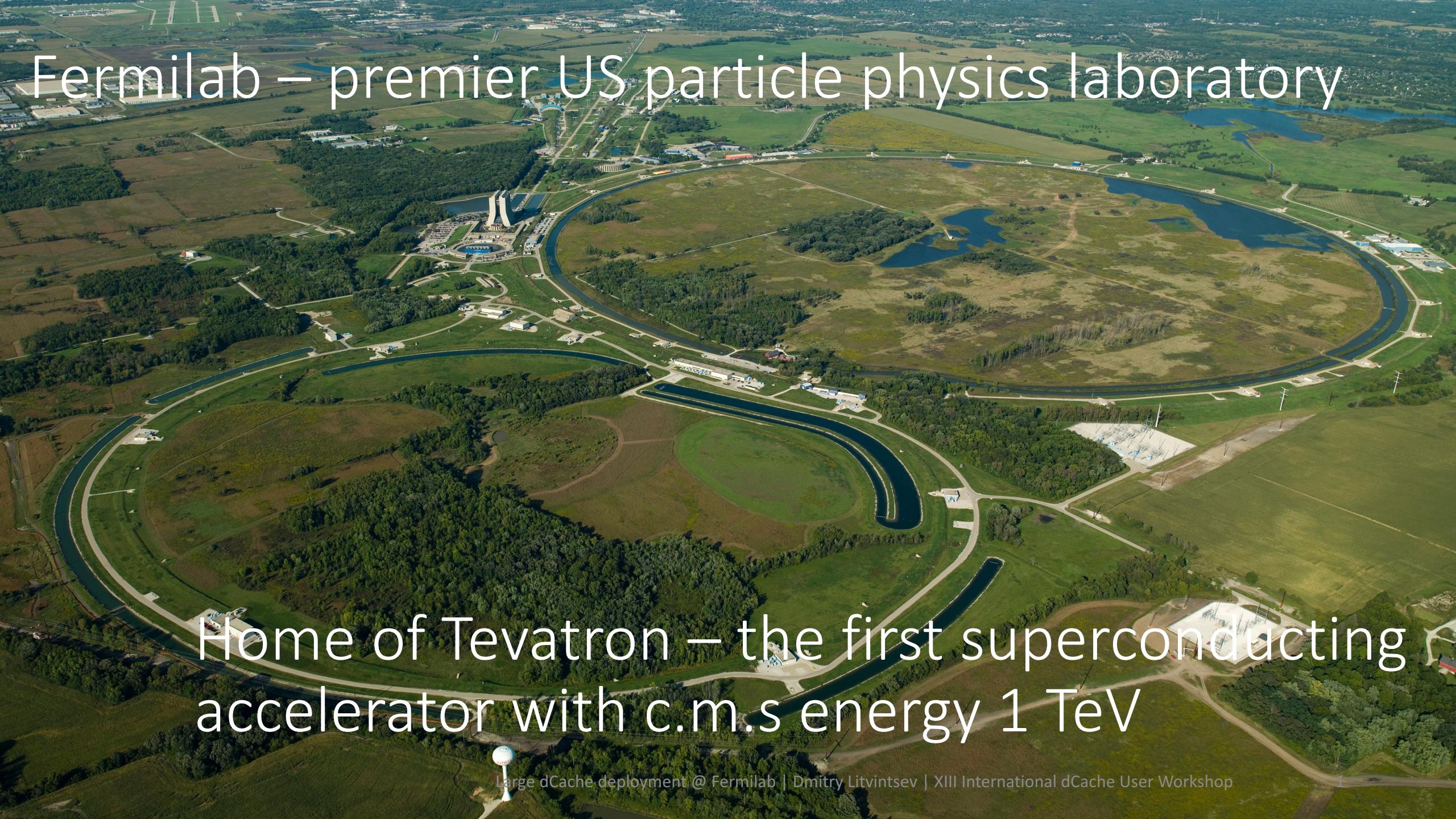


# Large dCache deployment @ Fermilab

Dmitry Litvintsev

13<sup>th</sup> International dCache user workshop

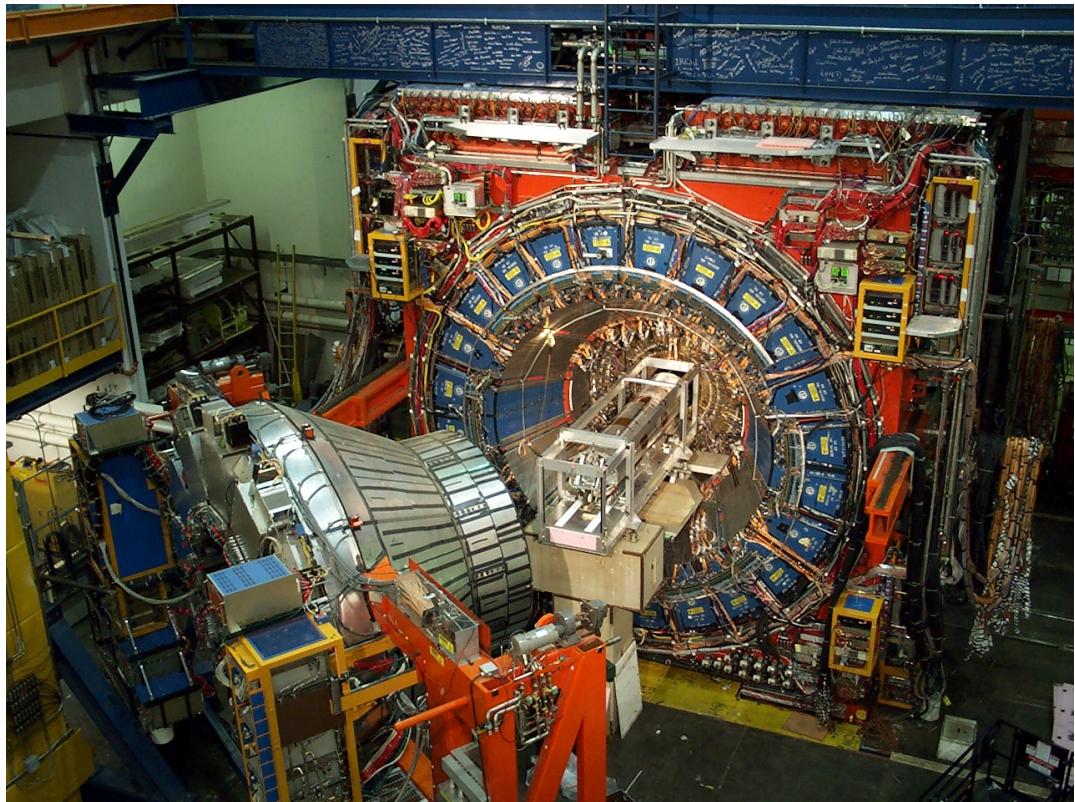
Madrid, Spain, May 21, 2019

An aerial photograph of the Fermilab complex, showing the large circular particle accelerator ring and various buildings and infrastructure. The facility is situated in a rural area with fields, roads, and some water bodies.

Fermilab – premier US particle physics laboratory

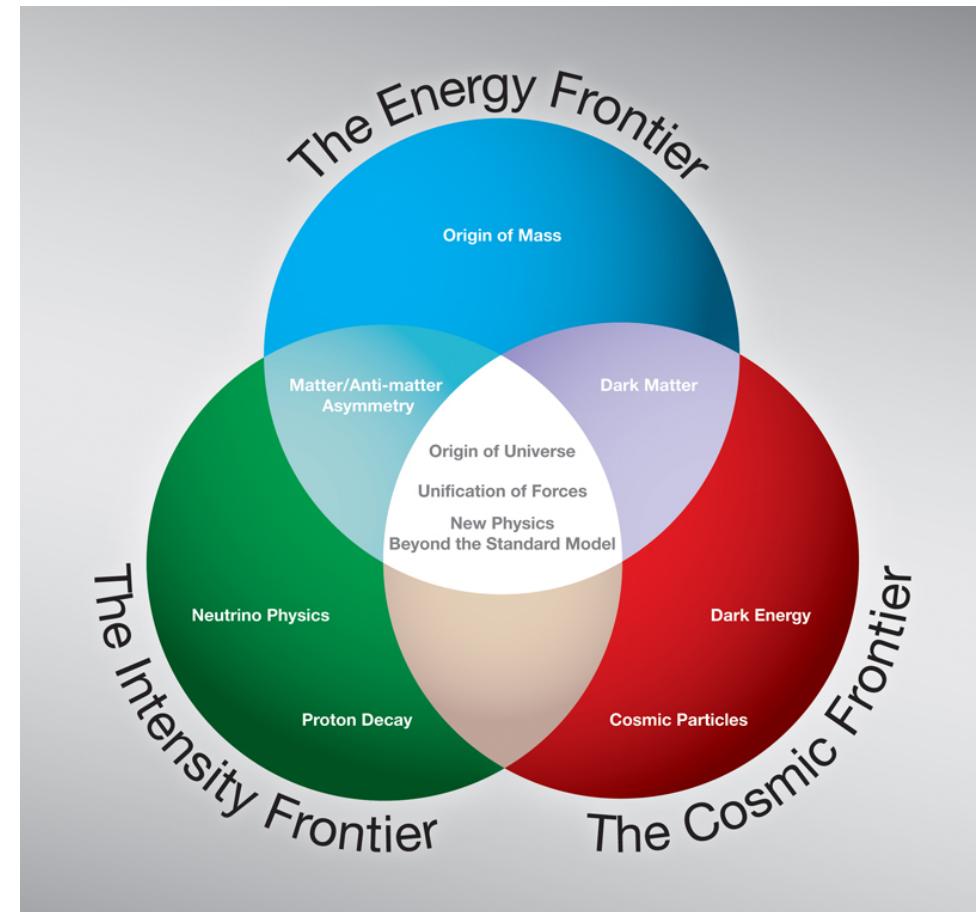
Home of Tevatron – the first superconducting  
accelerator with c.m.s energy 1 TeV

# Fermilab: home of CDF and Bisons



# Fermilab : pursuit of discoveries

- Started as HEP lab:
  - Discovery of b-quark (Nobel prize)
  - Discovery of t-quark
  - Observation of tau-neutrino
  - Observation of direct CP violation in Kaon decays
  - Observation of CP violation in B-decays
  - Observation of Bs oscillations
- Tevatron shut down in 2011
- Expanding scientific reach:
  - LQCD
  - Astrophysics
  - Neutrino Physics
  - Rare/exotic, BSM phenomena
  - Accelerator science
  - Hosting CMS T1 and USLPC

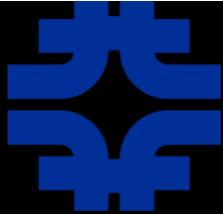


Success of the Fermilab scientific program depends  
crucially on Computing and Storage Resources

