

Lustre Find Utility

Artem Blagodarenko, Ph.D.

Hiroshi Nishida, Ph.D.



Why Lustre FS Needs LFU

- Billions of files and directories
- Demand to understand what files are present
- Find subset of files with desired attributes

Tools today:

- Client scan of files, attributes: *slow, many RPCs*
- Proprietary solutions: *closed, partial features*
- Each tool implements redundant scanning



LFU is an open-source, extensible, and **modular solution**

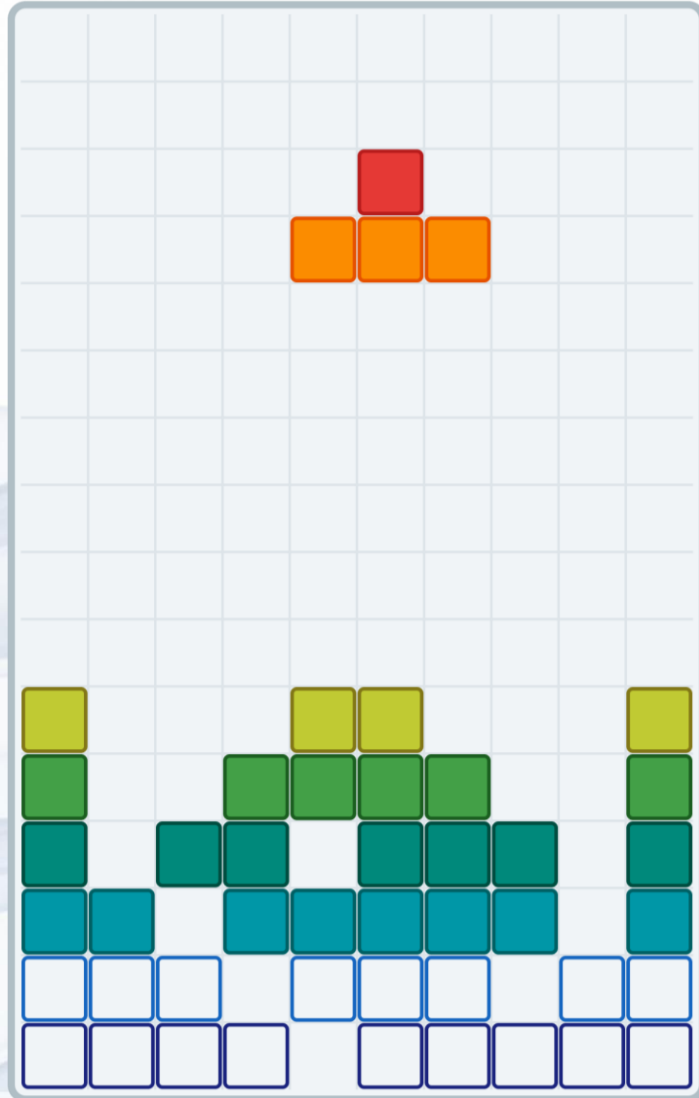
Integrated into the **Lustre source tree** for broad availability

Address wide range of **file-finding** and **metadata-search** requirements

Core functionality reusable for integration into many tools

Case: Migrating Files Between Storage Tiers

SSD pool



Files	
main.c	14.02.2021 09:32
config.yaml	03.06.2021 14:07
notes.txt	28.10.2022 18:45
report.docx	19.01.2023 08:12
data.csv	07.05.2024 11:58
index.html	22.09.2025 16:20

List (changelog)

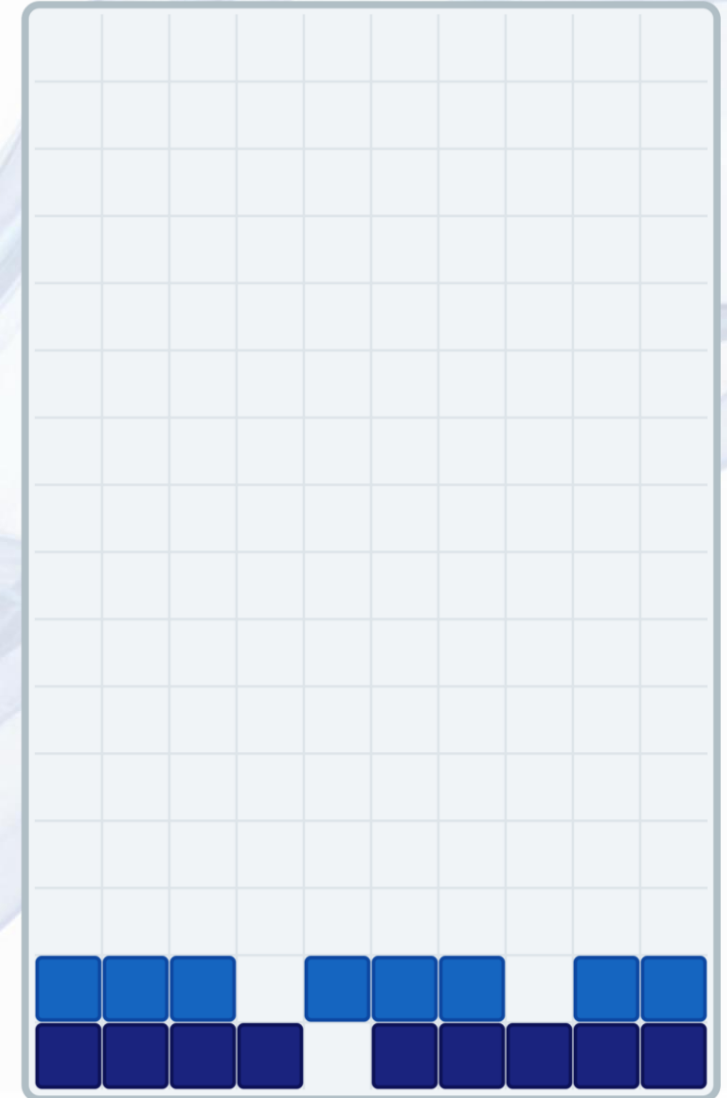


Scan (lfs find)

