Delivery of columnar data to analysis systems

B. Galewsky[A], R. Gardner[B], M. Neubauer[A], J. Pivarski[C], L. Gray[E]
I. Vukotic[B], G. Watts[D], M. Weinberg[B]

A. University of Illinois at Urbana-Champaign, B. The University of Chicago,

C. Princeton University, D. University of Washington, E. Fermilab

HSF Event Delivery Working Group Kickoff Meeting

8/5/19

Delivering data to new analysis platforms

First introduced in Feb 2018 whitepaper: delivery of data from lakes to clients (insulate from upstream systems, remove latencies, reformat, filter & possibly accelerate)

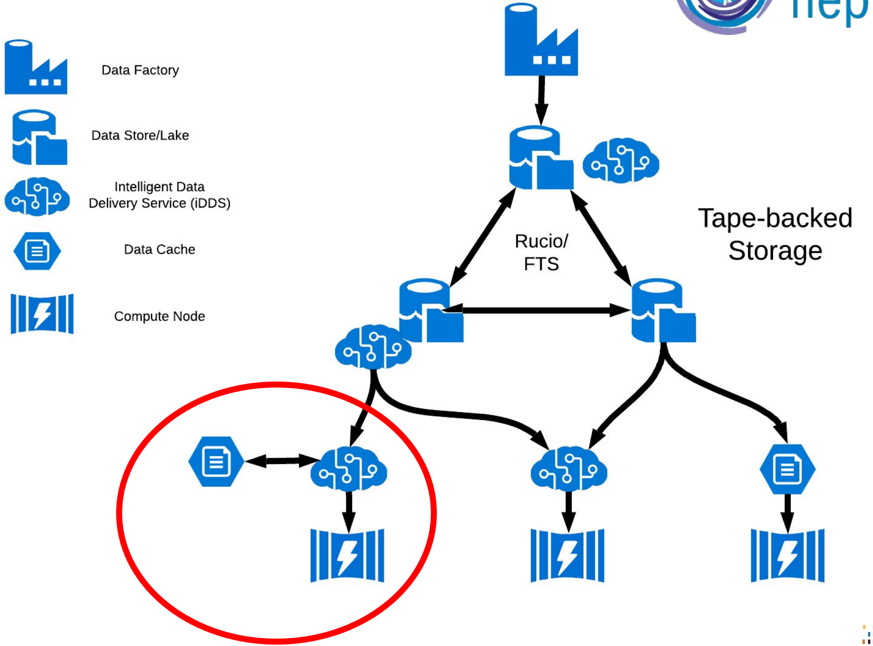
Nicely aligned with existing Event Service/ESS concepts

iDDS introduced by IRIS-HEP as framework to support development of both ideas

iDDS/ESS – (BNL, Wisconsin) – focuses on integration with PanDA WMS

iDDS/ServiceX – (Chicago, Illinois) – focuses on integration with Rucio and reformatting for pythonic tools & **endstage analysis systems**

Goal: reusable, **containerized service components** (e.g. caching, filters, reformatters), **interchangeable** between PanDA & generic analysis platforms (Coffea, Spark, Dask, ..)



IRIS-HEP R&D efforts in DOMA & Analysis Systems

- DOMA/AS groups interested in ***R&D*** for data delivery for analysis of columnar and other data formats
- Supports multiple input types (xAOD, flat ntuples, …) and common data mgt (Rucio, XCache)
- Utilize industry standard tools (GKE, on-prem Kubernetes, Helm, Kafka, Redis, Spark, …)
- Reproducible, portable deployments

