

# Inside Azure Storage

# Yves Goeleven

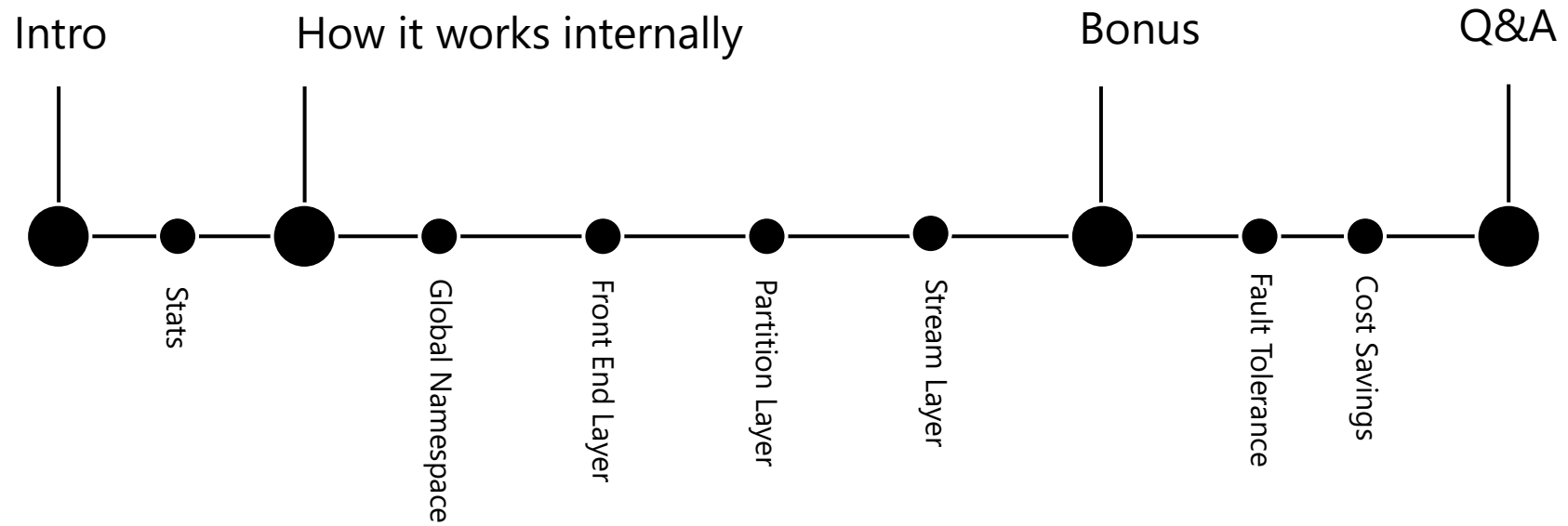


Solution Architect - Particular Software

- NServiceBus
- Shipping software since 2001
- Azure MVP since 2010
- Co-founder & board member AZUG
- MessageHandler & Clubmanagement.io

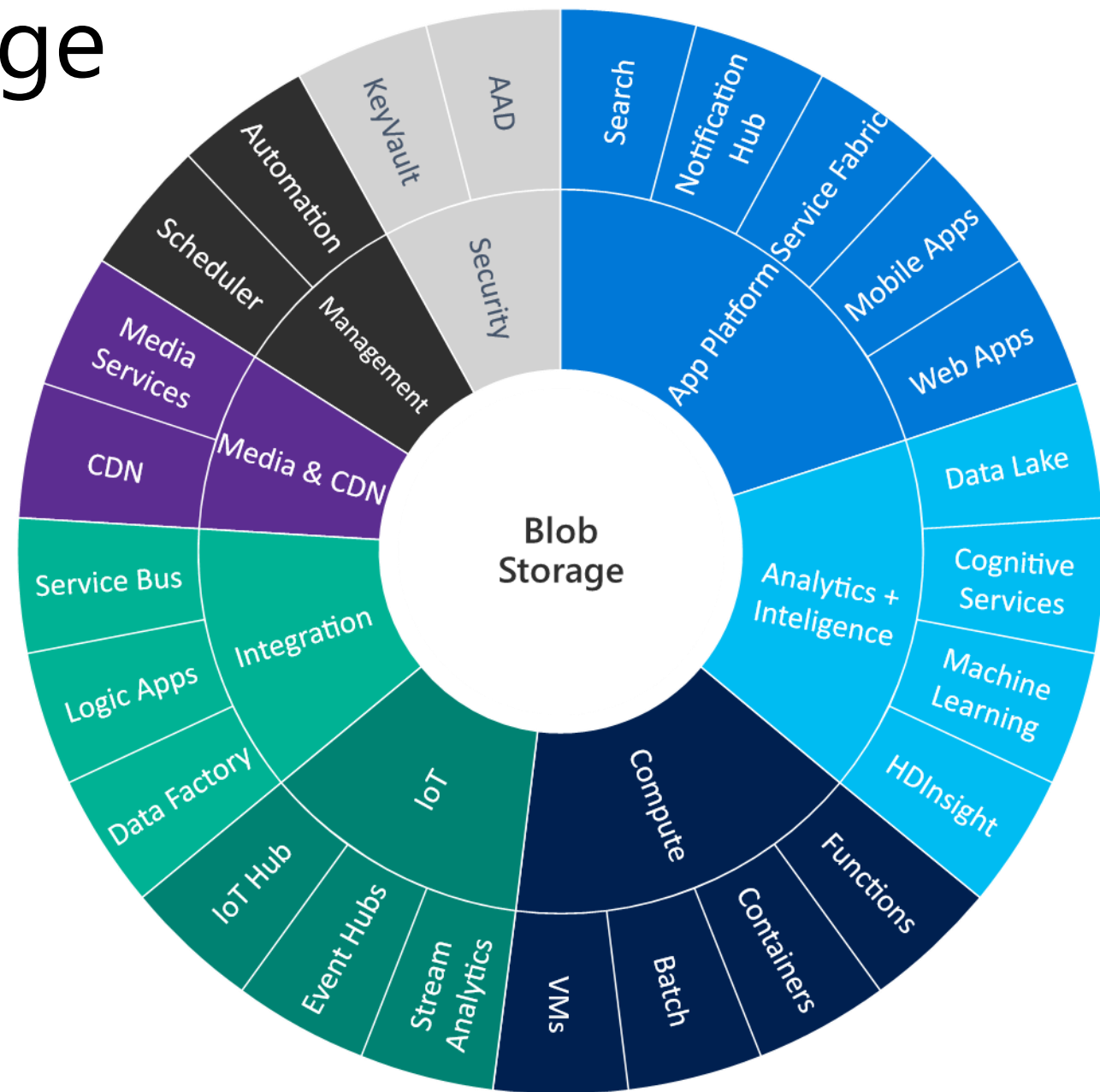
# Agenda

## Deep Dive Architecture



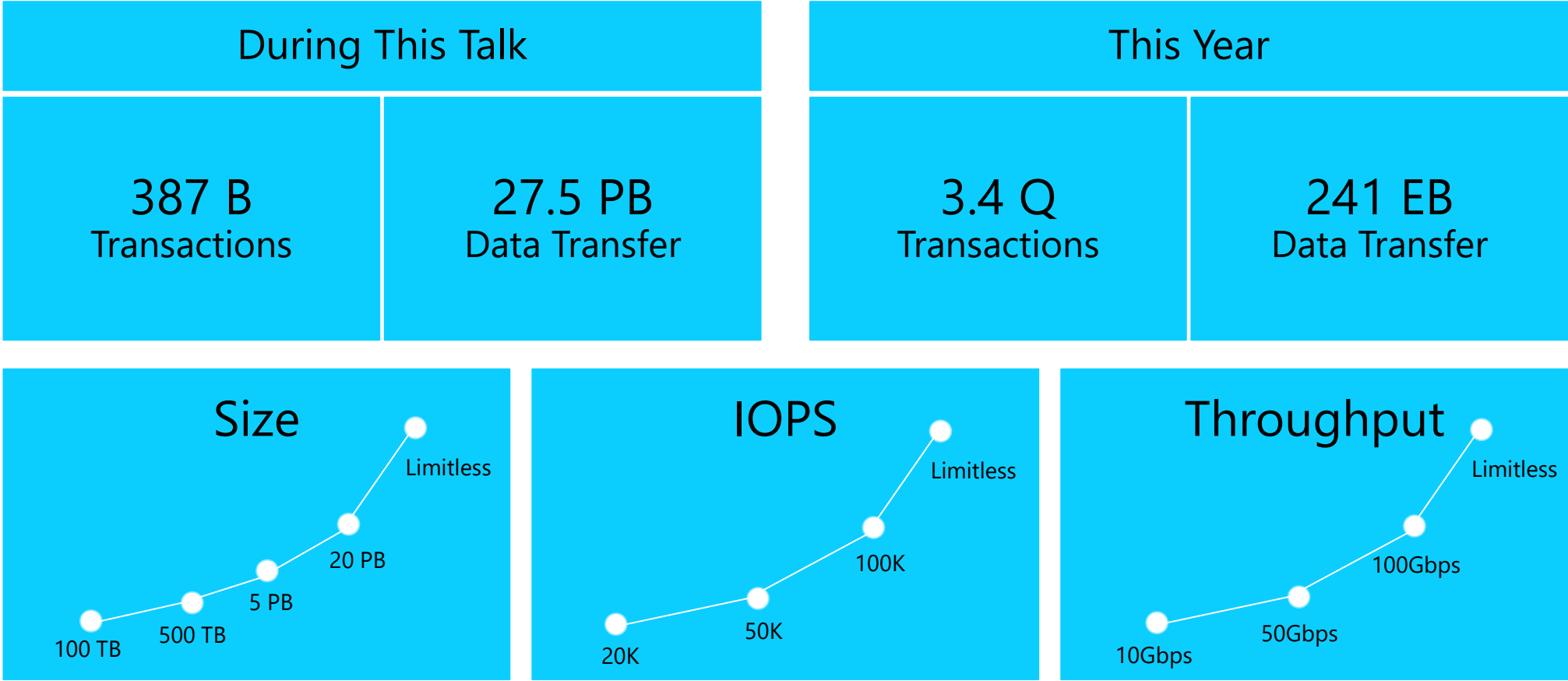
# Blob Storage

Core service



# Statistics

Pretty impressive



Let's dive in!

Global  
namespace

## Internal Architecture

Front End  
Layer

Partition  
Layer

Stream  
Layer

# Global namespace

Every request gets sent to

- `http(s)://AccountName.Service.core.windows.net/PartitionName/ObjectName`



# Global namespace

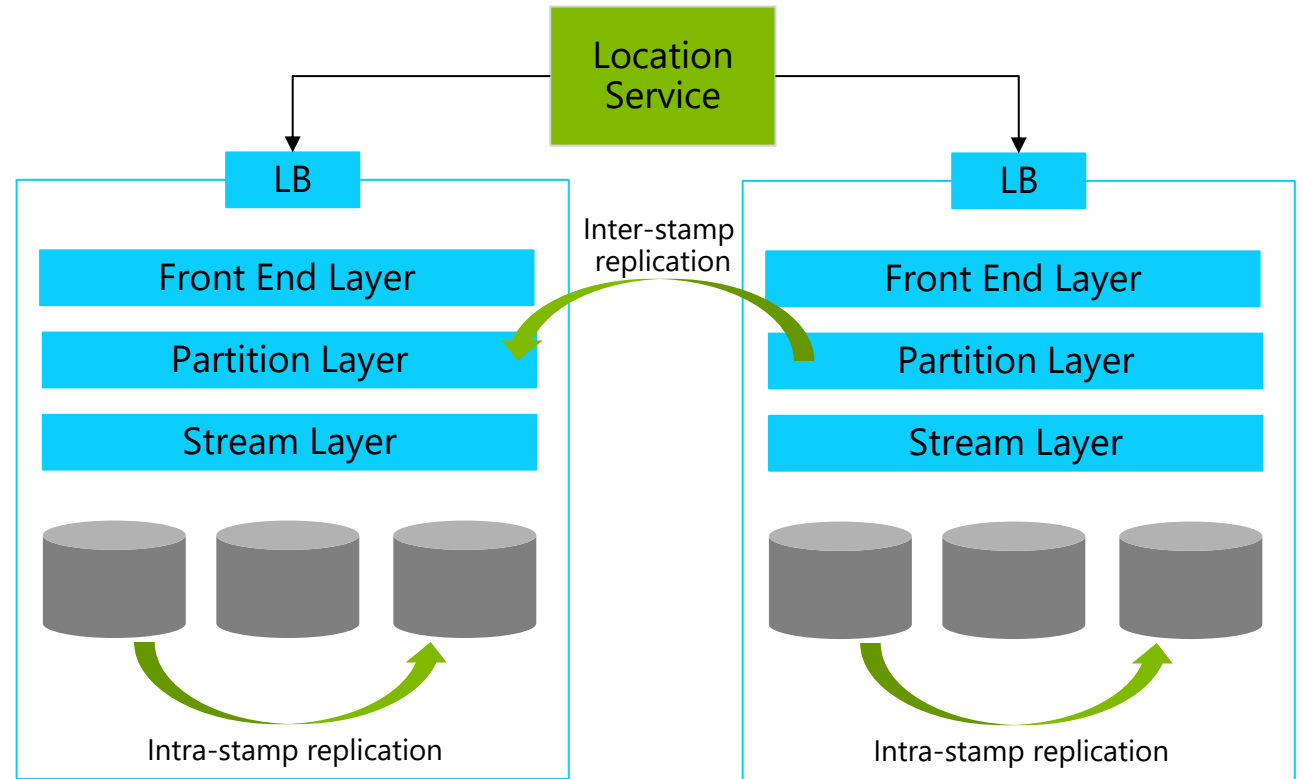
## DNS based service location

- `http(s)://AccountName.Service.core.windows.net/PartitionName/ObjectName`
- DNS resolution
- Datacenter (f.e. Western Europe)
- Virtual IP of a Stamp

