Retrospective Provenance without a Runtime Provenance Recorder

Timothy McPhillips Shawn Bowers
Khalid Belhajjame Bertram Ludäscher*

Overview / Exec Summary

- Scientific Workflows: ASAP!
- **Scripts are Scientific Workflows, too!**
  - .. so how can we help workflow script authors?
- **YesWorkflow (YW)**
  - **Prospective** provenance through YW-annotations:
    - script + annotations @begin, @end, @in, @out => workflow model
- **YW-Recon: Retrospective** provenance **without** a provenance recorder:
  - .. add @URI template-annotations

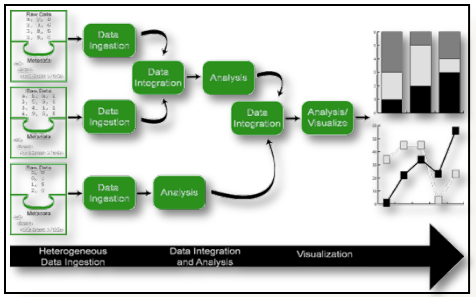
  => linking script persisted (meta-)data with workflow model

... => YW Retrospective Provenance Queries

- .. using YW workflow model to query YW-reconstructed runtime provenance!

Scientific Workflows: ASAP!

- **Automation**
  - wfs to **automate** computational aspects of science



- **Scaling** (exploit and optimize ***machine*** *cycles*)
  - wfs should make use of **parallel compute resources**
  - wfs should be able handle **large data**

Kepler

swift

- **Abstraction, Evolution, Reuse** *(**human** cycles)*
  - wfs should be easy to **(re-)use, evolve, share**

Taverna

- **Provenance**
  - wfs should capture **processing history**, **data lineage**
  - ➔ traceable data- and wf-evolution
  - ➔ **Reproducible Science**

VisTrails

TRIANA

ASKALON

Trident Workbench

Es war einmal …

Scientific Workflows …

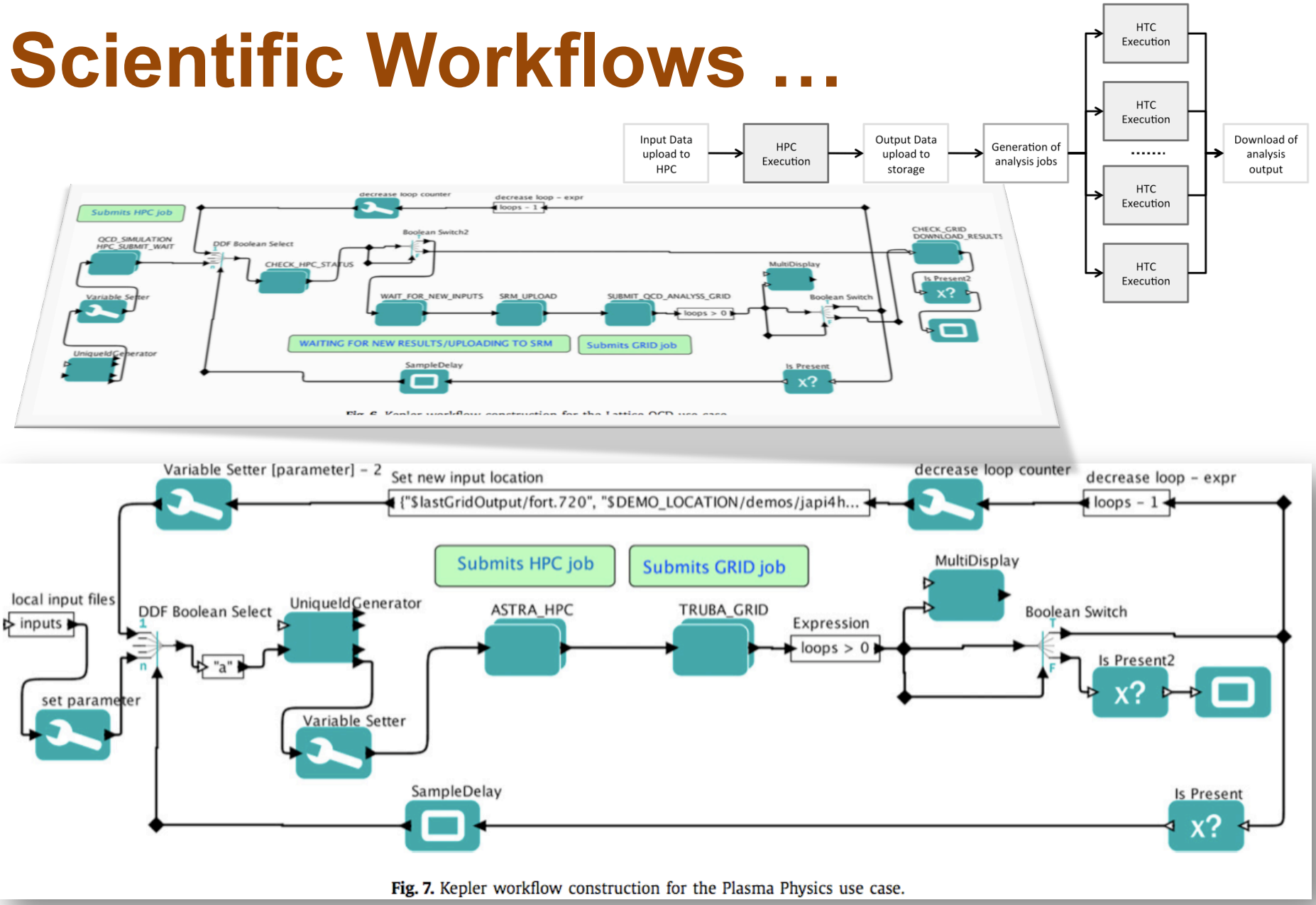
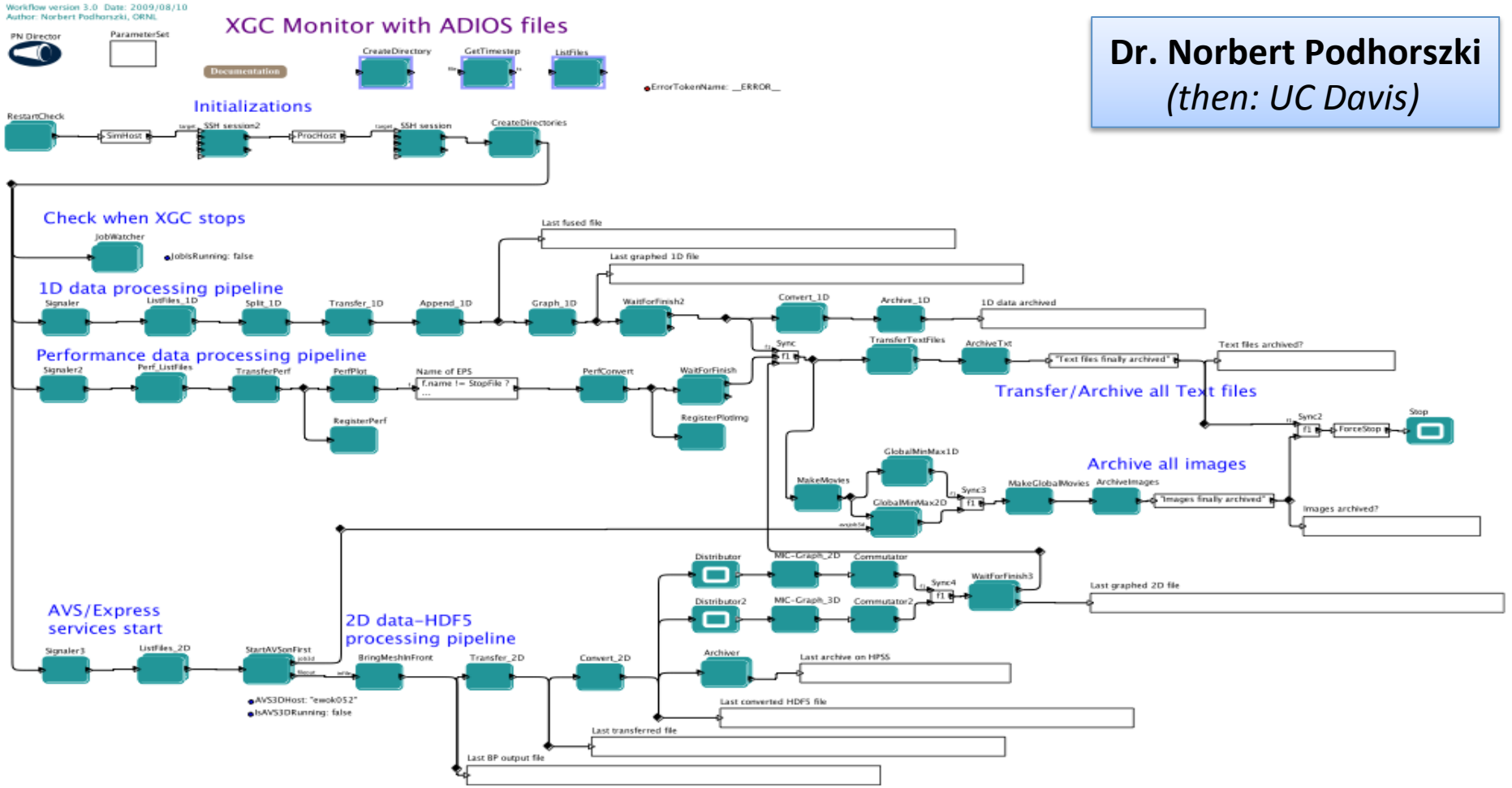


Fig. 7. Kepler workflow construction for the Plasma Physics use case.

Cabellos et al. *Computer Physics Communications* 182, 2011

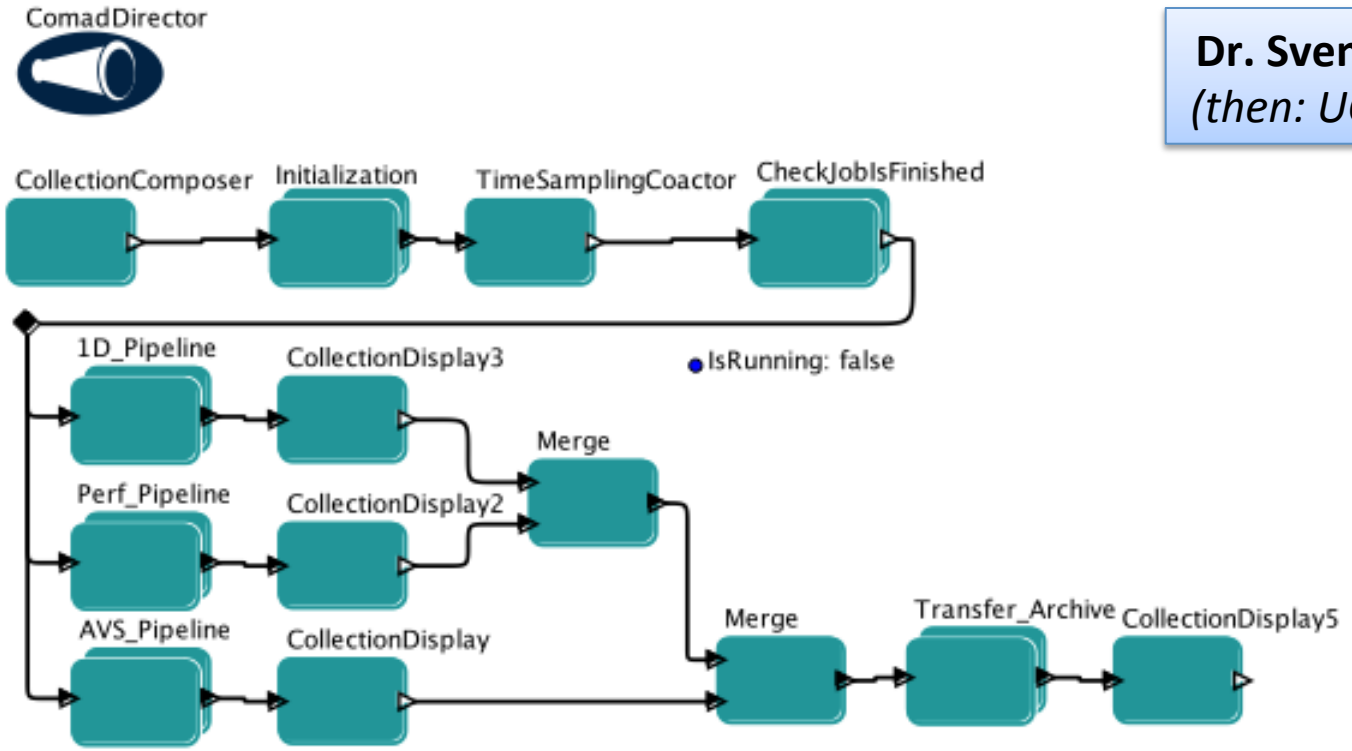
… are a wonderful thing …



Dr. Norbert Podhorszki
(then: UC Davis)

