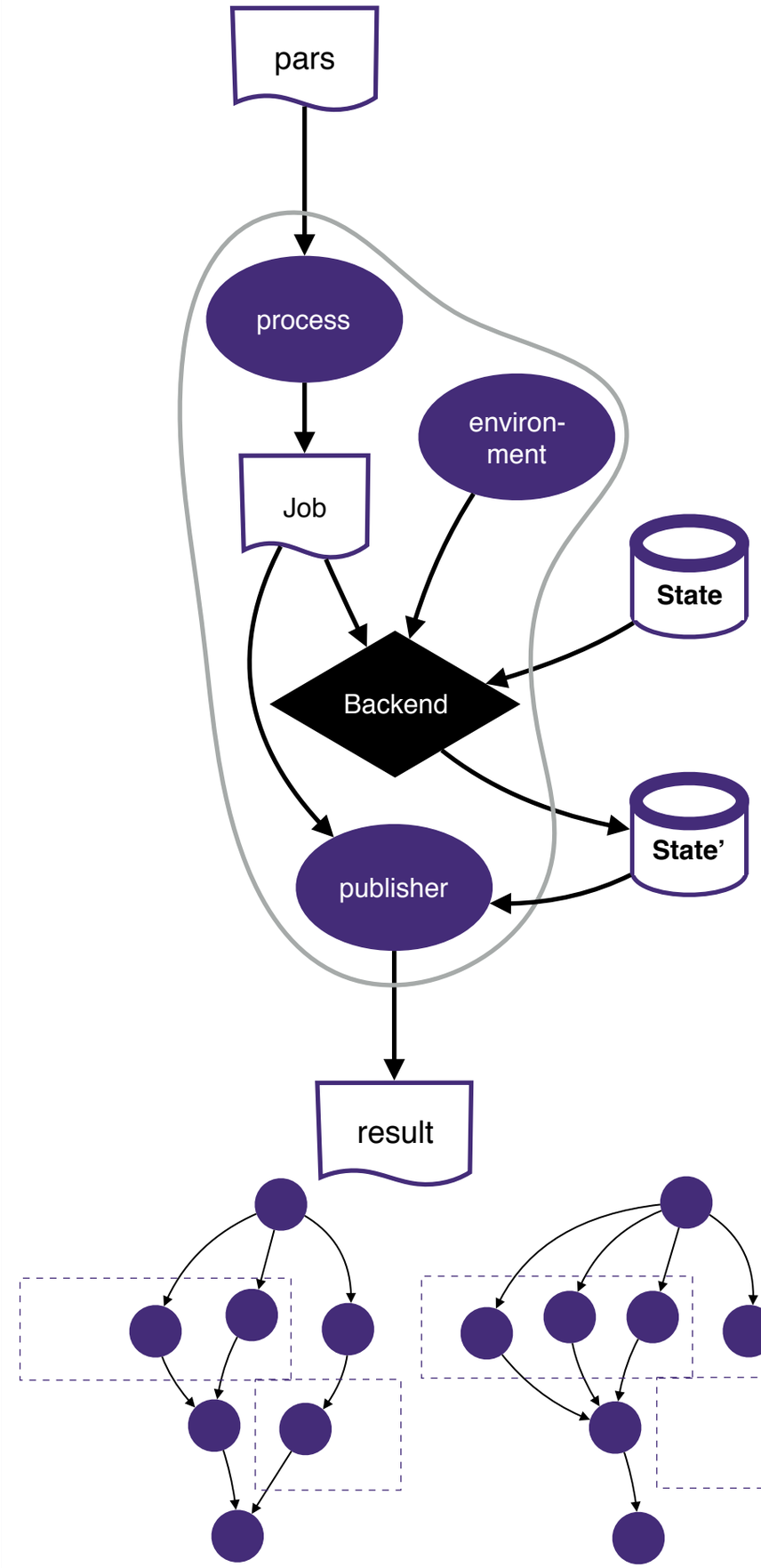
Analysis Preservation and Reinterpretation

Lukas Heinrich

CHEP 2016, San Francisco

NEW YORK UNIVERSITY



Analysis Ingredients:

- ↪ measurement of rates and distribution of *pp collision* observables
- ↪ estimation of expected measurements under model hypotheses
- ↪ statistical evaluation, interval estimation on model parameters

$$\text{result} = f_{\text{analysis}}(\text{data}|\text{model})$$

observable distributions, confidence intervals on model parameters

reconstruction, event selection, stat. evaluation

collision data from LHC detector

model hypothesis (SM, many SUSY models, etc..)

Current Standards / Preservation / Archival efforts:

Efforts like DASPOS and CERN Analysis Preservation try to fill a gap in the preservation efforts.
Data only is not enough.We need software and environment as well.

HepData | **???** | **DPHEP CERN OpenData** | **UFO/SLHA** (HepData?)

$$\text{result} = f_{\text{analysis}}(\text{data}|\text{model})$$

observable distributions, confidence intervals on model parameters

reconstruction, event selection, stat. evaluation

collision data from LHC detector

model hypothesis (SM, many SUSY models, etc..)

NEW YORK UNIVERSITY

Analysis should be preserve the analysis in a **parametrized form**, not only its concrete application, separately from its application on given dataset.

$$f_{\text{analysis}}(\,\cdot\,), \quad \text{data}, \quad \text{model}$$

Given a **parametrized preservation of an analysis** (even w/ fixed data), we gain ability to extract **new results** using existing resources.

Reinterpretation of Single Analysis under multiple models

Combination of multiple analyses w.r.t. one model
(increased stat. power)

$$\begin{array}{ll} f_{\text{a}}(\text{data}|\text{model}_1) & f_{\text{a}}(\text{data}|\text{model}_2) \\ f_{\text{b}}(\text{data}|\text{model}_1) & f_{\text{b}}(\text{data}|\text{model}_2) \\ f_{\text{c}}(\text{data}|\text{model}_1) & f_{\text{c}}(\text{data}|\text{model}_2) \end{array}$$

How to preserve $f_{\mathrm{analysis}}(\,\cdot\,)$?

Issues:

- preserving software environments
- preserving execution instructions (what programs/scripts to call, etc)
- Analyses consist of many individual steps, possibly complex dependency structure
- mutual incompatibility of environments between steps
- composability / sharing across analyses
- many more…

Strategy:

1. Preserve individual processing steps
2. Preserve workflow / order between steps

How to preserve $f_{\mathrm{analysis}}(\,\cdot\,)$?

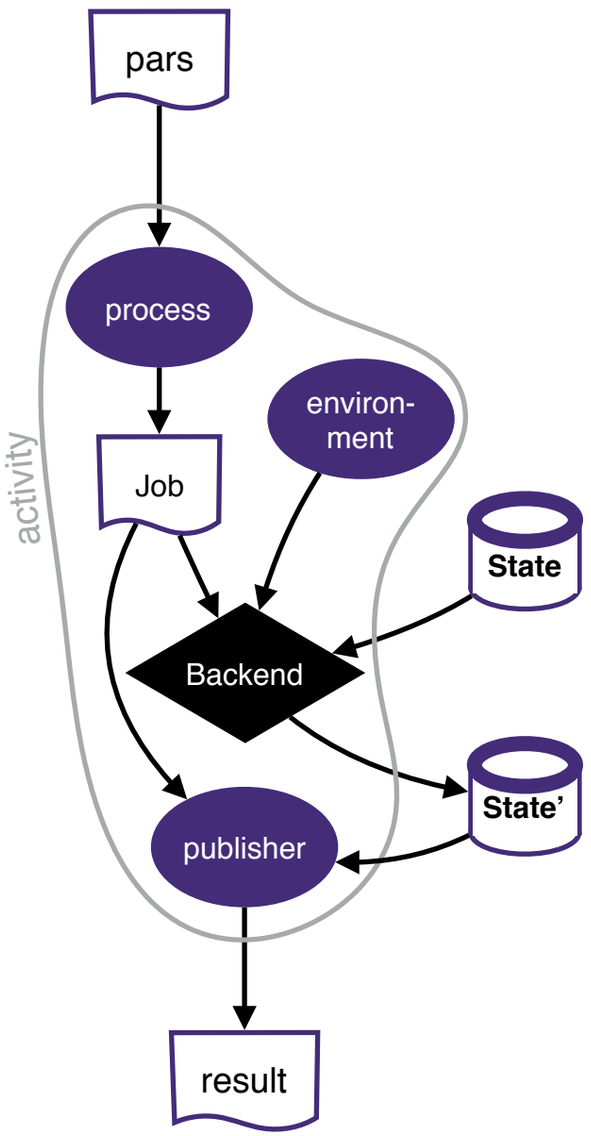
1. Problem: Preserve Individual Processing Steps

