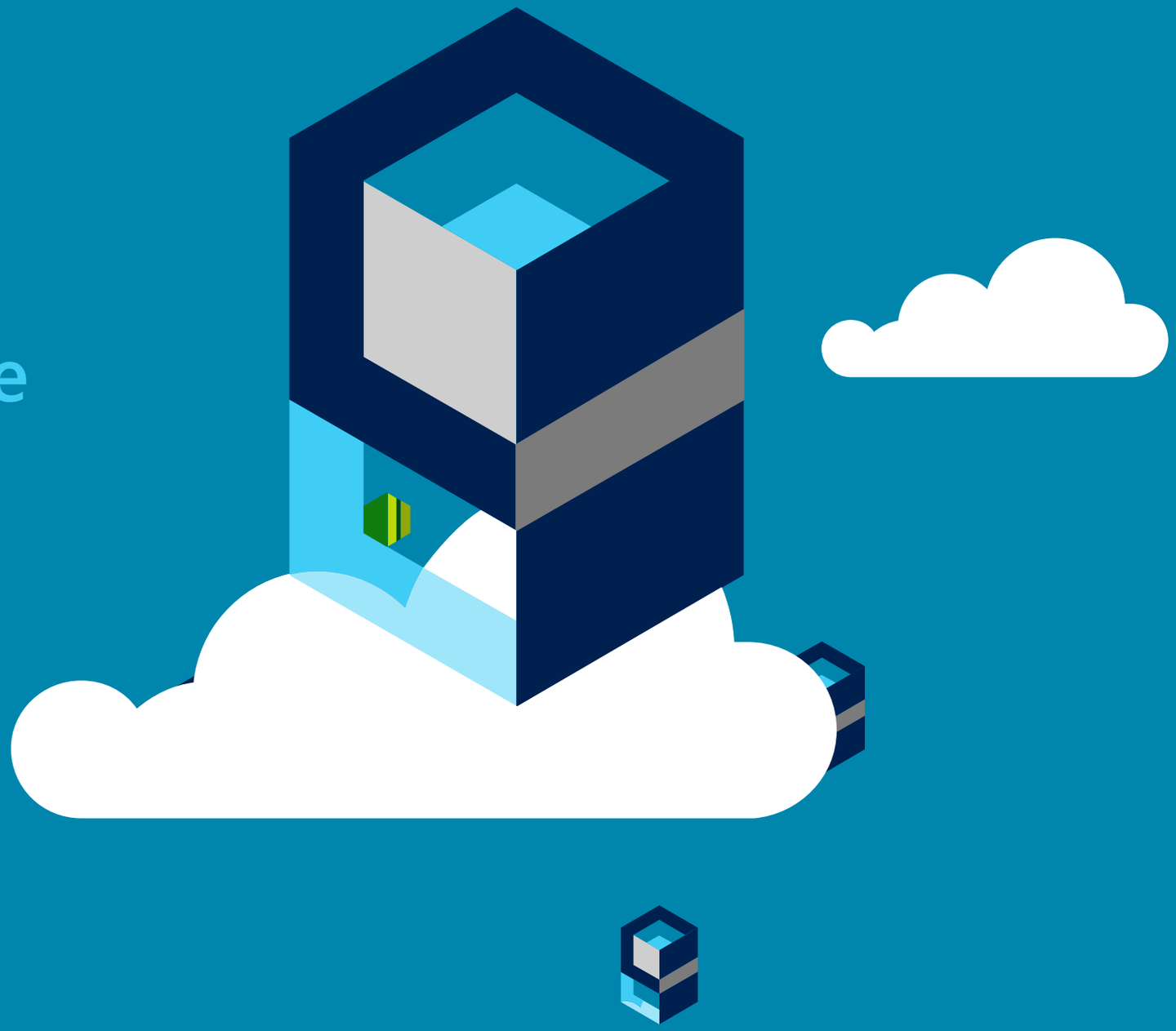
Microsoft

Modernizing Your Data Warehouse



Microsoft

SQL DW Operational Best Practices

Kal Yella

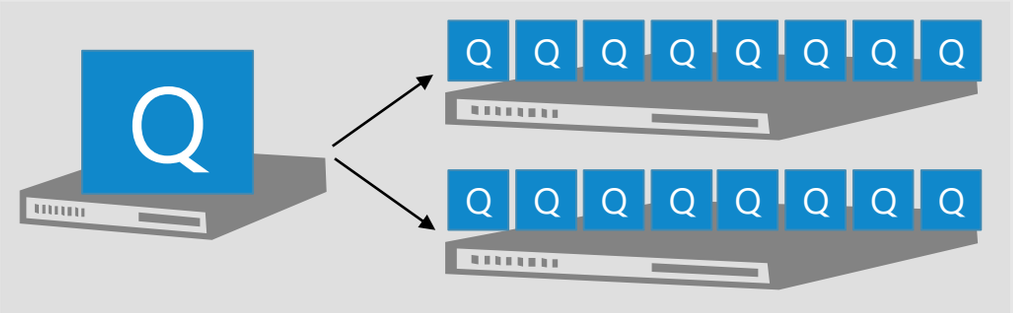
kalyany@Microsoft.com

Data and Compute are Distributed!

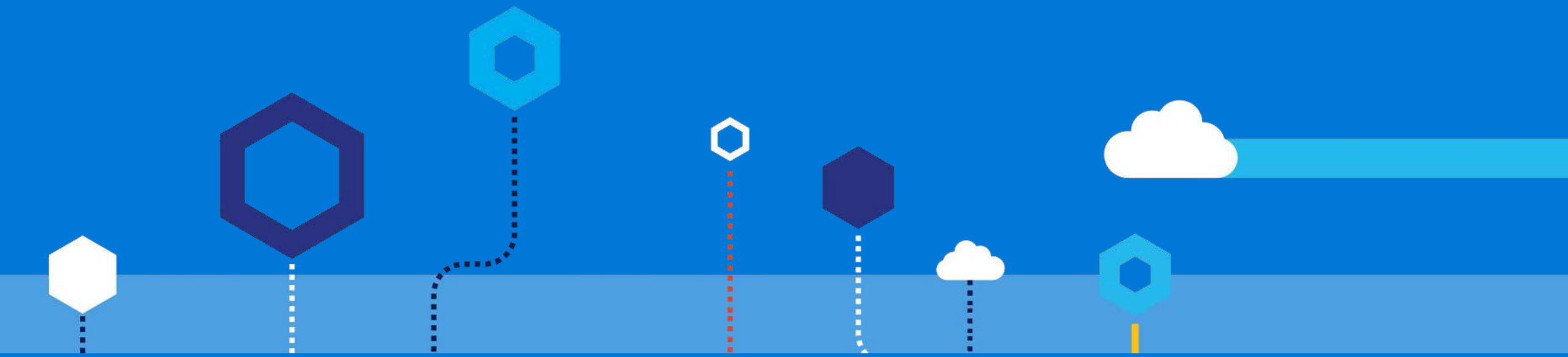
- Massively Parallel Processing (MPP)
  - Multiple compute nodes with dedicated CPU, memory, storage
  - Handles and hides query complexity
- Good for scalability
  - Data is spread (distributed) across servers
- Introduces overheads
  - Data is not all in the same place!
  - Don't know where specific values are located
  - May need to move the data then process it

