

Where to Start?



MagicWorks™ have settled on Azure SQLDW, but what's next?

- Review our Options
- Design our Data Model
- Load a sample Data Set

Agenda

Distributed tables

Replicated tables

Partitioning tables

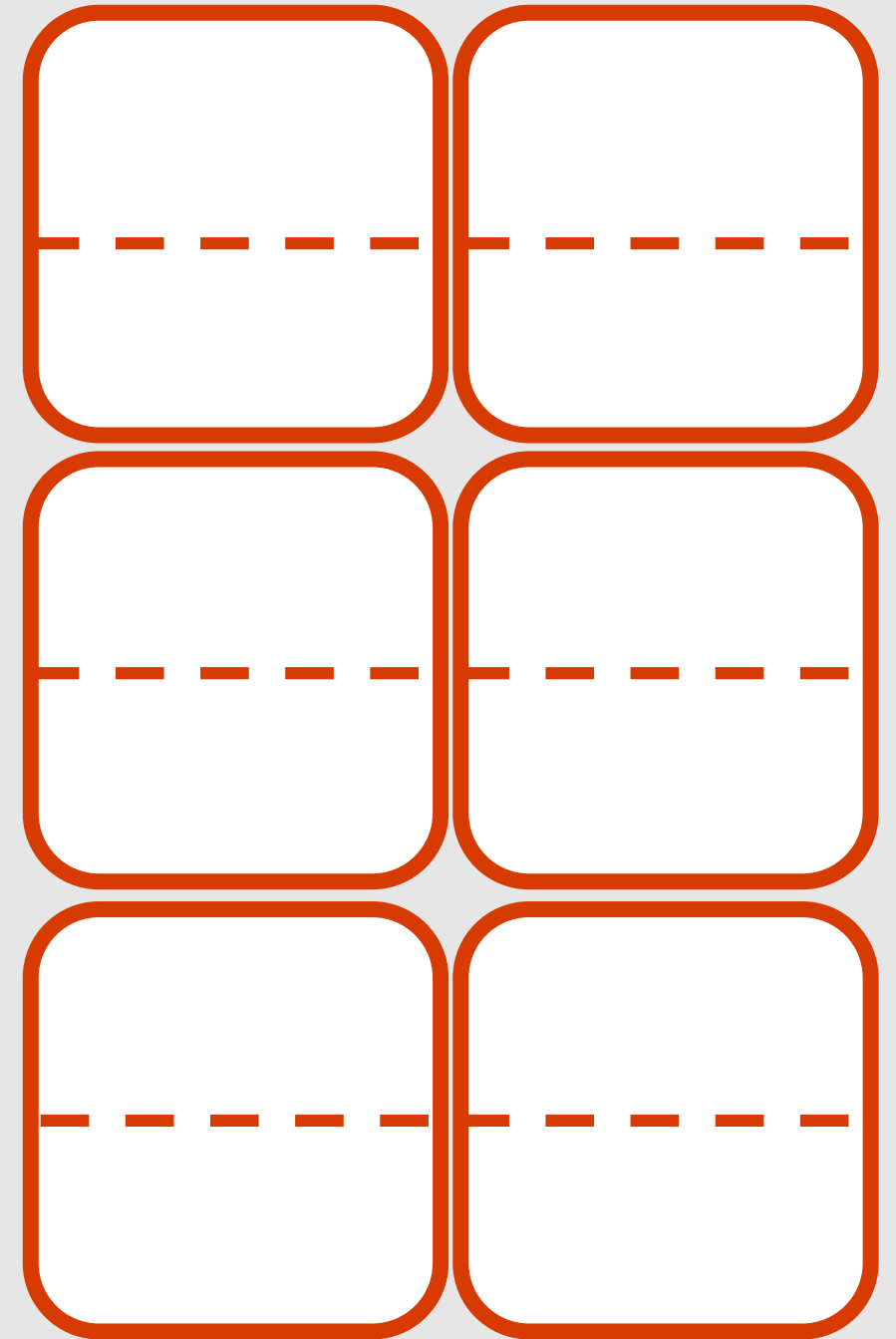
Maximizing performance

Distributed tables

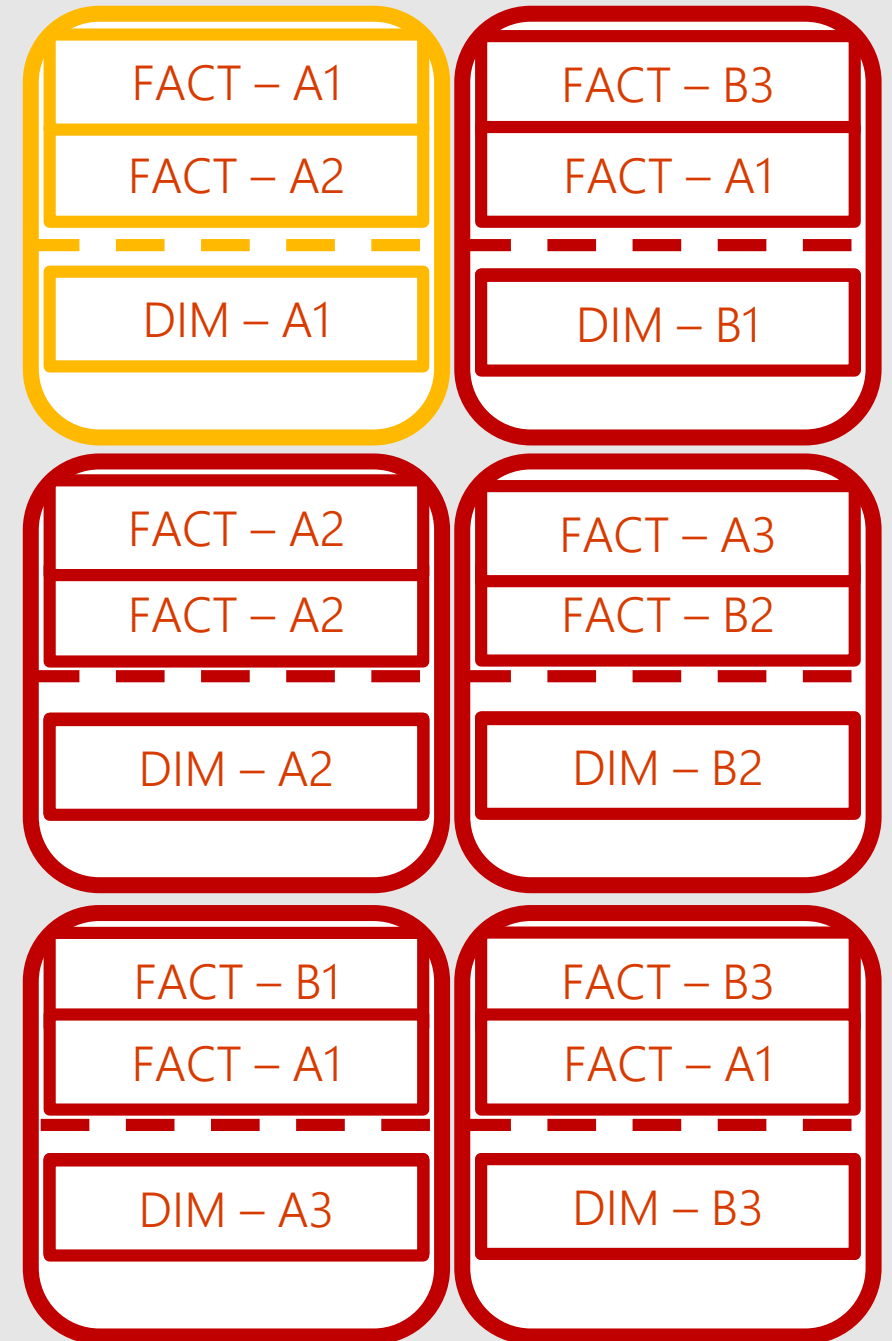
FACT – A1	FACT – A1
FACT – A1	FACT – B3
FACT – A2	FACT – A3
FACT – B2	FACT – A2
FACT – A2	FACT – B1
FACT – A1	FACT – B3