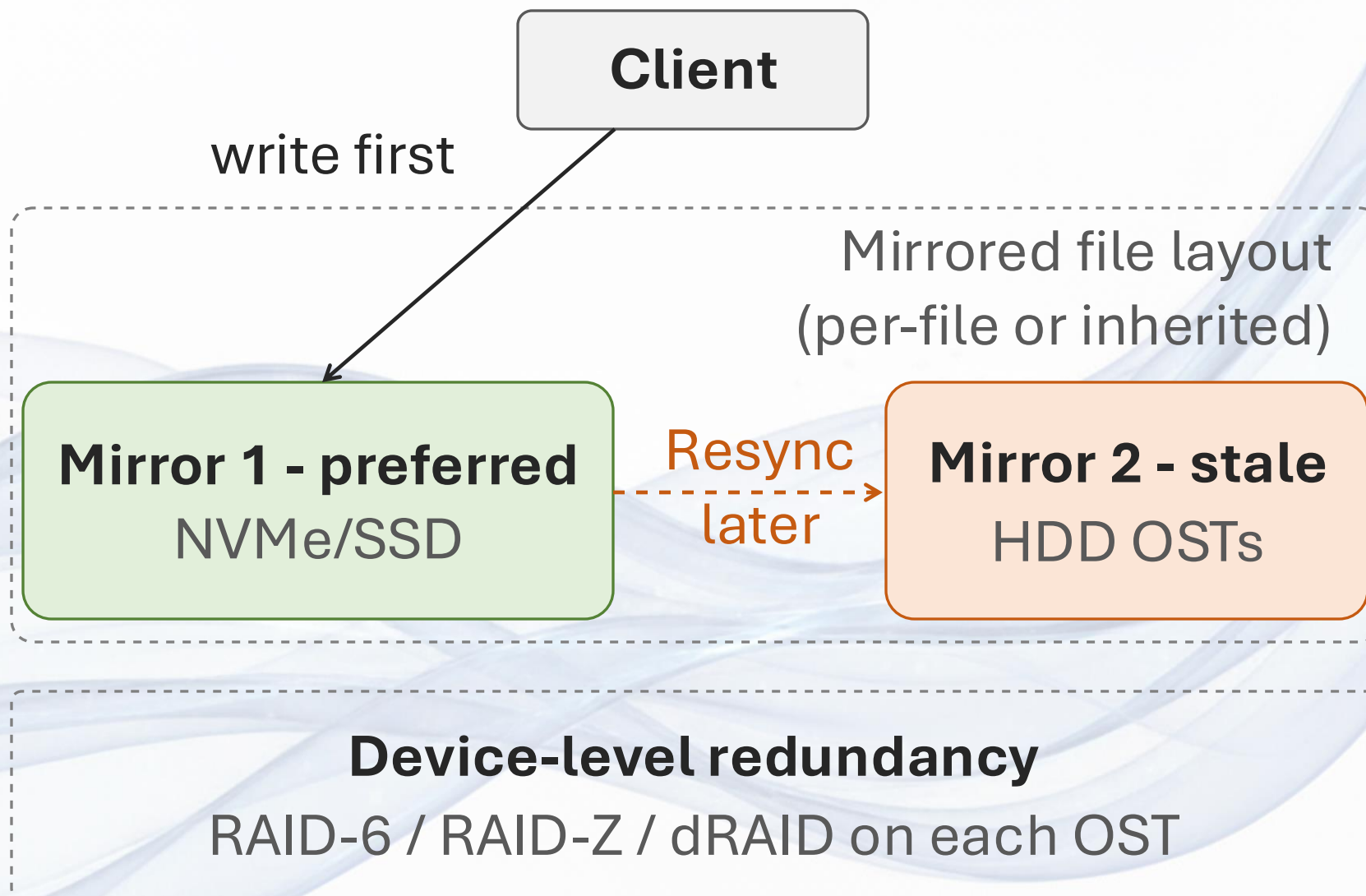


Move (lfs migrate)

HDD pool

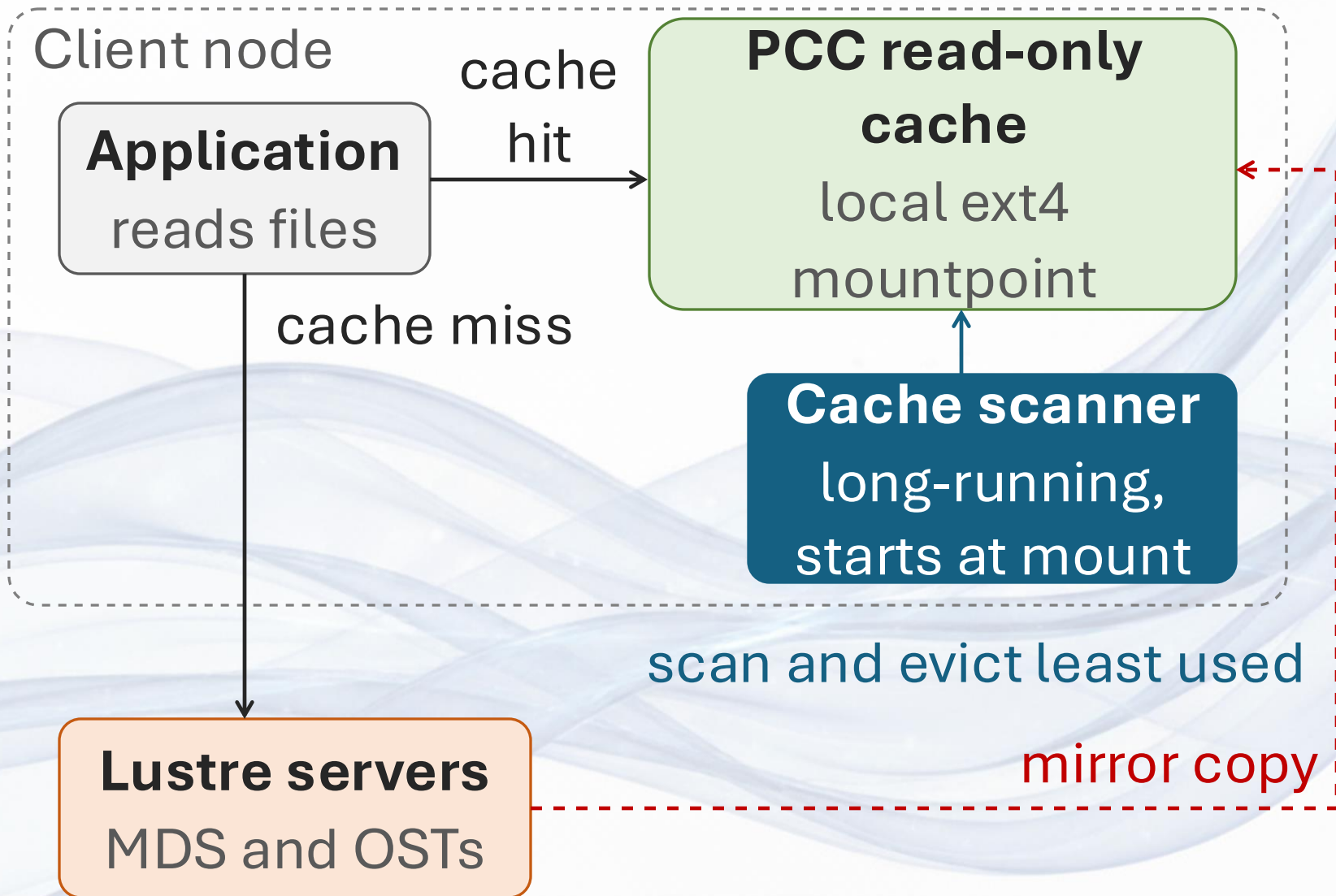


Case: FLR Delayed Mirroring



- Resync **all recently written files** (e.g. Changelog)
- Only files **with file layouts** using built-in redundancy
- Next FLR phases require even more **sophisticated metadata management**

Case: PCC Client-Side File Cache



- Local cache directory (e.g. **ext4** mountpoint)
- **Scanned periodically** to find oldest or least accessed files
- Local cache copy of file is removed when space is needed

