The University of Texas at Austin

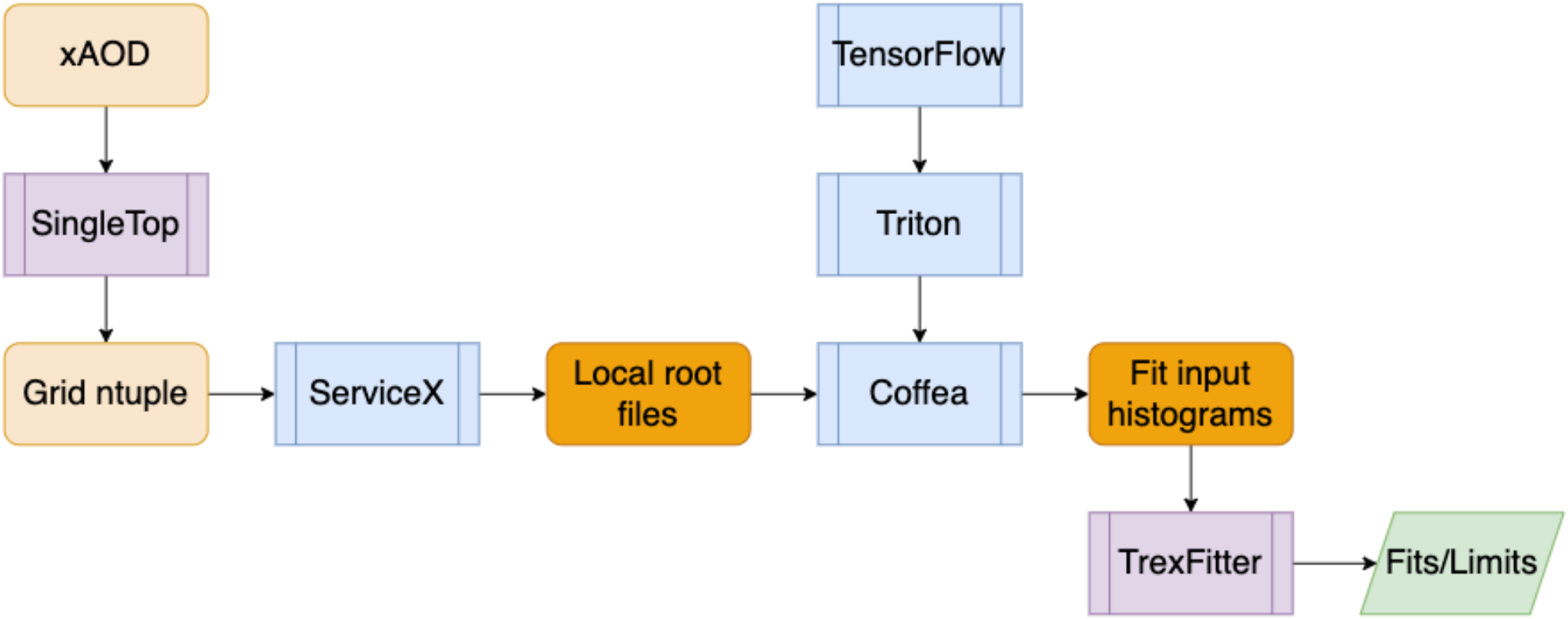
ATLAS EXPERIMENT

User experience: an ATLAS analysis using ServiceX + coffea

Marc Tost, KyungEon Choi

Overview

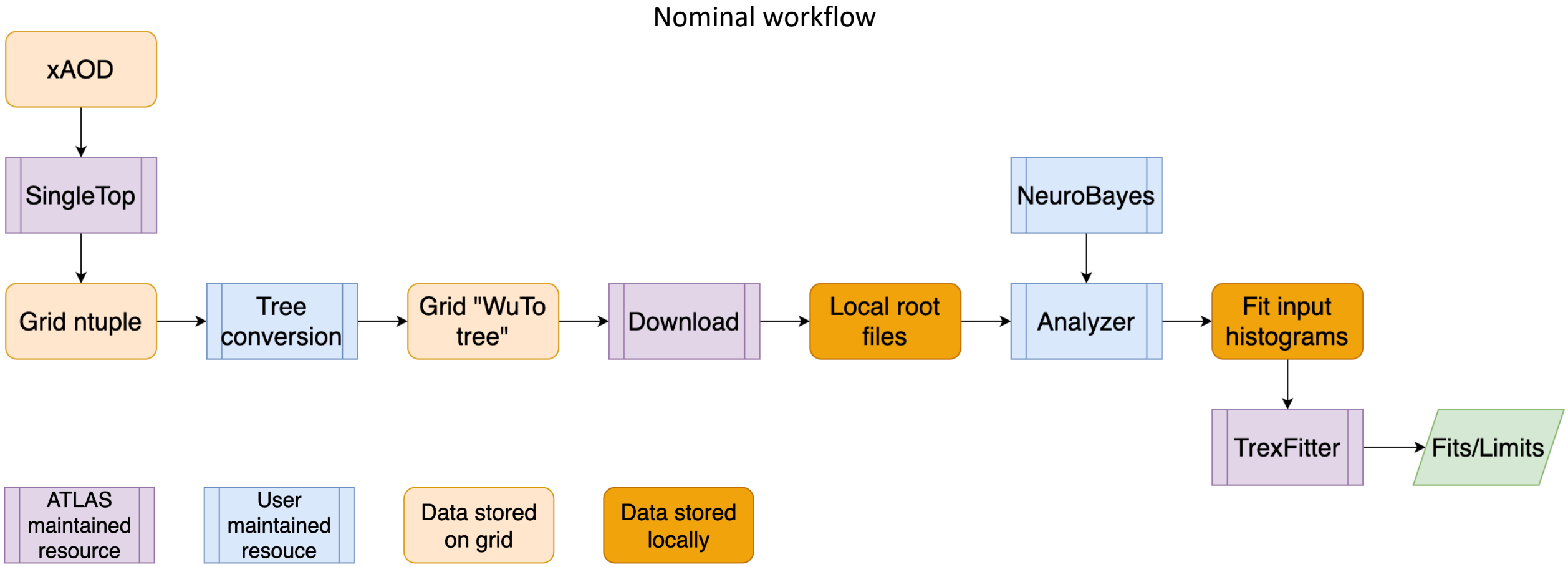
- Attempting to modernize workflow used for HEP-ex analyses
- Bringing iris-hep tools into an existing ATLAS analysis: FCNC tHq multi-lepton