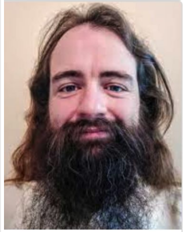
Marc Weinberg
University of Chicago

Ben Galwesky
National Center for Supercomputing Applications

Mark Neubauer
University of Illinois at Urbana-Champaign

Gordon Watts
University of Washington

Ilija Vukotic



Jim Pivarski
Princeton University

Lindsey Gray
Fermilab

Rob Gardner
University of Chicago

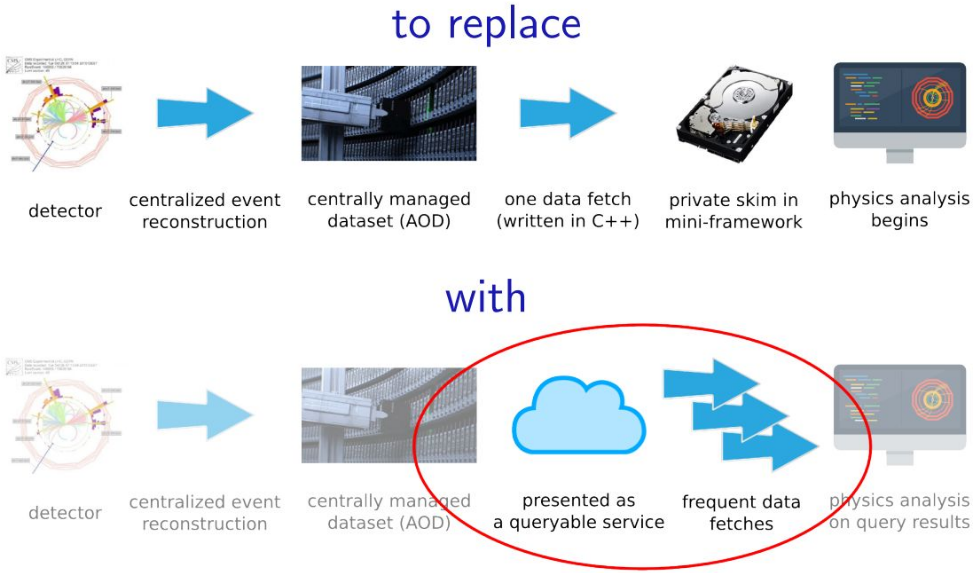
Classic analysis workflow

xAOD ⟶ DAOD ⟶ flat ntuples ⟶ skimmed ntuples ⟶ histograms/plots

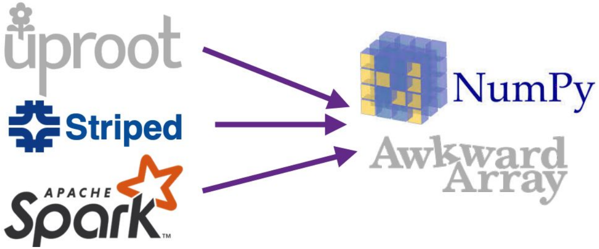
- First two formats are prescribed, but enormous variation in after that

Standard analysis might have “primary ntuple”

- Write ntuplization code to dump xAOD into flat trees with specialized objects
- Submit jobs to HTCondor by hand
- Primary ntuple then skimmed/trimmed; some data replicated (multiple times)
- Selections/cutflows baked into analysis
- Adding new variables means throwing previous skim, replicating everything


to replace
detector
centralized event reconstruction
centrally managed dataset (AOD)
one data fetch (written in C++)
private skim in mini-framework
physics analysis begins
with
detector
centralized event reconstruction
centrally managed dataset (AOD)
presented as a queryable service
frequent data fetches
physics analysis on query results

