Reproducible computational workflows with REANA

Tibor Šimko

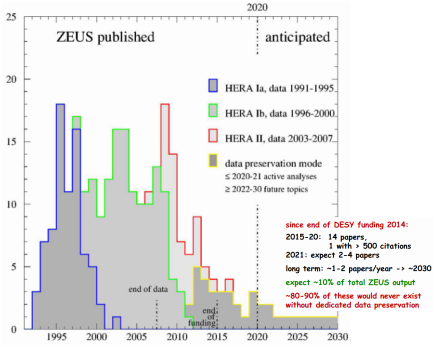
Department of Information Technology
CERN

EURO-LABS Advanced Training: Open Science and Data Management
Ebernburg, Germany, November 24th–29th 2024

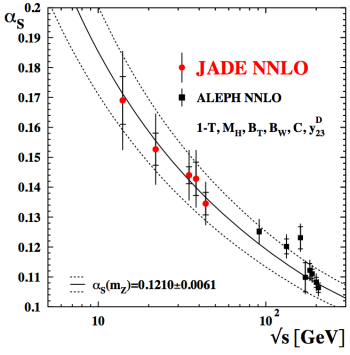
https://indico.gsi.de/event/19808

Computational reproducibility

Long-term value of data!



Achim Geiser `https://indico.cern.ch/event/1009487`



DPHEP `https://arxiv.org/abs/1205.4667`

Collaborations publish papers even fifteen years after data taking ends.

JADE data (1979–1986) still unique even forty years later.

Long-term value of knowledge?

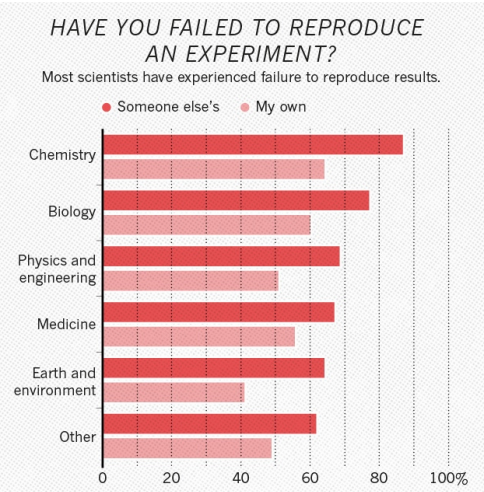
CMS collaboration

First LHCb paper `arXiv.1008.3105`

Experimental physics done by large groups of thousands of physicists.

High turnover of young researchers. Half of LHCb authors remain after ten years.

Half of researchers cannot reproduce their own results



https://www.nature.com/news/1-500-scientists-lift-the-lid-on-reproducibility-1.19970

Reproducibility? Reusability? Repeatability? Replicability?

The Turing Way model

Analysis	Data: Same	Data: Different
Same	Reproducible	Replicable
Different	Robust	Generalisable

https://the-turing-way.netlify.app/reproducible-research/overview/overview-definitions.html

The PRIMAD model

Label	Data: Parameters	Data: Raw Data	Platform / Stack	Implementation	Method	Research Objective	Actor	Gain
Repeat	-	-	-	-	-	-		Determinism
Param. Sweep	x	-	-	-	-	-		Robustness / Sensitivity
Generalize	(x)	x	-	-	-	-		Applicability across different settings
Port	-	-	x	-	-	-		Portability across platforms, flexibility
Re-code	-	-	(x)	x	-	-		Correctness of implementation, flexibility, adoption, efficiency
Validate	(x)	(x)	(x)	(x)	x	-		Correctness of hypothesis, validation via different approach
Re-use	-	-	-	-	-	x		Apply code in different settings, Re-purpose
Independent x *(orthogonal)*							x	Sufficiency of information, independent verification

Figure 1 PRIMAD Model: Categorizing the various types of reproducibility by varying the (P)latform, (R)esearch Objective, (I)mplementation, (M)ethod, (A)ctor and (D)ata, analyzing the gain they bring to computational experiments. x denotes the variable primed i.e. changed, (x) a variable that may need to be changed as a consequence, whereas – denotes no change.

https://drops.dagstuhl.de/opus/volltexte/2016/5817/pdf/dagrep_v006_i001_p108_s16041.pdf

From “reproducible” to “reusable” analyses

Good practices are long known, but the uptake is slow

G. K. Sandve, A. Nekrutenko, J. Taylor, E. Hovig: *"Ten Simple Rules for Reproducible Computational Research"* (2013) https://doi.org/10.1371/journal.pcbi.1003285

1. For every result, keep track of how it was produced
2. Avoid manual data manipulation steps
3. Archive the exact versions of all external programs used
4. Version control all custom scripts
5. Record all intermediate results, when possible in standardized formats
6. For analyses that include randomness, note underlying random seeds
7. Always store raw data behind plots
8. Generate hierarchical analysis output, allowing layers of increasing detail to be inspected
9. Connect textual statements to underlying results
10. Provide public access to scripts, runs, and results

Challenges are both sociological and technological

Survey of 1008 researchers from a leading machine-learning conference (NIPS):

