Whamcloud

Cache Replacement Policies for Storage Tiering of Lustre

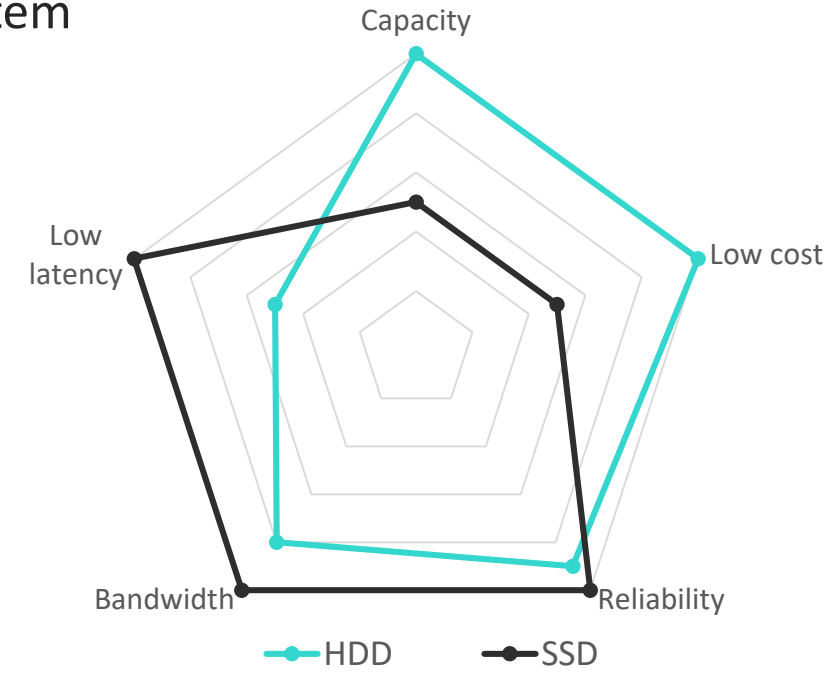
Li Xi

Sept 2019

DDN® STORAGE

Background

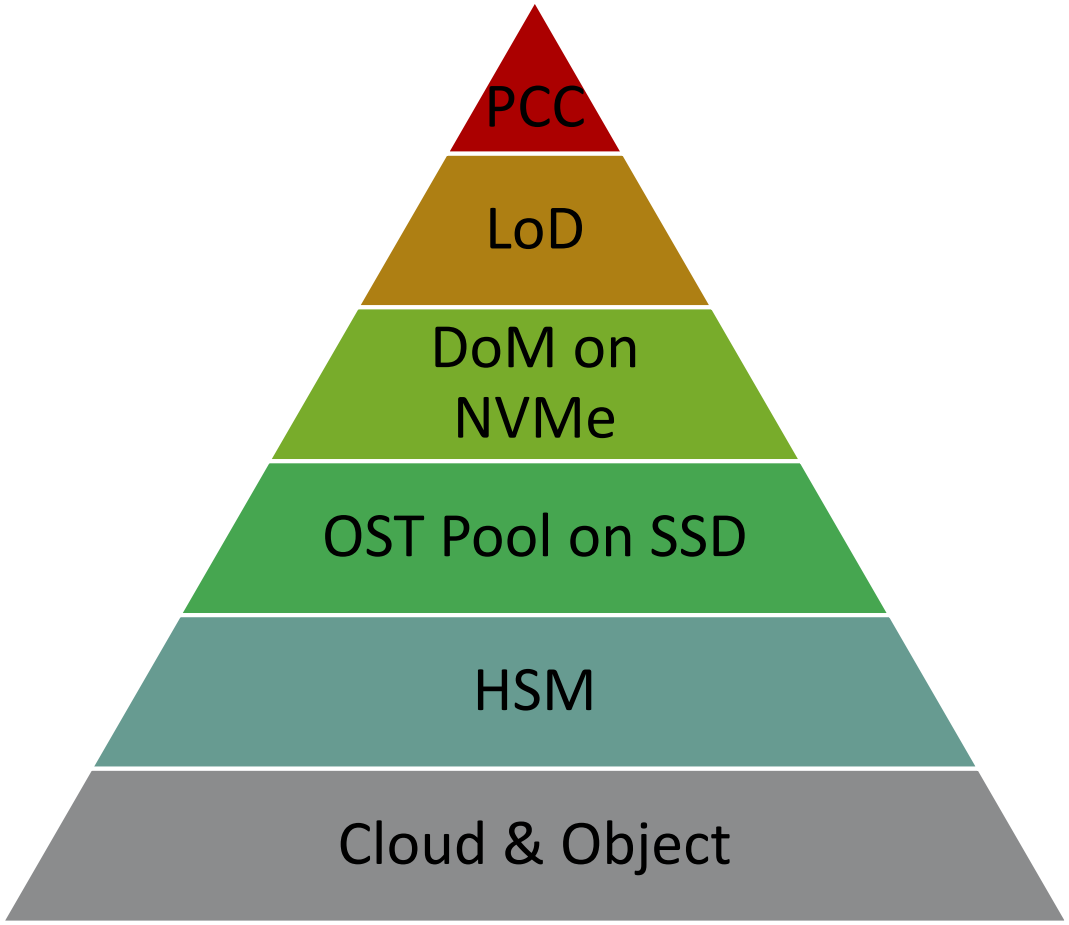
- Lustre architecture is becoming more heterogeneous
- Heterogeneous media are becoming common in a Lustre file system
  - Different specifications: Capacity, Latency, Bandwidth, Reliability, Cost
  - HDD for big capacity
  - SSD/NVME for quick metadata operations
- Different network bandwidths to storage in a Lustre file system
  - Different network bandwidths from a client to different OSTs
  - Extreme condition: Local OSTs on a Lustre client
- Trend: multiple tiering levels inside Lustre
  - Higher performance with acceptable cost
  - Better QoS (Quality of service) guarantee
  - Utilize storage locality
  - Move the storage closer to compute
  - Promote the entire efficiency of the storage system