DIM – A1	DIM – B1
DIM – A2	DIM – B2
DIM – A3	DIM – B3

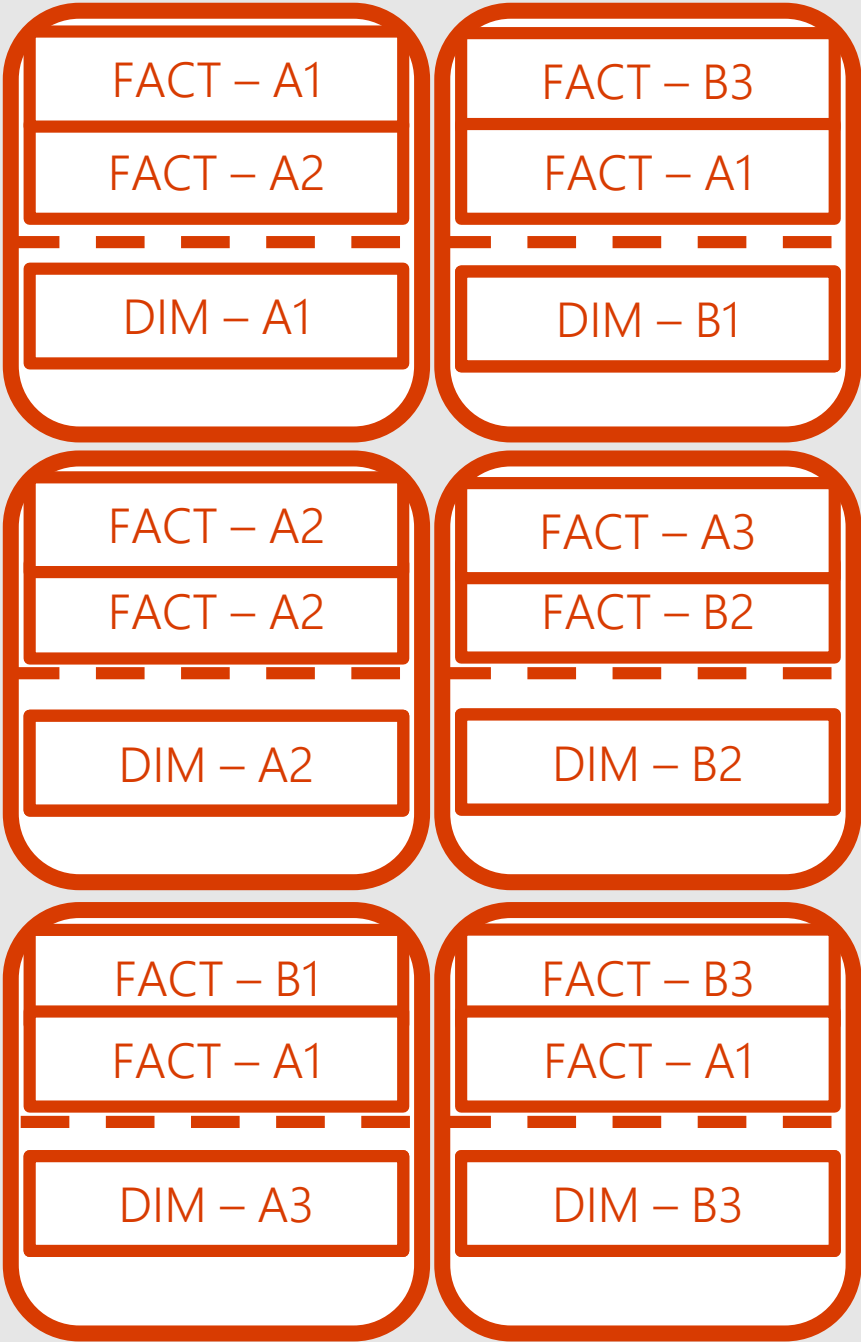
Distribution: Round Robin



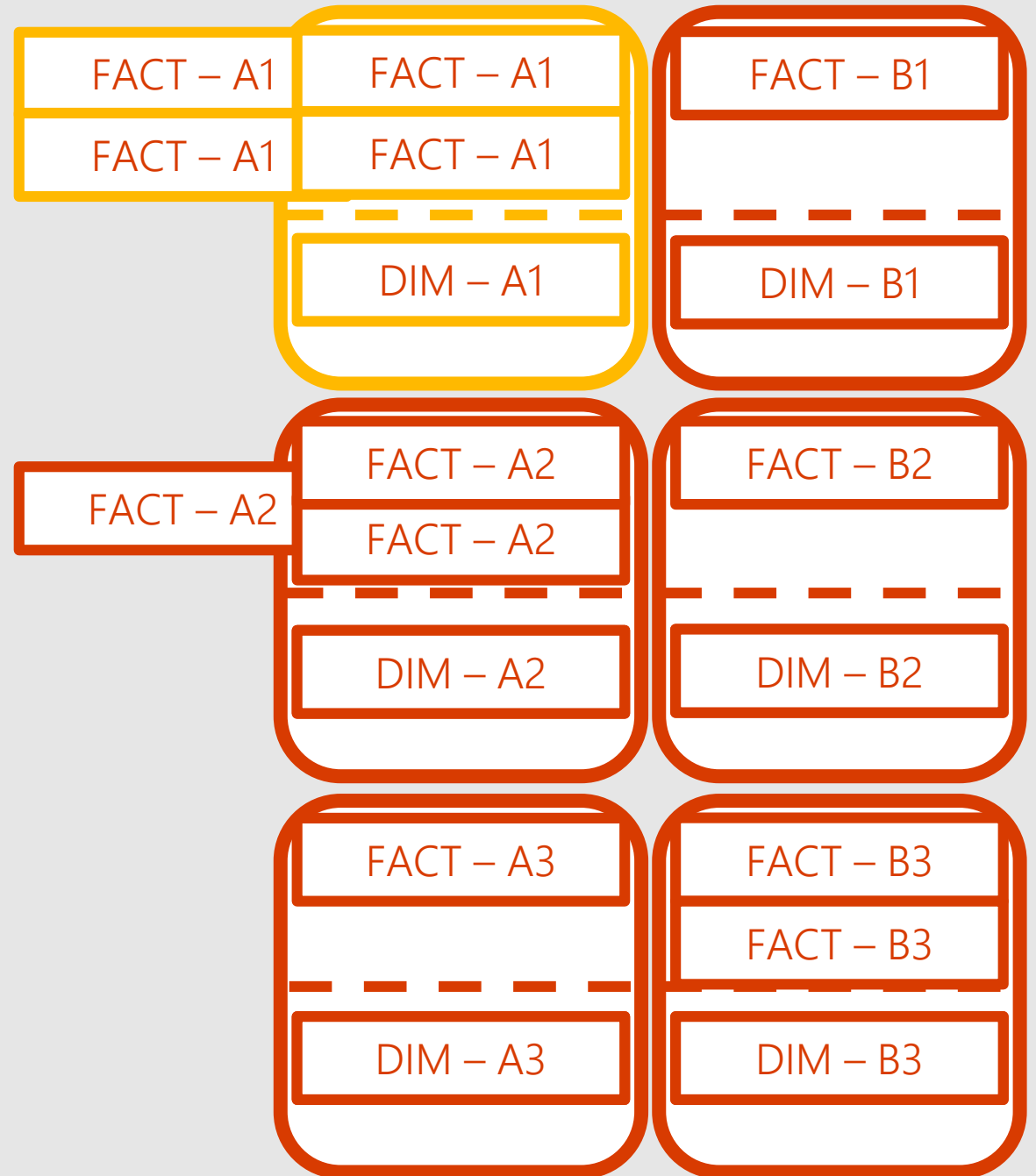
Query Execution



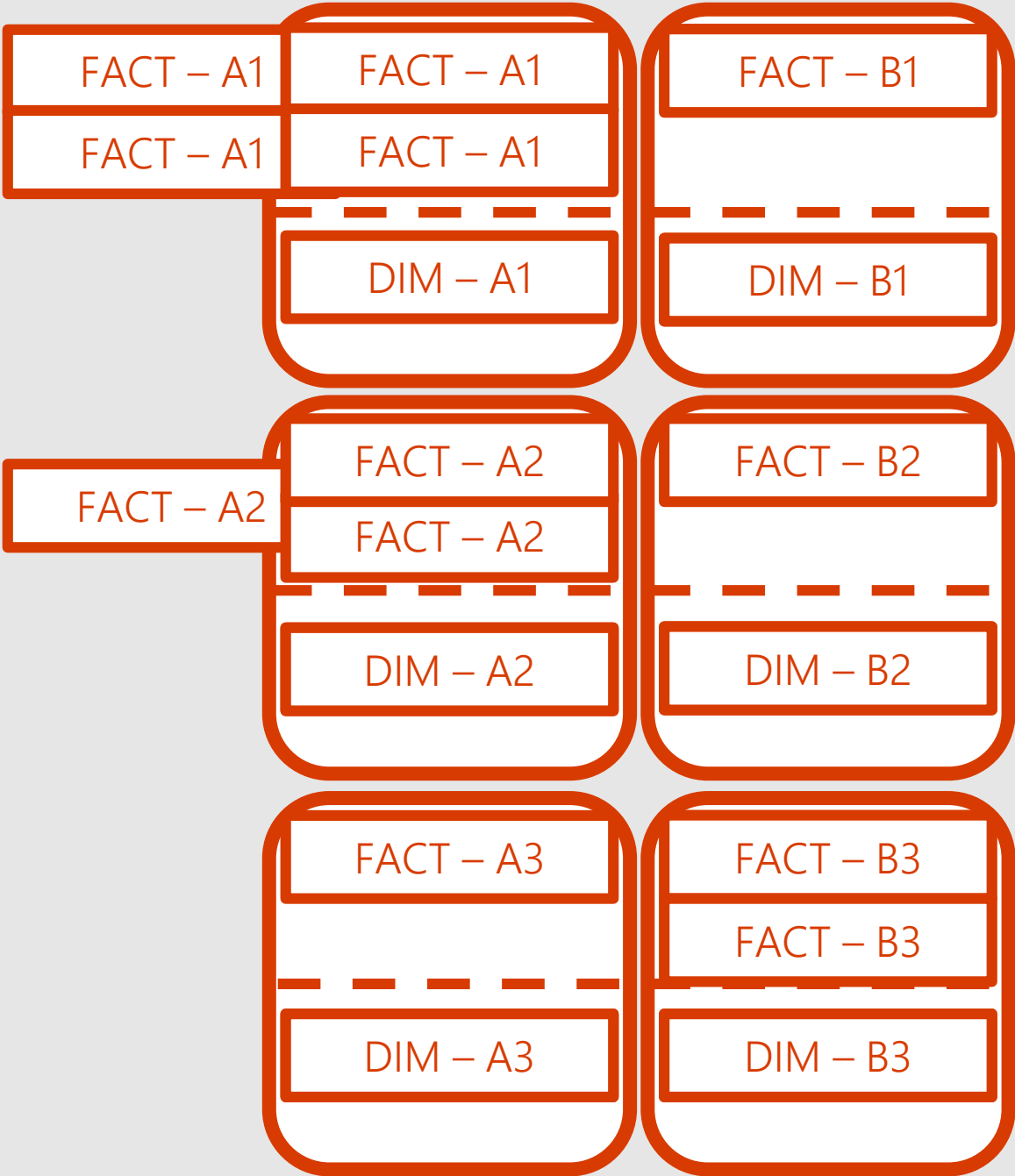
Distribution: HASH Column



Query Execution



```
SELECT DIM.NAME, COUNT(*)  
FROM FACT  
INNER JOIN DIM ON FACT.KEY = DIM.KEY  
GROUP BY DIM.NAME
```



Creating tables

```
CREATE TABLE [dbo].[DimStore]
```

```
(  
    [StoreKey]                int                NOT NULL  
, [GeographyKey]             int                NOT NULL  
, [StoreName]                 nvarchar(100)      NOT NULL  
, [StoreType]                 nvarchar(15)       NULL  
, [StoreDescription]          nvarchar(300)    NOT NULL  
, [Status]                    nvarchar(20)      NOT NULL  
, [OpenDate]                  datetime      NOT NULL  
, [CloseDate]                 datetime      NULL  
, [ETLLoadID]                 int                NULL  
, [LoadDate]                  datetime      NULL  
, [UpdateDate]                datetime      NULL  
)
```

```
WITH
```

```
( CLUSTERED INDEX([StoreKey])  
  DISTRIBUTION = ROUND_ROBIN  
)  
;
```

Row

```
CREATE TABLE [dbo].[FactOnlineSales]
```

```
(  
    [OnlineSalesKey]          int                NOT NULL  
, [DateKey]                  datetime          NOT NULL  
, [StoreKey]                 int                NOT NULL  