Comparison of approaches

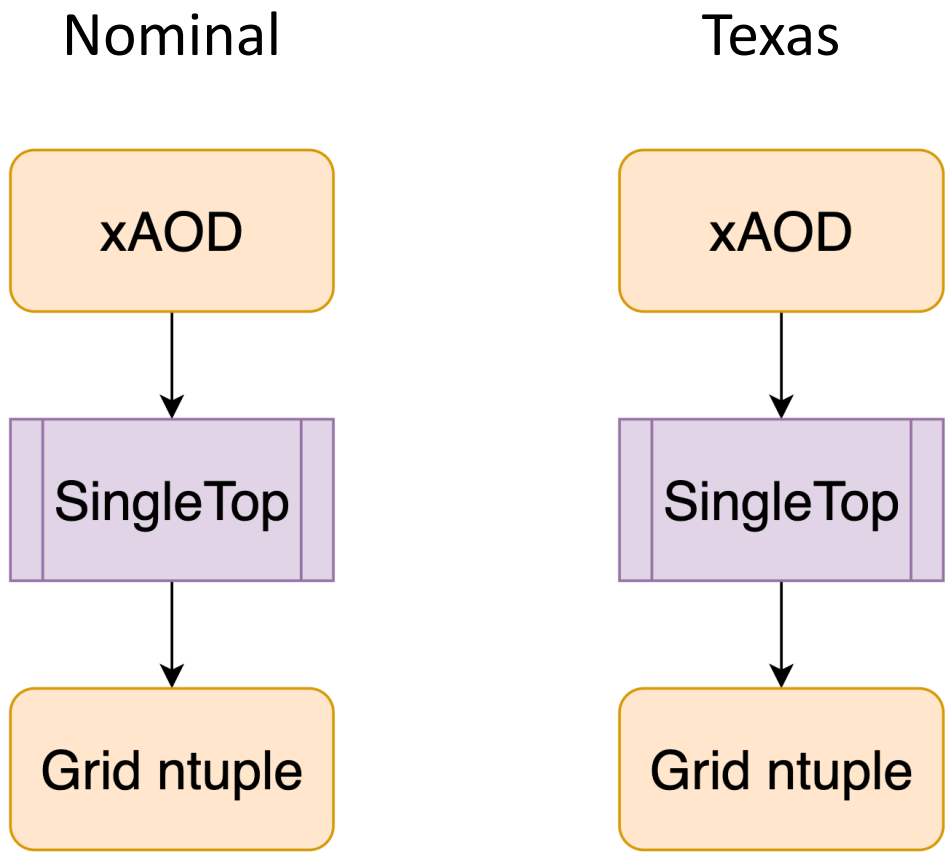
- Working together with a group from Wuppertal
  - The “nominal” version of the analysis is done using a standard approach for a search of this nature
  - Mostly written in C++ and using ATLAS Athena tools
  - Gives the opportunity to directly compare:
    - Runtime
    - Disk usage
    - Ease of use/readability
    - Physics results!

