github.com/hopshadoop

www.hops.io

@hopshadoop

HopsFS & ePipe

Mahmoud Ismail <maism@kth.se>
Gautier Berthou <gautier@sics.se>

Agenda

- From HDFS to HopsFS
- ePipe to enable a searchable HopsFS
- Tutorial

A File System with a million ops/sec and searchable in sub seconds?!

Bill Gates’ biggest product regret?

next

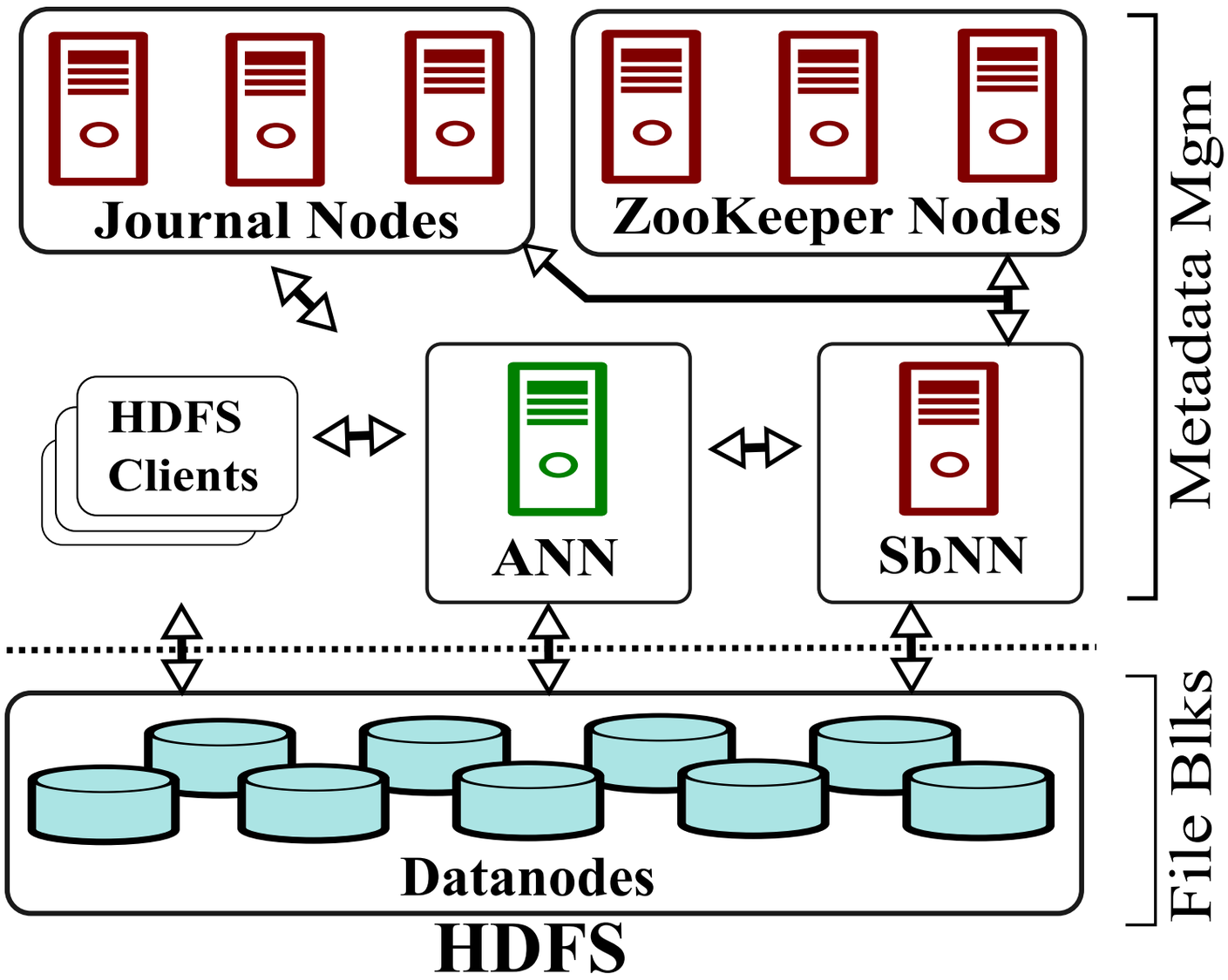
WinFS

- *"WinFS was an attempt to bring the benefits of schema and relational databases to the Windows file system. …The WinFS effort was started around 1999 as the successor to the planned storage layer of Cairo and died in 2006 after consuming many thousands of hours of efforts from really smart engineers."*

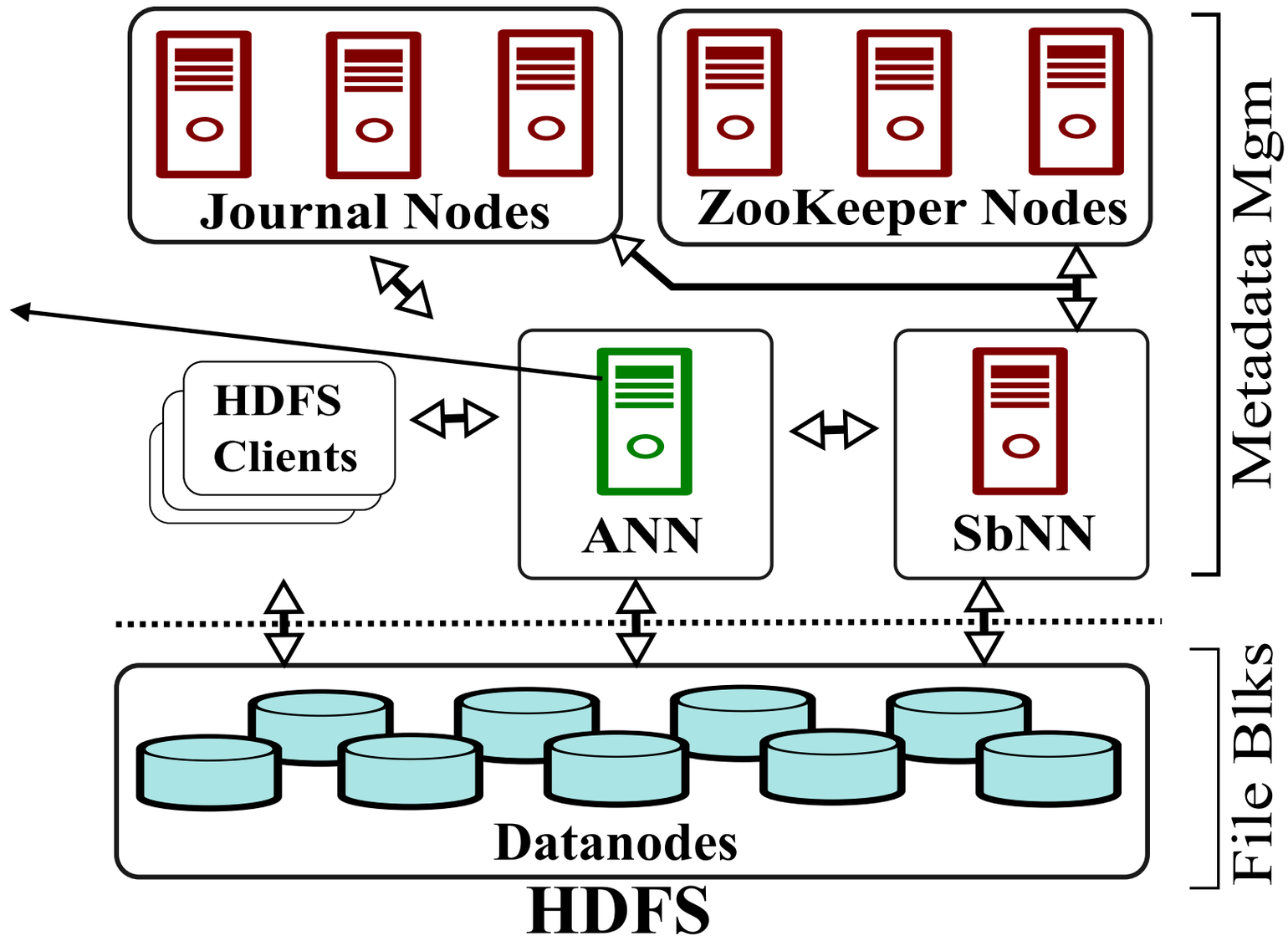
  - [Brian Welcker]*

*http://www.zdnet.com/article/bill-gates-biggest-microsoft-product-regret-winfs/

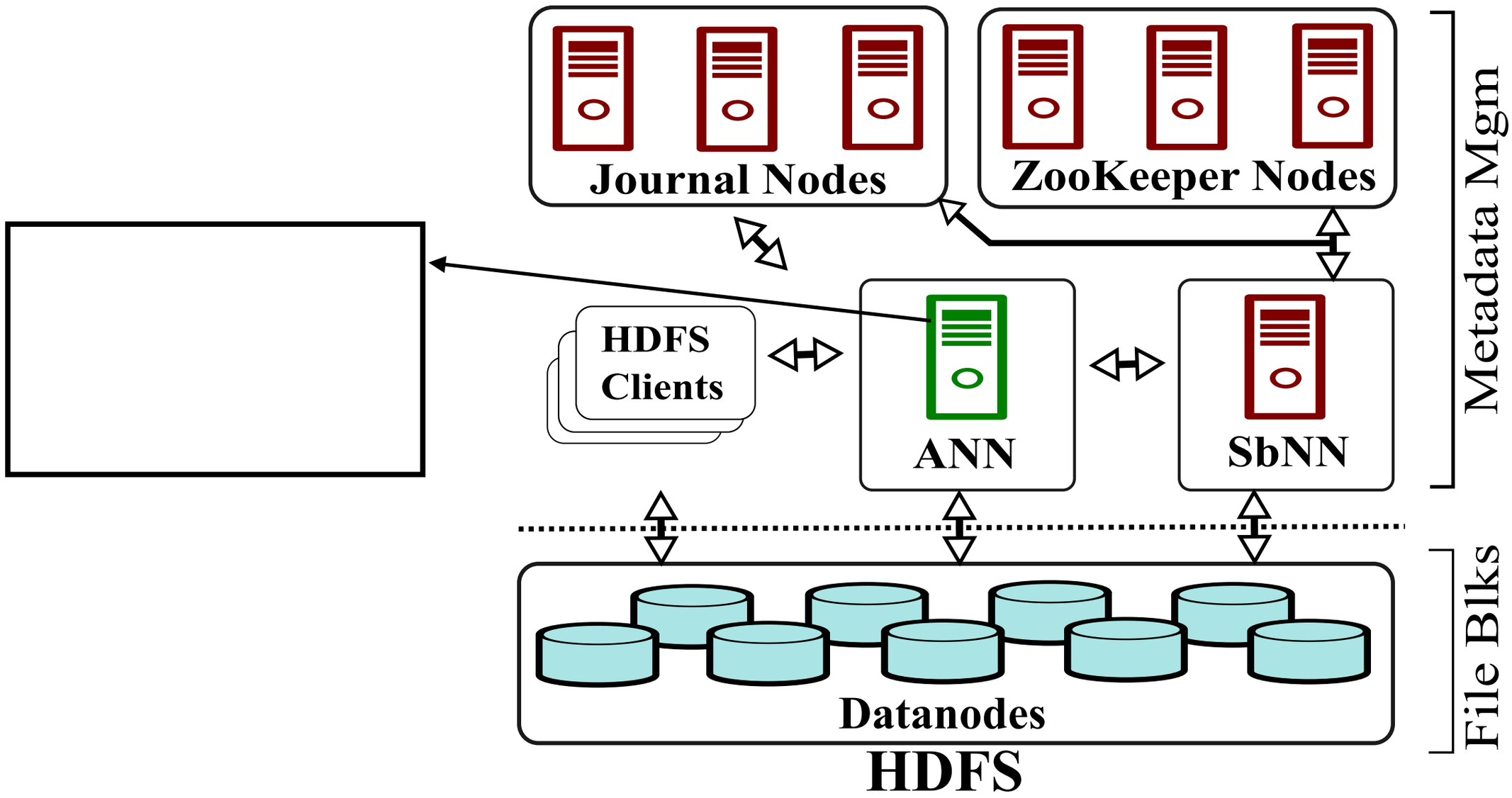
HDFS


Journal Nodes
ZooKeeper Nodes
Metadata Mgm
HDFS Clients
ANN
SbNN
File Blks
Datanodes
HDFS

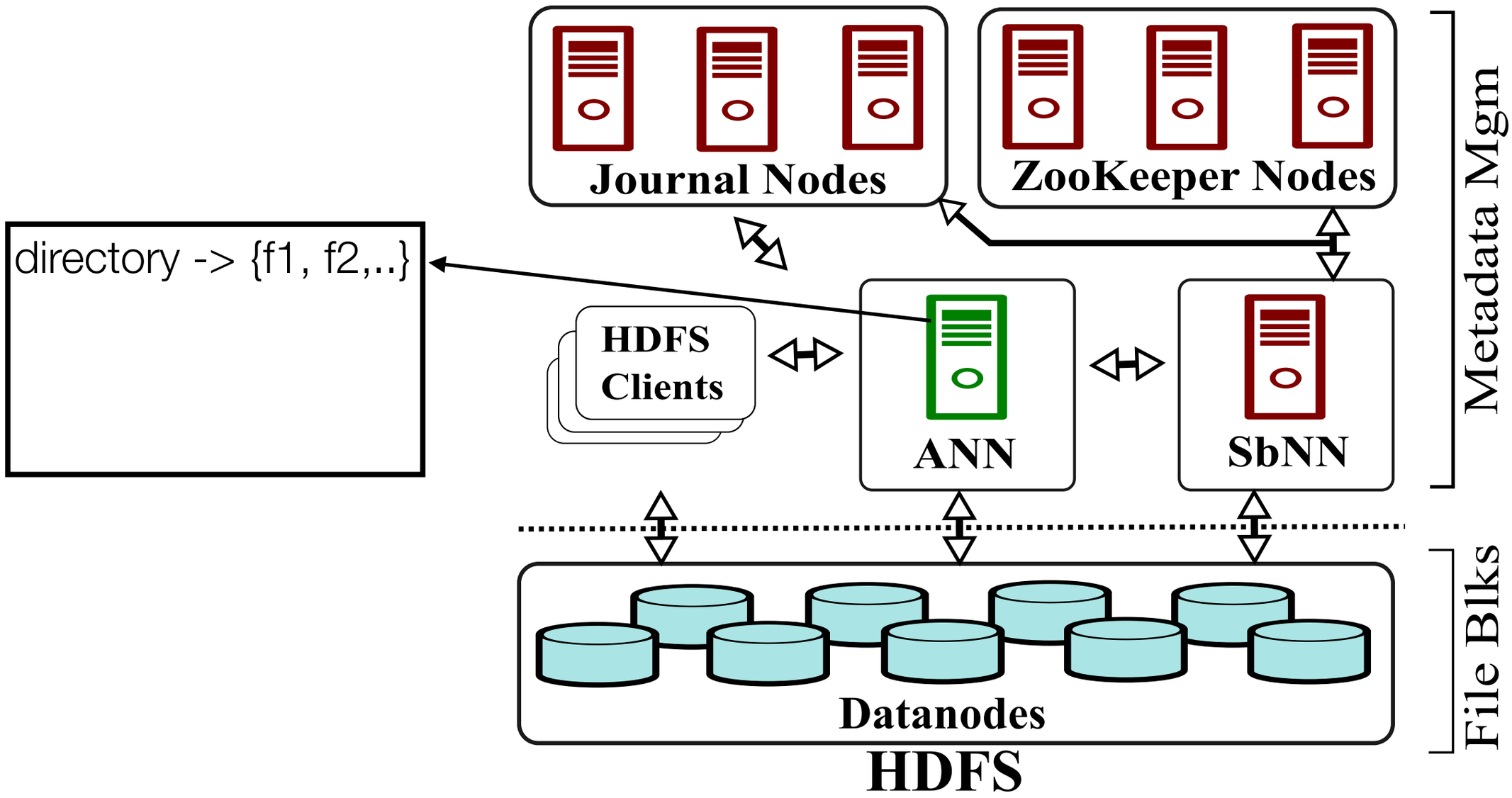
HDFS


Journal Nodes
ZooKeeper Nodes
Metadata Mgm
HDFS Clients
ANN
SbNN
File Blks
Datanodes
HDFS

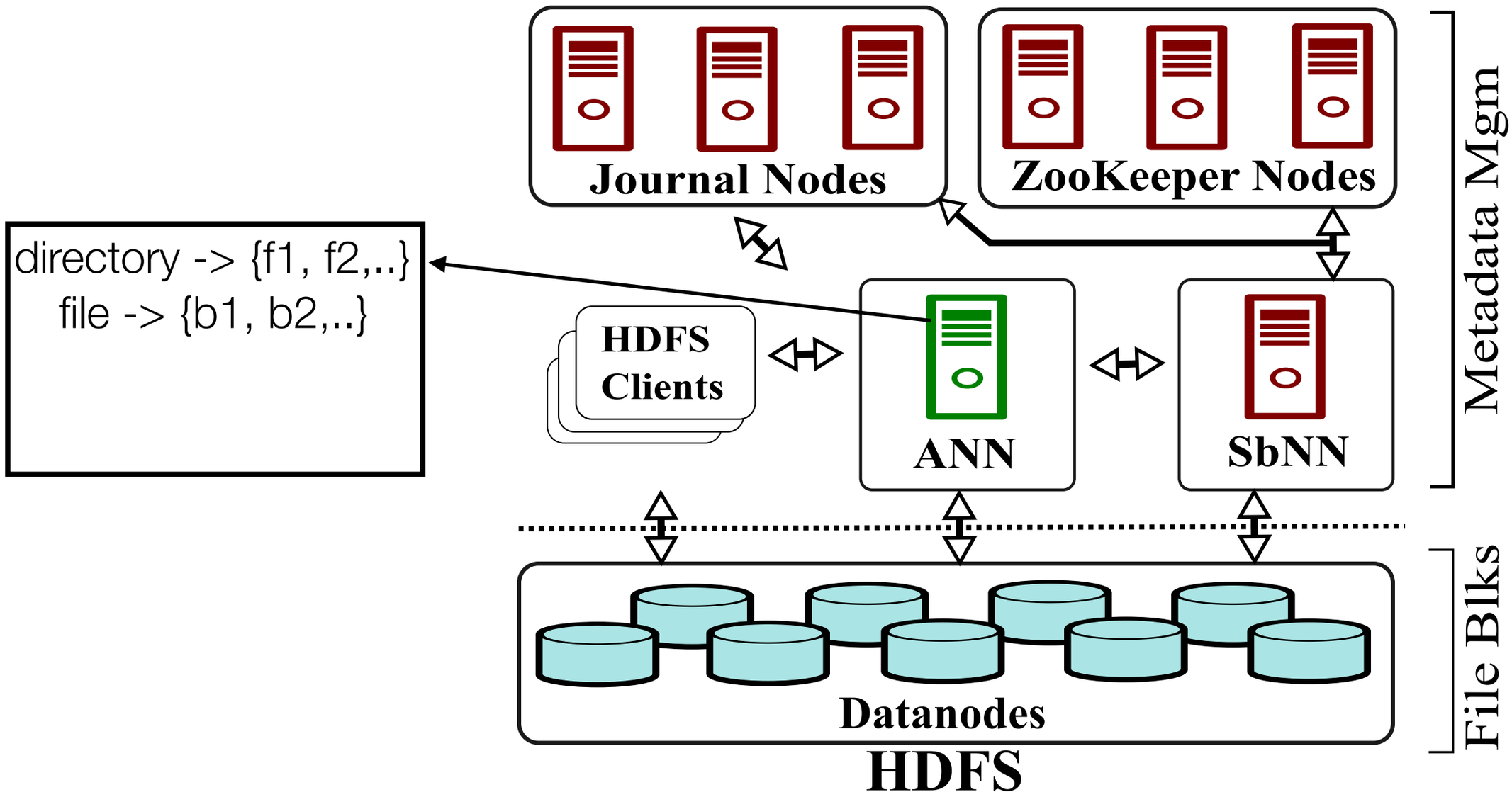
HDFS


Journal Nodes
ZooKeeper Nodes
Metadata Mgm
HDFS Clients
ANN
SbNN
Datanodes
File Blks
HDFS

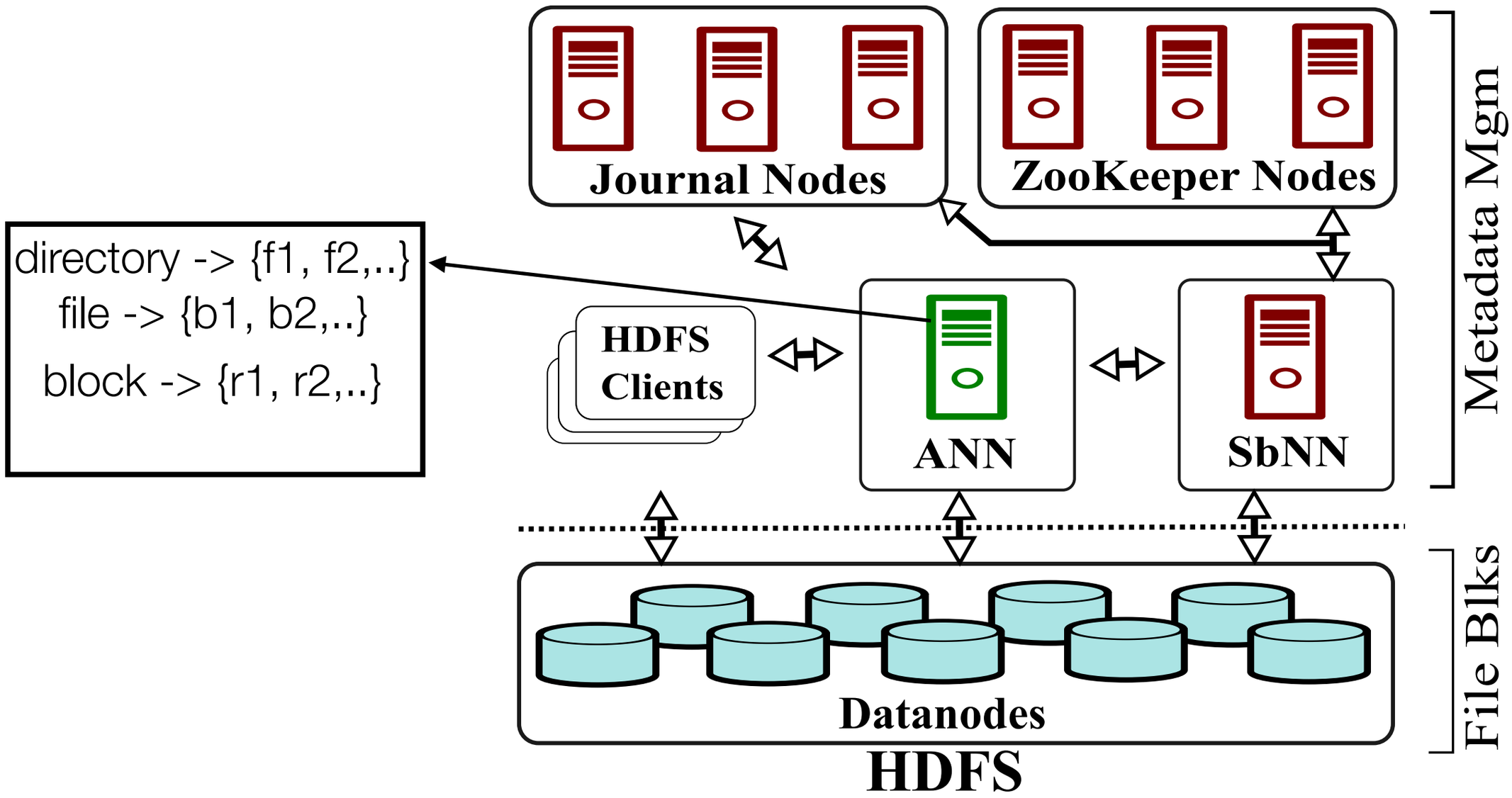
HDFS


directory -> {f1, f2,..}
Journal Nodes
ZooKeeper Nodes
Metadata Mgm
HDFS Clients
ANN
SbNN
File Blks
Datanodes
HDFS

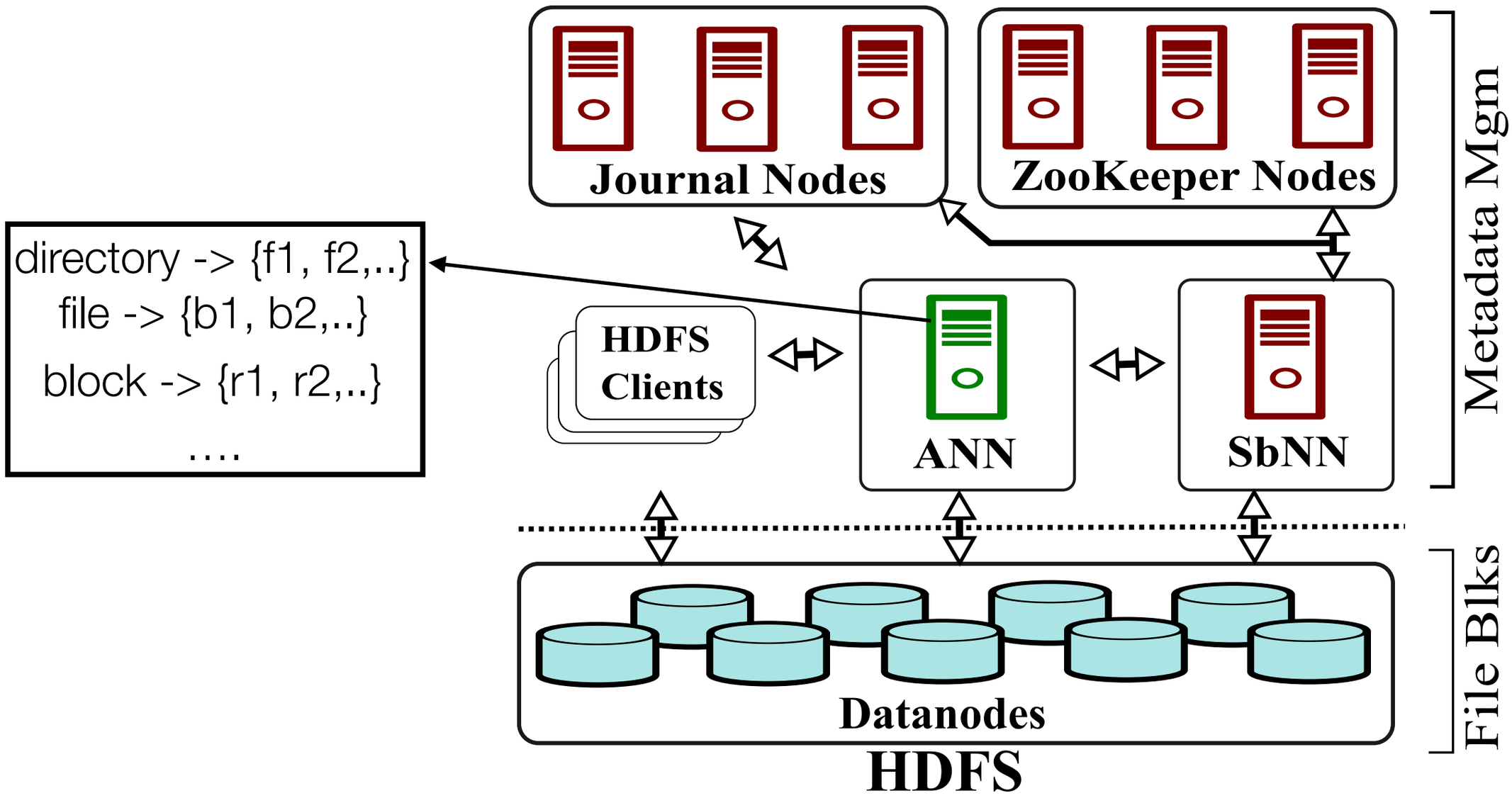
HDFS


Journal Nodes
ZooKeeper Nodes
Metadata Mgm
directory -> {f1, f2,..}
file -> {b1, b2,..}
HDFS Clients
ANN
SbNN
Datanodes
File Blks
HDFS