Figure 5.1: The original Monitoring Workflow. The top-level PN workflow shows all steps of the workflow. Additional parameters are contained in the parameter set in the top left corner. In the upper left corner, the workflow starts with initialization actors. Then, four pipelines that use a polling technique to observe a job execution follow: The fist pipeline just contains one actor that observes the job execution and adjusts the flag "JobIsRunning" according to the status of the job. The other three pipelines start with a sampling actor

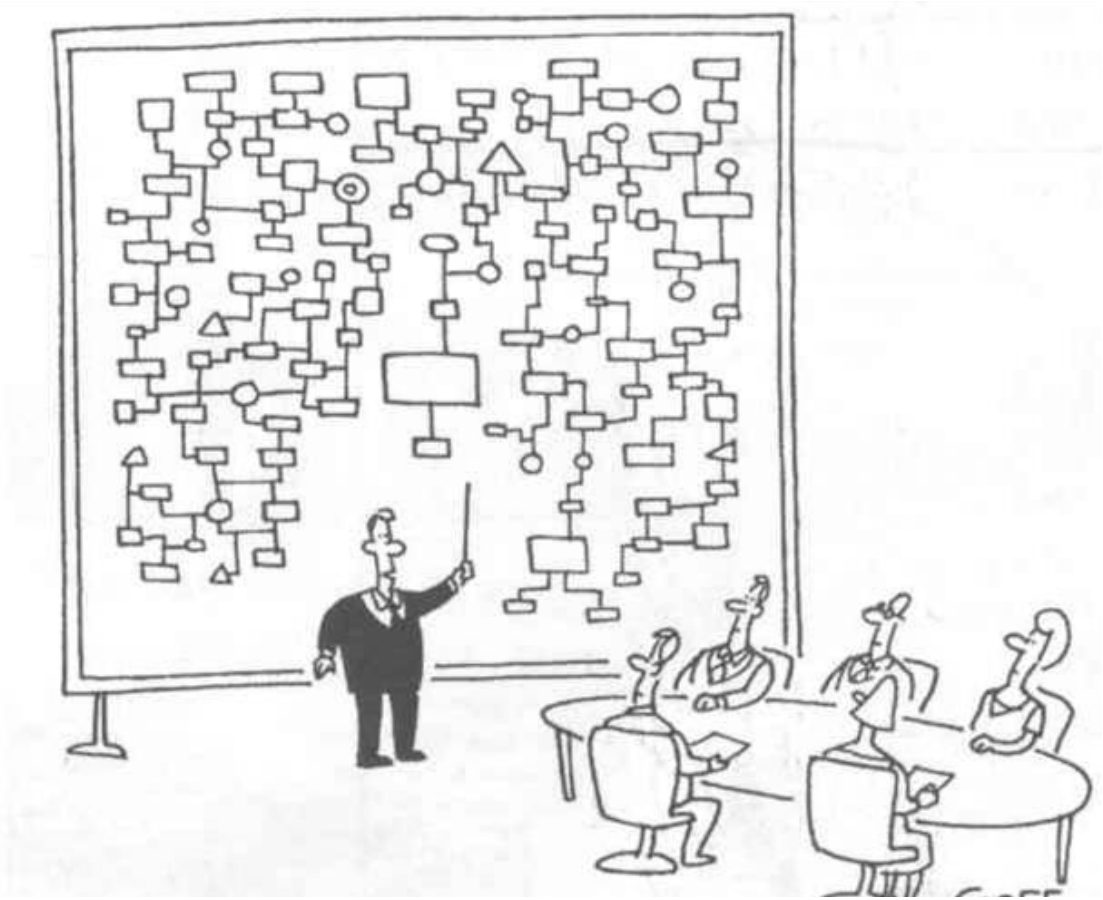
… after simplifying a bit (here: Kepler/**COMAD**)



Dr. Sven Köhler
(then: UC Davis)

Figure 5.2: Remodeled Monitoring Workflow using COMAD. Top-level model uses the COMAD director. Composites use SDF and DDF directors, but they do not need to handle stop tokens anymore.

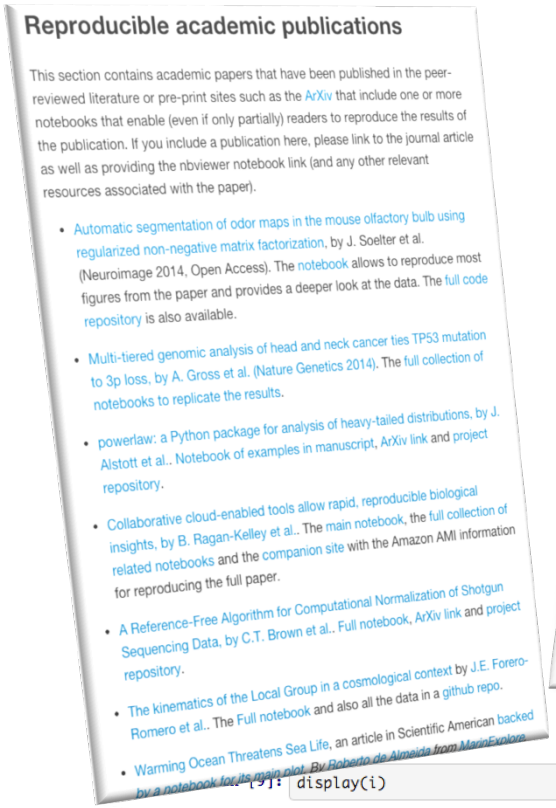
I beg your pardon, *I never promised you ..*

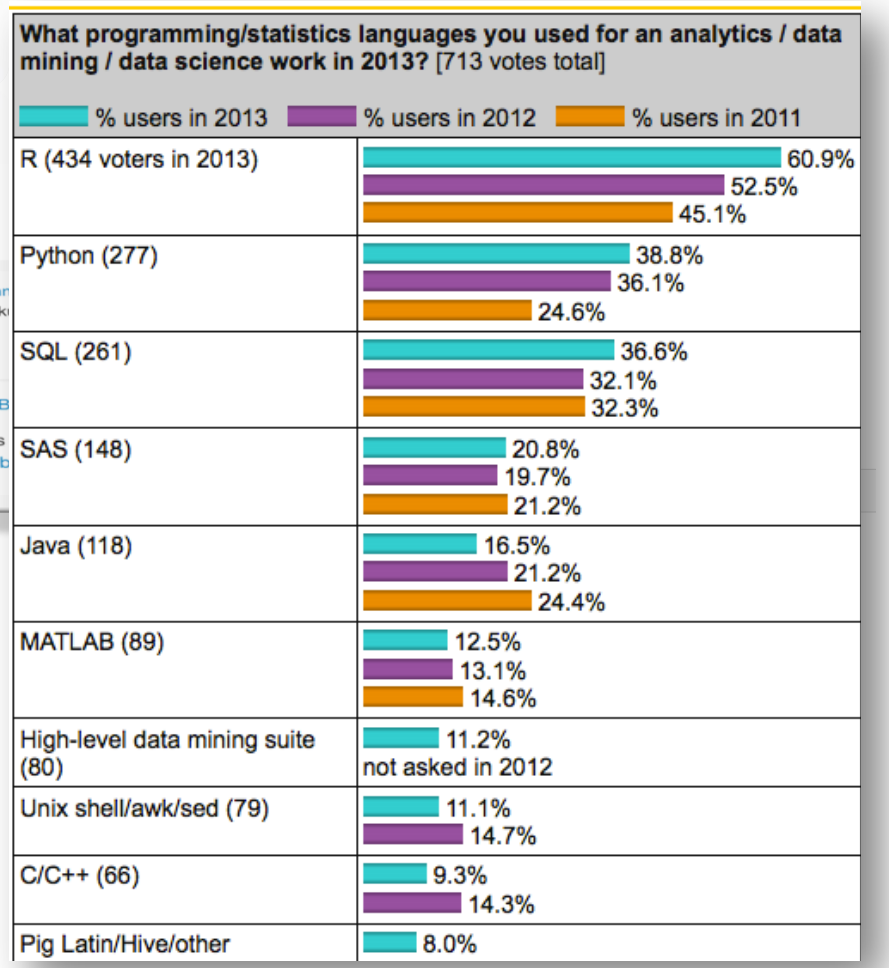


*"Thanks to our **Graphical UI** your scientific workflows will be much easier to develop, understand and maintain!"*

Hmm… this was supposed to be ***easier*** than programming!

Meanwhile, on a nearby planet …

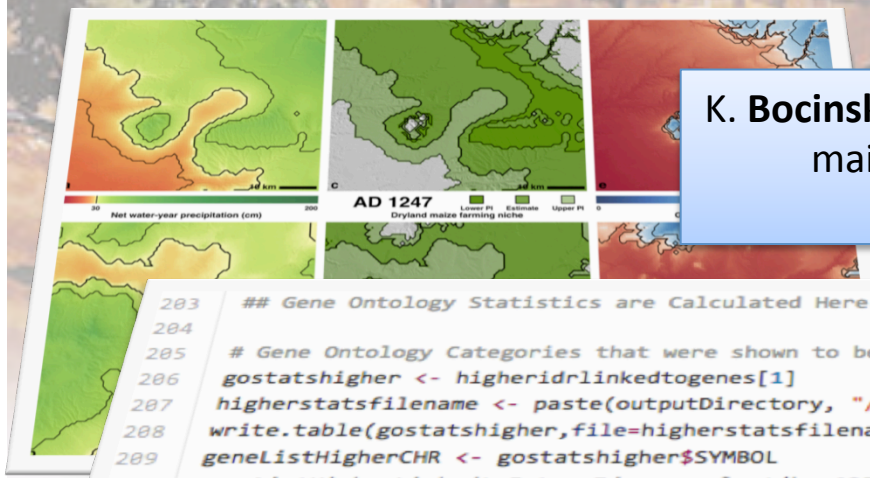




SKOPE: Synthesized Knowledge Of Past Environments

Bocinsky, Kohler *et al.* study rain-fed maize of **Anasazi**

- Four Corners; AD 600–1500. **Climate change** influenced **Mesa Verde Migrations**; late 13th century AD. Uses **network of tree-ring chronologies** to **reconstruct a spatio-temporal climate** field at a fairly high resolution (~800 m) from AD 1–2000. Algorithm estimates joint information in tree-rings and a climate signal to identify "best" tree-ring chronologies for climate reconstructing.



