Object Stores for CMS data

- Object data formats provide new data management capabilities
  - Compared to current tier-based EDM file model
  - Reduce disk storage requirements for re-processing, obviate the need to define data tiers
- In a prototype framework accessing a Ceph S3 service, I/O performance is excellent
  - On-disk data and metadata volume is as expected
- To fully utilize, more software development will be needed
  - New data management service requirement: column tracking
  - Full data format requires provenance and auxiliary data handling

ACAT 2022

Nick Smith, Bo Jayatilaka, David Mason, Oliver Gutsche, Alison Peisker, Robert Illingworth, Chris Jones (FNAL)

Work supported by the US-CMS HL-LHC R&D initiative, and Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

Legend

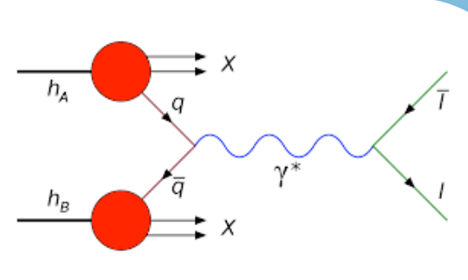
Data volume order of magnitude [bytes]

On disk

On tape

Primary dataset

Abstract, "what kind of events."
e.g. hard scatter process for simulation, trigger filter for data



Data tiers

AOD 1e6/event
Data columns pertaining to low-level reconstruction
.root 1e9/file

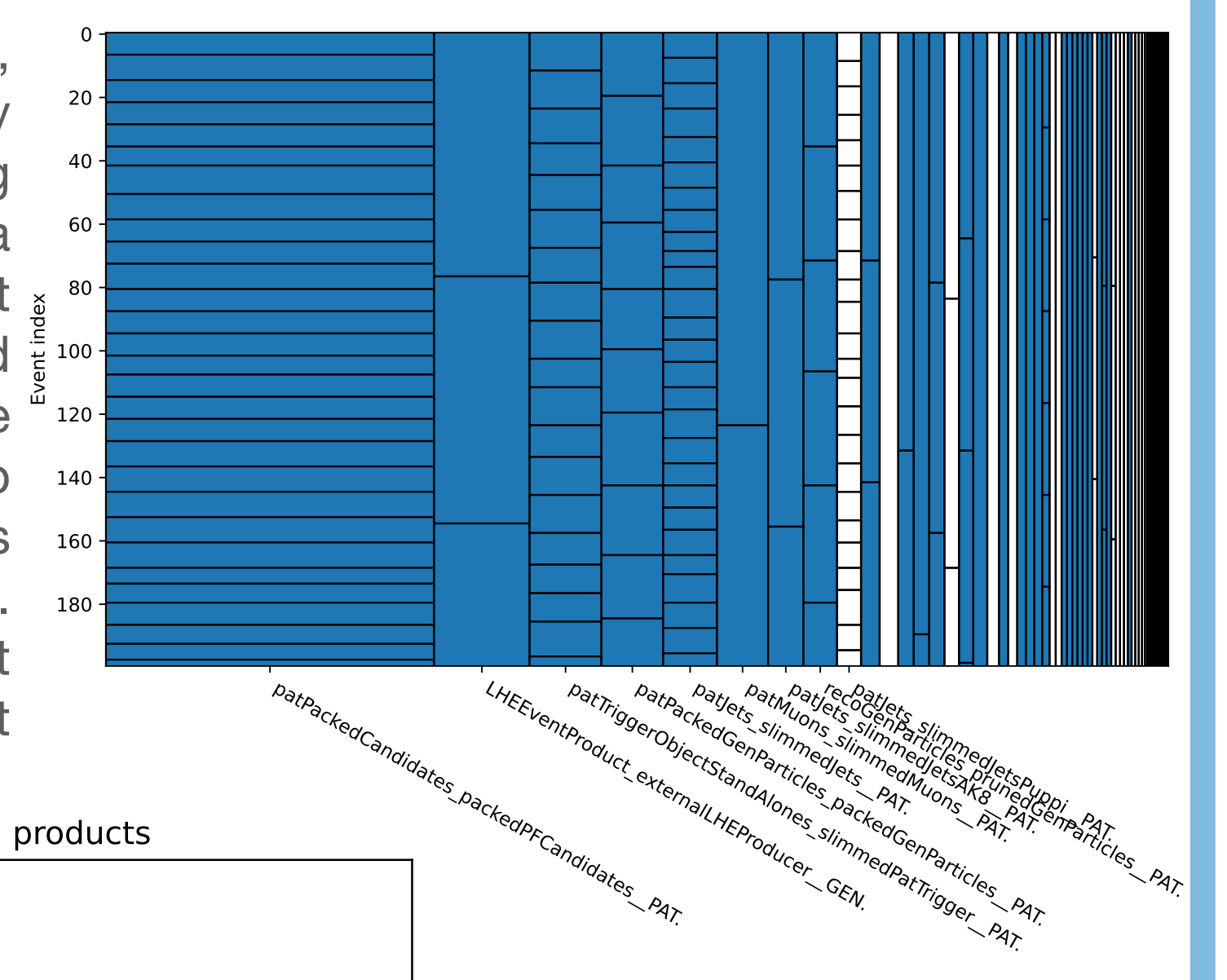
MiniAOD 1e5/event
Calibrated physics objects
Particle-flow candidates
.root 1e9/file

