



# NEEDLE\* – Workflow Orchestration for Large-Scale NSBI Deployment

SBI Blueprint Workshop 2026

**DESY** Levi Evans, Ulrich Husemann,  
Stephen Jiggins, Stefan Katsarov,  
Judith Katzy

**KIT** Felix Kahlhoefer, Niklas Reus, Lena  
Rathmann, Kylian Schmidt, Nicolò  
Trevisani

**U. Zagreb** Nino Kovacic

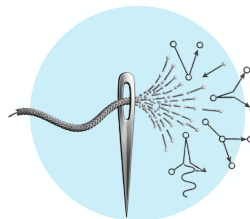
\* NEural-basEd Diffusion Likelihood Estimation

# The NEEDLE Collaboration

Joint project between KIT and DESY funded by Helmholtz AI, the German research association for Artificial Intelligence

## Our mission

- Bring NSBI to the wider public
- Scale methods to High-Performance Computing clusters
- Develop new flow-based density estimation methods



HELMHOLTZAI



# Outline

- 1. Orchestration for practical NSBI**
- 2. Large-scale data ingestion with dask**

# Orchestration for practical NSBI

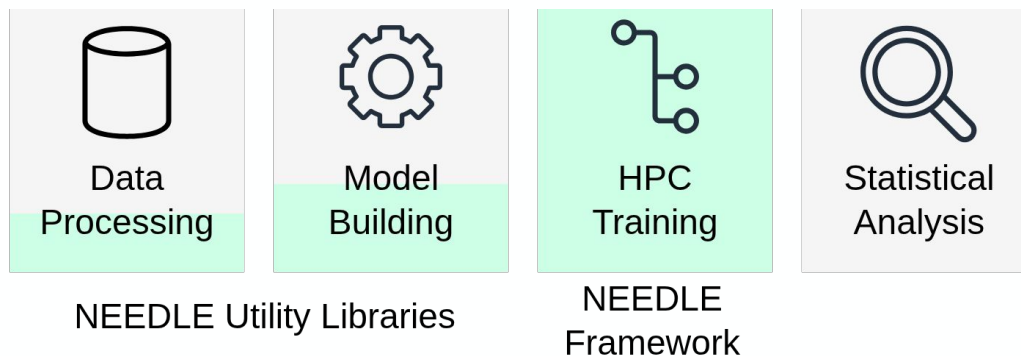
Scaling complex models to HPCs

# NSBI Orchestration Tool

## Motivation:

- Methods are developed first and workflows come second out of necessity
- Managing large amounts of networks becomes a **bottleneck** during development
- Fortunately, NSBI workflows mostly share the same steps
- **Common framework** would be a huge timesaver for the community

## Where NEEDLE fits in the NSBI ecosystem:



# Orchestrating hundreds of networks

- Popular workflow management tools like snakemake, law (CMS) or b2luigi are all candidates
  - Organize code execution (also locally)
  - Submit to batch systems with HTCondor, Slurms, etc...
  - Scale to large analyses and week-long jobs
- Balance capabilities with adoption potential
- We chose **law** due to its many HEP-style utilities



reana



Snakemake



b2luigi  
(Belle II)



Airflow



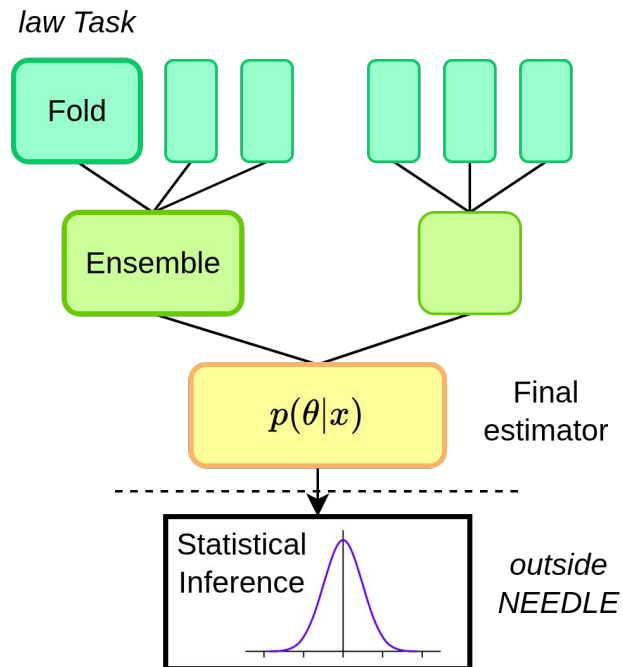
# Building NSBI as a graph

## Each Law Task

- Atomizes the full training of a single model
- Fails if any input / output are missing
- Can be rearranged and integrated into larger workflows

