Containers in ATLAS

Lukas Heinrich

CERN IT-CM Group Meeting

NEW YORK UNIVERSITY

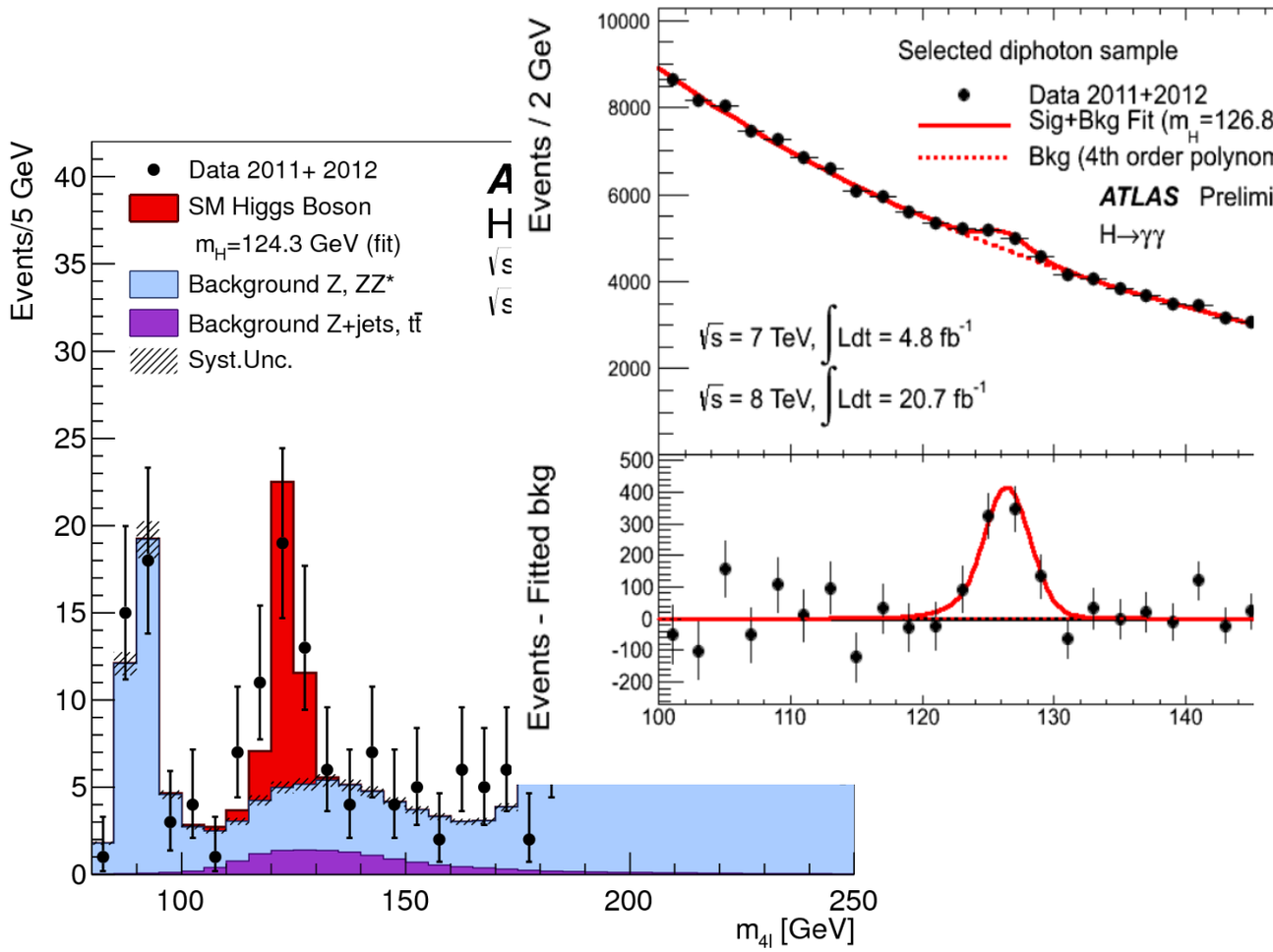
ATLAS Physics…

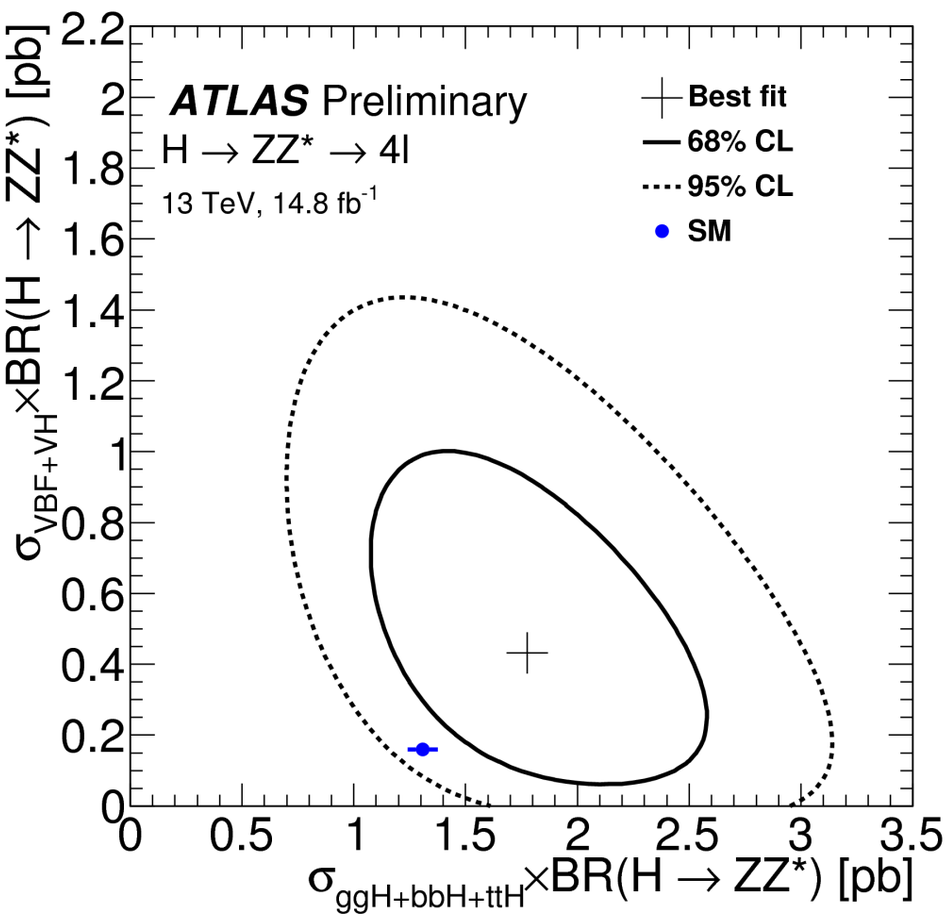
NEW YORK UNIVERSITY

Two high-level objectives for the ATLAS Physics program

1. Precision Measurement of known physics (Standard Model)

- We found the missing piece of the Standard Mode — the Higgs — now we want to measure its properties

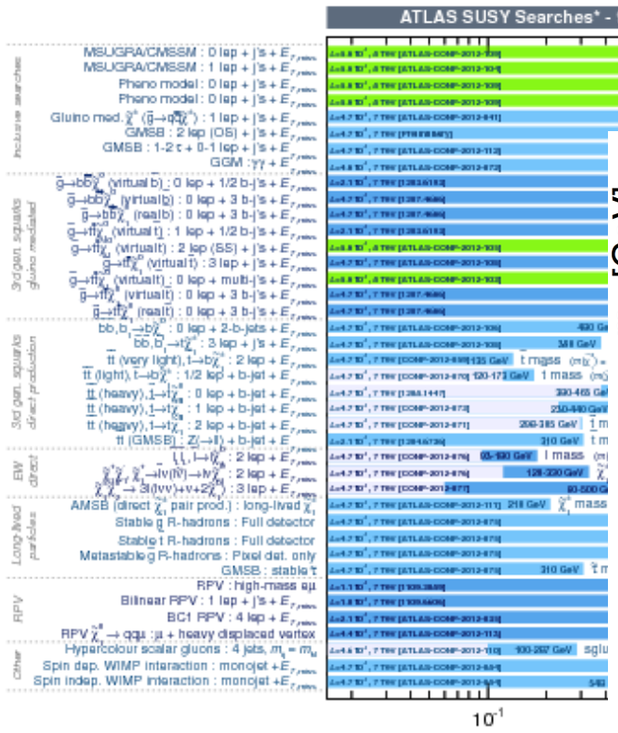


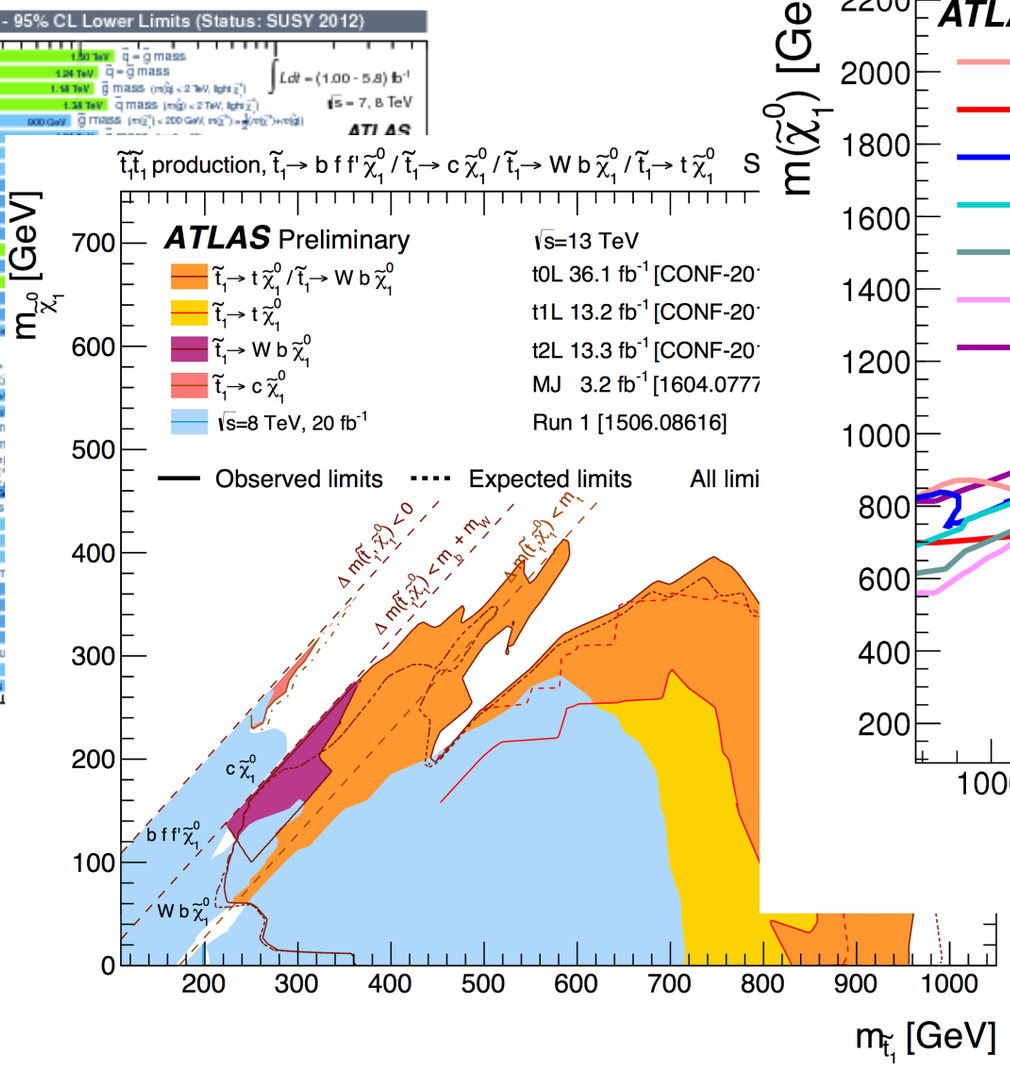


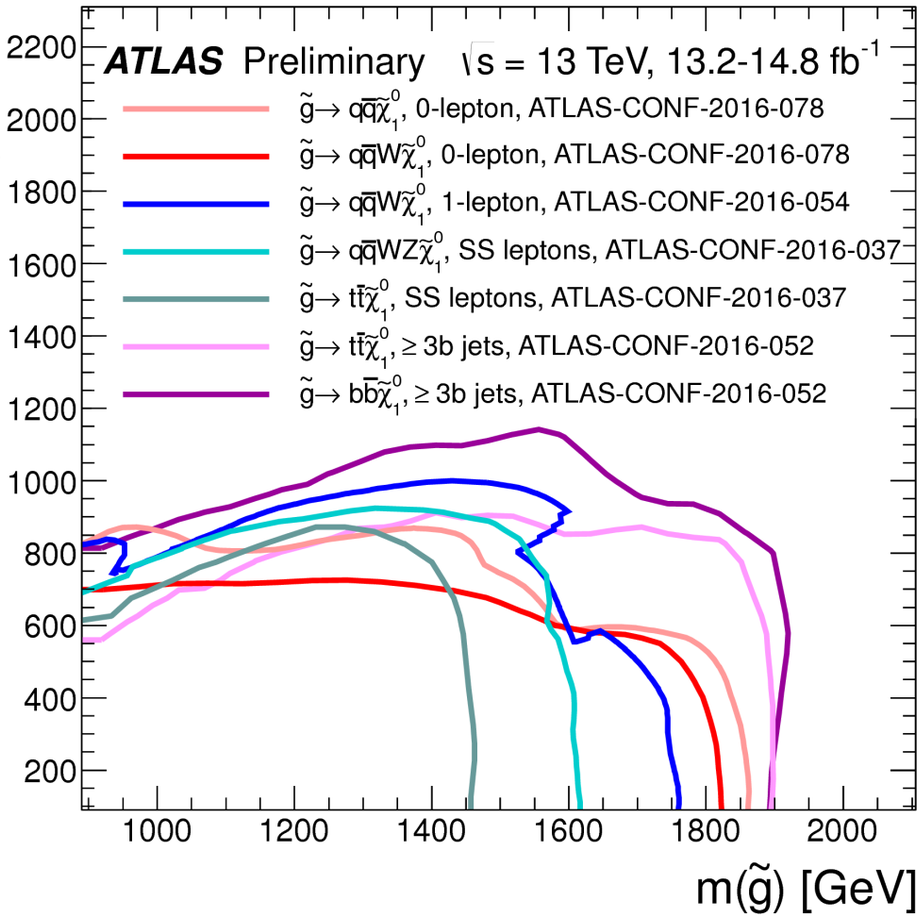
Two high-level objectives for the ATLAS Physics program

2. Searches for Physics Beyond the Standard Model (BSM)

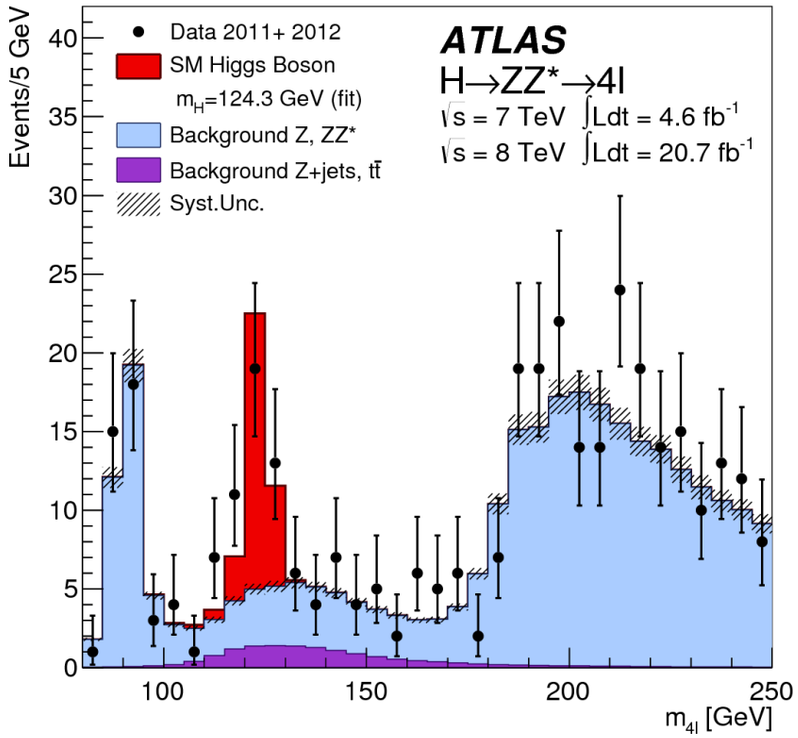
- We know, the Standard Model is not the final answer. Search for new phenomena
- Most intensely studied candidate **Supersymmetry**