Fermilab uses dCache and Enstore as principal  
disk/tape storage technology

This talk about dCache instances at Fermilab

# Back in 2012

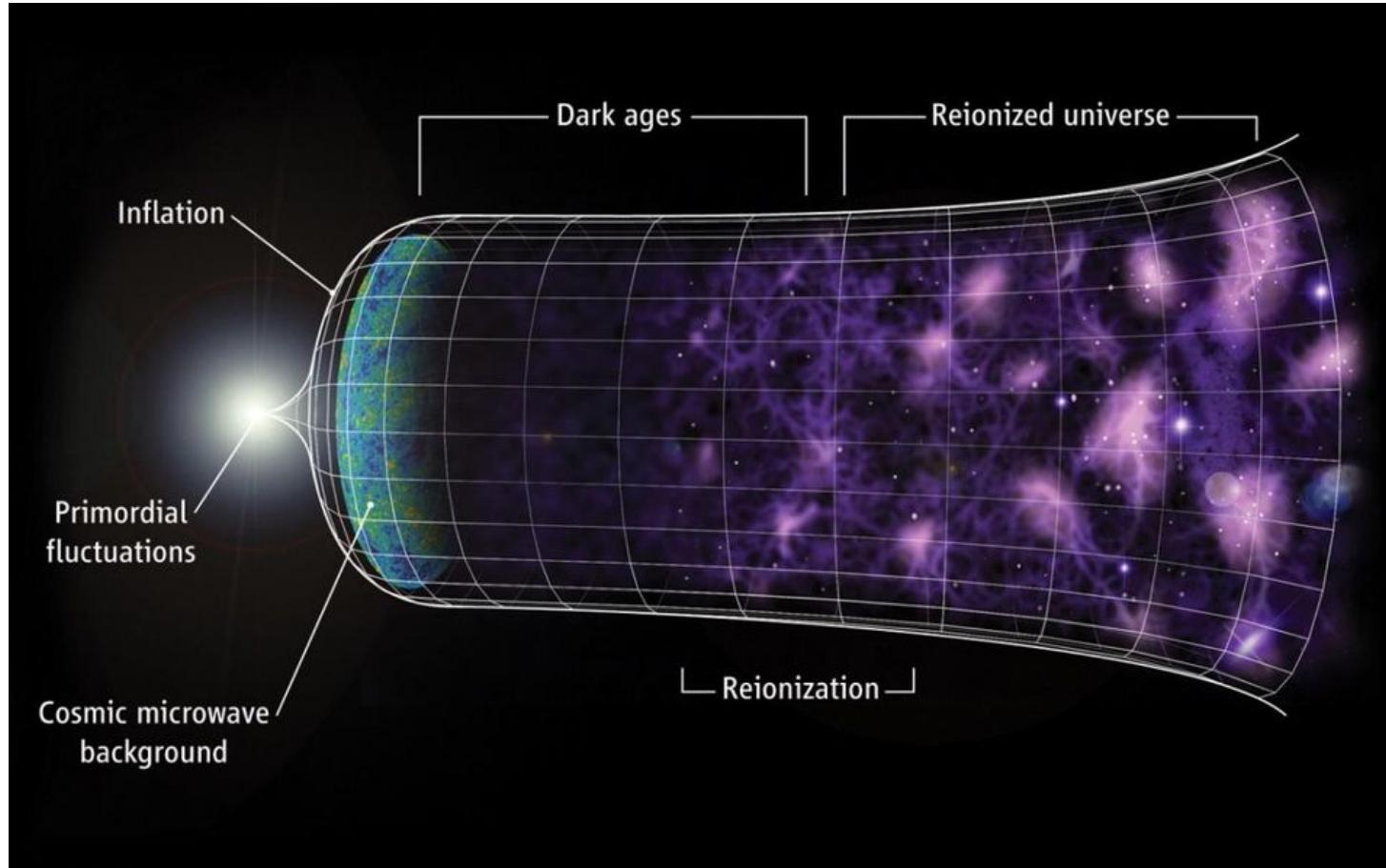


## dCache @ Fermilab

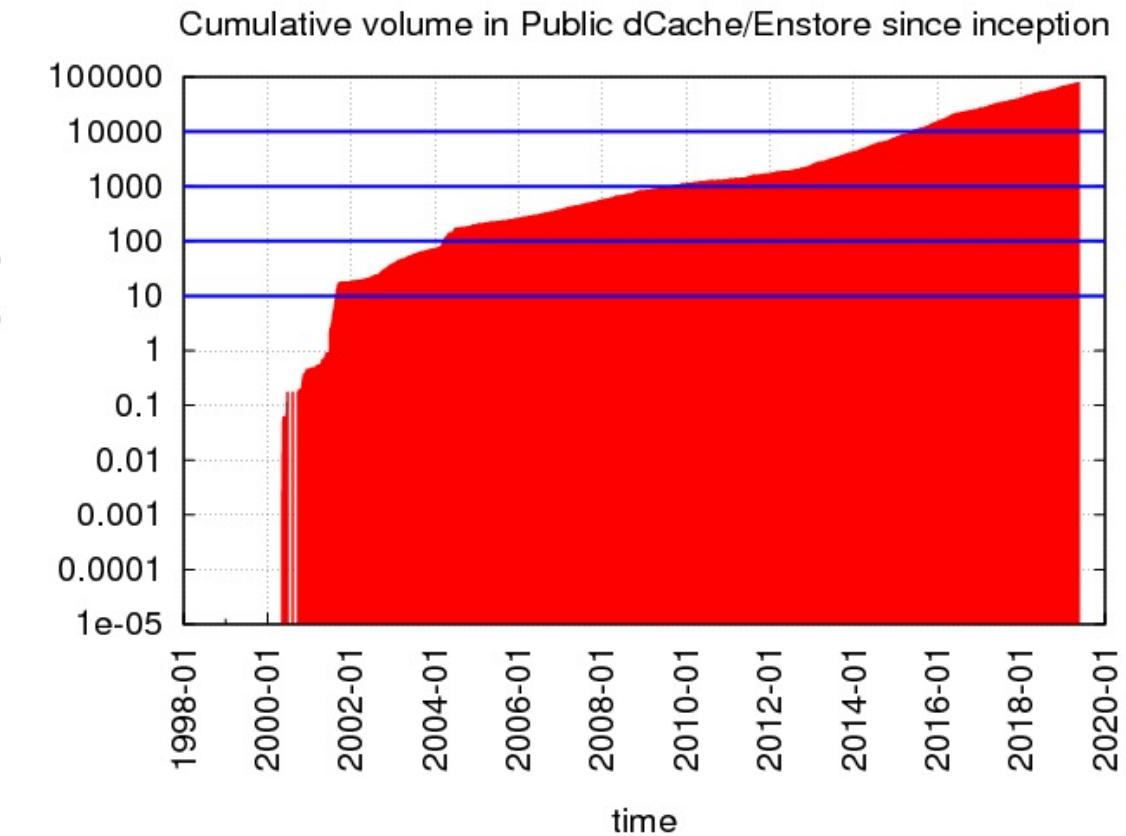
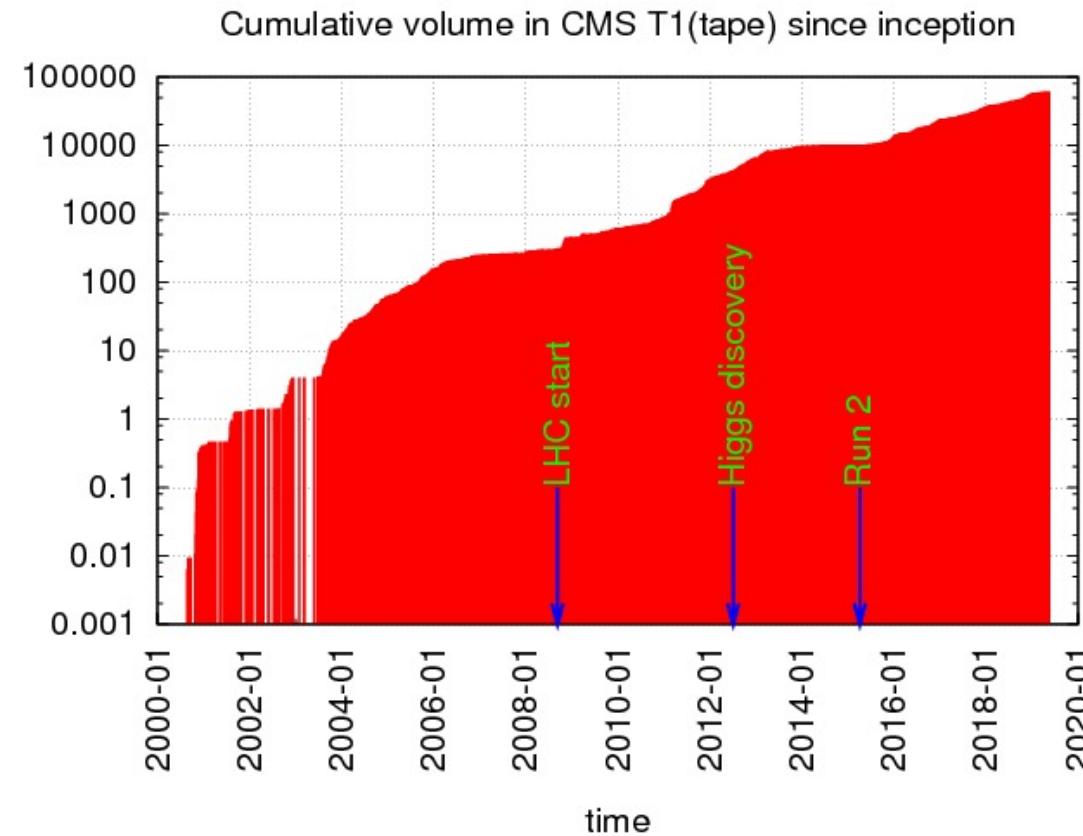
	total online	on tape	version
public	0.1 PiB	2.5 PiB	1.9.5-28
CDF	1.5 PiB	10 PiB	1.9.5-22
CMS	13.5 PiB	17 PiB	1.9.5-23

Size ↓

# Inflation stage



# Exponential data explosion



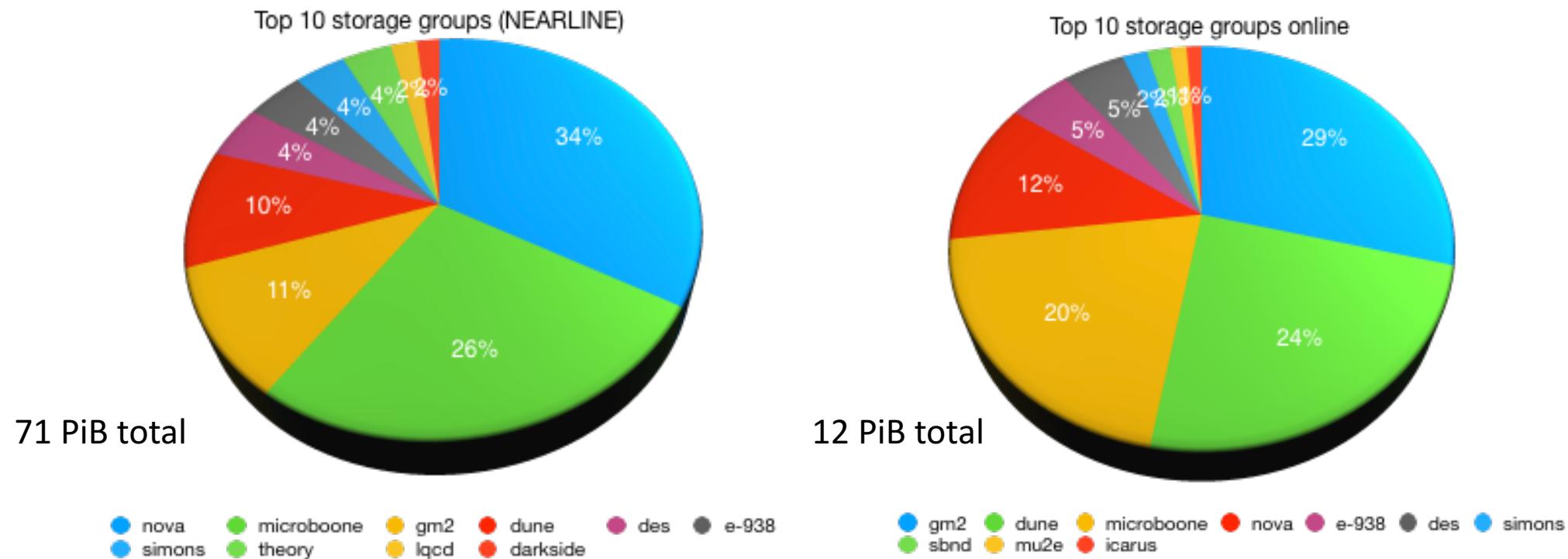
# Large dCache instances at Fermilab

Instance	total online	total tape	version
CDF	0.64 PiB	10.0 PiB	2.2.29-1
CMS T1 (tape)	1.7 PiB	60.0 PiB	2.13.53-1
Public	11.6 PiB	71.2 PiB	4.2.4-32
CMS T1 (disk)	22.7 PiB	0	2.13.53-1

