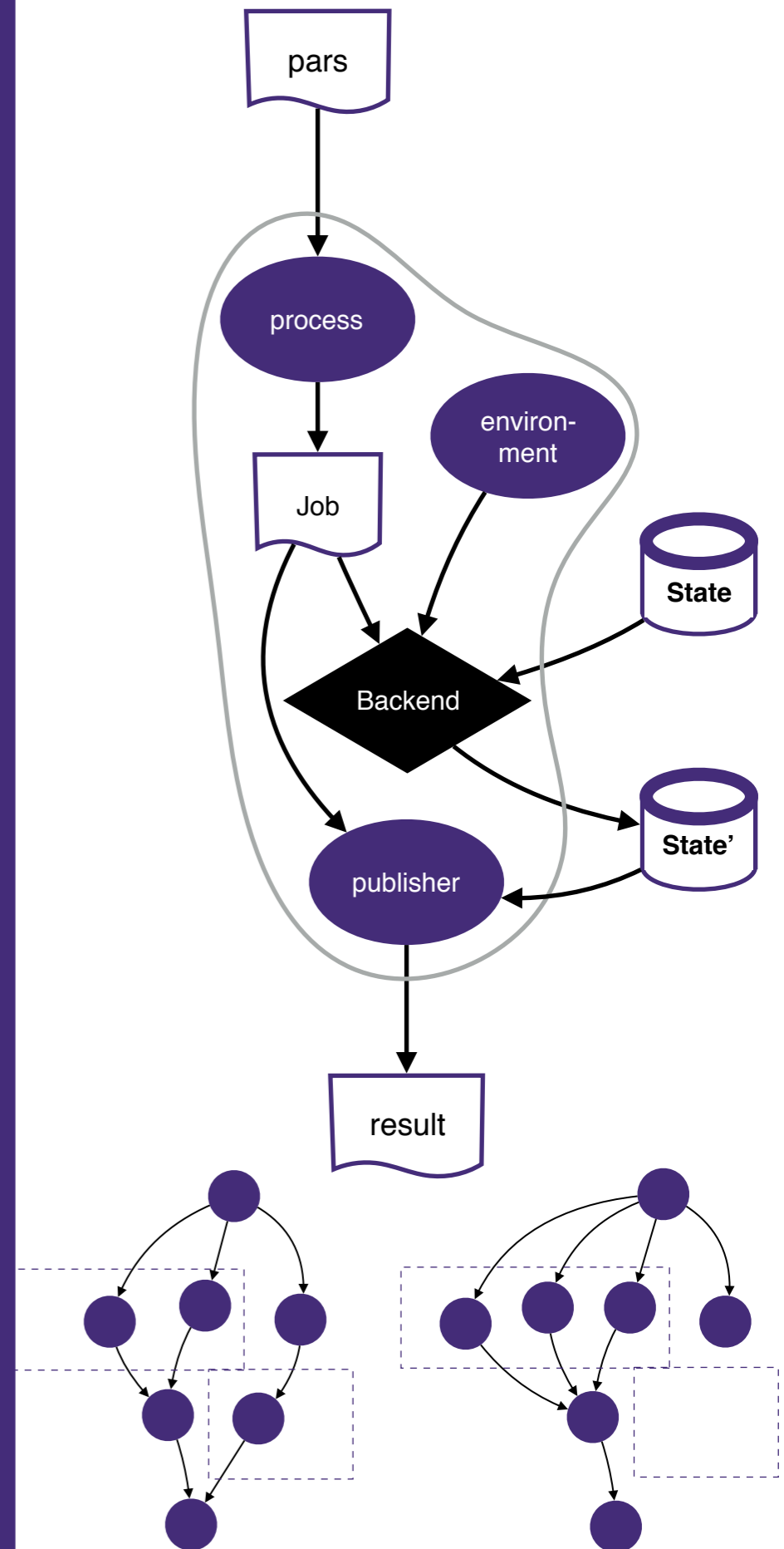


Analysis Preservation and Reinterpretation

Lukas Heinrich

CHEP 2016, San Francisco



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

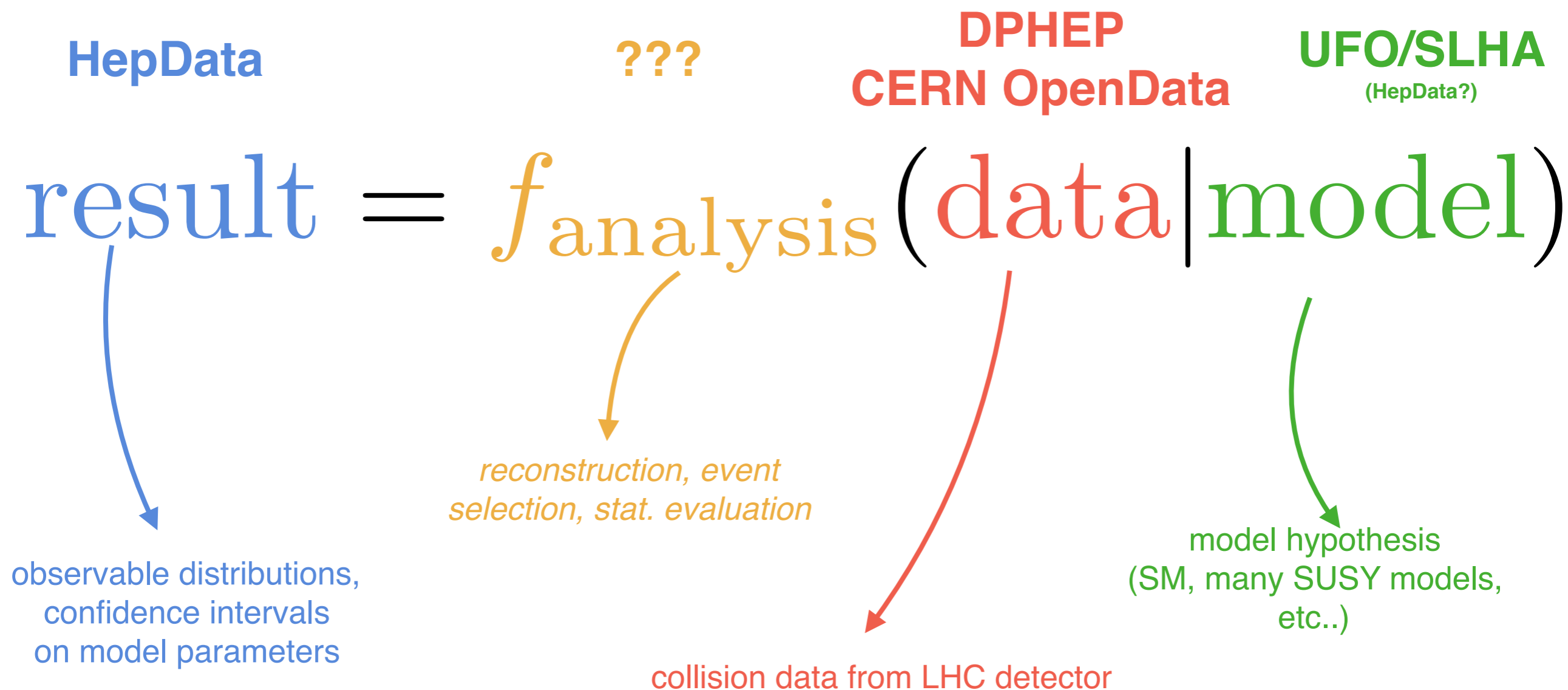
model hypothesis
(SM, many SUSY models,
etc..)



