SDC 18
September 24-27, 2018
Santa Clara, CA
www.storagedeveloper.org

Improving Azure File Service: Adding New Wings to a Plane in Mid-flight

David Goebel
Microsoft

Azure File Service

Talk Topics:

0. Review of Azure Files design
1. Schema impacting new features
2. How new features are enabled in a service with zero downtime
3. As time allows, more Azure Files topics.

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.

AFS[1] Fundamental Concepts

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

[1] Azure File Service, not CMU's Andrew File System nor Azure File Sync.

Current AFS Status/Limits

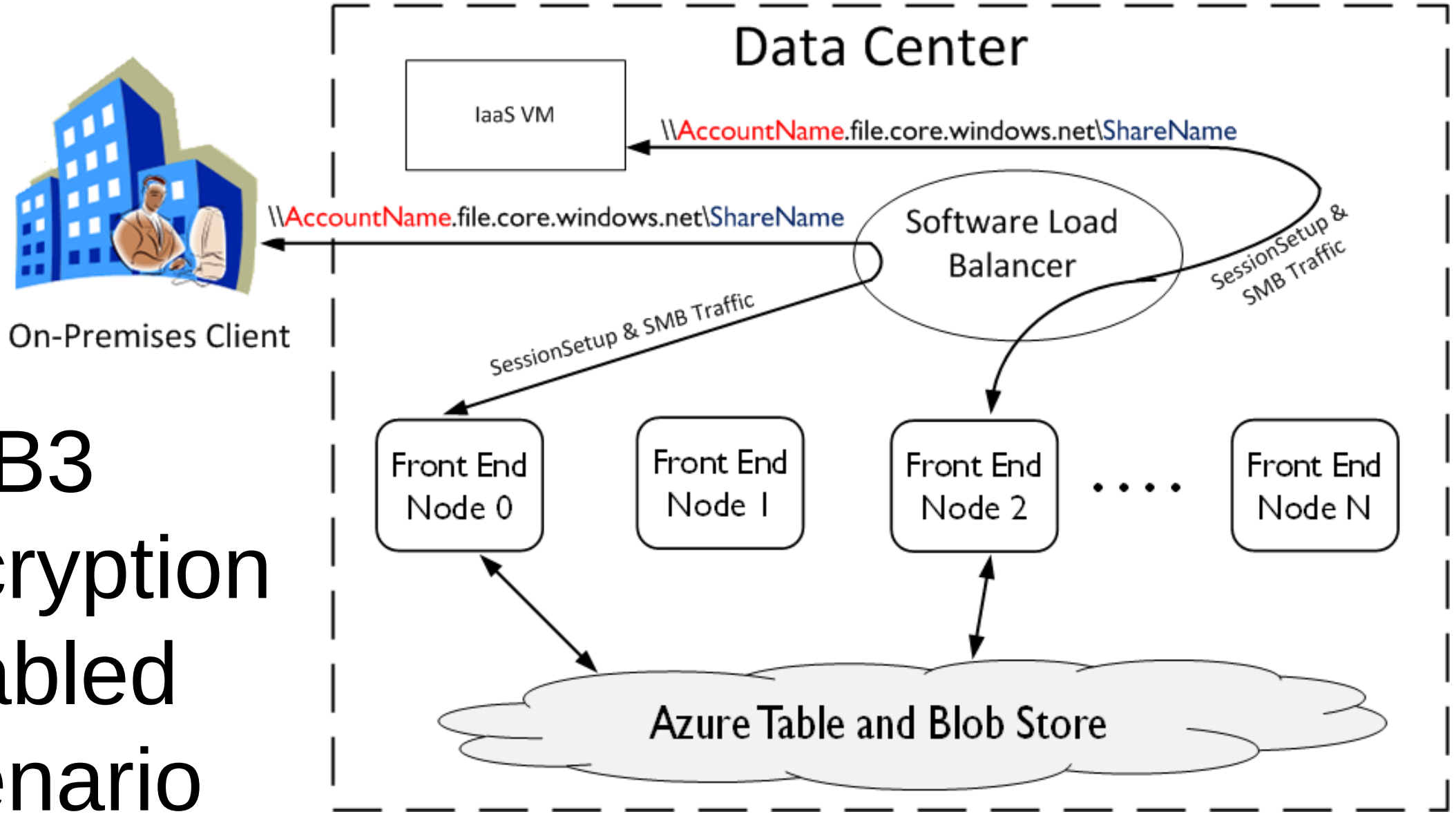
- SMB 3.0 w/encryption, persistent handles, and dir leases.
- 5TiB per share (100 TiB in private preview), and 1TiB per file.
- 1000 IOPS limit per share (up to 100,000 in private preview).
- ~60 MB/share/sec typical (up to ~5GiB/sec in private preview).
- Per-file ACLs with Kerberos authentication & AD integration in preview.
- Premium Files, i.e. SSD backed limited public preview.
- Share snapshots in production worldwide.
- Removed limitation on Private Use Area characters (0xE000 to 0xF8FF).
- Some NTFS features not supported.

Current Linux Support

Linux Distribution	Publisher	Kernel Version	CIFS Version	SMB3 Persistent Handles	SMB3 Encryption
Ubuntu Server 18.04 LTS	Canonical	4.15.0-34-generic	2.10	Yes	Yes
CentOS 7.5	Rogue Wave Software	3.10.0-862.2.3.el7	2.03	Yes	Yes
Debian 9	Credativ	4.9.0-3-generic	2.09	Yes	No
Open SUSE Leap 42.3	SUSE	4.4.156-5.1	2.08	Yes	Yes
SUSE Linux Enterprise Server 15	SUSE	4.12.14-25.16.1	2.09	Yes	Yes
Ubuntu Server 18.04 LTS	Canonical	4.15.0-34-generic	2.10	Yes	Yes

Note: Directory lease support has been in the mainline kernel for a couple releases, but hasn't been backported.

SMB3 Encryption Enabled Scenario



Note: Port 445 outbound must be unblocked.

DEMO

Scenarios Enabled By AFS

- Existing file I/O API (Win32, CRT, etc.) based applications, i.e. most business applications written over the last 30 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.

