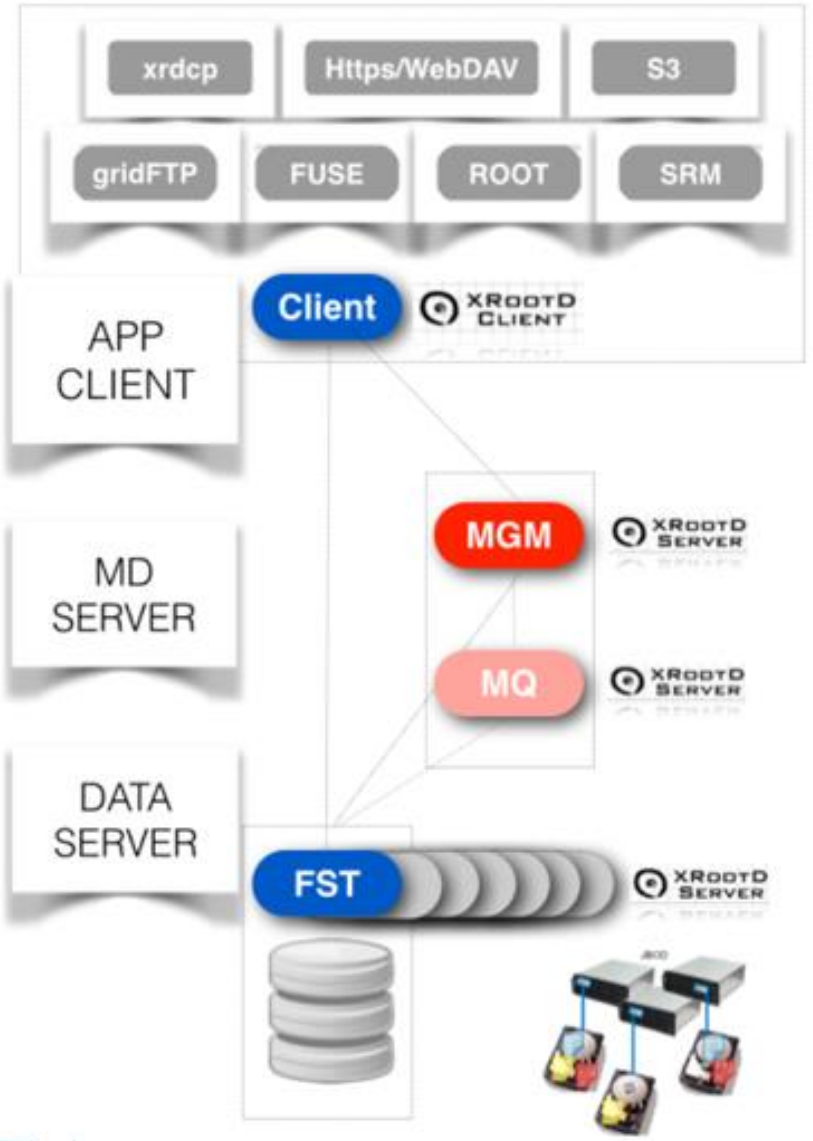
The new namespace and QuarkDB

An overview, and current status

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

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

Sample file entry

Logical filename: /eos/somedir/filename

Inode number: 134563

Parent directory inode: 1234, referring to /eos/somedir

Size: 19183 bytes

File layout: 2 replicas

- **Physical replica 1:** Filesystem #23, referring to fst-1.cern.ch:/mnt34/
- **Physical replica 2:** Filesystem #45, referring to fst-2.cern.ch:/mnt11/

