CCIN2P3

cnrs

Centre de Calcul
de l'Institut National de Physique Nucléaire
et de Physique des Particules

Formats de fichiers et accès aux données après l'acquisition, usages au CC-IN2P3

Journées Online 2025, IJClab

Loïc Tortay

Formats de fichiers/données (communs au CC-IN2P3) :

- ROOT : format historique/dominant HEP
- FITS : images, en-têtes texte+binaire encodé, AstroPy & libcfitsio
- HDF5 & Zarr : auto-description du contenu
  Zarr « optimisé pour les accès parallèles » (hiérarchies de répertoires/fichiers & métadonnées, pas « un » fichier)
- Archives (tar, zip, etc.), comme conteneurs logiques (Zarr, ROOT, ...) ou pas
- NumPy & Pickle : simplicité de création et relecture en Python
- texte y compris JSON
- SQlite & base de données clefs/valeurs monofichier (BDB, etc.)
- autres : GWF, Google Protocol Buffers, « dat » et autres inconnus, ...

Systèmes de stockage (de fichiers?) au CC-IN2P3 : [1/4]

- PBS : répertoires personnels et de groupes (collaborations, labos) pour données *importantes* (sauvegarde + snapshots accessibles aux utilisateurs)
  - système de fichiers ⇨ exécution de programmes, modification/accès directs (API POSIX), *accès direct* ⇨ lecture/écriture de portions de fichiers
  - quota utilisateur de base : 20 Gio, groupe : 100 Gio, mais quotas plus élevés communs
- SPS : répertoires de groupes pour données actives (pas de sauvegarde)
  - système de fichiers ⇨ programmes & modification/accès directs (POSIX)
  - quotas de 5 Tio à 8.5 Pio (12 Pio/3 milliards de fichiers stockés)
  - nettoyage/gestion des données (DMP, etc.)

Systèmes de stockage (de fichiers?) au CC-IN2P3 : [2/4]

- XRootD : historiquement pour fichiers ROOT, au CC-IN2P3 : presque exclusivement en lecture seule, en frontal de HPSS (bandes)
  - accès direct avec ROOT ou à des fichiers complets avec `xrdcp` ou `xrdfs`
  - pas un système de fichiers : pas d'exécution de programmes ou de modification en place (dans les cas où l'écriture est possible)
- dCache : données utilisées par les jobs « grilles » (WLCG + EGEE + LSST)
  - pas d'accès direct, « portes » ROOT ou WebDAV (préféré, permet l'accès direct en lecture)
  - principal stockage de données actives au CC (~55 Po stockés)
  - données sur disques et/ou transférées vers HPSS (choix utilisateur/groupe), mais pas accessibles sans utiliser dCache

Systèmes de stockage (de fichiers?) au CC-IN2P3 : [3/4]

- iRODS : gestion et transfert de données, principalement en frontal de HPSS
  - méthode recommandée pour écrire des données (au final) sur bandes
  - au CC-IN2P3 : connexion XRootD ⇨ fichiers écrits avec/par iRODS dans HPSS accessibles avec XRootD (et iRODS)
  - petits fichiers & autres conservés sur disque
- HPSS : stockage de masse sur bandes (~220 Po stockés, données semi-actives, peu actives ou inactives)
  - accès possible avec les outils historiques (RFIO : `rfcp`, etc.)
  - iRODS/XRootD recommandés pour lire/écrire les données
  - bandes ⇨ efficaces avec des gros accès/fichiers, très inefficaces avec des petits fichiers

Systèmes de stockage (de fichiers?) au CC-IN2P3 : [4/4]

- Base de données (relationnelles ou pas) : PostgreSQL, MariaDB (MySQL), Oracle ou MongoDB
  - sauf exception, en 2025, pas de données expérimentales (mais calibration et autres types de données communs)
- Autres :
  - Ceph : pas de service Ceph pour les utilisateurs, multiples clusters Ceph pour infrastructures Web, SPS, LSST, OpenStack, etc.
  - S3 (ou équivalents) : comme Ceph, utilisations internes