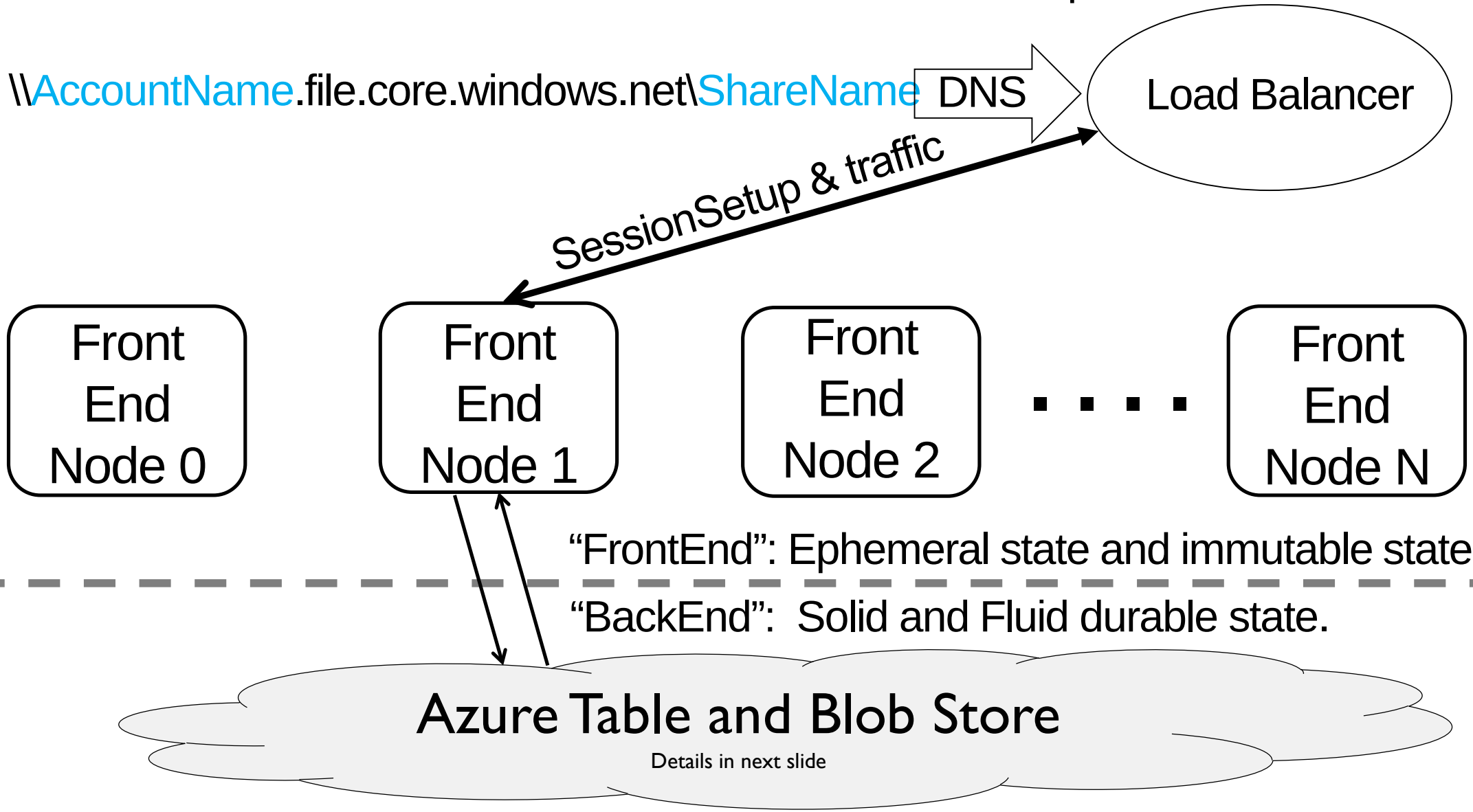
What about REST?

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 properties, Get/Set metadata
- Directory Operations:
  Create, Delete, Get Properties, List (Same as ListFiles)
- File operations:
  Create, List, Delete, Get/Set properties, Get/Set metadata, Get Contents, Put Ranges, List Ranges

for example 157.56.217.32:445

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



Azure Table and Blob Store

- AFS 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.
- There are two types of keys: partition and row.
- Table operations are transactional within a “partition”.

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.
- Currently, a share is wholly contained within a partition.
- In private preview, a file share may now span many partitions.
- This new feature is not limited to just new accounts.
- The schema impacting change is made on the fly as needed.

AFS Tables

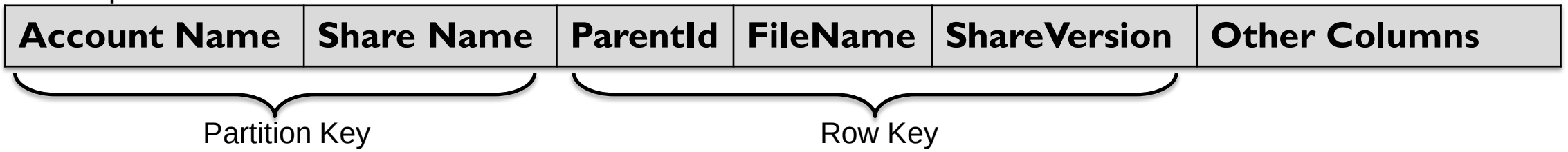
- Azure Tables allows associating a set of tables as a group.
- Within a partition, a single request can atomically operate on multiple tables in this group.
- An AFS share's metadata is stored as a group of tables, the most notable of which are:

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

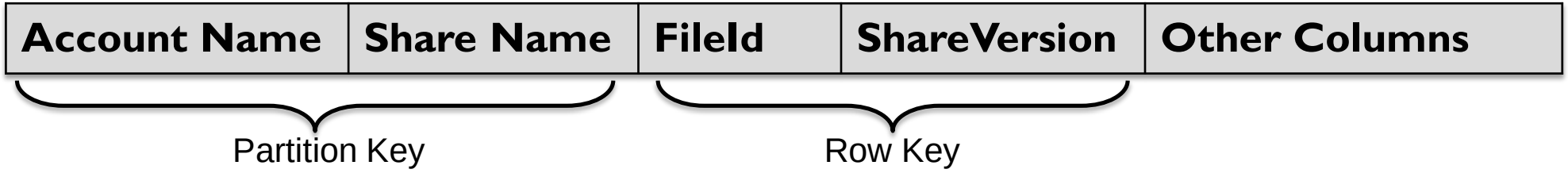
AFS File Table Row Keys

- 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



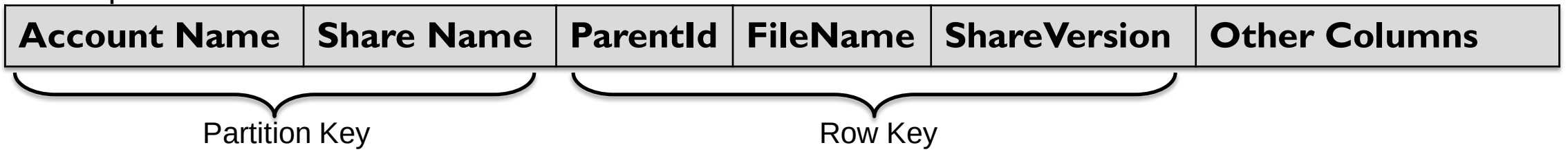
Data Rows: Current Production



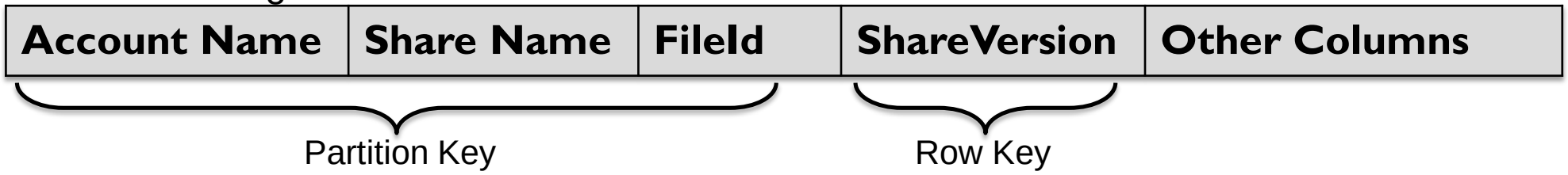
AFS File Table Row Keys

- 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



Data Rows: Large File Share Version

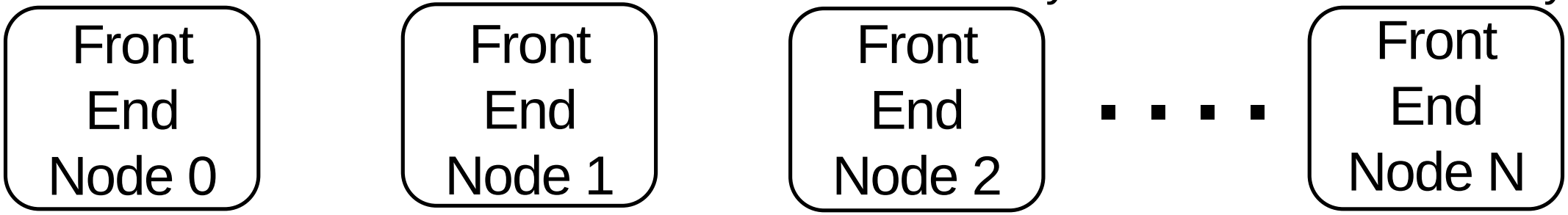


File Row Migration

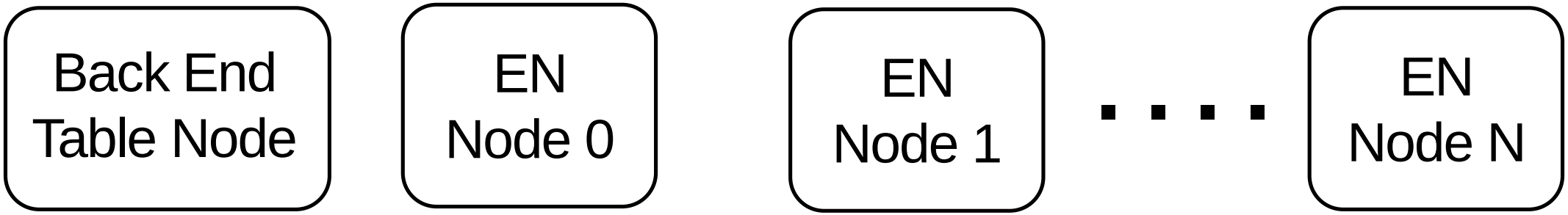
- Changing the partition key is a big deal.
- Additional state now needs to be replicated between the namespace and data partitions, namely handles and leases.
- A background process performs this for all open files.
- A partition can be in a “mixed” state for an extended period.
- If a handle is used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

Mapping AFS to hardware

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

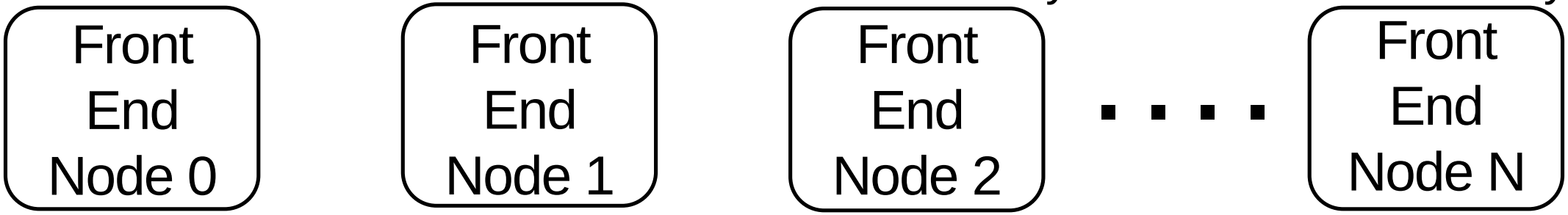


Currently a single share/container is within a single partition which is at any time "owned" by a single BE Table Node. A TableMaster service manages moving partition ownership in the case of BE node failure or for load balancing. Page blobs are managed by EN nodes.

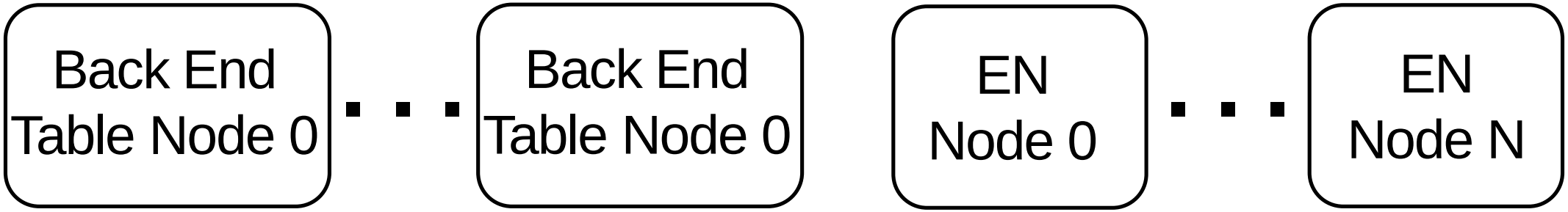


Mapping AFS to hardware

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



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



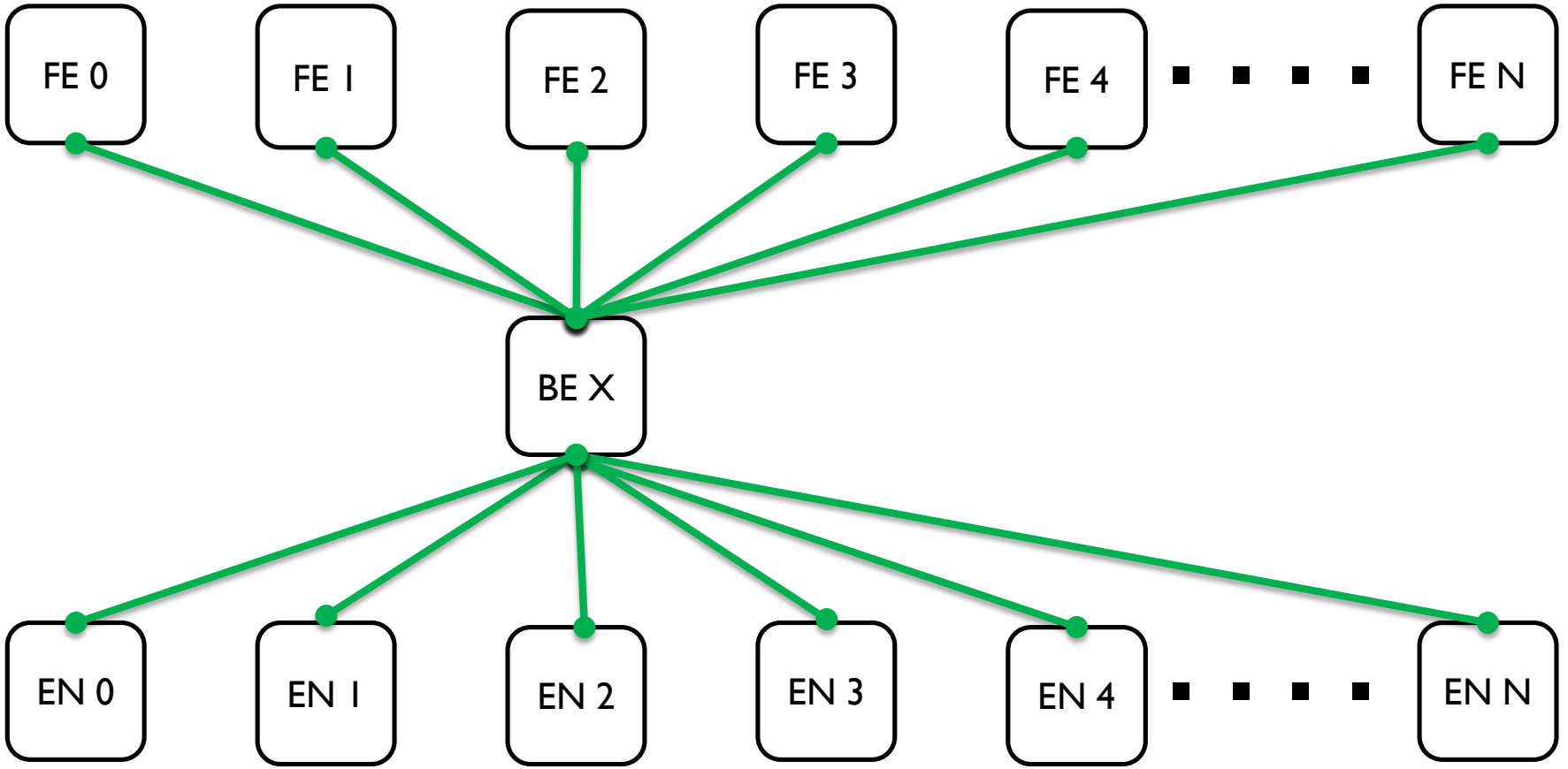
Original State / Data Flow Topology on a Single Share

