On the 4th of September 2023…

On the 4th of September 2023…

Subject **EOS deadlock for HELP**

Dear EOS Experts,

The eos instance of lhaaso experiment at IHEP experienced a major failure.
We observed that when mgm started, a deadlock occurred, causing the locked-up to become larger and larger, ultimately causing the client to be unable to access /eos.

Our eos version is 5.1.27, and the results of `eos ns` are shown as follows:
[root@eos01 ~]# eos ns

ALL File cache max num 40000000
ALL File cache occupancy 19839
ALL In-flight FileMD 0
ALL Container cache max num 5000000
ALL Container cache occupancy 3426
ALL In-flight ContainerMD 0
--
ALL eosViewRWMutex status locked-up (52s)
ALL eosViewRWMutex peak-latency 34ms (last) 34ms (1 min) 104ms (2 min) 13871ms (5 min)
--
ALL QClient overall RTT 0ms (min) 0ms (avg) 214ms (max)
ALL QClient recent peak RTT 1ms (1 min) 214ms (2 min) 214ms (5 min)
--

On the 4th of September 2023…

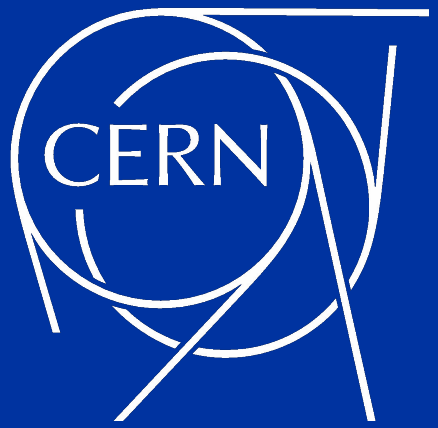
After investigation

- No deadlock!
- Cause: very intensive metadata operation performed by LHAASO on the EOS namespace
  - High contention on the namespace lock…

On the 4th of September 2023…

After investigation

- No deadlock!
- Cause: very intensive OS namespace
  - High contention on t



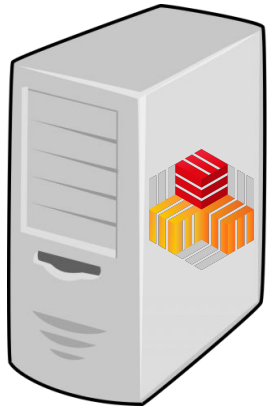
Evolution of the EOS namespace locking

Presented by Cedric Caffy on behalf of the EOS team

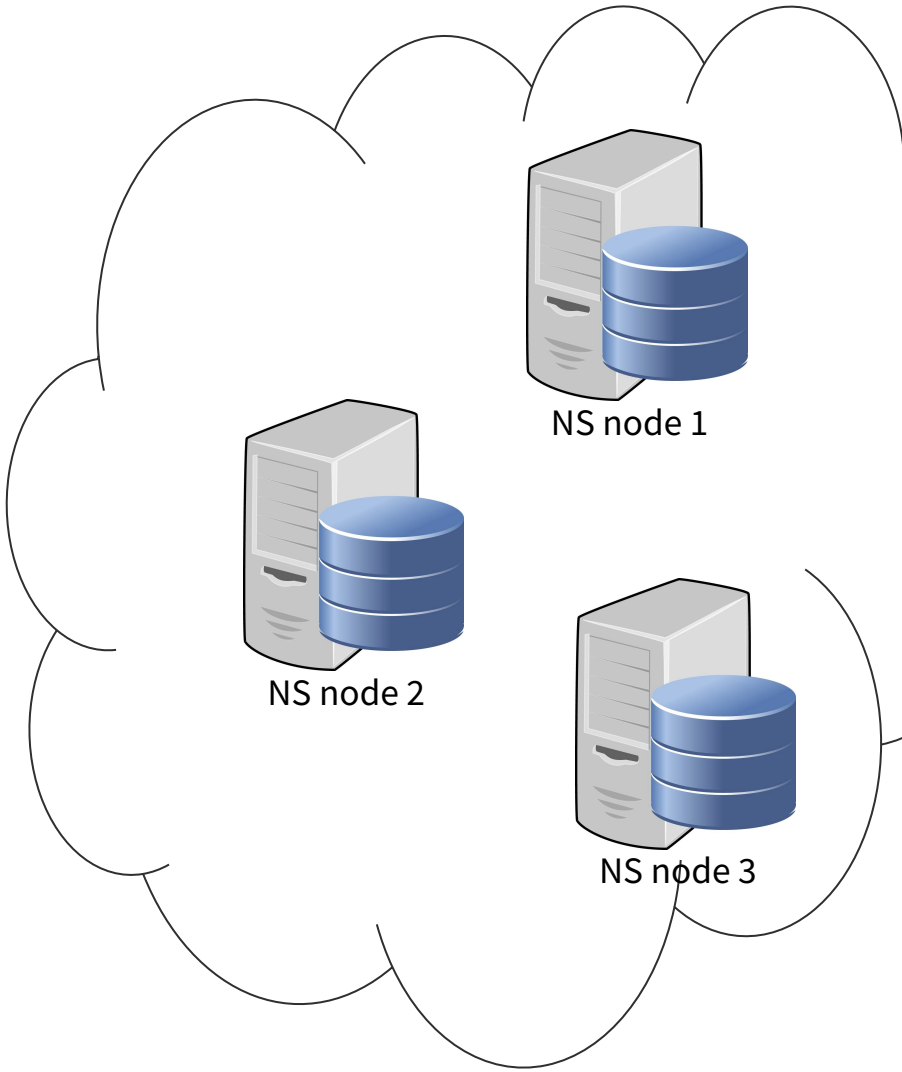
EOS Workshop 2024
15/03/2024

What is the EOS namespace used for?

