



The RDF $t\bar{t}$ -analysis implementation

Analysis Grand Challenge

Andrii Falko

Home university:

Taras Shevchenko National University of Kyiv

Mentors:

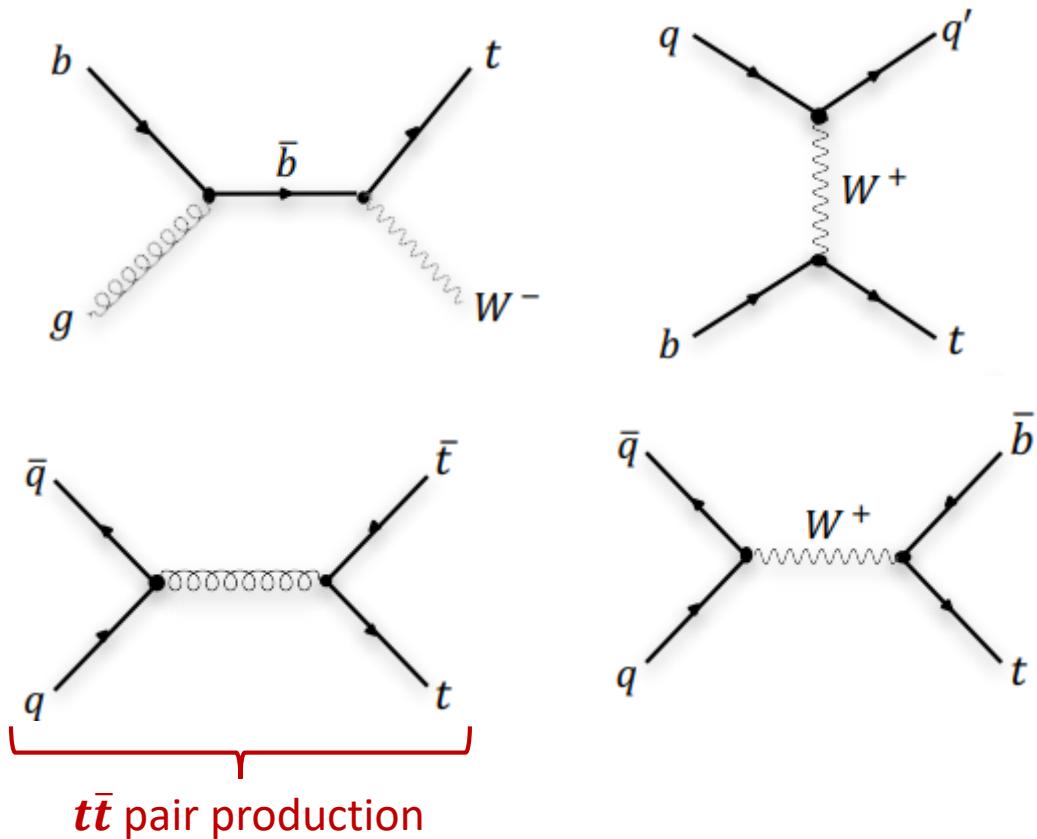
Enrico Guiraud, Alexander Held, Oksana Shadura

Outline

- $t\bar{t}$ -analysis basics
 - input data
- $t\bar{t}$ -analysis algorithm
 - produced histograms
 - performance measurements

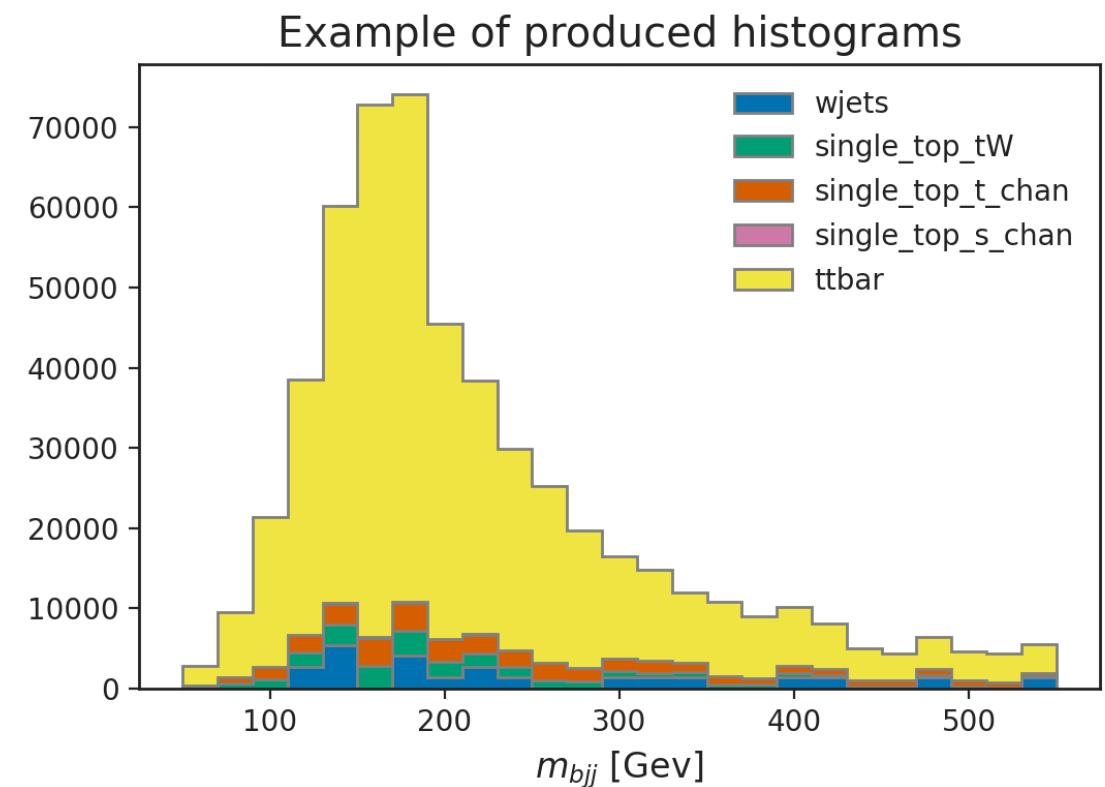
Project background: $t\bar{t}$ -analysis

