Lustre HSM Project

Lustre User Advanced Seminars

Aurélien Degrémont
aurelien.degremont@cea.fr

Thomas Leibovici
thomas.leibovici@cea.fr

Outline

- **Presentation**
  - Architecture
- **Usage**
  - Start Lustre/HSM
  - New file states
  - Coordinator and HSM requests
  - Multiple HSM backends
- **Policy Engine – Robinhood**
  - Presentation
  - Policies
  - Reports

- **Internals**
  - Archive request sequence
 - APIs

Presentation

- **Features**

  - Migrate data to an external storage (HSM)

  - Free disk space when needed

  - Bring back data on cache-miss

  - Policy management (migration, purge, soft rm,…)

  - Import from existing backend

  - Disaster recovery (restore Lustre filesystem from backend)

- **New components**

  - Coordinator

  - Archiving tool (backend specific user-space daemon)

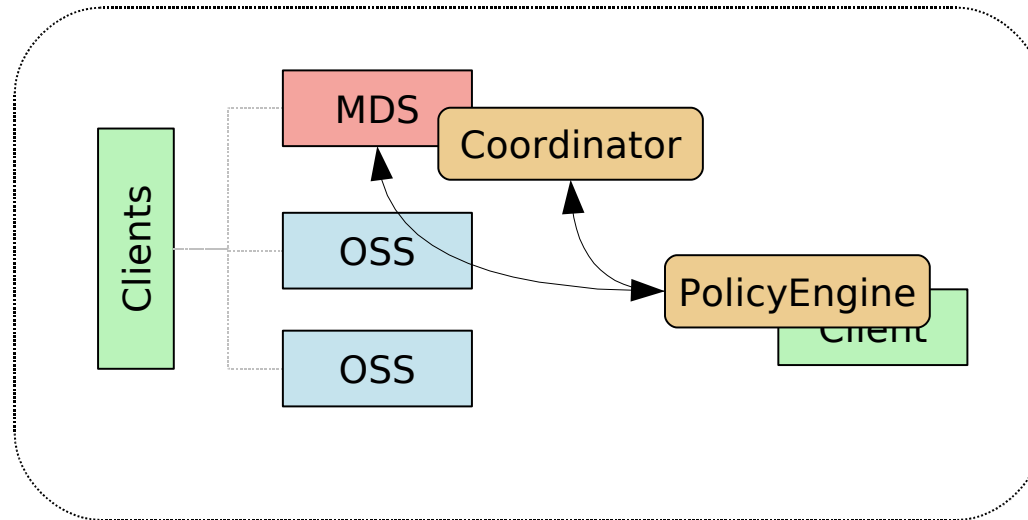 - Policy Engine (user-space daemon)

Architecture (1/2)
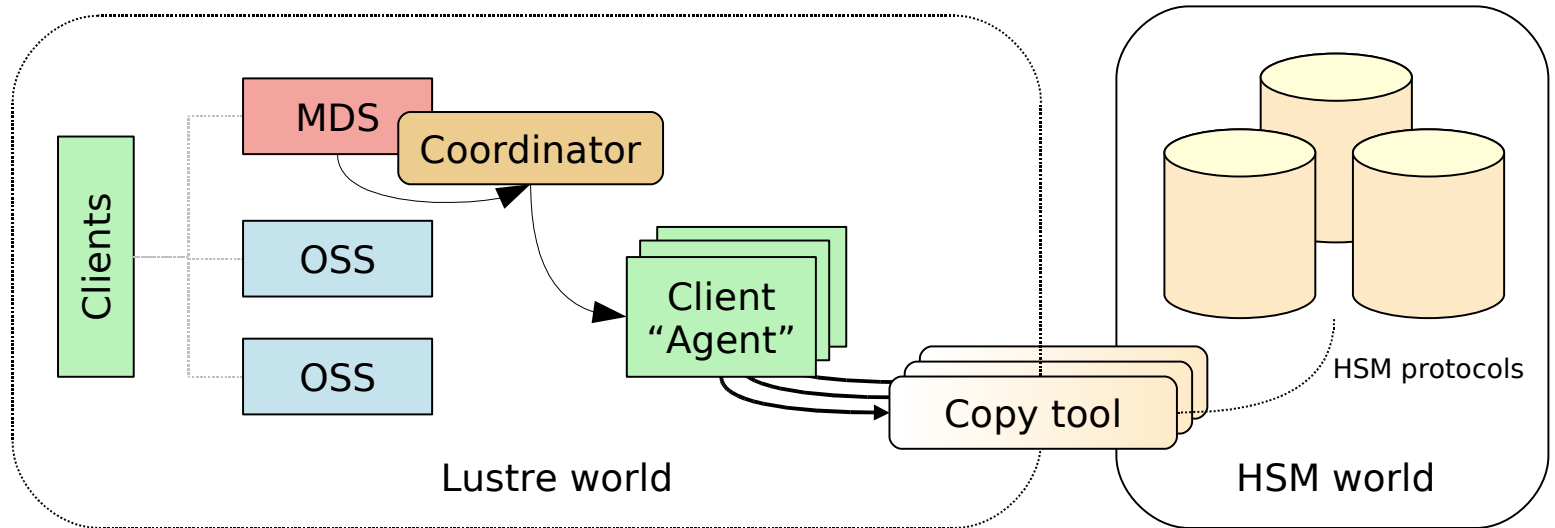
- **New components: Coordinator, Agent and copy tool**

