Los Alamos
NATIONAL LABORATORY
EST. 1943

Delivering science and technology
to protect our nation
and promote world stability

NNSA
National Nuclear Security Administration

LA-UR-18-24085

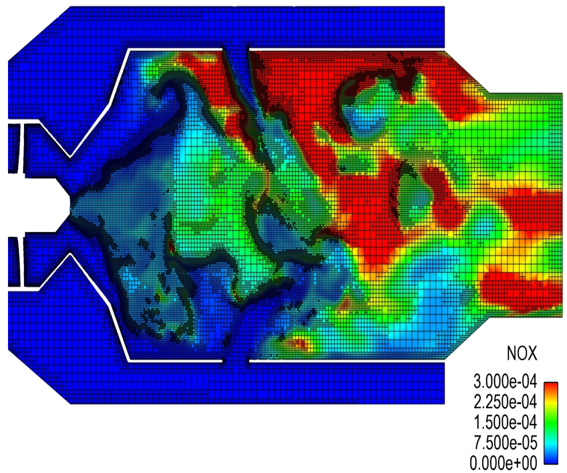
Massive Scale Metadata

Efforts and Solutions

David Bonnie

May 16th, 2018

Metadata at Scale



DeltaFS

A File System Service for Simulation Science

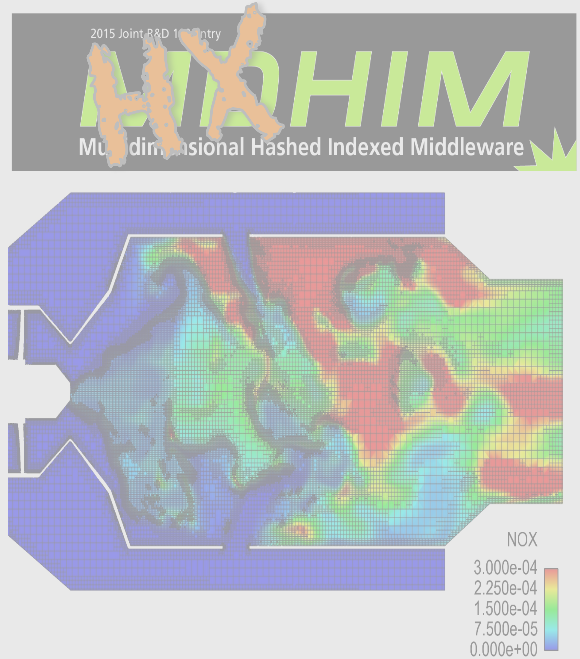
HXHIM

Indexing for Scientific Data

GUFI

Fast Userspace Metadata Query

Metadata at Scale



DeltaFS

A File System Service for Simulation Science

