SDC 18

September 24-27, 2018
Santa Clara, CA

www.storagedeveloper.org

Storage Lessons from HPC: A Multi-Decadal Struggle

Gary Grider
HPC Division Leader, LANL/US DOE

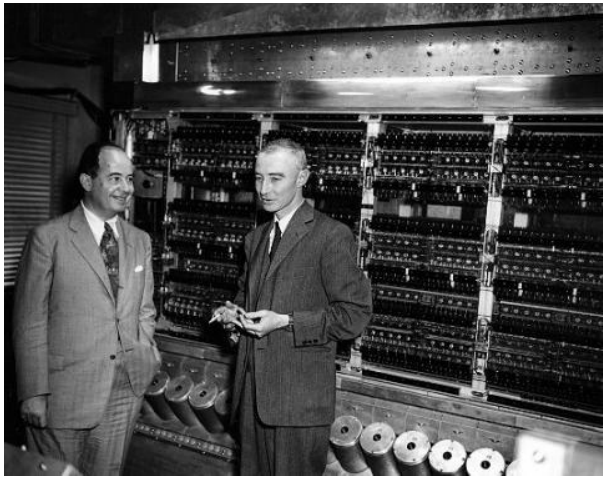
2018

LA-UR-18-24612

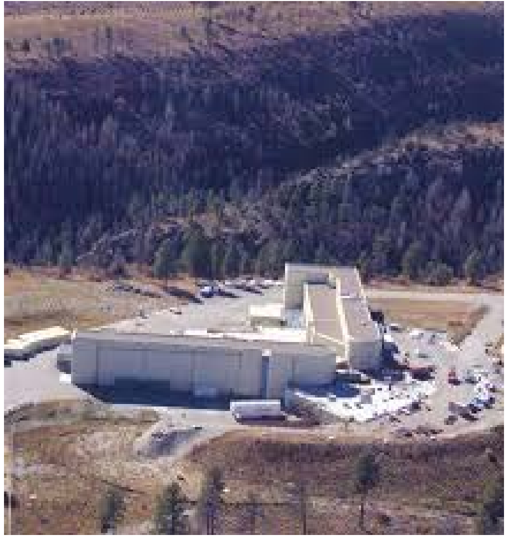
Los Alamos
NATIONAL LABORATORY
EST.1943

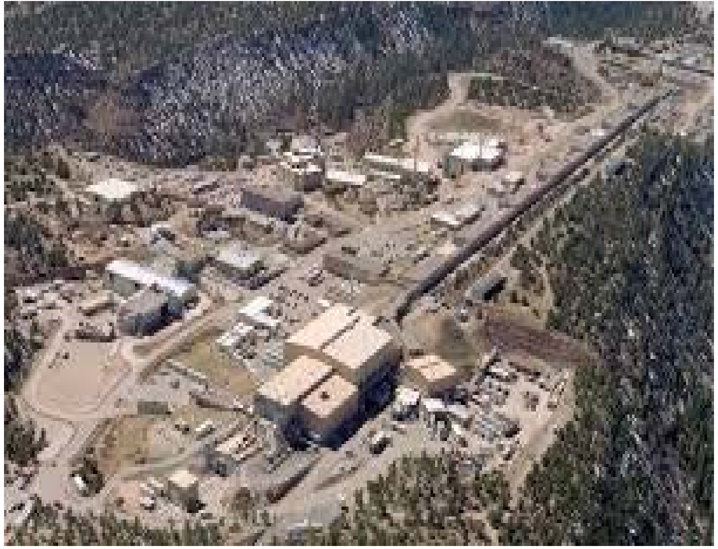
NNSA

What is Los Alamos



LOS ALAMOS PROJECT MAIN GATE
PASSES MUST BE PRESENTED TO GUARDS
POST No. 1





Eight Decades of Production Weapons Computing to Keep the Nation Safe

Maniac

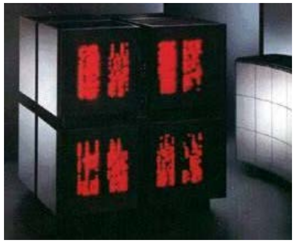
IBM Stretch

CDC

Cray 1

Cray X/Y

CM-2



CM-5

SGI Blue Mountain

DEC/HP Q

IBM Cell Roadrunner

Cray XE Cielo

Cray Intel KNL Trinity

Ising DWave

Cross Roads

CROSSROADS
An APEX Collaboration

Some Storage Products You May Not Realize Were Funded/Heavily Influenced by DOE/LANL

panasas

lustre

HDF

CRAY THE SUPERCOMPUTER COMPANY

Data Warp

IBM GPFS

UniTree software

pNFS

DataTree CFS

HPSS High Performance Storage System

ceph

DDN STORAGE

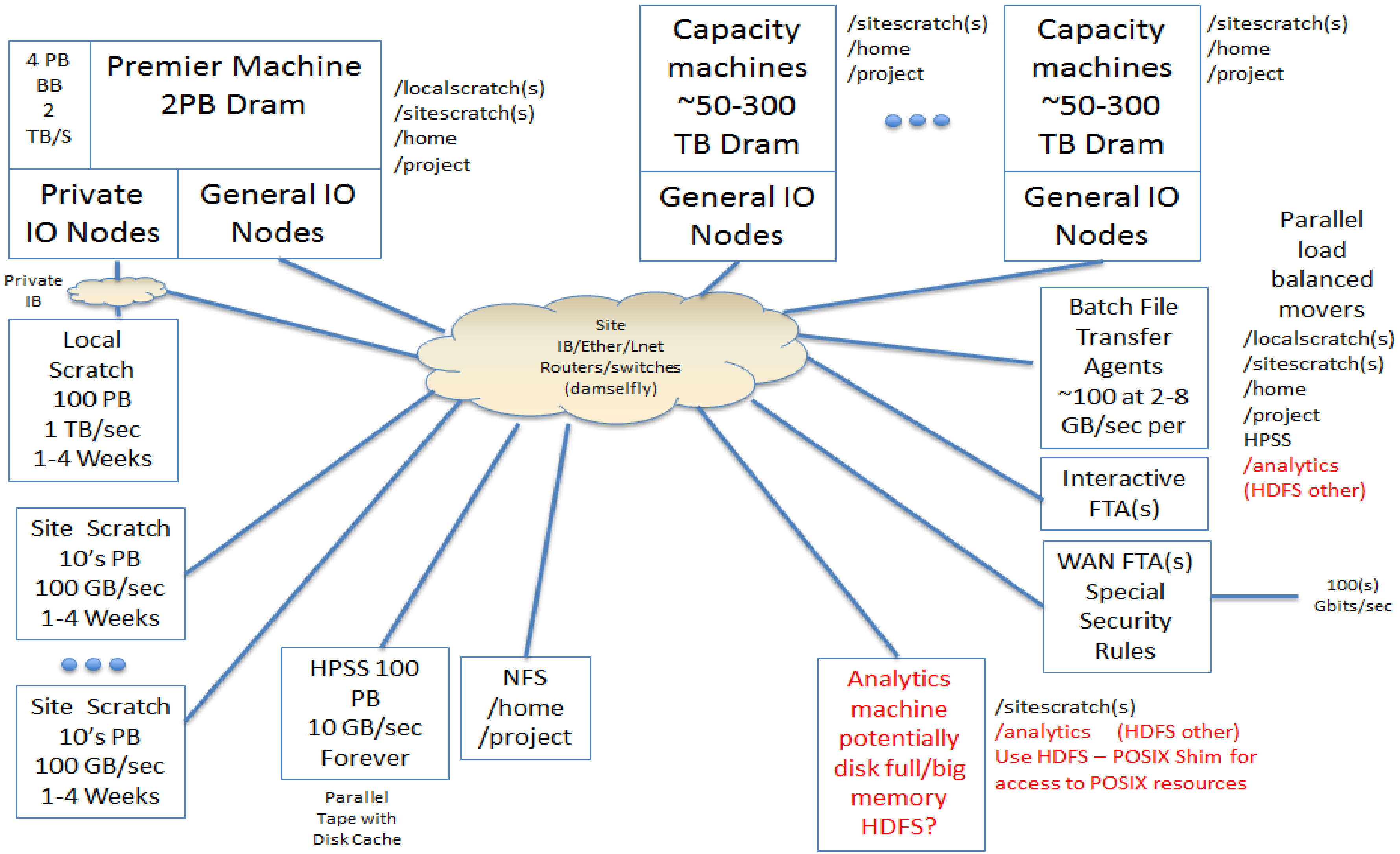
INFINITE MEMORY ENGINE

IBM Photo-store

Tokutek

Extreme HPC Background

Simple View of our Computing Environment


4 PB BB 2 TB/S
Premier Machine 2PB Dram
/localscratch(s)
/sitescratch(s)
/home
/project
Private IO Nodes
General IO Nodes
Private IB
Local Scratch 100 PB 1 TB/sec 1-4 Weeks
Capacity machines ~50-300 TB Dram
General IO Nodes
/sitescratch(s)
/home
/project
Capacity machines ~50-300 TB Dram
General IO Nodes
/sitescratch(s)
/home
/project
Site IB/Ether/Lnet Routers/switches (damselfly)
Parallel load balanced movers
/localscratch(s)
/sitescratch(s)
/home
/project
HPSS
/analytics
(HDFS other)
Batch File Transfer Agents ~100 at 2-8 GB/sec per
Interactive FTA(s)
WAN FTA(s) Special Security Rules
100(s) Gbits/sec
Site Scratch 10's PB 100 GB/sec 1-4 Weeks
Site Scratch 10's PB 100 GB/sec 1-4 Weeks
HPSS 100 PB 10 GB/sec Forever
Parallel Tape with Disk Cache
NFS /home /project
Analytics machine potentially disk full/big memory HDFS?
/sitescratch(s)
/analytics (HDFS other)
Use HDFS – POSIX Shim for access to POSIX resources

Current Largest Machine Trinity (circa 2016)

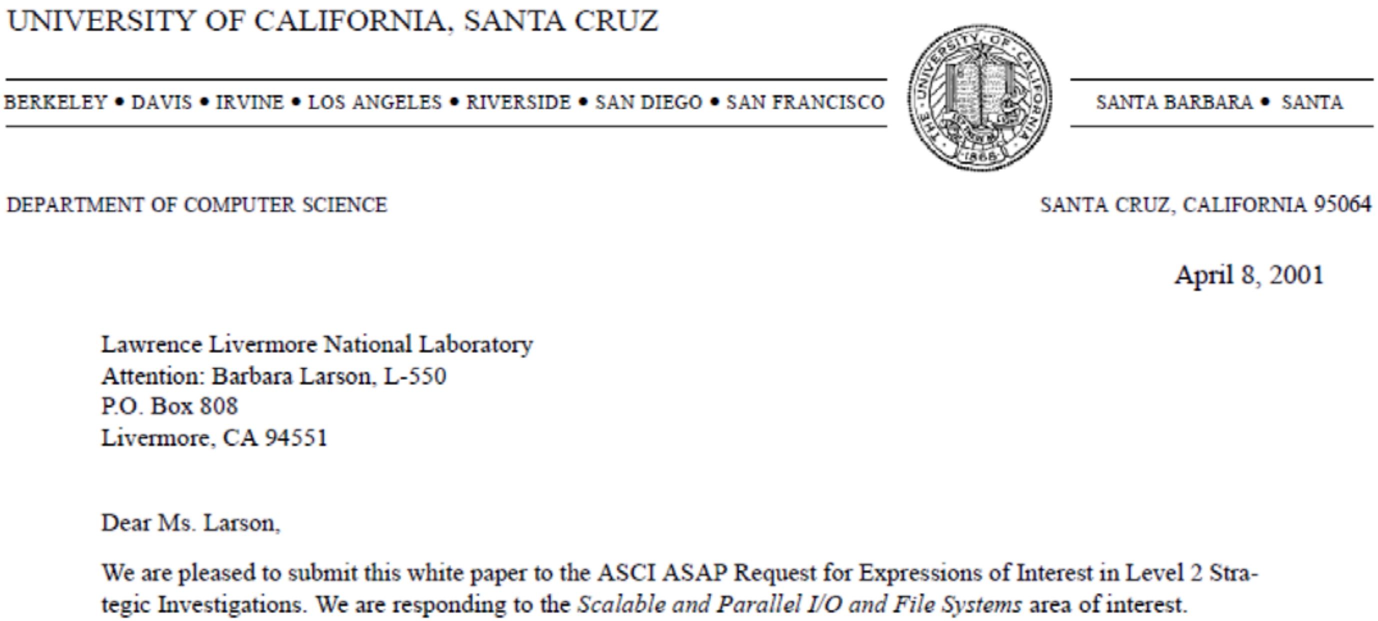
- Haswell and KNL
- 20,000 Nodes
- Few Million Cores
- 2 PByte DRAM
- 4 PByte NAND Burst Buffer ~ 4 Tbyte/sec
- 100 Pbyte Scratch PMR Disk File system ~1.2 Tbyte/sec
- 30PByte/year Sitewide SMR Disk Campaign Store ~ 1 Gbyte/sec/Pbyte (30 Gbyte/sec currently)
- 60 PByte Sitewide Parallel Tape Archive ~ 3 Gbyte/sec

Begin the battle for parallel data storage.

