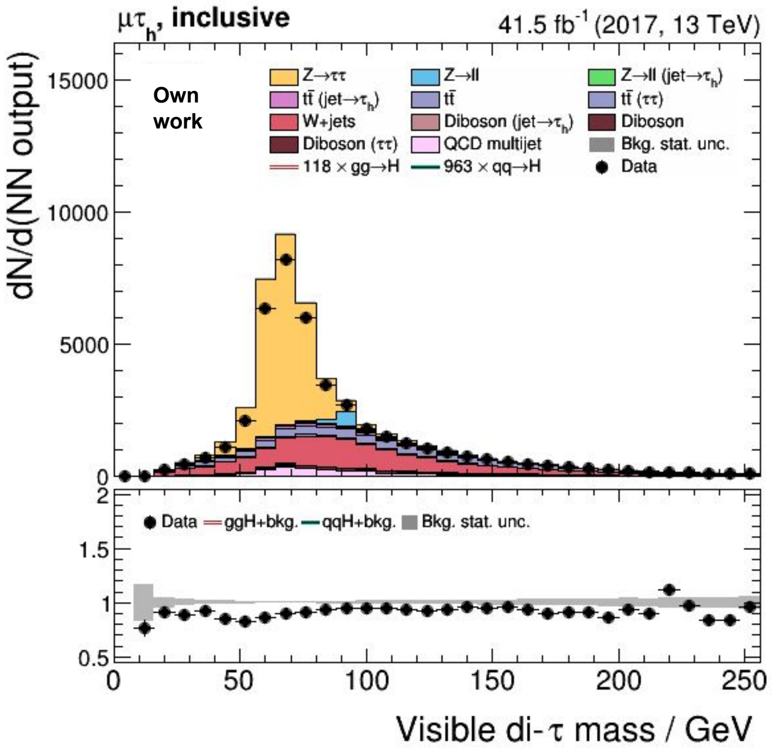
Benchmarking an RDataFrame Complex Analysis

Massimiliano Galli

massimiliano.galli@cern.ch



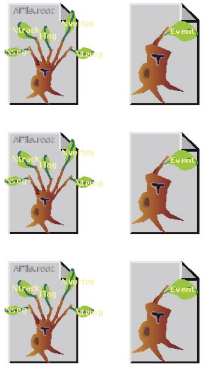
Recap - Motivation



- CMS search for Higgs decay into tau-tau final state
- Full Run 2 analysis
- Production of many plots with data and simulations of signal and various background contributions
- Obtain the same results in a faster and more efficient way using modern ROOT facilities (`RDataFrame` vs `TTree::Draw()`)

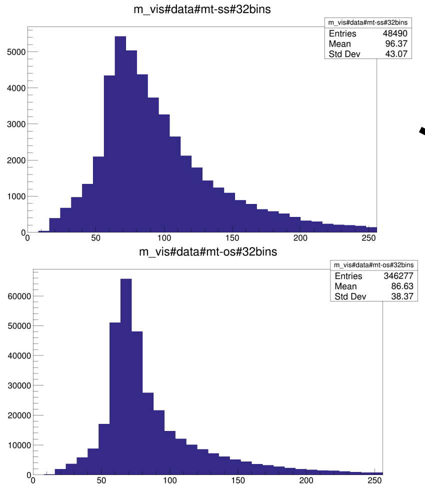
Recap - Orders of Magnitude

NTuples



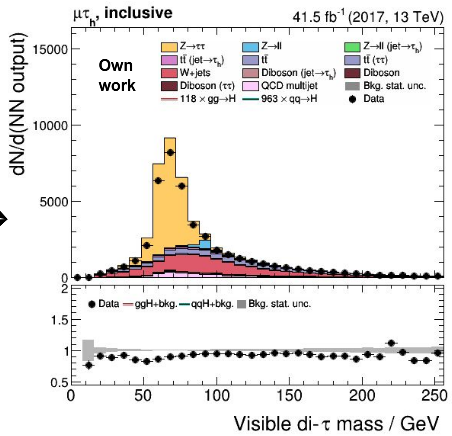
1.1 Tb (main NTuples)
120 Gb (friends)

Histograms



~ Mb

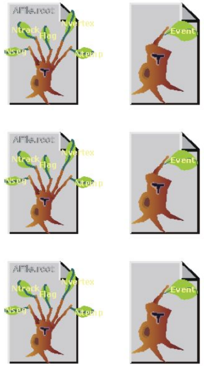
Stack Plots



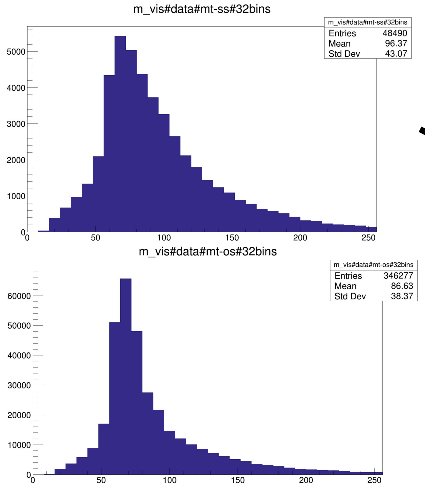
~ Mb

Recap - Orders of Magnitude

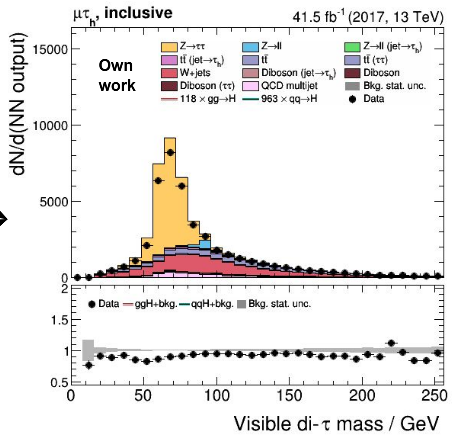
NTuples



Histograms



Stack Plots



~100 ROOT files,
100-150 TTrees
each

~ 3.6 * 10^5
histograms

~ 100 stack plots

Programming Model

Book Results

For every histogram that want to produce we declare initial dataset, cuts, weights and systematic variations that we want to apply

Optimize Computations

Datasets, selections and histogram productions are treated as nodes of a graph. The common ones are merged to perform every action only once

Run Computations

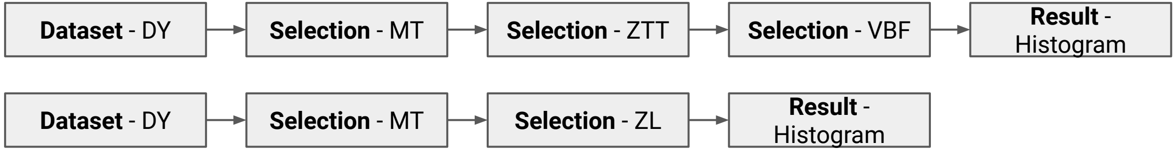
The previous graphs are converted to the language of RDataFrame and the event loop is run

