# Los Alamos
NATIONAL LABORATORY
EST.1943

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

NNSA
National Nuclear Security Administration
Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

LA-UR-17-23966

# MarFS

## A Scalable Near-POSIX Name Space over Cloud Objects – Production Experience and New Features


**David Bonnie**

May 2017

National Nuclear Security Administration

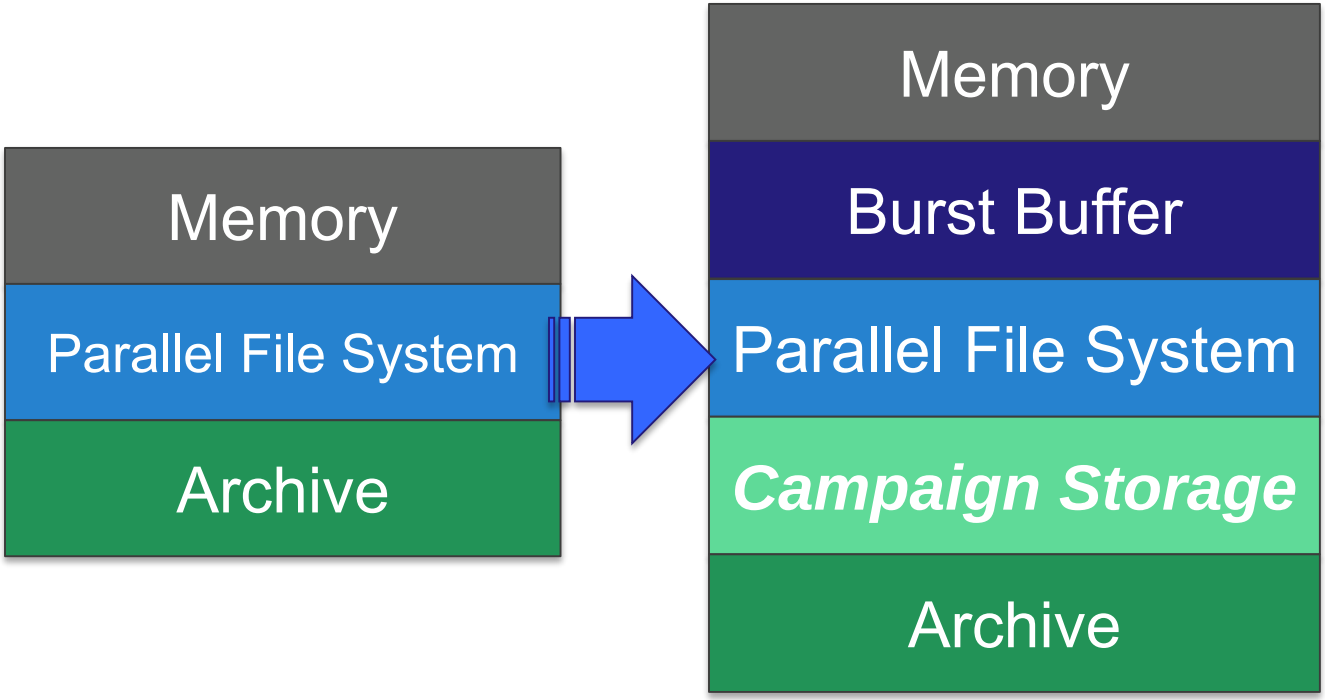
Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

# Overview

- **Summary of MarFS**
  - Motivation
  - Design
- **Production experience**
  - How is it going?
  - What did we do right?
  - What did we do wrong?
- **New features**
  - DAL / MDAL
  - Multi-Component Storage
  - Erasure on tape

# What's the problem?

- Campaign Storage (Trinity+ Version)



# Why existing solutions don't work

- So we need a high capacity, reasonable bandwidth storage tier…
  - Parallel tape is hard and expensive
  - Object solutions?
  - Big POSIX expensive, $$$ hardware
- Existing solutions don't make the right compromises (for us)
  - Petabyte scale files, and bigger
  - Billions of "tiny" files
  - Try to maintain too much of POSIX, this leads to complicated schemes, too many compromises

# What do we really need?

- Large capacity storage, long residency
- No real IOPs requirement for data access
- “Reasonable” bandwidth for streaming
- Metadata / tree / permissions management that’s easy for people and existing applications
- Do we need all of POSIX?

# So what is MarFS?

- MarFS is a melding of the parts of POSIX we (people) like with scale-out object storage technology
  - Object style storage is scalable, economical, safe (via erasure), with simple access methods
  - POSIX namespaces provide people usable storage

- Challenges:
  - Objects disallow update-in-place (efficiently)
  - Access semantics totally different (permissions, structure)
  - Namespace scaling to billions/trillions of files
  - Single files and datasets in the many petabyte+ range

# So what is MarFS?

- What are we restricting?
  - No update in place, period
  - Writes only through data movement tools, not a full VFS interface
    - But, 100% serial writes through FUSE are okay if admins allow – pipes into files
- What are we gaining (through the above)?
  - Nice workloads for the object storage layer
  - Full POSIX metadata read/write access, full data read access

