PROV-IO+: A Provenance Framework for Scientific Data on HPC Systems

Runzhou Han[1], Mai Zheng[1], Suren Byna[2], Houjun Tang[3], Bin Dong[3], Dong Dai[4]

Yong Chen[5], Dongkyun Kim[5], Joseph Hassoun[5], David Thorsley[5], Matthew Wolf[5]

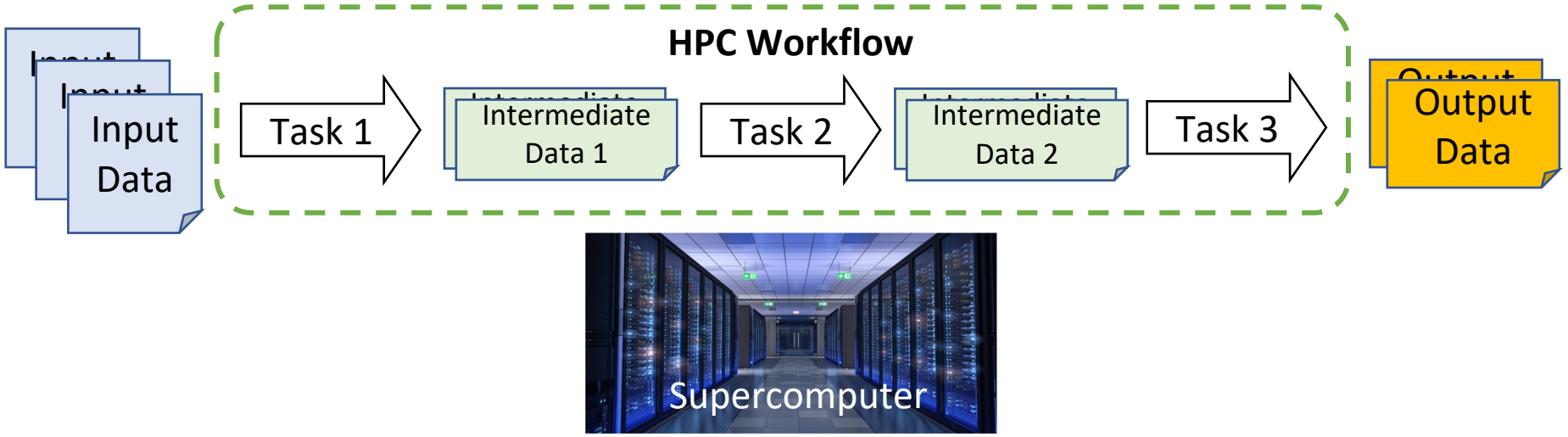
[1]*Iowa State University* [2]*The Ohio State University* [3]*Lawrence Berkeley National Laboratory*

[4]*University of North Carolina at Charlotte* [5]*Samsung*

Motivation

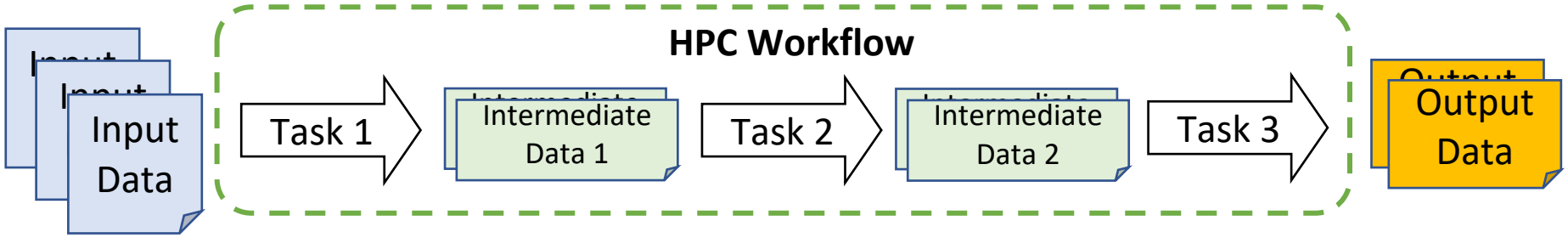
Scientists Want to Know Their Workflows Better

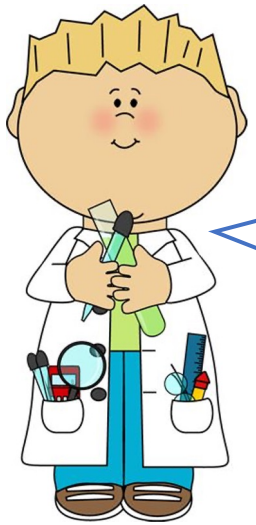
- Workflows running on HPC systems are complicated



Scientists Want to Know Their Workflows Better

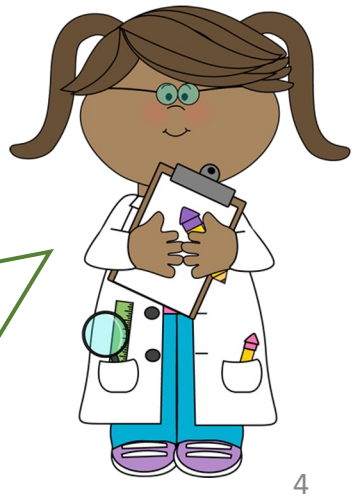
- Scientists could have a variety of questions about the workflow





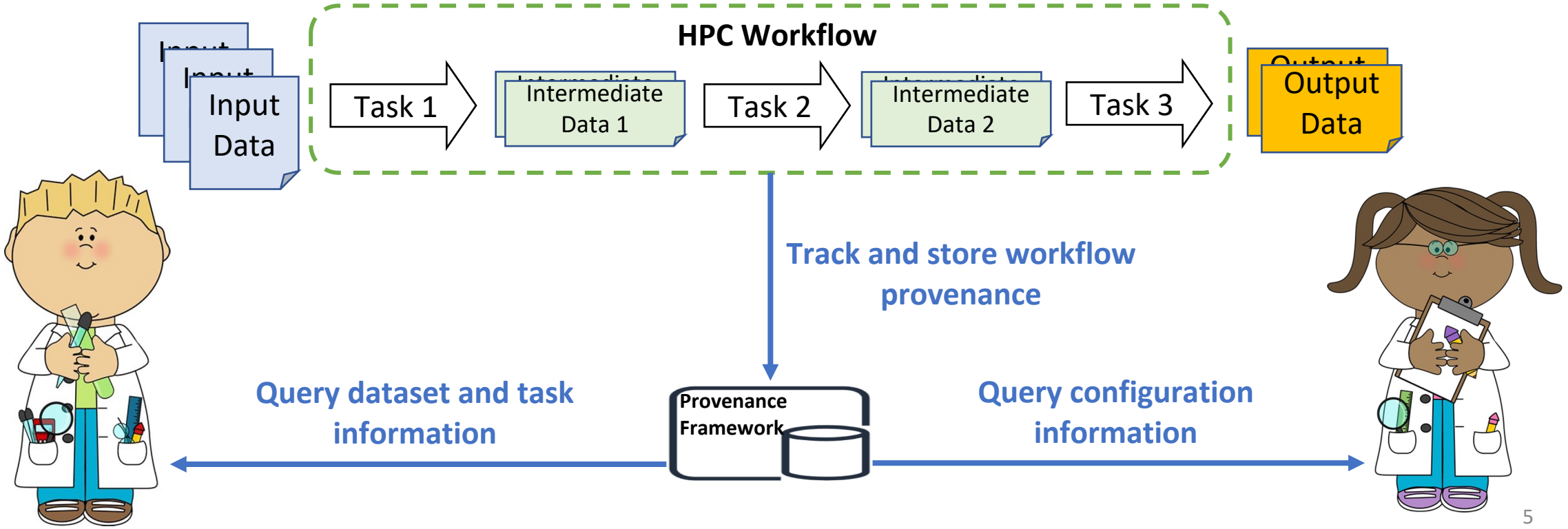
Which dataset slows down the training process? Where does the bottleneck take place? …

Which set of hyperparameters have been used? Which set of data preselection has the best training accuracy? …



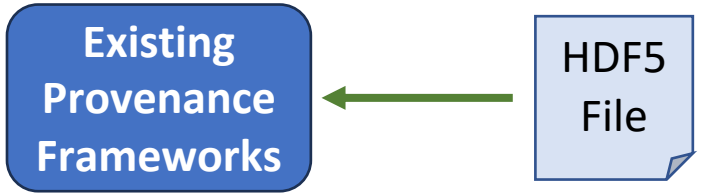
Provenance Framework Is Designed to Help

- **Provenance Frameworks** are used to collect execution metadata
  - E.g., PASS (ATC'06), PASSv2 (ATC'09), Komadu (JORS'15), ProvLake (eScience'19)



Limitations of State-of-the-art

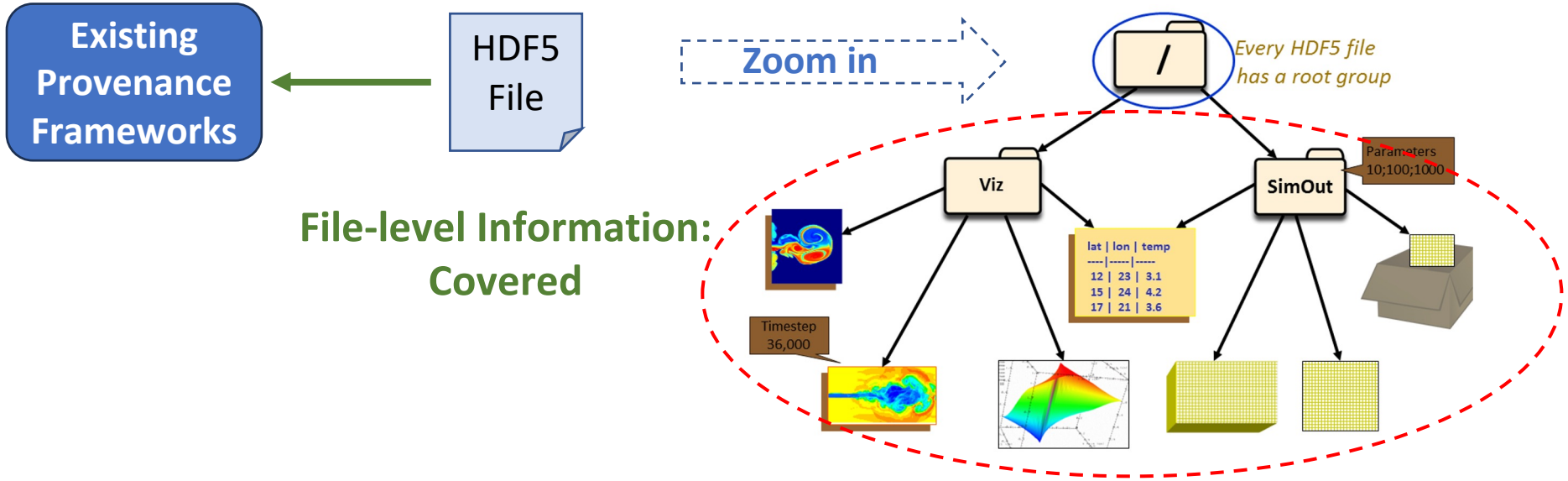
- Limitation 1: Granularity
  - Cannot cover inner hierarchies of scientific data or detailed I/O operations



File-level Information: Covered

Limitations of State-of-the-art

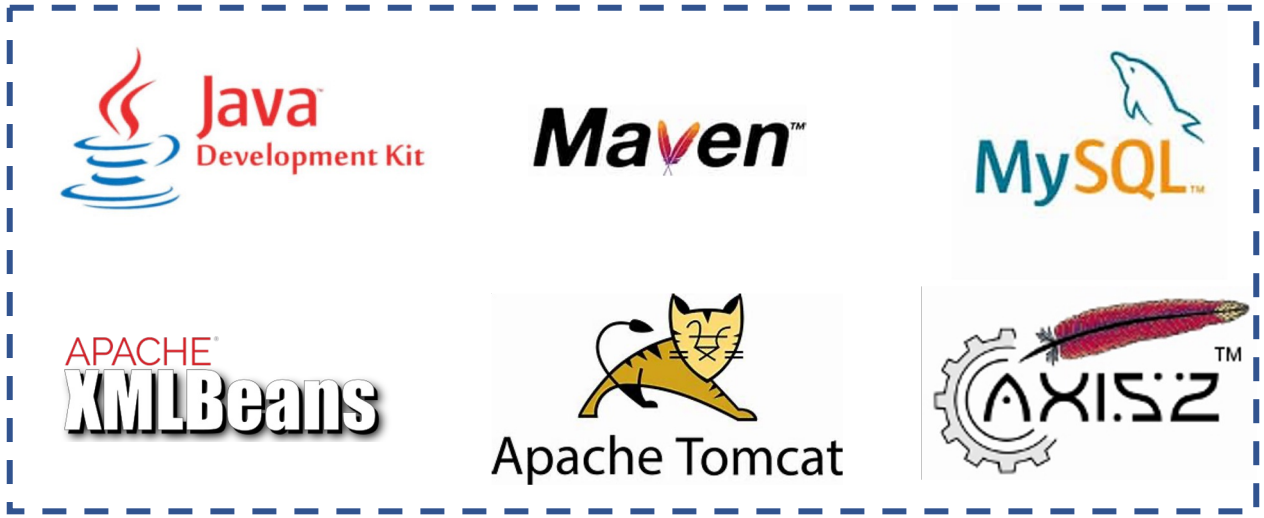
- Limitation 1: Granularity
  - Cannot cover inner hierarchies of scientific data or detailed I/O operations