SMP Query Execution

MPP Query Execution



Data Movement



Why data moves

Data has to be co-located to be operated on…

Common reasons:

- Incompatible join
- Incompatible aggregation

Distributed Data Movement

ProductSales

	AccountID	SalesAmt	…
Node 1:	47	$1,234.36	…
Node 2:	36	$2,345.47	…
Node 3:	14	$3,456.58	…
Node 4:	25	$4,567.69	…
Node 5:	48	$5,678.70	…
Node 6:	37	$6,789.81	…
	…	…	…

SalesAccountTerritory

SATerritoryID	AccountID	…
444	37	…
333	25	
111	36	…
222	47	…
445	14	
334	48	…
…	…	…

Shuffle

SATName	TotalSales
North	$6,789.81
South	$5,678.70
NorthEast	$4,567.69
SouthWest	$3,456.58
East	$2,345.47
West	$1,234.36

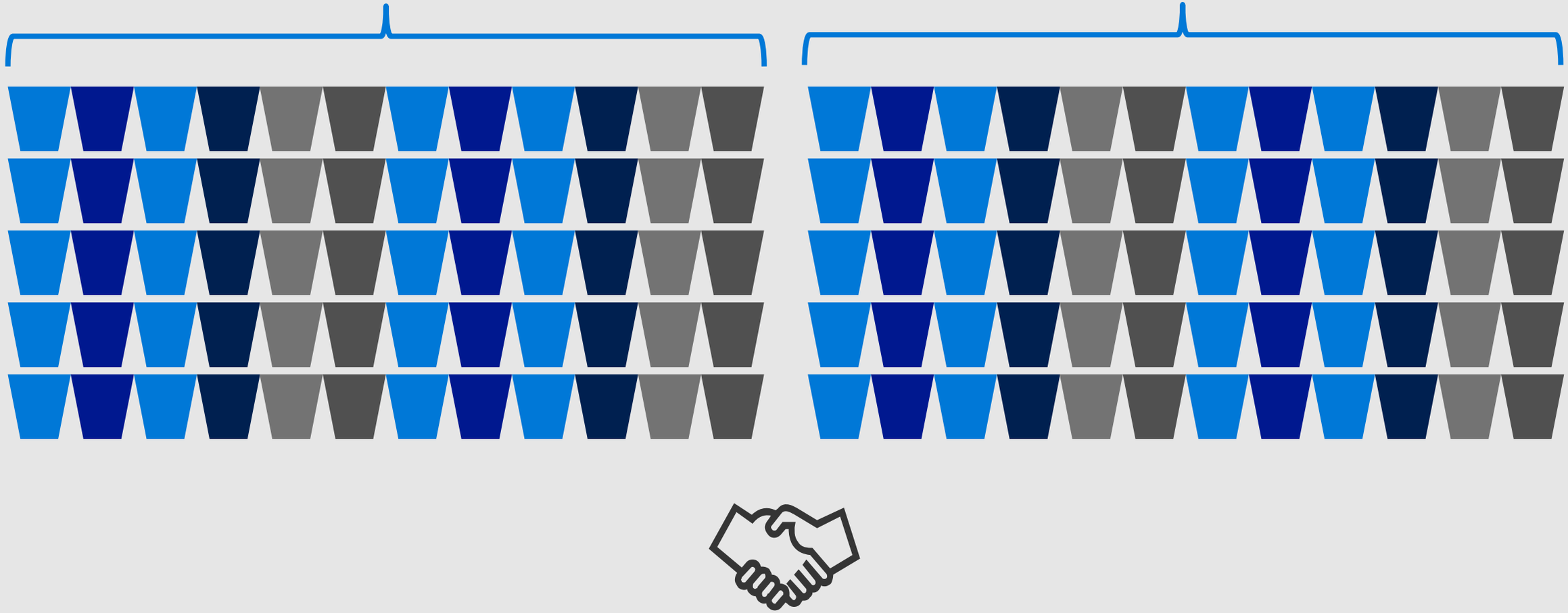
D	SATName	…
	West	…
	East	
	SouthWest	…
	NorthEast	…
	South	
37	North	…
…	…	…

```
CREATE TABLE ProductSales
WITH (DISTRIBUTION=HASH(AccountID))
AS...

CREATE TABLE SalesAccountTerritory
WITH (DISTRIBUTION=HASH(SATerritoryID))
AS...

SELECT TOP 25 a.SalesAccountTerritoryName
      ,TotalSales = SUM(p.SalesAmt)
FROM ProductSales p
JOIN SalesAccountTerritory a
ON   a.AccountID = p.AccountID
GROUP BY a.SalesAccountTerritoryName
ORDER BY 2 DESC
```