Accès aux données :

- Les systèmes de fichiers (POSIX) exposent une taille d'entrées/sorties « préférée » (`stat().st_blksize`), qui évite des choix arbitraires
- Parallélisme d'accès en :
  - lecture : « facile », en particulier avec des gros fichiers/fichiers multiples
  - écriture : verrous pour des fichiers partagés, pas nécessairement avec des fichiers multiples/indépendants
- `mmap()` pas nécessairement efficace avec du stockage distribué/parallèle (i.e. pas toujours une « bonne » optimisation)

Tailles de fichiers : [1/2]

- Nouveaux usages (IA générative) ⇨ multiplication des très petits fichiers (PBS+SPS)
- Fichiers « auxiliaires » (*ancillary data*) : SQlite, attributs étendus ?
- Archives/conteneurs de (petits) fichiers (ZIP pour ROOT ou LSST, Zarr, ...)
- Différents systèmes de stockage & tailles de fichiers :
  - HPSS : 100 Mio ou plus (Gio préférés), 4 Tio maxi (évolution possible)
  - SPS : pas de limite basse *imposée*, 4 Tio maxi, fichiers > 128 Gio peu communs, > 1 Tio rares
- Raisonnablement en 2025 (transferts et autres) : fichiers de 8 à 32 voire 64 Gio, au delà ⇨ problématiques de gestion (transfert efficace, curation, etc.)

Tailles de fichiers : [2/2]

- Petits fichiers très communs dans SPS :

Directory or file in "/sps[redacted]	Space used	# of files	Average file size	Median file size	# of directories	# of symlinks	Most recent file access	Most recent file modification
↳↳ users[redacted]	44.65 TiB	16.18M	2.89 MiB	85 B	3.16M	0	2025/01/16 21:22	2025/01/13 21:17

User	Space used	On disk overhead	Files	Hard-linked files	Minimum file size	Median file size	Average file size	Maximum file size	Directories	Symlinks	Most recent file access	Most recent file modification
[redacted]	1.85 TiB	634.52 GiB	6.27M	0	0 B	8 B	316.19 KiB	60.76 GiB	70.31k	530	2025/11/09 09:19	2025/11/09 09:19

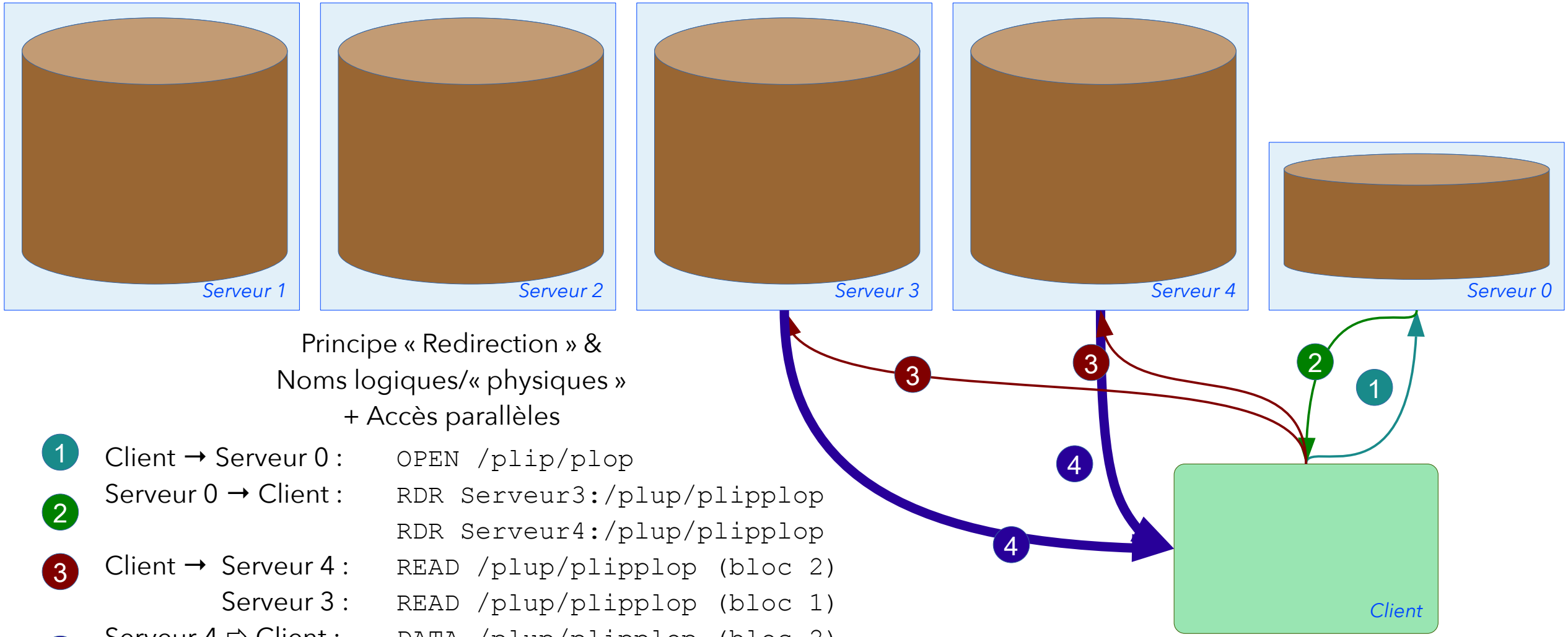
- Extrême opposé possible (liste *openssh-unix-dev@mindrot.org*, mars 2025, pas au CC-IN2P3) :