Checksum: md5-567c100888518c1163b3462993de7d47

In-memory namespace implementation

- The MGM always holds 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

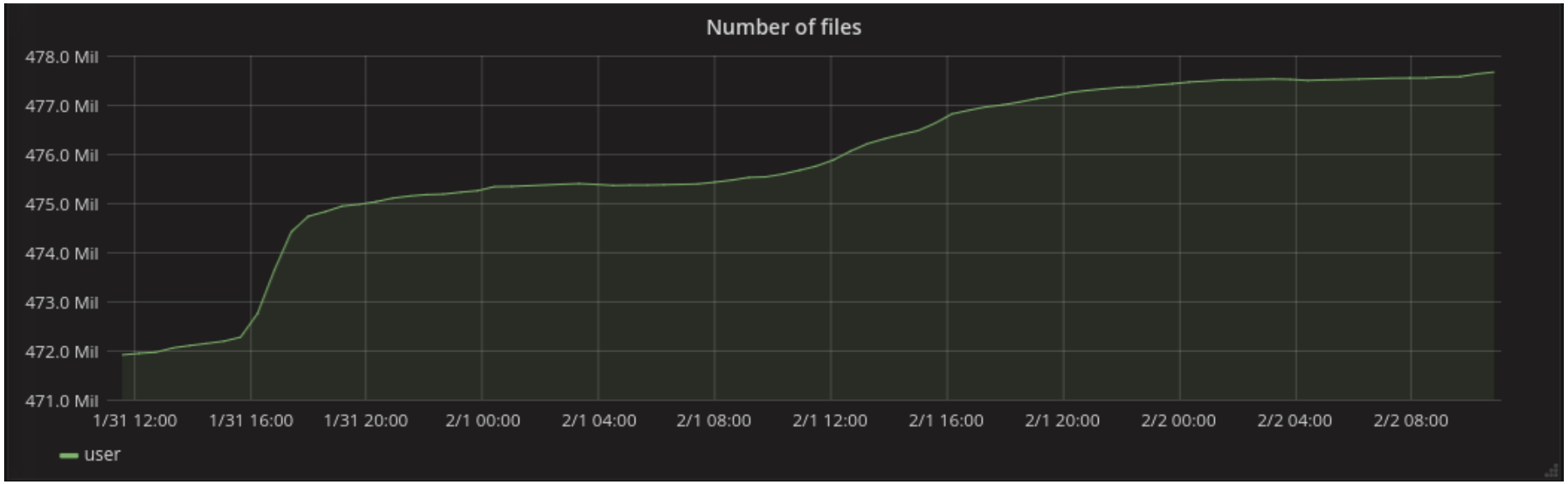
In-memory namespace implementation (2)

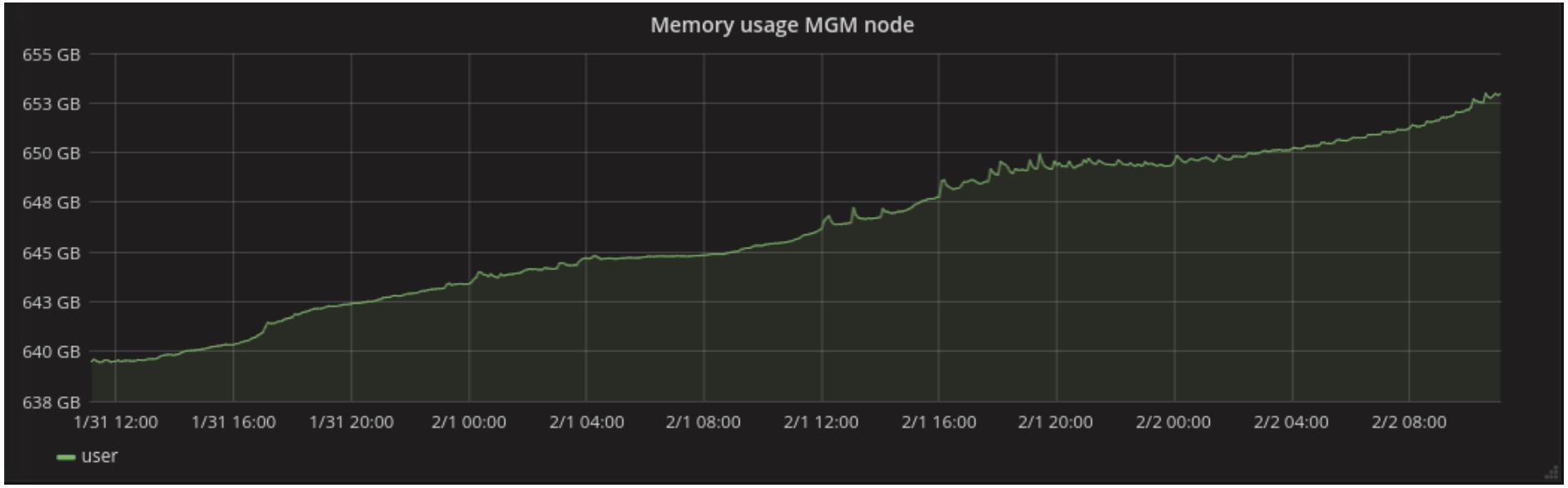
- The good
  - It's **fast**. Lookups done with a hashmap, no I/O.
  - Reliable and **proven**.

