SEPHORIZON LEARNING

# Data Management Concepts for Efficient and User-Friendly HPC

Sepehr Mahmoodian

sephorizon.com

Sephorizon edition – November 2026

# Prior Knowledge

- You should have some very basic experience with using our HPC systems:
  - Log in to the HPC via SSH
  - Navigate with the terminal
  - Copy data
  - Submit jobs
- You are right in this course if you are struggling to
  - organize your own data set
  - understand what is SCRATCH(-GRETE, -EMMY), HOME, PROJECT, LOCAL, ...
  - share data within a project
  - maintain oversight of your input and output data
  - reasonable I/O performance
  - understand if your data is secure
  - adhere to good scientific practice

# Learning Objectives

- At the end of this course you should be able to:
  - Use tiered storage systems employed by HPC systems
    - to optimize performance and prevent data loss
  - Organize your HPC workloads within workflows
    - Implement user isolation and data sharing in projects using UNIX permissions
  - Design your own HPC workflows
    - Describe independent tasks and data sets with standardized annotations
    - Classify these steps and data sets using annotations
    - Identify the correct storage tier/system based on these annotations
  - Identify scenarios where a data catalog is useful
  - Recognize opportunities to ask for dedicated data management support
    - E.g. via our Ticket system

# Your Trainer


- **Sepehr Mahmoodian**
  - PhD researcher at GWDG and the University of Göttingen
  - Part of AG C
  - HPC, cloud and distributed systems
  - s.mahmoodian@gwdg.de
  - sephorizon.com
  - LinkedIn: sepehr-mahmoodian




# Table of Contents

1 Motivation

2 Experimental Planning

3 Conceptual Mapping

4 Metadata

5 Data Staging

6 Permissions

7 Practical

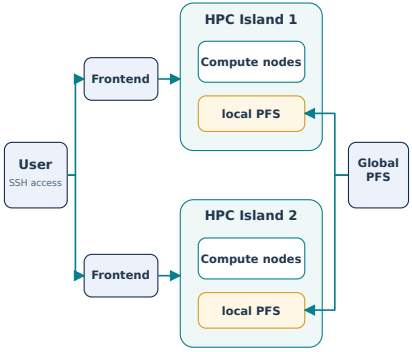
8 Summary

# Motivation for Data and Life Cycle Management

- Modern HPC systems are really, really complicated
- It is unbelievably easy to lose oversight
- This makes collaborations close to impossible
- The only solution is careful planning
- And rigorous execution of this plan
- This plan has to become the single source of truth

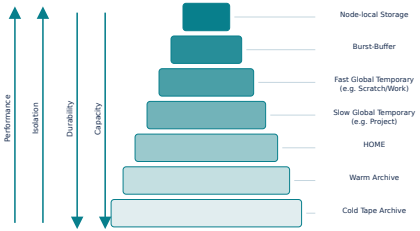
# General Overview of HPC Architecture

- The different clusters form compute islands
  - For example, Emmy, Grete
- Each of these islands has its own fast storage
- And there are more centralized storage
- Access to each island via centralized Slurm
- Not GWDG specific, see Dragonfly topologies



# Why Storage Tiering?

- In the previous example, two storage tiers were shown
- Users have different requirements depending on the type of data
  - Think of Software as compared to hot data
- The different storage systems differ in many attributes, e.g.
  - Size
  - Speed
  - Data Durability
  - Backups
  - **Locality**
  - Lifetime



# Historical GWDG Storage Example

| Category | Speed | Size | Durability | Size Project[1] | Size User[1] | Purpose |
|---|---|---|---|---|---|---|
| HOME | medium | medium | snapshots, backups | | ~ 40 GiB | User HOMEs |
| Project | medium | medium | snapshots, backups | ~ 3 TiB | | Medium term data |
| Workspaces | medium to fast | big | RAID only | ~ 10 / 40 TiB | | Active data |
| SCRATCH/WORK ([2]) | fast | big | RAID only | ~ 2 / 12 TiB[3] | ~ 1 / 2 TB[3] | Active data |
| COLD | medium | big | snapshots | ~ 10 TiB | | Medium term data |
| ARCHIVE/PERM | very slow | medium | backups | ~ 8 TiB | | Archival to tape |

- Historical teaching example; capacities, quotas and policies may change
- Check current GWDG storage documentation before use

# Historical GWDG Example: Filesystems (1/3)

| Project/ User | Name | Filesystem Kind | Where | Pool/ Fileset | Quota Enforced | Quota Kind | How Collected |
|---|---|---|---|---|---|---|---|
| Project | VAST NHR | VAST | VAST | NHR | yes | directory | bulk REST call |
| | SCRATCH MDC (HDD) | DDN Lustre | NHR MDC Lustre | HDD | yes | project | call program for each |
| | SCRATCH MDC (SSD) | DDN Lustre | NHR MDC Lustre | SSD | yes | project | call program for each |
| | SCRATCH RZG | DDN Lustre | NHR RZG Lustre | | yes | project | call program for each |
| | SCC SCRATCH | BeeGFS | SCC SCRATCH | | no | | |
| | CIDBN SCRATCH | BeeGFS | CIDBN SCRATCH | | no | | |
| | Cramer/Soeding SCRATCH | BeeGFS | Cramer/Soeding SCRATCH | | no | | |
| | VAST KISSKI | VAST | VAST | KISSKI | yes | directory | bulk REST call |

Historical teaching example. See current storage documentation for policies and limits.

