

# AGC scalability stress tests

---

Enrico Guiraud  
presenting contributions from various authors

September 14, 2023  
AGC demonstration, Madison (WI)

# This presentation

A quick showcase of the different ways the AGC is being used for benchmarking, testing, validating infrastructure and analysis tools.

# This presentation

A quick showcase of the different ways the AGC is being used for benchmarking, testing, validating infrastructure and analysis tools.

Measurements are not exhaustive  
and sometimes so fresh that issues are not completely understood.  
Hopefully still interesting information.

# This presentation

A quick showcase of the different ways the AGC is being used for benchmarking, testing, validating infrastructure and analysis tools.

Measurements are not exhaustive  
and sometimes so fresh that issues are not completely understood.  
Hopefully still interesting information.

Many thanks to the various authors that contributed measurements!

# The datasets

## AGC v0.1

- custom flat TTree schema, ZLIB-compressed
- 3.6 TB across 2269 files
- 5% actually read (180 GB)

## AGC v1

- CMS NanoAOD schema, ZLIB-compressed
- 1.78 TB
- 4.1% actually read (73 GB)

# Julia at UChicago

---

AGC v1.0 (incl. pyhf), from Jerry Ling (Harvard)

[https://github.com/Moelf/LHC\\_AGC.jl](https://github.com/Moelf/LHC_AGC.jl)

# Original goal of these measurements

Showcasing HEP analysis interfaces in Julia.

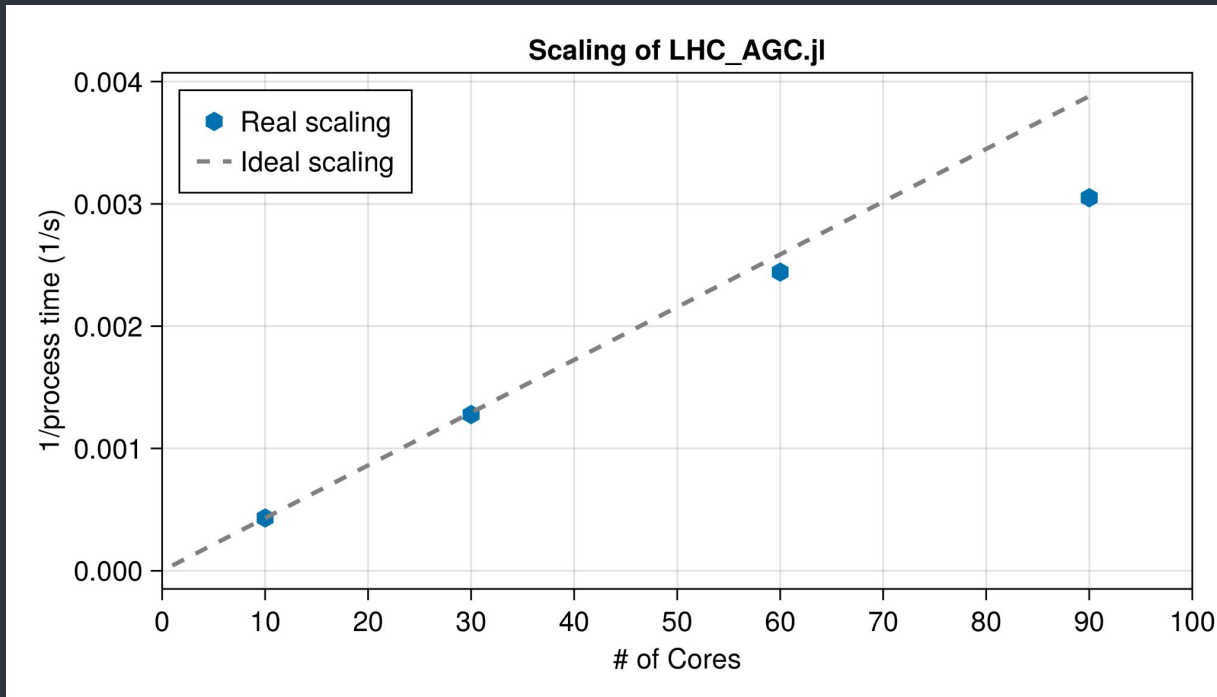
Demonstrating the Julia-HEP ecosystem can run analyses of this complexity, with scale out.

# Setup

- HTCondor cluster @ af.uchicago
- data read via network (cephfs)
- runtime includes all processing time up until statistical inference via pyhf (but not time spent waiting for nodes)



# Results



#workers	runtime (s)
10	2319
30	782
60	409
90	327

Julia can run the full AGC v1 task on a typical computing cluster in around five minutes.

# Coffea, RDF on CERN's EOS, XCache

---

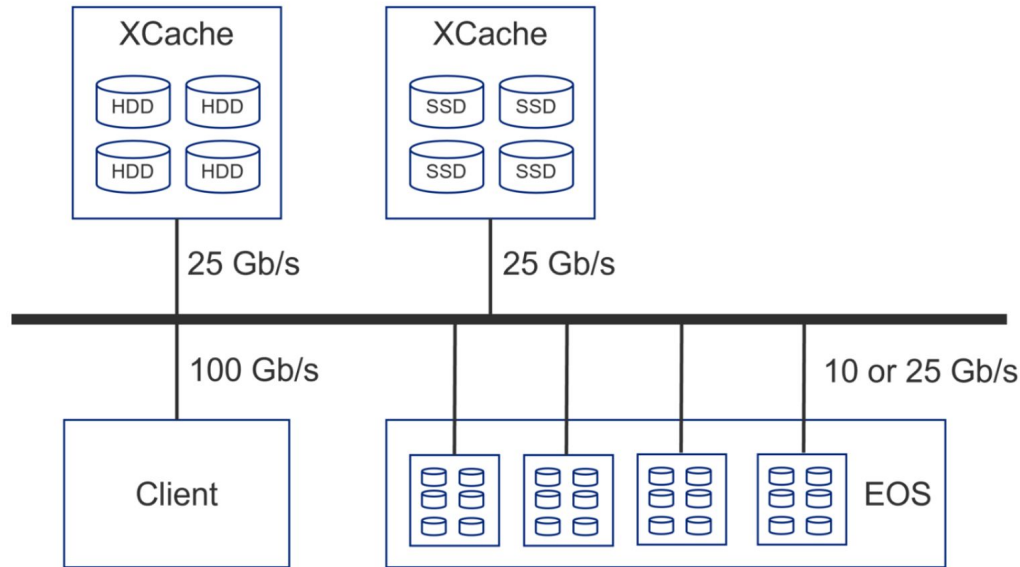
AGC v0.1, from Andrea Sciabà (CERN IT)