Case: Recompress File Data In-Place (CSDC)

First, write data fast with low compression level

```
$ mkdir /mnt/lustre/dir
```

```
$ lfs setstripe -E1G -c 1 -Z lz4:1 -E eof -c 8 /mnt/lustre/dir
```

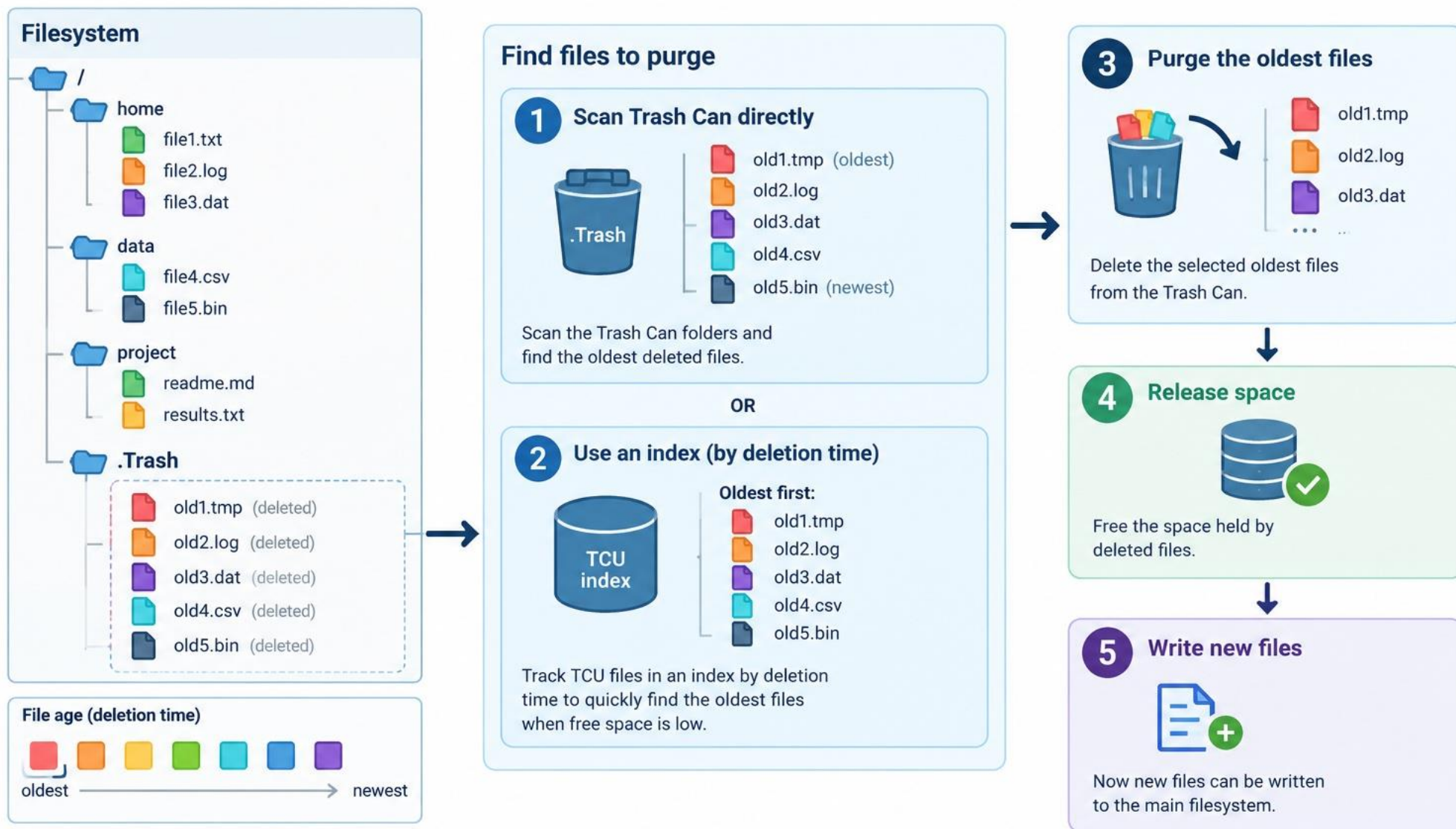
```
$ cp /mnt/data/AMSR_E_L3_DailyOcean_V05_20111003.hdf  
/mnt/lustre/dir/ocean_simulation
```

When system is idle, recompress files (e.g. background service process)

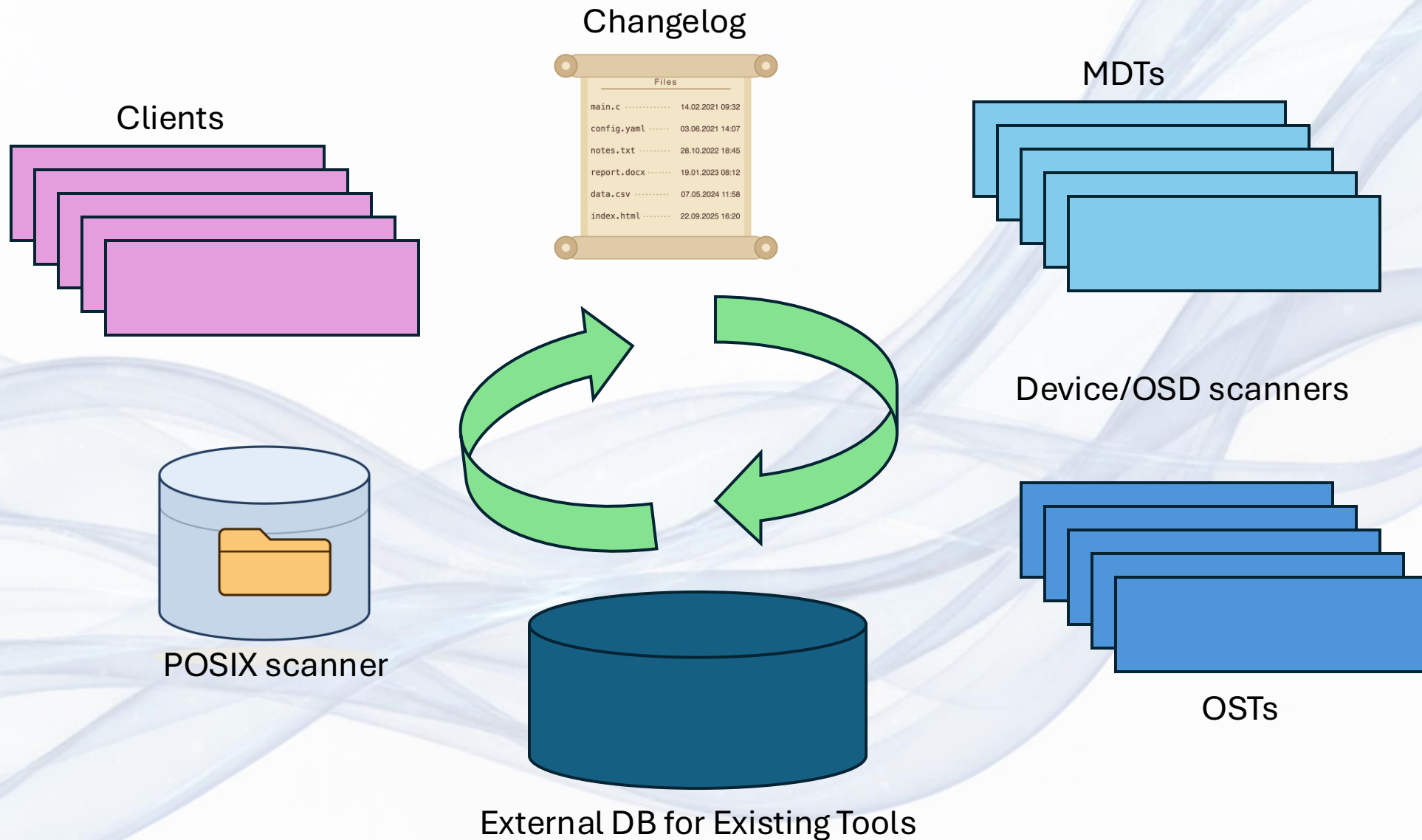
```
# lfs find /mnt/lustre --compress-level=-5 --mtime +1d |  
lfs migrate -Z lz4:9
```

