Workflow Enablement & AdVanced Environments (WEAVE)

AWS RADIUSS Tutorial

Charles Doutriaux
and the WEAVE team: Dan Laney, David Domyancic, Sarah ElJurf, Brian Gunnarson, Jorge Moreno, Alex Murray, Lina Muryanto, Jeremy White

LLNL-PRES-854153 This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

WEAVE aims to enable a focused WSC/CP effort in supporting user workflows and developing and maintaining production workflow tools

1. Dedicated support for workflow tools (orchestration, data, & analysis)
2. Single team to share user support and staff outreach, marketing efforts
3. Make workflow implementation in user-community a first-class objective
4. Centralized Workflow Tools Development Team
   - Shared knowledge base
   - Cross tool collaboration and co-design
   - Move toward common Python CI/CD infrastructure
5. Centralized project with clear organizational structure and responsibilities
   - WSC/CP takes responsibility for production workflow tools
   - Users know who to reach out to for help
   - Outside efforts know who to interface with

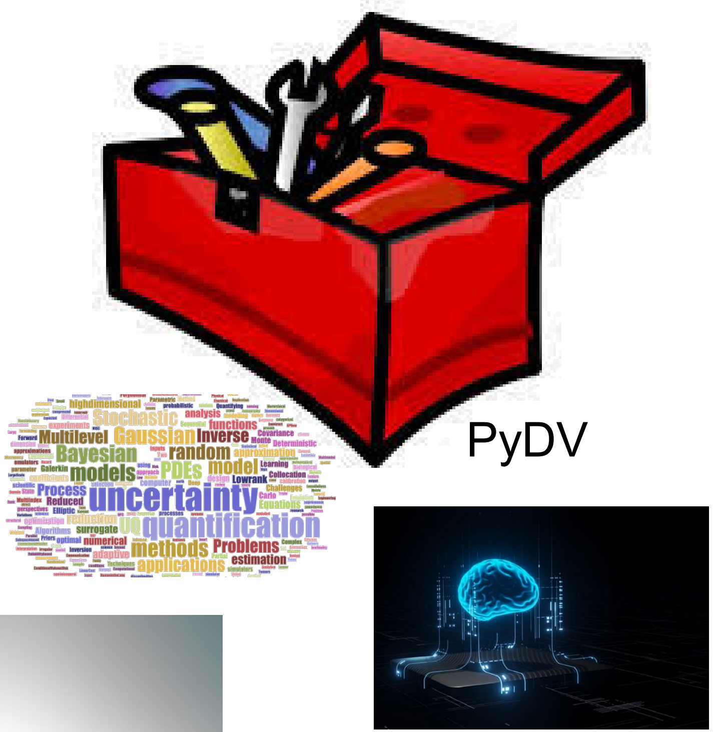
Our Vision: enhance user productivity and scientific quality through common infrastructure and tool integration

- Decrease human time from the posing of a problem to its answer
- Enable our users to take advantage of exascale computers like **El Capitan**
- Smooth integration and interoperation of **problem setup, simulation management, and analysis tools**
- Enable and support **code-agnostic inputs and outputs** for WSC/CP applications
- Allow users to work at a higher level than a **“Sea of Files”**
  - But let them **“see their files!”**
- Provide documentation and examples to enable **“tool builders”** in the user community to leverage a common workflow toolkit

WEAVE Tools Components

maestro

Merlin



PyDV

KOSH 2.0

Sina

{JSON}

HDF

csv

WSC's WEAVE team supports multiple tools for building complex HPC workflows

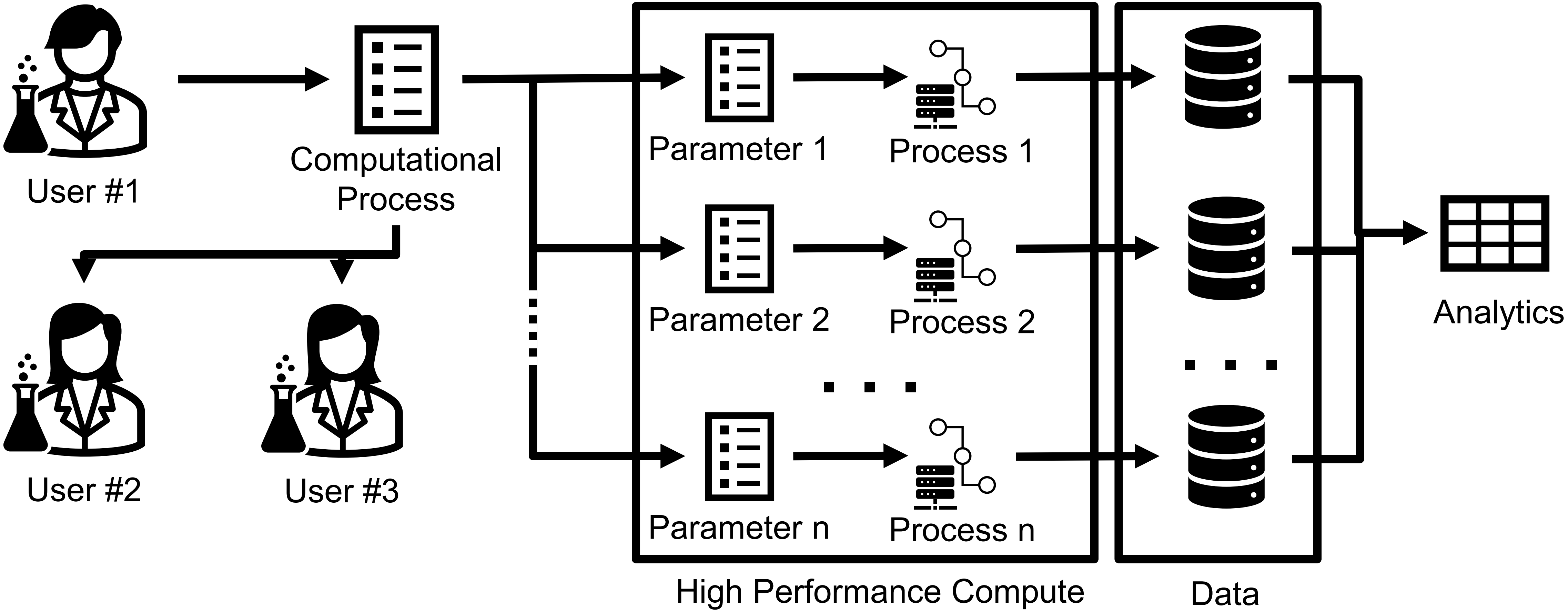
- Maestro
  - Provides minimal DSL for converting a task-graph of work into a human readable, machine agnostic specification
  - Manage execution on variety of HPC resources
  - Focus on shareability, repeatability, and provenance of what was run
  - Dependency lite for easier extendibility to enable an ecosystem of tools derived from it

- Merlin
  - Extension of Maestro to use distributed task queues for cross machine workflows
  - Tackle scaling problems with running many short jobs that overwhelm system level schedulers

- Additional Workflow managers built on top of these tools:
  - Encore: Developed in GS, is an extension of Maestro exploring addition of study level decision making and control flow for improved iterative/chained workflows

Orchestration: Maestro

Maestro aims to enable reproducible computational science following the model set by the experimental disciplines