- The bad

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

We’re running out of time: **eosuser**



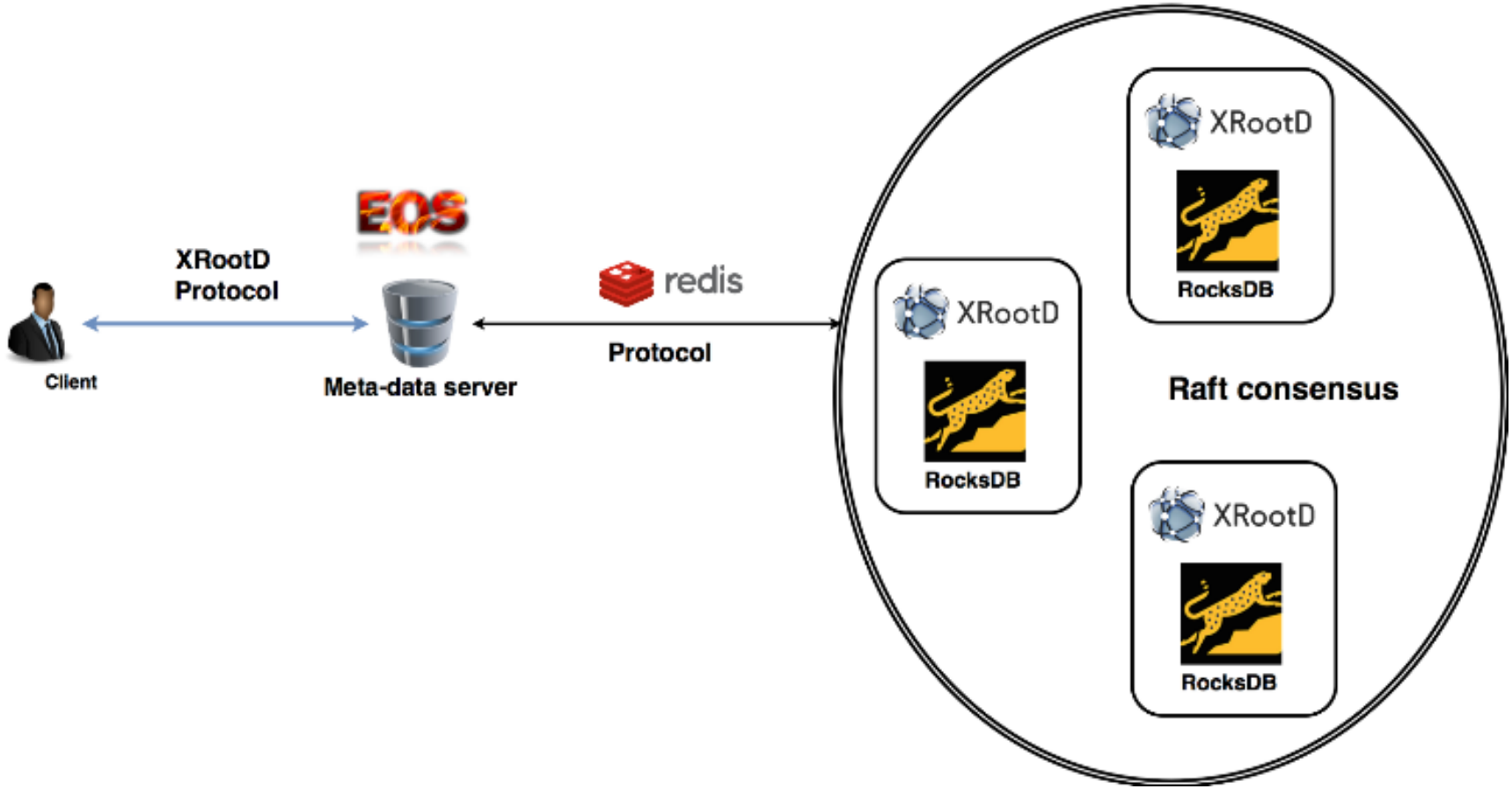


The need for high-availability

- 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



Architectural evolution (2)