# Stamp

## Scale unit

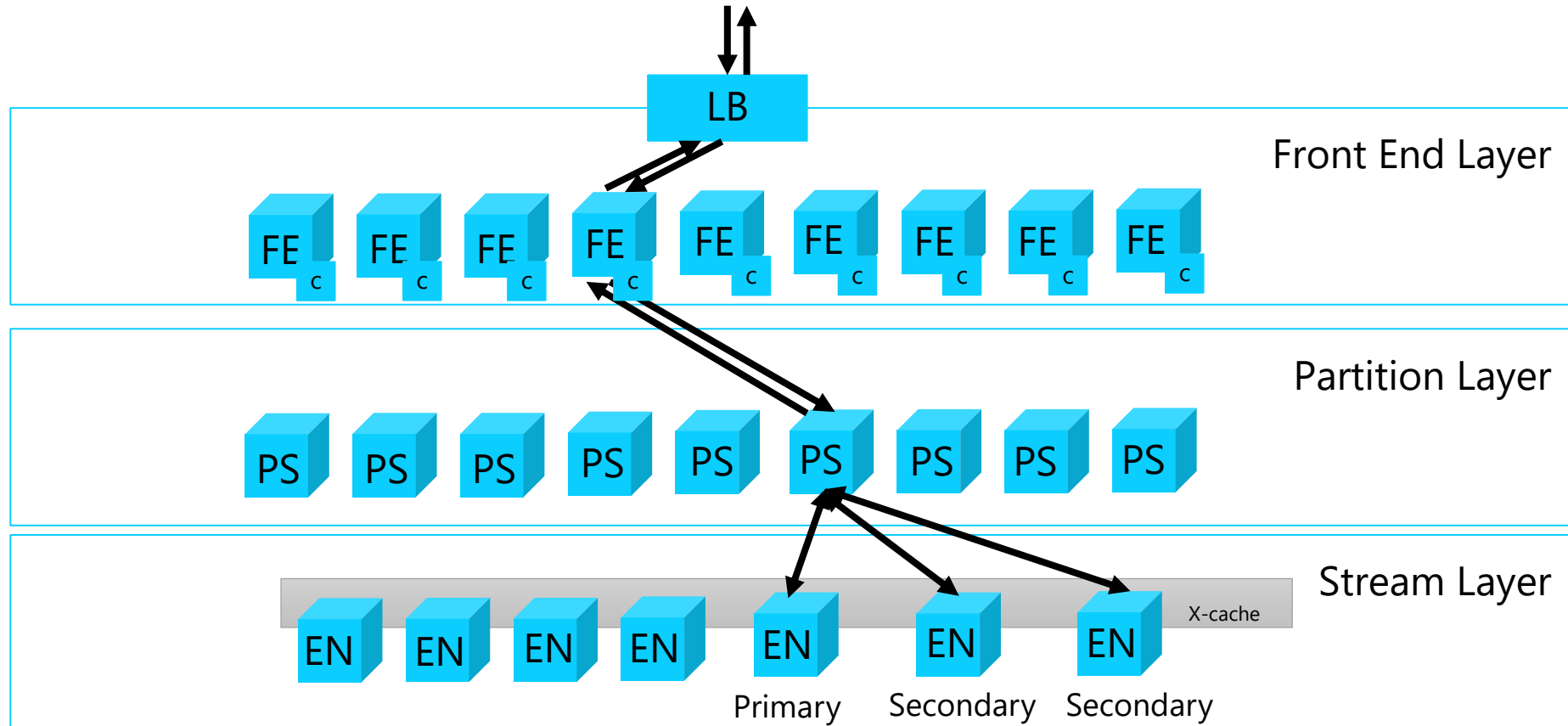
- A windows azure deployment
- 3 layers
  - Typically 20 racks each
  - Roughly 360 XL instances in total
- Instances with disks attached
  - 30 Petabyte
- Durability Models
  - LRS: 3 replica's, 1 region, 1 zone
  - ZRS: 3 replica's, 1 region, 3 zones
  - GRS: 6 replica's, 2 regions, 1 zone
  - RA-GRS: GRS + read access second



# Incoming read request

## Arrives at Front End

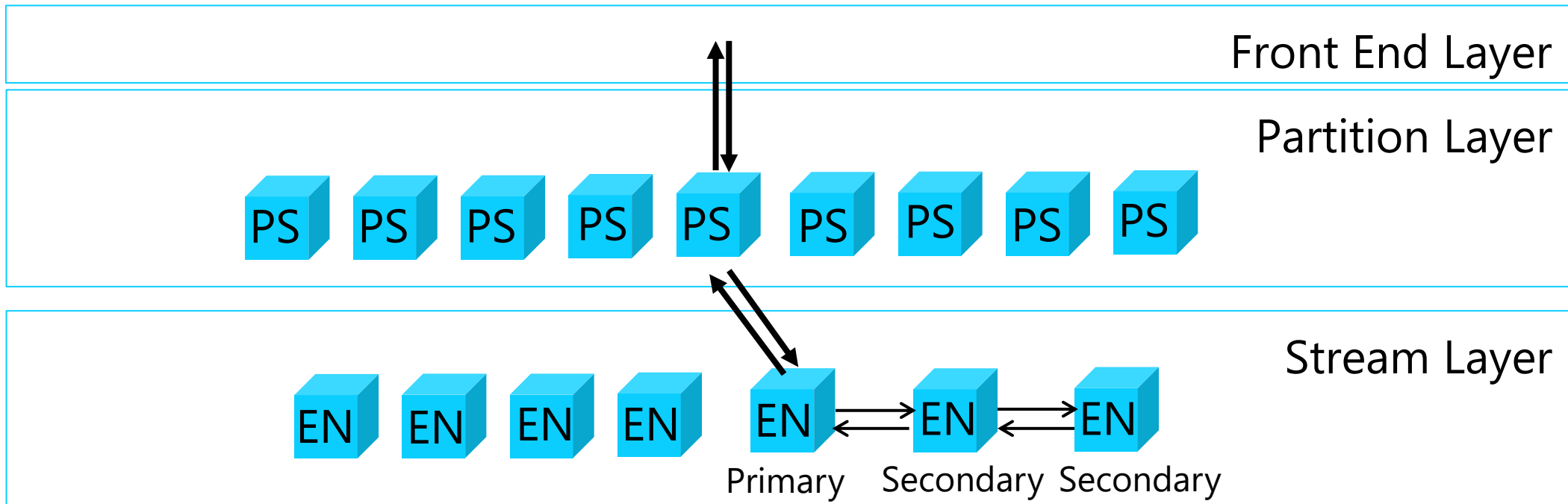
- Routed to partition layer,  
`http(s)://AccountName.Service.core.windows.net/PartitionName/ObjectName`
- Reads usually from memory or cache, if not: parallel read



# Incoming Write request

## Written to stream layer

- Synchronous replication
- 1 Primary Extend Node
- 2 Secondary Extend Nodes



## Internal Architecture

Global  
namespace

Front End  
Layer

Partition  
Layer

Stream  
Layer

# Front End Nodes

## Stateless role instances

- Authentication & Authorization
  - Shared Key (storage key)
  - Shared Access Signatures
  - Oauth
  - AAD
  - ACL's
- Service Endpoints (Firewall)
  - IP range, vnet, ...
- Throttling
- Routing to Partition Servers
  - Based on PartitionMap (in memory cache)
- Versioning
  - x-ms-version: 2011-08-18
- Tiering: x-ms-access-tier, x-ms-archive-status
  - Premium
  - Hot
  - Cold
  - Archive



Front End Layer

# Partition Map

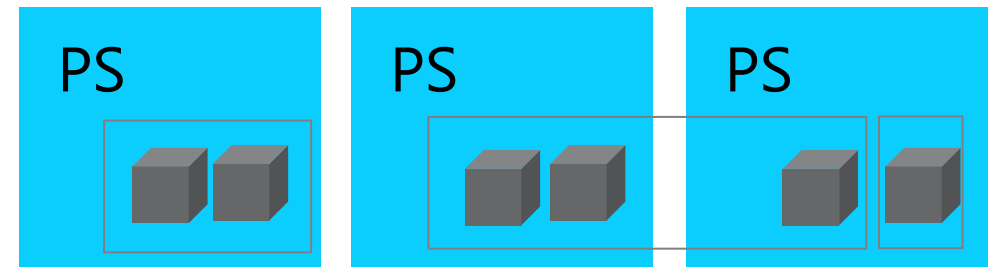
## Determine partition server

- Used By Front End Nodes
  - Managed by Partition Manager (Partition Layer)
- Ordered Dictionary
  - Key:
    - `http(s)://AccountName.Service.core.windows.net/PartitionName/ObjectName`
    - (Block token)
  - Value: Partition Server
- Range Partitioning
  - Split across partition servers
  - Designed for lots of small blobs, records or messages
- Token based Partitioning
  - Designed for large blobs

# Range Partitioning

## Practical Example

- Each blob is a partition
- Blob uri determines range
- Range matters for listing -> Continuation token
- Same container: More likely same range & server
  - <https://myaccount.blobs.core.windows.net/mycontainer/subdir/file1.jpg>
  - <https://myaccount.blobs.core.windows.net/mycontainer/subdir/file2.jpg>
- Other container: Less likely
  - <https://myaccount.blobs.core.windows.net/othercontainer/subdir/file1.jpg>