(Example: Run Detector Simulation + Reconstruction on MC events)

Steps (“activities”) process data obtained by a global state, and modify state with (eg. writing new files, modify existing files)

$$\mathrm{result\ data},\ \mathrm{state}' = g_{\mathrm{step}}(\mathrm{state}, \mathrm{parameters})$$

It’s useful to have machine readable result data to e.g. identify newly created files.



Three ~orthogonal ingredients that can be described individually:

parametrized process:
template job from which we can produce concrete job
template: “`./DelphesHepMC <input file> <output file>`”
concrete: “`./DelphesHepMC /input/file/path.hepmc /output/file.root`”

environment:
description of computing env in which above job can run. Multiple options, promising: *Linux Containers* (investigating Umbrella, etc)

publisher:
recipe how to extract parsable result data after job completion
e.g. globbing files in a work directory

How to preserve $f_{\mathrm{analysis}}(\,\cdot\,)$?

1. Problem: Preserve Individual Processing Steps

(Example: Run Detector Simulation + Reconstruction on MC events)

Data Format: JSON

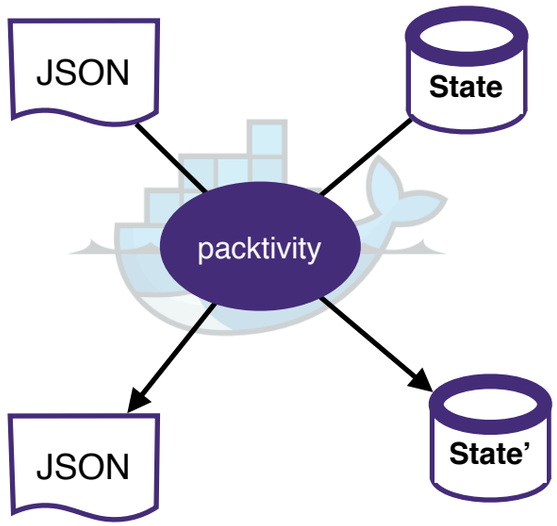
- as interchange format for parameters and result data
- as declarative description format for *process/env/publisher*
  - incl. JSON schemas for validation

Essentially, a self-consistent "packaged activity" — a "packtivity"

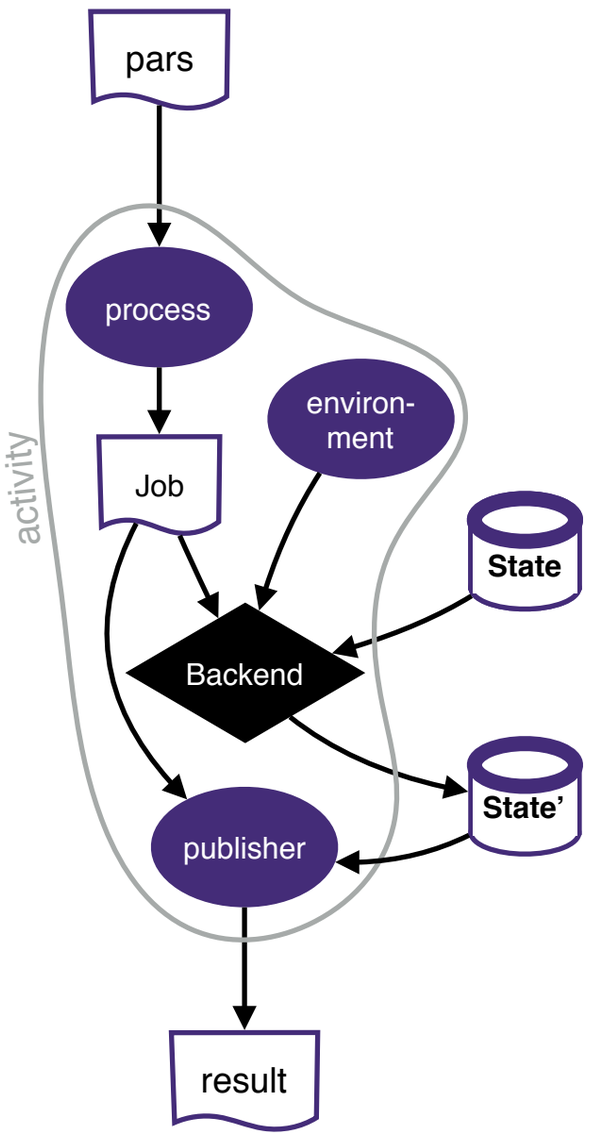
- JSON API
- archivable, declarative description as JSON
- dependencies captured in environment
  - e.g. Docker Image

$$\text{result data},\ \text{state}' = g_{\mathrm{step}}(\text{state}, \text{parameters})$$

=



=



How to preserve $f_{\mathrm{analysis}}(\cdot)$?

1. Problem: Preserve Individual Processing Steps

(Example: Run Detector Simulation + Reconstruction on MC events)

Example:

```
process:
  process_type: 'string-interpolated-cmd'
  cmd: 'DelphesHepMC  {delphes_card} {outputroot} {inputhepmc}'
publisher:
  publisher_type: 'frompar-pub'
  outputmap:
    rootfile: outputroot
environment:
  environment_type: 'docker-encapsulated'
  image: lukasheinrich/root-delphes
```

CLI tool

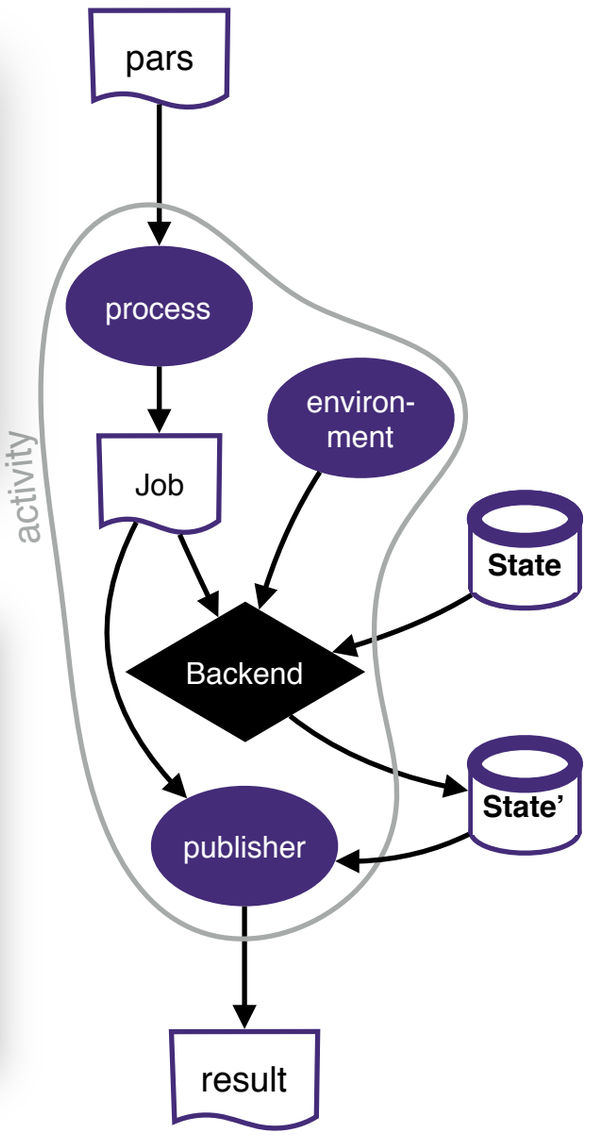
```
172-27-219-223 — -zsh — 64×12
[$>  ls
delphes.yml  input.hepmc  pars.yml
[$>  pygmentize -g pars.yml
delphes_card: 'delphes/cards/delphes_card_ATLAS.tcl'
inputhepmc: '{workdir}/input.hepmc'
outputroot: '{workdir}/out.root'
[$>  packtivity-run delphes.yml pars.yml
{'rootfile': '/Users/lukas/chep2016/out.root'} (prepublished)
$>
```