## Training

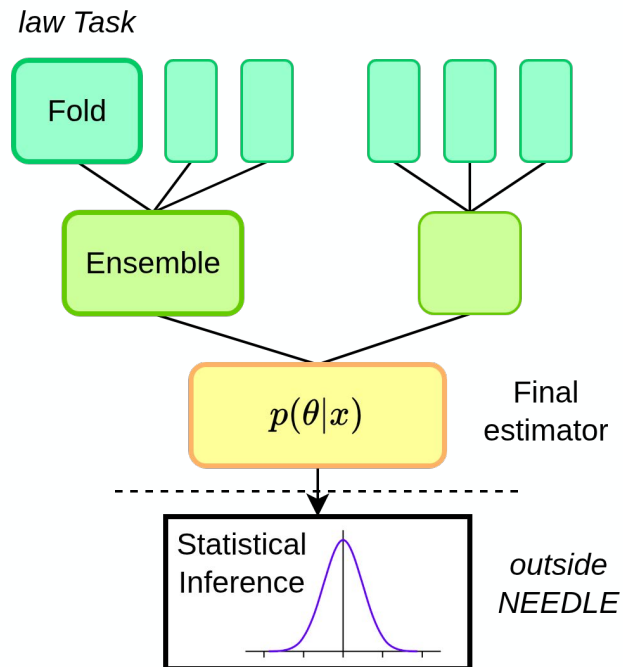
- Dependency graph fixed from single config
- Track individual trainings with MLFlow and Tensorboard
- Aggregate results from previous trainings and store the weights
- Question about model storage (.pt or ONNX) is open