Columnar data R&D efforts

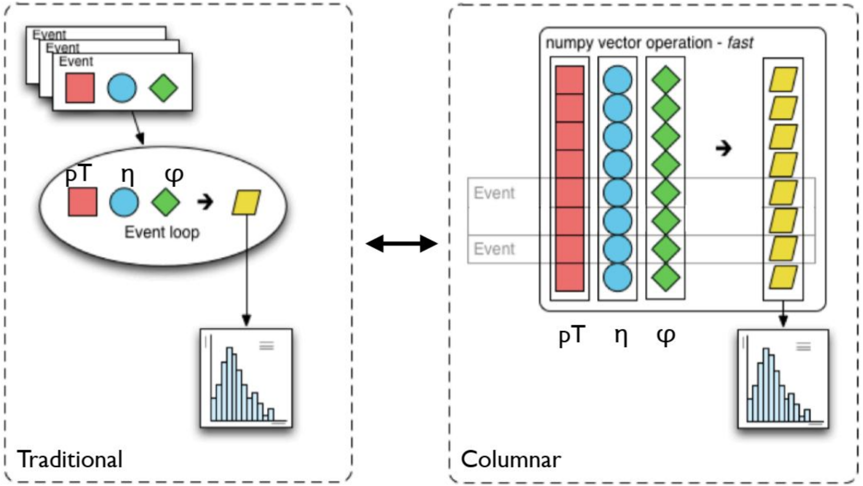


Recast data so attributes of physics objects grouped into contiguous columns, rather than grouping by event and then object

- Much more efficient for processing!
- Updating event content (or corrections) can be done by adding columns to existing data
- Can cache only necessary data for computation;
  No longer need to load entire event in memory

However, this is a significant change for analyzer

- New syntax can be simpler, more expressive
- Imagine analysis code with no for() loops…



Loop–less array programming

Can do all kinds of stuff with optimized linear algebra computations

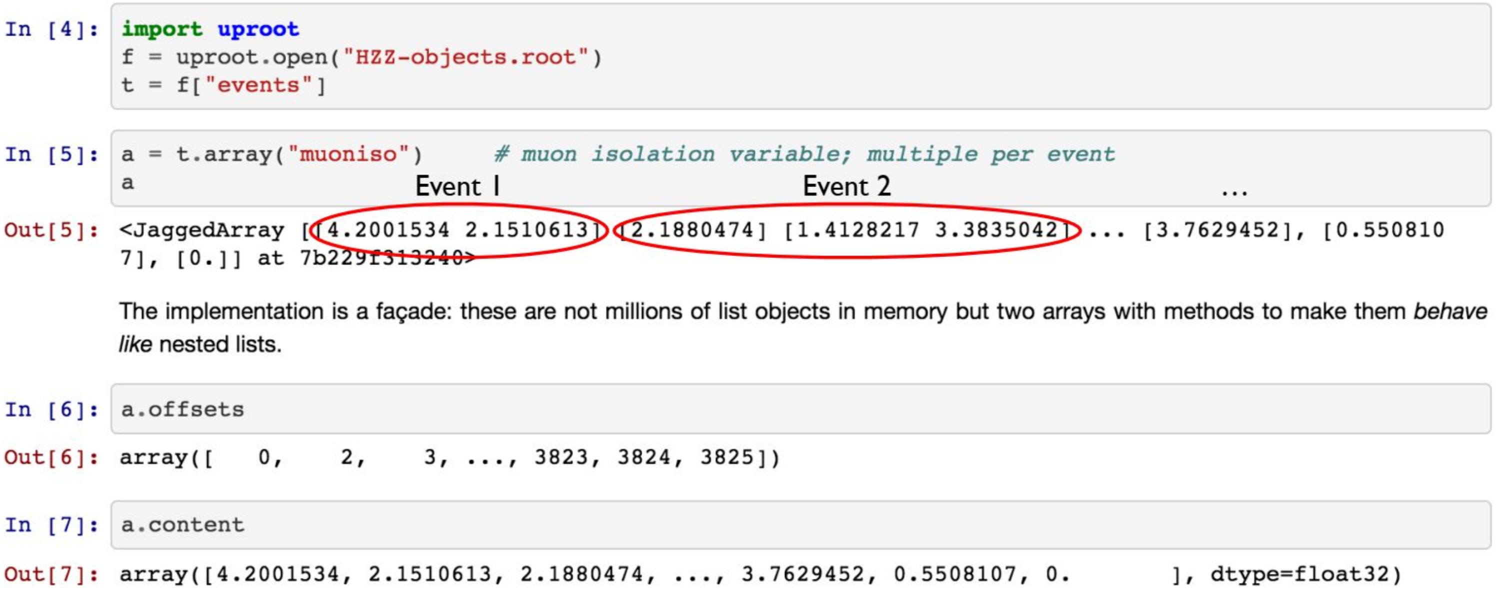
- Multidimensional slices of data
- Element–wise operations (e.g. `muons_pz = muons_pt * sinh(muons_eta)`)
- Broadcasting (e.g. `muon_phi - 2 * pi`)
- Event masking, indexing, array reduction, etc.

