Object Stores for CMS data

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

CCE-IOS meeting

13 Dec 2022

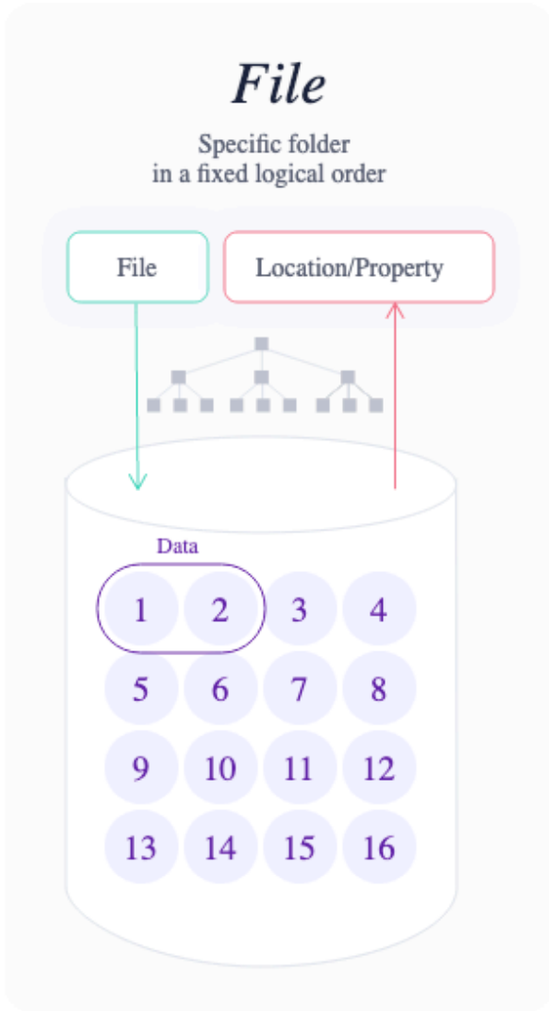
Rothko, Number 19 (1949)

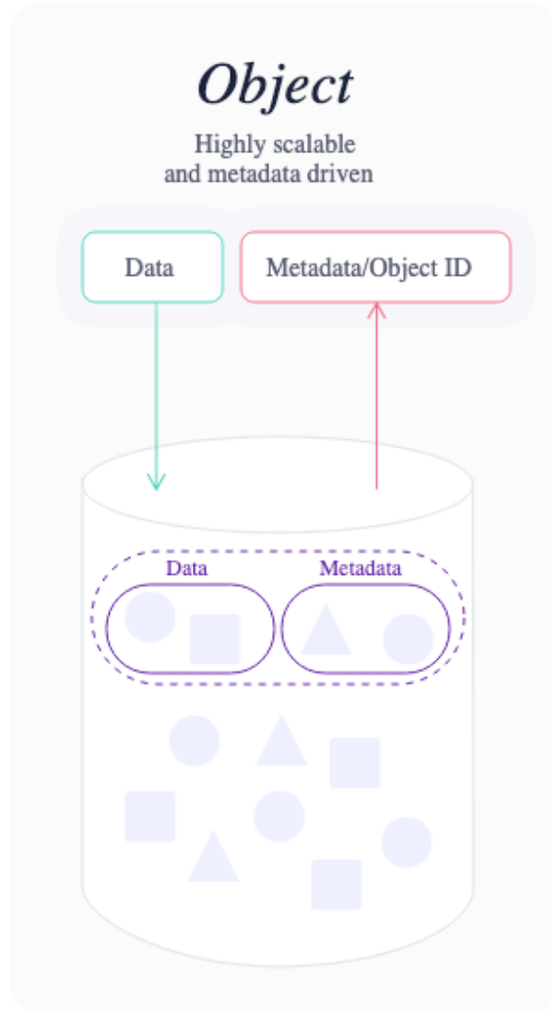
Overview

- What is an object store?
- How can we store our data as objects?
- What benefits does this have vs. files?
- Can we read and write data efficiently with objects?
  - Test results with prototype framework

Object store vs. filesystem

- Traditional data storage technology: distributed filesystem
  - e.g. NFS, EOS, dCache, Lustre, HDFS*, …
  - Often with remote access protocol (xrootd)
  - Files are concurrently read/writeable
- Popular new-ish technology: object store
  - Native remote access (http)
  - Objects are immutable (overwrite possible)

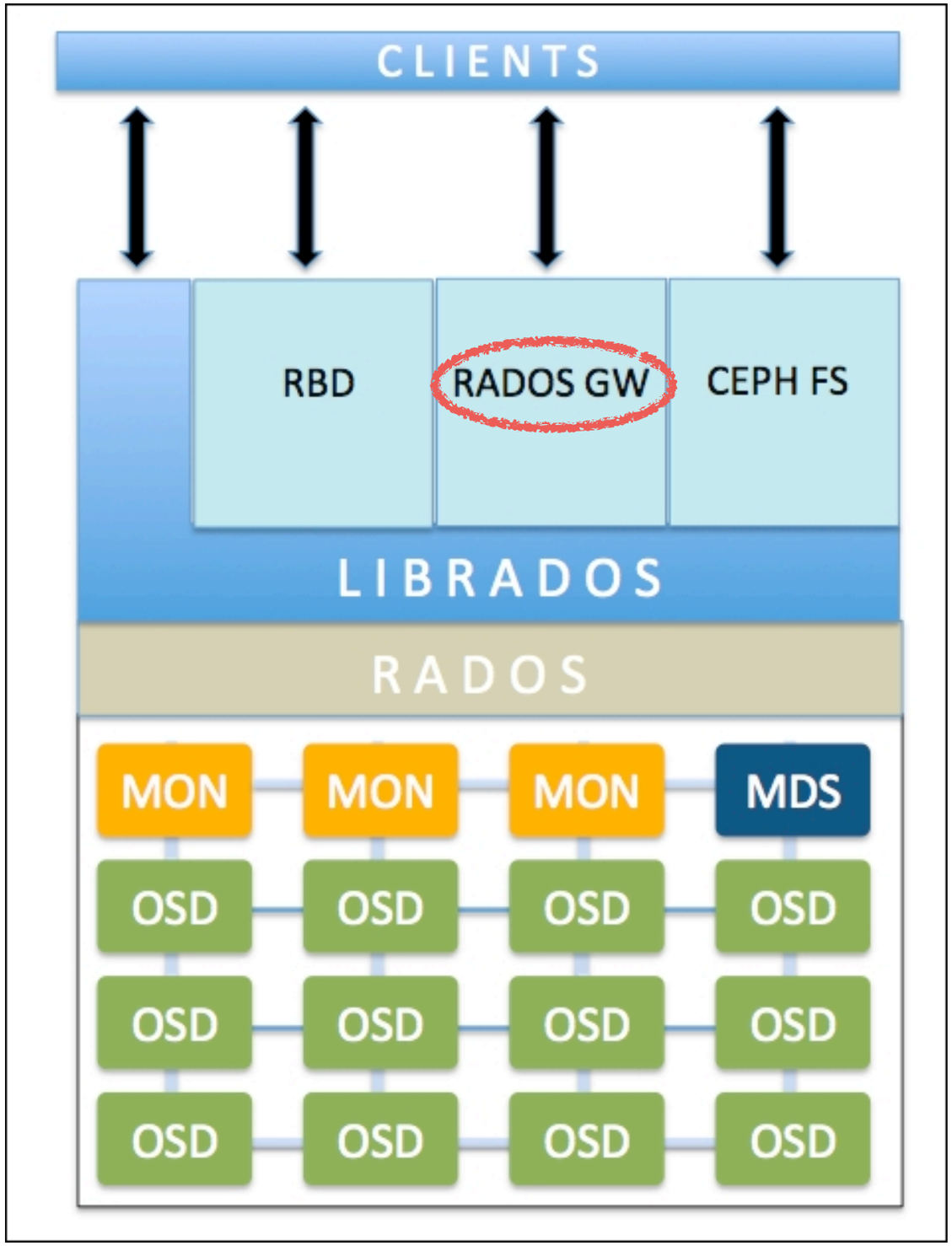




attrib

Ceph technology

- Ceph cluster
  - Distributed storage system
  - Scalable, fault-tolerant, software-defined
- RadosGW
  - Ceph implementation of S3 *protocol*
  - Unrelated to, but carries features (baggage?) of Amazon S3 service



attrib

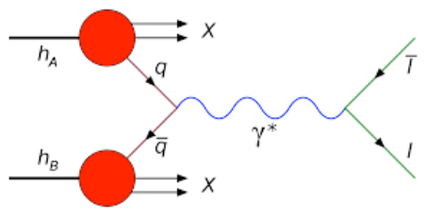
Fermilab

CMS Data organization

- CMS stores events (rows) in *datasets* (~100k)
  - Each dataset has many *files* (~100M), each has several *replicas* stored at WLCG sites
  - *Data tiers* statically define what information (columns) is included
  - Analysis uses the smallest practical data tier

Primary dataset

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

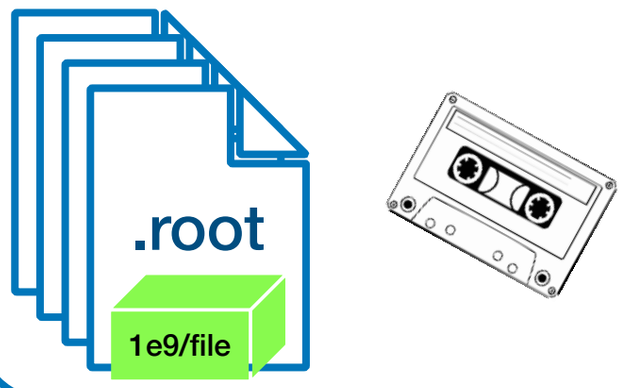


Data tiers

AOD

1e5/event

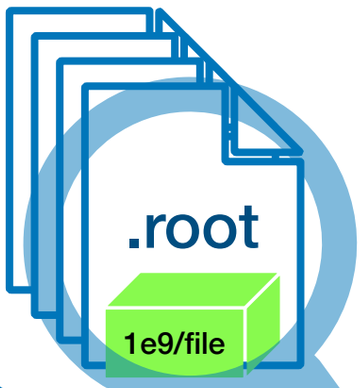
Data columns pertaining to low-level reconstruction