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.

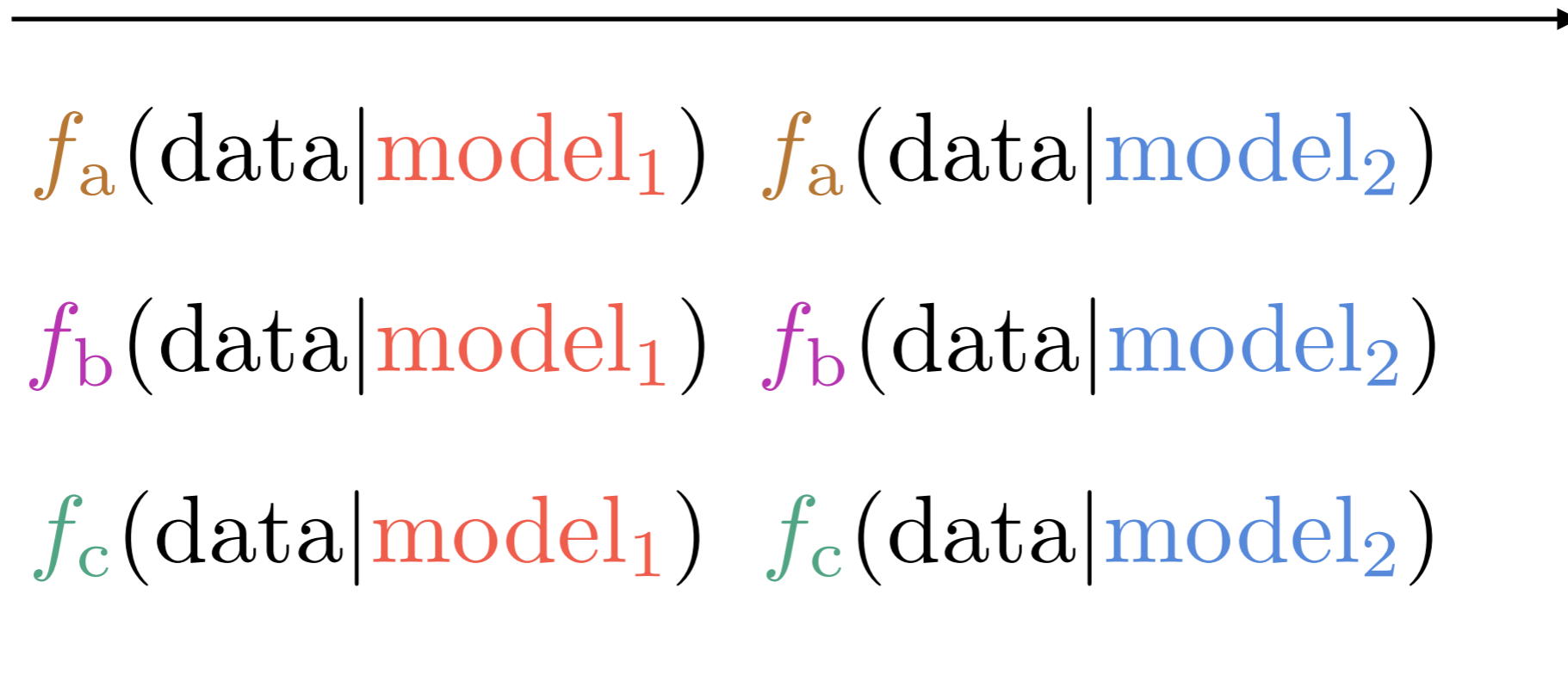


Analysis should be preserve the analysis in a **parametrized form**, not only its concrete application, separately from its application on given dataset.

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

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)



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

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 $f_{\text{analysis}}(\cdot)$?

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)

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

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>”

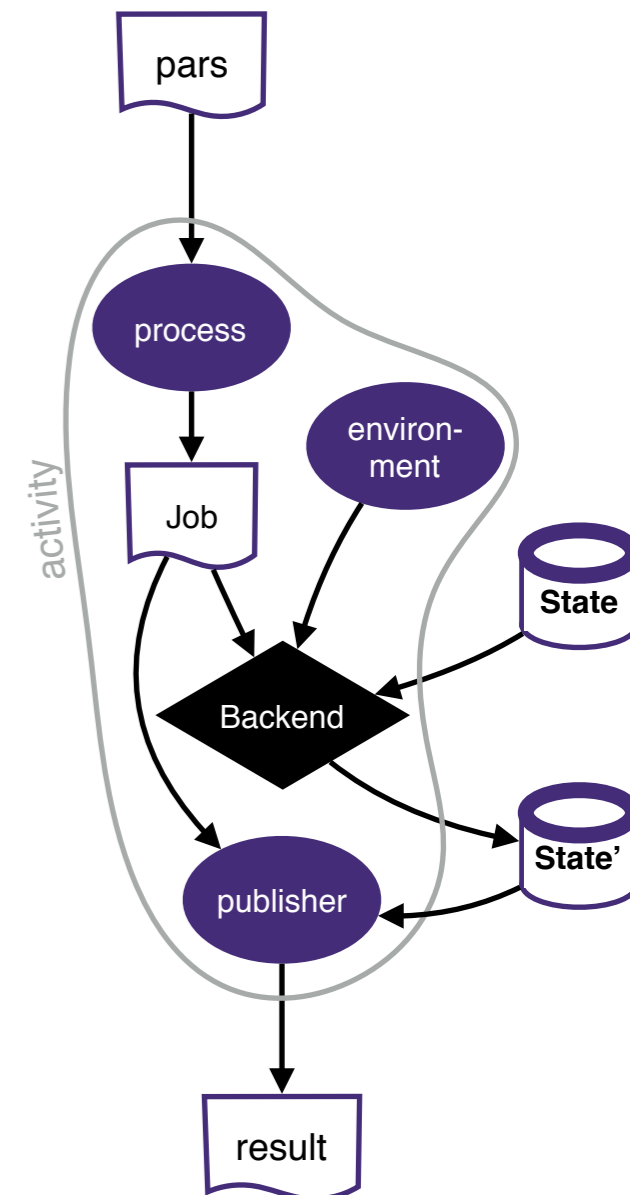
concrete: “./DelphesHepMC /input/file/path.hepmc /output/file.root”

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 $f_{\text{analysis}}(\cdot)$?