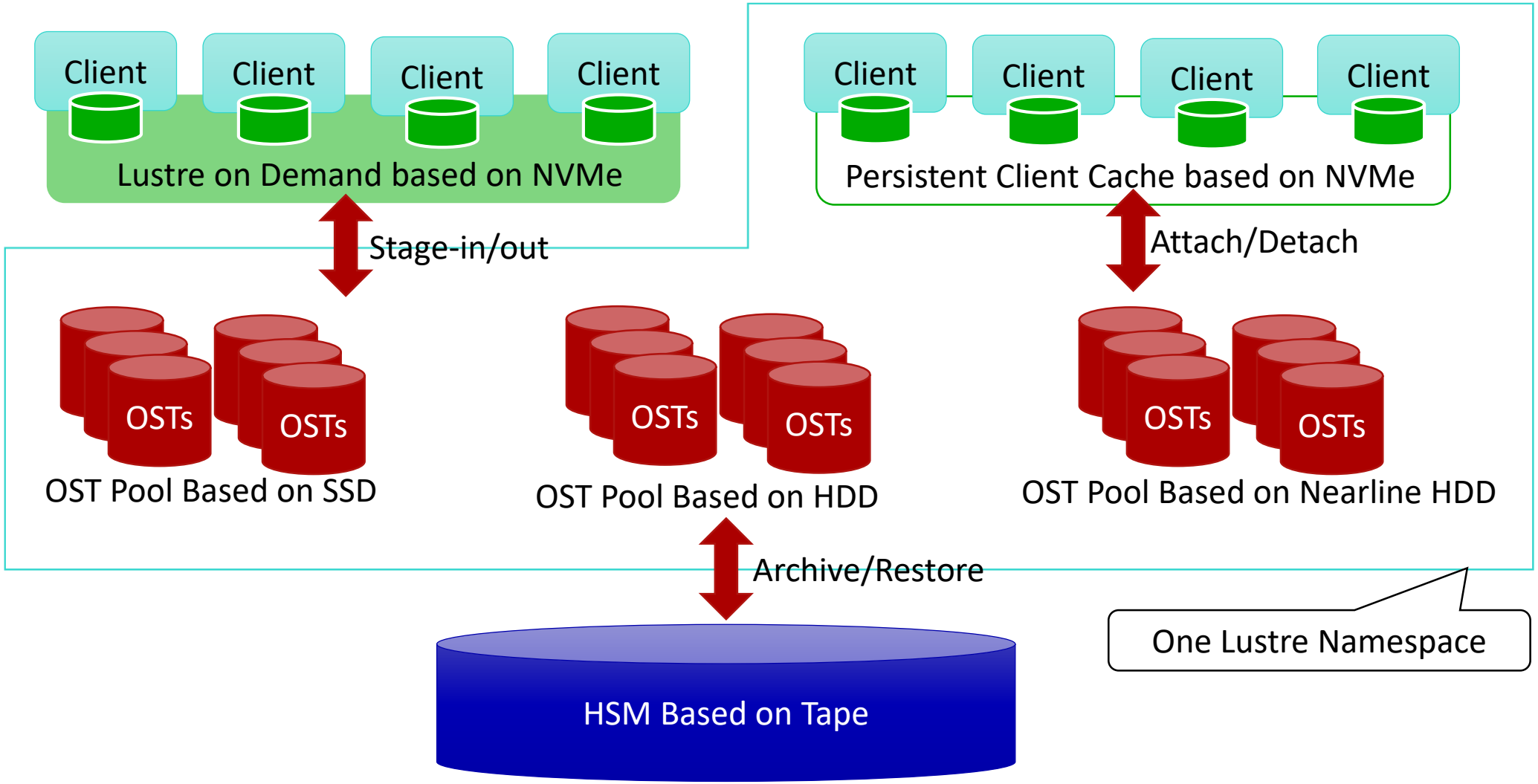
Multiple Tiers of Lustre file system

Whamcloud

- Persistent Client Cache(LU-10918)
  - Same namespace
  - I/O pattern limitation
- Lustre on Demand
  - Separate namespaces
  - Integration with job scheduler
- Data on MDT with SSD/NVMe storage
  - Size limitation of MDT
- OST pool based on SSD for cache
  - Same namespace
- HSM storage
  - Same namespace
  - Transparent access to archived data
- Data movement between Lustre and Cloud/S3
  - Separate namespaces
  - WAN connection



