SAMBA eXPERIENCE

Azure Files: “mount” the Cloud

2 June 2022

David Goebel
Microsoft

Azure Files (AF)

Talk Topics:

0 Review of Azure Files design.

1 Schema impacting new features, using the example of Large File Shares (LFS).

2 How new features are enabled in a service with zero downtime.

SMB Protocol History

- <= Srv03 was V1.3 and carried **lots** of baggage.
- Vista: SMB 2.02 was a vast simplification to reduce chattiness and map SMB commands to NT Irps and compound commands. Durable handles allowed handles to be reconnect after a network hiccup.
- Win7: SMB 2.1 introduced resilient handles and “leases” which supersede the older oplock scheme taken from 1.3
- Win8: SMB 3.0 added encryption, leases on directory handles, multichannel & RDMA (SMB Direct), and Persistent handles to support CA (Continuously Available) shares on clustered servers.
- Win8.1: SMB 3.0.2 added cluster enhancements.
- Win10: SMB 3.1.1 adds negotiated encryption algorithm, secure negotiate and other security features.

Fundamental Concepts

- AF is not the Windows SMB server (srv2.sys) running on Azure nodes.
- AF is a completely new SMB server implementation which uses Azure Tables and Blobs as the backing store.
- AF leverages the highly available and distributed architecture of Tables and Blobs to imbue those same qualities to the file share.

Current AF Status

- SMB 3.1.1 with encryption, persistent handles, and Multichannel.
- Three tiers: Standard, Standard Large File Share (LFS), and Premium Files (fully SSD backed storage).
- Per-file ACLs with Kerberos authentication & AD integration. NTLMv2 with AccountKey still an option.
- Share snapshots.
- Removed limitation on Private Use Area characters (0xE000 to 0xF8FF).
- Some NTFS features not supported.

Current AF Limits

Metric Tier	Max Share Size	Max Share IOPS*	Max Share Throughput	Max File IOPS	Max File Throughput
Standard	5 TiB	1000	60 MiB/sec	1000	60 MiB/sec
LFS	100 TiB	20,000	300 MiB/sec	1000	60 MiB/sec
Premium	100 TiB	100,000	10,340 MiB/sec	8000	300 MiB/sec

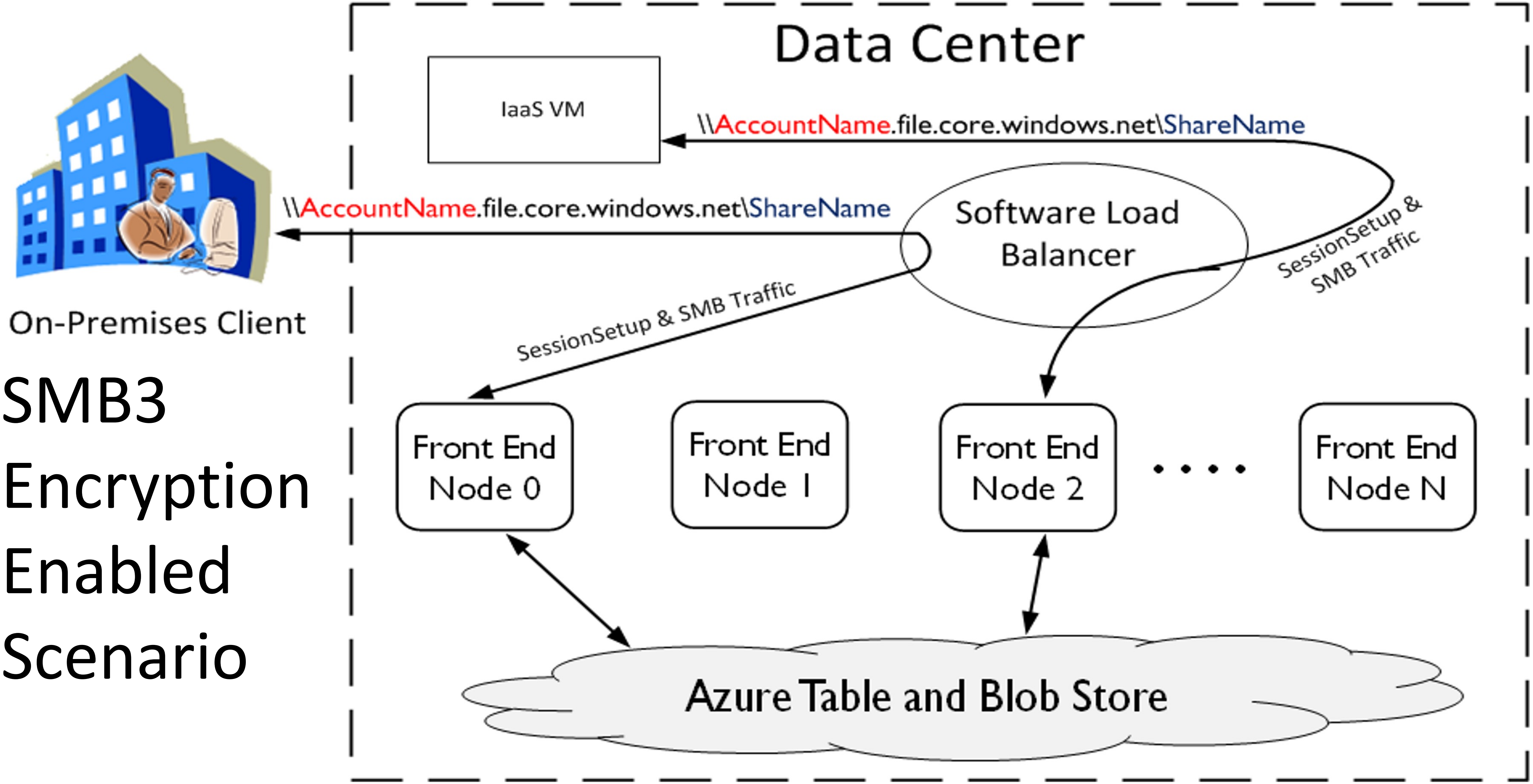
*AF shares support bursting, so IOPS computations are not completely straightforward. For details see: https://docs.microsoft.com/en-us/azure/storage/files/understanding-billing#provisioning-method

Linux Client Support Notes

- Directory leases added in 4.18 kernel
- Encryption added in 4.11
- Persistent Handles mount option added in 4.1 kernel (and important fixes added in 4.11)

- RHEL7.5 supports Persistent Handles and encryption but not Directory Leases
- RHEL8.1 and later (RHEL8.6 and RHEL9 are the current LTS versions) supports all 3.
- Debian 10 and Debian 11 support all 3
- Ubuntu 20.04 (5.4 kernel and later), Ubuntu 21 and Ubuntu 22.04 (most current LTS) all support all 3
- SLES15SP2 and later (SuSE enterprise server 15, service pack 2 or 3) support all 3
- OpenSuSE Leap version 15.2 and later support all 3.

SMB3 Encryption Enabled Scenario



Data Center

IaaS VM

\\AccountName.file.core.windows.net\ShareName

On-Premises Client

\\AccountName.file.core.windows.net\ShareName

Software Load Balancer

SessionSetup & SMB Traffic

SessionSetup & SMB Traffic

Front End Node 0

Front End Node I

Front End Node 2

....

Front End Node N

Azure Table and Blob Store

Note: Port 445 outbound must be unblocked.

Scenarios Enabled By AF

- Existing file I/O API (Win32, CRT, etc.) based applications, i.e. most business applications written over the last 40 years, should "just work"®. More on this in "Lessons Learned" later.
- A business can stage existing workloads seamlessly into the cloud without modification to mission critical applications.
- Some minor caveats that will become more minor over time.

What about REST?

