SDC
STORAGE DEVELOPER CONFERENCE
SNIA SANTA CLARA, 2016

Azure-consistent Object Storage in Microsoft Azure Stack

Ali Turkoglu
Principal Software Engineering Manager
Microsoft

Mallikarjun Chadalapaka
Principal Program Manager
Microsoft

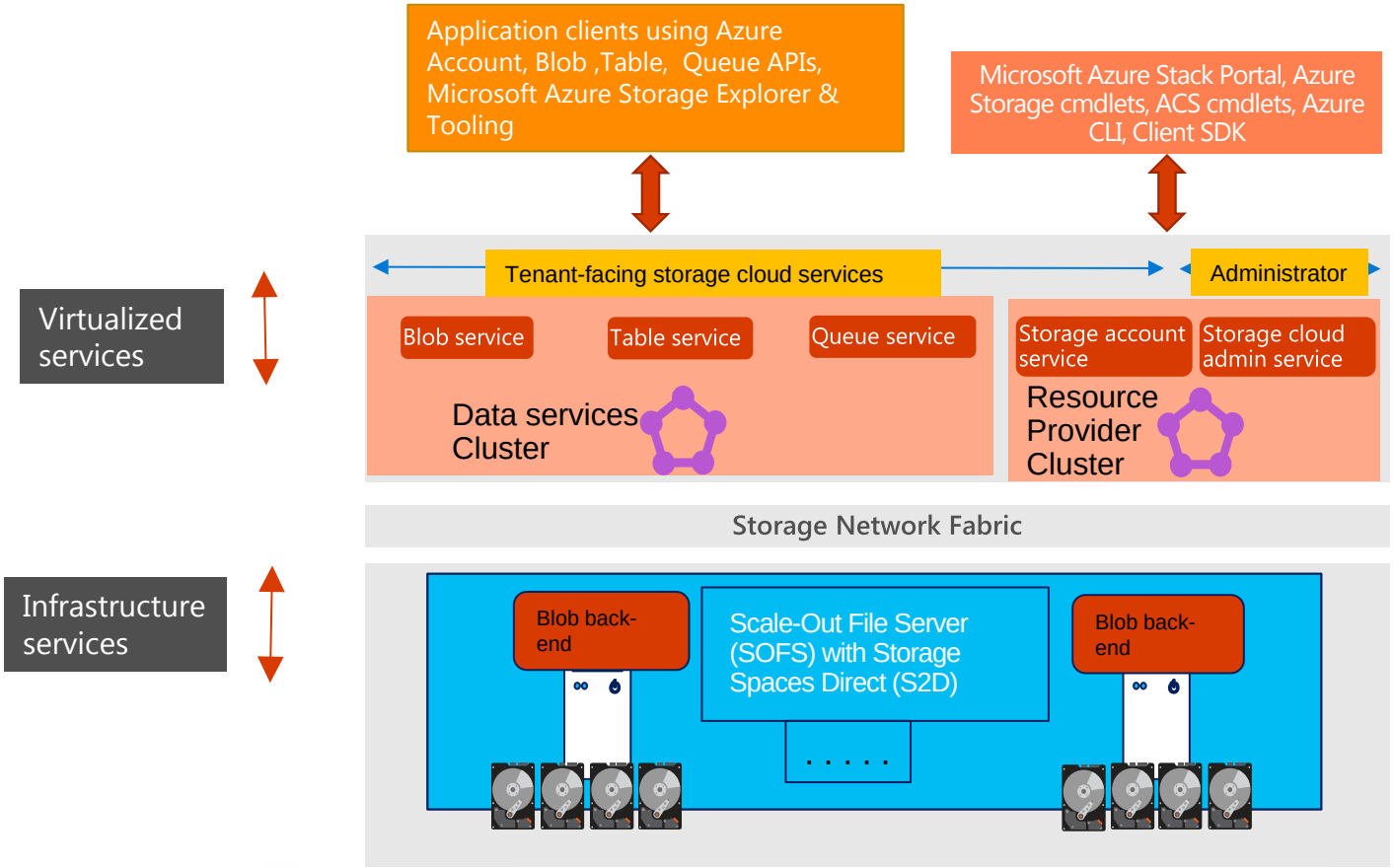
Agenda

- Context, Solution Stack, Architecture, ARM & SRP
- ACS Architecture Deep Dive
- Blob Service Architecture & Design
- Questions/Discussion

Azure-consistent storage

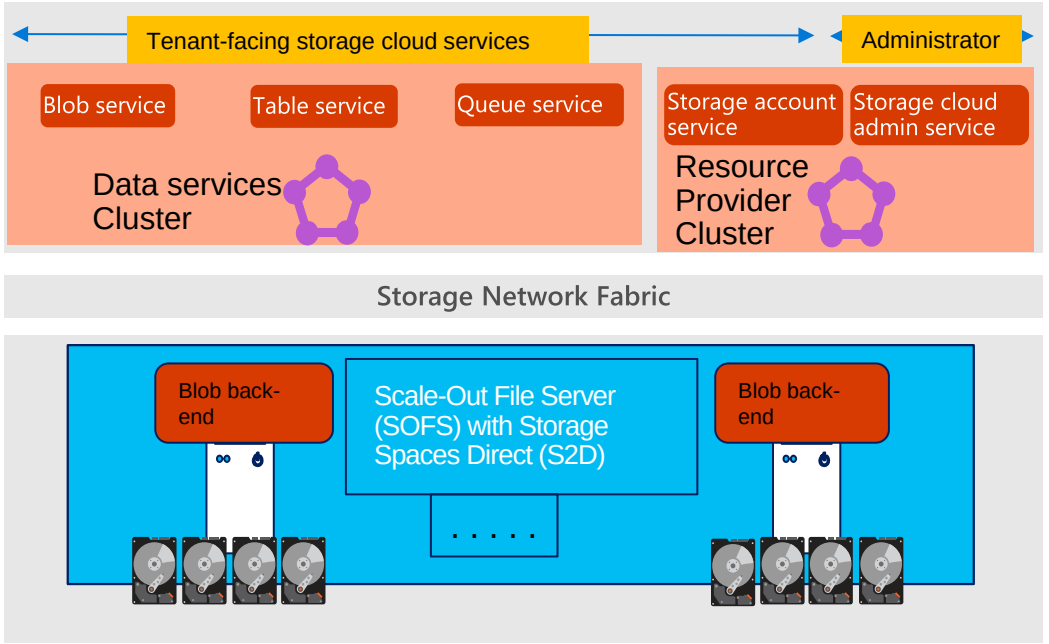
- Cloud storage for Azure Stack
- Azure-consistent blobs, tables, queues, and storage accounts
- Administrator manageability
- Enterprise-private clouds or hosted clouds from service providers
- IaaS (page blobs) + PaaS (block blobs, append blobs, tables, queues)
- Builds on & enhances WS2016 Software-Defined Storage (SDS) platform capabilities

