aws S3 protocol

Hadoop FS

CSI

Apache Hadoop Ozone

hadoop.apache.org/ozone

THE APACHE SOFTWARE FOUNDATION

Why?

Scalability problem

- HDFS is designed for huge files
  - 200 million files for regular users
  - Companies with core devs 400-500 million
- New Opportunities and Challenges
  - Cloud
  - Streaming
  - Small files are the norm

HDFS Scalability

- Small files for HDFS
  - Memory pressure on Namenode
  - Higher network traffic (BlockReports)

Other solution

- HDFS-5477 Separate Block Management (open)
- HDFS-8286 Partial namespace in memory (open)
- HDFS-1052 HDFS federation (resolved)
- HDFS-10467 Router based federation (resolved/in-progress)
- HDFS-7240 "Scaling HDFS" (Ozone)

Ozone borrows many idea learned from these efforts and is a super set of these approaches

Design Tenets

- Strong Consistency
  - Easier to write applications. No need for S3Guard.
- Simple architecture
  - Easy to understand
- Use proven building blocks
  - Raft protocol for consensus - E.g. HA and write pipeline.
  - RocksDB from Facebook
- Open Source
  - Work well with the Apache Hadoop/Spark ecosystem

Usability problem

- HDFS can be used from Hadoop ecosystem?
  - Machine learning code? Python?
  - S3 compatible code?
  - Kubernetes? Containerization? File system?
- Object store seems to be a more universal abstractions

Apache Hadoop Ozone

- Ozone is a scalable, redundant, and distributed object store for Hadoop

SCALABLE

Ozone is designed to scale to tens of billions of files and blocks and, in the future, even more.

CONSISTENT

Ozone is a strongly consistent object store. This consistency is achieved by using protocols like RAFT.

CLOUD-NATIVE

Ozone is designed to work well in containerized environments like YARN and Kubernetes.

SECURE

Ozone integrates with kerberos infrastructure for access control and supports TDE and on-wire encryption.

MULTI-PROTOCOL SUPPORT

Ozone supports different protocols like S3 and Hadoop File System APIs.

HIGHLY AVAILABLE

Ozone is a fully replicated system that is designed to survive multiple failures.

“Ozone is a
spiritual successor
to Hdfs”

History.md

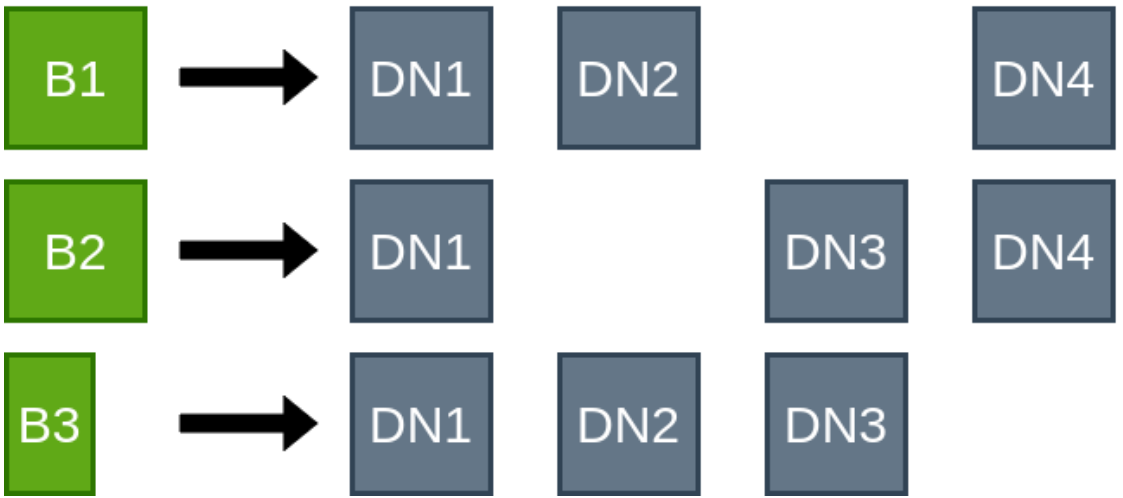
- Started as a feature branch in Hadoop
- Merged in 2018 to Hadoop trunk
  - Built by optional profile
  - Separated release lifecycle
  - Separated subproject (“HDDS”)
- 2019 Q4: Moved to a separated git repostory (apache/hadoop-ozone)
- 2020.03: First beta release

How to store?

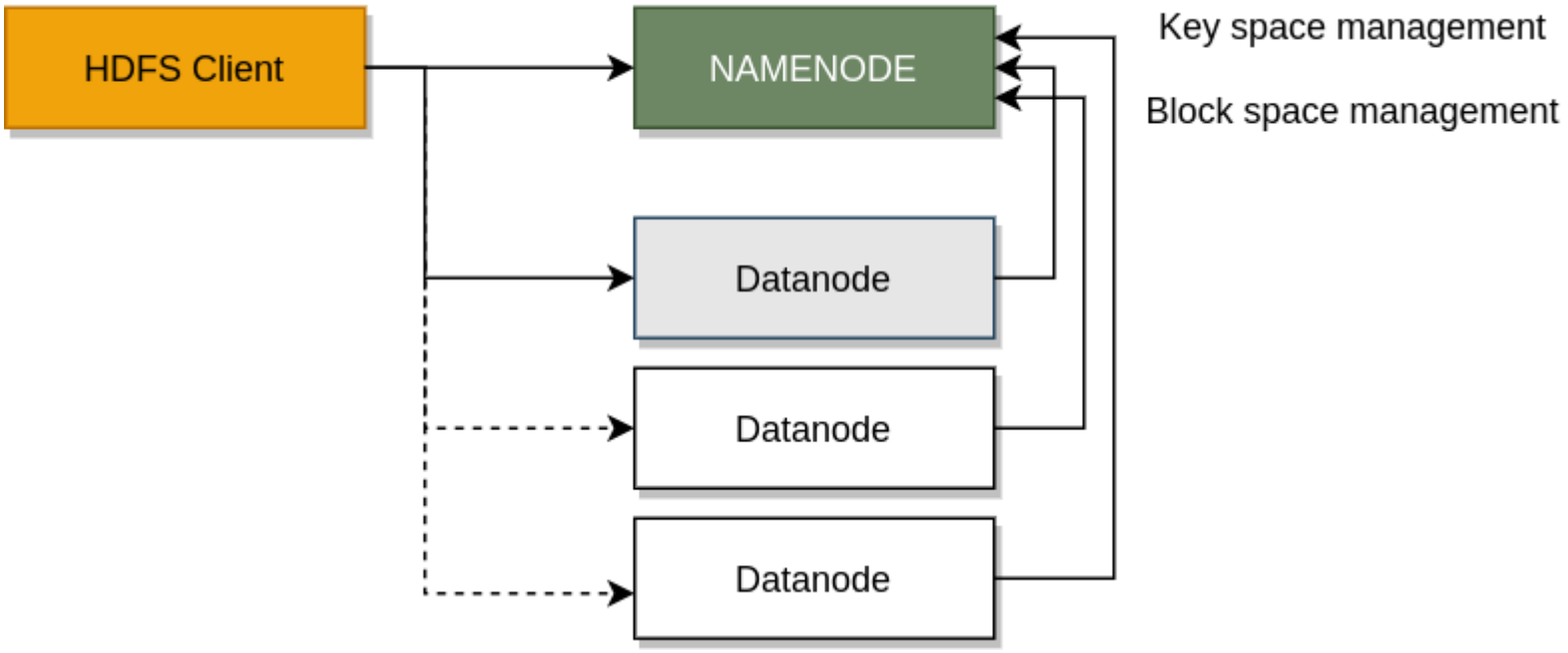
How to Store files?