*HDF5 hierarchy figure from HDF5 official website: Introduction to HDF5 (hdfgroup.org)

Limitations of State-of-the-art

- Limitation 2: Compatibility & Portability
  - Heavy dependencies on third party tools which are difficult to port (e.g., “Komadu”)



Komadu Dependencies

Limitations of State-of-the-art

- Limitation 3: Transparency
  - Scientists have to instrument workflows with specific APIs manually (e.g., “ProvLake”)

```
from provlake import ProvLake
from provlake.capture import ProvWorkflow, ProvTask, ProvCycle

...
prov = ProvLake.get_persister(workflow_name=workflow_name, managed_persistence=False)
workflow = ProvWorkflow(prov).begin()
while current_n > 1:
    with ProvCycle(prov, cycle_name=cycle_name, iteration_id=iteration_id):

        with ProvTask(prov, data_transformation_name=dt_name,
                      parent_cycle_name=cycle_name, parent_cycle_iteration=iteration_id,
                      input_args={"current_n": current_n}) as prov_task:

            result = result * current_n
            current_n = current_n - 1

            prov_task.end({"factorial_result": result})

        iteration_id += 1

print("Finished workflow")
print(result)

workflow.end()
```

Initialize a provenance instance

Insert tracking APIs at multiple hierarchies into the workflow loop

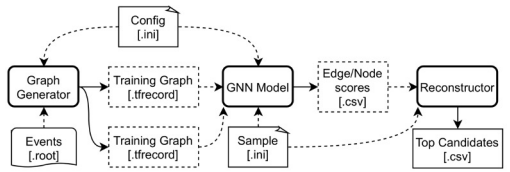
Finalize provenance instance at multiple hierarchies

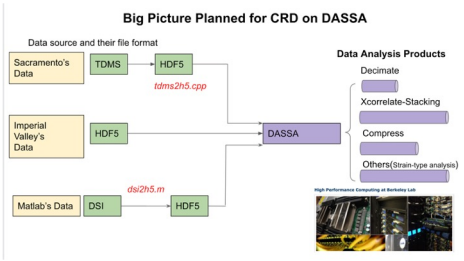
An Example of Instrumenting Workflow with ProvLake

Approach

Survey on Practical Needs of Domain Scientists

- Discussed with four research teams from different domains
  - Learn about their workflow & provenance needs





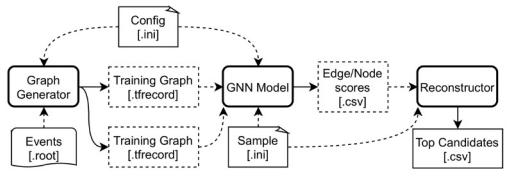
NVIDIA/**Megatron-LM**

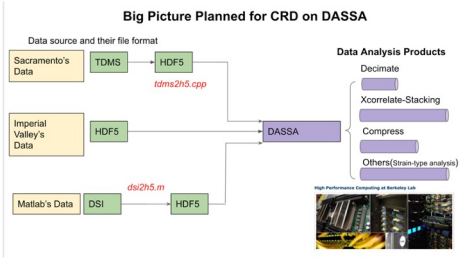
Ongoing research training transformer models at scale

Name	Top Reco	DASSA	H5Bench	Megatron-LM
Domain	GNN for physical emulation	acoustic sensing	synthetic, performance benchmarking	large language model

Survey on Practical Needs of Domain Scientists

- Discussed with four research teams from different domains
  - Learn about their workflow & provenance needs





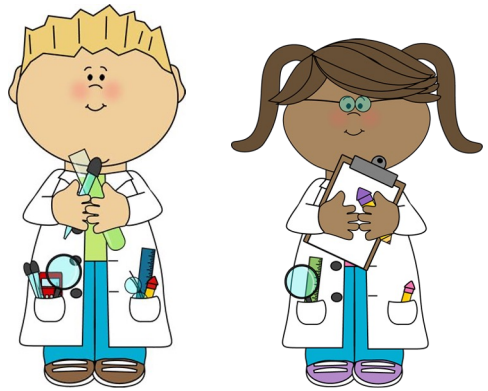
NVIDIA/**Megatron-LM**

Ongoing research training transformer models at scale

Name	Top Reco	DASSA	H5Bench	Megatron-LM
Domain	GNN for physical emulation	acoustic sensing	synthetic, performance benchmarking	large language model
Provenance need	**metadata version control**	**lineage of data products**	**I/O stats & performance bottleneck**	**checkpoint-config. consistency**

Survey on Practical Needs of Domain Scientists

- Summary of provenance needs



We want to know …

Data information

End-to-end data information including each intermediate state

Task information

Information of tasks at multiple granularities, e.g., program, function call

Configuration information

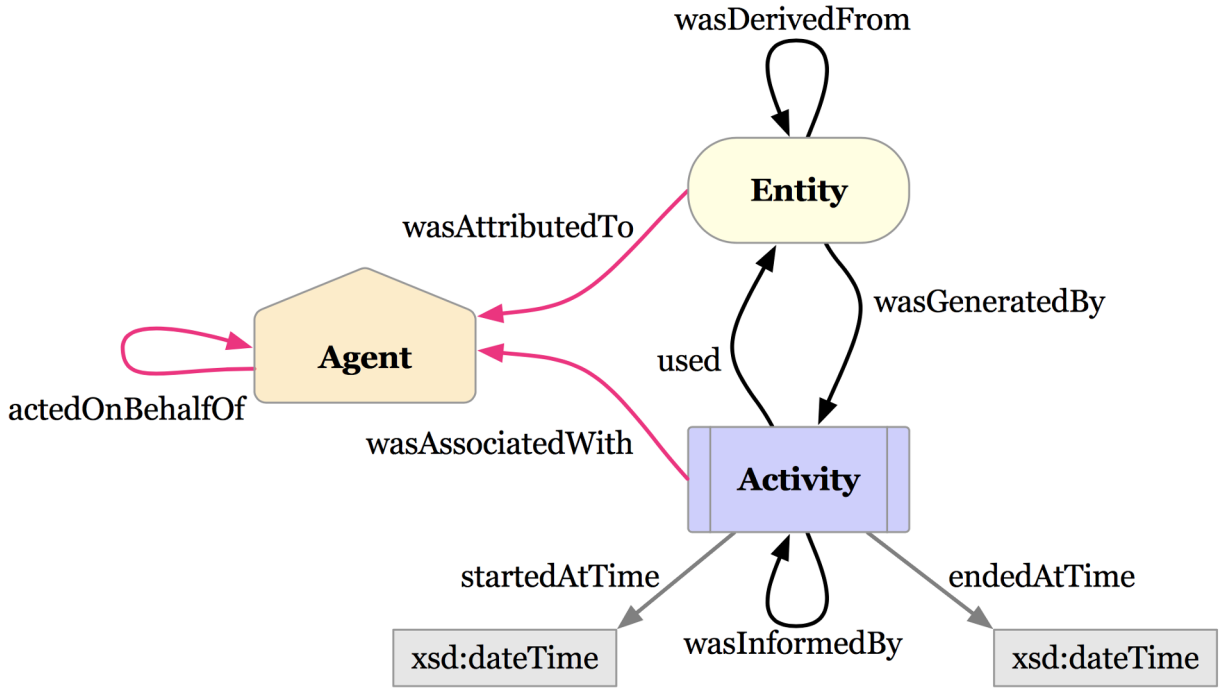
Workflow configurable parameters

Relation information

Relations between above information

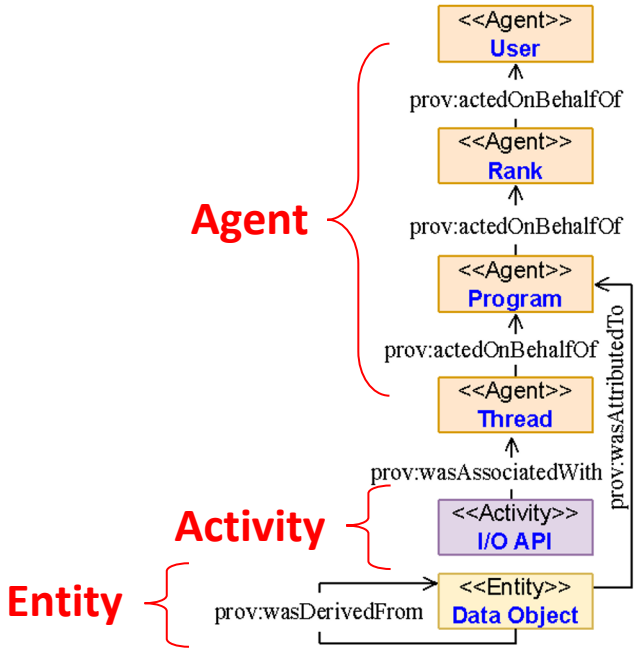
Design PROV-IO+ Model Based on Needs

- Derived from W3C provenance data model (PROV-DM)
  - Widely adopted by provenance frameworks (e.g., Komadu, ProvLake)
  - Provides a mapping to Resource Description Framework (RDF) triples



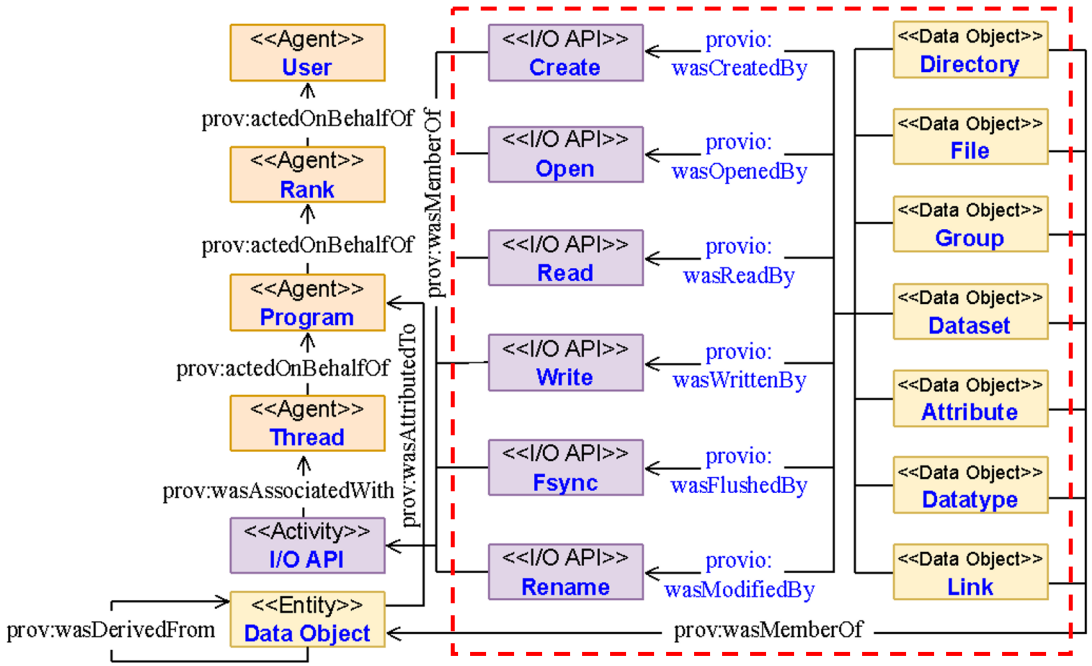
Design PROV-IO+ Model Based on Needs

- PROV-IO+ model: a PROV-DM-compliant data model
  - Interoperable with other PROV-DM-compliant data models (e.g., ProvLake/PROV-ML, Komadu model)