Table 11: Most Influential Reasons Not to Share Data, by Non-sharer and Sharer

	Closed	Open	p-value for diff
The time it takes to document for release	57.95%	52.38%	0.6818
The possibility that your dataset may be used without citation	50.00%	28.57%	**0.0342**
Legal barriers, such as copyright	42.37%	40.00%	1.0000
The potential loss of future publications using these data	39.33%	30.95%	0.4629
Dealing with questions from users about the data	38.64%	26.19%	0.2310
The time it takes to verify privacy or other admin data concerns	38.20%	41.46%	0.8724
Competitors may get an advantage	37.08%	30.95%	0.6245
The web doesn't allow me to track others use of the data	30.68%	14.28%	**0.0729**
Technical limitations, ie. webspace platform space constraints	29.54%	26.19%	0.8504
Whether there is intense competition in the topic	29.55%	16.67%	0.1731
Whether you put in a large amount of work building the dataset	24.72%	26.19%	1.0000
Availability of other data that might substitute for your own	12.36%	19.05%	0.4540
	<10%	<10%	

Table 12: Most Influential Reasons Not to Share Code, by Non-Sharer and Sharer

	Closed	Open	p-value for diff
The time it takes to clean up and document for release	82.22%	71.43%	0.2363
Dealing with questions from users about the code	54.44%	47.62%	0.5863
The possibility that your code may be used without citation	47.19%	37.71%	0.2964
The possibility of patents, or other IP constraints	38.89%	40.48%	1.0000
Competitors may get an advantage	34.44%	23.81%	0.3040
The potential loss of future publications using this code	31.11%	28.57%	0.9264
The code might be used in commercial applications	28.89%	23.81%	0.6888
Legal barriers, such as copyright	28.81%	44.00%	0.2727
The web doesn't allow me to track others use of the code	26.67%	14.29%	0.1745
Technical limitations, ie. webspace platform space constraints	23.33%	14.29%	0.3327
Availability of other code that might substitute for your own	22.22%	17.07%	0.6580
Whether you put in a large amount of work building the code	22.22%	14.29%	0.4049
Whether there is intense competition in the topic	15.56%	21.43%	0.5604
	<10%	<10%	

V. Stodden, *"The Scientific Method in Practice: Reproducibility in the Computational Sciences"* (2010) `http://dx.doi.org/10.2139/ssrn.1550193`

What's in it for me?

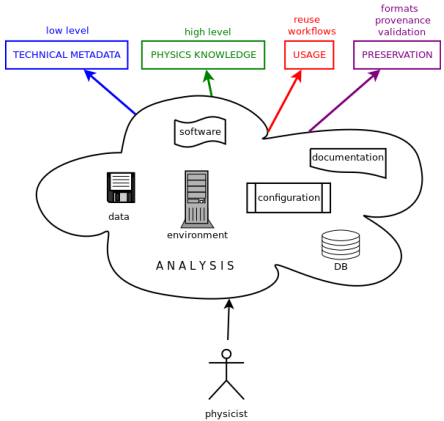
"Your closest collaborator is you six months ago but you don't reply to email."

– Karl Broman, *"Tools for Reproducible Research"*

`https://kbroman.org/Tools4RR/`

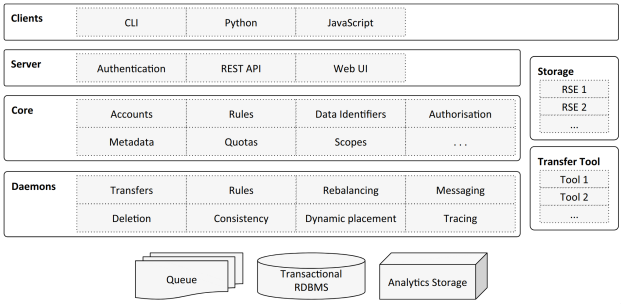
The elements of analysis preservation

Preserving analysis knowledge



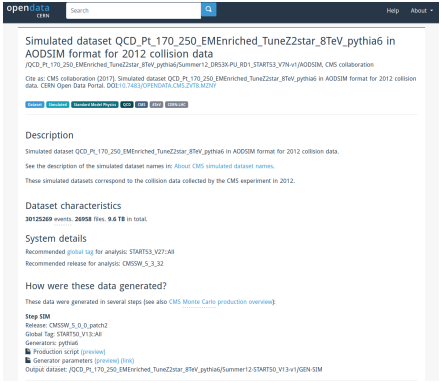
Capturing structured analysis knowledge in “actionable” formats

I. Data: Preserving (parts of) experimental data



https://doi.org/10.1007/s41781-019-0026-3

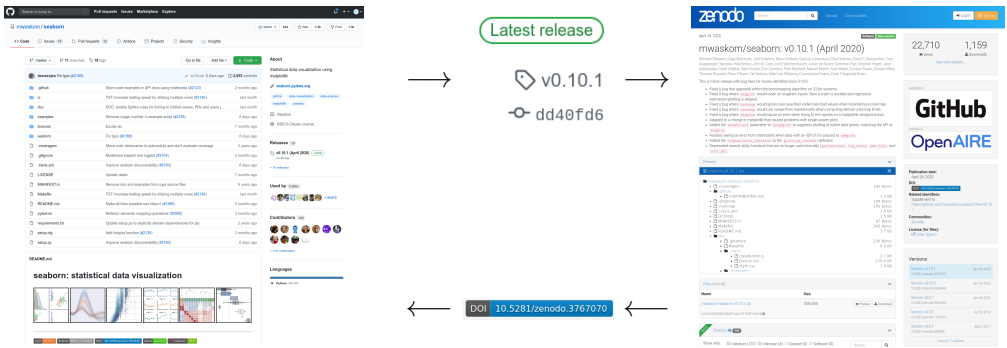
Rucio



CERN Open Data

Data live in scientific management systems; can be preserved in digital repositories