Azure-consistent storage: Solution view



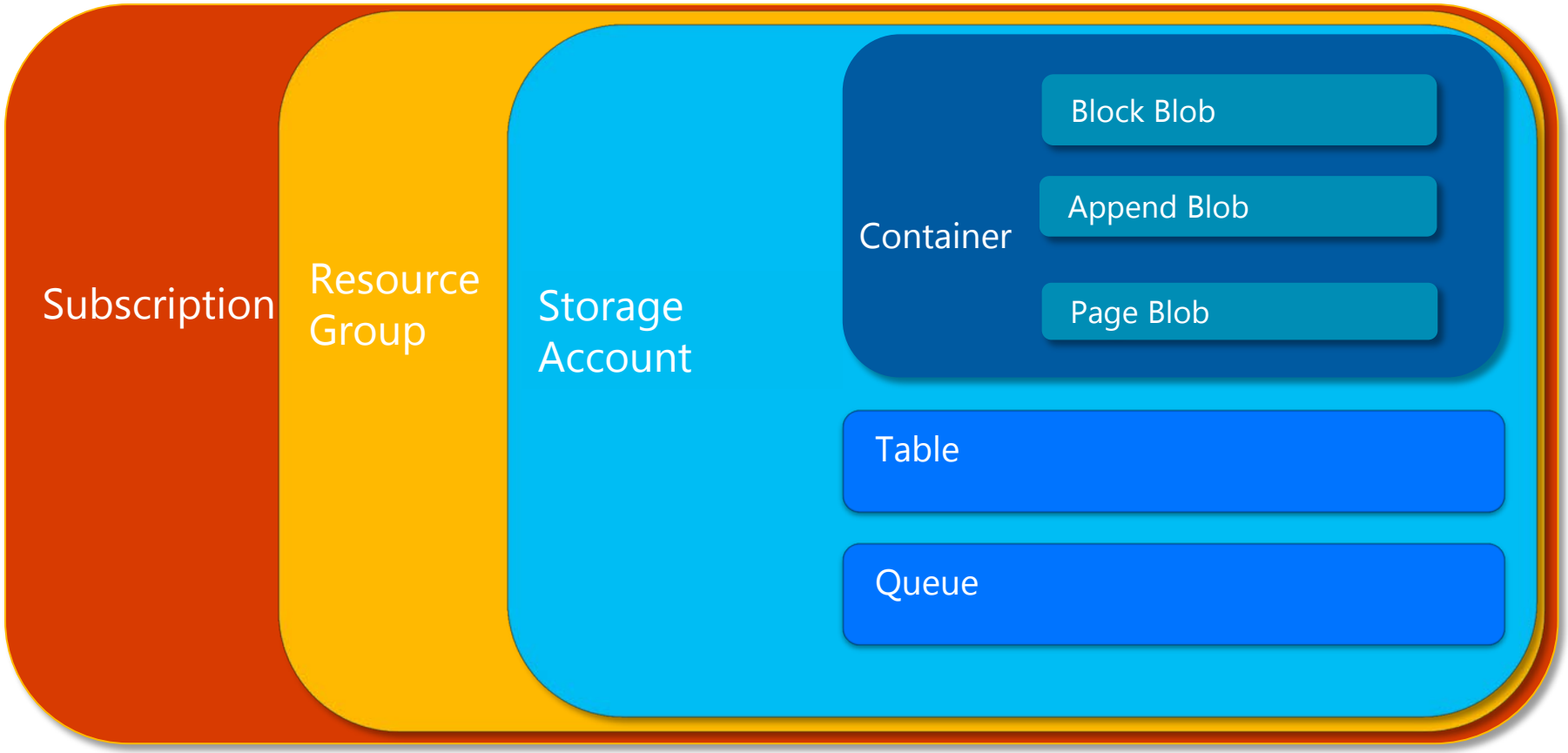
Clustering Architecture

- Azure Service Fabric (ASF) guest clusters for cloud services
- Hyper-converged Windows Server Failover Cluster (WSFC) Host fabric
- WSFC enhances ASF cluster resiliency
  - HA VMs via Hyper-V host clustering
  - Anti-affinity policies on VMs to ensure all VMs never failover to same host
  - Application health monitoring on Service Fabric service for timely detection of service hang situations
- WSFC & CSVFS* provide the basis for blob service HA model



*Cluster Shared Volume File System

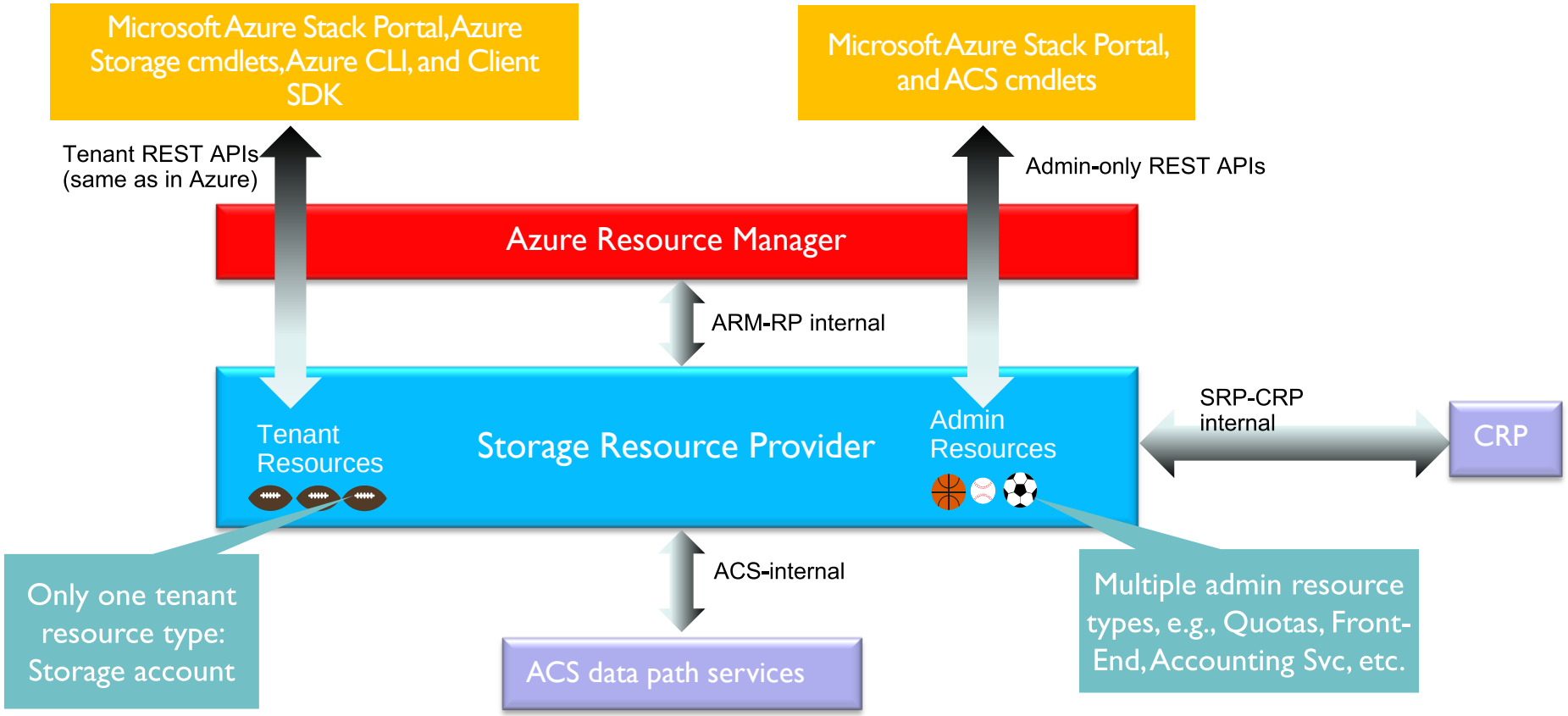
Relating Azure Storage Concepts



ARM & Resource Providers

- Azure Resource Manager (ARM) in Azure Stack
  - Azure-consistent management
  - Clients use REST, PS, or Portal
- Resource Provider (RP) manages a type of infra
  - Plug-in to ARM
  - Compute (CRP)
  - Network (NRP)
  - Storage (SRP)
  - ….
- Users express desired state via templates
  - Template = declarative statement
  - ARM → necessary orchestration + imperative directives to RPs

Azure-consistent storage Management Model



IaaS VM Storage

- All VM storage in Azure Stack resides in blob store
- Every OS or Data Disk is a page blob
  - Page blob → ReFS file
  - Starts in REST API access mode
- CRP and SRP collaborate behind the scenes
  - Page blob toggles to ‘SMB-only’ at VM run time
  - Super-optimized Hyper-V-over-SMB I/O path