Example architecture of a tiered Lustre file system



Requirements for Data Management between Tiers

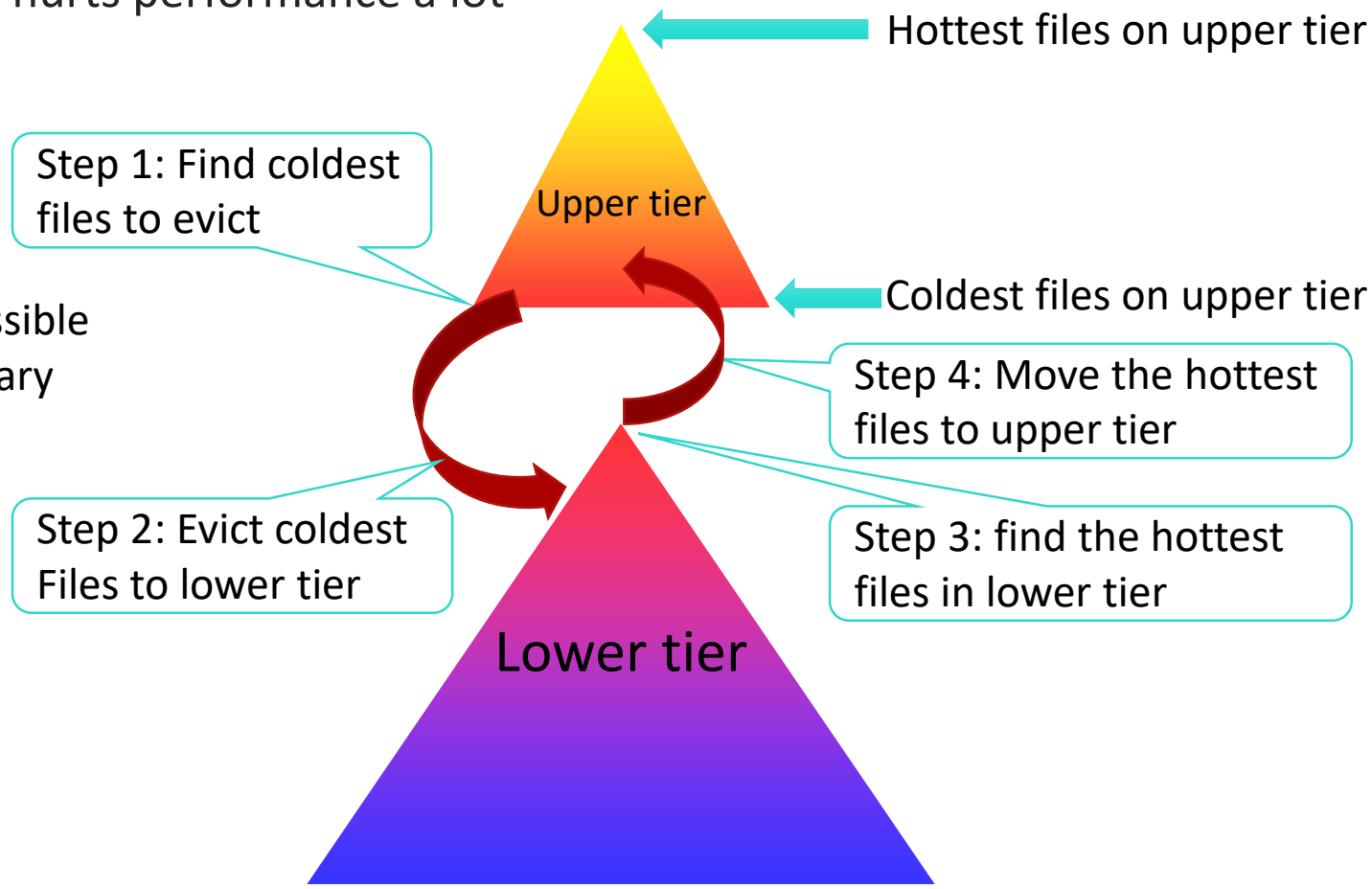
Whamcloud

- Data placement and location at the creation time
  - PCC: rule-based policies to determine whether to create data on PCC directly
  - Lustre on Demand: new files of the running job
  - DoM with SSD/NVMe: stripe configuration inherited from parent
  - OST pool: Data Placement Policy mechanism (LU-11234) for rule-based policies
- Data movement mechanism between tiers
  - PCC: reuse HSM copytool
  - Lustre on Demand: cp or MpiFileUtils
  - DoM with SSD/NVMe: lfs migrate
  - OST pool: lfs migrate
- Find the correct data to move between tiers
  - The hottest file to keep in quick tiers
  - The coldest file to evict from quick tiers
  - Cache replacement policy is very important

Why Cache Replacement Policy is Important?

- Cache replacement of Lustre tiering is expensive
- Bad cache replacement hurts performance a lot

Key principles:
1. Keep upper tier as full as possible
2. Only evict data when necessary
3. Evict as little as possible



A Quantitative Analysis of Cache Replacement Effect

Whamcloud

- Perf[upper]: Performance of upper tier (Bytes/s)
- Perf[lower]: Performance of lower tier (Bytes/s)
- Access[cold]: The access amount of the evicted cold data (Bytes)
- Access[hot]: The access amount of the fetched hot data (Bytes)
- Overhead: The time overhead because of the cache replacement amount of the fetched hot data (Seconds)

Saved time because of cache replacement:

$$\frac{Access[cold]}{Perf[upper]} + \frac{Access[hot]}{Perf[lower]} - \frac{Access[hot]}{Perf[upper]} - \frac{Access[cold]}{Perf[lower]} - Overhead$$

$$= \frac{(Access[host] - Access[cold]) \bullet (Perf[upper] - Perf[lower])}{Perf[upper] \bullet Perf[lower]} - Overhead$$

