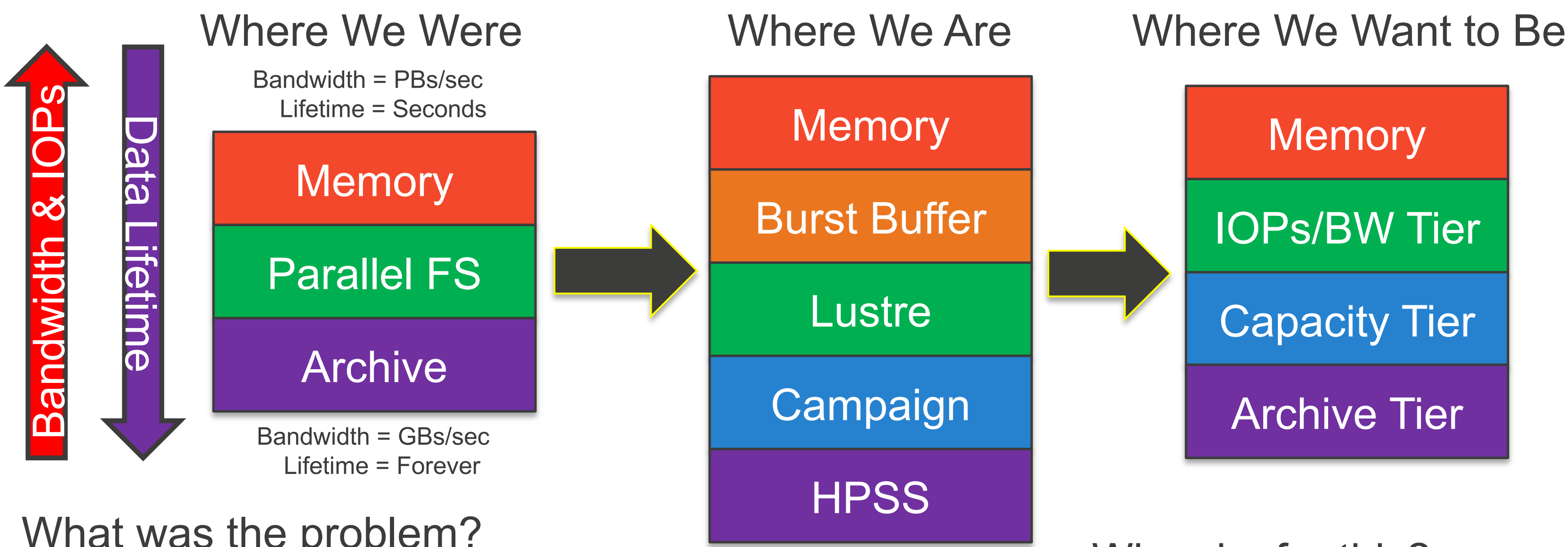
MarFS as a Multi-Level Erasure Archive

Garrett Ransom

May 24th, 2023

Managed by Triad National Security, LLC for the U.S. Department of Energy's NNSA

LANL's HPC Storage – Past / Present / Future



What was the problem?
Parallel FS was doing too much:

- Low Latency
- High Bandwidth
- High Capacity
- Long Residency

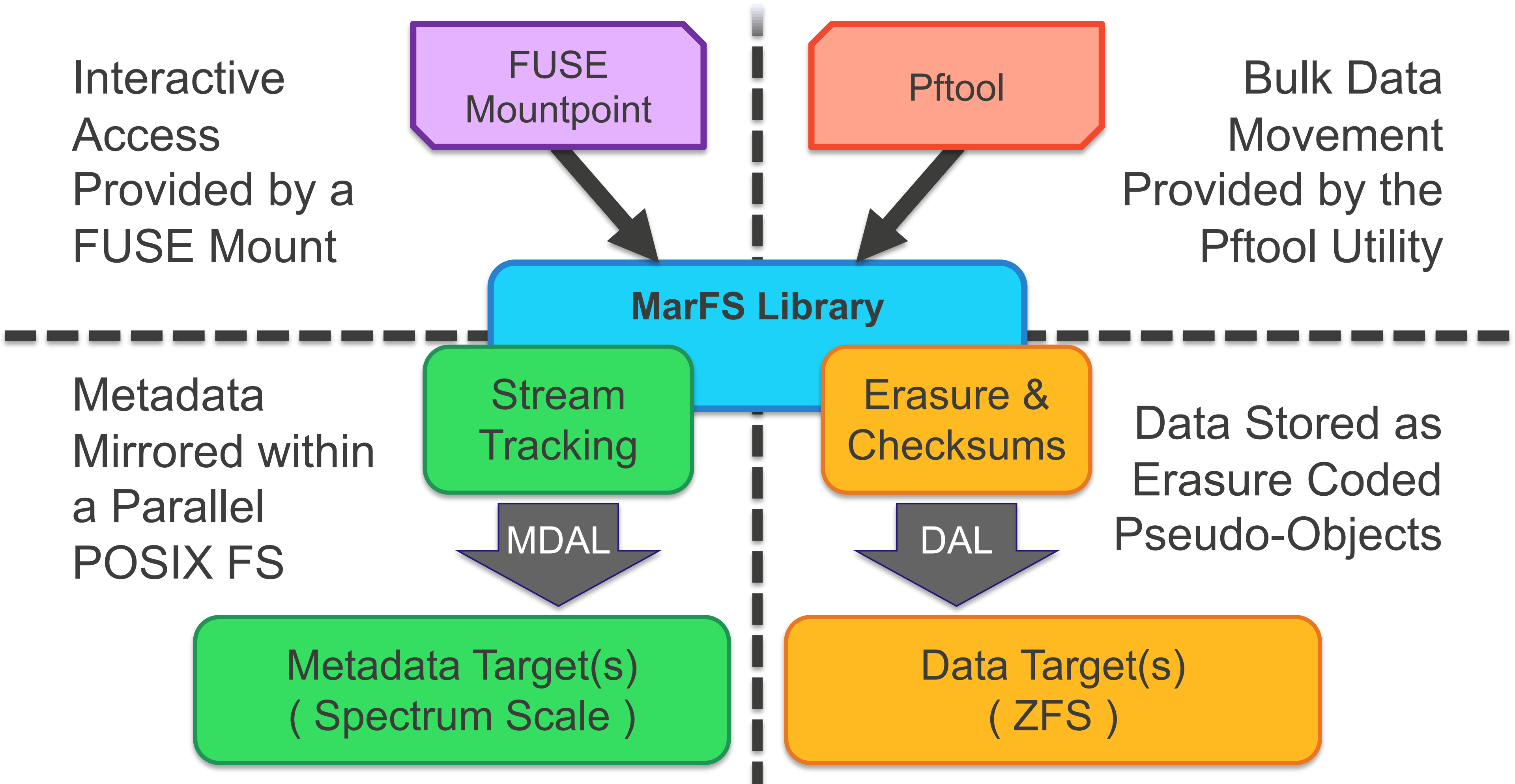
Why aim for this?
Trying to **avoid**:

- Buying flash for capacity
- Buying tape for bandwidth
- Keeping bulk data forever

Campaign Storage – Implementation Goals

- **High Capacity**
  - As Lustre systems target flash media, they will shrink
  - Campaign Storage must absorb that capacity pressure
- **Long Data Residency**
  - NOT permanent storage
  - Roughly 6 months to a few years
- **High Throughput**
  - NOT low latency
- **Dataset Agnostic**
  - NOT workload agnostic
  - Performant for massive datasets with widely varied file counts

Implementation Result – MarFS



MarFS – Design Tradeoffs

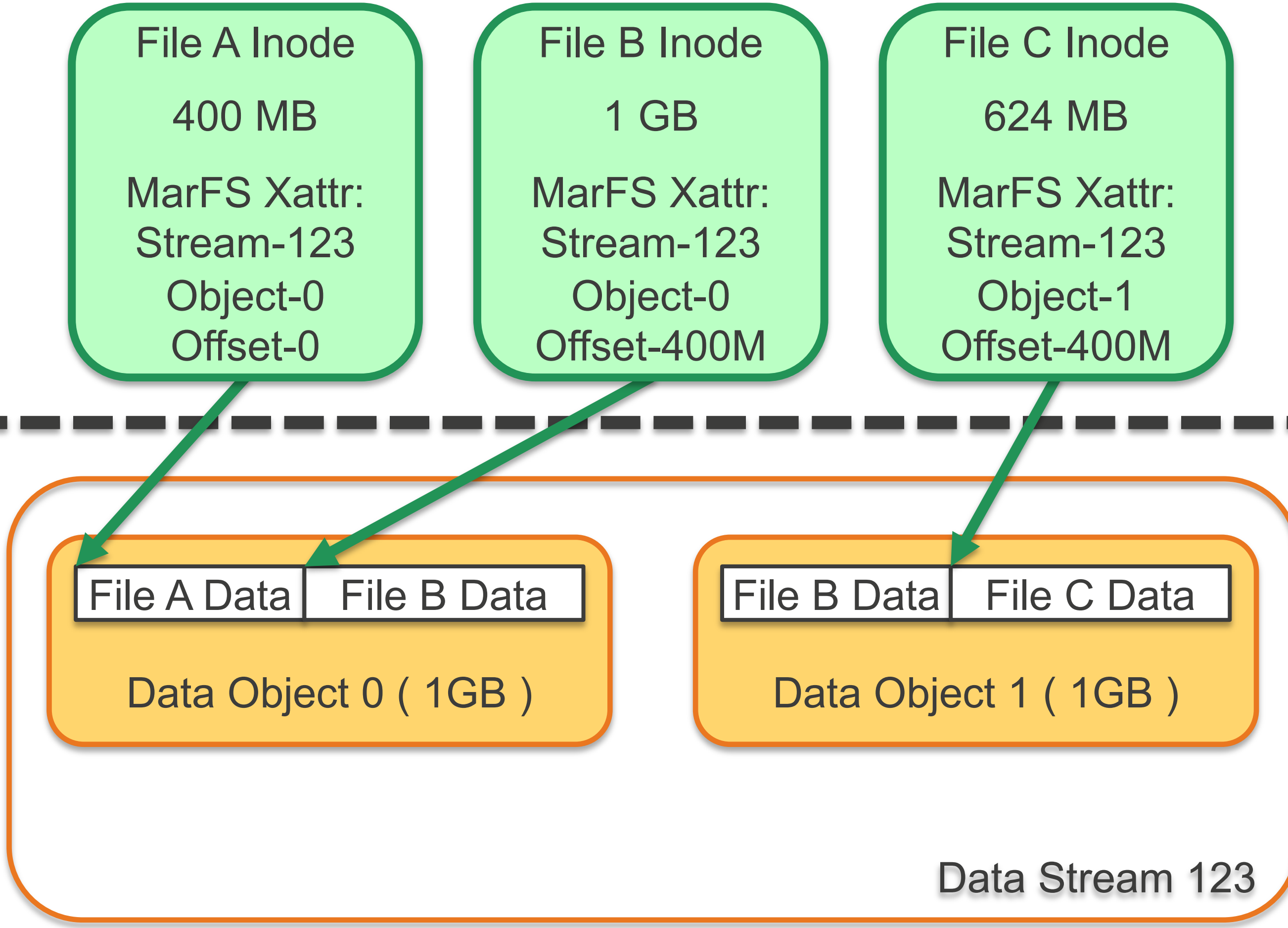
- **Simplicity of Design**
  - Heavily leaning on existing block-storage solutions
  - More of a ‘compatibility layer’ than a scratch-made FS
  - DAL (Data Abstraction Layer) and MDAL (Metadata Abstraction Layer) provide a degree of insulation from the underlying storage target design
  - Extra fault tolerance layered on top of the data storage targets
  - Extra information embedded into the metadata targets

