Is It Enough?

We helped MagicWorks™ design their data model then left them to finish data loading… but it hasn't gone well

They need help tuning their data loading processes to get maximum performance from the SQLDW

Agenda

Loading for performance

Partitioning

Transformations

Load optimizations

Best practices

Loading for performance

ELT for maximizing scan performance

Do large batch loads

Follow table design best practices

Don’t over partition

Use more memory to avoid premature trimming

Manage updates and deletes carefully

Achieve segment elimination by ordering or partitioning

Do Large Batch Loads

Target >100,000 per columnstore in each load

- With no partitions, this means > 100,000 * 60 rows (~6 million) rows per CTAS or bulk insert
- With 4 partitions , this means > 100,000 * 60 * 4 (~24 million) rows per CTAS or bulk insert

Ideally 1 million per Columnstore in each load

- With no partitions, this means > 1,000,000 * 60 rows (~60 million) rows per CTAS or bulk insert
- With 4 partitions , this means > 1,000,000 * 60 * 4 (~240 million) rows per CTAS or bulk insert

Batching trickle loads

Scenario: 100,000 rows per Columnstore, no partitions
6,000,000+ rows required for each batch load

	500 Rows/Sec	1000 Rows/Sec	2000 Rows/Sec
Load threshold exceeded (hours)	<3.5 hours	< 2 hours	< 1 hour

Partitioning Guidance

Row groups cannot cross partition boundaries

- Over-partitioning impacts row group quality
- Over-partitioning impacts compression

Make sure your targeted data set allows for at least 6 million rows per partition. Ideally 60 million rows per partition!

Loading across partitions can impact performance

Avoid getting trimmed row groups

Compute memory required for quality row groups

- Memory guidance for Columnstore
- Use views provided.

Grant sufficient memory

- Use correct resource class (avoid smallrc)
- Scale if needed
- Keep load query simple – stage to a Heap/CI if needed
- Force serial execution if needed

Avoid dictionary pressure

- Isolate problematic string columns into a separate table if needed

Stage data with row stores for transformation

Use heaps or clustered indexes when

- Many modifications required to base data
- Data requires transformation to match target table

Alter Index Reorganize/Rebuild to defragment

- Reorganize is lighter weight and online
- Rebuild is heavy weight and offline

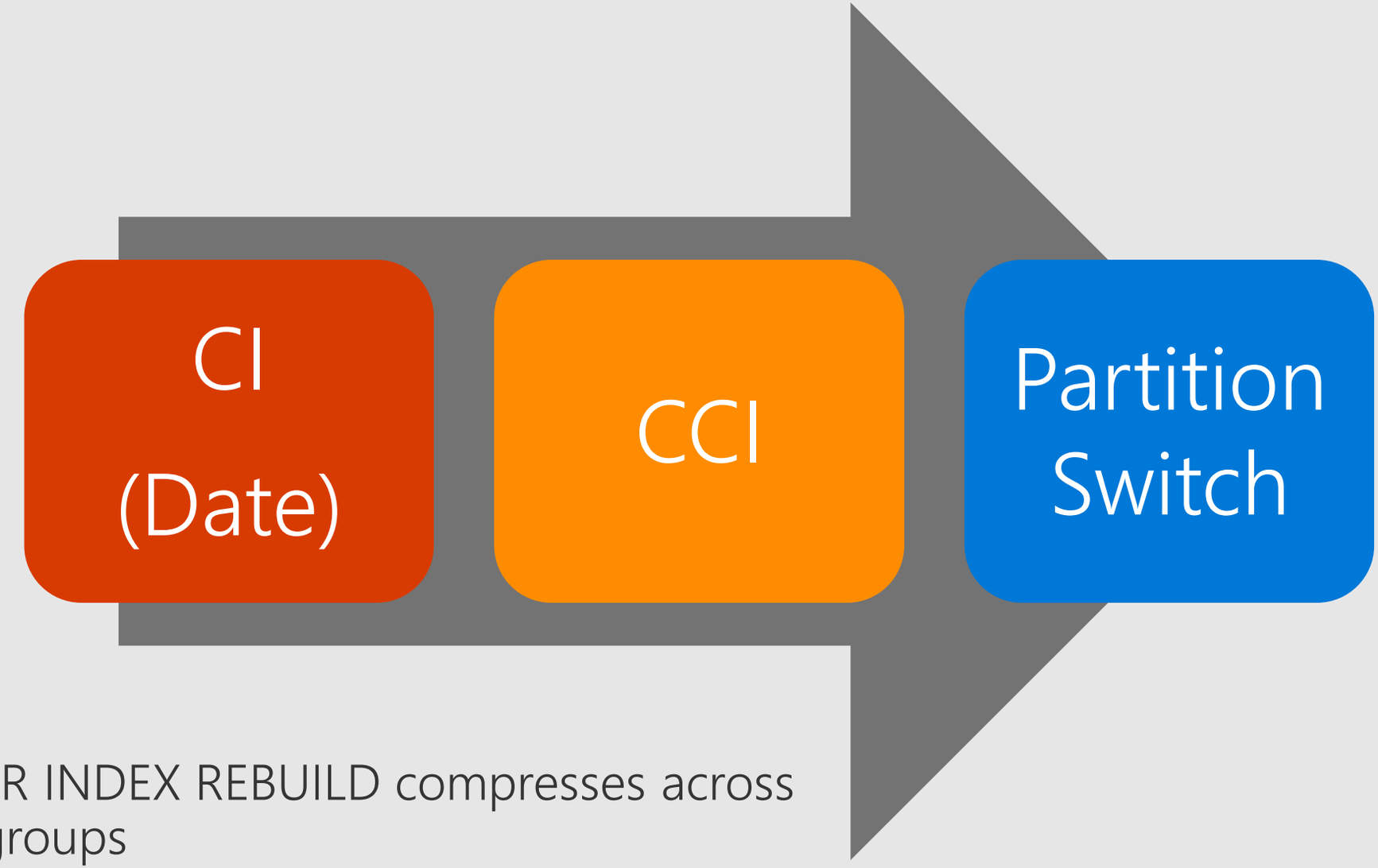
Segment Elimination

Columnstore is not ordered

- Min,max metadata used to filter out segments
- Helps if data arrives naturally ordered e.g. timestamp
- Rebuilding indexes will not keep ordering intact

Min	Max
Col 1 val	Col1 val
Col 2 val	Col 2 val
…	…

Ordered CCI for improved segment elimination



ALTER INDEX REBUILD compresses across rowgroups

Removes any manual ordering from CCI

Surrogate keys

Identity ≠ Unique

- Can have holes and duplicates

Current Limitations:

- Cannot CTAS with Identity:
- CREATE TABLE then INSERT…INTO
- Identity column cannot be distribution key

Using the IDENTITY property

```
CREATE TABLE dbo.Dim1
(C1 INT IDENTITY(1,1)
,C2 INT
)
WITH
(DISTRIBUTION = HASH(C2)
,CLUSTERED COLUMNSTORE INDEX
)
;
```