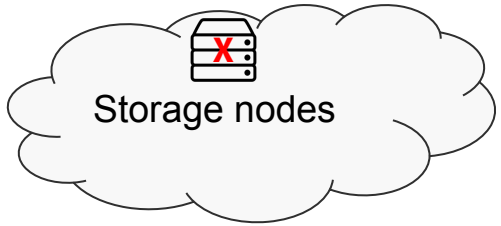
FILE



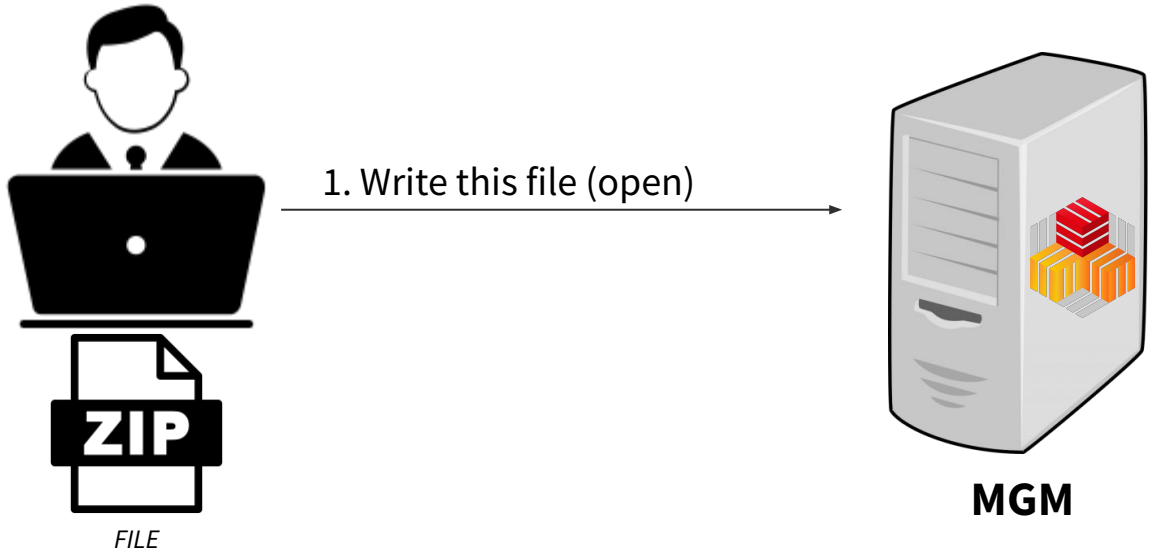
MGM

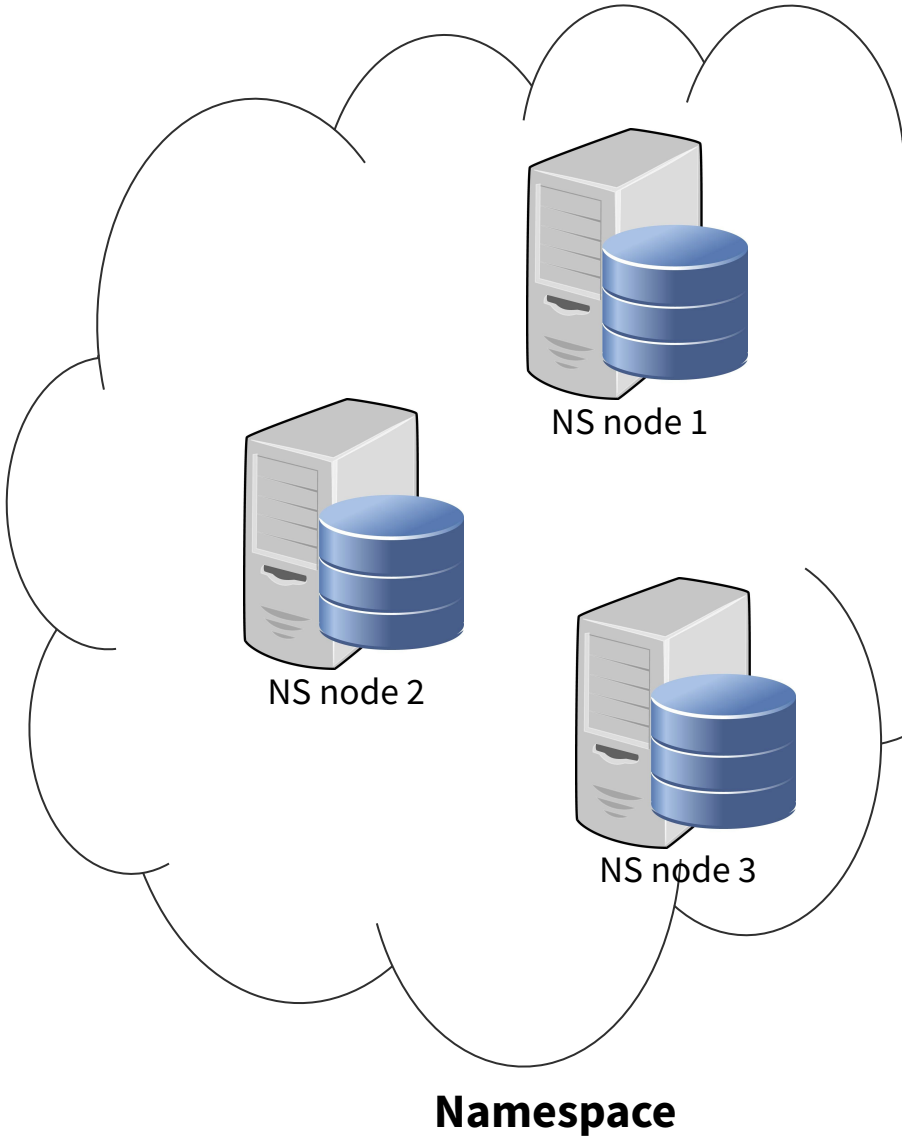


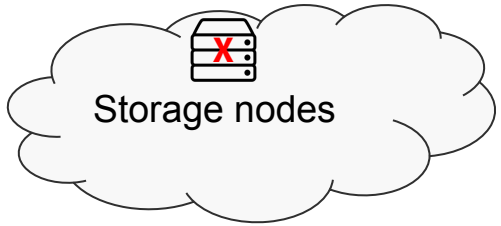
Namespace



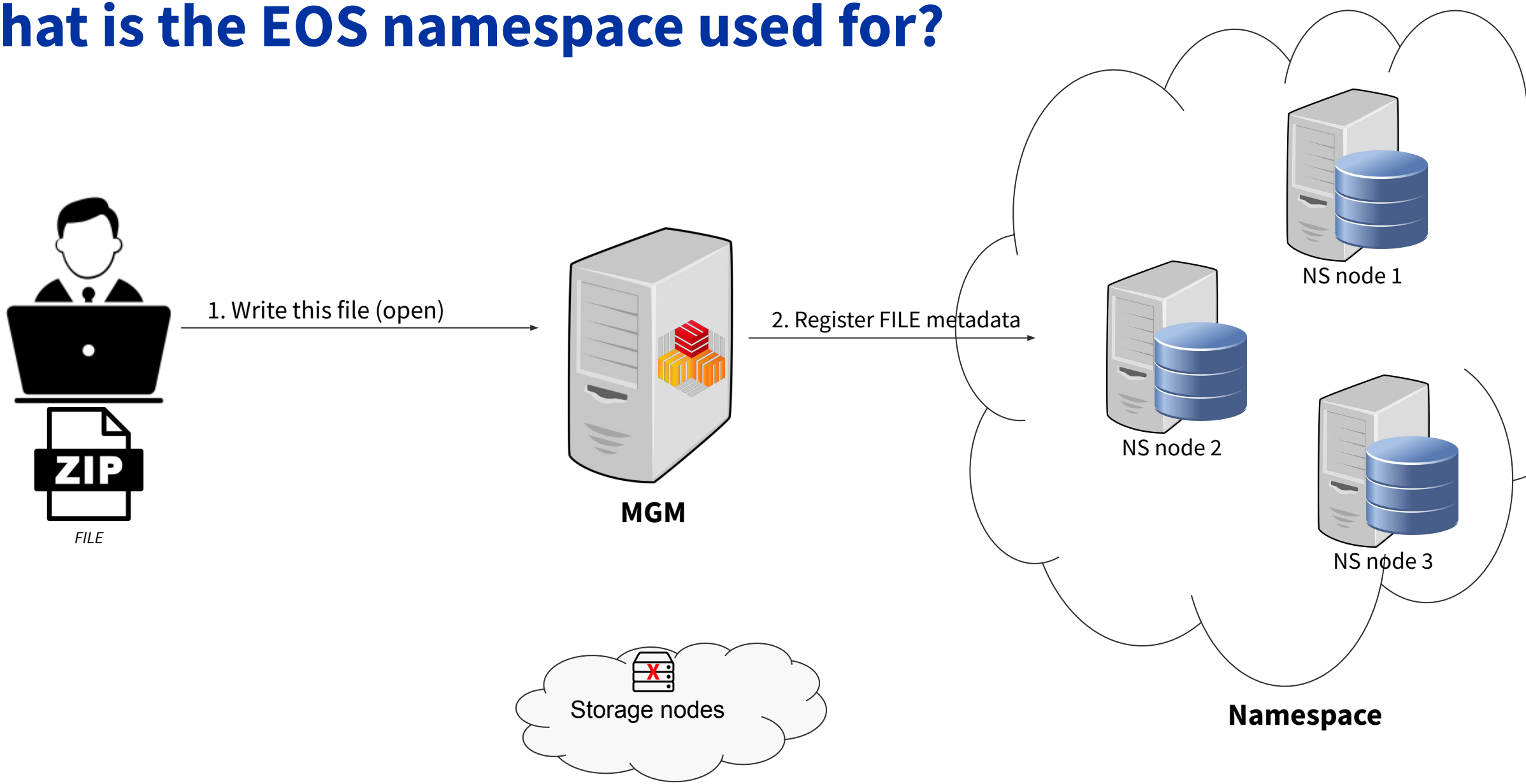
What is the EOS namespace used for?



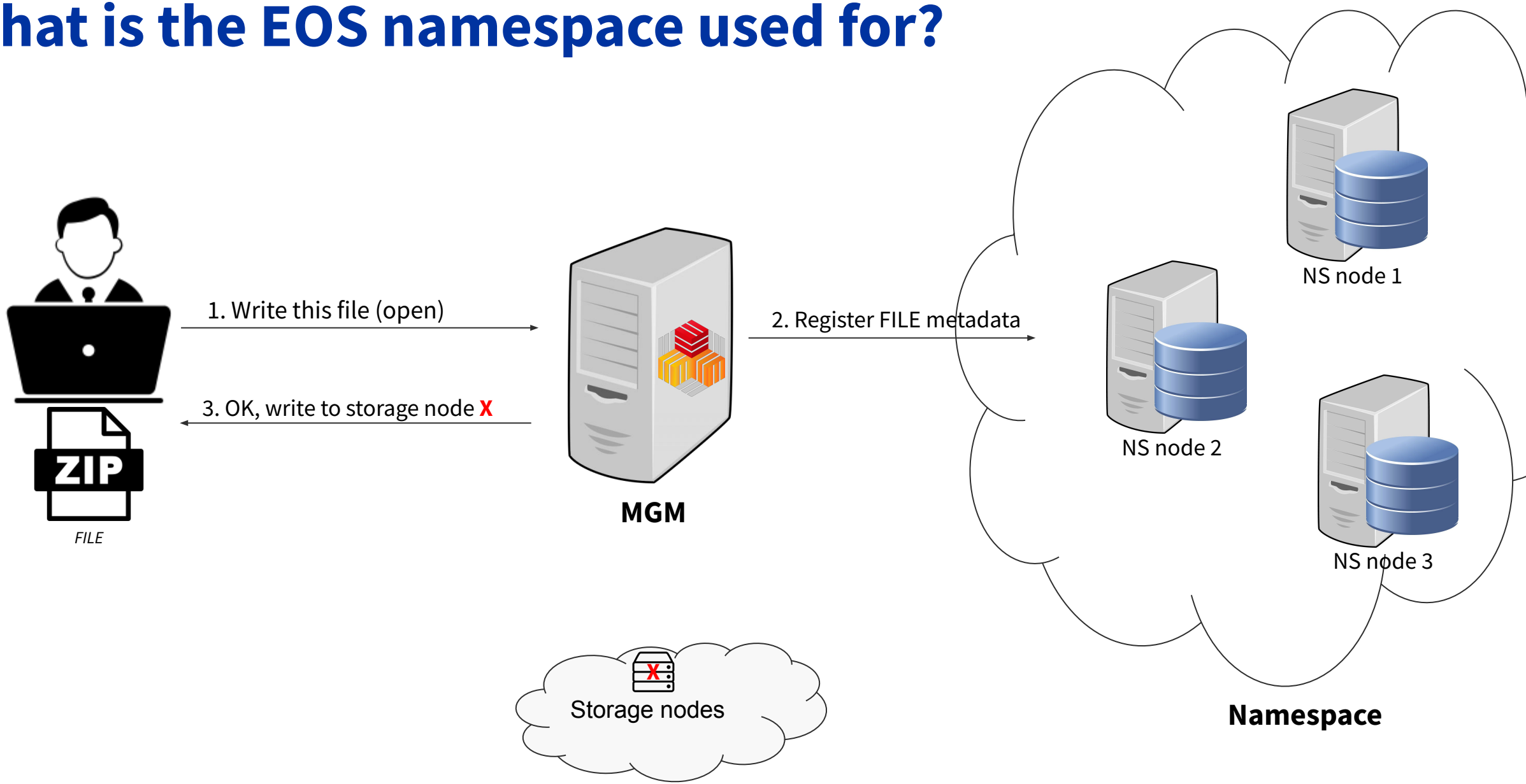




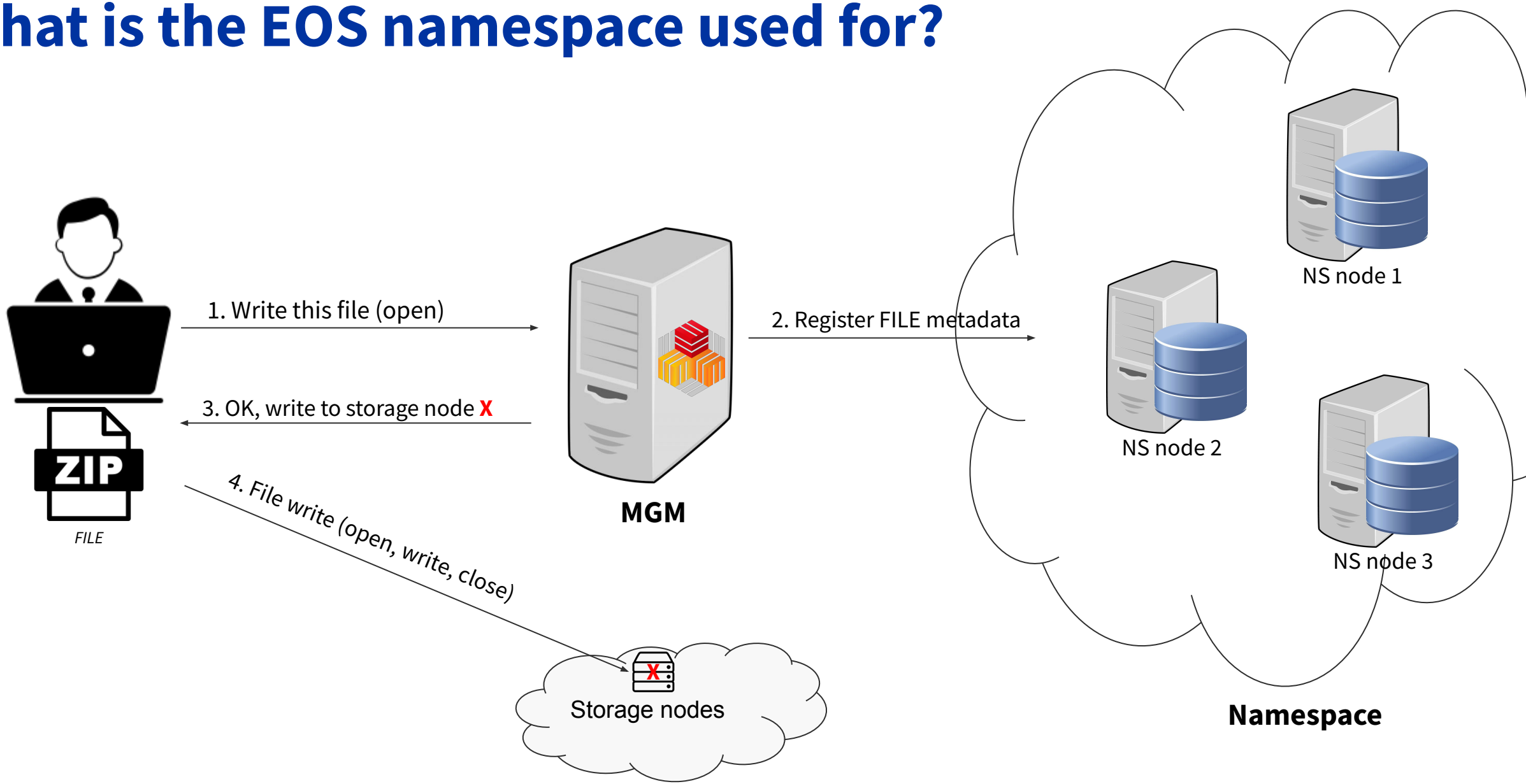
What is the EOS namespace used for?



