WHY A CLOUD DATA PLATFORM?

	On Premises EDW	1st Gen Cloud EDW	Data Lake, Hadoop	Cloud Data Platform
All Data	✗	✗	✓	✓
All Users	✗	✗	—	✓
Fast Performance	✓	✓	✗	✓
Easy to use	✓	✓	✗	✓

HISTORICAL PERFORMANCE ASSUMPTIONS

Limited, fixed resources

Resources are fixed, so must be sized to the maximum load and protected from overuse

Tune for performance

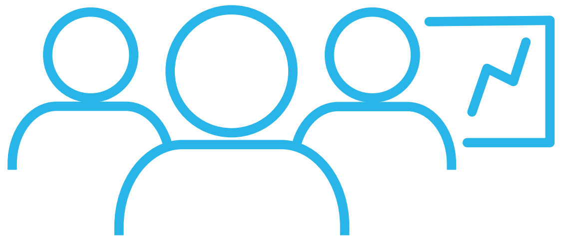
Performance issues must be dealt with through tuning, indexing, re-partitioning, and other “knob” turning

Manual upkeep

Performance tuning and software upgrades are manual, requiring ongoing maintenance over time

SNOWFLAKE APPROACH TO PERFORMANCE

Workload Isolation

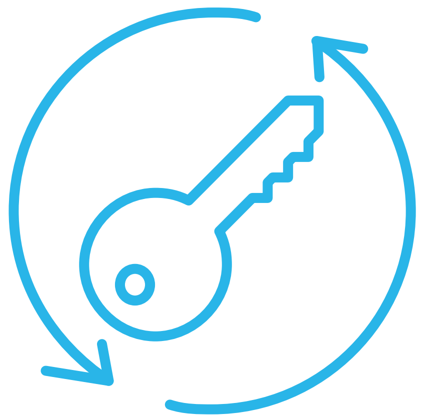


Unlimited compute clusters
serve a diverse array
of workloads, on-demand

Simplicity

The Snowflake service
self-tunes. The user has the
option of using materialized
views or cluster keys to further
enhance performance if needed

Automation



All performance processes in
Snowflake were designed to
run or maintain themselves,
and software updates are
automatic

SNOWFLAKE EDITIONS

STANDARD

Complete SQL data warehouse

Secure data sharing

Premier support 24x365

1 day of time travel

Enterprise-grade encryption

Dedicated virtual warehouses

Federated Authentication

Database Replication

External Functions

Snowsight

Create your own data exchange

Data Marketplace access

ENTERPRISE

Standard +

Multi-cluster warehouses

Up to 90 days of time travel

Annually rekey encrypted data

Materialized Views

Search Optimization Service

Dynamic Data Masking

External Data Tokenization

BUSINESS CRITICAL

Enterprise +

HIPAA support

PCI compliance

Data encryption everywhere

Tri-Secure secure

AWS PrivateLink support

Azure Private Link support

Database failover and failback

External Functions - AWS API

Gateway Private Endpoints

support

VIRTUAL PRIVATE SNOWFLAKE (VPS)

Business Critical +

Customer-dedicated virtual servers wherever the encryption key is in memory

Customer-dedicated metadata store

TABLE METADATA

TABLE METADATA

Table Name: TABLE_1
Table ID: 1189

Table Version	Query ID	Commit Time	Current MPs
1	1234	<ts>	1, 2
2	2336	<ts>	1, 2, 3
3	3346	<ts>	1, 3, 4
4	4208	<ts>	1, 3, 4, 5, 6
5	5778	<ts>	3, 4, 5, 6, 7, 8
6	5889	<ts>	3, 4, 5, 6, 7, 8, 9
7	6993	<ts>	3, 5, 6, 7, 8, 9, 10
8	7004	<ts>	9, 10, 11, 12

- Every committed transaction against a table creates a new version of the table
- Metadata tracks version information:
  - Table version
  - Query ID of transaction that changed table
  - Timestamp when change was committed
  - Micro-partitions for table version

TABLE VERSIONS

```
CREATE TABLE mytable ...
```

ID	Name

Representation of Table Metadata

Ver	QID	Commit Time	Current MPs
1	1106	2021-01-12 09:22:37.005	-

TABLE VERSIONS

```
COPY INTO mytable ...
```

ID	Name	
1	John	MP1
2	Scott	
3	Mary	
4	Jane	MP2
5	Jack	
6	Claire	

Representation of Table Metadata

Ver	QID	Commit Time	Current MPs
1	1106	2021-01-12 09:22:37.005	-
2	1234	2021-01-12 11:41:40.846	1, 2

TABLE VERSIONS

```
INSERT INTO mytable ...
```

ID	Name	
1	John	MP1
2	Scott	
3	Mary	
4	Jane	MP2
5	Jack	
6	Claire	
7	Pierre	MP3

Representation of Table Metadata

Ver	QID	Commit Time	Current MPs
1	1106	2021-01-12 09:22:37.005	-
2	1234	2021-01-12 11:41:40.846	1, 2
3	2336	2021-01-12 13:12:10.334	1, 2, 3

TABLE VERSIONS

```
UPDATE mytable ...
```

ID	Name	
1	John	MP1
2	Scott	
3	Mary	
4	Jane	MP2
5	Jack	
6	Claire	
7	Pierre	MP3
4	Janet	MP4
5	Jack	
6	Claire	

Representation of Table Metadata

Ver	QID	Commit Time	Current MPs
1	1106	2021-01-12 09:22:37.005	-
2	1234	2021-01-12 11:41:40.846	1, 2
3	2336	2021-01-12 13:12:10.334	1, 2, 3
4	2842	2021-01-12 15:01:00.885	1, 3, 4

TIME TRAVEL

TABLE VERSIONS AND TIME TRAVEL

- Query past versions of a table using AT or BEFORE

```
SELECT * FROM <table>
   AT (timestamp => '2021-01-12 12:00:00.000'::timestamp);

SELECT * FROM <table>
   AT (offset => -600);

SELECT * FROM <table>
   BEFORE (statement => '2842');
```

TABLE VERSIONS AND TIME TRAVEL

- Using table metadata, we can construct any version of a table

```
SELECT * FROM <table>
   AT (timestamp => '2021-01-12 12:00:00.000'::timestamp);
```

Representation of Table Metadata

Ver	QID	Commit Time	Current MPs
1	1106	2021-01-12 09:22:37.005	-
2	1234	2021-01-12 11:41:40.846	1, 2
3	2336	2021-01-12 13:12:10.334	1, 2, 3
4	2842	2021-01-12 15:01:00.885	1, 3, 4

TABLE VERSIONS AND TIME TRAVEL

- Using table metadata, we can construct any version of a table

```
SELECT * FROM <table>
   AT (timestamp => '2021-01-12 12:00:00.000'::timestamp);
```

Representation of Table Metadata

Ver	QID	Commit Time	Current MPs
1	1106	2021-01-12 09:22:37.005	-
2	1234	2021-01-12 11:41:40.846	1, 2
3	2336	2021-01-12 13:12:10.334	1, 2, 3
4	2842	2021-01-12 15:01:00.885	1, 3, 4

Version 2 of the table was active at noon on 1/12/2021

Version 2 Returned

ID	Name
1	John
2	Scott
3	Mary

MANAGED ACCESS SCHEMAS

CENTRALIZE OR LOCK DOWN PRIVILEGE MANAGEMENT FOR OBJECTS

- Designed to centralize management of grants for objects

Regular Schemas	Managed Access Schemas
Object owners can grant access to their objects, including the right to further grant access to others `GRANT SELECT ON ALL TABLES IN SCHEMA <X> TO ROLE <R> WITH GRANT OPTION;`	Only the schema owner, `SECURITYADMIN`, or a custom role with `MANAGE GRANTS` privileges can grant access to the objects in the schema

Code example:
 	
`USE DATABASE myDB;`
`CREATE SCHEMA mySchema;`
	Code example:
	
	`USE DATABASE myDB;`
	`CREATE SCHEMA mySchema WITH MANAGED ACCESS;`

WHAT YOU REALLY NEED TO KNOW

- You will be given access to one or more roles
- The role you are using determines what data you can see, and what you can do with it
- Granted privileges allow you to do specific things
  - `GRANT USAGE ON WAREHOUSE elt_wh TO ROLE elt;`
  - `GRANT CREATE DATABASE ON ACCOUNT TO ROLE object_mgr;`
  - `GRANT SELECT ON ALL TABLES IN DATABASE main TO ROLE main_analyst;`
- If you create an object, the role you were using owns the object – anyone in the role can do anything with the object (and with objects contained within it)
  - If a role can create a schema, all role members can create objects inside those schemas

ACCESS METHODS FOR EXTERNAL DATA

Slowest

Type	Data Location	Micro-partition Pruning	Schema
External Data	External to Snowflake	None	Schema at query time
External Table	External to Snowflake	Coarse, based on file path	Can define external table and schema using views on table
External Table + MV	External to Snowflake; view result set is materialized	Fine, based on micro-partitions	Can define external table and schema using views on table

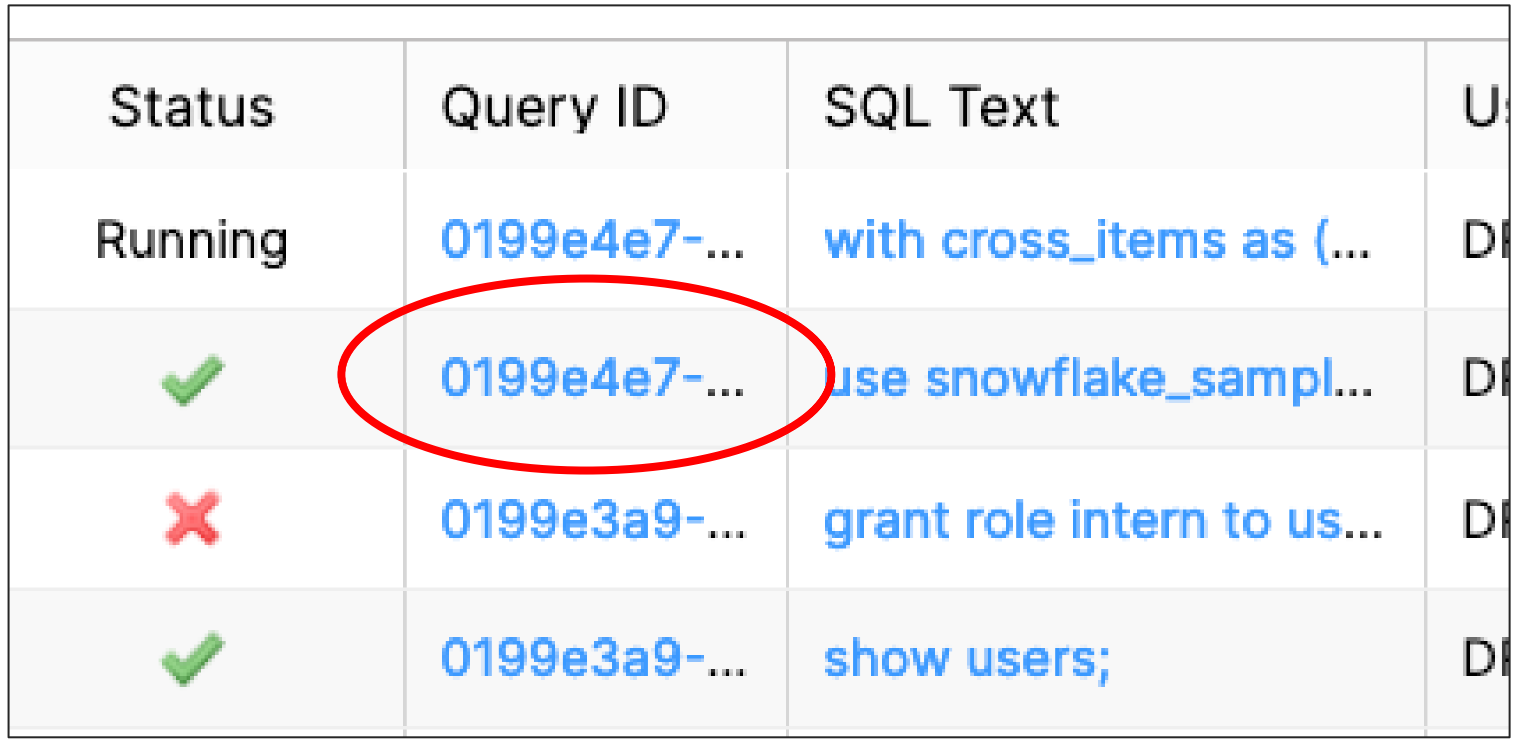
Fastest

EXPLAIN AND THE QUERY PROFILE

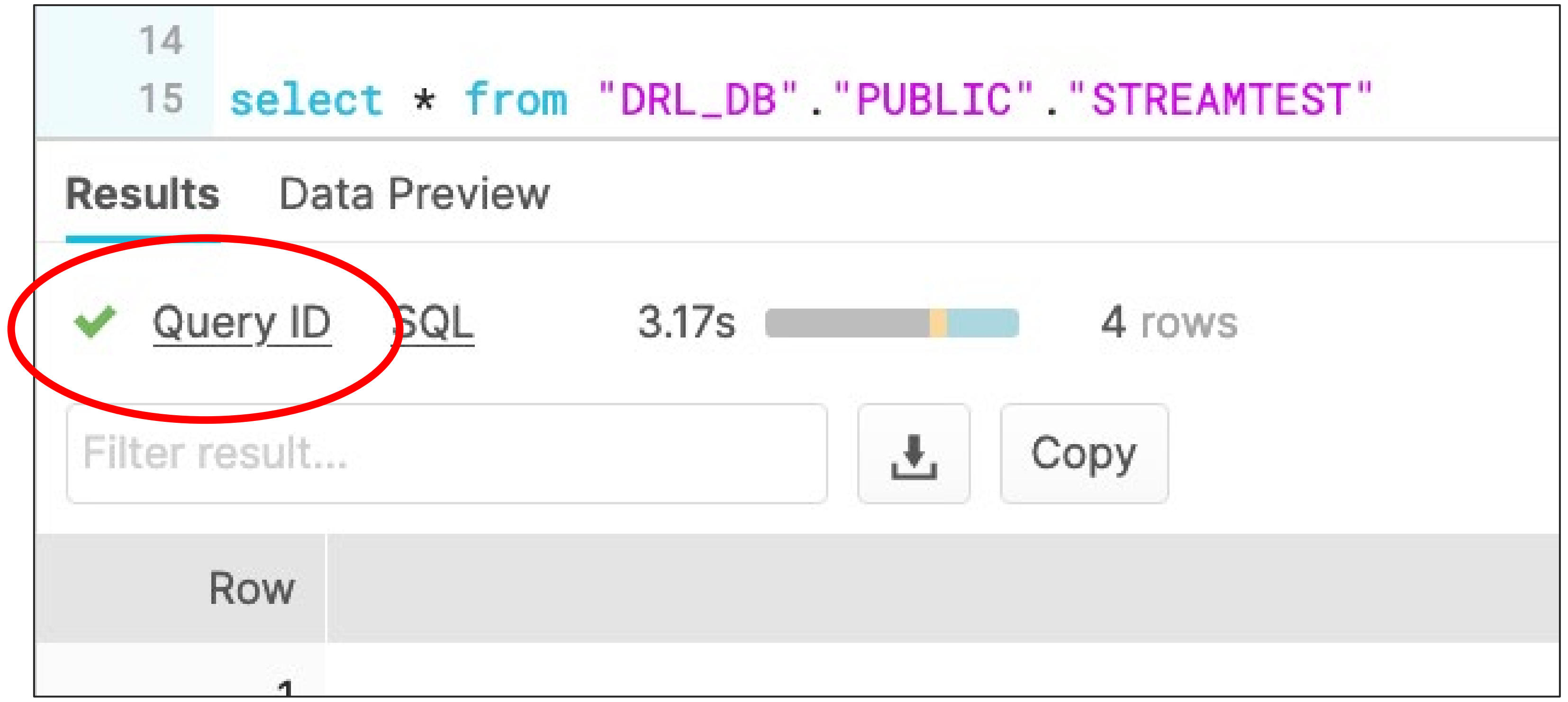
QUERY PROFILE

- One of the easiest ways to see how a query performed
- Accessed via the Query ID link in either the History tab, or the result pane in the worksheet

History Pane



Worksheet Result Pane



QUERY PROFILE

GREAT STARTING POINT FOR EVALUATING PERFORMANCE

Breakdown of time spent in phases of execution

Are you making use of the data cache?

Profile Overview (Finished)

Total Execution Time (17m 31.871s)

(100%)

Processing	62 %
Local Disk IO	17 %
Remote Disk IO	3 %
Synchronization	2 %
Initialization	16 %

Total Statistics

IO

Scan progress	100.00 %
Bytes scanned	16.02 GB
Percentage scanned from cache	3.49 %
Bytes written to result	22.98 GB

Pruning

Partitions scanned	2,044
Partitions total	2,044

Spilling

Bytes spilled to local storage	57.10 GB

How much micro-partition pruning are you getting?

Are you spilling to local or remote storage?

ANALYZE A QUERY USING `EXPLAIN`

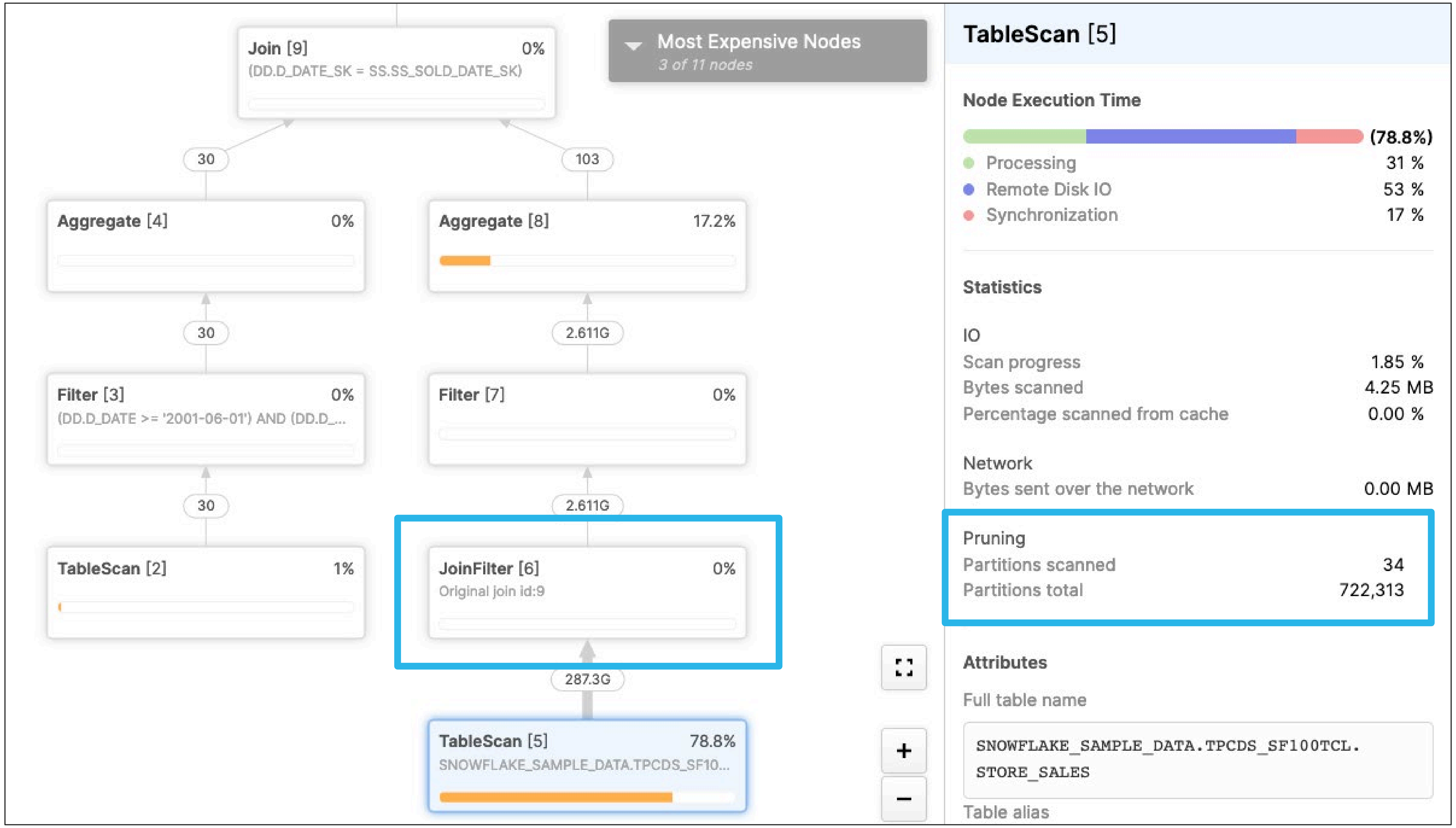
```
EXPLAIN SELECT COUNT(1) AS row_count
FROM date_dim dd
JOIN store_sales ss ON dd.d_date_sk = ss.ss_sold_date_sk
WHERE dd.d_date BETWEEN '2001-06-01' AND '2001-06-30'
GROUP BY dd.d_date;
```

step	id	parent	operation	objects	alias	expressions	partitionsTotal	partitionsAssigned	bytesAssigned
NULL	NULL	NULL	GlobalStats	NULL	NULL	NULL	722314	1841	12017736239616
1	0	NULL	Result	NULL	NULL	COUNT(COUNT_IN...	NULL	NULL	NULL
1	1	0	InnerJoin	NULL	NULL	joinKey: (DD.D_DAT...	NULL	NULL	NULL
1	2	1	Filter	NULL	NULL	(DD.D_DATE >= '20...	NULL	NULL	NULL
1	3	2	TableScan	SNOWFLAKE_S...	DD	D_DATE_SK, D_DATE	1	1	2232832
1	4	1	Filter	NULL	NULL	SS.SS_SOLD_DATE...	NULL	NULL	NULL
1	5	4	JoinFilter	NULL	NULL	joinKey: (DD.D_DAT...	NULL	NULL	NULL
1	6	5	TableScan	SNOWFLAKE_S...	SS	SS_SOLD_DATE_SK	722313	1840	12017734006784

STATIC PRUNING SHOWN IN RESULTS

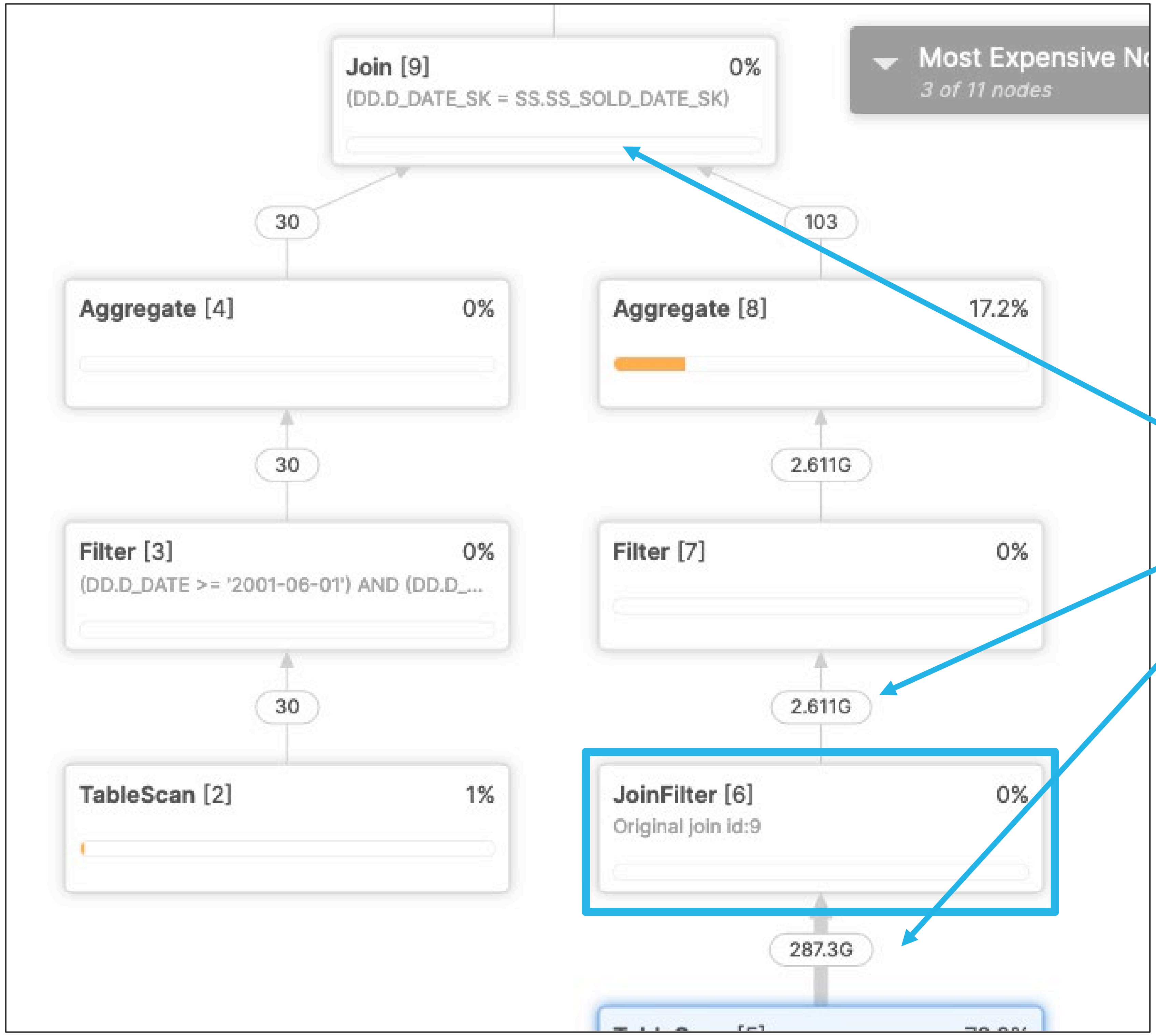
objects	alias	expressions	partitionsTotal	partitionsAssigned	bytesAssigned
NULL	NULL	NULL	722314	1841	12017736239616
NULL	NULL	COUNT(COUNT_IN...	NULL	NULL	NULL
NULL	NULL	joinKey: (DD.D_DAT...	NULL	NULL	NULL
NULL	NULL	(DD.D_DATE >= '20...	NULL	NULL	NULL
SNOWFLAKE_S...	DD	D_DATE_SK, D_DATE	1	1	2232832
NULL	NULL	SS.SS_SOLD_DATE...	NULL	NULL	NULL
NULL	NULL	joinKey: (DD.D_DAT...	NULL	NULL	NULL
SNOWFLAKE_S...	SS	SS_SOLD_DATE_SK	722313	1840	12017734006784

QUERY MAY PRUNE FURTHER AT RUN TIME



- JOIN filter pushed down at run time
- Partitions scanned in EXPLAIN: **1841**
- Partitions actually scanned: **34**

QUERY MAY PRUNE FURTHER AT RUN TIME



- Note the row reduction from 287.3 billion rows to only 2.6 billion rows
- This happens before the JOIN

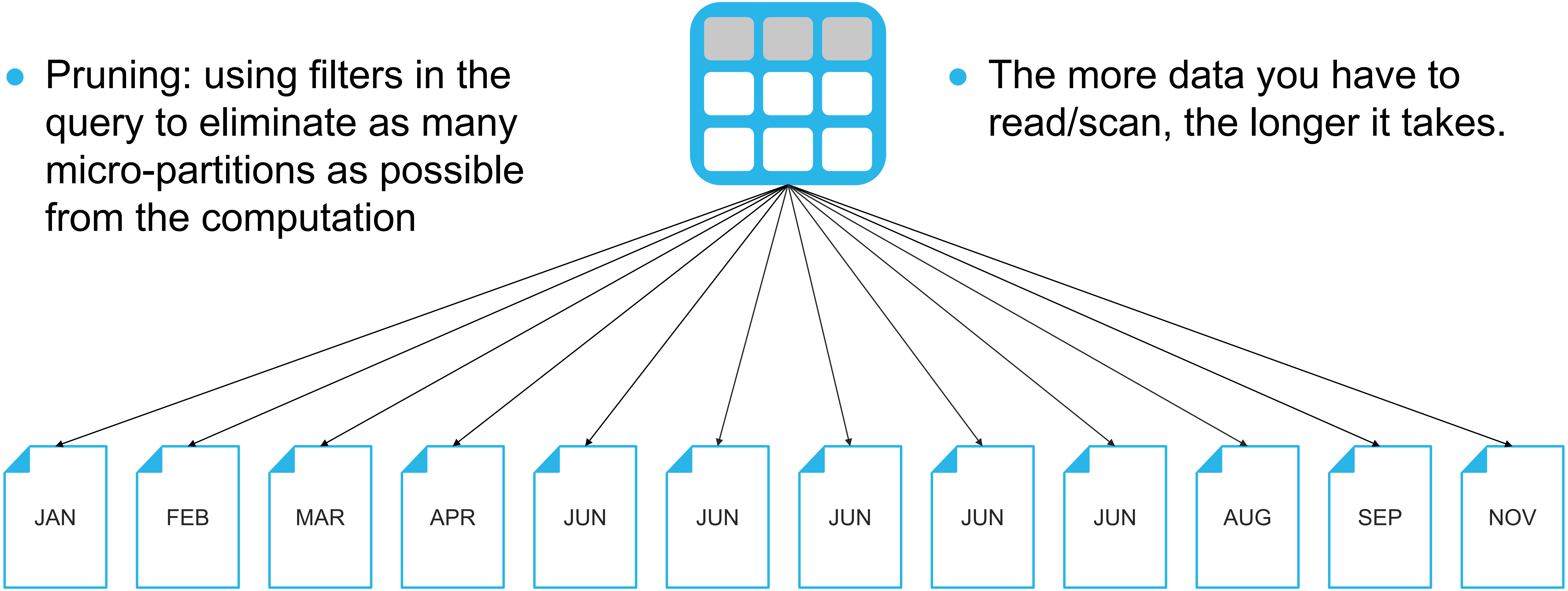
AUTOMATIC CLUSTERING

WHAT IS QUERY PRUNING?

```
SELECT * FROM sales;
```

- Pruning: using filters in the query to eliminate as many micro-partitions as possible from the computation

- The more data you have to read/scan, the longer it takes.