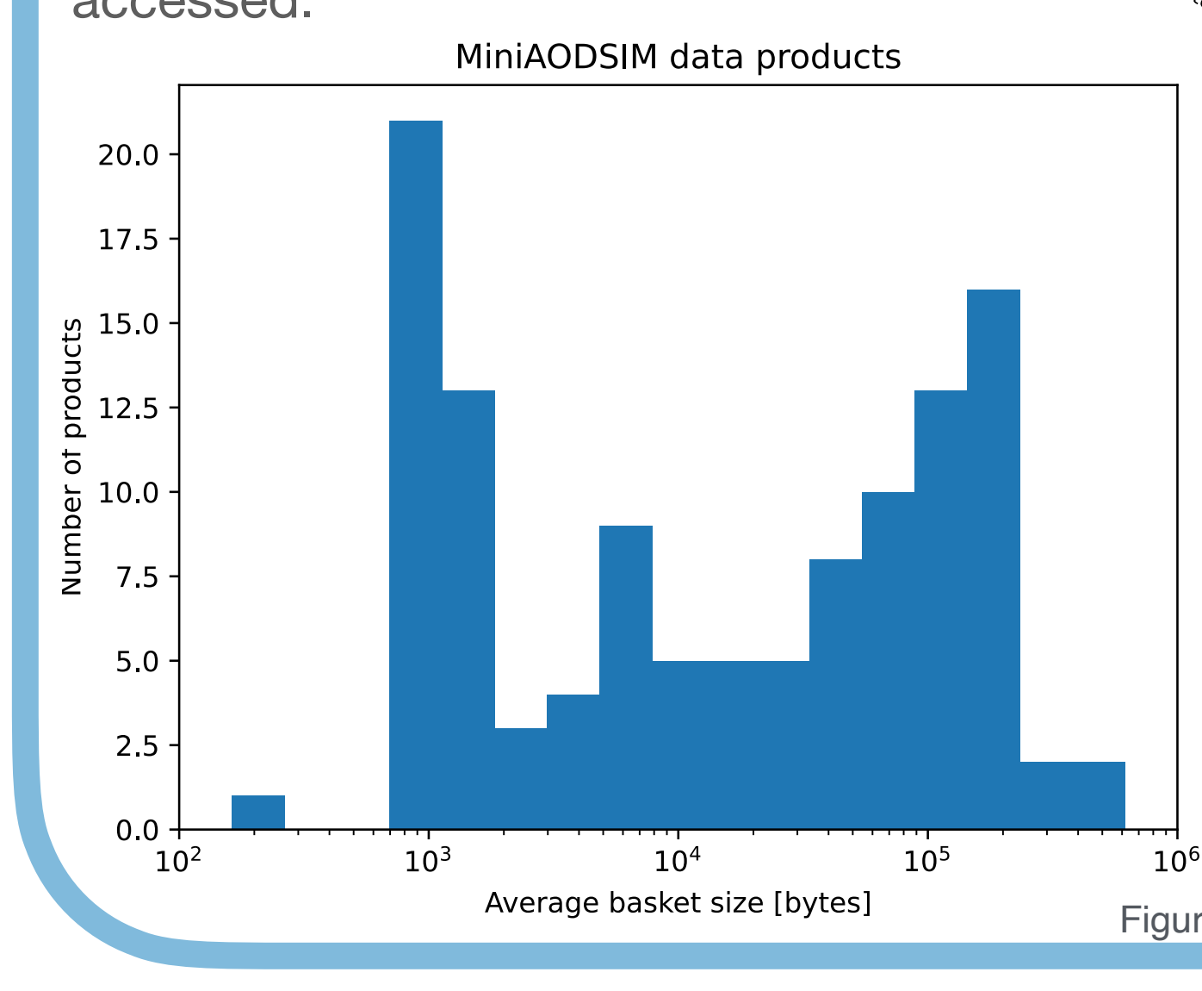
...

EDM file format

A file in ROOT format with an *Events* TTree and extra metadata. Each *data product* is a TBranch in the TTree. Some data product branch elements are further split into smaller columns for improved compression, but a product is accessed as one unit within the CMS offline software framework. TBranch objects reference several TBaskets, which contain serialized data products for a range of events.

In the diagram to the right, each TBasket accessed by a CMS workflow reading MiniAOD is represented as a rectangle, where the height is the number of events and width is proportional to the compressed bytes. The top 6 largest MiniAOD products represent half of the file size. Unfilled rectangles represent baskets that were not accessed.





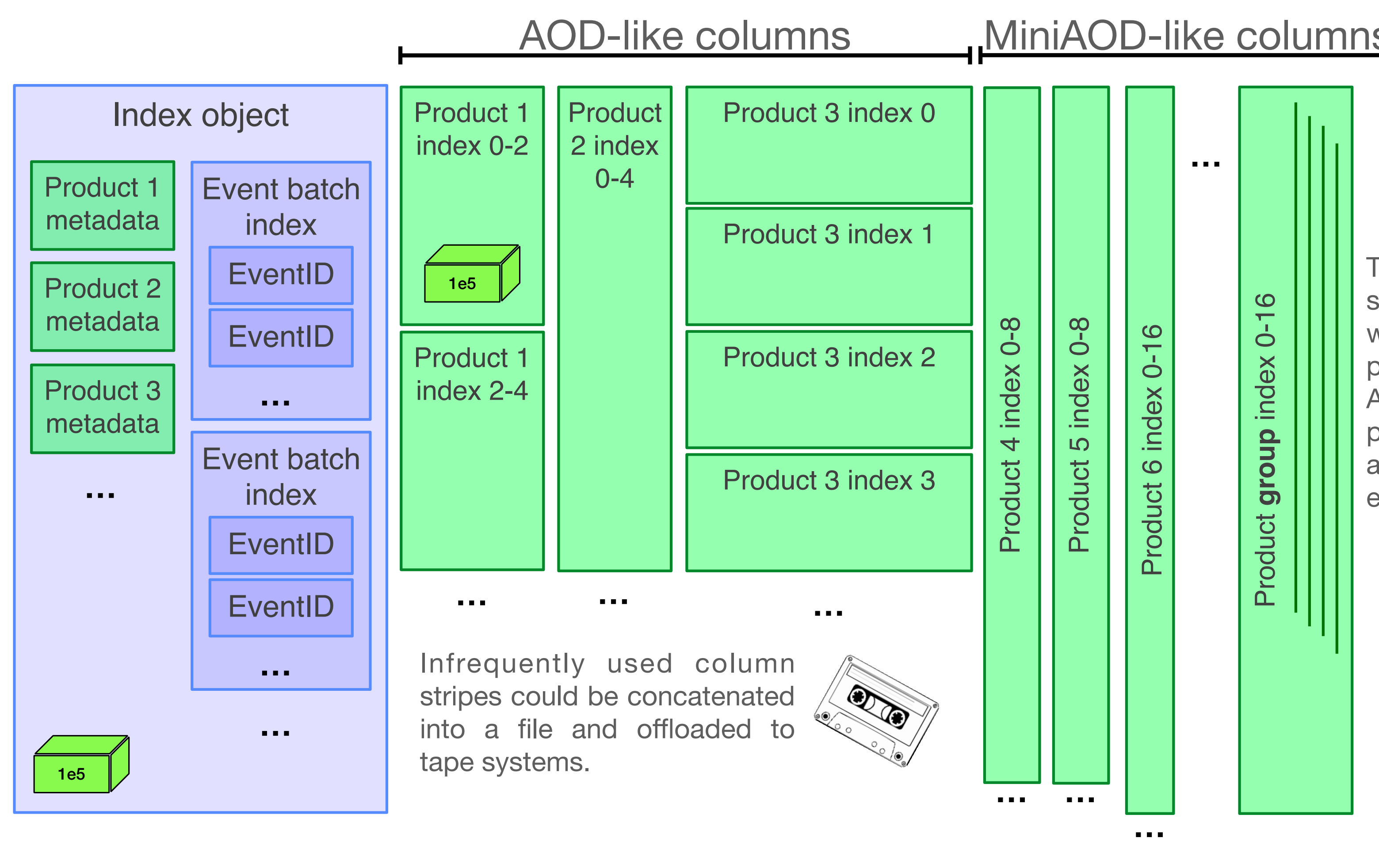
The distribution of basket sizes varies over three orders of magnitude.