HXHIM

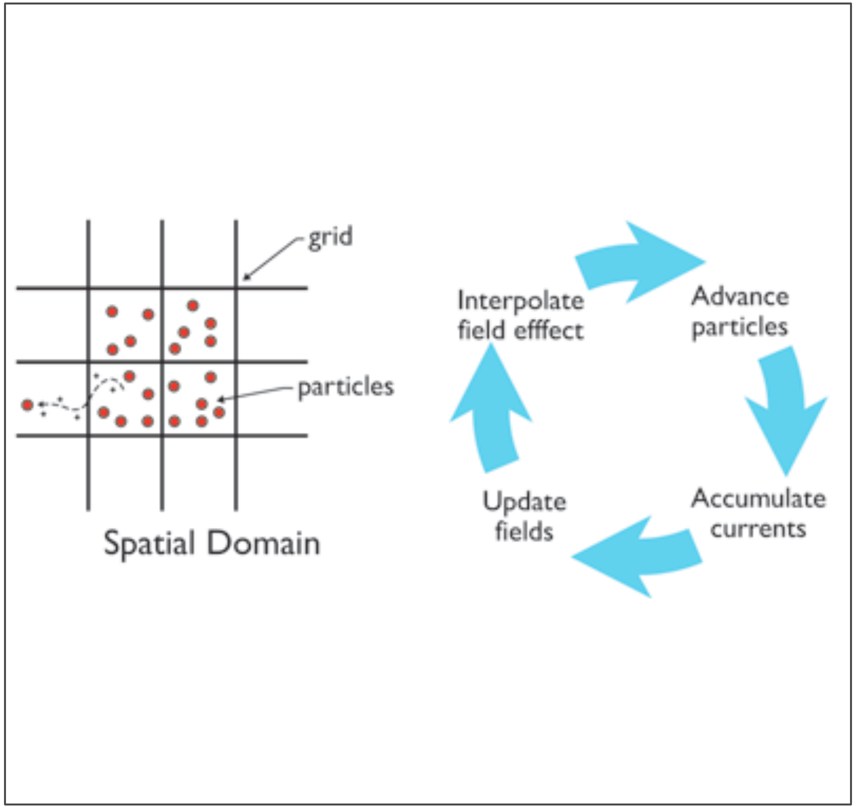
Indexing for Scientific Data

GUFI

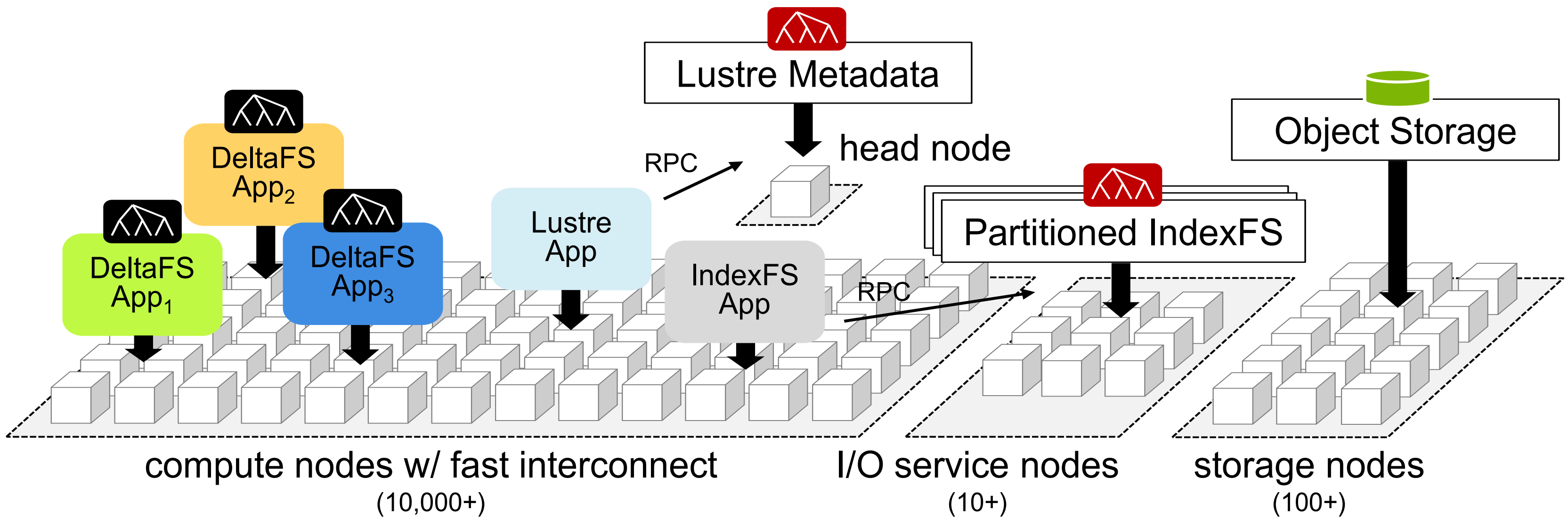
Fast Userspace Metadata Query

Brief VPIC Overview

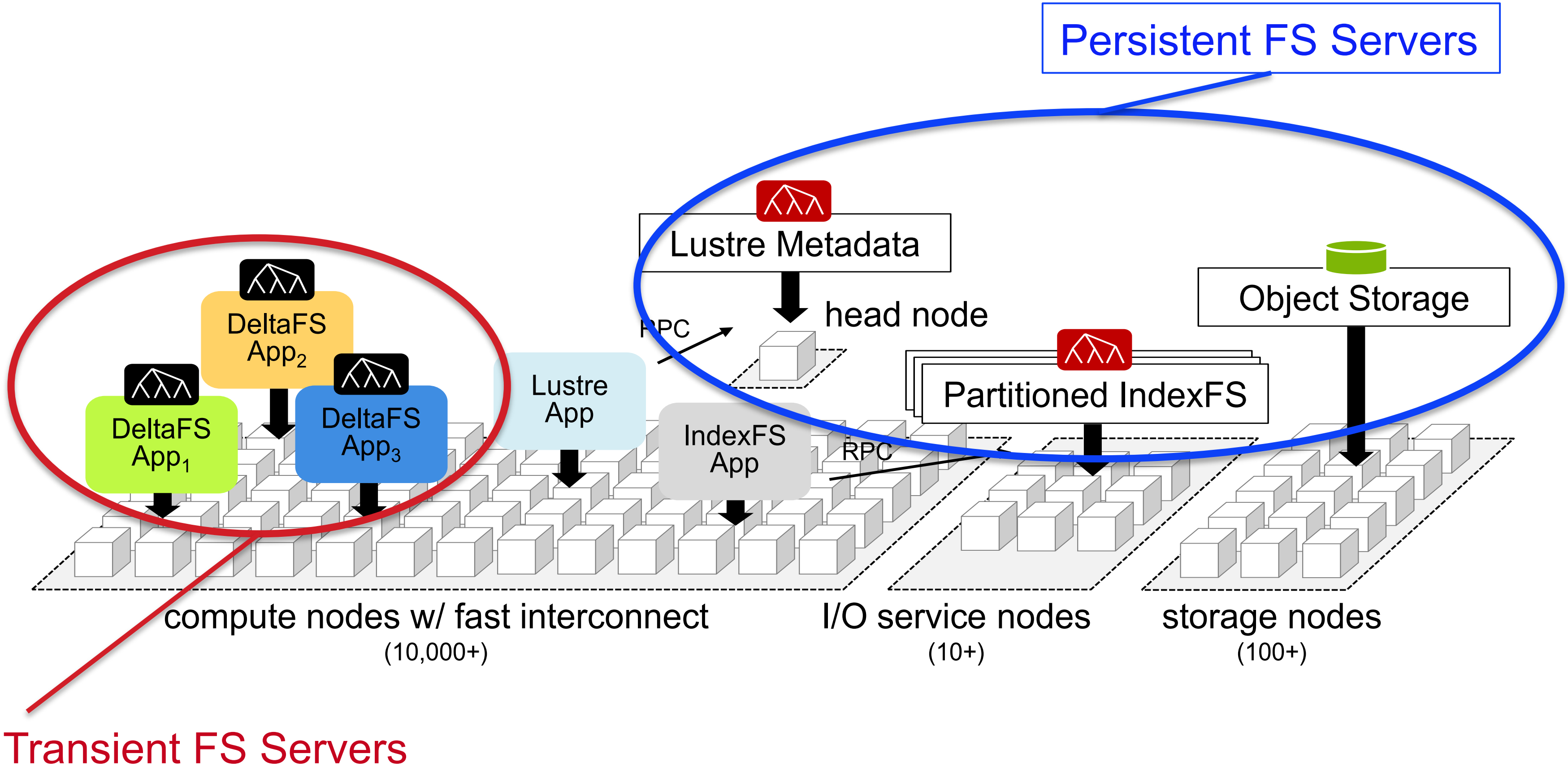
- **Particle-in-cell MPI code (scales to ~100K processes)**
  - Fixed mesh range assigned to each process
  - 32 – 64 Byte particles
  - Particles move frequently between 10's of thousands of processes
  - Million particles per node (Trillion particle in target simulation)
  - Interesting particles identified at ***simulation end***