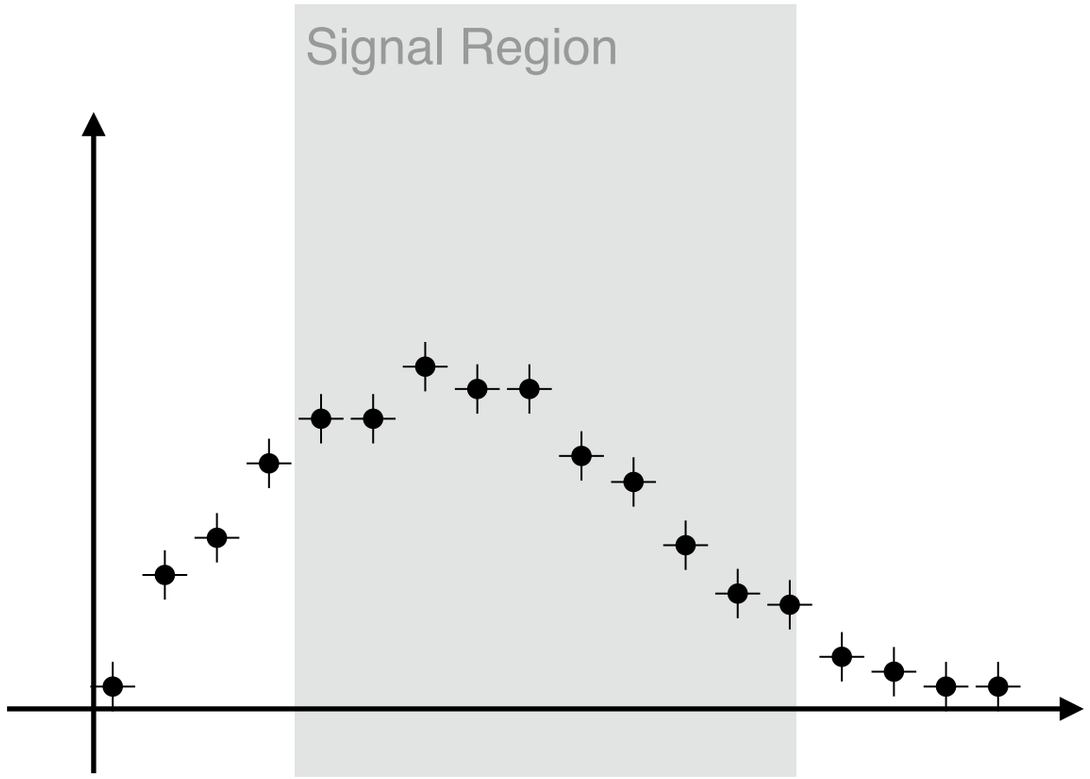






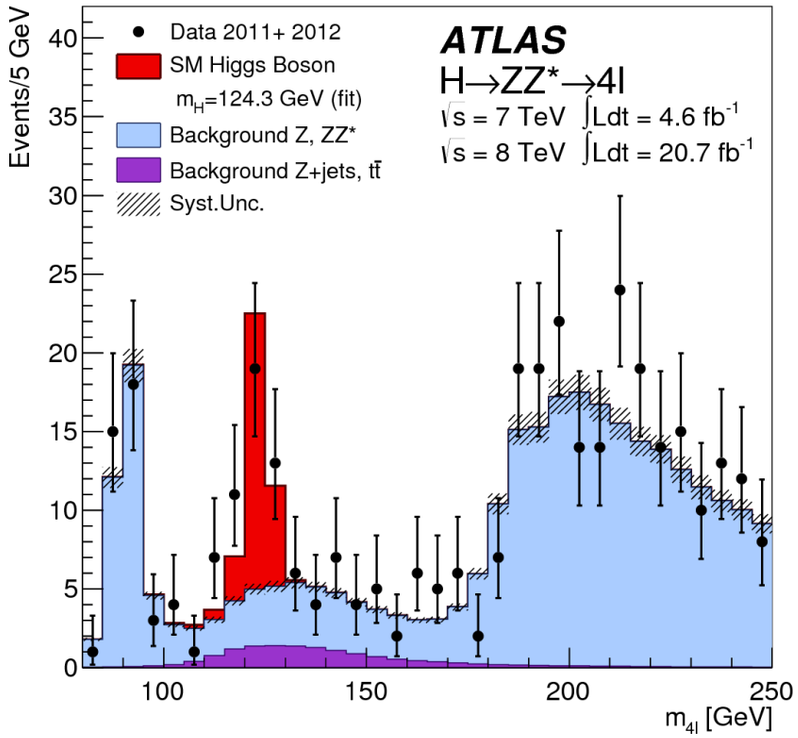
New Physics = Data - Known Physics

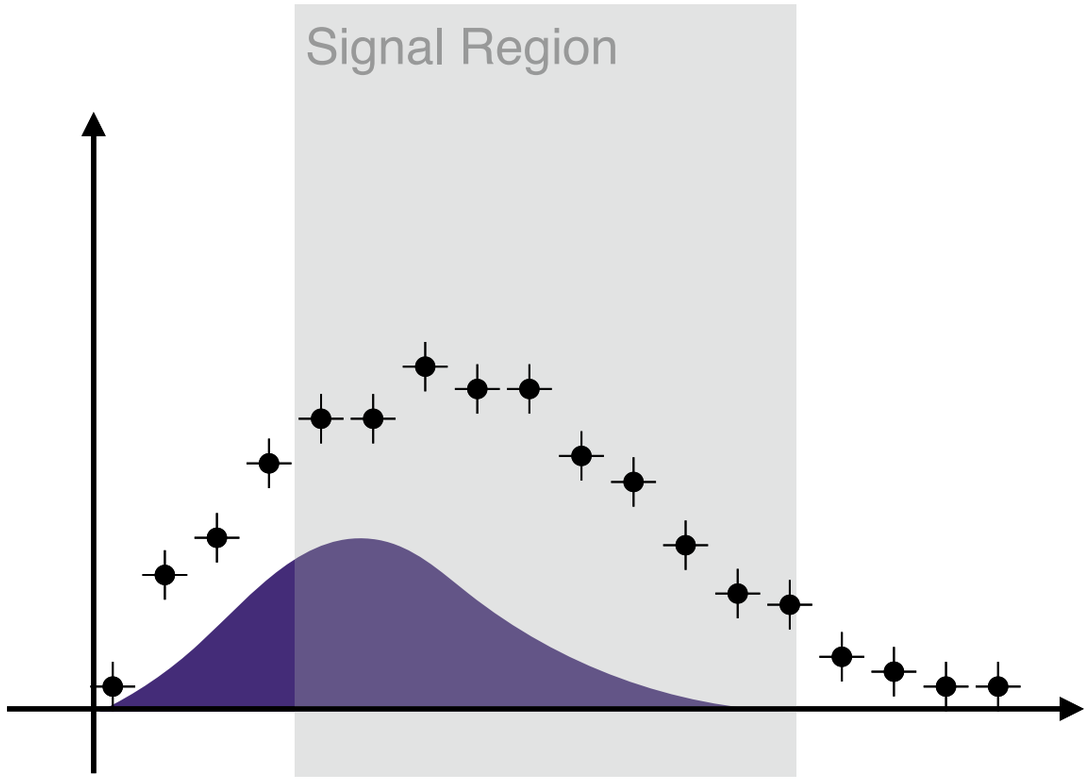




ATLAS data

New Physics = Data - Known Physics

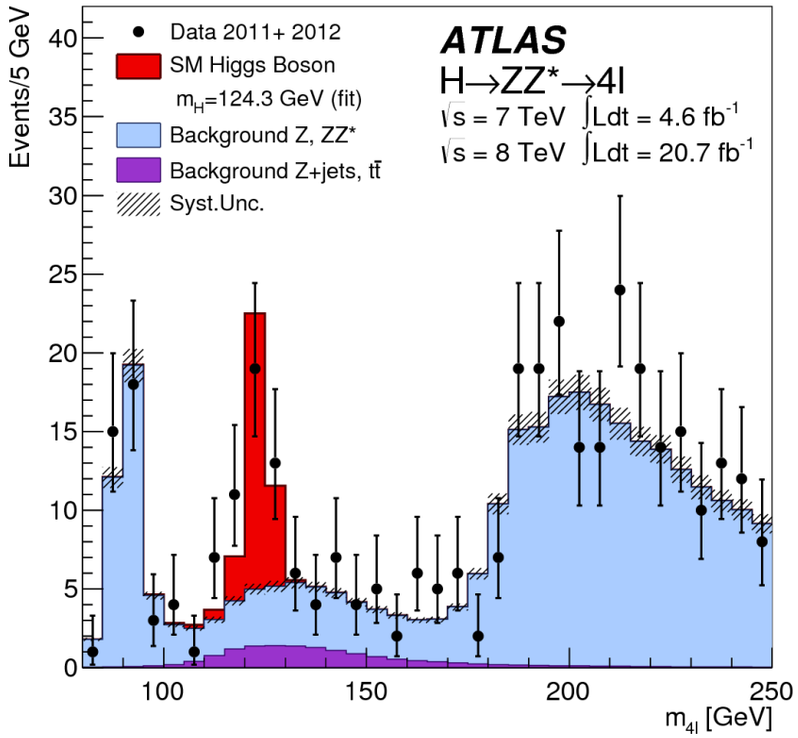


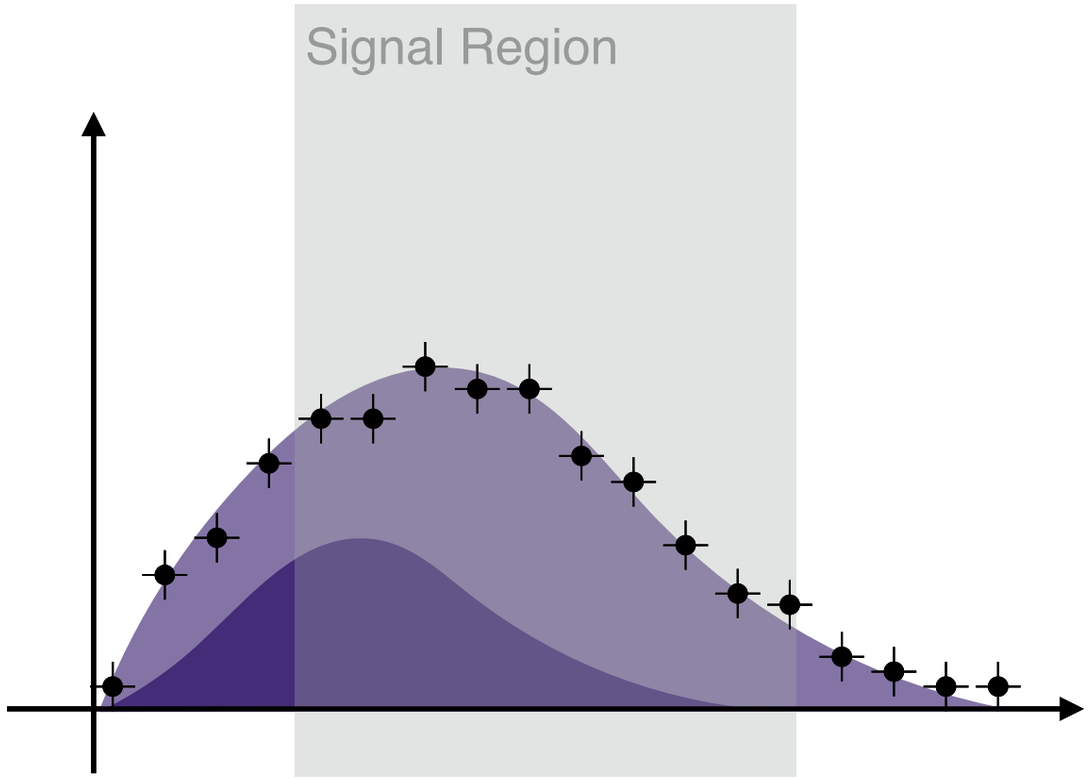


ATLAS data

Known Process 1

New Physics = Data - Known Physics





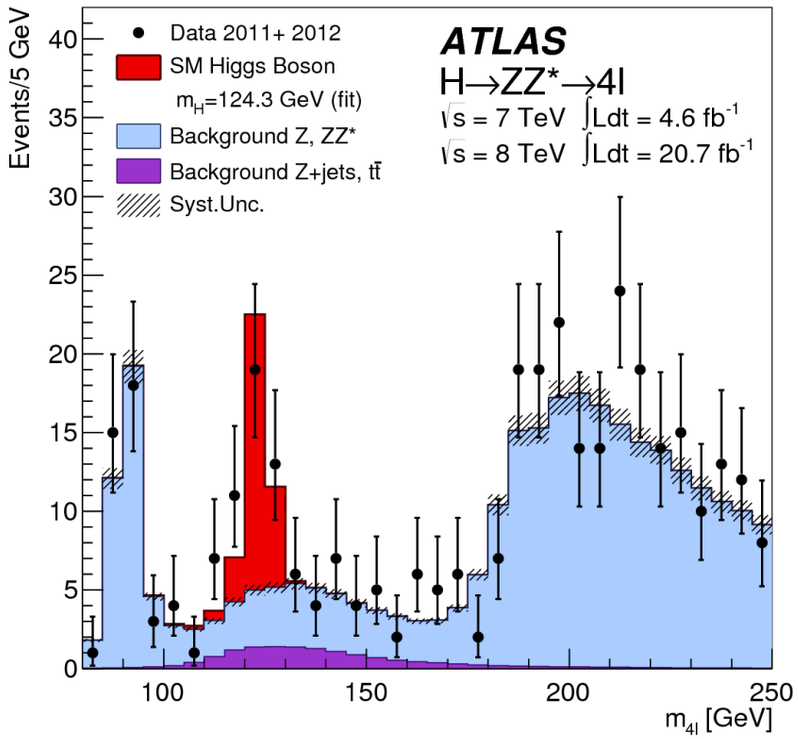
ATLAS data

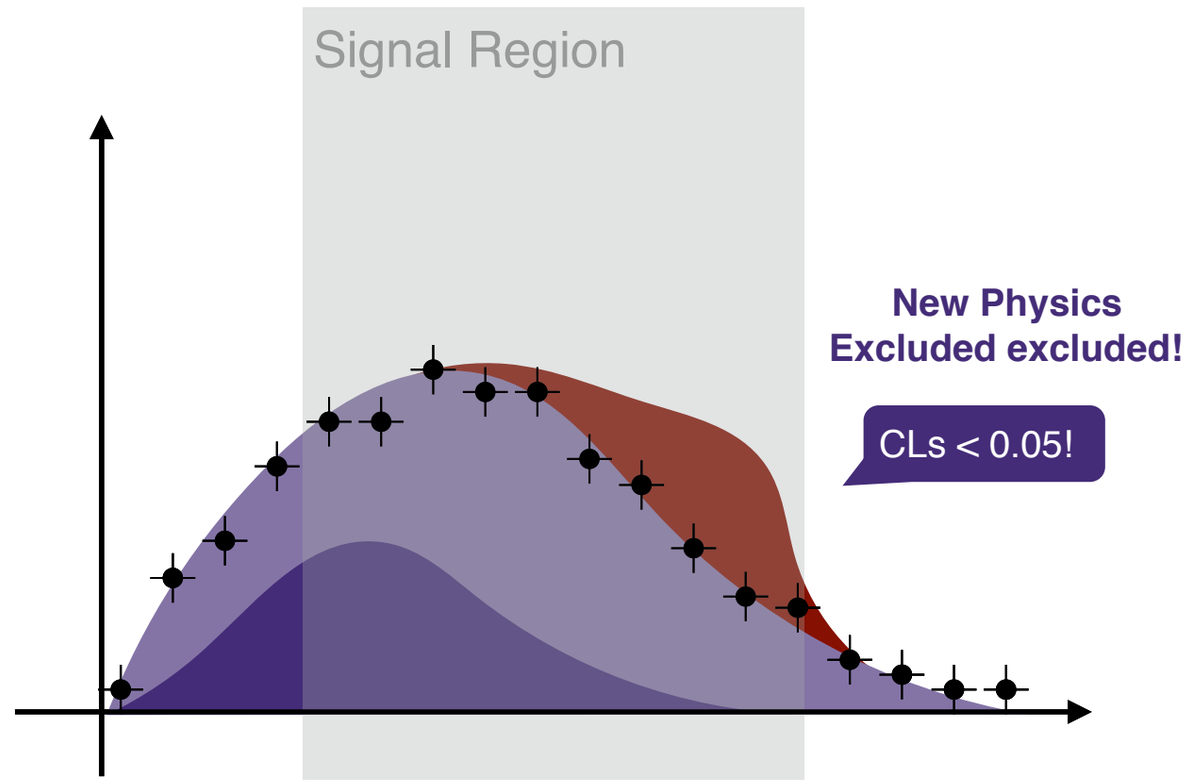
Known Process 1

Known Process 2

New Physics = Data - Known Physics

if Data ≈ Known Physics → We can exclude possible new phenomena





ATLAS data

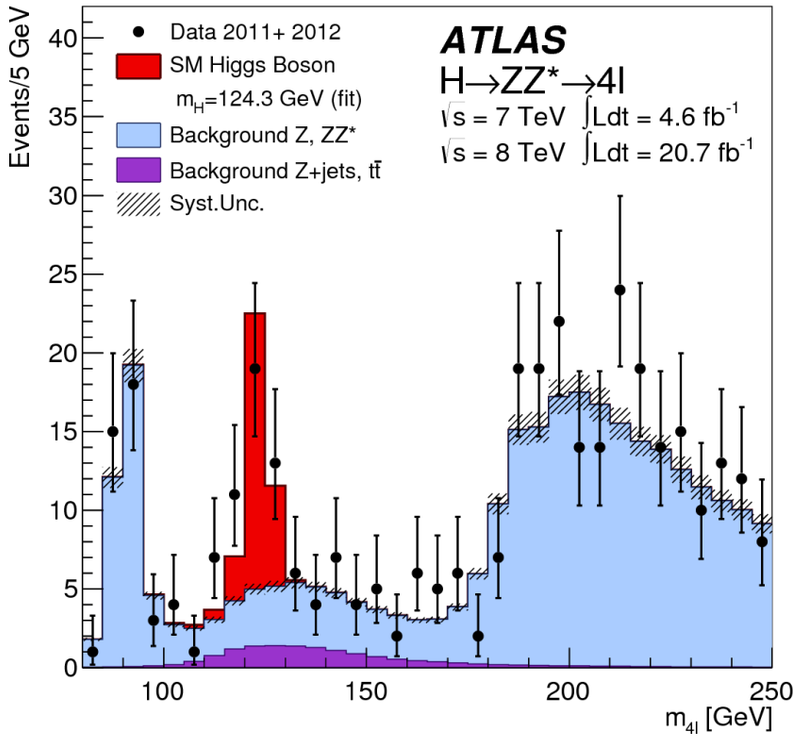
Known Process 1

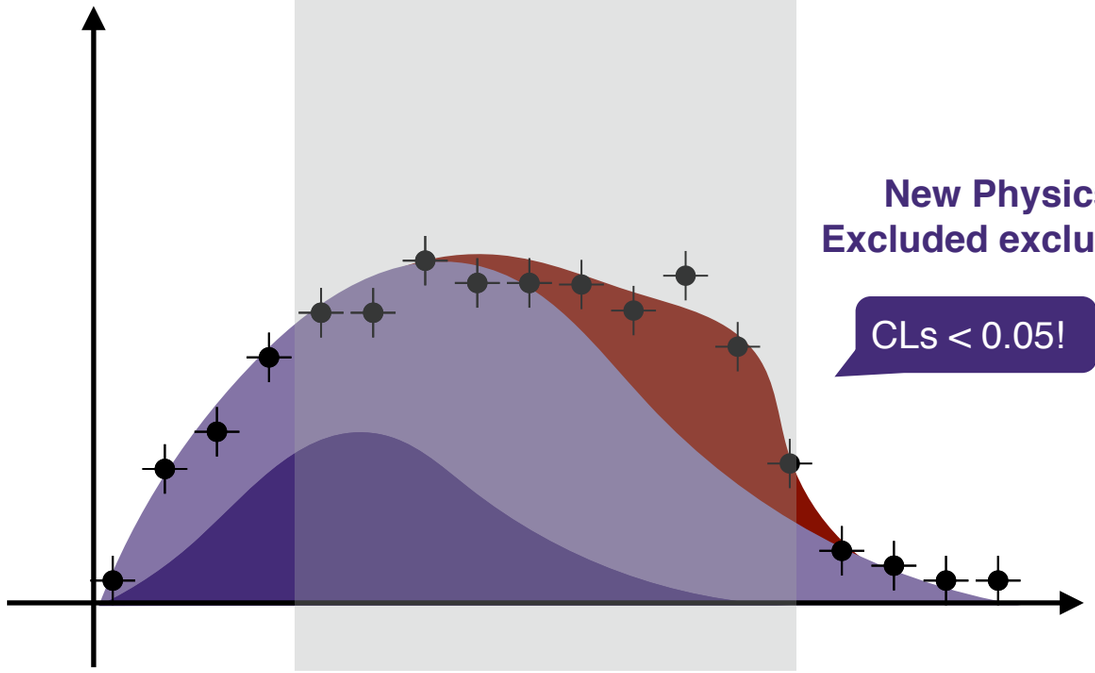
Known Process 2

Possible New Process

New Physics = Data - Known Physics

if Data ≠ Known Physics → Discovery





ATLAS data

Known Process 1

Known Process 2

Possible New Process

ATLAS Computing…

ATLAS Computing Usage

most of ATLAS computing resources are spent on two things:

1. simulating known physics + possible new physics

Simulate Physics Processes (e.g. production, decay of particles) + detector (full geometry)

2. data transformation + reduction

- **Transform** raw data (or pseudo-data in MC case) using multiple steps into higher and higher representations (raw calorimeter data → electromagnetic cluster → electrons)
- **Filter** data (only store some electrons, only store some variables of electron, e.g. Energy)