> Does OpenSSH scp/sftp mode transfer sparse files correctly, i.e. are holes skipped and not transferred as chunks of 0 bytes? [1]
>
> We're asking about sparse **files in the >= 1PB range, which consists of multi-TB holes with around 600-2000GB of valid data.**

Merci

Backup Slides : *basic storage best practices* (recyclage de supports pour le DU Data Science Université Clermont-Auvergne/CC-IN2P3)

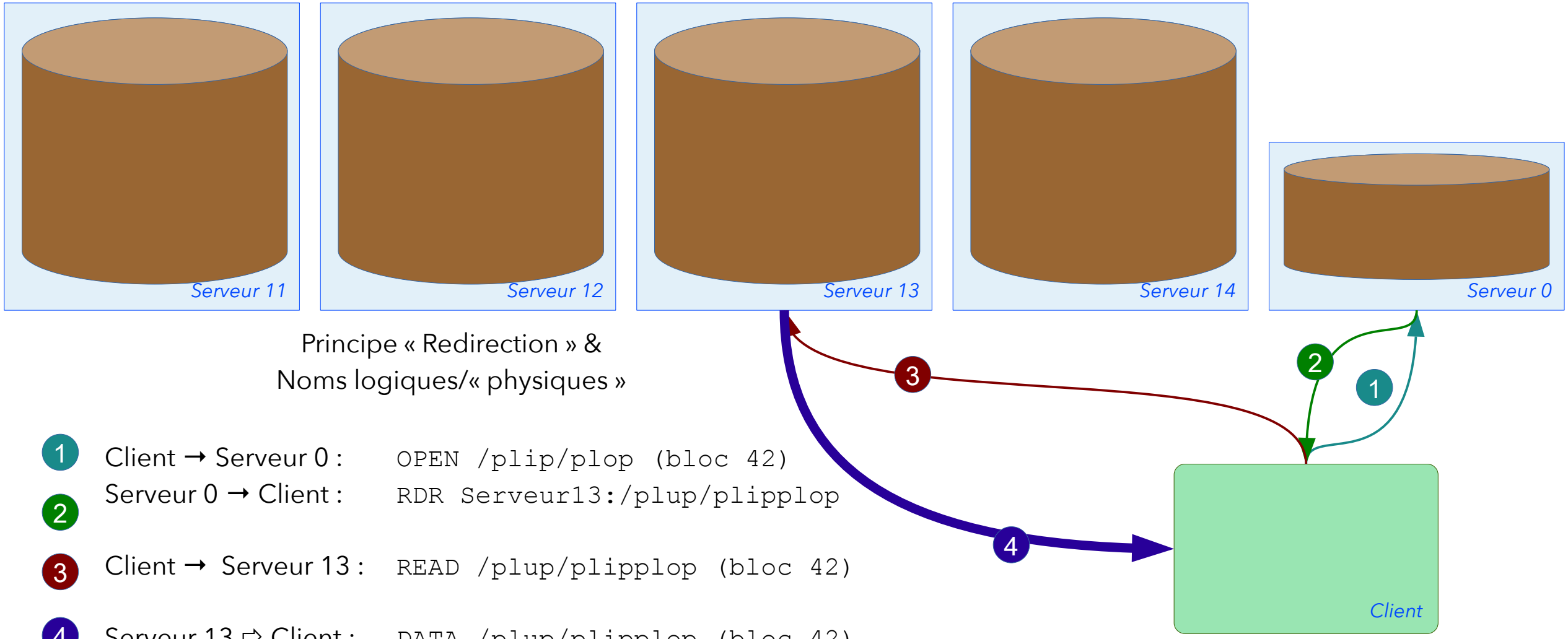
Distributed storage: redirection/parallel access