- **Oh, you want to build thousands of nodes clusters and more than one, what about parallel storage? (Circa 1995)**
- Well, gosh we will need something we might call a global parallel file system
  - Beg for money 1995-1999
  - Have the intergalactic global parallel file system workshop 1999
  - Help get Panasas Started
  - Help get PIOFS -> GPFS
  - Write the contract to U Michigan CITI to start NFSv4 and pNFS work
  - Write the contract that started Ceph at UCSC
  - Issue contract for Lustre 2001
  - 2001 invent the scalable parallel backbone
  - 2009 runner up best SC paper for Parallel Log Structured File Systems

Ceph’s humble start – DOE ASCI Univ Alliances

Planned to be a user space object FS for Scalable Metadata and Security Research



UNIVERSITY OF CALIFORNIA, SANTA CRUZ

BERKELEY • DAVIS • IRVINE • LOS ANGELES • RIVERSIDE • SAN DIEGO • SAN FRANCISCO SANTA BARBARA • SANTA

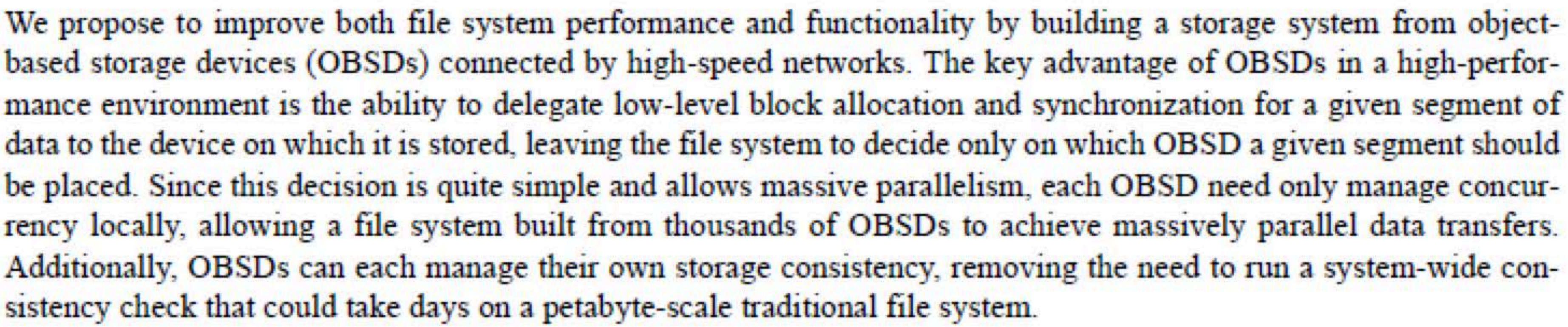
DEPARTMENT OF COMPUTER SCIENCE SANTA CRUZ, CALIFORNIA 95064

April 8, 2001

Lawrence Livermore National Laboratory
Attention: Barbara Larson, L-550
P.O. Box 808
Livermore, CA 94551

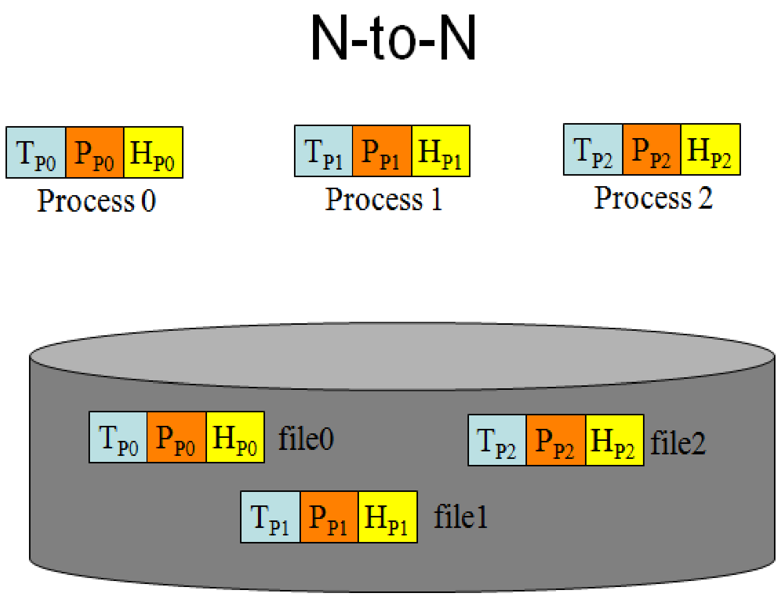
Dear Ms. Larson,

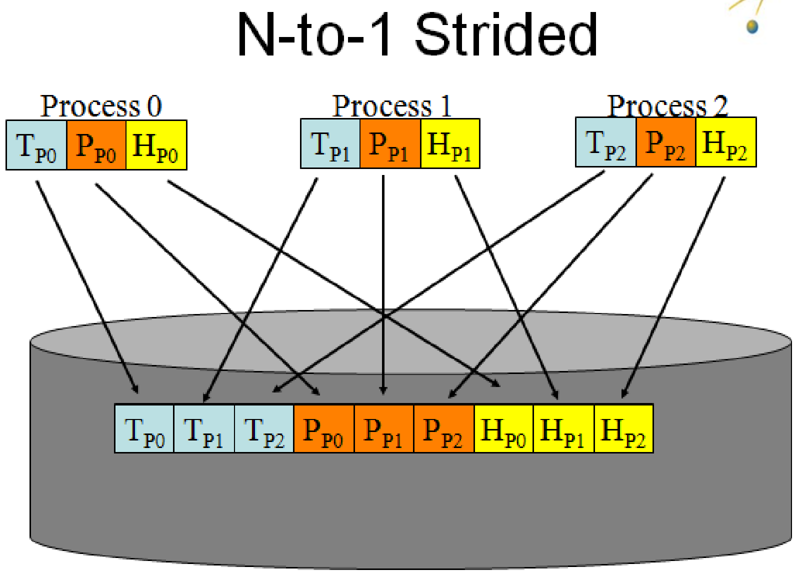
We are pleased to submit this white paper to the ASCI ASAP Request for Expressions of Interest in Level 2 Strategic Investigations. We are responding to the *Scalable and Parallel I/O and File Systems* area of interest.



We propose to improve both file system performance and functionality by building a storage system from object-based storage devices (OBSDs) connected by high-speed networks. The key advantage of OBSDs in a high-performance environment is the ability to delegate low-level block allocation and synchronization for a given segment of data to the device on which it is stored, leaving the file system to decide only on which OBSD a given segment should be placed. Since this decision is quite simple and allows massive parallelism, each OBSD need only manage concurrency locally, allowing a file system built from thousands of OBSDs to achieve massively parallel data transfers. Additionally, OBSDs can each manage their own storage consistency, removing the need to run a system-wide consistency check that could take days on a petabyte-scale traditional file system.

2 HPC IO Patterns still persist as important


N-to-N
T_{P0} P_{P0} H_{P0}
Process 0
T_{P1} P_{P1} H_{P1}
Process 1
T_{P2} P_{P2} H_{P2}
Process 2
T_{P0} P_{P0} H_{P0} file0
T_{P2} P_{P2} H_{P2} file2
T_{P1} P_{P1} H_{P1} file1


N-to-1 Strided
Process 0
T_{P0} P_{P0} H_{P0}
Process 1
T_{P1} P_{P1} H_{P1}
Process 2
T_{P2} P_{P2} H_{P2}
T_{P0} T_{P1} T_{P2} P_{P0} P_{P1} P_{P2} H_{P0} H_{P1} H_{P2}

- ❒ Million files inserted into a single directory at the same time
- ❒ Millions of writers into the same file at the same time
- ❒ Jobs from 1 core to N-Million cores
- ❒ Files from 0 bytes to N-Pbytes
- ❒ Workflows from hours to a year (yes a year on a million cores using a PB DRAM)

Workflow Taxonomy from APEX Procurement

A Simulation Pipeline

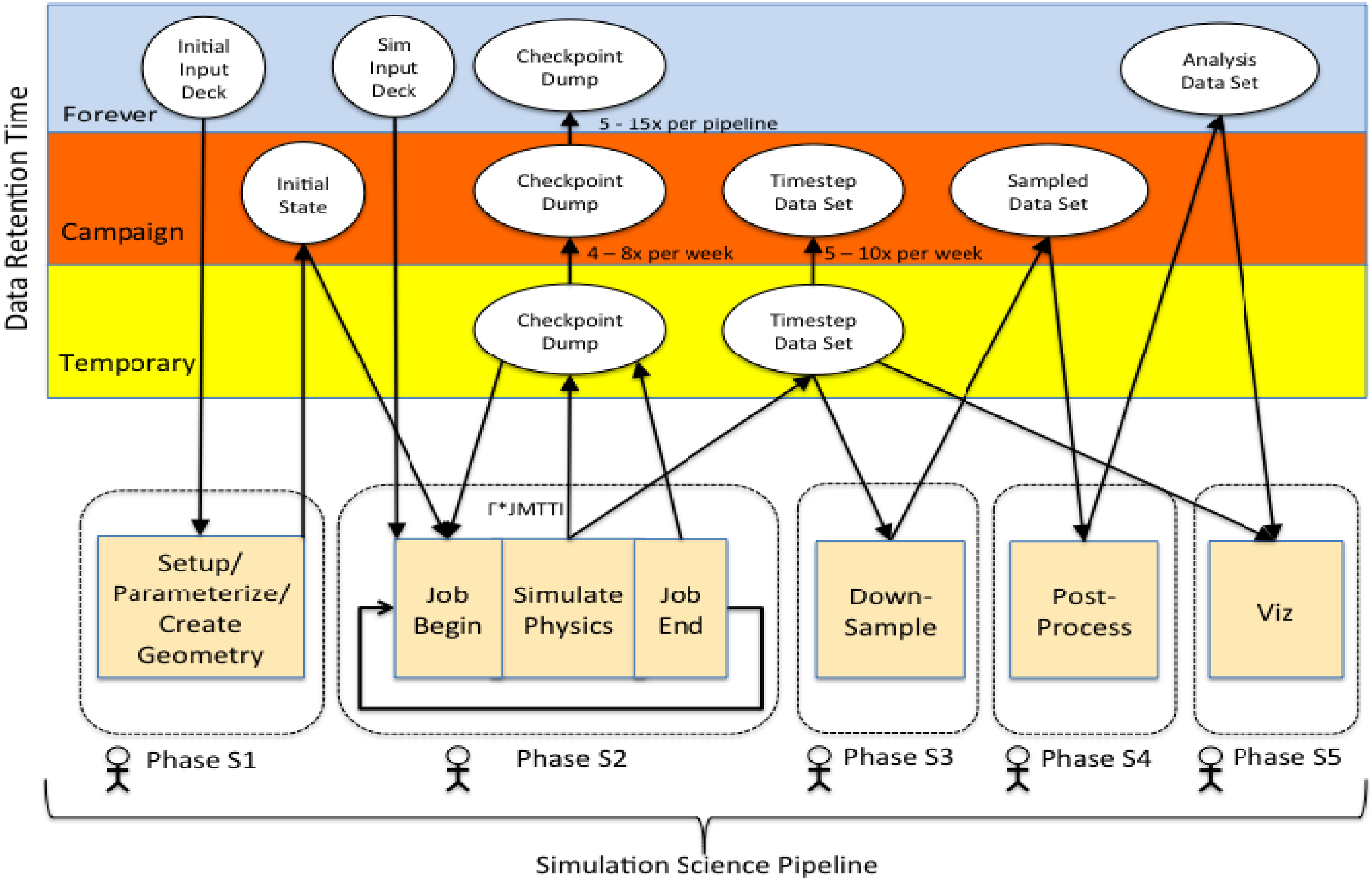


Figure 1: An example of an APEX simulation science workflow.

