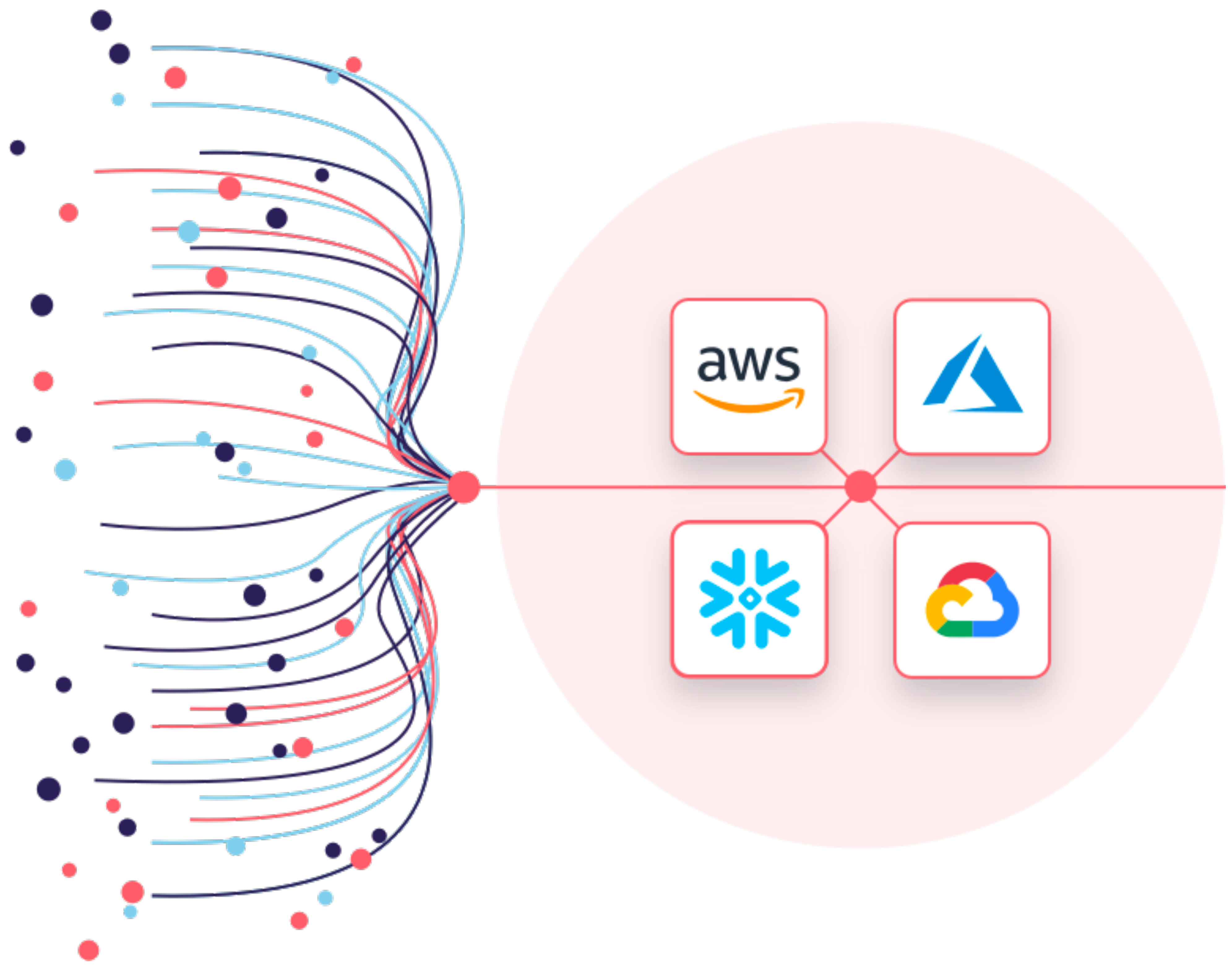
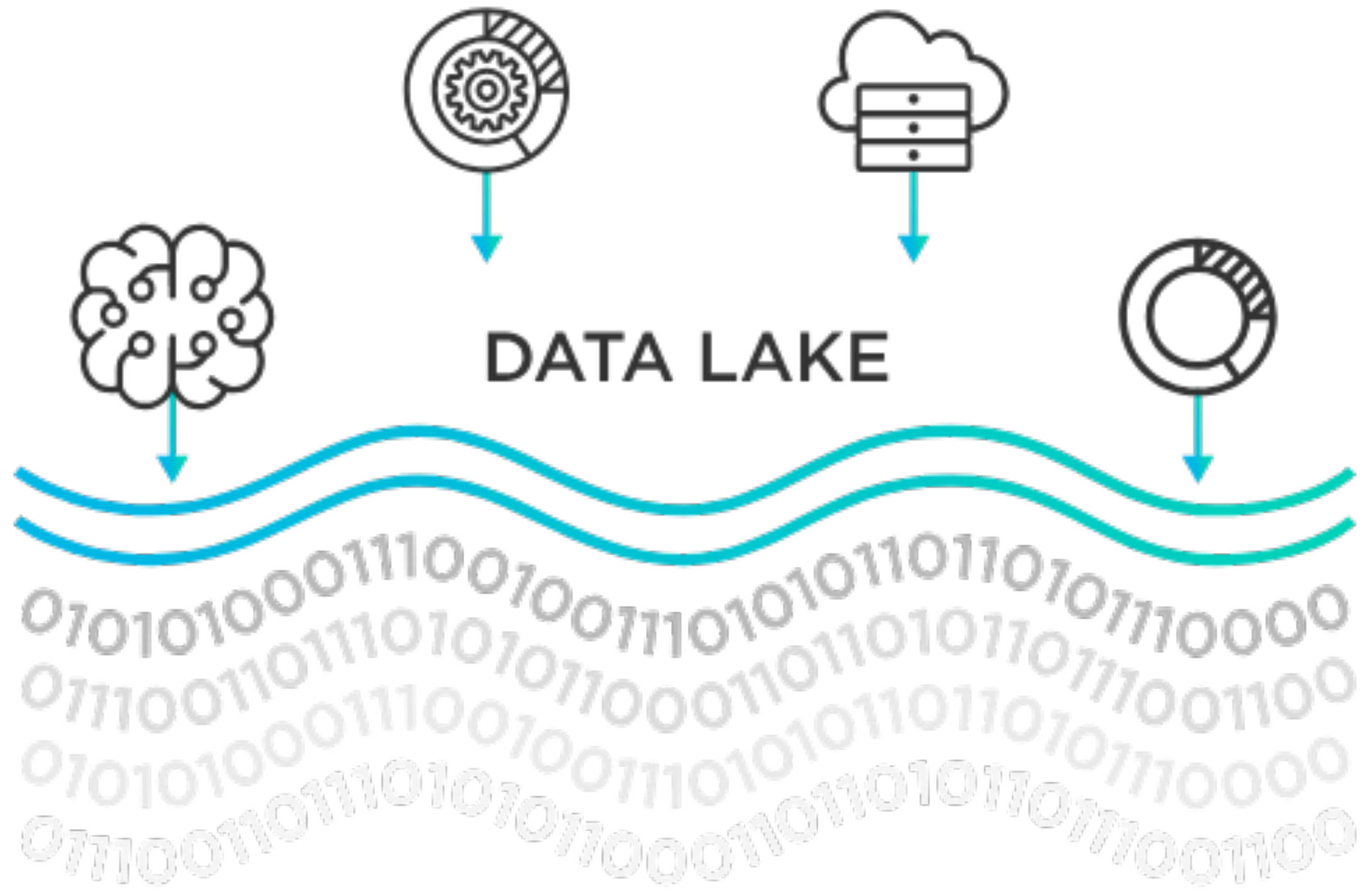
FileScale

