Preparing Applications for Next Generation IO/Storage

Gary Grider

HPC Division

Los Alamos National Laboratory

LA-UR-13-29428

Abstract

Applications use of IO/storage has had a very stable interface for multiple decades, namely POSIX. Over that period, the shortcomings of POSIX for use in scalable supercomputing apps and systems have become more understood and more pronounced over time. To deal with these shortcomings, the HPC community has created lots of middleware. It is becoming clear that middleware alone can not overcome the POSIX limitations going forward. The Parallel Log Structured File System (PLFS) is a good example of loosening POSIX semantics, where applications promise to not overwrite data in a file from two different processes/ranks. To begin to prepare for for this eventuality that the long loved/hated POSIX API will have to be stepped around or avoided, we need to examine the drivers for this change and how the changes will be manifested. Dealing with Burst Buffers is but one example of how the IO/Storage stack will change. This talk examines the drivers and directions for changes to how applications will address io/storage in the near future.

Preparing Applications for Next Generation IO/Storage

Gary Grider

HPC Division

Los Alamos National Laboratory

LA-UR-13-29428

Drivers for Change

- Scale
  - Machine size (massive failure)
  - Storage size (must face failure)
- Size of data/number of objects
- Bursty behavior
- Technology Trends
- Economics

IO Computer Scientists Have a Small Bag of Tricks

- Hierarchy/buffering/tiers/two phase
- Active/defer/let someone else do it
- Transactions/Epochs/Versioning
- Entanglement/alignment/organization/ structure

- Which leads applications programmers to say:
  - I HATE IO ☺

Hierarchy

- Why:
  - Scale
  - Size of data
  - Bursty behavior
  - Technology Trends
  - Economics
- How: Memory->Burst Buffer->PFS->Campaign Storage->Archive
- Where/When : Trinity 2015 N PB memory, 2-3N PB Burst Buffer, 300-500 PB Campaign Storage

Burst Buffer Economics

System attributes	2010	"2015"		"2018"		2018
System peak	2 Peta	200 Petaflop/sec		1 Exaflop/sec		
Power	6 MW	15 MW		20 MW		29.52
System memory	0.3 PB	5 PB (Looks Probable)		32-64 PB		384.16
Node performance	125 GF	0.5 TF	7 TF	1 TF	10 TF	4.000
Node memory BW	25 GB/s	0.1 TB/sec	1 TB/sec	0.4 TB/sec	4 TB/sec	250.000
Node concurrency	12	O(100)	O(1,000)	O(1,000)	O(10,000)	1000.000
System size (nodes)	18,700	50,000 (Probably more like 15,000-40,000)	5,000	1,000,000	100,000	0.024
Total Node Interconnect BW	1.5 GB/s	20 GB/sec		200 GB/sec		48.000
MTTI	days	O(1day) (Maybe)		O(1 day) (Doubt it)		0.012

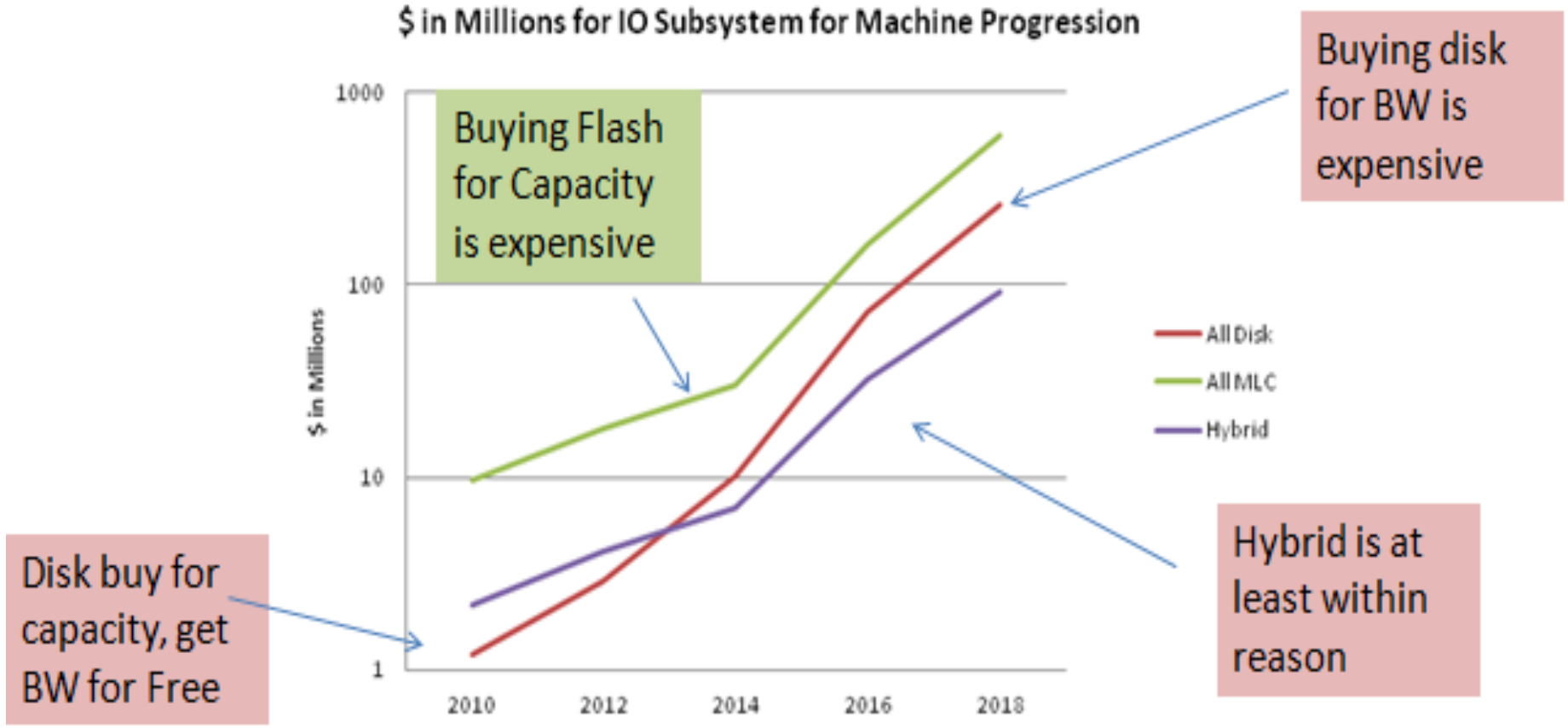
Year						
DISC						
Disk Capacity TB						29.52
Disk Speed MB/s						384.16
Disk Power Watts						4.000
$/device						250.000
Tot $/device	4					1000.000
FLASH						
MLC Capacity TB						0.024
MLC BW MB/s						48.000
SLC Capacity TB						0.012
SLC BW MB/s						192.000
Flash Power Watts						1.000
$/device mlc						4.00
Tot $/device mlc	4					16.000
machine mem pb		0.080	0.300	1.000	8.000	30.000
time to dump sec		1000.000	750.000	500.000	400.000	300.000
Required Rate GB		80.000	400.000	2000.000	20000.000	100000.000
Required Capacity pb	30	2.400	9.000	30.000	240.000	900.000
disk only						
num devices capacity						30492.163
num devices BW						1000000.000
Total Cost M						1,000.000
MLC Flash only						
num devices capacity		600000.000	1125000.000	1875000.000	10000000.000	37500000.000
num devices BW		10000.000	25000.000	62500.000	416666.667	2083333.333
Total Cost M		9.600	18.000	30.000	160.000	600.000
Hybrid MLC/Disk		assume 3X memory for Flash and 30X memory for disk				
$ MLC		160,000.000	400,000.000	1,000,000.000	6,666,666.667	33,333,333.333
$ Disk		1,200,000.000	2,295,918.367	3,904,623.074	15,937,237.036	30,492,162.696
Tot $ M		1.360	2.696	4.905	22.604	63.825