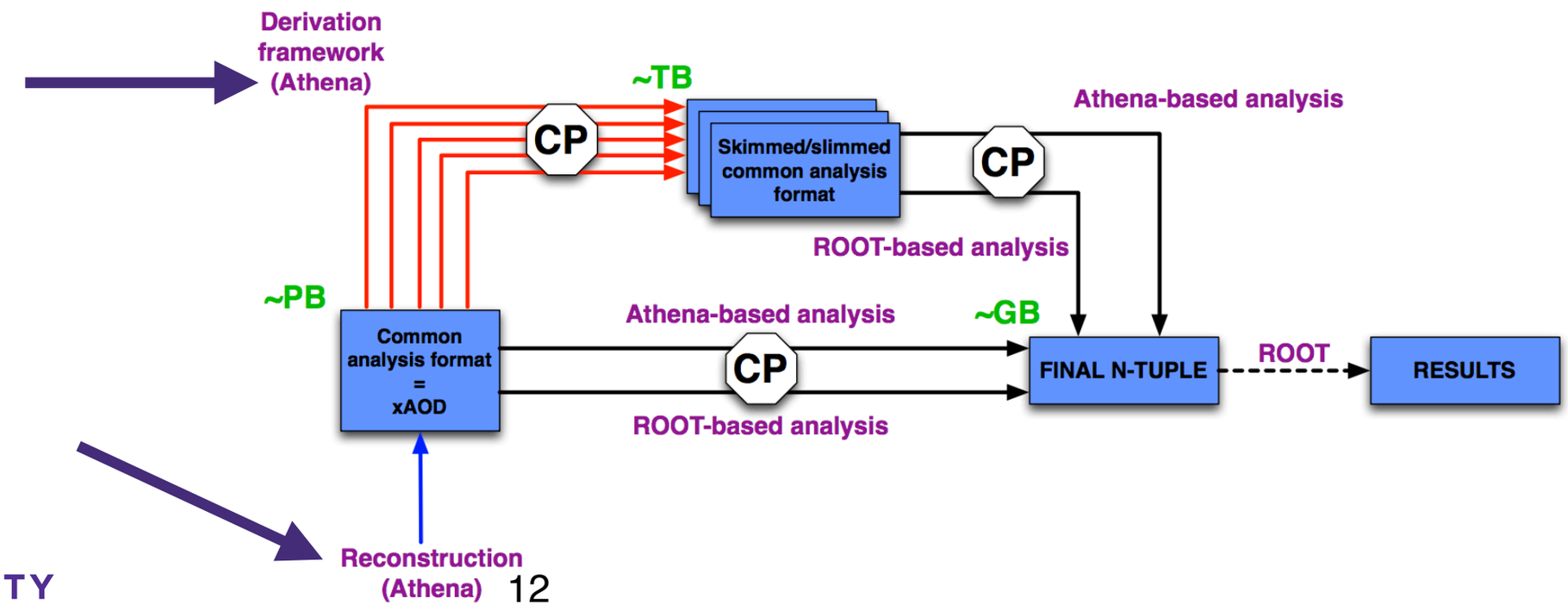
Essentially, a giant map-reduce.

ATLAS Software Stack

AtlasOffline

three main code bases:

1) “Atlas Offline”: Athena / Gaudi framework + associated packag
   - used for event generation, simulation and reconstruction.
     main job paradigm: data “format transforms”
   - used mainly by reconstruction experts, data preparation team or for detailed, low-level analyses / studies
   - central production of configurable reduces dataset for analysis via “derivation framework”
   - build system was “cmt", transitioning to cmake
   - mostly expert devs, highly structured release management (nightlies, etc..)

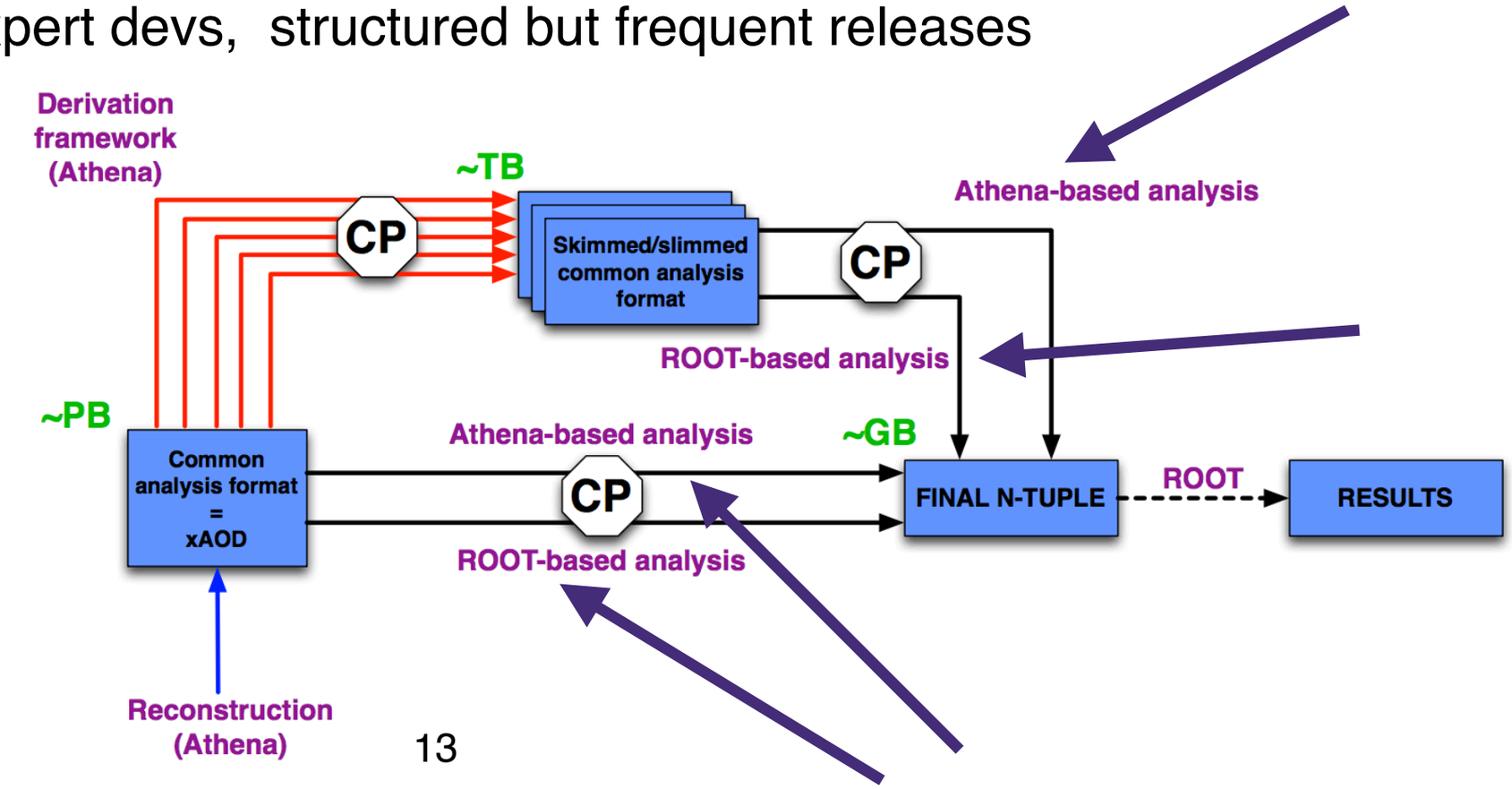


ATLAS Software Stack

AtlasOffline Analysis Releases

three main code bases:

2) “Analysis Releases”
- main entry-point / stack for physics analysis. Two flavors: a) Athena, b) pure ROOT frameworks with custom event loop, tool management
- build systems: cmt (for Athena), RootCore (for pure ROOT)
- packages (e.g. from performance and physics groups) written as “dual-use” compile and run in both environments
- common packages for common tasks
- expert and non-expert devs, structured but frequent releases

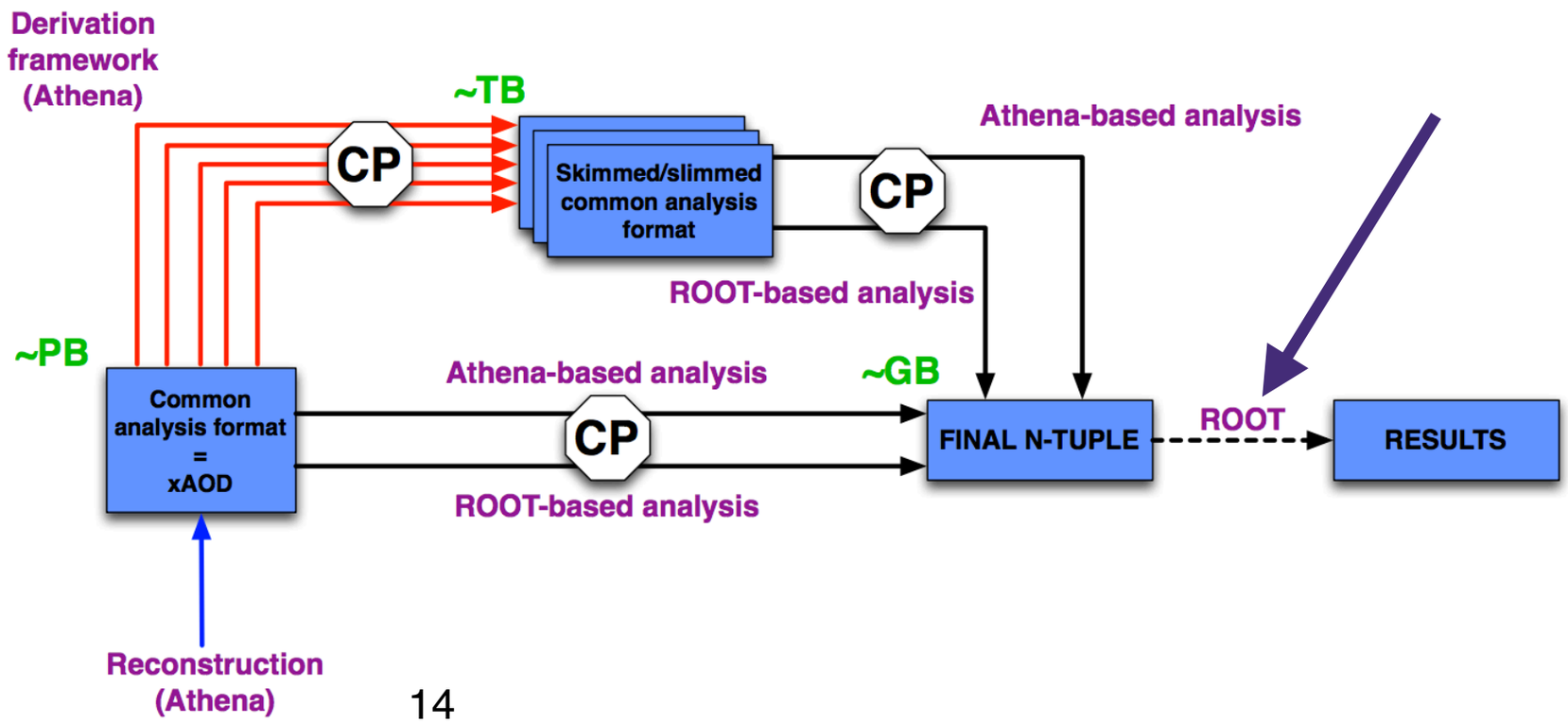


ATLAS Software Stack

AtlasOffline | Analysis Releases | User Code

three main code bases:

3) End-User Analysis Code
   - highly specific code for a given data analysis, not multi-use libraries
   - small teams, often graduate students
   - can be messy (one-off shell scripts, ROOT macros, …)
   - no standard development process / release managment (hopefully has repo)



Software Distribution in HEP

In HEP we develop code locally, but compute globally. We need efficient ways to distribute our code, reproducibly and efficiently.

Generally can choose between two options:

1. **Send Code to Remote Sites at Job Submission**

   Panda inputTarBall, HTCondor file transfer, PROOF archive.

   philosophy: send volatile bits, hope for longer-term dependencies (ATLAS releases, compilers, interpreters, etc…) to be fulfilled on remote site

2. **Have Code Available at Remote Site (shared, global filesystem)**

   - CVMFS, NFS, AFS, CephFS, etc…
   - transparent, same state between development and remote execution.
   - CVMFS as global read-only file-system is a massive success for LHC experiments
   - yet, many unknown bits, system-level dependencies not on CVMFS

Software Distribution in HEP

In HEP we develop code locally, but compute globally. We need efficient ways to distribute our code, reproducibly and efficiently.

Third Option: Containers

- Hybrid between shared filesystem + sending code (send full spec, but can benefit from caching)
- Singularity Containers are distributed via CVMFS
- Fully reproducible software environment

Container use-cases for ATLAS

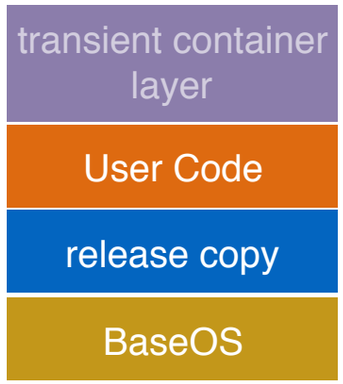
- reproducible, interactive development environments for personal development / software tutorials
- development env / job env parity on distributed/batch systems
- continuous integration, release testing
- analysis preservation and reusability

before we can use containers, we need images

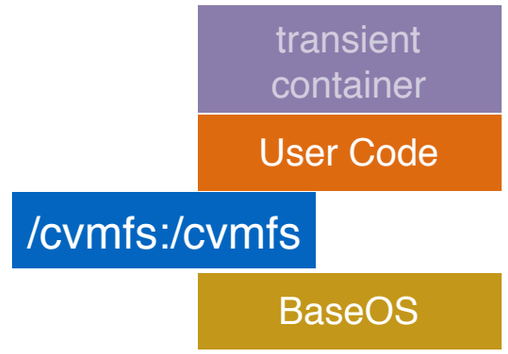
Running ATLAS workloads in in Containers

Three “macro” layers in s/w stack, various options how to get functioning container. Investigating all of them in ATLAS

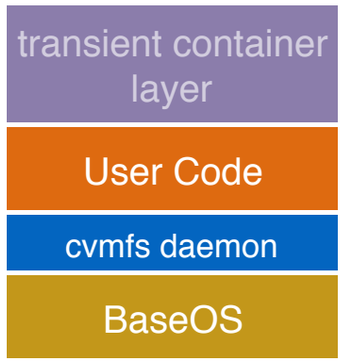
1) build full release into a container image



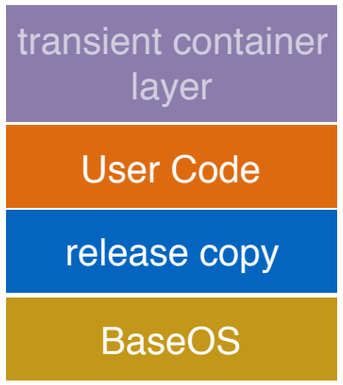
2) take release from external CVMFS mount (possibly tagged)



3) run cvmfs daemon inside container



Running ATLAS workloads in in Containers



1) build full release into a container image

Advantages:
best encapsulation, easiest for computational backends to handle (HTCondor w/ Docker Universe, GCE, Kubernetes on CERN OpenStack etc..)

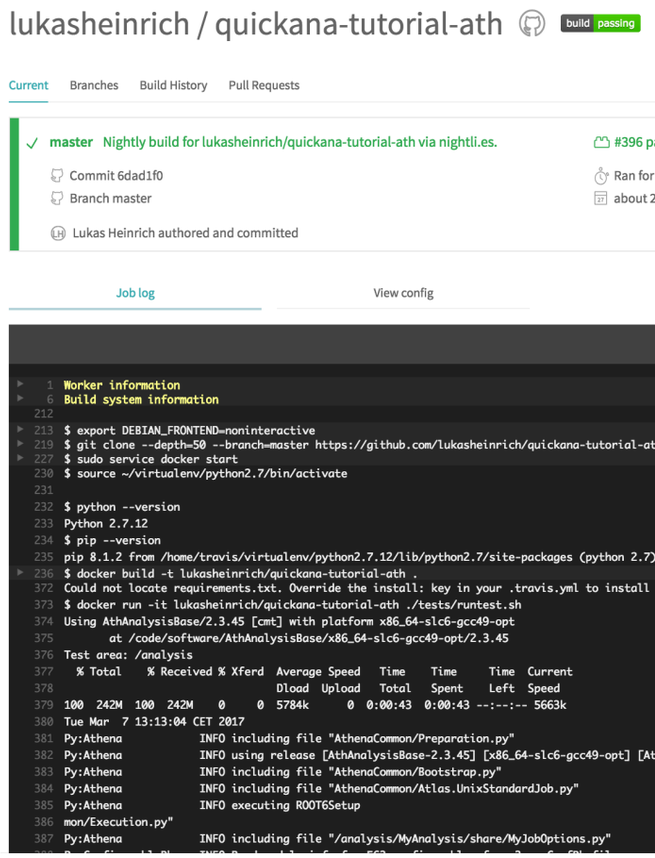
Disadvantages:
large image sizes 2-3GB for analysis releases, ~10GB for offline releases. will need to be careful in image building, to profit from layer caching. proliferation of images, needs clear image building / tagging workflow (try to learn form other large projects e.g. gcc..)
what about conditions/calibration data?

Ops Considerations:
when using Kubernetes with large images, large images and non-optimal network can become problematic (will back off of image pull).

Experience:
built a number of analysis releases using cmt build system[1] (both flavors), works very well on top of official cern/cc7-base

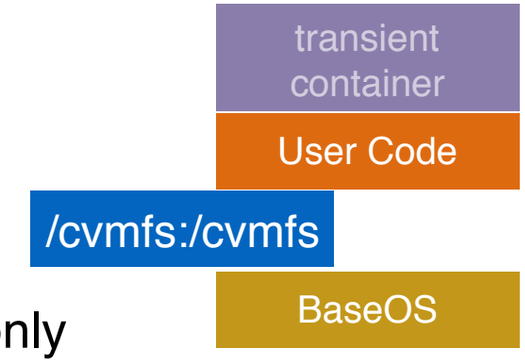
started to investigate AtlasOffline installation of cmake releases with promising results

Athena running on Travis CI



[1] Dockerfiles at github: lukasheinrich/asg-docker

Running ATLAS workloads in in Containers



2) cvmfs from external mount

Advantages:

easily achievable, just need base layer → small image sizes (even with user code).
No need to additional image management, can piggy-back off of cvmfs installations, only occasional updates to base image.

Disadvantages:

breaks encapsulation. container unusable without cvmfs. Difficult to add User Code layer (cannot use Dockerfile if you want to to non-trivial things, like compile user code using cvmfs. Workarounds by committing running containers.

