Data storage, transfer and sharing

damien.francois@uclouvain.be

november 2024

Goal of this session:

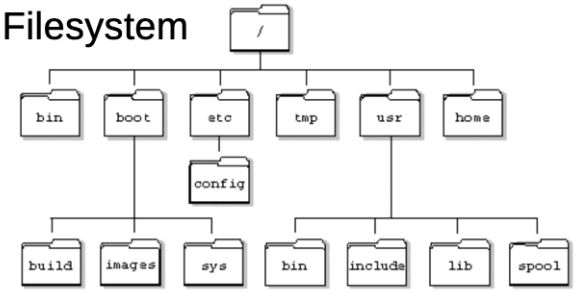
> Share tools, tips and tricks related to the **storage**, **transfer**, and **sharing** of scientific data on the clusters.

Data storage

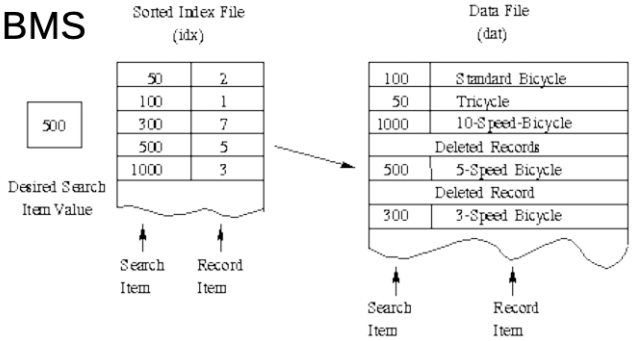
- File
  - Filesystems
  - File formats
  - Common problems with files
- Object storage
- Database systems

Data storage paradigms

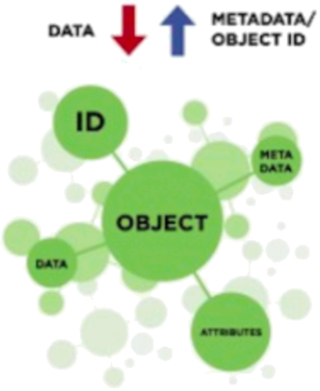
Filesystem



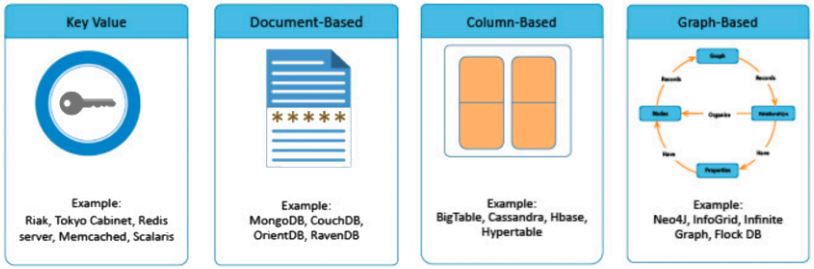
RDBMS



Objects store



NoSQL



Filesystems

> Technology (method and data structure) used by the operating system to store and retrieve files.

Can be

- *local* on disk or in RAM, viewed only from one server, or
- *shared* through the network, visible from multiple servers.

Shared filesystems

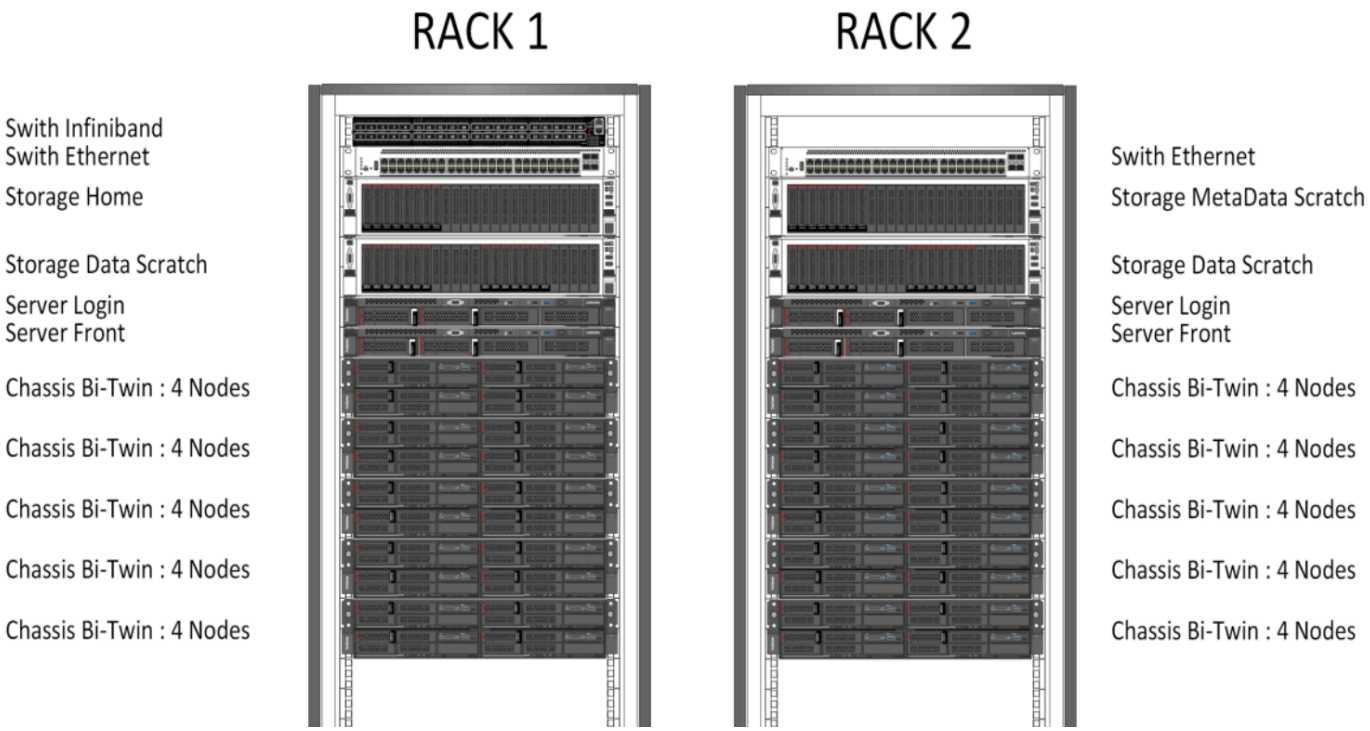
Network filesystems

- one server multiple clients (NFS, CIFS)
- typically used for the `$HOME` directories

Parallel filesystems

- multiple servers multiple clients (BeeGFS, GPFS, Lustre)
- typically used for the global scratch `$GLOBALSCRATCH`

Lemaitre4


RACK 1
RACK 2
Swith Infiniband
Swith Ethernet
Storage Home
Storage Data Scratch
Server Login
Server Front
Chassis Bi-Twin : 4 Nodes
Chassis Bi-Twin : 4 Nodes
Chassis Bi-Twin : 4 Nodes
Chassis Bi-Twin : 4 Nodes
Chassis Bi-Twin : 4 Nodes
Swith Ethernet
Storage MetaData Scratch
Storage Data Scratch
Server Login
Server Front
Chassis Bi-Twin : 4 Nodes
Chassis Bi-Twin : 4 Nodes
Chassis Bi-Twin : 4 Nodes
Chassis Bi-Twin : 4 Nodes
Chassis Bi-Twin : 4 Nodes

Lemaitre4 filesystems

```
[dfr@lm4-f001 ~]$ df -khT -x tmpfs
Filesystem                                                Type      Size  Used Avail Use% Mounted on
/dev/sda3                                                 xfs        48G  6.3G   42G  14% /
/dev/sda2                                                 xfs      1006M  202M  805M  21% /boot
/dev/sda1                                                 vfat      599M  5.8M  594M   1% /boot/efi
/dev/sda4                                                 xfs        16G  6.9G  9.2G  44% /tmp
/dev/sda6                                                 xfs       378G  2.7G  375G   1% /localscratch
gw-ucl:/CECI/gateway/proj                                 nfs        32T   24T  8.1T  75% /CECI/proj
lm4-n001-ib:/soft/localsoft/RedHat-8_25-17-1_Infiniband nfs       1.3T  664G  617G  52% /opt/sw/arch
beegfs_nodev                                              beegfs    318T  267T   51T  84% /globalscratch
10.44.3.1:/home                                           nfs4       22T  3.3T   19T  16% /home
```

Source:

- `/dev/sd...` → local disk
- `<machine>:<path>` → NFS
- other (e.g. `beegfs_nodev`) → specific filesystem