Need to **scan metadata** to understand compression parameters

Case: Trash Can Undelete (TCU)



Minimal LFU (based on the use cases)



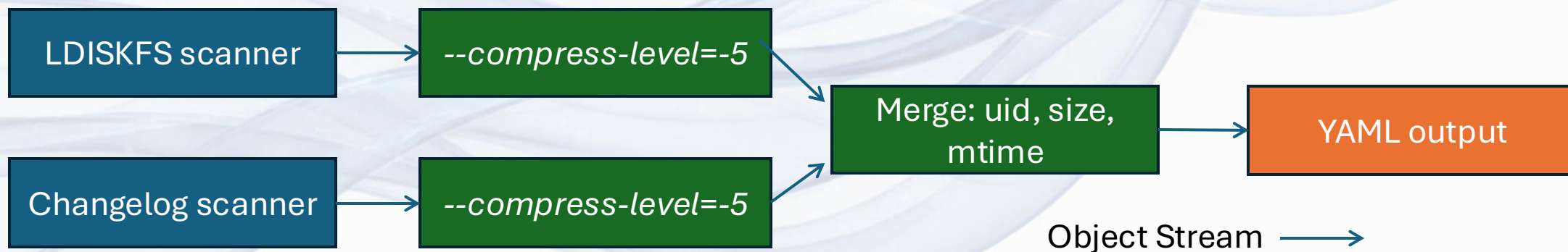
LFU Base Requirements

https://wiki.lustre.org/Lustre_Find_Utility

- **Modular.** Core infrastructure has a modular/pluggable architecture
 - Allows the **initial development** of a usable basic infrastructure
 - Facilitates LFU to be **extended incrementally**
 - Can handle other input sources, filters, and outputs in the future
- **Binary Data exchange.** Efficient interface between modules
 - Object Stream passes object FIDs and attributes between modules
 - **Efficient without additional reparsing** or data restructuring at each layer
 - Provisions in Object Stream protocol to allow **future expansion**
 - Attributes passed between modules allows interop with new modules

Modules types

- **Input Scanner Module** generates a list of objects from a source (MDT, OST, Changelog or index)
- **Filter Rule Module** Process one or more Object Streams and produce another object stream based on the object attributes and matching criteria
- **Output Format Module** Process an Object Stream and act on it, or produce output in a different format for other tools



Modular System for Flexible Usage

Input Scanner

LDISKFS device scanner

FS Namespace scanner

Changelog Scanner

Raw Read Scanner

OSD API Scanner

Index Scanner

Aggregate Scanner

Filter Rule

lfs find Filter/output

Stream Merge/Split

Changelog Filter

Advanced Operators

Server Bulk RPC Filter

Client Bulk RPC Filter

Smart Input Selection

Output Formats

Text: FID + Pathname

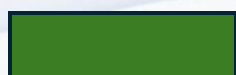
Raw Write Output

Structured Formats

Database Plugins

Persistent Aggregates

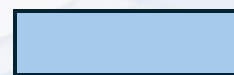
Grafana dashboard



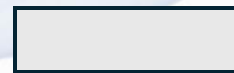
Already implemented in Hiroshi Nishida's patches
<https://review.whamcloud.com/68818>