# Original goal of these measurements

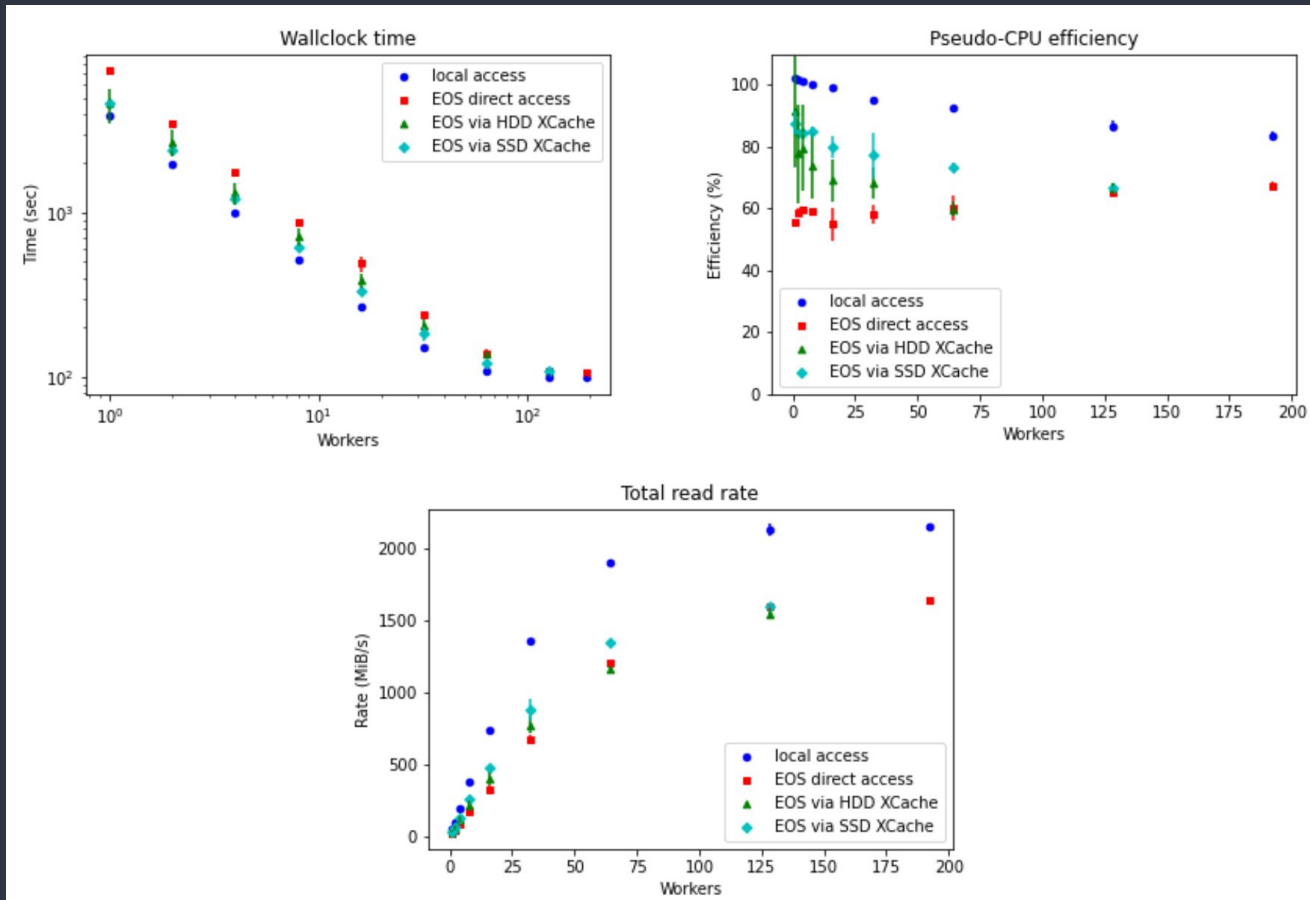
Validation of CERN's infrastructure,  
individuation of potential bottlenecks at the level of CERN  
storage and network.

# Setup

- a client node with two AMD EPYC 7702 CPUs for a total of 128 cores, 1 TB of RAM and 40 TB of SSD storage;
- two XCache instances with two Intel Xeon Silver 4216 for a total of 32 cores and 192 GB of RAM each, one instance with 1 PB in hard drives and the other with 32 TB in SSDs;
- the EOS storage system at CERN [3].



