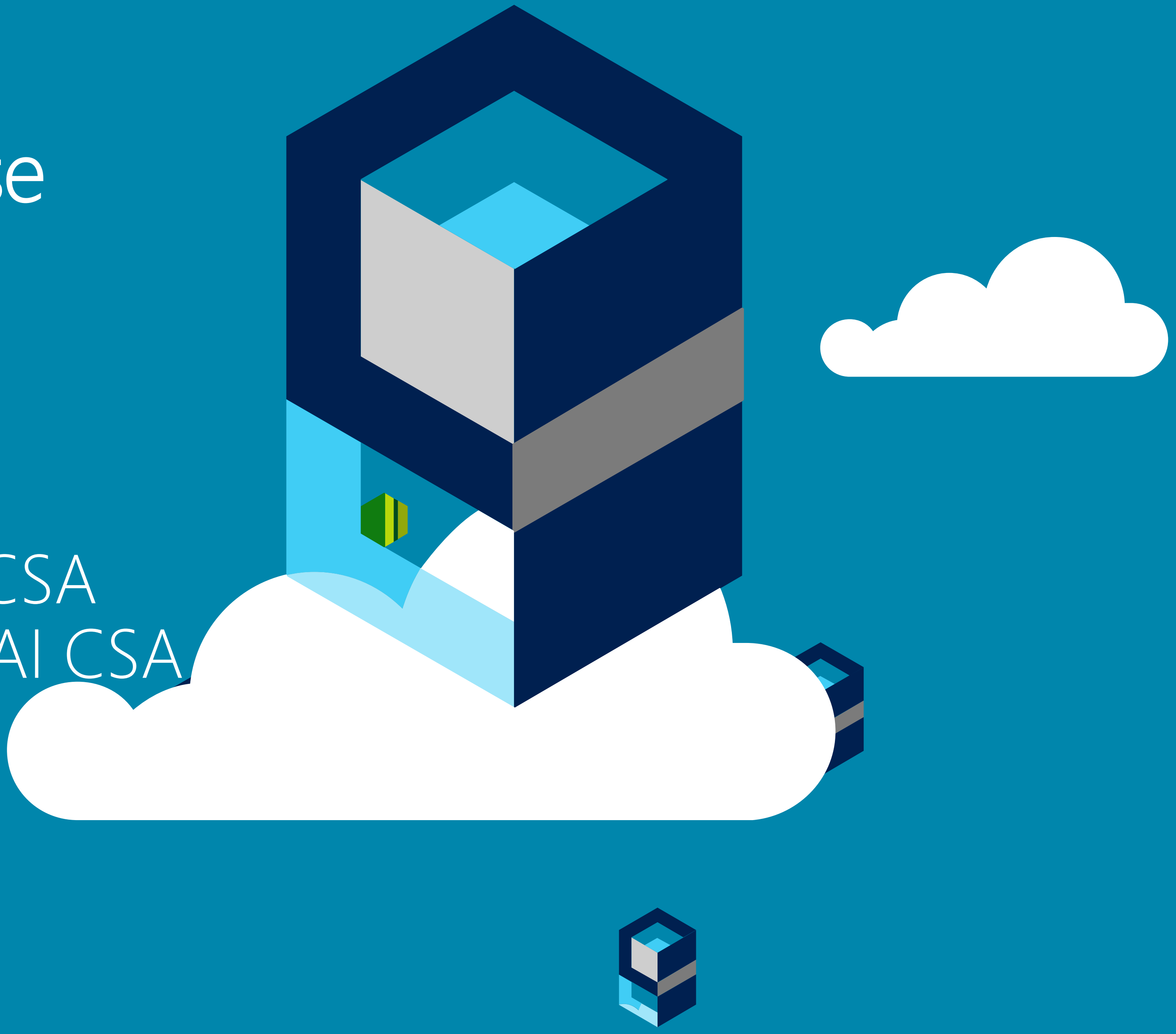
04.SQL DW Operational best practices

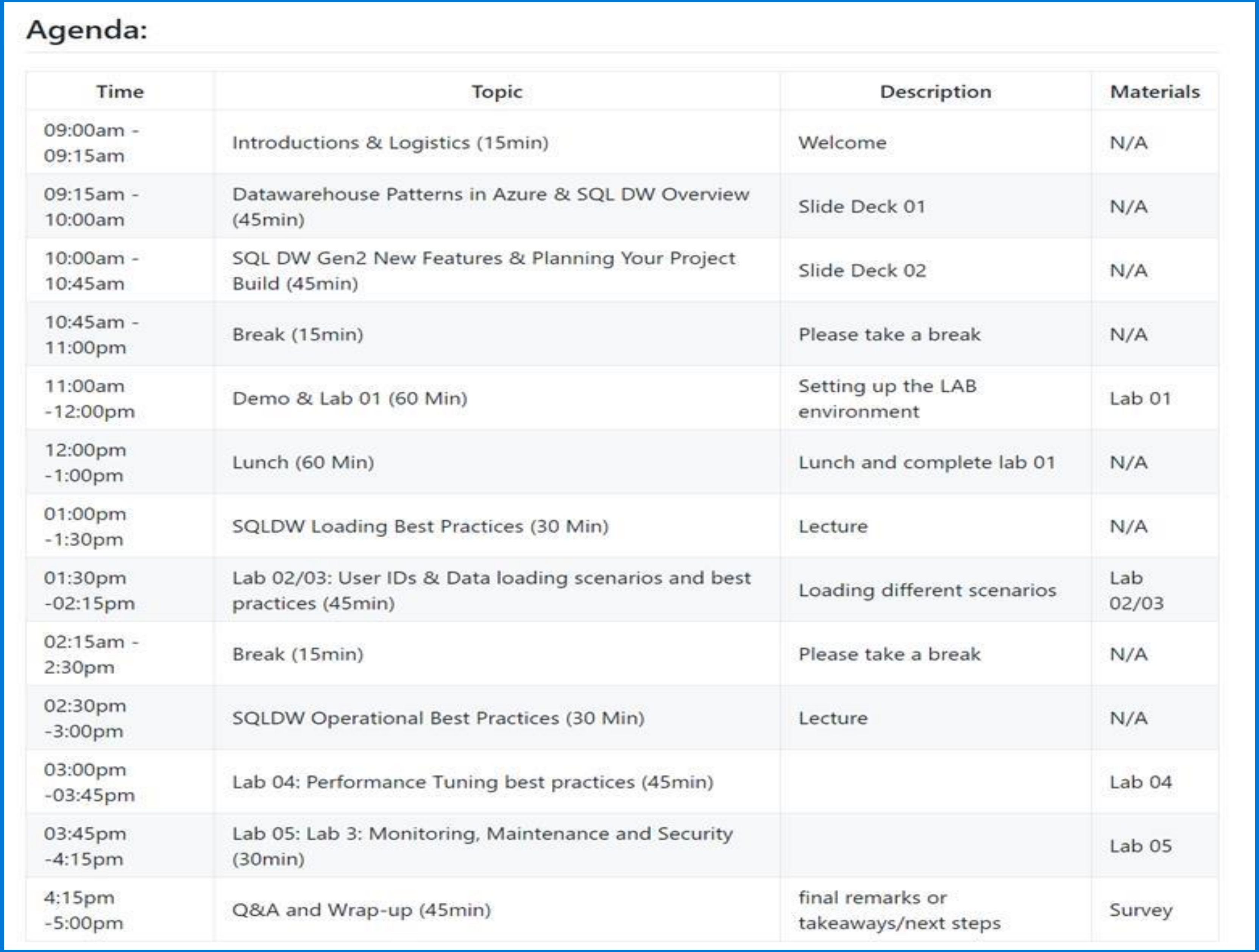
Microsoft

Azure Data Warehouse
In-A-Day



Steve Young - Data & AI CSA
Rebecca Young - Data & AI CSA

Agenda



Agenda:

Time	Topic	Description	Materials
09:00am - 09:15am	Introductions & Logistics (15min)	Welcome	N/A
09:15am - 10:00am	Datawarehouse Patterns in Azure & SQL DW Overview (45min)	Slide Deck 01	N/A
10:00am - 10:45am	SQL DW Gen2 New Features & Planning Your Project Build (45min)	Slide Deck 02	N/A
10:45am - 11:00pm	Break (15min)	Please take a break	N/A
11:00am -12:00pm	Demo & Lab 01 (60 Min)	Setting up the LAB environment	Lab 01
12:00pm -1:00pm	Lunch (60 Min)	Lunch and complete lab 01	N/A
01:00pm -1:30pm	SQLDW Loading Best Practices (30 Min)	Lecture	N/A
01:30pm -02:15pm	Lab 02/03: User IDs & Data loading scenarios and best practices (45min)	Loading different scenarios	Lab 02/03
02:15am - 2:30pm	Break (15min)	Please take a break	N/A
02:30pm -3:00pm	SQLDW Operational Best Practices (30 Min)	Lecture	N/A
03:00pm -03:45pm	Lab 04: Performance Tuning best practices (45min)		Lab 04
03:45pm -4:15pm	Lab 05: Lab 3: Monitoring, Maintenance and Security (30min)		Lab 05
4:15pm -5:00pm	Q&A and Wrap-up (45min)	final remarks or takeaways/next steps	Survey

SQL DW Operational best practices

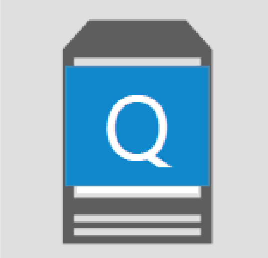
Topics

- Data Movement
- Replicated Tables
- Concurrency, Concurrency Slots and Resource Classes
- Index/Stats
- Monitoring

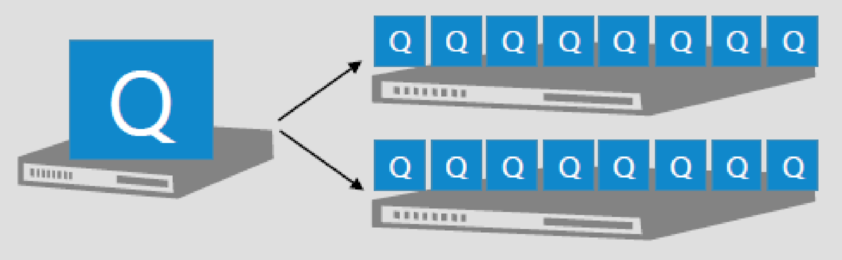
Data and Compute are Distributed

- 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

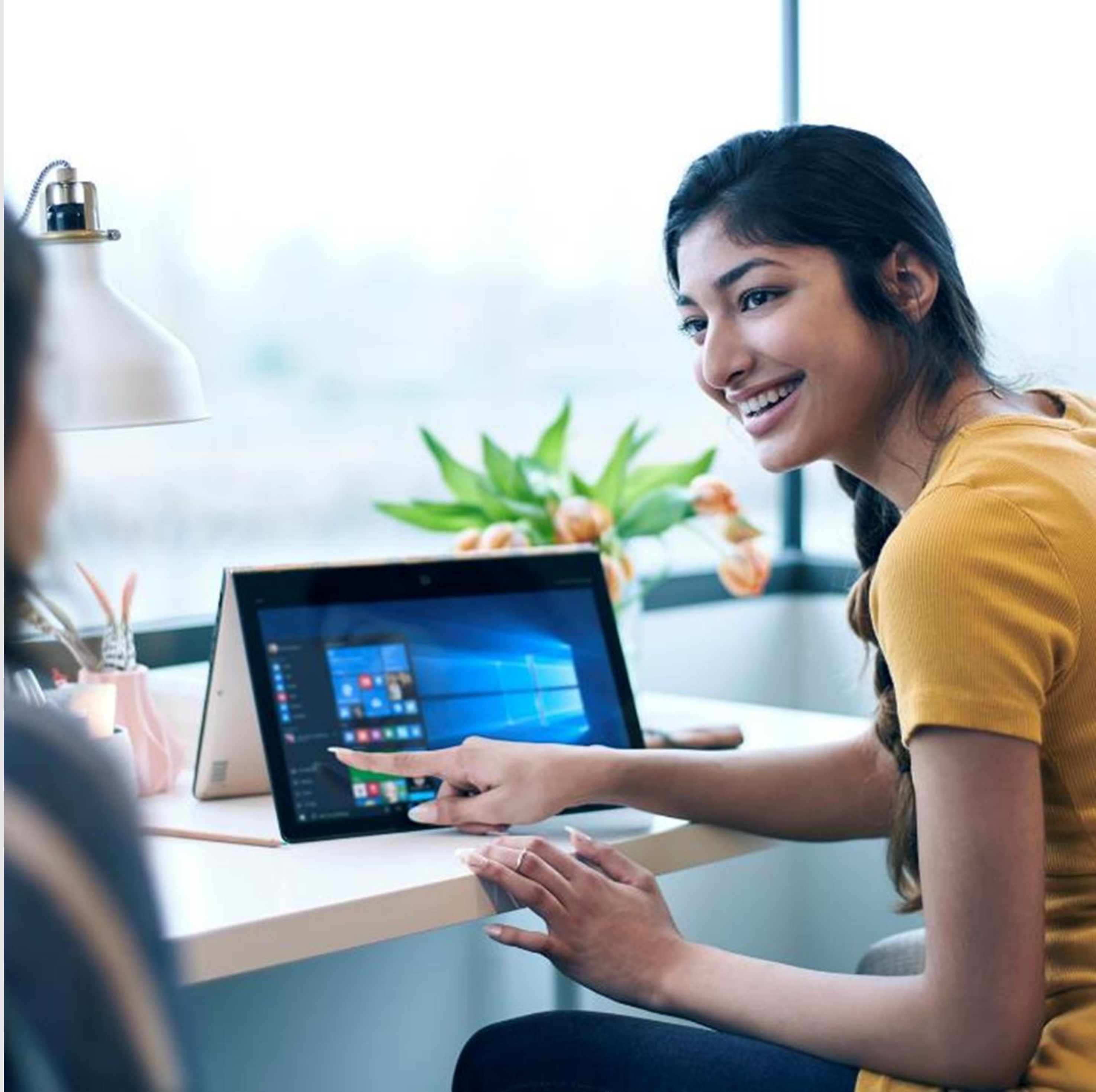
SMP Query Execution



MPP Query Execution



Data Movement



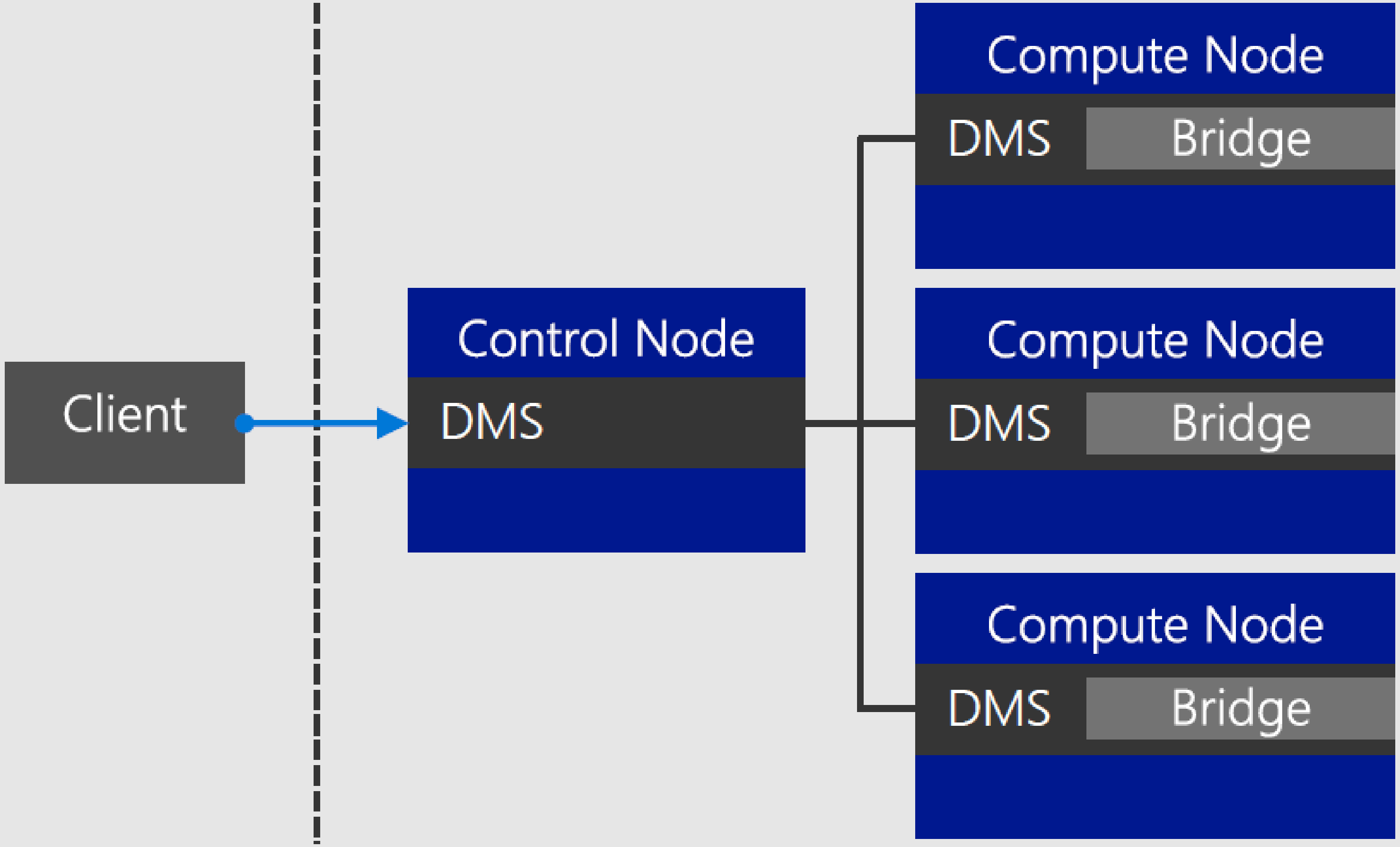
Why Data Moves

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

Common reasons:

- Incompatible join
- Incompatible aggregation

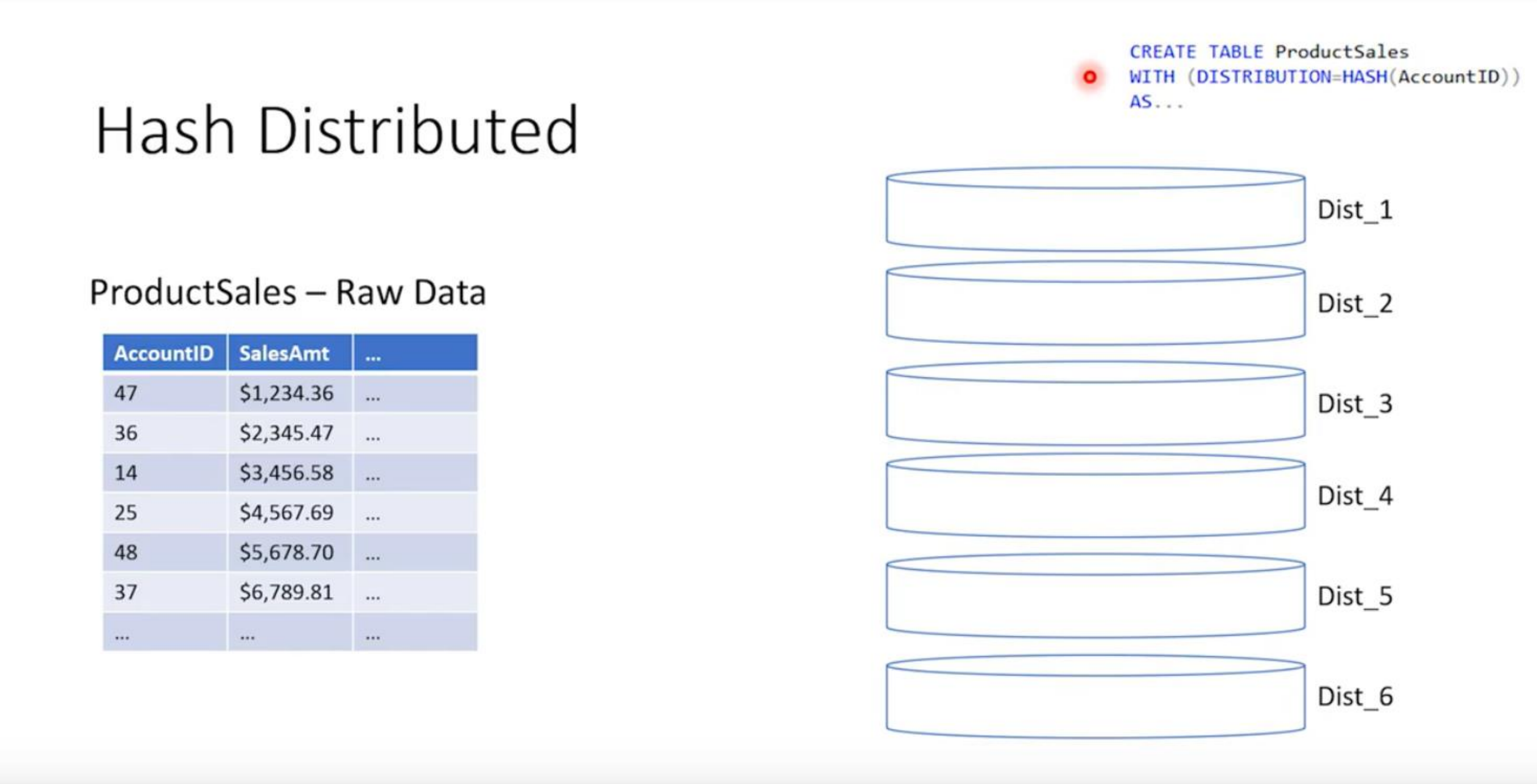
Why Data Moves



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.

Hash Distributed Data Movement



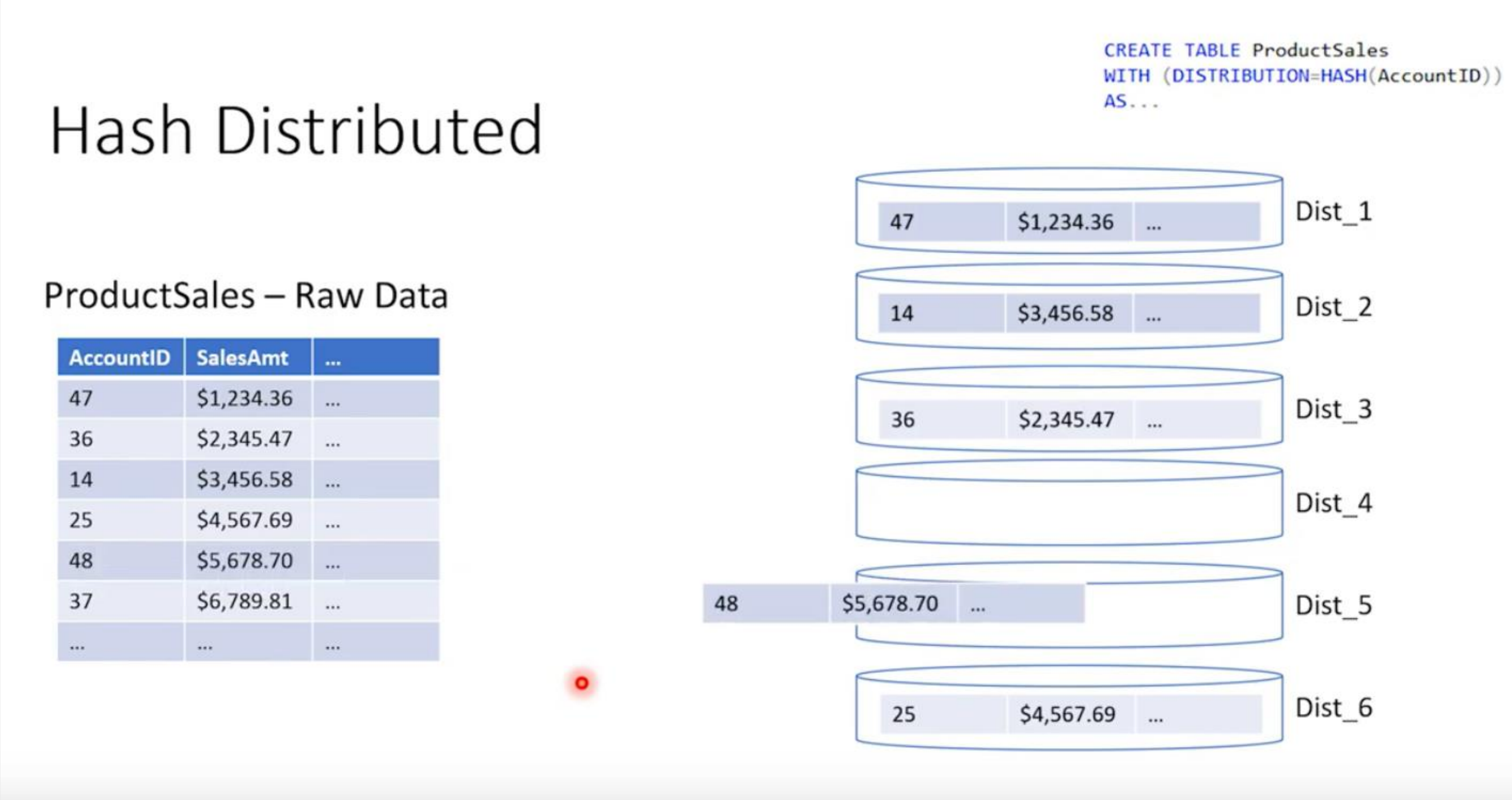
Hash Distributed

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

ProductSales – Raw Data

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

Hash Distributed (AccountID)



Hash Distributed

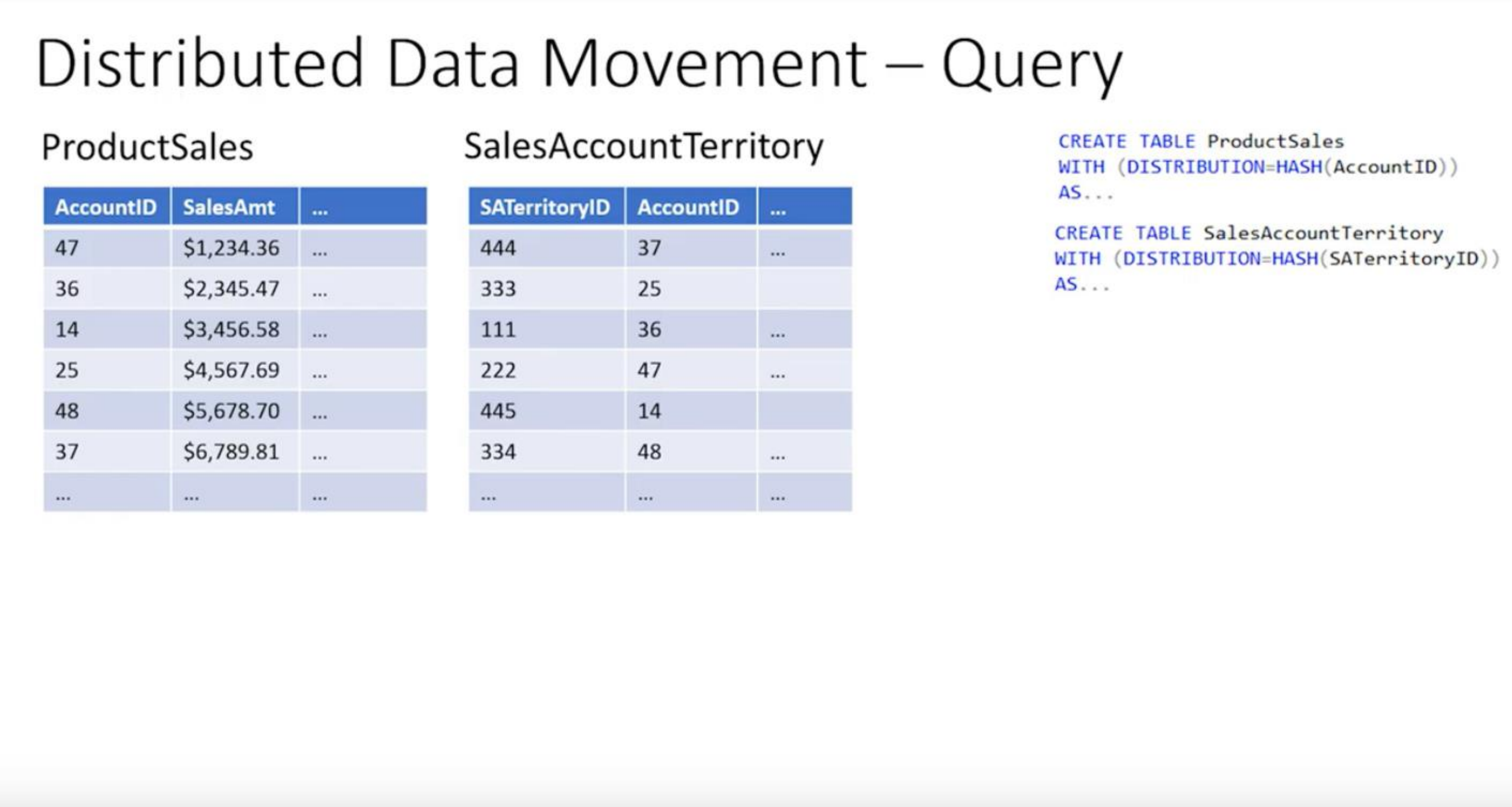
```
CREATE TABLE ProductSales
WITH (DISTRIBUTION=HASH(AccountID))
AS...
```

ProductSales – Raw Data

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

47	$1,234.36	...	Dist_1
14	$3,456.58	...	Dist_2
36	$2,345.47	...	Dist_3
			Dist_4
48	$5,678.70	...	Dist_5
25	$4,567.69	...	Dist_6

Distributed Data Movement



Distributed Data Movement – Query

ProductSales

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

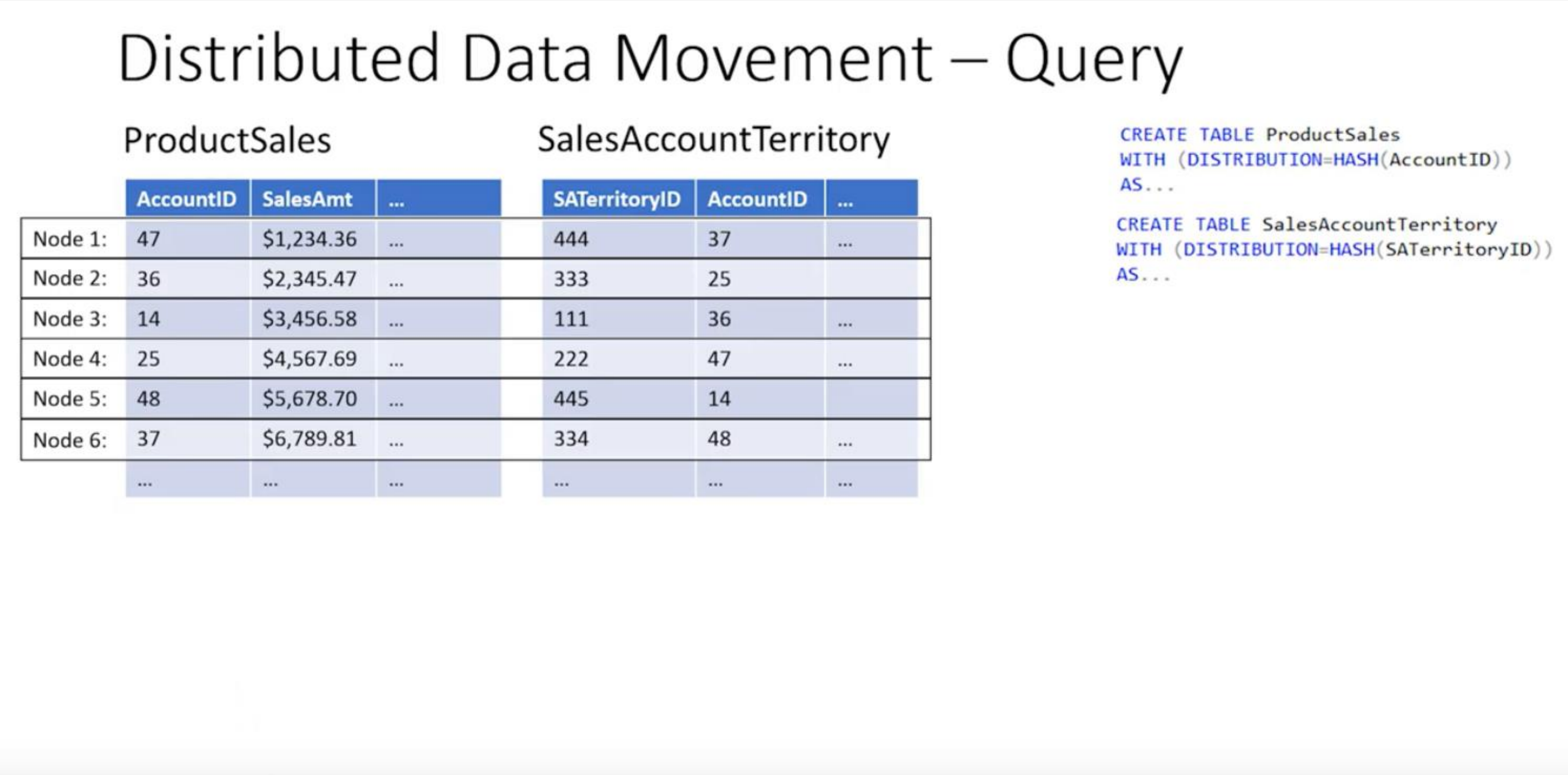
SalesAccountTerritory

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

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

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

Distributed Data Movement



Distributed Data Movement – Query

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	...
	...	...	...