Prototype implemented for TLC's release

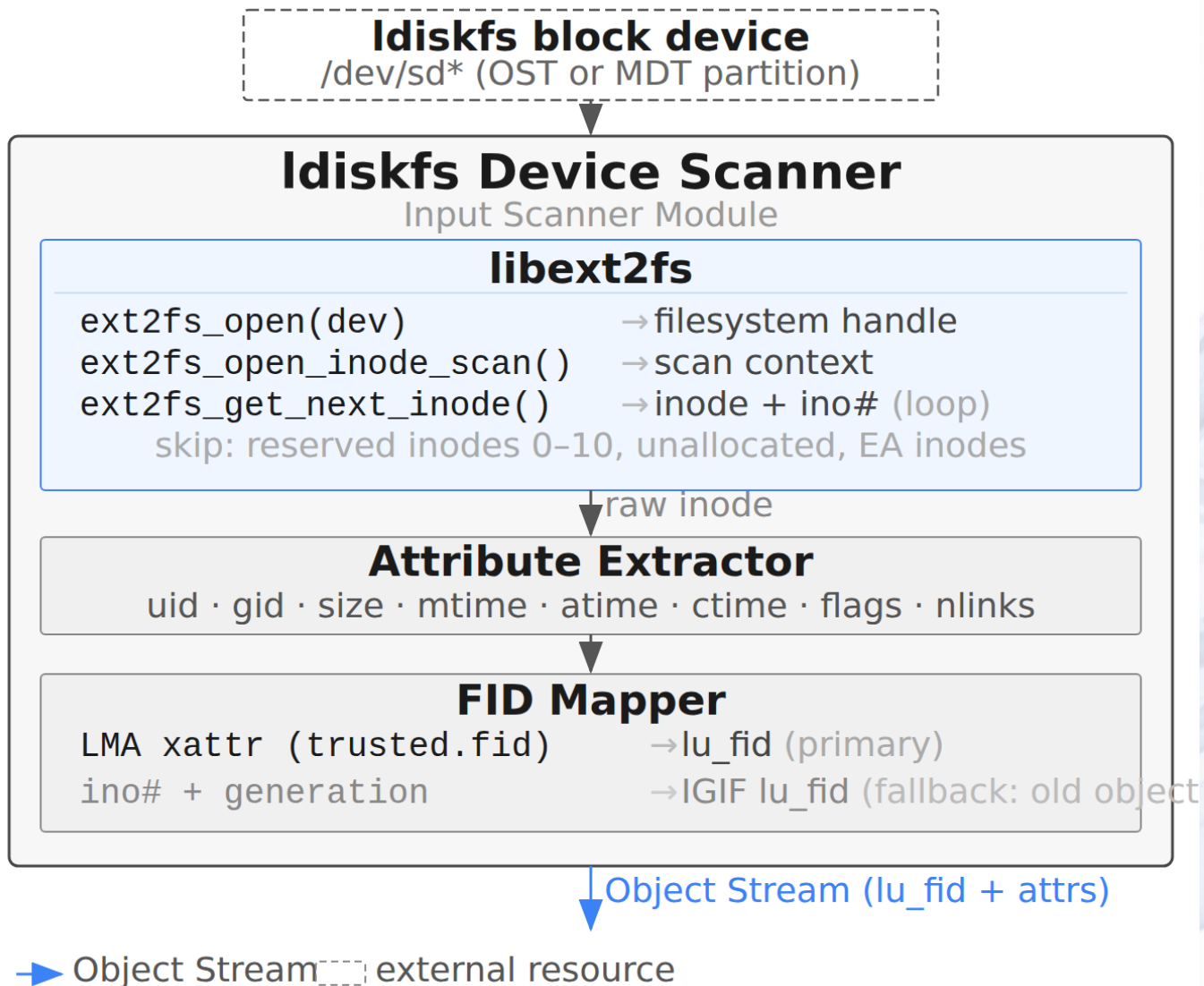


Plan to implement for this release



To be implemented in later release

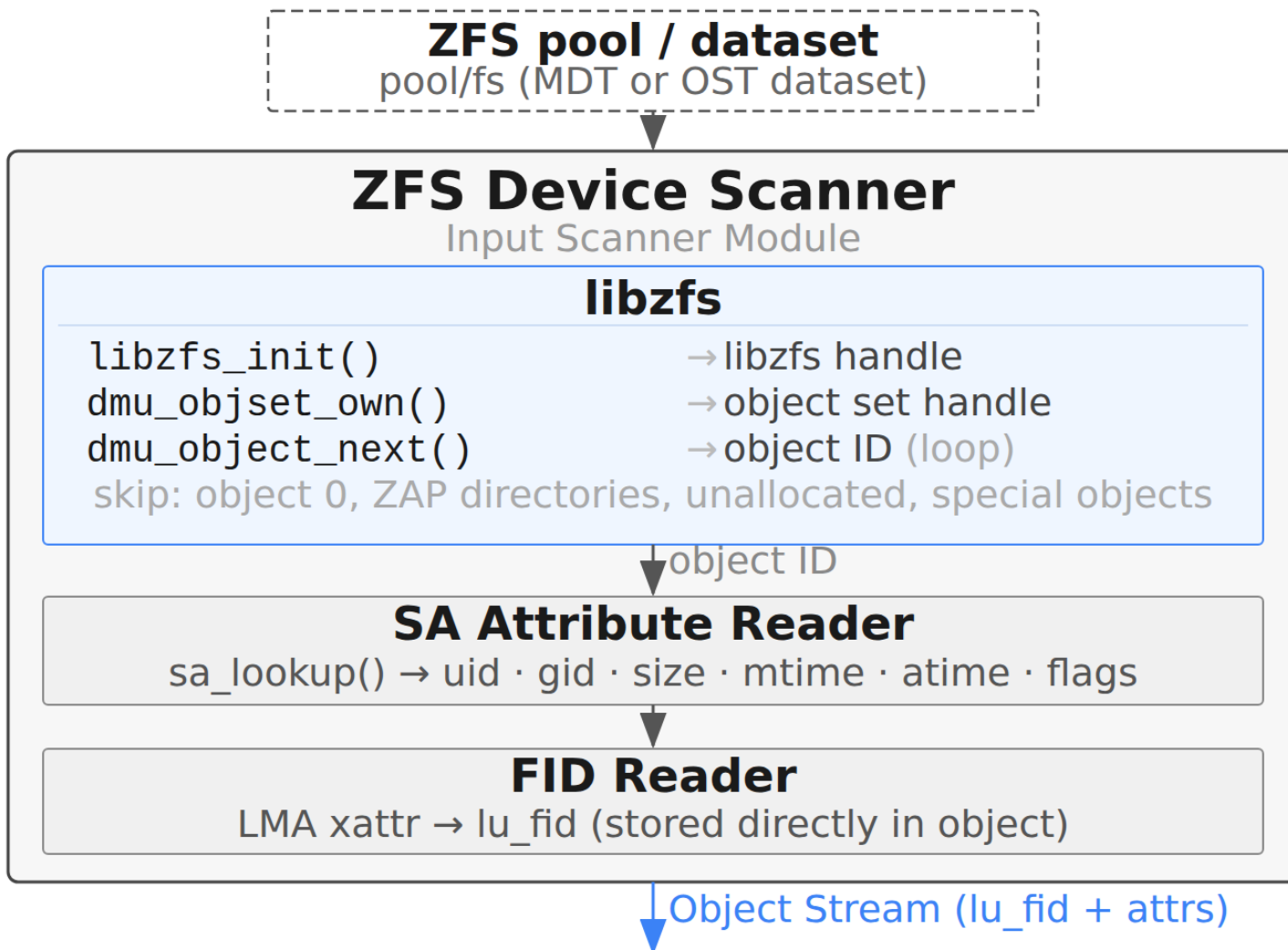
• LDISKFS Device Scanner (LU-20606)



- Uses libext2fs to achieve high-performance MDT and OST object scanning
- Generates LFU Object Stream reading MDT/OST metadata directly from the block device
- Runs in userspace at full device read bandwidth

ZFS Device Scanner

(LU-20613)