1. A particle abundantly produced at the LHC:



2. Input dataset: [2015 CMS Open Data](#)

3. Existing [Coffea implementation](#) by Alexander Held:



4. The goals stated and achieved in the project :

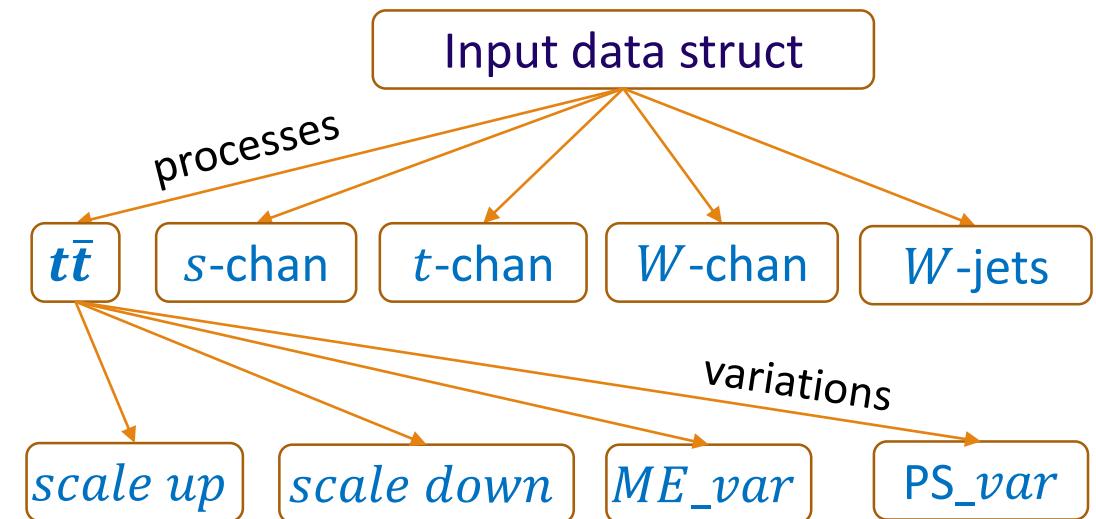
- [\$t\bar{t}\$ -analysis specification](#) in plain English
- [RDF \$t\bar{t}\$ -analysis implementation](#)
- [Benchmarking](#)

5. Benefits from done work:

- validation of ROOT's modern analysis interfaces:
 - produced histograms are exactly the same as with Coffea
 - RDF shows good performance
- A good example of a realistic analysis pipeline with RDF

Input

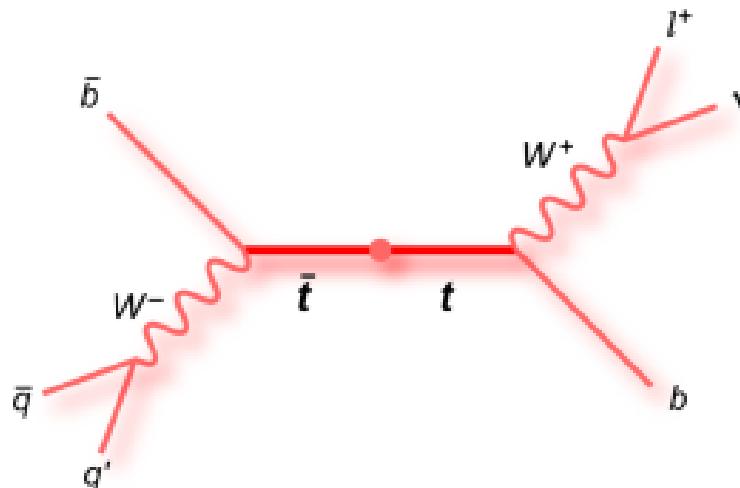
- 1) [2015 CMS Open Data](#)
- 2) 9 subsets of root files produced in MC simulation
- 3) 5 interaction channels → 5 processes involved
- 4) 4 kinds of variations → given as 4 additional sets



What is the $t\bar{t}$ -analysis logic?

1. Semileptonic decay diagram

- 1 lepton
- two quarks
- two b-quarks



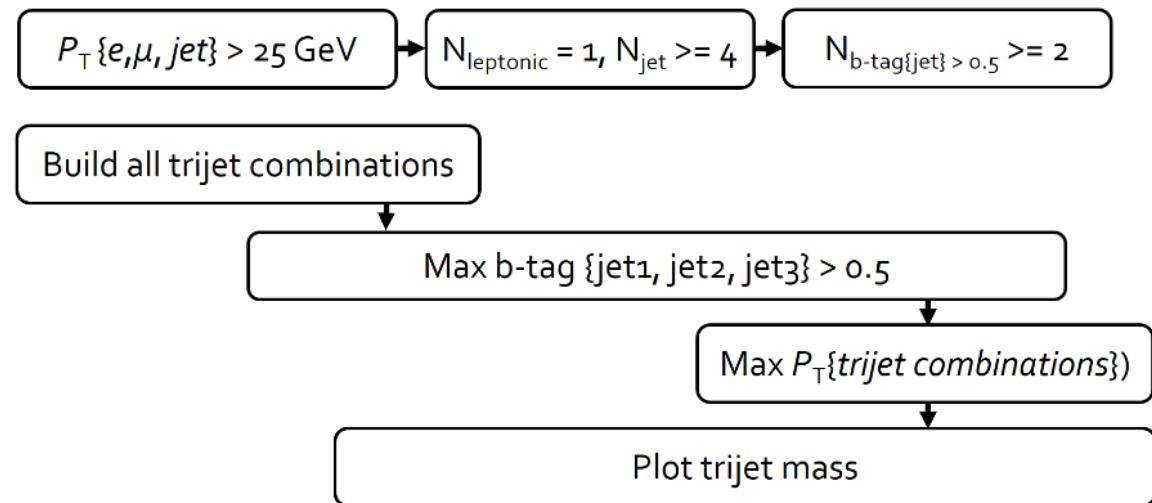
2. Schema summarizing $t\bar{t}$ -analysis from my document:

Two main stages of analysis

- Select events containing lepton, 4 jets
- Find the trijet combination which is the decay product of top-quark among all other possible three-jet combinations in every event

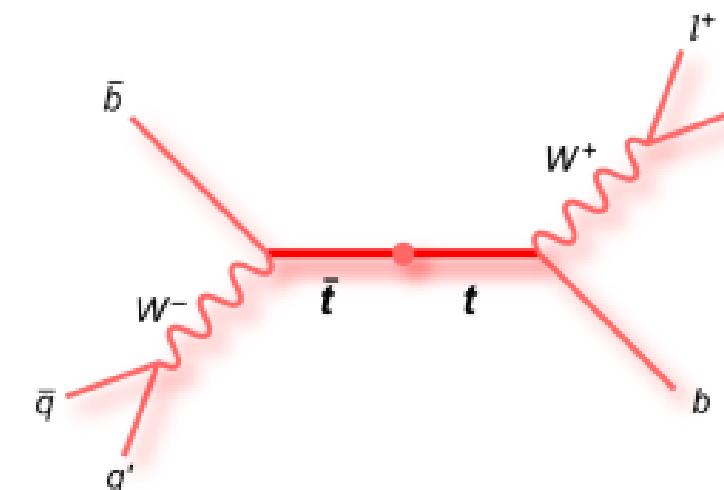
3. Described in my [analysis spec](#)