MiniAOD

1e4/event

Calibrated physics objects
Particle-flow candidates



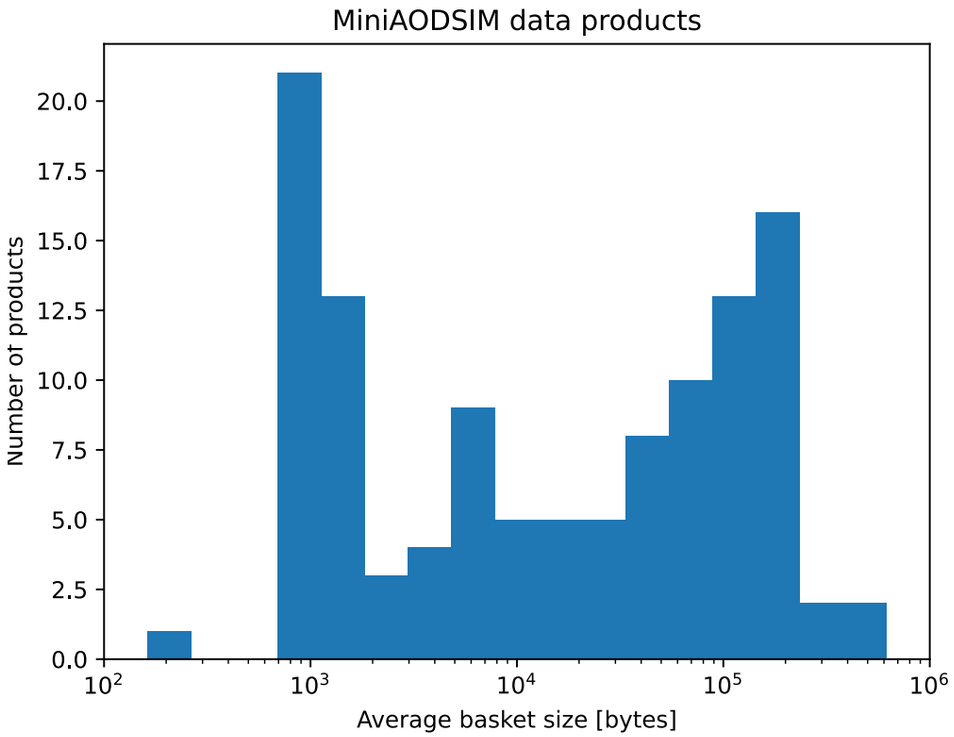
. . .

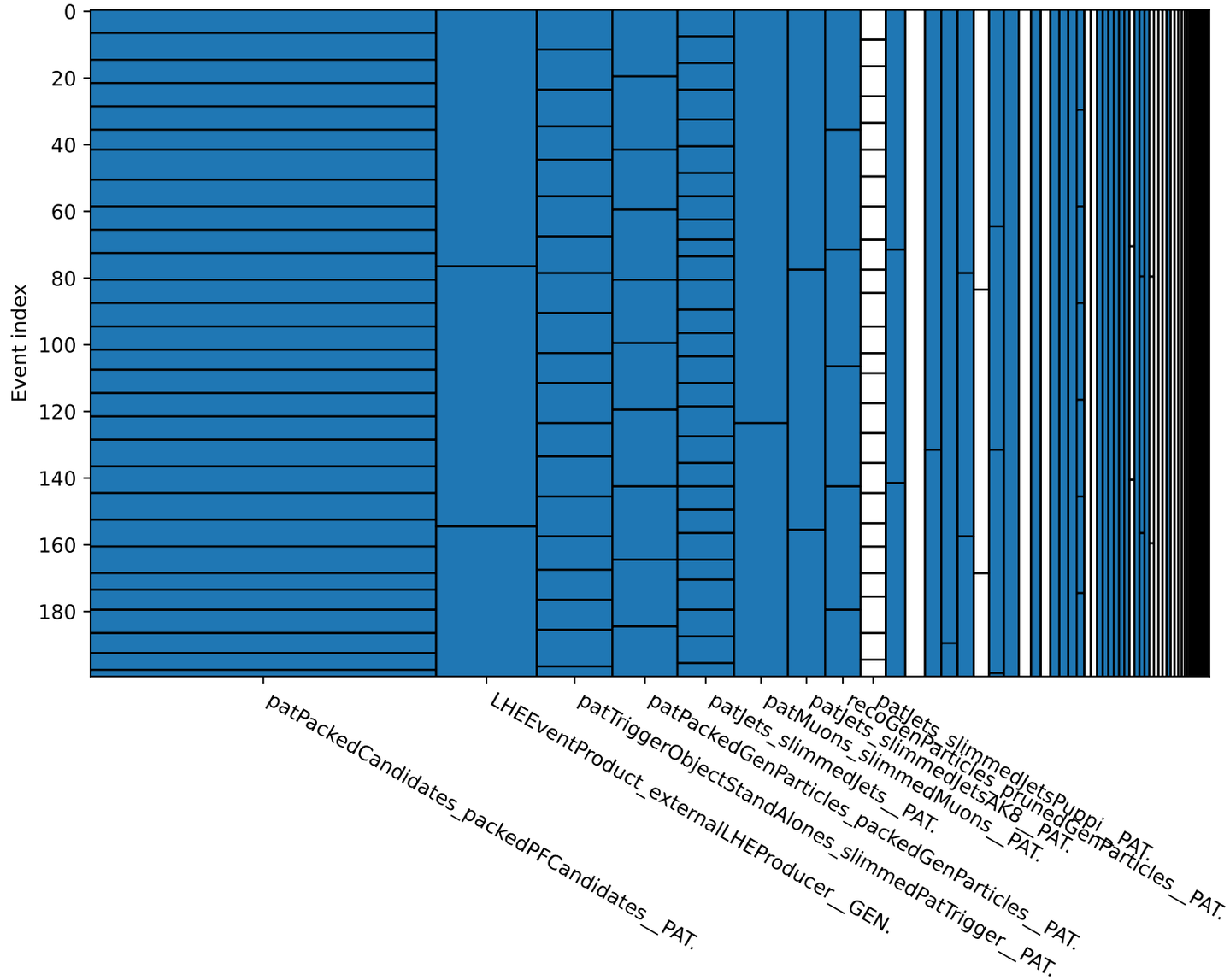
Data volume order of magnitude [bytes]

Fermilab

File format

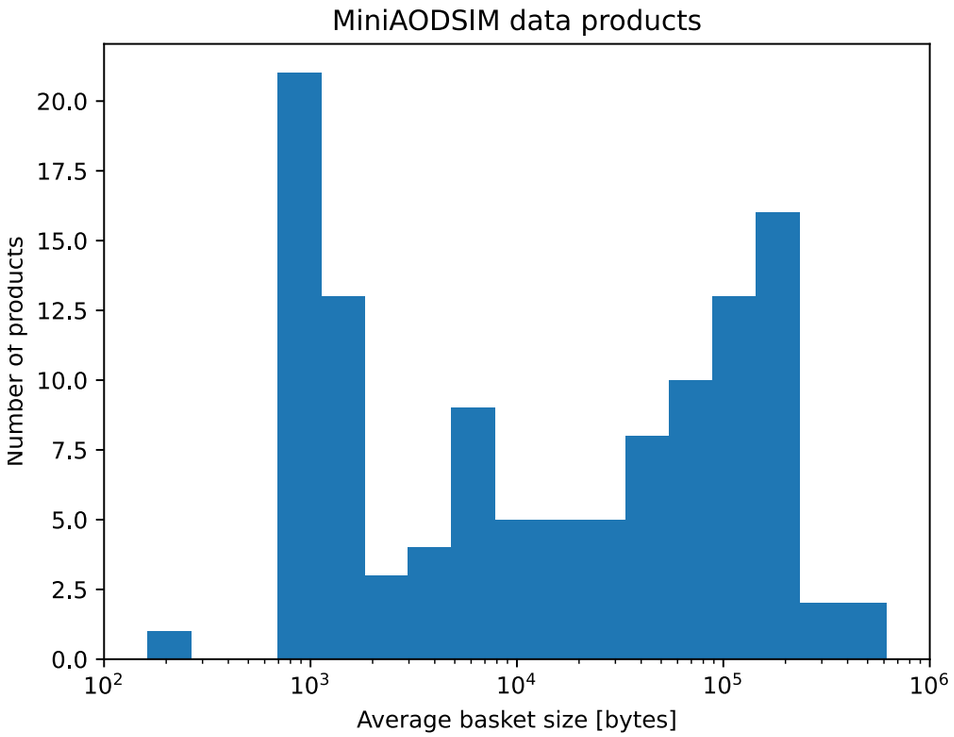
- Event Data Model (TTree)
- Branch: metadata about C++ data type, basket positions
- Basket: serialized C++ objects stored contiguously*