Data vault key: Use HASHBYTES

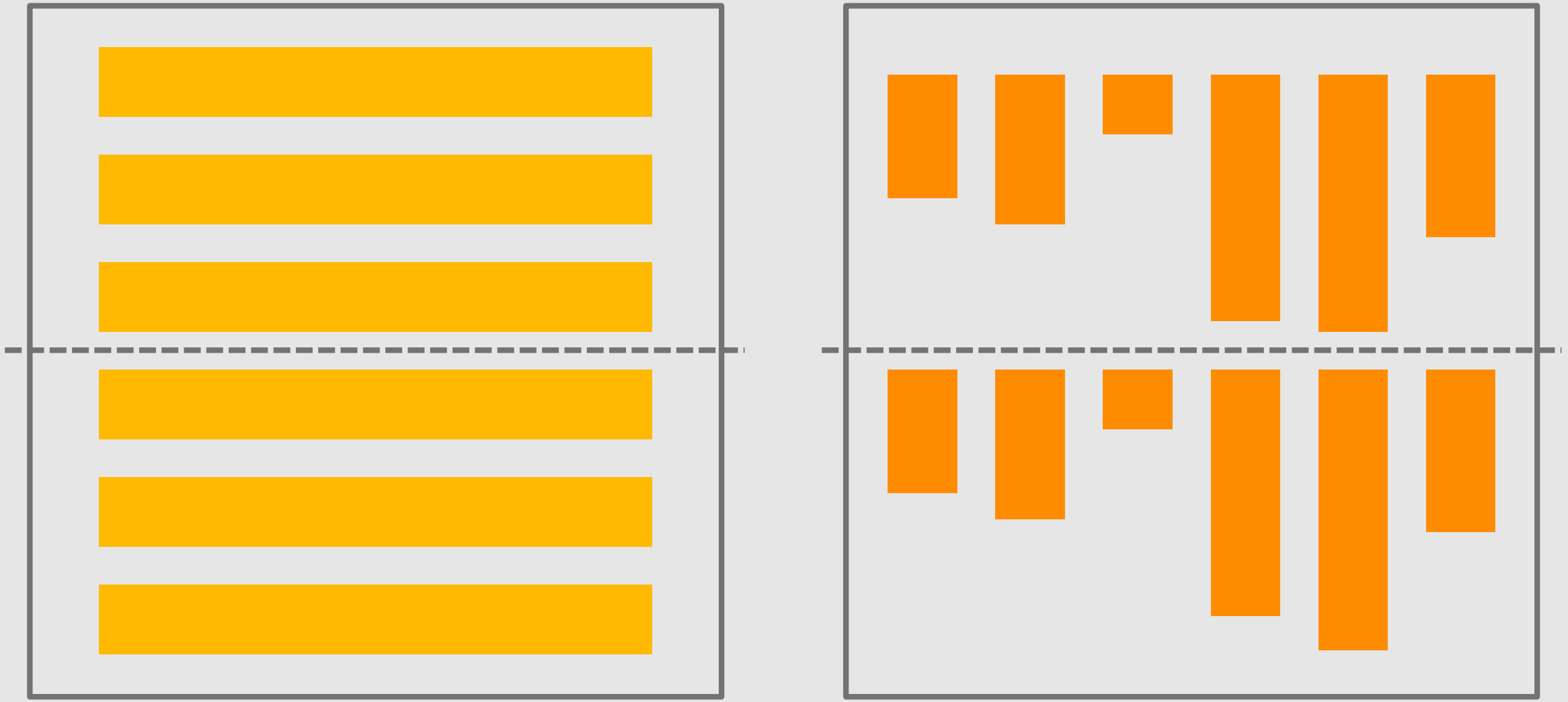
```
DECLARE @i VARCHAR(8000) = REPLICATE('X',8000)

SELECT
LOWER(CONVERT(CHAR(32),HASHBYTES('MD5',@i),2))
;
SELECT
LOWER(CONVERT(BIGINT,HASHBYTES('MD5',@i),2))
;
```

Lab 005 - Managing Surrogates

Table partitioning

Row & Column Store & Partition



Why Partition?

Benefit to Loads

- Data Lifecycle Management
  - Drop partition avoids transaction logging
  - Insert to empty table/partition avoids transaction logging

  ⇒ **Partition Switching pattern**
- Targeted Index Builds

Benefit to Queries

- Partition Elimination

CREATE TABLE with partitions

```
CREATE TABLE [cso].[FactOnlineSales_PTN]
(    [OnlineSalesKey]       int          NOT NULL
,    [DateKey]              datetime     NOT NULL
,    [StoreKey]             int          NOT NULL
,    [ProductKey]           int          NOT NULL
,    [CurrencyKey]          int          NOT NULL
,    [SalesQuantity]        int          NOT NULL
,    [SalesAmount]          money        NOT NULL
,    [UnitPrice]            money            NULL
)
WITH
(   CLUSTERED COLUMNSTORE INDEX
,   DISTRIBUTION = HASH([ProductKey])
,   PARTITION
    (
        [DateKey] RANGE RIGHT FOR VALUES
        (
            '2007-01-01 00:00:00.000','2008-01-01 00:00:00.000'
        ,   '2009-01-01 00:00:00.000','2010-01-01 00:00:00.000'
        )
    )
)
;
```

Creating a partitioned table with CTAS

```
CREATE TABLE [cso].FactOnlineSales_PTN
WITH
(   CLUSTERED COLUMNSTORE INDEX
,   DISTRIBUTION = HASH([ProductKey])
,   PARTITION
    (
        [DateKey] RANGE RIGHT FOR VALUES
        (
              '2007-01-01 00:00:00.000','2008-01-01 00:00:00.000'
         ,    '2009-01-01 00:00:00.000','2010-01-01 00:00:00.000'
        )
    )
)
AS
SELECT  *
FROM    [cso].[FactOnlineSales]
;
```

Partitioning guidance

Partition for data management

- Don’t need to use partition elimination for faster performance

Don’t over partition!

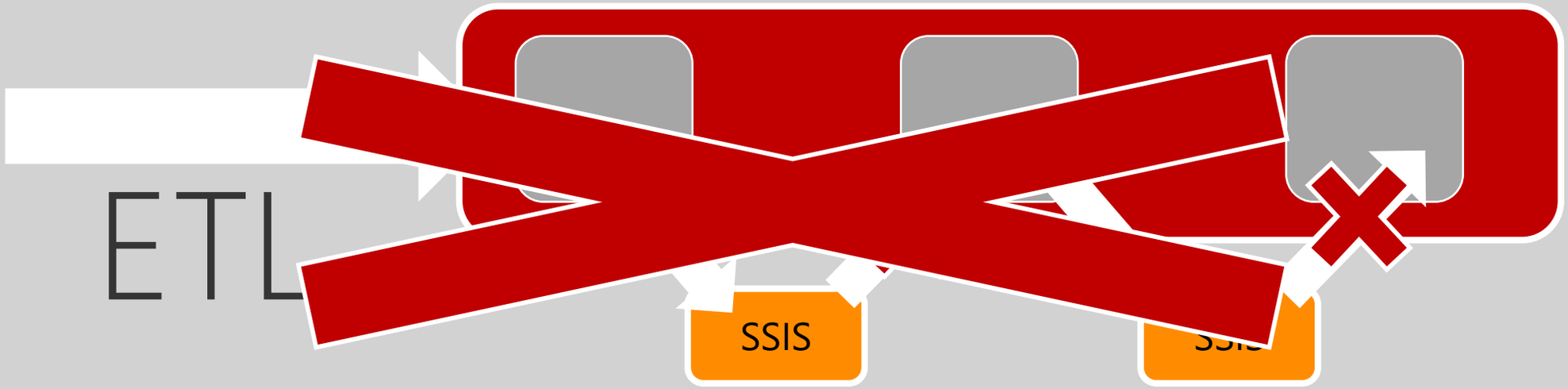
- Data already spread across 60 distributions
- Partitioning granularity likely to differ to SQL Server
- Columnstore index row groups give ideal performance when full
- Full rowgroups need at least 60 million rows per partition

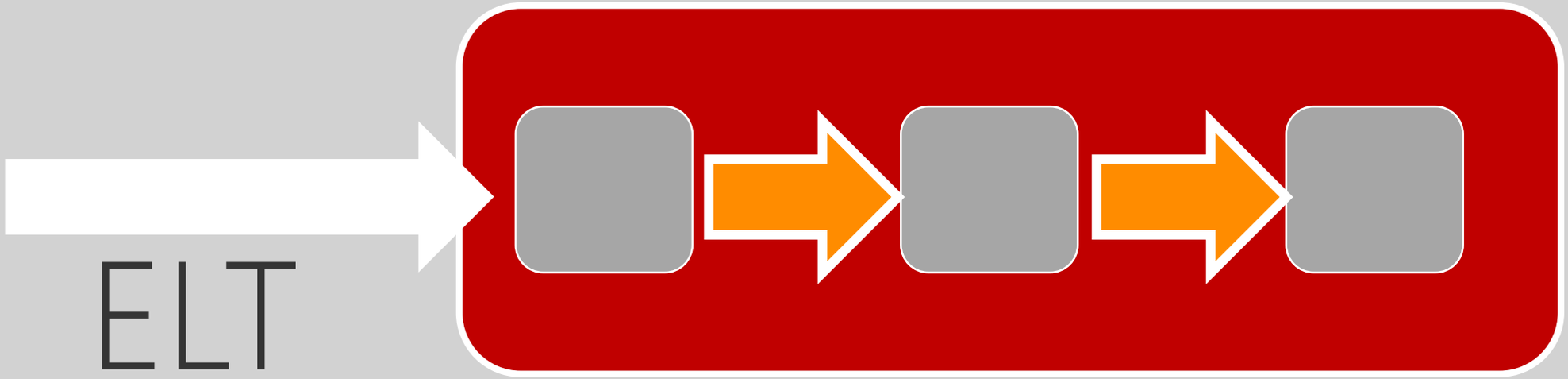
Example

Table has 60 billion rows spanning 3 years

Partition granularity	# Rows per distribution	# Partitions	# Rows per partition	# Row groups per partition
Year	1,000,000,000	3	333,333,333	334
Quarter	1,000,000,000	12	83,333,333	84
Month	1,000,000,000	36	27,777,777	28
Day	1,000,000,000	1095	913,242	1
No partitioning	1,000,000,000	1	1,000,000,000	1000