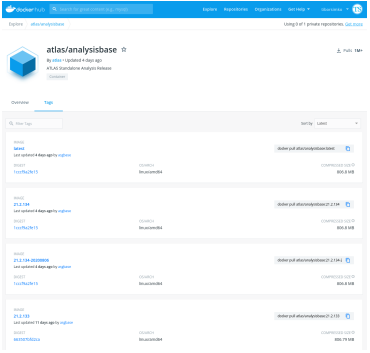
II. Code: Preserving research software



https://guides.github.com/activities/citable-code

GitHub ↔ Zenodo bridge to automatically preserve software releases

III. Computing environment: Containers



ATLAS collaboration

`https://hub.docker.com/r/atlas/analysisbase/tags`

CMS collaboration

`https://gitlab.cern.ch/cms-cloud/cmssw-docker`

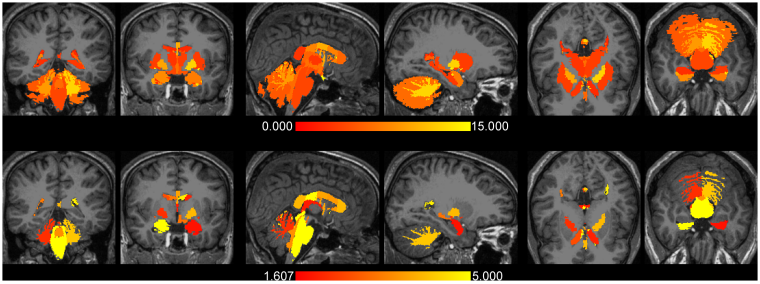
Container technology helps to encapsulate the computing environment

III. Computing environment: An example from life sciences

The Effects of FreeSurfer Version, Workstation Type, and Macintosh Operating System Version on Anatomical Volume and Cortical Thickness Measurements

Ed H. B. M. Gronenschild, Petra Habets, Heidi I. L. Jacobs, Ron Mengelers, Nico Rozendaal, Jim van Os, Machteld Marcelis

Published: June 1, 2012 • DOI: 10.1371/journal.pone.0038234



Software changes (Freesurfer 4.3.1, 4.5.0, 5.0.0): 8.8±6.6% (volume) and 2.8±1.3% (thickness)

Operating system changes (macOS 10.5, 10.6): "about factor two smaller"

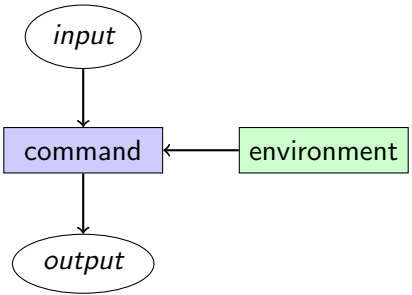
III. Computing environment: Beyond containers

```
❯ ls -l /cvmfs/cms-opendata-conddb.cern.ch/
total 1655262
drwxr-xr-x. 2 cvmfs cvmfs         24 Jan 21  2016 FT_53_LV5_AN1
drwxr-xr-x. 2 cvmfs cvmfs         24 Feb 22  2016 FT_53_LV5_AN1_RUNA
drwxr-xr-x. 2 cvmfs cvmfs        366 Jun 21  2017 FT53_V21A_AN6
drwxr-xr-x. 2 cvmfs cvmfs        365 Nov 29  2017 FT53_V21A_AN6_FULL
drwxr-xr-x. 2 cvmfs cvmfs        365 Jun 23  2017 FT53_V21A_AN6_RUNC
drwxr-xr-x. 2 cvmfs cvmfs          3 Oct 20  2017 FT_R_42_V10A
drwxr-xr-x. 2 cvmfs cvmfs        248 Nov  9  2018 START42_V17B
drwxr-xr-x. 2 cvmfs cvmfs        282 Jan 21  2016 START53_LV6A1
drwxr-xr-x. 2 cvmfs cvmfs        394 Jun 21  2017 START53_V27
drwxr-xr-x. 2 cvmfs cvmfs        296 Nov 30  2018 START53_V7N
-rw-r--r--. 1 cvmfs cvmfs 1002414080 Oct 31  2018 102X_upgrade2018_design_v9.db
-rw-r--r--. 1 cvmfs cvmfs  691593216 Oct 31  2018 80X_mcRun2_asymptotic_2016_TrancheIV_v8.db
-rw-r--r--. 1 cvmfs cvmfs      82944 Jan 21  2016 FT_53_LV5_AN1.db
-rw-r--r--. 1 cvmfs cvmfs      82944 Feb 22  2016 FT_53_LV5_AN1_RUNA.db
-rw-r--r--. 1 cvmfs cvmfs     119808 Jun 21  2017 FT53_V21A_AN6.db
-rw-r--r--. 1 cvmfs cvmfs     120832 Nov 29  2017 FT53_V21A_AN6_FULL.db
-rw-r--r--. 1 cvmfs cvmfs     120832 Jun 23  2017 FT53_V21A_AN6_RUNC.db
-rw-r--r--. 1 cvmfs cvmfs      64512 Oct 20  2017 FT_R_42_V10A.db
-rw-r--r--. 1 cvmfs cvmfs      72704 Nov  9  2018 START42_V17B.db
-rw-r--r--. 1 cvmfs cvmfs      84992 Jan 21  2016 START53_LV6A1.db
-rw-r--r--. 1 cvmfs cvmfs     130048 Jun 21  2017 START53_V27.db
-rw-r--r--. 1 cvmfs cvmfs      89088 Nov 30  2018 START53_V7N.db
```