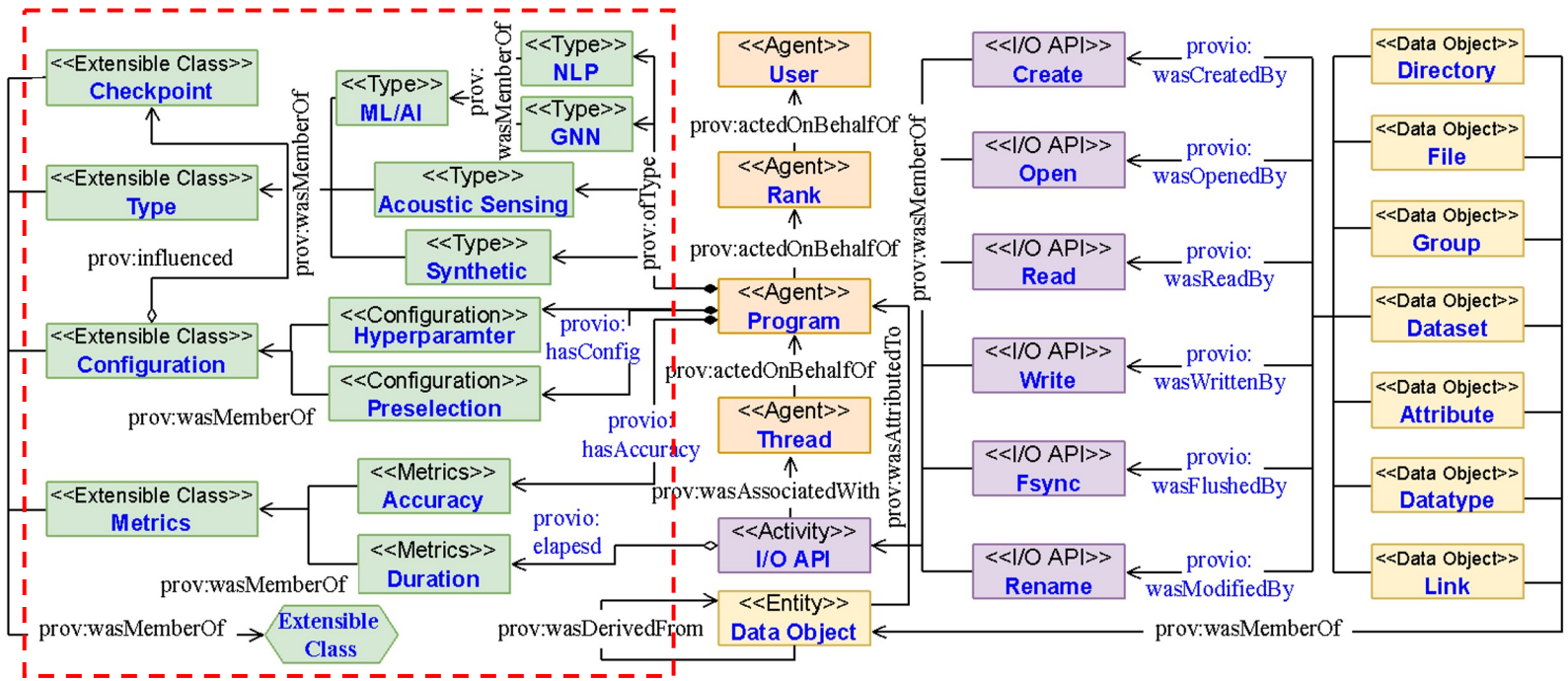
Design PROV-IO+ Model Based on Needs

- PROV-IO+ model: a PROV-DM-compliant data model
  - Covers most of the metadata concepts & I/O operations in popular HPC I/O libraries (e.g., HDF5, POSIX Syscall C library)



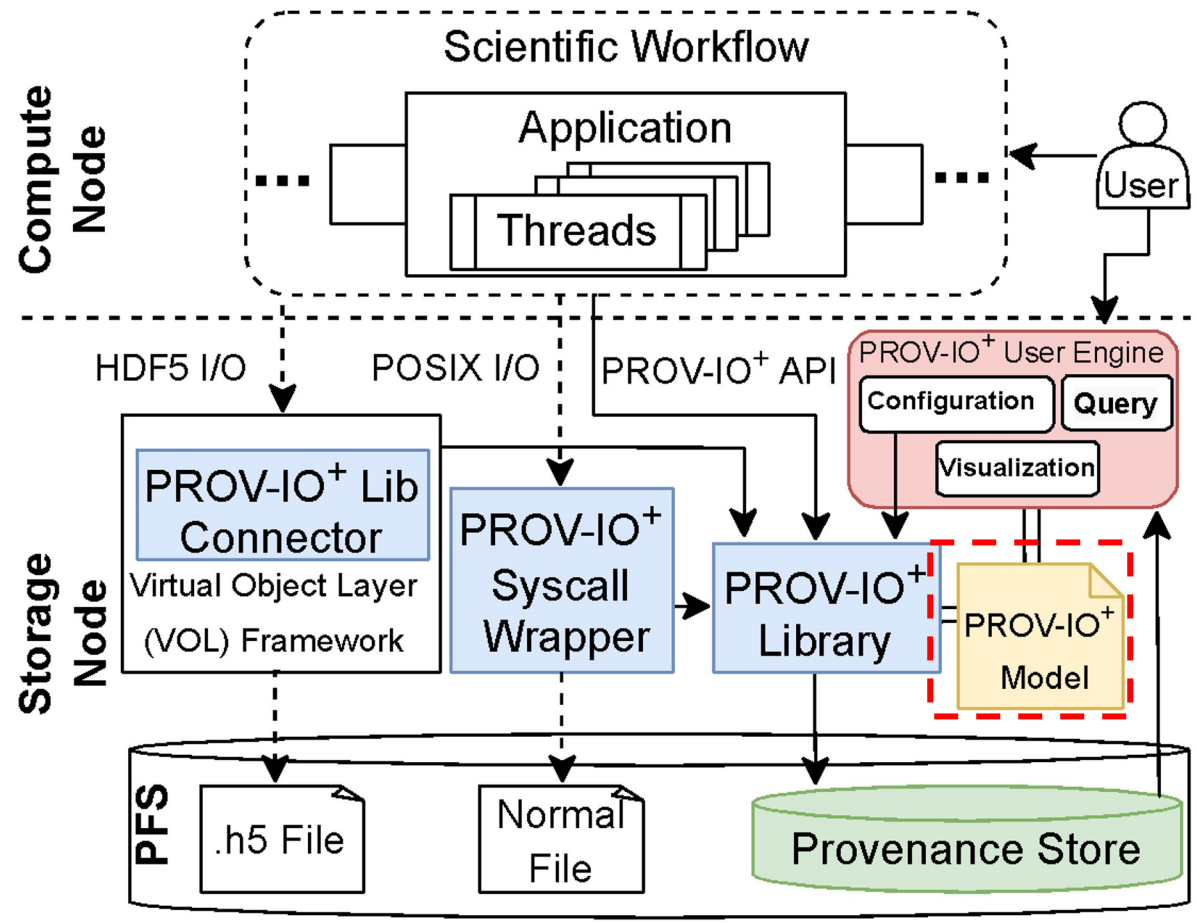
Design PROV-IO+ Model Based on Needs

- PROV-IO+ model: a PROV-DM-compliant data model
  - Provides an interface for users to extend the PROV-IO+ model with new concepts/relations per their needs