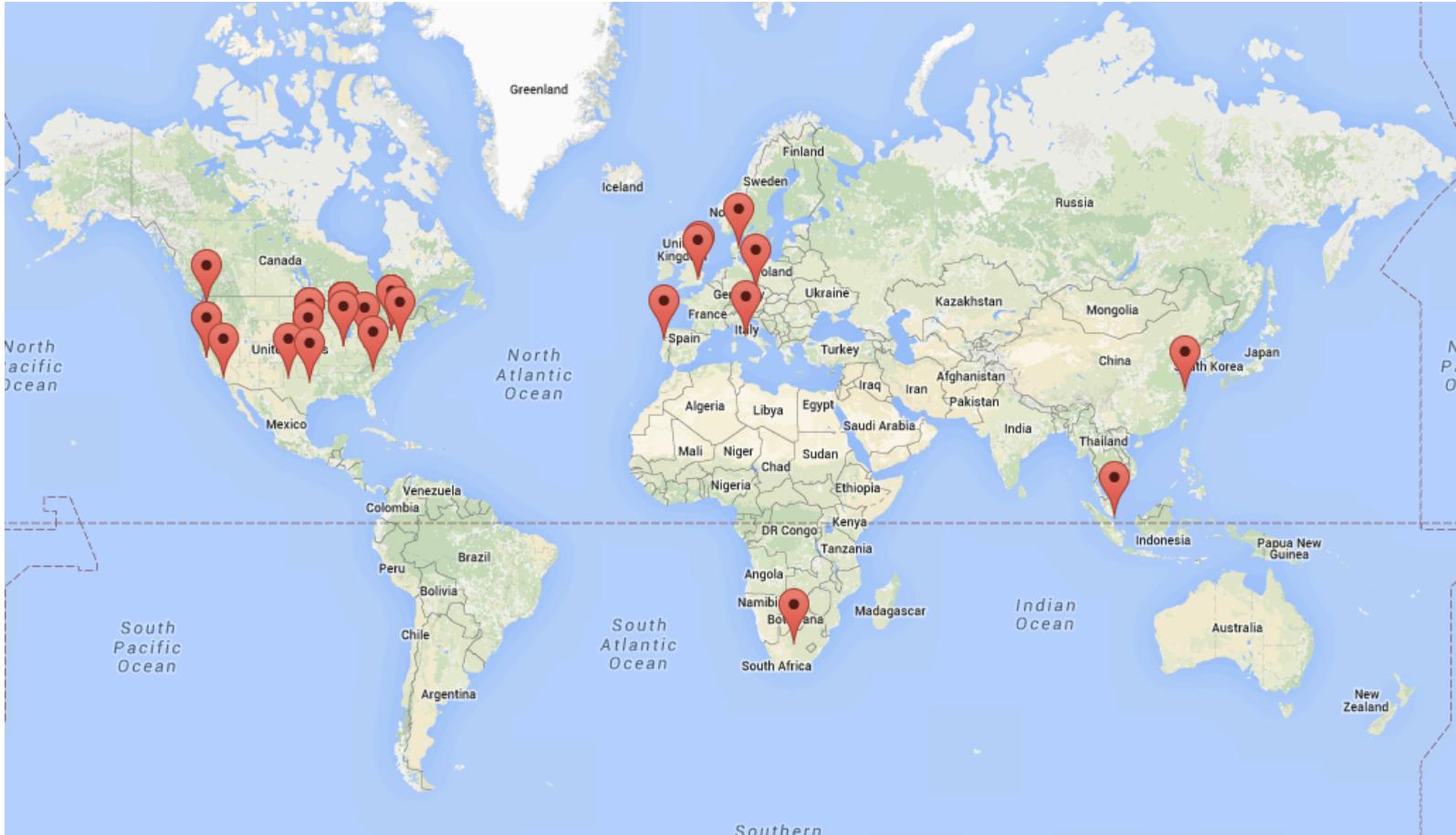
# Generalities

- CMS T1 dCache:
  - tape attached instance
    - Single VO, highly organized and restricted access to tape
  - disk-only instance
    - Used for data access. Plays a role of disk cache for tape attached instance with data placement controlled by PheDex
    - Single VO, almost exclusively xrootd access.
- Public dCache
  - tape attached.
    - No restrictions on user tape access
  - seriously multi-tenant (about 100 VOs (or experiment groups)). Individual users are members of multiple VOs requiring access to the data across VOs.
  - multi-protocol data access.
- Tape backend:
  - Enstore managed tape robotic libraries
  - SL8500 being decommissioned, transitioned to IBM TS4500)
  - LTO8 media (migrating existing LTO4, T10K media to LTO8)
- dCache Personnel:
  - 2 admin FTE + 1.5 development FTE.

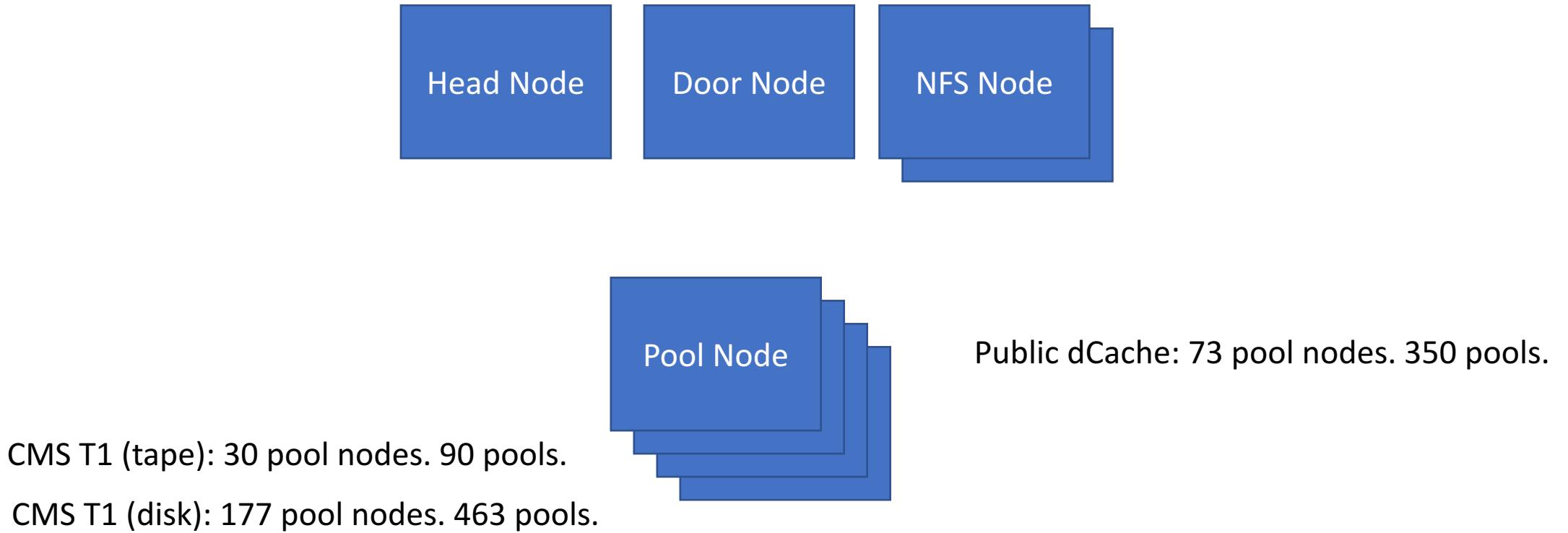
# Public dCache data distribution by VO



# Public dCache: Locations that moved at least 1TB of data in a week



# Setup



# Public dCache Setup

- Head Node:
  - admin, alarms, frontEnd, history, http, info, PoolManager, gPlazma, Zookeeper (embedded)
- Door Node:
  - dCap (plain, Kerberos, GSI)
  - XrootD (GSI, anonymous r/o)
  - (G)FTP (GSI, Kerberos, password auth)
  - WebDAV (GSI, anonymous r/o)
  - SRM (both front and back-ends)
  - PinManager
- NFS Node:
  - PnfsManager
  - NFS server
  - cleaner
- Pool Node
  - multiple pools
  - GFTP door
- CMS is similar. CMS only runs GSI XRootD doors and GFTP doors.
- Not using SRM space manager.

# Hardware

- NFS node (hosts chimera DB)

Instance	CPU	Memory	Disk
CMS	32 x AMD Opteron, 6320 @ 2.8 GHz	128 GiB	2 x 2.2 TB, SSD, RAID1
Public	32 x Intel Xeon E5-2650 v2 @ 2.60GHz	128 GiB	2 x 3.5 TB, SSD, RAID1 (Intel Enterprise SSD)

- Chimera DBs are replicated to identical hosts
- CMS uses postgresql 9.2, Public dCache 9.6

# Hardware

- Disk storage : Nexan e60, RAID6, FC attached to pool nodes.
- Pool nodes – server class box purchased at whatever optimal price point at the time.
  - Public dCache:
    - From 32 x AMD Opteron 6320 @ 2.8 GHz, 64GB RAM to 40 x Intel Xeon E5-2640 v4 @ 2.40GHz, 128GB RAM
    - CMS T1 (tape):
      - 16 x Intel Xeon Silver 4110 @ 2.10GHz, 96 GB RAM
    - CMS T2 (disk)
      - 32 x AMD Opteron 6128 @ 2 GHz, 64 GB RAM to 40 x Intel Xeon E5-2630 v4 @ 2.20GHz, 128GB RAM
  - Head and Door nodes are similar to pool nodes sans attached RAID
  - Each node has 10 Gb/s connectivity
  - SLF6

# Pool node partitioning

- Typical CMS layout

Filesystem	Type	Size	Used	Avail	Use%	Mounted on
/dev/sdb	xfs	65T	29T	37T	44%	/storage/data1
/dev/sdc	xfs	65T	30T	36T	45%	/storage/data2
/dev/sdd	xfs	65T	29T	37T	44%	/storage/data3

- Typical Public dCache layout

Filesystem	Type	Size	Used	Avail	Use%	Mounted on
/dev/sdb1	xfs	73T	20T	54T	27%	/diska
/dev/sdc1	xfs	73T	66T	7.3T	90%	/diskb
/dev/sdd1	xfs	73T	20T	54T	27%	/diskc

- Public dCache pools share partitions
- If Pool Node hosts HSM-backed pool, it mounts dCache namespace via NFS v3 served by secondary NFS server so that user NFS traffic with all associated issues do not impact storing to tape. Encp (Enstore copy client) uses NFS when writing files to tape.

# Configuration management

- CMS : puppet
- Public dCache: configuration RPM. Pre-dates puppet days. Transition to puppet is foreseen at some point.
- What is configuration RPM? When installed, a bash script is run that creates dCache environment based on roles assigned to a node based on hostname match. Pretty simple and bulletproof.
- Disadvantage – no enforcement of no local changes.