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

Original State / Data Flow Topology on a Single Share

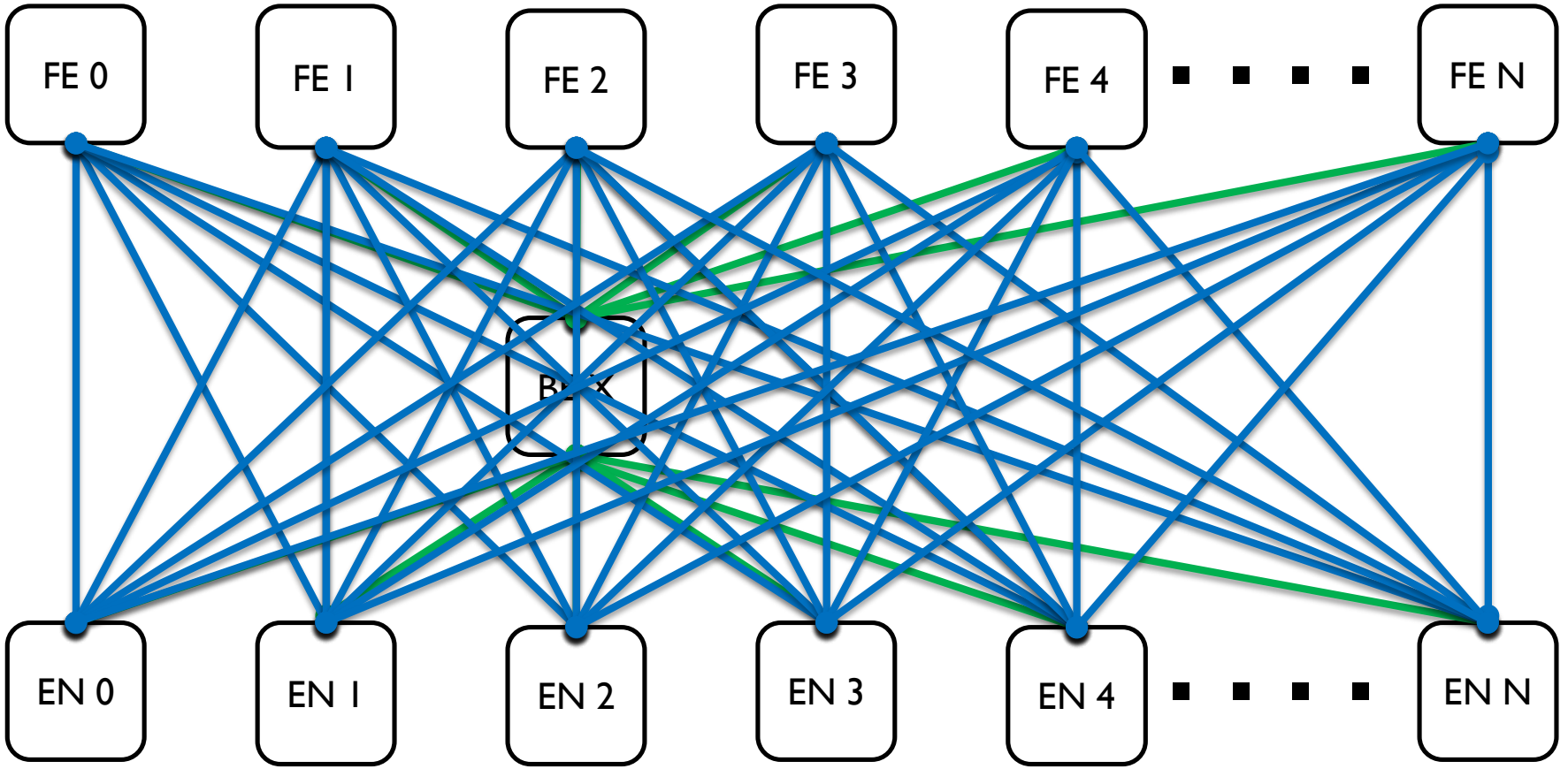
FE = Front End Node
(client connection)

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

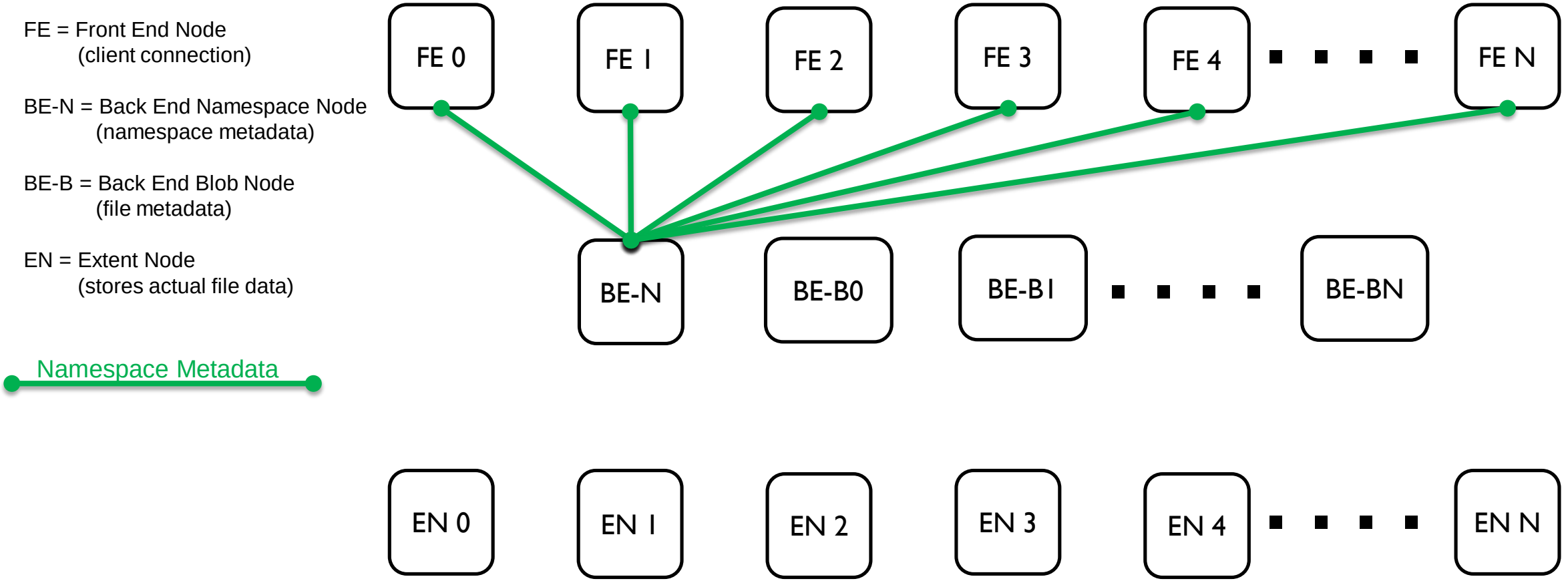
New State / Data Flow Topology on a Single Share

