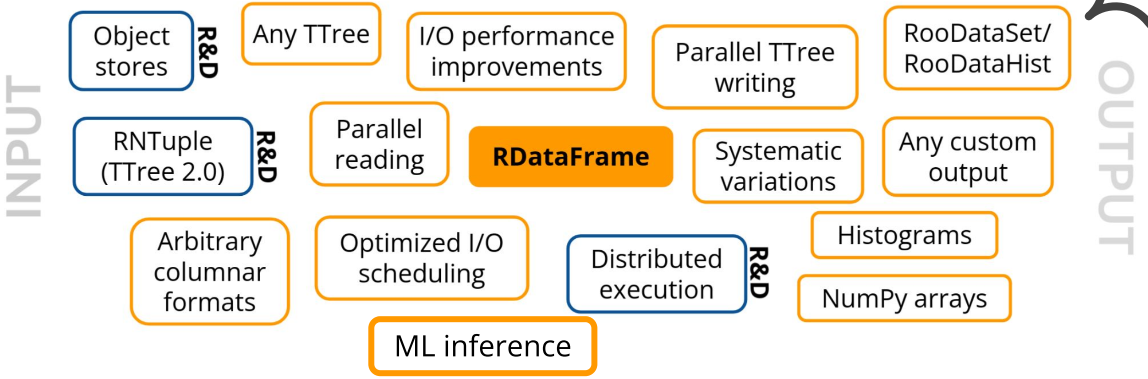
RDataFrame: interactive analysis at scale by example

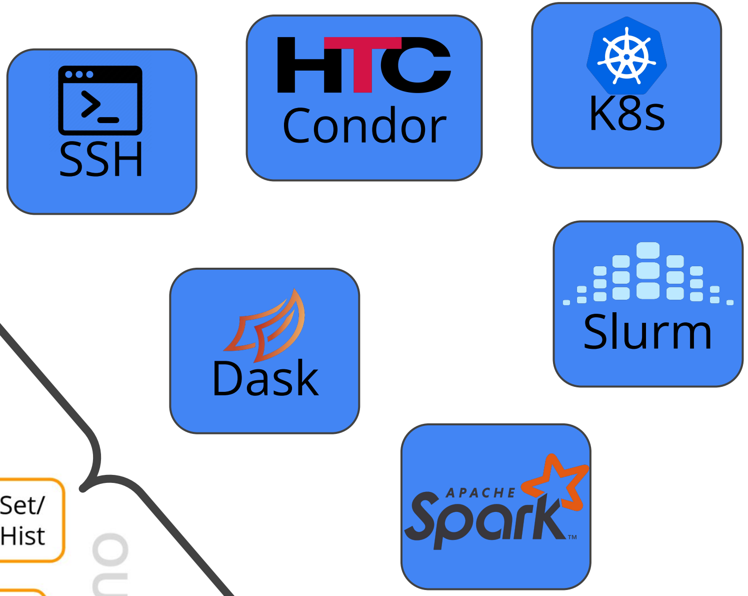
Padulano V.E., Taider S., Tejedor Saavedra E., Czurylo M., Boulis J., Guiraud E., Falko A.