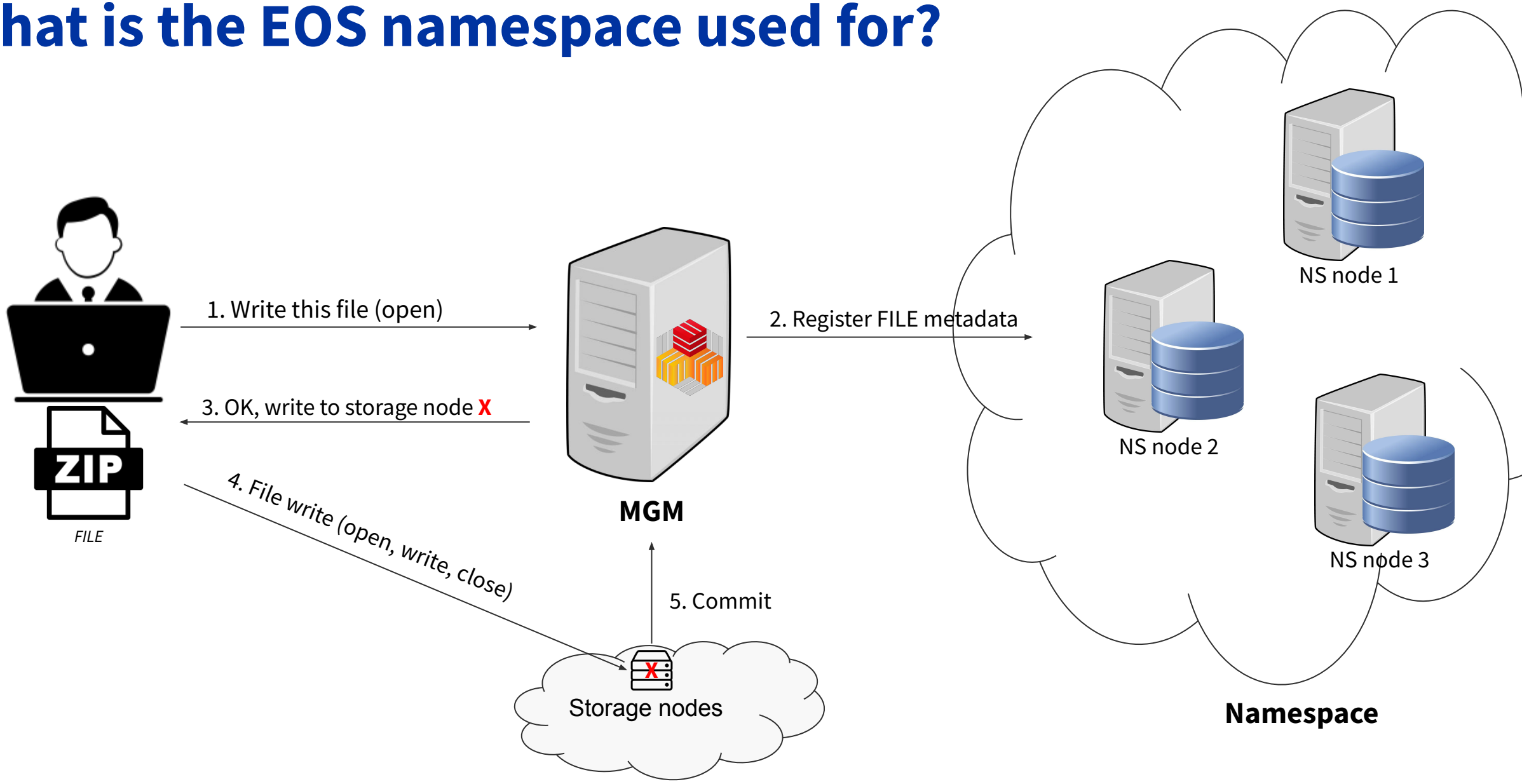
What is the EOS namespace used for?



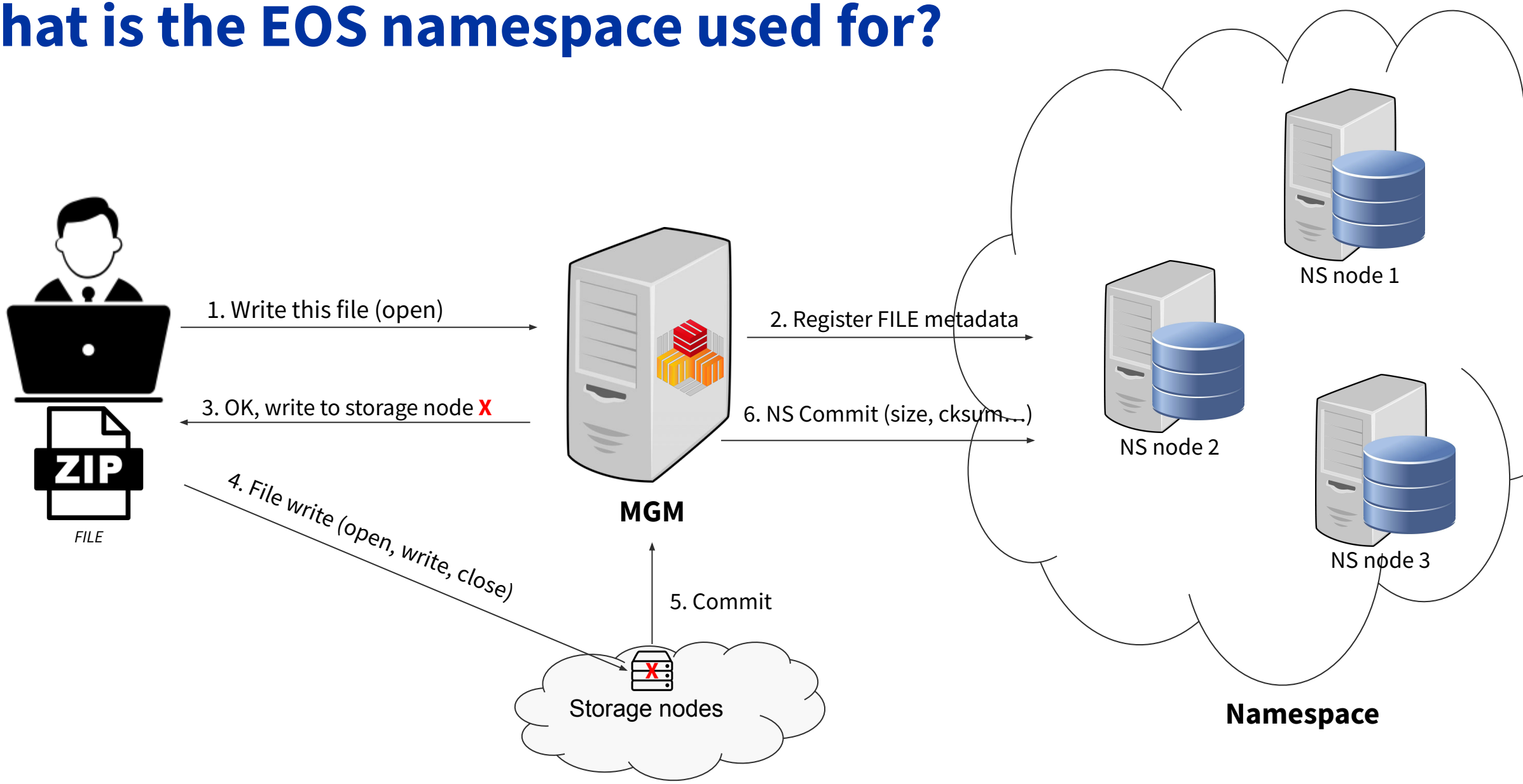
What is the EOS namespace used for?



What is the EOS namespace used for?



What is the EOS namespace used for?



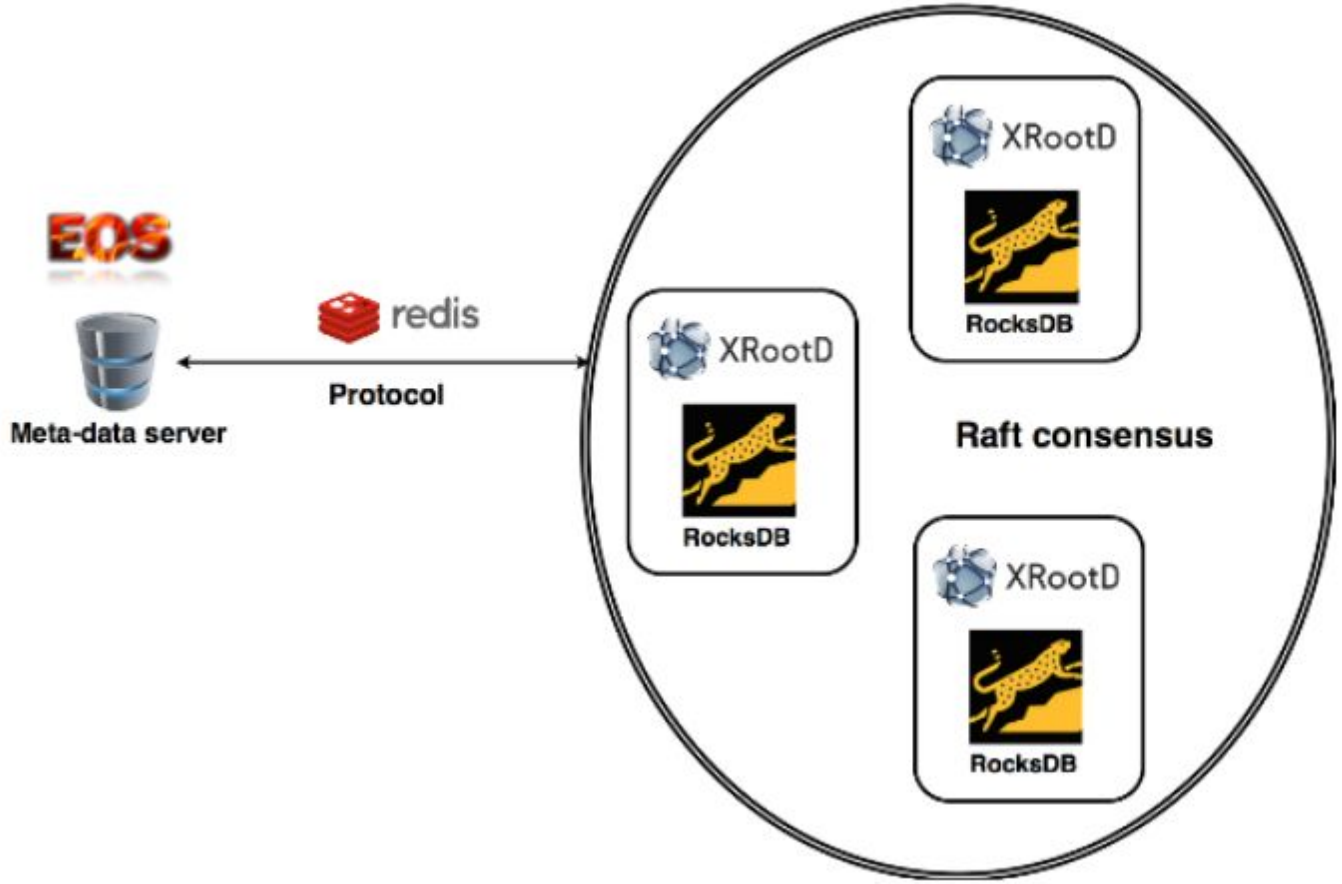
What is the EOS namespace used for?

Store the metadata of all files/directories in EOS

- Path
- Creation time
- Size
- Checksum
- ….