Ops Considerations:

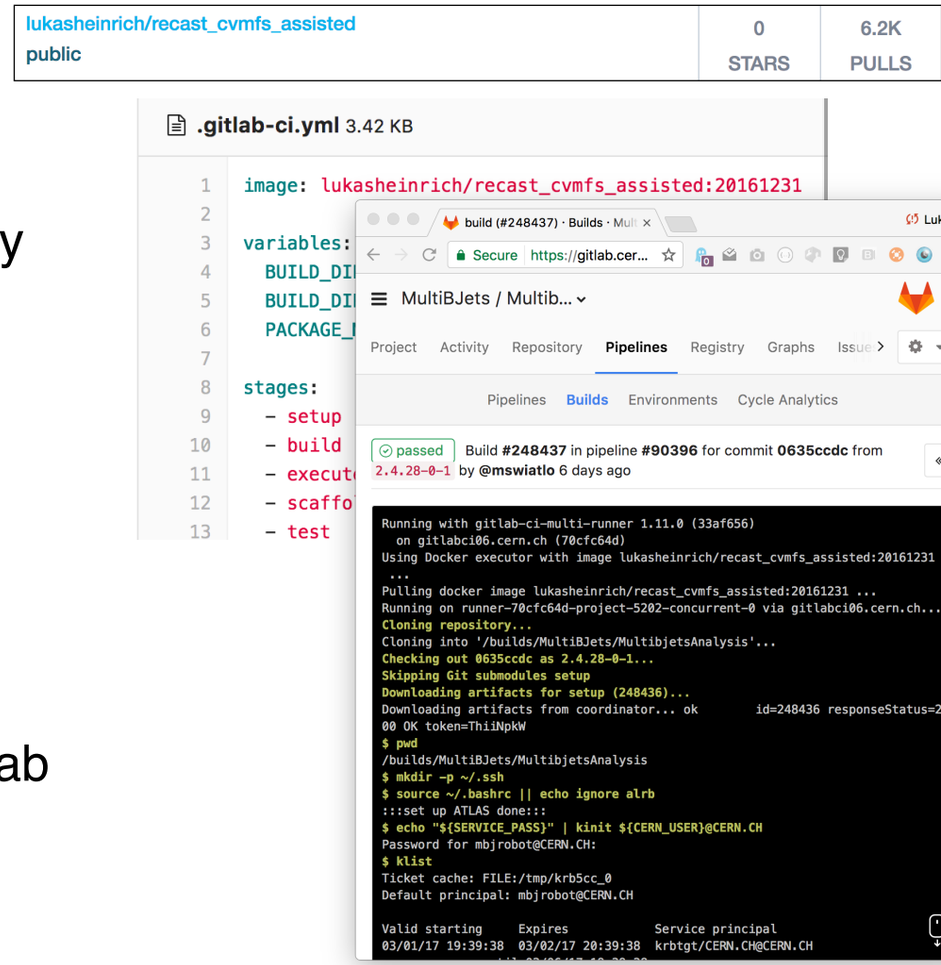
Very good experience with CVMFS volume plugin
Used for example by GitLab to provide CVMFS. Mounting /cvmfs directly tricky when auto-mounting (stale file descriptors)

encapsulation breakage can be mitigated by using specific cvmfs tag

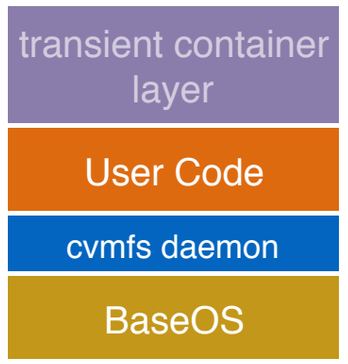
```
-v atlas.cern.ch#<some-hash>:/cvmfs/atlas.cern.ch
-v atlas.cern.ch@trunk-previous:/cvmfs/atlas.cern.ch
```

Experience:

Easy on-boarding of analysis groups to use it as CI base image on GitLab



Running ATLAS workloads in in Containers



3) cvmfs inside container

Advantages:

neither need to have cvmfs installed on host, nor release installed in image. works “out of the box”.

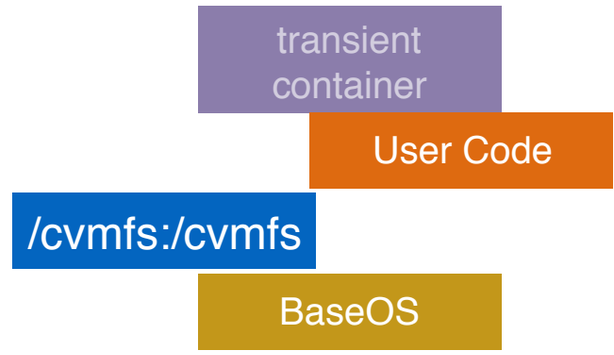
Disadvantages:

still breaks encapsulation. each container runs its own cvmfs cache (probably needs to be in a volume, difficultt to share across containers)

Experience:

Works, with sufficiently well-prepared base image, but not tested extensively. Can be a good option for software tutorials (quick image pull, but no cvmfs on host required)

Containers on Grid / Batch systems



Batch systems are starting to support containers:

- HTCondor Docker Universe
- Singularity transparently supported (single binary, no daemon). Used at HPC centers (NERSC). Works with e.g. SLURM, etc.

Grid sites are moving to containerized infrastructure to ease their maintenance

- decouple site OS from experiment OS (e.g. cluster on RHEL7, ATLAS jobs on slc6)
- use orchestration tools (Mesos, Kubernetes) to scale their grid site
- nothing changes for users, at this level purely sys-admin stuff. Experiments provide base images, that are curried with CVMFS and User Code

More fundamental shift possible: **Let Users define container image to run their job in** at GRID middleware layer.

Instead of:

```
prun exec –inputTarBall mysoftware.tar.gz –cmtconfig “x86_64-slc6-gcc47-opt” –inputData data12.foo.bar/
```

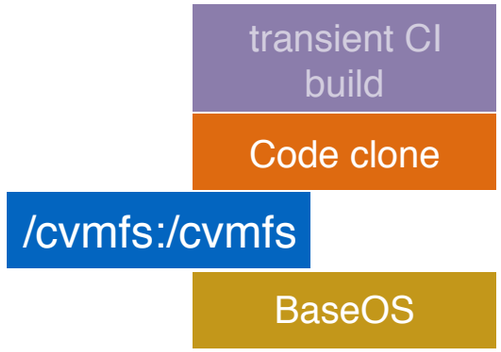
allow

```
prun exec –containerImage lukasheinrich/mysoftware –inputData data12.foo.bar/
```

Potentially Fundamental Change on How we run our code

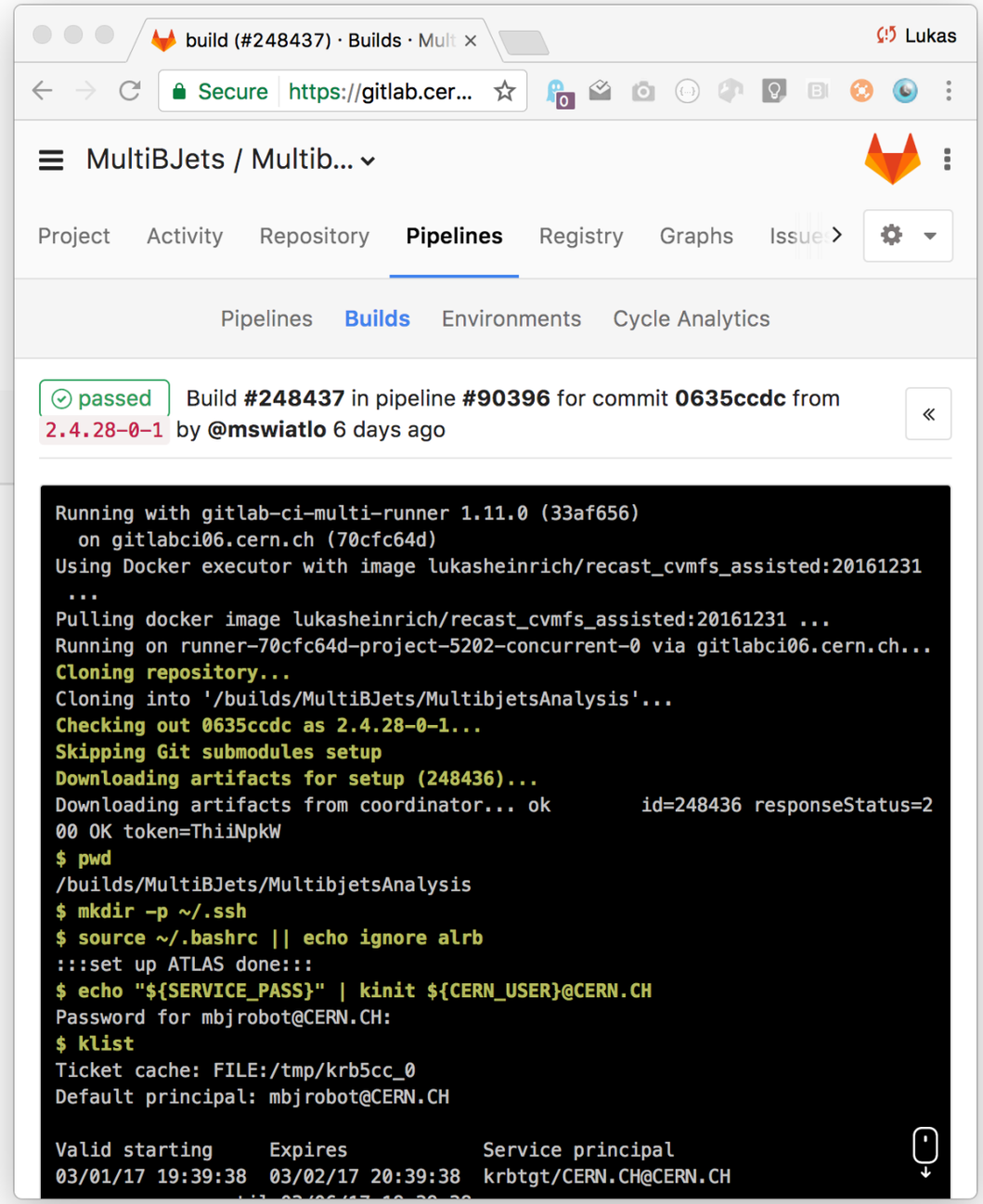
(not so far future: GRID = globally federated kubernetes clusters with kubernetes jobs?)

Containers for continuous integration



main ATLAS code base tested continuously with Jenkins etc, many individual analysis teams moving to GitLab

- CI builds on top of docker base images work great!
- CVMFS plugin makes option (2) feasible in this setting

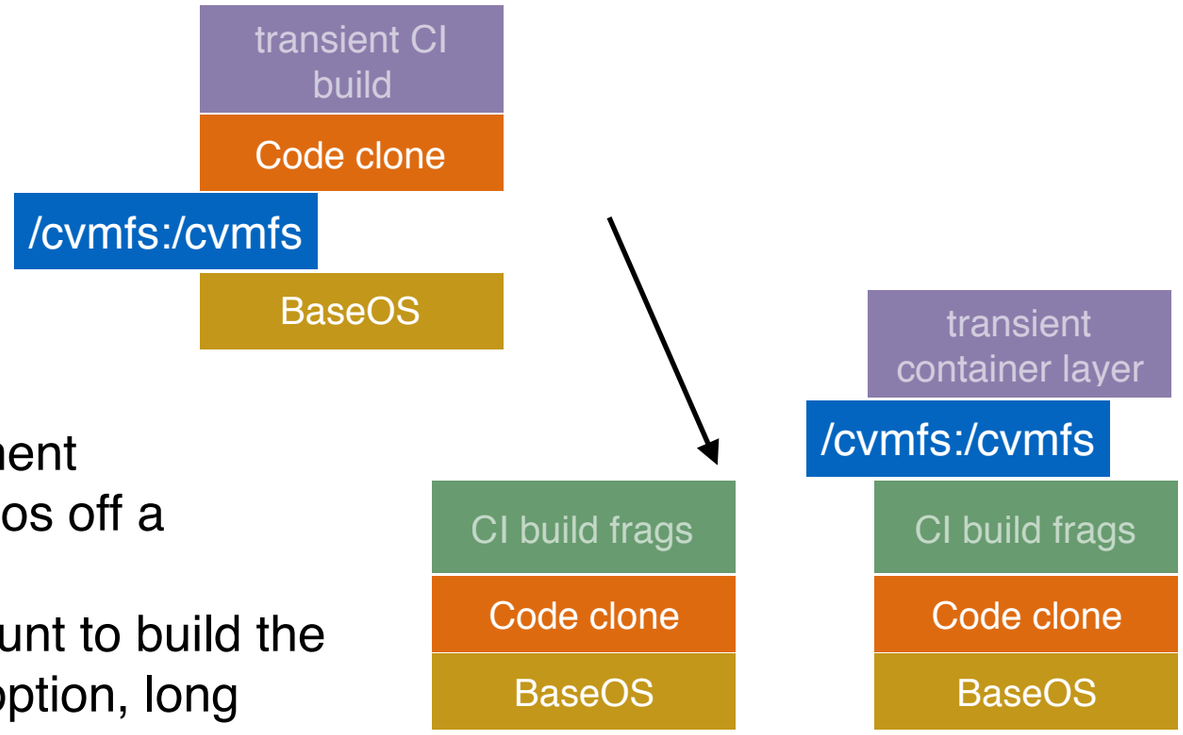


.gitlab-ci.yml 3.42 KB

```
1  image: lukasheinrich/recast_cvmfs_assisted:20161231
2
3  variables:
4    BUILD_DIR: MBJ_TestArea
5    BUILD_DIR_ABS: "${CI_PROJECT_DIR}/${BUILD_DIR}"
6    PACKAGE_NAME: MultibjetsAnalysis
7
8  stages:
9    - setup
10   - build
11   - execute
12   - scaffold
13   - test
```

Docker images as build fragments

Want: build docker images automatically during CI



- Running jobs during CI build works
- but ideally want to have *docker image* as a build fragment
- CERN GitLab offers dedicated runners for building repos off a Dockerfile
- **But**: most in option (2) style setup, we need cvmfs mount to build the code. Can't build via Dockerfile (no build-time only -v option, long issue docker/issues/14080)
- currently, need hacks, would be great if this could somehow be supported centrally, since very CERN-specific problem
- Curiously need to rm -rf /cvmfs in separate step to make it work

lab-ci.yml 1.08 KB Raw | Blame | History | Permalink | Edit | Replace | Delete

```
testjob:
  tags:
    - rocha-swarm-builder
  image: docker.io/lukasheinrich/codecapture
  script:
  - docker volume create -d cvmfs --name=atlas-condb.cern.ch
  - docker volume create -d cvmfs --name=atlas.cern.ch
  - docker volume create -d cvmfs --name=sft.cern.ch
  - docker inspect -f '{{json .Mounts}}' "$(hostname)-build" | jq '.[]|select(.Destination|match("/build"))'> /tmp/buildmount.json
  - export MOUNT=$(jq '"\(.Source):\(.Destination)"' /tmp/buildmount.json|xargs echo)
  - docker login -u $DOCKERHUB_USER -p $DOCKERHUB_PW -e lukas.heinrich@gmail.com
  - docker run  --security-opt label:disable -v $MOUNT -v atlas.cern.ch:/cvmfs/atlas.cern.ch -e BUILDSOURCE=$PWD --cidfile build.cid -w /code lukasheinrich/recast_c
  - docker commit -c 'WORKDIR /code' $(cat build.cid) docker.io/lukasheinrich/gitlab-codecapture-demo-prebuild
  - printf "FROM docker.io/lukasheinrich/gitlab-codecapture-demo-prebuild\nRUN rm -rf /cvmfs" | docker build -t docker.io/lukasheinrich/gitlab-codecapture-demo -
  - docker push docker.io/lukasheinrich/gitlab-codecapture-demo
```

Container use-cases for ATLAS

- reproducible, interactive development environments for personal development / software tutorials
- development env / job env parity on distributed/batch systems
- continuous integration, release testing
- **analysis preservation and reusability**

Recent Successes in Reusability…

It’s the difference between if you had airplanes where you threw away an airplane after every flight, versus you could reuse them multiple times.

— Elon Musk

It's the difference between if you had ~~airplanes~~ where you threw away an ~~airplane~~ after ~~every flight~~, versus you could reuse them multiple times.

— Elon Musk

RECAST

NEW YORK UNIVERSITY

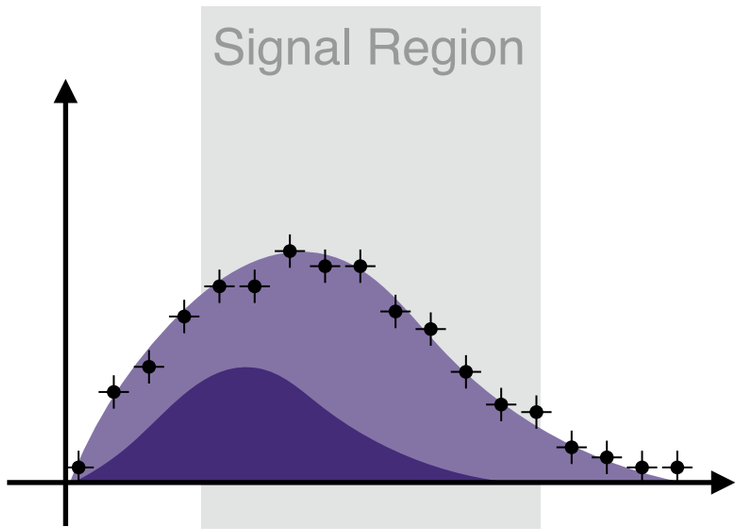
The analyses ATLAS prepares are **high-effort, expensive projects**: non-trivial amount of person-power, time, and computing resources devoted to achieving a publication-quality result.

Most of the work goes into: **taking data, designing**, **validating** the analysis strategy, **understanding Standard Model backgrounds.** Effectively: a measurement of observed and backgrounds in interesting phase space regions.

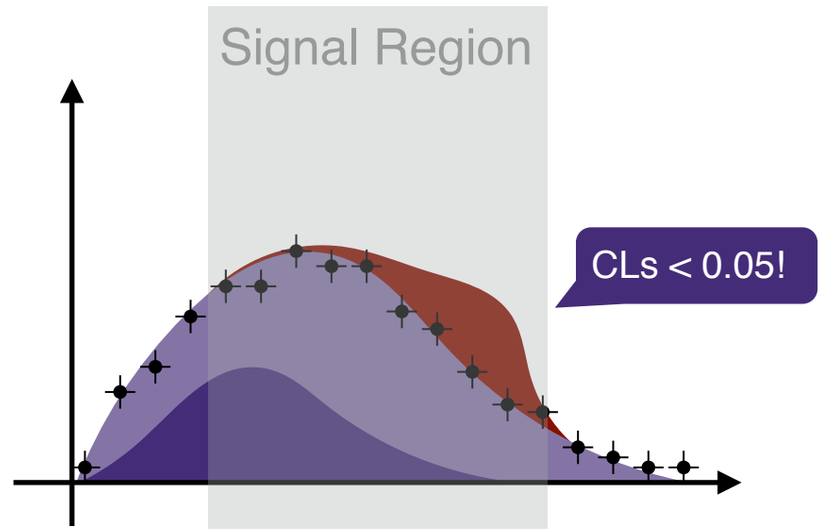
Model interpretation come at the end, and are technically the **easiest part:** analysis pipeline is **fixed** after unblinding, MC dataset sizes small. Analysis teams routinely check hundreds of parameter points (of their favorite model).

But: most analyses only **interpreted once** within limited set of models.

