
CERN

EOS developments

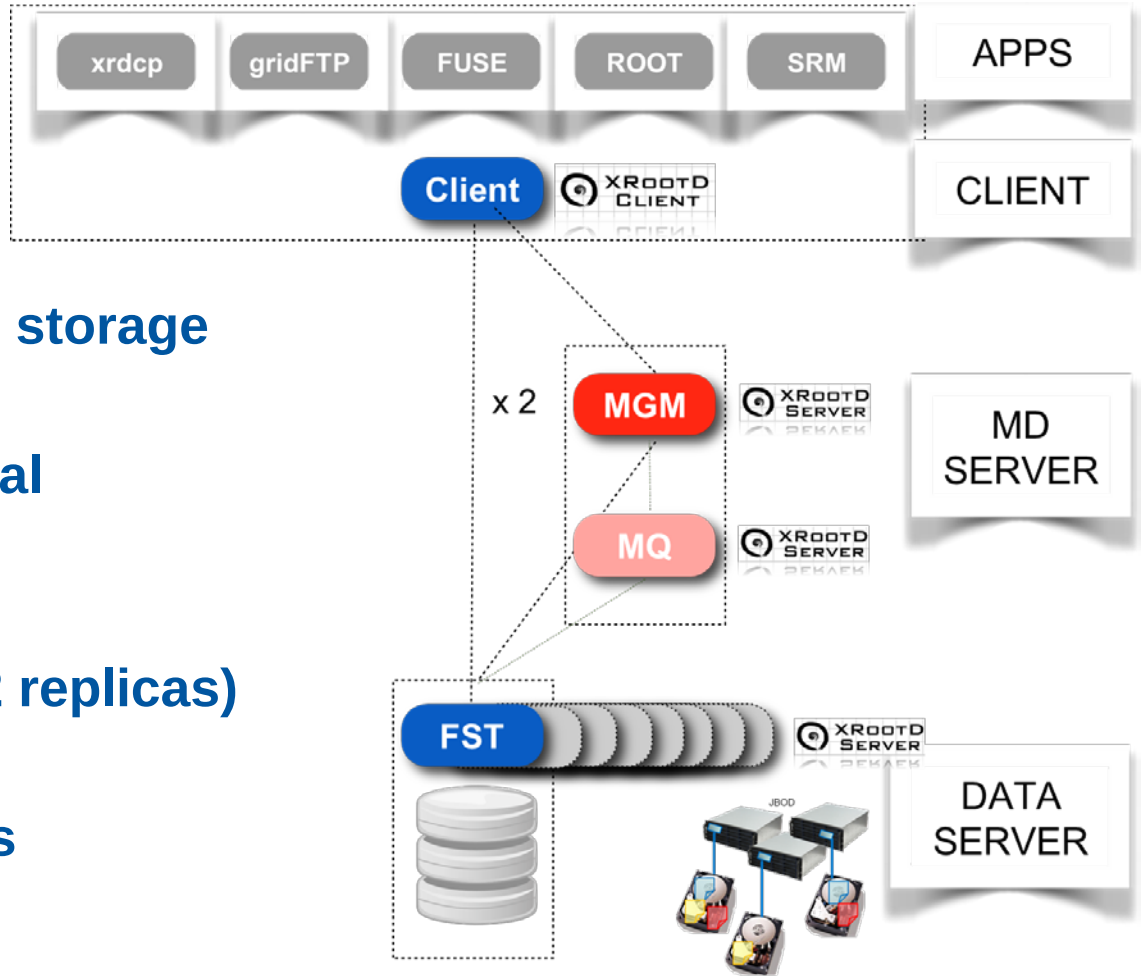
Elvin Sindrilaru - on behalf of the
EOS team and IT Storage Group

CHEP 2016 – San Francisco

Outline

- EOS architecture
- Releases and branches
- EOS FUSE status and improvements
- EOS Kinetic integration
- Future namespace architecture

EOS architecture



- Disk only physics file storage
- In memory hierarchical namespace
- File layouts (default 2 replicas)
- Physics data & others
- Low latency access

EOS releases and branches

- **Production** version
  - Branch: **beryl_aquamarine**
  - Release number: >= 0.3.210

- **Development** version (master)
  - Branch: **citrine**
  - Release number: >= 4.1.4
  - Requires **XRootD** 4.4.0

- **Feature branches** get merged into master e.g. kinetic, geo-scheduling, namespace devel. etc.