Comparison of approaches



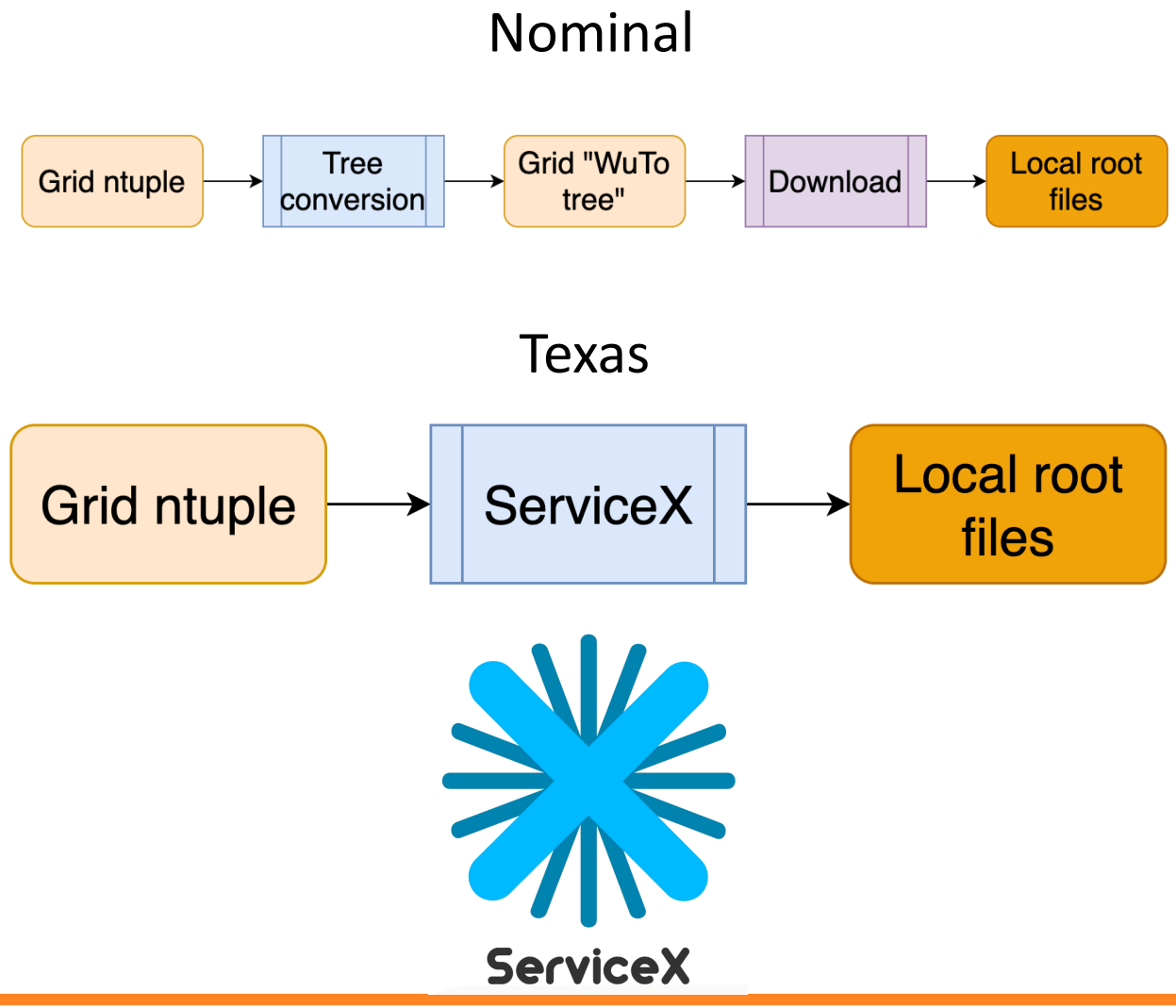
Comparison of approaches

- To access physics objects and uncertainties, both Data and MC needs to be run through an analyzer
  - Using SingleTop, though there are multiple other tools leading to the same place
  - Origin of tree structure
- Analyzed ntuples are shared to insure both groups start on same footing