MTTI will likely go down

At Exascale, have to meet two requirements
100 TB/Sec Burst from ~billion processing elements
(30PB burst for 5 minutes every 1 hour)
1 EB of Scratch Capacity (30ish memories)

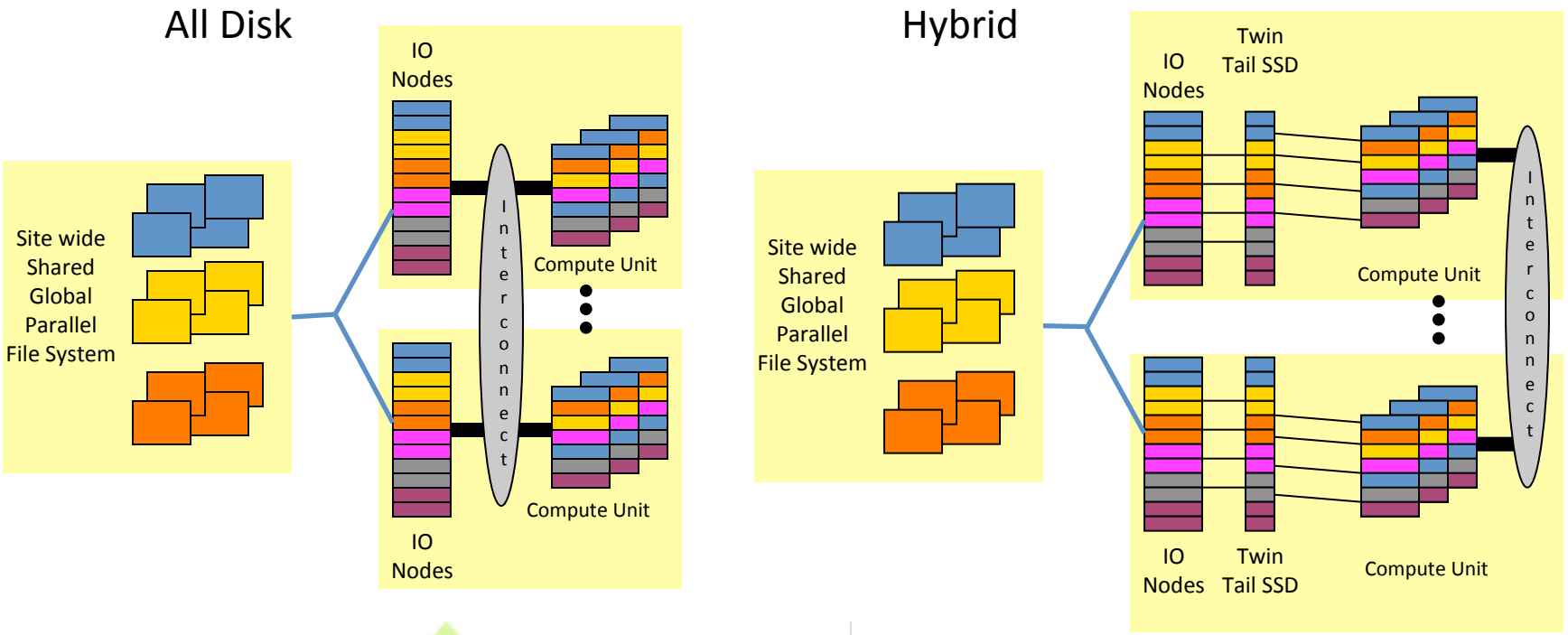
Background: Economic Analysis Results (circa 2009)

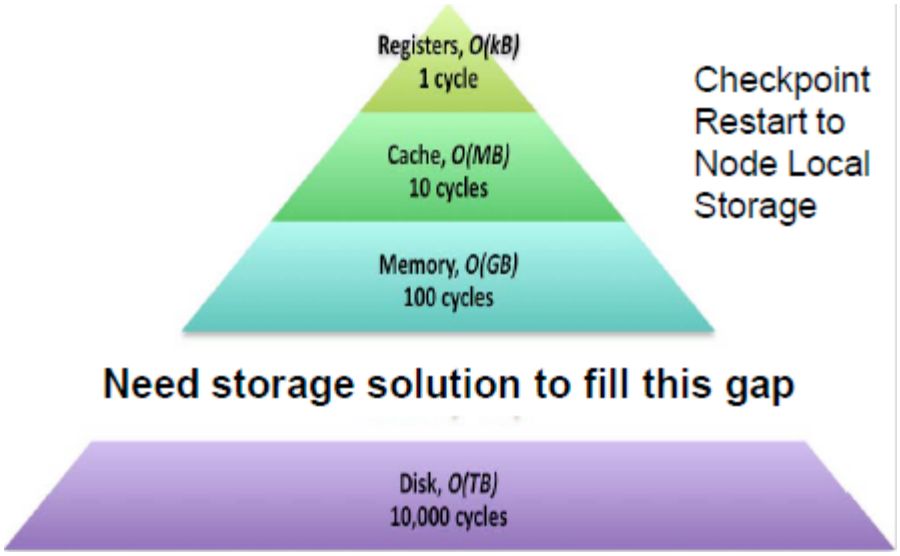
Must Meet Two Requirements: **1 EB Capacity and 100 TB/sec**



Take away: $M in Log Scale means you have hit the big time!

Hybrid Disk (Capacity)/SSD (Bandwidth)





Target Trinity system will have
-2-4 PB of DRAM
-5-12 PB of Flash/burst buffer (4-10 TB/sec)
-100 ish PB of disk (1-3 TB/sec)

Coming to a store near you!