Joining HASH tables

Store_Sales HASH([ProductKey])
[ProductKey] **INT** NULL

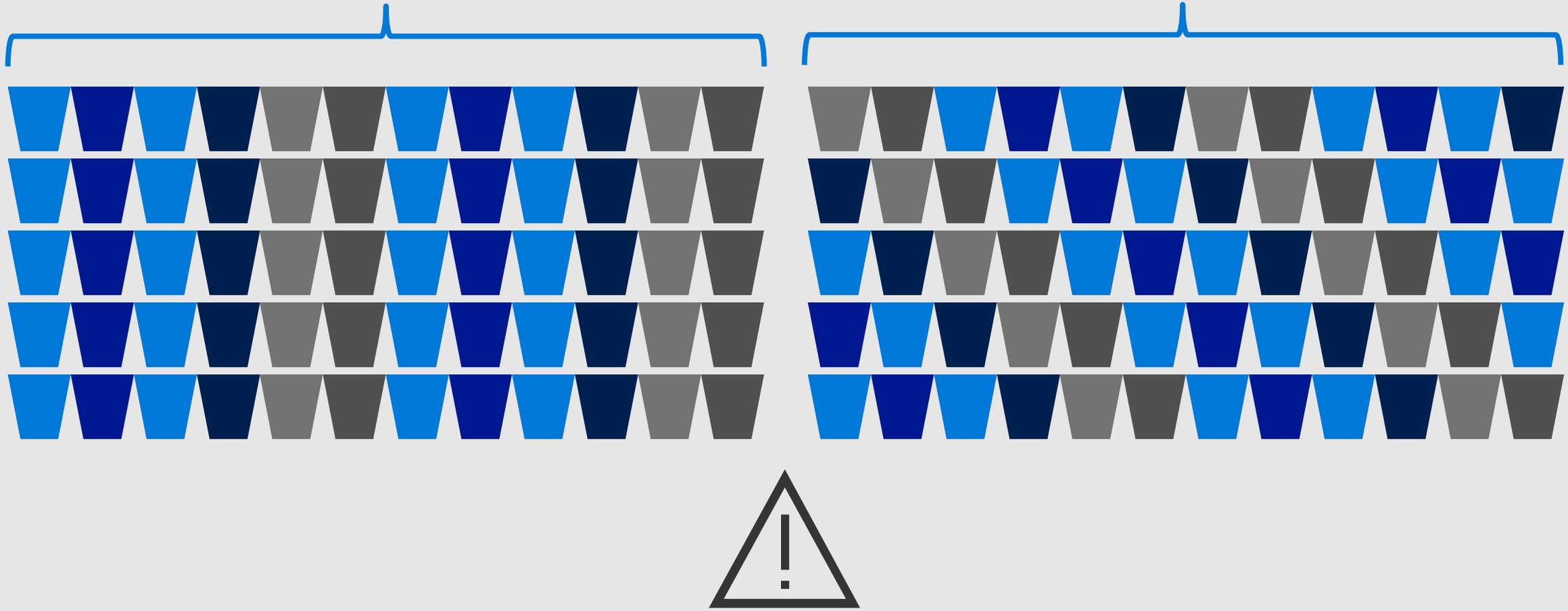
Web_Sales HASH([ProductKey])
[ProductKey] **INT** NULL



Joining HASH tables

Store_Sales HASH([ProductKey])
[ProductKey] **INT** NULL

Web_Sales HASH([ProductKey])
[ProductKey] **BIGINT** NULL



Aggregation - compatible

Resolved completely on each compute node

No Data Movement

1. Hash Distribution Key is contained in the group by keys
2. Count Distinct on distribution key

```
-- FactOnlineSales distributed on ProductKey

SELECT      COUNT_BIG(*)
FROM        [cso].[FactOnlineSales]
GROUP BY    [ProductKey]

SELECT  COUNT_BIG(DISTINCT ([ProductKey]))
FROM    [cso].[FactOnlineSales]
```

Aggregation - Incompatible

Partially aggregated on each node

Shuffle move co-locates rows with same group by key

1. Table is round robin distributed
2. Hash Distribution key is not contained in group by keys
3. Count Distinct on non-distribution key or on round robin table

```
-- FactOnlineSales distributed on ProductKey

SELECT      COUNT_BIG(*)
FROM        [cso].[FactOnlineSales]
GROUP BY    [StoreKey]

SELECT  COUNT_BIG(DISTINCT [DateKey])
FROM    [cso].[FactOnlineSales]
```

Data Movement Types for a Query

DMS Operation	Description
ShuffleMoveOperation	Distribution → Hash algorithm → New distribution Changing the distribution column in preparation for join.
PartitionMoveOperation	Distribution → Control Node Aggregations - count(*) is count on nodes, sum of count
BroadcastMoveOperation	Distribution → Copy to all distributions Changes distributed table to replicated table for join.
TrimMoveOperation	Replicated table → Hash algorithm → Distribution When a replicated table needs to become distributed. Needed for outer joins.
MoveOperation	Control Node → Copy to all distributions Data moved from Control Node back to Compute Nodes resulting in a replicated table for further processing.

Distribution Guidance

For large fact tables, best option is to Hash Distribute

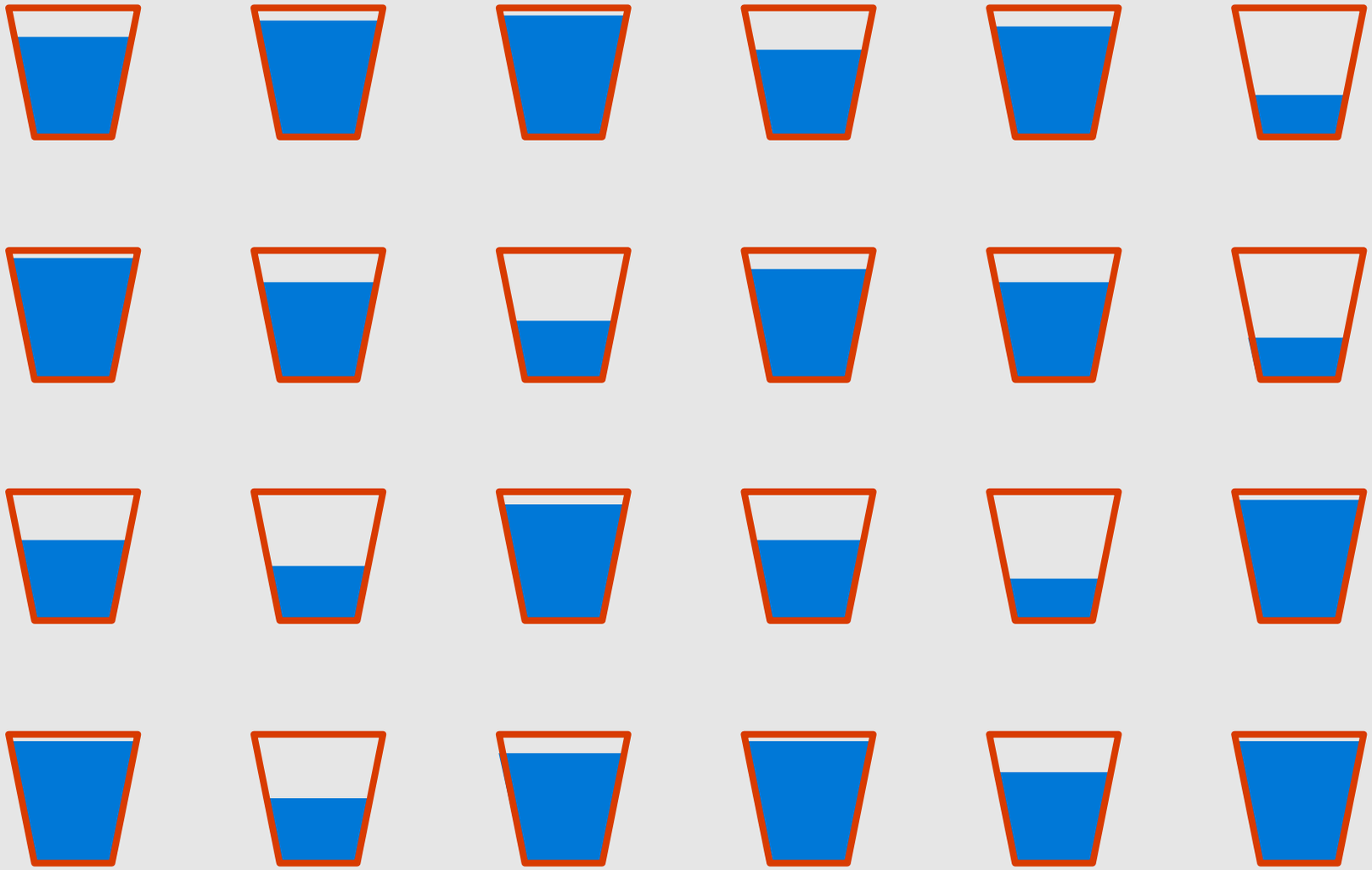
- Clustered Columnstore
- Distribute on column that is joined to other fact tables or large dimensions
- Primary or surrogate key maybe a good choice for distribution