Condition database snapshots for CMS open data on CVMFS

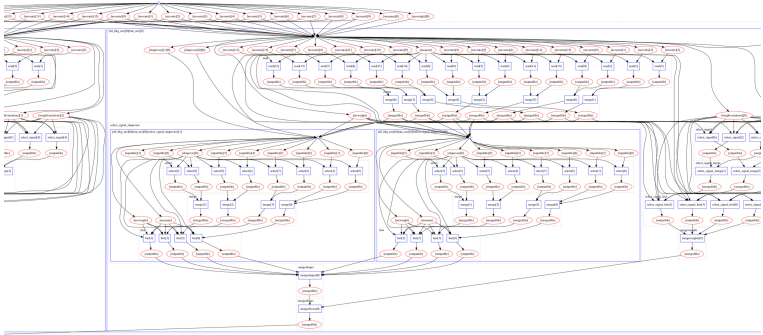
Computing environments may interact with other runtime services; these may need "encapsulation" as well in order to allow future reuse

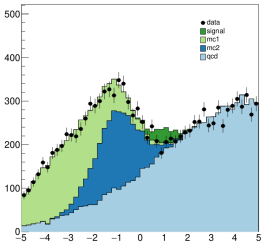
IV. Computational recipes: One step



A recipe on how to arrive from the input data to the desired output

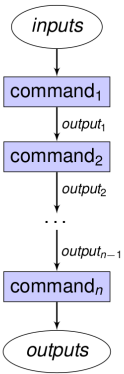
IV. Computational recipes: Many steps (Directed Acyclic Graphs)

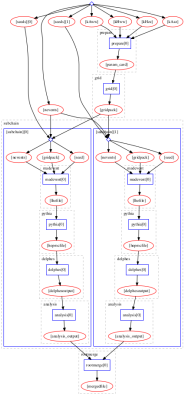


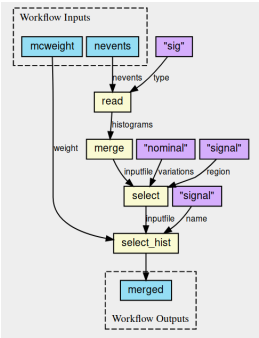


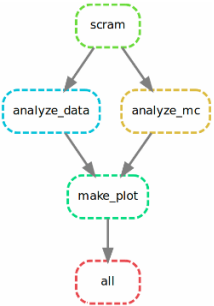
Realistic physics analysis workflows may consist of O(1k) computational steps

IV. Computational recipes: A variety of computational workflow languages



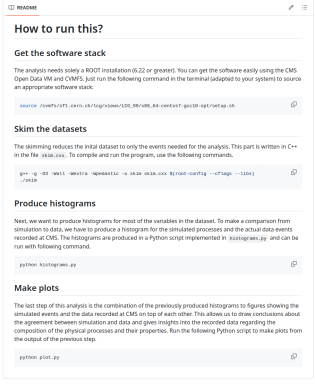






Serial Yadage CWL Snakemake

IV. Computational recipes: Make it actionable



README

How to run this?

Get the software stack

The analysis needs solely a ROOT installation (6.22 or greater). You can get the software easily using the CMS Open Data VM and CVMFS. Just run the following command in the terminal (adapted to your system) to source an appropriate software stack:

```
source /cvmfs/sft.cern.ch/lcg/views/LCG_98rc4b6_64-centos7-gcc10-opt/setup.sh
```

Skim the datasets

The skimming reduces the inital dataset to only the events needed for the analysis. This part is written in C++ in the file `skim.cxx`. To compile and run the program, use the following commands.

```
g++ -g -O3 `wait -Wextra `wpedantic -o skim skim.cxx $(root-config --cflags --libs)
./skim
```

Produce histograms

Next, we want to produce histograms for most of the variables in the dataset. To make a comparison from simulation to data, we have to produce a histogram for the simulated processes and the actual data events recorded at CMS. The histograms are produced in a Python script implemented in `histograms.py` and can be run with following command.

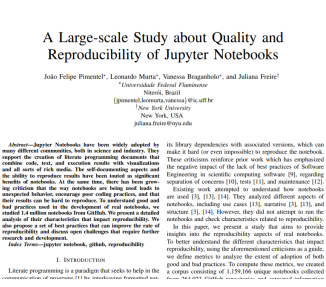
```
python histograms.py
```

Make plots

The last step of this analysis is the combination of the previously produced histograms to figures showing the simulated events and the data recorded at CMS on top of each other. This allows us to draw conclusions about the agreement between simulation and data and gives insights into the recorded data regarding the composition of the physical processes and their properties. Run the following Python script to make plots from the output of the previous step.

```
python plot.py
```

How-to-run recipes in README files are a good start; but they are not actionable



A Large-scale Study about Quality and Reproducibility of Jupyter Notebooks

João Felipe Pimentel*, Leonardo Murta*, Vanessa Braganholo*, and Juliana Freire†
*Universidade Federal Fluminense
Niterói, Brasil
{jpimentel,leomurta,vanessa}@ic.uff.br
†New York University
New York, USA
juliana.freire@nyu.edu

Abstract—**Jupyter Notebooks have been widely adopted by many different communities, both in science and industry. They support the creation of literate programming documents that combine code, text, and execution results with visualizations and all sorts of rich media. The self-documenting aspects and the ease of sharing notebooks have fostered the reproducibility benefits of notebooks. At the same time, there has been growing criticism that the way notebooks are being used leads to unexpected behavior, encourage poor coding practices, and that their results can be hard to reproduce. To understand good and bad practices used in the development of real notebooks, we studied 1.4 million notebooks from GitHub. We present a detailed analysis of their characteristics that impact reproducibility. We also propose a set of best practices that can improve the rate of reproducibility and discuss open challenges that require further research and development.**

***Index Terms*—jupyter notebook, github, reproducibility**

I. Introduction

Literate programming is a paradigm that seeks to help in the

its library dependencies with associated versions, which can make it hard (or even impossible) to reproduce the notebook. These criticisms reinforce prior work which has emphasized the negative impact of the lack of best practices of Software Engineering in scientific computing software [9], regarding separation of concerns [10], tests [11], and maintenance [12].