, [ProductKey]               int                NOT NULL  
, [PromotionKey]             int                NOT NULL  
, [CurrencyKey]              int                NOT NULL  
, [CustomerKey]              int                NOT NULL  
, [SalesOrderNumber]         nvarchar(20)      NOT NULL  
, [SalesOrderLineNumber]     int                NULL  
, [SalesQuantity]            int                NOT NULL  
, [SalesAmount]              money             NOT NULL  
)
```

```
WITH
```

```
( CLUSTERED COLUMNSTORE INDEX  
  DISTRIBUTION = HASH([ProductKey])  
)  
;
```

Column

Distribution

Distributed table design goals

Minimize data
skew

Minimize data
movement

Data Skew

Finding Skew

DBCC PDW_SHOWSPACEUSED

- Very quick

- Not a programmatic interface

DMV gives more detail and control

- `sys.dm_pdw_nodes_db_partition_stats`

- Examples in documentation

Documentation: [Table Size Queries](#)

Using the vTableSizes view

```
SELECT      [distribution_id]
,           SUM([row_count]) AS [total_distribution_row_count]
FROM        [dbo].[vTableSizes]
WHERE       [schema_name]      = 'Fact'
AND         [table_name]       = 'Flights'
GROUP BY    [distribution_id]
ORDER BY    [total_distribution_row_count]
;
```

Demo: checking for skew

Lab 002 – Monitoring Skew

10 Mins

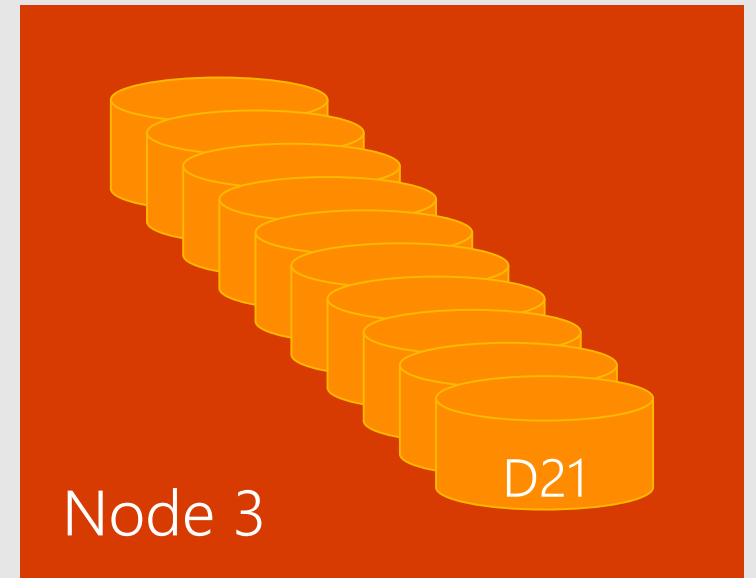
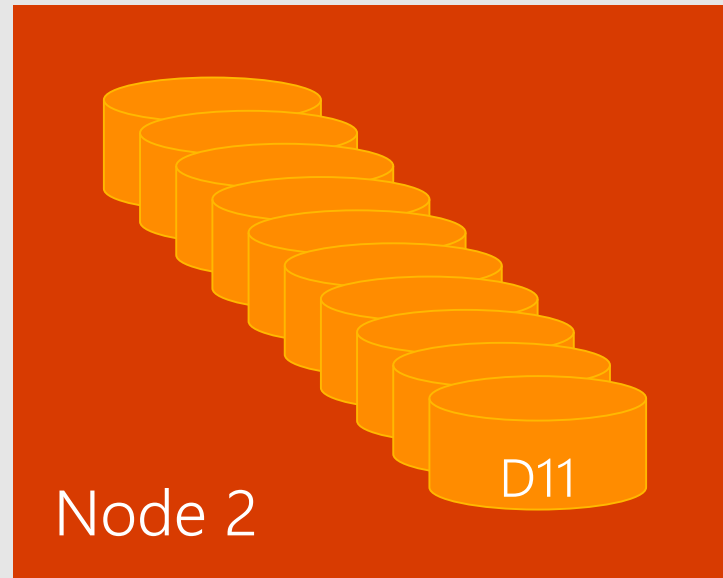
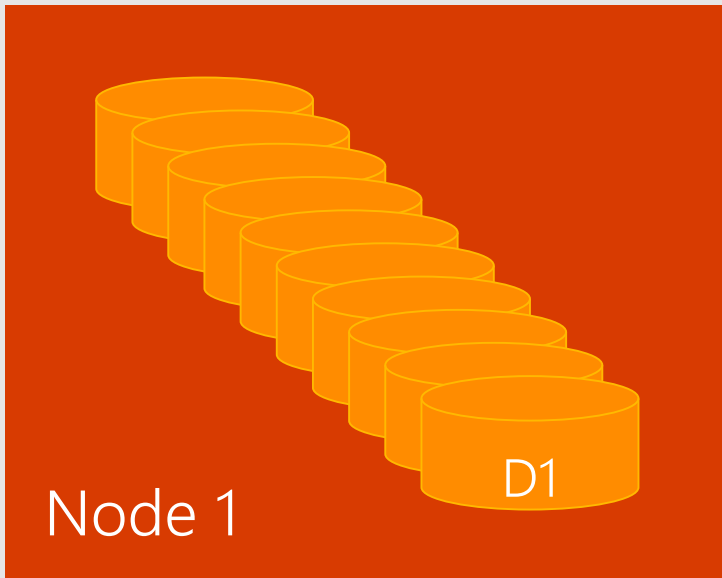
Replicated Tables

