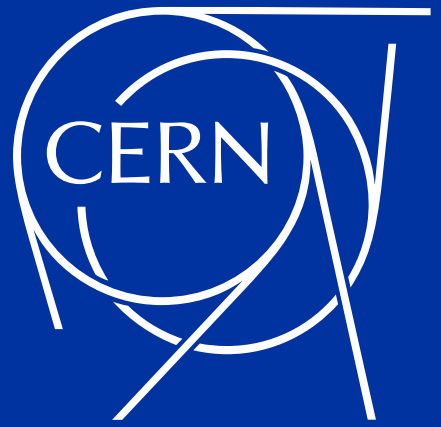


A brief overview of the CTA mount scheduling logic

Cédric Caffy on behalf of the CTA team

Wednesday the 3rd of March 2021

Introduction

The need for a scheduling logic

- Optimize drive usage
  - $Tape\ mount\ time + tape\ head\ positioning + tape\ unmount = 2\ minutes$
- Tape infrastructure is **shared by all the experiments**
- Meet the user needs
  - *How long a user is willing to wait to get his files from tape ?*

Tape mount scheduling in CTA

- Is a tape worth to be mounted ?
  - If yes, mount it
  - If multiple tapes are worth, **prioritize** !
    - Data **archival** is more important than data **retrieval**

Plan

I. Generalities

a. The CTA scheduling logic is ran concurrently

b. The CTA objectstore (simplified)

c. The CTA queues

II. The scheduling parameters

a. Is a tape worth to be mounted ?

b. Ensure the fair-share of the tape infrastructure between the experiments (VO)

c. Prioritize if multiple tapes are worth to be mounted

III. The scheduling process

a. The queueing of a user request

b. General overview of the scheduling

c. Fetch all the potential mounts

d. Fetch all the existing mounts

e. Filtering of the potential mounts

f. Sorting of the potential mounts

I. Generalities

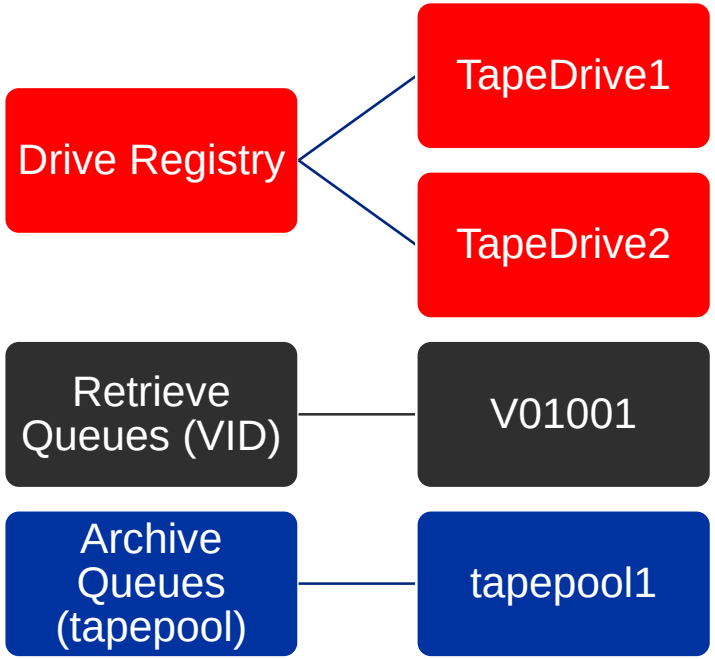
The CTA scheduling logic is ran concurrently

- Every tapeserver of the CTA instance try to schedule a mount !
  - Occurs every 10 seconds if they are not transferring any data
- Race conditions avoided by the internal implementation of the CTA objectstore

I. Generalities

The CTA objectstore (simplified)