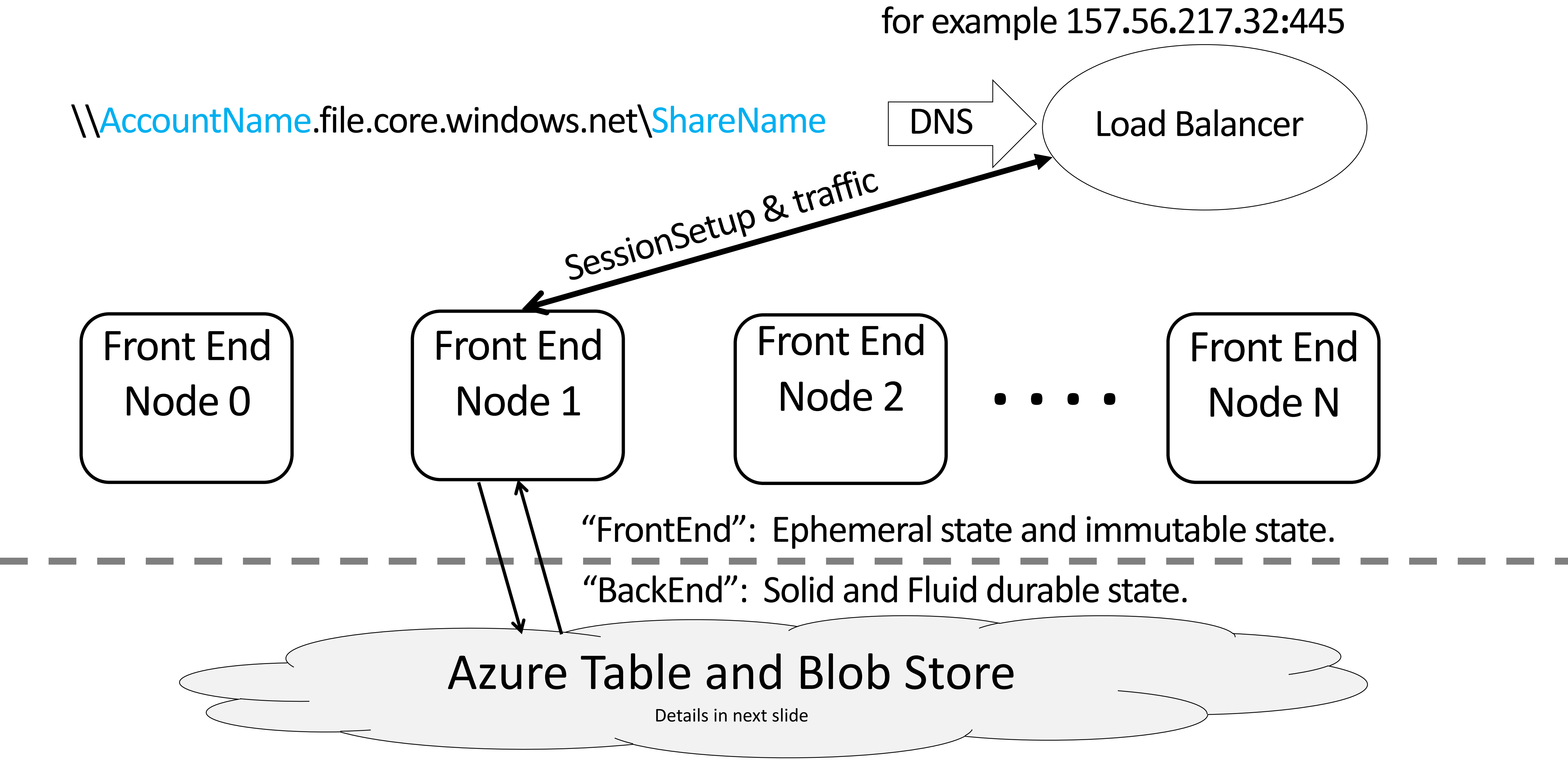
If you’re a true believer in the benefits of statelessness, SMB and REST access the same data in the same namespace so a gradual application transition without disruption is possible.

- Container operations: Create, List, Delete, Get/Set properties, Get/Set metadata, Get/CreatePermission, Lease, Snapshot, and Restore

- File/Directory Operations: Create, Delete, Get/Set properties, Get/Set metadata, Rename, ListHandles, and ForceCloseHandles

- File only operations: ListRanges, GetFile, PutRange, CopyFile, and LeaseFile

- Directory only operations: List



for example 157.56.217.32:445

\\AccountName.file.core.windows.net\ShareName

DNS

Load Balancer

SessionSetup & traffic

Front End Node 0

Front End Node 1

Front End Node 2

. . . .

Front End Node N

“FrontEnd”: Ephemeral state and immutable state.

“BackEnd”: Solid and Fluid durable state.

Azure Table and Blob Store

Details in next slide

SMB is a stateful protocol,

but not all states require expensive distributed transactional semantics

- Some aspects of a file's state are immutable, such as FileId and whether it's a file or a directory.
- Some state is transient, such as open counts, and can be optimized if loss of this state is acceptable in a disaster.
- Some state is also maintained by the client, like CreateGuid, drastically reducing the cost of tracking clients.
- State associated with connection mechanics is ephemeral.

Azure Table and Blob Store

- AF uses the underlying Azure Tables infrastructure to store metadata associated with files/dirs, open handles to them and other state like byte range locks, leases, etc.
- An Azure Table is a simple NoSQL collection of rows with a common column schema and sorted / searchable by a subset of ordered ‘key’ columns.
- Two types of keys: Partition (coarse) or Row (fine).

AF Tables

- Azure Tables allows associating a set of tables as a group.
- Within a partition, a single request can atomically operate on multiple tables in this group.
- An AF share's metadata is stored as a group of tables, the most notable of which are:

File	A table of all files and directories. It is a hybrid type, keyed by either ParentId & FileName, or FileId (64bit like NTFS).
Page	The allocated file ranges and their backing page blobs.
Handle	All open handles to files and directories.
Lease	All currently active SMB leases.
Change Notify	Registered change notifies.
Byte Range Locks	All currently active byte range locks

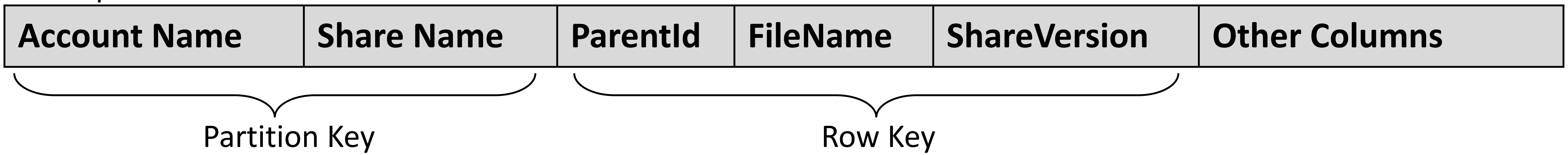
Leveraging Azure Table Infrastructure

- Internal transaction APIs allow multiple rows from multiple tables to be modified with ACID semantics in a single transaction, as long as they have the same partition key.
- By default, a standard share is wholly contained within a partition.
- With Large File Shares (LFS), a file share may now span many partitions.
- Premium Files are always LFS.

AF File Table Row Keys

- There are two types of file rows: Namespace and Data
  (technically a single merged row type, but showing them separate here for clarity)
- There are two types or keys: Partition and Row