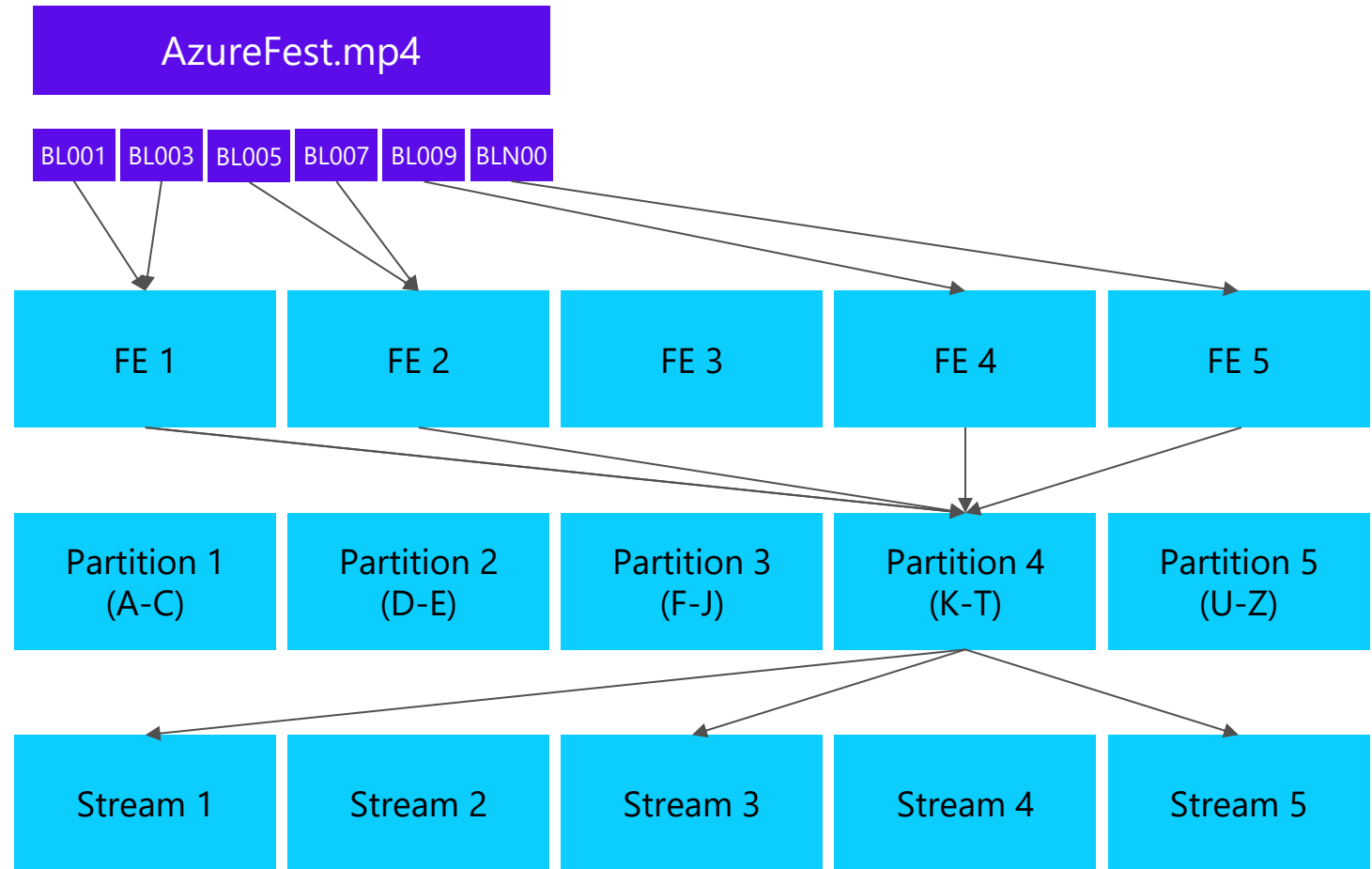




# Range Partitioning

## Large File Upload Problem

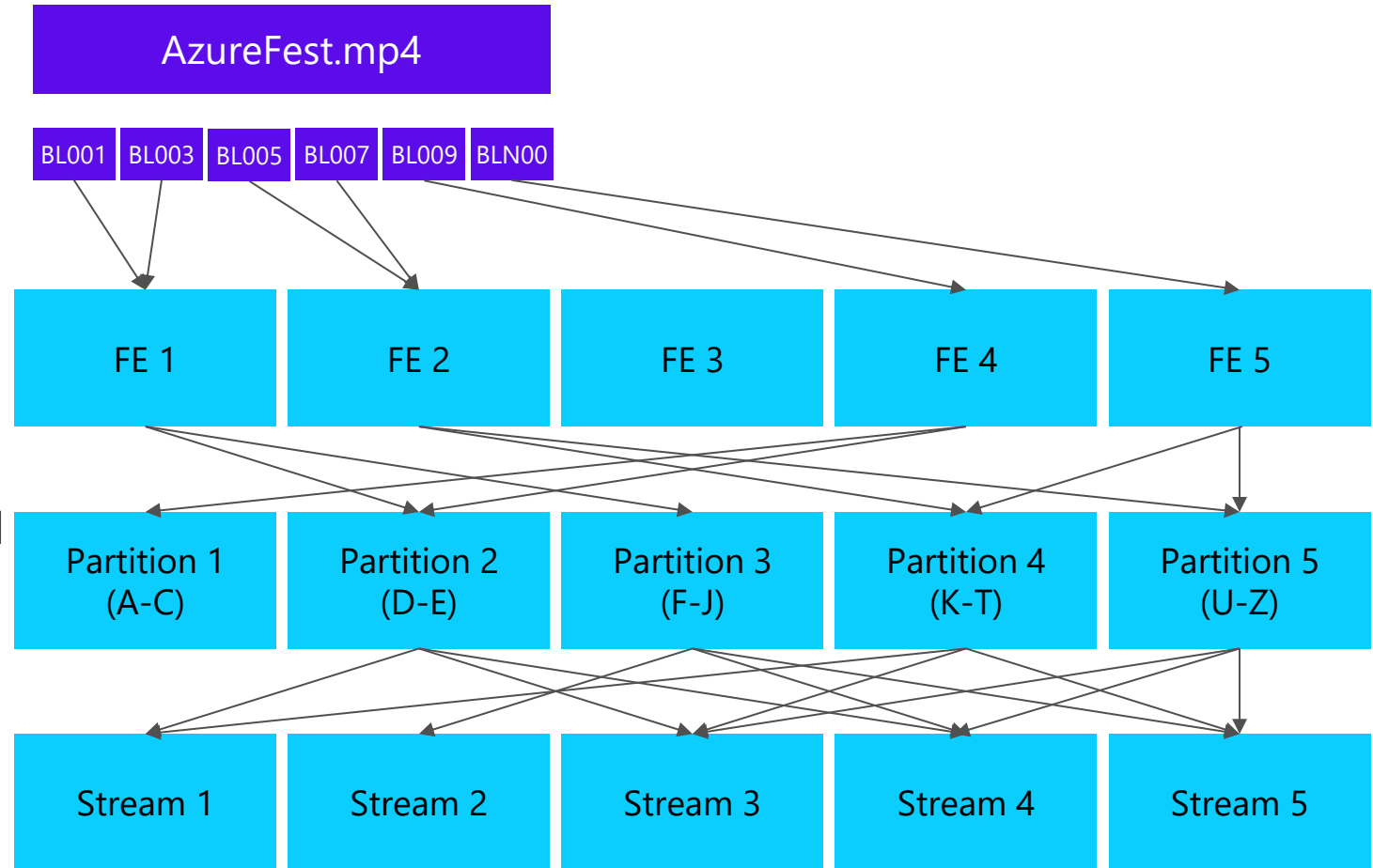
- Range partitioning was designed for table records, queue messages & small blobs.
- A single blob is served by a single partition server
- Parallel writes to FE serialize on a single partition server
- Writes to multiple streams helps, but not enough



# Token based partitioning

## Improved write performance

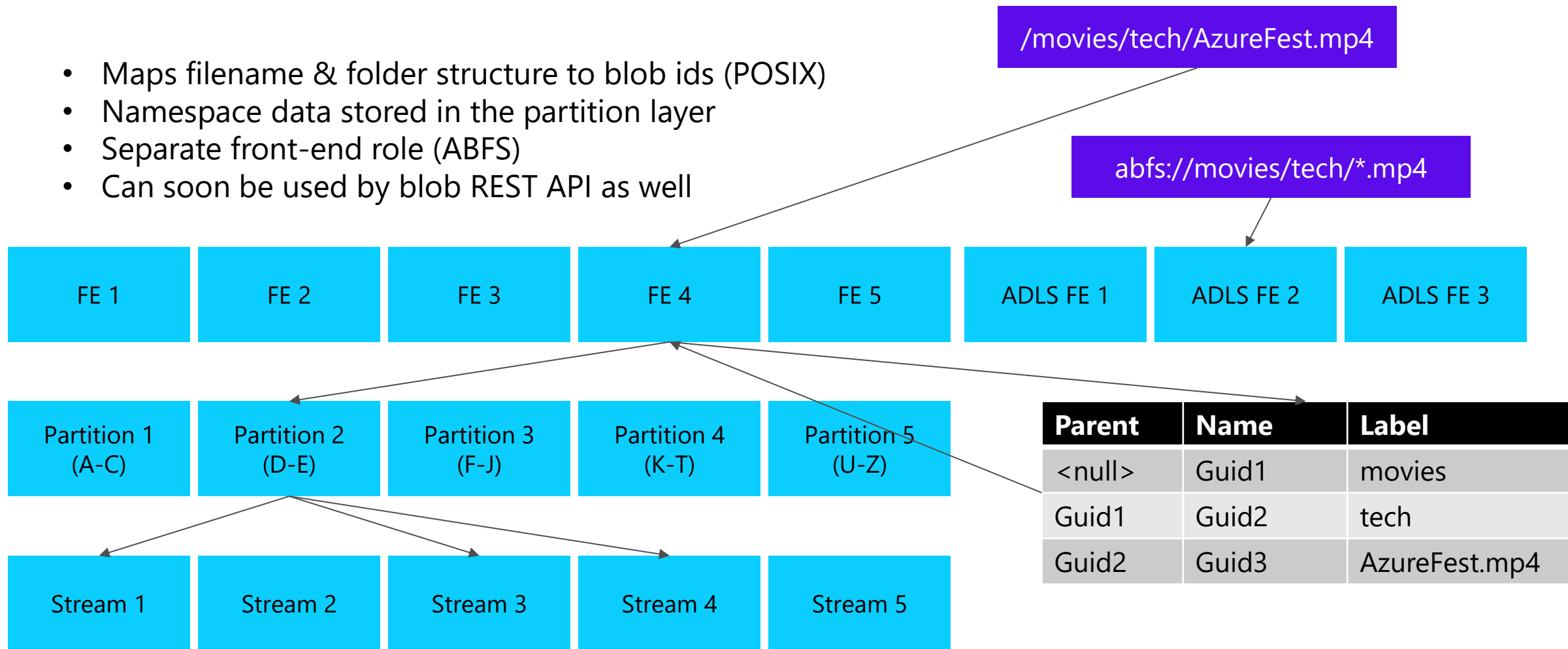
- For blobs > 4096Kb
- High throughput block blob
- Each block has a unique id assigned to it
- The block id range is pre-partitioned uniformly across all partition servers
- A single blob's writes can potentially be served by all partition servers: theoretical write throughput = Storage account's limit



# Hierarchical namespace

## Azure Data Lake Store Gen 2

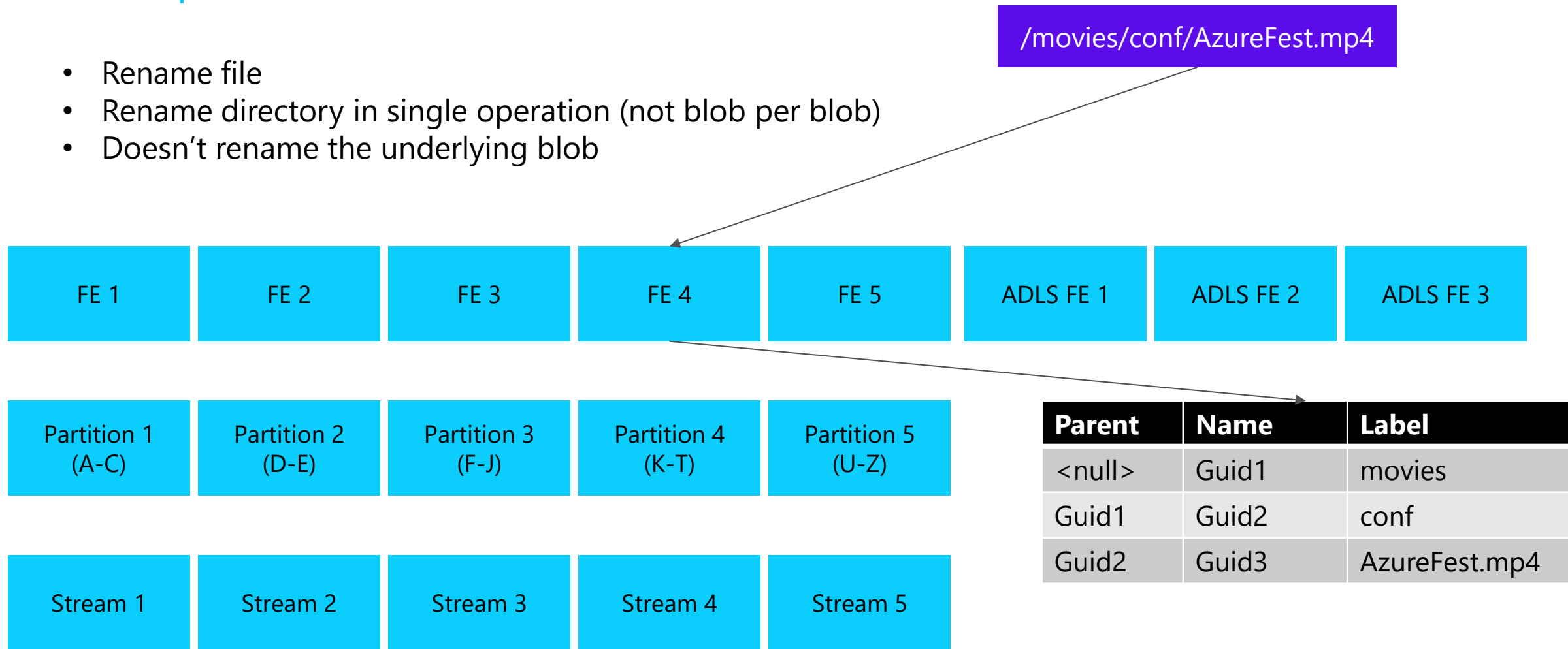
- Maps filename & folder structure to blob ids (POSIX)
- Namespace data stored in the partition layer
- Separate front-end role (ABFS)
- Can soon be used by blob REST API as well



# Hierarchical namespace

## Allows improved renames

- Rename file
- Rename directory in single operation (not blob per blob)
- Doesn't rename the underlying blob