Programming Model - 1. Book Results

Book Results

For every histogram that want to produce we declare initial dataset, cuts, weights and systematic variations that we want to apply

```
Unit(dataset=dy, selections=[mt, ztt, vbf], histo)
Unit(dataset=dy, selections=[mt, zl], histo)
```



Programming Model - 1. Book Results

Book Results

For every histogram that want to produce we declare initial dataset, cuts, weights and systematic variations that we want to apply

Types of systematic variations:

- ChangeDataset

- AddWeight
- ReplaceWeight
- SquareWeight
- RemoveWeight

- AddCut
- RemoveCut

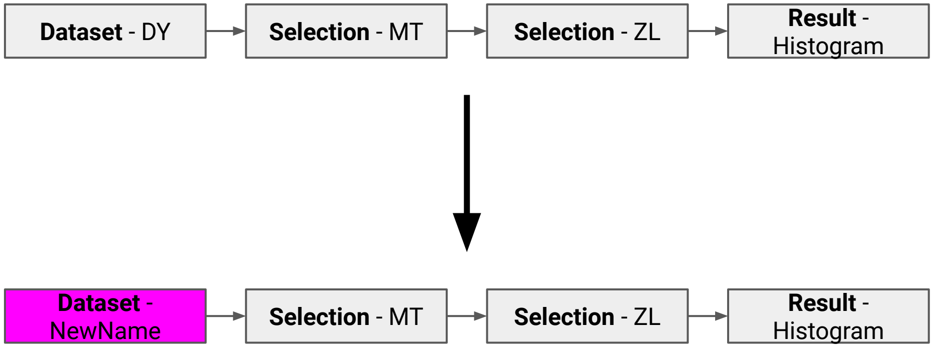
Programming Model - 1. Book Results

Book Results

For every histogram that want to produce we declare initial dataset, cuts, weights and systematic variations that we want to apply

Ex: ChangeDataset

```
var = ChangeDataset("NewName", "NewDirectory")
um.book([zl_unit], [var])
```



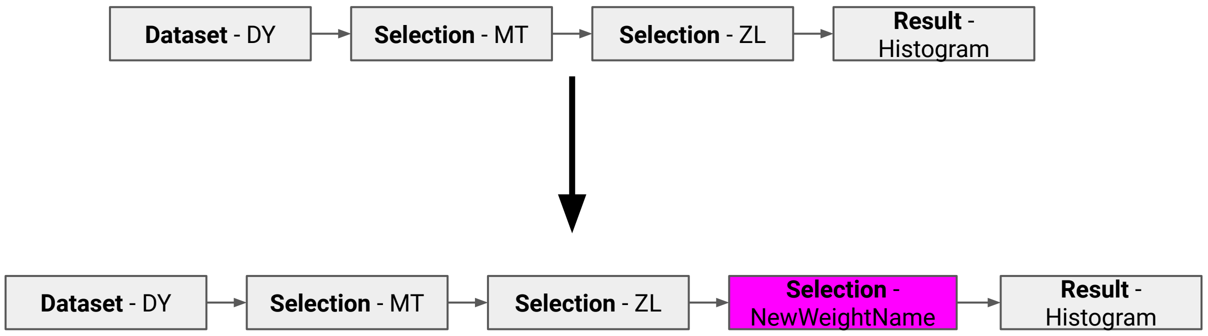
Programming Model - 1. Book Results

Book Results

For every histogram that want to produce we declare initial dataset, cuts, weights and systematic variations that we want to apply

Ex: AddWeight

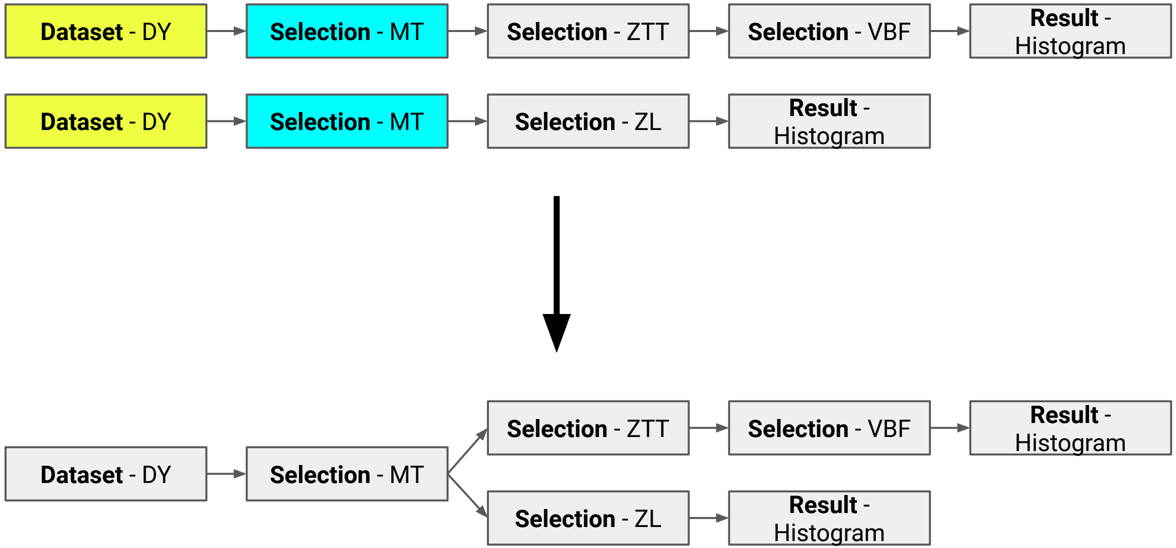
```
var = AddWeight("VariationName", Weight("NewWeightExp",
"NewWeightName"))
um.book([zl_unit], [var])
```



Programming Model - 2. Optimize Computations

Optimize Computations

Datasets, selections and histogram productions are treated as nodes of a graph. The common ones are merged to perform every action only once



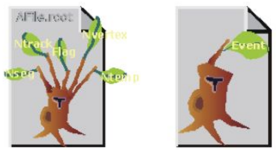
Programming Model - 3. Run Computations

Run Computations

The previous graphs are converted to the language of RDataFrame and the event loop is run

- One RDataFrame for each node of type ‘dataset’
- Support for splitting in jobs and sending them to different computing environments

Full Systematics Analysis - Data Size



50 ROOT Files

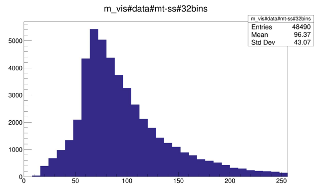
1174 of **5521** TTrees

~20 of **~550** branches

~4 x 10^{9} events

18 Gb of **595 Gb**

no friends



436 histograms for 1 variable

Benchmark Scenarios

1. Merge common datasets and nodes
2. Multiprocessing
3. Multithreading
4. Many variables scaling

1. Merge Common Datasets and Nodes

No optimization	→	**436** datasets	Current approach: 436 calls to **TTree::Draw()**
Merge common datasets	→	**153** datasets	New approach: 153 **RDataFrames**