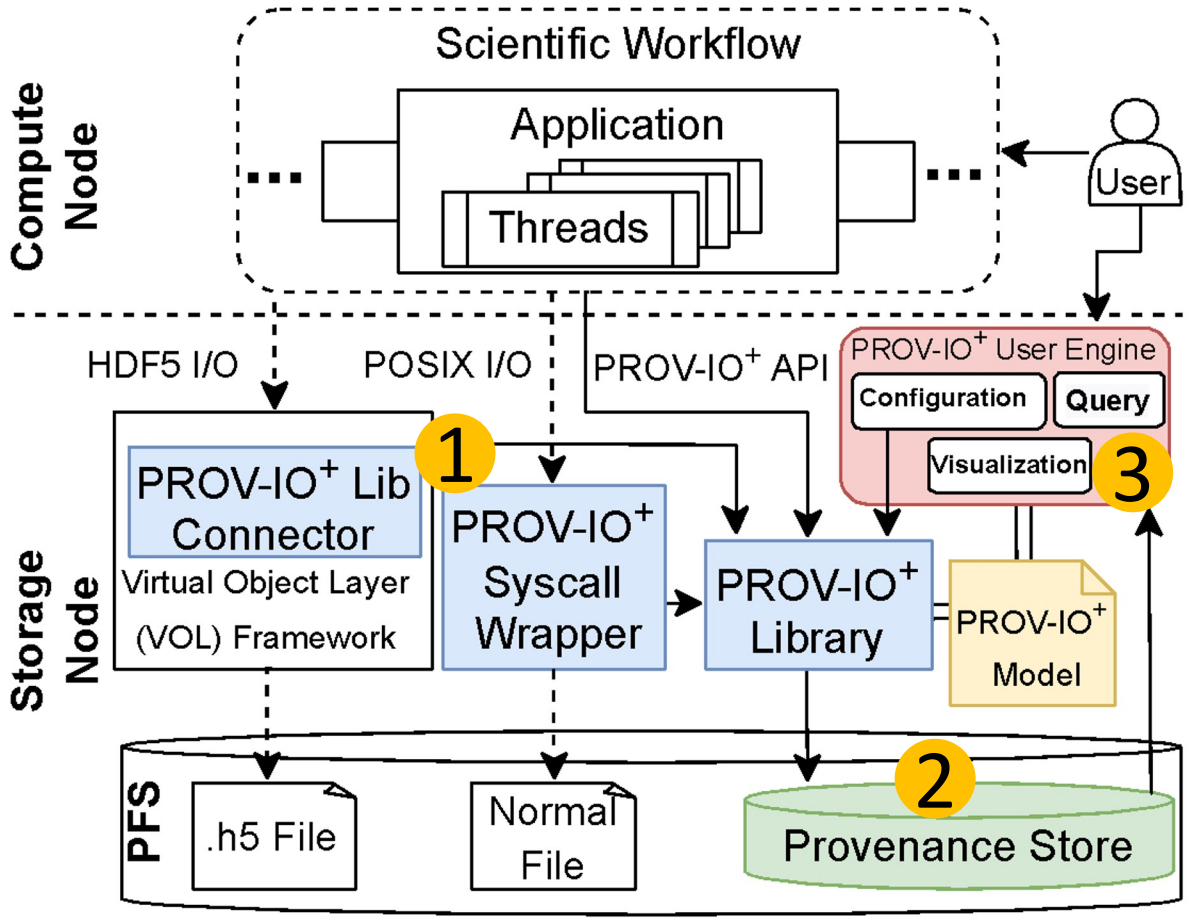
PROV-IO+ Framework

- Overview
  - **PROV-IO+ model**



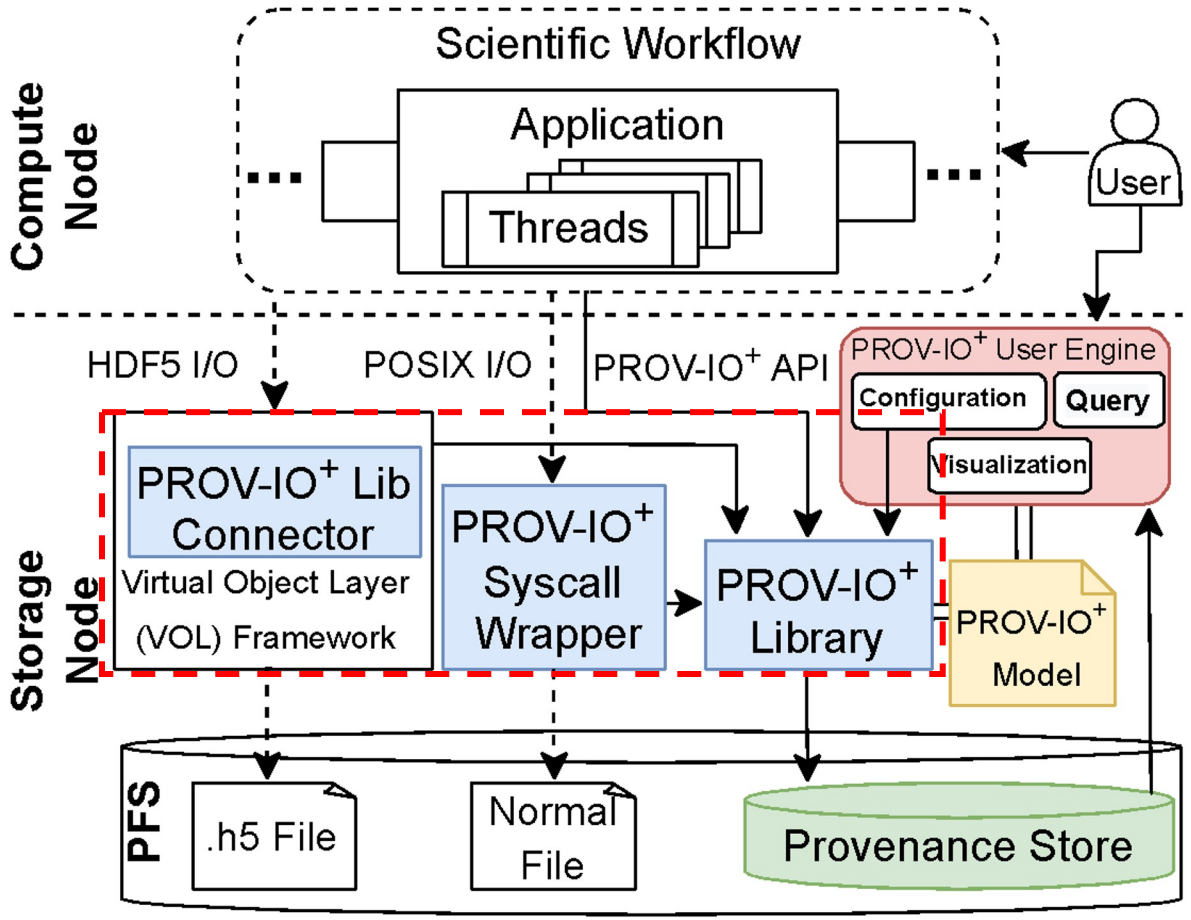
PROV-IO+ Framework

- Overview
  - PROV-IO+ model
  - **Three main components based on PROV-IO+ model**
    1. Tracking (blue)
    2. Store (Green)
    3. User engine (Red)



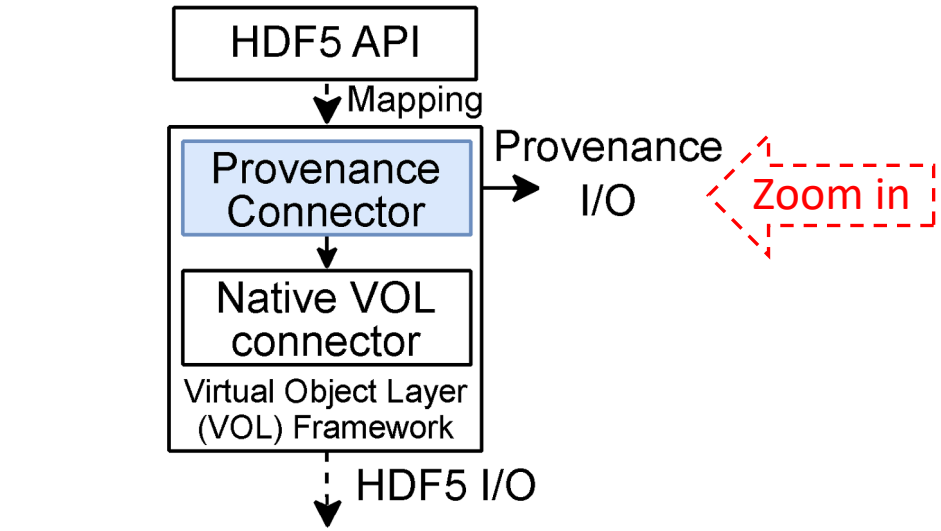
PROV-IO+ Framework

- Provenance tracker (blue)
  - Track I/O operations transparently by intercepting library I/O



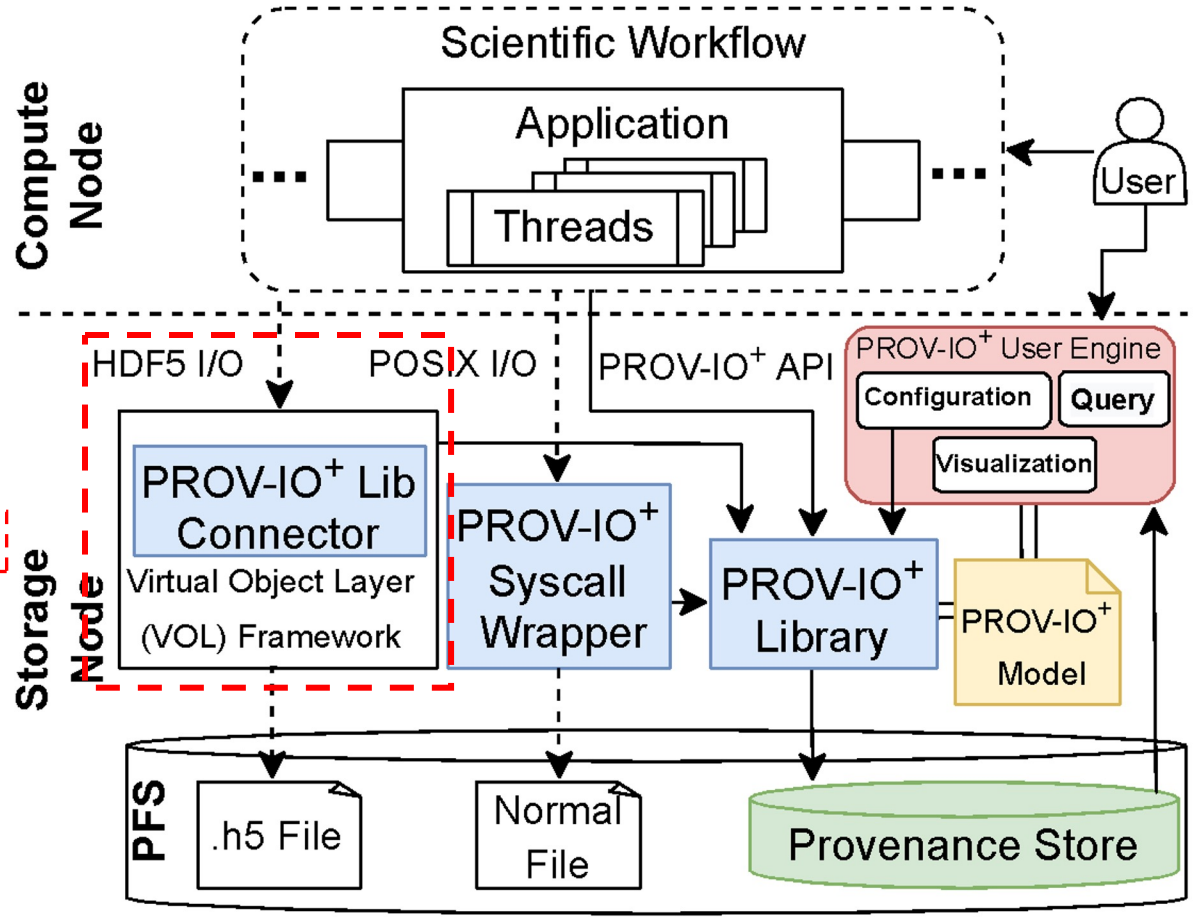
PROV-IO+ Framework

- Provenance tracker (blue)
  - Track I/O operations transparently by intercepting library I/O