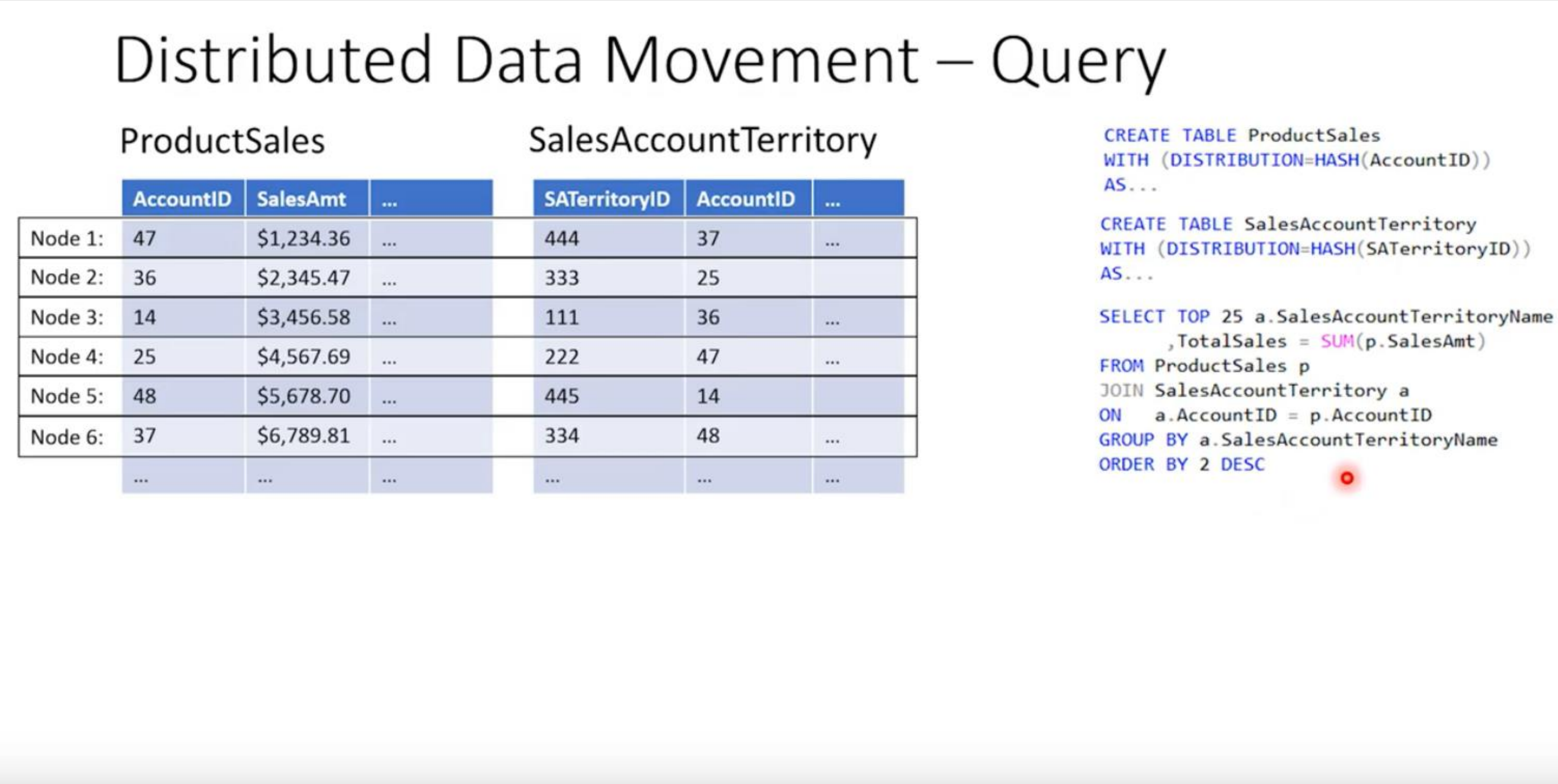
SalesAccountTerritory

	SATerritoryID	AccountID	...
Node 1:	444	37	...
Node 2:	333	25	
Node 3:	111	36	...
Node 4:	222	47	...
Node 5:	445	14	
Node 6:	334	48	...
	...	...	...

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

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

Distributed Data Movement



Distributed Data Movement – Query

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

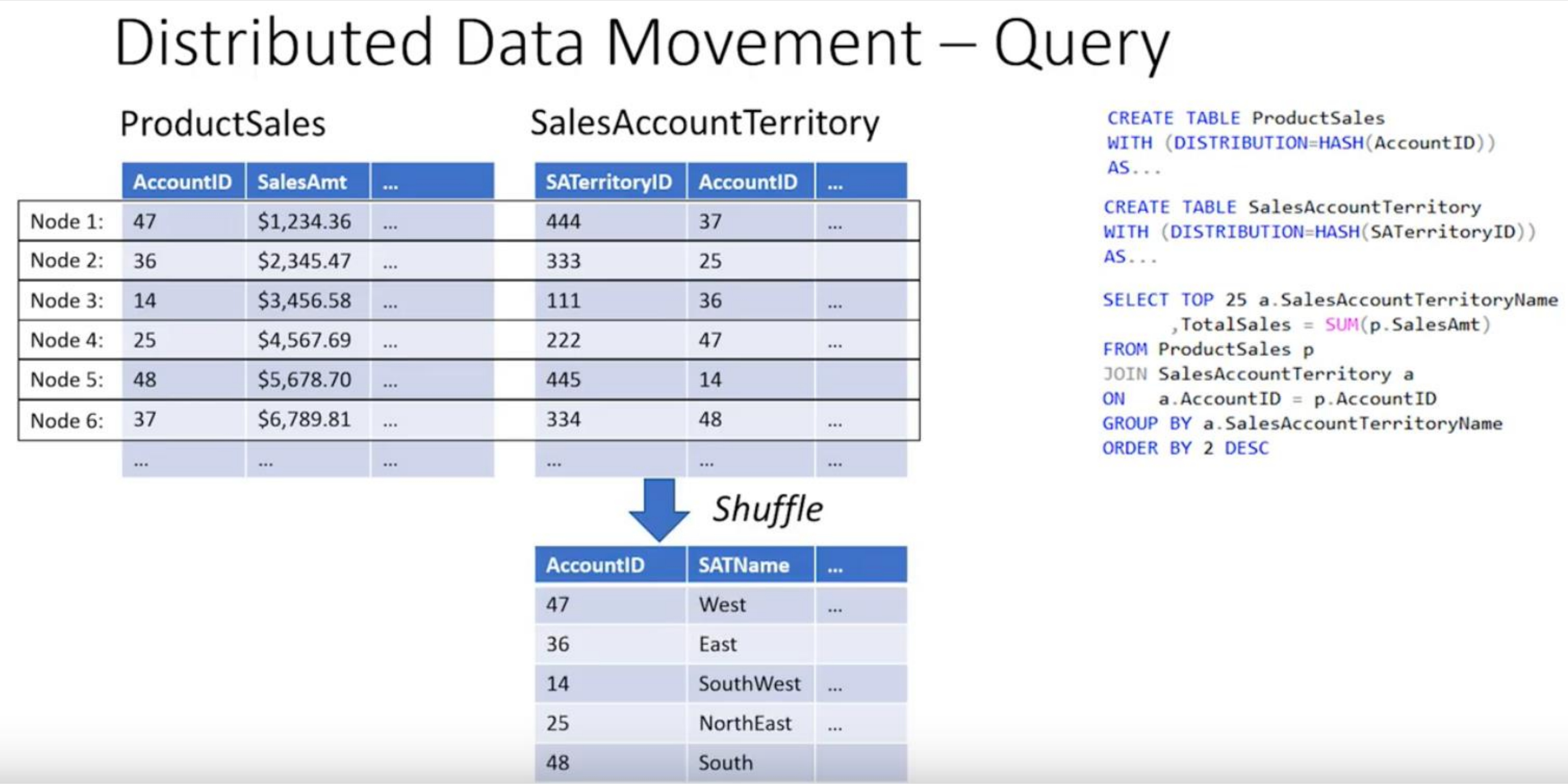
	SATerritoryID	AccountID	...
Node 1:	444	37	...
Node 2:	333	25	
Node 3:	111	36	...
Node 4:	222	47	...
Node 5:	445	14	
Node 6:	334	48	...
	...	...	...

```sql
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
```

Distributed Data Movement



Distributed Data Movement – Query

	ProductSales			SalesAccountTerritory		
	AccountID	SalesAmt	...	SATerritoryID	AccountID	...
Node 1:	47	$1,234.36	...	444	37	...
Node 2:	36	$2,345.47	...	333	25	
Node 3:	14	$3,456.58	...	111	36	...
Node 4:	25	$4,567.69	...	222	47	...
Node 5:	48	$5,678.70	...	445	14	
Node 6:	37	$6,789.81	...	334	48	...
	...	...	...	...	...	...

Shuffle

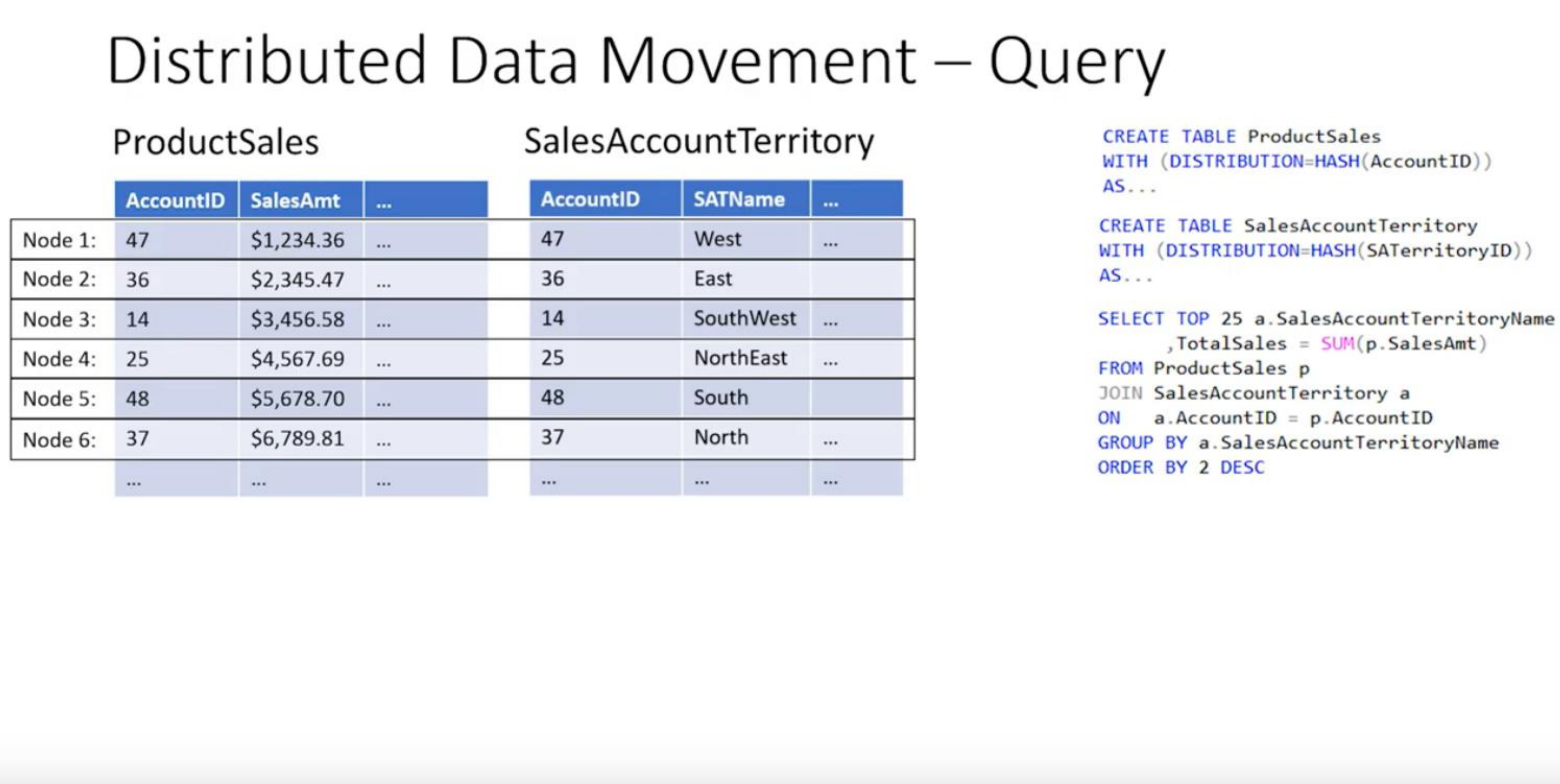
AccountID	SATName	...
47	West	...
36	East	
14	SouthWest	...
25	NorthEast	...
48	South	

```
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
```

Distributed Data Movement



Distributed Data Movement – Query

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

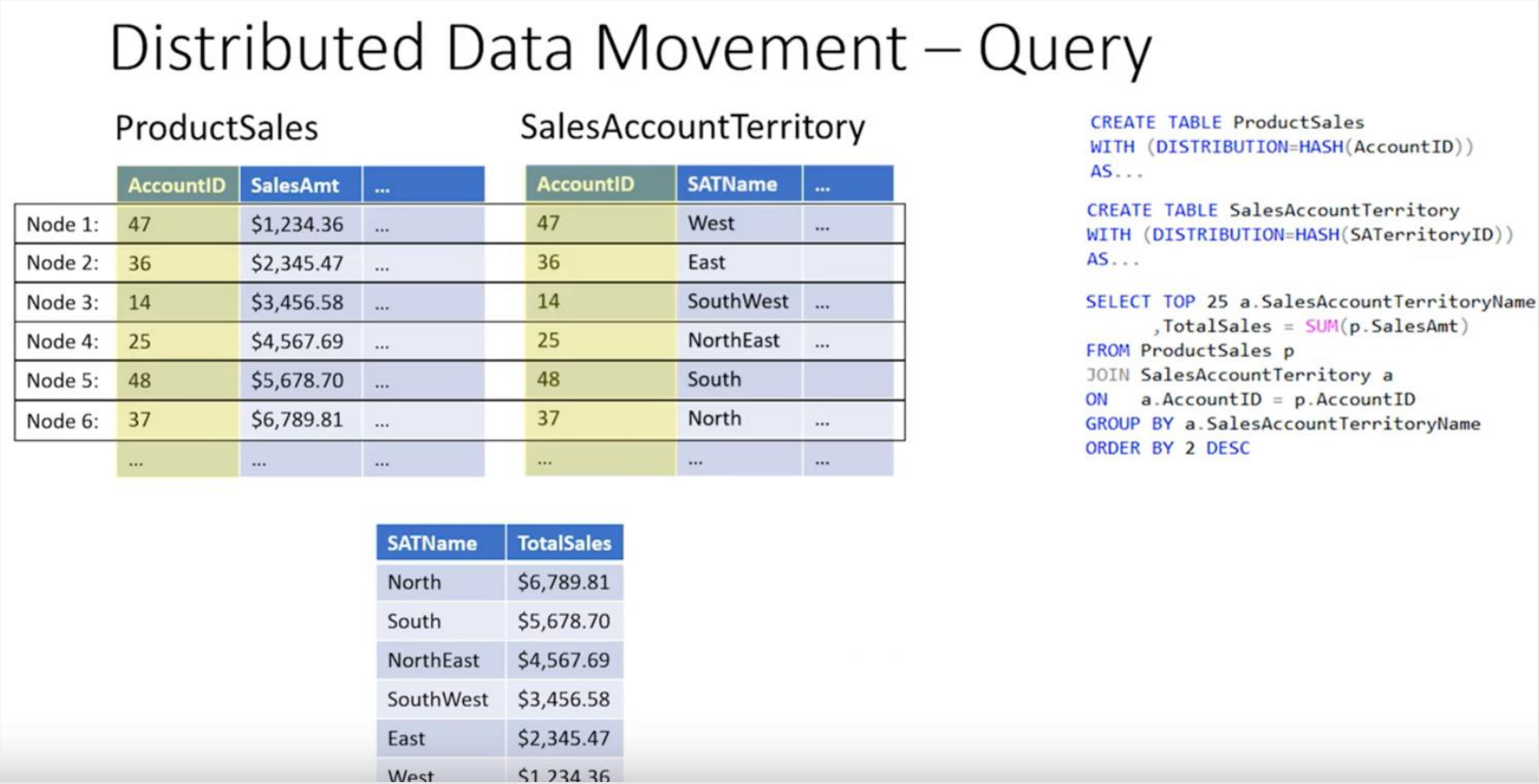
	AccountID	SATName	...
Node 1:	47	West	...
Node 2:	36	East	
Node 3:	14	SouthWest	...
Node 4:	25	NorthEast	...
Node 5:	48	South	
Node 6:	37	North	...
	...	...	...

```
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
```