- analysis team pushing for conference deadline
- interesting models proposed by hep-ph *after* they've seen the paper / note.



observed **data** and
estimated backgrounds



interpreted w.r.t **model A**

The analyses ATLAS prepares are **high-effort, expensive projects**: non-trivial amount of person-power, time, and computing resources devoted to achieving a publication-quality result.

Most of the work goes into: **taking data, designing**, **validating** the analysis strategy, **understanding Standard Model backgrounds.** Effectively: a measurement of observed and backgrounds in interesting phase space regions.

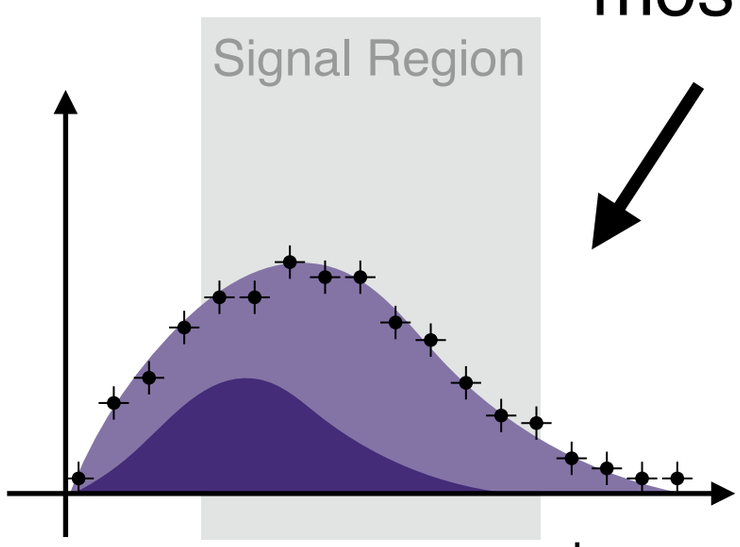
Model interpretation come at the end, and are technically the **easiest part:** analysis pipeline is **fixed** after unblinding, MC dataset sizes small. Analysis teams routinely check hundreds of parameter points (of their favorite model).

But: most analyses only **interpreted once** within limited set of models.

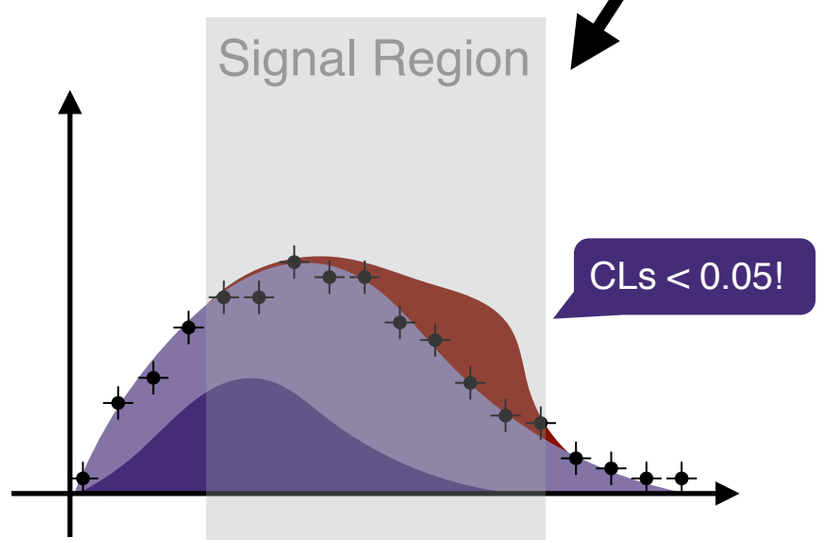
- analysis team pushing for conference deadline
- interesting models proposed by hep-ph *after* they've seen the paper / note.

~ easy

most of the work!



observed **data** and
estimated backgrounds



interpreted w.r.t **model A**

how to maximize scientific output of an analysis?

how to maximize scientific output of an analysis?

make analyses (rapidly) reusable.

NEW YORK UNIVERSITY

Reinterpretation Recipe:

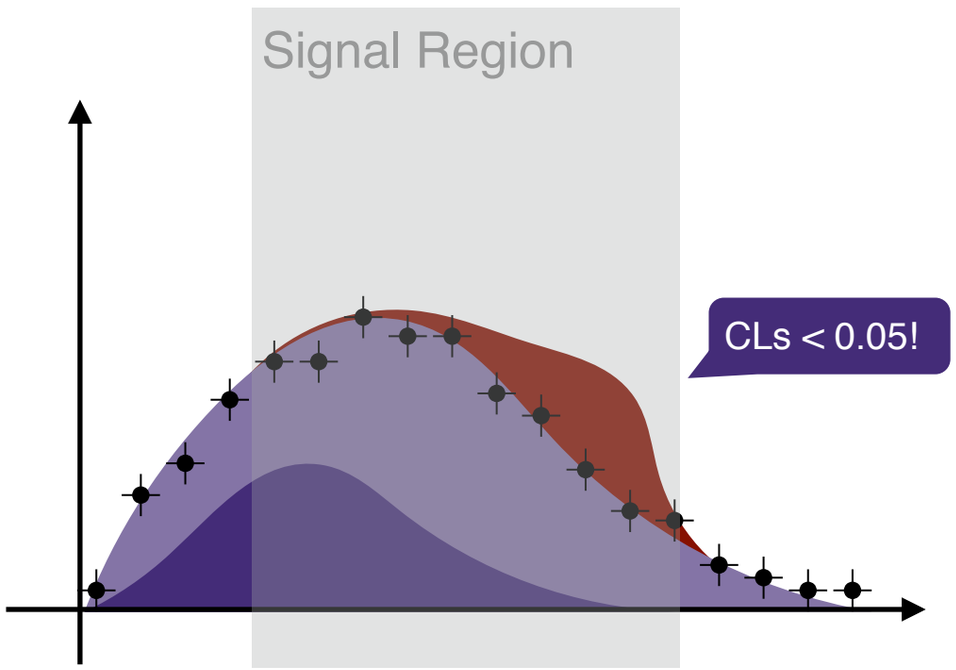
At publication time:

1. archive data + background estimates
2. **preserve original analysis pipeline** (at least such that we can run new signal sample).

Later:

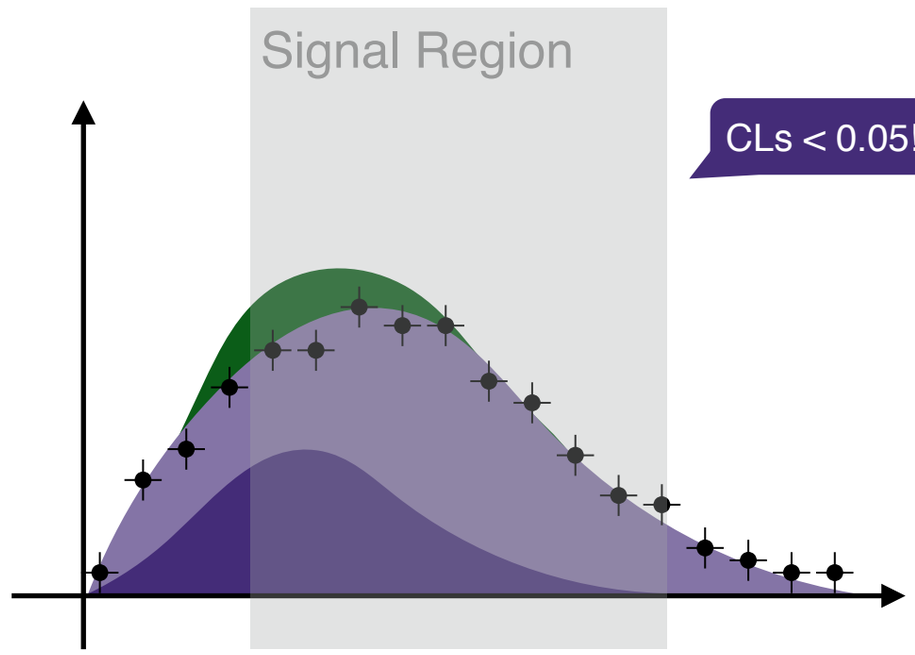
1. Generate **new signal dataset** with same/compatible settings as original analysis (simulation, reconstruction, etc…)
2. Run dataset through **original analysis pipeline,** compare/fit against archived data and backgrounds.

Result: Limit data (exlusion) for a new signal sample



original analysis (w.r.t **model A**)

RECAST



original analysis (recast to **model B**)

we need to be able to preserve analysis pipelines…
… such that we can re-execute them on *new input*

i.e. *parametrized* analysis preservation

Analysis Preservation Efforts / Reinterpretation in ATLAS

Growing topic in LHC experiments as we're looking to optimally exploit science output.

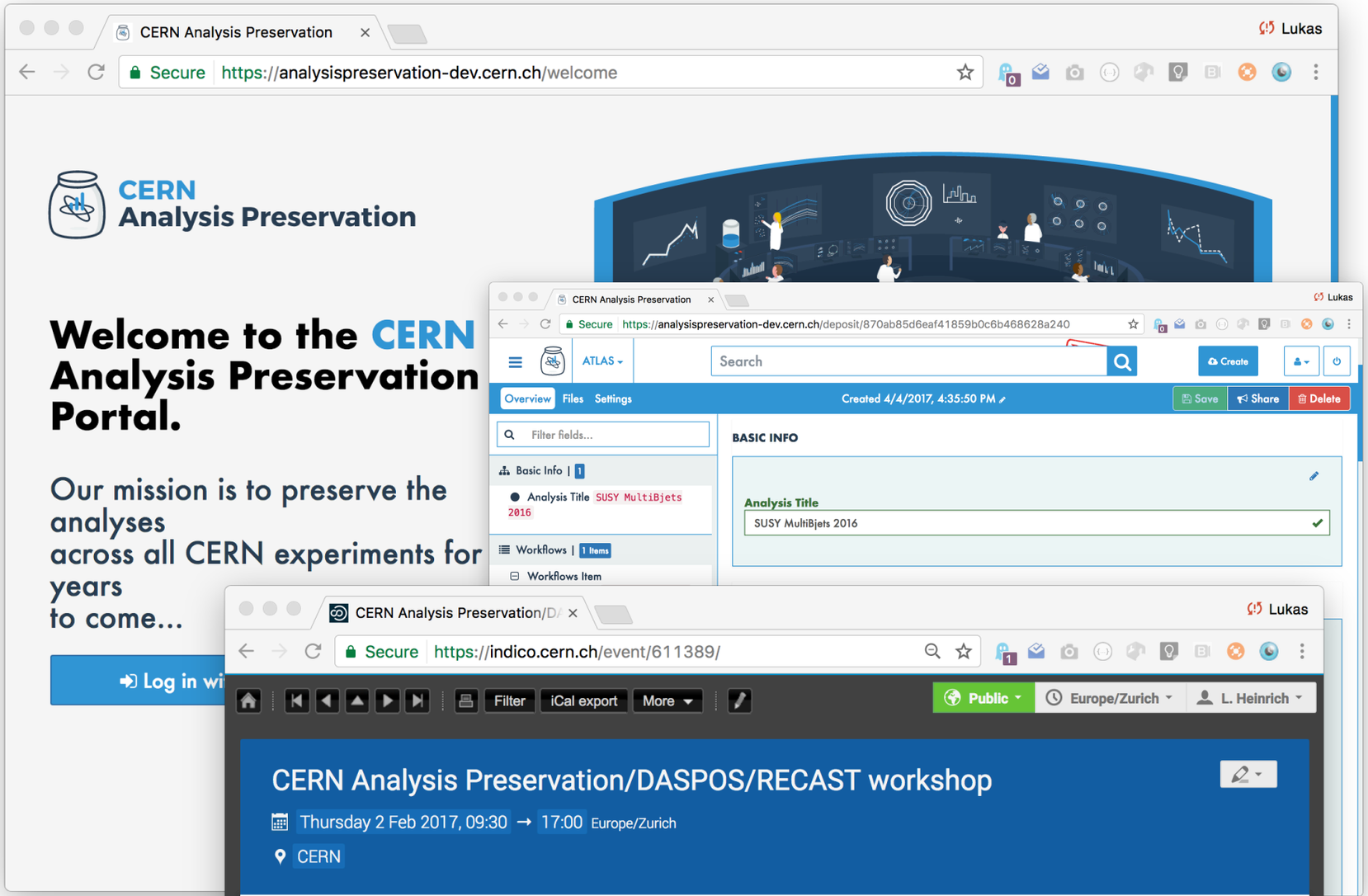
ATLAS activity coordinated with LHC-wide efforts of a "**CERN Analysis Preservation Portal**" (CAP) service. Closely related to DPHEP (data preservation), DASPOS (software preservation), DIANA-HEP (modern HEP software infrastructure) grants/projects.

DPHEP

DASPOS

dianahep

NEW YORK UNIVERSITY



Analysis Preservation: two-step process

Modern HEP analysis:

- Multiple steps/code-bases, possibly developed by independent teams, with differing software requirements. Example: one team developing the event selection, another team developing the statistical analysis

Need to capture:

1. Individual processing steps

- code bases
- software environments
- identify binaries, scripts in code base
- templates how to run binaries (semantic description of arguments, naming etc..)
- description of step output, what are the relevant data fragments

2. How to connect these steps

- How to wire individual steps together
- What outputs of which steps, are used as inputs for other steps, …

Goal: capture all this with least amount of work for analysis teams, preferably *while analysis is being developed*. *Should not take make than a few days*

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)

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

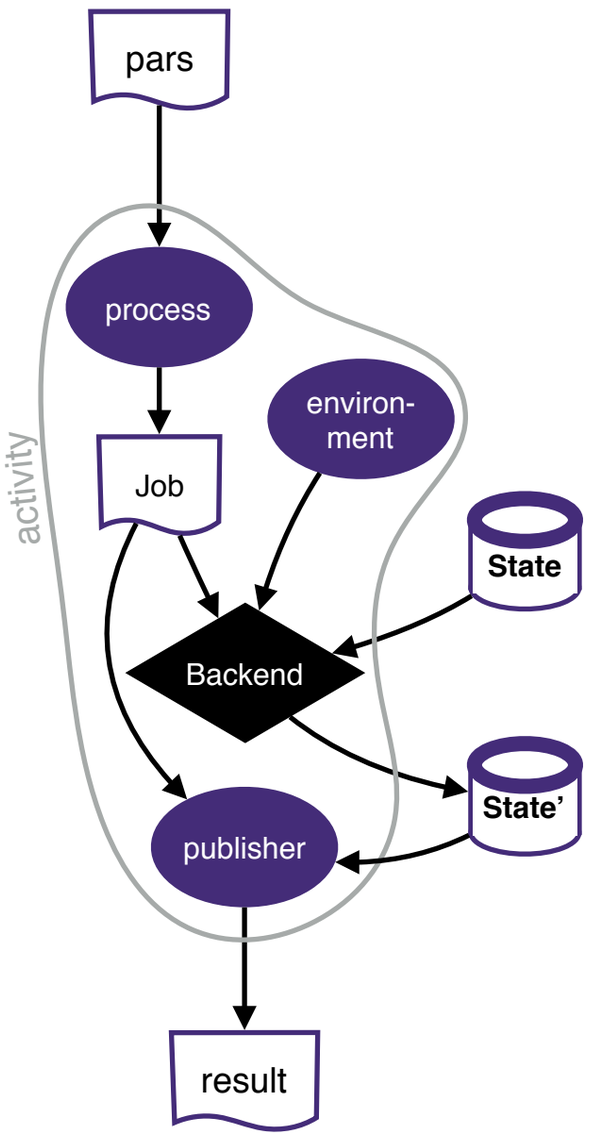
It’s useful to have machine readable result data to e.g. identify newly created files.

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.
Best bet: *containers*

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



1. Problem: Preserve Individual Processing Steps

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

Data Format: JSON

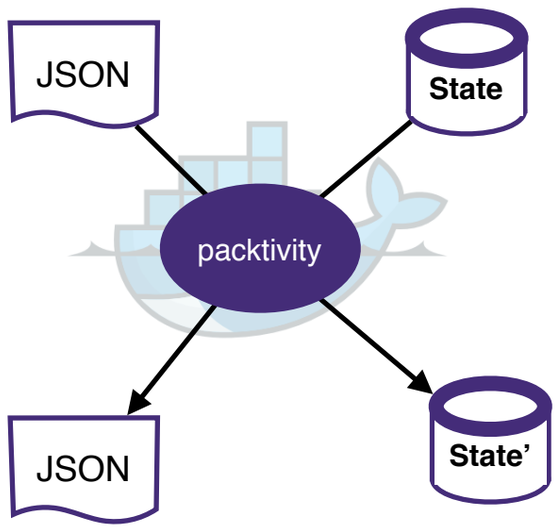
- as interchange format for parameters and result data
- as declarative description format for *process/env/publisher*
  - incl. JSON schemas for validation

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

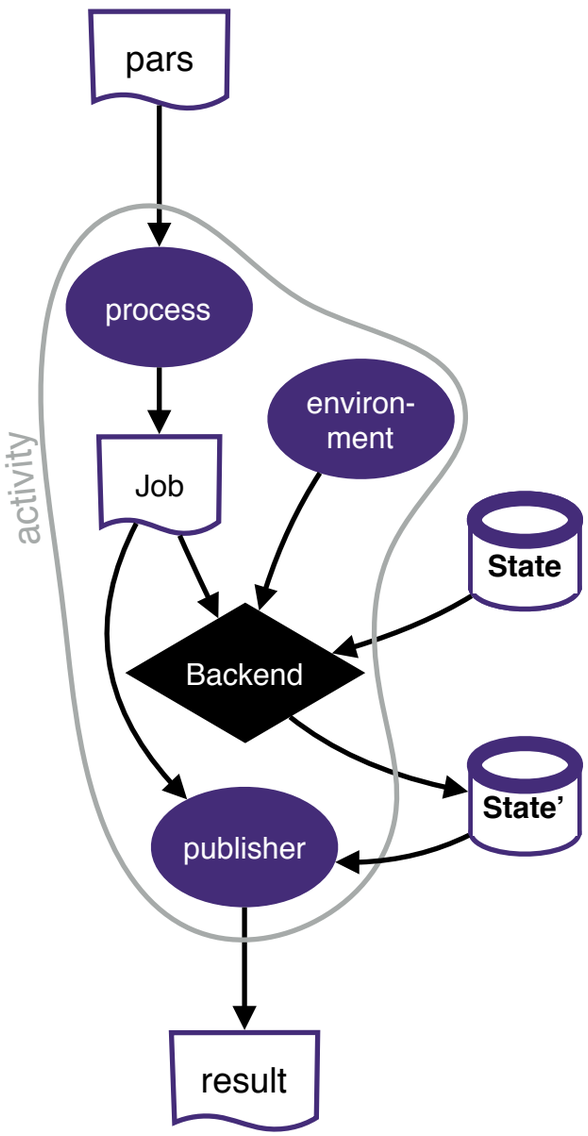
- JSON API
- archivable, declarative description as JSON
- dependencies captured in environment
  - e.g. Docker Image