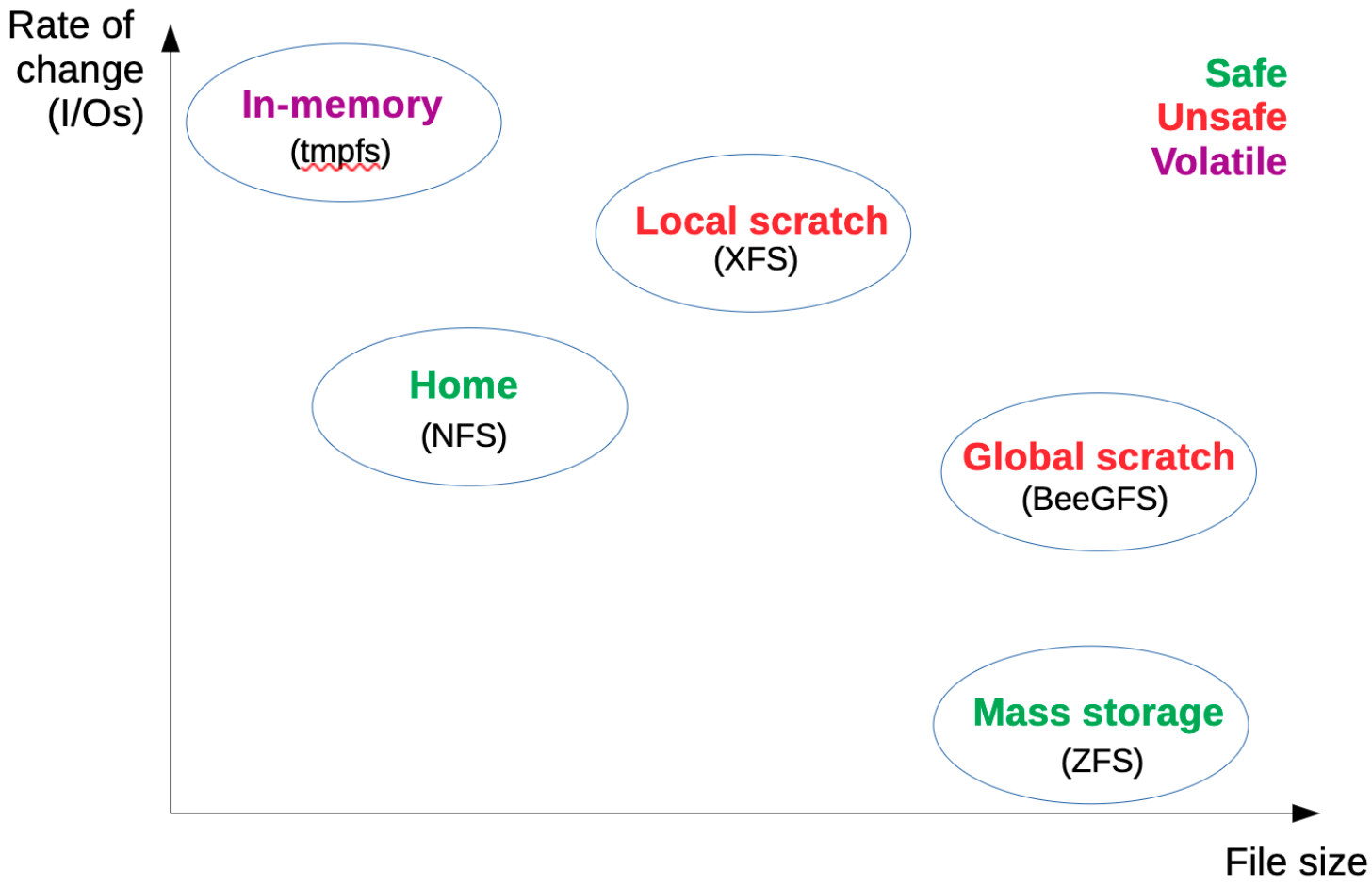
Lemaitre4 filesystems

```
[dfr@lm4-f001 ~]$ df -khT -x tmpfs -i
Filesystem                                                Type     Inodes IUsed IFree IUse% Mounted on
/dev/sda3                                                 xfs         24M  208K   24M    1% /
/dev/sda2                                                 xfs        512K    19  512K    1% /boot
/dev/sda1                                                 vfat          0     0     0     - /boot/efi
/dev/sda4                                                 xfs        8.0M   20K  8.0M    1% /tmp
/dev/sda6                                                 xfs        189M    12  189M    1% /localscratch
gw-ucl:/CECI/gateway/proj                                 nfs        108M   39M   69M   37% /CECI/proj
lm4-n001-ib:/soft/localsoft/RedHat-8_25-17-1_Infiniband nfs        128M  3.3M  125M    3% /opt/sw/arch
beegfs_nodev                                              beegfs        0     0     0     - /globalscratch
10.44.3.1:/home                                           nfs4       37G   28M   37G    1% /home
```

INodes:

- entries in the file datastructure
- rougly proportional to the number of files (and their size)
- **often disregarded but more important than volume!**

What filesystem for what usage



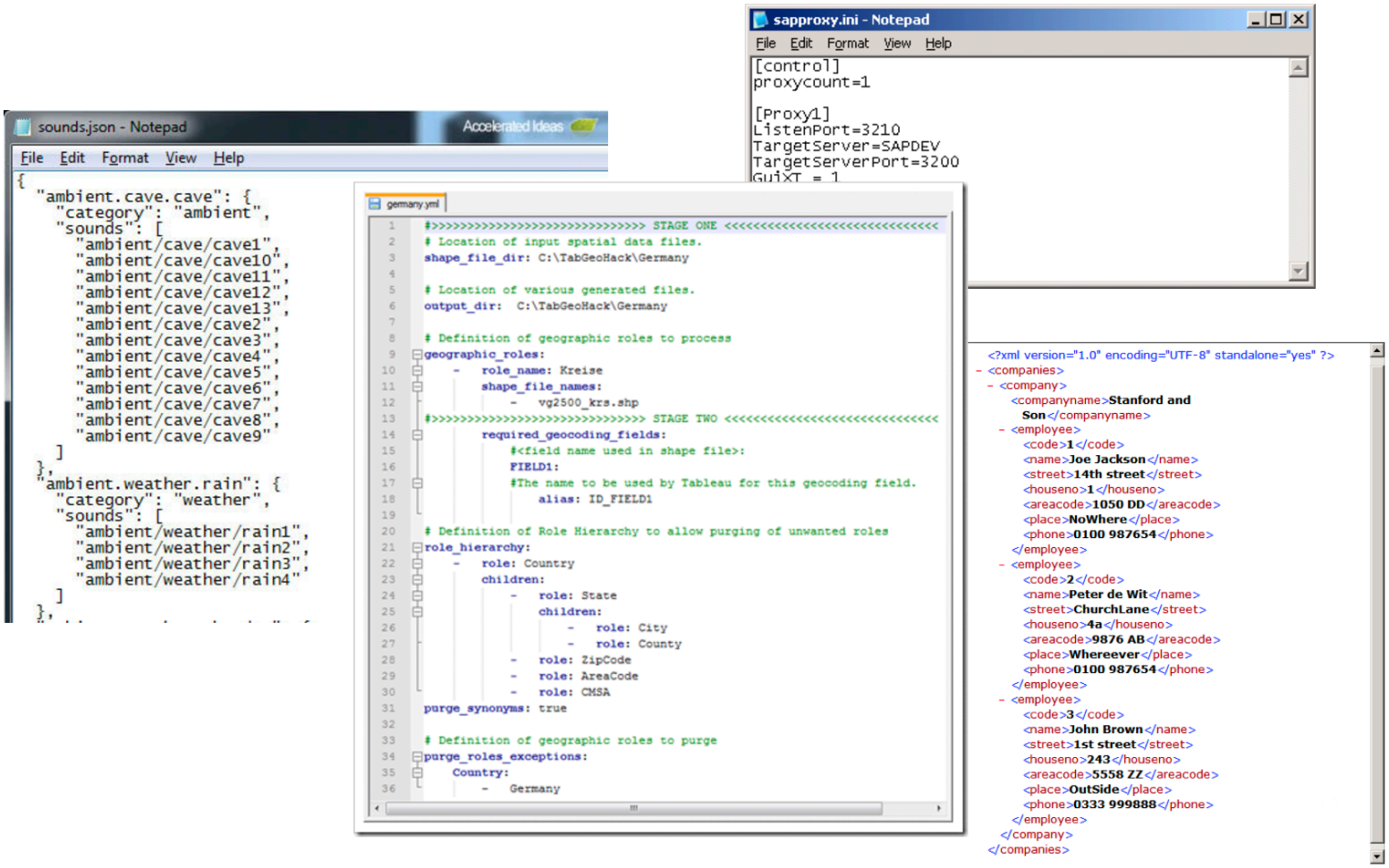
File formats

> Standard way information is organized and encoded in a file

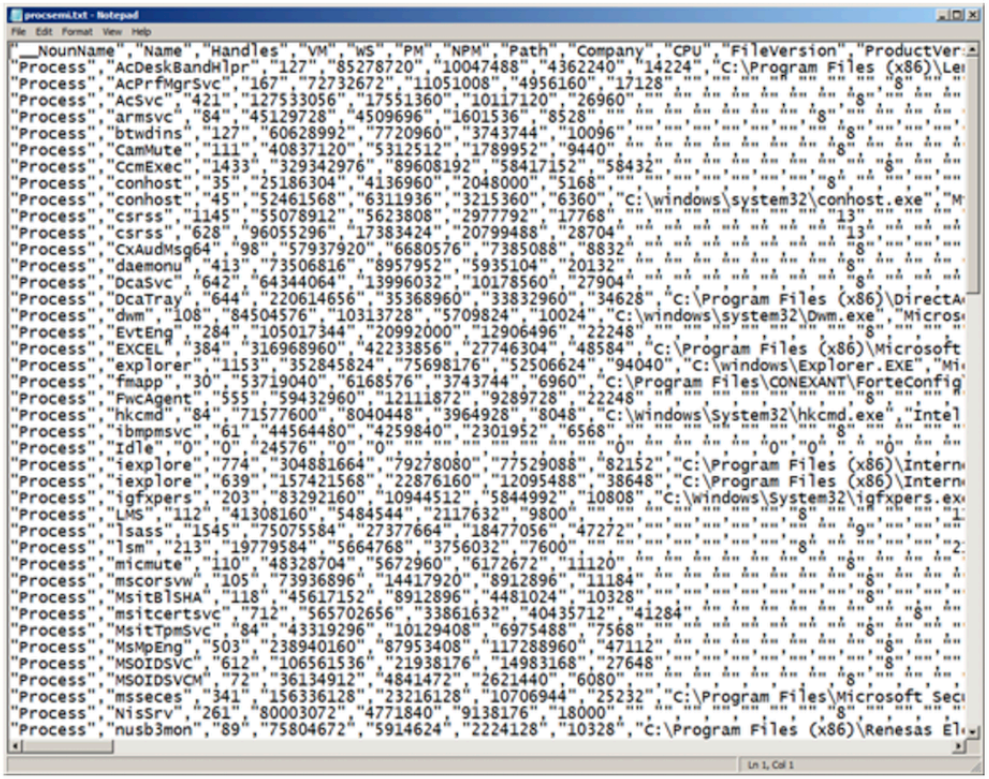
Can be

- *text*, readable by a human, but space-inefficient
- *binary*, readable by dedicated software, often compressed.