Conclusions of the Quantitative Analysis

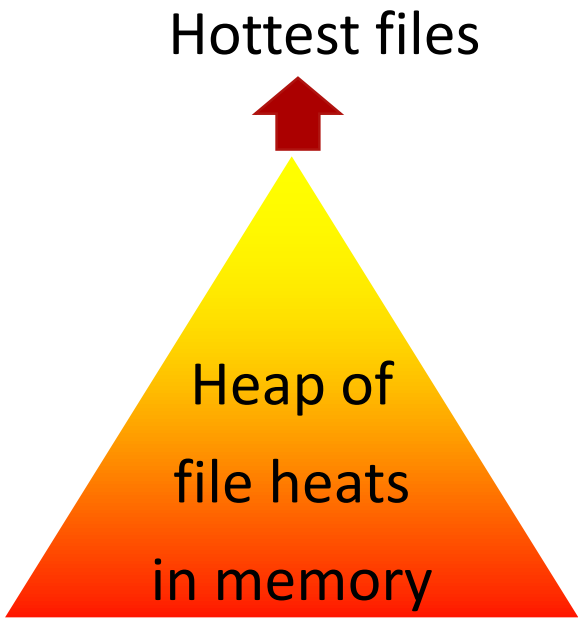
Whamcloud

- Performance difference between storage tiers should be huge
  - Data movement between HDD tier and NVMe is likely to be worthwhile
  - Data movement between busy OST and idle OST with the same media might not be worthwhile
- Access amount difference between the evicted data and the fetched data should be huge
  - Evicted data should be as cold as possible
  - Fetched data should be as hot as possible
  - Finding the coldest/hottest data is important!
- Reducing data replacement overhead improves cache efficiency immediately
  - The process of finding the coldest/hottest files need to be quick
  - Parallel data copy/removal to reduce overhead of data movement
- Bad replacement is much worse than no replacement if data movement overhead is large
  - Choosing the correct data to move is extremely important

Why Is It Hard to Find the Coldest Files to Evict?

Whamcloud

- Millions/billions of files to manage
  - The size of file list exceeds memory size
  - In-memory algorithms (LRU/heap) are not applicable
- The distribution of access time can't be predicted precisely
  - "Files not accessed for a given time period" might find too many/few files
- Coldest files might never be accessed since the beginning
  - Files that have ever been accessed might not the coldest
  - Need full scan of the whole tier
- Low time cost is critical
  - Small finding overhead benefits cache efficiency a lot
  - Entire scanning of the tier could take minutes or hours
  - Policy engines needs O(N) time to scan all files, thus is too expensive
  - The implementation needs to be O(1) time