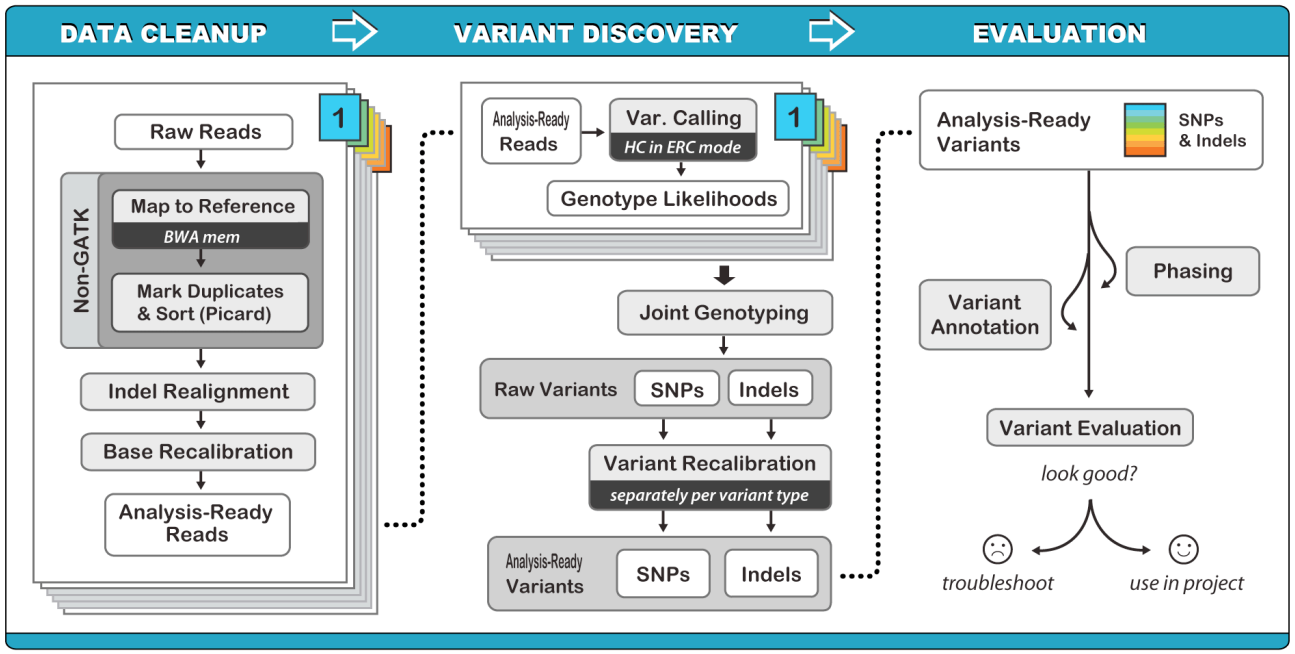
K. **Bocinsky**, T. **Kohler**, A 2000-year reconstruction of the rain-fed maize agricultural niche in the US Southwest. ***Nature Communications***. doi:10.1038/ncomms6618

```
203  ## Gene Ontology Statistics are Calculated Here.
204
205  # Gene Ontology Categories that were shown to be relatively Higher (more expressed) in the Experimental Condition.
206  gostatshigher <- higheridrlinkedtogenes[1]
207  higherstatsfilename <- paste(outputDirectory, "/", runName, "_", conditions[1],"_GOStatsHigher_",mytestcond[1], "_v
208  write.table(gostatshigher,file=higherstatsfilename, row.names=FALSE, col.names=FALSE, quote=FALSE, sep="\t")
209  geneListHigherCHR <- gostatshigher$SYMBOL
210  geneListHigherLinkedtoEntrezIds <- select(hgu133plus2.db, keys= geneListHigherCHR, "ENTREZID", "SYMBOL")
211  GOstatsGenesH <- geneListHigherLinkedtoEntrezIds[,2]
212
213  x <- org.Hs.egACCNUM
214  mapped_genes <- mappedkeys(x)
215  xx <- as.list(x[mapped_genes])
216  geneUniverse <- (unique(names(xx)))
```

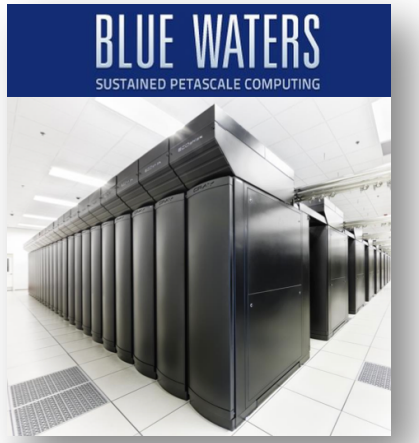
… implemented as an R Script …

… HPCBio Workflows @ Illinois

Broad Institute: Recommended workflow for variant analysis



Quickly, say: ***#!/bin/bash***



National Petascale Computing Facility

Liudmila Mainzer,
Victor Jongeneel
HPC Bio @ Illinois

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

It's time to shift control …

- … back from being **consumers** of someone else's (= our) tools ..
  - *"Just click here!"*
- … to **tool makers**!
  - Scientists who author workflows as scripts!

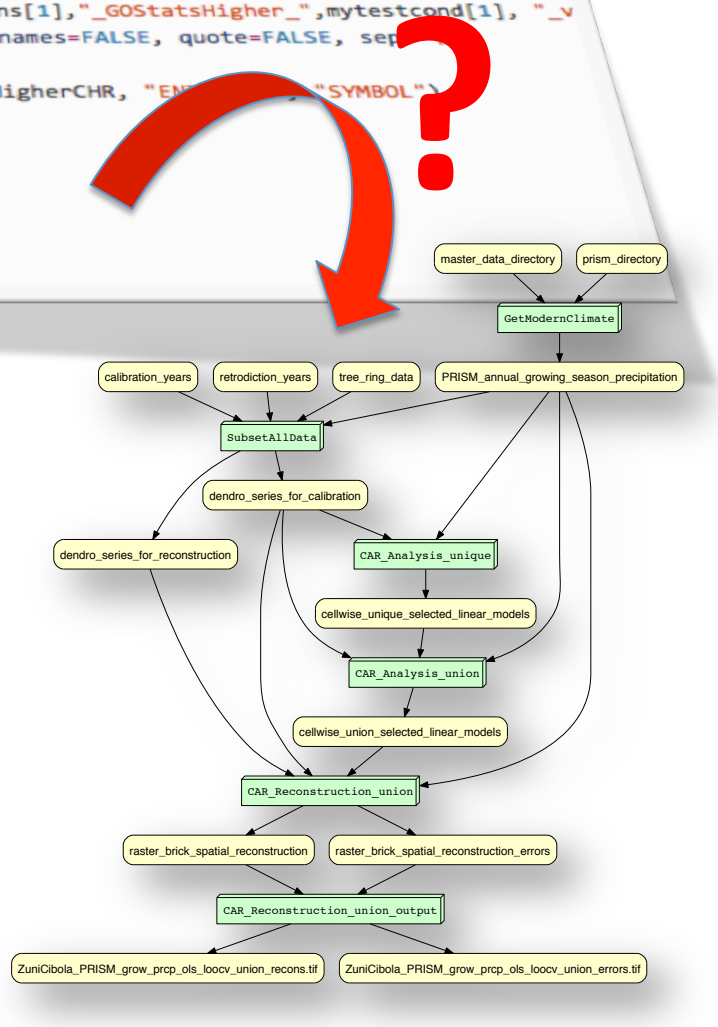
- **Go where the wild things (*users!*) are …**
  - Yes, develop for "end users" …
  - … but don't forget the **tool makers**!

- Can we do this **together**?

YesWorkflow:

Yes, scripts are workflows, too!

```
203  ## Gene Ontology Statistics are Calculated Here.
204
205  # Gene Ontology Categories that were shown to be relatively Higher (more expressed) in the Experimental Condition.
206  gostatshigher <- higheridrlinkedtogenes[1]
207  higherstatsfilename <- paste(outputDirectory, "/", runName, "_", conditions[1],"_GOStatsHigher_",mytestcond[1], "_v
208  write.table(gostatshigher,file=higherstatsfilename, row.names=FALSE, col.names=FALSE, quote=FALSE, sep
209  geneListHigherCHR <- gostatshigher$SYMBOL
210  geneListHigherLinkedtoEntrezIds <- select(hgu133plus2.db, keys= geneListHigherCHR, "EN       "SYMBOL")
211  GOstatsGenesH <- geneListHigherLinkedtoEntrezIds[,2]
212
213  x <- org.Hs.egACCNUM
214  mapped_genes <- mappedkeys(x)
215  xx <- as.list(x[mapped_genes])
216  geneUniverse <- (unique(names(xx)))
```



- **Script** *vs* Workflows/**ASAP**:
  - **A**utomation: *****
  - **S**caling: **
  - **A**bstraction: *
  - **P**rovenance: **

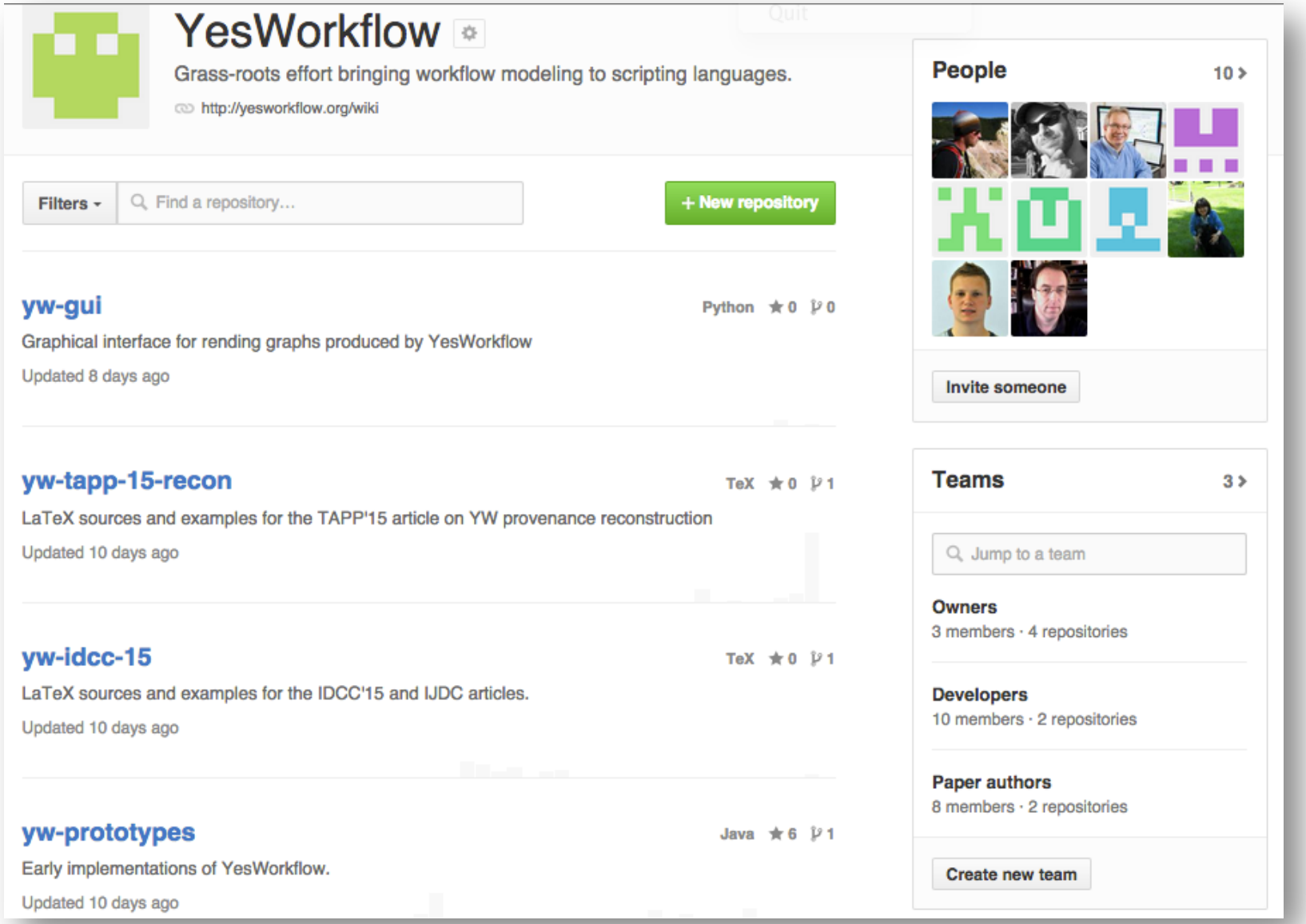
Enter: YesWorkflow! (yesworkflow.org)

- **YesWorkflow (YW)**
  - Grass-roots effort
  - … meeting the scientists/users **where they R!**
    - R, Matlab, (i)Python, Jupyter, …
  - Scripts + **simple** user annotations
- => **Reveal the workflow model/abstraction**

  … that underlies the (script) *implementation*

- => YW can give us more of **ASAP**!
  - First **YW:** AS**A**P (**A**bstraction)...
  - Then **YW-recon**: ASA**P** (reconstructing **runtime P**rovenance)

YesWorkflow.org



Related Work, other Approaches

… to bring workflow/provenance benefits to scripts:

- **Runtime Provenance Recorders:**
  - use (R, Python, ..) **libraries** and/or **code instrumentation** to capture **runtime observables**
    - file read/write, function calls, program variables & state, …
  - **noWorkflow** system
    - [Murta-Braganholo-Chirigati-Koop-Freire-IPAW14]
    - exploit Python profiling library to capture runtime provenance

  => helps with "S" and "P"
- **OS-level capture of (system) provenance**
  - Some talks at TaPP !?