$t\bar{t}$ -analysis pipeline



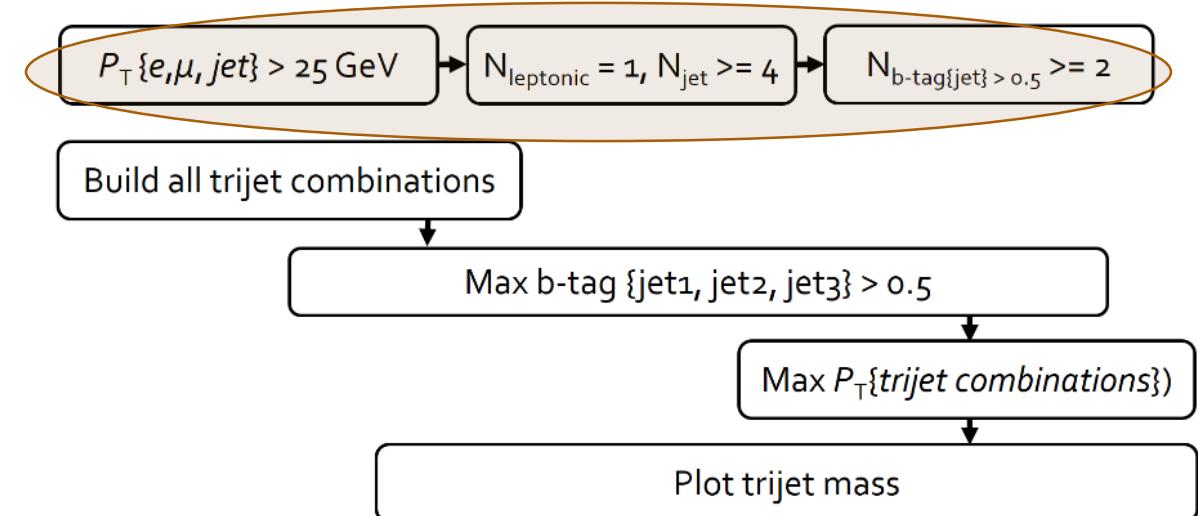
Selection events

```
d = d.Define('electron_pt_mask', 'electron_pt>25')\
    .Define('muon_pt_mask', 'muon_pt>25')\
    .Define('jet_pt_mask', 'jet_pt>25')\
    .Filter('Sum(electron_pt_mask) + Sum(muon_pt_mask) == 1')\
    .Filter('Sum(jet_pt_mask) >= 4')\
    .Filter('Sum(jet_btag[jet_pt_mask]>=0.5)>=2')
```



- Creating masks
- Applying masks to select events we need
- single lepton with P_T higher than 25
- four jets with P_T higher than 25
- At least two jets with a btag higher than 0.5
- Both threshold values are customized

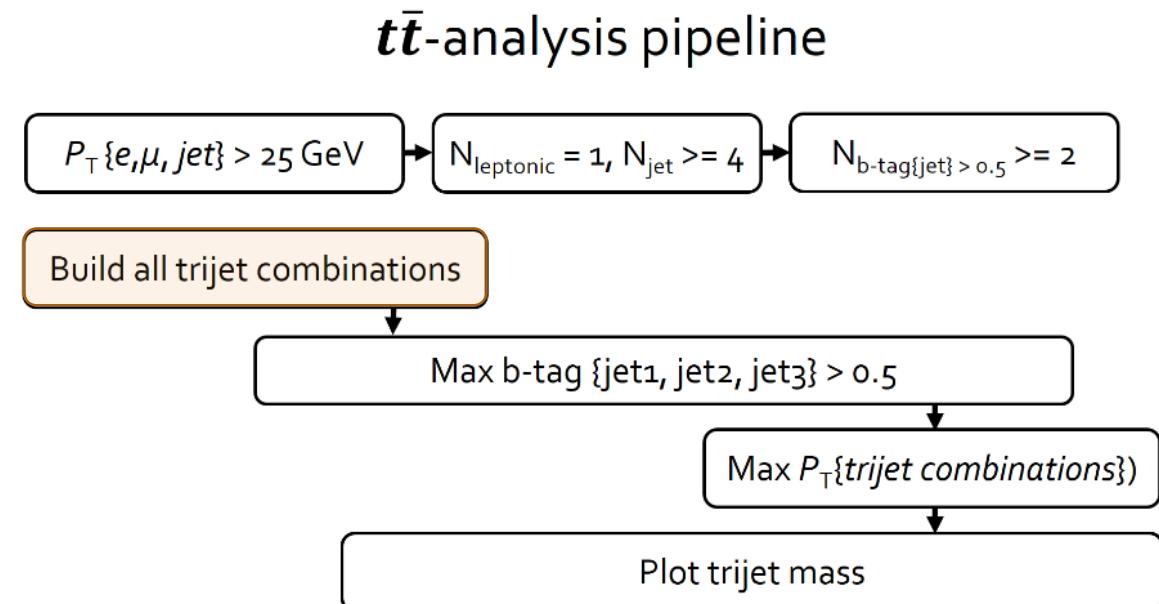
$t\bar{t}$ -analysis pipeline



Constructing trijets

```
# building trijet combinations
d = d.Define('trijet',
    'ROOT::VecOps::Combinations(jet_pt[jet_pt_mask],3)'
)
```

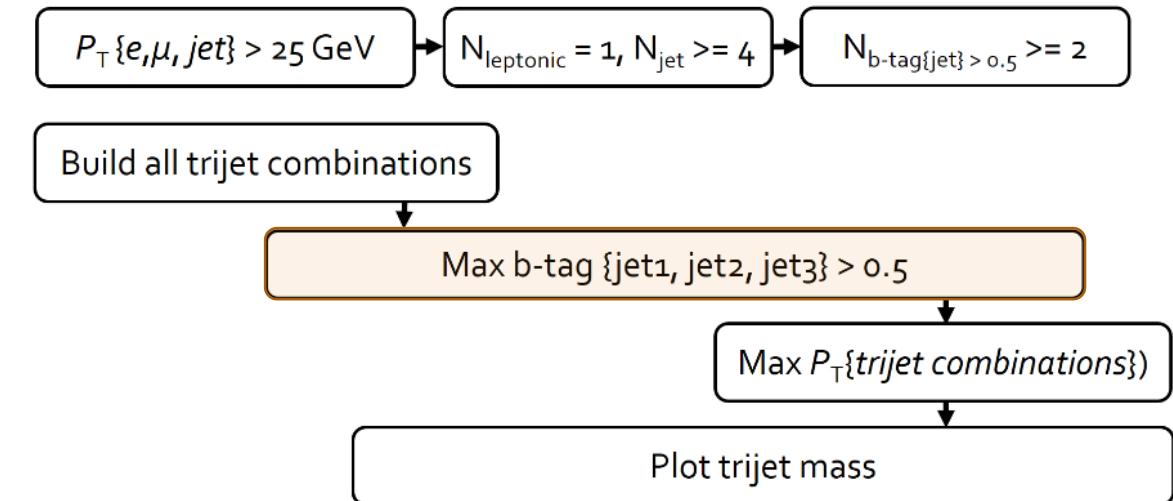
- Build all three jets combinations using [Combinations](#)



