
CERN

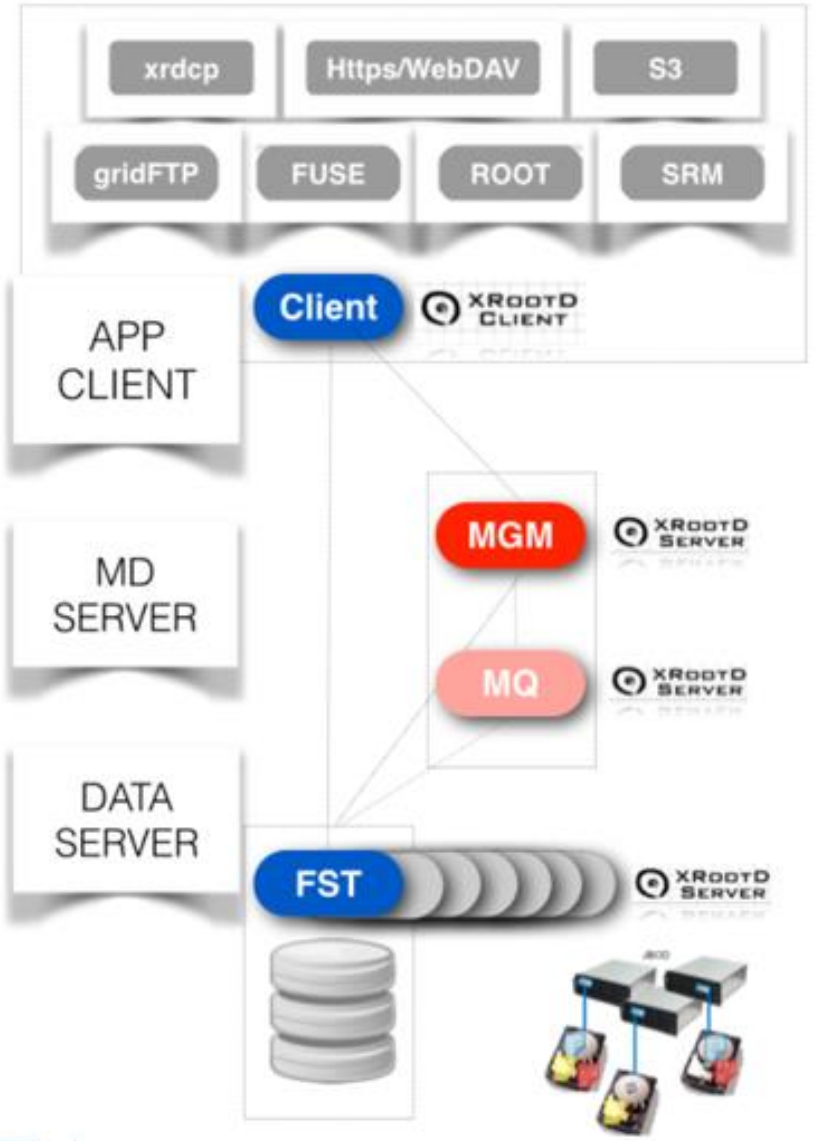

# Scaling the EOS namespace

## Quick overview, and current status

Georgios Bitzes, Elvin Sindrilaru, Andreas J. Peters, Andrea Manzi (speaker)
CERN

# EOS Architecture

- File Storage Nodes **(FST)**: Management of physical disks, file serving
- Metadata Servers **(MGM)**: Namespace + client redirection to FSTs
- Message Queue **(MQ)**: Inter-cluster communication, heartbeats, configuration changes



# The namespace subsystem

- EOS presents **one single** namespace to files it manages
  - … even though they are typically spread across **hundreds** of disk servers and **thousands** of physical disks
- Handles file permissions, metadata, quota accounting, **mapping** between logical filenames and physical locations
  - `-rw-rw-r-- 1 user group 21 Jul  2 10:02 dir1/filename`

# In-memory namespace implementation

- The MGM held the **entire** namespace in-memory. Each file / directory entry allocates up to 1kb as a C++ structure in memory.