# Coffea (futures executor) 1/2



# Coffea (futures executor) 2/2

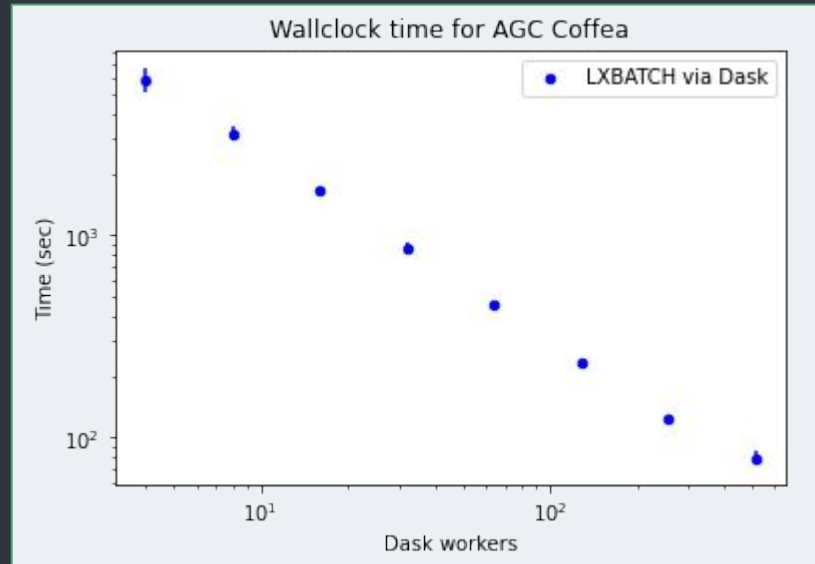
- when running on local files, the CPU efficiency remains very high, which suggests that the workload is CPU-constrained and the I/O relatively lightweight. The scalability breaks for high numbers of workers due to a considerable increase in the CPU time;
- when reading directly from EOSCMS, the CPU efficiency stays fairly constant around a value of 60%, which points at the network latency as the bottleneck. The scalability shows a similar behavior as for the previous case. When reading from EOSUSER very similar results are obtained and are not shown;
- running via an XCache instance does not bring any performance improvements with respect to direct access to EOS, although it would still be useful in case the dataset is originally hosted at a remote site; moreover, SSDs do not perform noticeably better than HDDs (in our very basic, single-user scenario).

The performance when reading via FUSE was also measured, but the total read rate saturates around 100 MB/s; as expected, using FUSE is strongly discouraged to access input data.

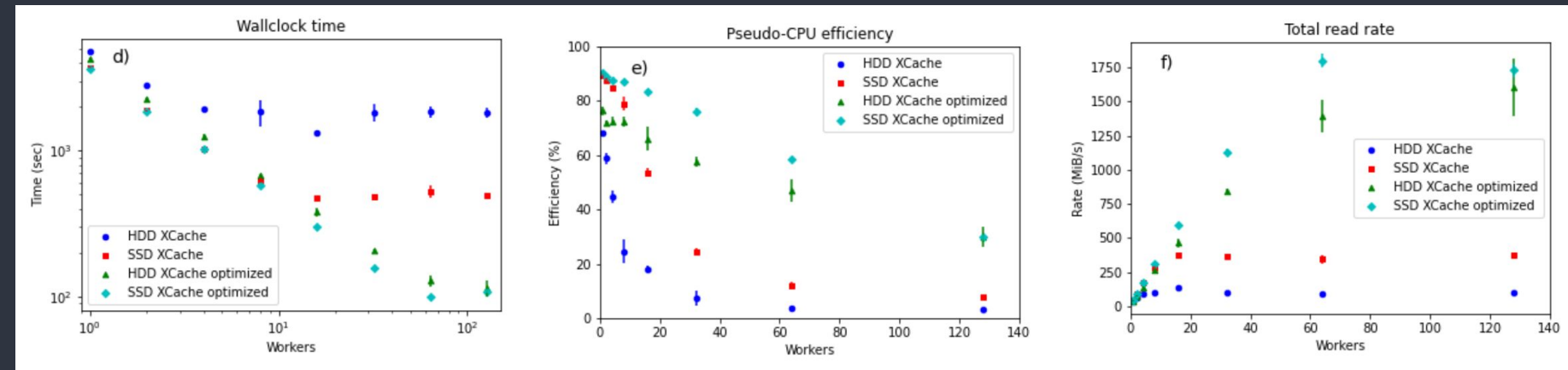
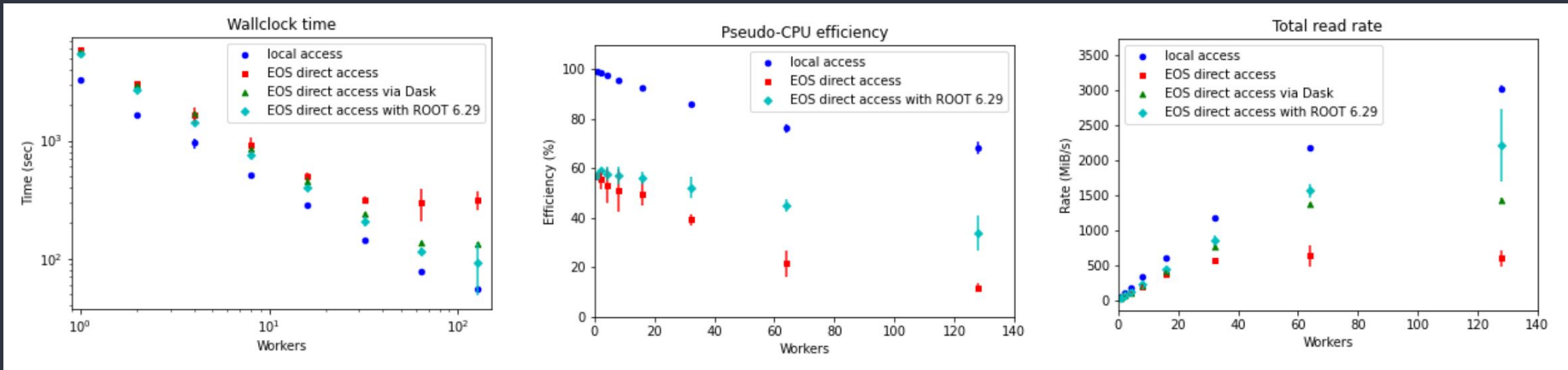
Finally, the performance metrics were measured also for direct access to the Nebraska storage via XrootD and via both a cold and a warm HDD-based XCache; not surprisingly, access via a cold cache was slower than direct access ( $\approx 610$  s and  $\approx 440$  s respectively), while a warm cache performed exactly like EOS direct access.

