DDN®
STORAGE

Lustre Integrated Policy Engine

Li Xi

DataDirect Networks

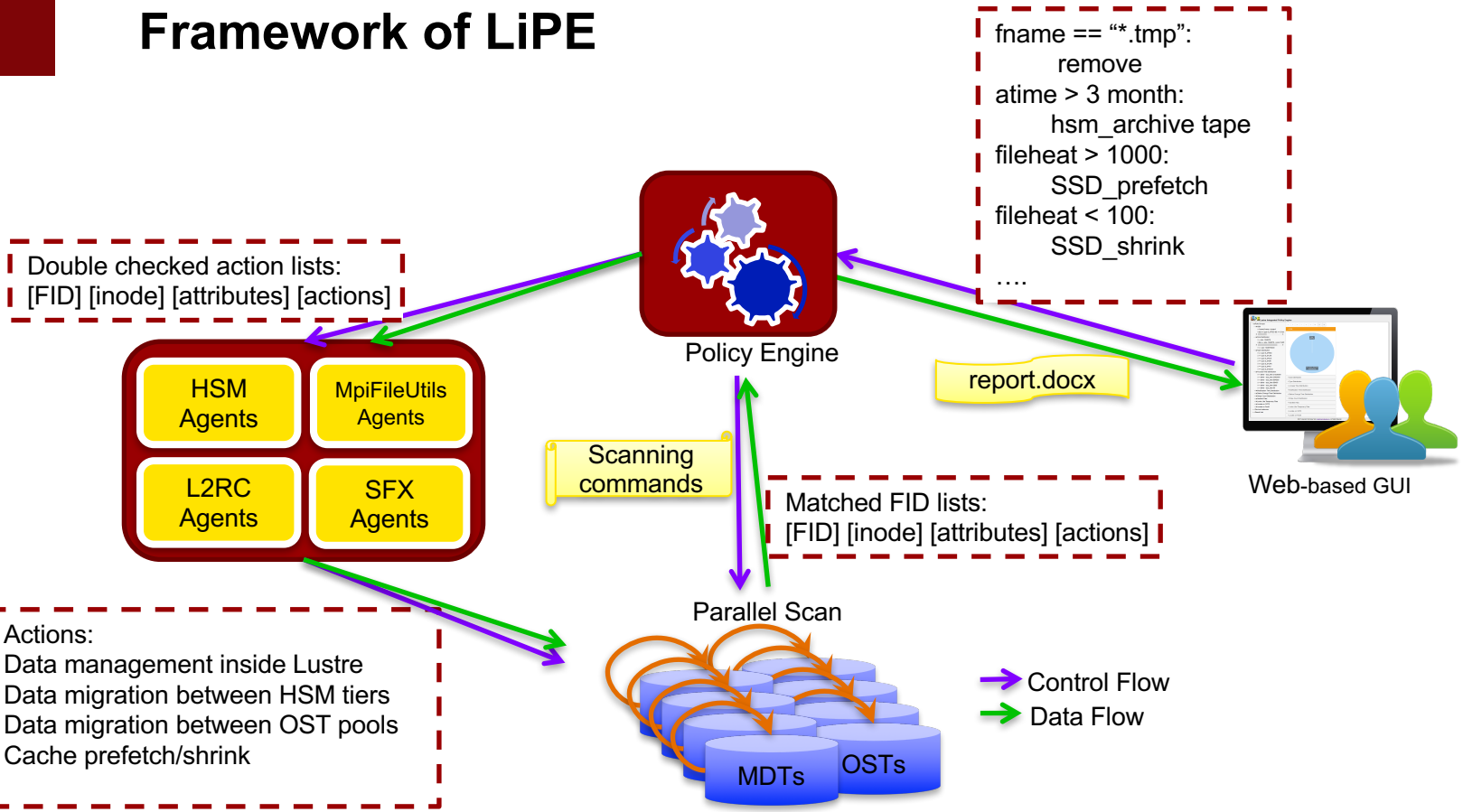
Why LiPE?