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)



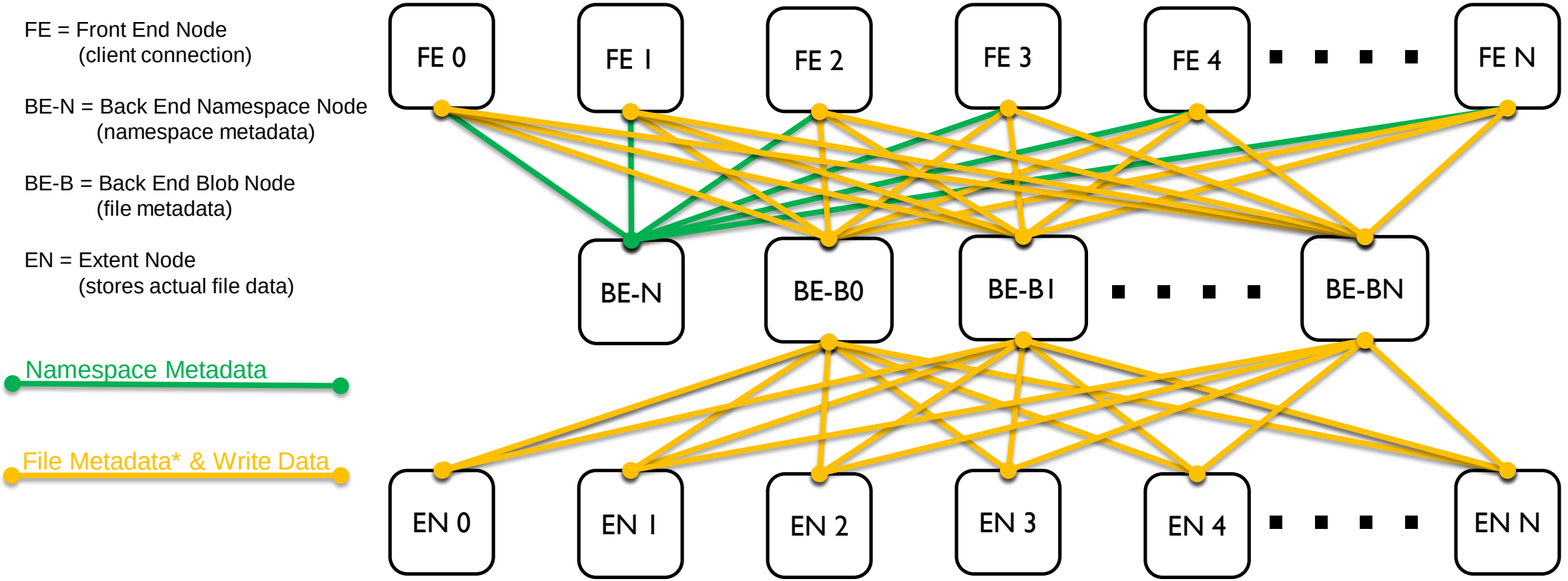
New State / Data Flow Topology on a Single Share

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

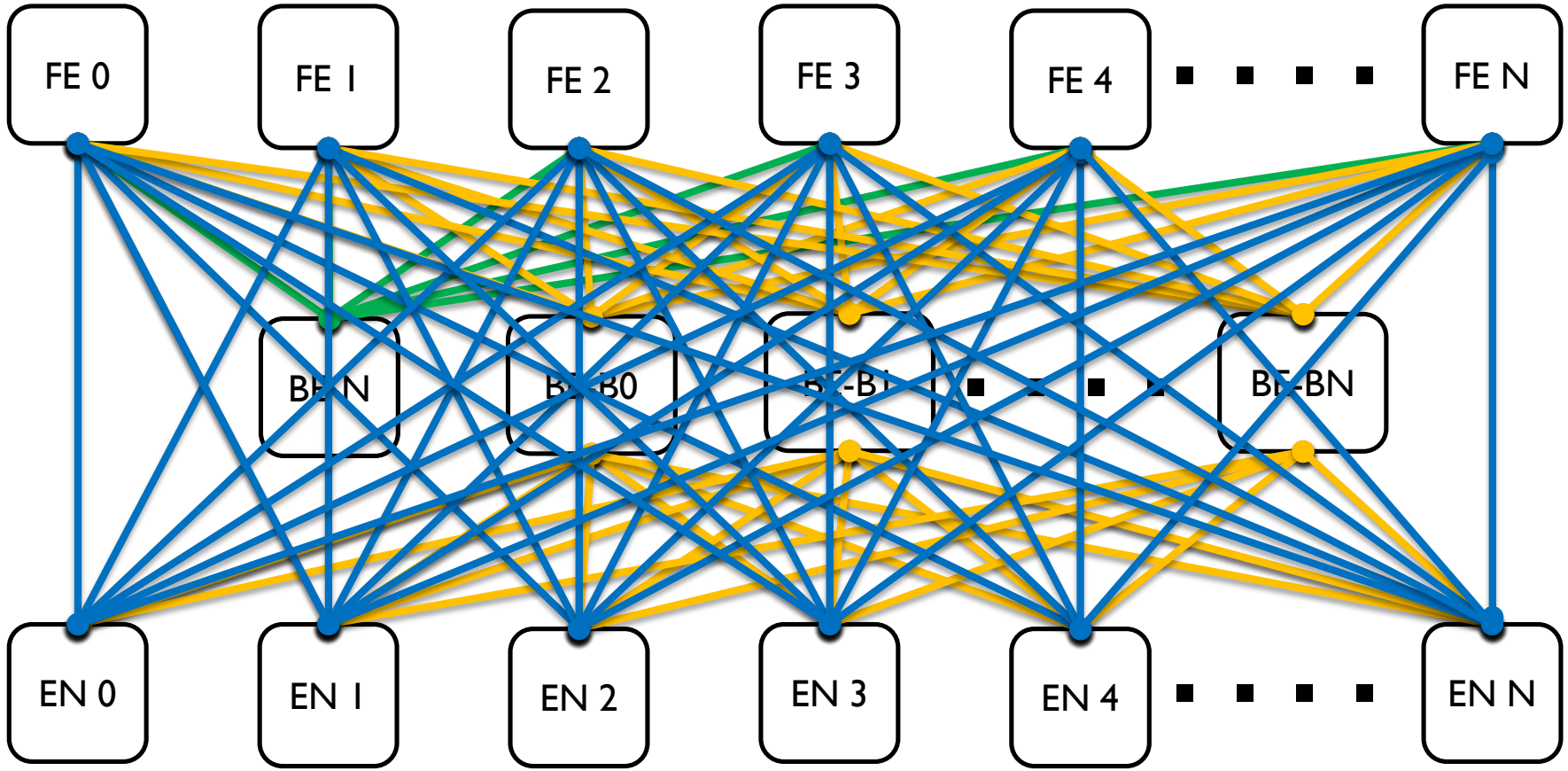
New State / Data Flow Topology on a Single Share

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

Multi-table requests

- Namespace oriented requests make the heaviest use of transactions across multiple tables.
- 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 makes this relatively painless….before large file shares.

