STORAGE DEVELOPER CONFERENCE
SDC 21
BY Developers FOR Developers
Virtual Conference
September 28-29, 2021
A SNIA Event

# Ozone - Architecture and Performance at Billions' Scale

Lokesh Jain
Software Engineer, Cloudera

# About Me

- Senior Software Engineer, Cloudera
- PMC and committer for Apache Ozone, Apache Ratis and Apache Hadoop
- Contributing for past 4 years

- Introduction
- Architecture
- Performance

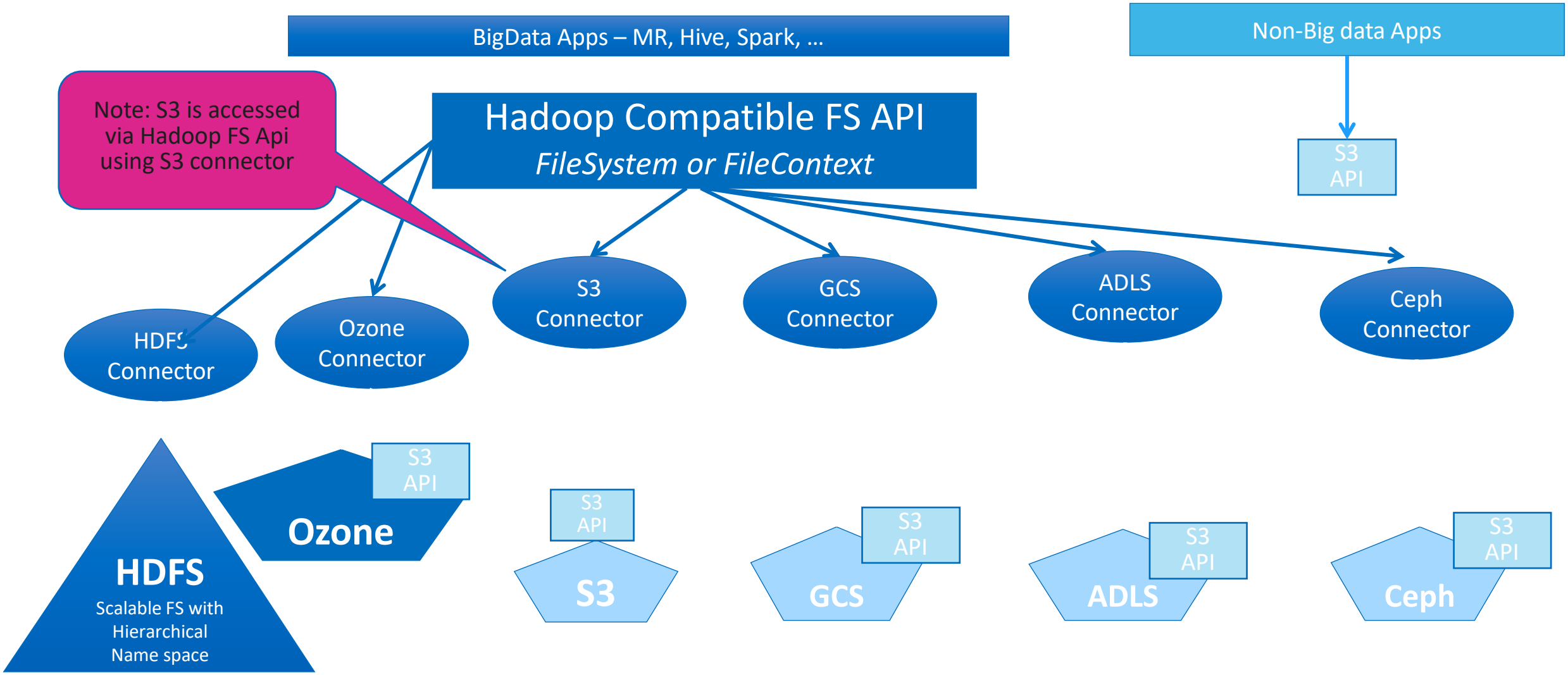
# Ozone

- Distributed Object Store – Volumes, Buckets, Keys
- Object Store, Filesystem and S3 API
- Started as sub project in Hadoop, currently a top level project in Apache