But we don’t have a simple rectangular arrays

- Nested variable–size data structures everywhere in HEP
- Jagged arrays handle this with two 1D arrays:
  - First array contains long list of values, one per object
  - Second array contains breaks that give event boundaries

Loop–less array programming

- But this is shown to the user as a list containing lists of various lengths:



```
In [4]: import uproot
        f = uproot.open("HZZ-objects.root")
        t = f["events"]

In [5]: a = t.array("muoniso")     # muon isolation variable; multiple per event
        a

Out[5]: <JaggedArray [[4.2001534 2.1510613] [2.1880474] [1.4128217 3.3835042] ... [3.7629452], [0.550810
        7], [0.]] at 7b229f313240>
```

The implementation is a façade: these are not millions of list objects in memory but two arrays with methods to make them *behave like* nested lists.

```
In [6]: a.offsets

Out[6]: array([    0,     2,     3, ..., 3823, 3824, 3825])

In [7]: a.content

Out[7]: array([4.2001534, 2.1510613, 2.1880474, ..., 3.7629452, 0.5508107, 0.       ], dtype=float32)
```

ServiceX goals

Adding components to an overall iDDS ecosystem. Input-agnostic service to enable on-demand data delivery.

Tailored for **nearly-interactive,** high-performance **array-based analyses**

- Provide uniform interface to data storage services; users don't need to know how or where data is stored
- Capable of **on-the-fly data transformations** into variety of formats (ROOT files, HDF5, Arrow buffers, Parquet files, ...)
- Pre-processing functionality: Unpack compressed formats, filter events in place, project data columns

Support for columnar analyses. Start from any format, extract only needed columns

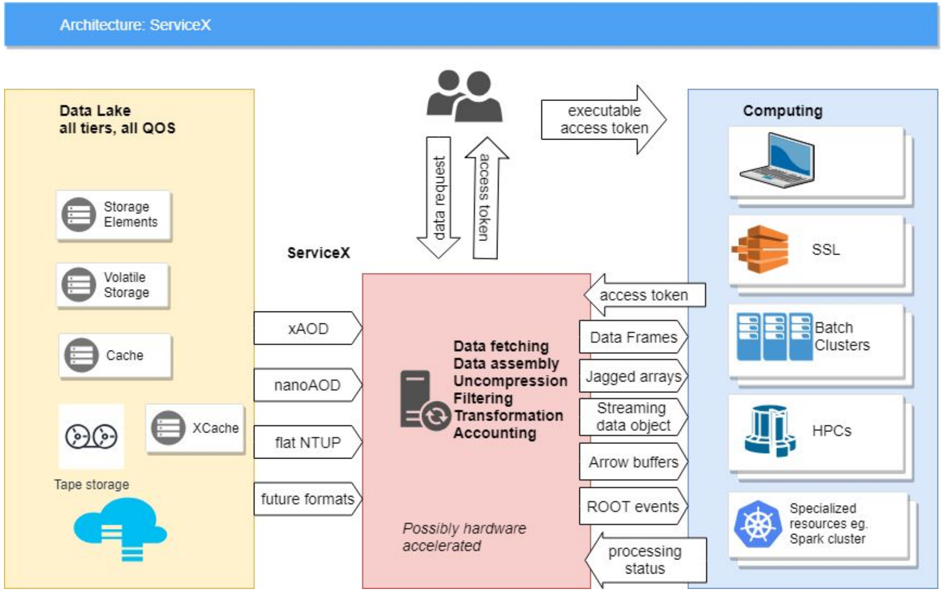
ServiceX components

Users specify needed events/columns and desired output format

- Use metadata tags (real/sim data, year, energy, run number, …)
- Any required preselection