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.
- This segregation of state together with table transaction support in Azure enable active/active support.
- Large File Share support has added distributed transactions.

Tiering State By Durability Requirement

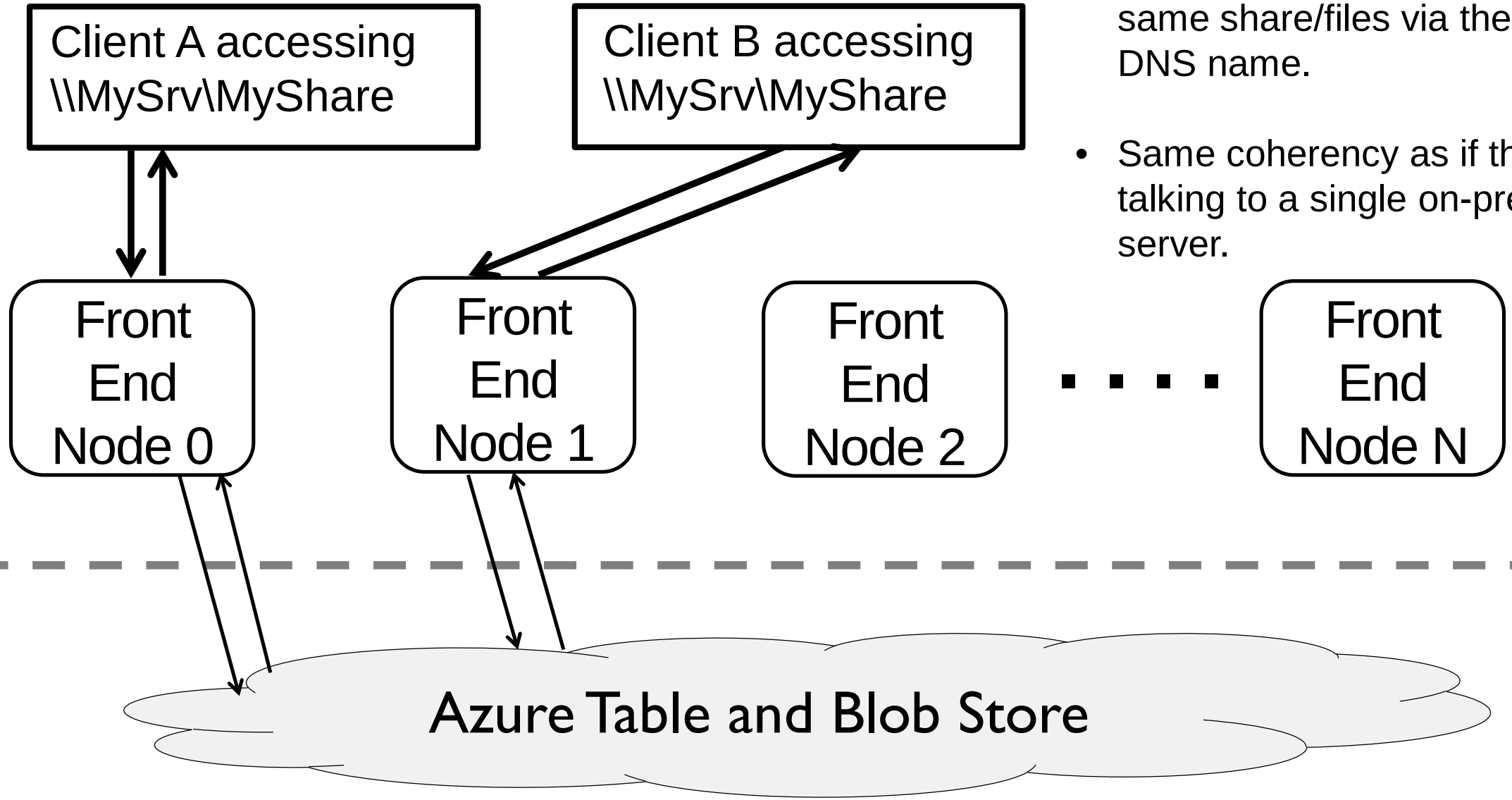
- A conventional file server treats only actual file data and essential metadata (filesize, timestamps, etc) as needing to be durably committed before an operation is acknowledged to the client (and even then only if opened WriteThrough).
- For true active/active high availability and coherency between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

SMB is a stateful protocol,

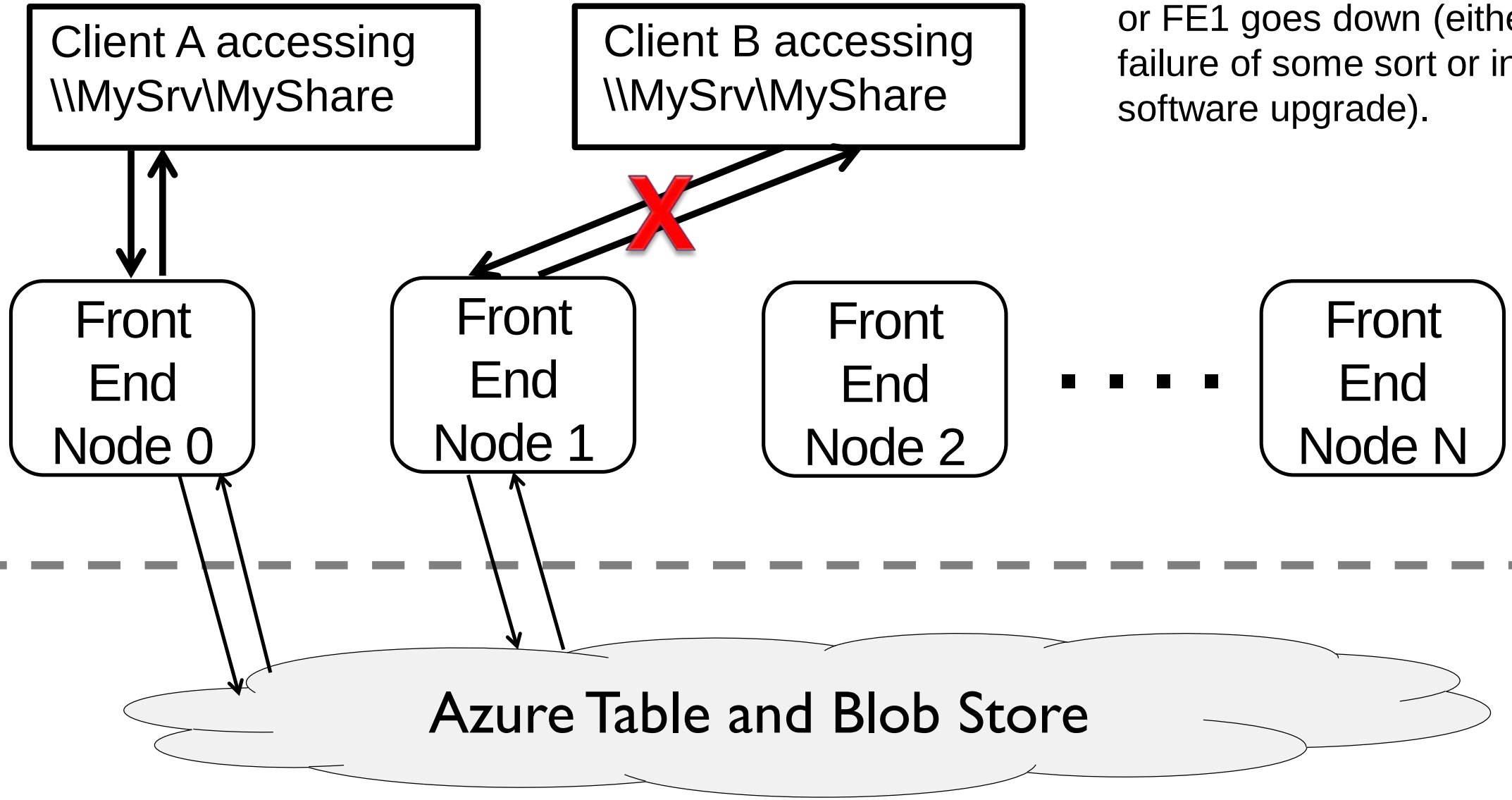
but not all states require expensive distributed transactional semantics