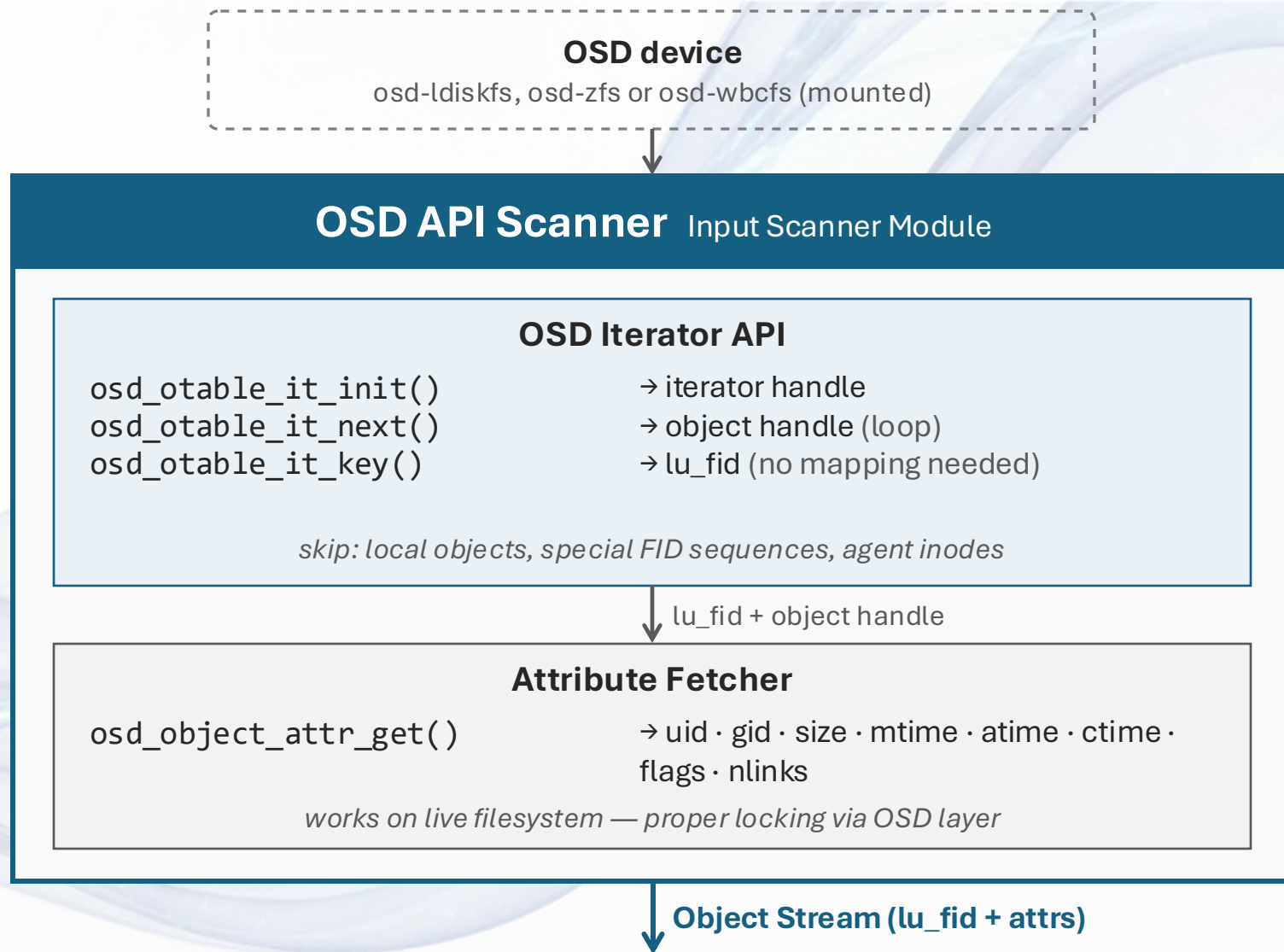
➔ Object Stream external resource (not a module)

- Use libzpool to achieve high-performance MDT OST object scanning
- Generate Object Stream by reading MDT/OST metadata directly from the block device
- Runs in userspace at full device read bandwidth

OSD Device Scanner

(LU-20591)

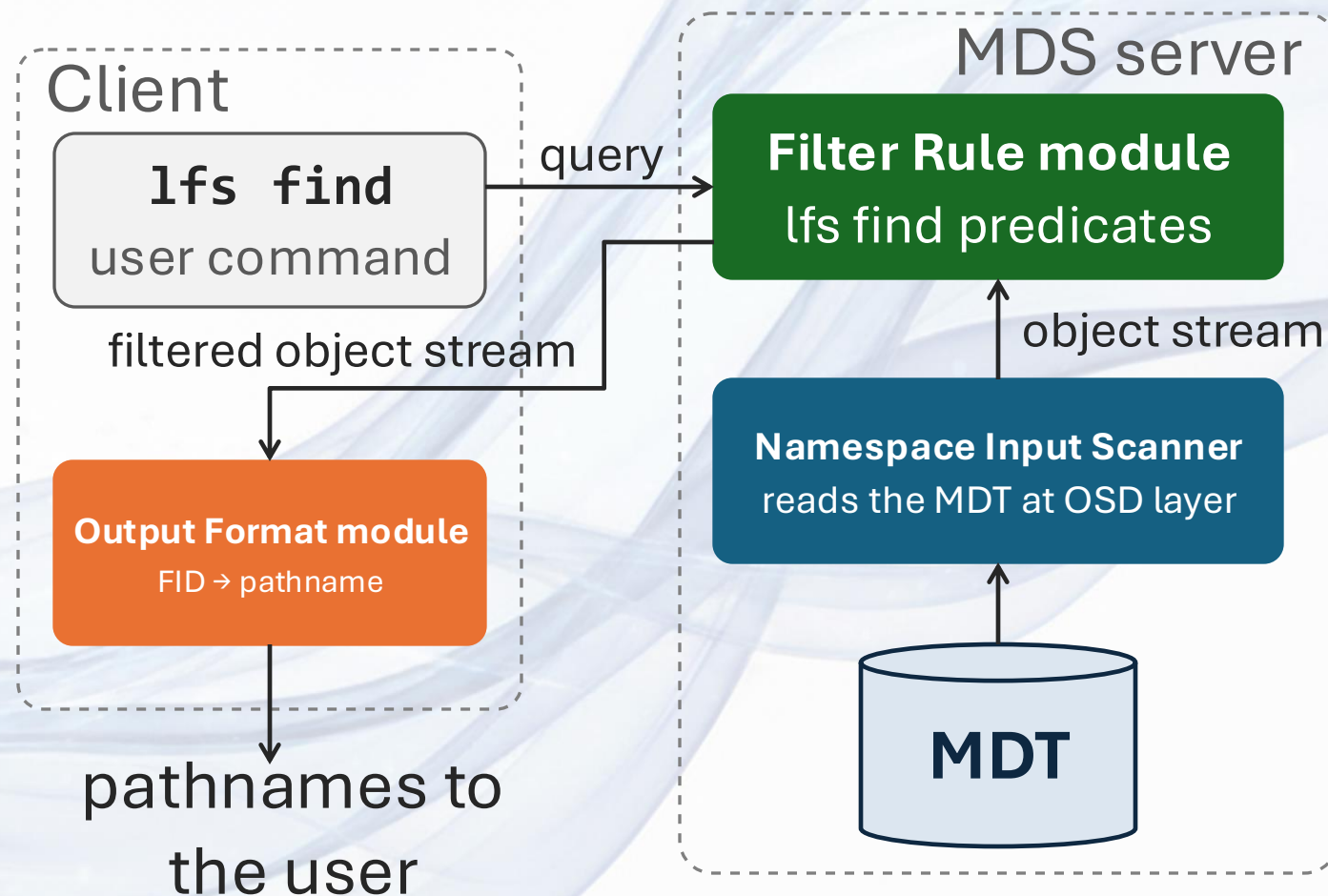
- Scan in kernel via existing OSD iterator
- Return attributes, not just FIDs
- Never directly access block device
- One scan, many consumers
- Better consistency
- Multithreaded scan