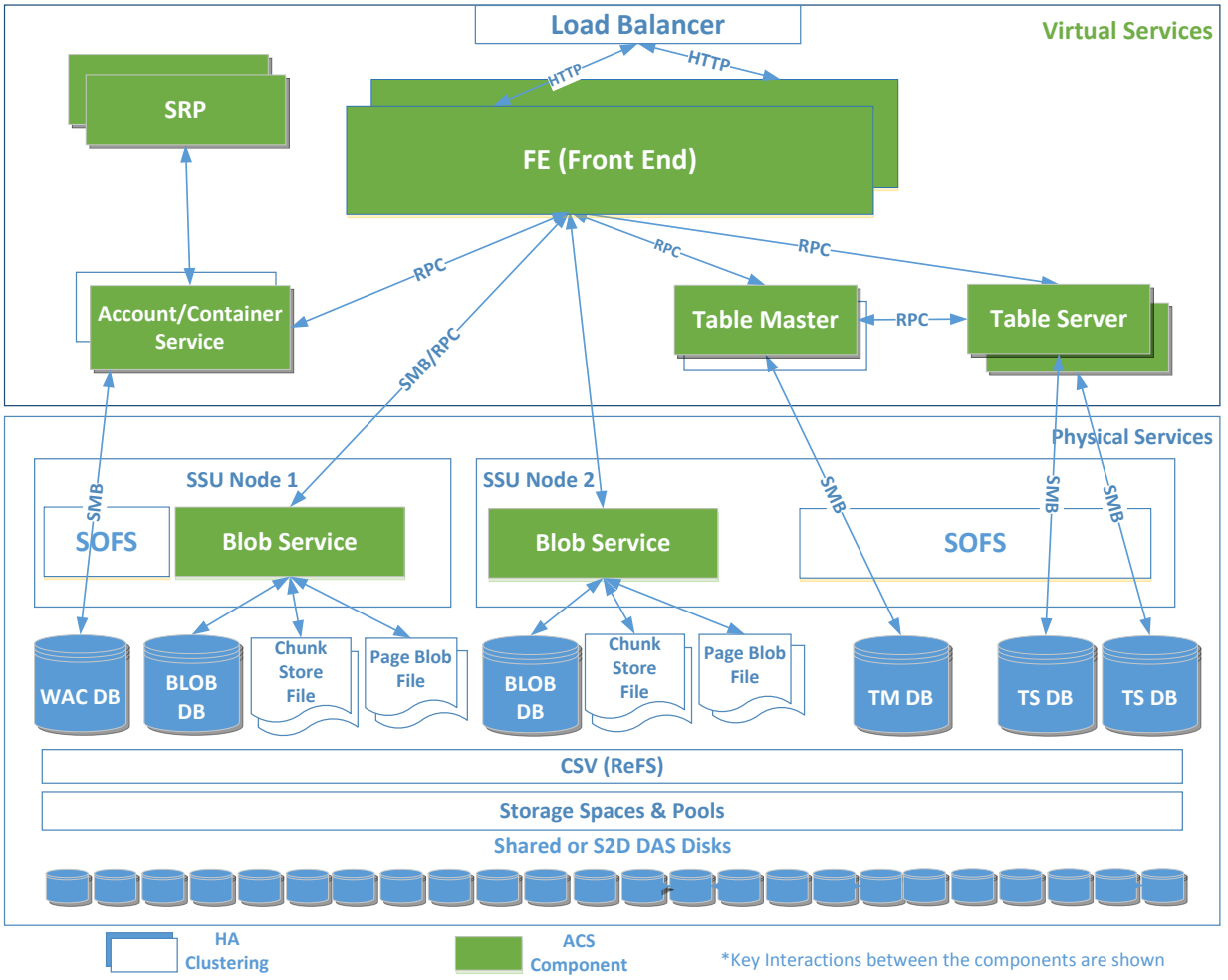
ACS Architecture Deep Dive

Key Requirements and Challenges for Object Storage on MAS

Atomicity guarantees	Data Consistency guarantees	Immutable blocks	Snapshot isolated copy for reads
512 byte page alignment for page blobs	Distributed List Blob (enumeration)	Durability: Synchronous 3-copy replication.	Scalable to millions of objects
99.9% High availability read/write	Fault tolerance & Crash consistency	No performance regression relative to Hyper-V over SMB	Adapt to smaller cloud scale

ACS Architecture

- **SRP (Storage Resource Provider):** Integrates with ARM and exposes resource management REST interfaces for storage service overall.
- **FE**: Provides REST interface Front End, consistent with Azure.
- **WAC:** Storage account management, user requests authentication, and container operations.
- **Blob Service:** Implements the blob service backend. Stores block and page blob data in file system/Chunk Store, and metadata in ESENT DB.
- **Table Master:** Maps user table to database/TS instances.
- **Table Server:** Handles table query & transactions in databases.
- **Storage:** SOFS exposes a unified view of all tiered storage to compute nodes as CA shares. Provides fault tolerance & local replication.



Blobs - Semantic Requirements: See MSDN

BLOCK BLOBS

- Client uploads individual **immutable blocks** with PUT-BLOCK for future inclusion in a block blob. Block size may be up to **4 MB**. A blob can have up to 100,000 uncommitted blocks. Maximum size of uncommitted block list is 400 GB.
- Followed by a PUT-BLOCK-LIST call to assemble the blob. Maximum size supported for Block Blob is **200 GB** and **50,000 committed blocks**.
- Blocks must retain their identity to permit later PUT-BLOCK-LIST calls to re-arrange.
- Unused blocks are lazily cleaned up after a PUT-BLOCK-LIST request. In the absence of PUT-BLOCK-LIST, uncommitted blocks are garbage collected after 7 days.
- Blob names are **case-sensitive**. At least one character long, and at most 1024 characters.
- All Blob operations **guarantee atomicity** where it either happened as a whole, or it has not happened at all. There is no undetermined state at failure.
- For Block Blobs each GET-BLOB request gets a **snapshot isolated copy** of the Blob data (or request fails if this cannot be accommodated.

PAGE BLOBS

- Client creates a new *empty* page blob by calling PUT-BLOB. A page Blob starts as sparse and its size can be up to 1 TB.
- Random Read/Write access
- Client than calls PUT-PAGE to add content to the Page Blob. PUT-PAGE operation writes a **range of pages** to a Page Blob. **Put-page operation must guarantee atomicity.**
- Calling Put Page with the **Update** option performs an in-place write on the specified page blob. Any content in the specified page is overwritten with the update.
- Calling Put Page with the **Clear** option releases the storage space used by the specified page. Pages that have been cleared are no longer tracked as part of the page blob.
- Each range of pages submitted with Put Page for an update operation may be up to **4 MB in size**. The start and end range of the page must be aligned with **512-byte boundaries**

Azure Blobs Object API : See MSDN for details

- Common Blob Operations

  Put Blob, Get Blob, Get/Set Blob Properties, Get/Set Blob Metadata, Lease Blob, Snapshot Blob, Copy Blob, Abort Copy Blob, Delete Blob

- Operations on Block Blobs

  Put Block, Put Block List, Get Block List

- Operations on Append Blobs

  Append Block

- Operations on Page Blobs

  Put Page, Get Page Ranges

Block Blob Object Design Challenges on Traditional File System

- Why not implement Block Blobs as file objects?
  - Isolation/atomicity and unique composition requirements are the key offenders.
  - Lack of any form of acceptable atomicity support on NTFS.
    - Rename/Create path on file system is prohibitively expensive
  - API semantics does not map to files, immutable vs. random access
  - Enumeration & Rich Metadata operations requires Index and DB, and transactions.
  - Namespace should be in database, but not in file system, to meet scale demands, and other requirements
  - Kernel mode filter driver complexity. Need for Index & Transaction support

ACS Service Design Principles

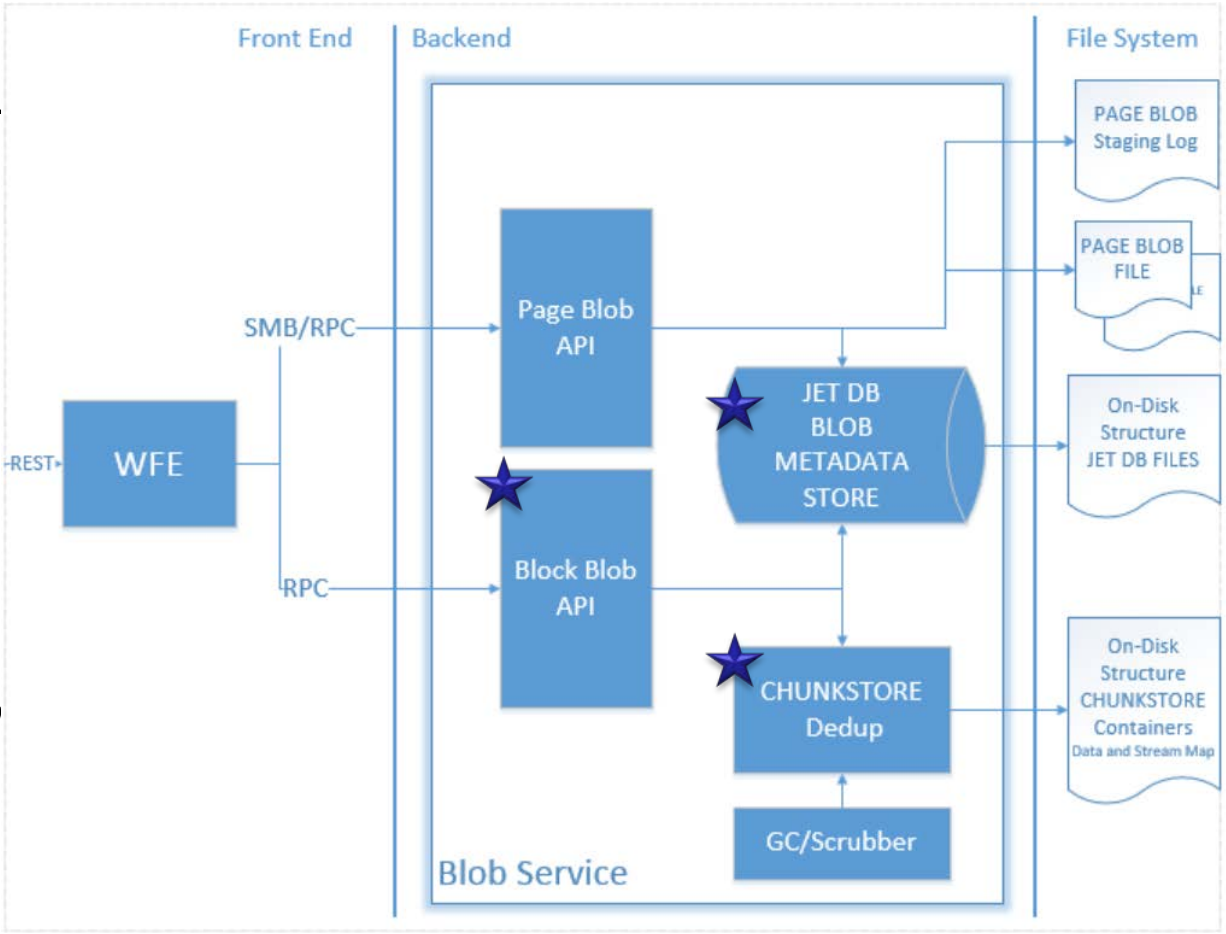
- Keep it simple
  - Achievable limit in V1.
- More than API consistency
- Build on available technologies and not reinvent the wheel
  - Depend on SOFS (with ReFS, Storage Spaces Direct), CSVFS
  - Use ESE/NT
  - Leverage Dedup Chunk Store
  - ServiceFabric/WSFC for high availability, scaling out and load balancing.