A cat

Figures from CMS 2017 UL QCD simulation

Object data format

One index object holds metadata for an entire primary dataset. For each *data product*, or column of data, a *stripe* of events is written as an object in an S3 bucket. Each event data product is serialized with standard ROOT IO. The stripe size is chosen on first write, once the compressed output buffer reaches a target size (100k-1MB) and the number of events evenly divides the event batch size. The index object may eventually be replaced by a database, to more easily add and remove products. The metadata volume grows as (N products) + (M events) despite the N*M growth of the number of stripes.

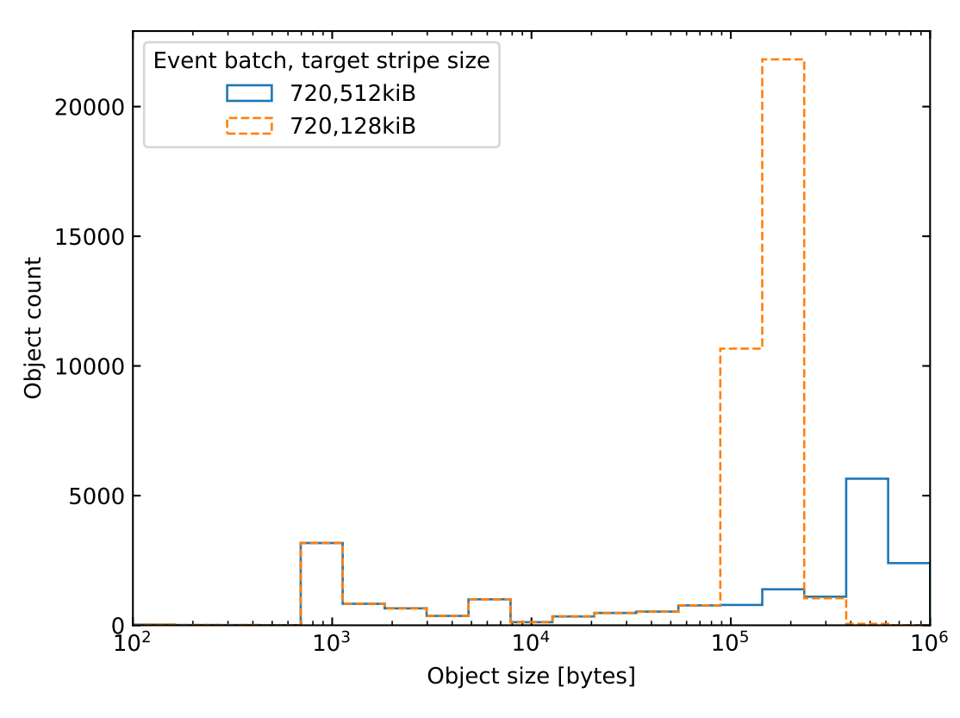


To handle very small products, we envisage using product groups. At present, small product buffers are force-flushed every event batch.

Infrequently used column stripes could be concatenated into a file and offloaded to tape systems.

The format has a total size higher than an example MiniAOD input due primarily to using ZSTD compression instead of LZMA. Small objects in Ceph incur additional storage overhead (listed in % on right) due to the object size granularity.

Format	KB per event
MiniAOD input	55.7
Objects: - event batch size 720 - target stripe size 128kiB	71.4 + 6%
Objects: - event batch size 720 - target stripe size 512kiB	70.6 + 2%



Tier-based scheme

MiniAOD Data product	KB per event	
	v1	v2
packed+pruned genParticles	5.7	5.7
slimmedElectrons	1.3	1.3
Others	48.7	48.7
Total	55.7	55.7

Significant improvements in long-term disk storage needs can be realized with an object store scheme. In a mock example, two MiniAOD versions (v1, v2) were produced from the same parent dataset, where the latter was produced to update electron data. In the object store scheme, there is no need to re-produce other (unchanged) data products. In any case, data products such as genParticles would never need reproduction.