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

Skewed Distribution



Finding Skew

DBCC PDW_SHOWSPACEUSED

Not a programmatic interface

DMV gives more detail and control

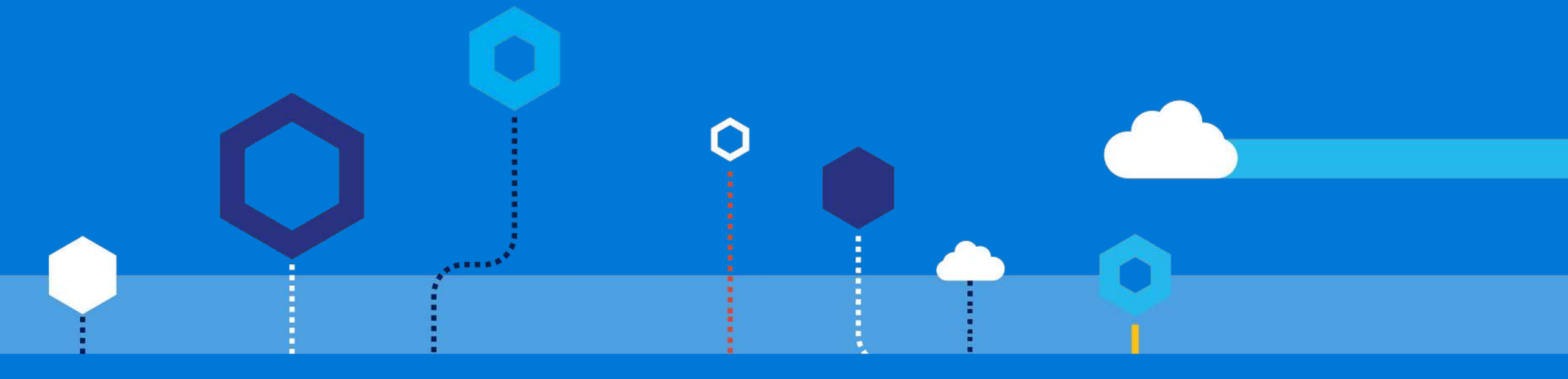
sys.dm_pdw_nodes_db_partition_stats

Refer to the view dbo.vTableSizes provided in docs (Table Size Queries)

```
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;
```