# So what is MarFS?

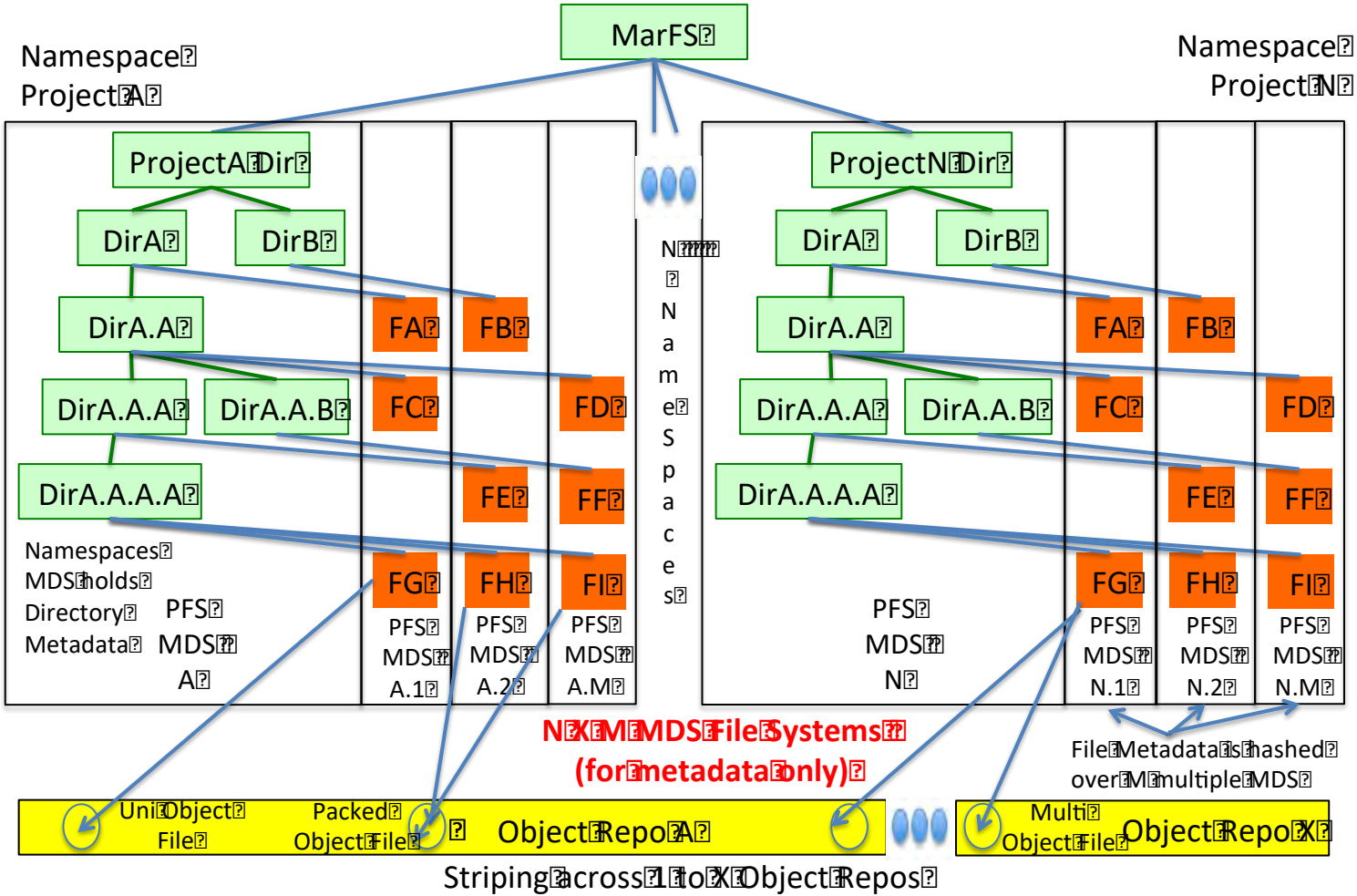
- Stack overview:
  - Library that the below utilize as a common access path
  - Smallish FUSE daemon for interactive metadata manipulation, viewing, small data access, etc
  - Parallel file movement tool (copy/sync/compare/list) - pftool
  - A handful of tools for data management (quotas, trash, packing)
- Metadata stored in at *least* one global POSIX namespace
  - Utilizes standard permissions for security, xattr/sparse file support
- Data stored in at *least* one object/file storage system
  - Very small files packed, very large files split into “nice” size objects

# Scaling basics

- So how does the namespace scale?
  - Up to N-way scaling for individual directories/trees
  - Up to M-way scaling *within* directories for file metadata
  - We're using GPFS (for now), lists are easy, so it's manageable!

- How does the data movement scale?
  - No real limit on number of data storage repositories
  - New data can be striped within a repo
    - Repos can be scaled up and scaled out
    - New repos can be added at any time