- **Administrators are short of tools to do data management on Lustre/Storage**
- **Writing simple scripts without Lustre internal knowledge are far from enough**
  - Not easy to achieve high speed.
    - Scanning directory tree is not efficient
  - Not able to extract Lustre attributes
    - Stripe information
    - HSM status
    - Link EA
- **LiPE is a policy engine which knows Lustre well**
  - Optimized for Lustre: scanning Ldiskfs device directly
  - Understands Lustre: is able to extract all Lustre attributes saved on disk
  - Powerful & flexible

Advantages of LiPE

- **Scans MDTs directly**
  - No need to add extra server or storage
  - No need to duplicate the metadata
  - No need to sync based on Lustre Changelog
  - No need to do initial scan for data injection
  - Quick scanning because MDTs are usually based on SSDs
  - Can scale up with DNE by adding more MDTs and MDS
- **Precise definition of rules**
  - Precise arithmetic expressions are used in rules to define the exact behavior of policies
    - Avoid vagueness that causes misunderstanding
    - Avoid subtle changes of semantic implication between versions
- **Easy to setup and configure**
  - Only one user-space RPM is needed
  - Web-based GUI is provided to help administrators to:
    - Configure policy rules with tips and correctness checking
    - Choose apply policies
    - Get graph-format reports
- **Multiple Lustre file systems for different purposes can be managed together in a single LiPE system**

Framework of LiPE



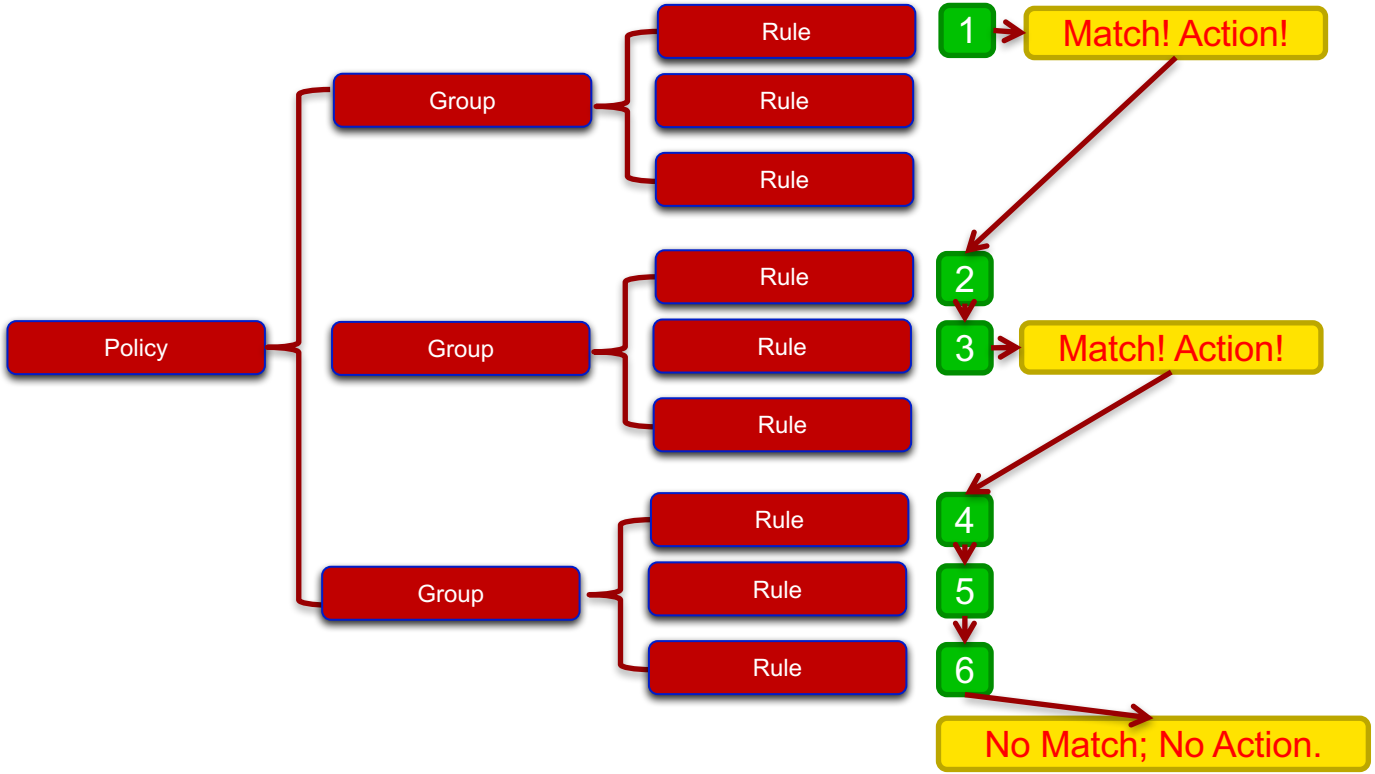
Design of LiPE (1)