EOS FUSE status

- **Goal:** Help AFS retire gracefully

- Improved meta-data caching using the **Kernel buffer cache**
- Faster directory listing using **bulk meta-data queries**

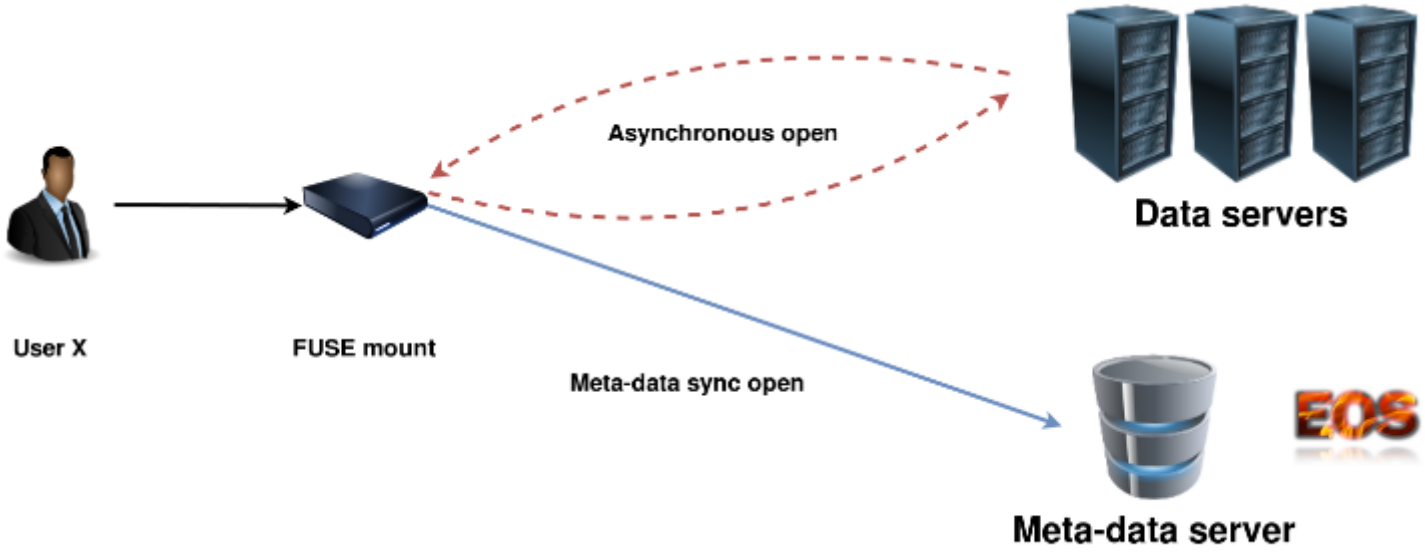
- **Multi-user mount** supporting user private **Kerberos** and **X509** authenticated connections
  - Already deployed on lxplus and lxbatch
  - Supports **user** and **session** bindings
  - Use **autofs** for better use experience

EOS FUSE multi-user mount

EOS FUSE latency optimisations

- **Write-back cache** with request aggregation

- **Lazy-open** implementation RO/RW
  - Separate meta-data and data paths
  - Data-server open happens on the first I/O operation
  - Hide latency using asynchronous open on data-server