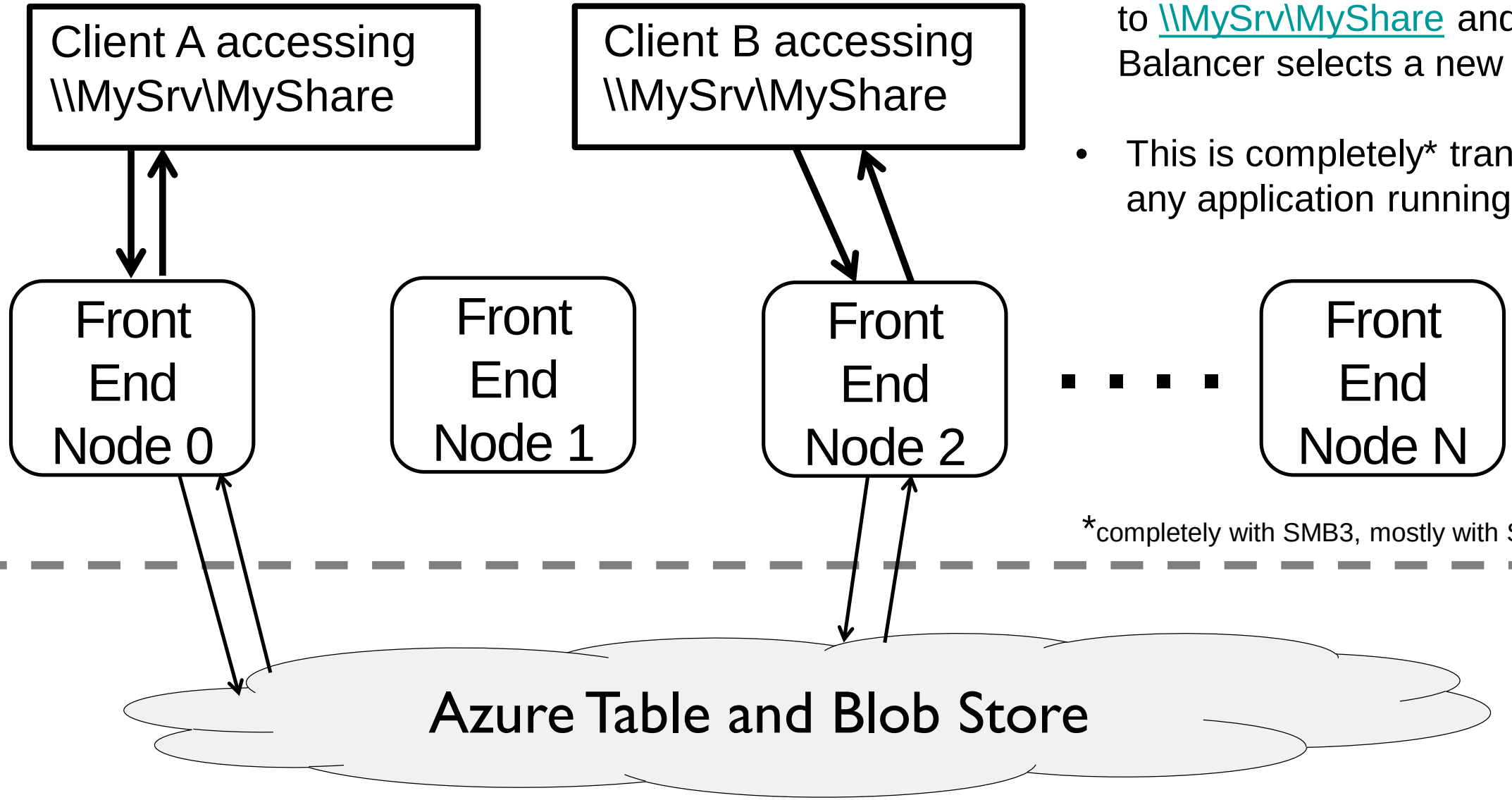
- Some aspects of a file's state are immutable, such as FileId and whether it's a file or a directory.
- Some state is transient, such as open counts, and can be optimized if loss of this state is acceptable in a disaster.
- Some state is also maintained by the client, like CreateGuid, drastically reducing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

- 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.



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





- 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

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 AFS and not generally changeable by normal actions of the client/application while “Fluid” is fully changeable by File APIs.