EMC²

ABBA

DataDirect™ NETWORKS

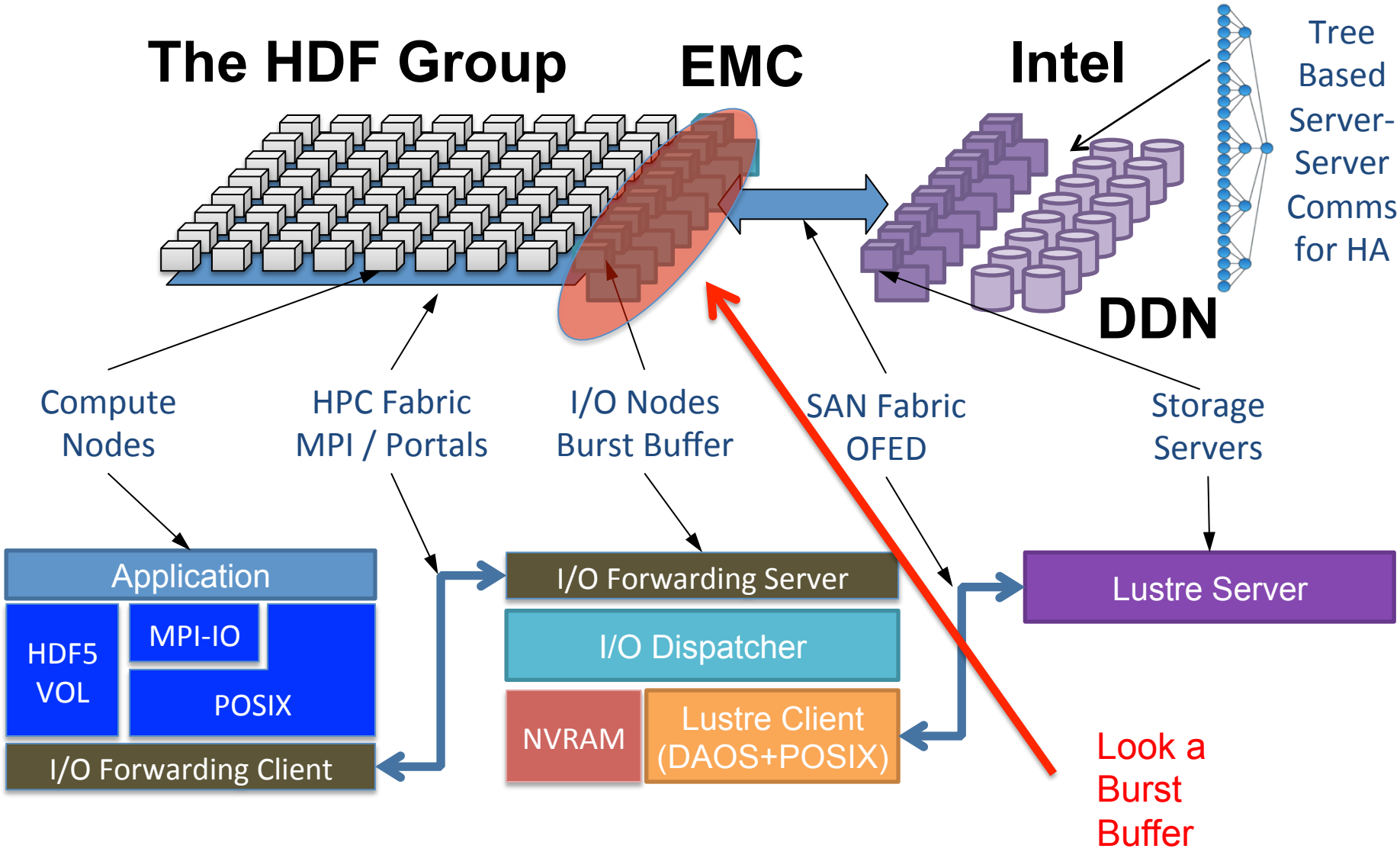
INFINITE MEMORY ENGINE™

NetApp

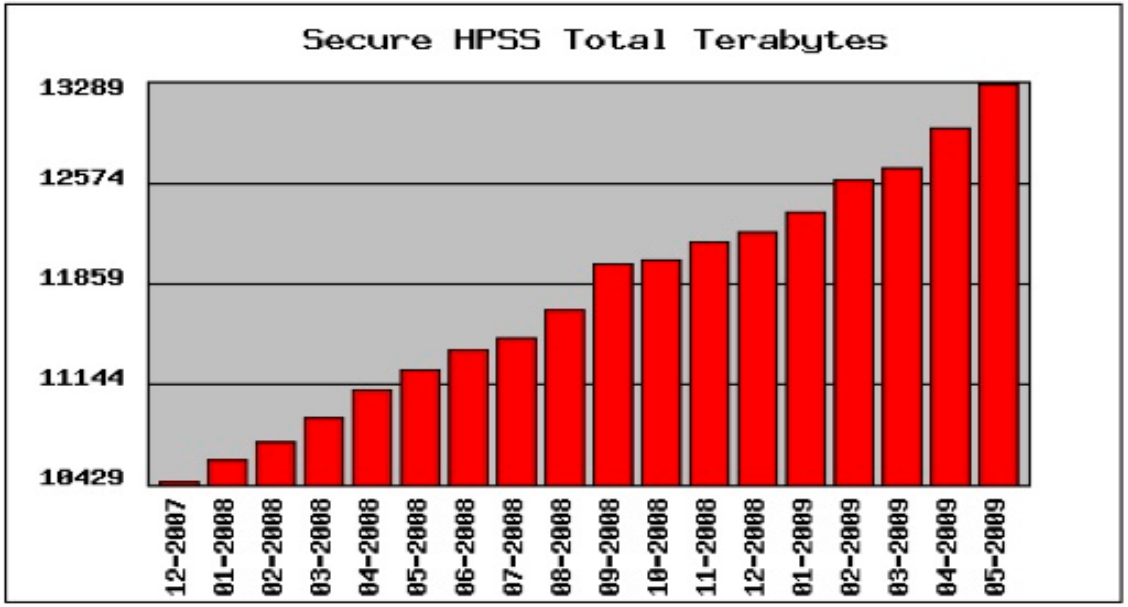
The Burst Buffer

SC12 most over used word Co-Design
SC13 most over used word Burst

Fast Forward I/O Architecture

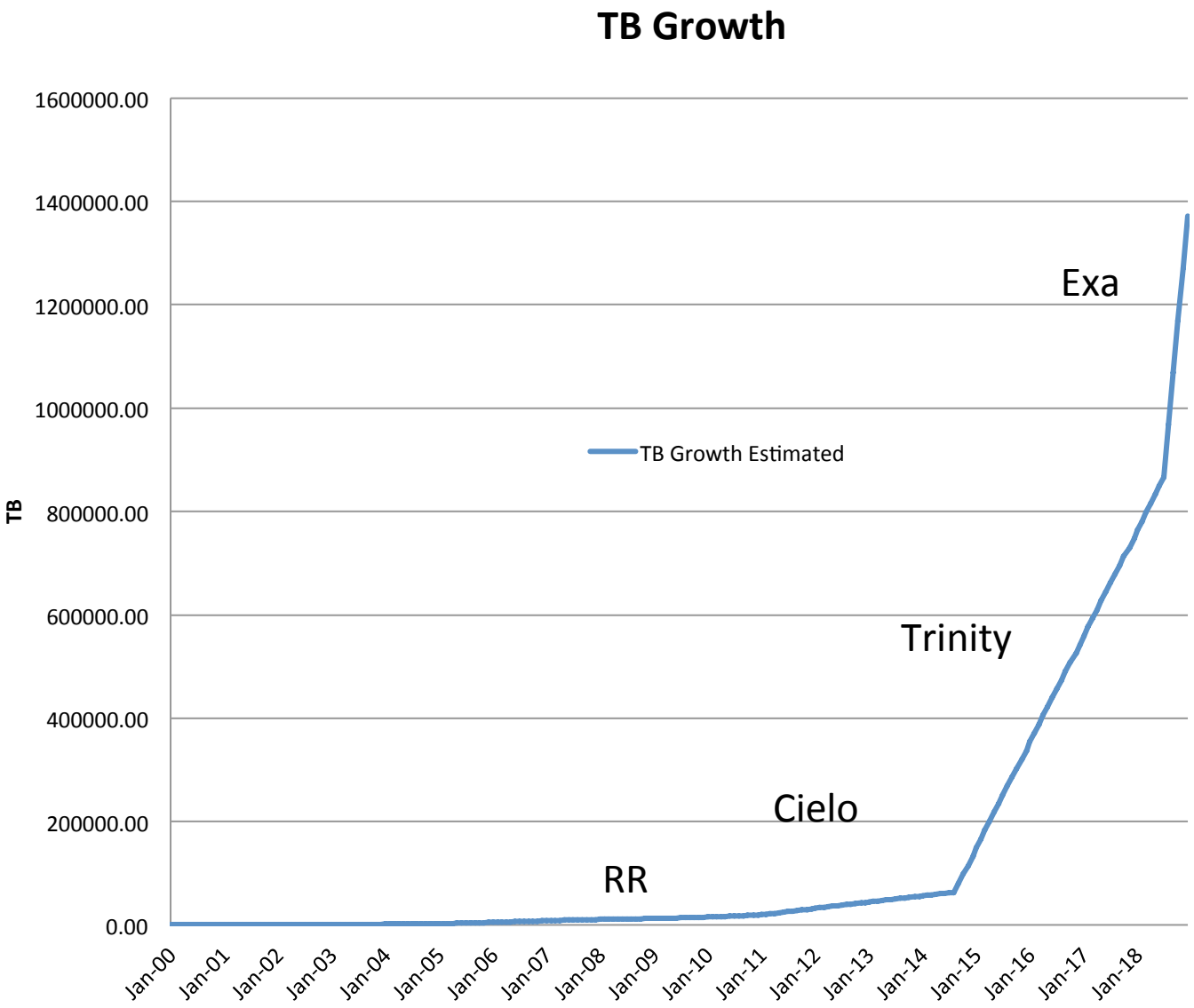


Archive Economics



- Unlimited archive will become cost prohibitive
- Past method of using bandwidth to archive as rate limiter may not be adequate going forward

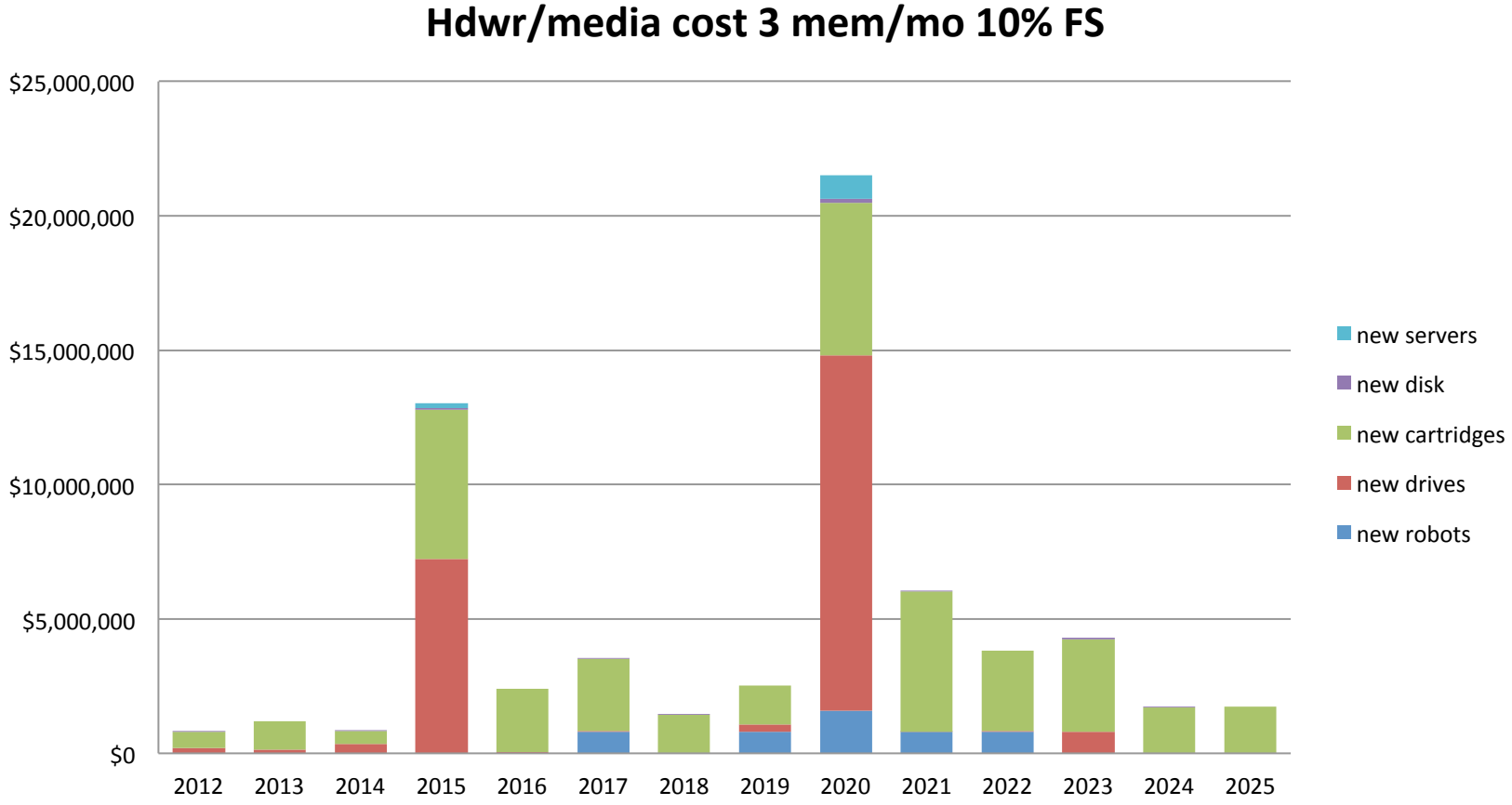
Archive Growth TB



Notice the slope changes when the memory on the floor changes (avg 3 memory dumps/ month and it has been pretty constant for a decade

Model of Hardware/Media Costs 3 Memories/Month

The spikes are cause by tape drives, moving these huge memory objects (5-50PB) to tape in a reasonable time is costly, tape drives are expensive



Capacity is no longer the sole cost driver for Archive as it has been in the last 25 years
Bandwidth is now contributing in a major way to TCO of Archives

What do we do about this Archive Problem?