Object store scheme

Data product	KB per event	
	v1	v2
packed+pruned genParticles	5.7	-
slimmedElectrons	1.3	-
Others	48.7	-
Updated slimmedElectrons	-	1.3
Total	55.7	1.3

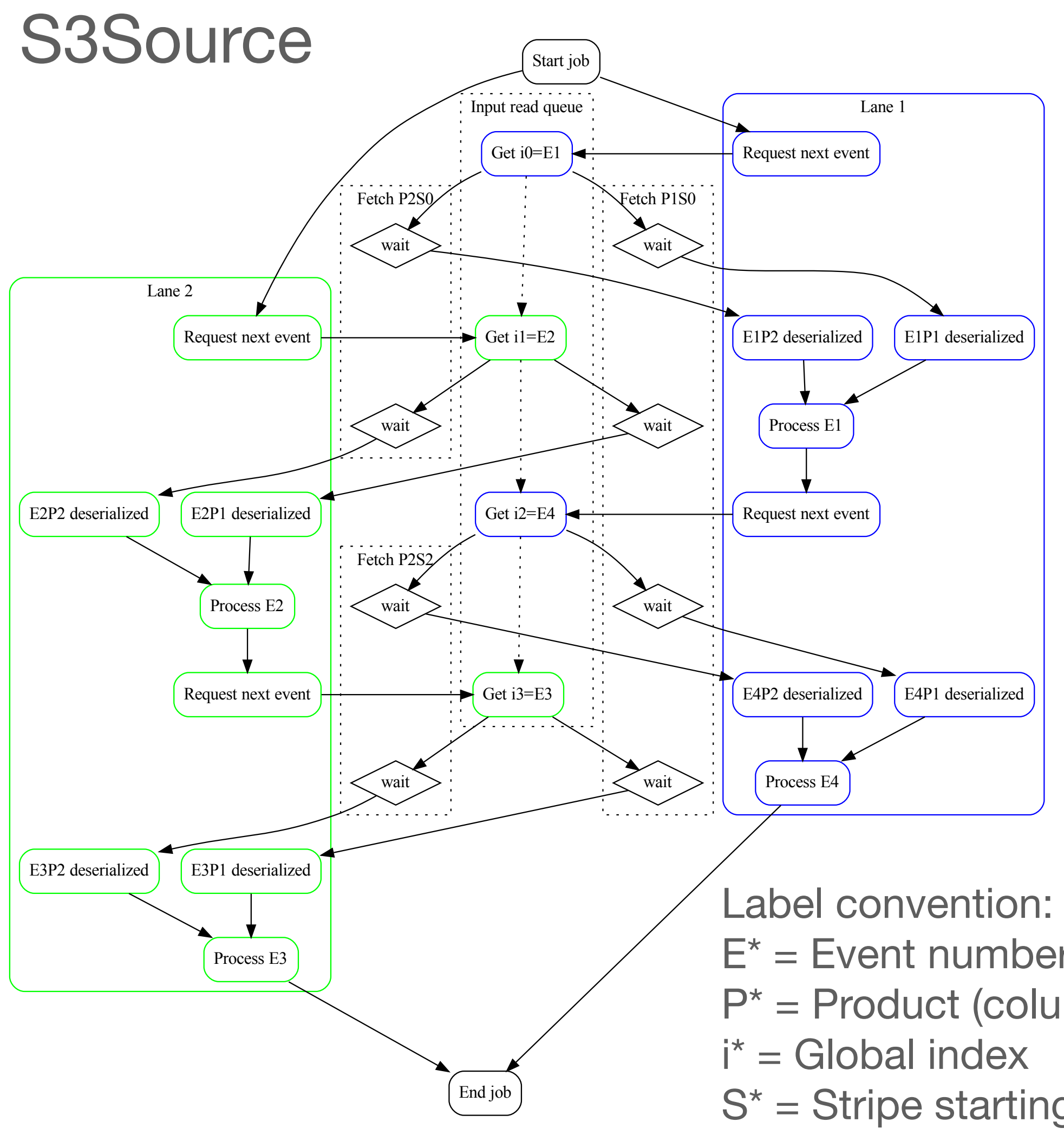
Size/event figures from CMS 2017 UL semileptonic TTBar simulation