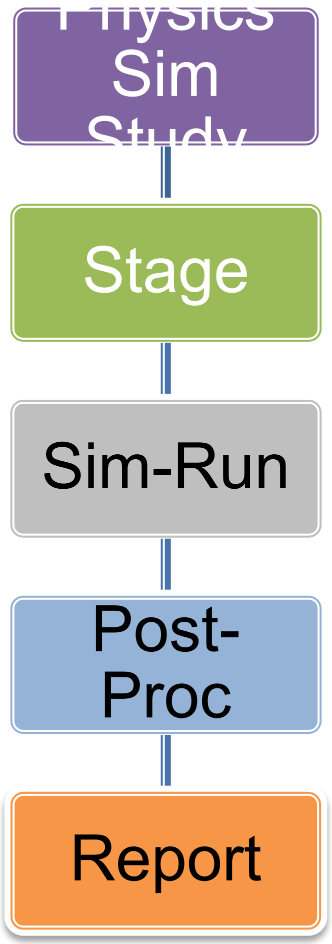
Maestro's vision is to become a toolchain and ecosystem of tools adaptable to user's needs

Build up your workflow starting with the intent of the process

YAML Specification

```
description:
   name: physics_sim
   description: Run study using a parallel physics code.

study:
   - name: stage
     description: Copy input decks and post-proc tools into study workspace.
  ...

 - name: sim-run
      description: run the simulation
   ...

- name: post-proc-run
      description: processes the simulation outputs for a single run
   ...

- name: report
      description: collects extracted data and performs summary analysis and reporting
   ...
```

Workflow Process



Physics Sim Study

Stage

Sim-Run

Post-Proc

Report

Minimal DSL enables a straightforward extension layer on top of familiar batch script based HPC workflows

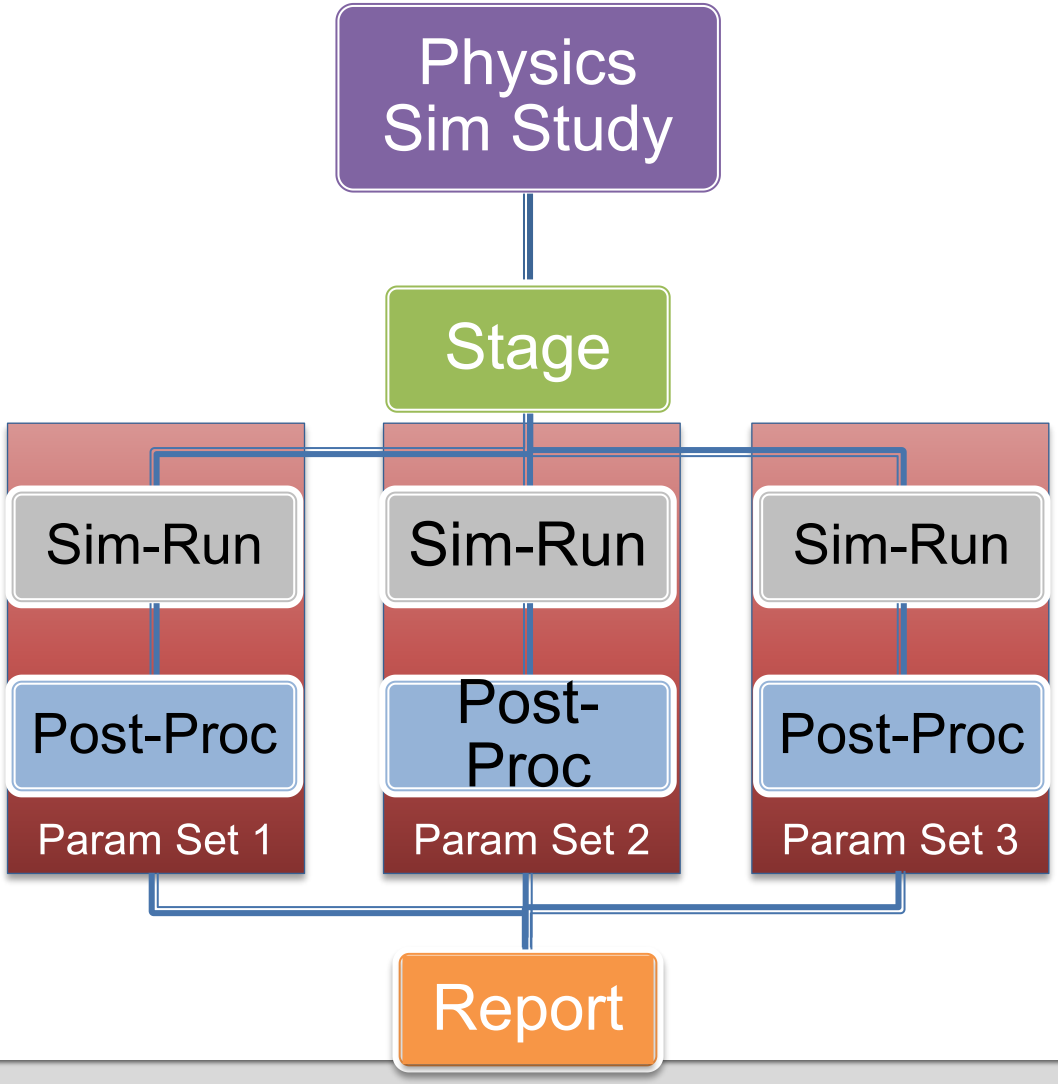
- Turns your workflow description into a generic template that Maestro inflates at run time
  - Tokens: $(TOKEN_NAME)
- Define variables and dependencies (path, git, ..) and insert in your workflow steps using tokens
- Launcher token for decoupling steps from schedulers: $(LAUNCHER)
- Flexible definition of parameter tokens via user defined python functions for complex sampling
- Intuitive way to specify step dependencies
- Reference parent step workspaces in a portable way for sharing data between steps

```
env:
    variables:
        OUTPUT_PATH: ./phys_studies
        SIM_CODE: /path/to/physics/code/install/

    labels:
        INPUT_FILE: model.input
        INPUT_PATH: $(SPECROOT)/$(INPUT)

study:
 - name: sim-run
      description: run the simulation.
      run:
        cmd: |
            cp $(stage.workspace)/$(INPUT_FILE) .

            $(LAUNCHER) $(SIM_CODE)/sim_exe -in $(INPUT_FILE) –p 4 --res $(RES)
        procs: 4
        nodes: 1
        walltime: "00:10:00”
        depends: [stage] # Require stage step to complete before running this step

global.parameters:
   RES:
        values: [10, 20, 30]
        label: RES.%%   # Expanced by Maestro to RES.10, RES.20, …
```

Having a complete, concrete, repeatable process, Maestro can run experiments with it by inflating the graph with parameters