Transformations


ETL
SSIS


ELT

CTAS

```
CREATE TABLE #Nums
WITH (DISTRIBUTION=REPLICATE,LOCATION=USER_DB)
AS
WITH    L0      AS (SELECT 1 AS c UNION ALL SELECT 1)
,       L1      AS (SELECT 1 AS c FROM L0 AS A, L0 AS B)
,       L2      AS (SELECT 1 AS c FROM L1 AS A, L1 AS B)
,       L3      AS (SELECT 1 AS c FROM L2 AS A, L2 AS B)
,       L4      AS (SELECT 1 AS c FROM L3 AS A, L3 AS B)
,       L5      AS (SELECT 1 AS c FROM L4 AS A, L4 AS B)
,       Nums    AS (SELECT ROW_NUMBER() OVER(ORDER BY c) AS n FROM L5)
SELECT  CAST(n AS BIGINT) as Number
FROM    Nums
WHERE   n BETWEEN @num_Start AND @num_End
OPTION (LABEL='fn_nums : #nums create')
;
```

Based on fn_nums by Itzik Ben-Gan

CTAS vs. Create Table

CTAS

Table populated

Rowcount set to actual

Data Type from Select

Nullability from Select

Default constraints: no

CREATE TABLE

Table is empty

Rowcount set to 1000

Data types from DDL

Nullability from DDL

Default constraints: yes

CTAS vs. INSERT…SELECT

CTAS

New object created

No column statistics on table

Row count updated

Page count updated

INSERT..SELECT

Existing object populated

Any stats maintained – UPDATE

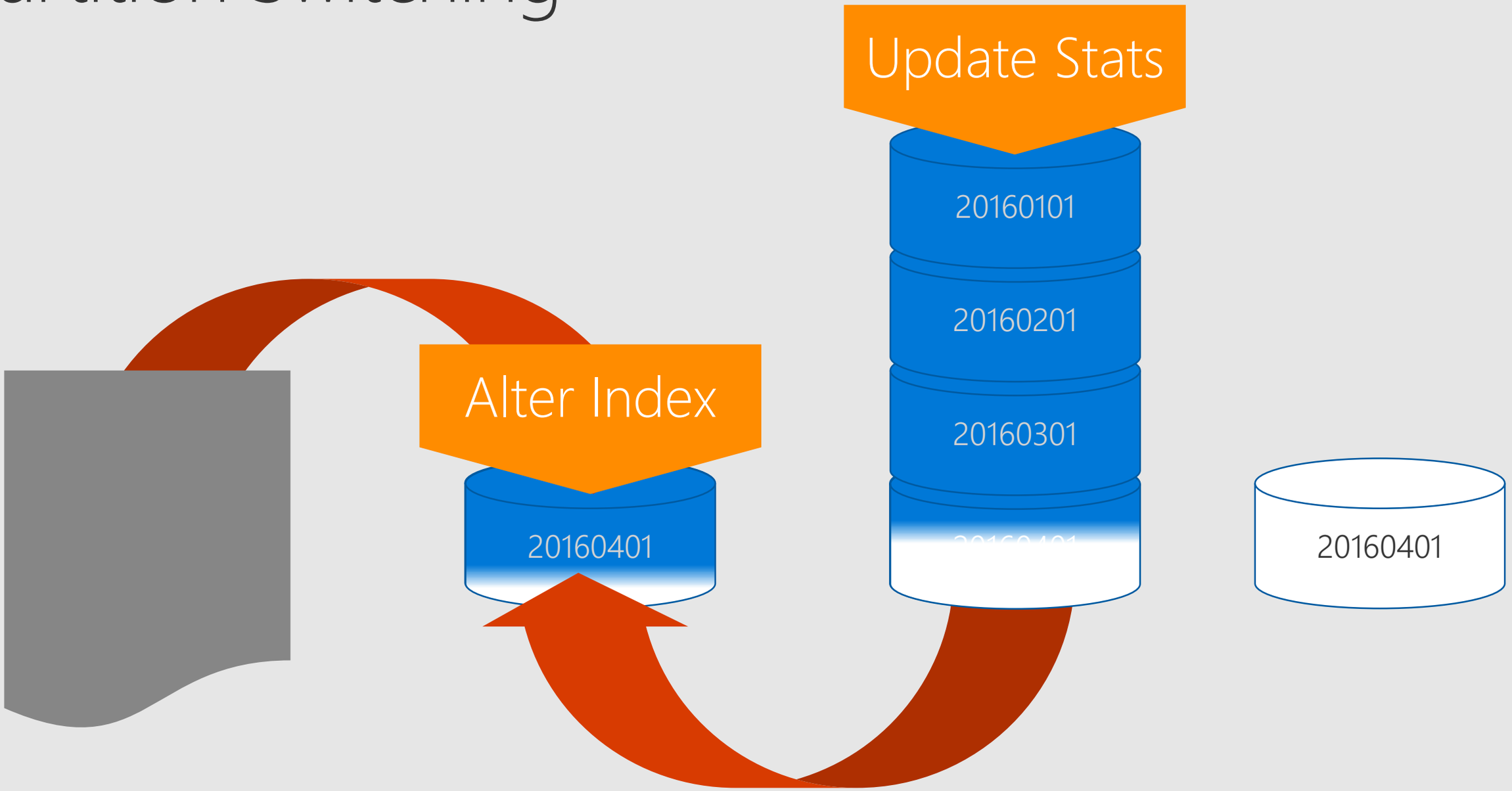
Row count not updated

Page count not updated

UPSERTing data with CTAS

```
CREATE TABLE [tmp].[DimProduct]
WITH (DISTRIBUTION = ROUND_ROBIN)
AS -- New rows and new versions of rows
SELECT      s.[ProductKey]
,           s.[ProductName]
,           s.[ColorName]
FROM        [src].[DimProduct] s
UNION ALL --Keep rows that are not being updated
SELECT      p.[ProductKey]
,           p.[ProductName]
,           p.[ColorName]
FROM        [cso].[DimProduct] p
WHERE NOT EXISTS
(   SELECT *
    FROM   [src].[DimProduct] s
    WHERE  s.[ProductKey] = p.[ProductKey]
)
;
```