YW (*prospective*) and YW-Recon (*retrospective*) Provenance

- **1. YW: Annotate Script => YW Model**
  - Annotate **@BEGIN..@END, @IN, @OUT**
  - Visualize, share, be happy ☺
- 2. Run script
  - Files are read and written
  - Folder- & Filenames have metadata
- **3. YW-Recon**
  - Use **@URI** tags **that link YW Model ⇔ Persisted Data**
  - Run URI-template queries
    - cf. "ls -R" & RegEx matching
- **4. YW-Query**
  - Answer the user's provenance queries

YW annotations: Model your Workflow!

```
1  # @BEGIN collect_data_set
2  # @PARAM cassette_id @PARAM accepted_sample @PARAM num_images @PARAM energies
3  # @OUT sample_id @OUT energy @OUT frame_number
4  # @OUT raw_image_path @AS raw_image
5  # ... @URI file:run/raw/{cassette_id}/{sample_id}/e{energy}/image_{frame_number}.raw
6  run_log.write("Collecting data set for sample {0}".format(accepted_sample))
7  sample_id = accepted_sample
8  for energy, frame_number, intensity, raw_image_path in collect_next_image(
9                          cassette_id, sample_id, num_images, energies,
10                         "run/raw/{cassette_id}/{sample_id}/e{energy}/image_{frame_number:03d}.raw"):
11     run_log.write("Collecting image {0}".format(raw_image_path))
12 # @END collect_data_set
13
14 # @BEGIN transform_images
15 # @PARAM sample_id @PARAM energy @PARAM frame_number
16 # @IN raw_image_path @AS raw_image
17 # @IN calibration_image @URI file:calibration.img
18 # @OUT corrected_image @URI file:run/data/{sample_id}/{sample_id}_{energy}eV_{frame_number}.img
19 # @OUT corrected_image_path @OUT total_intensity @OUT pixel_count
20     corrected_image_path = "run/data/{0}/{0}_{1}eV_{2:03d}.img".format(sample_id, energy, frame_number)
21     (total_intensity, pixel_count) = transform_image(raw_image_path, corrected_image_path, "calibration.img")
22     run_log.write("Wrote transformed image {0}".format(corrected_image_path))
23 # @END transform_images
```

mark the code block

… and data inputs/outputs

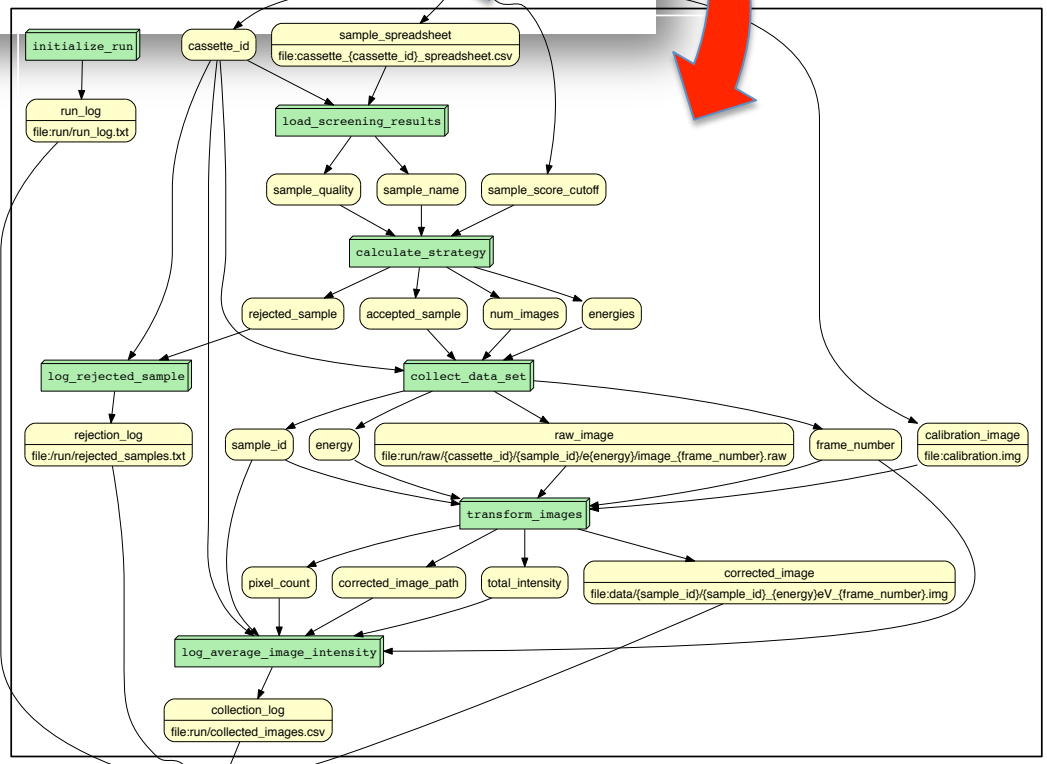
Figure 1. YW-annotated fragment of a Python script for data collection from protein crystal samples. YW-annotations `@BEGIN` and `@END` delimit code blocks; `@IN` and `@OUT` tags model relevant input and output data elements of a block; `@PARAM` identifies a block's parameters. `@URI` templates for raw images (line 5) and corrected images (line 18) link conceptual-level data elements such as `raw_image` with runtime resources (data files and their file paths). Executable script code is greyed out to emphasize YW-annotations. A program variable (`raw_image_path`) is highlighted in the code (lines 8, 11, 21): aliases (lines 4, 16) are used to link such program-level objects to the scientist's concepts (here: `raw_image`). Full example available from [MBL15].

YesWorkflow: Prospective & Retrospective Provenance … (almost) for free!

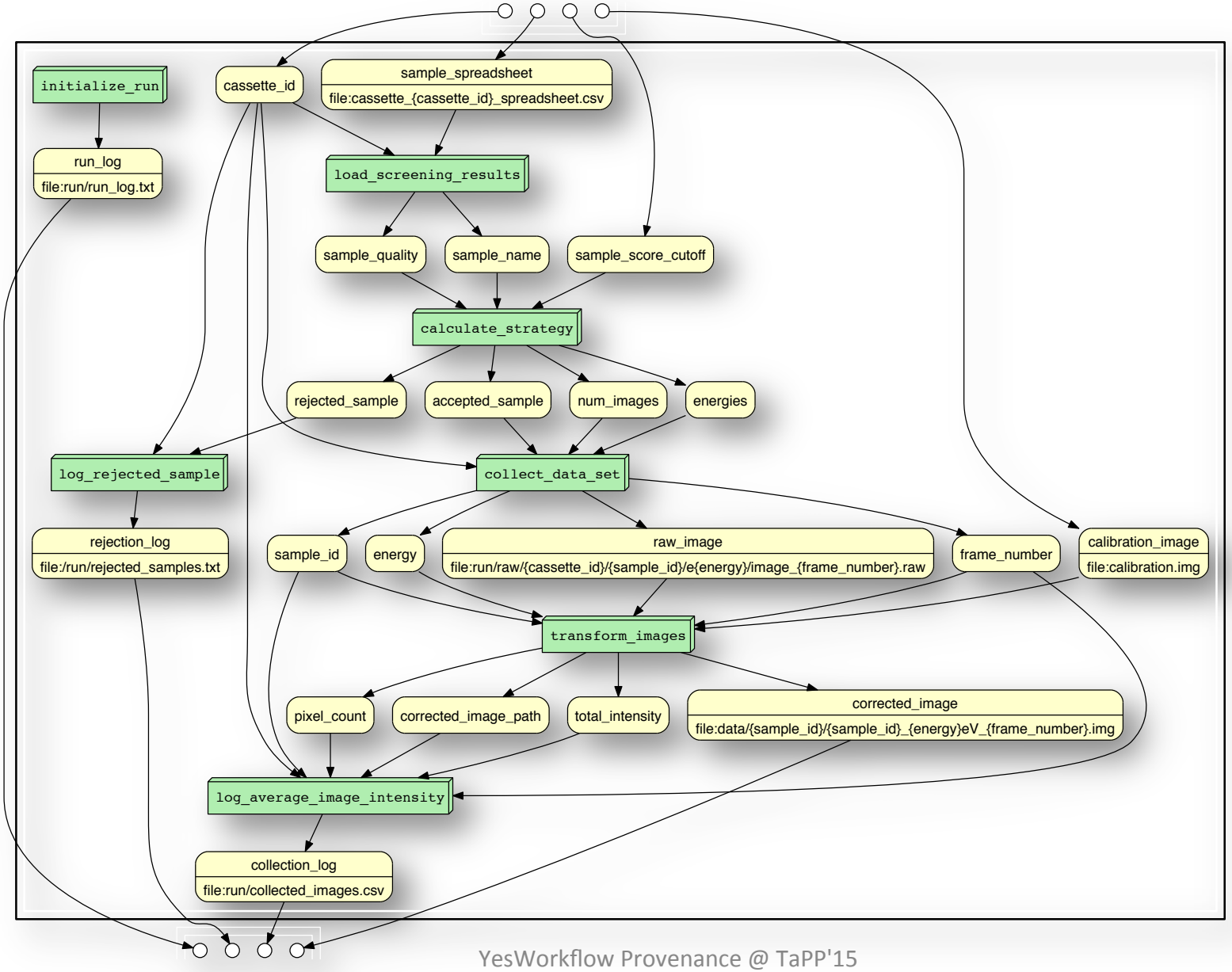
```
# @BEGIN collect_data_set
# @PARAM cassette_id @PARAM accepted_sample @PARAM num_images @PARAM energies
# @OUT sample_id @OUT energy @OUT frame_number
# @OUT raw_image_path @AS raw_image
# @URI file:run/raw/{cassette_id}/{sample_id}/e{energy}/image_{frame_number}.raw
run_log.write("Collecting data set for sample {0}".format(accepted_sample))
sample_id = accepted_sample
for energy, frame_number, intensity, raw_image_path in collect_next_image(
                        cassette_id, sample_id, num_images, energies,
                        "run/raw/{cassette_id}/{sample_id}/e{energy}/image_{frame_number}.raw"):
    run_log.write("Collecting image {0}".format(raw_image_path))
# @END collect_data_set
```

YW!