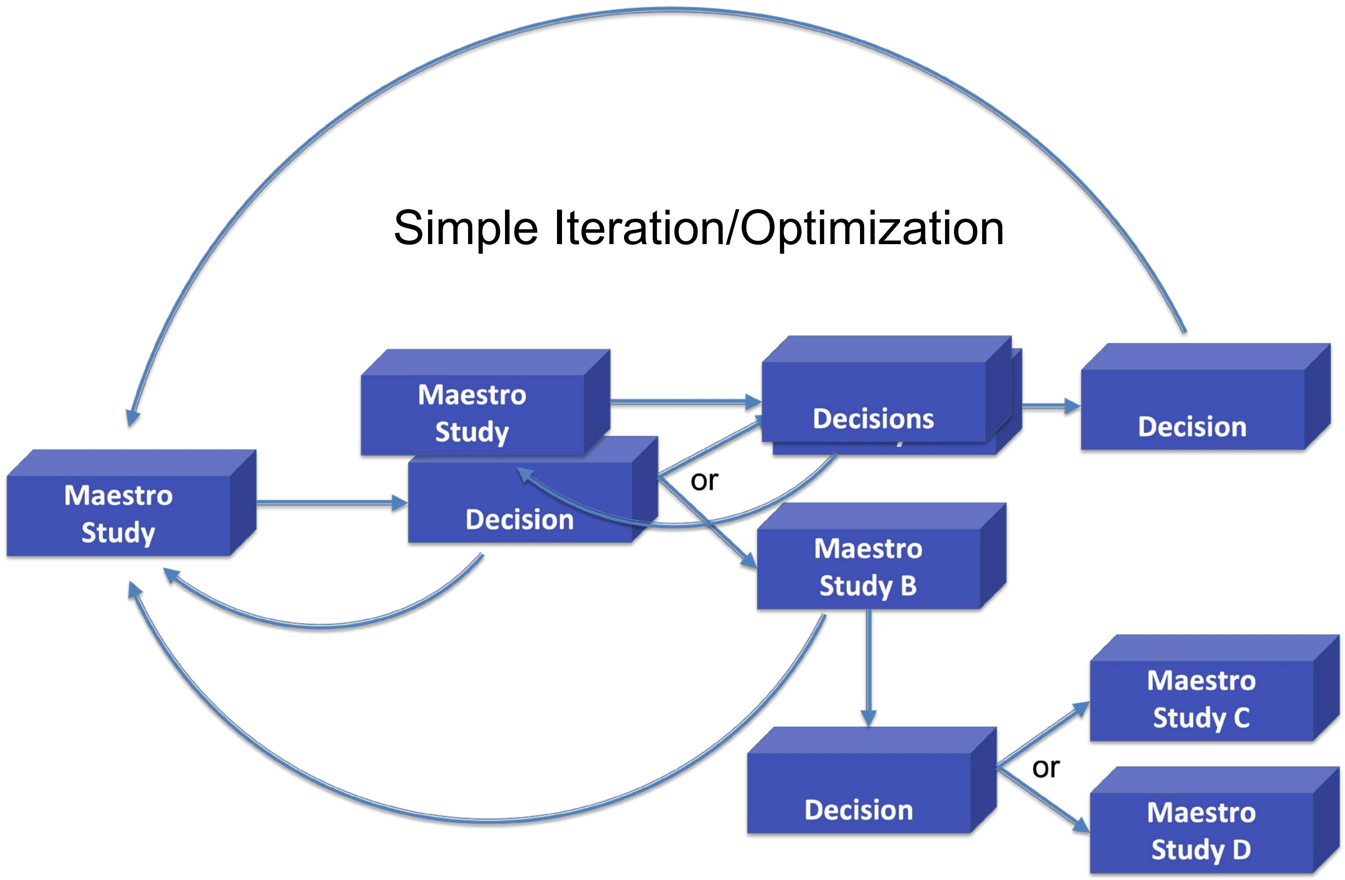
YAML Specification

```
study:
                                        …
 - name: sim-run
     description: run the simulation.
     run:
       cmd: |
          cp $(stage.workspace)/$(INPUT_FILE) .

          $(LAUNCHER) $(SIM_CODE)/sim_exe -in $(INPUT_FILE) –p 4 –res
$(RES)
       procs: 4
       nodes: 1
       walltime: "00:10:00”
                                        …

global.parameters:
    # Resolutions to run
    RES:
       values : [1, 2, 3]
       label : RES.%%
                                        …
```



Vision is to be a tool chain/ecosystem of orchestration tools for building complex workflows

- Two such examples already:
  - **Merlin**: exploring an implementation of distributed task queues and recursive type iterative workflows
  - **Encorewf**: adds outer layer with explicit decision-making steps and tracking of per-study iterations.
    - Simple single study spec based iterative and optimization type workflows
    - Chain disparate Maestro study spec's for more complex multi-phase workflows



Complex multi-phase iteration/optimization

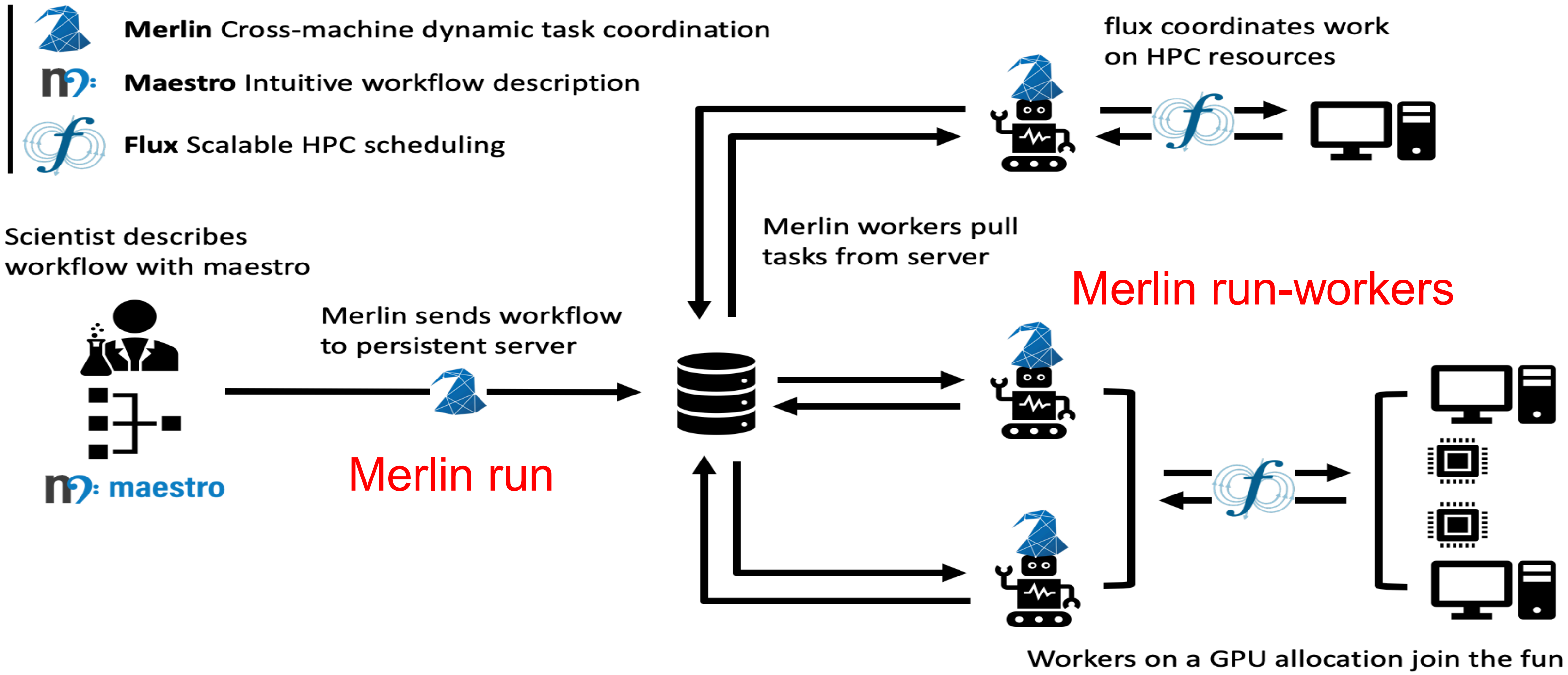
Simple Iteration/Optimization

Merlin – Another workflow orchestration tool

- Merlin was created with the goal of making it easy to build, run, and process the kinds of large-scale HPC workflows needed for cognitive simulation
- Built as an extension of Maestro