HDFS


directory -> {f1, f2,..}
file -> {b1, b2,..}
block -> {r1, r2,..}
Journal Nodes
ZooKeeper Nodes
HDFS Clients
ANN
SbNN
Metadata Mgm
Datanodes
File Blks
HDFS

HDFS


directory -> {f1, f2,..}
file -> {b1, b2,..}
block -> {r1, r2,..}
....
Journal Nodes
ZooKeeper Nodes
HDFS Clients
ANN
SbNN
Metadata Mgm
Datanodes
File Blks
HDFS

JVM Heap is the limit

JVM Heap is the limit

- Storing NameNode metadata in JVM Heap

JVM Heap is the limit

- Storing NameNode metadata in JVM Heap
- Very efficient, yet

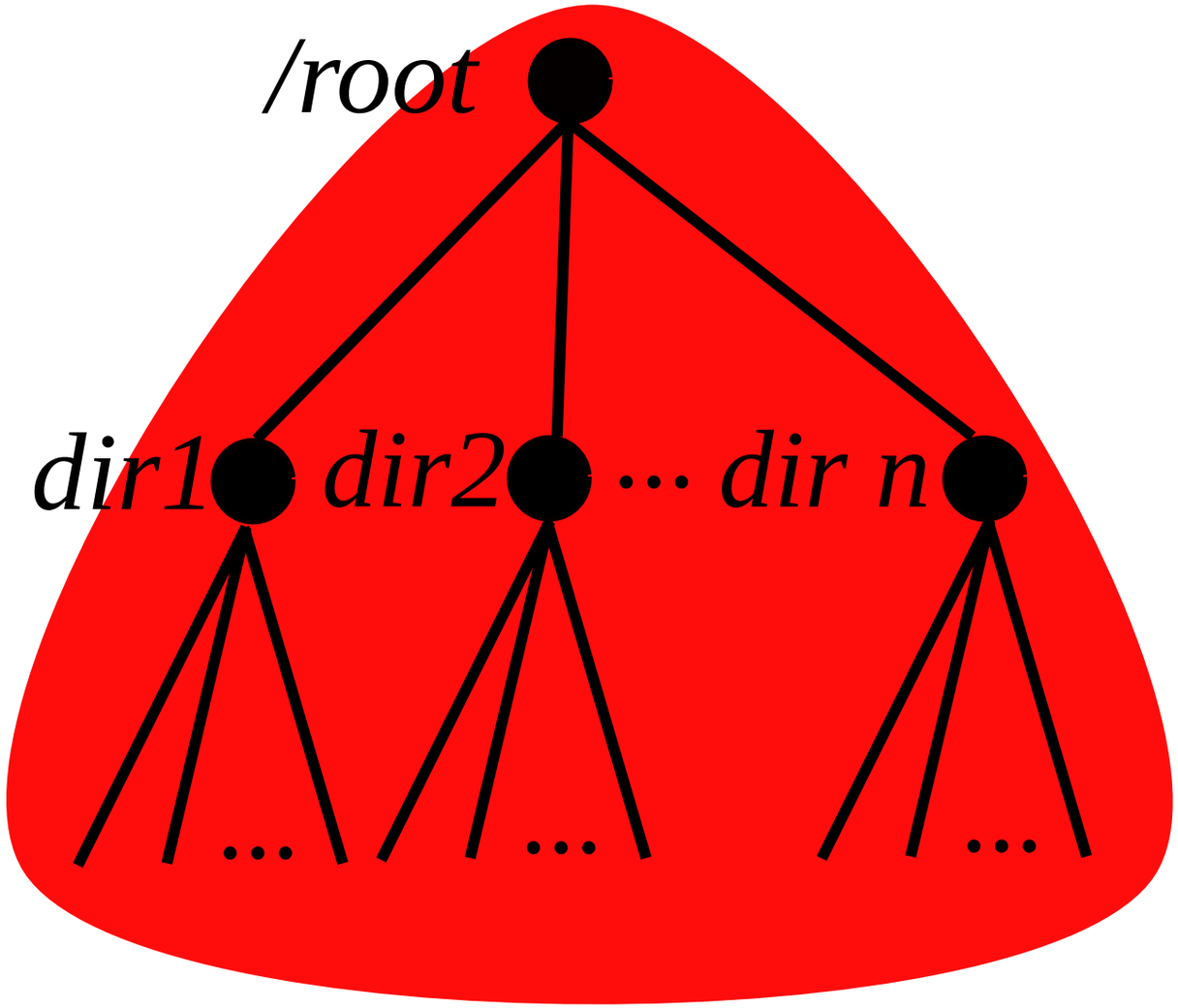
JVM Heap is the limit

- Storing NameNode metadata in JVM Heap
- Very efficient, yet
- Number of files/directories are limited

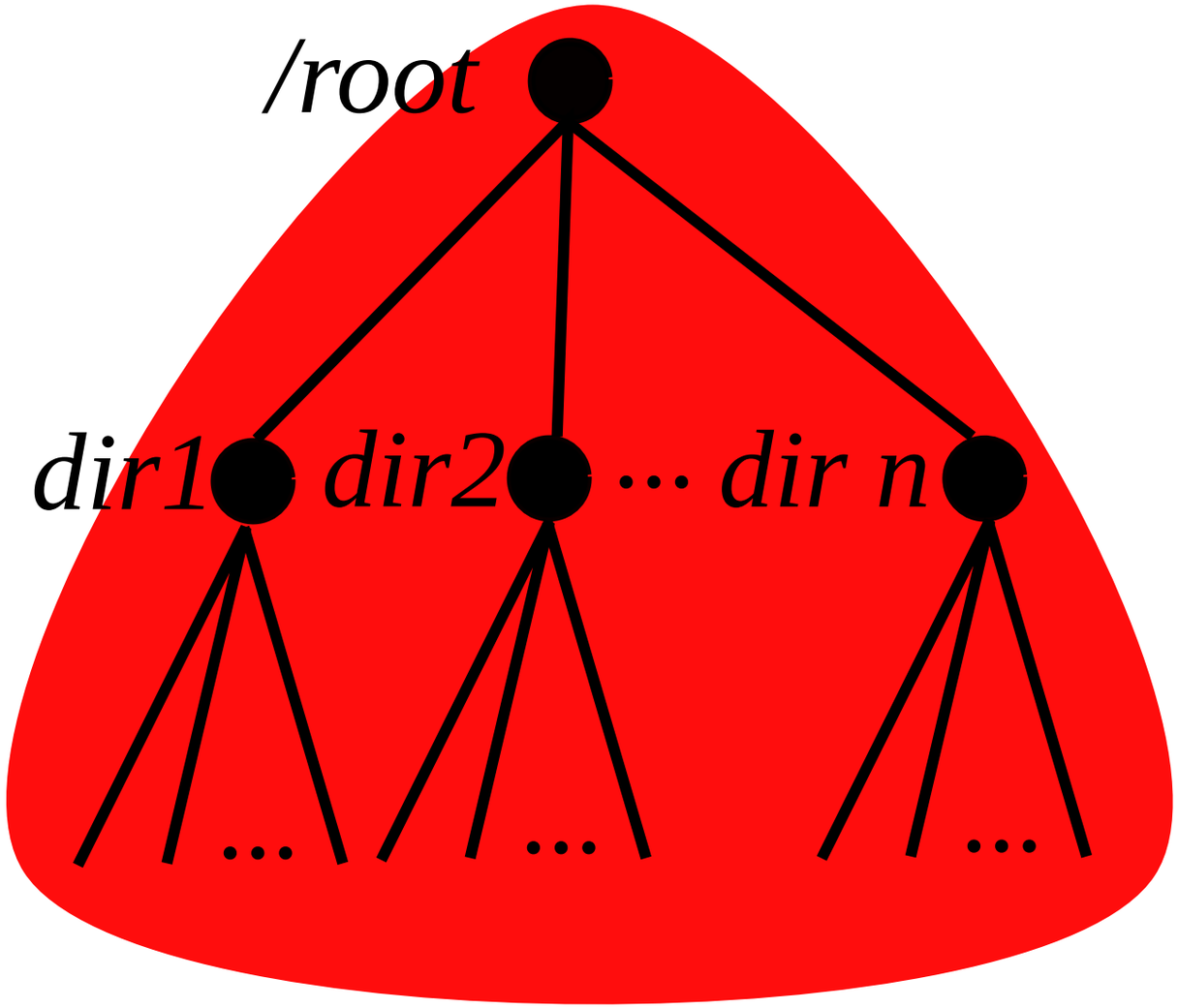
JVM Heap is the limit

- Storing NameNode metadata in JVM Heap
- Very efficient, yet
- Number of files/directories are limited
- Garbage collection pause times

One Lock to rule them all



One Lock to rule them all



multi-reader, single writer concurrency semantics

Move the NameNode metadata off the JVM Heap

Move the NameNode metadata off the JVM Heap

- Stateless NameNode

Move the NameNode metadata off the JVM Heap

- Stateless NameNode
- Multiple NameNodes to increase Throughput

Move the NameNode metadata off the JVM Heap

- Stateless NameNode
- Multiple NameNodes to increase Throughput
- Throughput dependent on our chosen data store

Move the NameNode metadata off the JVM Heap

- Stateless NameNode
- Multiple NameNodes to increase Throughput
- Throughput dependent on our chosen data store
- Choosing a data store?

Move the NameNode metadata off the JVM Heap

- Stateless NameNode
- Multiple NameNodes to increase Throughput
- Throughput dependent on our chosen data store
- Choosing a data store?
  - An in-memory storage system that can be efficiently queried and managed. Preferably Open-Source.

Move the NameNode metadata off the JVM Heap

- Stateless NameNode
- Multiple NameNodes to increase Throughput
- Throughput dependent on our chosen data store
- Choosing a data store?
  - An in-memory storage system that can be efficiently queried and managed. Preferably Open-Source.
  - Row-level locking

Move the NameNode metadata off the JVM Heap

- Stateless NameNode
- Multiple NameNodes to increase Throughput
- Throughput dependent on our chosen data store
- Choosing a data store?
  - An in-memory storage system that can be efficiently queried and managed. Preferably Open-Source.
  - Row-level locking
  - Efficient Cross partition transaction

MySQL Cluster (NDB) to the rescue

MySQL Cluster (NDB) to the rescue

- **NewSQL (Relational) DB**
  - User-defined partitioning
  - Row-level Locking
  - Distribution-aware transactions
  - Partition-pruned index scans
- **Real-time, 2-Phase Commit**
  - 1.2 sec TransactionInactive timeouts

MySQL Cluster (NDB) to the rescue

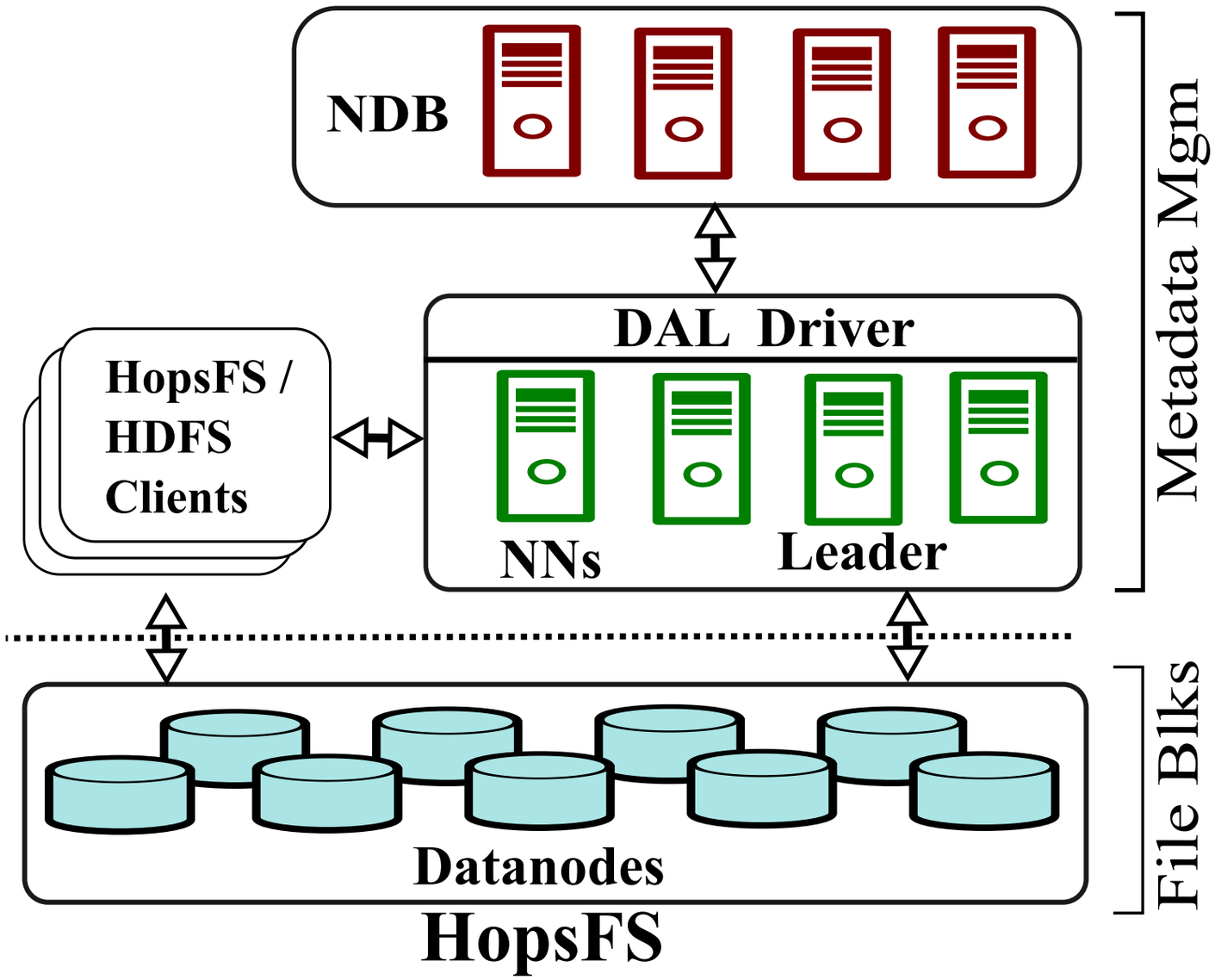
- **NewSQL (Relational) DB**
  - User-defined partitioning
  - Row-level Locking
  - Distribution-aware transactions
  - Partition-pruned index scans
- **Real-time, 2-Phase Commit**
  - 1.2 sec TransactionInactive timeouts
- **Commodity Hardware**
  - Scales to 48 nodes
  - Supports on-disk columns
- **SQL API**
- **C++/Java Native API**
- **C++ Event API**

MySQL Cluster (NDB) to the rescue

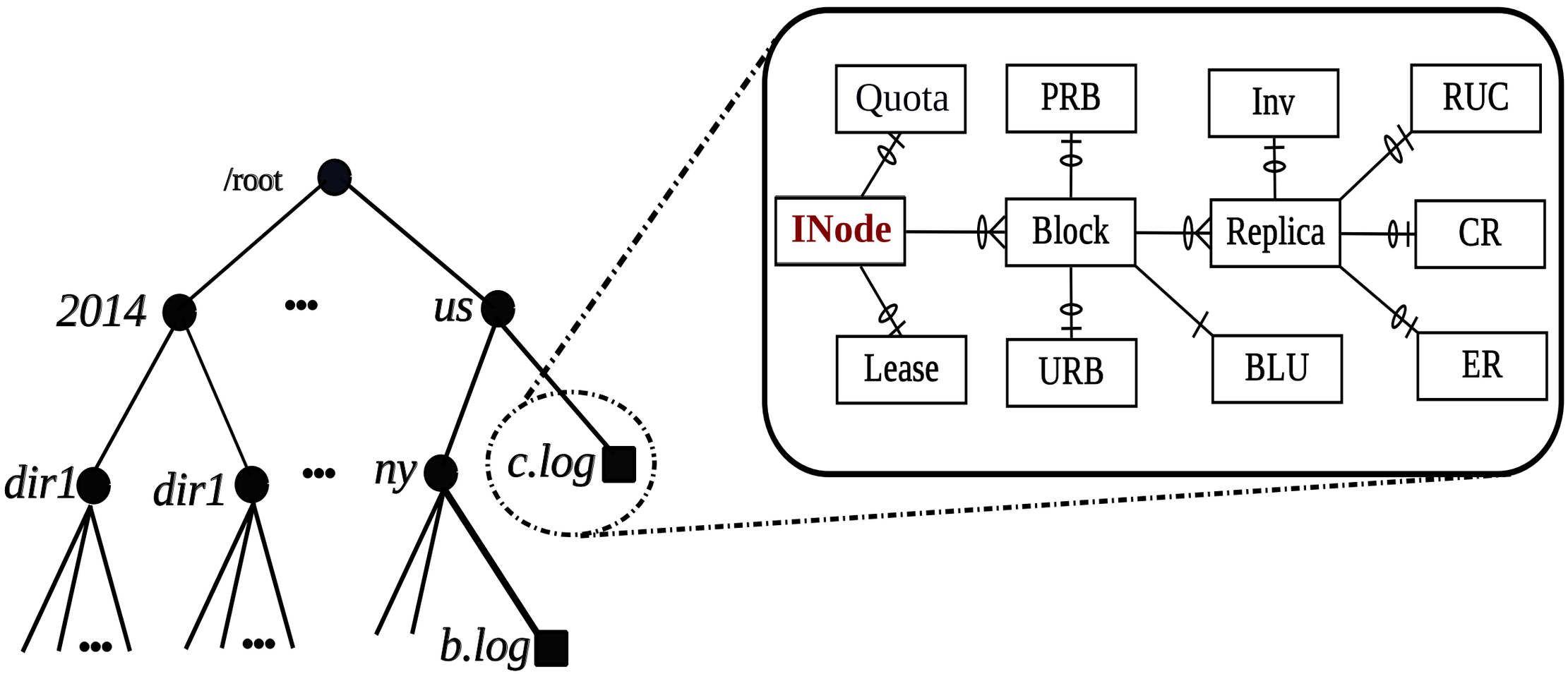
- **NewSQL (Relational) DB**
  - User-defined partitioning
  - Row-level Locking
  - Distribution-aware transactions
  - Partition-pruned index scans
- **Real-time, 2-Phase Commit**
  - 1.2 sec TransactionInactive timeouts
- **Commodity Hardware**
  - Scales to 48 nodes
  - Supports on-disk columns
- **SQL API**
- **C++/Java Native API**
- **C++ Event API**

200 Million NoSQL Ops/sec

HopsFS

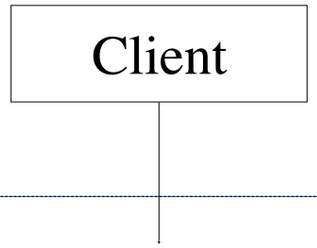

NDB
DAL Driver
HopsFS /
HDFS
Clients
NNs
Leader
Metadata Mgm
Datanodes
File Blks
HopsFS

From Memory to Database



Apache NameNode Internals

Client: mkdir, getblocklocations, createFile,…..

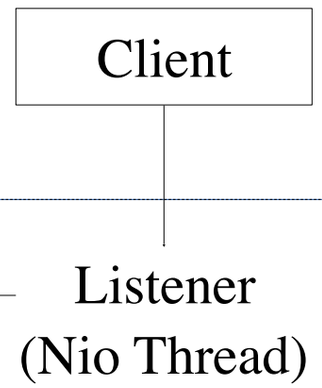


NameNode

Journal Nodes

Apache NameNode Internals

Client: mkdir, getblocklocations, createFile,…..



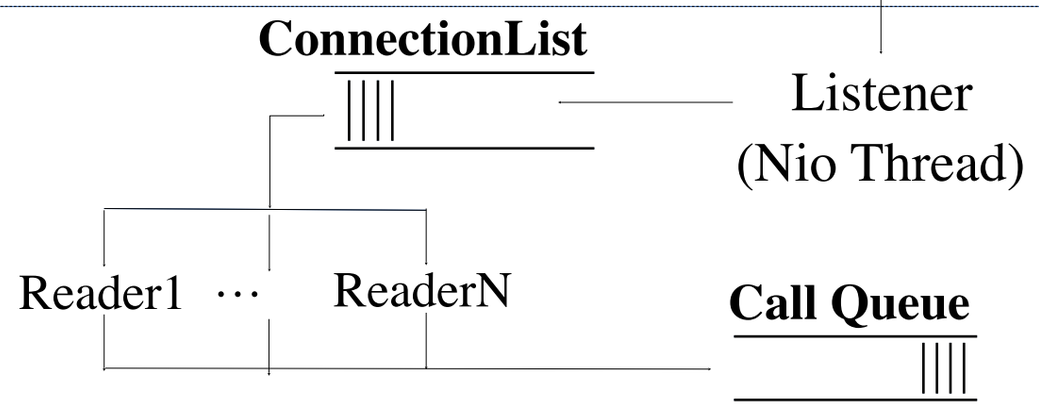
NameNode

Journal Nodes

Apache NameNode Internals

Client: mkdir, getblocklocations, createFile,…..

NameNode

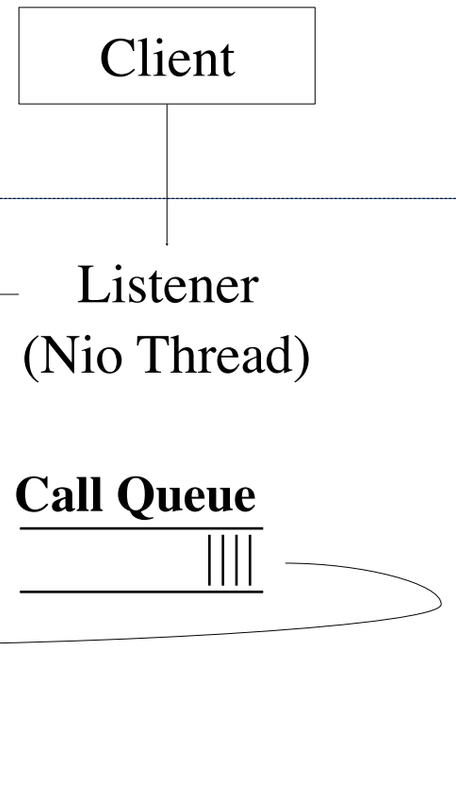


ipc.server.read.threadpool.size
(default 1)

Journal Nodes

Apache NameNode Internals

Client: mkdir, getblocklocations, createFile,…..



NameNode

ipc.server.read.threadpool.size
(default 1)

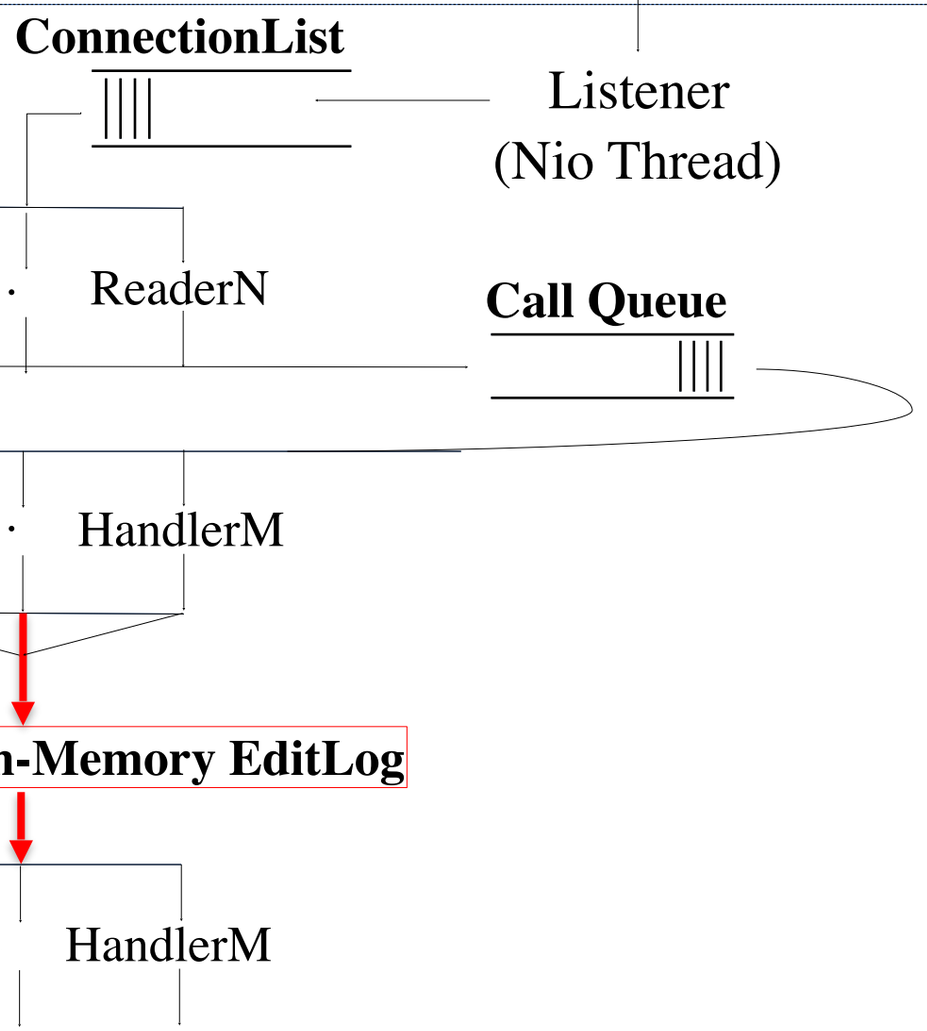
dfs.namenode.service.handlercount
(default 10)

Journal Nodes

Apache NameNode Internals

Client: mkdir, getblocklocations, createFile,…..