# Historical GWDG Example: Filesystems (2/3)

| Project/ User | Name | Filesystem Kind | Where | Pool/ Fileset | Quota Enforced | Quota Kind | How Collected |
|---|---|---|---|---|---|---|---|
| | VAST REACT | VAST | VAST | REACT | yes | directory | bulk REST call |
| | Ceph HDD | CephFS | Ceph HDD | | yes | directory | reading xattr on each directory |
| | PERM | Stornext | NHR PERM | | yes | directory | bulk report made by gperm |
| User | NHR HOME (VAST) | VAST | VAST | NHR | yes | directory | bulk REST call |
| | SCC HOME (VAST) | VAST | VAST | standard | yes | directory | bulk REST call |
| | SCC HOME (Stornext) | Stornext | SCC HOME | various | yes | user | |
| | KISSKI HOME | VAST | VAST | KISSKI | yes | directory | bulk REST call |
| | REACT HOME | VAST | VAST | REACT | yes | directory | bulk REST call |

Historical teaching example. See current storage documentation for policies and limits.

# Historical GWDG Example: Filesystems (3/3)

| Project/ User | Name | Filesystem Kind | Where | Pool/ Fileset | Quota Enforced | Quota Kind | How Collected |
|---|---|---|---|---|---|---|---|
| | SCRATCH MDC (HDD) | DDN Lustre | NHR MDC Lustre | HDD | yes | project | call program for each |
| | SCRATCH MDC (SSD) | DDN Lustre | NHR MDC Lustre | SSD | yes | project | call program for each |
| | SCRATCH RZG | DDN Lustre | NHR RZG Lustre | | yes | project | call program for each |
| | SCC SCRATCH | BeeGFS | SCC SCRATCH | | no | | |
| | CIDBN SCRATCH | BeeGFS | CIDBN SCRATCH | | no | | |
| | Cramer/Soeding SCRATCH | BeeGFS | Cramer/Soeding SCRATCH | | no | | |
| | PERM | Stornext | NHR PERM | | yes | group (legacy), directory (inodes and for new) | bulk report made by gperm |

Historical teaching example. See current storage documentation for policies and limits.

# Resulting Problem

- People are overwhelmed and do wrong data placement
  - Hot data sits on cold storage
  - Standard datasets sit on expensive backed up storages
  - Important results are on fragile storage
  - The wrong storage system for the wrong cluster island is used
- Many storage tiers quickly lead to a loss of oversight
  - Data is not cleaned up
  - Data is not reproducible, unclear where it belongs to
  - Data loss, hard to find
  - How can I select all data with a certain property?

# Abstraction Level Goal

**Available options**



**User perspective**



# Interactive Q&A

- Did you experience any problems with our (storage) systems?
- Were you aware of this circumstance?
- What were your solutions?

# Outline

1 Motivation

2 Experimental Planning

3 Conceptual Mapping

4 Metadata

5 Data Staging

6 Permissions

7 Practical

8 Summary

# Experimental Planning I

- The experimental planning is defined within a data management plan (DMP)
  - It defines properties and policies of the measured or computed data
  - It is a living document
  - It should be designed at the beginning of a project and then continuously updated
- It starts at the data source(s)
  - Where is what data generated by whom?
  - How much data is generated per day/week/month?
  - Does the data need to be checked for validity?
  - How can the data be transferred from the source and to where?
  - Usually, raw data is extremely valuable, how is it being backed up
  - Who needs access to it?
  - What are the semantic metadata to describe this data?

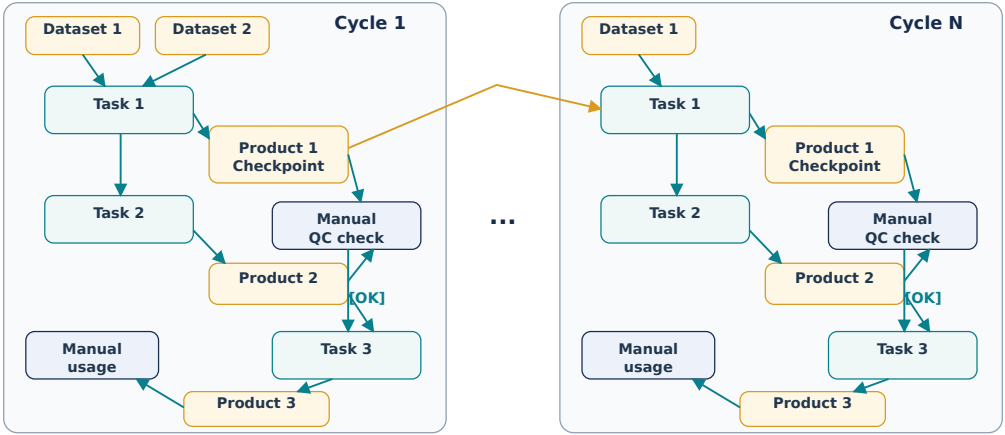
# Experimental Planning II

- Define the first processing step
  - Is it automatic or manual/interactive?
  - Who does it?
  - Is it compute intensive?
  - Is it I/O intensive?
  - What are the data products?
- First Data product
  - Only temporary/intermediate result?
  - Should it be retained?
  - Manual check?
  - Semantic Metadata
  - How will it be processed?
  - Will it be used IO intensively?

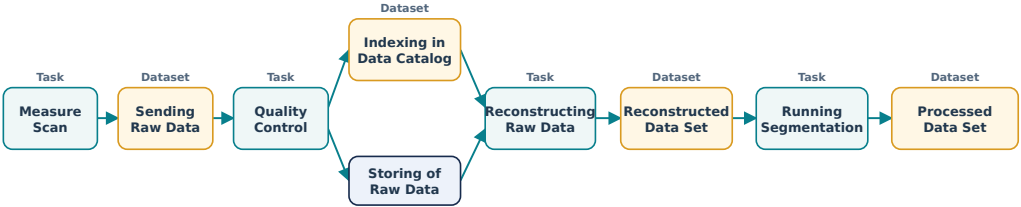
# Thinking within Structured Workflows

- It is useful to organize an HPC experiment within an overarching workflow
- In the first step, data sets and tasks are linked
- Then, these can be further annotated, e.g.
  - Who needs access
  - Semantic Metadata
  - Temporary or permanent result
  - ...
- **What do you think is important as well?**