ServiceX

- Queries backend (Rucio) to find data
- Gives unique token to identify request
- Access data from storage through XCache
- Validates request
- Perform data transformations
- Keeps track of data delivered and processed; ensures all events processed



ServiceX implementation

System designed to be modular

- Can switch out modules to transform different types of input data, swap schedulers, …

Implemented as central service in Kubernetes cluster on GCP

- Easy to deploy: Just use Helm chart to define plugins to run
- Will deploy on k8s-based IRIS-HEP Scalable Systems Lab (SSL) when available
- **Reproducible pattern** for deployment on Kubernetes clusters (e.g. **Tier2s**, **institutional k8s T3**?)

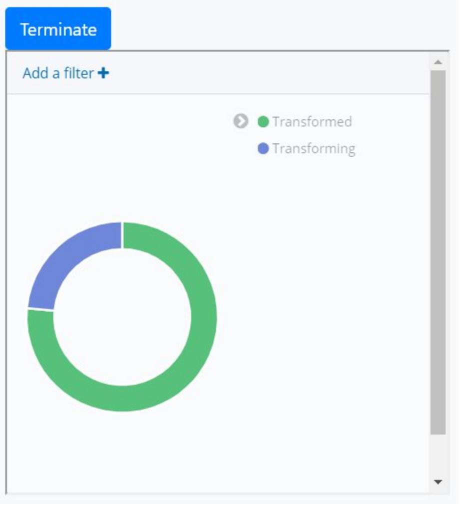
Composed of multiple deployments: API server, DID finder, transformer, Kafka manager, …

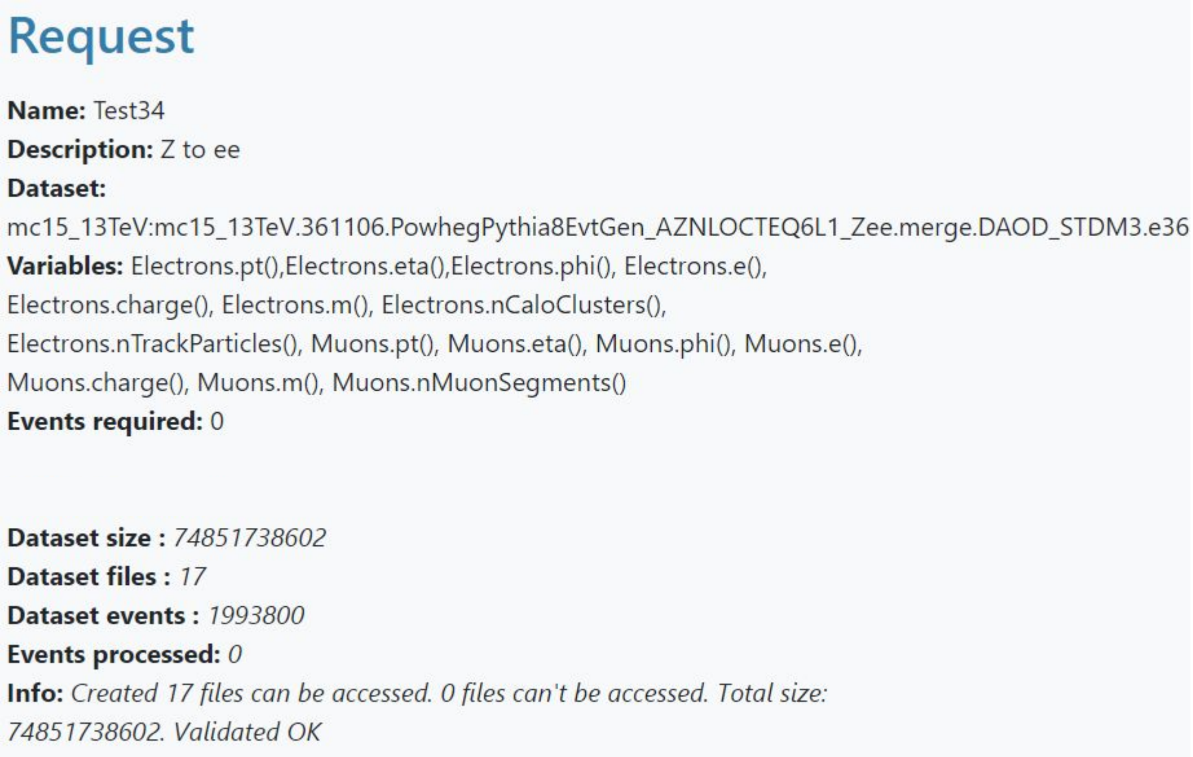
- API server: Interface for users, manages requests via DB
- DID finder: Queries data lake via Rucio, writes ROOT files to XCache
- Transformer: Takes input files from Xcache, outputs in various formats (flat trees, Awkward arrays, Parquet, Arrow tables, …)
- Kafka manager: Receives input from producer (currently transformer) and makes topics available to consumers (analysis jobs)

