
CERN

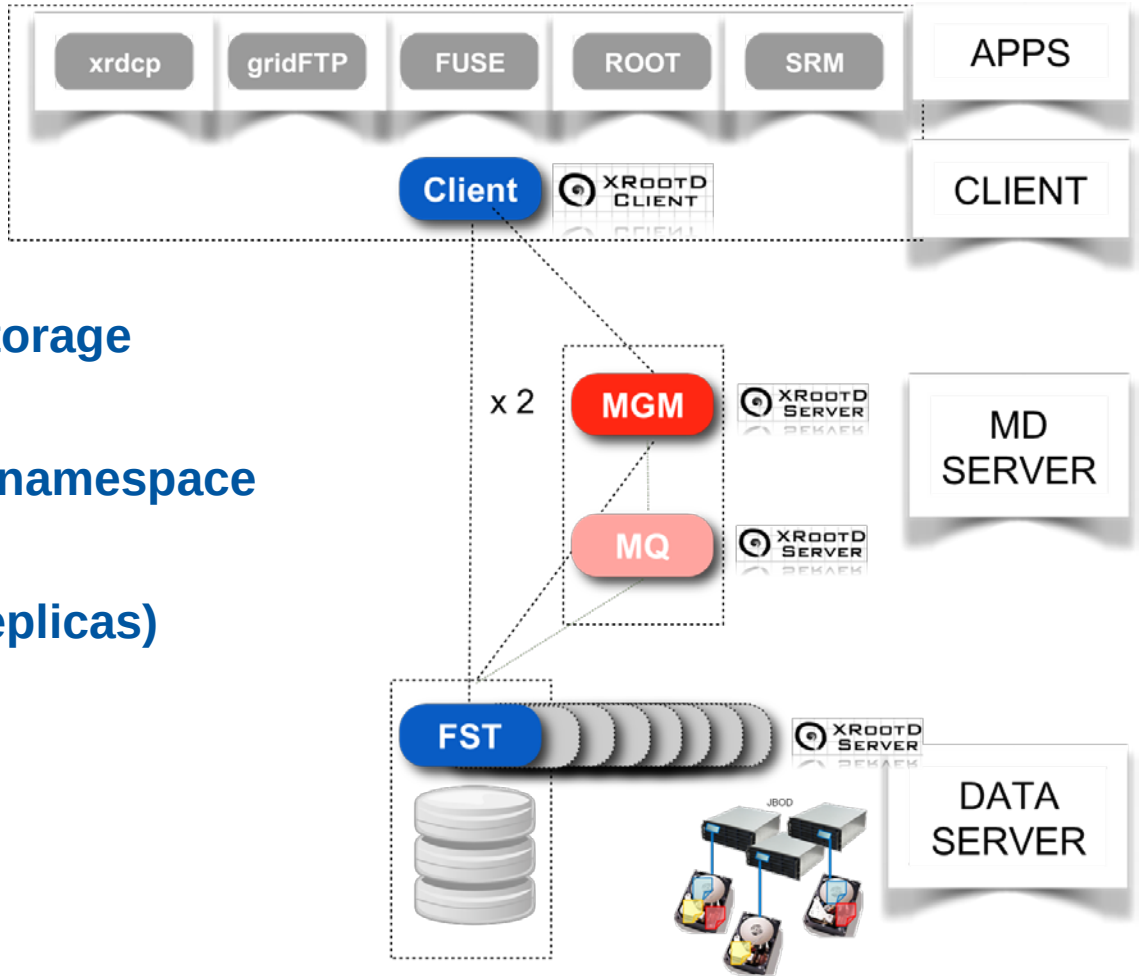
EOS namespace on top of a key-value store