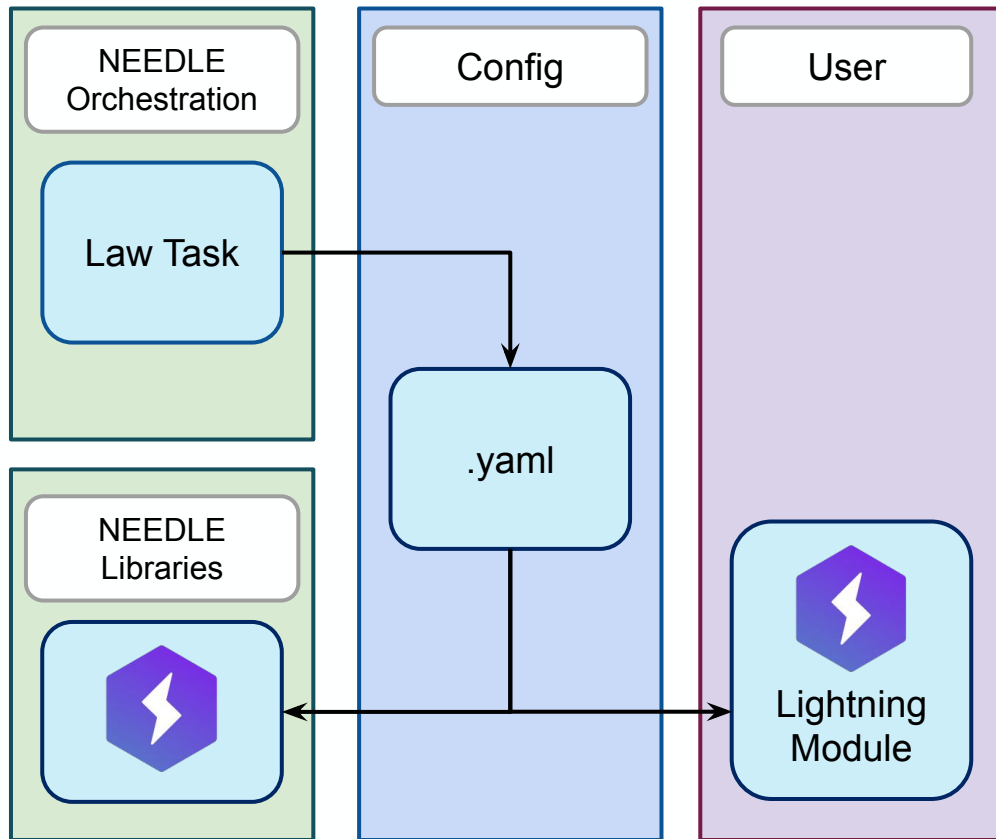


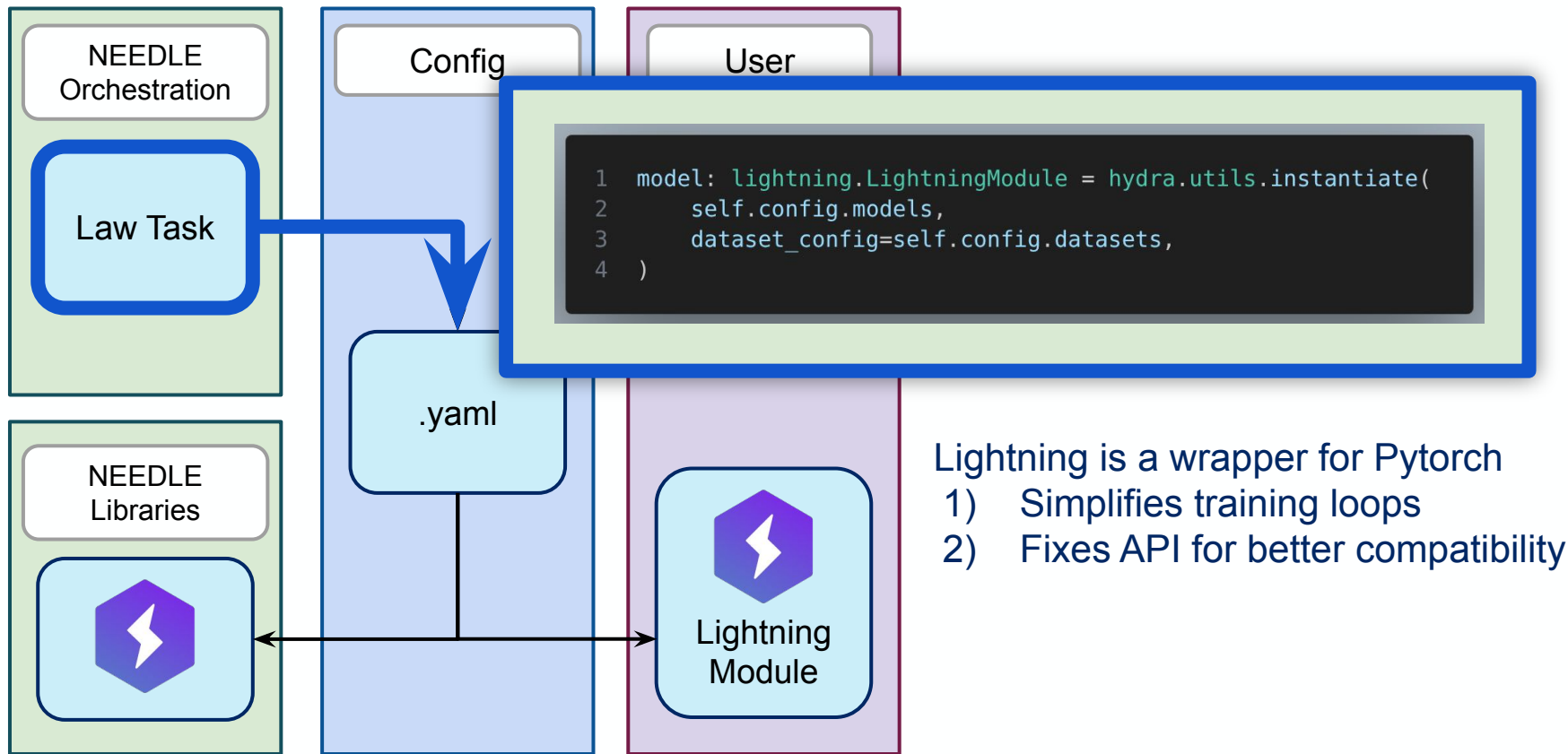
# Building NSBI as a graph

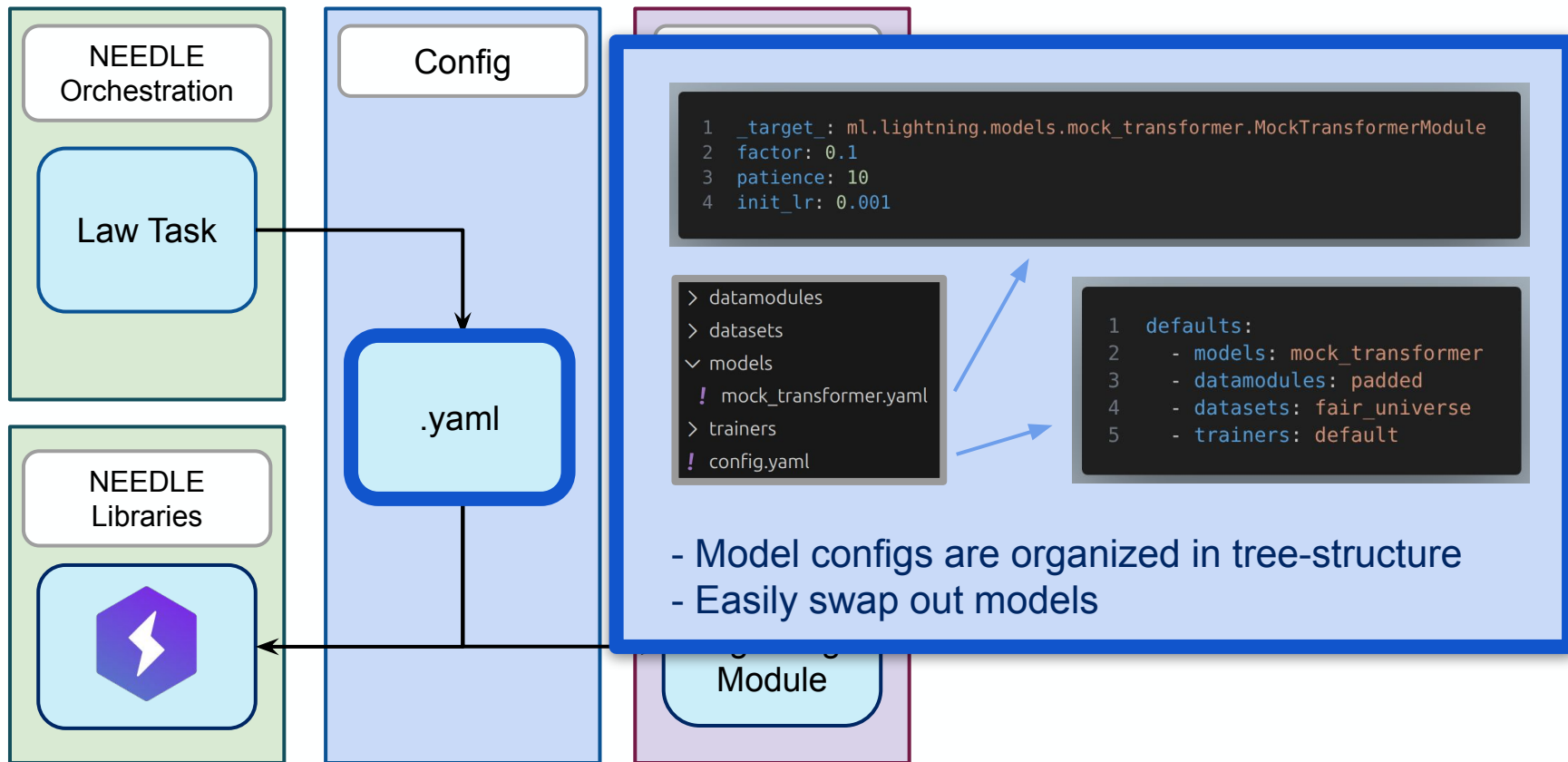