# Internal Architecture

Global  
namespace

Front End  
Layer

Partition  
Layer

Stream  
Layer

# Partition layer

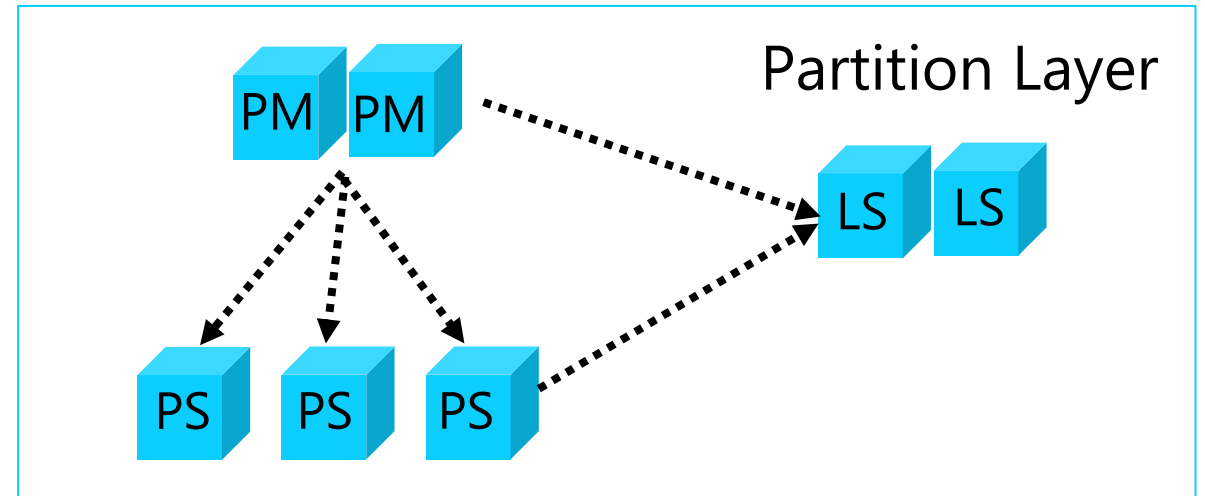
## Dynamic Scalable Object Index

- Object Tables
  - System defined data structures
  - Account, Entity, Blob, Message, Schema, PartitionMap, Namespace
- Range partitions: Dynamic Load Balancing
  - Monitor load to each part of the index to determine hot spots
  - Index is dynamically split into thousands of Index Range Partitions based on load
  - Index Range Partitions are automatically load balanced across servers to quickly adapt to changes in load
  - Updates every 15 seconds
- Token based partitions: no load balancing

# Partition layer

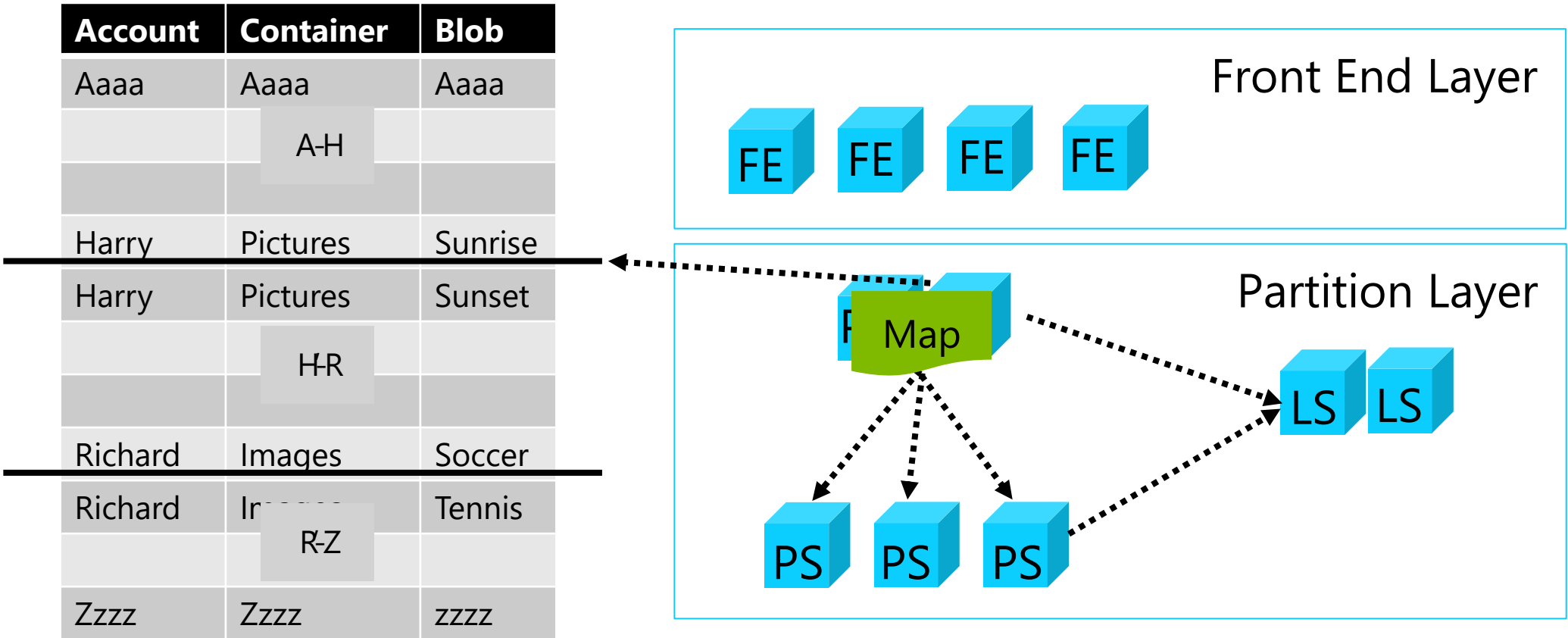
## Consists of

- Partition Master
  - Distributes Object Tables
  - Partitions across Partition Servers
  - Dynamic Load Balancing
- Partition Server
  - Serves data for exactly 1 Partition
  - Manages Streams
- Lock Server
  - Master Election
  - Partition Lock



# Dynamic Scalable Object Index

## How it works

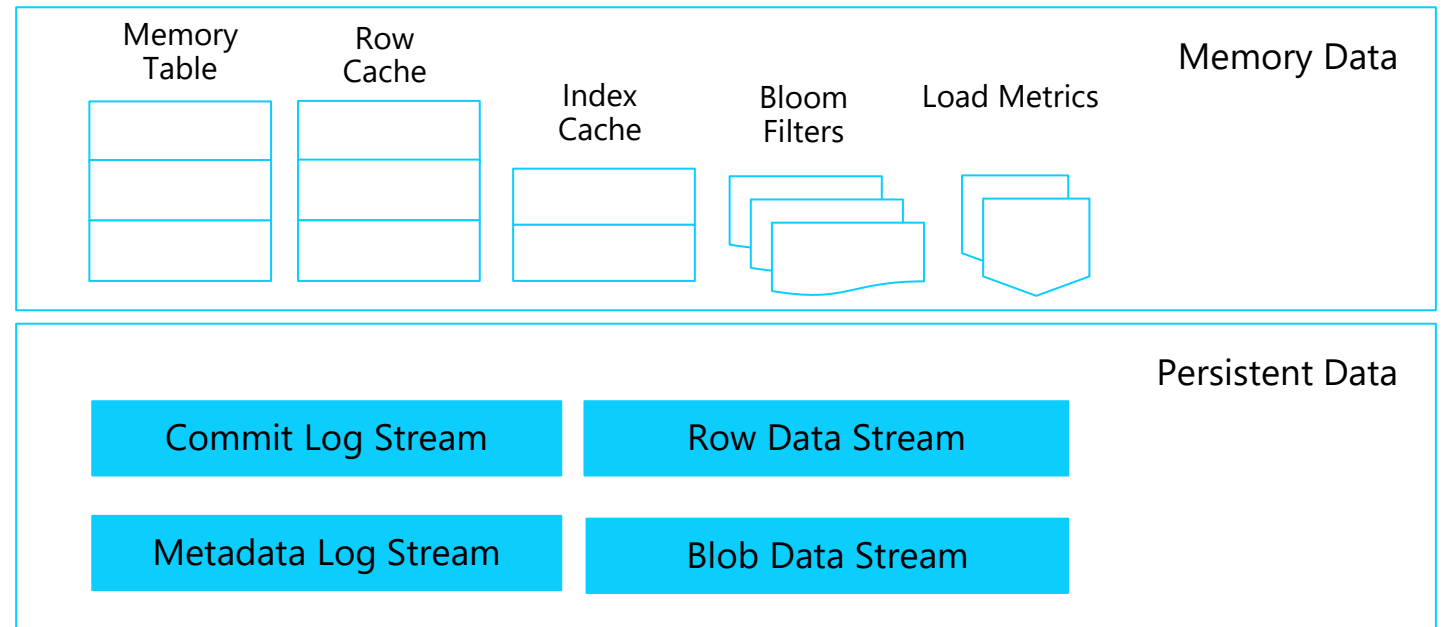




# Partition Server

## Architecture

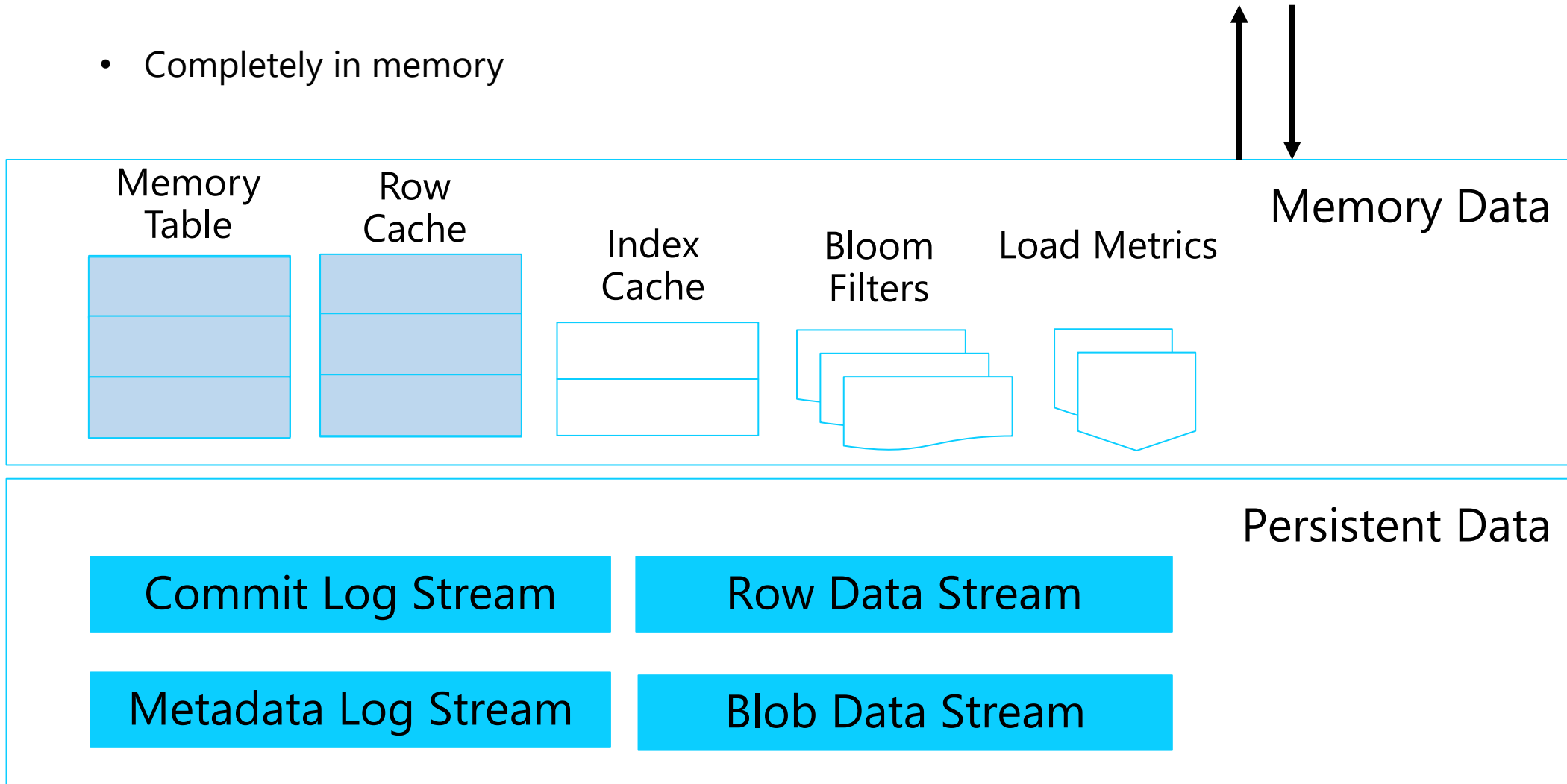
- Log Structured Merge-Tree
  - Easy replication
- Append only persisted streams
  - Commit log
  - Metadata log
  - Row Data (Snapshots)
  - Blob Data (Actual blob data)
- Caches in memory
  - Memory table (= Commit log)
  - Row Data
  - Index (snapshot location)
  - Bloom filter
  - Load Metrics