# Coffea (Dask executor)

- Using SWAN to run the latest version of the workload on LXBATCH HTCondor cluster via Dask
  - 4 cores per job, 3 GB per core available



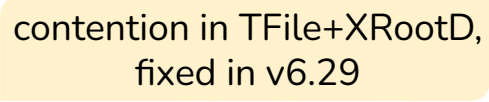
# RDataFrame 1/2 (EOS above, XCache below)





# RDataFrame 2/2

the scalability “out of the box” using ROOT 6.26 is extremely poor, but this was understood as caused by three factors:

1. by default, the XRootD client uses only one event loop thread to hand the asynchronous I/O, which creates a considerable bottleneck when many concurrent file operations are performed. Setting the environment variable `XRD_PARALLELEVTLOOP` to 10 removes this bottleneck [1];
2. by default, XRootD will multiplex several network connections from the same client process to a remote XRootD server (such as an EOS disk server or an XCache instance) into one, which is in general a good idea, but will create a bottleneck when the client process uses many threads connecting to a single server. This becomes apparent in the case of XCache, as it is a single server, but it does not affect access to EOS, where files are almost guaranteed to be spread over a large number of disk servers. The multiplexing can be disabled by the user;  
 contention in TFile+XRootD, fixed in v6.29
3. a significant lock contention affects RDF with large numbers of threads and files read via XRootD. This problem was fixed for in the ROOT library.

# Limiting factors

Running speed is limited by CPU when reading data from local SSDs.

When reading over the network, the bottleneck is likely *latency* in accessing the data:  
better async pre-fetching could help? maybe with RNTuple?

# RDF on CERN HPC cluster

---

AGC v0.1, from Vincenzo Padulano (ROOT team)

<https://github.com/root-project/analysis-grand-challenge>

# Original goal of these measurements

Validation of the scaling of distributed RDataFrame,  
demonstration of RDF's API,  
individuation of potential bottlenecks at the level of ROOT.



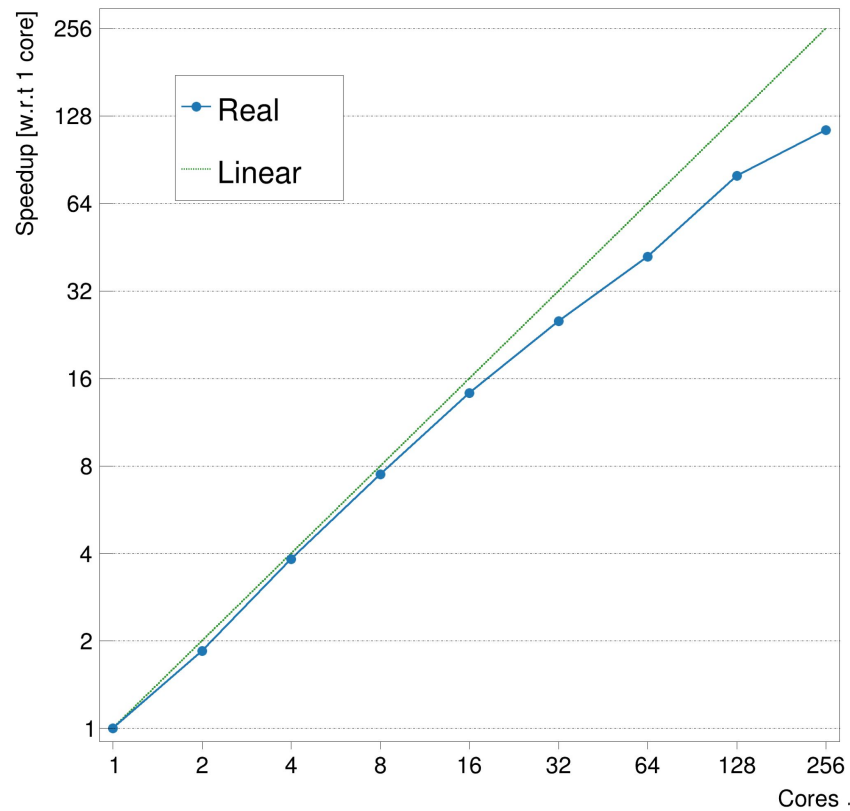
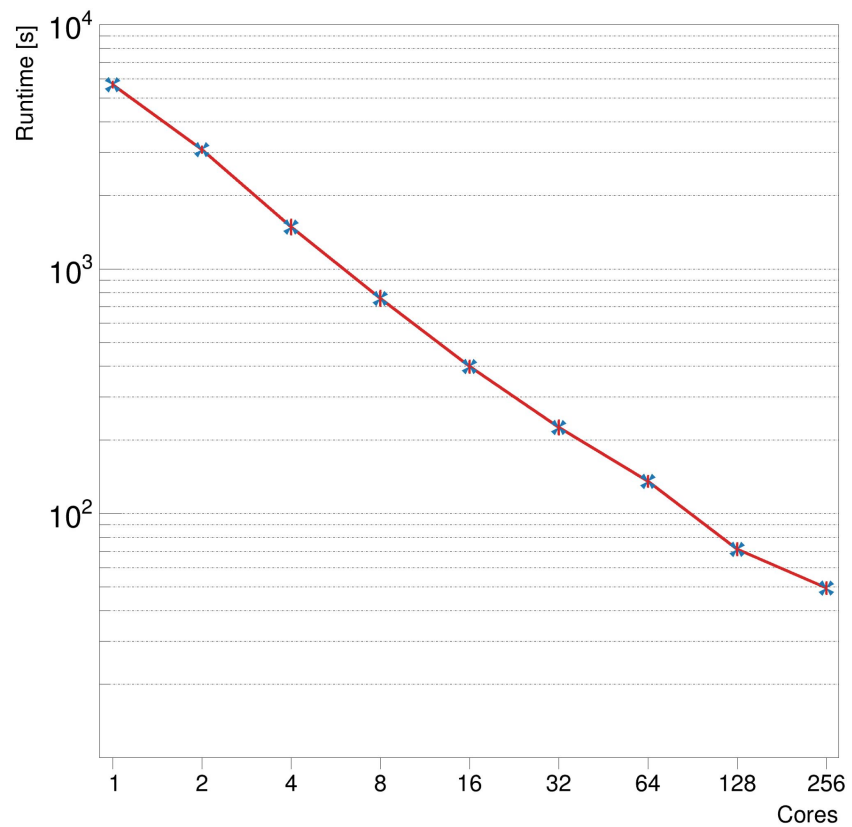
HPC cluster at CERN ([link](#)). 8 computing nodes, each with:

- ▶ CPU: 2x AMD EPYC 7302 16-Core.
- ▶ Memory: 512GB DDR4 3200Mhz.
- ▶ Network: 10Gbit ethernet connection.