Azure Data Studio

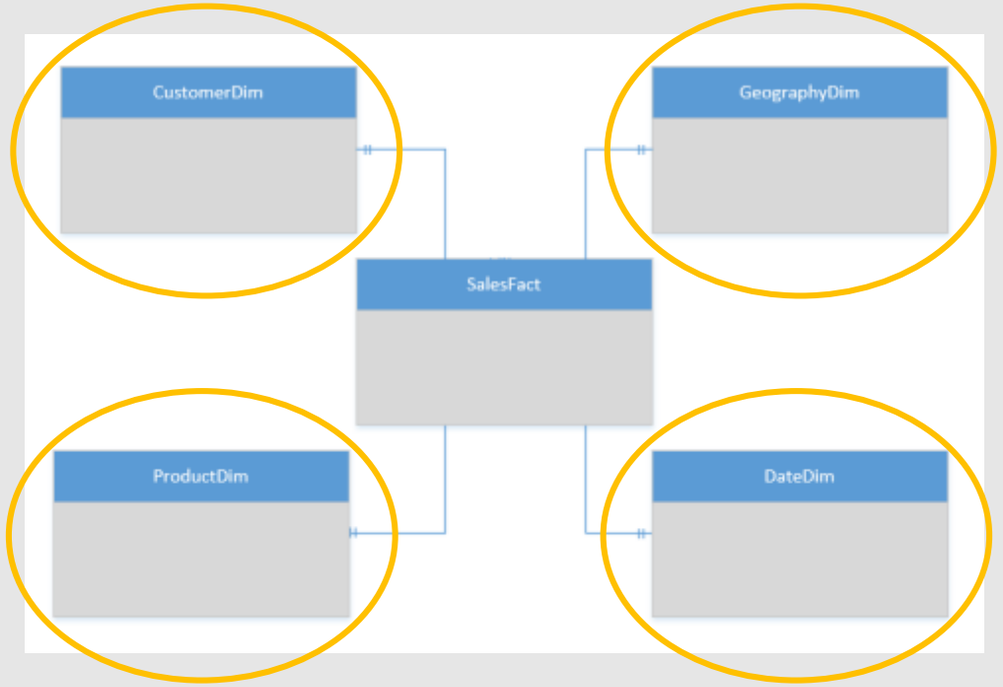
Replicated Tables



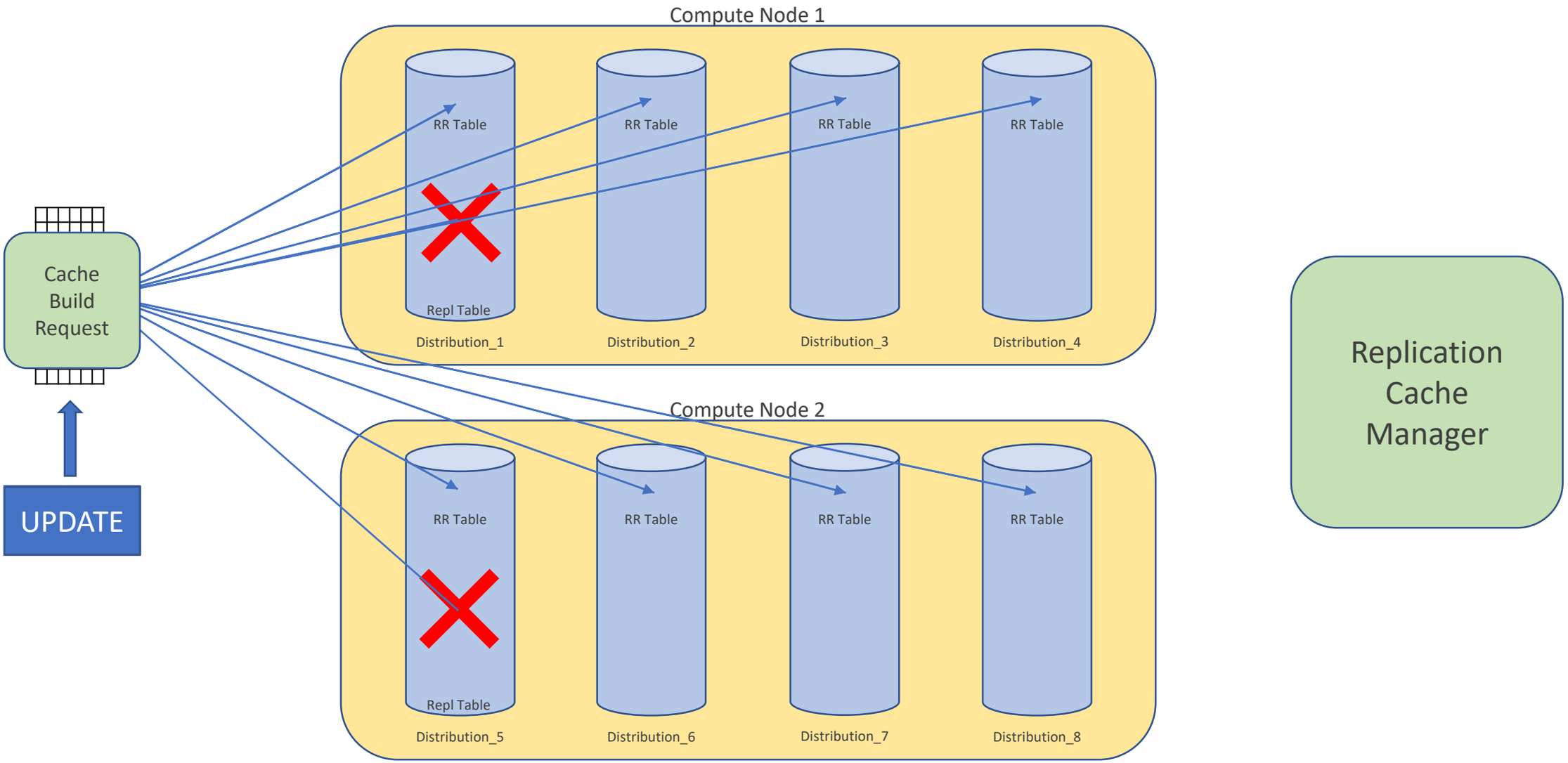
Replicated Table Scenarios

Scenarios to consider using Replicated tables:

- Star schema reporting
  - Dimensions – descriptive entities about fact data
- ETL master data, common domain data used during transaction loading



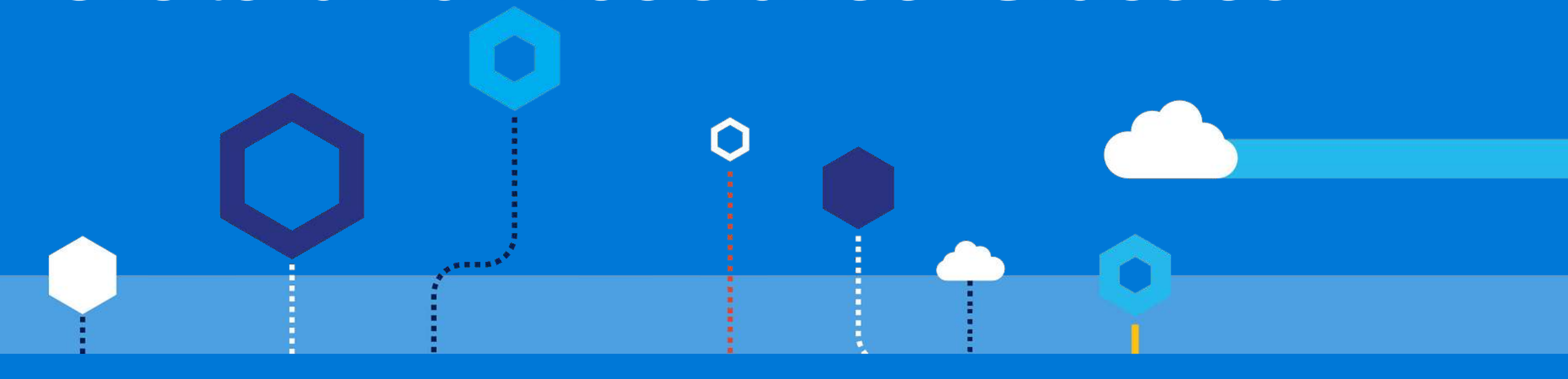
Replicated Tables… How it works



Replicated Table Best Practices

- Good for
  - Tables used frequently in Joins
  - Tables size less than 2GB on disk
  - Queries with simple predicates (Ex: Equal, not equal)
- Not good for
  - Tables with frequent modifications
  - Frequent scaling of DW
  - Large Tables (>2GB)
  - Tables with large number of columns (but query small number of columns)