# Incoming Read request

## HOT Data

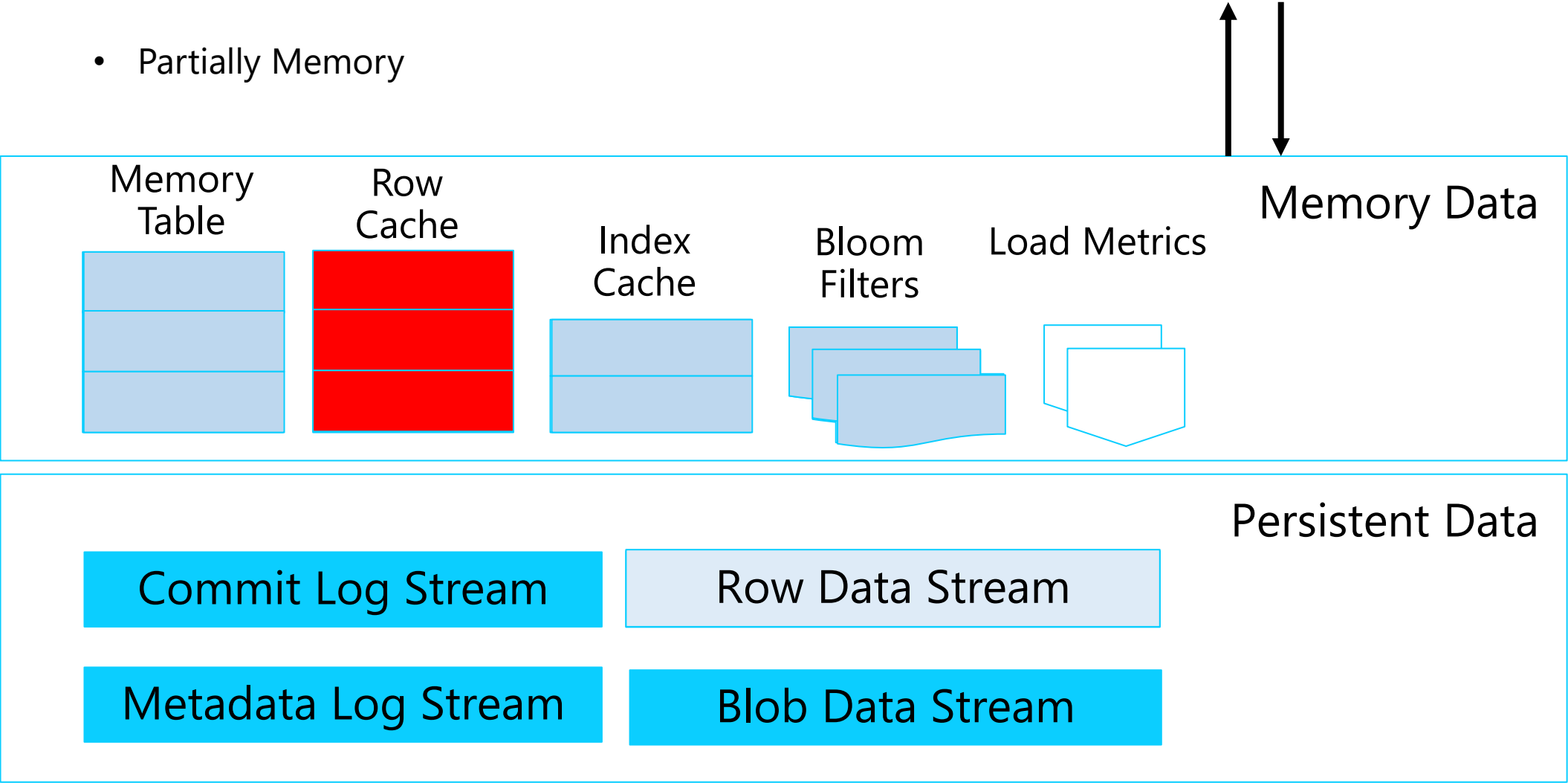
- Completely in memory



# Incoming Read request

## COLD Data

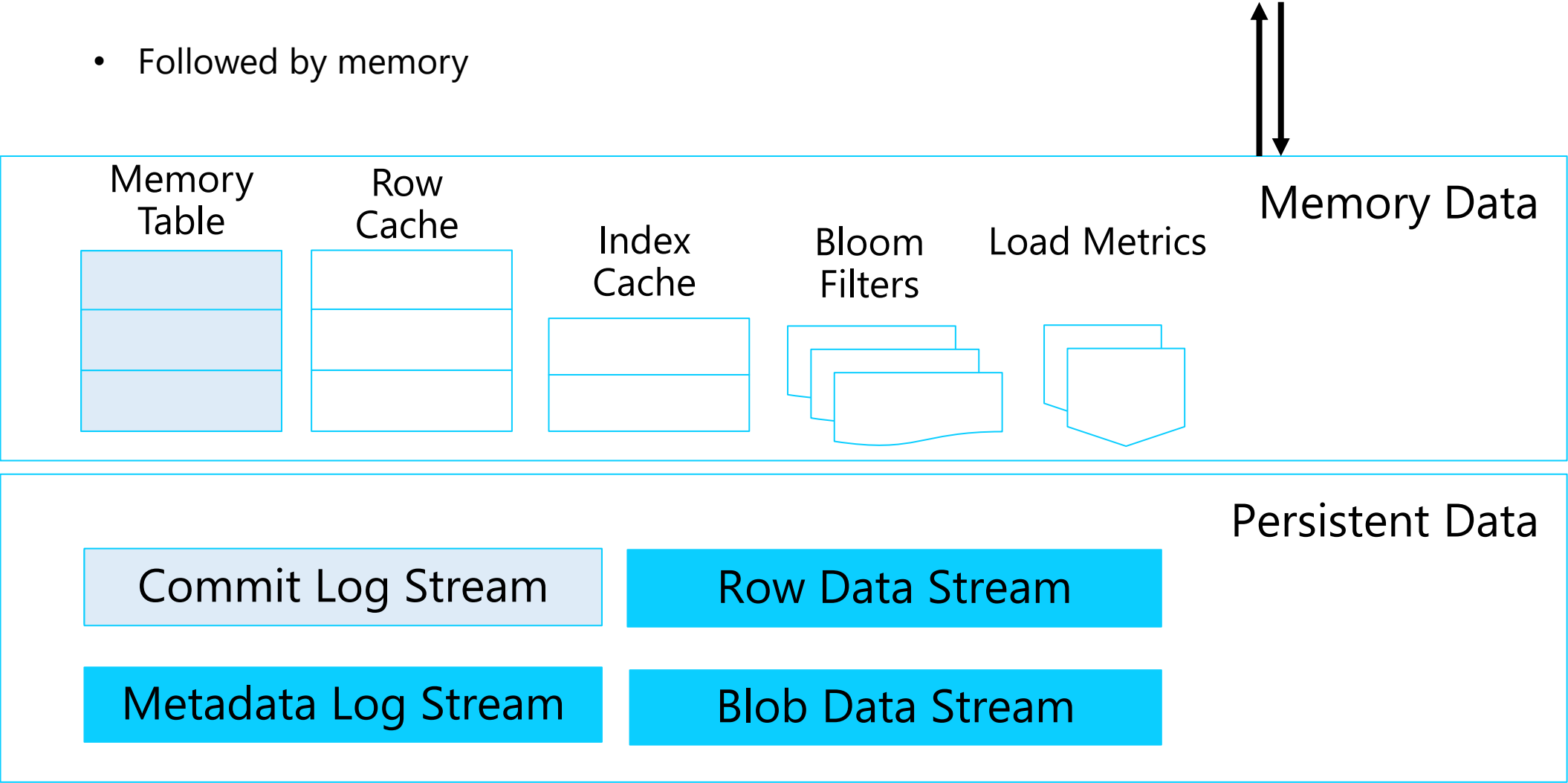
- Partially Memory



# Incoming Write Request

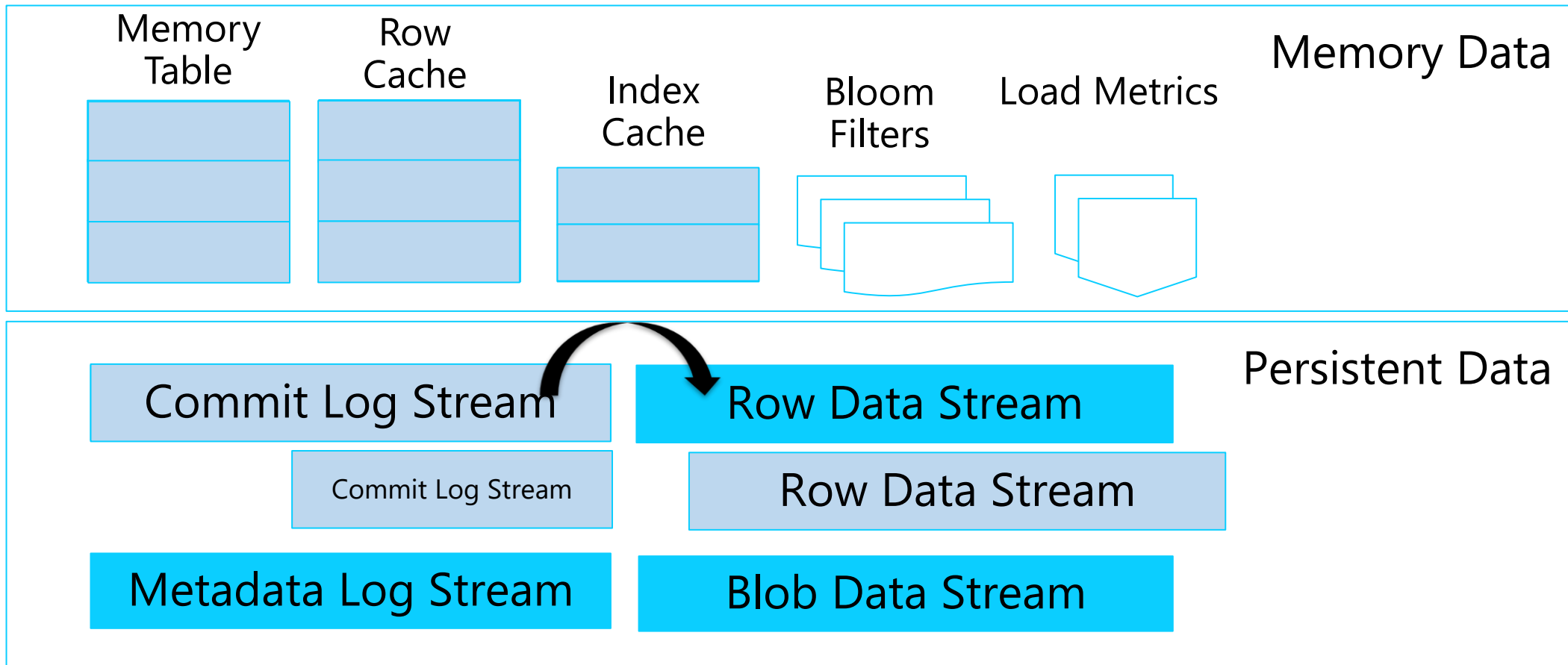
## Persistent First

- Followed by memory



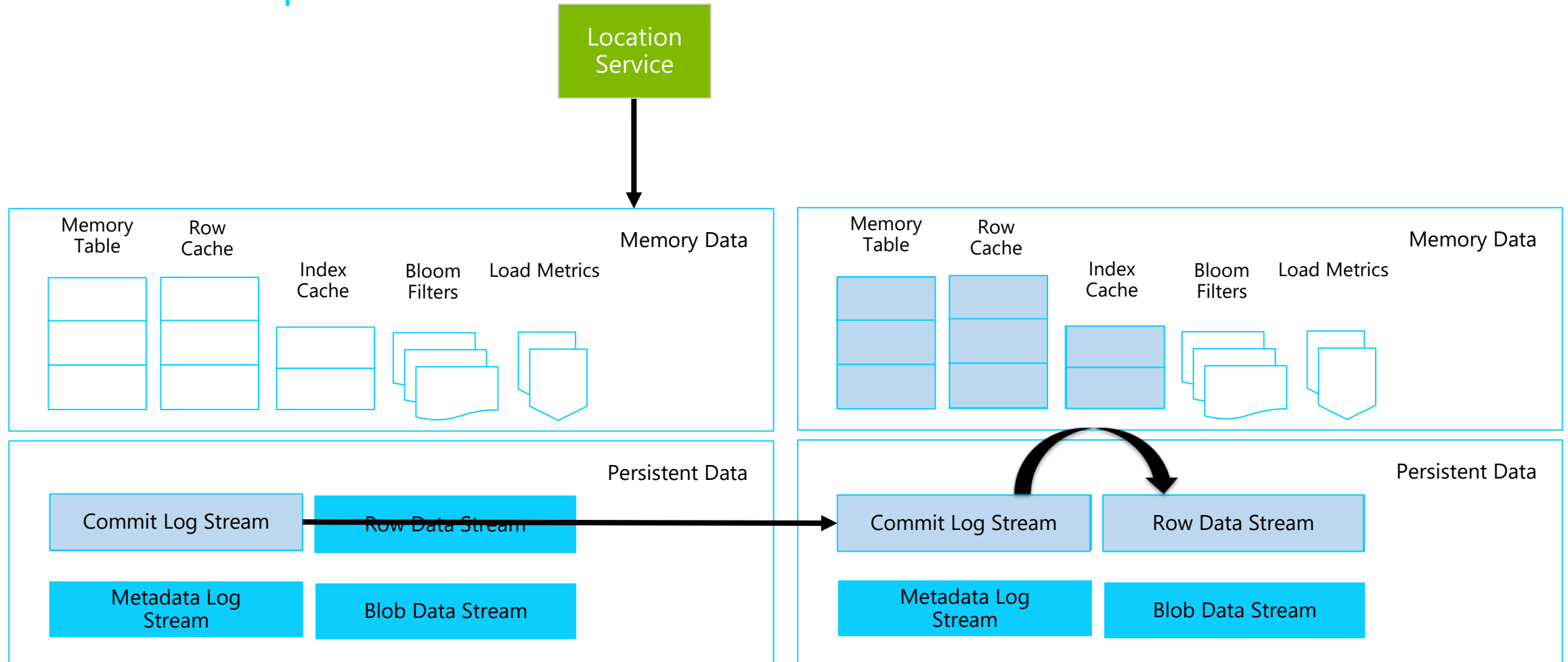
# Checkpoint process

Not in critical path!