S3 I/O software prototype

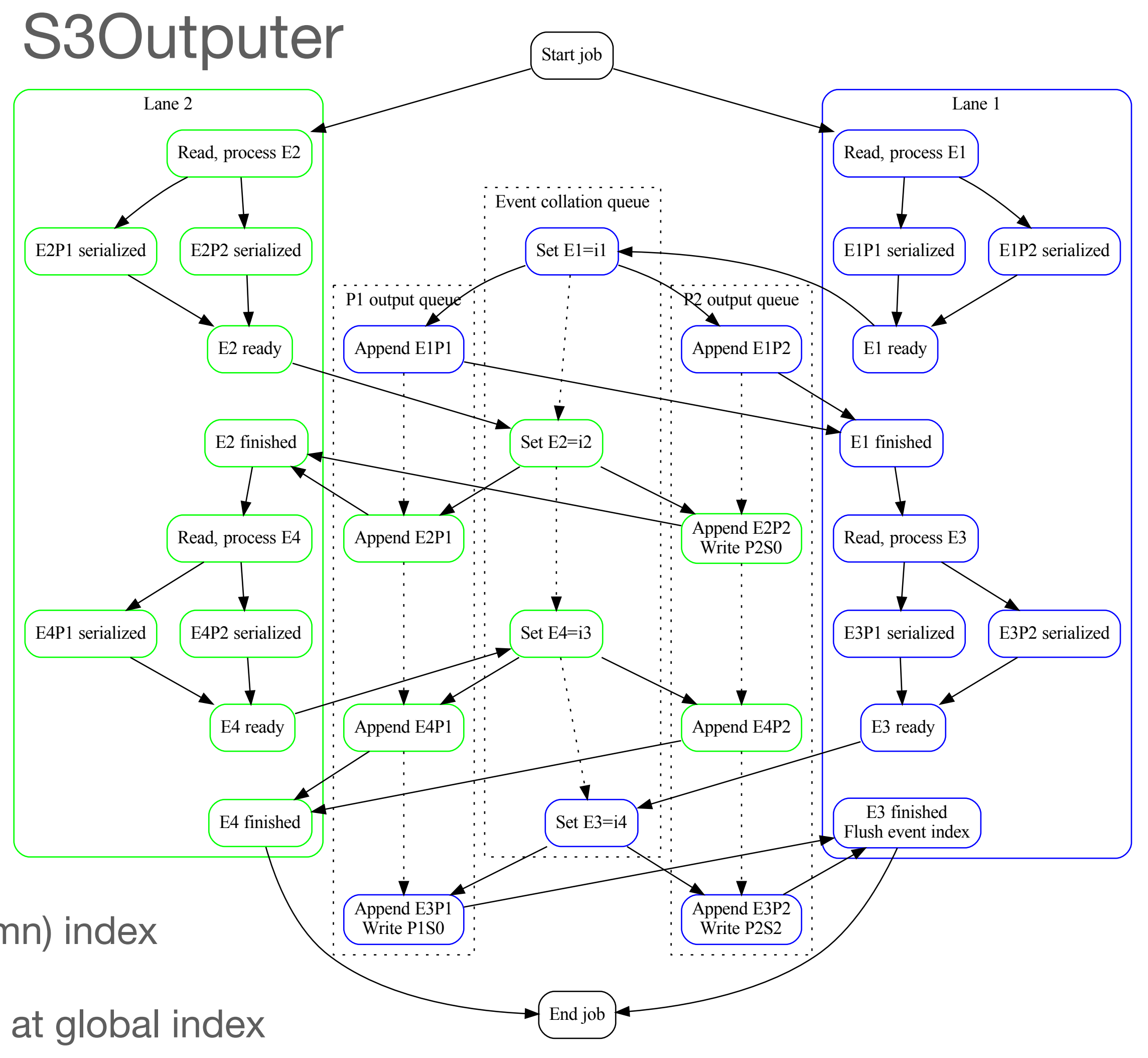
The HEP-CCE root_serialization project provides a C++ framework for performance experiments with various I/O packages from within a multi-threaded program. The program mimics behaviors common for HEP data processing frameworks. Within this framework, the S3Source and S3Outputter modules have been developed to read from and write to S3 buckets, respectively. This allows to explore the performance and parallelization behavior of writing to an Ceph S3 service in comparison to local files with ROOT or other formats.

The S3 protocol is implemented with libs3. Requests are run asynchronously in a separate thread from the Intel Thread Building Blocks (TBB) thread pool that is used for all other tasks. Network errors are handled with an exponential backoff retry mechanism. The task graphs below illustrate module behavior for a configuration with two lanes, effectively TBB task groups (blue and green nodes.) The same number of lanes, threads, and task groups is used in all tests. In this example, product 1 has stripes written every 4 events, while product two has stripes written every two events. Tasks in queues are run sequentially, with dashed arrows indicating the order. Object stripe read events gather a set of pending tasks that are released once the data has been received.