Fast and Elastic Metadata Management for Distributed File Systems

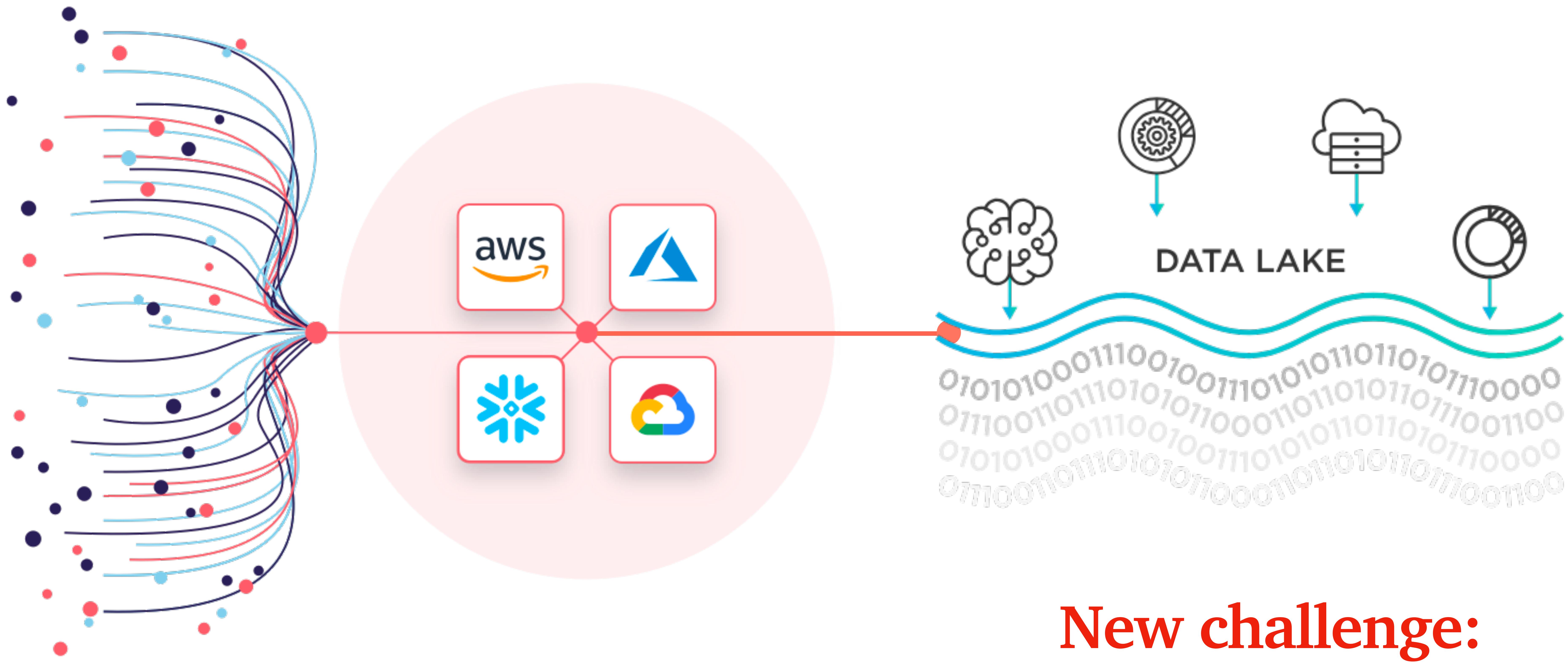
Gang Liao and Daniel J. Abadi

Data Warehouse / LakeHouse





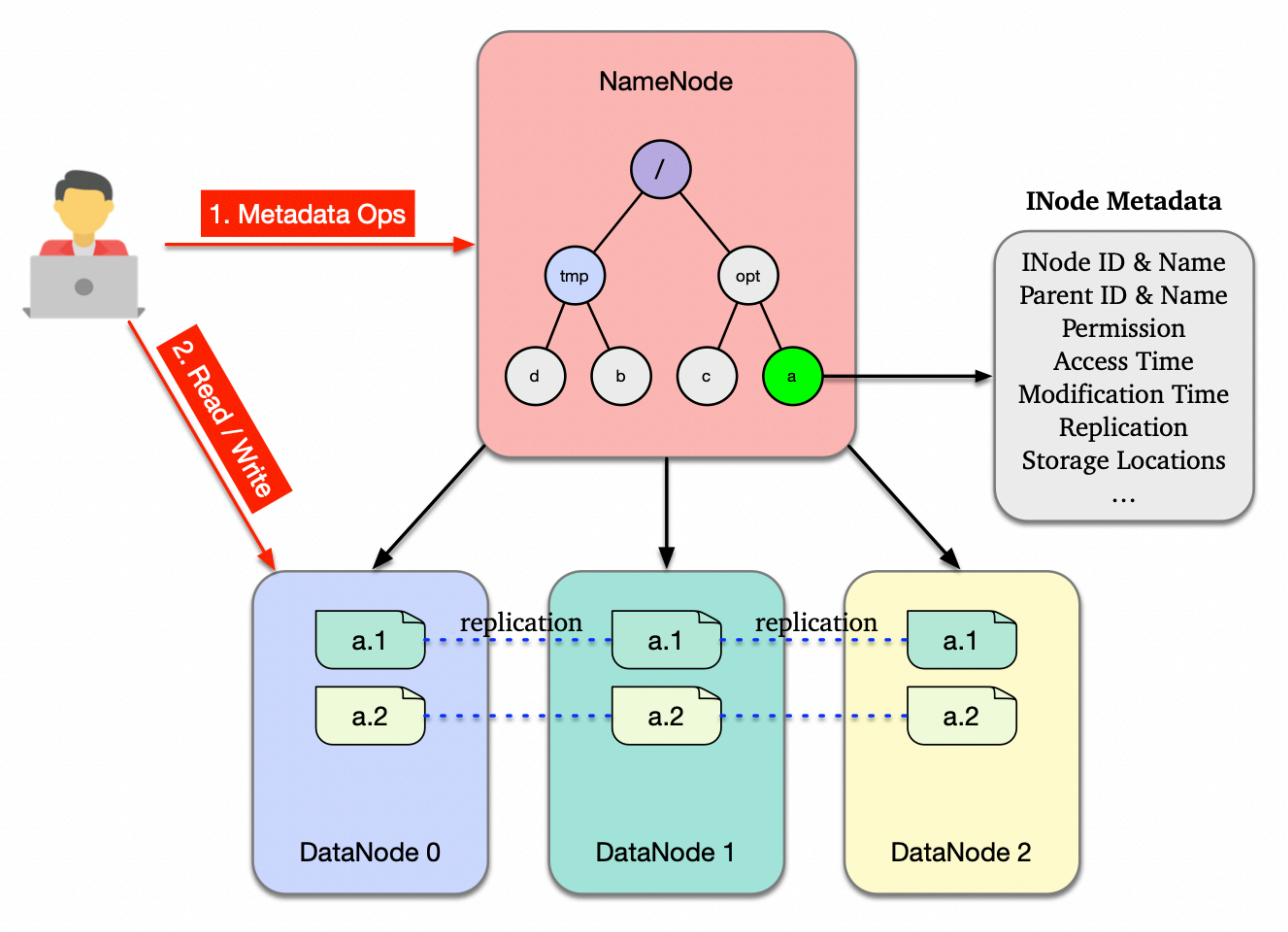
Storage Systems



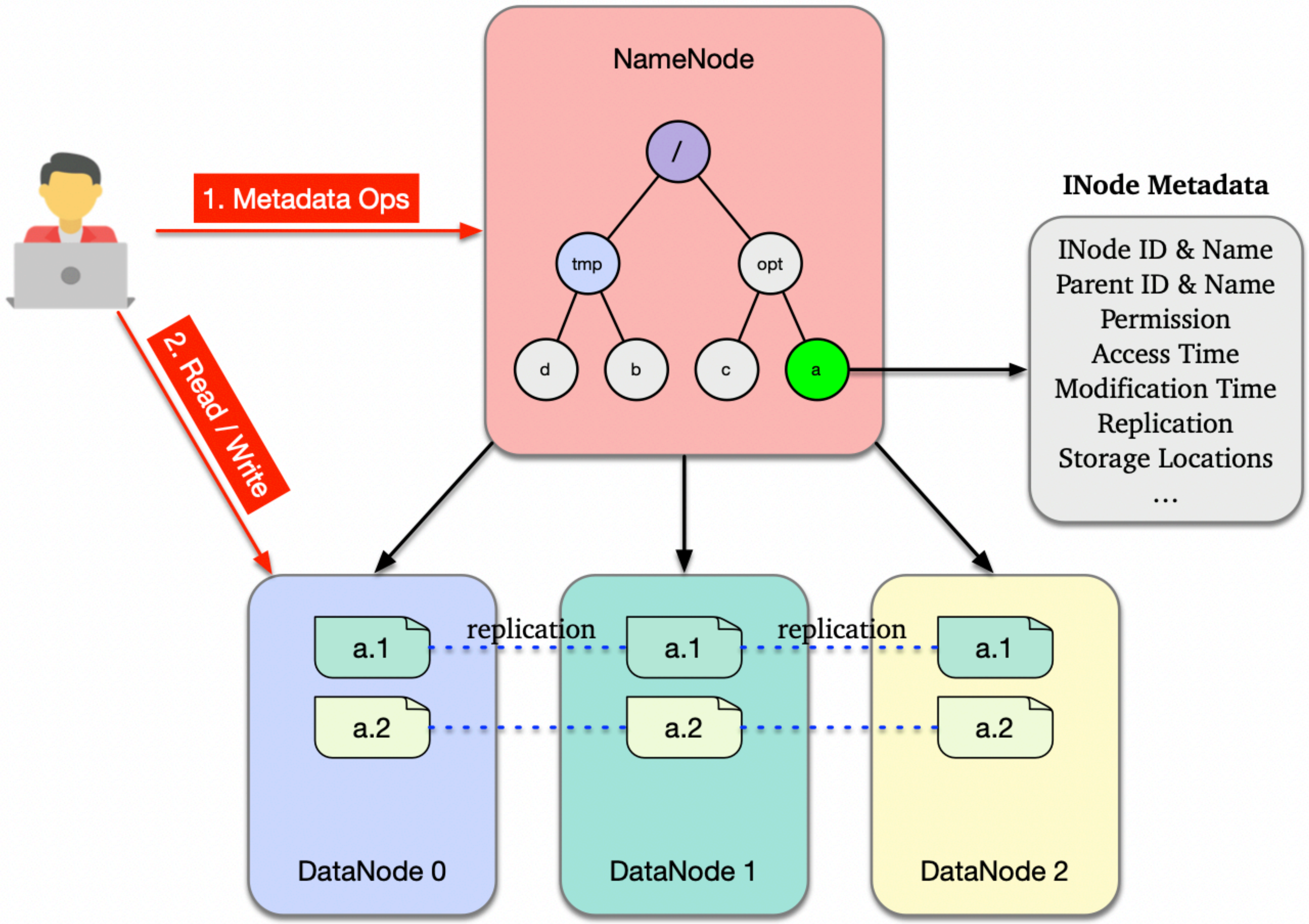
- Amazon S3
- Google Colossus
- HDFS

New challenge:
When metadata is big data

Metadata Management in HDFS



Metadata Management in HDFS

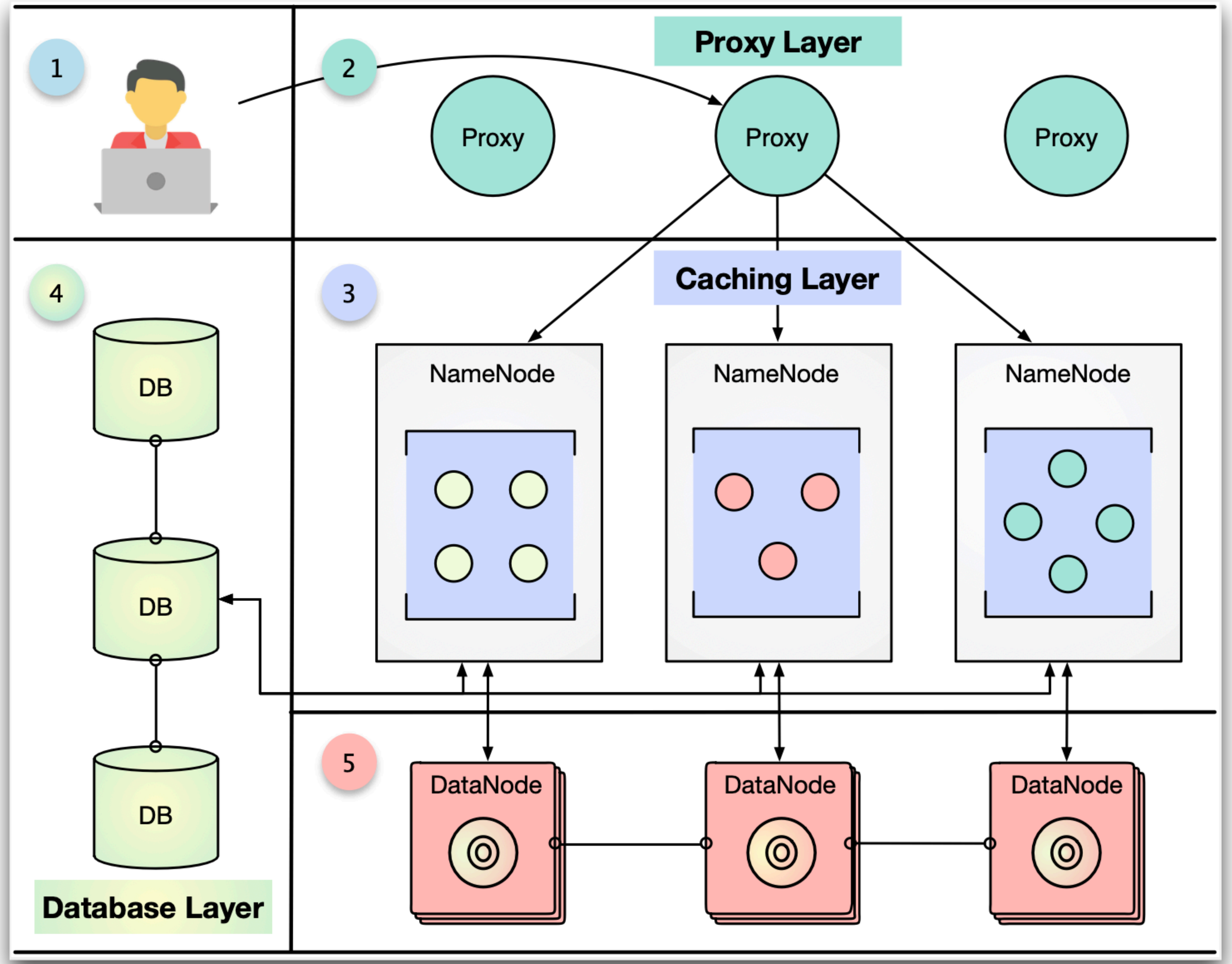


Scalability Problems

- Memory bottleneck
- Network bottleneck
  - Concurrent user requests
  - DataNodes heartbeats

FileScale

A three-tiered architecture

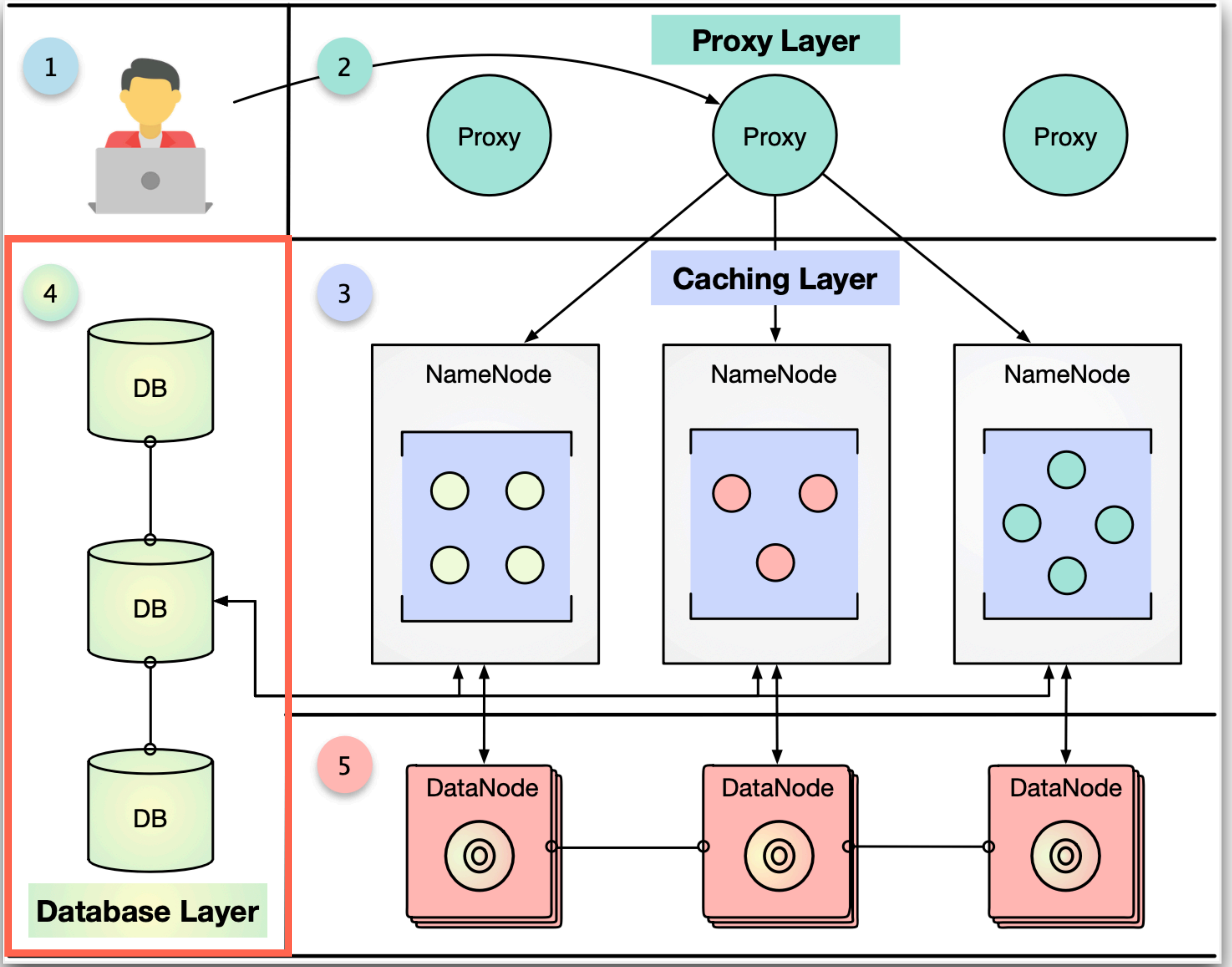


System Architecture of FileScale

FileScale

A three-tiered architecture

- **Database Layer**
  - ACID-compliant SQL database systems: VoltDB, Apache Ignite, etc.
  - Relational data model
  - Distributed transactions
  - Pre-compiled stored procedures



System Architecture of FileScale

FileScale - Database Layer

Primary key (parent name, inode name) → full path

inode2block

block-id	id	index
1073741825	16386	0
1073741826	16386	1
1073741827	16386	2
1073741828	16388	0
1073741829	16389	0

datablocks

block-id	kbytes	stamp	replica
1073741825	131072	1001	1
1073741826	131072	1002	1
1073741827	45056	1003	1
1073741828	6.6	1004	1
1073741829	1628.2	1005	1

block2storage

block-id	idx	storage-id
1073741825	0	DS-e3d5de23
1073741826	0	DS-e3d5de23
1073741827	0	DS-08989547
1073741828	0	DS-dc8aa54e
1073741829	0	DS-dc8aa54e

inodes

id	pid	pname	name	access-time	update-time	header	permission
16385	0	null	/	0	1545261571024	0	1099511693805
16386	16385	/	event_data	1545267685278	1545264231090	281474976710672	1099511693823
16387	16385	/	dnn_model	0	1545267685104	0	1099511693805
16388	16386	/dnn_model	graph.ckpt.pbtxt	1545267685125	1545267685125	281474976710672	1099511693823
16389	16386	/dnn_model	model.ckpt.data0	1545267685224	1545267685224	281474976710672	1099511693823

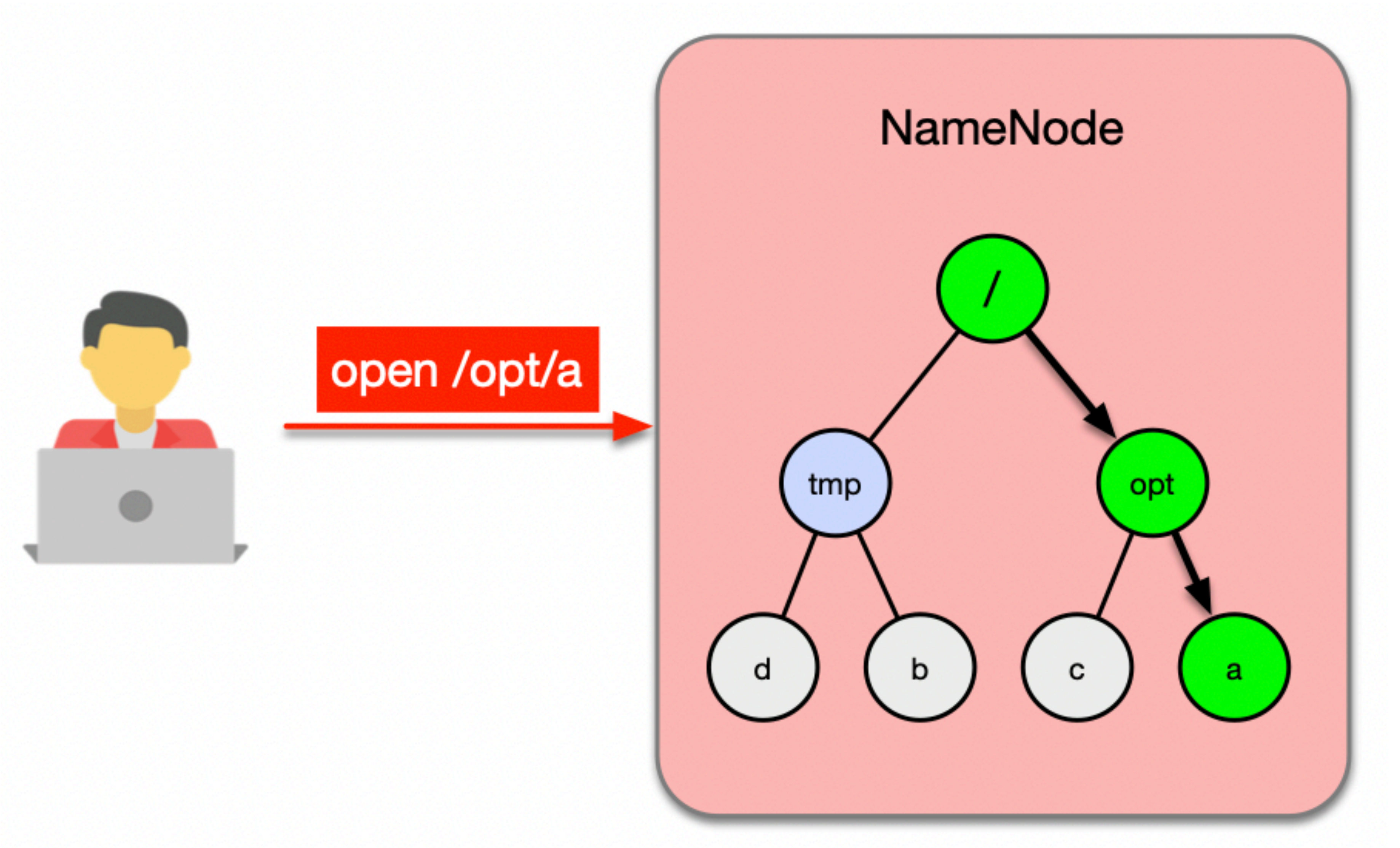
Data Model in FileScale

FileScale - Database Layer

Primary key (parent name, inode name) → full path

Compared with using id as the primary key, what are the advantages?