1. Problem: Preserve Individual Processing Steps

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

Data Format: JSON

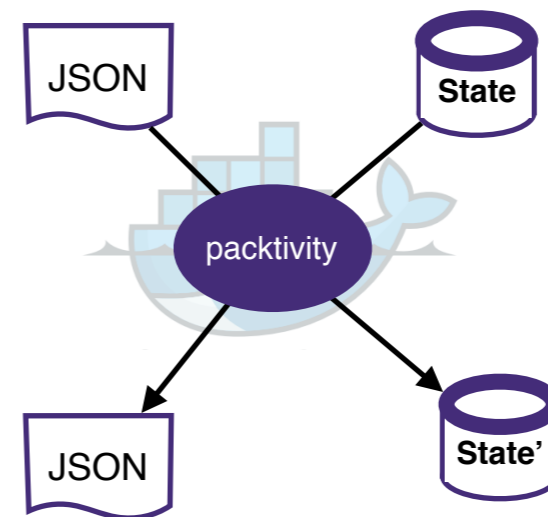
- as interchange format for parameters and result data
- as declarative description format for *process/env/publisher*
 - incl. JSON schemas for validation

Essentially, a self-consistent “packaged activity” – a “packtivity”

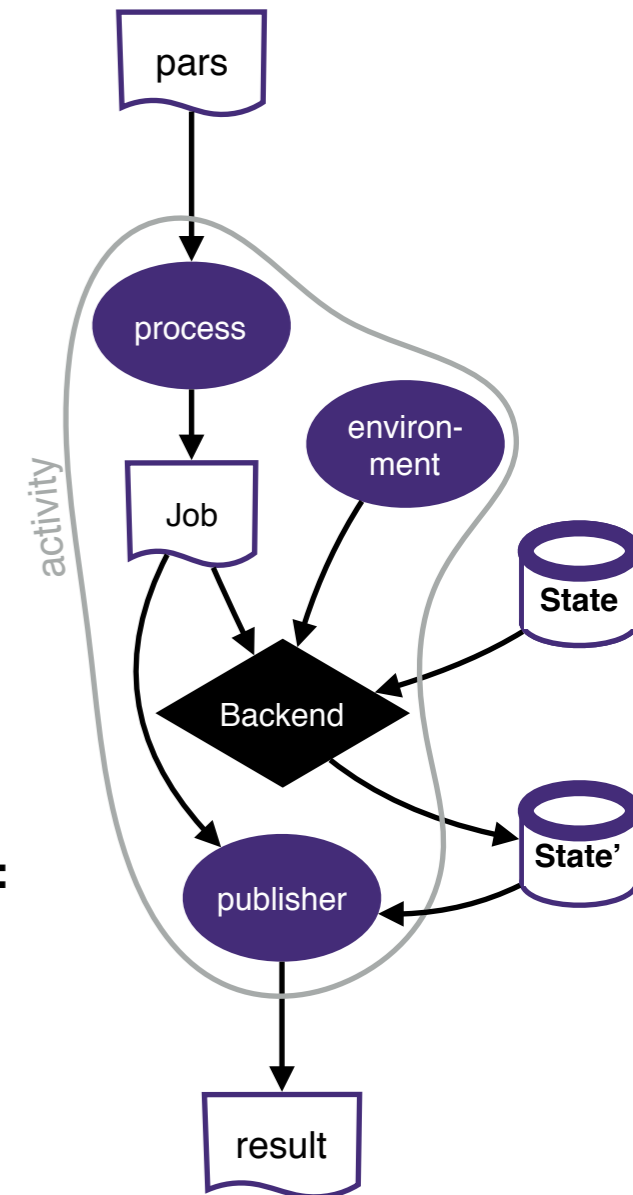
- JSON API
- archivable, declarative description as JSON
- dependencies captured in environment
 - e.g. Docker Image

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

=



=



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

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
```

python package: “packtivity”

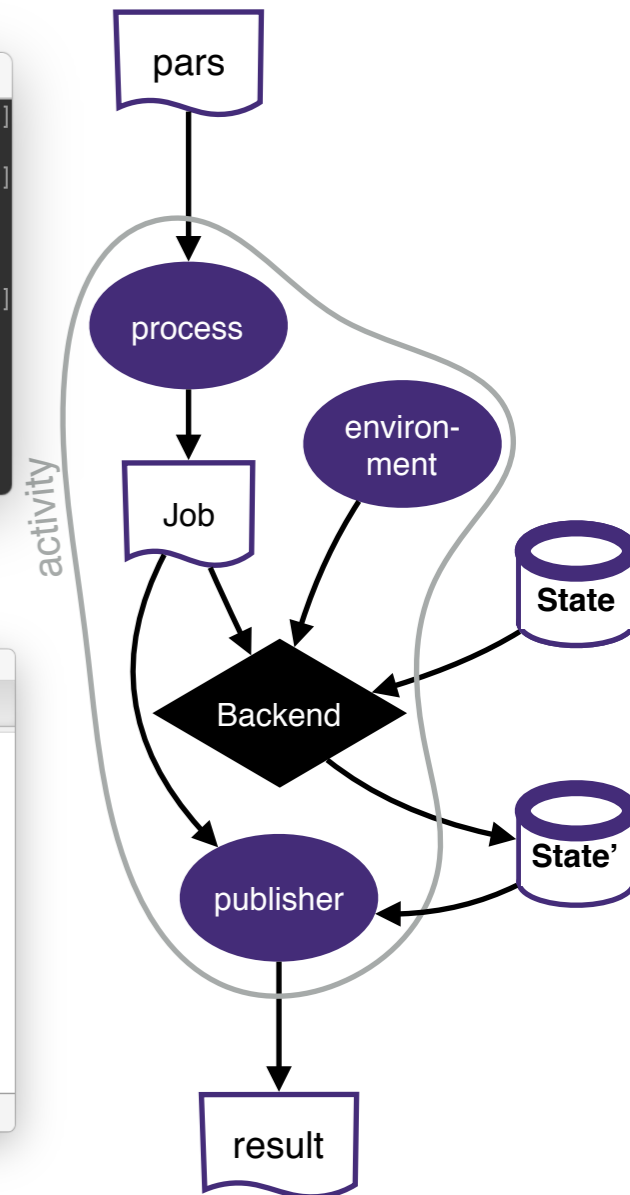
- executes packtivities according to JSON spec for given parameters
- cli tool and python bindings
- multi-host / remote execution ready via e.g. celery or remote Docker

CLI tool

