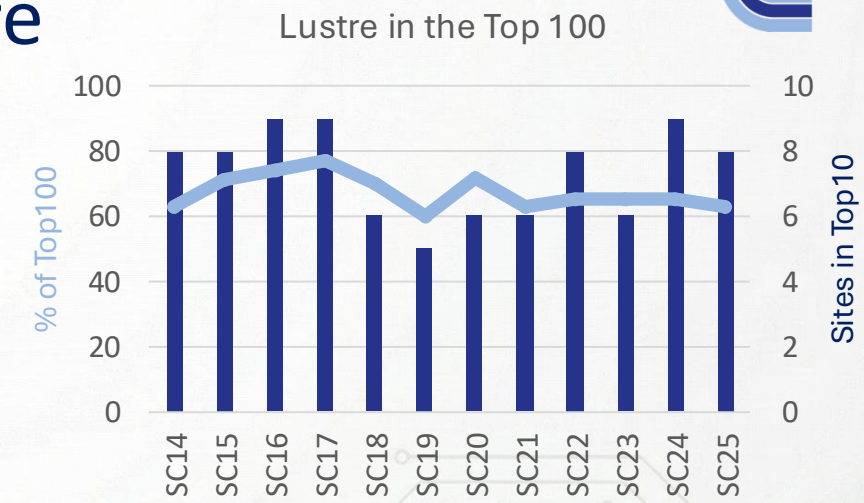
Lustre 2.18 and Beyond

Andreas Dilger, Lustre Principal Architect

Lustre Continues Evolving for the Future



Still the preferred choice for the world's largest systems

- Majority of Top10 and Top100 HPC systems use Lustre
- World's largest HPC, AI systems (ORNL, LLNL, xAI, NVIDIA, …)
  … or 2RU server for lab with a handful of GPU nodes

Scalability of servers and clients almost without limits

- 200 M+ IOPS, 10 M+ metadata op/sec, 500 B+ files
- Capacity for any need – 10s TB/s read **AND** write, 700 PB+ today, 1 EB+ soon
- Fully support large clients - 100s of cores, TBs of RAM, multi-400Gbps NICs, RDMA
- Flash-optimized, and HDD-native performance and capacity since day one

Continued improvements for large system deployments

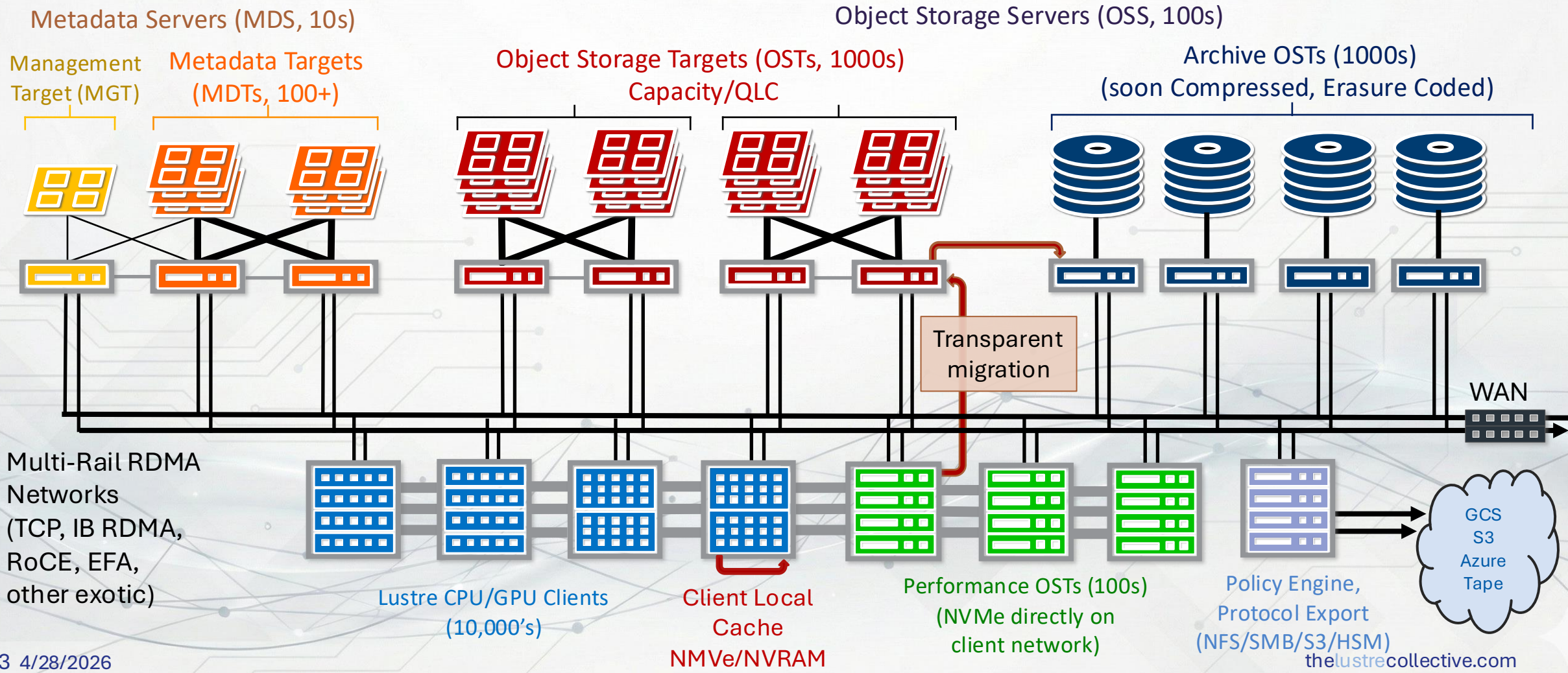
- Steady feature development to meet evolving system and application needs
- Virtualization of filesystem for user isolation and data privacy

Ongoing improvement for ease-of-use, reliability, and efficiency

- Platform agnostic and fault tolerant on commodity hardware
- Faster failover and less disruption for running applications
- API-driven configuration and management for automation at scale