Distributed Data Movement



Distributed Data Movement – Query

ProductSales

SalesAccountTerritory

	AccountID	SalesAmt	...	AccountID	SATName	...
Node 1:	47	$1,234.36	...	47	West	...
Node 2:	36	$2,345.47	...	36	East	
Node 3:	14	$3,456.58	...	14	SouthWest	...
Node 4:	25	$4,567.69	...	25	NorthEast	...
Node 5:	48	$5,678.70	...	48	South	
Node 6:	37	$6,789.81	...	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
```

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

Distribution Guidance

For large fact tables, best option is to Hash Distribute

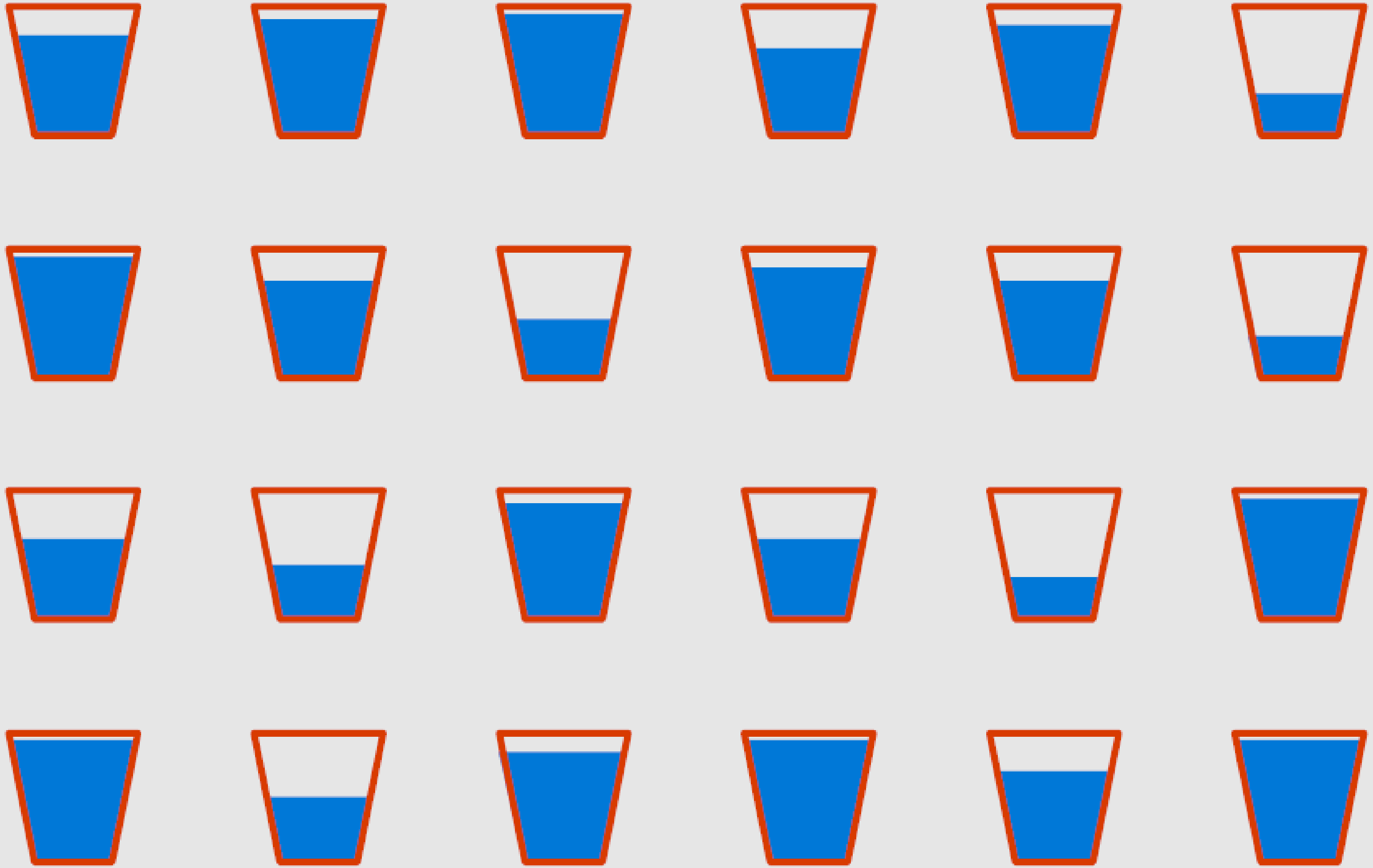
- Clustered Columnstore
- Distribute on column that is joined to other fact tables or large dimensions
- Primary or surrogate key maybe a good choice for distribution

However, be mindful of …

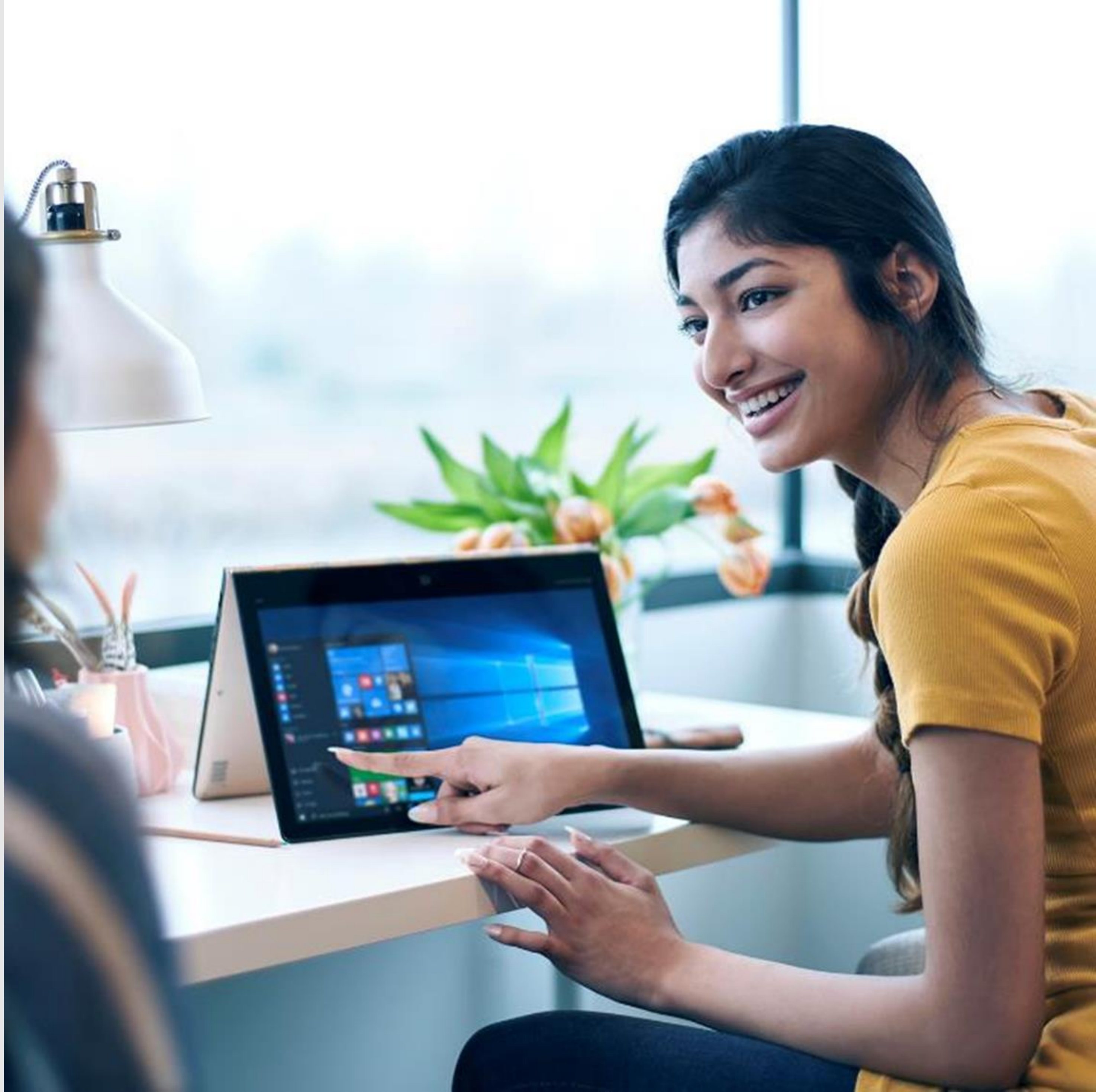
- Hash column should have highly distinct values (Minimum 600 distinct values)
- Avoid distributing on a date column
- Avoid distributing on column with high frequency of NULLs and default values (e.g. -1)