Concurrency, Concurrency Slots and Resource Classes

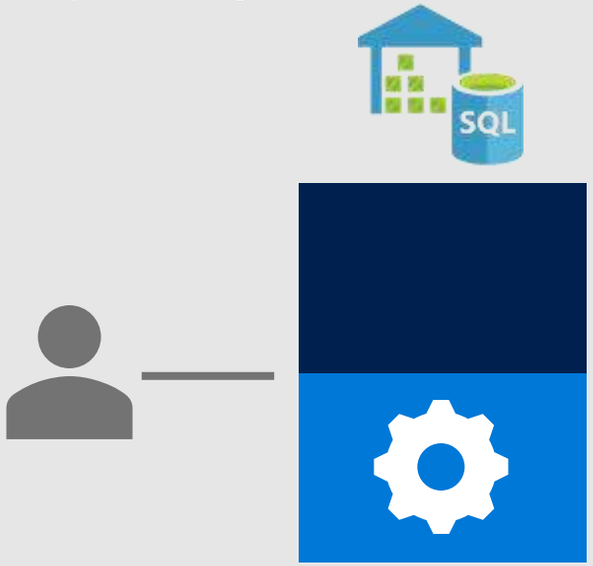


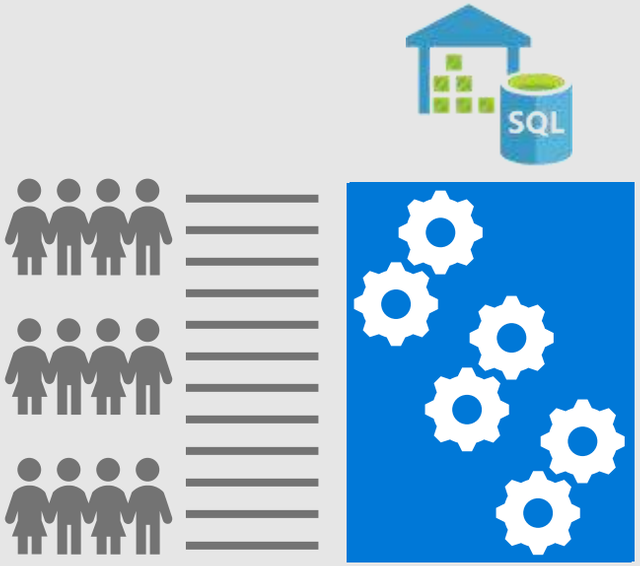
Importance of concurrency

Parallel queries = more throughput (even on large, scale-out DWs)

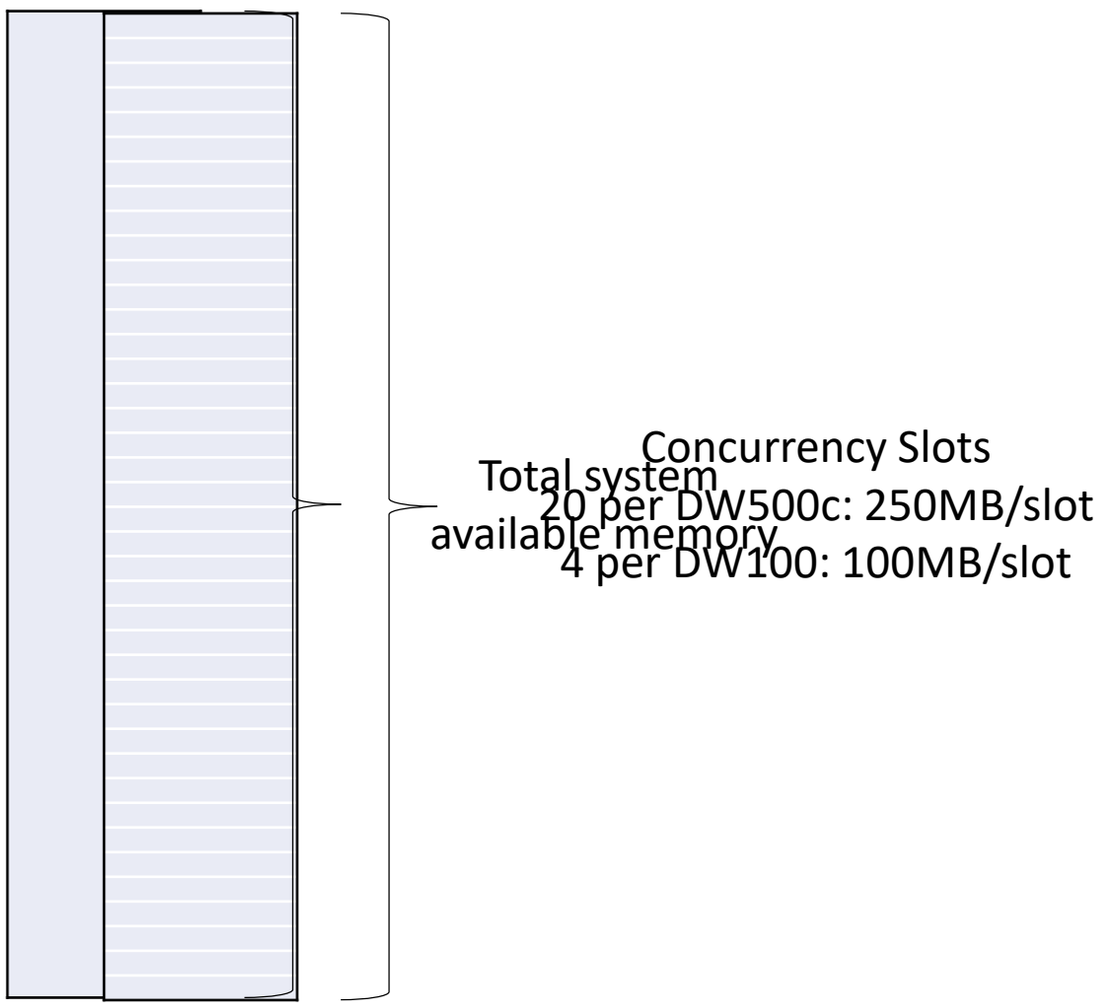
Mixed workloads (transform, load, export, query)

A single query is not expected to consume all resources





Concurrency slots



Dynamic Resource Class Example at DW3000c

System	Concurrency Slots
Concurrency Slots	120
Query Memory, GB	1,800

Resource Class	Concurrency Slots	Query Memory, GB	Concurrency
SmallRc	3	45	32
MediumRc	12	180	10
LargeRc	26	390	4
XLargeRc	84	1,260	1

Dynamic Resource Class Example at DW6000c

System	Concurrency Slots
Concurrency Slots	240
Query Memory, GB	3,600

Resource Class	Concurrency Slots	Query Memory, GB	Concurrency
SmallRc	7	105	32
MediumRc	24	360	10
LargeRc	52	780	4
XLargeRc	168	2,520	1