- 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

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

Consistency guarantees

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

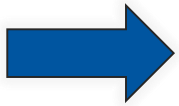
- **Linearizability:** once a client receives an ACK to a write, all future reads (from *any* client) are guaranteed to return *that value, or a future one*.
  - *even if the leader crashes right after the ACK*

Redis command translation

Redis command

rocksdb

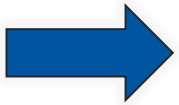
HSET myhash field contents



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

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

SADD myset element

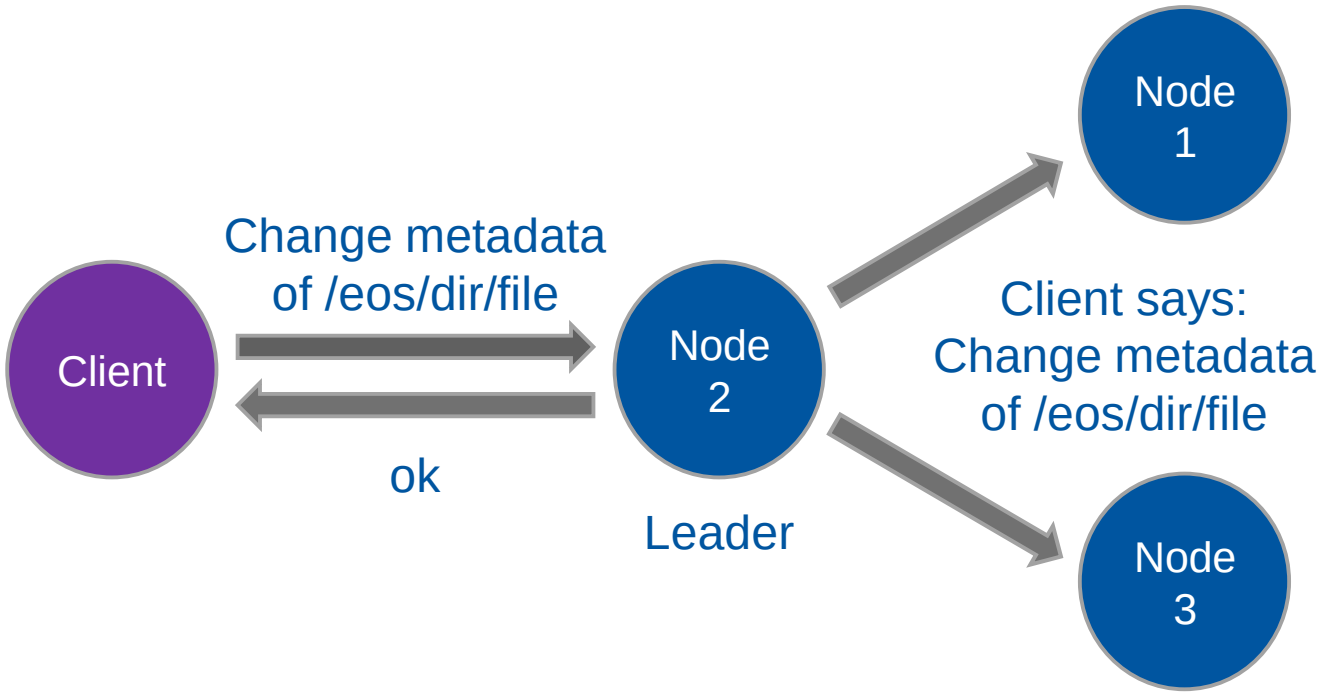


Key descriptor: “**dmyset**” =>
“**This key is a set, current size is 8**”