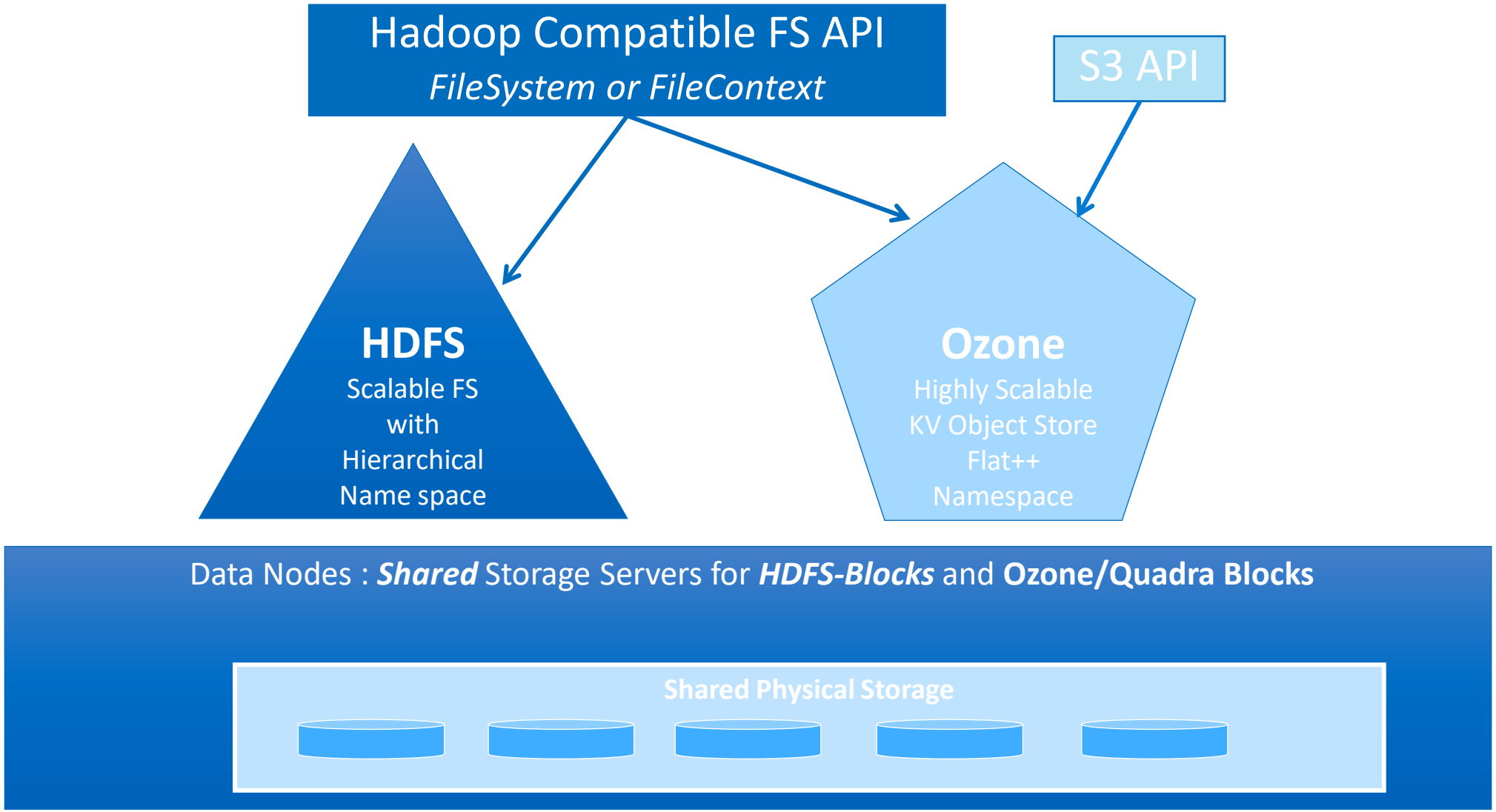
- Introduction
- Architecture
- Performance

