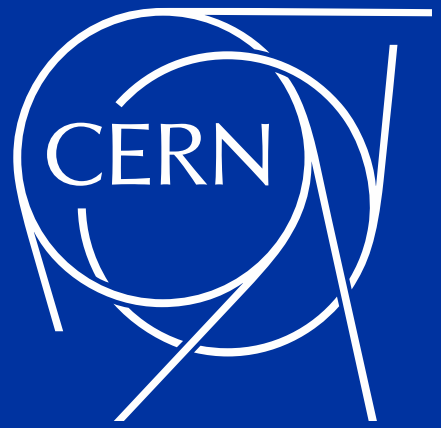


ALICE and the CTA garbage collectors

Author: Steven Murray

Co-authors: Vladimir Bahyl, Eric Cano, Michael Davis, David Fernandez Alvarez, Aurelien Gounon, Oliver Keeble, Julien Leduc, Volodymyr Yurchenko, Cedric Caffy

What is tape-aware garbage collection

The automatic deletion of a disk copy that has a copy safely stored on tape

- **Files to be garbage collected are never deleted directly**
- **Files to be garbage collected are always passed to the “`eos stagerrm`” command**
- **“eos stagerrm” will only delete a disk copy if there is another copy on tape**

```
vi eos/mgm/proc/admin/StagerRmCmd.cc
…
 34 eos::console::ReplyProto
 35 eos::mgm::StagerRmCmd::ProcessRequest() noexcept
 36 {
…
122     // we don't remove anything if it's not on tape
123     if ((buf.st_mode & EOS_TAPE_MODE_T) == 0) {
124       errStream << "error: no tape replicas for file '" << path << "'" << std::endl;
125       ret_c = EINVAL;
126       continue;
127     }
```

Why do we need tape-aware garbage collection

- In the recommended layout of a CTA deployment we don't 😃
  - Ideally we only want two small SSD buffers in front of our tape drives, an archive buffer and a retrieve buffer
  - Two buffers prevents archival jobs from blocking retrieval jobs and vice versa
  - SSDs avoid the performance penalties associated with HDD thrashing
  - When archiving to tape:
    - Delete disk copy from the archive buffer as soon as it is copied to tape
  - When retrieving from tape:
    - Delete disk copy from the retrieve as soon as it is copied to the client

Being pedantic

disk buffer ≠ disk cache

Buffer - disk copy lifetime is extremely short
Cache - disk copy lifetime is as long as space permits

- However 🤔
  - Some users, such as the ALICE experiment, require the more traditional HSM approach
    - A large HDD disk cache - 5 petabytes
    - Disk copies staged into the large HDD disk cache that have not been recently used should be auto "magically" garbage collected to make room for newly retrieved files
    - Disk and tape files live in the same namespace
  - CTA is not perfect
    - We need to protect against file move operations possibly leaving files behind

The 3 types of CTA tape-aware garbage collector (TGC)

EOS MGM - "delete when archived" TGC

It's so simple that it's hard to call it a garbage collector

- **Part of the EOS MGM source code: https://gitlab.cern.ch/dss/eos/-/blob/master/mgm/WFE.cc#L2240**

EOS MGM - Least Recent Used (LRU) TGC:

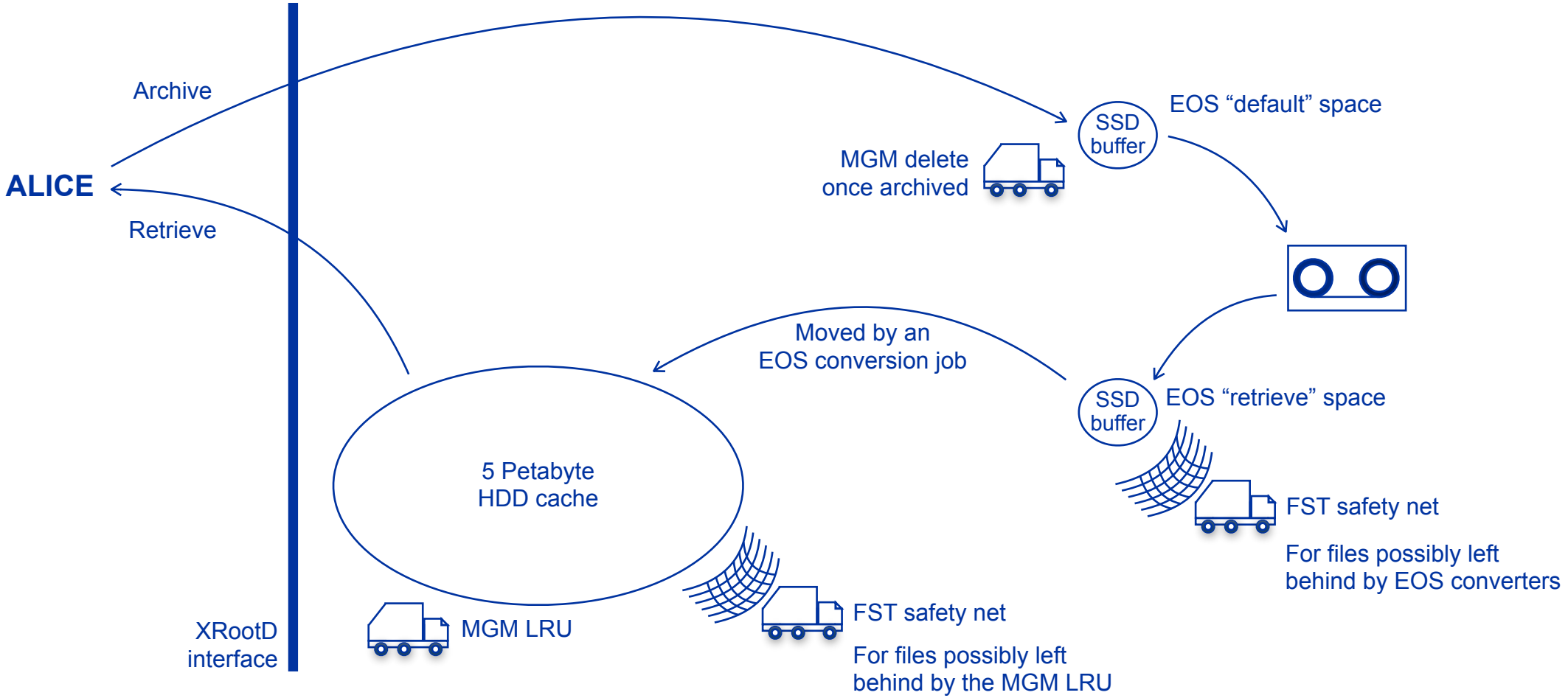
- **Part of the EOS MGM source code: https://gitlab.cern.ch/dss/eos/-/tree/master/mgm/tgc**
- **Fast with but not perfect - we may leave a file behind!**

EOS FST - safety net TGC:

- **Standalone program written in Python**
- **Slow, simple with no persistent state - no file is left unchecked!**
- **A safety net for:**
  - **The EOS MGM LRU tape-aware garbage collector**
  - **Files possibly left behind by move operations - EOS converters and/or FTS transfers**

FTS is not used by ALICE and therefore won't be discussed further

Tape-aware garbage collection of ALICE files



The MGM LRU TGC - the main workhorse for ALICE

- **Manages the 5 Petabyte disk cache of ALICE**
  - **Known by the EOS MGM as the “spinners” EOS space**
- **The MGM LRU TGC has to be told which space to work on:**

```
vi /etc/xrd.cf.mgm
…
mgmofs.tapeenabled true
mgmofs.tgc.enablespace spinners
```

Prevents disk-only EOS instances from getting involved with tape

- **It also needs to know when it should take action**

```
eos space config spinner space.tgc.availbytes=20T
eos space config spinner space.tgc.qryperiodsecs=60
eos space config spinner space.tgc.totalbytes=100T
```

When the MGM LRU TGC should take action

`space.tgc.availbytes`

The threshold when the TGC considers there to be enough free/available space. The MGM TGC will not attempt to garbage collect any files if the actual amount of free/ available space is above this number.