Storage Management Across Entire Cluster

Data locality, with direct client access to all storage tiers



Planned Feature Release Highlights

2.18 has lots of features landing or ready to go

- **File Level Erasure Coding - Read Only** (FLR-ECRO) – M:N file data redundancy (WC, ORNL, TLC)
- **Fault Tolerant MGS** (LMR-FTM) – replicate MGS config and recovery across MDS (WC)
- **Trash Can/Undelete** (TCU) – allow file recovery after accidental/malicious deletion (WC)
- **Persistent Client Cache – Read Only** (PCC-RO) – "Tier 0" cache in client NVMe (WC, TLC)
- **Client-side Data Compression** (CSDC) – reduce network and storage usage/cost (WC, TLC)

2.19 (3.0?) has several features under development and design

- **Lustre Metadata Redundancy** (LMR-MDT0) – MDT0000 services redundancy on other MDTs (WC)
- **Reduced Recovery Time** (RRT) – reduce recovery time after target failover (TLC)
- **File Level Immediate Write Mirror** (FLR-IWM) – redundancy as clients write files (WC, TLC)

2.20 (3.1?) features under discussion for development

- **Lustre Metadata Redundancy** (LMR-ROOT) – R00T/ directory mirroring to other MDTs
- **Metadata Writeback Cache** (WBC2) – single-client metadata speedup (WC, TLC)
- **Lustre Metadata Redundancy** (LMR-DIR) – subdirectory mirroring to other MDTs

Client File Level Erasure Code Read-Only (2.18+ WC, ORNL, TLC)

Dramatic improvement in data availability with low overhead

FLR-ECRO - Redundant *file data* without 2-3x mirror overhead (LU-10911)

- Improve data availability/resilience to in the face of network and server hardware issues
- Initially benefits large ***read-mostly*** files, improve redundancy/availability

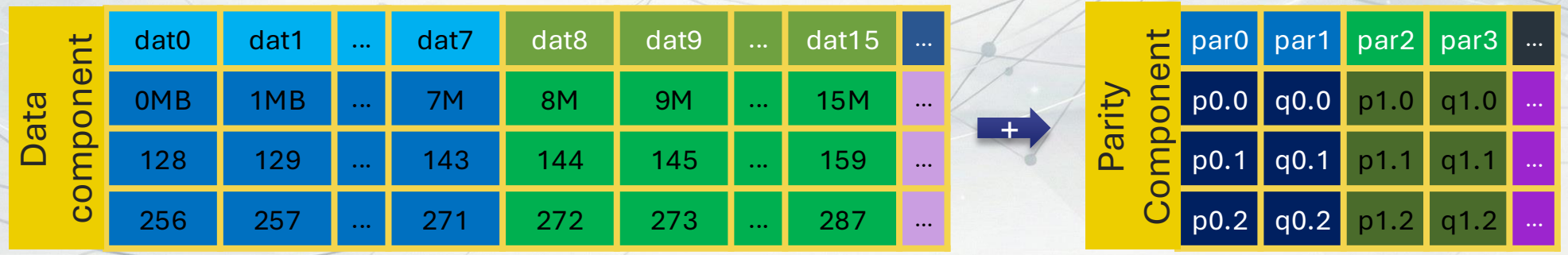
Add EC to new (*and old)* striped files ***after*** write finished

- **Delayed redundancy** avoids overhead during initial application write (and easier to do!)

Large striped files - add N+M RAID Sets (e.g. 8d+2p or 16d+3p)

- Fixed RAID-4 per file component, de-clustered across files, CPU-optimized (Intel ISA-L)
- EC RAID sets independent of number of stripes, works well with PFL layouts
- Survive OST loss with minimal read delay (seconds), transparent to applications

Will implement Immediate Write Mirror and Immediate Write EC in later phases (LU-13643)



Existing RAID-0 data stripes in 128-stripe file (16x8 RAID Sets)...

16 sets of parity stripes written later...

Dynamic Server NID Addition (LU-10360 2.17 WC)

New NID(s) can be dynamically added to servers online

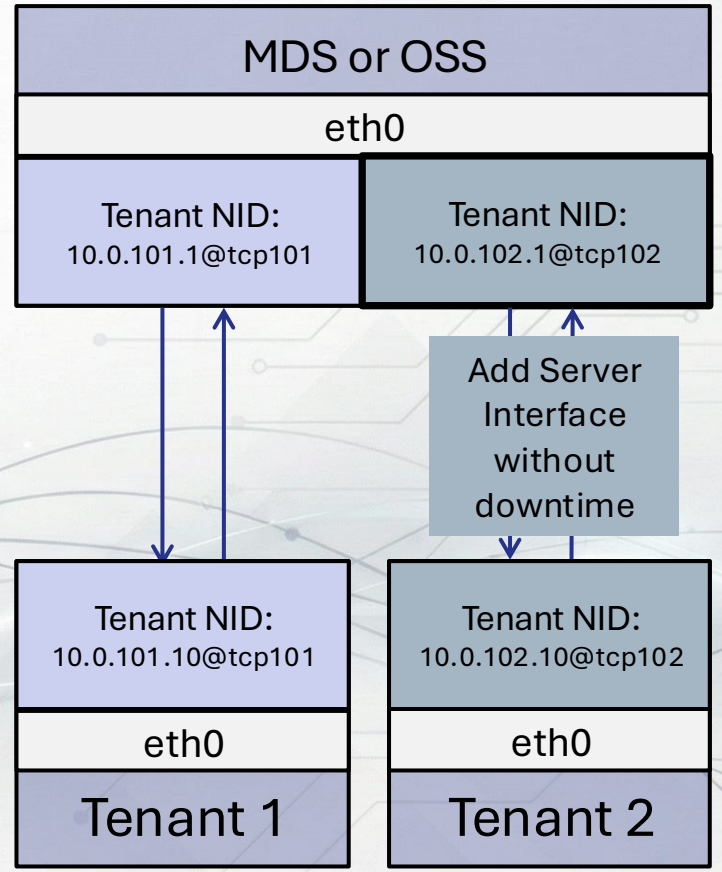
- Allow thousands of dynamic server NIDs to be configured
- Useful for VLANs assigned to subsets of clients

No need to store dynamic NIDs in config log

- No `writeconf` outage to re-address/migrate
- Flexibility for dynamic network environments, multitenancy

Dynamic NID configuration process

- OSTs/MDTs advertise new server NID(s) to MGS
- MGS (and replicas) store NIDs for each target in memory
- Client learns learn target NIDs from MGS
- Tunable: `mdc.*.enable_dynamic_nids=1`
  - Default if no server NIDs found in config log by client