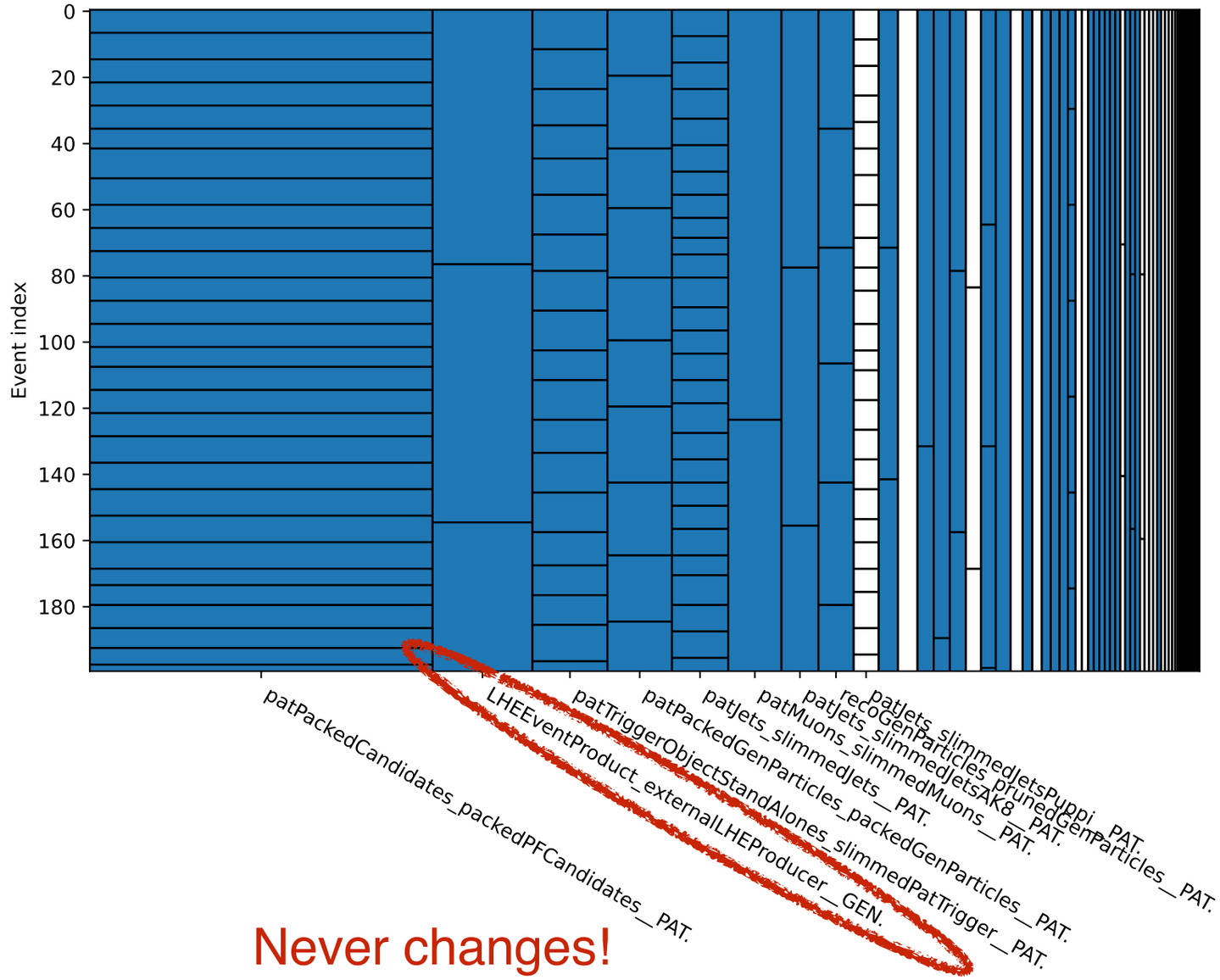




File format

- Event Data Model (TTree)
- Branch: metadata about C++ data type, basket positions
- Basket: serialized C++ objects stored contiguously*



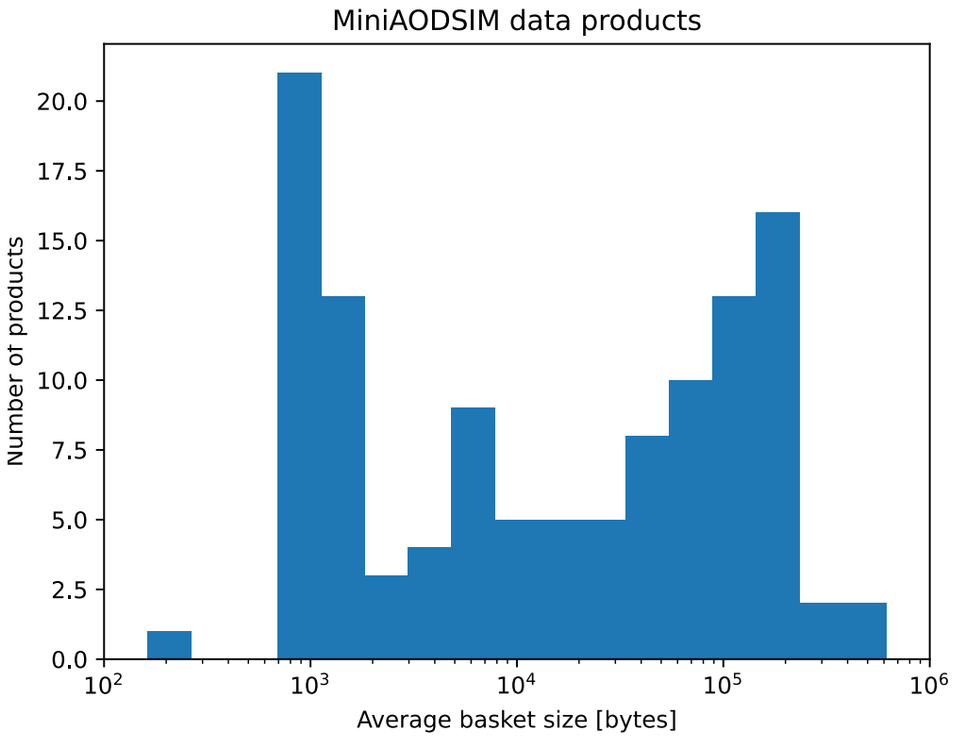


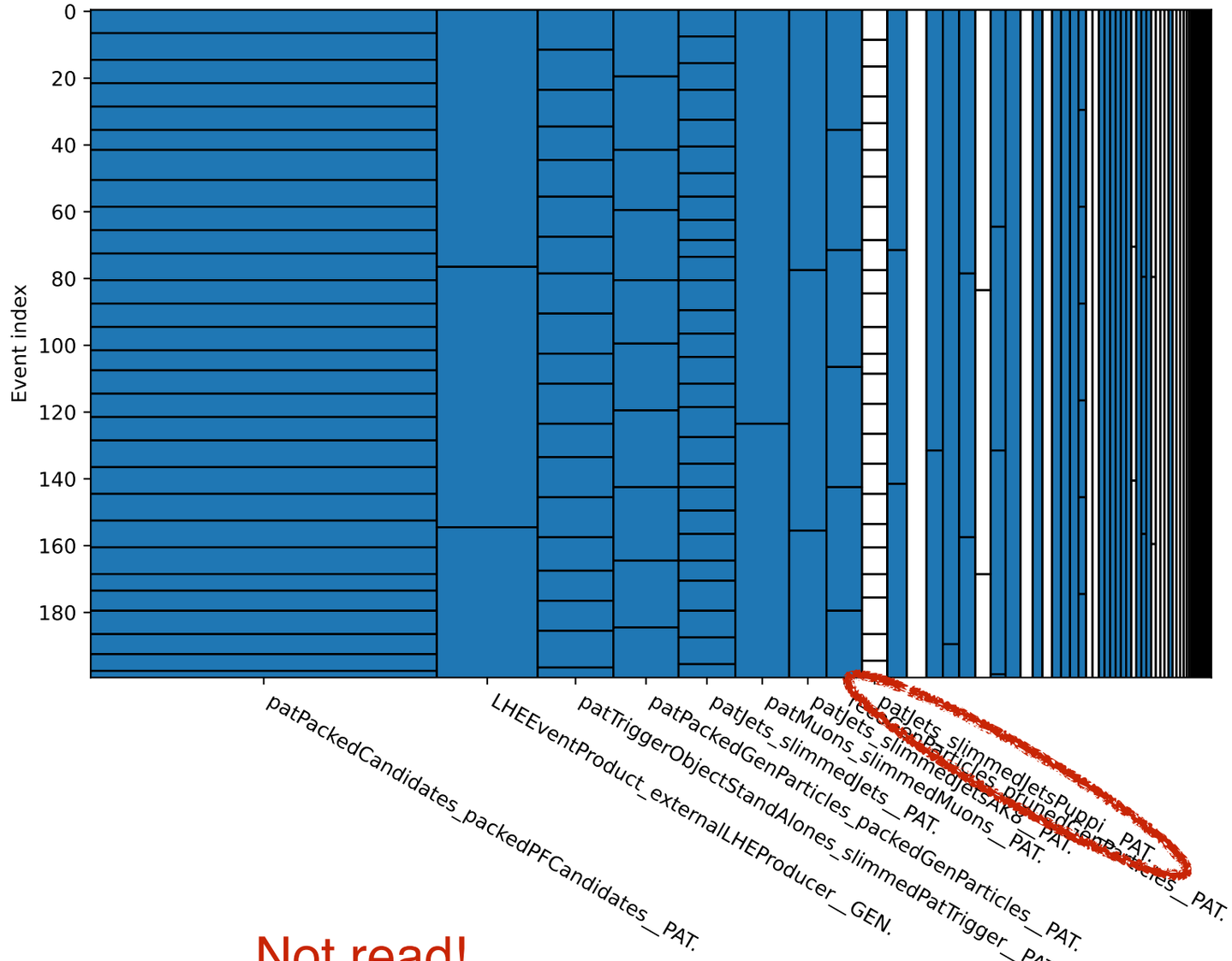
Never changes!

Fermilab

File format

- Event Data Model (TTree)
- Branch: metadata about C++ data type, basket positions
- Basket: serialized C++ objects stored contiguously*





Not read!

Fermilab

Breaking down the ROOT file

- Essentially storing (+ moving) smaller units
  - This is usually a bad thing

[Intermodal container](#)

?