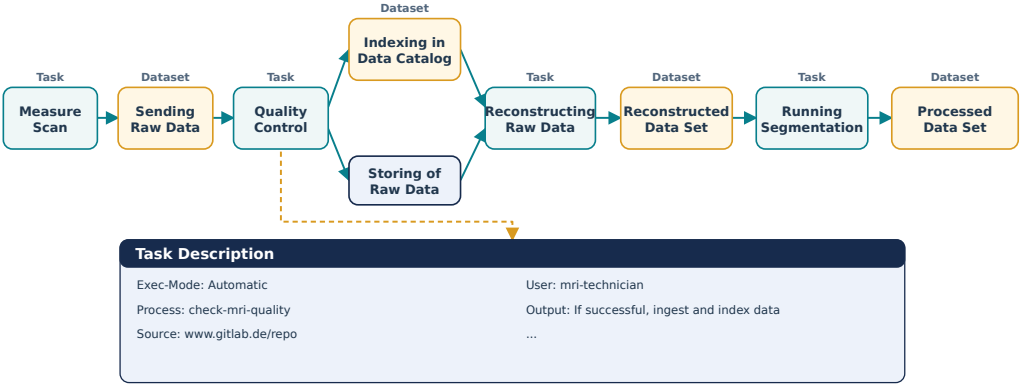
# Thinking within Structured Workflows: Example



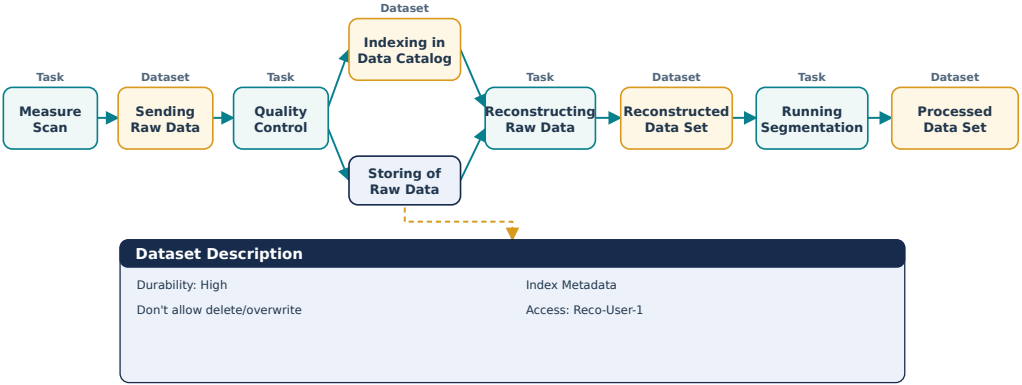
# Experimental Planning Example: MRI Workflow Overview



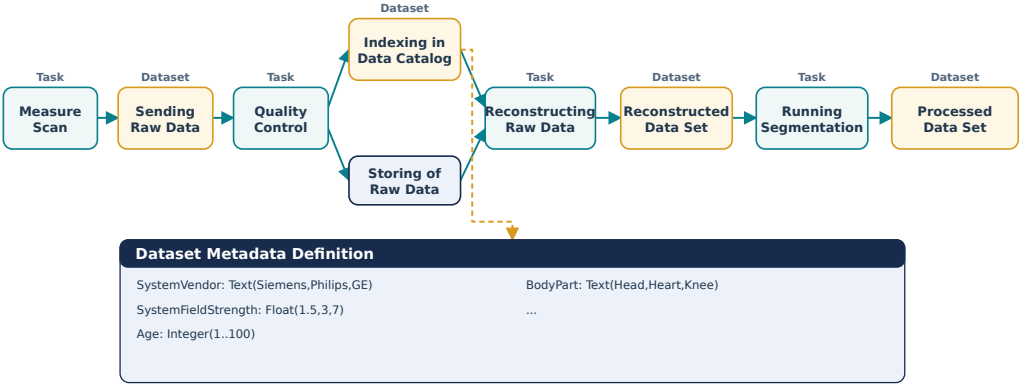
# Experimental Planning Example: Workflow Annotations I



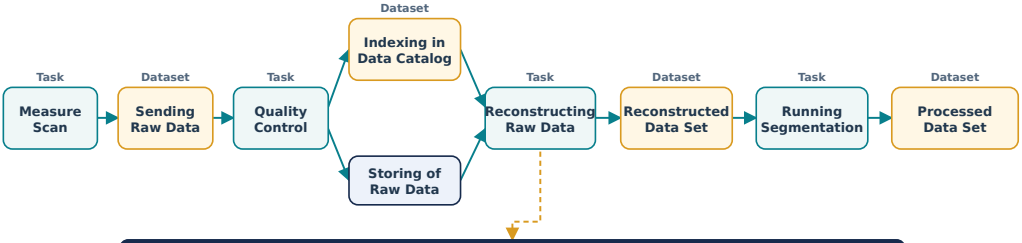
# Experimental Planning Example: Workflow Annotations II



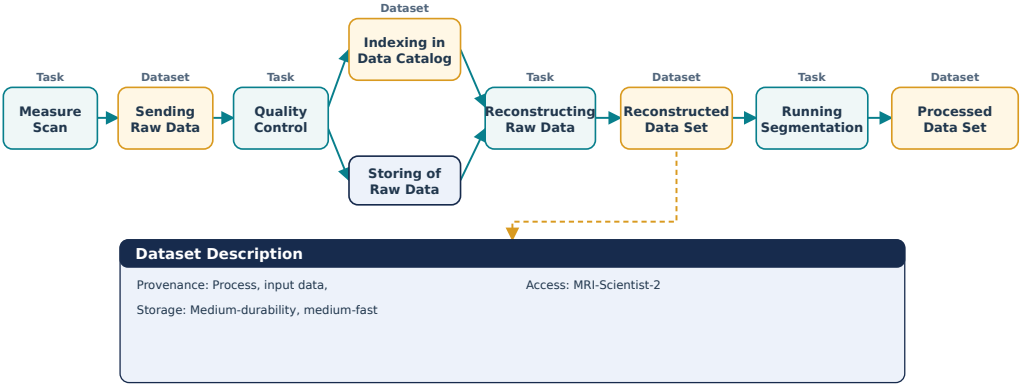
# Experimental Planning Example: Workflow Annotations III