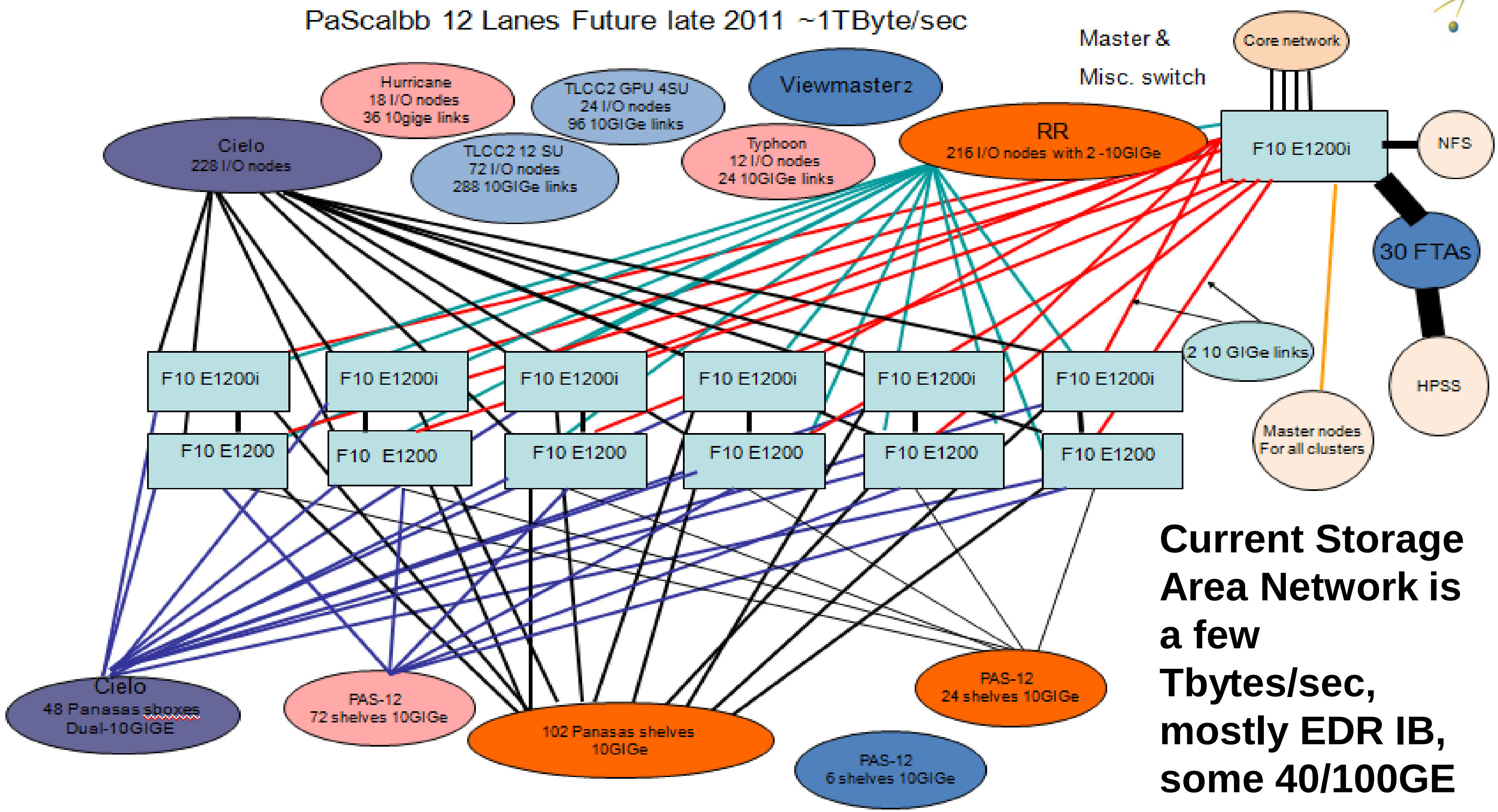
APEX Workflows

LANL, NERSC, SNL

SAND2015-10342 O
LA-UR-15-29113

Making Parallel File Systems Global

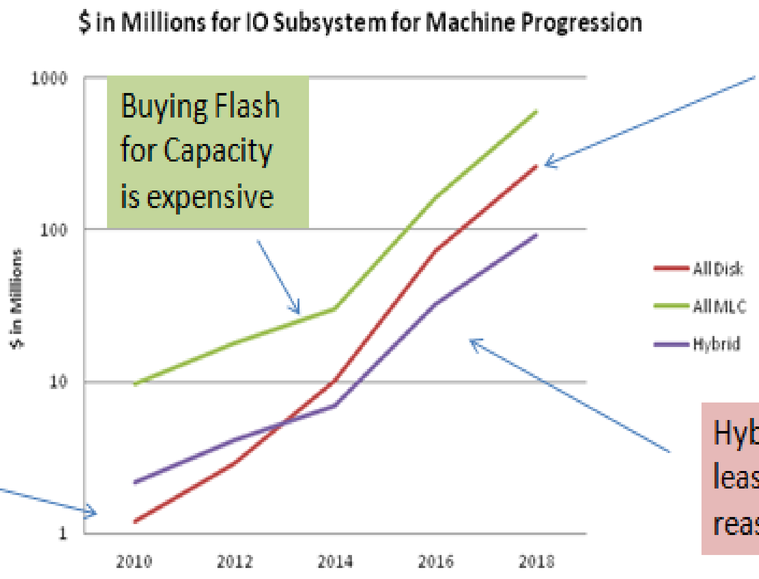
HPC Storage Area Network Circa 2011 Today ~ 2-8 TB/sec



Current Storage Area Network is a few Tbytes/sec, mostly EDR IB, some 40/100GE

Enough with the HPC background
How about some Recent Storage Economics

Economics have shaped our world
The beginning of storage layer proliferation 2009



Buying Flash for Capacity is expensive

Buying disk for BW is expensive

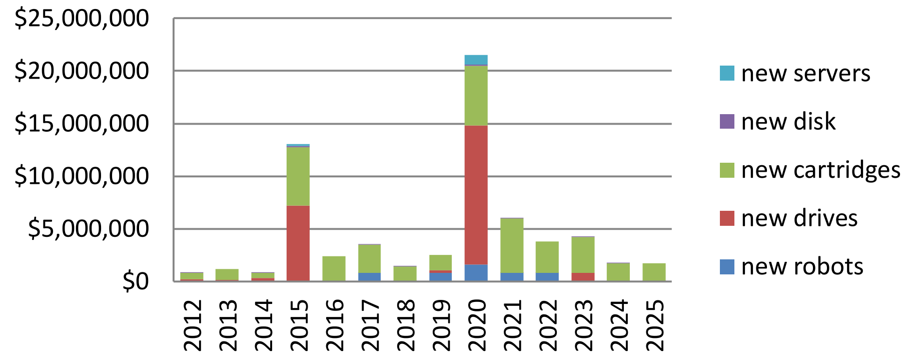
Disk buy for capacity, get BW for Free

Hybrid is at least within reason

- Economic modeling for large burst of data from memory shows bandwidth / capacity better matched for solid state storage near the compute nodes

- Economic modeling for archive shows bandwidth / capacity better matched for disk

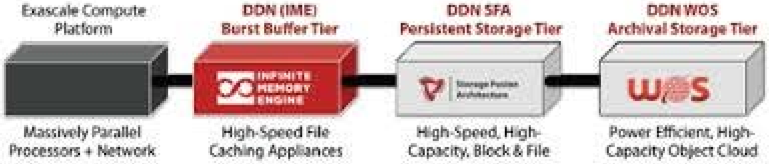
Hdwr/media cost 3 mem/mo 10% FS



The Burst Buffer Hoopla Parade circa 2014

DDN STORAGE

INFINITE MEMORY ENGINE™

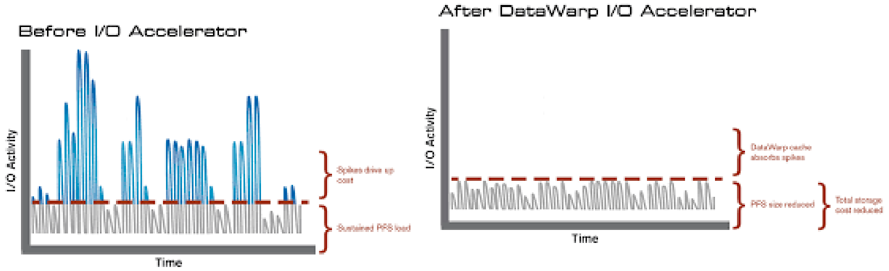

Exascale Compute Platform
DDN (IME) Burst Buffer Tier
DDN SFA Persistent Storage Tier
DDN WOS Archival Storage Tier
INFINITE MEMORY ENGINE
Storage Fusion Architecture
WOS
Massively Parallel Processors + Network
High-Speed File Caching Appliances
High-Speed, High-Capacity, Block & File
Power Efficient, High-Capacity Object Cloud

CRAY

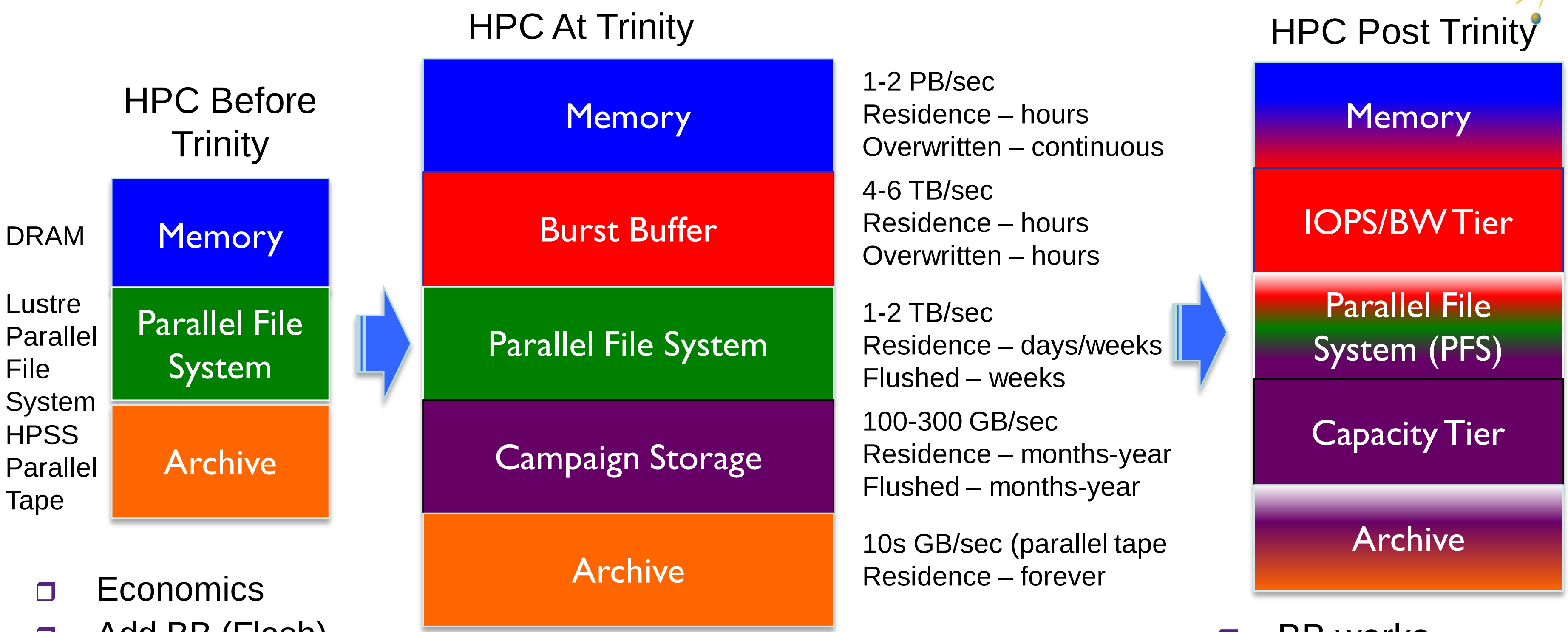
THE SUPERCOMPUTER COMPANY

Data Warp

Powered By
Dilithium Crystals



What are all these storage layers? Why do we need them?