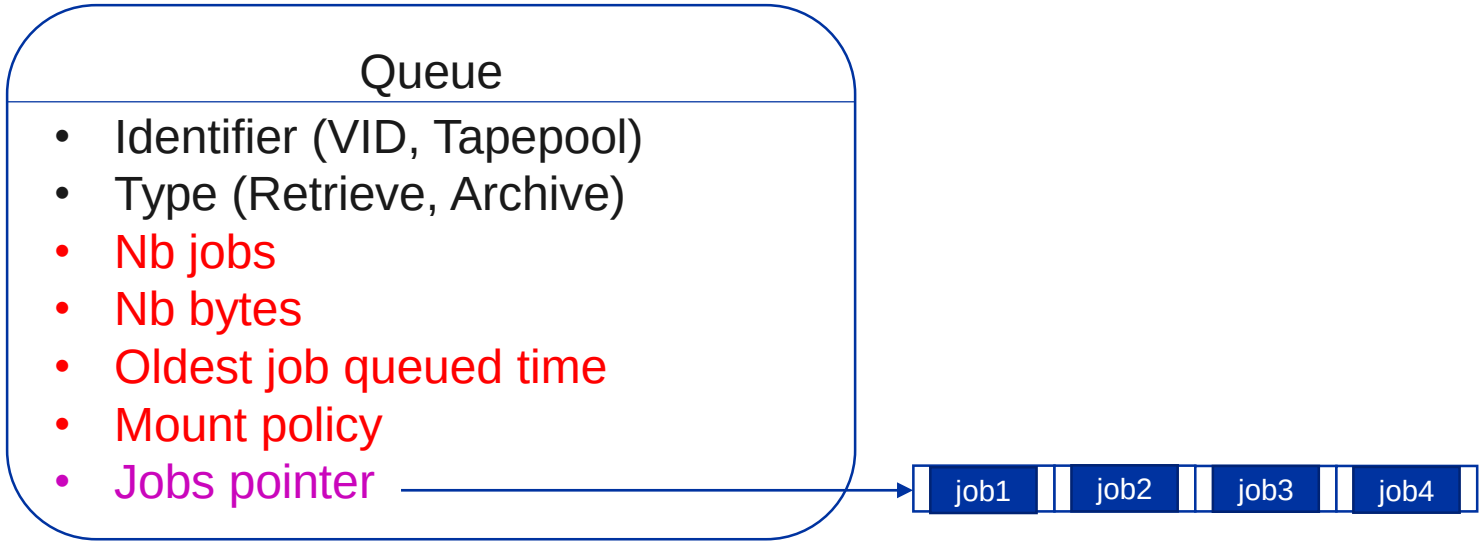
- Also called **Scheduler Database**
  - **NOT** to be confused with the CTA Catalogue database…



I. Generalities

The CTA queues

- Inform the scheduler that work is available !
- CTA queue
  - Unique identifier
  - Type (Retrieve, Archive)
  - Statistics about the queued jobs
  - Pointer to the queued jobs



II. The scheduling parameters

Is a tape worth to be mounted ?

- Optimize drive usage
  - The ***minimum amount of bytes queued*** to warrant a mount

II. The scheduling parameters

The minimum of bytes queued to warrant a mount

- Configurable value in the tapeserver configuration file

II. The scheduling parameters

Is a tape worth to be mounted ?

- Optimize drive usage
  - The ***minimum amount of bytes queued*** to warrant a mount
- Meet user needs
  - The ***minimum request age*** after which a mount should be scheduled

II. The scheduling parameters

The minimum request age after which a mount should be scheduled

- Comes from the **Mount Policy**
  - Attached to the queue at *request queueing*

Mount Policy	Minimum Request age (seconds)	Priority
mountPolicy1	10800	2

II. The scheduling parameters

Is a tape worth to be mounted ?

- Optimize drive usage
  1. The ***minimum amount of bytes queued*** to warrant a mount
- Meet user needs
  2. The ***minimum request age*** in a queue after which a mount should be scheduled

Ensure the fair-share of the tape infrastructure between the experiments (VO)

3. Limitation of the **amount of allocated drives** per expirement (VO)

II. The scheduling parameters

Limitation of the amount of allocated drives per experiment (VO)

- Limitation of the amount of allocated drives per experiment (VO)
- Done by **Virtual Organization (VO)**

VO	Read max drives	Write max drives
XYZ	6	2

II. The scheduling parameters

Is a tape worth to be mounted ?