[Break-bulk cargo](#)

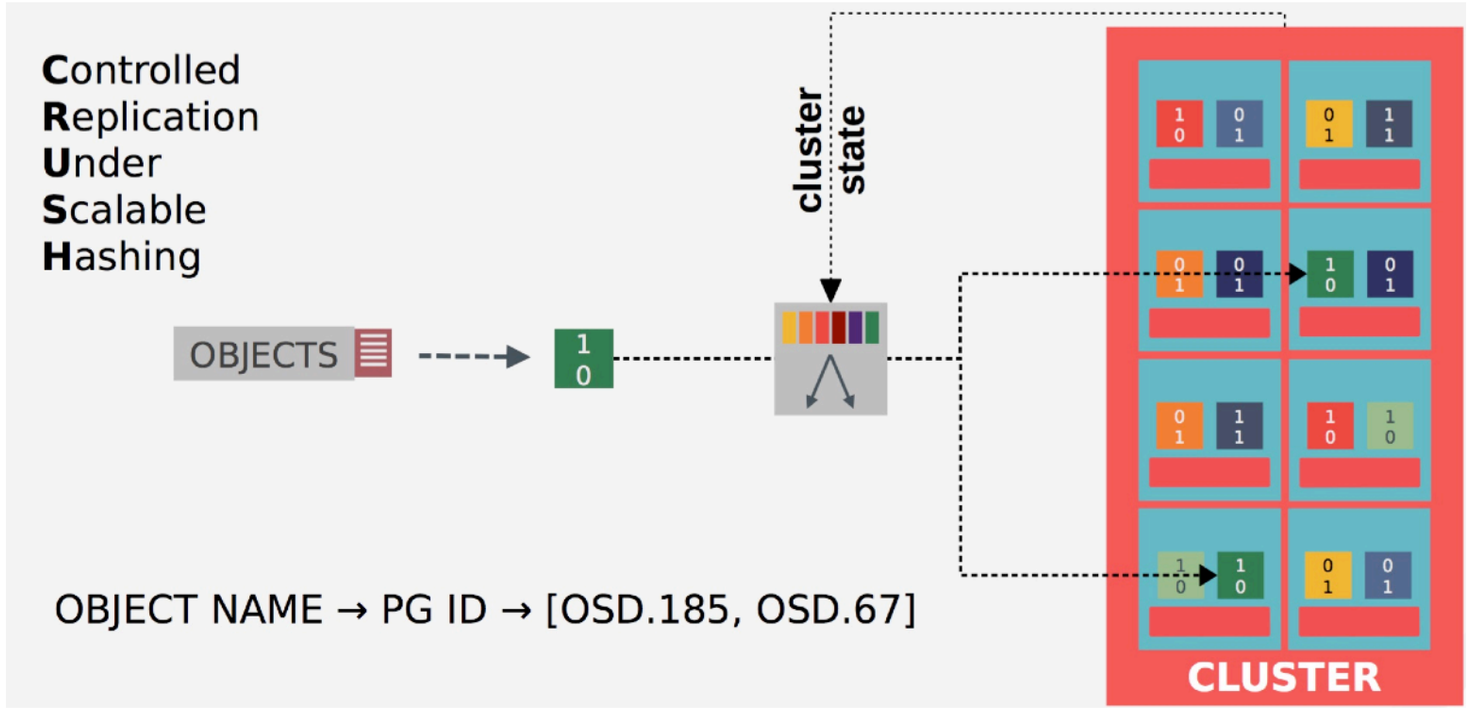
Fermilab

Breaking down the ROOT file

- Essentially storing (+ moving) smaller units
  - This is usually a bad thing
- Calculated placement
  - Like a hash, client-side
  - Downside: cluster state change causes reshuffle
    - Consistent hashing to minimize movement

Intermodal container

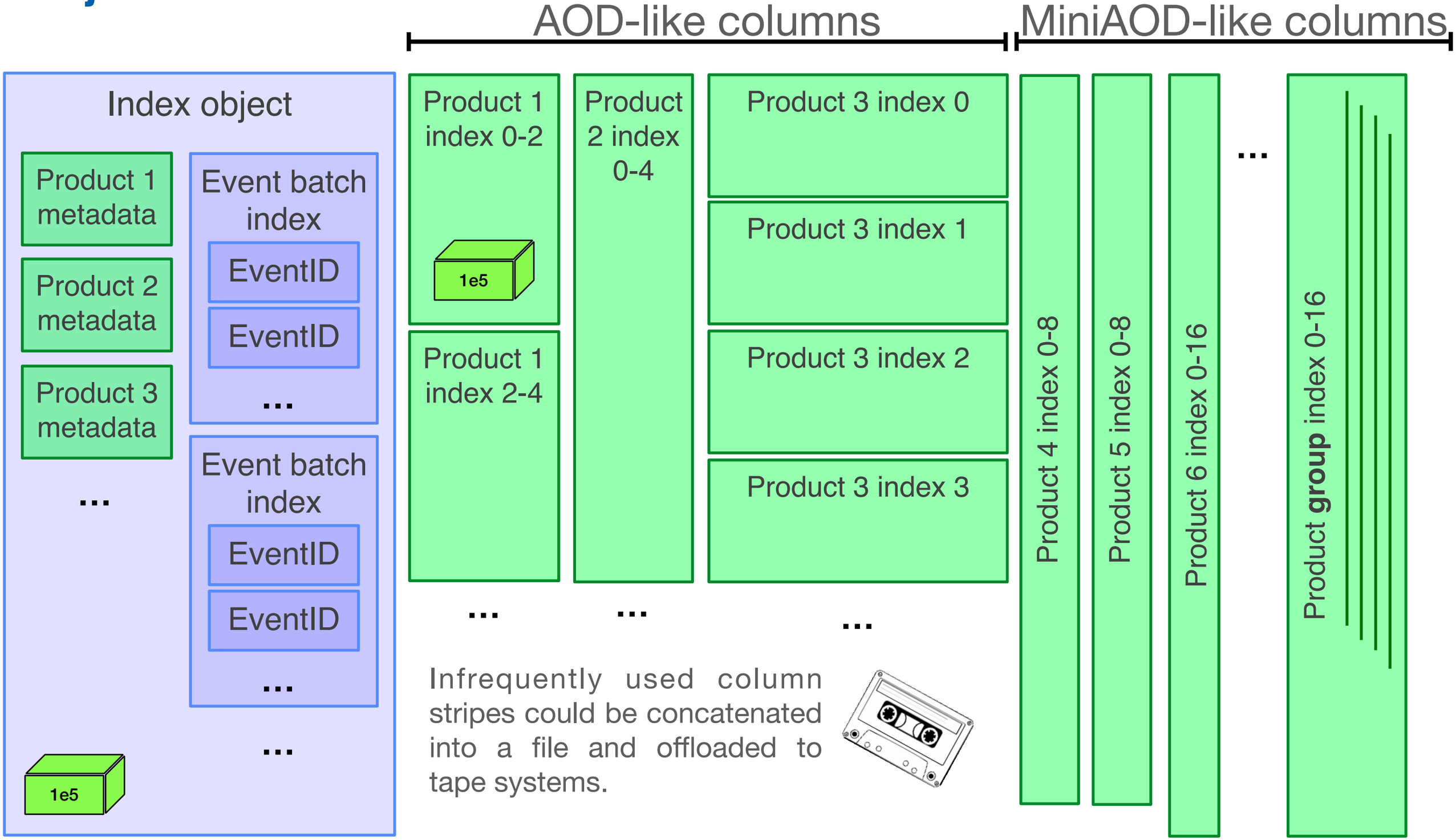
?



Break-bulk cargo

Fermilab

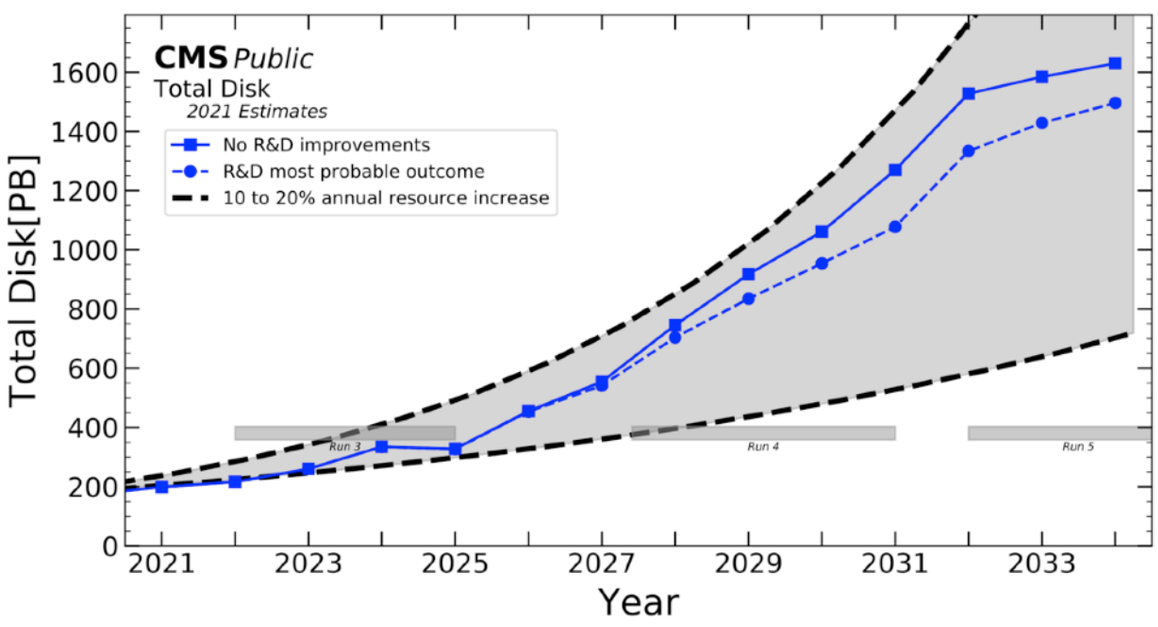
Object data format

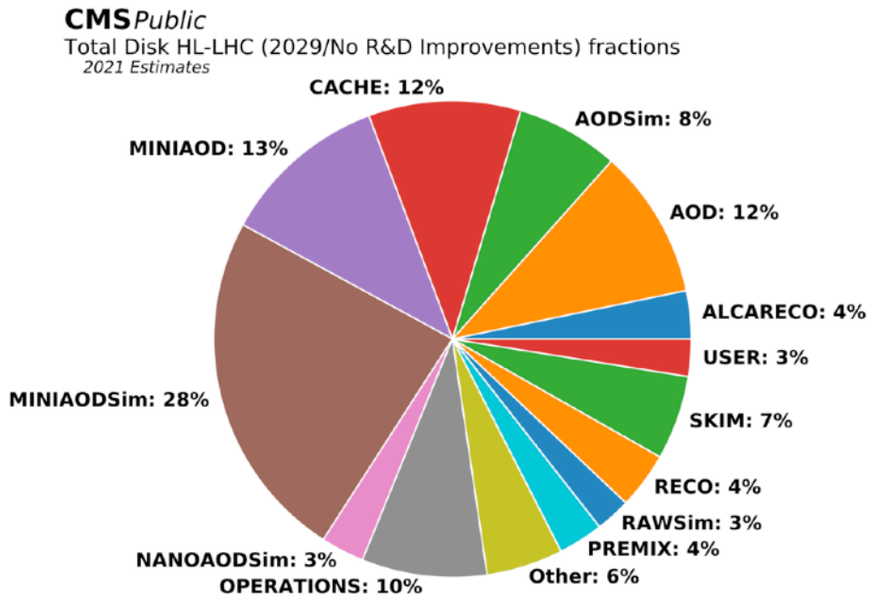


Fermilab

Disk as a cache

- Disk is expensive (vs. tape)
  - Only MiniAOD, NanoAOD data tiers reliably on disk now
    - Ok because of 10+y experience with detector to know what we need
    - For HL-LHC, new detectors may require more time with low-level information
- Best cache: all the columns you need, none you don't
  - Different set of columns needed for different PDs, analyses
  - Not all rows read if filtering (skimming)
  - **How much can we reduce disk use from PD*tier granularity we have now?**





https://twiki.cern.ch/twiki/bin/view/CMSPublic/CMSOfflineComputingResults

Strawman

- Most content does not change with re-processing
  - Even for UltraLegacy, already two MiniAOD versions
- Keeping only new products would save a lot of disk