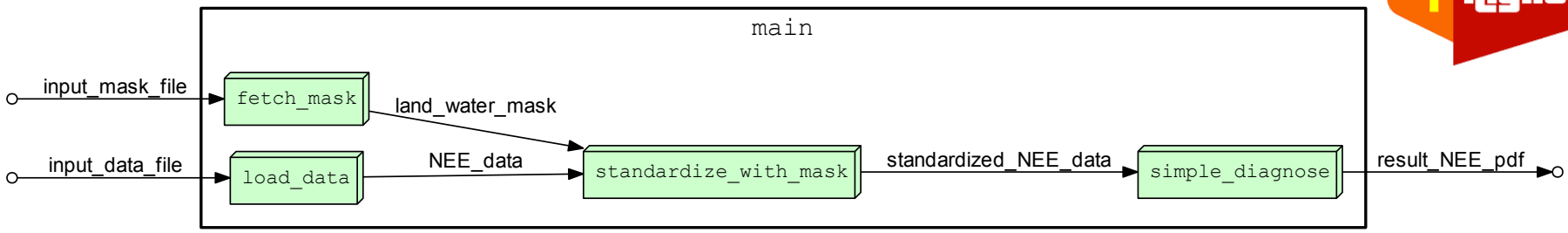
- YW annotations in the script (R, Python, Matlab) are used to **recreate the workflow view** from the script …



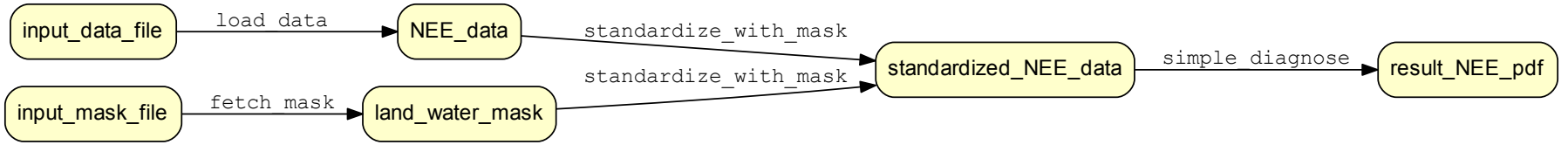
Voila! The Workflow revealed!



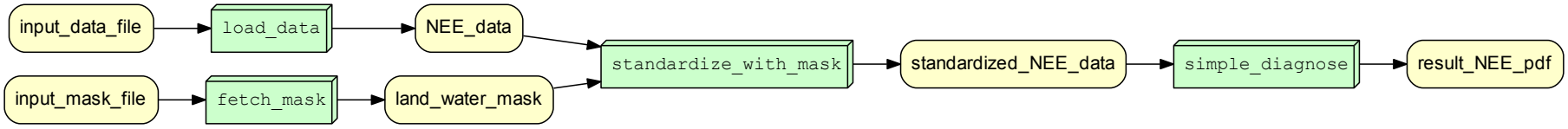
Get 3 views for the price of 1!



Process view



Data view



Combined view

Paleoclimate Reconstruction (EnviRecon.org)

- … explained using **YesWorkflow!**

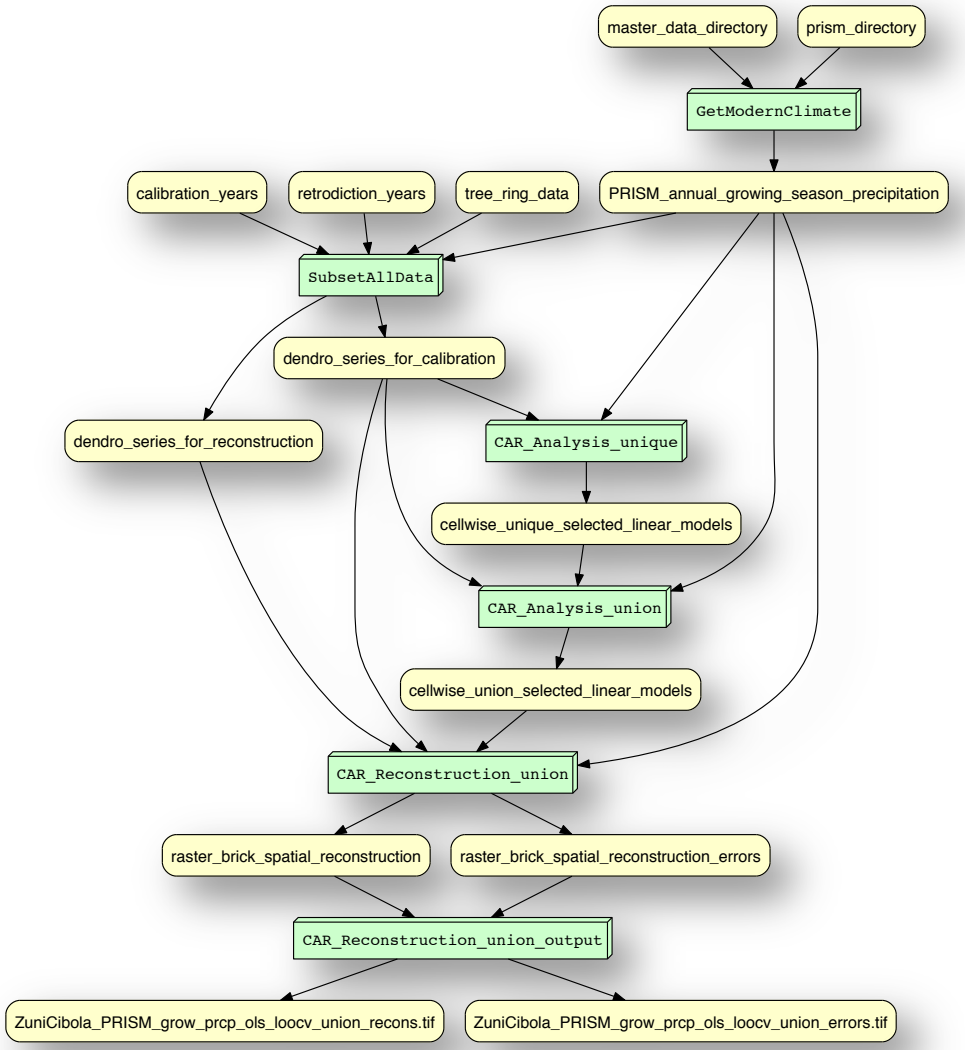
Kyle B., (computational) archaeologist:

"It took me about 20 minutes to comment. Less than an hour to learn and YW-annotate, all-told."

SKOPE + Kurator + DataONE

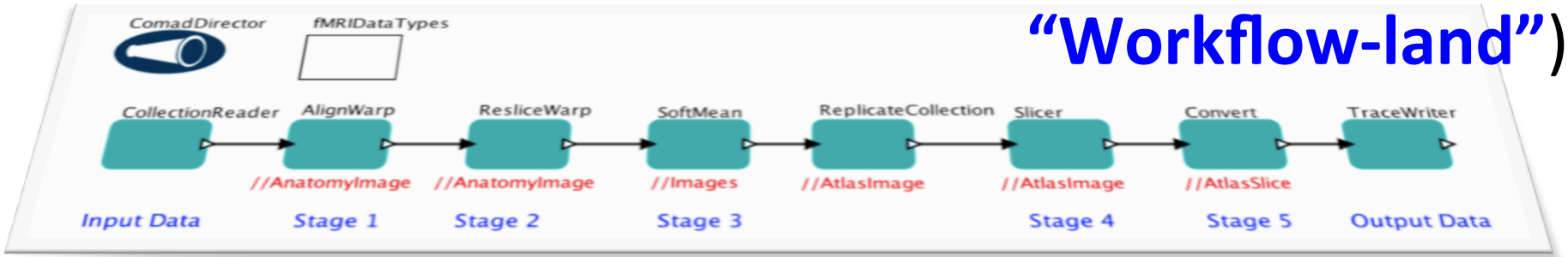
Data Observation Network for Earth

=> YesWorkflow.org

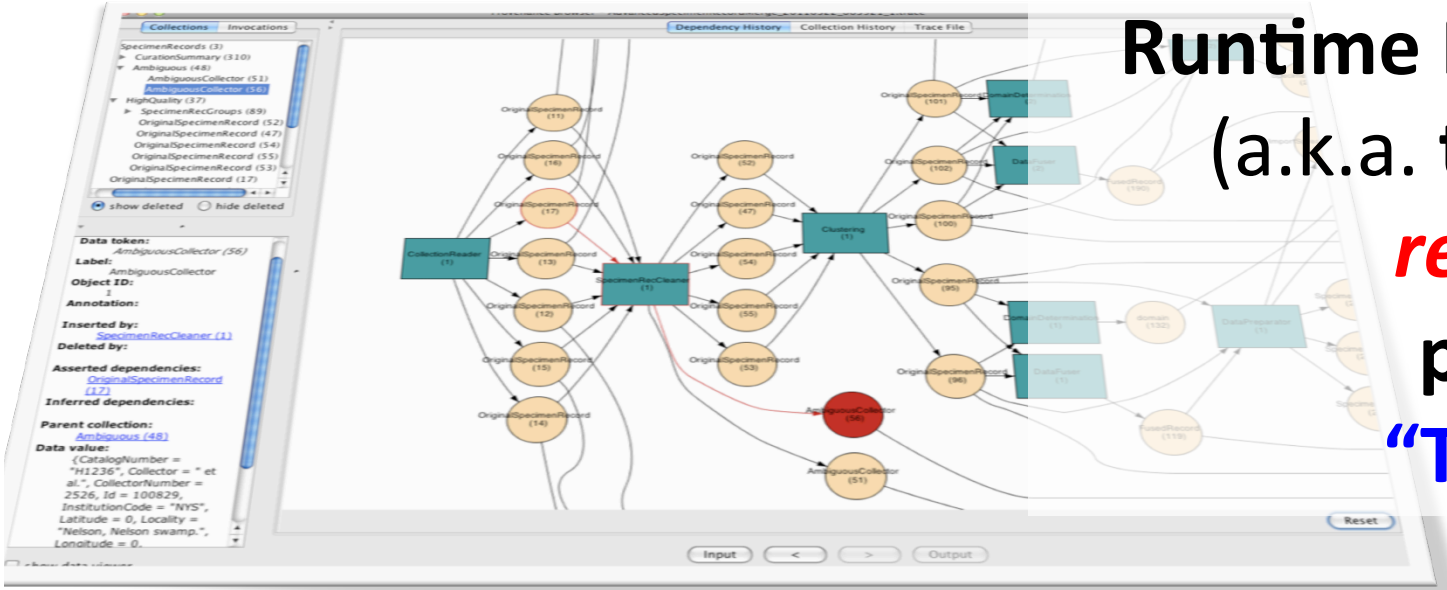


Provenance Lands

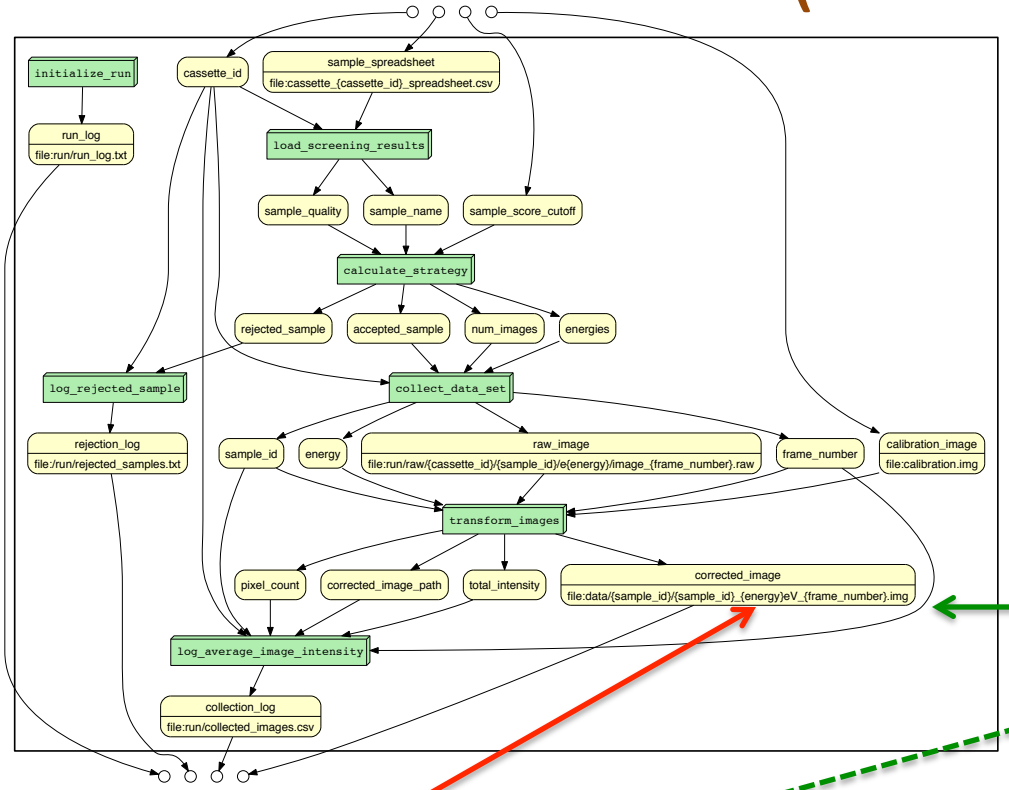
Workflow Modeling & Design (a.k.a. *prospective* provenance "Workflow-land")