Slurm jobs via Dask, exclusive access to the nodes, data is read from EOS via xrootd.



# Plots





# Raw numbers

<b>total_cores</b>	<b>time (mean)</b>	<b>speedup</b>	<b>standard error of mean</b>
1	5679	1	146.56
2	3078	1.85	87.40
4	1486	3.82	100.45
8	758	7.49	52.90
16	399	14.24	21.68
32	225	25.25	14.41
64	135	42.04	6.84
128	71	79.68	3.94
256	49	114.71	2.60



# Data processing throughput

- ▶ AGC v0.1
  - dataset size = 3.6 TB
- ▶ Actual size read is 5%
- ▶ Maximum data processing throughput (end to end)
  - w.r.t. actual data read: 3.6 GB/s or 14 MB/s/core



# Coffea at UNL coffea-casa - Flatiron

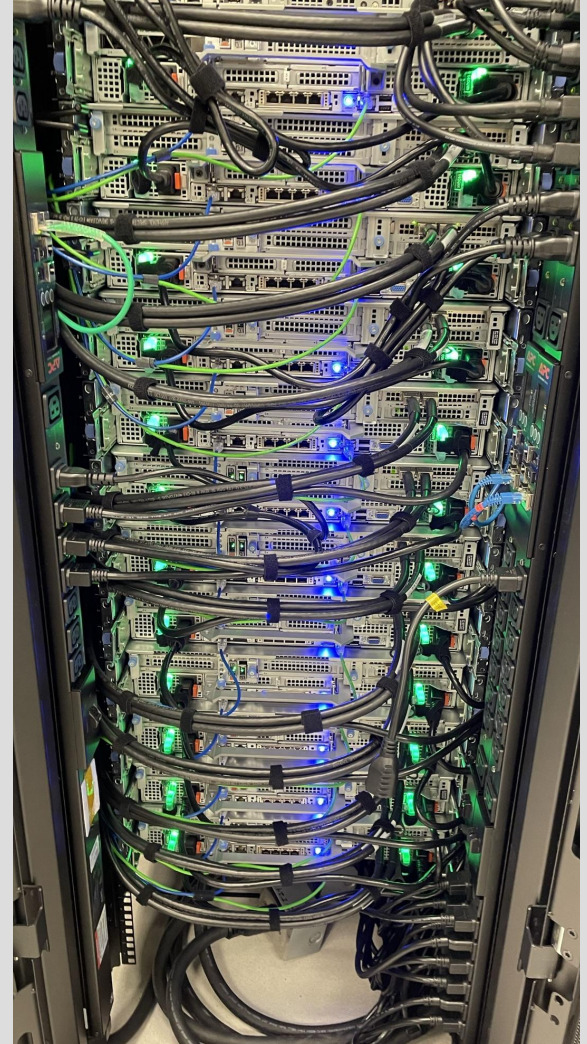
---

AGC v1.0, from Alexander Held, Oksana Shadura and others (IRIS-HEP)