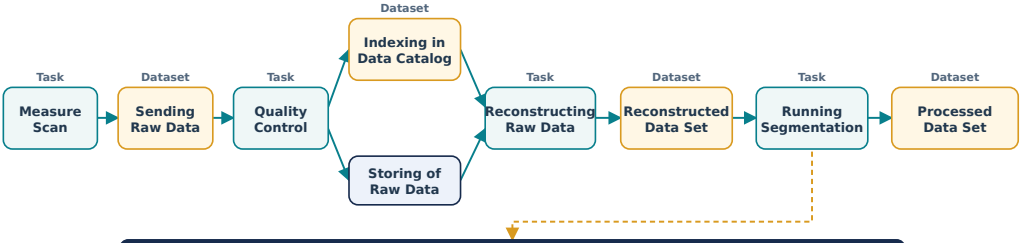
# Experimental Planning Example: Workflow Annotations IV



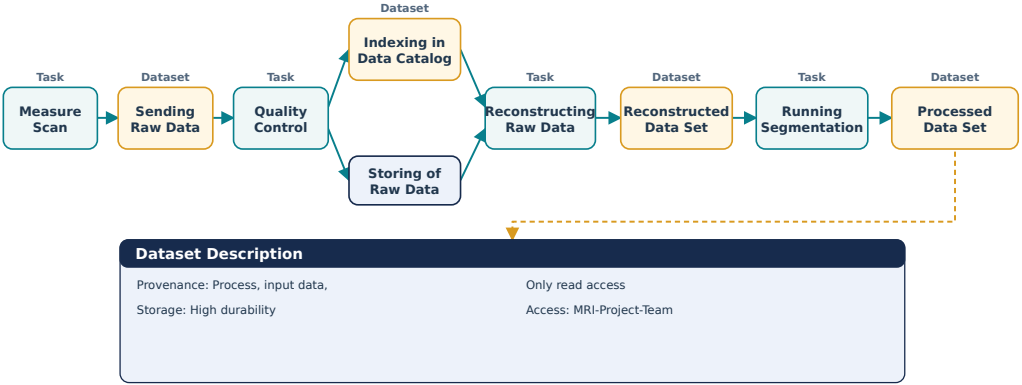
# Experimental Planning Example: Workflow Annotations V



# Experimental Planning Example: Workflow Annotations VI



# Experimental Planning Example: Workflow Annotations VII



# Experimental Planning: Interactive Q&A

- Before coming back to the storage tiering, I have a few questions:
  - Who of you has done something similar in the past?
    - More complicated, or way simplified?
    - What were the differences/similarities?
    - What was your use case?
  - If not, how do you organize your experiments instead?
  - Do you think such a planning is useful?
    - Or just a waste of time?

# Outline

1 Motivation

2 Experimental Planning

3 Conceptual Mapping

4 Metadata

5 Data Staging

6 Permissions

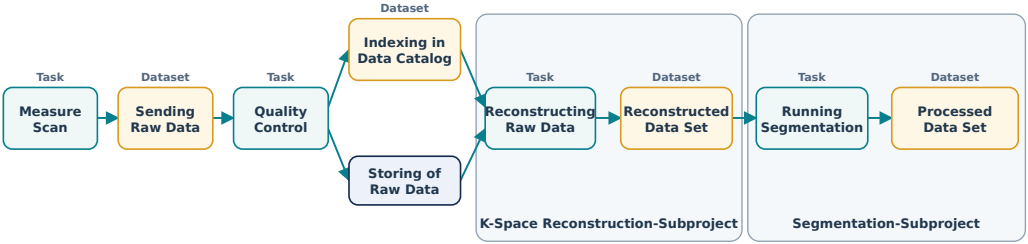
7 Practical

8 Summary

# Experimental Planning Example: Project Decomposition I

- This overarching *MRI-Project* can be further divided into two sub-projects:
  - *K-Space Reconstruction*-Project
  - *Segmentation*-Project
- Advantages:
  - Every (physical/human) user can get a dedicated `uid`
  - Allows fine-granular permission control
  - (FUTURE) Fine-granular Quota enforcement

# Experimental Planning Example: Project Decomposition II



# Mapping to User Roles and File Paths: Step 1

- Now, the high-level experimental plan gets mapped on user roles and file paths
- User Roles:
  - MRI-Technician as a **Data Steward** to upload curated data
- File Paths:
  - Upload to `/projects/mri-project/raw/`
    - **rw** access for user *mri-technician*, **r** access for user *mri-reco-scientist*
    - located on durable, but slow cold/warm storage

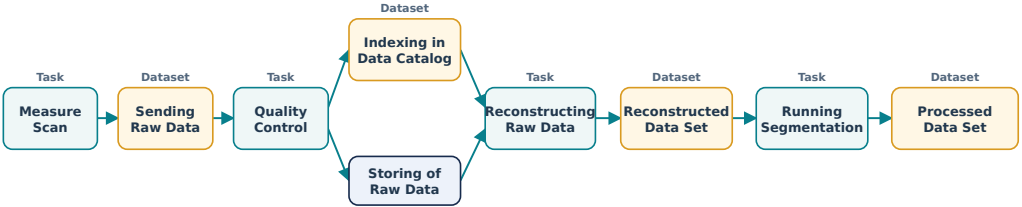
# Mapping to User Roles and File Paths: Step 2