Split file to blocks

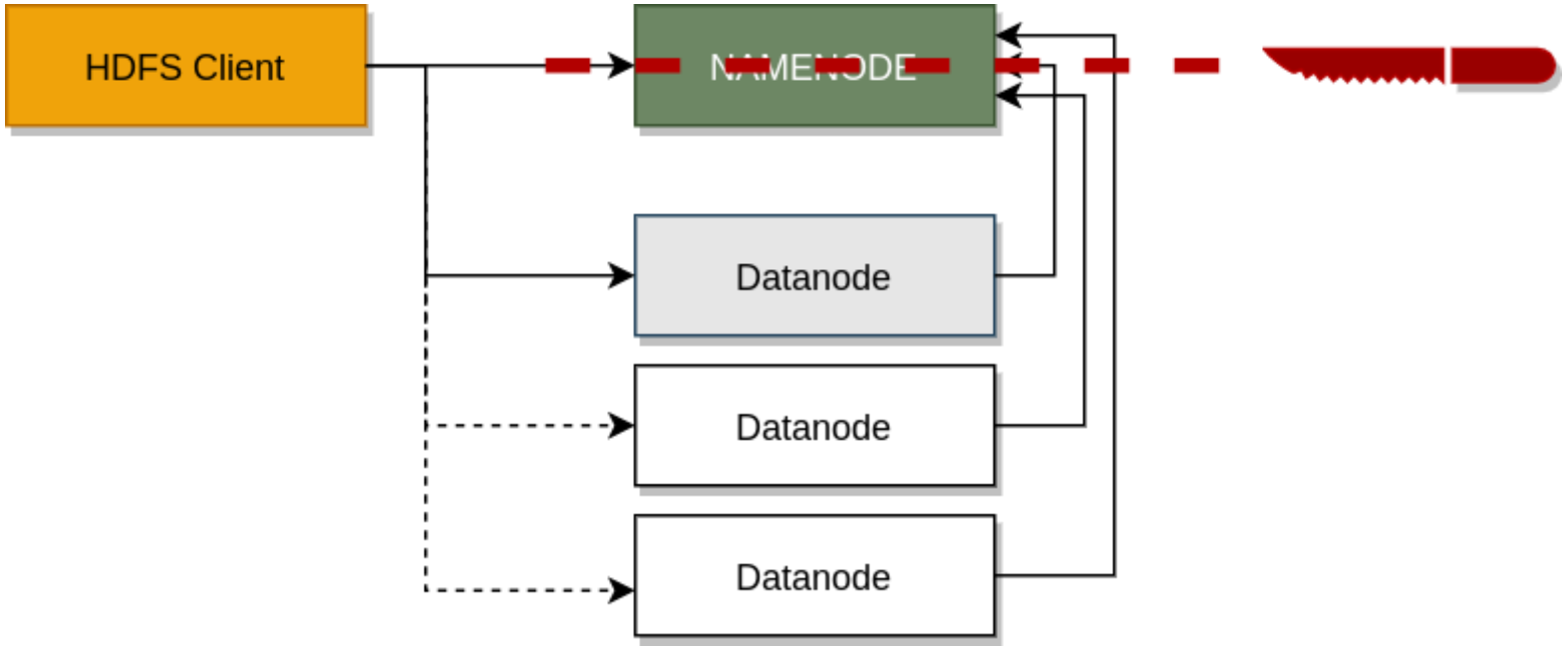
Store replicas of blocks on Datanodes



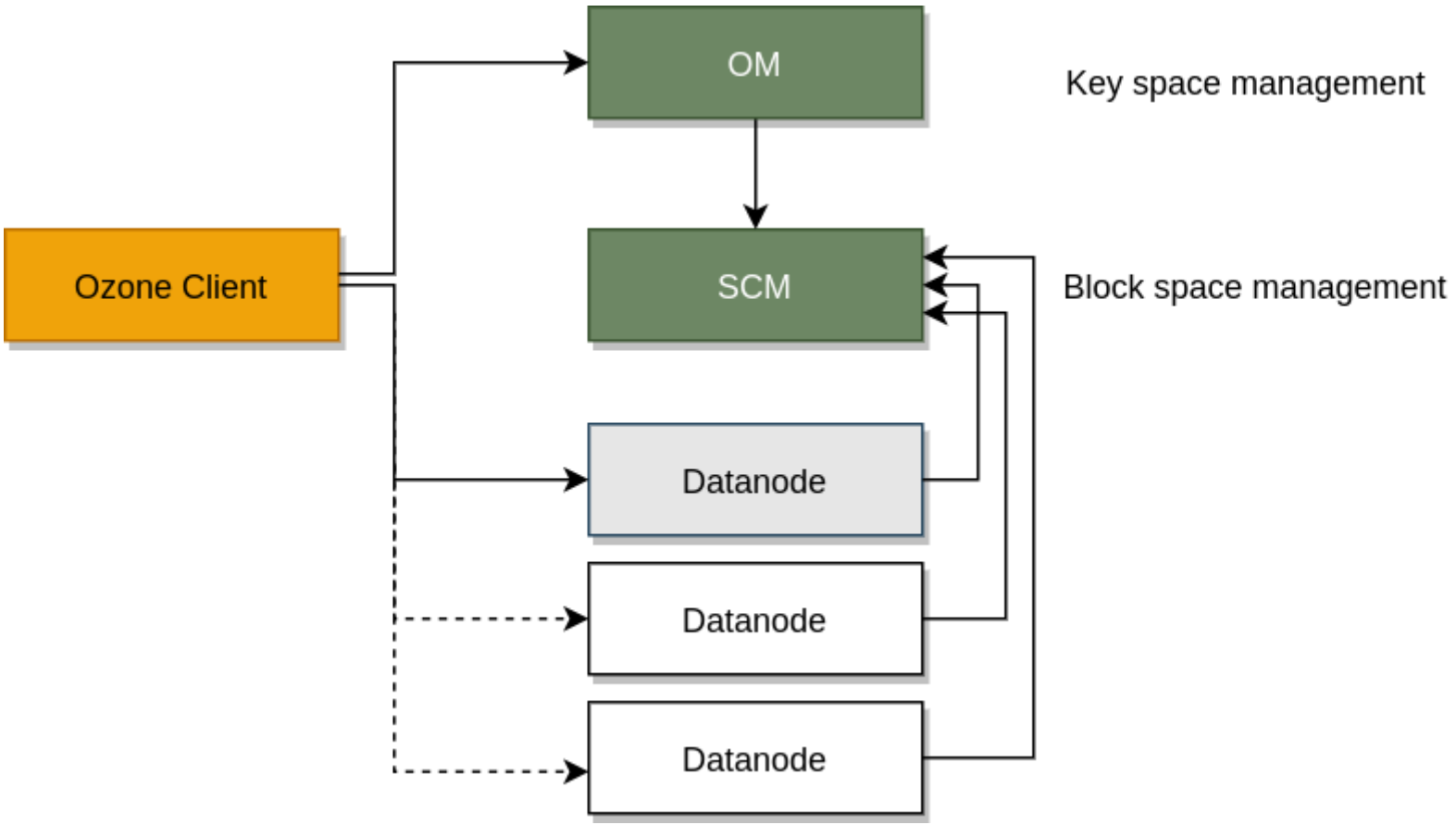
HDFS components



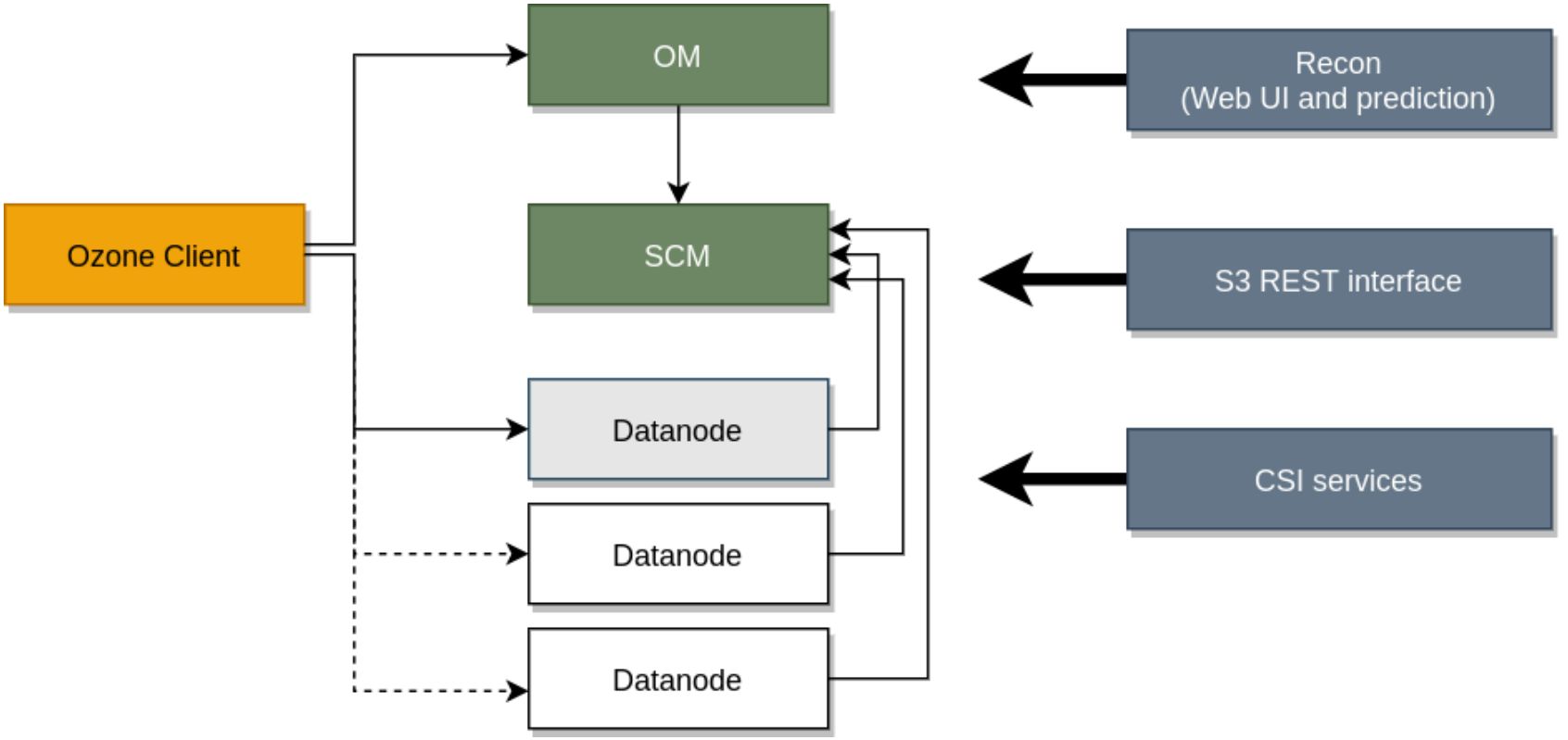
Separate key / block space management



Ozone components



Ozone components (full picture)



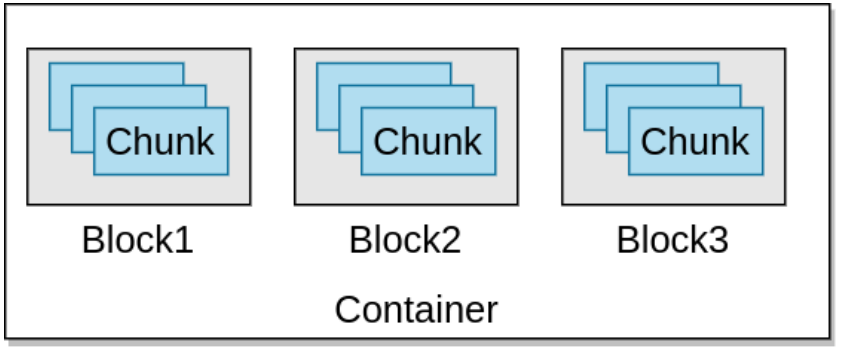
Key concepts

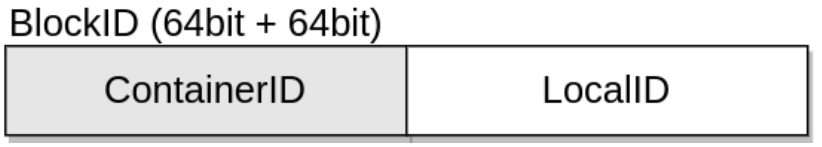
Key space

- 3 levels of hierarchy **/vol1/bucket1/key1**
  - Volumes *(~ user namespaces)*
    administration unit, managed by admins
  - Buckets *(~ dirs)*
    created/deleted by users
  - Keys *(~files)*
    flat hierarchy (with indexes)

Block space

- Each file is stored in blocks
- Blocks are replicated in groups
  - Container is unit of replication