(cfr. In the control plots analysis presented last time we merged 22 into 7)

1. Merge Common Datasets and Nodes

No optimization

Merge common datasets

Merge common datasets + merge common selections

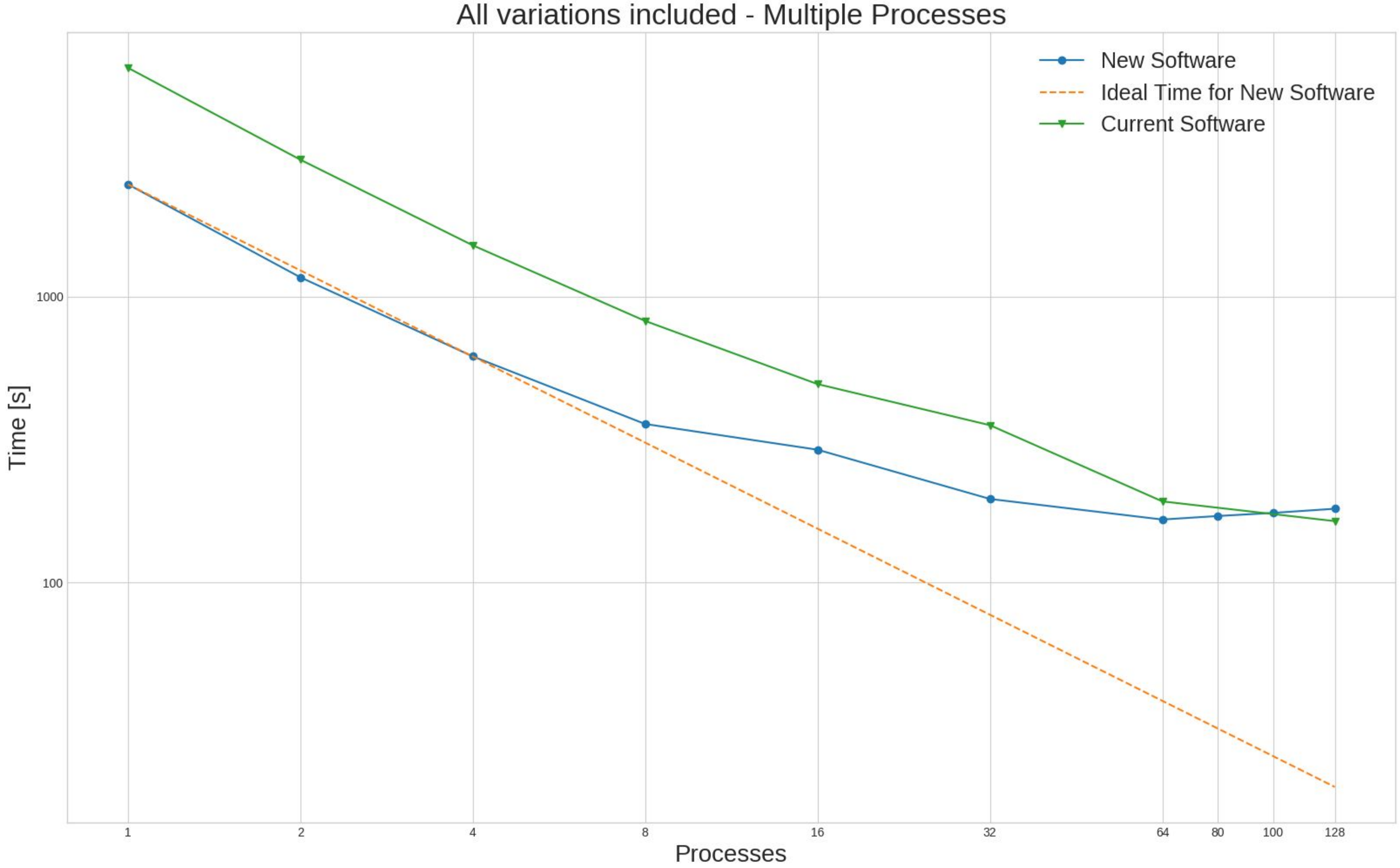
~38.55%

~0.12%

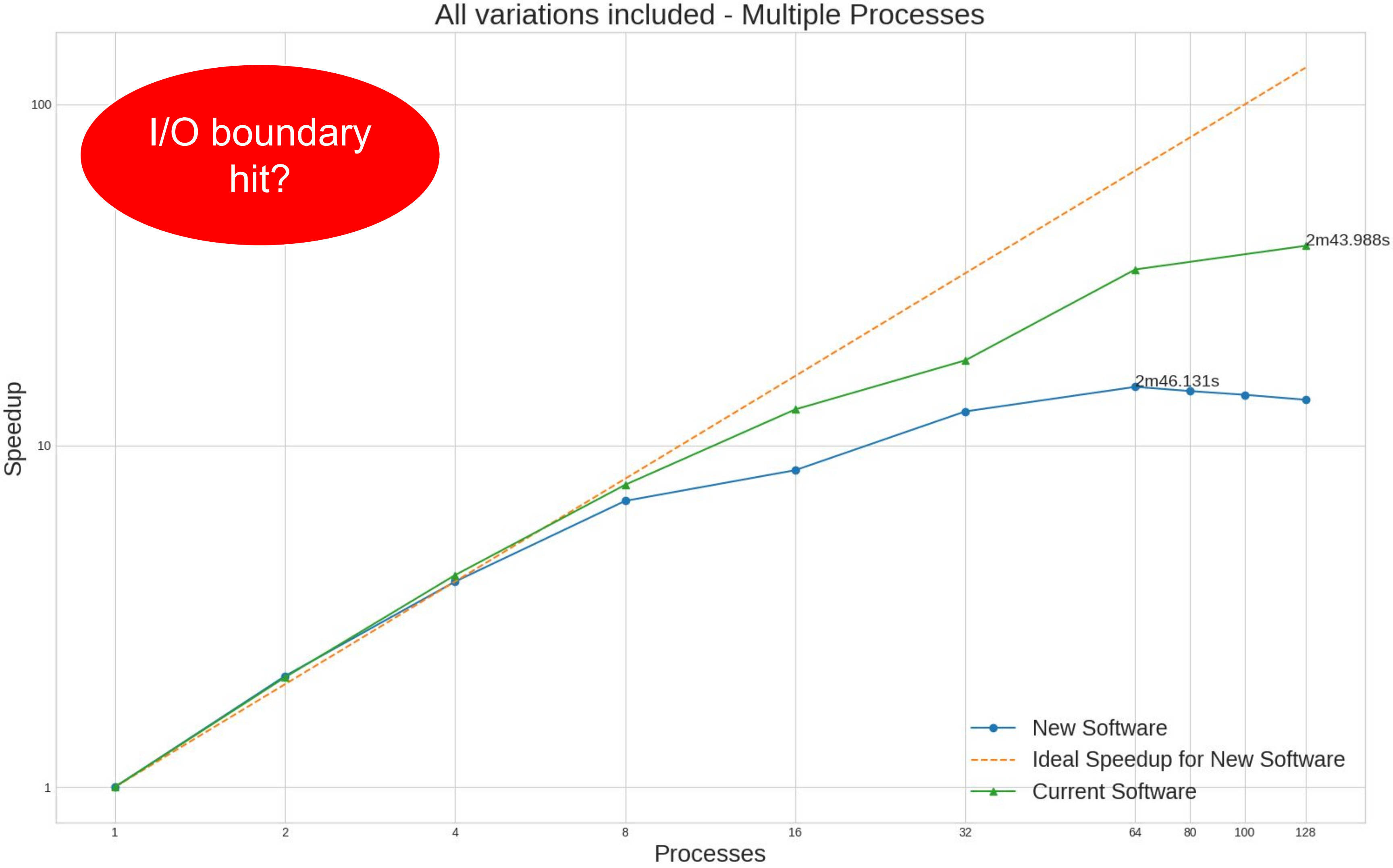
It's thus possible to get a huge speedup by just wisely setup the analysis workflow