Brief DeltaFS Overview

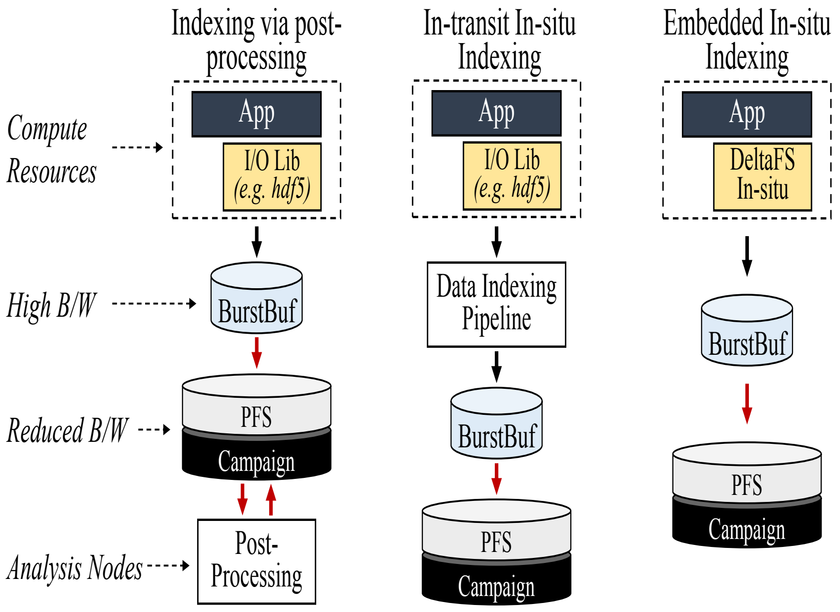


Brief DeltaFS Overview



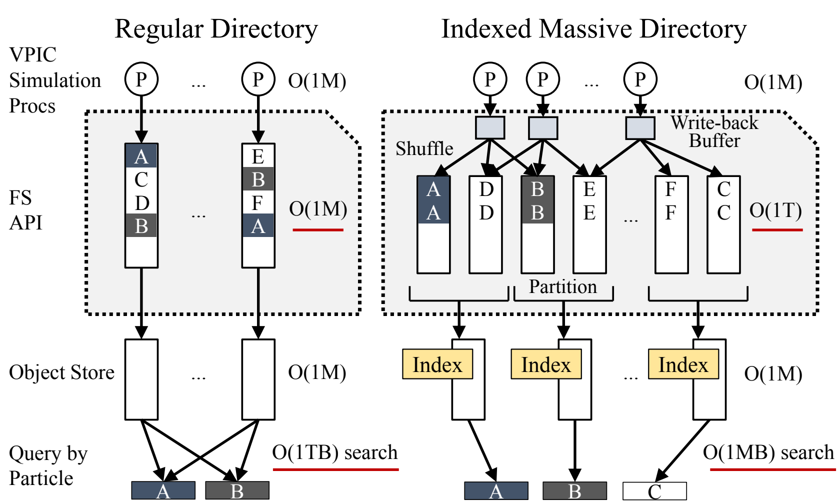
VPIC + DeltaFS: Storing a Particle per File

- **Store each particle as a file in a single directory**
  - TableFS metadata organization
  - New DeltaFS data plane
- **Embedded indexing/partitioning pipeline**
  - Leverage idle resources during bulk-synchronous output
- **Custom storage organization**
  - Partitioning + hash tables + clustered indexes