# Understanding the Hadoop FS Application API



# HDFS & Ozone – Can Share Storage Servers and Physical Storage

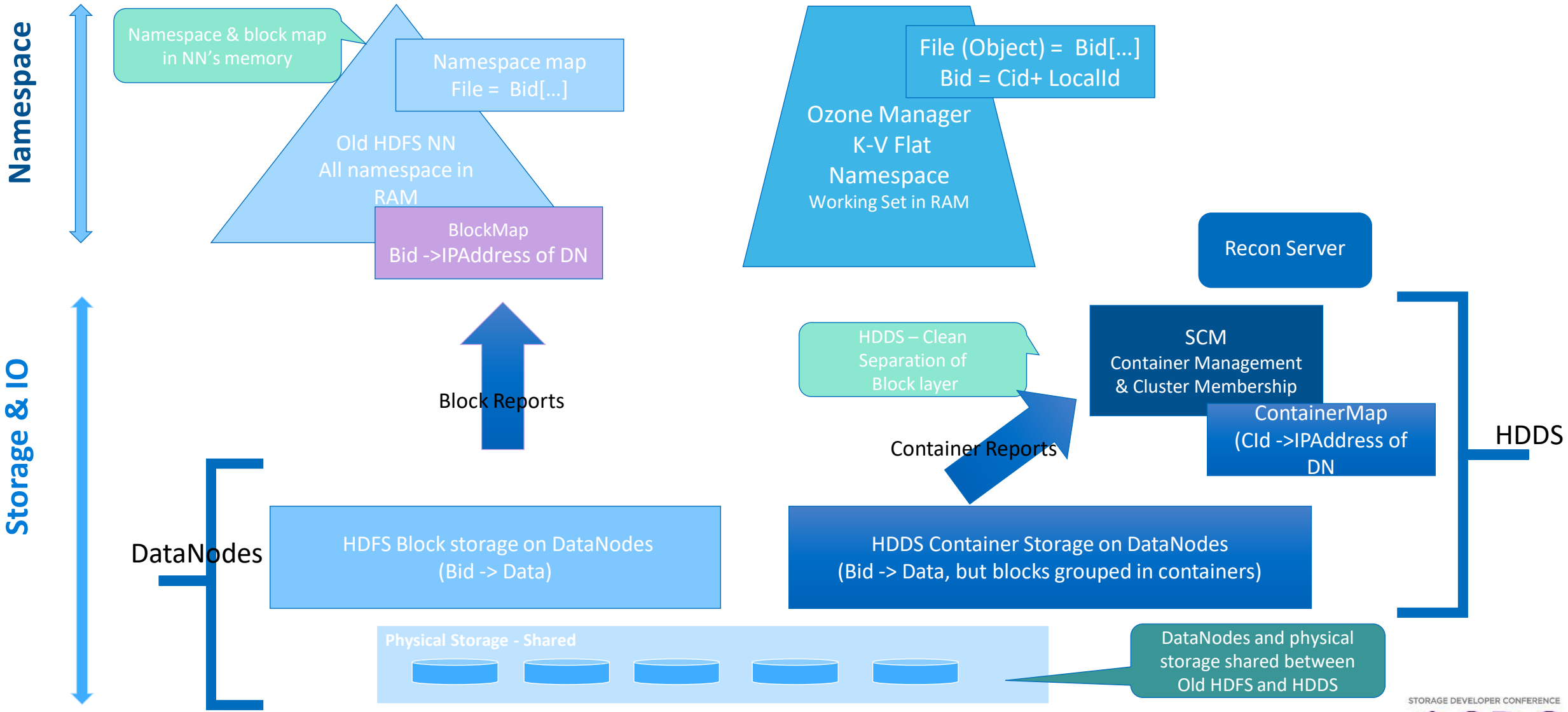


# How it all Fits Together

***Existing HDFS***

***New***



# Ozone – what does it offer?

## Carries forward the best of HDFS

- Horizontal scalability of PB and IO
- Fault tolerance storage layer
- Storage & Compute can scale independently
  - Supports locality if needed
- Strong Hadoop Security
- Rolling upgrade
- Basic ACLs plus Ranger plugability

## Improves over HDFS

- File, block and client/rpc **scalability**
  - Architecture/Design – 1 Trillion objects
  - Current implementations design:
    - 10 Billion objects
- **DN density**
- **Manageability**
  - Heap tied to working-set, not # of files
  - Faster startup
  - Master and DNs can co-locate if needed
    - good for small or embedded systems