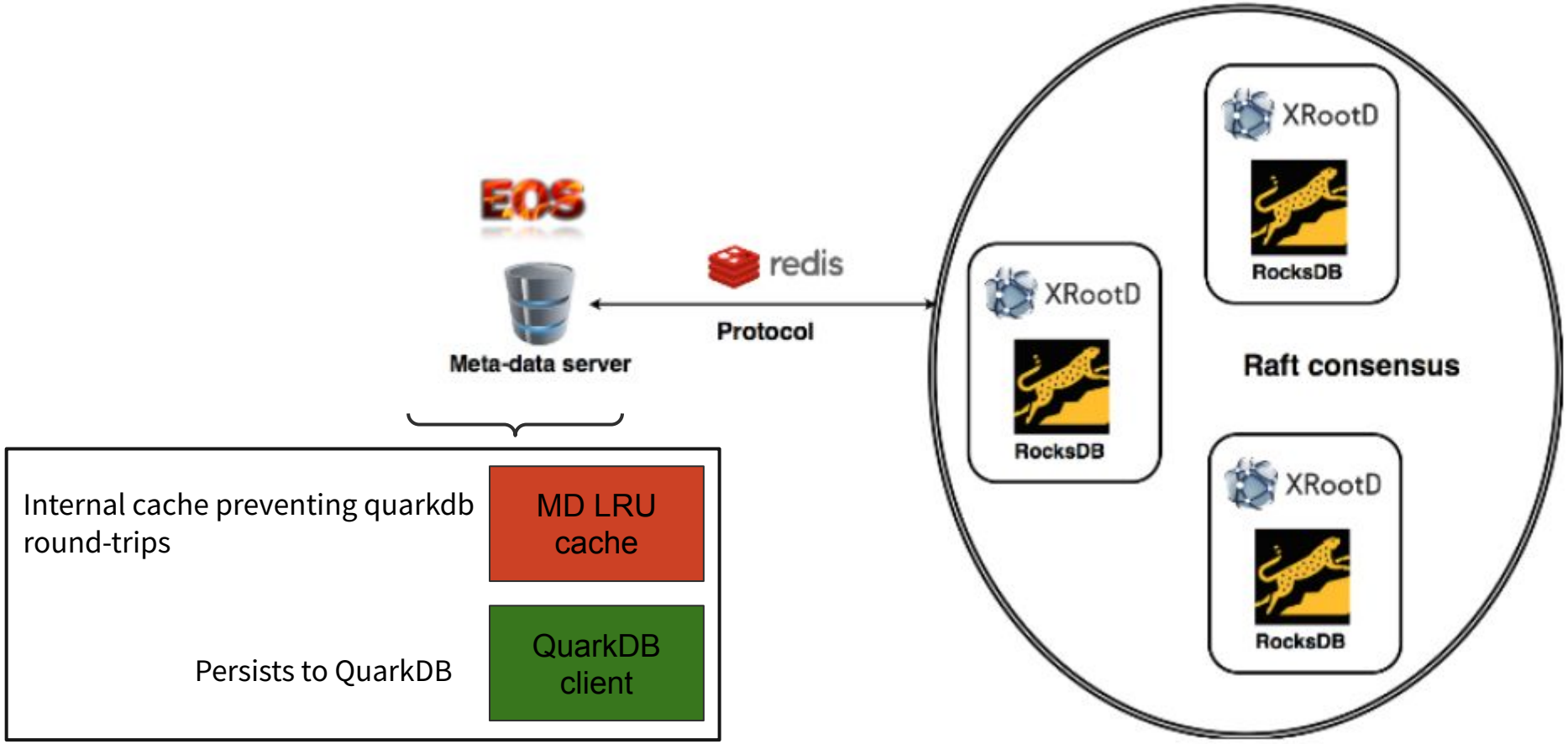
The EOS namespace persistency

QuarkDB - A highly-available key-value store



The EOS namespace persistency

QuarkDB - A highly-available key-value store



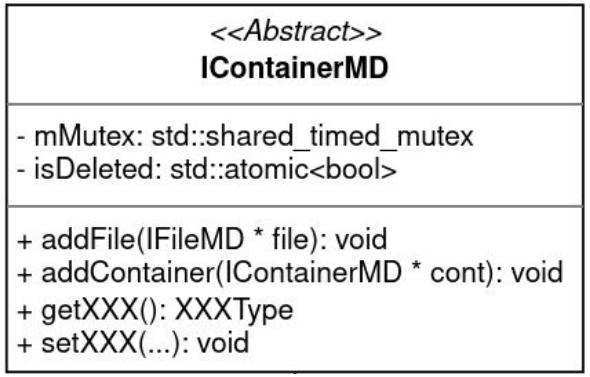
A file path in the EOS namespace

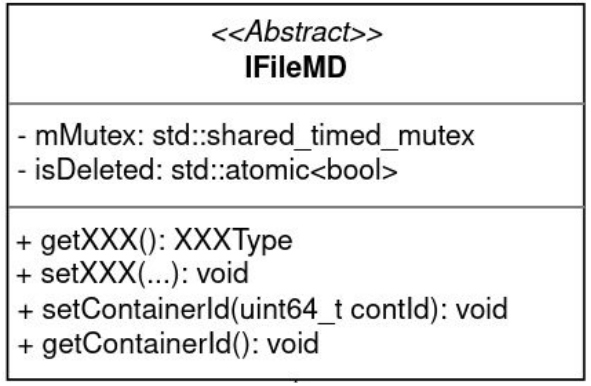
/ eos / workshop2024 / presentations / namespace_locking_presentation.pdf

Containers

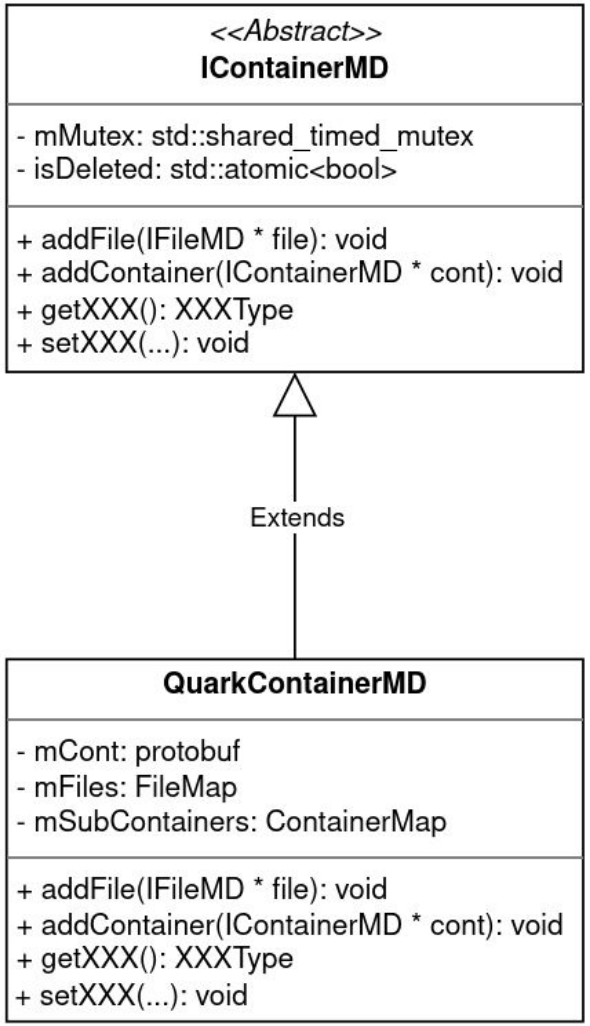
File

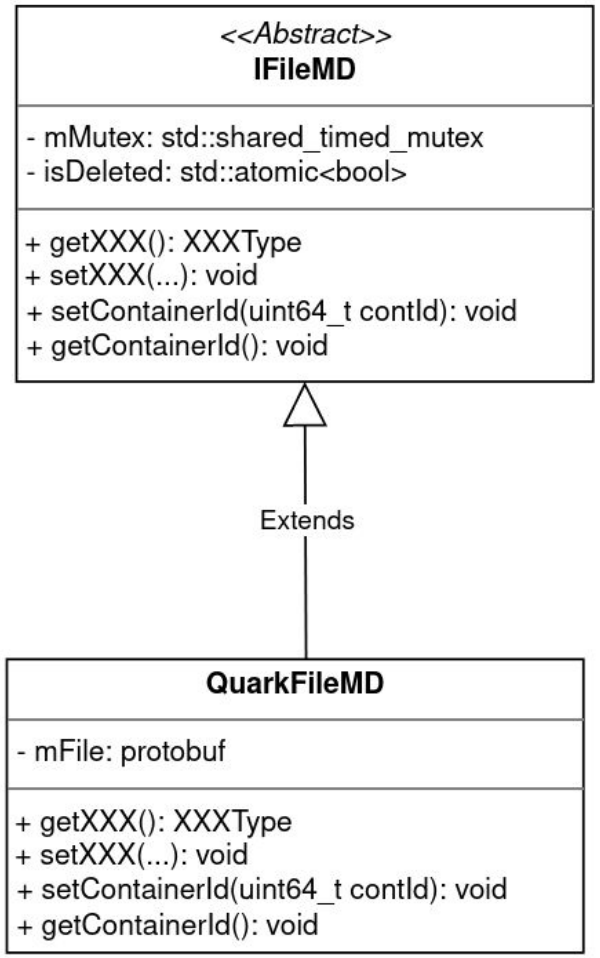
C++ representation of containers and files





C++ representation of containers and files





What is the EOS Namespace global lock?

What is the EOS Namespace global lock?

mkdir, stat, FSctl, exists, chown, chmod…

```
class XrdMgmOfs : public XrdSfsFileSystem, public eos::common::LogId
{
    //[...]
    public:
    eos::common::RWMutex eosViewRWMutex; ///< RW namespace mutex
    //[...]
};
```

It's just a Mutex… Accessible from everywhere in the EOS MGM code

What is the EOS Namespace global lock?

Specificity of the EOS Namespace global lock

- Used to protect concurrent files/container metadata operation

What is the EOS Namespace global lock?

Specificity of the EOS Namespace global lock

- Used to protect concurrent files/container metadata operation
- Implemented using the Abseil mutex (https://abseil.io/docs/cpp/guides/synchronization)
  - Since EOS 5.2.0

What is the EOS Namespace global lock?

Specificity of the EOS Namespace global lock

- Used to protect concurrent files/container metadata operation
- Implemented using the Abseil mutex (https://abseil.io/docs/cpp/guides/synchronization)
  - Since EOS 5.2.0
- Provides read and write locks (like std::shared_mutex)
  - Write-lock - read locks will wait that it is released
  - Read-lock - no contention for reads if no write lock taken

What is the issue with the global namespace lock?

What is the issue with the global namespace lock?

Long-lasting metadata operations

```
//Write lock the namespace
eos::common::RWMutexWriteLock wr_lock(gOFS->eosViewRWMutex);
std::shared_ptr<IContainerMD> cont;

for (auto const& elem : containers) {
  try {
    //Get the container by ID
    cont = mContainerMDSvc->getContainerMD(elem.first);
    //Update the tree size of the container (simple setter)
    cont->updateTreeSize(elem.second);
    //Persist the container to QuarkDB
    mContainerMDSvc->updateStore(cont.get());
  } catch (const MDException& e) {
    continue;
  }
}
```

It is used everywhere we need to perform operations on File/Container MD…

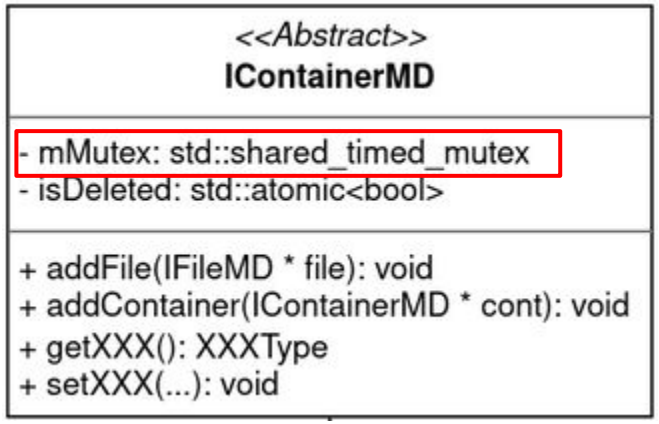
What if we have plenty of directories to update here?

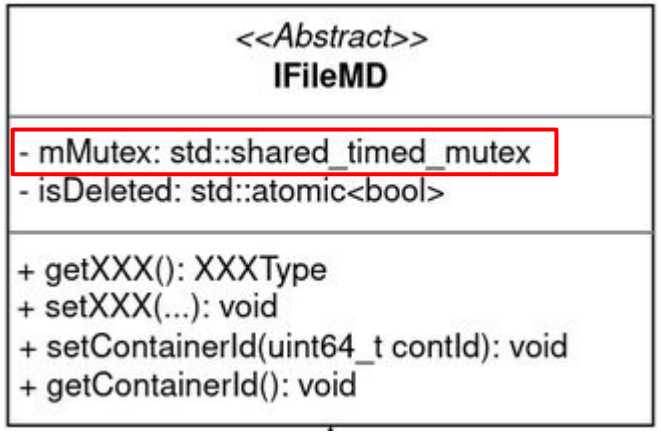
All operations on the MGM will be blocked during these updates!!

Can we improve the situation?

YES!