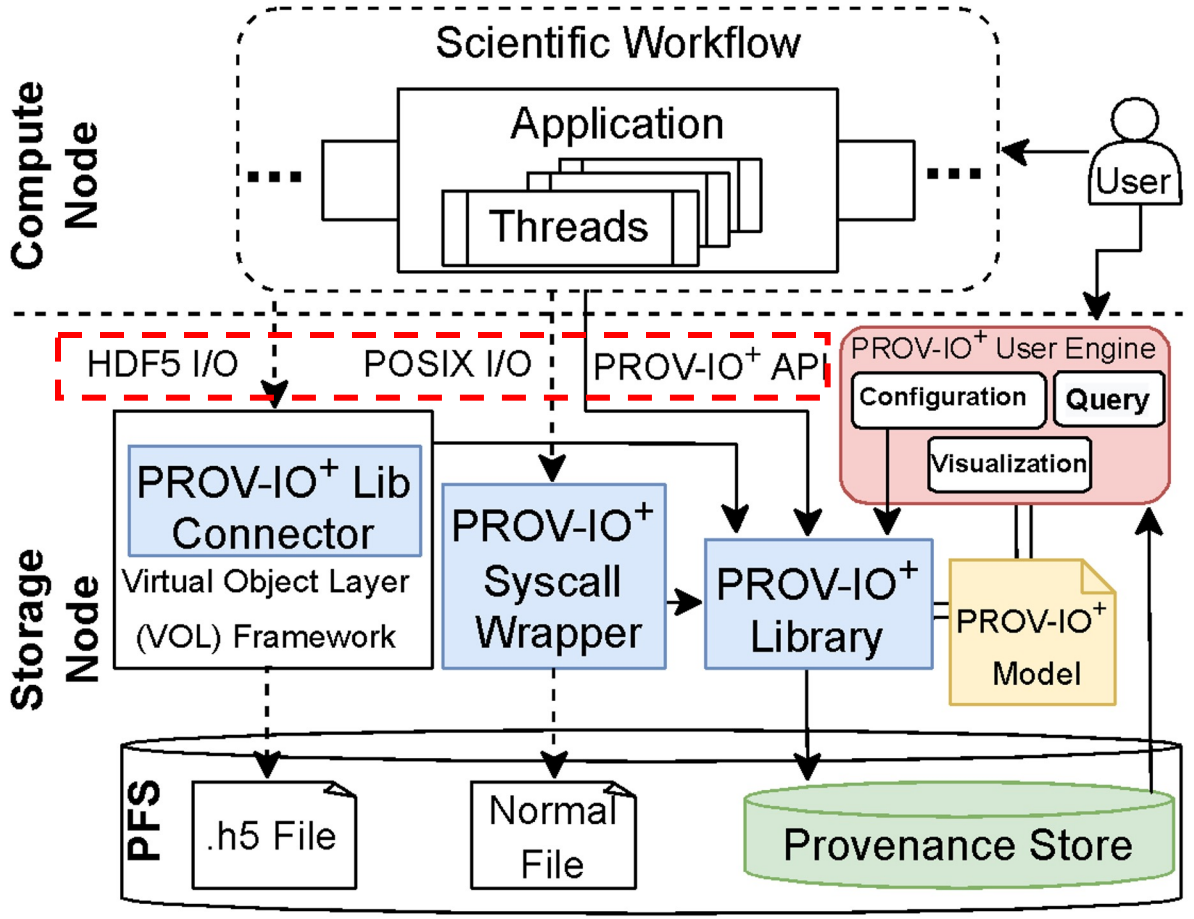
HDF5 Provenance VOL connector

Zoom in



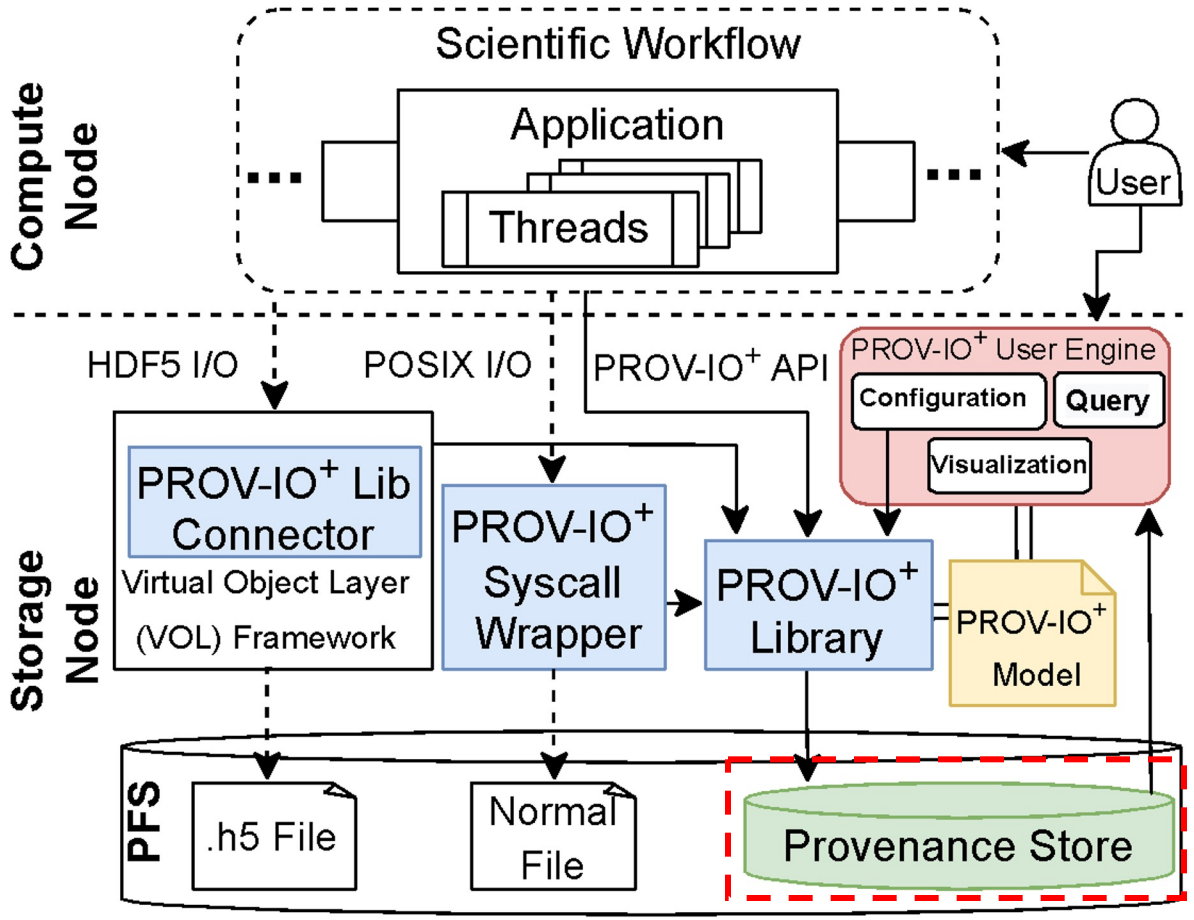
PROV-IO+ Framework

- Provenance tracker (blue)
  - Track I/O operations transparently by intercepting library I/O
  - Support for popular I/O libraries use by HPC workflows
  - Provide a Python interface for manually instrumentation



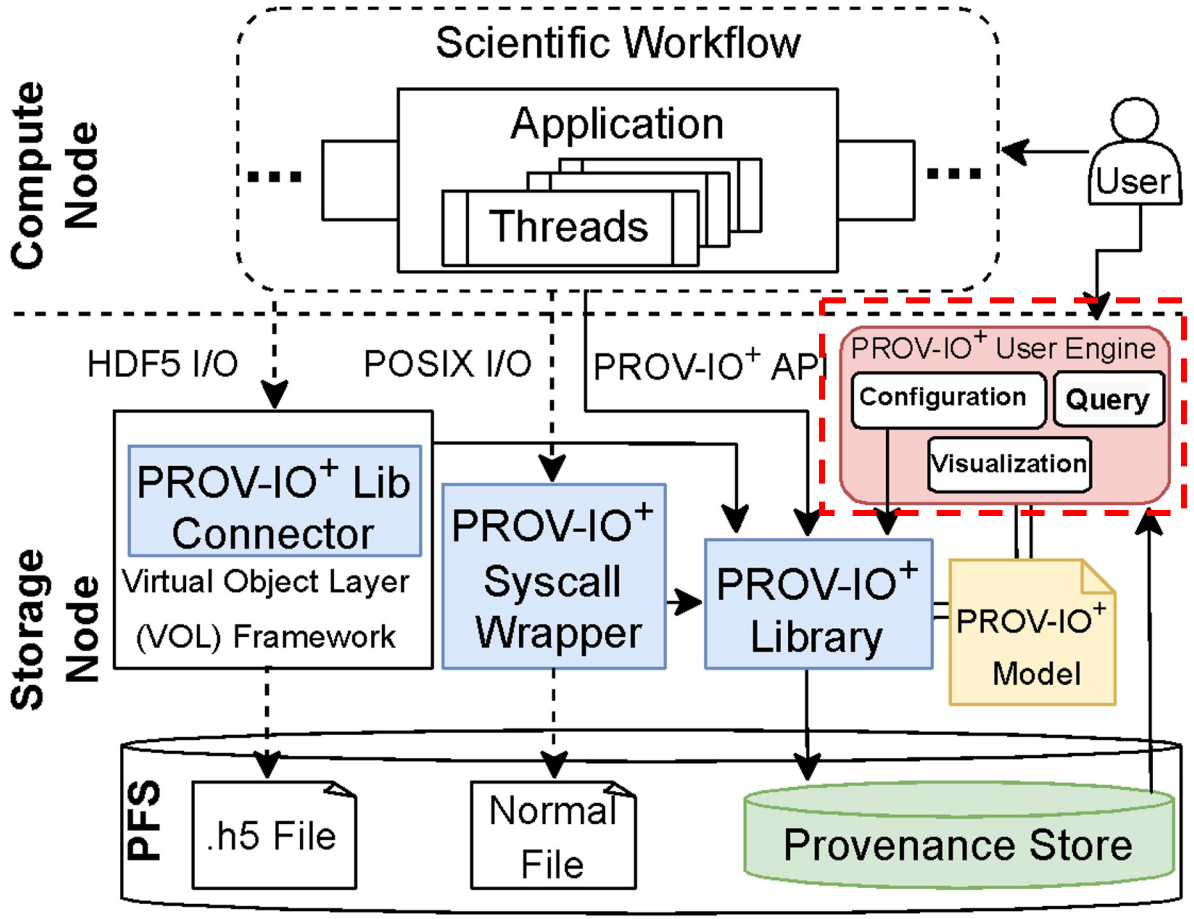
PROV-IO+ Framework

- Provenance store (green)
  - Serialize provenance as RDF
  - Avoid concurrent provenance serialization by having each thread write to its own file
  - Consolidate provenance files offline



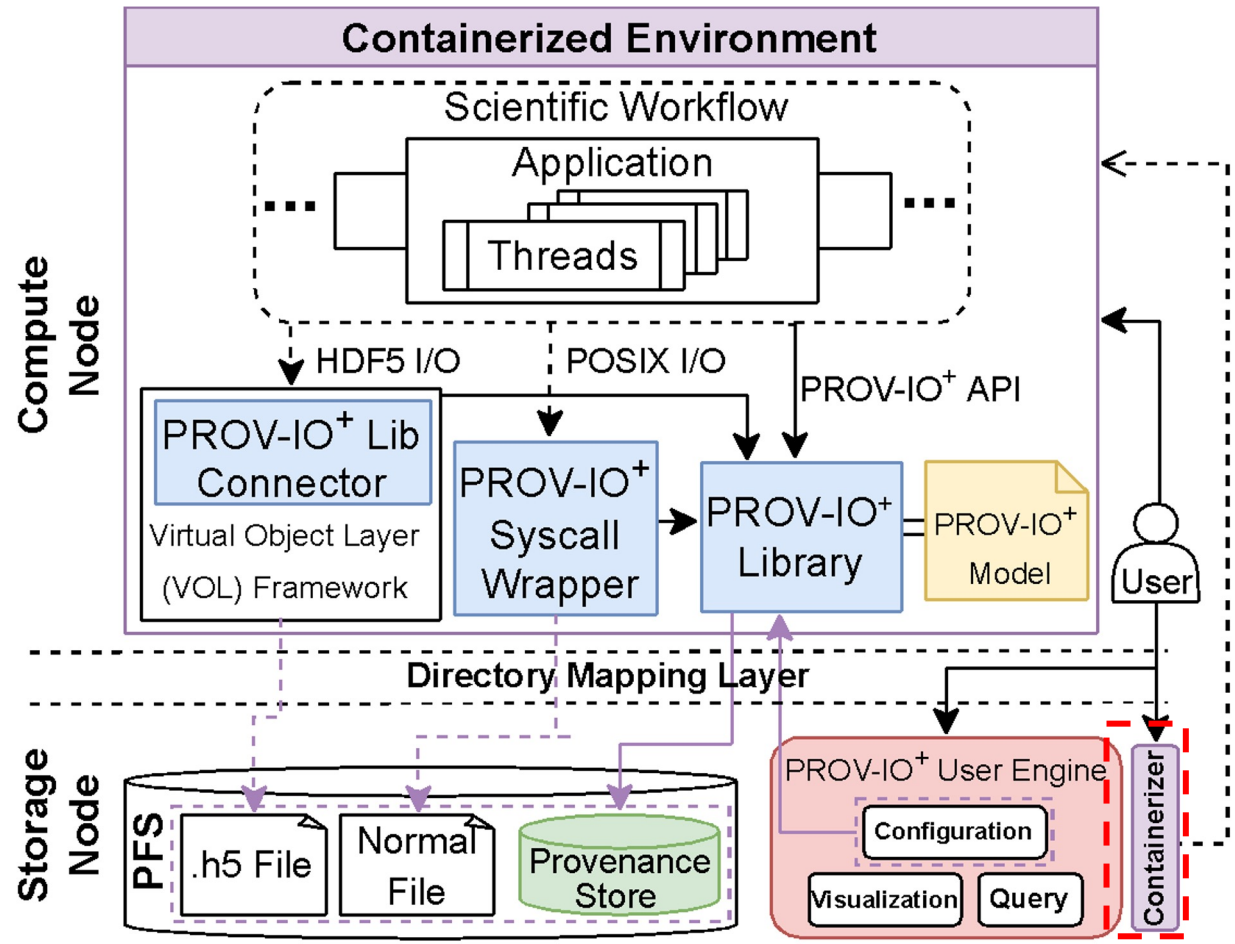
PROV-IO+ Framework

- User engine (red)
  - A configuration interface for user
  - Provenance query with SPARQL
  - Provenance visualization



PROV-IO+ Framework

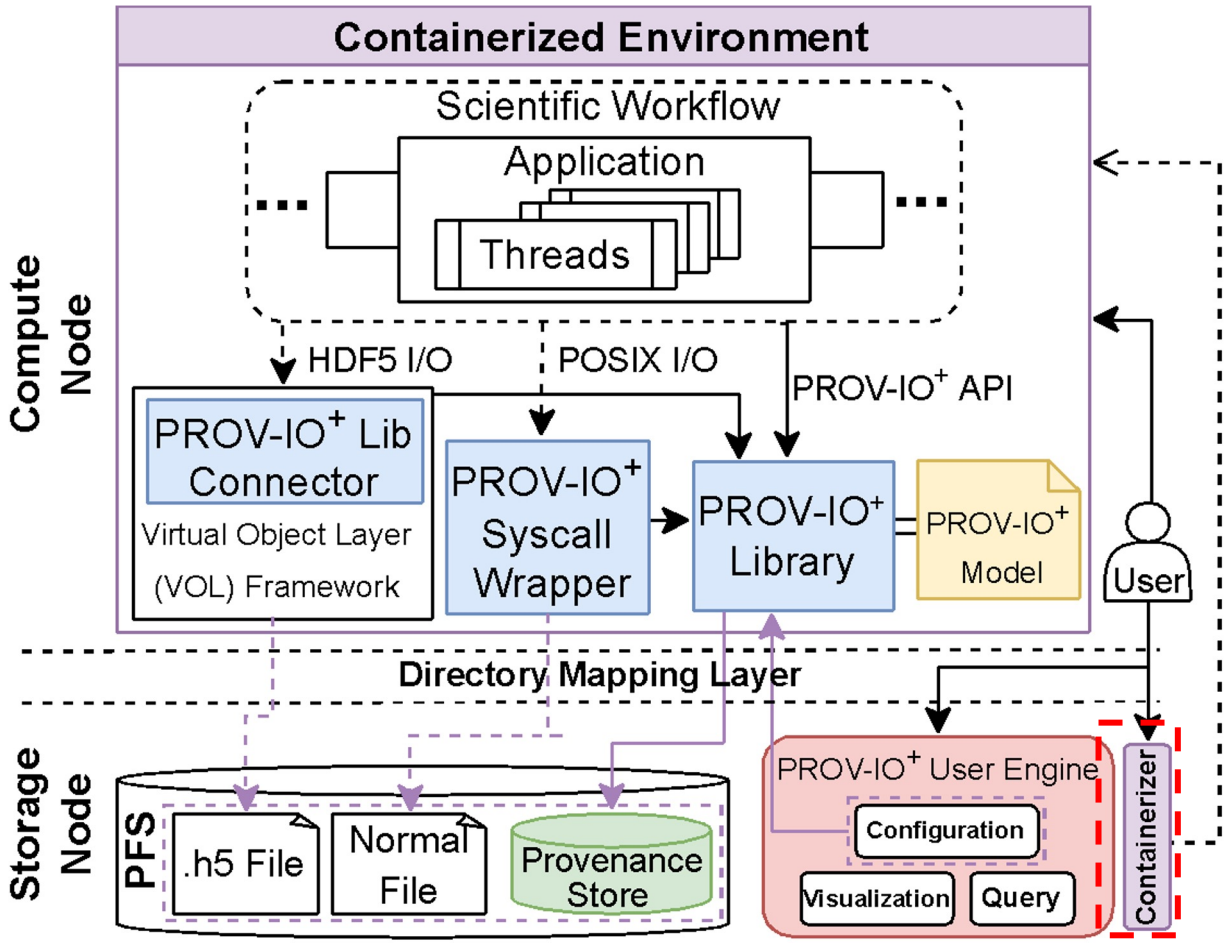
- Overview
  - PROV-IO+ model
  - Three main components based on PROV-IO+ model
  - **Support for containerized environment**
    - Newer HPC systems may have containerized job management (e.g., Singularity)