NameNode



ipc.server.read.threadpool.size
(default 1)

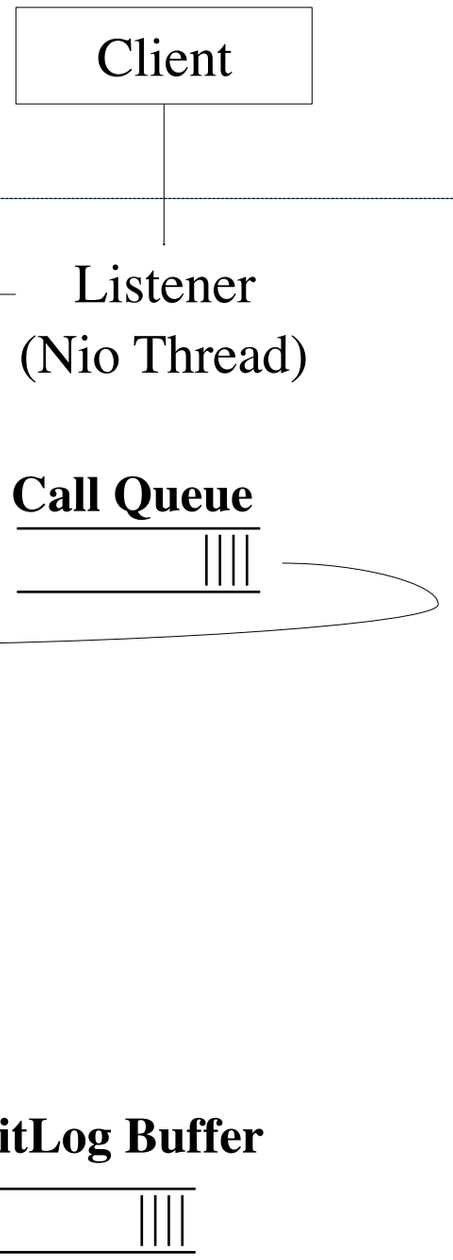
dfs.namenode.service.handlercount
(default 10)

FSNameSystem Lock

Journal Nodes

Apache NameNode Internals

Client: mkdir, getblocklocations, createFile,…..

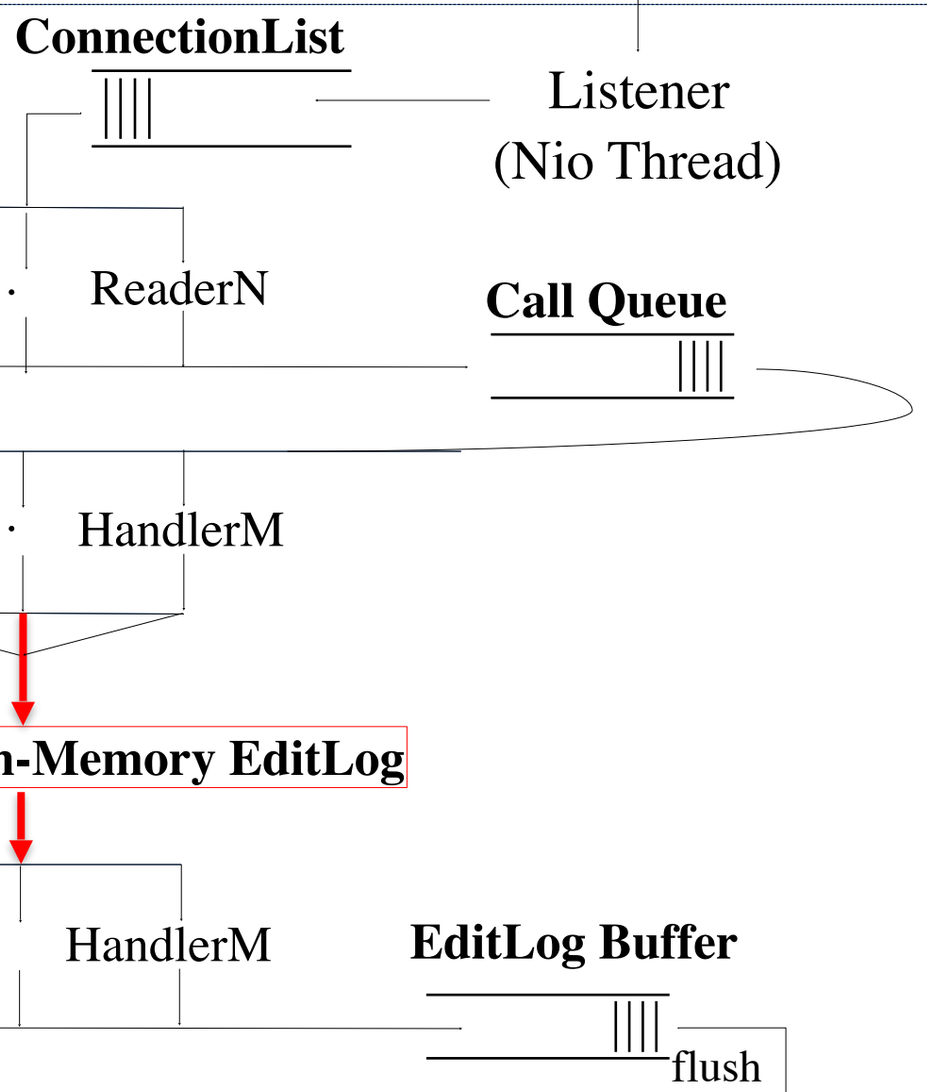


Journal Nodes

Apache NameNode Internals

Client: mkdir, getblocklocations, createFile,…..

NameNode



ipc.server.read.threadpool.size
(default 1)

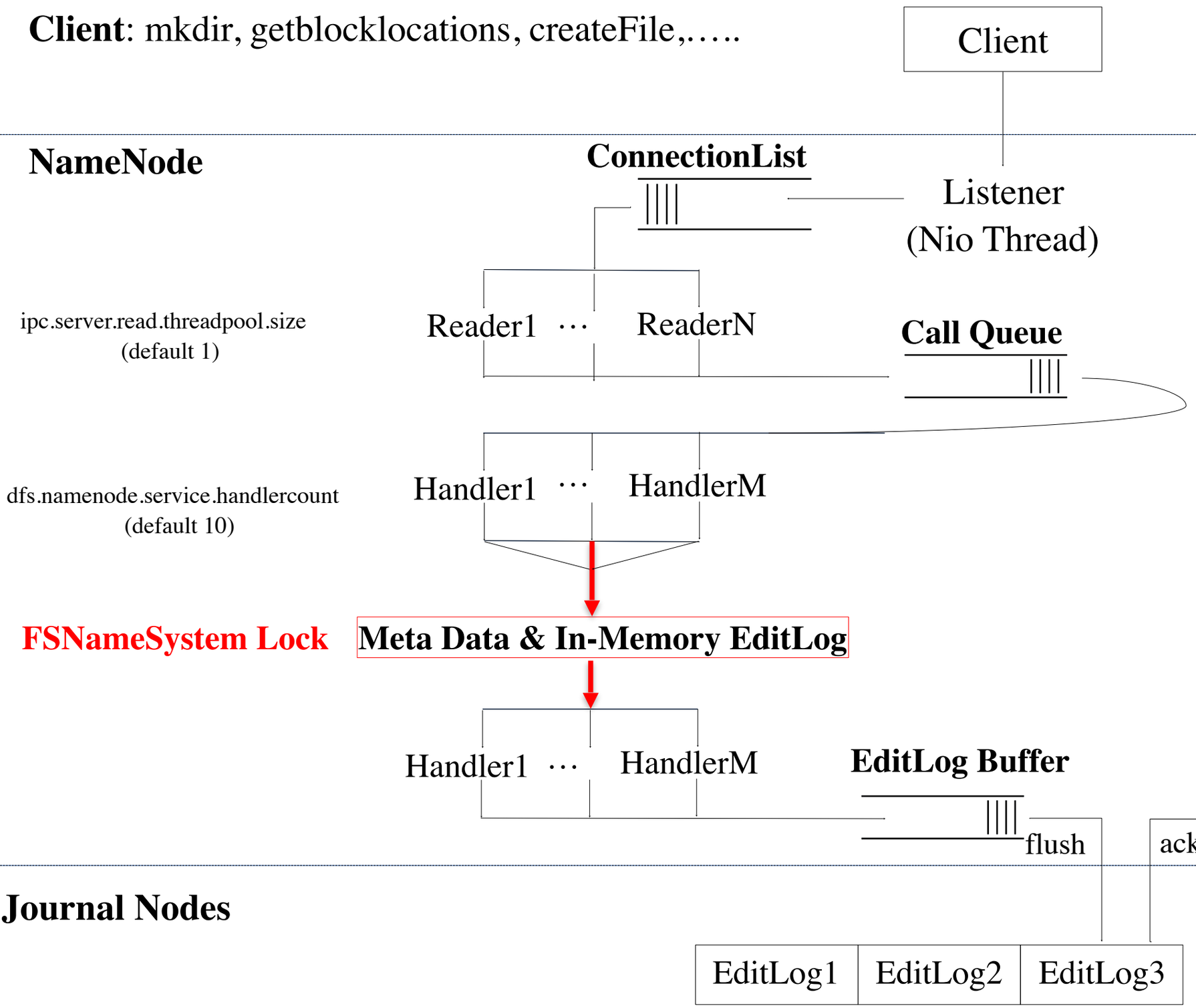
dfs.namenode.service.handlercount
(default 10)

FSNameSystem Lock

Journal Nodes

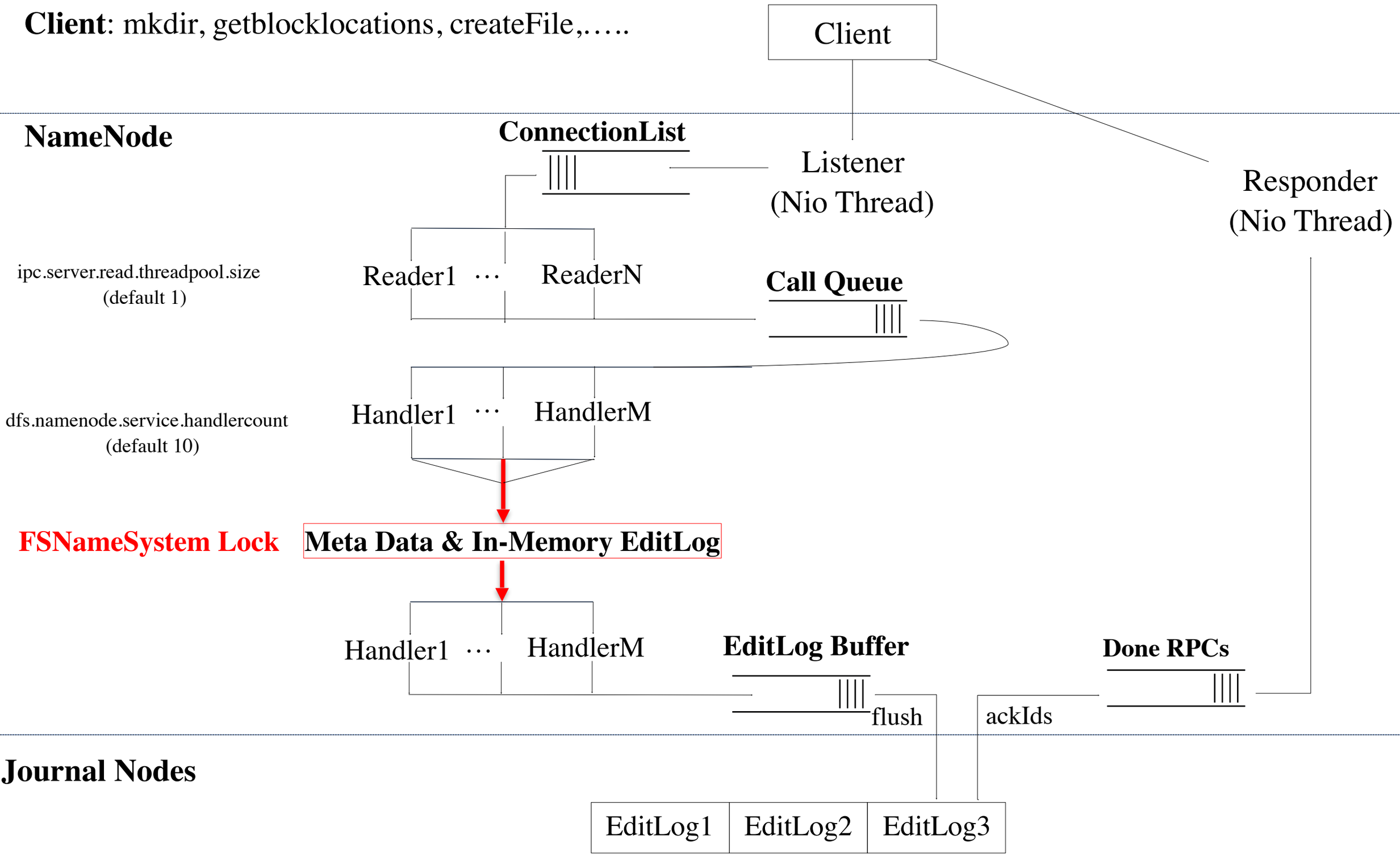
Apache NameNode Internals

Client: mkdir, getblocklocations, createFile,…..



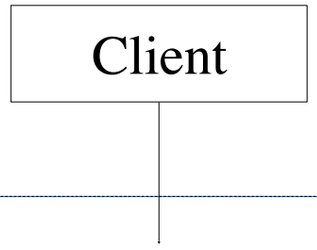
Apache NameNode Internals

Client: mkdir, getblocklocations, createFile,…..



HopsFS NameNode Internals

Client: mkdir, getblocklocations, createFile,…..



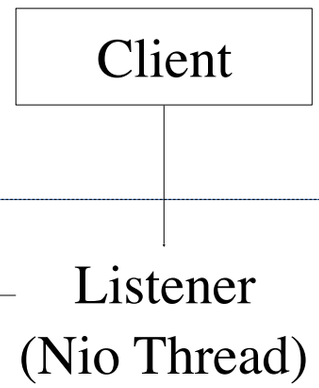
NameNode

DAL-Impl

NDB

HopsFS NameNode Internals

Client: mkdir, getblocklocations, createFile,…..



NameNode

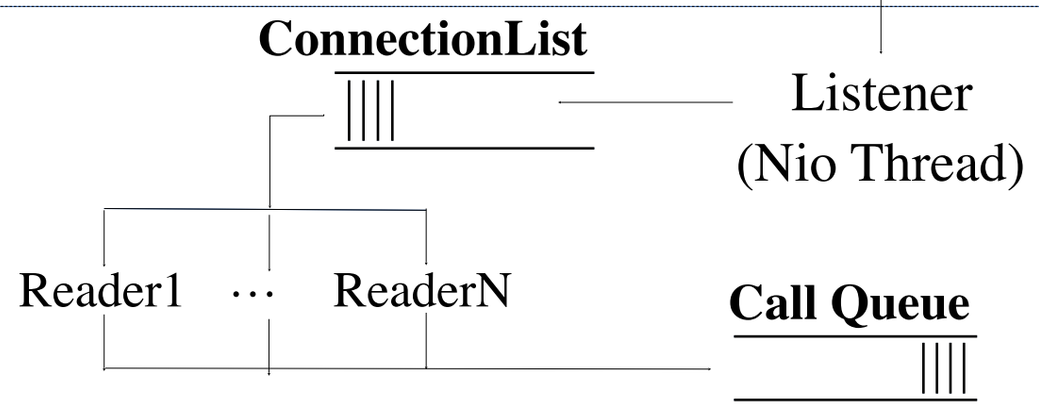
DAL-Impl

NDB

HopsFS NameNode Internals

Client: mkdir, getblocklocations, createFile,…..

NameNode



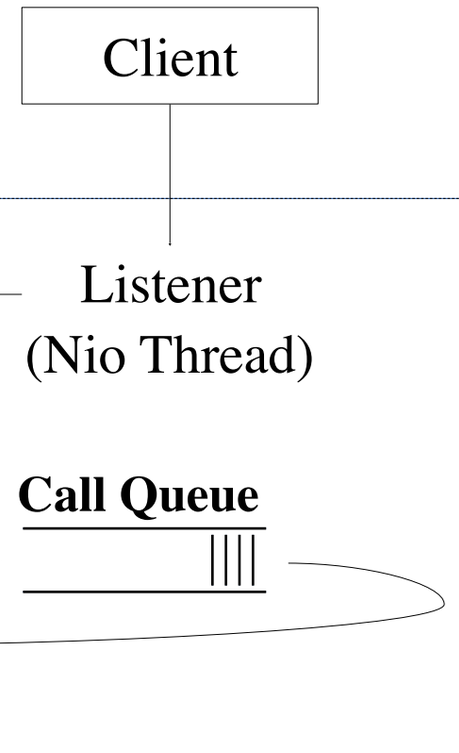
ipc.server.read.threadpool.size
(default 1)

DAL-Impl

NDB

HopsFS NameNode Internals

Client: mkdir, getblocklocations, createFile,…..



NameNode

ipc.server.read.threadpool.size
(default 1)

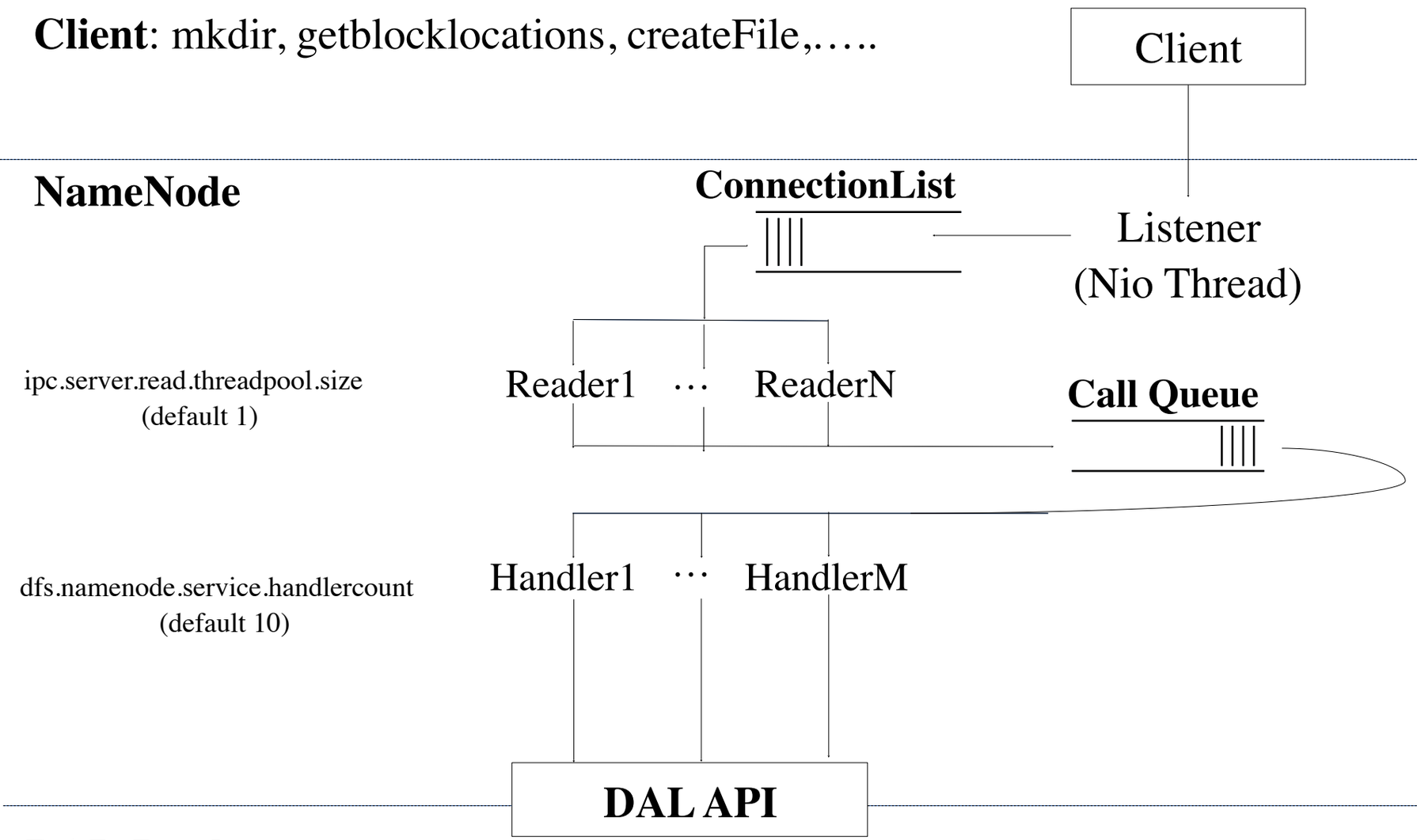
dfs.namenode.service.handlercount
(default 10)

DAL-Impl

NDB

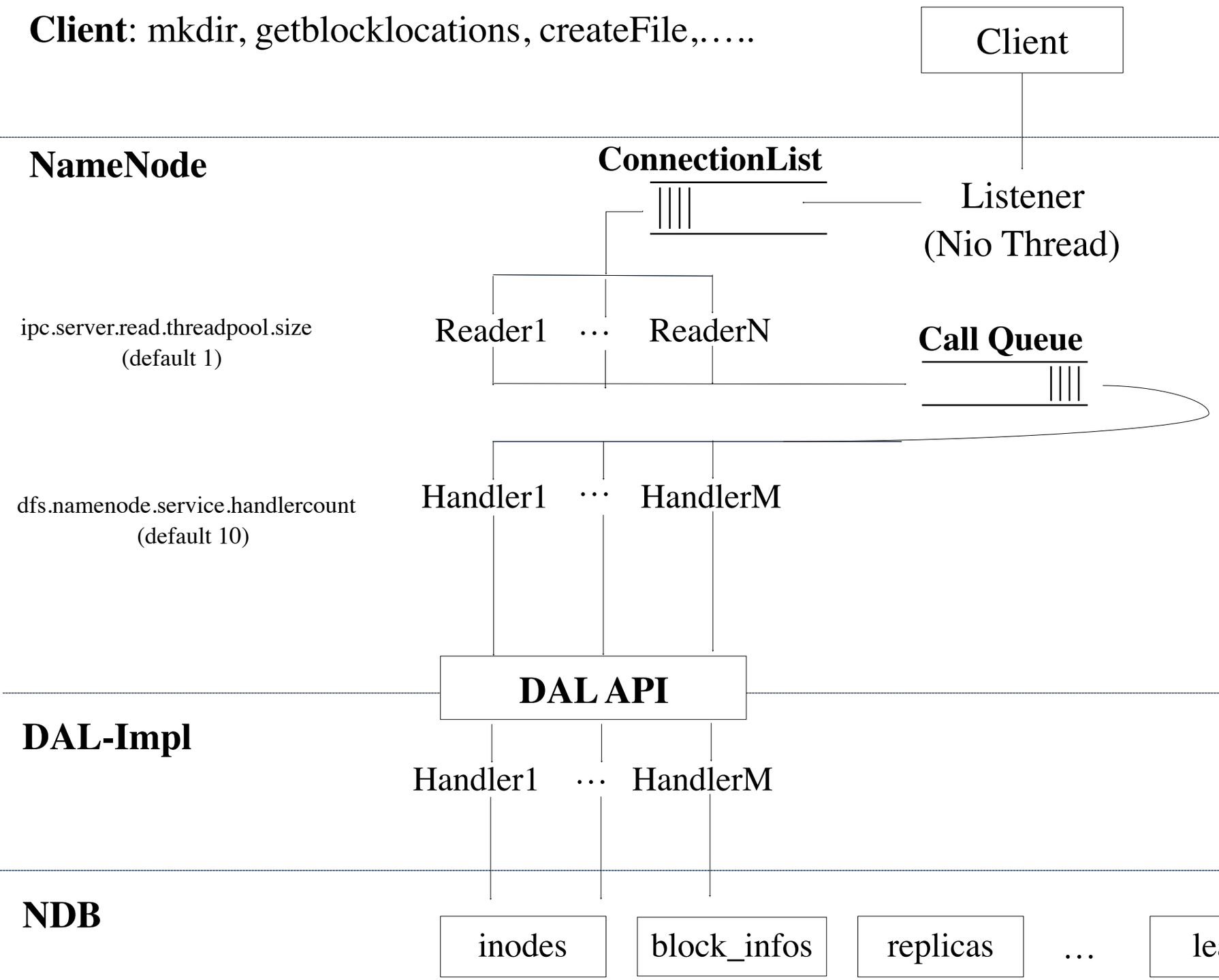
HopsFS NameNode Internals

Client: mkdir, getblocklocations, createFile,…..



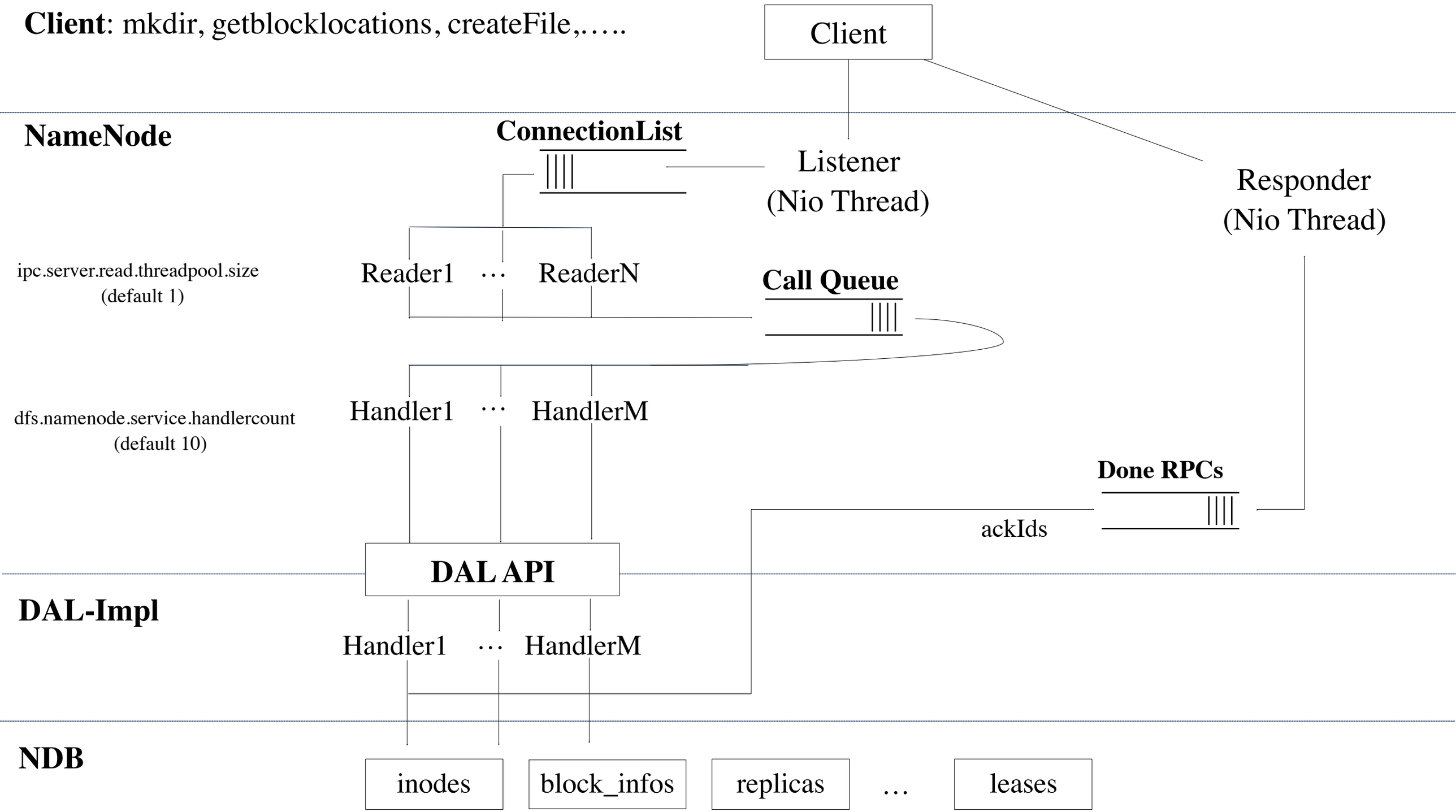
HopsFS NameNode Internals

Client: mkdir, getblocklocations, createFile,…..