HPC Before Trinity

DRAM	Memory
Lustre Parallel File System	Parallel File System
HPSS Parallel Tape	Archive

HPC At Trinity

Memory	1-2 PB/sec Residence – hours Overwritten – continuous
Burst Buffer	4-6 TB/sec Residence – hours Overwritten – hours
Parallel File System	1-2 TB/sec Residence – days/weeks Flushed – weeks
Campaign Storage	100-300 GB/sec Residence – months-year Flushed – months-year
Archive	10s GB/sec (parallel tape Residence – forever

HPC Post Trinity

- Memory
- IOPS/BW Tier
- Parallel File System (PFS)
- Capacity Tier
- Archive

- ❐ Economics
- ❐ Add BB (Flash)
- ❐ Add Campaign (low cost disk, slightly lower function than PFS)

- ❐ Happy users
- ❐ Running larger scale than ever for longer than ever at any national lab

- ❐ BB works
- ❐ Campaign Works
- ❐ Lower Flash cost
- ❐ Disk harder to write to in favor of density

Won't cloud technology provide the capacity solution?

- ❐ Erasure enables low cost hardware
- ❐ Object to enable massive scale
- ❐ Simple minded API, get put delete
- ❐ Problem solved → NOT

Campaign Storage Rqmnts
-Billions files / directory
-Trillions files total
-Files from 0 to 100 PB
-Multiple writers into one file and dir

- ❐ Works great for newly written apps for simple interface
- ❐ Doesn't work well for people, people need folders and rename and, N writers into single file, etc.
- ❐ Doesn't work for the $trillions of apps out there that expect some modest name space capability (parts of POSIX)

Won’t someone else do it, PLEASE circa 2015?

- There is evidence others see the need but no magic bullets yet: (partial list)
  - Cleversafe/Scality/EMC/Ceph ViPR/Swift etc. attempting multi-personality data lakes over erasure objects, all are young and assume update in place for posix
  - GlusterFS is probably the closes thing to MarFS. Gluster is aimed more for the enterprise and midrange HPC and less for extreme HPC. Glusterfs is a way to unify file and object systems, aiming at different uses
  - General Atomics Nirvana, Storage Resource Broker/IRODS optimized for WAN and HSM metadata rates. There are some capabilities for putting POSIX files over objects, but these methods are largely via NFS or other methods that try to mimic full file system semantics including update in place. These methods are not designed for massive parallelism in a single file, etc.
  - Bridgestore is a POSIX name space over objects but they put their metadata in a flat space so rename of a directory is painful.
  - Avere over objects is focused at NFS so N to 1 is a non starter.
  - HPSS or SamQFS or a classic HSM? Metadata rates designs are way low.
  - HDFS metadata doesn’t scale well.

Need a Scalable Near-POSIX Name Space over Cloud style Object Erasure?

Enter MarFS
The Sea of Data

- Best of both worlds
  - Objects Systems
    - Massive scaling and efficient erasure techniques
    - Friendly to applications. People need a name space.
    - Huge Economic appeal (erasure -> inexpensive storage)
  - POSIX name space is powerful but has issues scaling
- The challenges
  - Mismatch of POSIX an Object metadata, security, read/write semantics, efficient object/file sizes.
  - No update in place with Objects
  - Scaling POSIX name space to trillions of files/directories

MarFS (see talk on this)

What it is

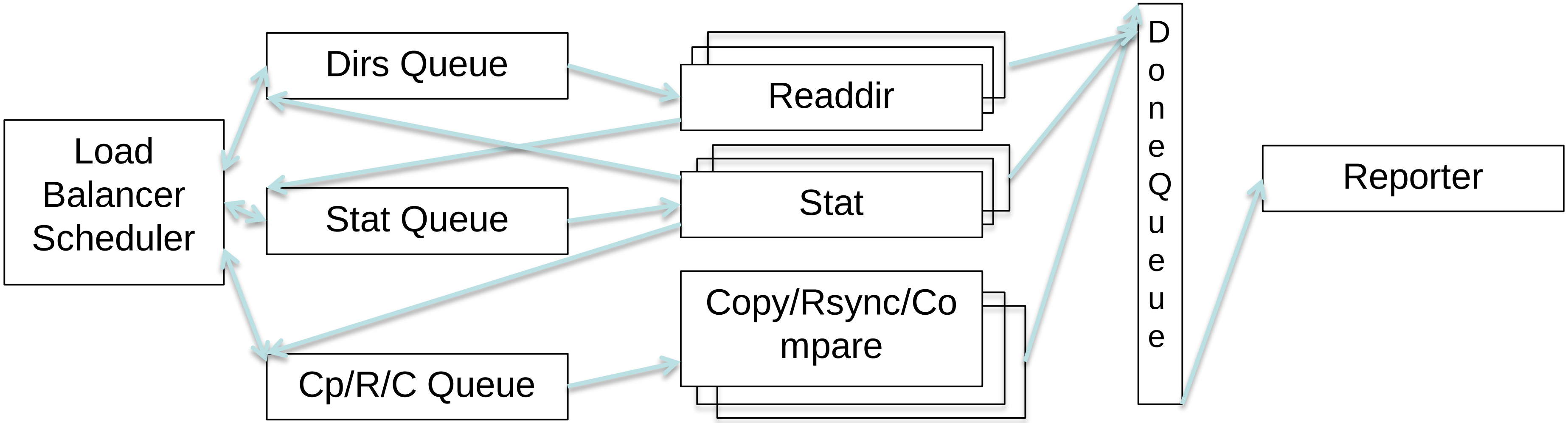
- 100-1000 GB/sec, Exabytes, Billion files in a directory, Trillions of files total
- Plug-able interface for metadata path and data path
  - Near-POSIX global scalable name space over many POSIX like metadata servers
  - Data spread over many POSIX or Object data repositories (Scalable object systems - CDMI, S3, etc.)
    - (Scality, EMC ECS, all the way to simple erasure over ZFS's)
- It is small amount of code (C/C++/Scripts)
  - A small Linux Fuse
  - A pretty small parallel batch copy/sync/compare/ utility
  - A moderate sized library both FUSE and the batch utilities call
- Data movement scales just like many scalable object systems
- Metadata scales like NxM POSIX name spaces both across the tree
- It is friendly to object systems by
  - Spreading very large files across many objects
  - Packing many small files into one large data object

What it isnt

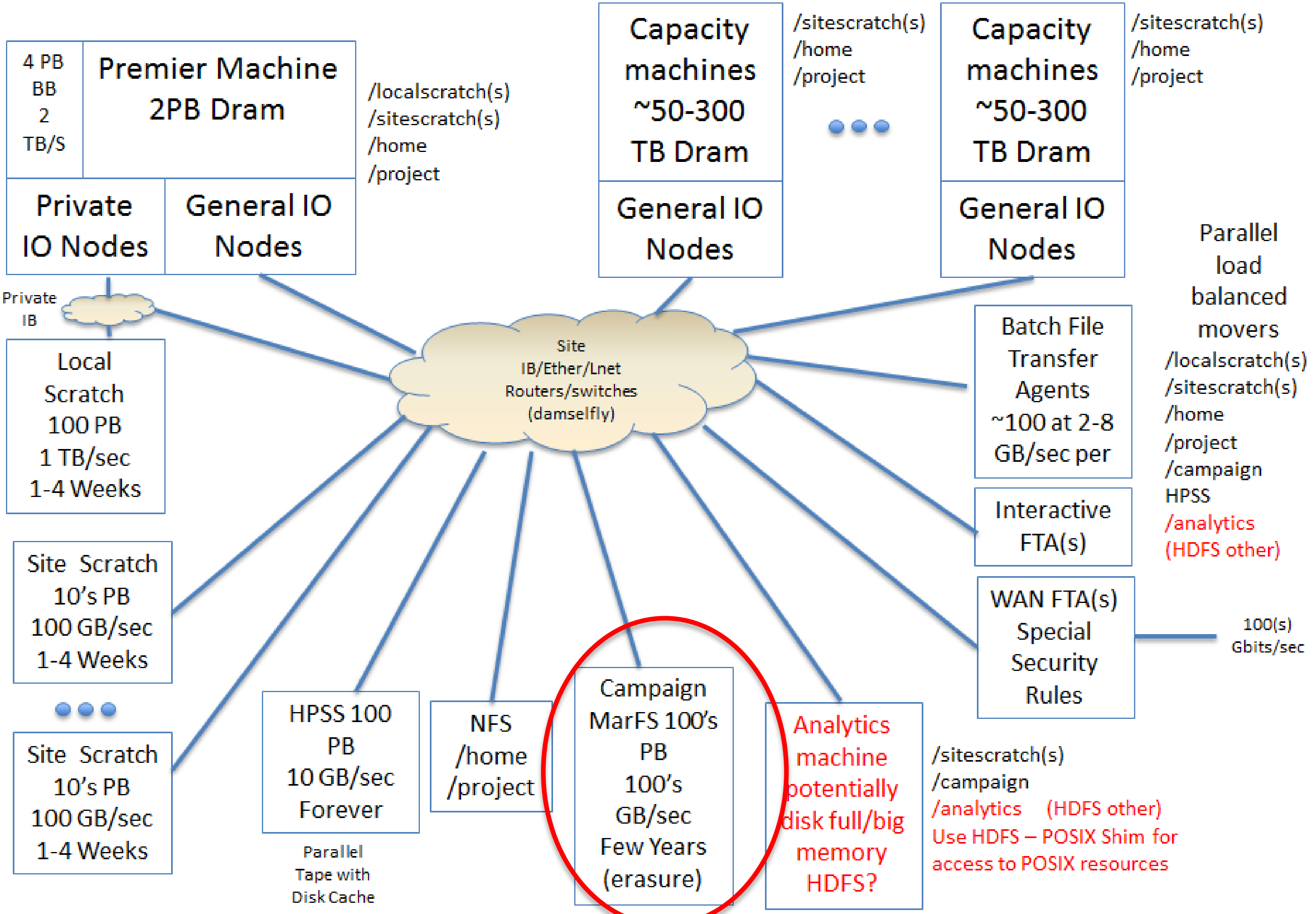
- No Update in place! Its not a pure file system, Complete overwrites are fine but no seeking and writing.

Pftool – parallel copy/rsync/compare/list tool

- Walks tree in parallel, copy/rsync/compare in parallel.
    - Parallel Readdir's, stat's, and copy/rsinc/compare
  - Dynamic load balancing
  - Restart-ability for large trees or even very large files
  - Repackage: breaks up big files, coalesces small files
  - To/From NFS/POSIX/parallel FS/MarFS



How does it fit into our environment in FY16



4 PB BB 2 TB/S

Premier Machine 2PB Dram

/localscratch(s)
/sitescratch(s)
/home
/project

Private IO Nodes

General IO Nodes

Private IB

Capacity machines ~50-300 TB Dram

General IO Nodes

/sitescratch(s)
/home
/project

Capacity machines ~50-300 TB Dram

General IO Nodes

/sitescratch(s)
/home
/project

Site IB/Ether/Lnet Routers/switches (damselfly)

Local Scratch 100 PB 1 TB/sec 1-4 Weeks

Site Scratch 10's PB 100 GB/sec 1-4 Weeks

Site Scratch 10's PB 100 GB/sec 1-4 Weeks

HPSS 100 PB 10 GB/sec Forever

Parallel Tape with Disk Cache

NFS /home /project

Campaign MarFS 100's PB 100's GB/sec Few Years (erasure)

Analytics machine potentially disk full/big memory HDFS?