PROV-IO+ Framework

- Overview
  - PROV-IO+ model
  - Three main components based on PROV-IO+ model
  - **Support for containerized environment**
    - Newer HPC systems may have containerized job management (e.g., Singularity)

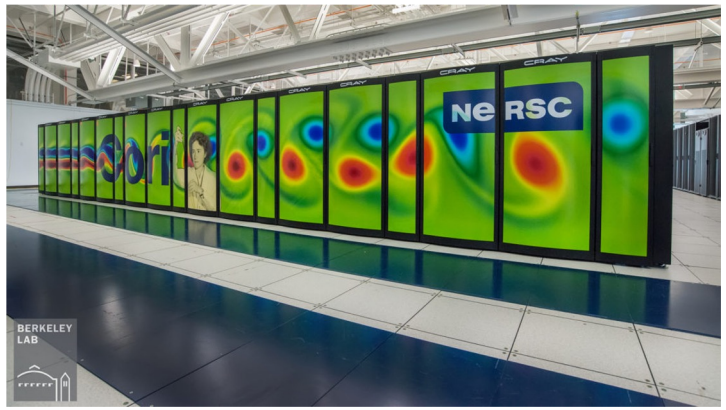
For more design/implementation details, please refer to PROV-IO+ paper at: https://arxiv.org/abs/2308.00891



Evaluation

Experimental Methodology

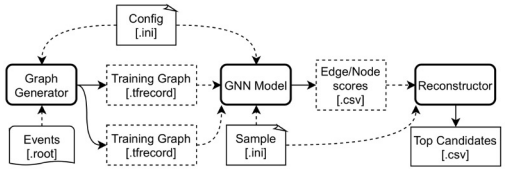
- Platforms
  - Cori @LBNL (traditional workflows)
    - Up to 64 nodes (4096 MPI ranks)
    - Measured workflows
      - Top Reco (GNN for physical emulation)
      - DASSA (acoustic sensing)
      - H5Bench (synthetic)
  - Samsung supercomputer (containerized workflows)
    - 8 A100 GPUs (due to strict quota)
    - Measured workflow
      - Megatron-LM (large language model)

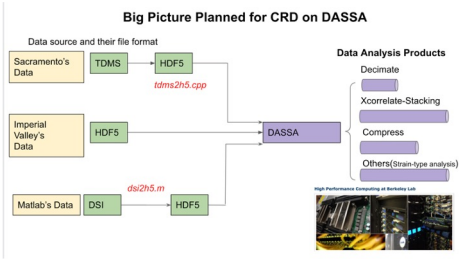


*Picture from LBNL website

Experimental Methodology

- Information tracked for each workflow





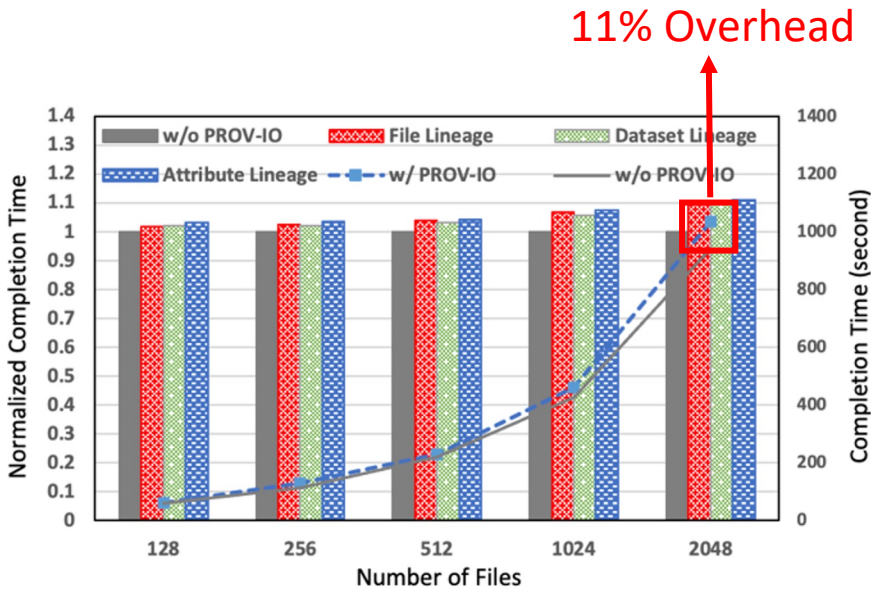
NVIDIA/**Megatron-LM**

Ongoing research training transformer models at scale

Name	Top Reco	DASSA	H5Bench	Megatron-LM
Domain	GNN for physical emulation	acoustic sensing	synthetic, performance benchmarking	large language model
Provenance need	metadata version control	lineage of data products	I/0 stats & performance bottleneck	checkpoint-config. consistency
Information tracked	**1. hyperparameter 2. data preselection 3. training accuracy**	**1. program name 2. I/O API (HDF5) 3. file/dataset/attr**	**1. I/O API (HDF5) 2. I/O API duration 3. user/rank/program/file**	**1. checkpoint info 2. training loss 3. model configuration**

Tracking Overhead

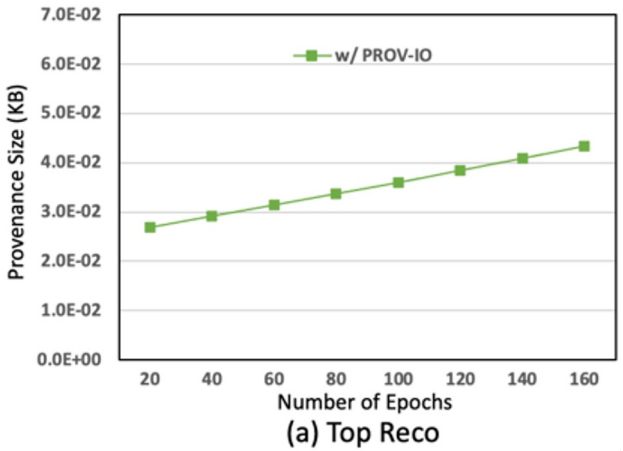
- Over all experiments, tracking overhead is ***11% at maximum***
- More than ***97%*** of the experiments has overhead ***less than 3%***



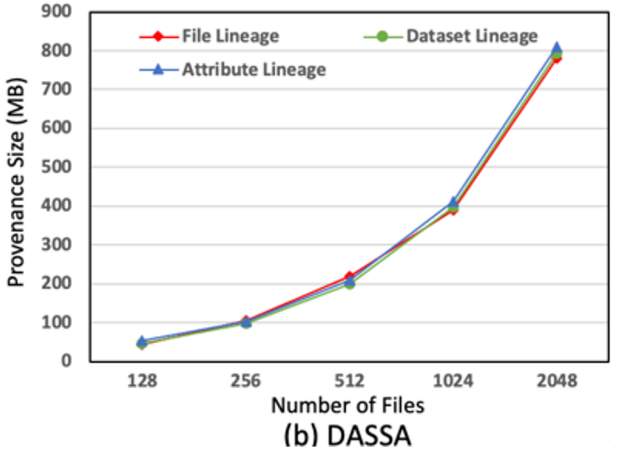
Tracking overhead on DASSA workflow

Storage Overhead

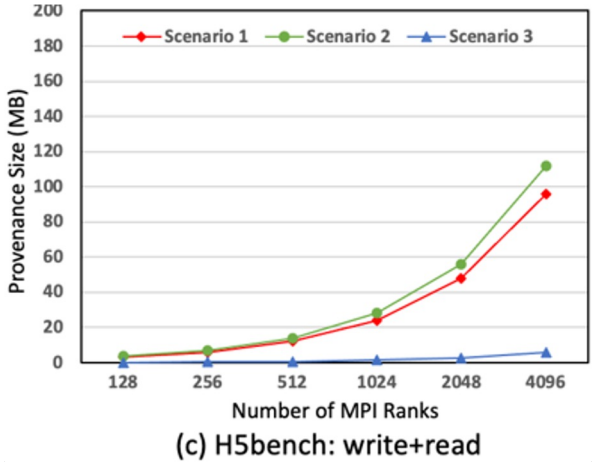
- Provenance size ***increases linearly*** with experimental scale



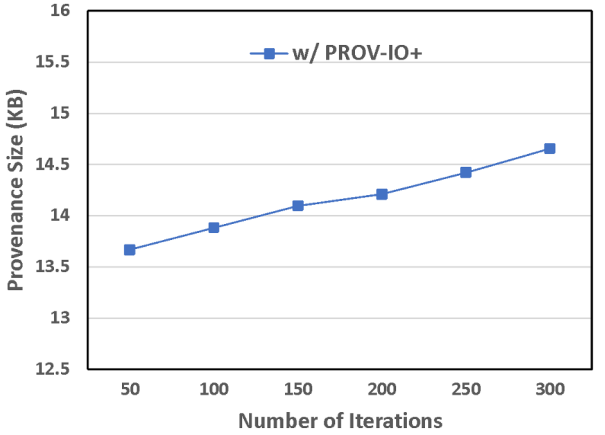
(a) Top Reco



(b) DASSA



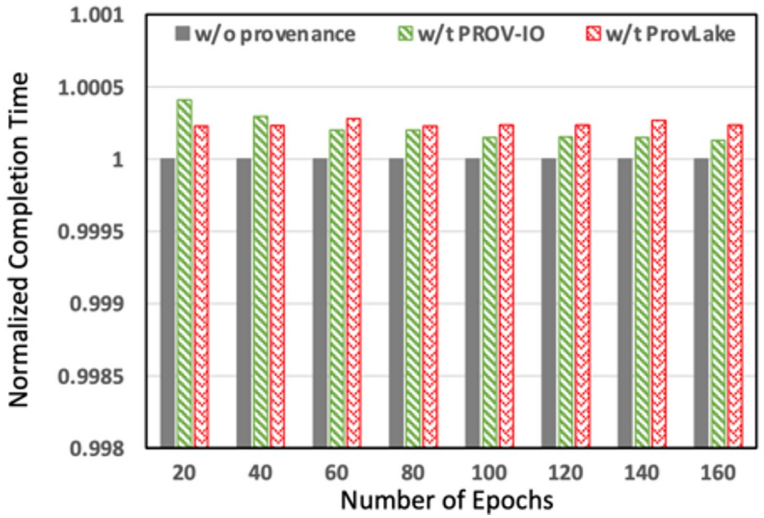
(c) H5bench: write+read



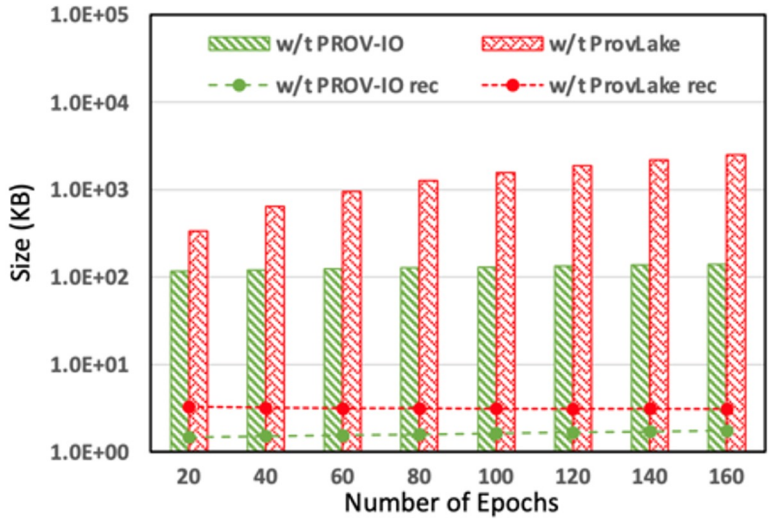
(d) Megatron-LM

Comparison with IBM ProvLake

- PROV-IO+ has ***lower tracking overhead*** in experiments ***with more training epochs***
- PROV-IO+ always has ***less storage overhead***