Example: Durable Handle Reconnect

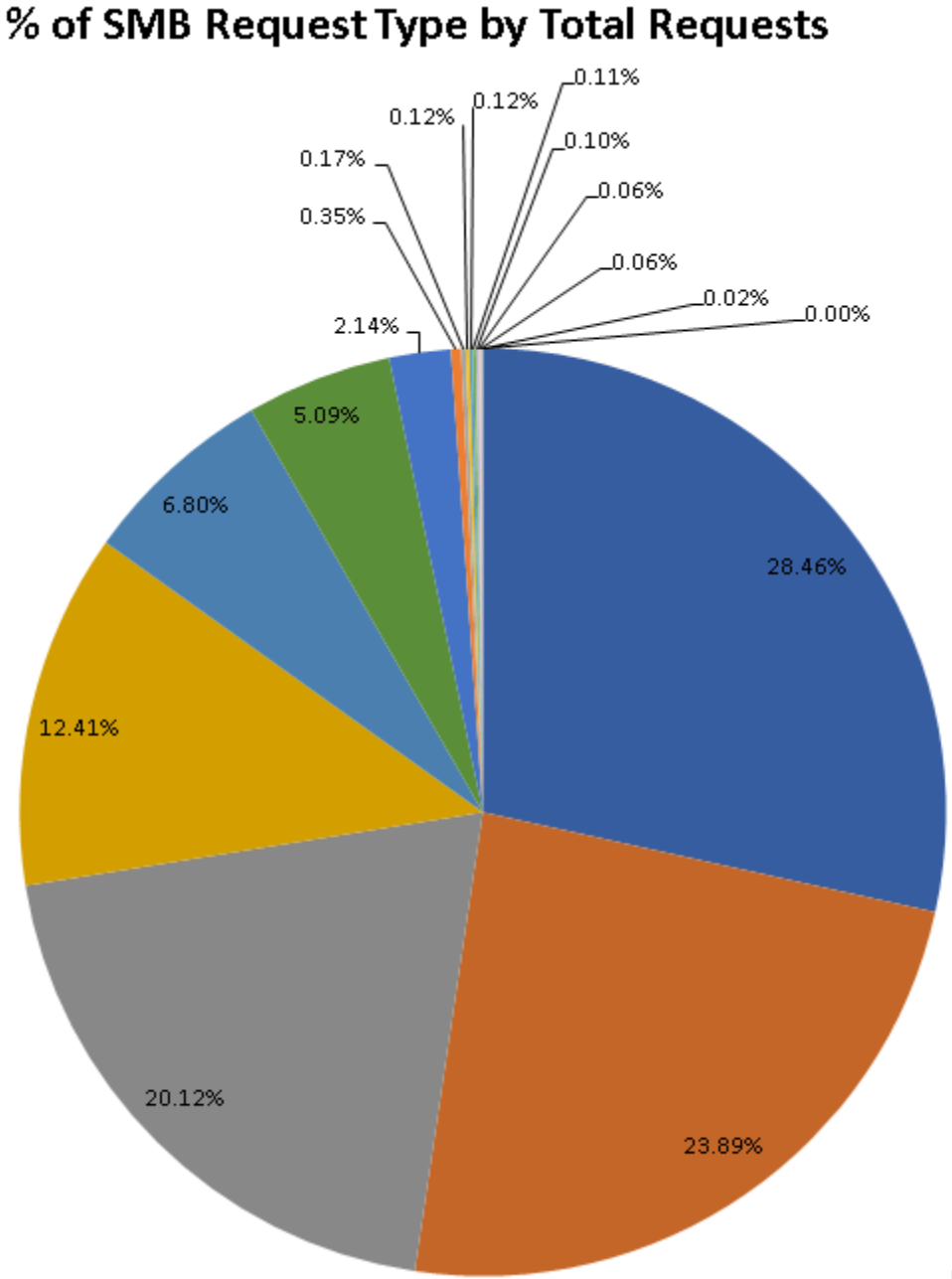
- Intended for network hiccups as it assumes all state is still valid on the server.
- On AFS this state is durably persisted on our BackEnd so we're able to 'stretch' durable handles to recover from FrontEnd AFS failures (planned or otherwise) since it's transparent to the client.
- This is important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

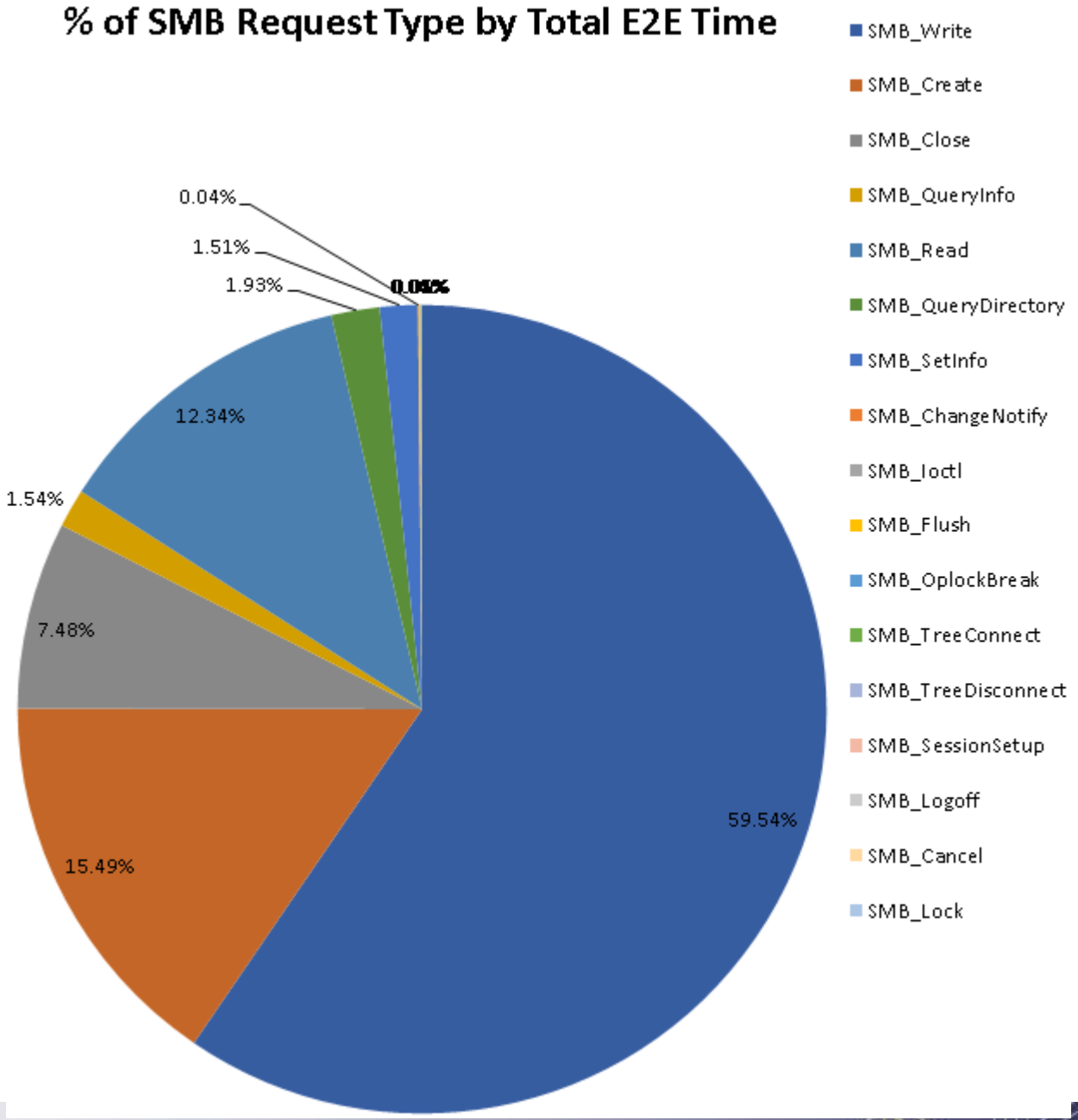
Example: Persistent Handles

- Unlike 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.

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, AFS 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.





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”).
- Variability in performance.
- Shared namespace with REST limited by HTTP restrictions.
- In general, poorly written Apps.

Resources:

- Getting started blog with many useful links: http://blogs.msdn.com/b/windowsazurestorage/archive/2014/05/12/introducing-microsoft-azure-file-service.aspx
- Generally Availability announcement: https://azure.microsoft.com/en-us/blog/azure-file-storage-now-generally-available
- 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

Thank You!

Questions?