Preserving analyses: where decomposition into graphs helps

Follow up on Research Objects

August 2026

Eric / Maxim /

Summary version

An analysis can be decomposed into a graph

It started with data cards that describes dataset, but do not describe what is done with/to that dataset.

- An analysis is a set of connected research objects, not a single object: the dataset.
- The graph is a decomposition of the analysis into its components and the relations between them.
- It makes explicit dependencies that are otherwise spread across the notes, the code, and other sources.
- It connects the machine-resolvable references to the note that explains the analysis. It does not replace the note.
- For now, the graphs are built only from the analysis notes and the code: they identify the software and configuration an analysis used, but not the experiment-wide knowledge of how such components are normally built, configured, accessed, and run.

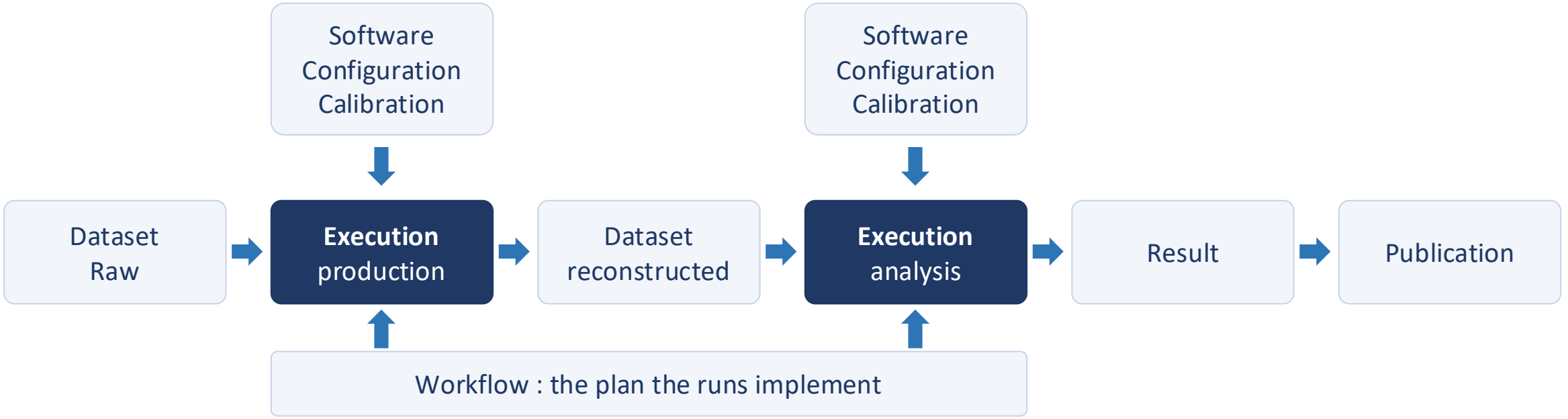
Example

  - *A published result depends on multiple components: dataset, analysis code, configuration, calibration, and execution. The graph states those connections rather than leaving them implicit.*

A data card describes the dataset. The graph describes what was done to it.

Nodes and edges

Calibration is itself produced by an execution,



Nine classes (the boxes): *Configuration*, *Execution, Dataset, Software, Workflow, Artifact, Publication, Calibration, Analysis*
And **relations (the arrows)**: *used, was_generated_by, was_associated_with, was_derived_from, is_part_of, published_as*
Classes and relations follow the PROV-O provenance vocabulary.

How an object is recorded

Each object is a short machine-readable record.

```
id:      configuration/star/run12-uu-good-list
class:   Configuration
label:   U+U 193 GeV good run list
identifier:
    scheme:  DOI                # assigned when the object is published
reference:
    locator: Run12UUgood.list
    grade:   prose              # pinned | released | moving | runtime | prose | absent
uses:    [publication/star/psn0604]
used_by: [execution/star/run12-uu-treemaker]
```

- Both directions are recorded. `uses` and `used_by` are written at registration, so a consumer can be found easily.
- A LinkML schema is under discussion with JLAB colleagues.

A durable identifier, a DOI, keeps a reference resolvable.

Decomposition makes reproducibility gaps visible

The graph formalizes complex dependencies among objects, making them explicit and amenable to inspection.

- Each component should have a clear identity and a resolvable reference.
- A gap appears when a reference is missing, ambiguous, points to something that can change, or is resolved only when the job runs.

Examples

- ***Moving reference****: a citation points to a branch rather than to the version used for the publication.*
- ***Input with no released version****: a run list, or a file produced by one analyst and never versioned.*
- ***Resolved at run time****: a library is loaded from a fixed filesystem path, with no version recorded.*
- ***Service dependency****: an object can be resolved only while the underlying database or service remains available and intact.*

Being able to find a component is not the same as knowing which version was used.

Common patterns

Inspecting several graphs reveals recurring patterns.

- The same kinds of preservation issues appear across different analyses.

Examples

- **Citation without pinning** → cite the exact commit, release or production tag used.
- **Input without a released version** → give run lists and intermediate files an authoritative version.
- **Inputs and parameters of a run not recorded** → retain what the execution actually resolved and used.
- **Published numbers typed by hand** → generate them from the execution that produced them.
- **Documentation drift** → review the note against the code and the workflow.
- **And more**…

Practices suggested by the graph decomposition

- Pin every input to a fixed state: a release, production identifier, commit, manifest or snapshot.
- Give objects a durable identifier: a DOI where one can be minted, so the handle outlives the location.
- Give shared inputs an authoritative released version: run lists and common selections first.
- Record what the execution actually used: inputs, parameters, selections, tables and figures.
- Preserve the effective selection: not only the individual cuts, but how they were combined.
- Keep the note and the code linked: the note carries intent; the code carries behaviour.
- Review the note against the code state: not only against its citations.
- And more…