# Admin Scripting

- The system is controlled and monitored with help of mixture of `check\_mk` and cron job scripts written in jython, bash and python that utilize:
  - admin shell
  - infoProvider
  - and now restful API
- Rely on dCache Alarms service

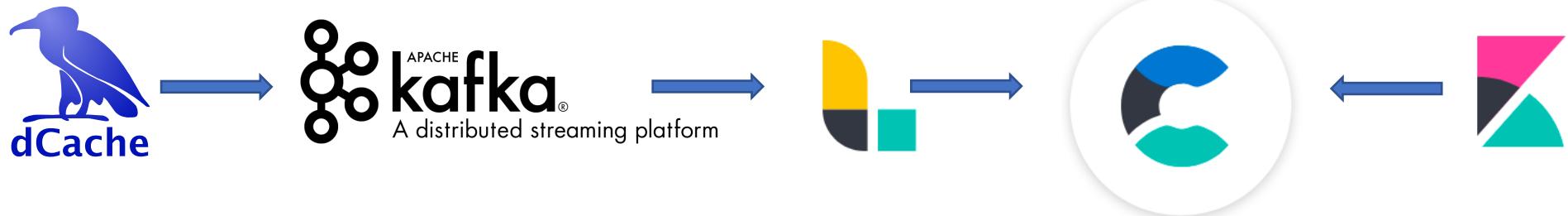
# PagedCache: test all doors

```
[root@fnca2a pagedcache]# python pagedcache.py fnca
2019-05-19 11:38:01 : SUCCESS for WebDAV-fnca4a-2@webdavDomain1
2019-05-19 11:38:01 : SUCCESS for DCap03-fnca4a@dcap03-fnca4aDomain
2019-05-19 11:38:01 : SUCCESS for DCap00-fnca4a@dcap00-fnca4aDomain
2019-05-19 11:38:01 : SUCCESS for DCap01-fnca4a@dcap01-fnca4aDomain
2019-05-19 11:38:01 : SUCCESS for Xrootd02-fnca4a@xrootd02-fnca4aDomain
2019-05-19 11:38:03 : SUCCESS for GFTP-stkendca43a@gridftp-stkendca43aDomain
2019-05-19 11:38:03 : SUCCESS for GFTP-stkendca11a@gridftp-stkendca11aDomain
2019-05-19 11:38:12 : SUCCESS for DCap-Kerberos-00-fnca4a@kerberosdcap00-fnca4aDomain
2019-05-19 11:38:14 : SUCCESS for SRM-fnca4a@srm-fnca4aDomain

.....
2019-05-19 11:38:31 : SUCCESS for KFTP-fnca4a@kerberosftp-fnca4aDomain
2019-05-19 11:38:31 : SUCCESS for KFTP-fnca3a@kerberosftp-fnca3aDomain
2019-05-19 11:38:31 : Finish
```

<https://github.com/DmitryLitvintsev/scripts/tree/master/python/dcache/pagedcache>

# Monitoring and data analysis



```
dcache.enable.kafka=true  
dcache.kafka.bootstrap-servers=lssrv03:9092,lssrv04:9092,lssrv05:9092  
dcache.kafka.topic=ingest.dcache.billing
```

<https://landscape.fnal.gov/kibana/goto/b5c88983e8d3ba36553841fd73ecd00c>

# Upgrades

- Major dCache upgrades
  - character building exercises
  - atonement for whatever sins you might have committed
- For years stayed behind, letting other brave souls to take the bullet.
  - still we seem to be tickling dCache in all the wrong places
- With 4.2 decided to be on the forefront as we needed RESTful and web monitoring
  - advantage is that now we can apply most recent patches w/o having to adapt them too much, say if we ran way older versions.
- Upgrade all at once. No rolling upgrades.

# Pool grouping

- CMS instances each have 1 large pool group that aggregates space of all available pools.
- Public dCache has far more complex pool group structure owing to multi-tenancy:
  - `readWritePools` – shared read/write pools, HSM attached.
    - Resilient pool group. Replication factor 20 for a specific storage class. Intended for scaling of delivery of the same file to thousands of jobs (typically tar files with code libraries or calibration/geometry constants).
  - A few write only pool groups, each “owned” by different VO/experiment
  - `AnalysisPools` (a.k.a. persistent pools) are disk-only pool groups per VO/experiment. Intended for user and production job output.
  - `PublicScratchPools` – shared volatile pools intended to store files temporarily.

# Pool selection

- CMS instances - no network or storage group specific pool selection exist. Use random pool selection partition.
- Public dCache pools use WASS partition with significant cpufactor weighting.
- Pool selection is based on directory tags:

```
psu create unit -store dune.persistent@enstore
psu addto ugroup DuneAnaSelGrp dune.persistent@enstore
psu create pgroup DuneAnalysisPools
psu addto pgroup DuneAnalysisPools p-dune-stkendca41a-4
...
psu create link DuneAnalysis-link DuneAnaSelGrp any-protocol world-net
psu set link DuneAnalysis-link -readpref=20 -writepref=20 -cachepref=0 -
p2ppref=-1 -section=persistent
psu add link DuneAnalysis-link DuneAnalysisPools
```

# Directory structure

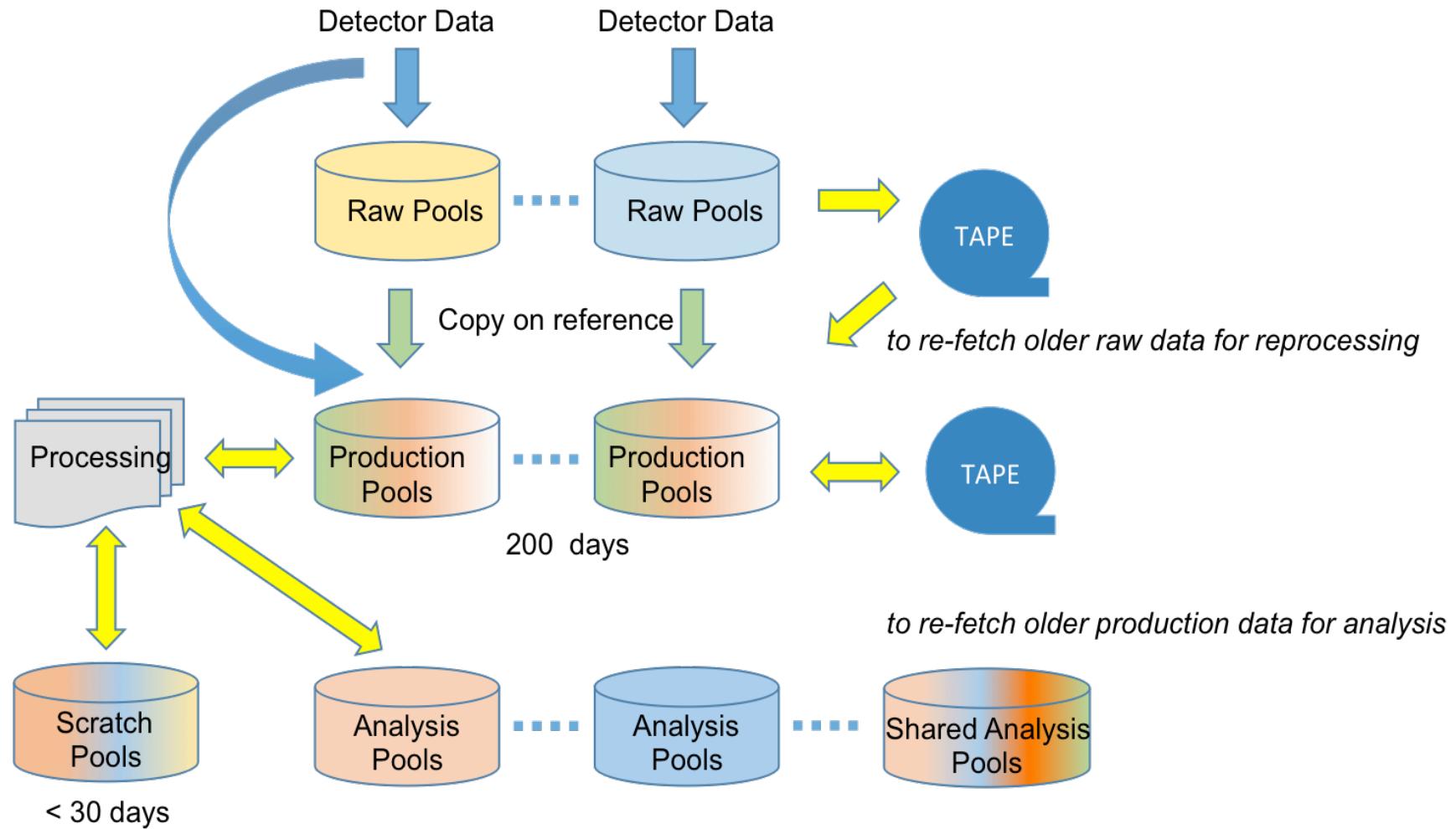
```
[root@fnndca2a dune]# pwd  
/pnfs/fNAL.gov/usr/dune  
[root@fnndca2a dune]# ls -al  
total 4  
drwxr-xr-x 8 50381 9010 512 May 1 2018 .  
drwxr-xr-x 113 root root 512 May 18 11:06 ..  
drwxrwxr-x 3 50381 9010 512 May 1 2018 archive  
drwxrwxr-x 19 50381 9010 512 Mar 28 15:55 persistent  
drwxrwxr-x 5 50381 9010 512 May 17 2018 resilient  
drwxrwxr-x 16 50381 9010 512 Apr 25 14:43 scratch
```

# Directory tags

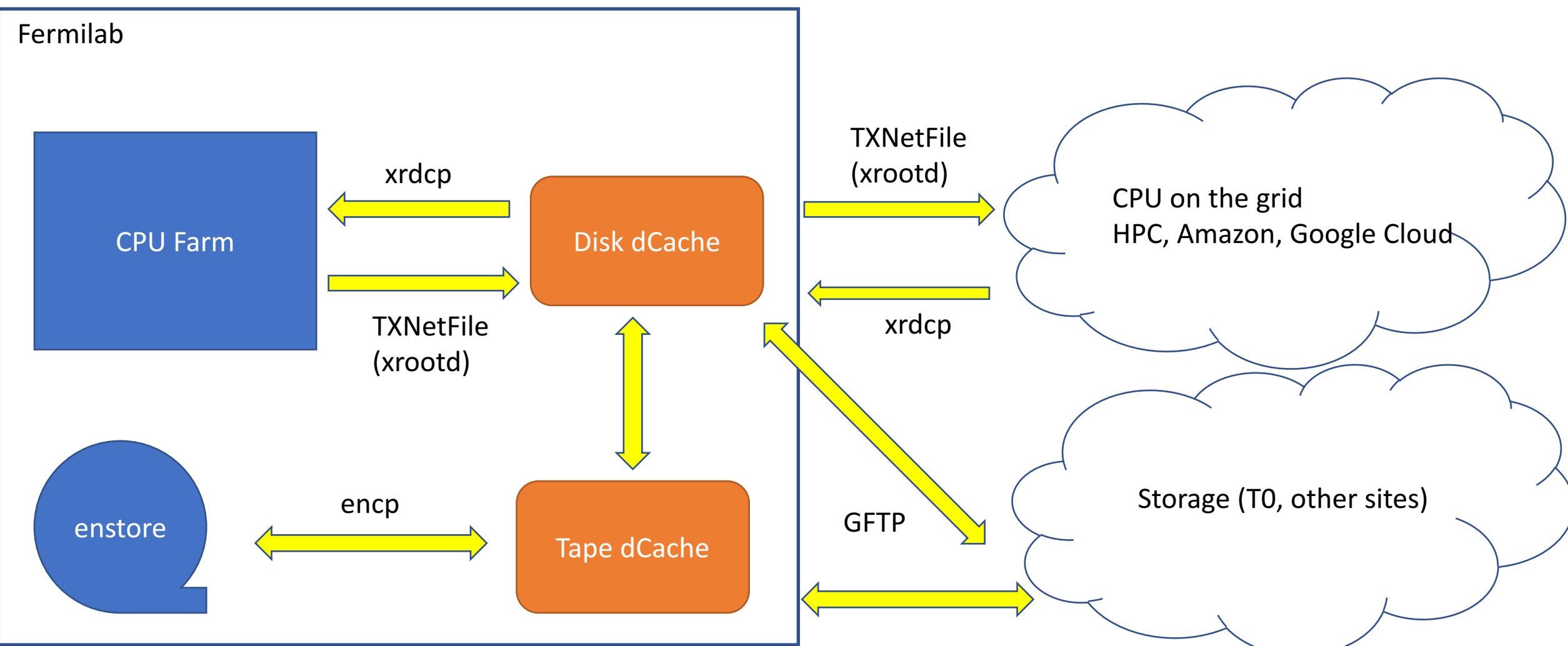
```
root@fnndca2a dune]# cd persistent/  
[root@fnndca2a persistent]# grep "" $(cat ".(tags) ()")  
. (tag) (AccessLatency) :ONLINE  
. (tag) (file family) :persistent  
  . (tag) (file_family_width) :1  
  . (tag) (file_family_wrapper) :cpio_odc  
  . (tag) (library) :CD-LTO8F1  
  . (tag) (OSMTemplate) :StoreName sql  
  . (tag) (RetentionPolicy) :REPLICA  
  . (tag) (sGroup) :chimera  
  . (tag) (storage_group) :dune
```