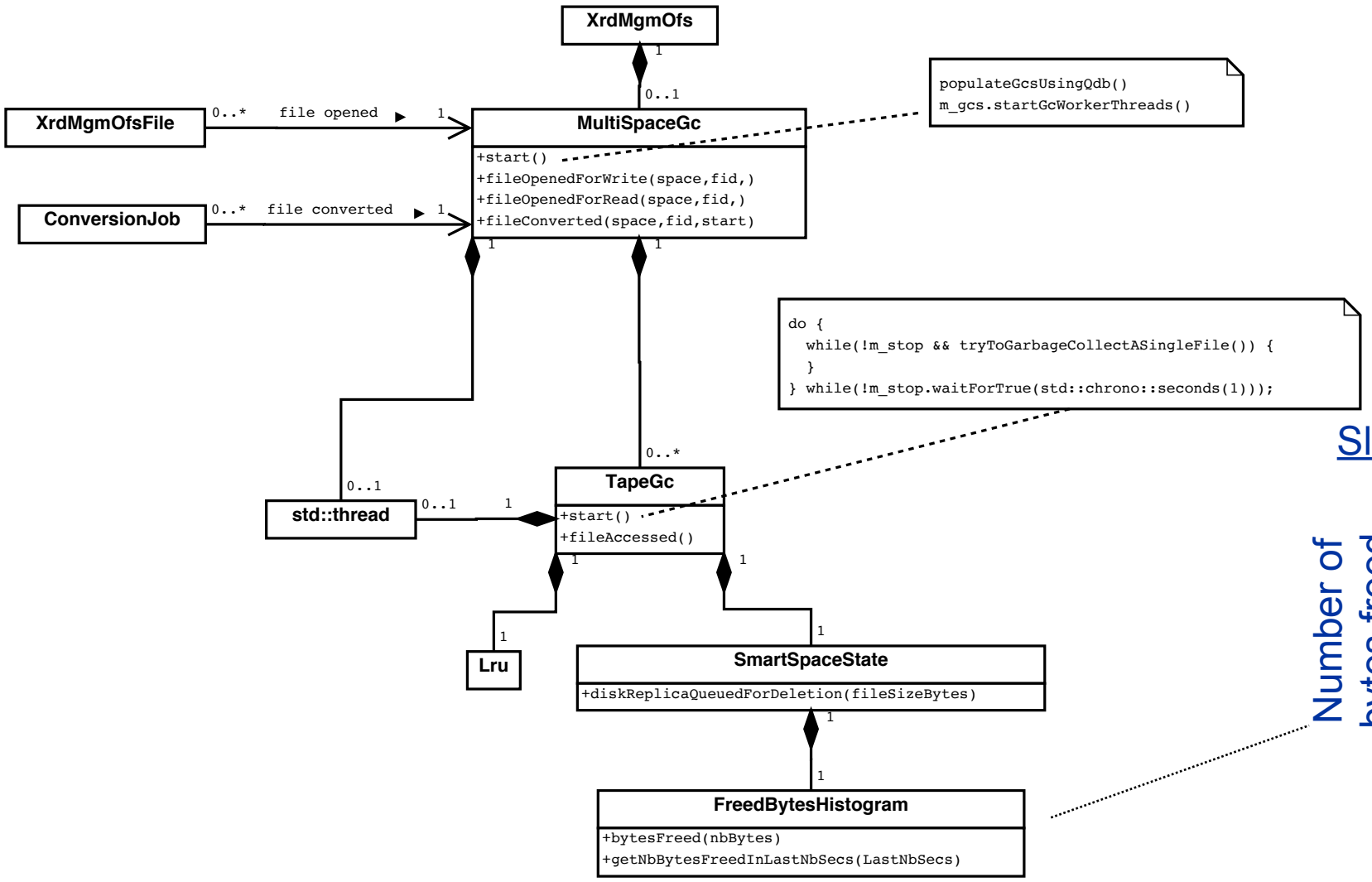
`space.tgc.qryperiodsecs`

The period at which the TGC should query for statistics about the EOS space being managed.