Data Security and User Isolation via Nodemaps (2.17+)

Strongly isolate user/tenant data for legal/operational reasons

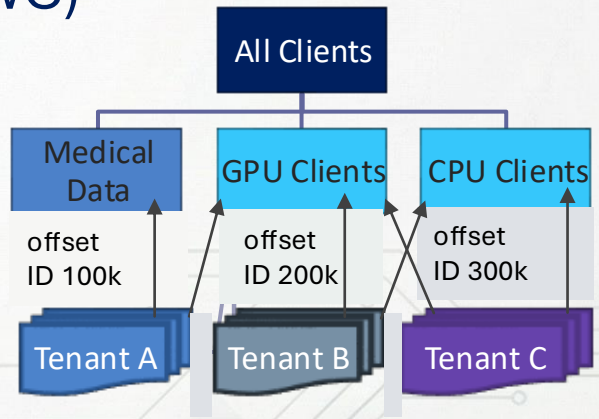
- Nodemap ID offset range for UID/GID/PROJID mapping (LU-18109 WC)
- Client-local `root` chown/quota within nodemap (LU-18694 WC)
- Multiple and read-only filesets per nodemap (LU-18357 WC)
- **Persistently ban clients by NID/nodemap/`default`** (LU-19157 WC)

2.17
- Auto-load SSK `.lgss` key from client mountpoint (LU-19326 WC)

2.18
- **Client nodemap mapping only by SSK/KRB5 (no NIDs!)** (LU-19079 WC)
- Multiple SSK keys on *client* for different Nodemaps (LU-19079 WC)
- Multiple SSK keys on *server* for key rotation/expiry (LU-8229 WC)
- Allow setting RBAC on `default` nodemap (LU-19975 WC)
- New GSS IAM authentication mechanism (LU-19921 Google, WC, TLC)

2.18
- Add `foreign_dir`, `lqa`, `projid` ops to nodemap RBAC (LU-19884, LU-19963 TLC, WC)

2.19
- Encrypted `fscrypt` backup/restore without key (LU-16374 WC, TLC)



Lustre Quota Aggregate (LQA) (LU-18222 2.18 WC)

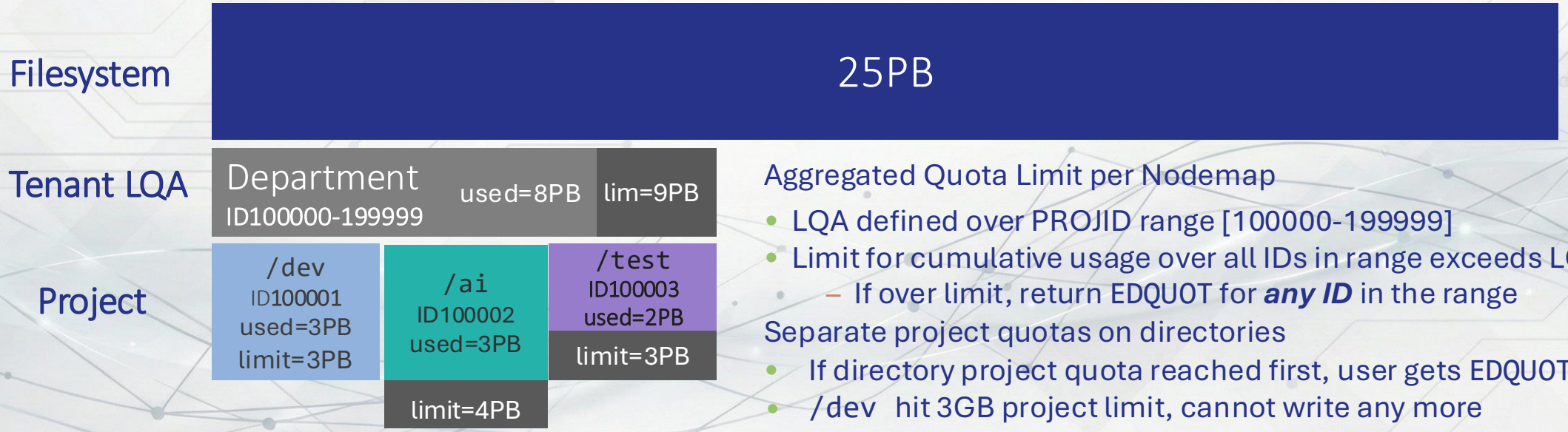
Aggregates quotas across a ranges of IDs

Range quota independent of limit on individual UID/GID/PROJID

- **Lowest** quota limit reached for any ID is enforced first

Allows multi-level quota limits for nodemaps

- Delegate usage of Project, User, Group quota to tenant admins



Aggregated Quota Limit per Nodemap

- LQA defined over PROJID range [100000-199999]
- Limit for cumulative usage over all IDs in range exceeds LQA limit
  - If over limit, return EDQUOT for ***any ID*** in the range

Separate project quotas on directories

- If directory project quota reached first, user gets EDQUOT
- `/dev` hit 3GB project limit, cannot write any more
- `/ai`, `/test` together have 1PB for LQA limit

Persistent Client Cache (PCC-RO) (LU-10499 2.18 WC, TLC)

Leverage client-local storage to accelerate workloads ("Tier 0")

Reduce latency, avoid network traffic to servers, fast small/unaligned IOPS

- Ideal for read-heavy workloads reusing files multiple times (e.g. training, KV Cache)
- Cache large shared libraries (Python!), large datasets, current training data, etc.