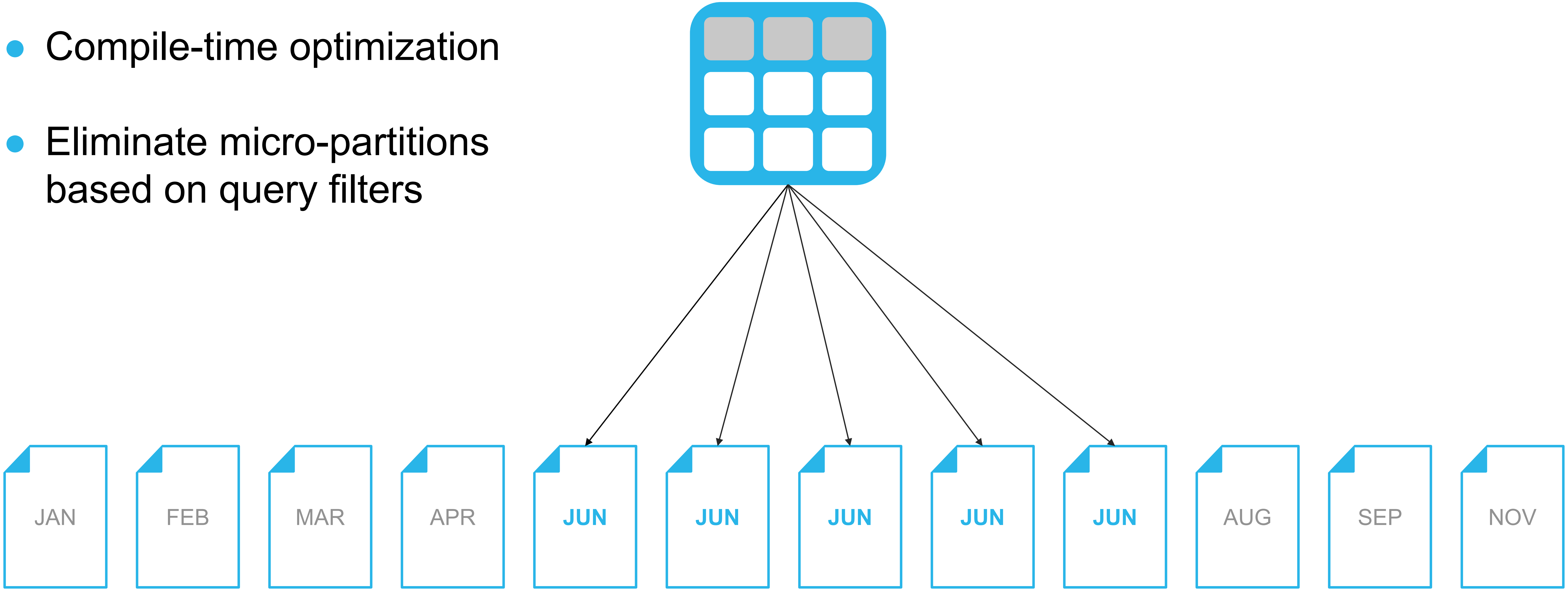


Micro-Partitions

STATIC QUERY PRUNING

```
SELECT * FROM sales WHERE month='June';
```

- Compile-time optimization
- Eliminate micro-partitions based on query filters

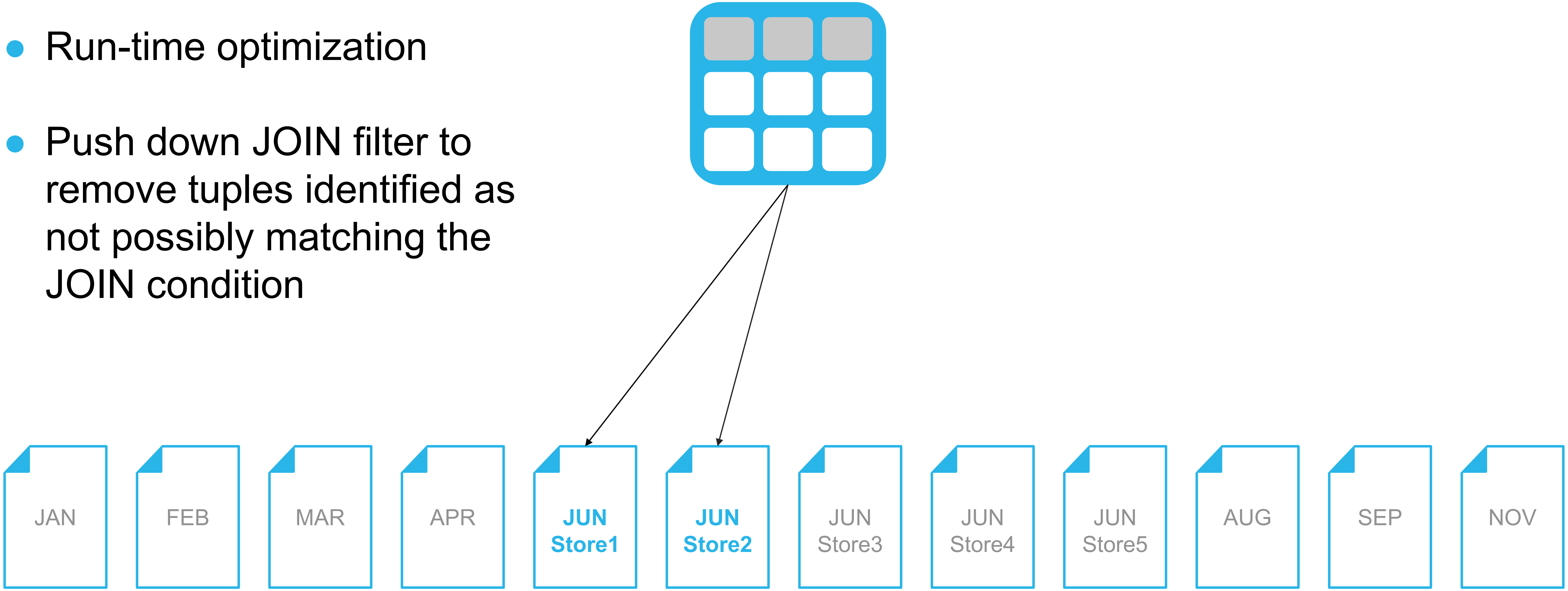


Micro-Partitions

DYNAMIC QUERY PRUNING

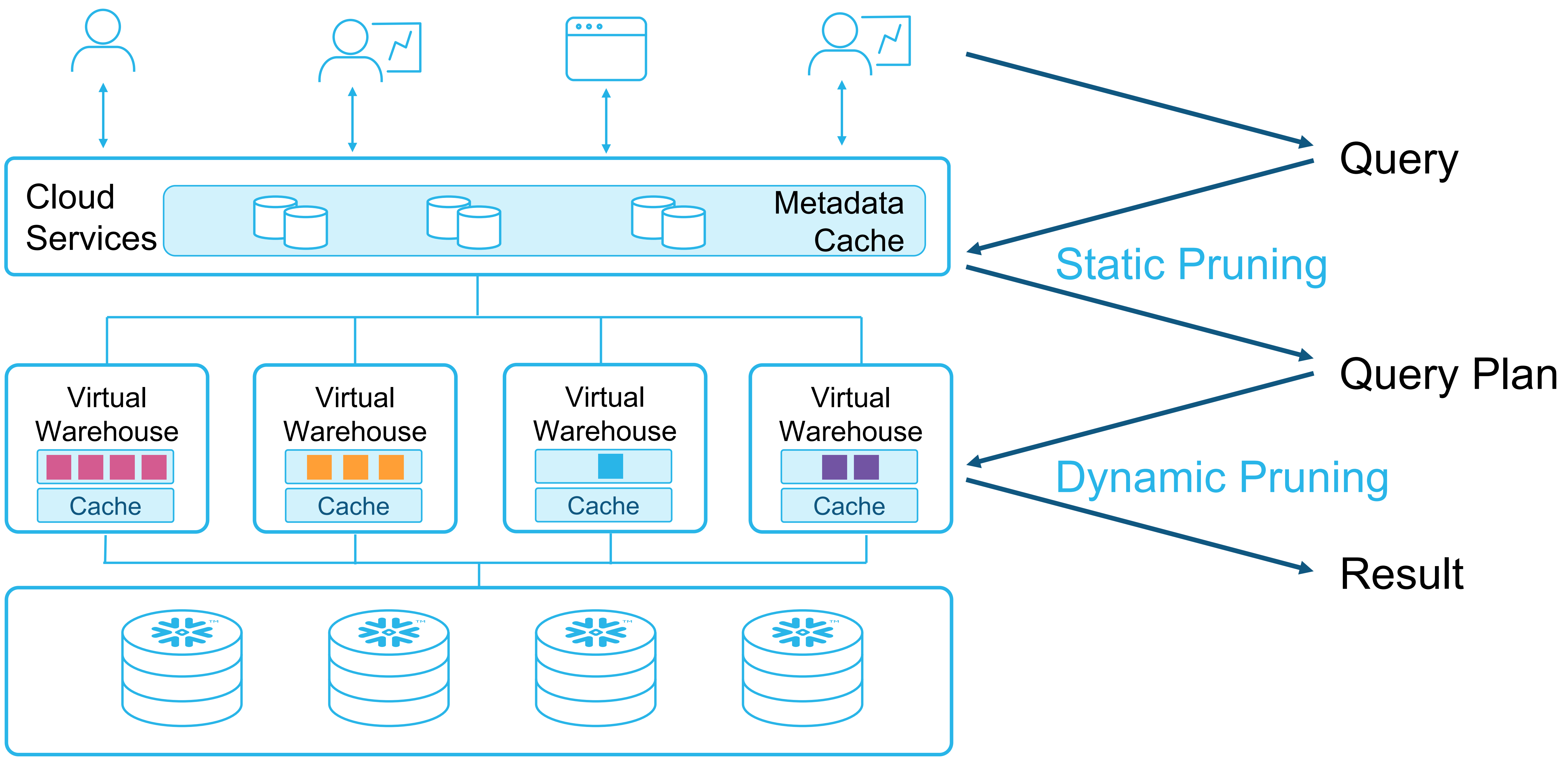
```
SELECT * FROM sales ... JOIN stores ON...WHERE store IN ('Store1', 'Store2');
```

- Run-time optimization
- Push down JOIN filter to remove tuples identified as not possibly matching the JOIN condition



Micro-Partitions

LIFE CYCLE OF A QUERY



SUMMARY

- Micro-partition pruning uses metadata to determine micro-partitions needed for the query
  - Unneeded micro-partitions are pruned out
- Static pruning, based on the `WHERE` clause, happens at compile time
- Dynamic pruning, based on `JOIN` filters (and other constructs), happens at run time
- Use `EXPLAIN` to reveal static pruning, and the query plan to identify dynamic pruning

WHAT IS CLUSTERING?

ORDER_DATE	LAST_NAME
Jan 01, 2021	Williams
Jan 01, 2021	Brooke
Jan 01, 2021	Haddock
Jan 01, 2021	Yellen
Jan 01, 2021	Dubois
Jan 01, 2021	Nguyen
Jan 02, 2021	Jordan
Jan 02, 2021	Yao
Jan 02, 2021	Khatri
Jan 02, 2021	Allen
Jan 02, 2021	Martin

. . .

Jan 10, 2021	Patel
Jan 10, 2021	Hargis
Jan 10, 2021	Brown

- Clustering refers to how well-ordered values are within a column
- In general, dates tend to naturally be in some sort of order (are well-clustered for pruning)
- Columns like last name tend to be more randomized (are poorly clustered for pruning)

WHAT DETERMINES NATURAL CLUSTERING?

- Natural clustering is determined simply by **how the data is organized within the files that are loaded** into Snowflake

- The only logic that Snowflake uses (at load/ingest time) is “Are we able to create the file size that we want?”
    - Data order is not analyzed or changed during load

WIDE RANGE OF VALUES IN SOURCE FILES

POORLY CLUSTERED

File 1	File 2	File 3.. etc...
Records Contain order dates that span	Records Contain order dates that span	Records Contain order dates that span
January - December	January - December	January - December

- Each file contains records with order dates that span January through December
- Result: Must scan every micropartition when querying against order date

NARROW RANGE OF VALUES IN SOURCE FILES

WELL CLUSTERED

File 1	File 2	File 3.. etc...
Records Contain order dates that span	Records Contain order dates that span	Records Contain order dates that span
January	February	March

- Each file contains records for a single month
- Result: Little to no overlap in dates within micropartitions
- Excellent micro-partition pruning when querying by order date

EVALUATE CLUSTERING

SYSTEM$CLUSTERING_INFORMATION

```
SELECT
SYSTEM$CLUSTERING_INFORMATION('table1', '(col1)');
```


```
{
 "cluster_by_keys" : "LINEAR(O_ORDERDATE)",
 "total_partition_count" : 3242,
 "total_constant_partition_count" : 1409,
 "average_overlaps" : 2.5122,
 "average_depth" : 2.4563,
 "partition_depth_histogram" : {
   "00000" : 0,
   "00001" : 1364,
   "00002" : 1362,
   "00003" : 272,
   "00004" : 151,
   "00005" : 89
```


- Results in JSON format
- **`average_depth`**: lower numbers indicate better clustering
- **`total_constant_partition_count`**: higher numbers indicate better clustering

PARTITION DEPTH HISTOGRAMS


```
"partition_depth_histogram" : {
 "00000" : 0,
 "00001" : 0,
 "00002" : 0,
 "00003" : 0,
 "00004" : 0,
 "00005" : 0,
 "00006" : 0,
 "00007" : 0,
 "00008" : 0,
 "00009" : 0,
 "00010" : 0,
 "00011" : 0,
 "00012" : 0,
 "00013" : 0,
 "00014" : 0,
 "00015" : 0,
 "00016" : 0,
 "01024" : 1022
```


Poorly clustered for filters on tested column


```
"partition_depth_histogram" : {
 "00000" : 0,
 "00001" : 1364,
 "00002" : 1362,
 "00003" : 272,
 "00004" : 151,
 "00005" : 89,
 "00006" : 4,
 "00007" : 0,
 "00008" : 0,
 "00009" : 0,
 "00010" : 0,
 "00011" : 0,
 "00012" : 0,
 "00013" : 0,
 "00014" : 0,
 "00015" : 0,
 "00016" : 0
```


Well clustered for filters on tested column

CLUSTERING METRICS - WIDTH

- Width of a micropartition for a specific column (MAX – MIN)

Clustering key = **AGE**

Micropartition 1: Wide on age column

Sam	18	USA
Trevor	78	Canada

CLUSTERING METRICS - WIDTH

- Width of a micropartition for a specific column (MAX – MIN)

Clustering key = **AGE**

Micropartition 1: Wide on age column

Sam	18	USA
Trevor	78	Canada

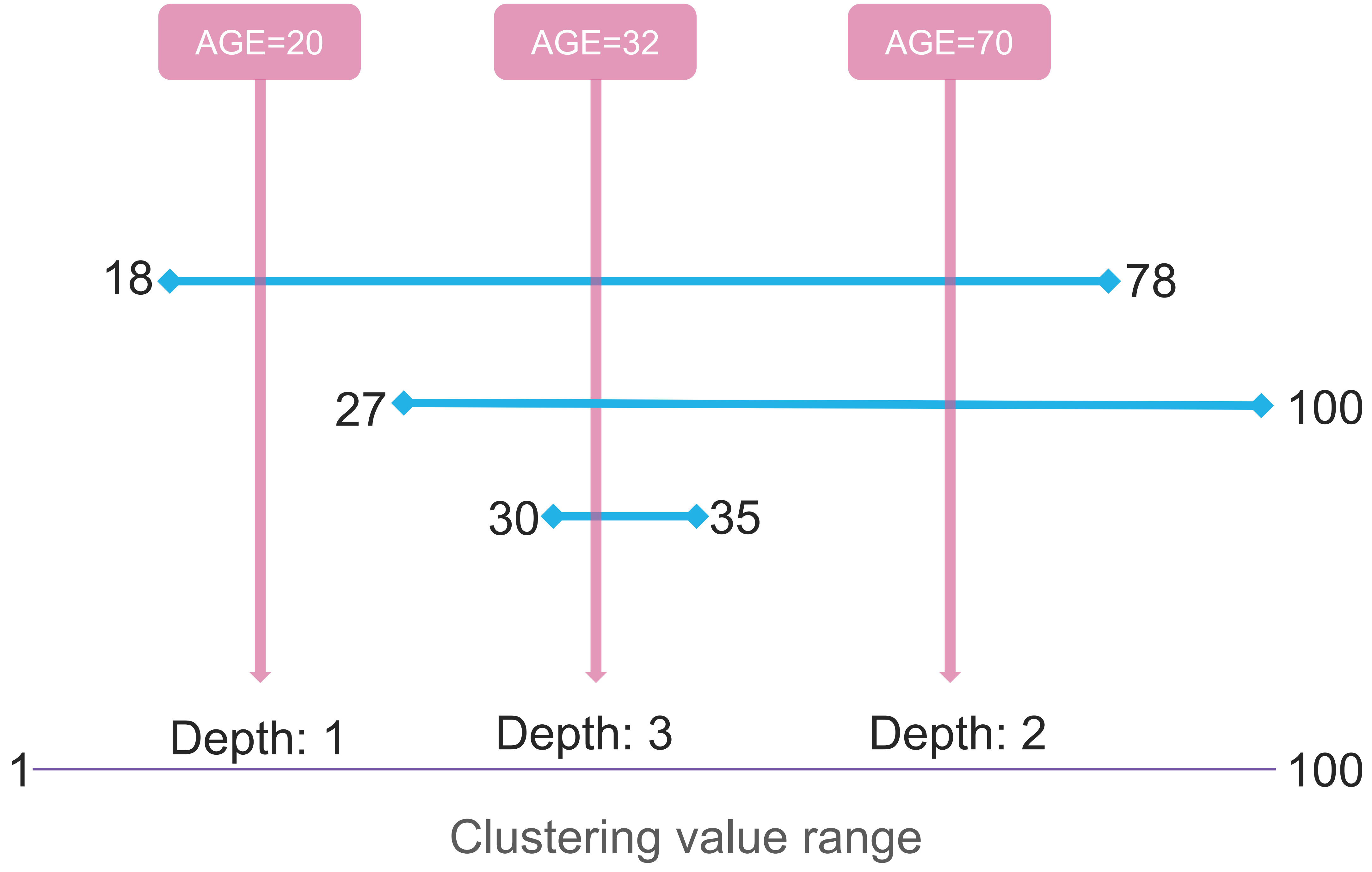
Micropartition 2: Narrow on age column

Anna	30	England
Raj	35	India

CLUSTERING METRICS - DEPTH

- Number of micropartitions overlapping at a certain value in the clustering range

- Average_depth: on average, how many micropartitions would need to be searched for a given value?



WHAT IS A CLUSTERING KEY?

- An explicit declaration of columns in a table to sort the data by
- Useful for very large tables where the natural ordering is not ideal, or extensive DML has caused the table’s natural clustering to degrade
- Can be defined at table creation or afterward
- Maintained by Snowflake's automatic clustering service
- Can be altered or dropped at any time

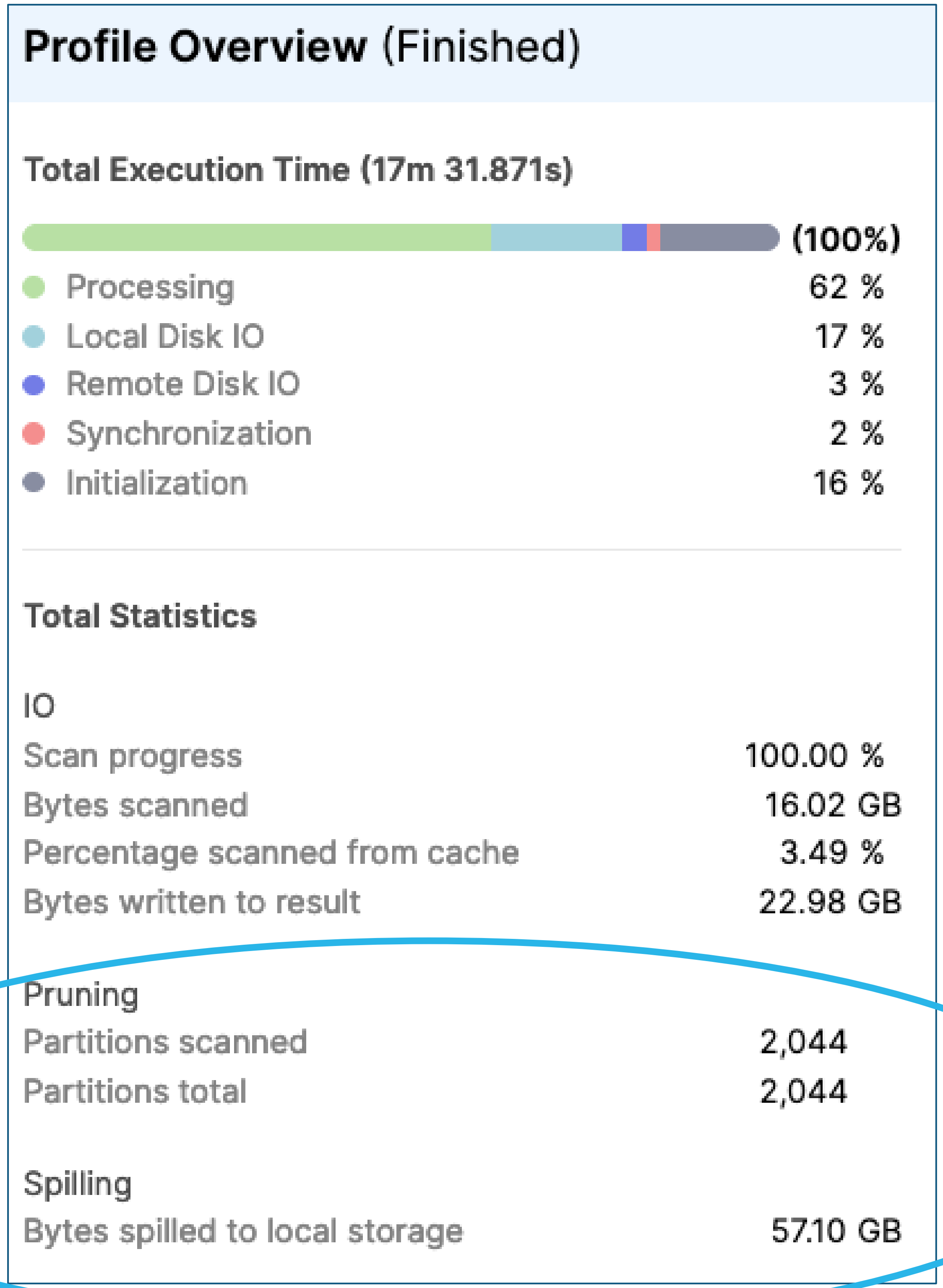
CLUSTERING COMMAND SAMPLES

```
CREATE TABLE t1 (c1 date, c2 string, c3 number)
CLUSTER BY (c1, c2);

CREATE TABLE t2 (c1 timestamp, c2 string, c3 number)
CLUSTER BY (TO_DATE(c1), SUBSTRING(c2, 3, 3));

ALTER TABLE t1
CLUSTER BY (c1, c3);

ALTER TABLE t2
CLUSTER BY (SUBSTRING(c2, 5, 2), TO_DATE(c1));
```

METHODOLOGY FOR EXPLICIT CLUSTERING

1. Identify appropriate performance use case and query access patterns

   - Long-running queries against very large (> 1TB) tables
   - Columns frequently used in WHERE clauses
   - JOIN columns
   - GROUP BY columns

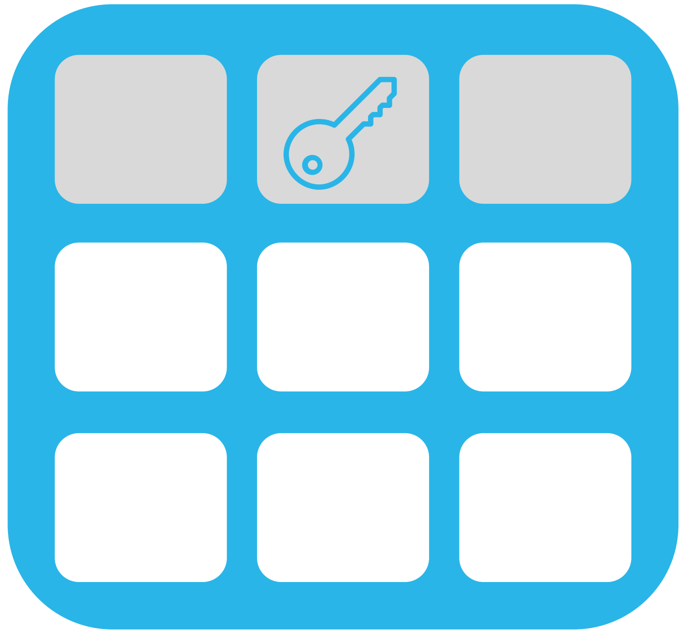


METHODOLOGY FOR EXPLICIT CLUSTERING

1. Identify appropriate performance use case and query access patterns
2. Run use case workload and gather baseline metrics

METHODOLOGY FOR EXPLICIT CLUSTERING

1. Identify appropriate performance use case and query access patterns
2. Run use case workload and gather baseline metrics
3. Assign clustering key(s)



```
ALTER TABLE my_table
CLUSTER BY (col2);
```