Principe « Redirection » &
Noms logiques/« physiques »
+ Accès parallèles

1. Client → Serveur 0 : `OPEN /plip/plop`
2. Serveur 0 → Client : `RDR Serveur3:/plup/plipplop`
   `RDR Serveur4:/plup/plipplop`
3. Client → Serveur 4 : `READ /plup/plipplop (bloc 2)`
   Serveur 3 : `READ /plup/plipplop (bloc 1)`
4. Serveur 4 ⇨ Client : `DATA /plup/plipplop (bloc 2)`
   Serveur 3 ⇨ Client : `DATA /plup/plipplop (bloc 1)`

Distributed storage: redirection/single block access



Principe « Redirection » &
Noms logiques/« physiques »

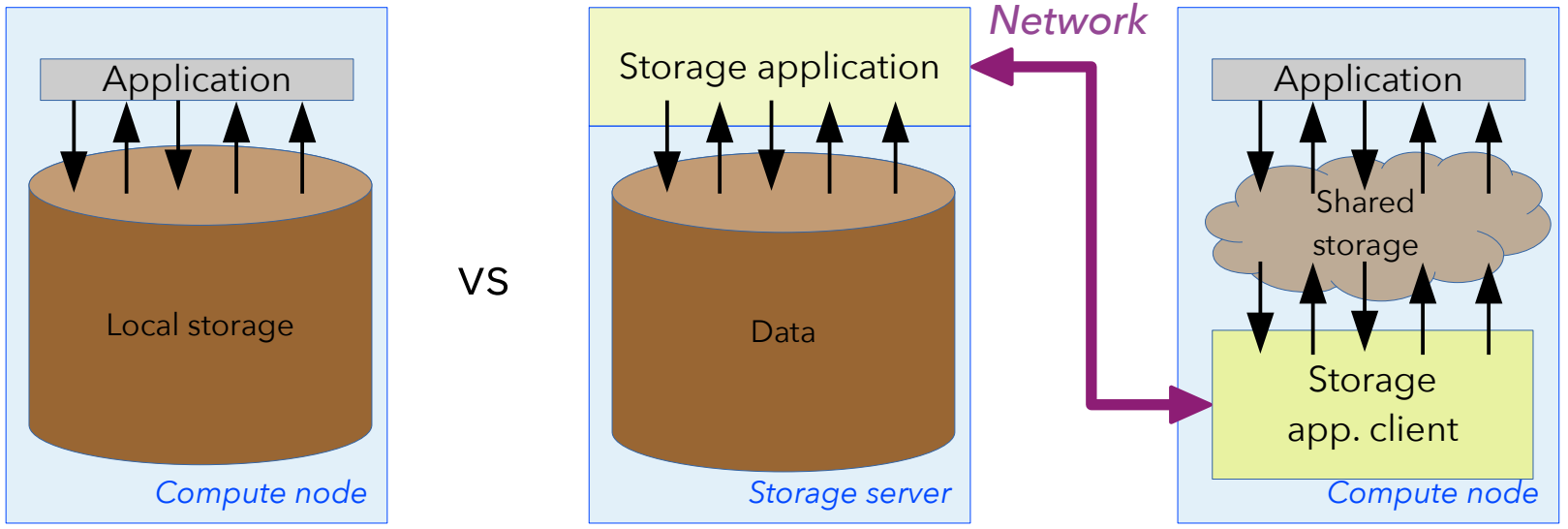
1. Client → Serveur 0 : `OPEN /plip/plop (bloc 42)`
2. Serveur 0 → Client : `RDR Serveur13:/plup/plipplop`
3. Client → Serveur 13 : `READ /plup/plipplop (bloc 42)`
4. Serveur 13 ⇨ Client : `DATA /plup/plipplop (bloc 42)`