Runtime Provenance (a.k.a. traces, logs, *retrospective* provenance, "Trace-land")



YW-RECON: Prospective & **Retrospective** Provenance … (almost) for free!



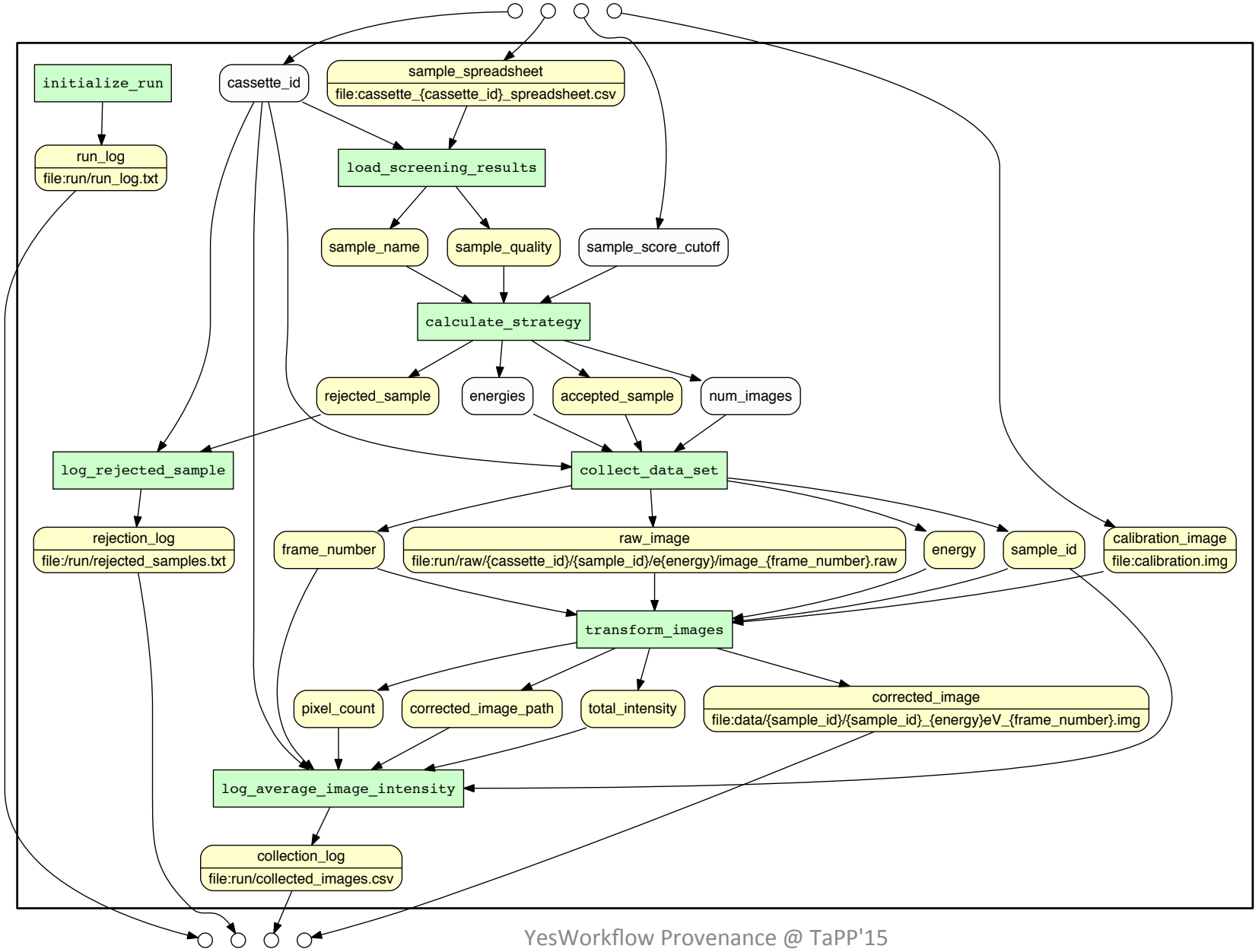

```
├── raw
│   └── q55
│       ├── DRT240
│       │   ├── e10000
│       │   │   ├── image_001.raw
...    ...  ... ...
│       │   │   └── image_037.raw
│       │   └── e11000
│       │       ├── image_001.raw
...    ...      ...
│       │   │   └── image_037.raw
│       └── DRT322
│           ├── e10000
│           │   ├── image_001.raw
...        ... ...
│           │   └── image_030.raw
│           └── e11000
│               ├── image_001.raw
...            ...
│               └── image_030.raw
├── data
│   ├── DRT240
│   │   ├── DRT240_10000eV_001.img
... ... ...
│   │   └── DRT240_11000eV_037.img
│   └── DRT322
│       ├── DRT322_10000eV_001.img
...    ...
│       └── DRT322_11000eV_030.img
│
├── collected_images.csv
├── rejected_samples.txt
└── run_log.txt
```

- **URI-templates *link*** conceptual entities to **runtime provenance** "left behind" by the script author …
- … facilitating provenance reconstruction

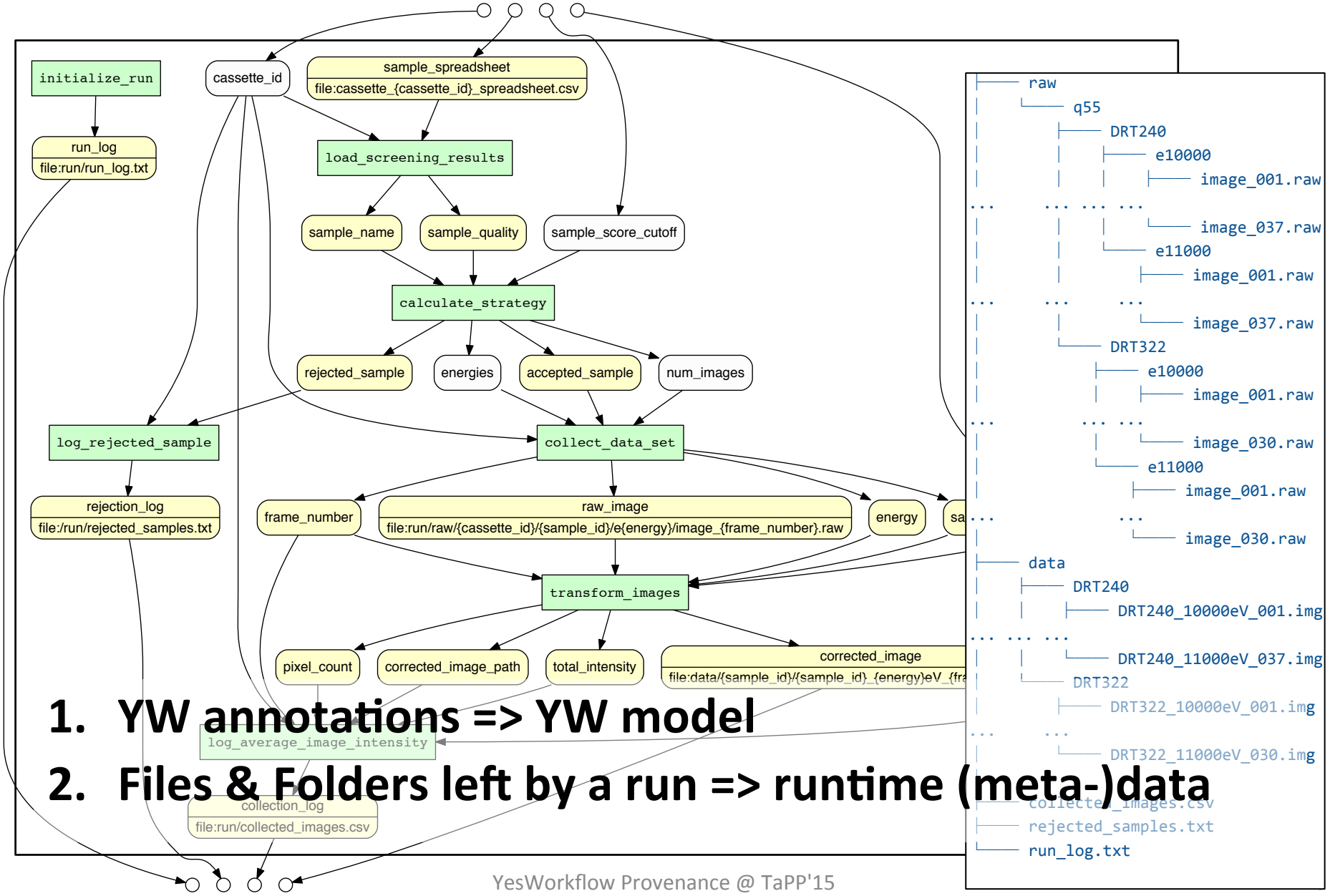
YW (*prospective*) and YW-Recon (*retrospective*) Provenance

- **1. YW: Annotate Script => YW Model**
  - Annotate **@BEGIN..@END, @IN, @OUT**
  - Visualize, share, be happy ☺
- **2. Run script**
  - Files are read and written
  - Folder- & Filenames have metadata
- **3. YW-Recon**
  - Use **@URI** tags **that link YW Model ⇔ Persisted Data**
  - Run URI-template queries
    - cf. "ls -R" & RegEx matching
- **4. YW-Query**
  - Answer the user's provenance queries

Data collection workflow (X-ray diffraction)