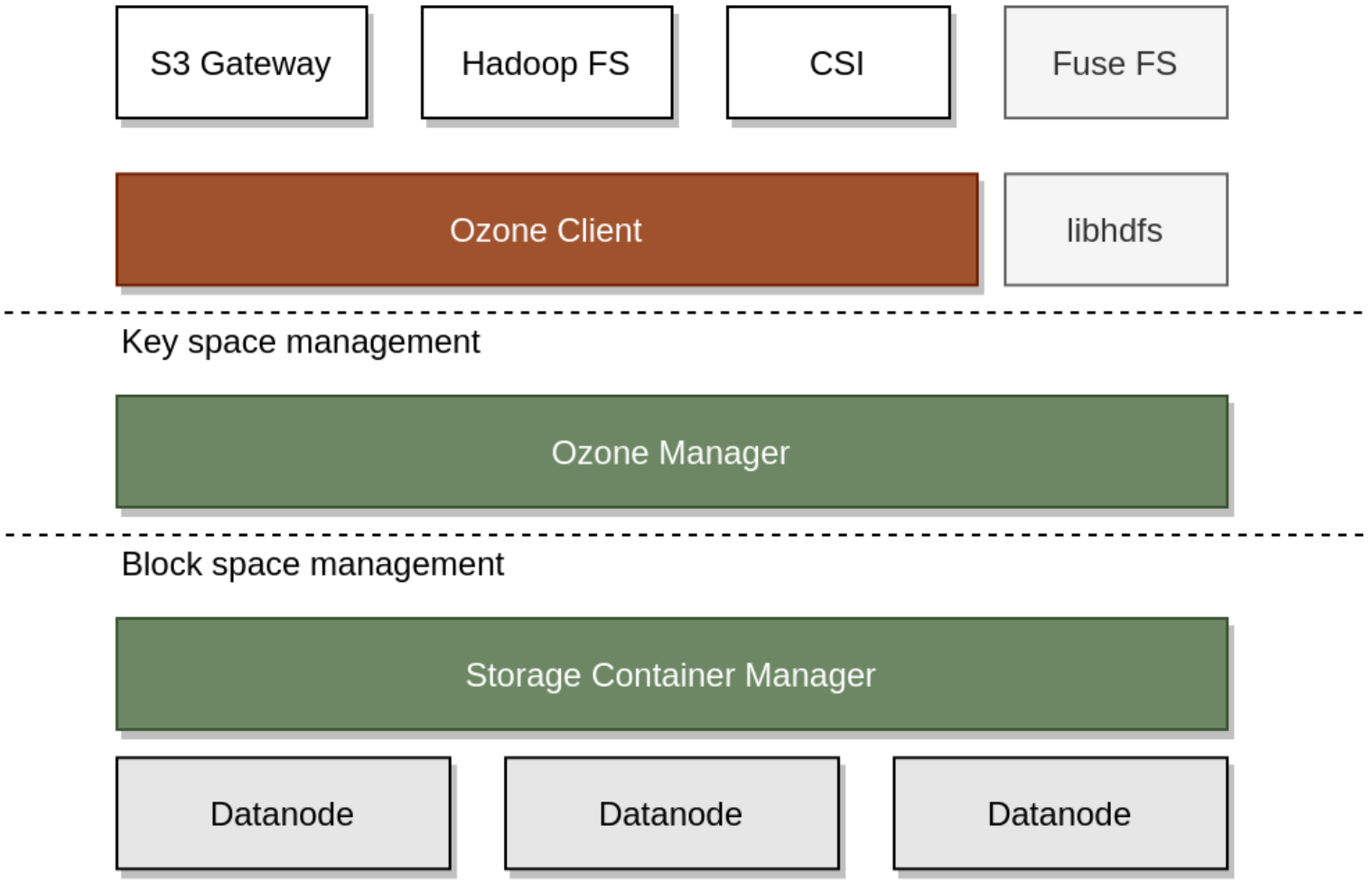
Open Containers

- Read / Write support
- Replicated with Ratis (sync)
- Leader has the latest state (Stale reads?)
- Closed: if full or in case of error

Closed Containers

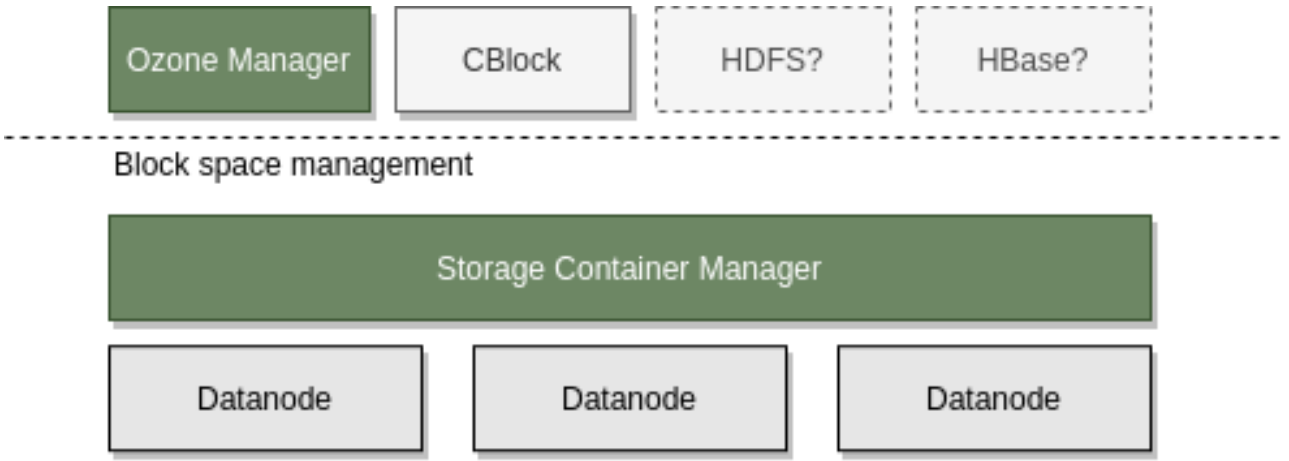
- **Immutable** (Read Only)
- Replicated with simple network copy (async)
- Easy to read from all members
- GC is required
  (to handle delete)

Ozone Layers



Original Vision

- Use block layer by other applications not just for Object Store



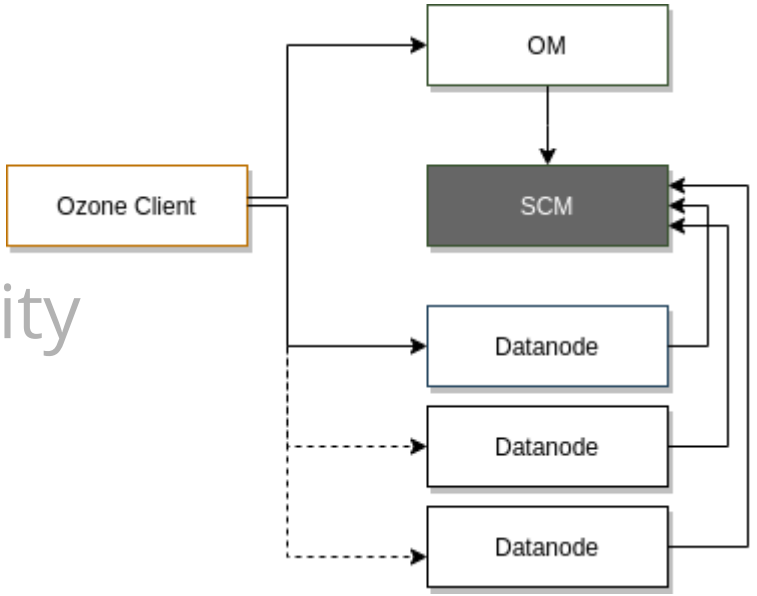
How to scale?

- Key / Block space are separated
  - Less memory pressure
- Report / replicate containers and not blocks
  - Smaller block reports
- Keep partial namespace in memory (if required)
  - RocksDB + SSD can provide good enough performace

Ozone Services

Storage Container Manager

- Block space management
  - Allocate blocks
  - Replicate containers
  - Manage certificates #security



SCM: network services

- **Pipelines:** List/Delete/Activate/Deactivate
  - Raft groups are planned by SCM
- **Containers:** Create / List / Delete containers
- Admin related requests
  - Safemode status/modification
  - Replication manager start / stop
- CA authority service

RPC

SCM: network services II.

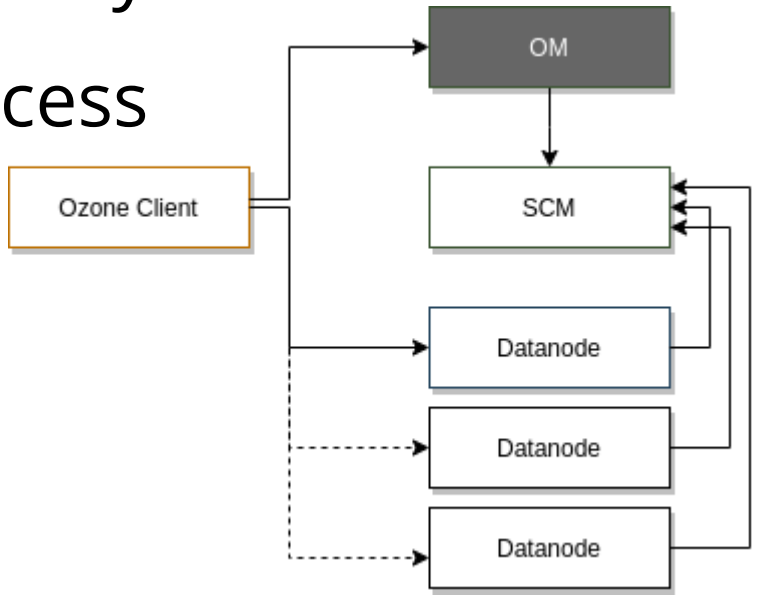
- Datanode HeartBeat protocol
  - From Datanode to SCM (30 sec by default)
  - Commands can be added to the response

SCM: persisted data (RocksDB)

- Pipelines
- Contaeners
- Deleted blocks
- Valid certs
- Revoked certs
- *Node: in-memory*

Ozone Manager (OM)

- Key Space Management
  - Managing volumes/buckets/keys
  - Secondary indexes for fs access



OM: network services

- **Key, Bucket, Volume** / CRUD
  - Multipart upload (Initiate, Complete...)
- FS related calls
  - GetFileStatus, CreateDirectory, CreateFile, LookupFile
- ACL related (for internal ACLs)
- Delegation token (Get / Renew / Cancel)
- Trash related commands (WIP)
- **Admin APIs**
  - Get S3 secret
  - ServiceList (to find SCM)
  - DBUpdates (Recon downloads snapshots)
  -

OM: persisted data (RocksDB)

- Volume / Bucket / Key tables
- OpenKey table (created key, but not committed)
- Delegation token table
- PrefixInfo table (ACL for prefixes)
- S3 secret table
- Multipart info table
- Deleted table

Datanodes

- Health check
  - reports (**Containers**, disk...) to the SCM as a heartbeat
  - Commands can be received in the response

    Close container, Delete container
- Open Containers:
  - Starting RAFT server and forward requests to it
- Closed Container
  - Replicate as immutable package

Datanodes

- Datanodes are forming RAFT groups (pipelines)
- Client is communicating with the leader
  - All requests are replicated via RAFT

Datanode: network services

- Datanode protocol (for the clients)
- Heartbeat (DN → SCM) for management
- Ratis / Raft endpoint (for other datanodes)

RPC

Datanode client calls

- **Containers:** Create/Read/Updated/DeleteList
- **Blocks:** Put/Get/DeleteList
- **Chunks**: Read/Delete/Write/List
- PutSmallFiles/ GetSmallFiles
- (CloseContainer)
- CopyContainer (export Container)

RPC

Datanode → SCM messages

- GetVersion/Register (Initial handshake)
- SendHeartbeat (request)
  - DatanodeDetails
  - NodeReport
  - ContainerReport
  - PipelineReport
  - IncrementalContainerReport
  - Container/Pipeline actions (request to close