python bindings

```
pack.py — /Users/lukas/chep2016
pack.py
1   import os
2   import capschemas
3   from packtivity import packtivity
4   from packtivity.statecontexts.poxisfs_context import make_new_context
5   packtivity_description = capschemas.load(
6       'delphes.yml',os.curdir,
7       'packtivity/packtivity-schema')
8   pars = {
9     delphes_card: 'delphes/cards/delphes_card_ATLAS.tcl',
10    inputhepmc: '{workdir}/input.hepmc',
11    outputroot: '{workdir}/out.root'
12  }
13  packtivity(packtivity_description,parameters,make_new_context(os.curdir))
Line 0  File 0  Project 0  ✔ No Issues  pack.py*  15:1  LF  UTF-8  Python  3 updates
```

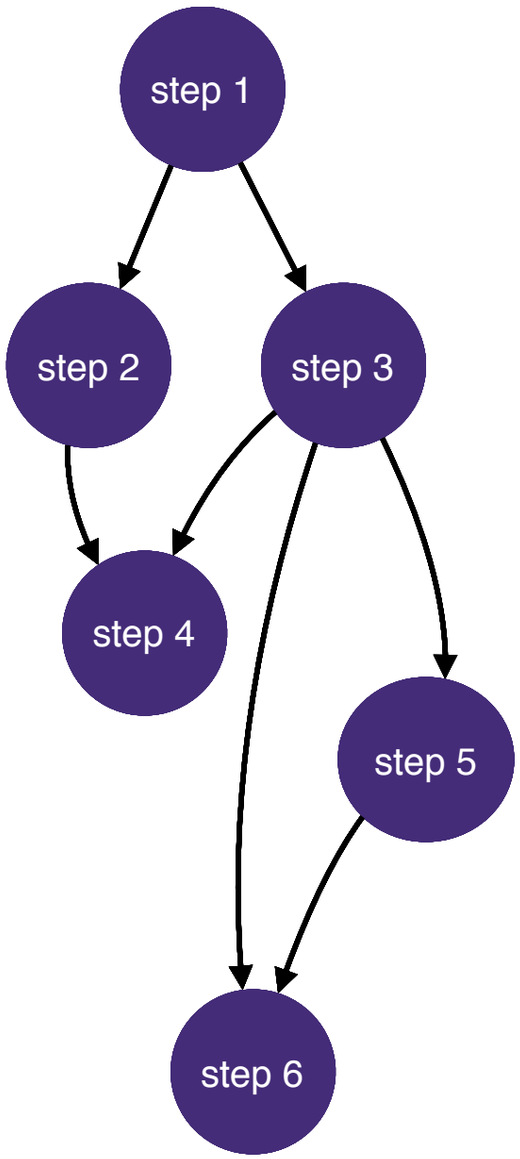
python package: “packtivity”

- executes packtivities according to JSON spec for given parameters
- cli tool and python bindings
- multi-host / remote execution ready via e.g. Docker Swarm



How to preserve $f_{\mathrm{analysis}}(\,\cdot\,)$?

2. Problem: Preserve Parametrized Workflow



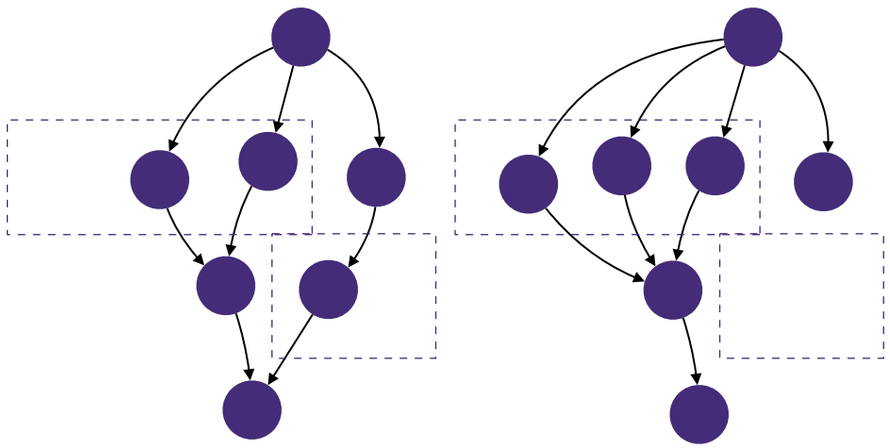
Natural Data Model: *directed acyclic graphs (DAGs)*

- **nodes**: individual steps
- **edges**: dependency relations

Two place where parametrization enter:

1. individual steps parametrized: covered by “packtivities”
   graph topology may *depend on the parameters* of the analysis and only emerge during run-time

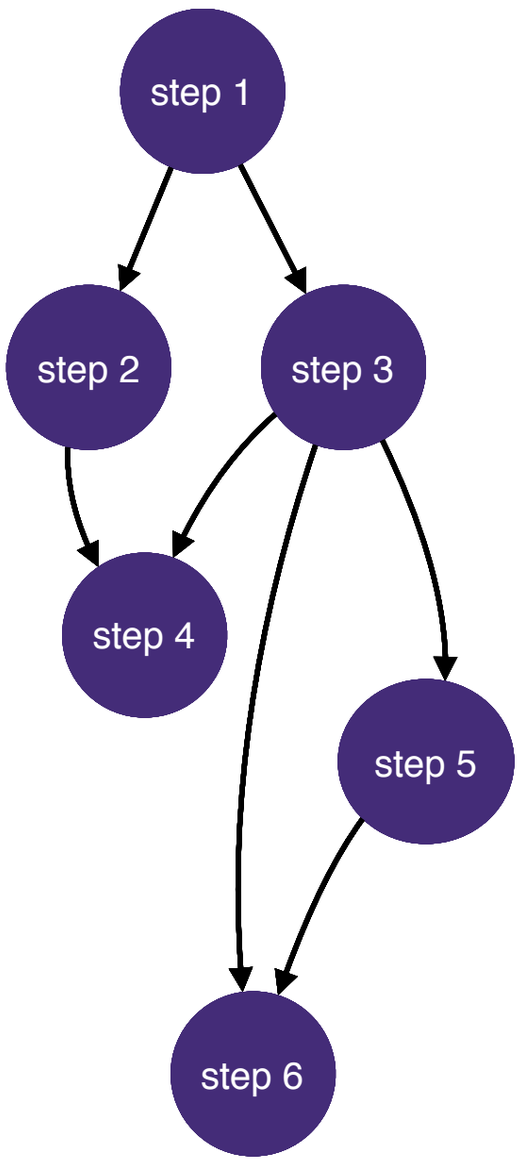
2. Examples:
   - variable number of created files during execution,
   - conditional choices (if/else)/flags do enable/disable steps, e.g. run systematics / not



Par. Set 1 **Par. Set 2**

How to preserve $f_{\text{analysis}}(\cdot)$?