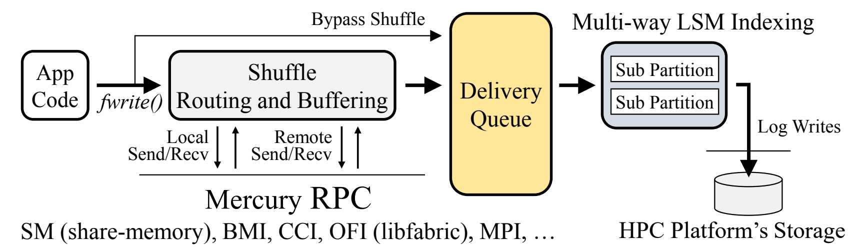


DeltaFS: Indexed Massive Directories

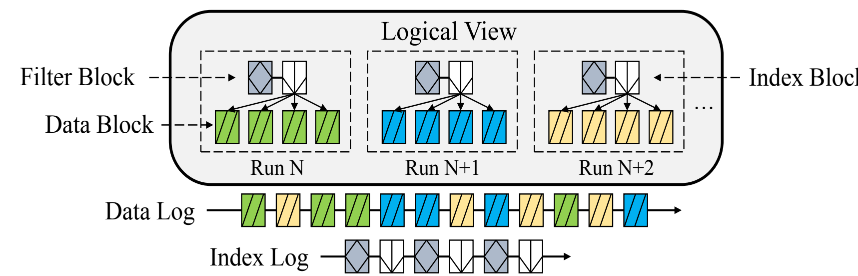
Concept Overview



Implementation

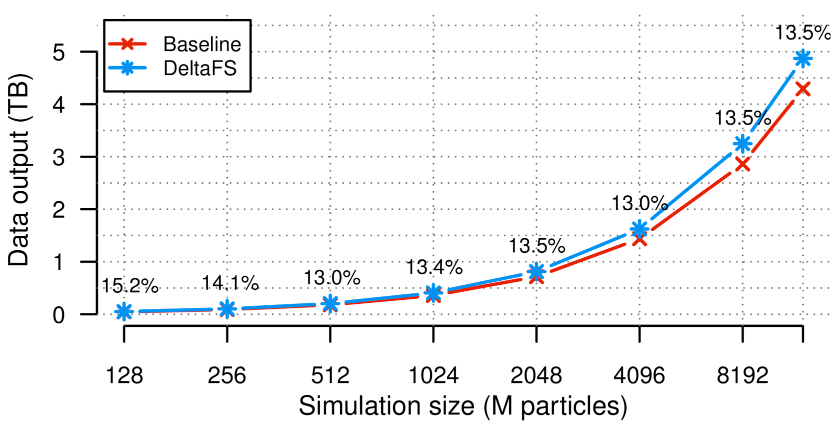


Storage Organization

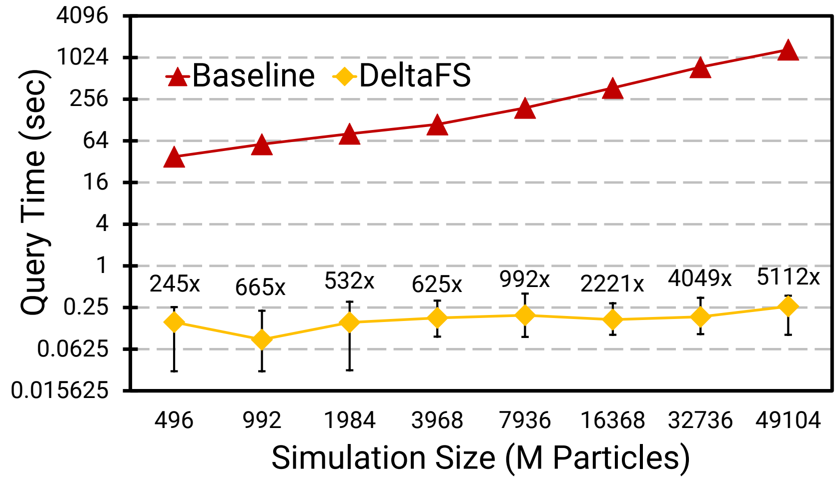


Tracking the Highest Energy Particles

VPIC Particle Dump Size



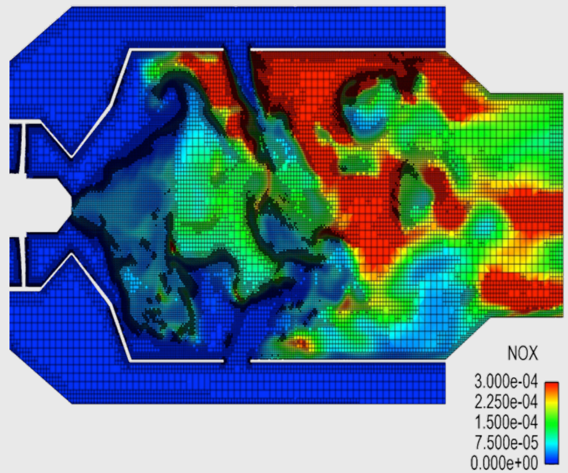
VPIC Particle Trajectory Query



Collaboration of CMU, LANL, ANL, HDF Group

(papers at PDSW 15, PDSW 17, SC18?)

Metadata at Scale



DeltaFS

A File System Service for Simulation Science

HXHIM

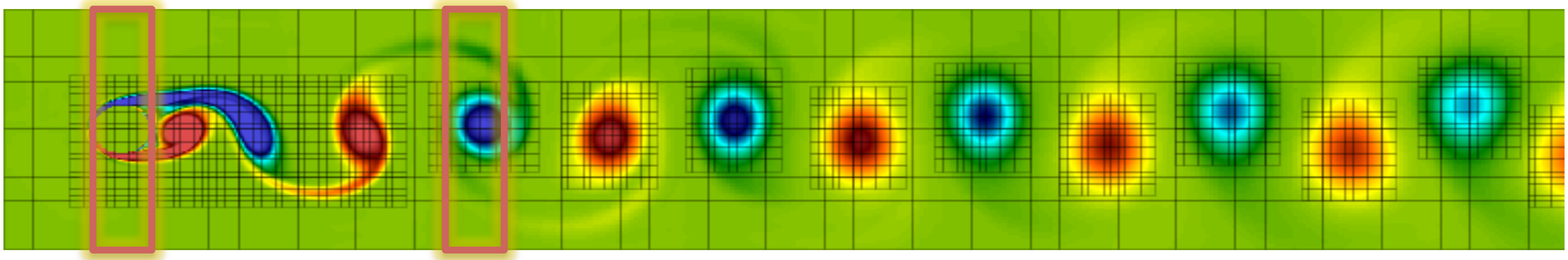
Indexing for Scientific Data

GUFI

Fast Userspace Metadata Query

HXHIM – Indexing for Unstructured Meshes

- **How do you store/represent an AMR mesh?**
  - In memory, dynamic tree and nested list structures are common



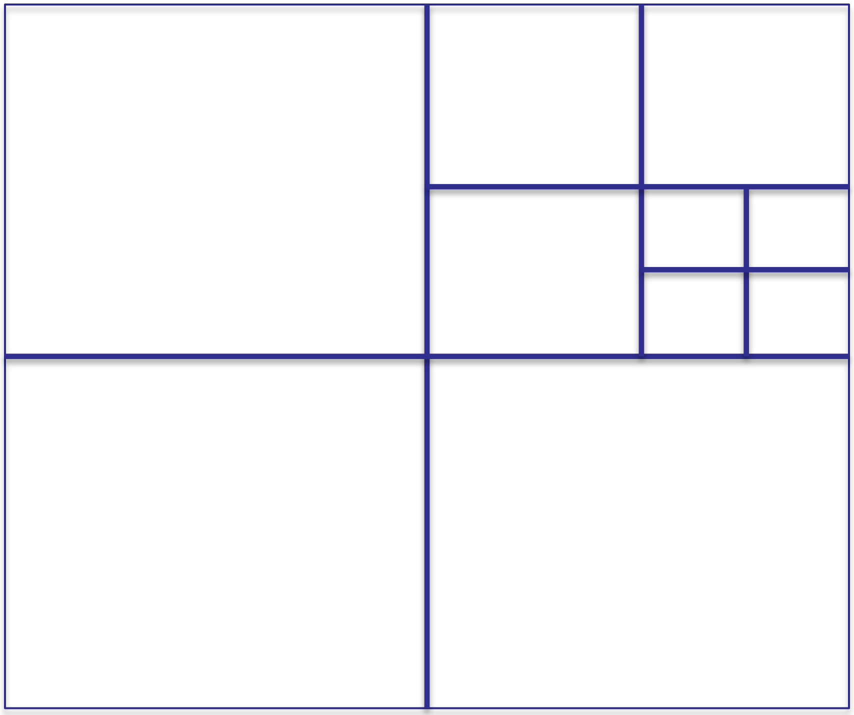
How many rows are in each of these columns?

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

Why Key-Value?