StorageClass = "dune" + "." + "persistent" + "@" + "enstore" (hsm name)

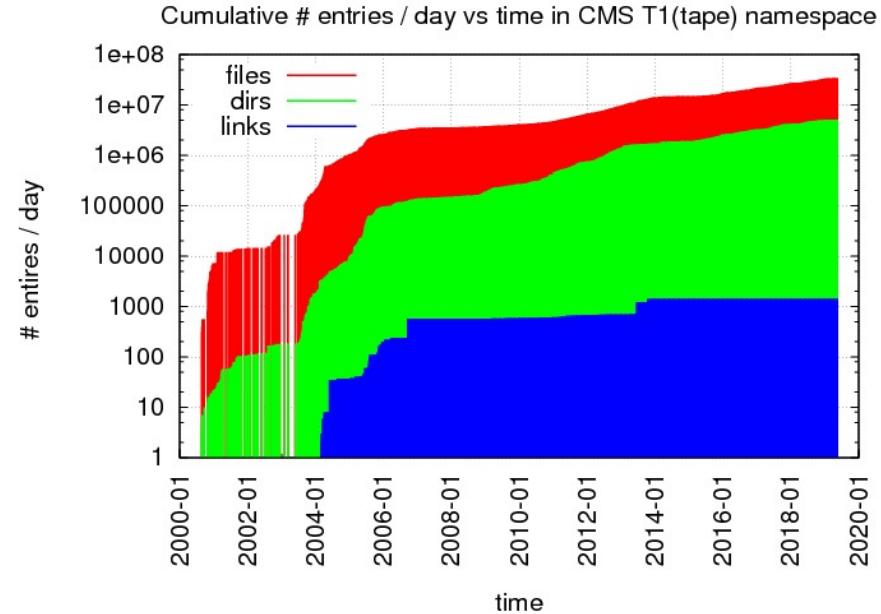
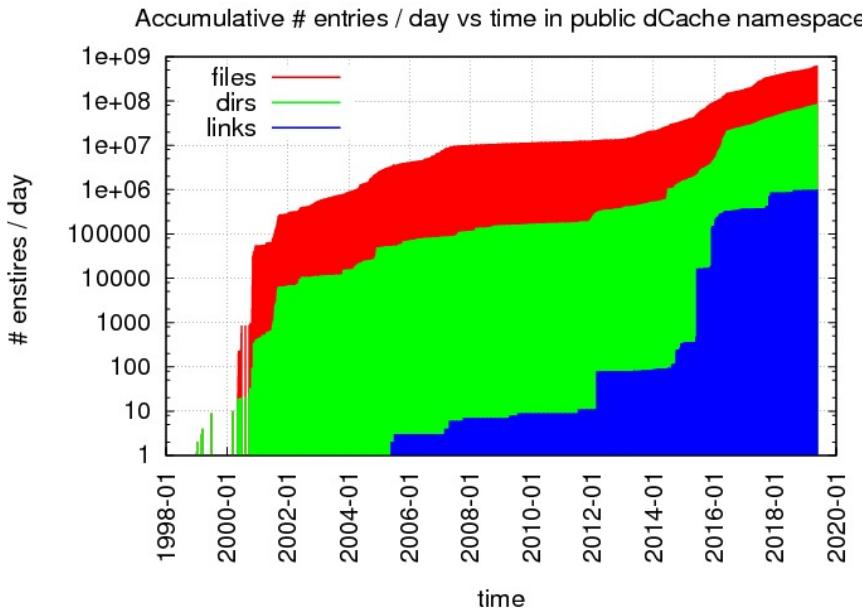
# Public dCache suggested workflow



# CMS workflow



# Explosion of namespace entries



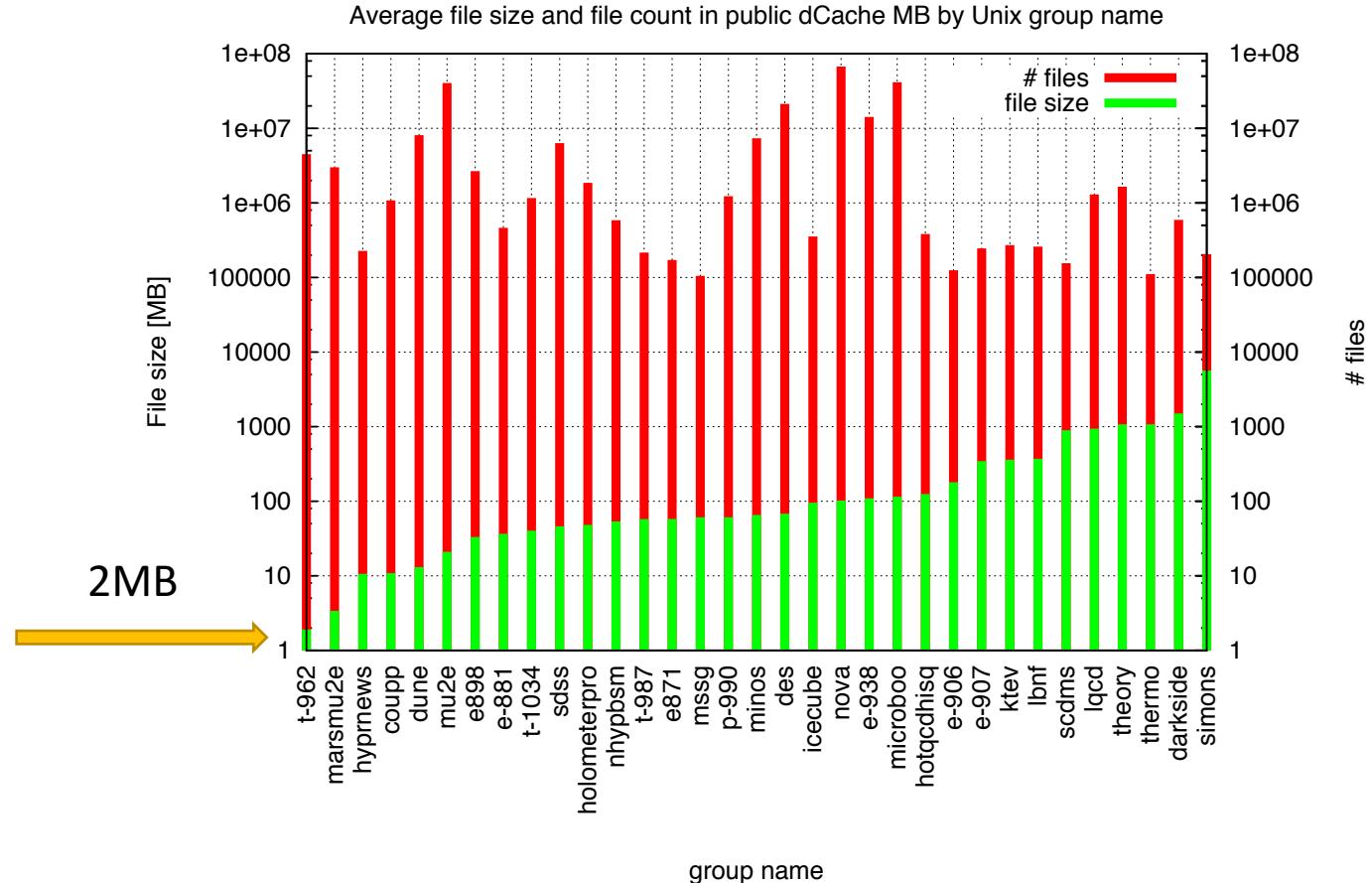
Entry type	Public dCache	CMS
files	548,636,084	29,298,537
dirs	83,652,764	5,101,037
volume	87,565,270,577,104,816	67,448,190,397,856,464
<file size>	152 MiB	2.1 GiB

# Chimera DB size

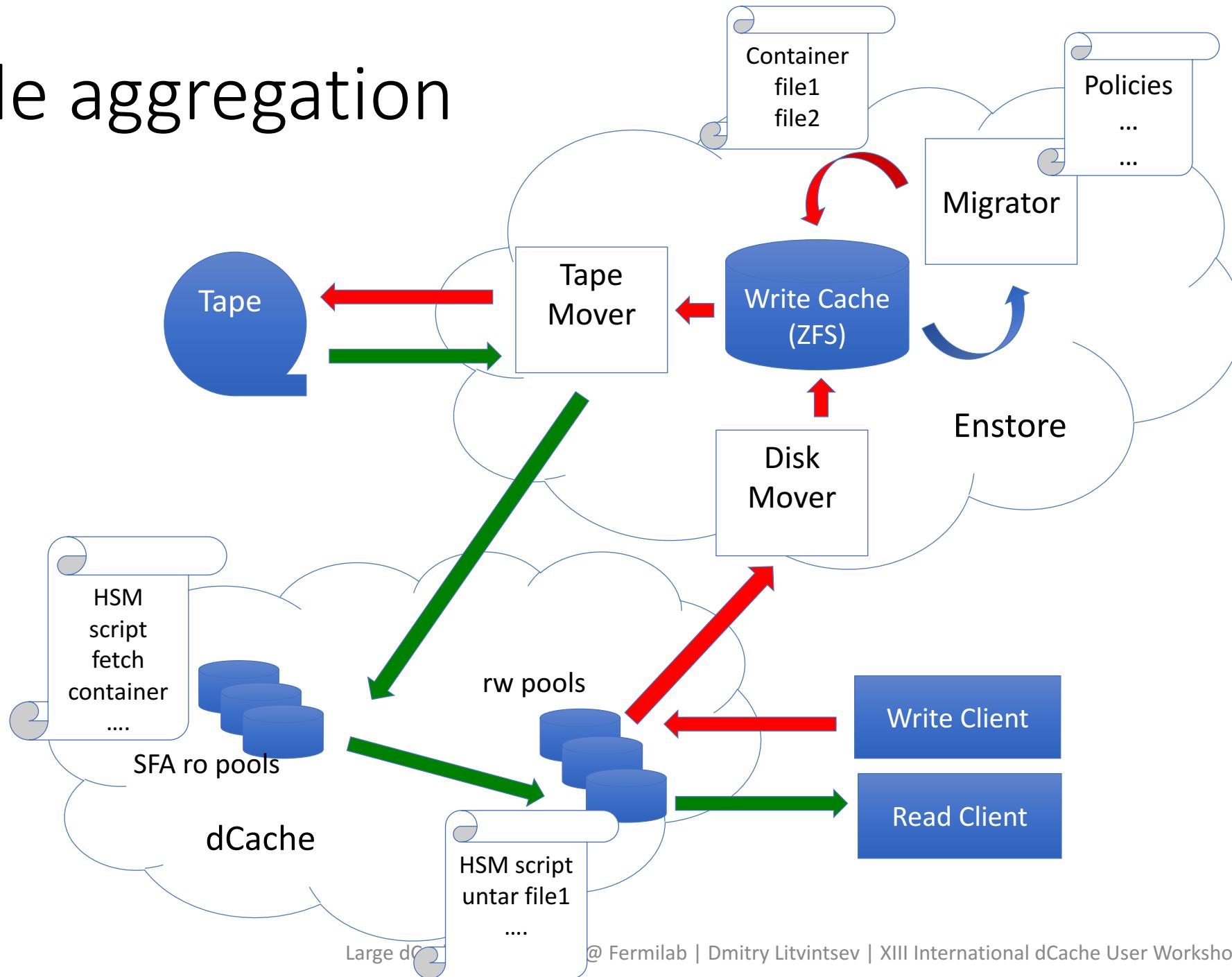
relation	total_size
public.t_inodes	706 GB
public.t_locationinfo	401 GB
public.t_dirs	180 GB
public.t_level_4	110 GB
public.t_level_2	97 GB
public.t_tags	94 GB
public.t_inodes_checksum	71 GB
public.t_level_1	32 GB
public.t_locationinfo_trash	23 GB
public.t_storageinfo	21 GB
(10 rows)	