```
172-27-219-223 — -zsh — 64x12
[$> ls
delphes.yml input.hepmc pars.yml
[$> 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))
```



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

2. Problem: Preserve Parametrized Workflow

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

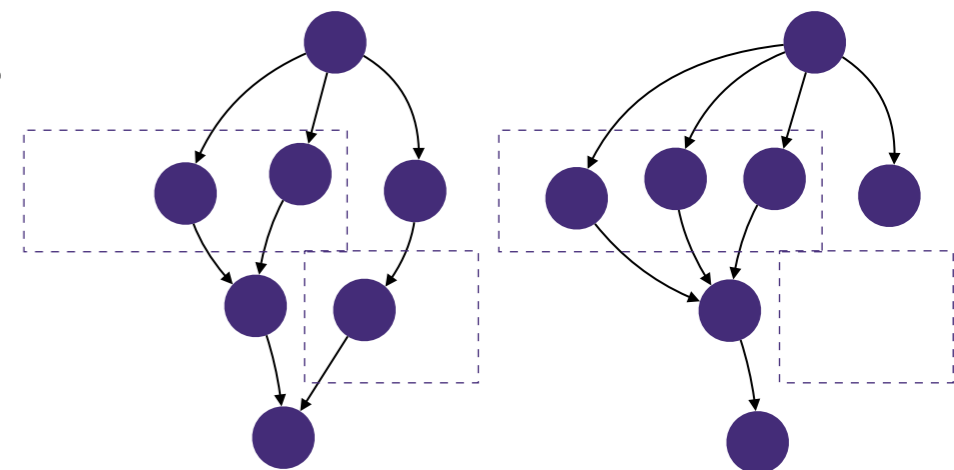
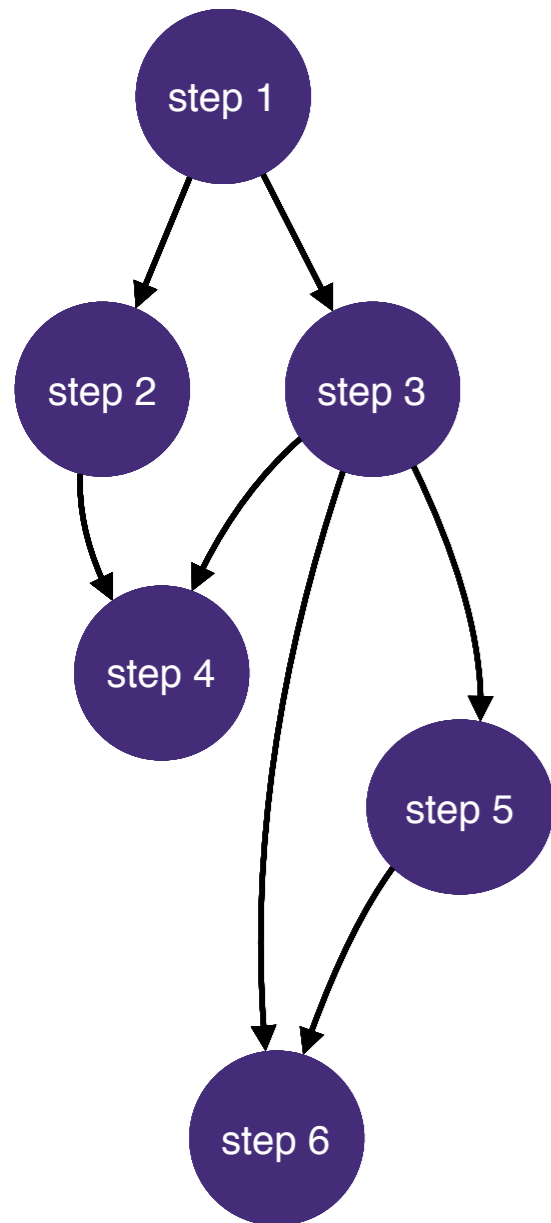
- **nodes**: individual steps
- **edges**: dependency relations

Two place where parametrization enter:

1. individual steps parametrized: covered by “packtivities”
graph topology may *depend on the parameters* of the analysis, as well

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

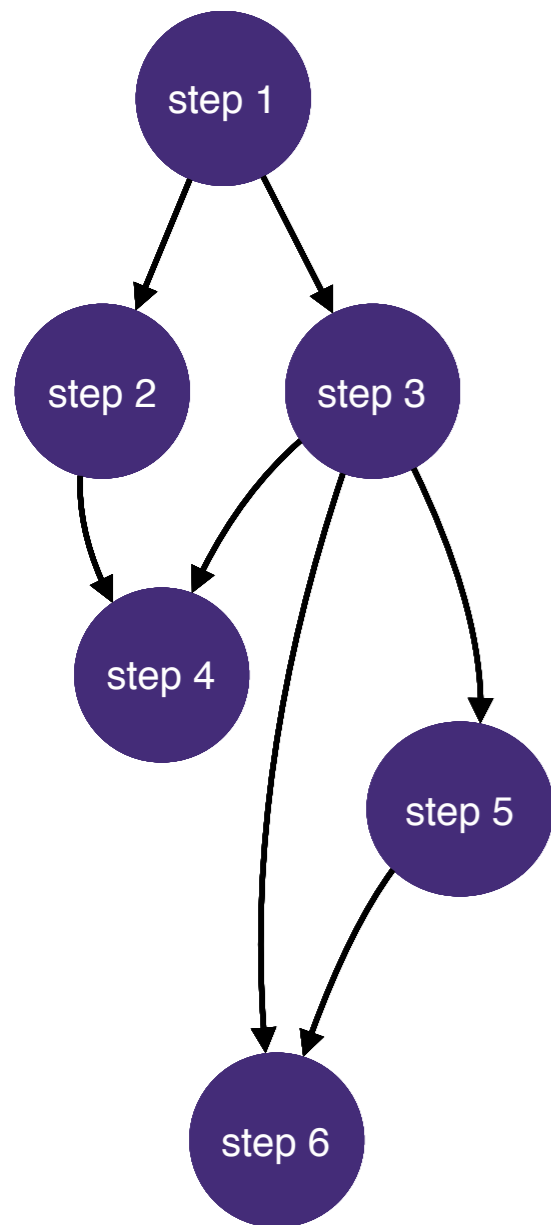


How to preserve $f_{\text{analysis}}(\cdot)$?

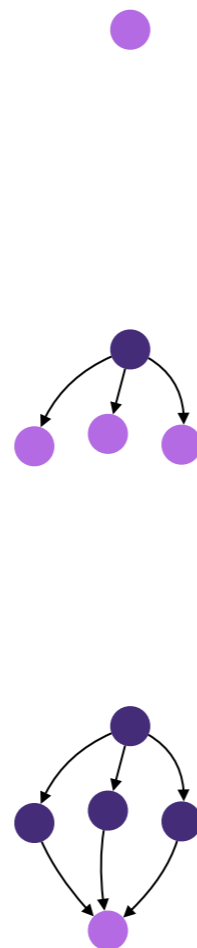
2. Problem: Preserve Parametrized Workflow

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**

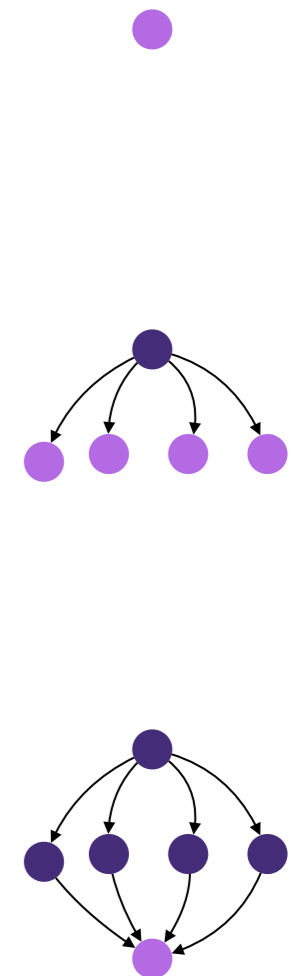


Par. Set 1

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)



Par. Set 2

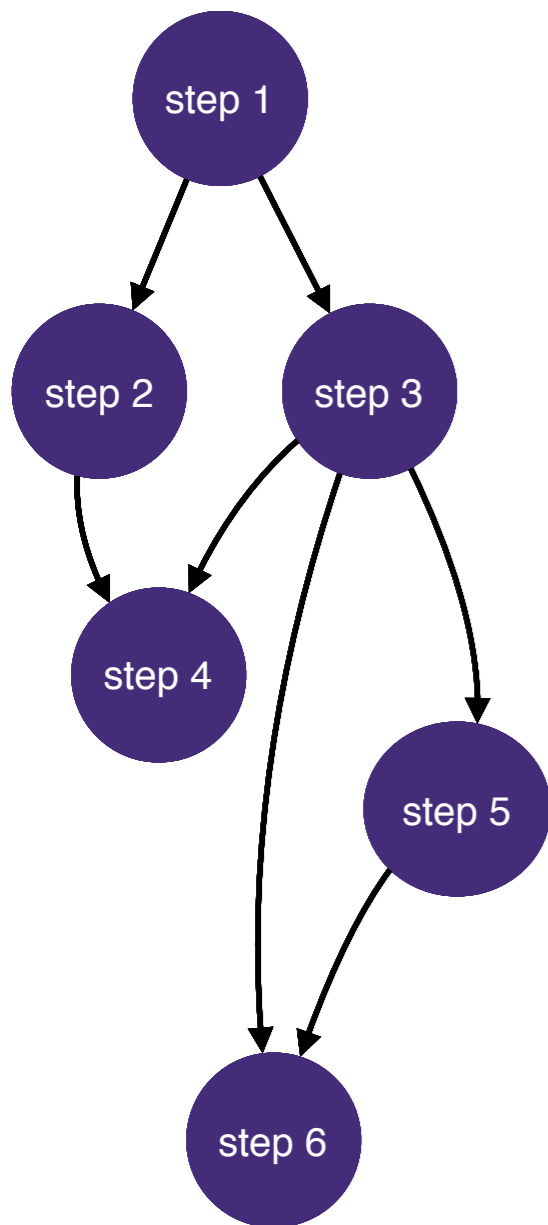


How to preserve $f_{\text{analysis}}(\cdot)$?

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