CLUSTERING AND CARDINALITY

- More keys are not better (though you can use as many as you want)
  - Try to stick with 1 or 2
- If you are defining a multi-column clustering key for a table, you should generally order columns from lowest cardinality (least number of unique values) to highest cardinality
- Use expressions to lower cardinality of leading key - for example, `DATE_TRUNC()`
- Putting a higher cardinality column before a lower cardinality column will reduce the effectiveness of clustering on the second column

CLUSTERING AND CARDINALITY EXAMPLE

Clustered by

(STATUS, STORE_NO)

STATUS	STORE_NO	CUST_NUM	TOTAL
APPROVED	17	01236	2245.87
APPROVED	17	11835	123.95
APPROVED	22	44390	225.87
APPROVED	22	33210	1.87
APPROVED	22	00236	2245.98
APPROVED	25	44039	1328.99
APPROVED	26	93012	145.33
APPROVED	26	01236	358.33
COMPLETE	17	44210	24.97
COMPLETE	21	55439	3.98
COMPLETE	22	22409	987.54
COMPLETE	22	42239	493.22
COMPLETE	22	64452	87.33
COMPLETE	25	99325	23.87
COMPLETE	26	01266	449.20
COMPLETE	26	32009	53.28
COMPLETE	27	52393	182.45
COMPLETE	27	43992	3356.87
COMPLETE	27	88423	192.45
COMPLETE	30	62345	312.87

Clustered by

(STORE_NO, STATUS)

STATUS	STORE_NO	CUST_NUM	TOTAL
APPROVED	17	01236	2245.87
APPROVED	17	11835	123.95
COMPLETE	17	44210	24.97
COMPLETE	21	55439	3.98
APPROVED	22	44390	225.87
APPROVED	22	33210	1.87
APPROVED	22	00236	2245.98
COMPLETE	22	22409	987.54
COMPLETE	22	42239	493.22
COMPLETE	22	64452	87.33
APPROVED	25	44039	1328.99
COMPLETE	25	99325	23.87
APPROVED	26	93012	145.33
APPROVED	26	01236	358.33
COMPLETE	26	01266	449.20
COMPLETE	26	32009	53.28
COMPLETE	27	52393	182.45
COMPLETE	27	43992	3356.87
COMPLETE	27	88423	192.45
COMPLETE	30	62345	312.87

METHODOLOGY FOR EXPLICIT CLUSTERING

1. Identify appropriate performance use case and query access patterns
2. Run use case workload and gather baseline metrics
3. Assign clustering key(s)
4. Allow clustering to complete

```
SELECT SYSTEM$CLUSTERING_INFORMATION('table1', '(col1)');
```


```
{
  "cluster_by_keys" : "LINEAR(O_ORDERDATE)",
  "total_partition_count" : 3242,
  "total_constant_partition_count" : 1409,
  "average_overlaps" : 2.5122,
  "average_depth" : 2.4563,
  "partition_depth_histogram" : {
```


When average_depth stops changing, clustering is "done"

METHODOLOGY FOR EXPLICIT CLUSTERING

1. Identify appropriate performance use case and query access patterns
2. Run use case workload and gather baseline metrics
3. Assign clustering key(s)
4. Allow clustering to complete
5. Measure benefits
    - Compare before-and-after performance

METHODOLOGY FOR EXPLICIT CLUSTERING

1. Identify appropriate performance use case and query access patterns
2. Run use case workload and gather baseline metrics
3. Assign clustering key(s)
4. Allow clustering to complete
5. Measure benefits
6. Monitor auto-clustering service and cost

```
SELECT * FROM
TABLE(information_schema.automatic_clustering_history());
```

METHODOLOGY FOR EXPLICIT CLUSTERING

1. Identify appropriate performance use case and query access patterns
2. Run use case workload and gather baseline metrics
3. Assign clustering key(s)
4. Allow clustering to complete
5. Measure benefits
6. Monitor auto-clustering service and cost
7. Evaluate DML impacts on maintenance of clustering layout

CLUSTERING COST

Monitoring

- Aggregate the *credits_used* column from the `INFORMATION_SCHEMA` table function across the desired time range
- Available in UI as a separate warehouse item

Key Factors to Credit Usage

- Number and cardinality of key(s)
- Number of micro-partitions involved in reclustering
- Size of table
- Impact from DML (frequency and pattern)

WHEN TO DEFINE CLUSTER KEYS

- Clustering should not be the first attempt to improve performance
- Good candidates should have some (preferably all) of the following characteristics:
    - The table is large (multiple TBs)
    - Large % of overall query time is spent scanning tables
    - Most of the time is spent scanning one table
    - Pruning ratio is low
    - You usually query by specific columns

AUTOMATIC CLUSTERING SERVICE

CONTINUOUS SORTING
AT PETABYTE SCALE

AUTOMATIC CLUSTERING SERVICE

Approximate
^
CONTINUOUS SORTING
AT PETABYTE SCALE

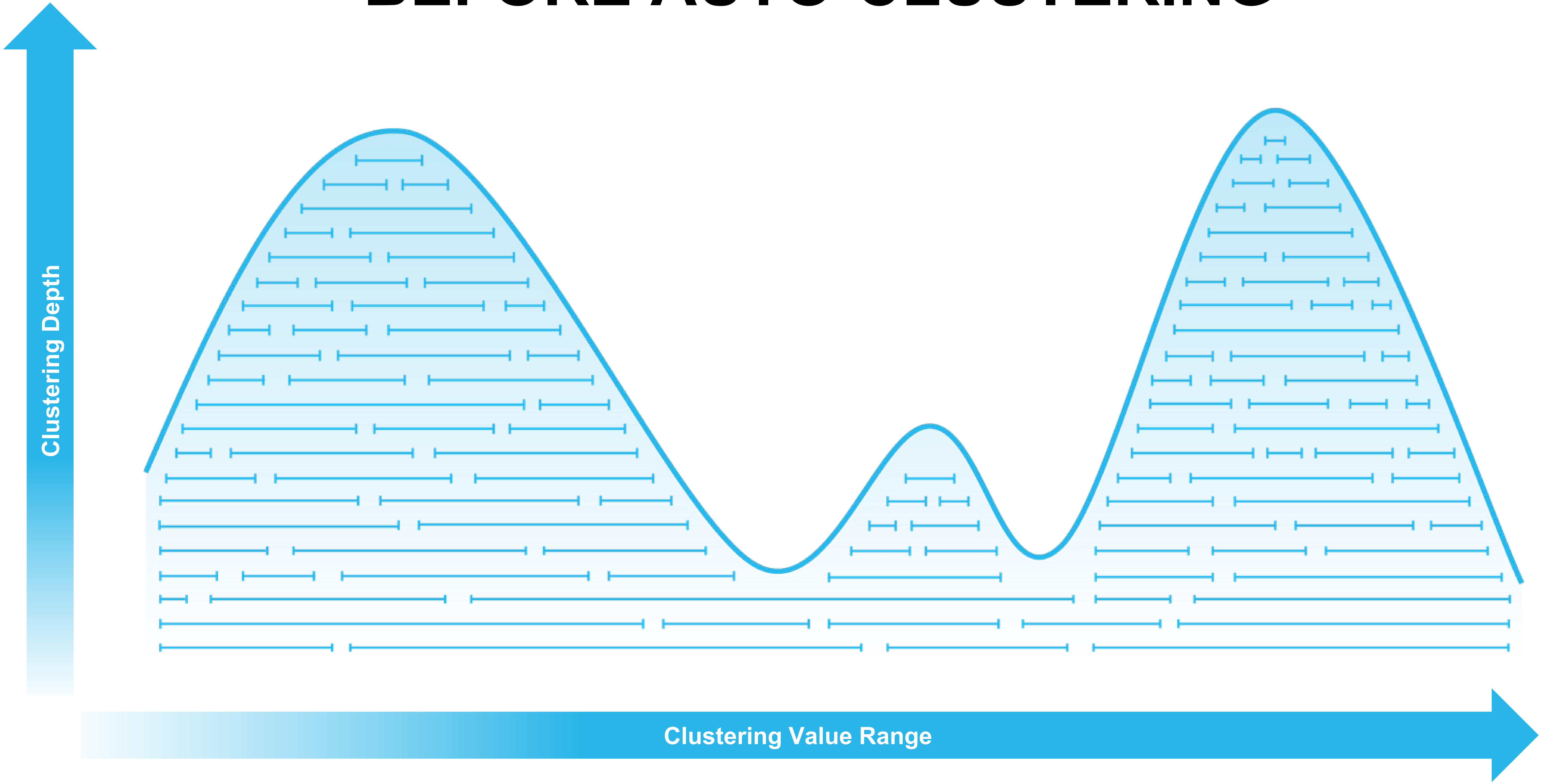
ALGORITHM GOAL

Reduce **Worst** Clustering Depth

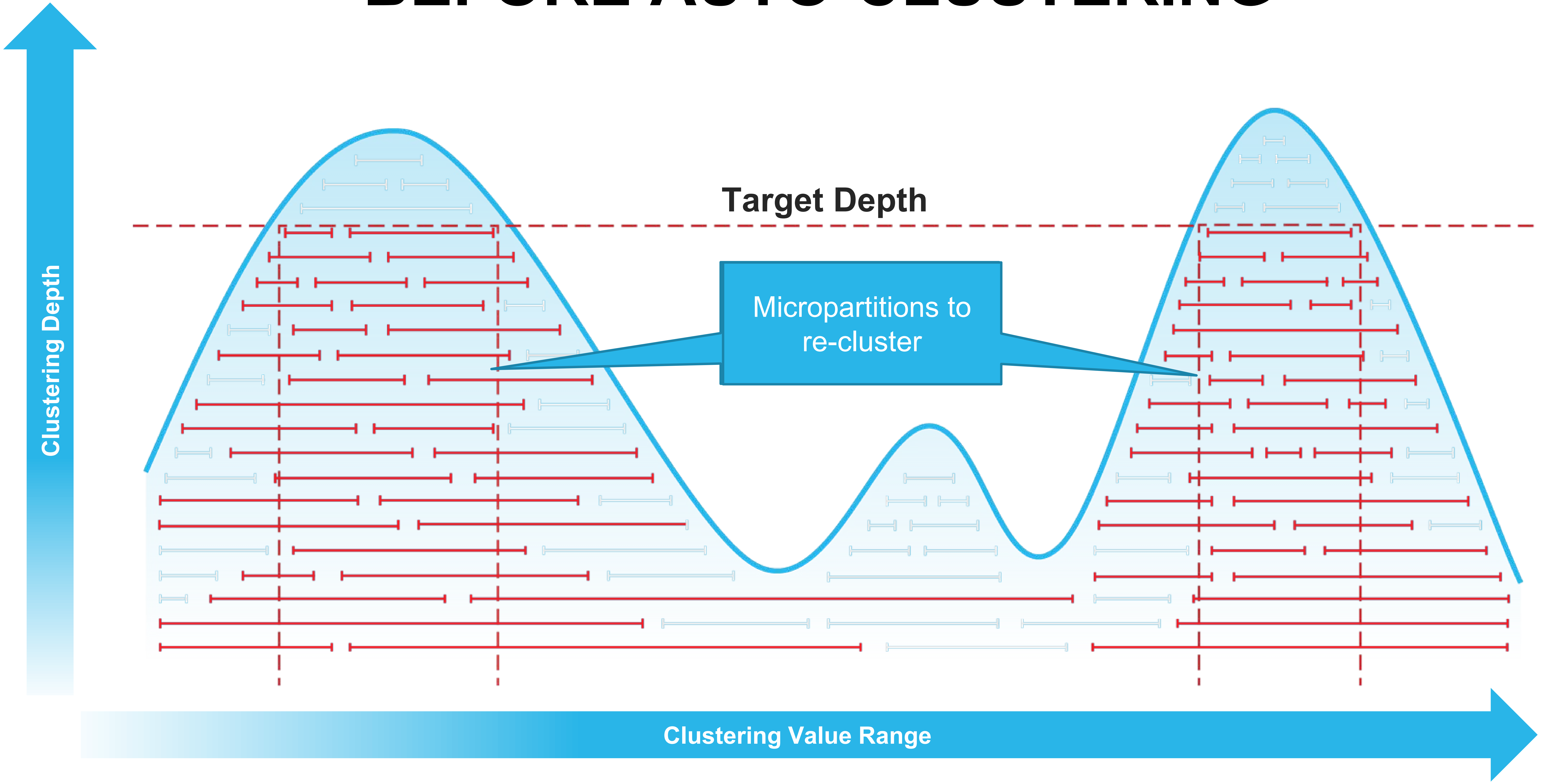
below an acceptable threshold to get

Predictable Query Performance

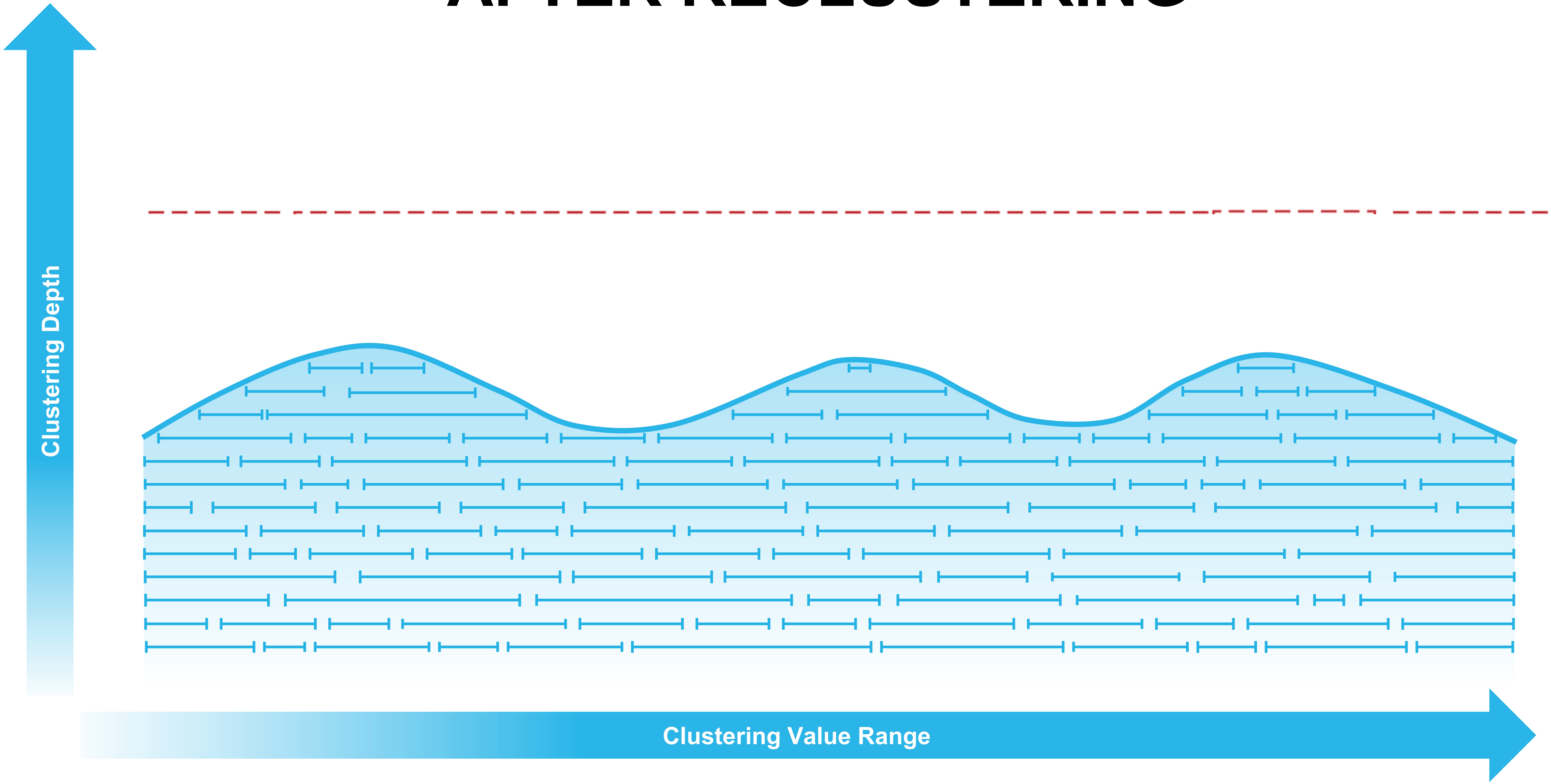
BEFORE AUTO-CLUSTERING



BEFORE AUTO-CLUSTERING



AFTER RECLUSTERING



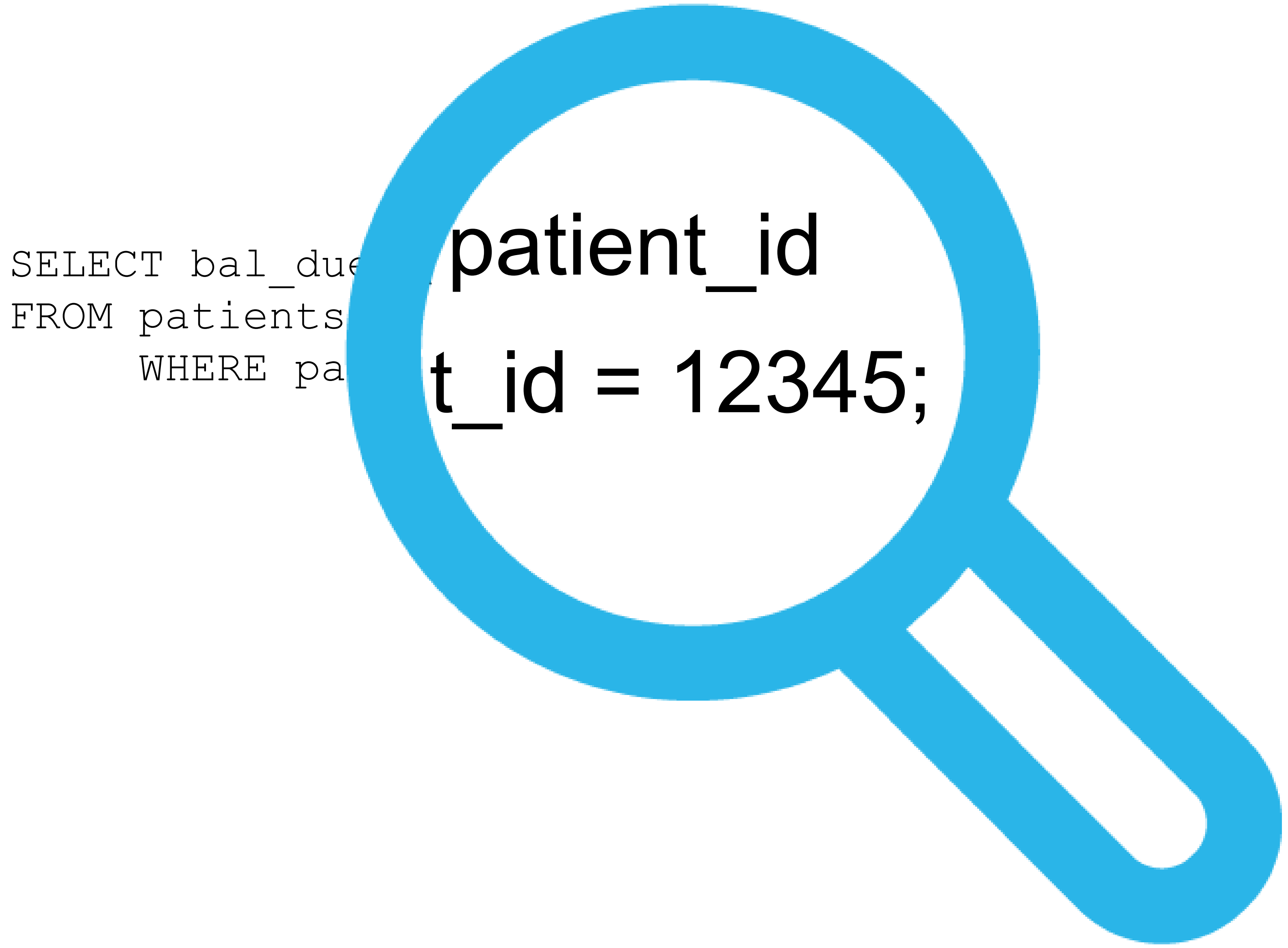
AUTOMATIC CLUSTERING SERVICE

- Serverless feature compute - no virtual warehouse needed
  - Billing in credits based on actual compute usage, measured to the second
- Snowflake maintains clustering of tables automatically
- Non-blocking to DML
- User can suspend / resume clustering service on each table

SEARCH OPTIMIZATION

OVERVIEW

WHEN TO USE SEARCH OPTIMIZATION



- Equality searches
  - Also known as point query or fast lookup query
  - Returns only a small % of rows
- Tune tables with frequent selective queries using Search Optimization

SELECTIVE QUERY EXAMPLE

- Looking for a single value (or small number) in a large table
- Put a Search Optimization on the target table to increase performance

Billions of rows in store_sales table

```
-- Find the sales ticket, items sold and sale date
-- for one customer

SELECT ss_ticket_number, ss_sold_date_sk, ss_item_sk
FROM store_sales
WHERE
ss_customer_sk = 3229284;
```

Only a small number of rows meet the criteria for result

CREATE A SEARCH PATH

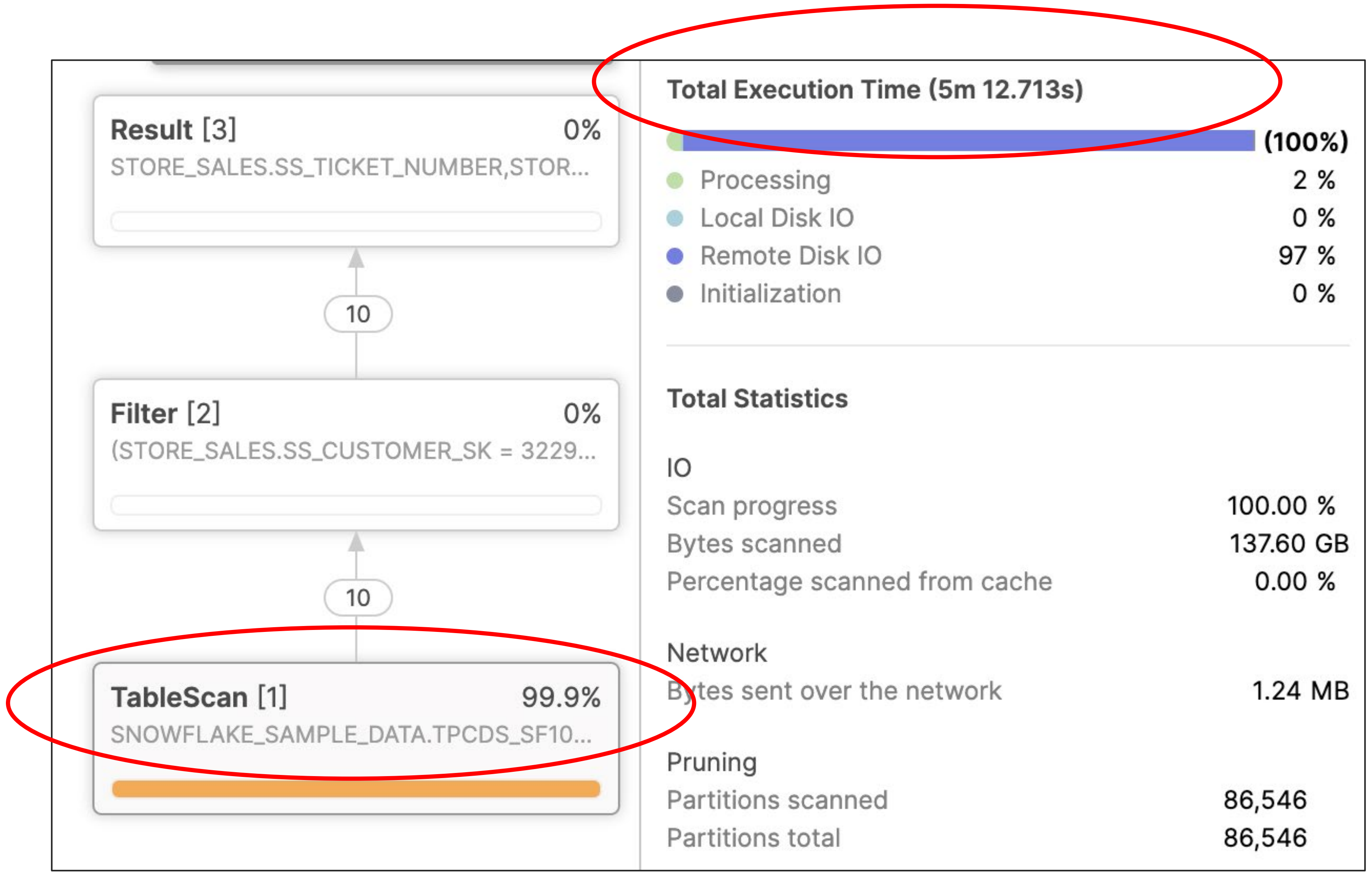
- Add table to the Search Optimization service:

```
ALTER TABLE store_sales
   ADD SEARCH OPTIMIZATION;
```

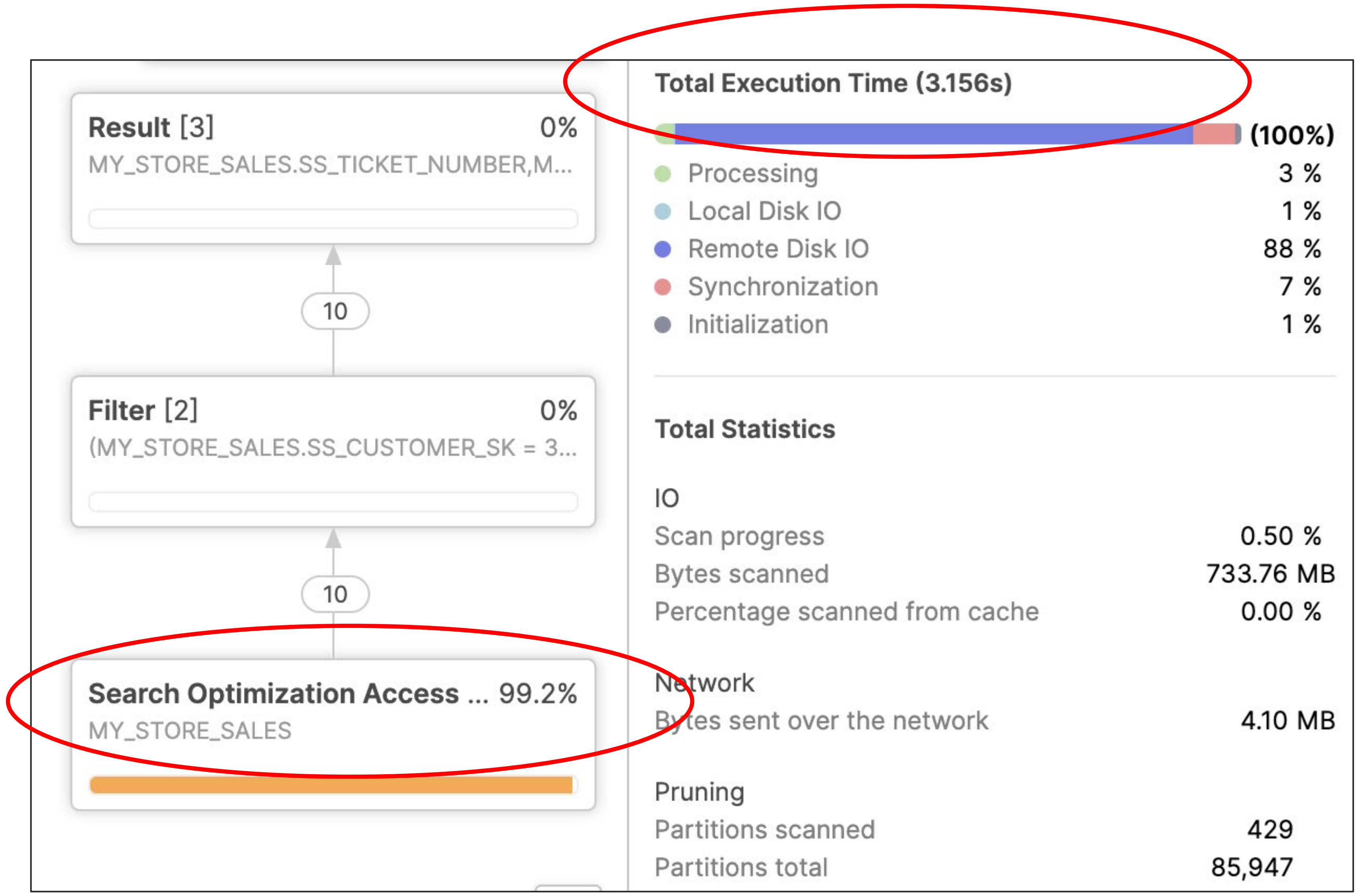
- Search service builds the search access path to optimize point queries on table
  - Will take some time to build the optimization