PCC integrates Lustre with **local cache storage** on client (NVMe, NVRAM, CMX+BlueField)

- A local directory (e.g. `/var/cache/lustre`) or mountpoint is used on the client
- **File data mirrored by client** on first read based on PCC policy, or explicit request
- Cache file version verified with Lustre on read, DLM lock avoids frequent reverify

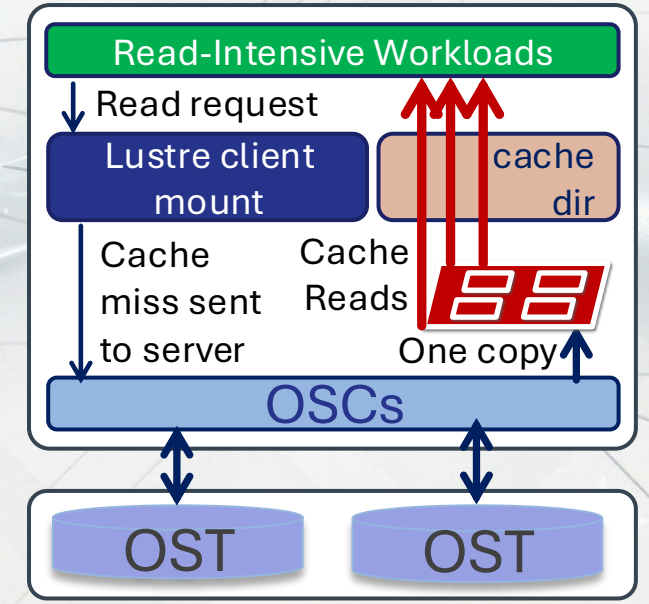
Apps transparently read from local file on cache hit, or from Lustre

- Cache files maintain `fscrypt` encryption if used in Lustre
- Can "`pin`" important datasets into cache, others cached on demand

Some work remaining to finish PCC-RO for 2.18 release

- ***Userspace tool for cache space management, removal of old files***
- Simplify config/usage (e.g. "`-o cache=/var/cache/lustre`")

PCC-*RW* prototype exists, more work needed for production



Client-Side Functionality Improvements

Ease-of-use and performance improvements for end users and admins

Ongoing updates for 6.x kernels and later upstreaming (ORNL, HPE, SuSE)

Kernel 6.9 folios (page extents) for efficient large IO/cache (LU-17916 HPE)

Allow CPU cores to be excluded from service thread usage (LU-17501 WC)

Obfuscate file/dir names in debug logs to limit PII exposure (LU-18810 WC)

Cache `statfs()` on project directory (LU-16771 WC)

Reduce RPC latency for client IO via CPU power states (LU-18446 NVIDIA)

Regexp to *exclude* `mkdir` from DNE3 MDT space balance (LU-19268 WC)

2.17 Latency by RPC size per target in `osc.*.io_latency_stats` (LU-18933 WC)

2.18 Client cache for sparse file reads (LU-19469 WC)

Client-side target performance stats via `OBD_STATFS` (LU-7880 George!)

Client-Side User Tools Improvements

Sometimes it's the small things that make a big difference…

`lfs df --mdt/--ost/--output` prints specific MDT, OST, fields (LU-18242 WC)

`lfs fid2path` lookup for **OST** object FIDs (LU-13527 WC)

`lfs find/project/quota` allows named projects in **`/etc/projid`** (LU-13335 WC)

`lfs find` multi-threaded parallel scanning (LU-17814 WC)

`lctl get/list_param` show **`--path`**, merge output **`--dshbak`** (LU-17343, LU-14590 WC)

`mount.lustre` automatically loads SSK/KRB5 key from mountpoint (LU-19326 WC)

`mount.lustre` applies **`/etc/lustre/mount.{client,FSNAME}.params`** (LU-11077 WC)

2.17 **`lctl find_param`** to locate parameter names by regexp (LU-18418 WC)

2.18 **`man4/`** pages for `lctl get/set_param` parameters (LU-*many* WC, TLC, Microsoft)

`lctl help_param` to show man page for matching parameter (LU-17231 WC)

`lfs_migrate` script merged into **`lfs migrate`** command (`-A`, `--restripe`) (LU-16449 WC)

2.18 **`llapi_*()`** functions for Changelogs, HSM, quota (LU-19296, LU-19276 WC, TLC)

2.19 **`lfs migrate`** migrate/replace a single object in a file to new OST (LU-9961)

`lfs migrate --skip-rebalance` auto-balances OST space usage (LU-19052 WC, TLC)

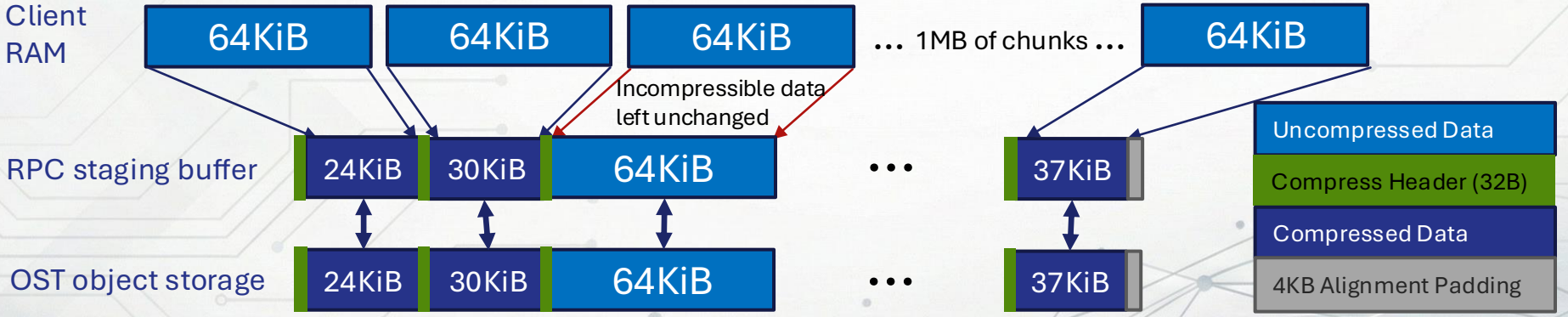
Client-Side Data Compression (LU-10026 2.18 WC, TLC, MS)

Parallel (de-)compression of RPCs on clients for GB/s speeds

- Lots of client cores, reduced server CPU load, network bandwidth

(De-)Compress (`gzip`, `lz4`, `zstd`,…) RPC on client in chunks (64KiB-4MiB+)

- Per directory or file component selection of algorithm, level, chunk size (PFL, FLR)
- Keep "uncompressed" chunks as-is for incompressible data/file (`.gz`, `.jpg`, `.mpg`, …)



Client write/read whole chunk(s), (de-)compress in RPC bounce buffer

- Large chunk size improves compression, but more overhead if file read non-sequentially

Could write uncompressed to one FLR mirror for random IO pattern, recompress to another

- Data (re-)compression during mirror/migrate to slow tier mirror (via data mover)

Readahead Optimizations (2.18+)

Readahead is fundamental to achieving high bandwidth for many applications

Readahead currently optimizes multiple access patterns

- Sequential, strided forward readahead (offset + N x {1,2,3,...})
- Fuzzy readahead for mmap (offset + N-ish x {1,2,3,...})
- Backward readahead "exists" (offset – N x {1,2,3}) but is incorrect in some cases

Separate readahead mode for mmap file access missing some features

Substantial improvements possible, work already in progress

- Improved backward fuzzy readahead to better match real client IO (LU-15274 TLC)
- Opportunistic lockahead for readahead to keep pipeline filled (LU-15155 TLC)
- Merge mmap and standard readahead logic so optimizations usable by all (LU-15516 TLC)

LNet and Configuration Improvements

Mount client without server NIDs in config log (LU-10360 WC)

- Simplify network setup, server migration/failover config, etc.