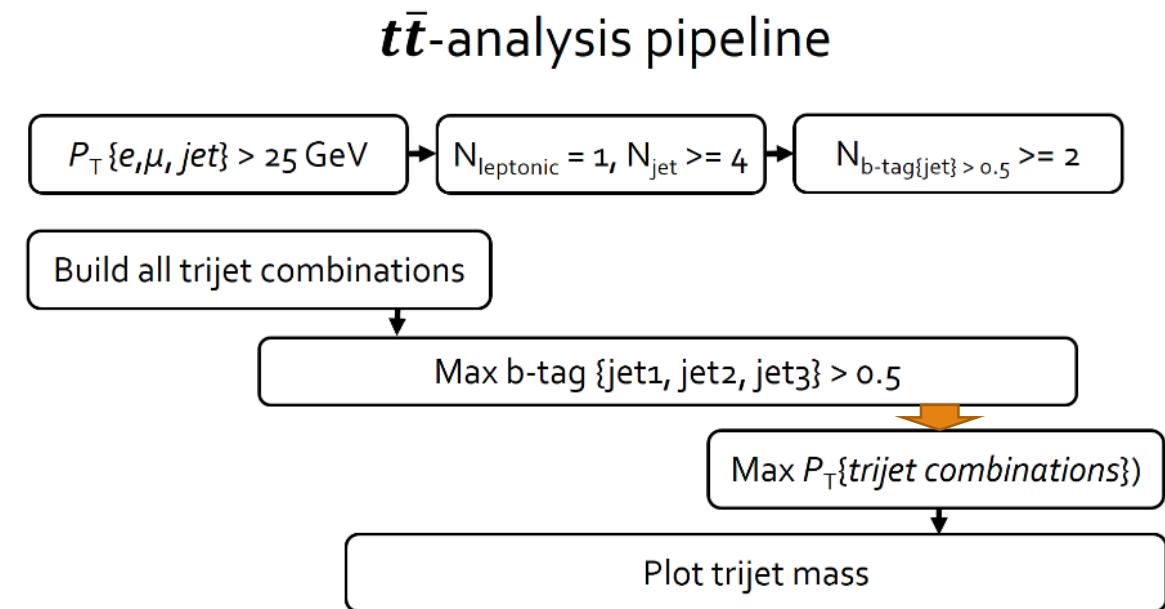
Identifying jets with high enough btag value

```
# btag_masks indentify trijets with btag higher than 0.5
d = d.Define('btag_mask',
    'auto ntrijet = trijet[0].size();' +\
    'ROOT::VecOps::RVec<bool> btag(ntrijet);' +\
    'for (int i = 0; i < ntrijet; ++i) {' +\
    'int j1 = trijet[0][i]; int j2 = trijet[1][i]; int j3 = trijet[2][i];' +\
    'btag[i]=std::max({jet_btag[j1], jet_btag[j2], jet_btag[j3]})>0.5;' +\
    '}' +\
    'return btag;' +\
) +\
.Define('ntrijet',
    'int ntrijet = 0;' +\
    'for (int i=0; i < btag_mask.size(); ++i)' +\
    'ntrijet+=btag_mask[i];' +\
    'return ntrijet;' +\
) # the number of “good” combinations
```

$t\bar{t}$ -analysis pipeline



- ▶ 1. Define jets *4-momentumes*
- 2. Define trijets *4-momentums*
- 3. Derive trijets P_T

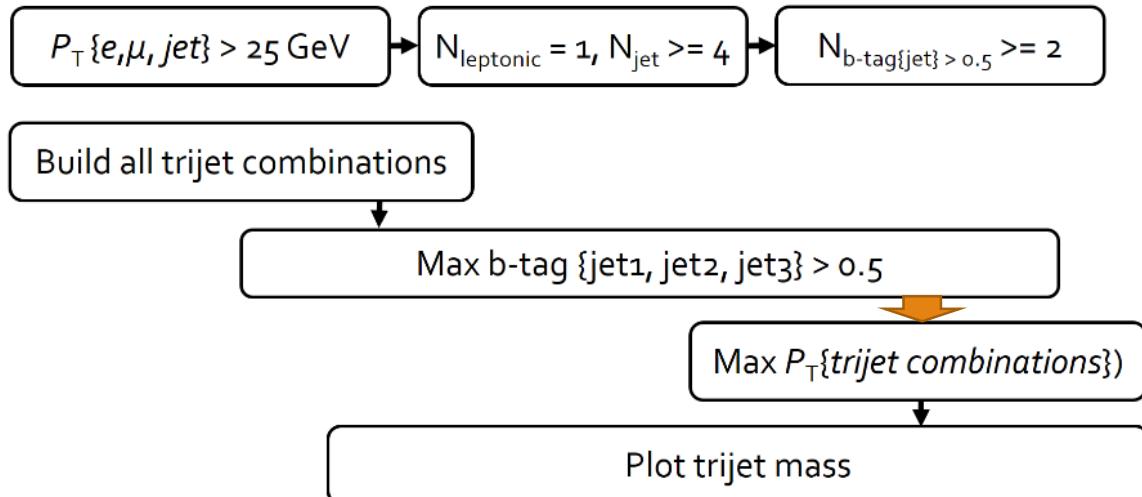


```

# define jets four-momentumes
d = d.Define("jet_p4",
  "ROOT::VecOps::Construct<ROOT::Math::PxPyPzMVector>(jet_px[jet_pt_mask],
jet_py[jet_pt_mask], jet_pz[jet_pt_mask], jet_mass[jet_pt_mask]")
)
  
```

$t\bar{t}$ -analysis pipeline

- 1. Define jets 4-momentums
- 2. Define trijets 4-momentums
- 3. Derive trijets P_T

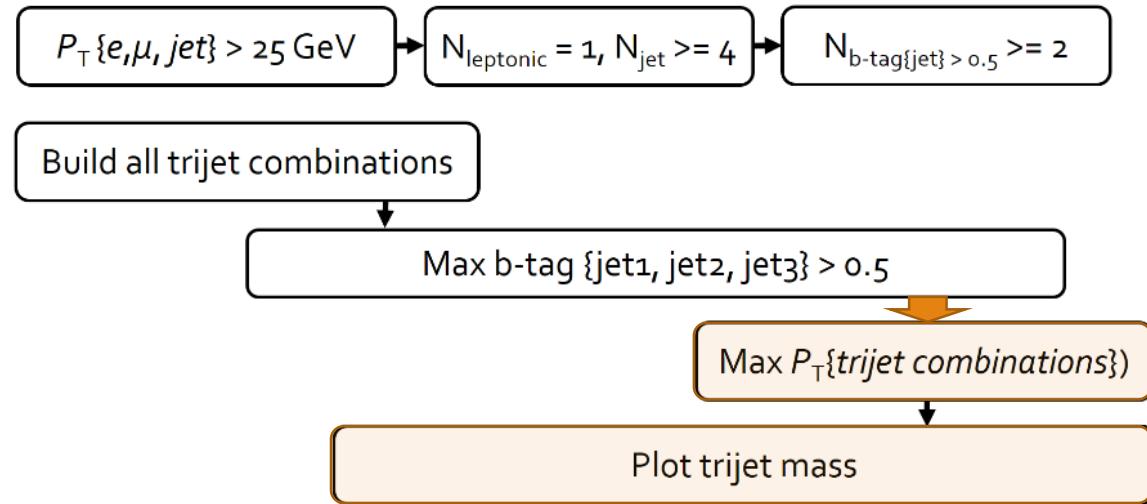


```

d = d.Define('trijet_p4',
             'ROOT::VecOps::RVec<ROOT::Math::PxPyPzMVector> trijet_p4(ntrijet);'
             'auto trijet0 = trijet[0][btags];'
             'auto trijet1 = trijet[1][btags];'
             'auto trijet2 = trijet[2][btags];'
             'for (int i = 0; i < ntrijet; ++i) {'
                 'int j1 = trijet0[i]; int j2 = trijet1[i]; int j3 = trijet2[i];'
                 'trijet_p4[i] = jet_p4[j1] + jet_p4[j2] + jet_p4[j3];'
             '}'
             'return trijet_p4; ' )
  
```

- 1. Define jets *4-momentumes*
- 2. Define trijets *4-momentums*
- 3. Derive trijets P_T

$t\bar{t}$ -analysis pipeline



getting trijet transverse momentum values from four-momentum vectors

```
d = d.Define('trijet_pt',
  'return ROOT::VecOps::Map(trijet_p4, [](const ROOT::Math::PxPyPzMVector &v) { return v.Pt() })')
```