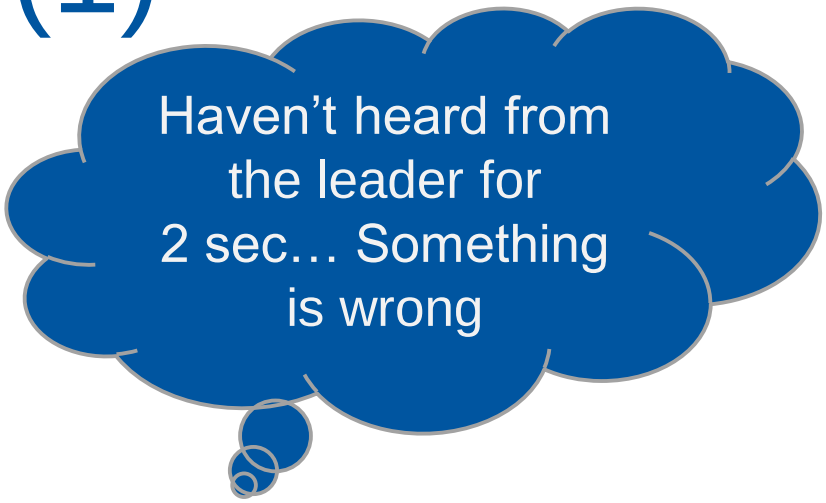
“**cmyset**##**element**” => “**1**”

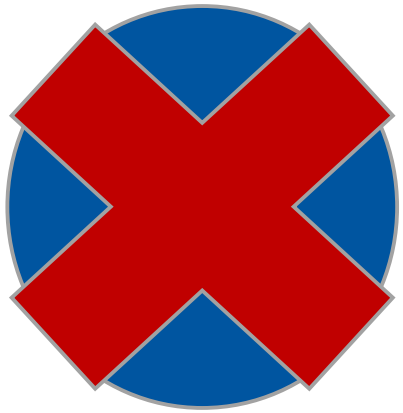
Replication

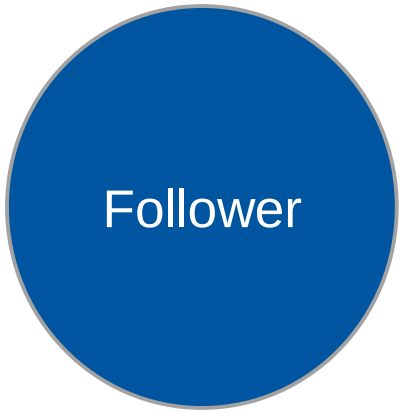
One of the nodes is elected to become the *master* (or *leader*)



Leader election (1)