Partition Switching



CTAS – Create partition for switch out

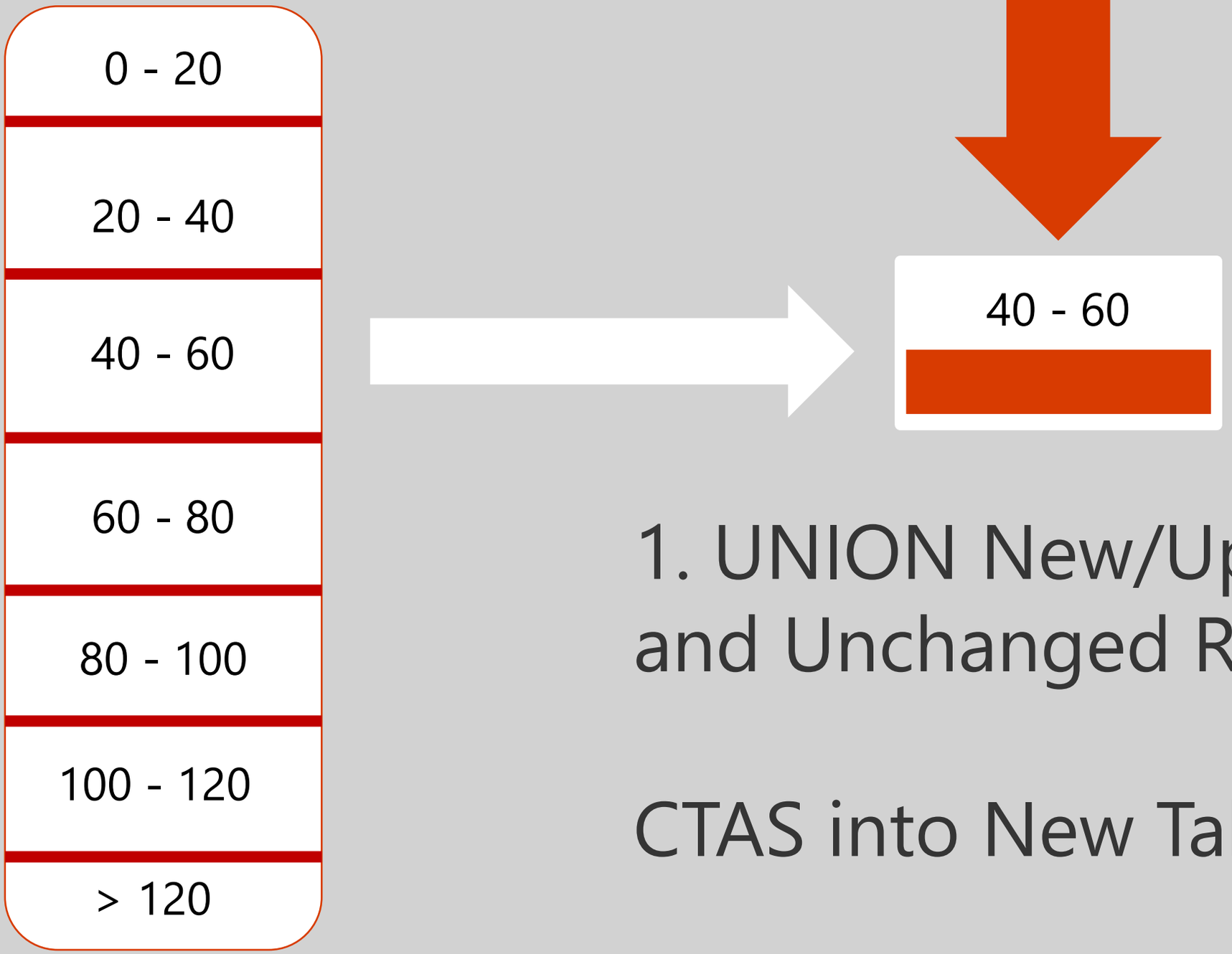
```
CREATE TABLE [cso].[FactOnlineSales_out]
WITH
(   DISTRIBUTION=HASH ([ProductKey])
,   CLUSTERED COLUMNSTORE INDEX
,   PARTITION ([DateKey]
    RANGE RIGHT FOR VALUES  ('2007-01-01 00:00:00.000'
                            ,'2008-01-01 00:00:00.000'
                            )
            )
)
AS
SELECT *
FROM   [cso].[FactOnlineSales]
WHERE 1=2;
```

CTAS – Create Partition for Switch In

```
CREATE TABLE [cso].[FactOnlineSales_in]
WITH
(   DISTRIBUTION=HASH ([ProductKey])
,   CLUSTERED COLUMNSTORE INDEX
,   PARTITION ([DateKey]
    RANGE RIGHT FOR VALUES ('2007-01-01 00:00:00.000','2008-01-01 00:00:00.000'
                            )
             )
)
AS
SELECT *
FROM   [cso].[FactOnlineSales_ptn] tgt
WHERE  tgt.[DateKey] >= '2007-01-01 00:00:00.000'
AND    tgt.[DateKey] <  '2008-01-01 00:00:00.000'
UNION ALL
SELECT *
FROM   [cso].[FactOnlineSales] stg
WHERE stg.[DateKey] >= '2007-01-01 00:00:00.000'
AND   stg.[DateKey] <  '2008-01-01 00:00:00.000'
;
```

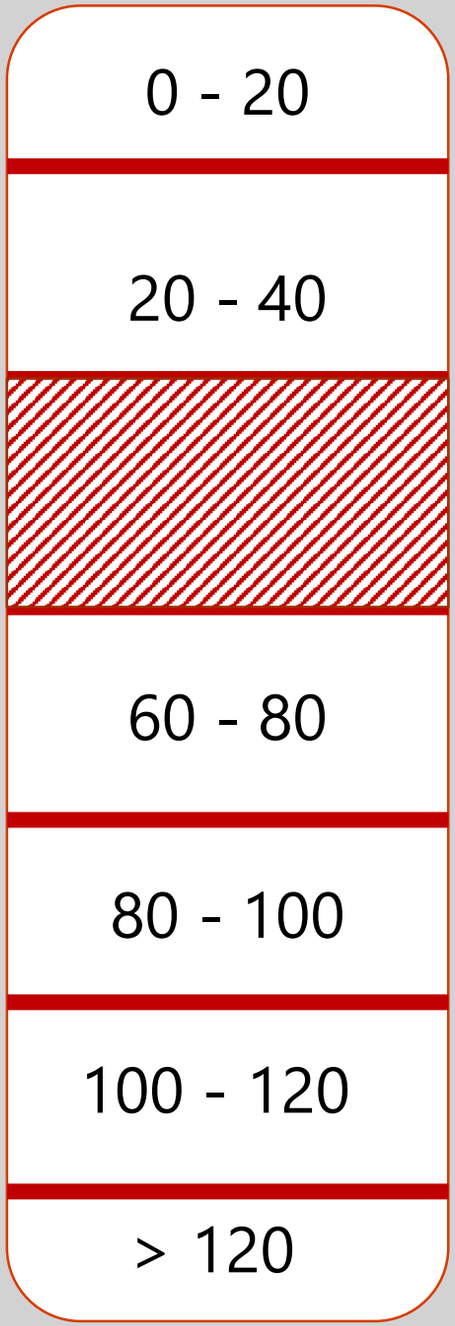
Perform the switches

```
ALTER TABLE [cso].[FactOnlineSales_ptn]
SWITCH PARTITION 2
TO [cso].[FactOnlineSales_out] PARTITION 2
;
ALTER TABLE [cso].[FactOnlineSales_in]
SWITCH  PARTITION 2
TO [cso].[FactOnlineSales_ptn] PARTITION 2
;
```



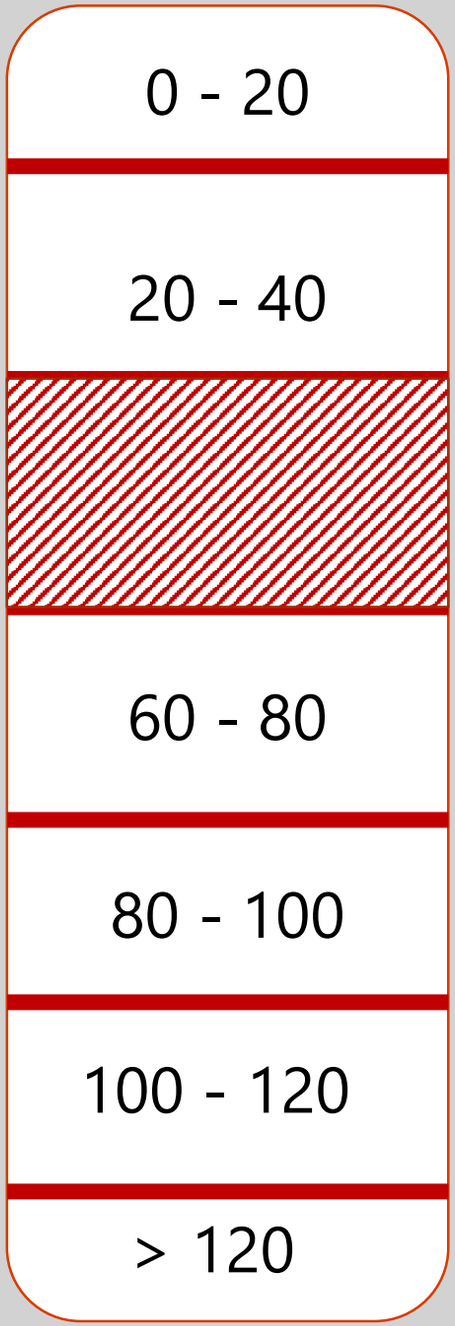
1. UNION New/Updated and Unchanged Records