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

=



=

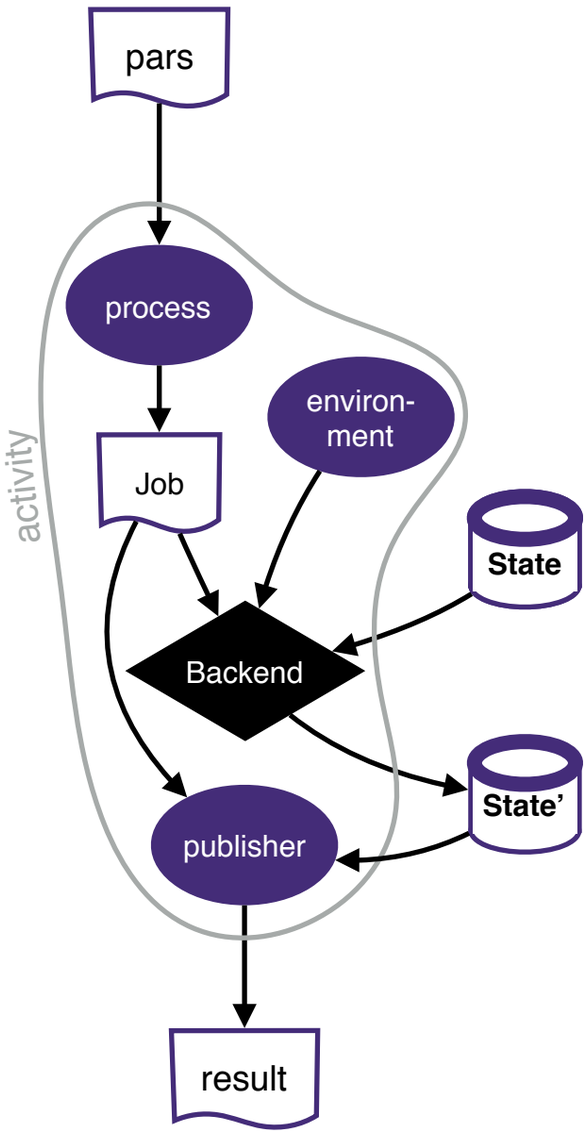


1. Problem: Preserve Individual Processing Steps

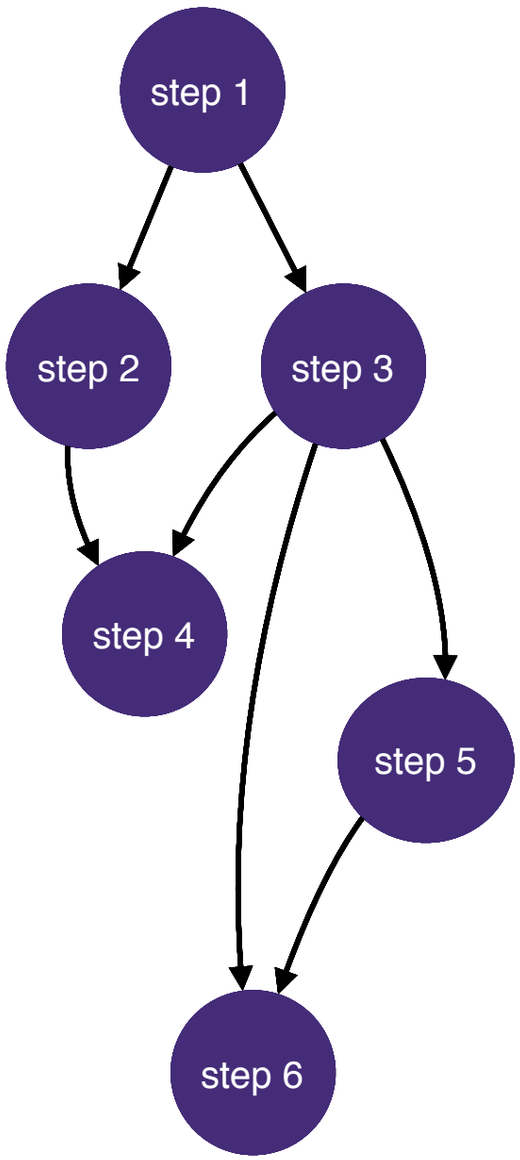
(Example: Run Detector Simulation + Reconstruction on MC events)

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
```



2. Problem: Preserve Parametrized Workflow



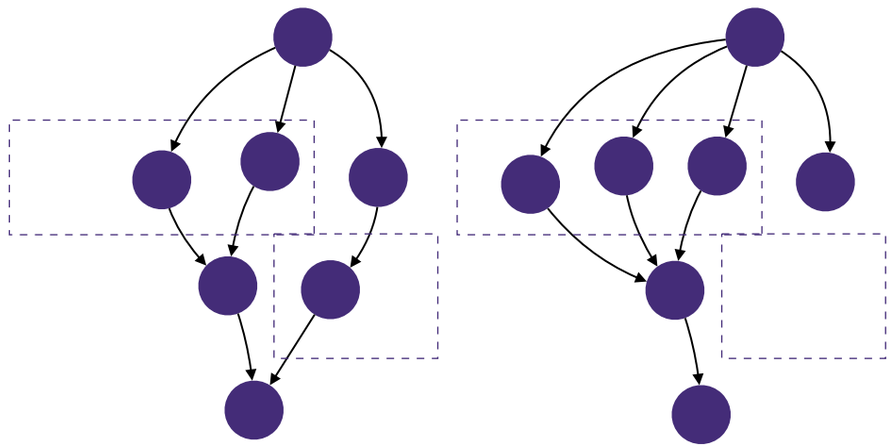
Natural Data Model: *directed acyclic graphs (DAGs)*

- **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

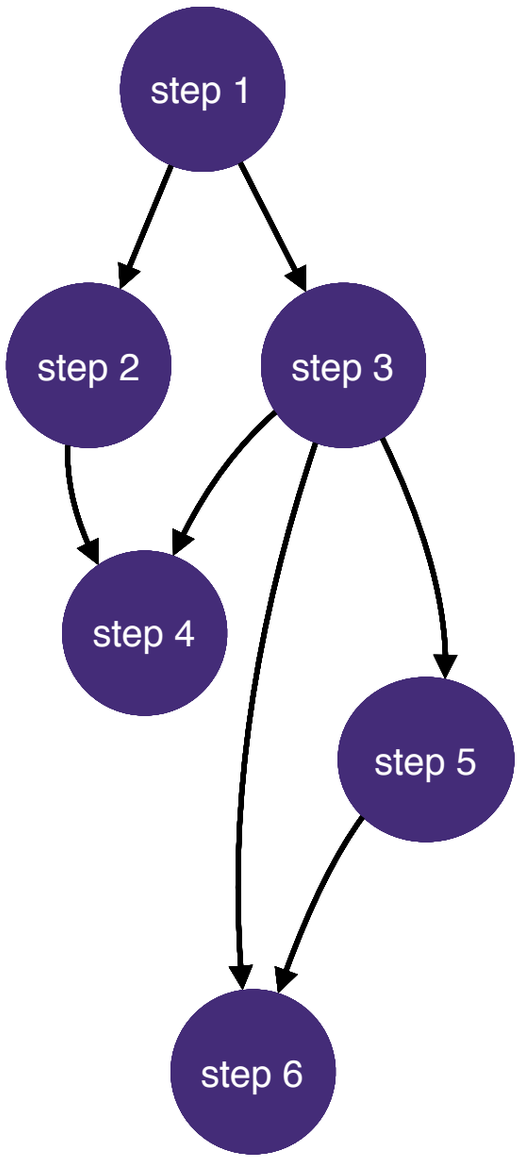


Par. Set 1 **Par. Set 2**

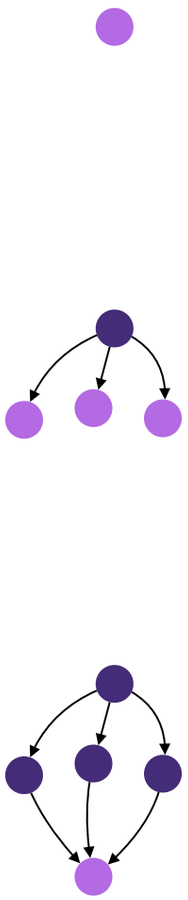
2. Problem: Preserve Parametrized Workflow

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

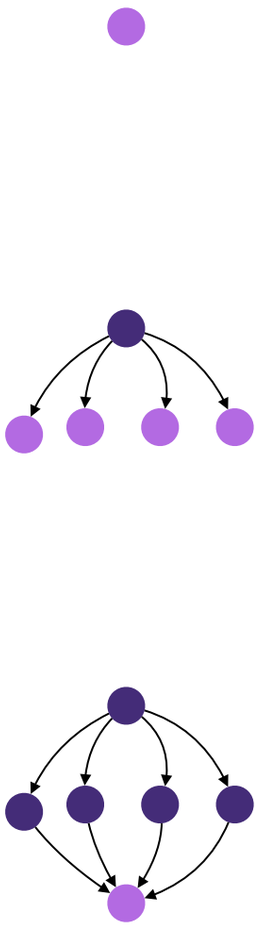


Par. Set 1

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
(**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



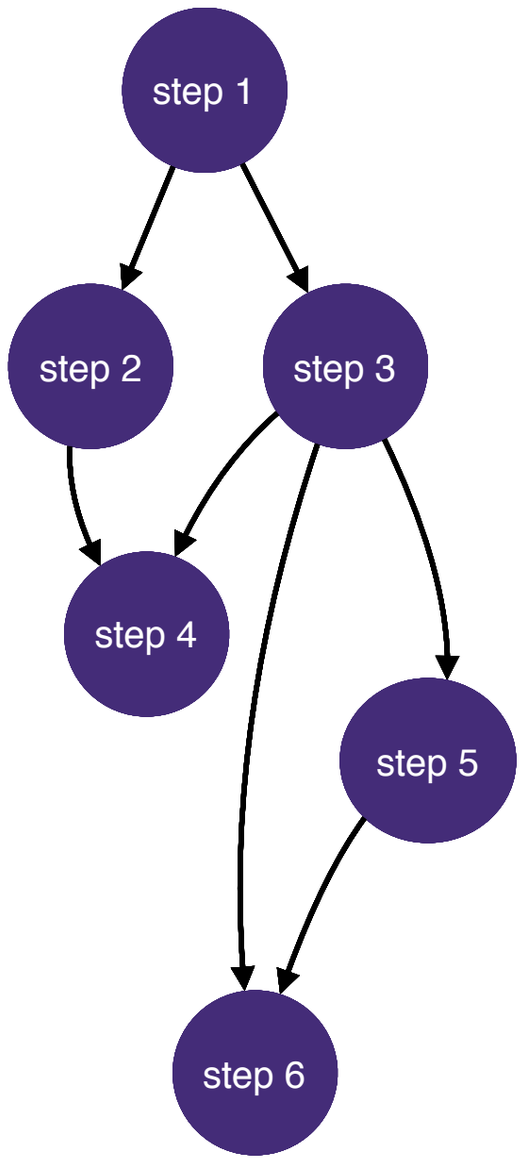
Par. Set 2

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
- uses JSON references for composability
- run-time package “yadage” developed as part of diana-hep

```
yadage-run workdir workflow.json -p nevents=100
```



Example:

```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'}
```

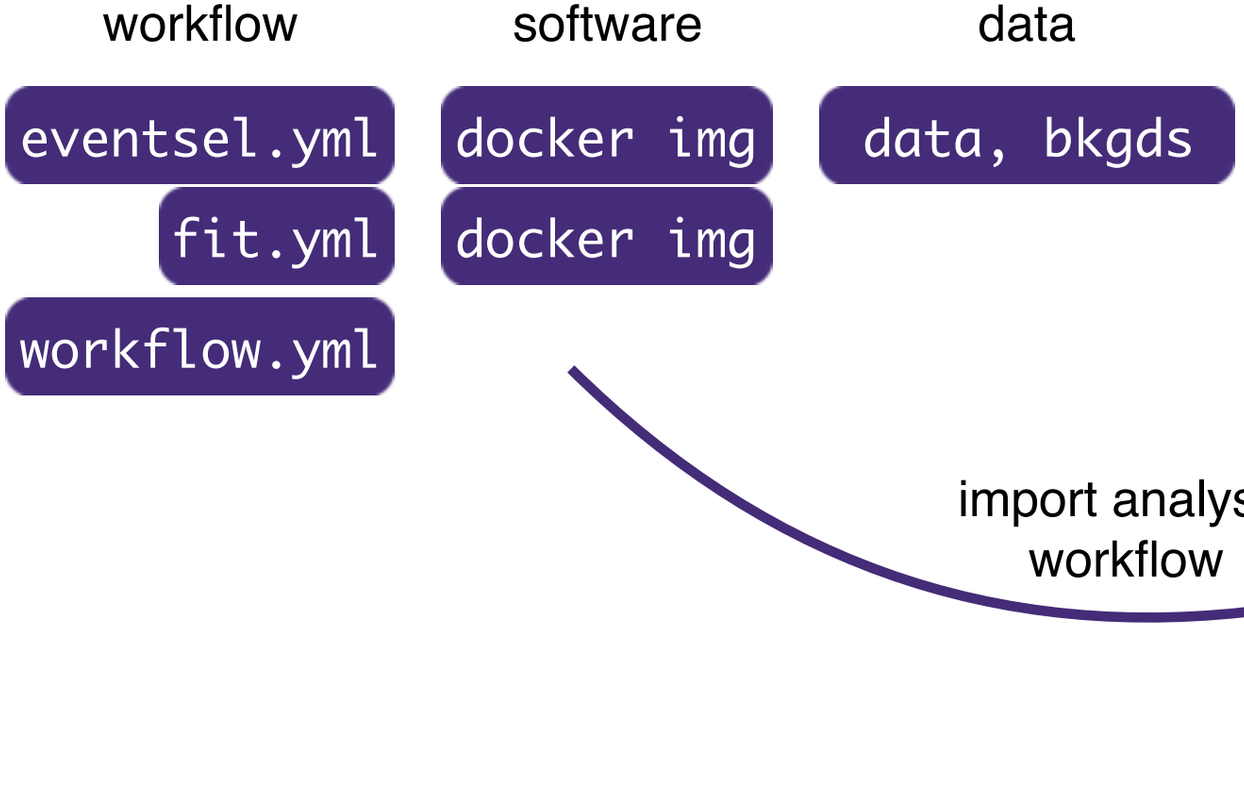
Stage 1:
unknown number of files. e.g. download & unpack archive with a priori unknown # of files

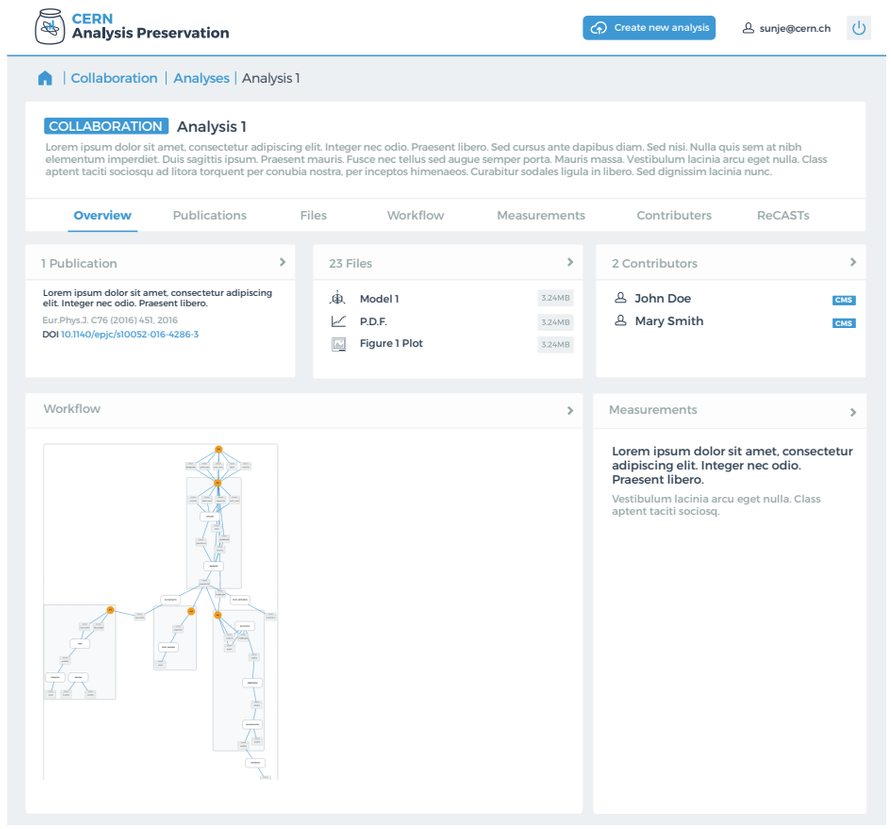
Stage 2:
for each file in the archive, add node to process it

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

Recap:

- workflow modeled as dynamic DAG. Sequentially build during run-time.
- each node modeled as a “packtivity”, parametrized container workload
- both modeled via JSON/YML schemas
- integrated into CERN Analysis Preservation





Running Workflows on CERN Container Infrastructure

NEW YORK UNIVERSITY

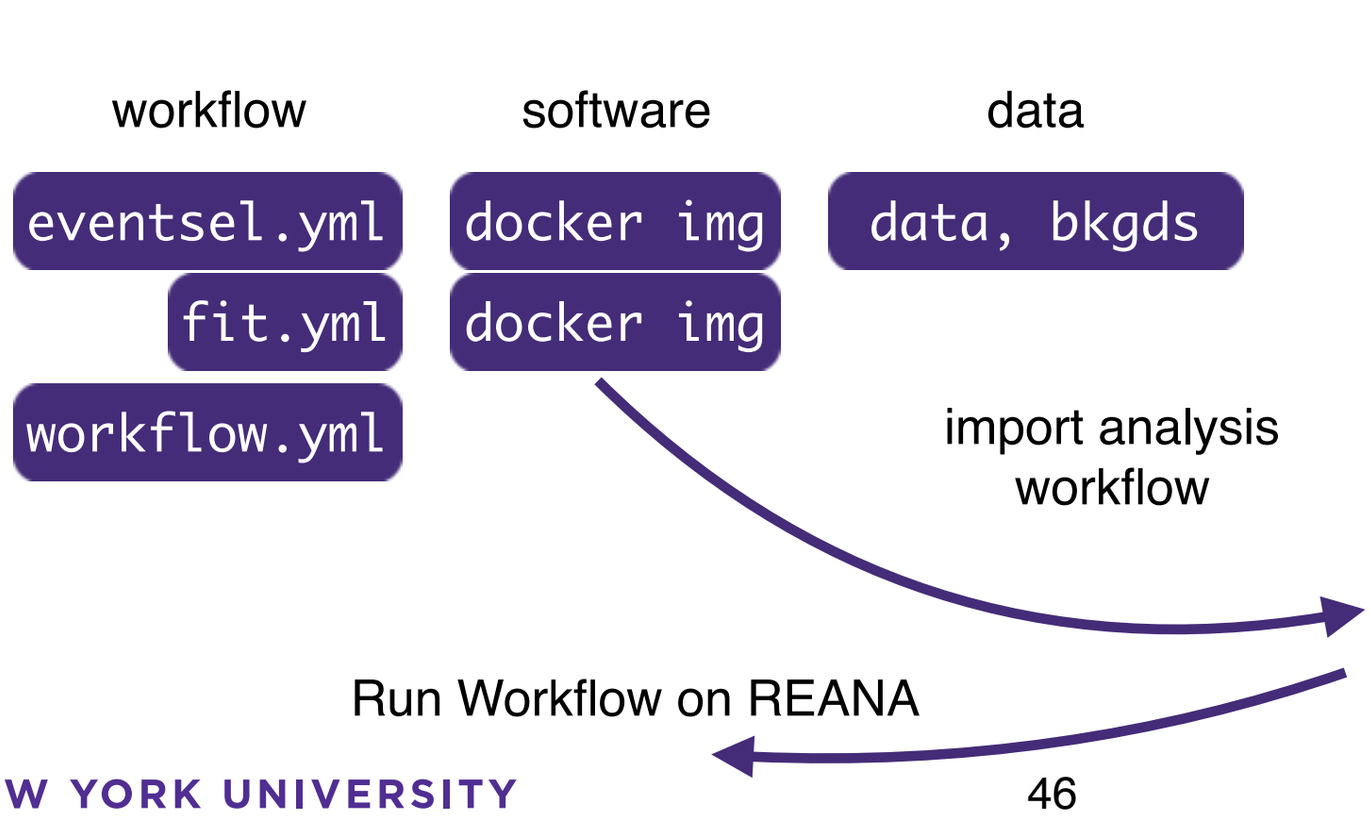
We have workflows in CAP, now we want to execute them!

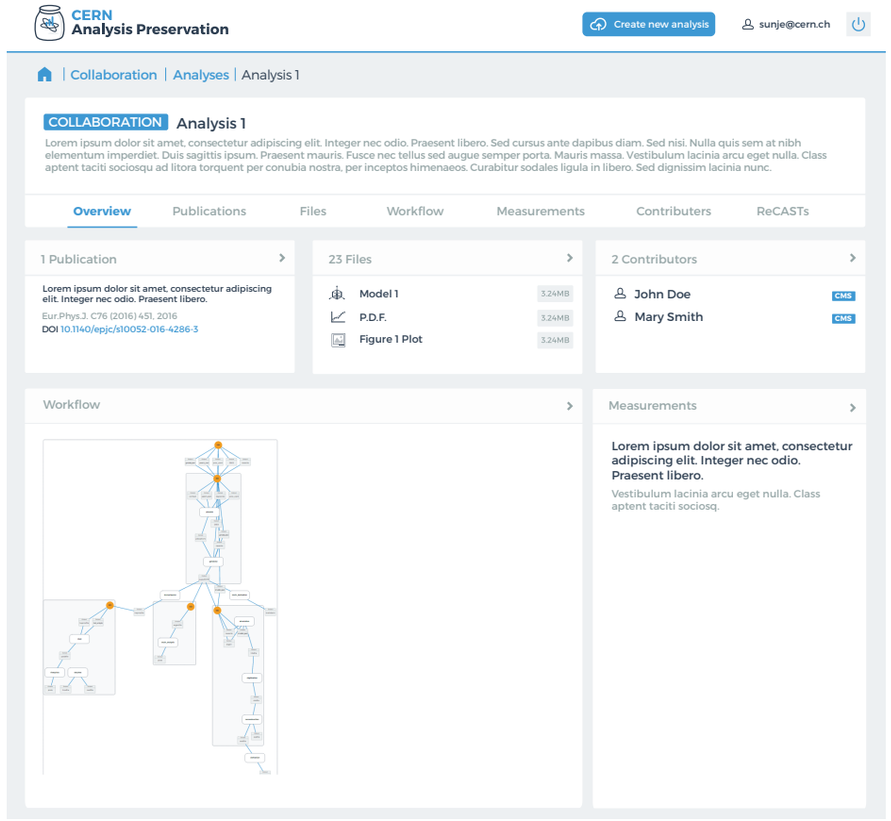
Requirements:

- Scalable (run things in parallel if possible)
- Runs on generic compute (let's not use the Grid)
- Read workflows from CAP

Joint project of RECAST and CAP
REANA: re-usable analyses.

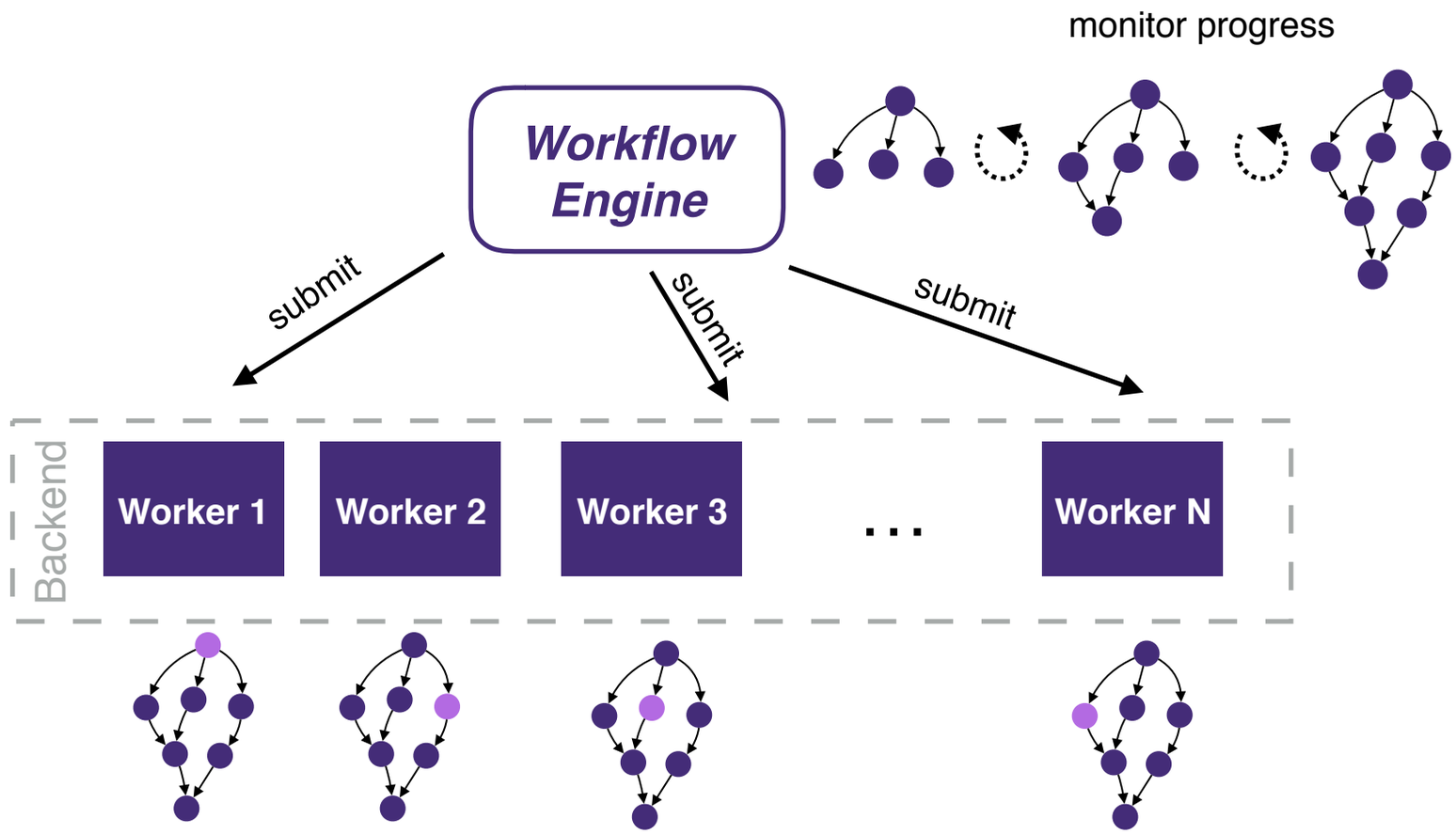
Goal: A cloud-based system to re-use scientific data-processing pipelines





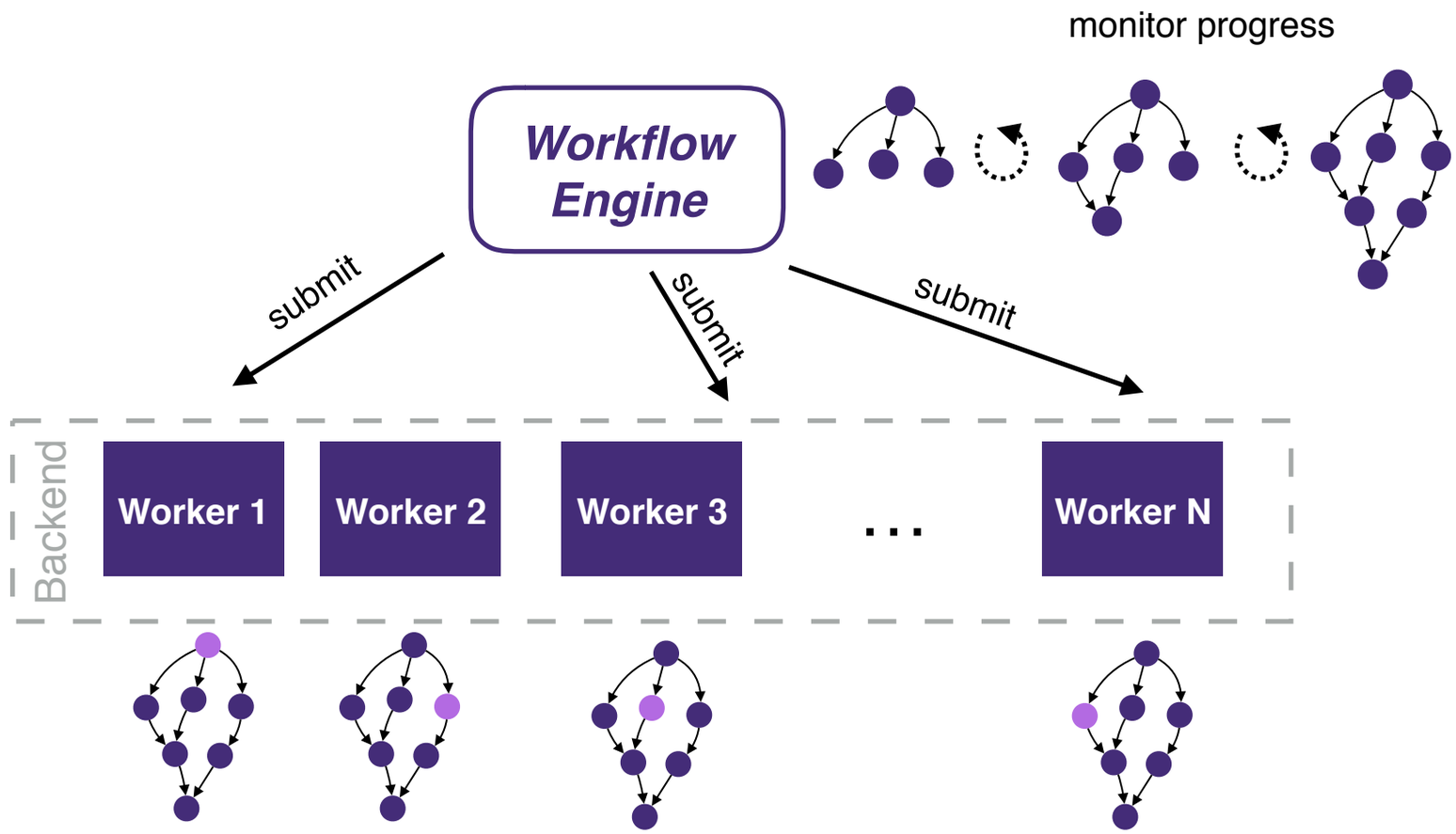
REANA rough sketch

- main unit of work is a containerized workload
- “workflow engines” control what workloads to submit



REANA rough sketch

- i.e. what we need to a async scheduling of container workloads connected to distributed storage



REANA implementation

- i.e. what we need to a async scheduling of container workloads connected to distributed storage
- **Immediate thought**: Kubernetes Jobs + CephFS

REANA implementation

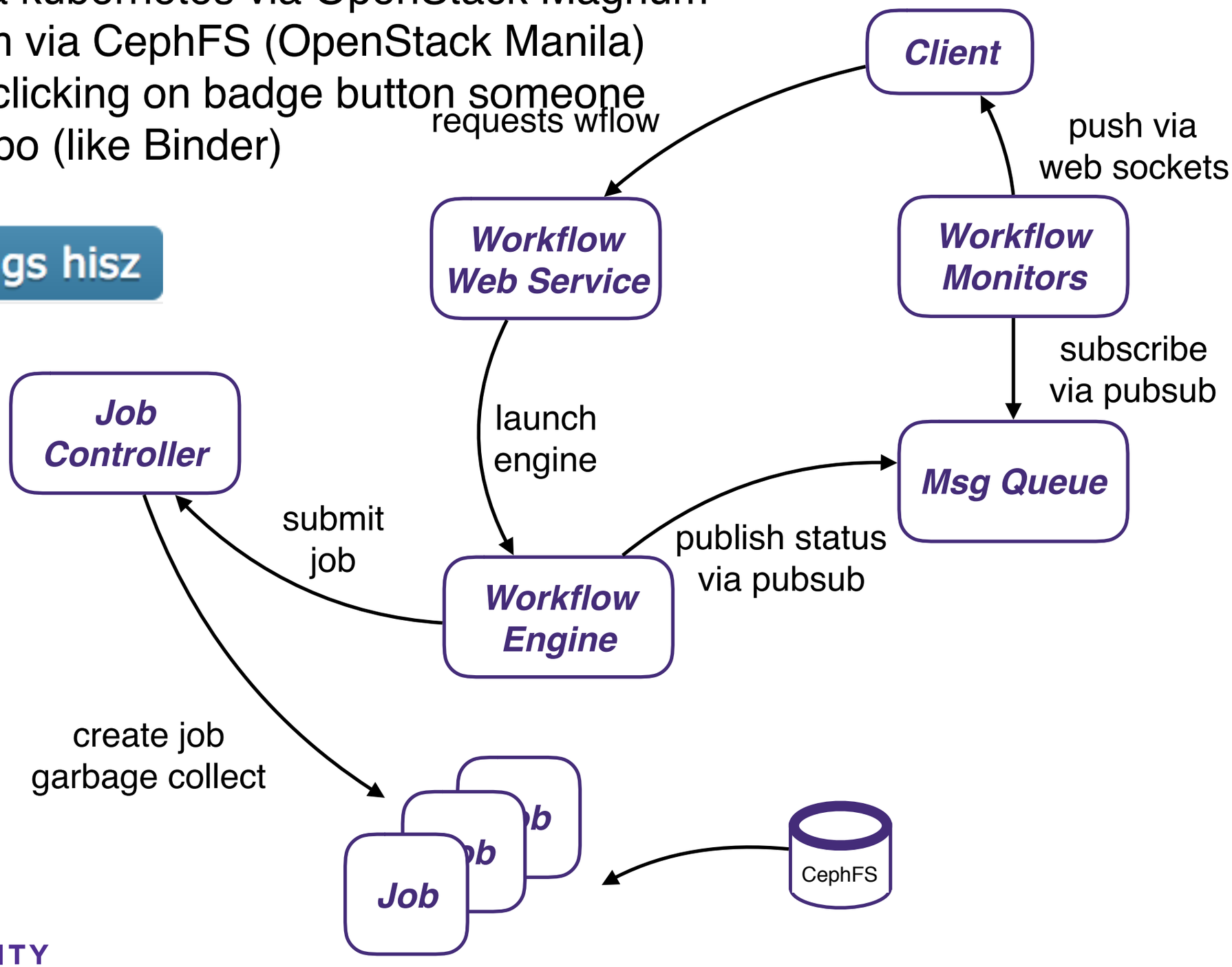
- i.e. what we need to a async scheduling of container workloads connected to distributed storage
- **Immediate thought**: Kubernetes Jobs + CephFS

Not so fast:

- Kubernetes Jobs do not behave like ordinary async batch-like jobs (a la LSF/ Condor/Celery). On failure, re-spawn in a endless loop until manually removed
- Need thin layer on top of Kubernetes: web-service with REST API to launch k8s jobs, and garbage collect them. A **job controller.**
  - long-term: make upstream contribution to k8s (kubernetes/issues/30243)

REANA infrastructure

- fully deployed via kubernetes via OpenStack Magnum
- shared filesystem via CephFS (OpenStack Manila)
- run workflow by clicking on badge button someone added to their repo (like Binder)



RECAST Infrastructure Overview:

With archived analysis workflows and scalable workflow execution, it becomes feasible to streamline the reinterpretation efforts. Reinterpretations as a community-wide service.
Originally suggested by Cranmer, Yavin [arXiv:1010.2506]

Idea:

- Produce reinterpretations of same fidelity as original result (not just approximations)
- Allow hep-ph community to *suggest* reinterpretations through a standard (web) interface. They provide most interesting points / scans to do. Auxiliary information such as run cards, SLHA spectra, UFO models
- LHC collaborations review suggestions and choose which to fulfill (based on scale of request, availability of a preserved analysis, physics case)
- Use archived analysis to (semi-) automatically run reinterpretation. Review results, approve (possibly on accelerated track, since analysis already approved).
- Publish and/or append original analysis HEPDATA record.
- Allows us to decouple original publication from reinterpretations. Publish early using benchmark signals, continuously re-interpret as samples become available

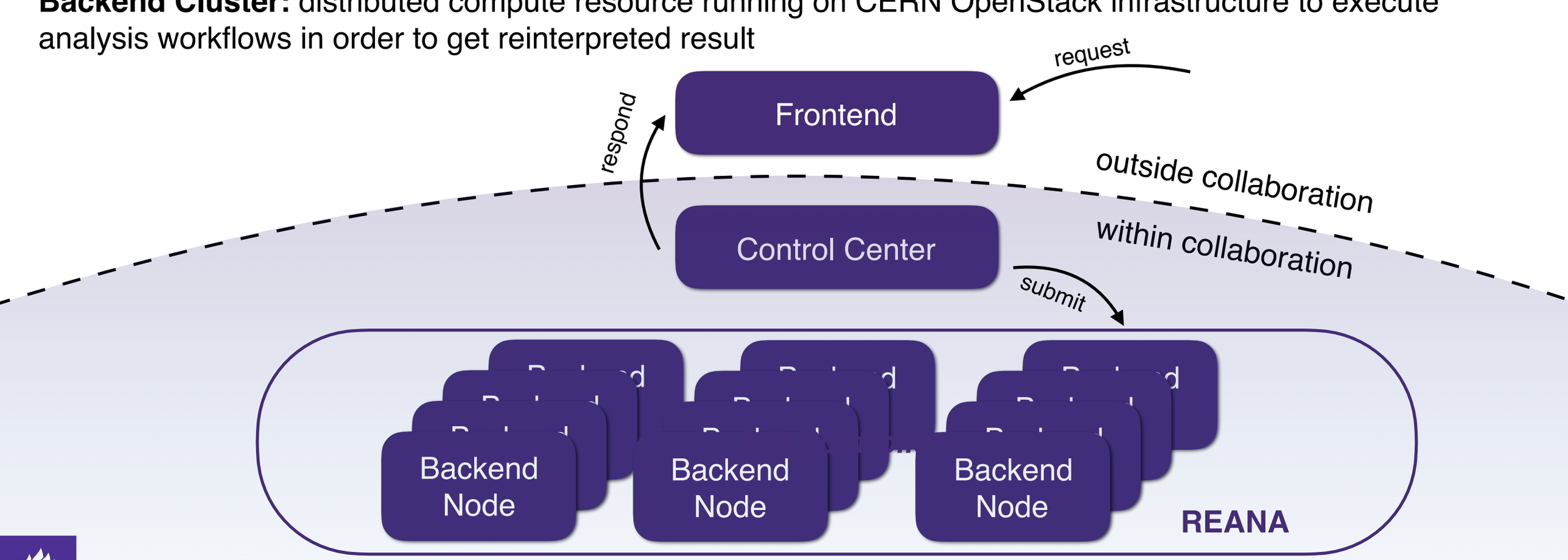
RECAST Infrastructure Overview:

With archived analysis workflows it becomes feasible to streamline reinterpretations-as-a-service.

Frontend: public-facing web-service (+ API). Let's e.g. theorists register interest in reinterpretations for specific published analyses. Allows them to provide auxiliary data needed for analysis
Note: No guarantee of fulfillment of request by collaborations.

Control Center: collaboration-internal web-service (+ API) to inspect incoming requests, compare against catalogue of archived analysis, allows submission to backend to actually perform reinterpretation. Can push "RECAST response" back to frontend.

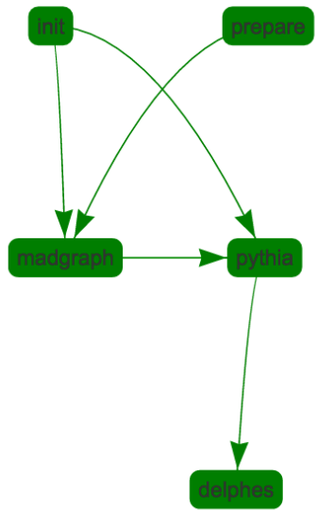
Backend Cluster: distributed compute resource running on CERN OpenStack infrastructure to execute analysis workflows in order to get reinterpreted result



recast

Backend Status | Implemented Analyses | All Requests | dummyuser | Logout

Workflow Visualization



Last seen: 2017-04-05 19:30:29

Log

Messages from the request processor will appear below.