```
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 $f_{\text{analysis}}(\cdot)$?

2. Problem: Preserve Parametrized Workflow

First implementation: **Yadage**

python package.

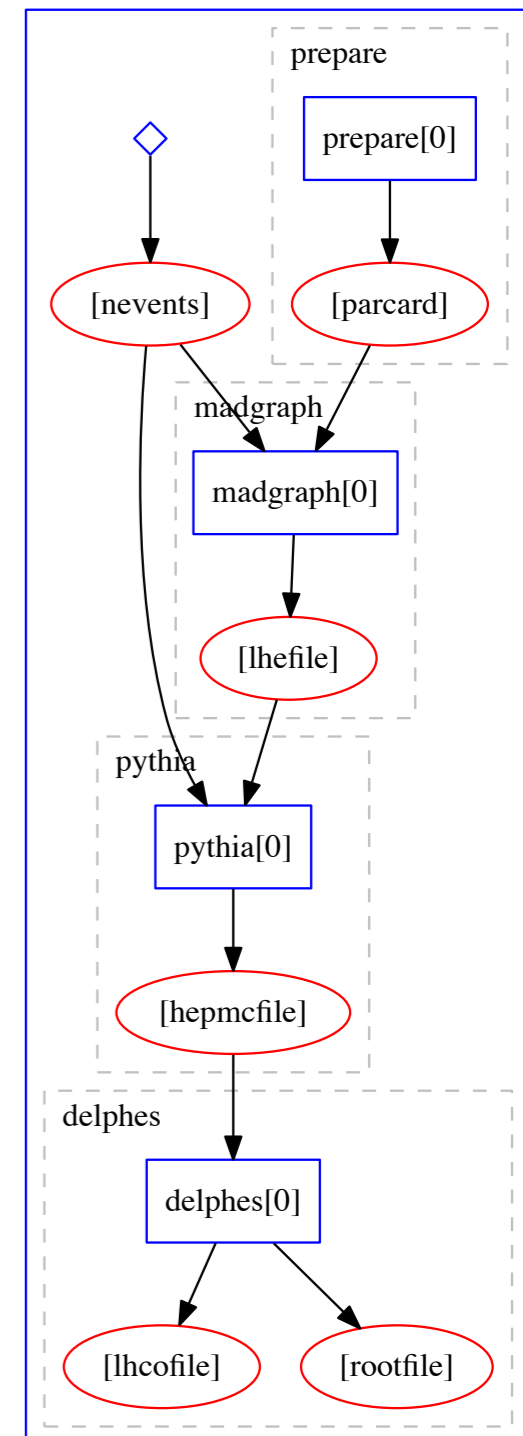
install from **PyPi** or ...

