



Snowflake Advanced

anup.vachali@credit-suisse.com^{16-Nov-2021}©Snowflake 2020-do-not-copy



INTRODUCTIONS

- Name
- Company & Role
- Experience with Snowflake
 - What is your experience with Snowflake?
 - How is your company using Snowflake?
- Tell us something about yourself



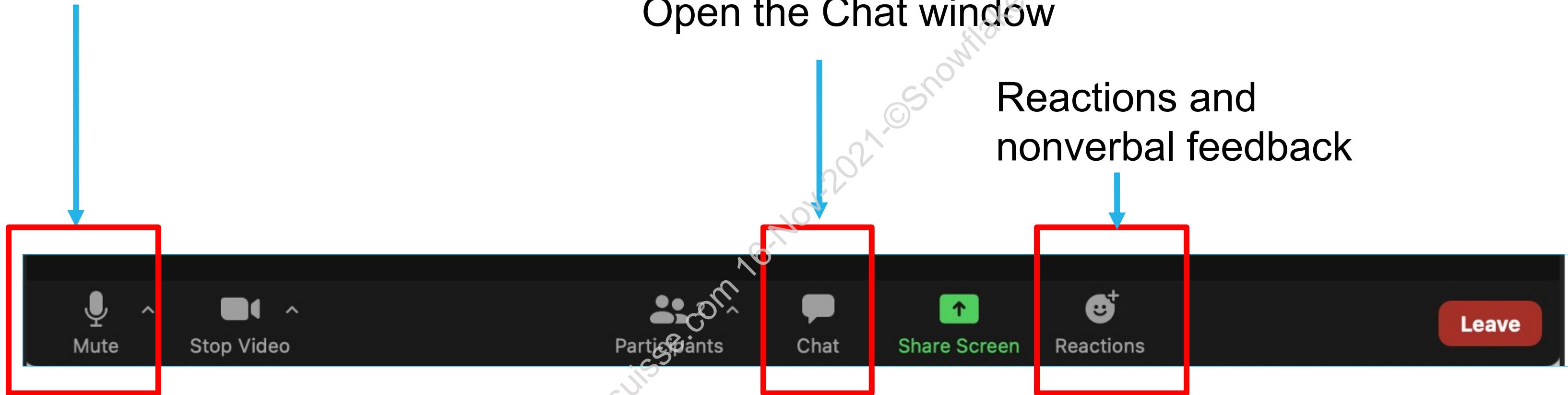
REMOTE ATTENDEE CONTROLS

- Hover over your Zoom interface to display the control panel at the bottom



CONTROL PANEL

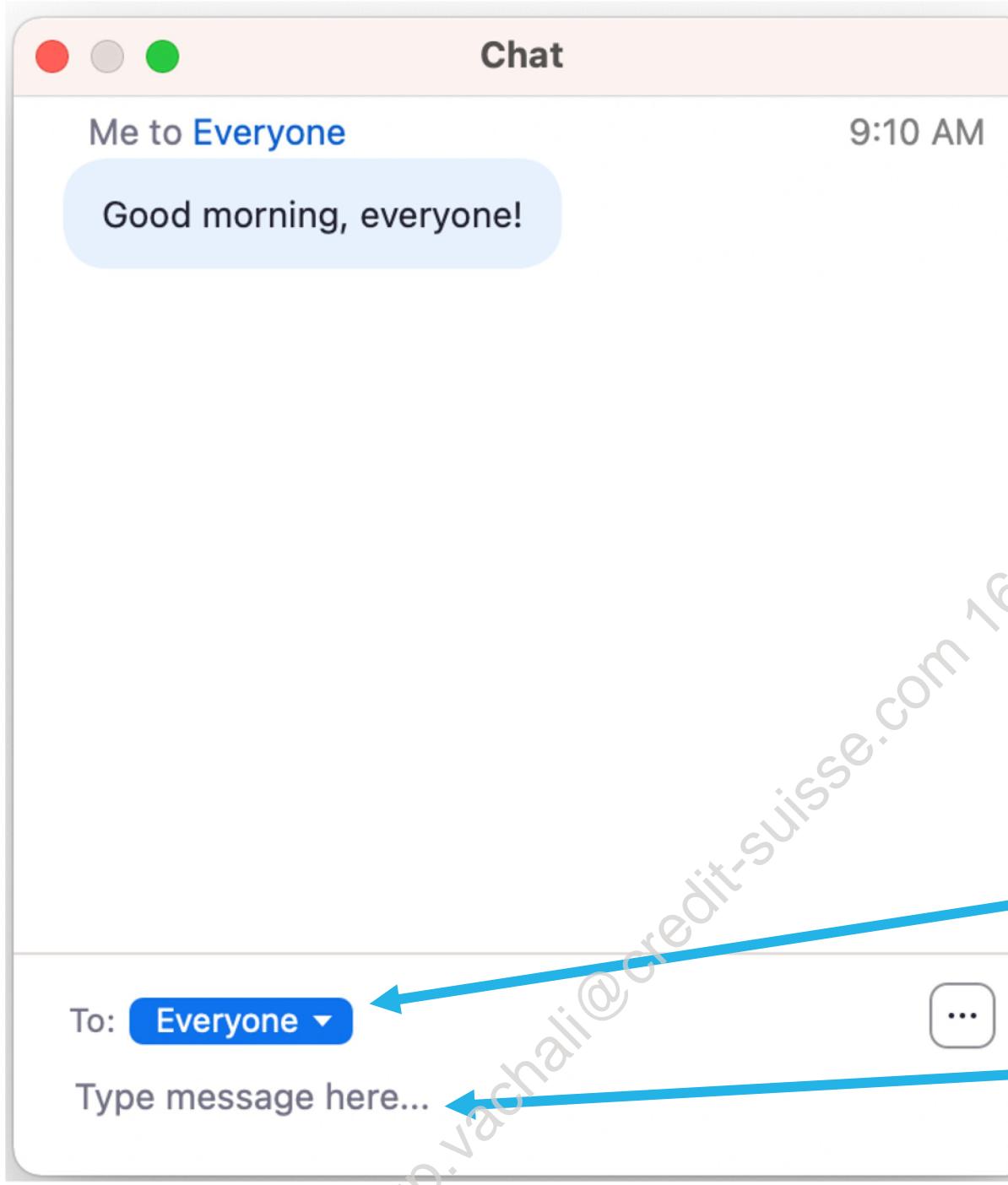
Mute/Unmute



Open the Chat window

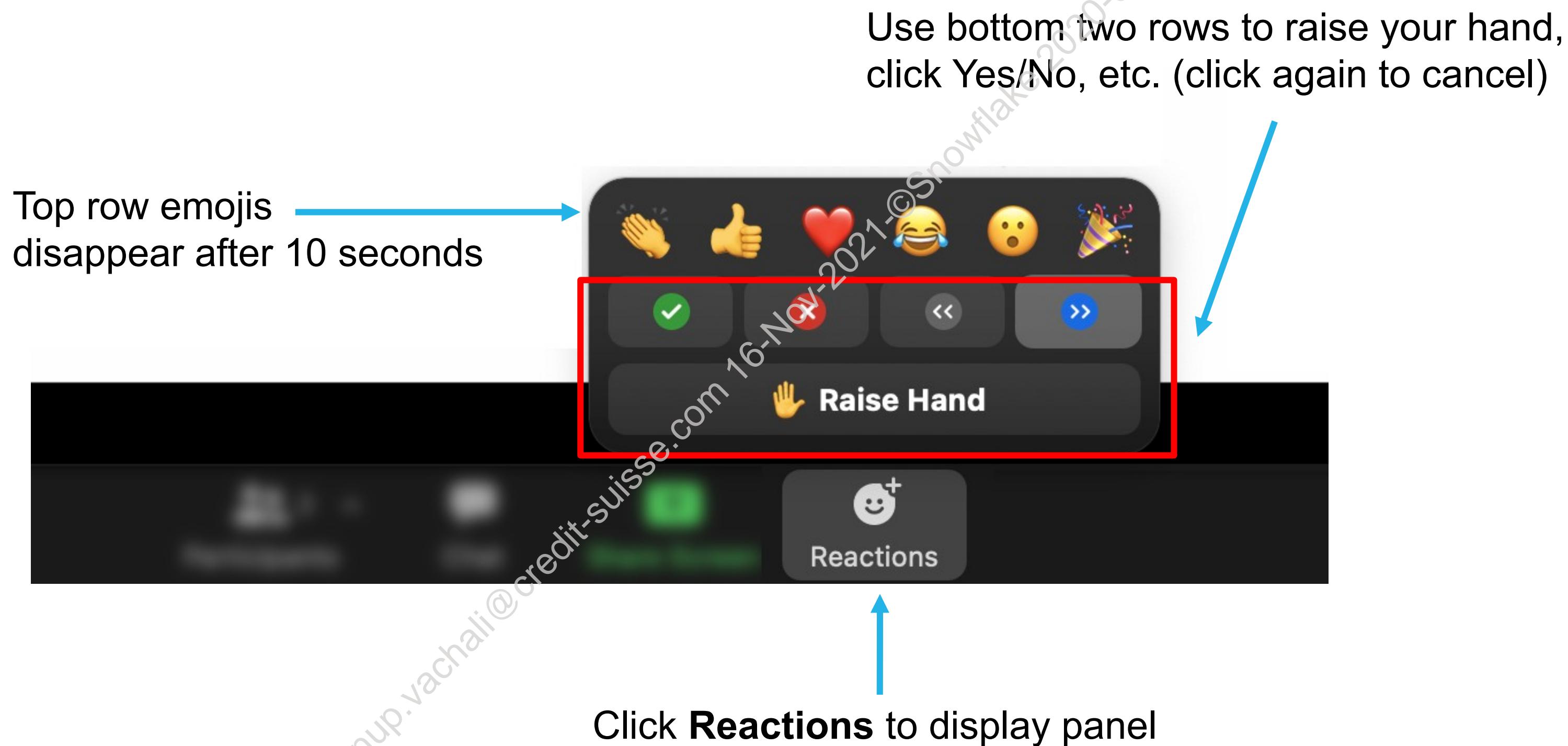
Reactions and
nonverbal feedback

CHAT INTERFACE



1. Use drop-down to choose recipient
2. Click to type your message and press Enter

REACTIONS AND NONVERBAL FEEDBACK



COURSE AGENDA

- Snowflake Architecture
- Build a Data Warehouse
- Work with Data
- Secure Data and Monitor Usage
- Build a Data Pipeline



anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy

SNOWFLAKE ARCHITECTURE

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy

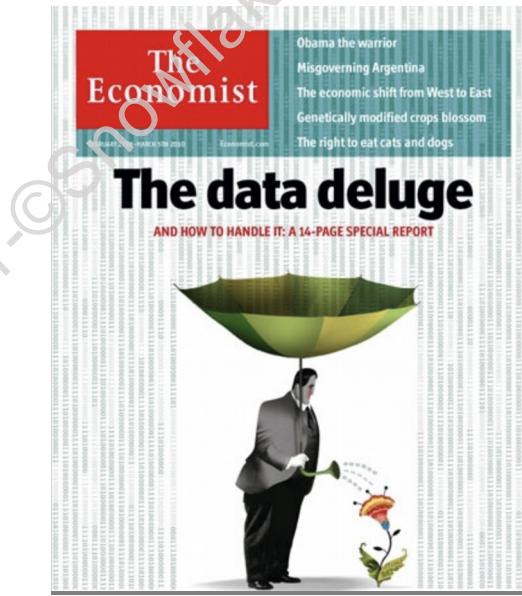
OVERVIEW

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



DATA

THE WORLD'S MOST VALUABLE RESOURCE



“The rapid rise of gathered/analyzed digital data is often core to the holistic success of the fastest growing and most successful companies of our time around the world.”

– Mary Meeker, Bond Capital

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	✓	✓	✗	✓



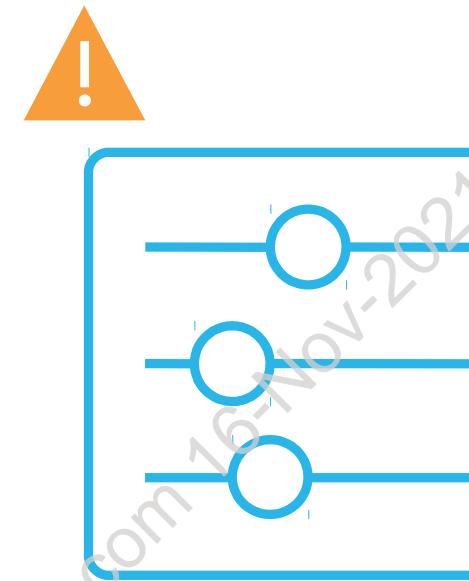
HISTORICAL PERFORMANCE ASSUMPTIONS

Limited, fixed resources



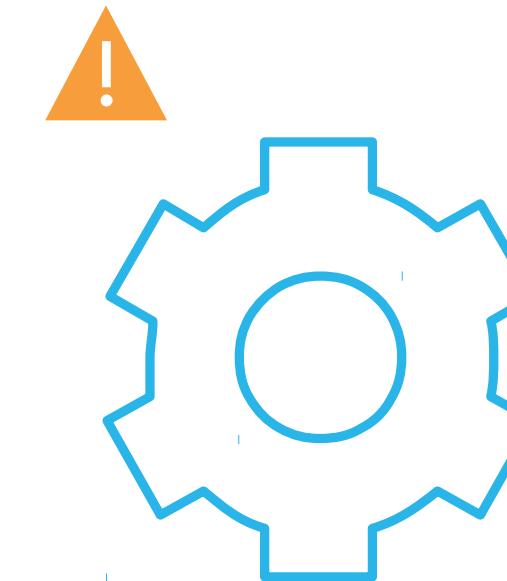
Resources are fixed, so must be sized to the maximum load and protected from overuse

Tune for performance



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

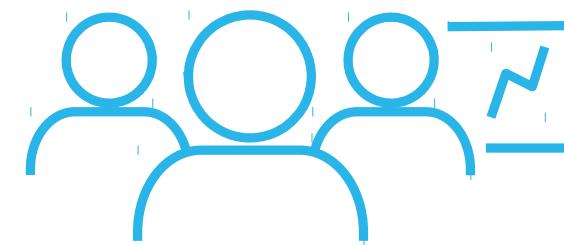
Manual upkeep



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

SNOWFLAKE APPROACH TO PERFORMANCE

Workload Isolation



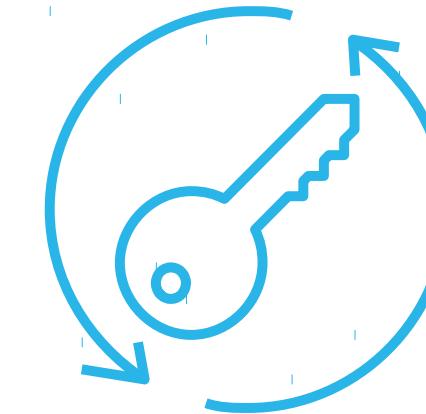
Unlimited compute clusters serve a diverse array of workloads, on-demand

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 fallback
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



ARCHITECTURE REVIEW

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



MODULE AGENDA

- Snowflake's 3-Layer Architecture
- Micro-partitions
- The Snowflake Object Model
- Caching Features Review
- SQL Support

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy

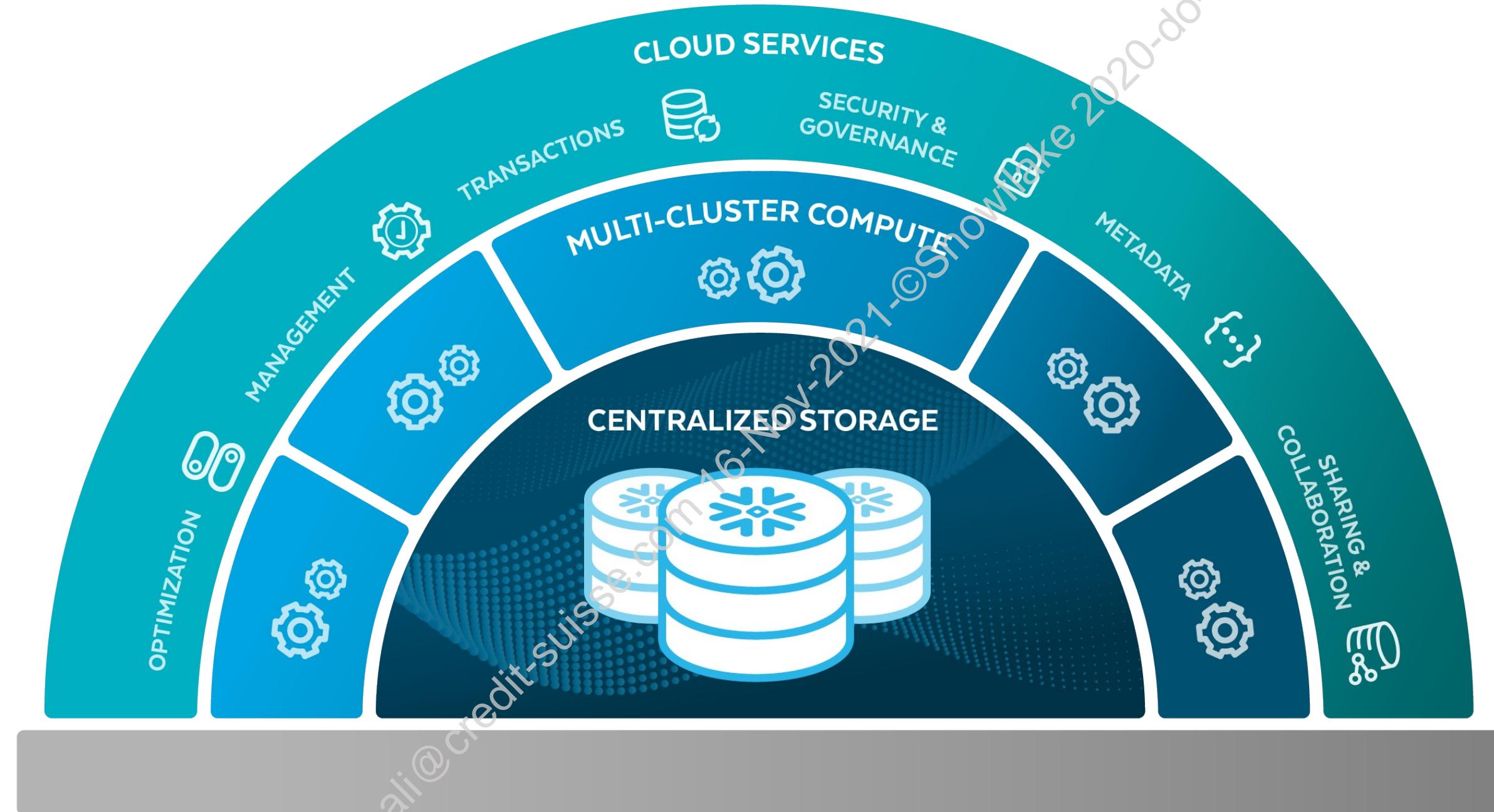


SNOWFLAKE'S 3-LAYER ARCHITECTURE



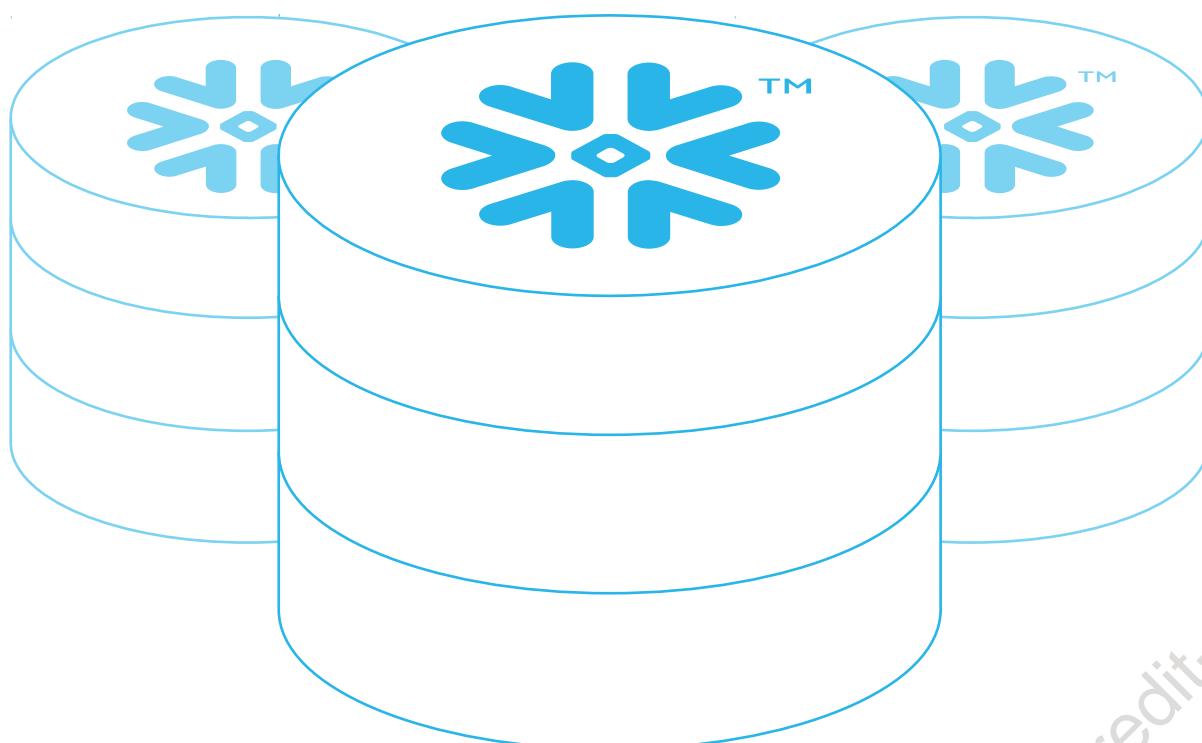
anup.vachali@credit-suisse.com Nov-2021 ©Snowflake 2020-do-not-copy

SNOWFLAKE ARCHITECTURE



STORAGE LAYER

SUPPORT FOR ALL YOUR DATA

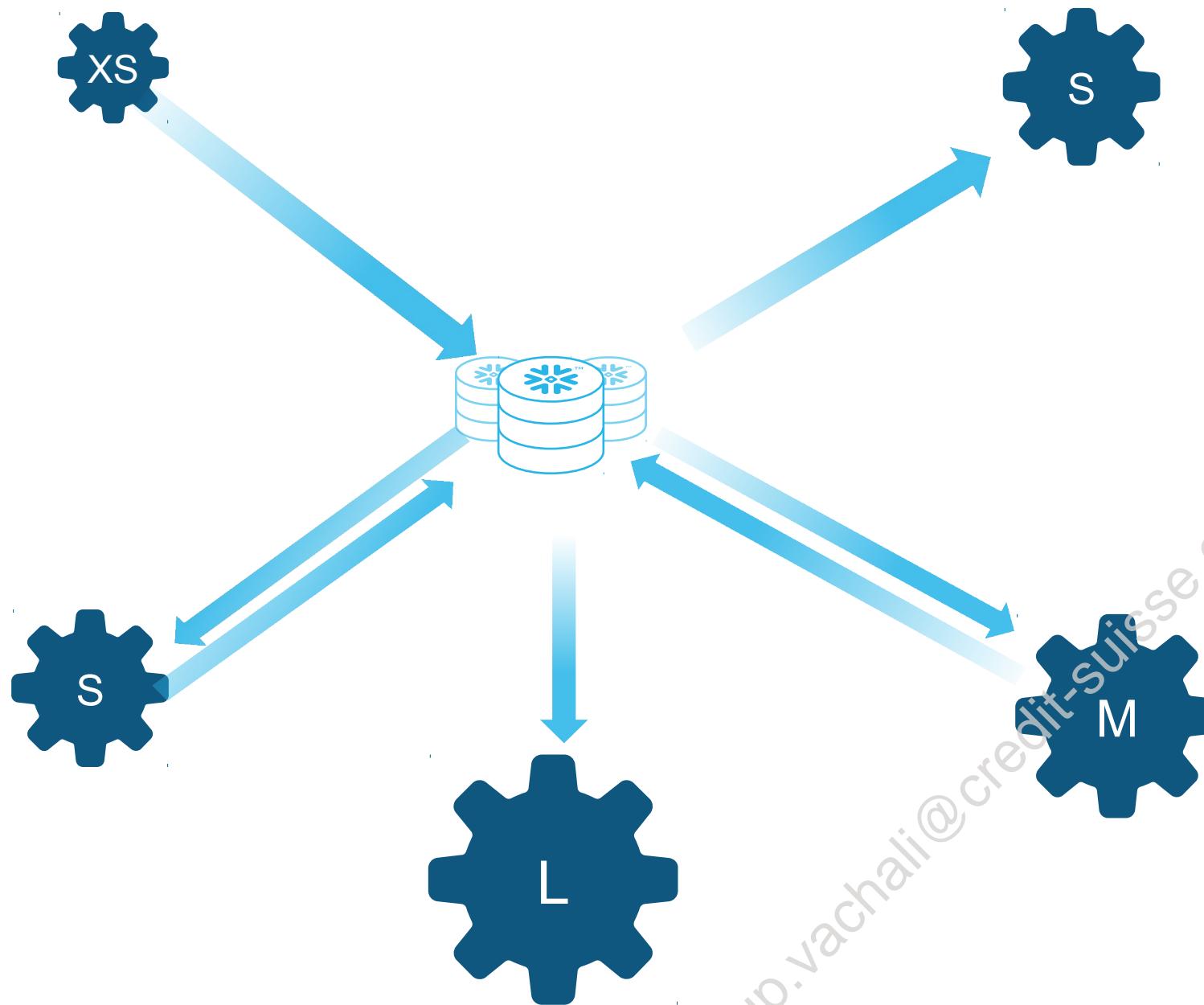


- Where your data is stored
- Infinitely scalable
- You pay only for the storage you use
- Data organized in micro-partitions, in hybrid columnar format
- Micro-partitions are IMMUTABLE