PERFORMANCE BENEFIT

Without search optimization

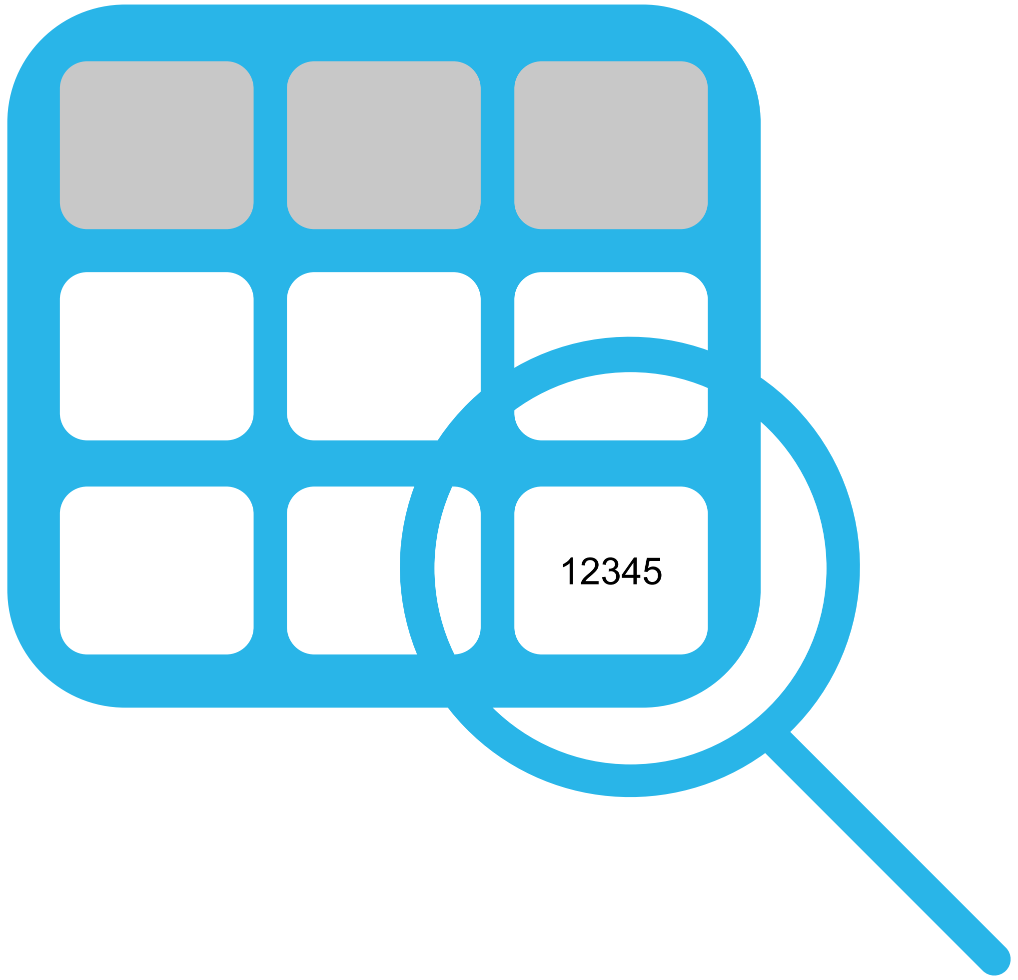


With search optimization



WHAT IS A SEARCH OPTIMIZATION?

- A data structure that stores the optimized search access path
  - One per table
- Contains **all** columns of a permanent table
- Maintained in a separate structure (storage)
- Maintained as changes are made to the base table



COST CONSIDERATIONS

- The larger the number of high cardinality columns, the higher the cost
- Tables with frequent mutations (from DML operations) will also result in higher cost
- Estimate search optimization costs with:

```
SYSTEM$ESTIMATE_SEARCH_OPTIMIZATION_COSTS('<table name>')
```

- Estimate requires 7 days of history

MONITORING SEARCH OPTIMIZATION SERVICE

- `SEARCH_OPTIMIZATION_HISTORY`
  - `ACCOUNT_USAGE` view in `SNOWFLAKE` database
  - `INFORMATION_SCHEMA` table function
- `SHOW TABLES`
  - `SEARCH_OPTIMIZATION` column exposes additional storage use for storage optimization

Column Name	Data Type
START_TIME	TIMESTAMP_LTZ
END_TIME	TIMESTAMP_LTZ
CREDITS_USED	TEXT
TABLE_ID	NUMBER
TABLE_NAME	TEXT
SCHEMA_ID	NUMBER
SCHEMA_NAME	TEXT
DATABASE_ID	NUMBER
DATABASE_NAME	TEXT

COMPARISON

Feature	Best Use Case
Clustering Keys	• Multi-terabyte table • Performing range lookups on a subset of columns • Large percentage of query time is spent in table scans • Cardinality of selected columns is proportional to the number of micro-partitions
Materialized View	• Queries against base table often include aggregations • Base table data is semi-structured but consumers expect structured data • Different groups query the base table in different ways
Search Optimization	• Equality lookups or small ranges • Searches may be on any column

LAB EXERCISE: 11

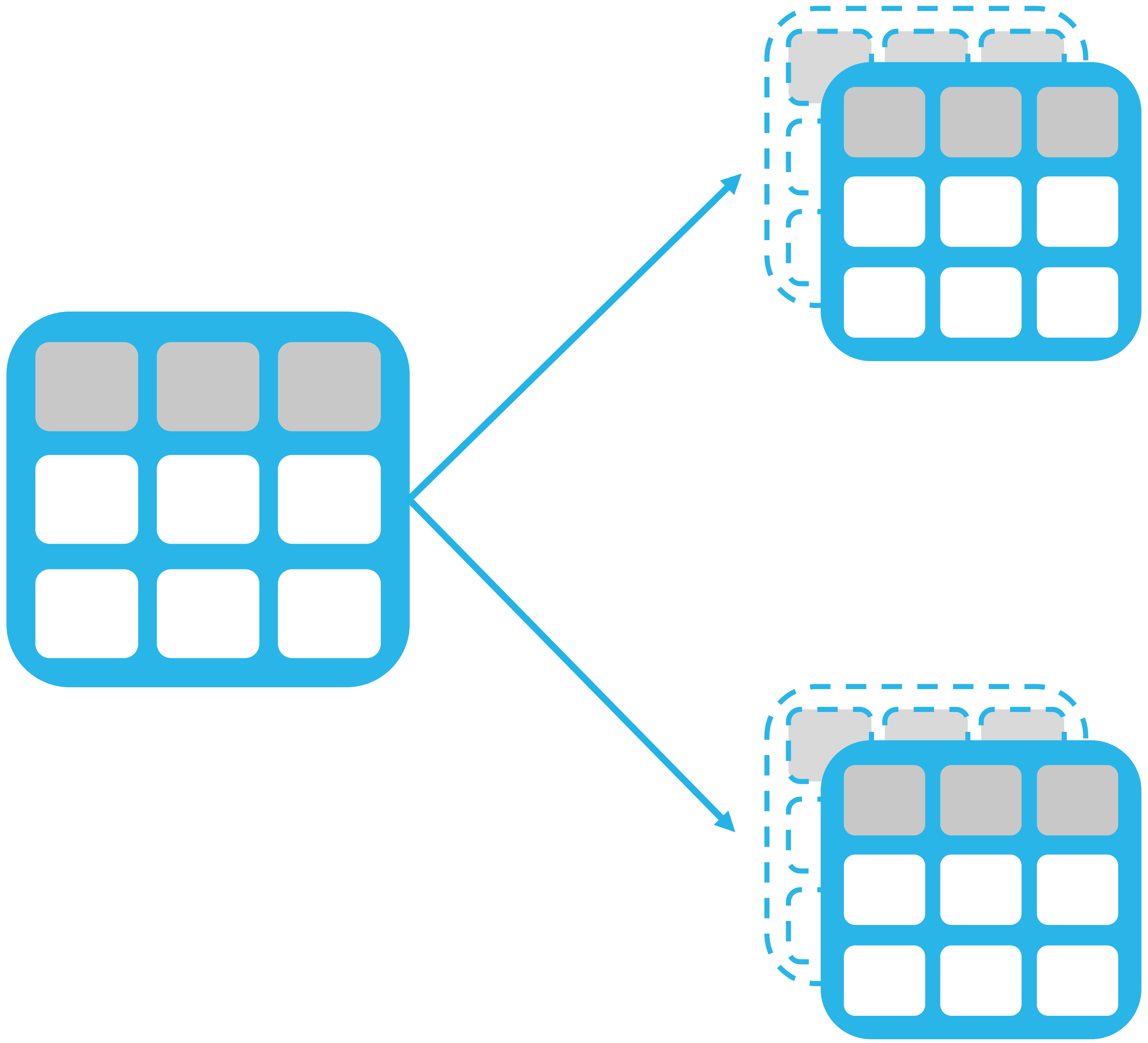
Query and Search Optimization

40 minutes

- Explore Query Performance
- Explore GROUP BY and ORDER BY Operation Performance
- Querying with LIMIT Clause
- JOIN Optimizations in Snowflake
- Enable Search Optimization Service
- Identify a Point Query
- Search Optimization Performance
- Explore Cost of Search Optimization

MATERIALIZED VIEWS

MATERIALIZED VIEWS



- Pre-computed data set derived from a query specification
- Querying a materialized view is faster than executing the query
- Reduce repetitive intensive computation
- Grant access control to materialized views like other database objects

MOTIVATING SCENARIOS

- Stored aggregates or CPU-intensive subqueries
- Different projections for different users
- Improve query performance with external tables
- Sharing data with multiple consumers
- Semi-Structured data analysis

SCENARIO 1

DIFFERENT PROJECTIONS FOR DIFFERENT USERS

CHALLENGE

- Multiple query workloads with different access paths and filters on the same table (or set of tables)
  - Different WHERE clause columns
  - Different JOIN keys
- How do you cluster the tables to maximize micropartition pruning?

SCENARIO 1

DIFFERENT PROJECTIONS FOR DIFFERENT USERS

CHALLENGE

- Multiple query workloads with different access paths and filters on the same table (or set of tables)
  - Different WHERE clause columns
  - Different JOIN keys
- How do you cluster the tables to maximize micropartition pruning?

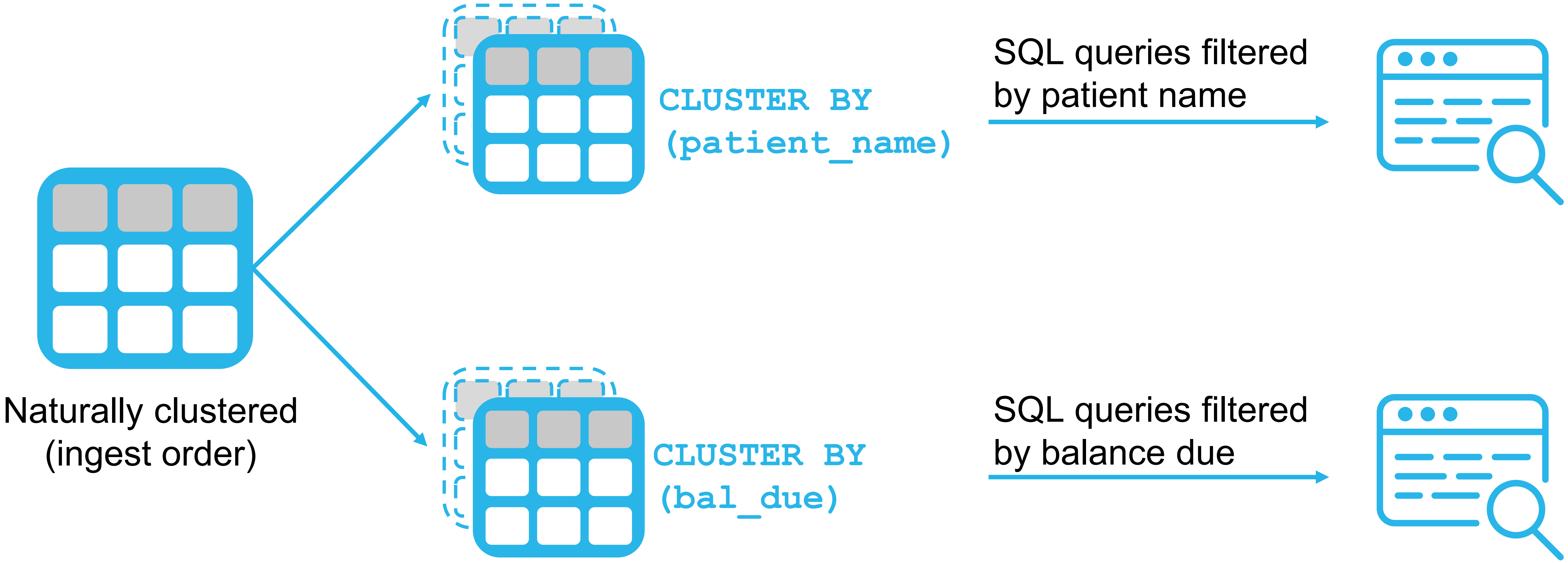
SOLUTION

Create multiple materialized views, with each view having a different clustering key defined

```
CREATE MATERIALIZED VIEW mv1…
   CLUSTER BY (patient_name)
   AS SELECT…

CREATE MATERIALIZE VIEW mv2…
   CLUSTER BY (bal_due)
   AS SELECT…
```

SCENARIO 1

DIFFERENT PROJECTIONS FOR DIFFERENT USERS



SCENARIO 2

SEMI-STRUCTURED DATA ANALYSIS

CHALLENGE:

- JSON does not provide data types for temporal data (dates and times) – how do you use these fields as filters and still get adequate performance?

SCENARIO 2

SEMI-STRUCTURED DATA ANALYSIS

CHALLENGE:

- JSON does not provide data types for temporal data (dates and times) – how do you use these fields as filters and still get adequate performance?

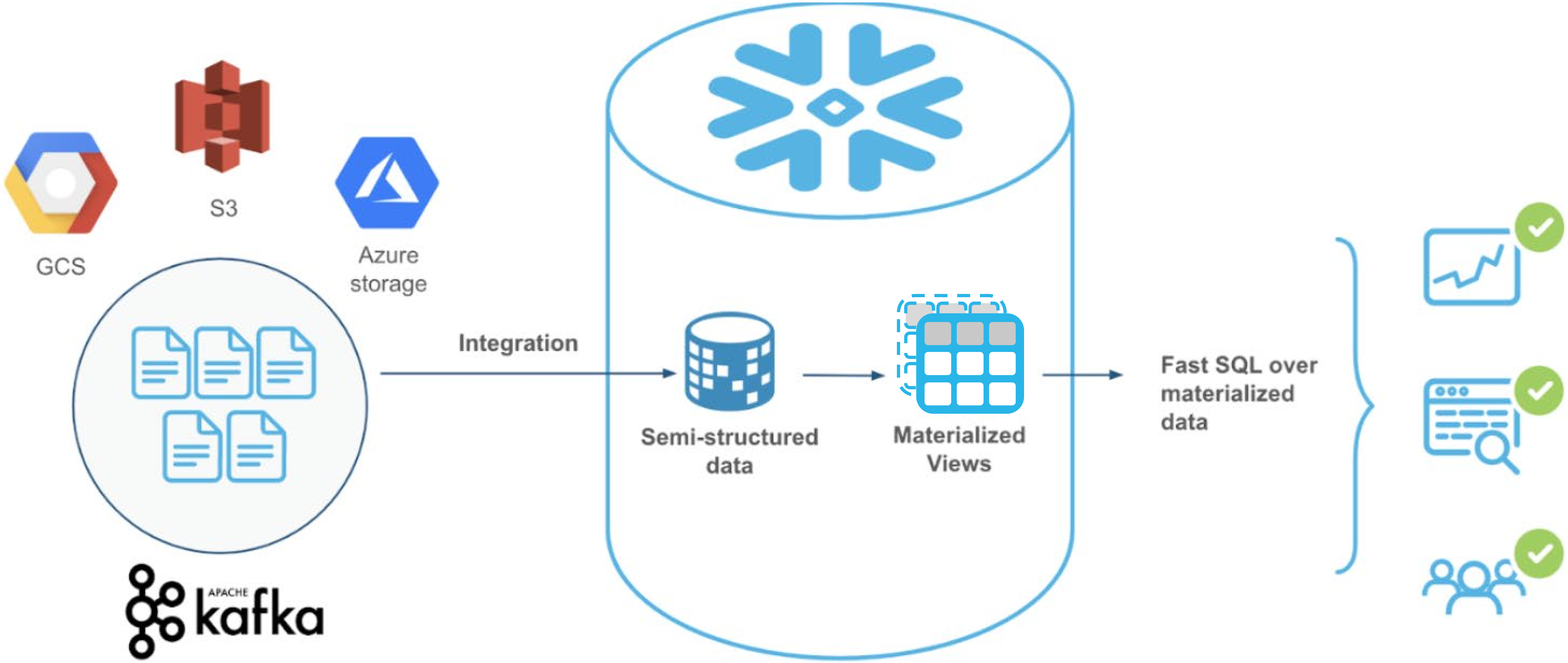
SOLUTION:

- Create a materialized view with the temporal data extracted into distinct columns, and cluster on those columns

```
CREATE MATERIALIZED VIEW mv1(…)
CLUSTER BY(TO_DATE(GET_PATH(DATA, 'raw_json.timestamp.instant_sec’)));
```

SCENARIO 2

SEMI-STRUCTURED DATA ANALYSIS



MATERIALIZED VIEW VS DIRECT QUERY

MATERIALIZED VIEW

Advantages

- Easier access from 3rd party tools
- Better pruning performance
- Direct access via column names (simpler syntax)

Disadvantages

- Accessing new elements requires ETL change (INSERT change)

DIRECT QUERY

Advantages

- Flexible access
- No duplicate storage
- Changes to structure involve only changing queries or views

Disadvantages

- Elements that should be typed (for example, TIMESTAMP) take up more space as STRINGs than they would as their native types

SCENARIO 3

QUERY EXTERNAL TABLES

CHALLENGE

- You have an external data lake where the files periodically change
- How do you get good query performance on changing external data?

SCENARIO 3

QUERY EXTERNAL TABLES

CHALLENGE

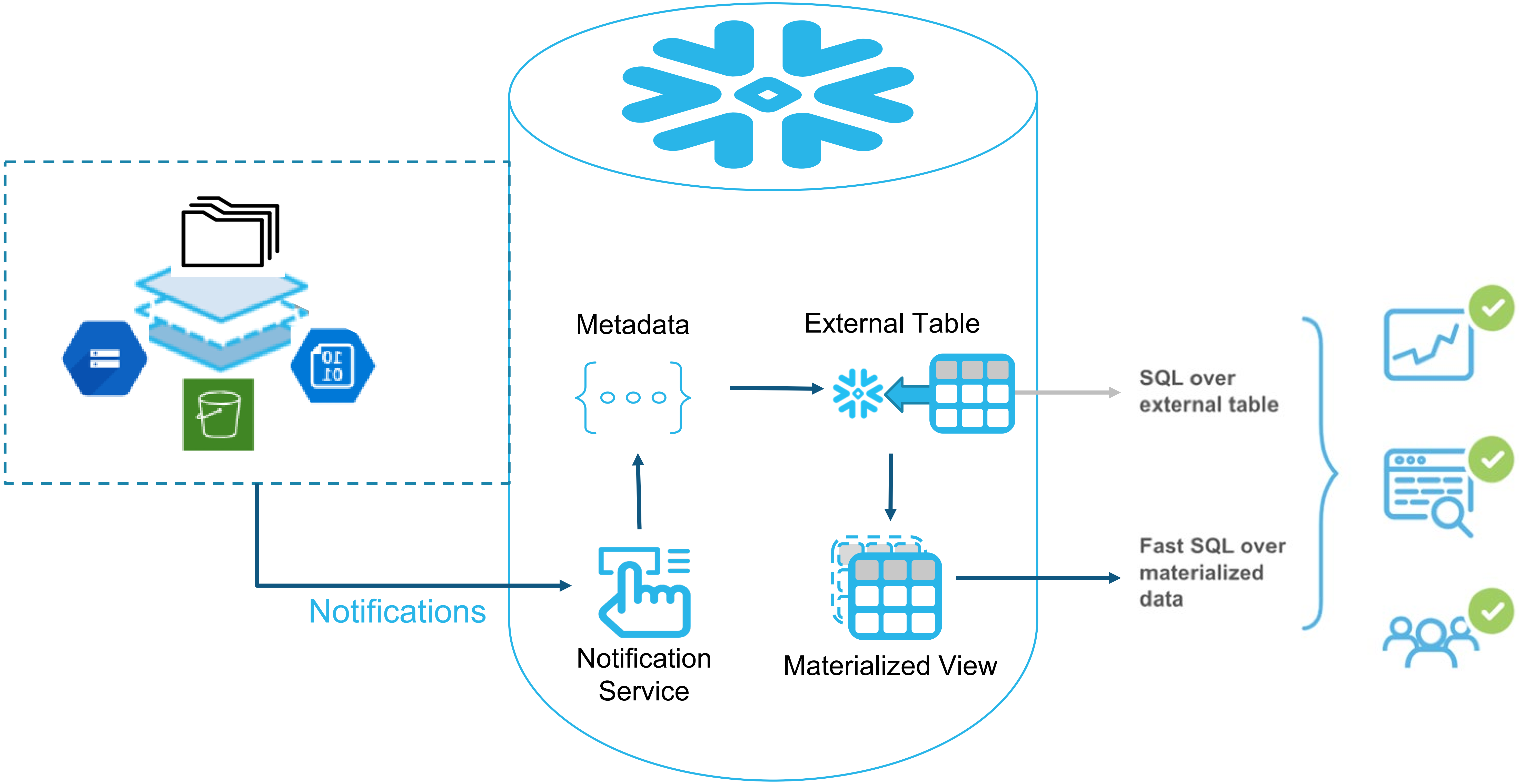
- You have an external data lake where the files periodically change
- How do you get good query performance on changing external data?

SOLUTION

- Create external tables to point to the external data lake
  - External table metadata updated as files change
- Create a materialized view on the external table
  - Materialized views updated as files in the data lake are modified or added

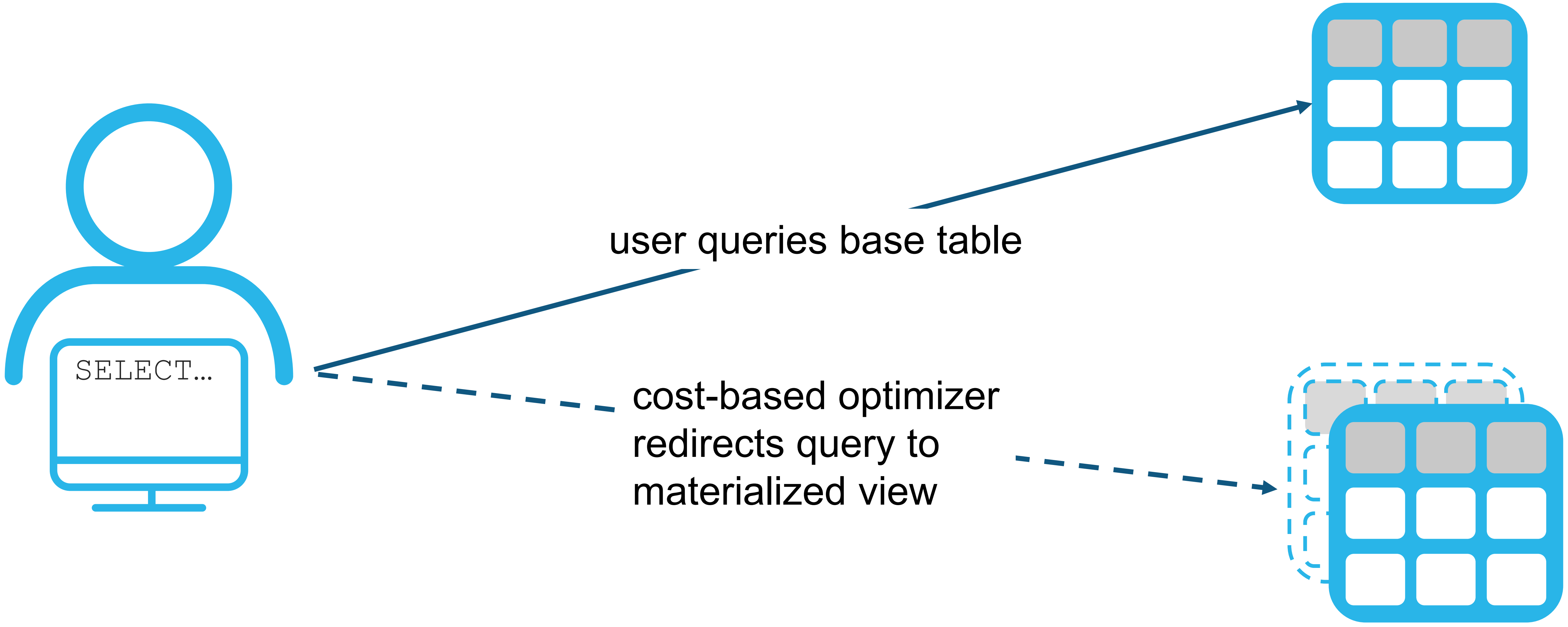
SCENARIO 3

QUERY EXTERNAL TABLES



AUTOMATIC QUERY REWRITES

- Users no longer need to know a materialized view exists, to take advantage of it



MAINTENANCE COSTS

- Materialized views are maintained automatically as a background process
- Each materialized view stores query results, which adds to the monthly storage usage of your account
- Serverless feature compute is used to maintain materialized views:
  - Based on the amount of data that changes in each base table
  - Based on the number of materialized views created on each table
  - Billed in 1-second increments

RECOMMENDATIONS

- Most materialized views should do one or both of the following:
    - Filter data (rows or columns)
    - Perform resource-intensive operations to store the result
- To optimize costs:
    - Use batched DML operations on base tables
    - Cluster the materialized view(s), leave the base table naturally clustered
- Apply clustering best practices when clustering materialized views
    - Limit the number of columns in the clustering key
    - With multiple columns, specify in order from lowest-to-highest cardinality
    - Use expressions where warranted

WHEN TO USE A MATERIALIZED VIEW?