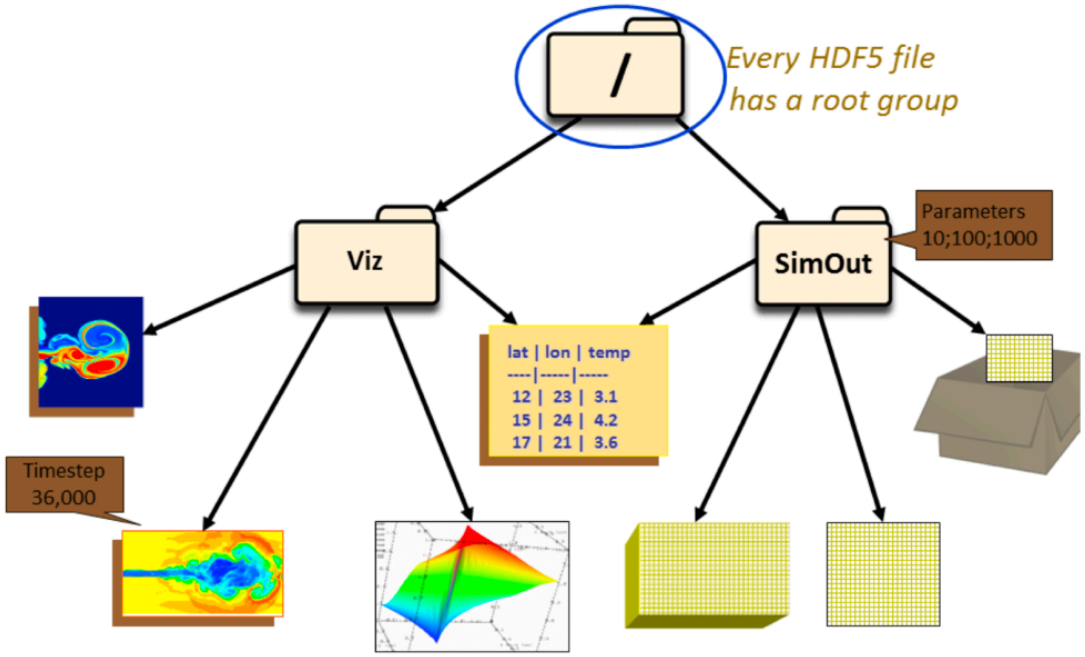
Text file formats



Text file formats



Binary file formats



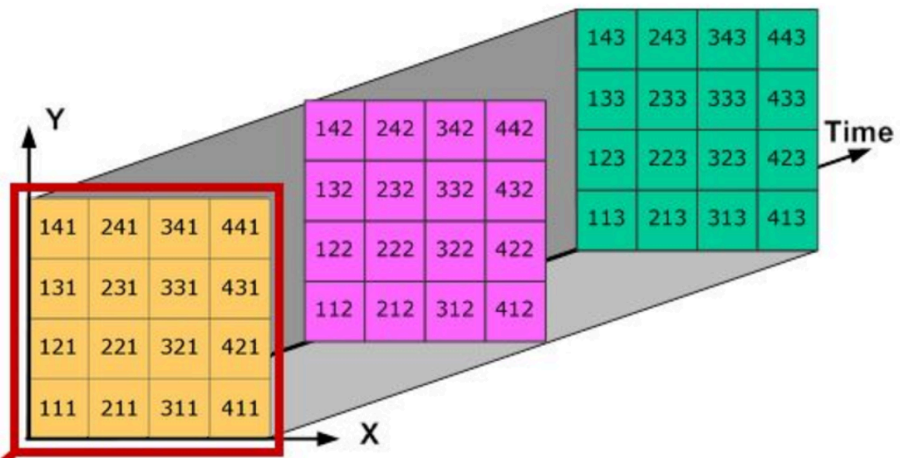
There are two groups in the HDF5 file depicted above: Vis and SimOut. Under the Viz group are a variety of images and a table that is shared with the SimOut group. The SimOut group contains a 3-dimensional array, a 2-dimensional array and a link to a 2-dimensional array in another HDF5 file.

https://support.hdfgroup.org/HDF5/Tutor/HDF5Intro.pdf

Binary file formats

```
netcdf mynetcdf {
dimesions:
        x=4;
        y=4;
        time=UNLIMITED;
variables:
        float x(x);
        float y(y);
        int time(time);
        float temperature(time,x,y);
data:
        x = 10, 20, 30, 40;
        y = 110, 120, 130, 140;
        time = 31, 59, 90;

Temperature = 111, 211, 311, 411, 121, 221, 321,
        421, 131, 231, 331, 431, 141, 241, 341, 441;
}
```



https://adyork.github.io/python-oceanography-lesson/17-Intro-NetCDF/index.html

What file format for what usage

Meta data:

- Configuration file: INI, YAML
- Result with context information: JSON

Data:

- Small data (kBs): CSV
- Medium data (MBs): compressed CSV
- Large data (GBs): netCDF, HDF5, ASDF, Zarr
- Huge data (TBs): Object store
- Huge number of small files (10^6*kBs): Database

The problems with files

Filesystems are not designed

- to host millions of files
- to manage concurrent accesses diligently
- to finely organise sharing of the files

Problem 1: ZOT files

"Zillions of tiny" files

- Over-consume resources
  - Inode spaces is finite on most filesystems
  - Minimal chunk size often large on HPC filesystems
- Makes system slower (`ls` , `rsync` , `rm` etc.)
  - Inodes operations cannot be buffered/cached easily
  - Event simple ls -l can put heavy burden on MD servers slowing all the operations
- Easy to loose control

Problem 1: ZOT files (solutions)

Write a single file:

- TAR archive
- Singularity container

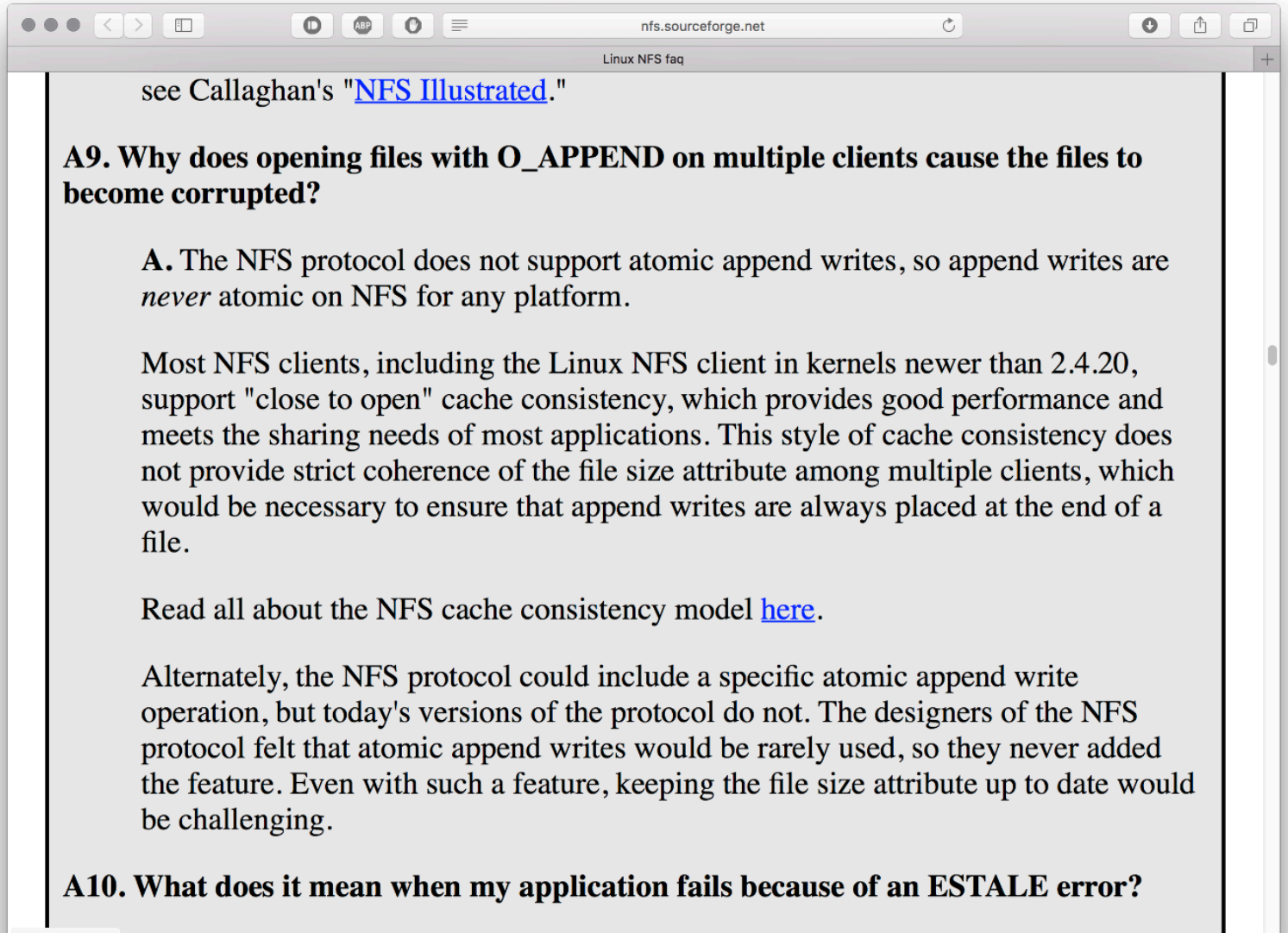
Write them to local filesystems

- Even better to memory filesystems

Even better to another storage type:

- Object store (rich efficient meta data)
- Database (strong structure)

Problem 2: Concurrent access



see Callaghan's "NFS Illustrated."