Elvin Sindrilaru & Georgios Bitzes
on behalf of the **EOS team**

Outline

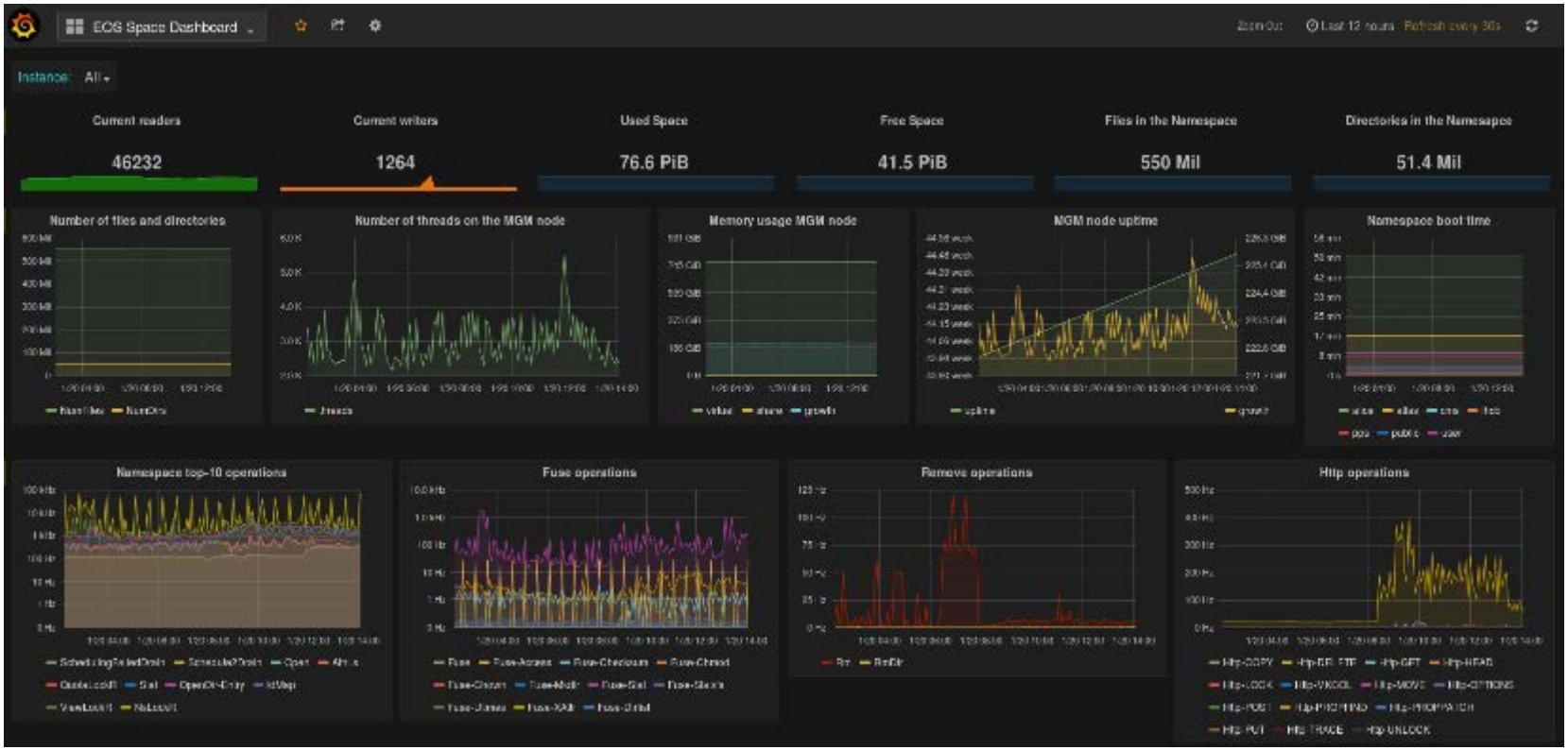
- EOS architecture
- Namespace architecture and bottlenecks
- Redis or XRootD with rocksdb as backend
- Migration tool
- Plans for the future

EOS architecture



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

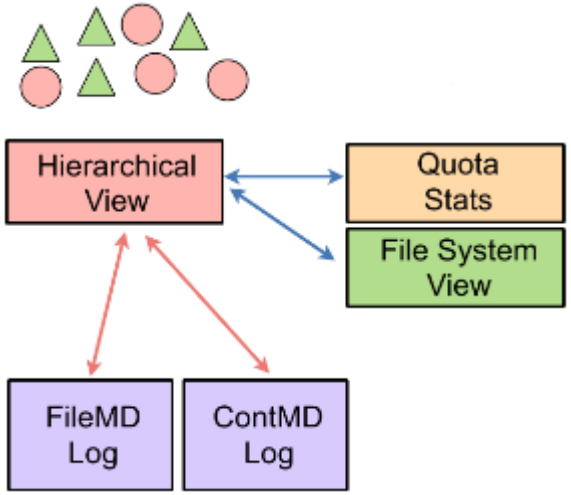
What scale are we talking about?