COMPUTE LAYER

SUPPORT FOR ALL YOUR USERS



- Virtual Warehouses are collections of compute resources (RAM, CPU, SSD)
- Warehouses start when work needs to be done, and shut down when idle
- Pay when the warehouse is running
- Each size increase doubles the resources available
- Storage and compute resources scale independently

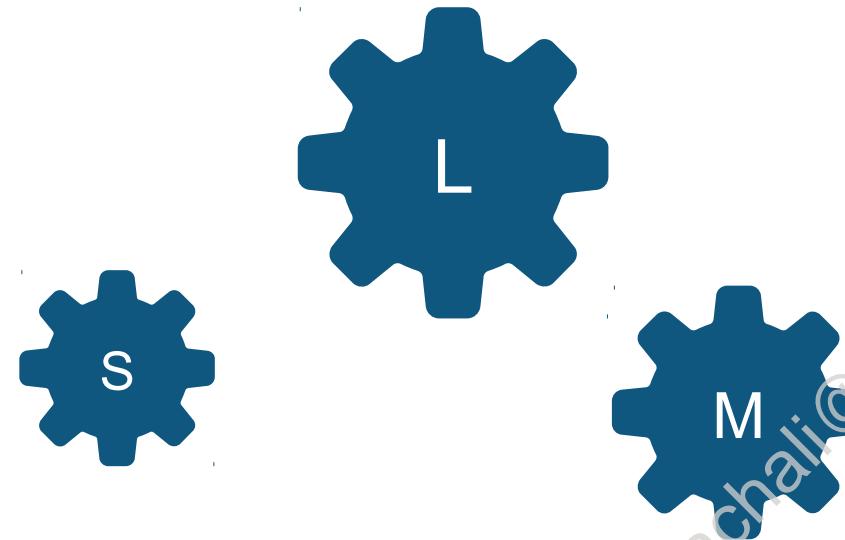


THREE TYPES OF COMPUTE

Compute provisioned by you:

- **Virtual Warehouse Compute**

- Run on virtual warehouses that you create, resume and suspend
- May be auto-resumed or auto-suspended
- Billed per second with a one-minute minimum



Compute provisioned by Snowflake:

- **Compute for Serverless Features**

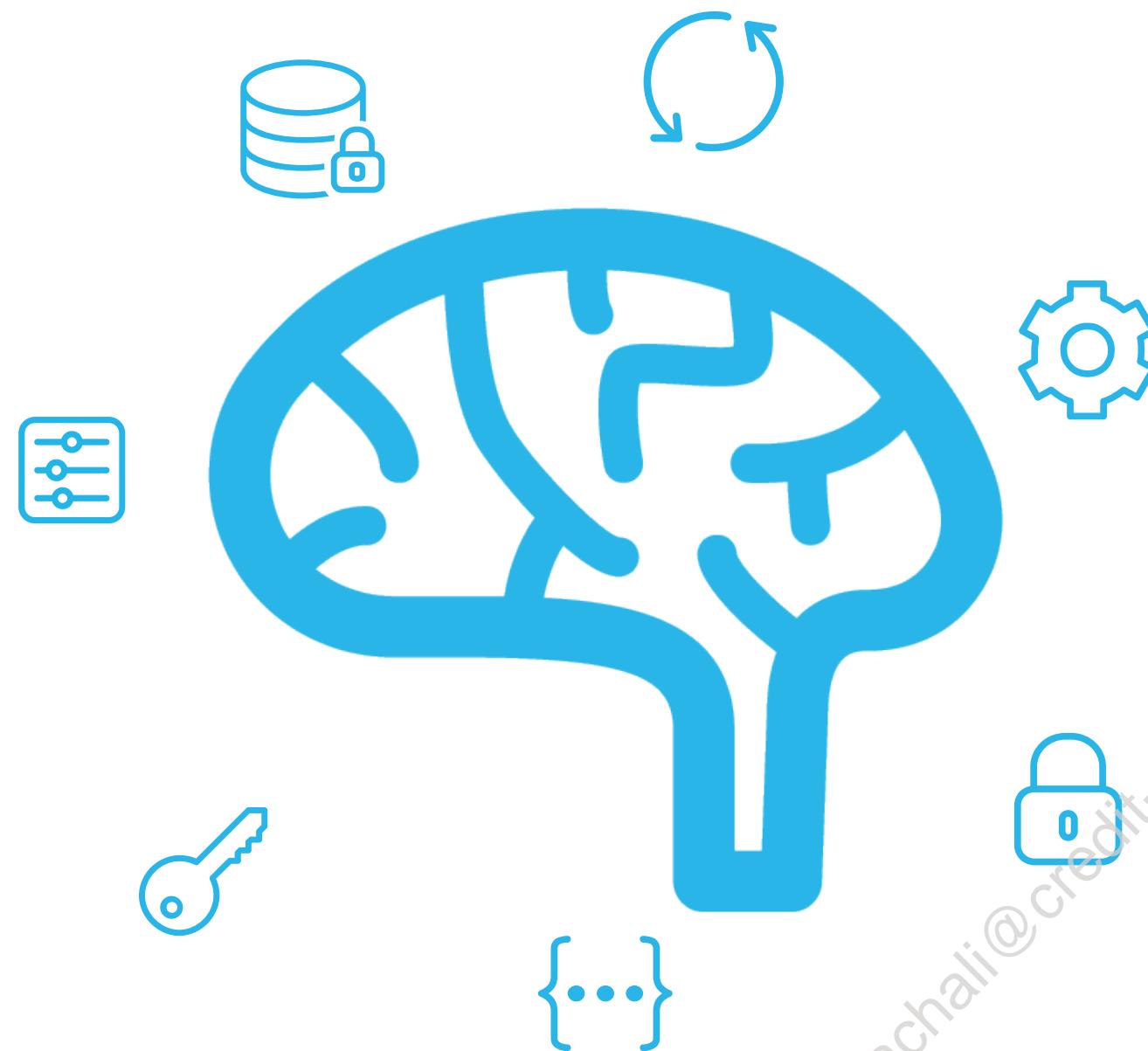
- Billed per second with no minimum
- Used for specific features (for example, Snowpipe and our Clustering service)

- **Cloud Services Compute**

- Used for things like cache lookups and cloning tables
- Billed per second with no minimum
- Only charged for any amount that exceeds 10% of total compute usage

CLOUD SERVICES LAYER

THE BRAINS



Responsible for:

- Security
- Encryption
- Metadata
- Updates and patches
- Query optimization
- Transaction control
- Cache management
- Replication management
- Micro-partitioning



MICRO-PARTITIONS

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



WHAT IS A MICRO-PARTITION?

Input File

ORDER_NUM	LAST_NAME	ITEM_NUM
20211003	Littlefield	80504
20211004	Doan	1000456
20211005	Isakov	300567
20211006	Yoo	154

Load Data



20211003	20211004	20211005	20211006
Littlefield	Doan	Isakov	Yoo
80504	1000456	300567	154

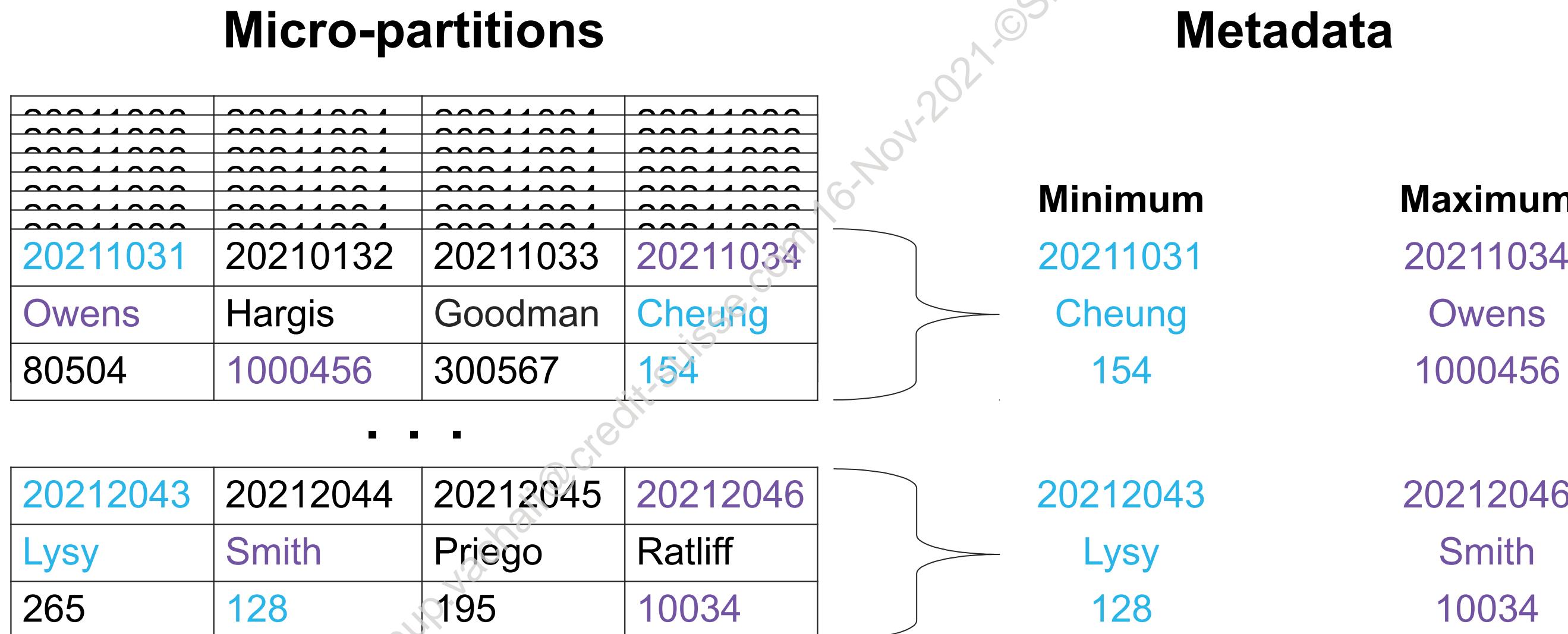
Micro-partition 1

- A small file (~16 MB) that contains some number of complete rows of table data
- As data is loaded, groups of rows are stored in micro-partitions in ingest order
- Inside each micro-partition, data is stored in columnar format
- Micro-partitions are immutable
- Metadata about micro-partitions is stored in the Cloud Services Layer



MICRO-PARTITION METADATA

- Metadata about each micro-partition is handled by the metadata service
- Includes the minimum and maximum value for each column, in each micro-partition



MICRO-PARTITION PRUNING

	Column	Minimum	Maximum
MP1	ORDER_NUM	20211031	20211034
	LAST_NAME	Goodman	Owens
	ITEM_NUM	100096	1000456

MP2	ORDER_NUM	20211035	20211038
	LAST_NAME	Adams	Phillips
	ITEM_NUM	918	22456

MP3	ORDER_NUM	20211039	20211044
	LAST_NAME	Bethel	Zimmerman
	ITEM_NUM	2248	99432

MP4	ORDER_NUM	20211045	20211049
	LAST_NAME	Clark	Rodriquez
	ITEM_NUM	3327	3335

- The optimizer uses WHERE clause and micro-partition metadata to determine which micro-partitions are needed

- For example:

```
SELECT ORDER_NUM FROM orders  
WHERE item_num = 1096;
```



MICRO-PARTITION PRUNING

	Column	Minimum	Maximum
MP1	ORDER_NUM	20211031	20211034
	LAST_NAME	Goodman	Owens
	ITEM_NUM	100096	1000456

MP2	ORDER_NUM	20211035	20211038
	LAST_NAME	Adams	Phillips
	ITEM_NUM	918	22456

MP3	ORDER_NUM	20211039	20211044
	LAST_NAME	Bethel	Zimmerman
	ITEM_NUM	2248	99432

MP4	ORDER_NUM	20211045	20211049
	LAST_NAME	Clark	Rodriquez
	ITEM_NUM	3327	3335

- The optimizer uses WHERE clause and micro-partition metadata to determine which micro-partitions are needed

- For example:

```
SELECT ORDER_NUM FROM orders  
WHERE item_num = 1096;
```

The only micro-partition that **might** contain relevant data is MP2 – which eliminates ("prunes") 75% of the data



MICRO-PARTITION PRUNING

	Column	Minimum	Maximum
MP1	ORDER_NUM	20211031	20211034
	LAST_NAME	Goodman	Owens
	ITEM_NUM	100096	1000456
MP2	ORDER_NUM	20211035	20211038
	LAST_NAME	Adams	Phillips
	ITEM_NUM	918	22456
MP3	ORDER_NUM	20211039	20211044
	LAST_NAME	Bethel	Zimmerman
	ITEM_NUM	2248	99432
MP4	ORDER_NUM	20211045	20211049
	LAST_NAME	Clark	Rodriquez
	ITEM_NUM	3327	3335

- The optimizer uses WHERE clause and micro-partition metadata to determine which micro-partitions are needed
- For example:

```
SELECT ORDER_NUM FROM orders
WHERE item_num = 1096;
```
- The only micro-partition that **might** contain relevant data is MP2 – which eliminates ("prunes") 75% of the data
- Only two columns from MP2 need to be read and processed to satisfy the query

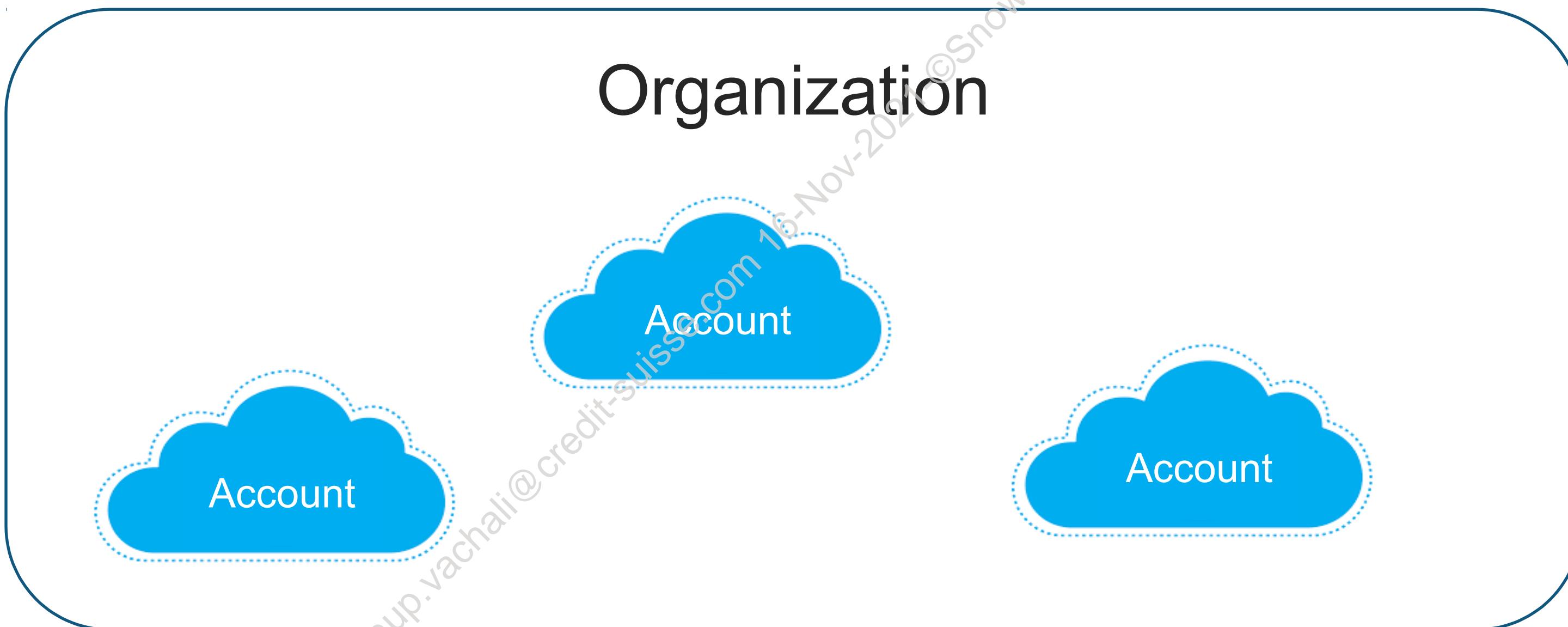


THE SNOWFLAKE OBJECT MODEL

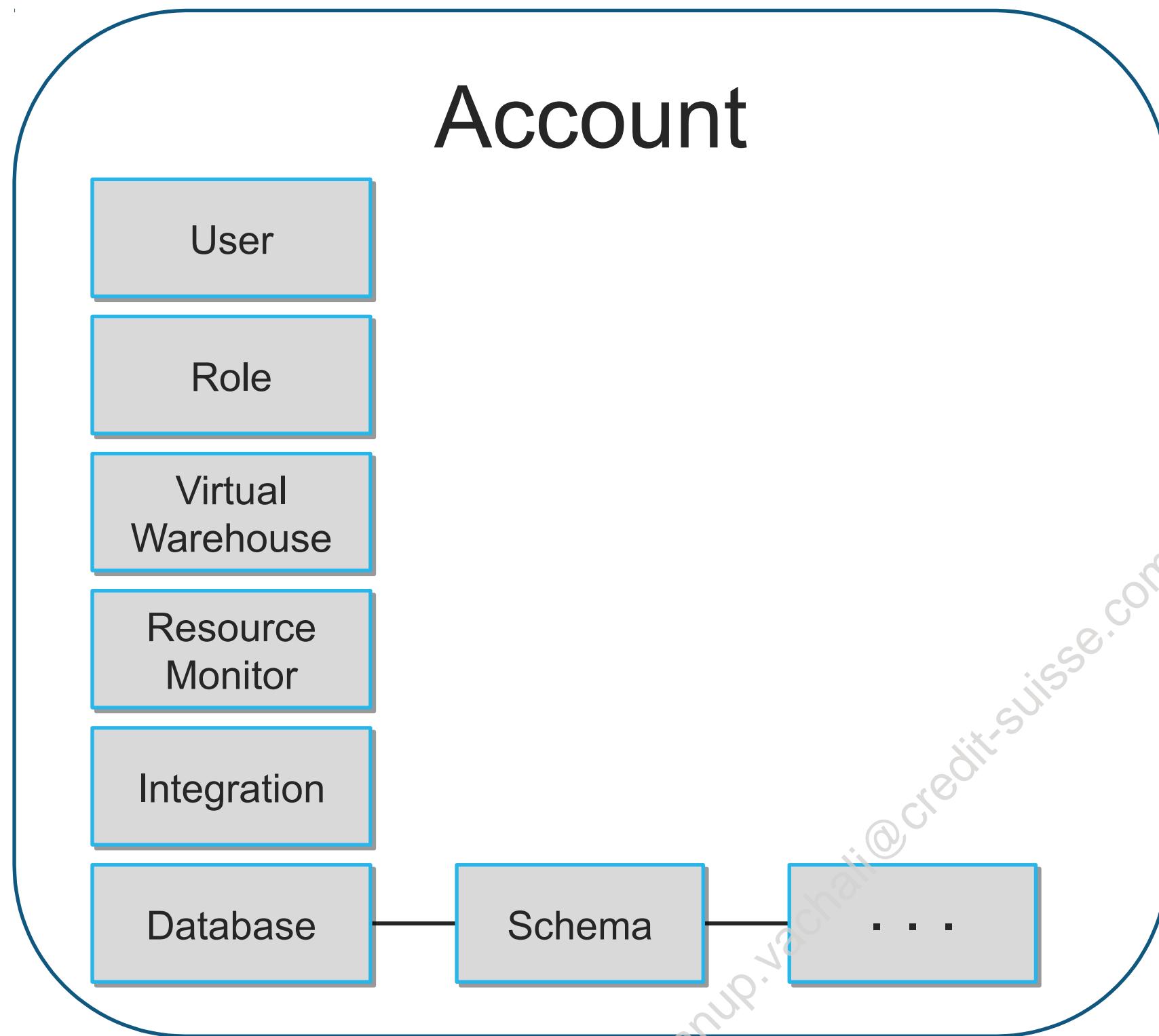


ORGANIZATIONS AND ACCOUNTS

- Each Snowflake customer has at least one account
- An organization is a collection of accounts for the same customer



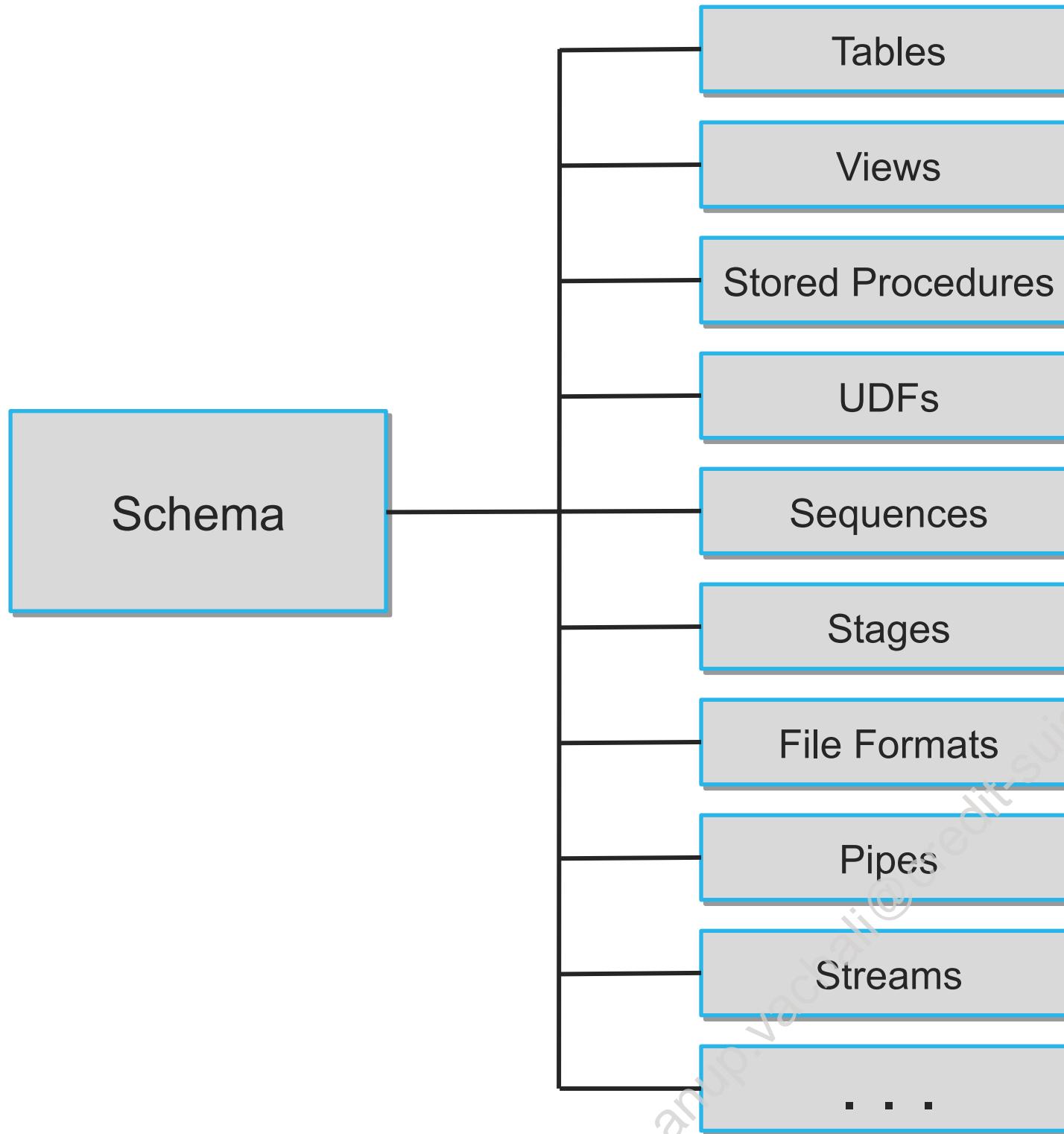
ACCOUNT OBJECTS



- All Snowflake objects reside within an account
- An account can have more than one of each of these types of objects
- The objects User, Role, Warehouse, Resource Monitor, Integration, and Database are at the account level
- All other objects are contained in Schemas, which are inside Databases