Hundreds of NIDs for tenant networks (LU-19498 WC)

Filter server NIDs for local client network (LU-19299 WC)

EFA optimized LND for Amazon FSX (LU-18808 AWS)

Improve handling of MGS with many NIDs (LU-16738 WC)

- Display MGS hostname instead of ugly NID list in `mount/df` output

2.17

- **`mount.lustre`** round-robin DNS lookup for MGS failover NIDs

2.18 Finish IPv6 NID support (LU-18417 ORNL, HPE, WC, TLC)

Improve data transfer for sparse file reads (LU-16897 WC)

lnetdump - network agnostic eBPF packet analyzer for LNet (LU-19410 AWS, TLC)

- Generate packet dumps for any LNet network type (TCP / IB / EFA / Loopback / KFI)
- Performant enough for use in production, enable real time monitoring and alerting

Server-side Improvements

Improved robustness and ease of management

OST emergency read-only shutoff parameter (LU-12515 WC)

RAM-based OSD for test/shared memory (LU-17995, LU-18813 AWS, WC)

- Initially for API testing, code consolidation, benchmarking

Read and return small directory contents on open (LU-18448 HPE)

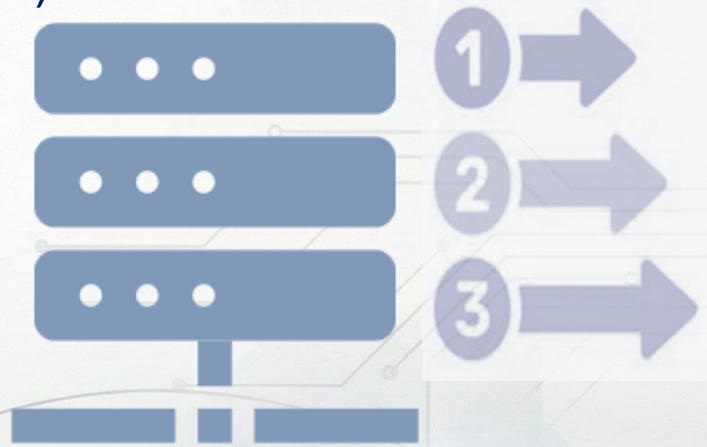
Network Request Scheduler TBF (QOS) improvements

- Persistent NRS TBF rules via "`lctl set_param -P`" (LU-17920 WC)
- Compound rules for different TBF types (LU-18192 WC)
- TBF rules by projid, ID ranges (LU-17166 WC) 2.17
- TBF rules by nodemap name (LU-17902 WC) 2.18

New NRS Fairshare policy to avoid "noisy neighbor" (LU-17296 HPE)

- Avoids throttling IOs when no contention is present

Enable default PFL layout on newly-formatted filesystems (LU-11918)



Ongoing Backend Storage Improvements

Persistent `TRIMMED` flag to avoid full `fstrim` on remount (LU-17980 WC)

Direct T10-PI integration of RPC and storage checksum (LU-10472 WC)

Hybrid `ldiskfs` OST storage devices (NVMe+HDD) (LU-16750 WC)

- Store OST metadata on NVMe for performance, data on HDDs for cost
- Significantly improved mount, e2fsck, stat, update performance

2.17 ZFS updated to 2.4 release

2.18 `io_latency_stats` per target by RPC size (LU-18934 WC)

ldiskfs "-o `fstrim`" mount option for smart TRIM operations (LU-14712 WC)

Export `fstrim` statistics and history per filesystem (LU-19158 WC)

Submit ldiskfs `dirdata` patches to upstream ext4 (LU-17423 TLC)

Delayed block allocation for writeback cache in `ldiskfs` (LU-12916 WC)

2.18

- Aggregate small writes in server RAM to reduce IOPS overhead

2.19 Parallel `e2fsck` pass2/3 (direntries, name linkage) (LU-14679 WC)

Reduced Recovery Time (LU-19199 2.18+TLC)

Have your cake and eat it too?

Lustre IO typically very fast due to stateful clients and async writes …

- Reduce write time during normal operation (avoid sync IO = less application waiting)

… but servers need to recover client state if they fail to avoid losing unwritten client data

- Server failover "pauses" filesystem access during this time (5s-600s, depends on load/scale)
- Delays over 10s unacceptable for some workloads, even if saving hours of IO time per day
- 10000+ clients can be a DDOS on Lustre servers, "wait less" is not a robust solution

Multiple areas need improvement to avoid application "pause"

- FLR-mirror, FLR-ECRO – will hide failover delay for file reads, FLR-IMW/ECWR will do for writes
- LMR directory/inode mirroring will hide failover delay for file/directory lookup and access
- Servers should actively reconnect to slow clients after 50% of clients have reconnected
- Store client interaction (shared file, dir, JobID, nodemap) on target to isolate recovery after restart
  - Many modern workloads are isolated "ensemble" processed rather than large shared MPI jobs
  - Limit eviction to clients within the same cohort that showed recent interaction before server restart
  - Allow clients with no interactions (separate dirs) to restart quickly, or reconnect **after** recovery over
- Limit recovery time once 90% of clients reconnect (no long wait for dead clients)

API-Based Management and Configuration (2.18+ TLC)

API-driven configuration, management, monitoring

Efficient NetLink interface to Lustre statistics (LU-19768 TLC)

- No more opening, scraping, parsing hundreds of stats files per second

Improve Lustre Library (`llapi_*()`) to cover existing functionality (LU-*many*)