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

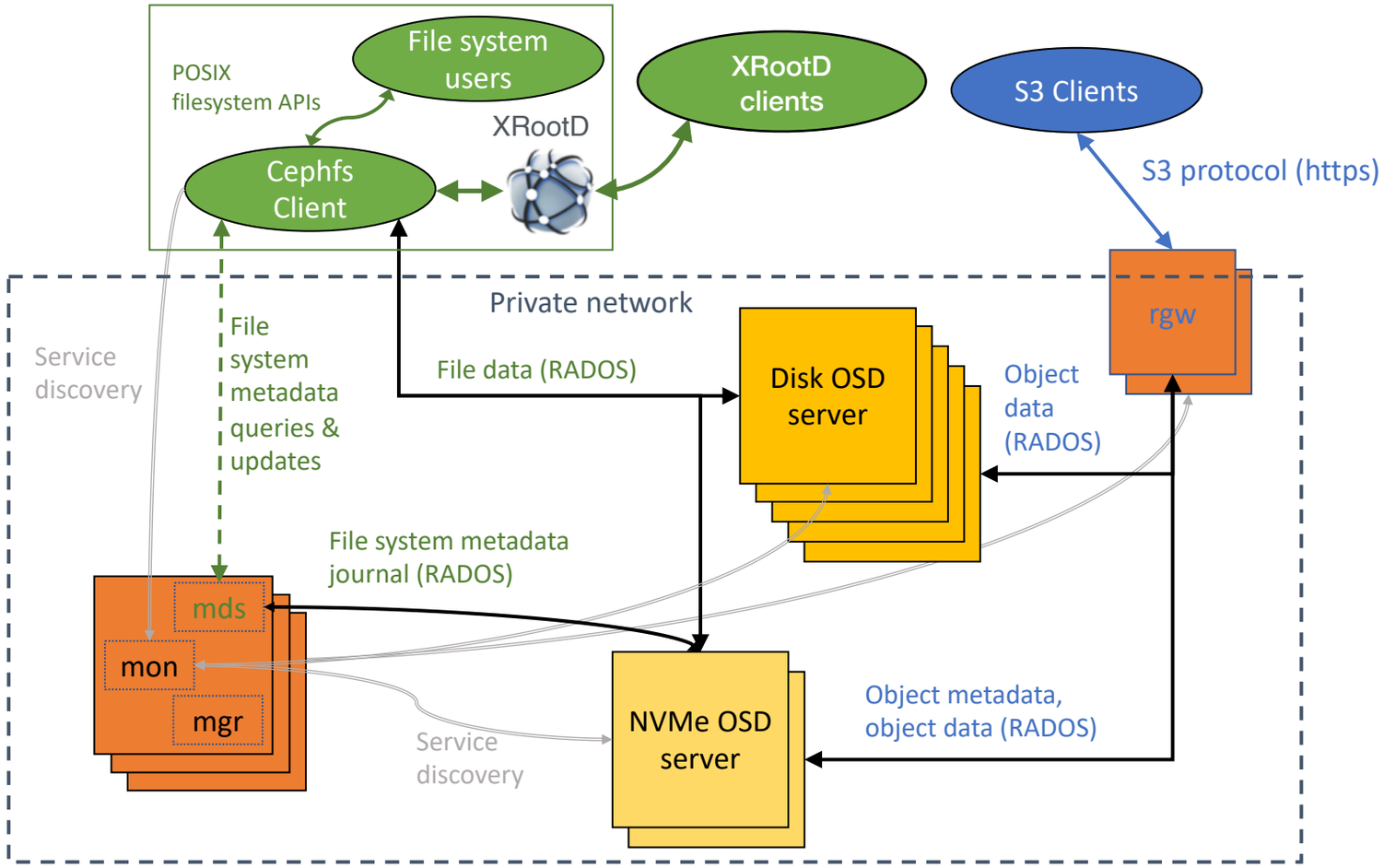
Object store scheme

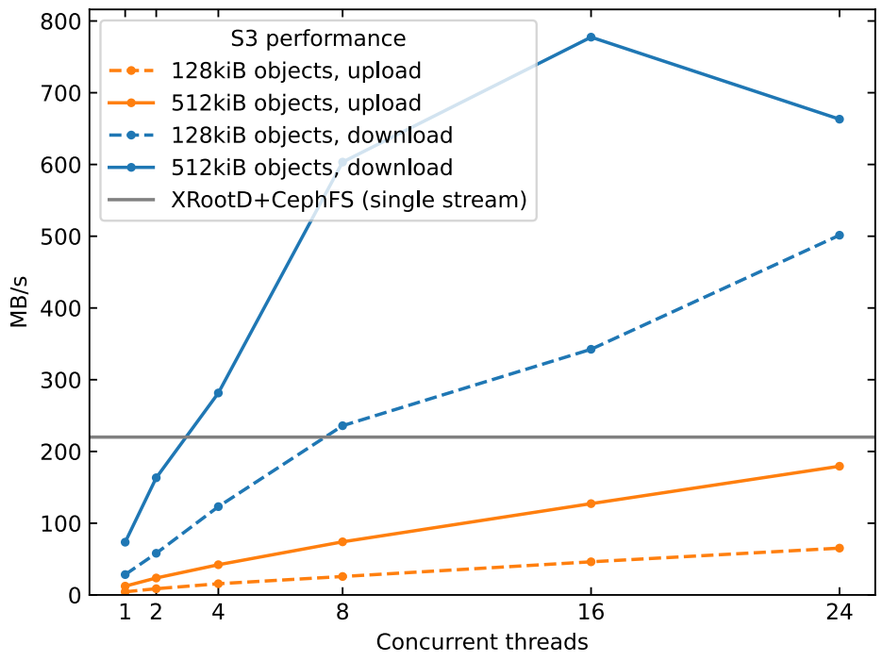
MiniAOD Data product	KB per event	
	v1	v2
packed+pruned genParticles	5.7	-
slimmedElectrons	1.3	-
Others	48.7	-
Updated slimmedElectrons	-	1.3
Total	55.7	1.3

Numbers sourced from a UL17 TTBar simulation file

Test cluster

- Ceph pilot cluster setup at FNAL
  - 9 retired dCache machines
    - Total 2 PB HDD, circa 2014-2018
  - Two servers for metadata
    - 20TB NVMe
  - Edge machines for xrootd door to CephFS, Ceph management daemons (mon,mgr)





Fermilab

Object I/O performance

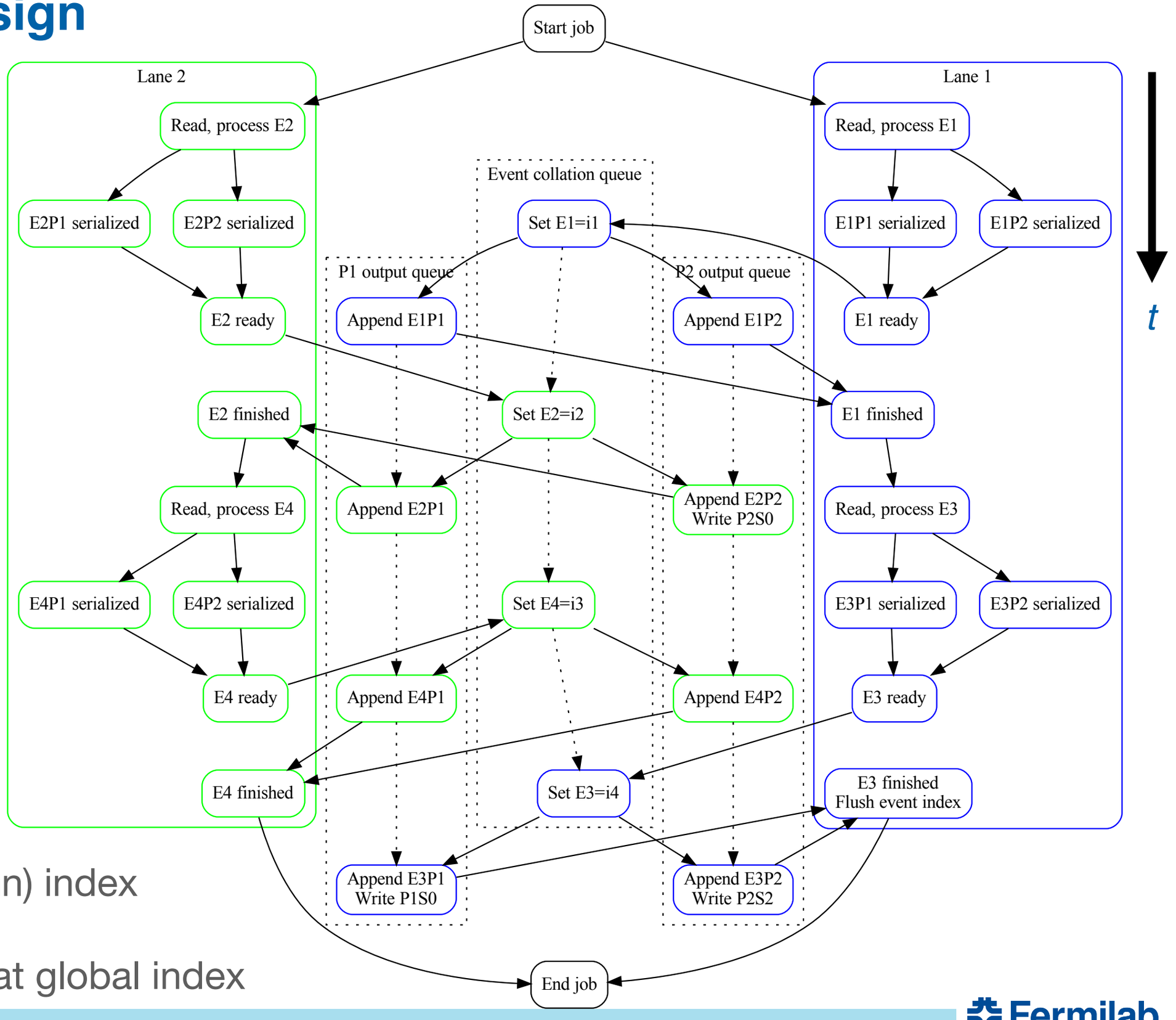
- CCE-IOS project to evaluate CMSSW (& other) I/O performance
  - Output module is a bottleneck for CMSSW above 8 threads
  - With (in development) ROOT concurrent I/O, issue may go away
  - However, still worth investigating performance of writing to many objects vs. one file
- Framework to evaluate alternative I/O strategies
  - https://github.com/hep-cce2/root_serialization
  - Easy to add new output modules, simulate event processing, and test I/O
- Wrote an S3Source and S3Outputer in the framework
  - Using libs3 as backend, fully asynchronous with TBB tasks
  - https://github.com/hep-cce2/root_serialization/pull/47
  - Key features:
    - Parallel stream compression
    - Asynchronous I/O
    - Row-wise to column-wise pivot