Existing work attempted to understand how notebooks are used [3], [13], [14]. They analyzed different aspects of notebooks, including use cases [13], narrative [3], [13], and structure [3], [14]. However, they did not attempt to run the notebooks and check characteristics related to reproducibility.

In this paper, we present a study that aims to provide insights into the reproducibility aspects of real notebooks. To better understand the different characteristics that impact reproducibility, using the aforementioned criticisms as a guide, we define metrics to analyze the extent of adoption of both good and bad practices. To compute these metrics, we created a corpus consisting of 1,159,166 unique notebooks collected

https://leomurta.github.io/papers/pimentel2019a.pdf

"Out of 863,878 attempted executions of valid notebooks (...) only 24.11% executed without errors and only 4.03% produced the same results"

“Notebooks” and “workflows”: a march of history

“Notebooks”

- Started as interactive Python IDE
- Been adding kernels (Julia, R)
- Been adding explicit parallel DAG processing (ipyparallel)
- Been adding implicit parallel DAG processing (HTCondor, Spark, Torch)

“Workflows”

- Started as batch tools
- Been standardising “random” glue scripting practices
- Been orchestrating thousands of batch jobs (HPC, HTC, AWS...)
- Been adding IDEs (Arvados, Rabix)

IDE tools adding batch support $\longrightarrow$

happy users

$\longleftarrow$ Batch tools adding IDE support

Summary: Four pillars of reusable computational research

I. Input data

What is your input data?

- input files
- input parameters

II. Analysis code

Which code analyses it?

- user code
- software frameworks

III. Computing environment

What is your environment?

- operating system
- database calls

IV. Computational recipes

Which steps did you take?

- shell commands
- notebooks and workflows

REANA

Reusable Analyses

Reproducible research data analysis platform

Flexible

Run many computational workflow engines.

Scalable

Support for remote compute clouds.

Reusable

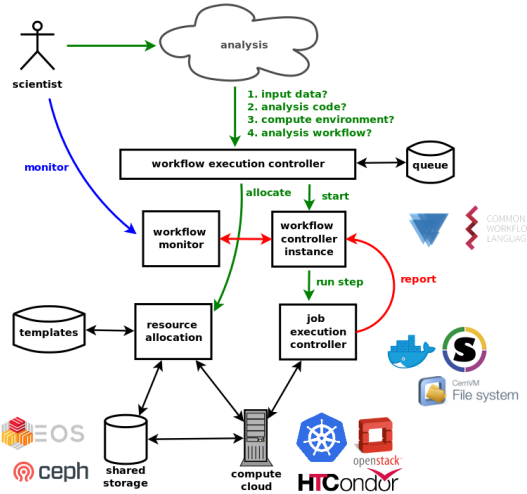
Containerise once, reuse elsewhere. Cloud-native.

Free

Free Software. MIT licence. Made with ❤ at CERN.

`https://www.reana.io/`

REANA architecture



Respecting diverse habits of diverse research groups

- multiple workflow systems (CWL, Serial, Snakemake, Yadage)
- multiple container technologies (Docker, Singularity)
- multiple compute backends (Kubernetes, HTCondor, Slurm)
- multiple shared storage platforms (Ceph, CVMFS, EOS, NFS)

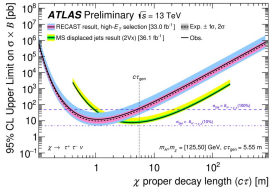
REANA command-line and web interface

```yaml
version: 0.6.0
inputs:
  files:
    - code/gendata.C
    - code/fitdata.C
  parameters:
    events: 20000
    data: results/data.root
    plot: results/plot.png
workflow:
  type: serial
  specification:
    steps:
      - name: gendata
        environment: 'reanahub/reana-env-root6:6.18.04'
        commands:
        - mkdir -p results && root -b -q 'code/gendata.C(${events},"${data}")'
      - name: fitdata
        environment: 'reanahub/reana-env-root6:6.18.04'
        commands:
        - root -b -q 'code/fitdata.C("${data}","${plot}")'
outputs:
  files:
    - results/plot.png
```

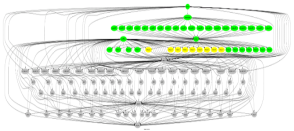
Structure data analysis by means of declarative workflows

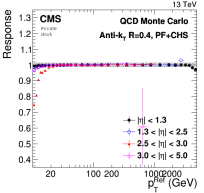
Use command-line and web interfaces to run analysis on remote compute clusters

Data analysis and data production examples



ATLAS `https://cds.cern.ch/record/2714064`





CMS `https://github.com/alintulu/reana-demo-JetMETAnalysis`

Data analysis example: ATLAS displaced jet search reinterpretation

Data production example: CMS jet energy resolution and corrections

Example: ATLAS searches for new physics

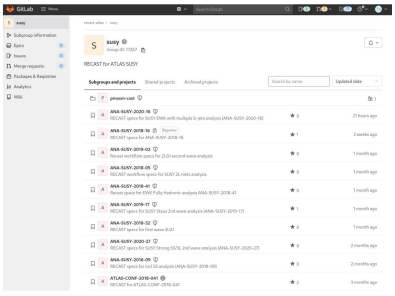


Figure 1. A screenshot of the ATLAS SUSY group analyses preserved on GitLab. Each repository is labeled with the internal ATLAS analysis identifier and contains both workflow files and additional data files needed for the computational processing.