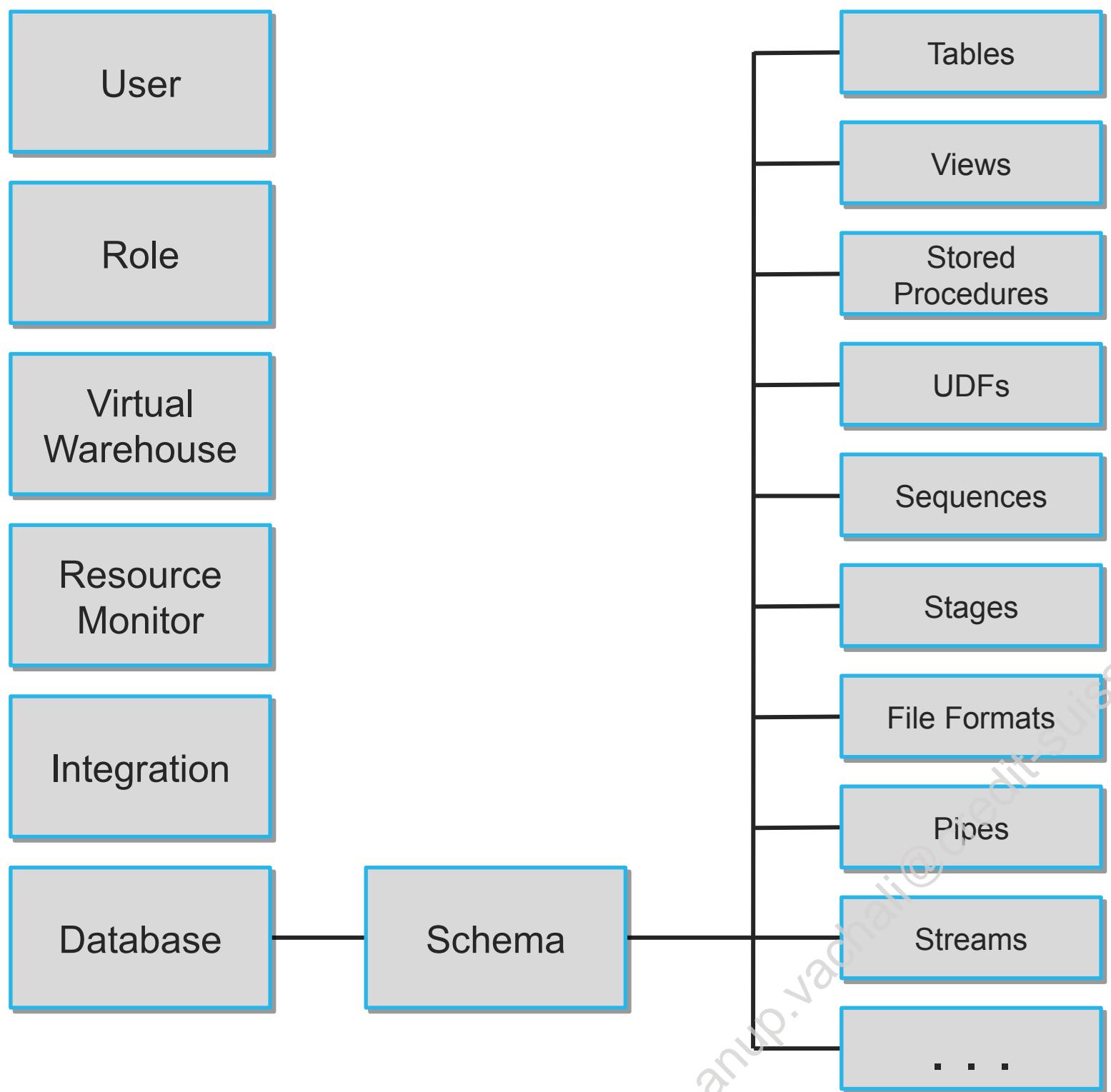
DATABASE AND SCHEMA OBJECTS



- Each database has two built-in schemas
 - PUBLIC
 - INFORMATION_SCHEMA
- Databases typically also contain custom schemas created by users
- Schemas are organizational objects
- Schemas contain objects like tables, views, stored procedures, and file formats



HIERARCHY OF OBJECTS



- Each object is individually securable
- Sample privileges:
 - Create warehouse
 - Insert into tables
 - Use file formats



CACHING FEATURES REVIEW

anup.vachali@credit-suisse.com 16 Nov 2021 - ©Snowflake 2020-do-not-copy



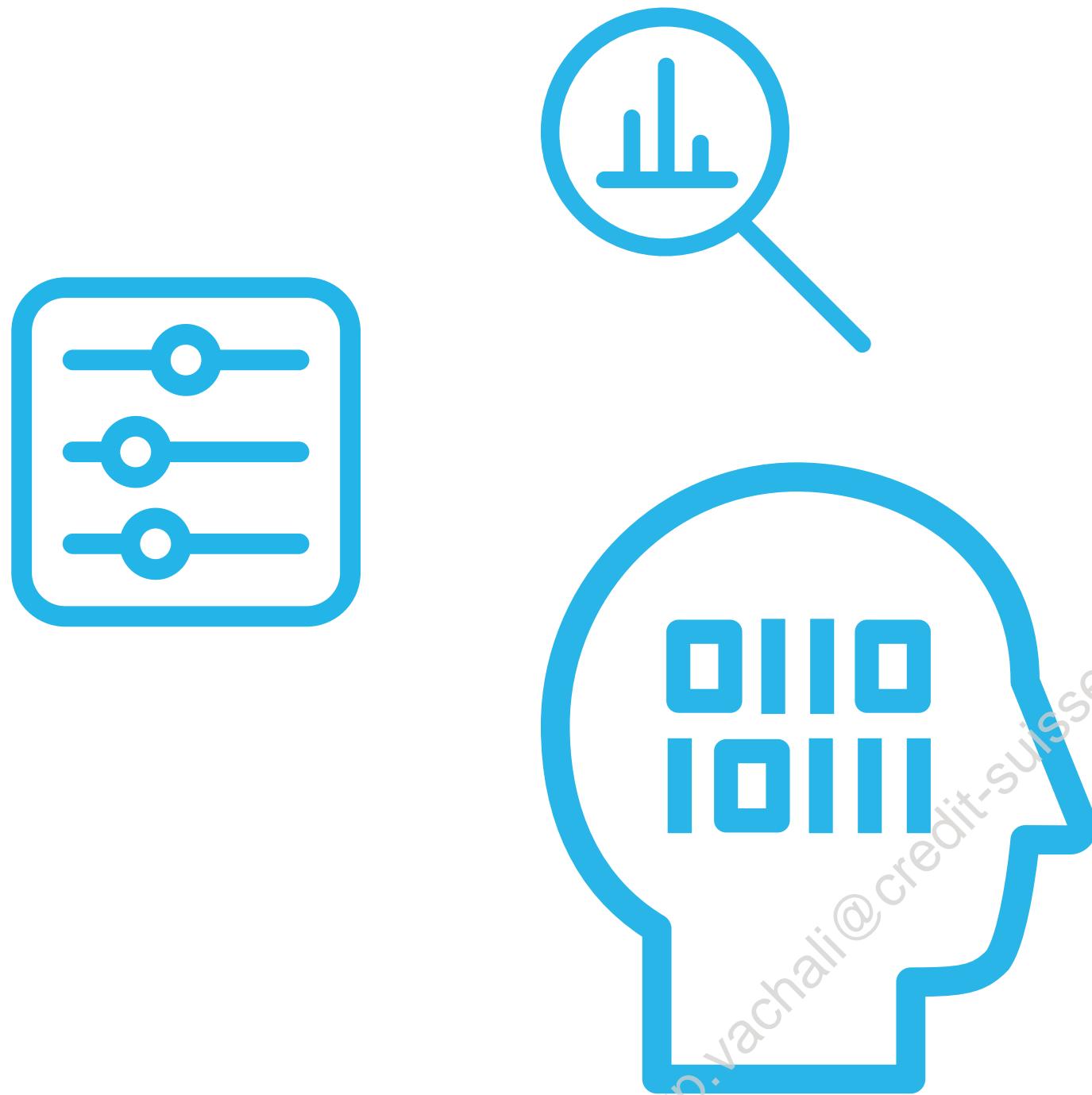
THREE LEVELS OF CACHING

- Metadata cache
- Query Result Cache
- Data Cache



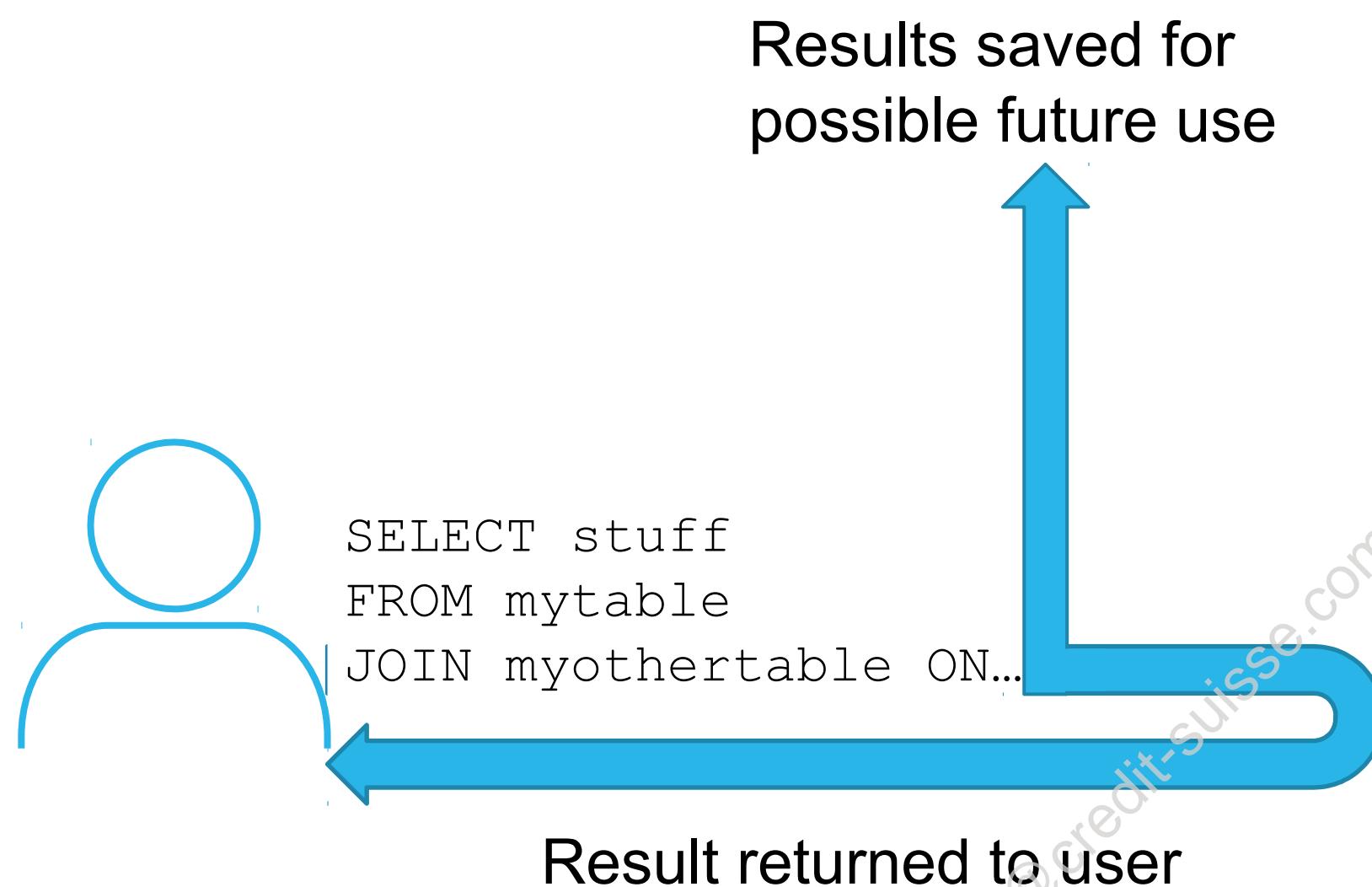
anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy

METADATA CACHE



- Used by the SQL optimizer for partition pruning
- Some queries can be answered completely by metadata

QUERY RESULT CACHE



- Results of each query are cached
- Can be used by anyone, as long as the:
 - Query is identical
 - Query result is deterministic
 - Results have not changed
 - User has all needed privileges to the data
- Access result set with `RESULT_SCAN` table function

DATA (WAREHOUSE) CACHE



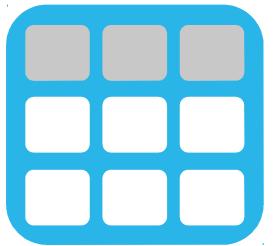
- Data used by a query is cached on the virtual warehouse
- Used by later queries if the data has not changed
 - Reduced remote I/O improves performance
- Flushed out in a Least Recently Used (LRU) manner
- Optimizer attempts to assign tasks to nodes that have required data cached

SQL SUPPORT

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



TABLE TYPES



PERMANENT

- Persist until dropped
- Designed for data that requires the highest level of data protection and recovery
- Default table type

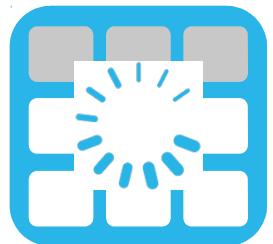
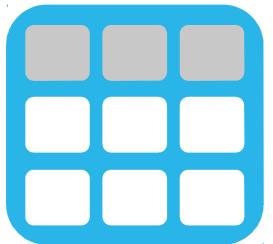
Time Travel

Up to 90 days
with Enterprise

Failsafe



TABLE TYPES



PERMANENT

- Persist until dropped
- Designed for data that requires the highest level of data protection and recovery
- Default table type

TEMPORARY

- Persist and tied to a session (think single user)
- Used for transitory data (for example, ETL/ELT)

Time Travel

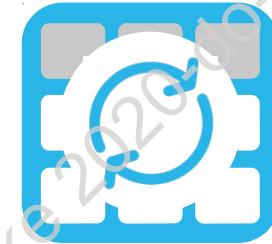
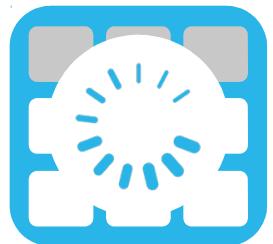
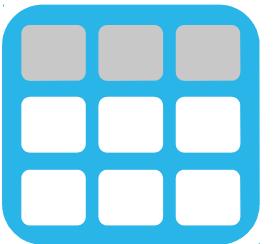
Up to 90 days with Enterprise

0 or 1 days

Failsafe



TABLE TYPES



PERMANENT

- Persist until dropped
- Designed for data that requires the highest level of data protection and recovery
- Default table type

TEMPORARY

- Persist and tied to a session (think single user)
- Used for transitory data (for example, ETL/ELT)

TRANSIENT*

- Persist until dropped
- Multiple user
- Used for data that needs to persist, but does not need the same level of data retention as a permanent table

Time Travel

Up to 90 days with Enterprise

0 or 1 days

0 or 1 days

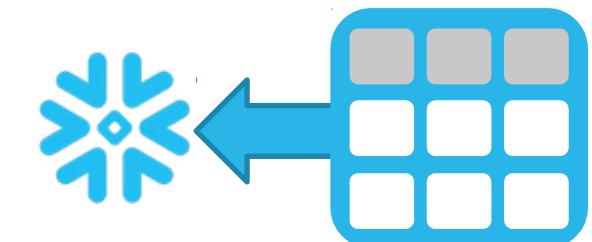
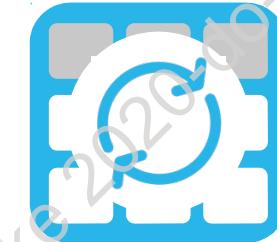
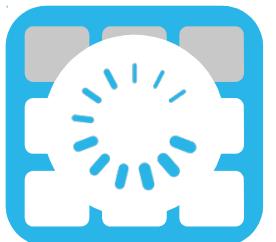
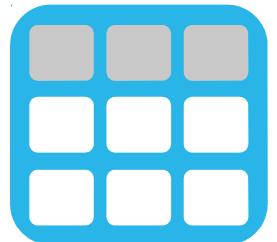
Failsafe



*Transient applicable to Database, Schema and Table



TABLE TYPES



PERMANENT

- Persist until dropped
- Designed for data that requires the highest level of data protection and recovery
- Default table type

TEMPORARY

- Persist and tied to a session (think single user)
- Used for transitory data (for example, ETL/ELT)

TRANSIENT*

- Persist until dropped
- Multiple user
- Used for data that needs to persist, but does not need the same level of data retention as a permanent table

EXTERNAL

- Persist until removed
- Snowflake “over” an external data lake
- Data accessed via an external stage
- Read-only

Time Travel

Up to 90 days with Enterprise

0 or 1 days

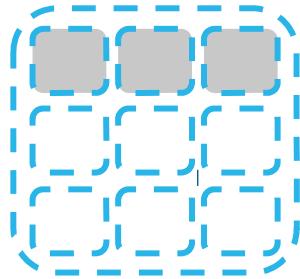
0 or 1 days

x

Failsafe



VIEW TYPES

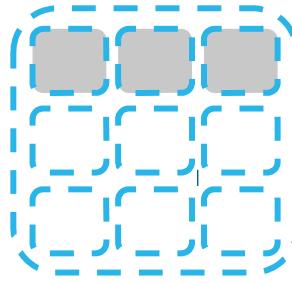


Standard View

- Default view type
- Named definition of a query--
SELECT statement
- Executes as owning role
- Underlying DDL available to any
role with access to the view.

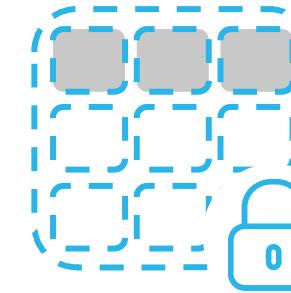


VIEW TYPES



Standard View

- Default view type
- Named definition of a query-- SELECT statement
- Executes as owning role
- Underlying DDL available to any role with access to the view.

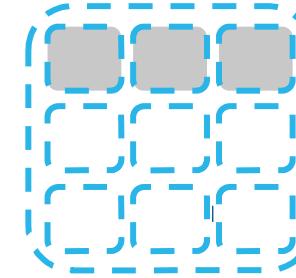


Secure View

- Definition and details only visible to authorized users
- Executes as owning role
- Snowflake query optimizer bypasses optimizations used for regular views

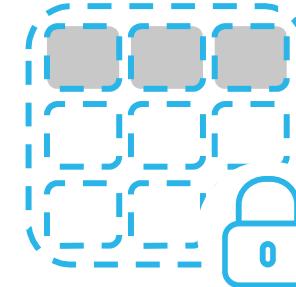


VIEW TYPES



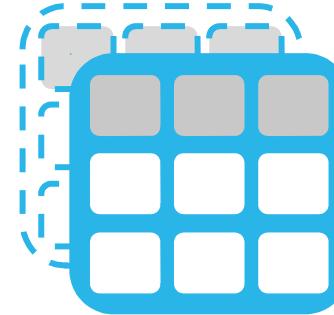
Standard View

- Default view type
- Named definition of a query-- SELECT statement
- Executes as owning role
- Underlying DDL available to any role with access to the view.



Secure View

- Definition and details only visible to authorized users
- Executes as owning role
- Snowflake query optimizer bypasses optimizations used for regular views



Materialized View

- Behaves more like a table
- Results of underlying query are stored
- Auto-refreshed
- Secure Materialized View is also supported



NUMERIC DATA TYPES

Type	Notes
NUMBER, DECIMAL, NUMERIC	Synonymous with NUMBER. Default precision/scale are (38,0)
INT, INTEGER, BIGINT, SMALLINT	Synonymous with NUMBER except precision and scale cannot be specified
DOUBLE, DOUBLE PRECISION, FLOAT, FLOAT4, FLOAT8, REAL	Stored as DOUBLE.



STRING, BINARY, AND LOGICAL DATA TYPES

Type	Notes
VARCHAR, STRING, TEXT	Synonymous with VARCHAR. Default (and maximum) is 16,777,216 bytes.
CHAR, CHARACTER	Defaults to VARCHAR(1); maximum is 16,777,216 bytes.
BINARY, VARBINARY	Synonymous with BINARY
BOOLEAN	BOOLEAN – supported only for accounts provisioned after 1/25/2016



DATE AND TIME DATA TYPES

Type	Notes
DATE	DATE
TIME	TIME
TIMESTAMP	Alias for one of the TIMESTAMP variations, set as a parameter. TIMESTAMP_NTZ by default.
TIMESTAMP_LTZ	TIMESTAMP with local time zone. Time zone, if provided, is not stored.
TIMESTAMP_NTZ, DATETIME	TIMESTAMP with no time zone. Time zone, if provided, is not stored.
TIMESTAMP_TZ	TIMESTAMP with time zone.



OTHER DATA TYPES

Type	Notes
VARIANT, OBJECT, ARRAY	Used for storing semi-structured data
GEOGRAPHY	Follows the WGS 84 standard



UNSUPPORTED DATA TYPES

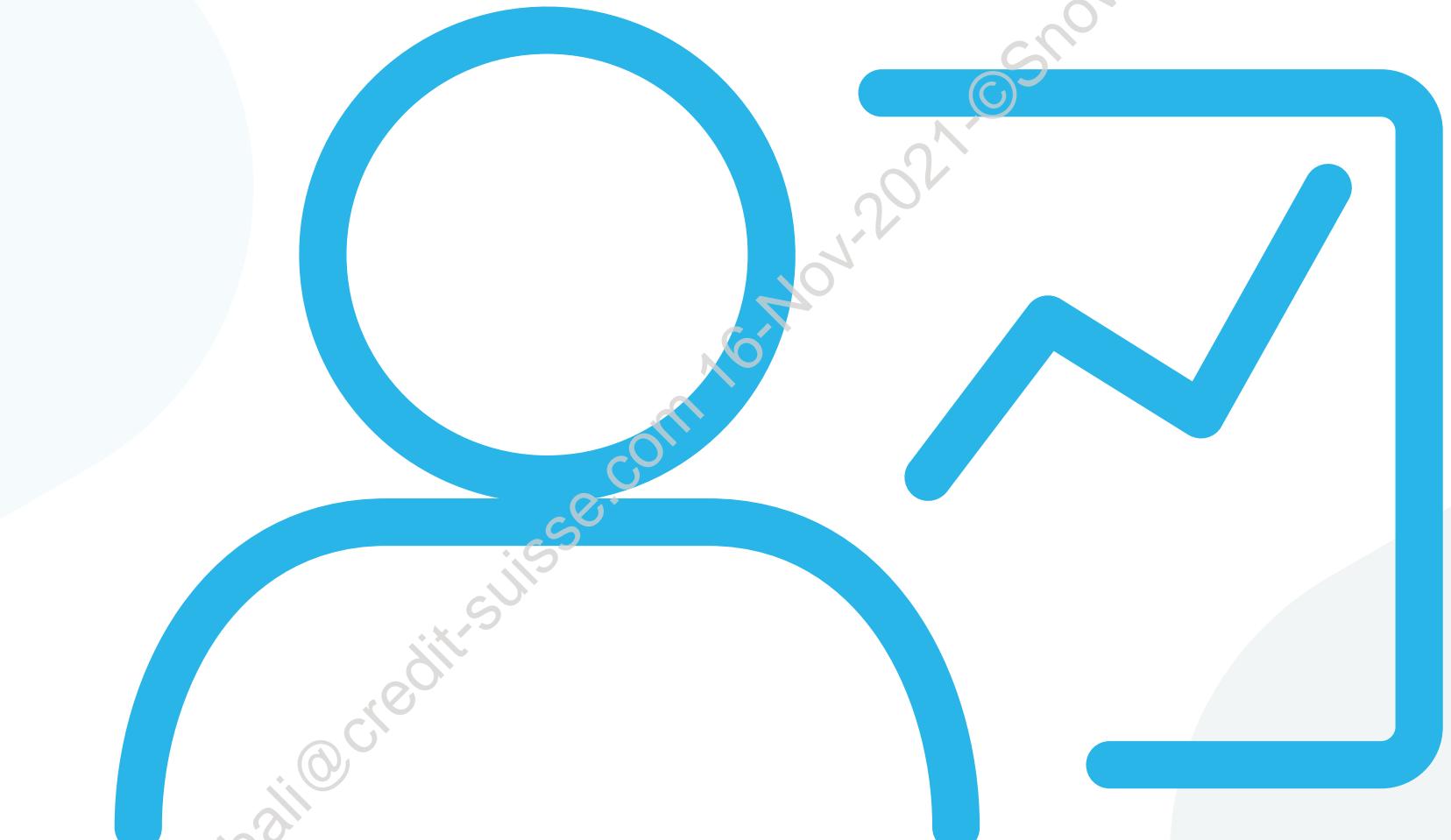
Type	Notes
BLOB	Use BINARY instead; maximum of 8,388,608 bytes
CLOB	Use VARCHAR instead; maximum of 16,777,216 bytes
ENUM	Not supported
User-defined data types	Not supported