- 4. Trijet with max P_T is the product of t-quark!

```
d=d.Define('mass', 'trijet_p4[ROOT::VecOps::ArgMax(trijet_pt)].M();')
```

Output

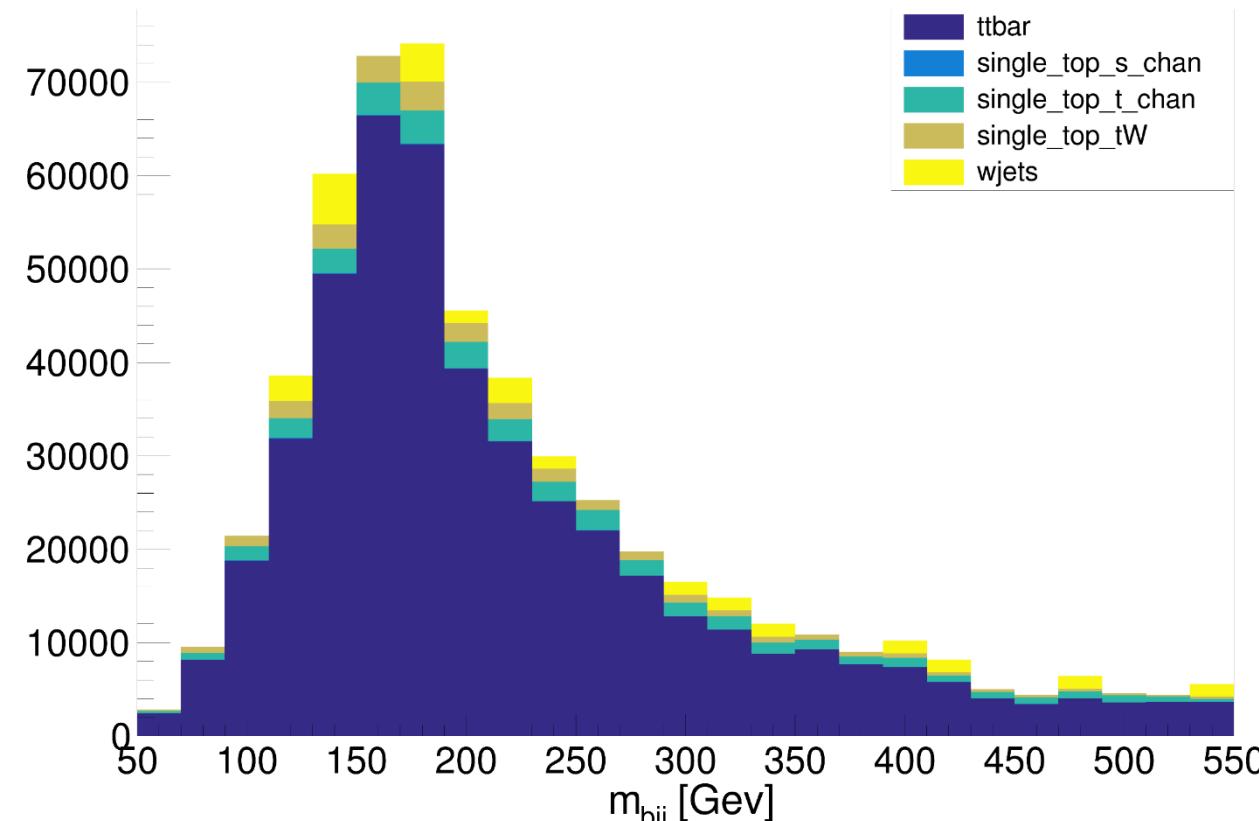
12

Two selection requirements:

```
fork = d.Filter('Sum(jet_btag[jet_pt_mask]>=0.5)>1')
```

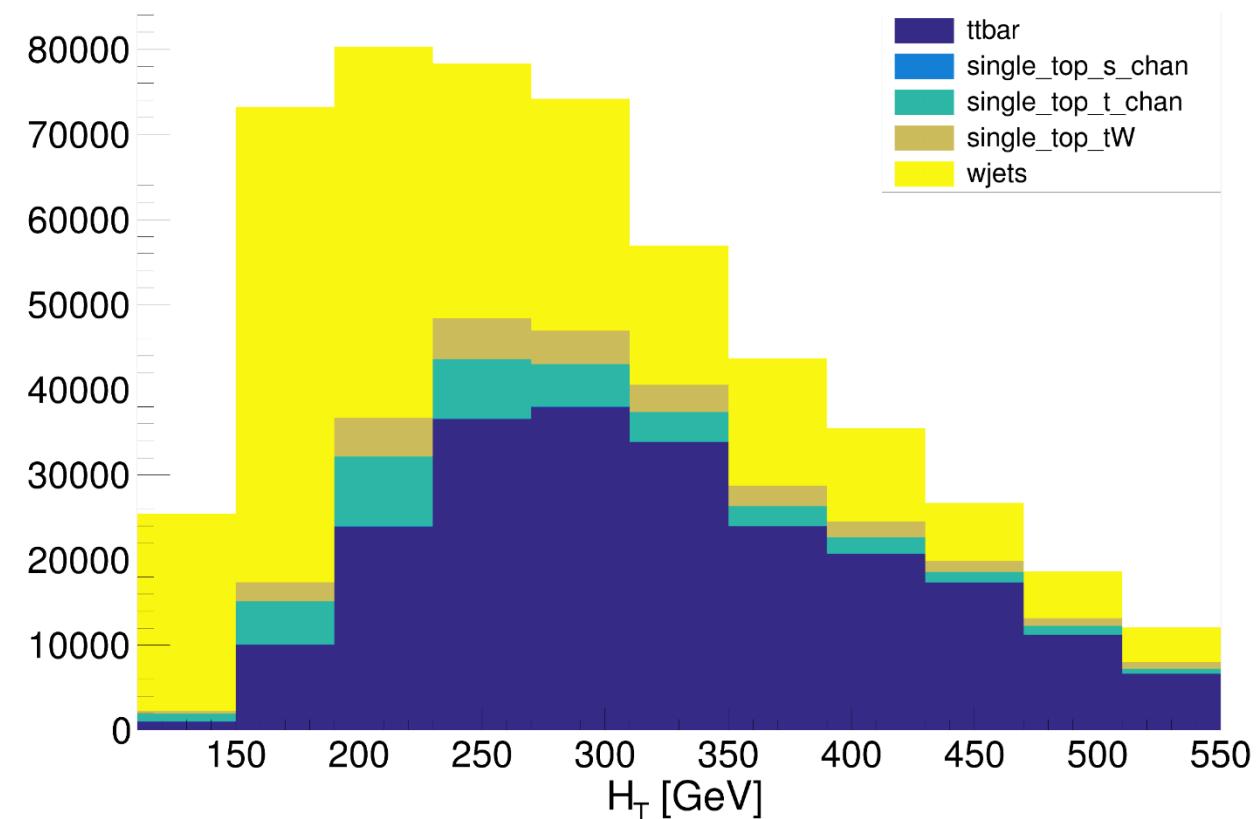
```
fork = d.Filter('Sum(jet_btag[jet_pt_mask]>=0.5)==1')
```