RPC

SCM → Datanode messages

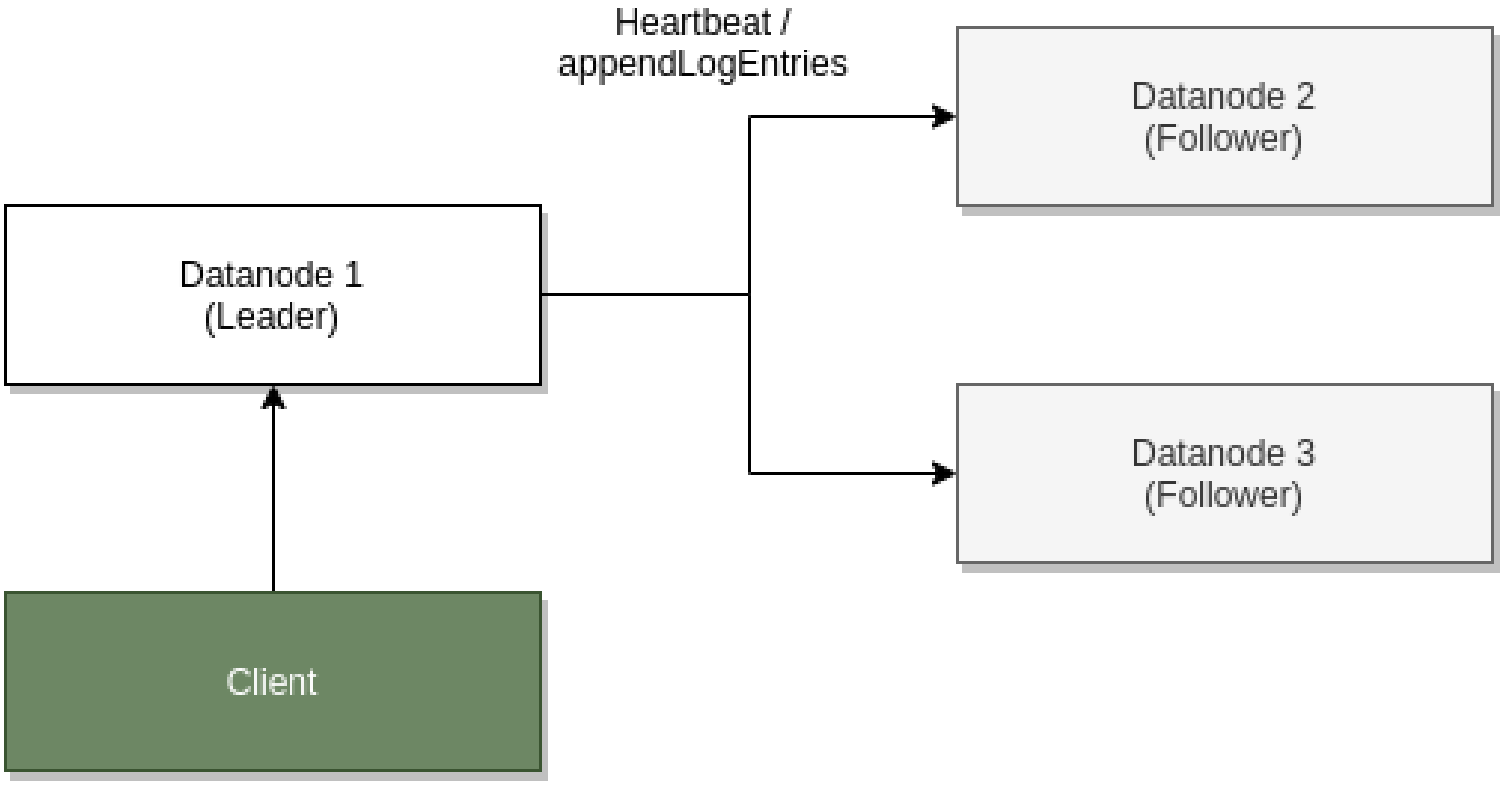
- Sent in the response
- Reregister command
- Container replication commands
  - Replicate, Delete, Close,
- Pipeline commands

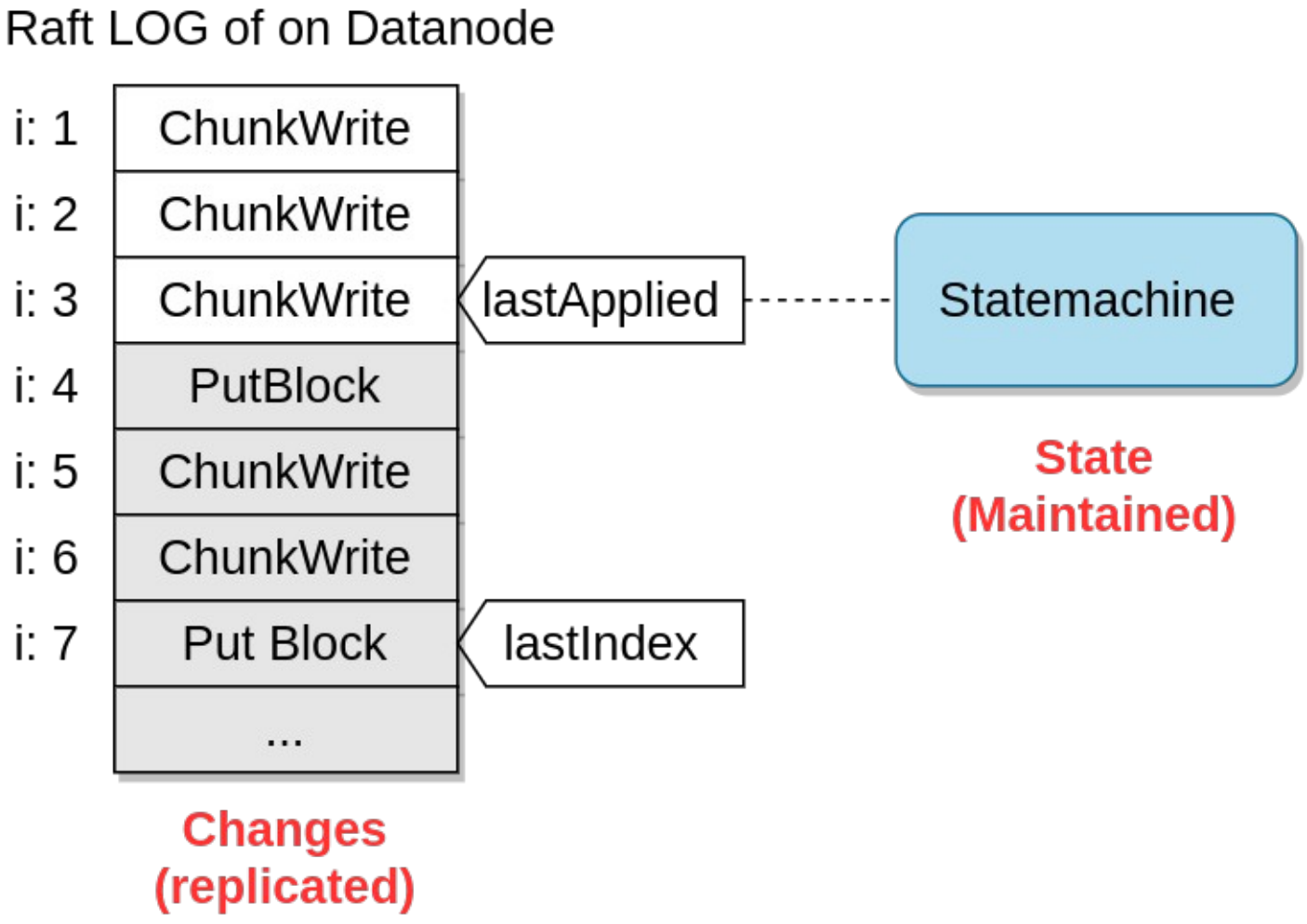
Datanode: network services

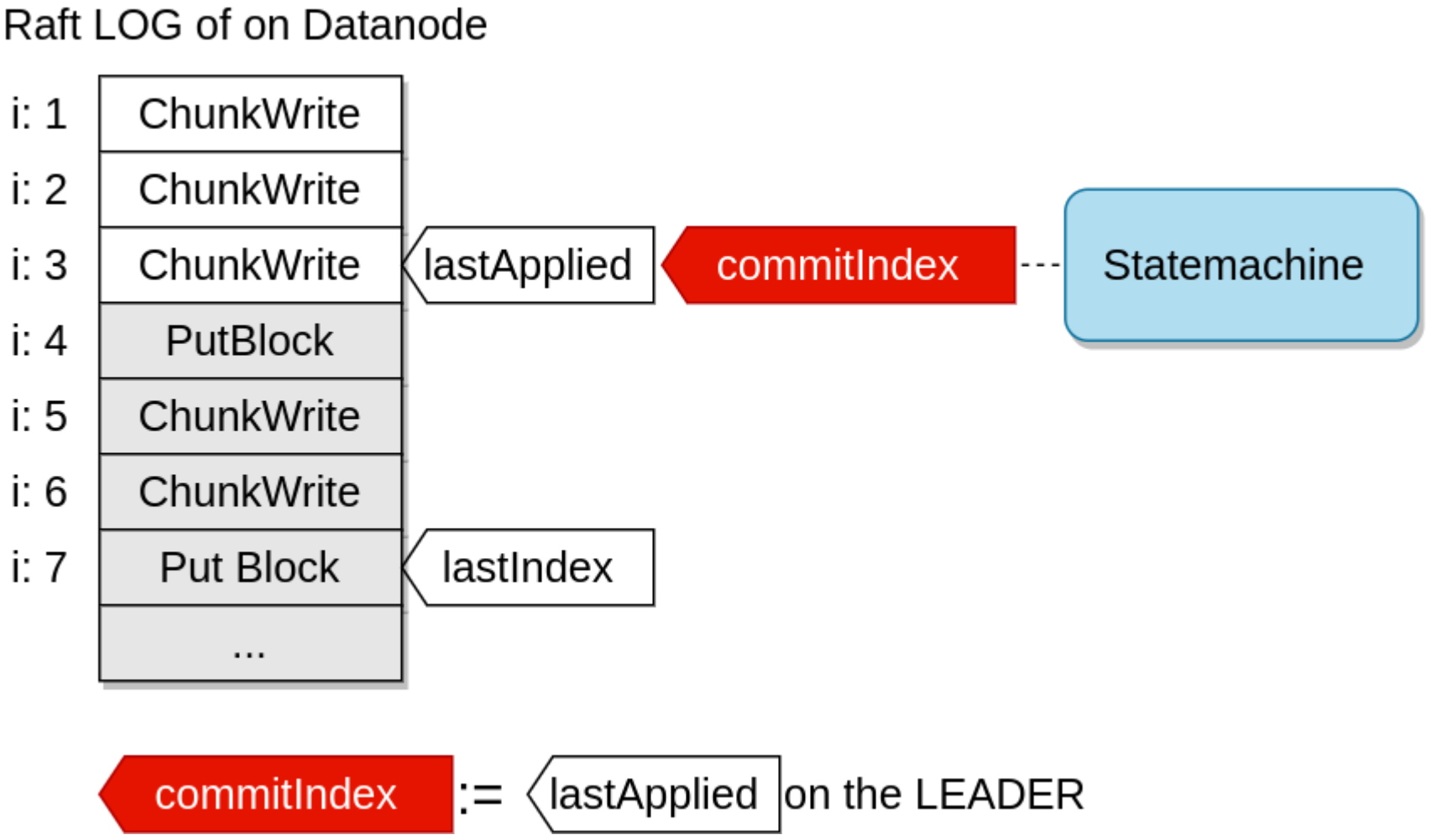
- Datanode protocol (for the clients)
- Heartbeat (DN → SCM → DN) for management
- **Ratis / Raft endpoint (for other datanodes)**