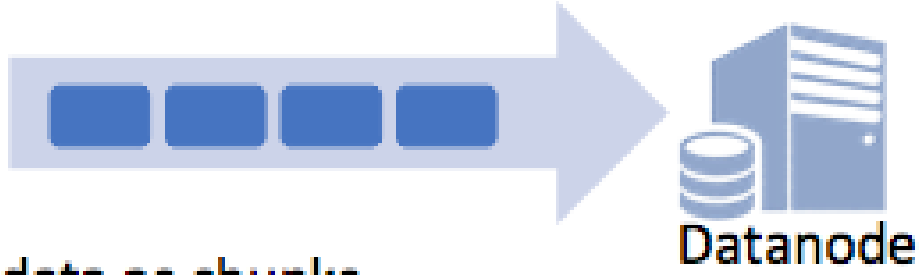
# Design Tenets

- Strong Consistency
- Simple Architecture (really!)
- Operational Ease
- Use proven building blocks (Raft, RocksDB, Hadoop security)
  - Don’t reinvent the wheel
- 100% open source and part of Apache Hadoop since day 1
  - Design and development in open source

# Ozone Write a Key

1. Client → PutKey → Ozone Manager → Allocate Block /Container/Pipeline → SCM

2. Client →  → Datanode

- Writes data as chunks
- Update metadata of a block

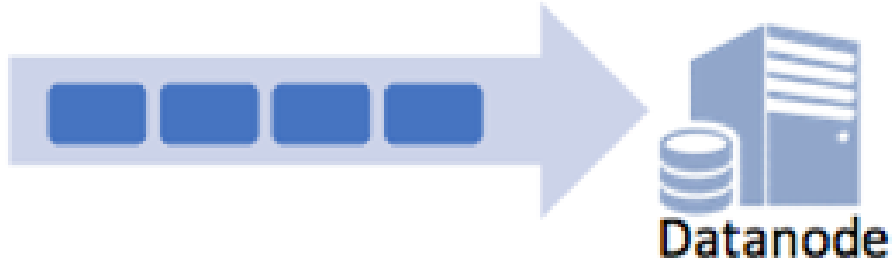
| Block | Value |
|---|---|
| Block 001 | { List of Chunks } |
| Block 002 | { List of Chunks } |

3. Client → Commit Key → Ozone Manager

# Ozone Read a Key

1. Client → GetKey → Ozone Manager
   Ozone Manager → KeyLocationInfos → Client
2. Client → Datanode




| Block | Value |
|---|---|
| Block 001 | { List of Chunks } |
| Block 002 | { List of Chunks } |

- Read data blocks as chunks

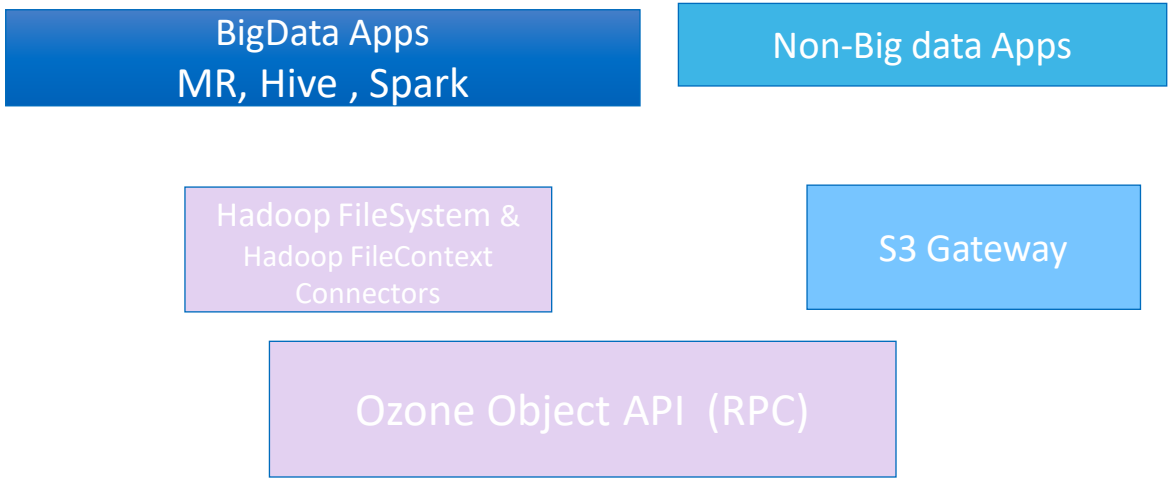
# Details of the Namespace Layer

# High Level Concepts & API

- Name (Key)*: /Volume/bucket/dir1/dir2/*
- Volumes - **Unit of management, admin**
  - E.g. /home, /users, /tmp, /data-sales, /data-marketing
  - Atomic directory renames
- Ozone is Consistent
- Renames only within bucket
- Encryption at the bucket-level
- ACLs at Volume, bucket, dirs.
  - Ranger prefix-based policies.