Using finer-grained locks → the file/container object will be locked instead of the global namespace mutex





Using finer-grained locks - before the evolution

```
inline uint64_t  getTreeSize() const override
{
  std::shared_lock<std::shared_timed_mutex> lock(mMutex);
  return mCont.tree_size();
}
```

```
inline void setTreeSize(uint64_t treesize) override
{
  std::unique_lock<std::shared_timed_mutex> lock(mMutex);
  mCont.set_tree_size(treesize);
}
```

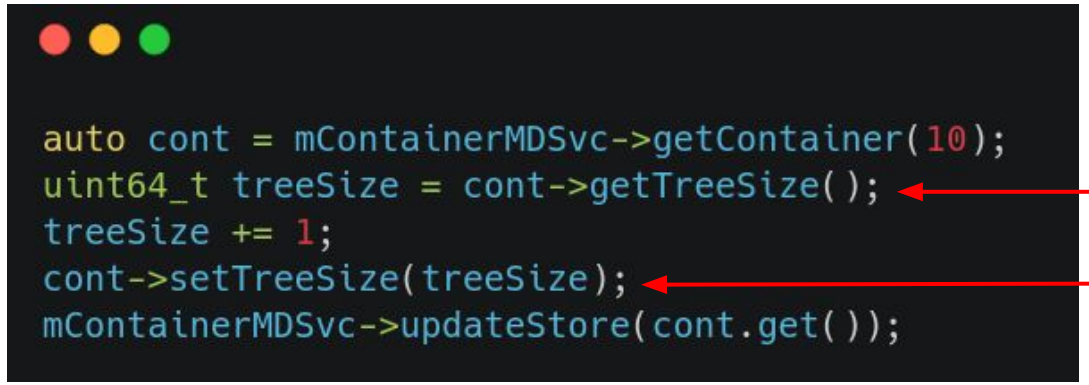
Using finer-grained locks - before the evolution

```
auto cont = mContainerMDSvc->getContainer(10);
uint64_t treeSize = cont->getTreeSize();
treeSize += 1;
cont->setTreeSize(treeSize);
mContainerMDSvc->updateStore(cont.get());
```

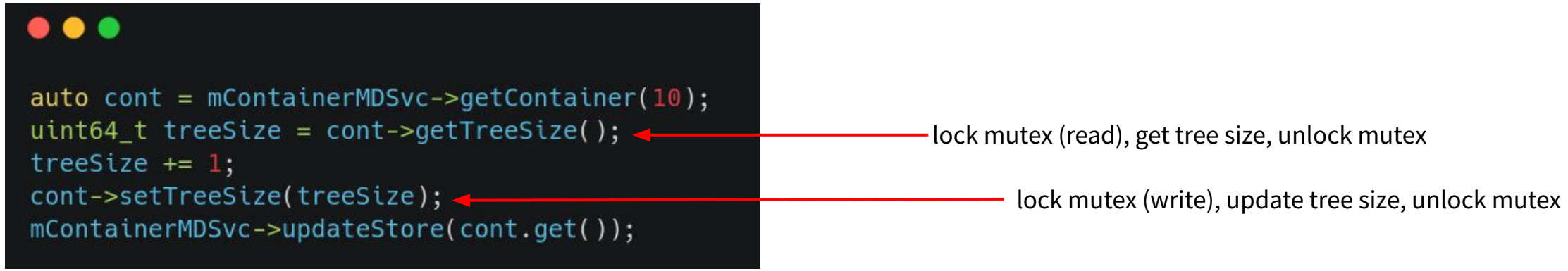
Using finer-grained locks - before the evolution

```
auto cont = mContainerMDSvc->getContainer(10);
uint64_t treeSize = cont->getTreeSize();
treeSize += 1;
cont->setTreeSize(treeSize);
mContainerMDSvc->updateStore(cont.get());
```

Using finer-grained locks - before the evolution



Using finer-grained locks - before the evolution



Goal: run multiple operations under the same lock! (not the global NS one…)

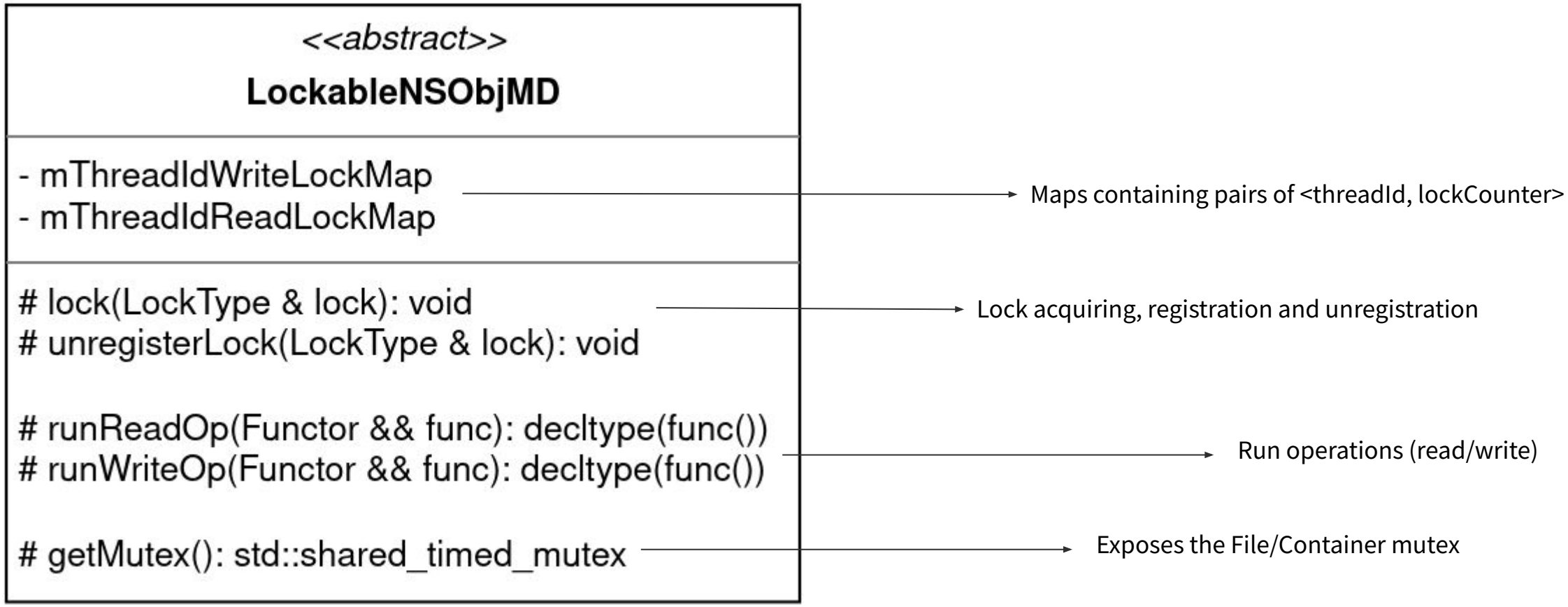
Run multiple operations under the same lock

Creation of the LockableNSObjMD abstract class

- Lock registration
- Run read/write operation
    - Taking into account lock registration

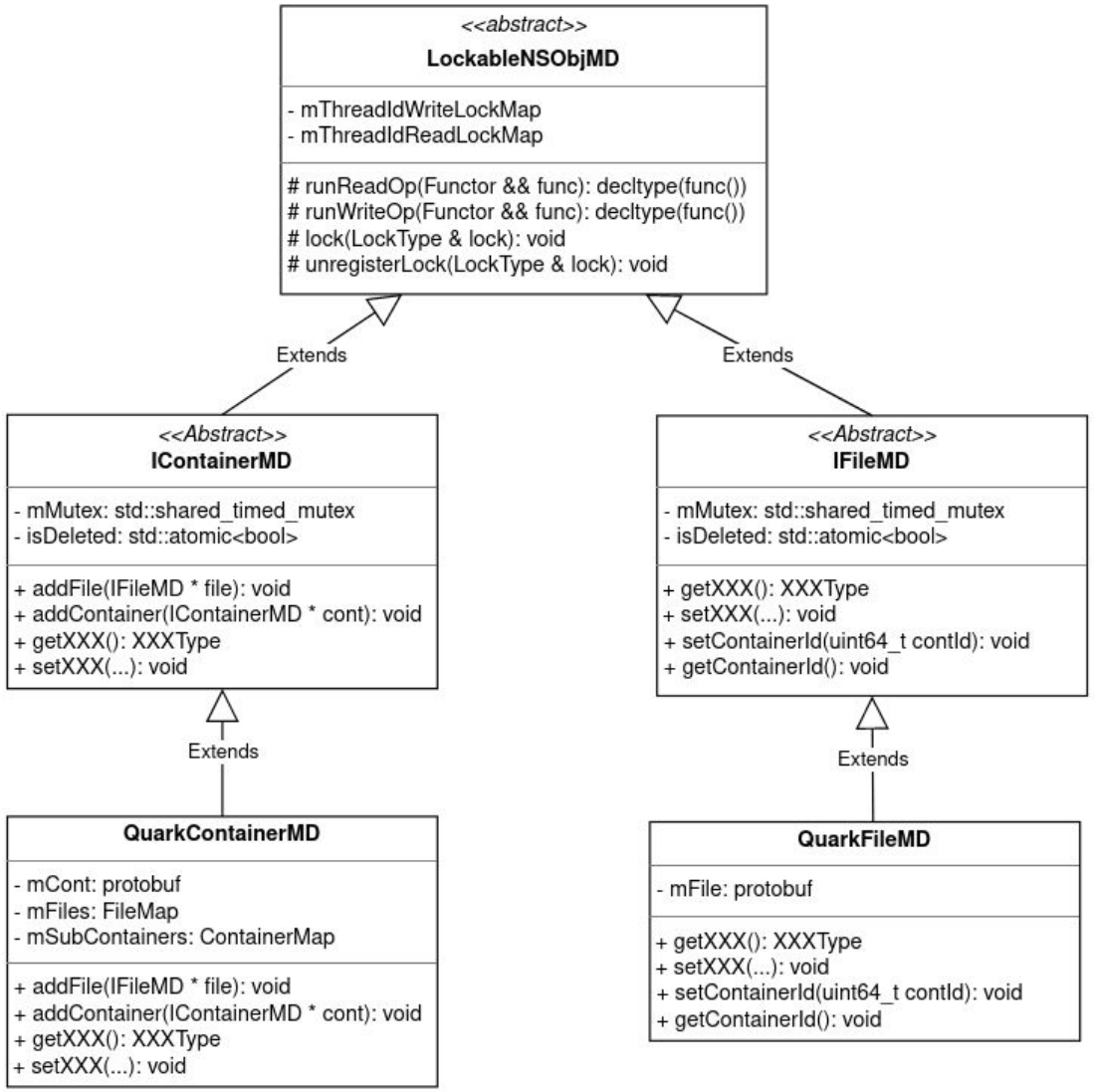
Run multiple operations under the same lock

Creation of the LockableNSObjMD interface



Run multiple operations under the same lock

Let's use it!



Run multiple operations under the same lock

Let's use it!

```
inline void setTreeSize(uint64_t treesize) override
{
  std::unique_lock<std::shared_timed_mutex> lock(mMutex);
  mCont.set_tree_size(treesize);
}
```

Run multiple operations under the same lock

Let's use it!

```
inline void setTreeSize(uint64_t treesize) override
{
  std::unique_lock<std::shared_timed_mutex> lock(mMutex);
  mCont.set_tree_size(treesize);
}
```

```
inline void setTreeSize(uint64_t treesize) override
{
  runWriteOp([this,treesize](){
    mCont.set_tree_size(treesize);
  });
}
```

Run multiple operations under the same lock

Let's use it!

```
inline uint64_t  getTreeSize() const override
{
  std::shared_lock<std::shared_timed_mutex> lock(mMutex);
  return mCont.tree_size();
}
```

Run multiple operations under the same lock

Let's use it!

```
inline uint64_t  getTreeSize() const override
{
  std::shared_lock<std::shared_timed_mutex> lock(mMutex);
  return mCont.tree_size();
}
```

```
inline uint64_t getTreeSize() const override
{
  return runReadOp([this](){
    return mCont.tree_size();
  });
}
```

Are we done?

Run multiple operations under the same lock

No, we only provided a lock-registration mechanism and member functions to run read or write operations…

Run multiple operations under the same lock

Let's provide a File/Container locking class

- RAII-style (Resource Acquisition Is Initialization)
  - Constructor will lock the File/Container object
  - Destructor will unlock it

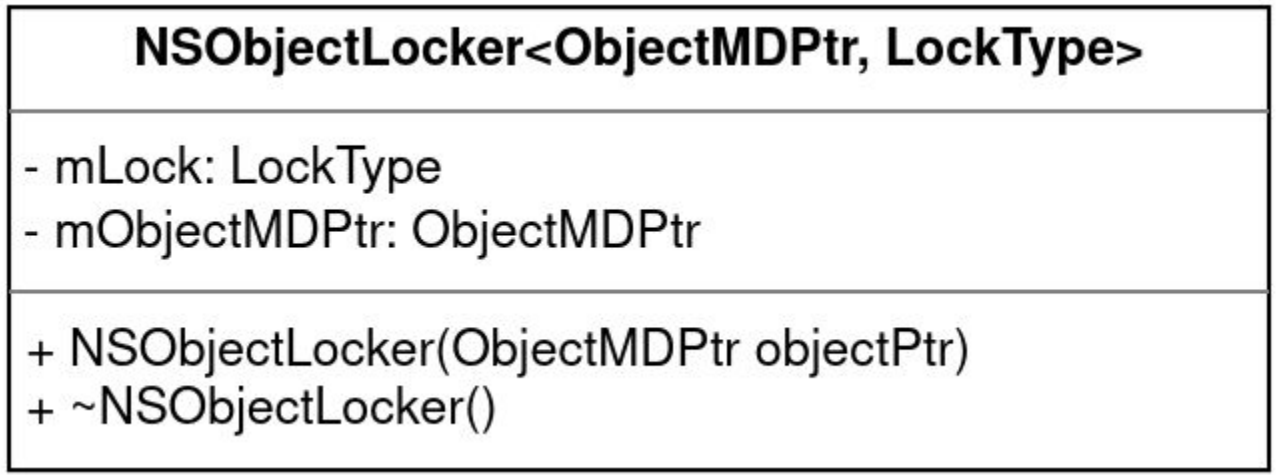
Run multiple operations under the same lock