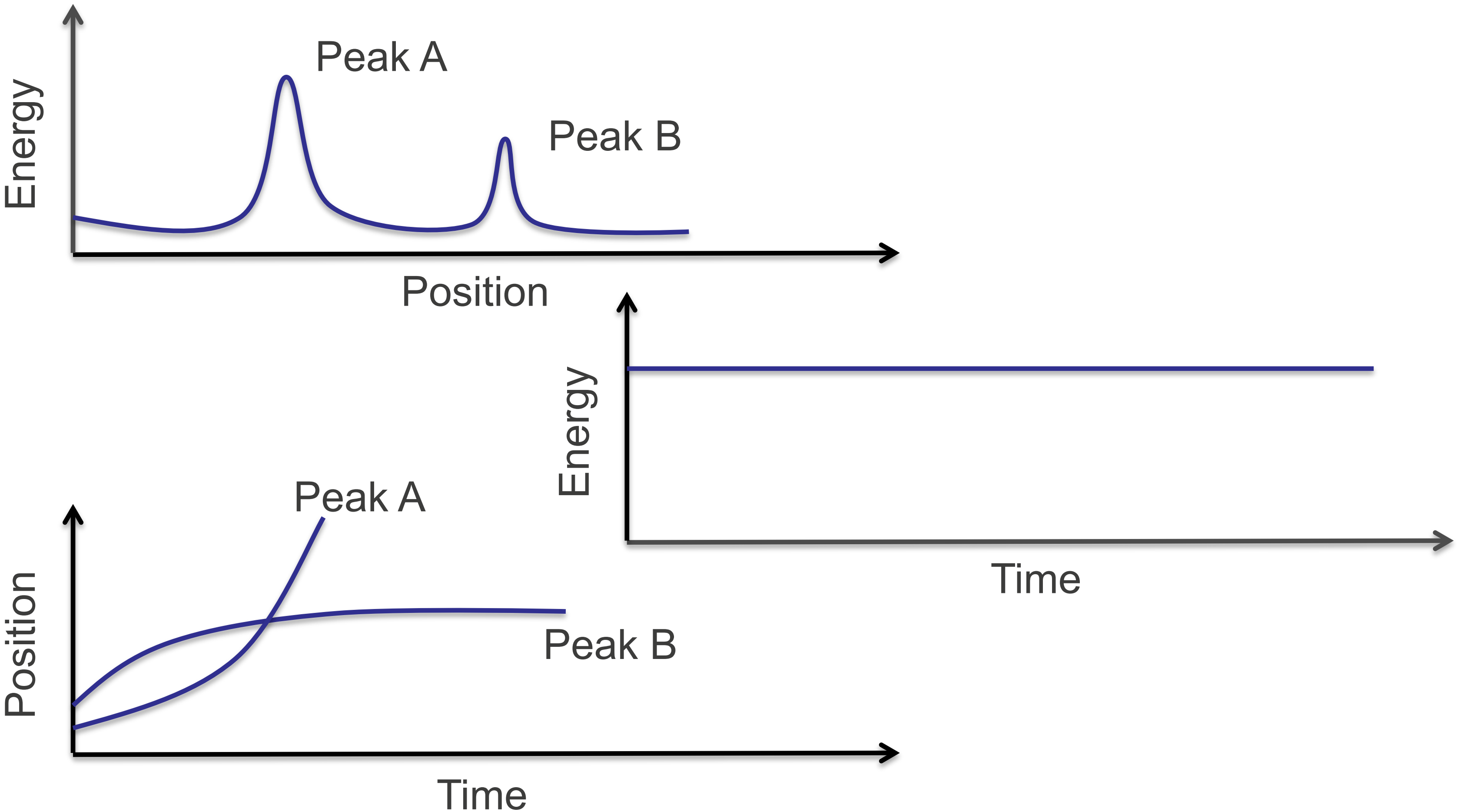
- **Key-value exposes the data structures underlying most FS**
  - B-Tree, Log-Structured Merge Trees, LSM Tries, B-epsilon Trees …
  - Can be tailored to range or point queries, different key-value size ratios
- **Key-value allows fine-grained data annotation**
  - Not a unique benefit
  - Key-space flexibility is useful – additional metadata is additive/auxiliary
  - Resource Description Format layers neatly on top
- **Key-Value is an effective way to expose a lot of CS research**
  - Hybridized indexing for point queries and/or range queries
- **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

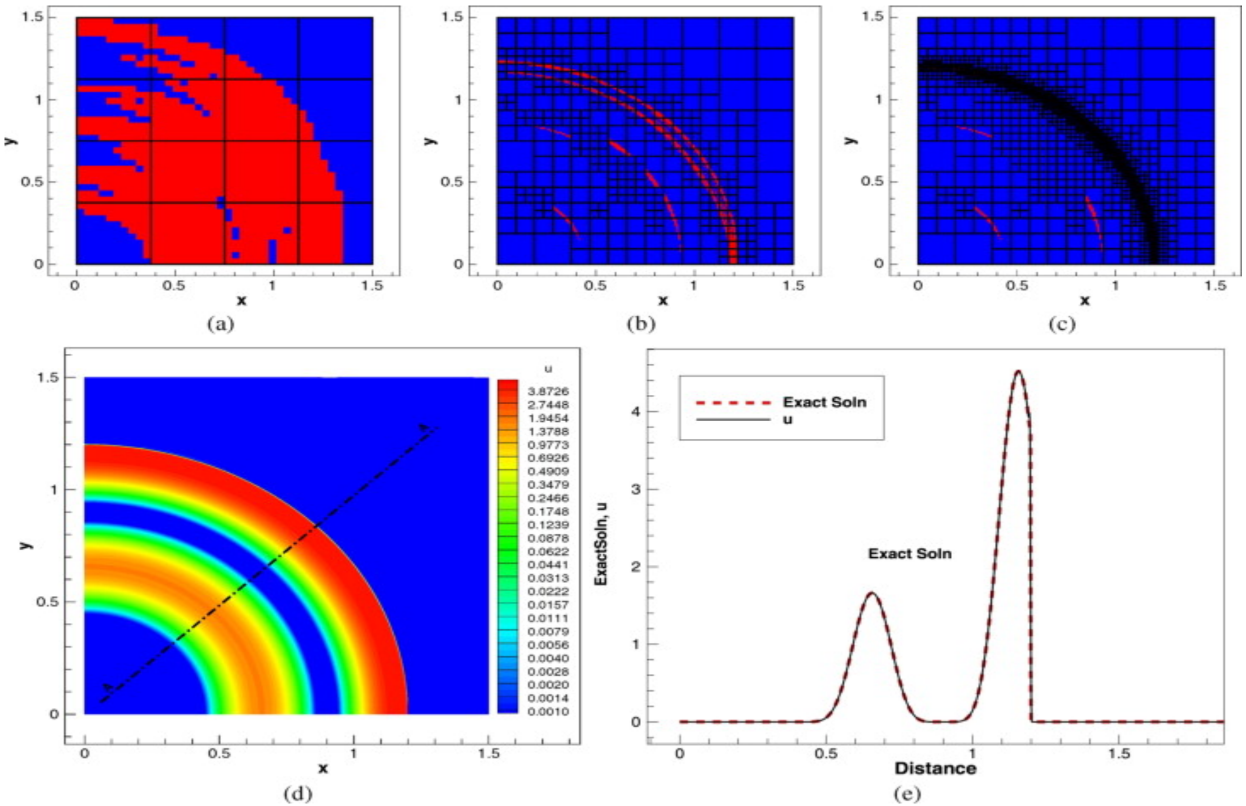


Subject	Predicate	Object
mesh	name	“My Mesh”
sim	timestep	3.0
c0	position	[0.0,0.0]
c1	position	[0.1,0.0]
c2	position	[0.0,0.1]
c3.0	position	[0.1,0.1]
c3.1.0	position	[0.15,0.1]
c3.1.1	position	[0.175,0.1]
c3.1.2	position	[0.125,0.15]
c3.1.3	position	[0.125,0.125]
c3.2	position	[0.1,0.15]