# Public dCache: file sizes

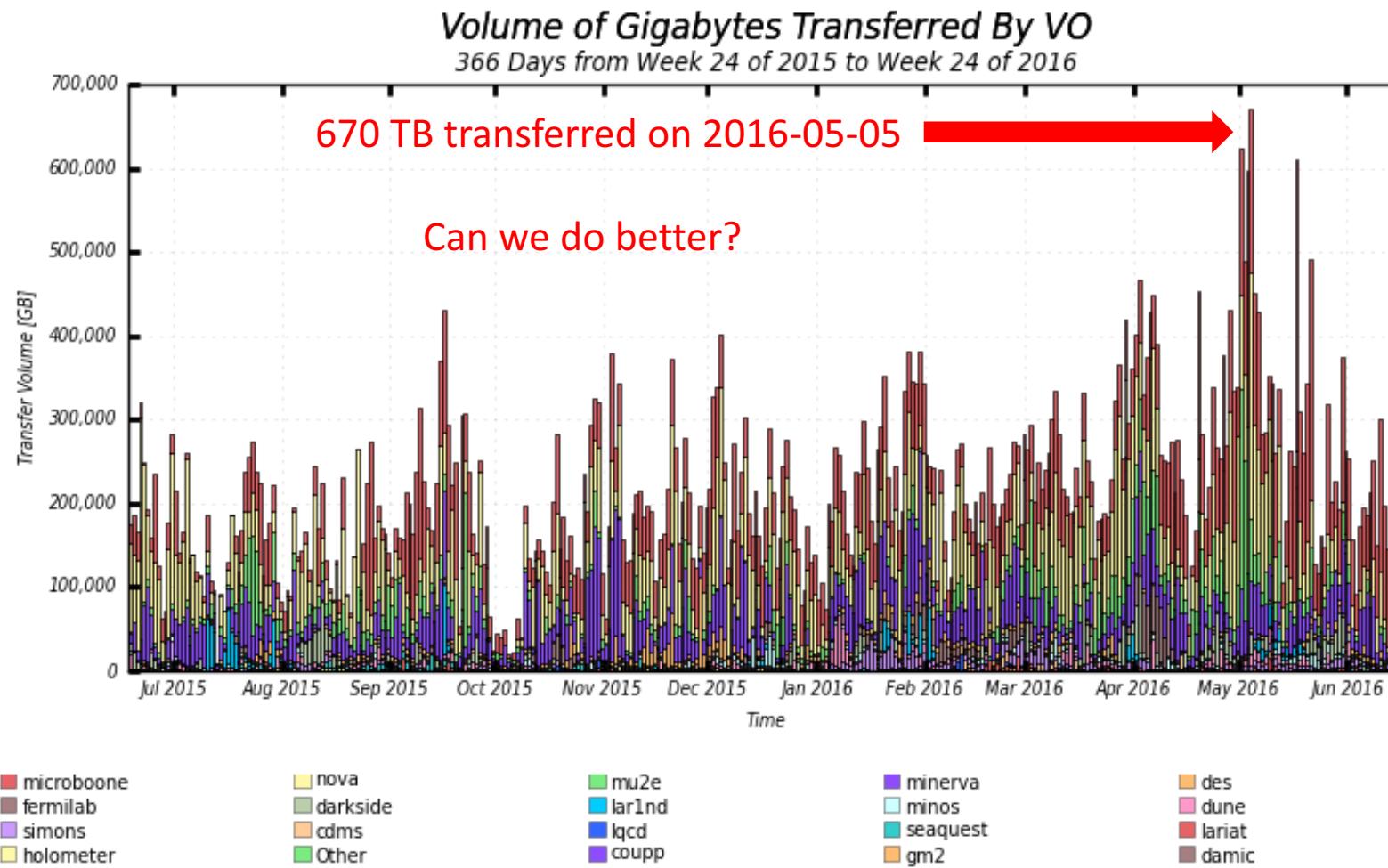
- All groups having > 100,000 files
- dCache is a distributed storage system and latencies associated with staging or even accessing each file over the network need to be taken into consideration. This is not a filesystem mounted off a local disk.
- To Protect tape access we use Small File Aggregation (SFA)



# Small file aggregation



# Public dCache transfer volumes



# Public dCache small file problem

- For the same amount of data small files cause:
  - Large number of files per directory.
  - Increase in depth and width of directory tree.
  - Datasets containing millions of files are harder to manage.
- dCache users are shielded from tape related overheads on writes as they occur out of band.
- **Read tape overheads per file contribute directly to “slow” file delivery for files that are not cached on disk.**
- Our current definition of small file is 200MB.
- Some small files may not belong to storage system (like file describing other file’s metadata, or constants files etc.). These data best belong to a database (SAM or calibrations/conditions DB).

# Small files

- Number of entries per directory

- Look what we have:

```
time ls /pnfs/fs/usr/xxx/xxx/xxx/xxx/xxx/DW_b2.13_24^3x64_ms0.04-mu0.005 | wc  
168827 168827 11307918  
  
real    0m49.801s  
user    0m2.642s  
sys     0m5.798s
```

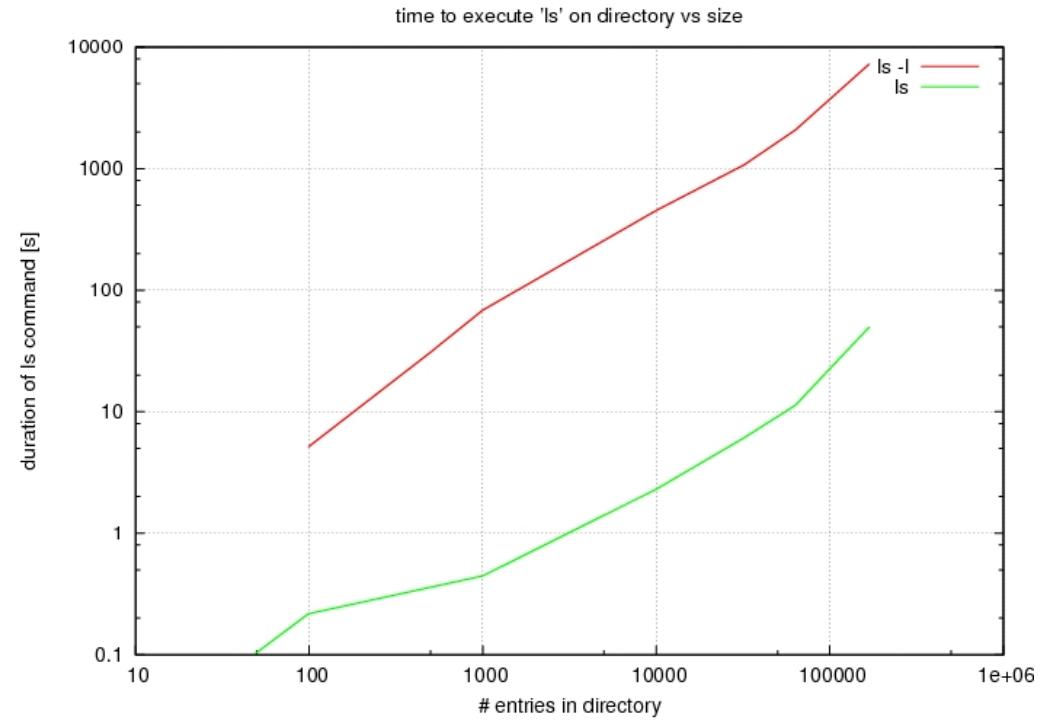
- 168827 / 50 sec, 3.3 kHz. Not too bad. But, try “-l” and it explodes:

```
time ls -l /pnfs/fs/usr/xxx/xxx/xxx/xxx/xxx/DW_b2.13_24^3x64_ms0.04-mu0.005 | wc  
168827 168827 11307918  
  
real  121m27.368s  
user  0m4.760s  
sys   1m36.412s
```

- Need to have reasonable number of entries per directory. Recommendation is under 2K / directory. Good news that with Chimera NFS, other users will not suffer when someone is running an ls on large directory (in contrast to PNFS before).

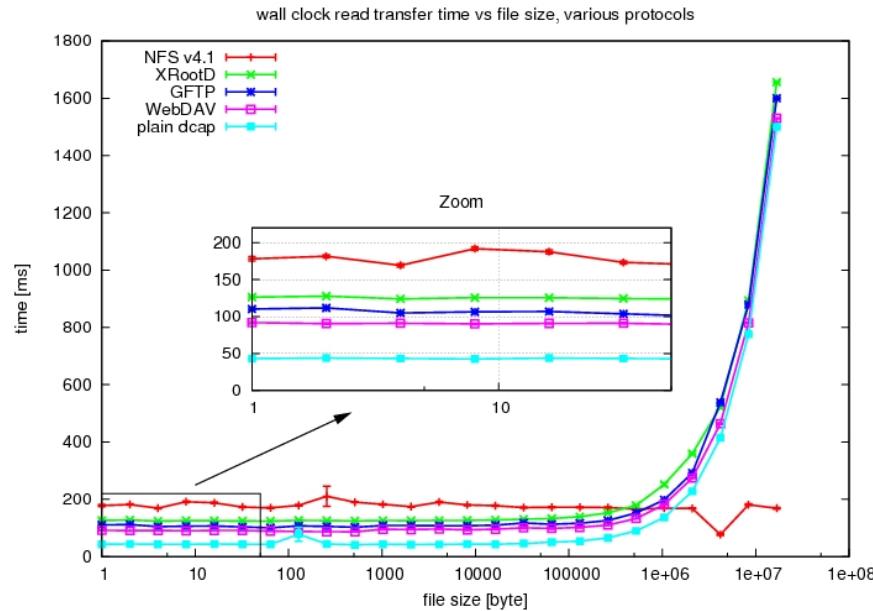
# `ls` timing vs directory size

- Directory listing in dCache is not fast
- dCache is about providing I/O rather than super-fast data catalogue



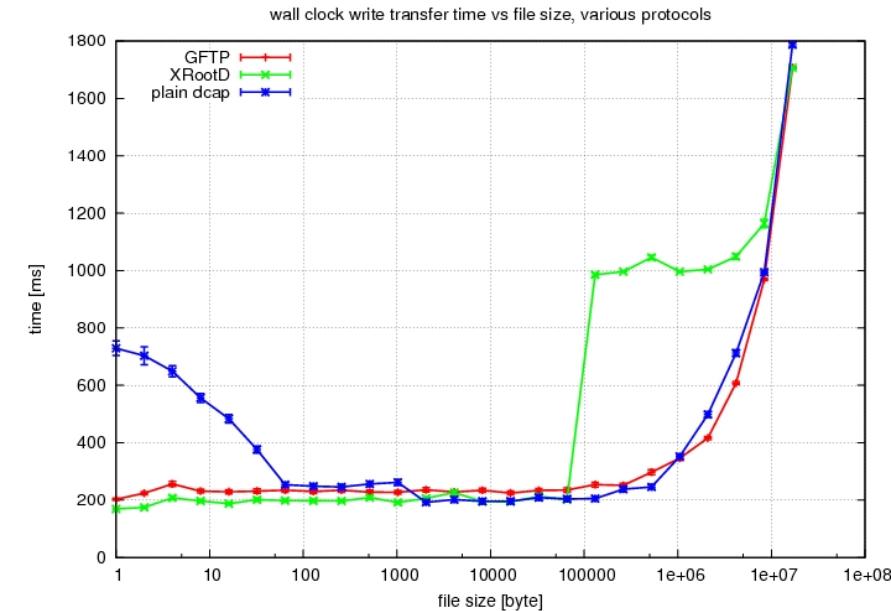
# Small file access latency for cached files.