Namespace Rows

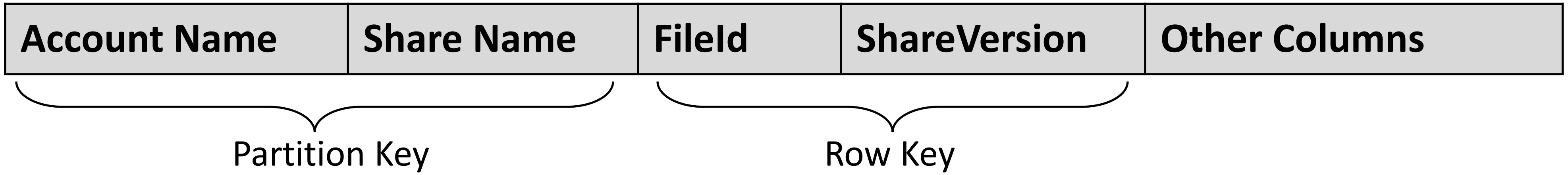


Account Name	Share Name	ParentId	FileName	ShareVersion	Other Columns

Partition Key

Row Key

Data Rows: Pre-LFS



Account Name	Share Name	FileId	ShareVersion	Other Columns

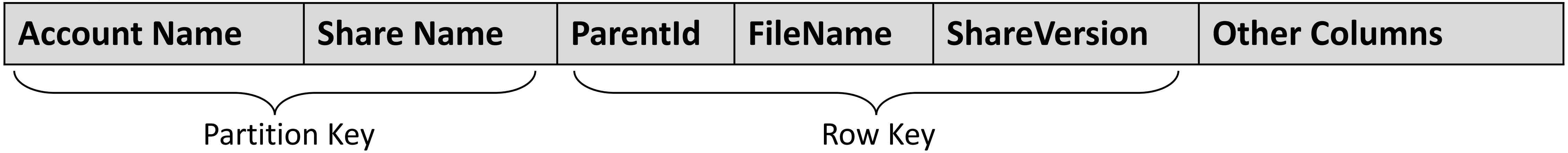
Partition Key

Row Key

AF File Table Row Keys

- There are two types of file rows: Namespace and Data
  (technically a single merged row type, but showing them separate here for clarity)
- There are two types or keys: Partition and Row

Namespace Rows

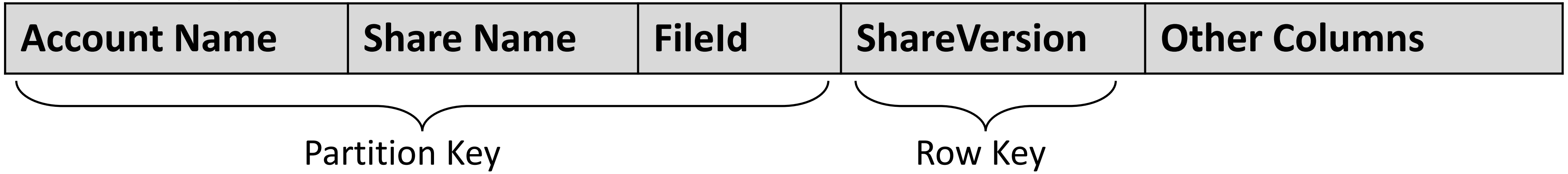


Account Name | **Share Name** | **ParentId** | **FileName** | **ShareVersion** | **Other Columns**

Partition Key: Account Name, Share Name

Row Key: ParentId, FileName, ShareVersion

Data Rows: Large File Share Version



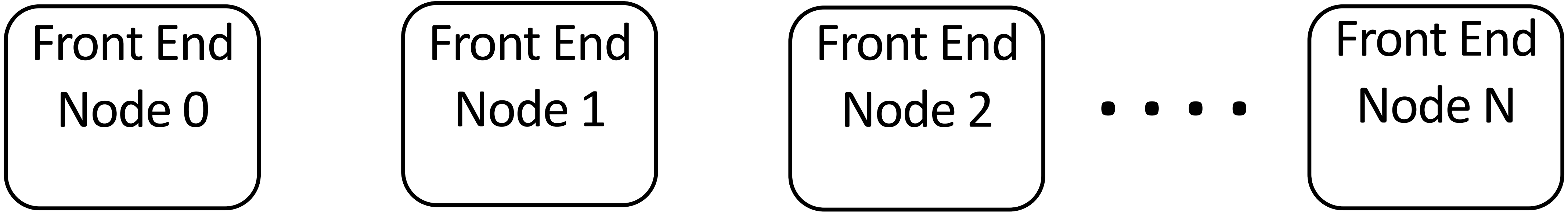
Account Name | **Share Name** | **FileId** | **ShareVersion** | **Other Columns**

Partition Key: Account Name, Share Name, FileId

Row Key: ShareVersion

Mapping AF to hardware

FrontEnd nodes receive connections from clients. Any FE node can service any share.



Front End Node 0

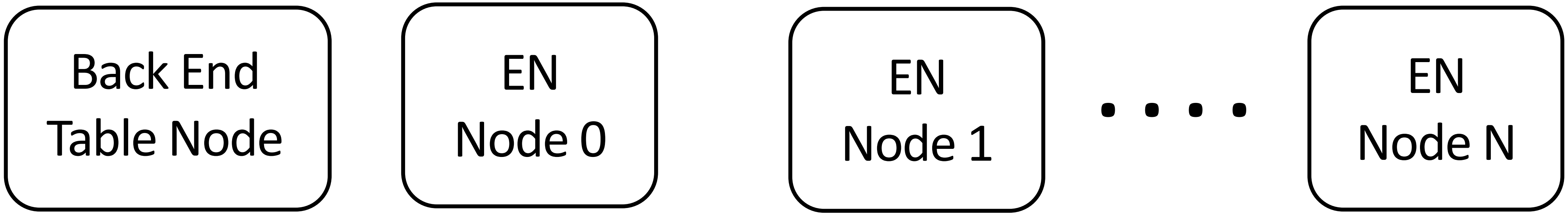
Front End Node 1

Front End Node 2

. . . .

Front End Node N

Pre-LFS a single share/container was within a single partition which is at any time “owned” by a single BE Table Node. A TableMaster service manages moving partition ownership in the case of BE node failure or for load balancing. Page blobs are managed by EN nodes.



Back End Table Node

EN Node 0

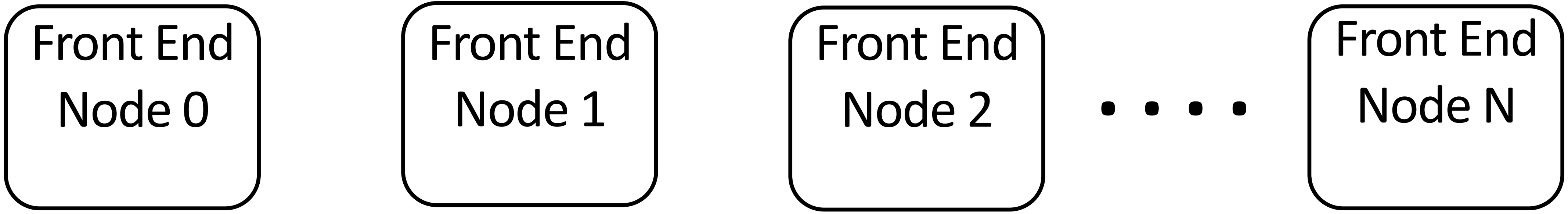
EN Node 1

. . . .

EN Node N

Mapping AF to hardware

FrontEnd nodes receive connections from clients. Any FE node can service any share.



Front End Node 0

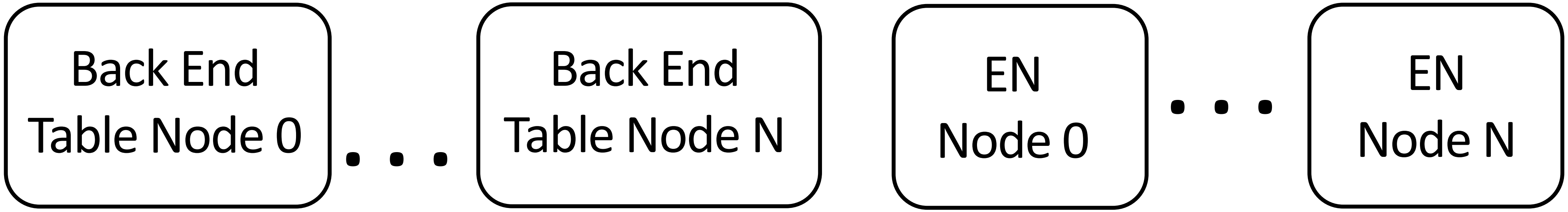
Front End Node 1

Front End Node 2

. . . .

Front End Node N

With LFS a single share/container is partitioned (by FileId) with those partitions “owned” by a collection of BE Table Nodes. The TableMaster splits/merges partitions to maintain uniform load. Page blobs are managed by EN nodes (hasn’t changed).



Back End Table Node 0

. . .

Back End Table Node N

EN Node 0

. . .