- Move core logic out of `lctl`, `lfs`, configuration scripts into shared library code

Rust-based interface (`rustreapi`) for robust development (LU-19561 TLC, WC)

Simplified high-level commands/API for configuration management, ease of use

- **add OST** – format block device, apply parameters, safely add into filesystem
- **remove OST** – migrate all objects off OST, remove from configuration
- **rebalance OSTs** – migrate files/objects across OSTs after new OSTs are added
- **tenant add** – add virtual LNet+NIDs, nodemap ID range, subdirectory, LQA, SSK keys, fscrypt

Trash Can/Undelete (LU-18465 2.18 WC)

Allow files/directories to be undeleted after `rm`/`rmdir`

- Rescue users from fat-finger mistakes, broken scripts, bad actors
- Handle "`rm -r`" properly to allow whole-tree recovery

Deleted files in trash are flagged and treated specially

- Removed from user/group/project quota and `df` usage
- Files in trash cannot be written to avoid abuse

Virtual `.Trash` directory visible in every directory

- Can use normal tools to list and recover files or directories
- `.Trash` is *hidden* from normal directory listing (no backup, app issues)

Users can view and recover their own files until cleanup

- Configurable time/fullness before cleanup (e.g. max age = 7d, 80% full)
- Complex cleanup policies run in userspace (e.g. by user, project, nodemap)

Lustre Find Utility (LFU) (2.19+ TLC)

Lustre integrated/optimized namespace scanning and reporting

- Server-based target scanner for speed (Lester the Lustre Lister, OSD OI Scrub, Changelog)
- Structured binary Object Stream for efficiency and flexibility (FlatBuffers, MsgPack, ?)
- **Parallel at multiple levels, minimizes data transfer/translation during scan**
- Specific indexes on MDT for optimized scans (e.g. atime, largest size, stale mirrors)

Modular stackable interface to allow component re-use

- Merge Object Streams while scanning multiple targets in parallel
- Remote bulk RPC transport to client (with appropriate access control filters)
- Efficient zero-copy lockless ring-buffer export from kernel to userspace

Pluggable filter rules to isolate objects and attributes of interest

- Filter by attrs (ranges), IDs (range, list), xattrs (regexp), MDT/OST (range, list), pool, ...
- Aggregates of attributes for efficient reporting/filtering (min (+list), max (+list), count, sum, histogram, ...)
- Selectively pass attributes of interest on to upper modules (path, FID, attrs, everything)

Flexible "output" modules to consume Object Stream for end user

- "`lfs find`" -> text filename/FID list; JSON, CSV, Parquet and other structured text/binary formats
- Directly input Object Stream into utilities for FLR mirror/EC resync, Trash Can cleanup, PCC-RO cache, ...

LFU Data Flow Overview

```
lfs find /lfs02 -atime +30d -blocks +1G -projid 1999 -printf %LF
```
(LFU: "scan:all MDTs, filter: atime, blocks, projid, output: FID")

Input Scanner
- ldiskfs scanner
- OSD-agnostic API
- Changelogs
- Directory crawl

- Scan all inodes in `lfs02` (with user access control)

Requests / Object Stream

Filter Rules
- Attribute filter
- Aggregate streams
- Fork stream
- PtlRPC transport

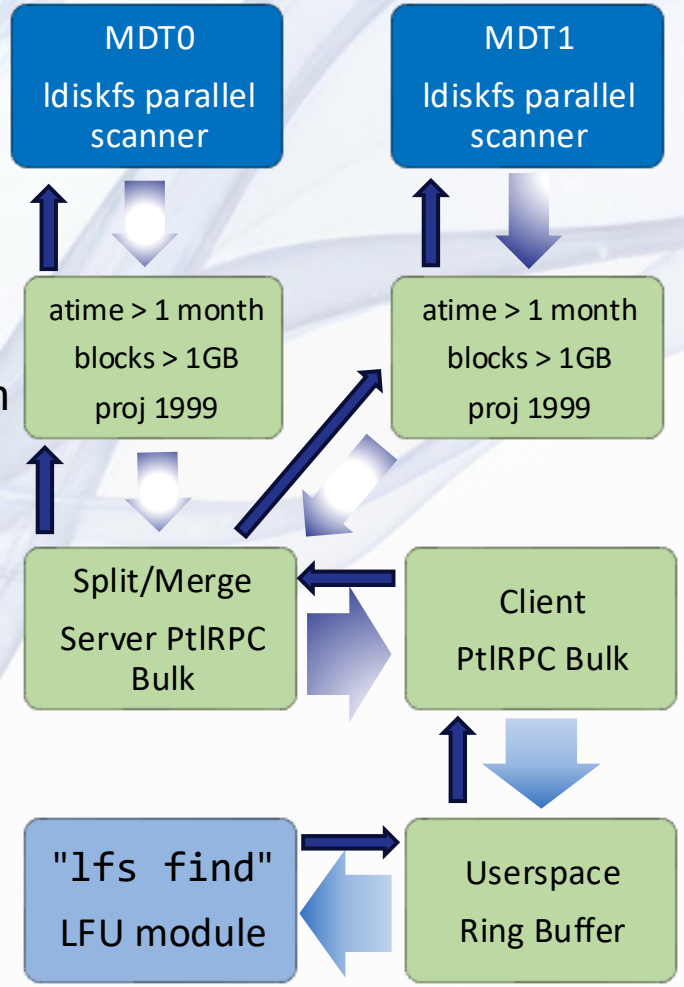
- Filter all but matching
- Only FIDs in Object Stream

Requests / Object Stream

Output Format
- Text file/FID list/printf
- JSON/BSON, Parquet
- FlatBuffer, MsgPack
- Direct binary import

- Split request to MDTs
- Merge reply streams

- Scan atime, blocks, projid
- Output FIDs only, or add pathnames at this point



LFU Consumers and Implementation

LFU doesn't need to do *everything* before it becomes very useful

- Available out-of-the-box on every Lustre installation, no database needed
- Optimized "`lfs find`" is enough to start
- Implement incrementally (server only, client root only, PtlRPC transport, regular users, ...)