2. Multiprocessing

- 1 channel, 1 variable
- All systematic variations included
- Production of 436 histograms
- Scale from 1 to 128 processes
- Test on machine with 128 (64) logical (physical) cores
- Comparison with the current software (`TTree::Draw()`)

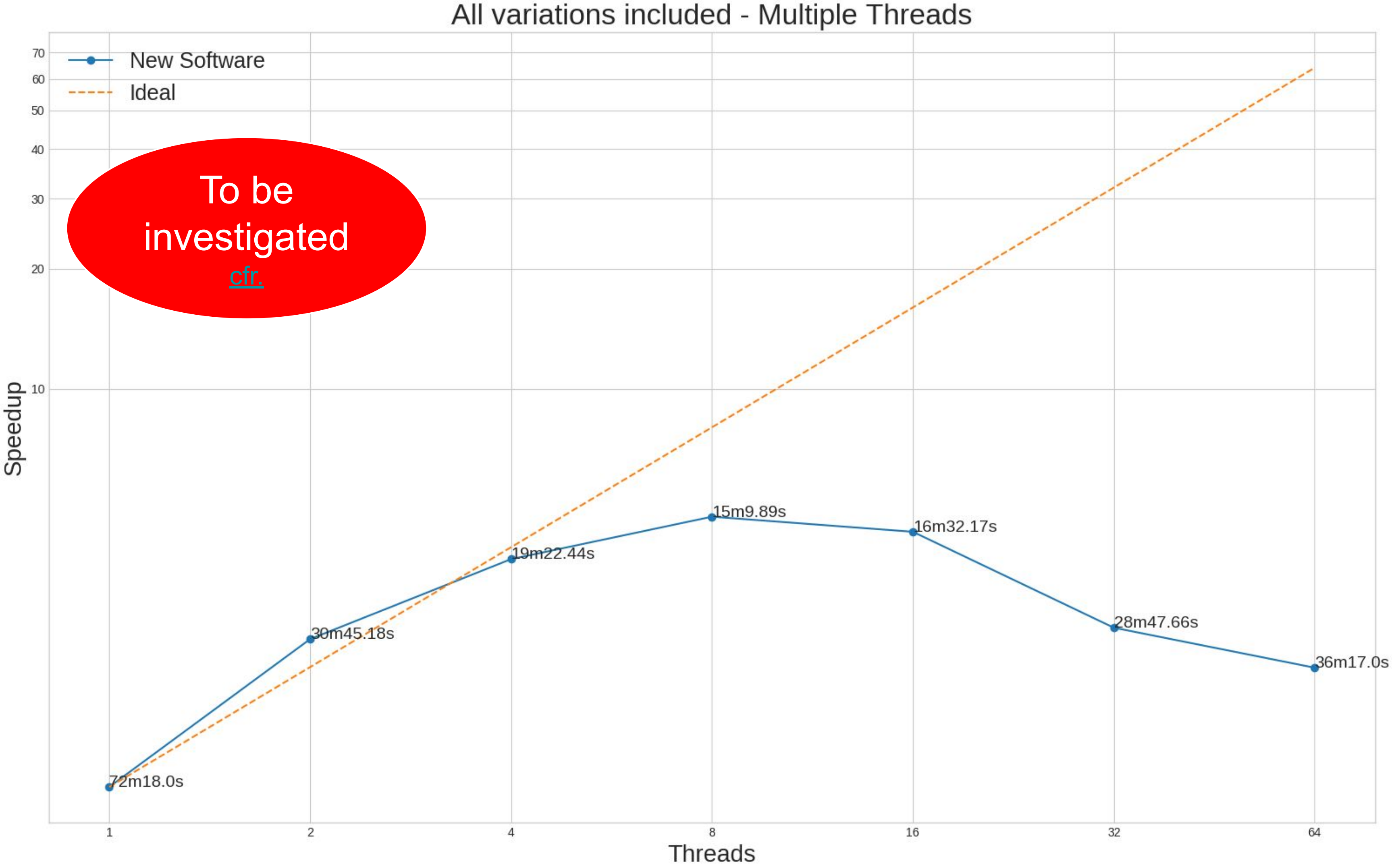


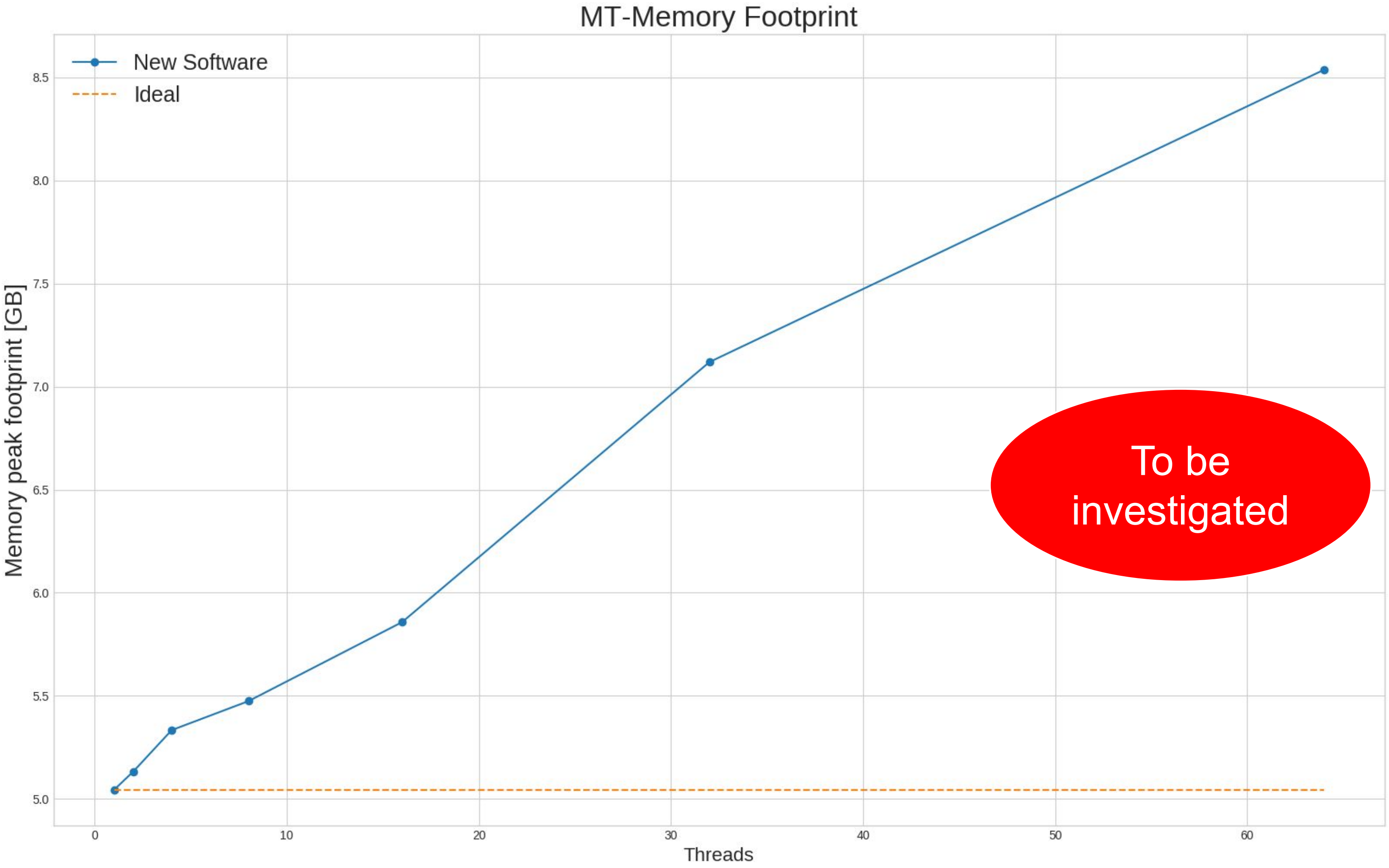
All variations included - Multiple Processes
New Software
Ideal Time for New Software
Current Software
Time [s]
1000
100
1
2
4
8
16
32
64
80
100
128