  - The coordinator gathers archiving requests and dispatches them to agents.

  - Agent is a client which runs a copytool which transfers data between Lustre and the HSM.

Architecture (2/2)

- **PolicyEngine manages pre-migration and purge policies.**

  - A user-space tool which communicates with the MDT and the coordinator.

  - Watch the filesystem changes (using Changelogs).

  - Trigger actions like pre-migration, purges and removal in backend.

New HSM file states

- **Files have new possible states and special flags**

  - *Exist*: Some copy exists in a HSM (may be incomplete)

  - *Archived*: A full copy was done for this file.

  - *Dirty*: The Lustre file has been modified since last copy.

  - New file state: *Released*

    - ☛A *released* file still has its inode in MDT.

    - ☛But LOV information and LOV objects on OST are removed.

- **Manually set flags**

  - *Lost*: The file copy has been lost. The file could not be restored.

  - *No release*, *no archive*: Policy flags

- **Users could manually set some of them**

  - *No release*, *no archive*, *dirty*

How-to start

- **Format your devices as usual**
- **Start your MDT with additional mount option**
  - # mount -t lustre … -o hsm_cdt
- **On each node which will act as a transfer node (Agent), starts a copytool**
  - # hsm_copytool_posix --hsm_root /tmp/arc lustre
- **Your filesystem is ready to archive, try:**
  - # lfs hsm_archive /mnt/lustre/my_file

lfs hsm commands

- **Get and change HSM states**

  $ lfs hsm_state /mnt/lustre/my_file

      /mnt/lustre/my_file

      states: (0x00000009) exists archived

  $ lfs hsm_set --norelease /mnt/lustre/my_file

  $ lfs hsm_clear --noarchive /mnt/lustre/my_file

HSM request handler: coordinator

- **New thread on MDT component.**

  - Centralize HSM requests. Ignore duplicate ones.

  - Dispatch and balance them on available copytools.

- **Manage internally a llog of all requests.**

  - Replay request if MDT has crashed.

  - Can be manually canceled

  - Behavior is tunable

    - ☞Can be stopped, resumed, purged

    - ☞Retry/No retry on error

    - ☞Timeouts, number of simultaneous requests, ...

- **Information/Tunings**

  - lctl get_param mdt.lustre-MDT0000.hsm.*

HSM requests

- **Archive**
  - Archiving a file means pre-copying a file from Lustre to an external HSM.
  - A copytool reads file content and copy it in its HSM.
  - File is ready to be released.
- **Release**
  - Remove all file data objects.
  - Synchronous action which does not involve copytool nor coordinator.
- **Restore**
  - All file accesses are blocked until the file is fully restored.
  - Copytool will write file data back.
  - File data accesses are unblocked when restore is finished.

Manage requests

- **Manual HSM requests**

  - lfs hsm_archive /mnt/lustre/my_file

  - lfs hsm_release /mnt/lustre/my_file

  - lfs hsm_restore /mnt/lustre/my_file

    - ☞Asynchronous, non-blocking restore.

    - ☞File open also restores it, but it is synchronous and blocking.