Merlin runs its’ workflows using a producer/consumer model

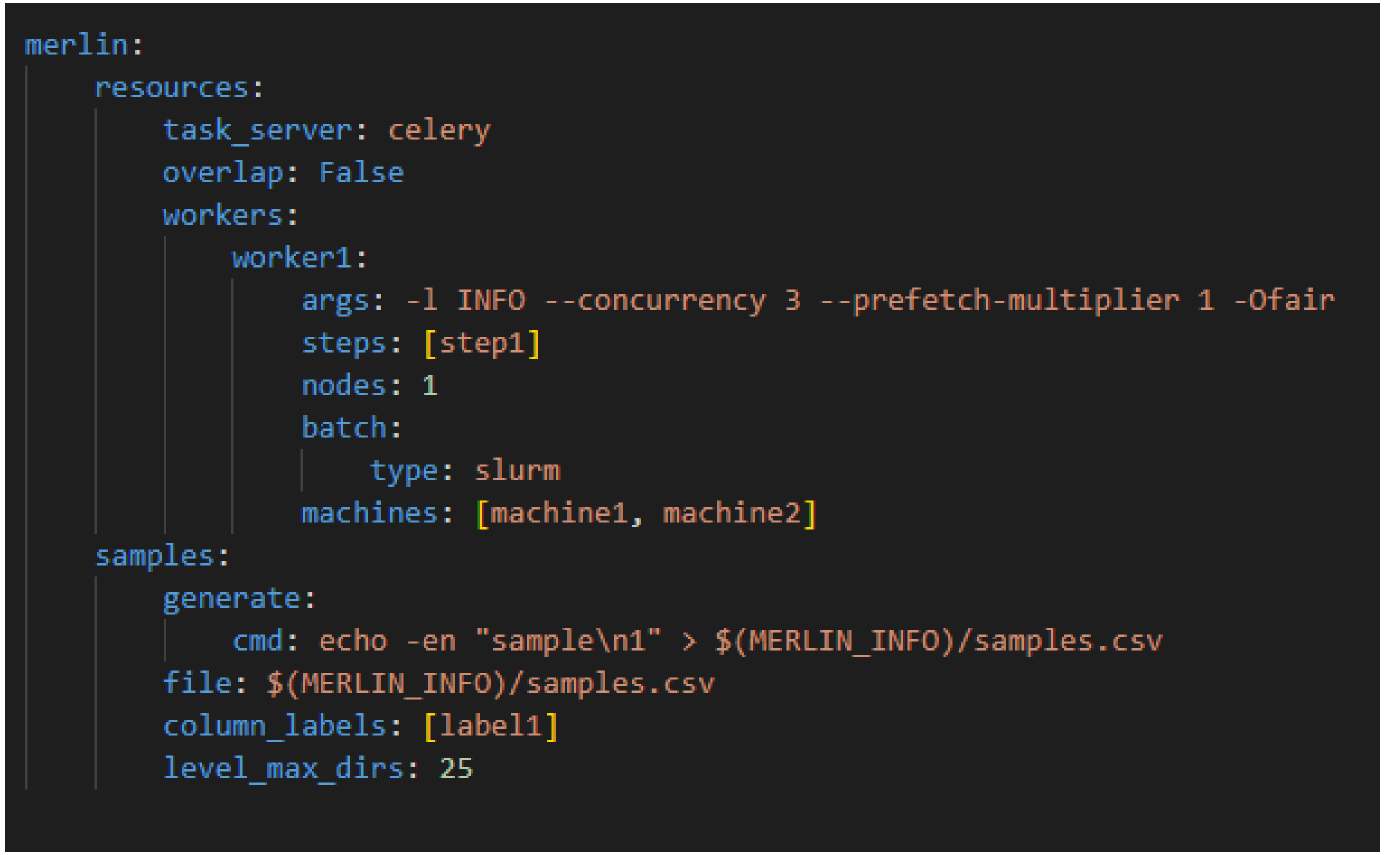
- Merlin uses the same YAML spec format as Maestro
- Central-server based system
- Two commands for the producer/consumer model
- HPC scheduling via Flux, Slurm, LSF, or PBS (untested w/ PBS so be wary)



Merlin Cross-machine dynamic task coordination
Maestro Intuitive workflow description
Flux Scalable HPC scheduling
Scientist describes workflow with maestro
maestro
Merlin sends workflow to persistent server
Merlin run
Merlin workers pull tasks from server
flux coordinates work on HPC resources
Merlin run-workers
Workers on a GPU allocation join the fun

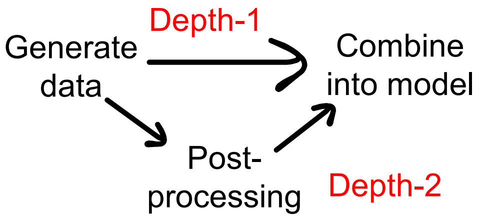
Differences From Maestro

- CLI substitutions
  - e.g. variables, samples, etc.
- Merlin block in spec file
  - Samples (run for each set of parameters)
  - Work not tied to resources
  - Work can be shared across:
    - Batch jobs
    - Machines

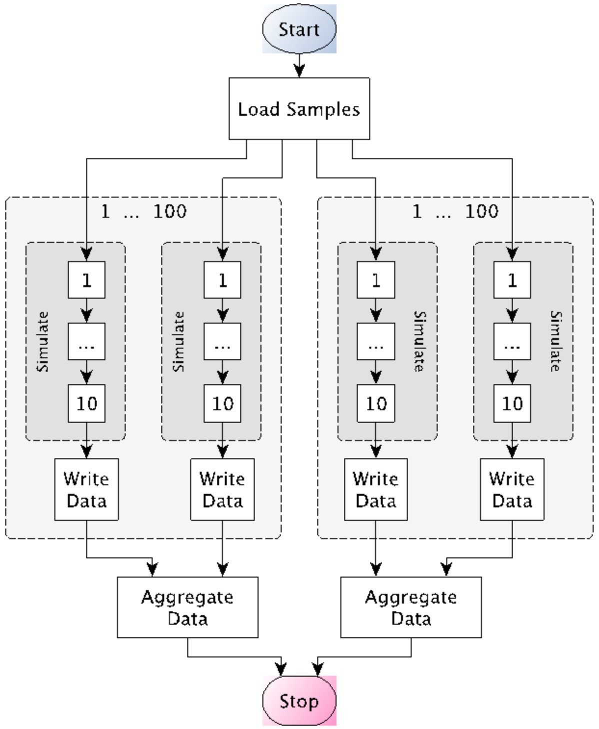


```yaml
merlin:
    resources:
        task_server: celery
        overlap: False
        workers:
            worker1:
                args: -l INFO --concurrency 3 --prefetch-multiplier 1 -Ofair
                steps: [step1]
                nodes: 1
                batch:
                    type: slurm
                machines: [machine1, machine2]
    samples:
        generate:
            cmd: echo -en "sample\n1" > $(MERLIN_INFO)/samples.csv
        file: $(MERLIN_INFO)/samples.csv
        column_labels: [label1]
        level_max_dirs: 25
```

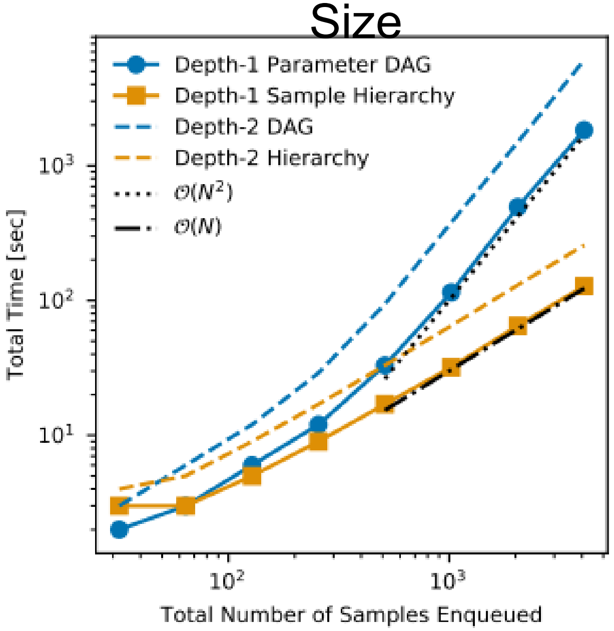
Scalability



Large-Scale Ensembles



Ideal Task Overhead – Independent of Ensemble Size



Maestro DAG

Merlin Hierarchy

J. L. Peterson *et al.* “Enabling Machine Learning-Ready HPC Ensembles with Merlin.” Future Generation Computer Systems. **131**:255-268 (2022).
J. L. Peterson, “Merlin Short Overview”, Lawrence Livermore National Laboratory, 2022.