PyHEP 2023

RDataFrame to scale

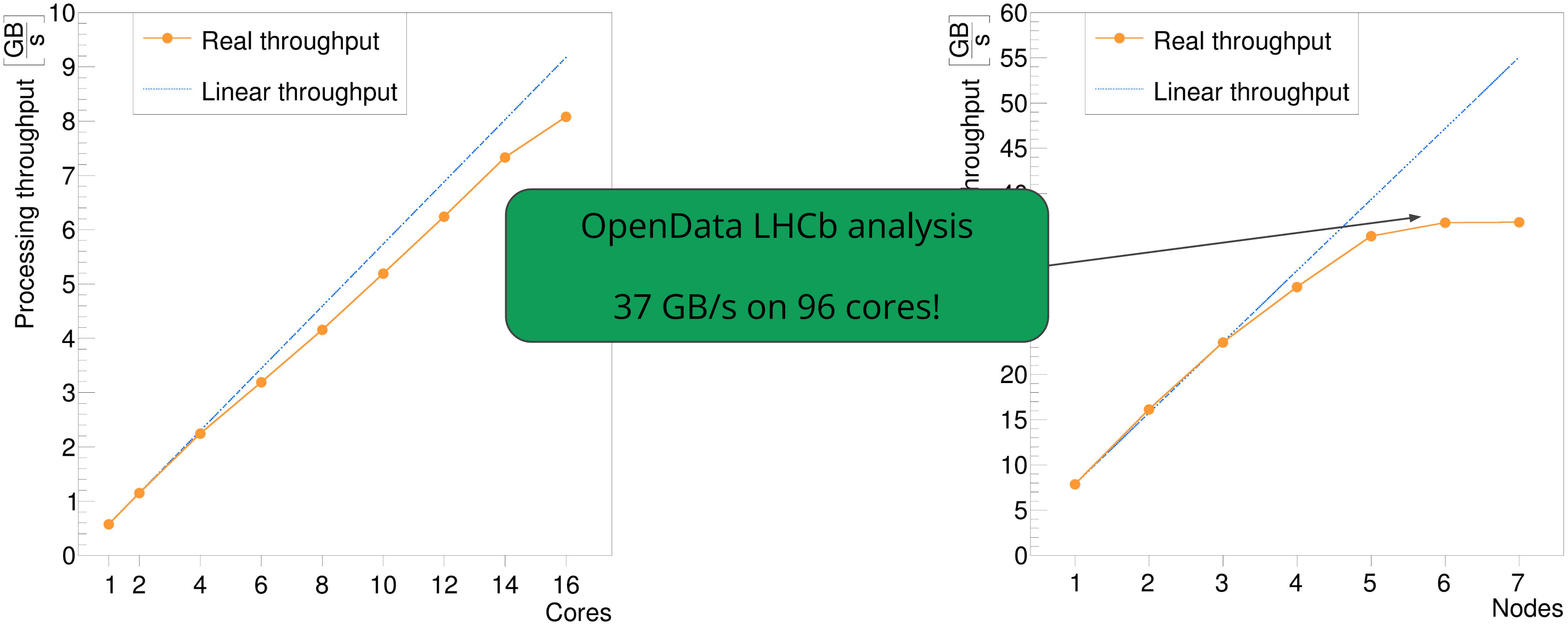
- RDataFrame: entry point to modern ROOT
- Seamless scaling out to computing clusters





RDataFrame + bleeding edge object store

Distributed RDataFrame on 1TB LHCb ntuples in a DAOS distributed object store [1]



Single node

Multi node (16 cores)

OpenData LHCb analysis

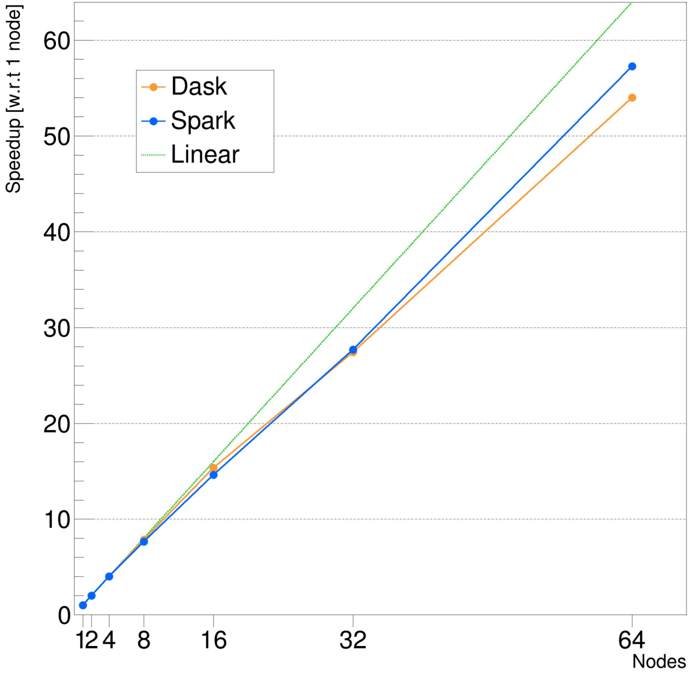
37 GB/s on 96 cores!