A9. Why does opening files with O_APPEND on multiple clients cause the files to become corrupted?

A. The NFS protocol does not support atomic append writes, so append writes are *never* atomic on NFS for any platform.

Most NFS clients, including the Linux NFS client in kernels newer than 2.4.20, support "close to open" cache consistency, which provides good performance and meets the sharing needs of most applications. This style of cache consistency does not provide strict coherence of the file size attribute among multiple clients, which would be necessary to ensure that append writes are always placed at the end of a file.

Read all about the NFS cache consistency model here.

Alternately, the NFS protocol could include a specific atomic append write operation, but today's versions of the protocol do not. The designers of the NFS protocol felt that atomic append writes would be rarely used, so they never added the feature. Even with such a feature, keeping the file size attribute up to date would be challenging.

A10. What does it mean when my application fails because of an ESTALE error?

Problem 2: Concurrent access (solutions)

- Use a library for (organised) parallel writes (e.g. netCDF)
- Write to local filesystems and merge afterwards
- Use a database

Problem 3: sharing

- Everyone must have access to the same computer
- UNIX permissions are not so flexible
    - Groups must be set by admins
    - No organization (hierarchy) of groups

Problem 3: sharing (solutions)

- ~~Dumb it down: chmod 777~~
- Abuse UNIX permissions
- Use an Object store

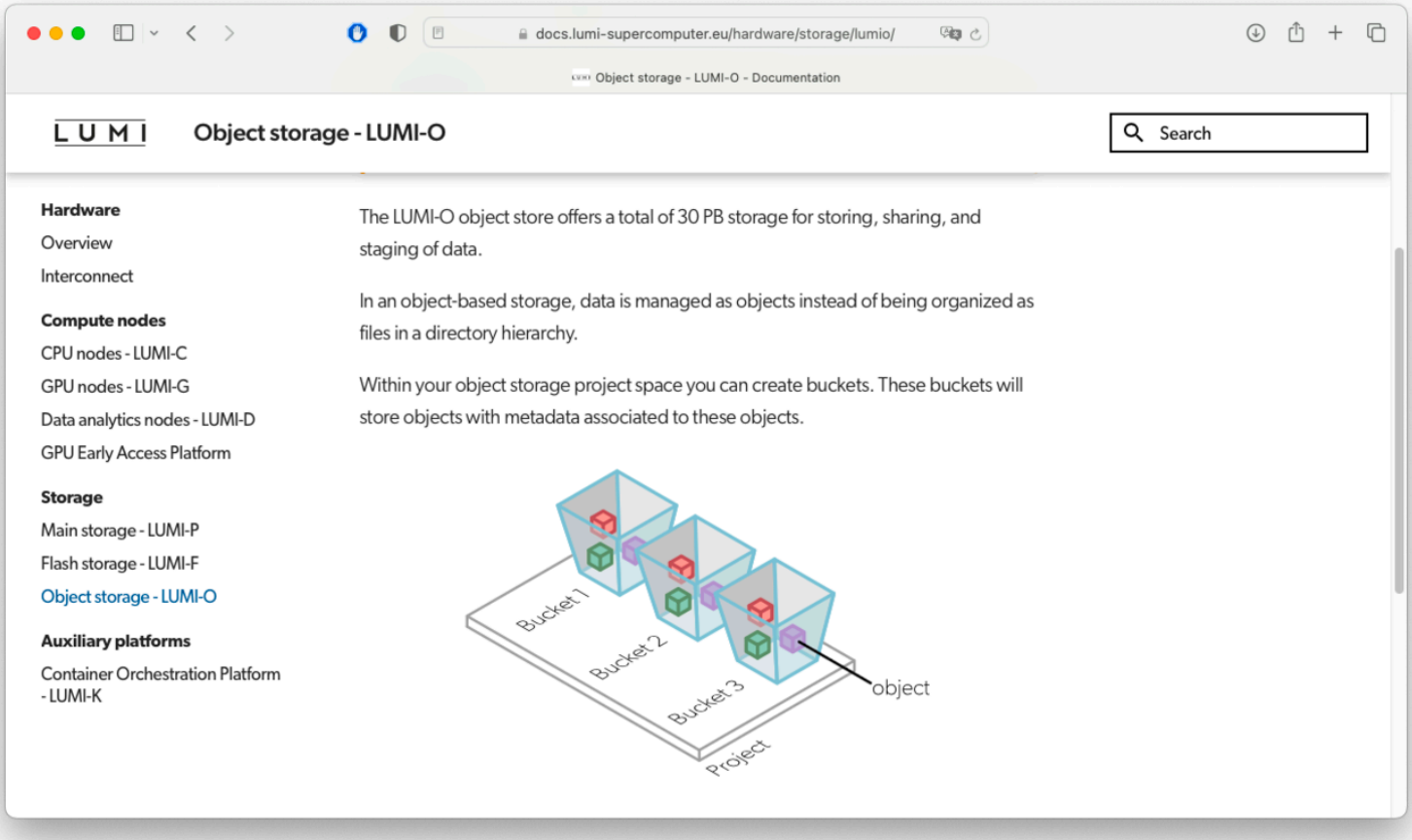
Object store

> data storage technology that manages data as objects that include the data itself, a variable amount of metadata, and a globally unique identifier

Useful for web applications but coming to scientific world

Access with REST API (through HTTP)

Object store on Lumi



Access with code

Access with command line

Setup your own

https://pypi.org/project/boto/ -- https://rclone.org -- https://github.com/minio/minio

Object store

When to use

- For data that is written once and then read multiple times from multiple remote locations as a whole
- Input data for in-memory decompression
- Output files for egress or sharing

When not to use

- When direct access to portions of a file are needed
- When data is not meant to be read sequentially

Database systems

> system that interfaces users, applications, and the database itself to capture and analyze the data

useful for enterprise data but coming to the HPC world

access via query language and software libraries

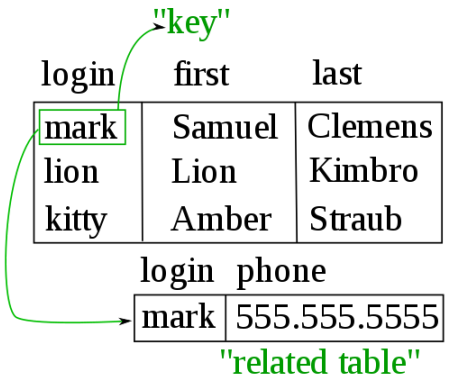
Relational database

Query language : SQL

```
create table Users (login varchar(255), first varchar(255), last varchar(255));

insert into Users values (“mark”, ‘Samuel”, “Clemens”);

select first,last from Users where login=’lion’;
select login, phone from Users join PhoneNb on Users.login=Phone.login;
```



Setup your own

File-based database system, easy way to replace a large collection of small files with a single file.

rqlite

The lightweight, distributed relational database

No-setup server on top of SQlite that can cope with concurrent accesses.

https://www.sqlite.org -- https://rqlite.io

NoSQL


key-value
Amazon DynamoDB (Beta)
ORACLE BERKELEY DB 11g
redis
graph
Neo4j the graph database
InfiniteGraph
sones
column
HBASE
riak
Cassandra
document
CouchDB relax
mongoDB
elastic

Setup your own

File-based document-oriented database system, easy way to replace a large collection of small files with a single file.

No-setup key-value database server that can cope with concurrent accesses.