CTAS into New Table



2. Switch Out Old Partition

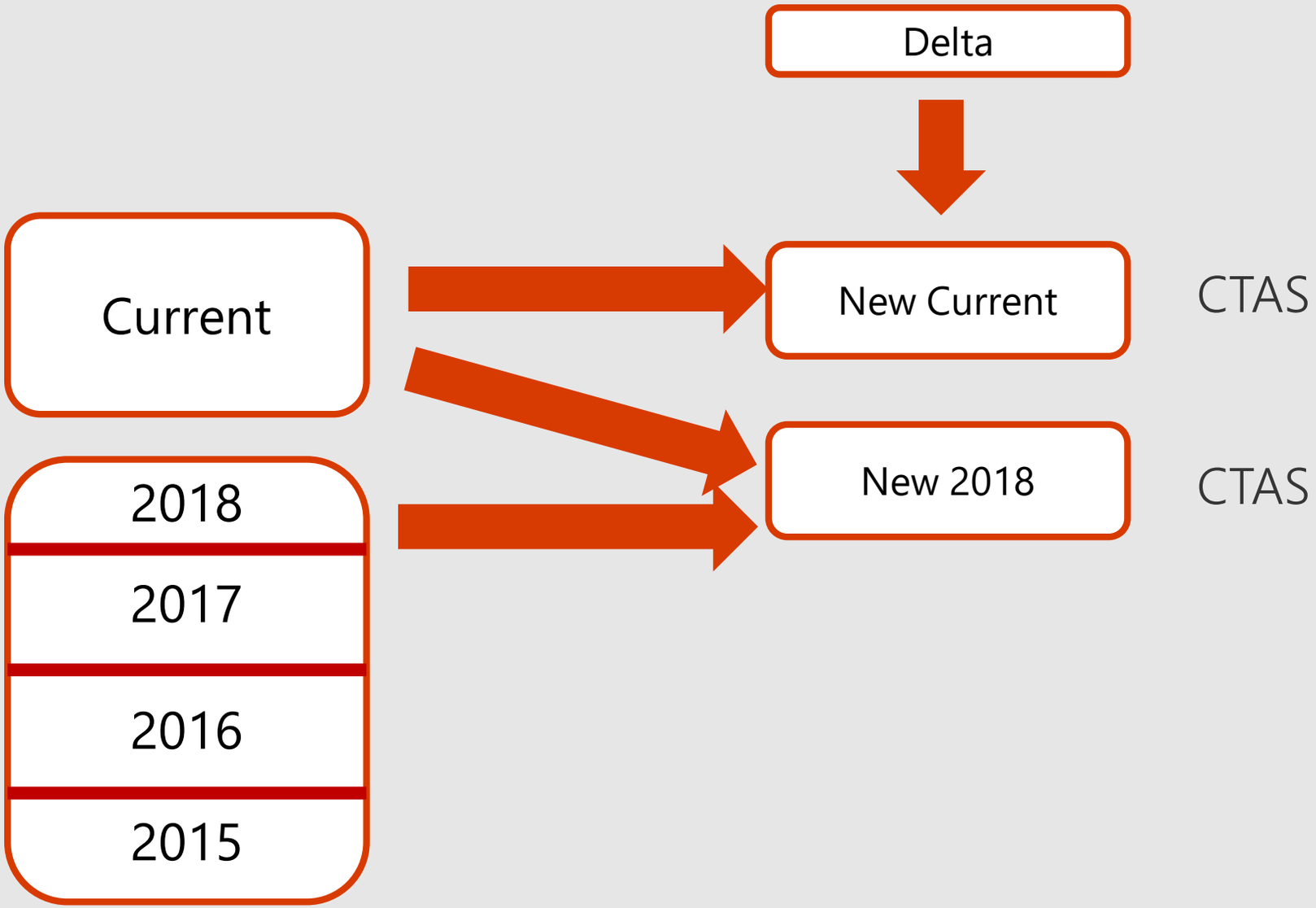
ALTER TABLE dbo.Fact SWITCH PARTITION 3 TO dbo.Fact_out;



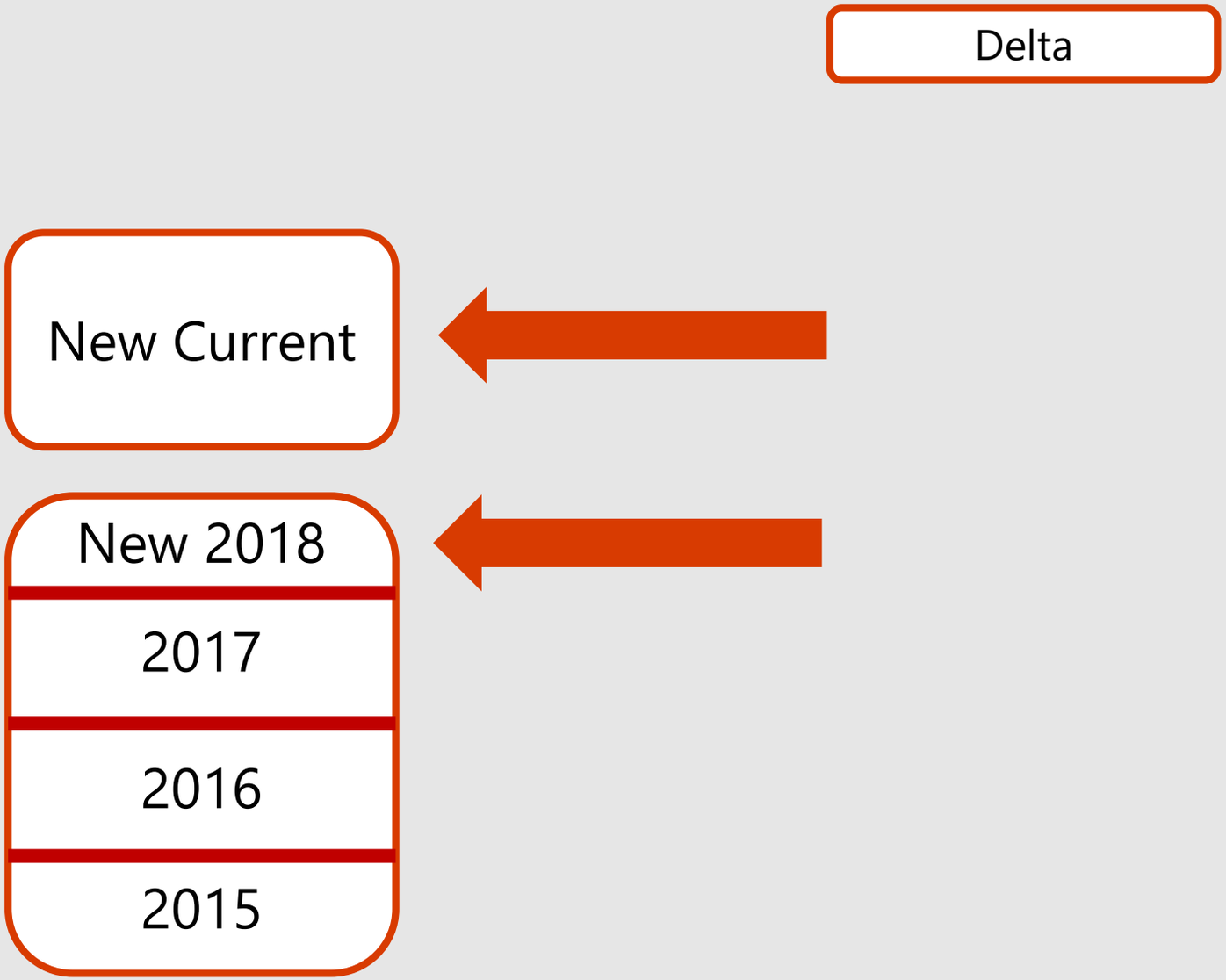
3. Switch In New Partition

ALTER TABLE dbo.Fact_new SWITCH TO dbo.Fact PARTITION 3;

Applying Type 4 Partitioning



Applying Type 4 Partitioning



Demo: Partitioning Scripts

Each data movement is performed by a stored procedure that follows a standard pattern, using a set of utility procedures