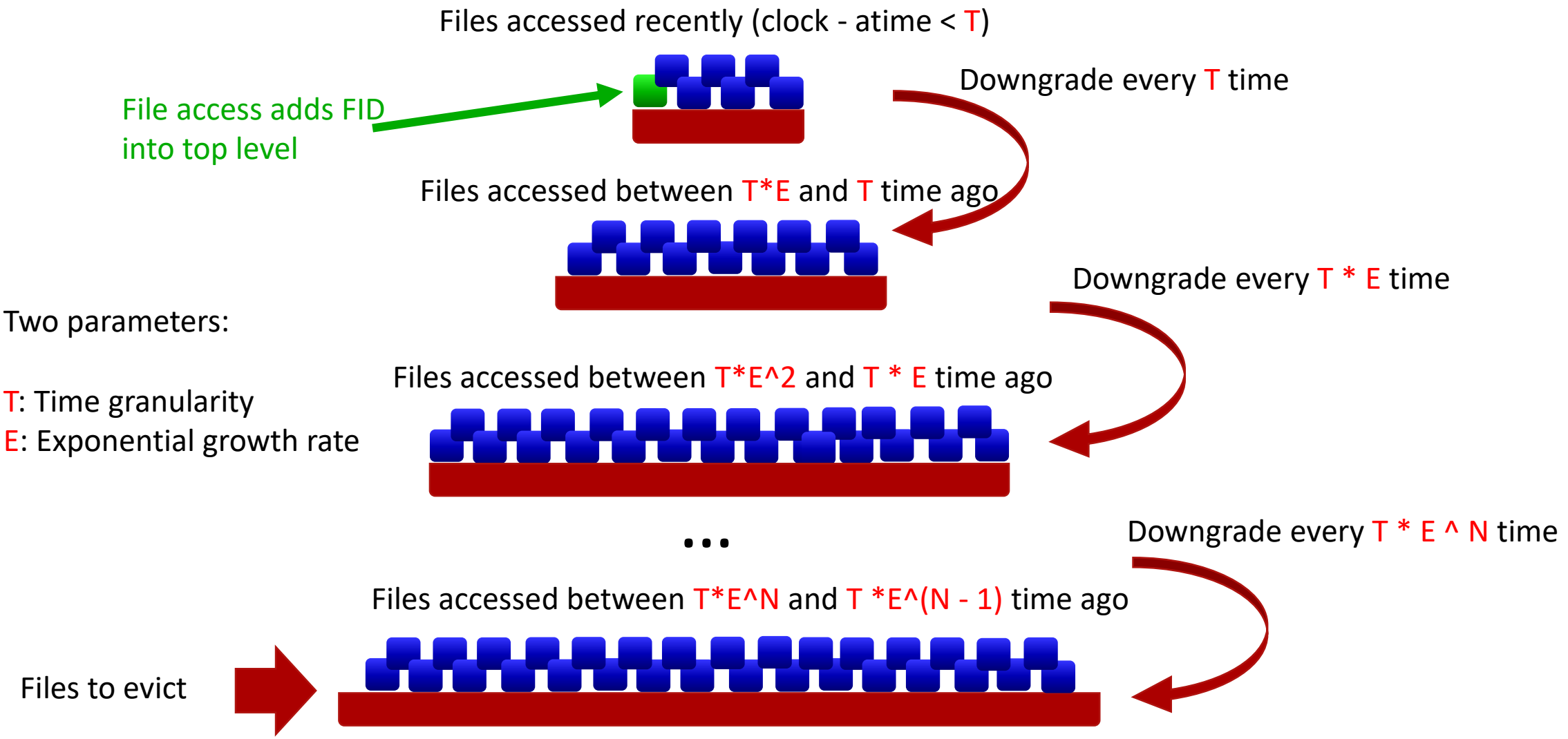
Hottest files are easier to find

Solution: LCMP (Lustre Cache Management Policy)

Whamcloud

- https://github.com/DDNStorage/lcmp
- Maintain a on-disk structure of file lists
  - LCMP can support Lustre with billions of files
- The structure includes multiple levels
  - Level 0 includes the most recently accessed files
  - The lower levels include files that have not been access for a long time
- The structure is synced from time to time with Lustre file system
  - Lustre Changelog indicates what files have been accessed recently
- Time is divided into epochs
  - 1T, 2T, …, i * T, …
  - At the end of each epoch, file list from top level downgrades to the lower level
- Each level has different time epoch of downgrading
  - Lower levels have longer time epoch
- The coldest files can be found in the bottom level within O(1) time