No-setup document-oriented database server that can cope with concurrent accesses.

https://tinydb.readthedocs.io/en/latest/ -- https://redis.io -- https://zincsearch-docs.zinc.dev

Database

When to use

- when you have a large collection of small files
- when you perform a lot of direct writes in a large file
- when you want to keep structure/relations between data

Many small results

When not to use

- only sequential access
- simple matrices/vectors, etc.
- direct access on fixed-size records and no structure

Redis example

Scirpt `redis-server.sh`:

```
#! /bin/bash
#SBATCH -n1 --mem 4G
module load redis
hostname -s > $HOME/redisserver
redis-server
```

Script `work.sh`:

```
#! /bin/bash
#SBATCH -t 10:00 -n 1 -c 4 --mem-per-cpu 4G
#SBATCH --array 1-1000
module load redis

./myprog | redis-cli -h <(<$HOME/redisserver) -x SET res-$SLURM_TASK_ARRAY_ID
```

Submission:

```
J=$(sbatch --parsable redis-server.sh)
sbatch --dependency=after:$J work.sh
```

Data transfer

- SCP
- RSYNC
- TAR
- Parallel RSYNC

SCP

Simplest (and least efficient) way to copy a file to/from a remote server:

```
scp somefile lemaitre4:destination/directory
scp lemaitre4:destination/directory/somefile .
```

Copy remote to remote:

```
scp lemaitre4:some/directory/somefile destination:
scp -3 lemaitre4:some/directory/somefile destination:
```

Use `-3` if `source` cannot reach `destination` directly

Use `-r` ("recursive") for directories

Use `-O` (Capital `O`) for OpenSSH version 9+

Note: to transfer from one CÉCI cluster to another, use the common storage

RSYNC

Efficient way to synchronise directories to/from a remote server:

```
rsync -va directory lemaitre4:some/directory
rsync -va lemaitre4:some/directory .
```

Rsync will only transfer the parts of the files that changed.

Can be used to resume an interrupted `SCP` transfer:

```
scp somelargefile lemaitre4:destination/directory # Interrupted for some reason
rsync --append somelargefile lemaitre4:destination/directory
```

RSYNC

Progress monitoring: use

- `-v --progress` for large files
- `--info=progress2 --no-i-r` for many smaller files

Verify what will be transfered before transfering with

- `--dry-run`

Choose files to transfer with

- `--exclude`
- `--include`

Change group on the fly with

- `-g --groupmap=*:ceci_group`

TAR+SSH

Often, for ZOT files, creating a single large file and transfering that file will be more efficient.

```
tar czf - /path/to/data | ssh server “tar xzf - -C destination/directory"
```

This will compress and uncompress data on the fly.

Parallel RSYNC

http://www.fpart.org/fpsync/

```
[dfr@lm4-f001 Data]$ time scp -qr linux-6.9.8 manneback:

real    26m7.000s
user    0m3.856s
sys     0m10.972s
```

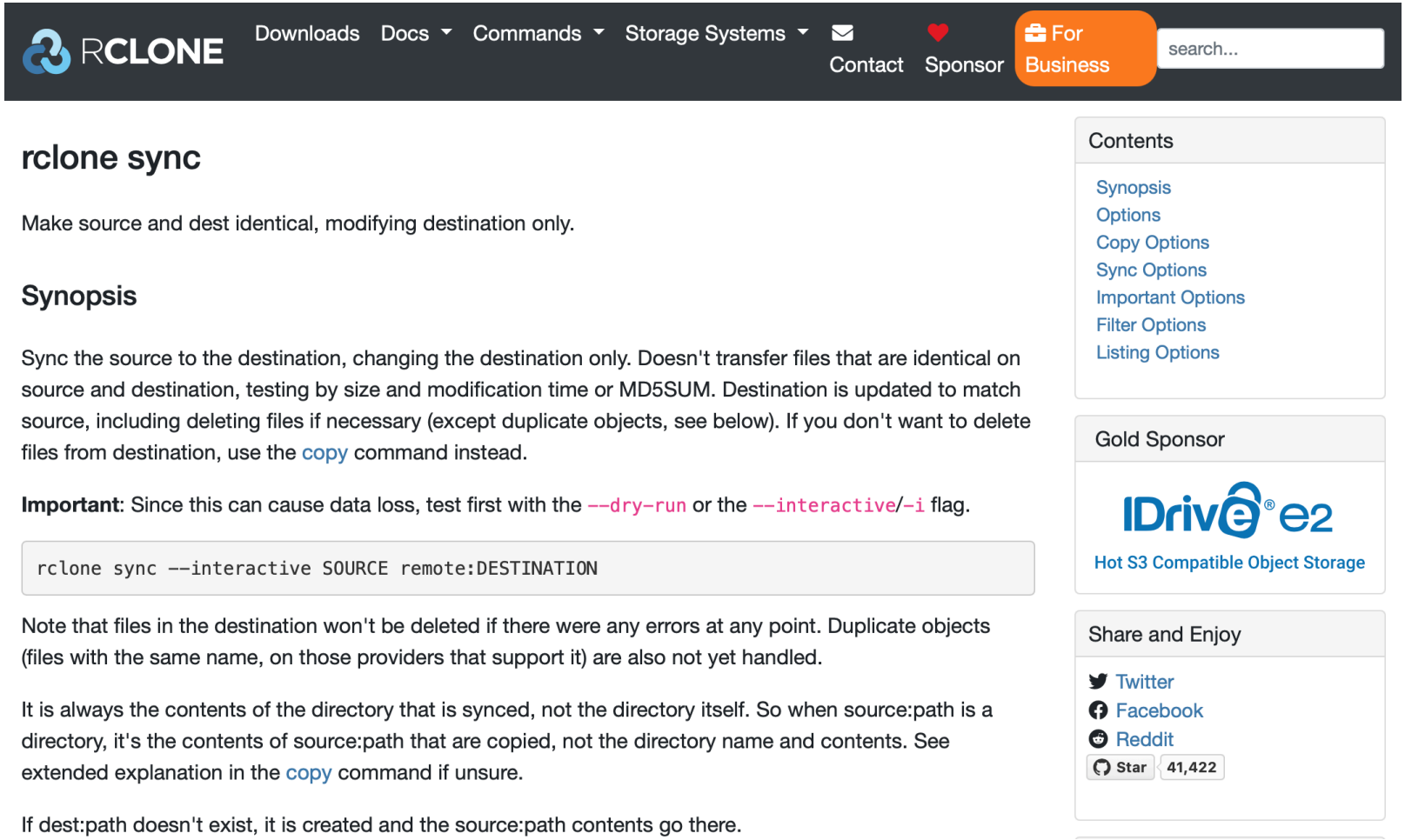
```
[dfr@lm4-f001 Data]$ time { tar czf - linux-6.9.8 | ssh manneback tar xzf -  ; }

real    16m56.519s
user    0m33.286s
sys     0m4.805s
```

```
[dfr@lm4-f001 Data]$ time fpsync $HOME/Data/linux-6.9.8 manneback:$HOME/Data/linux-6.9.8

real    11m51.561s
user    0m52.537s
sys     3m16.098s
```

Parallel RSYNC



https://rclone.org/commands/rclone_sync/

Data sharing

- Personal/Sensitive data
- UNIX Permissions
- Encryption
- Nextcloud
- Dataverse

Personal/Sensitive data

The clusters are designed for performance by default, not privacy

About the responsibilities:

what	who
describing what specific protection measures to take	the project PI, or the institution's DPO
implementing protection measures	the user
making sure the infratructure is safe and secure from external threats	the sysadmins

Personal/Sensitive data

Four possible recommendations for personal and/or sensitive data:

- work only on local, mono-user, computer
- encrypt the data
- anonymize the data
- pseudonimize the data

It all depends on the data and the project.

Encryption

- in *transit* -- this is always the case on clusters with SSH
- at *rest* on disks (when not processed by a job) -- it is the user's responsibility to do so, and system administrators can help set up what is needed
- at *work* in RAM (for the duration of the job) -- this is almost impossible to ensure on clusters ;

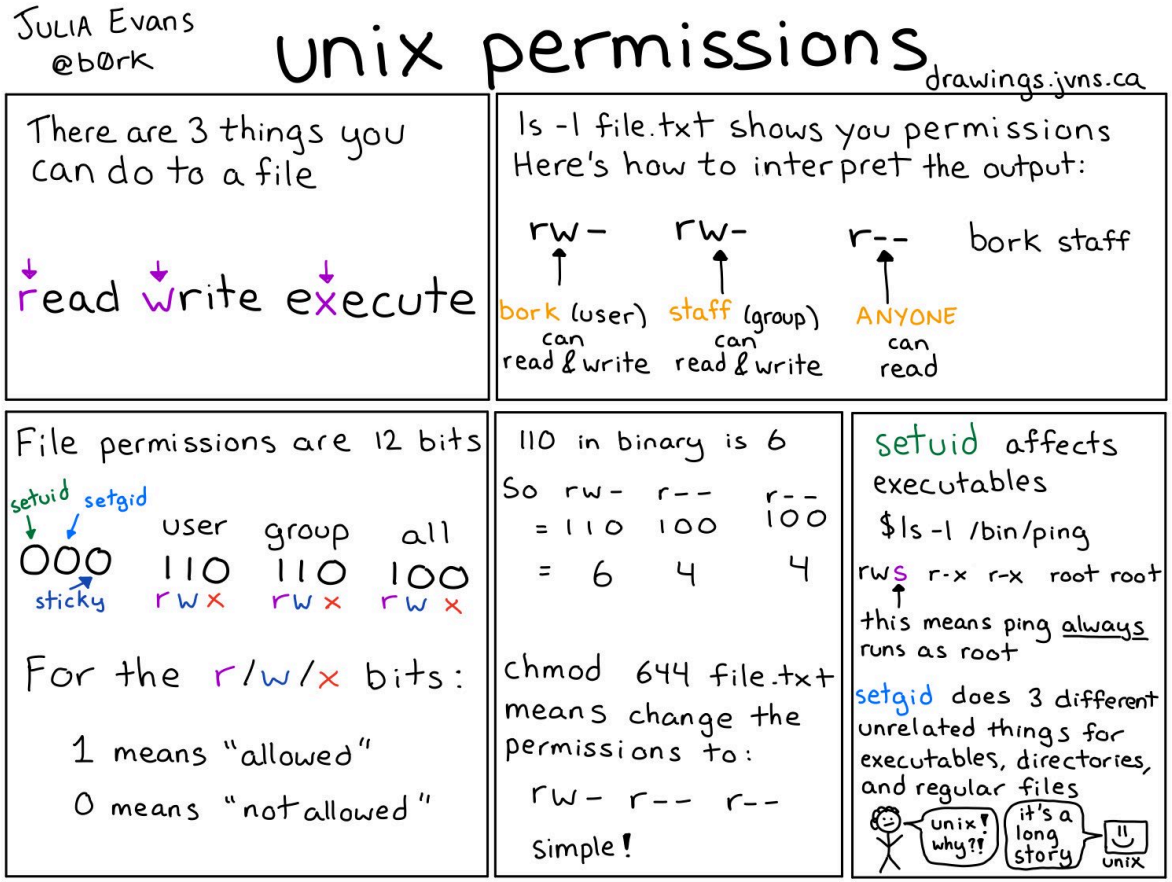
Data Anonymization

- The process of removing or modifying personal information so that individuals cannot be identified directly or indirectly.
- Irreversible; once data is anonymized, it cannot be traced back to the individual.
- Fully anonymized data is not subject to GDPR regulations.
- **Examples**: Removing names, addresses, and any other identifiers from a dataset, often not easy to achieve.