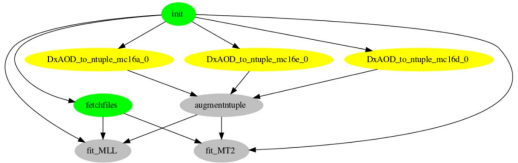


Figure 2. A typical pMSSM workflow. The computational runtime is about 10 minutes without systematics (test payload) and about 10 hours with all systematics (real payload).

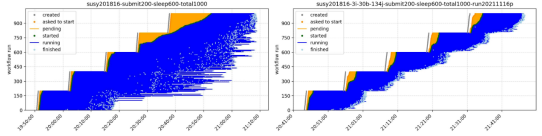


Figure 8. A scalability test submitting 200 workflows every 10 minutes. A cluster with 448 cores (left) cannot keep up with the load. A cluster with 1072 cores (right) can comfortably hold the incoming workload.

https://arxiv.org/abs/2403.03494

Imperative vs declarative programming

Separating “what” from “how”

Scenario: Please give me names of all teenagers from a list of people.

- **imperative** programming: specifying “how” exactly to arrive at results (C)

```
for (int i = 0; i < sizeof(people) / sizeof(struct people); i++) {
    if (people[i].age < 20) {
        printf("%s\n", people[i].name)
    }
}
```

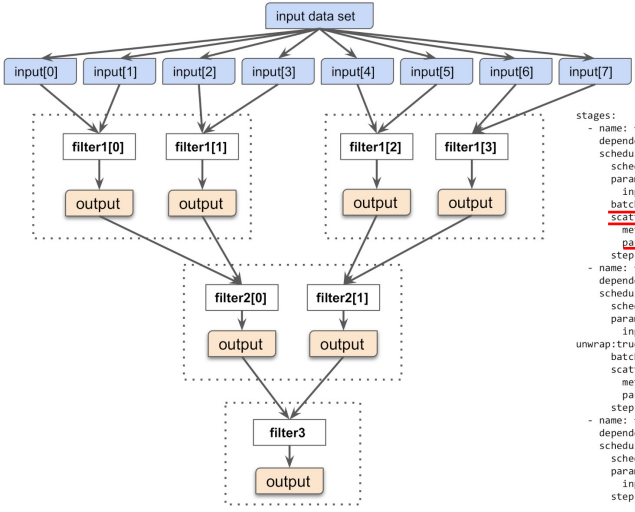
- **declarative** programming: specifying “what” is desired (SQL)

```
SELECT name FROM people WHERE age<20
```

- so far so good; but what if the data structure does not fit in memory?

Useful for separating “problem-domain knowledge” from “operational boilerplate”

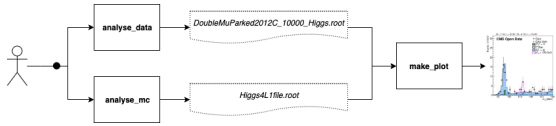
Example: multi-cascading scatter-gather paradigm

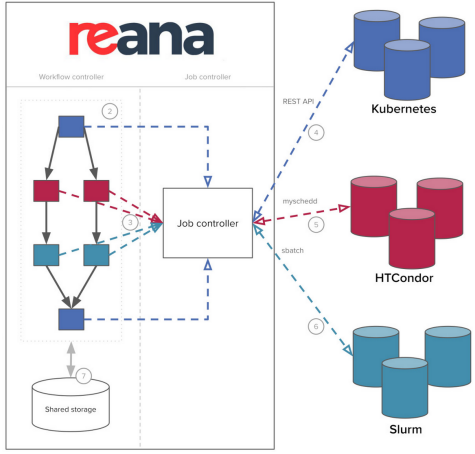



```
stages:
  - name: filter1
    dependencies: [init]
    scheduler:
      scheduler_type: multistep-stage
      parameters:
        input: {stages: init, output: input, unwrap: true}
      batchsize: 2
      scatter:
        method: zip
        parameters: [input]
      step: {$ref: steps.yaml#/filter}
  - name: filter2
    dependencies: [filter1]
    scheduler:
      scheduler_type: multistep-stage
      parameters:
        input: {stages: filter1, output: output,
unwrap:true}
      batchsize: 2
      scatter:
        method: zip
        parameters: [input]
      step: {$ref: steps.yaml#/filter}
  - name: filter3
    dependencies: [filter2]
    scheduler:
      scheduler_type: singlestep-stage
      parameters:
        input: {stages: 'filter2', output: output}
      step: {$ref: steps.yaml#/filter}
```