Merlin allows for iterative workflow runs via recursive studies

- Allows for a base-case where you can do whatever is needed with the results of your iterative runs
  - E.g. running a post-processing script over all result files generated
- Once workers are started, they'll keep pulling tasks from their queues until they're stopped
  - This allows workers to keep completing work even if the work is queued from a different run of a study
  - i.e. no need to start the workers a second time

```
env:
   variables:
      ITER: 1
      MAX_ITER: 10
      PYTHON: <path to python>
      POST_PROC: <path to cumulative post processing script>
      ITER_OUTPUT: <path to iteration outputs>

study:
<other steps>
 - name: iterate
   description: run another iteration or post process all iterations
   run:
      cmd: |
         if [ $(ITER) -ge $(MAX_ITER) ] ; then
                  echo "done"
                  $(PYTHON) $(POST_PROC) $(ITER_OUTPUT)/iter_*_results.json
         else
                  next_iter=$(ITER)
                  ((next_iter=next_iter+1))
                  echo "Starting iteration " $next_iter
                  cd $(SPECROOT)
                  merlin run $(SPECROOT)/iterative_demo.yaml --vars ITER=$next_iter
          fi
      task_queue: iterative_queue
```

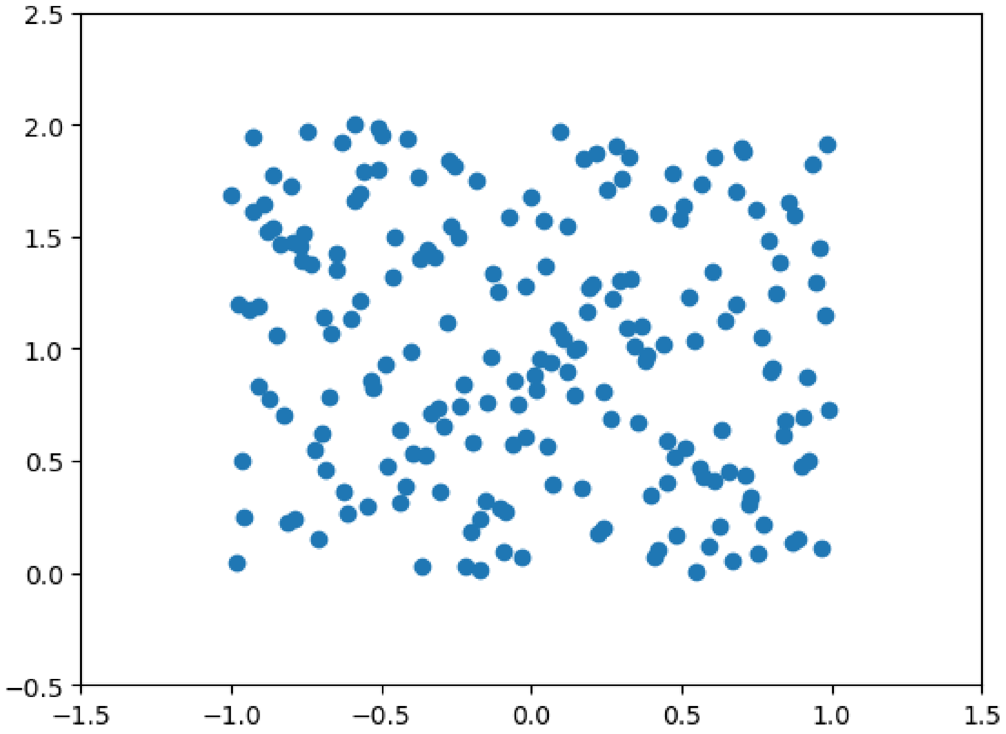
TRATA

- Trata is a Bayesian sampling package designed to allow users to explore a parameter space by generating sample points
- Trata is a Spanish word from the Latin *trahō* “to extract” and is used colloquially to mean “try” or “sample”
- Trata includes the following modules:
  - **`composite_samples`**
  - **`sampler`**
  - **`adaptive_sampler`**

TRATA

- The **`sampler`** module contains 16 sampling methods users can choose from to generate samples across a design parameter space

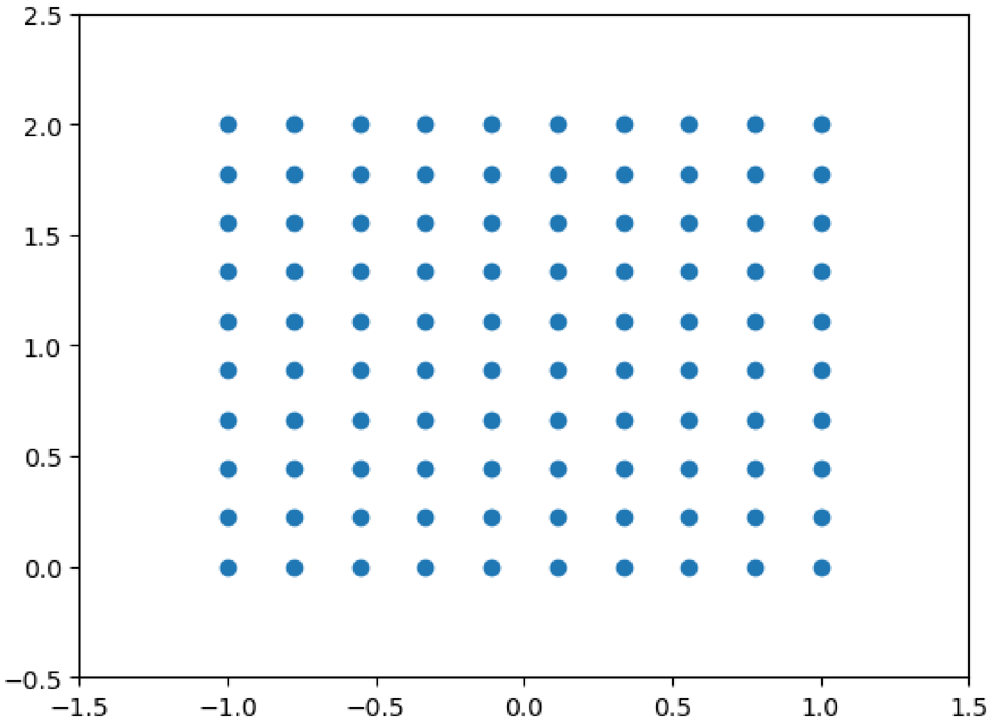
- Common methods include:
  - LatinHyperCube



TRATA

- The `sampler` module contains 16 sampling methods users can choose from to generate samples across a design parameter space