Processes

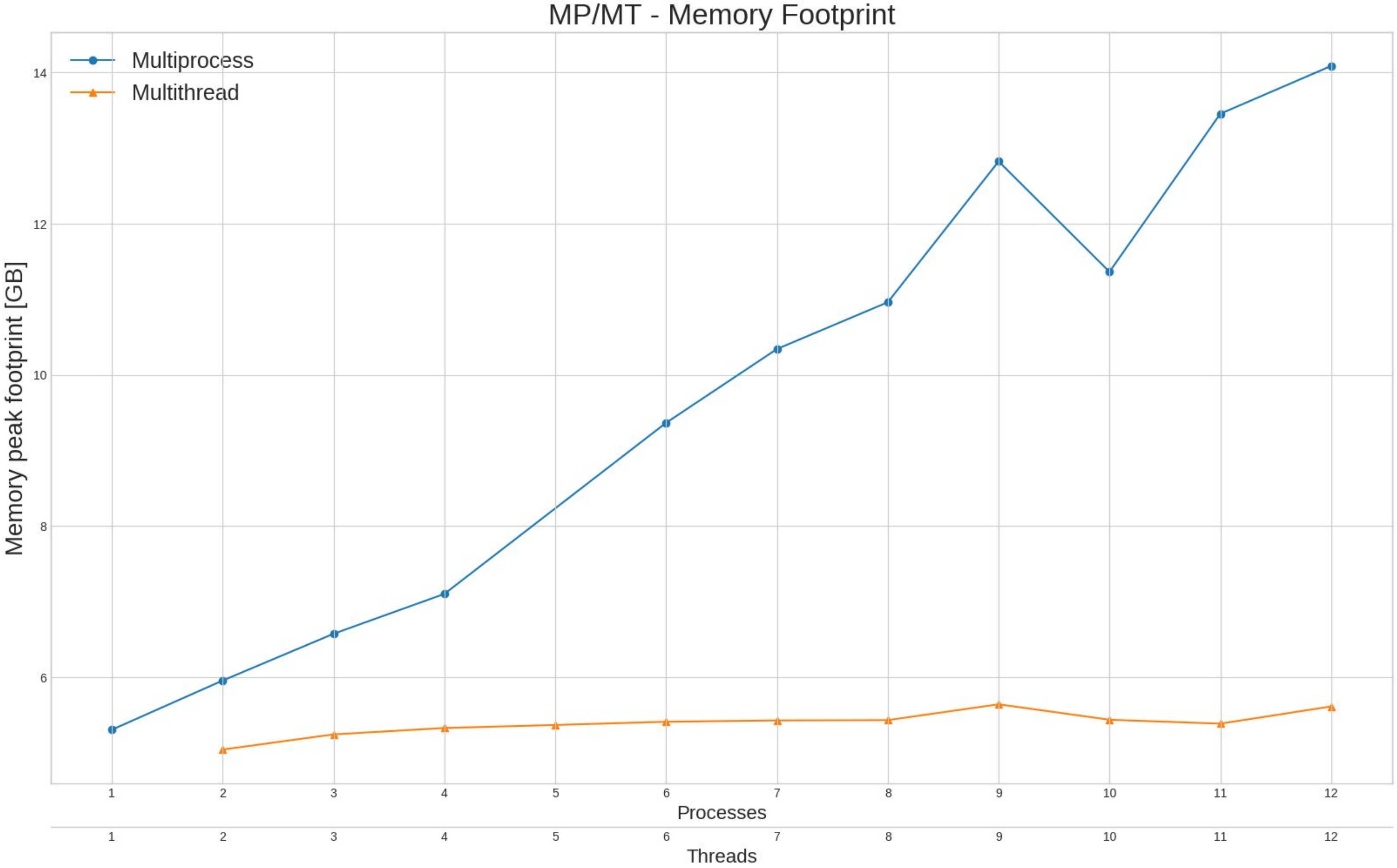

All variations included - Multiple Processes
I/O boundary hit?
Speedup
100
10
1
Processes
1
2
4
8
16
32
64
80
100
128
2m43.988s
2m46.131s
New Software
Ideal Speedup for New Software
Current Software

3. Multithreading

- Enabled with `RDataFrame.EnableImplicitMT(N)`
- 1 channel, 1 variable, single process
- Production of 436 histograms
- Scale from 1 to 64 threads
- Benchmark speedup
- Benchmark memory footprint and compare to multiprocessing