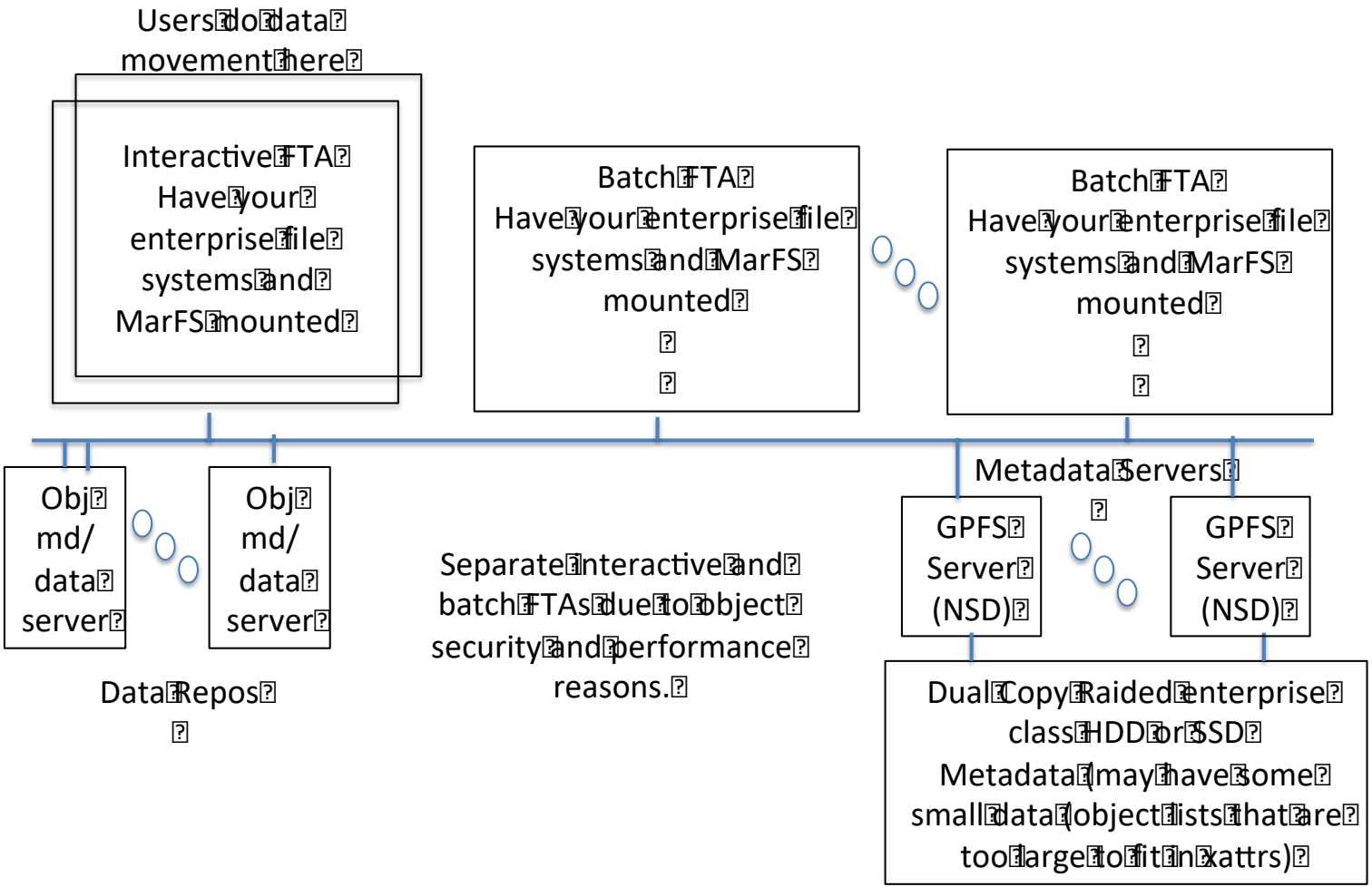
# MarFS Scaling



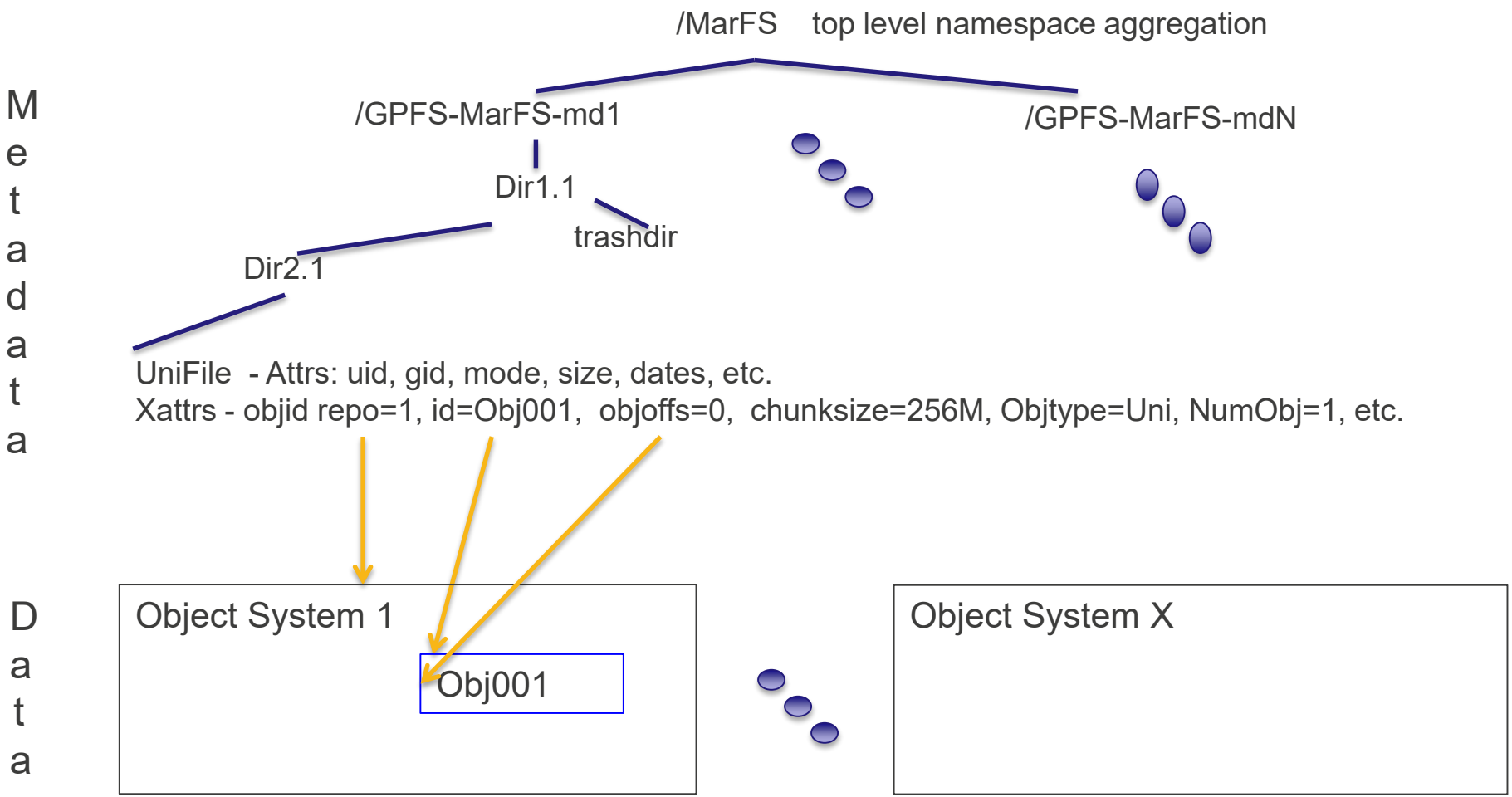
# Metadata scaling demo

- **We built a test harness with MPI to push the limits of the MarFS design**
- **Initial MD scaling runs on Cielo (1.1 PF, ~140,000 cores, ~9000 nodes)**
  - 968 billion files, one single directory
  - 835 million file creations *per second* to metadata repos on each node
  - No cheating! Every create traversed the network from client to server
    - QD=1 for each client, 8 clients per node, 8 servers per node
- **Further testing on Mustang (230 TF, ~38,000 cores, ~1600 nodes)**
  - Large directory readdir tested from 10-50 nodes (sequential and parallel)
  - Peak of 300 million files per second across 50 nodes
  - More than 400X speedup over a single client sequential readdir