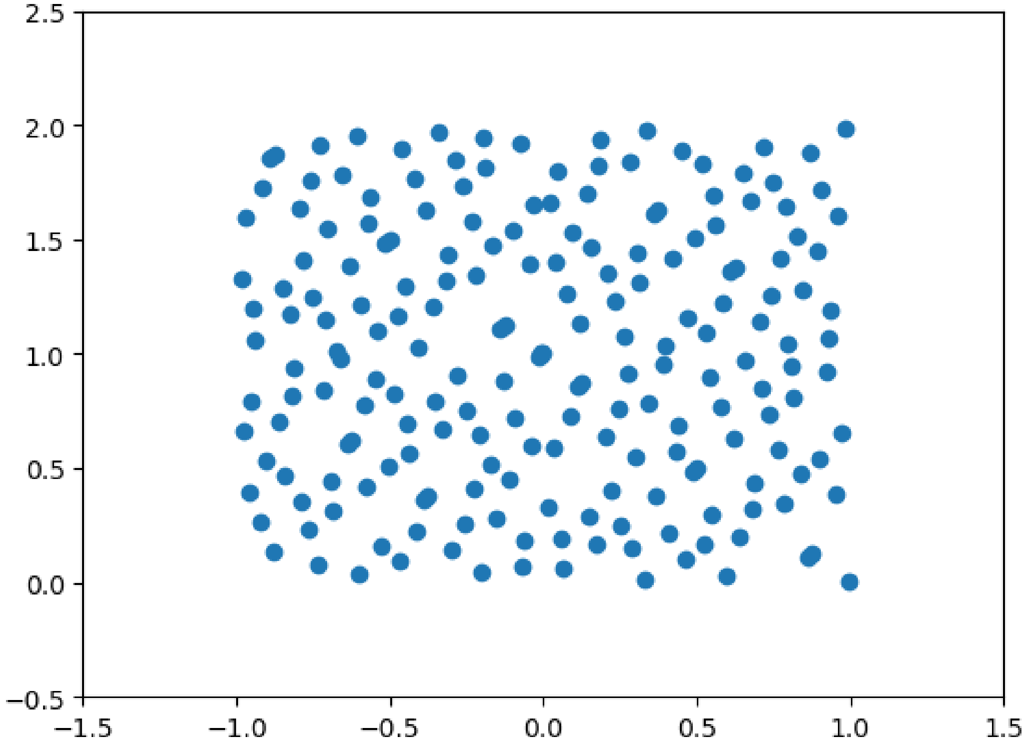
- Common methods include:
  - LatinHyperCube
  - CartesianCross



TRATA

- The `sampler` module contains 16 sampling methods users can choose from to generate samples across a design parameter space

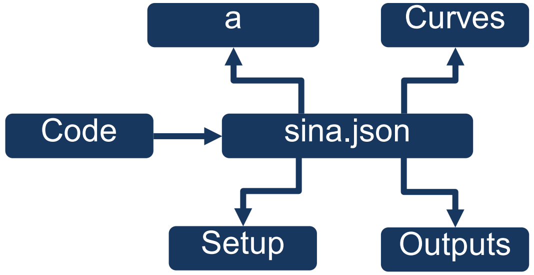
- Common methods include:
  - — LatinHyperCube
  - — CartesianCross
  - — QuasiRandom

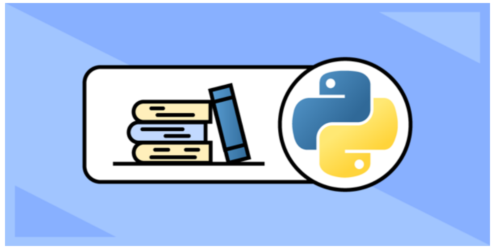


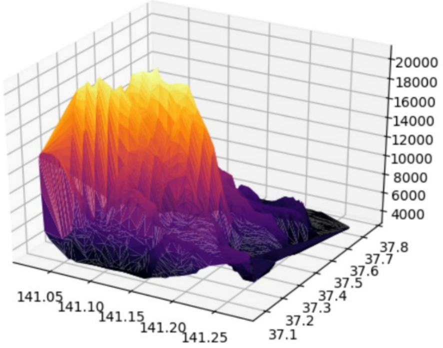
WEAVE’s Sina Tool

Sina handles data munging

- Created to increase the availability of simulation data with minimal user effort
  - Integrates directly with codes to pull data
  - Outputs data into an easily-automated format
  - Provides tooling to render this data interactive (through Jupyter) and easily processed (through a Python API) across one run or many
- Intended especially for use with run metadata, allowing users to easily and efficiently find simulation runs that match their criteria.





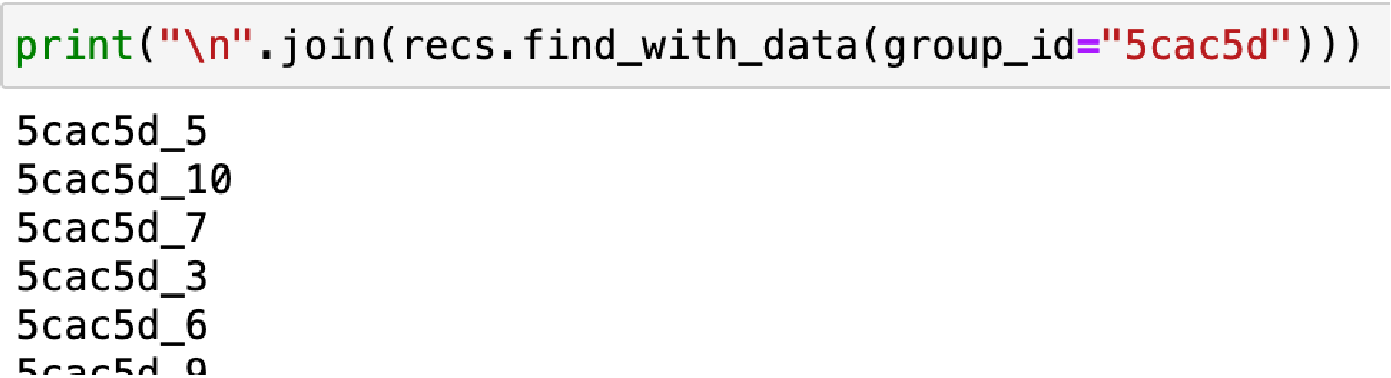


Use cases and place in ecosystem

- Allow users to query and view hundreds, thousands, millions of result sets at once
- Drill down to “interesting” runs
- Targets data that **identify a run as interesting within the context of other runs**
- Does not orchestrate
- Does not replace VisIt or similar tools

Sina: Search API

- Queries are fast, terse, and flexible, ideally being intelligible without previous Sina experience

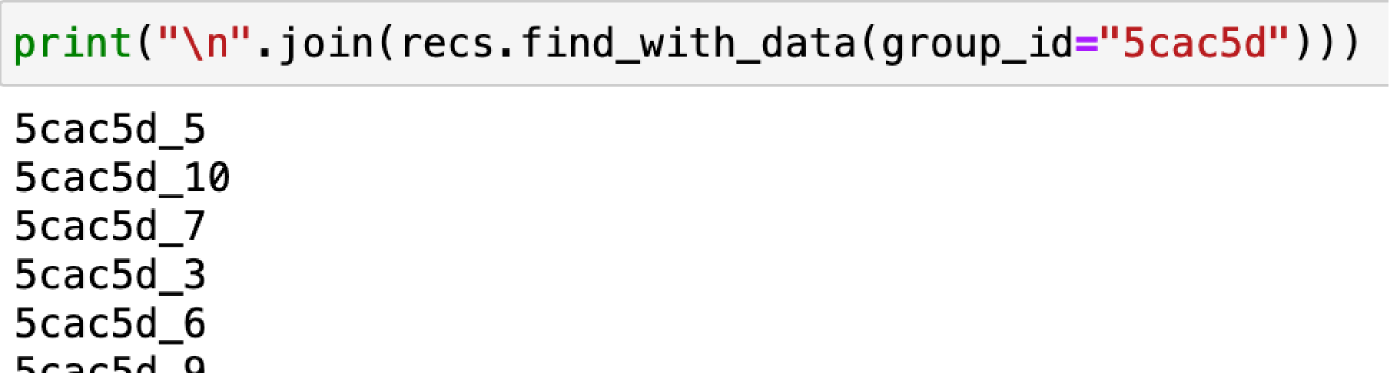


```
print("\n".join(recs.find_with_data(group_id="5cac5d")))
```

```
5cac5d_5
5cac5d_10
5cac5d_7
5cac5d_3
5cac5d_6
5cac5d_9
```