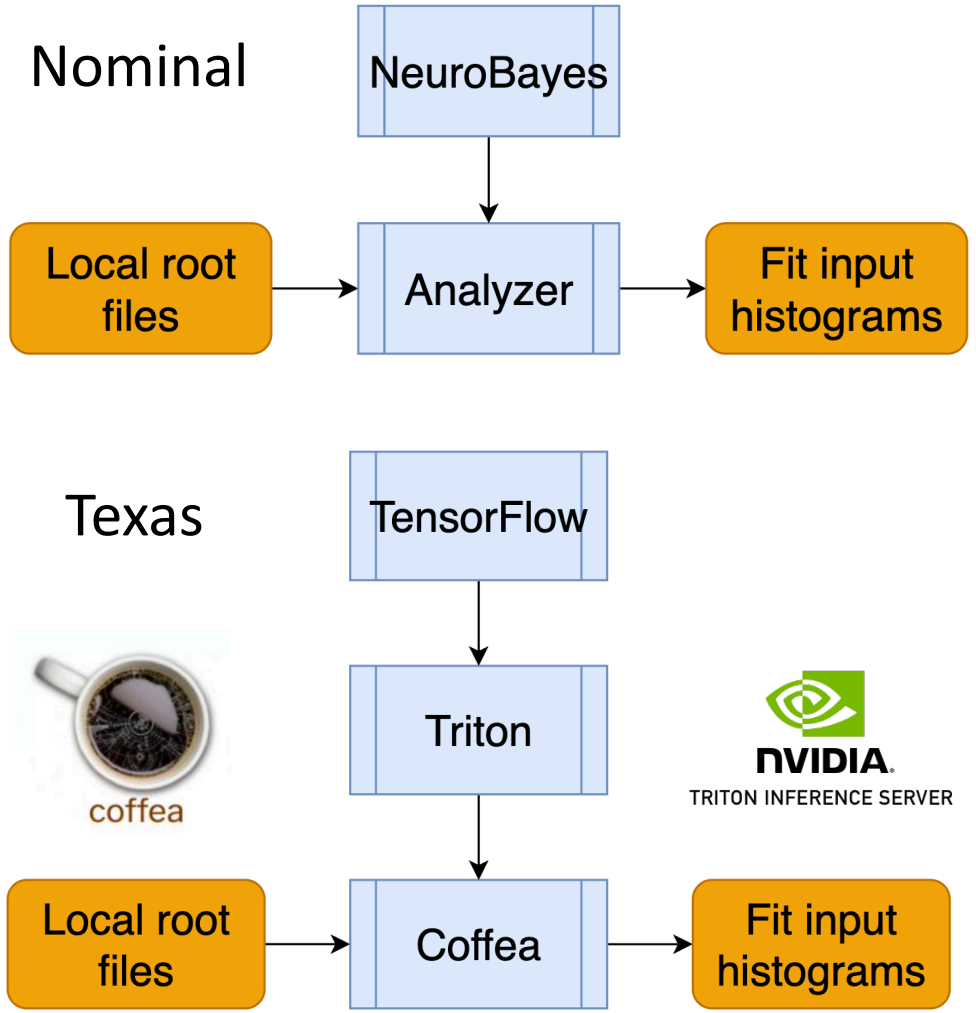
Comparison of approaches

- Nominal analysis needs a conversion process to create trees in the correct analysis format
  - No slimming/size reduction
  - These trees are downloaded from grid to local storage, a time consuming process
- ServiceX is able to combine the processing and downloading
  - Includes ability to apply pre-selection, reducing download time and disk impact
- Saves over 24 hours of computing time, as well as significant disk space (serviceX reduces by 4x)