- Biggest instance **ALICE**
  - **~ 375M files** and ~ **100K directories**
  - Using ~**390GB** of RAM
  - Boot time **~60 min**

What is the EOS namespace?

- C++ library used by the EOS MGM node
- Provides API for dealing with hierarchical collections of files
- It's single threaded with fast in-memory operations



- **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

What's the target?

- Still fast and consistent
- Scale-out the namespace → avoid one machine's memory limitation
- Reduce the boot time i.e. service down time
  - By **partitioning the namespace** i.e. faster boot time for each partition
  - Use master – slave **replication** to ensure high availability

EOS Namespace Interface

- Prepare the setting for different namespace implementations
- Abstract a **Namespace Interface** to avoid modifying other parts of the code – keep compatibility with old implementation

- **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

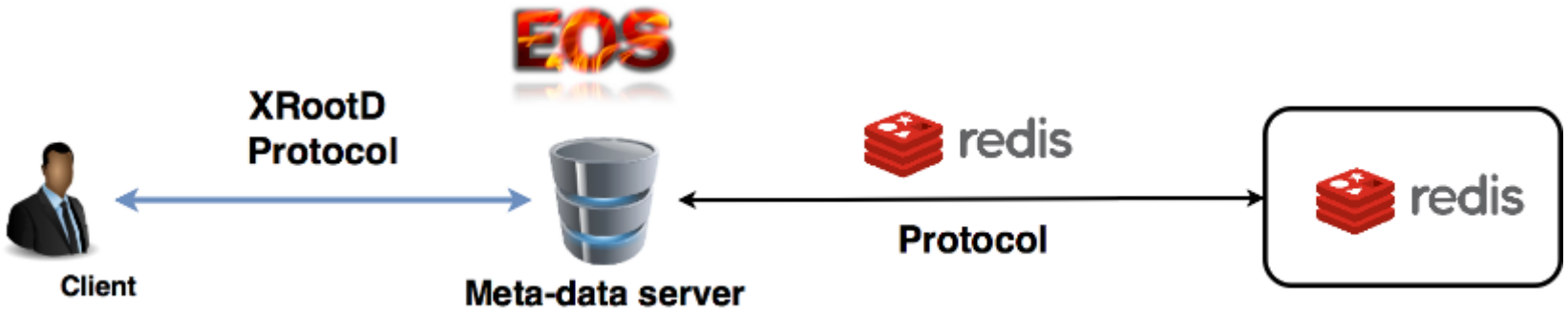
  - **libEosNsOnRados.so** – not existing but can be imagined based on CEPH

  - **libEosNsOnFilesystem.so** – not existing based on a Linux filesystem

Why Redis?

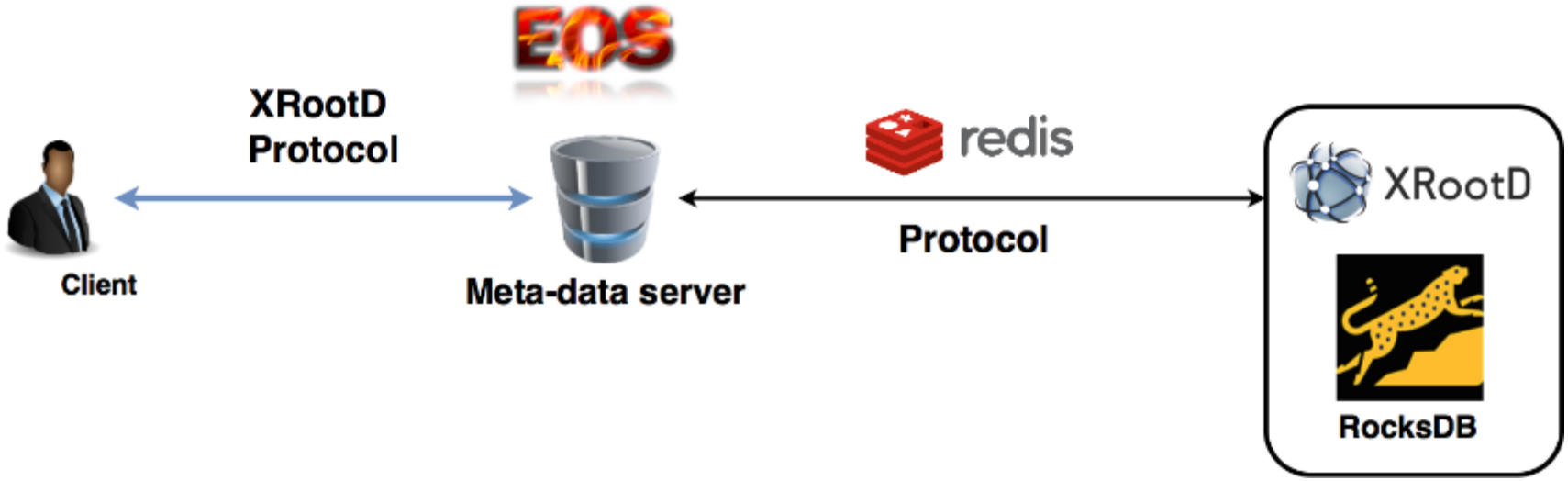
- **Redis** – in-memory **data structure store**
- Separate data from the application logic and user interface
- Supports various data structures: strings, hashes, lists, sets, sorted sets etc.