[1] Cluster Computing Journal paper

RDataFrame + HPC centers

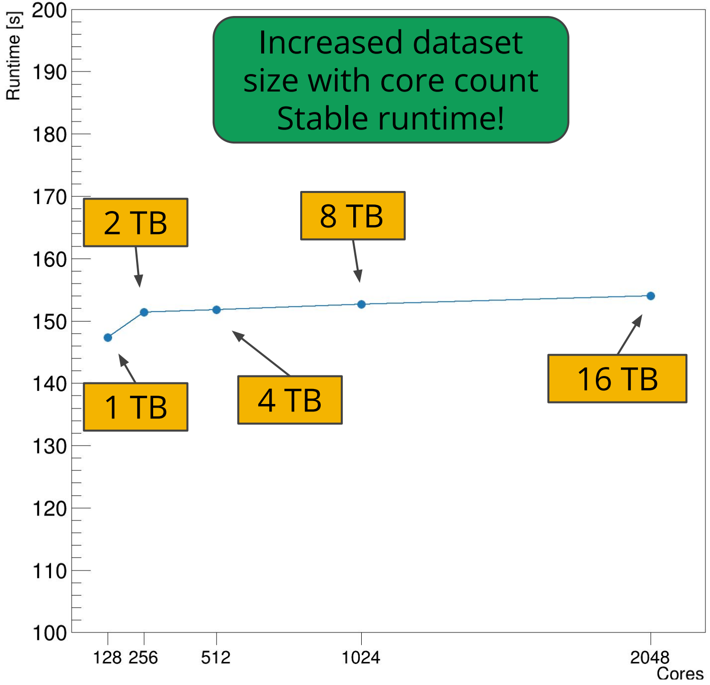
CERN HPC

- Slurm jobs (Spark/Dask)
- ~100 GB/s on 2048 cores
- JGC publication



Jülich HPC

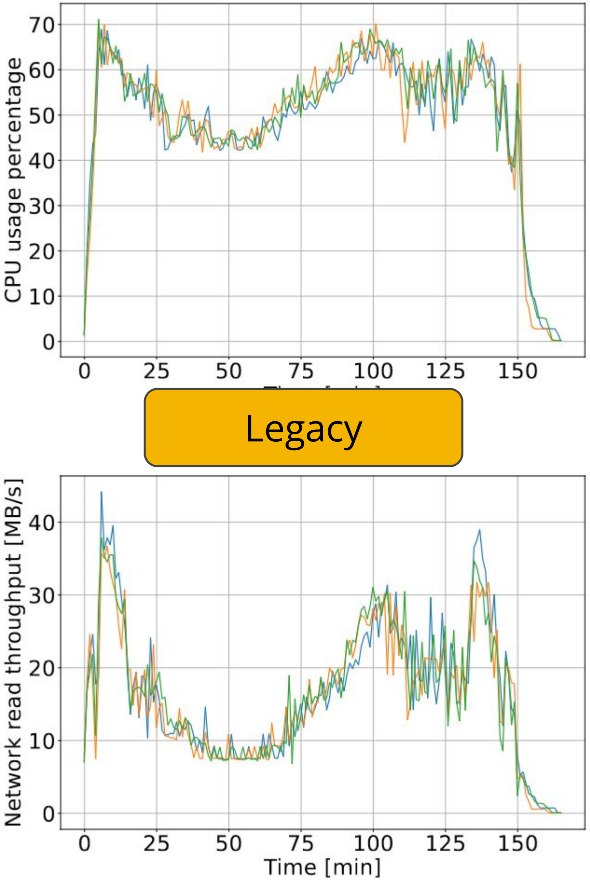
- Collaboration with OpenLab
- Slurm jobs (via Dask)
- Presentation