- **Streaming Workload Optimized**
  - Pftool gives massive parallelism for bulk data movement
  - Large files are ‘chunked’ into simultaneous strided writes
  - Small files are ‘packed’ into large single writes
  - No update in place
  - No writing through the FUSE mountpoint

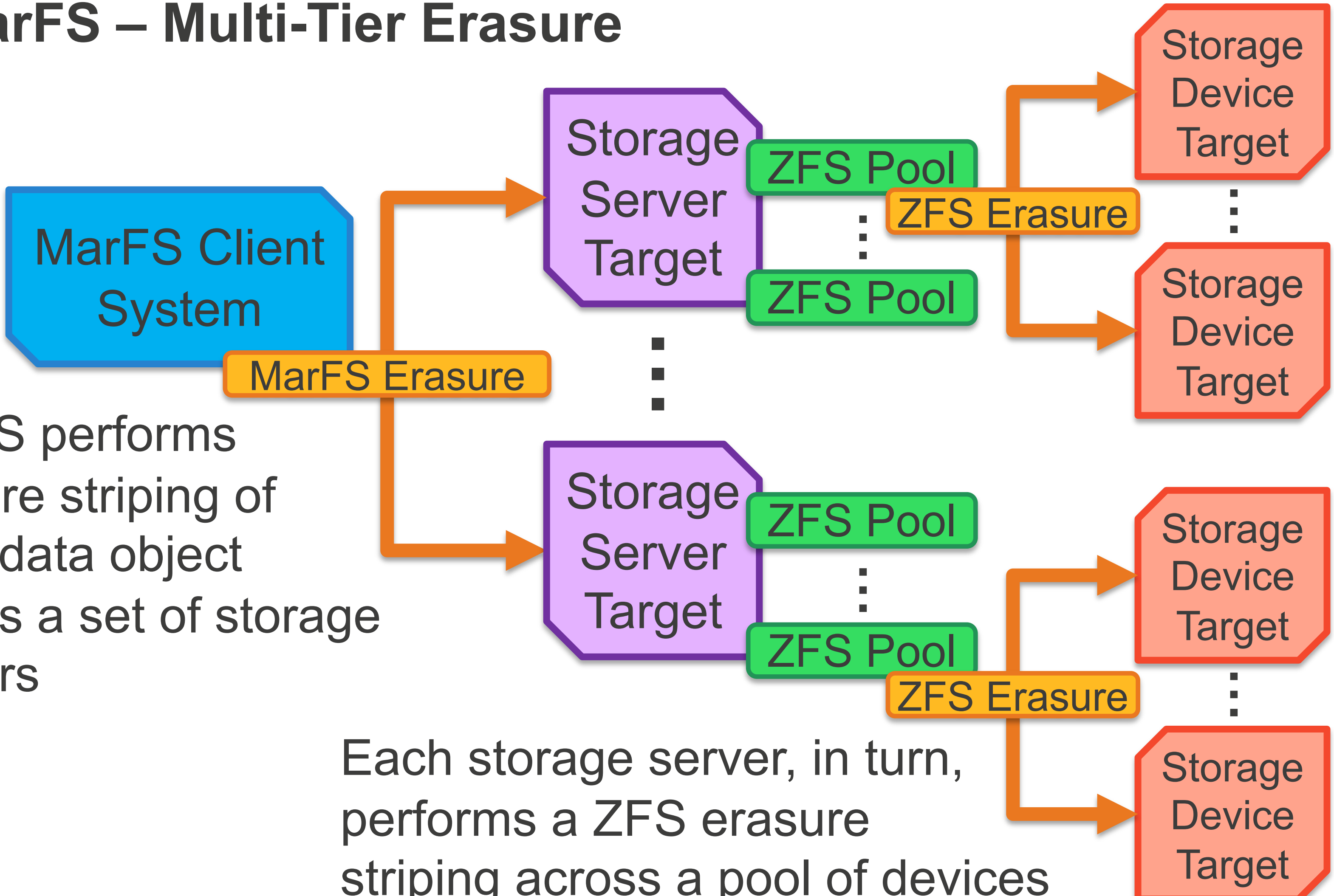
MarFS – Storage Structure

Files organized into ‘streams’, with data object references stored as extended attributes