Sample Query: Tracking a Wave thru Time



Sample Query: Tracking a Wave thru Time



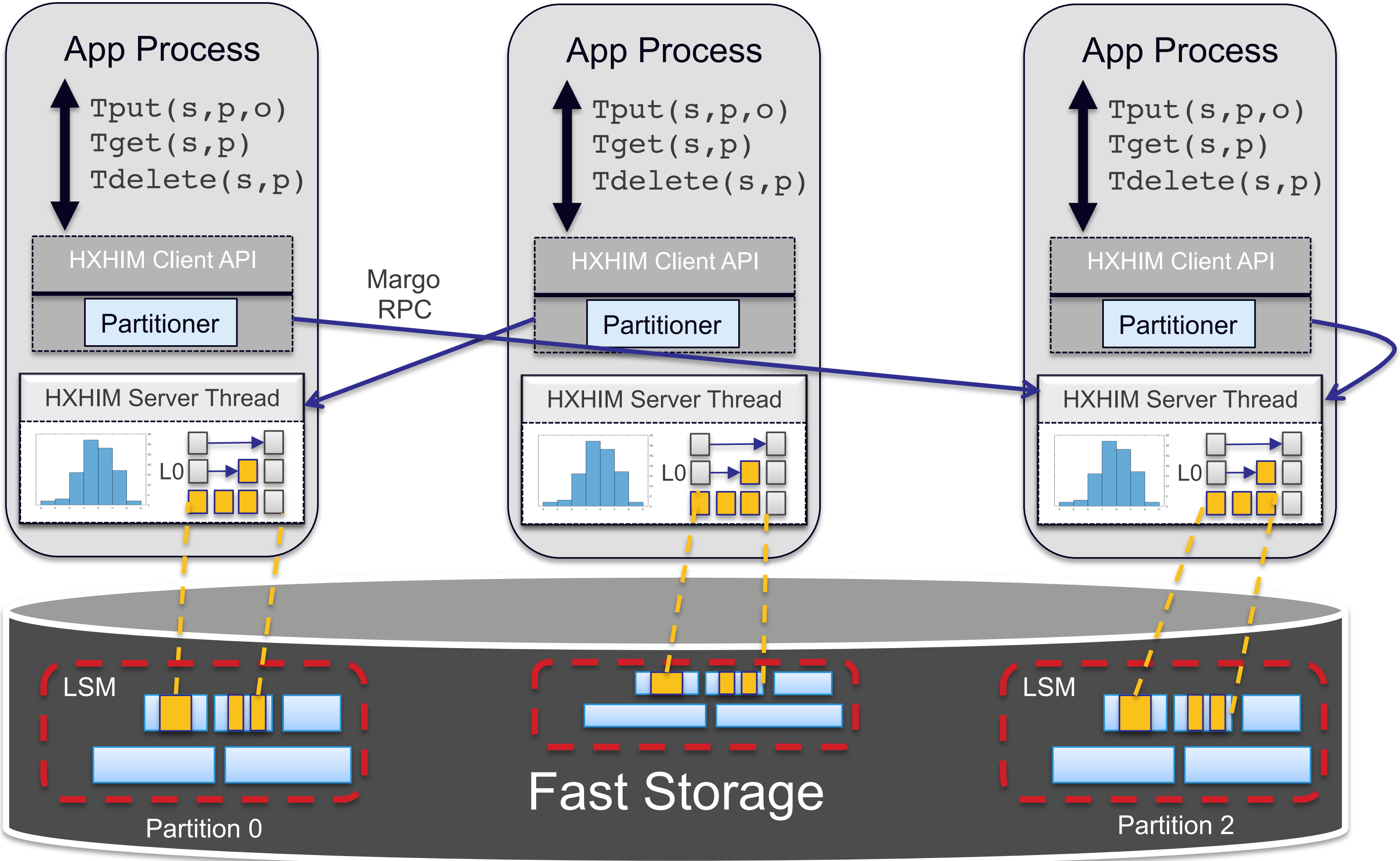
(a) (b) (c) (d) (e)

Requirements

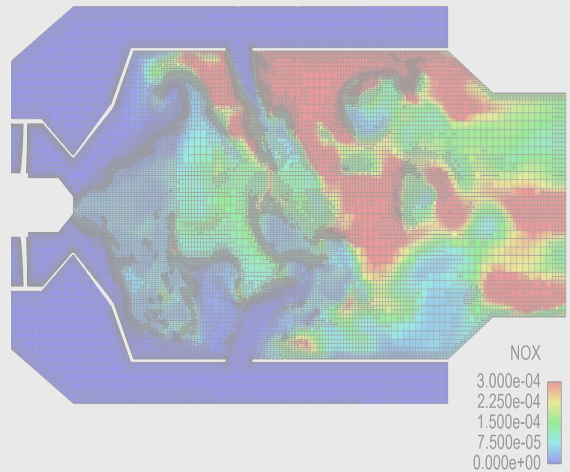
- **A fast multi-dimensional index**
  - Time is discretized separately (indexing not required)
  - Energy and position must both be indexed (and not trivially)
    - Level arrays will result in pointers to pointers (you can often skip the third level of indirection)
    - Energy extrema search is worse 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


App Process
Tput(s,p,o)
Tget(s,p)
Tdelete(s,p)
HXHIM Client API
Partitioner
HXHIM Server Thread
L0
App Process
Tput(s,p,o)
Tget(s,p)
Tdelete(s,p)
HXHIM Client API
Partitioner
HXHIM Server Thread
L0
App Process
Tput(s,p,o)
Tget(s,p)
Tdelete(s,p)
HXHIM Client API
Partitioner
HXHIM Server Thread
L0
Margo RPC
LSM
Partition 0
Fast Storage
LSM
Partition 2