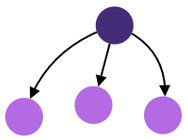
2. Problem: Preserve Parametrized Workflow



Therefore: Sequentially build up graph, as sufficient information becomes available, using a number of stages that add nodes and edges

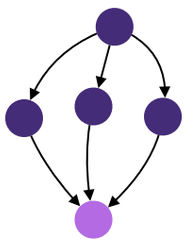
To capture analysis workflow, capture the stages.

Example: Parametrized Map-Reduce

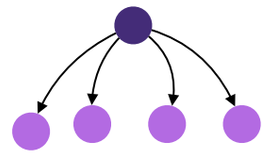


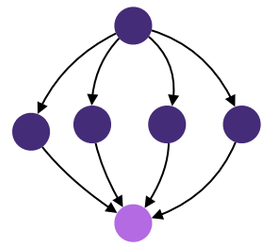
Stage 1:
unknown number of files. e.g. download & unpack archive with a priori unknown # of files

Stage 2:
for each file in the archive, add node to process it
(**only possible after first node done)**



Stage 3:
add a node that merges results of the map nodes
node/edge can be added before execution of map nodes



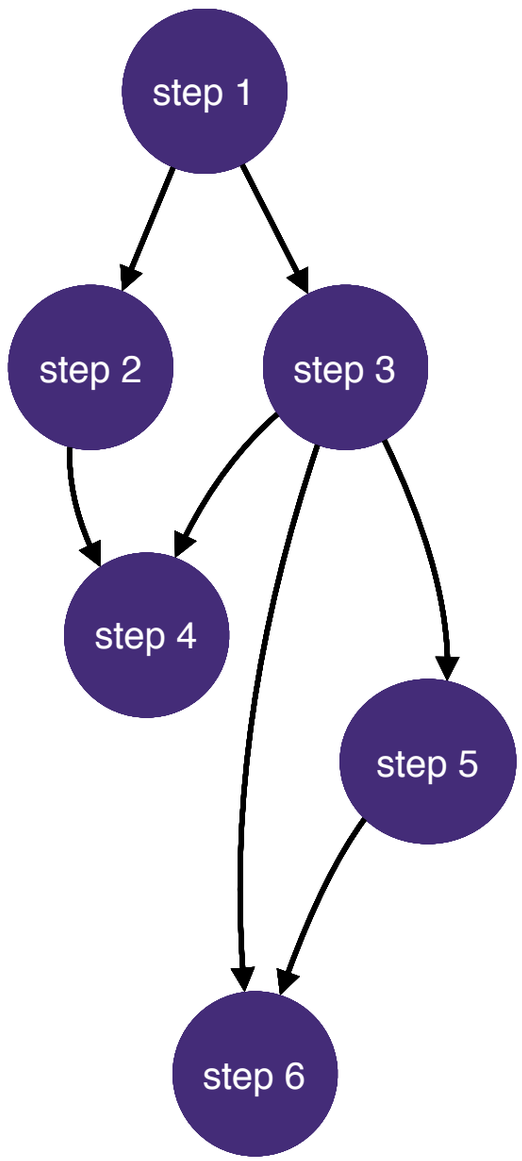


Par. Set 1

Par. Set 2

How to preserve $f_{\text{analysis}}(\cdot)$?

2. Problem: Preserve Parametrized Workflow



Stages captured in JSON (incl. schemas for validation)

- easy to archive in repositories (e.g. GitHub / CERN Analysis Preservation
- easy to parse for clients

```yaml
stages:
  - name: acquisition
    dependencies: ['init']
    scheduler:
      scheduler_type: singlestep-stage
      parameters:
        source: {stages: init, output: sourcefile, unwrap: true}
        localname: '{workdir}/mydata.zip'
      step: {$ref: 'steps.yml#/acquire'}
  - name: map
    dependencies: ['acquisition']
    scheduler:
      scheduler_type: multistep-stage
      parameters:
        inputfile: {stages: 'acquisition', output: 'unzipped', unwrap: true}
        outputfile: '{workdir}/mapout.txt'
      scatter:
        method: zip
        parameters: ['inputfile']
      step: {$ref: 'steps.yml#/mapstep'}
  - name: reduce
    dependencies: ['map']
    scheduler:
      scheduler_type: singlestep-stage
      parameters:
        mapoutputs: {stages: map, output: mapout}
        combinedfile: '{workdir}/out.txt'
      step: {$ref: 'steps.yml#/reducestep'}
```

Stage 1:
unknown number of files. e.g. download & unpack archive with a priori unknown # of files

Stage 2:
for each file in the archive, add node to process it

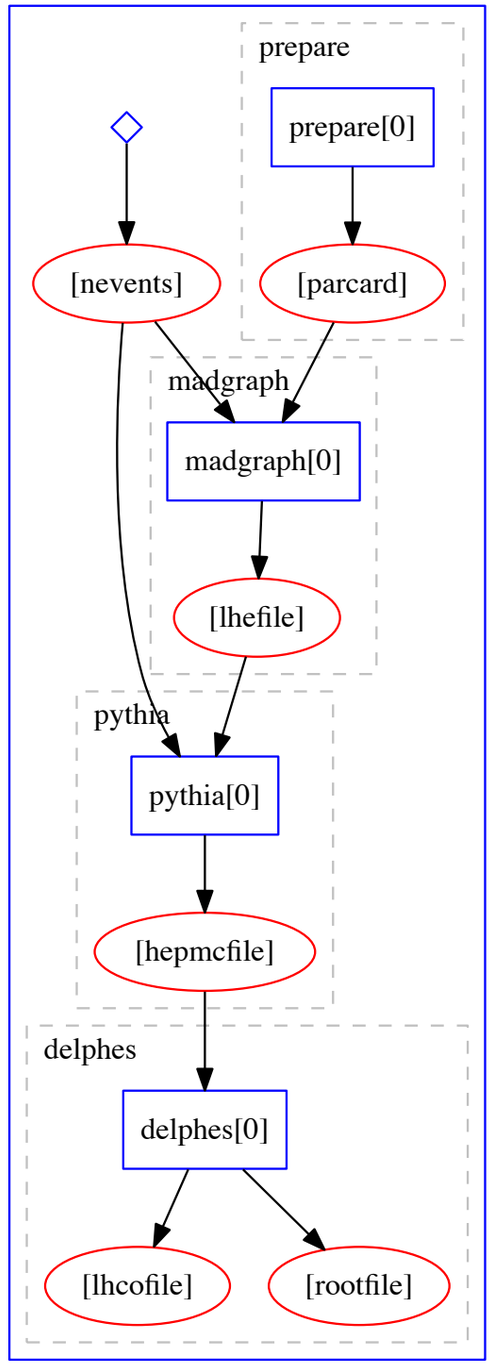
Stage 3:
add a node that merges results of the map nodes (node/edge can be added before execution of map nodes

How to preserve $f_{\mathrm{analysis}}(\,\cdot\,)$?

2. Problem: Preserve Parametrized Workflow

First implementation: **Yadage**

python package.
install from **PyPi** or **…**
… run from from *within docker*
can run complex workflows with vastly different environments on hosts that know nothing about the domain software out of the box just need docker.