- **Light-weight EOS MGM** node that can easily be restarted or updated



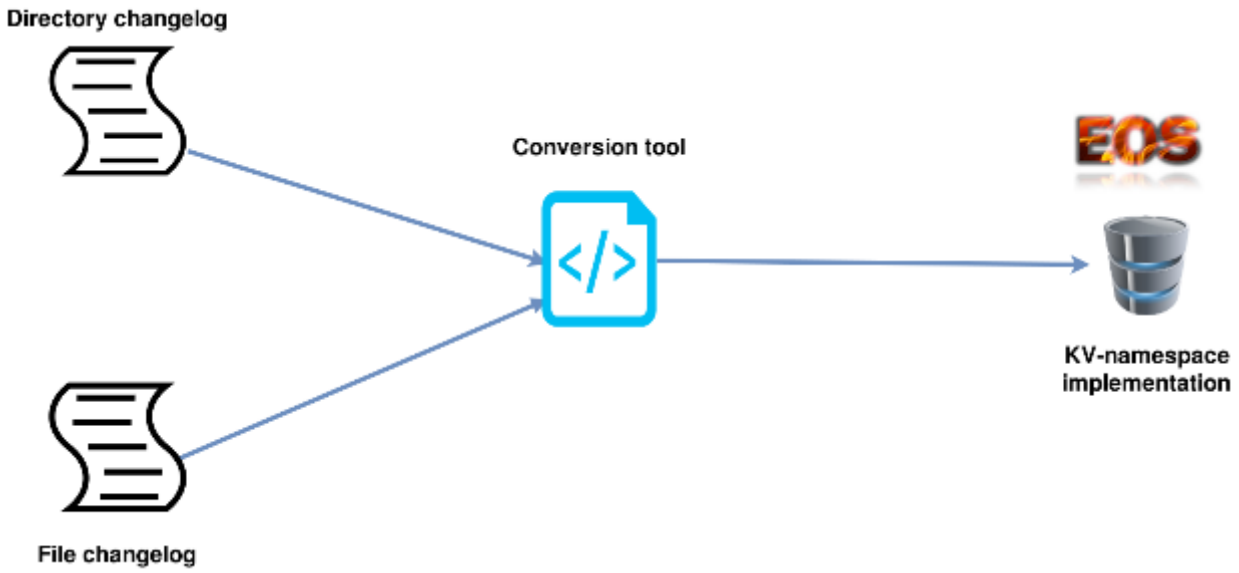
XRootD and Redis

- Replace Redis backend with XRootD
- Implemented as an XRootD **protocol plugin**
- XRootD can use **RocksDB** as persistent key-value store ➔ **QuarkDB**