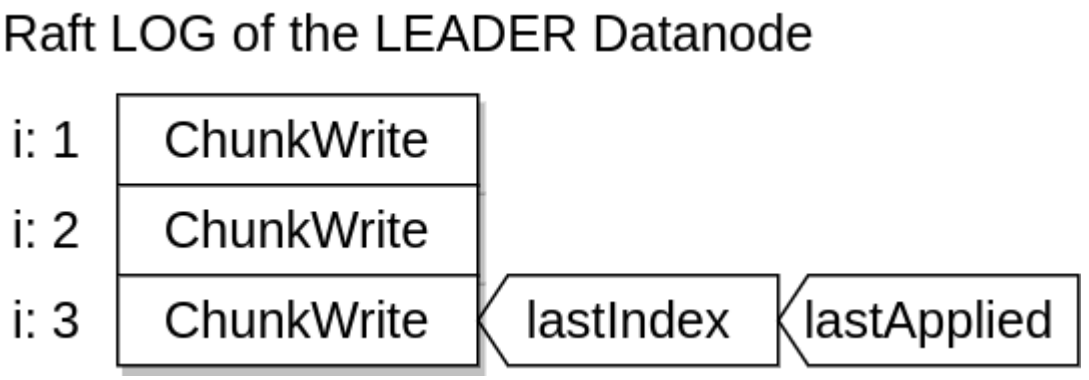
Raft

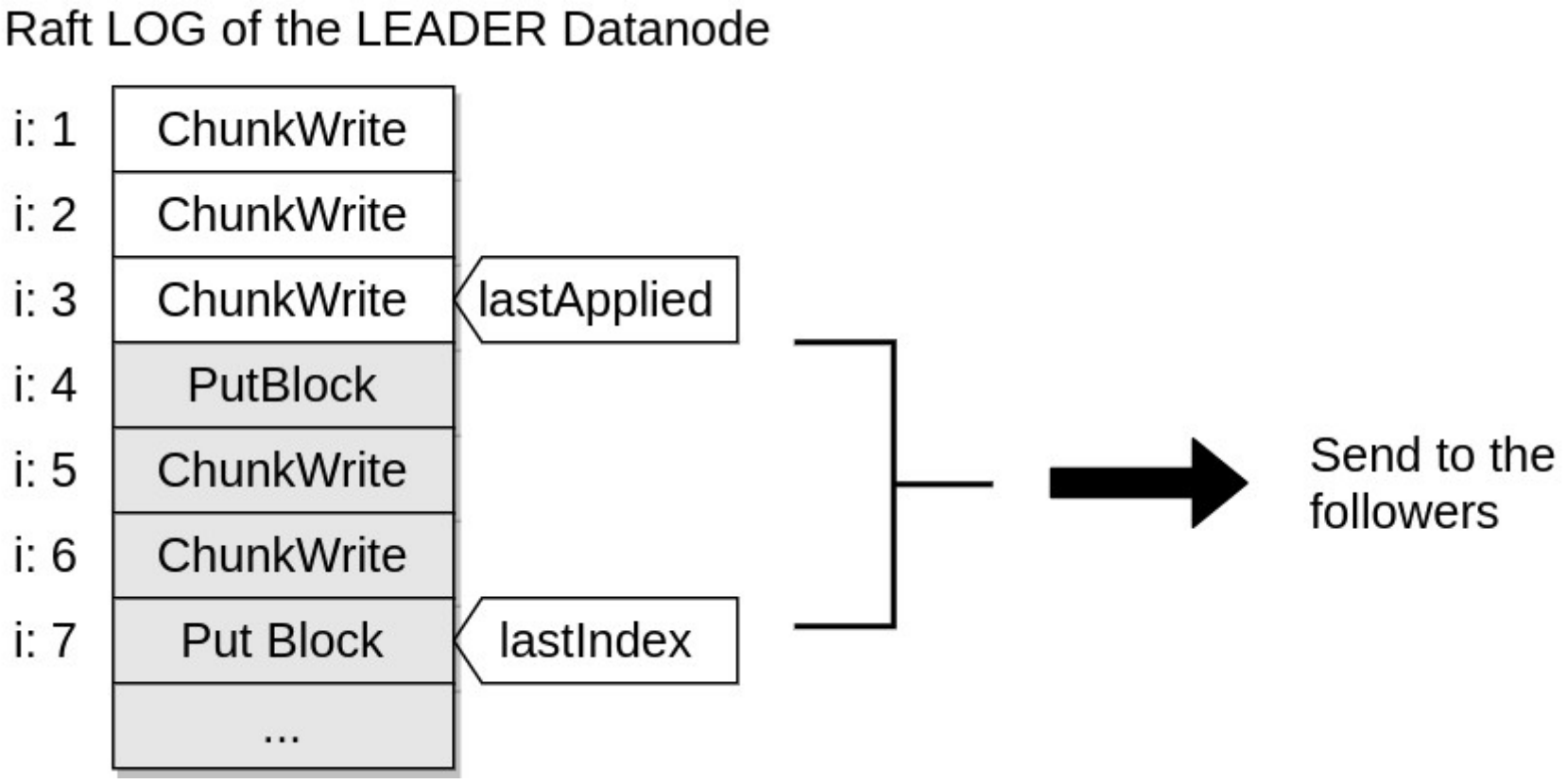
- *“Raft is a consensus algorithm for managing a replicated log”*
- *“Raft more understandable thanPaxos and also provides a better foundation for building practical systems”*


Heartbeat /
appendLogEntries
Datanode 2
(Follower)
Datanode 1
(Leader)
Datanode 3
(Follower)
Client

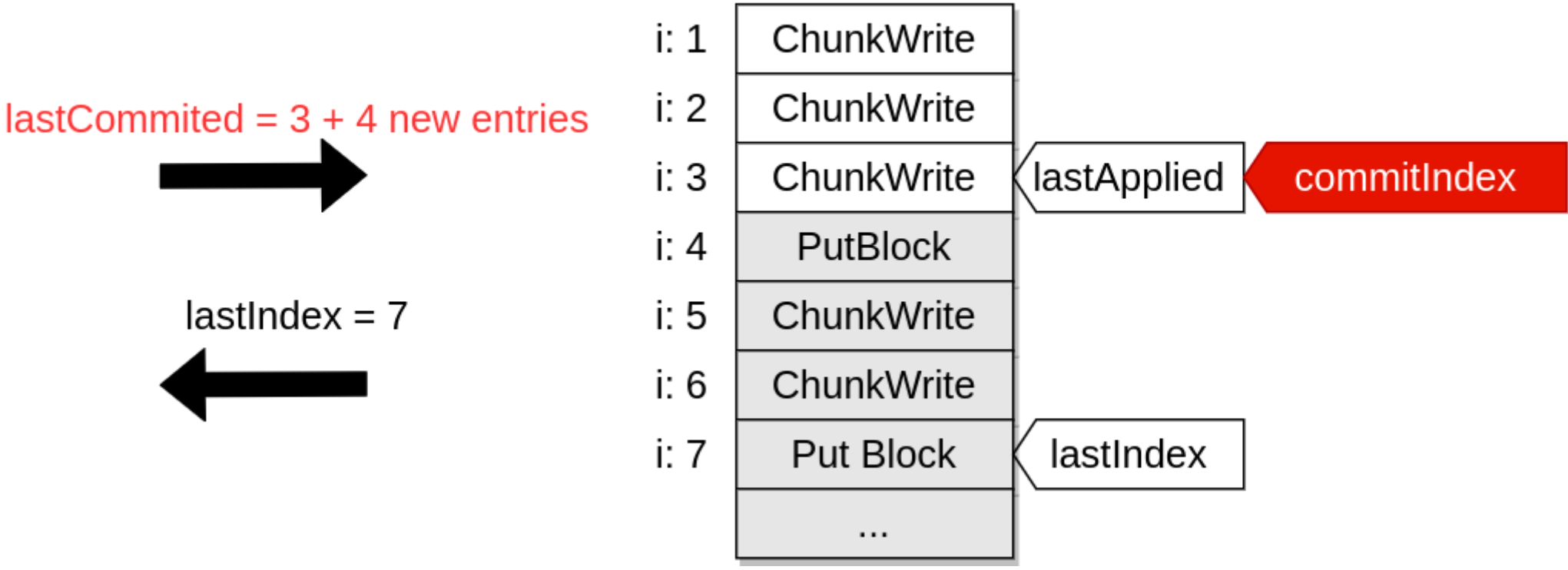

Raft LOG of on Datanode
i: 1
ChunkWrite
i: 2
ChunkWrite
i: 3
ChunkWrite
lastApplied
Statemachine
i: 4
PutBlock
i: 5
ChunkWrite
i: 6
ChunkWrite
i: 7
Put Block
lastIndex
...
State
(Maintained)
Changes
(replicated)


Raft LOG of on Datanode
i: 1
ChunkWrite
i: 2
ChunkWrite
i: 3
ChunkWrite
lastApplied
commitIndex
Statemachine
i: 4
PutBlock
i: 5
ChunkWrite
i: 6
ChunkWrite
i: 7
Put Block
lastIndex
...
commitIndex
:=
lastApplied
on the LEADER

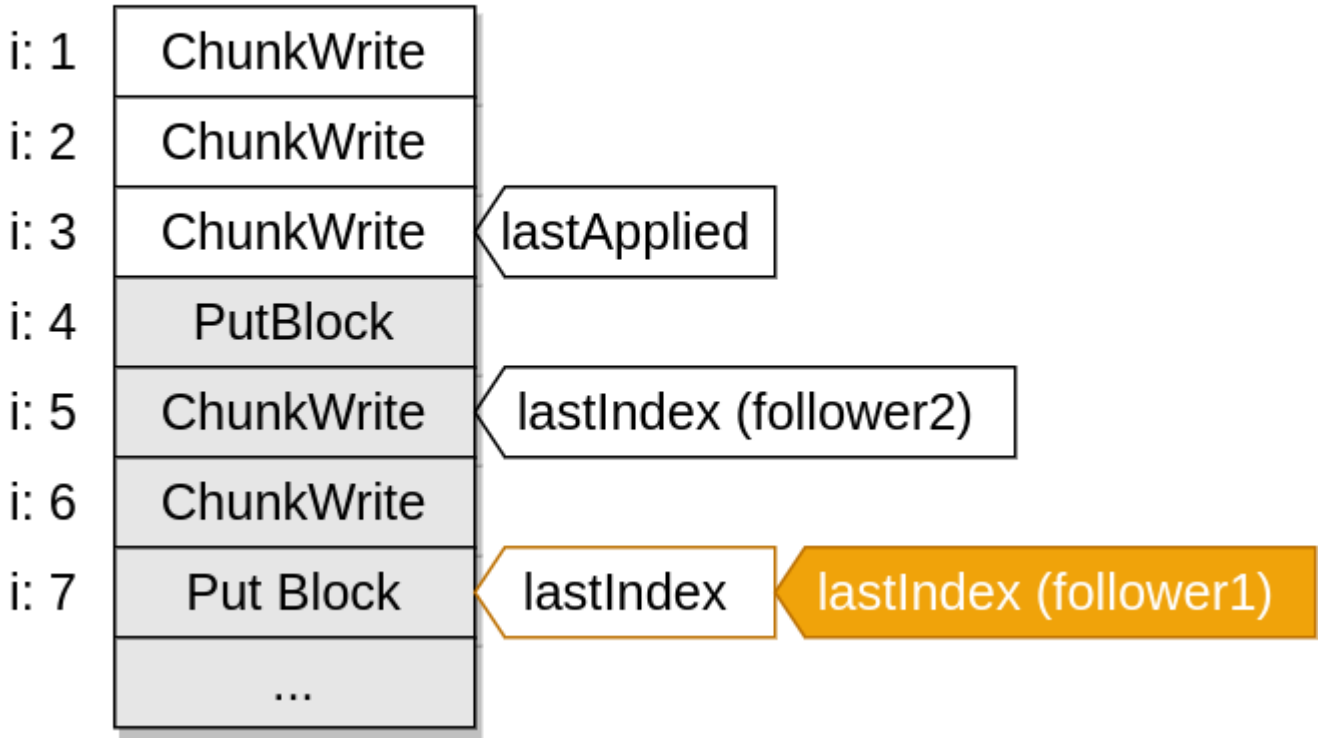

Raft LOG of the LEADER Datanode
i: 1
ChunkWrite
i: 2
ChunkWrite
i: 3
ChunkWrite
lastIndex
lastApplied