- Linear on-disk **changelogs** to track all namespace changes
  - file additions, metadata changes, physical location migrations …
  - One for files, one for directories

- The in-memory contents are **reconstructed** on reboot by replaying the changelogs

# In-memory namespace implementation (2)

- Need for a new namespace implementation

  - **Long** boot time, proportional to the number of files on an instance. For large namespaces can exceed **1hour**

  - Requires **a lot** of RAM

# The need for high-availability

- EOS has become critical for data at CERN. MGM loss means *long downtime*, great disruption.

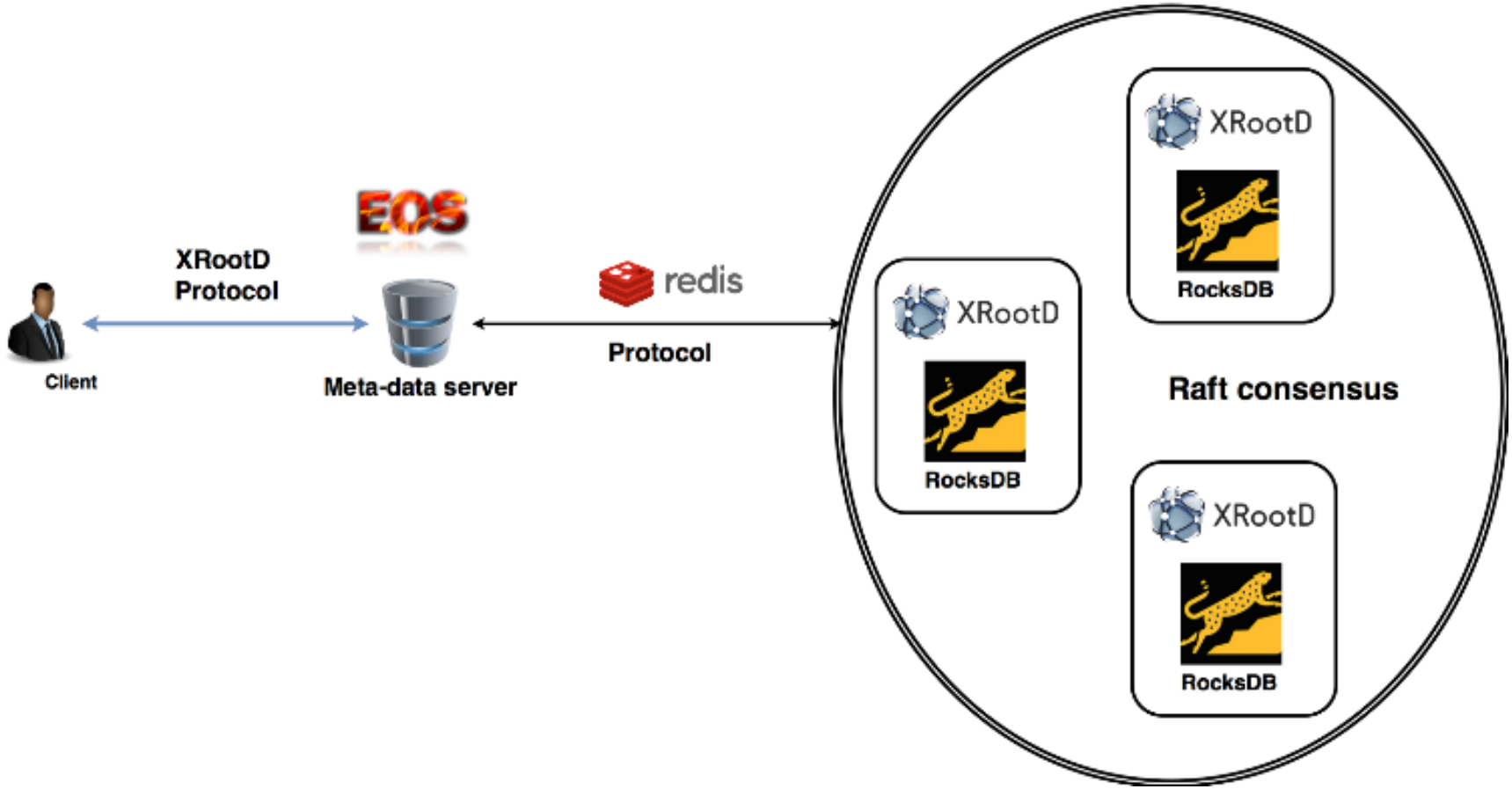
- Ideally:
  - Transparent failover, *no service interruption*
  - No single point of failure

# Architectural evolution

- We've designed and implemented **QuarkDB**, a highly available datastore for the namespace.
  - **Redis protocol**, supports a small subset of Redis commands.
  - **RocksDB** as the underlying storage backend.
  - High availability: **Raft consensus** algorithm.