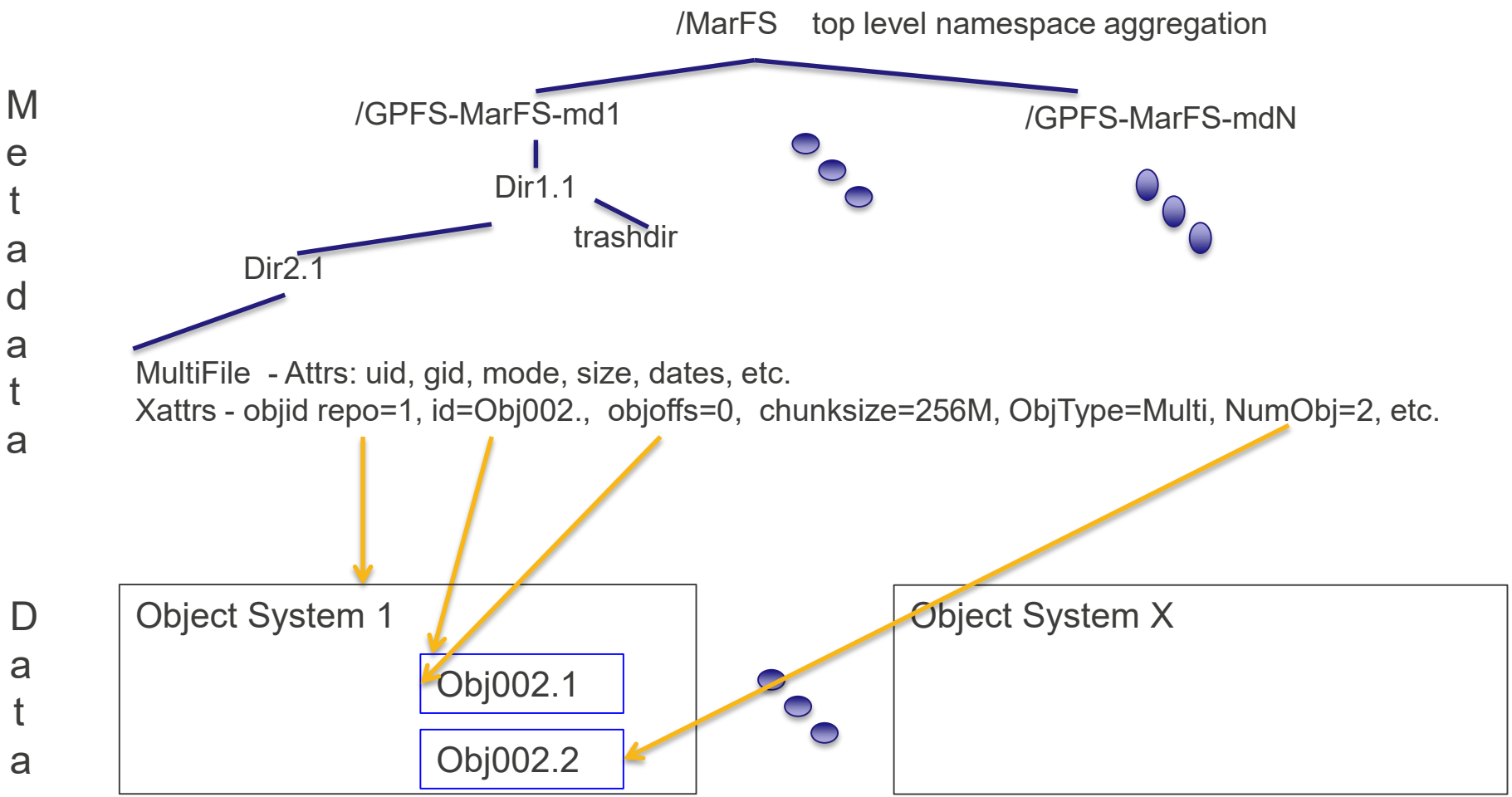
# Simple MarFS Deployment



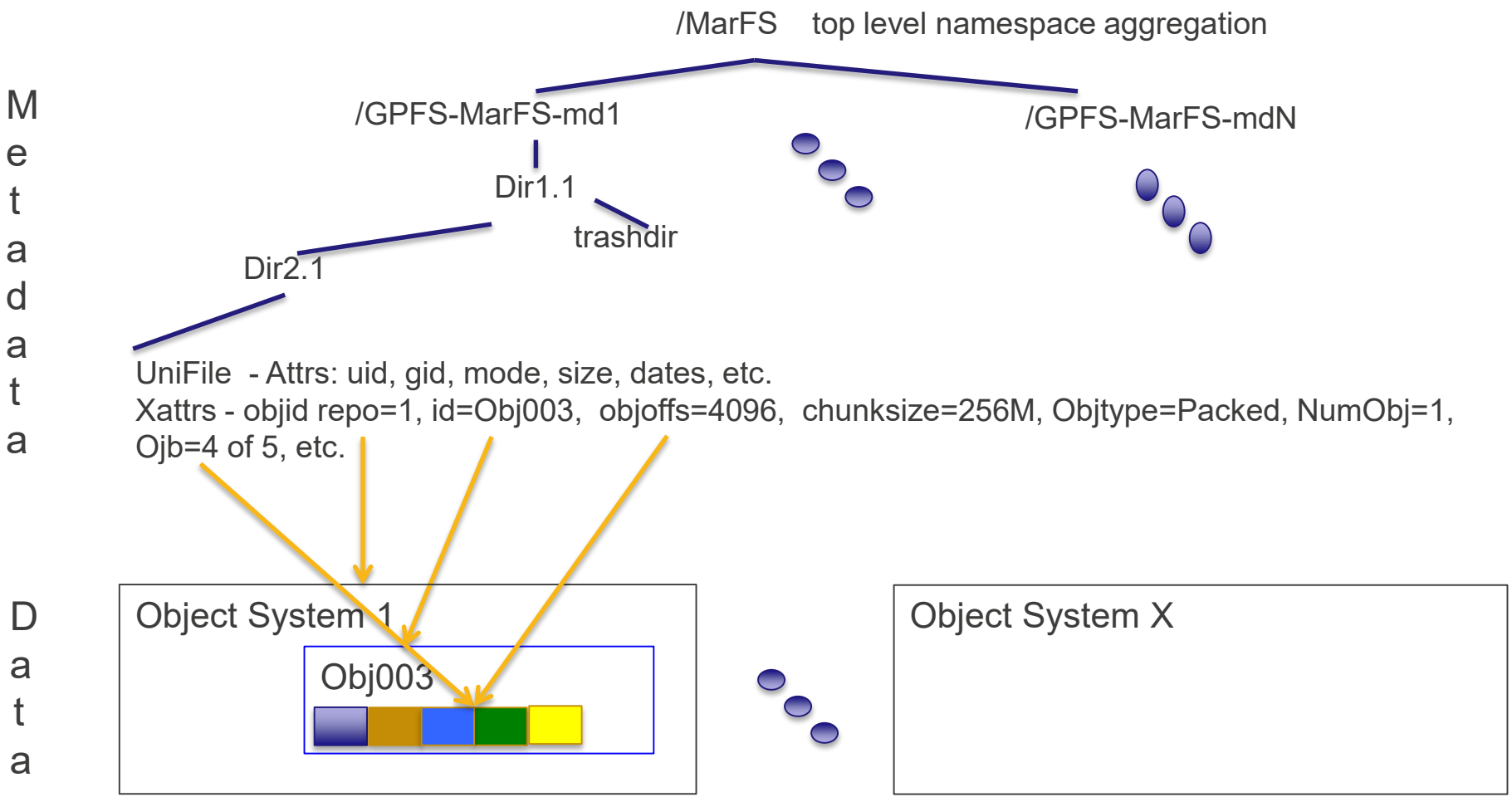
# MarFS Overview Uni-File



# MarFS Overview Multi-File



# MarFS Overview Packed File



# Configuration

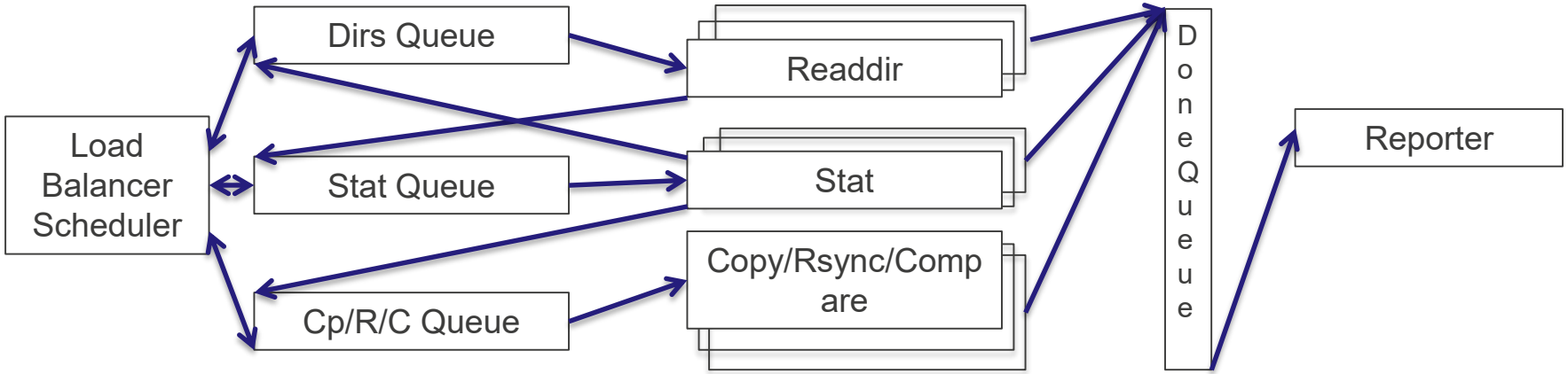
- **Top level MarFS mountpoint**
- **Stanza for every name space/metadata file system you want to bring into MarFS**
  - Describes how metadata is to be handled in this part of the MarFS system
  - Describes where data is to be put for files of various sizes/shapes (into which data repo)
- **Stanza for every Repo - object system/area of object system, or other access method for data**
  - Describes how data is to be stored into this data repo, chunksizes/methods/etc.

# Recoverability

- **Parallel backup of the metadata file system(s) backs up the metadata for MarFS**
  - (METADATA ONLY)
- **Objects are encoded with CREATE TIME info**
  - path, uid, gid, mode, times, MarFS/other xattrs, etc.
- **Parallel backup of object metadata**
  - Object name has some recovery info in it, so just listing and saving the objects/buckets is useful
  - Any other info your object server will allow you to back up

# Pftool

- **A highly parallel copy/rsync/compare/list tool**
- **Walks tree in parallel, copy/rsync/compare in parallel.**
  - Parallel Readdir's, stats, and copy/rsync/compare
  - Dynamic load balancing
  - Restart-ability for large trees or even very large files
  - Repackaging: breaks up big files, coalesces small files
  - To/From NFS/POSIX/PFS/MarFS



# First Production Design

- **22 PB of data storage (Scality RING 20+4 with 6 failure domains of ~500 drives each)**
  - 48 R530 servers, 100 Gbps EDR running IPoIB
  - 48 Seagate 5U84 enclosures, 8 TB Seagate Archive SMR drives (60 drives used per enclosure)
- **6 TB of metadata storage (GPFS, triple replication + backup)**
  - 3 R530 servers, 100 Gbps EDR, GPFS using native RDMA
- **A small cluster of File Transfer Agents (mount all external resources)**
  - 30 R530 servers, 100 Gbps EDR, IPoIB for Scality access, RDMA Verbs for GPFS access
- **~24 GB/s for multi-user workloads**