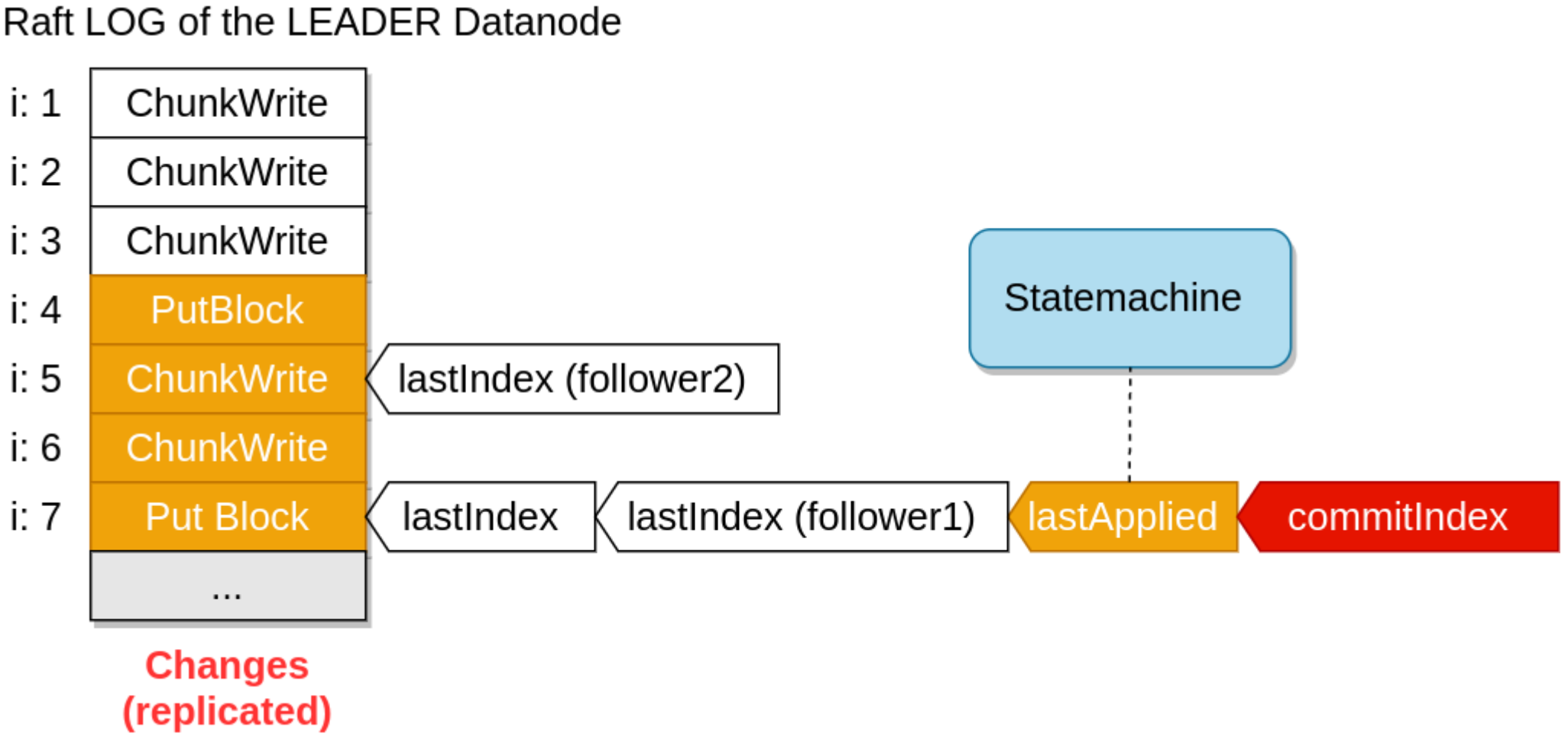

Raft LOG of the LEADER Datanode
i: 1
ChunkWrite
i: 2
ChunkWrite
i: 3
ChunkWrite
lastApplied
i: 4
PutBlock
i: 5
ChunkWrite
i: 6
ChunkWrite
i: 7
Put Block
lastIndex
...
Send to the followers

Raft LOG of the FOLLOWER Datanode

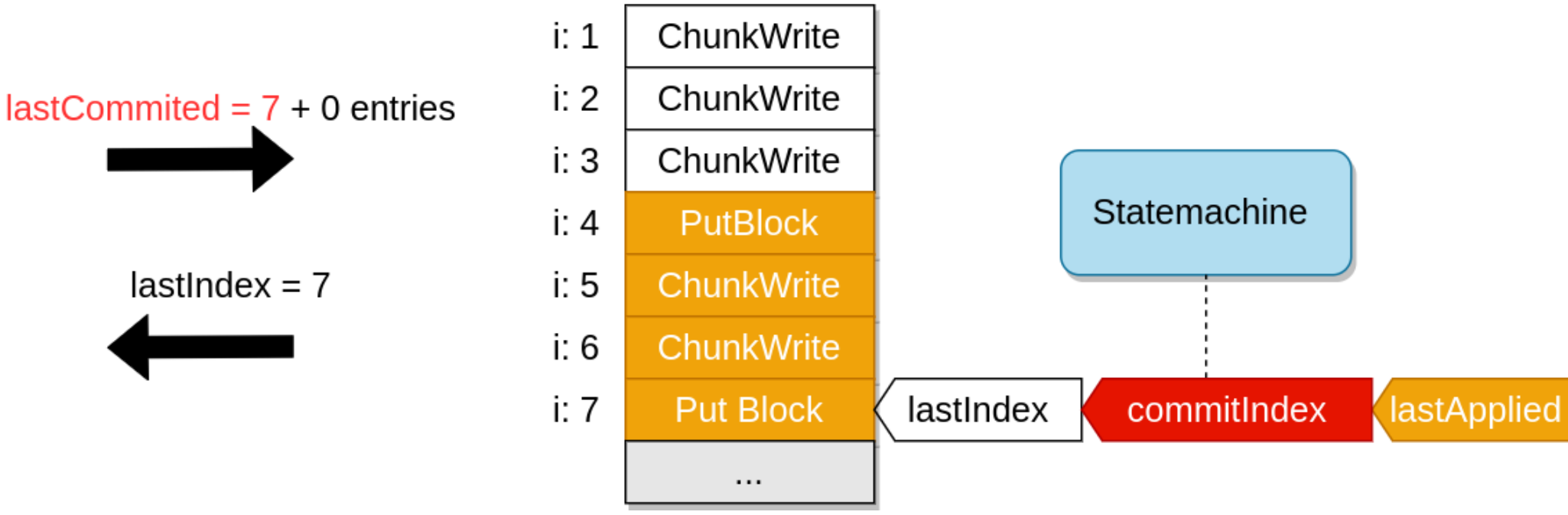


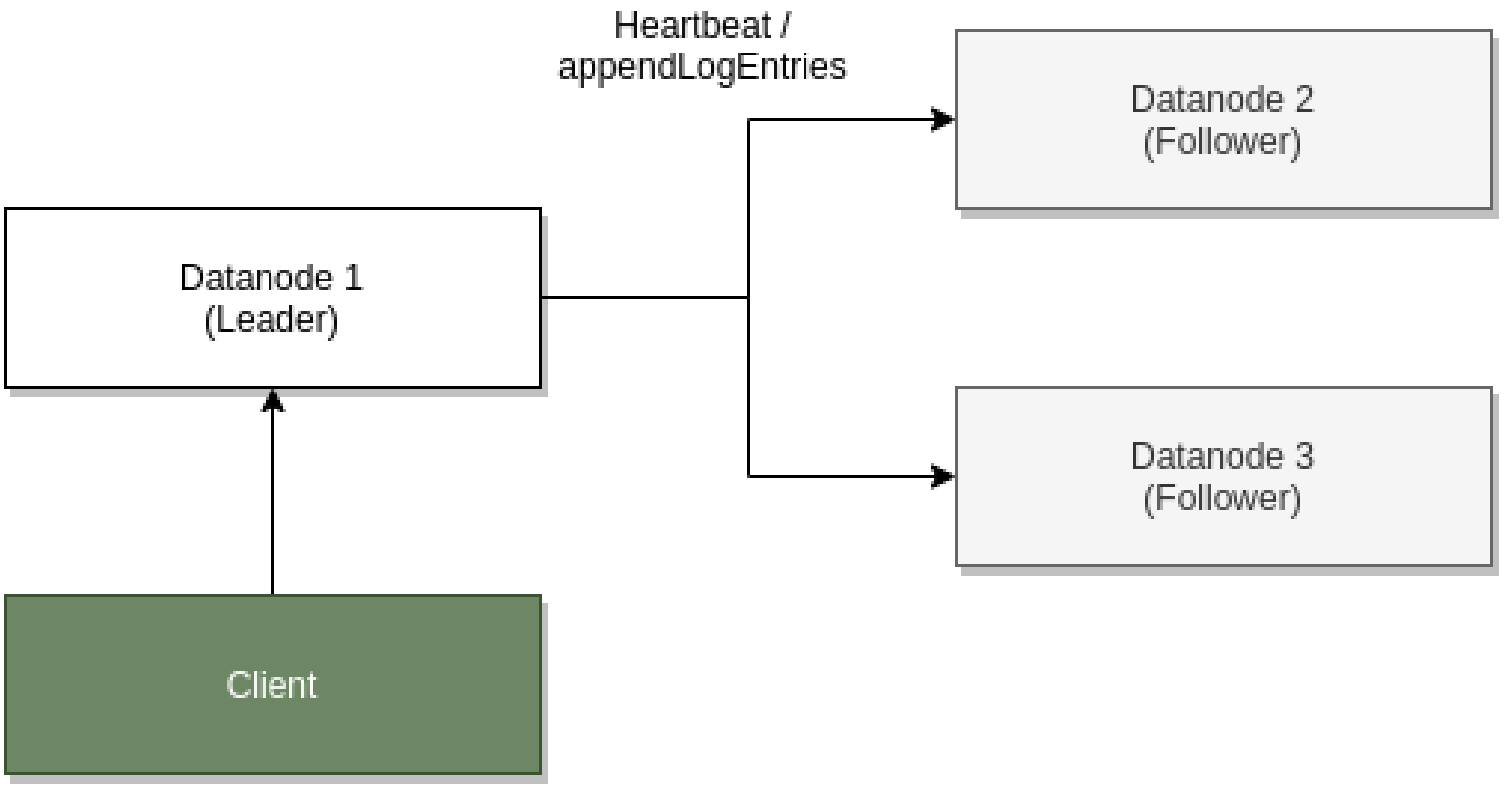
Raft LOG of the LEADER Datanode




Raft LOG of the LEADER Datanode
i: 1
ChunkWrite
i: 2
ChunkWrite
i: 3
ChunkWrite
i: 4
PutBlock
i: 5
ChunkWrite
lastIndex (follower2)
i: 6
ChunkWrite
i: 7
Put Block
lastIndex
lastIndex (follower1)
...
Statemachine
lastApplied
commitIndex
Changes
(replicated)

Raft LOG of the FOLLOWER Datanode



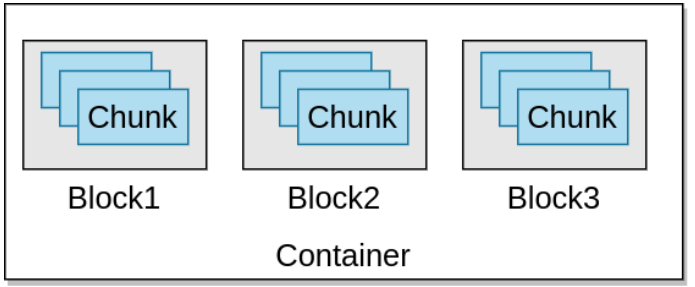

Heartbeat /
appendLogEntries
Datanode 2
(Follower)
Datanode 1
(Leader)
Datanode 3
(Follower)
Client