<https://github.com/iris-hep/analysis-grand-challenge>

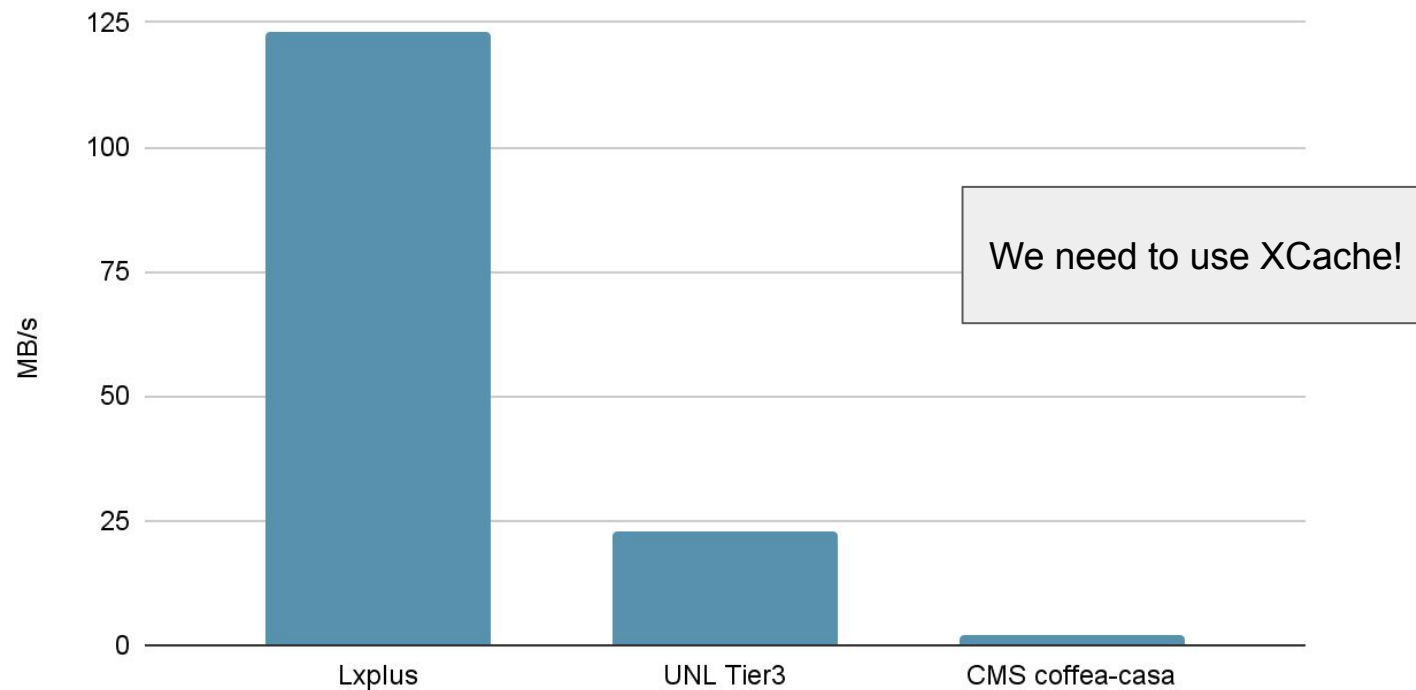
# UNL Coffea-Casa Hardware – Flatiron

- 12 Dell R750 Servers, 512 GB Ram, 10 3.2 TiB NVMe Drives  
Intel(R) Xeon(R) Gold 6348 CPU @ 2.60GHz (56 threads/CPU, 2 CPU per node)
- 2 x 100Gbps Networking, Calico + BGP
- Alma Linux 8.8, Kubernetes (v1.26.2)
- Ceph-Rook Filesystem @ 183 TiB
- 2 x P100, 1x V100 GPUs for Triton
- Ceph @ 8.7 PiB Usable for Tier2 CEPH storage
- 4.8TiB @ SAS HDDs for XCACHE
- Cert-manager, Dex, external-dns, sealed-secrets, Traefik, CVMFS

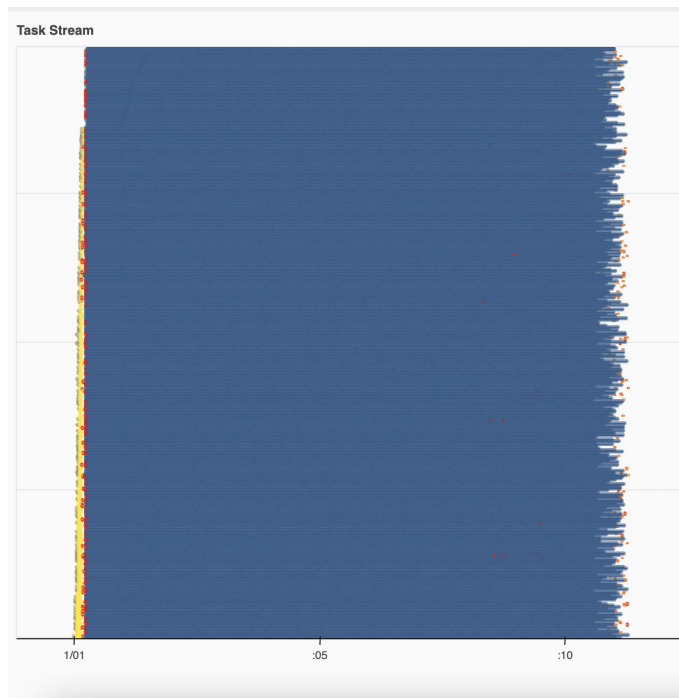


# Data access from Flatiron to EOS Public

Data access- EOS Public (xrdcp)

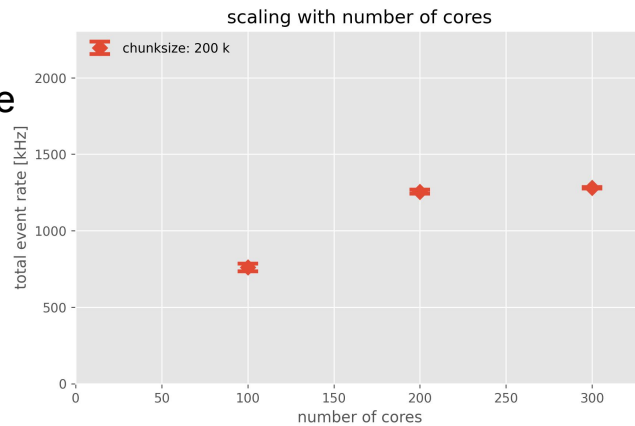
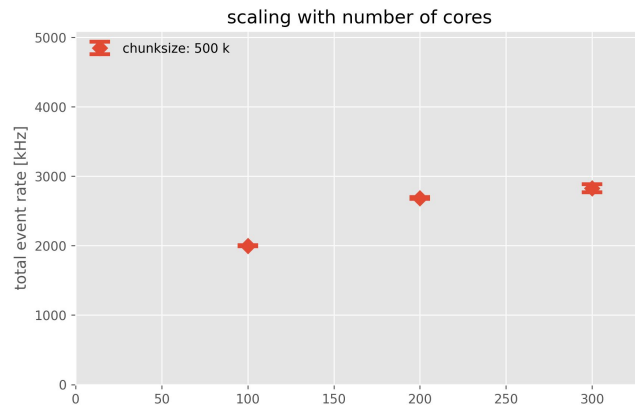


# Horizontal scaling at UNL: AGC v1 and pre-v2



AGC v1

main (pre-v2),  
including inference



# Coffea at UChicago ATLAS

## coffea-casa

---

AGC v1.0, from Ilija Vukotic (UChicago)

<https://github.com/iris-hep/analysis-grand-challenge>

# UChicago AF XCache Hardware

- Dell R740XD
- 2x Intel Xeon Silver 4214 CPU @ 2.20GHz
  - 24 cores / 48 hyperthreads
- 192GB RAM
- 24x 1.5TB NVMe
- 2x 25Gbps Network

# UChicago AF Hardware

- 16 IRIS-HEP nodes, Dell R750
  - Intel(R) Xeon(R) Gold 6348 CPU at 2.60 GHz
    - 56 cores / 112 hyperthreads
  - 384GB RAM
  - 10x 3.2TB NVMe
    - 2x dedicated to /scratch on each node (6.4TB/node)
  - 2x 25Gbps Networking
  - CentOS 7
- Mix of V100, A100 and older GPUs available
- Kubernetes v1.25.12
- Ceph v16.2.6 via Rook
  - 3.6 PB raw / 1.2 PB usable (3x replication)
  - 366 OSDs, mix of NVMe and HDD (separate pools)
  - Capacity pool capable of ~10GB/s reads with large files at high concurrency