# Production Experience

- **Where are we now?**
  - Over the course of 6 months, users stored ~2 PiB of data.
    - We attribute this to both PFS stability and lack of outreach on our part
  - Admin team is essentially 1.3 people (2 partial humans, lots of scripts)
  - Migration from Scality -> Multi-Component Repositories recently completed, painfully
    - We learned a lot from a full-scale stress test of MC repos (edge cases)
    - We also learned that the key to stability with a skeleton crew is monitoring, monitoring, MONITORING – hindsight is 20/20, but silent daemon crashes are *bad*

# Production Experience

- **So what did we do correctly initially?**
  - Metadata stability / scalability was great
    - Zero metadata related problems
    - Metadata was faster than the storage could accommodate (always data limited)
  - No performance issues with very consistent speeds under mixed workloads
    - 3-5 GB/s per user, 24 GB/s aggregate with multiple transfers in flight
  - Very little downtime in normal operation
    - Only pre-scheduled DSTs and one unforeseen downtime due to hardware failure
  - Quotas! Fast enough scans through GPFS to react to users exceeding allocation.
    - ~30 mins runtime at 150 million files, run every 2 hours.

# Production Experience

- **…and what did we do wrong?**
  - Monitoring and analysis of logs
    - Daemons would crash, but due to our diskless design, we didn't see them directly – they manifested as retries via pftool and a handful of log messages
    - Getting Scality running diskless was a bit of a hack, and veiled a few issues we normally would have caught (like the daemons crashing)
    - Single point of failure in the "logging" box for Scality and the diskless "master" node
  - Underestimated the "badness" of some user workloads
    - Some users had badly formed data – many tens of millions of < 1 MB files
    - Packing of files into objects only compacted 128:1 (so 128 4 KB files resulted in a 512 KB PUT).
  - User education (or lack thereof)
    - We did not communicate the purpose or existence of Campaign Storage well enough to users, so uptake was slower than we would have liked