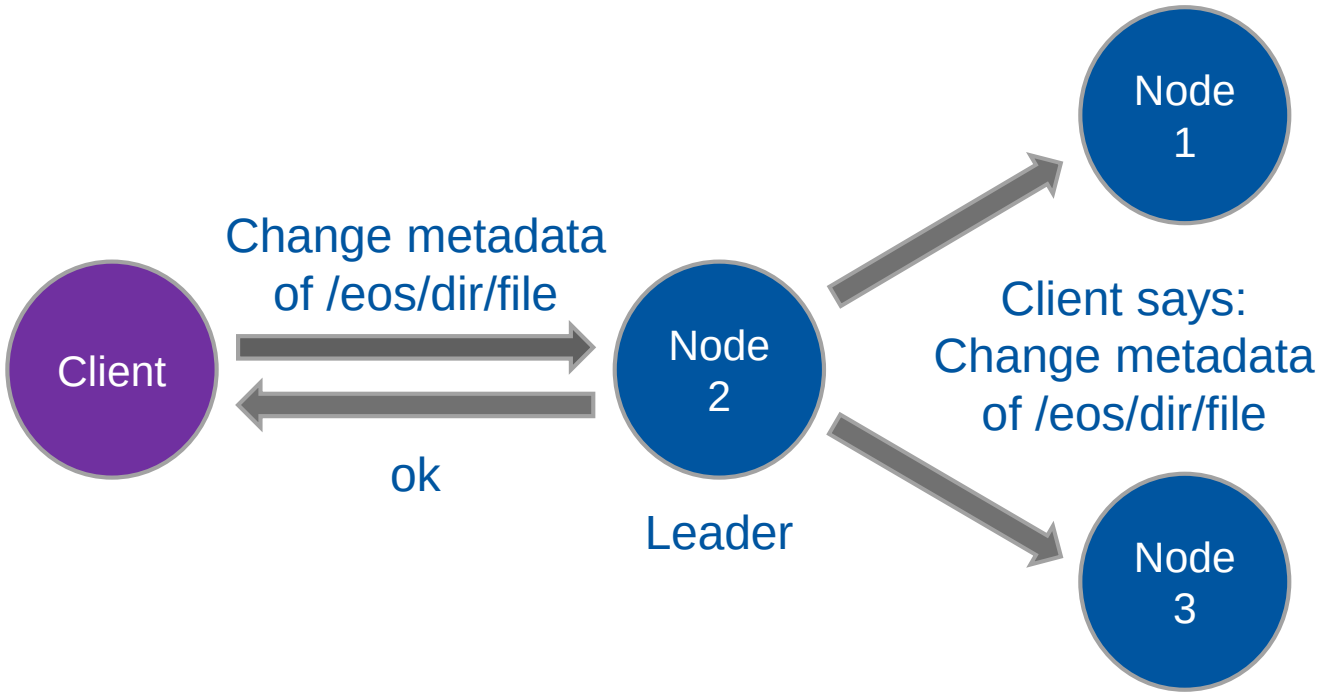
- Implement the minimum necessary, and keep the system simple
  - **QuarkDB** runs as a plug-in to the **XRootD** server framework used by EOS