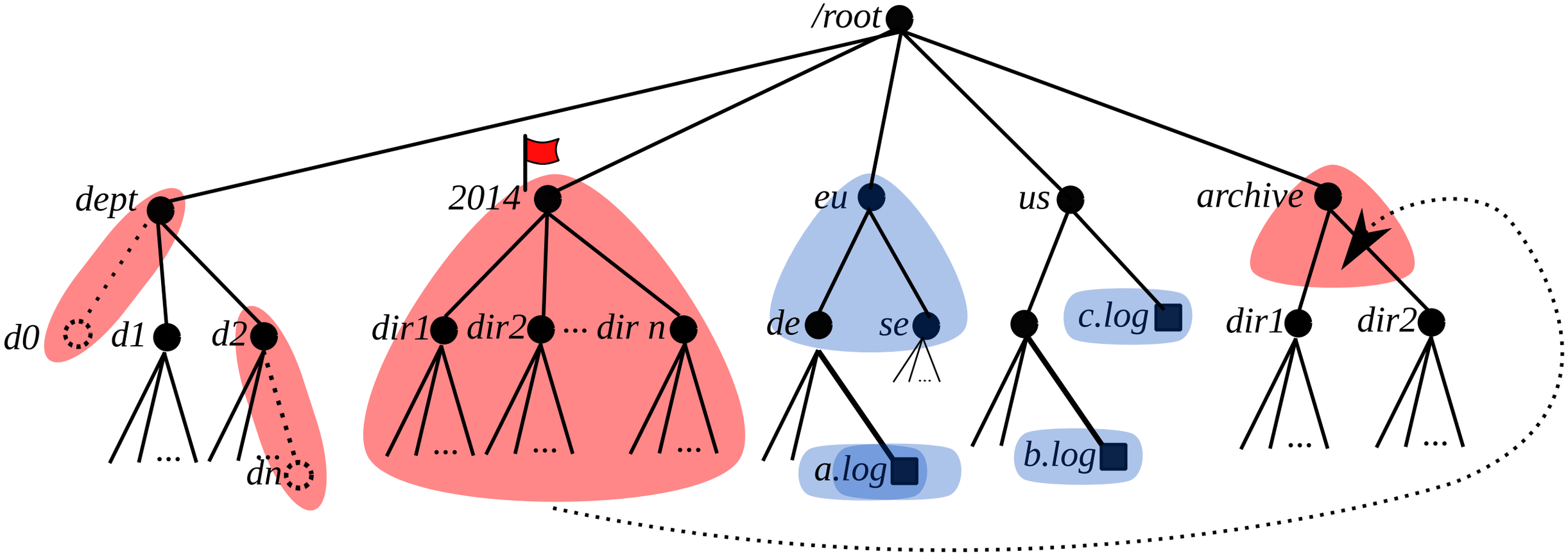


HopsFS NameNode Internals

Client: mkdir, getblocklocations, createFile,…..

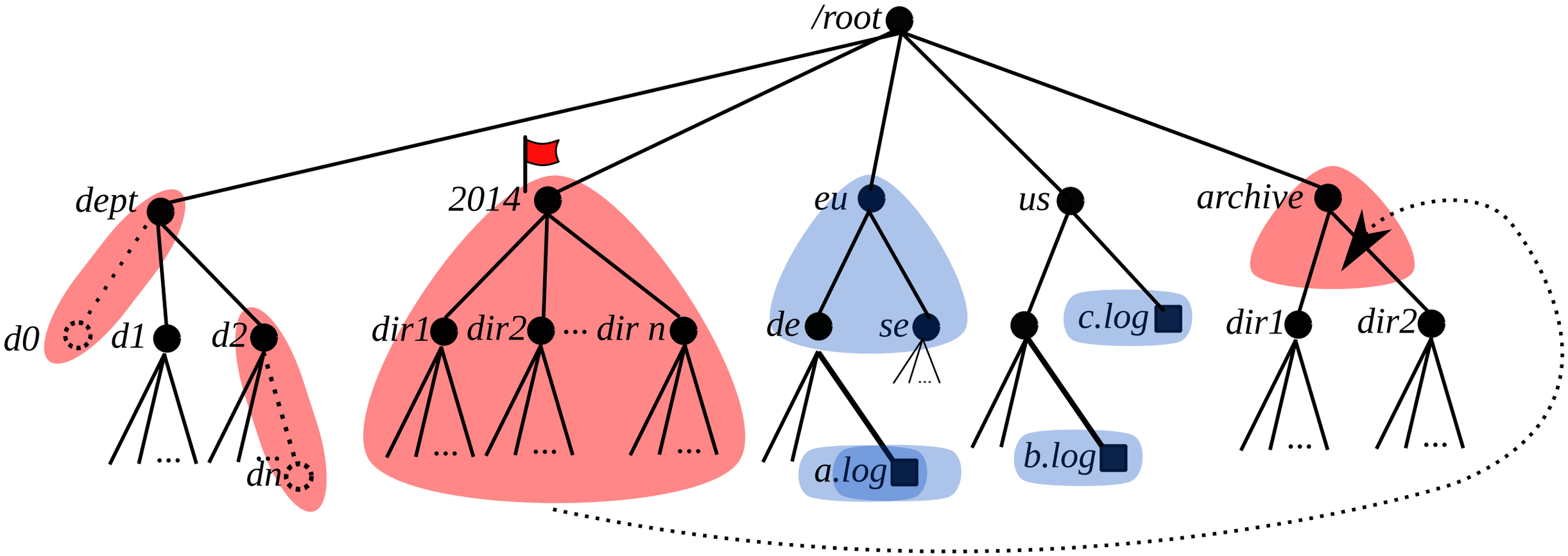


Fine grain Locking



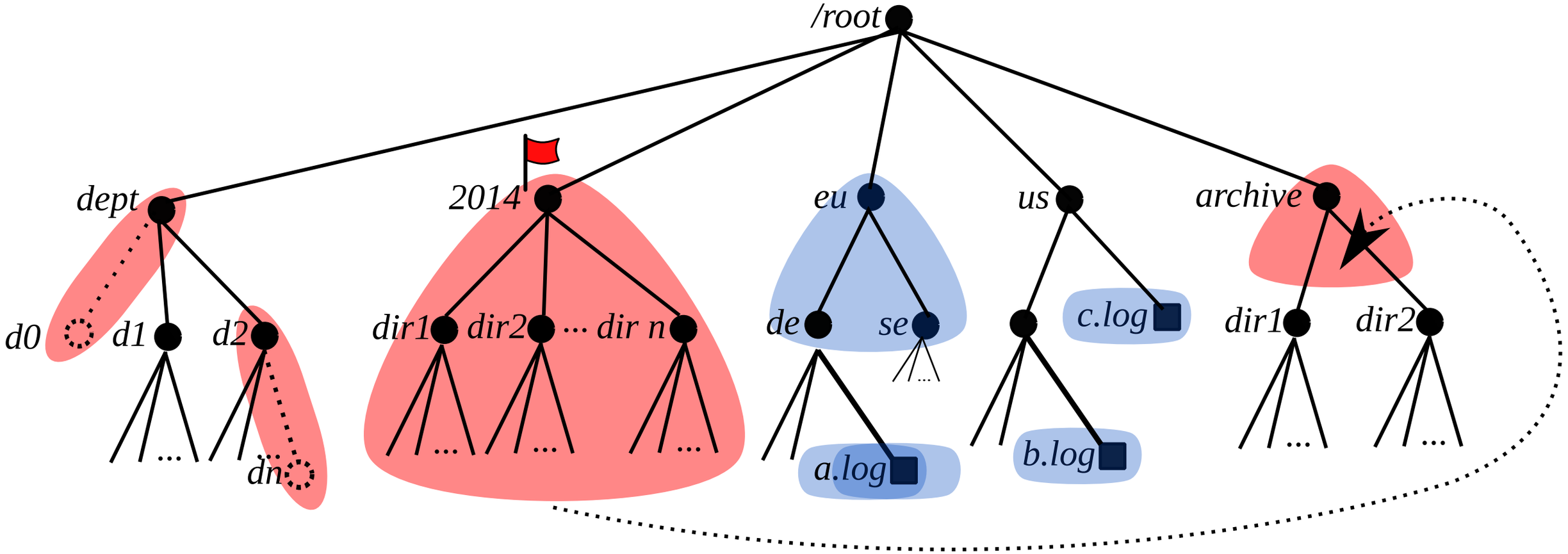
Fine grain Locking

- Hierarchical Locking (Implicit Locking)

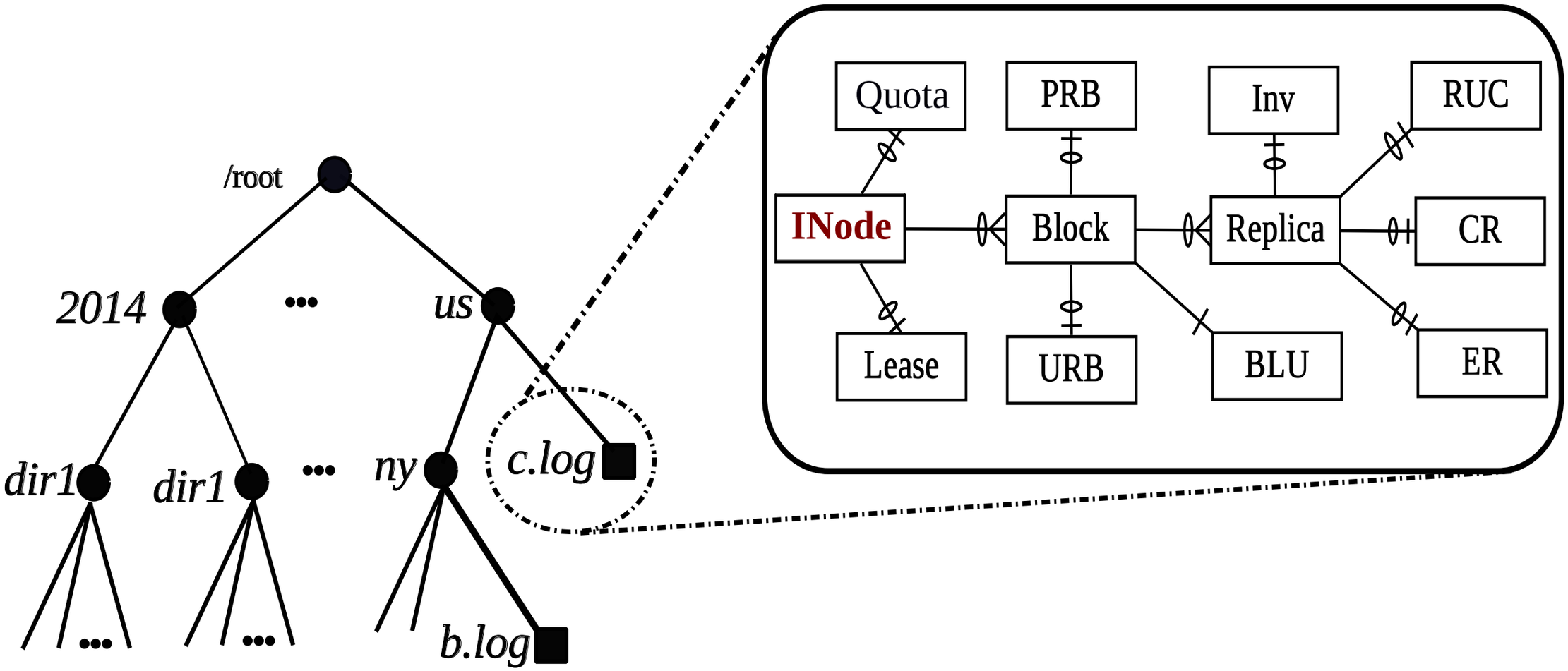


Fine grain Locking

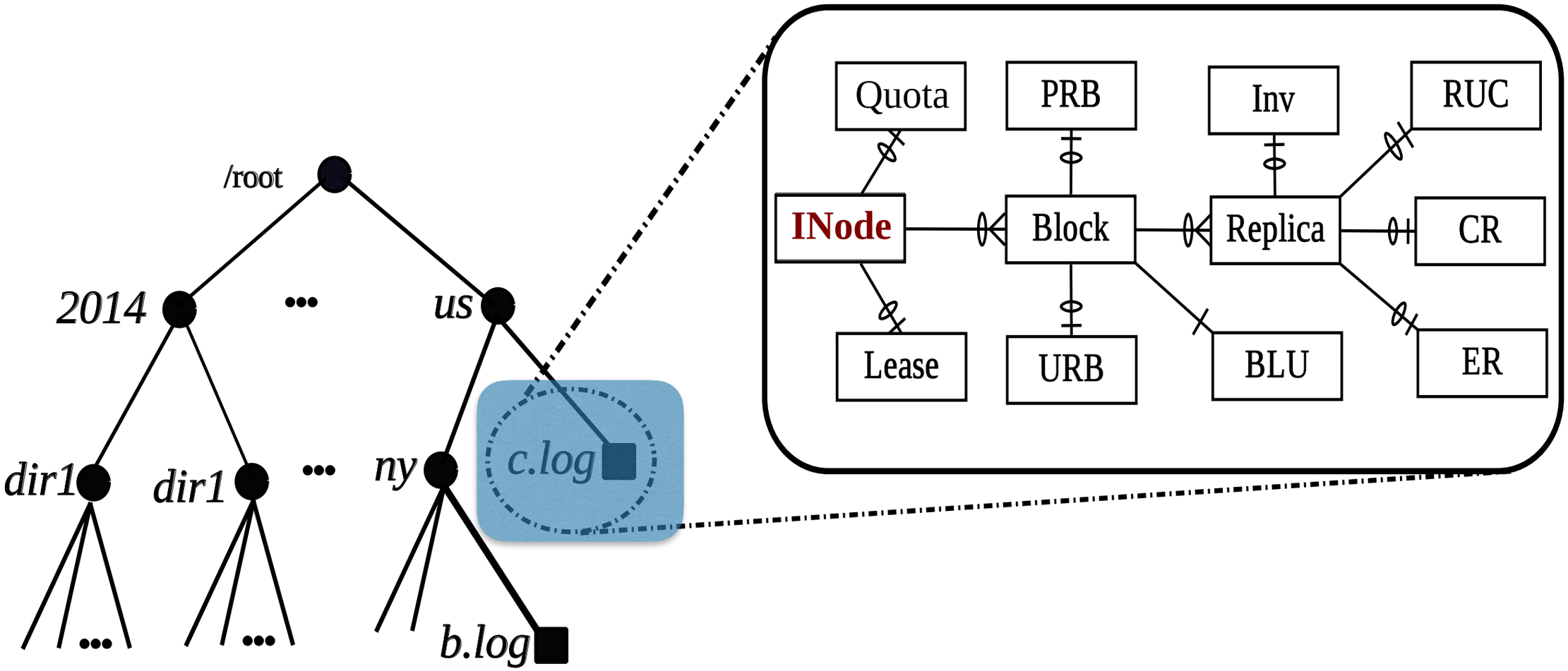
- Hierarchical Locking (Implicit Locking)
- Subtree Locking