  - lfs hsm_cancel /mnt/lustre/my_file

- **Display current request info**

  - lctl get_param mdt.*.hsm.requests

```
fid=[0x200000400:0x2:0x0] compound/ cookie=0x4ba73f88/ 0x4ba73f87

action=ARCHIVE archive#=0 extent=0x0-0xffffffffffffffff gid=0x0
```
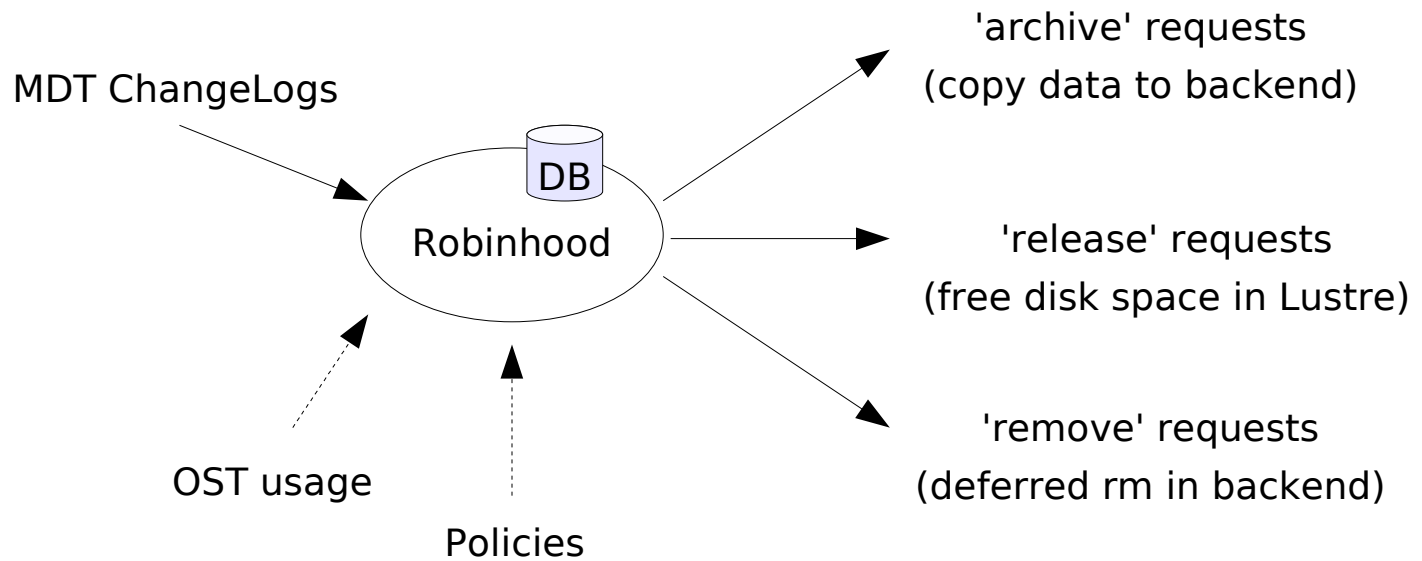
Robinhood

- **Robinhood is a user-space daemon for managing temporary filesystems**
  - Purge oldest files when needed
  - Custom policies
- **With a database backend**
  - Persistent, avoid scanning for each action
  - Currently MySQL and SQLite are supported.
- **Supports and use Lustre features like:**
  - File striping, disk usage per OST, per pool
  - Lustre FID (2.0), Changelogs (2.0)
- **Adapted for Lustre/HSM binding purpose.**
  - To control/schedule file archiving, release, …
- **Flexible policy language**
- **Largest filesystem currently managed: 100 million entries**

Robinhood policies

- **Robinhood manages 3 kinds of policies**
  - Migration policy
  - Purge policy
  - Removal policy
- **Policies**
  - File class definitions, associated to policies
  - Based on file attributes (path, size, owner, age, xattrs, …)
  - Rules can be combined with boolean operators
  - LRU-based migration/purge policies
  - Entries can be white-listed

MDT ChangeLogs

DB

Robinhood

OST usage

Policies

'archive' requests
(copy data to backend)

'release' requests
(free disk space in Lustre)

'remove' requests
(deferred rm in backend)

Robinhood: example of migration policy

- **File classes:**

```
Filesets {
     FileClass small_files {
         definition { tree == "/mnt/lustre/project" and size < 1MB }
         migration_hints = "cos=12" ;
…
     }
}
```