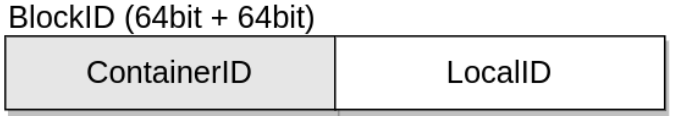
Apache Ratis (incubator)

- RAFT implementation as a Java **library**
  - Embeddable, pluggable protocol, statemachine,...
- Support high performance with many optmization
- Off the RAFT log data
  - Metadata is saved to the Raft log, chunk data is not
  - Multi Raft support (one datanode can be part of multiple raft ring)
  - Batching and async processing

DN: persisted data

- Per container directory
  - Yaml file (metadata, verison, path,...)
  - RocksDB:
    - Key: LocalID
    - Value:
      - Map<String,String> (Generic metadata)
      - List<ChunkInfo> (ChunkInfo: chunkName, length, blockOffset, checksum)
  - Chunk data
    - v1: 1 file per chunk
    - v2: 1 file per block



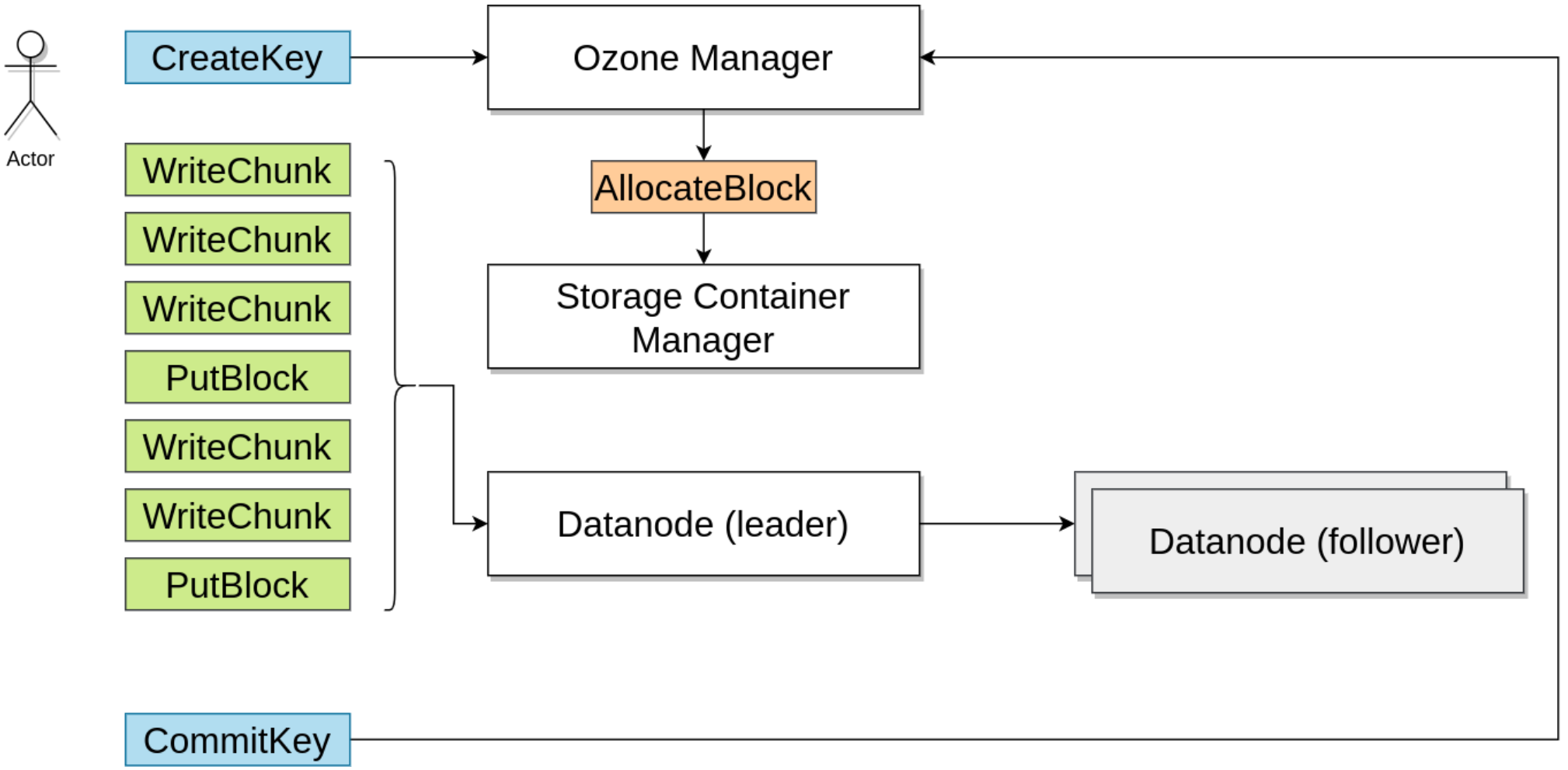


+2 other services

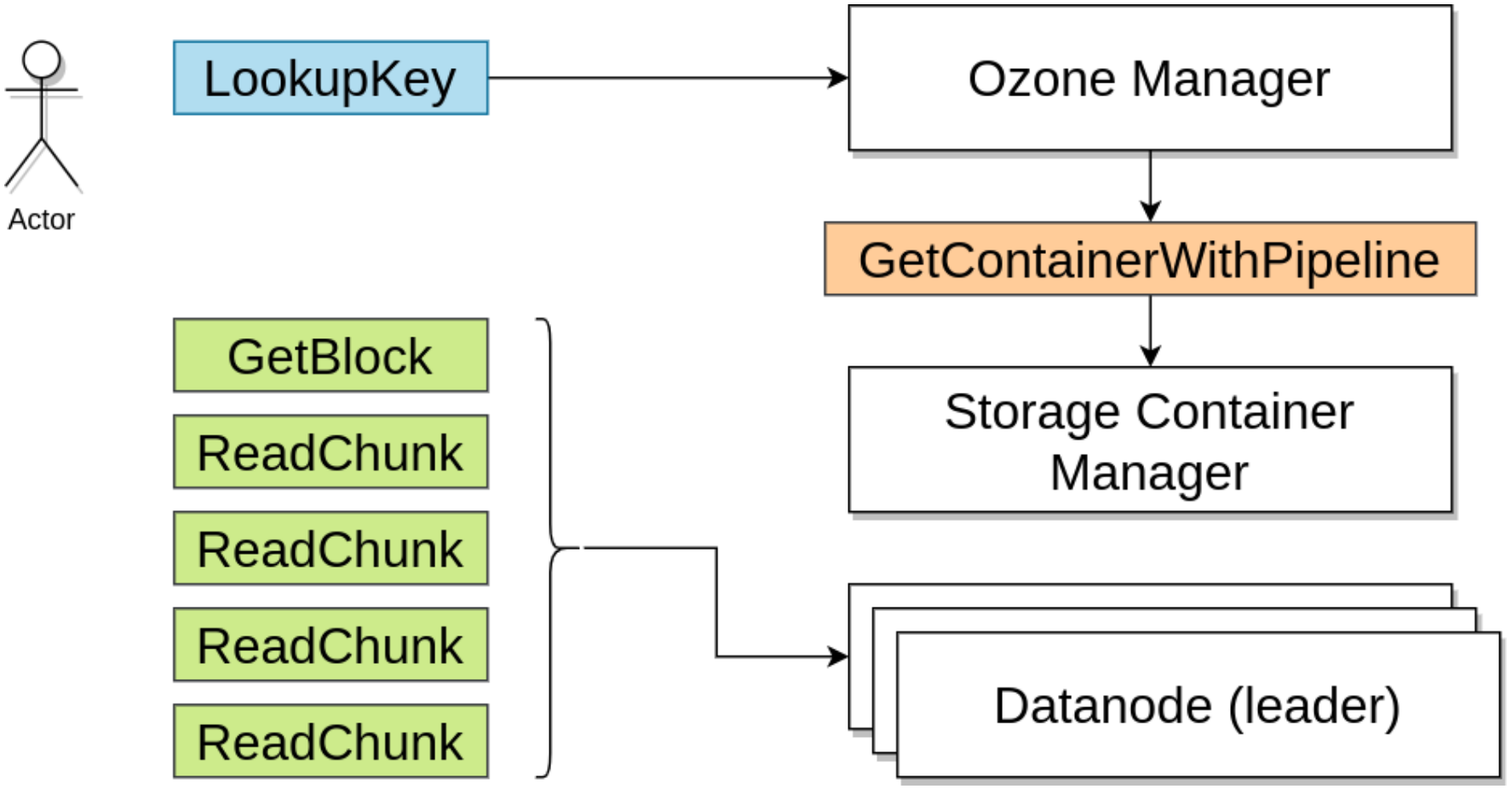
- Recon
  - Web UI, prediction, analytics
- S3 gateway
  - stateless REST → Ozone RPC translator

Read / write path

Write path


Actor
CreateKey
Ozone Manager
WriteChunk
WriteChunk
WriteChunk
PutBlock
WriteChunk
WriteChunk
PutBlock
AllocateBlock
Storage Container Manager
Datanode (leader)
Datanode (follower)
CommitKey

Read path


Actor
LookupKey
Ozone Manager
GetContainerWithPipeline
GetBlock
ReadChunk
ReadChunk
ReadChunk
ReadChunk
Storage Container Manager
Datanode (leader)

Use it!

aws S3 protocol

Hadoop FS

CSI

Apache Hadoop Ozone

hadoop.apache.org/ozone

THE APACHE SOFTWARE FOUNDATION

Hadoop file system (old style)

- **o3fs://**
- View to one specific bucket only!
- hdfs dfs -ls o3fs://bucket1.volume1/dir1/file1.txt

Hadoop file system (new style)

- **ofs://**
- View to the whole keyspace
- hdfs dfs -ls o3fs://om:9862/vol1/bucket1/dir1/file1.txt
- DistCp friendly

S3 compatible REST api

- Stateless gateway
  - AWS S3 REST call → Ozone RPC call
- All the important S3 endpoints are implemented, to use with
  - Hadoop s3a
  - AWS Cli
  - S3 compatible FUSE file system (s3)
  - Python based clients (boko)
  - Go based clients

AWS Cli

- aws s3api --endpoint http://localhost:9878 create-bucket –bucket=bucket1
- aws s3cp --endpoint http://localhost:9878 README.txt s3://wordcount

Security

Security

- HDFS security is based on Kerberos
  - Kerberos cannot sustain the scale of applications running in a Hadoop Cluster.
- HDFS relies on Delegation tokens and block tokens.
  - Ozone uses the same, so applications have no change.
- SCM comes with its own Certificate Authority.
  - End users do NOT know about it.
- Allows us to move away from the need of Kerberos setup for each data node. We need only Kerberos on OM and SCM.

Dev/ops experience

3..., 2..., 1..., UP!