What is a replicated table

One complete copy of the data on each node

Improves read performance

Impacts write performance



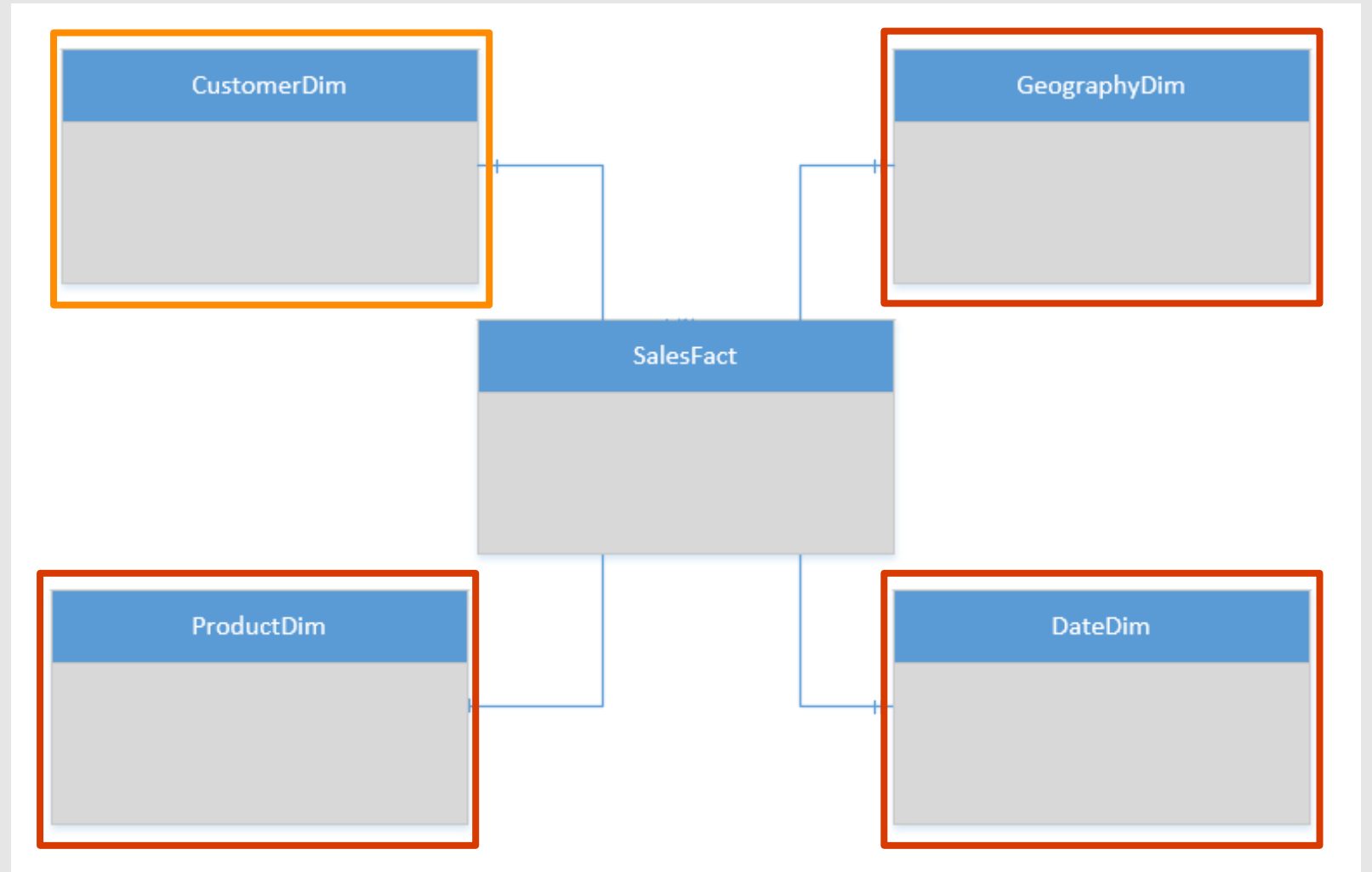
Replicated Table Scenarios

Dimensions

Master data

Reference data

Lookup tables in ELT



T-SQL: Create a Replicated table

```
CREATE TABLE dbo.DimCustomer
(
    CustomerKey          int          NOT NULL
,   GeographyKey        int          NULL
,   CustomerAlternateKey nvarchar(15) NOT NULL
,   Title               nvarchar(8)  NULL
,   FirstName           nvarchar(50)  NULL
,   LastName            nvarchar(50)  NULL
,   BirthDate           date          NULL
,   Gender              nvarchar(1)  NULL
,   EmailAddress        nvarchar(50)  NULL
,   YearlyIncome        money         NULL
,   DateFirstPurchase   date          NULL
)
WITH
(
    CLUSTERED COLUMNSTORE INDEX
,   DISTRIBUTION = REPLICATED
)
;
```