- Path Resolution: validate the entire path and check user permissions and quota configuration recursively



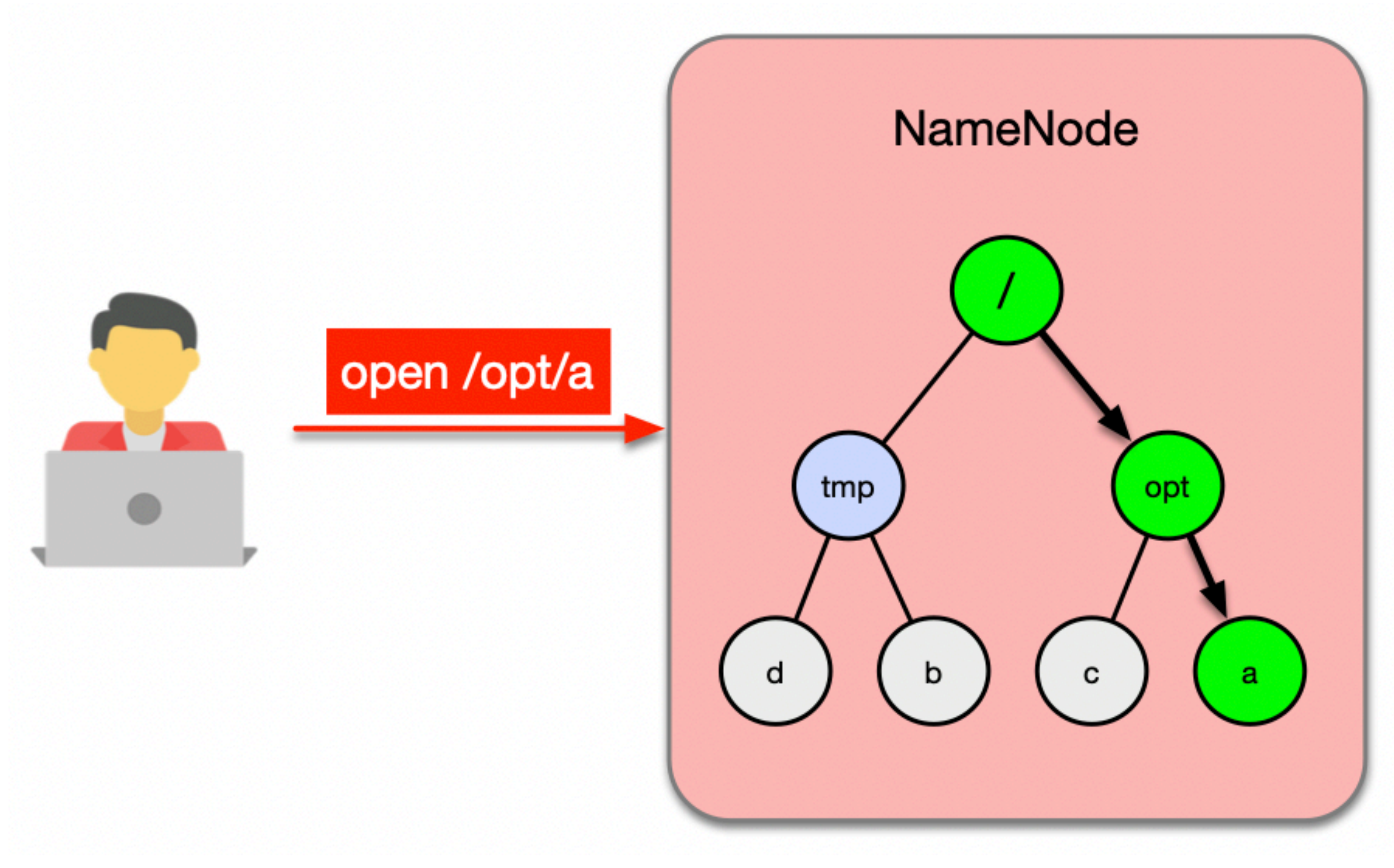
HDFS

FileScale - Database Layer

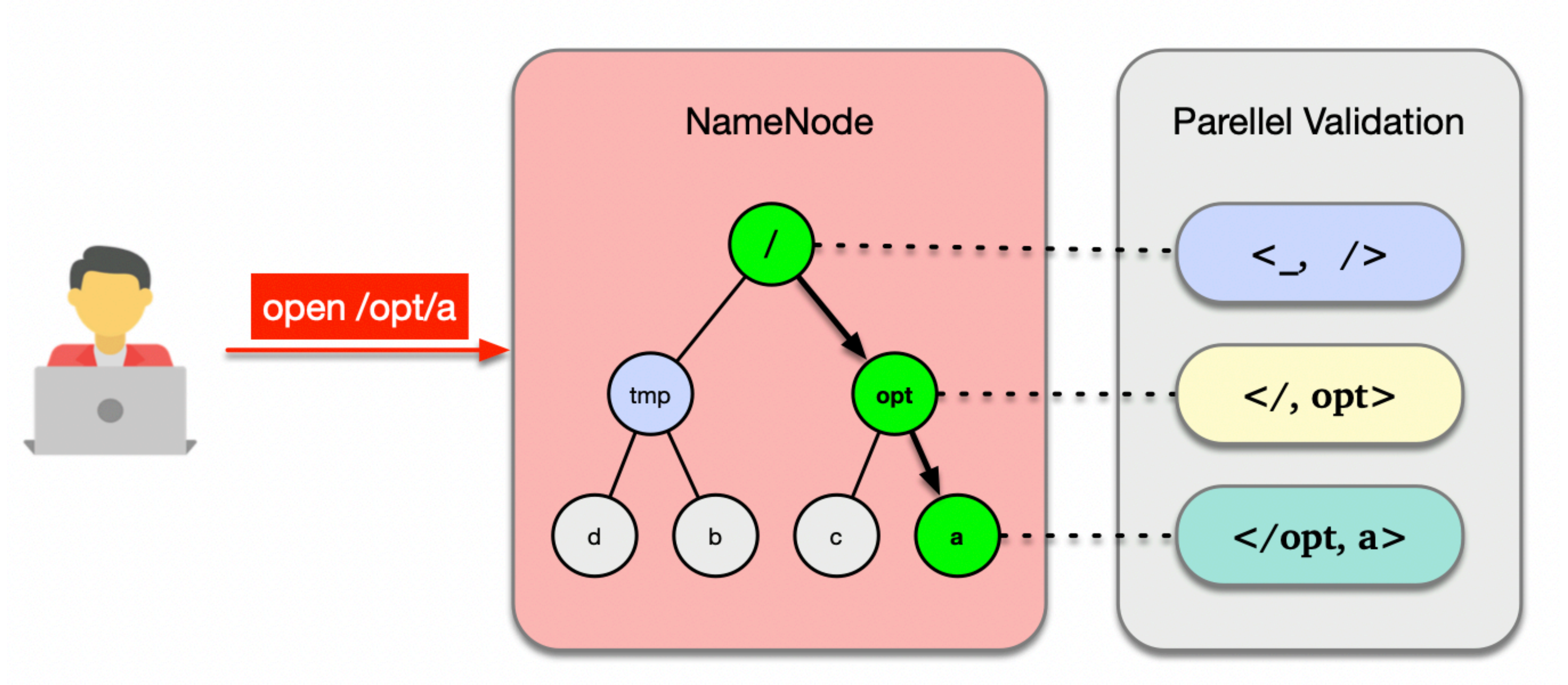
Primary key (parent name, inode name) → full path

Compared with using id as the primary key, what are the advantages?

- Parallel Path Resolution



HDFS



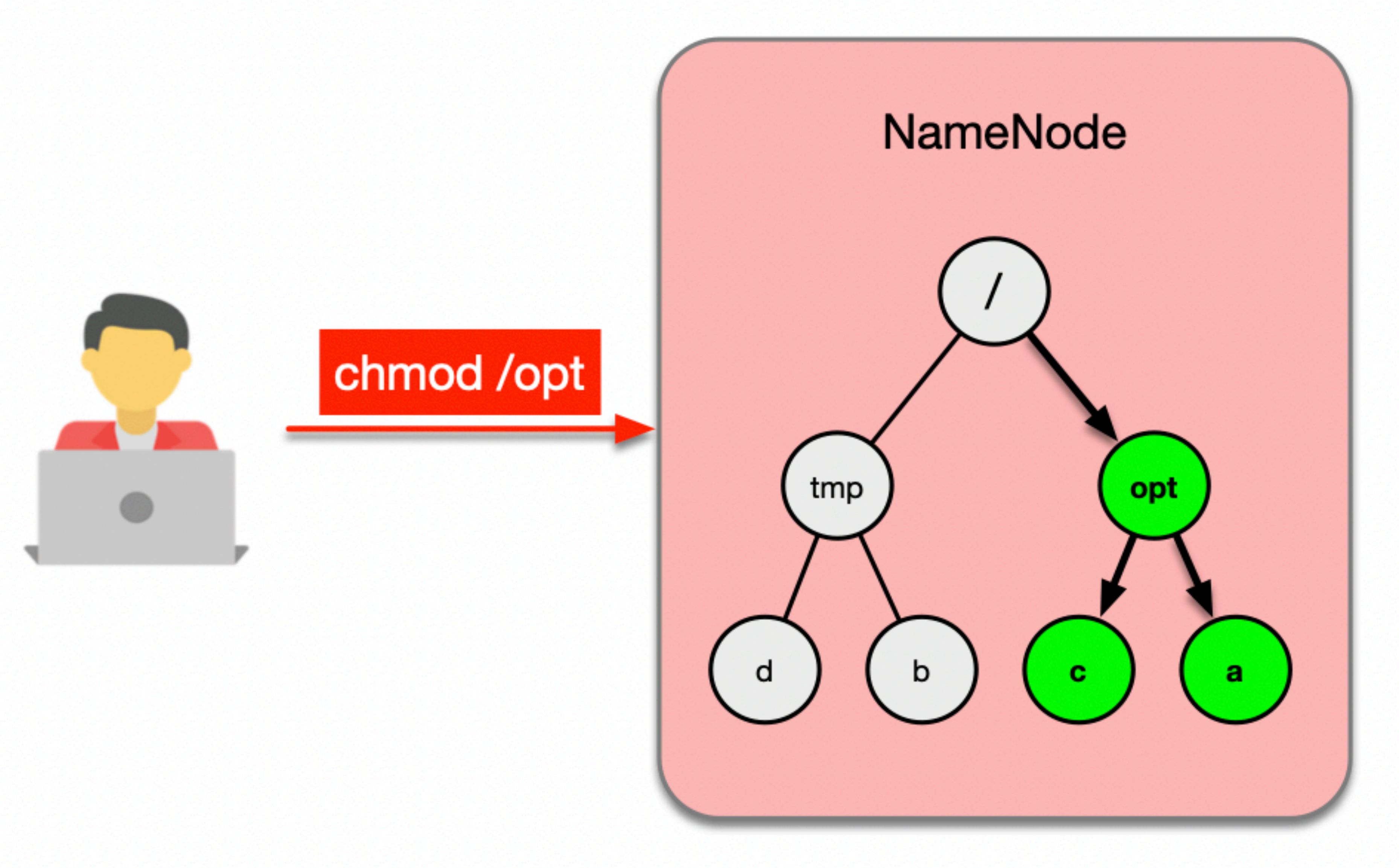
FileScale

FileScale - Database Layer

Primary key (parent name, inode name) → full path

Compared with using id as the primary key, what are the advantages?

- Parallel Path Resolution
- **Subtree operations**



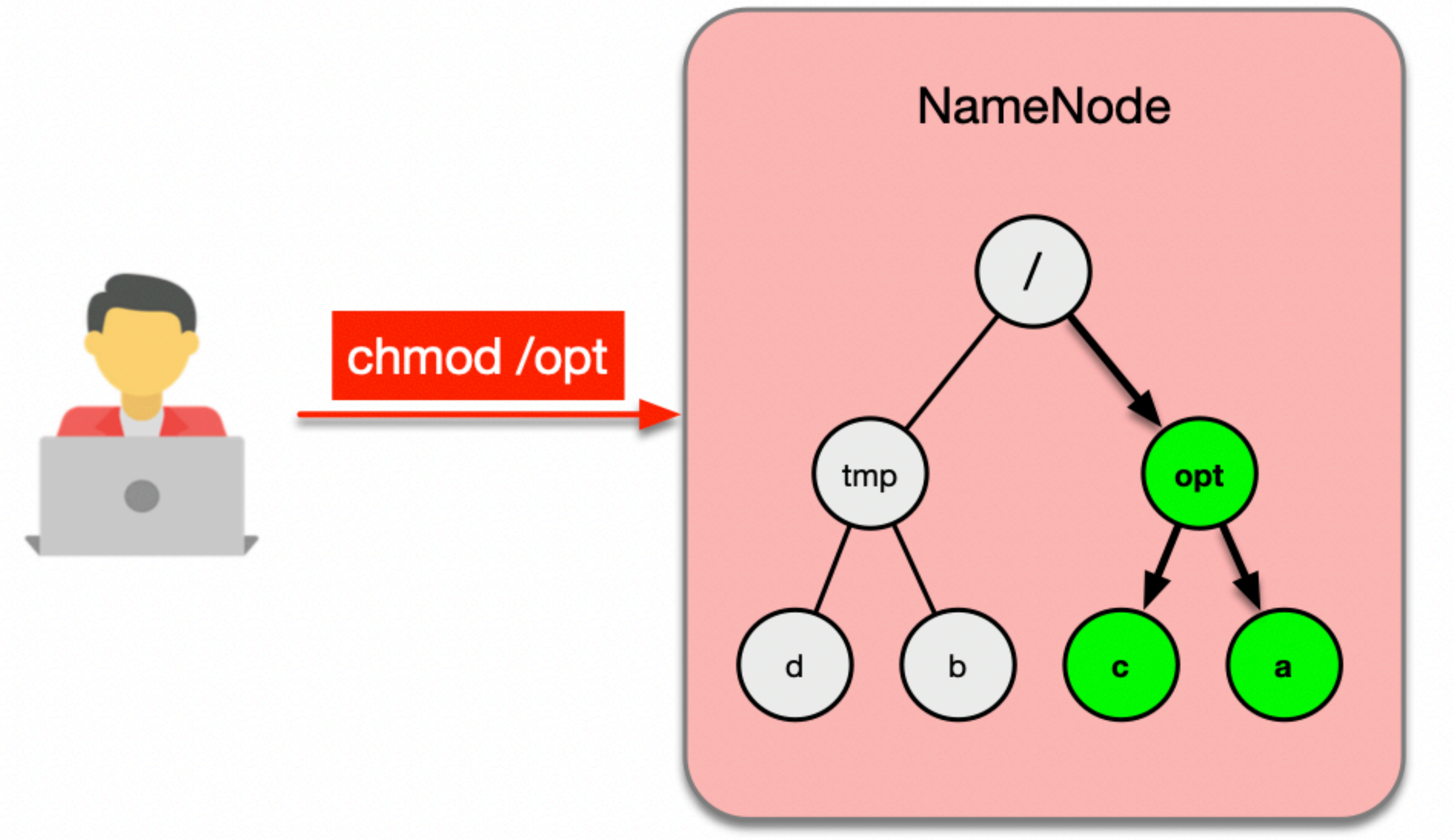
FileScale

FileScale - Database Layer

Primary key (parent name, inode name) → full path

Compared with using id as the primary key, what are the advantages?

- Parallel Path Resolution
- **Subtree operations via the SQL LIKE or STARTS WITH clause**



FileScale

Primary Key <Parent Name, INode Name>

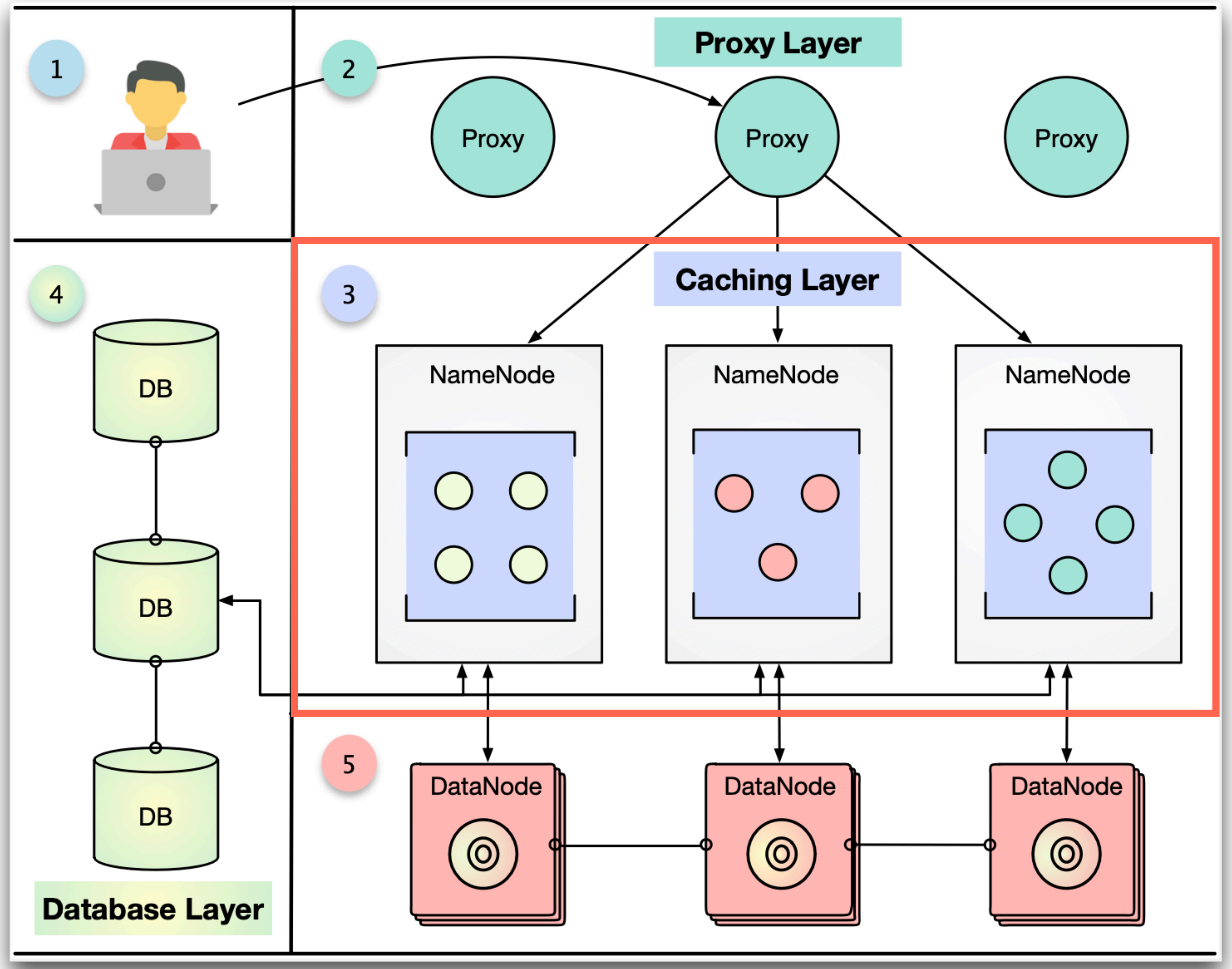
```
# 1. Update all children in the subtree
UPDATE inodes SET permission = ?
   WHERE parent_name STARTS WITH ?;

# 2. Update the root inode of the subtree
UPDATE inodes SET permission = ?
   WHERE parent_name = ? AND inode_name = ?;
```