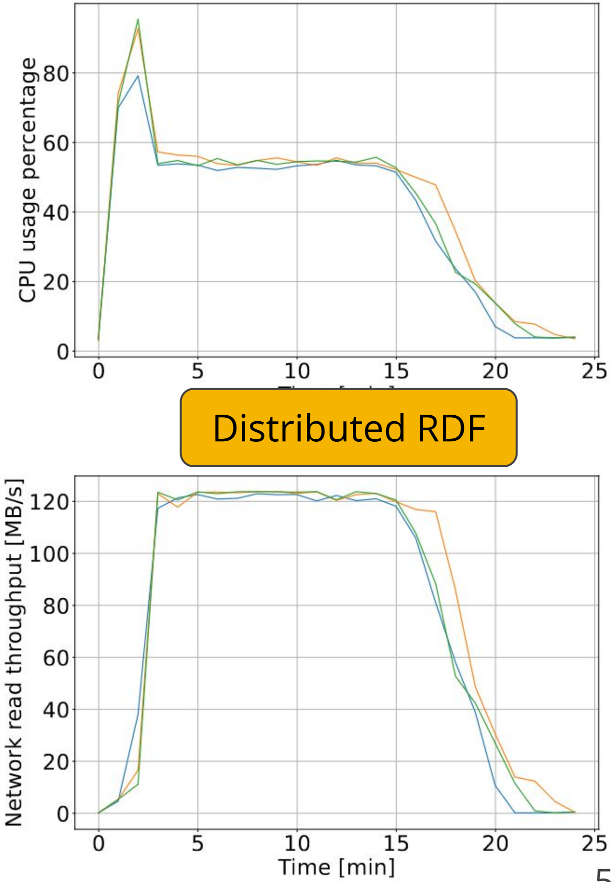
Increased dataset size with core count Stable runtime!

RDataFrame + CMS analysis facilities

- CMS production analysis
- Before: Python **for-loop** with NanoAODtools, **manual** job submission
- After: **Interactive** distributed RDataFrame
- O(10) speedup
- Publication