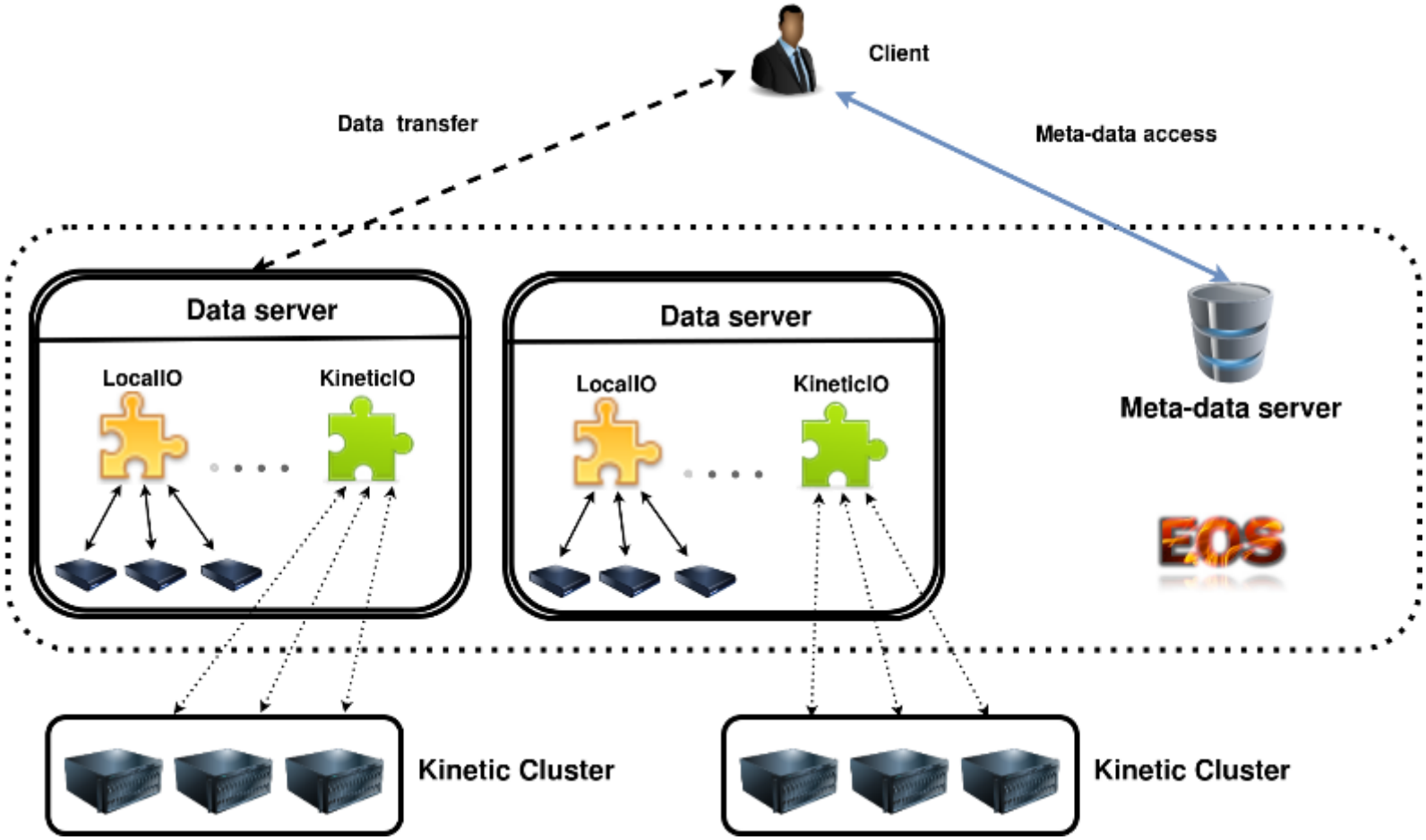
EOS Kinetic integration

- **Kinetic Open Storage Project**
  - HDDs with Ethernet interface
  - Key-value instead of block interface
  - Multi-vendor support: Seagate, Dell, Toshiba, RedHat, Cisco etc.
- **Benefits**
  - Reduced total cost of ownership (**TCO**)
  - **Robustness & scalability** – built-in replication, compression and CRC
  - Simple **abstract interface** – future proof against storage technology changes. Supported operations: put, get, delete, getnext etc.
- EOS integration done by **Paul Hermann Lensing, Seagate**