EN Node N

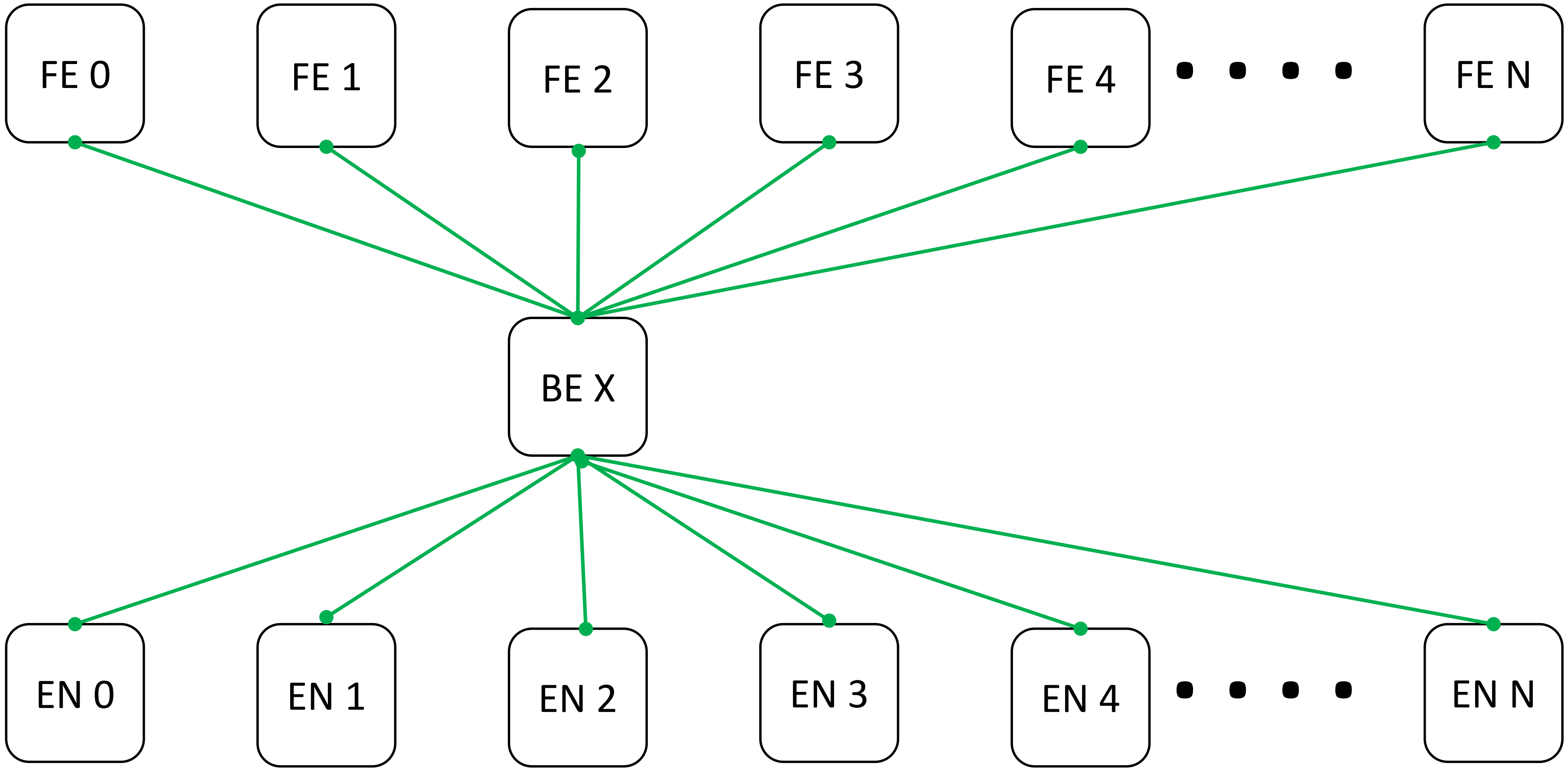
Pre-LFS State / Data Flow Topology on a Single Share

FE = Front End Node
(client connection)

BE = Back End Node
(manages metadata)

EN = Extent Node
(stores actual file data)

Metadata* & File Write Data



*Write Data Only to EN

Pre-LFS State / Data Flow Topology on a Single Share

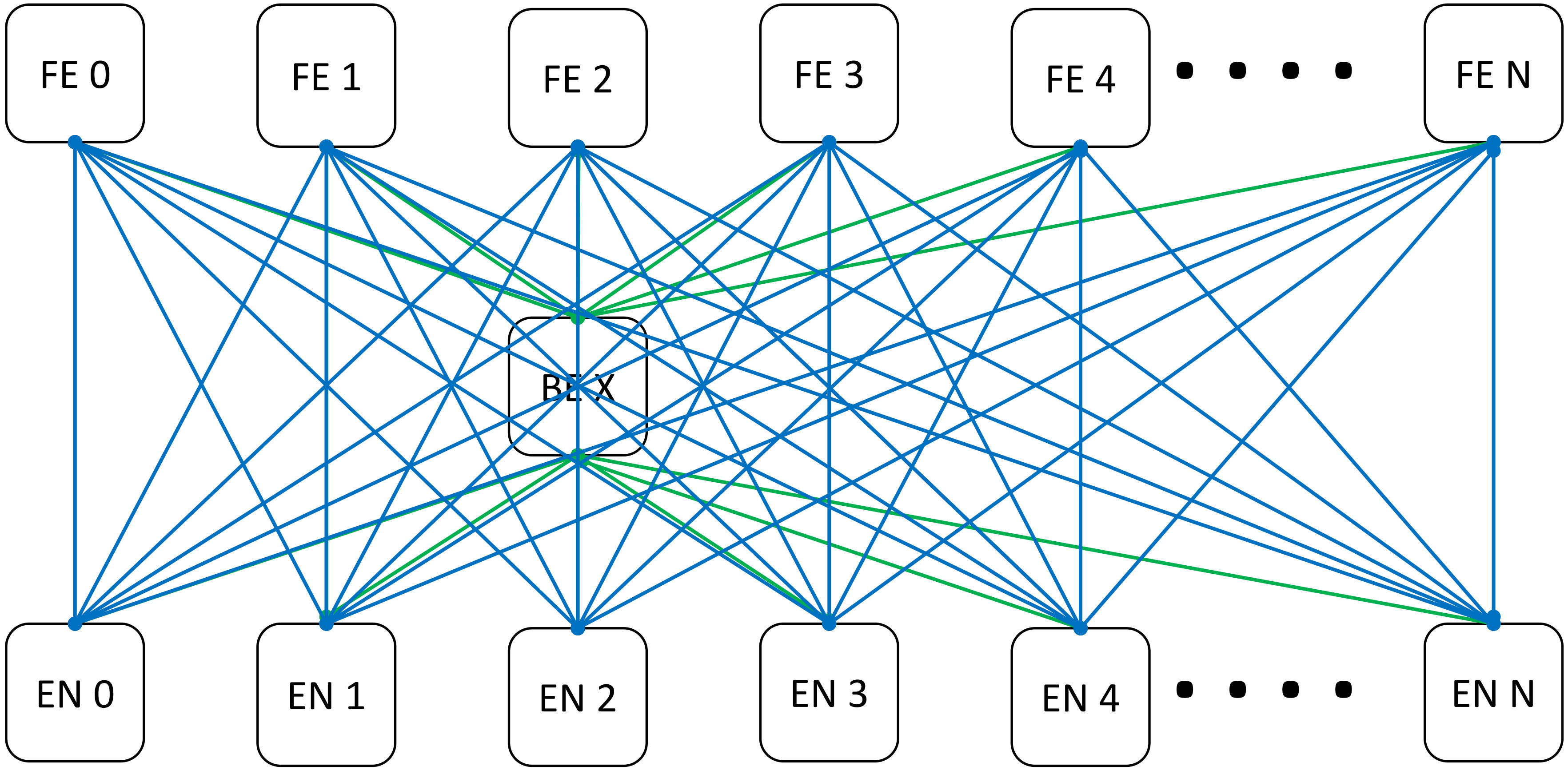
FE = Front End Node
(client connection)

BE = Back End Node
(manages metadata)