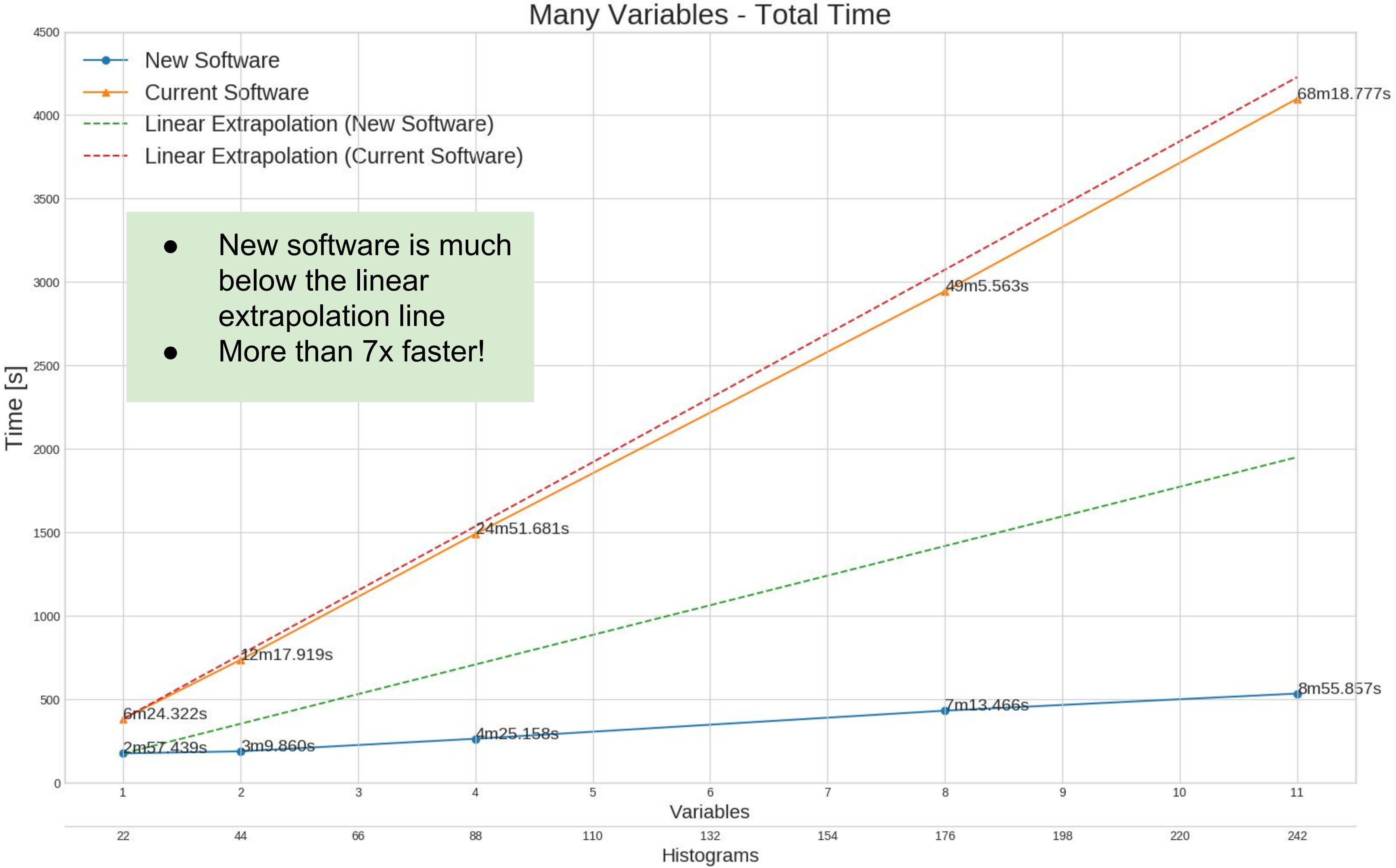

All variations included - Multiple Threads
New Software
Ideal
To be investigated
cfr.
Speedup
Threads
72m18.0s
30m45.18s
19m22.44s
15m9.89s
16m32.17s
28m47.66s
36m17.0s
70
60
50
40
30
20
10
1
2
4
8
16
32
64


MT-Memory Footprint
New Software
Ideal
Memory peak footprint [GB]
8.5
8.0
7.5
7.0
6.5
6.0
5.5
5.0
0
10
20
30
40
50
60
Threads
To be investigated