Chmod Operations

FileScale - Caching Layer

A three-tiered architecture

- Database Layer
- **Caching Layer**
  - Object cache<fullpath, inode object>
  - Cache eviction policies



System Architecture of FileScale

FileScale - Caching Layer

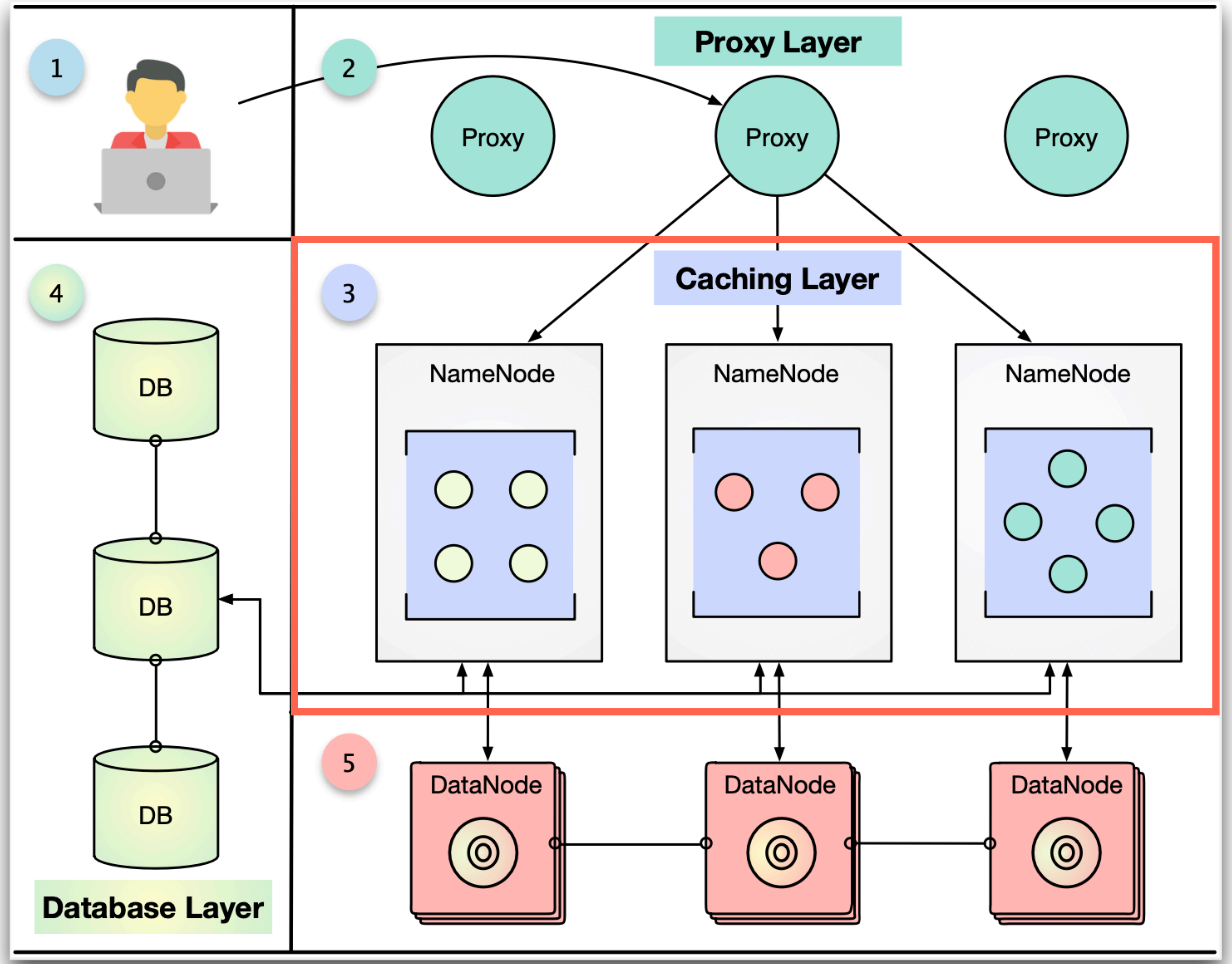
A three-tiered architecture

- Database Layer

Async propagation
- Periodic flush

Sync propagation
- Expiration
- Multi-partition requests

- **Caching Layer**
  - Object cache<fullpath, inode object>
  - Cache eviction policies

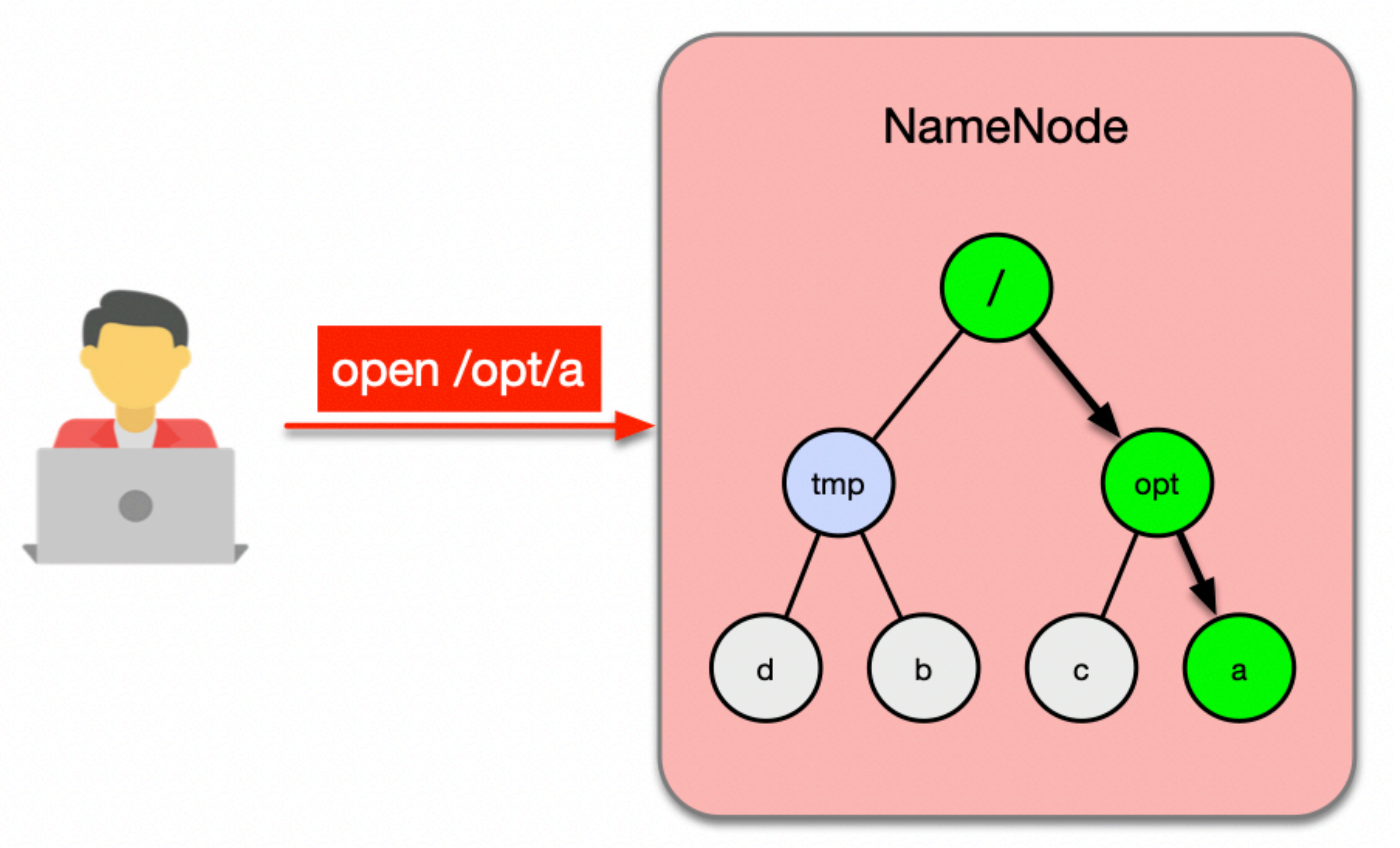


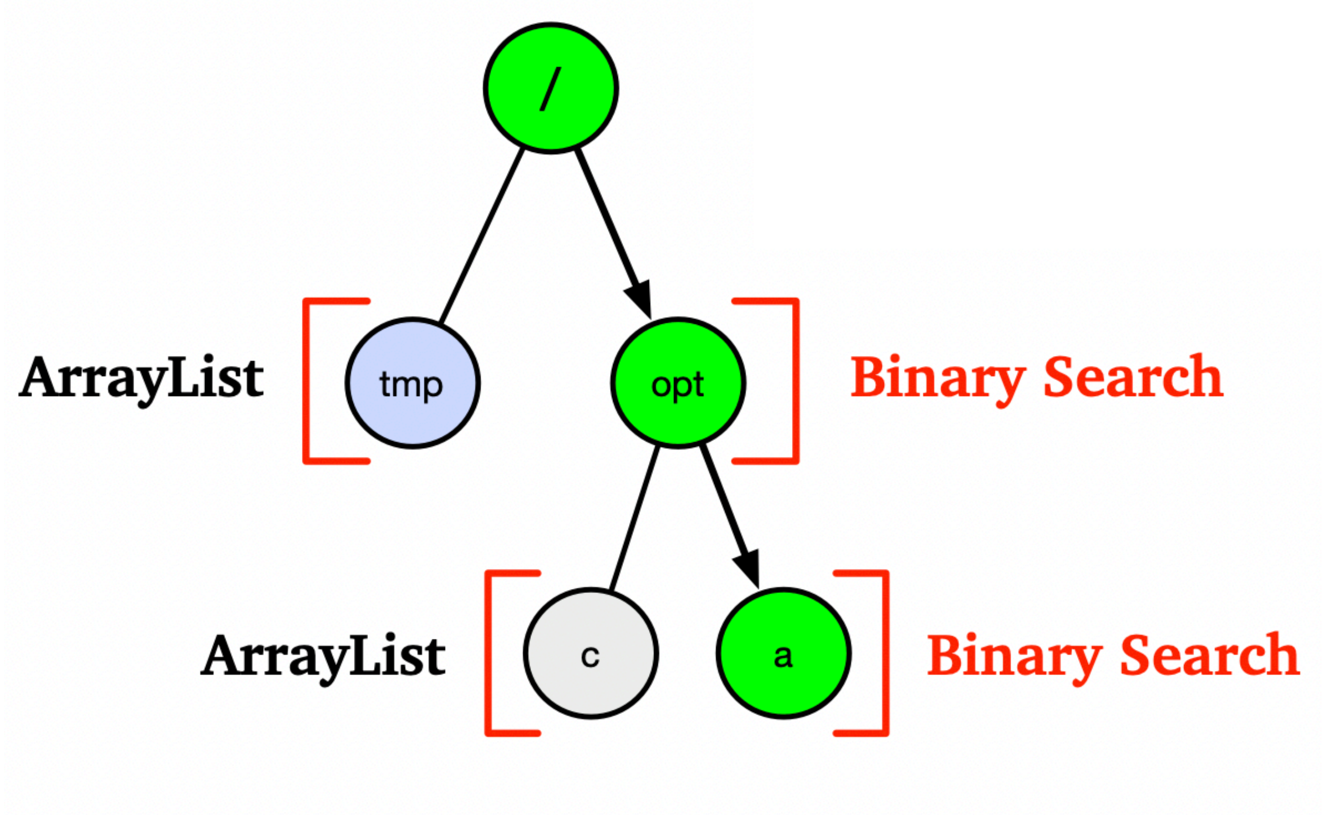
System Architecture of FileScale

FileScale - Caching Layer

What is the difference between HDFS and FileScale?