INSTRUCTOR DEMO

The Snowflake User Interface

10 minutes



anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



LAB EXERCISE: 1

Take a Quick Test Drive

25 minutes

- Explore context
- Create objects
- Run queries on sample data

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



TIME TRAVEL AND CLONING

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



MODULE AGENDA

- Table Metadata
- Time Travel
- Zero-Copy Cloning
- Agile Development

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



TABLE METADATA

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



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 ...`

Representation of Table Metadata

ID	Name
	mytable

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



TABLE VERSIONS

`COPY INTO mytable ...`

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

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	MP3
7	Pierre	
4	Janet	
5	Jack	MP4
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

anup.vachali@credit-suisse.com 16-Nov-2021-©Snowflake 2020-do-not-copy



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



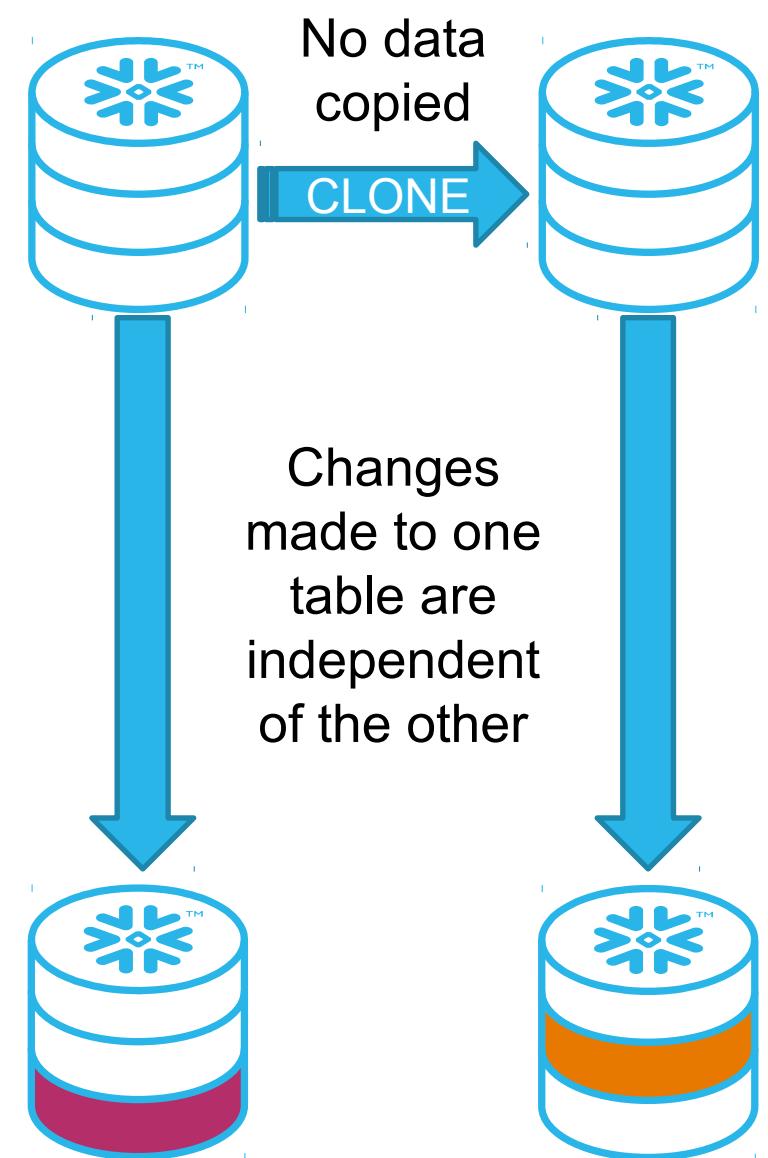
ZERO-COPY CLONING

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



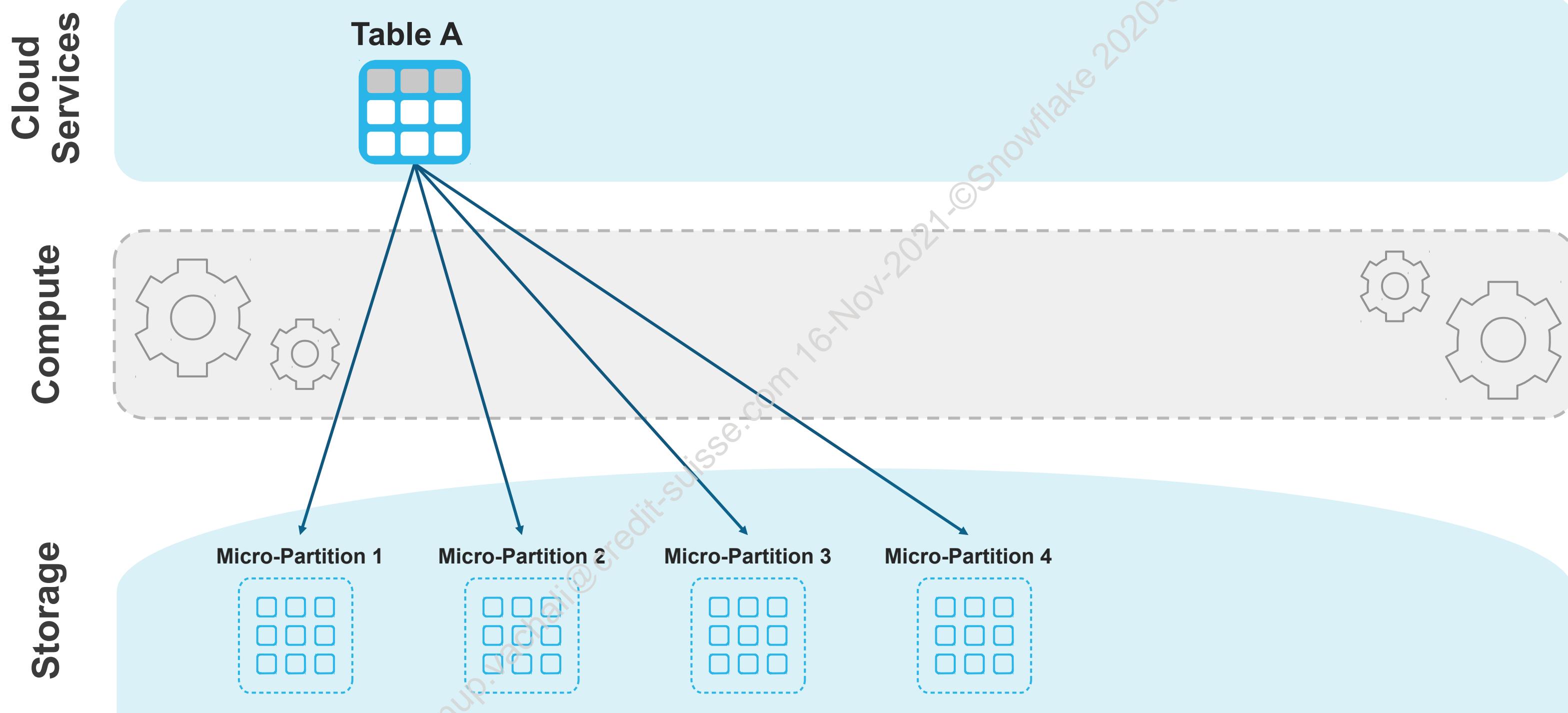
ZERO-COPY CLONING

- Quickly take a “snapshot” of any table, schema, or database
 - Spin up Dev and Test/QA environments
 - Create data backups at a point in time
- No additional storage costs incurred until changes are made to the original or clone
- Clones can be cloned as many times as needed
- At the instant the clone is created:
 - All micro-partitions in both tables are fully shared
 - Storage associated with the micro-partitions is owned by the oldest table in the clone group



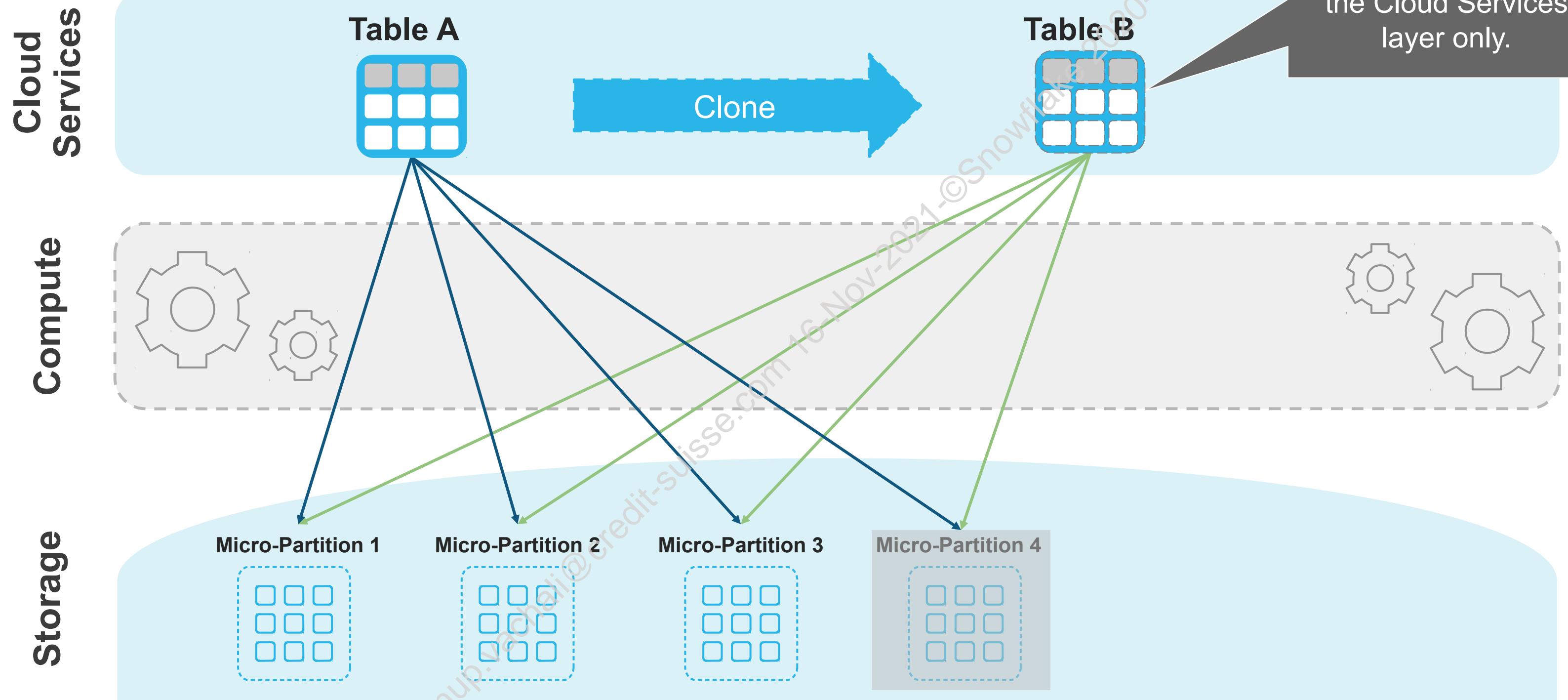
CLONING EXAMPLE

BEFORE CLONE



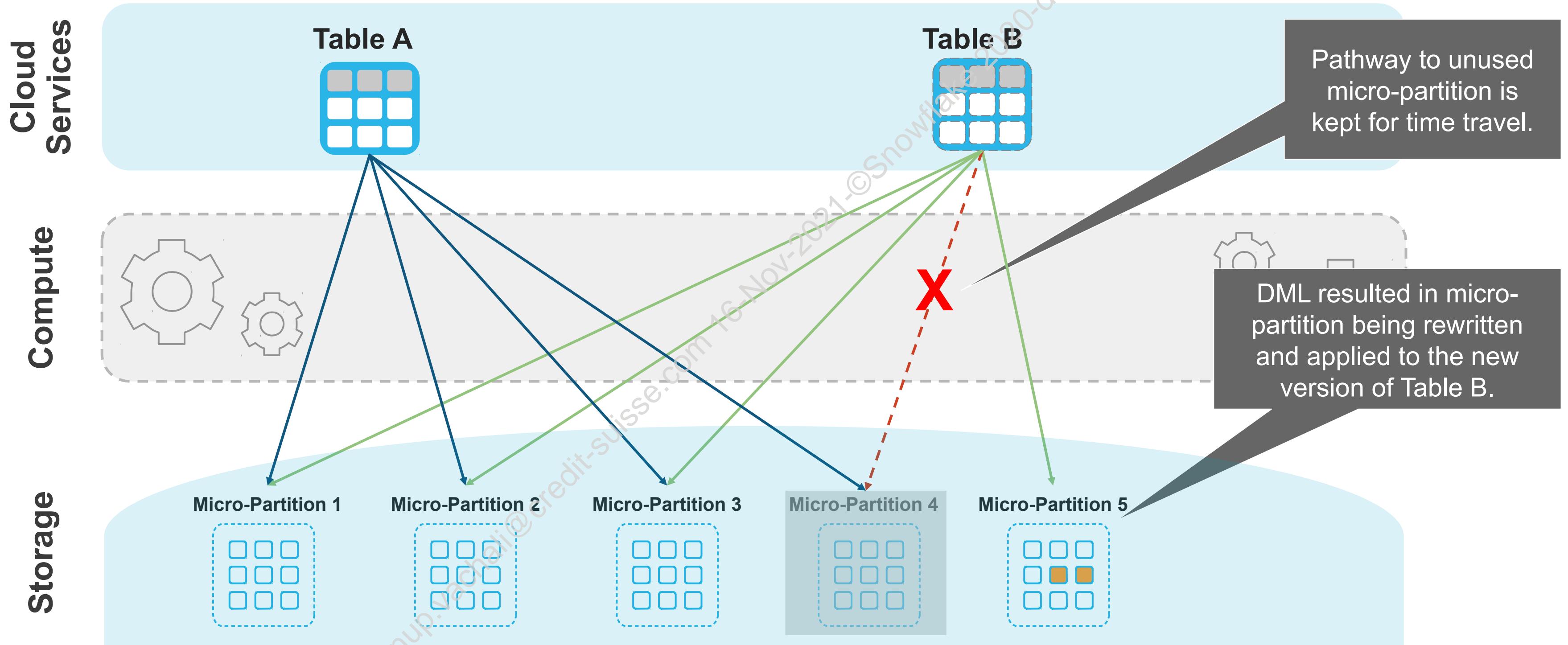
CLONING EXAMPLE

TABLE A IS CLONED TO TABLE B



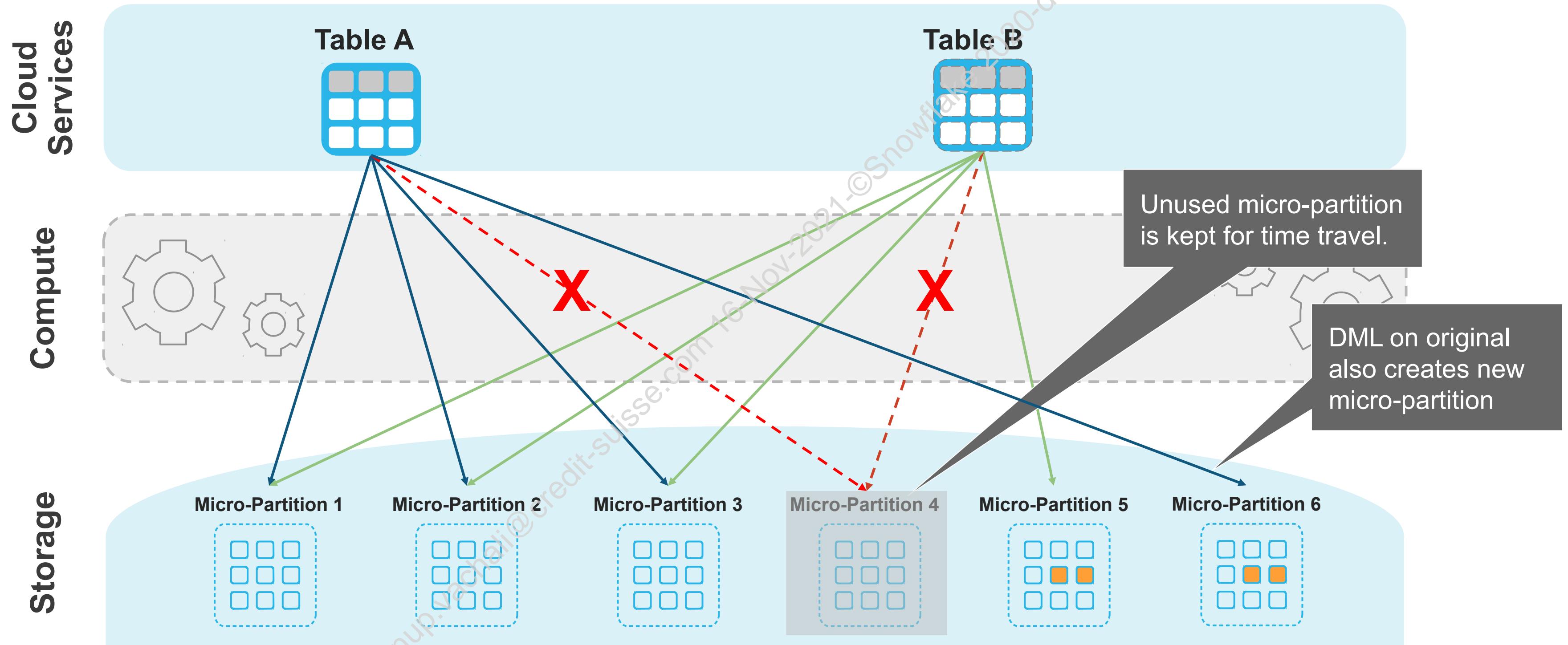
CLONING EXAMPLE

TABLE B IS CHANGED



CLONING EXAMPLE

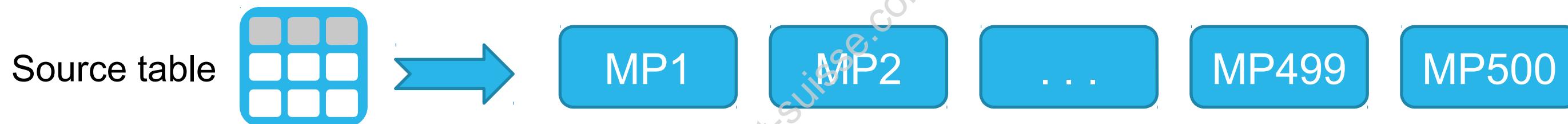
NEITHER TABLE NEEDS MICRO-PARTITION 4



DML DURING CLONING

Avoid DML during cloning if Time Travel is set to 0 on the source table

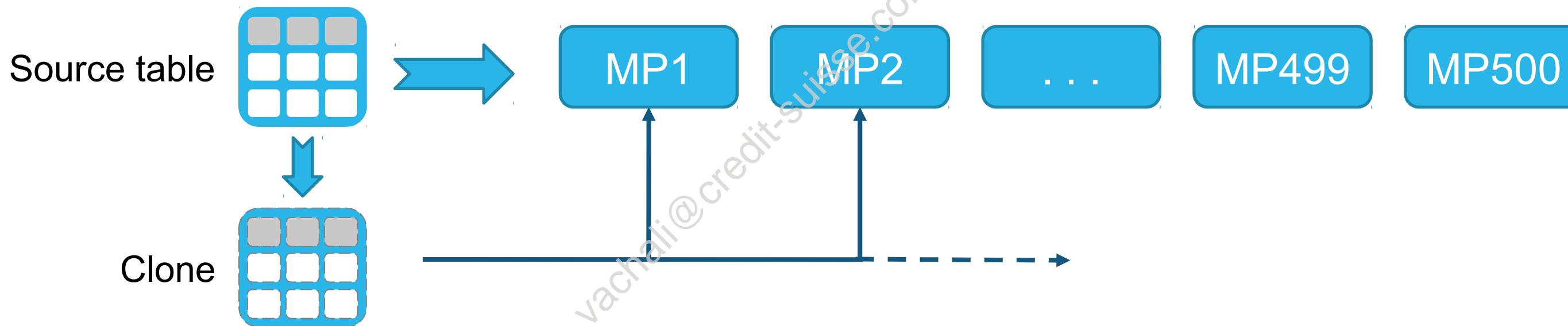
1. Source table contains micro-partitions 1 through 500



DML DURING CLONING

Avoid DML during cloning if Time Travel is set to 0 on the source table

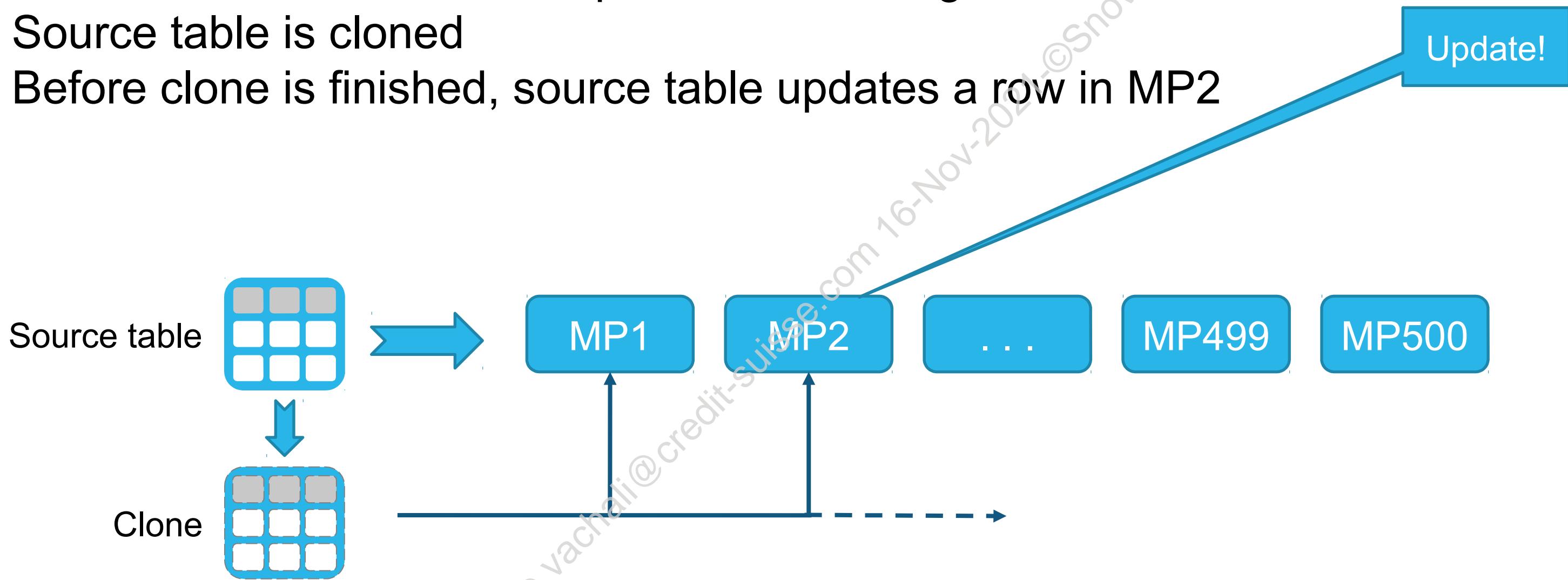
1. Source table contains micro-partitions 1 through 500
2. Source table is cloned



DML DURING CLONING

Avoid DML during cloning if Time Travel is set to 0 on the source table

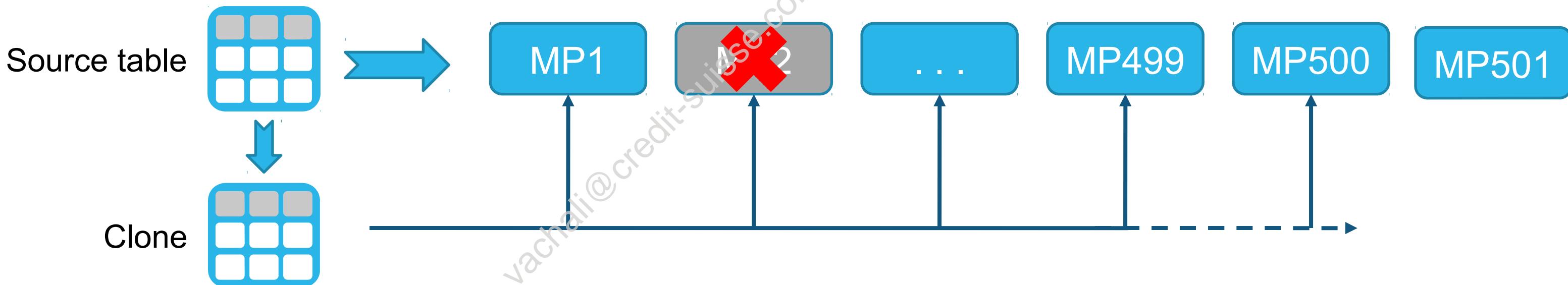
1. Source table contains micro-partitions 1 through 500
2. Source table is cloned
3. Before clone is finished, source table updates a row in MP2