Use a materialized view when:

- Results of the view don’t change often
- Results of the view are used often
- The query consumes significant resources

Use a standard view when:

- Results of the view change often
- Results are not used often
- The query is not resource-intensive, so it is not costly to re-run it

LAB EXERCISE: 12

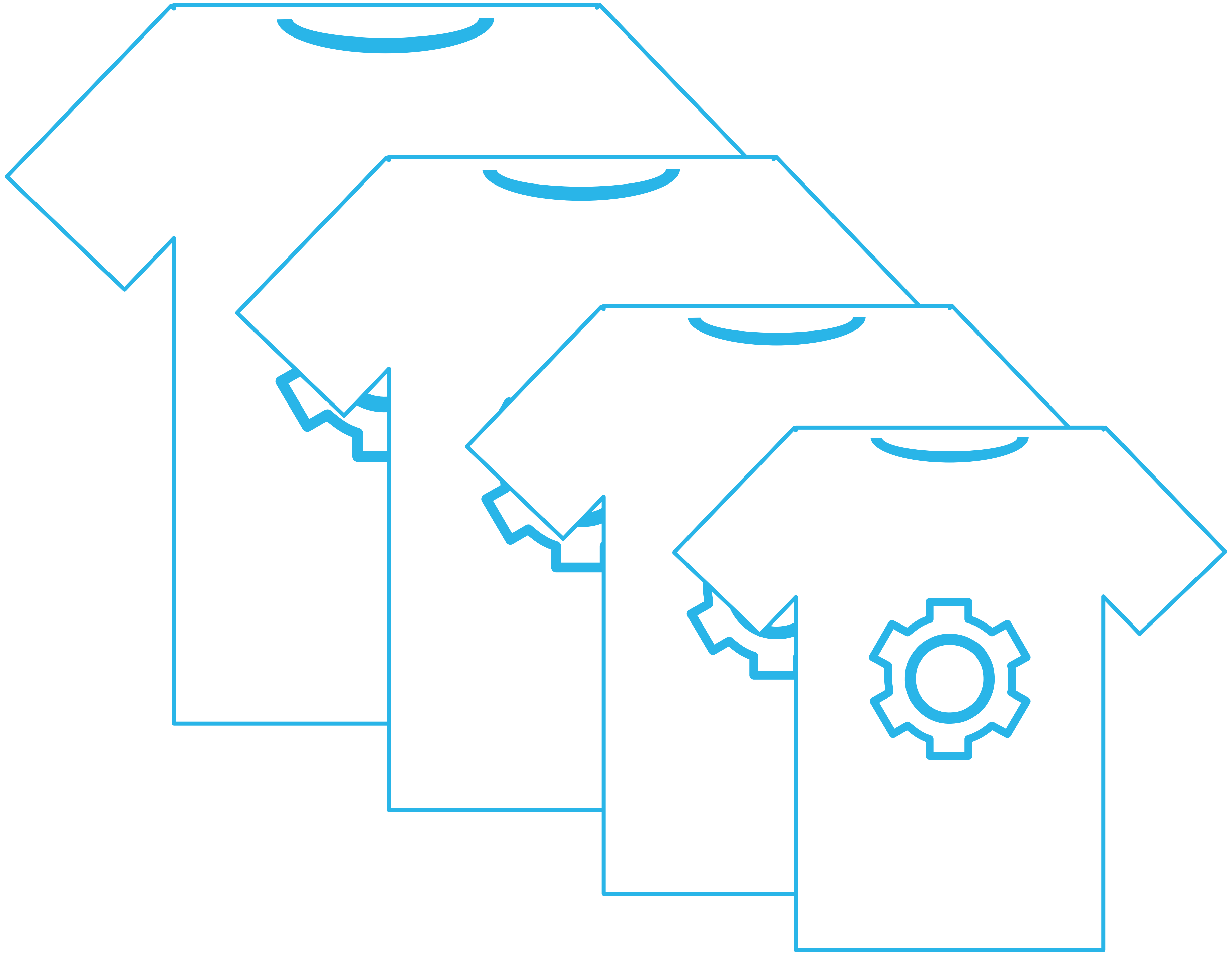
Materialized View Use Cases

20 minutes

- Cluster a Table Using a Timestamp Column Type
- Cluster a Table to Improve Performance
- Automatic Transparent Rewrite on Materialized Views
- Materialized Views on External Tables

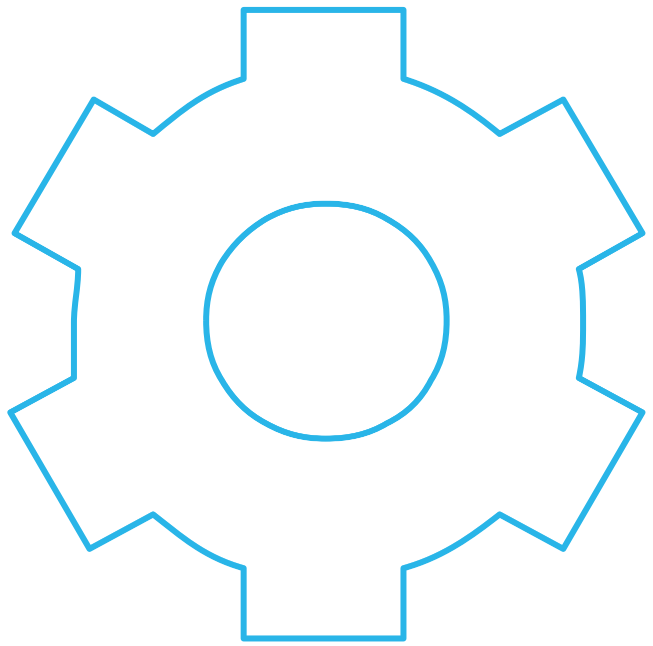
OPTIMIZE WAREHOUSE UTILIZATION

VIRTUAL WAREHOUSES



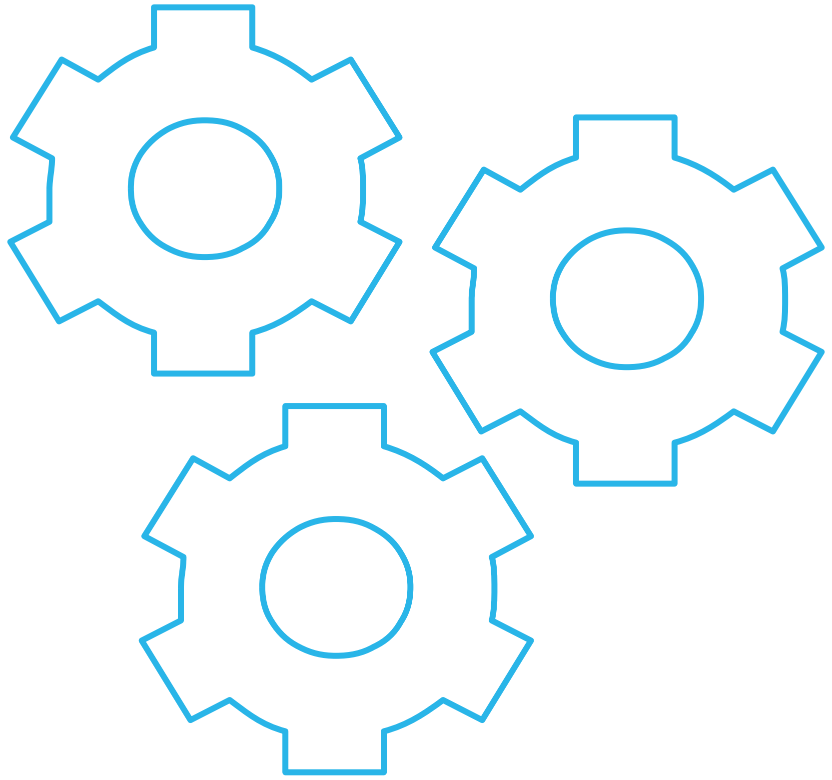
- A named wrapper around a cluster of servers with CPU, memory, and SSD
- "T-shirt" sizes, X-Small through 4X-Large
  - XS – equivalent of one server/cluster
  - Each size up doubles the resources
  - 4XL – equivalent of 128 servers/cluster
- Three Vectors of Warehouse Scaling
  - Across - Different warehouses for different types of workloads
  - Scale up to improve job performance
  - Scale out to reduce queuing time for concurrent-user warehouses

VIRTUAL WAREHOUSE TYPES



Standard

- Will only ever have a single compute cluster

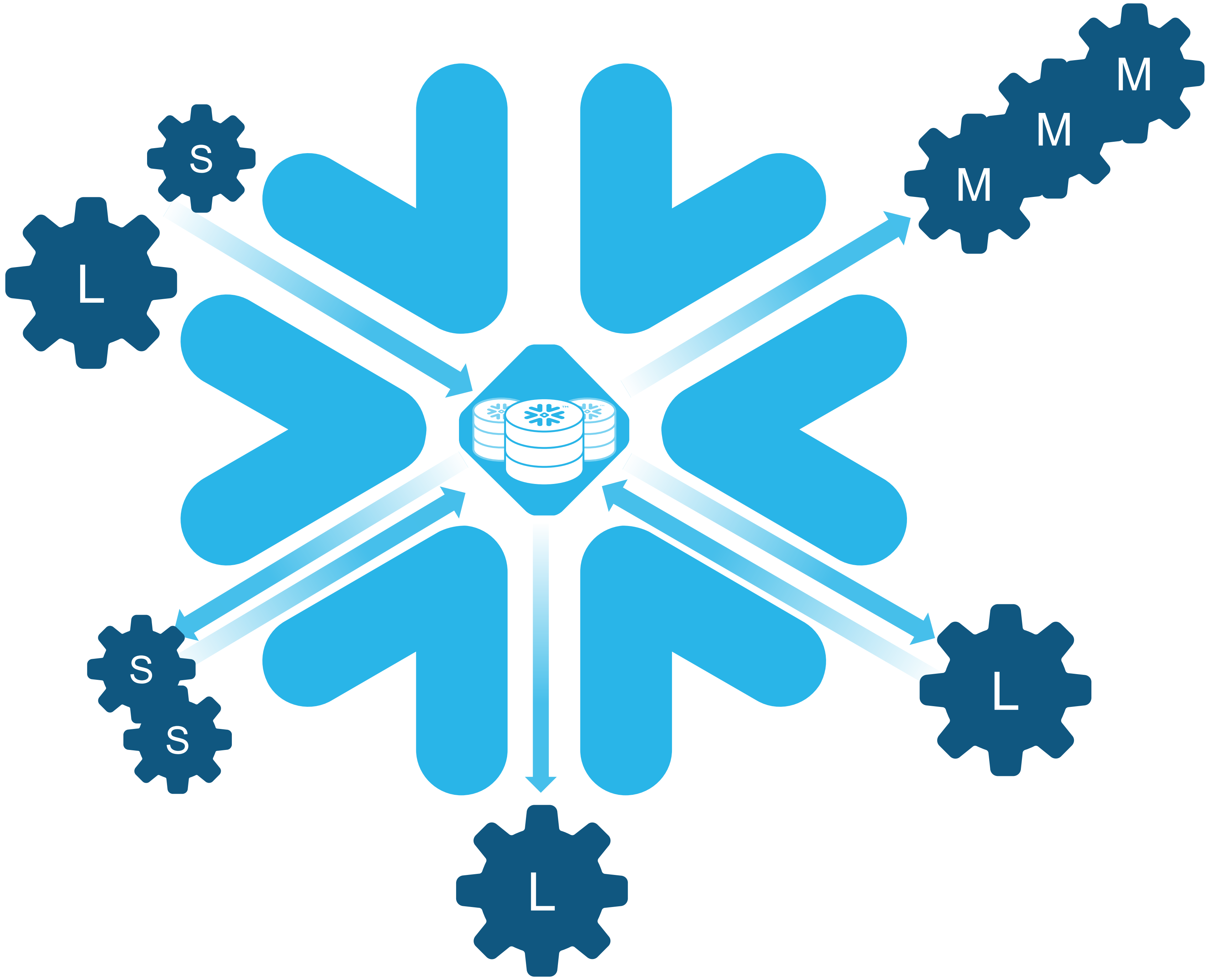


Multi-Cluster Warehouse (MCW)

- Can spawn additional compute clusters, or shut them down, to manage changes in user and concurrency needs
- Enterprise Edition feature

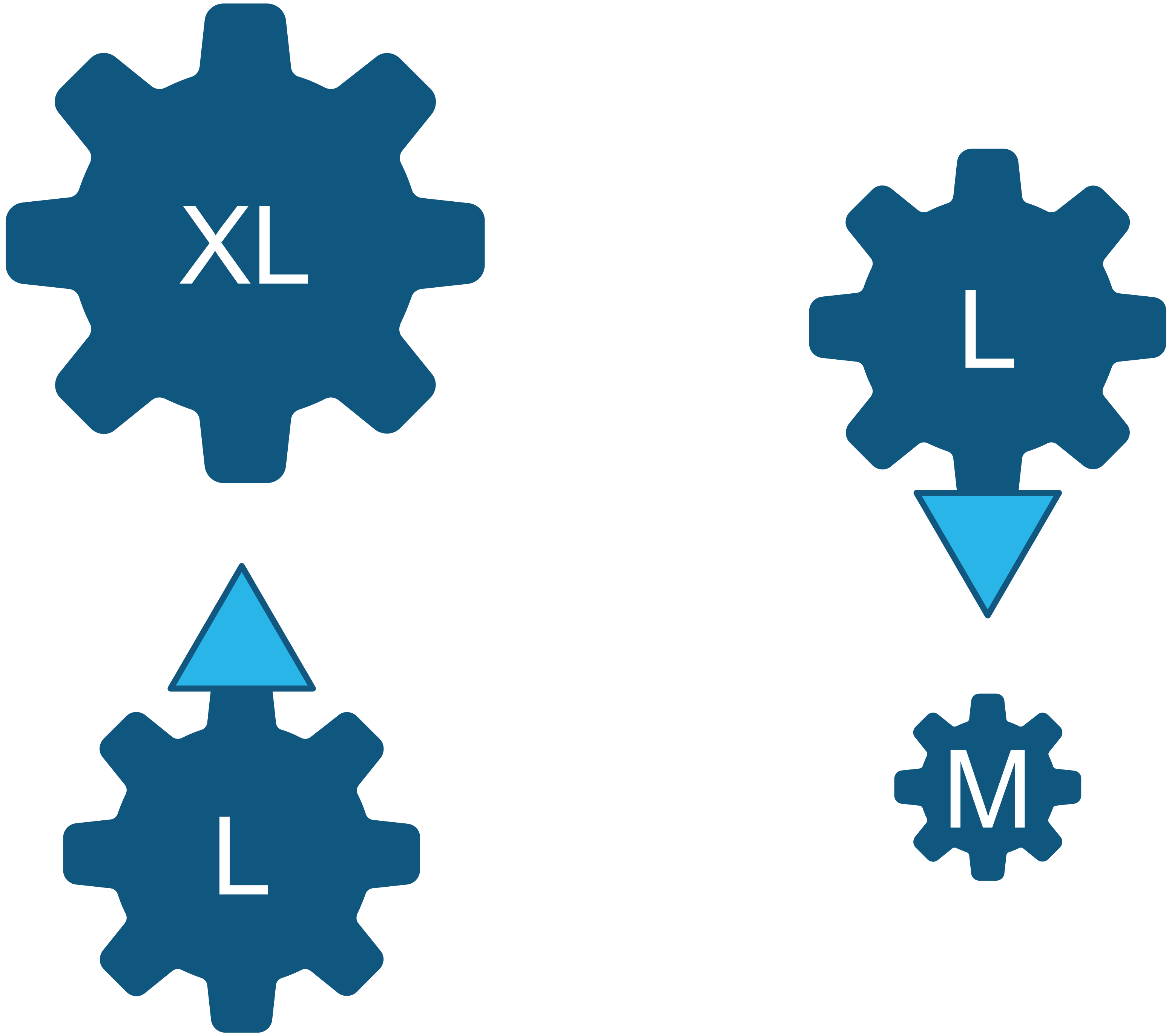
SCALE ACROSS

ELIMINATE RESOURCE CONTENTION



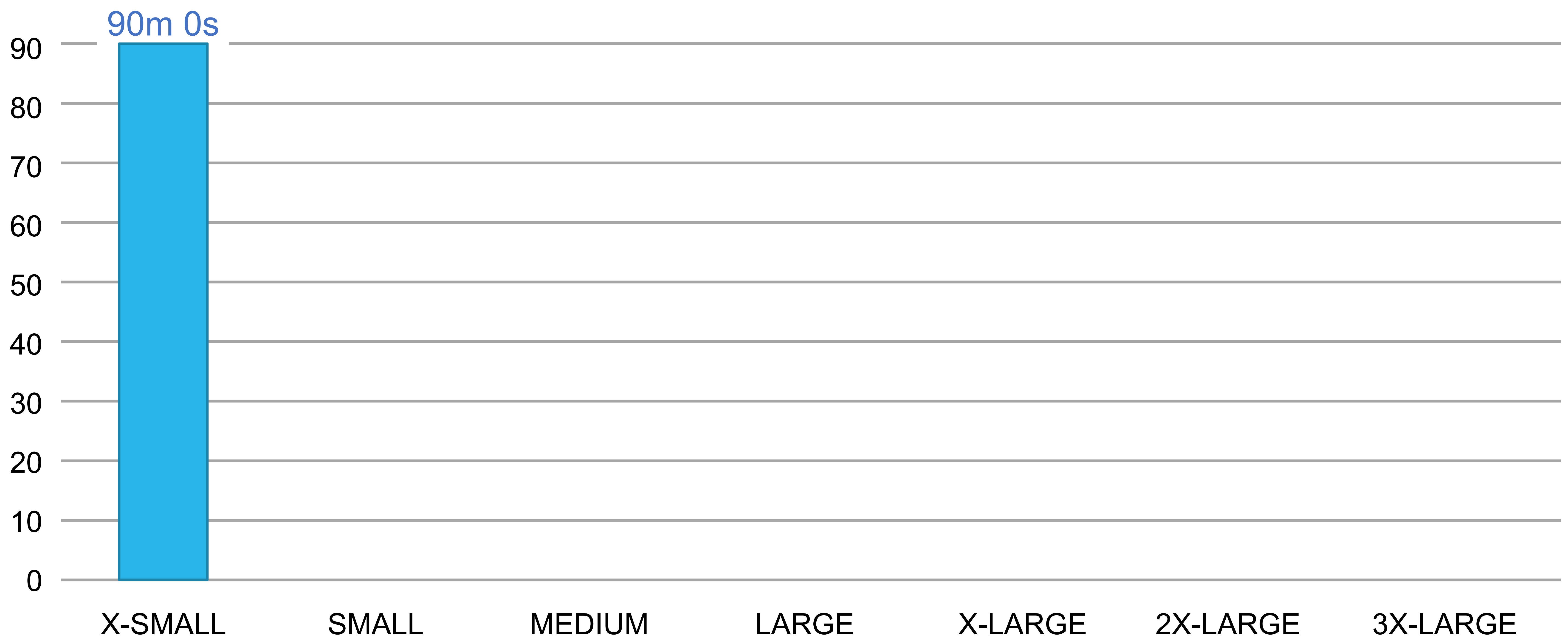
- Segregate virtual warehouses by workload
- Size each virtual warehouse to the task(s) it performs
- Use multi-cluster warehouses where appropriate
- If needed, assign warehouses of different sizes to a workload
  - Normal load ingests 16 files
  - Once a month, load ingests 200 files

SCALE UP (OR DOWN)

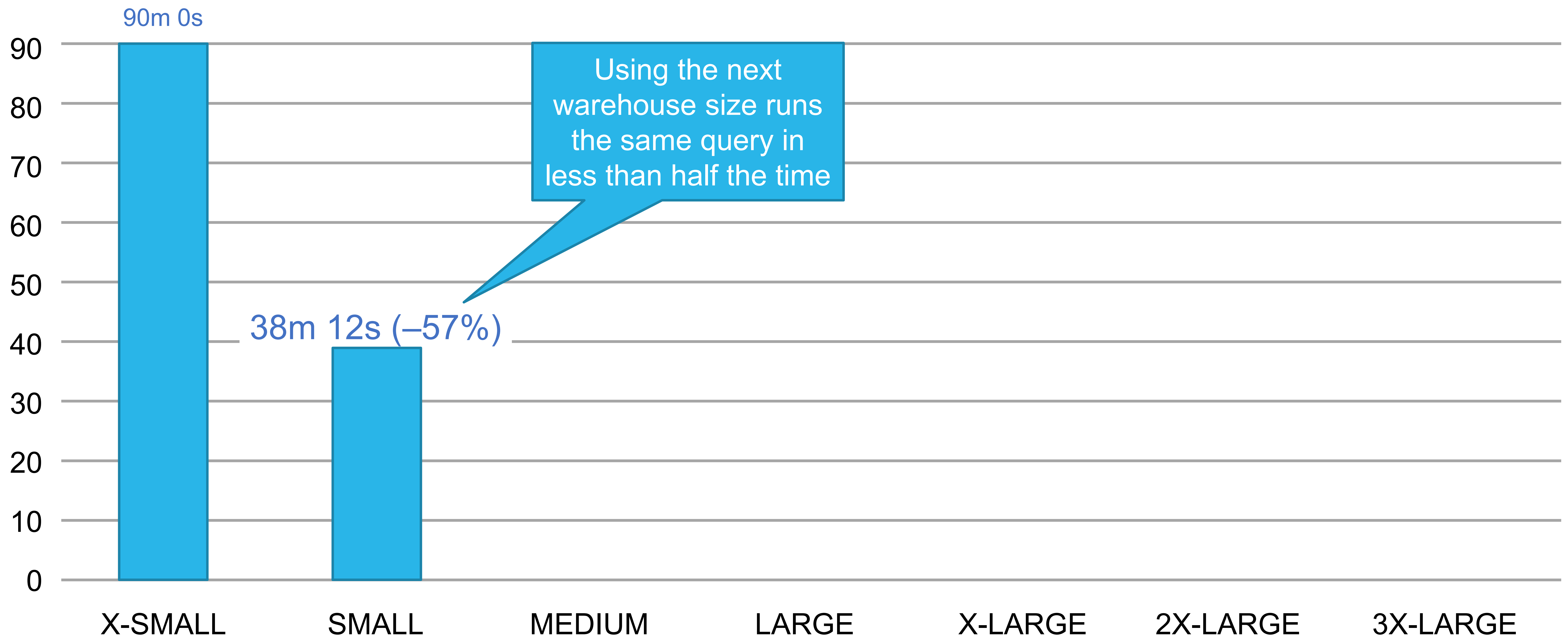


- Scale up to increase performance
- Make sure the warehouse is the right size for the workload
  - Each size up doubles the resources (and doubles the cost per second)
- Scaling is generally linear, to a point
  - Smaller warehouses are not necessarily more cost-efficient
- Find the sweet spot