The data content of a ‘stream’ of files is split across a sequence of consistently sized objects



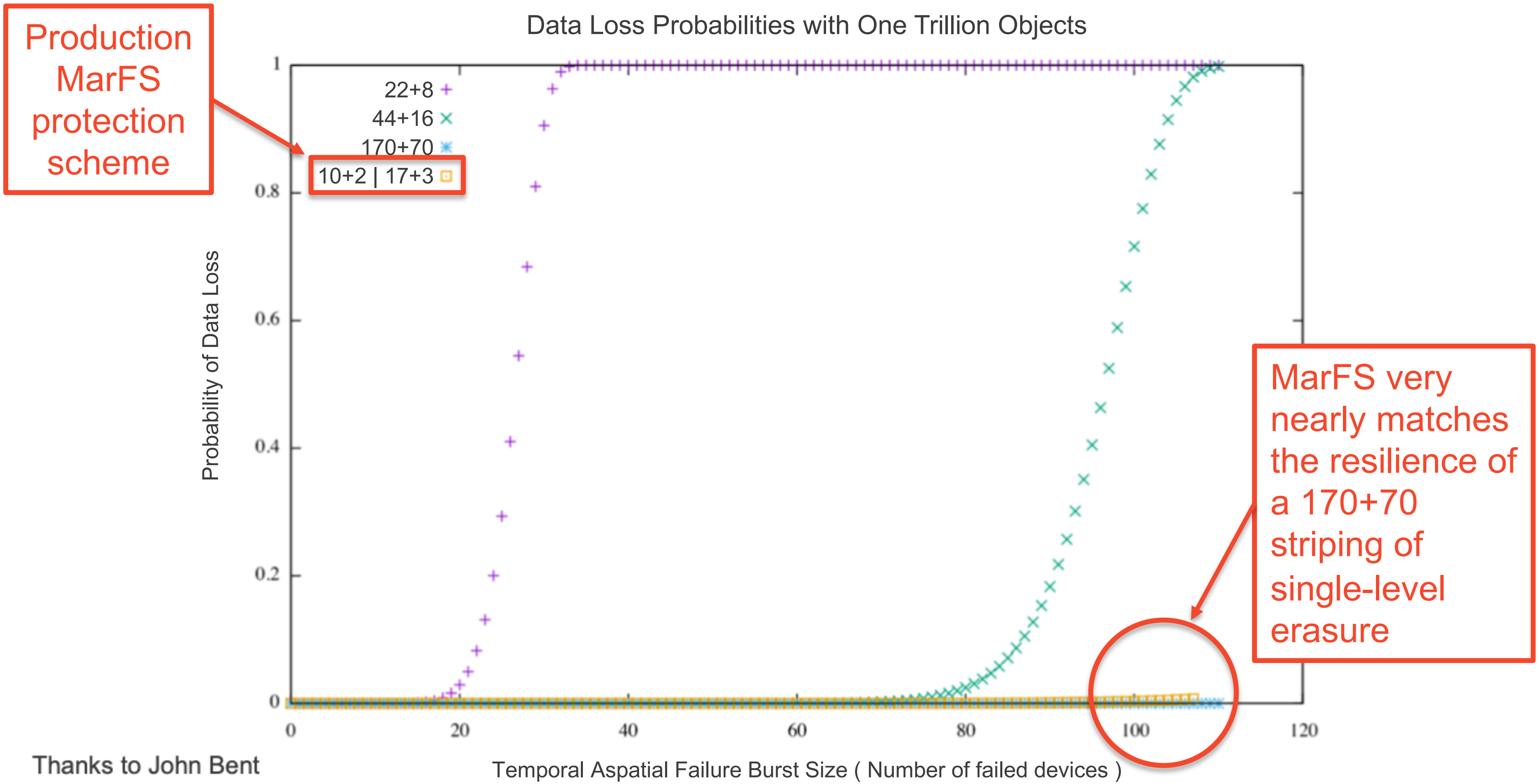
MarFS – Multi-Tier Erasure



MarFS performs erasure striping of each data object across a set of storage servers

Each storage server, in turn, performs a ZFS erasure striping across a pool of devices

Multi-Level vs. Single Level Erasure



Thanks to John Bent

MarFS Multi-Level Erasure Tradeoffs

- Increased Resilience for Less Overhead
  - Pushes down probability of initial data loss while being less computationally intensive to generate than a comparable single-tier design
  - However, if data loss does occur, the ‘blast radius’ will be larger
  - For HPC at LANL, avoiding any data loss at all is vastly preferable to minimizing the volume of data loss