- Begin Audit
- CTAS
- Gather Rowcount
- Generate Stats
- Finalise Audit

Load optimizations

Factors improving performance

Index choice

System scale

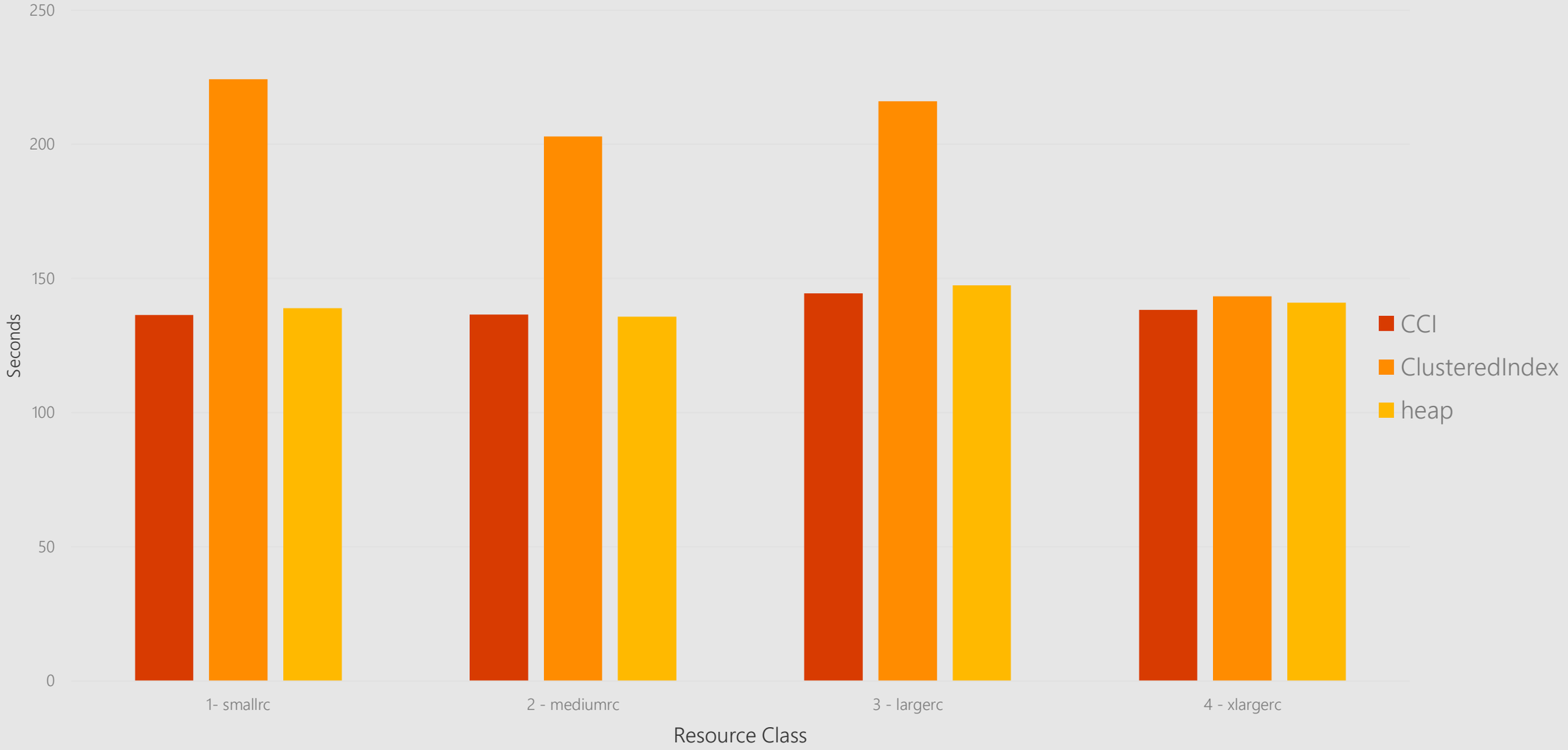
Distribution type

File format type

File system layout

Row number in file

Index type matters



Scaling matters

DWU	Max External Readers	Max Writers CCI / Heap	Max Writers CI / NCI
100	8	60	60
200	16	60	60
300	24	60	60
400	32	60	60
500	40	60	60
600	48	60	60
1000	80	60	60
1200	96	60	60
1500	120	120	60
2000	160	120	60
3000	240	240	60
6000	480	480	60