EN = Extent Node
(stores actual file data)

Metadata* & File Write Data

File Read Data



*Write Data Only to EN

LFS State / Data Flow Topology on a Single Share

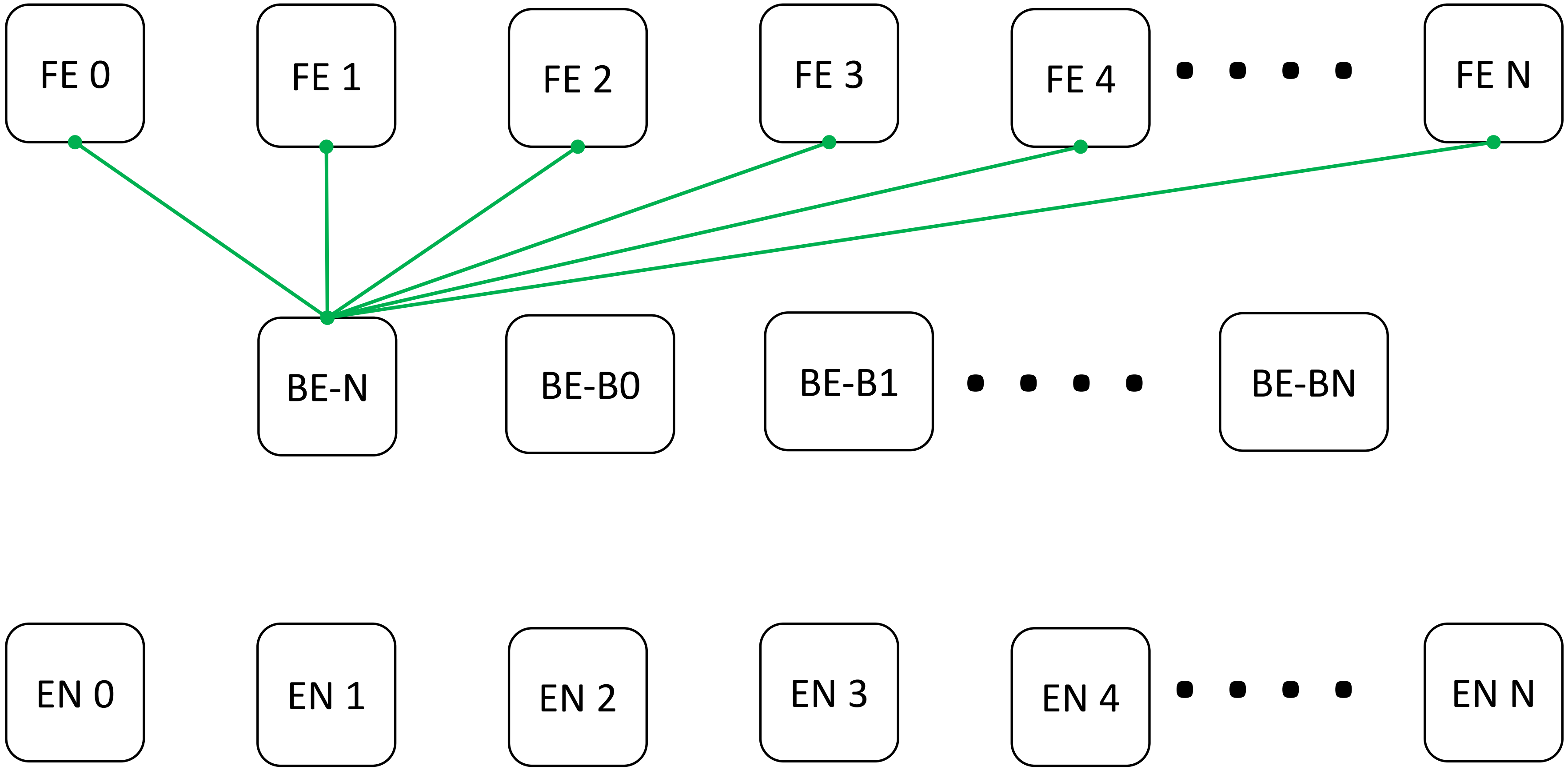
FE = Front End Node
(client connection)

BE-N = Back End Namespace Node
(namespace metadata)

BE-B = Back End Blob Node
(file metadata)

EN = Extent Node
(stores actual file data)

Namespace Metadata



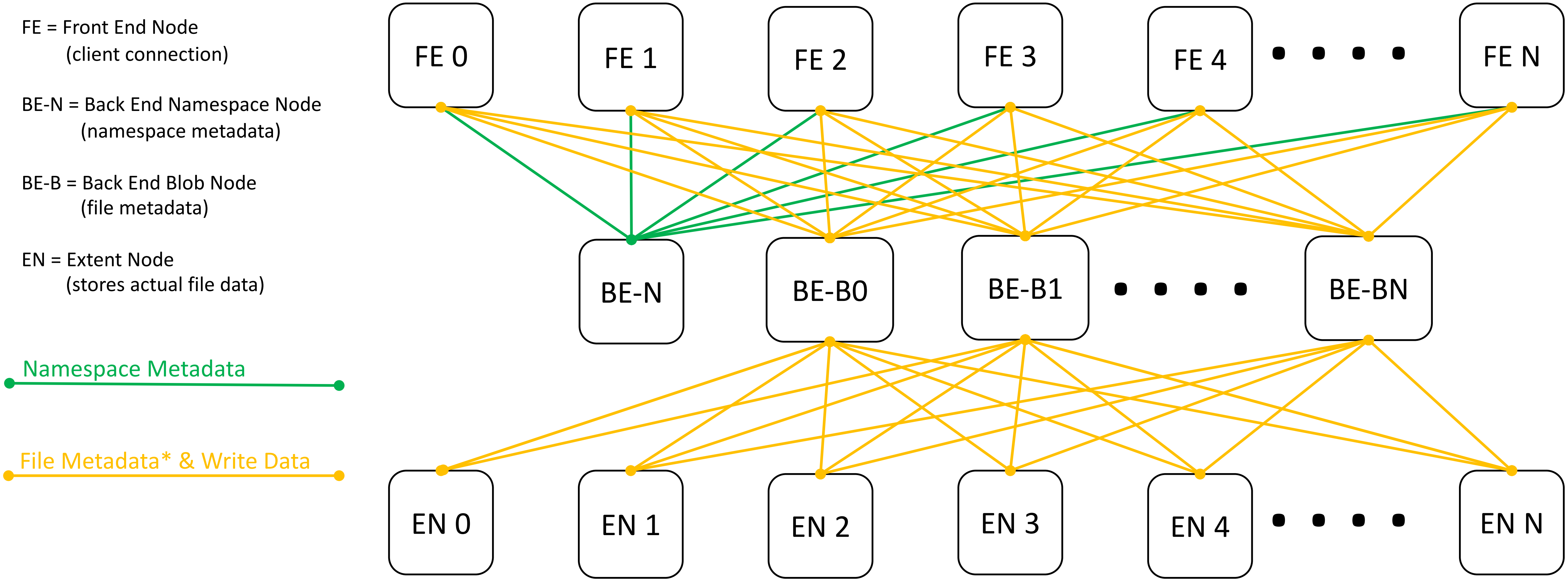
LFS State / Data Flow Topology on a Single Share

FE = Front End Node
(client connection)

BE-N = Back End Namespace Node
(namespace metadata)

BE-B = Back End Blob Node
(file metadata)

EN = Extent Node
(stores actual file data)



*Write Data Only to EN

LFS State / Data Flow Topology on a Single Share

FE = Front End Node
(client connection)

BE-N = Back End Namespace Node
(namespace metadata)