# Architectural evolution(2)

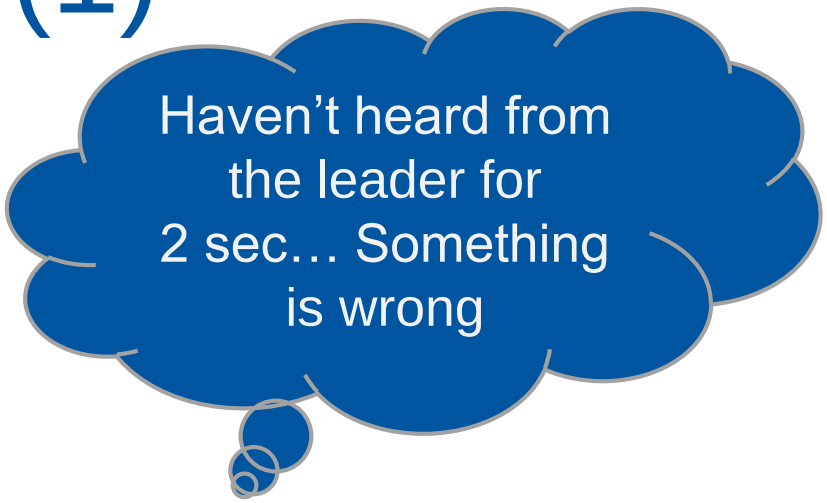


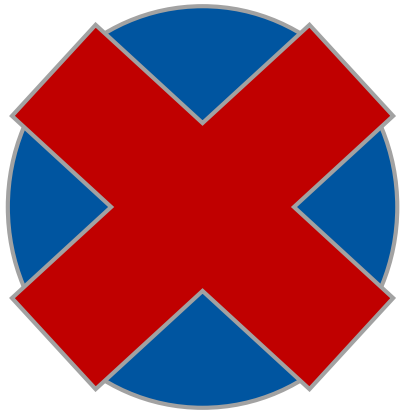
# Replication

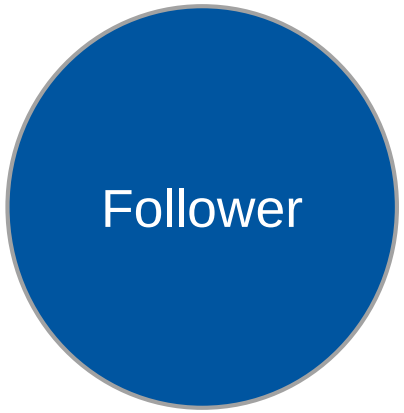
One of the nodes is elected to become the *master* (or *leader*)