## Evaluation

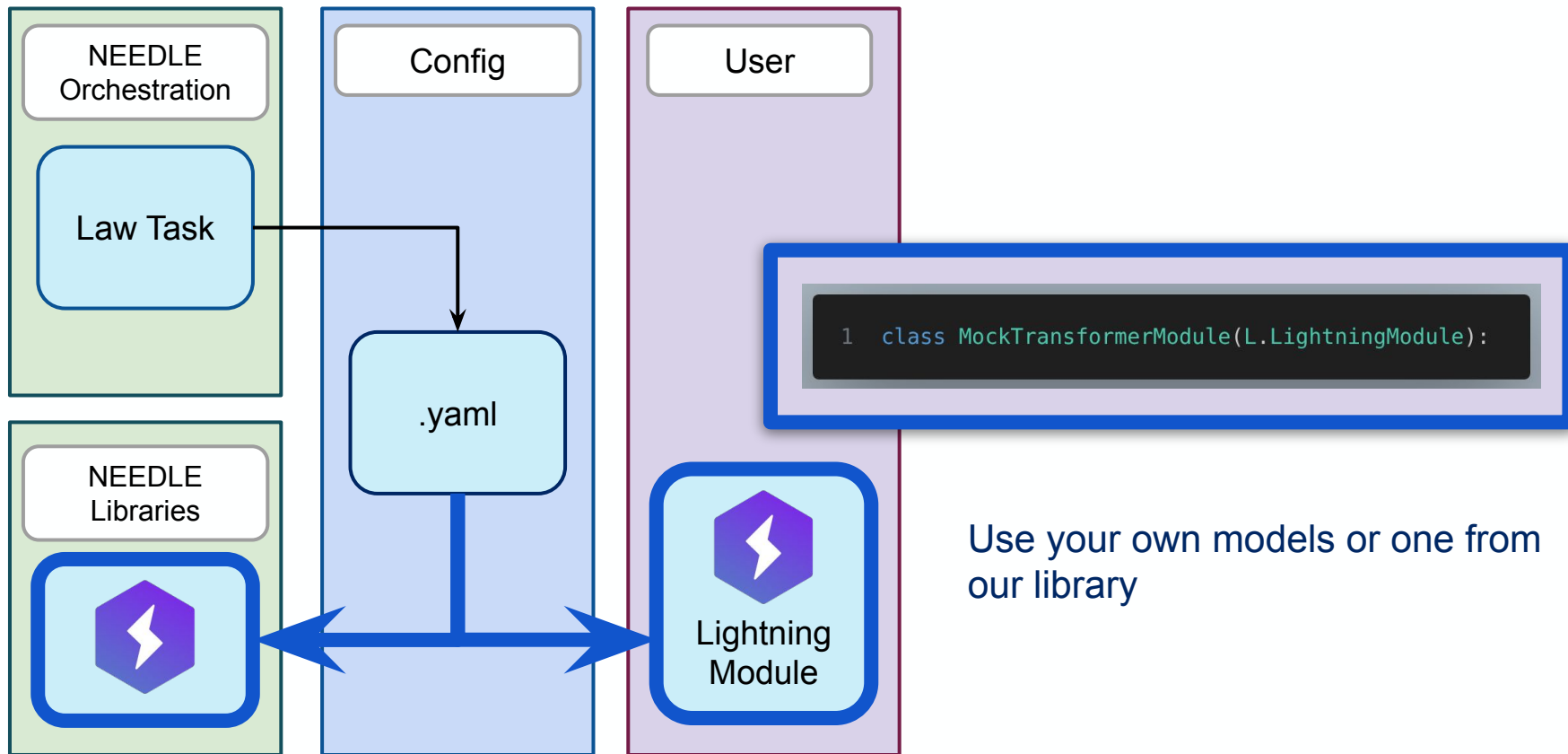
- Final estimator is a pseudo-model
  - Composition of all models and their weights
  - Depends on the flavor of NSBI
  - Fast local evaluation
- Compatibility with statistical modelling tools will be key
- We will try to match the standards imposed by the community