`space.tgc.totalbytes`

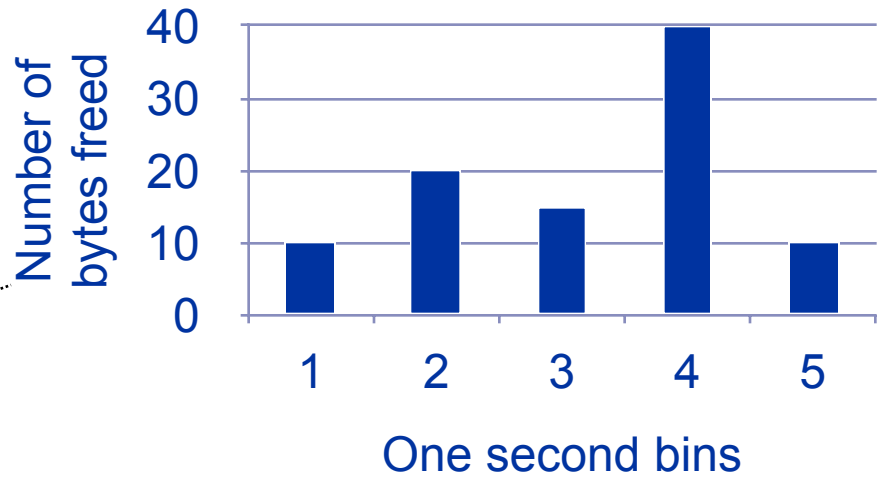
The amount of storage bytes that must be available before the TGC can even begin to take action. This parameter solves a “startup” problem. Once an MGM is started, the TGC must not immediately start considering files for garbage collection because the FSTs will not have had time to register their free/available space.

MGM LRU TGC implementation details



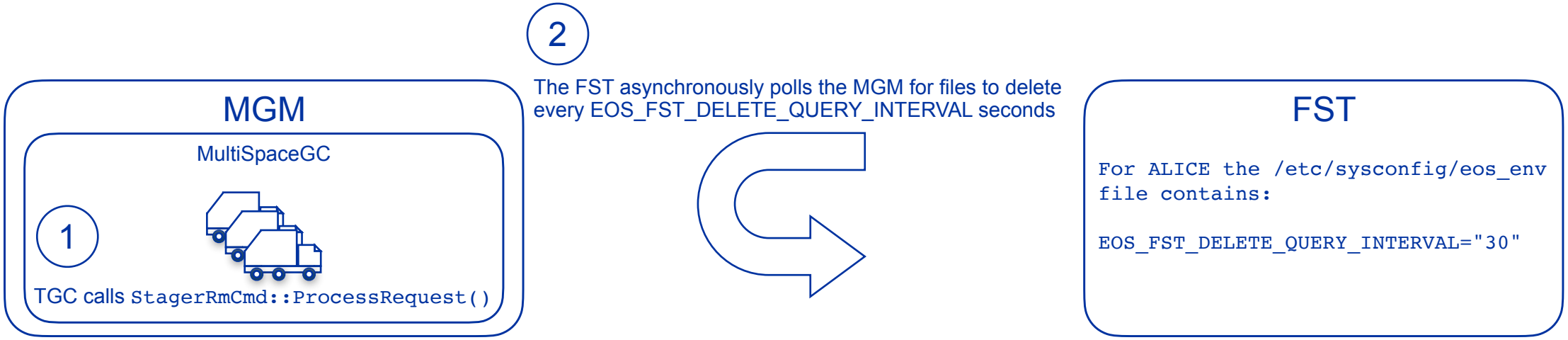
- There is one `TapeTGC` object per EOS space being garbage collected
- Each `TapeTGC` has its own:
  - Thread for garbage collecting
  - Persistent LRU data structure
  - View of free space
- The `MultiSpaceGC` object contains the `TapeTGC` objects
- The `MultiSpaceGC` object has its own thread to concurrently recover the persistent state of the `TapeTGC` objects from QuarkDB when the MGM is started
- EOS objects representing files (`XrdMgmOfsFile`) and conversion jobs (`ConverionJob`) notify the `MultiSpaceGc` object of files being opened and converted
- The MultiSpaceGc dispatches these notifications to the appropriate `TapeTGC` objects
- When a `TapeTGC` object garbage collects a file it updates its view of free space - its “sliding window histogram of freed bytes”

Sliding window histogram of freed bytes



Why the sliding window histogram of freed bytes is required

- **The TapeTGC objects do not know the instantaneous amount of free space in the EOS system**
- **The FST asynchronously queries the MGM for physical disk copies to delete - Every 30 seconds for ALICE**



- **The FST asynchronously publishes its free space every `publish.interval` - Every 10 seconds for ALICE**
- **The value of `space.tgc.qryperiodsecs` is 60 seconds for ALICE because it should be twice the value of `EOS_FST_DELETE_QUERY_INTERVAL` and 5 seconds greater than the value of `publish.interval`**