- Optimize drive usage
  1. The ***minimum amount of bytes queued*** to warrant a mount
- Meet user needs
  2. The ***minimum request age*** in a queue after which a mount should be scheduled

Ensure the fair-share of the tape infrastructure between the experiments (VO)

3. Limitation of the **amount of allocated drives** per expirement (VO)

Prioritize if multiple tapes are worth to be mounted

4. Mount **priority**

II. The scheduling parameters

Mount priority

- Comes from the **Mount Policy**
  - Attached to the queue at *request queueing*

Mount Policy	Minimum Request age (seconds)	Priority
mountPolicy1	10800	2

II. The scheduling parameters

Summary

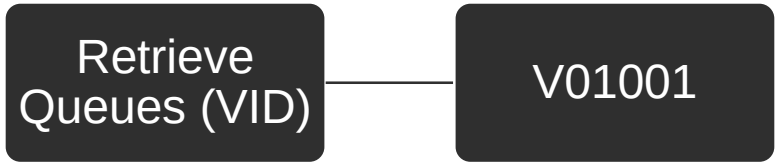
4 scheduling parameters in total

- **3** scheduling **filter** parameters
  1. The ***minimum amount of bytes queued*** to warrant a mount
  2. The ***minimum request age*** in a queue after which a mount should be scheduled (Mount Policy)
  3. Limitation of the **amount of allocated drives** per expirement (VO)
- **1 priority** scheduling parameter
  4. Mount priority (Mount Policy)

III. The scheduling process

The queueing of the user request (Retrieve)

- **Retrieve requests are queued by tape name (Volume IDentifier or VID)**

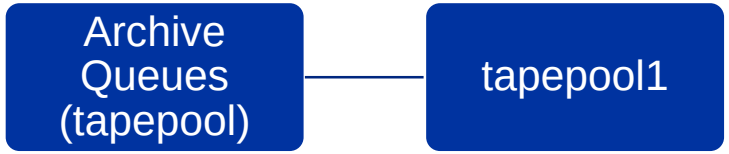


  - The CTA Catalogue database contains information about where the files are located on tape

III. The scheduling process

The queueing of the user request (Archive)

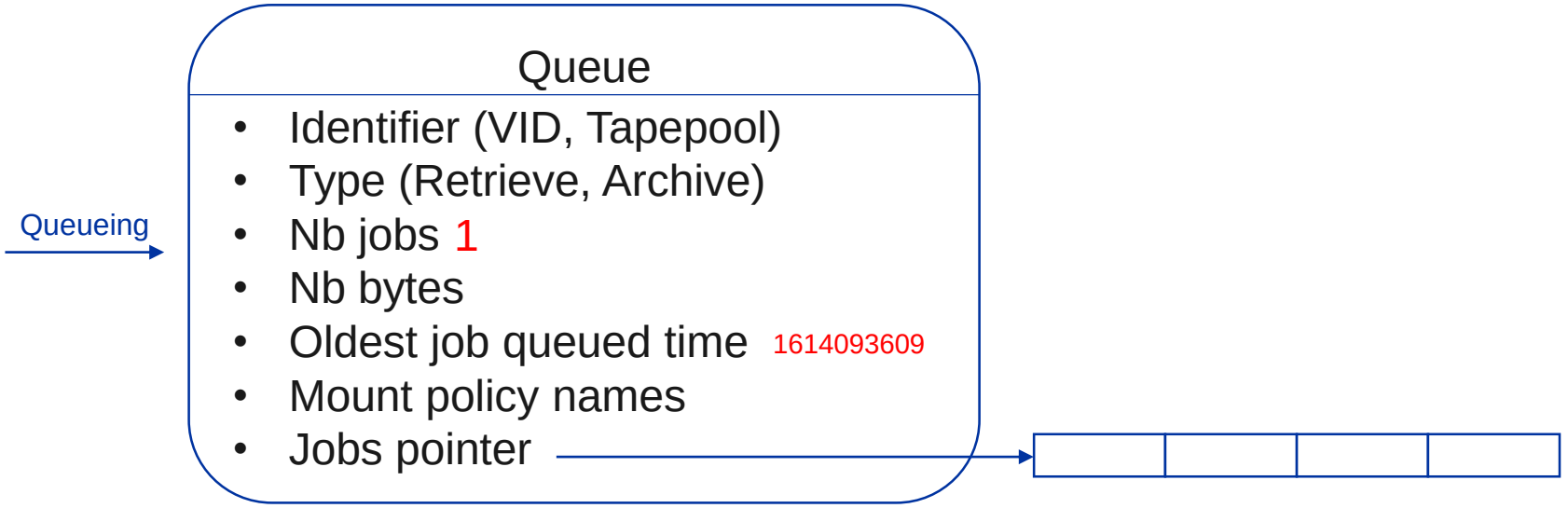
- **Archive** requests are queued by **tapepool** name