- Two APIs:
  - Hadoop File system API
  - S3 API



# Ozone Manager (OM)

- Keeps working set in memory
  - Need enough memory for working set
  - SSD – cache miss latency
- Uses a local RocksDB to store the namespace
- Replicated using RAFT
- Can shard at bucket or volume in future

# HDDS – The Storage layer

# HDDS

- **HDDS layer : *Strictly* Separate from namespace layer**

# Key High-Level Concepts

**Container: set of blocks (5GB)**

- Replicated as a group (using Raft)
- Each Container has a unique ContainerId
  - Every block within a container has a local id
    - BlockId = ContainerId, LocalId

**SCM – Storage Container manager**

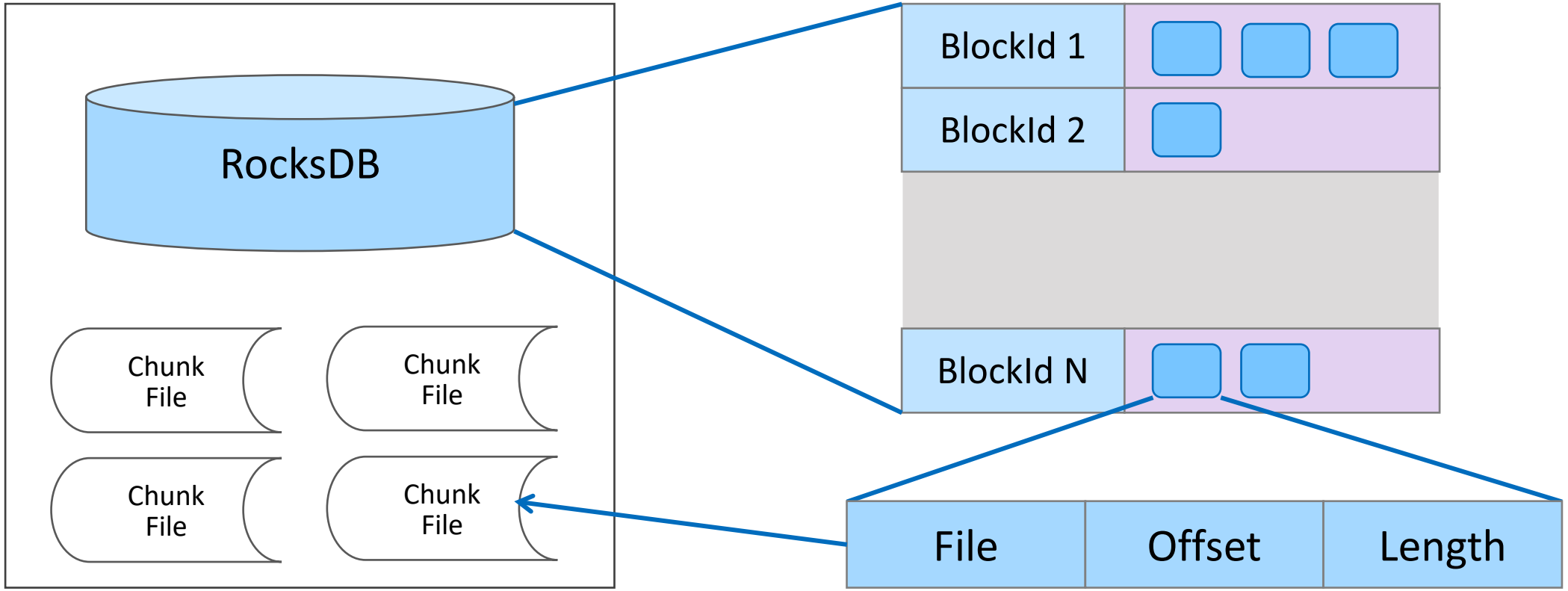
- Cluster membership
- Receives container reports from DNs
- Manages container replication
- Maintained Container Map (Cid->IPAddr)

**Data Nodes – HDFS & HDDS can share DNs**

- DNs contain a set of containers
  - just like DNs used to contain blocks
- DNs send Container-reports to SCM
  - like block reports

# Structure of a Storage Container



# Container Structure (Using RocksDB)

- An embedded LSM/KVStore (RocksDB)
- Block Id is the key.
  - Filename of local chunk file is value
- Optimizations
  - Small blocks (< 1MB) can be stored directly in RocksDB
  - Compaction for block data to avoid lots of files (Future improvement)

# Open and Closed Containers

**Closed** – when full or hit a failure in the past

- Why close a container on failures?
  - We originally considered keeping it open and bringing in a new DN –wait for new replica to catch up.
  - However closure on failure gives immutability. Easier to reason about failure recovery.
  - Can re-open later or can merge with other closed containers.