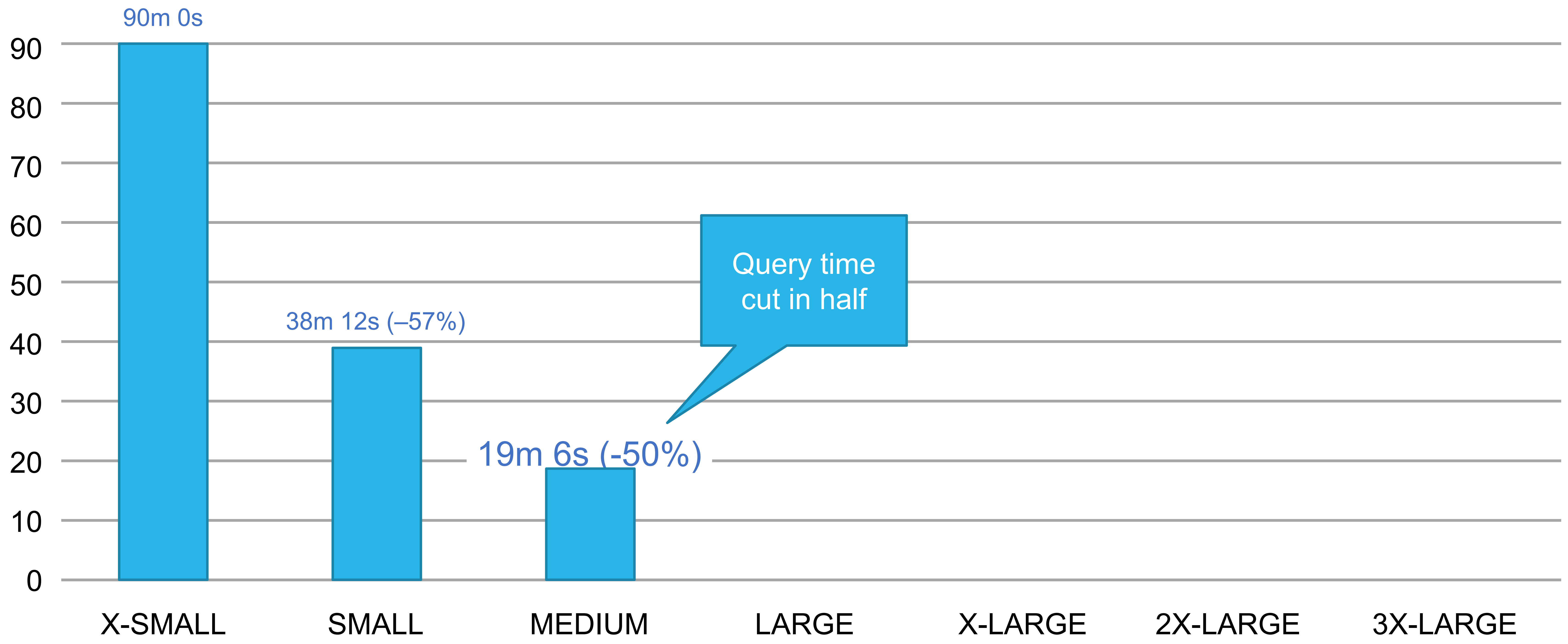
QUERY TIME BY WAREHOUSE SIZE



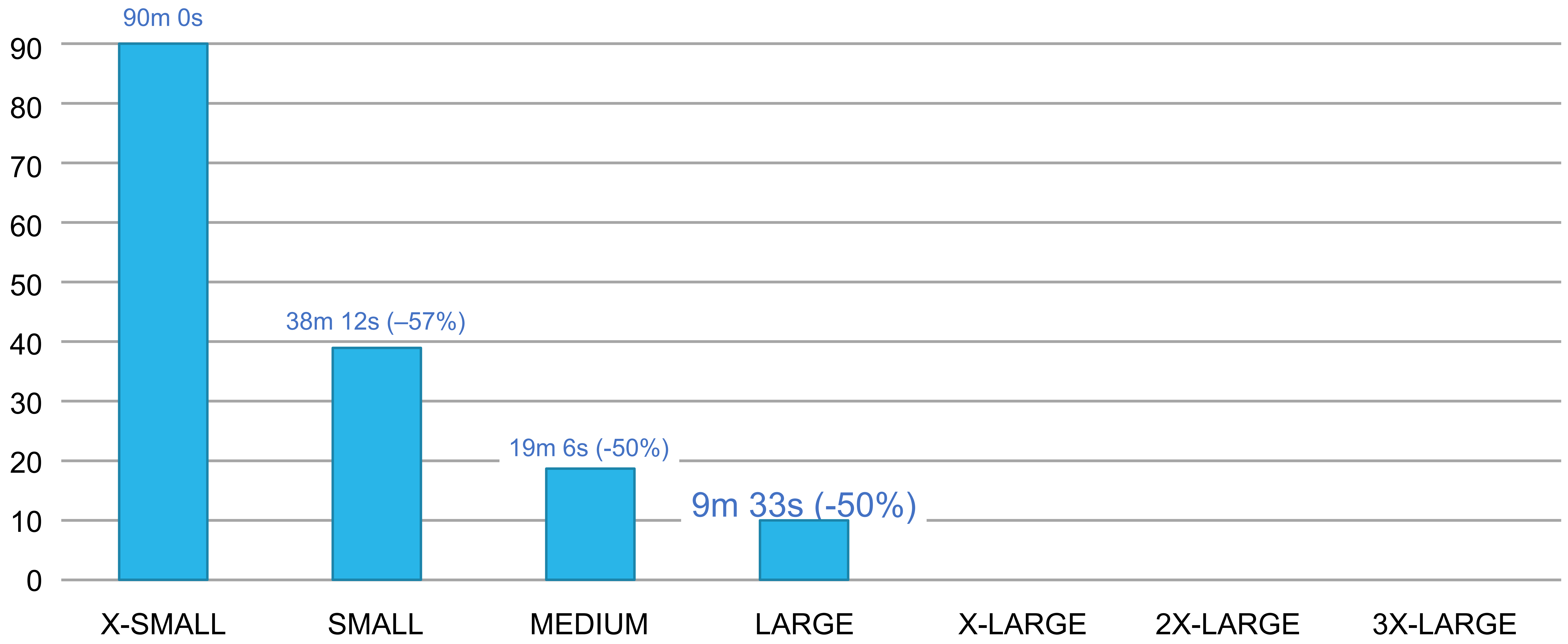
QUERY TIME BY WAREHOUSE SIZE



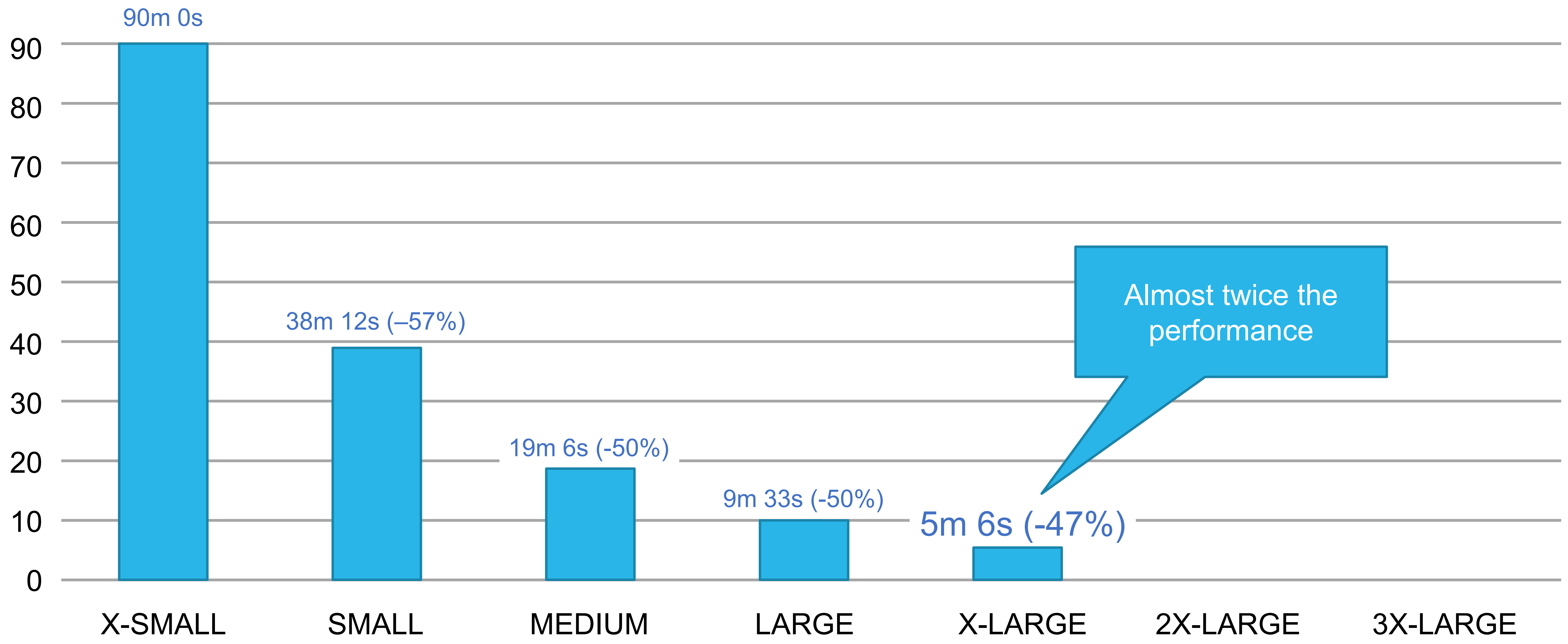
QUERY TIME BY WAREHOUSE SIZE



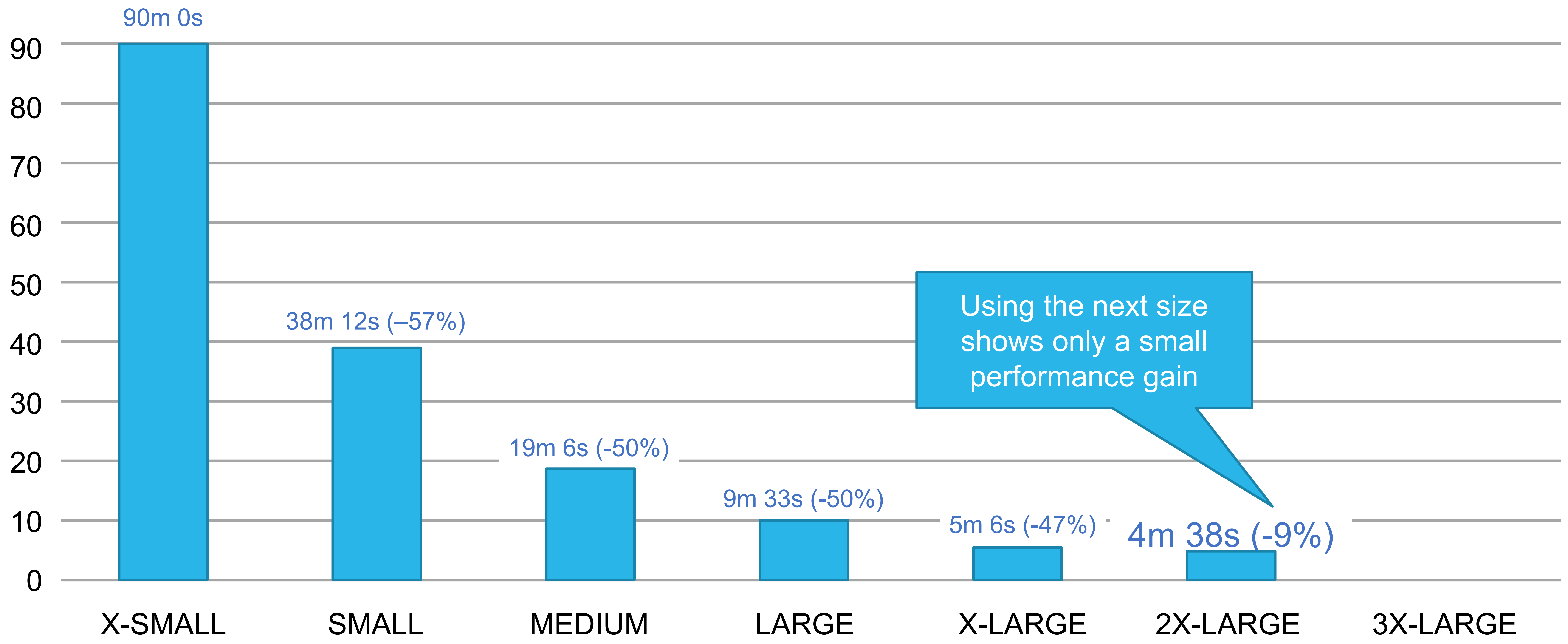
QUERY TIME BY WAREHOUSE SIZE



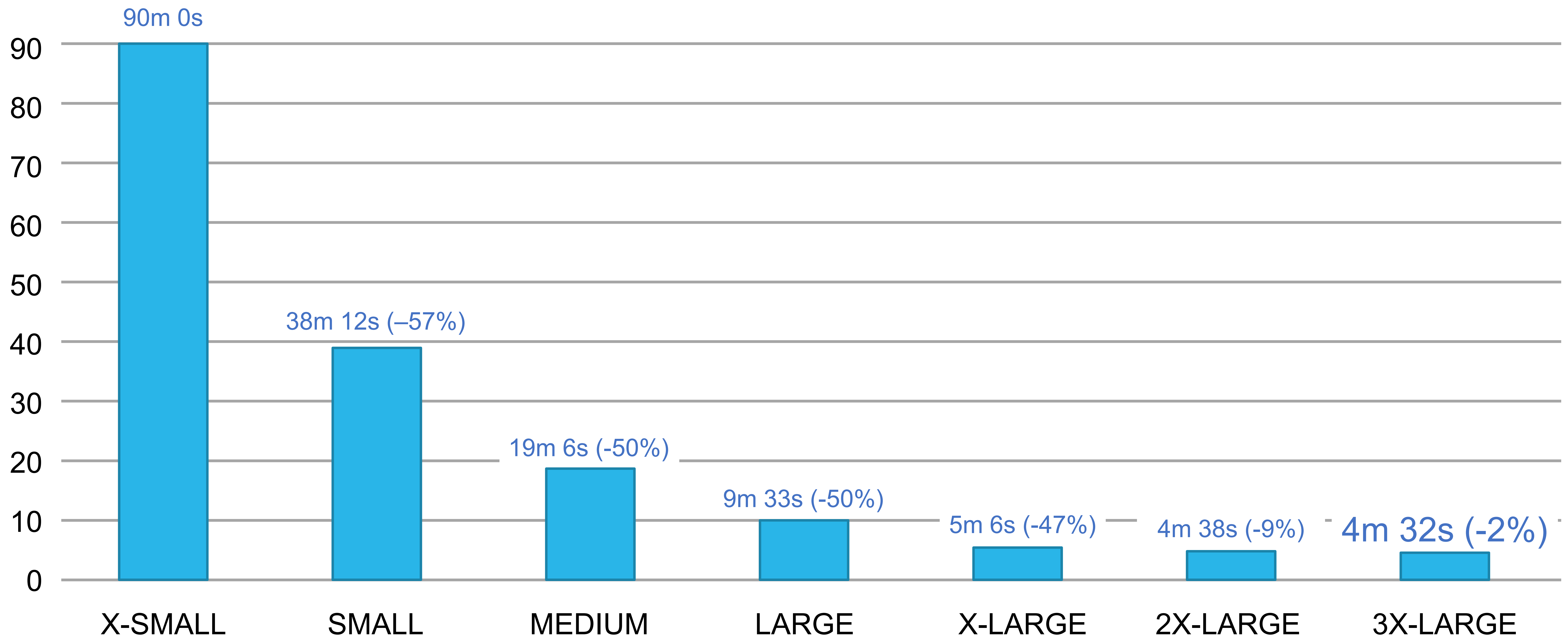
QUERY TIME BY WAREHOUSE SIZE



QUERY TIME BY WAREHOUSE SIZE

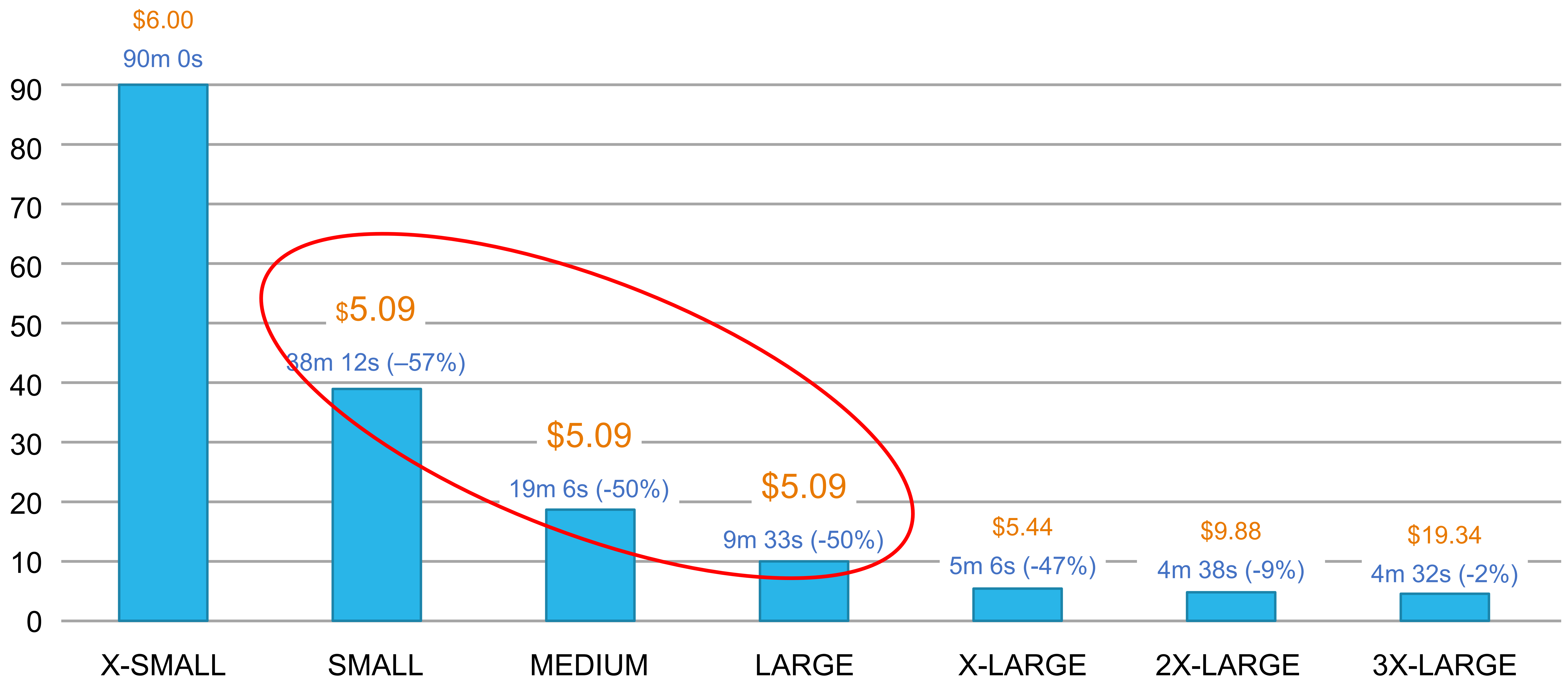


QUERY TIME BY WAREHOUSE SIZE



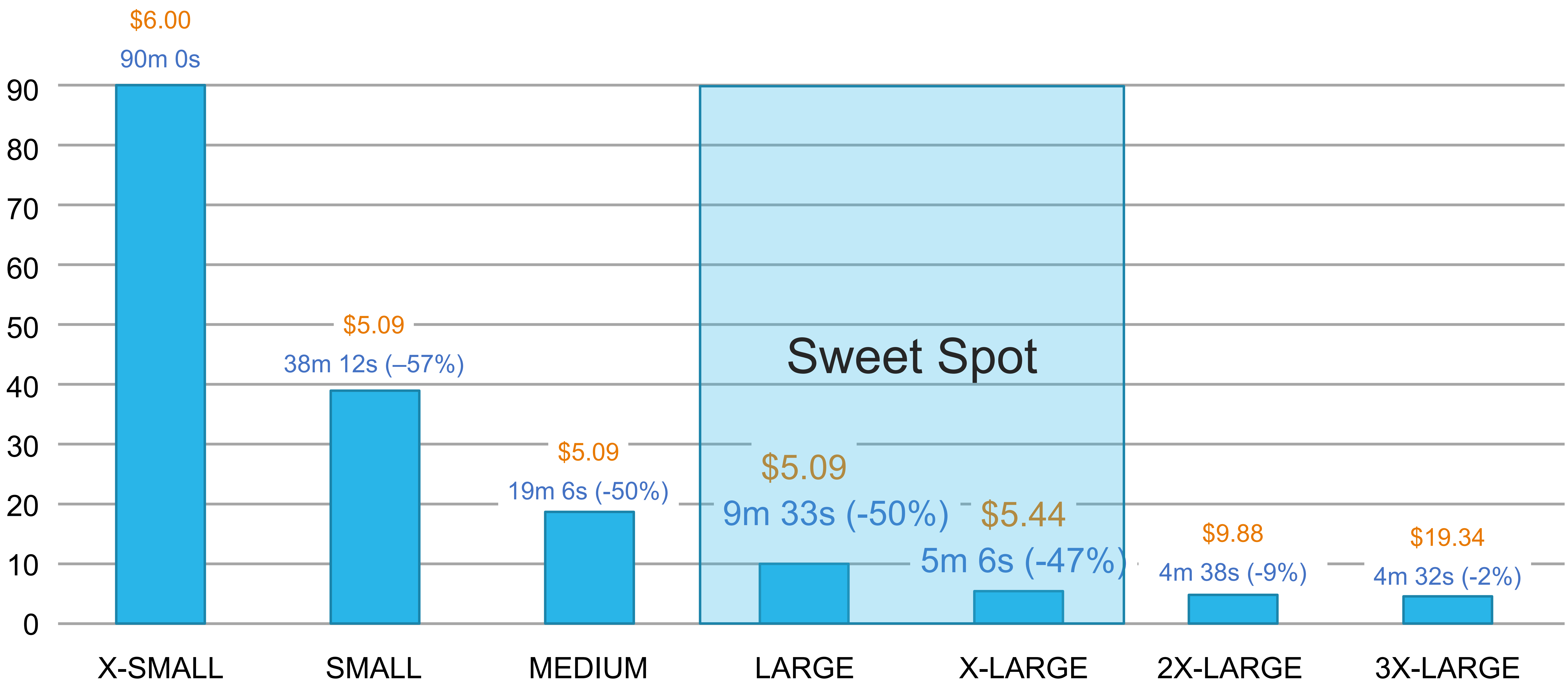
COST PER QUERY

CALCULATED USING $4.00 PER CREDIT



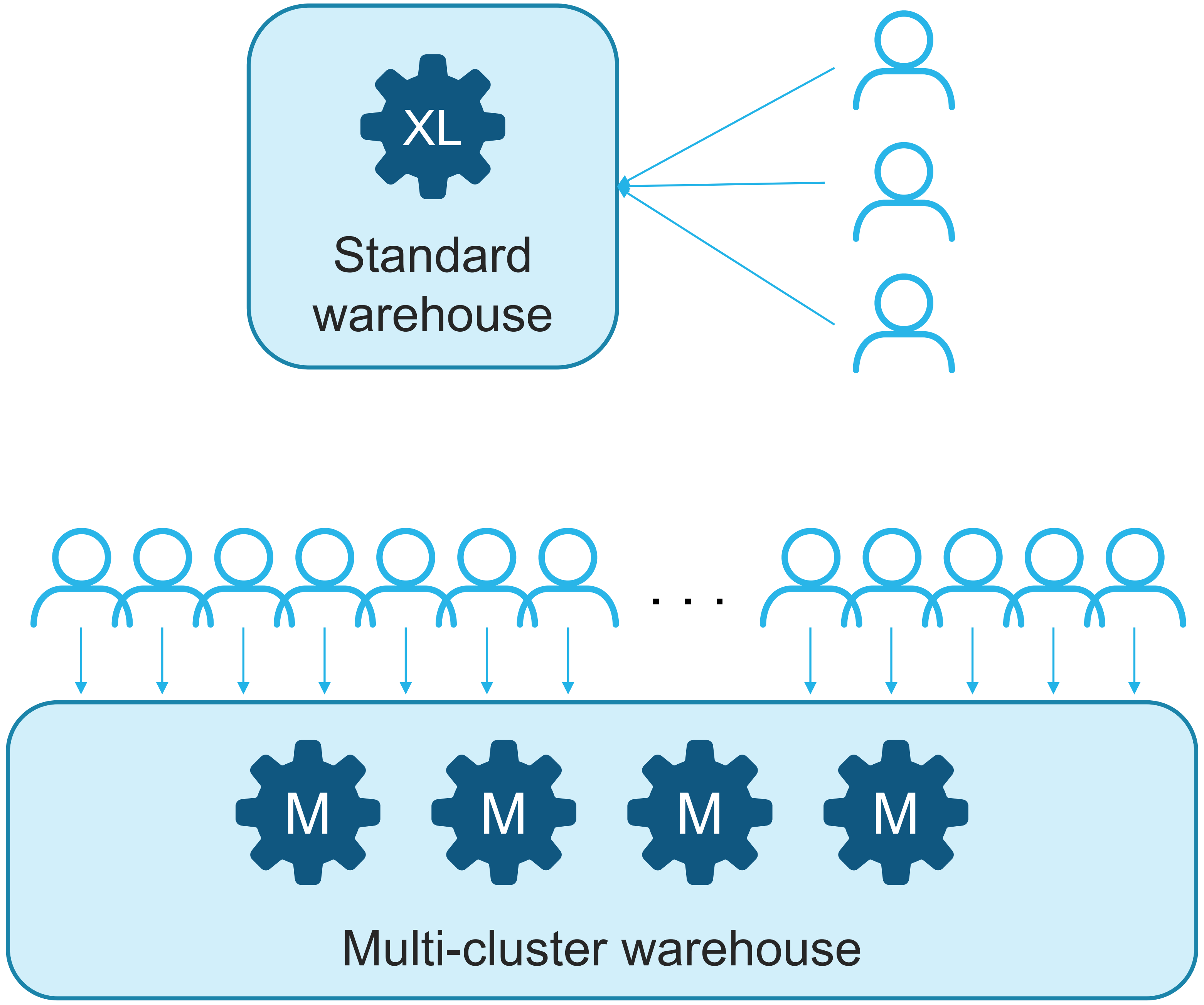
COST PER QUERY

CALCULATED USING $4.00 PER CREDIT



SCALE OUT (OR BACK)

HANDLE GREATER CONCURRENCY



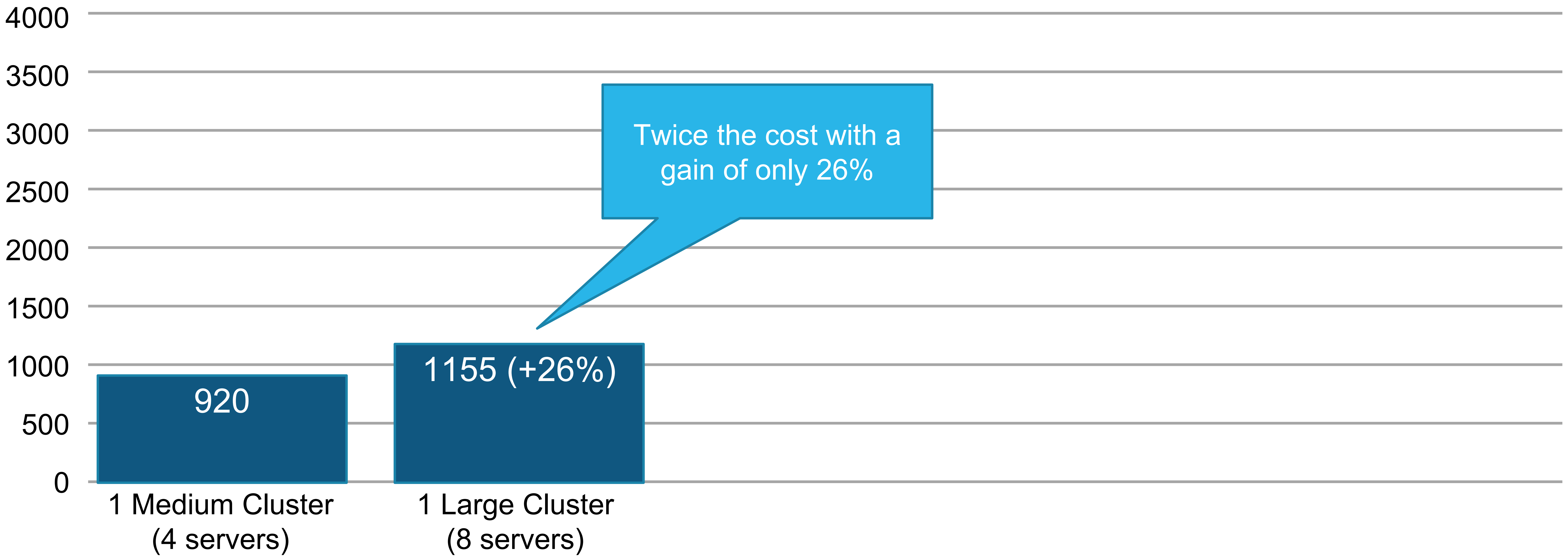
- Use a multi-cluster warehouse when many users are hitting the same warehouse
  - Example: reporting tools
- Scaling up will help to some extent with concurrency, but scaling out will help much more
- Each cluster added is the same size as the original
- Snowflake automatically scales out and back based on usage