Consolidate and speed up multiple similar functions already in Lustre utils

Needed for many existing and upcoming features in Lustre

- PCC-RO cache manager, FLR mirror/EC resync, tiered storage manager, HSM (S3, GCS, ...)
- Efficient reporting of filesystem statistics for admins and users

Open to incremental enhancement for things we haven't thought of yet

Optimized scanner for other tools if needed (RobinHood, PoliMor, GUFI, ...)

- Co-opetition in some cases, but if they scan the filesystem they may as well do it efficiently

Fault Tolerant MGS (LMR-FTM) (LU-17819 2.18 WC)

Replicate MGT to multiple server nodes for better availability

- Allow clients to get config logs from **any server node**
- Reduces mount time/timeouts, reduced MGS load in large clusters
- MGS Imperative Recovery works even if “primary” MGS node restarts

Initial steps for replication of other MDT0000 databases

- FLDB, Quota, Nodemap, …

Implementation well underway for 2.18

Lustre Metadata Redundancy (LMR 2.19+)

Metadata mirroring across MDT devices

LMR-FTM: Fault Tolerant MGS (LU-19116, LU-19117 WC)

LMR-MDT0: MDT0000 service redundancy (LU-18823)

- Mirror FLDB, Quota, `flock( )` to other MDTs

LMR-FID: Dir entries with multiple FIDs (LU-16742 TLC)

- Infrastructure for inodes mirrored to multiple MDTs
- Replicas visible from client, directly access if MDT offline

LMR2: Replicate directories for availability (LU-17820)

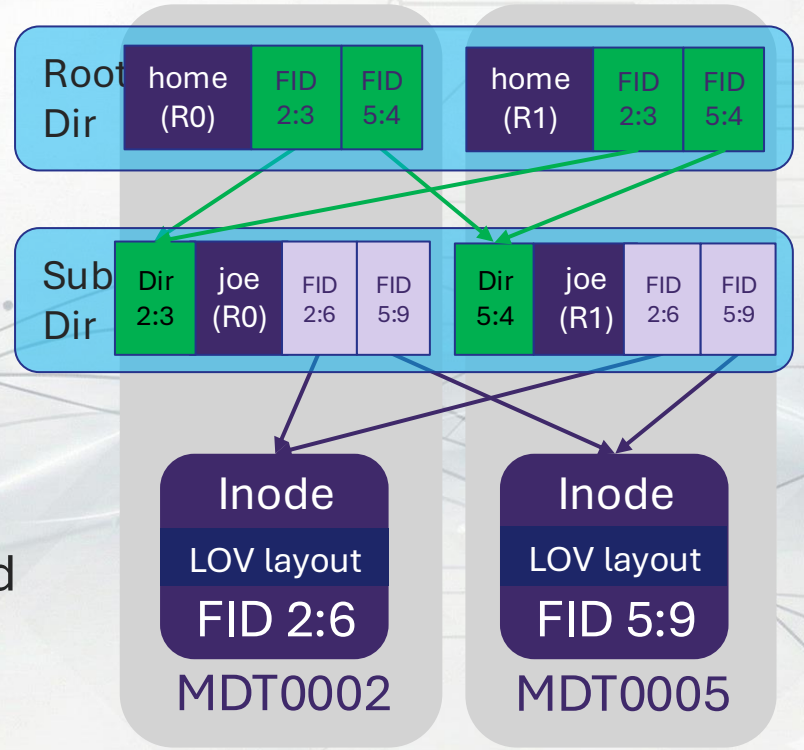
- ROOT: Mirror `ROOT/` directory (rarely changed) to other MDTs
- DIR: Subdirectory mirror to 2+ MDTs (`setdirstripe`)
- FILE: Per-file metadata replication for regular files

LMR3: Complex recovery and write handling

- RECOV: Write to directory while mirror offline, full MDT rebuild

LMR4: LFSCK to repair/rebuild inconsistent mirrors

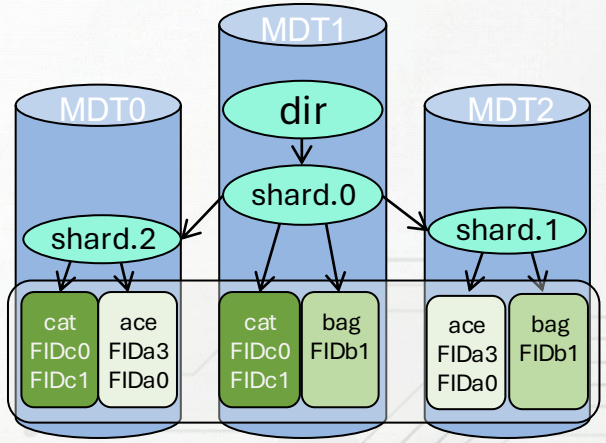
- LFSCK: reconcile inconsistent mirrors, rebuild offline MDTs



MDT0000 replacement (LMR1b/2a 2.19+)

Remove service dependency on MDT0000 (LU-18823)

- FLDB (FID->MDT/OST map) redundancy (LU-15414)
- Nodemap config redundancy
- Quota ID limit settings redundancy (usage can be rebuilt)



Quota Manager distributed to multiple servers

- Hash distribution by UID/GID/PROJID, easily parallel operations

LDLM `flock()` distribution and scaling over multiple MDS nodes

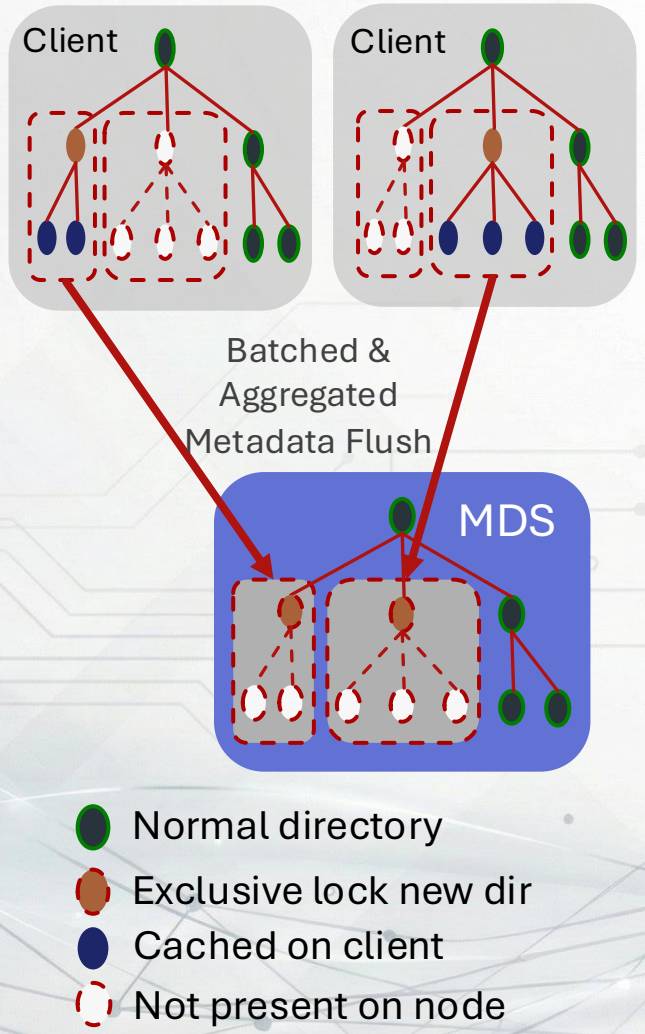
- `flock()` locking is not really related to MDT inodes, so can run anywhere
- Need to handle deadlock detection, but is disjoint for process groups, nodemaps