- All disk Pre-Archive (Campaign Storage)
  Contemplate 100k-1 Million disks
  - Power management, with spin up 10,000 drives for parallel transfer
  - Vibration management if you want to use inexpensive drives running 10,000 in parallel
  - Mean Time To Data Loss needs to rival tape at a reasonable cost and disk array rebuild is probably not tolerable.
- Similar to ShutterFly/DropBox in usage patterns but we have to eat Petabyte single images, their max image size is a 5 orders of magnitude smaller
- Direction may be to leverage cloud server side provider technologies like:
  - Erasure
  - Power management
  - Object Store mechanisms

Take away from Hierarchy Trickery

Ex-IO researchers turned managers use spreadsheets to do their bidding!

Active

- Why:
  - Scale
  - Size of data
  - Technology Trends
  - Economics

- How: Ship function to data – limit data movement

- When:
  - Trinity 2015 N PB memory, 2-3N PB Burst Buffer, opportunity for in-transit analysis
  - Campaign Storage 2015 – data online longer, make better decisions about what to keep
  - Analysis shipping in commercial IO stack, post DOE Fast Forward/ productization phase
  - BGAS/Seagate Kinetic drives, Burst Buffer Products, etc.

Transactions

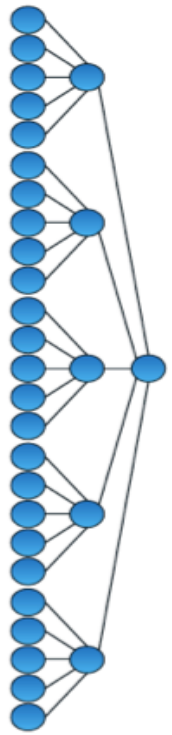
- Why:
  - Scale
  - Size of data
  - Economics

  DARE TO ALLOW THE STORAGE SYSTEM TO FAIL!

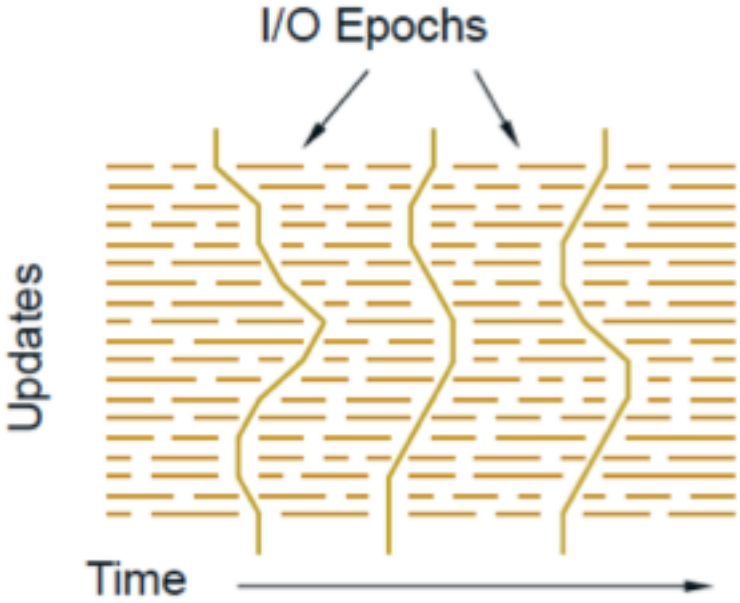
- How: Transactional/Epoch/Versioning

- When:
  - Post DOE Fast Forward/productization phase

Server Communications, Collectives Reliability, Transaction/Versions



- Storage Fast Forward is adding check-summing capability from end to end starting at compute node memory through stack and back
- Storage Fast Forward doing Server Side Comms
  - Out of band low latency quorum/health
  - Collective client eviction/client health monitoring
  - Collective attribute and other intelligent caching
  - Discovery, Gosip, Paxos, etc.



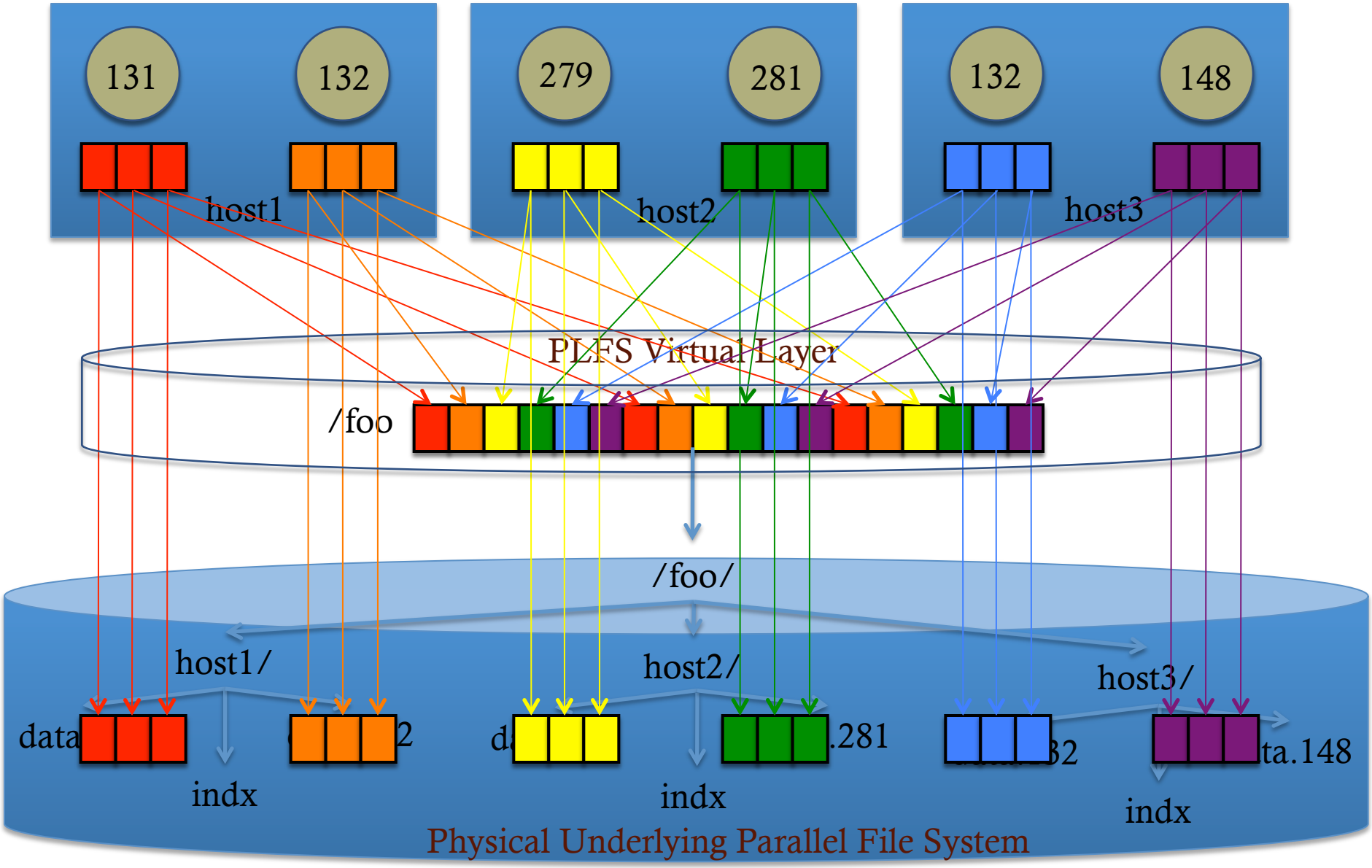
- Guarantees required on any and all failures
- Required at all levels (IOD, DAOS) for both metadata and data
- Non blocking protocols
- Way to demark globally consistent snapshots, recovery, roll back,
- User/app driven consistency