- Distribution column is NOT updatable
- For compatible joins use the same data types for two distributed tables

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

Skewed Distribution



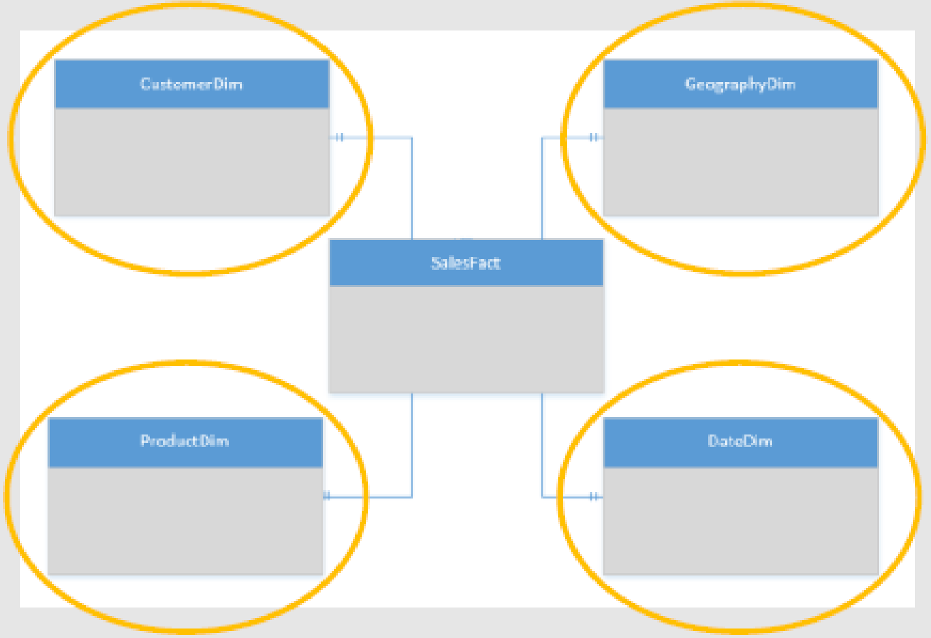
Replicated Tables



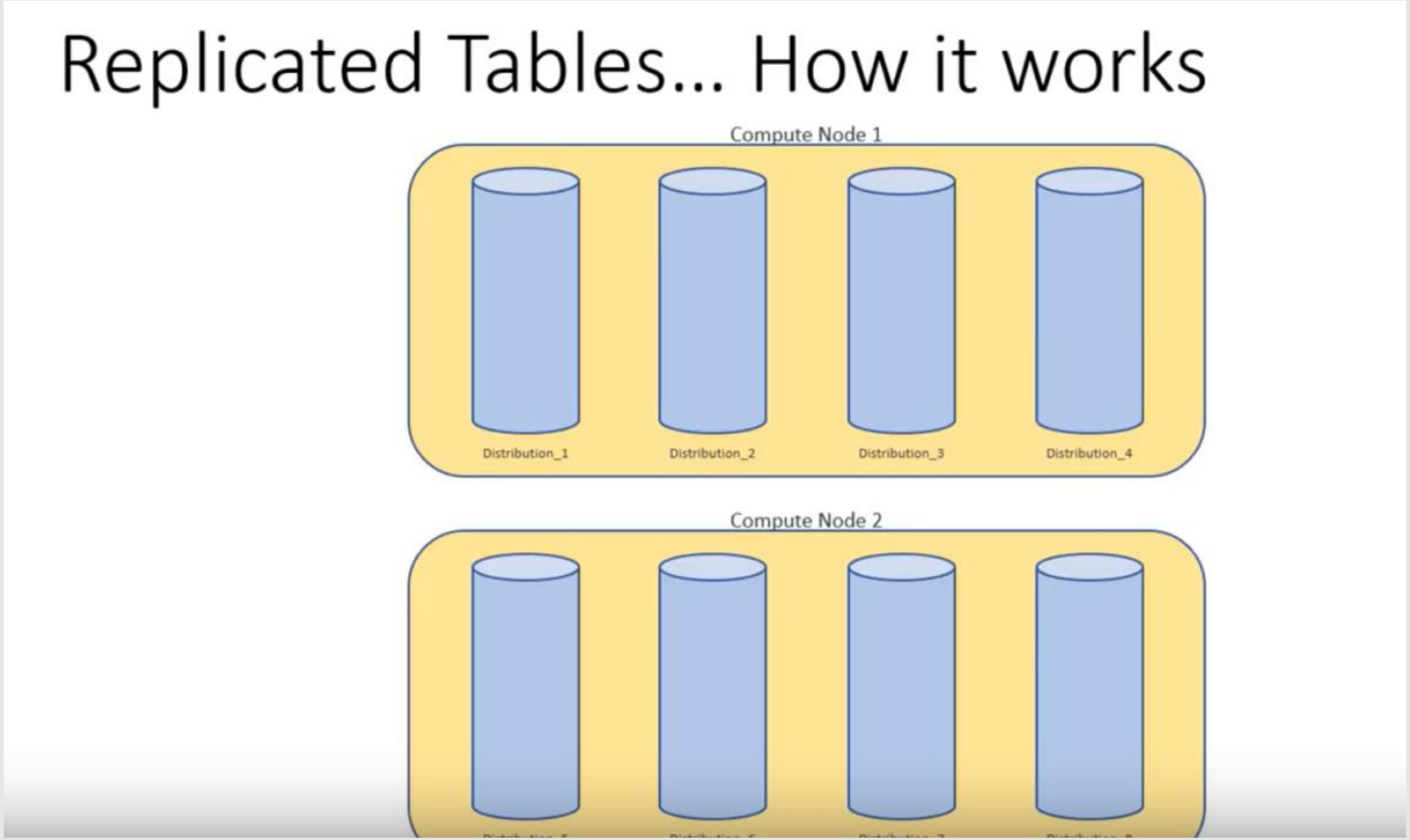
Replicated Tables Scenarios

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

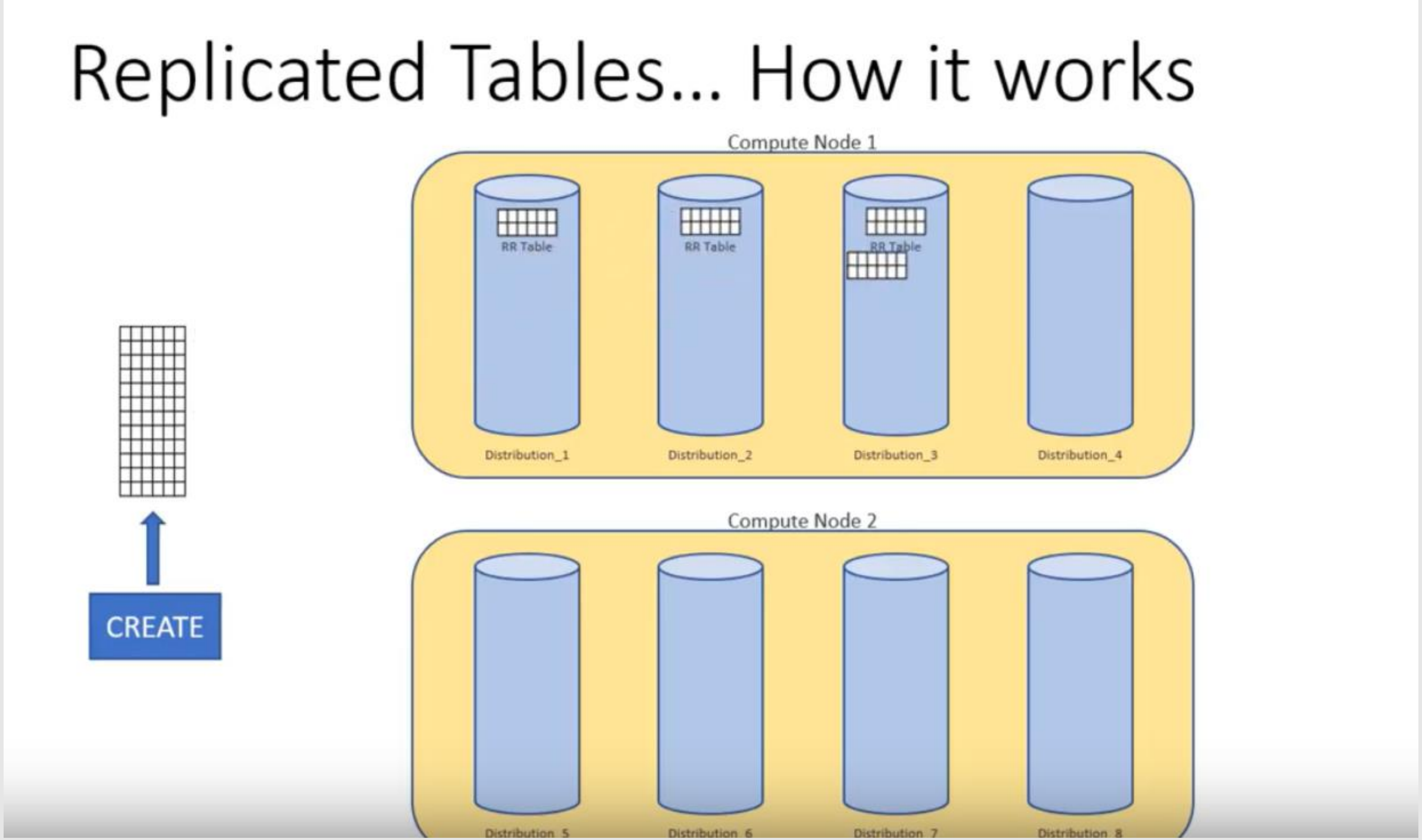


Replicated Tables .. How it works



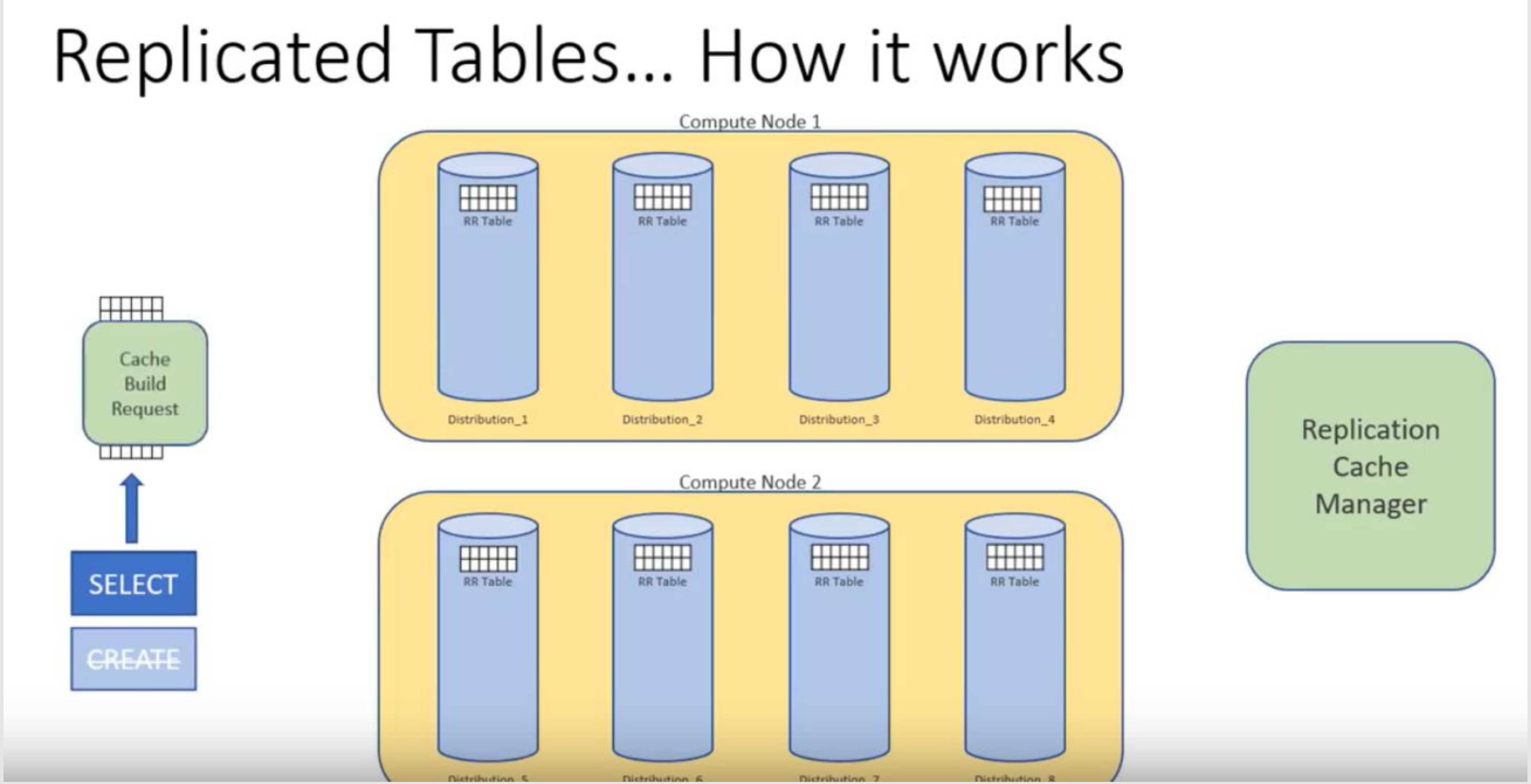
Replicated Tables... How it works

Replicated Tables .. How it works



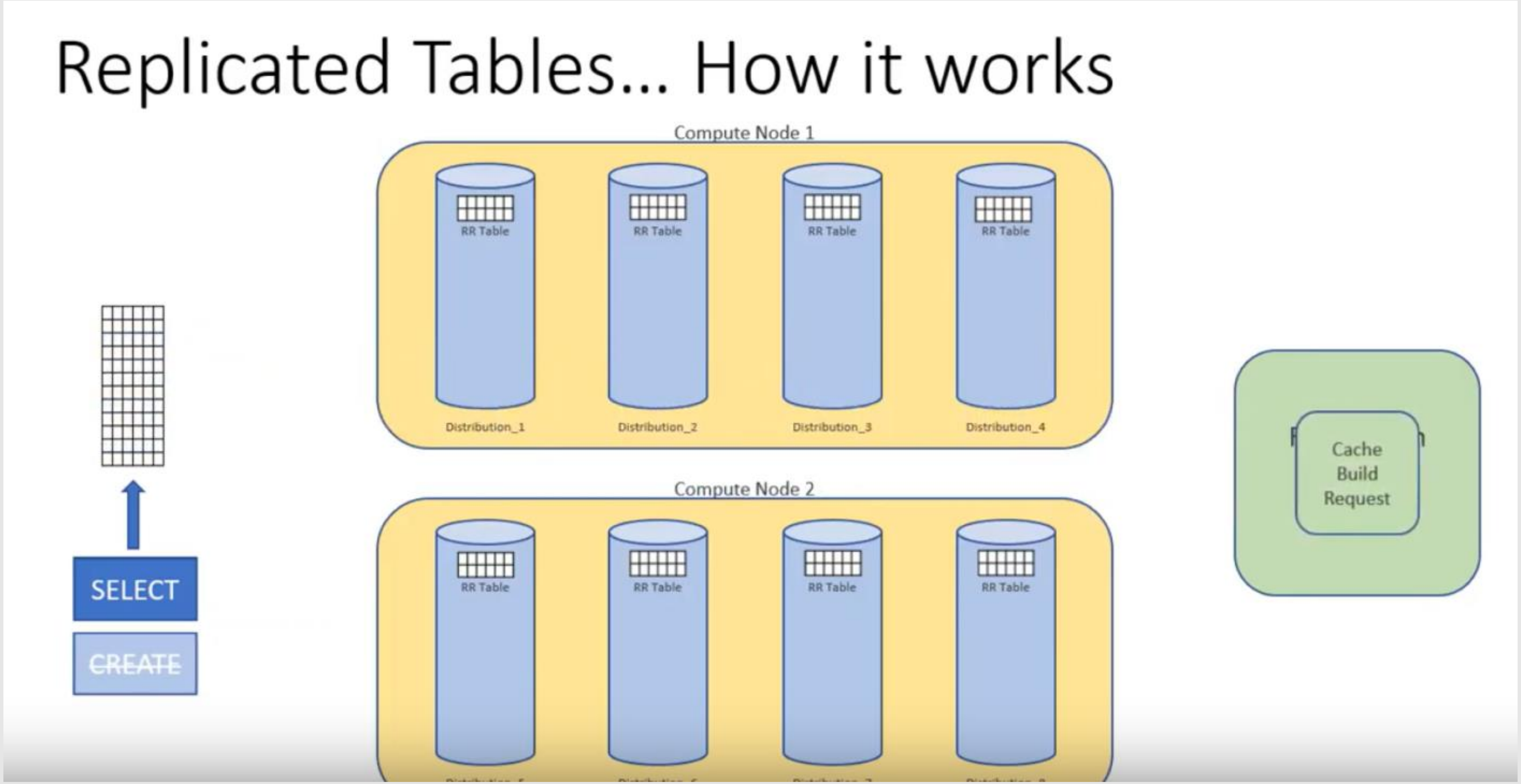
Replicated Tables... How it works

Replicated Tables .. How it works



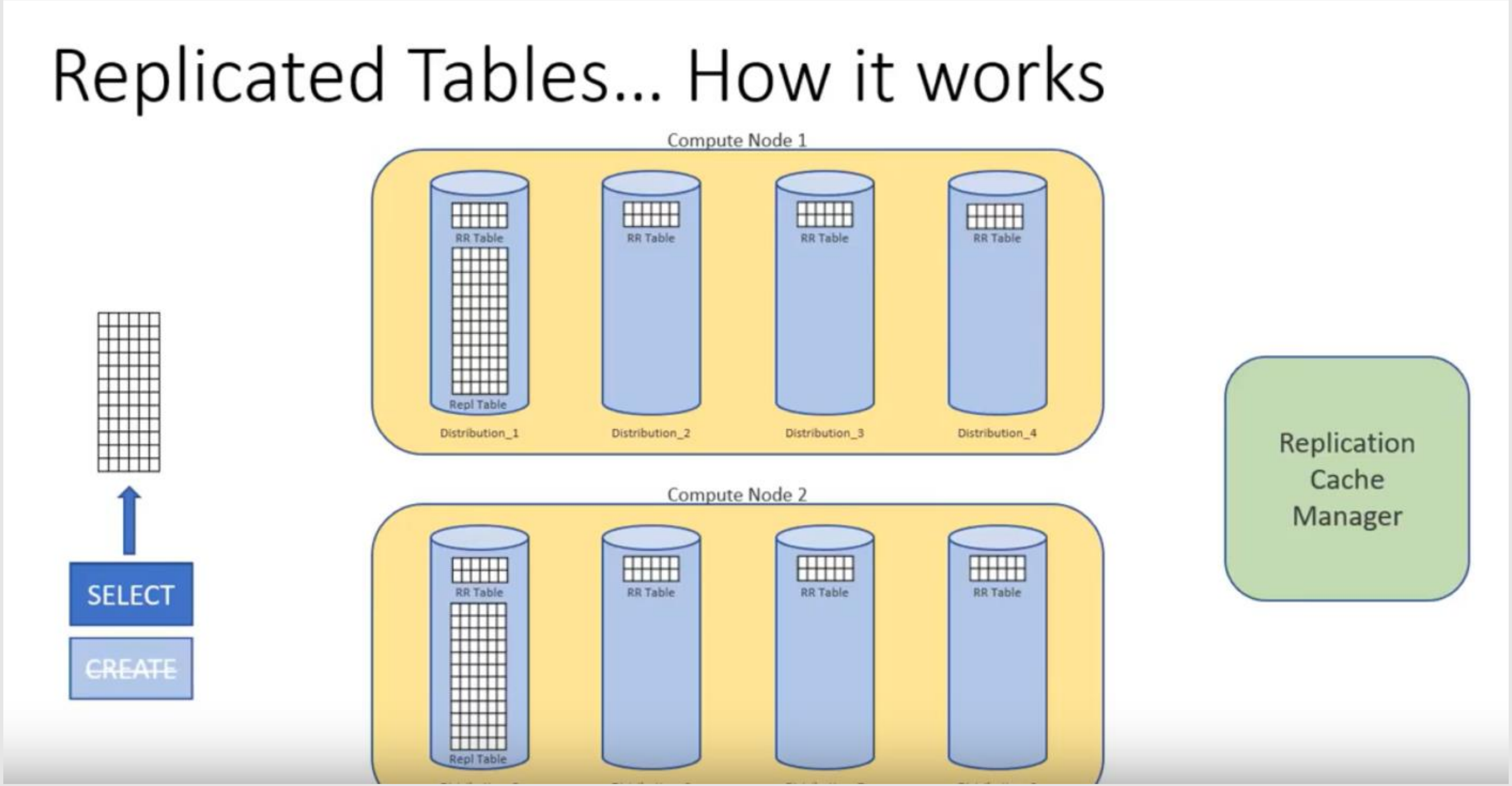
Replicated Tables... How it works

Replicated Tables .. How it works



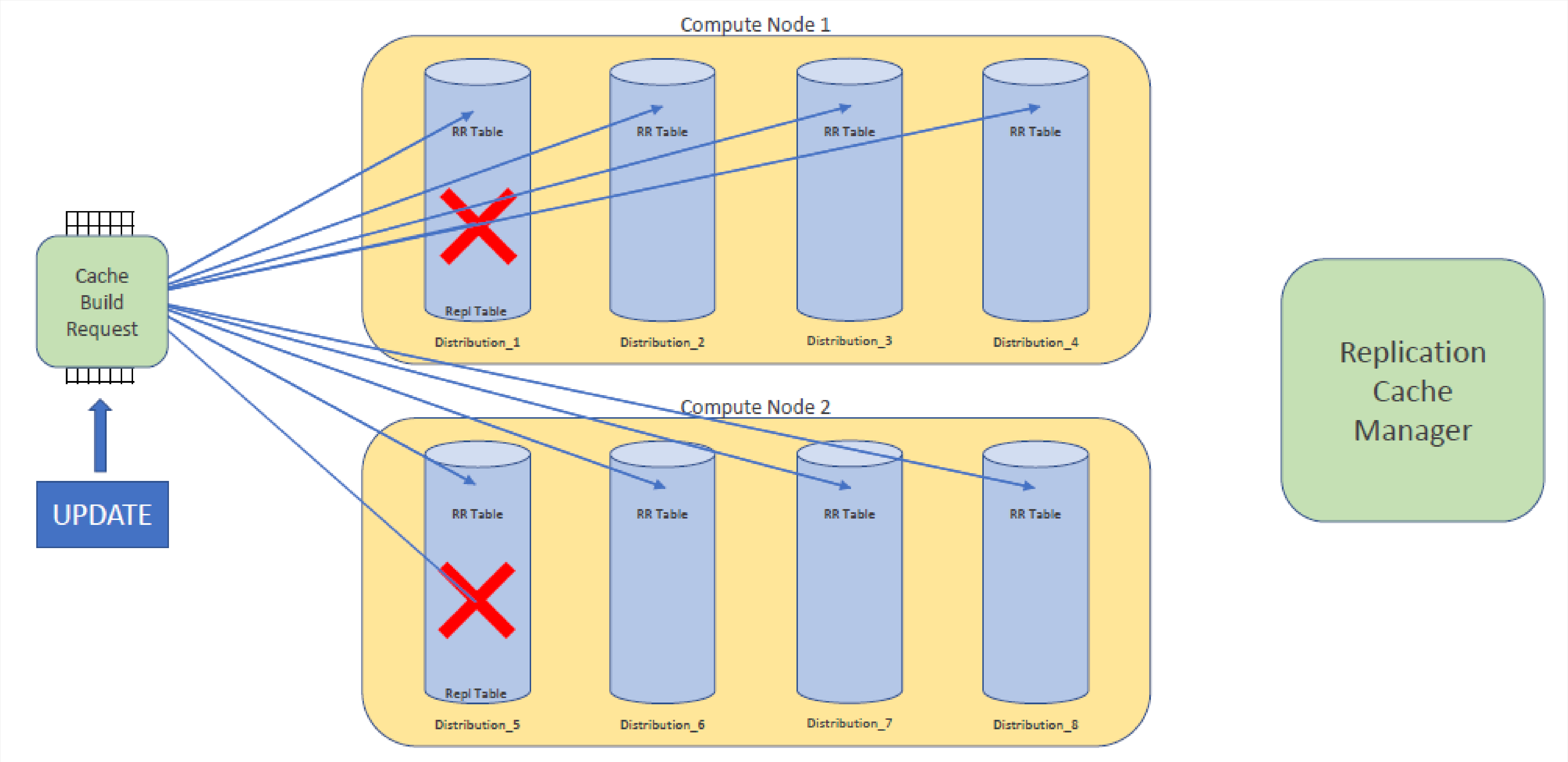
Replicated Tables... How it works

Replicated Tables .. How it works



Replicated Tables... How it works

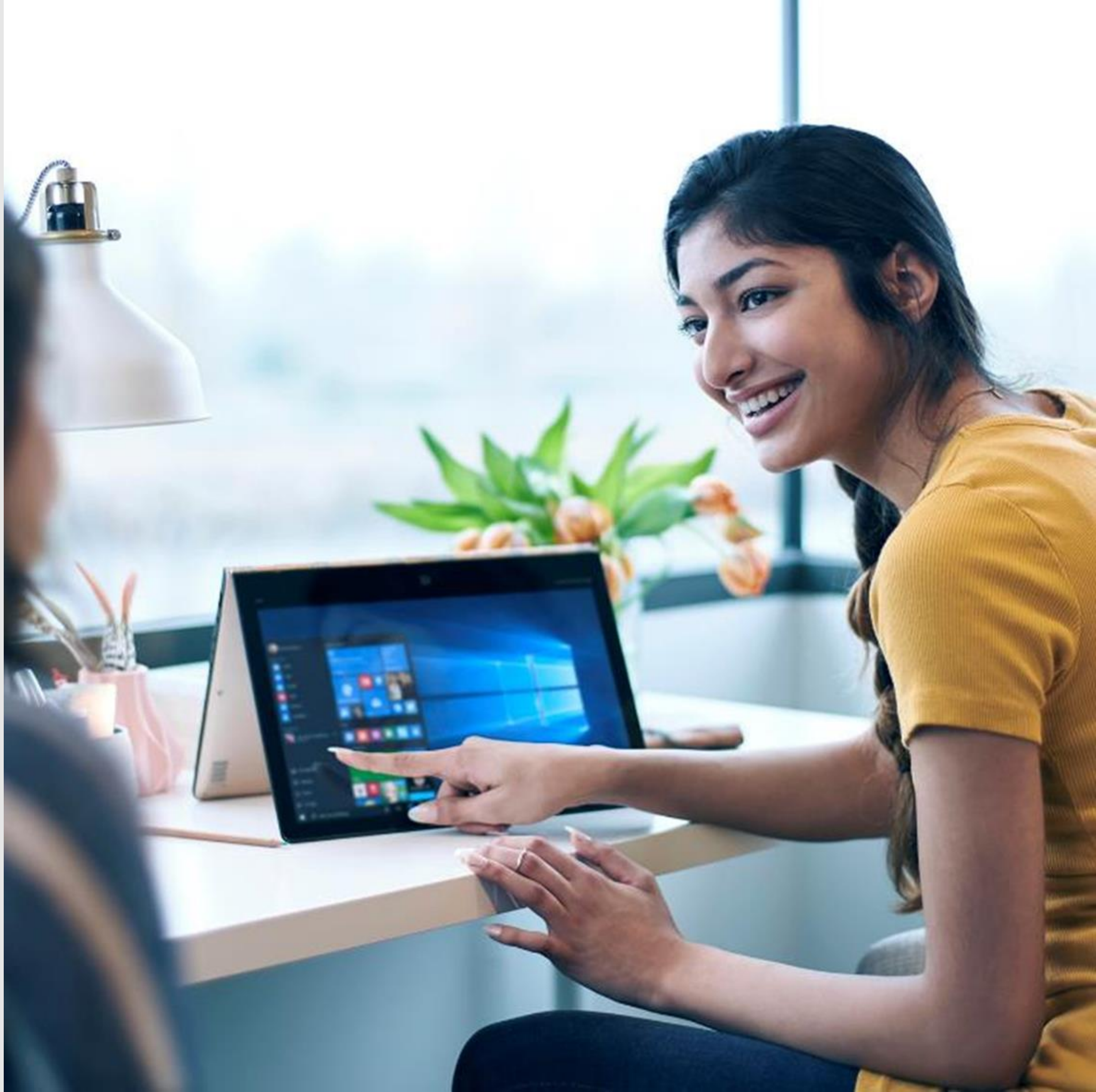
Replicated Tables .. How it works



Replicated Tables 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