/sitescratch(s)
/campaign
/analytics (HDFS other)
Use HDFS – POSIX Shim for access to POSIX resources

Batch File Transfer Agents ~100 at 2-8 GB/sec per

Interactive FTA(s)

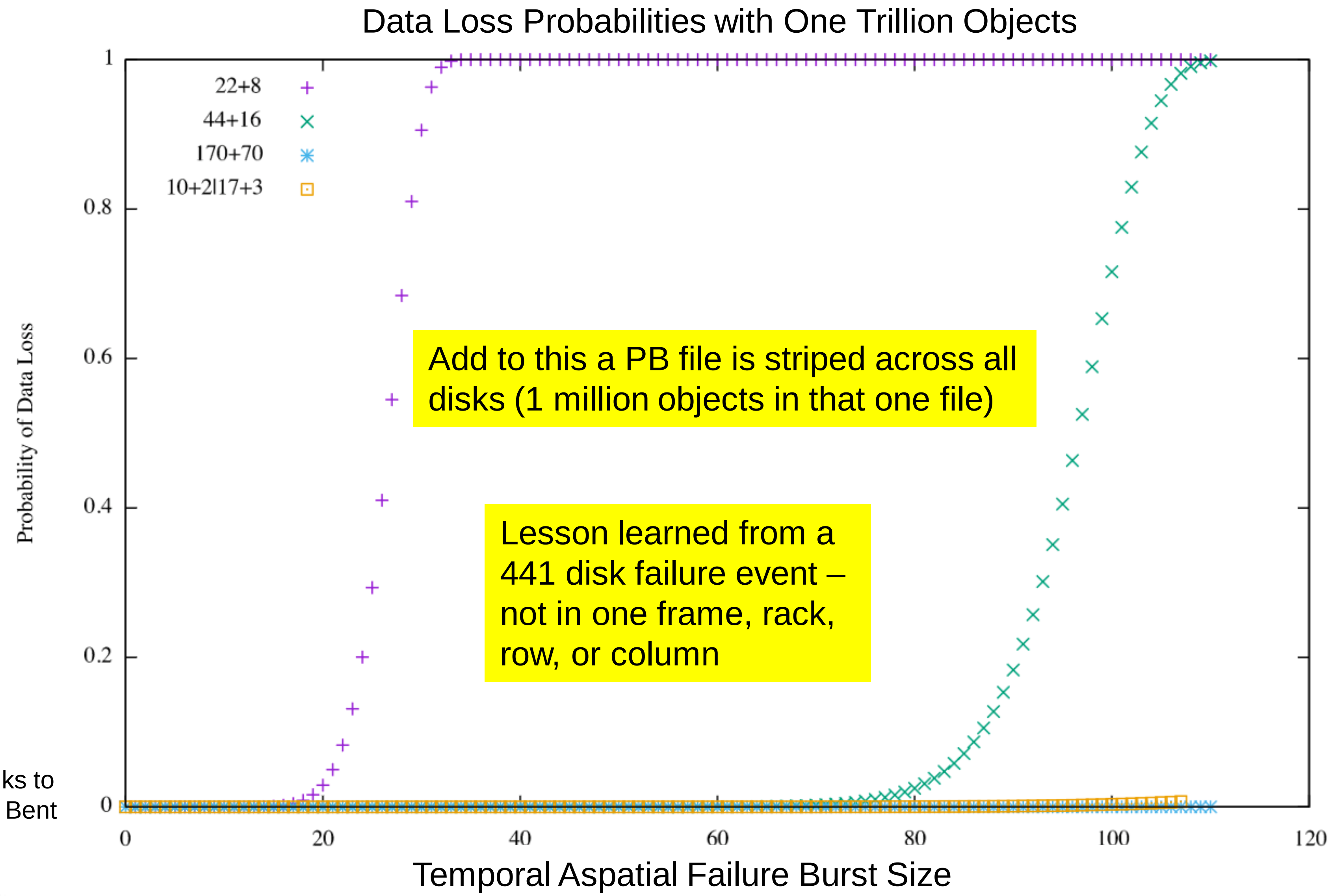
WAN FTA(s) Special Security Rules

100(s) Gbits/sec

Parallel load balanced movers

/localscratch(s)
/sitescratch(s)
/home
/project
/campaign
HPSS
/analytics
(HDFS other)

What about keeping the data for a long time

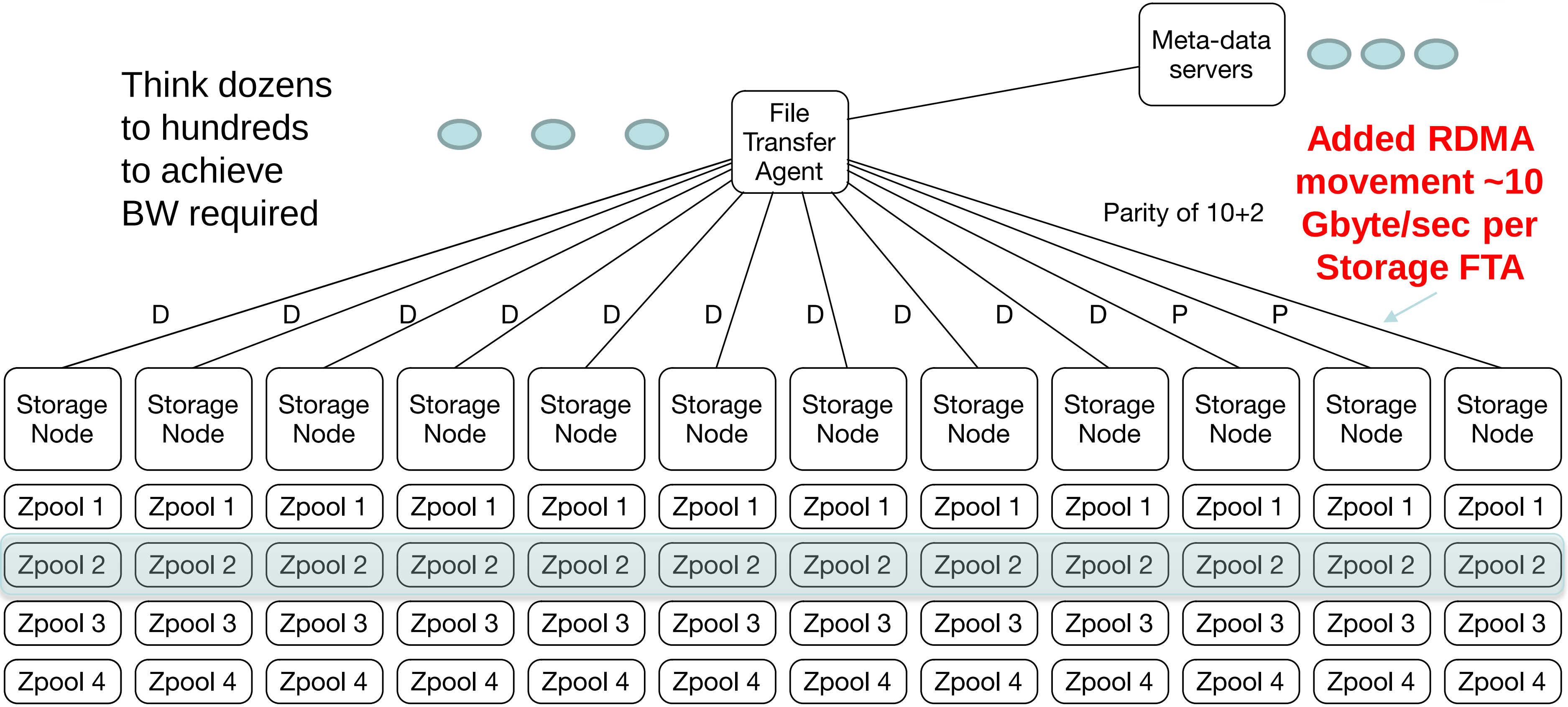


Add to this a PB file is striped across all disks (1 million objects in that one file)

Lesson learned from a 441 disk failure event – not in one frame, rack, row, or column

Thanks to
John Bent
DDN

Overcoming Correlated Failures – Two Tiered Erasure (see talk on this)



Think dozens
to hundreds
to achieve
BW required

File Transfer Agent

Meta-data servers

Parity of 10+2

Added RDMA movement ~10 Gbyte/sec per Storage FTA

Storage Node	Storage Node	Storage Node	Storage Node	Storage Node	Storage Node	Storage Node	Storage Node	Storage Node	Storage Node	Storage Node	Storage Node
Zpool 1	Zpool 1	Zpool 1	Zpool 1	Zpool 1	Zpool 1	Zpool 1	Zpool 1	Zpool 1	Zpool 1	Zpool 1	Zpool 1
Zpool 2	Zpool 2	Zpool 2	Zpool 2	Zpool 2	Zpool 2	Zpool 2	Zpool 2	Zpool 2	Zpool 2	Zpool 2	Zpool 2
Zpool 3	Zpool 3	Zpool 3	Zpool 3	Zpool 3	Zpool 3	Zpool 3	Zpool 3	Zpool 3	Zpool 3	Zpool 3	Zpool 3
Zpool 4	Zpool 4	Zpool 4	Zpool 4	Zpool 4	Zpool 4	Zpool 4	Zpool 4	Zpool 4	Zpool 4	Zpool 4	Zpool 4

Each Zpool
is a 17+3

Storage nodes
in separate racks

Multiple JBODs
per Storage Node

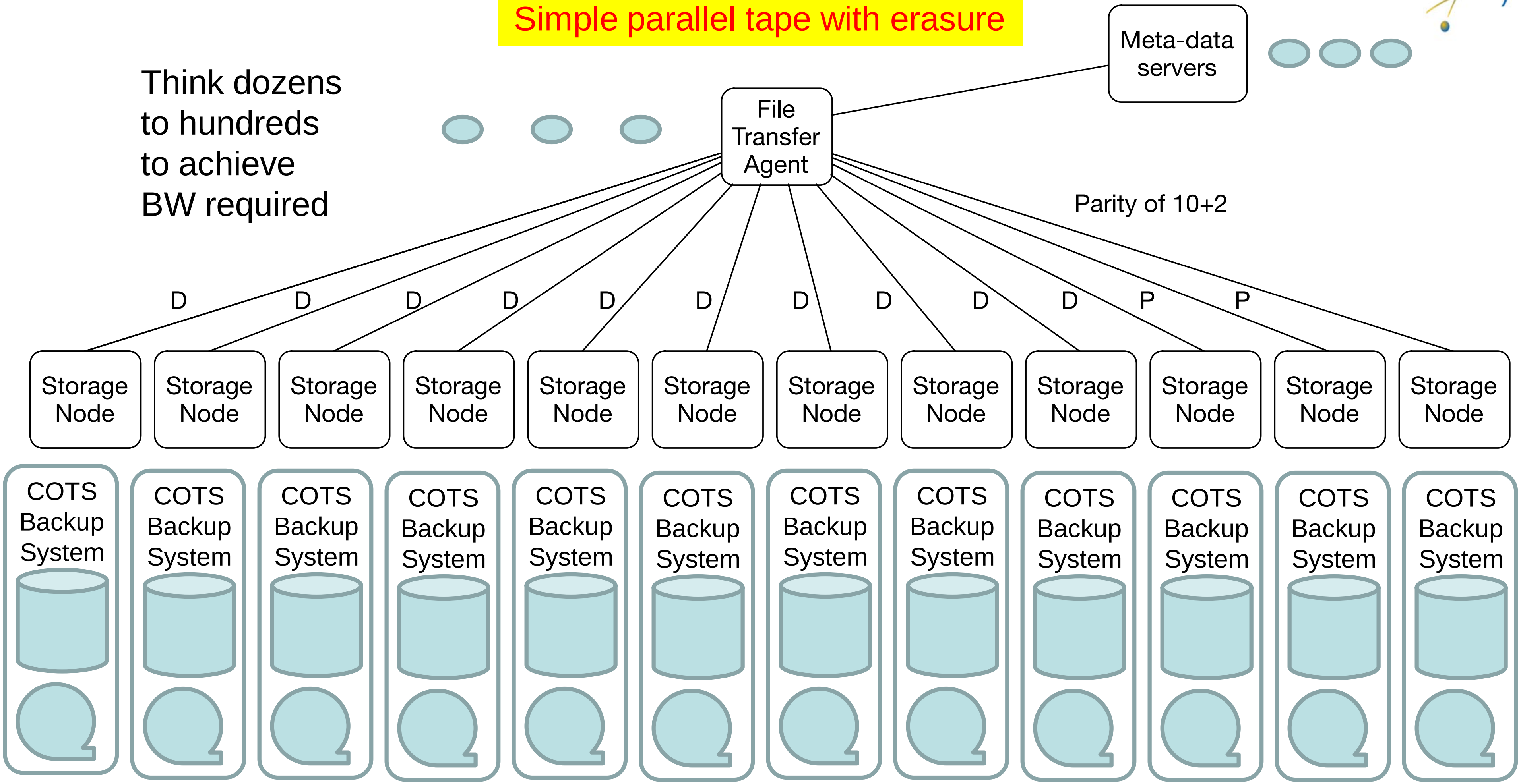
Data and Parity are round-robined
to storage nodes