- **Policy definitions:**

```
Migration_Policies {
     ignore { size == 0 or xattr.user.no_copy == 1 }
     ignore { tree == "/mnt/lustre/logs" and name=="*.log" }

     policy migr_small {
             target_fileclass = small_files;
             condition { last_mod > 6h or last_copyout > 1d }
     }
     …
     policy default {
             condition { last_mod > 12h }
             migration_hints = "cos=42";
     }
}
```

Robinhood: example of purge policy

● **Triggers:**

```
Purge_trigger {
    trigger_on = ost_usage;
    high_watermark_pct = 80%;
    low_watermark_pct = 70%;
}
```

● **Policy definitions:**

```
Purge_Policies {
    ignore { size < 1KB }
    ignore { xattr.user.no_release = 1 or owner == "root" }

    policy purge_quickly{
        target_fileclass = classX;
        condition { last_access > 1min }
    }
    …

    policy default {
        condition { last_access > 1h }
    }
}
```

Robinhood administration

- **Control actions: rh-hsm**

  - Robinhood daemon reads events and applies policies.

  - Could be used to manually apply specific purges or migration

    - ☛# rh-hsm --purge-ost=2,10

    - ☛# rh-hsm --sync

- **Display reports/status: rh-hsm-report**

  - Uses Robinhood database to generate reports on the filesystem:

    - ☛Statistics per user, group, largest files, disk space consumers

    - ☛List files per user, group, OST, ...

- **Full administration guide:**

  - See "Documentation" on: http://robinhood.sf.net/

Copytool

- **It is the interface between Lustre and the HSM.**

- **It reads and writes data between them. It is HSM specific.**

- **It stands on a standard Lustre client (called Agent).**

- **2 of them are already availables:**

  - HPSS copytool. (HPSS 7.3+). CEA development which is freely available to all HPSS sites.

  - Posix copytool. Can be used with any system supporting a posix interface. (supports SAM/QFS)

- **More supported HSM to come**

  - DMF

  - Enstore

Import existing HSM

- **It is possible to import an existing HSM namespace into a Lustre filesystem.**

- **All files are imported as *released* and are ready to be restored at first access.**