Extend well-known utilities and interfaces

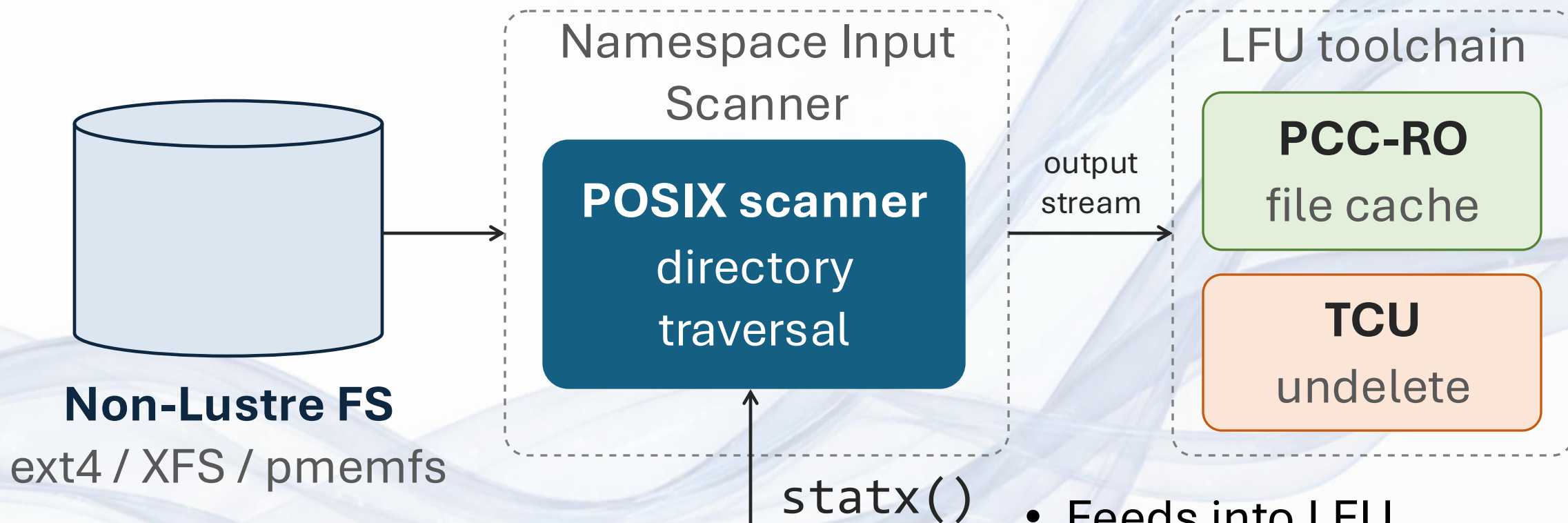
- Early API feedback
- Validates data structures
- Existing tool, new pipeline
- No future technical debt
- First real consumer

Other existent tools can be switched to LFU, so users get new performance with familiar interface.



lfs find becomes the first Object Stream consumer

POSIX Input Scanner



- POSIX directory traversal
- Non-Lustre storage support
- `statx()` gets file attributes
- Unified LFU Output Stream

- Feeds into LFU toolchain
- Enables PCC-RO, TCU
- No duplicate scanner for each scan function

lfs find Server-side Target Scanning

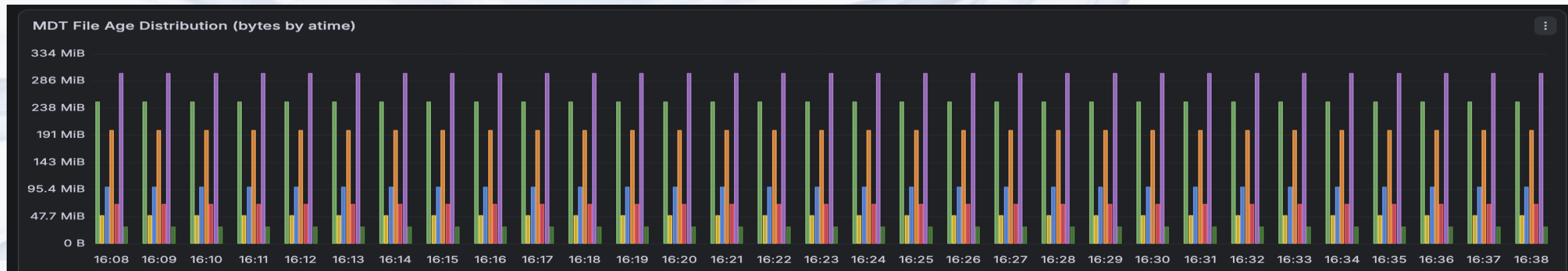
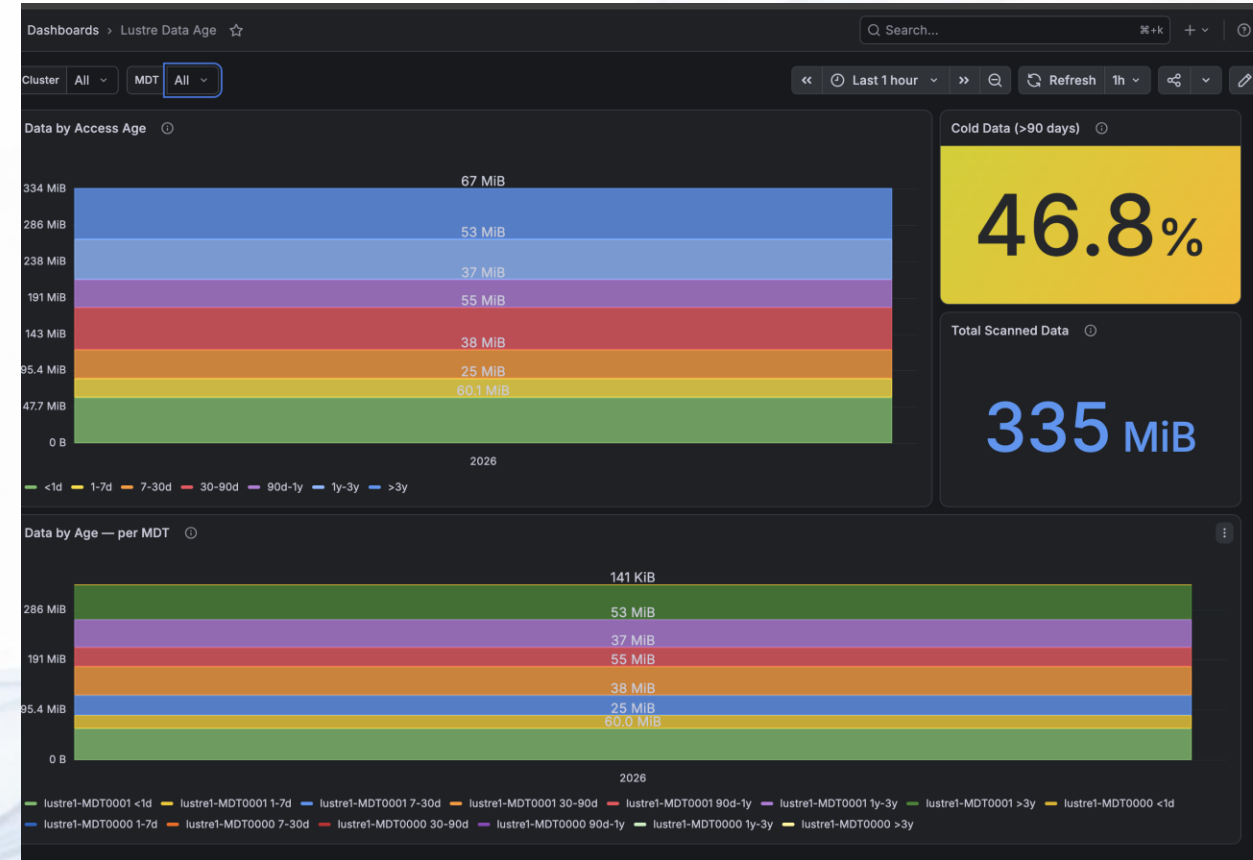
- Server-side scan integrated directly in `lfs find` via `--device`, `--target`, `--fsname`, `--local`, `--internal`
- A user who already knows `lfs find` learns only the new options
- Nothing needs to be mounted, so snapshots and failover partners' LUNs can be searched like serving targets.