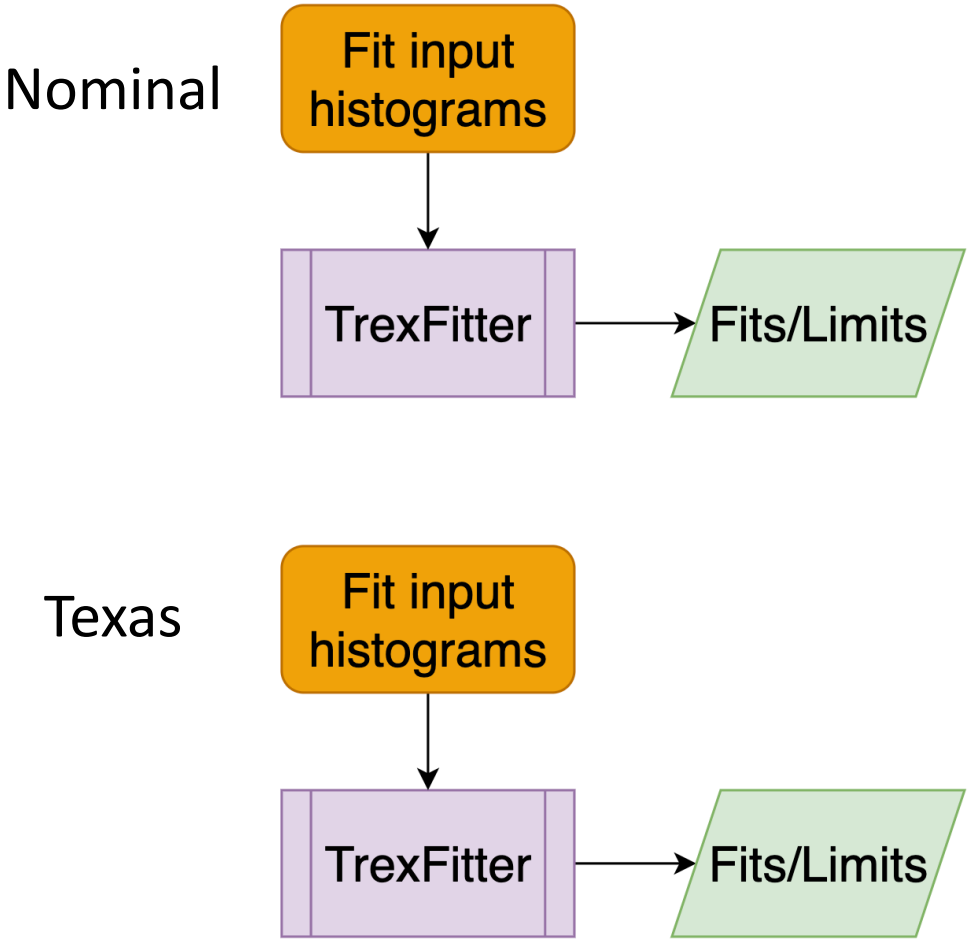
Comparison of approaches

- Nominal analysis uses for-loop style analysis code to process all events
  - NeuroBayes is used to create and apply a neural network classifier
- We use coffea to analyze data in columnar fashion
  - A series of TensorFlow classifiers are applied using Triton Inference Server, hosted on a GPU
- Runtime:
  - Nominal: 45-60 minutes (36 hours using condor jobs)
  - Texas: 3-5 minutes on coffea-casa (24 hours running back-to-back)