Let's provide a File/Container locking class

- RAII-style (Resource Acquisition Is Initialization)
  - Constructor will lock the File/Container object
  - Destructor will unlock it
- Acts on shared_ptr of Container/Files
  - shared_ptr are returned by the NS persistency layer

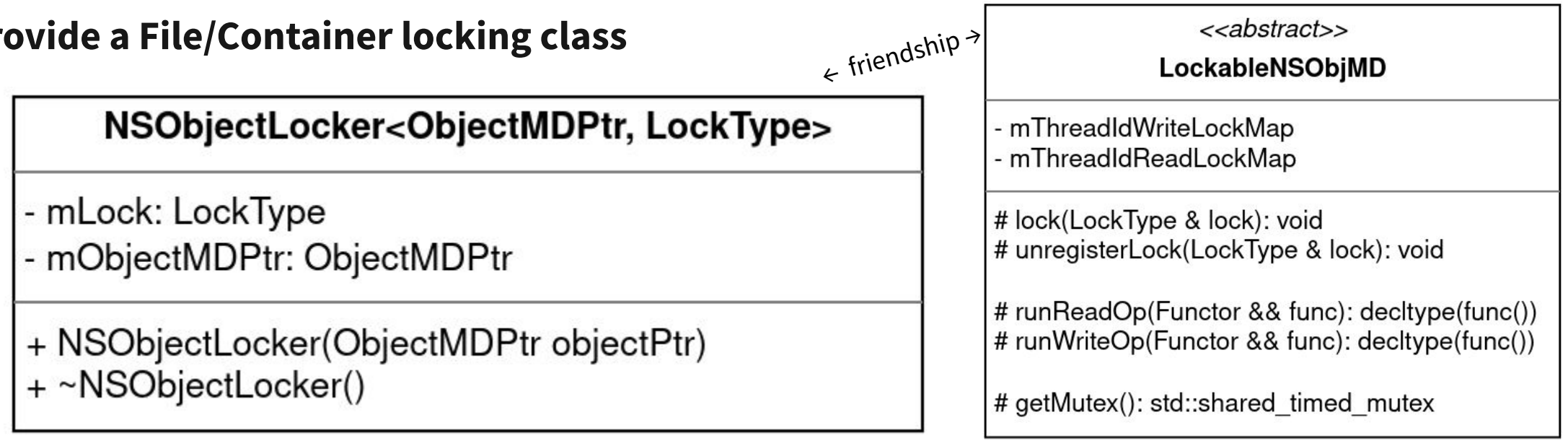
Run multiple operations under the same lock

Let's provide a File/Container locking class



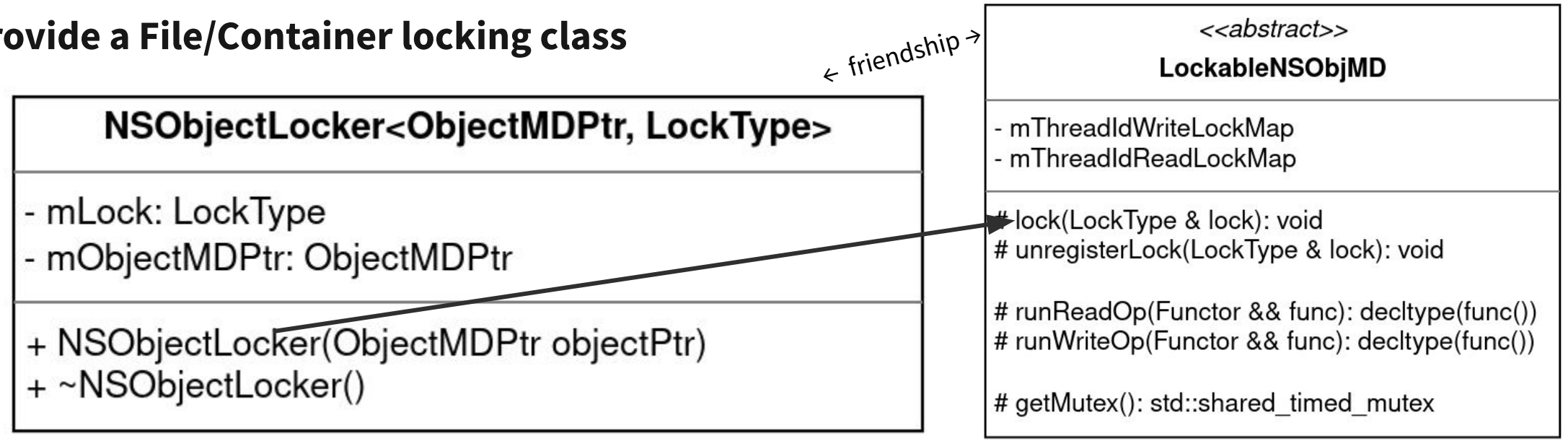
Run multiple operations under the same lock

Let's provide a File/Container locking class



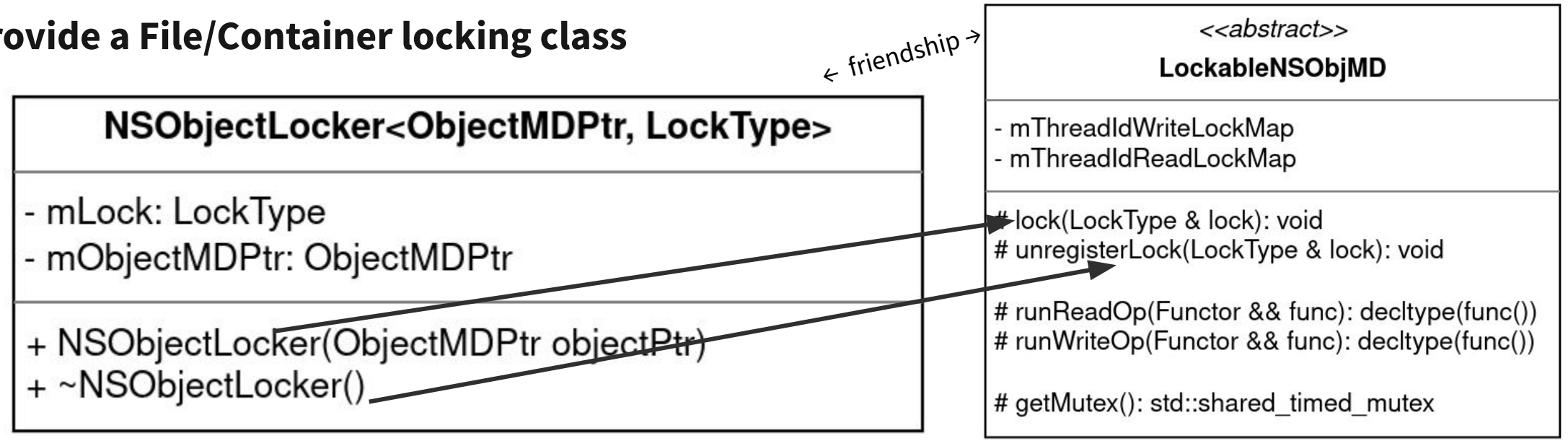
Run multiple operations under the same lock

Let's provide a File/Container locking class



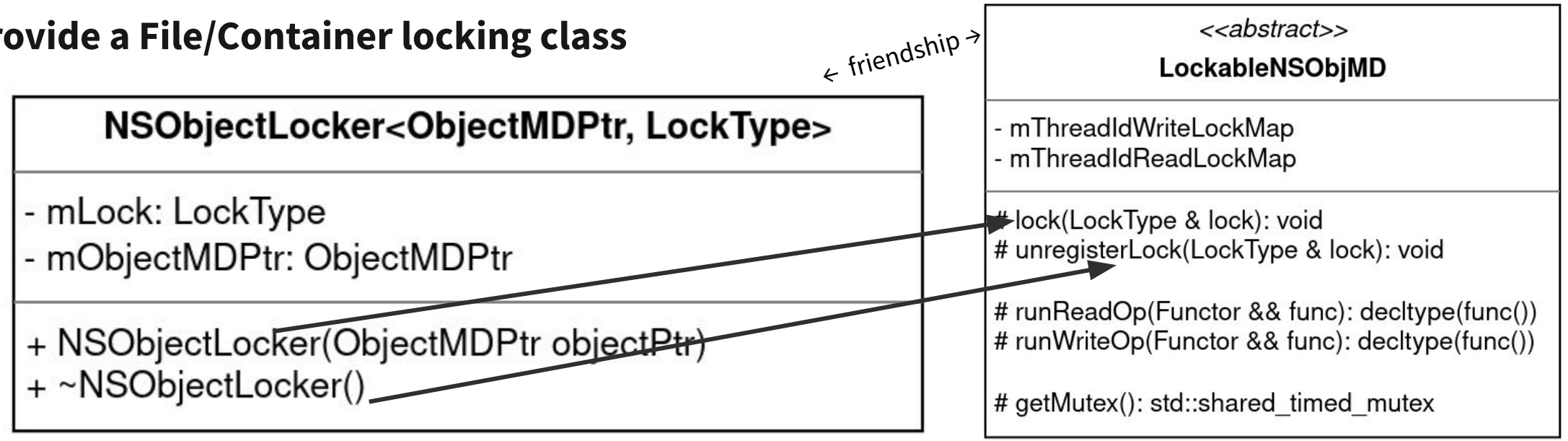
Run multiple operations under the same lock

Let's provide a File/Container locking class



Run multiple operations under the same lock

Let's provide a File/Container locking class




```
using IContainerMDReadLocker = NSObjectMDLocker<IContainerMDPtr,MDReadLock>;
using IContainerMDWriteLocker = NSObjectMDLocker<IContainerMDPtr,MDWriteLock>;

using IFileMDReadLocker = NSObjectMDLocker<IFileMDPtr,MDReadLock>;
using IFileMDWriteLocker = NSObjectMDLocker<IFileMDPtr,MDWriteLock>;
```

Run multiple operations under the same lock

Usage of the NSObjectLocker class

```
{
  auto cont = mContainerMDSvc->getContainer(10);
  eos::IContainerMD::IContainerMDWriteLock lock(cont);
  // Do stuff while being write locked
}
```

Application to the tree size update thread

```
//DON'T WRITE LOCK THE NAMESPACE LOCk
// eos::common::RWMutexWriteLock wr_lock(gOFS->eosViewRWMutex)
std::shared_ptr<IContainerMD> cont;
for (auto const& elem : containers) {
  try {
    // Get the container by ID
    cont = mContainerMDSvc->getContainerMD(elem.first);
    // Write-lock it
    eos::IContainerMD::IContainerMDWriteLocker locker(cont);
    // Update the container tree size (simple setter)
    cont->updateTreeSize(elem.second);
    // Persist the container to QuarkDB
    mContainerMDSvc->updateStore(cont.get());
  } catch (const MDException& e) {
    continue;
  }
}
```

mutex not used anymore :)

The good things about these changes

- Interfaces left untouched
- We don't change the current behaviour if we don't use the locker class
- Easy to use for programmers
- Unit tested → Creation of a MockContainerMD to check the locking/unlocking registration…

EOS operations using the new namespace locking

- Files / directories move operation - *eos mv source destination*
- Internal find - former *eos find …* used for recursive operations
- Extended attributes management - *eos attr ls/get/set… (recursive and non-recursive)*

Results

Results

Results - deadlocks

EOS / EOS-6064

MGM stuck (namespace locking)

Edit | Add comment | Assign | More | Resolved

Details

Type:	Bug	Resolution:	Fixed
Priority:	Minor	Fix Version/s:	5.2.17
Affects Version/s:	5.2.9		
Component/s:	MGM	Security Level:	Internal Data (Only authenticated CERN users can see this issue)
Labels:	None		

Description

EOSHOME-i02 was stuck for several hours this morning (i.e did not recover by itself), had to be restarted manually.
Initial investigation points to some issue with the new 5.2.x namespace locking.