# Current / Second Production Design

- **30 PB pre-compression data storage (Multi-Component using NFS/ZFS)**
  - 48 R530 servers, 100 Gbps EDR running IPoIB
  - 48 Seagate 5U84 enclosures, 8 TB Seagate Archive SMR drives (84 drives used per enclosure)
  - 48 more enclosures (with 30 PB more storage) ready to deploy
- **6 TB of metadata storage (GPFS, triple replication + backup)**
  - 3 R530 servers, 100 Gbps EDR, GPFS using native RDMA
- **A small cluster of File Transfer Agents (mount all external resources)**
  - 30 R530 servers, 100 Gbps EDR, IPoIB for NFS access, RDMA Verbs for GPFS access
- **~24 GB/s read (single or multiple user)**
- **~12 GB/s write (multiple user)**

# Recent Enhancements

- **Data Abstraction Layer (DAL)**
- **Meta-Data Abstraction Layer (MDAL)**
- **Multi-component Repos**
- **Parity over multiple ZFS pools**
- **Packing in pftool (currently 128:1, exploring 10K:1)**
  - Transparent to users
  - Saves data layer IOPs, improves bandwidth, reduces data layer metadata load / size

# DAL / MDAL

- **Plug and play layer for both data and metadata storage**
  - DAL
    - Logically separate all functions (easy, since we designed for REST)
      - Read sequential
      - Read offset
      - Write sequential
      - Delete
  - MDAL
    - Same concept as DAL, abstract all needed functions
  - Very thin shim to write for each new data repository type
    - No reliance on any vendor-specific functions in the main code base
    - Allows for interesting new storage layers (key value store for metadata possible)

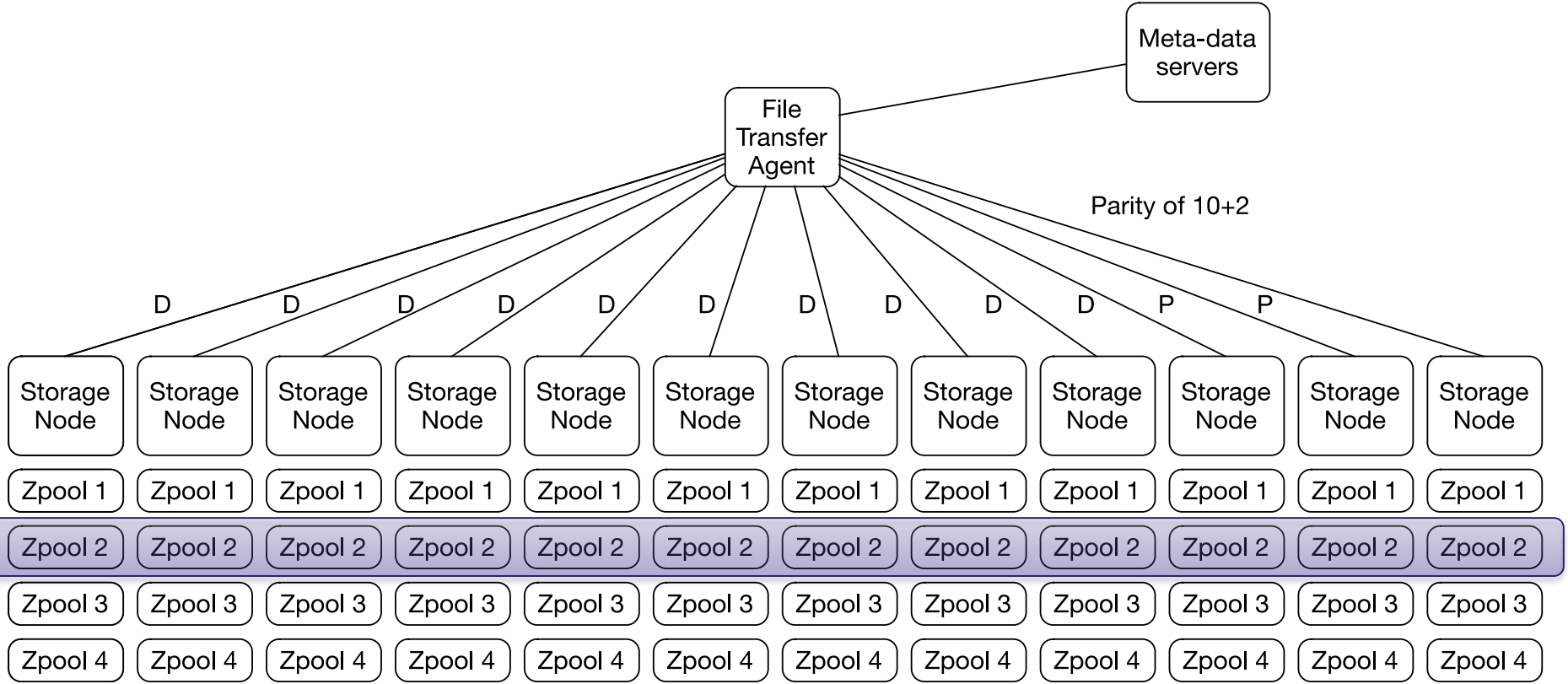
# Multi-Component Repos

- **Why?**
  - As we scale up/out the storage tier, our failure domains grow (since datasets span the entire system, millions of objects, petabyte scale files)
  - MC repos logically separate smaller segments of storage into units to stripe over
  - Additionally, we can add an extra layer of protection!
  - Provides scalable way to increase storage capacity and bandwidth
- **What?**
  - Stripe over components of some base size and bandwidth (think small pools of storage in a greater sea)
  - Add additional protection through erasured stripes (i.e. 10+2 over 12 components)