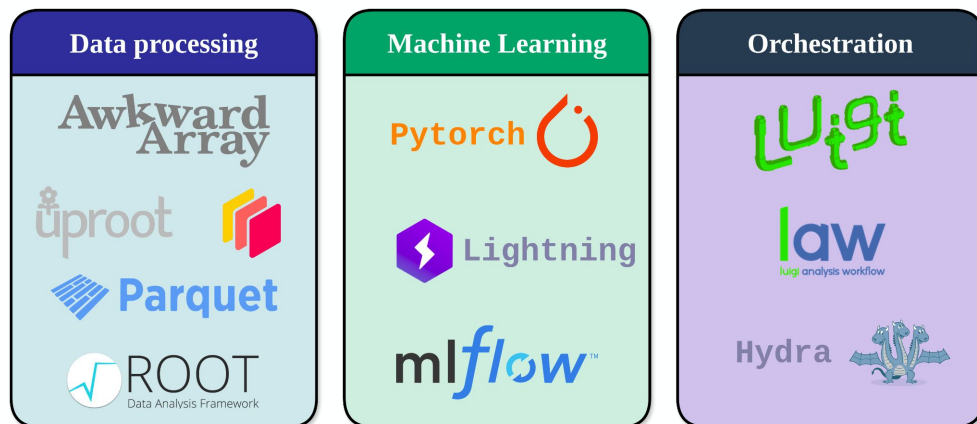




# API

Our recommendation for modular design

- Encapsulate all models as torch **Lightning** modules
- Expose hyperparameters as **yaml** configs (though other formats would also work)



# Large scale data ingestion

Computational efficiency for a given training

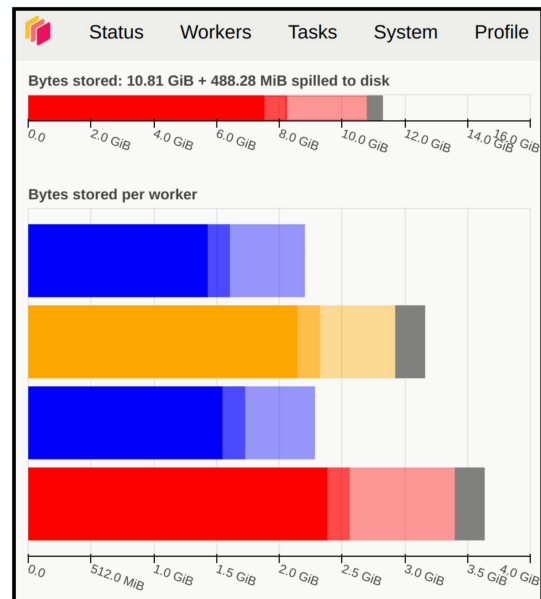
# Iterating through large datasets

Tasks on HPCs require strict resource management

- For large datasets (>100GB) loading all the data into memory is not feasible
- Simplest solution in pytorch is to ***iterate through each file***
- ***Drawbacks***: no full-array preprocessing and no memory control

We are experimenting with *dask*

- Provides a ***single array view*** over all files
  - **Chunking per file**
  - **Automatic memory management**
- Allows exact slicing, padding and format validation



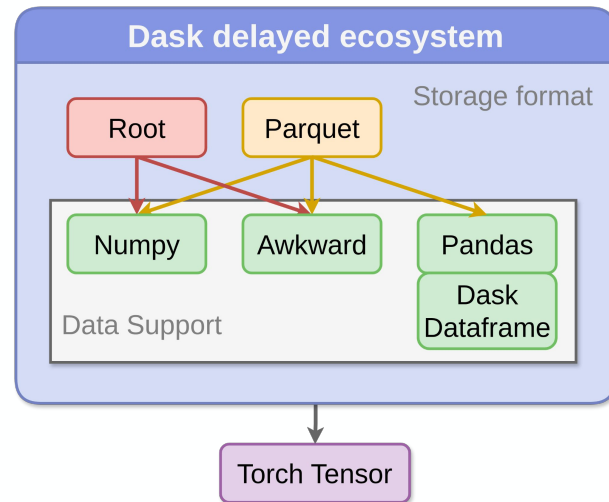
# ML dataloading with dask-awkward

## dask-awkward

- Bridges awkward arrays with dask delayed
- Account for irregular features per event
- *Alternatives*: dask-numpy or dask-pandas