- **Importing should be implemented for each HSM backend.**

- **Similar mechanism for recovery.**

Multiple HSM backends

- **Start copytools with different archive number:**

  - gateway1# hsm_copytool_posix --hsm_root /tmp/arc1 --archive 1

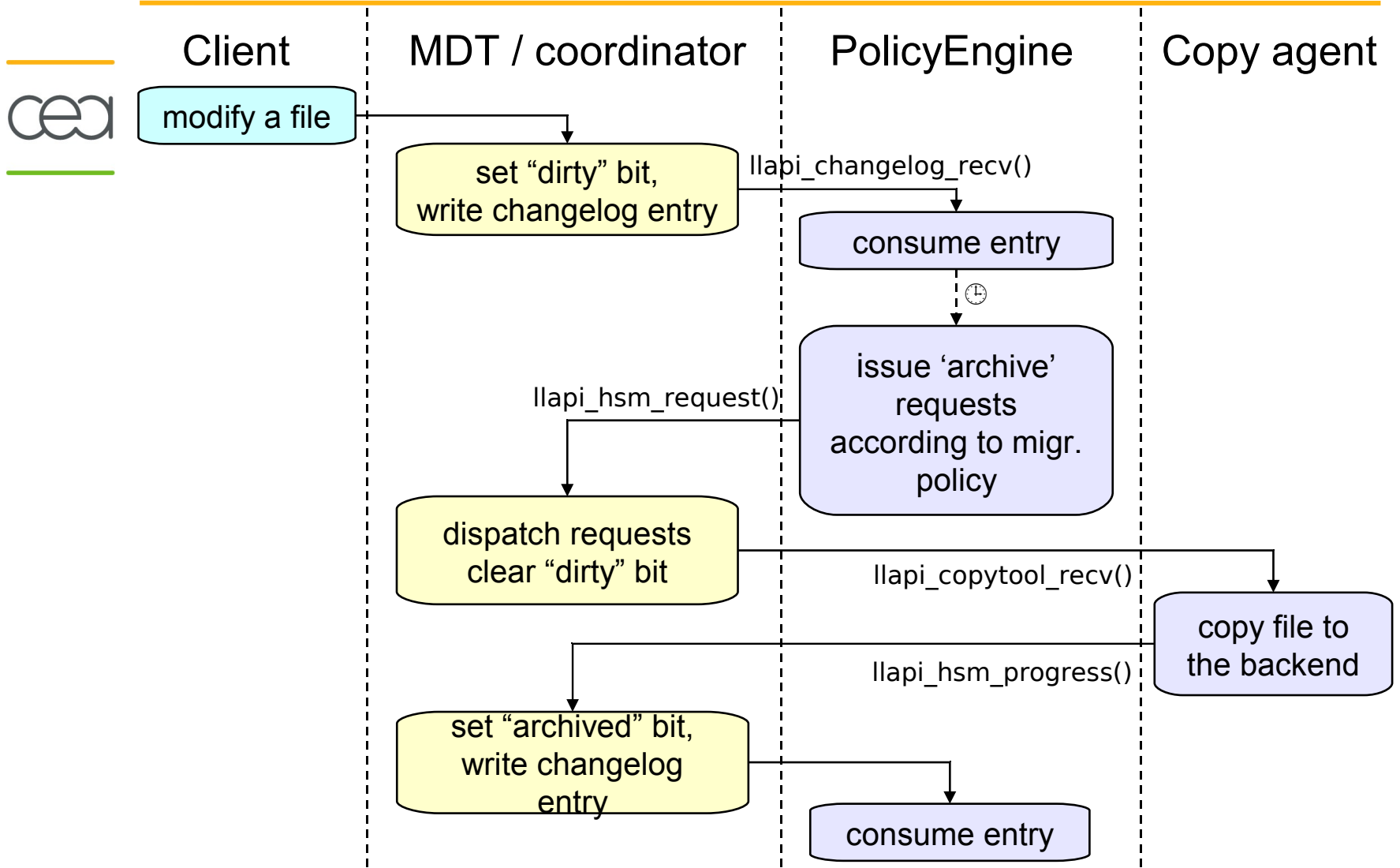 - gateway2# hsm_copytool_posix --hsm_root /tmp/arc2 --archive 2

- Migrate precising which backend you want to use:

  - $ lfs hsm_archive /mnt/lustre/my_file --archive 2
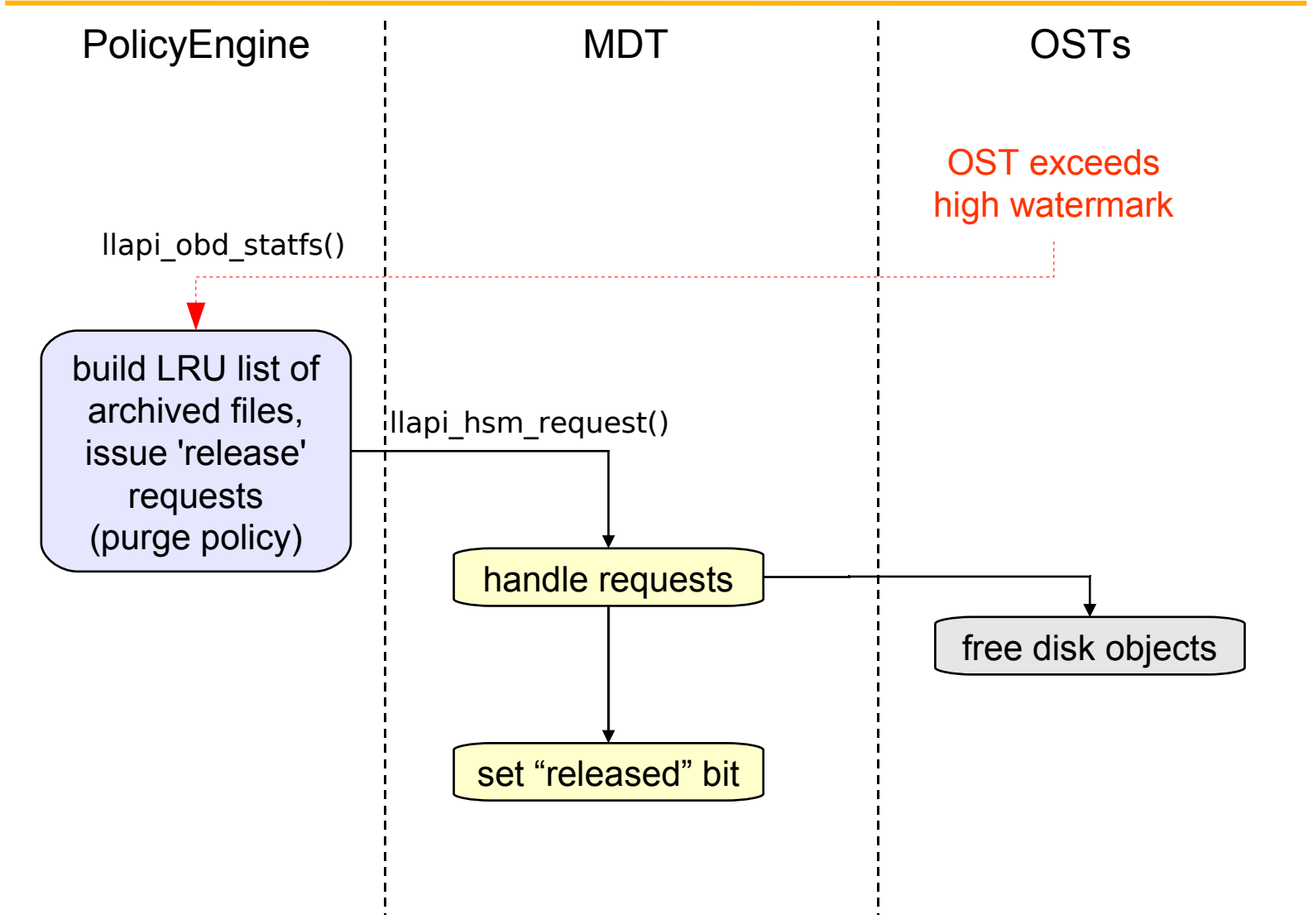
- This can be specified in RobinHood policies:

```
# user files in /mnt/lustre/project_A
FileClass user_files_A
{
      definition
      {
         tree == "/mnt/lustre/project_A"
         and
         owner != "root"
      }
      migration_hints = "cos=14" ;
      # target storage backend
      archive_num = 2;
}
```

Internals: archive modified file

Internals: release OST disk space

PolicyEngine	MDT	OSTs

OST exceeds
high watermark

llapi_obd_statfs()

build LRU list of
archived files,
issue 'release'
requests
(purge policy)

llapi_hsm_request()

handle requests

free disk objects

set "released" bit

Internals: restore released file on read

Client	MDT / coordinator	OSTs	Copy agent

open a "released" file

open()

(blocking)

lock the file, send copy request to agent

→ create objects

llapi_copytool_recv() → **restore file data**

llapi_hsm_progress()