```
2017-04-05 17:15:31  workflow registered. processed by celery id: d1385b94-e6e1-4b41-baba-14d60d87817f
2017-04-05 19:15:31  INFO - running analysis on worker: worker-369599623-pwgxb hello
2017-04-05 19:15:31  INFO - setting up for context {u'shipout_spec': {u'user': u'root', u'host': u'recast-backend-shiptarget.d…
2017-04-05 19:15:31  INFO - prepared workdir workdirs/7a6e655f-e567-498c-9a52-20899cd0d787
2017-04-05 19:15:31  WARNING - No input archive specified, skipping download
2017-04-05 19:15:31  INFO - setting up entry point recastyadage.backendtasks:recast
2017-04-05 19:15:32  INFO - and off we go with job 7a6e655f-e567-498c-9a52-20899cd0d787!
2017-04-05 19:15:32  INFO - running yadage workflow for context: {u'shipout_spec': {u'user': u'root', u'host': u'recast-backen
```

Containers in ATLAS have use-cases across a broad spectrum of computing activities

Most drastic effect in feasibility of analysis preservation and re-use. Makes it technologically feasible to archive and re-execute analysis pipelines that are captured as containers once, re-run many times.

Developed YAML/JSON spec to easily define pipeline logic with run-time dynamism and parametrization: yadage/packtivity. DAG of container workloads

Leverage latest CERN IT cloud offerings (OpenStack Magnum + Manila, Kubernetes, CephFS) to prototype re-interpretation service RECAST at scale

Joined forces with CERN Analysis Preservation to develop generic cloud-based distributed workflow service REANA.

Most of this stuff would not have been possible without the amazing support and resource I've been given from CERN IT

Thanks for indulging me and my nagging questions :)

NEW YORK UNIVERSITY

Questions?

Appendix

Analysis as a function mapping data and models to results

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

observable distributions, confidence intervals on model parameters

reconstruction, event selection, stat. evaluation

collision data from LHC detector

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

archive analysis in a **parametrized form**, such that we can quickly run a new model

$$f_{\text{analysis}}(\text{data}|\ \cdot\ )$$

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) ↓

$f_{\text{a}}(\text{data}\|\text{model}_1)$	$f_{\text{a}}(\text{data}\|\text{model}_2)$
$f_{\text{b}}(\text{data}\|\text{model}_1)$	$f_{\text{b}}(\text{data}\|\text{model}_2)$
$f_{\text{c}}(\text{data}\|\text{model}_1)$	$f_{\text{c}}(\text{data}\|\text{model}_2)$

Case Study: Multi B-jets analysis

Defining the individual Workflow steps

- need script that tell us how to run the code once we are in the right environment. parametrized by a few variables (input file names etc)
- can use simple shell script, but also anything else

lumi/xsec/KF/FE weighting of HF tree

23 lines (22 sloc) | 714 Bytes — Raw | Blame | History

```
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 --doH
    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
```

direct SH Driver reads signal dataset (a SUSY10 derivation) via XrootD writes out HistFitter tree

49 lines (42 sloc) | 1.42 KB — Raw | Blame | History

```
process:
  process_type: 'interpolated-script-cmd'
  script: |
    #!/bin/bash
    source ~/.bashrc
    setupATLAS
    lsetup "root 6.06.02-x86_64-slc6-gcc48-opt"
    cd /code/multib/HistFitter
    source ./setup.sh
    cd analysis/analysis_multib


    /recast_auth/getkrb.sh
    #klist
    #exit
    cd input
    python mergeTrees.py {selectionoutput} --filters filters/filters_ht.json --weights {weightsfile} --did-to-group {groupingfi
    cd ..

    lumi="5807.51"
    grid="{gridname}"
    region="Gbb_A"
    tag="tag2.4.11-1-0_July00"
    echo '["{pointname}"]' > point.json
    cat point.json
    export HF_MBJ_SIGNALJSON="point.json"
    export HF_MBJ_BACKGROUNDFILE={bkgtree}
    export HF_MBJ_DATAFILE={datatree}
    export HF_MBJ_SIGNALFILE='input/Sig.root'
    HistFitter.py -wtpf -F excl python/My3bGtt.py _signalRegion $region _lumi $lumi _unblind true _doHFSplitting false 2>&1 | t

    resultfile=$(ls results/My3bGtt_*fixSigXSecNominal*_hypotest.root)
    echo "result file is:  $resultfile"
    root -b -q 'root2json.C("'"$resultfile"'","hypo_Gbb_%f_%f")'

    jsonfile=$(ls *harvest_list.json)
    python recast_format.py $jsonfile {outputjson}
```

Run HF

Extract Results into JSON format

Case Study: Multi B-jets analysis

Stringing the workflow together

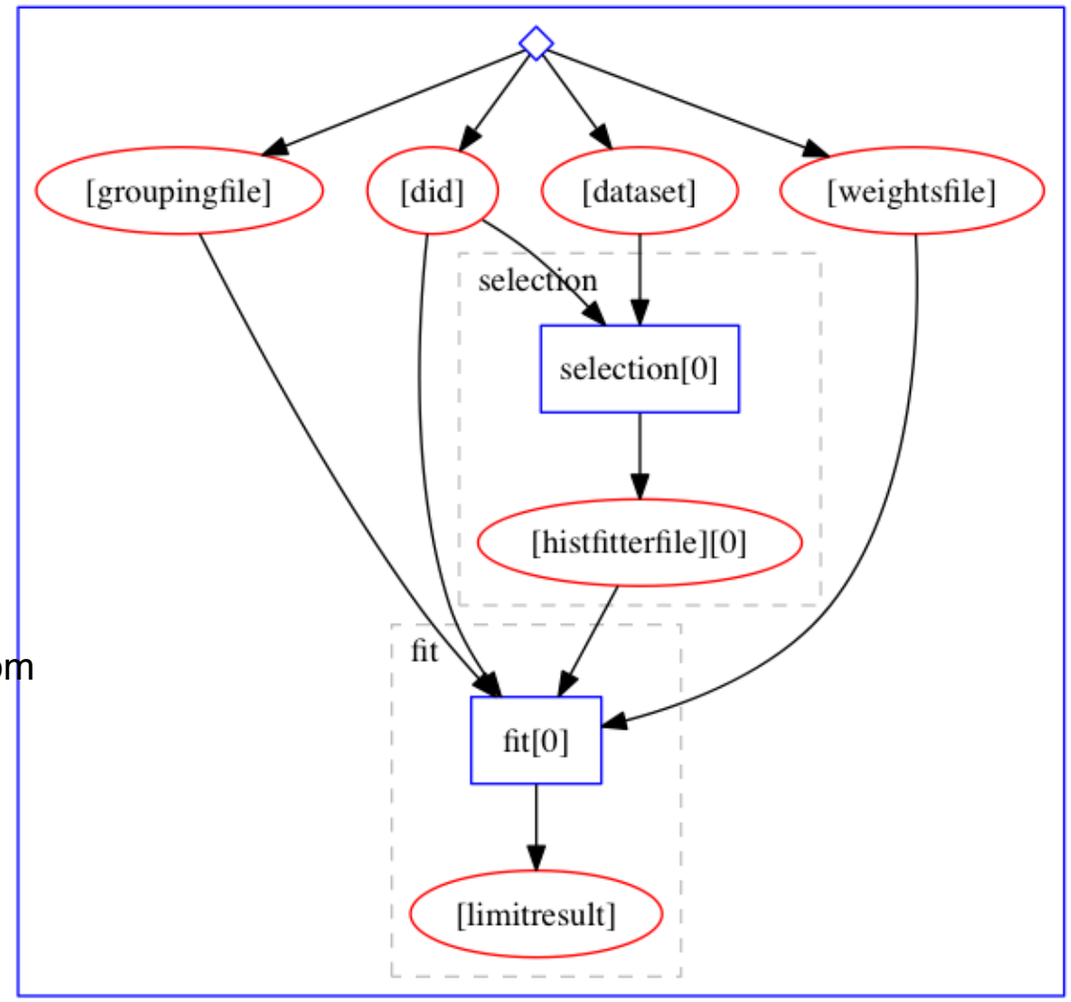
- small file on how the individual pieces fit together.
- Here: dataset, AMI info file etc provided as input parameters, define EOS location of signal and background trees, declare that signal histfitter tree comes from previous selection step etc

27 lines (26 sloc) | 1.07 KB — Raw | Blame | History

```yaml
stages:
  - name: selection
    dependencies: ['init']
    scheduler:
      scheduler_type: singlestep-stage
      parameters:
        dataset: {stages: init, output: dataset, unwrap: true}
        submitdir: '{workdir}/submitdir'
        outputprefix: '{workdir}/histfitter.root'
        did: {stages: init, output: did, unwrap: true}
      step: {$ref: 'selscript.yml#'}
  - name: fit
    dependencies: ['selection']
    scheduler:
      scheduler_type: singlestep-stage
      parameters:
        bkgtree: 'root://eosuser.cern.ch///eos/project/r/recast/Bkg_2.4.15-2-0_merged.root'
        datatree: 'root://eosuser.cern.ch///eos/project/r/recast/Data_2.4.15-2-0.root'
        outputjson: '{workdir}/fitoutput.json'
        pointname: 'Gbb_1600_200'
        gridname: Gbb
        selectionoutput: {stages: selection, output: histfitterfile, unwrap: true}
        weightsfile: {stages: init, output: weightsfile, unwrap: true}
        groupingfile: {stages: init, output: groupingfile, unwrap: true}
        did: {stages: init, output: did, unwrap: true}
      step: {$ref: 'fitscript.yml#'}
```

data and background trees archived in access-controlled location

take signal HF tree from previous step



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

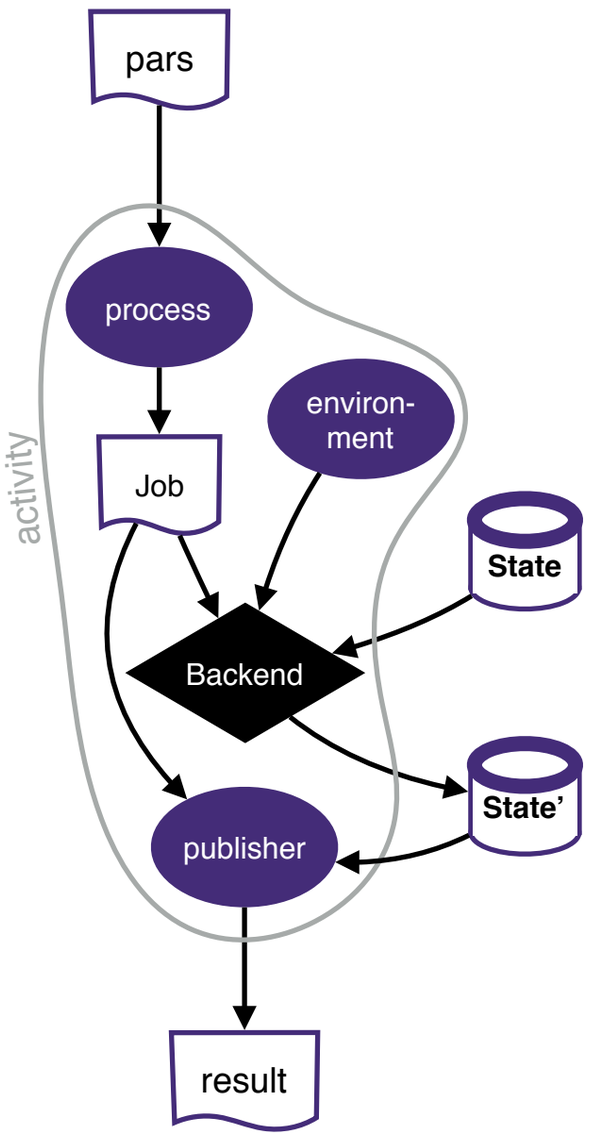
1. Problem: Preserve Individual Processing Steps

(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

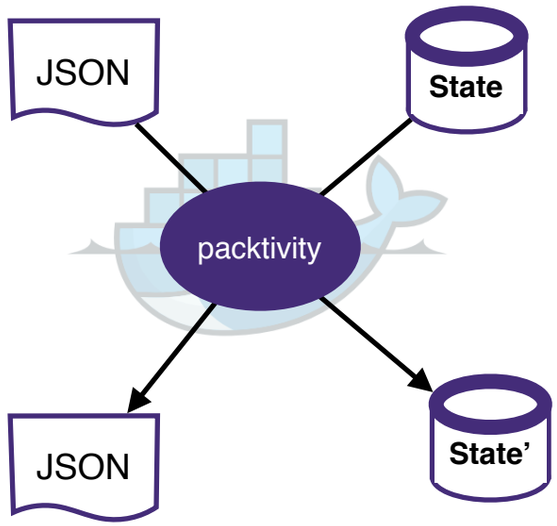
- as interchange format for parameters and result data
- as declarative description format for *process/env/publisher*
  - incl. JSON schemas for validation

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

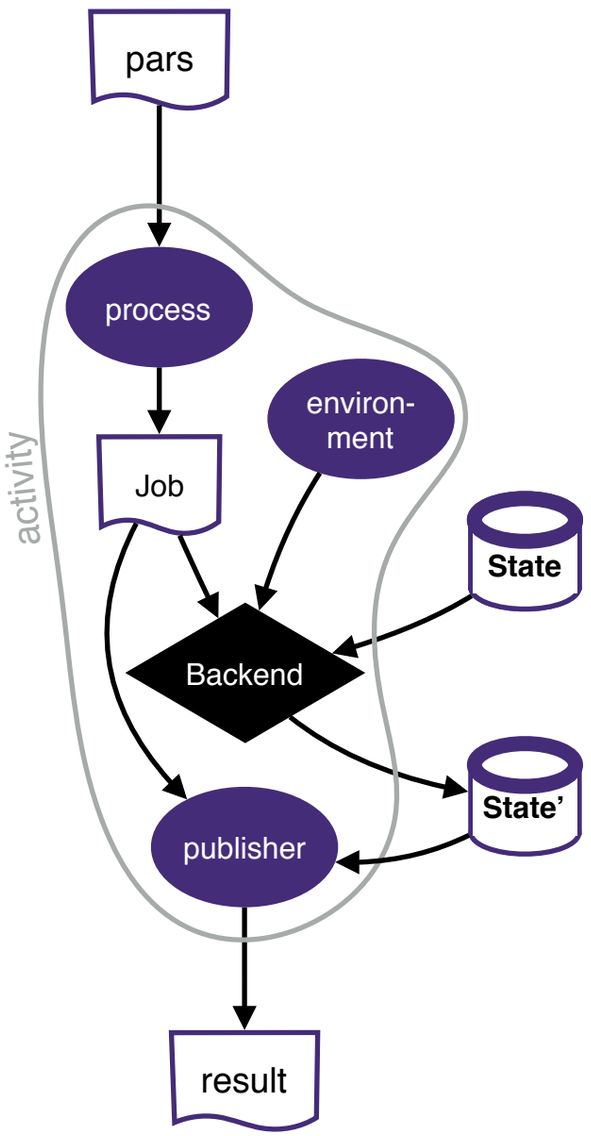
- JSON API
- archivable, declarative description as JSON
- dependencies captured in environment
  - e.g. Docker Image

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

=



=



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

1. Problem: Preserve Individual Processing Steps

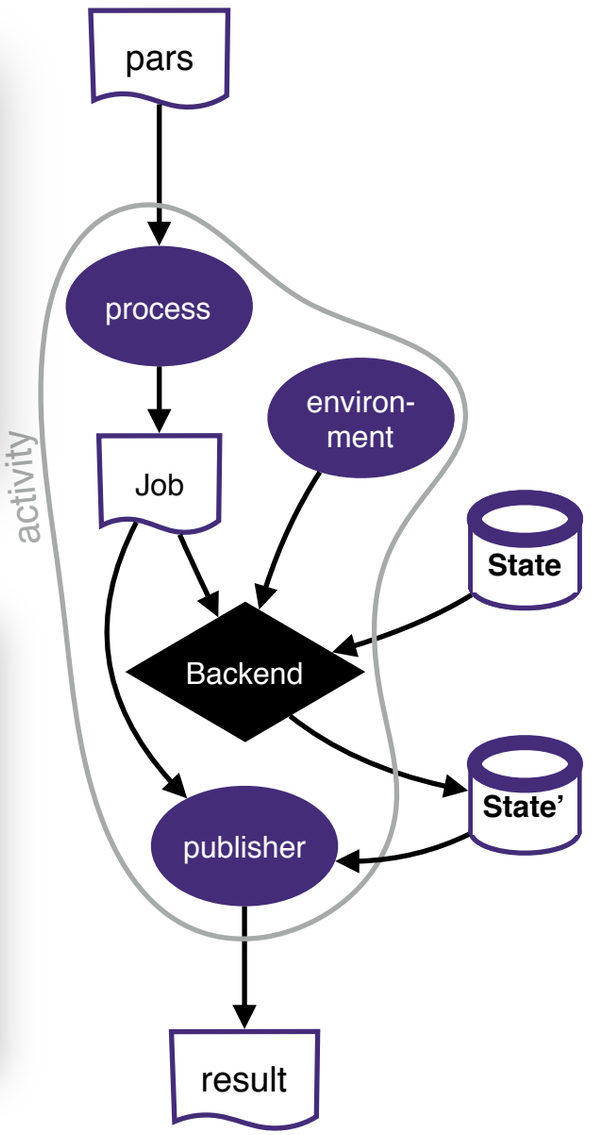
(Example: Run Detector Simulation + Reconstruction on MC events)

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 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

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
```