- 10 secs from build to deploy
  - Even a full secure cluster can be started locally
- docker-compose based example clusters
  - Working as documentation

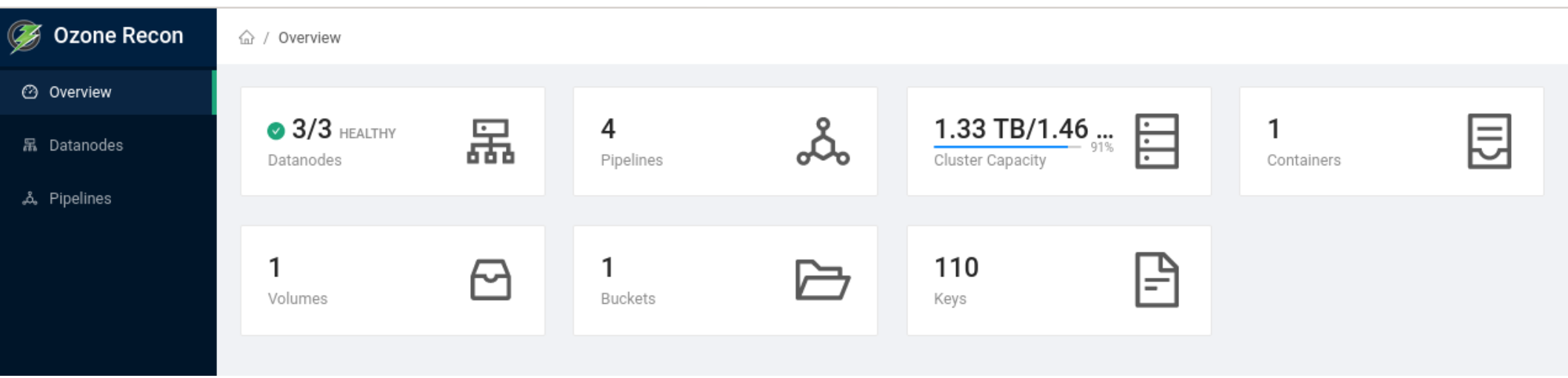
Observability

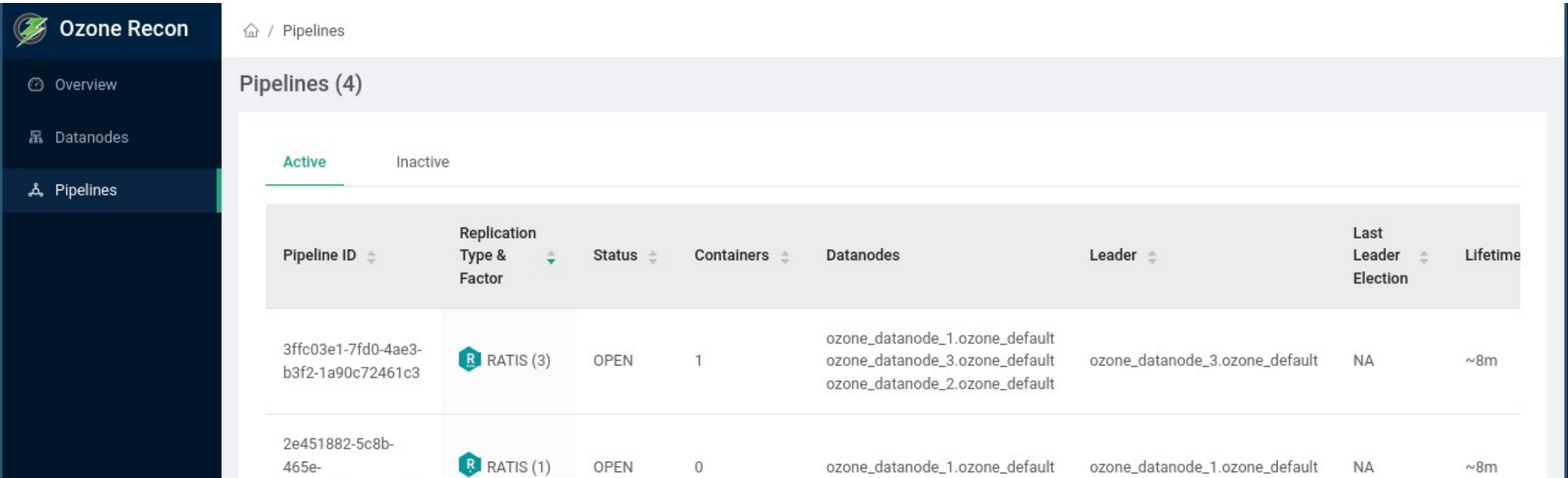
Observability

- Prometheus support
- Distributed tracing support
- Ozone Recon
  - Web UI, historical data, prediction
- Developer / admin tools
  - “What’s going on”?

Recon

- Separated component with SQL backend
- Stores historical data
  - From prometheus
  - Snapshots from other services
  - Receives Datanode heartbeats with reports
- Can help to debug
- Can predict problems (space usage?)
- Can help to understand the current state


Ozone Recon
Overview
Datanodes
Pipelines
/ Overview
3/3 HEALTHY
Datanodes
4
Pipelines
1.33 TB/1.46 ...
91%
Cluster Capacity
1
Containers
1
Volumes
1
Buckets
110
Keys


Ozone Recon
Overview
Datanodes
Pipelines
/ Pipelines
Pipelines (4)
Active
Inactive
Pipeline ID
Replication Type & Factor
Status
Containers
Datanodes
Leader
Last Leader Election
Lifetime
3ffc03e1-7fd0-4ae3-b3f2-1a90c72461c3
RATIS (3)
OPEN
1
ozone_datanode_1.ozone_default
ozone_datanode_3.ozone_default
ozone_datanode_2.ozone_default
ozone_datanode_3.ozone_default
NA
~8m
2e451882-5c8b-465e-
RATIS (1)
OPEN
0
ozone_datanode_1.ozone_default
ozone_datanode_1.ozone_default
NA
~8m

Ozone Insight

- Command line tool to make debug easier
- Profiles to define
  - metrics
  - logs
  - configuration

Testing

Levels of testing

- Acceptance tests
  - Each PR is tested with full secure / unsecure clusters
- MiniOzoneChaos test
- Freon: load generator
- Functional tests
  - TPC-DS with Spark
  - HBase test
- Fault Injection
  - Injected errors with Fuse File system
- 1 billion object key

Kubernetes support

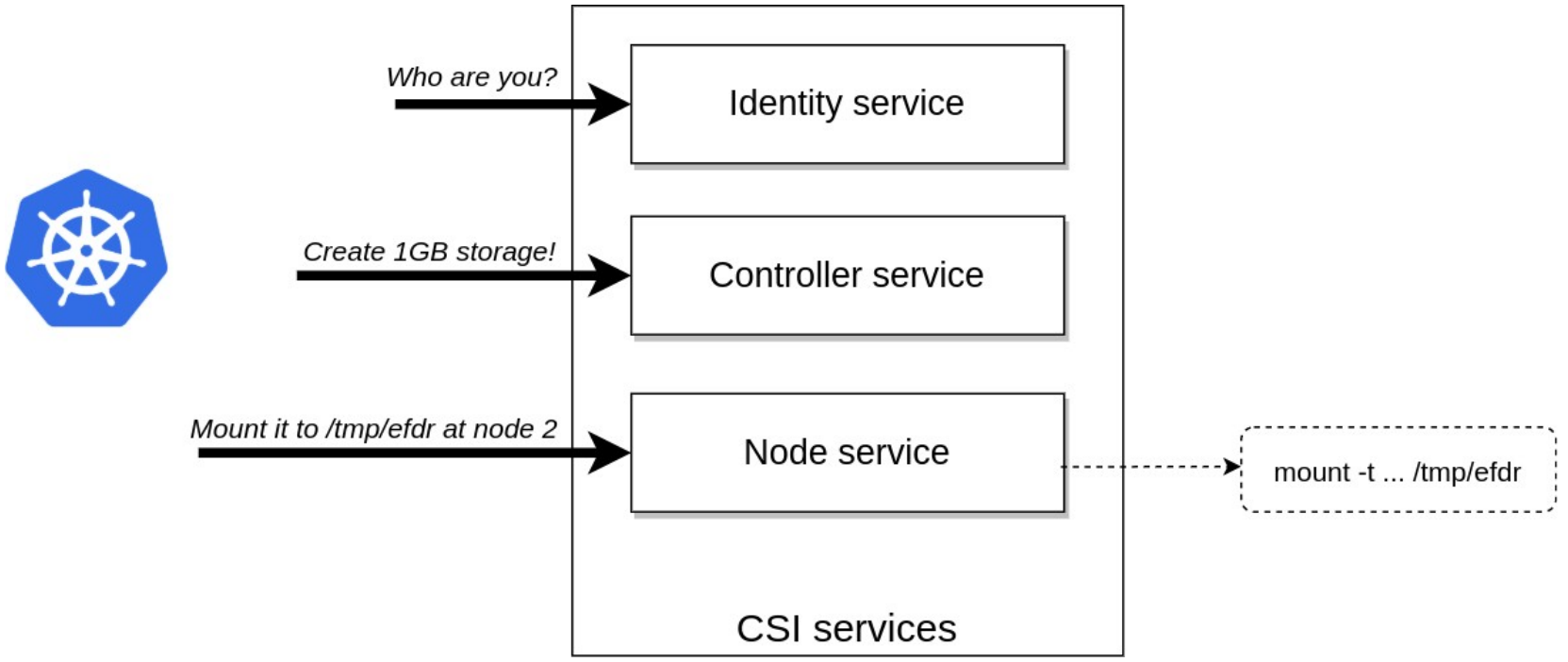
Kubernetes support

- Ozone ♥ Kubernetes
  - Ozone can be started quickly in K8s
  - Easy way to run chaos / performance testing
  - Tested in k8s / multiple configuration sets are provided
- Kubernetes ♥ Ozone
  - Ozone provides storage for container orchestrator
  - ozone csi service is a CSI service implementation

CSI

- Container Storage Interface
  - Standard to provide storage for container orchestrator
  - Supported by Yarn, Kubernetes, Mesos, etc...
  - A common language to request and mount storage
  - Ozone CSI daemon

CSI


Who are you?
Identity service
Create 1GB storage!
Controller service
Mount it to /tmp/efdr at node 2
Node service
mount -t ... /tmp/efdr
CSI services

Use Ozone Storage as a FS

- CSI is the easy part:
  - Create storage
  - Mount storage to a specific dir
- Hard part: how to mount?

CSI: Mount options

- Use S3 Fuse driver
  - goofys → Ozone S3 gateway → Ozone cluster
  - NFS → Ozone NFS gateway → Ozone cluster
  - fuse driver → libhdfs → OzoneFileSystem → ...

Ozone Operator?

- Ozone has native Kubernetes support
  - It doesn’t require to use an operator
- Operator pattern doesn’t do gitops
  - Most of the use cases can be covered by better tools

Summary

Other implemented features

- TDA: encrypt data in rest
  - Very similar to HDFS
- GDPR support
  - Right to be forgotten: Similar to TDE, but delete encryption key
- Hadoop 2.7+ support (classoader magic)

Not (yet) implemented

- Ereasure Coding
- In-place Hdfs upgrade (planned)

Summary

- Scalable (1 billion keys)
- Multiple interfaces (Hadoop, S3, CSI)
- Cloud native

aws S3 protocol

Hadoop FS

CSI

Apache Hadoop Ozone

hadoop.apache.org/ozone

THE APACHE SOFTWARE FOUNDATION