(a) Tracking overhead

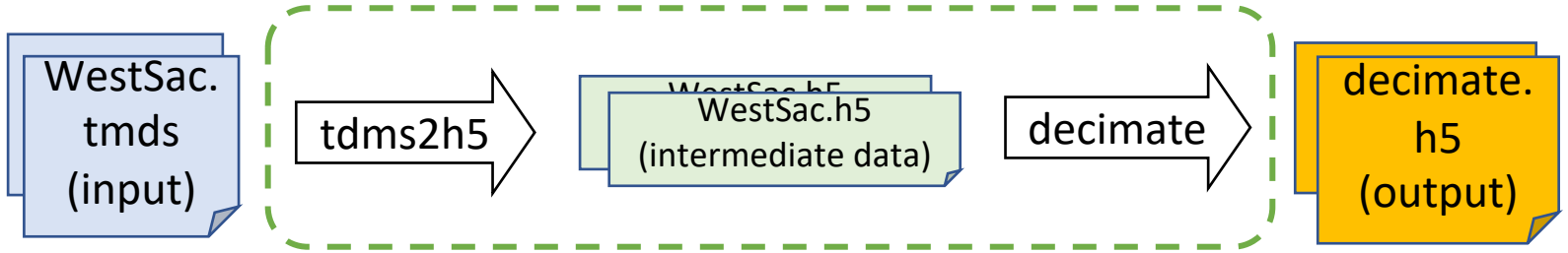


(b) Storage overhead

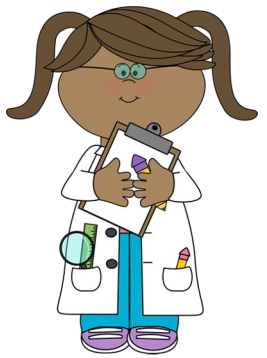
Comparison on Top Reco Workflow

Provenance Query & Visualization

- Data lineage backward tracing example with DASSA

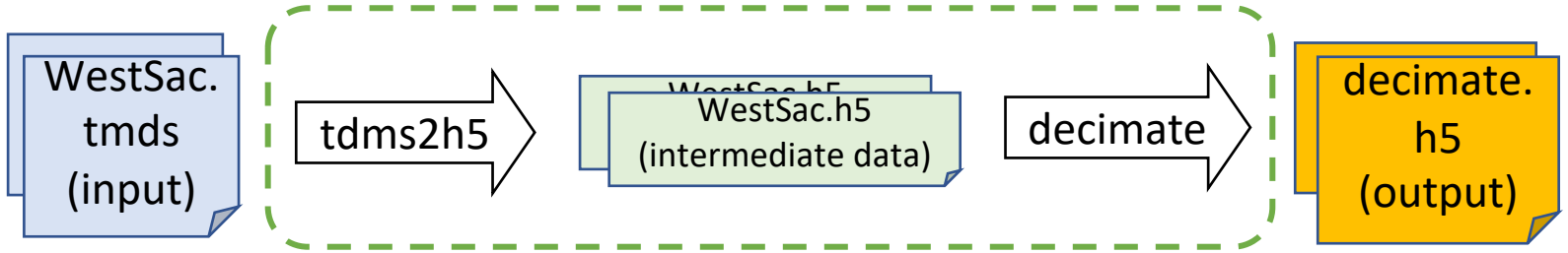


What's the origin of the output data decimate.h5?

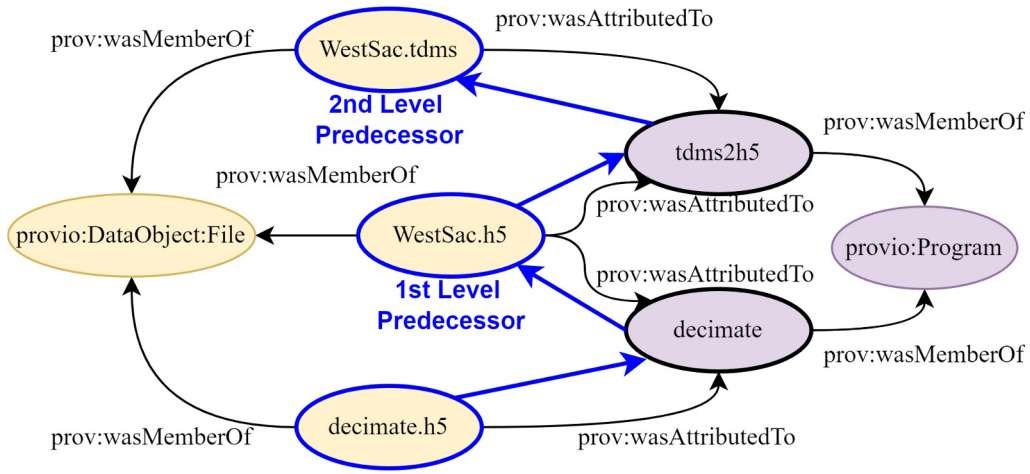


Provenance Query & Visualization

- Data lineage backward tracing example with DASSA



Visualized Provenance



Example SPARQL query to locate 1st level predecessor:

Q1: Decimate.h5 prov:wasAttributedTo ?

Q2: ? prov:wasAttributedTo decimate
provio:wasReadBy H5Fopen

2nd level predecessor can be obtained with similar query.

Conclusion & Future Work

- Conclusion
  - Identified domain scientists' real provenance needs
  - Built PROV-IO$^+$ framework under the guidance of PROV-IO$^+$ model
  - Evaluated PROV-IO$^+$ framework on two HPC systems
    - PROV-IO$^+$ can address domain scientists' concerns effectively & efficiently
- Future work
  - More efficient provenance post processing
  - Advanced query API to help users analyze workflows more efficiently

Conclusion & Future Work

- Conclusion
  - Identified domain scientists' real provenance needs
  - Built PROV-IO+ framework under the guidance of PROV-IO+ model
  - Evaluated PROV-IO+ framework on two HPC systems
    - PROV-IO+ can address domain scientists' concerns effectively & efficiently
- Future work
  - More efficient provenance post processing
  - Advanced query API to help users analyze workflows more efficiently

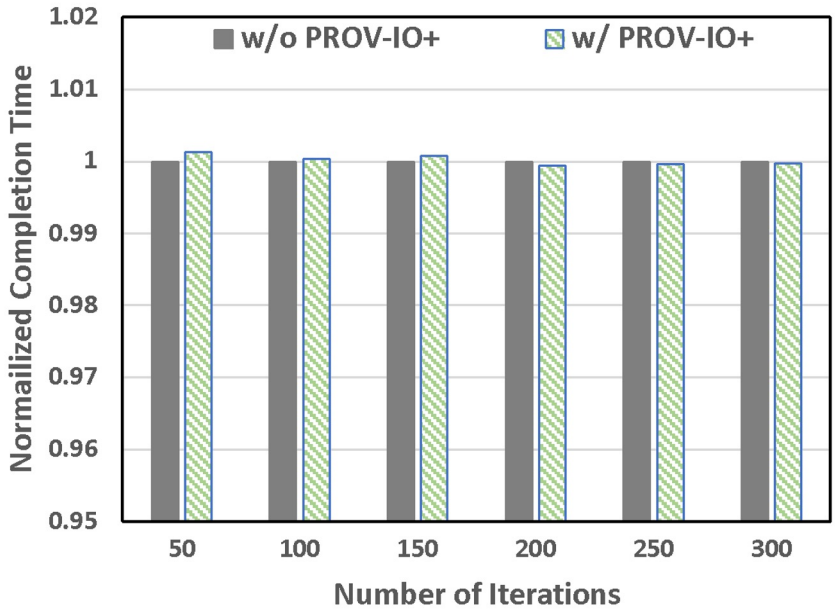
Source code is available at:
https://github.com/data-storage-lab/prov-io
Docker image is available at:
https://hub.docker.com/repository/docker/rzhan/prov-io/general

Thank You & Questions?

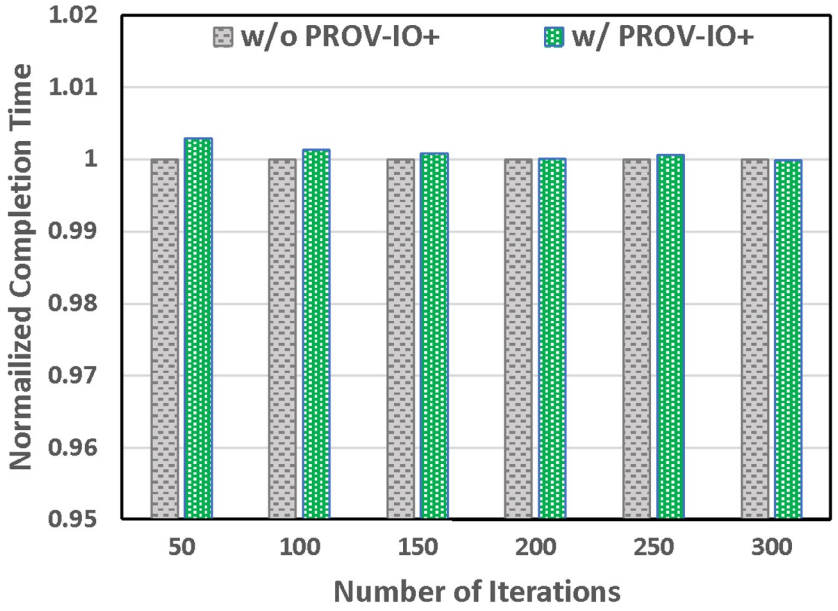
Backup Slides

Containerization Overhead

- ***Negligible containerization overhead*** observed in Megatron-LM use case on Samsung Supercomputer