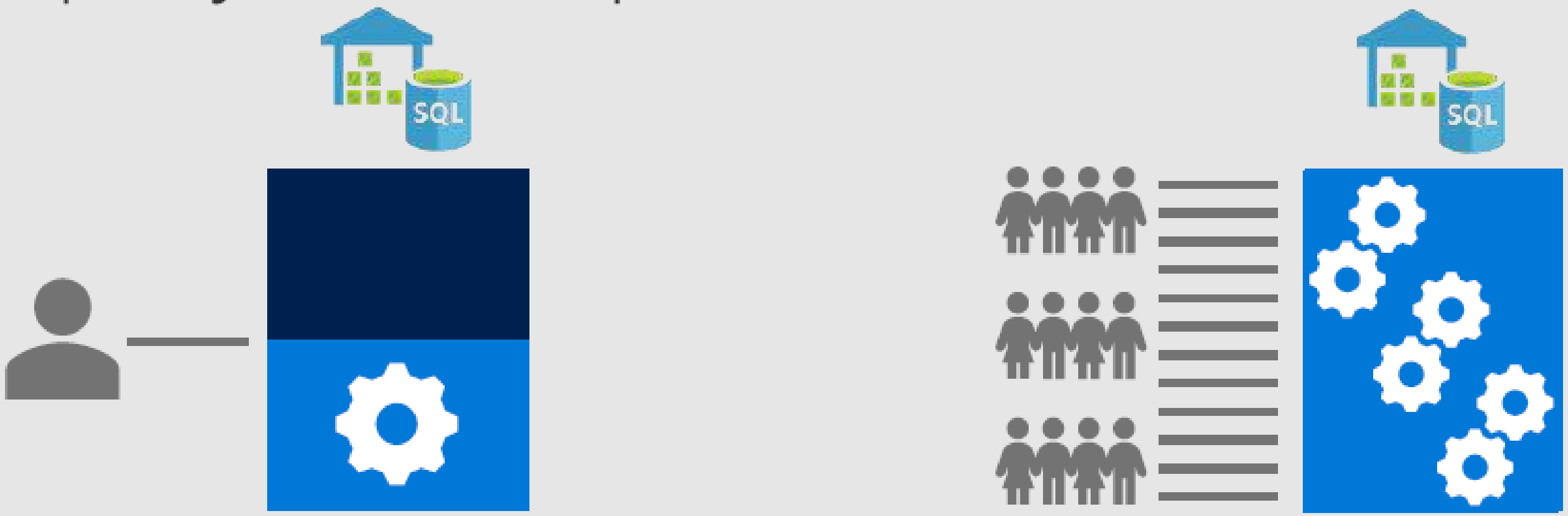


Importance of Concurrency

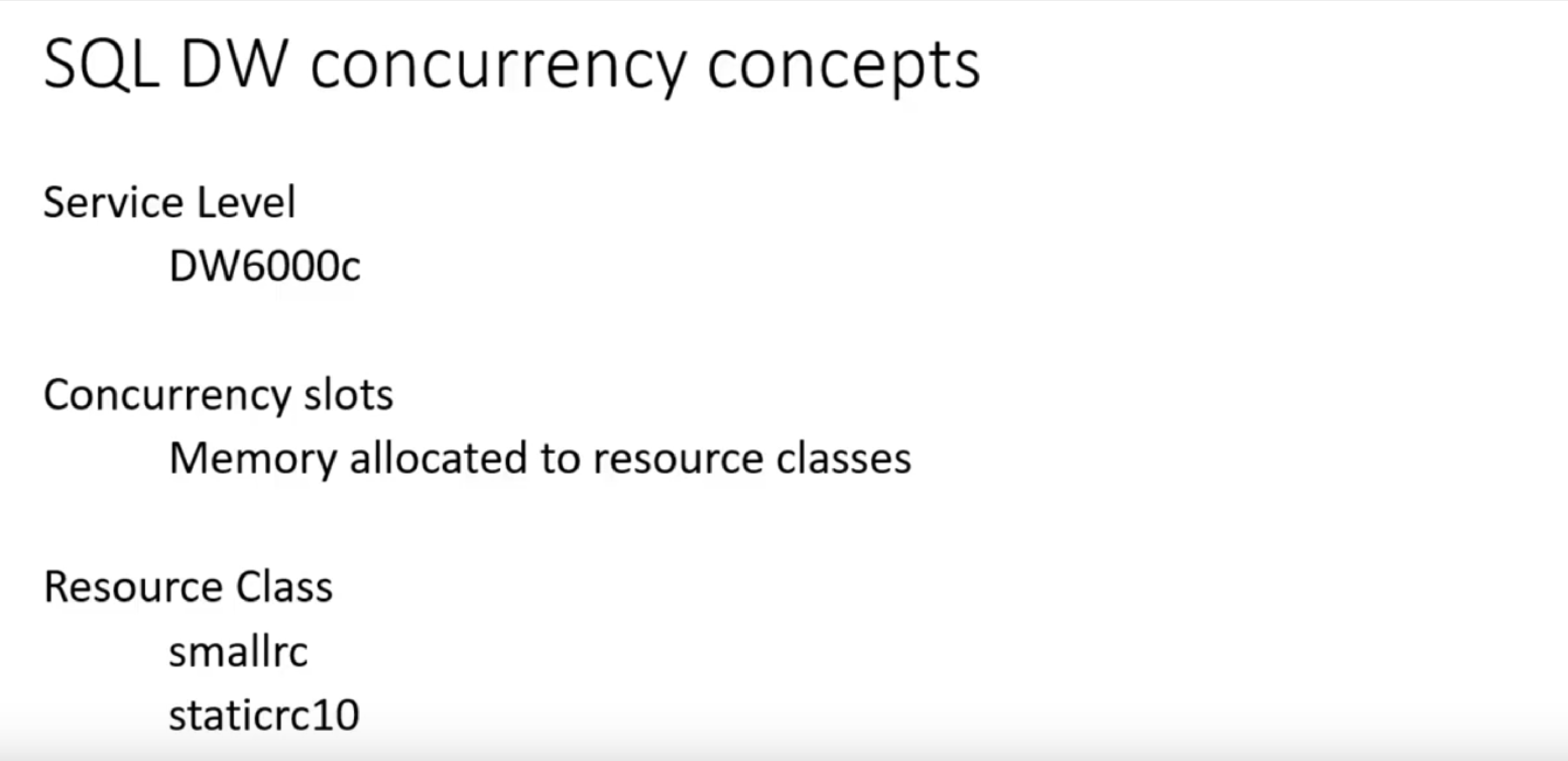
Parallel queries = more throughput (even on large, scale-out DWs)

Mixed workloads (transform, load, export, query)

A single query is not expected to consume all resources



Concurrency Slots



SQL DW concurrency concepts

Service Level

- DW6000c

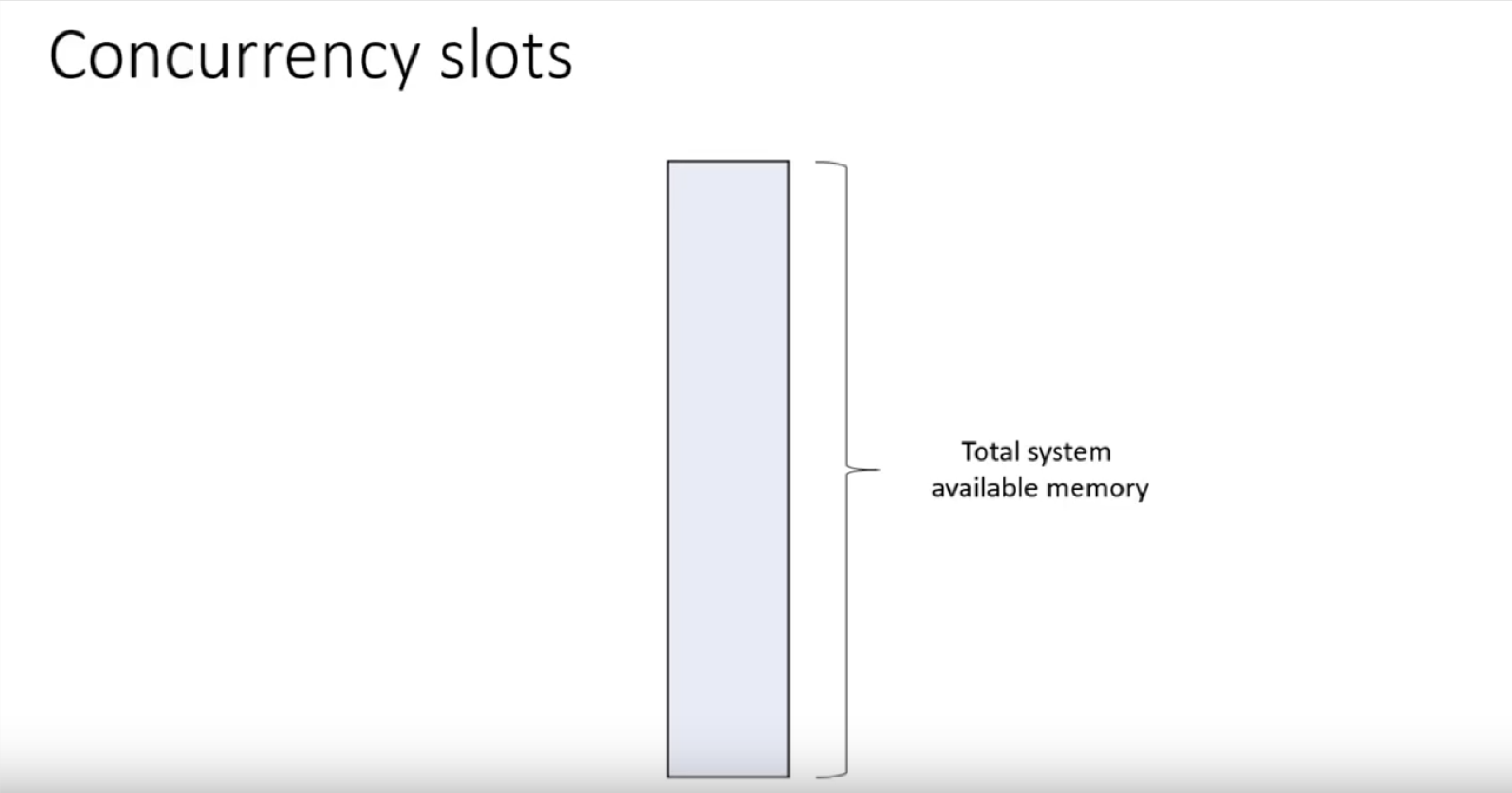
Concurrency slots

- Memory allocated to resource classes

Resource Class

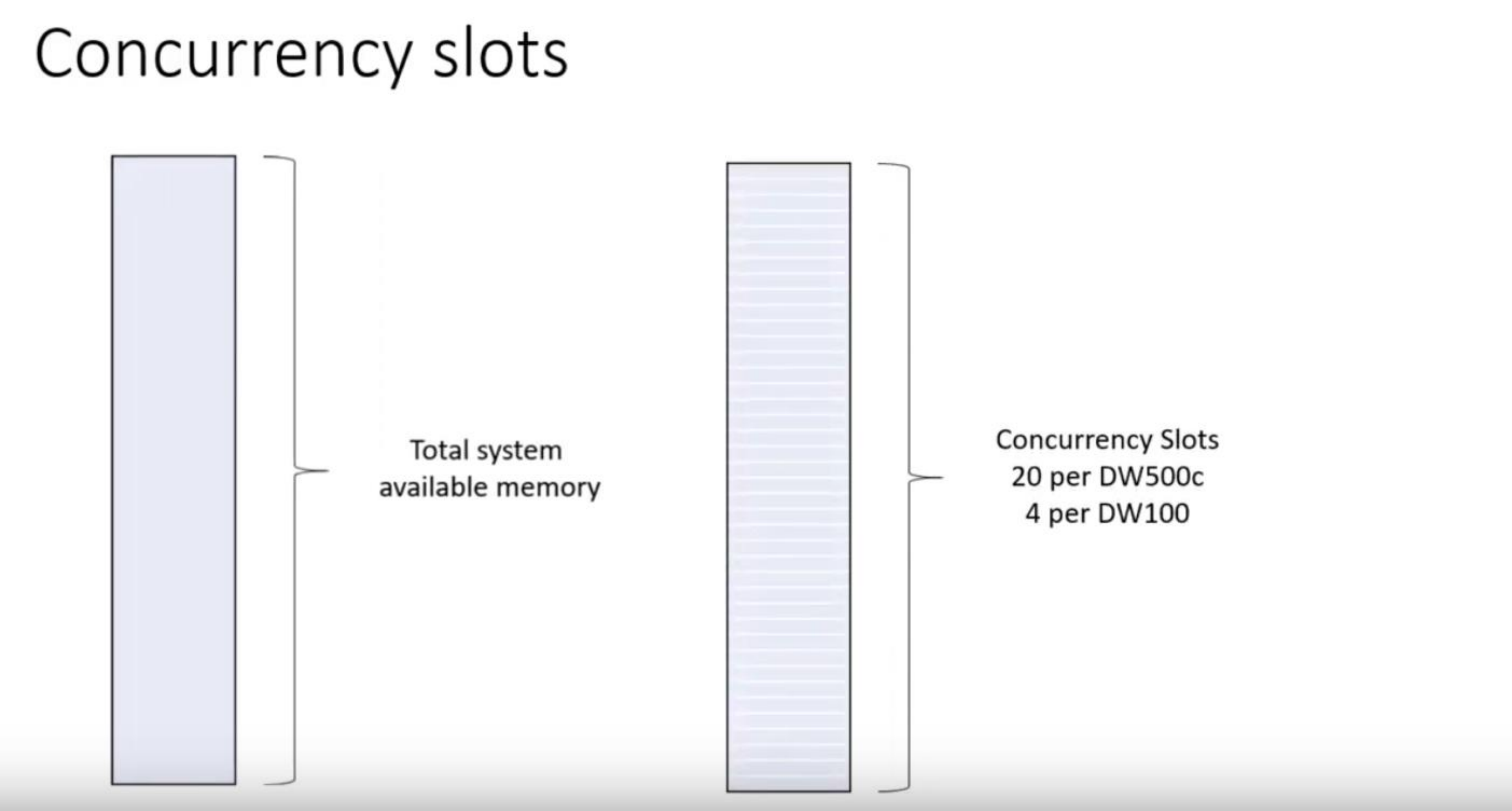
- smallrc
- staticrc10

Concurrency Slots



Concurrency slots

Concurrency Slots

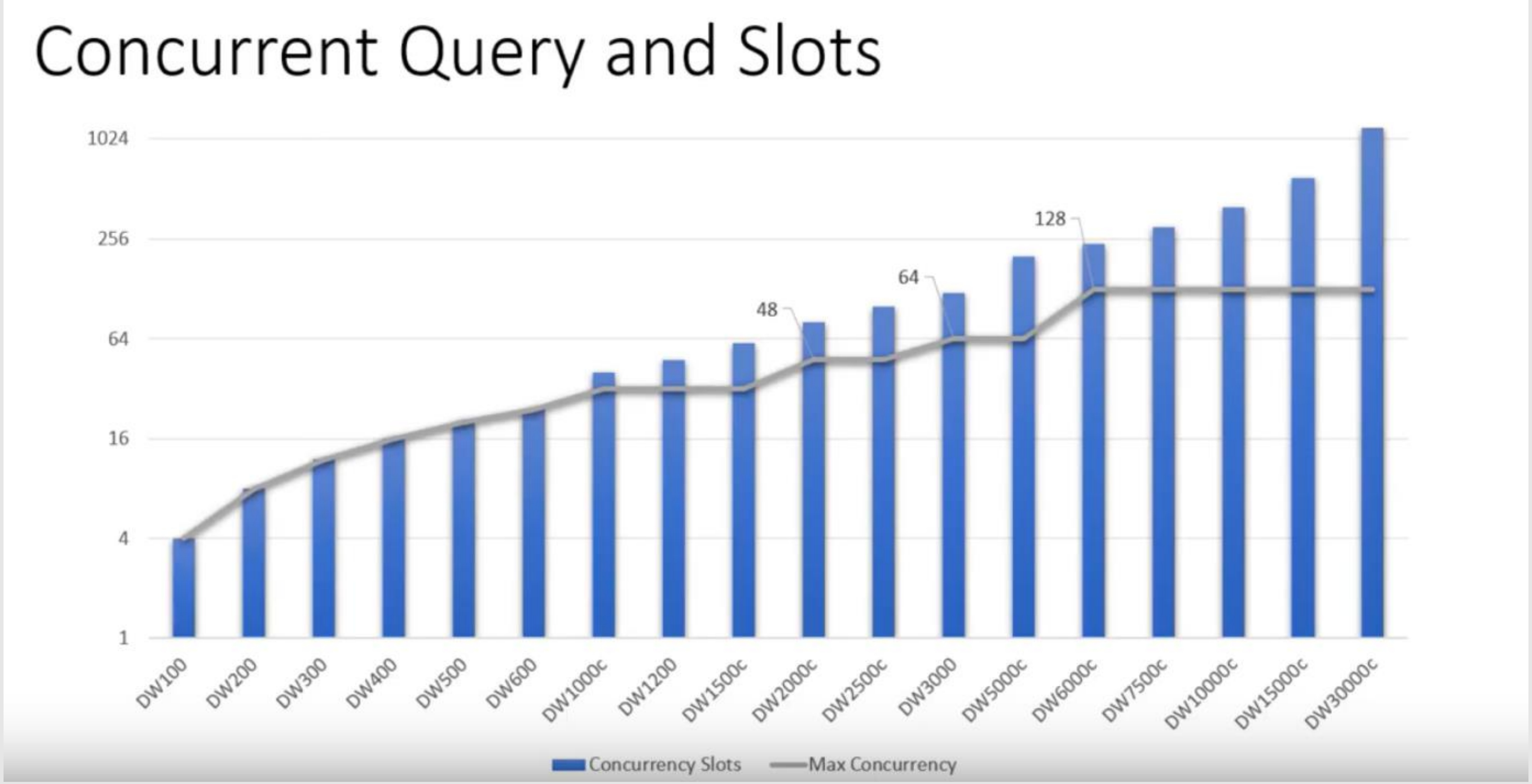


Concurrency slots

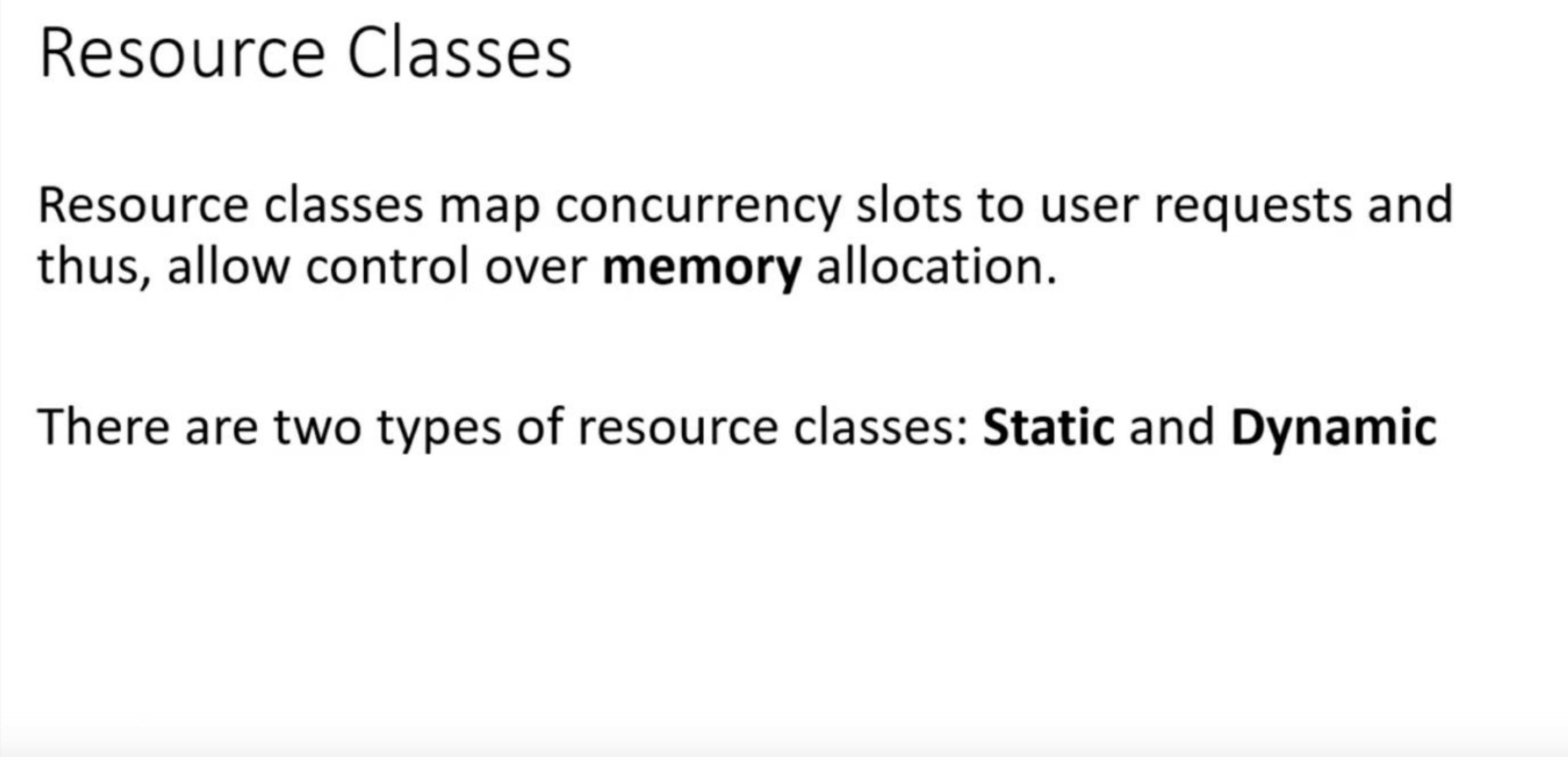
Total system
available memory

Concurrency Slots
20 per DW500c
4 per DW100

Concurrency Slots



Resource Classes

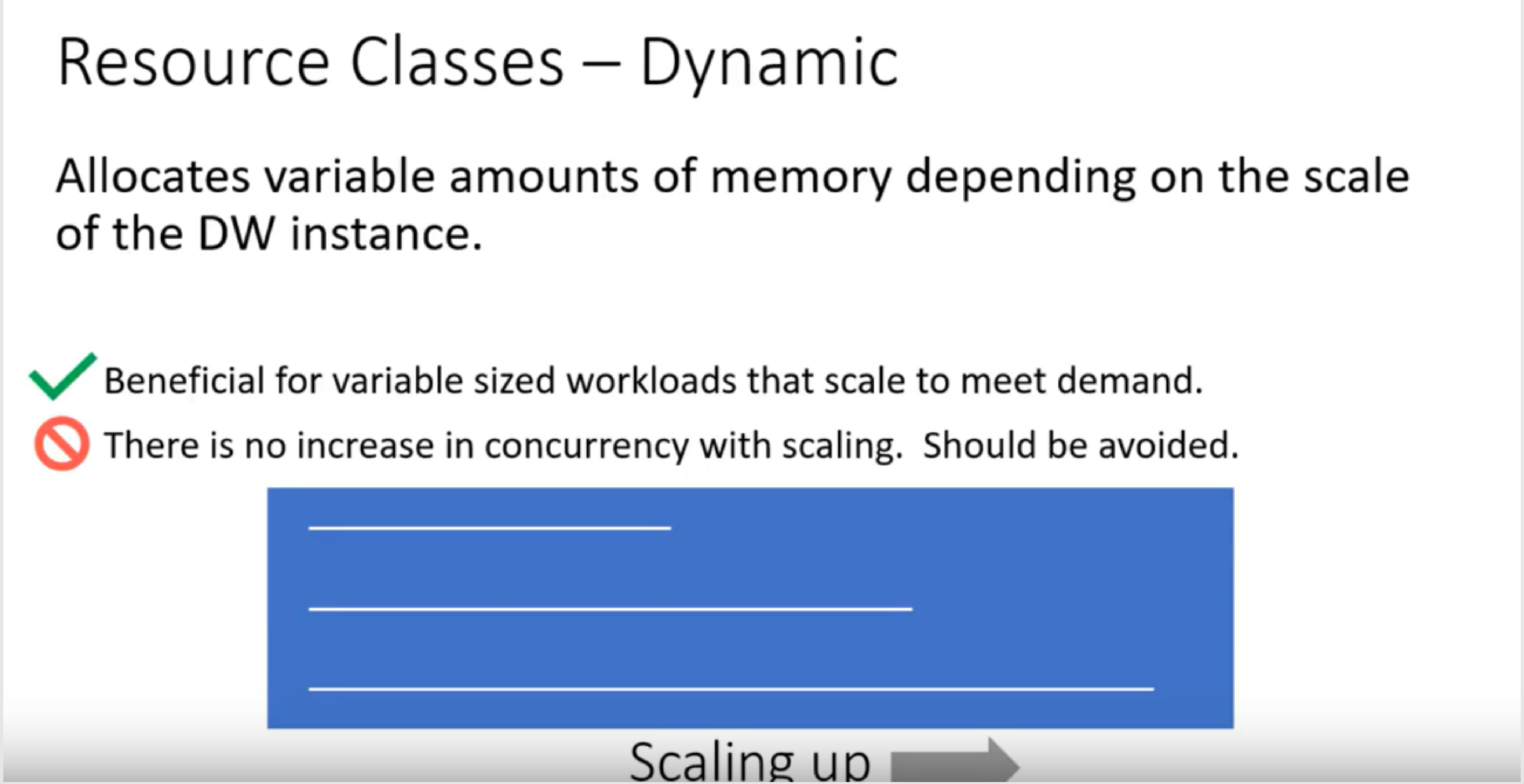


Resource Classes

Resource classes map concurrency slots to user requests and thus, allow control over **memory** allocation.

There are two types of resource classes: **Static** and **Dynamic**