# Leader election (1)

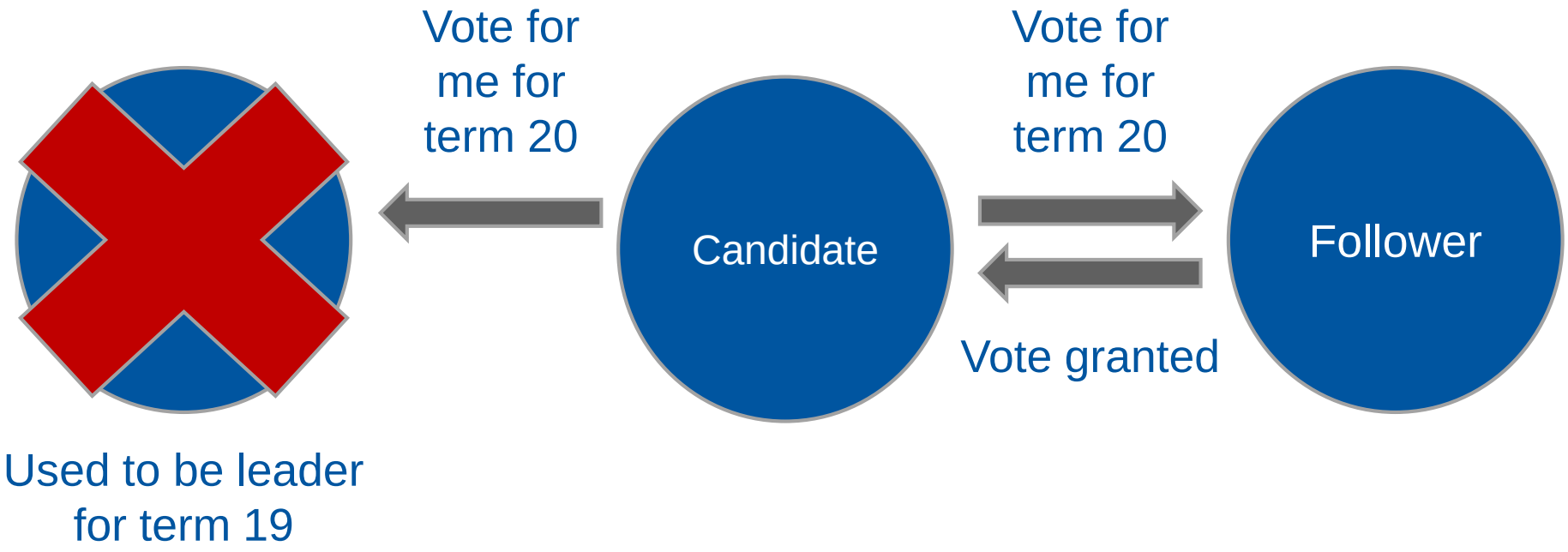


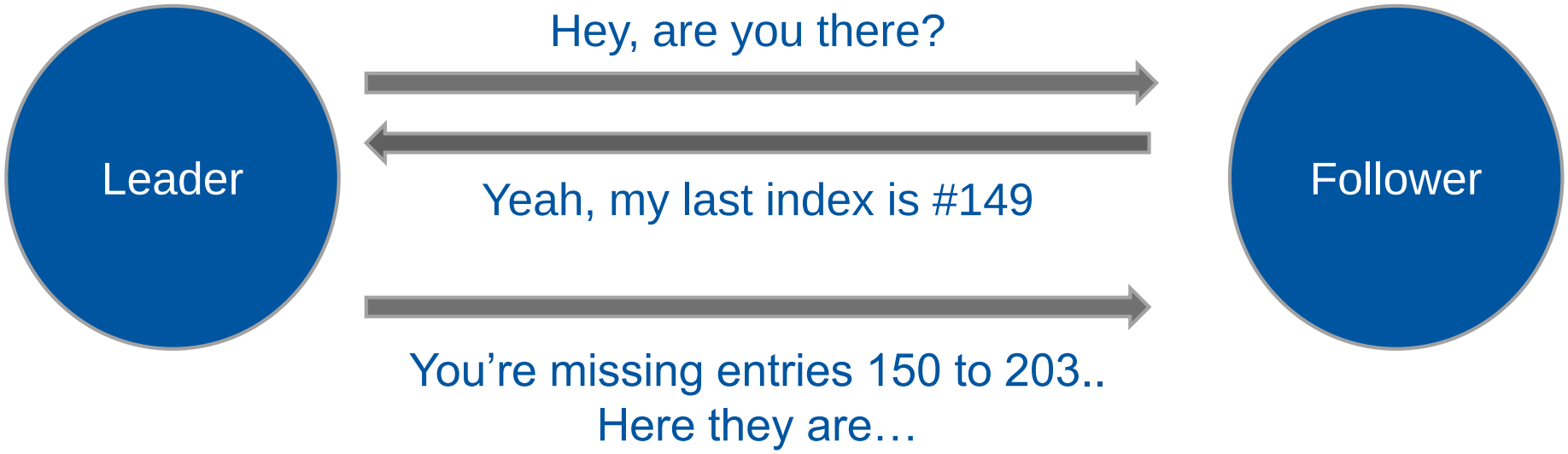


# Leader election (2)

A successful election: 2 out of 3 nodes agree on the new leader



# Log replication



# QuarkDB Testing

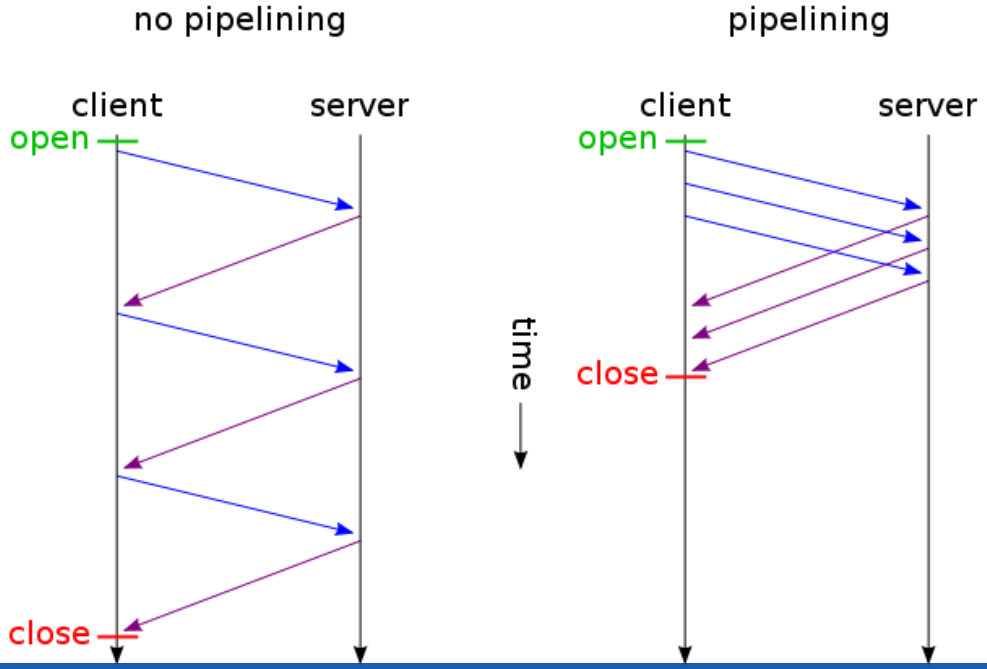
- QuarkDB is being tested extensively.
  - Unit, stress, chaos tests: From testing parsing utility functions, to simulating constant leader crashes and ensuring nodes stay in sync.
  - Test coverage: **91%**, measured on each commit.
  - All tests running under AddressSanitizer & ThreadSanitizer, on each commit.

# Problem: Network latency

- Previous assumption throughout the MGM code: Access latency to the NS is *minimal*, all lives in-memory.
- But with QDB, metadata lives a network roundtrip away…