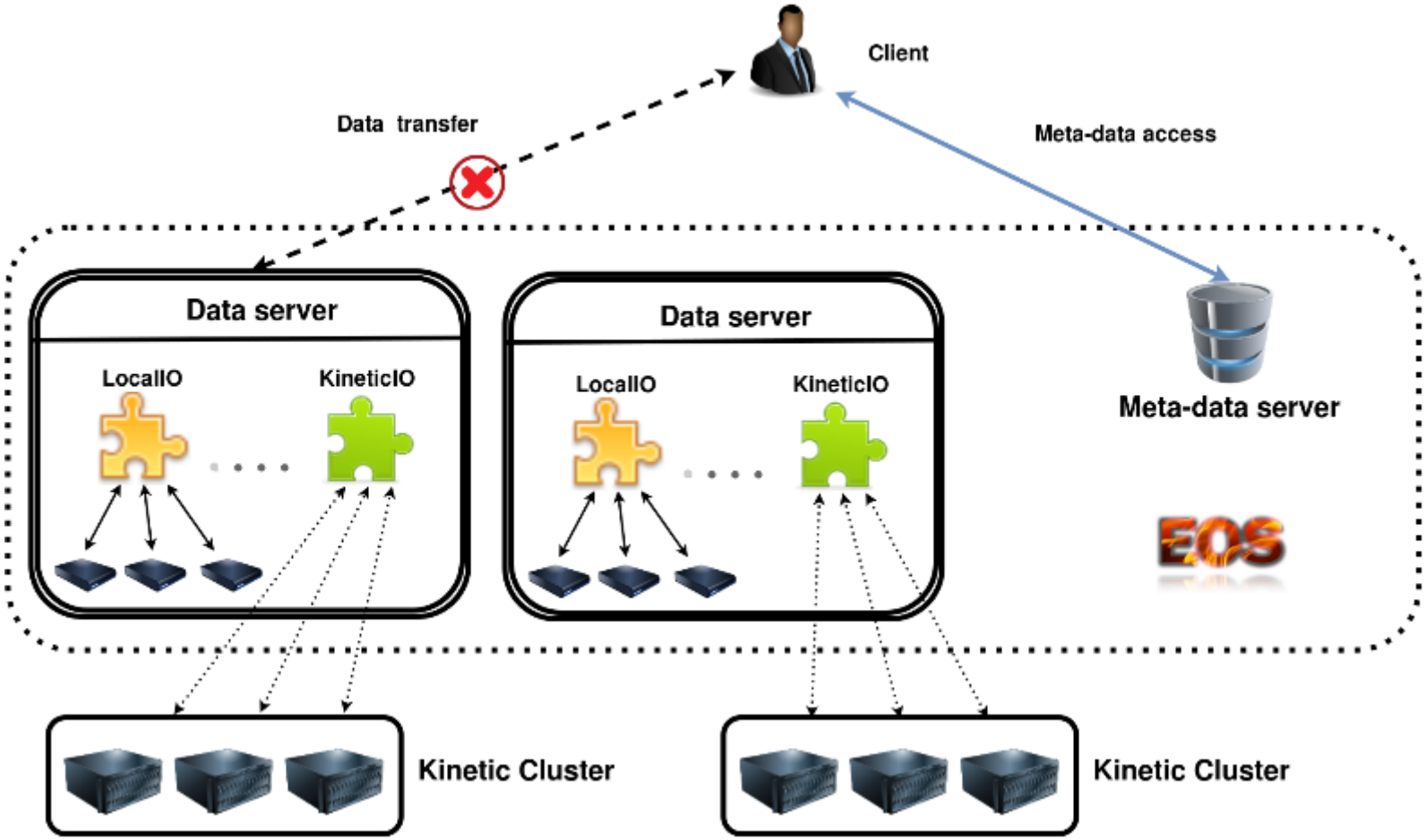
How EOS uses Kinetic?