# UChicago SSL(dev) Hardware

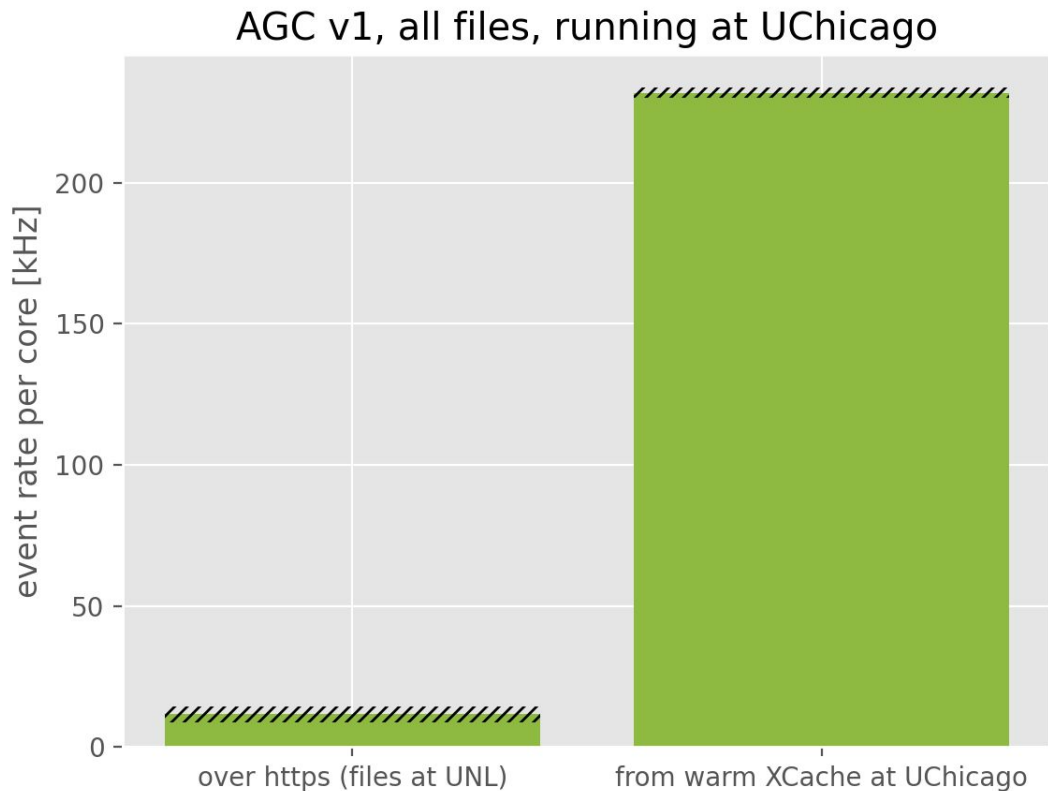
- 11 nodes, LENOVO System x3550 M5
  - Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz
    - 24 cores / 48 hyperthreads
  - 256 GB RAM
  - 2x 750G SSD
  - 10Gbps Networking
- Kubernetes v1.25.11
- Rook
  - ~1TB available



# Effect of XCache at UChicago

x20 speedup via XCache

Compare to [ACAT slides](#)  
page 9: similar event rates  
as running over local files  
on local-network CephFS.

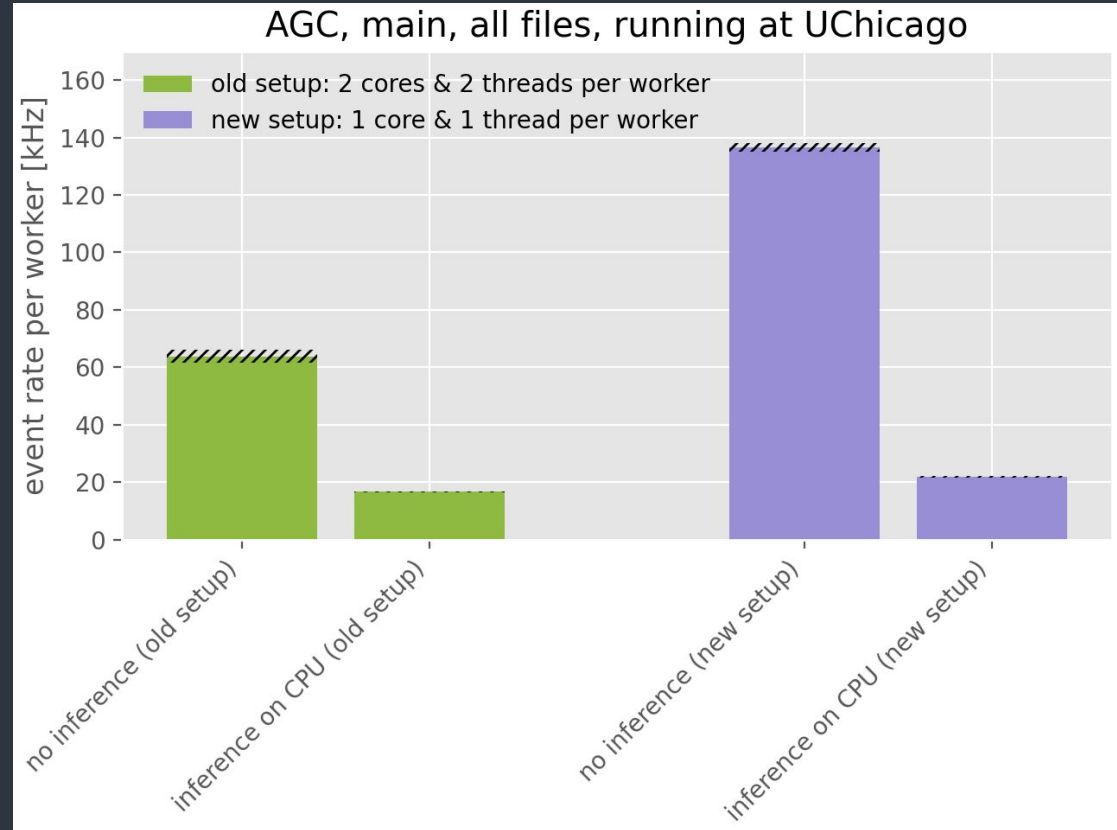


# Adding ML inference to the mix

Adding ML inference significantly **increases CPU cost** (many extra operations to calculate input variables).