172-27-219-223 — -zsh — 99×9

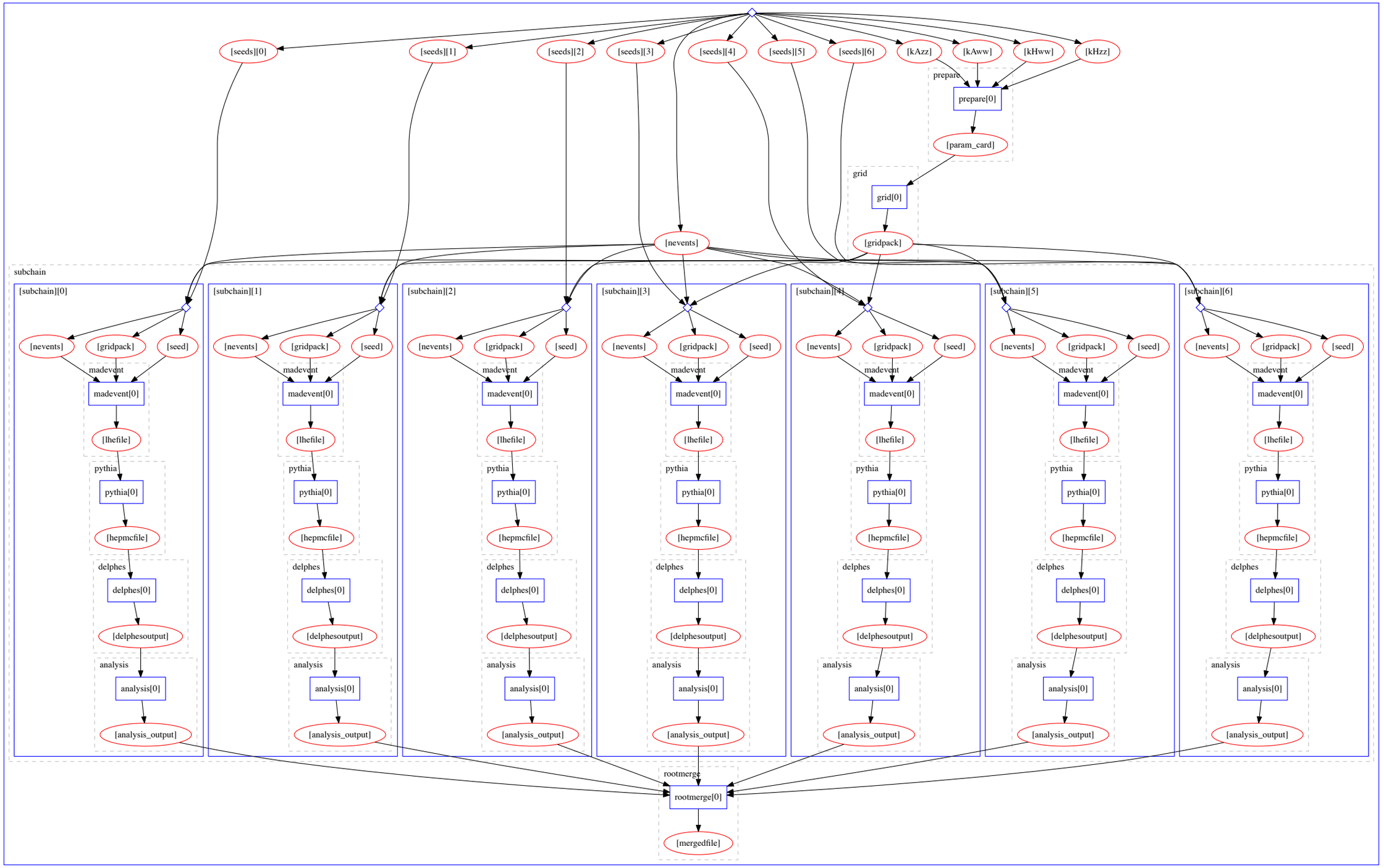
```
[$>  eval "$(curl https://raw.githubusercontent.com/diana-hep/yadage/master/yadagedocker.sh)"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   182  100   182    0     0   1831      0 --:--:-- --:--:-- --:--:--  1838
[$>  yadage-run -t from-github/phenochain workdir madgraph_delphes.yml -p nevents=100
[$>  ...
[$>  ls workdir/delphes
_packtivity/  output.lhco  output.root
$>
```

172-27-219-223 — -zsh — 99×8

```
[$>  pip install yadage
[$>  ...
[$>  yadage-run -t from-github/phenochain workdir madgraph_delphes.yml -p nevents=100
[$>  ...
[$>  ls workdir/delphes
_packtivity/  output.lhco  output.root
$>
```

Example:

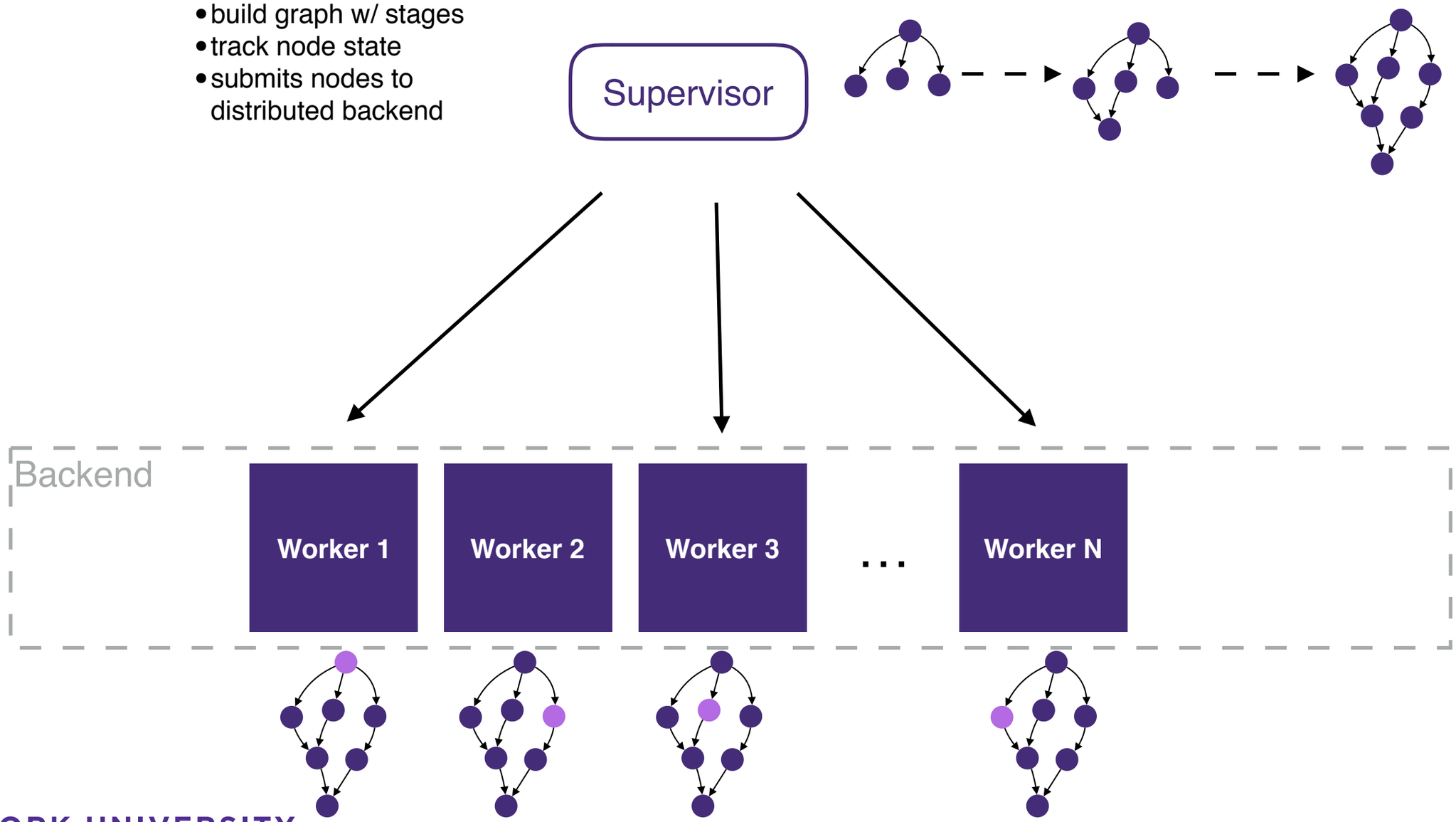
typical analysis using HEP tools (Madgraph, Pythia, Delphes, Custom Analysis Code.