Scale matters

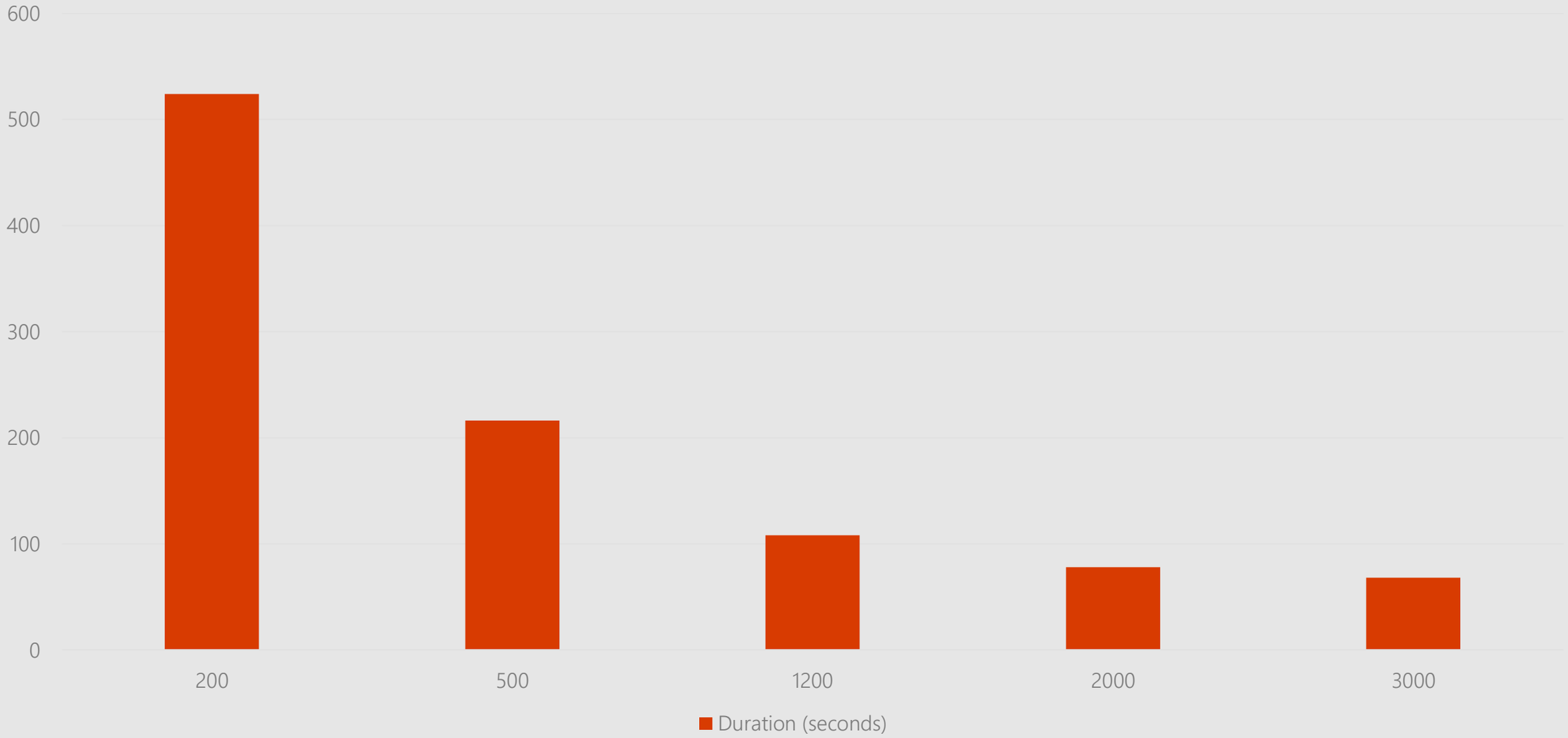
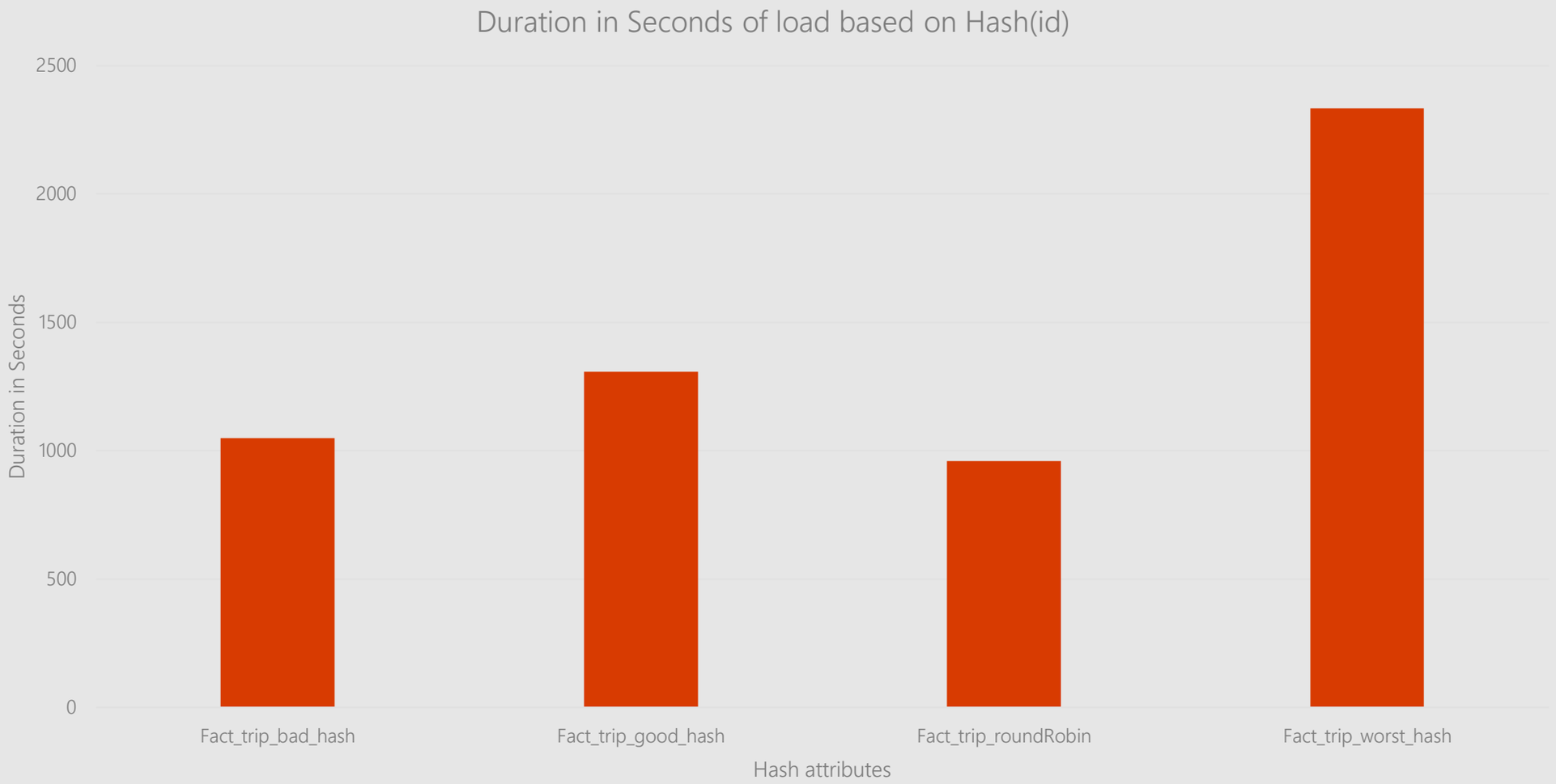
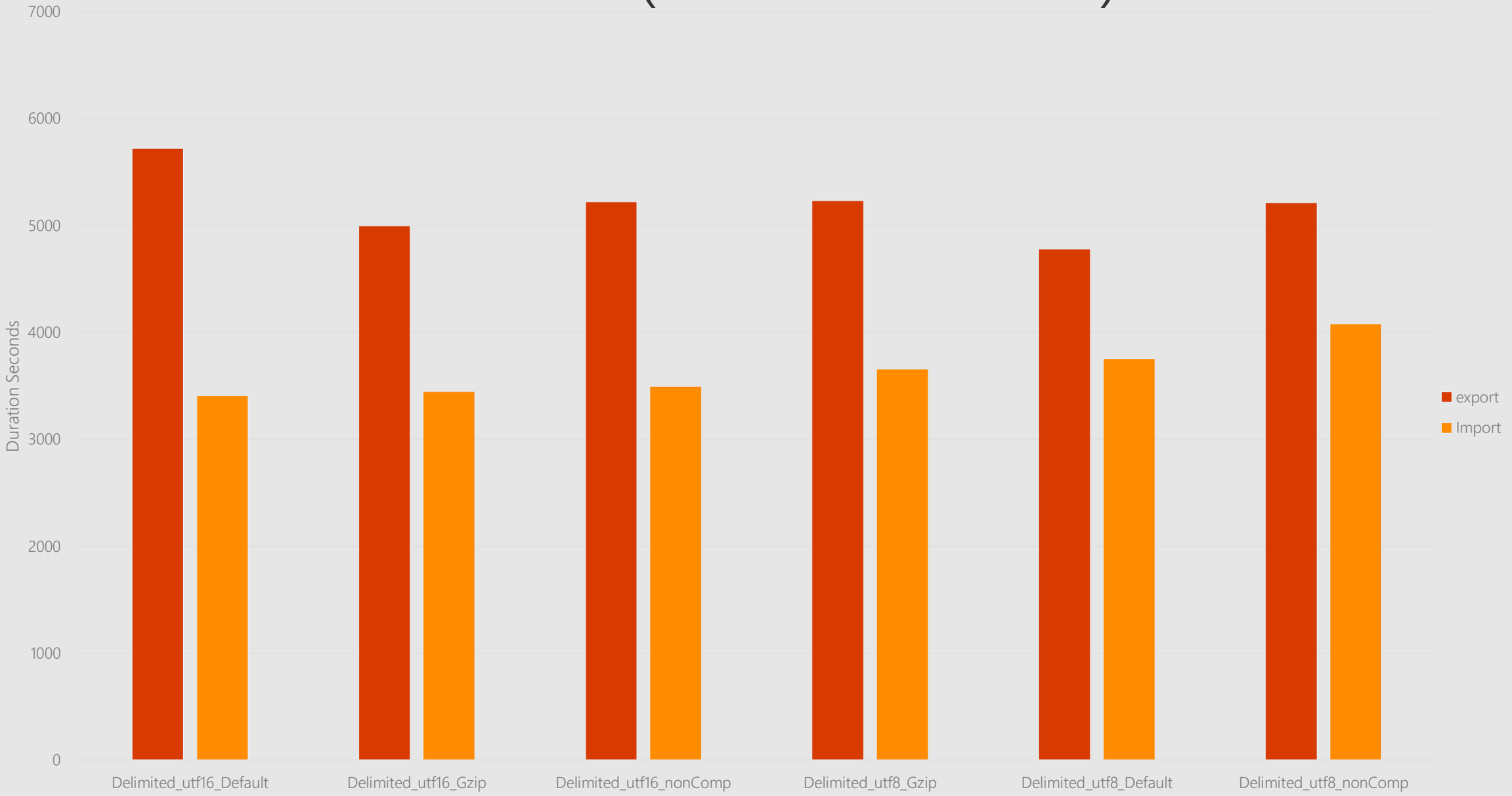


Table distribution matters



File format matters (delimited text)