Entanglement

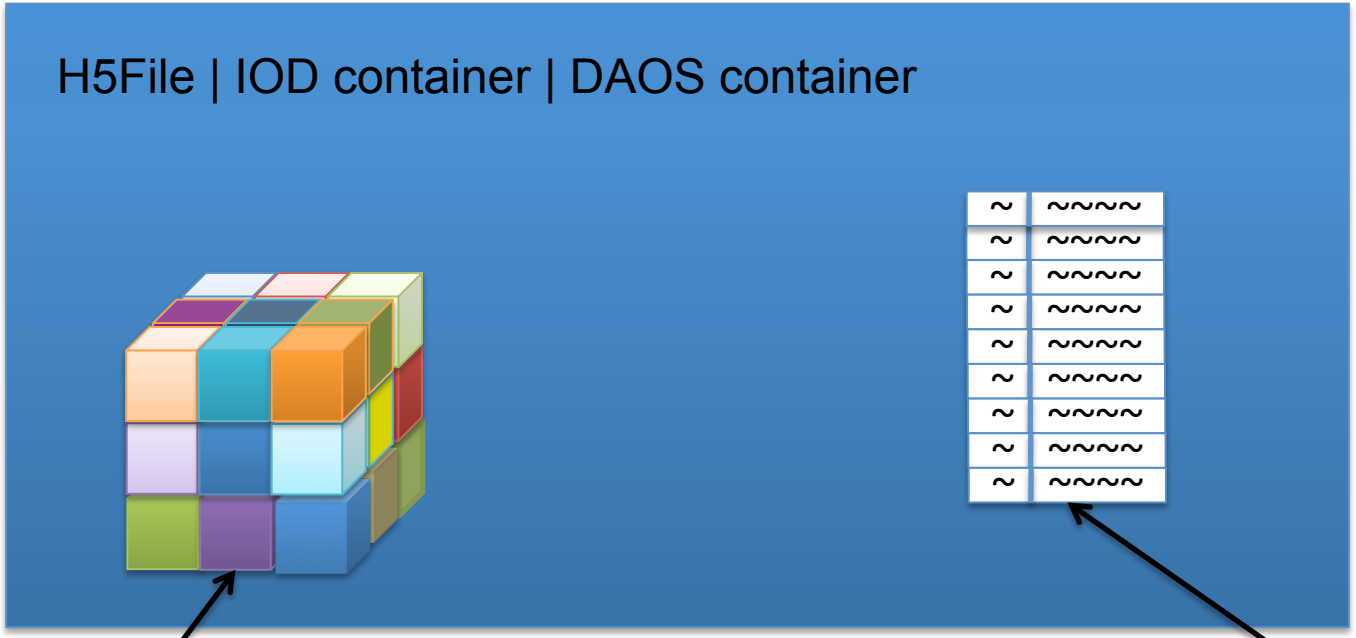
- Why:
  - Scale
  - Bursty Behavior
  - Size of data
  - Economics

- How: Pass Structure down IO Stack/PLFS like untangling, etc.
- When:
  - PLFS and other similar middleware solutions
  - Post DOE Fast Forward/productization phase

Decouples Logical from Physical



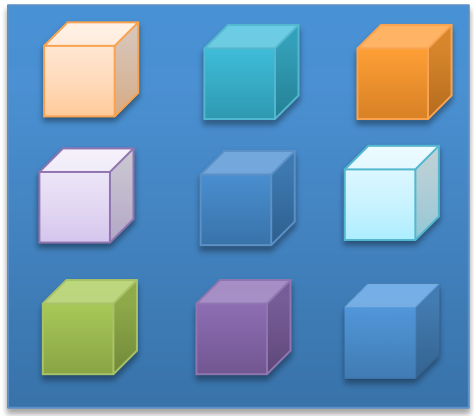
H5File
(somewhat IOD-centric perspective)

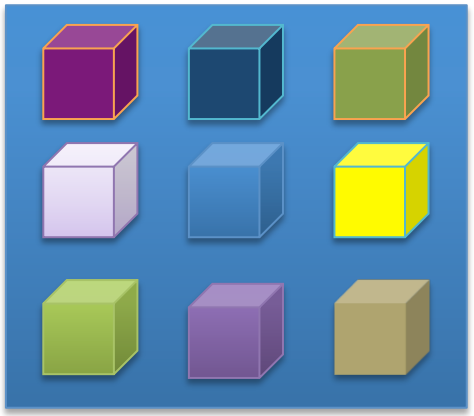


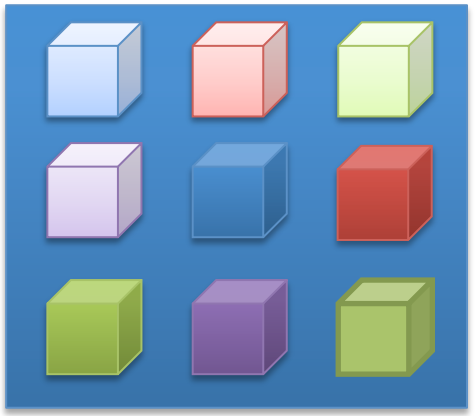
HDF Dataset
IOD Array Object

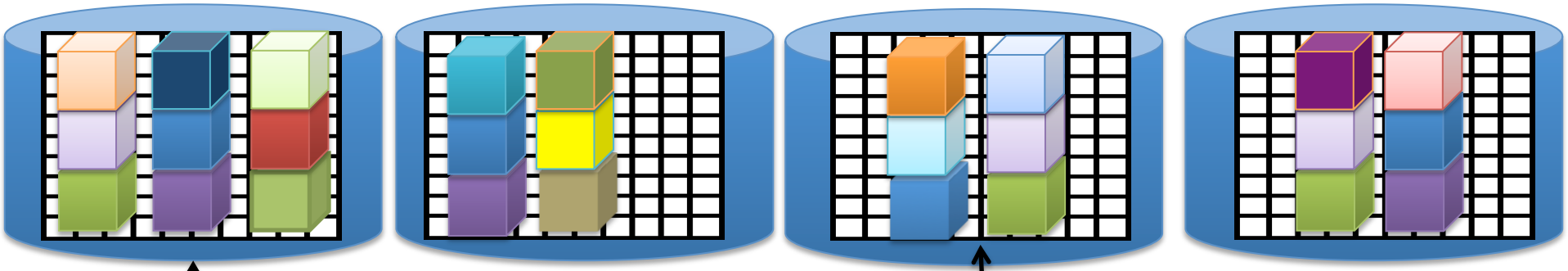
IOD Array Object on ION's, stream per writer, migrate semantically to DAOS