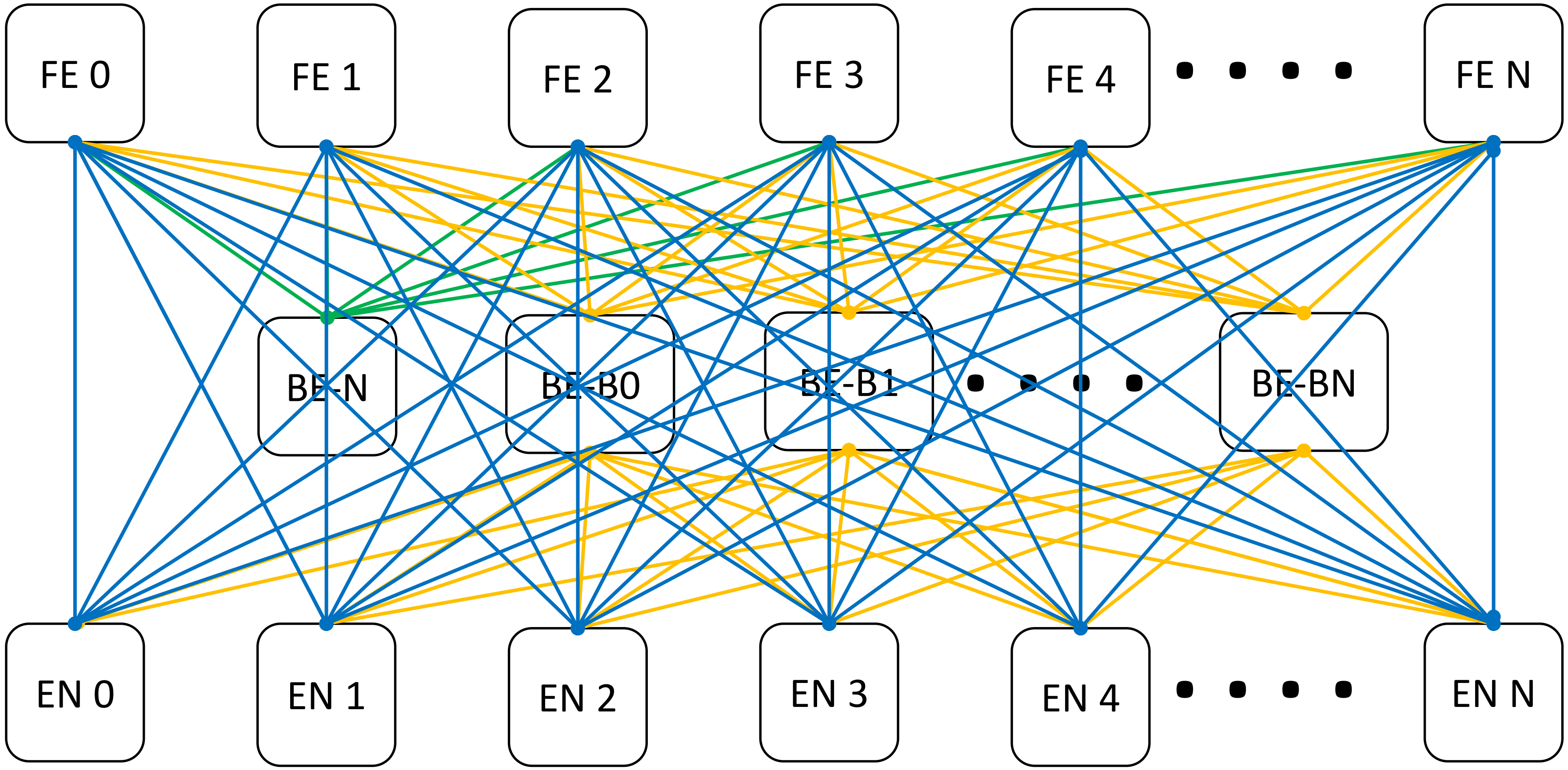
BE-B = Back End Blob Node
(file metadata)

EN = Extent Node
(stores actual file data)

Namespace Metadata

File Metadata* & Write Data

File Read Data



*Write Data Only to EN

File Row Migration

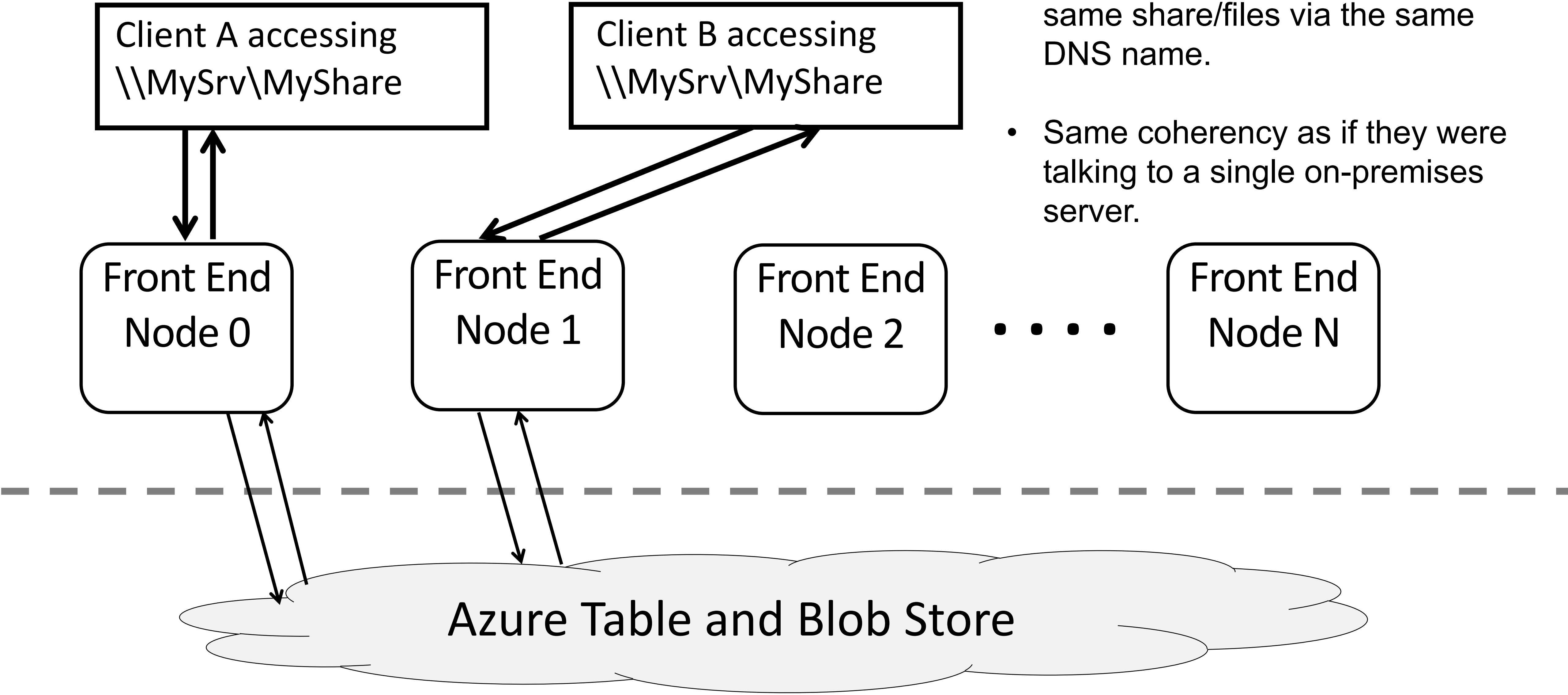
- Changing the partition key was a big deal.
- Additional state now needs to be replicated between the namespace and data partitions, namely handles and leases.
- A background process performed this for all open files.
- A partition could be in a “mixed” state for an extended period.
- If a handle was used before its file row had been migrated an on-demand migration is performed.
- This created a seamless experience for customers, unaware that a major schema change happened underneath them.