Example: job dispatch

```
steps:
  analyse_data:
    run: analyse_data.cwl
    hints:
      reana:
        compute_backend: slurmcern
    out: [DoubleMuParked2012C_10000_Higgs.root]
  analyse_mc:
    run: analyse_mc.cwl
    hints:
      reana:
        compute_backend: htcondorcern
    out: [Higgs4L1file.root]
  make_plot:
    run: make_plot.cwl
    hints:
      reana:
        compute_backend: kubernetes
    in:
      DoubleMuParked2012C_10000_Higgs: >
        analyse_data/DoubleMuParked2012C_10000_Higgs.root
      Higgs4L1file: >
        analyse_mc/Higgs4L1file.root
    out: [mass4l_combine_userlvl3.pdf]
```

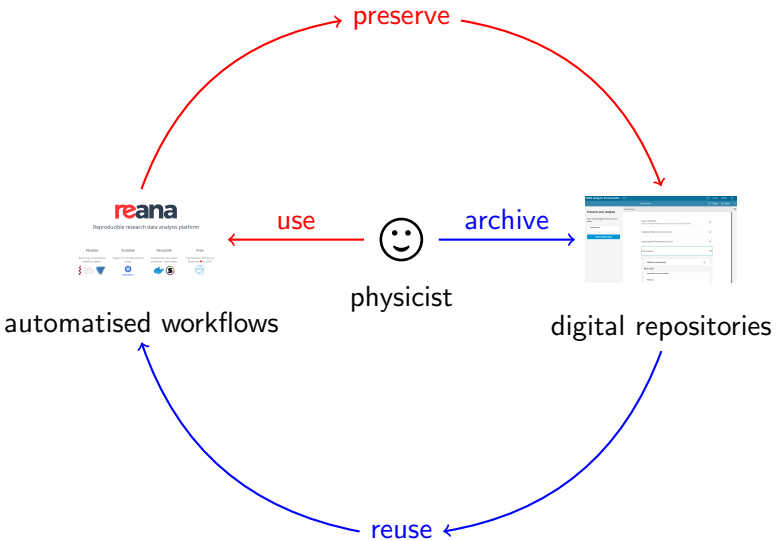




Custom workflow hints for hybrid dispatch

Reproducibility vs preproducibility

Reproducibility ⇌ Preservation



“Preproducible” analyses

Nature **557** (2018) 613

WORLD VIEW · 24 MAY 2018

Before reproducibility must come preproducibility

Instead of arguing about whether results hold up, let’s push to provide enough information for others to repeat the experiments, says Philip Stark.

https://doi.org/10.1038/d41586-018-05256-0

Driving preproducibility via Continuous Integration with source code management systems

Exercise 1: Using REANA

Exercise 1: Modelling event-data distributions (1 of 4)

Scenario

We shall generate data and fit it against a model using RooFit library. This scenario allows to familiarise yourself with the basic reproducible workflow practices with REANA.

Follow along

1. In your browser, open `https://reana-p4n.aip.de` and sign in.
2. In your browser, request your access token.
3. In your terminal, install `reana-client`. (See next page.)
4. Follow along with the presenter to run your first containerised analysis example!

Exercise 1: Modelling event-data distributions (2 of 4)

Installing reana-client

First option, use Python virtual enviroment:

```
$ virtualenv ~/.virtualenvs/reana
$ source ~/.virtualenvs/reana/bin/activate
$ pip install reana-client
```

Second option, use standalone executable (on GNU/Linux systems):

```
$ wget https://github.com/reanahub/reana-client/releases/download/0.9.3/
    reana-client-0.9.3-x86_64.AppImage
$ chmod u+x ./reana-client-0.9.3-x86_64.AppImage
$ mv ./reana-client-0.9.3-x86_64.AppImage ./reana-client
```

Exercise 1: Modelling event-data distributions (3 of 4)

Clone the demo example

```
$ git clone --depth 1 -b eurolabs-training-2024 \
    https://github.com/reanahub/reana-demo-root6-roofit
$ cd reana-demo-root6-roofit
$ rm -rf .git
```

Listen to explanations

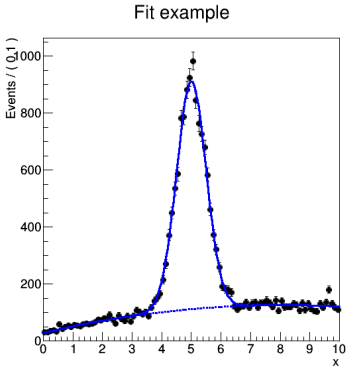
Stop here and follow along with the presenter to understand and run the analysis using `reana-client`.

Exercise 1: Modelling event-data distributions (4 of 4)

Problem: Consider that we are not satisfied with the produced plot and that we would like to change the title from "Fit example" to "My fit example".

Task 1: Change the analysis source code and rerun the analysis to produce the new desired plot.

Task 2: Have you run only those workflow steps that are really necessary to produce the new plot? If not, how can you avoid rerunning the (possibly computationally expensive) data generation steps?



Exercise 2: Writing workflows from scratch

Exercise 2: Scatter-gather analysis (1 of 10)

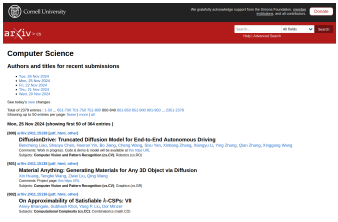
Scenario

We shall write a simple workflow from scratch that will analyse nine arXiv papers. We want to simply know the total word count of these papers.

Note

This example will demonstrate basic parallelisation techniques in declarative workflow languages and put the concept of "map-reduce" or "scatter-gather" paradigm in practice.

You would similarly process physics datasets by parallel computations, "scattering" the analysis process (such as production of histograms) across samples, before "gathering" the calculations (such as adding histograms) back.