- Reads



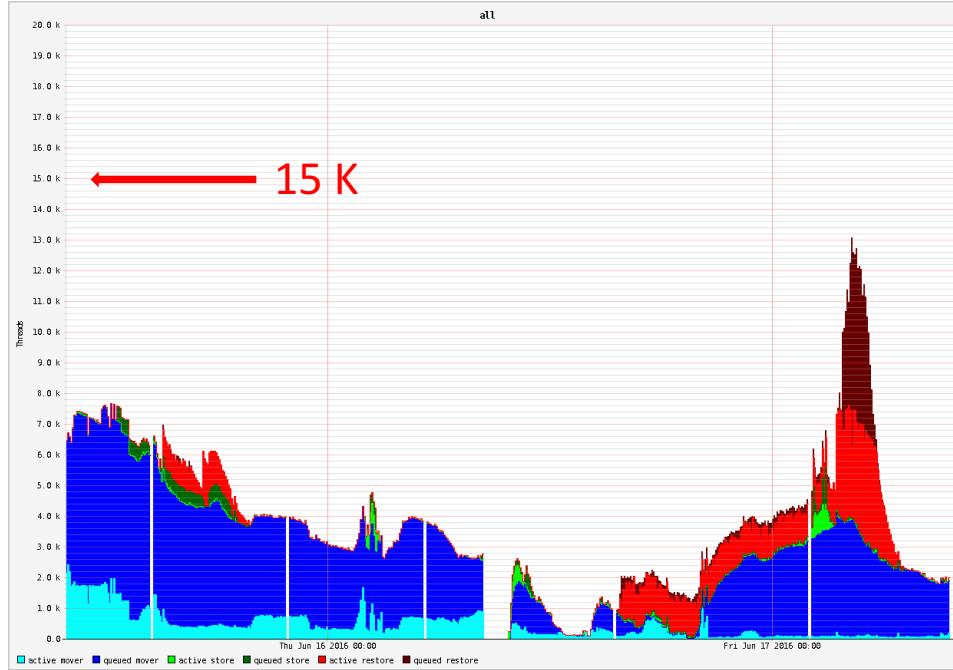
Protocol	Latency [ms]
plain dcap	43
WebDAV	90
GFTP	105
xrootd	126
NFS v4.1	175

- Writes



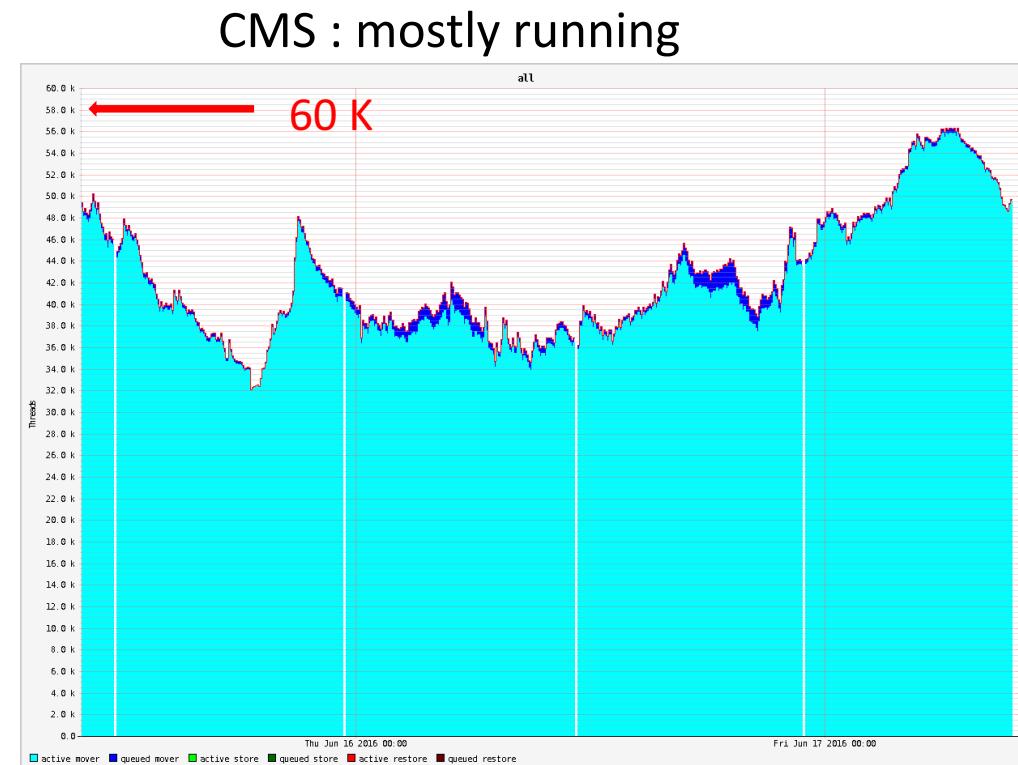
Protocol	Latency [ms]
xrootd	180
GFTP	200
NFS v4.1	300
plain dcap	729 (200)

# Public dCache vs CMS dCache (disk)



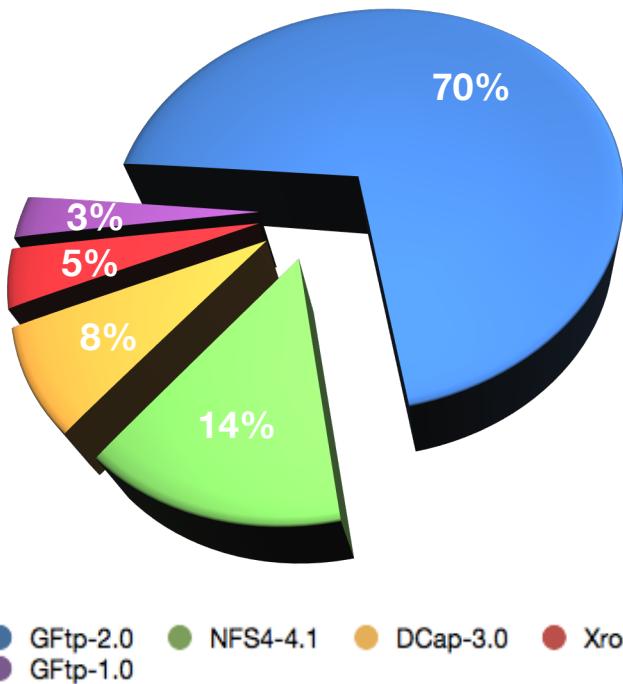
Public dCache : mostly queueing

Why?

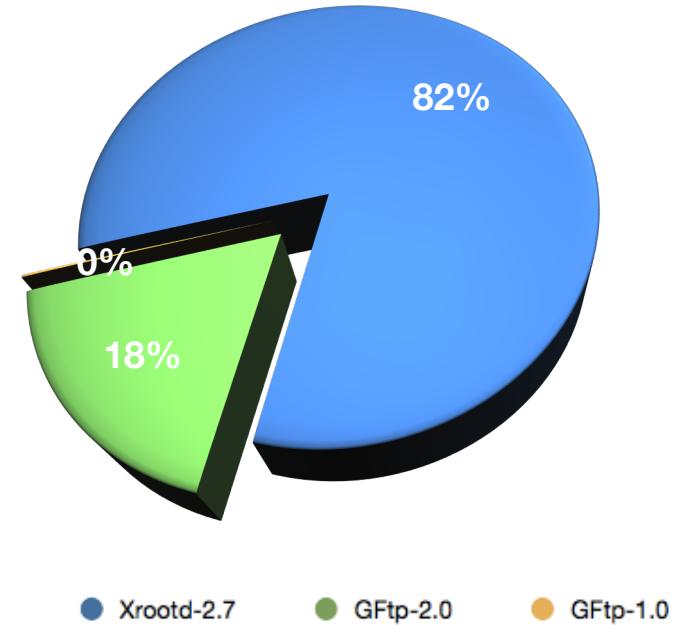


# Data access by protocol(by volume)

Public dCache



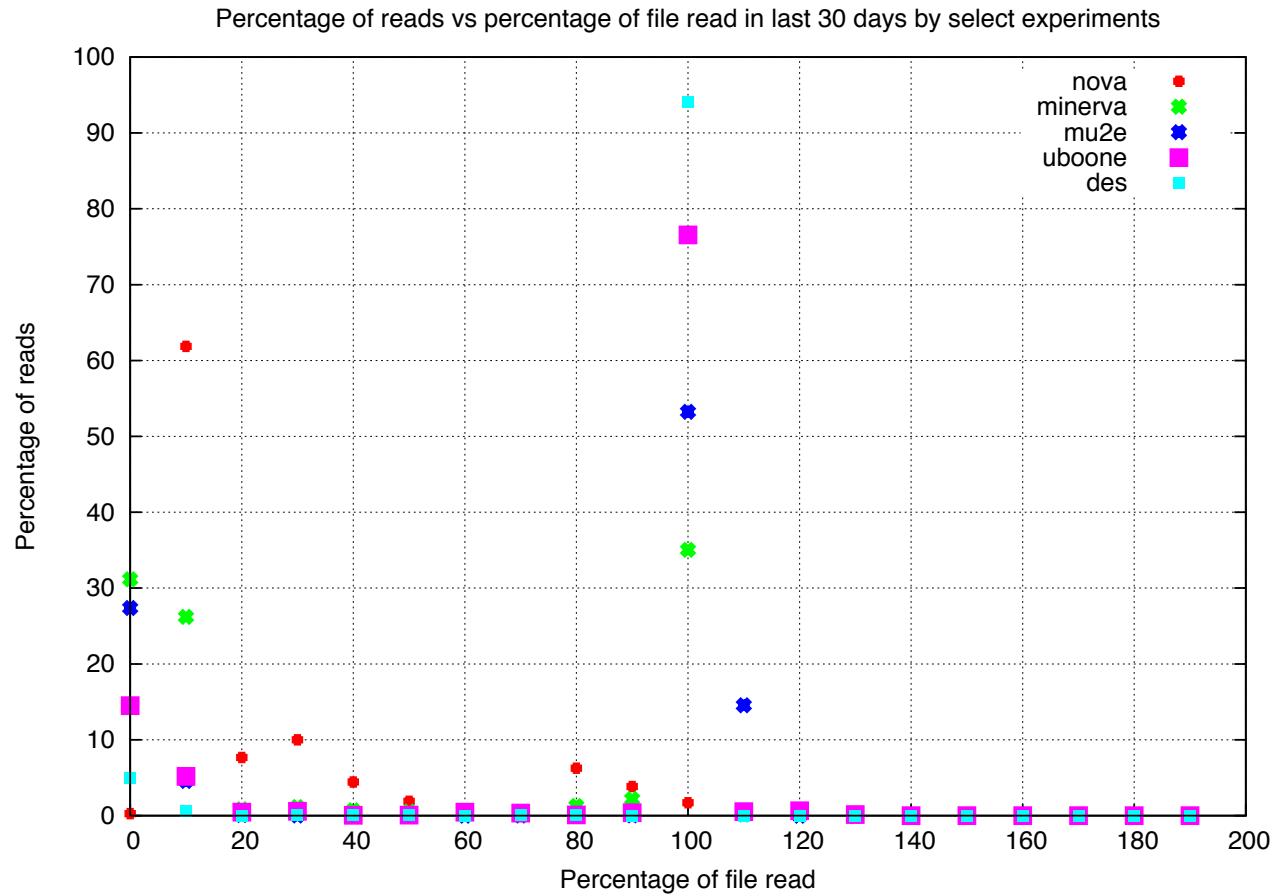
CMS T1 Disk



CMS is mostly streaming, Public dCache is mostly copying?