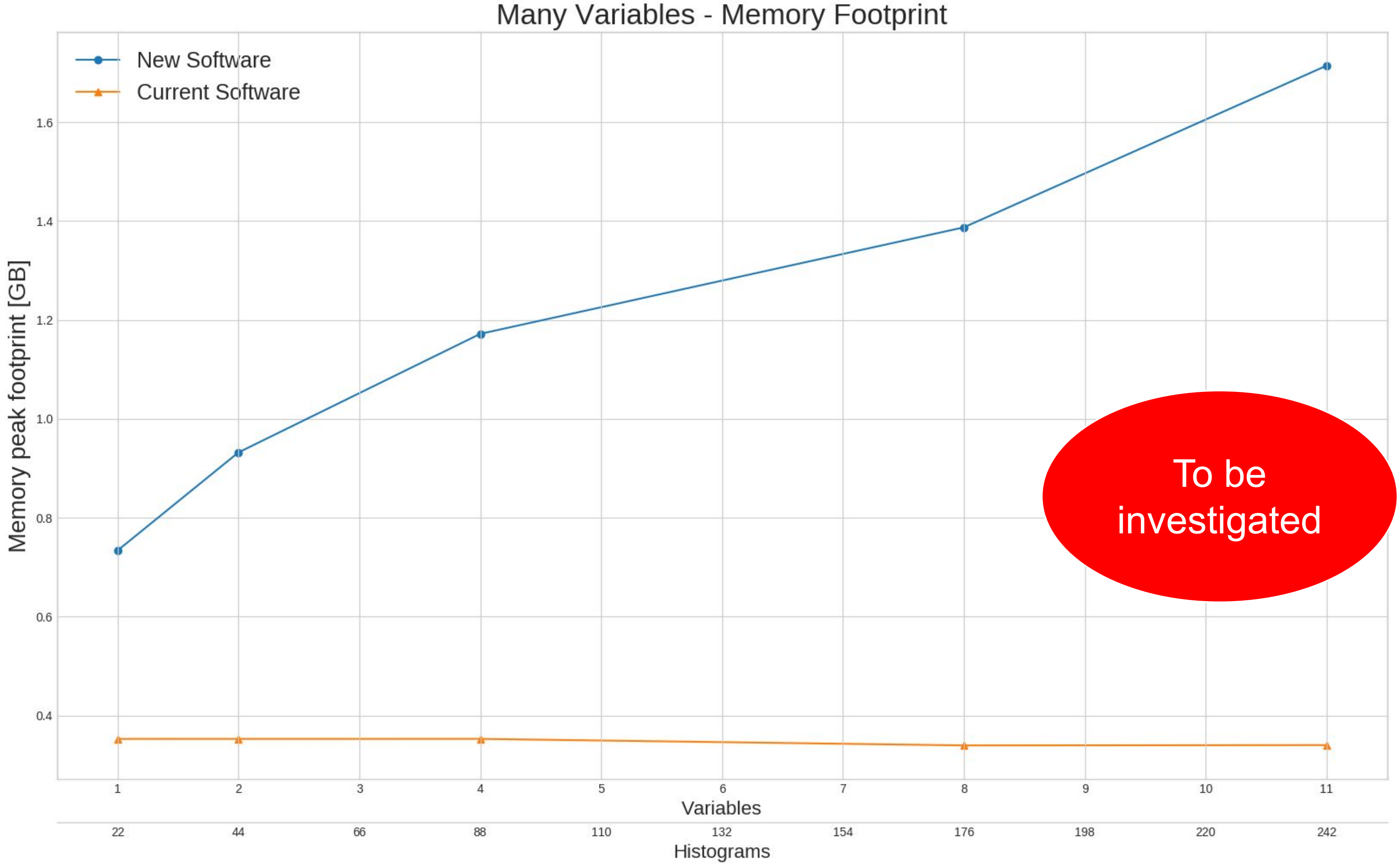

MP/MT - Memory Footprint
Multiprocess
Multithread
Memory peak footprint [GB]
14
12
10
8
6
1 2 3 4 5 6 7 8 9 10 11 12
Processes
1 2 3 4 5 6 7 8 9 10 11 12
Threads

4. Many Variables Scaling

- Single process, single thread
- Inclusive analysis (22 histograms produced for each variable)
- Different number of variables each time
- Benchmarked event loop time scaling, time per single histogram, memory footprint
- Comparison with the current software



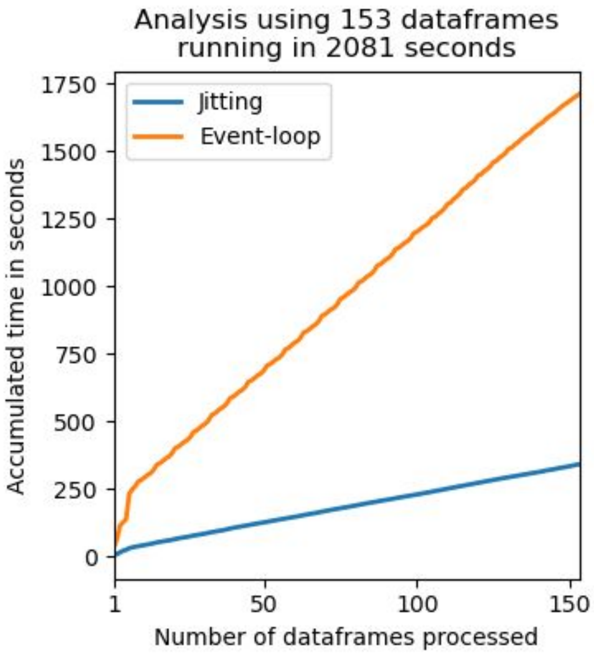
- New software is much below the linear extrapolation line
- More than 7x faster!

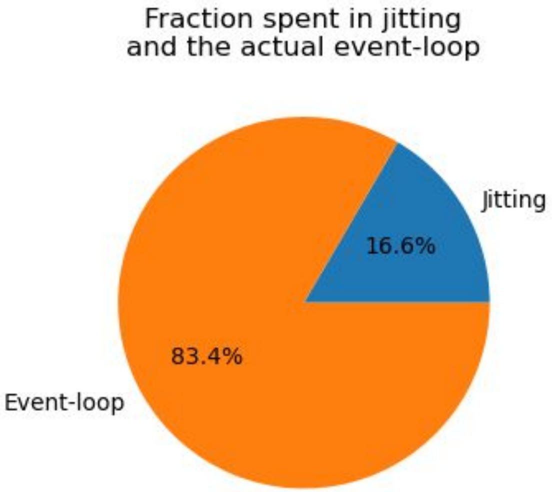


Many Variables - Memory Footprint
New Software
Current Software
Memory peak footprint [GB]
1.6
1.4
1.2
1.0
0.8
0.6
0.4
1
2
3
4
5
6
7
8
9
10
11
Variables
22
44
66
88
110
132
154
176
198
220
242
Histograms
To be investigated

Time spent in RDF event-loop

- Follow up to last weeks PPP
- Analysis with all systematics uses 153 dataframes
- Measuring time spent in RDF event-loop
  - Jitting
  - Actual event-loop