Sina: Search API

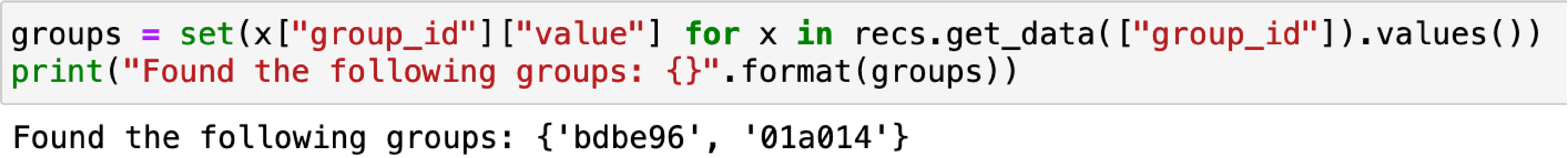
- Queries are fast, terse, and flexible, ideally being intelligible without previous Sina experience



```
print("\n".join(recs.find_with_data(group_id="5cac5d")))
```

```
5cac5d_5
5cac5d_10
5cac5d_7
5cac5d_3
5cac5d_6
5cac5d_9
```

- Because returned objects are pure Python (rather than db-specific), further manipulation is easy



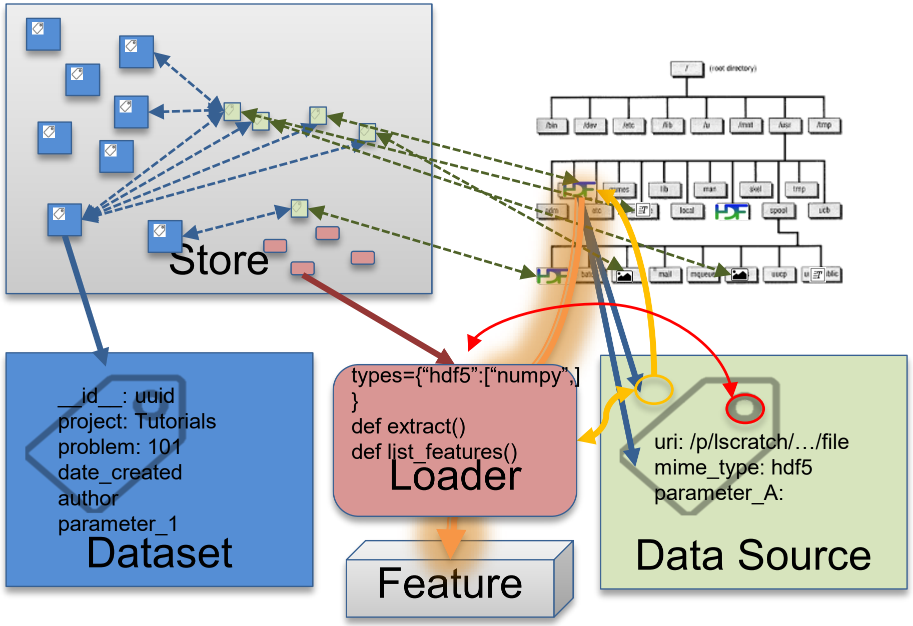
```
groups = set(x["group_id"]["value"] for x in recs.get_data(["group_id"]).values())
print("Found the following groups: {}".format(groups))
```

```
Found the following groups: {'bdbe96', '01a014'}
```

WEAVE’s Kosh Tool

Why use Kosh?

- Kosh is built on top of Sina:
  - — Kosh can load a Sina store. Sina can query a Kosh store.
  - — Sina is being integrated with Physics Codes.
    - Lots of Data and Metadata Accessible out of the box.
  - — Multiple Database backends
    - Includes LC’s Kubernetes deployed MySQL and Cassandra DB.
  - — Optimized query search
- Kosh has unique features:
  - — Kosh can access and process data in a consistent fashion, decoupled from data format and location via
    - "loaders“
    - "transformers“
    - "operators



Store

__id__: uuid
project: Tutorials
problem: 101
date_created
author
parameter_1
Dataset

types={“hdf5”:[“numpy”,]
}
def extract()
def list_features()
Loader

Feature

uri: /p/lscratch/…/file
mime_type: hdf5
parameter_A:
Data Source

Features

- Locating and sharing data
  - Once data is in the store, end user doesn’t need to know where it is.
  - Very easy to extend and add your own “loaders”.

```
dataset.associate(“/pth/to/some/file”, mime_type=“hdf5”)
```

- Moving data
  - Kosh can move data as it typically won’t sit in one place.
  - Can be done after the fact as well with re-associate function

```
kosh mv –store mystore.sql –sources file1.hdf5 movies/*.mp4 dir1 –destination some_dest_dir
```

- Metadata validation
  - Kosh schemas help enforce consistency in your store

```
required = {“valid": [True, False]}
optional = {“training": [1, "yes“, 0, “y” ,”n” ,”no”, True, False]}
schema = kosh.KoshSchema(required, optional)
dataset.valid = 1
schema.validate(dataset) # -> Error
dataset.valid = True
dataset.schema = schema # -> goes into store and will be enforced
dataset.valid = 1 # -> Error
```

Features (cont.)

- Data manipulation
  - “Transformers”: allow data to be further post-processed after extraction from it's original URI and they can be chained

Post-process single feature (crop, etc…)

```
shuffle = kosh.transformers.npy.Shuffle()
data = ds.get_execution_graph(“feature”, transformers=[shuffle,])
```

  - “Operators”: allow data to be further post-processed coming from different features either from the same source or not.

Combine multiple features from one or many datasets, will find loader for each features so that the data comes in compatible.

```
class ADD(kosk.KoshOperators):
    types = {“numpy”:[“numpy”,]}  # mime types supported
     def operate(*inputs, **kargs):
         [some code to add inputs]
         return new_result
twice = ADD(data, data)[:]
```

IBIS

- IBIS (Interactive Bayesian Inference and Sensitivity) is designed to allow users to generate statistical models after physics simulations have run to completion
- These models can be used to predict the results of future simulation runs, and to perform sensitivity and uncertainty quantification analyses
- *Ibis* comes from the Egyptian word for the long-legged wading bird associated with the ancient Egyptian deity Thoth, god of science and magic

IBIS

- Markov chain Monte Carlo (MCMC) methods are commonly used in uncertainty quantification to assign probability distributions to uncertain design parameters

- Polynomial chaos expansion (PCE) models are an alternative to MCMC and provide the statistical characteristics of results with greatly reduced computational cost

- IBIS includes the following modules:
  - `filter`
  - `likelihoods`
  - `mcmc`
  - `mcmc_diagnostics`
  - `pce_model`
  - `sensitivity`
  - `plots`