Data Pseudonymization

- The process of replacing private identifiers with fake identifiers or pseudonyms.
- Reversible; data can be re-identified if the pseudonymization key is available.
- Pseudonymized data is still considered personal data under GDPR and must be protected accordingly.
- Useful for internal data processing, analysis, and scenarios where data needs to be re-identified later.
- **Examples**: Replacing names with unique codes, removing other identifiers.

Sharing with other users



Note: `x` on a directory means `traverse` permission

https://wizardzines.com/comics/permissions/

Sharing with other users

Make `directory` writable for the group

```
chmod g+rwx directory
```

Make `file` readable by everyone

```
chmod o+r file
```

Make `directory` reachable by everyone

```
chmod o+x directory
```

Make `directory` contents readable by everyone, recursively

```
chmod o+rX directory
```

Sharing with other users

All parent directories must be *traversable*

```
[dfr@lm4-f001 Data]$ namei -l $(realpath random.dat)
f: /home/users/d/f/dfr/Data/random.dat
dr-xr-xr-x root root /
drwxr-xr-x root root home
drwxr-xr-x root root users
drwxr-xr-x root root d
drwxr-xr-x root root f
drwxr-x--x dfr  dfr  dfr
drwxrwx--- dfr  dfr  Data
-rw-rw-r-- dfr  dfr  random.dat
```

Sharing with a group

See which groups you are part of:

```
[dfr@lm4-f001 ~]$ id
uid=3000003(dfr) gid=3000003(dfr) groups=3000003(dfr),
4999998(adminucl),4999999(sysadmin)
```

Change group ownership (as a regular user):

```
[dfr@lm4-f001 ~]$ ls -ld Data
drwxrwx--- 4 dfr dfr 7 Sep 17 11:35 Data
[dfr@lm4-f001 ~]$ chgrp adminucl Data/
[dfr@lm4-f001 ~]$ ls -ld Data
drwxrwx--- 4 dfr adminucl 7 Sep 17 11:35 Data
```

Sharing with a group

By default, the group of a newly created file is the creator's primary group.

```
[dfr@lm4-f001 Data]$ touch testone
[dfr@lm4-f001 Data]$ ls -l testone
```

Unless `newgrp` is used to change the group for the current session:

```
[dfr@lm4-f001 Data]$ newgrp adminucl
[...]
[dfr@lm4-f001 Data]$ touch testtwo
[dfr@lm4-f001 Data]$ ls -l testtwo
-rw-rw---- 1 dfr adminucl 0 Sep 18 10:45 testtwo
[dfr@lm4-f001 Data]$ exit
```

Sharing with a group

By default, the group of a newly created file is the creator's primary group.

```
[dfr@lm4-f001 Data]$ touch testone
[dfr@lm4-f001 Data]$ ls -l testone
```

or the parent directory has `sgid` permission bit set:

```
[dfr@lm4-f001 Data]$ ls -ld .
drwxrwx--- 4 dfr adminucl 9 Sep 18 10:45 .
[dfr@lm4-f001 Data]$ chmod g+s .
[dfr@lm4-f001 Data]$ ls -ld .
drwxrws--- 4 dfr adminucl 9 Sep 18 10:45 .
[dfr@lm4-f001 Data]$ touch testthree
[dfr@lm4-f001 Data]$ ls -l testthree
-rw-rw---- 1 dfr adminucl 0 Sep 18 10:48 testthree
```

Sharing and hiding

When a common group is not available for sharing, the file can be *world-readable* in a *non-readable* but *traversable* directory. Then only users who know about the file exact name can open it.

```
[dfr@lm4-f001 ~]$ chmod o+x Download
[dfr@lm4-f001 Downloads]$ namei -l $(realpath rqlite-v7.21.1-linux-amd64.tar.gz)
f: /home/users/d/f/dfr/Downloads/rqlite-v7.21.1-linux-amd64.tar.gz
dr-xr-xr-x root root /
drwxr-xr-x root root home
drwxr-xr-x root root users
drwxr-xr-x root root d
drwxr-xr-x root root f
drwxr-x--x dfr  dfr  dfr
drwxrwx--x dfr  dfr  Downloads
-rw-rw-r-- dfr  dfr  rqlite-v7.21.1-linux-amd64.tar.gz
```

```
[bvr@lm4-f001 ~]$ ls ~dfr/Downloads/
ls: cannot open directory '/home/ucl/pan/dfr/Downloads/': Permission denied
[bvr@lm4-f001 ~]$ ls ~dfr/Downloads/rqlite-v7.21.1-linux-amd64.tar.gz
/home/ucl/pan/dfr/Downloads/rqlite-v7.21.1-linux-amd64.tar.gz
[bvr@lm4-f001 ~]$ file ~dfr/Downloads/rqlite-v7.21.1-linux-amd64.tar.gz
/home/ucl/pan/dfr/Downloads/rqlite-v7.21.1-linux-amd64.tar.gz: gzip compressed data, ...
```

Sharing and encrypting

The gocryptfs tool makes the process easy.

1. Install it

```
wget https://github.com/.../gocryptfs_v2.4.0_linux-static_amd64.tar.gz
tar xvzf gocryptfs_v2.4.0_linux-static_amd64.tar.gz
chmod +x gocryptfs
mv gocryptfs [some directory in your PATH]
```

https://github.com/rfjakob/gocryptfs/releases/download/v2.4.0/gocryptfs_v2.4.0_linux-static_amd64.tar.gz

Sharing and encrypting

2. Create a directory that will contain the encrypted files and initialise a *vault*

```
[dfr@lm4-f001 ~]$ mkdir $CECIHOME/SecretFolder
[dfr@lm4-f001 ~]$ gocryptfs -init $CECIHOME/SecretFolder
Choose a password for protecting your files.
Password:
Repeat:

Your master key is:

    1a88a6b1-8f072fe8-7aac5356-1d025115-
    7574f7c3-627cbbdb-12b96ca8-09bfb39a

If the gocryptfs.conf file becomes corrupted or you ever forget your password,
there is only one hope for recovery: The master key. Print it to a piece of
paper and store it in a drawer. This message is only printed once.
The gocryptfs filesystem has been created successfully.
You can now mount it using: gocryptfs /CECI/home/ucl/pan/dfr/SecretFolder MOUNTPOINT
```

Sharing and encrypting

3. *mount* the vault in a temporary directory

```
[dfr@lm4-f001 ~]$ gocryptfs $CECIHOME/SecretFolder ./Tests/ClearFolder
Password:
Decrypting master key
Filesystem mounted and ready.
```

4. Write files to the temporary directory

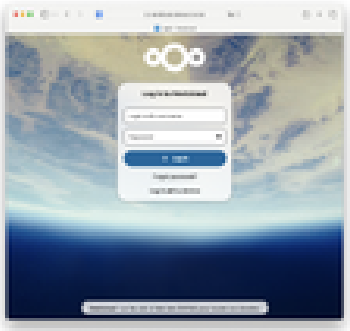
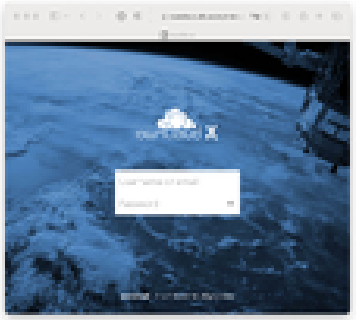
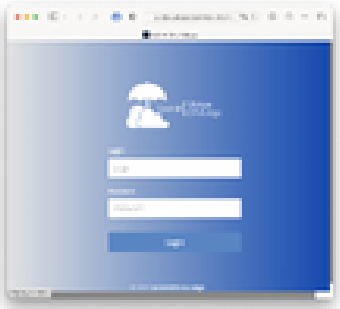
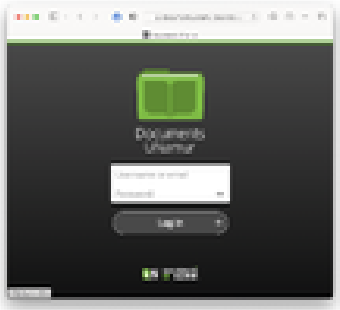
```
[dfr@lm4-f001 ~]$ echo test > ./Tests/ClearFolder/test.txt
[dfr@lm4-f001 ~]$ ls ./Tests/ClearFolder
test.txt
[dfr@lm4-f001 ~]$ ls $CECIHOME/SecretFolder
e6AxIMr4RuztuwpA-o_uOQ  gocryptfs.conf  gocryptfs.diriv
```

The files are encrypted on the fly.