≥ 4 jets, 2 b-tag



Top-quark mass peak plot

≥ 4 jets, 1 b-tag



Scalar sum of transverse momenta plot

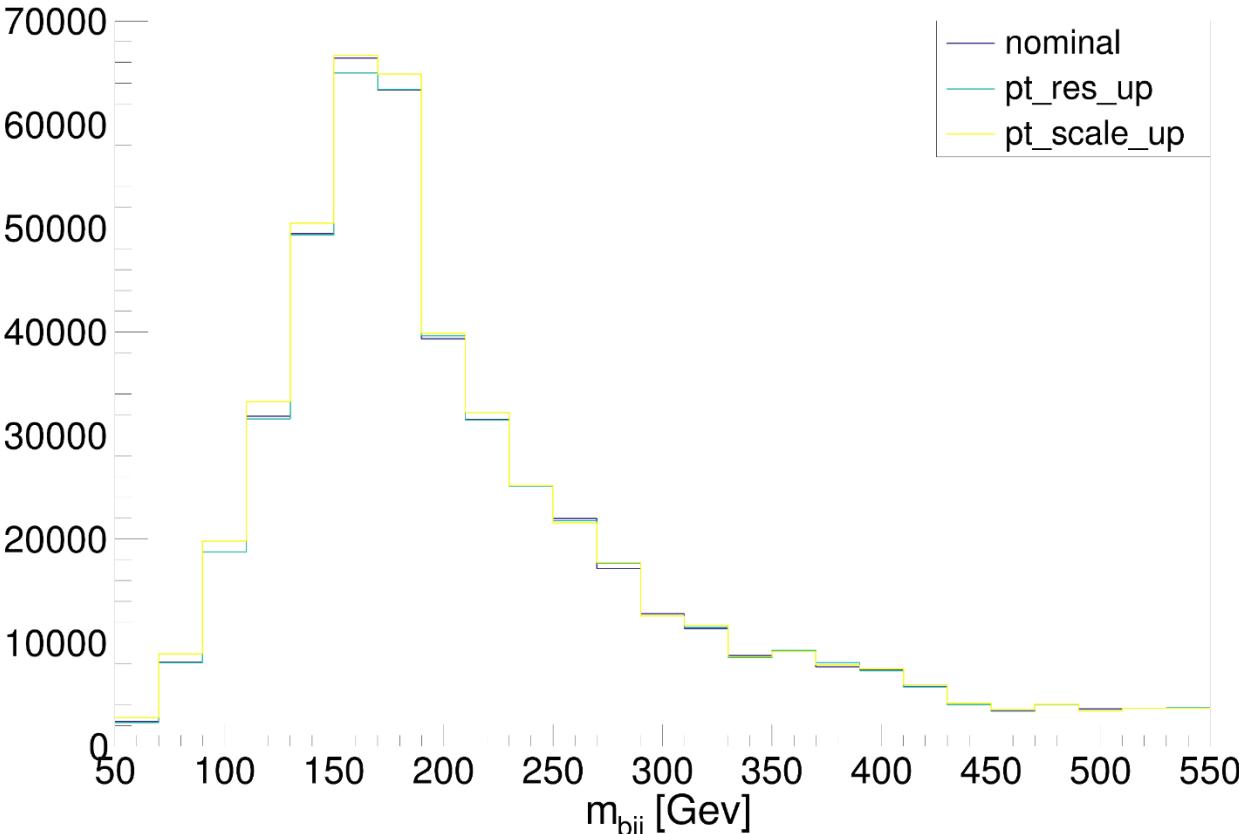
Variations

13

```
fork = d.Filter('Sum(jet_btag[jet_pt_mask]>=0.5)>1')
```

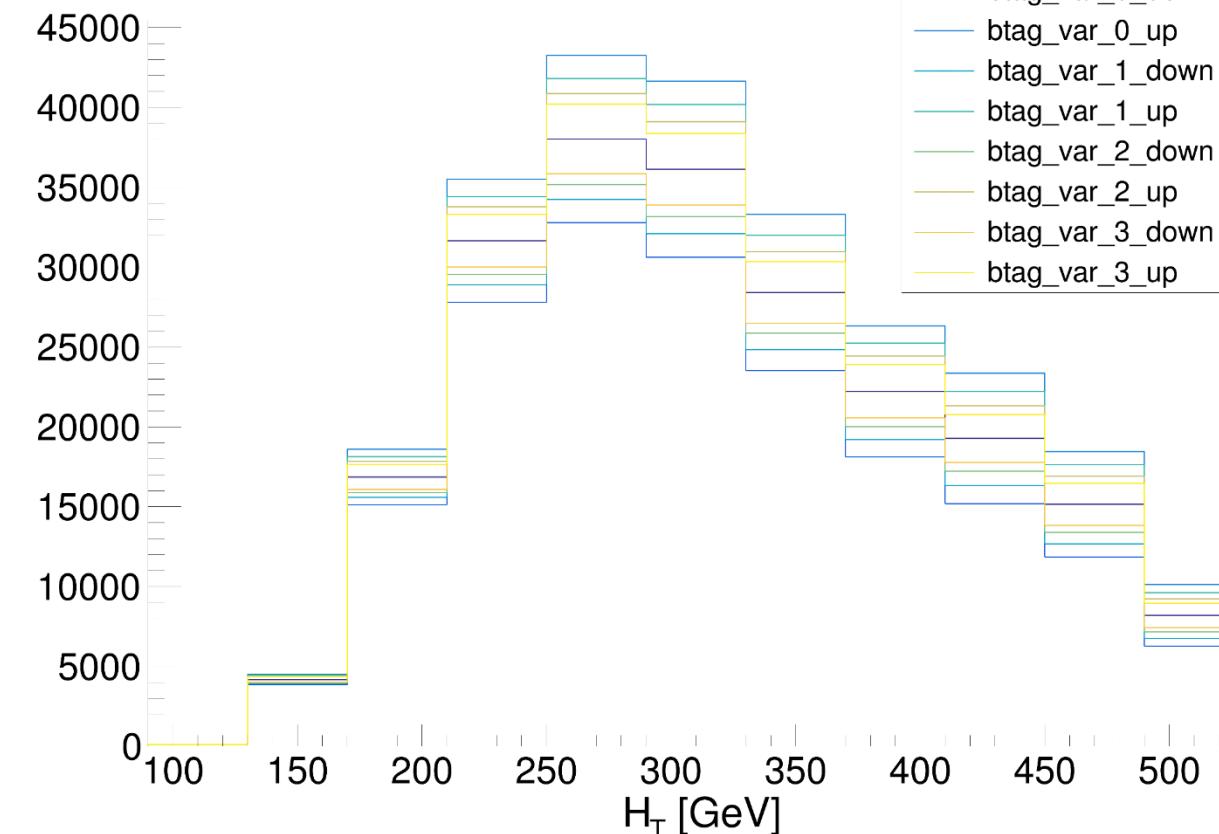
```
fork = d.Filter('Sum(jet_btag[jet_pt_mask]>=0.5)==1'
```

Jet energy variations



Top-quark mass peak plot

b-tagging variations



Transverse momentum plot

Total of 122 histograms! Link to [jupyter-notebook](#) with implementation

Benchmark setup

1. Machine specifications:

- 64 physical cores
- 128 logical cores
- 125 Gi RAM

2. Input data

- Data files are downloaded to the fast SSD

3. Measuring quantities:

- Time taken
- RAM usage
- Throughput [events/s/core]
- Throughput [files/s/core]