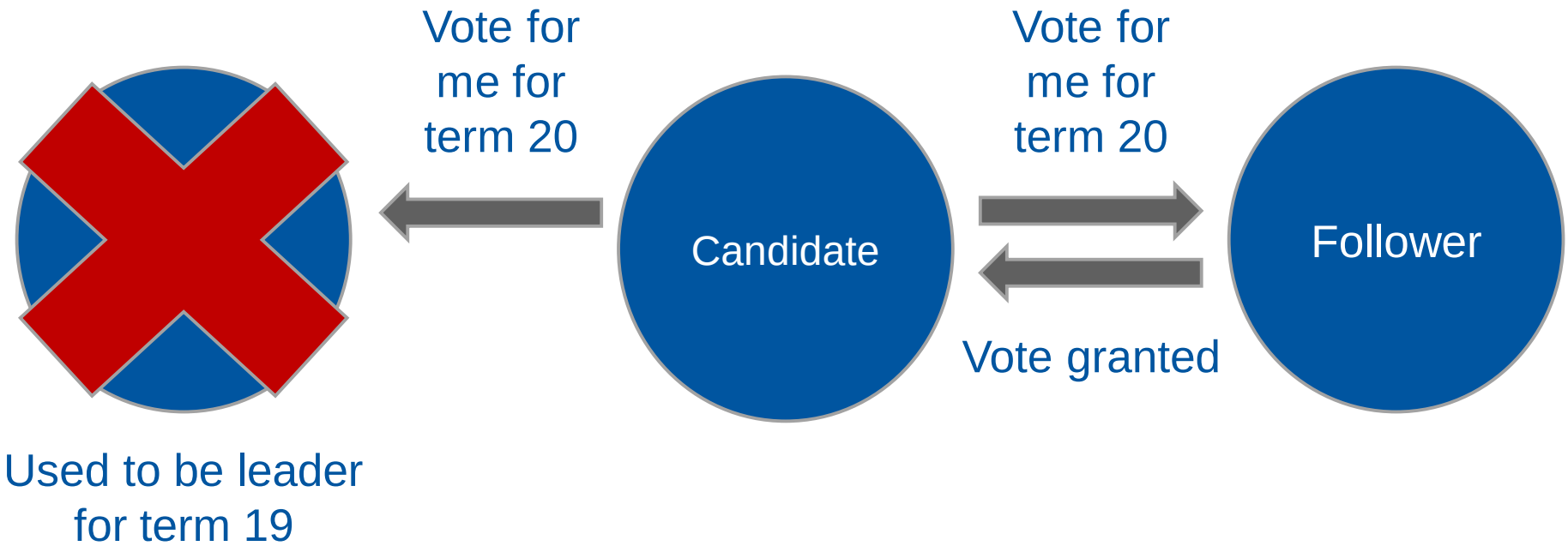




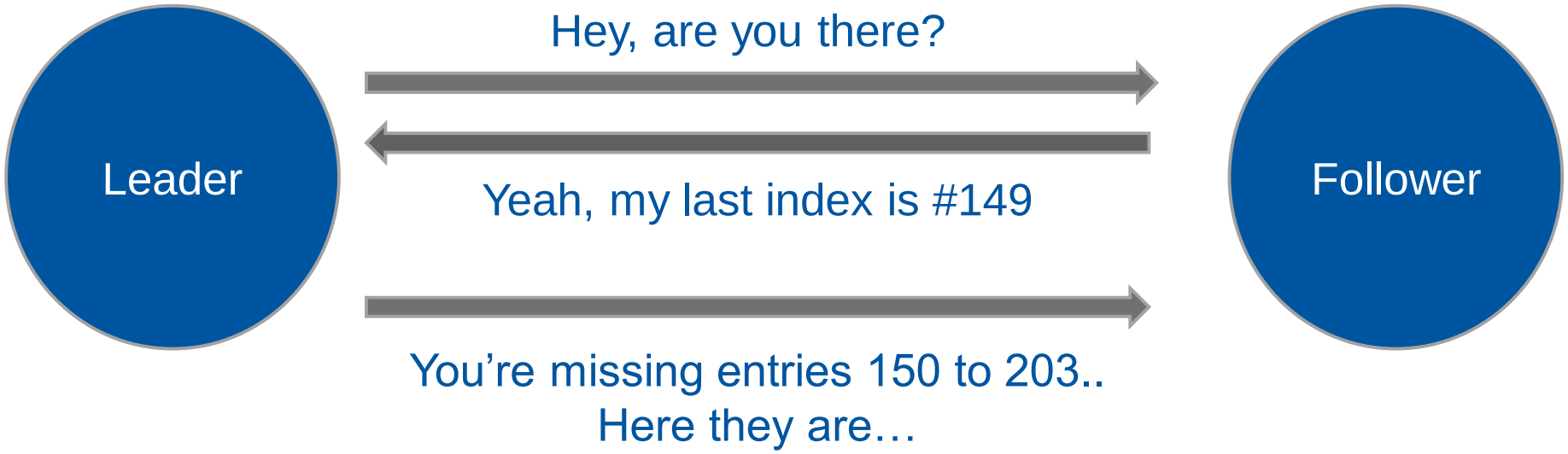


Leader election (2)

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



Log replication



QuarkDB Testing

- Consensus bugs would be horrible to trace, it *has* to be correct.

- QuarkDB is being tested without mercy.
  - Unit, stress, chaos tests: From testing parsing utility functions, to simulating constant leader crashes and ensuring nodes stay in sync.
  - Test coverage: **90%**, measured on each commit.

  - All tests running under AddressSanitizer & ThreadSanitizer, on each commit.

    Praise be to our Lord and Savior, ThreadSanitizer

Current status

- **EOSPPS**: our pre-production instance runs NS on QuarkDB since a few months
- **2.3 billion files** (most are empty) – 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.