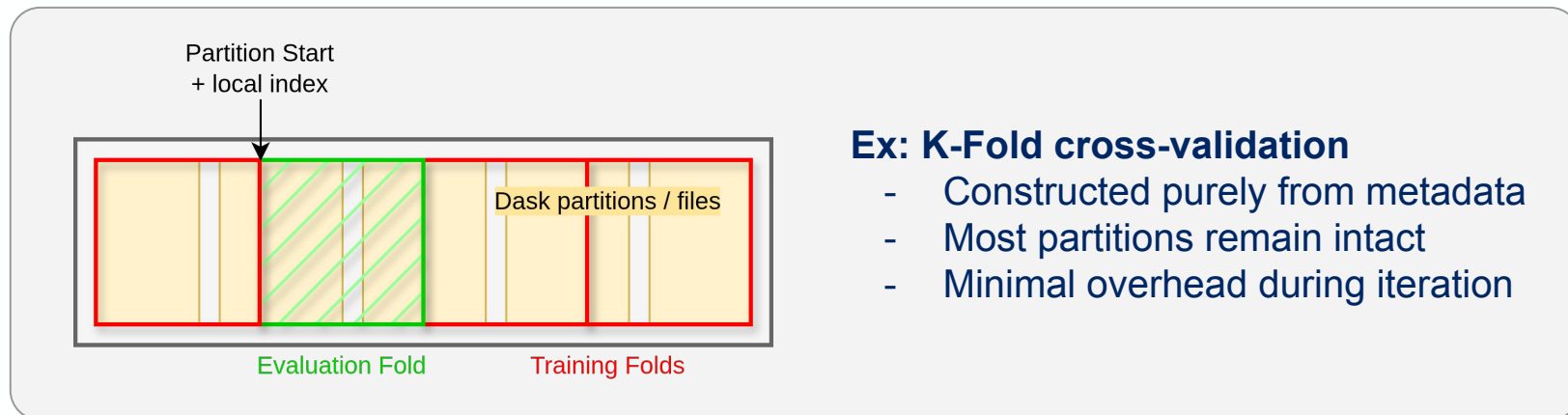
## NEEDLE Data Utility Library

- Compatible with root Tree structures and schemas for nested arrays
- Columnar access of branches
- Convert to standardized torch tensors
  - Events x Particle x Features
- Ready-to-use pytorch datasets
  - Parallel IO with dask or parallel processing with torch

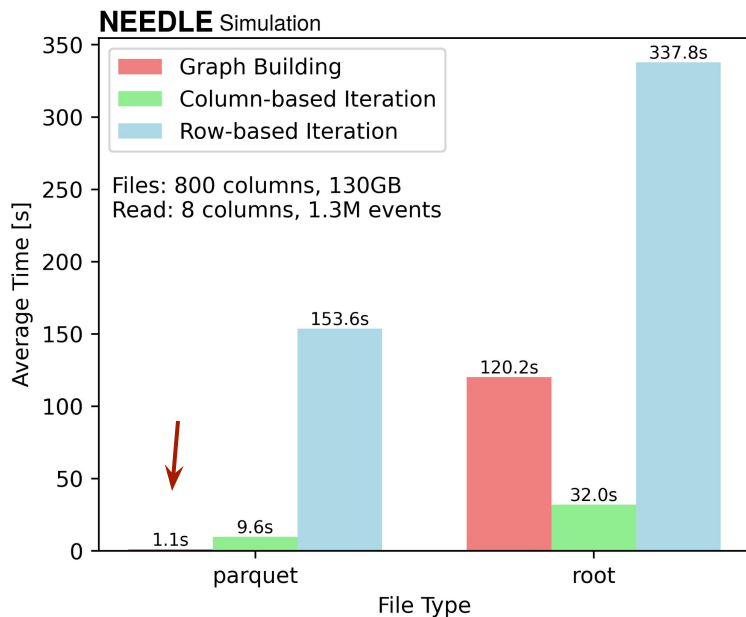


# Making the most out of dask

- File boundaries remain relevant when manipulating the data
- ***Processing along partitions*** for maximum efficiency
- We added common functionalities such as k-folds, shuffling and more



# Dask performance benchmarks



**parquet** `dask_awkward.from_parquet()`

- Fast graph building from stored metadata
- Columnar (fast)

Works nicely within the dask ecosystem

**root** `uproot.dask()`

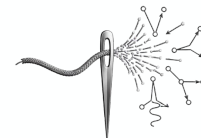
- No metadata → build the dask graph from scratch each time
- Row-based (slower)

Convenience of dask not worth the performance loss

# Summary: Practical NSBI with NEEDLE

- Build a common framework together with the NSBI community – help is welcome
- Let you focus on your flavor of NSBI with minimal constraints
- Available as a **utility library** or **full workflow manager**