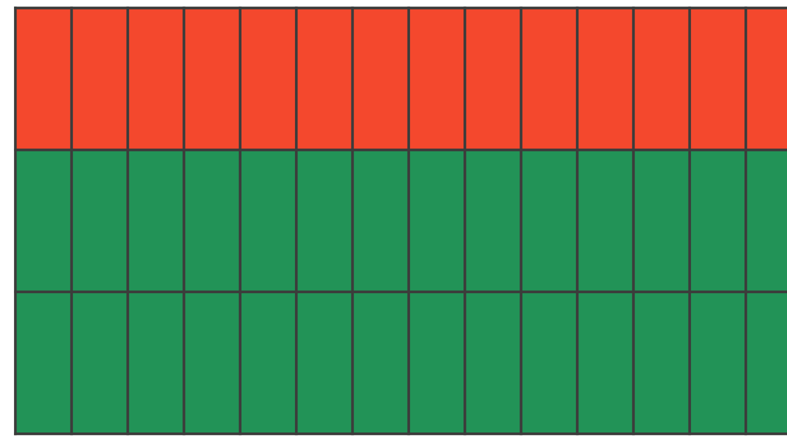
- Simplified Architecture
  - Compared to alternatives like LRC, multi-level erasure coding (MLEC) is comparatively simple to implement
  - The division of erasure computation between client and server is a natural fit for our system architecture
  - MLEC provides a trivial means of achieving cross-server and even cross-rack redundancy

MarFS Near Disaster Example

In April of this year, the primary MarFS production system at LANL suffered a temporary failure of two rows of disk in a JBOD enclosure

JBOD Drawer1 – 14 Disks Offline

JBOD Drawer2 – 14 Disks Offline

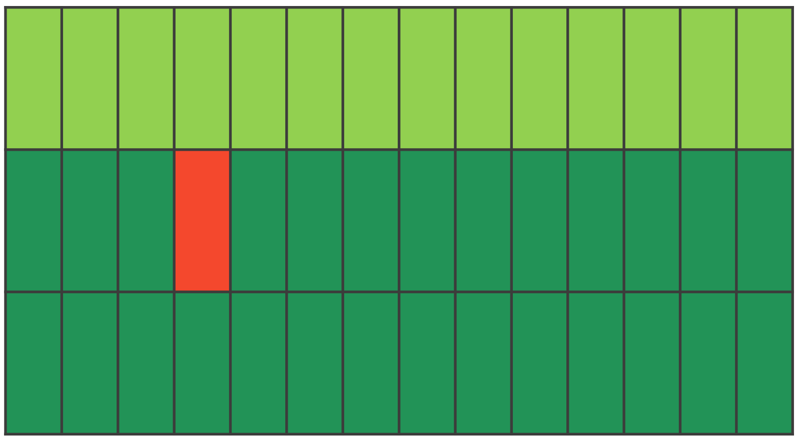


Due to the heavily conservative layout of the resident ZFS pools (raidz2 across only 6 drives), the pools continued to operate through the night without interruption

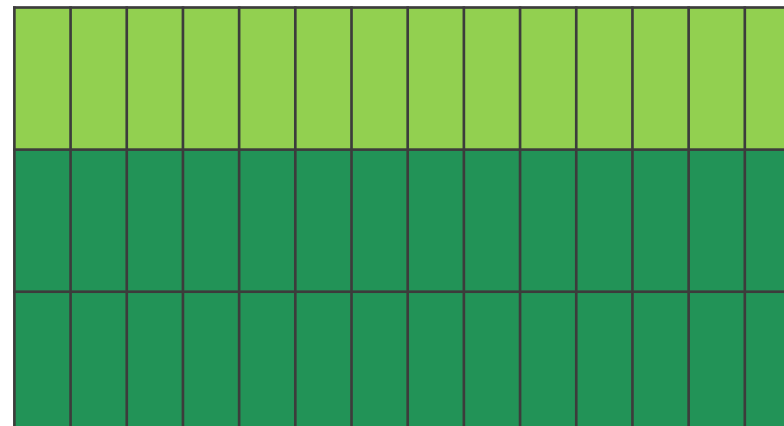
MarFS Near Disaster Example

The following morning, in an effort to recover these disks, we shutdown the storage server and replaced the problem JBOD enclosure

JBOD Drawer1 – 14 Disks Recovered
1 Disk Offline



JBOD Drawer2 – 14 Disks Recovered

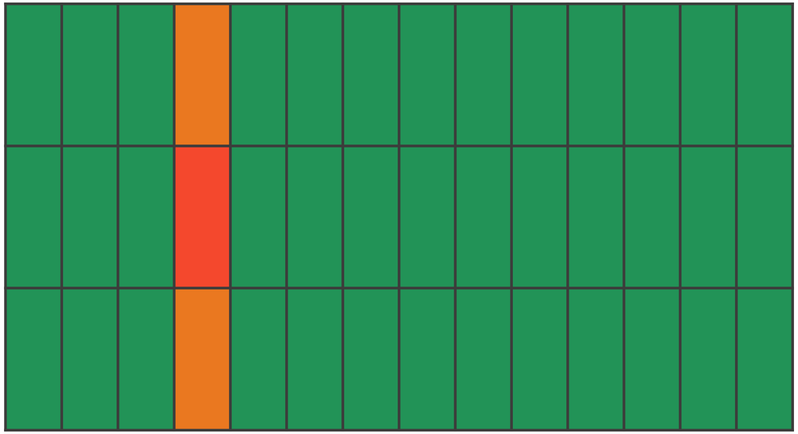


This process did restore the unavailable drives.
However, a single additional drive failed unrecoverably.