- **Local cluster**
  - Attached to each individual **data-server**
  - Add Kinetic as a new **IO Plugin**
  - EOS is completely **agnostic** of the underlying IO access type

EOS with Kinetic local clusters



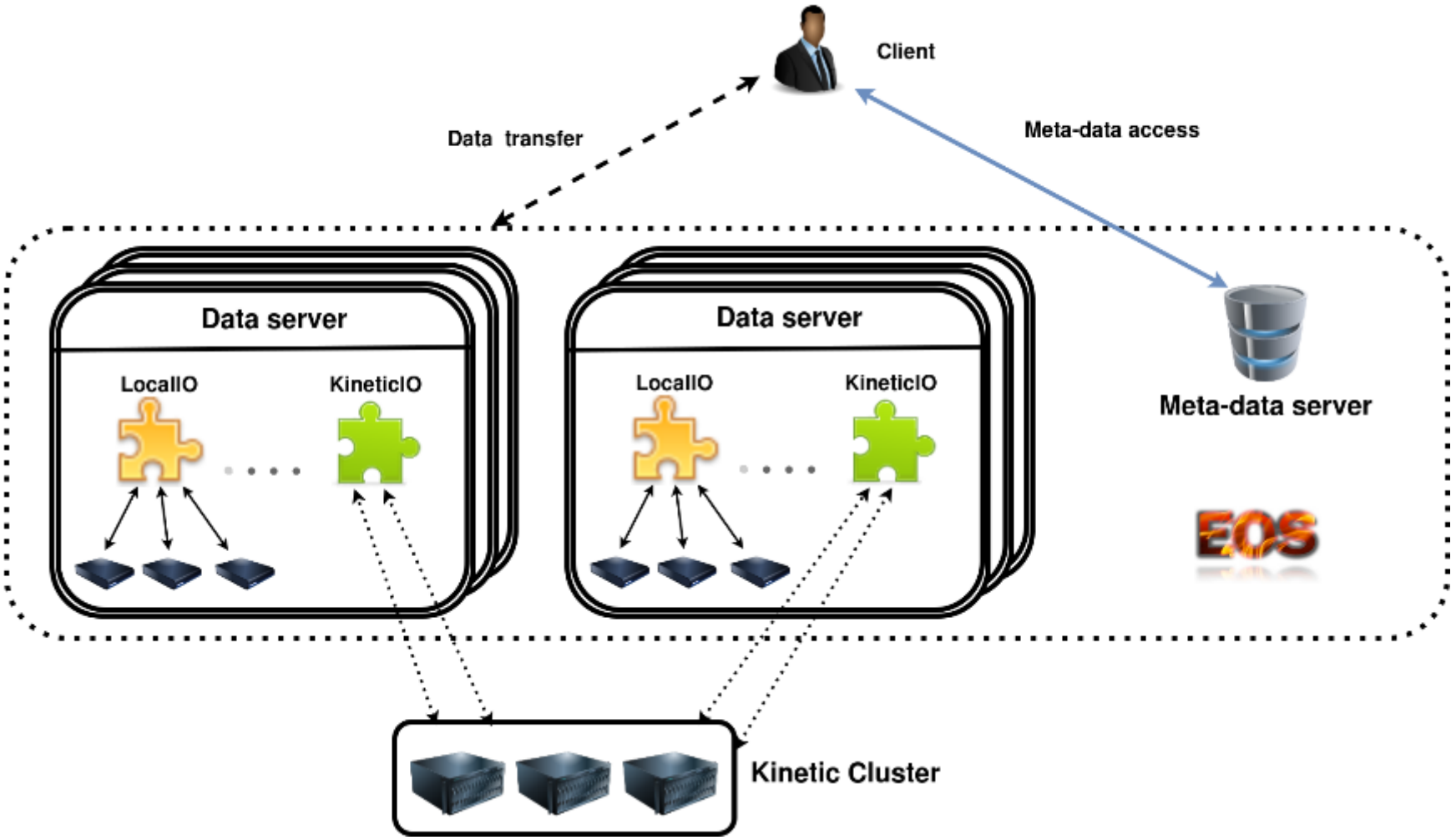
EOS with Kinetic local clusters



EOS Kinetic multi-path

- **One Kinetic cluster** shared by many data-servers
- Requires load-balancing and concurrency resolution → **Kinetic aware-scheduling**
- Fewer data-server can supply **higher storage capacity**
  - Data-server → Kinetic gateway
  - Fully utilize the combined **data-server network capacity**

EOS Kinetic multi-path

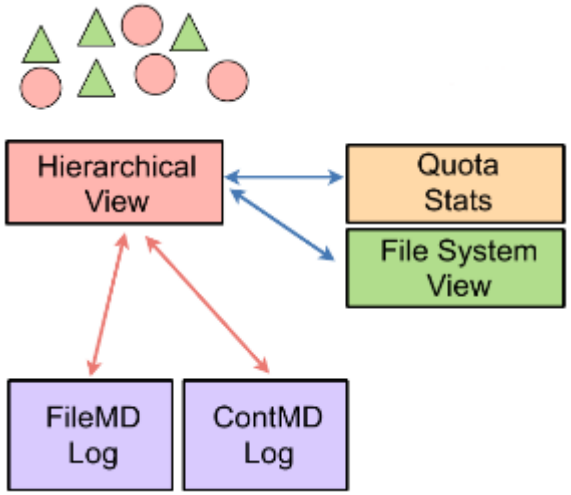


What is the EOS namespace?

- C++ library used by the EOS MGM node single-threaded
- Provides API for dealing with hierarchical collections of files

- **Filesystem elements**
  - Containers & files

- **Views**
  - Aggregate info about filesystem elem.
  - E.g QuotaView, FileSystemView etc.