Basic data access best practices [1/2]

- Write only what matters, while storage may be "cheap", data management is not
- Avoid intermediary files (or write them locally)
- Read/write largest possible relevant *chunks*, some storage systems provide a *preferred* I/O size information (POSIX API)
- Group writes (at the thread, process or file level)
- Use *efficient* high-level data libraries (ROOT, HDF5, NumPy, ...) instead of creating your own
- Avoid *synchronous* I/O unless you're sharing a file between machines or if it's the final output
- Please do not `stat()` (or `access()`) a file before opening it, `open()` will tell you if/why it fails

Basic data access best practices [2/2]

- Report the actual system error (message or `errno` value) to the user
- Copy or directly write output to a shared storage space iff data needs to persist after the (batch) job
- Avoid using shared storage (filesystems) as scratch space (unless directed to), local (temporary) storage is often faster (for throughput and latency)



Basic non-performance best practices

- Data management plans (DMP) are important, data management is tedious work
- Avoid putting all your files in the same directory
- Limit the number of files: more files ⇨ more work to manage data
- Avoid (lots of) extremely small files (< 512 B or 1 KiB), use a database when it makes sense
- Data structure in memory and on persistent storage do not need to be identical (*serialization/marshalling*)
- On Unix/Linux, be careful when you *seek* inside a file opened for:
  - reading: going past the end of a file will yield an error
  - writing: going past the end of a file will **not** (append & sparse files)

Basic best practices for filenames

- Use meaningful (and concise) directory names & filenames
- Even with Unicode, try to limit files/directories names to simple printable characters (`-+:%@_.`) and *alphanumericals* (`0-9, A-Z, a-z`):
  - avoid characters which can be interpreted by the shell or programming language, or difficult to use in command arguments
  - please **avoid** creating files named: `*, $, \, \n, \ESC^C^Z^D`
- When creating filenames with a program, make sure you create a printable name (C/C++, ROOT, ...): format numbers as text
- When a filename contains a date, use ISO-8601 date format (e.g. something between `2025-11-19` and `2025-11-19T09:58:19Z`)

Basic best practices for permissions

- Do **not** use `chmod 777`, `chmod -R 777` (or `chmod -R a+rwx`) or similar
- In most cases, removing (or renaming) a file does **not** require write access to the file itself but to the directory containing the file
- Extended access control lists (ACLs) can easily become counter-intuitive and difficult to manage, even with *default* ACLs (when supported)
- When required, extended ACLs are most useful when set on directories (so they apply to all files in the directory), instead of on individual files
- Many text editors (including vi/vim & emacs) don't preserve extended ACLs when saving files

Application design

- If possible, do not focus on a preferred storage solution:
  - parallel filesystems are nice, but does your use case actually need one (global consistency, in-place updates, single namespace, etc.) ?
  - object storage is nice, but do you actually need billions of objects or pseudo-infinite scalability ?
- **Ideally** use an I/O abstraction layer which will allow a switch to another storage system without changing the application itself
- For user defined metadata: use tags/extended attributes if available, otherwise use a database (even SQLite) instead of ancillary files (files about other files content)