Resource Classes



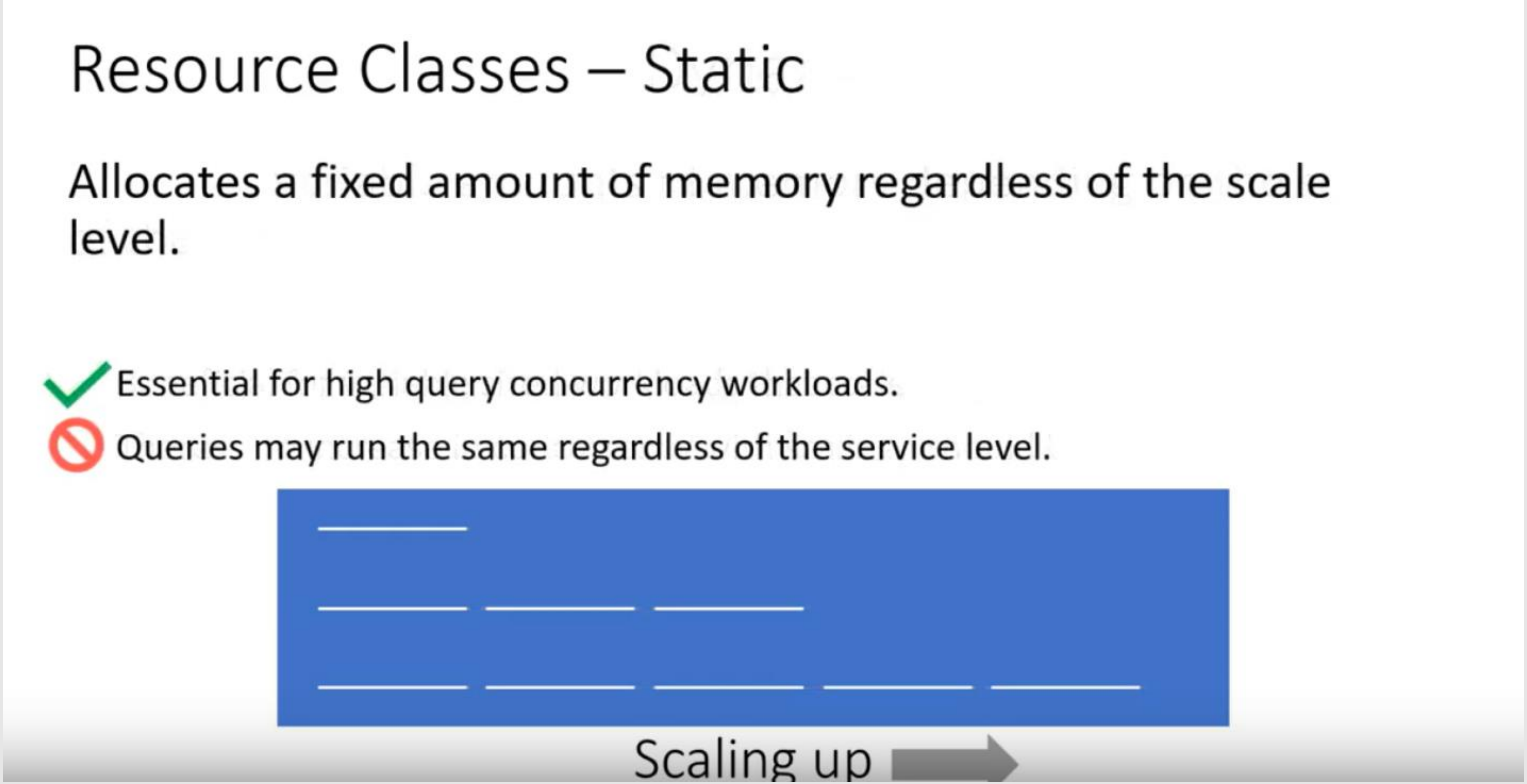
Resource Classes – Dynamic

Allocates variable amounts of memory depending on the scale of the DW instance.

✓ Beneficial for variable sized workloads that scale to meet demand.

🚫 There is no increase in concurrency with scaling. Should be avoided.

Resource Classes



Resource Classes – Static

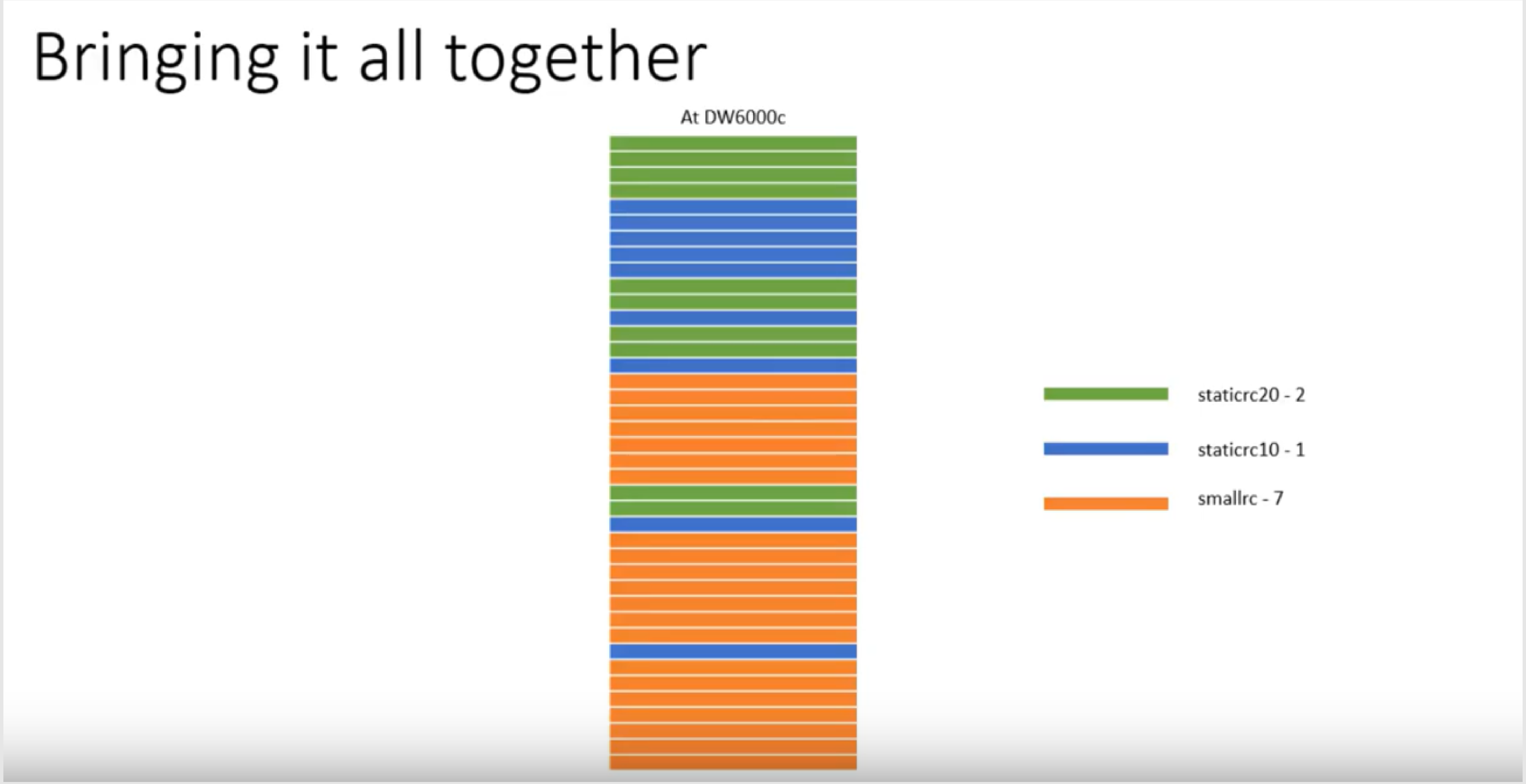
Allocates a fixed amount of memory regardless of the scale level.

✓ Essential for high query concurrency workloads.

⊘ Queries may run the same regardless of the service level.

Scaling up

Resource Classes

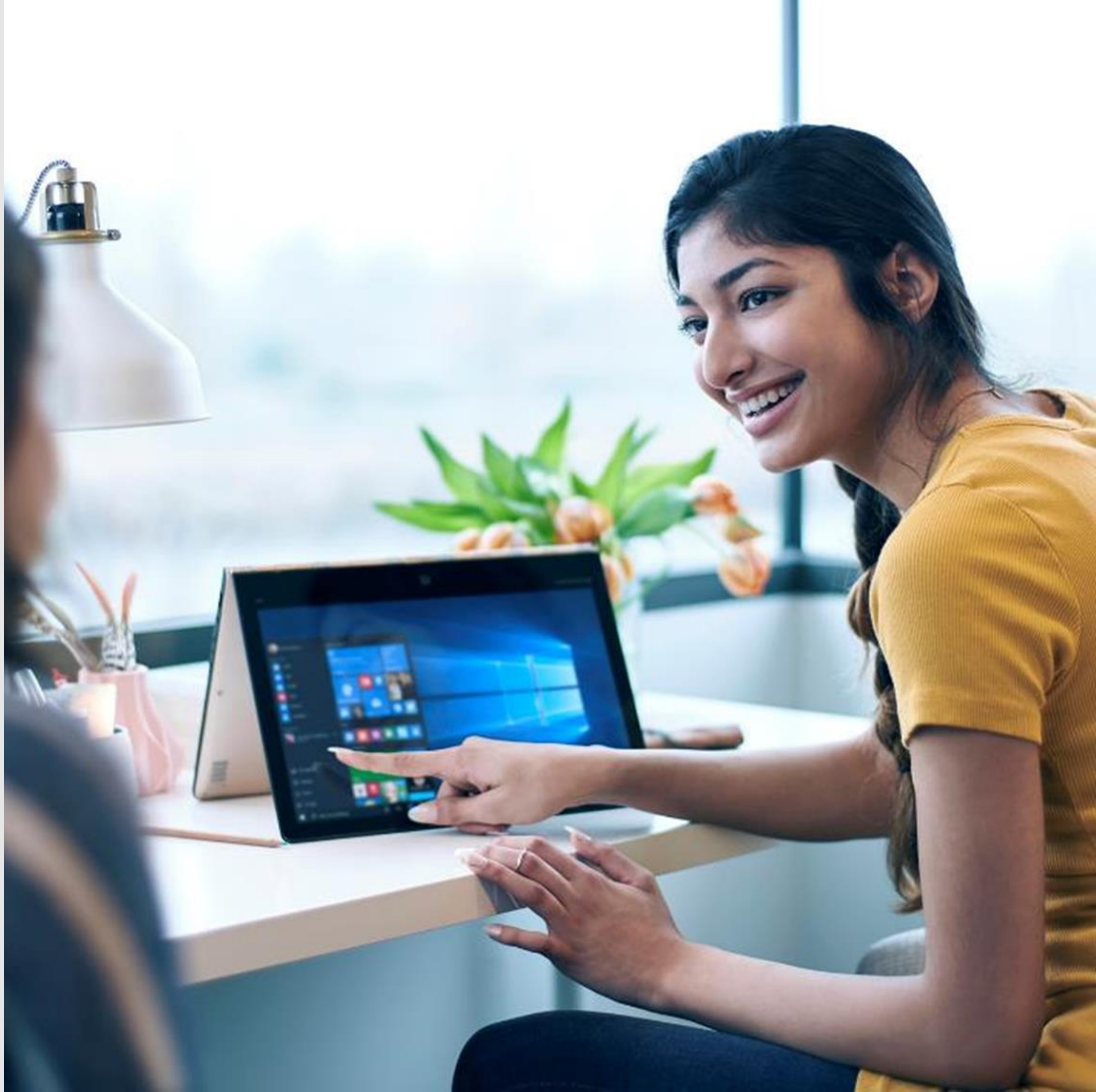


Bringing it all together

Resource Class Best Practices

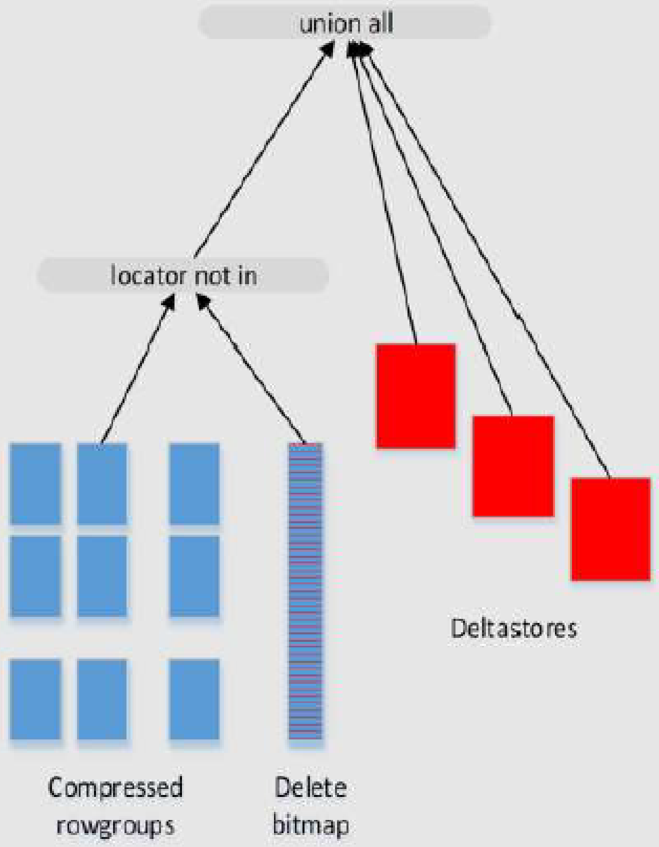
- Be aware of the user binding
- Use right resource class for right workload
- Prefer Static for higher concurrency
- Prefer Dynamic for increased perf with scale
- Be aware of the RC precedence
  - Dynamic over Static
  - Higher RC over lower
- Explicitly remove user from unnecessary RCs