S3Source



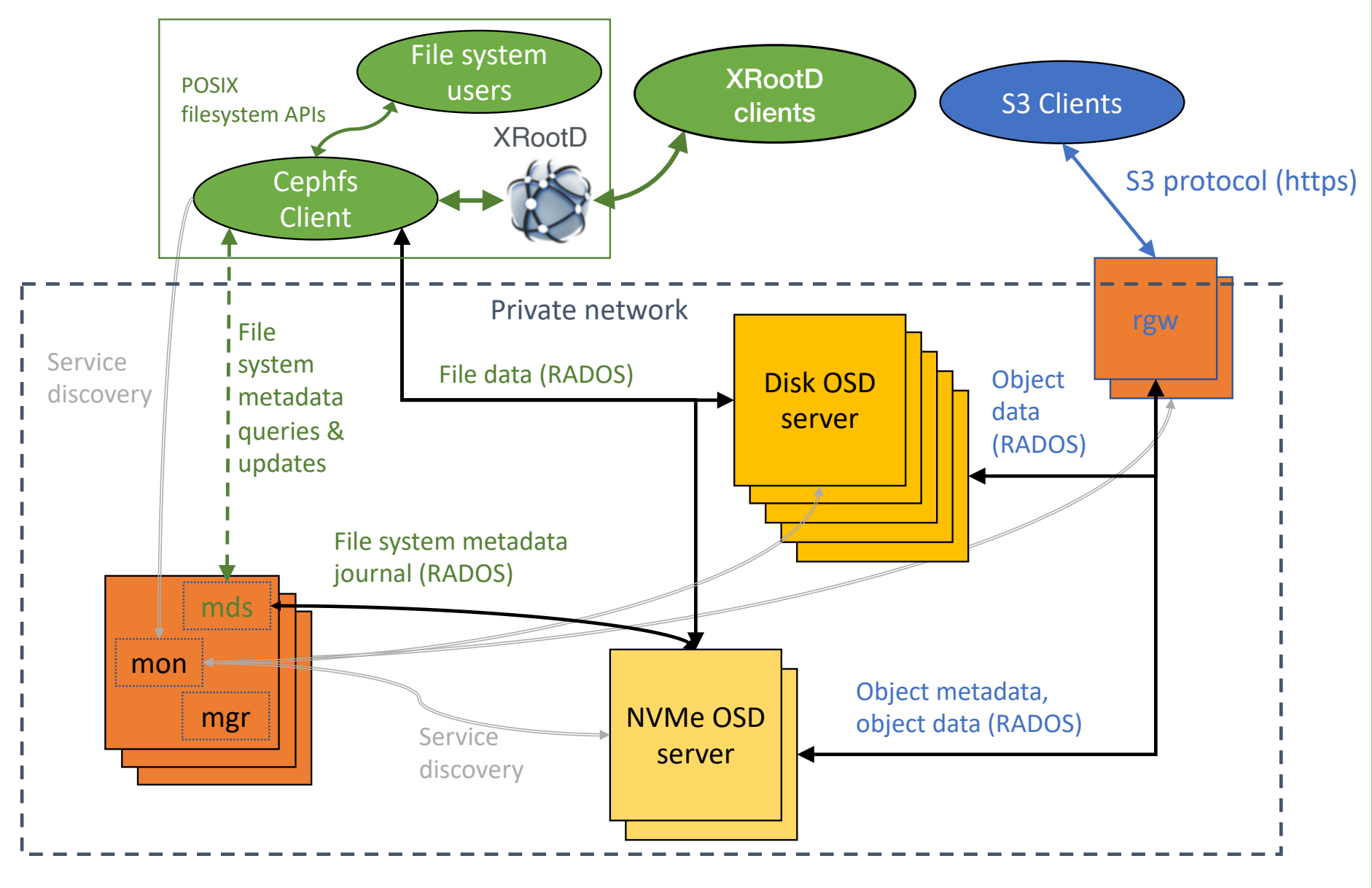
S3Outputter

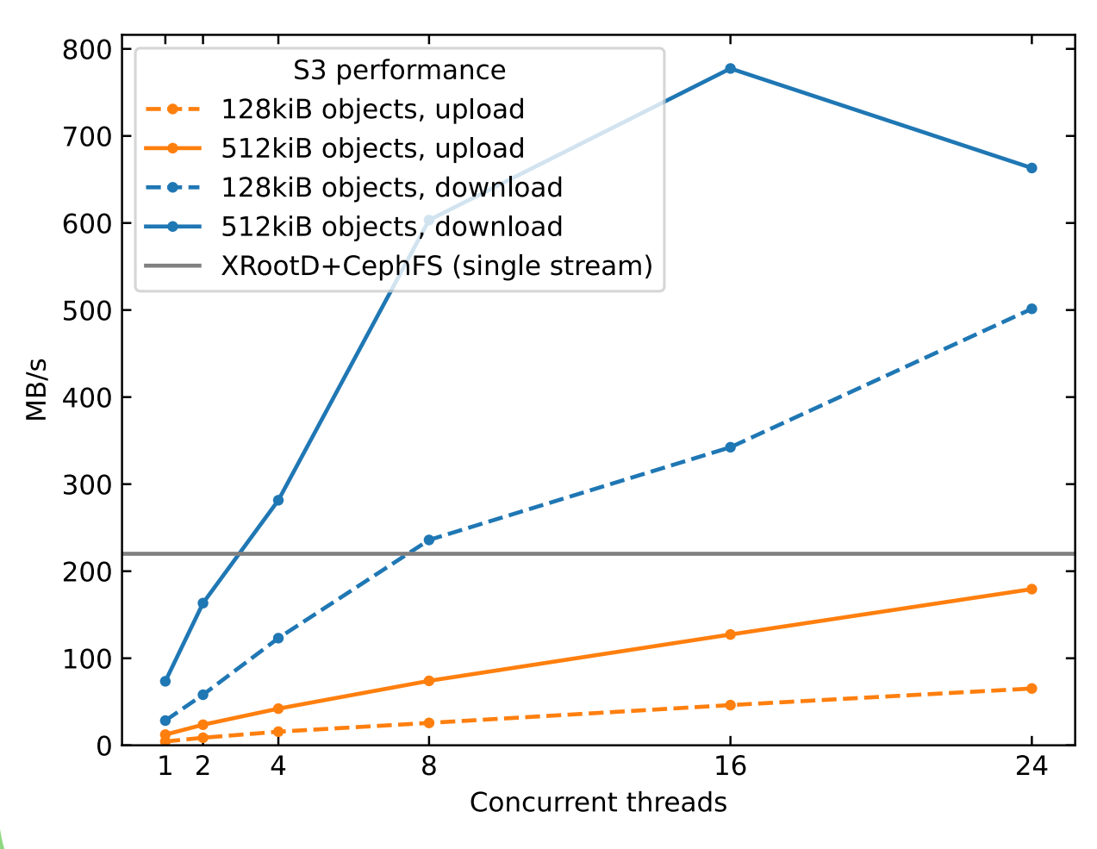


Label convention:
E* = Event number
P* = Product (column) index
i* = Global index
S* = Stripe starting at global index

Test cluster

A pilot cluster has been assembled to gain experience in managing a Ceph installation. Nine servers provide 2 PB of HDD, and two servers provide 20 TB of NVMe for metadata. All machines have 10 to 100Gbps networking. The cluster supports filesystem and S3 object access mechanisms. An xrootd server exposes the filesystem to hosts within the Fermilab network.