clear "released" bit, unlock the file, write changelog entry

open() is unblocked
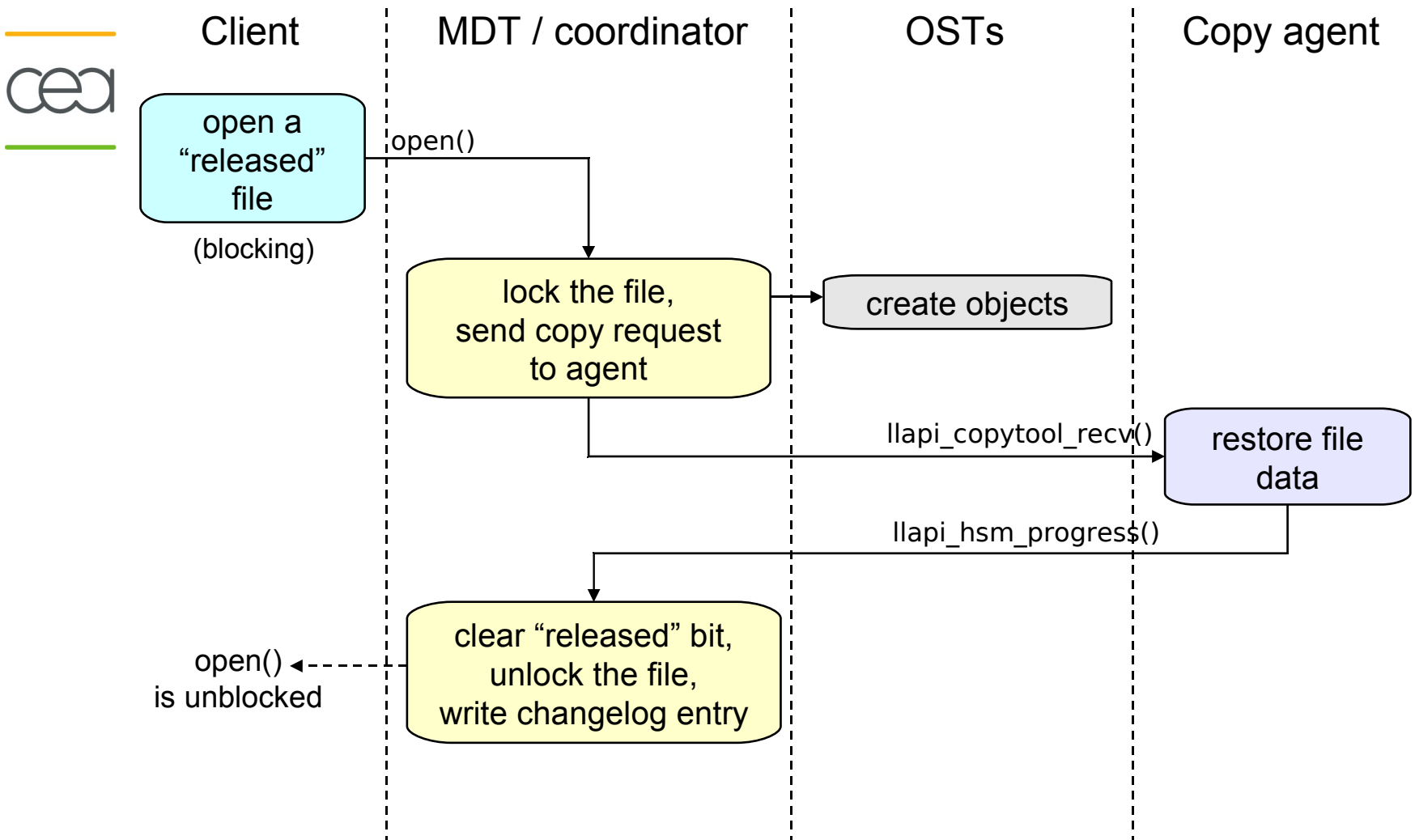
Internal: deferred rm in backend