- HDFS: in-memory pointers between directory and files



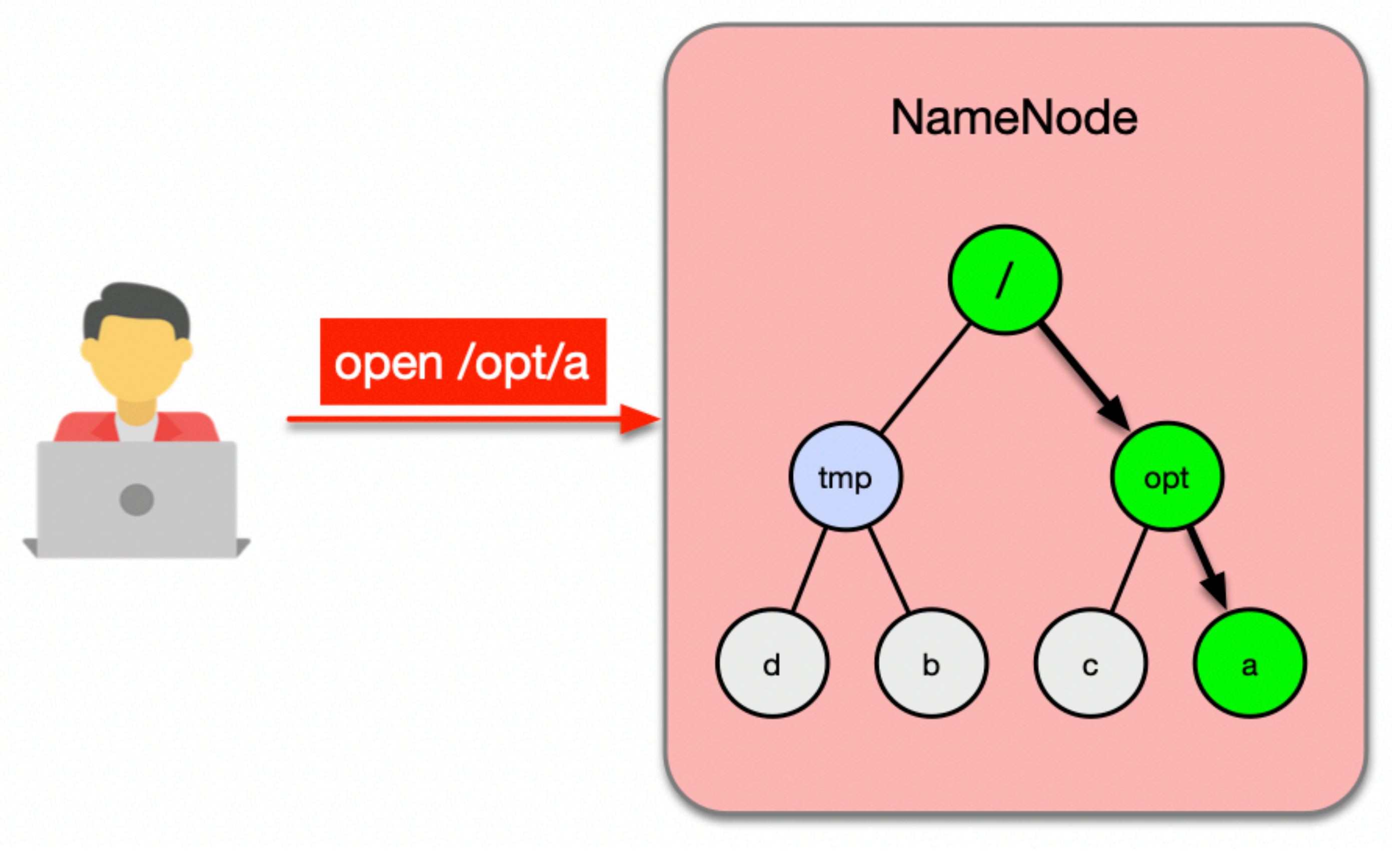


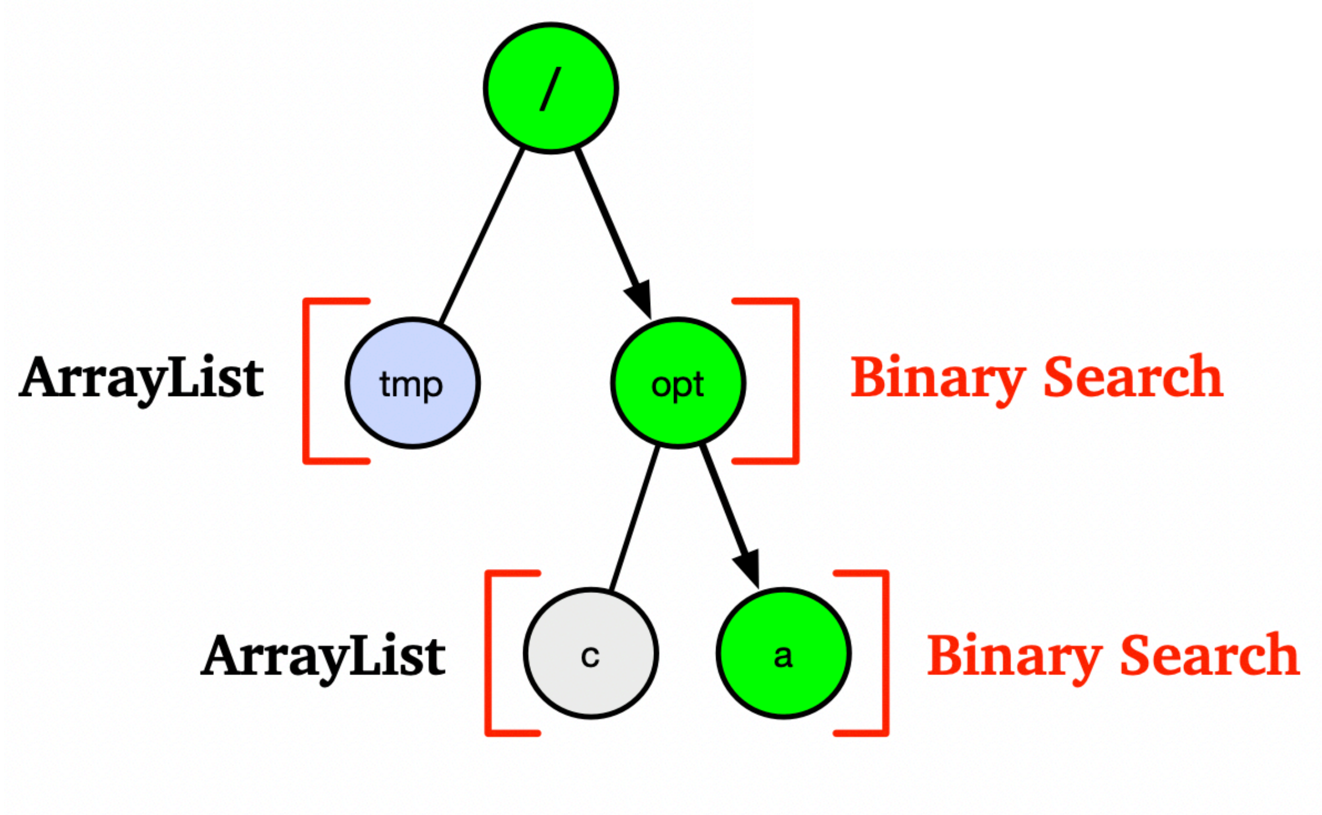
HDFS

FileScale - Caching Layer

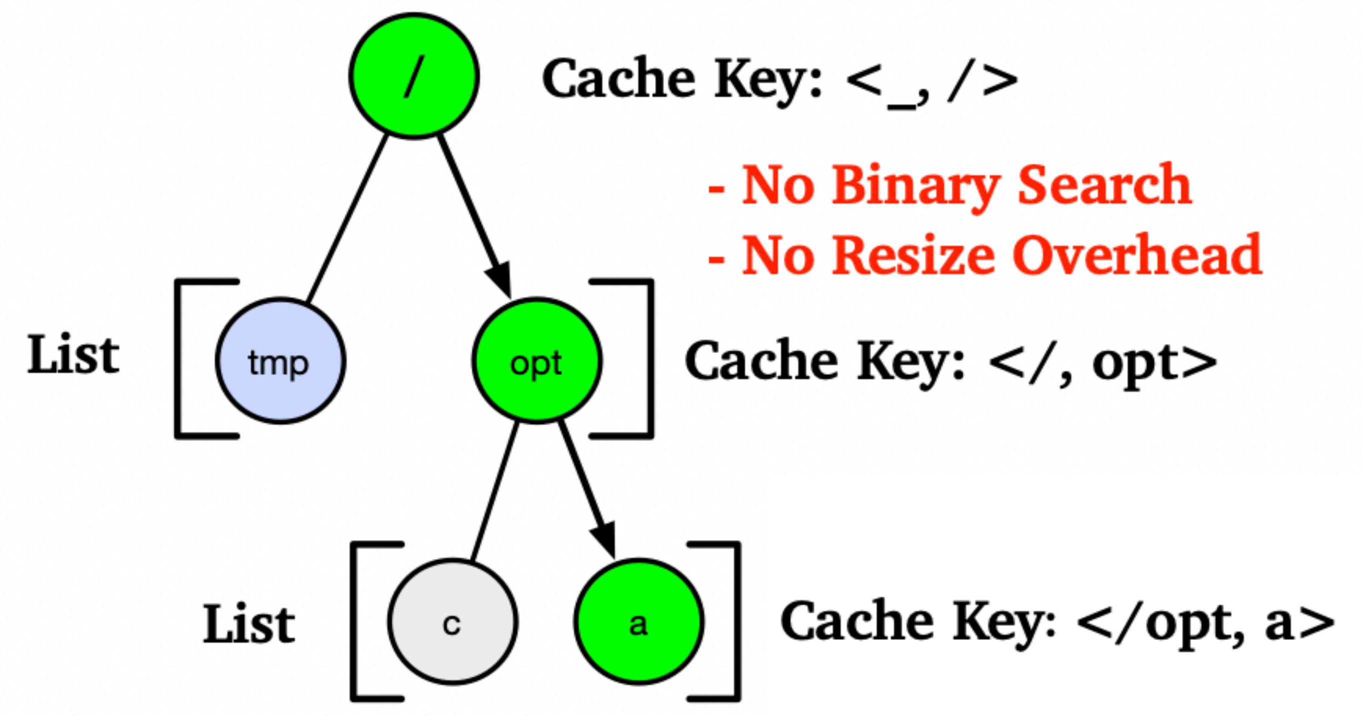
What is the difference between HDFS and FileScale?

- HDFS: in-memory pointers between directory and files
- **FileScale: path resolution → cache key**





HDFS

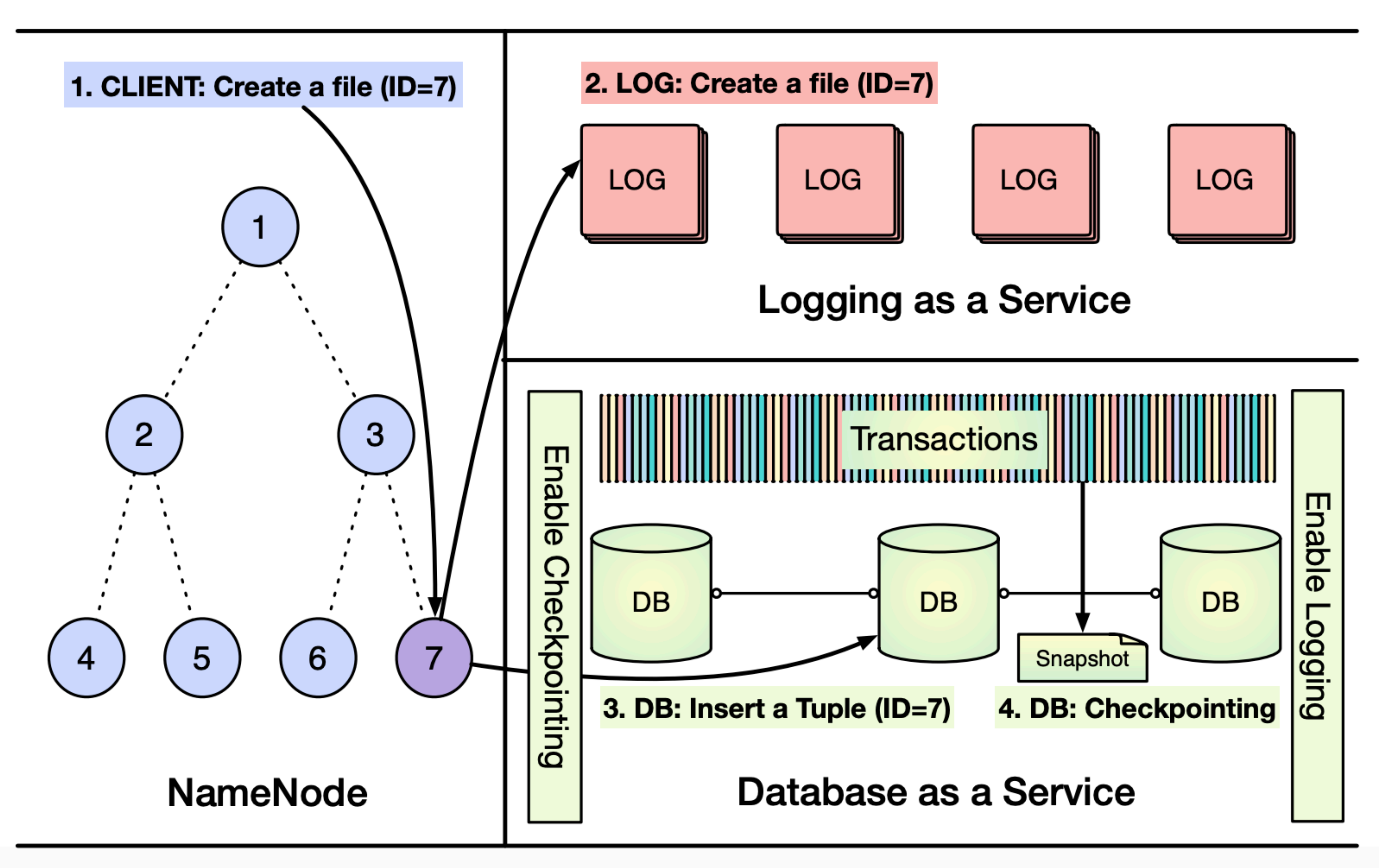


FileScale

FileScale - Caching Layer

The database log is not sufficient to guarantee system-wide durability.

We build a write-ahead logging mechanism based on an extension of HDFS's EditLog.

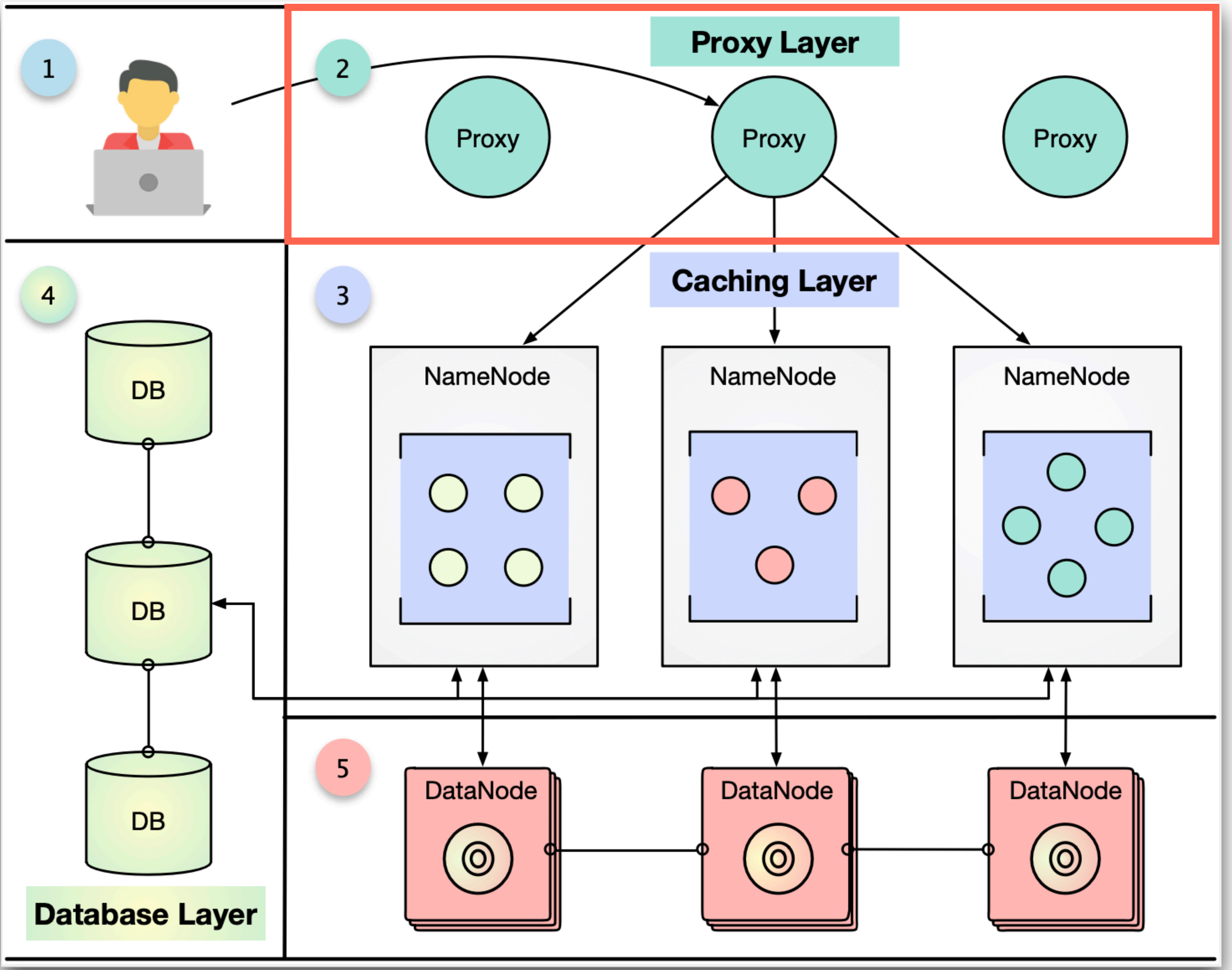


The workflow of file-create (metadata) operation

FileScale - Proxy Layer

A three-tiered architecture

- Database Layer
- Caching Layer
- **Proxy Layer**
  - Horizontally scales the name service
  - Disjoint partition of the name space
  - **Multi-partition (multi-NameNode) transactions**