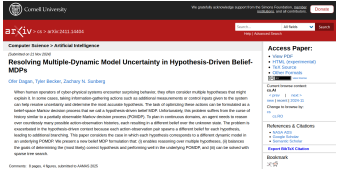


Exercise 2: Scatter-gather analysis (2 of 10)

I. Input data?

The only necessary input parameters are paper arXiv IDs.

```
papers = ["arXiv:2411.14404", "arXiv:2411.14368",
          "arXiv:2411.14214", "arXiv:2411.14033",
          "arXiv:2411.14012", "arXiv:2411.13932",
          "arXiv:2411.13867", "arXiv:2411.13749",
          "arXiv:2411.13653"]
```



Exercise 2: Scatter-gather analysis (3 of 10)

II. Analysis code?

There is no need to write code this this example.

We shall use the regular Unix tools:

- `curl` to download papers in PDF
- `podftotext` to convert papers to plain text
- `wc` to count words
- `bc` to sum up counts

Exercise 2: Scatter-gather analysis (4 of 10)

III. Computing environment?

Let us build a `Dockerfile` to create a container image with all the tools we need:

```
$ cat Dockerfile
FROM docker.io/library/ubuntu:24.04
RUN apt-get update -y && \
    apt-get install --no-install-recommends -y \
      bc=1.07.1-3ubuntu4 \
      curl=8.5.0-2ubuntu10.5 \
      poppler-utils=24.02.0-1ubuntu9.1 && \
    apt-get autoremove -y && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

$ docker build -t docker.io/tiborsimko/wordcount:1.0 .
```

Note: (i) versioning of all dependencies; (ii) keeping Docker image layers slim

Exercise 2: Scatter-gather analysis (5 of 10)

IV. Workflow? Download papers

A step where the PDF files are downloaded from arXiv based on their IDs.

```
rule download:
    output:
        "papers/{paper}.pdf"
    container:
        "docker://docker.io/tiborsimko/wordcount:1.0"
    shell:
        "mkdir -p papers && cd papers && \
         curl -ko {wildcards.paper}.pdf
             https://arxiv.org/pdf/{wildcards.paper}"
```

Note: usage of "wildcards" to make Snakemake rules general

Exercise 2: Scatter-gather analysis (6 of 10)

IV. Workflow? Convert papers

A step where the PDF files are converted to plain text.

```
rule convert:
    input:
        "papers/{paper}.pdf"
    output:
        "texts/{paper}.txt"
    container:
        "docker://docker.io/tiborsimko/wordcount:1.0"
    shell:
        "mkdir -p texts && pdftotext papers/{wildcards.paper}.pdf \
                                     texts/{wildcards.paper}.txt"
```

Note: uses now-familiar combination of “wildcards” again

Exercise 2: Scatter-gather analysis (7 of 10)

IV. Workflow? Count words

A step where the words are counted in the plain text files.

```
rule count:
    input:
        "texts/{paper}.txt"
    output:
        "counts/{paper}.txt"
    container:
        "docker://docker.io/tiborsimko/wordcount:1.0"
    shell:
        "mkdir -p counts && wc -w {input} | cut -d' ' -f1 > {output}"
```

Note: the "map" part of map-reduce; the "scatter" part of scatter-gather

Exercise 2: Scatter-gather analysis (8 of 10)

IV. Workflow? Sum counts

A step where the counted words are summed up across files.

```
rule sum:
    input:
        expand("counts/{paper}.txt", paper=papers)
    output:
        "wordcount.txt"
    container:
        "docker://docker.io/tiborsimko/wordcount:1.0"
    shell:
        "cat {input} | paste -s -d+ | bc > {output}"
```

Note: the role of “expand” to express a list of dependencies

Note: the “reduce” part of map-reduce; the “gather” part of scatter-gather

Exercise 2: Scatter-gather analysis (9 of 10)

Tying it all up

This simple example analysis can be described in the following `reana.yaml`

```
inputs:
  files:
    - Snakefile
workflow:
  type: snakemake
  file: Snakefile
outputs:
  files:
    - wordcount.txt
```

Note: All the details "live" in the `Snakefile` that we have been composing.

Hint: If time presses, download the solution from
`https://github.com/tiborsimko/reana-demo-wordcount`

Exercise 2: Scatter-gather analysis (10 of 10)

Running the example

The analysis example can be run as before

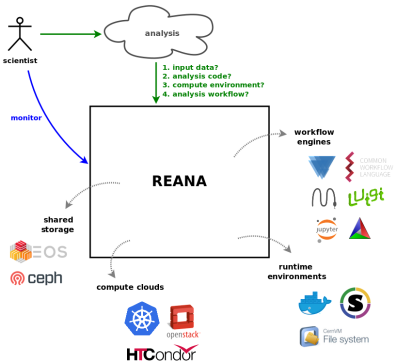
```
$ reana-client create -w wordcount
$ reana-client upload -w wordcount
$ reana-client start -w wordcount
...
$ reana-client status -w wordcount
$ reana-client logs -w wordcount
$ reana-client ls -w wordcount
$ reana-client download -w wordcount
```

Task 1: Modify the example to calculate the number of lines rather than words.

Conclusions

Conclusions

- driving reuse through preproducibility
- data + code + environment + workflow
  → reproducible analyses
- technology challenges: large containers, complex computational workflows
- sociology challenges: declarative programming paradigm, publish-or-perish culture
- synergies with computational reproducibility needs in astronomy, life sciences



https://www.reanahub.io https://github.com/reanahub