- **Expression of rule**
  - Expression: A arithmetic expression in the form of Polish notation that has a value of unsigned 64-bit integer
  - Operators
    - Arithmetic operators: +, -, *, /, %
    - Relational and logical operators (==, !=, >, >=, <, <=)
    - Bitwise operators: &, |, ^, <<, >>
  - Unsigned 64-bit integers could be used in the rules.
  - Constants could be used in the rules, e.g. the Lustre internal constants
  - System attributes could be used in the rules, e.g. date time, free inode number, free disk space, etc.
  - Object attributes could be used in the rules, e.g. atime, mtime, ctime, size, mode, uid, gid, blocks, type, flags, nlink, rdev, blksize, hsm stat, etc.
  - Functions
    - fname($ARG)emanf: Whether the dentry name matches with regular expression $ARG
    - ost($ARG)tso : Whether the file locates on OST(s) with regular expression $ARG
    - pool($ARG)loop: Whether the file locates on OST pool(s) with regular expression $ARG
    - …
- **Example**
  - The inode should be regular file that was accessed earlier than one year ago
  - && == type S_IFREG < atime - sys_time 31536000000

Design of LiPE (2)

- **Action of rule**
  - Counter increase action: LAT_COUNTER_INC
  - File removal action: LAT_COUNTER_REMOVE
  - HSM actions : LAT_HSMA_*
  - Classification based on UID/GID/HSM state: LAT_COUNTER_CLASSIFY
  - Project quota action: LAT_SET_PROJID
  - Ladvise actions: LAT_LADVISE_*
  - Set the inode immutable: LAT_IMMUTABLE
- **Rule group**
  - One or more rules could be gathered as a rule group with order
  - If an rule in a group is evaluated as matched, the rest rules in that group won't be matched
- **Multiple rule groups could be defined**
  - A group to summarize size distribution
  - A group to summarize access time distribution
  - A group to summarize trigger HSM actions
  - A group to summarize trigger backup actions
  - …