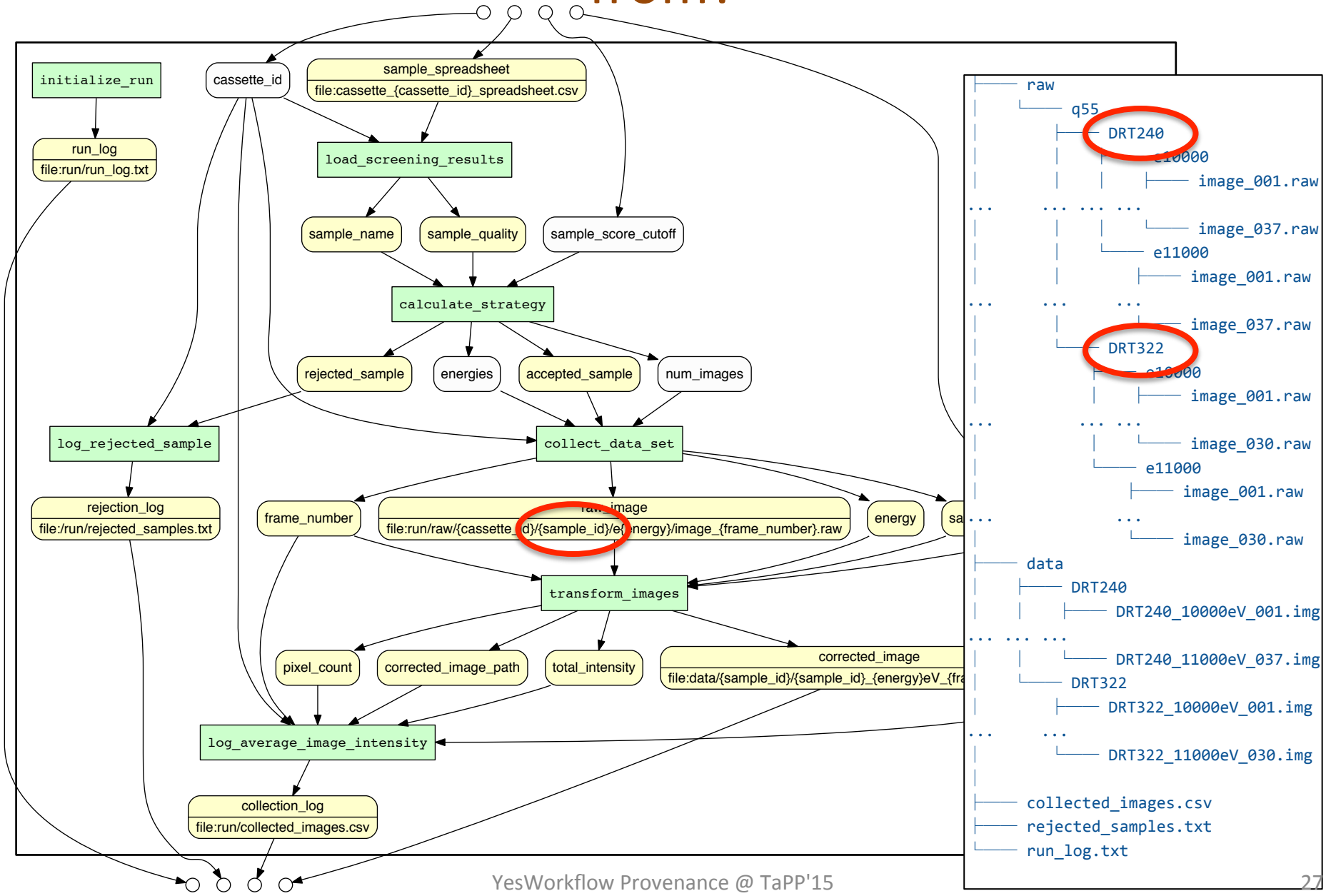


Data collection workflow: **runtime data**

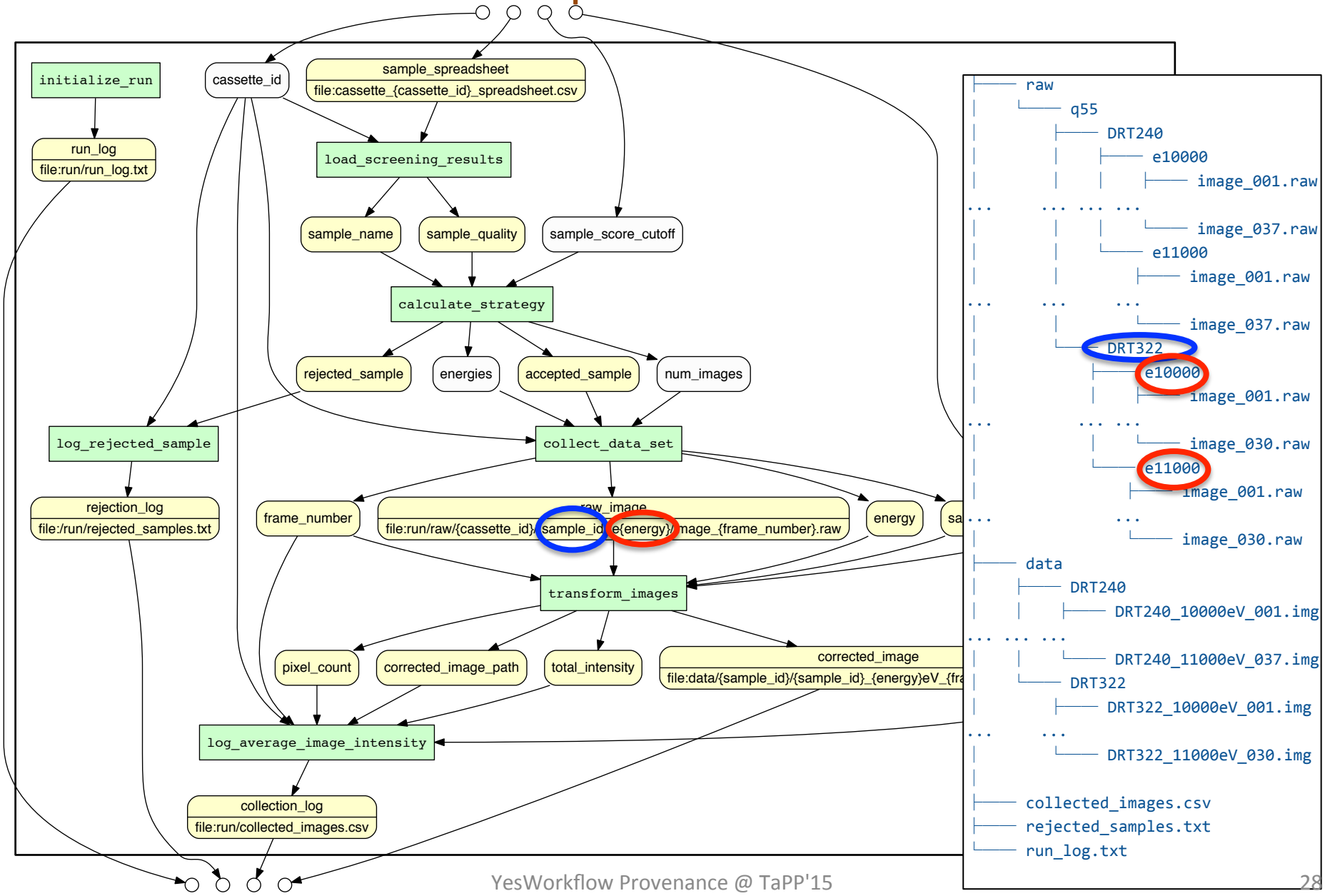


1. **YW annotations => YW model**
2. **Files & Folders left by a run => runtime (meta-)data**

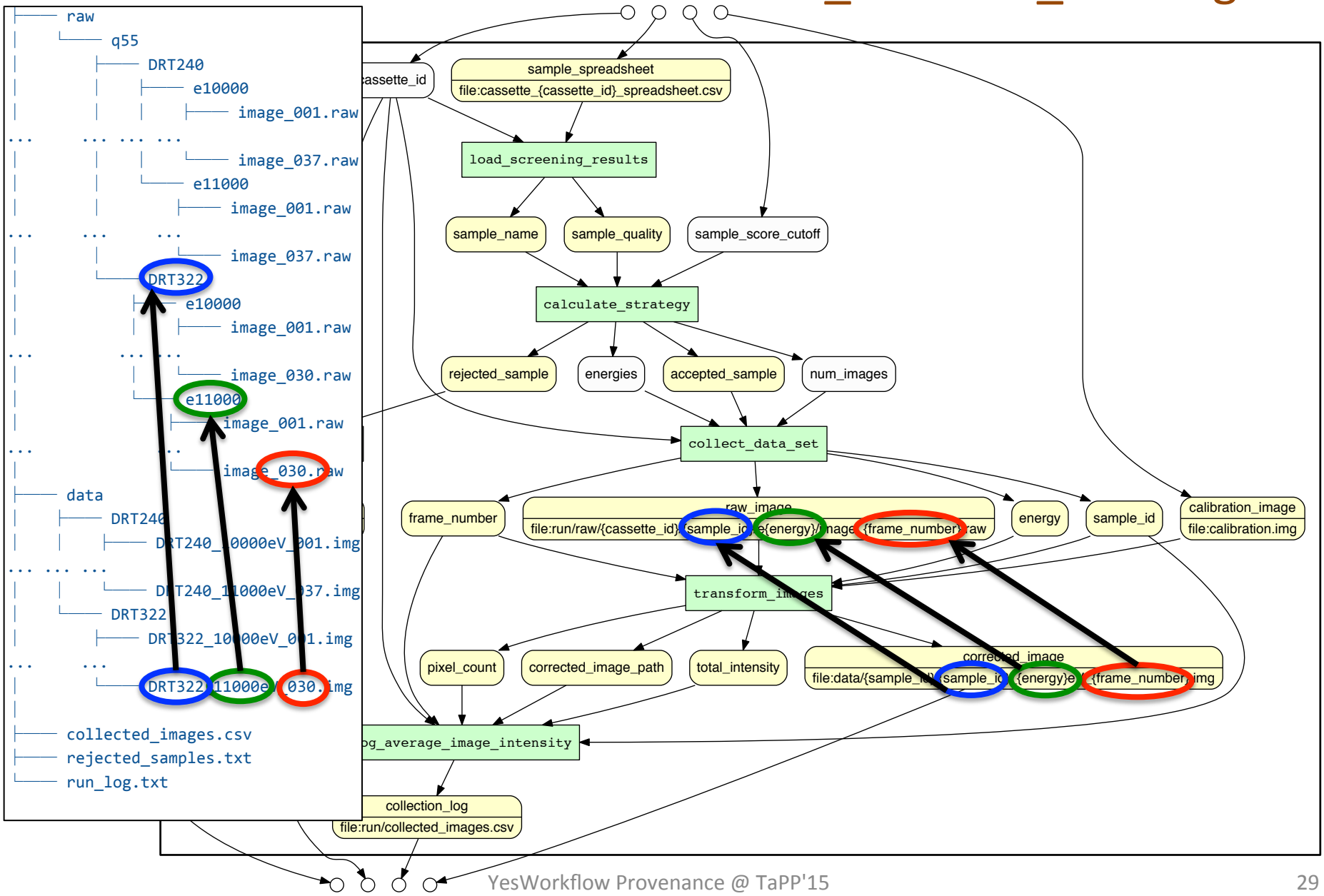
Q_1: What **samples** did the script run collect images from?