S3Outputer design

Each box is a TBB task
Color = task group

Product 1 has stripes written every 4 events

Product 2 has stripes written every 2 events



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

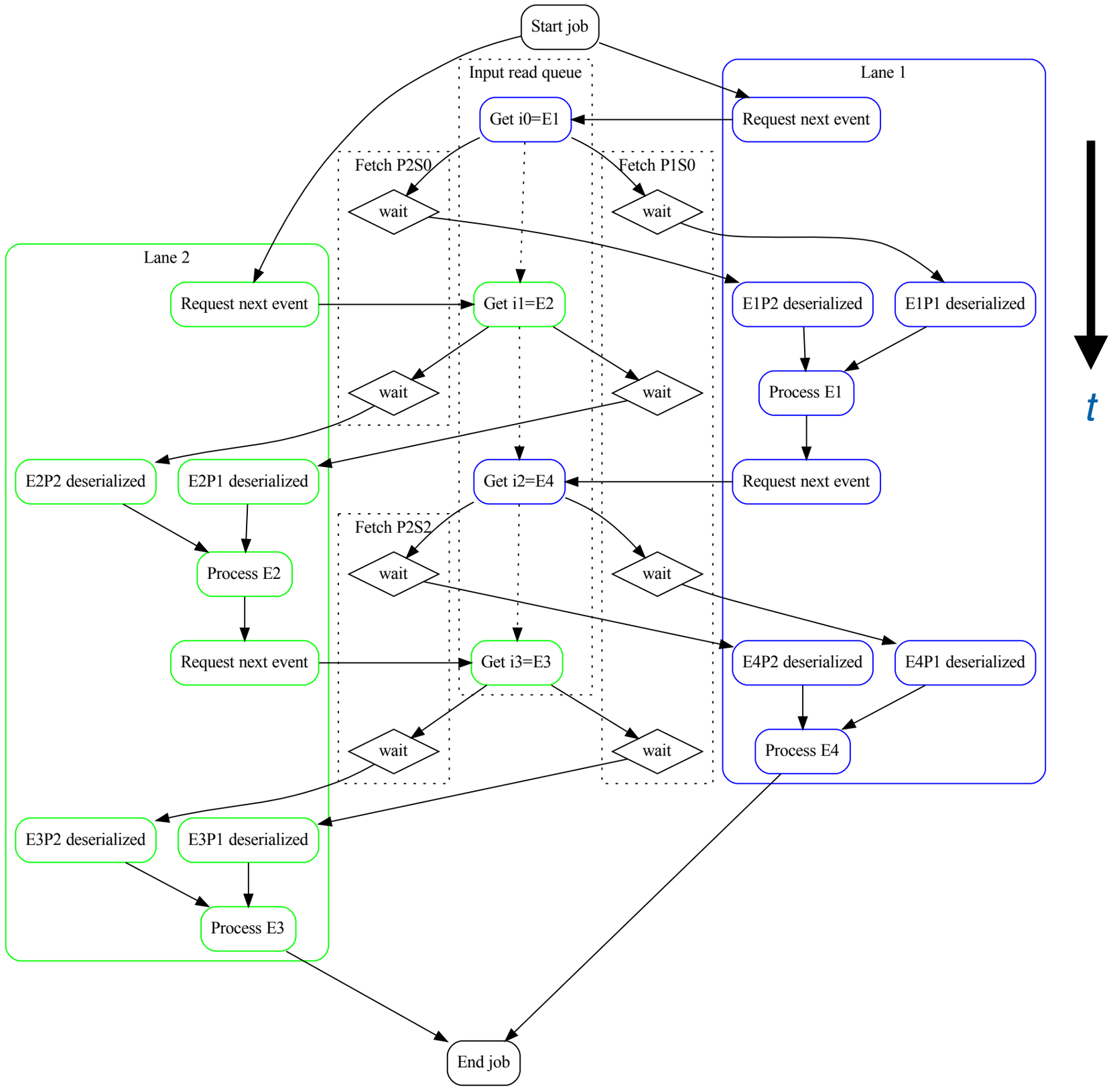
Fermilab

S3Source design

Each box is a TBB task
Color = task group

Product 1 has stripes
read every 4 events

Product 2 has stripes
read every 2 events

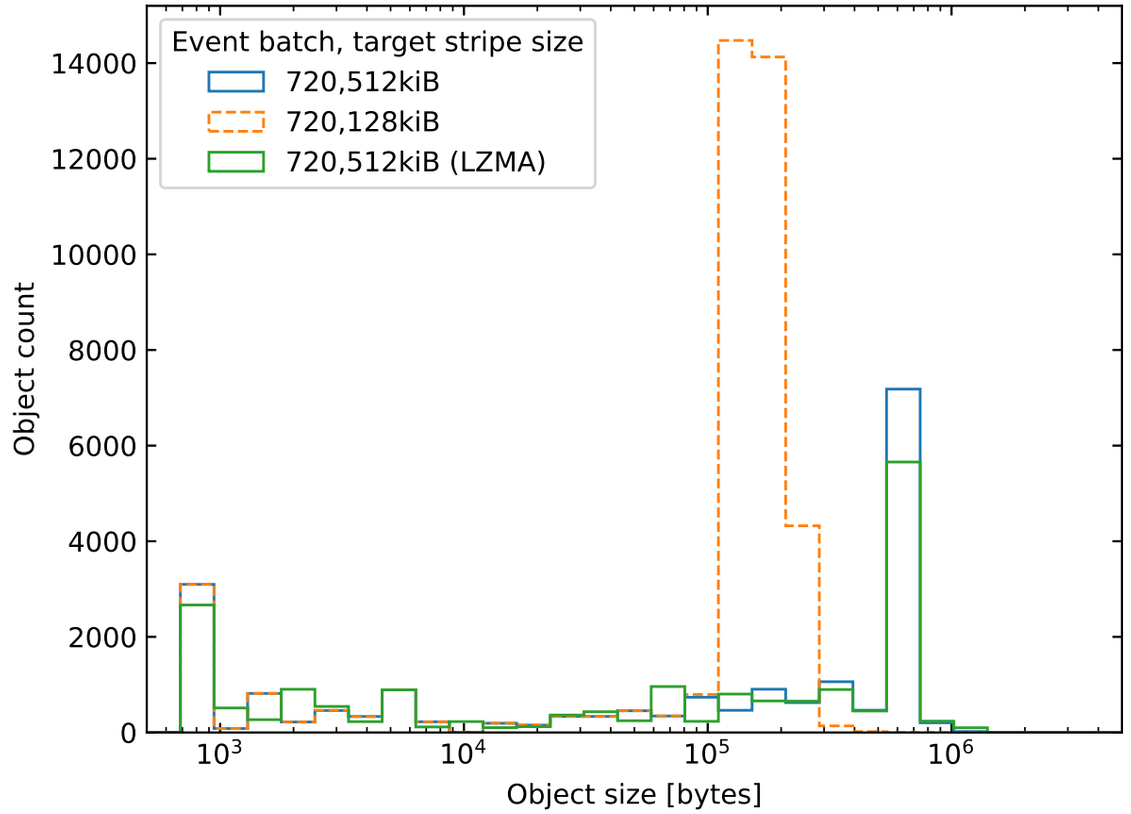


Label convention:
E* = Event number
P* = Product (column) index
i* = Global index
S* = Stripe starting at global index