Tiering State By Durability Requirement

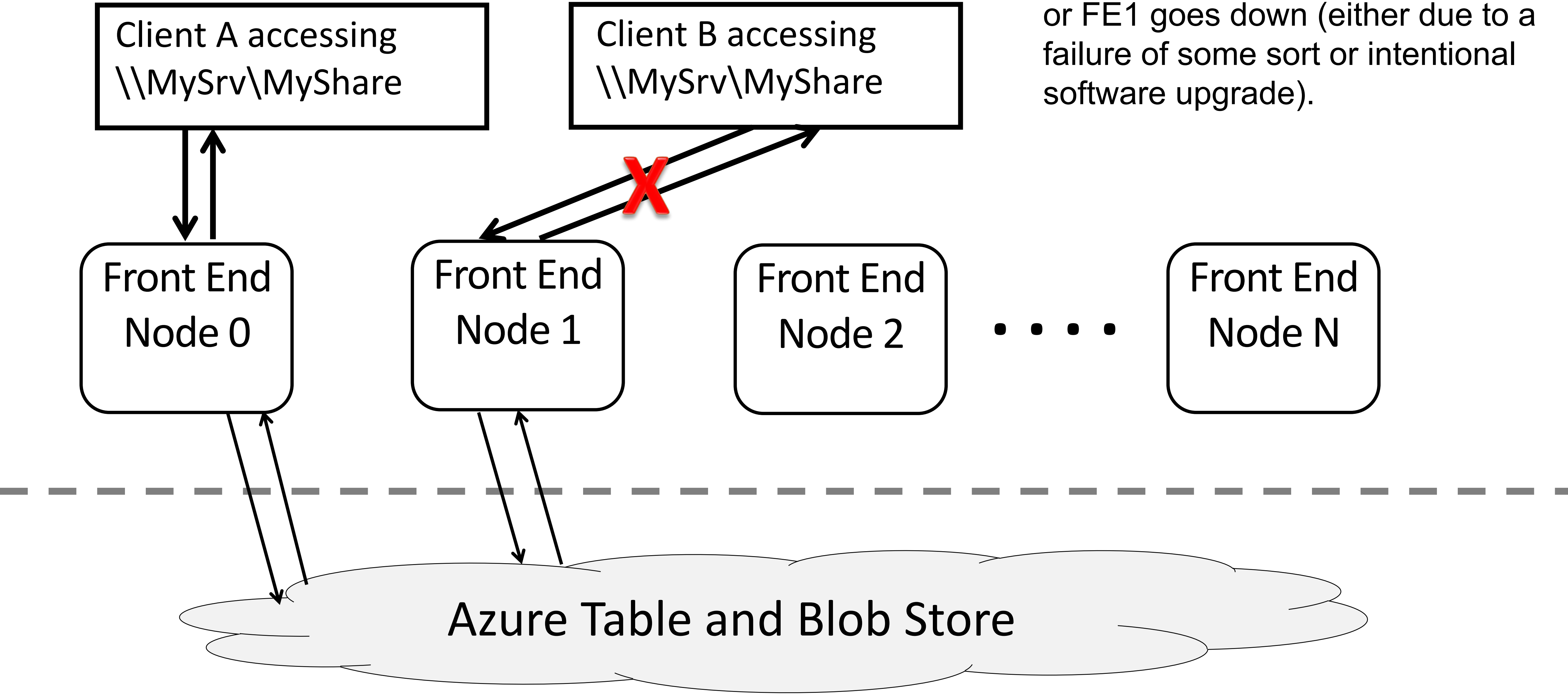
- A conventional file server treats only actual file data and essential metadata (filesize, timestamps, etc) as needing to be durably committed before an operation is acknowledged to the client (and even then only if opened WriteThrough).
- For true active/active high availability and coherency between FrontEnd nodes, modified state that normally exists only in server memory must be durably committed.

Example: Persistent Handles

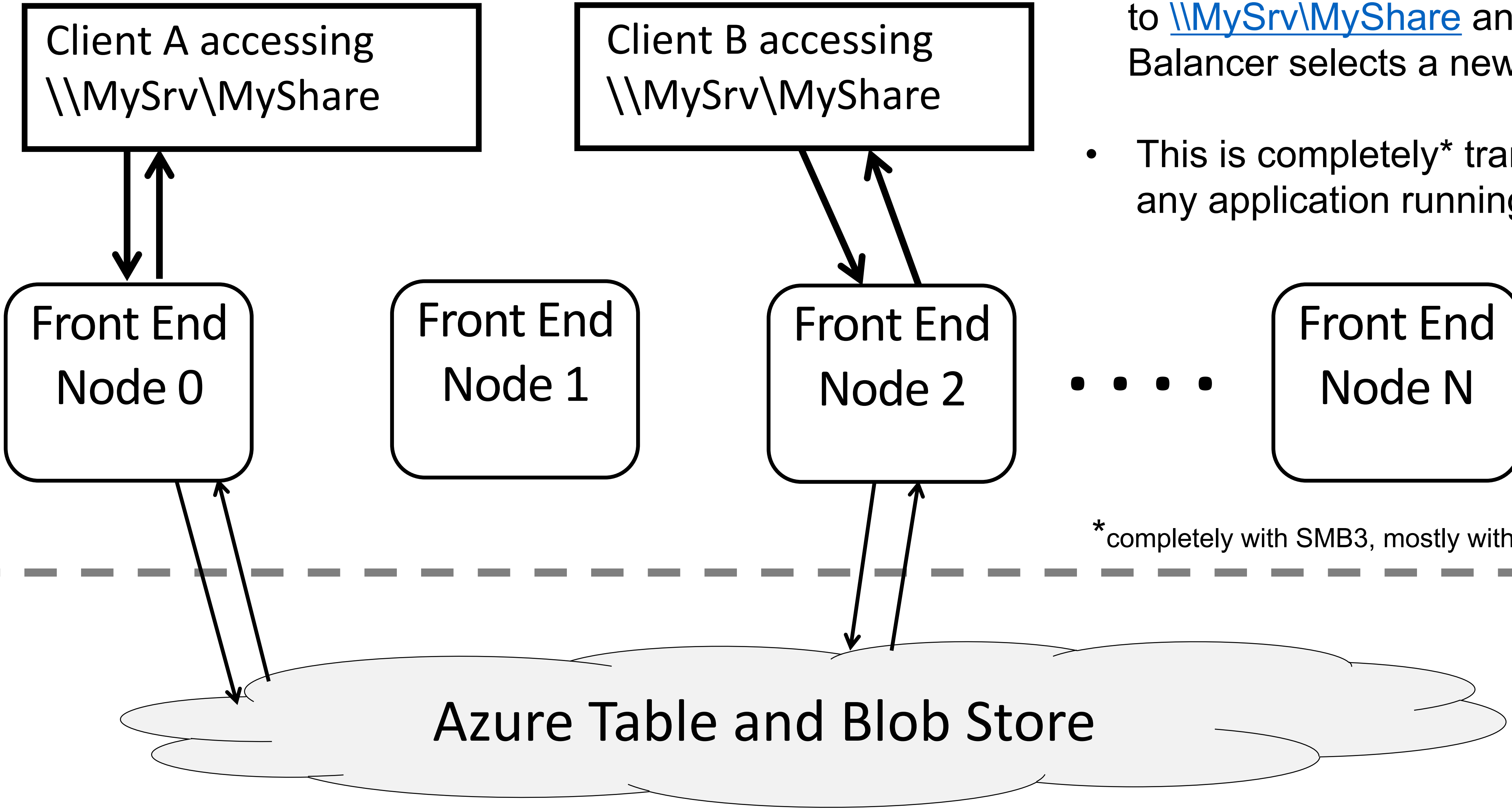
- New in SMB3, an improved version of Durable Handles.
- Persistent Handles are actually intended to support Transparent Failover when the server dies.
- Leverages state on the client for replay detection so that ‘once only’ operations are only executed once.
- More create request details durably committed with the handle.
- With Durable Handles SMB 2.1 protocol compliance required us to artificially limit our capability. With Persistent Handles we have seamless Transparent Failover.


Client A accessing \\MySrv\MyShare
Client B accessing \\MySrv\MyShare
Front End Node 0
Front End Node 1
Front End Node 2
. . . .
Front End Node N
Azure Table and Blob Store

- Clients A & B both accessing the same share/files via the same DNS name.
- Same coherency as if they were talking to a single on-premises server.


Client A accessing \\MySrv\MyShare
Client B accessing \\MySrv\MyShare
X
Front End Node 0
Front End Node 1
Front End Node 2
• • • •
Front End Node N
Azure Table and Blob Store

- Clients B loses connection to FE1 or FE1 goes down (either due to a failure of some sort or intentional software upgrade).