Cleanup with `fusermount -u ./Tests/ClearFolder`

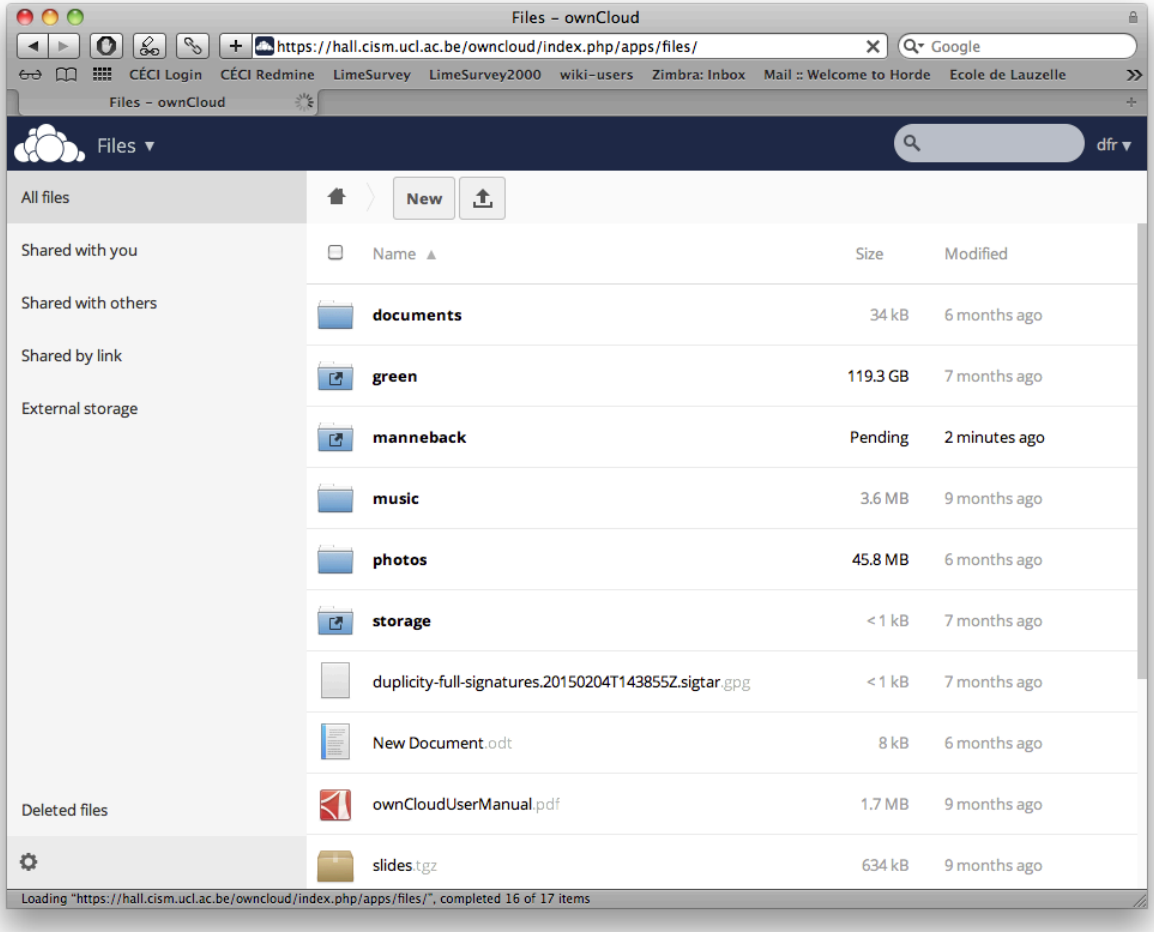
Sharing with external colleagues

Private cloud

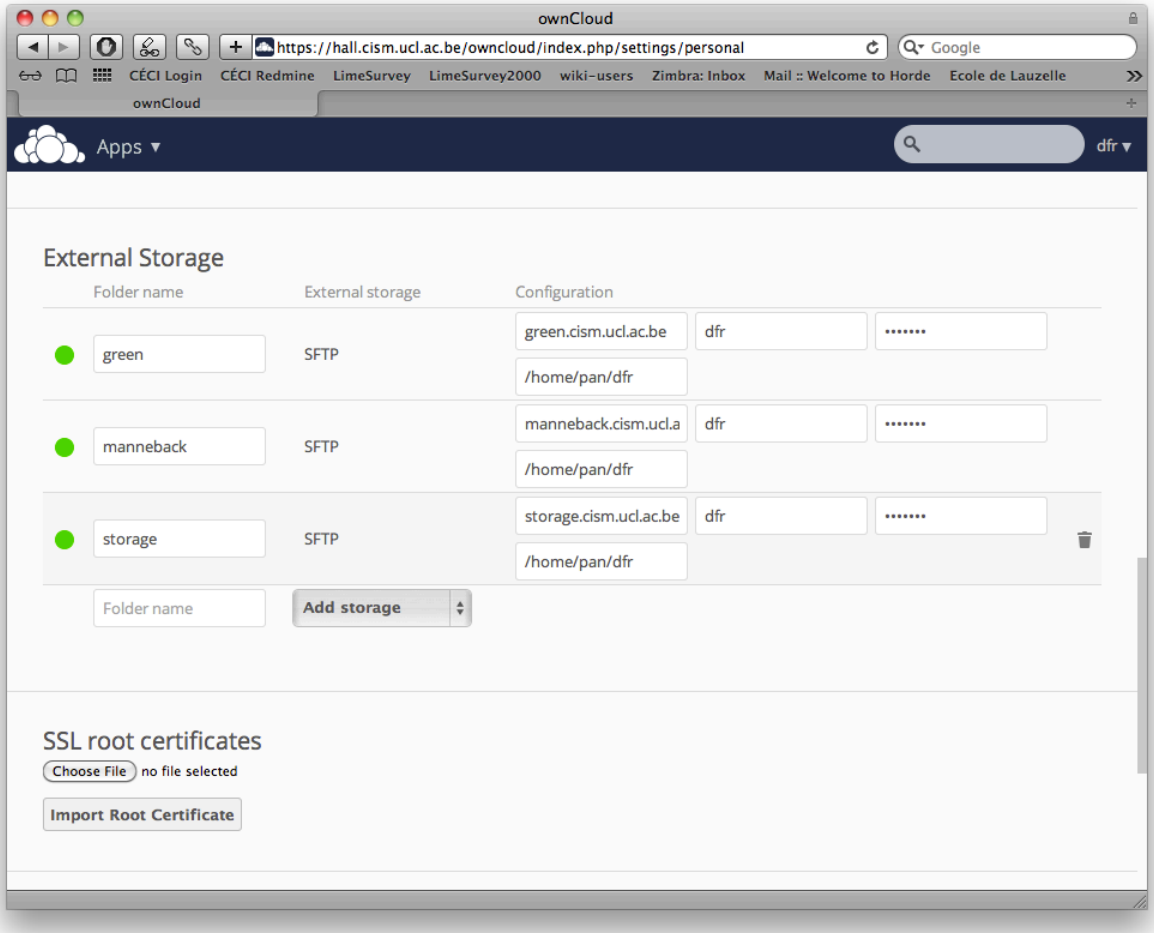
Open data


Private cloud



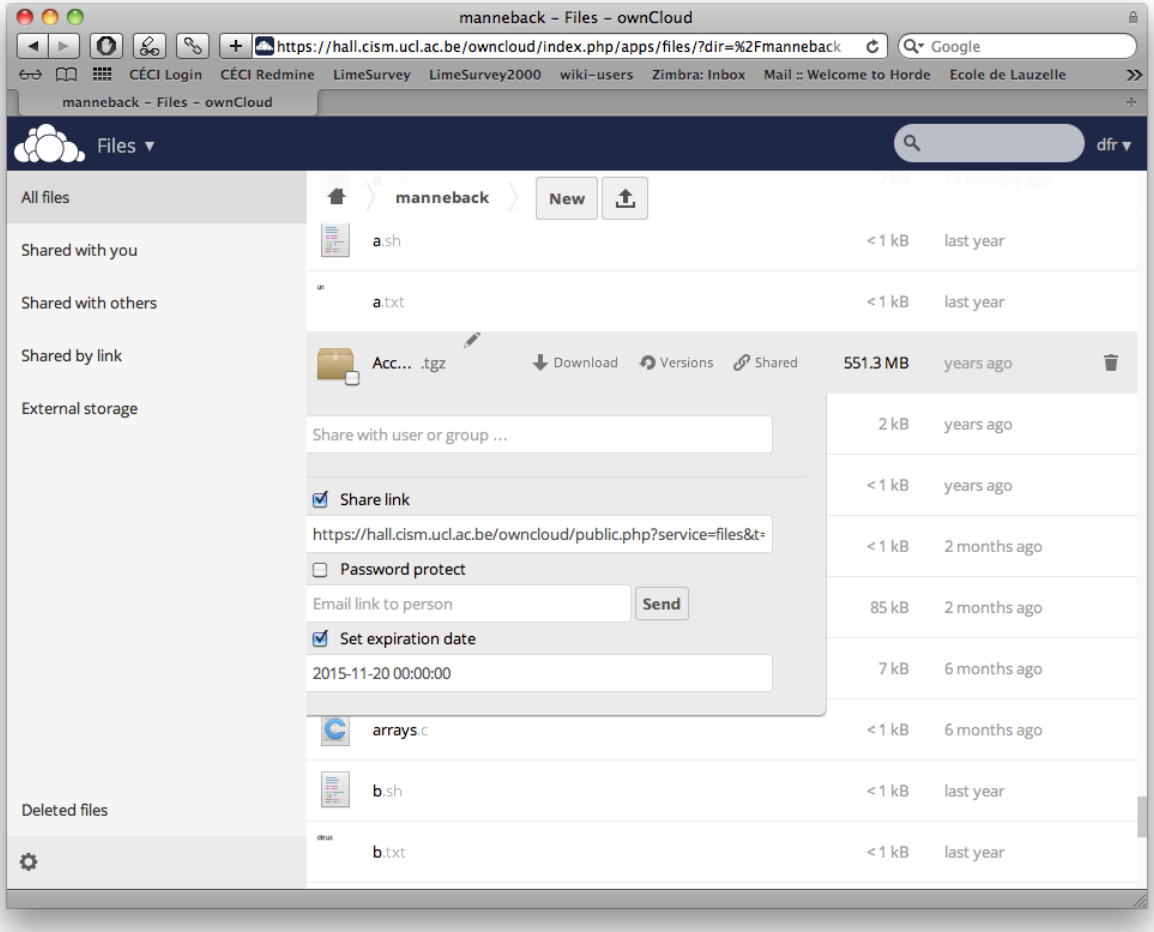
Similar to Dropbox, OneDrive, Google Drive, etc.