DML DURING CLONING

Avoid DML during cloning if Time Travel is set to 0 on the source table

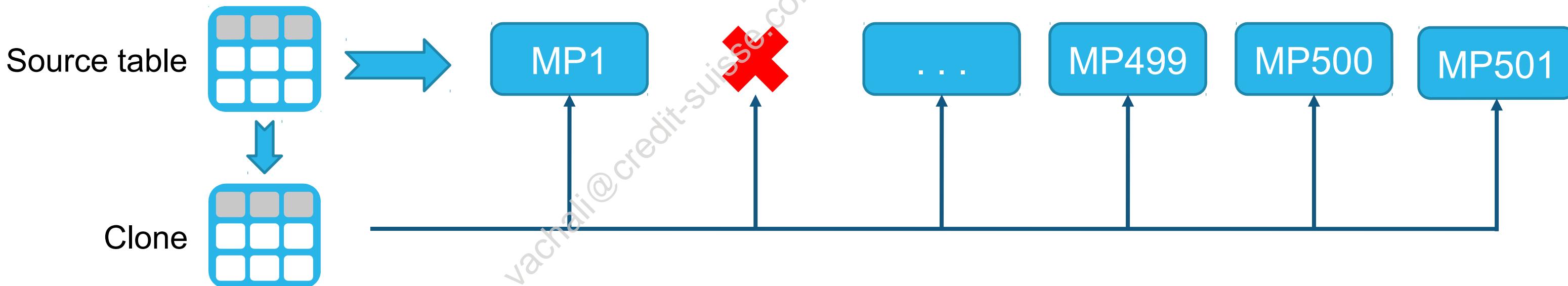
1. Source table contains micro-partitions 1 through 500
2. Source table is cloned
3. Before clone is finished, source table updates a row in MP2
4. Micro-partition 501 created, micro-partition 2 marked for delete



DML DURING CLONING

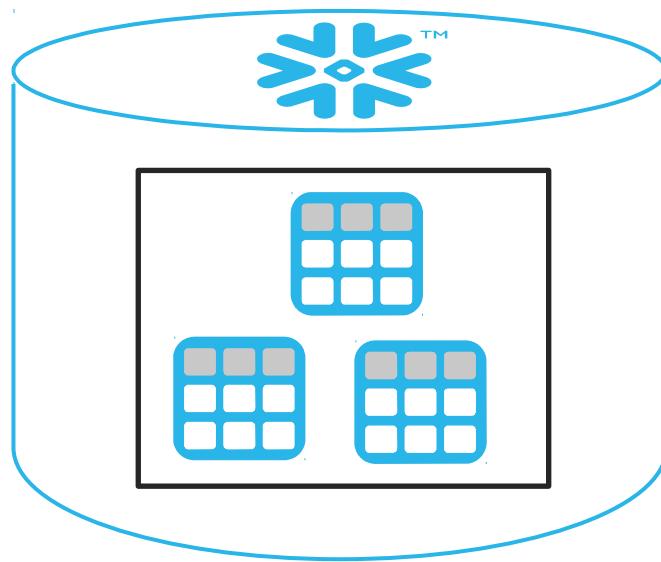
Avoid DML during cloning if Time Travel is set to 0 on the source table

1. Source table contains micro-partitions 1 through 500
2. Source table is cloned
3. Before clone is finished, source table updates a row in MP2
4. Micro-partition 501 created, micro-partition 2 marked for delete
5. Finished clone points to non-existent micro-partition 2; **cloning operation fails**



ACCESS CONTROL FOR CLONES

- The object cloned does not have any associated privileges, they must be assigned
- If the clone is a database or schema, objects **contained in** the clone retain their privileges



Role TEST_ROLE has:

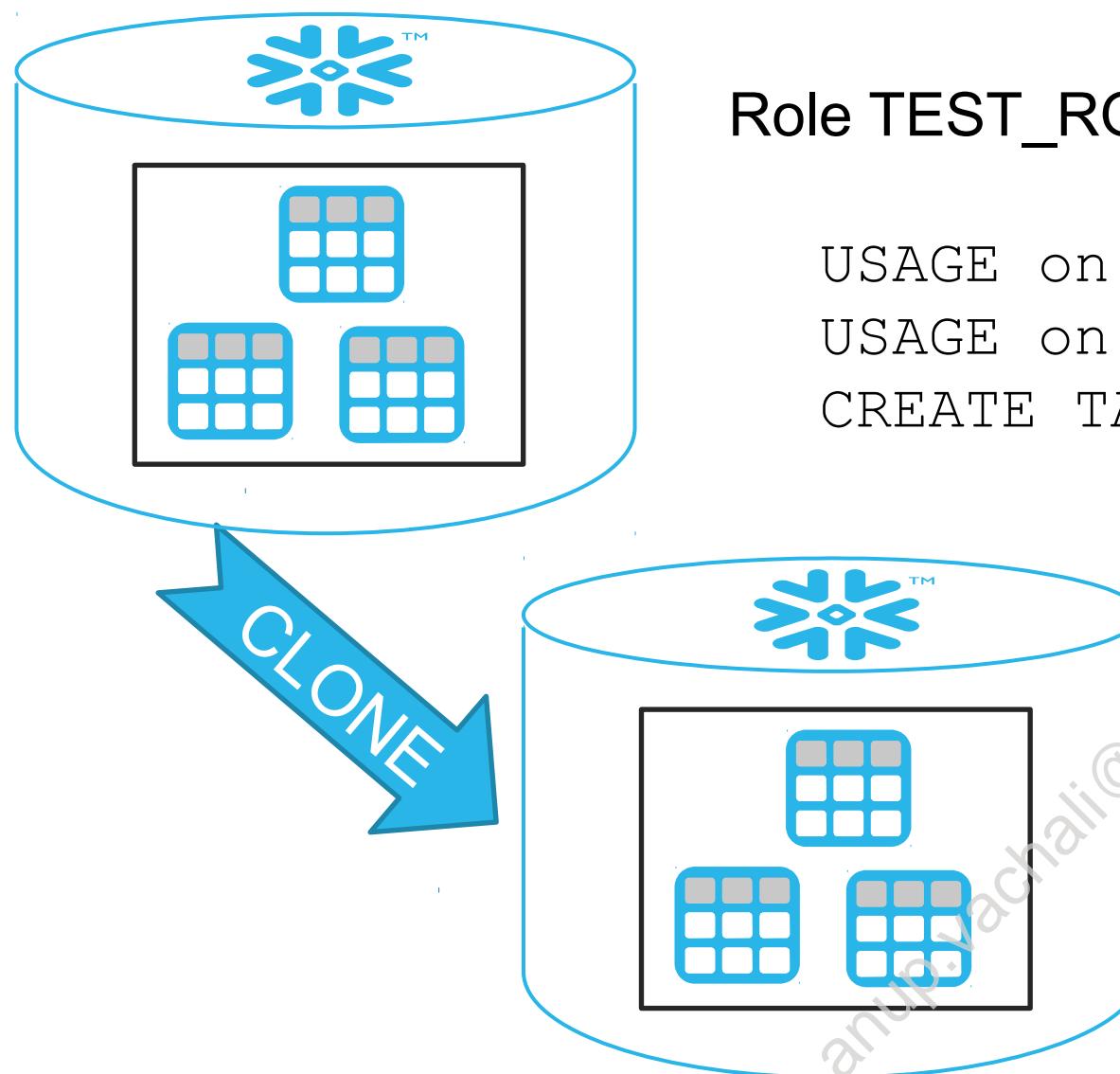
USAGE on the database

USAGE on the schema

CREATE TABLE in schema

ACCESS CONTROL FOR CLONES

- The object cloned does not have any associated privileges, they must be assigned
- If the clone is a database or schema, objects **contained** in the clone retain their privileges



Role TEST_ROLE has:

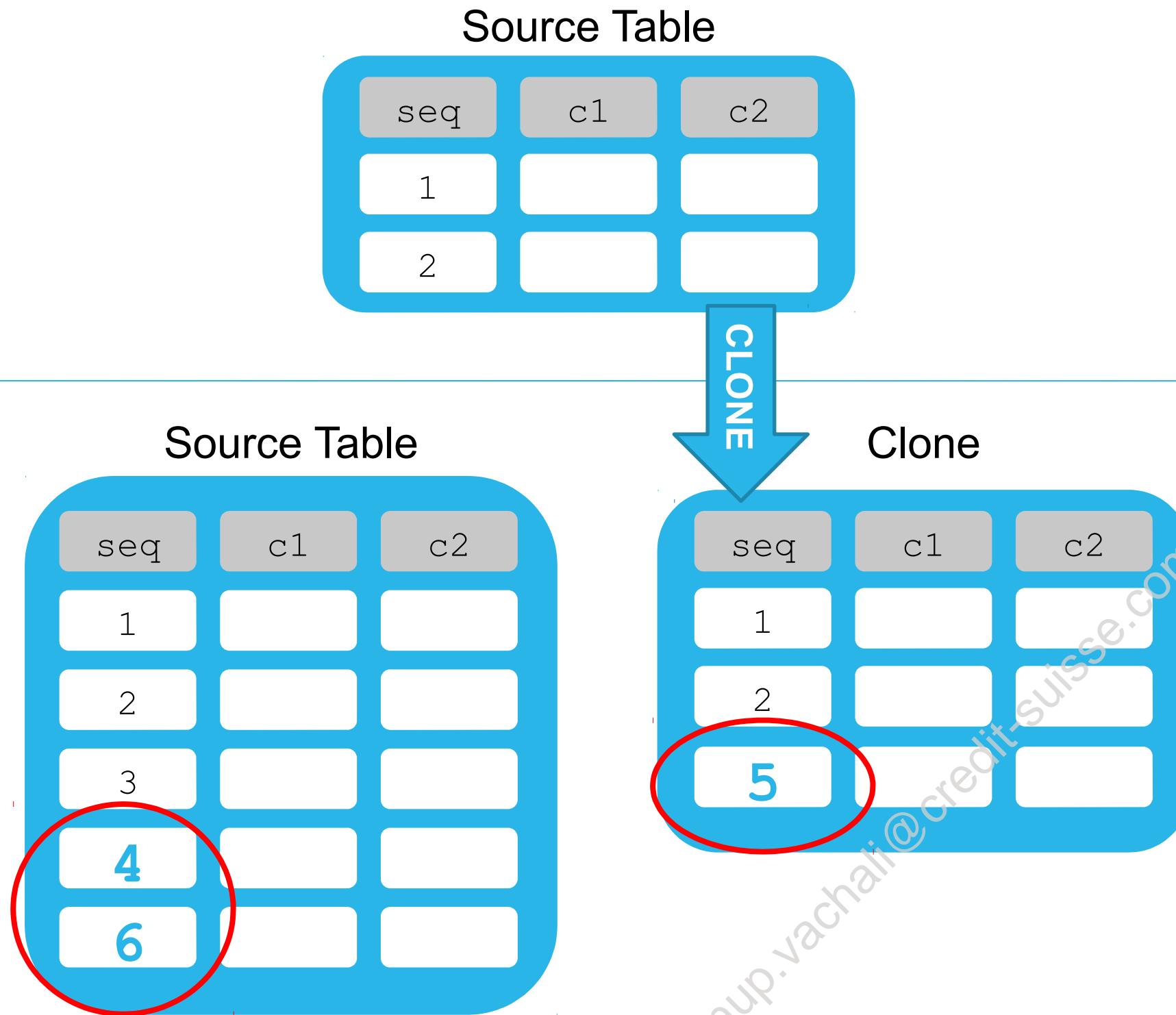
USAGE on the database
USAGE on the schemas
CREATE TABLE in schemas

TEST_ROLE retains the ability
to use the schemas in the
database and create tables –
but must first be granted
USAGE on the new database

Role TEST_ROLE has:

~~USAGE on the database~~
USAGE on the schemas
CREATE TABLE in schemas

CLONING TABLES WITH DEFAULT SEQUENCES



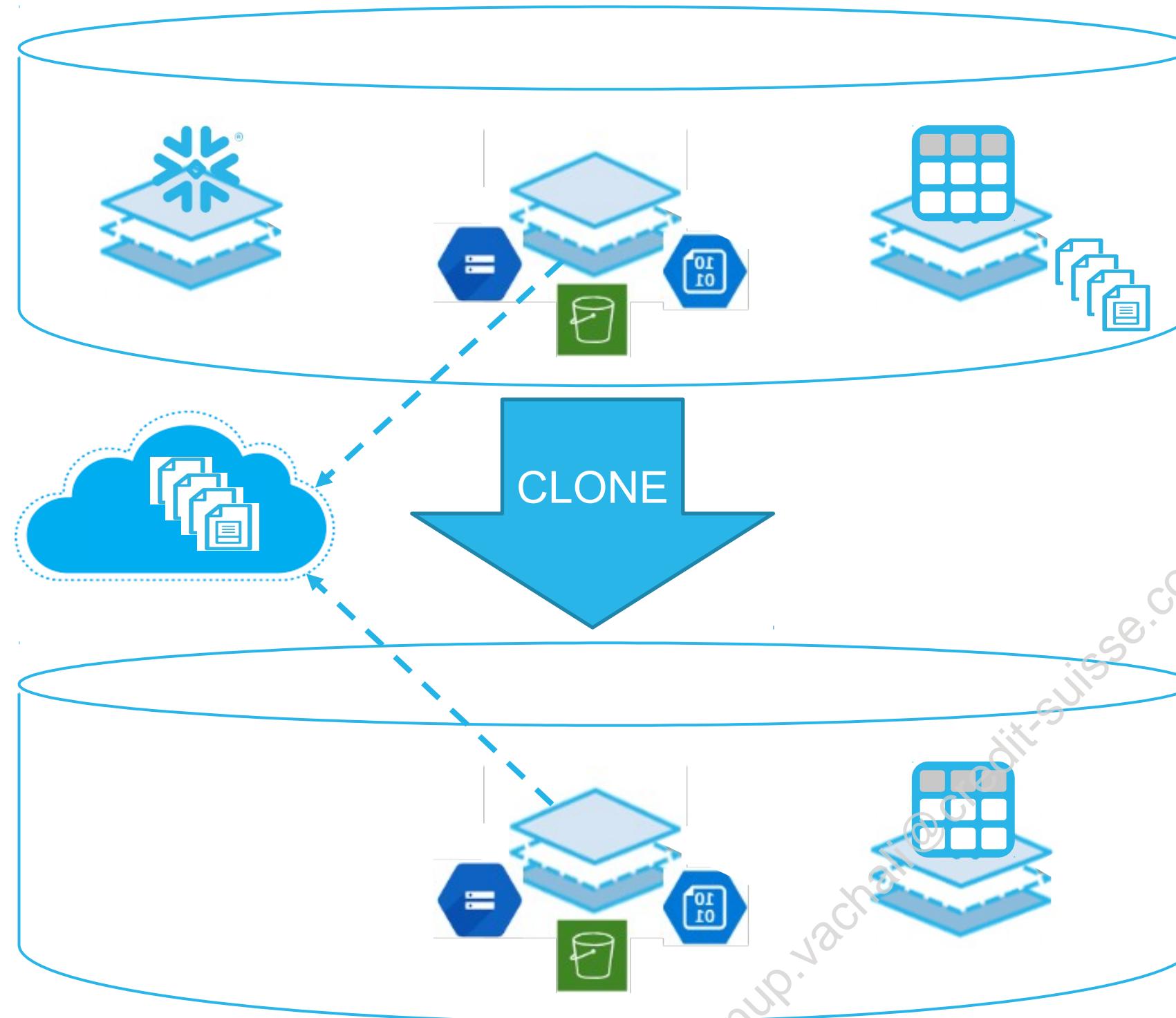
- If you clone a table that has a sequence column with a default value, the cloned table will reference the same sequence object as the original

- To prevent this, set up a new sequence for the clone, and run this command on the clone:

```
ALTER TABLE <cloned_table>
ALTER COLUMN <seq_col>
SET DEFAULT <new_seq>.nextval;
```



CLONING AND STAGES

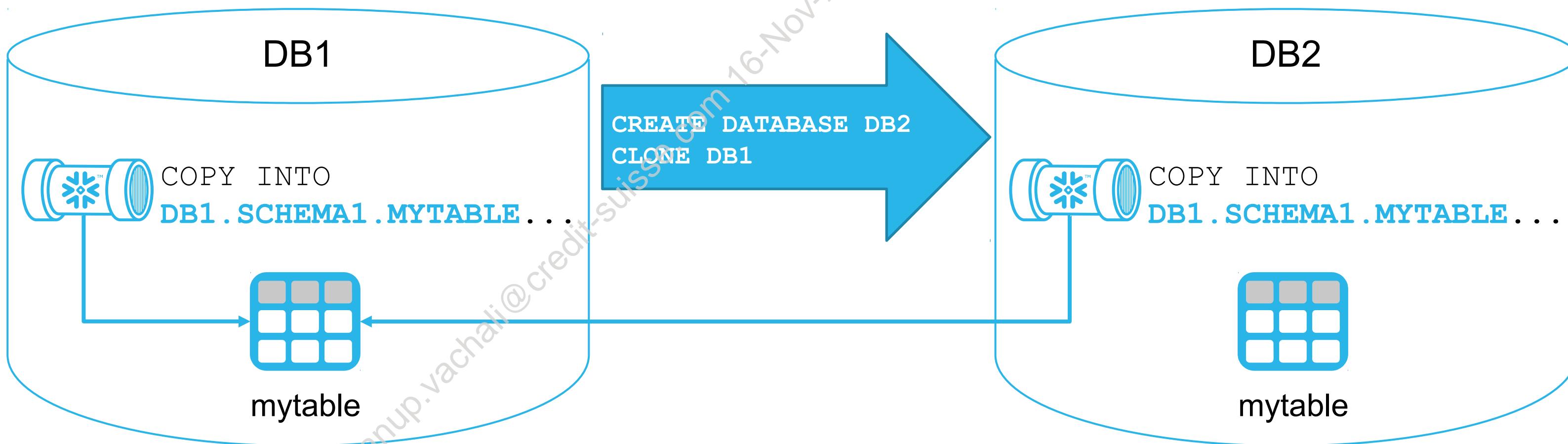


When cloning a database or schema that contains a stage:

- External named stages in the source will be cloned
 - Has no impact on the referenced cloud storage
- Tables (and their associated table stages) will be cloned
 - Any data files in the source's table stage are NOT cloned
- Internal named stages are not cloned

CLONING AN OBJECT CONTAINING A PIPE

- Pipes that reference an internal stage **are not** cloned
- Pipes that reference an external stage **are** cloned
 - If the pipe writes to a fully-qualified table name, duplicate data may get loaded into the source target table from both pipes



CLONING WITH STREAMS AND TASKS

- When an object containing a stream is cloned, the cloned stream's offset will be set to the time the clone occurred
 - Unconsumed records will be inaccessible to the clone
- When an object containing a task is cloned, the tasks in the clone are suspended
 - Tasks can be resumed with `ALTER TASK...RESUME`



AGILE DEVELOPMENT

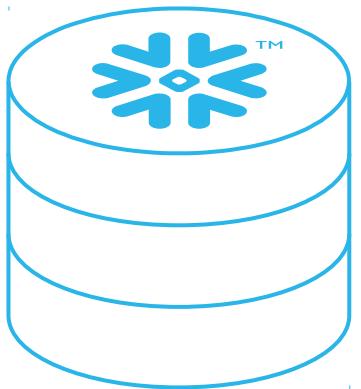
anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



AGILE DEVELOPMENT

USING CLONING

- You need to test scripts before deploying them against production data

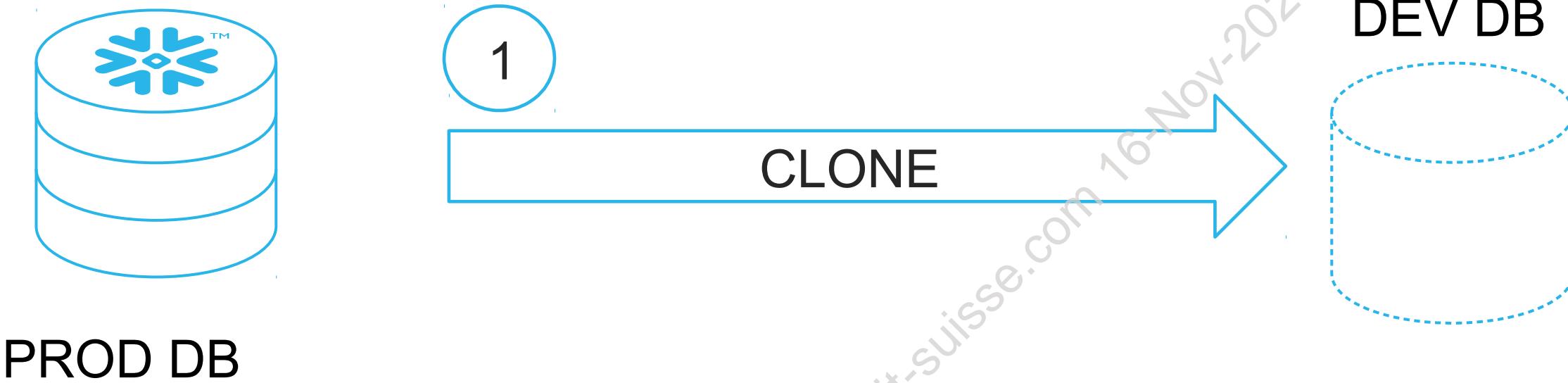


PROD DB

AGILE DEVELOPMENT

USING CLONING

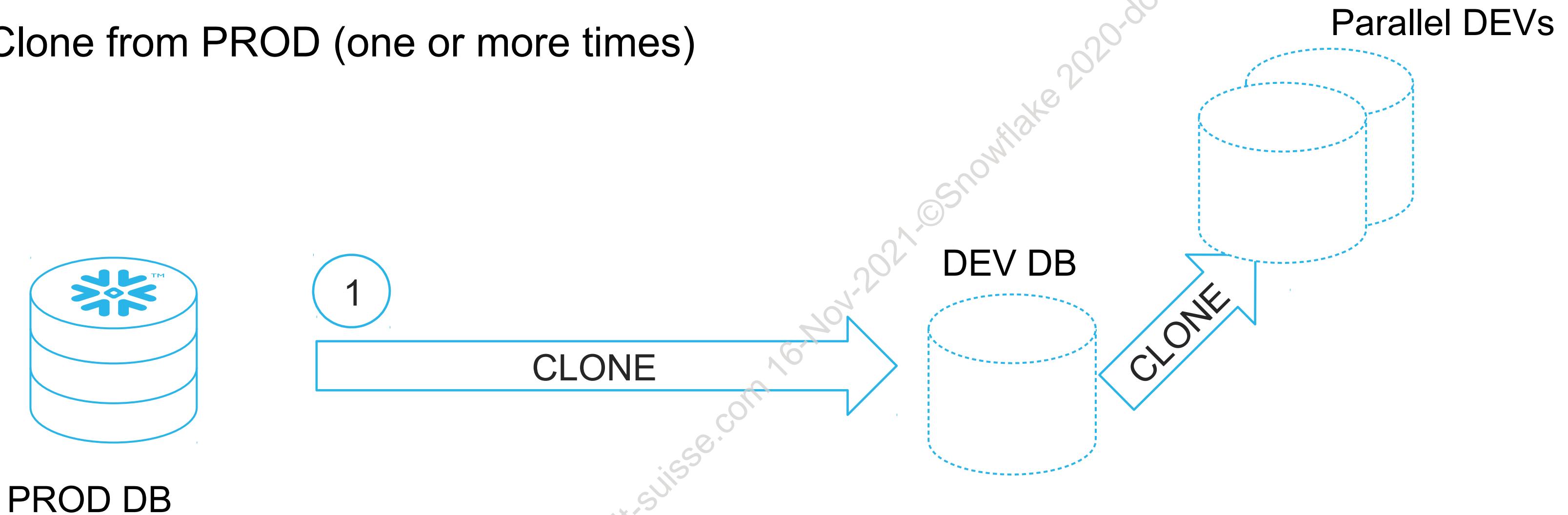
1. Clone from PROD



AGILE DEVELOPMENT

USING CLONING

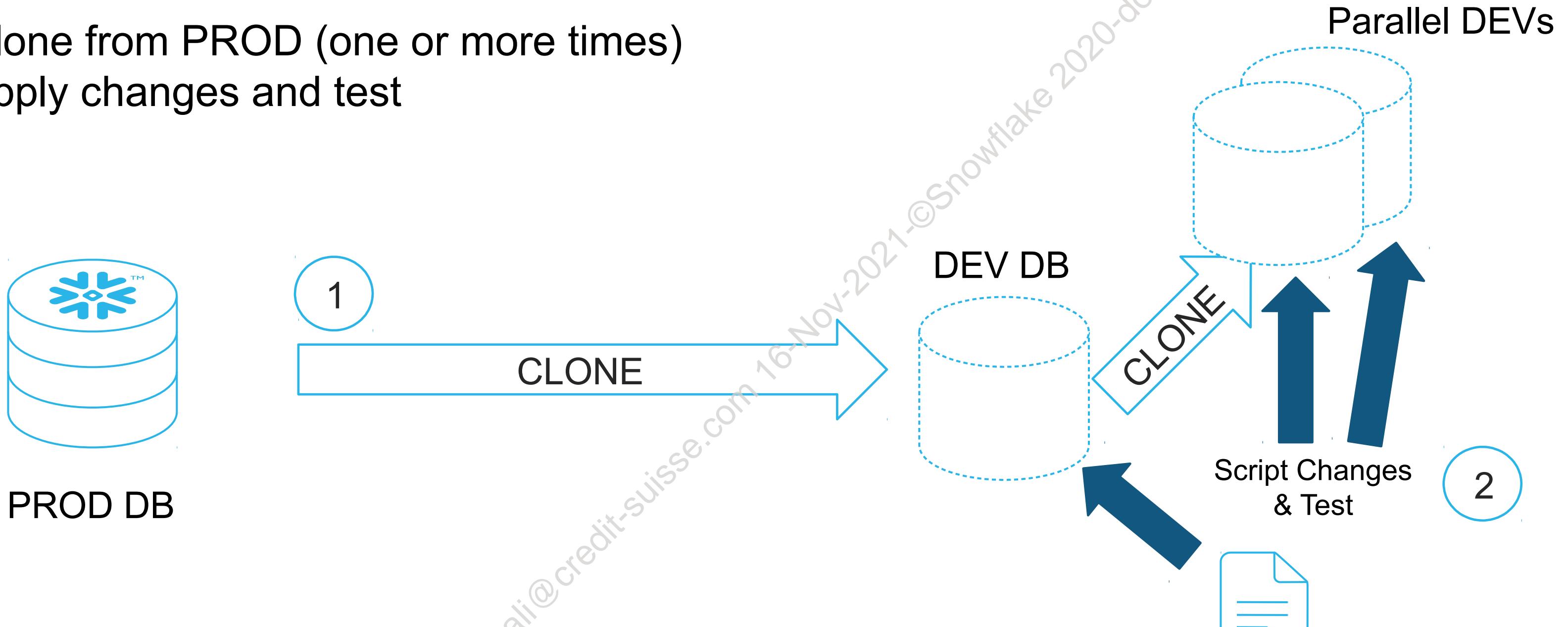
1. Clone from PROD (one or more times)