Storage efficiency

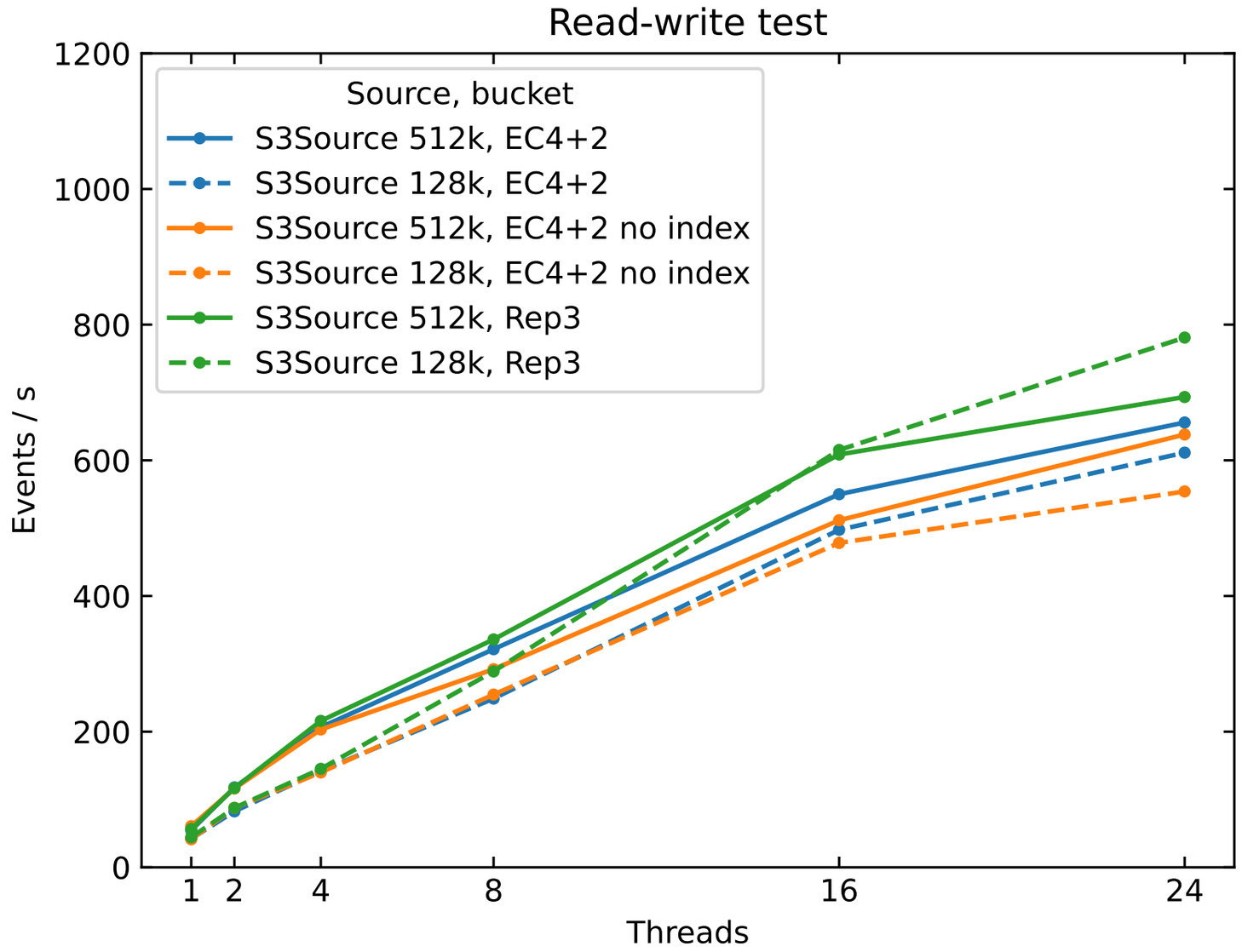
- Input: 80k event MiniAOD file
  - LZMA compression
- Three S3Outputer configurations tested:
  - Event flush count 576, product flush size 128kiB, ZSTD4 compression
  - Event flush count 720, product flush size 512kiB, ZSTD4 compression
  - Event flush count 720, product flush size 512kiB, LZMA9 compression
- For k+m erasure-coded storage pool, minimum object granularity of k*4kiB
  - Overhead for EC4+2 in % listed below

Format	KB per event
MiniAOD input	**55.7**
Objects: **- event batch size 720** **- target stripe size 128kiB**	**71.4 + 6.5%**
Objects: **- event batch size 720** **- target stripe size 512kiB**	**70.6 + 3.5%**
Objects (LZMA): **- event batch size 720** **- target stripe size 512kiB**	**61.8 + 3.7%**



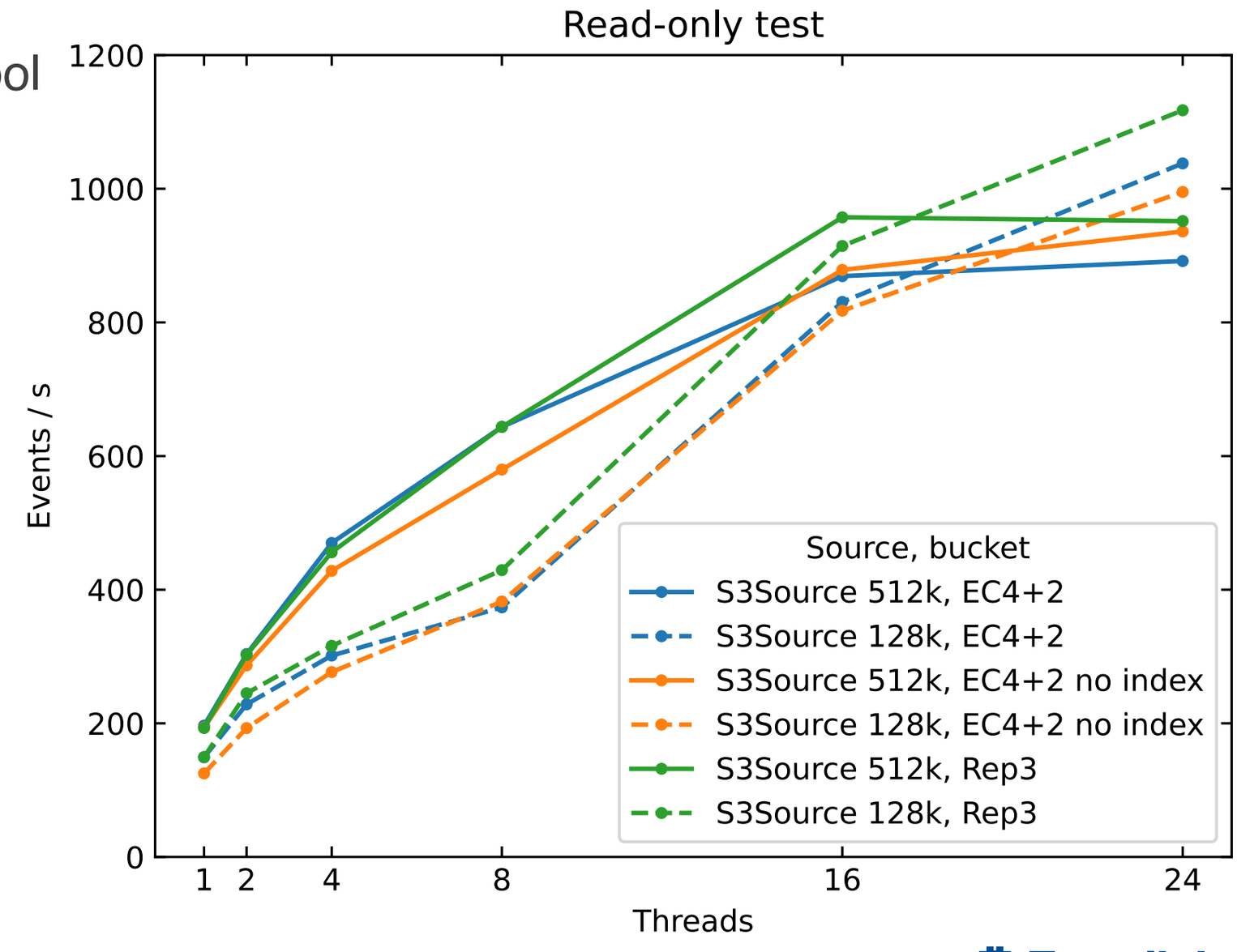
S3Source -> S3Outputer

- Full chain: fetch→decompress→deserialize→serialize→compress→push
- Repeating same test, with different bucket configurations
  - Baseline: EC4+2, bucket index in rep3 ssd pool
  - “no index” disable bucket index
  - “Rep3” data in 3x replica hdd pool
- Vary also S3Outputer config
  - No LZMA (very slow)