SCALE UP VS SCALE OUT

RECORDS PROCESSED BY 50 CONCURRENT USERS

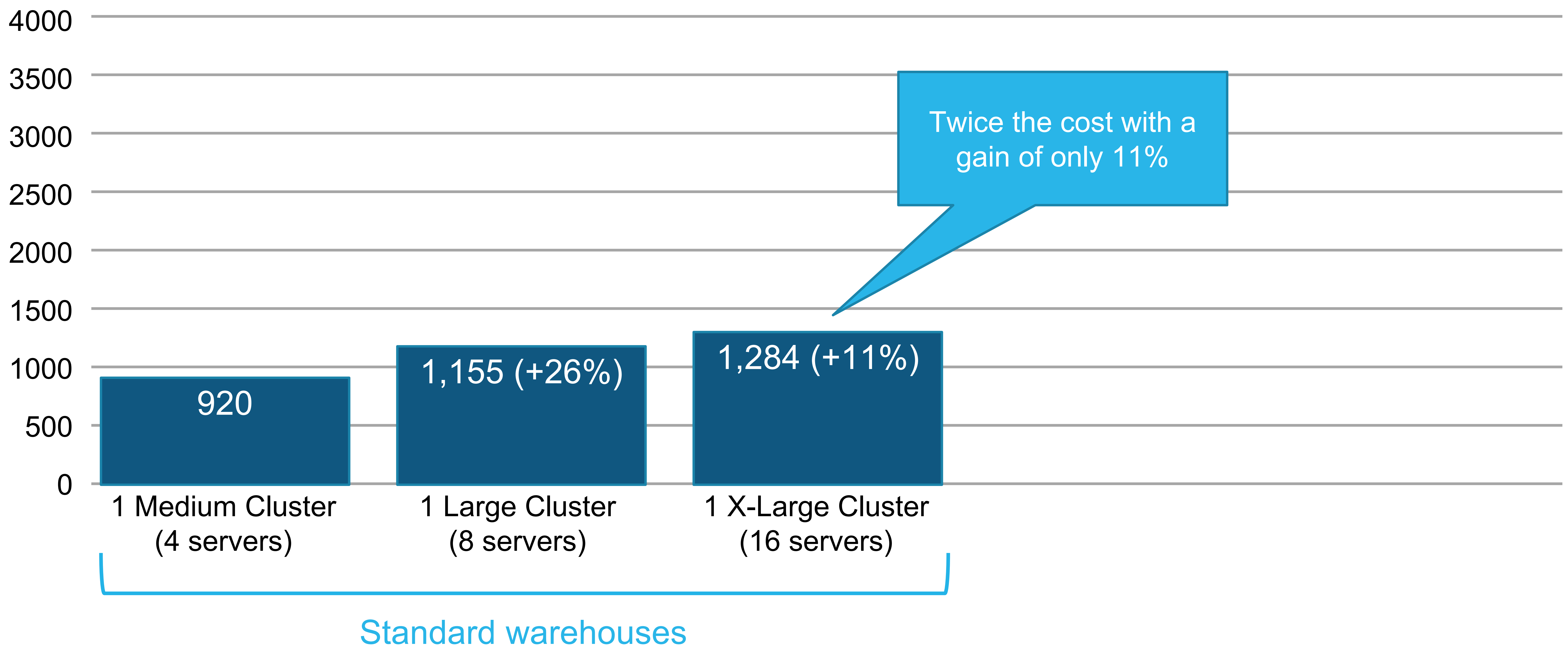
SCALE UP VS SCALE OUT

RECORDS PROCESSED BY 50 CONCURRENT USERS



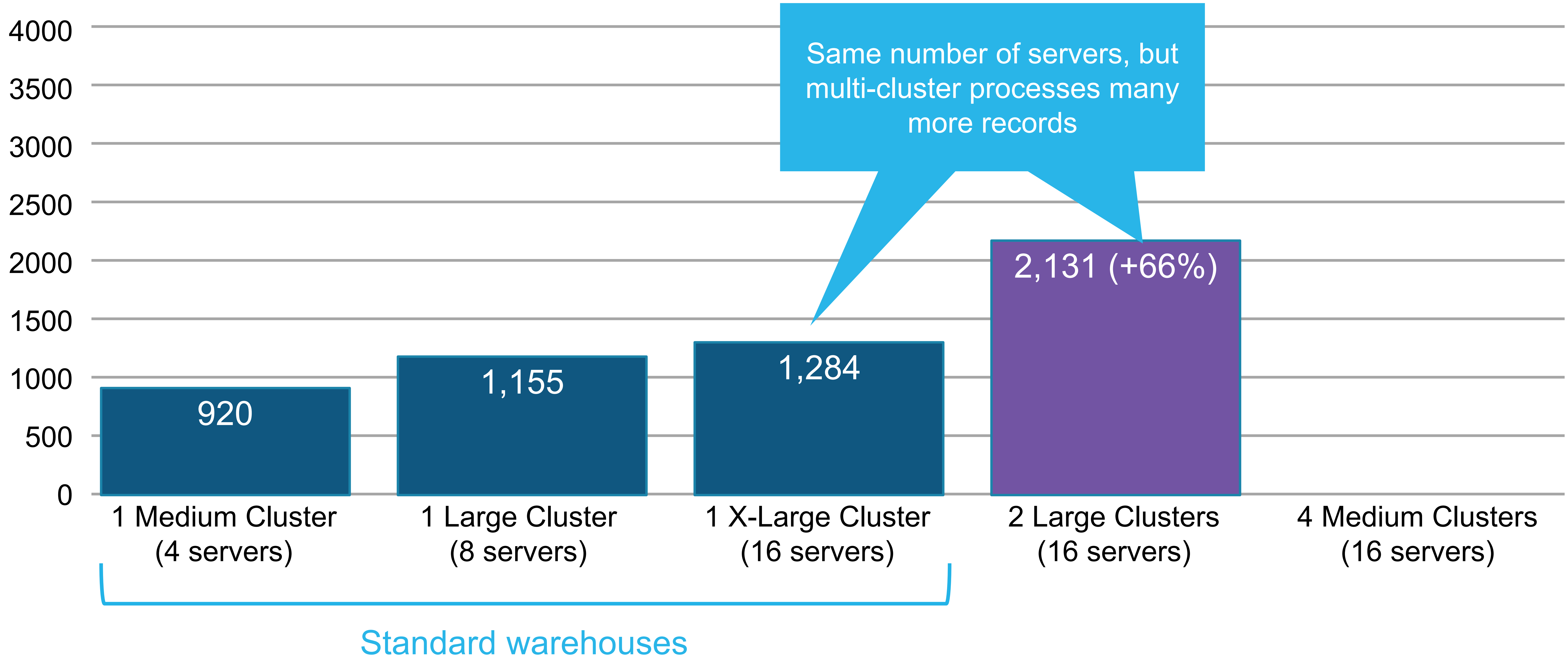
SCALE UP VS SCALE OUT

RECORDS PROCESSED BY 50 CONCURRENT USERS



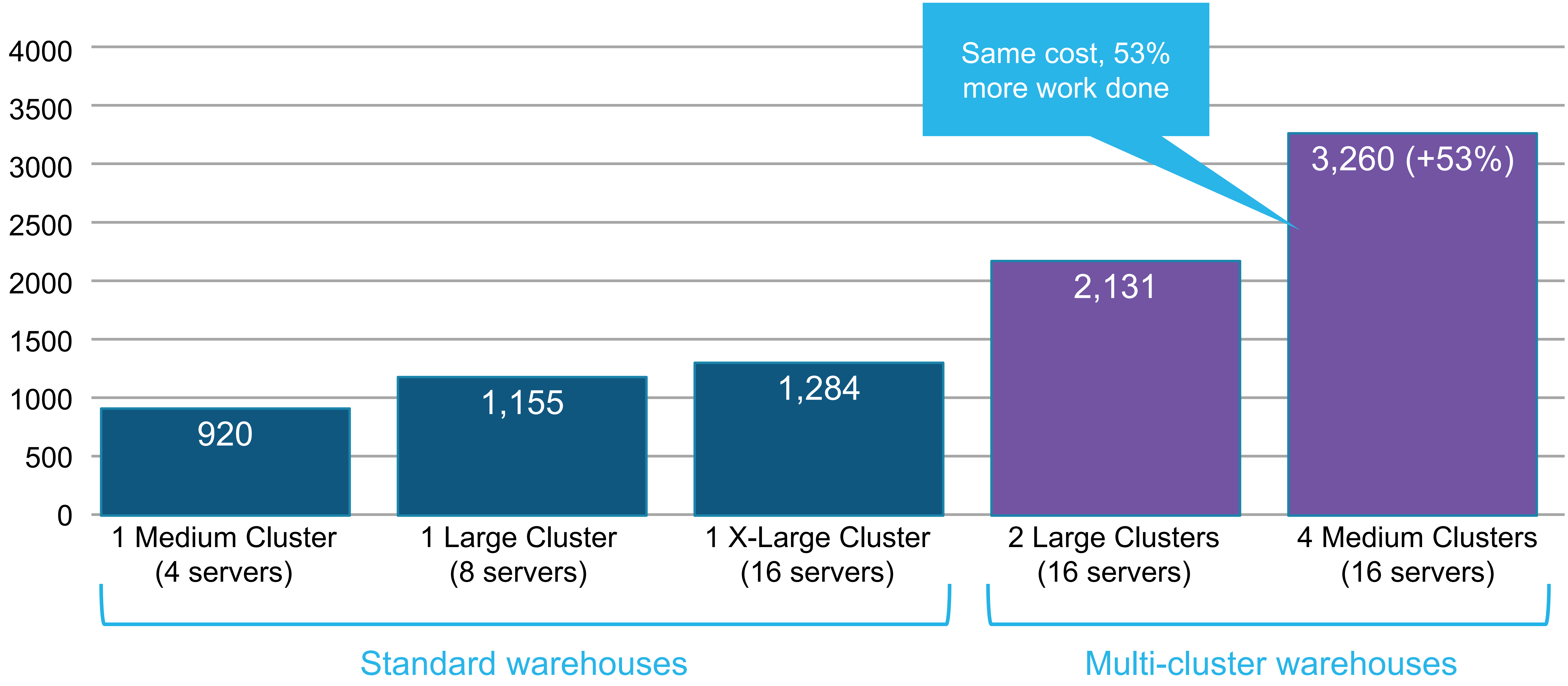
SCALE UP VS SCALE OUT

RECORDS PROCESSED BY 50 CONCURRENT USERS



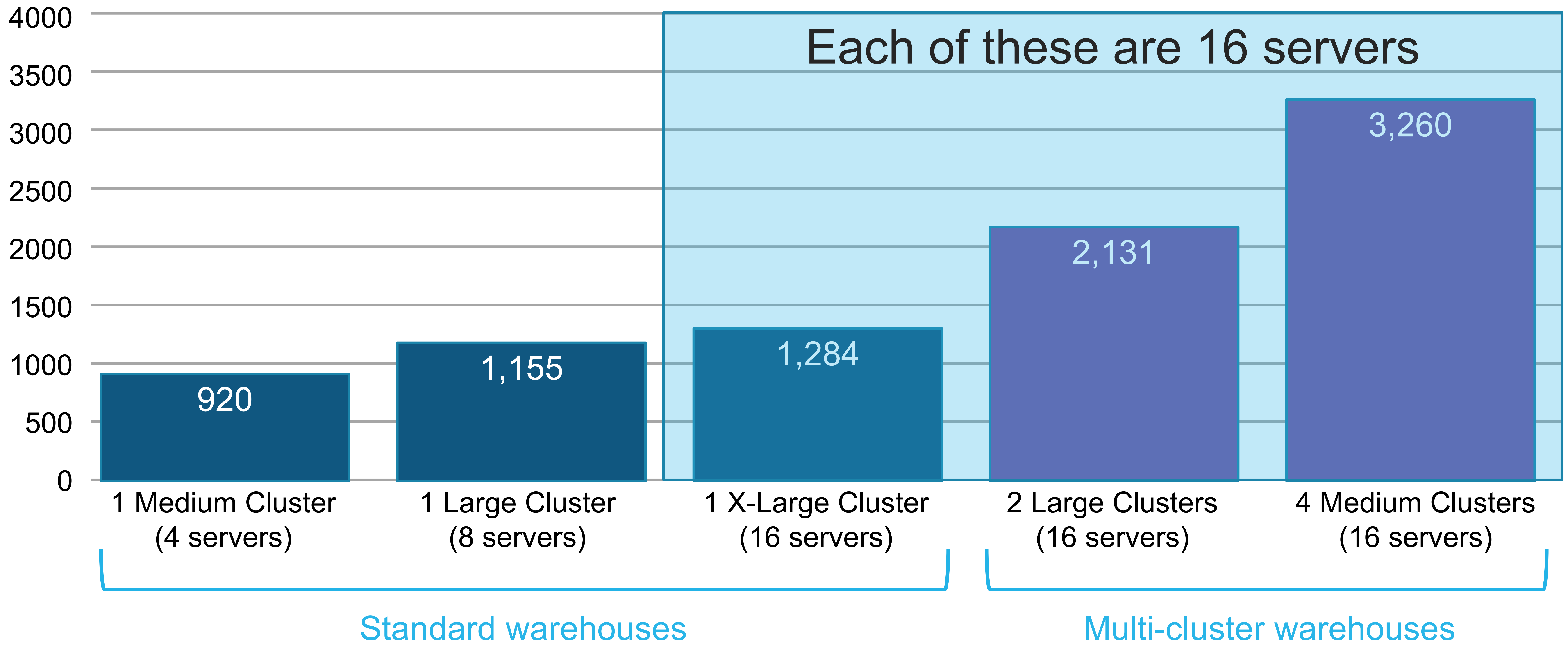
SCALE UP VS SCALE OUT

RECORDS PROCESSED BY 50 CONCURRENT USERS

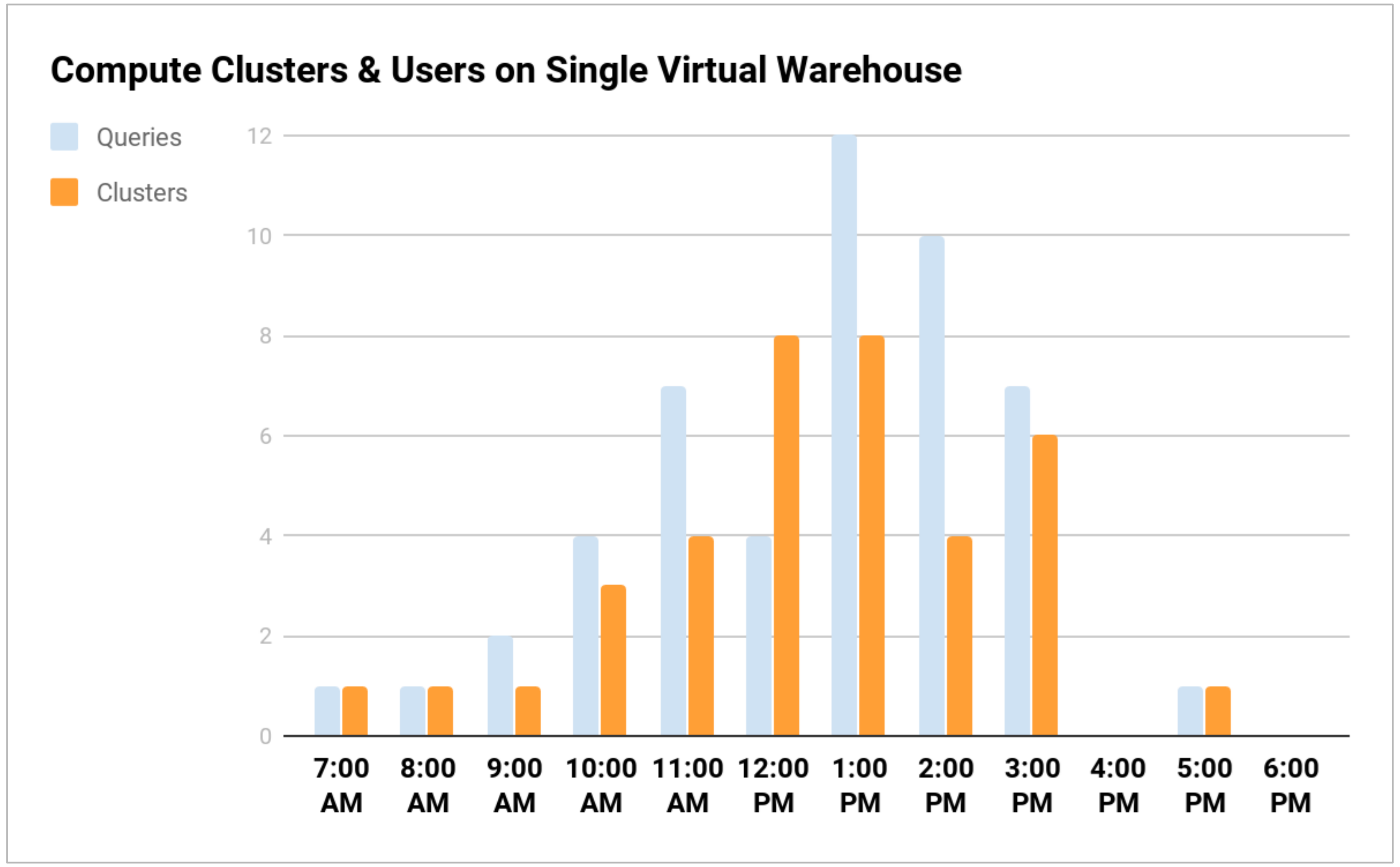


SCALE UP VS SCALE OUT

RECORDS PROCESSED BY 50 CONCURRENT USERS



AUTO-SCALING POLICIES
FOR MULTI-CLUSTER WAREHOUSES



- Standard policy
  - Scales out/back very quickly when usage changes
  - Best for relatively smooth usage curves
  - Increases or decreases in traffic tend to indicate trends, rather than anomalies
- Economy policy
  - Takes a "wait and see" approach when usage changes
  - Queries may queue for up to 6 minutes before a new cluster is started

USING STREAMS FOR CHANGE DATA CAPTURE

MODULE AGENDA

- Overview
- Streams and Offsets
- Using Streams

OVERVIEW

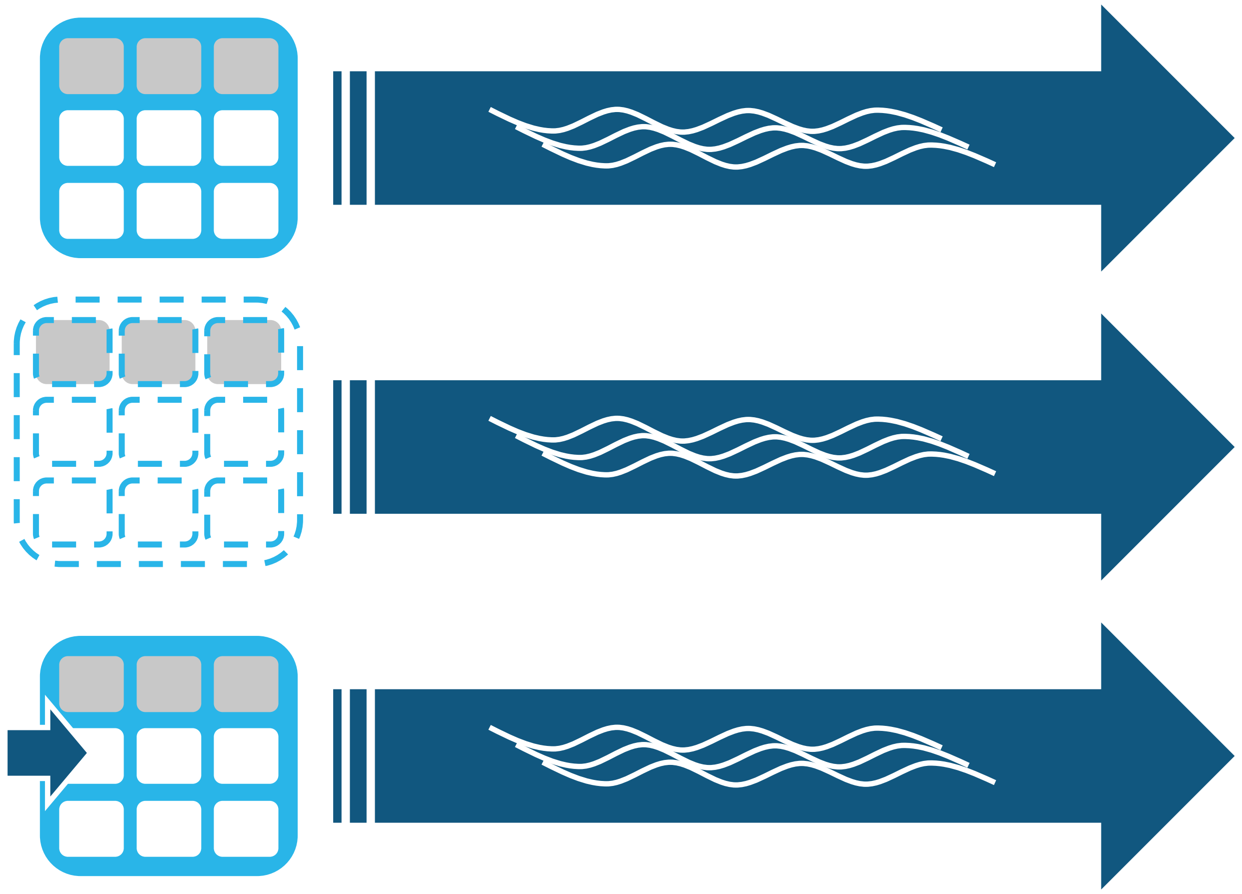
STREAMS

CHANGE DATA CAPTURE (CDC)

```
CREATE STREAM <stream> ON TABLE <src_table>;

CREATE STREAM <stream> ON TABLE <src_table>
   APPEND_ONLY = TRUE;
```

- A stream tracks DML changes made to a source table, along with metadata
- Change records in a stream can be "consumed" to take action based on the changes in the table
  - Example: Transform data added to a staging table, insert into production table
- Cannot insert or update streams (only the base tables)
- Two types
  - Standard (tracks inserts, updates, deletes)
  - Append-only (tracks inserts only)

FLEXIBILITY
CHANGE DATA CAPTURE (CDC)



Streams can be created on:

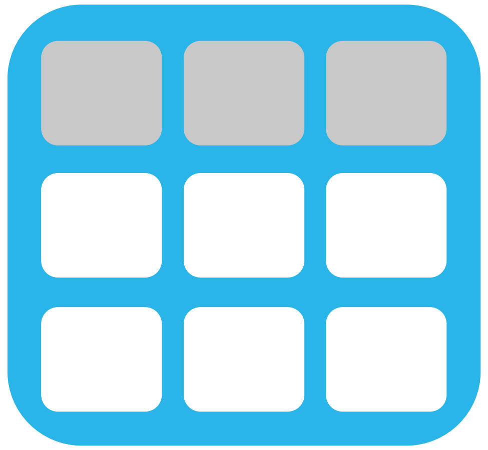
- Tables
- Shared Tables
- External Tables

TABLES VS STREAMS

DATA AT REST – DATA IN MOTION

TABLE

- Stores data
- Represents a single point in time
  - Reflects the most recent version



STREAM

- Stores metadata about the source table
- Represents every point in time
  - Each point is known as an offset

STREAM METADATA

- `METADATA$ACTION`
  - Indicates the action recorded (insert or delete)
- `METADATA$ISUPDATE`
  - Indicates whether the insert or delete actions were part of an UPDATE command
- `METADATA$ROW_ID`
  - Unique and immutable ID for the row

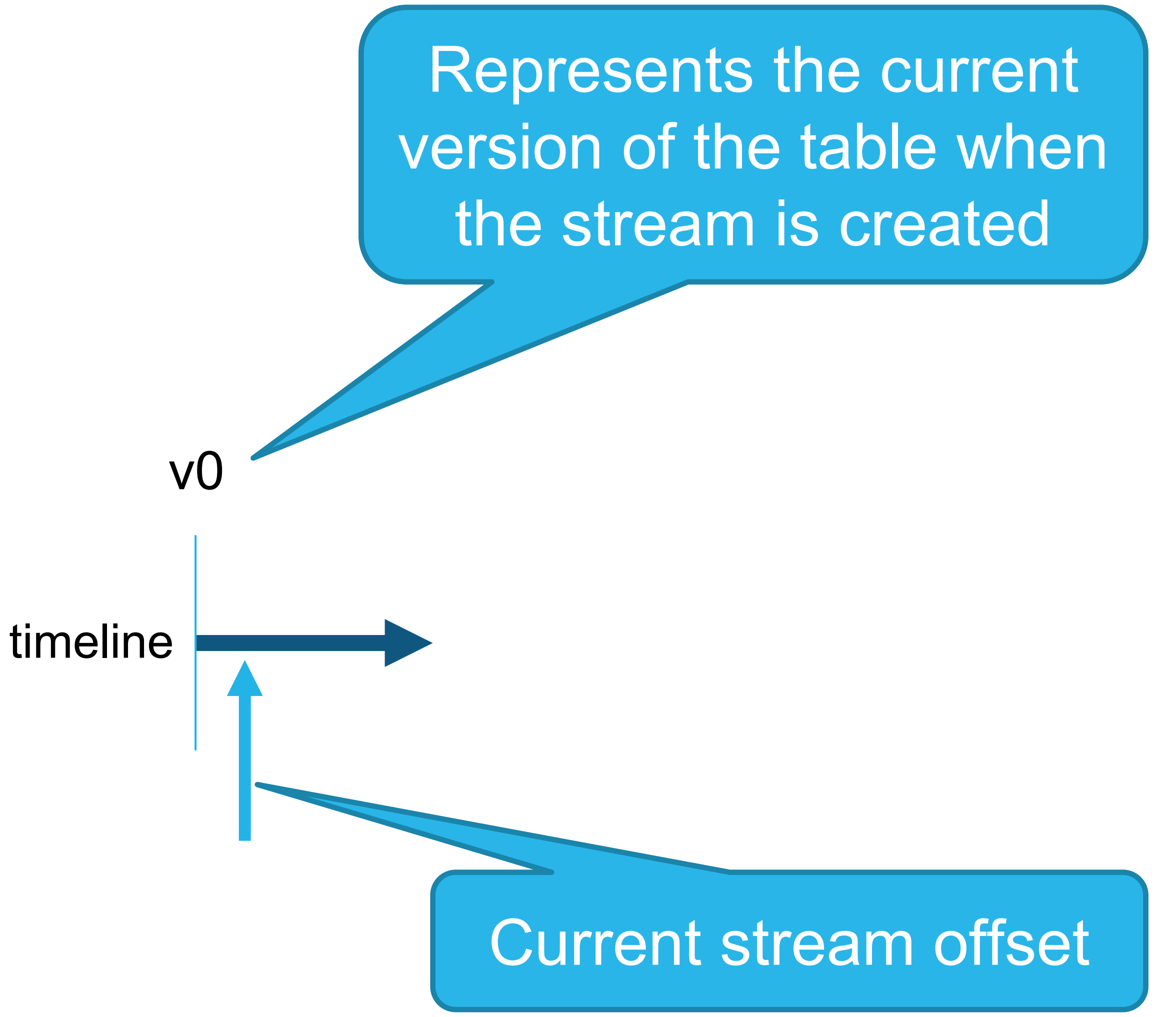
fruit	qty	METADATA$ACTION	METADATA$ISUPDATE	METADATA$ROW_ID
apple	5	INSERT	FALSE	17ccc3966ddc95be4b0a52124dd...
orange	2	INSERT	FALSE	d0a544dde34613eb0a1124c2321...
banana	6	DELETE	TRUE	22e6522bcce2de1332592cc5ee0...
banana	3	INSERT	TRUE	22e6522bcce2de1332592cc5ee0...

STREAMS AND OFFSETS

HOW DOES IT WORK?

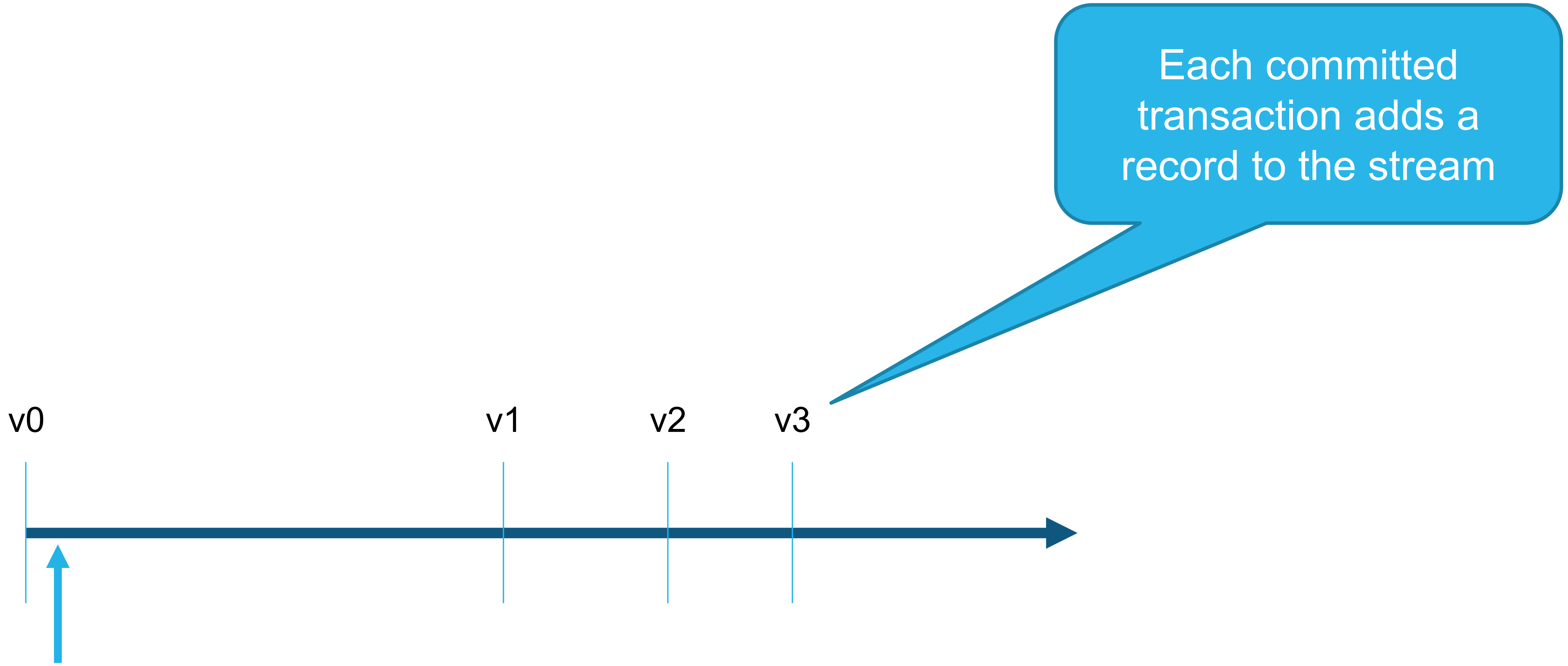
1. Stream is created

```
CREATE STREAM fruit_stream ON TABLE fruit_orders;
```



HOW DOES IT WORK?