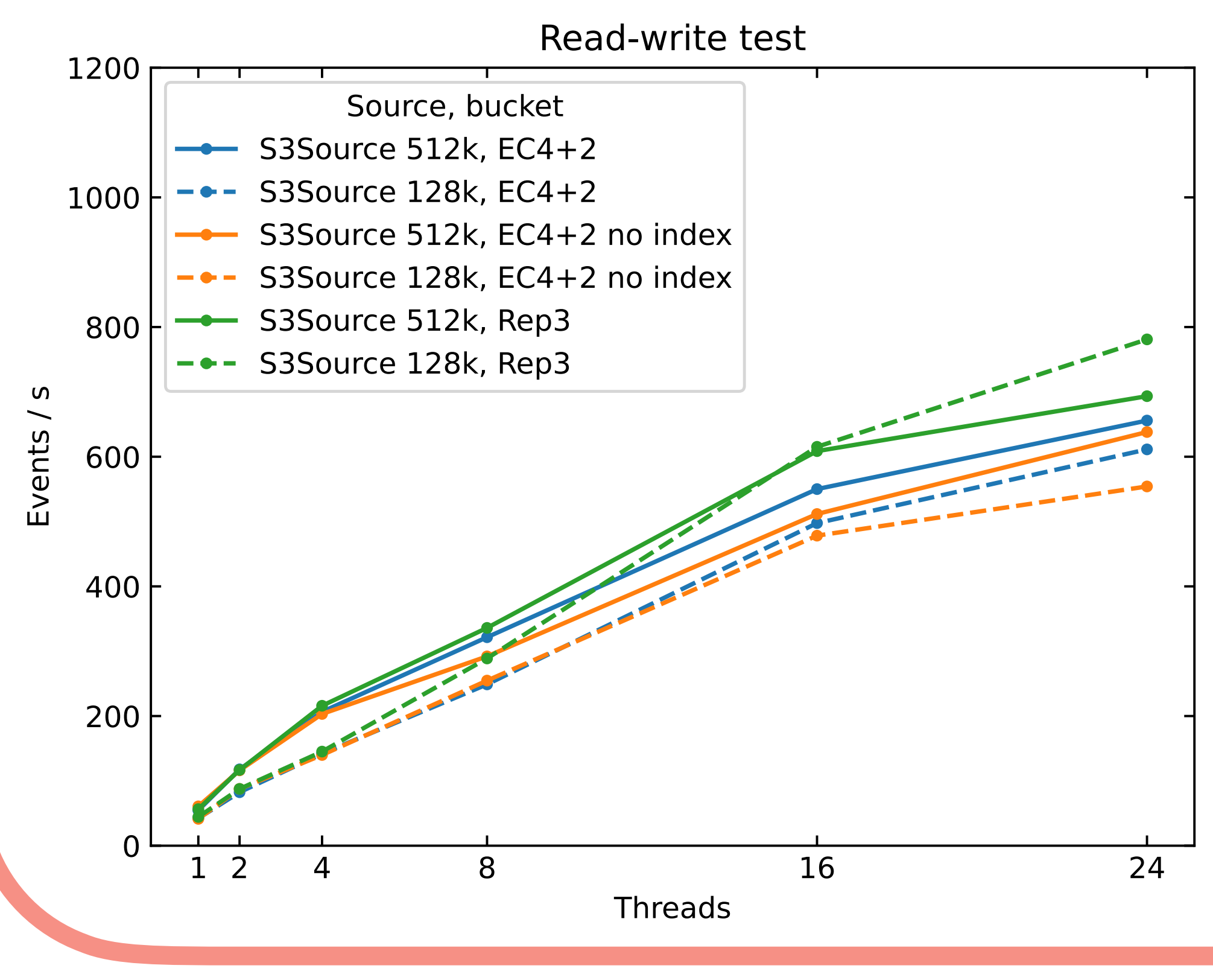
Raw read and write performance of the cluster is within expectation given the current hardware configuration. Here we are testing the performance of a pure-I/O workload to an erasure-coded pool.