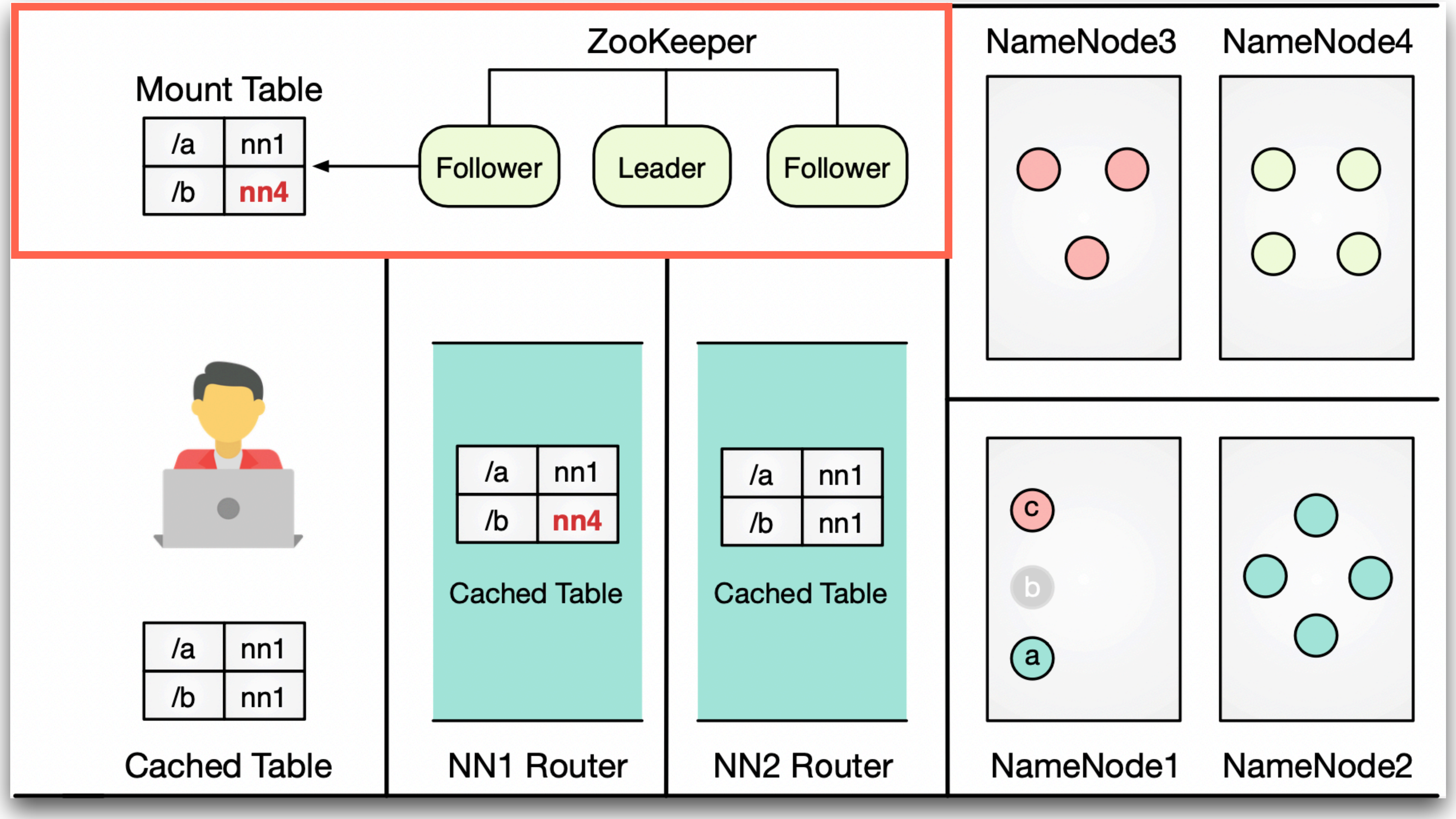


System Architecture of FileScale

FileScale - Proxy Layer

Mount Table

- Stored in Zookeeper

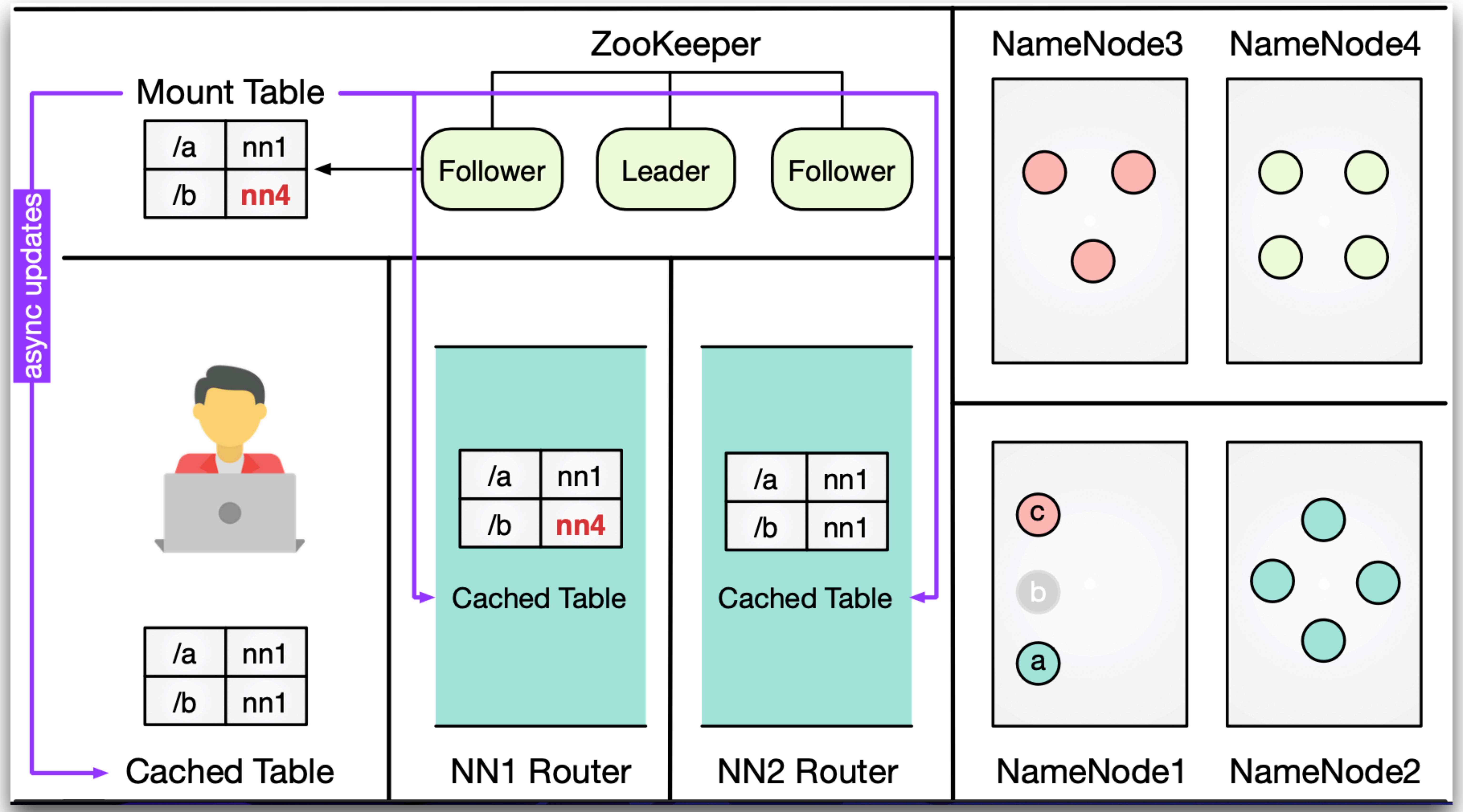


Mount Table in Proxy Layer

FileScale - Proxy Layer

Mount Table

- Stored in Zookeeper
- Cached in the routers and client-side.

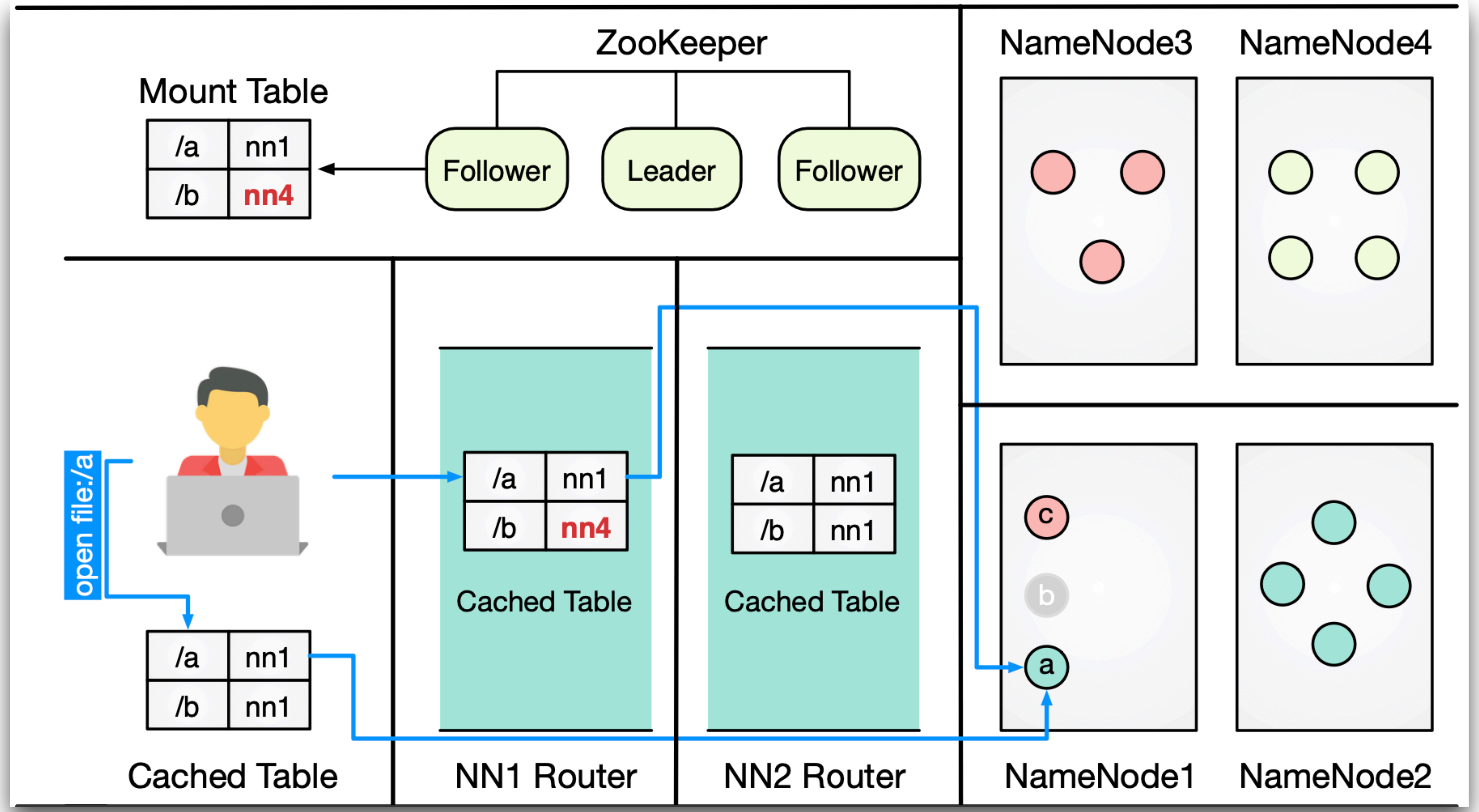


Cached Mount Table in Routers and Client-side

FileScale - Proxy Layer

Request Routing

- Proxy mode
  - A middleware layer
- Watch mode
  - Save a network hop
  - Cached in client-side

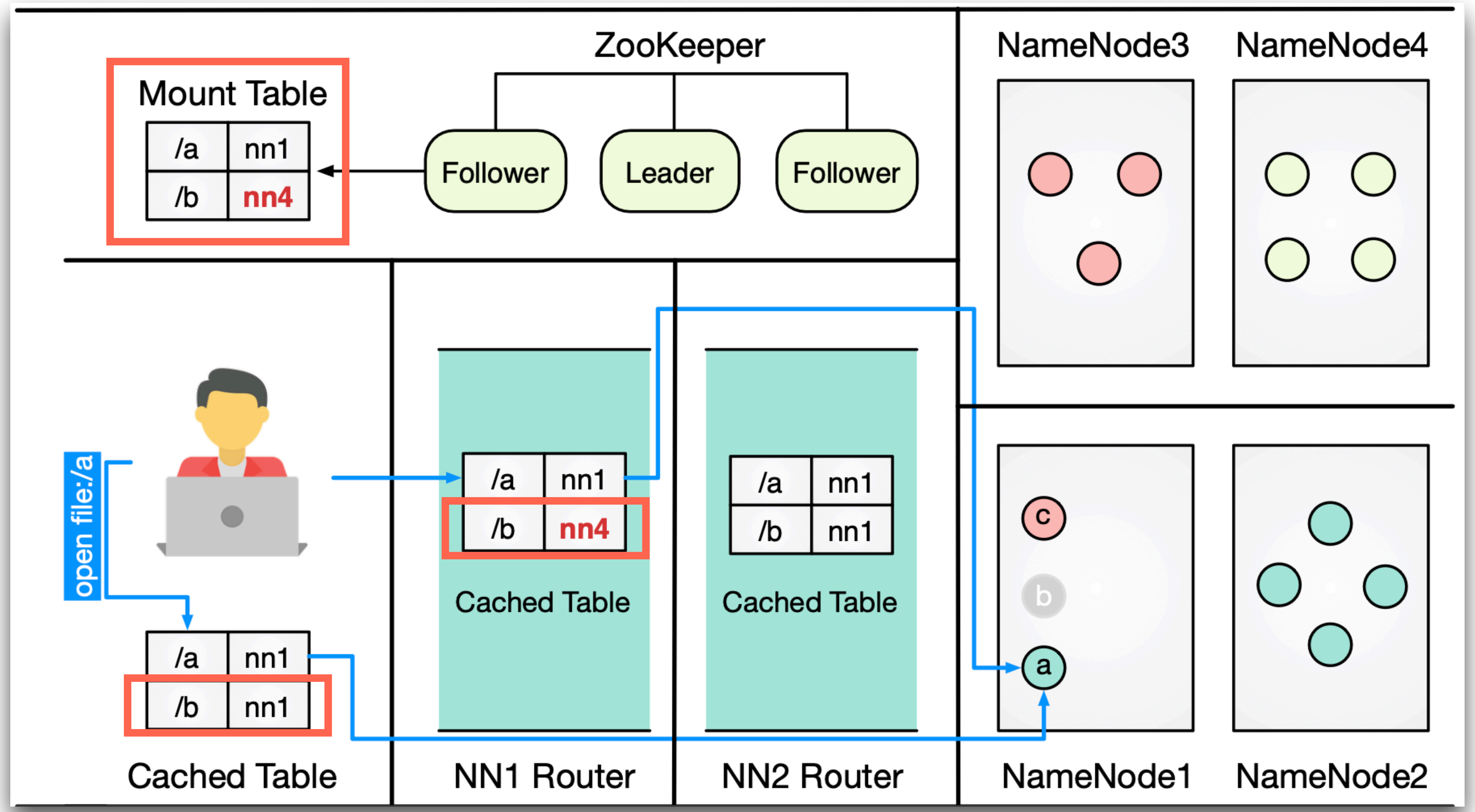


Request Routing in FileScale

FileScale - Proxy Layer

Request Routing

- Proxy mode
  - A middleware layer
- Watch mode
  - Save a network hop
  - Cached in client-side
- Preventing Stale Read ?



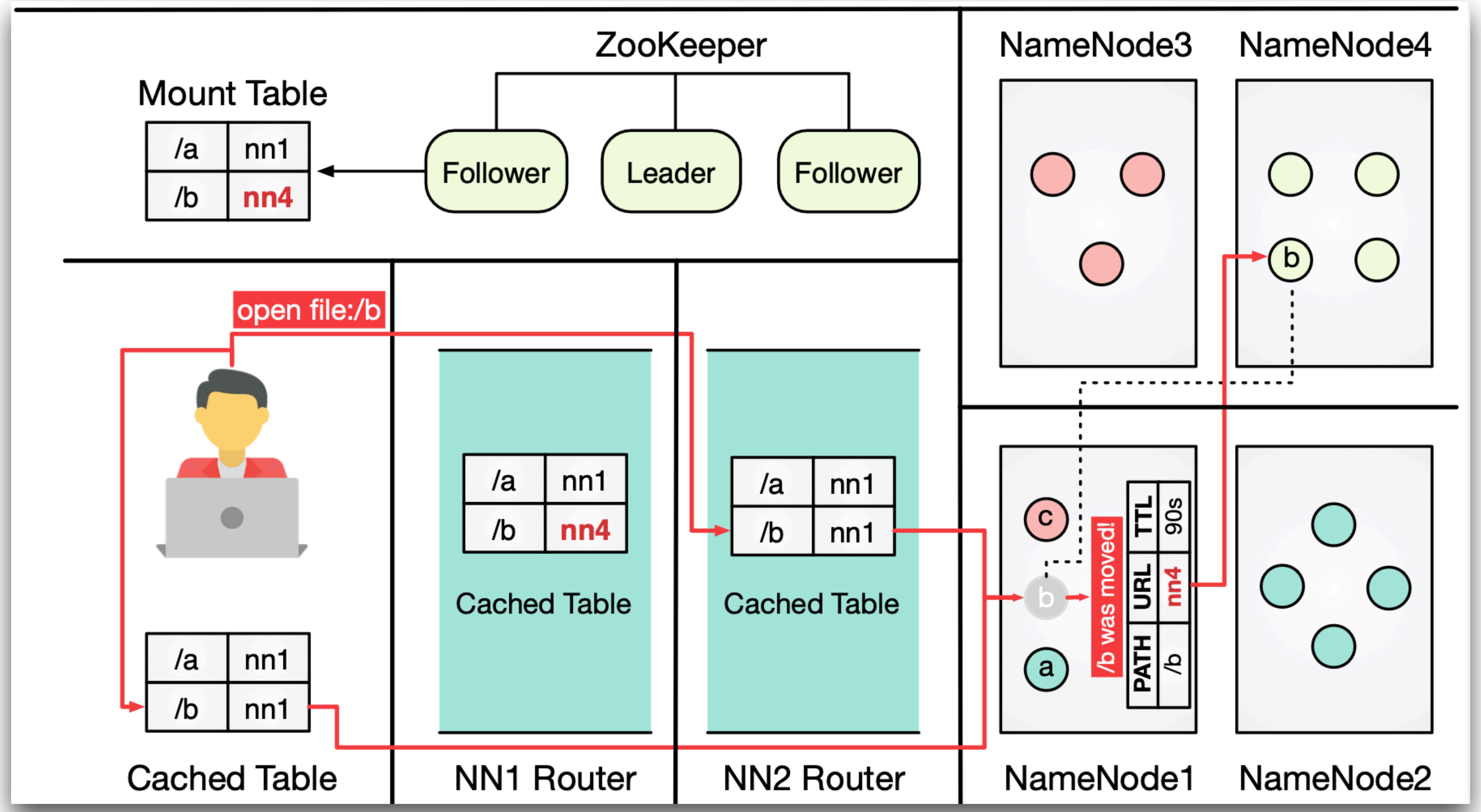
Request Routing in FileScale

FileScale - Proxy Layer

On occasion, a name space partition may be moved from one NameNode to another, and causing misrouting of requests.