AGILE DEVELOPMENT

USING CLONING

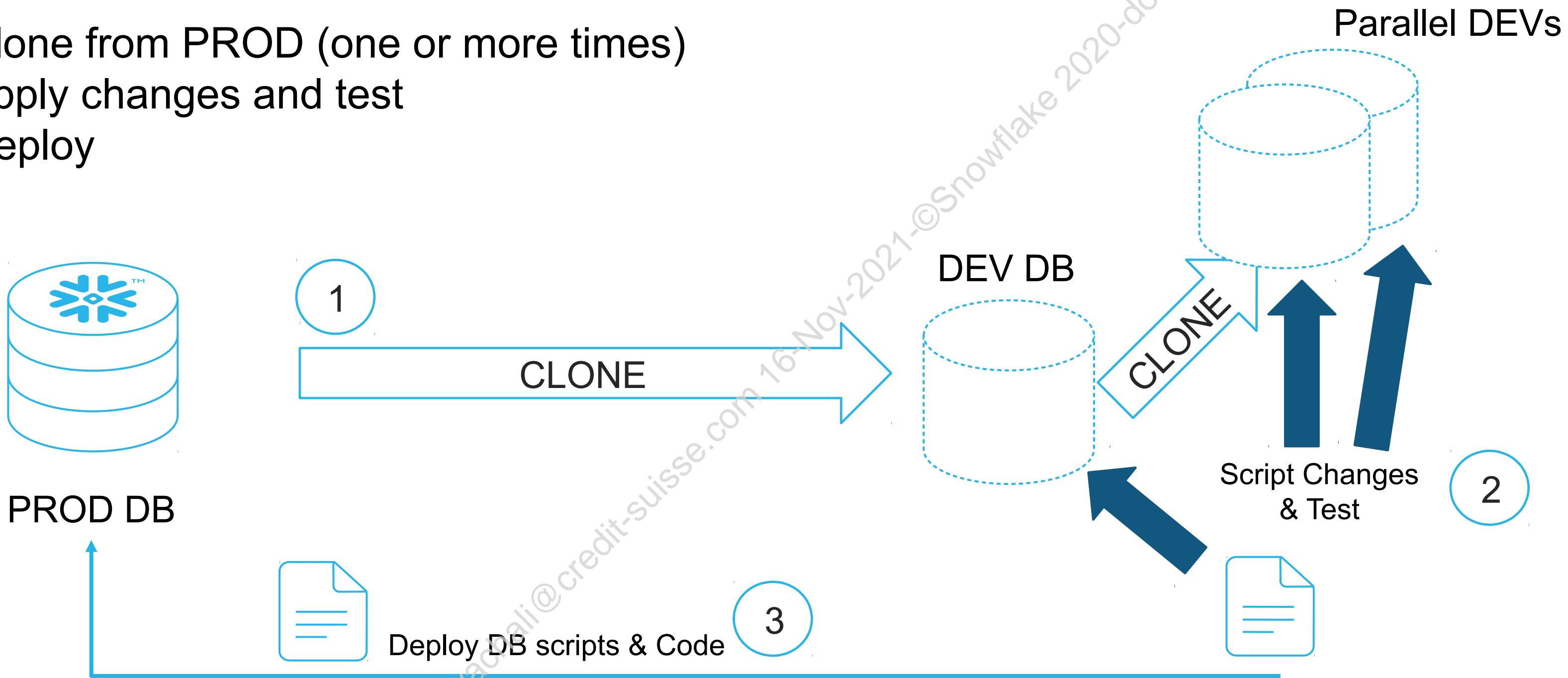
1. Clone from PROD (one or more times)
2. Apply changes and test



AGILE DEVELOPMENT

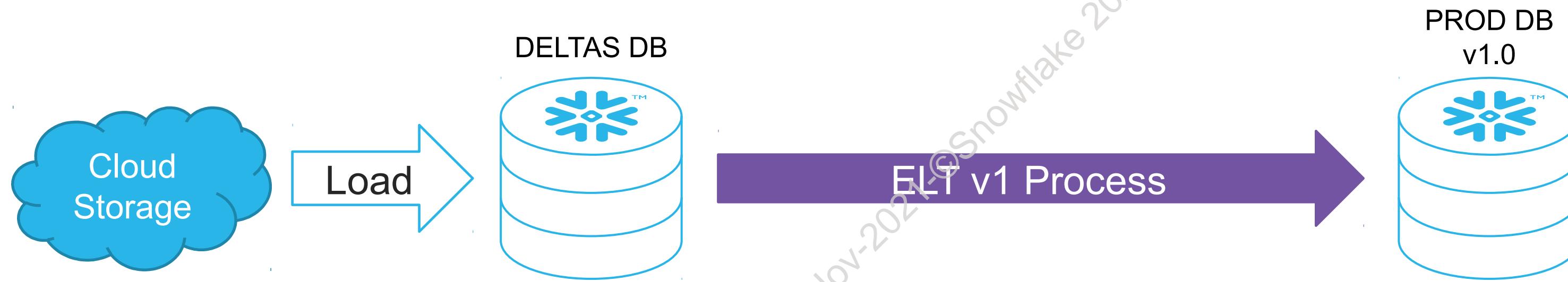
USING CLONING

1. Clone from PROD (one or more times)
2. Apply changes and test
3. Deploy



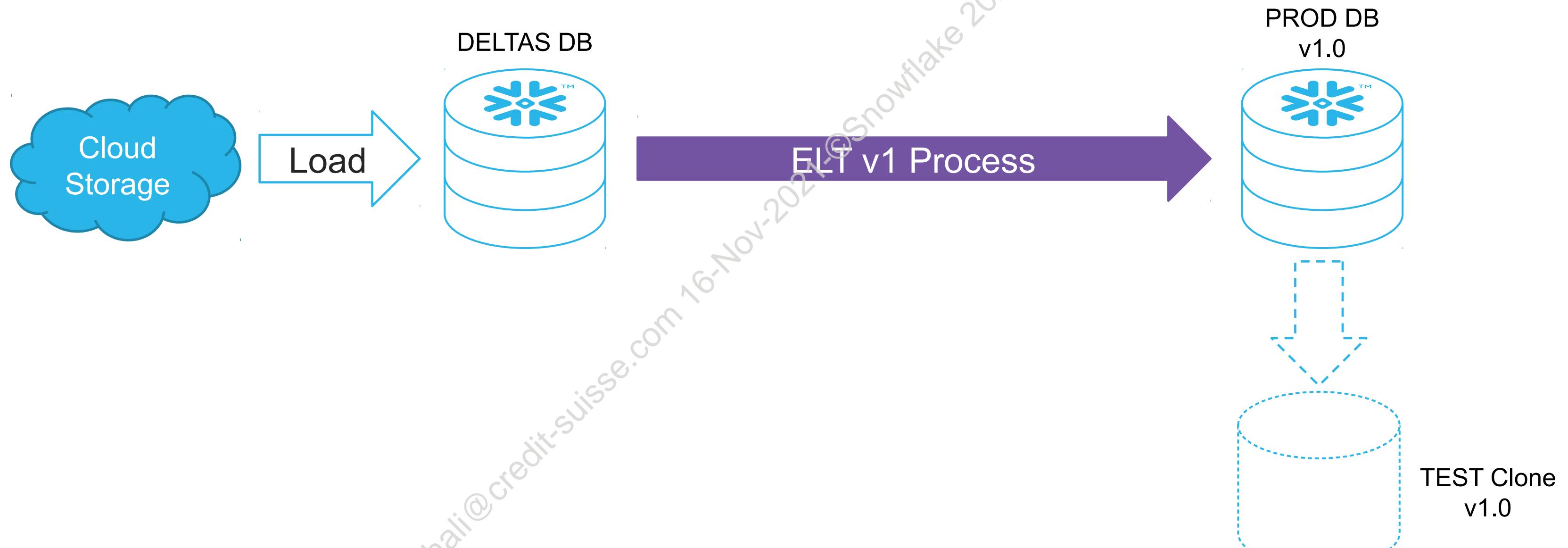
AGILE DEVELOPMENT & TESTING

APPLY DELTA CHANGES TO PROD



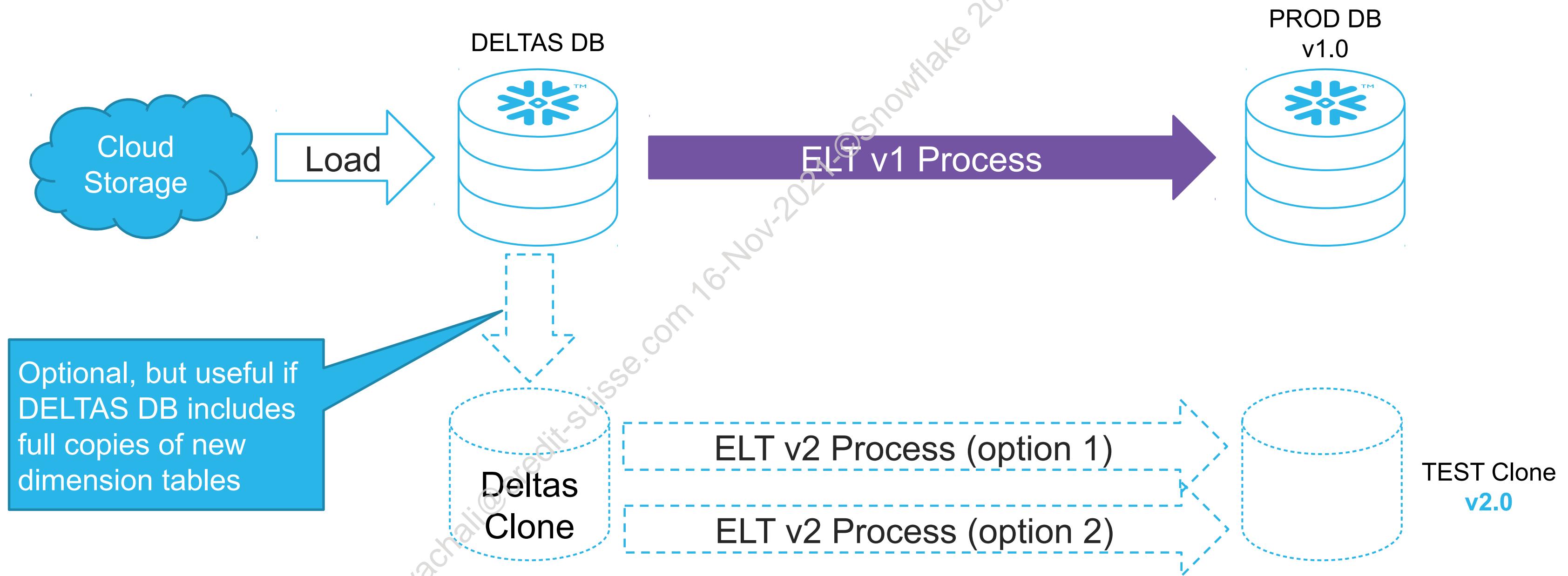
AGILE DEVELOPMENT & TESTING

CLONE PROD TO TEST



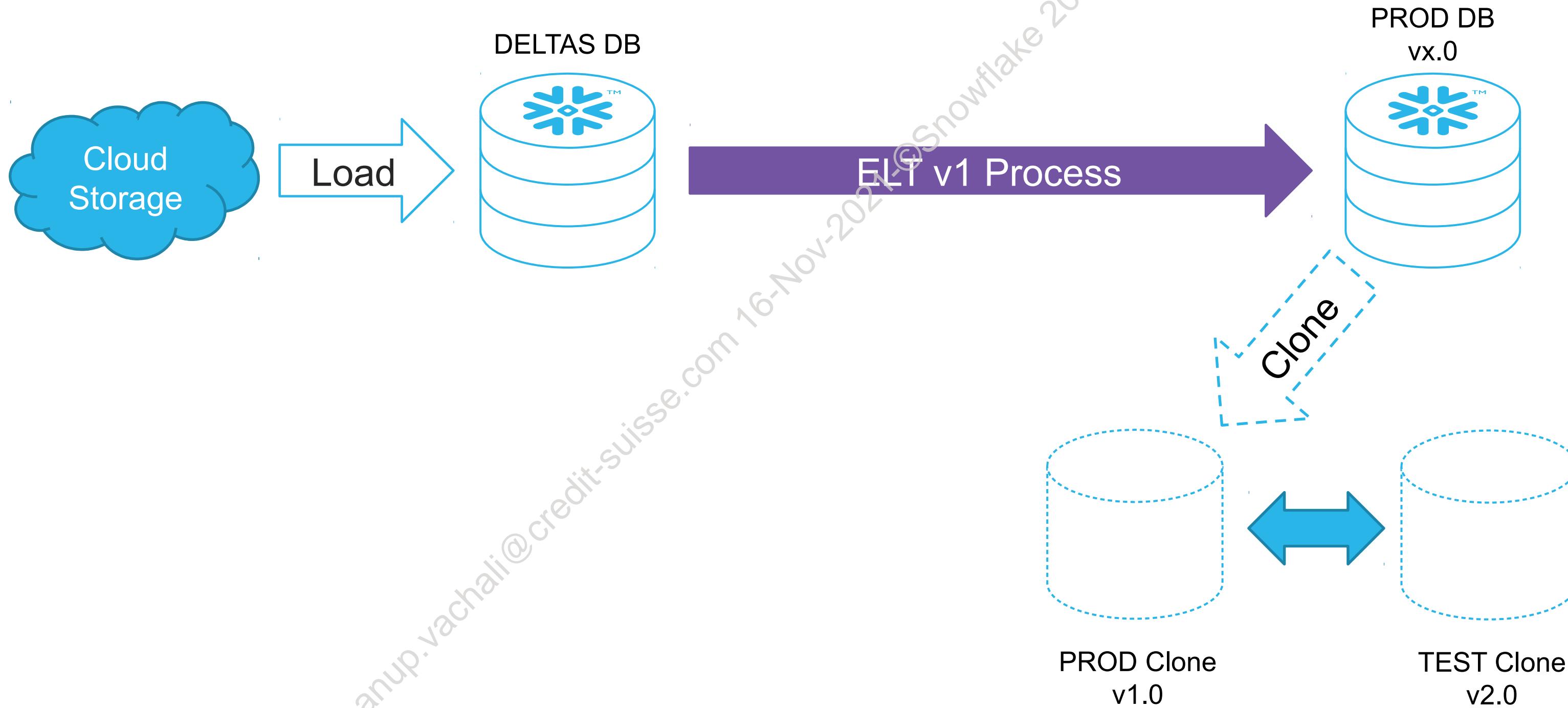
AGILE DEVELOPMENT & TESTING

CLONE DELTAS DB AND VERIFY NEW ELT PROCESS ON TEST



AGILE DEVELOPMENT & TESTING

CLONE PROD AGAIN AND COMPARE FOR REGRESSION TESTING



CONSIDERATIONS

- Will increase storage costs (amount depends on how much data is changing)
- Cloning a single table doesn't clone the access grants
 - Cloning a Schema or Database does retain access grants for the objects inside
- When cloning a database, external stages are cloned (internal stages are not cloned)
 - Files in external stages are not cloned