Table Size

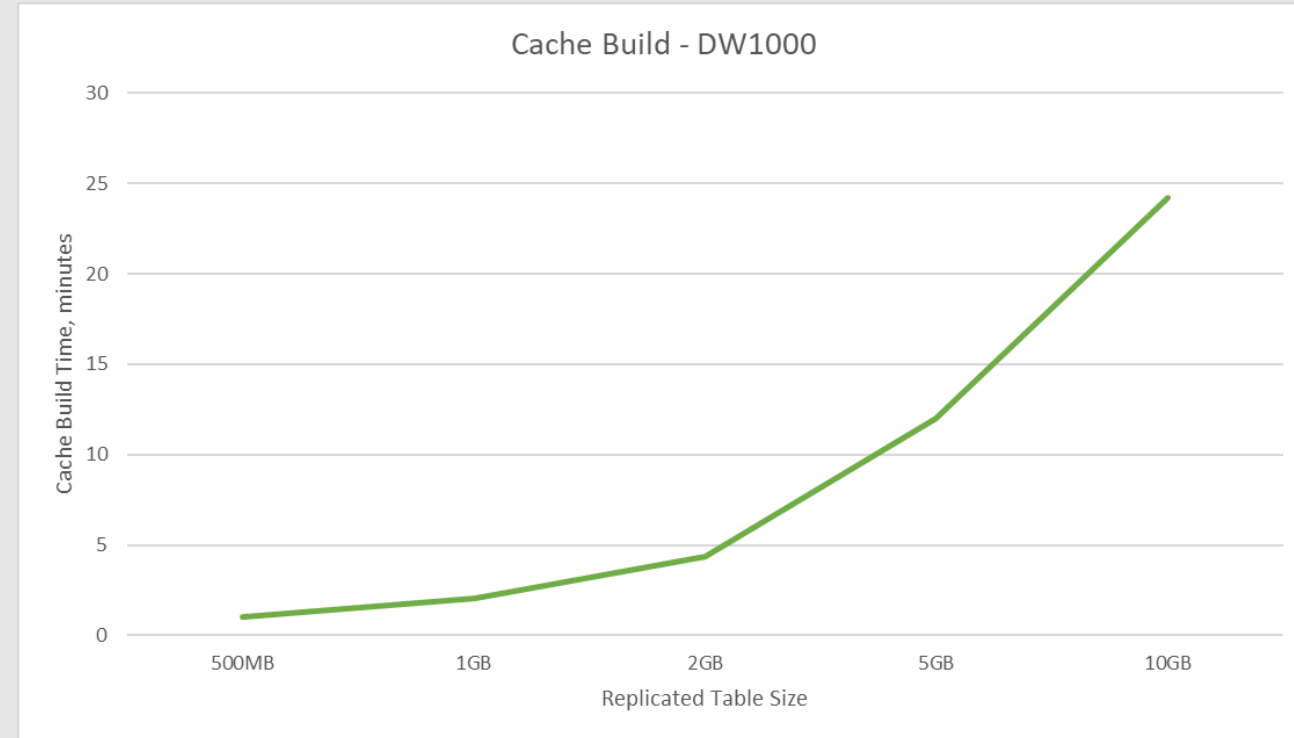
Guideline is 2GB

Compressed on disk size (typically observe 5x compression)

Regardless of the number of rows

Guideline grounded in physics of tenable cache refresh time

Carefully consider additional indexes as this adds to build time



Replicated Table Candidates

Tables involved in frequent joins that do not have a suitable HASH distribution column.

Use `sys.dm_pdw_request_steps`

BroadcastMoveOperation: replicated table at runtime...

Results			Messages		
	step_index	operation_type			
1	0	RandomIDOperation			
2	1	OnOperation			
3	2	BroadcastMoveOperation			
4	3	RandomIDOperation			
5	4	OnOperation			
6	5	BroadcastMoveOperation			
7	6	OnOperation			
8	7	PartitionMoveOperation			
9	8	OnOperation			
10	9	OnOperation			
11	10	ReturnOperation			
12	11	OnOperation			

Query Cost Considerations

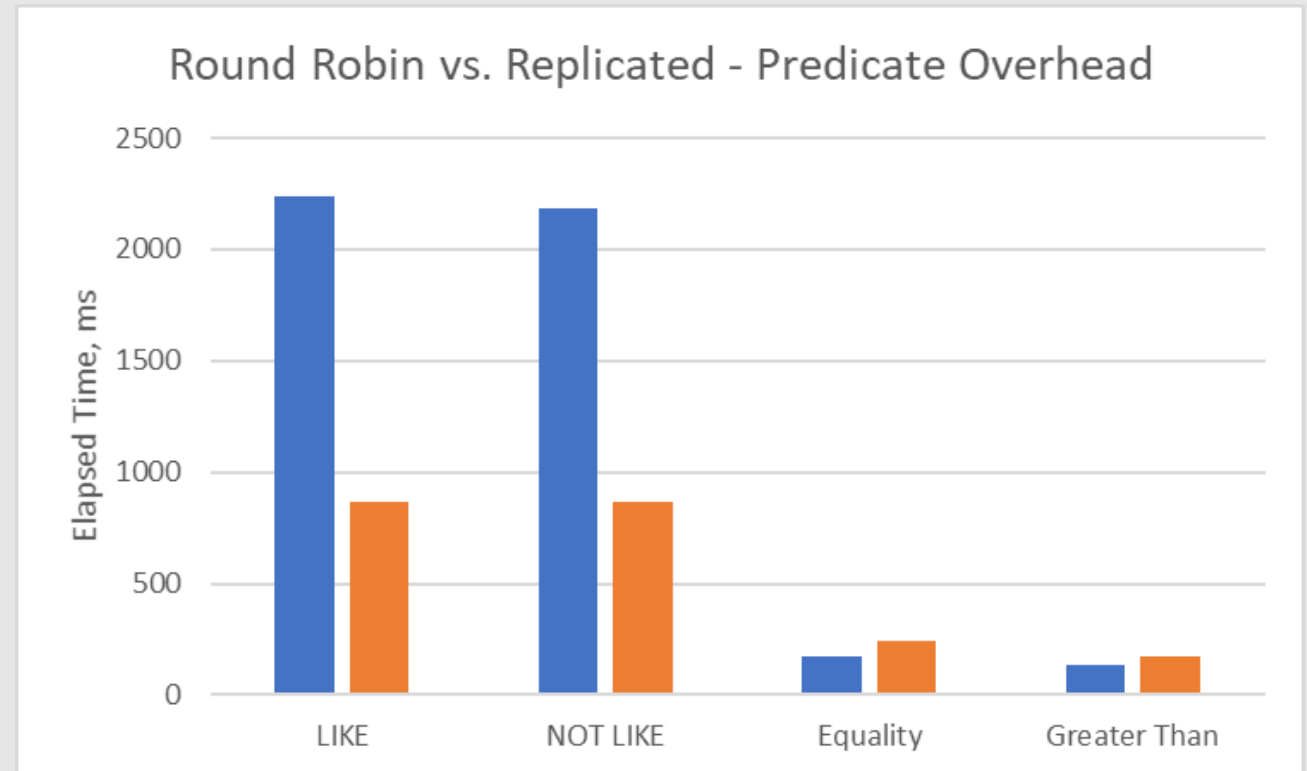
Data movement cost must outweigh predicate and join overhead.

Resource-intensive queries perform best when the work is distributed across all of the Compute nodes.

Query cost (select from replicated table):

Replicated table: [Query CPU Cost] \times [1 Compute Node]

Round Robin: [Query CPU Cost] \div [# Compute Nodes]



Replicated Table DMV

```
SELECT  t.[name]                                AS Table_Name
,        c.[state]                              AS Cache_State
,        p.[distribution_policy_desc]           AS Dist_Type
FROM    sys.[tables]                            AS t
JOIN    sys.[pdw_replicated_table_cache_state] AS c
ON      c.[object_id] = t.[object_id]
JOIN    sys.pdw_table_distribution_properties  AS p
ON      p.[object_id] = t.[object_id]
;
```

Demo: Redistributing Data

Lab 003: Redistributing Data

15 Mins

Table re-cap

Table structure options

Hash Distributed (to optimize)

Data divided across nodes based on hashing algorithm

Same value will always hash to same distribution

Single column only

Round Robin Distributed (default)

Data spread evenly across nodes & distributions

Easy place to start, don't need to know anything about the data

Simplicity at a cost

Replicated (In Preview)

Data repeated on every node

Simplifies many query plans and reduces data movement


Best with joining hash table

Check for Data Skew,
NULLS, -1

Will incur more data
movement at query time

Consumes more space
Queries using just
replicated tables executed
on one node only

Design Guidance Doc



SALES 1-800-867-1389 ▼CONTACT SALESMY ACCOUNTPORTALSearch

Why AzureSolutionsProductsDocumentationPricingTrainingMarketplacePartnersBlogResourcesSupport

FREE ACCOUNT >

[Azure](#) / [SQL Data Warehouse](#)

Filter

> Overview

> Get Started

▼ How To

> Backup and restore

> Connect

> Create

Design guidance for using replicated tables in Azure SQL Data Warehouse

2017-7-14 • 7 min to read • Contributors 

This article gives recommendations for designing replicated tables in your SQL Data Warehouse schema. Use these recommendations to improve query performance by reducing data movement and query complexity.