... run from from *within docker*

can run complex workflows with vastly different environments on hosts that know nothing about the domain software. just need docker

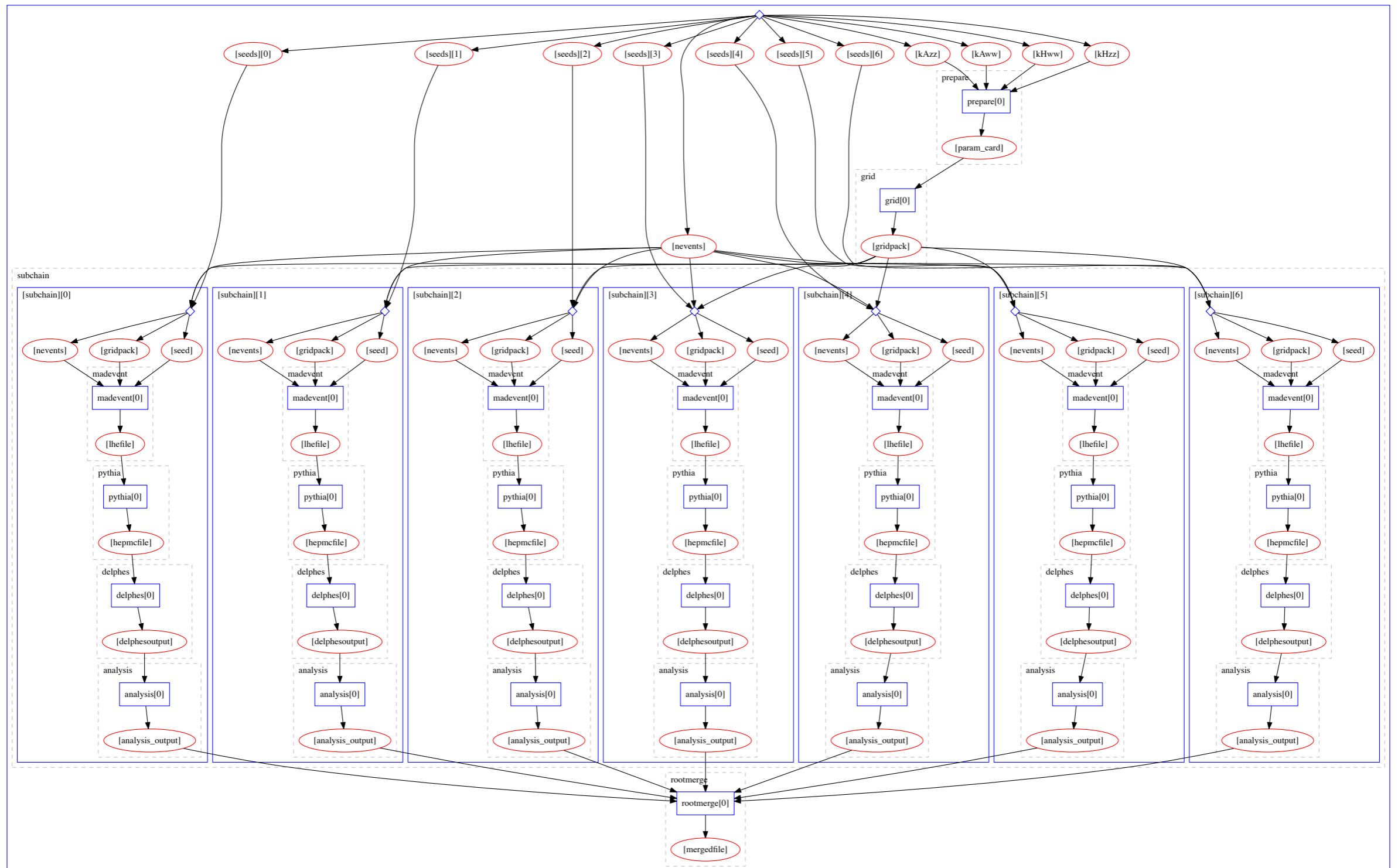
```
172-27-219-223 — -zsh — 99x9
[$> eval "$(curl https://raw.githubusercontent.com/diana-hep/yadage/master/yadagedocker.sh)"
% Total % Received % Xferd Average Speed Time Time Time Current
          Dload Upload Total Spent Left Speed
100 182 100 182 0 0 1831 0 --:--:-- --:--:-- --:--:-- 1838
[$> yadage-run -t from-github/phenochain workdir madgraph_delphes.yml -p nevents=100
[$> ...
[$> ls workdir/delphes
_packtivity/ output.lhco output.root
[$>
```

```
172-27-219-223 — -zsh — 99x8
[$> pip install yadage
[$> ...
[$> yadage-run -t from-github/phenochain workdir madgraph_delphes.yml -p nevents=100
[$> ...
[$> ls workdir/delphes
_packtivity/ output.lhco output.root
[$>
```