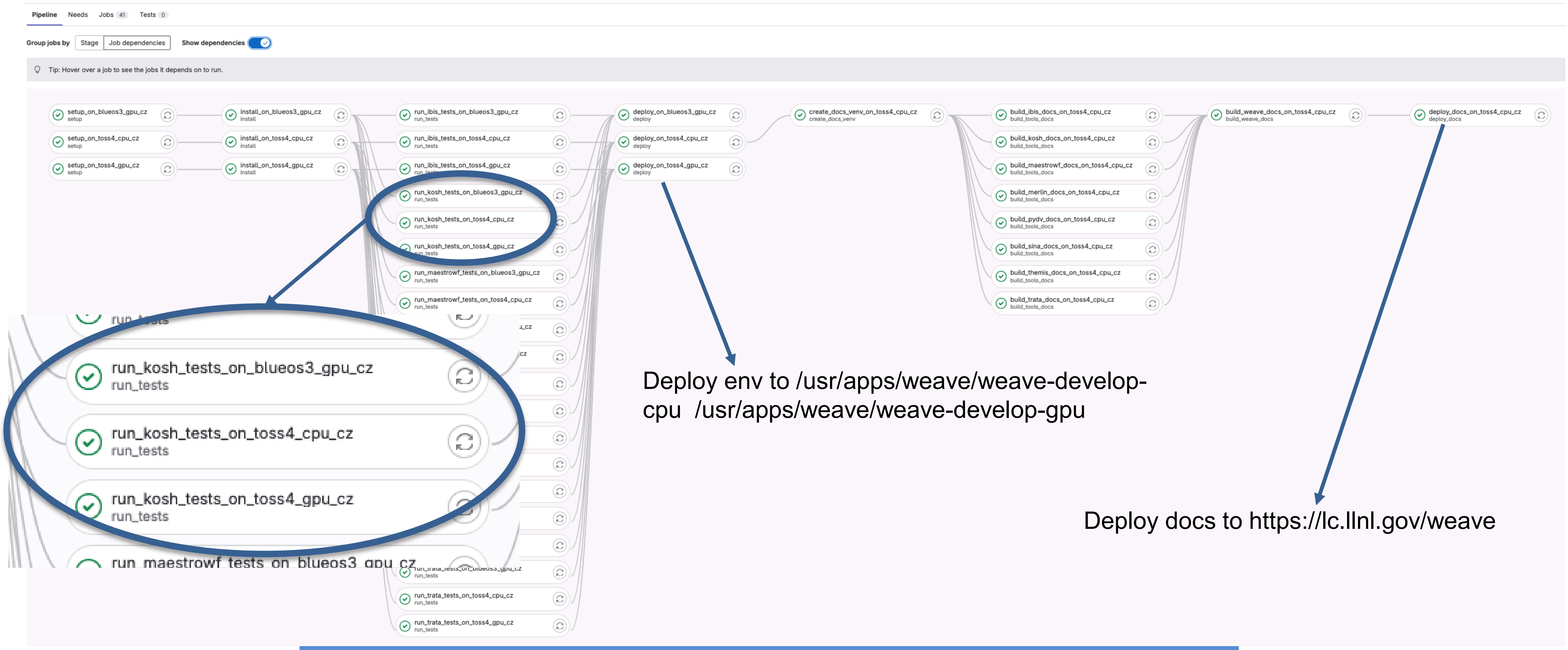
IBIS and Trata

- Trata and IBIS are often used jointly as sampling is largely a pre-req to any UQ analysis
- The following example shows how sensitivity analysis can be used on a polynomial function that only acts on the first 10 variables
- Mutual information rankings order the measured amount of information obtained about one random variable by observing another
  - Recall a high mutual information indicates a large reduction in uncertainty; low mutual information indicates a small reduction; and zero mutual information between two random variables means the variables are independent

Where can I get these tools? And can I trust they work?

- Everything is under LLNL’s Github umbrealla at:
- http://github.com/LLNL/maestrowf
- http://github.com/LLNL/merlin
- http://github.com/LLNL/sina
- http://github.com/LLNL/kosh
- http://github.com/LLNL/trata
- http://github.com/LLNL/ibis

WEAVE CI – automates documentation build and deployment



Pipeline | Needs | Jobs 41 | Tests 0

Group jobs by: Stage | Job dependencies — Show dependencies ☑

Tip: Hover over a job to see the jobs it depends on to run.

```
setup_on_blueos3_gpu_cz            install_on_blueos3_gpu_cz      run_ibis_tests_on_blueos3_gpu_cz        deploy_on_blueos3_gpu_cz   create_docs_venv_on_toss4_cpu_cz   build_ibis_docs_on_toss4_cpu_cz       build_weave_docs_on_toss4_cpu_cz   deploy_docs_on_toss4_cpu_cz
setup                              install                        run_tests                               deploy                     create_docs_venv                   build_tools_docs                      build_weave_docs                   deploy_docs
setup_on_toss4_cpu_cz              install_on_toss4_cpu_cz        run_ibis_tests_on_toss4_cpu_cz          deploy_on_toss4_cpu_cz                                        build_kosh_docs_on_toss4_cpu_cz
setup                              install                        run_tests                               deploy                                                        build_tools_docs
setup_on_toss4_gpu_cz              install_on_toss4_gpu_cz        run_ibis_tests_on_toss4_gpu_cz          deploy_on_toss4_gpu_cz                                        build_maestrowf_docs_on_toss4_cpu_cz
setup                              install                        run_tests                               deploy                                                        build_tools_docs
                                                                  run_kosh_tests_on_blueos3_gpu_cz                                                                      build_merlin_docs_on_toss4_cpu_cz
                                                                  run_tests                                                                                             build_tools_docs
                                                                  run_kosh_tests_on_toss4_cpu_cz                                                                        build_pydv_docs_on_toss4_cpu_cz
                                                                  run_tests                                                                                             build_tools_docs
                                                                  run_kosh_tests_on_toss4_gpu_cz                                                                        build_sina_docs_on_toss4_cpu_cz
                                                                  run_tests                                                                                             build_tools_docs
                                                                  run_maestrowf_tests_on_blueos3_gpu_cz                                                                 build_themis_docs_on_toss4_cpu_cz
                                                                  run_tests                                                                                             build_tools_docs
                                                                  run_maestrowf_tests_on_toss4_cpu_cz                                                                   build_trata_docs_on_toss4_cpu_cz
                                                                  run_tests                                                                                             build_tools_docs
                                                                  u_cz
                                                                  cz
                                                                  run_trata_tests_on_blueos3_gpu_cz
                                                                  run_tests
                                                                  run_trata_tests_on_toss4_cpu_cz
                                                                  run_tests
                                                                  run_trata_tests_on_toss4_gpu_cz
                                                                  run_tests
```

run_tests
run_kosh_tests_on_blueos3_gpu_cz
run_tests
run_kosh_tests_on_toss4_cpu_cz
run_tests
run_kosh_tests_on_toss4_gpu_cz
run_tests
run_maestrowf_tests_on_blueos3_gpu_cz

Deploy env to /usr/apps/weave/weave-develop-cpu /usr/apps/weave/weave-develop-gpu

Deploy docs to https://lc.llnl.gov/weave

All tools are systematically tested together in various environments

Thank You!

We look forward to helping you now and in the future.

Let’s move on to the tutorial. Led by Jorge Moreno.

Lawrence Livermore National Laboratory

Toy Tutorial Getting Started Steps

These are your login credentials (you should have received an email)

Instance IP: {ip_address}

SSH Username: radiuss{number}

SSH Password: {password}

You can connect to this instance by running this command:

```
ssh {username}@{ip_address}
```

then, enter the password when prompted.

Toy Tutorial Getting Started Steps

- micromamba activate
- micromamba activate /usr/apps/weave/weave-develop/
- cd weave-demos/CZ/ball_bounce/
- google-chrome &
- jupyter-lab &
  - Copy paste url from Jupyter-Lab into Google Chrome
- Follow README.md "WEAVE Workflow Example: How to run" section but skip Step 2 activate environment.
- Notebooks change: `%matplotlib notebook` to `%matplotlib inline`