Mirror ROOT/ directory to multiple MDTs (LU-17820)

- Rarely changes for most uses, some update overhead is acceptable

Enough to allow replacing MDT0000 on live system

Metadata Writeback Cache (WBC2 2.19+)



10-100x speedup for single-client create workloads

- Data ingest, producer/consumer, untar/build
- Gene extraction/scanning

Create new dirs/files in client RAM without RPCs

- Lock new directory exclusively at `mkdir` time
- Cache files/dirs in RAM until flush or remote access

No RPCs for metadata changes in new dir

Batch RPC for efficient directory read/cache flush

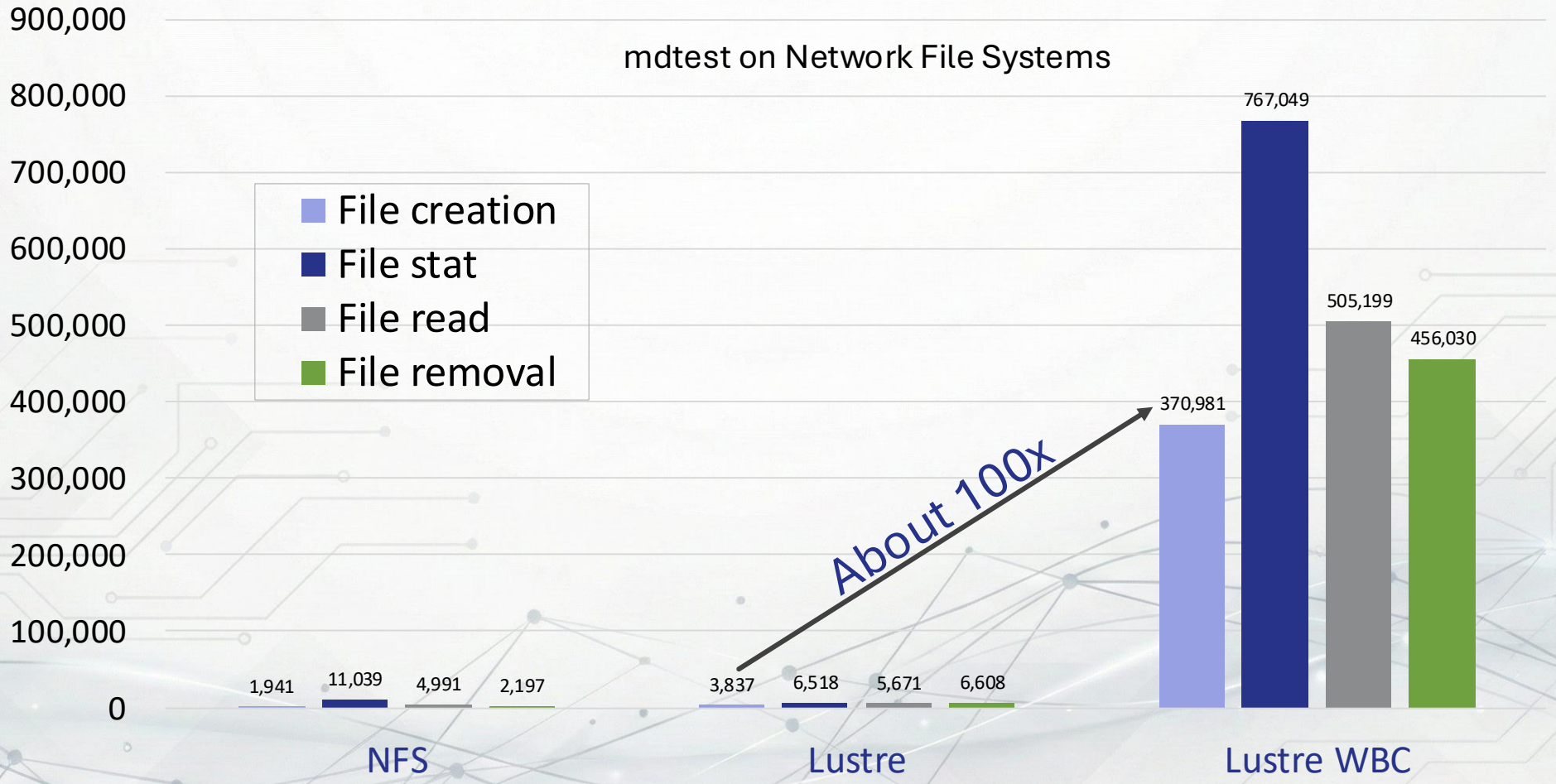
Files globally visible on remote client access

- Flush top dirs, exclusive lock new subdirs, unlock parent
- Flush rest of tree in background by age or size limits

WBC feature already significantly implemented

- Some complexity handling partially-cached directories
- Need to integrate space usage with quota/grant

Metadata WBC Performance Gains



Lustre: DDN AI400X Appliance (20 X SAMSUNG 3.84TB NVMe, 4X IB-HDR100)
Lustre clients: Intel Gold 5218 processor, 96 GB DDR4 RAM, CentOS 8.1
Local Filesystem on SSD: Intel SSDSC2KB240G8

THANK YOU!