  - Caching frequently accessed entries in the MGM helps a lot.
- Certain **locks** which were fine to hold for in-memory NS operations, were causing trouble for new NS.

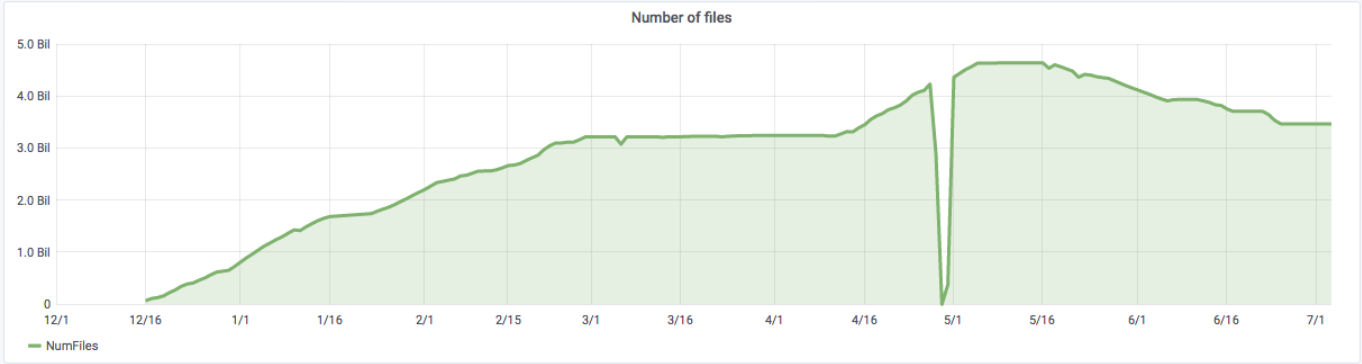
# Problem: Network latency (2)

- Using prefetching to first load metadata into the MGM cache, **before** holding any locks.
  - Allows staging in-flight requests from many clients simultaneously.
  - Allows pipelining, virtually eliminating effects of network latency for many operations
    (notably “ls”)



# Current status

- **EOSPPS**: our pre-production instance runs NS on QuarkDB since 7 months
  - **4.6 billion files reached**– larger namespace than all other instances combined
  - **Boot time:** A couple of minutes