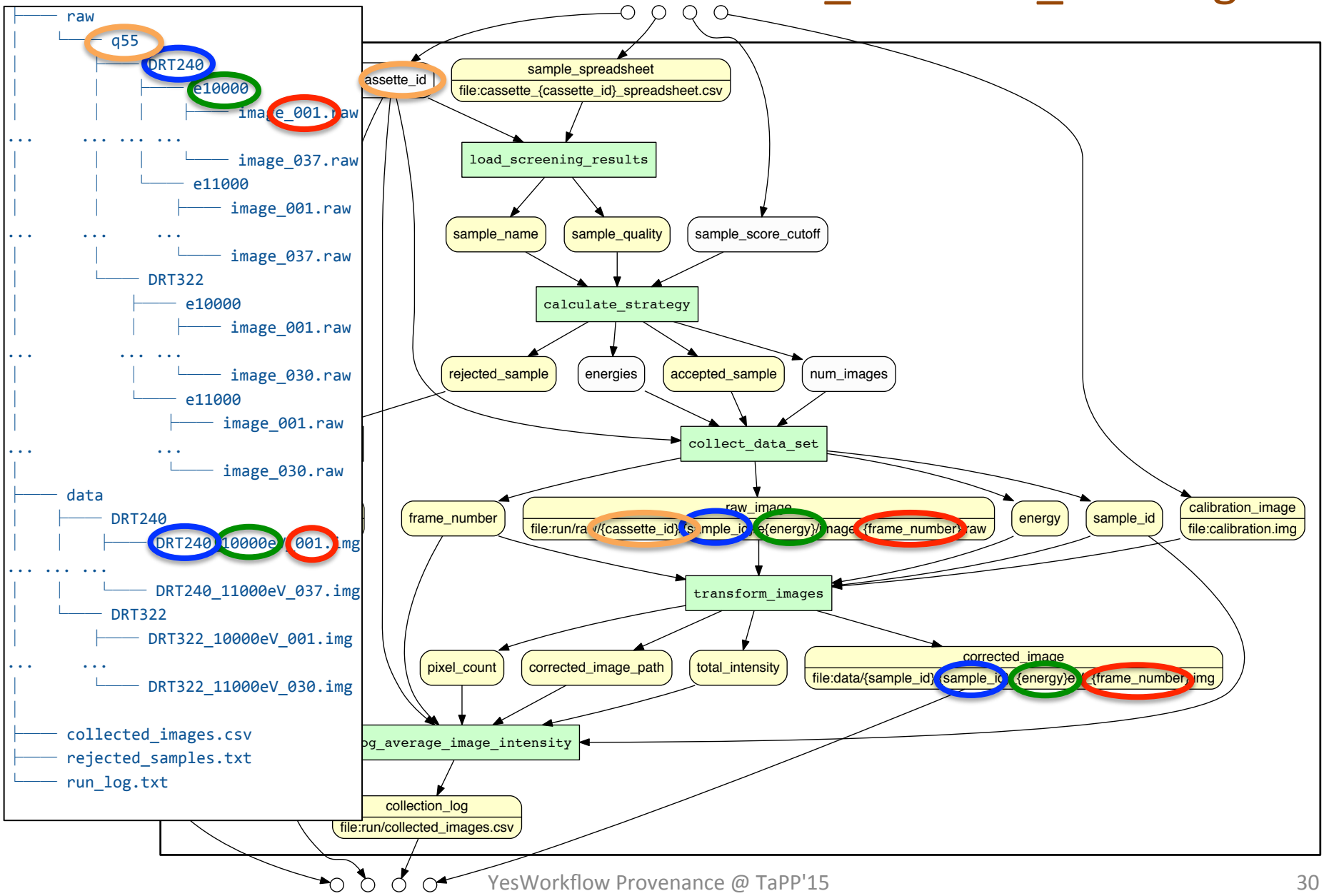
Q_2: What **energies** were used for image collection from sample **DRT322**?



Q_3: Where is the raw image of the corrected image DRT322_11000ev_030.img?



Q_5: What **cassette-id** had the sample leading to **DRT240_10000ev_001**.img?



Querying Provenance

(Q_2) What energies were used during collection of images from sample `DRT322`***?*** The scientist's solution is to look at the contents of the `run/raw/q55/DRT322` directory. The names of subdirectories indicate the values of the energies.

General solution using YW: Assume that 'what energies' and 'from sample `DRT322`' mean 'what values of energy as seen by `collect_data_set` when `sample_id` equals `DRT322` as seen by `collect_data_set`'. Then the solution is to look for all persisted outputs of `collect_data_set` that include both energy and `sample_id` in the expanded URI template for the output. Extract the value of energy from each such path for which `sample_id` equals `DRT322` and return the set of unique values. Q_2 can be expressed as the Datalog query:

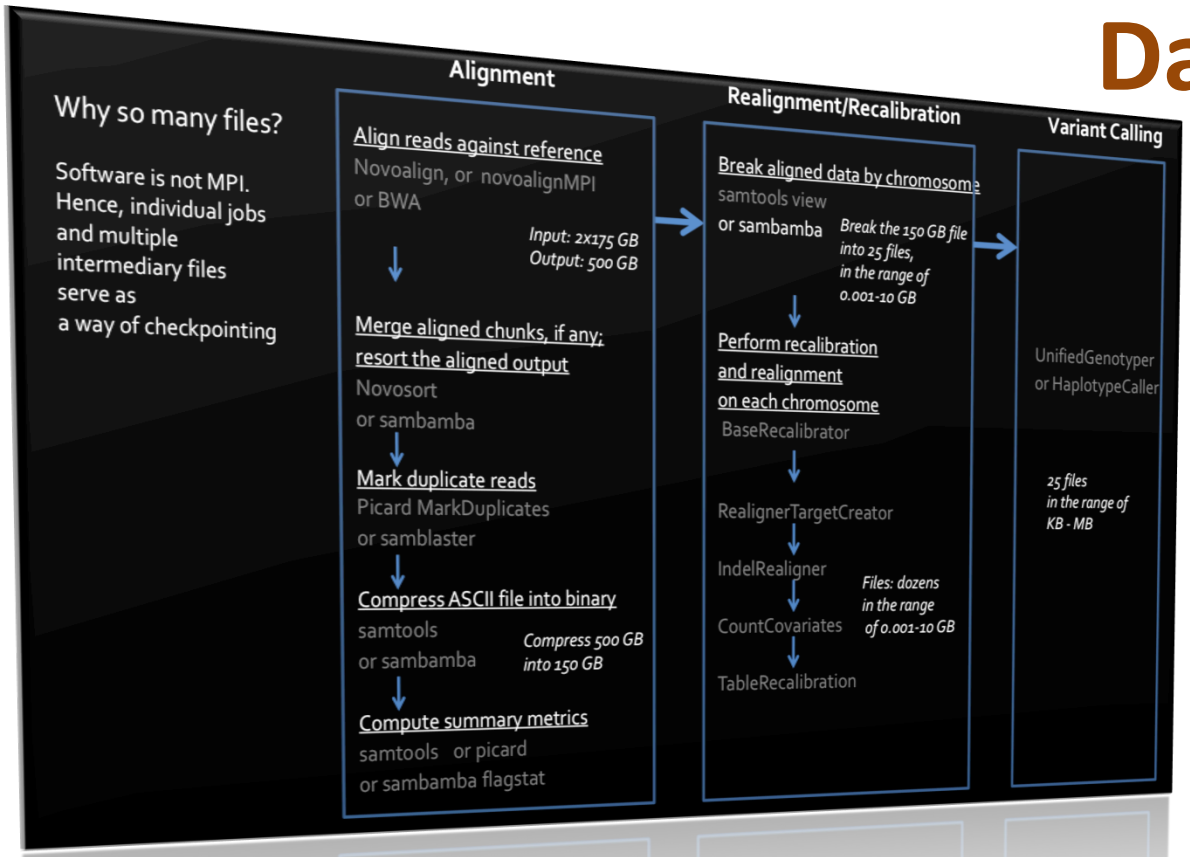
```
energies_used(EnergyValue) :-
    resource_metadata(collect_data_set,raw_image,ResourceId,
                      sample_id,"DRT322"),
    resource_metadata(collect_data_set,raw_image,ResourceId,
                      energy,EnergyValue).
```

For our example, the above query will yield the answers `10000` and `11000`.

Base Relation	***Description***
`program`(*id,name,begin_annot_id,end_annot_id*)	Identifier, name, and annotation identifiers for begin and end of workflow program blocks
`port`(*id,type,name,annot_id*)	Identifier, type (*in*, *out*, or *param*), name, and annotation identifier of program ports
`port_alias`(*port_id,alias_name*)	Alias names optionally given to ports (specified via `@AS` annotations)
`has_in_port`(*program_id,port_id*)	Input ports of program blocks
`has_out_port`(*program_id,port_id*)	Output ports of program blocks
`channel`(*id,binding*)	Assignment of port names (script variables) or aliases to channels
`port_connects_to_channel`(*port_id,channel_id*)	Assignment of ports to channels
`port_uri`(*port_id,uri_template*)	URI template optionally assigned to port
`uri_variable`(*id,name,port_id*)	Identifier, name, and associated port of URI metadata variables
`resource`(*id,uri*)	Identifier and expanded URI (file path) of resources created by a run of the script
`resource_channel`(*resource_id,channel_id*)	Resources that were input to or output by a channel during a run of the script

Table 1 Base relations generated by the YesWorkflow provenance reconstruction process

Data & Workflow Management



Source: L Mainzer, V Jongeneel (IGB & NCSA)

1. Large **total data** footprint
2. Large **number of files**
3. Large number of **simultaneous but independent** non-MPI **computations**
4. **Keeping track** of what was done to the data: large amount of **metadata**
5. Workflow bottlenecks: fans & merges; more fans

Taking YW for a spin …

- *"To document **on-the fly**, specifically for a given workflow configuration invoked:*
  - ***do not** insert annotations into code,*
  - *but rather **have code print annotations** into a special **log** during execution,*
  - *then parse that log!"* – Liudmila Mainzer

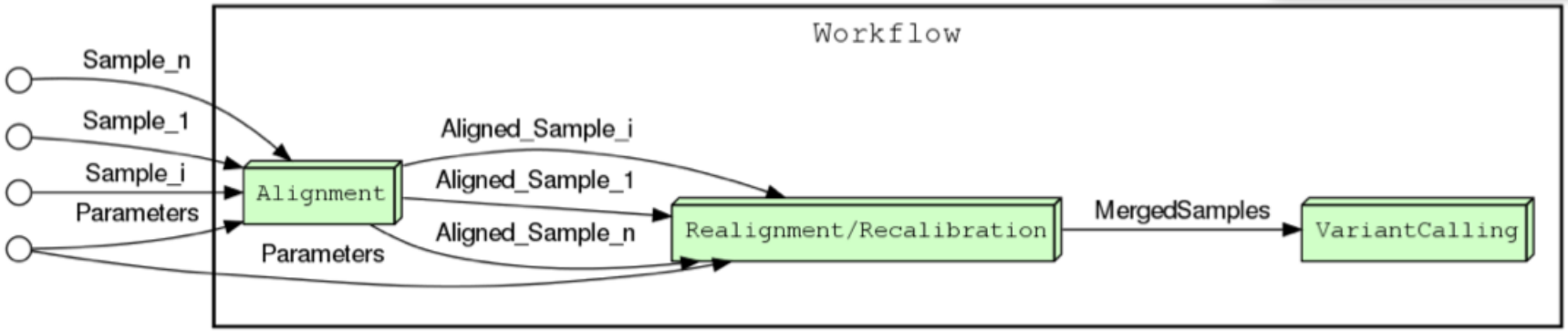
```
#!/bin/bash

# @begin Workflow
# @in x @as InputFileFolder
# @in y @as Parameters
# @in x1 @as Sample_1
# @in x1 @as Sample_i
# @in x1 @as Sample_n

# get input x somehow
# get input y somehow

# @begin Alignment
# @in y @as Parameters
# @in x1 @as Sample_1
# @in xi @as Sample_i
# @in xn @as Sample_n

etcetera
```



ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Source: L Mainzer, V Jongeneel (IGB & NCSA)

Conclusions & the Road ahead …

- **YW: Go where the users are!**
  - … they already capture provenance through metadata!
- **Beware your level of provenance abstraction**
  - Let the user provide a workflow model easily!
- **YW-Recon**:
  - … finishing support for **retrospective provenance *without*** using a runtime provenance recorder!
  - Key insight: scientists already leave provenance "bread crumbs" behind! (it's not an accident!)
  - Exploit this **annotations**: **URI**-templates
  - Extend YW to exploit log files
- **YW-GUI:**
  - Exploring & experimenting with a UI
    - Let script **author** immediately see effect of YW-annotations
    - Let scrip **(re-)user** explore the code and data provenance
- **YW-Query:**
  - ? Datalog, RPQ, … for querying provenance …