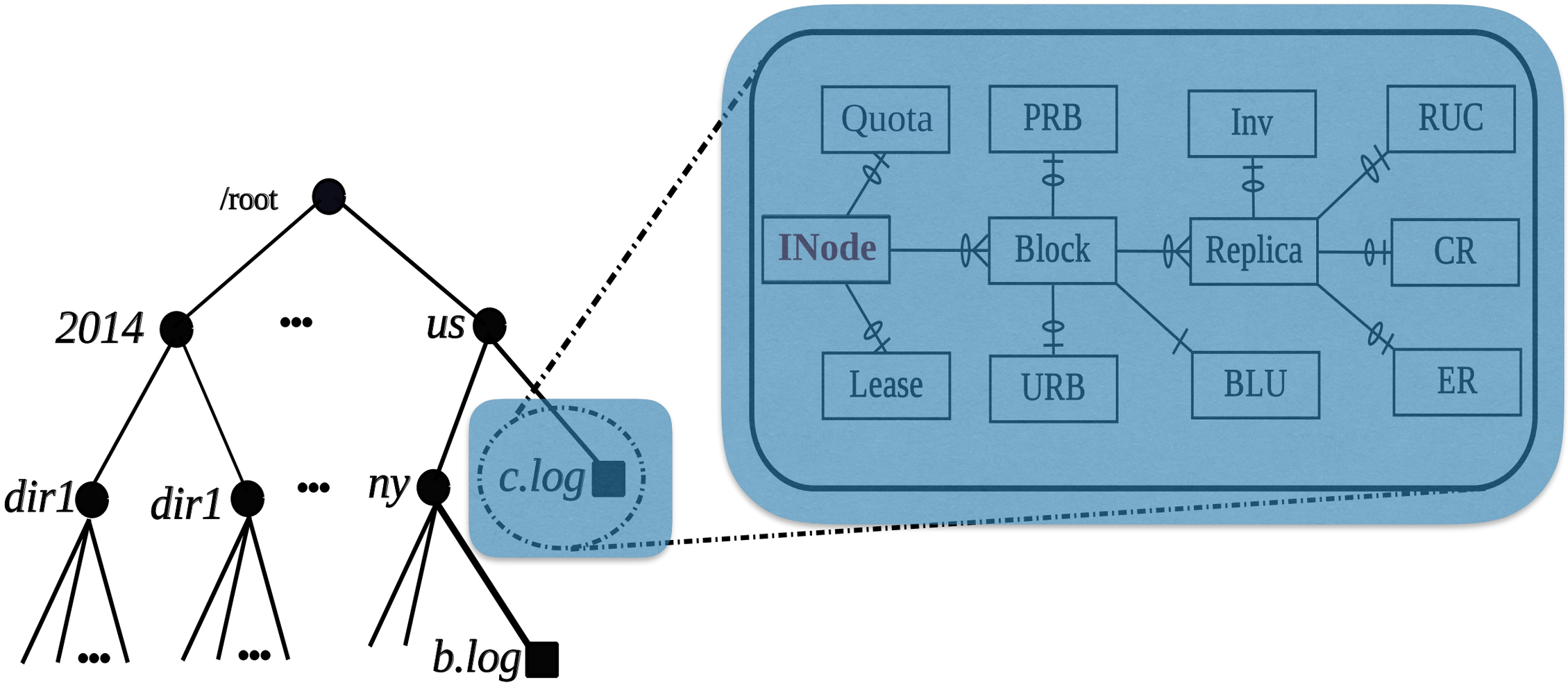
Implicit Locking



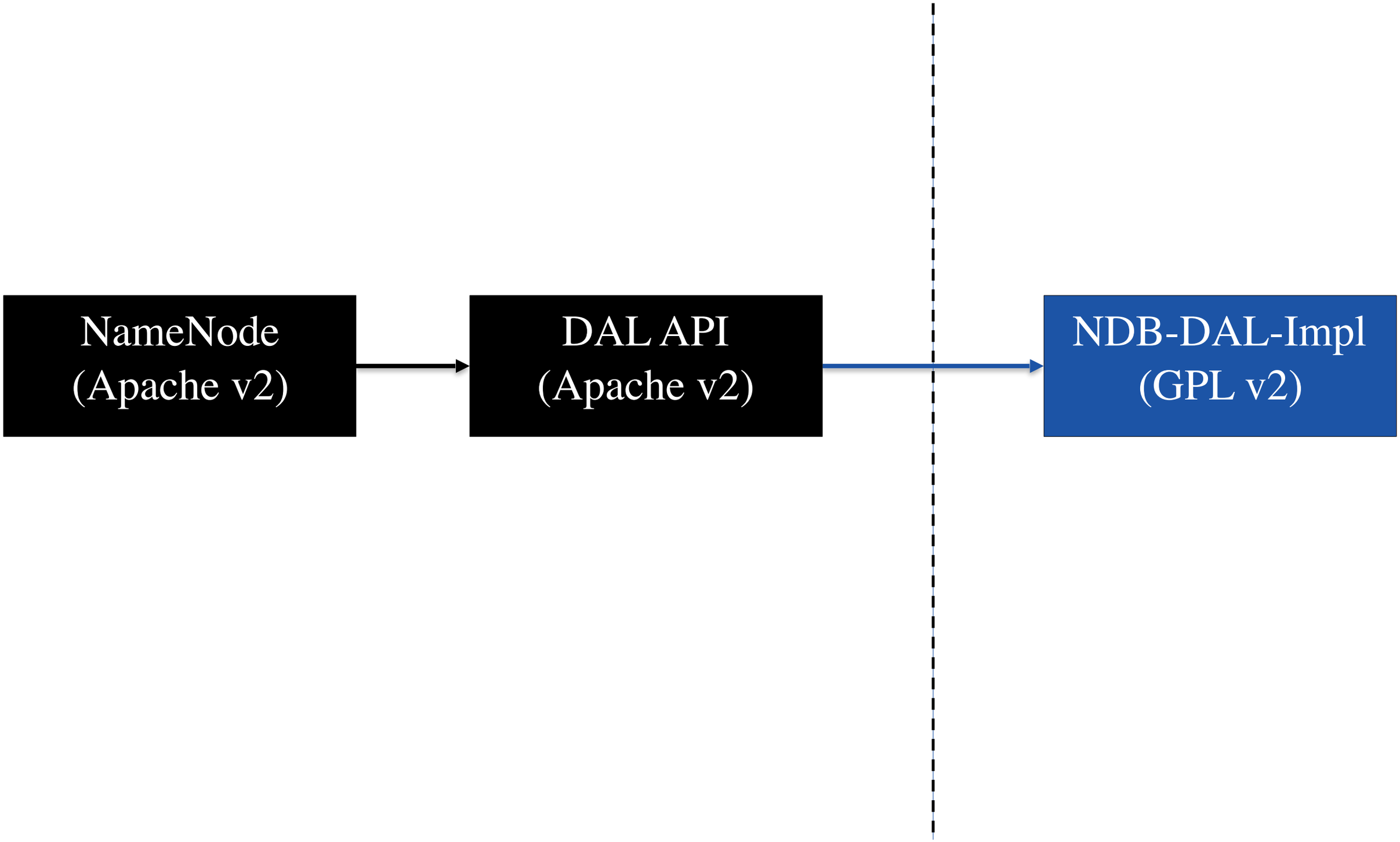
Implicit Locking



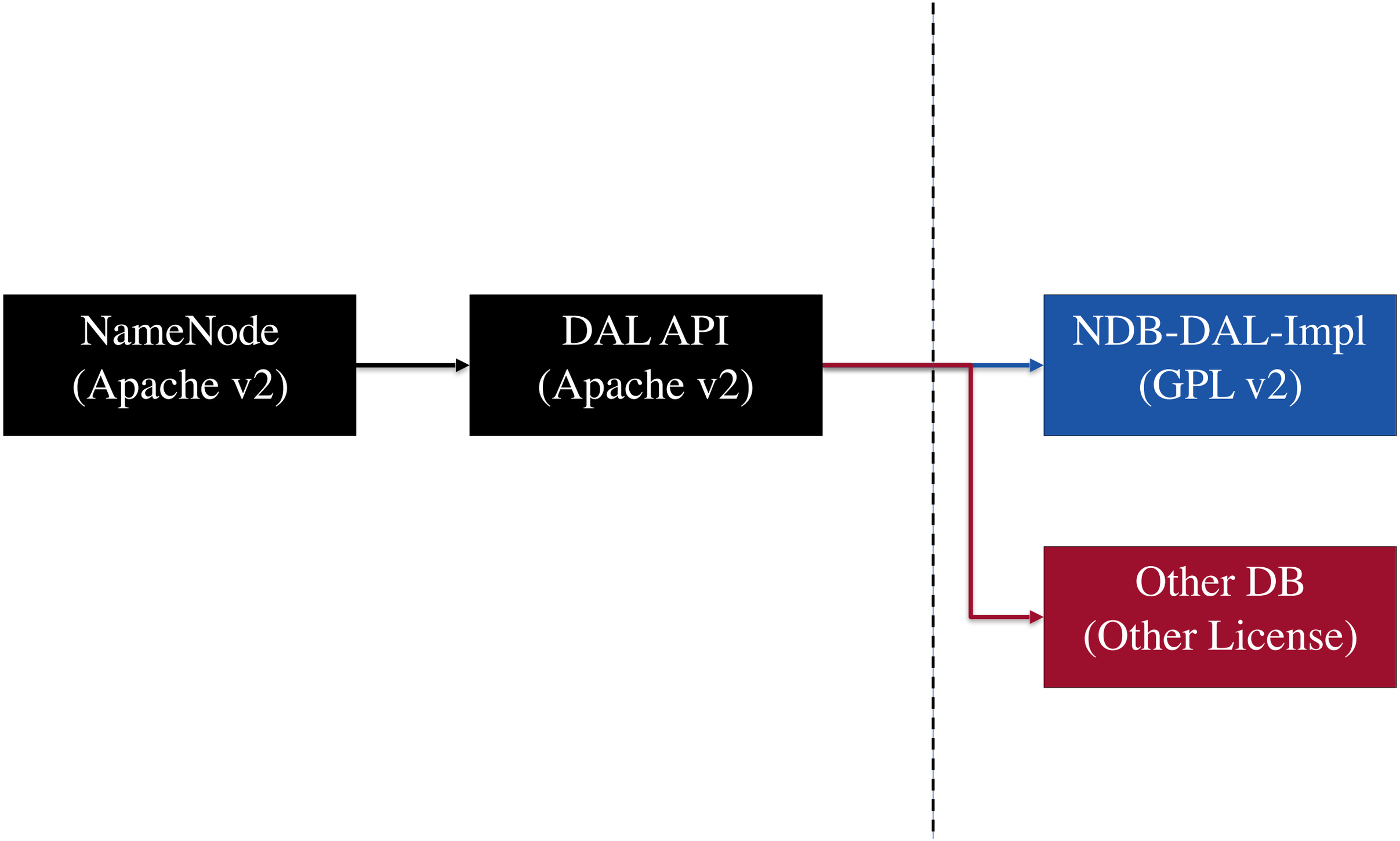
Implicit Locking



Pluggable DBs: Data Abstraction Layer



Pluggable DBs: Data Abstraction Layer



Spotify Workload

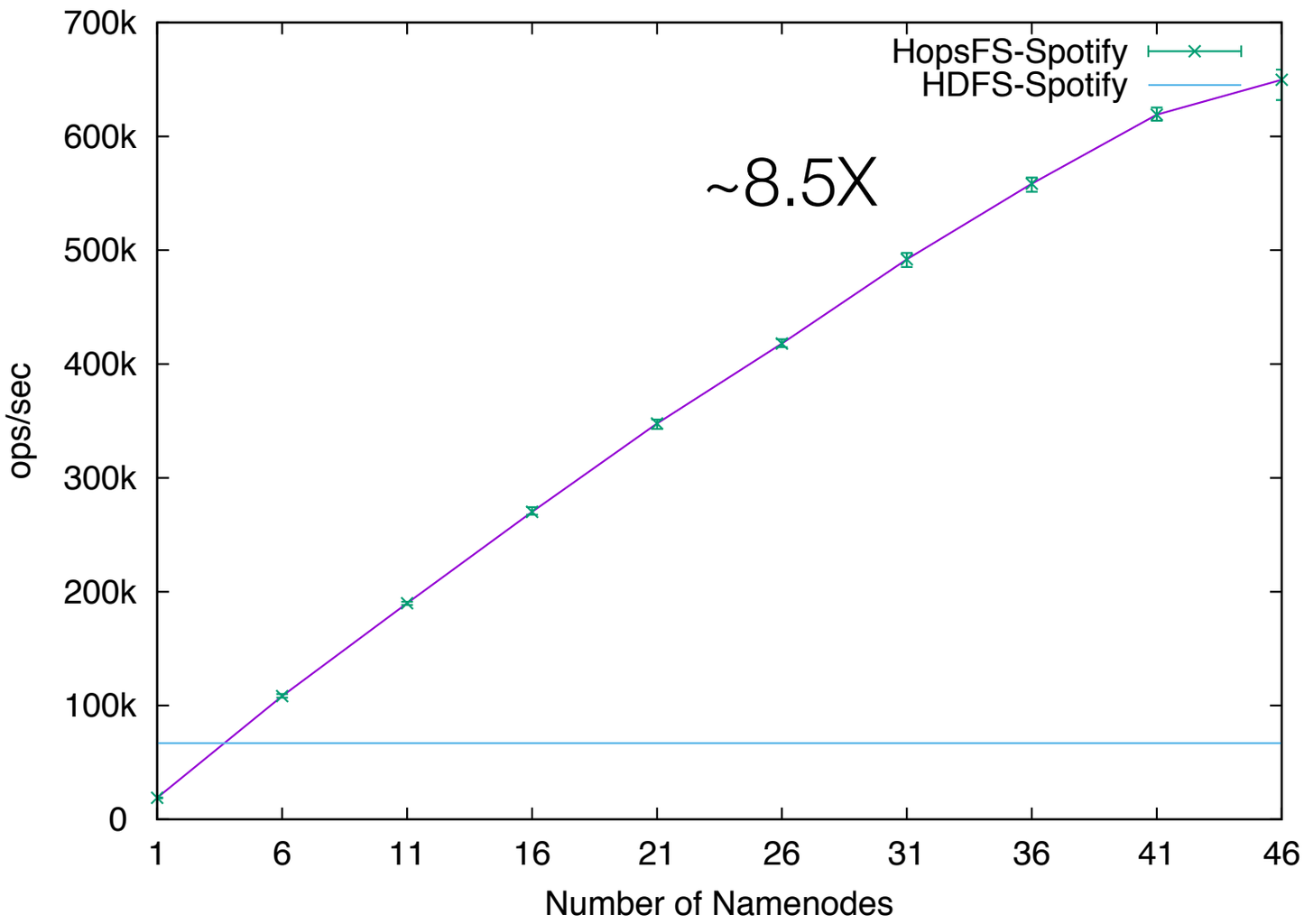
Op Name	Percentage	Op Name	Percentage
append file	0.0%	content summary	0.01%
mkdirs	0.02%	set permissions	0.03% [26.3%*]
set replication	0.14%	set owner	0.32 % [100%*]
delete	0.75% [3.5%*]	create file	1.2%
rename	1.3% [0.03%*]	add blocks	1.5%
list (*listStatus*)	**9% [94.5%*]**	**stat (*fileInfo*)**	**17% [23.3%*]**
read (*getBlkLoc*)	**68.73%**		

Relative frequency of operations on a Spotify HDFS cluster. List, read, and stat operations account for ≈ 95% of the metadata operations in the cluster.

*Of which, the relative percentage is on directories

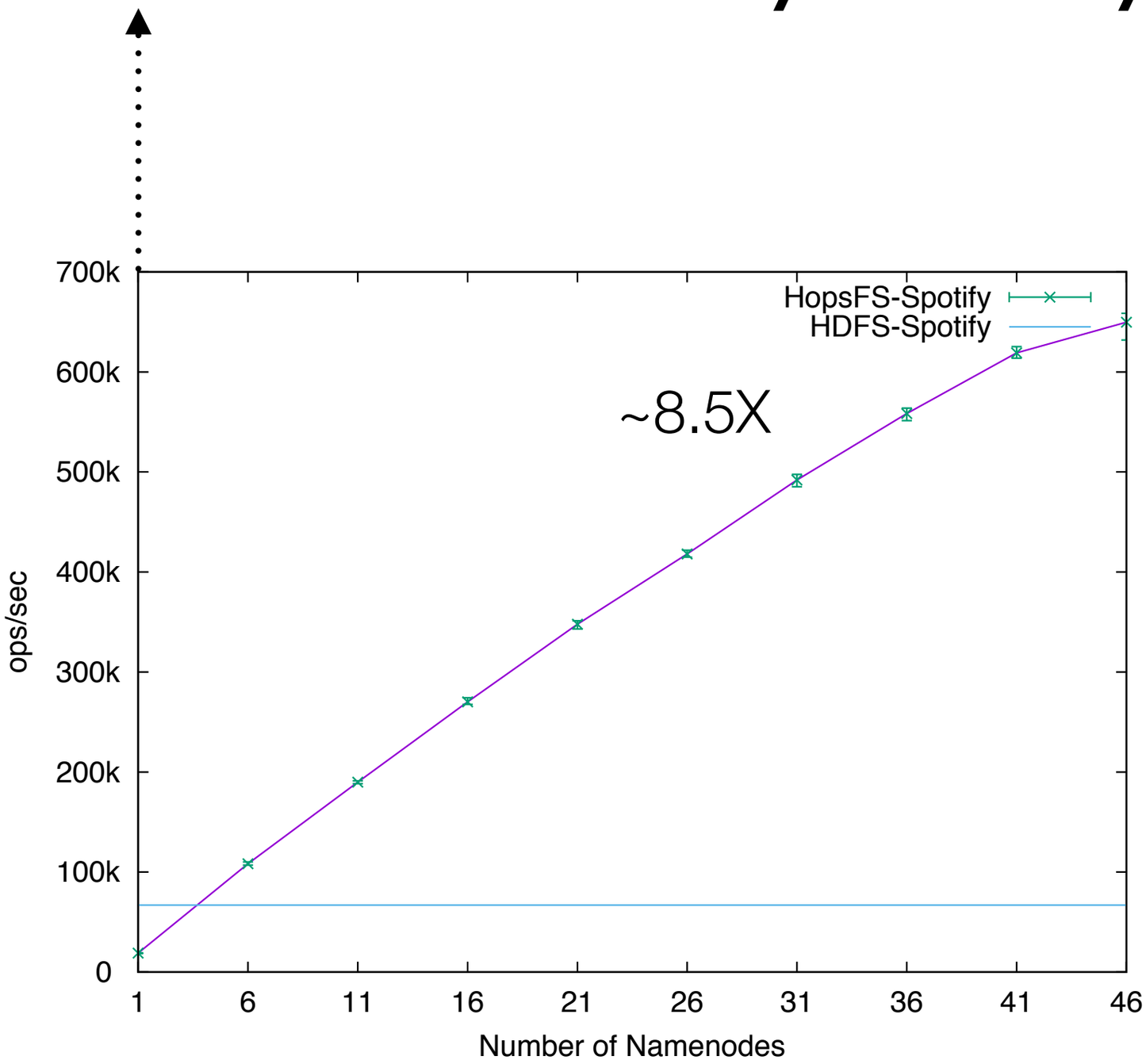
[Niazi, Salman, et al. "HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases." arXiv preprint (2016)]

To Infinity and Beyond



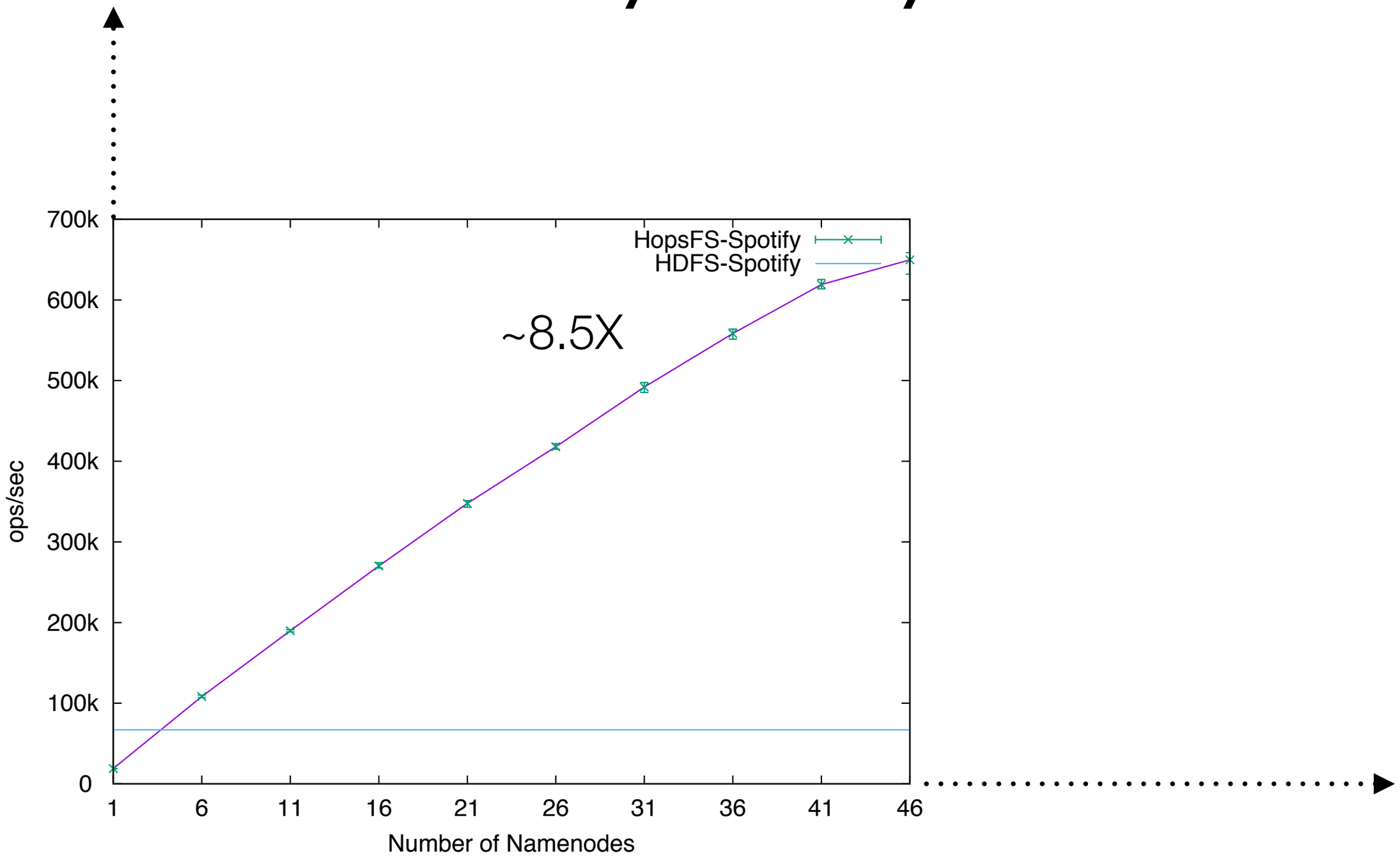
NDB Setup: 8 Nodes using Xeon E5-2620 2.40GHz Processors and 10GbE.
NameNodes: Xeon E5-2620 2.40GHz Processors machines and 10GbE.

To Infinity and Beyond



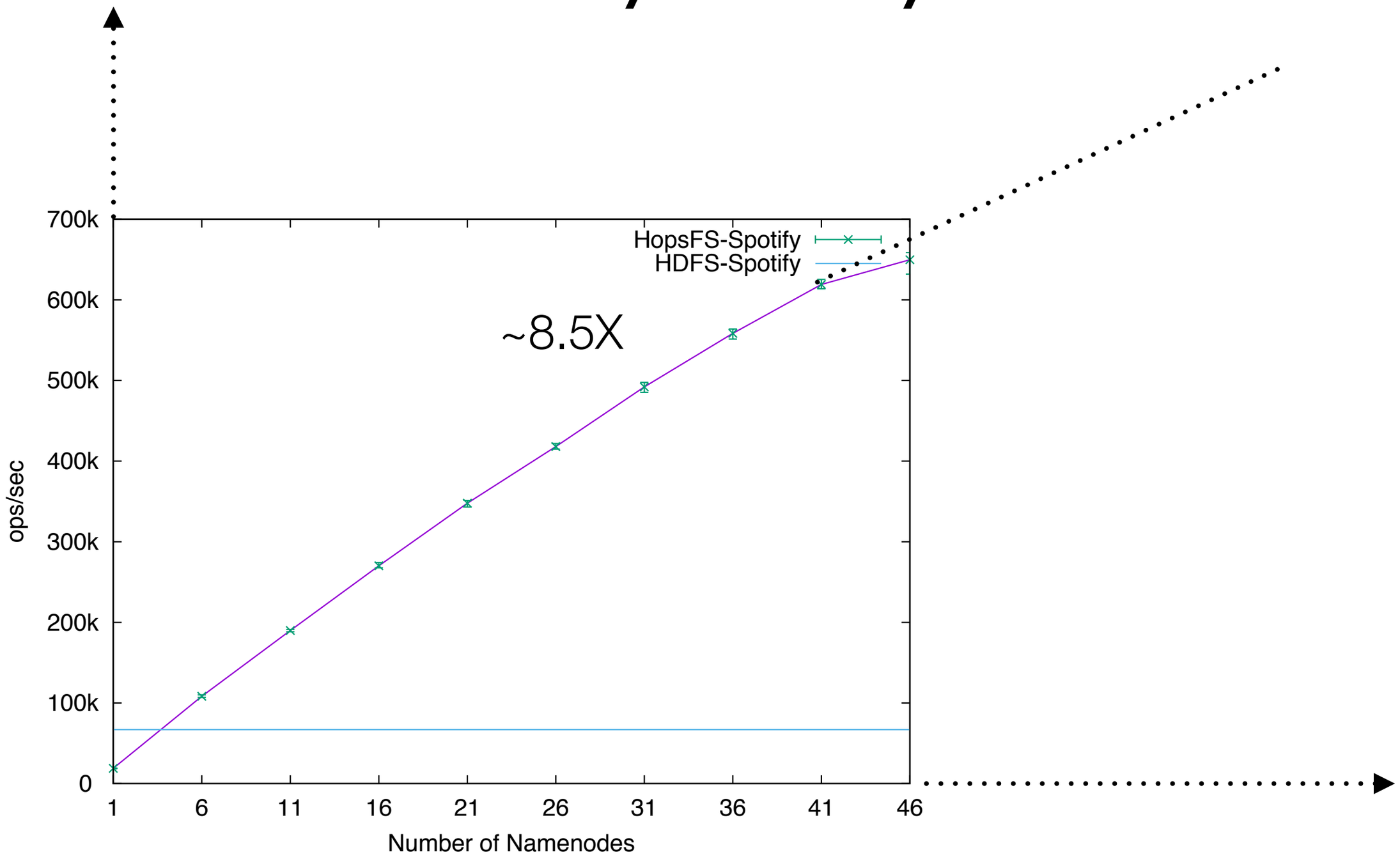
NDB Setup: 8 Nodes using Xeon E5-2620 2.40GHz Processors and 10GbE.
NameNodes: Xeon E5-2620 2.40GHz Processors machines and 10GbE.

To Infinity and Beyond



NDB Setup: 8 Nodes using Xeon E5-2620 2.40GHz Processors and 10GbE.
NameNodes: Xeon E5-2620 2.40GHz Processors machines and 10GbE.

To Infinity and Beyond



NDB Setup: 8 Nodes using Xeon E5-2620 2.40GHz Processors and 10GbE.
NameNodes: Xeon E5-2620 2.40GHz Processors machines and 10GbE.

Bigger Clusters

Memory	Number of Files	
	HDFS	**HopsFS**
1 GB	2.3 million	0.44 million
50 GB	115 million	22 million
100 GB	230 million	44 million
200 GB	**460 million**	88 million
500 GB	Does Not Scale	220 million
1 TB	Does Not Scale	440 million
24 TB	Does Not Scale	**10.8 billion**

Tinker Friendly Metadata

Tinker Friendly Metadata

- The Database (NDB) is the single source of truth

Tinker Friendly Metadata

- The Database (NDB) is the single source of truth
- Extending INodes (files/directories)

Tinker Friendly Metadata

- The Database (NDB) is the single source of truth
- Extending INodes (files/directories)
  - Adding a new table with a foreign key to the nodes table

Tinker Friendly Metadata

- The Database (NDB) is the single source of truth
- Extending INodes (files/directories)
  - Adding a new table with a foreign key to the nodes table
  - Attaching metadata to a file/directory

Tinker Friendly Metadata

- The Database (NDB) is the single source of truth
- Extending INodes (files/directories)
  - Adding a new table with a foreign key to the nodes table
- Attaching metadata to a file/directory
  - Schema-less

Tinker Friendly Metadata

- The Database (NDB) is the single source of truth
- Extending INodes (files/directories)
  - Adding a new table with a foreign key to the nodes table
- Attaching metadata to a file/directory
  - Schema-less
  - Schema-based

Schema-less Metadata

Schema-less Metadata

inodeID	Name	parentId	
1	/	0	
2	Users	1	
3	alice.txt	2	

Schema-less Metadata

inodeID	Name	parentId	
1	/	0	
2	Users	1	
3	alice.txt	2	

attach /Users/alice.txt ’{“age” : 20, “gender” : “female”, “about”: “I am alice”}’

Schema-less Metadata

inodeID	Name	parentId	
1	/	0	
2	Users	1	
3	alice.txt	2	

inodeID	metadata
3	{“age” : 20, “gender” : “female”, “about”: “I am alice”}

attach /Users/alice.txt ’{“age” : 20, “gender” : “female”, “about”: “I am alice”}’

Schema-less Metadata

inodeID	Name	parentId	
1	/	0	
2	Users	1	
3	alice.txt	2	

inodeID	metadata
3	{“age” : 20, “gender” : “female”, “about”: “I am alice”}

attach /Users/alice.txt ’{“age” : 20, “gender” : “female”, “about”: “I am alice”}’