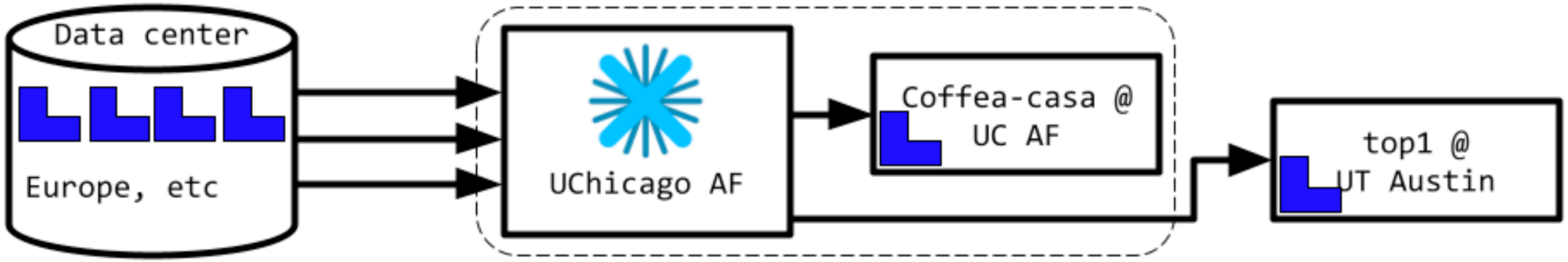
Comparison of approaches

- TRexFitter is used to complete fits to the data, and provides limits



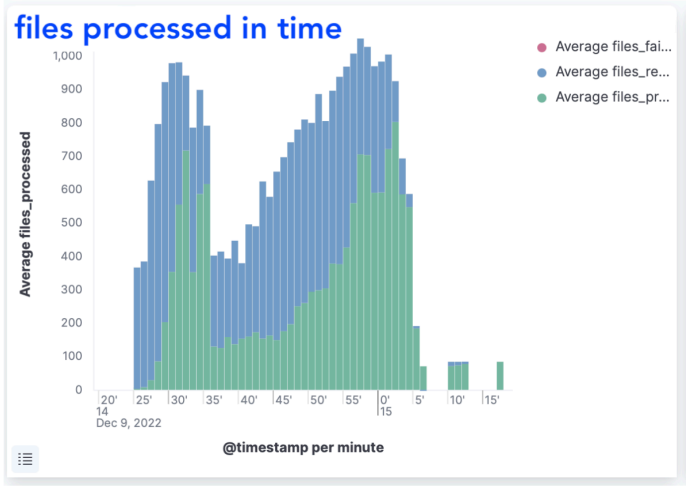
ServiceX

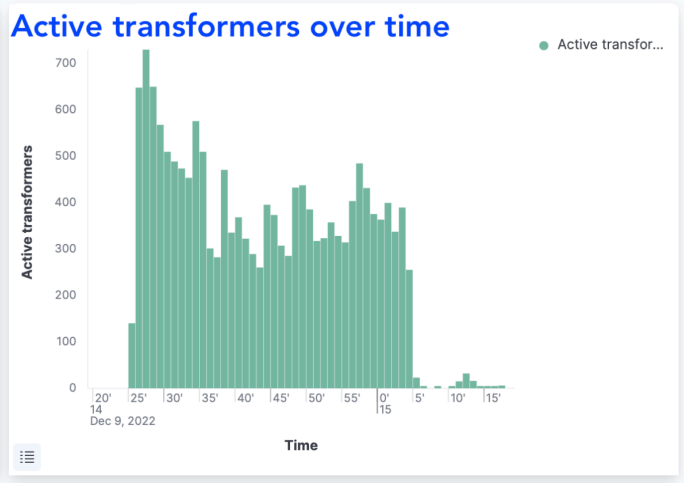
- Used for accessing and transforming data
- Inputs at grid level:
  - Flat ROOT ntuples
  - 1.1 TB of data spread over ~4k files
  - Ntuples contain over a hundred systematic trees (!!)



ServiceX (cont.)

- Using ServiceX DataBinder to consolidate requests into a single configuration file
- ServiceX automatically scales using built-in Kubernetes features to handle many requests at once
- Transform and delivery completed in under an hour (53 minutes with most recent setup)
- Caching further reduces runtime when small changes are made, making the user much more flexible





ServiceX (cont.)

- ATLAS ntuples contain hundreds of trees, each corresponding to an up/down fluctuation of a single systematic
- ServiceX is not optimized to handle this data management scheme, can only transform and deliver single trees
  - Need to deliver each tree as an individual file, and recombine at the local level
  - Increased disk impact, runtime (opening/closing the same file repeatedly), and inconveniences to the user
- Multi-tree capabilities likely necessary to increase use rate within ATLAS collaboration

Coffea

- Heart of the analysis
  - Used to apply selections, calculate kinematic distributions, apply event weights, and create histograms used for fitting and limit extraction
  - Ability to run in parallel massively improves total run time of the analysis
    - Currently using FuturesExecutor to automatically parallelize
- Faced many difficulties using non-ATLAS software, particularly in application of weights
  - Each event is scaled by ~15 weights, some of which are stored in separate trees
    - Those stored in nominal branch can be combined at ServiceX level, saving hassle and disk space
  - Have a functioning solution for *most* weight-related problems
    - Making use of a combination of user-specified meta-data inside coffea
    - Created .json files to give additional access to information such as k-factor, sum of weights, etc.

Coffea

- Analysis has 4 signal regions and 8 control regions
  - Selections are applied for all regions simultaneously