General Idea of LCMP: LRU levels

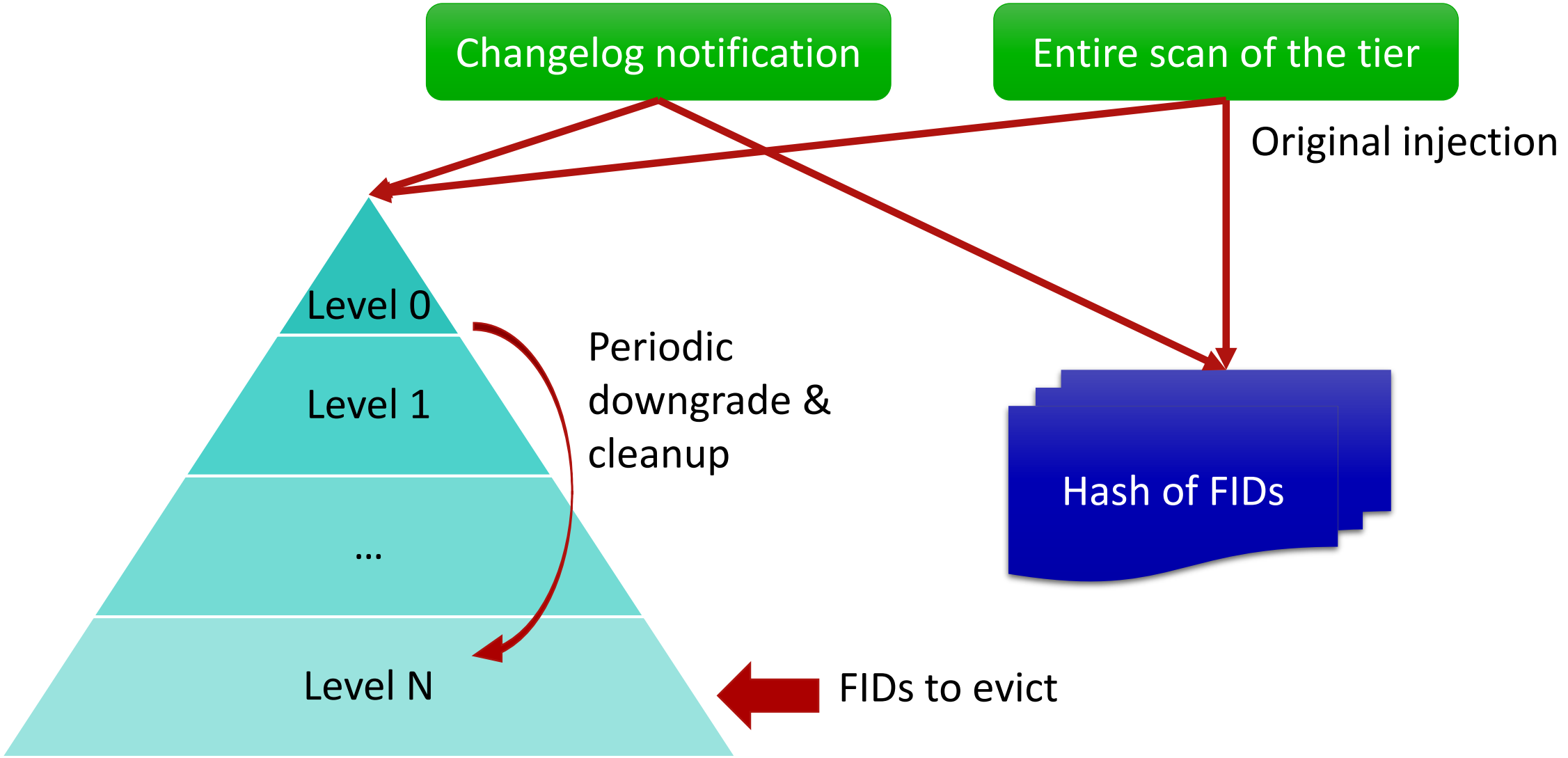


Implementation of LCMP

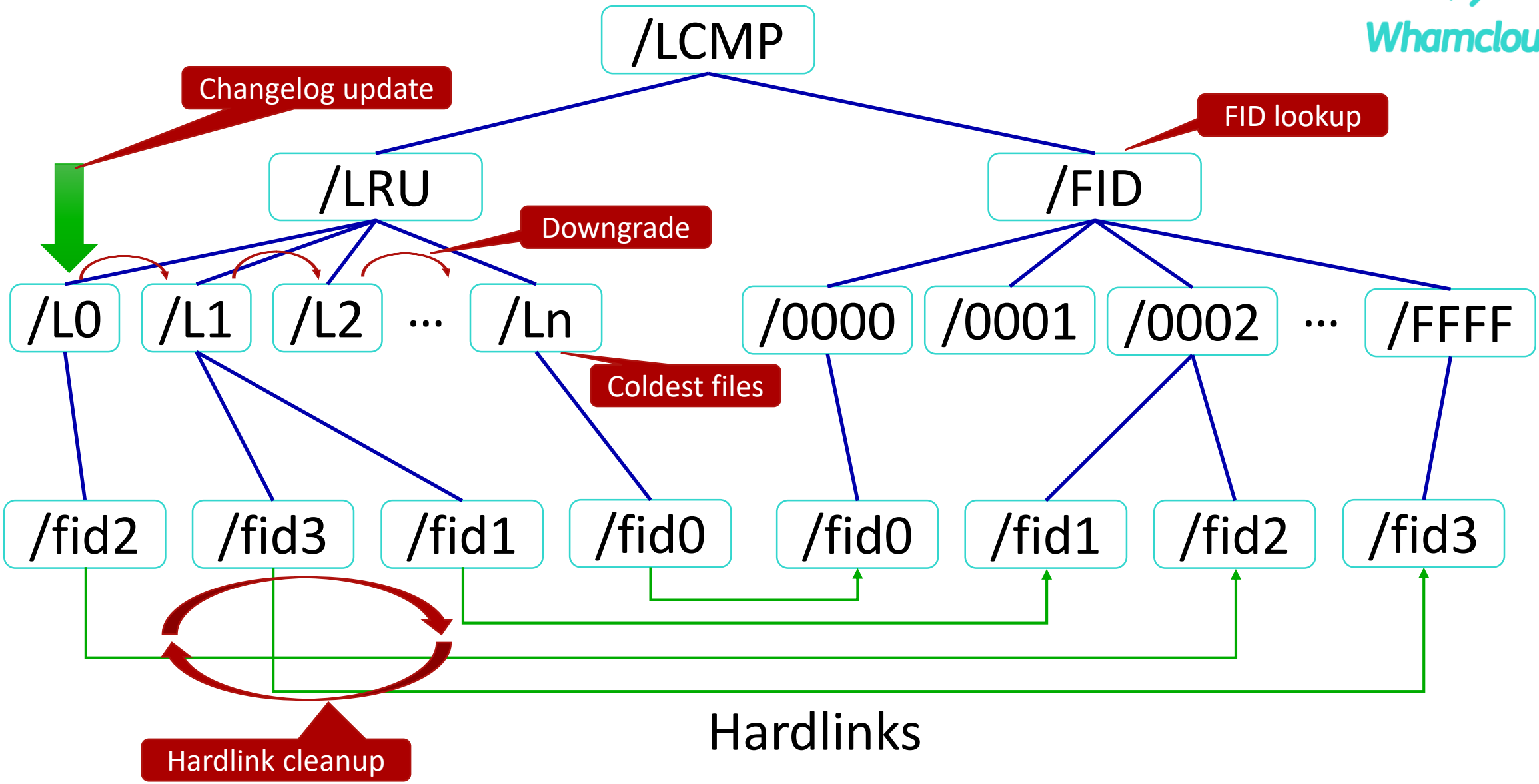
Whamcloud

- Use directory tree to save the data structure
- Maintain a FID directory tree to map from FIDs to inodes
  - All Lustre files has a corresponding file in the FID directory tree
  - Hash of FIDs are used as the directory path names
- Maintain a LRU directory tree to track LRU lists of FIDs
  - Each level of LRU is a directory under the LRU tree
  - Each level directory contains a bunch of hardlinks pointing to FID inodes in FID directory tree
  - The FID hardlinks of level 0 directory are generated by digesting Lustre Changelogs
  - When the epoch end of this level is reached, parent directory of all hardlinks will be moved to the next level
- Hardlink number indicates whether the FID has been accessed more recently
  - If hardlink number of a FID in the bottom level is larger than 2, then the file should have been accessed more recently
  - Hardlinks with more recent access should be removed from that level