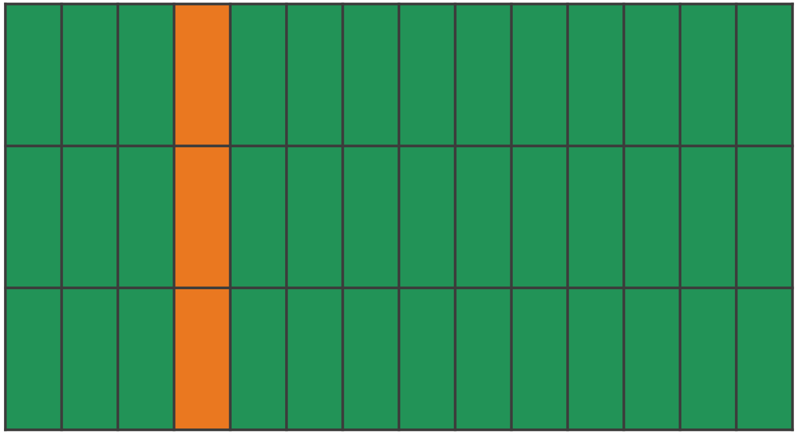
MarFS Near Disaster Example

For the majority of zpools, recovery of the temporarily offline drives was near instant (no data had been written to them overnight)

JBOD Drawer1 – 2 Disks Unimportable
1 Disk Offline



JBOD Drawer2 – 3 Disks Unimportable



However, a single zpool was rendered completely unrecoverable by the newly failed drive, despite only 1 disk now being 'truly' unavailable

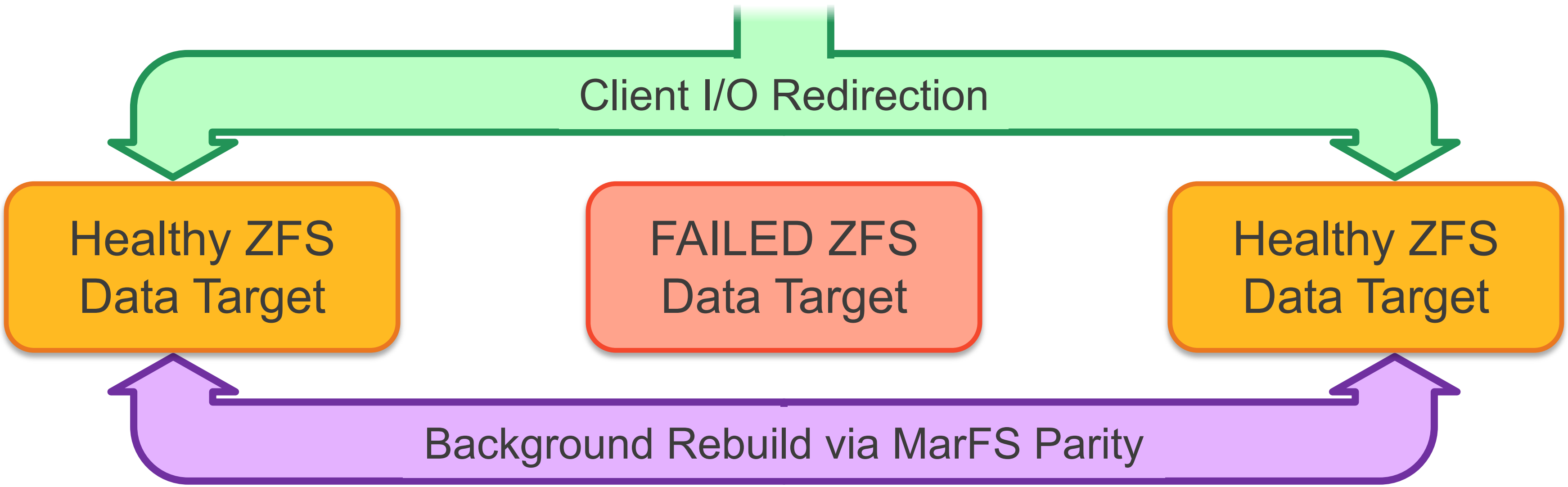
MarFS Near Disaster Example

- Why was this zpool unrecoverable?
  - Due to the previous temporary disk outages, two of the disks in this pool were missing recent ZFS transactions (though empty), rendering them unusable during pool import
  - The single additional drive failure meant that a total of 3 devices from the 6+2 zpool were unusable, making import impossible