Client	MDT / coordinator	PolicyEngine	Copy agent

delete an "archived" file

delete file resources, write changelog entry

consume entry

issue 'remove' request according to deferred rm policy

dispatch requests

delete file in backend
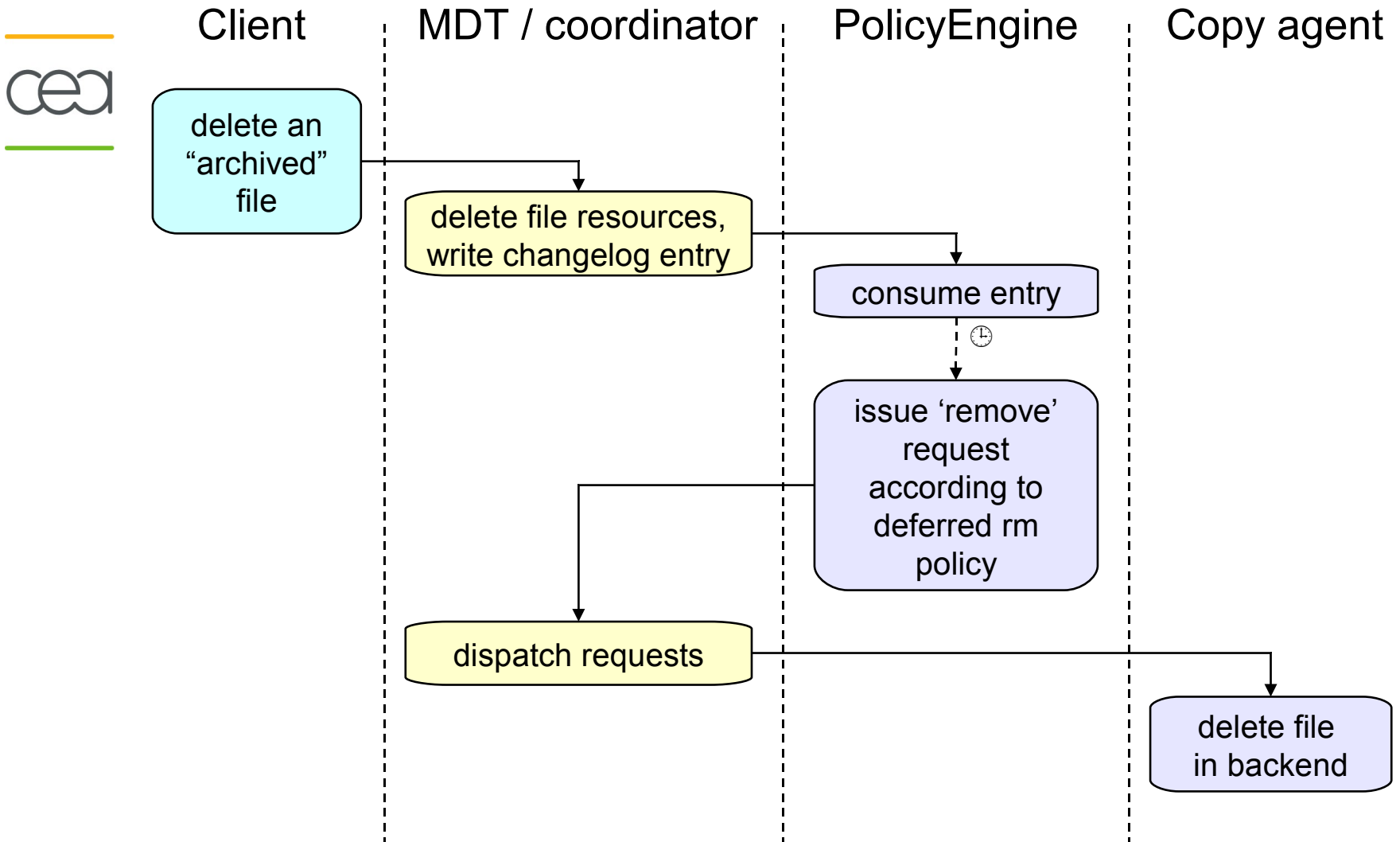
- **Interactions with Lustre/HSM binding:**

  - Classic POSIX access in Lustre (*open*, *read*, *stat*...)

  - liblustreapi / lfs command

- **liblustreapi**

  - Monitor file status (*dirty*, *released*...)

    - ☞llapi_hsm_state_get()

    - ☞llapi_hsm_state_set()

  - Perform hsm requests (*archive*, *release*...)

    - ☞llapi_hsm_request()

- **Changelogs**

  - To get HSM events, you just need to read Lustre changelogs

  - Example:

  ```
  1078 01CREAT 12:29:31.704664916 2010.04.02 0x0 t=[0x200000400:0xff:0x0] p=[0x7a19:0x19634843:0x0] TEST_FILE
  1079 16HSM   12:30:04.53633335 2010.04.02 0x1 t=[0x200000400:0xff:0x0]
  1080 16HSM   12:30:25.492272785 2010.04.02 0x2 t=[0x200000400:0xff:0x0]
  1081 06UNLNK 12:37:01.980434725 2010.04.02 0x3 t=[0x200000400:0xff:0x0] p=[0x7a19:0x19634843:0x0] TEST_FILE
  ```

  - Using command line:

    ☞ lfs changelog

  - Using liblustreapi:

    ☞ llapi_changelog_start()

    ☞ llapi_changelog_recv()

    ☞ llapi_changelog_clear()

    ☞ ...

- **Developing your own copytool**

  - Simply read in one place, write in another place.

  - Use liblustreapi to read requests from coordinator.

  - Inform the coordinator of copy progress/status.

- **liblustreapi**

  - ☛ llapi_copytool_start()

  - ☛ llapi_copytool_recv()

  - ☛ llapi_copytool_fini()

  - ☛ llapi_hsm_progress()

  - ☛ llapi_hsm_import()

Project status

- **Status**
  - Prototype is working

  - Coding is finishing

  - Integration tests, debugging, stress tests

Questions ?