Storage Nodes
NFS export to FTAs

Erasure on power managed disk or tape for extremely resilient archives

Simple parallel tape with erasure

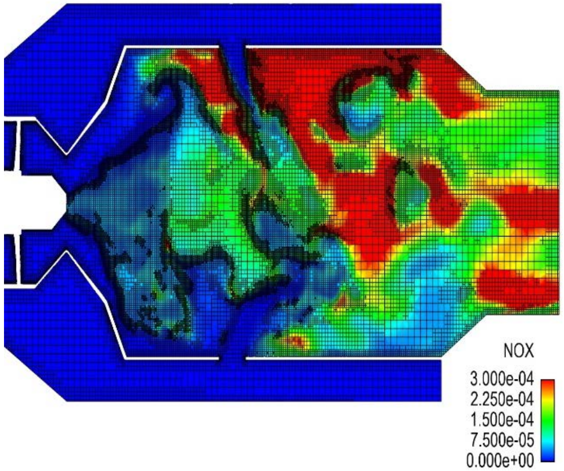
Think dozens to hundreds to achieve BW required



Hopefully we have whipped the scalable parallel data into submission on to Metadata pursuits

DeltaFS

A File System Service for Simulation Science



HXHIM

Indexing for Scientific Data

GUFI Grand Unified File Index

Fastest open-source software for supercomputer user-queried metadata

- Provides metadata queries of ultrascale file-system trees in seconds
- Maintains security structure while facilitating custom user metadata queries
- Economizes supercomputing resources — enhancing the role of the user
- Offers open-source software of a mere few thousand lines that is concise and extensible

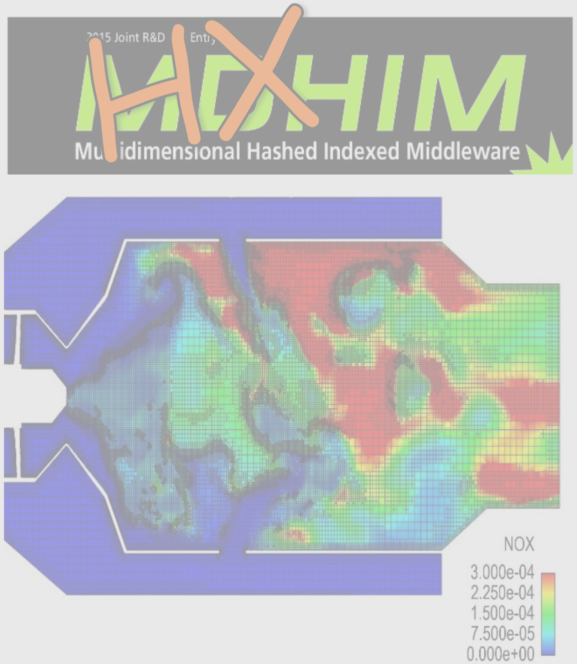
GUFI

Fast Userspace Metadata Query

A dynamically loadable namespace – DeltaFS
Lets make metadata scale with the application!

DeltaFS

A File System Service for Simulation Science



HXHIM

Indexing for Scientific Data

GUFI Grand Unified File Index

Fastest open-source software for supercomputer user-queried metadata

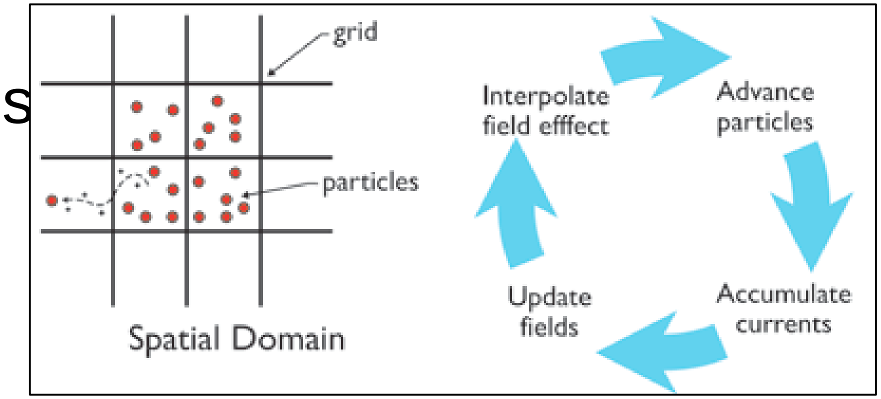
- Provides metadata queries of ultrascale file-system trees in seconds
- Maintains security structure while facilitating custom user metadata queries
- Economizes supercomputing resources — enhancing the role of the user
- Offers open-source software of a mere few thousand lines that is concise and extensible

GUFI

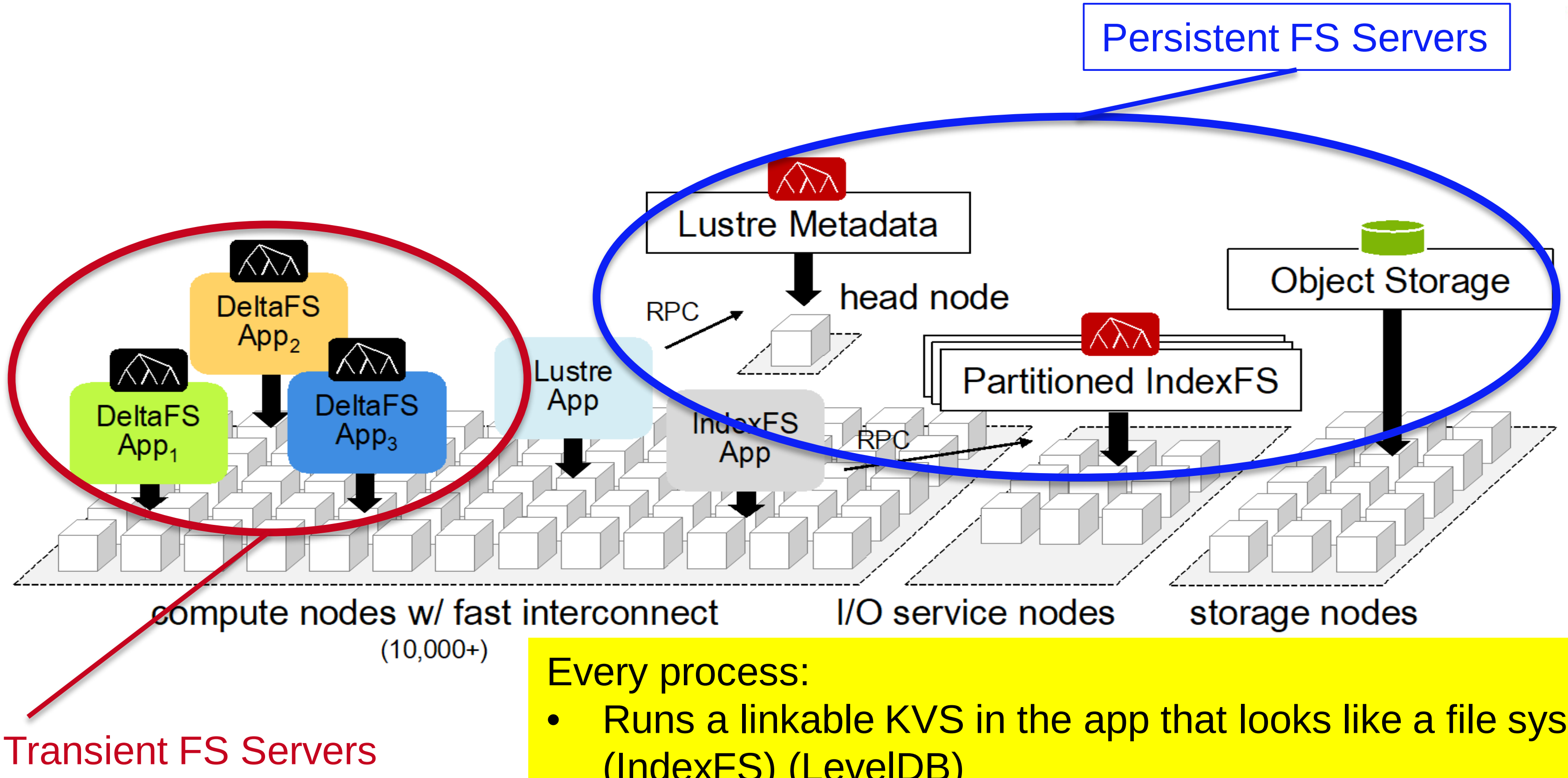
Fast Userspace Metadata Query

Brief VPIC Overview

- **Particle-in-cell MPI code (scales to ~100K processes)**
  - Fixed mesh range assigned to each process
  - 32 – 64 Byte particles
  - Particles move frequently between Millions of processes
  - Million particles per node (Trillion particle in target simulation)
  - Interesting particles identified at ***simulation end***



Brief DeltaFS Overview

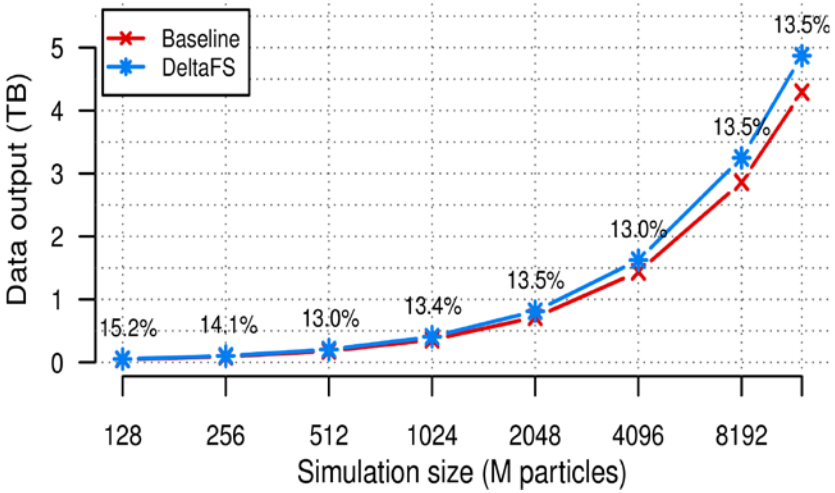


Every process:

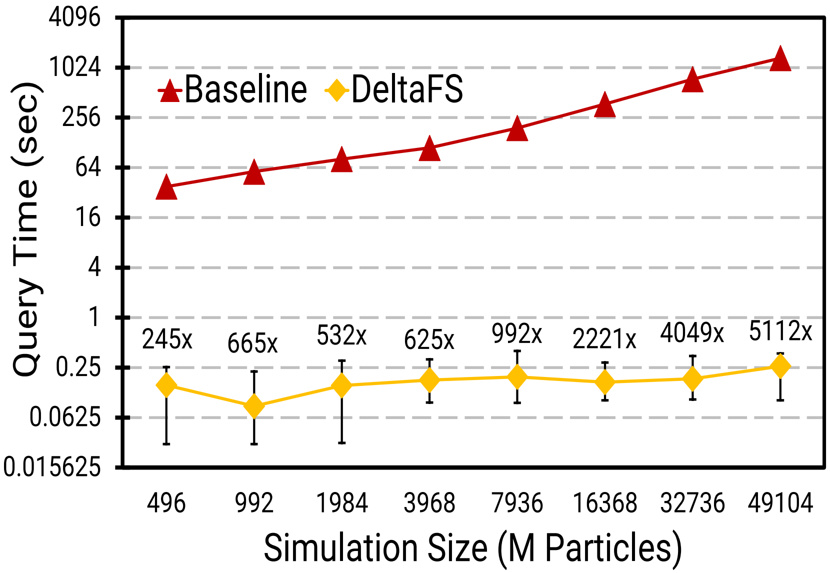
- Runs a linkable KVS in the app that looks like a file system (IndexFS) (LevelDB)
- “Checks Out” its namespace for the particles files it will hold – loads a LevelDB SSTable with hundreds of thousands of ”particle files” time stamp records.
- When Storing particle records(files) sent to appropriate srv
- This is writing data to a 10’s of thousands distributed KVS

Tracking the Highest Energy Particles
Recall the intent is 1 Trillion particles
These thousand particles are interesting, where have they been?