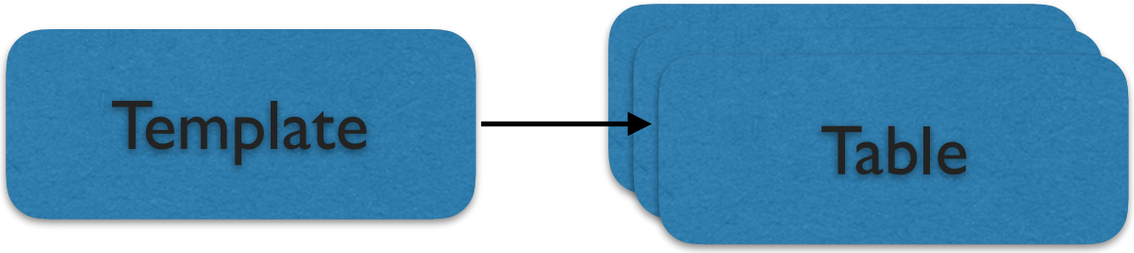
Schema-based Metadata

Schema-based Metadata

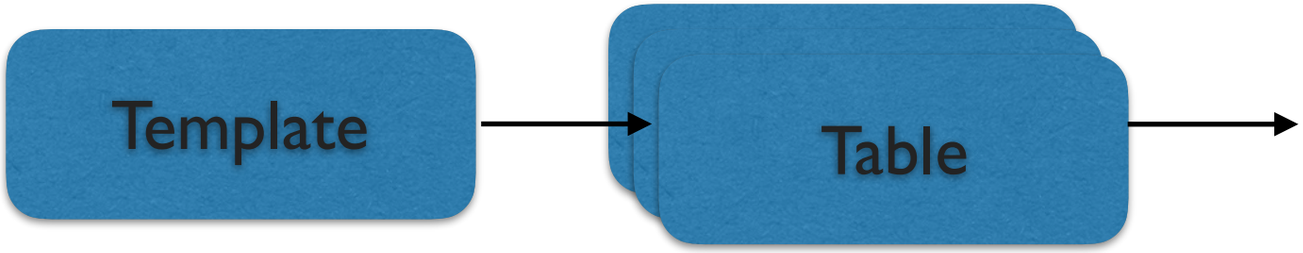
Template

Schema-based Metadata

Schema-based Metadata



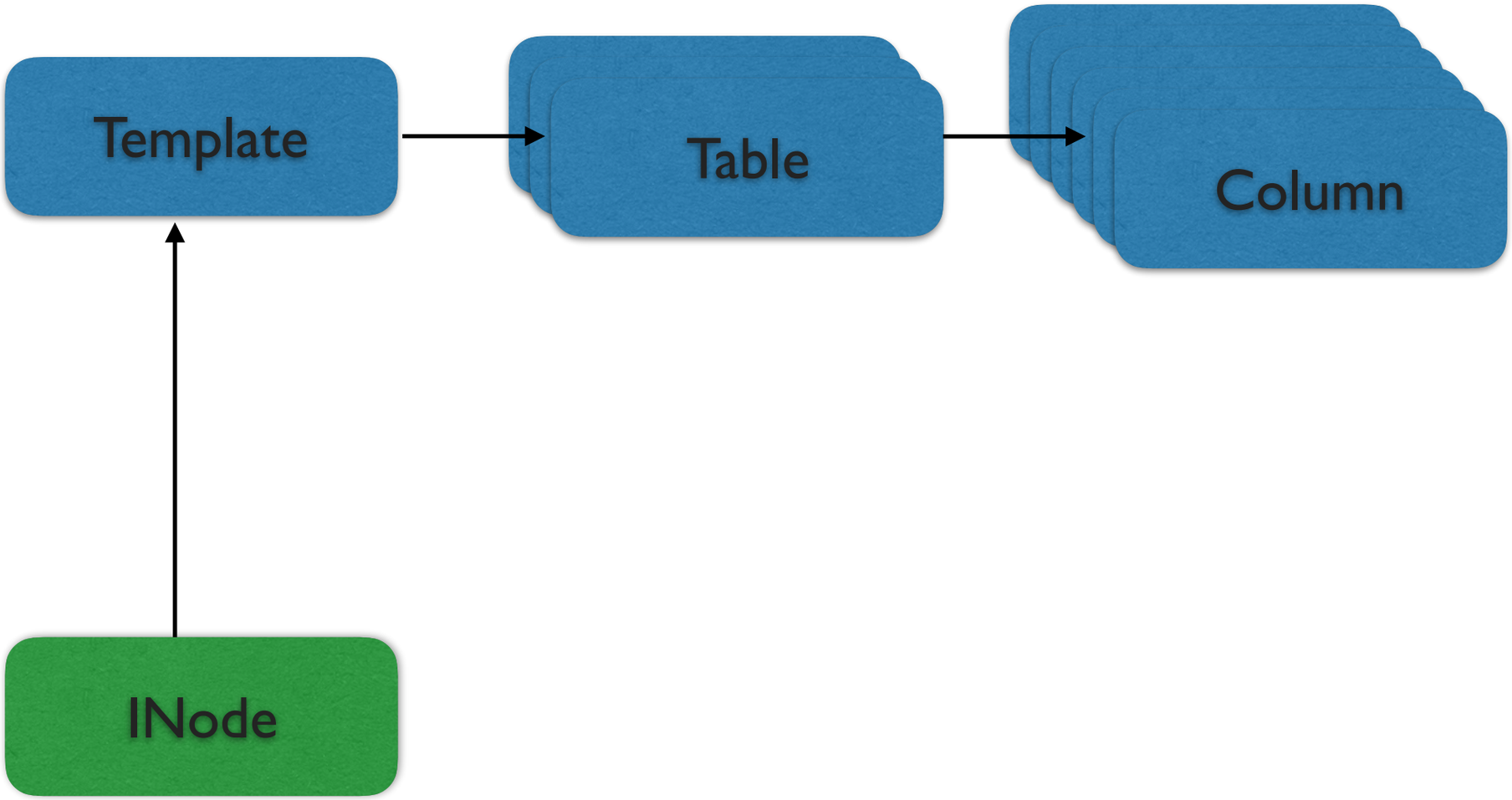
Schema-based Metadata



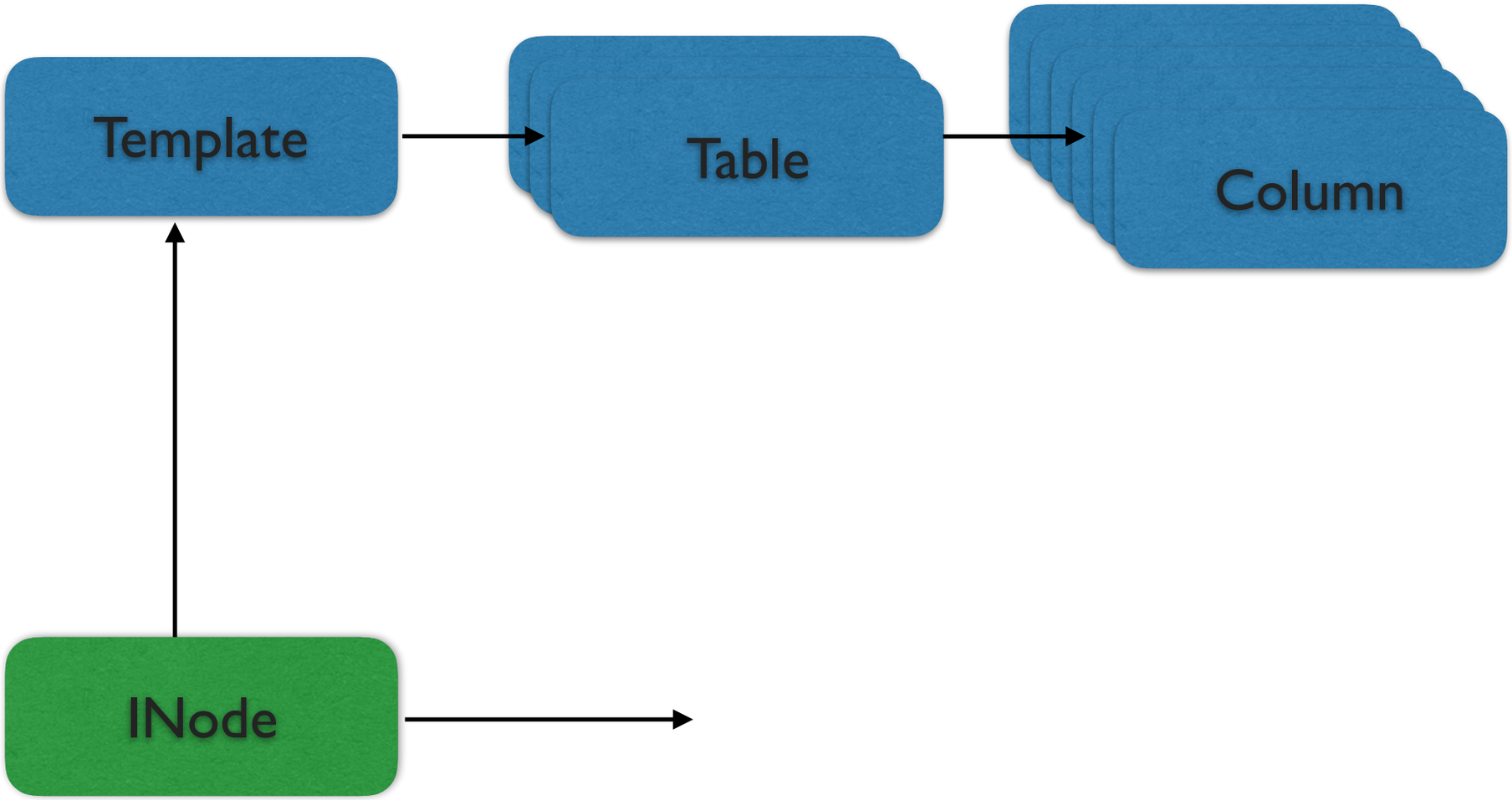
Schema-based Metadata

Schema-based Metadata

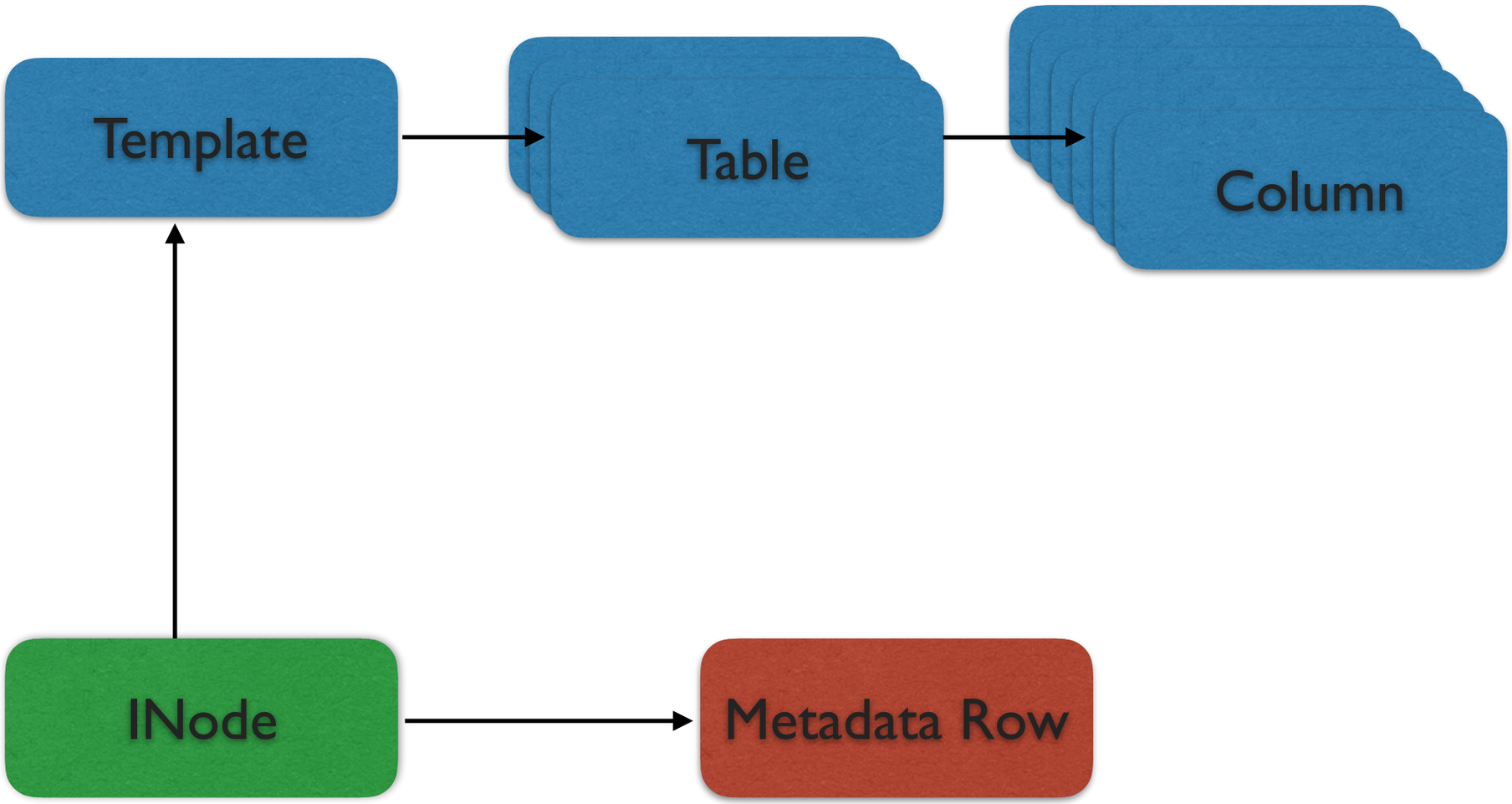
Schema-based Metadata



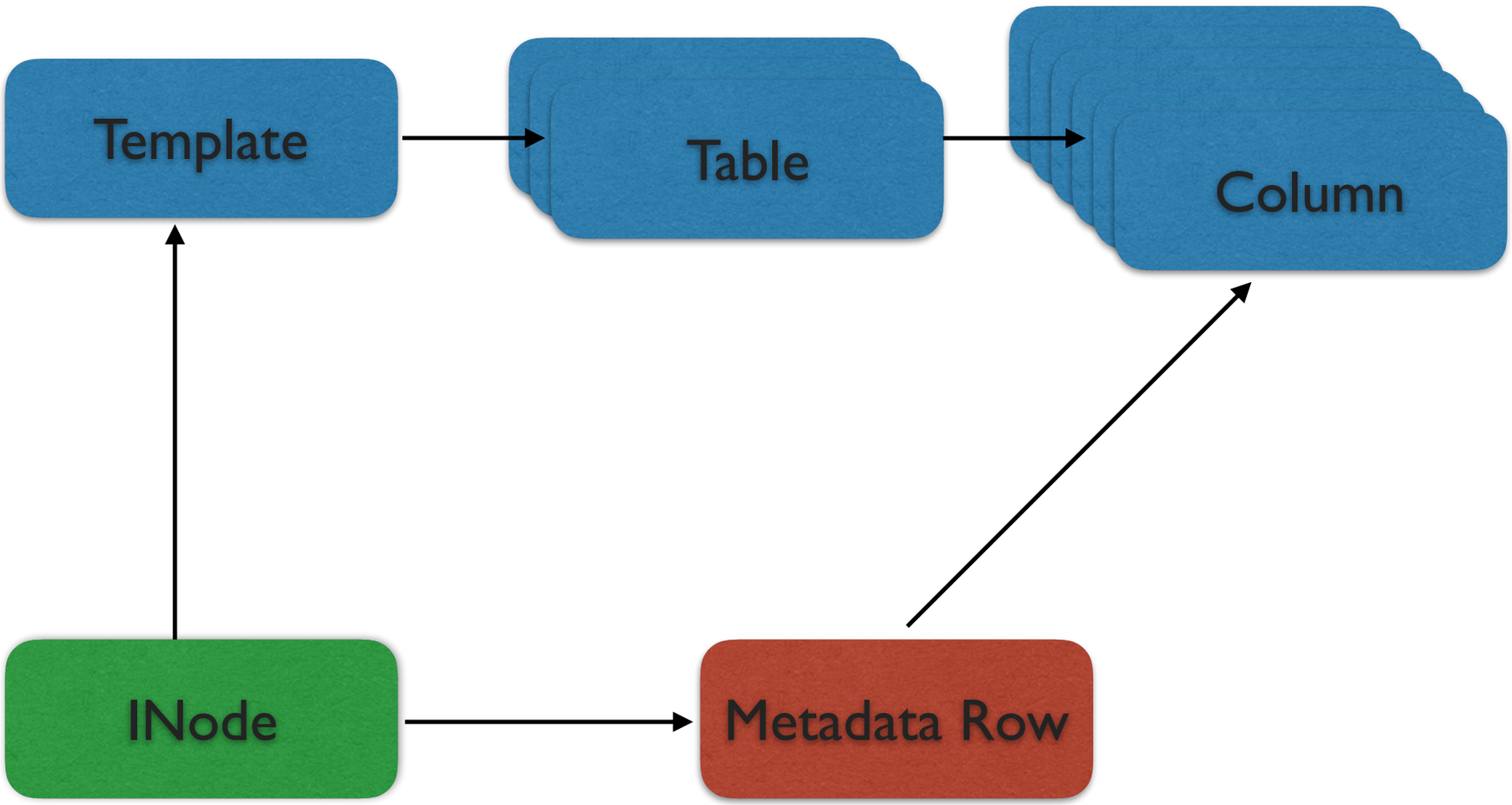
Schema-based Metadata



Schema-based Metadata



Schema-based Metadata



Searching through the Namespace

Searching through the Namespace

- hdfs -find

Searching through the Namespace

- hdfs -find
  - limited

Searching through the Namespace

- hdfs -find
  - limited
  - inefficient by design

Searching through the Namespace

- hdfs -find
  - limited
  - inefficient by design
- Pig/MapReduce

Searching through the Namespace

- hdfs -find
  - limited
  - inefficient by design
  - Pig/MapReduce
  - NameNode is a critical point, avoid overloading

Searching through the Namespace

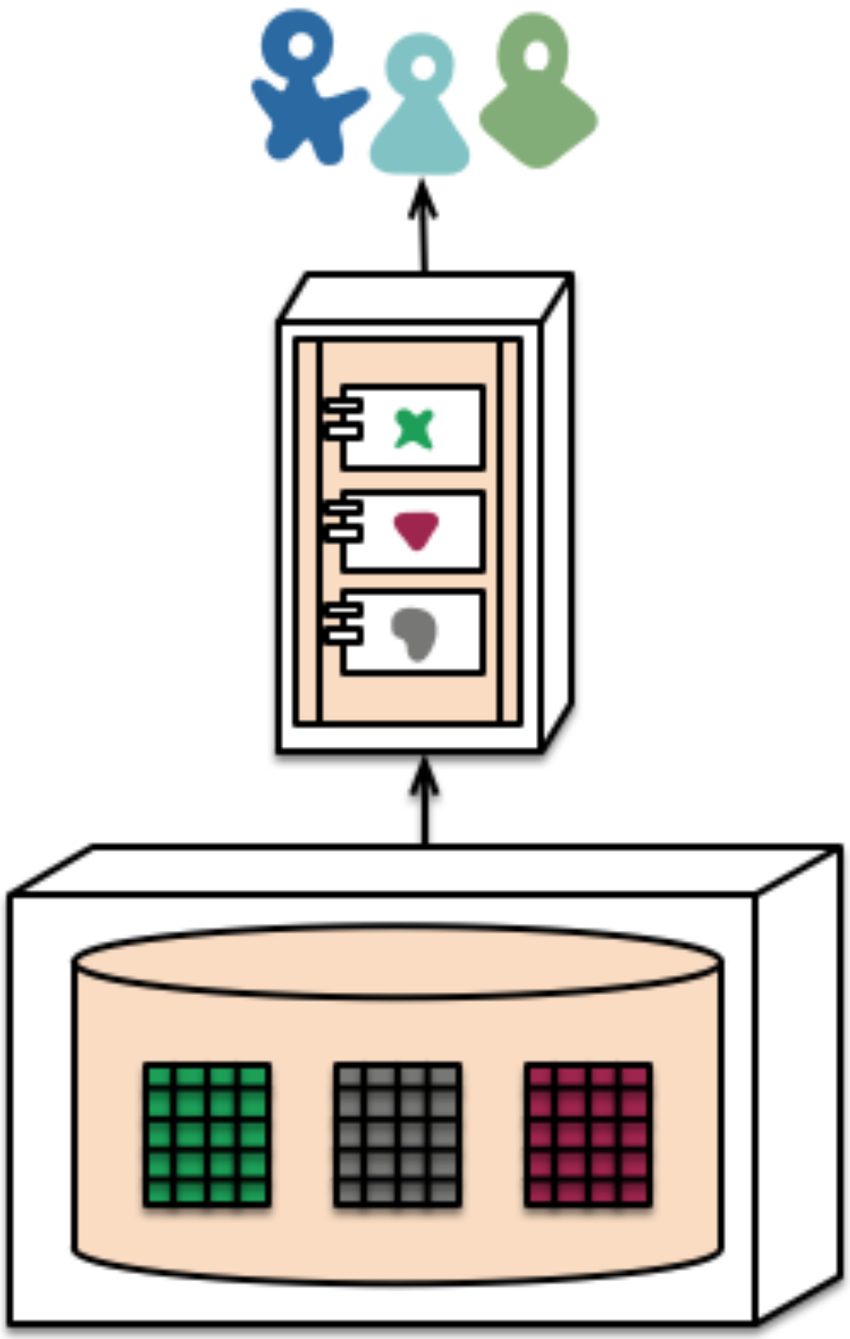
- hdfs -find
  - limited
  - inefficient by design
- Pig/MapReduce
- NameNode is a critical point, avoid overloading
- SQL Query on NDB

Searching through the Namespace

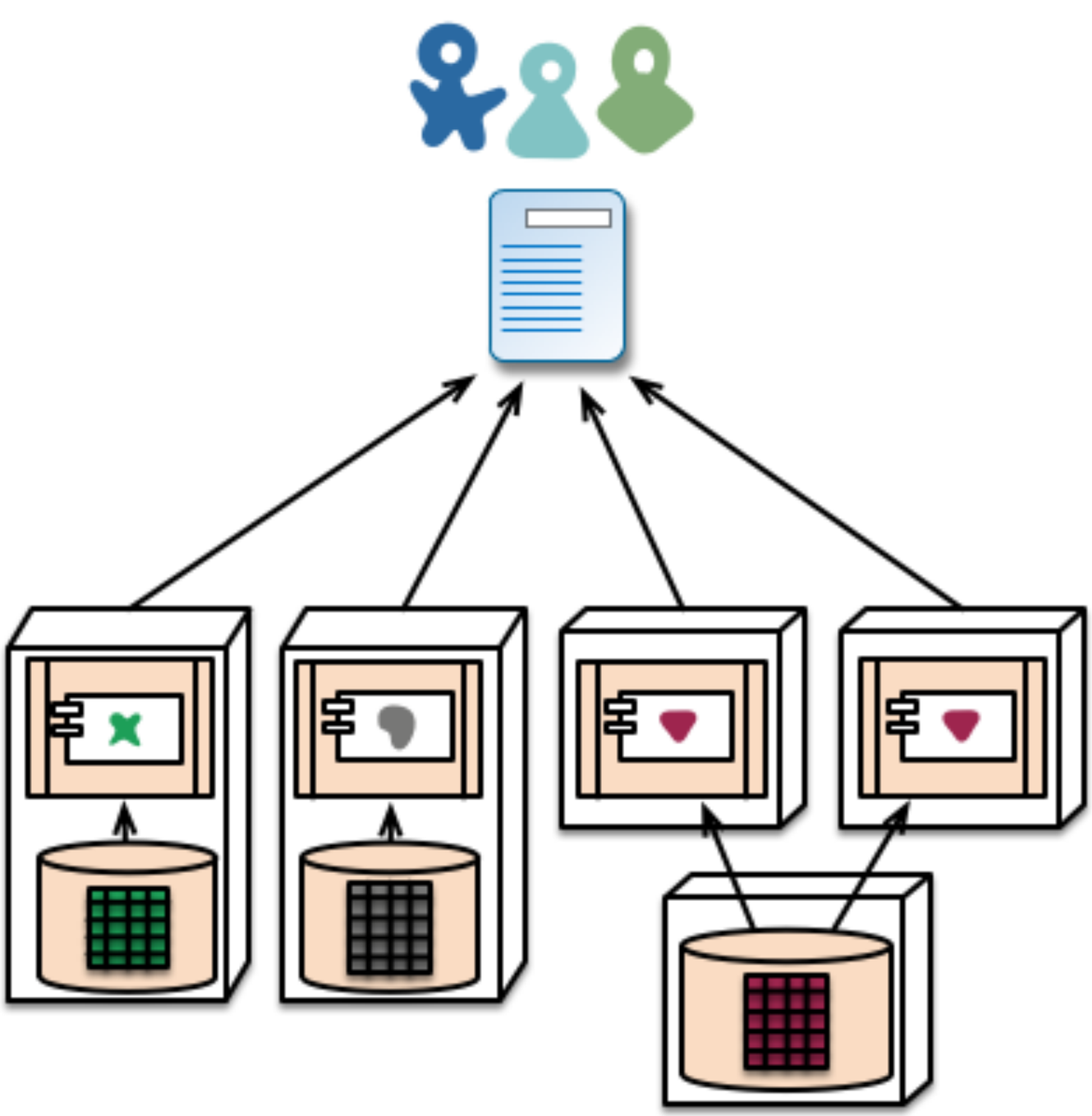
- hdfs -find
  - limited
  - inefficient by design
- Pig/MapReduce
- NameNode is a critical point, avoid overloading
- SQL Query on NDB
- One Size does not fit all

Polyglot Persistence: the Right Tool for the Job

Monolithic vs Polyglot Persistence



monolith - single database



microservices - application databases

[http://martinfowler.com/]

Asynchronous Update of the Metadata

Eventual Consistency for Metadata.

Metadata Integrity maintained by
Asynchronous Replication and Metadata Immutability.

Asynchronous Update of the Metadata

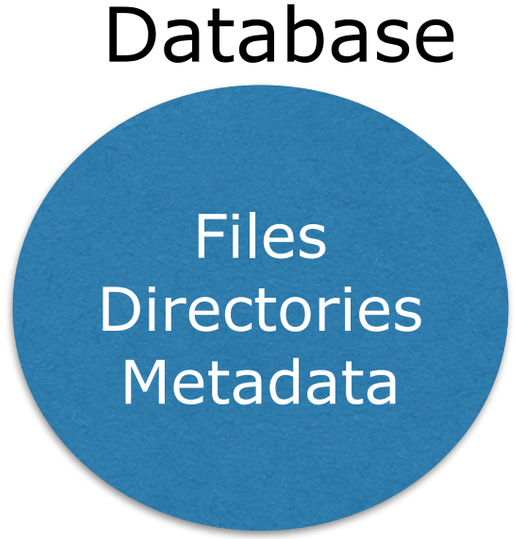
Eventual Consistency for Metadata.
Metadata Integrity maintained by
Asynchronous Replication and Metadata Immutability.

Database

Asynchronous Update of the Metadata

Eventual Consistency for Metadata.

Metadata Integrity maintained by
Asynchronous Replication and Metadata Immutability.



Asynchronous Update of the Metadata

Eventual Consistency for Metadata.
Metadata Integrity maintained by
Asynchronous Replication and Metadata Immutability.

Elasticsearch

Database

Asynchronous Update of the Metadata

Eventual Consistency for Metadata.
Metadata Integrity maintained by
Asynchronous Replication and Metadata Immutability.

Elasticsearch

Search Indexes

Database

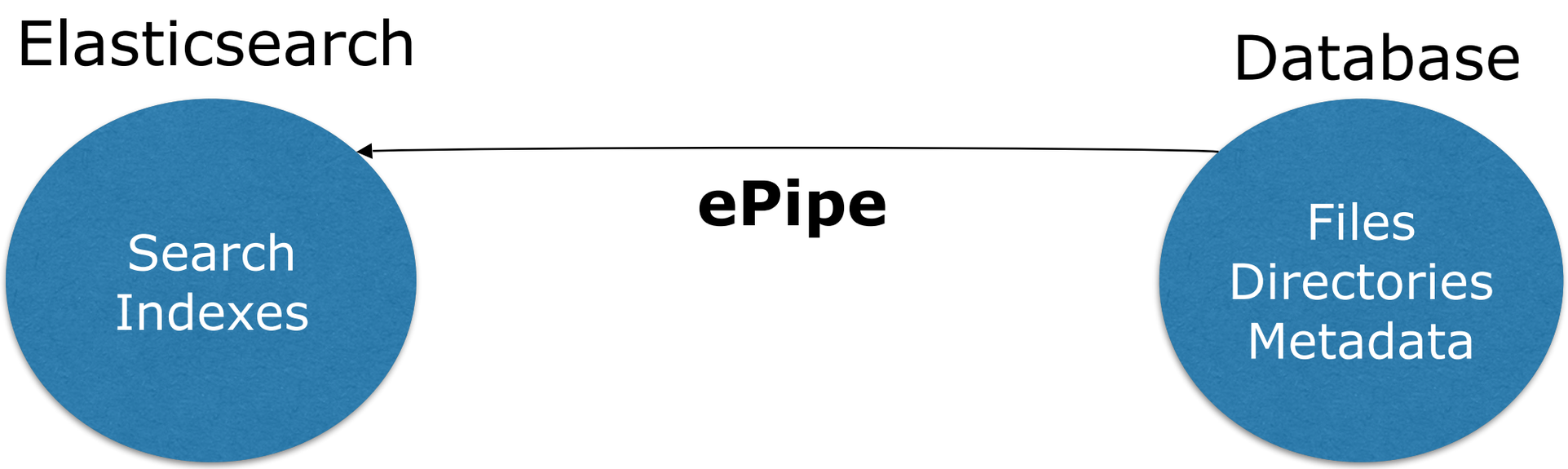
Files
Directories
Metadata

Asynchronous Update of the Metadata

Eventual Consistency for Metadata.
Metadata Integrity maintained by
Asynchronous Replication and Metadata Immutability.

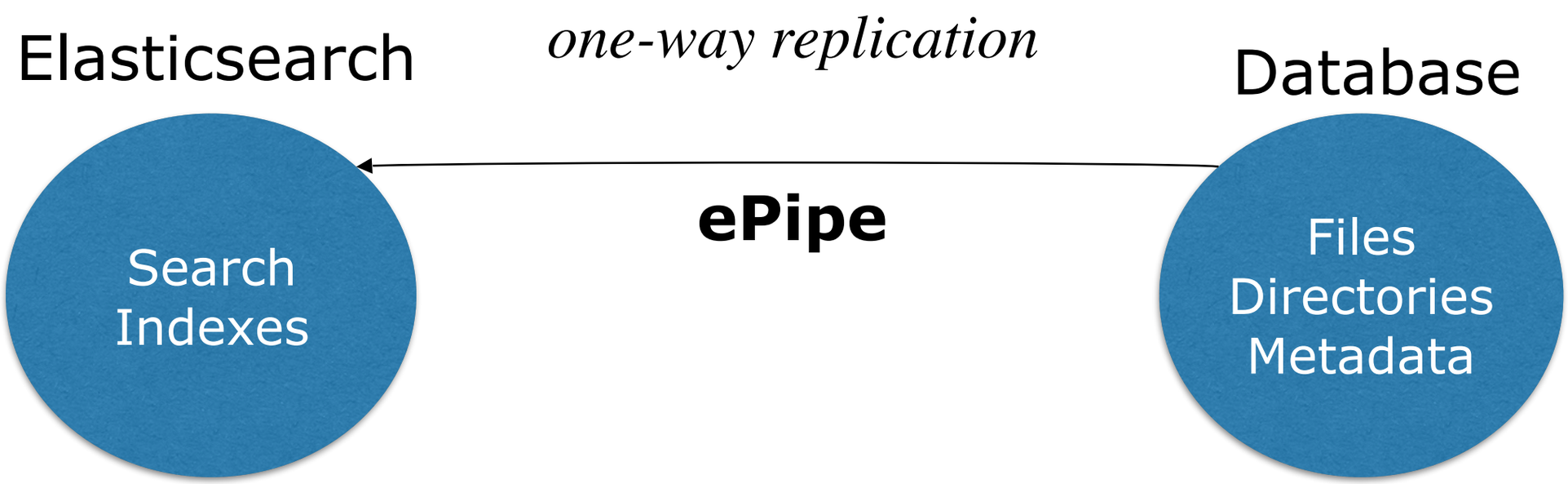
Asynchronous Update of the Metadata

Eventual Consistency for Metadata.
Metadata Integrity maintained by
Asynchronous Replication and Metadata Immutability.