We have several traces:

```
MGM on eoshome-i02-vega.cern.ch is 'hanging' since 9986sec, thread stacks in /var/eos/md/stacktrace.*, eu-stack in /root/mgm_eustack_47
```

Results - deadlocks

Reproducer

- Create two directories (a, b) with 1000 files each
- Perform n **parallel** file move from a to b and b to a
  - *eos mv a/i b/i &*
  - *eos mv b/j a/j &*
- Delete a and b in parallel
  - *eos rm -rF a &*
  - *eos rm -rF b &*

Results - files / directory move operation - deadlocks

Faulty code in *eos mv* logic…

```
// Move to a new directory
// Lock file to move
eos::IFileMD::IFileMDWriteLocker fileWriteLocker(file);
// Lock source and destination directory
eos::BulkNsObjectLocker<eos::IContainerMDPtr,
eos::IContainerMD::IContainerMDWriteTryLocker> helper;
helper.add(dir);
helper.add(newdir);
auto dirsLock = helper.lockAll();
// Perform the file move...
dir->removeFile(oPath.GetName());
// [...]
```

Results - files / directory move operation - deadlocks

Faulty code in *eos mv* logic…

```
// Move to a new directory
// Lock file to move
eos::IFileMD::IFileMDWriteLocker fileWriteLocker(file);
// Lock source and destination directory
eos::BulkNsObjectLocker<eos::IContainerMDPtr,
eos::IContainerMD::IContainerMDWriteTryLocker> helper;
helper.add(dir);
helper.add(newdir);
auto dirsLock = helper.lockAll();
// Perform the file move...
dir->removeFile(oPath.GetName());
// [...]
```

Results - files / directory move operation - deadlocks

Faulty code in *eos mv* logic…

```
// Move to a new directory
// Lock file to move
eos::IFileMD::IFileMDWriteLocker fileWriteLocker(file);
// Lock source and destination directory
eos::BulkNsObjectLocker<eos::IContainerMDPtr,
eos::IContainerMD::IContainerMDWriteTryLocker> helper;
helper.add(dir);
helper.add(newdir);
auto dirsLock = helper.lockAll();
// Perform the file move...
dir->removeFile(oPath.GetName());
// [...]
```

Thread 1 (*eos mv /a/i /b/j*)	Thread 2 (*eos rm -rF /a/*)
lock file i	

Results - files / directory move operation - deadlocks

Faulty code in *eos mv* logic…

```
// Move to a new directory
// Lock file to move
eos::IFileMD::IFileMDWriteLocker fileWriteLocker(file);
// Lock source and destination directory
eos::BulkNsObjectLocker<eos::IContainerMDPtr,
eos::IContainerMD::IContainerMDWriteTryLocker> helper;
helper.add(dir);
helper.add(newdir);
auto dirsLock = helper.lockAll();
// Perform the file move...
dir->removeFile(oPath.GetName());
// [...]
```

Thread 1 (*eos mv /a/i /b/j*)	Thread 2 (*eos rm -rF /a/*)
lock file i	- lock dir a - listChildren - lock file i **(CANNOT because locked by thread 1)**

Results - files / directory move operation - deadlocks

Faulty code in *eos mv* logic…

```
// Move to a new directory
// Lock file to move
eos::IFileMD::IFileMDWriteLocker fileWriteLocker(file);
// Lock source and destination directory
eos::BulkNsObjectLocker<eos::IContainerMDPtr,
eos::IContainerMD::IContainerMDWriteTryLocker> helper;
helper.add(dir);
helper.add(newdir);
auto dirsLock = helper.lockAll();
// Perform the file move...
dir->removeFile(oPath.GetName());
// [...]
```

Thread 1 (*eos mv /a/i /b/j*)	Thread 2 (*eos rm -rF /a/*)
lock file i	- lock dir a - listChildren - lock file i **(CANNOT because locked by thread 1)**
lock dir a **(CANNOT because locked by thread 2)**	

Results - files / directory move operation - deadlocks

Faulty code in *eos mv* logic…

```
// Move to a new directory
// Lock file to move
eos::IFileMD::IFileMDWriteLocker fileWriteLocker(file);
// Lock source and destination directory
eos::BulkNsObjectLocker<eos::IContainerMDPtr,
eos::IContainerMD::IContainerMDWriteTryLocker> helper;
helper.add(dir);
helper.add(newdir);
auto dirsLock = helper.lockAll();
// Perform the file move...
dir->removeFile(oPath.GetName());
// [...]
```

Thread 1 (*eos mv /a/i /b/j*)	Thread 2 (*eos rm -rF /a/*)
lock file i	- lock dir a - listChildren - lock file i **(CANNOT because locked by thread 1)**
lock dir a **(CANNOT because locked by thread 2)**	DEADLOCK
DEADLOCK	

Results - files / directory move operation - deadlocks

Faulty code in *eos mv* logic… And the solution

```
// Move to a new directory
// Lock file to move
eos::IFileMD::IFileMDWriteLocker fileWriteLocker(file);
// Lock source and destination directory
eos::BulkNsObjectLocker<eos::IContainerMDPtr,
eos::IContainerMD::IContainerMDWriteTryLocker> helper;
helper.add(dir);
helper.add(newdir);
auto dirsLock = helper.lockAll();
// Perform the file move...
dir->removeFile(oPath.GetName());
// [...]
```

```
// Move to a new directory
// Lock source and destination directory
eos::BulkNsObjectLocker<eos::IContainerMDPtr,
eos::IContainerMD::IContainerMDWriteTryLocker> helper;
helper.add(dir);
helper.add(newdir);
auto dirsLock = helper.lockAll();
// Lock file to move
eos::IFileMD::IFileMDWriteLocker fileWriteLocker(file);
// Perform the file move...
dir->removeFile(oPath.GetName());
// [...]
```

ALL GOOD! Fixed in EOS 5.2.17

Results - files / directory move operation - deadlocks

Faulty code in *eos mv* logic… Lesson learnt

- Having individual object locks instead of a global one adds complexity

Results - files / directory move operation - deadlocks

Faulty code in *eos mv* logic… Lesson learnt

- Having individual object locks instead of a global lock adds complexity
- The order of locking is important!
  - Lock containers before locking files
    - The Namespace is browsed by containers, then files

Results - files / directory move operation - deadlocks

Faulty code in *eos mv* logic… Lesson learnt

- Having individual object locks instead of a global one adds complexity
- The order of locking is important!
  - Lock containers before locking files
    - The Namespace is browsed by containers, then files
  - If two containers need to be locked
    - Lock them by the ascending order of their identifier
    - Try-lock mechanism
      - If the 2nd directory could not be locked, unlock the first one and retry until you could lock both

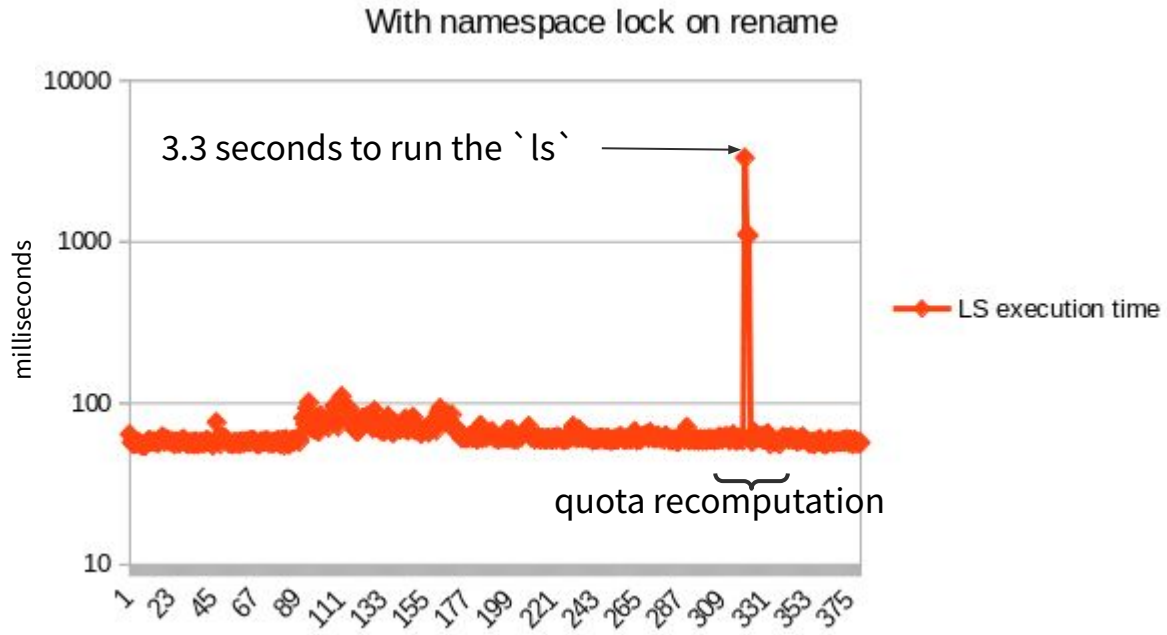
Results - files / directory move operation - deadlocks

Faulty code in *eos mv* logic… Lesson learnt

- Having individual object locks instead of a global one adds complexity
- The order of locking is important!
  - Lock containers before locking files
    - The Namespace is browsed by containers, then files
  - If two containers need to be locked
    - Lock them by the ascending order of their identifier
    - Try-lock mechanism
      - If the 2nd directory could not be locked, unlock the first one and retry until you could lock both
- It's not because it works on my dev box that it will work in production
  - Tests namespace read and write operations with high parallelism

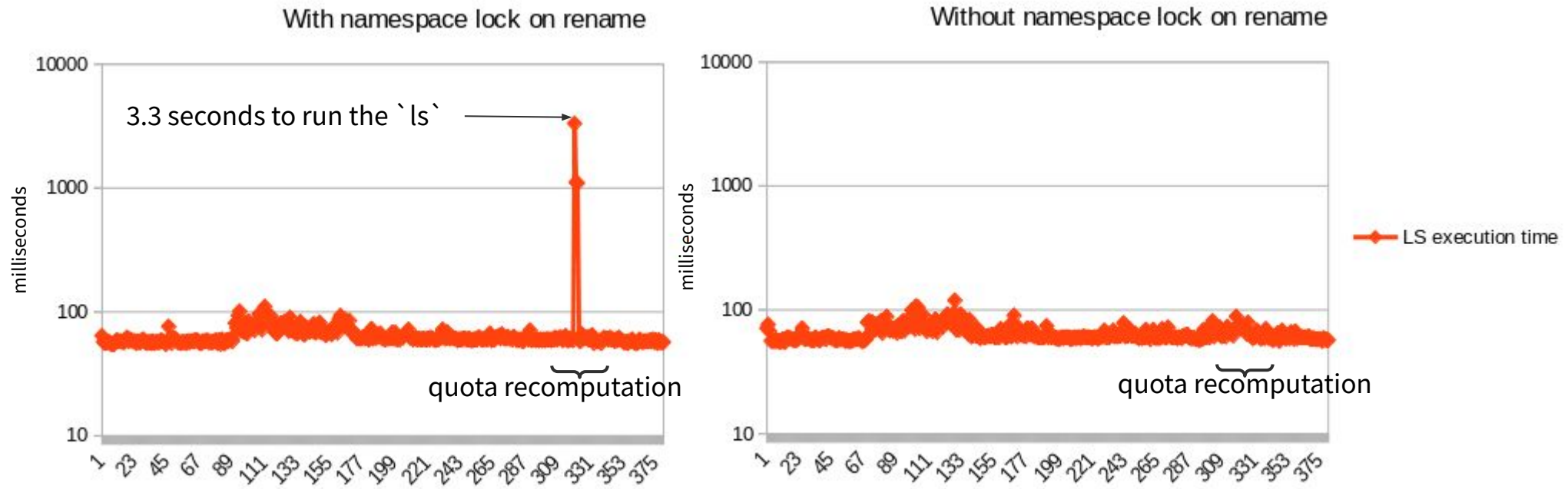
Results - files / directory move operation

- Test
  - Run `eos ls` in infinite loop
  - Move a directory containing 10k files from one quota dir to another quota dir (triggers quota computation)



Results - files / directory move operation

- Test
  - Run `eos ls` in infinite loop
  - Move a directory containing 10k files from one quota to another quota (triggers quota computation)



Results - Extended attribute management

- Create **n** directories, each containing 1000 directories
- Compute the time to run *eos attr -r set sys.test=1* in **n** directories in parallel (recursive find + attr set)