Metadata at Scale



DeltaFS

A File System Service for Simulation Science

HXHIM

Indexing for Scientific Data

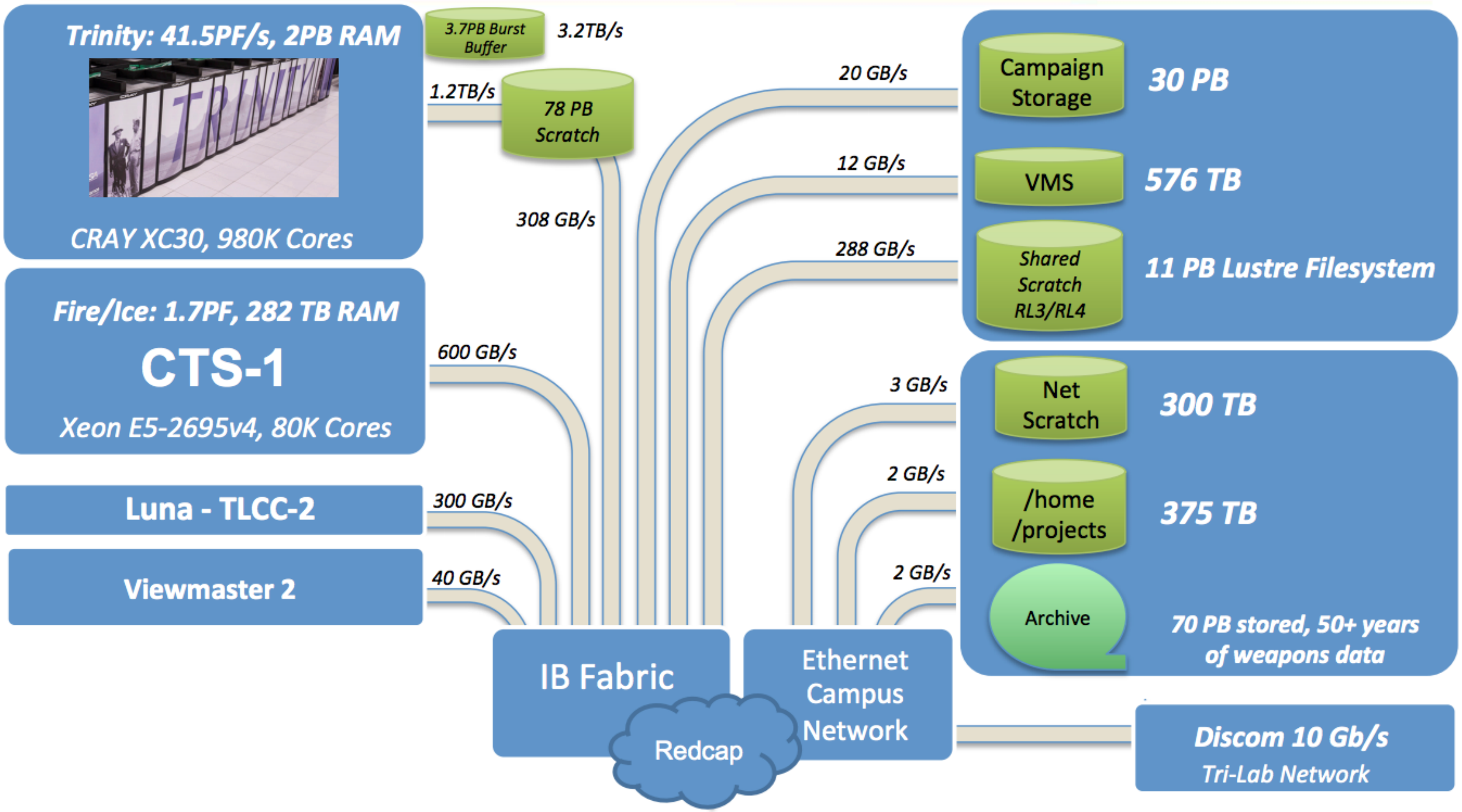
GUFI

Fast Userspace Metadata Query

Motivation

- **Many layers of storage at LANL**
  - By design – users would have us only buying storage if we used 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

LANL Compute/Storage Environment (Secure)