- User Roles:
  - MRI-Reco-Scientist can read raw data from the *MRI-Project* (located at `/projects/mri-project/`) as a **ReadOnlyMember**
  - MRI-Reco-Scientist can write data to their sub-project *K-Space-Reconstruction-Subproject* (located at `/projects/k-space-reco-project/`) as **PI**
- File Paths:
  - Stage data from `/projects/mri-project/raw/` to `/projects/k-space-reco-project/input-data/`
    - **rw** access for user *mri-reco-scientist*
    - fast storage for GPU, input data has a limited, preconfigured lifetime, e.g., 6 months
    - Completely different filesystem, but same semantic

# Mapping to User Roles and File Paths: Step 3

- User Roles:
  - MRI-Reco-Scientist can run arbitrarily their experiment on the *K-Space-Reconstruction-Subproject* (located at `/projects/k-space-reco-project/`) as **PI**
- File Paths:
  - Work on `/projects/k-space-reco-project/`

# Reminder: The total overview



# Mapping to User Roles and File Paths: Step 4

- User Roles:
  - MRI-Reco-Scientist puts their final artifacts into a staging area to share with *K-Space-Reconstruction-Subproject* (located at `/projects/k-space-reco-project/share`) as **PI**
  - MRI-Scientist-2 has read access to that staging area as **ReadOnlyMember**
  - A **Data Steward** can copy the staged data into the main *MRI-Project*
- File Paths:
  - Stage data from `/projects/k-space-reco-project/share`
    - **r** access for user *mri-scientist-2*
    - **rw** access for user *mri-reco-scientist*
    - fast storage for GPU, input data has a limited, preconfigured lifetime, e.g., 6 months
  - to `/projects/segmentation-project/input-data/`
    - **rw** access for user *mri-scientist-2*
    - fast storage for CPU, input data has a limited, preconfigured lifetime, e.g., 6 months
    - Different filesystem in a different island/partition

# Mapping to User Roles and File Paths: Step 5

- User Roles:
  - MRI-Scientist-2 can run arbitrarily their experiment on the *Segmentation-Subproject* (located at `/projects/segmentation-project/input-data/`) as **PI**
- File Paths:
  - `/projects/segmentation-project/input-data/`
    - **rw** access for user *mri-scientist-2*
    - fast storage for CPU, input data has a limited, preconfigured lifetime, e.g., 6 months

# Mapping to User Roles and File Paths: Step 6

- User Roles:
  - MRI-Scientist-2 stages their data product within the *Segmentation-Subproject* in a staging area (located at `/projects/segmentation-project/merge/`) as **PI**
  - **Data Steward** of overarching *MRI-Project* curates staged artifacts and merges them back to the main project (e.g. `/projects/mri-project/segmentations/`)
- File Paths:
  - `/projects/segmentation-project/merge/`
    - **rw** access for user *mri-scientist-2*
    - **r** access for **Data Steward** of parent *MRI-Project*
    - fast storage for CPU, input data has a limited, preconfigured lifetime, e.g., 6 months
  - `/projects/mri-project/segmentations/`
    - Only read access for most members
    - slower, but durable storage
    - long lifetime

# Mapping to User Roles and File Paths: Step 7

- User Roles:
  - **Data Steward** prepares staging area of a dataset to be published (e.g., located at `/projects/mri-project/stage-to-data-pool/`)
  - **Data Steward** provides metadata about this dataset for other users in a sidecar file
- File Paths:
  - `/data/mri-dataset/`
    - **r** access for users
    - either for all or only upon invitation

# Quick Recap

- We have created a workflow containing the experimental description
- The nodes of this workflow were annotated, containing information about
  - The users needing access
  - The storage requirement, e.g., fast, durable, ...
  - The processes that should run
  - Data Volume
  - Life Cycle
- Based on this description the overarching project was split into isolated subprojects
- Lastly, the involved users were mapped on roles and specific storage paths

# Interactive Q&A: Advantages of this approach

- What advantages/disadvantages do you see with this approach?

# Advantages of this approach

- A clear target state is defined
  - This allows to later on compare the is state to the ideal target state
- Different users can be isolated within a single overarching project
  - Scoped permissions reduce accidental changes by other users; backups are still necessary
  - Later: Restricted Quota assignment
- A controlled merge into the main project provides a place for quality checks
  - Can be data products, software environments, etc.
- Data Privacy: Not everyone needs to see all of the data

# Interactive Q&A

- Are there any open questions about this approach?
- Do you have a use case, that we could discuss?
  - Otherwise: Lets go through some imaginary here

# Group Work: Breakout Room Session

- Get together in groups and think of another use case
- Define the workflow as a graph
  - What subprojects can be defined?
  - How can the datasets be annotated?
  - How can the tasks be annotated?
- Lets reconvene in **20** minutes and discuss your use cases

# Outline

1 Motivation

2 Experimental Planning

3 Conceptual Mapping

4 Metadata

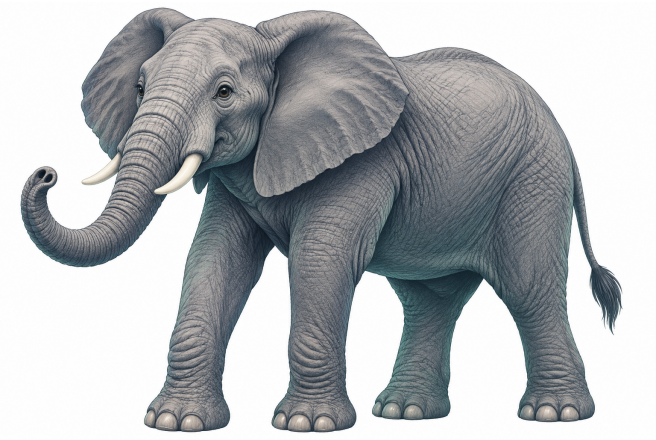
5 Data Staging

6 Permissions

7 Practical

8 Summary

# Talking about the same thing?



# Metadata: Data about data

- **WHO** created the data?
- **WHAT** is the content of the data?
- **WHEN** were the data collected?
- **HOW** were the data created?
- **WHERE** were the data collected?
- **HOW** it may be used
  1. Technical capabilities
  2. Legal restrictions (License)

We need to put data into context!