<code>lfs find --device /dev/sdb --type f --mtime +30d</code>	scans an unmounted ldiskfs target or image file, matching regular files modified more than 30 days ago
<code>lfs find --target pool/mdt0@snap --blocks +1G</code>	scans unmounted ZFS target or snapshot, matching objects over 1 GiB of allocated blocks
<code>lfs find --local --pool fast</code>	scans mounted target consuming object stream from in-kernel scanner through <code>/dev/lfu_scan</code> , match files on the OST pool named <code>fast</code>
<code>lfs find --device /dev/vdb --internal</code>	explicitly selects the zfs backend and shows options and the full filter list for it, rather than the generic front-end help.
<code>lfs find --fsname testfs --projid 1999</code>	scan the local node's target devices
<code>lfs find /dev/vdb -type f</code>	autodetect block device as path for ease of use

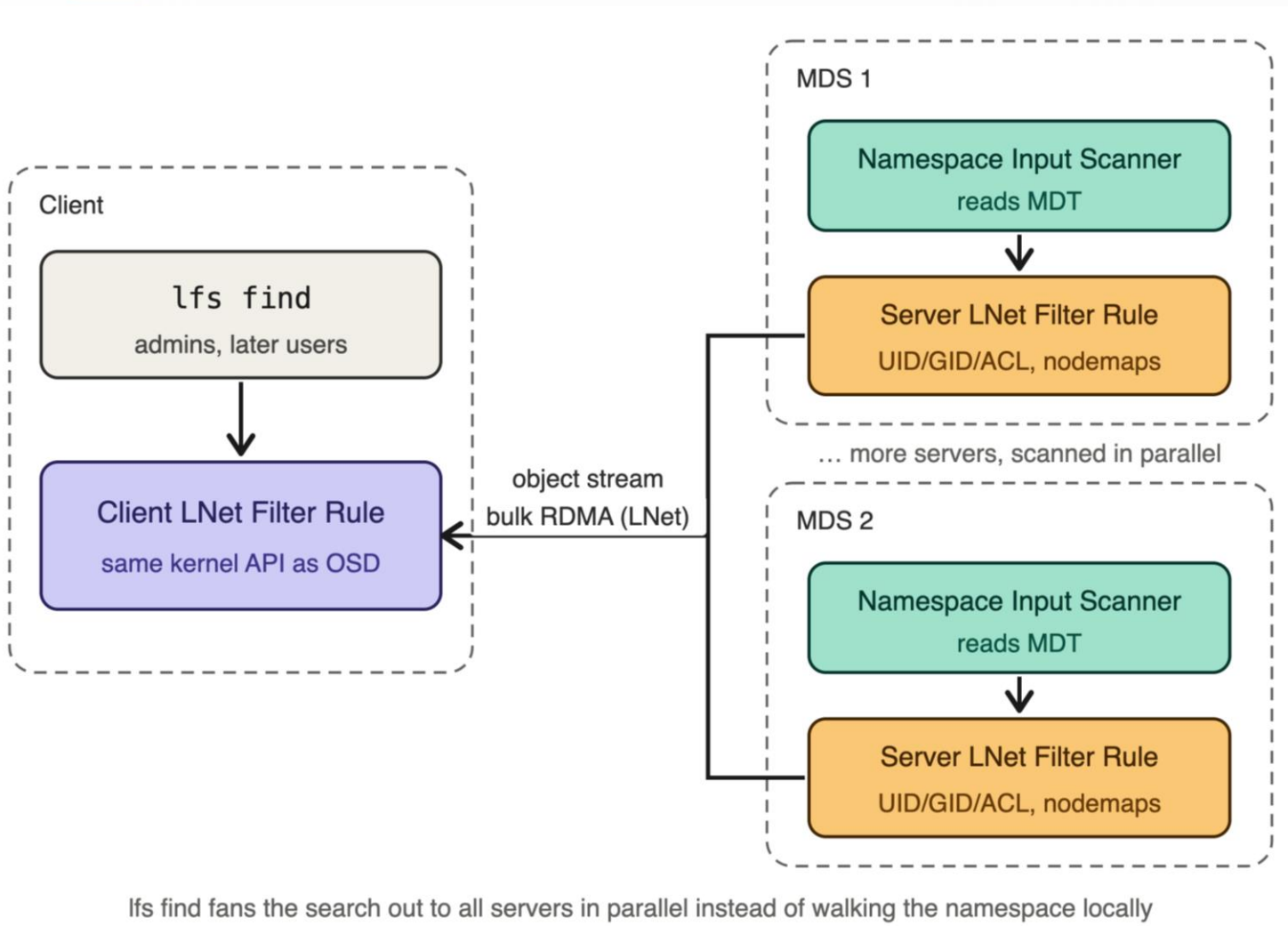


Age Dashboard

- Background MDT age-scanner to otel-lustre uses LFU to scan MDT block devices
- Per-bucket totals as Otel gauges
- Gives operators visibility into how cold their data is across age buckets (<1d, 1–7d, 7–30d, 30–90d, 90d–1y, 1y–3y, >3y).



Scanning from client



- Same kernel API as OSD scan
- Access enforced on server
- UID/GID/ACL, fileset, nodemap
- All servers scanned in parallel
- No local namespace walk
- Only filtered client-server traffic
- Proven by IO500 find offload

Compatibility

Old Servers	• OSD API Input Scanner not be available on an old server • LDISKFS Input Scanner available to directly scan MDT/OST device • No code changes in the Lustre kernel modules • Generate Object Stream by direct scan of LDISKFS device at line rate
Old Clients	• Use Lustre Namespace Input Scanner Module (directory scanner) • Generate Object Stream on old client using existing API for <code>lfs find</code> • Scan directories in filesystem and access object attributes from client • Not faster than running old "<code>lfs find</code>" • Allows old clients to start using LFU APIs to integrate related workflows

Future Development

TBD

- Lots of possibilities for enhancements
- Allow regular user device scans with mandatory UID filter
- Aggregations Filter – min/max/sum, sumsquare for each attribute
 - Generate histograms of attributes across filesystem or per-ID
 - Efficient cache of computed results for future scans
- Live Index for attributes (e.g. old atime or large file space)
- Binding of LFU llapi calls to RustreAPI, Python, ...
- Smart selection of Input Scanner based on requested attributes
 - Namespace vs. device scan if UID/GID/PRJID specified
 - MDT/OST direct scan if --mdt or --ost device/list specified
 - Use Aggregate or Index to filter directory tree walk (e.g. max age or size)
- Welcome input, feedback, contributions from community

Questions?