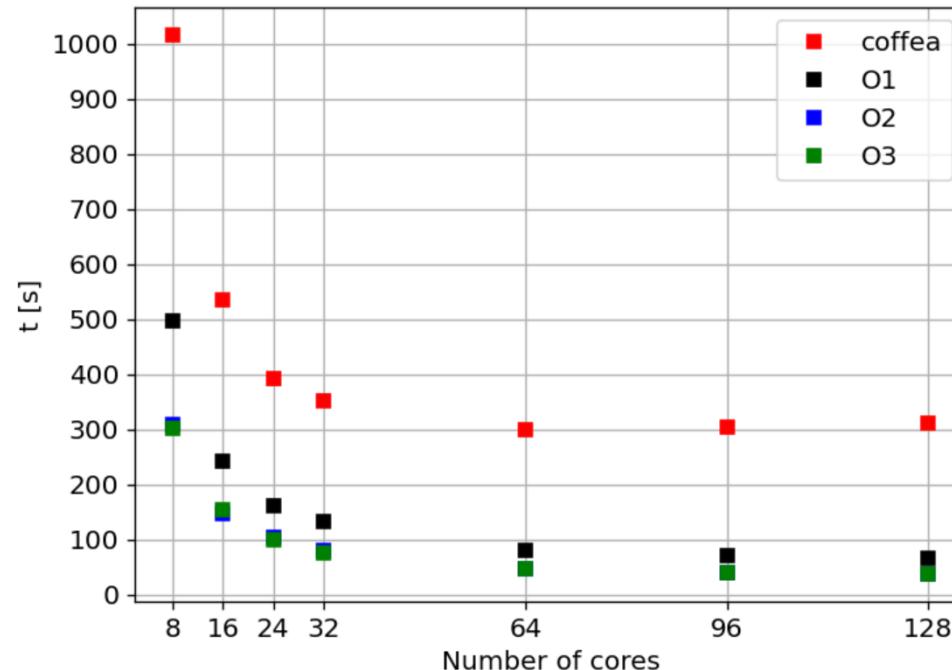
6. Dedicated [branch](#) with scripts I have ran

4. Configurations

- Coffea executors: Futures and Dask
- RDF optimization levels: O1, O2, O3
- ROOT master@<f0a240b855> (6.27/01)
- Coffea version 0.7.16

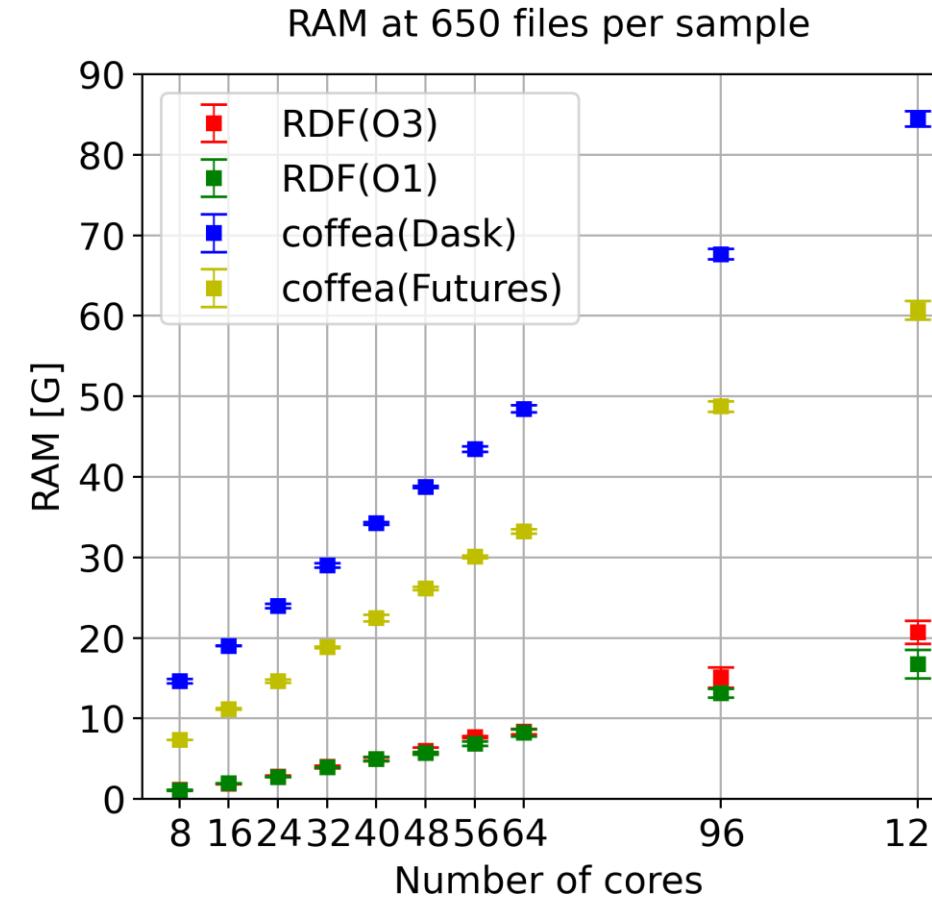
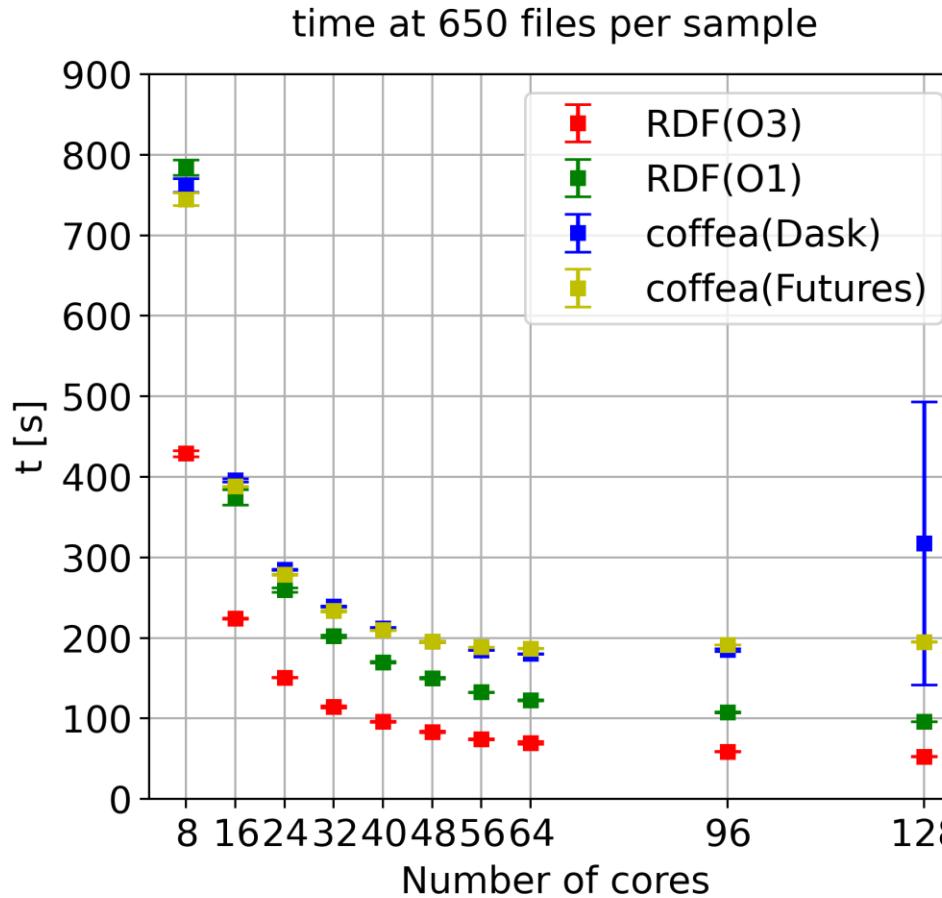
5. O1 vs O2 vs O3