  - When we queue, we do not know the tape where the files will go
  - We only know on which **pool** of tapes the files have to go

III. The scheduling process

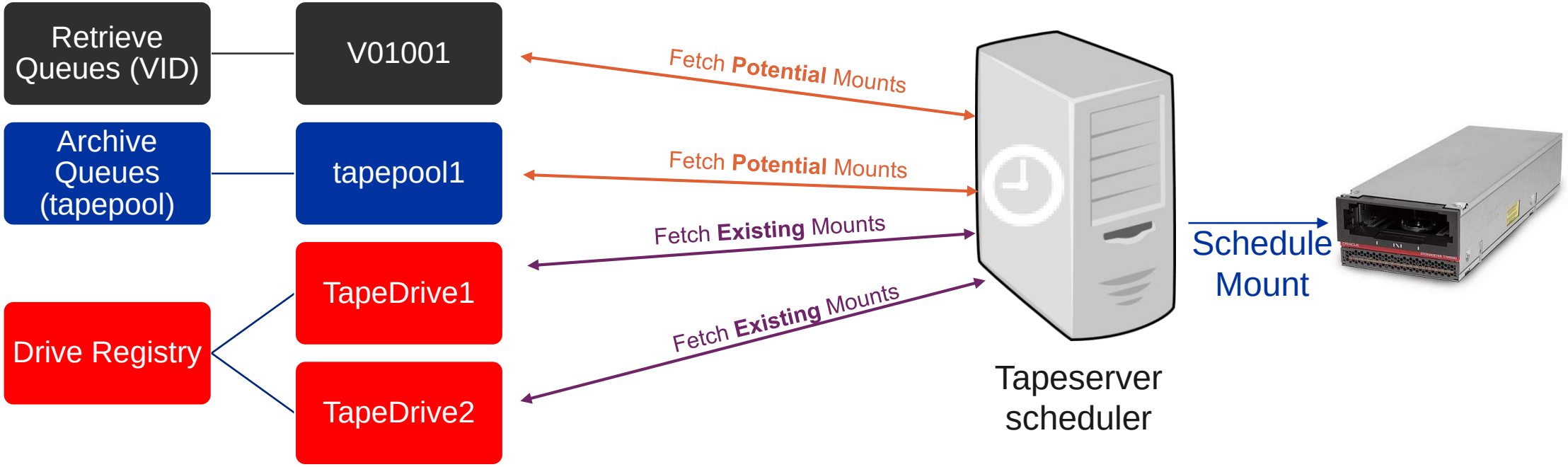
The queueing of the user request (Archive or Retrieve)



III. The scheduling process

General overview of the scheduling

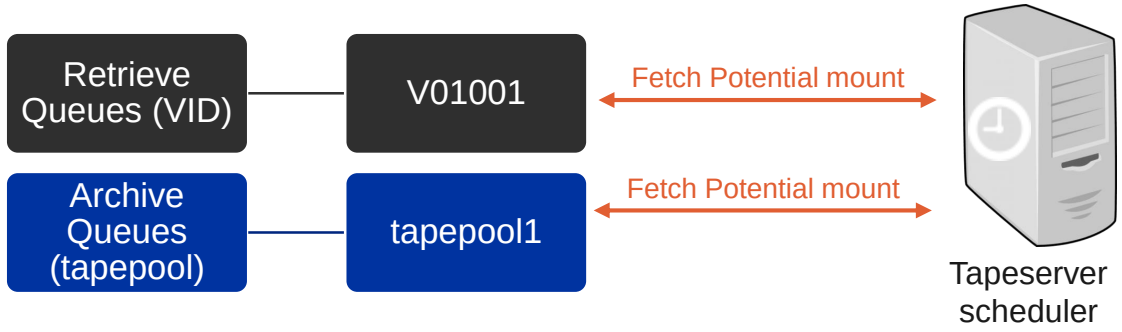
- Reminder: Scheduling occurs concurrently every 10 seconds when tapeservers have no activity



III. The scheduling process

Fetch all the potential mounts

- Loop through all the queues *statistics*
  - Create a list of **potential mounts**
- **Potential mount**
  - C++ object that contains the statistics of a specific queue
- **List of Potential Mounts**
  - *« What can we actually mount ? »*