- How much data was at risk?
  - Actual resident data on the failed zpool was fairly low (~20TB)
  - This is misleading though, as this data makes up essential components of much larger scientific datasets
  - The loss of any portion of these datasets renders the entire set unusable, potentially setting our scientists back by months or years of work on our largest compute clusters

MarFS Near Disaster – Recovery

Fortunately, the tiered erasure setup of MarFS meant we could simply engage the upper level parity to recover this failed pool



By redirecting data access from the failed zpool to other available pools, we were able to rebuild the lost data while triage of the unrecoverable pool continued

MarFS Near Disaster – Lessons Learned

- Extra MarFS resilience was worthwhile
  - Even instructing ZFS to throw out failed transactions, the import of the problem zpool never succeeded (ran for around 3 days before failing)
  - Without the upper tier of parity, we would have lost data

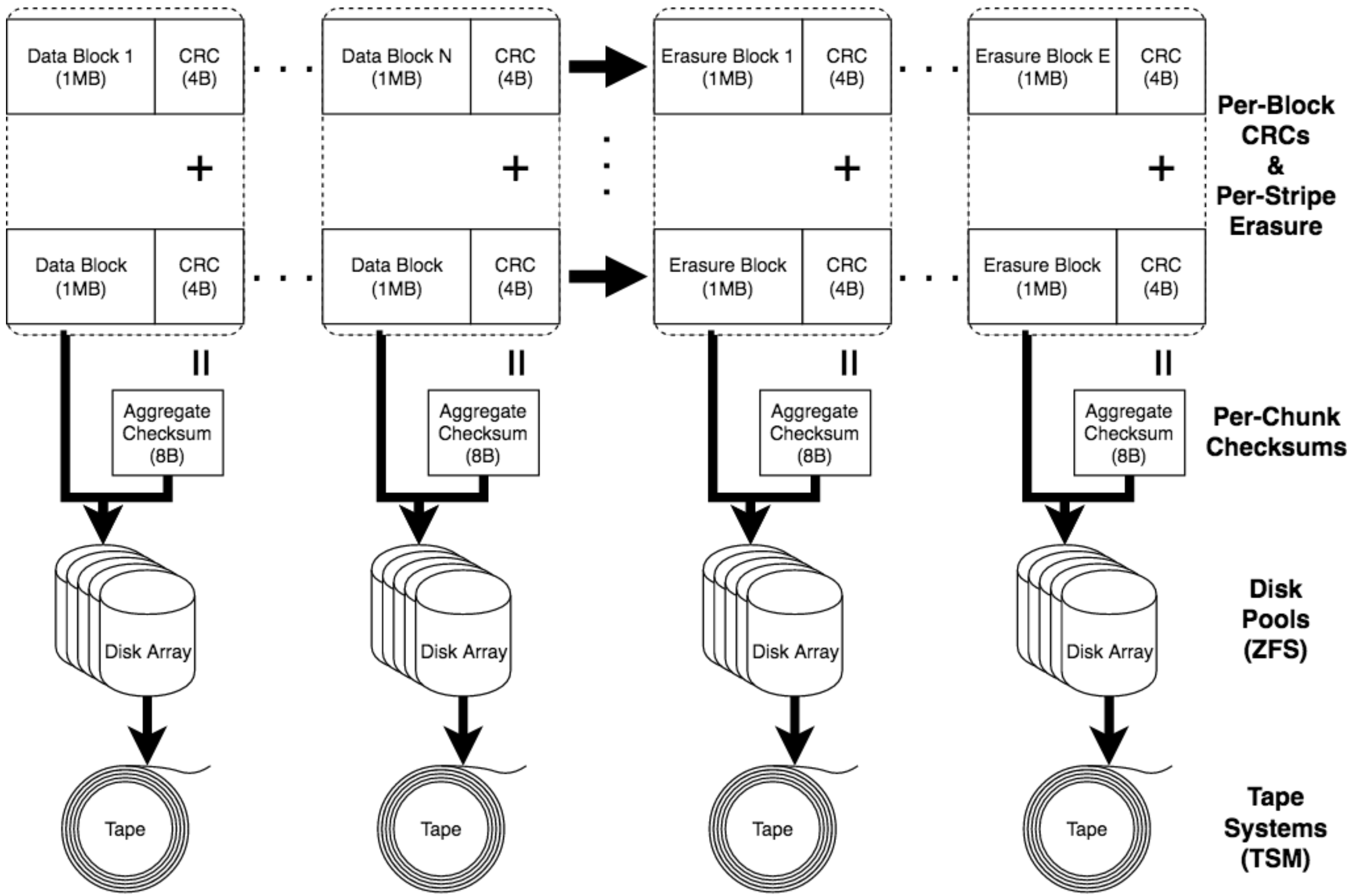
- Transparent designs give admins problem solving flexibility
  - At least in this version of MarFS, no utility existed to fully automate this client I/O redirection + distributed rebuild process
  - However, some simple scripting resulted in a workable solution in < 1hr

- Consequences of loss can exceed the obvious
  - However, the nature of LANL's large scientific datasets means that even small quantities of unrecoverable data can have magnified importance
  - More so than merely MTTDL calculations, it was driving down the probability of *any* data loss at all which led to this system design