# (Geo) replication process

Not in critical path!



# Internal Architecture

Global  
namespace

Front End  
Layer

Partition  
Layer

Stream  
Layer

# Stream layer

## Append-Only Distributed File System

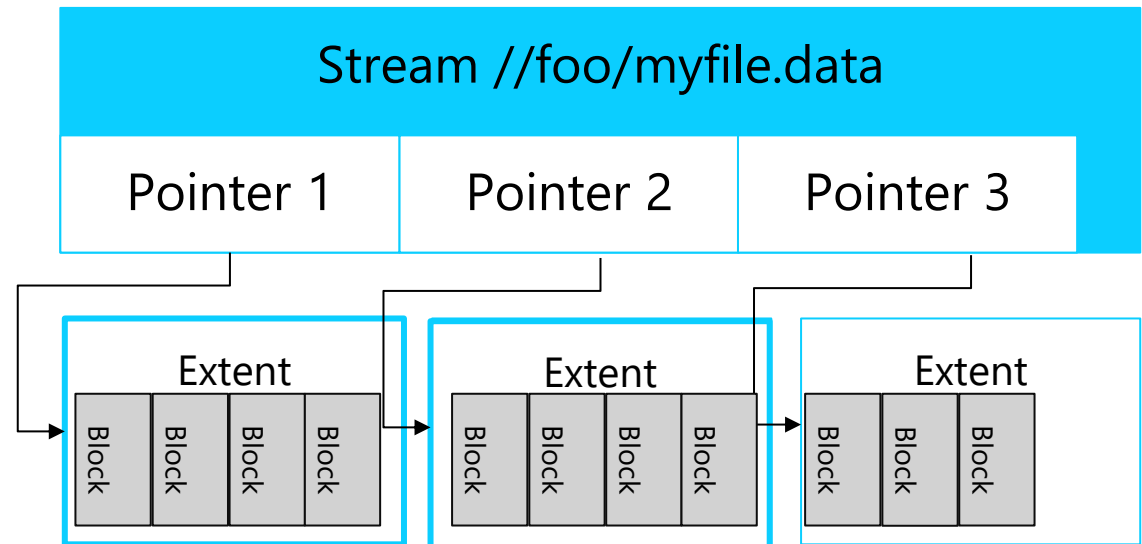
- Streams are very large files
  - File system like directory namespace
- Stream Operations
  - Open, Close, Delete Streams
  - Rename Streams
  - Concatenate Streams together
  - Append for writing
  - Random reads



# Stream layer

## Concepts

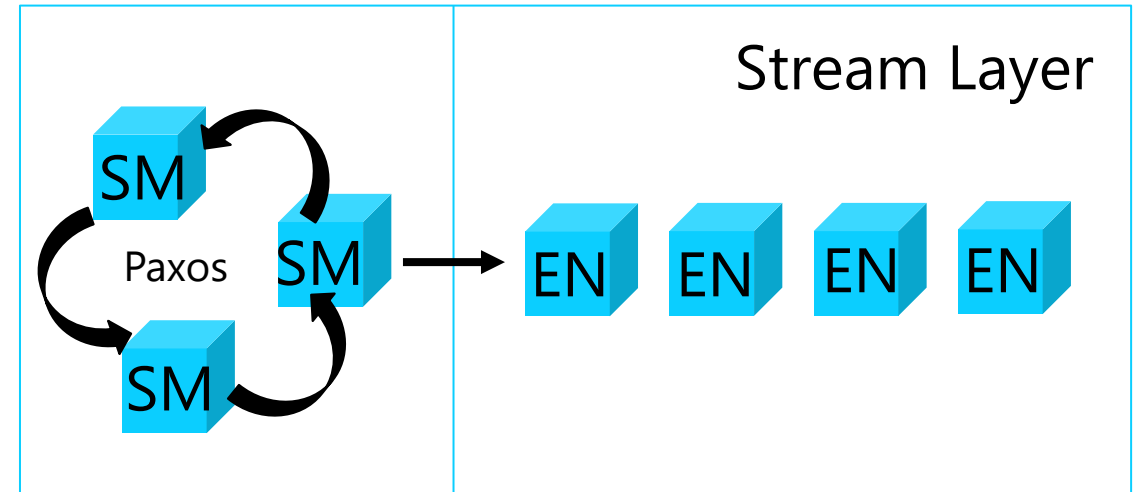
- Block
  - Min unit of write/read
  - Checksum
  - Up to N bytes (e.g. 100MB)
- Extent
  - Unit of replication
  - Sequence of blocks
  - Size limit (e.g. 1GB)
  - Sealed/unsealed
- Stream
  - Hierarchical namespace
  - Ordered list of pointers to extents
  - Append/Concatenate



# Stream layer

## Consists of

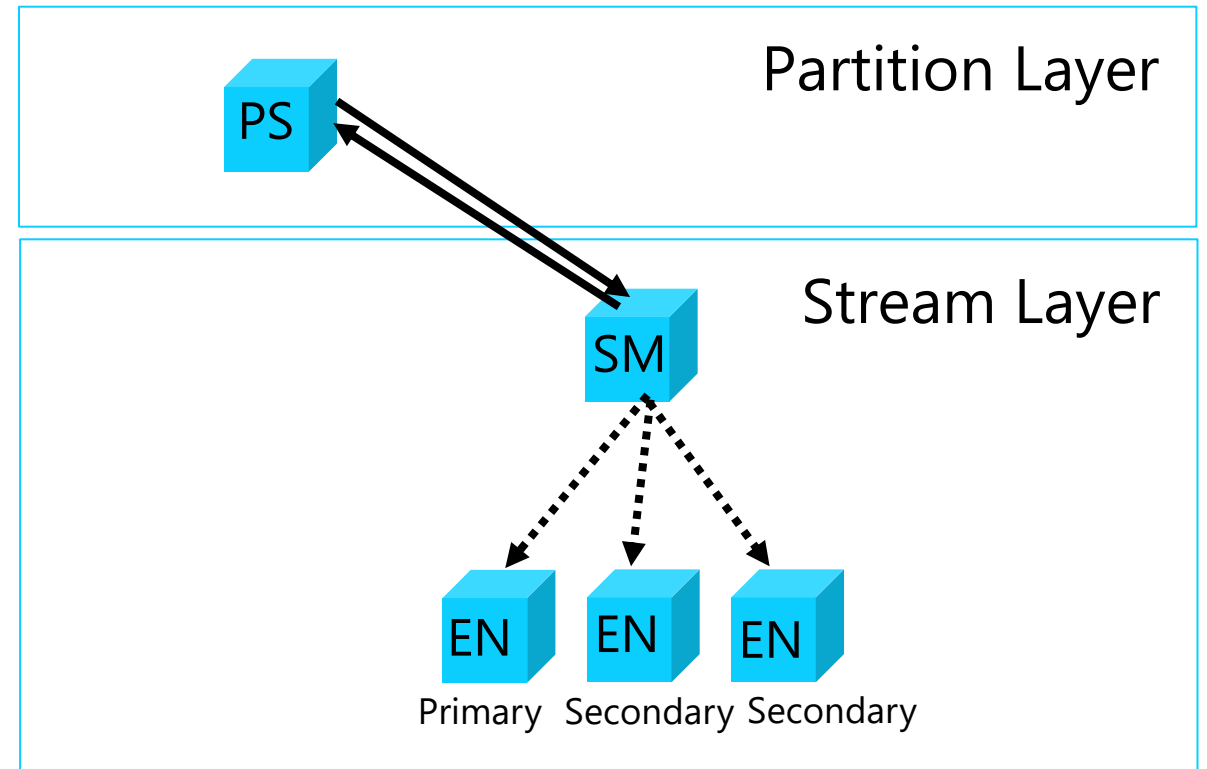
- Stream Master
  - Paxos System
  - Distributed consensus protocol
- Extend Nodes
  - Attached Disks
  - Manages Extents
  - Optimization Routines



# Extent creation

## Allocation process

- Partition Server
  - Request extent / stream creation
- Stream Master
  - Chooses 3 random extent nodes based on available capacity
  - Chooses Primary & Secondary's
  - Allocates replica set
- Random allocation
  - To reduce Mean Time To Recovery



# Replication Process

## Bitwise equality on all nodes

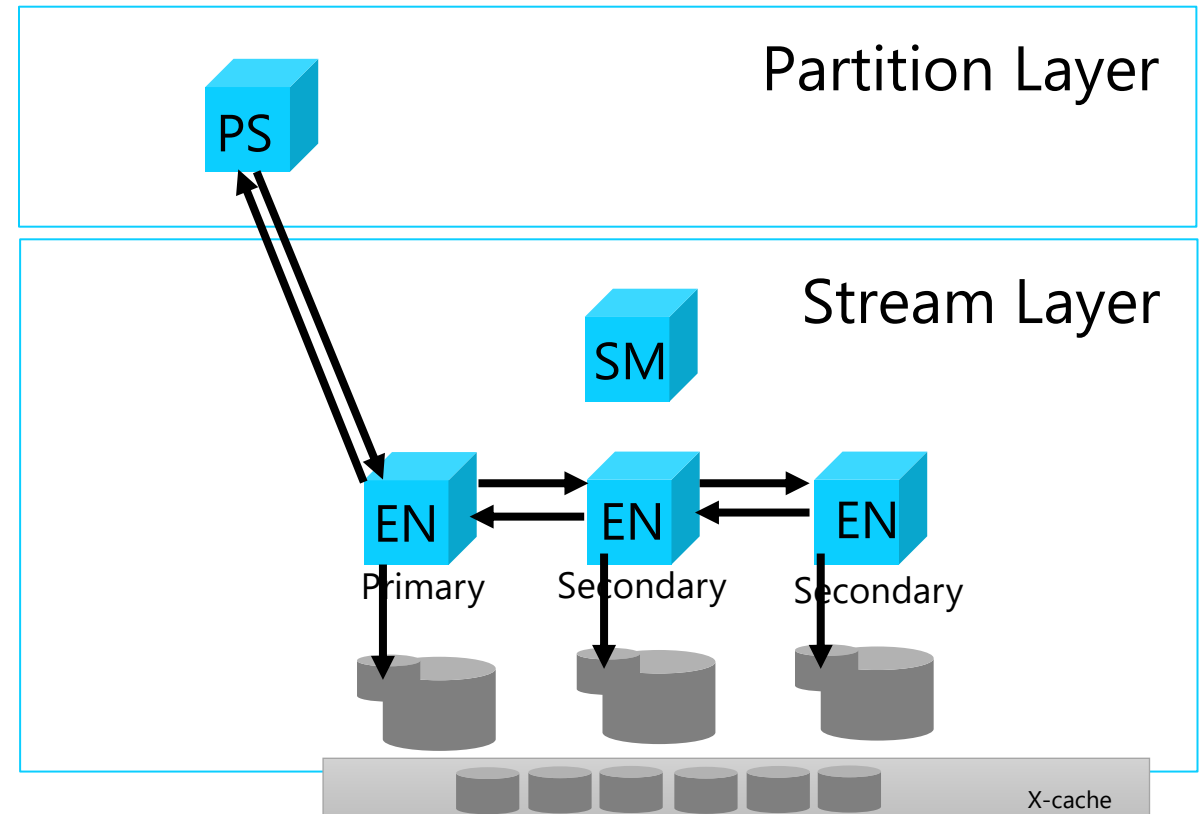
- Synchronous
- Append request to primary
- Written to disk, checksum
- Offset and data forwarded to secondaries
- Ack

## Journaling

- Dedicated disk (spindle/ssd) for writes
- Simultaneously copied to data disk (from memory or journal)

## Caching

- Simultaneously copied to x-cache
- Shared ssd drives



Bonus?