Key Design Decisions for Blob Service

- ReFS for Page Blobs, Snapshots/Extend Cloning,
  - 4 MB atomic write
- Page Blob and Block Blob share the same service.
  - Share the DB/Metadata
  - Block and page blobs are under same container
- Page Blobs as files
  - Highly optimized data path for Hyper-V over SMB
- Block Blobs as ChunkStore Objects
  - Block Blobs are not files, but immutable chunks
- RPC as data transport from WFE to Blob service
  - WFE cannot write directly to ChunkStore

Block Blob Service Design

- **Global namespace** exists in the DB.
- **User mode only access**/store. No file system access to the block blobs; neither for namespace nor for data.
- **Use ChunkStore** implementation to store committed and uncommitted blocks, and the stream maps.
- **Design for Azure Parity** from get go.
- Azure blob metadata is stored in ESE/NT DB.
  - To optimize metadata only operations
  - To implement Blob API GC semantics.
- DB Scope is set of containers and their blobs metadata.



Why ChunkStore?

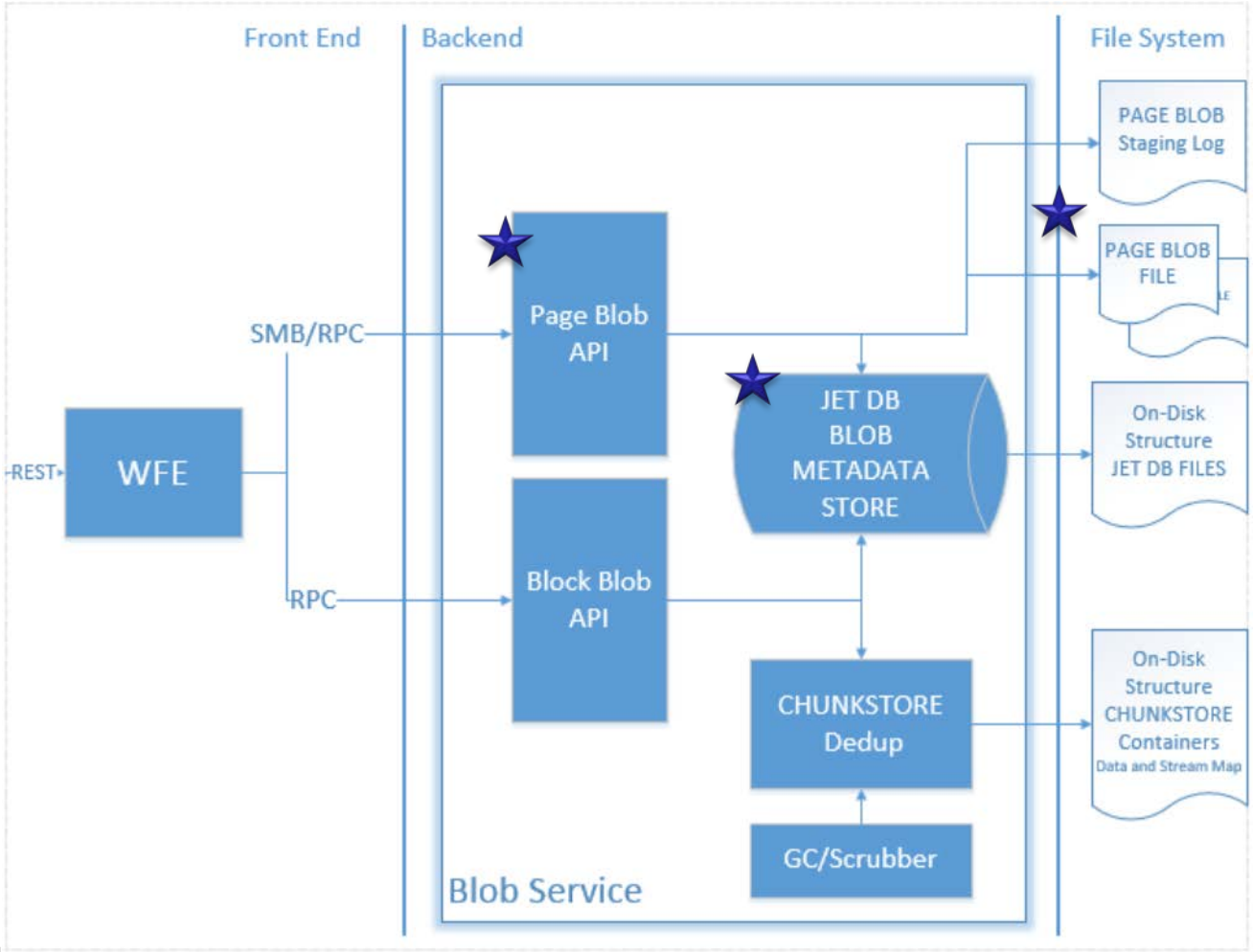
- Immutable file containers for chunk and stream maps.
- Append Only Chunk Insertion design
- Part of Deduplication Feature, proven.
- Integrity checks for detecting page corruptions, and read from a replica for in place patching.
- Shallow and Deep Garbage Collection.
- Data chunks shared by root blob and snapshots (deduplication)
- Guarantee crash-consistent file commit (Precise order of operations guarantees data integrity at each stage)
- Various chunk size support (up to 4MB)
- Efficient self contained chunk id referencing

Why ESE/NT?

- **ESE**
  - Extensible Storage Engine (ESE), also known as JET Blue, is an ISAM (Indexed Sequential Access Method) data storage technology. It provides transacted data update and retrieval
- **Transactions & Index**
  - The ESE database engine provides Atomic Consistent Isolated Durable (ACID) transactions.
- **Logging and crash recovery**
  - ESE has write ahead logging and snapshot isolation for guaranteed crash recovery.
  - The application-defined data consistency is honored even in the event of a system crash.
- Good **Backup/Restore** support
  - ESE supports on-line backup where one or more databases are copied, along with log files in a manner that does not affect database operations
- **Page corruption detection** and read from replica and patch in place (FSCTL_MARK_HANDLE / MARK_HANDLE_READ_COPY)
  - Automatic **DB scan** and corruption detection
  - ESENT engine self-detecting and auto-correcting checksum errors on a Jet database stored on a Spaces triple-replica remotely accessed via SMB/CSVFS
- **Used at scale** in exchange workloads.