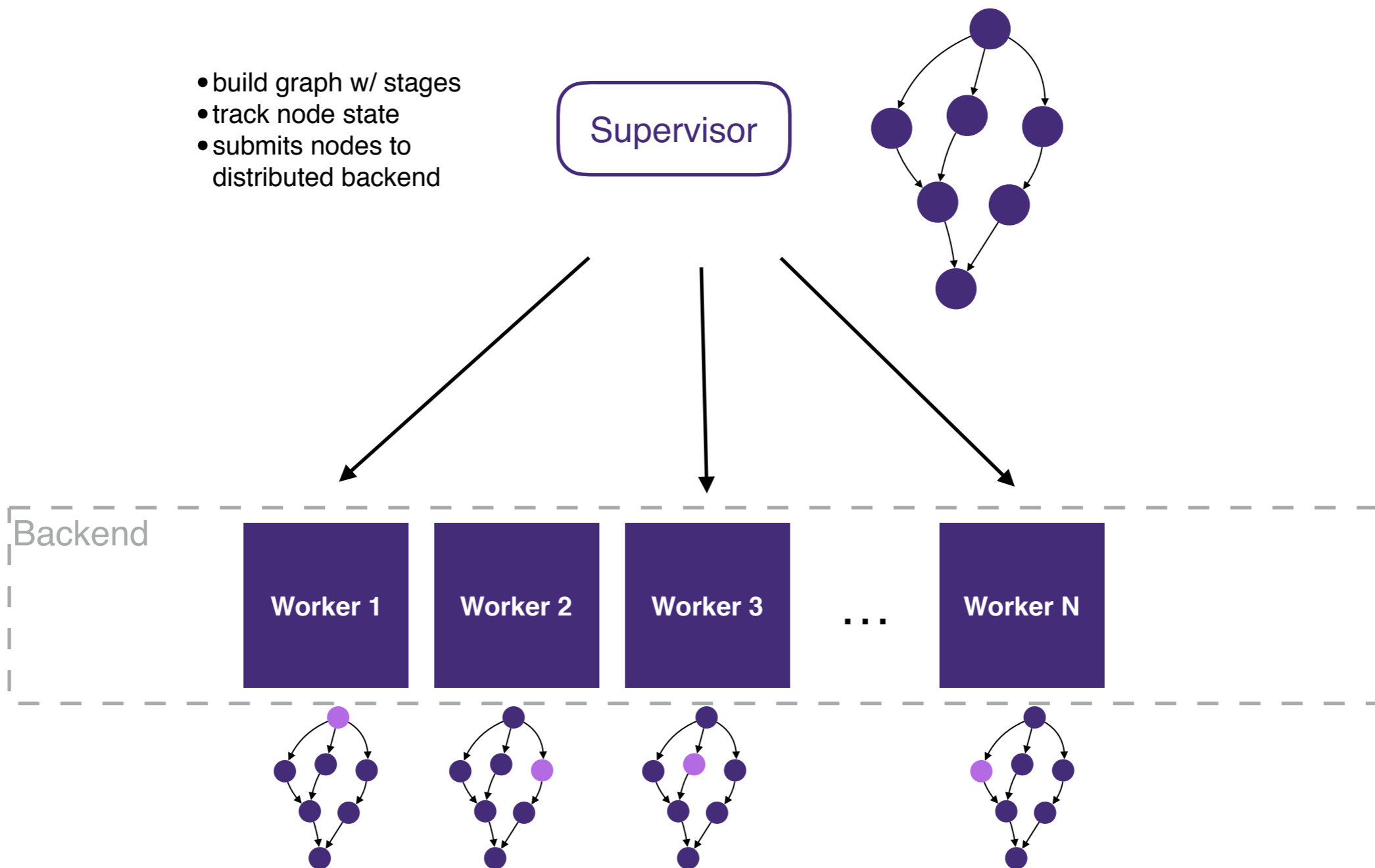
Example:

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



Distributing Workflows across multiple Hosts

easily distributable through packtivity python bindings via Celery



Web UI in development

The screenshot displays the YADAGEADMIN web interface. The browser tab is titled "A DAG Executor - User Dashbo x" and the address bar shows "cds-swg1.cims.nyu.edu/yadage/#". The user's name "Lukas" is visible in the top right corner. The main navigation bar includes "YADAGEADMIN" and links for "Home", "About", and "Help".

The left sidebar, titled "WORKFLOWS", lists workflow statuses: "Running" (0), "Idle" (3), "Failed" (0), and "Finished" (0). A "Refresh" button is located at the bottom of the sidebar. The selected workflow is "b28df62f-903f-412e-ac45-c94f2180c413 Madgraph Delphes".

The main content area shows the workflow diagram for "Madgraph Delphes b28df62f-903f-412e-ac45-c94f2180c413". The diagram is a Directed Acyclic Graph (DAG) with the following structure:

```
graph TD; prepare[prepare] --> init[init]; init --> madgraph[madgraph]; init --> pythia[pythia]; madgraph --> pythia; pythia --> delphes[delphes];
```

The workflow consists of five tasks: "prepare", "init", "madgraph", "pythia", and "delphes". The "prepare" task leads to "init". From "init", the flow branches to "madgraph" and "pythia". Both "madgraph" and "pythia" lead to the final "delphes" task.



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 auxiliary 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:

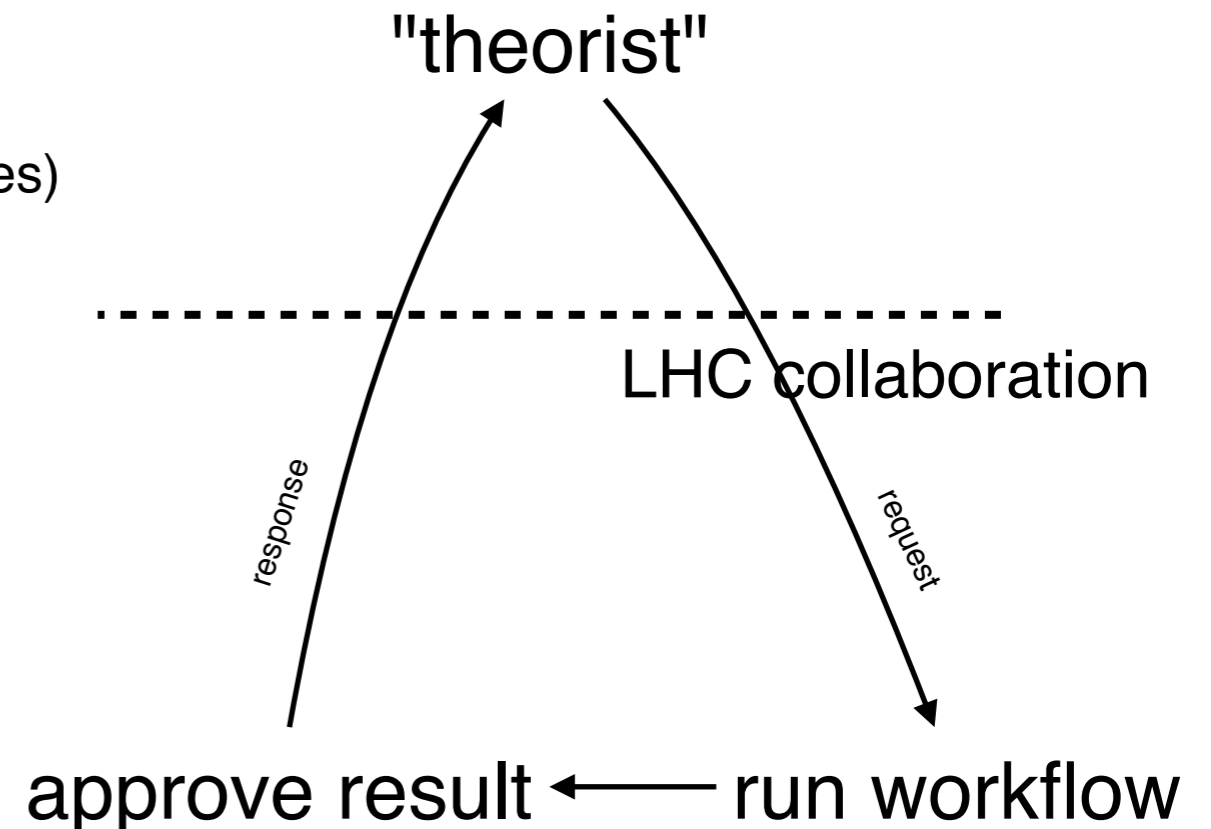
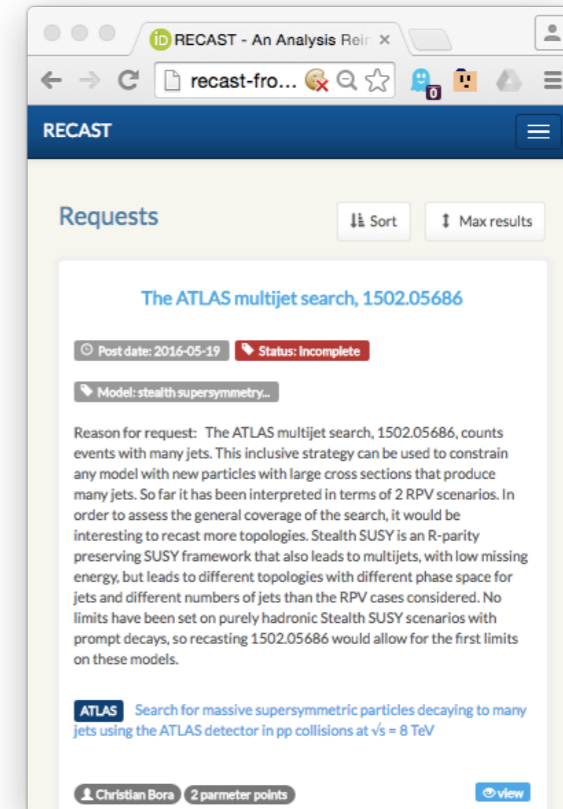
- original data
 - new model
- $$f_{\text{analysis}}(\text{data} | \cdot)$$