Early prototype up and running – UI

Create requests via web interface

Status of requests shown via Kibana





Request

Name: Test34

Description: Z to ee

Dataset: mc15_13TeV:mc15_13TeV.361106.PowhegPythia8EvtGen_AZNLOCTEQ6L1_Zee.merge.DAOD_STDM3.e36

Variables: Electrons.pt(),Electrons.eta(),Electrons.phi(), Electrons.e(), Electrons.charge(), Electrons.m(), Electrons.nCaloClusters(), Electrons.nTrackParticles(), Muons.pt(), Muons.eta(), Muons.phi(), Muons.e(), Muons.charge(), Muons.m(), Muons.nMuonSegments()

Events required: 0

Dataset size : *74851738602*

Dataset files : *17*

Dataset events : *1993800*

Events processed: *0*

Info: *Created 17 files can be accessed. 0 files can't be accessed. Total size: 74851738602. Validated OK*

Early prototype up and running – backend

Currently deployed on Kubernetes clusters on GKE

Independent auto-scaling of components based on load

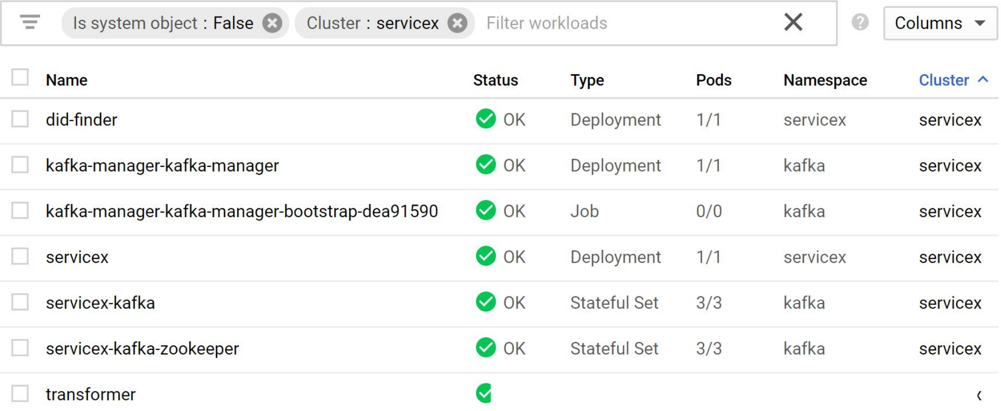
Caching of inputs using XCache

Transformation product is also persistified by Kafka

Implemented backpressure for large requests

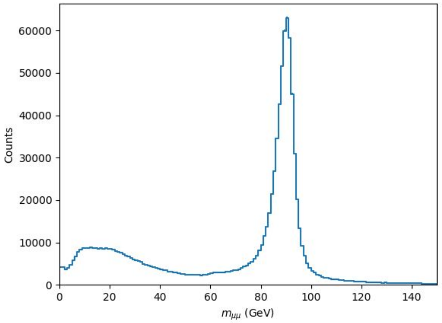
Additional K8s cluster running Spark to perform analysis step.

Uses Awkward array tools to perform calculations on columns



Is system object : False ⊗ Cluster : servicex ⊗ Filter workloads ✕ Columns ▾

□	Name	Status	Type	Pods	Namespace	Cluster ^
□	did-finder	✓ OK	Deployment	1/1	servicex	servicex
□	kafka-manager-kafka-manager	✓ OK	Deployment	1/1	kafka	servicex
□	kafka-manager-kafka-manager-bootstrap-dea91590	✓ OK	Job	0/0	kafka	servicex
□	servicex	✓ OK	Deployment	1/1	servicex	servicex
□	servicex-kafka	✓ OK	Stateful Set	3/3	kafka	servicex
□	servicex-kafka-zookeeper	✓ OK	Stateful Set	3/3	kafka	servicex
□	transformer	✓				‹
□	validator	✓				‹



Current work

- Switching transformer to compiled C++ to speed up read/write
- Testing scaling of service with multiple large concurrent requests
- Benchmarking at scale: send outputs to many analyzers (3, 10, 30, 100, …)
- Implementing swapable components: transformers, message brokers (Kafka and Redis), …
- Stability testing (e.g. randomly killing worker nodes)
- Performance optimization