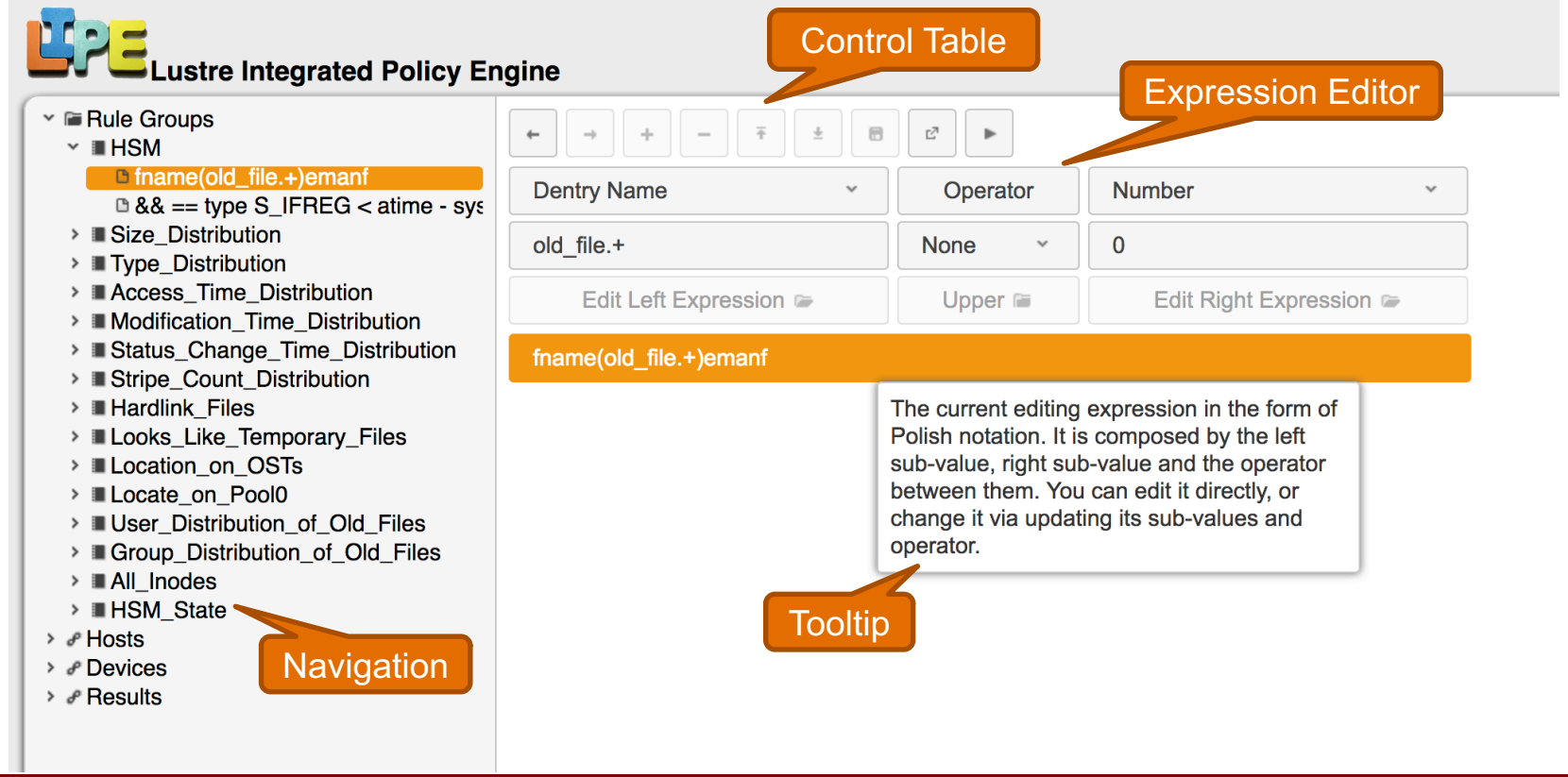
Evaluation of rules in LiPE



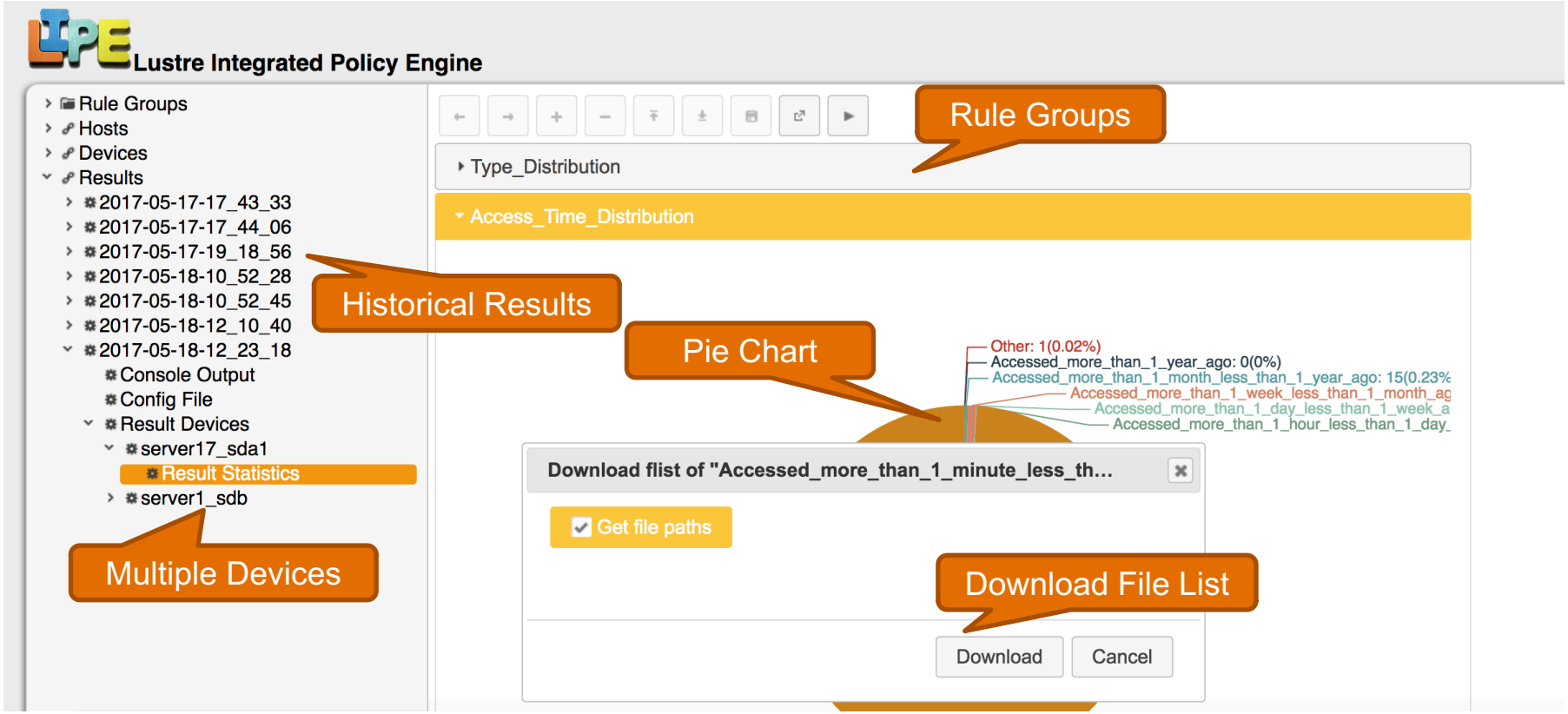
LiPE components

- **lipe_web**
  - “lipe_web” is a web-base GUI for administrators to configure rules, run policies and get reports
- **lipe**
  - “lipe” is a tool that scans the MDT and check whether the objects match rule groups
  - FID lists of matched files will be printed for rule groups
- **lipe_flist_handle**
  - “lipe_flist_handle” is a tool that carries out the actions on the FID lists printed by “lipe” command
- **lipe_launch**
  - “lipe_launch” is a tool that launches the whole process of scanning of MDTs, applying of actions on the file lists, and generating reports
  - “lipe_launch” can be configured in “crond” to schedule repetitive LiPE tasks
- **lipe_agent_manager**
  - “lipe_agent_manager” is a tool that manages the HSM copytools
- **HSM tools: “lipe_hsm_remover”, “lipe_hsm_check”, etc.**

GUI of LiPE (1)



GUI of LiPE (2)