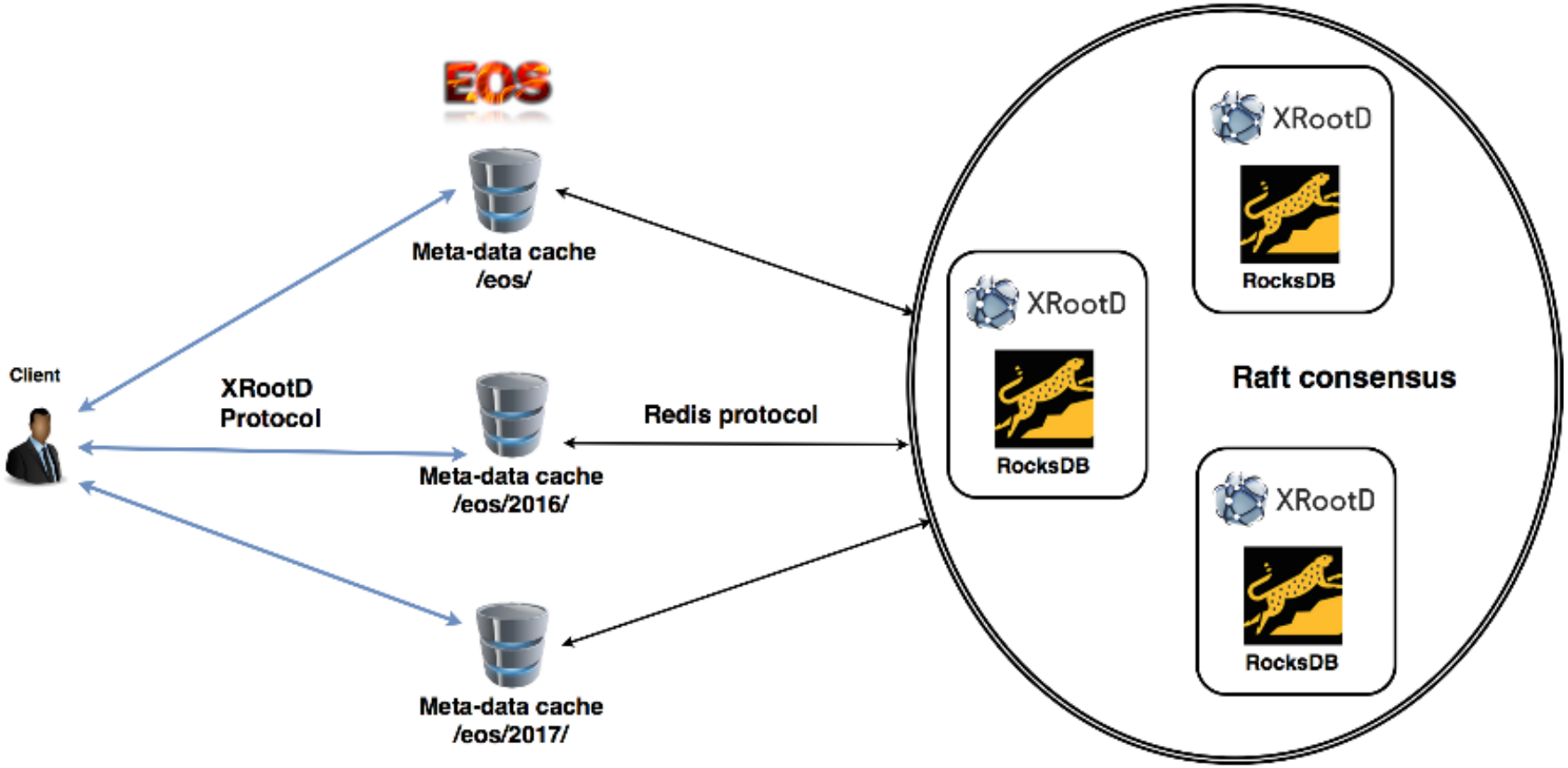
- A few remaining things to fix, for example:
  - Find command should bypass the cache to avoid thrashing it, talk to QDB directly.

The future

- Ideally:
  - High availability.
  - No single points of failure, whatsoever.
  - Transparent failover, self-healing.
  - Infinite scalability.

- We’re not there yet – making steps in that direction.
  - Starting with QuarkDB and the namespace.

The future (2)


EOS
Meta-data cache
/eos/
Client
XRootD
Protocol
Meta-data cache
/eos/2016/
Redis protocol
Meta-data cache
/eos/2017/
XRootD
RocksDB
XRootD
RocksDB
Raft consensus
XRootD
RocksDB

Thanks

- https://gitlab.cern.ch/eos/quarkdb
- Current status: **~15k** lines of code
  - including tests, tools
  - excluding dependencies

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

Questions, comments?