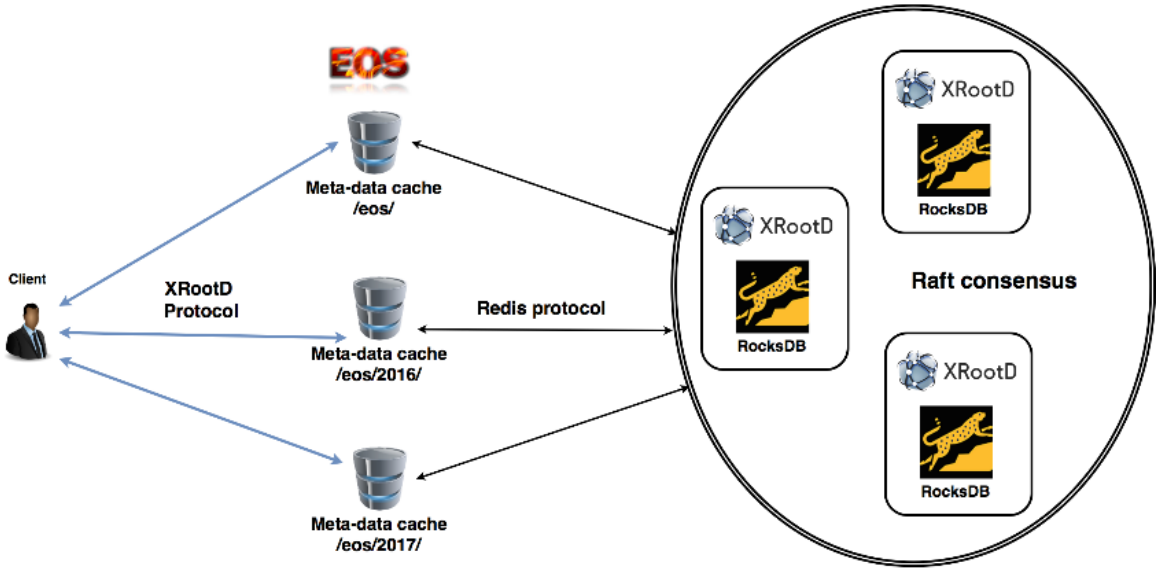
# Current status (2)

- Some more numbers:
  - Namespace size on disk: **~0.6 TB**
  - QuarkDB has been able to sustain **7-11kHz** of writes for weeks, translates to around **~1kHz** file creations.

- First **production** instance under deployment for the EOS HOME project

# Next Steps

- Implementing **HA** at the **MGM** level. Multiple standby MGMs, coordination through QuarkDB on who becomes active



# Thanks

- https://gitlab.cern.ch/eos/quarkdb
- Current status: **~18k** lines of code
  - including tests, tools
  - excluding dependencies

- More on Raft:
  - https://raft.github.io/raft.pdf
  - https://thesecretlivesofdata.com/raft/

Questions, comments?

# Backup Slides

# Possible alternatives

- **RDBMs:** scalability issues, complicates our setup


- **Redis:** scalable and fast, but…
  - high per-entry RAM overhead
  - redis cluster can lose acknowledged writes

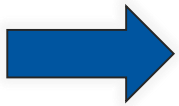
- **Cassandra:** scales very well, but…
  - adds significant complexity to setup & operation
  - performance / resource cost is high

# Redis command translation

**Redis command**

**rocksdb**

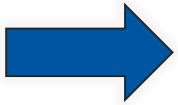
HSET myhash field contents



Key descriptor: “**dmyhash**” =>
“**This key is a hash, current size is 5**”

“**bmyhash**##**field**” => “**contents**”

SADD myset element



Key descriptor: “**dmyset**” =>
“**This key is a set, current size is 8**”

“**cmyset**##**element**” => “**1**”

# Consistency guarantees

- QuarkDB is a *strongly consistent* datastore (CP from CAP theorem)

- **Linearizability:** once a client receives an ACK to a write, all future reads (from *any* client) are guaranteed to return *that value, or a future one*.
  - *even if the leader crashes right after the ACK*