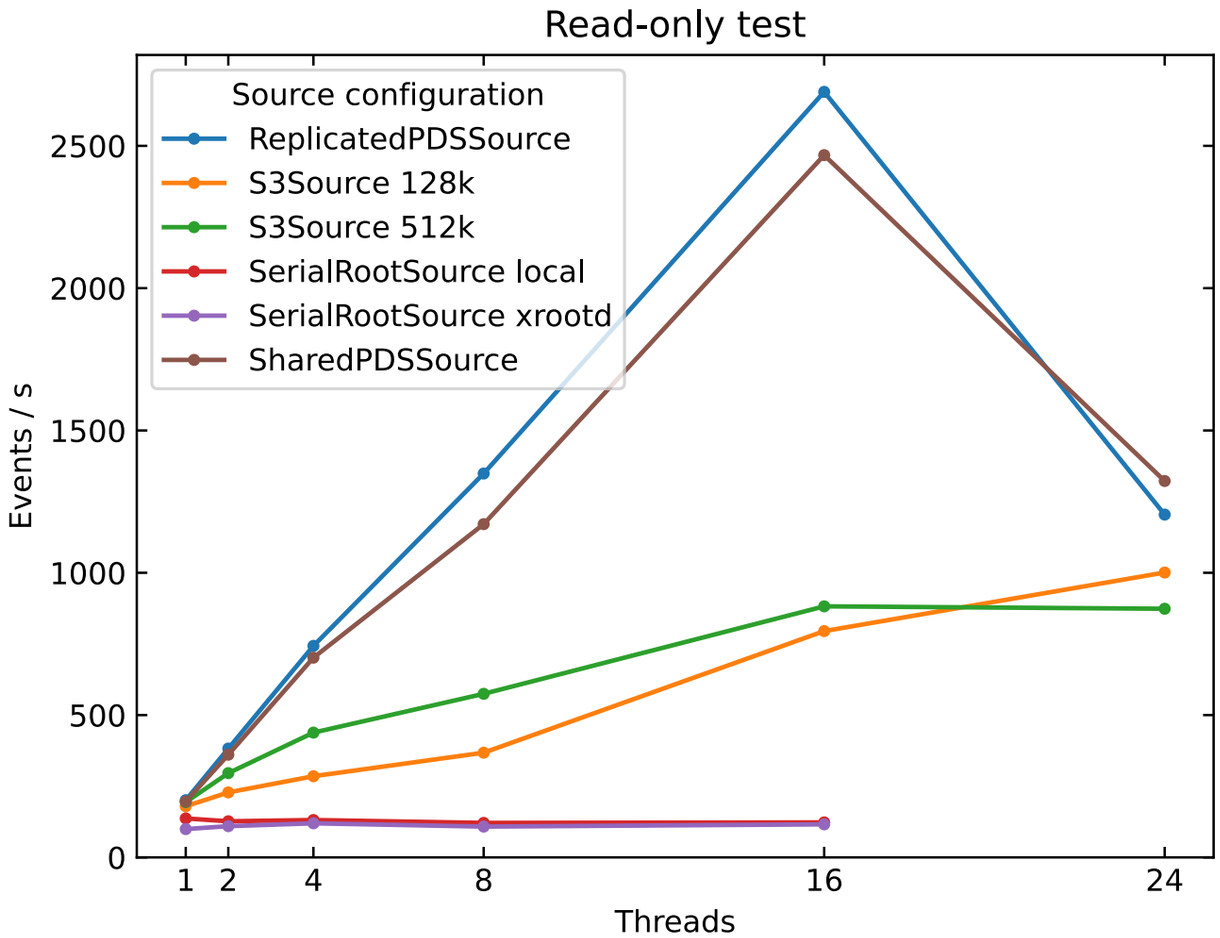
S3Source -> DummyOutputer

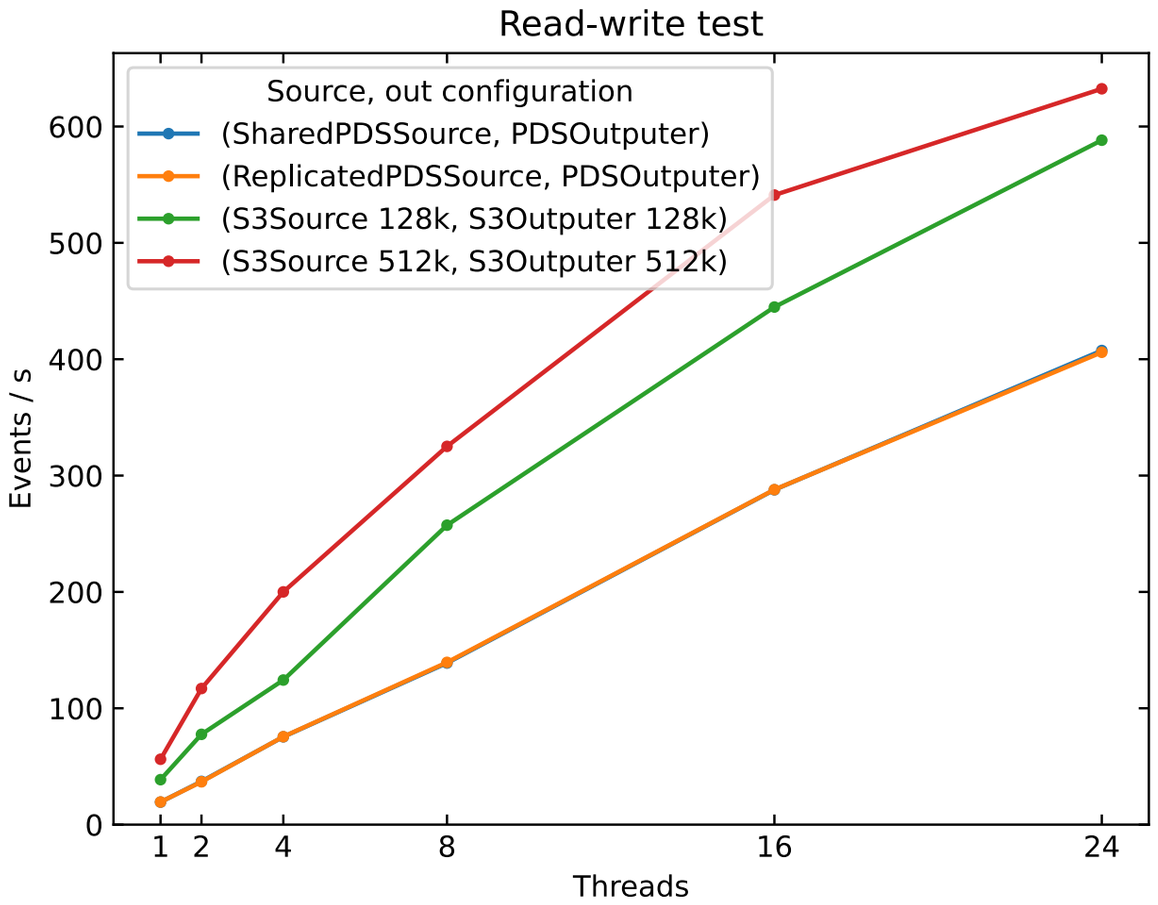
- Read-only chain: fetch→decompress→deserialize
- Repeating same test, with different bucket configurations
  - Baseline: EC4+2, bucket index in rep3 ssd pool
  - “no index” disable bucket index
  - “Rep3” data in 3x replica hdd pool
- Vary also S3Outputer config
  - No LZMA (very slow)



S3 vs. other Source/Outputers

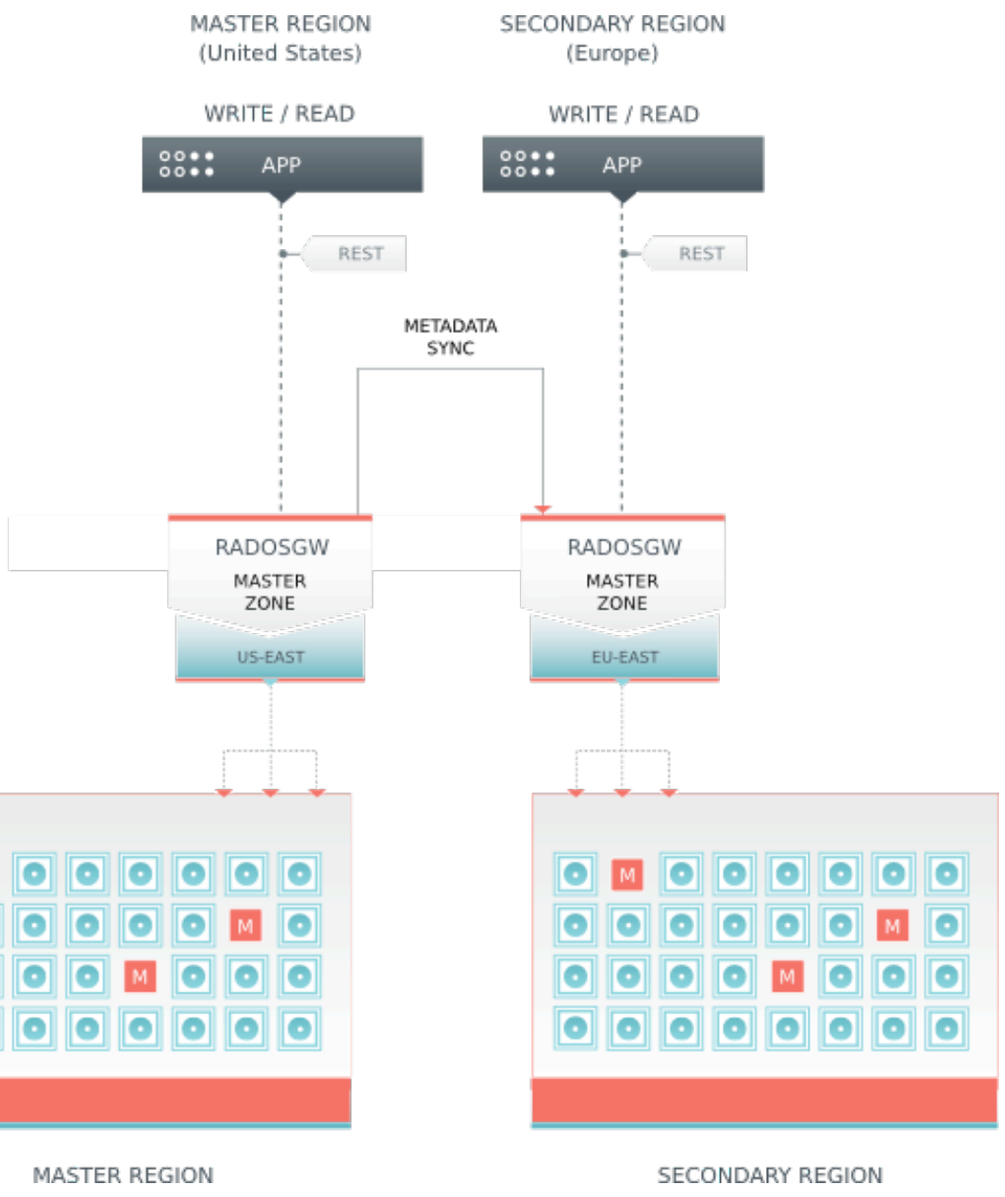
- ROOT source similar to a CMSSW grid job
  - Read file via xrootd, server has CephFS (same cluster) mounted
  - No ROOT outputer due to bug
- PDS source: write whole events sequentially
  - Very good thread scaling (last data point = all cores on machine)





Bonus: multi-site

- RadosGW, like Amazon S3, can have cross-zone sync
  - Further data resiliency in the lake
  - Probably still want edge caches



Fermilab

Conclusions

- Object stores provide clear new capabilities for data management
  - Reduce disk requirements
  - More flexibility in defining data tiers
- To fully utilize, much more software development needed
  - New data management service requirement: column tracking
  - Full data schema requires provenance, Runs/LumiSections TTree handling, etc.
- In a prototype, performance and thread scaling very favorable
  - Further scale tests needed (only few GB accessed so far)