  - Creating a cut object (awkward array containing Booleans for each event) before applying selection insures the minimum possible mathematic operations
  - Selections can be applied to entire input dataset in ~1 minute
- Multi-tree limitations
  - Like ServiceX, coffea is only able to access one tree at a time
  - Systematics are stored in trees on single root file
  - Coffea process needs to be run in sequence for each systematic
  - Coffea is very fast for single tree, but repeating process for hundreds of systematics is currently the leading order impact on grid-to-insight time

Triton inference server

- Use of Nvidia's Triton Inference Server drastically improves runtime when applying ML tools to an analysis
  - FCNC tHq employs 4 independent signal/background classifiers, using TensorFlow
- Rather than saving models locally, they can be added to a remote repo and need no further user input
- Enables use of GPUs with minimal changes to analysis code
- Shown is runtime of nominal dataset with various configurations
  - When optimized, inferencing less than 5% of total runtime

```
Without triton:
Processing 100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8562/8562 [ 0:10:24 < 0:00:00 | 30.9  chunk/s ]
   Merging 100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1318/1318 [ 0:10:26 < 0:00:00 | ?    merges/s ]

With triton (30 instances 5000 chunk size):
Processing 100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8562/8562 [ 0:07:10 < 0:00:00 | 33.8  chunk/s ]
   Merging 100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1026/1026 [ 0:07:14 < 0:00:00 | ?    merges/s ]

With triton (70 instances 5000 chunk size):
Processing 100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8562/8562 [ 0:05:33 < 0:00:00 | 35.5  chunk/s ]
   Merging 100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━  881/881   [ 0:05:36 < 0:00:00 | ?    merges/s ]

With triton (140 instances 5000 chunk size):
Processing 100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8562/8562 [ 0:05:32 < 0:00:00 | 36.1  chunk/s ]
   Merging 100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━  889/889   [ 0:05:35 < 0:00:00 | ?    merges/s ]

With triton (70 instances 10,000 chunk size):
Processing 100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7336/7336 [ 0:08:07 < 0:00:00 | 36.6  chunk/s ]
   Merging 100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━  966/966   [ 0:08:09 < 0:00:00 | ?    merges/s ]
```