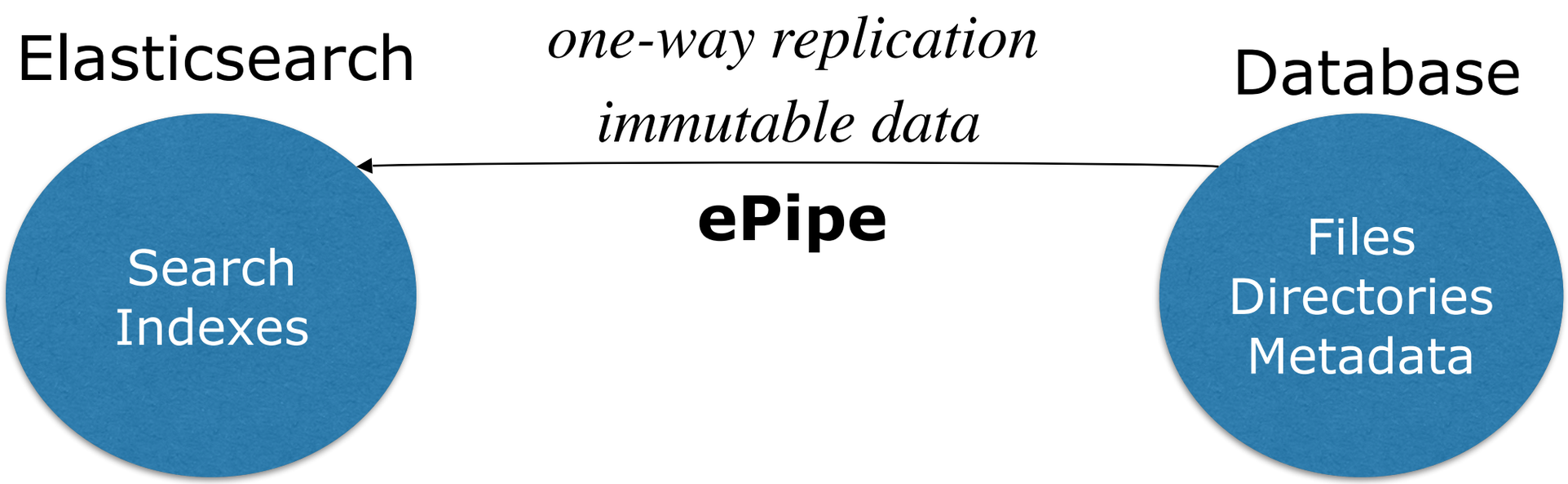
Asynchronous Update of the Metadata

Eventual Consistency for Metadata.
Metadata Integrity maintained by
Asynchronous Replication and Metadata Immutability.

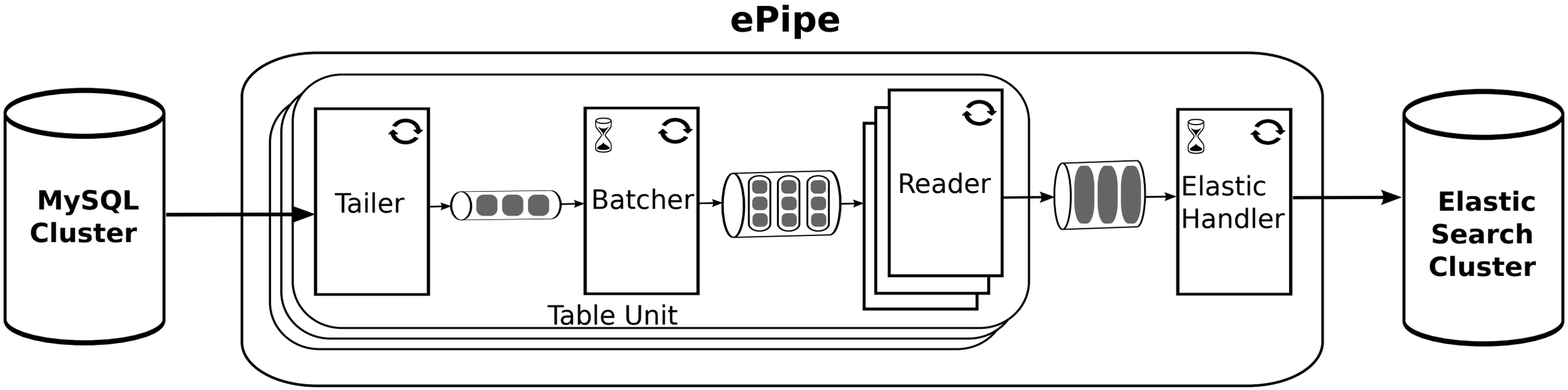


Asynchronous Update of the Metadata

Eventual Consistency for Metadata.
Metadata Integrity maintained by
Asynchronous Replication and Metadata Immutability.



HopsFS | ElasticSearch



Supporting Project-Level Multi-Tenancy

next

How can we introduce
GitHub-style projects to Hadoop?

Tutorial

Tutorial

- Karamel Automated Installation

Tutorial

- Karamel Automated Installation
- http://www.hops.io/?q=boss

Tutorial

- Karamel Automated Installation
- http://www.hops.io/?q=boss
- HopsWorks Clusters

Tutorial

- Karamel Automated Installation
- http://www.hops.io/?q=boss
- HopsWorks Clusters
  - http://52.210.205.125:8080/hopsworks/

Tutorial

- Karamel Automated Installation
- http://www.hops.io/?q=boss
- HopsWorks Clusters
  - http://52.210.205.125:8080/hopsworks/
  - http://52.209.143.68:8080/hopsworks/

Tutorial

- Karamel Automated Installation
- http://www.hops.io/?q=boss
- HopsWorks Clusters
  - http://52.210.205.125:8080/hopsworks/
  - http://52.209.143.68:8080/hopsworks/
  - http://52.18.186.135:8080/hopsworks/

Conclusions

- Hops is a next-generation distribution of Hadoop.
- HopsWorks is a frontend to Hops that supports multi-tenancy, free-text search, interactive analytics with Zeppelin/Flink/Spark, and batch jobs.
- Looking for contributors/committers
  - Check out our github (github.com/hopshadoop)

Questions

Karamel

Automated Installation

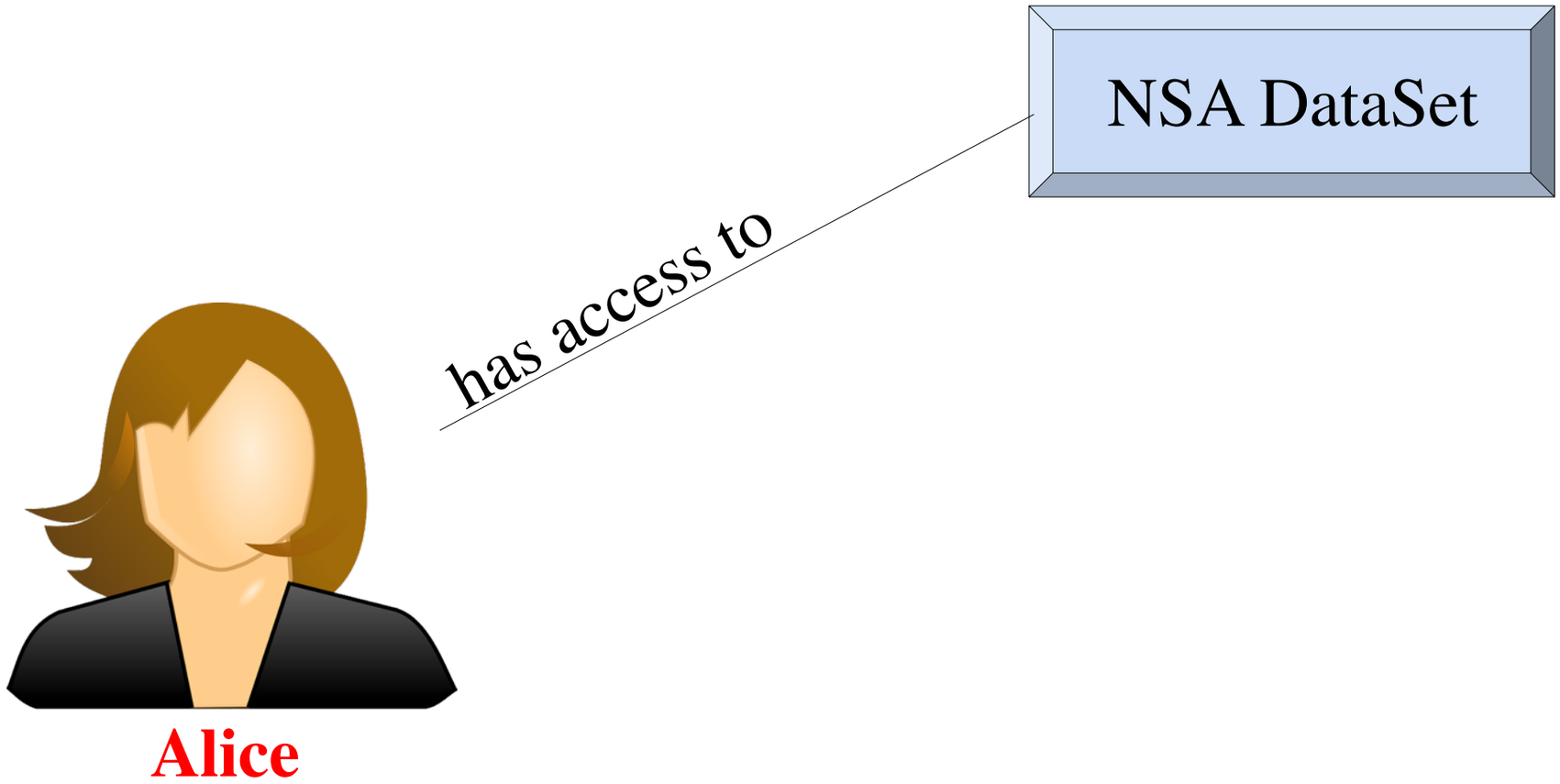
- Vagrant/Chef to spin up on a single host
- Karamel/Chef to deploy on AWS/GCE/OpenStack or on-premises

```
name: HopsWorks
ec2:
  type: m3.medium
cookbooks:
  hadoop: github: "hopshadoop/hopsworks-chef" version: "v0.1"
groups:
  ui:
    size: 1
    recipes:
      - hopsworks
metadata:
    size: 2
    recipes:
      - hops::nn
      - hops::rm
datanodes:
    size: 50
    recipes:
      - hops::dn
      - hops::nm
```

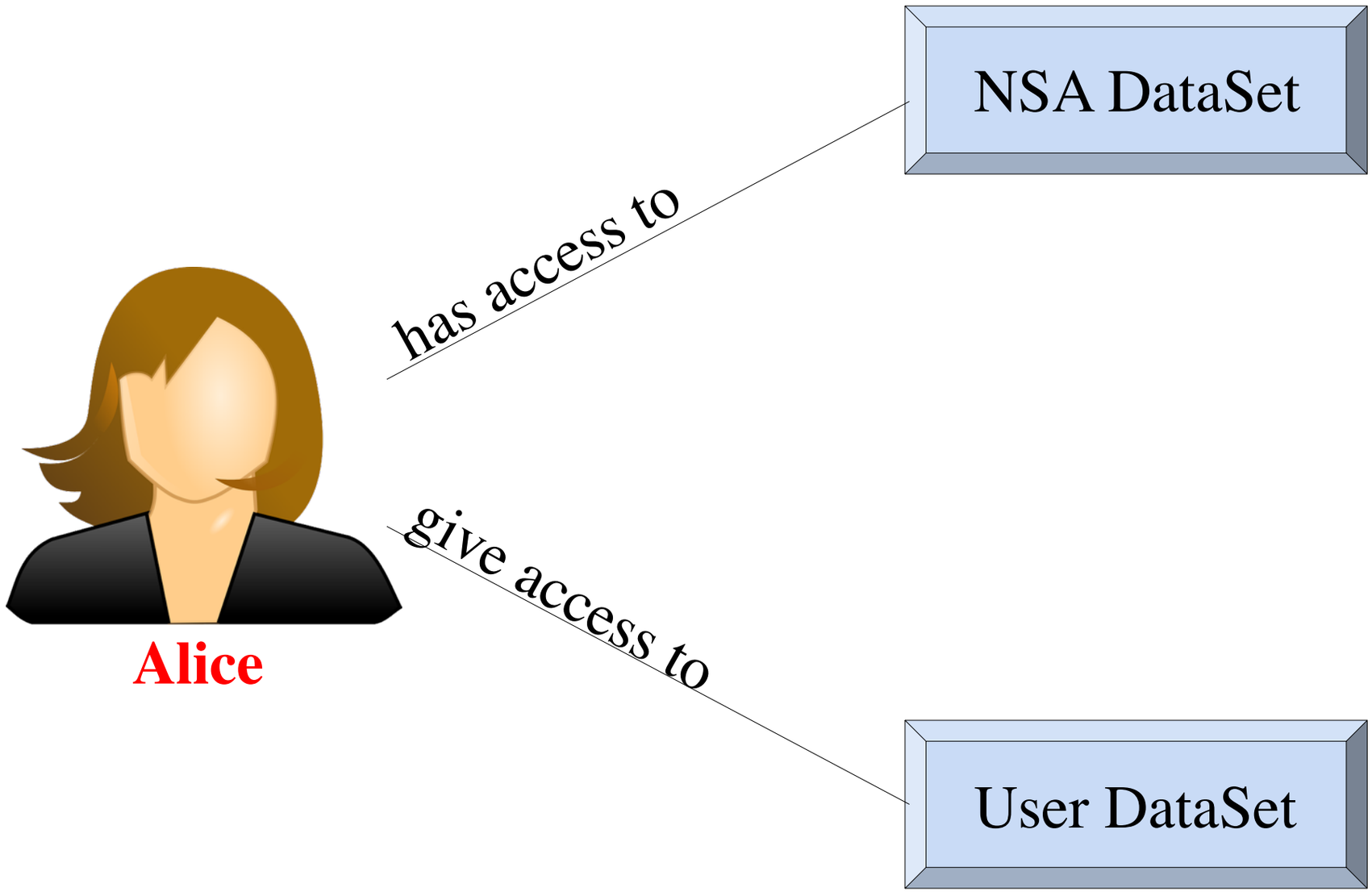
HopsWorks

next

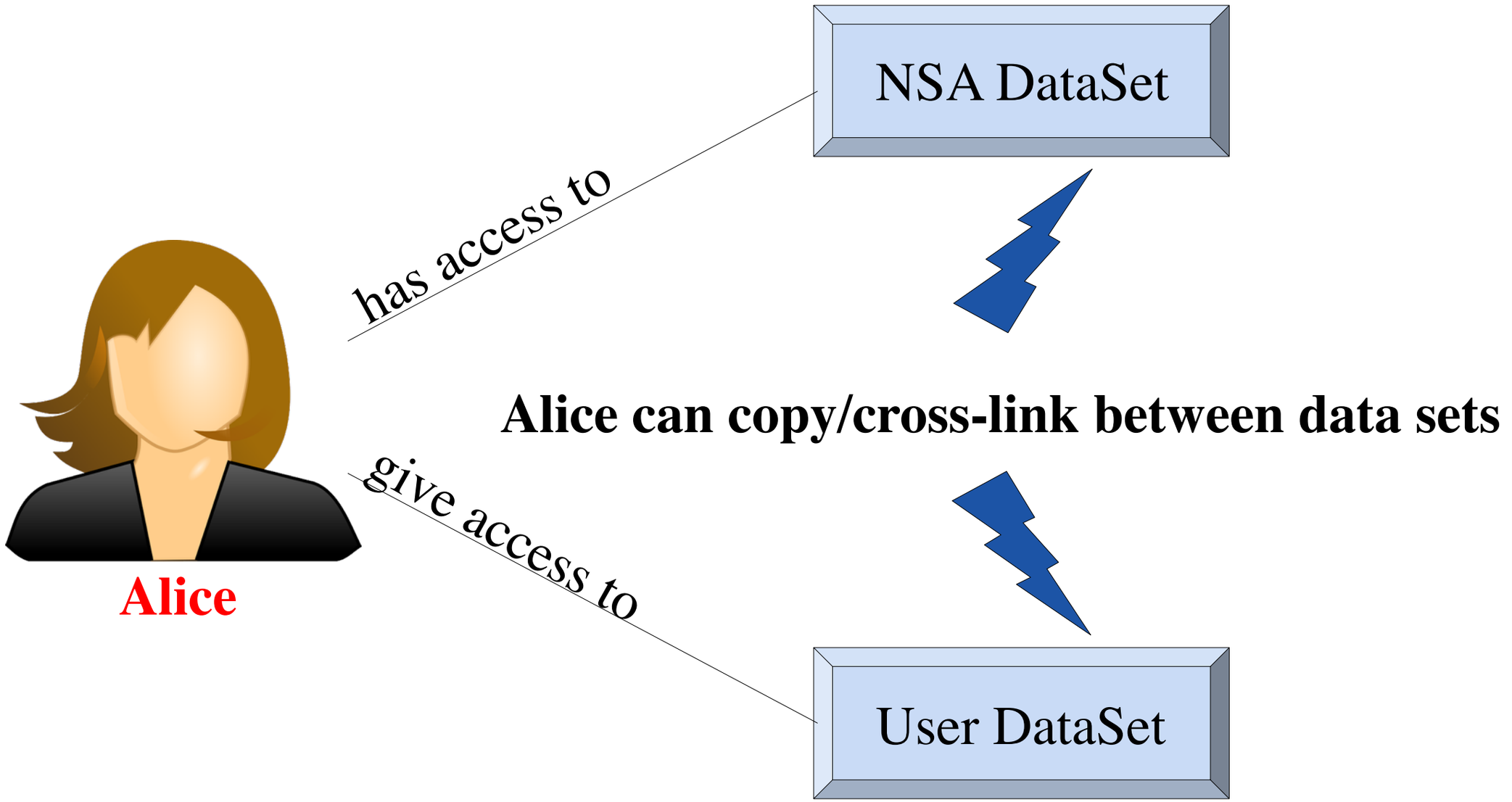
Problem: Sensitive Data needs its own Cluster



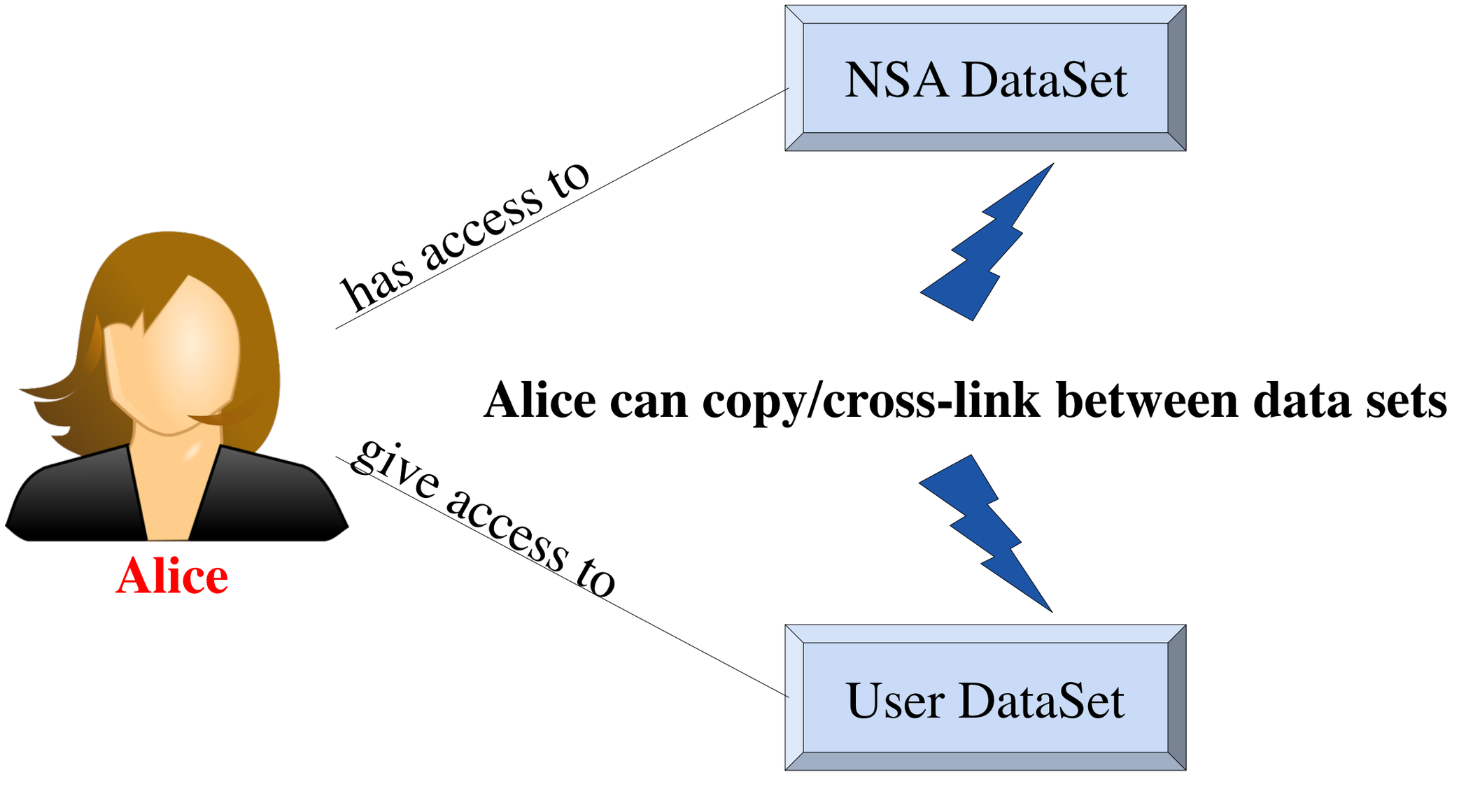
Problem: Sensitive Data needs its own Cluster



Problem: Sensitive Data needs its own Cluster



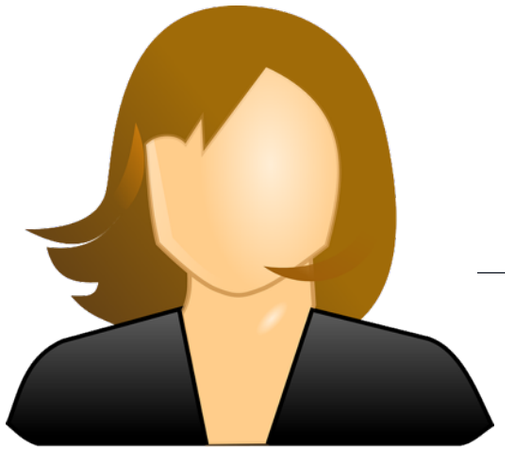
Problem: Sensitive Data needs its own Cluster



Alice has only one Kerberos Identity.

Neither attribute-based access control nor dynamic roles supported in Hadoop.

Solution: Project-Specific UserIDs



NSA__Alice

Member of

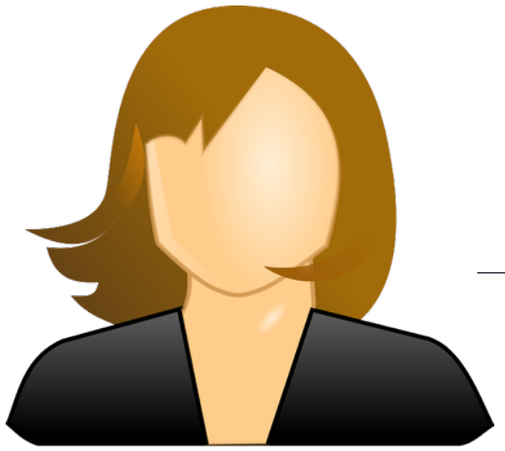
Project NSA

Users__Alice

Member of

Project Users

Solution: Project-Specific UserIDs



NSA__Alice

Member of

Project NSA

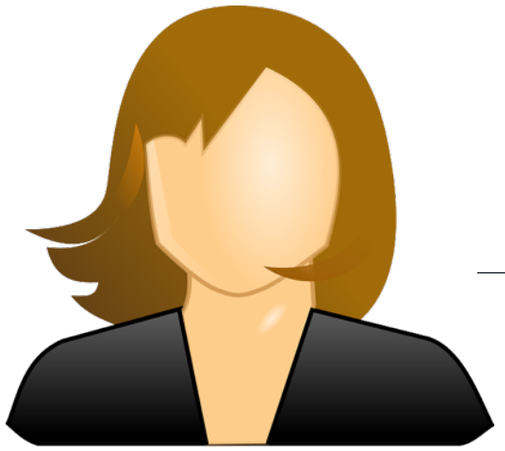
HDFS enforces access control

Users__Alice

Member of

Project Users

Solution: Project-Specific UserIDs



NSA__Alice

Member of

Project NSA

HDFS enforces access control

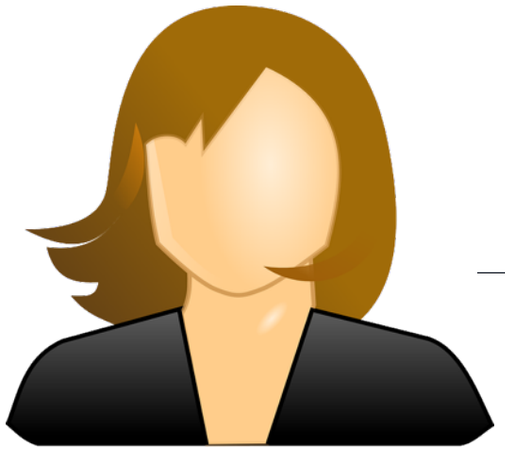
Users__Alice

Member of

Project Users

How can we share DataSets between Projects?