Preventing Stale Read

- A recent-memory of paths in each NameNode
  - Short Time to Live (TTL) for each moved path in memory
  - Re-forward requests to the right NameNode

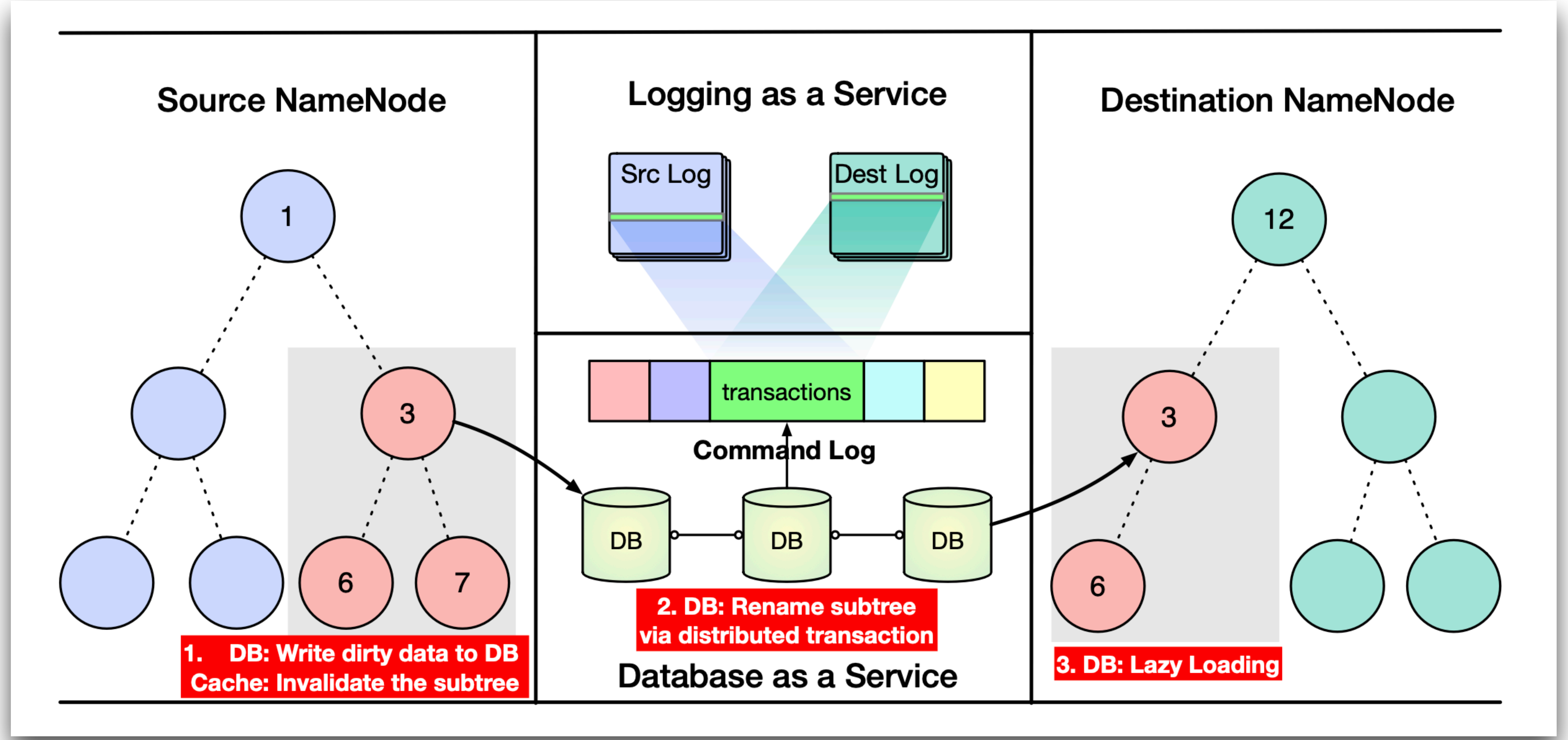


Request Routing in FileScale

FileScale: Multi-partition Requests

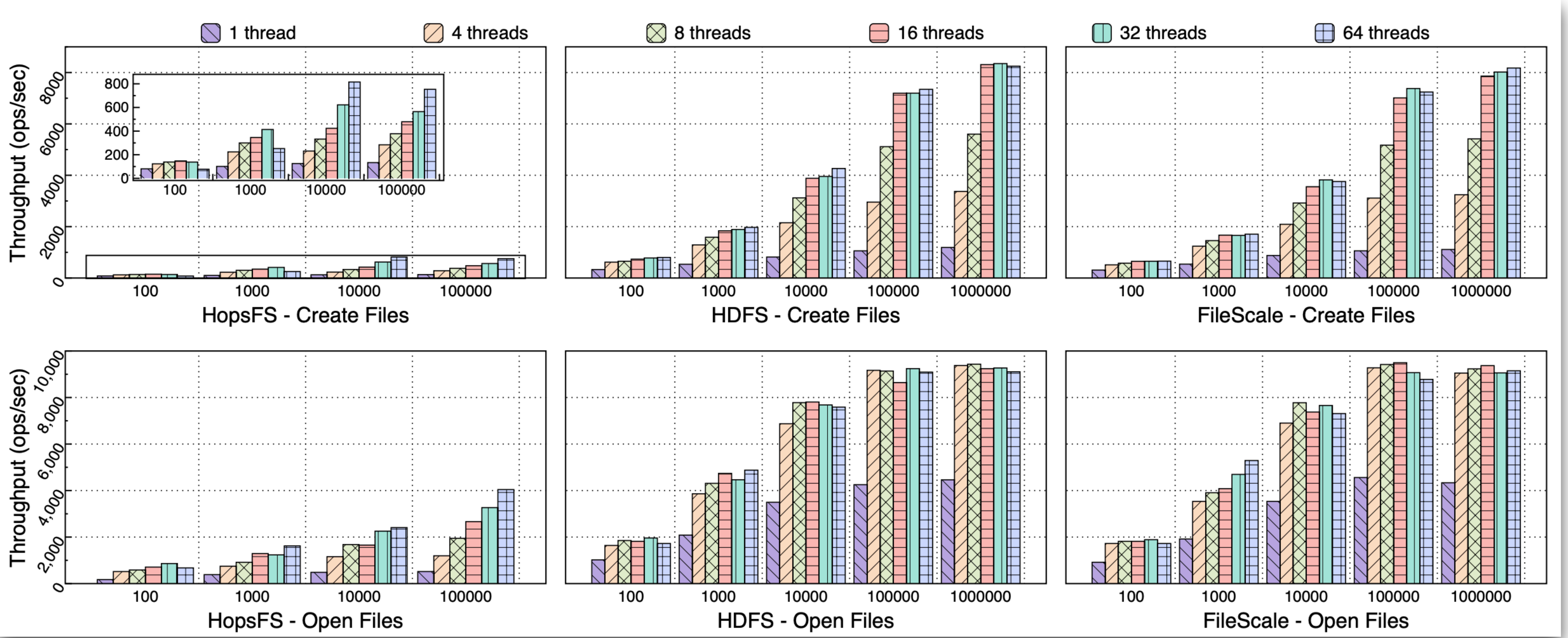
Concurrency Control

All data accessed by the transaction are removed from cache and prevented from being brought into cache while the transaction is ongoing.



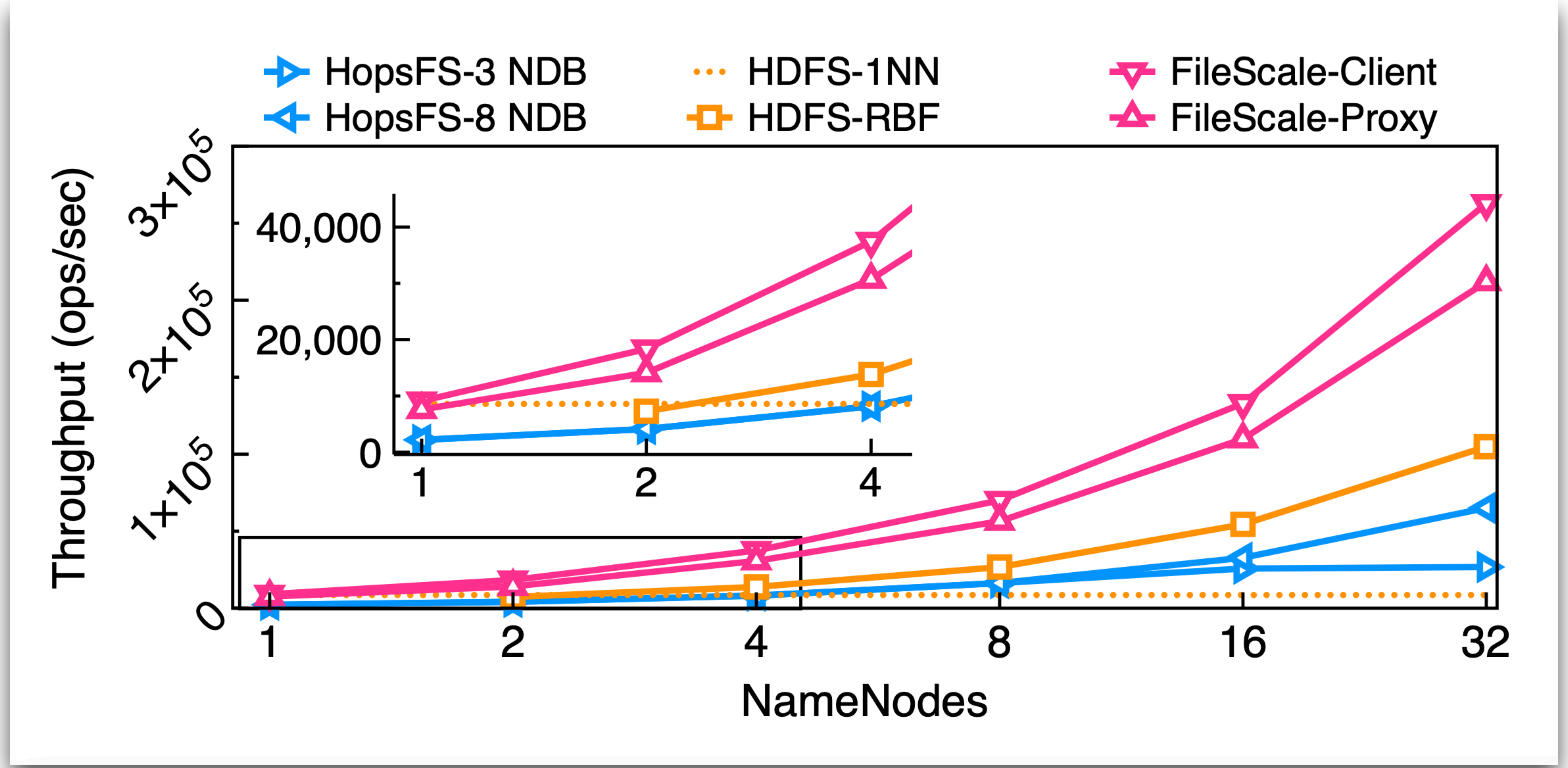
Move a folder across NameNodes

FileScale: Create and Open Operations



The throughput of basic operations including create, open on a EC2 instance — t3a.2xlarge

FileScale: Scalability Experiment



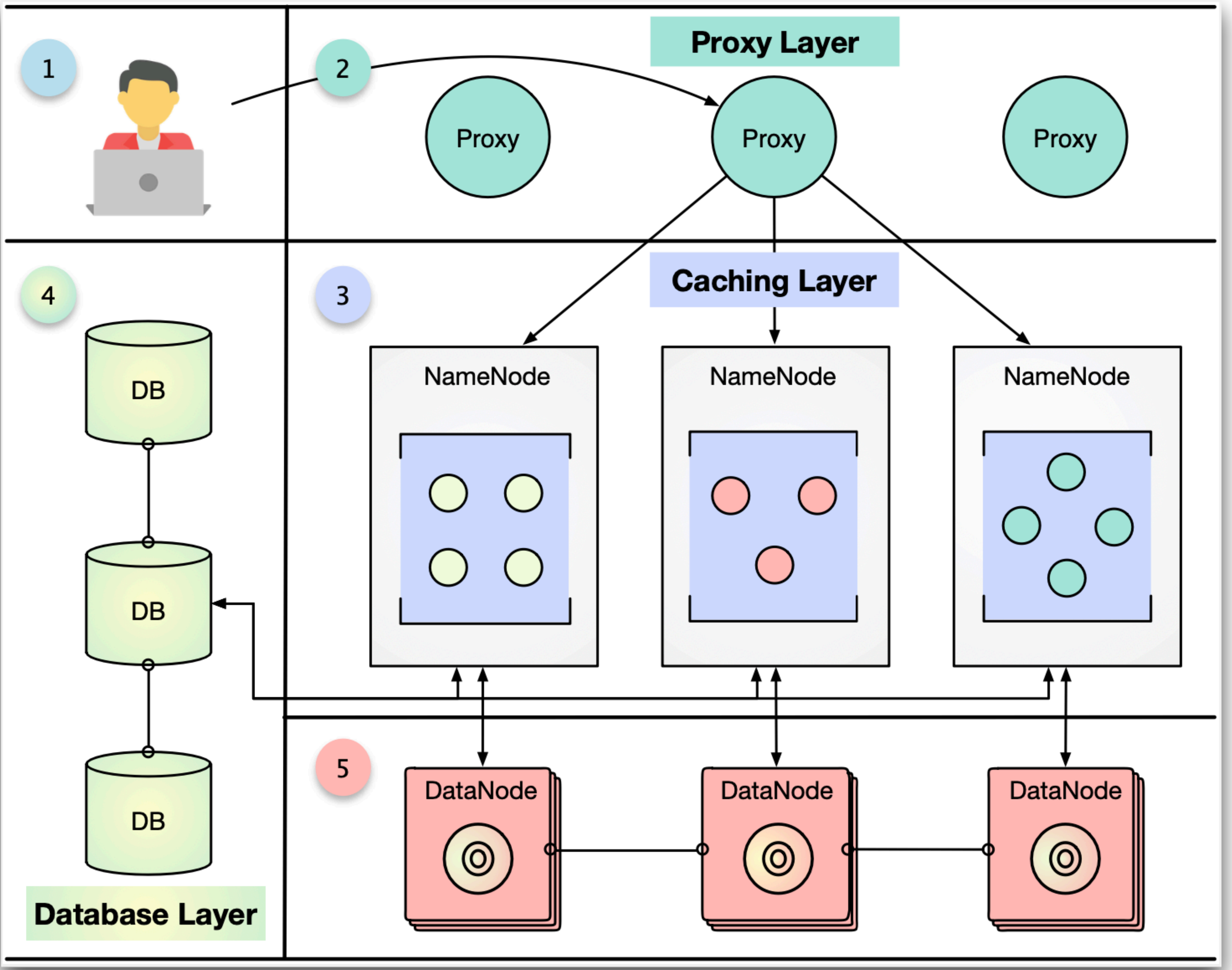
Throughput when scaling NameNodes

FileScale

A three-tiered architecture

- Database Layer
- Caching Layer
- Proxy Layer
- https://github.com/umd-dslam/FileScale
- ~40k LoC

FileScale's architecture enables elastic scaling of each layer in the architecture independently.