LAB EXERCISE: 2

Time Travel and Cloning

15 minutes

- Exploring Continuous Data Protection
- Using Clones for Recovery

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



BUILD A DATA WAREHOUSE

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy

ACCOUNT SETUP AND SECURITY

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



MODULE AGENDA

- Account Deployment Approaches
- Account Security
- Account Parameters

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



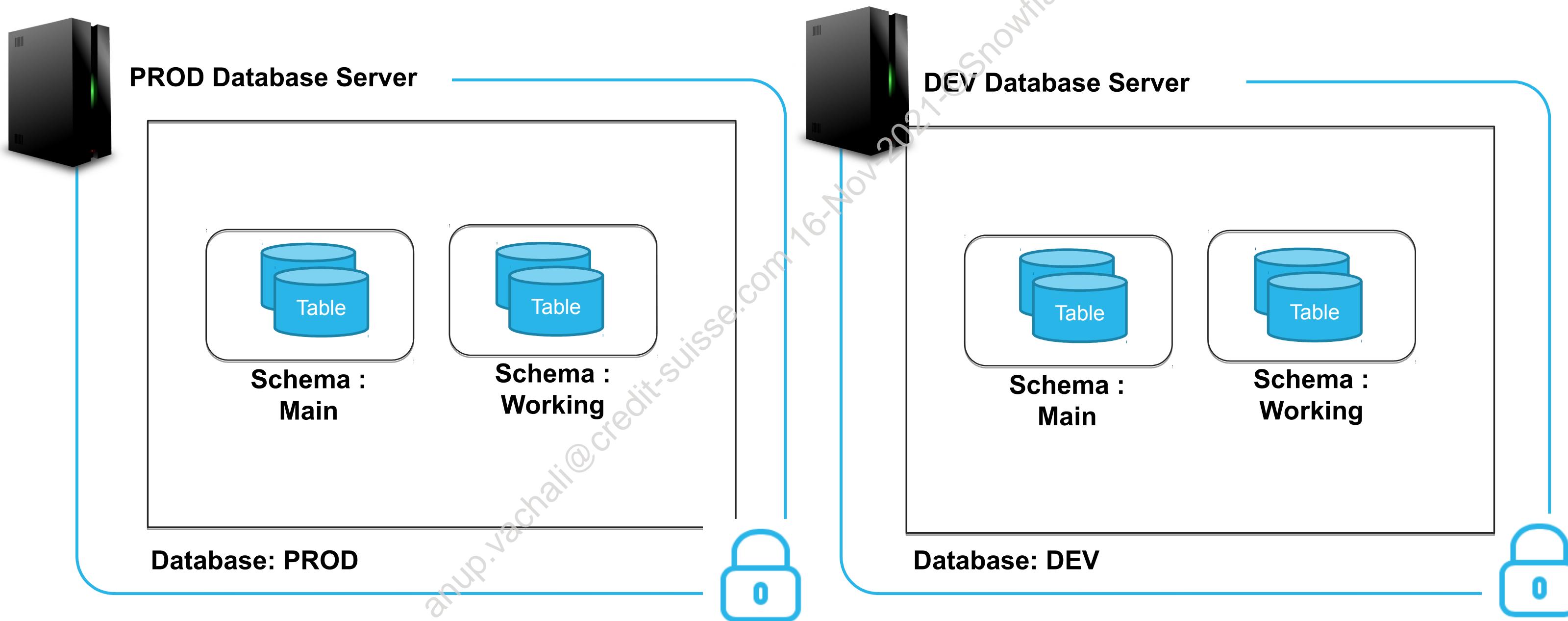
ACCOUNT DEPLOYMENT APPROACHES

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



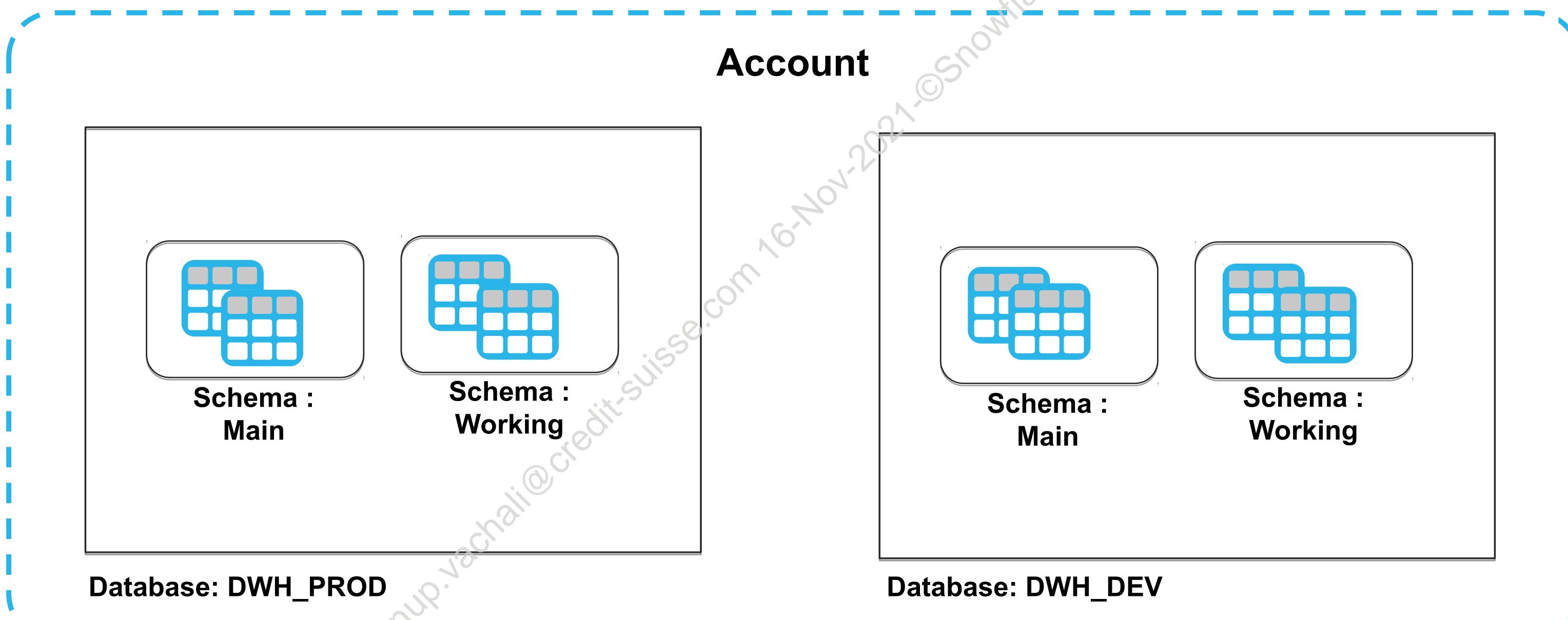
TRADITIONAL DATABASE DEPLOYMENT

Server/Database/Schema combination each on independent security domains

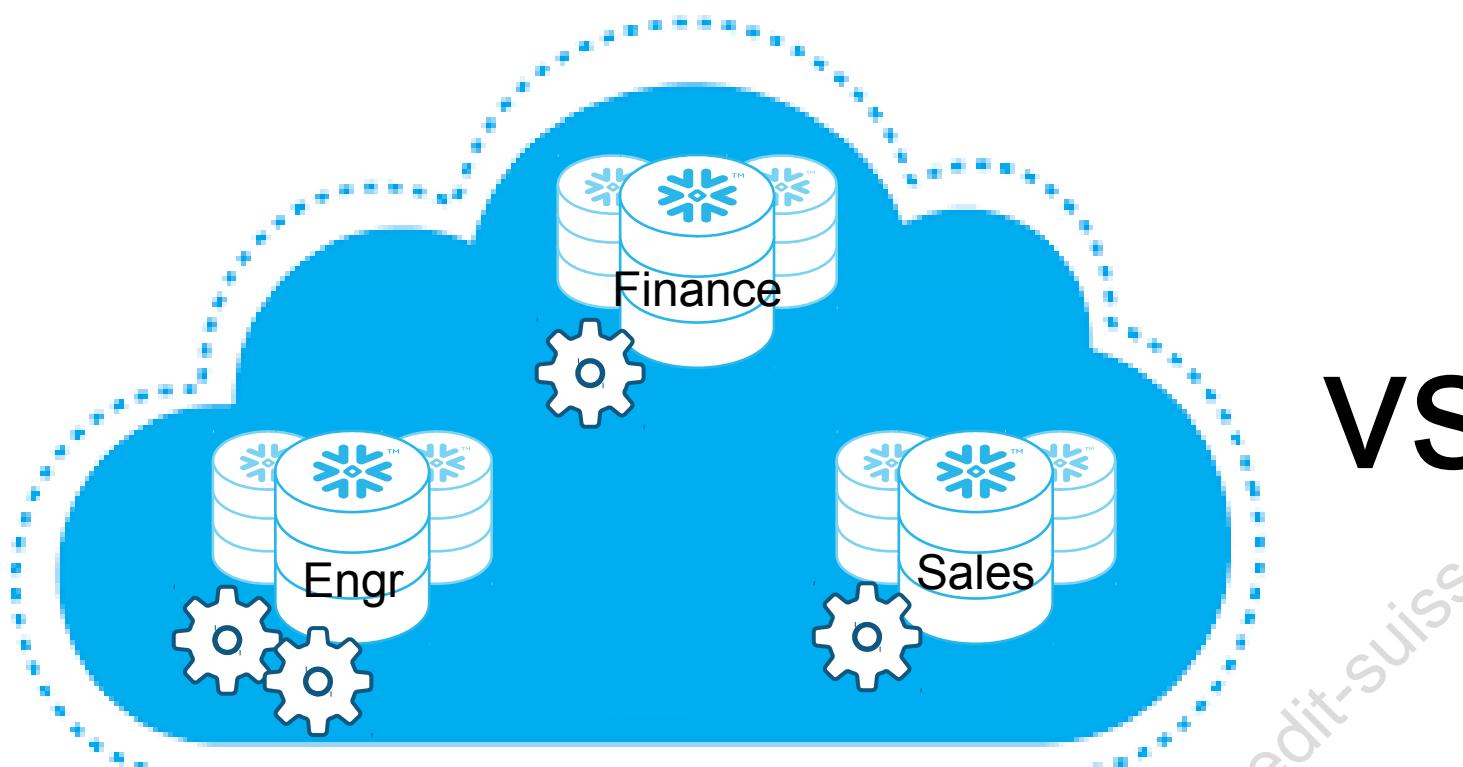


SNOWFLAKE ACCOUNT

- Account replaces multiple siloed databases; logical security model



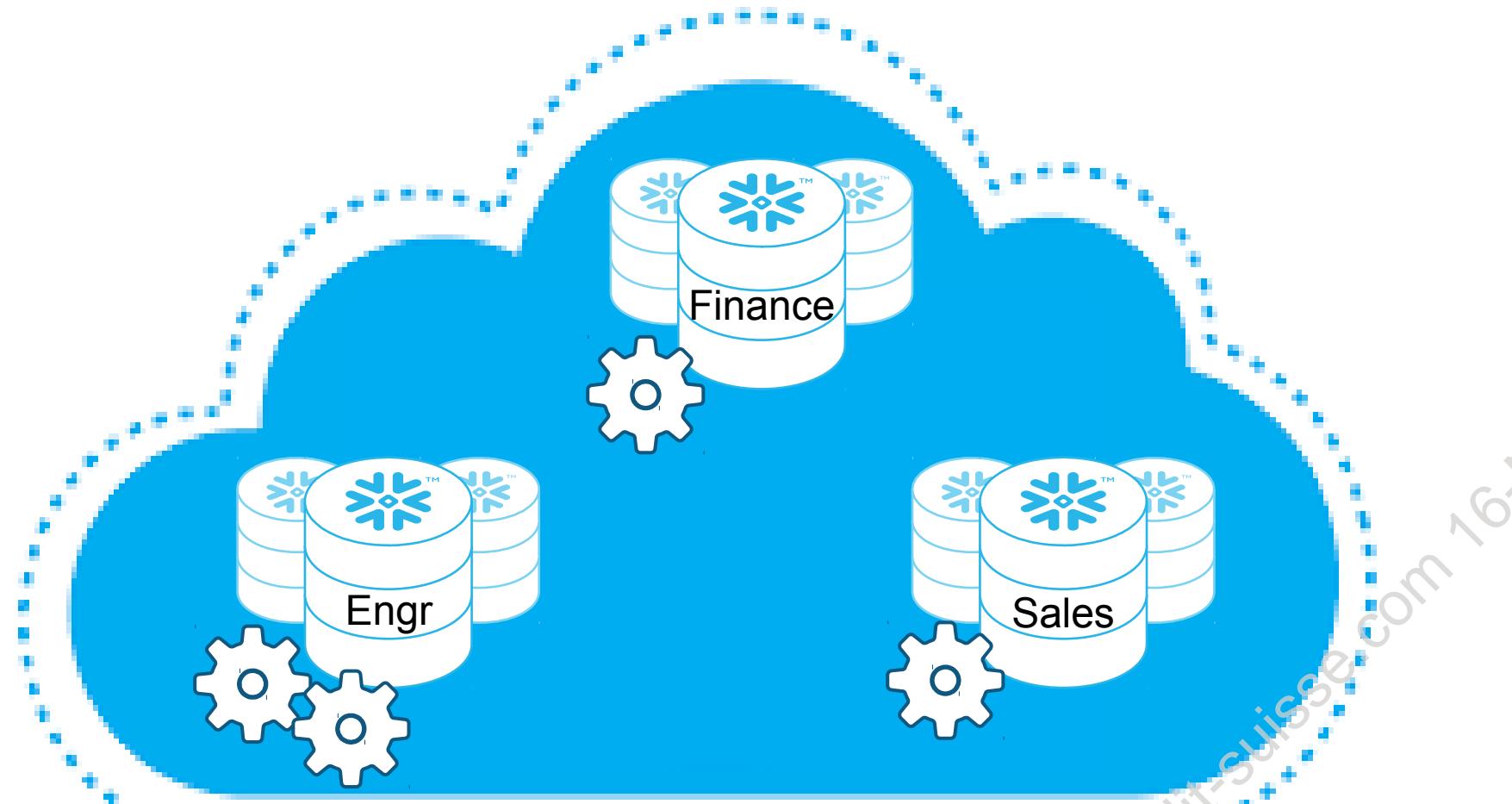
ONE ACCOUNT, OR MANY?



VS



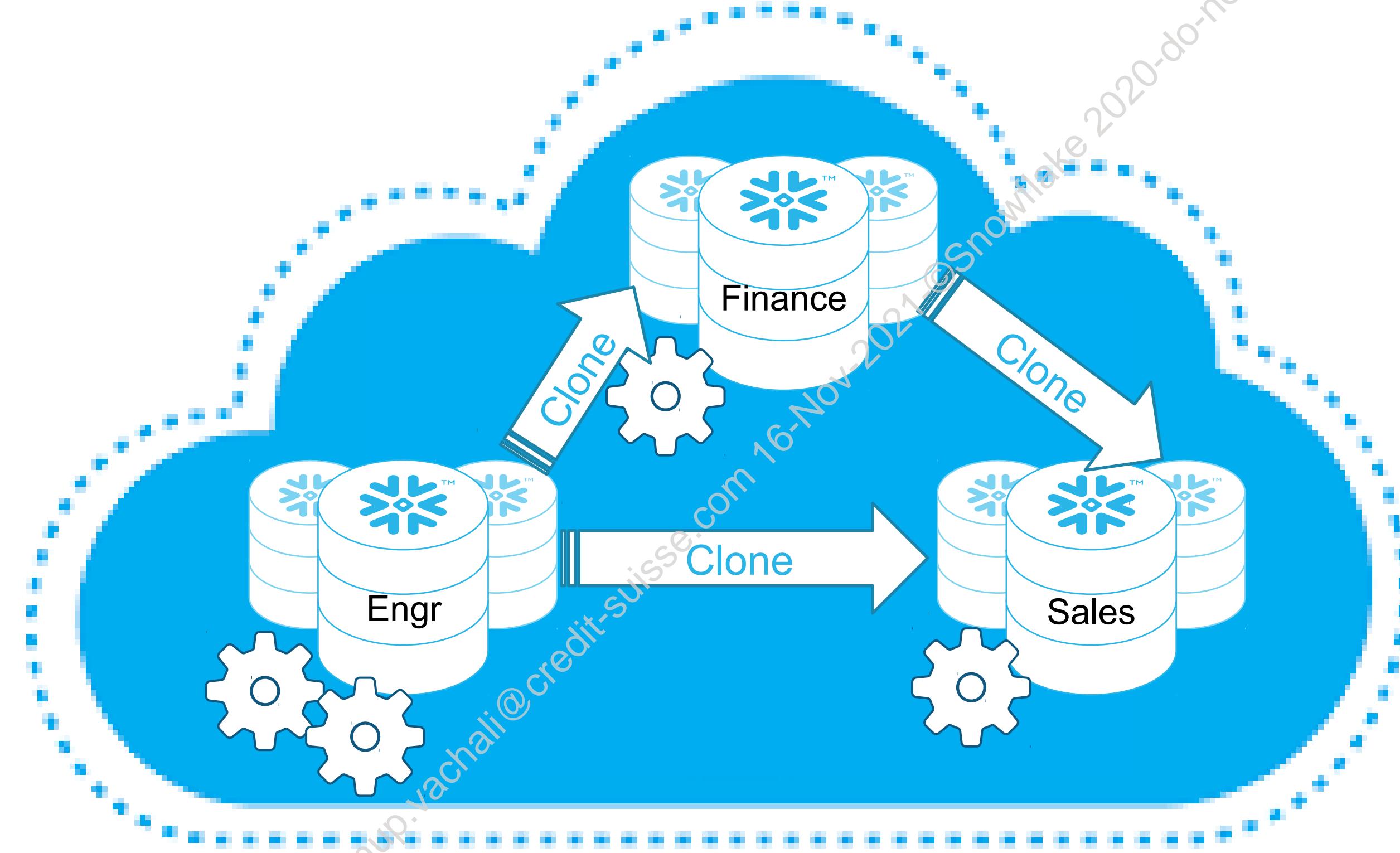
OPTION 1: SINGLE SNOWFLAKE ACCOUNT



- Support all business units on a single account
- Shared control of data
Segregated privileges by role
- Billing can still be allocated to different business units
- Consolidated
 - One cloud provider
 - One region
 - One Snowflake edition



COLLABORATE WITHIN AN ACCOUNT



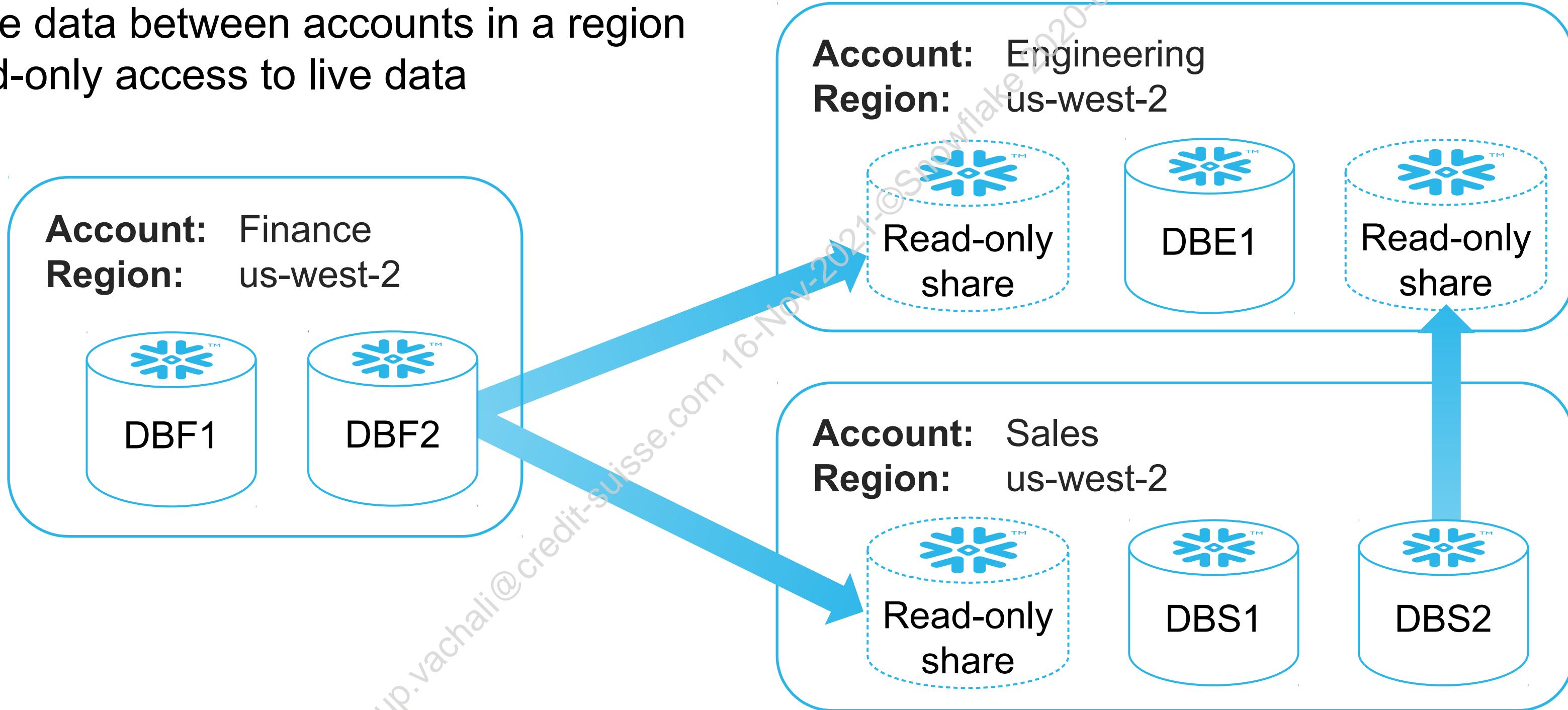
OPTION 2: SNOWFLAKE ORGANIZATION

- Separate accounts for business units for independent control
- Fully segregated billing
- Support multiple cloud providers, multiple regions, or multiple editions
- Accounts in the same region can share data



SHARE DATA BETWEEN ACCOUNTS

- Share data between accounts in a region
- Read-only access to live data



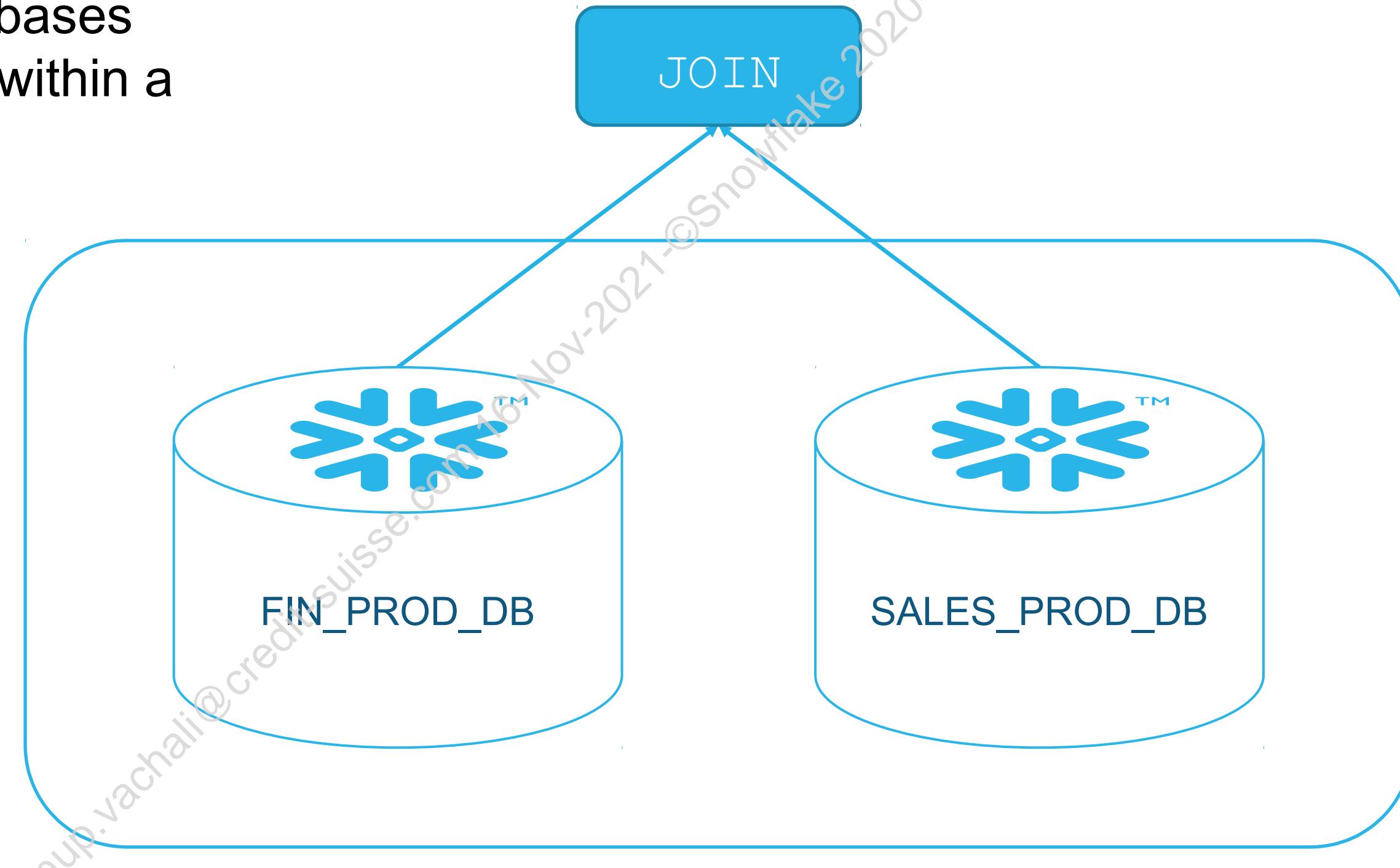
CLONE VS SHARE

	Clone	Share
Transfer Boundary	Clone within a single account	Share across accounts in a region
Data Refresh Rate	Static, point-in-time "snapshot"	Real-time access to live data
Write Privileges	Read/Write: clone can be altered independently from the original	Read-only: consumer can make a read/write copy using CTAS
Storage Consumption	Initially, no additional storage Clone takes up additional storage as changes are made	No additional storage (unless consumer creates a CTAS copy)



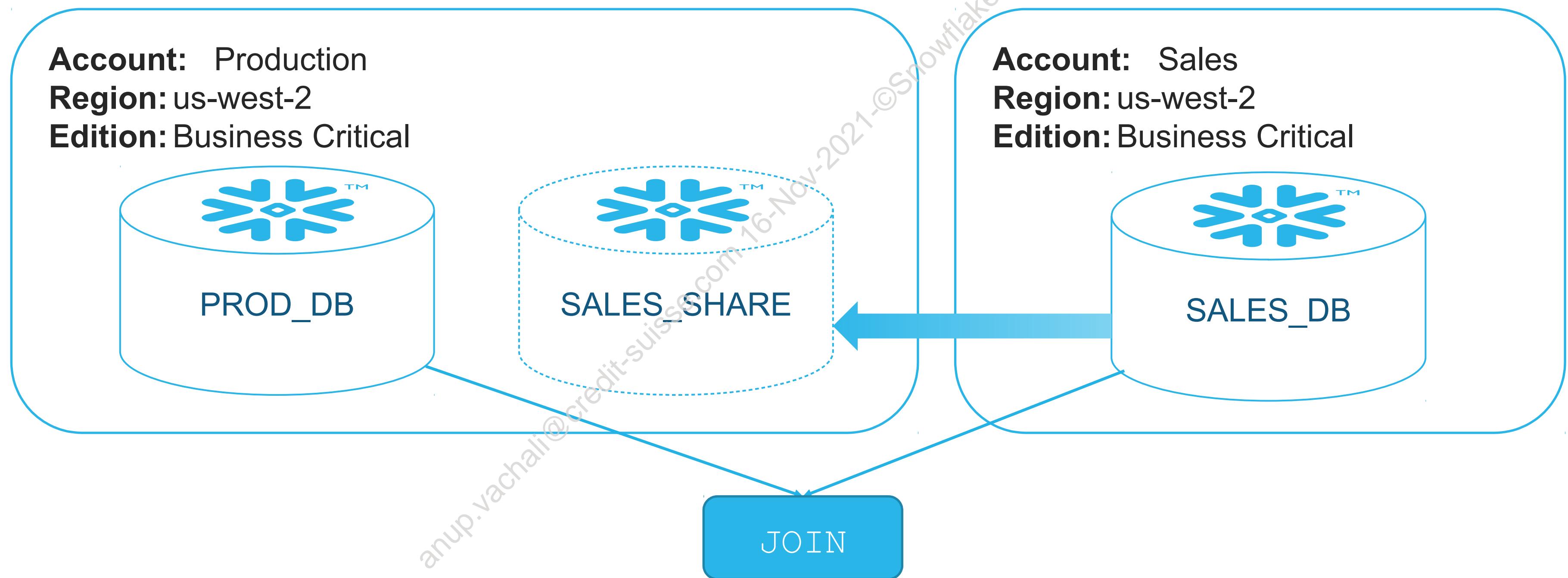
QUERIES BETWEEN DATABASES

- Queries between databases are as fast as queries within a single database



QUERIES WITH A SHARED DATABASE

- Queries across data shares are as fast as queries within the same account



SUMMARY

ONE ACCOUNT

- Logically separate business units
- Share data with cloning
- Single cloud provider
- Single Snowflake edition



MULTIPLE ACCOUNTS

- Physically separate business units
- Share data with data sharing
- Can be multiple cloud providers
- Can be different editions



LAB EXERCISE: 3

Querying Between Databases and Accounts

15 minutes

- JOIN two tables in your account
- JOIN two tables in a shared database
- JOIN a table in your account with a shared table