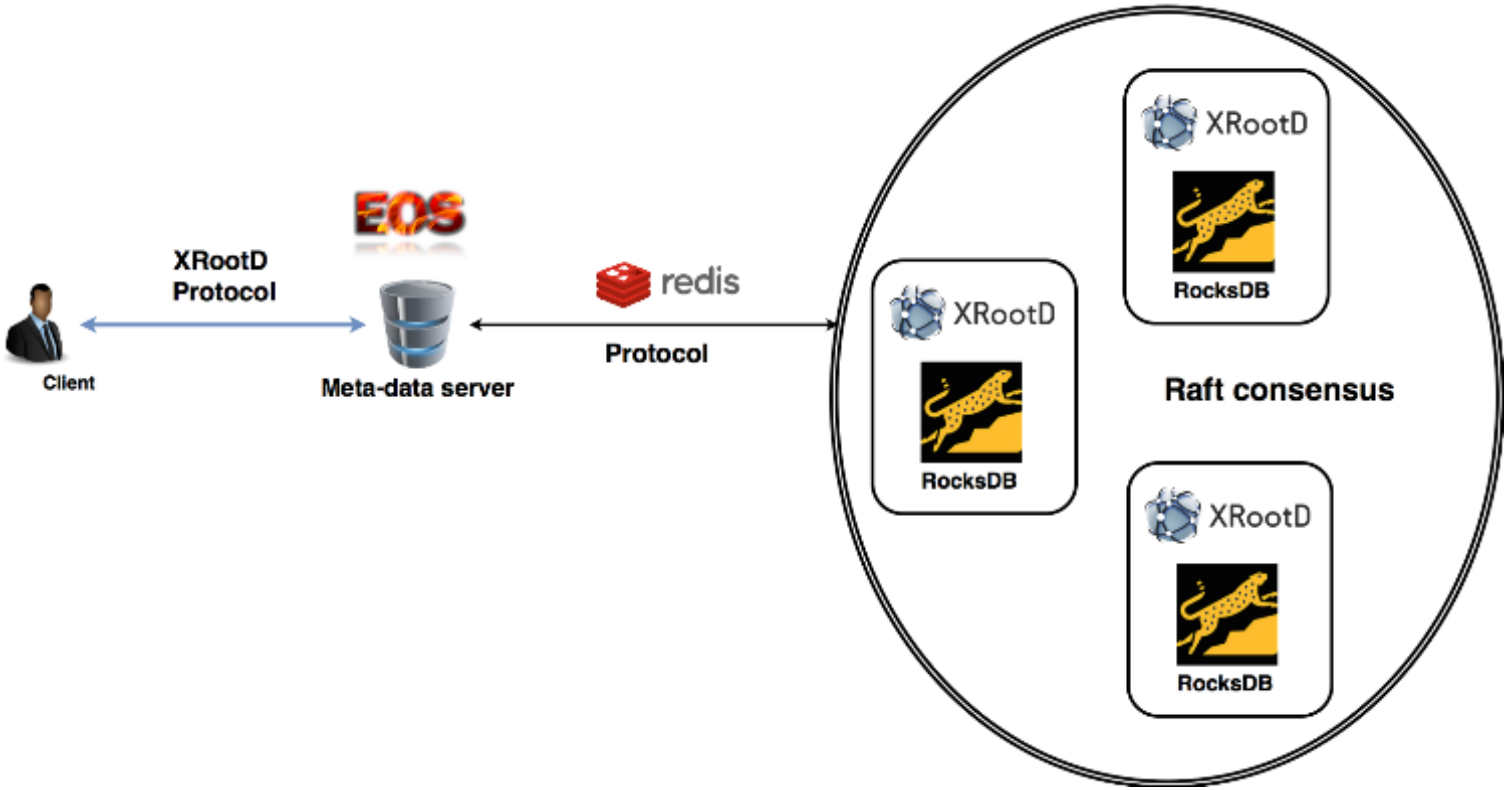
Migration procedure

- Tool to migrate from **in-memory to KV-backend** namespace
- Uses both the old and new namespace implementations
- Converting current **PPS namespace** (~10M entries)
  - **QuarkDB** - backend ~ 1.5h



Namespace HA

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



Ongoing work items

- **Evaluate performance** with QuarkDB:
  - With Redis-backend – eosnsbenchmark tool
    - **File creation rate:1.1 kHz** , ~ 4.3 RTT/entry
    - **Directory creation: 1.3 kHz**, ~ 4 RTT/entry

- Improve performance of the conversion tool

Accommodate “exotic” Views

- Used only in the **EOSUSER** instance for the **Owncloud client** sync requirements
- **Sync-time** accounting – propagation of the mtime up the hierarchical tree
- **Sub-tree accounting** – view providing sub-tree size/volume information. Think: *du –sh*
- **Plan** - use external KV-store as an add-on to provide the necessary functionality