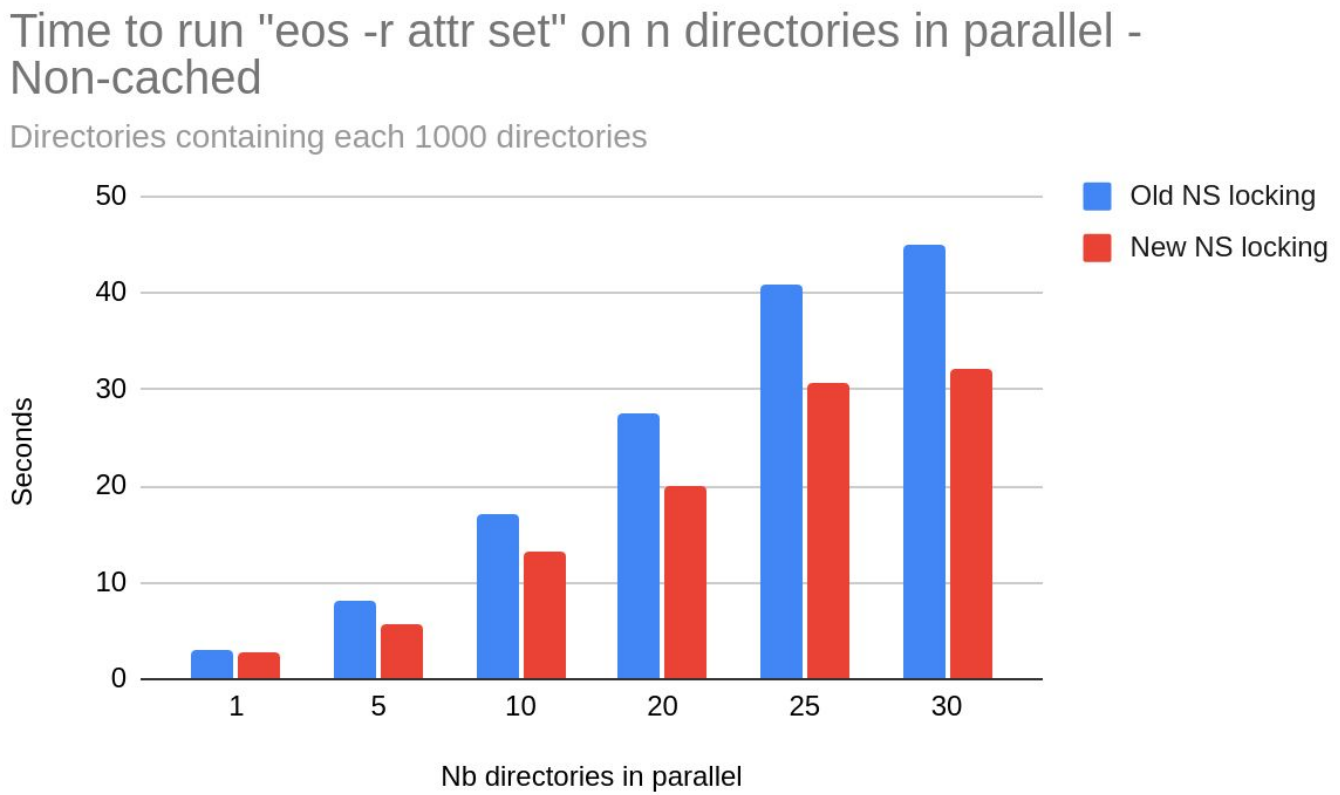
Results - Extended attribute management

- Create **n** directories, each containing 1000 directories
- Compute the time to run *eos attr -r set sys.test=1* in **n** directories in parallel (recursive find + attr set)

```
for j in `seq 1 $n`
do
  eos attr -r set sys.test=1 $dirName/$j/ &
done
wait
```

Results - Extended attribute management

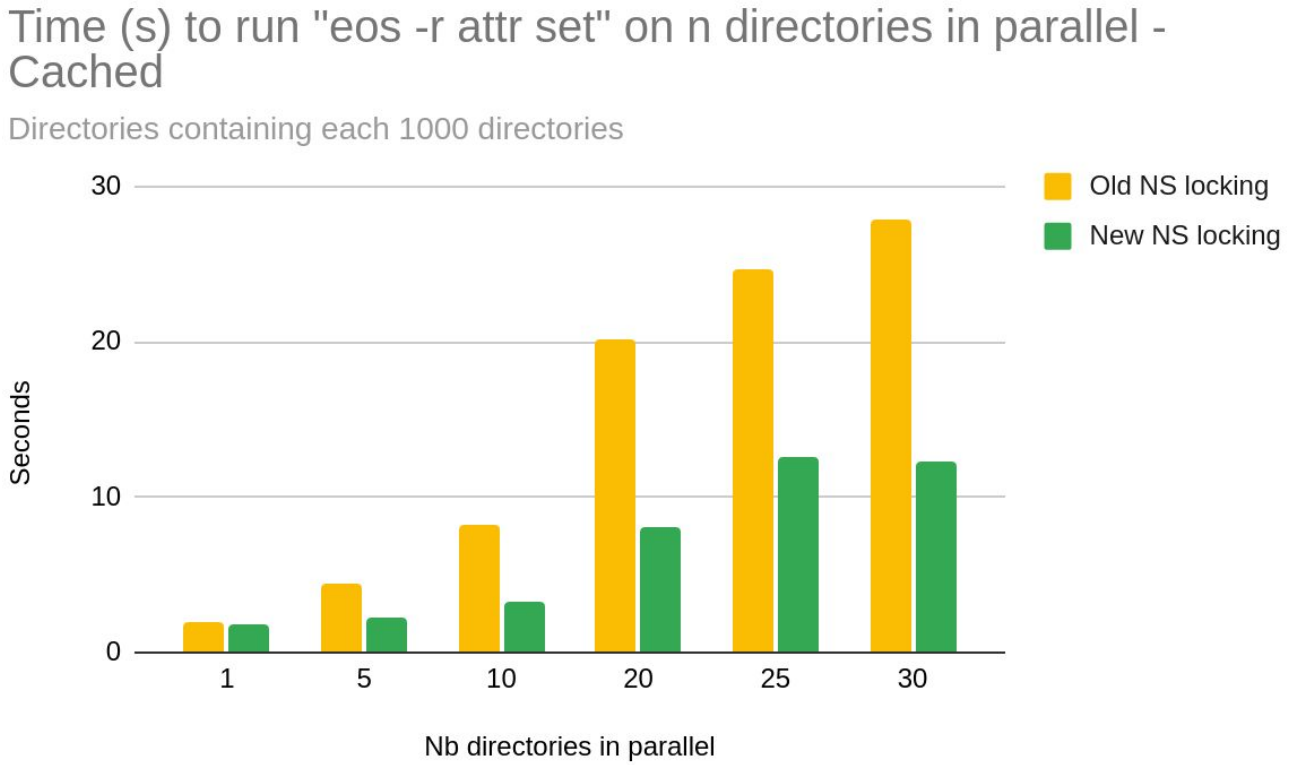
- Create **n** directories, each containing 1000 directories
- Compute the time to run *eos attr -r set sys.test=1* in **n** directories in parallel (recursive find + attr set)

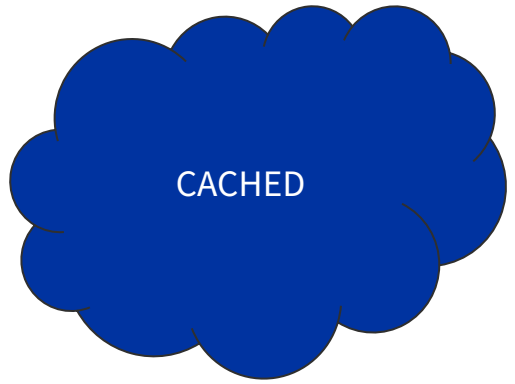


NON CACHED

Results - Extended attribute management

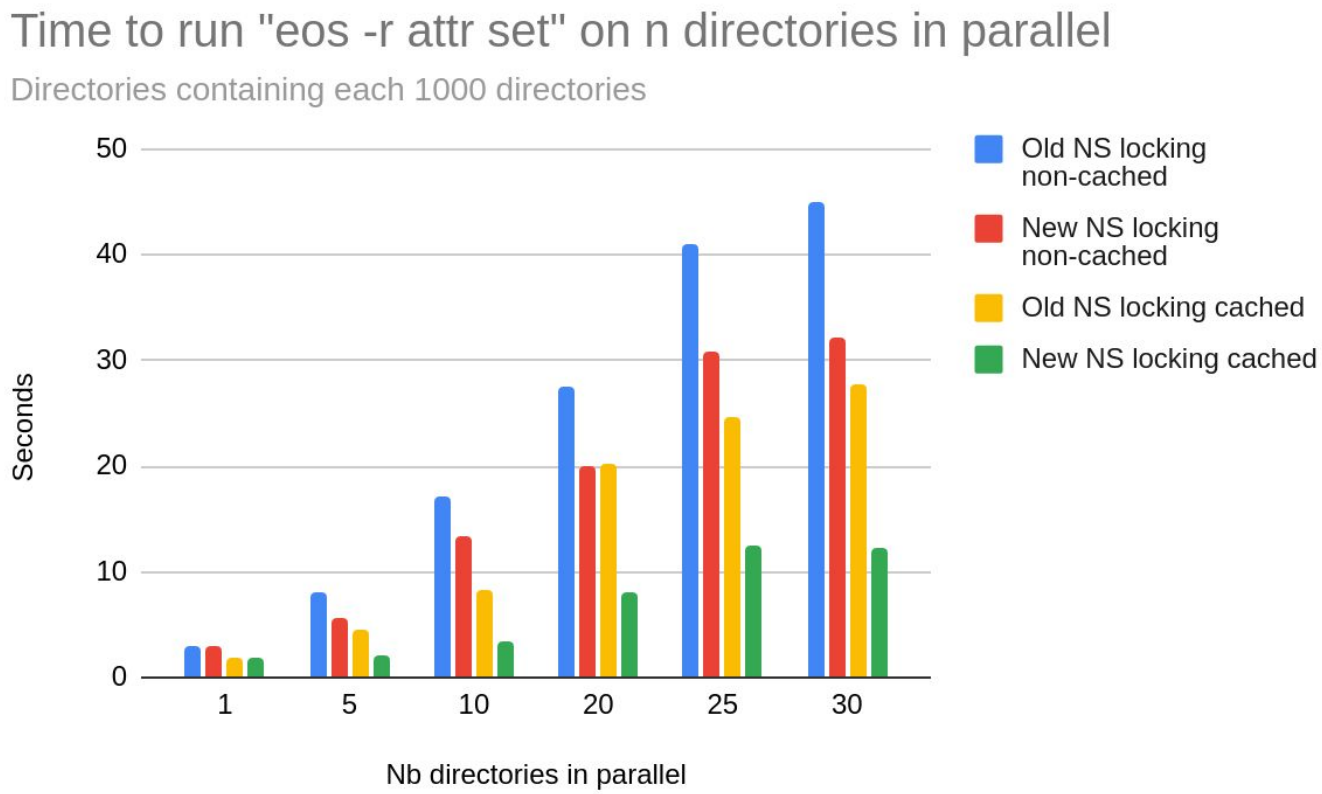
- Create **n** directories, each containing 1000 directories
- Compute the time to run *eos attr -r set sys.test=1* in **n** directories in parallel (recursive find + attr set)





Results - Extended attribute management

- Create **n** directories, each containing 1000 directories
- Compute the time to run *eos attr -r set sys.test=1* in **n** directories in parallel (recursive find + attr set)



Conclusion

- We managed to perform multiple File/ContainerMD operations under the same lock
  - RAII-style locking class + lock-registration mechanism

Conclusion

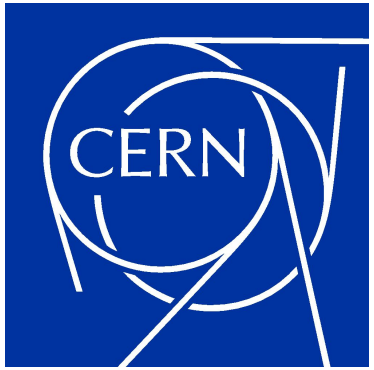
- We managed to perform multiple File/ContainerMD operations under the same lock
  - RAII-style locking class + lock-registration mechanism
- We are gradually removing the EOS global namespace lock
  - File/Directory move done
  - internal EOS find + recursive extended attribute management

Conclusion

- We managed to perform multiple File/ContainerMD operations under the same lock
  - RAII-style locking class + lock-registration mechanism
- We are gradually removing the EOS global namespace lock
  - File/Directory move done
  - internal EOS find + recursive extended attribute management
- Still a lot of work to be done
  - The EOS global namespace mutex is used everywhere in the EOS MGM code!

Conclusion

- We managed to perform multiple File/ContainerMD operations under the same lock
  - RAII-style locking class + lock-registration mechanism
- We are gradually removing the EOS global namespace lock
  - File/Directory move done
  - internal EOS find + recursive extended attribute management
- Still a lot of work to be done
  - The EOS global namespace mutex is used everywhere in the EOS MGM code!
- It's dangerous to change it… (beware of deadlocks)
- …But we can already observe the positive effect on performance thanks to a higher parallelism



home.cern