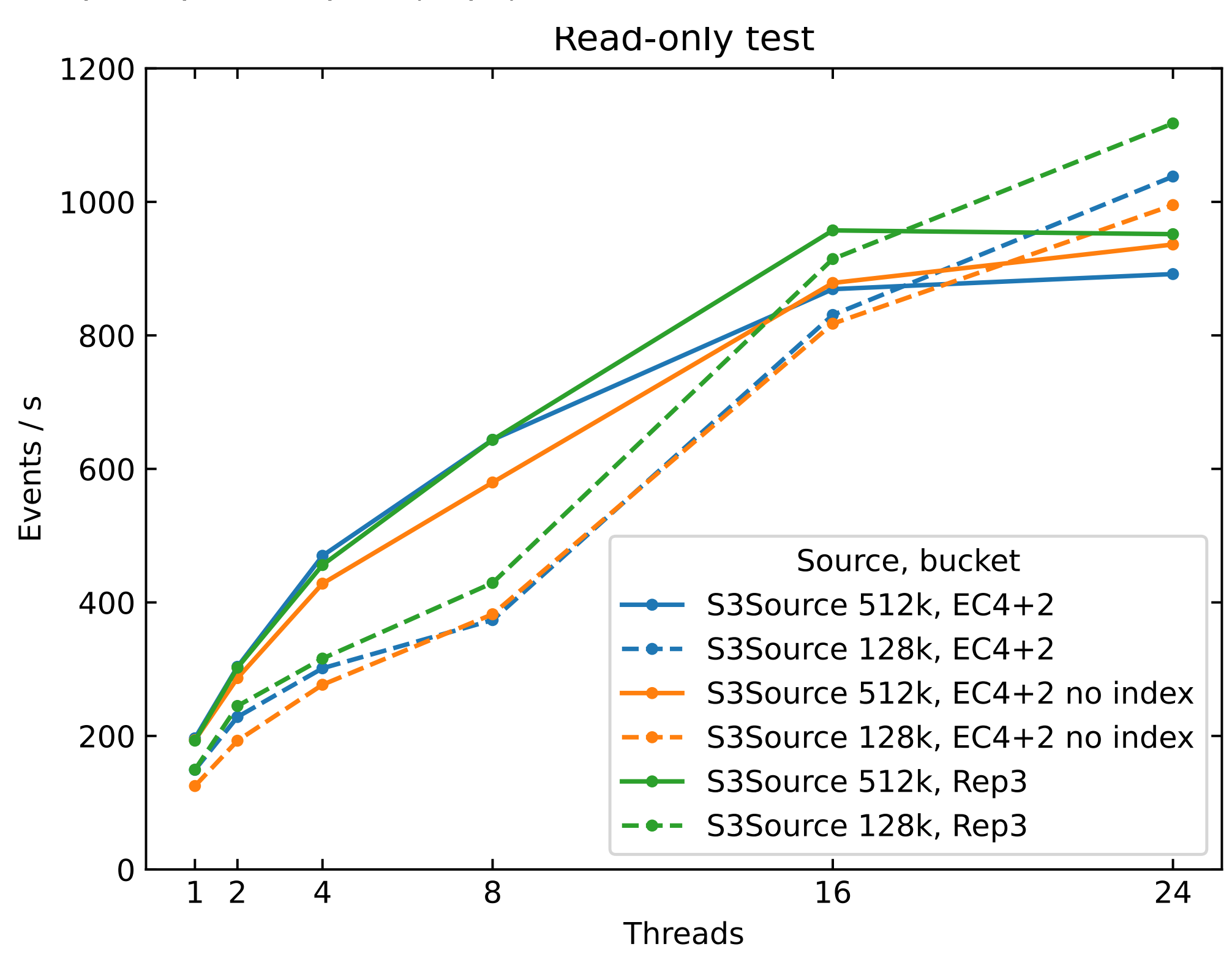
Performance tests

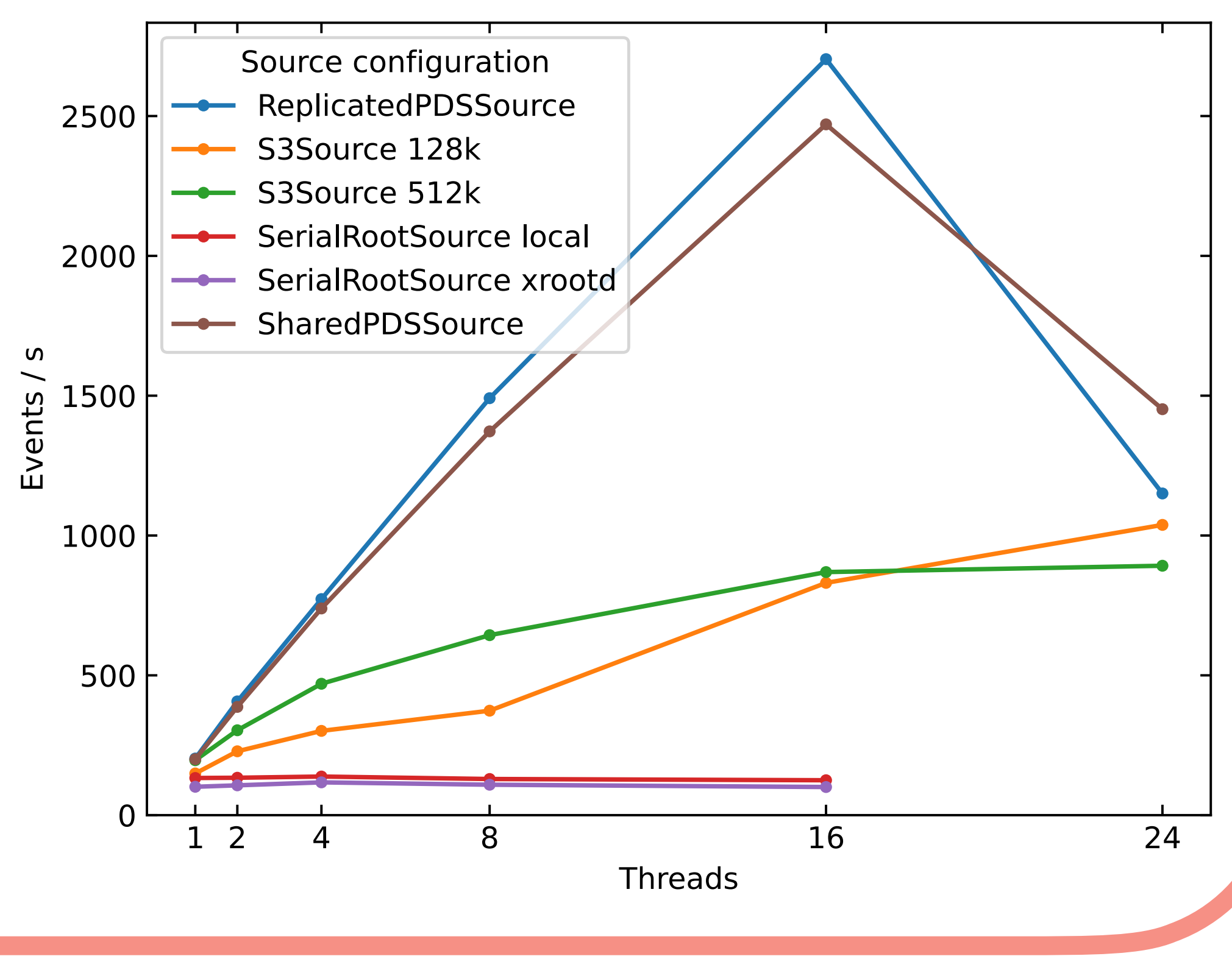
A set of tests where all data products in the MiniAOD tier are read and written as fast as possible was performed. The executable's thread scaling properties are probed, with the metric being the events processed per second. In the full chain test, events are: read, decompressed, deserialized, serialized, compressed, and written. For the read-only chain, only the first three steps are performed. Tests were run on a 24-core machine with 10Gbps network connection to the Ceph cluster. These tests show good scaling behavior with increased thread count.

The tests are performed for two configurations of event batch size and target stripe size. For each, the source and output modules target one of three S3 buckets with different Ceph pool configurations: an erasure coded configuration with 4 data blocks (4kiB) and 2 parity blocks (EC4+2), the same configuration with the bucket index disabled (about 300b metadata per object stored in 3x replicated NVMe pool), and a triple-replicated pool (Rep3).

Comparison to an input source similar to that used in a CMSSW grid job, where the input is read (either on-site or remotely) via xrootd. The xrootd server has CephFS mounted and is accessing a pool with the same erasure coding setup as the S3 source. In addition, a PDS source (file format with concatenated serialized events) is compared. For this test, data is read and discarded.


Read-write test
Source, bucket
S3Source 512k, EC4+2
S3Source 128k, EC4+2
S3Source 512k, EC4+2 no index
S3Source 128k, EC4+2 no index
S3Source 512k, Rep3
S3Source 128k, Rep3
Events / s
Threads


Read-only test
Source, bucket
S3Source 512k, EC4+2
S3Source 128k, EC4+2
S3Source 512k, EC4+2 no index
S3Source 128k, EC4+2 no index
S3Source 512k, Rep3
S3Source 128k, Rep3
Events / s
Threads


Source configuration
ReplicatedPDSSource
S3Source 128k
S3Source 512k
SerialRootSource local
SerialRootSource xrootd
SharedPDSSource
Events / s
Threads