Implementation of LCMP



Directory tree of LCMP



Advantages of LCMP

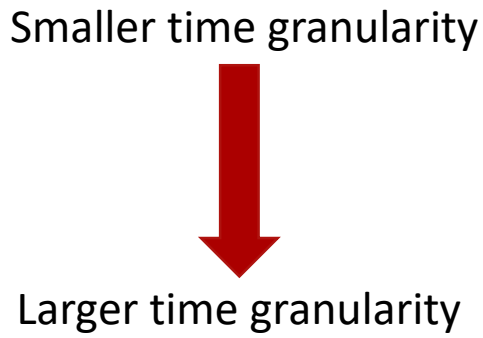
Whamcloud

- Good scalability
  - The needed maximum inode number is the same with the Lustre file number
  - The needed space per inode is small, because all files are empty
  - A device with the same size of MDT should be enough
  - LCMP can support large MDT with billions of files
  - LCMP can scale well for DNE by creating a separate directory for each MDT
- High performance
  - The coldest files can always be listed within O(1) time
  - LCMP is not I/O intensive, so no need to use fancy storage
- Easy to configure
  - Do not need to predict the distribution of access time precisely
  - Time granularity (T) of 10 seconds and exponential growth rate (E) of 2 should be suitable for most use cases
- Wide applicability
  - The same tool for different types of tiers (PCC, LoD, DoM, OST pool and HSM)
  - No requirement for Lustre version, as long as Changlog is supported

Future Optimization of LCMP

Whamcloud

- Entire scan of the file system for original injection
  - Walking through whole directory tree is an easy but general solution
  - Policy engines are good candidates for quick injection
- File filtering for smarter policies
  - Filter unnecessary files in Changelog records
  - UID/GID/ProjID/JobID filtering to avoid evicting VIP's data
- Customization and optimization and for different Lustre tiers
  - PCC: Need to implement notification mechanism in local PCC storage, e.g. Linux inotify
  - LoD: Need to integrate with job scheduler to predict I/O patterns for file filtering
  - DoM: Need to filter files that are not DoM
  - OST pool: Need to filter files that are not on quick OST pool
  - HSM: Need to filter files that are already archived or should never be archived



Conclusions

Whamcloud

- Multiple tiering levels inside Lustre are becoming common
- Cache replacement policy between layers is very important
- Finding coldest file to evict quickly is important but not easy
- We designed and implemented a tool to quickly find the coldest file: LCMP
  - O(1) time to list the coldest file

Whamcloud

DDN®
STORAGE