Private cloud



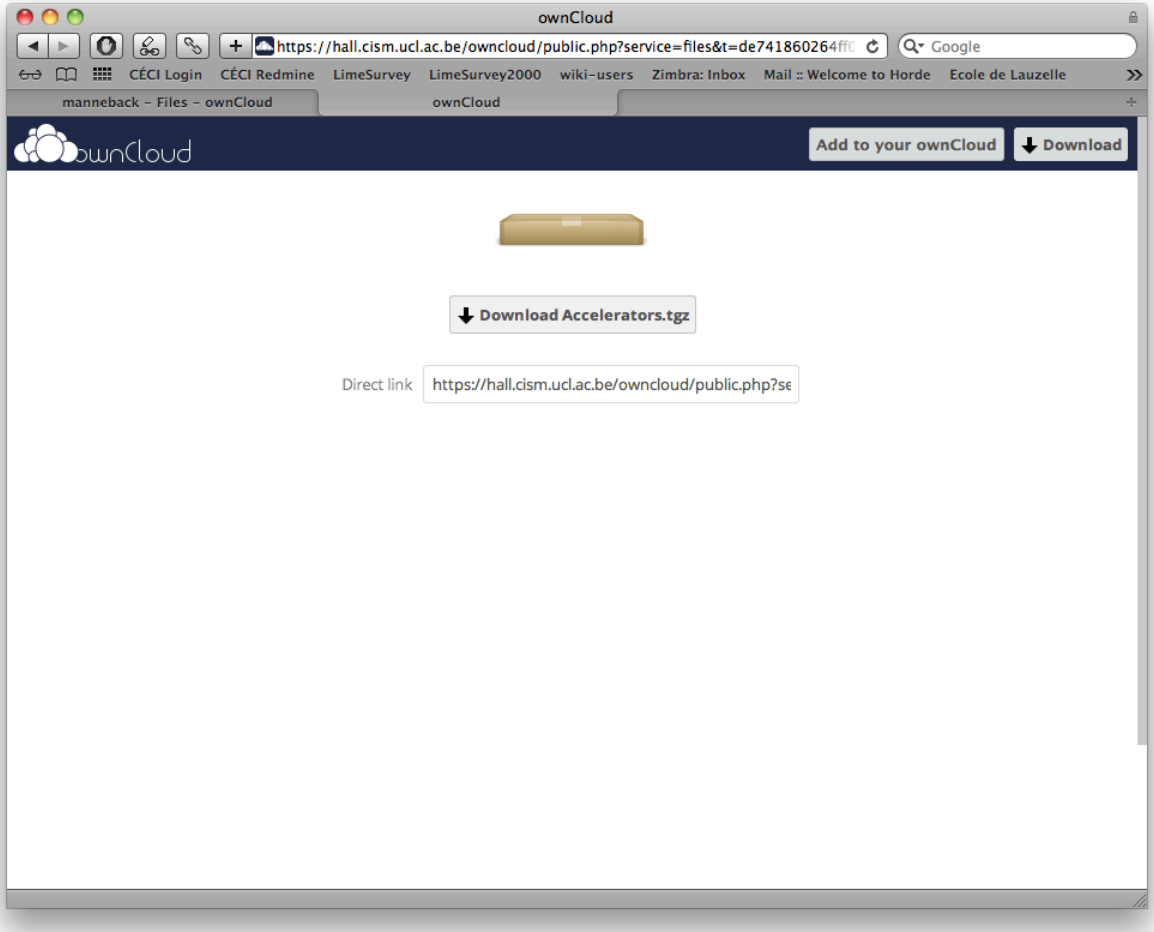
Possibility to connect external storage

Private cloud



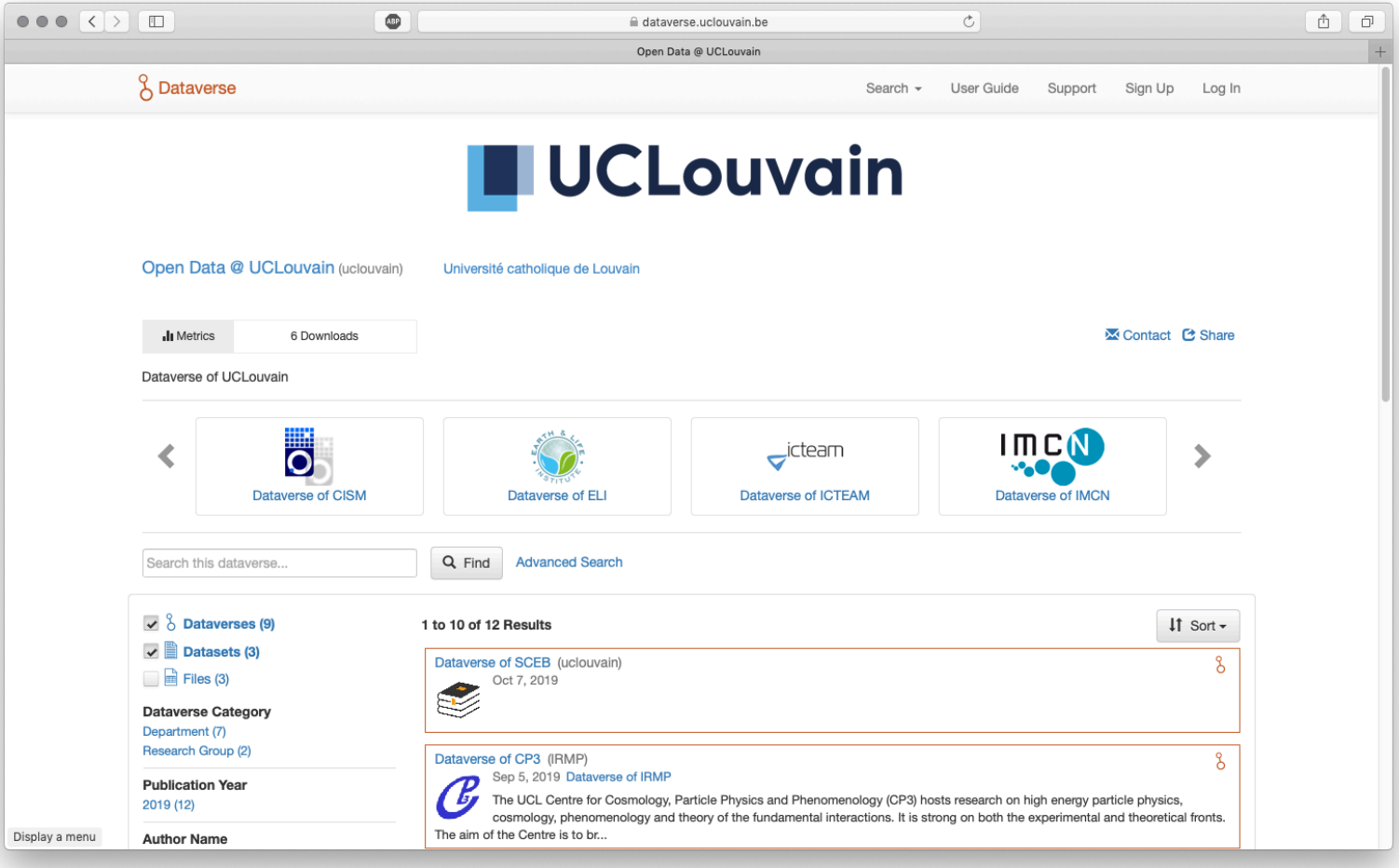
Share with a download link

Private cloud



Share with a download link

Open data



Publish and reference your data with a DOI

Summary and recap

Summary and recap

Storage: choose the right file system and the right file format and give other storage systems some consideration

Transfer: use rsync in parallel when you can

Sharing: use all the potential of the UNIX permissions and try Nextcloud and Dataverse