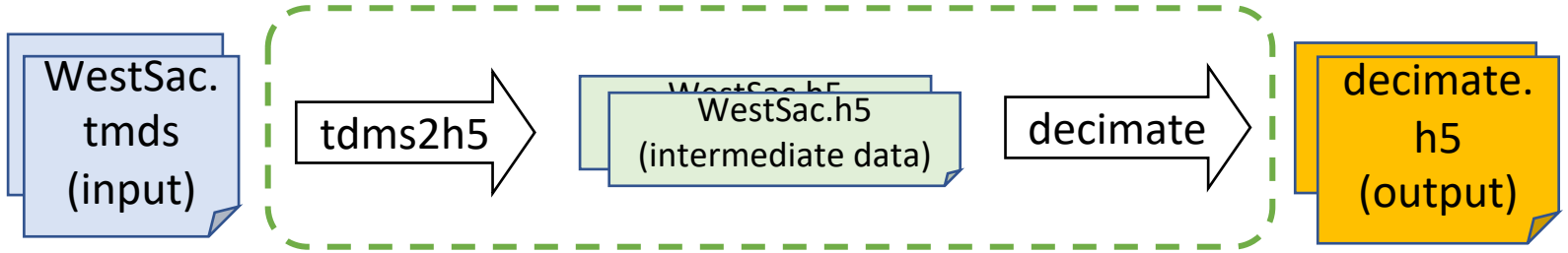
(a) Non-containerized workflow



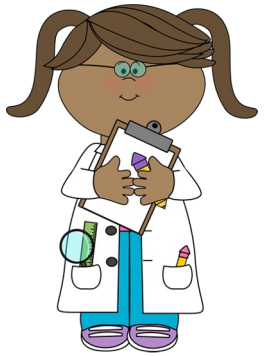
(b) Containerized workflow

How to Query Provenance

- Data lineage backward tracing example with DASSA

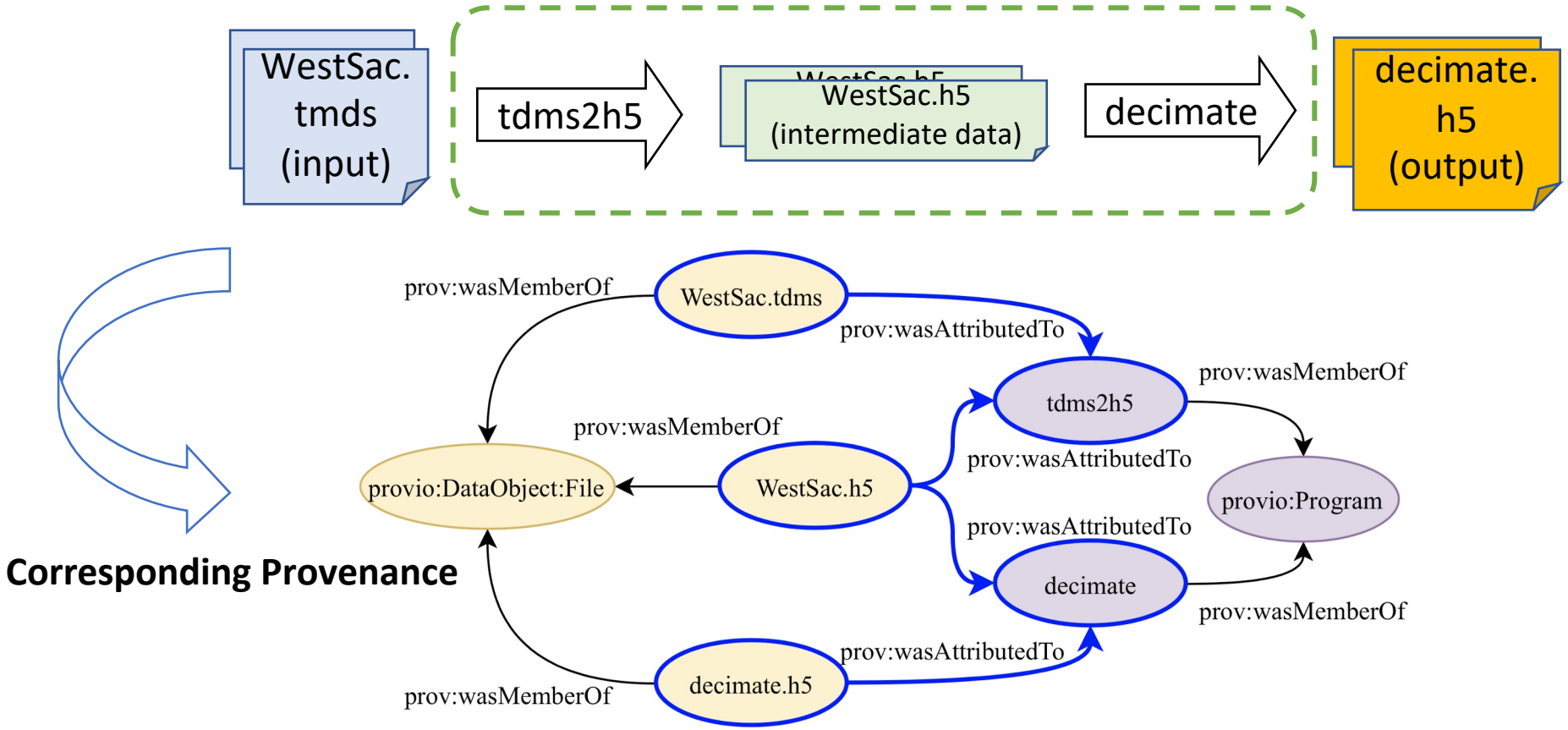


What's the origin of the output data decimate.h5?



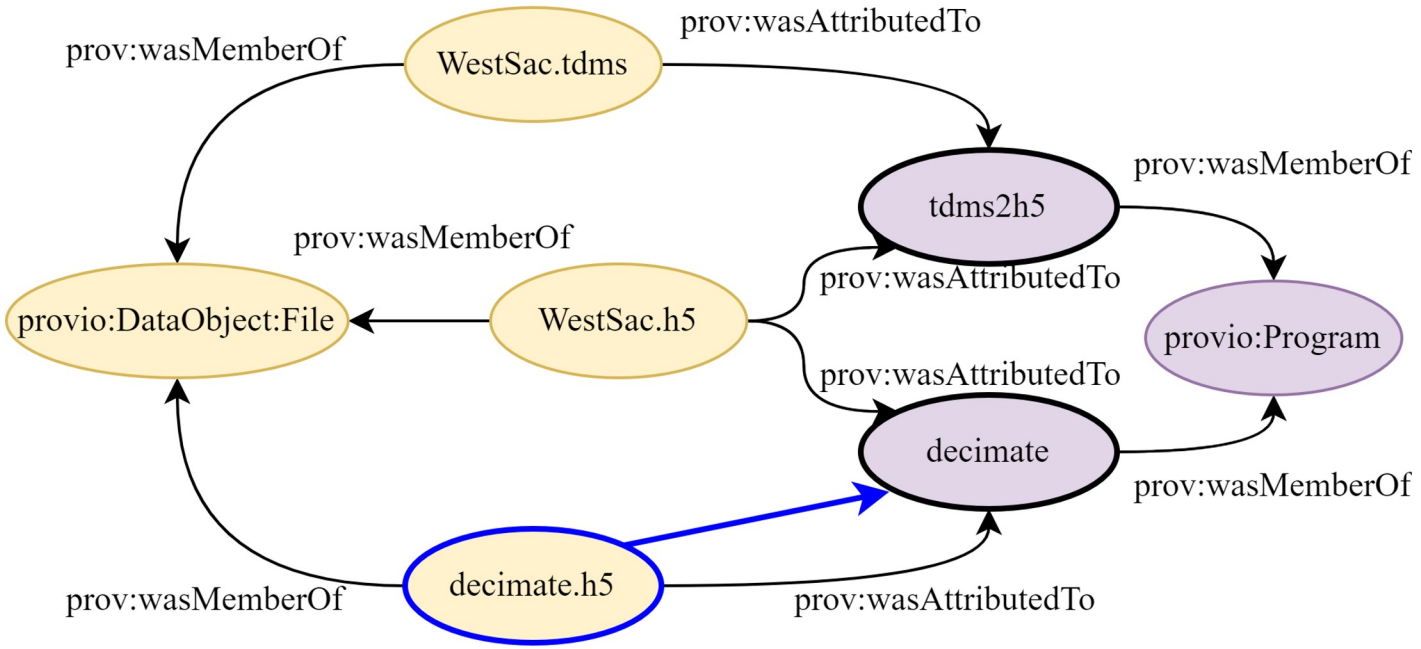
How to Query Provenance

- Data lineage backward tracing example with DASSA



How to Query Provenance

- Data lineage backward tracing example with DASSA



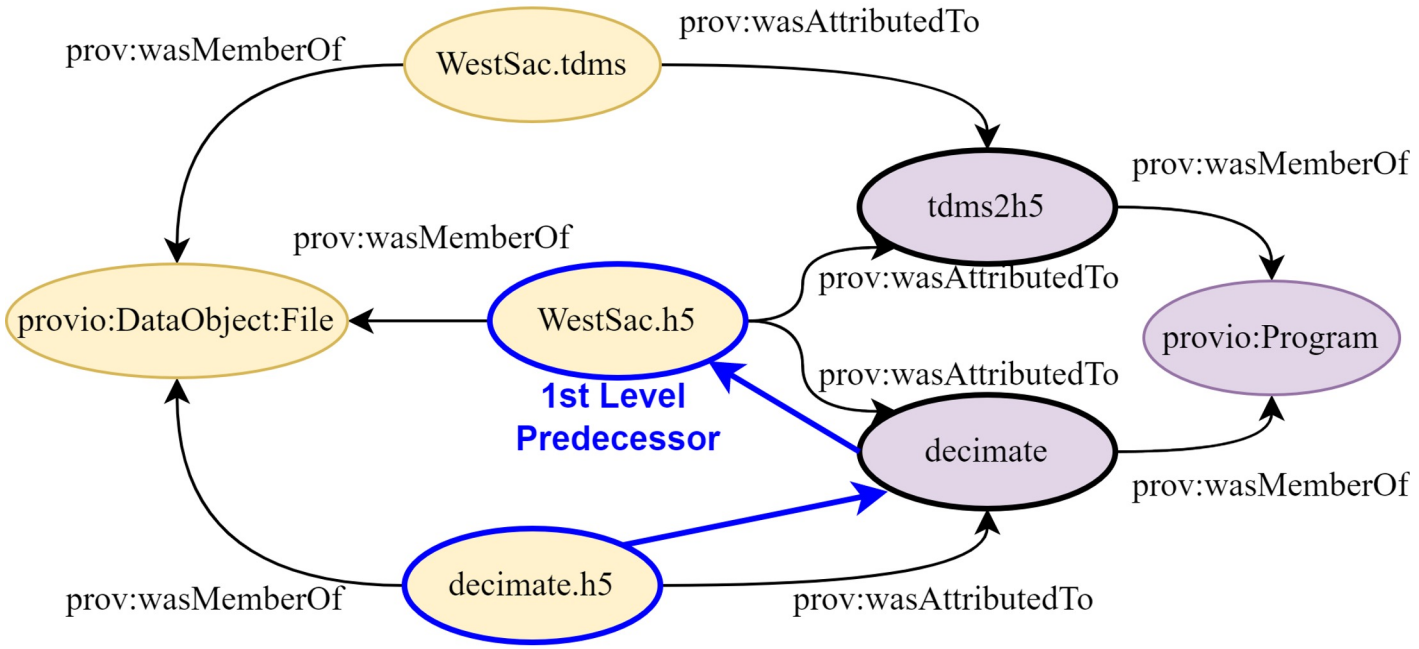
Step 1. Search the program whose output is decimate.h5

Example SPARQL query:

Decimate.h5 prov:wasAttributedTo ?

How to Query Provenance

- Data lineage backward tracing example with DASSA



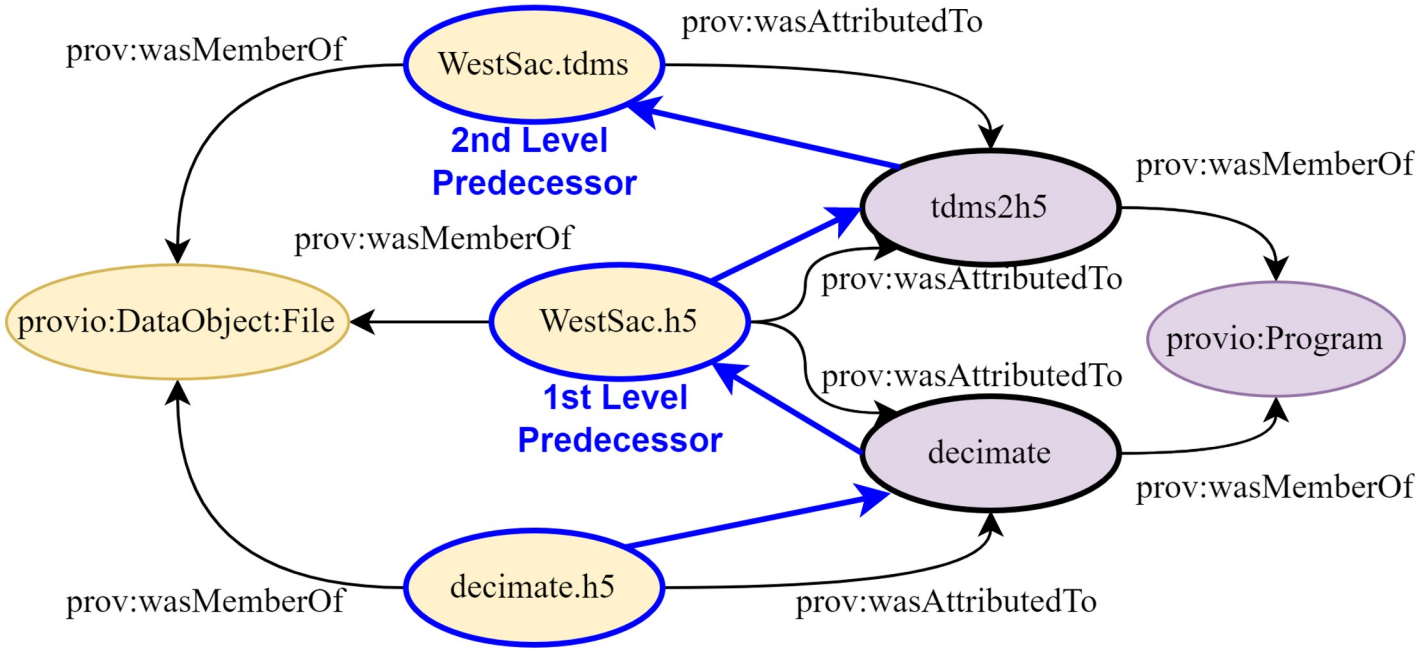
Step 2. Search the input data of decimate

Example SPARQL query:

? prov:wasAttributedTo decimate

How to Query Provenance

- Data lineage backward tracing example with DASSA



LoC of query = **N** * **3**
(**N** is level of predecessor data object)

Step 3&4. Repeat Step 1&2. Look for the program which created WestSac.h5 and then search the input of that program