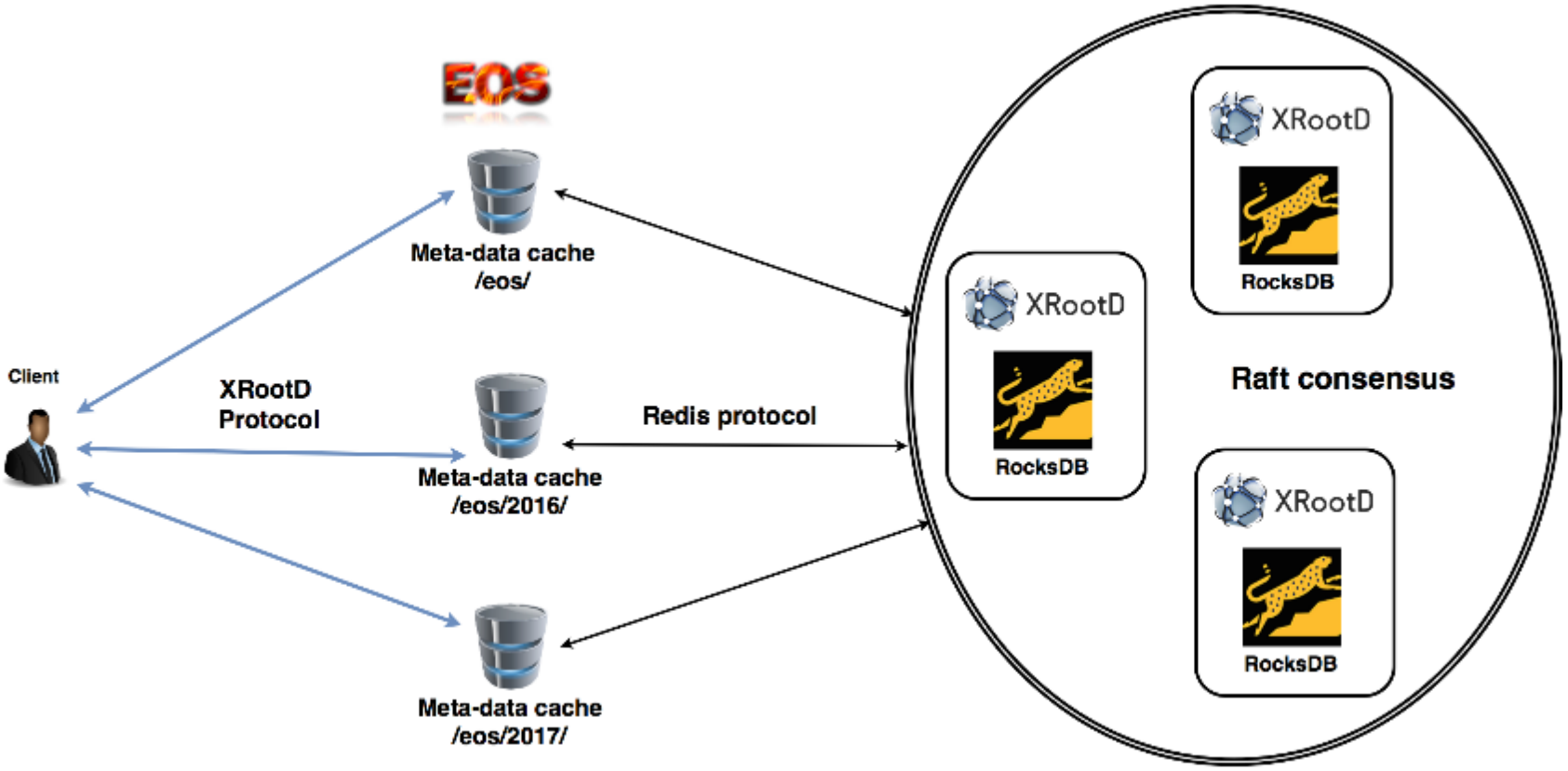
Deployment status

- **CentOS7** – hard requirement

- Deployed test cluster **PPS-citrine**:
  - 1 MGM node – 256 GB
  - 2 FST nodes - more FSTs to be moved in

- Instance using **systemd support** contributed by the **Comtrade team**

EOS NS – the future



www.cern.ch