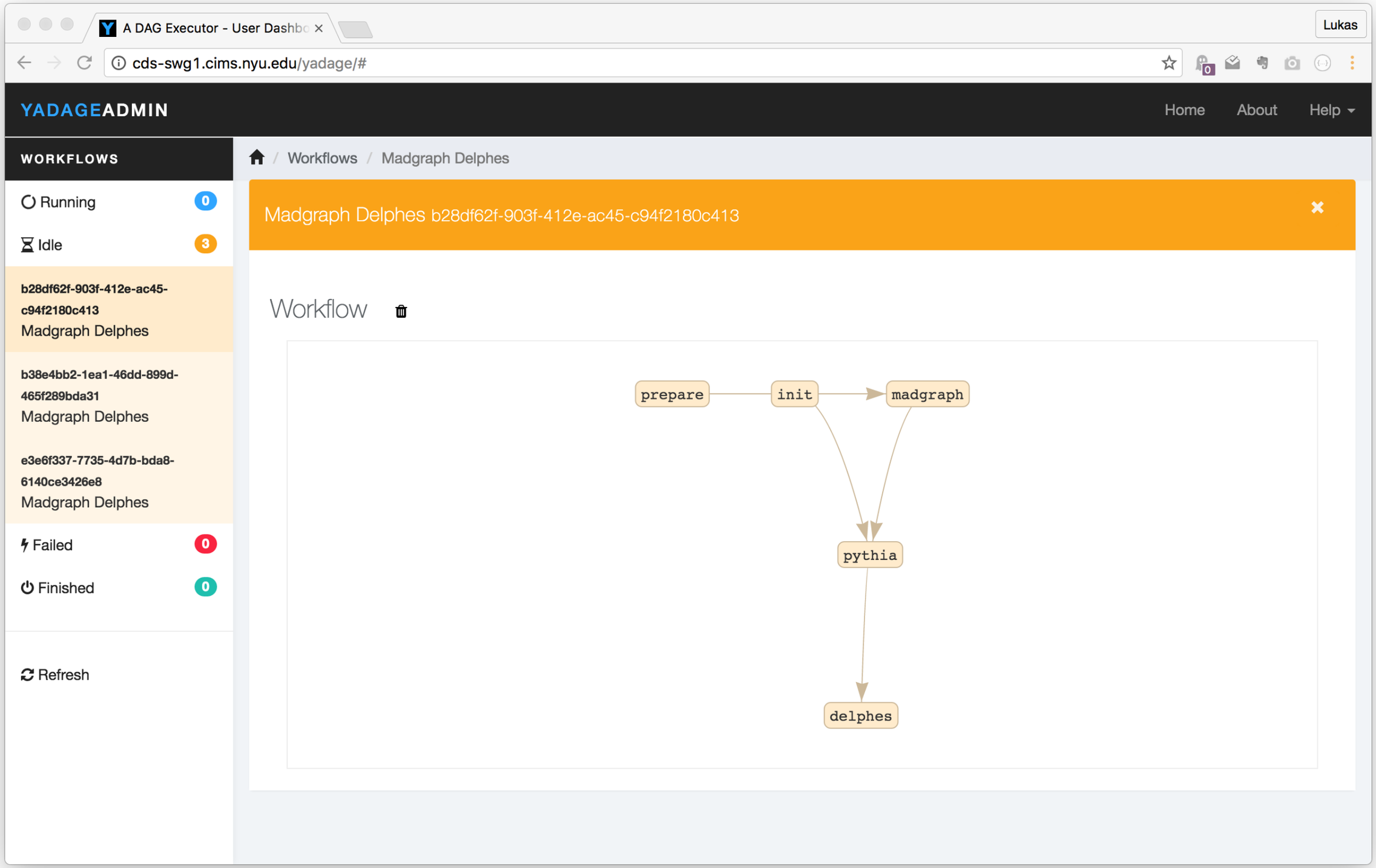
Distributing Workflows across multiple Hosts

easily distributable through packtivity python bindings via Celery

- build graph w/ stages
- track node state
- submits nodes to distributed backend



Web UI in development



Integration with CERN Analysis Preservation Portal (CAP)

JSON schemas integrated with CAP.

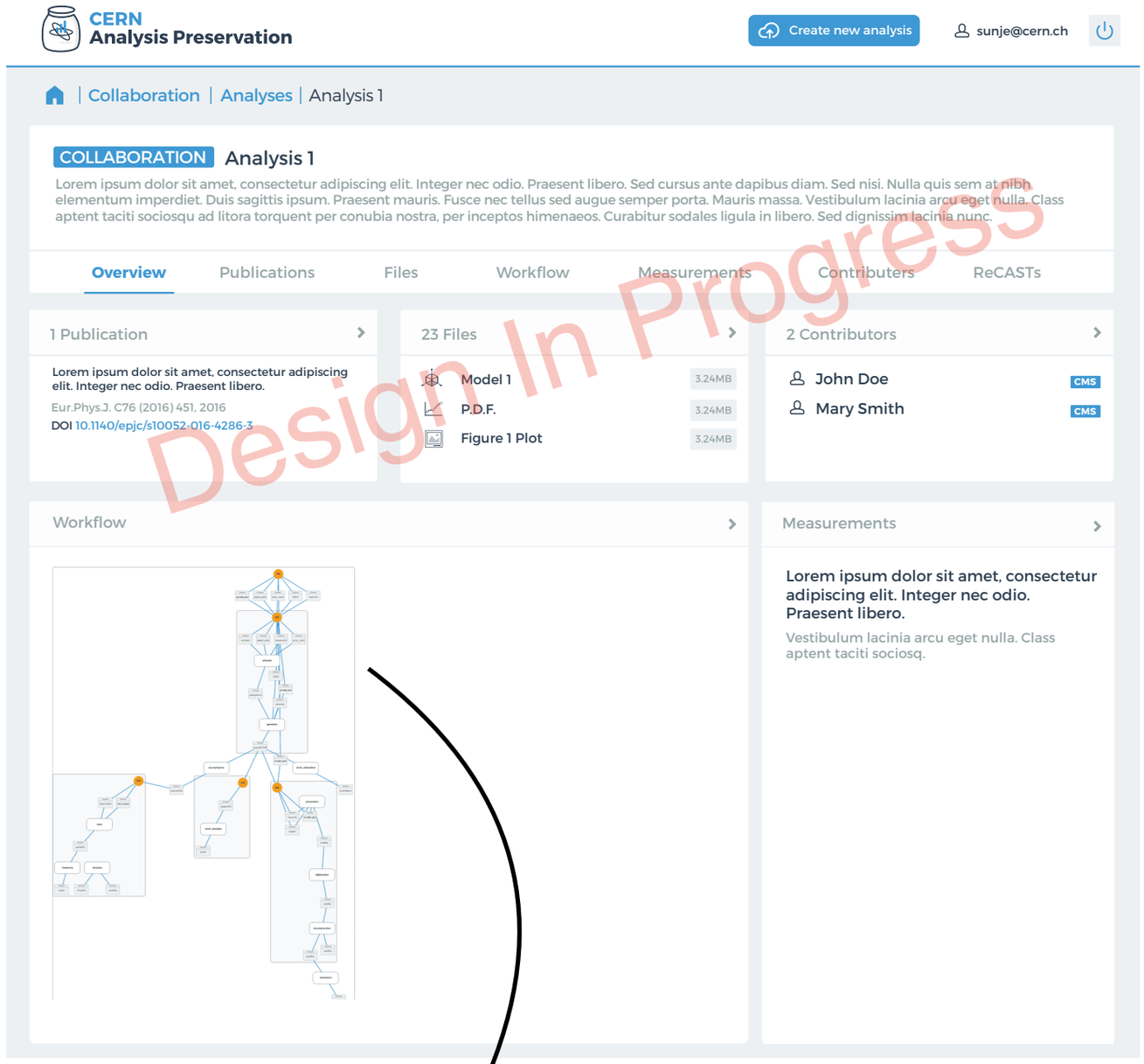
Analyses may store workflow(s) for relevant use cases