Legacy

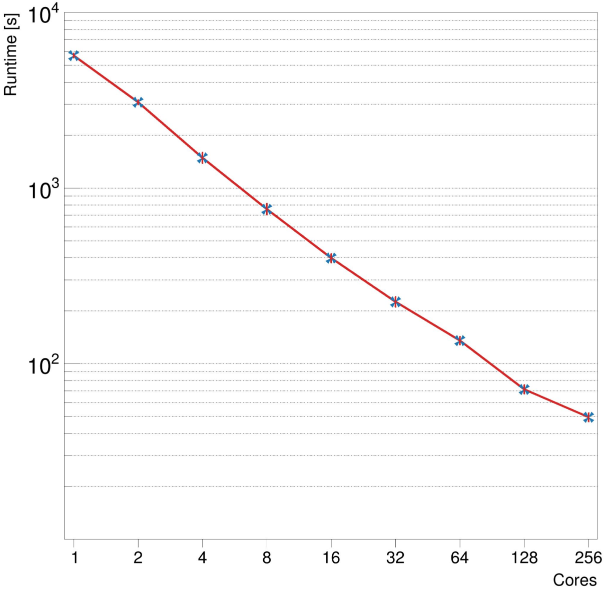


Distributed RDF

RDataFrame + Analysis Grand Challenge

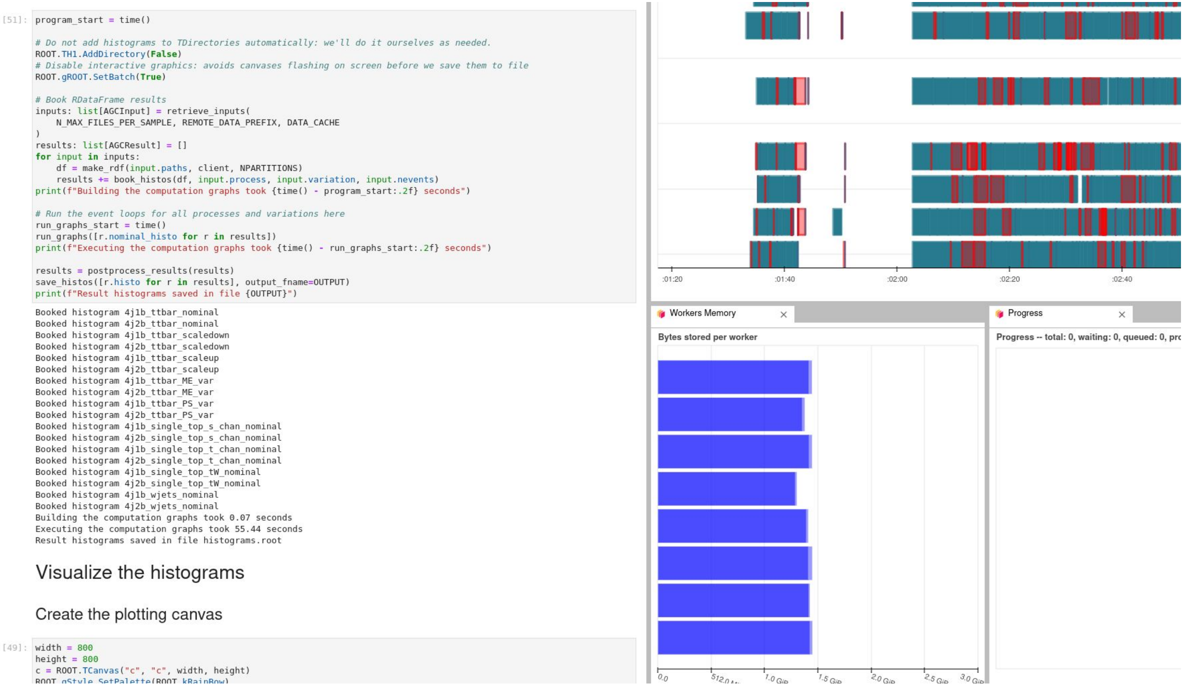
RDF+AGC on CERN HPC

- Demonstrated scalability
- ~50 seconds for the whole analysis on 256 cores
- CHEP'23 presentation



New! AGC on **SWAN**, scheduling with **Dask** on **CERN Condor** pools!
Rediscovering **existing** infrastructures and services in a modern way

cvmfs + EOS + CERN batch + ROOT $\overset{?}{=}$ CERN AF



```
[51]: program_start = time()

      # Do not add histograms to TDirectories automatically: we'll do it ourselves as needed.
      ROOT.TH1.AddDirectory(False)
      # Disable interactive graphics: avoids canvases flashing on screen before we save them to file
      ROOT.gROOT.SetBatch(True)

      # Book RDataFrame results
      inputs: list[AGCInput] = retrieve_inputs(
          N_MAX_FILES_PER_SAMPLE, REMOTE_DATA_PREFIX, DATA_CACHE
      )
      results: list[AGCResult] = []
      for input in inputs:
          df = make_rdf(input.paths, client, NPARTITIONS)
          results += book_histos(df, input.process, input.variation, input.nevents)
      print(f"Building the computation graphs took {time() - program_start:.2f} seconds")

      # Run the event loops for all processes and variations here
      run_graphs_start = time()
      run_graphs([r.nominal_histo for r in results])
      print(f"Executing the computation graphs took {time() - run_graphs_start:.2f} seconds")

      results = postprocess_results(results)
      save_histos([r.histo for r in results], output_fname=OUTPUT)
      print(f"Result histograms saved in file {OUTPUT}")
```

```
Booked histogram 4j1b_ttbar_nominal
Booked histogram 4j2b_ttbar_nominal
Booked histogram 4j1b_ttbar_scaledown
Booked histogram 4j2b_ttbar_scaledown
Booked histogram 4j1b_ttbar_scaleup
Booked histogram 4j2b_ttbar_scaleup
Booked histogram 4j1b_ttbar_ME_var
Booked histogram 4j2b_ttbar_ME_var
Booked histogram 4j1b_ttbar_PS_var
Booked histogram 4j2b_ttbar_PS_var
Booked histogram 4j1b_single_top_s_chan_nominal
Booked histogram 4j2b_single_top_s_chan_nominal
Booked histogram 4j1b_single_top_t_chan_nominal
Booked histogram 4j2b_single_top_t_chan_nominal
Booked histogram 4j1b_single_top_tW_nominal
Booked histogram 4j2b_single_top_tW_nominal
Booked histogram 4j1b_wjets_nominal
Booked histogram 4j2b_wjets_nominal
Building the computation graphs took 0.07 seconds
Executing the computation graphs took 55.44 seconds
Result histograms saved in file histograms.root
```