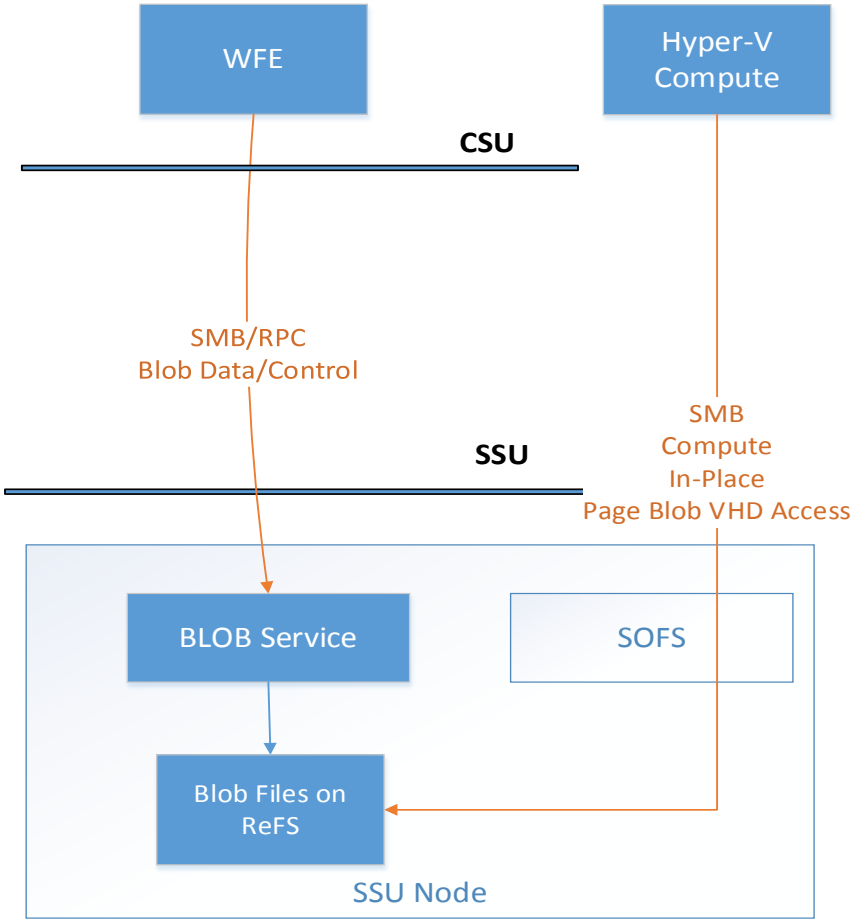
Page Blob Service Design

- **Global namespace** exists in the DB.
- **Page blobs are files** stored in ReFS volume.
- Blob is a linear mapping to the file. There is no stream map.
- **Page blobs also use DB** to store azure blob metadata.
- To enable exclusive in-place access (direct via SMB) for Hyper-V, check-out, and check-in semantics supplied.
- No concurrent REST vs. File System access.



Page Blobs - Compute vs. REST data access path

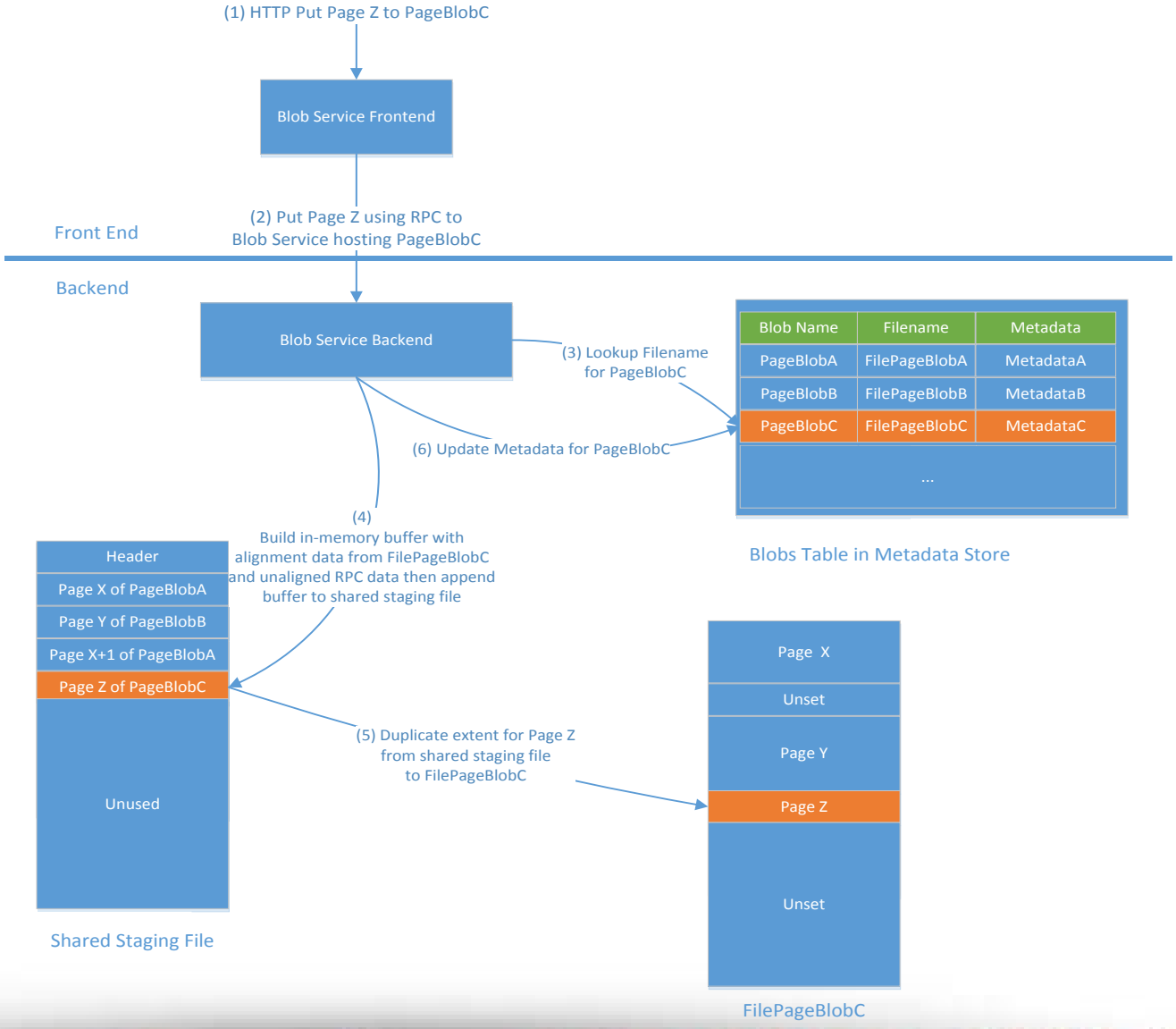
- Blob REST access is via the Blob service.
- Compute page blob access (Hyper-V accessing VHDs) is direct to the file over SMB, same path as today with RDMA etc.
- Once a blob is "checked out" for compute access, it is not accessible through REST.