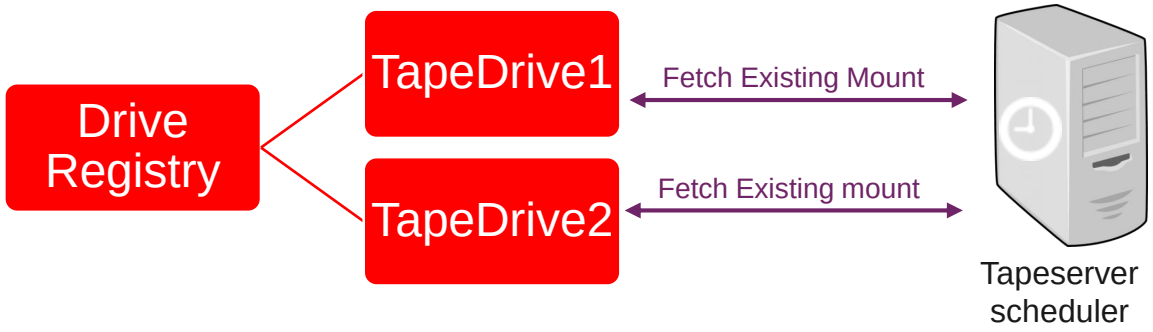


PotentialMount
- Identifier (VID, Tapepool) - MountType (Retrieve, Archive) - NbJobs - NbBytes - OldestJobQueueTime - MountPolicy (min req age + priority) - VO
+ bool operator<(const PotentialMount &other)

III. The scheduling process

Fetch all the existing mounts

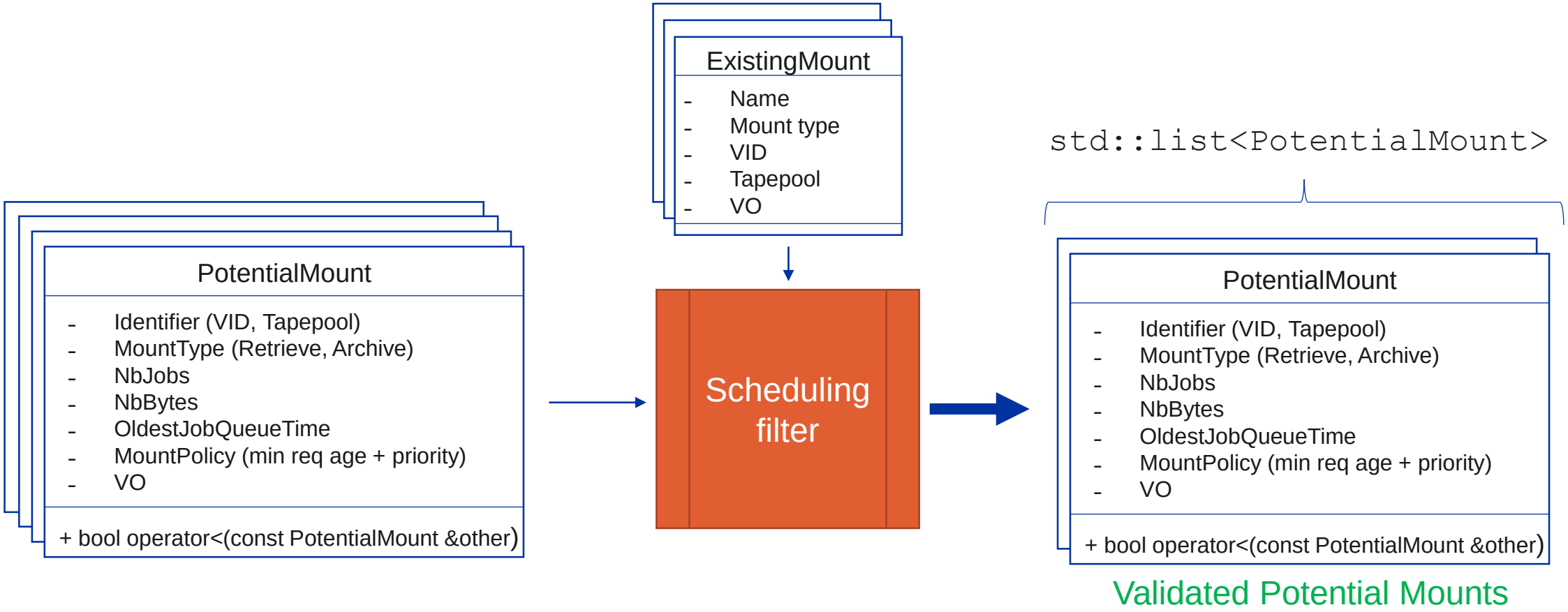
- Loop through all the drives
  - Create a list of **existing mounts**
- **Existing mount**
  - C++ object that contains information about what a drive is doing
- **List of existing mounts**
  - *« Which drives are working and for which experiment (VO) ? »*