    - Have moved transformer to SSL-River cluster; 256 GB / node
    - A lot of potential gain from separately optimizing each piece of the service

Future goals

- Post-transformation caching for often-reused requests (if proven beneficial)
- Coordinate development of common components for iDDS/ESS HPC production use cases
- Extending to more input formats

Backup slides

ServiceX features

- System efficiently serves only requested columns; allows simple filtering
- Each piece of system can automatically scale, independently from others
    - E.g. Cluster spins up from one to ten transformers, depending on request size
- Ultimate goal is to transform variety of input formats (xAOD, HDF5, CMS formats like miniAOD, …)
    - Want this to be useful to ATLAS community; start by implementing transformers for xAOD
    - With groundwork laid, easy to create transformers for CMS data (maybe other experiments…?)

ServiceX input architecture

- API server
  - Provides RESTful interface for users
  - Communicates with Elasticsearch (eventually RabbitMQ?)
- DID finder
  - Read from input queue: Query data lake via Rucio
  - Write filenames/locations to transform queue: Transfer ROOT files to XCache
- Dependencies
  - Data lake (read by Rucio)
  - Xcache (input for transformer)

ServiceX transform architecture

- Transformer
  - Takes input from transform queue; downloads files from Xcache
    - Container–based; one instance per input file
  - Currently focused on reading xAOD
  - Outputs in various formats: Flat trees, Awkward arrays, Parquet, Arrow tables, …
  - Efficiently serves only requested columns; allows simple filtering filtering
- Columnar cache (CC)
  - Holds transformed columns
    - K8s volume, multi–user, R/W–many
  - Becomes more useful as column reuse by users increases

ServiceX output architecture

- Cache manager
  - Listens to CC for row ranges ready to be published
  - Reads Arrow tables from CC
  - Writes to Kafka; uses request token as topic name
- Kafka
  - Receives Awkward arrays / Arrow tables from producer (currently transformer)
  - Makes topics available to consumers (analysis jobs)
  - Messaging/indexing