Blob Service Design Data Flow & Representation

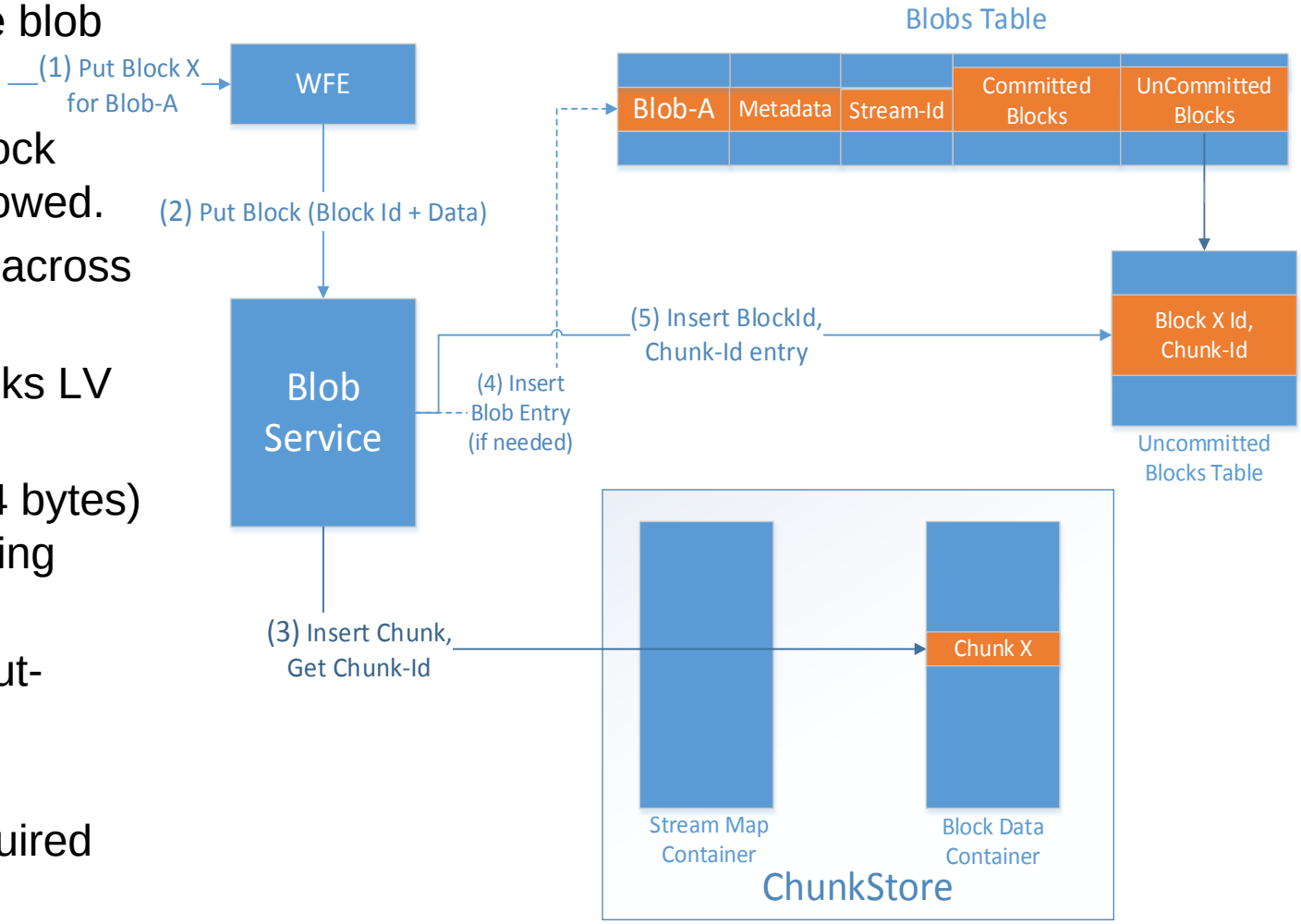
PAGE Blob Design on ReFS

- Single Log File per Volume.
- Append-only writes to staging log with control
- ReFS duplicate extents
- Final Commit call updates metadata
- Crash Consistency via staging log write, ReFS extent duplication.
- Check-pointing to manage space consumption/ and recovery.



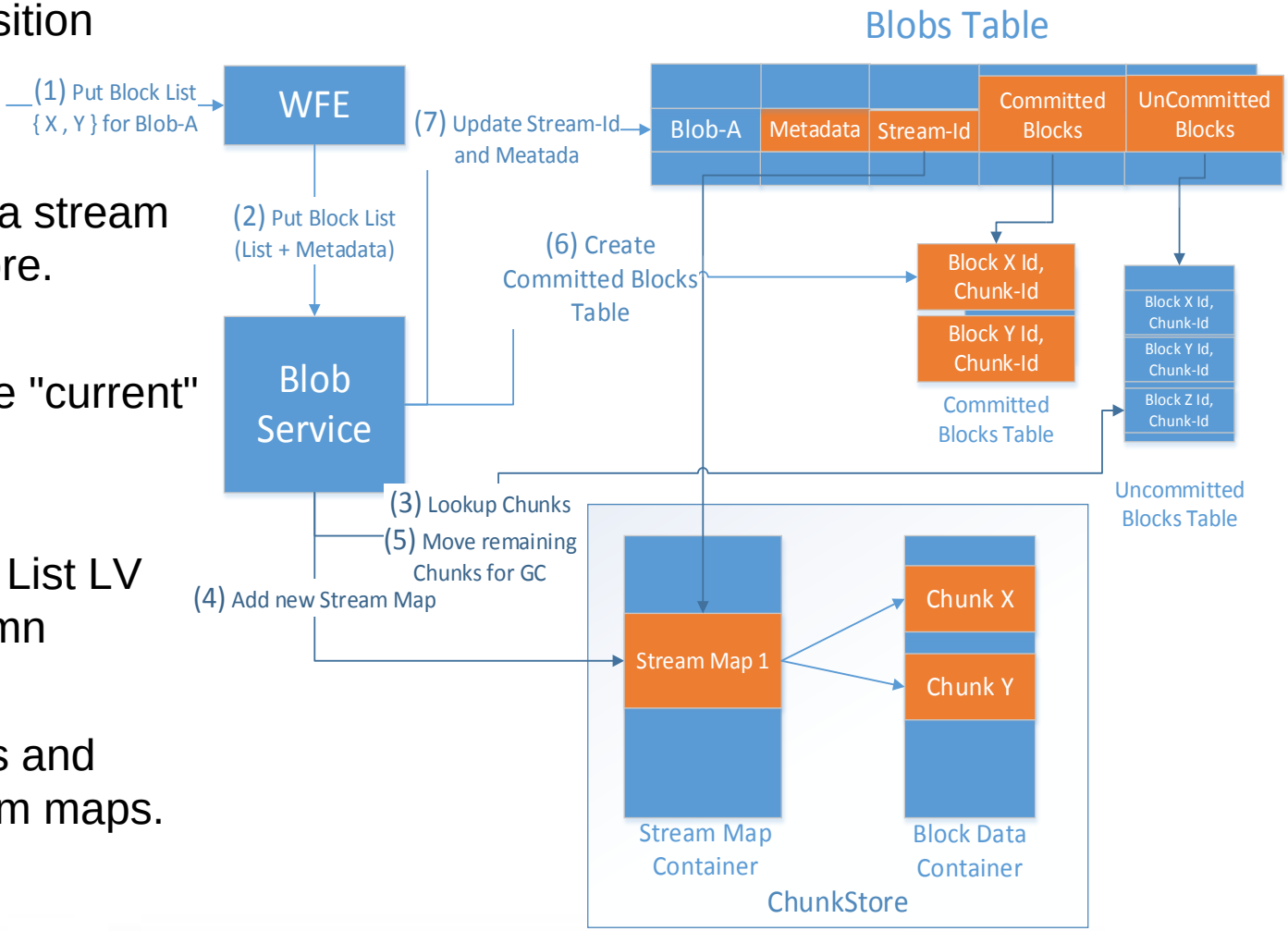
BLOCK BLOB – PUT BLOCK

- No change in composition of the blob
- Concurrent put block operations are allowed.
- No Block Sharing across different Blobs
- Uncommitted blocks LV Column in DB
- Azure Block Id (64 bytes) to Chunk Id mapping
- Efficient search uncommitted at Put-Block-List (or at recovery)
- For GC policy required by the Blob API.



BLOCK BLOB – PUT BLOCK LIST

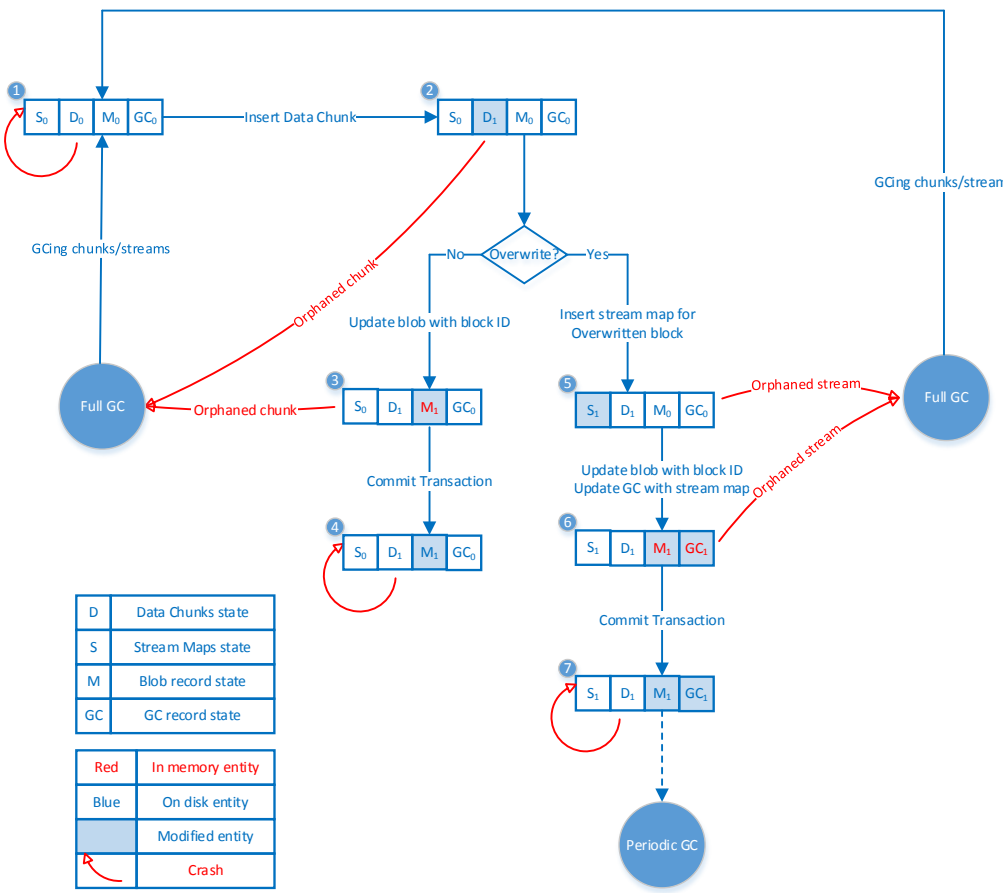
- Put Block List modifies metadata/composition
- Allocates/Inserts a stream map to ChunkStore.
- Blob entry has the "current" stream map ID.
- Committed Block List LV (long value) column
- Snapshots clones and inserts new stream maps.