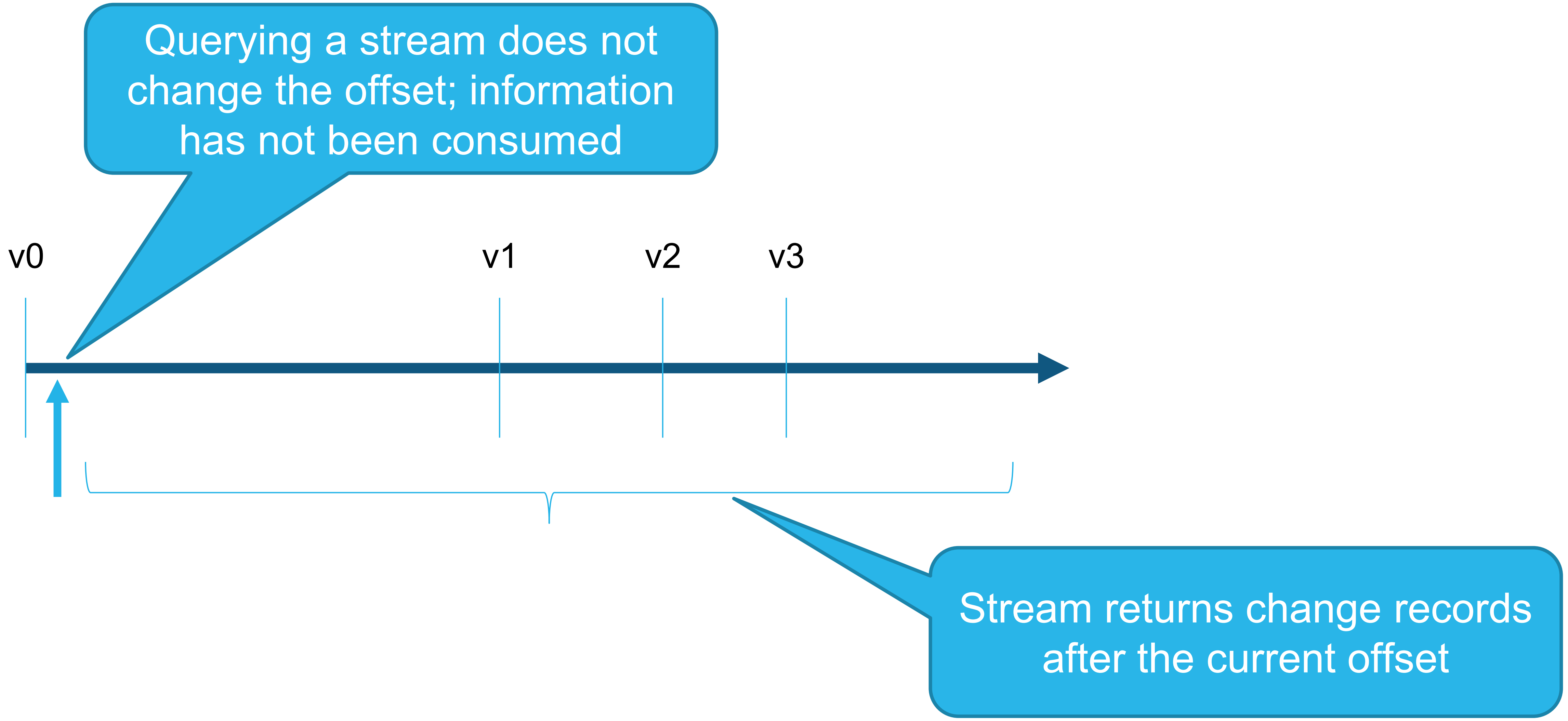
2. Transactions are committed on the source table



HOW DOES IT WORK?

3. Stream data is queried

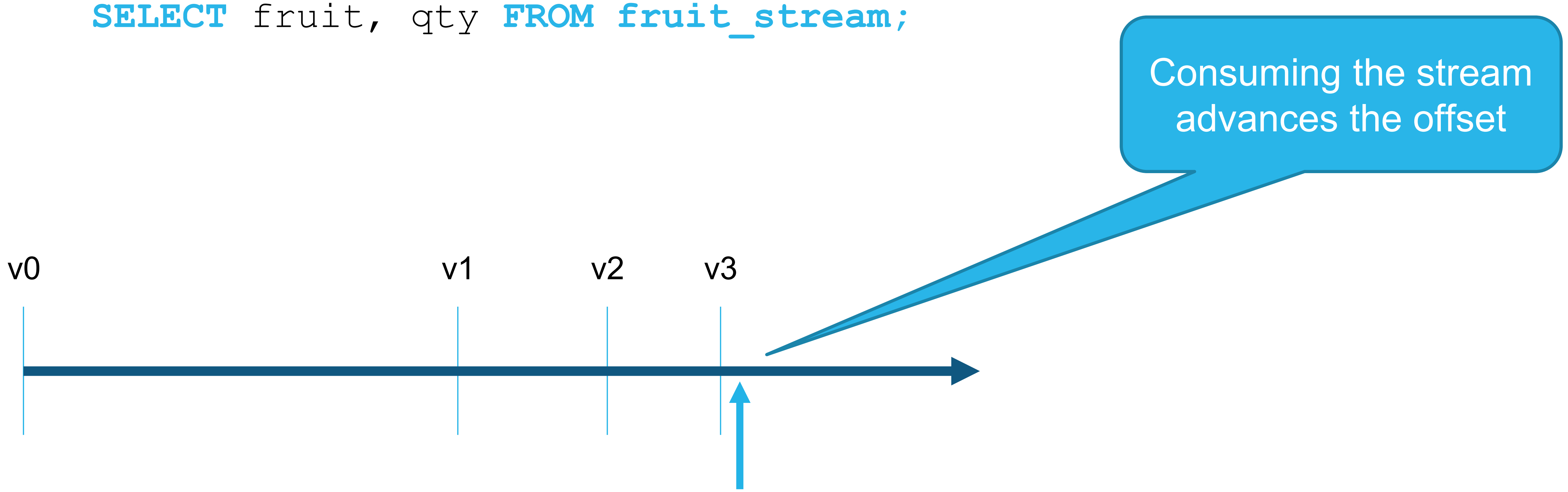
```
SELECT * FROM fruit_stream;
```



HOW DOES IT WORK?

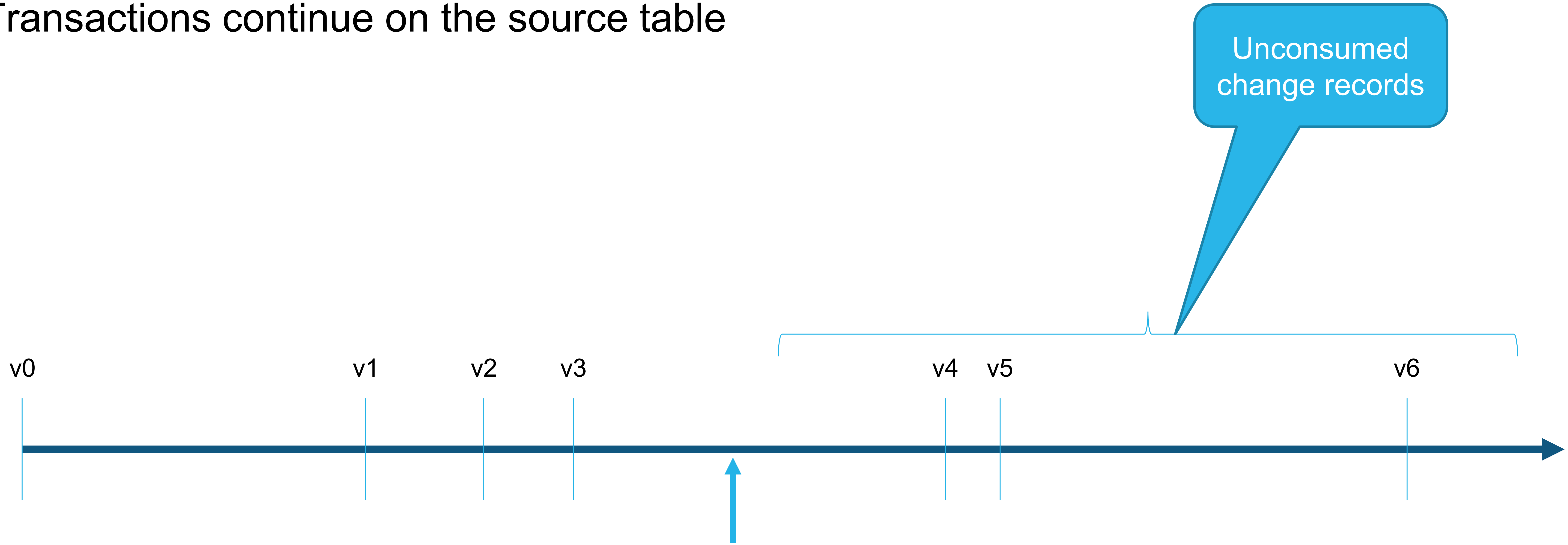
4. Stream data is used in a DML transaction ("consumed")

```
INSERT INTO order_history(product_name, quantity)
   SELECT fruit, qty FROM fruit_stream;
```



HOW DOES IT WORK

5. Transactions continue on the source table



TRACKING CHANGES

- Changes to source table are tracked by the stream
- When queried, stream returns change records

fruit	qty	METADATA$ACTION	METADATA$ISUPDATE	METADATA$ROW_ID
apple	5	INSERT	FALSE	17ccc3966ddc95be4b0a52124dd...
orange	2	INSERT	FALSE	d0a544dde34613eb0a1124c2321...

TRACKING CHANGES

- Effect of an update may be consolidated
  - Stream will always accurately reflect overall *effect* of transactions

```
INSERT INTO fruit_orders VALUES ('apple', 5), ('orange', 2);
SELECT * FROM fruit_stream;
```

fruit	qty	METADATA$ACTION	METADATA$ISUPDATE	METADATA$ROW_ID
apple	5	INSERT	FALSE	17ccc3966ddc95be4b0a52124dd...
orange	2	INSERT	FALSE	d0a544dde34613eb0a1124c2321...

```
UPDATE fruit_orders SET qty=1 WHERE fruit='orange';
SELECT * FROM fruit_stream;
```

fruit	qty	METADATA$ACTION	METADATA$ISUPDATE	METADATA$ROW_ID
apple	5	INSERT	FALSE	17ccc3966ddc95be4b0a52124dd...
orange	1	INSERT	FALSE	d0a544dde34613eb0a1124c2321...

TRACKING CHANGES

- If a row has been consumed, updates to that row cannot be consolidated

```
INSERT INTO fruit_stream VALUES (apple, 5), (orange, 2);
```

fruit	qty	METADATA$ACTION	METADATA$ISUPDATE	METADATA$ROW_ID
apple	5	INSERT	FALSE	17ccc3966ddc95be4b0a52124dd...
orange	2	INSERT	FALSE	d0a544dde34613eb0a1124c2321...

*** STREAM IS CONSUMED, OFFSET IS UPDATED ***

```
UPDATE fruit_stream SET qty=1 WHERE fruit=orange;
```

fruit	qty	METADATA$ACTION	METADATA$ISUPDATE	METADATA$ROW_ID
orange	2	**DELETE**	**TRUE**	d0a544dde34613eb0a1124c2321...
orange	1	**INSERT**	**TRUE**	d0a544dde34613eb0a1124c2321...

USING STREAMS

EXAMPLE: CONSUMING A STREAM

```
SELECT * FROM fruit_stream;
```

fruit	qty	METADATA$ACTION	METADATA$ISUPDATE	METADATA$ROW_ID
apple	5	INSERT	FALSE	17ccc3966ddc95be4b0a52124dd...
orange	1	INSERT	FALSE	d0a544dde34613eb0a1124c2321...

```
INSERT INTO target_table (fruit, qty)
SELECT fruit, qty FROM fruit_stream
WHERE metadata$action = 'INSERT'
```

```
SELECT * FROM fruit_stream;
```

fruit	qty	METADATA$ACTION	METADATA$ISUPDATE	METADATA$ROW_ID

STREAM FUNCTIONS

- See if the stream has any transactions past the current offset (new data to process)

```
SELECT system$stream_has_data('fruit_stream');
```

Row	SYSTEM$STREAM_HAS_DATA('FRUIT_STREAM')
1	FALSE

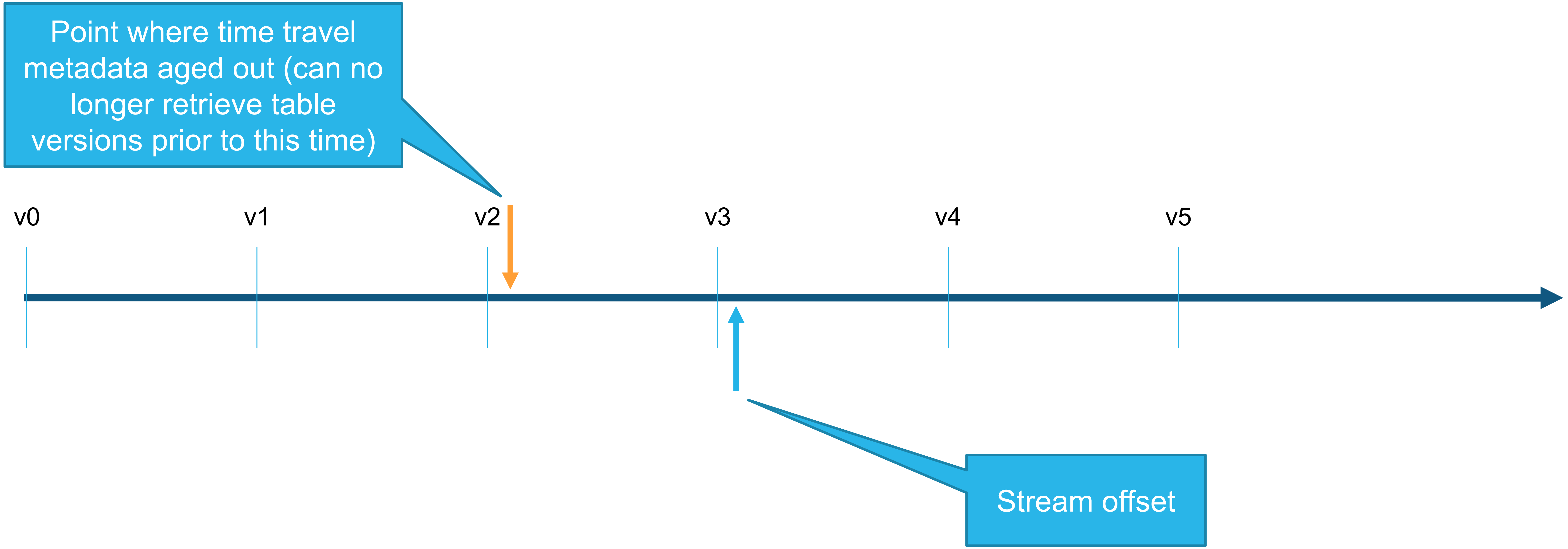
- Check the timestamp of the committed transaction where the offset is positioned

```
SELECT system$stream_get_table_timestamp('standard_stream_streamtest')
```

Row	SYSTEM$STREAM_GET_TABLE_TIMESTAMP('FRUIT_STREAM')
1	1611242753877

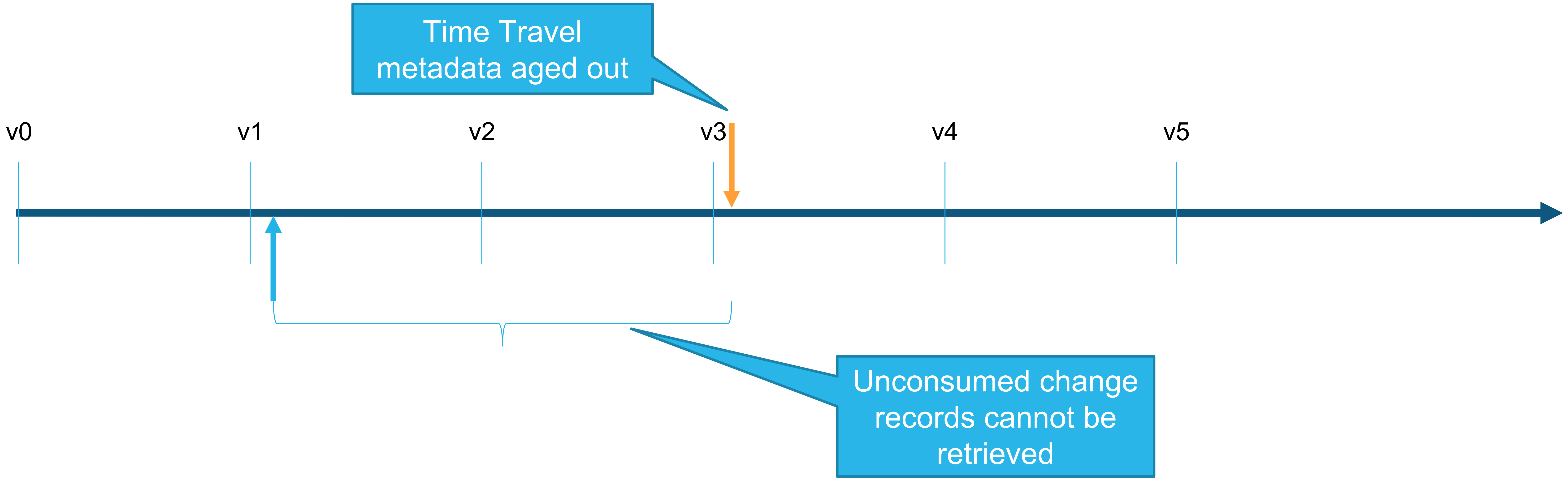
STREAMS AND TIME TRAVEL

- Consume transactions within the source table's retention time for Time Travel



STREAMS AND TIME TRAVEL

- When an offset is positioned earlier than the retention time for the table, the stream is stale
  - `DESCRIBE STREAM` or `SHOW STREAMS` will tell you if the stream is stale
  - To continue tracking changes, must drop and recreate the stream

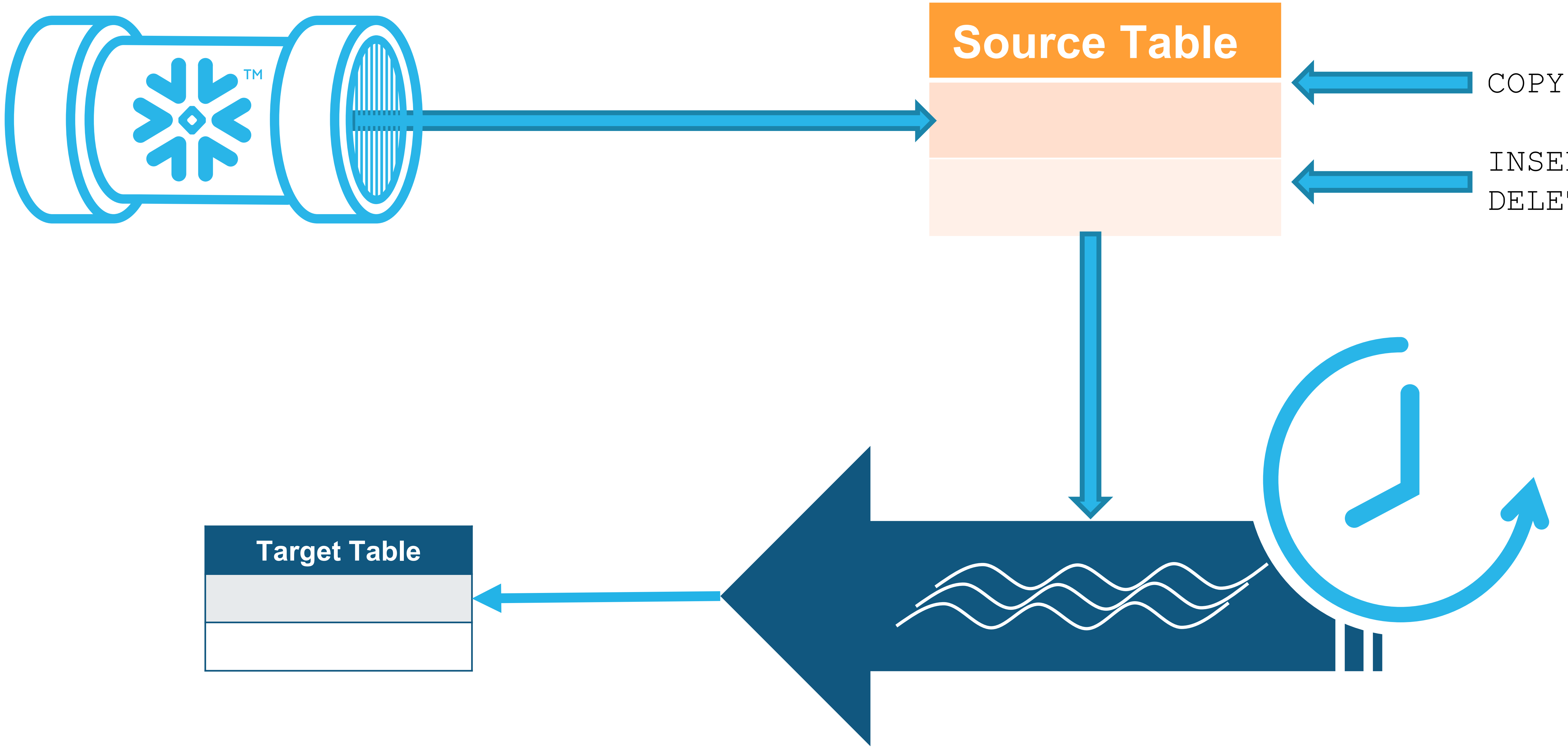


STREAMS AND TIME TRAVEL

- If the retention time on a source table is less than 14 days and its stream has not been consumed, Snowflake temporarily extends the data retention time (up to the value of `MAX_DATA_EXTENSION_TIME_IN_DAYS`) to prevent the stream from going stale
  - This incurs some additional storage cost

- Once the stream data is consumed, the retention time is set back to the source table's setting

PAIR STREAMS AND TASKS



COPY INTO...

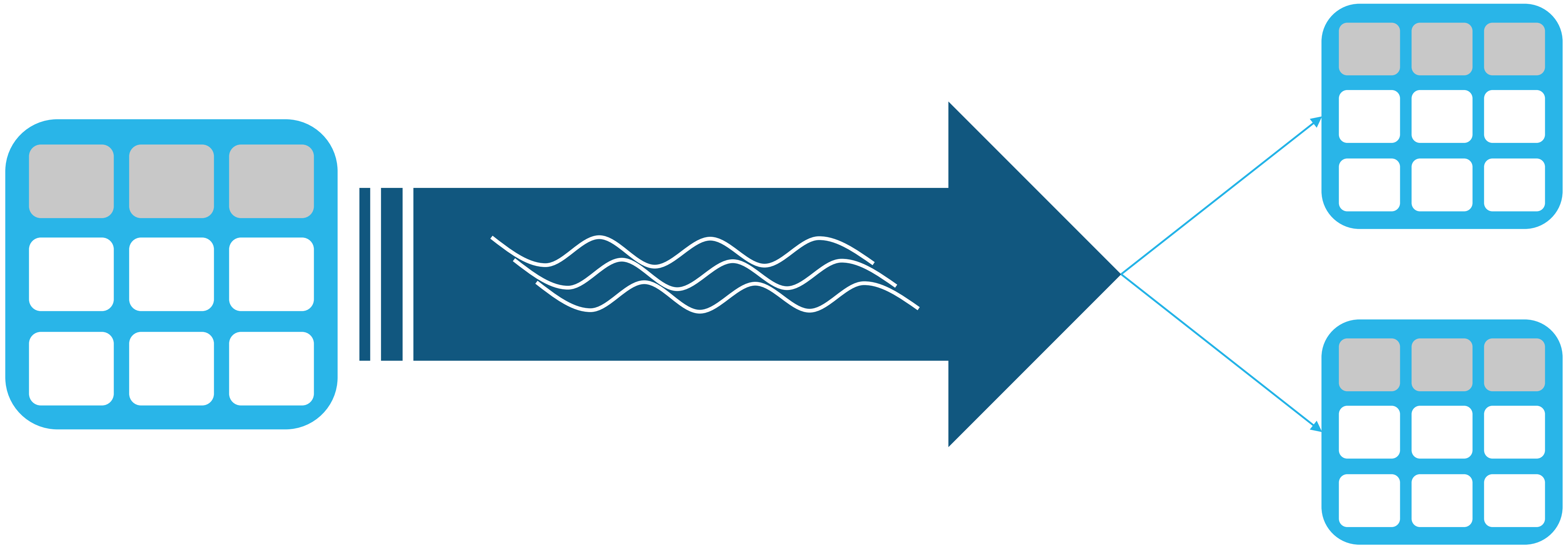
INSERT,UPDATE,
DELETE...

```
CREATE TASK consume_stream
   WAREHOUSE = my_warehouse
   SCHEDULE = '60 minute'
WHEN system$stream_has_data
AS
   <SQL command>;
```

UPDATE MULTIPLE TABLES

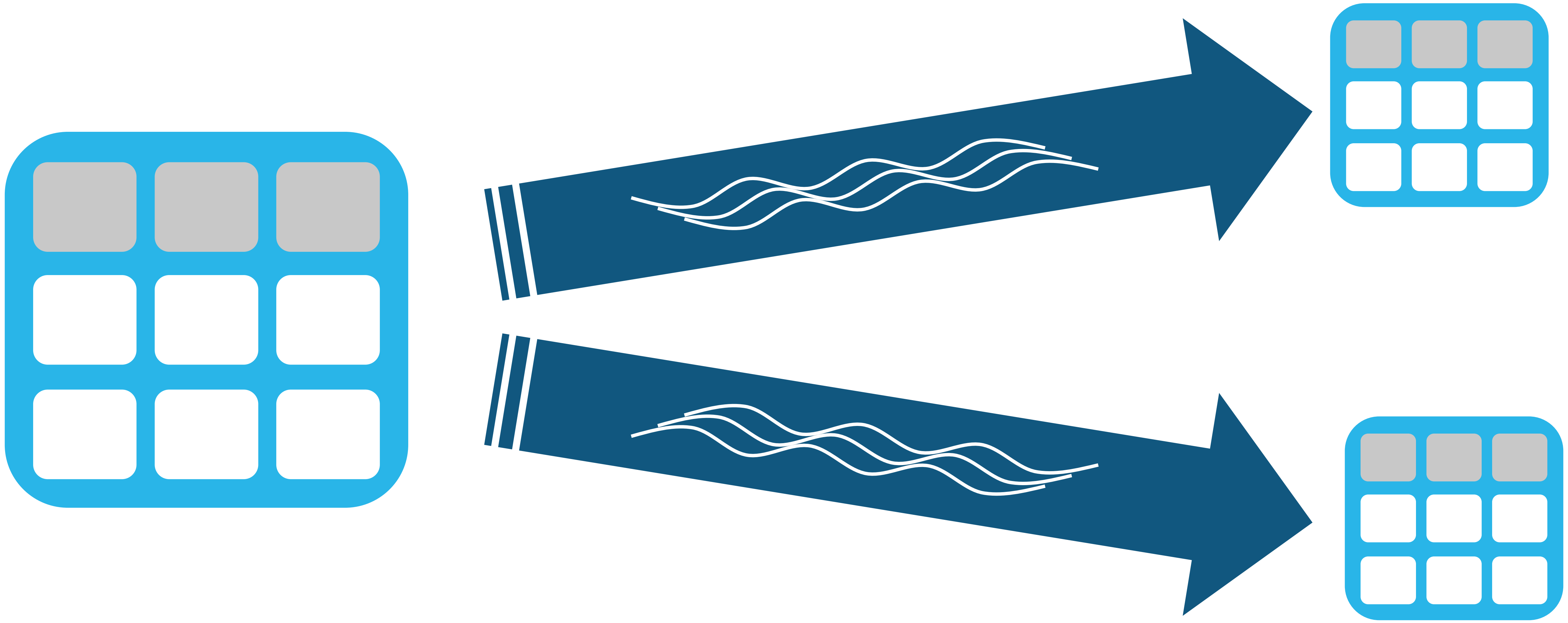
- A single stream can update multiple tables, as long as the stream DML for all target tables is performed within a single transaction

```
BEGIN;
   <update table1 from stream>
   <update table2 from stream>
   . . .
COMMIT;
```



UPDATE MULTIPLE TABLES

- A single table can have multiple streams on it
  - Recommendation: create a separate stream for each consumer of change records



STREAMS AND SHARES

- Consumer accounts can create streams on shares