VPIC 1 time step Particle Dump Size



VPIC Particle Trajectory Query



Collaboration of CMU, LANL, ANL, HDF Group
(papers at PDSW 15, PDSW 17, SC18)

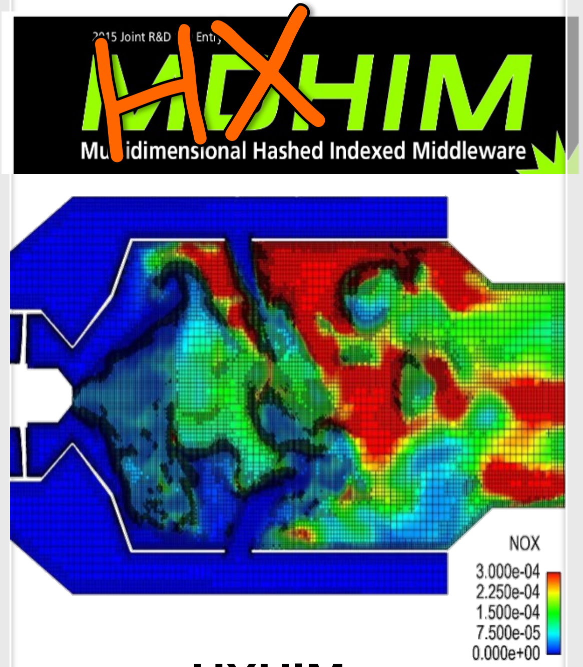
Application thought it was writing/reading from 1 file per trillion particles but really was writing records to massive parallel distributed KVS!
Today we are getting like 8 Billion Particle File Ops/Sec. (yes Billion)

Isn’t 8 Billion Metadata ops/sec good enough? Well maybe, but that was low dimensional Metadata. What about higher dimensional Metadata?

Now that I know “where the interesting particles were” what was going on around those interesting particles? What about complicated meshes? MDHIM->XDHIM (Thank you to DoD and DoE ECP funding)

DeltaFS

A File System Service for Simulation Science



HXHIM

Indexing for Scientific Data

GUFI Grand Unified File Index

Fastest open-source software for supercomputer user-queried metadata

- Provides metadata queries of ultrascale file-system trees in seconds
- Maintains security structure while facilitating custom user metadata queries
- Economizes supercomputing resources — enhancing the role of the user
- Offers open-source software of a mere few thousand lines that is concise and extensible

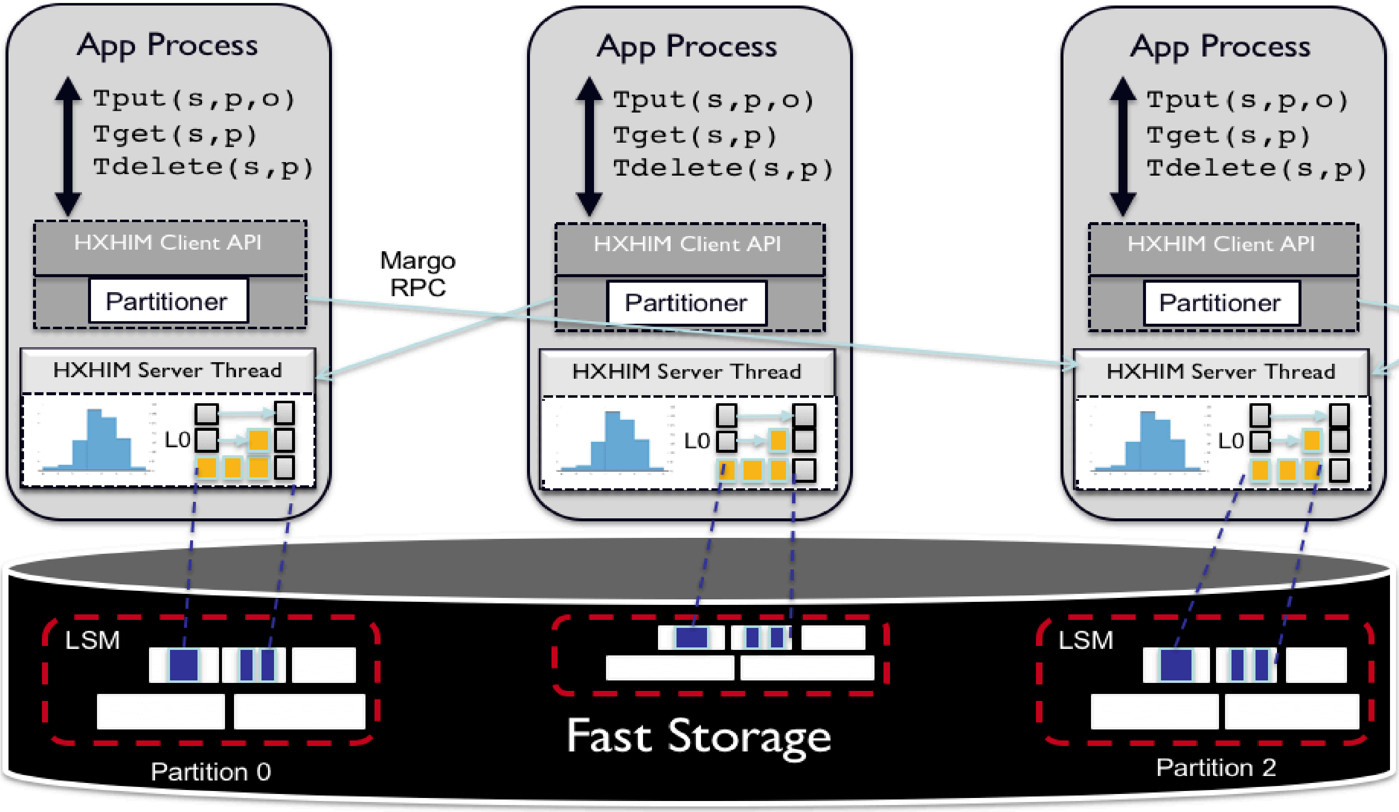
GUFI

Fast Userspace Metadata Query

MDHIM/XDHIM (why make100 thousand KVS’s look like one)

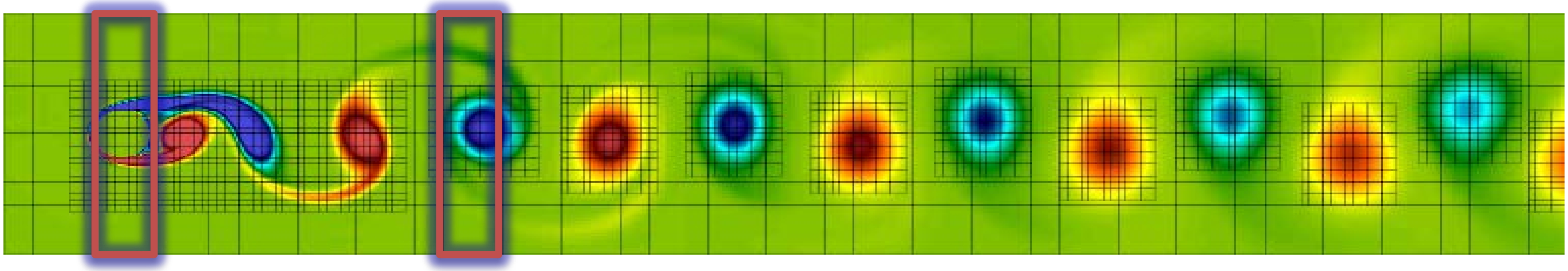
An application linkable parallel KVS Framework

KVS is linked in the application, bulk put/get ops, uses key range sharding and server side stored procedures, X Dimensional Sharded Index (Hexastore 6 dimensional linkable KVS is currently in use)



HXHIM – Indexing for Unstructured Meshes

- **How do you store/represent an AMR mesh?**
- **(What is AMR and Why Do We Care?)**



How many rows are in each of these columns?

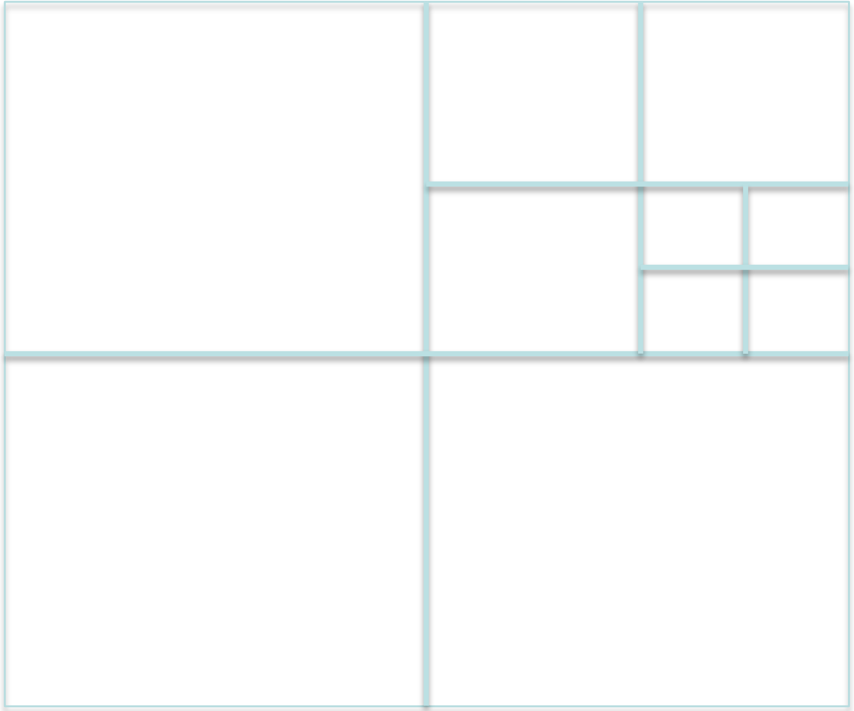
(For that matter, how many columns are in each of these columns?!)

How do you store this kind of time series data in a usable form?