 Comments

 Edit

 Share

Theme

Light ▼

In this article

Reference:

[Design guidance for using replicated tables](#)

Modelling Magic

MagicWorks

Dim Date

Dim Geography

Fact Orders

Dim Customers

Dim Product

Modelling Magic



Fact Orders

- >60 Million Rows
- Lots of missing values & NULL columns
- Target of many aggregation queries
- Distribution = ROUND ROBIN
- Index = Clustered Columnstore Index

Modelling Magic



Dim Customer

- ~20,000 records
- Lot of data skew
- Distribution = $\text{HASH}(\text{CustomerKey})$
- Index = Clustered Index

Modelling Magic

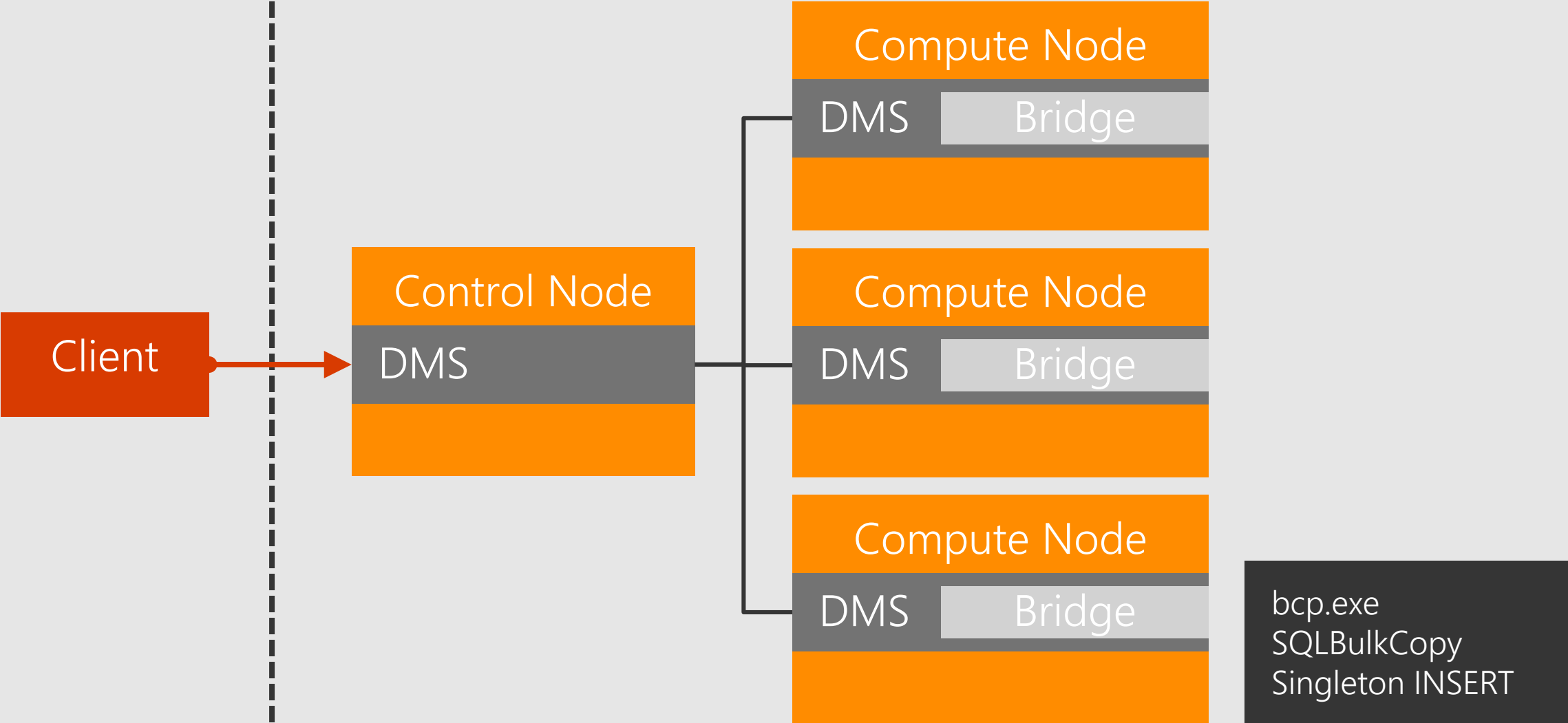


Dim Date

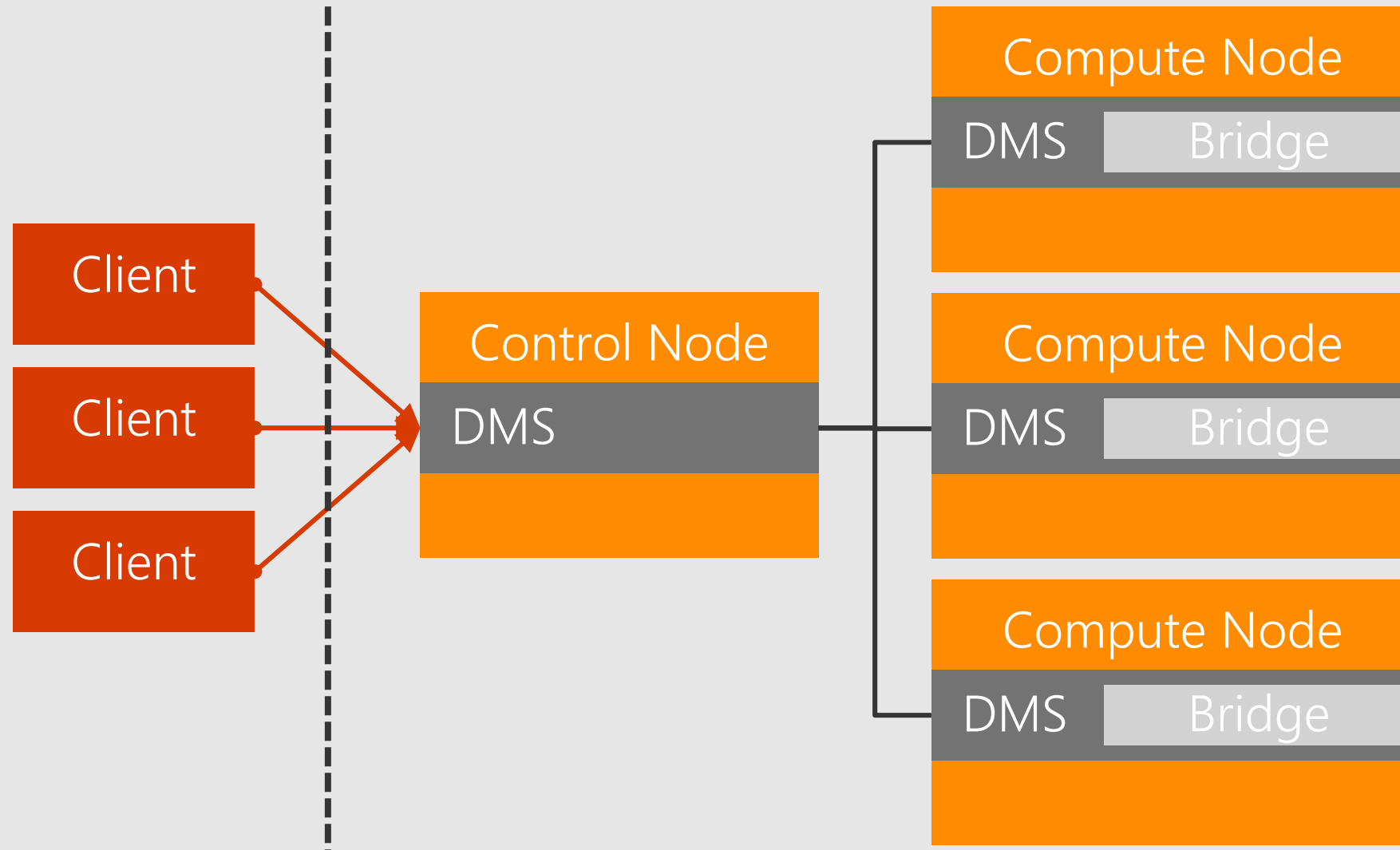
- Small number of Rows
- Commonly used for lookup / reference tasks
- Rarely updated
- Distribution = REPLICATED
- Index = Clustered Index