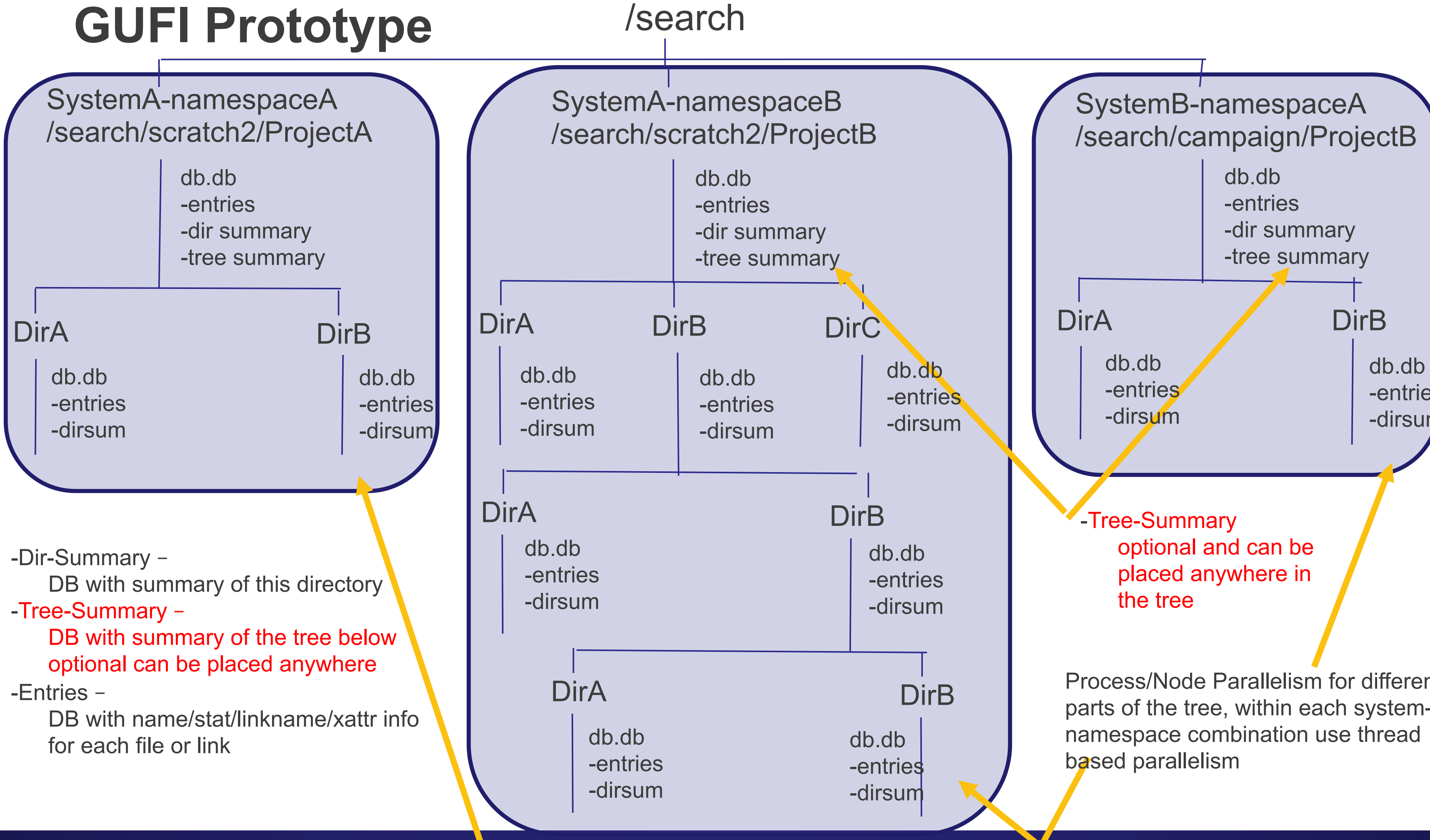
GUFI Goals

- **Unified** index over home, project, scratch, campaign, and archive
- Metadata only with attribute support
- Shared index for **users** and admins
- Parallel search capabilities that are very fast (minutes for billions of files/dirs)
- Can appear as mounted file system where you get a virtual image of your file metadata based on query input
- Full/Incremental update from sources with reasonable update 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

Initial Design Thoughts

- **Why not a flat namespace?**
  - Performance is great, but…
  - Rename high in the tree is terribly costly
  - Security becomes a nightmare if users/admins can access the namespace
- **Leverage things that already work well, reduce required records to scan:**
  - POSIX permissions / tree walk (readdir+)
  - Breadth first search for parallelization
  - Our trees have inherent namespace divisions for parallelism
  - Embedded DBs are fast if not many joins and individual DB size < TB
  - Flash storage is cheap enough to hold everything with order ~10K IOPs each
  - Entries in file system reduce to essentially <dir count> * 3
  - Dense directories reduce footprint dramatically
  - SQL is easily utilized for general queries of attributes

GUFI Prototype



-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

Draft DB Schemas

- **Parent-Inode mapping file "directories-parent-inode directories Inode"**
  - Parent inode is only kept for directories, not for files as that kills rename/move function performance
- **"CREATE TABLE entries(**
  - `name TEXT PRIMARY KEY,` name of file (Not path due to renames)
  - `type TEXT, inode INT,` f for file l for link inode
  - `mode INT,` posix mode bits
  - `nlink INT,` number of links
  - `uid INT, gid INT,` uid and gid
  - `size INT, blksize INT,` size and blocksize
  - `blocks INT,` blocks
  - `atime INT,` access time
  - `mtime INT,` file contents modification time
  - `ctime INT,` metadata change time
  - `linkname TEXT,` if link this is path to link
  - `xattrs TEXT);";` single text string, key/value pairs w/ delimiters

Draft DB Schemas (continued)

"CREATE TABLE summary(	**summary info for this directory**
name TEXT PRIMARY KEY,	name not path due to rename
type TEXT, inode INT,	d for directory inode
mode INT,	posix mode bits
nlink INT,	number of links
uid INT, gid INT,	uid gid
size INT, blksize INT, blocks INT,	size, blocksize, blocks
atime INT, mtime INT, ctime INT,	access time, dir contents mod time, md chg time
linkname TEXT, xattrs TEXT,	if link, path to link, xattrs key/value delimited string
totfiles INT, totlinks INT,	tot files in dir, tot links in dir
minuid INT, maxuid INT, mingid INT, maxgid INT,	min and max uid and gid
minsize INT, maxsize INT,	minimum file size and max file size
totltk INT, totmtk INT, totltm INT,	total number of files lt KB mt KB, lt MB,
totmtm INT, totmtg INT, totmtt INT,	total number of files mt MB mt GB, mt TB
totsize INT,	total bytes in files in dir
minctime INT, maxctime INT,	min max ctime
minmtime INT, maxmtime INT,	min max mtime
minatime INT, maxatime INT,	min max mtime
minblocks INT, maxblocks INT,	min max blocks
totxattr INT,	number of files with xattrs
depth INT);";	depth this directory is in the tree

Draft DB Schemas (continued)

- **"CREATE TABLE treesummary(** **summary info for this tree**
  - totsubdirs INT, tot subdirs in tree
  - maxsubdirfiles INT, maxsubdirlinks INT, maxfiles in a subdir max links in a subdir
  - maxsubdirsize INT, most bytes in any subdir
  - totfiles INT, totlinks INT, tot files in tree, tot links in tree
  - minuid INT, maxuid INT, mingid INT, maxgid INT, min and max uid and gid
  - minsize INT, maxsize INT, minimum file size and max file size
  - totltk INT, totmtk INT, totltm INT, total number of files lt KB mt KB, lt MB,
  - totmtm INT, totmtg INT, totmtt INT, total number of files mt MB mt GB, mt TB
  - totsize INT, total bytes in files in tree
  - minctime INT, maxctime INT, min max ctime
  - minmtime INT, maxmtime INT, min max mtime
  - minatime INT, maxatime INT, min max mtime
  - minblocks INT, maxblocks INT, min max blocks
  - totxattr INT, number of files with xattrs
  - depth INT);"; depth this tree summary is in the tree

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, directorysummar, 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 mid 2014 Macbook (quad core + nvme SSD)
- No tree 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: 2m10s (~1.75M files/sec)
- Search of all files and dirs: 2m19s (~1.63 M entries/sec)
- Search of all files and dirs, but exclude some very large dirs: 1m18s
- Search of all files and dirs, but exclude all < 1000 file directories: 1m59s

- …on a laptop!

Learn more!

- https://github.com/mar-file-system/GUFI

Open Source

BSD License

Partners Welcome