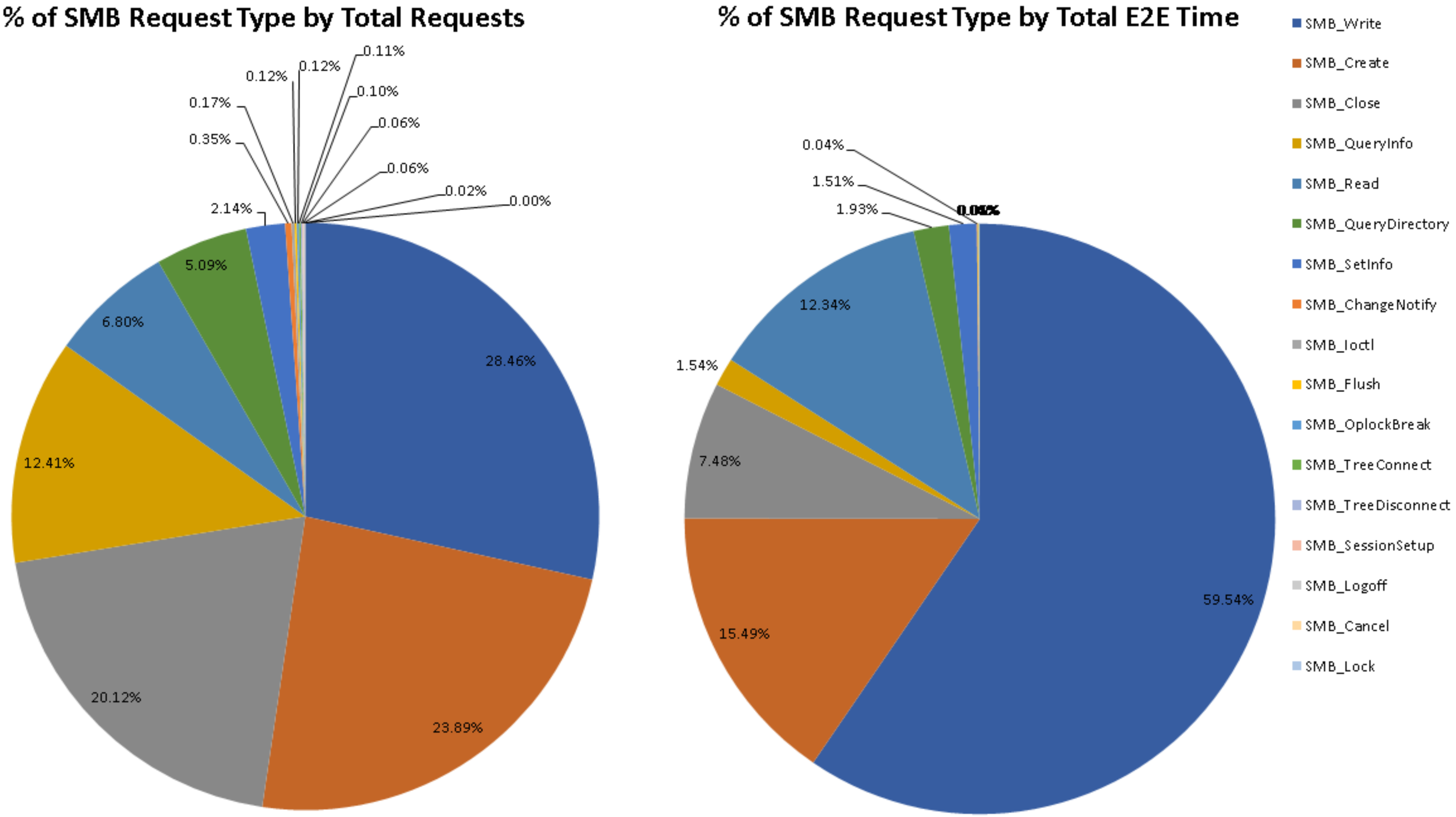

Client A accessing \\MySrv\MyShare
Client B accessing \\MySrv\MyShare
Front End Node 0
Front End Node 1
Front End Node 2
• • • •
Front End Node N
Azure Table and Blob Store

- Client B automatically reconnects to \\MySrv\MyShare and the Load Balancer selects a new FE.
- This is completely* transparent to any application running on ClientB.

*completely with SMB3, mostly with SMB2.1

Observations and Lessons Learned

- We now have some experience running the world’s largest SMB server.
- Metadata operations are unfortunately common and expensive for us.
- Even compared to srv2.sys on-prem, AF pays a high price for its durability. Open/Close and Write-Only handles are particularly bad.
- Some applications may not be suitable for “lift and shift”, especially if they have never even been run against an on-prem file server.
- In terms of total aggregate End-to-End request time, all that matters are Create, Close, Read and Write.



% of SMB Request Type by Total Requests

% of SMB Request Type by Total E2E Time

Specific Pain Points

- Leaked handles and the implication on (yet to be) deleted files.
- Leaked handles redux: absolute limits.
- Lack of server management people are used to on-prem.
- fopen(“foo”, “a”) on Windows.
- Variability in performance.
- Shared namespace with REST limited by HTTP restrictions.
- In general, poorly written Apps.

Specific Pain Points

- ~~Leaked handles and the implication on (yet to be) deleted files.~~
- ~~Leaked handles redux: absolute limits.~~ New REST APIs to mitigate.
- ~~Lack of server management people are used to on-prem.~~
- fopen(“foo”, “a”) on Windows.
- Variability in performance.
- Shared namespace with REST limited by HTTP restrictions.
- In general, poorly written Apps.

Resources:

- Azure Files Documentation Home: https://docs.microsoft.com/en-us/azure/storage/files
- NTFS features currently not supported: https://msdn.microsoft.com/en-us/library/azure/dn744326.aspx
- Naming restrictions for REST compatibility: https://msdn.microsoft.com/library/azure/dn167011.aspx
- Authentication Options for Azure Files: https://docs.microsoft.com/en-us/azure/storage/files/storage-files-active-directory-overview
- LFS Generally Availability announcement: https://azure.microsoft.com/en-gb/blog/announcing-the-general-availability-of-larger-more-powerful-standard-file-shares-for-azure-files

Thank You!

Questions?

EXTRA SLIDES

Additional Kerberos Details

1. Provision an object in customer’s respective domain service
   - User/computer object in AD/AAD DS and an Azure AD application/service principal in Azure AD Kerberos.

2. Object contains a service principal name
   - cifs/<storageaccount>.file.core.windows.net as well as a password that matches a storage account key specifically for Kerberos .

3. Sets up Kerberos authentication as it basically ‘joins” the storage account to the respective domain service.

Multi-table requests

- Namespace oriented requests make the heaviest use of transactions across multiple linked tables in a partition.
- Open/Create/Close will make modifications to at least two tables. Close can be particularly involved.
- Even reads/writes have to look at byte range locks and potentially break leases.
- The built-in transaction support made this relatively painless….before large file shares.

Examples of state tiering

- Ephemeral state: SMB2_FILEID.Volatile, credits, tcp socket details.
- Immutable state: 64bit actual FileId, IsDirectory
- Solid* durable state: SMB2_FILEID.Persistent, SessionId
- Fluid durable state: Open counts, file names, file size, lease levels and many more. This is the largest group of states.

*"Solid" here meaning the state is generated by AF and not generally changeable by normal actions of the client/application while "Fluid" is fully changeable by File APIs.

Achieving active/active high availability

- For a given share, state is tiered between FE nodes and the BE node depending on its durability requirements.
- All state required to correctly handle requests from different clients for the same file is managed by the BE node.
- This segregation of state together with table transactions in Azure enable active/active support.
- Large File Share support added distributed transactions.

Example: Durable Handle Reconnect

- Intended for network hiccups as it assumes all state is still valid on the server.
- Superseded by Persistent Handles on AF.
- On AF this state is durably persisted on our BackEnd so we’re able to ‘stretch’ durable handles to recover from FrontEnd AF failures (planned or otherwise) since it’s transparent to the client.
- This is important as we’re continually updating AF code requiring AF service restarts.