time for 3570 files per sample



Benchmark results

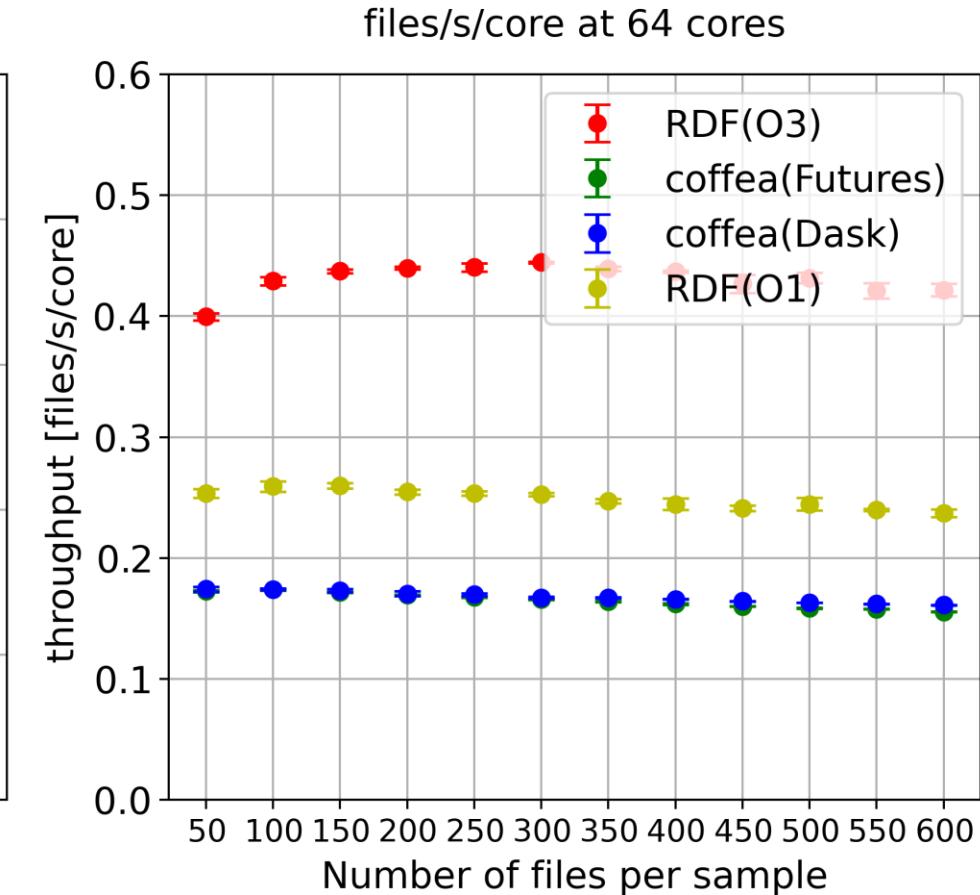
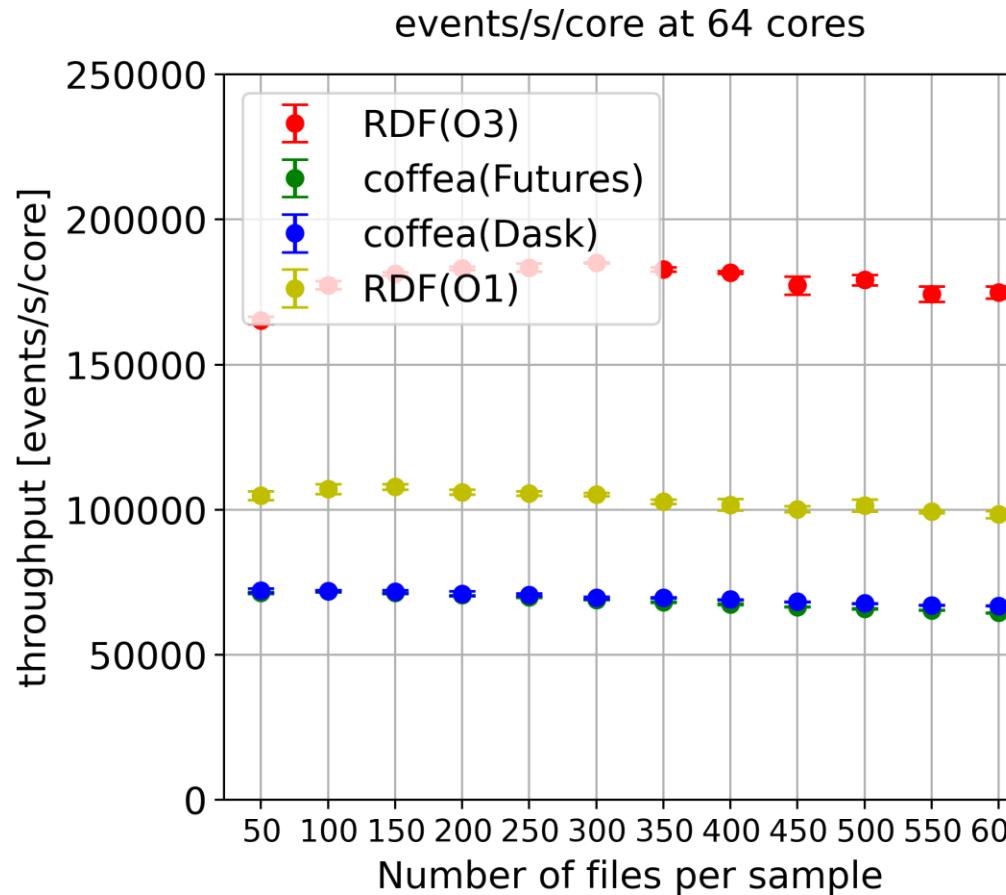
2. Time and RAM usage dependence on the number of used cores (765,723,174 events).
 - O1 vs O2 (vs O3) in RDF makes a noticeable difference -> it will be good to change the default!
 - Dask is not much faster than Futures, but it occupies more memory
 - RAM usage is 1 GB/core or less for both Coffea and RDF, but Coffea is more memory-hungry
 - hyper-threading does not help coffea much (it can even be detrimental), it helps very little with RDF



Benchmark results

3. Throughput in dependence from a number of files

- both RDF and Coffea throughput is constant with respect to the number of files (as expected)
- for these implementations and benchmark setup, RDF is $\sim 2x$ faster
- RDF is between 1.25x and 2x faster depending on the number of files processed and the average file size
- neither application is optimized for performance. In particular, Coffea is known to benefit from chunk sizes larger than the current default



**THANK YOU FOR YOUR
ATTENTION!**