Loading choices

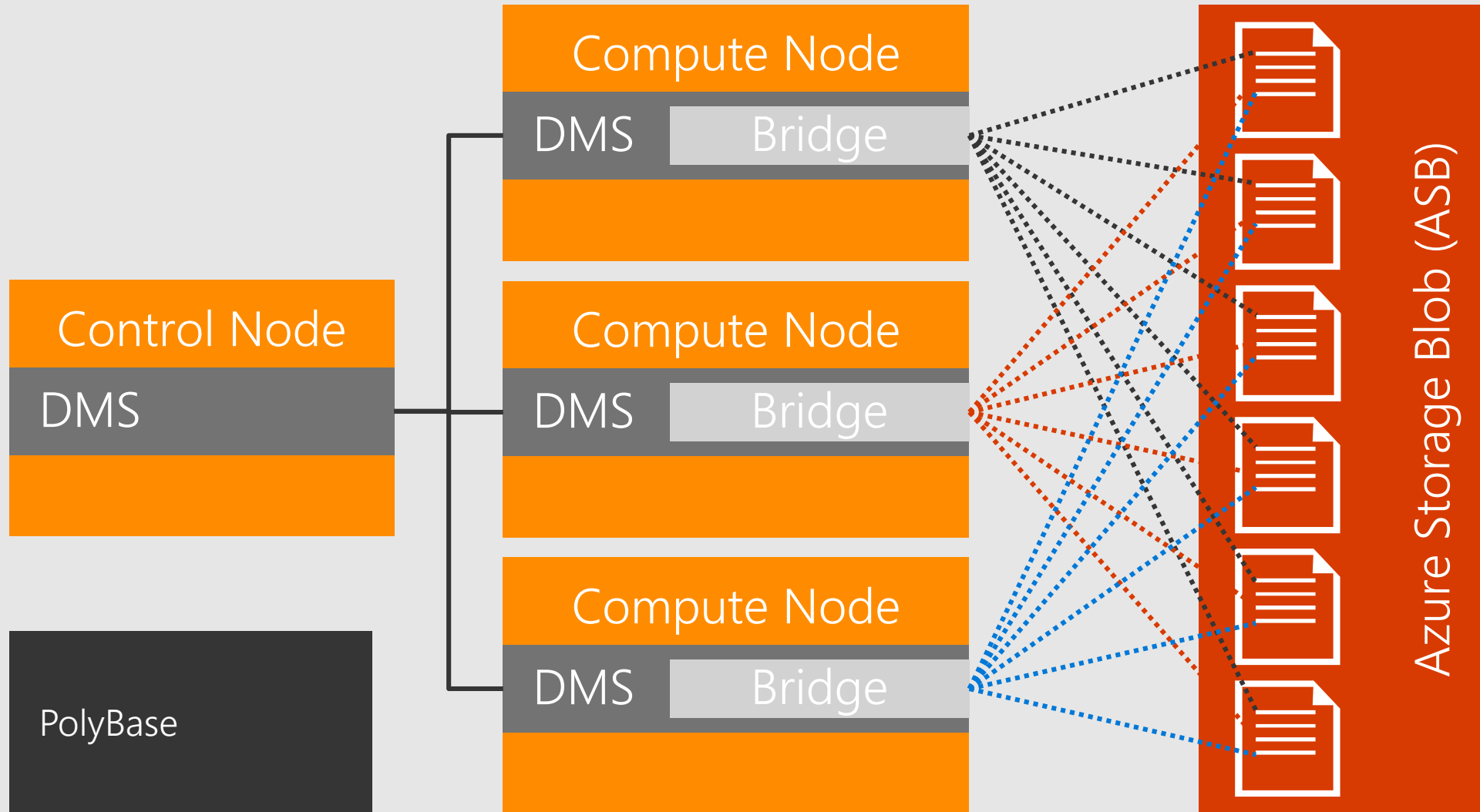
Single gated client



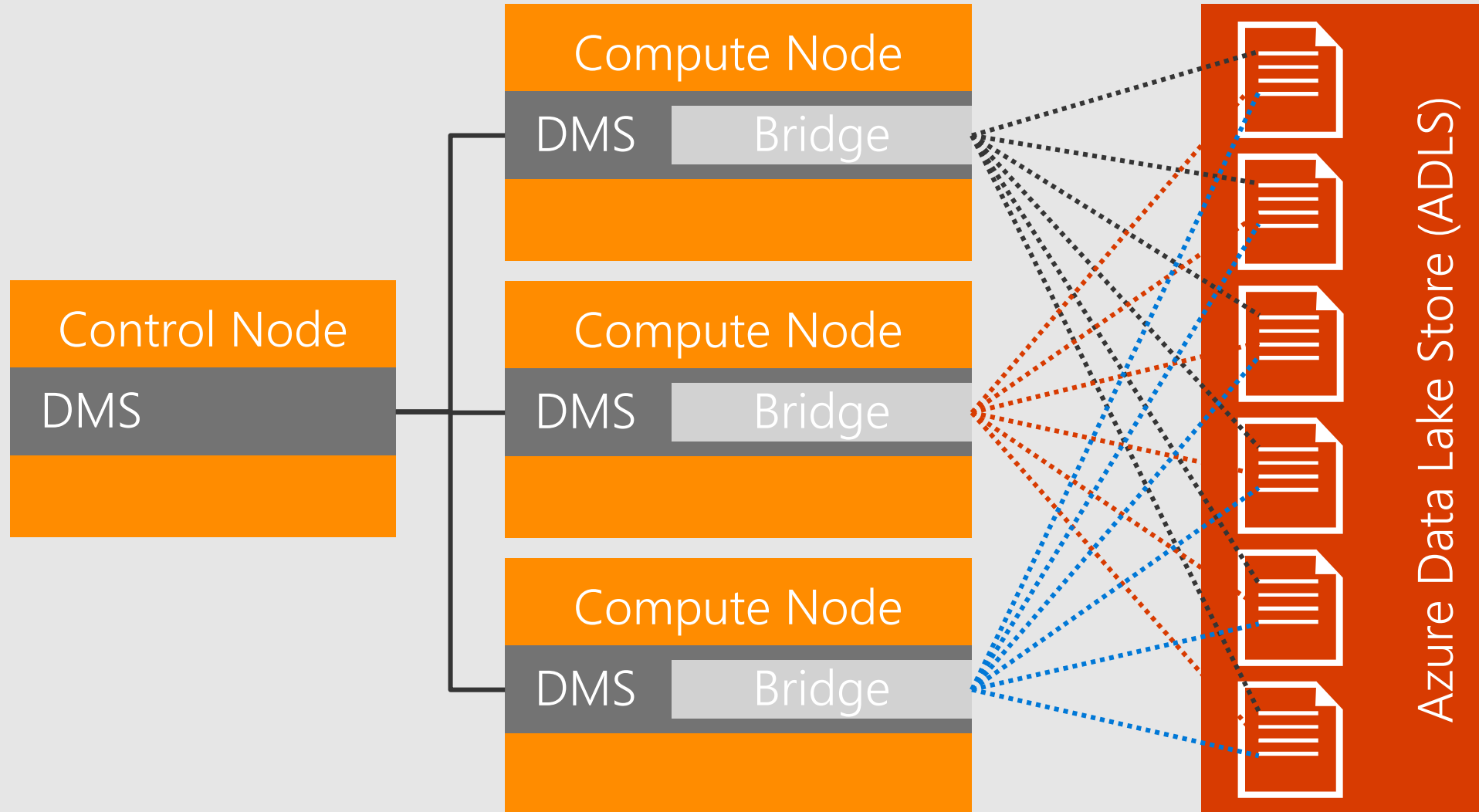
Single gated client parallelized




Parallel load via Azure Storage Blob



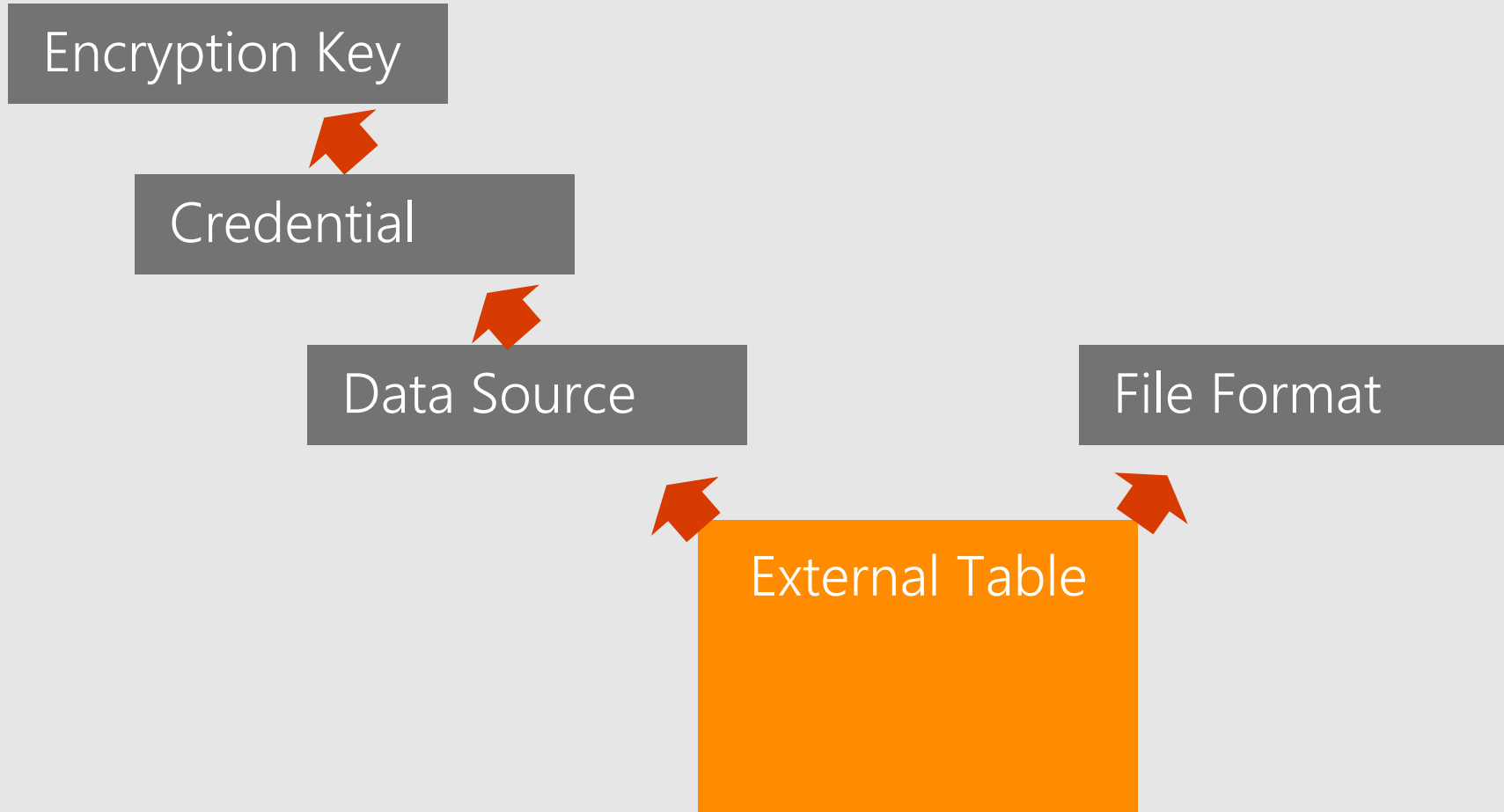
Parallel load via Azure Data Lake Store



Mechanism for loading

	PolyBase	BCP	SqlBulkCopy	SSIS
Rate	<div>Fastest  Slowest</div>			
Rate increase as DWU increases	Yes	No	No	No
Rate increases as you add concurrent load	No	Yes	Yes	Yes

External Table Object Structure



Create external tables

```
CREATE EXTERNAL DATA SOURCE WASBStor
WITH (TYPE = Hadoop,
      LOCATION = 'wasbs://<container>@<account_name>.blob.core.windows.net',
      Credential = <Database scoped credential>)
;

CREATE EXTERNAL FILE FORMAT TextFile
WITH ( FORMAT_TYPE = DELIMITEDTEXT,
      DATA_COMPRESSION = 'org.apache.hadoop.io.compress.GzipCodec',
      FORMAT_OPTIONS (FIELD_TERMINATOR = '|', USE_TYPE_DEFAULT = TRUE)
      )
;

CREATE EXTERNAL TABLE [dbo].[Customer_import] (
    [SensorKey] int NOT NULL,
    [CustomerKey] int NOT NULL,
    [Speed] float NOT NULL
)
WITH (LOCATION='<File path>',
      DATA_SOURCE = WASBStor,
      FILE_FORMAT = TextFile
)
;
```

Once per WASB container

Once per file format

File path

Load statements

CREATE TABLE AS SELECT

```
CREATE TABLE [dbo].[Customer]
WITH
(
    Distribution = ROUND_ROBIN
,
    Clustered Index (customerid)
)
AS
SELECT * FROM [dbo].[Customer_import]
```

Creates new table with specified distribution method and index.

Takes column definitions and nullability from External table

INSERT ... SELECT

```
INSERT INTO [dbo].[Customer]
SELECT * FROM [dbo].[Customer_import]
```

Inserts the data into existing relational table

Demo:
Simple table load