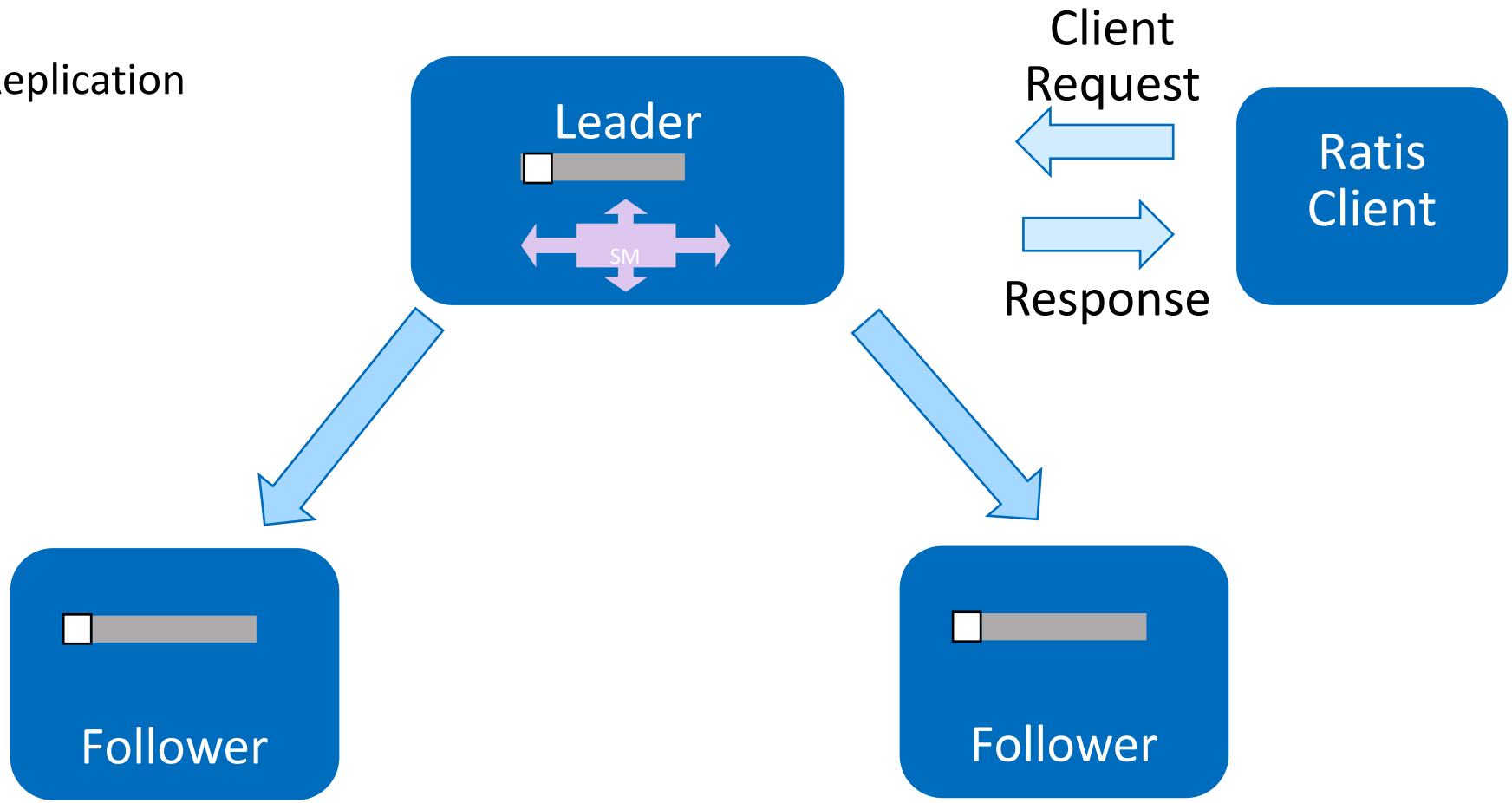
**Open** – Active Writers

- Need at least <*NumSpindles * Data nodes*> open active containers
- Data is spread across all data nodes
  - Improved IO and better chance of getting locality
- Keep DNs and ALL spindles busy