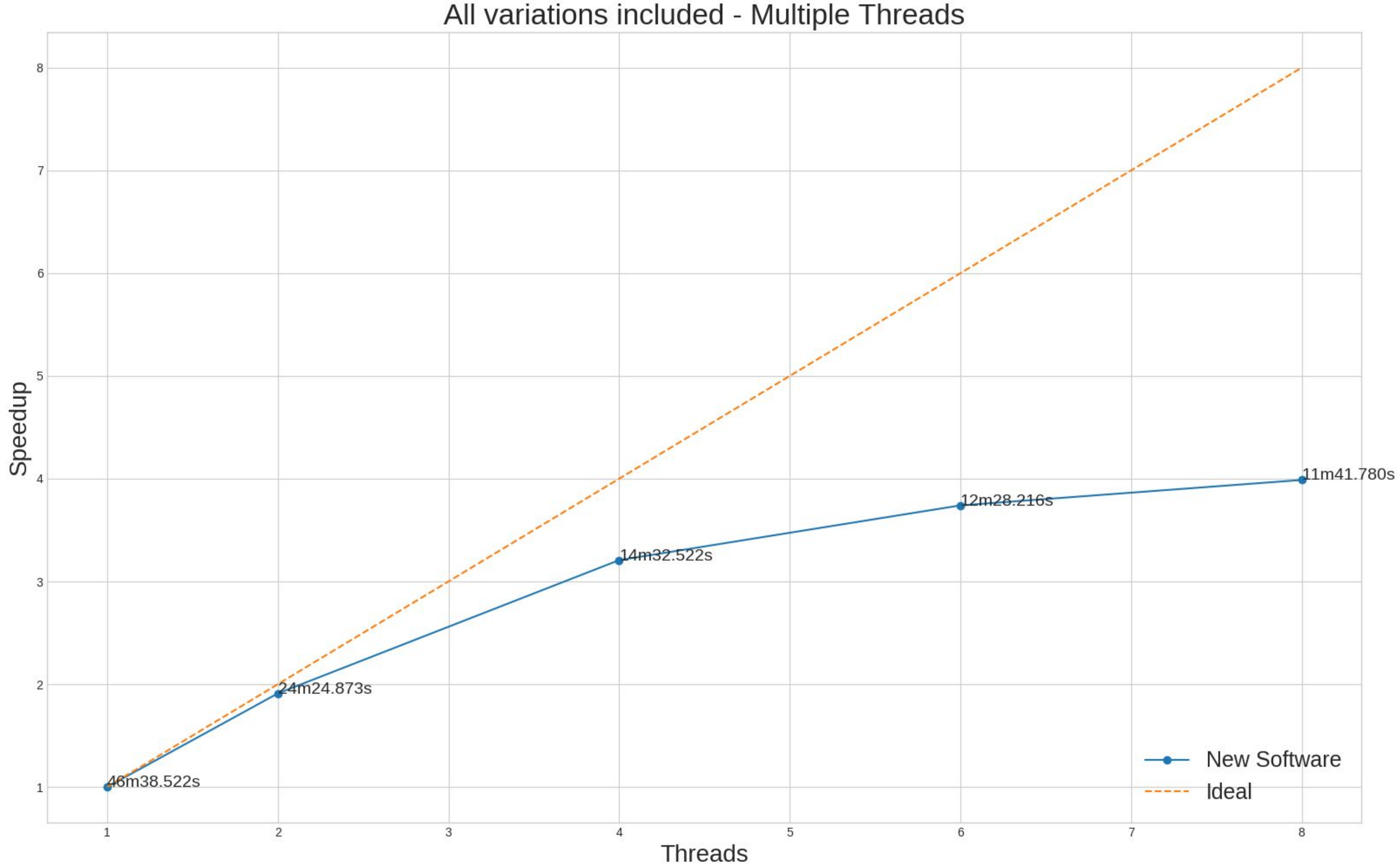
Discussion

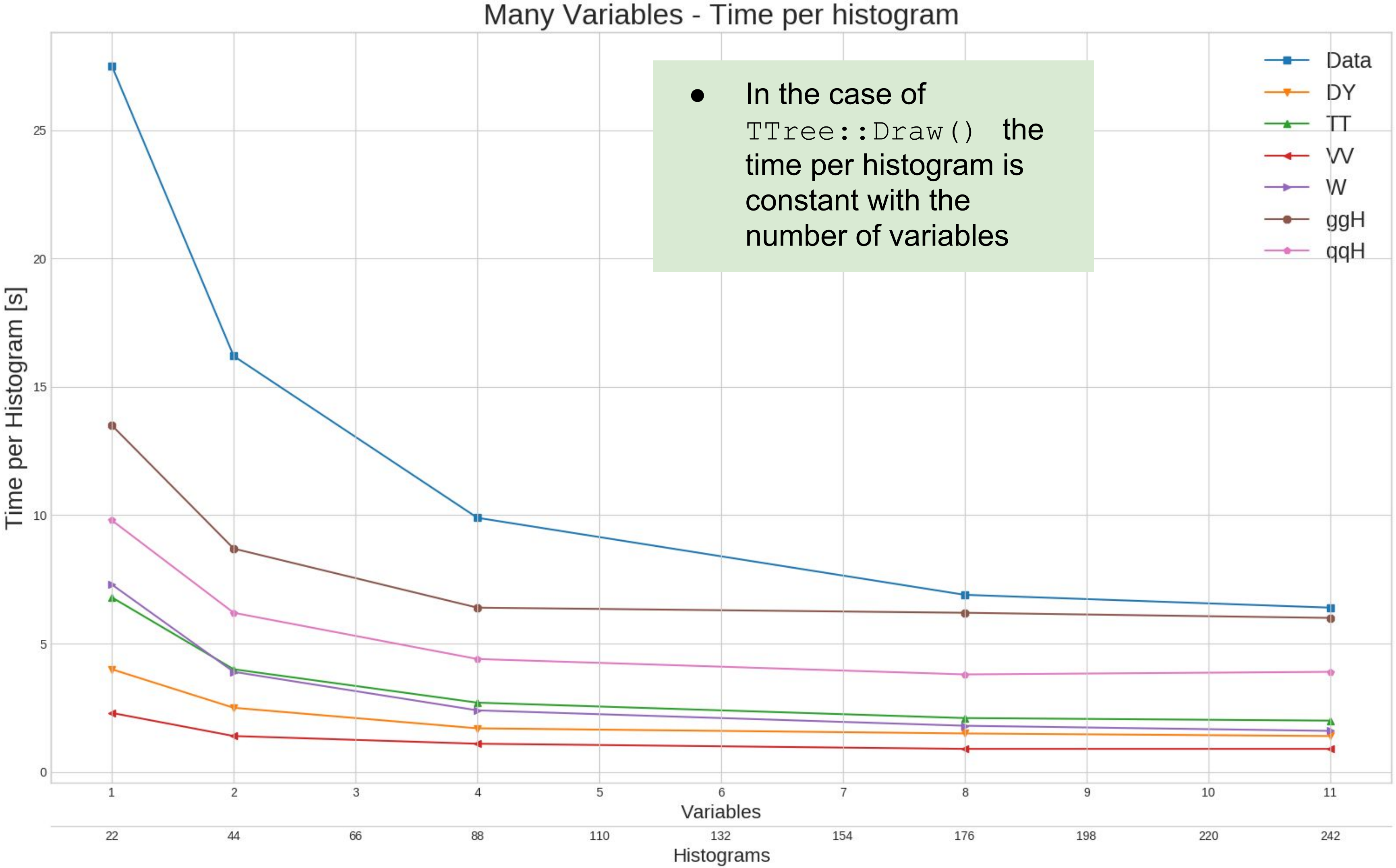
- Drop in scaling with MP: I/O boundary hit?
- Very bad scaling with MT: why?
- Further benchmarking: suggestions?
  - Is it worth combining ROOT MT with Python MP?

Backup

```
def get_nominal_datasets(channel):
    datasets = dict()
    for key, names in nominal_files.items():
        datasets[key] = dataset_from_nameset(
            key, names, channel + '_nominal', base_file, base_friends)
    return datasets
```

```
def get_nominal_units(channel, datasets):
    return {
            (...)
           'ztt' : Unit(datasets['DY'], [channel_selection(channel), DY_process_selection(channel),
                                          ZTT_process_selection(channel)],[histos]
                       ),
            (...)
          }
```


All variations included - Multiple Threads
Speedup
Threads
1
2
3
4
5
6
7
8
46m38.522s
24m24.873s
14m32.522s
12m28.216s
11m41.780s
New Software
Ideal



- In the case of `TTree::Draw()` the time per histogram is constant with the number of variables

Book Results - Concise and Structured

```
channels = ['mt', ...]

# Book nominal Units

nominals = {}
nominals['2017'] = {}
nominals['2017']['datasets'] = {}

# E.g. DY dataset

dy_dataset = dataset_from_nameset( 'DY', nominal_files['DY'], 'mt_nominal',
    base_file, base_friends)
```

```
# nominal_files is placed inside ntuple_config
nominal_files = {
   (...)
   'DY': [
       'DY2JetsToLLM50_RunIIFall17MiniAODv2_PU2017_13TeV_MINIAOD_madgraph-pythia8_ext1-v2',
       'DY2JetsToLLM50_RunIIFall17MiniAODv2_PU2017_13TeV_MINIAOD_madgraph-pythia8_v1',
       'DY3JetsToLLM50_RunIIFall17MiniAODv2_PU2017_13TeV_MINIAOD_madgraph-pythia8_ext1-v1',
       (...)
       ],
    (...)
}
```