Lab 003 – Data Loading through Polybase

15 Mins

Load scenarios

Loading Type	Source	Concerns	Advice
Batch load	WASB/ADLS	Latency of data	Do it!
Micro batch load	WASB/ADLS	Columnstore Index compression Impact on machine resources	Make sure that loads are big enough
Streaming load Trickle load	Azure Stream Analytics, BCP	Tight coupling limits elasticity Stream load performance	Split streams to parallelize Orchestrate pause and resume with source systems

Summary

Row store or Column store?

Physical ordering of data

Frequent updates

Small dimension tables



Row store

Default Recommendation

Append oriented loads

Fact tables & large dimensions



Column store

Wrap Up

Use column store for large fact tables

Use row store for specific problems

Know your biggest join queries

Use those to figure out your hash distribution keys

Use partitioning for data lifecycle management

Distribution Guidance

For large fact tables, best option is to Hash Distribute

- Clustered Columnstore

- Distribute on column that is joined to other fact tables or large dimensions

- Primary or surrogate key maybe a good choice for distribution

However, be mindful of ...

- Hash column should have highly distinct values (Minimum 600 distinct values)

- Avoid distributing on a date column

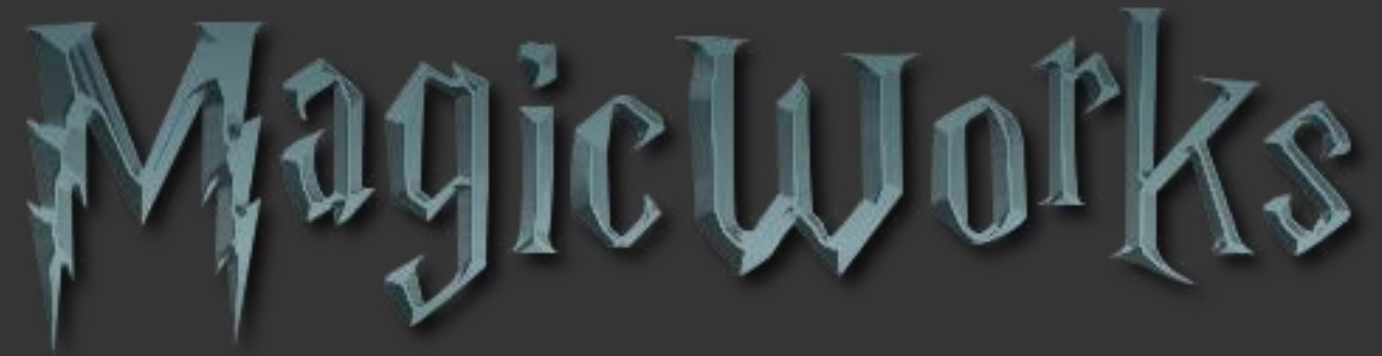
- Avoid distributing on column with high frequency of NULLs and default values (e.g. -1)

- Distribution column is NOT updatable

- For compatible joins use the same data types for two distributed tables

If there are no distribution columns that make sense, then use Round Robin as last resort

Let's Get Loading



Blob Location: `wasb://<container>@<account>.blob.core.windows.net`

Git Repo Location: `http://gitrepo>LoadingScripts`