- **Persistence objects**
  - Objects responsible for reading and storing filesystem elements
  - Implemented as binary change-logs

Namespace architectures pros/cons

- **Pros:**
  - Using hashes all in memory → **extremely fast**
  - Every change is logged → **low risk of data loss**
  - Views rebuilt at each boot → **high consistency**

- **Cons:**
  - For big instances it requires **a lot** of RAM
  - Booting the namespace from the change-log takes long

EOS Namespace Interface

- Prepare the setting for different namespace implementations
- Abstract a **Namespace Interface** to avoid modifying other parts of the code

- **EOS citrine 4.***

  - **Plugin manager** – able not only to dynamically load but also stack plugins if necessary

  - **libEosNsInMemory.so** – the original in-memory namespace implementation

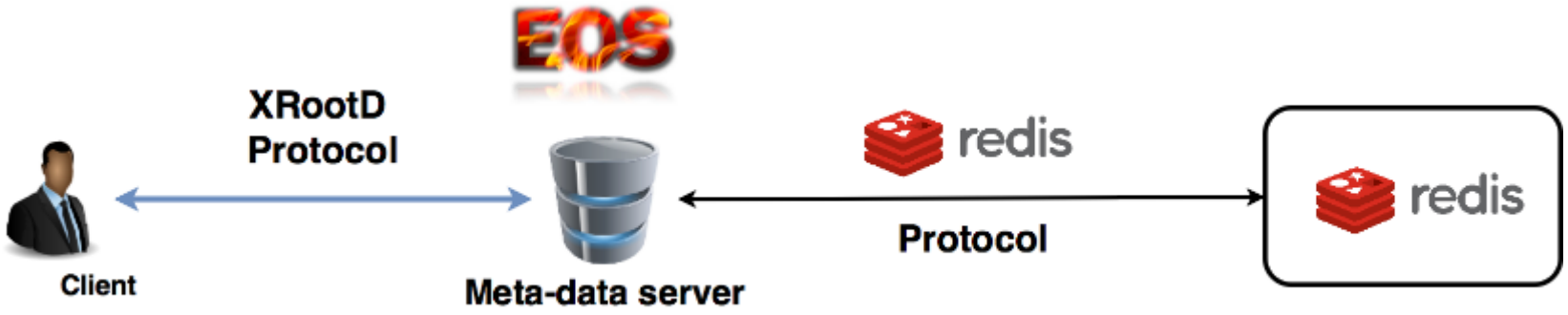
  - **libEosNsOnRados.so** – possible implementation on top of libRados