Static Resource Class Example at DW3000c

System	Concurrency Slots
Concurrency Slots	120
Query Memory, GB	1,800

Resource Class	Concurrency Slots	Query Memory, GB	Concurrency
StaticRc10	1	15	64
StaticRc20	2	30	60
StaticRc30	4	60	30
StaticRc40	8	120	15
StaticRc50	16	240	7
StaticRc60	32	480	3
StaticRc70	64	960	1
StaticRc80	64	960	1

Static Resource Class Example at DW6000c

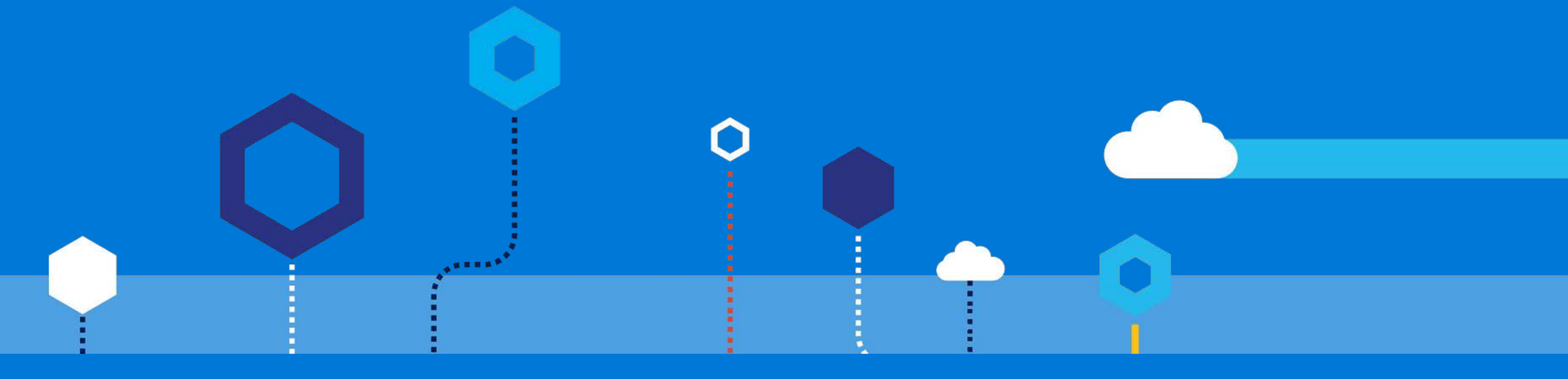
System	Concurrency Slots
Concurrency Slots	240
Query Memory, GB	3,600

Resource Class	Concurrency Slots	Query Memory, GB	Concurrency
StaticRc10	1	15	128
StaticRc20	2	30	120
StaticRc30	4	60	80
StaticRc40	8	120	30
StaticRc50	16	240	15
StaticRc60	32	480	7
StaticRc70	64	960	3
StaticRc80	128	1,920	1

Resource Class Best Practices

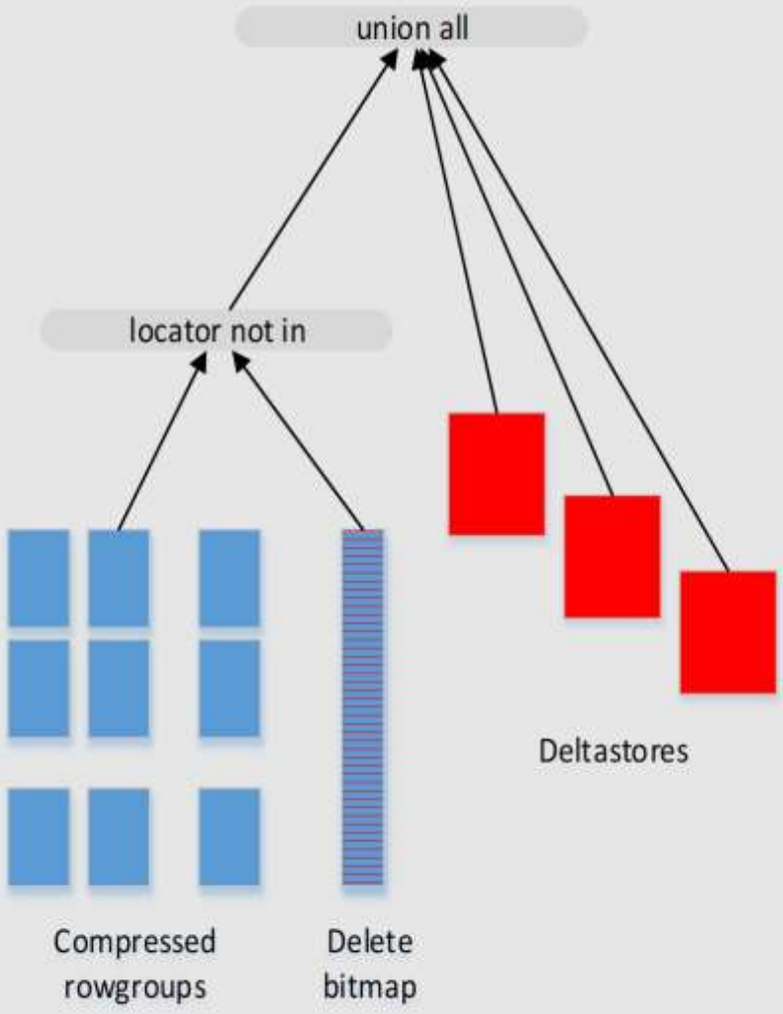
- Be aware of the user binding
- Use right resource class for right workload
- Prefer Static for higher concurrency
- Prefer Dynamic for increased perf with scale
- Be aware of the RC precedence
  - Dynamic over Static
  - Higher RC over lower
- Explicitly remove user from unnecessary RCs
- Use custom coding for automating RC assignment