Delimited text guidance

Evenly split the data into multiple files

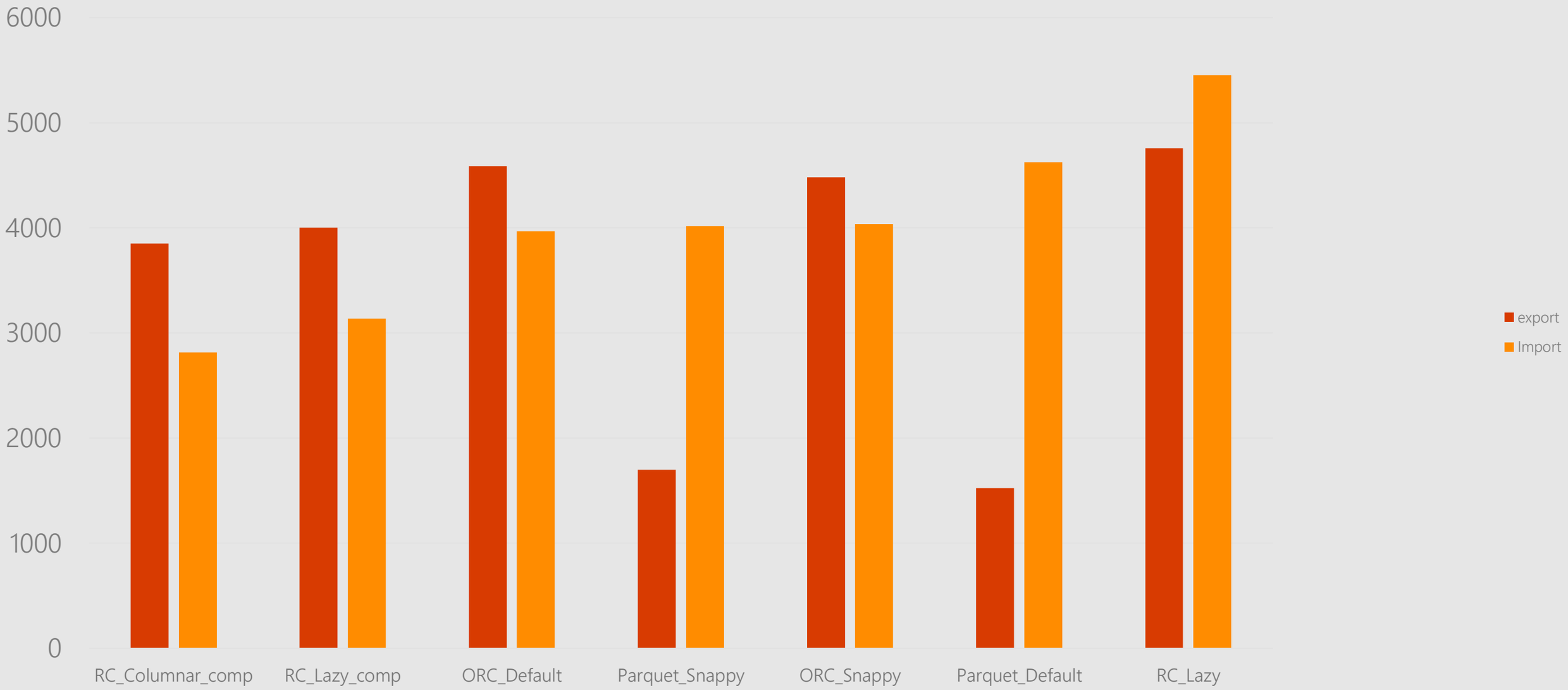
One file per reader

Delimited text is the fastest

Compressed text limits concurrent access to text files

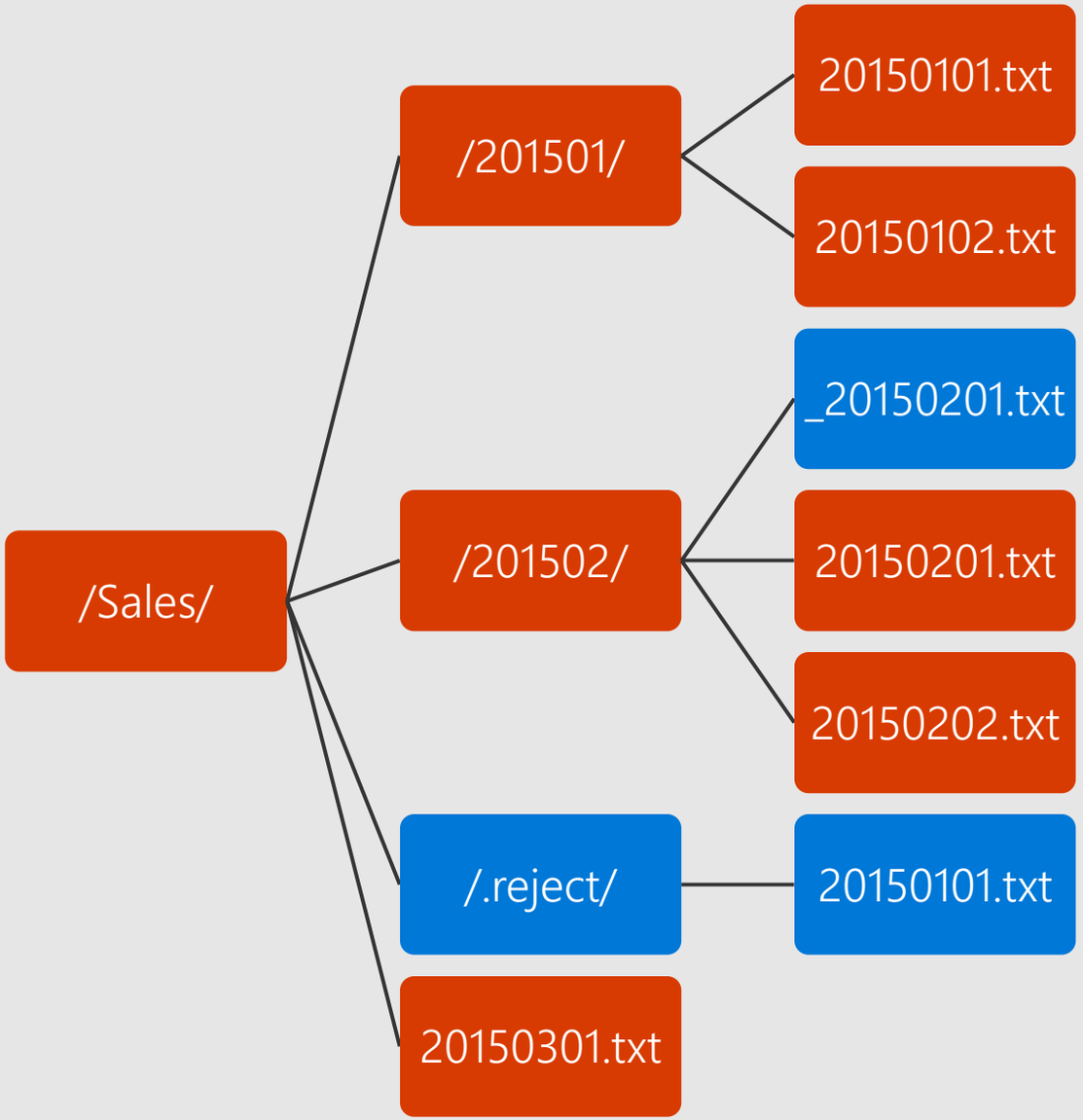
Split data across files
OR
use different file format

File format matters (Big Data)



File system layout

Objects prefixed with _ and . are ignored recursively



Optimised Source file

```
RowNumber,Operation,ProductKey,EnglishProductName,Color
1,I,10000,GriffyndorAreMagic,Red
2,U,10100,RavenclawAreStillQuiteGood,Yellow
3,D,10200,SlytherinPleaseFinishThird,White
4,I,10300,CongratulationsHufflepuff,Blue
```

Upsert Logic – CTAS optimised

```
CREATE TABLE tmp.DimProduct
WITH (DISTRIBUTION = ROUND_ROBIN)
AS -- New rows and new versions of rows
SELECT      s.ProductKey
,           s.ProductName
,           s.ColorName
FROM        [src].[DimProduct] s
WHERE       s.Operation IN ('I','U')
UNION ALL --Keep rows that are not being updated
SELECT      p.ProductKey
,           p.ProductName
,           p.ColorName
FROM        [cso].[DimProduct] p
WHERE NOT EXISTS
(   SELECT *
    FROM   [src].[DimProduct] s
    WHERE  s.ProductKey = p.ProductKey
);
```

Best Practices

Loading takeaways

Index Choice

- Data shows not much difference between CCI and heap tables
- Network is the bottleneck

DWU

- Increasing scale of system automatically increases load performance
- Particularly evident at the low-end

Distribution type

- Round Robin fastest
- Poor distributions can result in bad performance

Loading take-aways

File type

- UTF-16 faster due to less time converting encoding

Resource class

- Matters most for column store segment health than load speed

Dimension tables

Use round robin for small tables

- Perform all DML before inserting into Replicated table

Use clustered index for ordered scans

- Clustered columnstore index is not ordered
- Small tables (<60M) may benefit more from being row stores

Load in full where possible

- Dimensions are typically small and are typically mastered in MDM system

Use metadata rename to reload data

- Keeps old and new versions available for easy comparison

Fact tables

Land data from ASB in a staging table

- Adheres to an ELT pattern
- Greater control in T-SQL

Partitions reduce loading impact on production tables

- Partition switch is a metadata operation

ASB directory structure can scope load

- Improves performance
- Limits compute utilization

Summary

Summary

Do not load data via singleton inserts

- Very inefficient

Use PolyBase for batch loading

- Land data in blob storage first
- Use stored procedures to maximize control
- Adhere to an ELT load pattern

Trickle load/micro batch

- Use bcp if PolyBase isn't an option

So What?

Implement ELT Loading Scripts to:

- Use CTAS
- Partition Switch
- Index Design

Data Loading Script

Structure flat file storage effectively