**Facility configuration is very important!** Significant runtime decrease with new optimized setup.

Can also vary AGC configuration to simulate different I/O needs (see [ACAT slides](#)).



# Results

Strong dependence  
on user-defined  
chunk size

Throughput  
per worker

Chunksize	ServiceX	no ServiceX
10k	88.46s 4.04kHz	>200 minutes
20k	55.30s 5.89kHz	>100 minutes
50k	49.66s 7.46kHz	~50 minutes
100k	54.03s 7.26kHz	745.19s 36.58kHz
200k	55.96s 7.25kHz	400s 64.20 kHz

ServiceX → runtimes for  
Coffea reading pre-filtered  
data (1.5% of original  
dataset) from on-premise  
S3 instance

No ServiceX → runtimes  
for Coffea reading full  
dataset from on-premise  
XCache

ServiceX pre-filtering + caching avg: 290s, min: 270s, max: 305s

# Dask scheduling

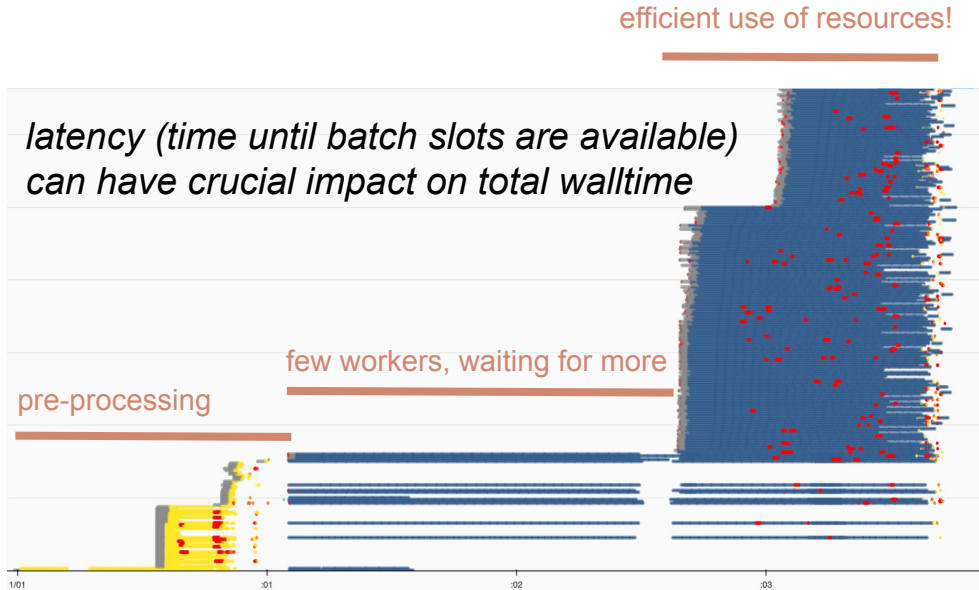
---

various AGC versions and facilities, from Alexander Held, Oksana Shadura and others (IRIS-HEP)

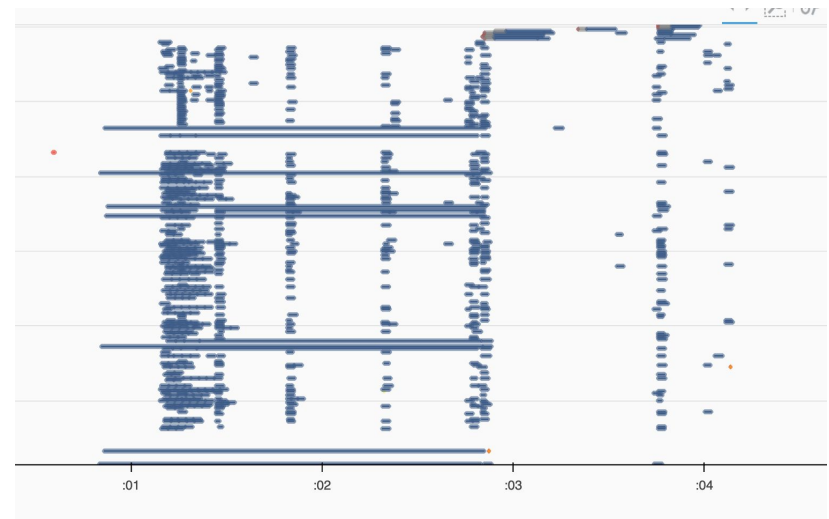
<https://github.com/iris-hep/analysis-grand-challenge>

# Task scheduling: the good, the bad, the ugly

Good scheduling efficiencies, walltime effects, scheduling inefficiencies



*occasionally observed: workers idling, very low task scheduling efficiencies!*  
-> investigations ongoing, to be understood



# Conclusions

---

# State of the art

- version 0.1/1.0 (no ML inference) runs in  $O(1 \text{ minute})$  on several facilities
- several setups achieve  $O(1 \text{ GB/s})$  of total throughput at  $O(100)$  cores
- processing rates per core between kHz and MHz depending on setup
- not quite "100 TB in 20 minutes" yet, more like 5 TB when reading all data
  - smart pre-filtering and caching speeds things up, ML inference slows them down
  - resource sharing, dataset joins, more complex schemas (physlite) will also impact performance

# Summary

The AGC is an incredibly useful integration test.

- it is being used to validate a number of different frameworks and setups
- it already helped uncover several performance issues and bottlenecks (several other that are still under investigation)
- caching via XCache and ServiceX is a common way to speed up processing
- Dask worker configuration and scheduling efficiency is critical