# Apache Ratis – Raft replication

Data and Metadata Replication



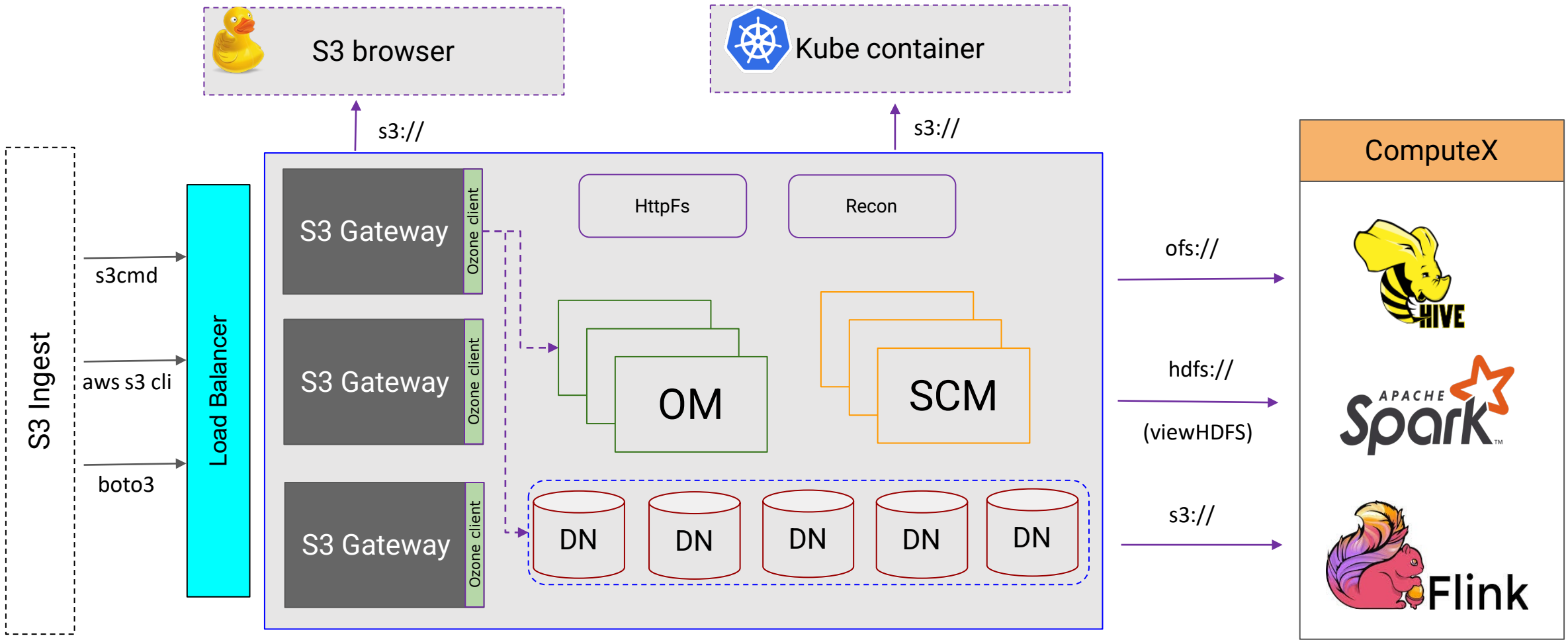

 

# Replication of an Open Container

- Use RAFT replication for both data and metadata
  - Proven to be correct
  - Traditionally Raft used for small updates and transactions, fits well for metadata
  - Performance considerations
    - When writing meta data into raft-journal, put data directly in block files
    - Raft-journal in separate disk – fast contagious writes without seeking
    - Data spread across the other disks
- The client uses Raft protocol to write data to the DNs storing the container

# Ozone S3 - protocol interchangeability



- Introduction
- Architecture
- Performance

# Metadata

- High throughput
- Horizontally scalable

# Horizontal scale – Block allocation

- Ability to use all datanodes and disks in write pipeline
- Set of containers vs set of pipelines