Visualize the histograms

Create the plotting canvas

```
[49]: width = 800
      height = 800
      c = ROOT.TCanvas("c", "c", width, height)
      ROOT.gStyle.SetPalette(ROOT.kRainBow)
```

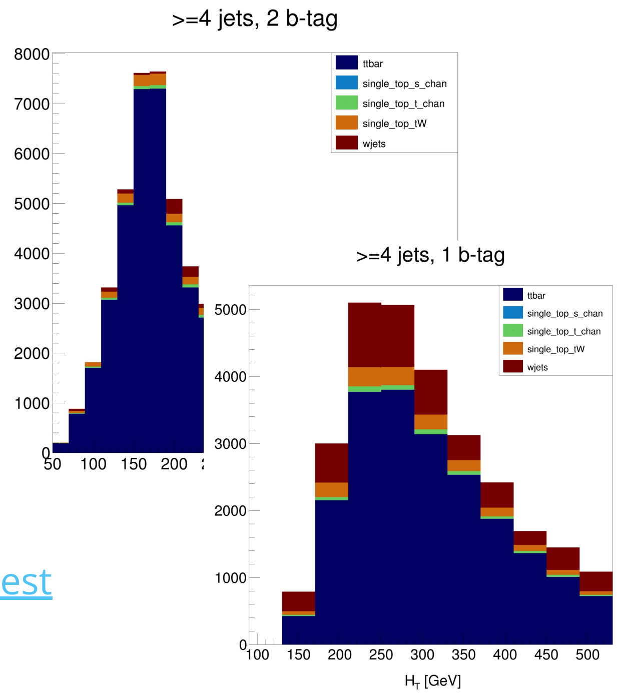
Progress -- total: 0, waiting: 0, queued: 0, prc

RDataFrame + Analysis Grand Challenge

Tight collaboration between ROOT team and IRIS-HEP.
One IRIS-HEP fellow, Andrii Falko, assigned to this task.

We tagged RDF AGC v1 this summer!

- local multi-thread and distributed Dask execution
- switched to the latest NanoAOD inputs
- ensured **bin-by-bin agreement** with reference implementation
- does not include statistical inference, left for later



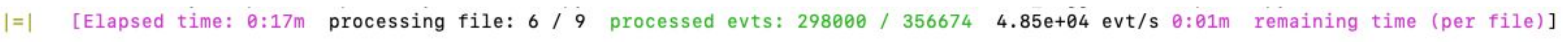
AGC **v2** still a moving target, however RDF implementation is being updated with no significant obstacles

- ML inference task has been added
- efficient BDT inference in multi-thread C++ event loop via FastForest
- plan to integrate FastForest upstream into ROOT

Progress bar

New! RDataFrame progress bar

```
ROOT.RDF.Experimental.AddProgressbar(df)
```



```
|=|   [Elapsed time: 0:17m  processing file: 6 / 9  processed evts: 298000 / 356674  4.85e+04 evt/s 0:01m  remaining time (per file)]
```

- Works in C++/Python, single-thread/**multi-thread**
- Support for distributed mode is coming
- Available in next ROOT release 6.30
- Check it out in the tutorials!

Live visualization

- Enable **real-time** visualizations while running distributed computations
- **No** need to **wait** for the whole 10k tasks to complete before seeing the plots!
- Available in next ROOT release 6.30
  - Check out the new tutorial!

```
# Pass a list or tuple of plots
LiveVisualize([graph, h_exp, tprofile_2d])

plot_callback_dict = {
        graph: set_marker,
        h_exp: fit_exp,
        tprofile_2d: None
}

# Pass a dictionary of plots and corresponding callback functions
LiveVisualize(plot_callback_dict)

# Trigger computations and see results instantly!
h_exp.Draw()
```

Conclusions

- RDataFrame: from one core to hundreds of machines
- Battle-tested with different analyses on different deployments
  - HPC clusters, **WLCG Tier 2s**, Existing CERN production services (**SWAN+CERN Condor**)
- Results backed by **publications** and **concrete contributions** to production ROOT releases