- workflows for reinterpretation
- workflows for auxiliaryy measurements
- …

Invenio-based CAP provides REST API for records

→ serve workflow spec via HTTP as JSON

CERN
Analysis Preservation




```
yadage-run -t from-cap/<recordID> <workdir> <workflow name> <parameters>
```

Integration with RECAST prototype

Reinterpretations are specific kind of workflows:

→ original data

→ new model $\quad f_{\text{analysis}}(\text{data}|\ \cdot\ )$

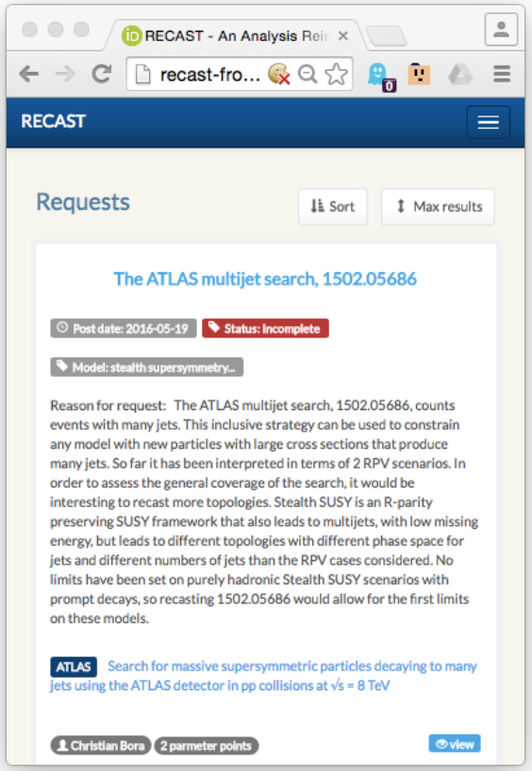
Prototype in development to enable LHC experiments to review requests for reinterpretation from larger HEP community, and if approved honor them.

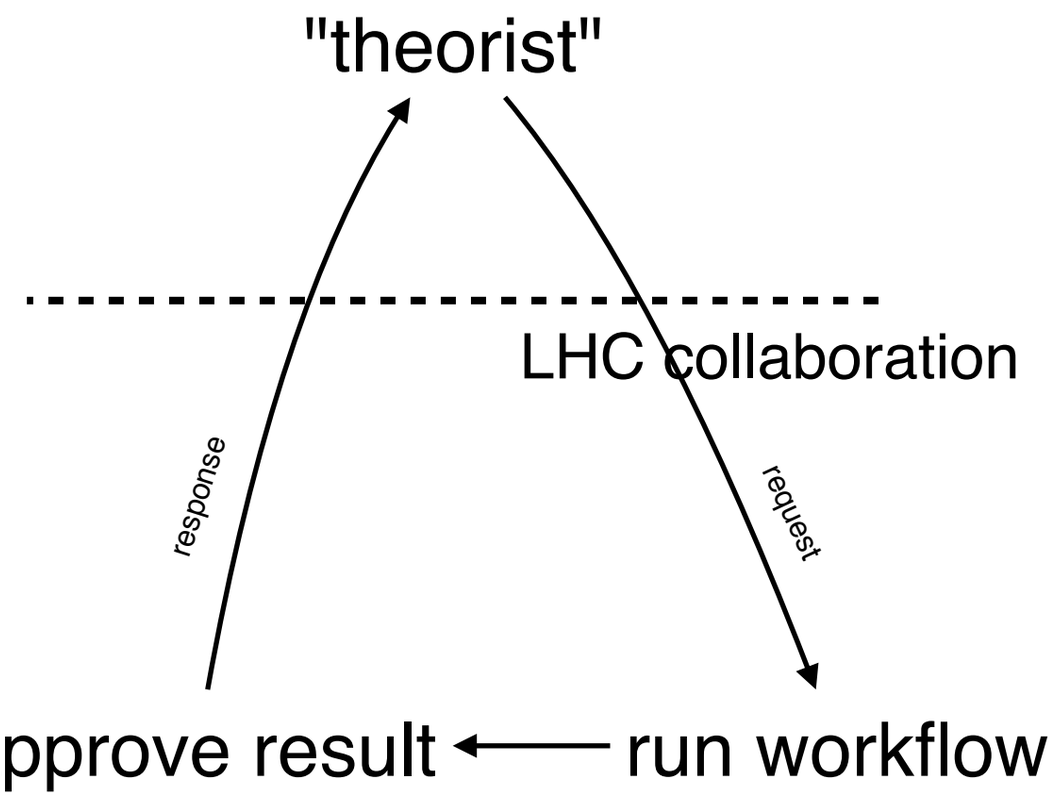
To be efficient / feasible re-usable analysis preservation is crucial

→ workflows where model is parametrized

→ possible interaction with CAP to retrieve workflows

Internal testing (deployed on CERN OpenStack resources) look promising





Yadage/Packtivity tools have been used in ATLAS Reinterpretation Campaigns

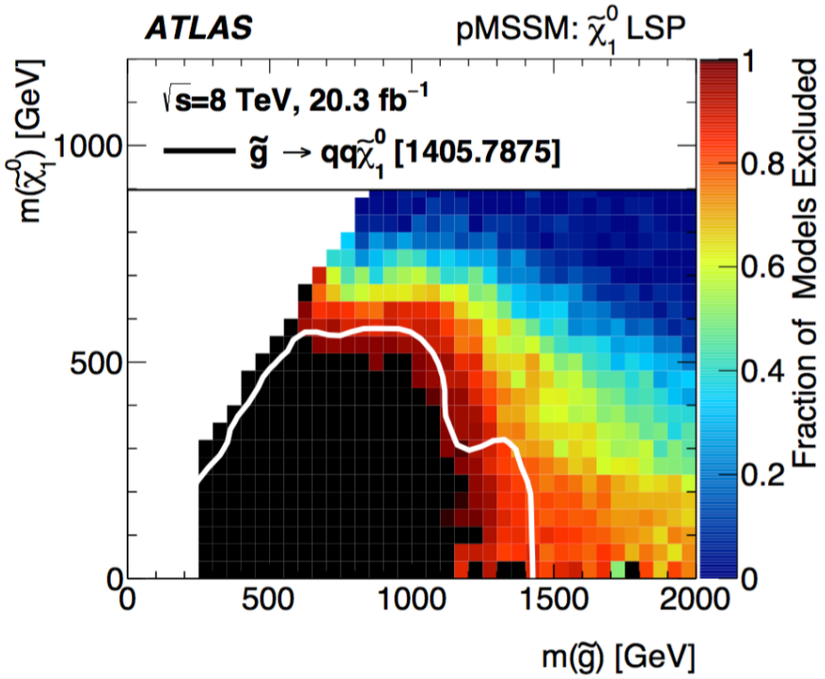
PUBLISHED FOR SISSA BY SPRINGER

RECEIVED: *August 27, 2015*
ACCEPTED: *September 23, 2015*
PUBLISHED: *October 21, 2015*

arXiv:1508.06608

19-D(!) pMSSM reinterpretation

Summary of the ATLAS experiment's sensitivity to supersymmetry after LHC Run 1 — interpreted in the phenomenological MSSM