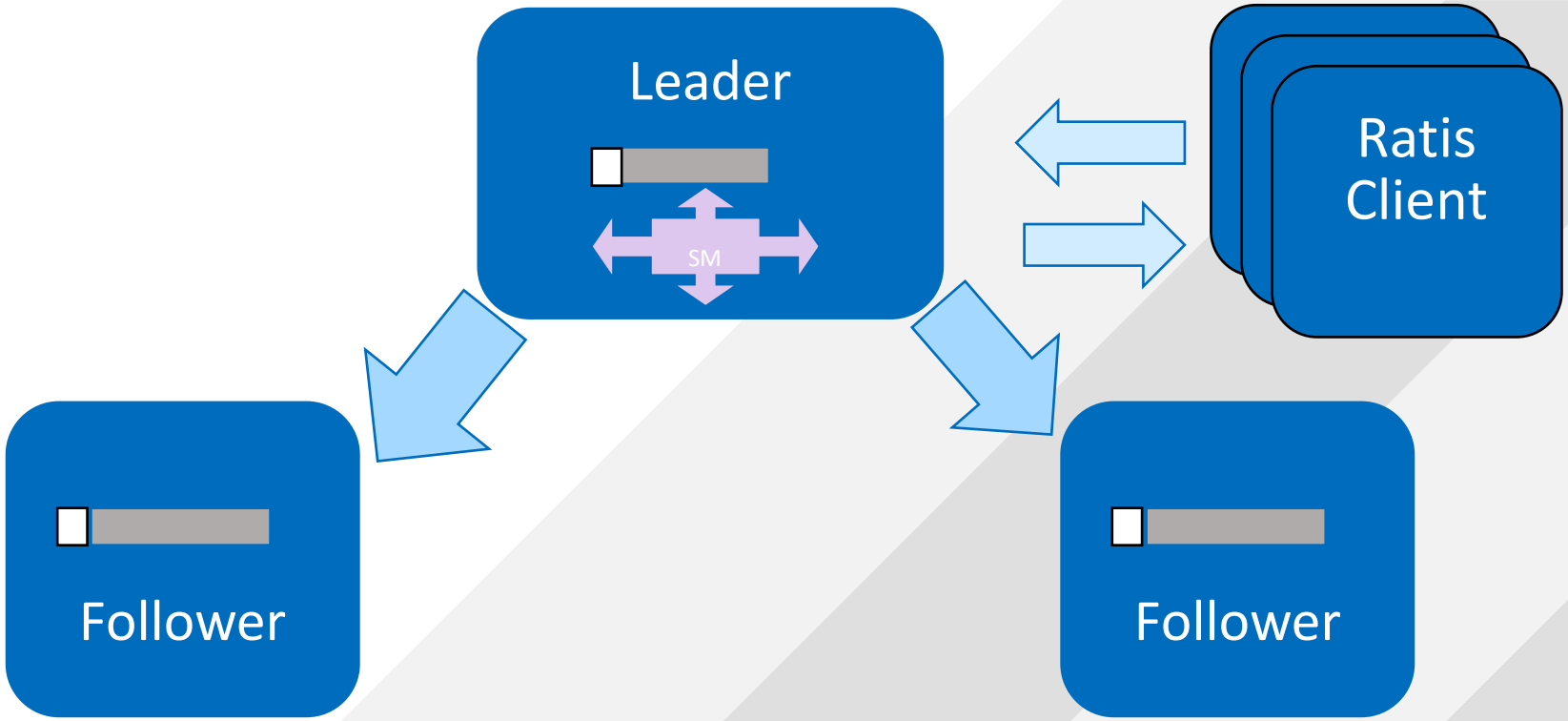
# Data

# Memory

- GBs of data writes
- Can’t be wasteful with Java

# Memory

- Retry Policy
  - Large GC pauses with fixed interval retry policy
  - Exponential/Linear Backoff policies provide stability



# Memory

- Linear Backoff policy
  - 5s, 5, 10s, 5, 15s, 5, 20s, 5, 25s, 5, 60s, 10
- Exponential Backoff – 4, 8, 16…

# Memory

- Buffer copies
  - Serialization/Deserialization
  - Ratis leader processes 3x amount of data

# Memory

- Bugs
  - Retained memory – cache
  - Unintentional buffer copies

# One line fixes

- Data transfer with Grpc
  - Netty assigns thread caches even for non-netty threads
  - -Dio.netty.allocator.useCacheForAllThreads=false
  - 3 min -> 37 seconds GC

# One line fixes

- RocksDb cache size
  - Default Cache size set to 64TB instead of 64MB
  - Unbounded cache – caching all blocks
  - 20 GB -> 3GB

Email -

ljain@apache.org

# Thank You

# Please take a moment to rate this session.

Your feedback is important to us.