Blob Service Crash Consistency

Example Crash Consistency Approach

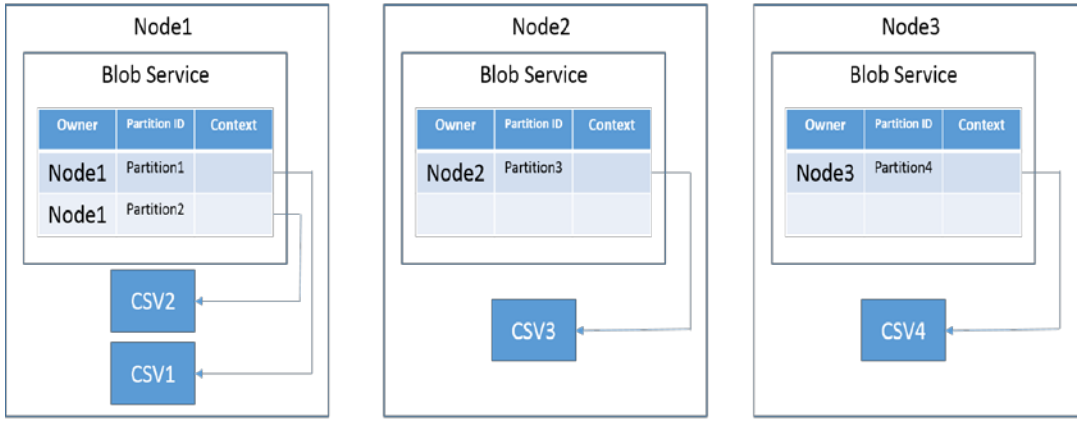
State	Crash just before normal action	Normal action	Coverage ID
1. The state consists of a valid metadata blob record that might have a pre-existing blob record or not (M_0), valid stream map state that might have a stream for the existing blob or no stream in case the blob doesn't exist (S_0) and a valid GC metadata table that has no records for related to this blob (GC_0).	We remain in this state.	If the operation conditions succeed, then insert the new data chunk into the chunk store. This takes us to state 2. Otherwise, if the operation conditions fails, then remain in this state	1
2. The state consists of a valid metadata blob record (M_0) that is not yet updated with the valid inserted data chunk ID (D_1).	On demand full GC. For more details about the full GC, please refer to the full GC crash consistency section.	If there is no uncommitted block with the same block id, then begin transaction to update the metadata store with the newly inserted data chunk id. This takes us to state 3. Otherwise, if there is an uncommitted block with the same block id, then create a stream map for it. This takes us to state 5.	If blob exists: 2, 301 Else 2, 100
3. The state consists of in memory metadata update for the blob record (M_1) that is updated with the valid data chunk id (D_1).	On demand full GC. For more details about the full GC, please refer to the full GC crash consistency section.	Commit the DB transaction. This takes us to state 4.	101, 102
4. The state consists of a valid metadata update for the blob record (M_1) that is updated with the valid inserted data chunk id (D_1).	Will remain in this state.	This is a successful terminal state.	103



Blob Service HA Model

Blob Service HA model on WSFC & CSVFS

- Multiple instances, one on each physical machine.
- Blobs are stored in a cluster file system (CSVFS).
- Blob namespace is partitioned among the blob service instances.
- Each Blob Service maintains a partition mapping table.
- CSV volumes can move between nodes to remain highly available.
- A Blob client maintains a mapping of the partition ID to the node name on which the partition is hosted.
- The mapping can change due to node failover and CSV failover.



Container ID	Partition ID
Container1	Partition3
Container2	Partition2
Container3	Partition4

Container ID	List of Blob IDs
Container1	Blob1,Blob2
Container2	Blob3,Blob4
Container3	Blob1,Blob3

Table Service

Azure Table Semantics See MSDN for details

- **Data Model**: Each row (entity) contains up to 1MB of **schema-less** data.
  - Each entity contains up to 252 properties including 3 system properties
  - **PartitionKey** + **RowKey** form the primary key for data query in Tables.
  - Support filter, LINQ queries and pagination for retrieving table entities.

PartitionKey	RowKey	TimeStamp	Status	…
A	Alice	May 29, 2016	"Online"	…
B	Brian	June 29, 2016	"Offlinee"	…

- **Atomic**: Support both entity-level transactions as well as Group Entity Transaction in one partition with maximum 100 entities.
- **Consistent**: After a successful write, any reads that happen after the write get the latest value.
- **Durable**: Synchronously store 3 replications before reporting success.
- **Scalable**: Millions of entities in a single table. Millions of tables in the whole system.
- **Highly Available**: 99.9% read/write for local replication

Azure Table API : See MSDN for details

- Common Table Operations

  Query Tables, Create Table, Delete Table, Get Table ACL, Set Table ACL

- Operations on Table Entities

  Query Entities, Insert Entity, Insert Or Merge Entity, Insert Or Replace Entity, Update Entity, Merge Entity, Delete Entity

Table Backend

Table master creates and keeps table ranged partition to Table Server instance mapping in Azure Service Fabric reliable collection, which is replicated across the cluster.

- Mapping also cached in FE to expedite lookup.

One **Table Server instance** serves one DB including multiple user tables

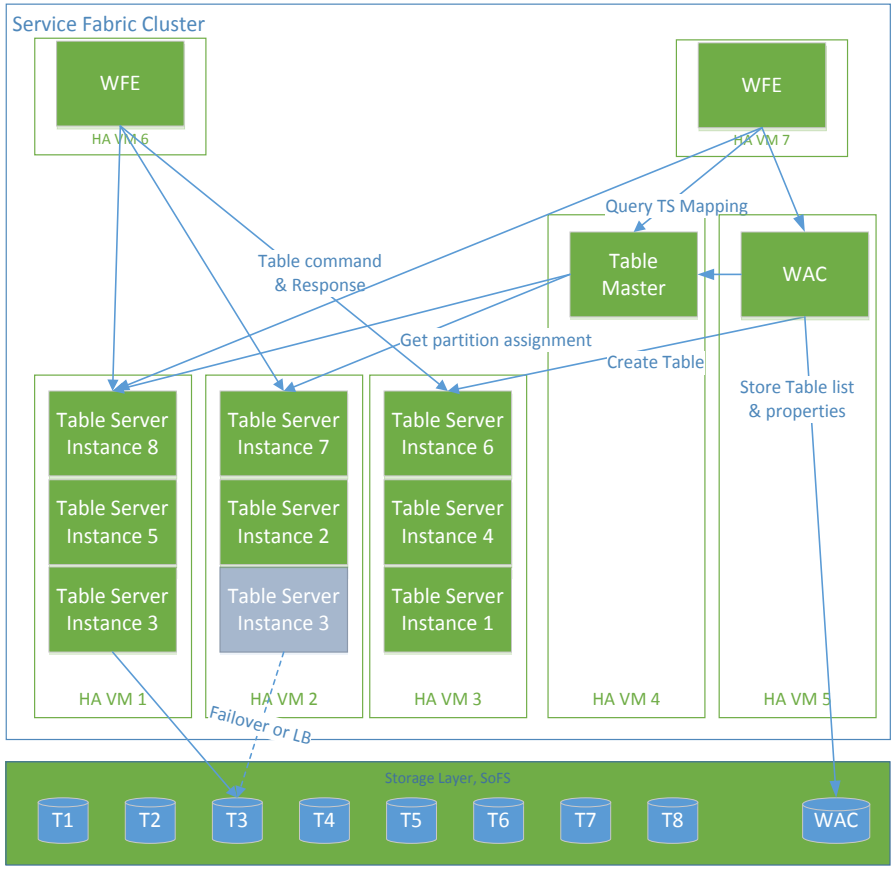
TableName = “CCADB32B-3955-41B8-B28B-062EE8791EE9"

PartitionKey	RowKey	TimeStamp	PropertyBag
C&E	100000	2016/08/29	Project = “ACS” …

Multiple TS instances share one process

TS instances acquire assigned database/ partition ranges from table master upon start

Use Service Fabric to achieve **High Availability, Scale-out,** and **Load Balancing**



Questions?