## *Project timeline*

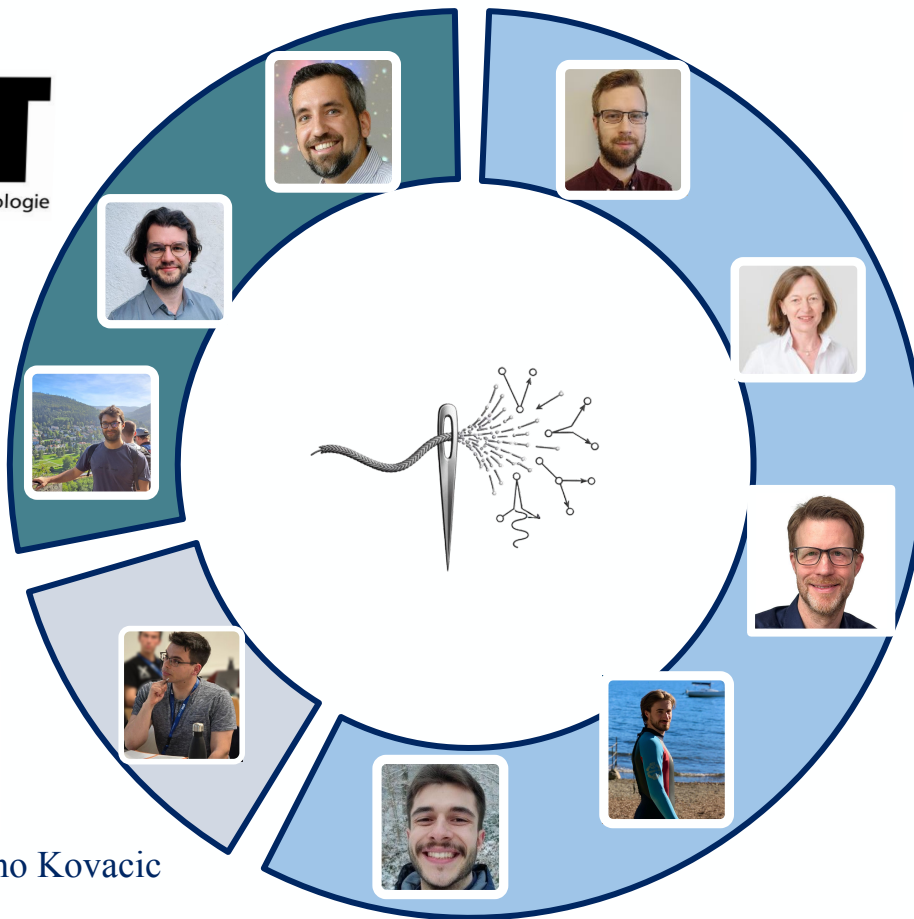


An abstract graphic consisting of white lines and dots on a dark blue background. The lines are horizontal on the left and branch out to the right, ending in small white circles. The overall shape is reminiscent of a stylized tree or a network diagram.

Backup



Felix Kahlhöfer  
Lena Rathmann  
Niklas Reus  
Nicolò Trevisani  
Kylan Schmidt



Ulrich Husemann  
Stephen Jiggins  
Judith Katzy  
Stefan Katsarov  
Levi Evans



ALICE



Nino Kovacic

## Orchestration-land (NEEDLE)

```
1 model: lightning.LightningModule = hydra.utils.instantiate(  
2     self.config.models,  
3     dataset_config=self.config.datasets,  
4 )
```

## Config-land

```
1 defaults:  
2   - models: mock_transformer  
3   - datamodules: padded  
4   - datasets: fair_universe  
5   - trainers: default
```

```
1 _target_: ml.lightning.models.mock_transformer.MockTransformerModule  
2 factor: 0.1  
3 patience: 10  
4 init_lr: 0.001
```

```
1 _target_: ml.lightning.data.padded_datamodule.PaddedDataModule  
2 multiprocessing_type: torch  
3 batch_size: 1024  
4 n_folds: 10
```

## Implementation-land (User)

```
1 class MockTransformerModule(L.LightningModule):
```

# Ready-to-use pytorch+dask datasets

## Current state

### Memory-aware pytorch datasets

- Eager in-memory
- Parallelized reading with dask and single-threaded processing
- Single-threaded reading and parallelized processing with torch

## Work in progress

- Share workers between dask and torch to parallelize whole chain
- Expand datasets to support NestedTensors

Share these utilities as standalone python package