# Percentage of file read

- Percentage of all reads vs percentage of file read by select experiments using “streaming” protocols like XRootD, NFS, DCap
- Copy would “win” if a file is read more than once (>100% in this plot)



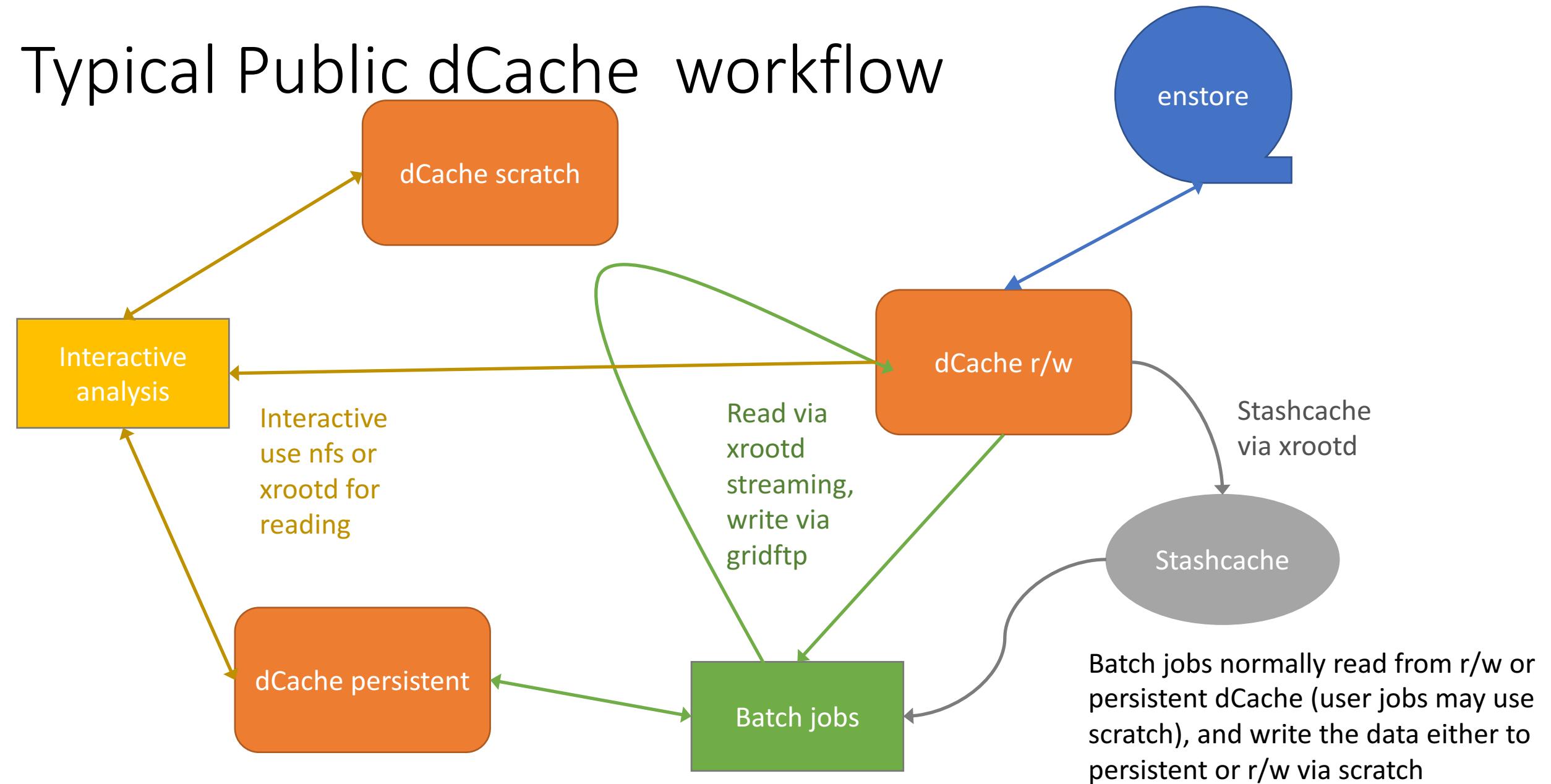
# Copy vs reading over network

- There would be no issue if pool nodes had infinite I/O and infinite bandwidth.
- We have to restrict number of active GFTP movers (transfers) per pool to 20 to avoid I/O subsystem overheating on pool nodes when they are hit by massive waves of fast transfers from CPU farms on Fermilab network or other fast networks.
- On the other hand slow transfers over the WAN occupy active slots for a long time eventually clogging up the system and leading to GFTP request queueing resulting in low job efficiency.
- Separating LAN and WAN traffic by directing them to different pools becomes cumbersome once there are many pool groups of different flavor belonging to different experiments.
- Streaming of data event by event does not stress pool I/O and allows opening of many more active mover slots per pool. E.g. we have 1000 XRootD active movers vs 20 GFTP per pool.

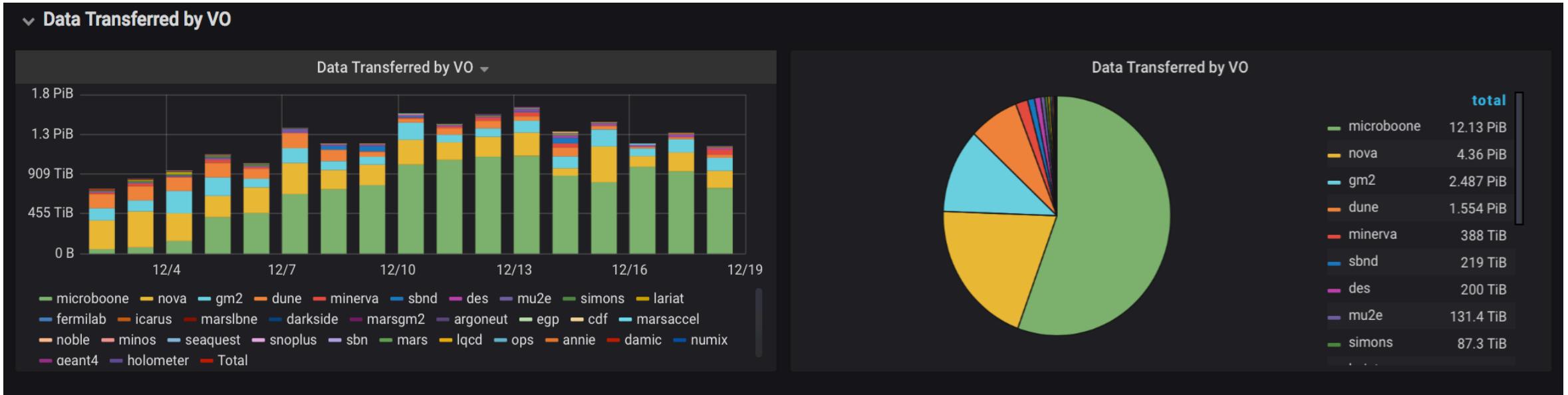
# Access to NEARLINE data

- Running jobs on NEARLINE data degrades job efficiency and creates havoc on tape back-end due to random staging
- Use stage protection by protocol to prevent staging via NFS
- Batch jobs are forced to be run on ONLINE files by data handling tools.

# Typical Public dCache workflow



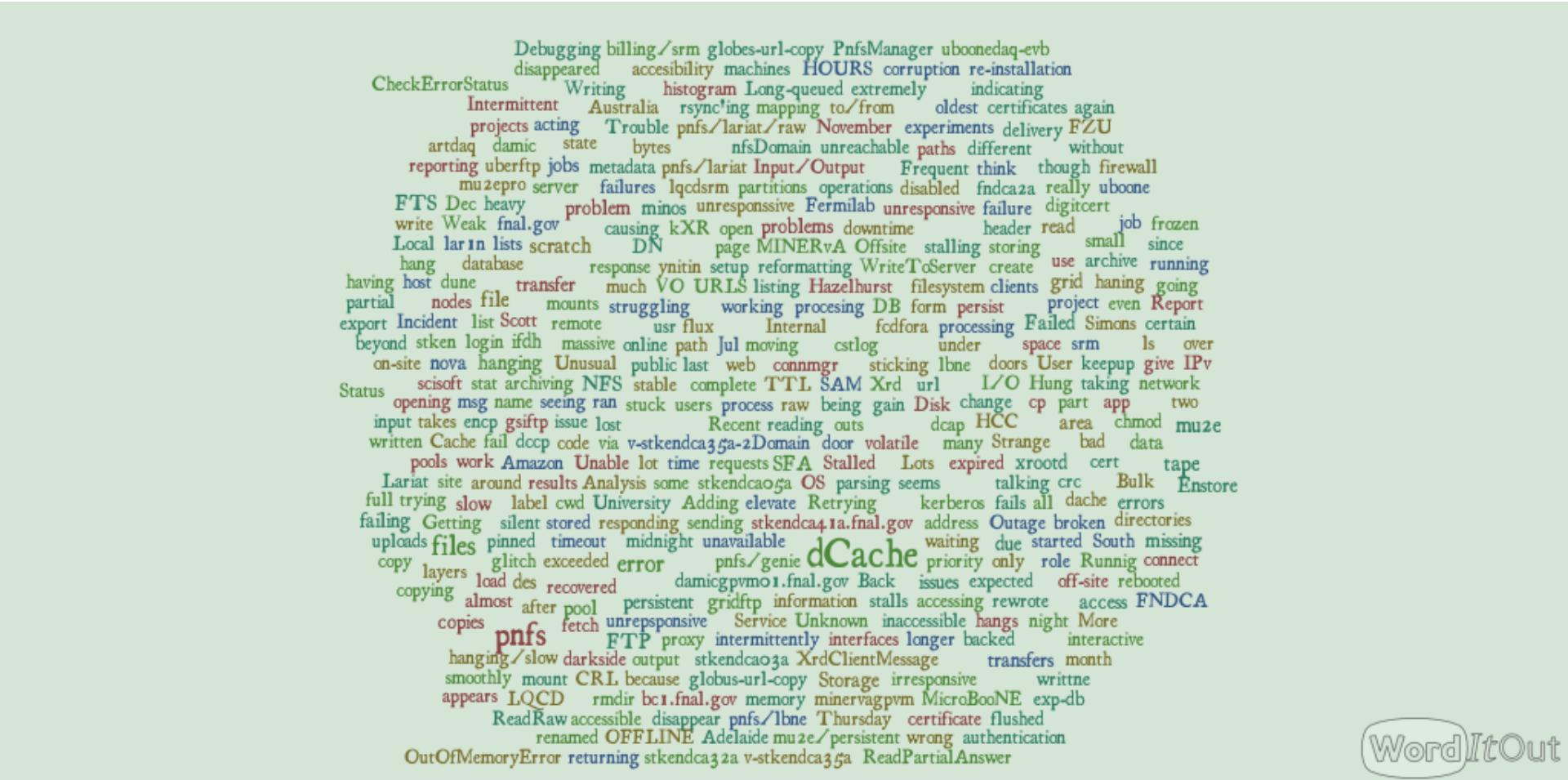
# Public dCache performance



Sustained well over 1PiB/day for > 2 weeks w/o issues.

<https://landscape.fnal.gov/d/000000117/transfer-summary>

# Public dCache issues: about 760 tickets in 2 years



# Problems

- Based on incident ticket statistics Public dCache has by far has more issues and lower user satisfaction compared to CMS dCache.
- This is due to a more chaotic data access in Public dCache.
- Major problem is NFS v4.1 access (which is completely unavailable for users on CMS):
  - tendency to use directory tree as file catalog
  - running interactive analysis on thousands of files from a single client node
  - NFS v4.1 susceptibility to adverse conditions (like network issues, unavailable pools, I/O bound pools)

# NFS problems

- Data corruption
- NFS is mounted on interactive nodes (mostly VM), typical issues:
  - slow / stuck / hung access
    - addressed by killing clients using admin interface
    - or client node power-cycle
- General weirdness:

```
[root@pip2gpvm01 ~]# mount -o vers=3 pnfs-stken:/pip2 /mnt
[root@pip2gpvm01 ~]# ls -al /mnt
total 5
drwxrwxr-x  3 55391 pip-ii  512 Mar 25 11:54 .
dr-xr-xr-x. 28 root    root   4096 May 15 09:26 ..
drwxrwxr-x  3 55391 pip-ii  512 Mar 25 11:57 scratch
[root@pip2gpvm01 ~]# umount /mnt
[root@pip2gpvm01 ~]# mount -o vers=4,minorversion=1 pnfs-stken:/pip2 /mnt
mount.nfs: mounting pnfs-stken:/pip2 failed, reason given by server: No such file or directory
```

# Conclusion

- Large dCache Instances @ Fermilab
  - CMS T1 (tape and disk) are stable and performant
  - Public dCache has experienced growing pains (a factor of 100 in size in 5 years):
    - Running 4.2 for the last 8 months, relatively happy
    - Moved from copying data w/ GFTP to "streaming"
    - Moving from NFS v4.1 to XRootD
    - Exploring TPC with HTTP and XrootD
    - Adoption of more sophisticated workflow frameworks (Rucio) will (hopefully) improve data access efficiency