Problem & solution (1): file size on MDT

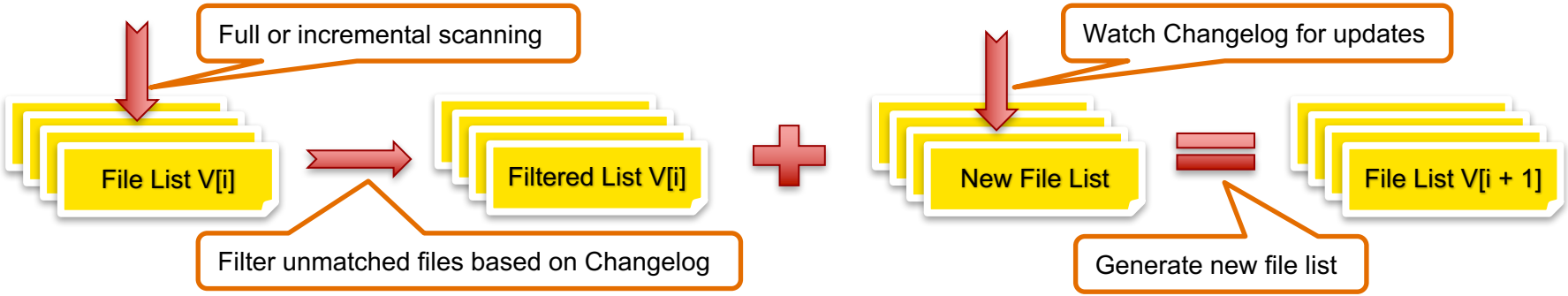
- **Problem: File sizes on Lustre MDTs are always zero**
  - LiPE is not able to apply rules based on correct file sizes
- **Solution: File size on MDT**
  - Implement with guarantee of eventual consistency
  - A new extended attribute for file size on MDT
  - File size on MDT will be synced
    - when the last file close finishes
    - when a significant time has been past since last sync
- **Strong guarantee of file size on MDT takes too much cost**
  - Too hard to guarantee synced file sizes between MDT and OSTs
- **Strong guarantee of file size on MDT is unnecessary for most cases**
  - Smart administrators will leave enough margin of data management
  - Most management actions can be withdrawn without losing any data
  - Data removing are usually double/triple checked before being committed
    - File sizes could be checked on client before applying unrecoverable actions

Problem & solution (2): RAoLU policy

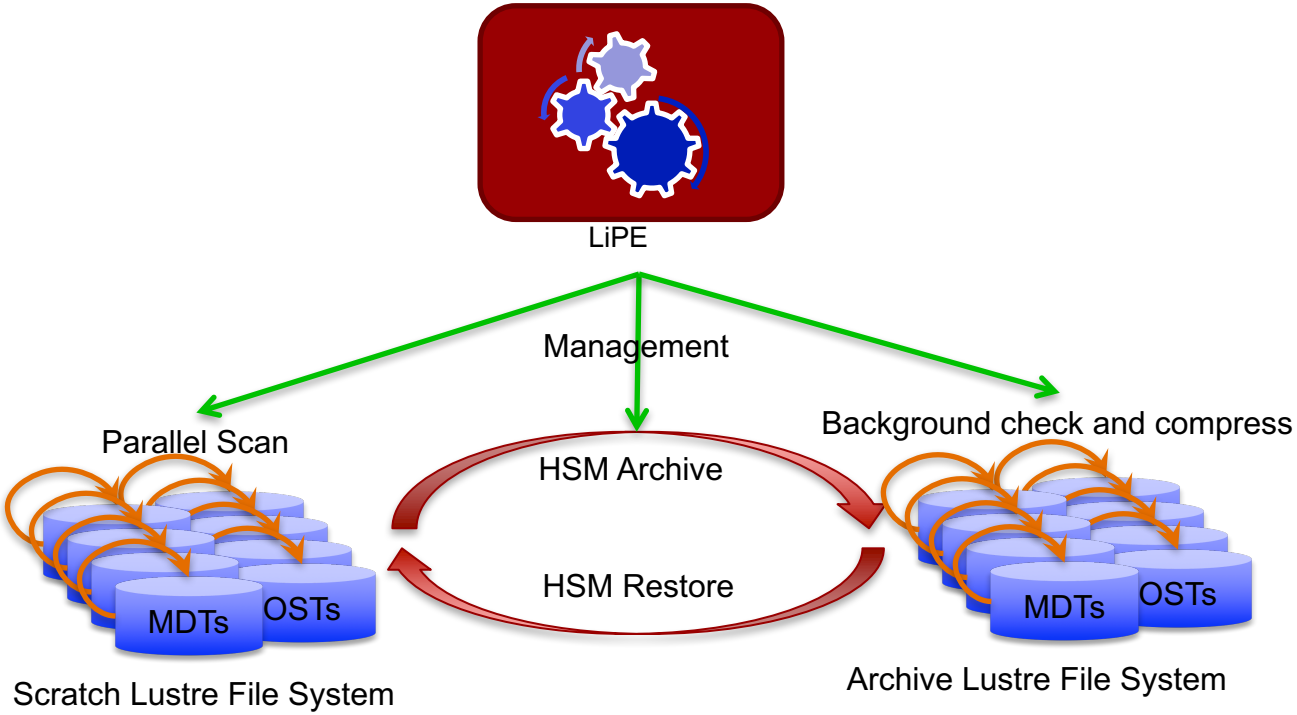
- **Problem: When an archived file on HSM is being removed from the file system, a HSM remove request is not triggered**
  - Space on HSM storage will leaked if Lustre Changelog is ignored
- **Walkarounds:**
  - Tool “lipe_hsm_remover” watches Lustre Changelog and remove unlinking files from HSM storage
  - Tool “lipe_hsm_check” scans HSM storage regularly to cleanup orphan HSM files
- **Solution: patch from Bruno Faccini**
  - LU-4640 mdt: implement Remove Archive on Last Unlink policy