Indexes/Stats



CCI Best Practices

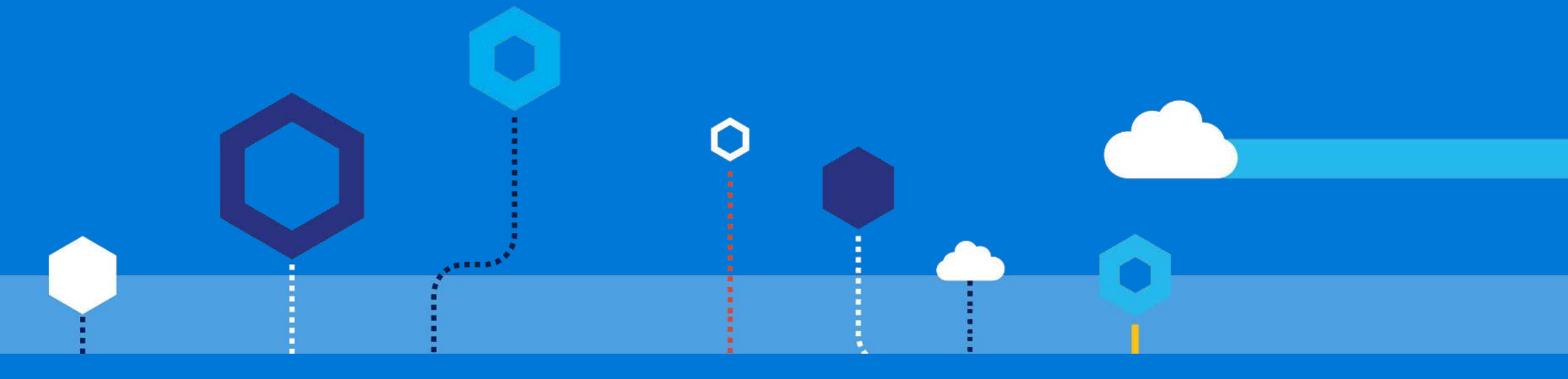
- Be aware of the default
- Not efficient for
  - transient data (frequent updates/deletes)
  - Singleton loads or micro-batches
  - Small tables (<100 Million rows)
- >100K rows per rowgroup
- Highest possible RC for loading
  - Calculate memory requirements
    - 72MB+(#rows*#columns*8B)+(#rows*#SSC*32B)+(#LSC*16MB)
- Reduce memory requirements
  - Small number of columns/table
  - Reduce columns with string data type
  - Do not over partition
  - Simplify load queries
  - Adjust MAXDOP (if needed)



Index/Stats Best Practices

- Perform regular CCI health checks to monitor
  - # of open row groups
  - # of rows per row group (100 K to 1 Million)
  - Reason for trimming
- Perform regular Index Maintenance
  - Rebuild/Reorganize
  - Partition Rebuild
  - CTAS/Partition switch if Needed
- Statistics
  - Use auto-create stats option
  - Create multi-column stats as needed
  - Update stats immediately after large data modifications
  - Auto-update stats option (coming soon)

Monitoring



Monitoring Options

- Azure Monitor
  - Insights
  - Alerts
  - Dashboard
  - Views
  - PBI Integration
- Query Store
- Operations Management Suite (OMS) integration

Command-line

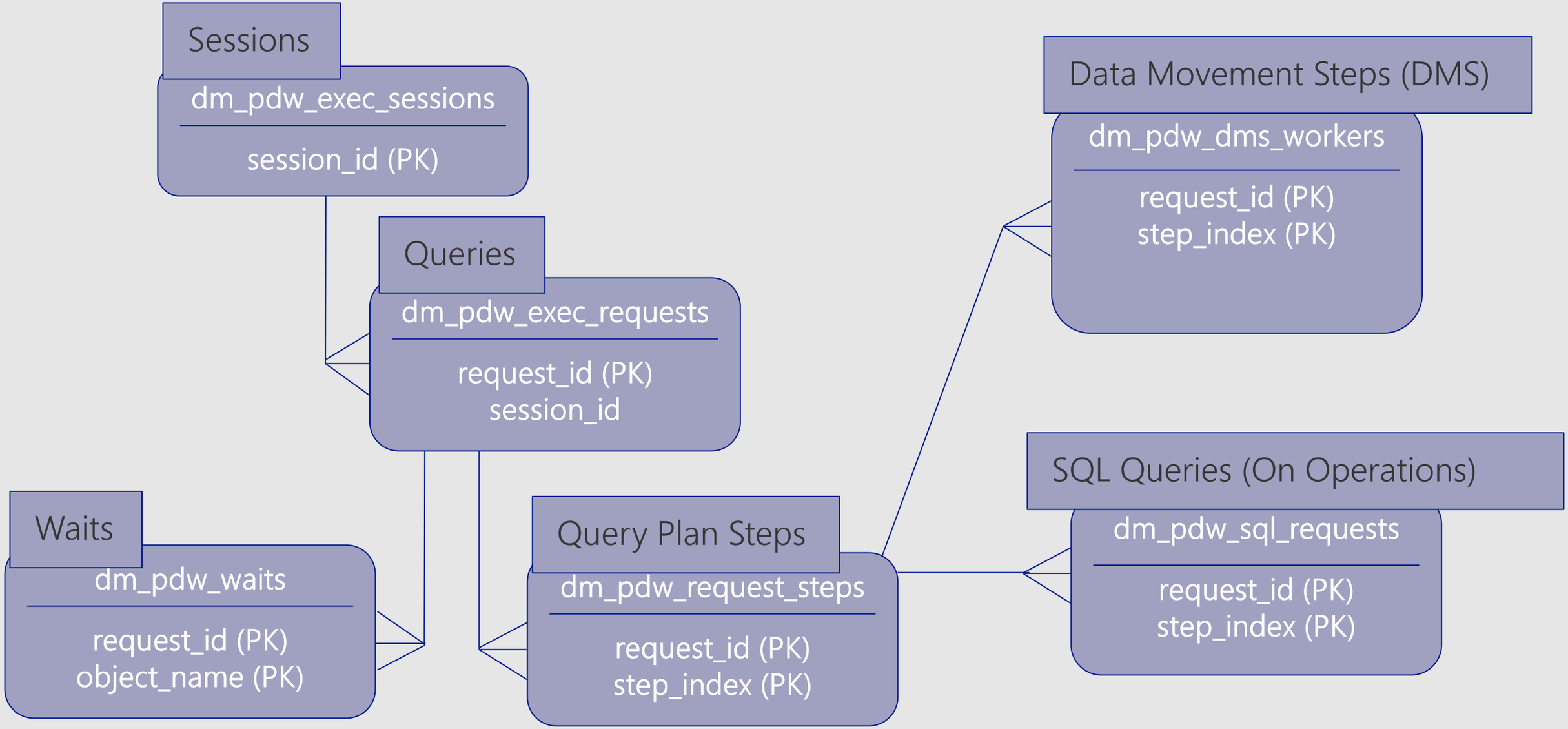
- Rich set of DMVs
- Lower level of diagnostics

Azure Portal

Key Metrics

- CPU Percentage
- IO Percentage
- DWU (limit, used and percentage used)
- Connections (successful, failed, blocked by firewall)
- Gen2: Cache metrics (used and cache hit percentage)

Execution DMVs



Microsoft

Q&A

Microsoft

Modernizing Your Data Warehouse