ION's







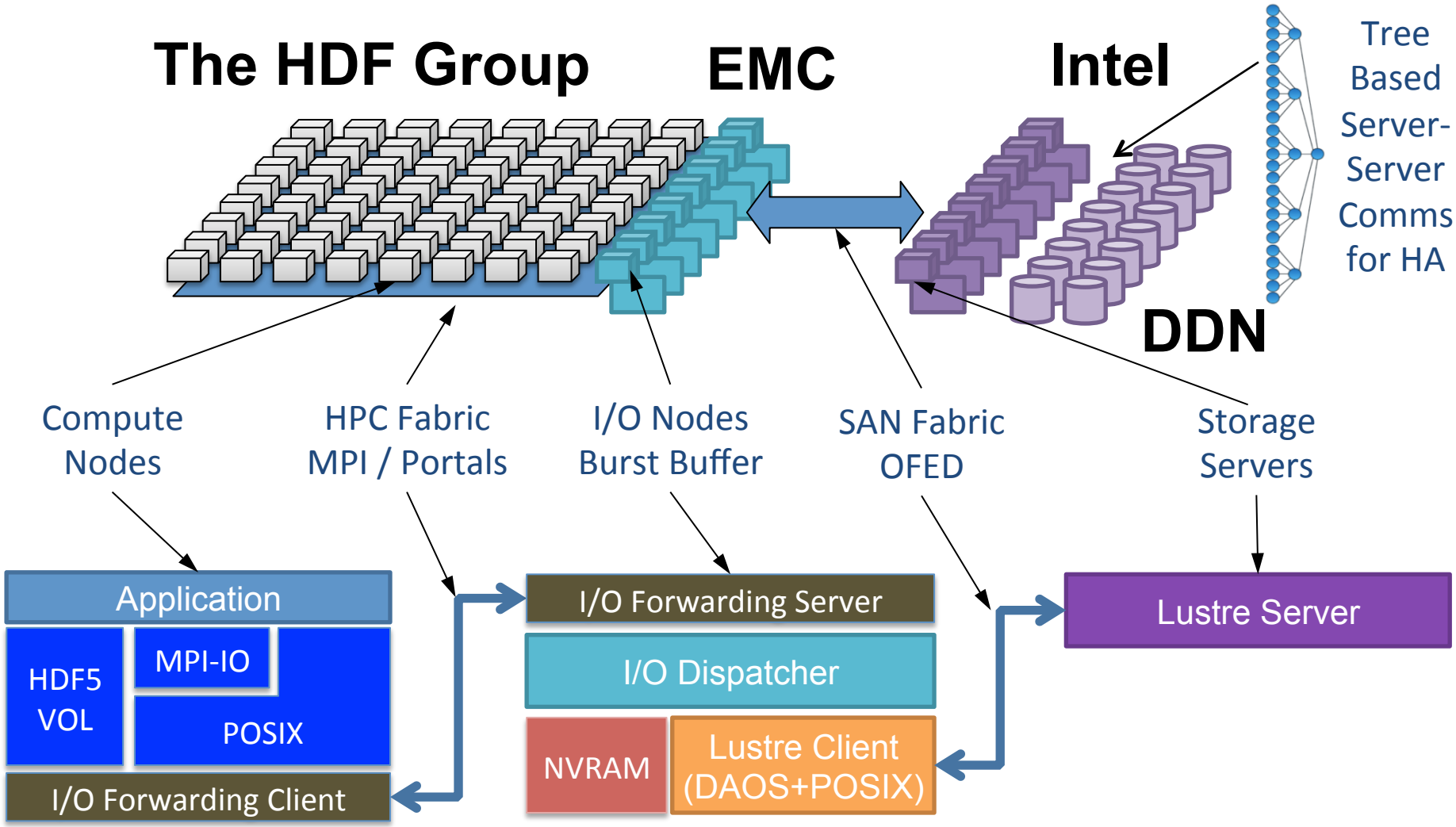


DAOS Storage Target

DAOS Shard

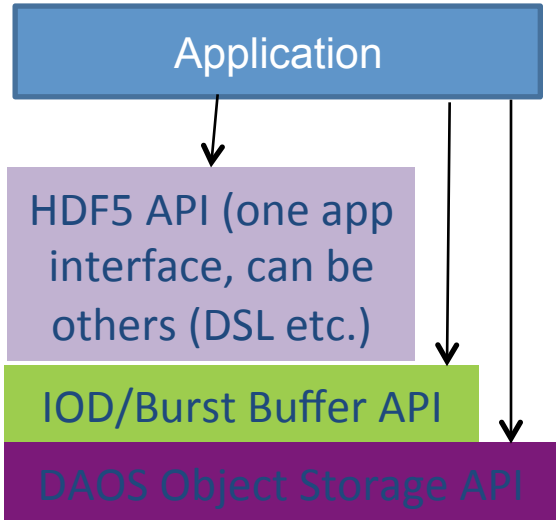
And now for the bad news, how will your apps change

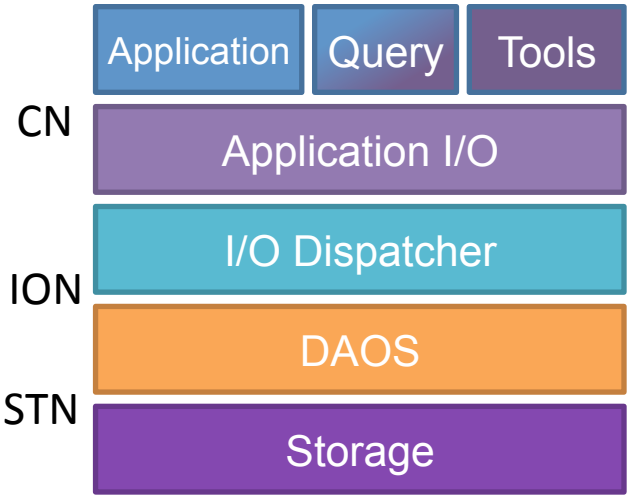
Fast Forward I/O Architecture



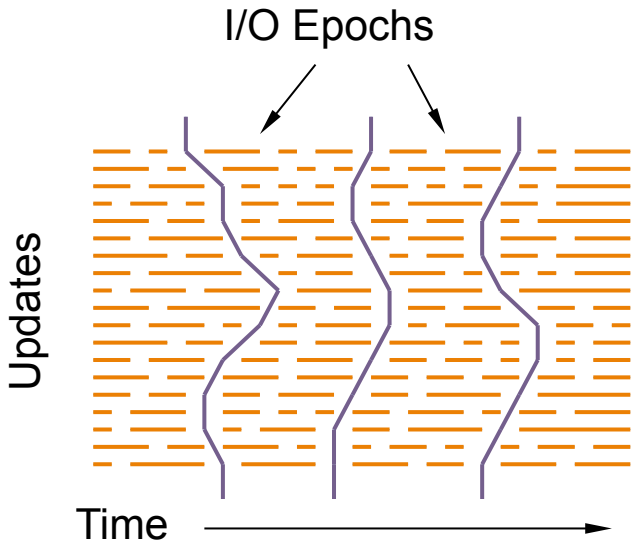
I/O stacks

- Application I/O
  - Multiple top-level APIs to support general purpose or application-specific I/O models
- I/O Dispatcher
  - Match conflicting application and storage object models
  - Manage NVRAM burst buffer / cache
- DAOS
  - Scalable, transactional global shared object storage