Problem & solution (3): Incremental scanning

- **Problem: LiPE currently always do full scanning of MDT**
  - Repeated scanning with small time interval is not efficient
- **Solution: incremental scanning based on Changelog**
  - Lustre Changelog will be watched
  - New matched file will be added into the file lists
  - Unmatched file in the old file lists will be filtered



Use case (1): Use LiPE to manage HSM



Use case (2): Use LiPE for file system report

- A report includes statistical charts of percentages based on either disk usages (implementing) or inode numbers
- All file lists can be downloaded for further check
- File lists can be pre-sorted based on UID/GID or size or any other attributes (implementing)
- A .docx format report can be downloaded (implementing)

Benchmark results of LiPE

- **Test environment**
  - A single SSD based MDT with read speed of about 549 MB/s: Samsung SSD 850
  - Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
- **Fully cached scanning speed**
  - 59 million inode/s
  - CPU is the limitation
  - 0.76KB memory cache is needed for each inode
- **None cached scanning speed**
  - 1.8 million inode/s
  - Disk speed is the main limitation of scanning speed
  - CPU usage is high but not the bottle neck of performance
  - On production system, LiPE should limit its CPU and disk bandwidth usage to avoid impacting Lustre service

Potentials of LiPE

- **LiPE + ZFS (Btrfs, etc.)**
  - Efficient scanning tools needs to be developed for new types of OSD
- **LiPE + ladvise**
  - Ladvise is a tool that can give file access advices or hints to Lustre servers
  - LiPE can automatically generate advices to be sent according to pre-defined policies
- **LiPE + Loris**
  - Loris: Lustre Online Reliability Improvement System
  - Loris backups MDTs to external storage for disaster recovery
  - LiPE can scan the MDT mirrors of Loris instead of MDTs to avoid metadata performance impact
  - Performance of scanning MDT mirrors is almost the same with scanning MDTs if storage is the same type
- **LiPE + L2RC**
  - L2RC: Lustre Level 2 Read Cache, a OSD level read cache for Lustre
  - Use the LiPE to manage the cache readahead of L2RC
- **LiPE + File Heat**
  - File heat: a value that reflects the access frequency of the objects
  - Scanning OSTs would be more useful if LiPE can apply rules based on the file heat values of OST objects
- **LiPE + MpiFileutils**
  - MpiFileutils: a suite of MPI-based tools to manage large datasets
  - Lipe could use MpiFileutils to scan all kinds of Posix file system

Conclusion

- Implemented a new policy engine for Lustre: LiPE
- LiPE scans MDTs/OSTs directly and requires no external storage
- LiPE has quick scanning speed
- LiPE has a lot of use case potentials, including HSM

Thank you!