Run at coffea-casa with 70 workers

Conclusion

- Presented is an iris-hep centric alternative to a traditional ATLAS analysis
- Total runtime is drastically improved:
  - Nominal grid-to-insight: 2.5 days (24 hours for download/reprocessing, 36 hours for analysis) + 48 hours for fitting
  - Texas grid-to-insight: 1 day (1 hour for download/reprocessing, 24 hours for analysis + 48 hours for fitting
- ServiceX leading order time-saver
- Code is much easier to read and manipulate, 95% simple python scripts/notebooks
- Differences in handling of systematics between CMS/ATLAS biggest obstacle for widespread adoption
  - Multi-tree capabilities have potential to further reduce runtime and ease of use for ATLAS users

Backup

Root vs. parquet

- Have delivered data as both ROOT and parquet files
  - Performance is similar at both serviceX and coffea stages
  - However, ability to manipulate ROOT files locally is a big advantage
  - After delivering files, hadd can be used to combine all files with systematic trees
  - Parquet is difficult to manipulate without complex scripts

DataBinder Configuration File

```
General:
    ServiceXBackendName: uproot_uc_af
    OutputDirectory: /data_ceph/kyungeon/fcnc_tHq_ML/ServiceXData_v7
    OutputFormat: root
    WriteOutputDict: out_paths_v7

Sample:
  - Name: ttH
    RucioDID: user.kchoi:user.kchoi.fcnc_tHq_ML.ttH.v7
    TransformerImage: kyungeonchoi/servicex_func_adl_uproot_transformer:fcnc_nominal
    Tree: nominal
    FuncADL: DEF_funcadl_prompt_NOMINAL

  - Name: ttH
    RucioDID: user.kchoi:user.kchoi.fcnc_tHq_ML.ttH.v7
    TransformerImage: kyungeonchoi/servicex_func_adl_uproot_transformer:fcnc_fullsim
    Tree: sys_fullsim
    FuncADL: DEF_funcadl_prompt_SYS

    ...

Definition:
  DEF_funcadl_prompt_NOMINAL: "Where(lambda e: e.el_truthIFFClass.Where(lambda i: i==2).Count() == e.el_truthIFFCl
  Where(lambda e: e.mu_truthIFFClass.Where(lambda i: i==4).Count() == e.mu_truthIFFClass.Count()).
  Select(lambda e: {'mu_pt': e.mu_pt, 'mu_eta': e.mu_eta, 'mu_phi': e.mu_phi, 'mu_e': e.mu_e, 'mu_charge': e.mu_ch
                    'el_pt': e.el_pt, 'el_eta': e.el_eta, 'el_phi': e.el_phi, 'el_e': e.el_e, 'el_charge': e.el_ch
                    'met_met': e.met_met, 'met_phi': e.met_phi,
                    'jet_pt': e.jet_pt, 'jet_eta': e.jet_eta, 'jet_phi': e.jet_phi, 'jet_e': e.jet_e, 'jet_tagWeig
                    'weights': e.weight_mc*e.weight_pileup*e.weight_leptonSF*e.weight_bTagSF_DL1r_Continuous*e.wei
                    'mcChannelNumber': e.mcChannelNumber, 'runNumber': e.runNumber, 'eventNumber': e.eventNumber,
                    'mc_generator_weights': e.mc_generator_weights,
```