ExistingMount
- Name - Mount type - VID - Tapepool - VO

III. The scheduling process

Filter the list of the Potential Mounts

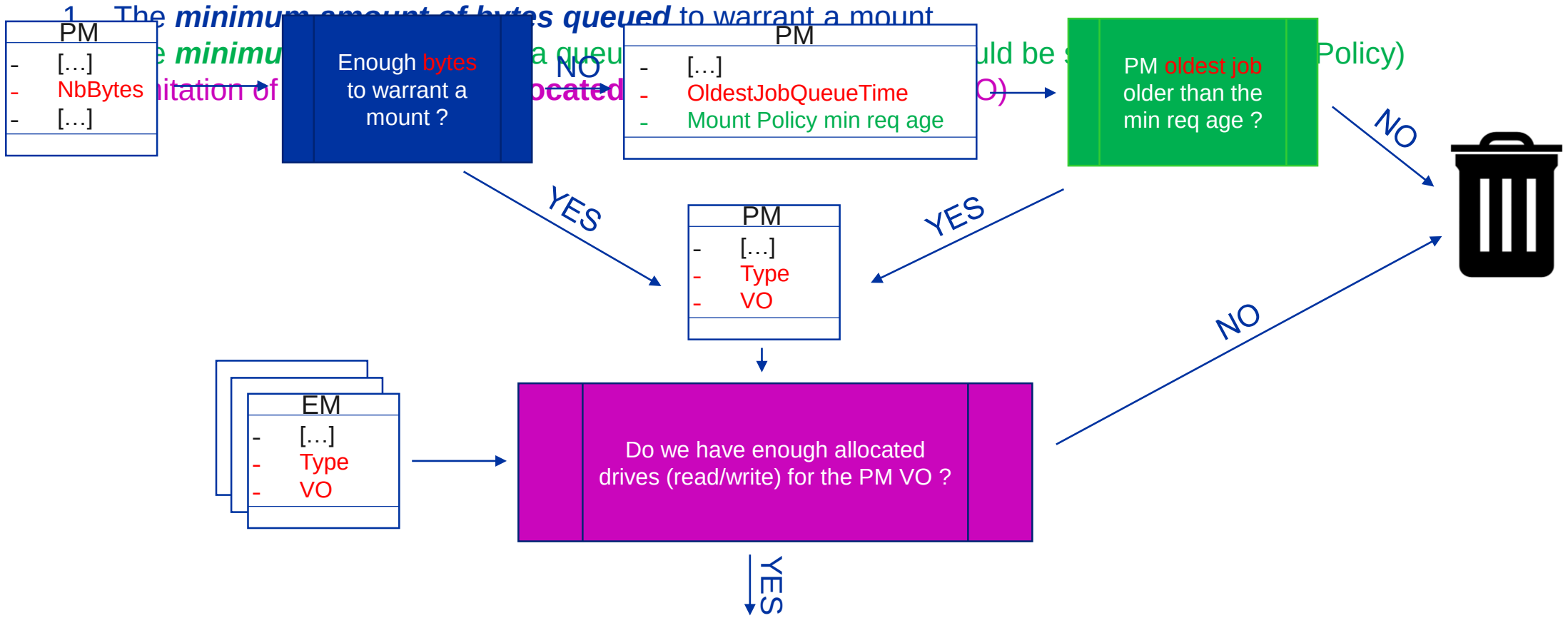