# Resources

# Resources

## Want to know more?

- White paper: <http://sigops.org/sosp/sosp11/current/2011-Cascais/printable/11-calder.pdf>
- Video by Brad Calder: <http://www.youtube.com/watch?v=QnYdbQO0yj4>
- More slides by Brad Calder: <http://sigops.org/sosp/sosp11/current/2011-Cascais/11-calder.pptx>
- Blog post by 8Kmiles: <http://8kmiles.com/azure-storage-high-level-architecture/>
- Erasure coding: <https://www.usenix.org/conference/atc12/technical-sessions/presentation/huang>

CloudBrew.be  
13-14 December



Q&A

# Fault Tolerance

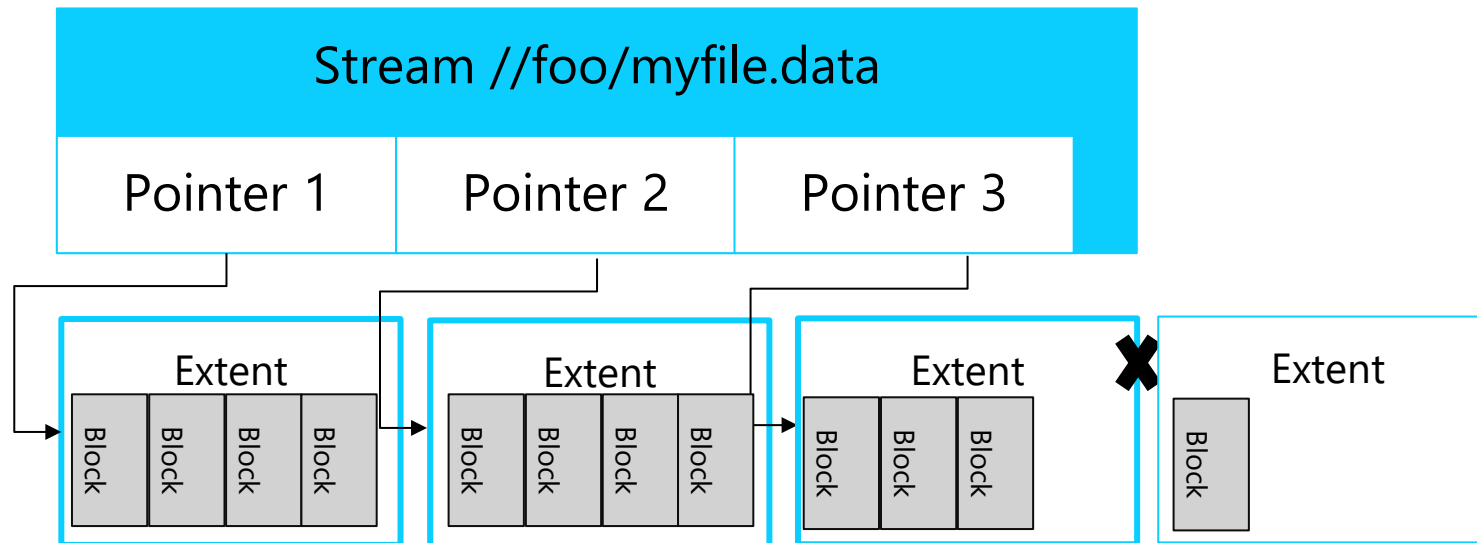
# Dealing with failure

## Ack from primary lost going back to partition layer

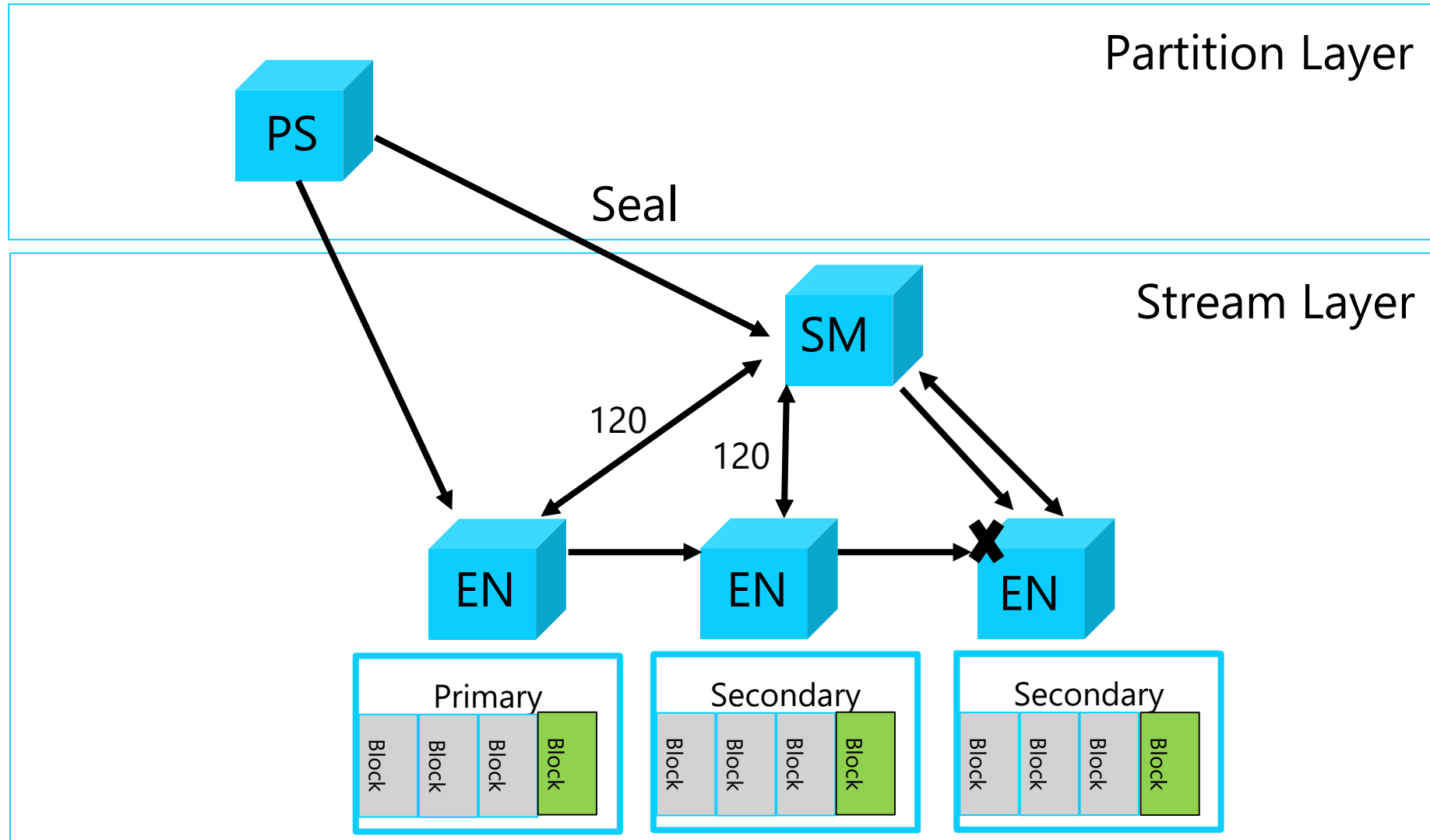
- Retry from partition layer
- can cause multiple blocks to be appended (duplicate records)

## Unresponsive/Unreachable Extent Node (EN)

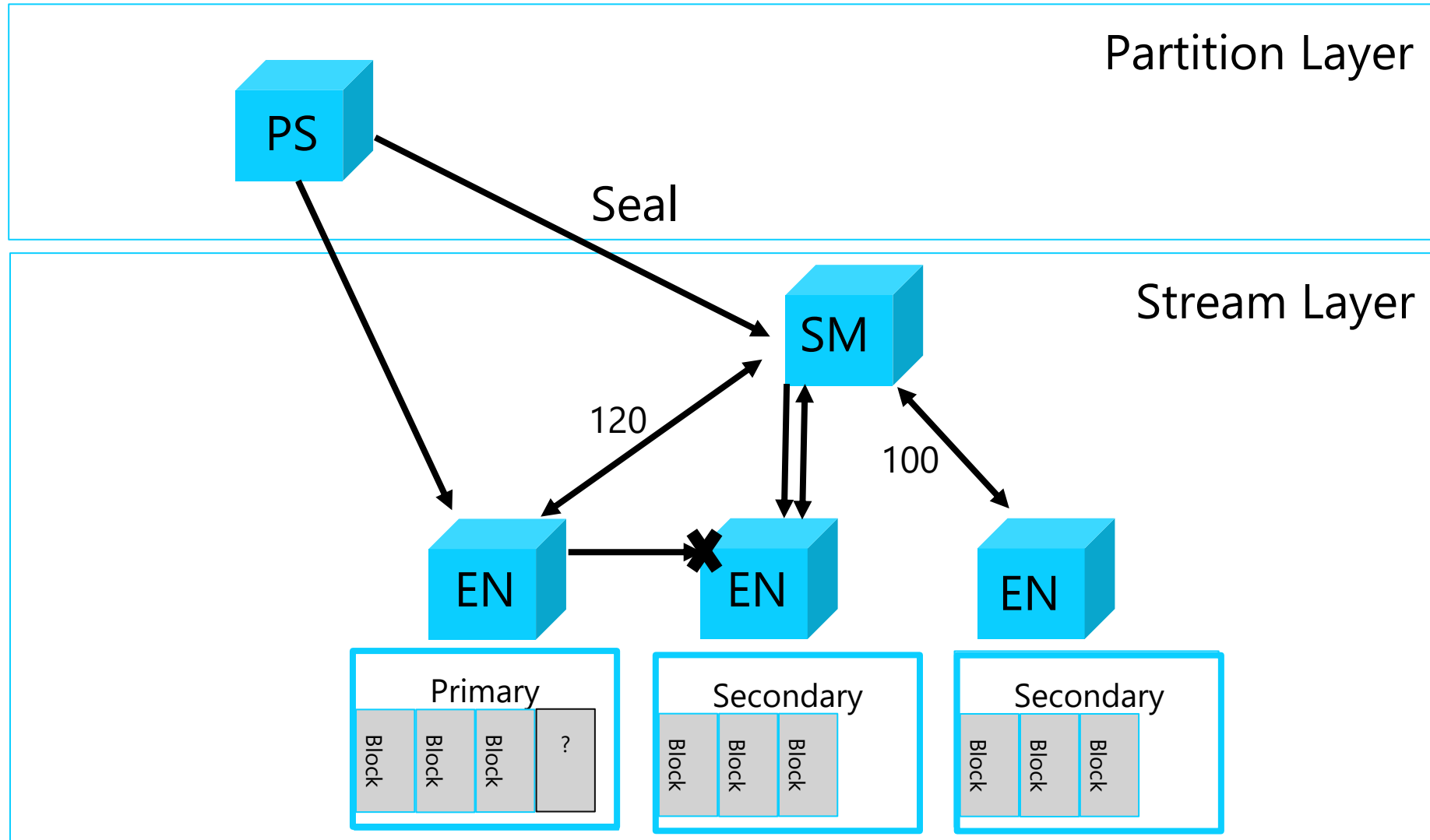
- Seal the failed extent & allocate a new extent and append immediately



# Replication Failures (Scenario 1)



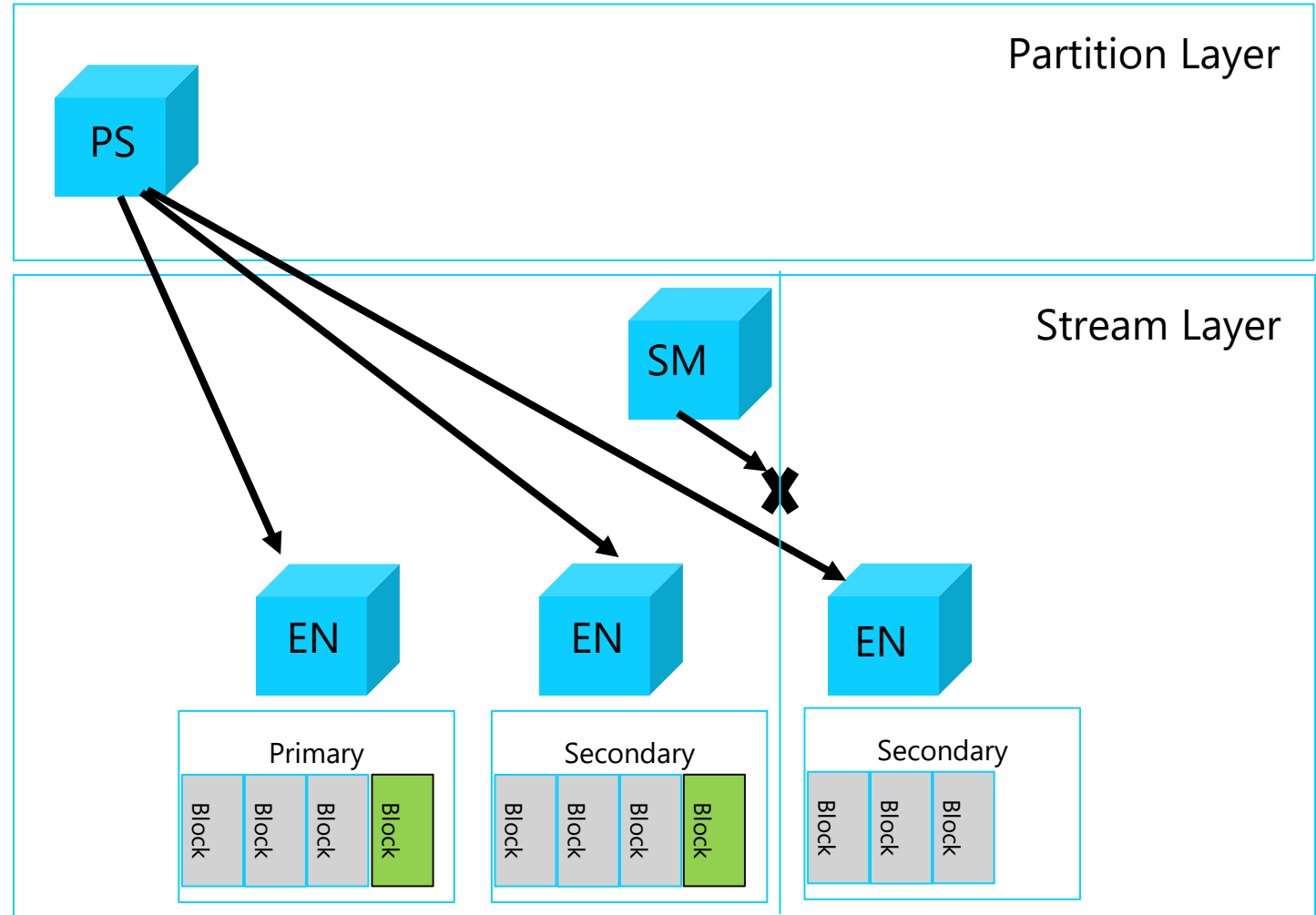
# Replication Failures (Scenario 2)