MarFS as a Long-Term Archive

- MarFS already offers…
  - Data validation via CRCs
  - Cross-server failure protection
  - Consistently sized objects via packing/chunking
  - Asynchronous garbage collection of deleted objects
- These sound like useful archive features
  - We can easily adapt the existing MarFS design to incorporate archival media

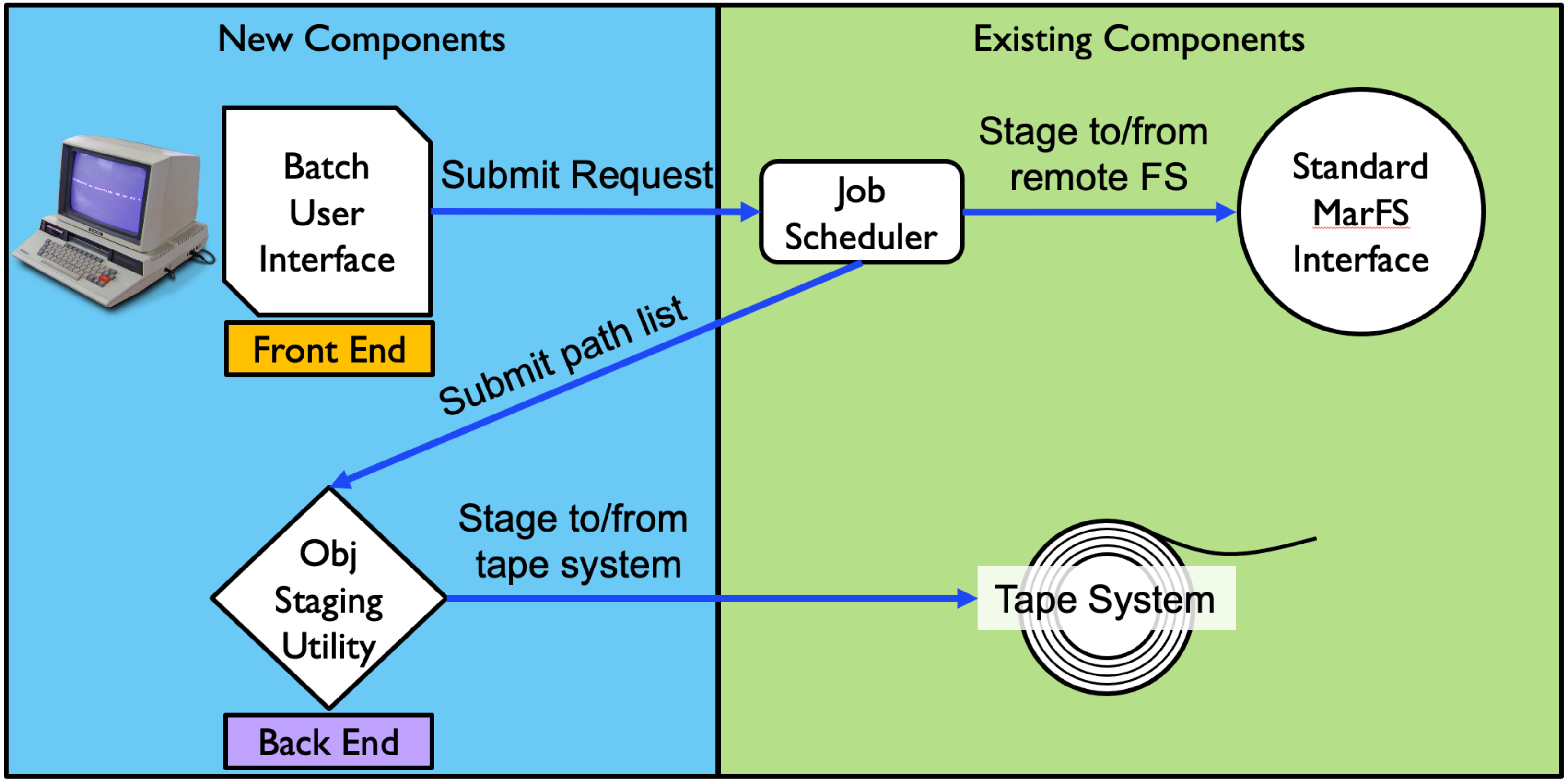
MarFS to Marchive



Marchive – Design Details

- **Robust Data Protection**
  - As MarFS data objects are already checksummed and erasure coded, we get those benefits across tape cartridges as well (RAIT, almost for free)
- **Batch Interface**
  - Interactive staging to/from tape media would be grossly inefficient
  - A ‘batch’-style interface allows for efficient bulk storage/recall without excessive tape remounts and re-seeks
  - Downside is slower archive responsiveness to read requests
- **Same Principles of Simplicity**
  - Makes use of the existing MarFS system design
  - Makes use of existing tape backup solutions
  - Makes use of HPC job schedulers to facilitate transfer queueing

Marchive – Components



Marchive – MarFS Improvements

- Extension to an archive system will require more of MarFS
  - Improved administration toolset
  - Erasure code optimization
  - Altered object garbage collection process
  - Object re-packer
  - Improved system stability overall
- Much of this work is rapidly approaching completion

Thank You for Your Attention!

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