III. The scheduling process

Scheduling filter

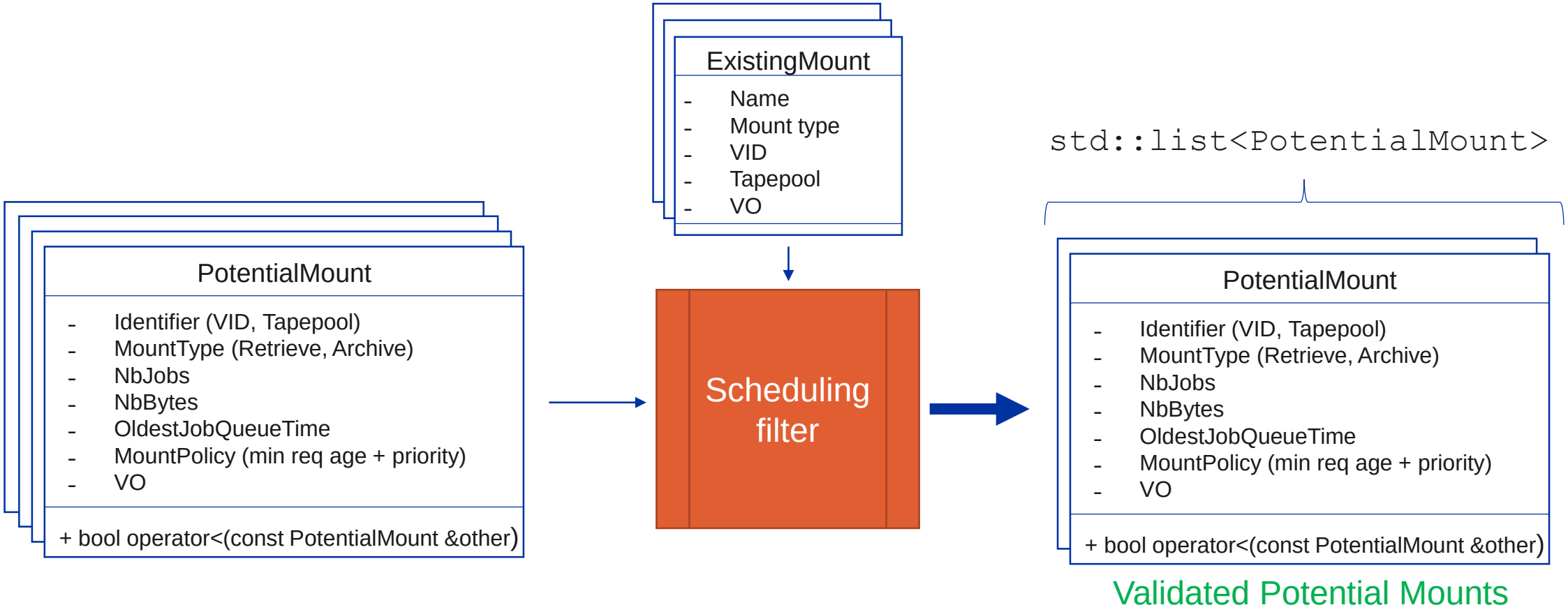
Filter the list of the Potential Mounts (PM)