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

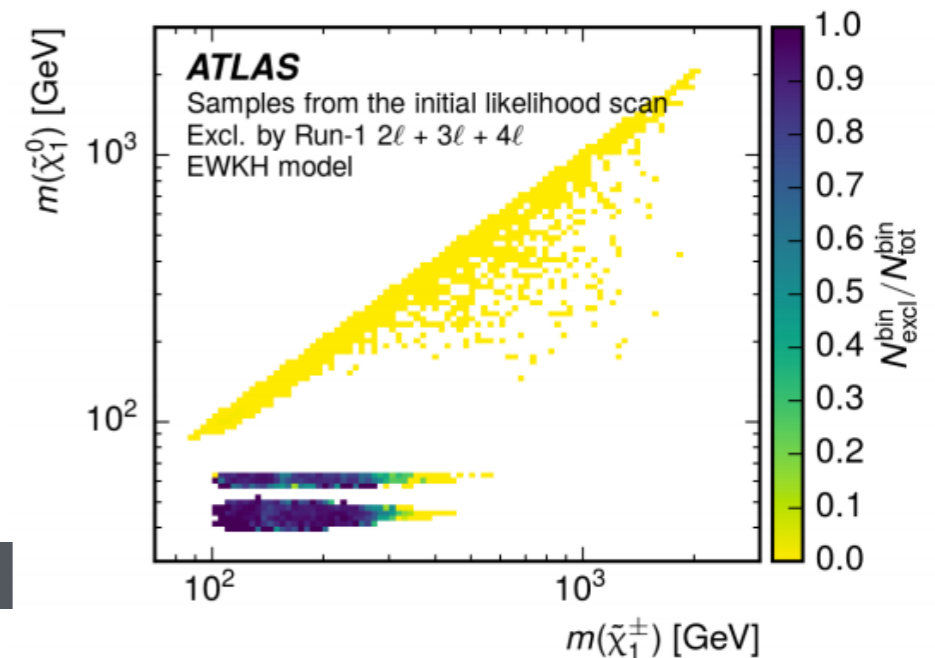
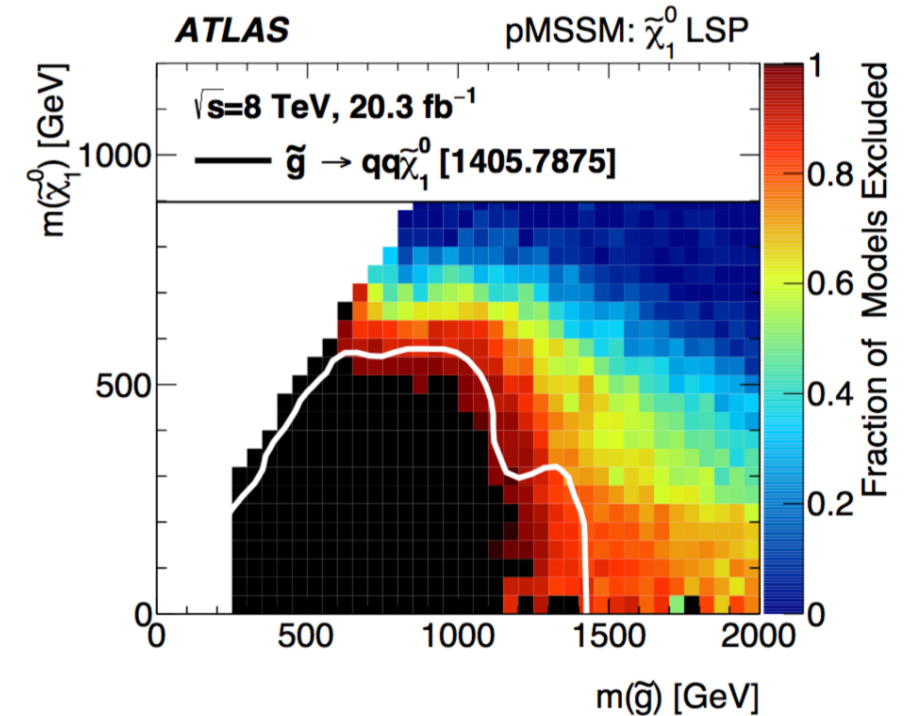
Dark matter interpretations of ATLAS searches for the electroweak production of supersymmetric particles in $\sqrt{s} = 8$ TeV proton-proton collisions

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

The ATLAS Collaboration

ATLAS-CONF-2016-033



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
- Dynamic DAGs are important for general solution
- Already producing new results with preserved analyses today with Reinterpretations.