# Parity over multiple ZFS pools

- **Our first instantiation of Multi-Component Repositories**
  - Take what we already know and love (ZFS) and stripe over sets of ZFS'es
  - Gain all the niceties of ZFS
    - Compression
    - Snapshots
    - CoW friendliness to shingled HDDs (SMR)
    - RAIDZ3 for local redundancy
    - Broad ecosystem with local expertise and community development
  - Rebuilds mostly happen local at the bottom layer, erasure over multiple racks, engage only when an entire lower layer is in danger or destroyed
    - Leverage layout formula for easy generation of missing stripes (any existing stripe gives enough metadata to regenerate any other stripe)
    - Scalable rebuild across components is in design stage

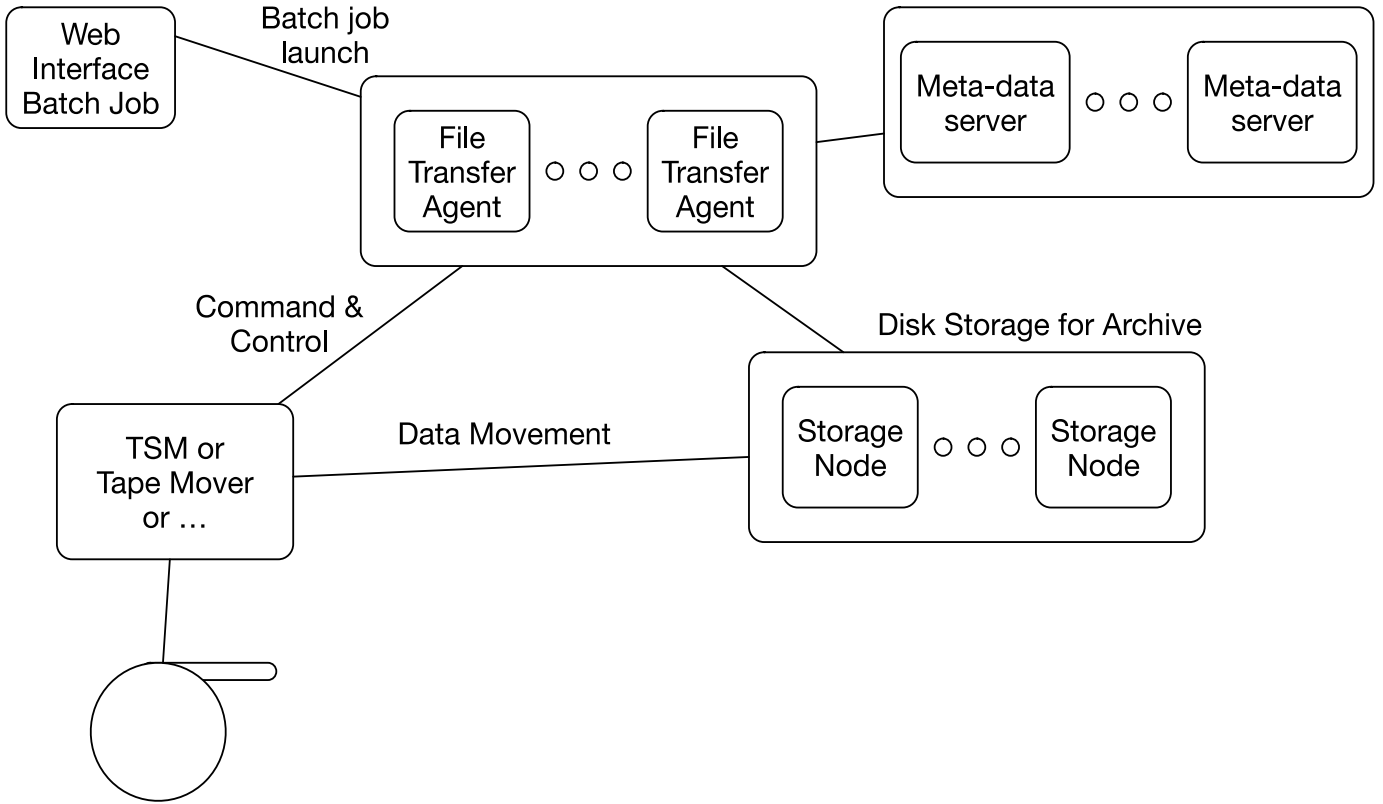
# Parity over multiple ZFS pools



# Looking forward

- **How can we utilize this architecture for longer-term storage?**
  - Disk is great, but tape is 100% offline and is much more “cold”
  - Can we leverage the same concepts for tape?

# MarFS “Archive”



# MarFS Archive Details

- Data is stored to Campaign storage nodes which act as the Landing Zone (requires a select amount of PMR drives for this purpose)
- We may select a different parity scheme for this Landing Zone vs. the current 10+2 over 17+3 so that we can match the solution writing to tape (think 5+1 or similar on tape)
- Will need to develop a batch processor for handling the Landing Zone
- Storage nodes act as TSM clients (or other solution if selected) and potentially TSM servers (allowing shared memory data transfer)
- Object/file manifests stored along with data on tape
- Will need additional logging of data movement and changes if needed/desired to feature-match other archive solutions

# Learn more!

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

## Open Source
## BSD License
## Partners Welcome



# Questions?