- Features & requirements
  - Non-blocking APIs
    - Asynchronous programming models
  - Transactional == consistent thru failure
    - End-to-end application data & metadata integrity
  - Low latency / OS bypass
    - Fragmented / Irregular data
  - Function/Analysis Shipping to enable In-Transit Analysis (second year)

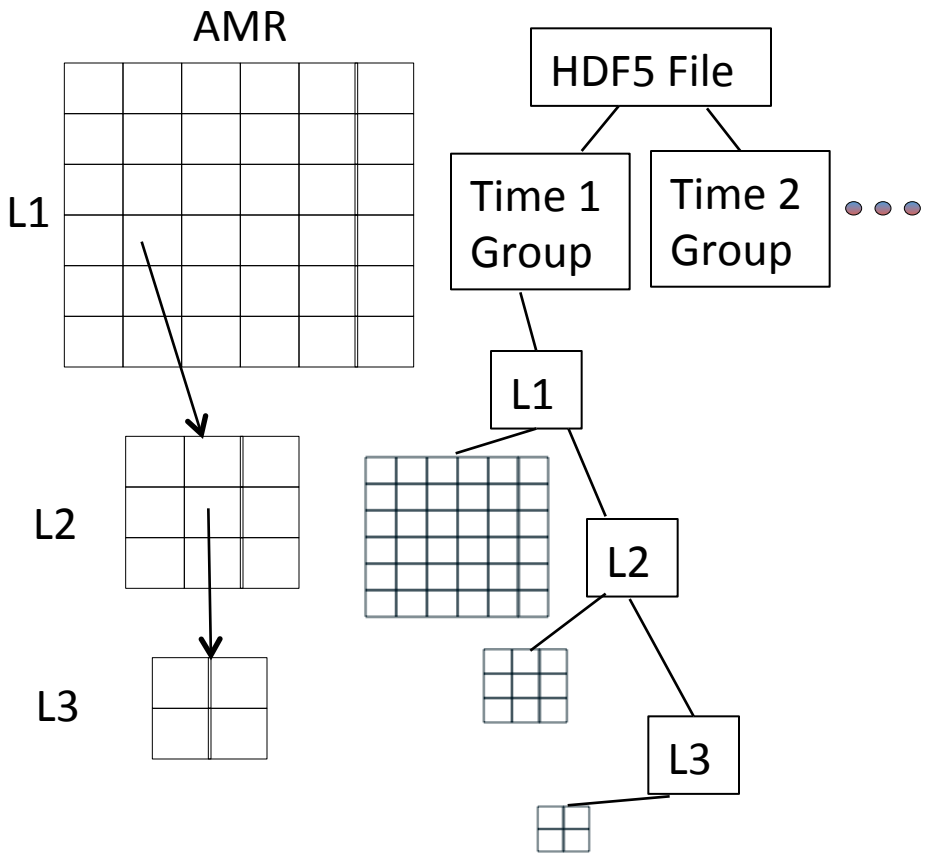


A New Storage Abstraction

- Support POSIX for quite a while – but exploitation requires changes
- Objects instead of files
  - Array objects , Blobs, and Key-values
- Containers instead of directories
  - Snapshots for efficient COW across sets of objects (with provenance)
  - Transactions for atomic operations across sets of objects
- List IO and Async
- Explicit Burst Buffer management exposed to app or system
  - Migrate, purge, pre-stage, multi-format replicas, semantic resharding
- End-to-end data integrity
  - Checksums stored with data, app can detect silent data corruption
- Co-processing analysis on in-transit data
  - Query and reorganize the placement of data structures before analysis shipping

– Work flow will need to take into consideration expense of going to disk and especially to archive

HDF5 (the current example of a high level API to this new IO stack)



- H5TRBegin-a (trans1, req1)
- H5Create-file-a (… trans, req2))
- H5Creaet-group-a(… trans, req3)
- H5TRCommit-a (trans1)
- Go do other work
- H5TRCheck/wait (trans1) or HTReqCheck/ wait(reqN)
- H5TRBegin-a(trans2,req4)
- H5Dwrite (object/array,trans2,rew5) (write all you want)
- H5TRCommit (trans2)
- Go do other work,
- You can even start a new transaction to do metadata or data ops with trans3++ and overlap as much IO and computation, including abort.
- You cant be sure anything made it to storage until H5TRCheck/wait say that transaction is secure.
- You can control structure, async behavior, rollback, etc.

IOD API

- OBJECT FUNCTIONS
  - iod_obj_create, create_list, open, close
  - iod_array_write, array_write_list, blob_write, blob_write_list
  - iod_obj_open_read read_list,
  - Iod_obj_blob_read, blob_read_list, array_read, array_readlist
  - iod_array_extend, query
  - iod_obj_set_layout and get_layout
  - iod_obj_unlink and unlink_listiod_obj_unlink_list
  - iod_obj_set_scratch, get_scratch
- DATA CONSISTENCY
  - iod_trans_query, trans_start, trans_slip, trans_finish, trans_persist, trans purge,
  - trans_fetch, trans_replica, trans_snapshot
- KEY VALUE FUNCTIONS
  - iod_kv_set, set_list, get_num, get_list, list_key, get_value, unlink_keys
- EVENT FUNCTIONS
  - iod_eq_create, destroy, poll, abort, query, init, finish

DAOS

- Event Queue Mgmt
- daos_eq_create, destroy, poll, query, init, finish, next, abort
-
- DAOS Storage Layout Mgmt
- daos_sys_open **cage/rack/node/target**, close, listen, query
-
- Container Structure Mgmt
- daos_container_open, unlink, snapshot, query, listen
-
- Collective operation APIs
- daos_local2global and global2local
-
- Shard API
- daos_shard_add, disable, query, flush
-
- Object API
- daos_object_open, close, read, write, flush, punch
-
- Epoch & Epoch functions
- daos_epoch_scope_close, catchup, commit

Hopefully You Are Confused Now

- Remember, applications programmers say:
  - I HATE IO ☺