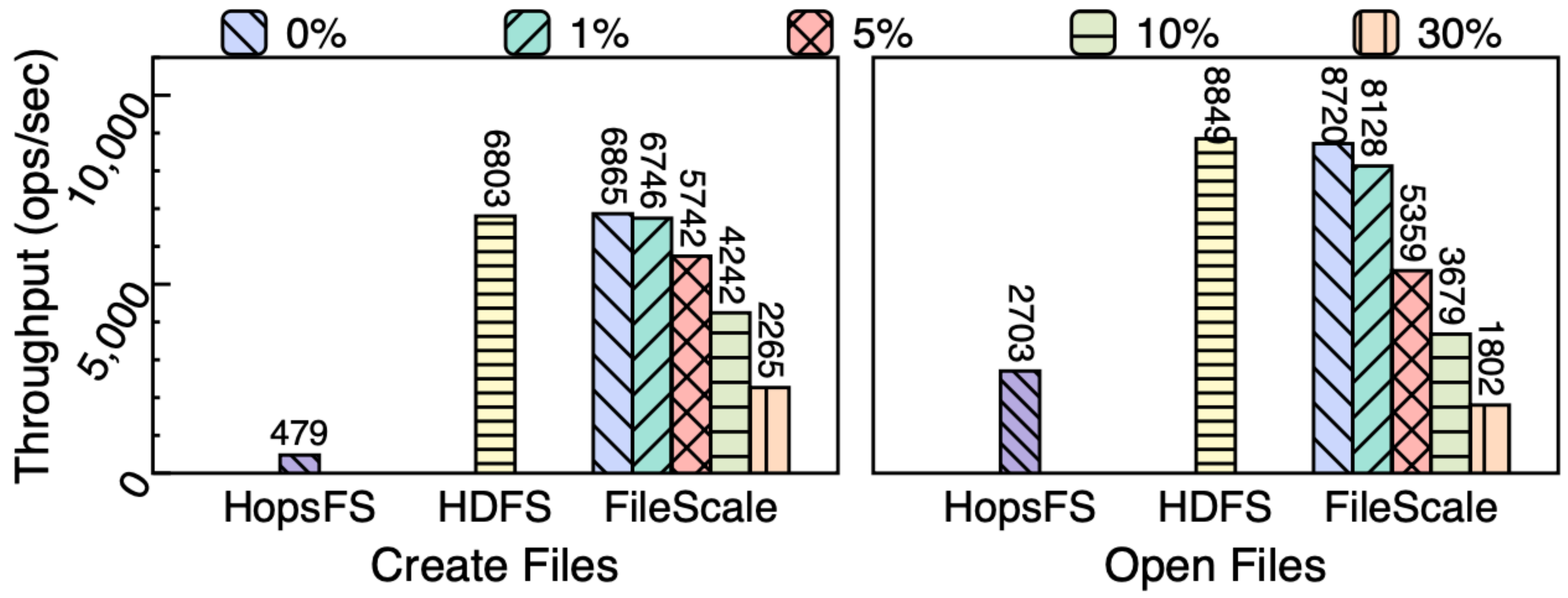


System Architecture of FileScale

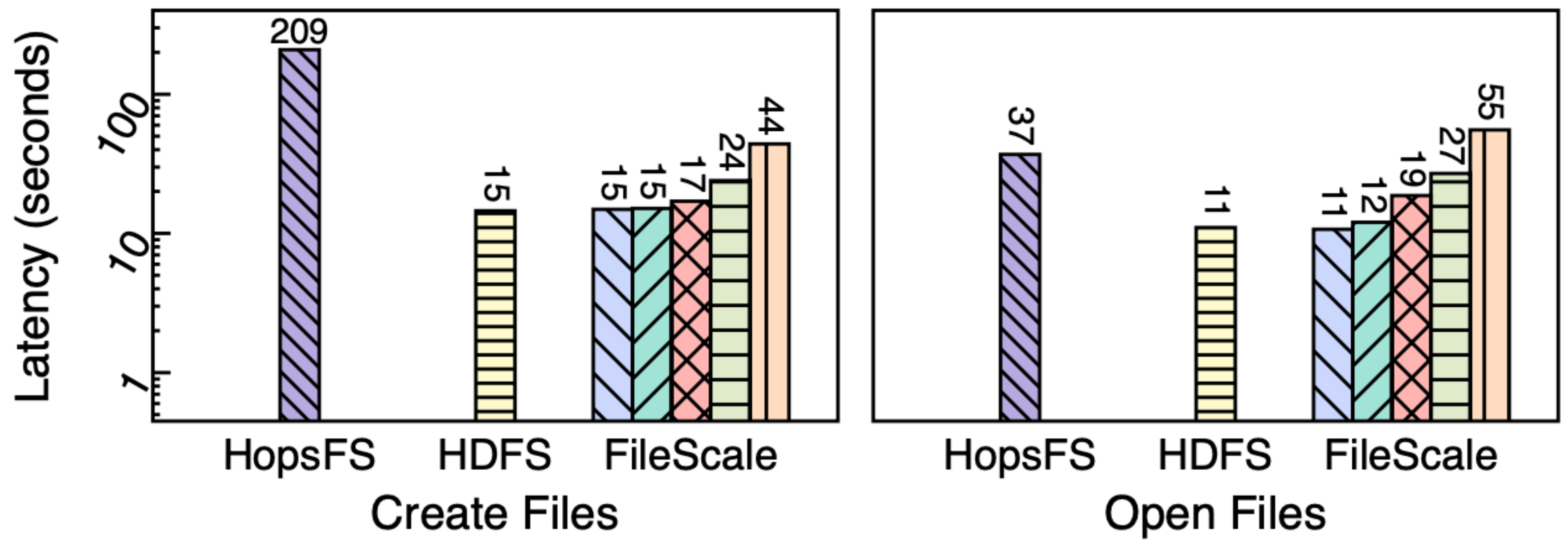
Thank You!

FileScale - Caching Layer

Cache Miss Penalty



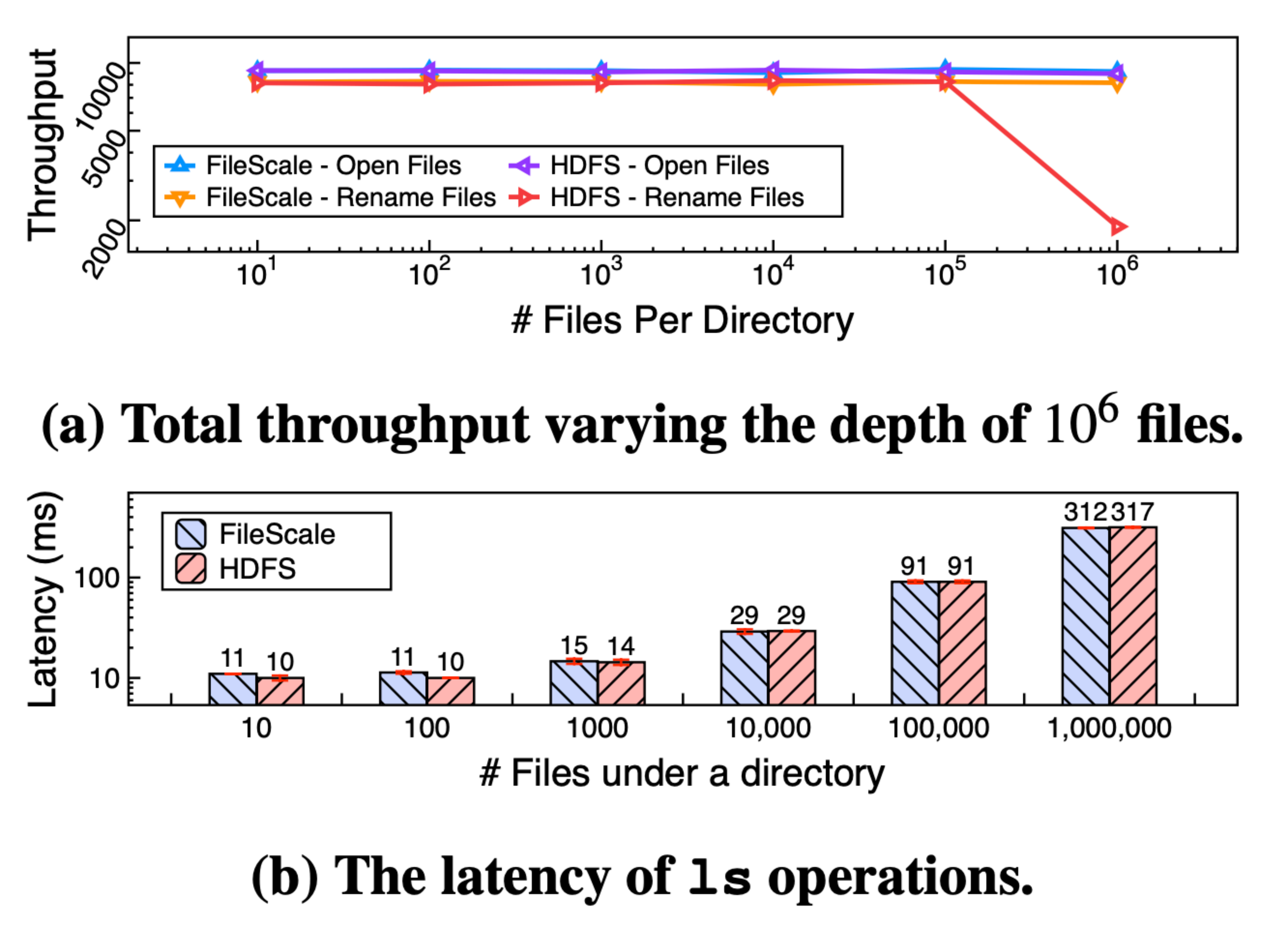
(a) The throughput of creating and opening files.



(b) The latency of creating and opening files.

FileScale - Caching Layer

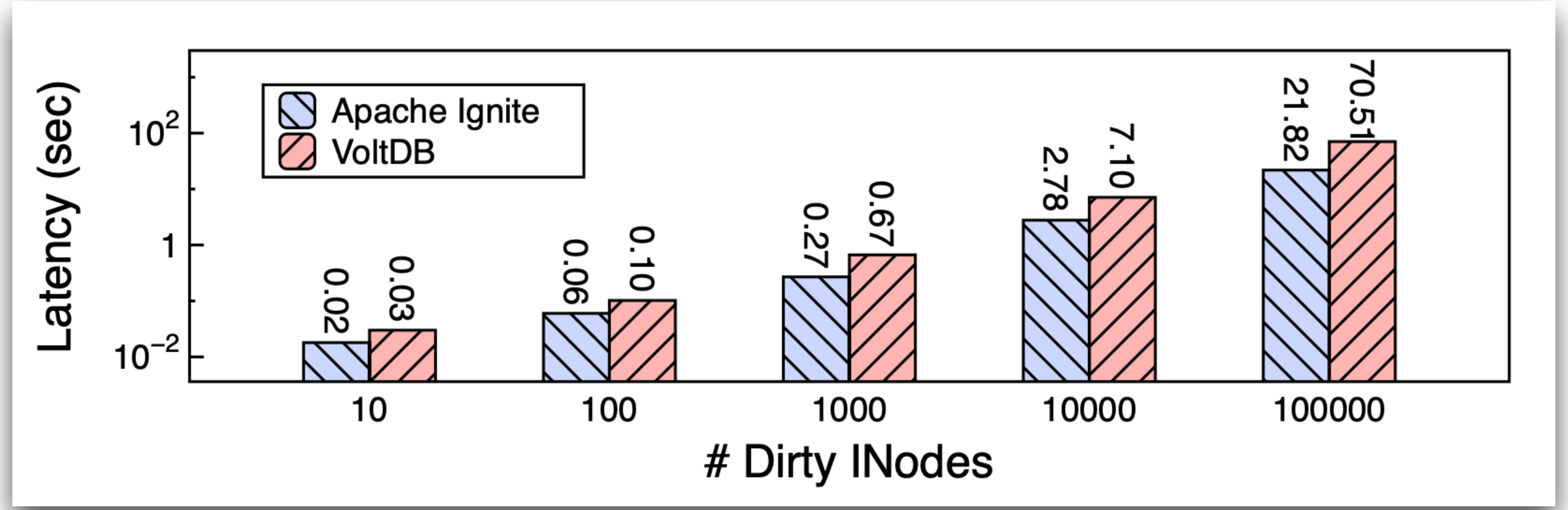
Large directory experiment



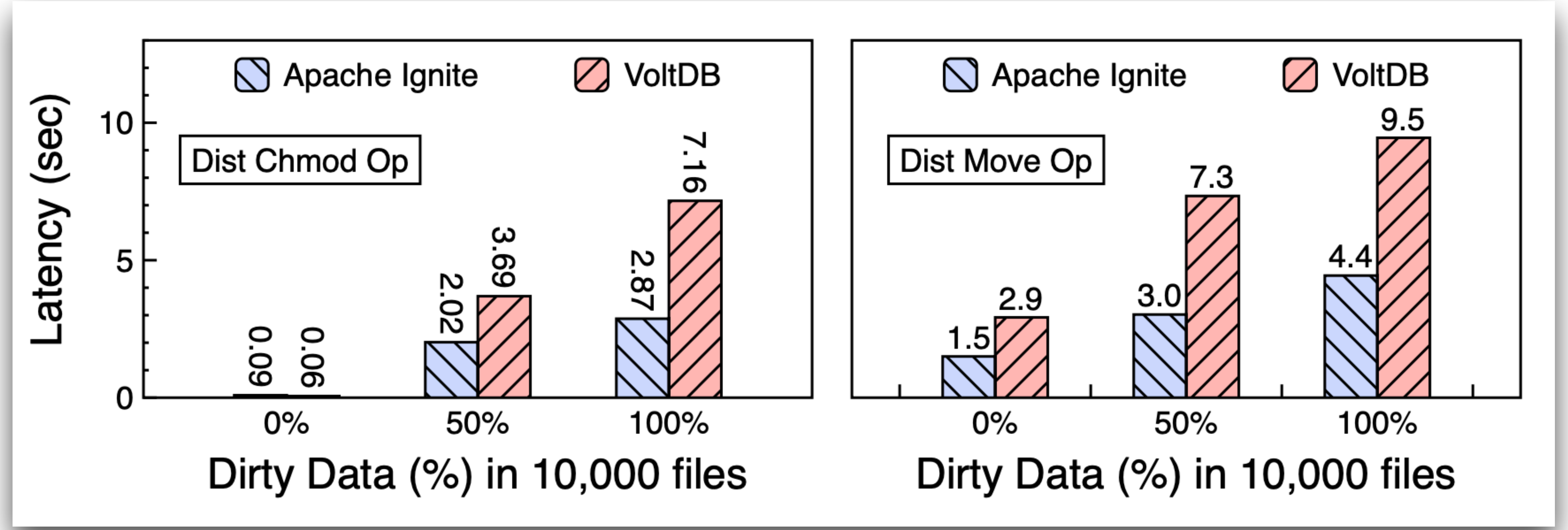
FileScale: Multi-Partition Transactions

Multi-Partition Requests

- Cache Flushing
- Distributed Transactions



Dirty data flush penalty



Distributed chmod and move operations

System Comparison

System	Metadata	Multi-Partition Operations	Single-node In-memory Performance
DeltaFS [55]	LevelDB	No	Yes
TableFS [43]	LevelDB	No	Yes
IndexFS [44]	LevelDB	No	Yes
ShardFS [52]	LevelDB	No	Yes
GiraffaFS [48]	HBase	No	No
Colossus [29]	BigTable	No	No
Tectonic [40]	ZippyDB [36]	No	No
ADLS [42]	Hekaton [25]	Yes	No
HopsFS [39]	MySQL NDB	Yes	No
CalvinFS [50]	Calvin [51]	Yes	No
ViewFS [11]	In-Memory	No	Yes
Giga+ [41]	LevelDB	Yes	No
HDFS RBF [8]	In-Memory	No	Yes
FileScale	Ignite, VoltDB	Yes	Yes

Table 2: Comparison of related scalable file systems.

System Comparison

System	Metadata	Multi-Partition Operations	Single-node In-memory Performance
DeltaFS [55]	LevelDB	No	Yes
TableFS [43]	LevelDB	No	Yes
IndexFS [44]	LevelDB	No	Yes
ShardFS [52]	LevelDB	No	Yes
GiraffaFS [48]	HBase	No	No
Colossus [29]	BigTable	No	No
Tectonic [40]	ZippyDB [36]	No	No
ADLS [42]	Hekaton [25]	Yes	No
HopsFS [39]	MySQL NDB	Yes	No
CalvinFS [50]	Calvin [51]	Yes	No
ViewFS [11]	In-Memory	No	Yes
Giga+ [41]	LevelDB	Yes	No
HDFS RBF [8]	In-Memory	No	Yes
FileScale	Ignite, VoltDB	Yes	Yes

Table 2: Comparison of related scalable file systems.

"By treating metadata management similar to data management, we built a system that can store very rich metadata and scale to very large tables, while also providing performant access to it from the query engine."

Big Metadata: When Metadata is Big Data

Pavan Edara and Mosha Pasumansky, Google BigQuery, PVLDB 2021

"Storing file metadata in BigTable allowed Colossus to scale up by over 100x over the largest GFS clusters."

Colossus under the hood: a peek into Google's scalable storage system

Dean Hildebrand and Denis Serenyi, Google Cloud Blog, April 19, 2021

FileScale - Database Layer

Primary key (parent name, inode name) → full path

Compared with using id as the primary key, what are the advantages?

The semantics of the hierarchical relationships between the files are not included in the ID.