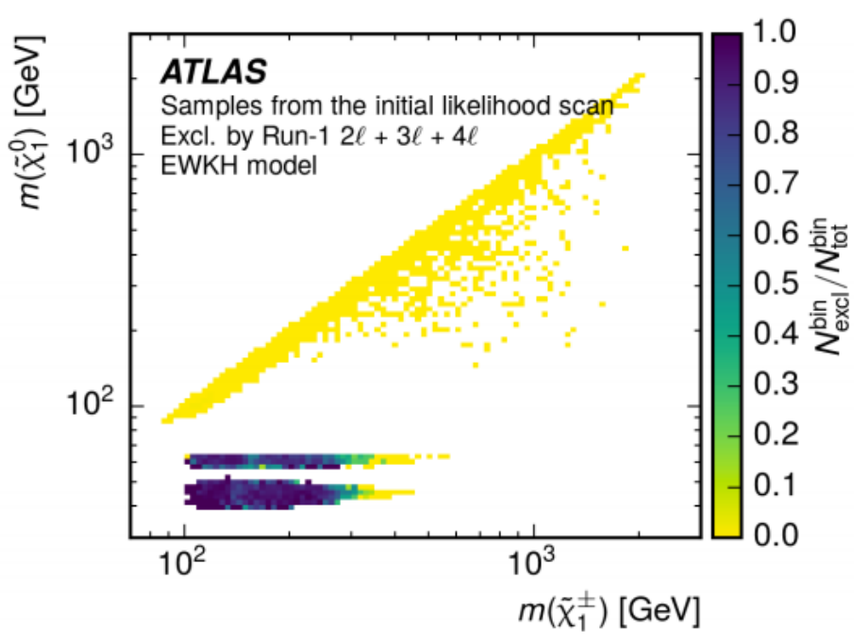
PUBLISHED FOR SISSA BY SPRINGER

RECEIVED: *August 3, 2016*
ACCEPTED: *September 22, 2016*
PUBLISHED: *September 30, 2016*

5-D scan of EWKH sector with help from STA

arXiv:1608.00872

Dark matter interpretations of ATLAS searches for the electroweak production of supersymmetric particles in $\sqrt{s} = 8$ TeV proton-proton collisions



A re-interpretation of $\sqrt{s} = 8$ TeV ATLAS results on electroweak supersymmetry production to explore general gauge mediated models

3-D recast for General Gauge Mediated SUSY Models

ATLAS-CONF-2016-033

The ATLAS Collaboration

Conclusions:

- Preserve analyses for Re-execution/Re-use, not just post-facto documentation.
- Source Code not enough, need full environment & notion on how to run software
- Linux Containers / Docker are powerful tools for preservation
- Separate declarative workflow description from concrete backend, aim for long-term preservation
- Dynamic DAGs are important for general solution
- Already producing new results with preserved analyses today with Reinterpretations.

Backup

Templated Scripts, various interpreters…

```yaml
process:
  process_type: 'interpolated-script-cmd'
  script: |
    #!/bin/bash
    echo "Hello"
    source ~/.bashrc
    setupATLAS
    source ./rcSetup.sh
    /recast_auth/getmyproxy.sh
    lsetup fax dq2
    python MultibjetsAnalysis/scripts/Run.py --dataSource 1 --doSyst 1 --doNTUPSyst 1 --doNTUP 0 --doxAOD 0 --doHistFitter 1
    mv {submitdir}/data-output_histfitter/*.root {outputprefix}.{did}.root
publisher:
  publisher_type: 'fromglob-pub'
  globexpression: '*.root'
  outputkey: histfitterfile
environment:
  environment_type: 'docker-encapsulated'
  image: lukasheinrich/multibsel_cvmfs
  resources:
    - CVMFS
    - GRIDProxy
```

```yaml
process:
  process_type: 'interpolated-script-cmd'
  interpreter: python
  script: |
    print 'Hello from Python'
    with open('{infile}') as input:
      with open('{outputfile}','w') as  output:
        output.write(input.read().upper())
publisher:
  publisher_type: 'frompar-pub'
  outputmap:
    outputfile: outputfile
environment:
  environment_type: 'docker-encapsulated'
  image: python
  imagetag: '2.7'
```

Nested Workflows

```yaml
stages:
  - name: prepare
    dependencies: ['init']
    scheduler:
      scheduler_type: singlestep-stage
      step: {$ref: 'steps.yml#/prepare'}
      parameters:
        kHzz: {stages: init, output: kHzz}
        kAzz: {stages: init, output: kAzz}
        kHww: {stages: init, output: kHww}
        kAww: {stages: init, output: kAww}
        param_card: '{workdir}/param.dat'

  - name: grid
    dependencies: ['prepare']
    scheduler:
      scheduler_type: singlestep-stage
      step: {$ref: 'steps.yml#/grid'}
      parameters:
        param_card: {stages: prepare, output: param_card}
        gridpack: '{workdir}/grid.tar.gz'

  - name: subchain
    dependencies: ['grid']
    scheduler:
      scheduler_type: multistep-stage
      workflow: {$ref: 'subchain.yml'}
      parameters:
        gridpack: {stages: grid, output: gridpack, unwrap: true}
        nevents: {stages: init, output: nevents, unwrap: true}
        seed: {stages: init, output: seeds, flatten: true}
      scatter:
        method: zip
        parameters: ['seed']

  - name: rootmerge
    dependencies: ['subchain']
    scheduler:
      scheduler_type: singlestep-stage
      step: {$ref: 'steps.yml#/rootmerge'}
      parameters:
        mergedfile: '{workdir}/anamerged.root'
        inputfiles: {stages: 'subchain.[*].analysis', output: analysis_output}
```

```yaml
stages:
  - name: madevent
    dependencies: ['init']
    scheduler:
      scheduler_type: singlestep-stage
      step: {$ref: 'steps.yml#/madgraph'}
      parameters:
        gridpack: {stages: init, output: gridpack}
        nevents: {stages: init, output: nevents}
        seed: {stages: init, output: seed}
        lhefile: '{workdir}/lhefile.lhe'

  - name: pythia
    dependencies: ['madevent']
    scheduler:
      scheduler_type: singlestep-stage
      step: {$ref: 'steps.yml#/pythia'}
      parameters:
        settings_file: /analysis/mainPythiaMLM.cmnd
        hepmcfile: '{workdir}/outputfile.hepmc'
        lhefile: {stages: madevent, output: lhefile}

  - name: delphes
    dependencies: ['pythia']
    scheduler:
      scheduler_type: singlestep-stage
      step: {$ref: 'steps.yml#/delphes'}
      parameters:
        detector_card: /analysis/template_cards/modified_delphes_card_ATLAS.tcl
        outputfile: '{workdir}/outputfile.root'
        inputfile: {stages: pythia, output: hepmcfile}

  - name: analysis
    dependencies: ['delphes']
    scheduler:
      scheduler_type: singlestep-stage
      step: {$ref: 'steps.yml#/analysis'}
      parameters:
        fromdelphes: {stages: delphes, output: delphesoutput}
        analysis_output: '{workdir}/anaout.root'
```