- Usage of the 3 scheduling **filter** parameters



III. The scheduling process

Filter the list of validated Potential Mounts



II. The scheduling process

Sort the list of validated PotentialMount

- Usage of the 4th scheduling parameter
  - Mount priority

`std::list<PotentialMount>`

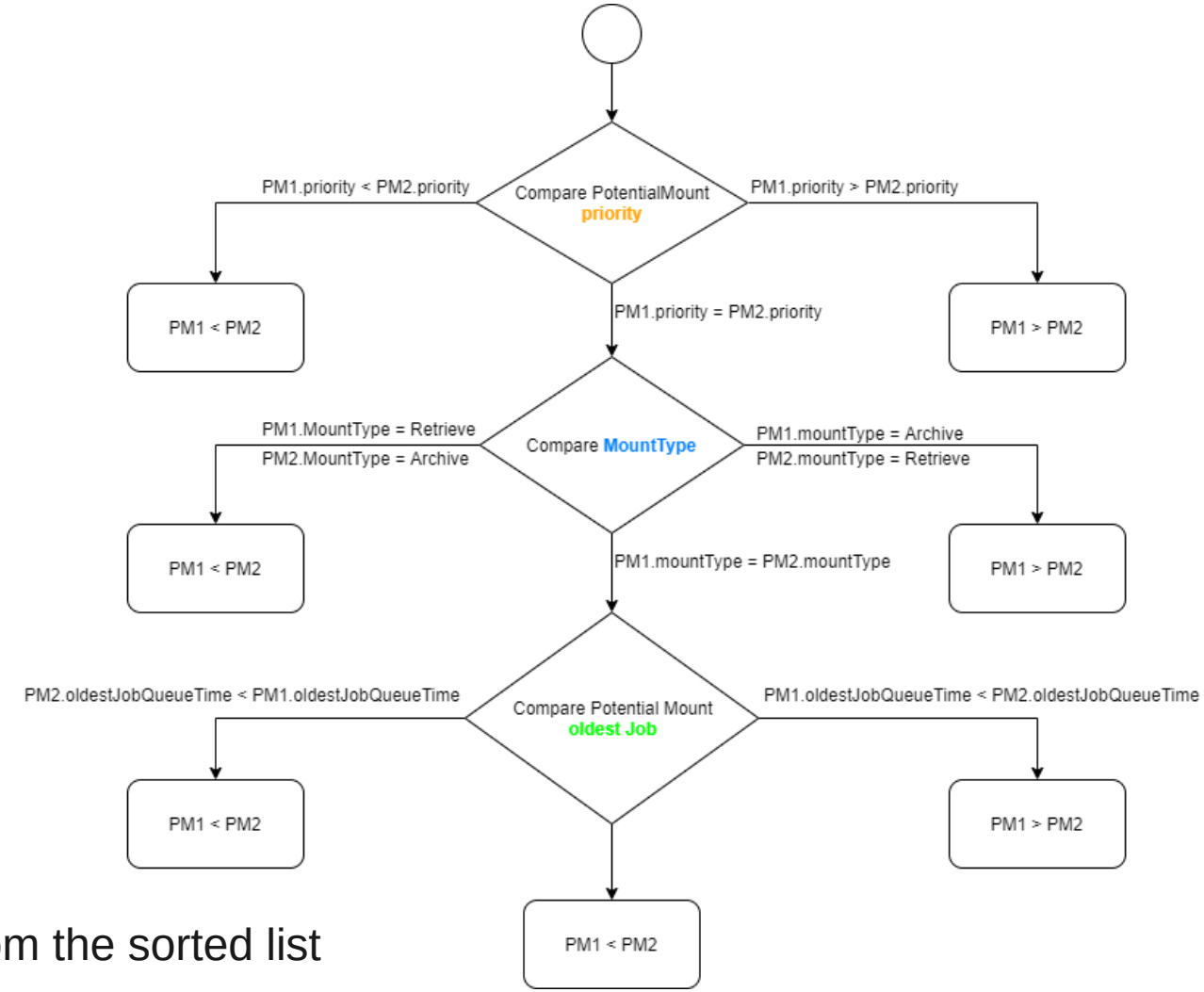
PotentialMount

- Identifier (VID, Tapepool)
- MountType (Retrieve, Archive)
- NbJobs
- NbBytes
- OldestJobQueueTime
- MountPolicy (min request age + priority)
- VO

+ bool operator<(const PotentialMount &other)

Validated Potential Mounts

`std::sort`

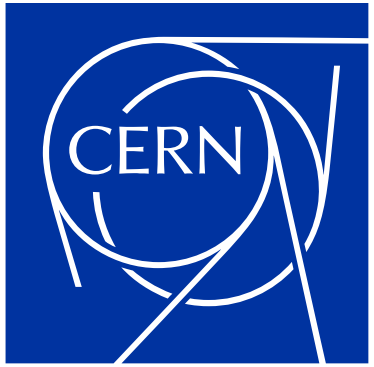


- Return the last PotentialMount (higher priority) from the sorted list
  - For **data archival**, get the tape to write files to from the Catalogue

III. The scheduling process

Summary

- Scheduling is ran **concurrently** by all the tapeservers
  - Occurs **every 10 seconds** if the tapeserver is inactive
- **4 scheduling parameters**
  - 3 **scheduling filter** parameters
  - 1 **priority** parameter
- **Potential Mounts (PM)** are fetched by looping over the objectstore queues statistics
- **Existing Mounts (EM)** are fetched by looping over the objectstore drive information
- Potential Mounts are **filtered** and then **sorted** by using the **4 scheduling parameters**
- The **top priority PotentialMount** will be used by the tapeserver to trigger a new mount



home.cern