Sharing DataSets between Projects



NSA__Alice

Member of

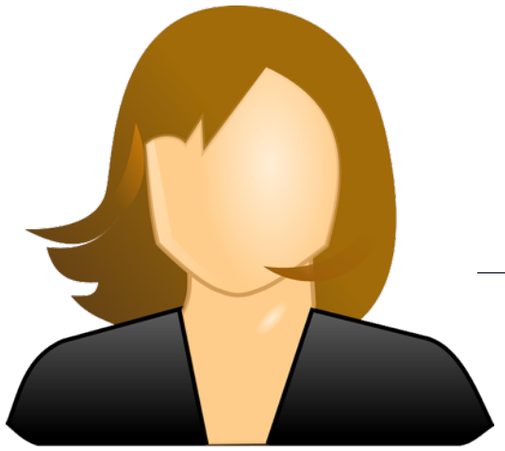
Project NSA

Users__Alice

Member of

Project Users

Sharing DataSets between Projects



NSA__Alice

Member of

Project NSA

Users__Alice

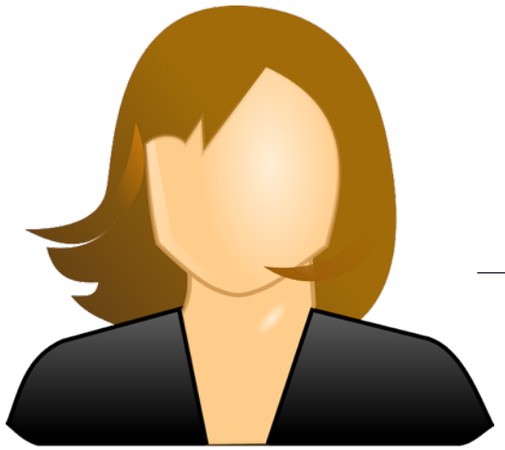
Member of

Project Users

owns

DataSet

Sharing DataSets between Projects



NSA__Alice

Member of

Project NSA

Add members of Project NSA to the DataSet group

Users__Alice

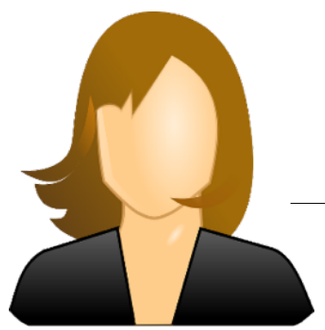
Member of

Project Users

owns

DataSet

HopsWorks enforces dynamic Roles



Alice@gmail.com

Authenticate

HopsWorks

Projects

NSA__Alice

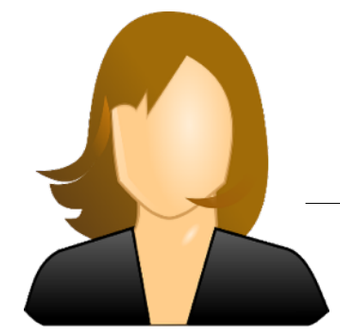
Users__Alice

HopsYARN

HopsFS

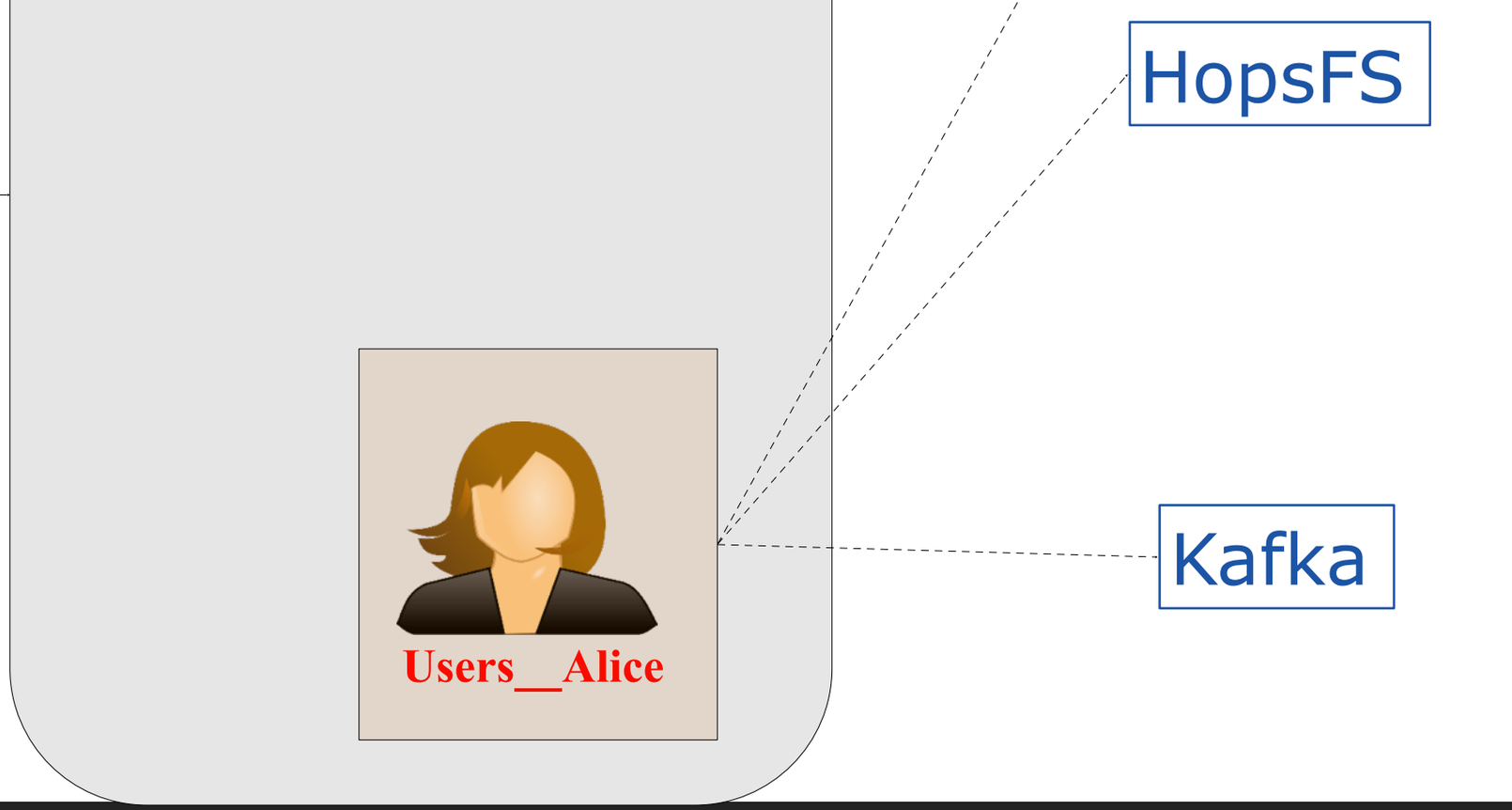
Kafka

HopsWorks enforces dynamic Roles

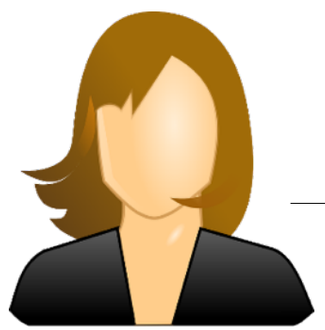


Alice@gmail.com

Authenticate



HopsWorks enforces dynamic Roles



Alice@gmail.com

Authenticate

HopsWorks

Projects

NSA__Alice

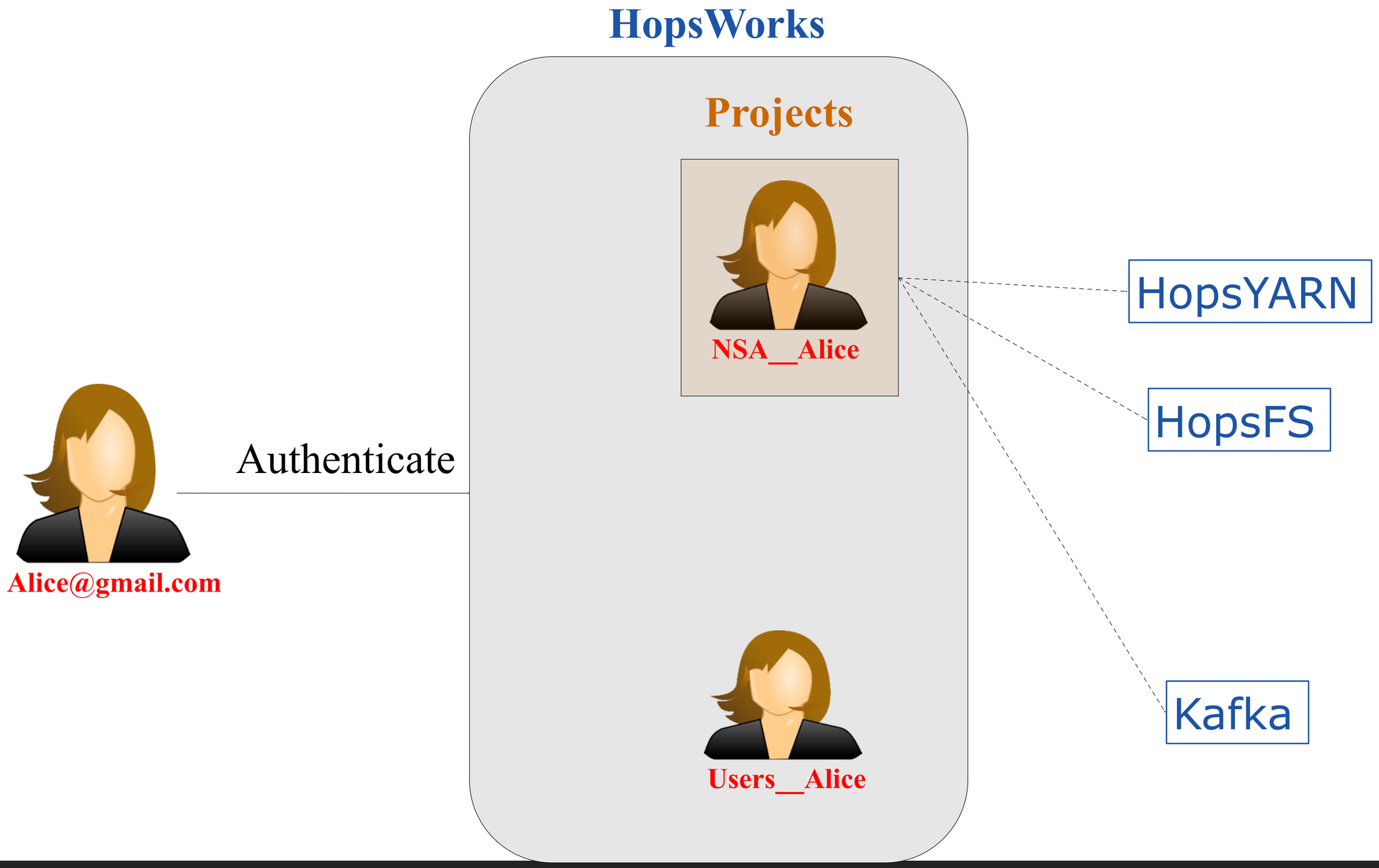
Users__Alice

HopsYARN

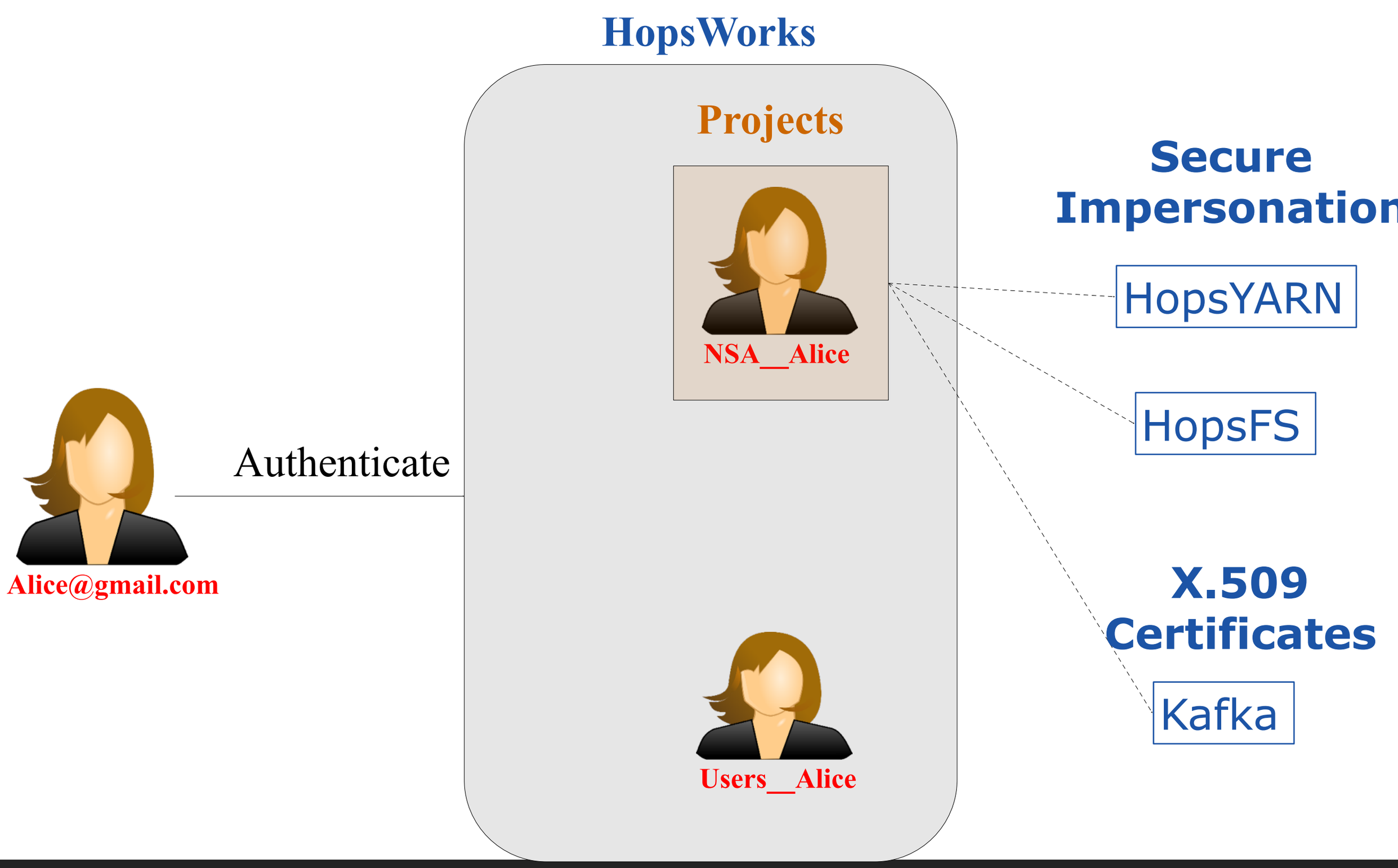
HopsFS

Kafka

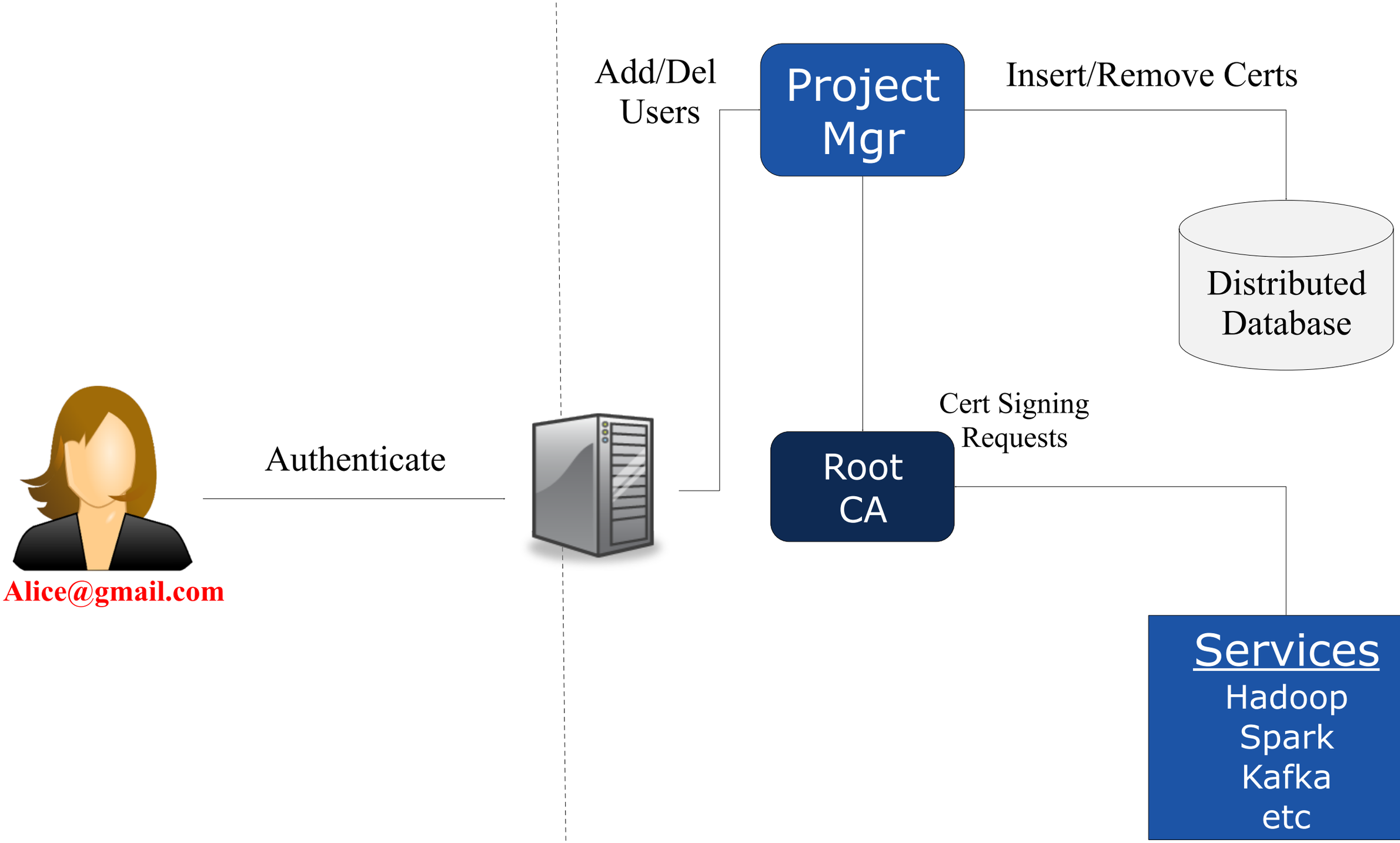
HopsWorks enforces dynamic Roles



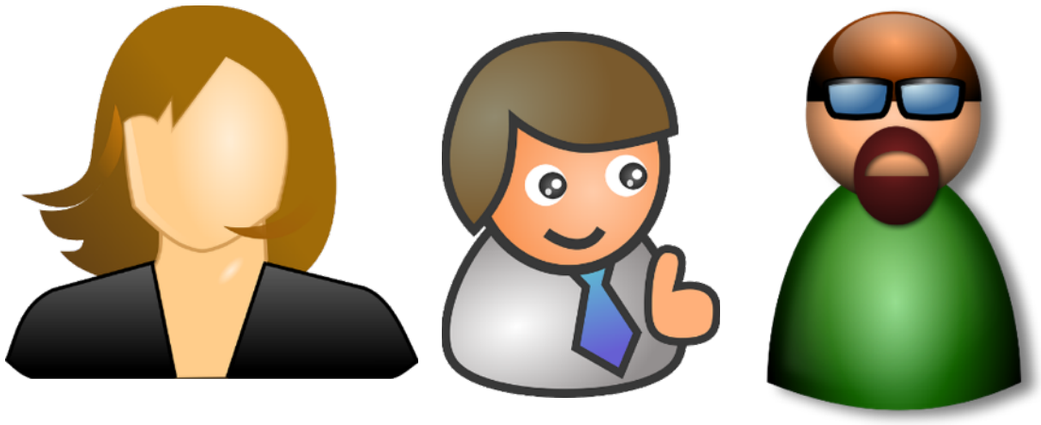
HopsWorks enforces dynamic Roles

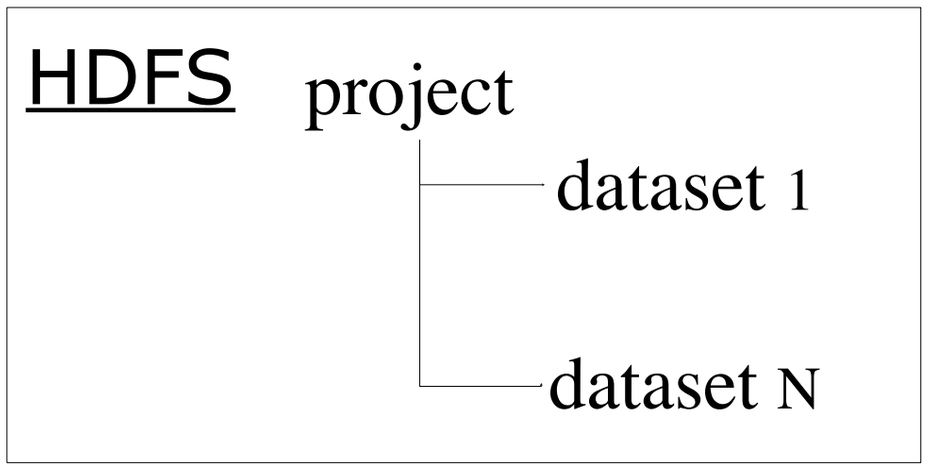


X.509 Certificate per project specific user



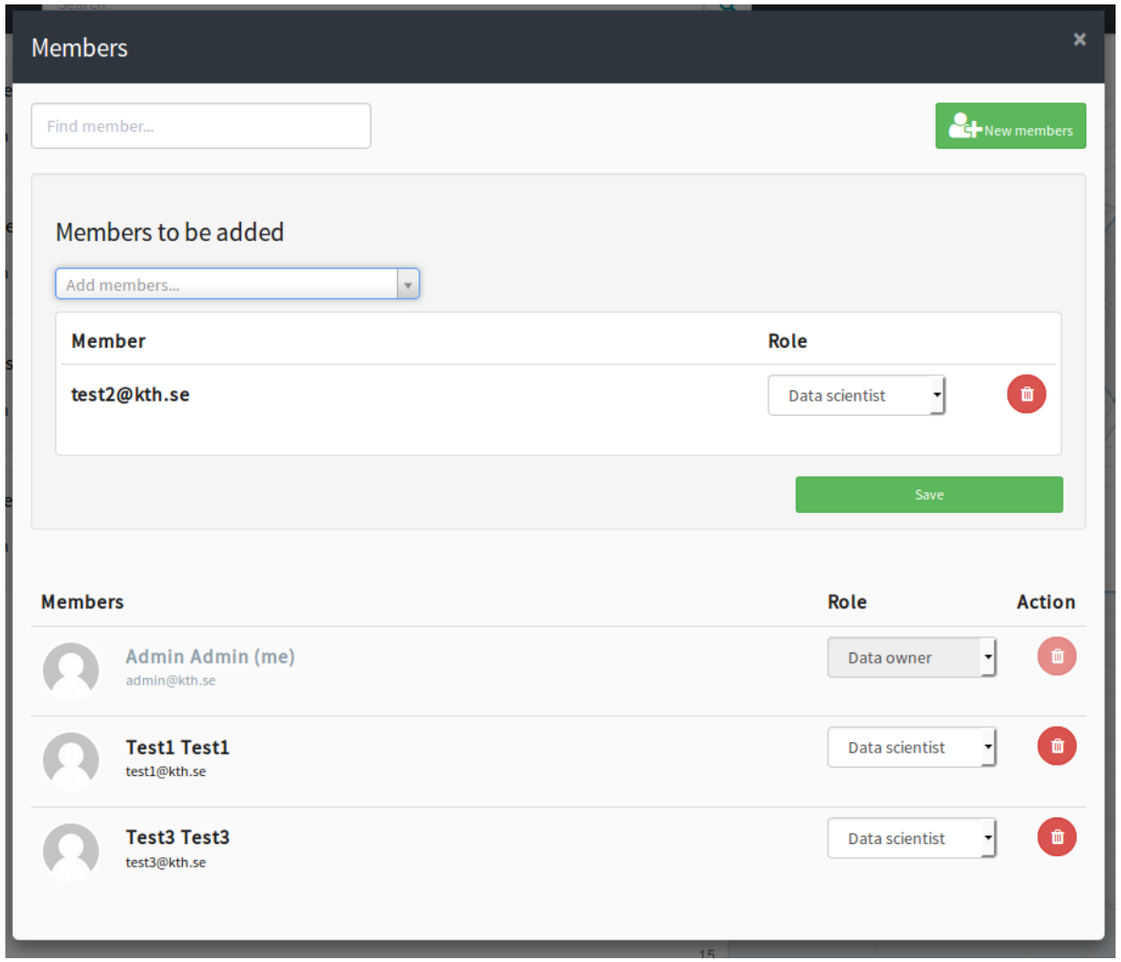
Project





- A project is a collection of
  - Members
  - HDFS DataSets
  - Kafka Topics
  - Notebooks, Jobs
- A project has an owner
- A project has quotas

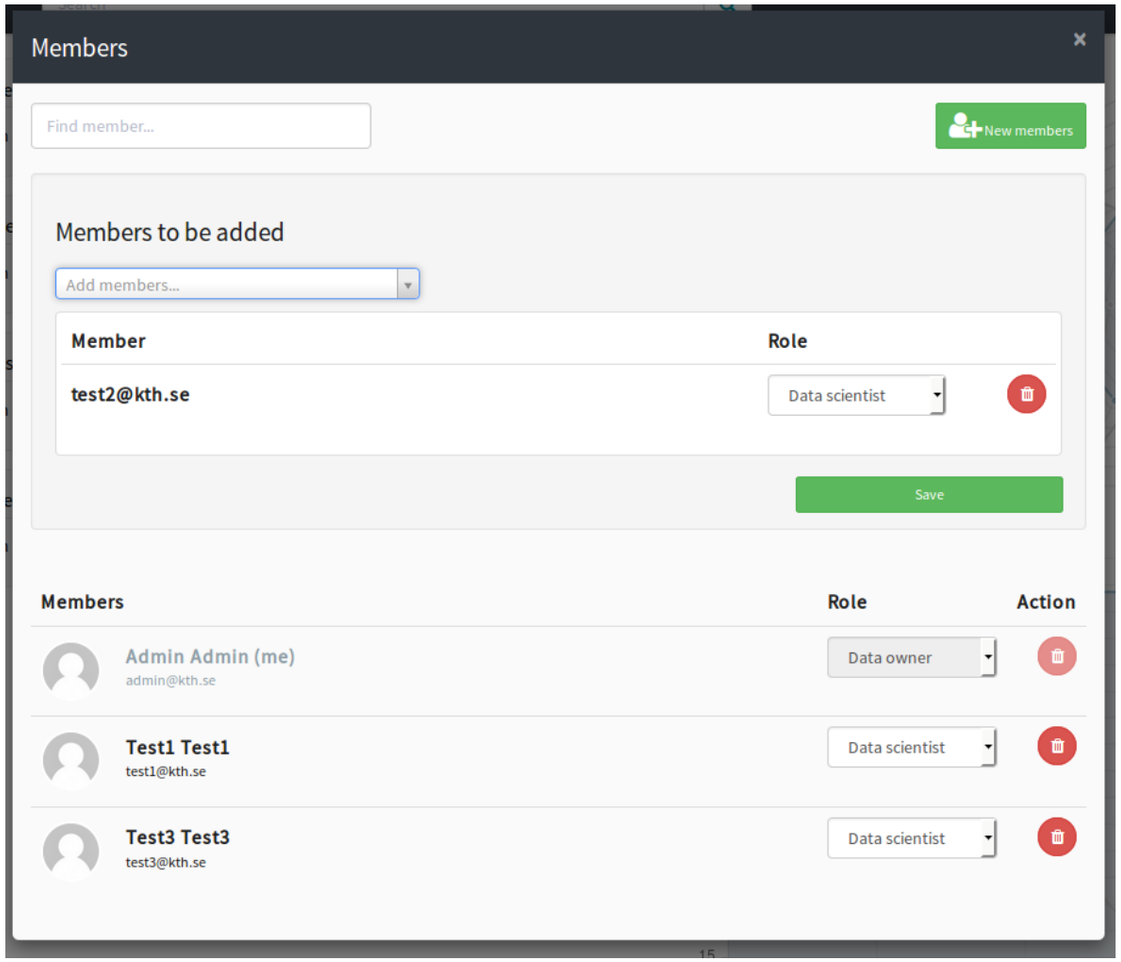
Project Roles



- Data Scientist Privileges

  - Write and Run code

Project Roles



- Data Scientist Privileges

  - Write and Run code

We delegate administration of privileges to users