- **Key-value exposes the data structures underlying most FS**
- **Key-value allows fine-grained data annotation**
- **Need to add some HPC research to make efficient for HPC platforms**
  - Mercury RPC and Margo (lightweight IO threads) for platform services
  - Multidimensional Hashing Indexing Middleware

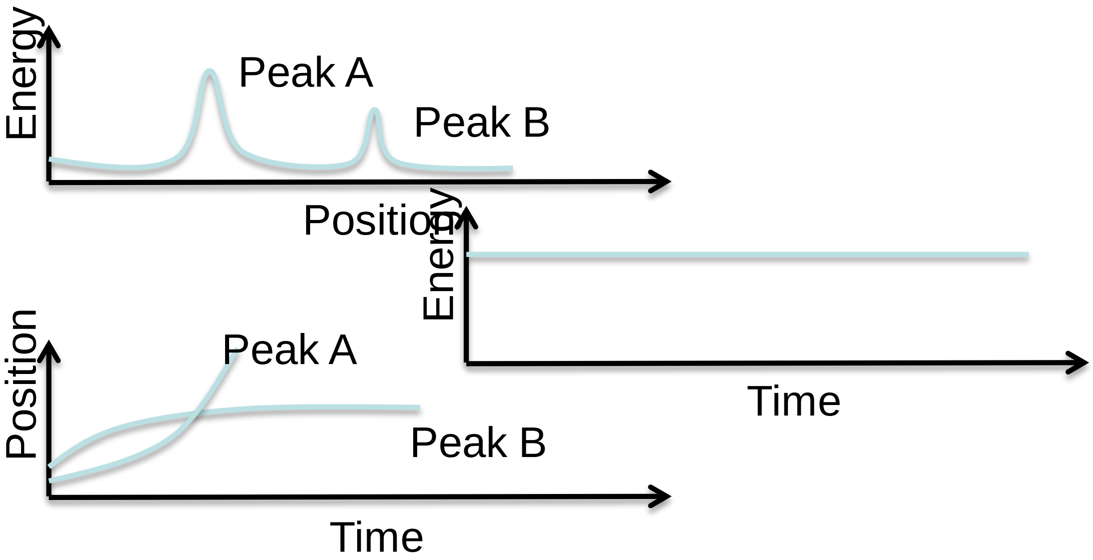
HXHIM Mesh Storage Example

- If “position” in the mesh is the key, and you keep subdividing the key, how do you have a reasonable key structure
- Old trick using hierarchy of keys (borrow from Farsite FS - Microsoft)



Subject	Predicate	Object
mesh	name	“My Mesh”
sim	timestep	3.0
c0	position	[0.0,0.0]
c1	position	[0.1,0.0]
c2	position	[0.0,0.1]
c3.0	position	[0.1,0.1]
c3.1.0	position	[0.15,0.1]
c3.1.1	position	[0.175,0.1]
c3.1.2	position	[0.125,0.15]
c3.1.3	position	[0.125,0.125]
c3.2	position	[0.1,0.15]

Sample Query: Tracking a Wave thru Time



- A fast multi-dimensional index
  - Time is discretized separately (indexing not required)
  - Energy and position must both be indexed (and not trivially)
    - Energy extrema search is worse than VPIC example!
  - Efficient filtering for contiguity!
    - We could probably work around most of these problems, but level arrays will always convert spatially contiguous workloads into disjoint query sets
    - Neighbor lists won't limit the pointer chasing
- Why do I think a Key-Value organization can do better?

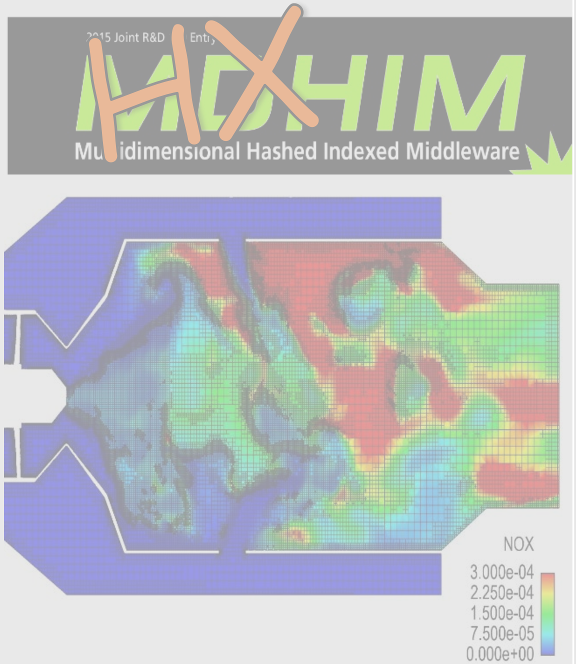
Range-based Iteration with Stored Procedures

- **Advantages of Key-Value Organization**
  - Decouples file size, I/O size from data set size (efficient I/O)
  - Keyspace *dimension* can change dynamically
    - Leverage naming technique described by Farsite FS
  - Supports iteration across multiple dimensions simultaneously
  - In-situ rather than post-hoc
- **Advantages of client-server architectures**
  - Even with the above we can't accomplish what we need
  - Stored procedures to identify extrema in-situ

How do we ever find anything in our trillions of files? GUFI Grand Unified File Index

DeltaFS

A File System Service for Simulation Science



HXHIM

Indexing for Scientific Data

GUFI Grand Unified File Index

Fastest open-source software for supercomputer user-queried metadata

- Provides metadata queries of ultrascale file-system trees in seconds
- Maintains security structure while facilitating custom user metadata queries
- Economizes supercomputing resources — enhancing the role of the user
- Offers open-source software of a mere few thousand lines that is concise and extensible

GUFI

Fast Userspace Metadata Query

Motivation

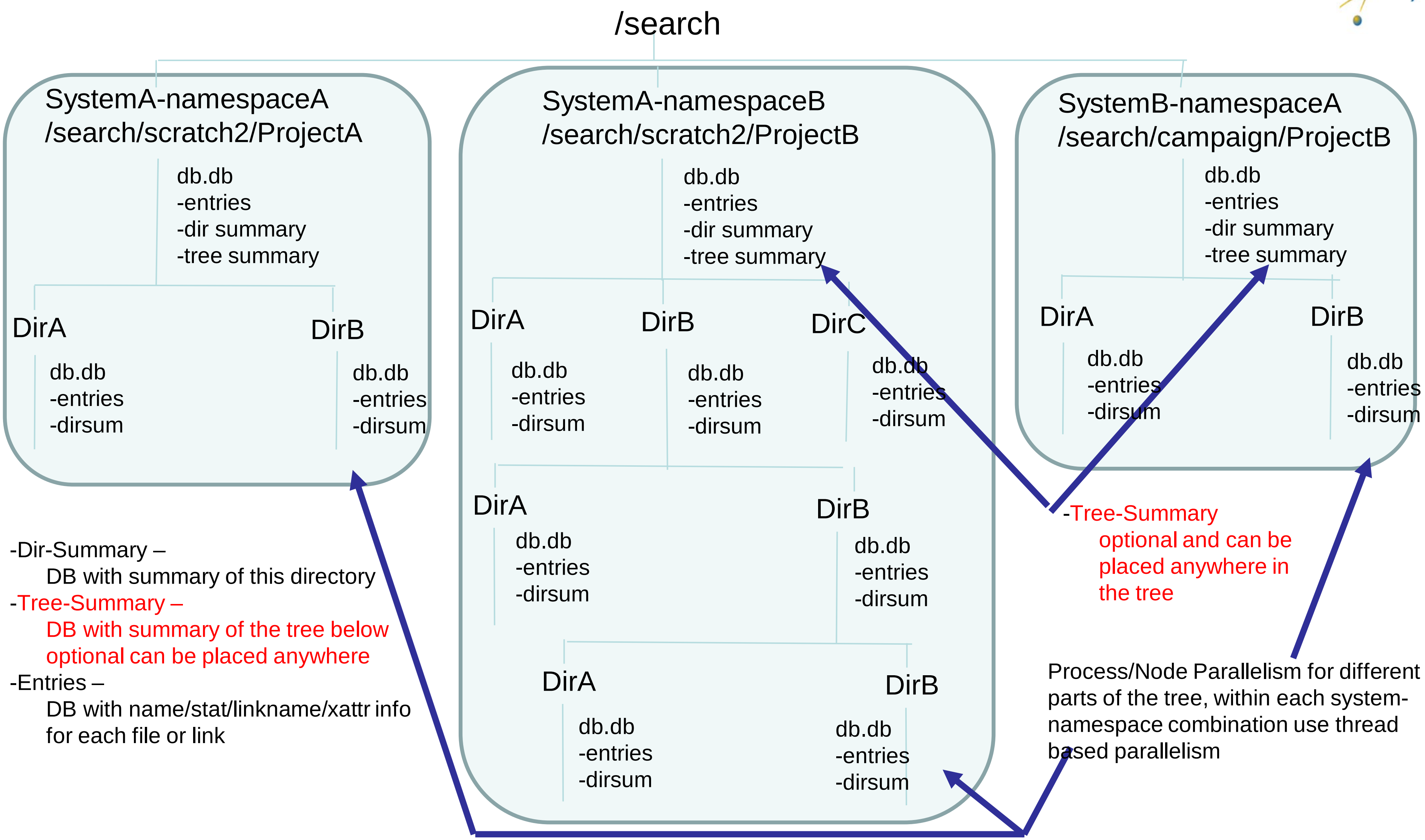
- **Many layers of storage at LANL**
  - By design – users would have us only buying storage if we used automatic policy driven HSMs
- **Data management by users is driven by need, sporadically**
  - Users go find unneeded data and delete, if prodded
  - Users have no easy way to find particular datasets unless they have a good hierarchy or they remember where they put it
  - Users have bad memories and bad hierarchies…(you can see where this leads)
  - ...lower (longer) tiers of storage systems accumulate cruft over time

GUFI Goals

- **Unified** index over home, project, scratch, campaign, and archive
- **Metadata only** with extended attribute support
- Shared index for **users** and admins
- **Parallel search** capabilities that are very fast (minutes for billions of files/dirs)
- Search results can appear as a **mounted File System**
- Full/Incremental update from sources in reasonable time/annoyance
- Leverage **existing tech** as much as possible both hdwr and software: flash, threads, clusters, sql as part of the interface, commercial db tech, commercial indexing systems, commercial file system tech, threading/parallel process/node run times, src file system full/incremental capture capabilities, posix tree attributes (permissions, hierarchy representation, etc.), open source/agnostic to leveraged parts where possible.
- **Simple** so that an admin can easily understand/enhance/troubleshoot

- **Why not a flat namespace?**
  - Performance is great, but…Rename high in the tree is terribly costly
  - Security becomes a nightmare if users and admins can access the namespace

GUFI Prototype



/search

SystemA-namespaceA
/search/scratch2/ProjectA

db.db
-entries
-dir summary
-tree summary

DirA

db.db
-entries
-dirsum

DirB

db.db
-entries
-dirsum

SystemA-namespaceB
/search/scratch2/ProjectB

db.db
-entries
-dir summary
-tree summary

DirA

db.db
-entries
-dirsum

DirB

db.db
-entries
-dirsum

DirC

db.db
-entries
-dirsum

DirA

db.db
-entries
-dirsum

DirB

db.db
-entries
-dirsum

DirA

db.db
-entries
-dirsum

DirB

db.db
-entries
-dirsum

SystemB-namespaceA
/search/campaign/ProjectB

db.db
-entries
-dir summary
-tree summary

DirA

db.db
-entries
-dirsum

DirB

db.db
-entries
-dirsum

-Dir-Summary –
DB with summary of this directory
-Tree-Summary –
DB with summary of the tree below
optional can be placed anywhere
-Entries –
DB with name/stat/linkname/xattr info
for each file or link

-Tree-Summary
optional and can be
placed anywhere in
the tree

Process/Node Parallelism for different parts of the tree, within each system-namespace combination use thread based parallelism

Programs Included / In Progress

- DFW – depth first walker, prints pinode, inode, path, attrs, xattrs
- BFW – breadth first walker, prints pinode, inode, path, attrs, xattrs
- BFWI – breadth first walker to create GUFI index tree from source tree
- BFMI – walk Robinhood MySQL and list tree and/or create GUFI index tree
- BFTI – breadth first walker that summarizes a GUFI tree from a source path down, can create treesummary index of that info
- BFQ – breadth first walker query that queries GUFI index tree
  - Specify SQL for treesummary, directorysummary, and entries DBs
- BFFUSE – FUSE interface to run POSIX md tools on a GUFI search result
- Querydb – dumps treesummary, directorysummary, and optional entry databases given a directory in GUFI as input
- Programs to update, incremental update (in progress):
  - Lustre, GPFS, HPSS

Early performance indicators

- All tests performed on a 2014 Macbook (quad core + SSD)
- No indexes used
- ~136k directories, mostly small directories, 10 1M entry dirs, 20 100K size dirs, and 10 20M size dirs
- ~250M files total represented
- Search of all files and dirs: 2m10s (~1.75M files/sec)
- Search of all files and dirs, but exclude some very large dirs: 1m18s
- …on a laptop!

Open Source
BSD License
Partners Welcome

https://github.com/mar-file-system/marfs
https://github.com/pftool/pftool
https://github.com/mar-file-system/GUFI
https://github.com/mar-file-system/erasureUtils

Thanks to all that participated in this work

Thanks For Your Attention