- Use custom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

Index/Stats



Clustered Columnstore Indexes(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)

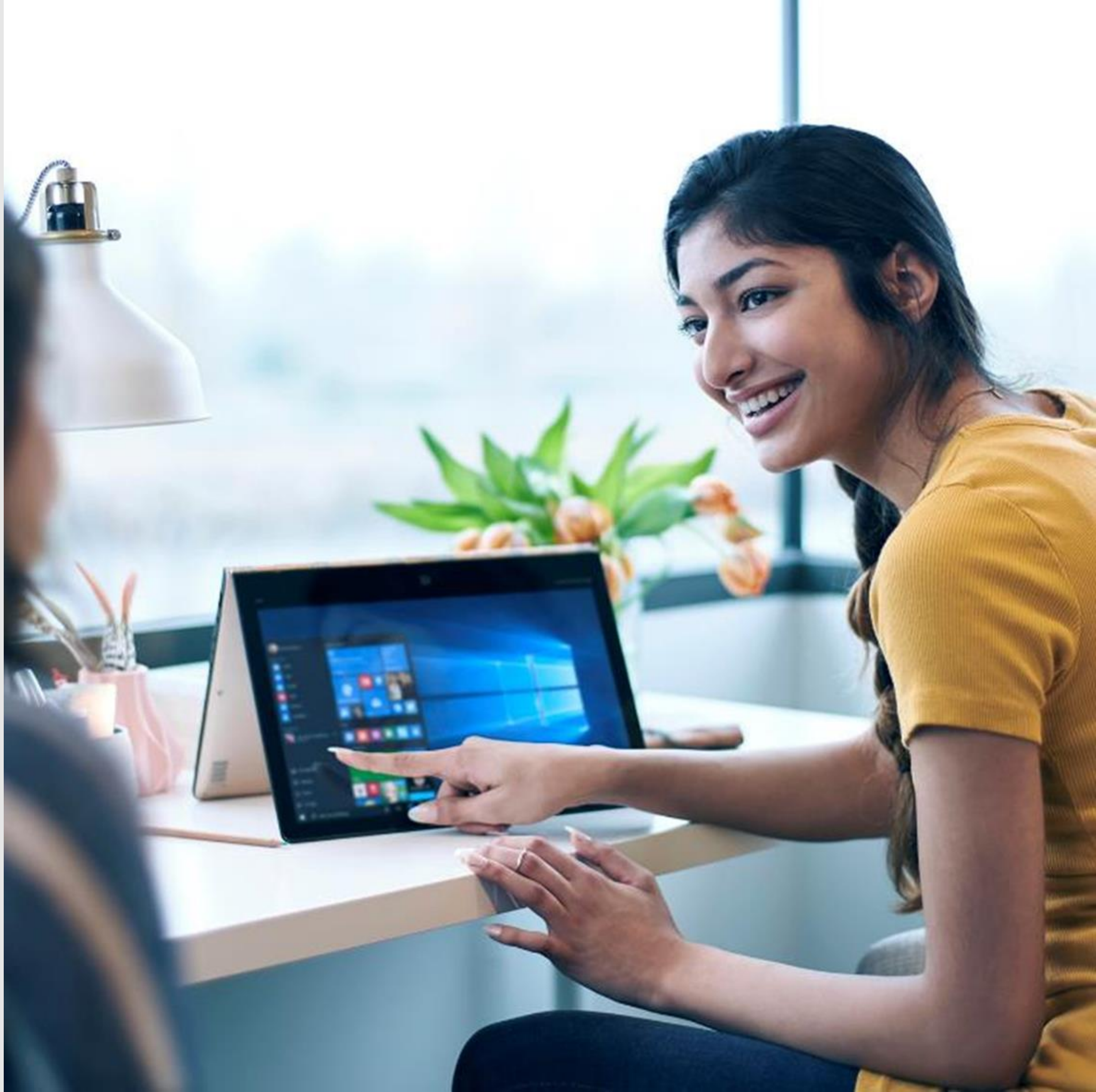


https://docs.microsoft.com/en-us/azure/sql-data-warehouse/sql-data-warehouse-tables-index

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

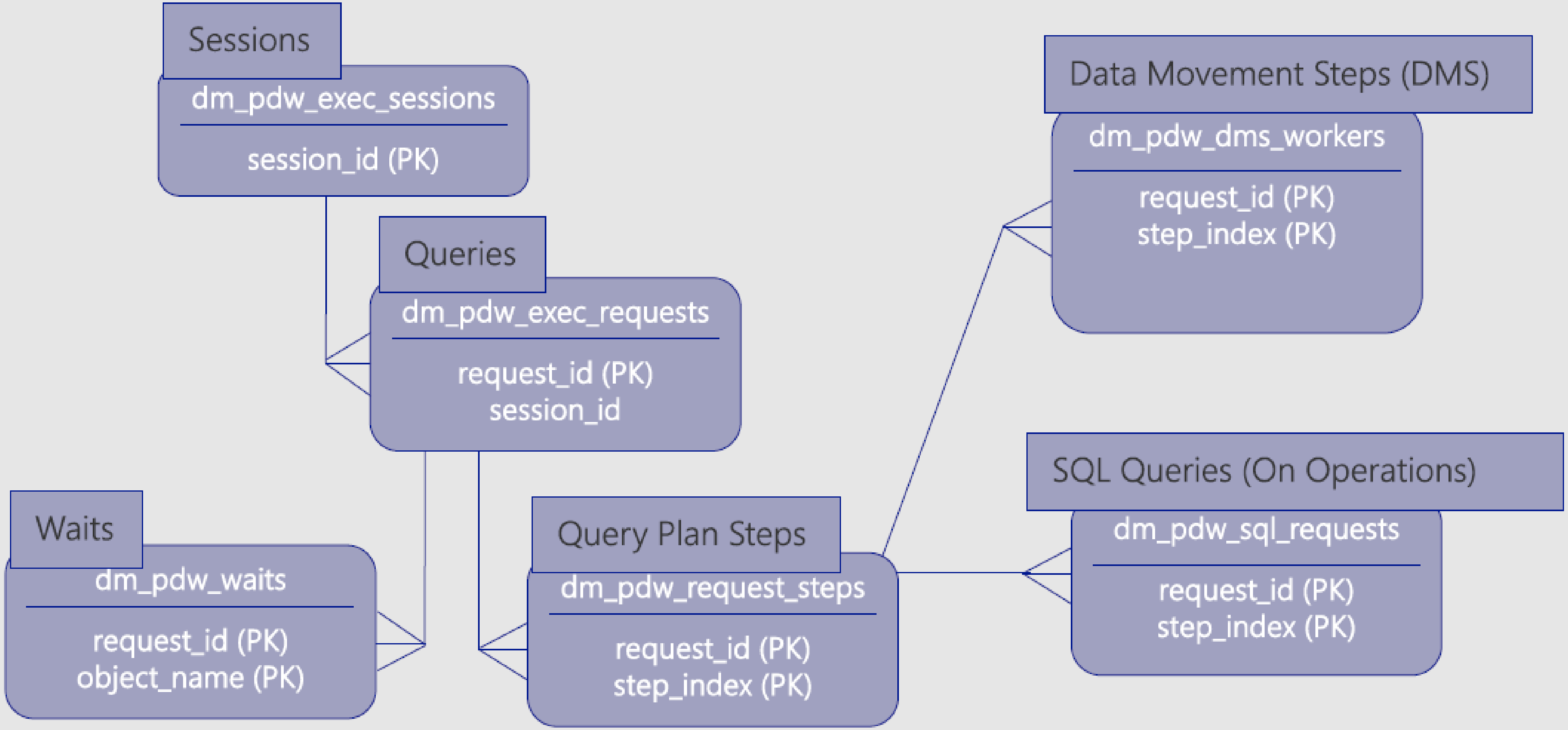
- Rich set of DMVs
- Lower level of diagnostics

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



Sessions

dm_pdw_exec_sessions

session_id (PK)

Queries

dm_pdw_exec_requests

request_id (PK)
session_id

Waits

dm_pdw_waits

request_id (PK)
object_name (PK)

Query Plan Steps

dm_pdw_request_steps

request_id (PK)
step_index (PK)

Data Movement Steps (DMS)

dm_pdw_dms_workers

request_id (PK)
step_index (PK)

SQL Queries (On Operations)

dm_pdw_sql_requests

request_id (PK)
step_index (PK)

Additional Resources

SQL Data Warehouse Documentation - https://docs.microsoft.com/en-us/azure/sql-data-warehouse/

Microsoft

Q&A