# Network partitioning during read

## Data Stream

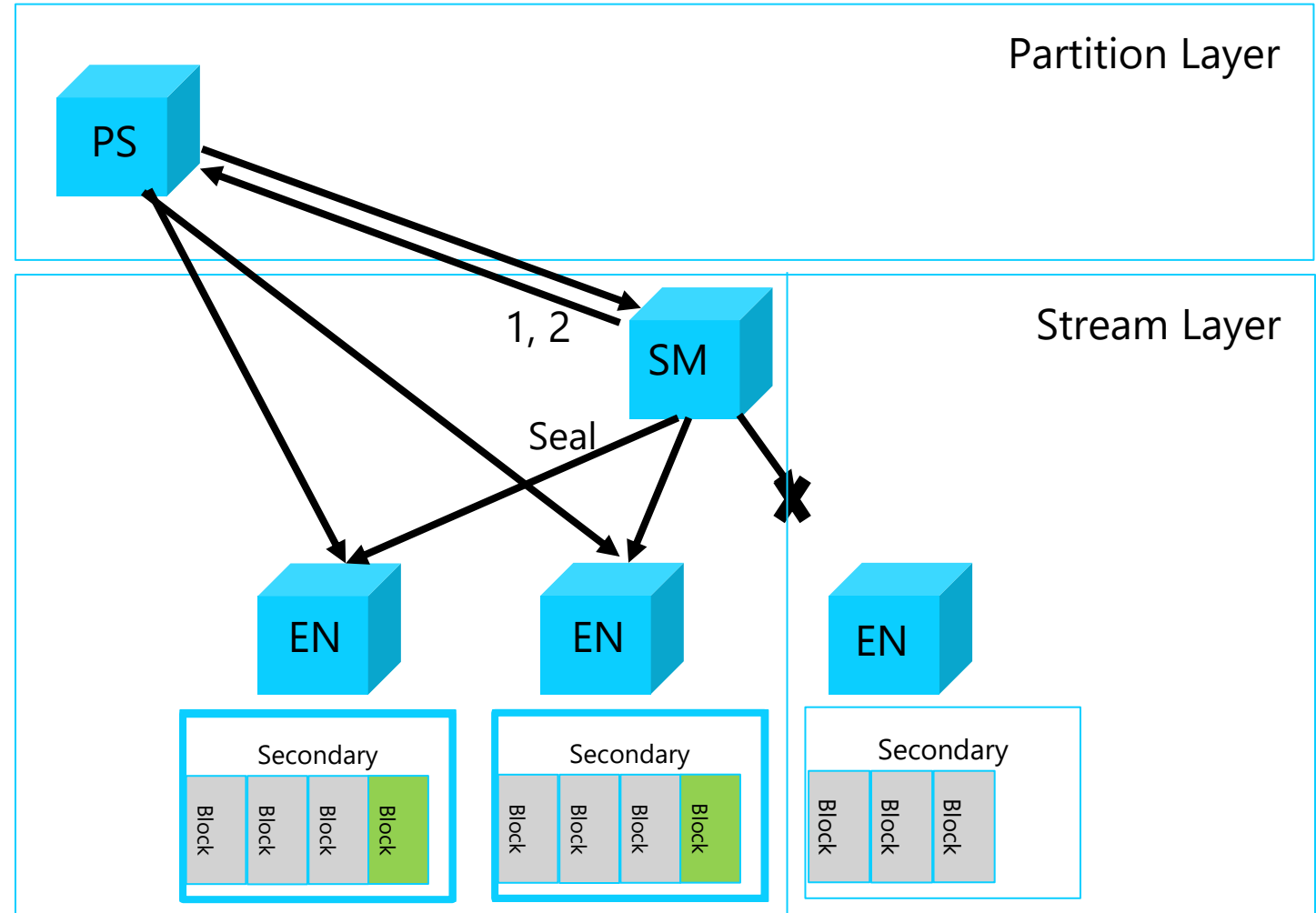
- Partition Layer only reads from offsets returned from successful appends
- Committed on all replicas
- Row and Blob Data Streams
- Offset valid on any replica
- Safe to read



# Network partitioning during read

## Log Stream

- Logs are used on partition load
- Check commit length first
- Only read from
  - Unsealed replica if all replicas have the same commit length
  - A sealed replica



# Cost Savings



# Garbage Collection

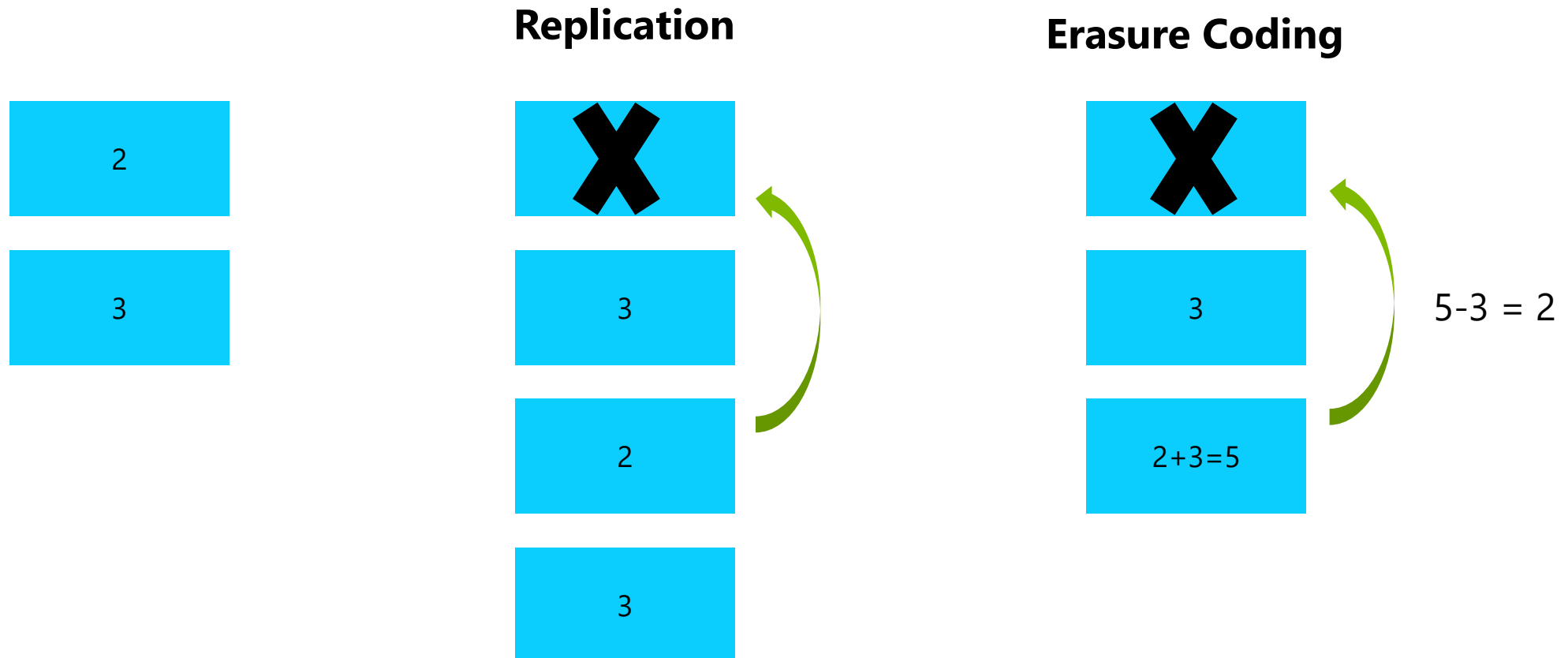
## Wasted disk space

- Lot's of wasted disk space
  - 'Forgotten' streams
  - 'Forgotten' extends
  - Unused blocks in extends after write errors
- Garbage collection process
  - Partition servers mark used streams
  - Stream manager cleans up unreferenced streams
  - Stream manager cleans up unreferenced extends

# Erasure coding

## Duplicated data

- Stream manager performs erasure coding on sealed extents
  - Allows recreation of cold streams after deleting/losing extents
- After erasure coding the replicas are deleted (3 replicas -> 1.5 replica)



# Reed-Solomon

## Algorithm

- Distributed the extent in equal parts
- Compute parity codes, also distributed
- Further reduce storage costs by increasing distribution & parity codes



# Reconstruction cost

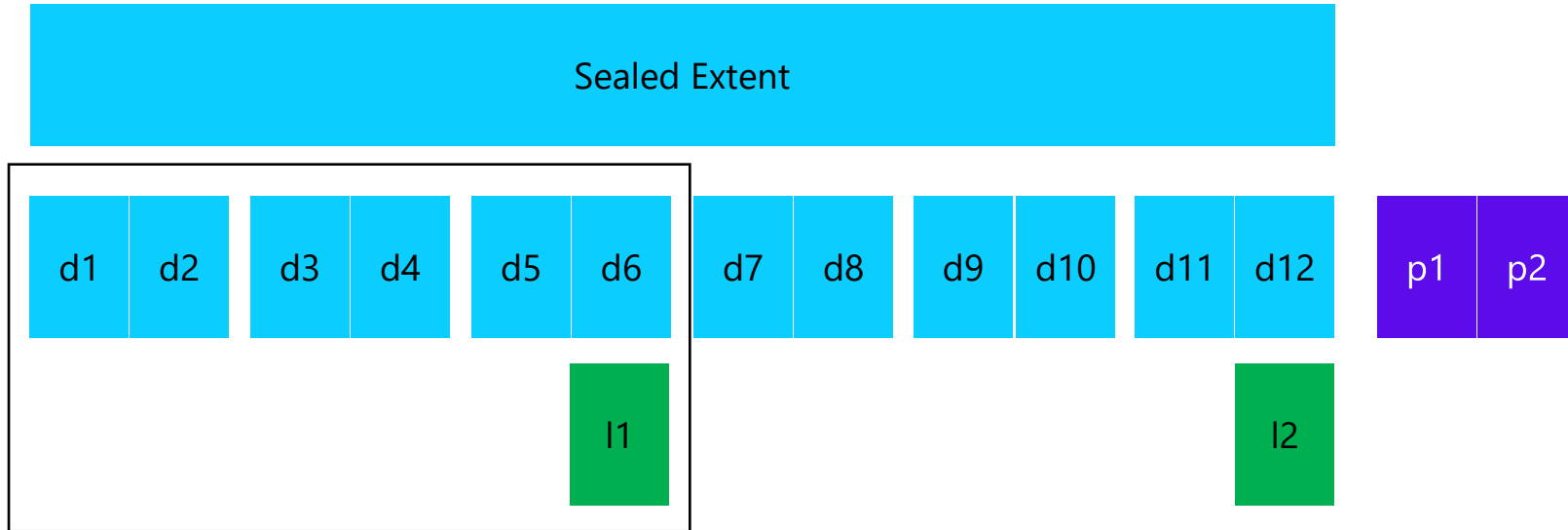
## Introduces extra cost on read

- $6+3 \Rightarrow 6 \times \text{disk I/O} + 6 \times \text{network transfer}$
- $12+4 \Rightarrow 12 \times \text{disk I/O} + 12 \times \text{network transfer}$
- Happens quite often
  - During load balancing
  - During rolling upgrade
  - On failures
- In critical path!

# Local Reconstruction Code

## Balance cost vs performance

- Not all faults are the same, higher % 1 fault, then 2 faults
- Compute code in clusters.
  - $12 + 2 + 2$ : 1 fault = 6 read cost, 2 faults = 12 read costs, 1.33x storage cost
  - $14 + 2 + 2$ : 7 / 14 read cost, 1.29x storage cost



Storage cost: 1.33x  
Read costs: 6