# What Type of Metadata Vocabularies

- **Controlled Vocabulary:** A carefully selected list of words and phrases, which are used to tag units of information
- **Taxonomies:** A controlled vocabulary with a hierarchical structure
- **Thesauri:** Terms have relationships to each other
- **Ontologies:** Formal descriptions of concepts and their relationships

# Types of Vocabularies: Extending each other

### List of Words

- Control of the terms

### Taxonomy

- Control of the terms
- Hierarchical relations

### Thesaurus

- Control of the terms
- Hierarchical relations
- Equivalence relation
- Association relation

### Ontology

- Control of the terms
- Hierarchical relations
- Equivalence relation
- Association relation
- Formal semantics and constraints can support inference

# Metadata Vocabularies: Dublin Core

Dublin Core Example (XML)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns:dc="http://purl.org/dc/elements/1.1/">
    <dc:title>On the Origin of Species</dc:title>
    <dc:creator>Darwin, Charles</dc:creator>
    <dc:subject>Evolution</dc:subject>
    <dc:description>Long description</dc:description>
    <dc:publisher>John Murray</dc:publisher>
    <dc:date>1859-11-24</dc:date>
    <dc:type>Text</dc:type>
    <dc:format>application/pdf</dc:format>
    <dc:language>en</dc:language>
</entry>
```

# Common Ontologies: Schema.org

Schema.org Example (JSONLD)

```
{
  "@context": "https://schema.org",
  "@type": "Movie",
  "@id": "https://example.com/movies/the-matrix",
  "name": "The Matrix",
  "genre": "Science Fiction",
  "contentRating": "R",
  "aggregateRating": {
    "@type": "AggregateRating",
    "ratingValue": "8.7",
    "bestRating": "10",
    "ratingCount": 1900000
  }
}
```

# FAIR (Meta-)data

## **F**indable

- Unique PID
- Rich Metadata
- Indexed in Searchable Resource
- Metadata specify PID

## **A**ccessible

- Retrivable by ID
- Open, Free, Standardized Protocol
- Metadata is accessible
  - even if data is not anymore

## **I**nteroperable

- Formal, shared, accessible formats
- FAIR vocabulary
- Include qualified references

## **R**e-usable

- Usage License
- Data Provenance
- Follow Community Standarts

# Outline

1 Motivation

2 Experimental Planning

3 Conceptual Mapping

4 Metadata

5 Data Staging

6 Permissions

7 Practical

8 Summary

# Fast Short Term Storage - Introduction

- Motivation: Provide a faster storage for temporary location
- Previously: `/scratch`, but many never deleted their data
  - High-performant storage wasted for cold data
  - Very high utilization $\Rightarrow$ Degraded Performance
- Workflow:
  - Permanent Storage: `$COLD` for NHR, `$PROJECT` otherwise
  - Move data to faster *temporary staging area* to work on
  - Afterwards: Move back to colder storage

## Solutions

- **Job-Specific Storage:** Temporary storage for single job.
- **Workspaces:** Multi-Job Short Term Storage

# Temporary Job-Specific Storage

- Fastest Possible Storage
- Use Case: Read frequently, particularly in random order
- (Un-)staging has to be done **manually**
- Link to docs [here]

## Typical Workflow

0. Allocate (ssd/mem) requirements
1. Copy files onto temp storage
2. Do computation on it
3. At end of job/breakpoint:
   Copy back to `$C0LD/$PR0JECT`

# Temporary Storage: Historical GWDG Comparison

| Kind | Shared | Performance | Capacity | Environmental Variable |
|---|---|---|---|---|
| Shared Memory (RAM) | local | max | tiny | SHM_TMPDIR |
| Local SSD (not all nodes) | local | high | small | LOCAL_TMPDIR |
| SSD SCRATCH/WORK | global | medium-high | medium | SHARED_SSD_TMPDIR |
| HDD SCRATCH/WORK | global | medium | large | SHARED_TMPDIR |

Historical teaching example. See current official documentation before use.

# Workspaces: Historical GWDG Example

- *Temporary* staging area with *expiration date* on *very fast* file systems
  - Example limits: 30 days, extendable to 90 days; actual limits vary by data store
- Focus on Performance over Robustness
  - No Backups, No Snapshots, only RAID
- Workspaces provide temporary access to supported Lustre data stores; availability depends on project and compute island
- Conceptually replaces `/scratch`

# Workspaces: Historical GWDG Storage Options

| Kind of Project | Name | Filesystem | Purpose |
|---|---|---|---|
| SCC, NHR, REACT | Ceph HDD WS (ceph-hdd) | CephFS | Medium-term storage for NHR Test Accounts and for projects that have a temporary demand for more storage space |
| SCC, NHR, REACT | Ceph SSD WS (ceph-ssd) | CephFS | Significantly faster than Ceph HDD, useful for jobs that need storage available from all Cluster Islands |
| NHR | Lustre RZG WS (lustre-rzg) | Lustre | High performance WORK filesystem for users of Grete and Emmy P3 |
| NHR | Lustre MDC WS (lustre-mdc) | Lustre | High performance WORK filesystem for users of Emmy P2 |

Historical teaching example. See current official documentation before use.

# Workspaces: Historical GWDG Example

- Dynamically creatable by users
- Permissions:
  - Everyone can see that the project exists
  - Only the owning group can see the contents
  - Sharing $\Rightarrow$ Using common group as owner
- Example expiration policy (check current data-store limits):
  - Default Time: 1 day. Maximum Time: 30 days (usually)
  - Extensions: 2·30 days (usually)
  - Afterwards: Grace Period to recover (30 days)
    - **Grace Period still counts for your quota!**
- Next: Data ingest and usage of workspaces

# Hands-On Session: Data Ingest

## rsync

- Works via ssh and is terminal-based (For all Linux users)
- Copy data from your system to the HPC:
  `rsync -avH --progress <source>/ <user>@glogin.hpc.gwdg.de:~/.project/<target>`
- A trailing slash on `<source>/` copies its contents; without it, rsync copies the directory by name

## Owncloud

- Go to `owncloud.gwdg.de` and create a folder
- Click on `...`, via **Details** to **Sharing** and create a **Public Link** and **Set a Password!**
- Extract the folder ID from the public link:
  `https://owncloud.gwdg.de/index.php/s/<folder_id>`
- On the HPC system within the terminal run for downloading:

```
rclone copy --webdav-url=https://owncloud.gwdg.de/public.php/webdav
--webdav-user="<folder_id>" --webdav-pass="$(rclone obscure
'<folder_password>')" --webdav-vendor=owncloud :webdav:  <local_dir>
```

## See current GWDG data-transfer documentation

# Hands-On Session: Data Ingest

- What are the differences between the two Ingestion methods?
- In which scenario would you use each of these?

# Hands On Session: Data Ingest Part 2 (Workspaces)

- Based on `https://github.com/holgerBerger/hpc-workspace`
- Available data stores depend on your account, project and compute island
  - Check `ws_list -lg` and the current documentation before the exercise
- Each command can be extended with `-h` for help
- Also, each command has a `man` page
- Docs: `https://docs.hpc.gwdg.de/how_to_use/storage_systems/workspaces/`

# Hands On Session: Data Ingest Part 2 (Workspaces)

- See your allowed data stores with `ws_list -lg`
- Allocate: `WS_PATH=$(ws_allocate -F ceph-hdd -g DMTest 1)`
  - Path goes to `stdout`, logging goes to `stderr`
- See your reservation: `ws_list`
- Set remaining lifetime to 10 days: `ws_extend -F ceph-hdd DMTest 10`
- Remove: `ws_release -F ceph-hdd --delete-data DMTest`
  - `--delete-data` to skip grace period

# Outline

1 Motivation

2 Experimental Planning

3 Conceptual Mapping

4 Metadata

5 Data Staging

6 Permissions

7 Practical

8 Summary

# Extra Information: UNIX Permissions

- You have developed your own workflow before
  - Let's implement it!
- A key aspect of achieving the envisioned user isolation are UNIX permission:

```
glogin1:   $ ls -lah $PROJECT
drwx------ 5 u19997 GWDG 4,0K 2.  Jun 2022 io500-isc
drwxr-x--- 2 u19997 GWDG 4,0K 19.  Jun 2023 data
-rw-r----- 1 u19997 GWDG 9,8G 19.  Jun 2023 file_1
```

- The leading `d` indicates a directory, the `-` that it is a file

# UNIX Permissions: Basics

- The next 9 positions are made up by 3 tuples:
  - **r:** read access
  - **w:** write access
  - **x:** execute permission
- The first tuple is for the user (owner): **u**
- The second for the group: **g**
- The last for others: **o**
- On files this is ”straight forward”
- On directories it means:
  - **r:** list files within a directory
  - **w:** create, remove or rename entries (also requires directory **x**)
  - **x:** cd or traverse into that directory

# UNIX Permissions: Interactive Basics Quiz

- Let’s discuss what the following permissions allow:

**1:** `drwxrwxr-x 2 lquenti lquenti folder1`
**2:** `drwx--xr-- 2 lquenti lquenti folder2`
**3:** `d--xrwx-w- 2 lquenti lquenti folder3`
**4:** `-rw-rw-r-- 1 lquenti lquenti file1`
**5:** `-r--r----x 1 lquenti lquenti file2`
**6:** `-r--r--rwx 1 lquenti lquenti file3`
**7:** `drwxrwxr-x 2 lquenti lquenti file4`

# UNIX Permissions: S-Bit

- SGID and sticky bits add directory-specific behavior
- SGID Bit on a directory
  - New files inherit the directory's **group**; new subdirectories also inherit SGID
  - Special Bit **s** on **g** (group)
  - If set, an **s** is where the **x** is usually located (for the group)
    - If group execute/search is not set, an uppercase **S** is displayed
- Sticky Bit on directory
  - Removal or rename is restricted to the file owner, directory owner or a privileged process; other permissions still apply
  - Special Bit **t** on **others**
  - If set, an **t** is where the x is usually located (for others)
    - If others' execute/search is not set, an uppercase **T** is displayed

# UNIX Permissions: Interactive S-Bit Quiz

- Let’s discuss what the following permissions enforce:

**1:** `drwxrwsr-x 2 lquenti lquenti folder1`

**2:** `drwxrwxr-T 2 lquenti lquenti folder2`

**3:** `drwxrwSr-x 2 lquenti lquenti folder3`

**4:** `drwxrwSr-t 2 lquenti lquenti folder4`

# Setting/Changing UNIX Permissions

- `chmod`: Command for changing permission
  - Example: `chmod u+rw TARGET` (add owner read/write)
- Few examples:
  - `chmod a+r test.txt` Gives everyone read permission
  - `chmod g= test.txt` Clears the group read/write/execute bits
  - `chmod u+x test` Allows execution of test
  - `chmod u+X test-dir` Adds owner search on directories, or execute on files already executable for someone
  - `chmod -R g+rwX test-dir` Recursively adds group read/write, directory search, and execute on already-executable files
  - `chmod +t test-dir` Sets sticky bit (shown as **t** or **T**, depending on others' search bit)
  - `chmod g+s test` Add SGID bit

# Optional: Setting/Changing UNIX Permissions Numerically

- Question: From a programmers perspective, now many bits do you need to store the permission information?
  - i.e. `rwxrw----`

# Optional: Setting/Changing UNIX Permissions Numerically

- Question: From a programmers perspective, now many bits do you need to store the permission information?
  - i.e. `rwxrw----`
- The basic user/group/other `rwx` permissions use 9 bits; special bits are separate
  - But **NOT** `chmod 111110000 file`
- Thus, for each type (user/group/other): 3 bit.
- Binary representation:
  - First (`read`) bit: $(100)_2 = 4$
  - Second (`write`) bit: $(010)_2 = 2$
  - Third (`execute`) bit: $(001)_2 = 1$

# Optional: Setting/Changing UNIX Permissions Numerically

- `rwx rwx rwx` ⇒ `4+2+1 4+2+1 4+2+1` ⇒ `777`
- `rwx rw- r--` ⇒ `4+2+1 4+2 4` ⇒ `764`
- `rw- r-x ---` ⇒ `4+2 4+1 0` ⇒ `650`

# UNIX Permissions: Ownership

- Every file and directory is owned by a user and a group
- Very important concept since the previous permissions are using the ownership
- Usage:
  - `chown NEW_OWNER TARGET` Change the ownership of target
    - Changing the file's owner requires privilege
    - `chown lquenti.lquenti file` (dangerous, as dots are allowed in usernames!)
    - `chown lquenti:lquenti file` (recommended)
  - `chgrp NEW_GROUP TARGET` Change the group of target
    - File owners may select a group they belong to; arbitrary changes require privilege
    - `chgrp cdrom file`
  - `whoami` Show own username

# Group Work: Overview Data Management

- List all aspects discussed today for efficient data management
- State their individual contributions
- Describe how you would proceed to implement a real use case
- Lets compare in **7 minutes**

# Group Work: Overview Data Management

- Key Aspects:
  - Write down your workflow as a graph
    - Differentiate between tasks and data sets
  - Annotate each node (Or even edge?)
    - Defining semantic metadata
    - Access Control Lists
    - ...
  - Setup project structure accordingly
  - Check that the definition and implementation match
    - Integrate quality control where defined
- Individual Contributions:
  - Comprehensibility
  - User isolation
  - Correct Data Placement (Findability, Performance, Prevent Data Loss)
  - Quality Control

# Outline

1 Motivation

2 Experimental Planning

3 Conceptual Mapping

4 Metadata

5 Data Staging

6 Permissions

7 Practical

8 Summary

# Recap

- Use tiered storage systems employed by HPC systems
  - to optimize performance and prevent data loss
- Organize your HPC workloads within workflows
  - Implement user isolation and data sharing in projects using UNIX permissions
- Design your own HPC workflows
  - Describe independent tasks and data sets with standardized annotations
  - Classify these steps and data sets using annotations
  - Identify the correct storage tier/system based on these annotations
- Identify scenarios where a data catalog is useful
- Recognize opportunities to ask for dedicated data management support
  - E.g. via our Ticket system

# Outlook

- This was the introductory course
- Further topics to explore:
  - Data catalogs
  - Data pools
- **Do you have any wishes?**

# GöDL Data Lake: Data Catalog

- A data catalog for HPC systems
  - Allows data selection based on semantic metadata
  - Transparent integration of various storage tiers
- High user-friendliness through semantic access
  - Data can be staged to fast storage before training
  - Data migration between different storage systems
  - Artifacts are monitored, can support reproducibility

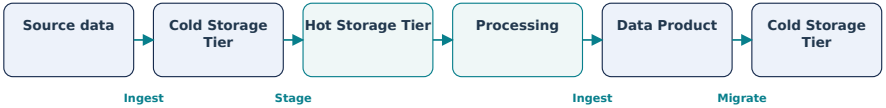
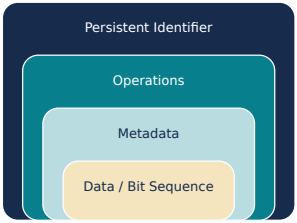



Figure: Data Lifecycle and related GöDL Operations

# Data Lake: Data Modeling

- Model generic data as FAIR digital objects
  - Can virtually represent real entities
  - Simple, non-technical communication with customers
- Data has a unique type
  - e.g. MRI scan, EHM or measurement result XY
- Key-value pairs define semantic metadata
  - Allows data selection based on properties
  - Customers describe the data they want → and don't have to remember complicated paths
- Interface to highly scalable computing capacity



# GöDL: Technical Implementation

- **HPC-native Implementation**
  - **Application agnostic:** No Hadoop/Spark required
  - **File format agnostic:** Using JSON sidecars
  - **Staging is a simple copy:** All nodes have same mounts
  - **No file system abstractions:** Parallel File System Performance
- **Key operations:**
  - **Ingest**: Add datasets with (JSON-formatted) custom metadata
  - **List:** Search datasets by metadata criteria
  - **Stage:** Dynamically load data into local environment based on metadata

**In a Nutshell:** Instead of many

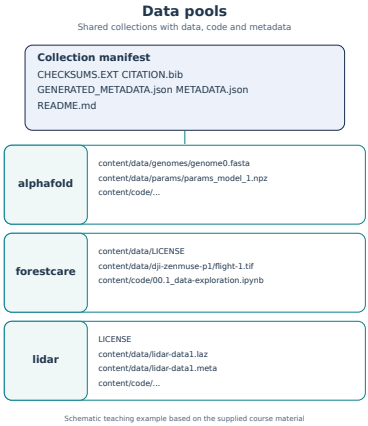
```
cp /projects/mrt-1/dataset5/hum_3242/scan_1.nii /fast/user/dataset5/
```

Only one

```
goedl stage Type=MRI,Protocol=T1,Gender=Male --target /fast/user/dataset5
```

# Data Sharing: Data Pools

- Datasets relevant to **multiple project** over a **longer period** of time
- Access options: *Public* or *On-Request*
  - *Public*: Accessible to all cluster users
  - *On-Request*: Access granted after request
- Enables inter-project data sharing
- Features:
  - Fast storage, close to Cluster
  - Machine-readable metadata
  - Semi-automatic validation
- Example datasets:
  - Pre-trained machine learning models
  - Public datasets used by multiple projects



# Feedback

- In order to improve our next iteration we require your help!
- Please give us your feedback:
  - Share questions and suggestions by email
  - Via mail: `s.mahmoodian@gwdg.de`