  - **libEosNsOnFilesystem.so** – possible implementation on top of a Linux filesystem

Why Redis?

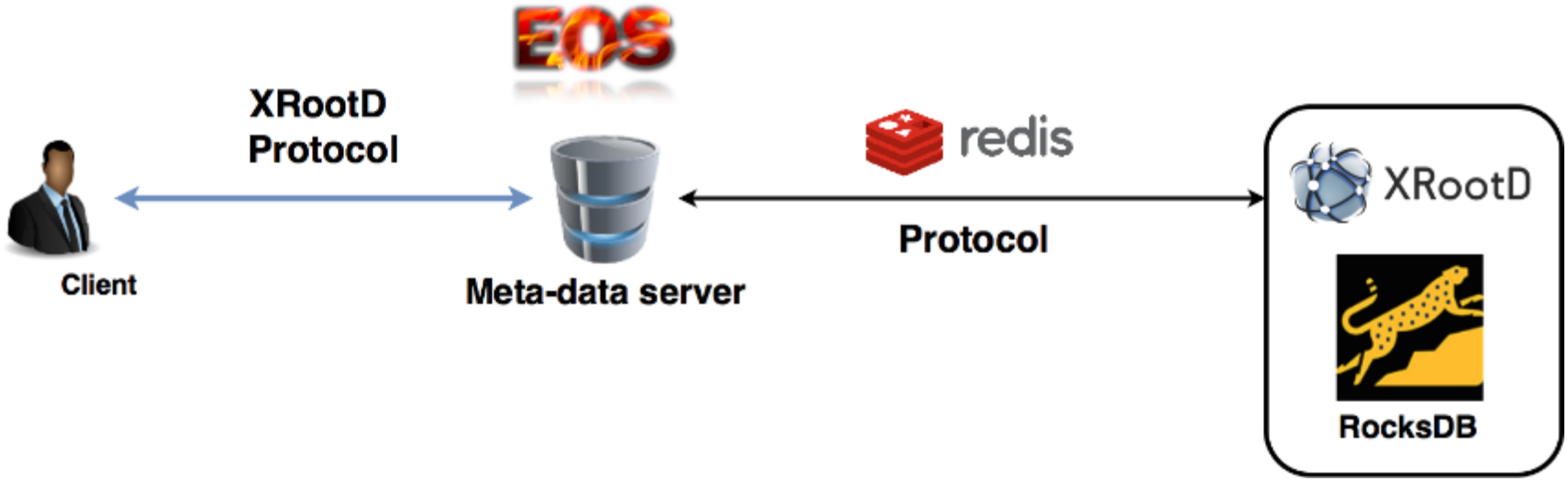
- **Redis** – in-memory **data structure store**
- Separate data from the application logic and user interface
- Supports various data structures: strings, hashes, lists, sets, sorted sets etc.