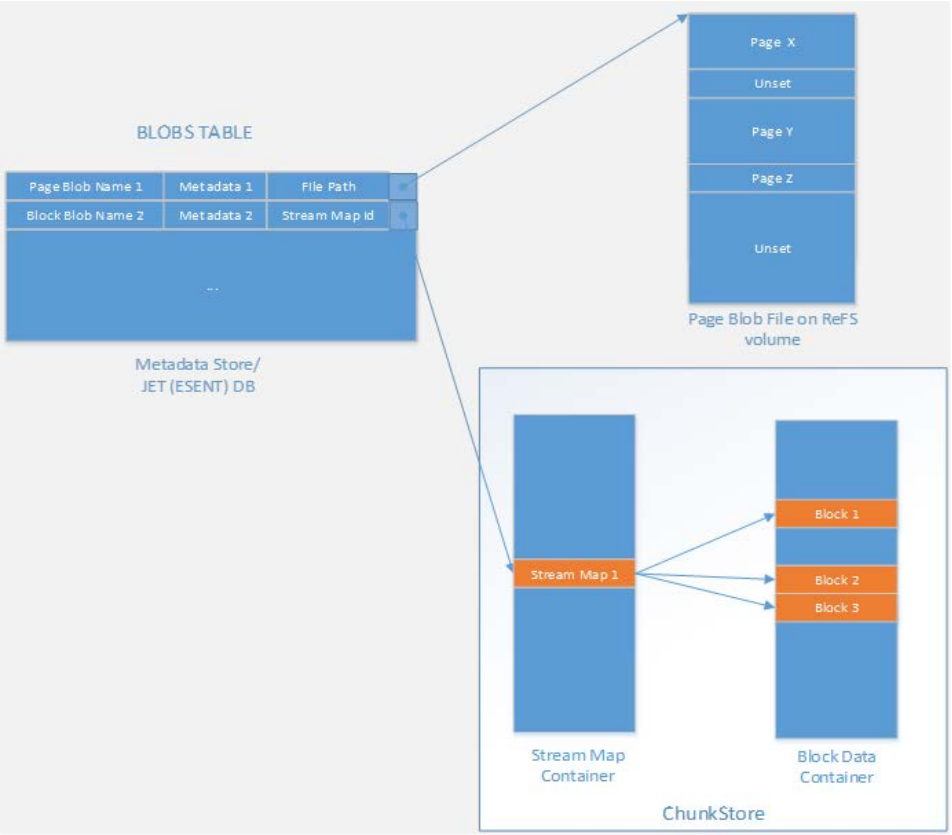
Appendix

Differences against Azure Storage

- No Standard_RAGRS or Standard_GRS
- Premium_LRS: no performance limits or guarantees
- No Azure Files yet
- Usage meter differences
  - No IaaS transactions in Storage Transactions
  - No internal vs. external traffic distinction in Data Transfer
- No account type change, or custom domains

Blob Service Metadata Persistence Model

- Stream Maps being evaluated to move to DB
- Single table with mixed row/blob types

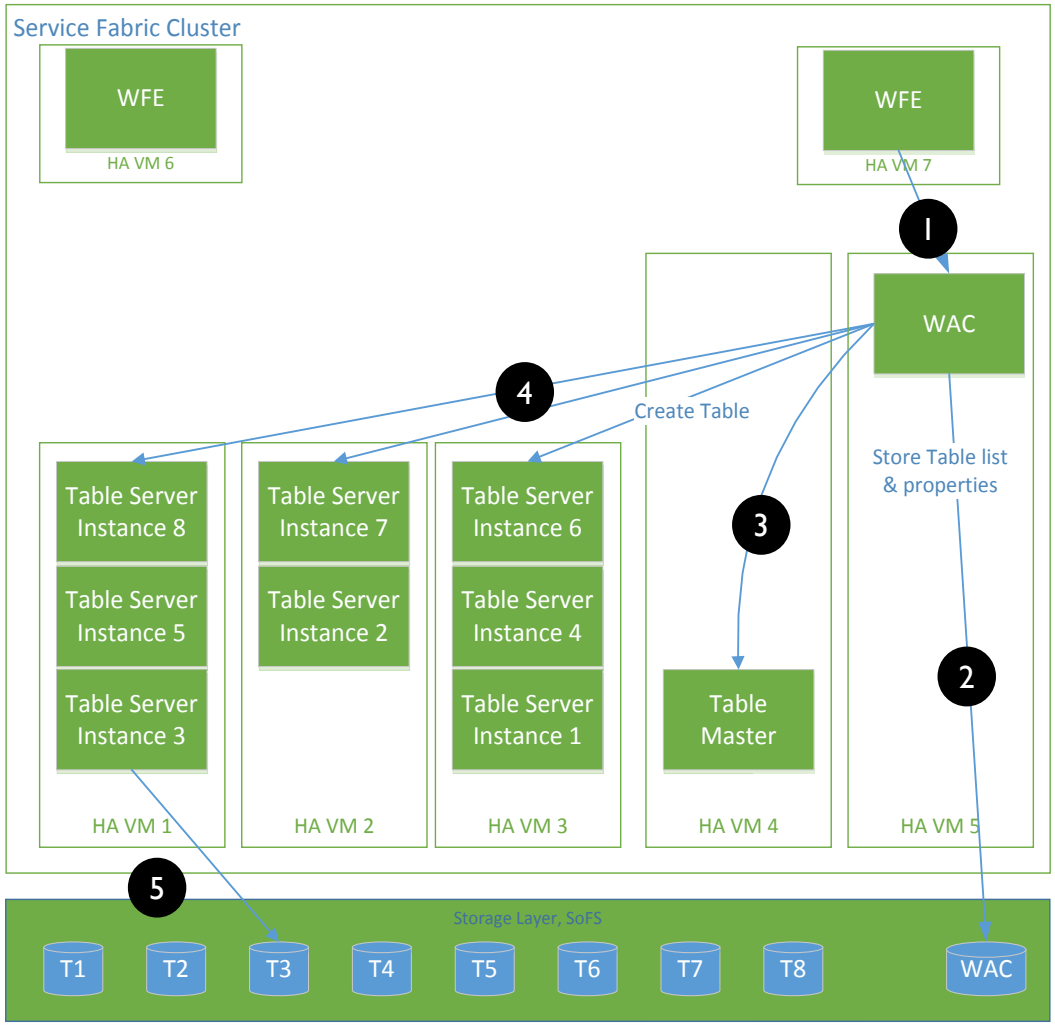


ACS HA Model

- Service Fabric is used by WFE, TM/TS, WAC, SRP instances for failover, load balance, deployment/upgrade .
- All ACS service roles are co-locatable.
- SRP collocates with other RPs in the same cluster.
- Compute cluster VM health monitoring complements in-guest Service Fabric by providing VM-level recovery.
- Blob service is running as an HA service on the SSU cluster directly on physical nodes, as a peer of SMB. It monitors CSV movements in the cluster and attach/detach to them.
- Blob service does not failover. It is multiple active configuration.

Create Table Data Flow

1. WFE authenticates with Account service (WAC) and sends Table Creation request to WAC
2. WAC adds a new table entry in WAC metadata DB
3. WAC queries Table Master for corresponding Table Server Instance
4. WAC requests Table Server to create the new table
5. TS instance creates the user table in the DB



Insert Entity Data Flow

1. WFE authenticates & authorizes requests with WAC (if not already cached by WFE)
2. WFE queries Table Master for the TS instance for the table (if it's not already cached by WFE)
3. WFE sends Insert Entity request to specified TS instance
4. TS instance updates DB containing the corresponding table and sends success/failure back to WFE