How the EOS space of a tape file is determined - part 1

```
vi eos/mgm/XrdMgmOfsFile.cc
...
 344 int
 345 XrdMgmOfsFile::open(eos::common::VirtualIdentity* invid,
...
1478   // select space and layout according to policies
1479   Policy::GetLayoutAndSpace(path, attrmap, vid, new_lid, space, *openOpaque,
1480                             forcedFsId, forced_group);
...
2881   // Notify tape garbage collector if tape support is enabled
2882   if (gOFS->mTapeEnabled) {
2883     try {
2884       eos::common::RWMutexReadLock tgc_ns_rd_lock(gOFS->eosViewRWMutex, __FUNCTION__,
2885           __LINE__, __FILE__);
2886       const auto tgcFmd = gOFS->eosFileService->getFileMD(fileId);
2887       const bool isATapeFile = tgcFmd->hasAttribute("sys.archive.file_id");
2888       tgc_ns_rd_lock.Release();
2889
2890       if (isATapeFile) {
2891         if (isRW) {
2892           const std::string tgcSpace = nullptr != space.c_str() ? space.c_str() : "";
2893           gOFS->mTapeGc->fileOpenedForWrite(tgcSpace, fileId);
2894         } else {
2895           const auto fsId = getFirstDiskLocation(selectedfs);
2896           const std::string tgcSpace = FsView::gFsView.mIdView.lookupSpaceByID(fsId);
2897           gOFS->mTapeGc->fileOpenedForRead(tgcSpace, fileId);
2898         }
2899       }
2900     } catch (...) {
2901       // Ignore any garbage collection exceptions
2902     }
2903   }
```

Use the normal EOS rules when writing the file to EOSCTA for the first time:

- Use the `eos.space` query parameter if available and allowed
- Otherwise use the EOS space policy configuration

Use the EOS space where the first disk copy is physically stored when reading the file from disk

How the EOS space of a tape file is determined - part 2

```
vi eos/mgm/convert/ConversionJob.cc
...
138 //------------------------------------------------------------------------------
139 // Execute a third-party copy
140 //------------------------------------------------------------------------------
141 void ConversionJob::DoIt() noexcept
142 {
...
348   // Notify the tape garbage collector if tape support is enabled
349   if (gOFS->mTapeEnabled) {
350     try {
351       eos::common::RWMutexReadLock fs_rd_lock(FsView::gFsView.ViewMutex, __FUNCTION__,
352                                               __LINE__, __FILE__);
353       eos::common::RWMutexReadLock ns_rd_lock(gOFS->eosViewRWMutex, __FUNCTION__,
354                                               __LINE__, __FILE__);
355       const auto fmd = gOFS->eosView->getFile(mSourcePath);
356
357       if (nullptr != fmd && fmd->hasAttribute("sys.archive.file_id")) {
358         const auto fsId = getDiskFsIdOfFile(*fmd);
359         const std::string tgcSpace = FsView::gFsView.mIdView.lookupSpaceByID(fsId);
360         gOFS->mTapeGc->fileConverted(tgcSpace, fmd->getId());
361       }
362     } catch (...) {
363       // Ignore any garbage collection exceptions
364     }
365   }
```

Simply use the EOS space where the conversion job has written the disk copy

The safety net FST TGC

```
Forever loop
  Run ‘eos -r 0 0 fs ls -m’ to get file system to space map
  For each file system loop
    For each sub directory loop
      For each file loop
        Skip if file system not in list of EOS spaces to be garbage collected
        If over absolute maximum age or space needs
        to be freed and over GC age then
          Try to garbage collect file
        End if
      End loop
    End loop
  End loop
End loop
```

Not efficient but allows for other not yet specified checks and actions to be taken per file

Two garbage collection strategies:
- Absolute maximum age
  - Protects file move operations
  - For archive and retrieve buffers
- Space needs to be freed and over GC age
  - Protects the MGM LRU TGC
  - For large HDD disk cache

Conclusions

- **The tape-aware garbage collection of CTA has been deliberately kept simple**
- **So far the implementation is good enough**
- **The combination of the fast but forgetful MGM LRU TGC with the slow but no file gets left behind FST TGC is working well**

Possible future work

- Integrate with the high availability mechanism of the EOSCTA MGMs
- Persistently store the LRU as an LRU
  - Currently after an MGM restart, FIFO order is reconstructed from the disk copy creation times stored in QuarkDB
  - This FIFO solution is good enough for ALICE when they stage in 5 petabytes of data, work on it for a while and then stage in the next 5 petabytes
  - Future users may need true LRU reconstruction after an MGM restart