- Namespace implementation: **libEosOnRedis.so**



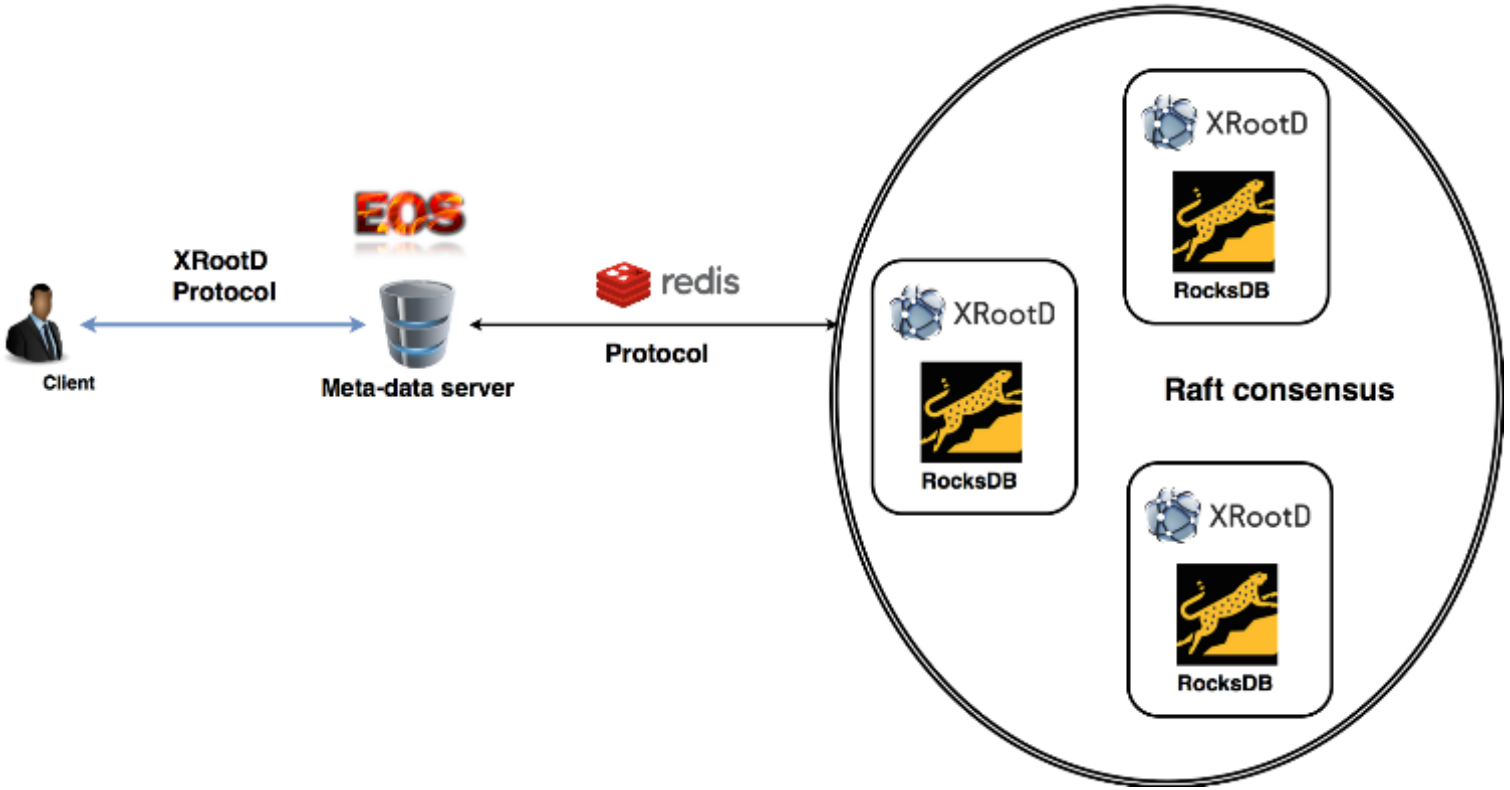
XRootD and Redis

- Replace Redis backend with XRootD
- Implemented as an XRootD **protocol plugin** – to be contributed upstream
- XRootD can use **RocksDB** as persistent key-value store



Namespace HA

- Ensure high-availability using the **Raft consensus algorithm**



Summary

- **EOS FUSE**
  - Strategic development to satisfy as many use-cases as possible
- **EOS Kinetic plugin storage backend**
  - Evaluated different deployment scenarios
  - Fully integrated with the existing system
- **EOS Namespace**
  - Separate the data form the application logic
  - Prototype on top of Redis and HA using Raft

www.cern.ch