SECURE YOUR ACCOUNT

anup.vachali@credit-suisse.com 16 Nov 2021 ©Snowflake 2020-do-not-copy



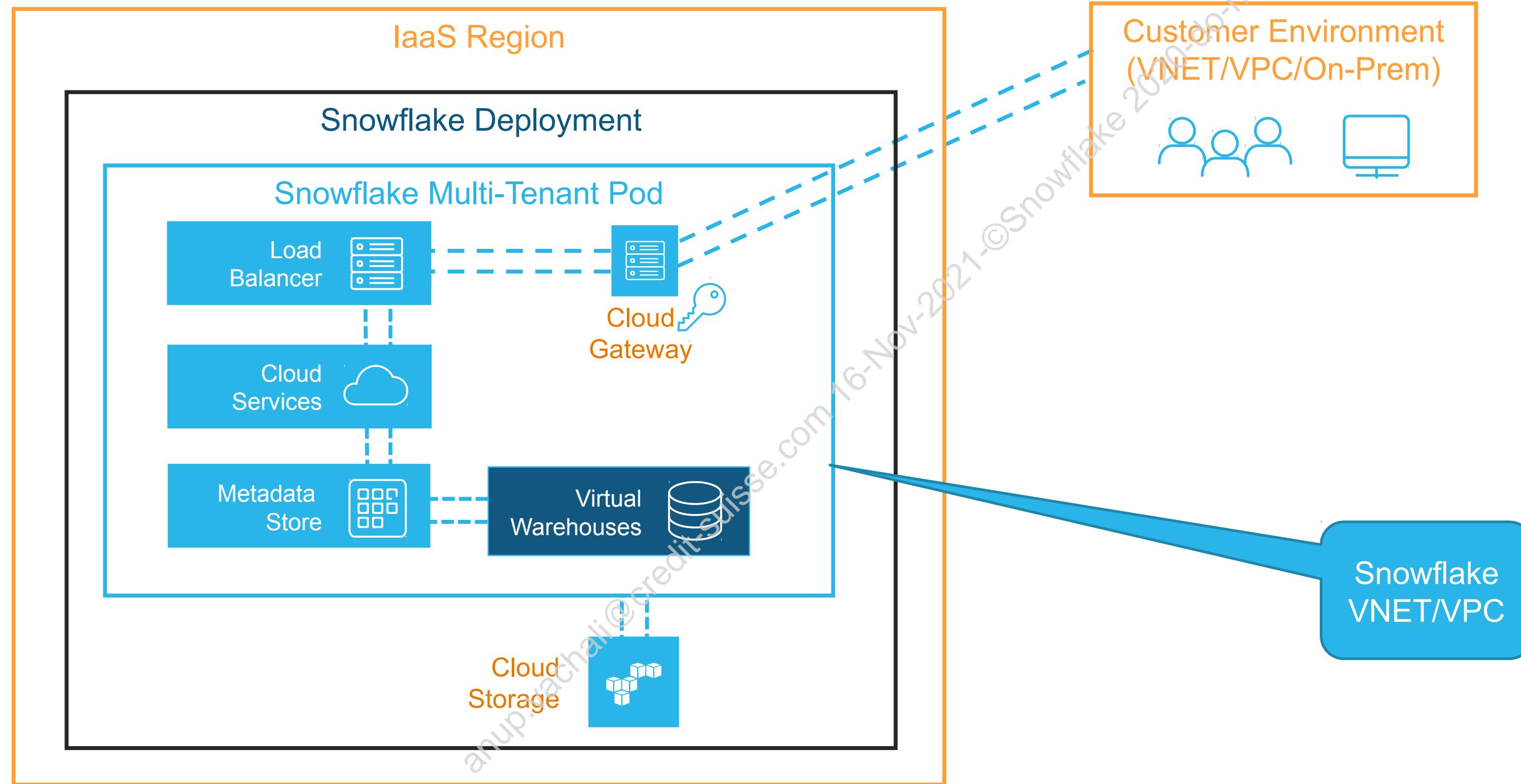
DEPLOYMENT SECURITY AT A GLANCE

IaaS Region

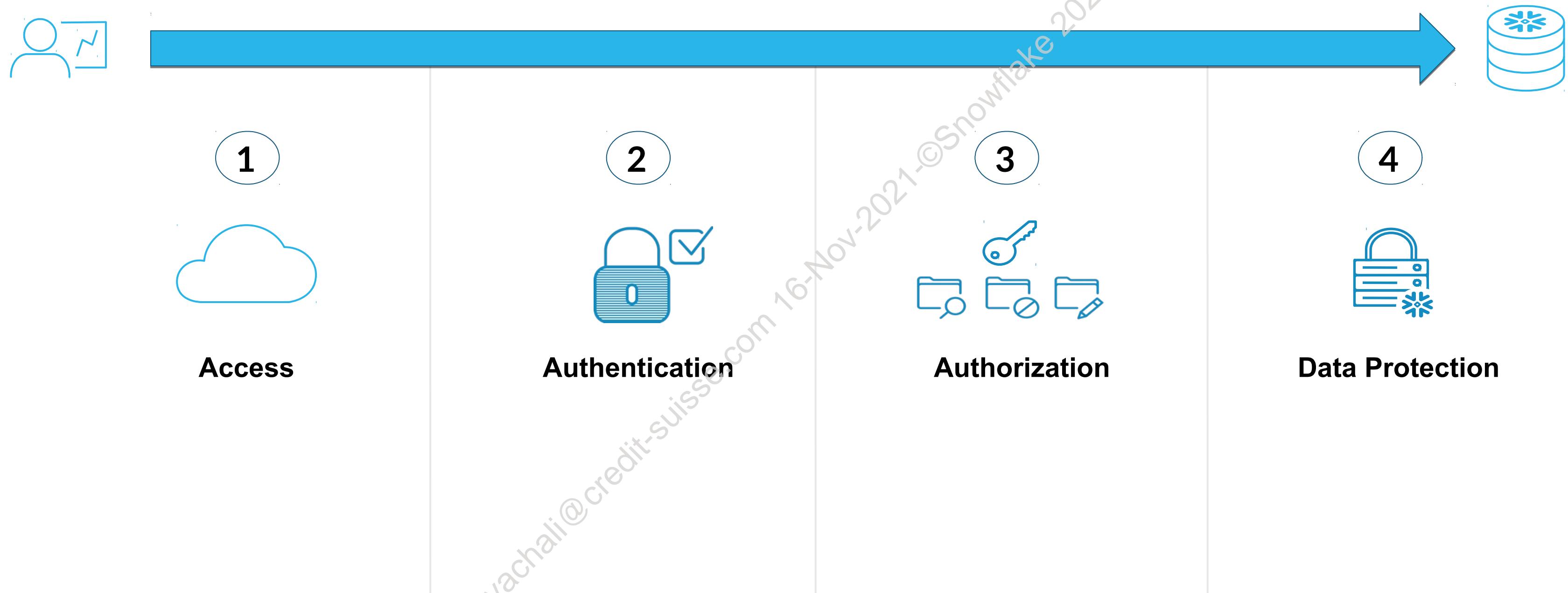
Snowflake Deployment



DEPLOYMENT SECURITY AT A GLANCE

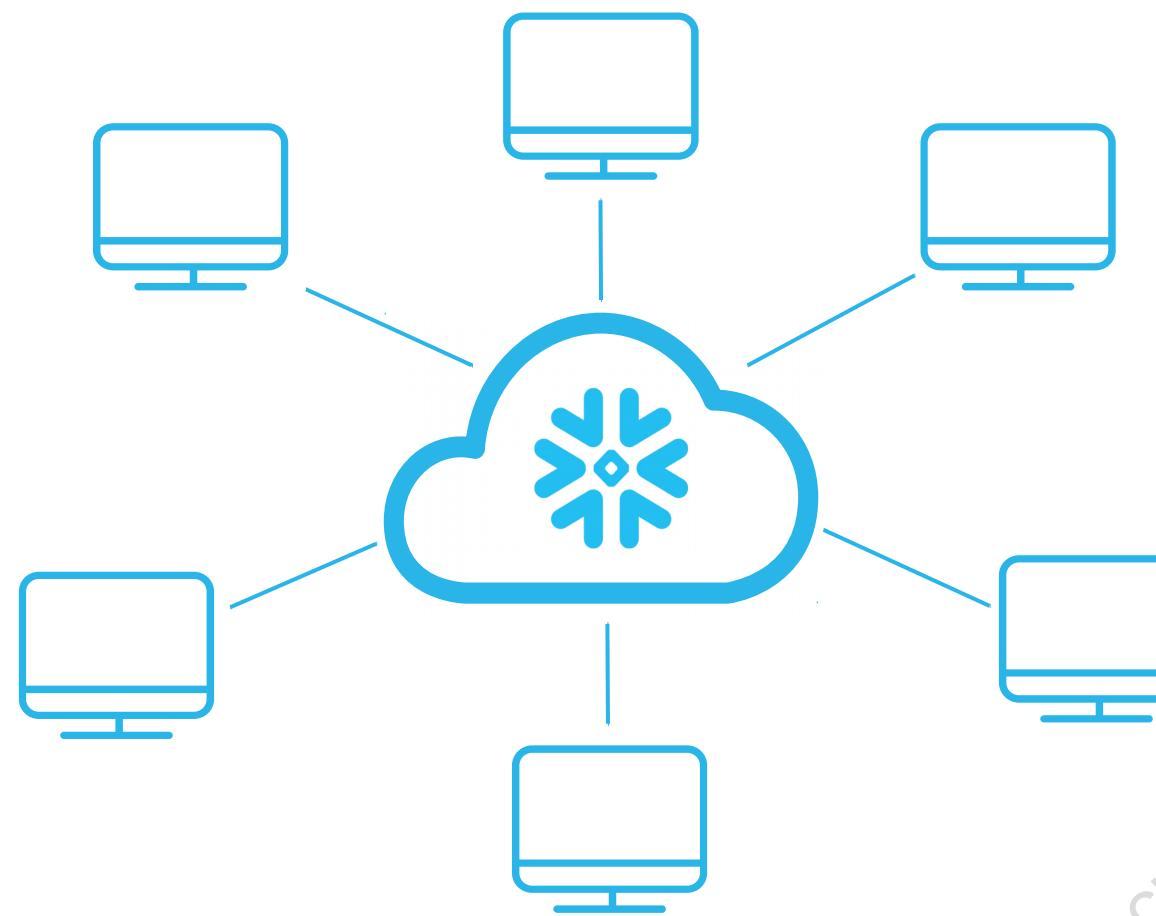


SNOWFLAKE LAYERED SECURITY



NETWORK POLICIES

PROTECT ACCESS TO YOUR ACCOUNT



- A network policy describes a list or range of IP addresses that can connect to the Snowflake account
- Network policies can be applied to the account, or to specific users



CREATE A NETWORK POLICY

CREATE NETWORK POLICY <policy_name>

ALLOWED_IP_LIST = (<IP addresses or CIDR blocks>)

[BLOCKED_IP_LIST = (<IP addresses or CIDR blocks>)] ;

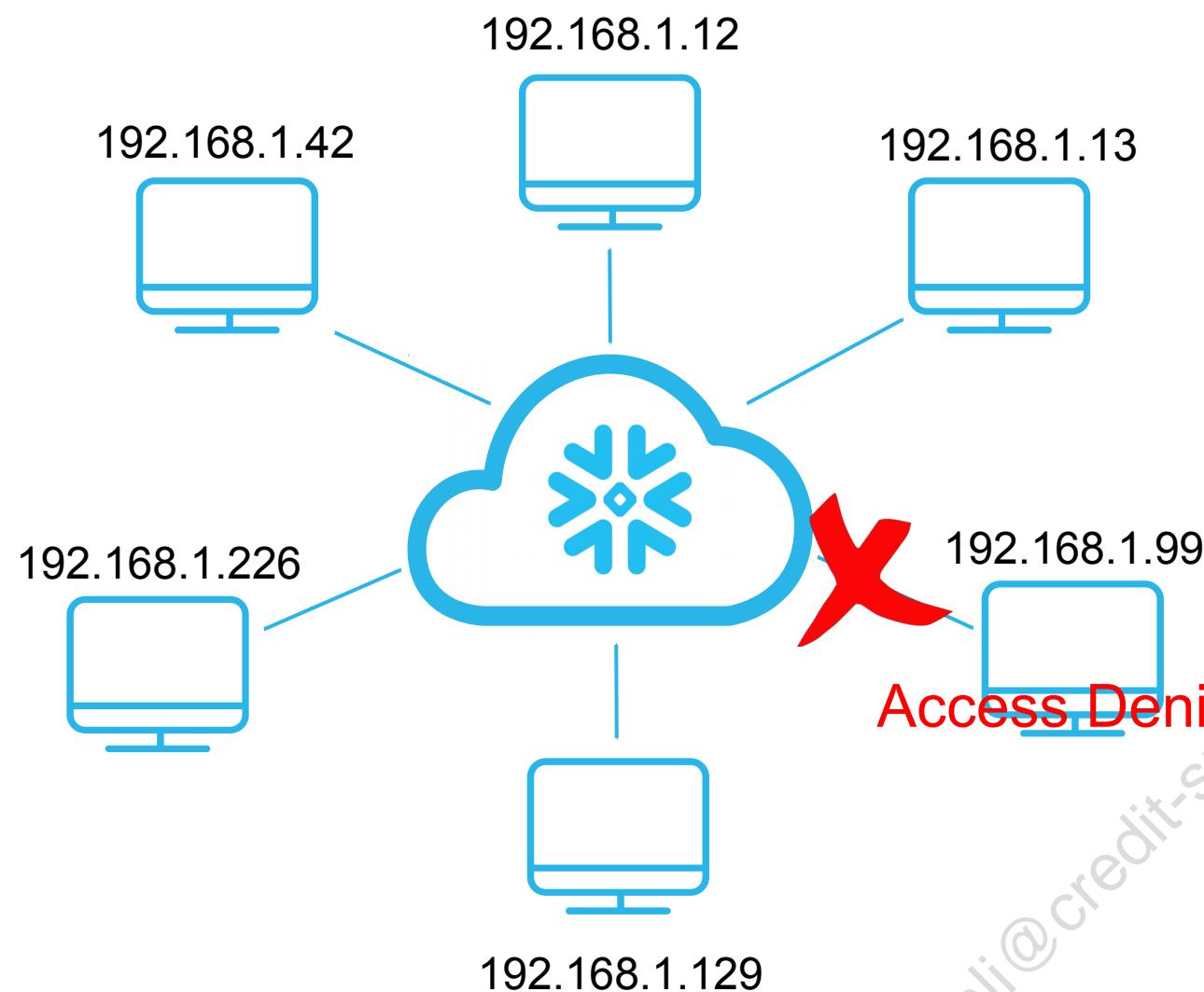
ALTER {ACCOUNT | USER <user_name>}

SET NETWORK_POLICY = <policy_name>;

- Addresses not in the ALLOWED_IP_LIST will be blocked
 - If you also need to block one or more addresses in your ALLOWED list, specify those in the BLOCKED_IP_LIST
- You must set the network policy for a USER or ACCOUNT before it becomes active



SAMPLE NETWORK POLICY



```
CREATE NETWORK POLICY acct_policy  
ALLOWED_IP_LIST = ('192.168.1.0/24')  
BLOCKED_IP_LIST = ('192.168.1.99');
```

ALTER ACCOUNT

```
SET NETWORK_POLICY = acct_policy;
```

- The blocked list is applied first; if an IP address is in both the blocked and allowed lists, it will be blocked
- A network policy can also be set for individual users

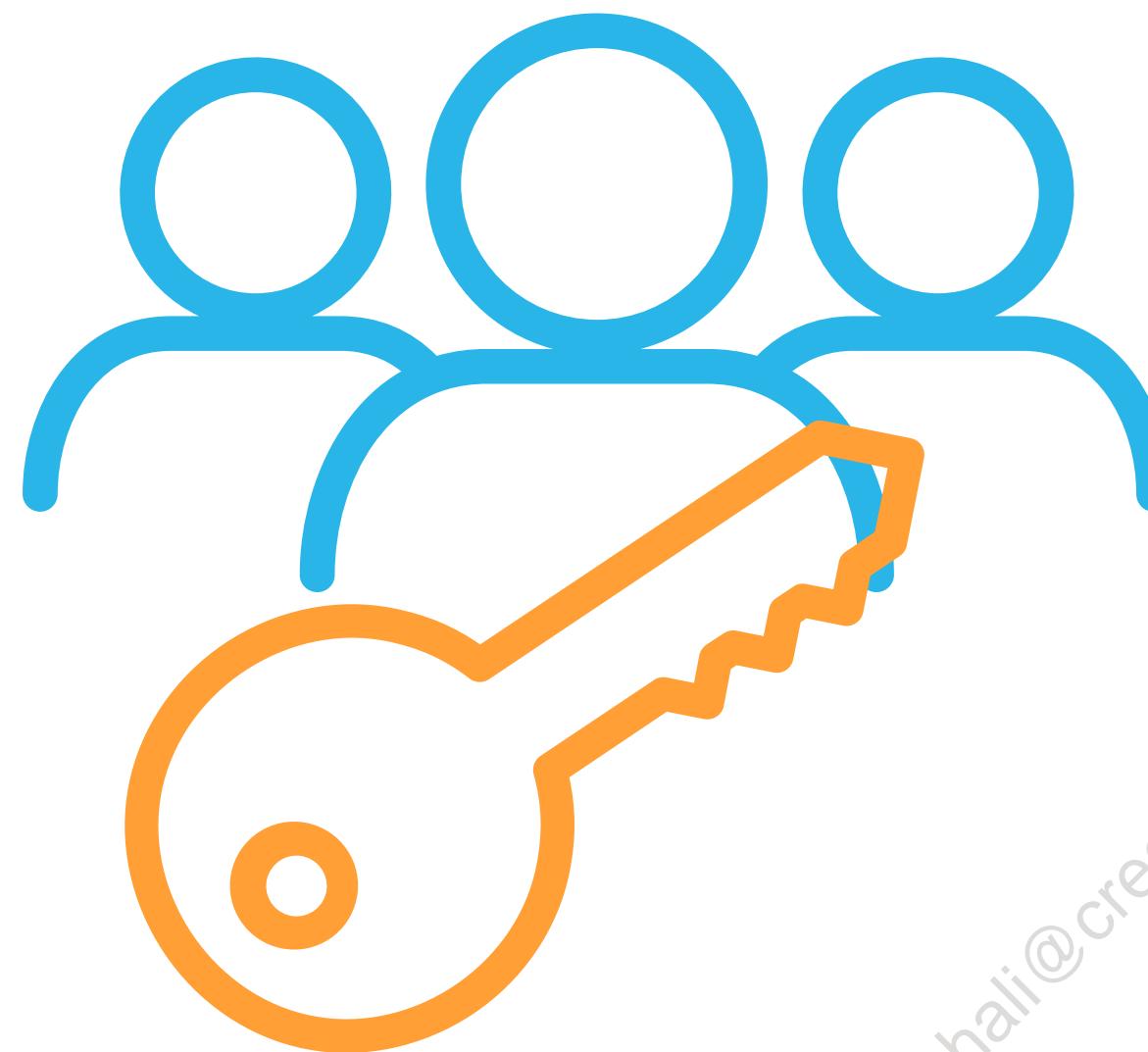
ALTER USER jerry

```
SET NETWORK_POLICY = jerry_pol;
```



PASSWORD POLICY

AUTHENTICATION

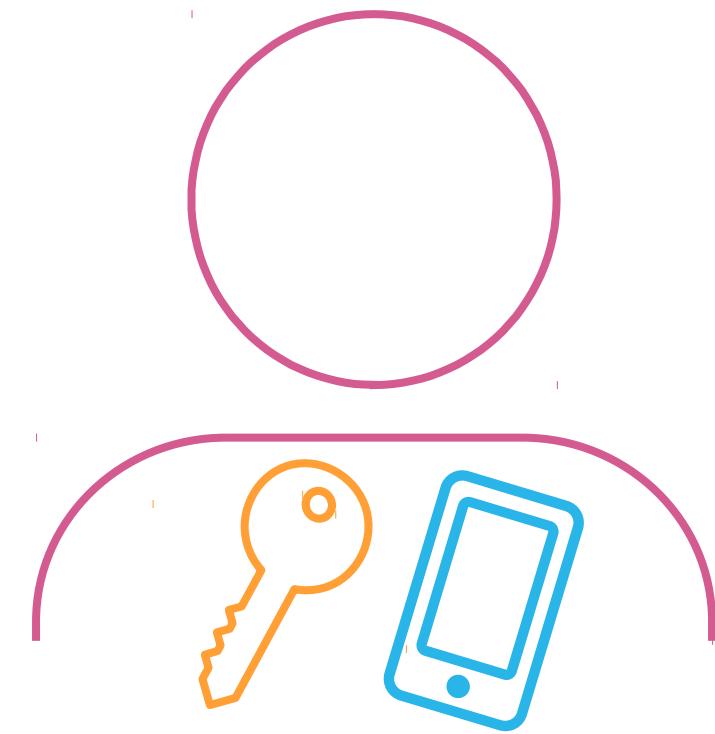


- Absolute Minimum:
 - Must be 8 characters long
 - At least one digit
 - At least one upper/lower case letter

Snowflake strongly recommends that users enable Multi-Factor Authentication (MFA)

MANAGE MFA

- To enable:
 1. Opt-in through UI
 2. Download Duo App to Smartphone or Apple Watch
 3. Follow the instructions
- To disable **temporarily** (user loses or changes phone):
 - USE ROLE SECURITYADMIN;
ALTER USER <name> SET **MINS_TO_BYPASS_MFA** = 5;
- To disable **permanently**:
 - USE ROLE SECURITYADMIN;
ALTER USER <name> **DISABLE_MFA** = true;



ADDITIONAL CONNECTION OPTIONS

- Federated authentication via a SAML 2.0-compliant identity provider (IdP)
- Key-Pair authentication
- Proxy servers
- OCSP (Online Certificate Status Protocol)
 - Fail-Open (default)
 - Response with revoked certification results in failed connection
 - Response with other certificate errors/statuses connects, but notes a WARNING in the logs
 - Fail-Close (must be configured)
 - If client does not receive a valid OCSP CA response for any reason, the connection fails



SET ACCOUNT PARAMETERS

anup.vachali@credit-suisse.com | Nov-2021 | ©Snowflake 2020-do-not-copy



PARAMETERS AND LEVELS

- Parameters can be set to control certain functionality
- There are different levels at which parameters can be set
 - Account
 - Session
 - Object
- Some parameters can be set at multiple levels
 - Example: DATA_RETENTION_TIME_IN_DAYS for Time Travel
 - Generally, the setting at the lower level takes precedence



ACCOUNT-LEVEL PARAMETERS

- Can be set **only** at the account level, by a role with appropriate privileges
 - Cannot be overridden at a lower level

Parameter	Notes
ALLOW_ID_TOKEN	Enable/disable connection caching in browser-based SSO for clients
CLIENT_ENCRYPTION_KEY_SIZE	Used for encryption of staged files
INITIAL_REPLICATION_SIZE_LIMIT_IN_TB	Used for database replication
PERIODIC_DATA_REKEYING	Enable/disable annual data rekeying
PREVENT_UNLOAD_TO_INLINE_URL	If true, must use a stage or storage integration for unloading data
REQUIRE_STORAGE_INTEGRATION_FOR_STAGE_CREATION	Affects creating external stages
REQUIRE_STORAGE_INTEGRATION_FOR_STAGE_OPERATION	Affects using external stages
SSO_LOGIN_PAGE	Enable/disable preview mode for SSO testing



SESSION-LEVEL PARAMETERS

- Most parameters are session-level
- Can be set at the account, user, warehouse, or session level
- Appropriate privileges are required to ALTER ACCOUNT or ALTER USER
- Users can ALTER SESSION to set parameters for their own sessions

Examples:

Parameter	Notes
STATEMENT_TIMEOUT_IN_SECONDS	Account default is 2 days (your admin may have changed)
STATEMENT_QUEUED_TIMEOUT_IN_SECONDS	Account default is never (your admin may have changed)
AUTO_COMMIT	Default is auto-commit
USE_CACHED_RESULT	Whether to use query results cache



OBJECT-LEVEL PARAMETERS

- Can be set at the account, database, schema, table and user level
- Appropriate privileges are required to ALTER ACCOUNT or CREATE/ALTER <object>
- Setting at the "lowest" level will be used

Examples:

Parameter	Notes
STATEMENT_TIMEOUT_IN_SECONDS	Can be set on session, warehouse (object), account and user
STATEMENT_QUEUED_TIMEOUT_IN_SECONDS	Can be set on session, warehouse (object), account and user
DATA_RETENTION_TIME_IN_DAYS	Used for time travel
DEFAULT_DDL_COLLATION	Default is UTF-8

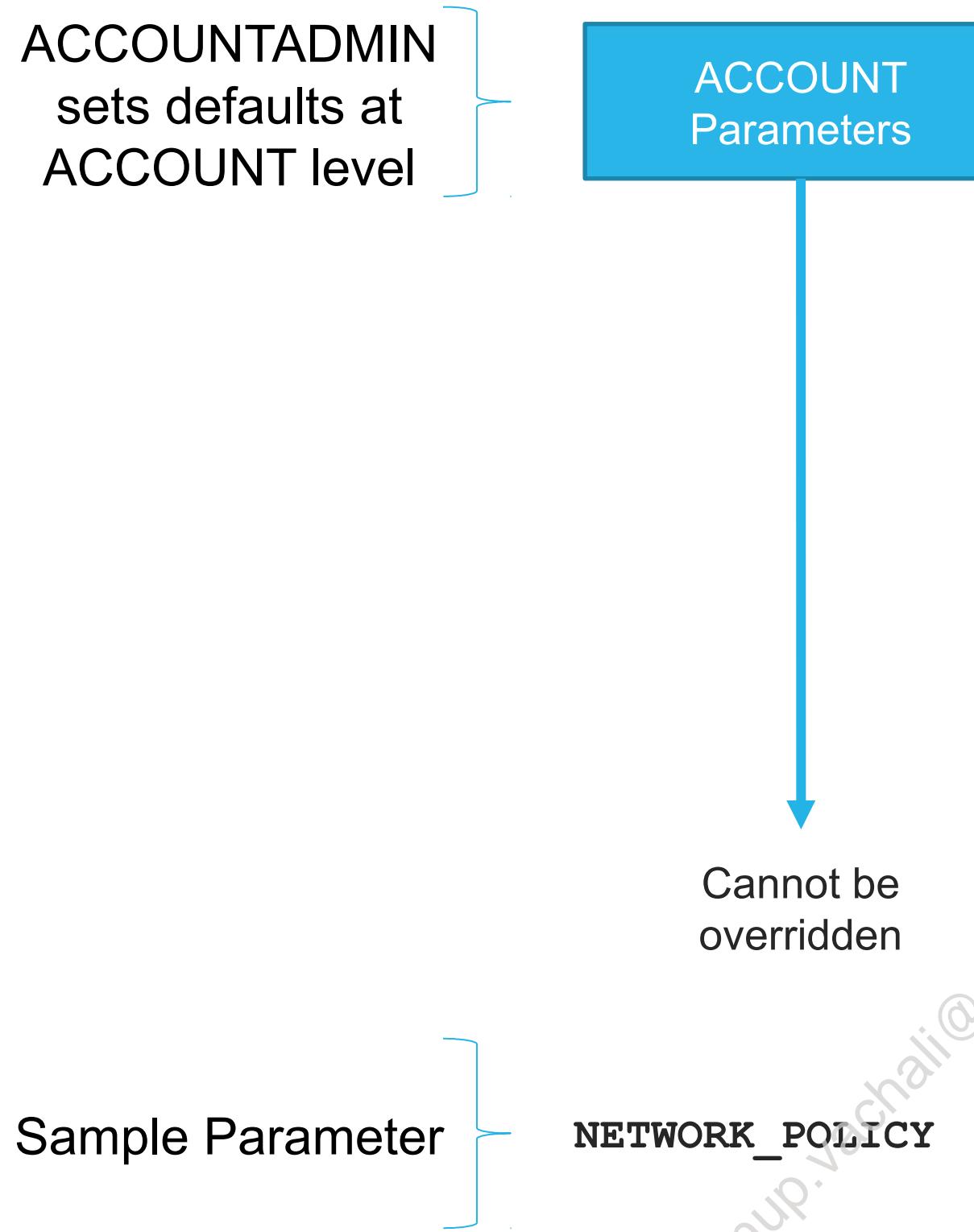


MULTI-LEVEL PARAMETERS

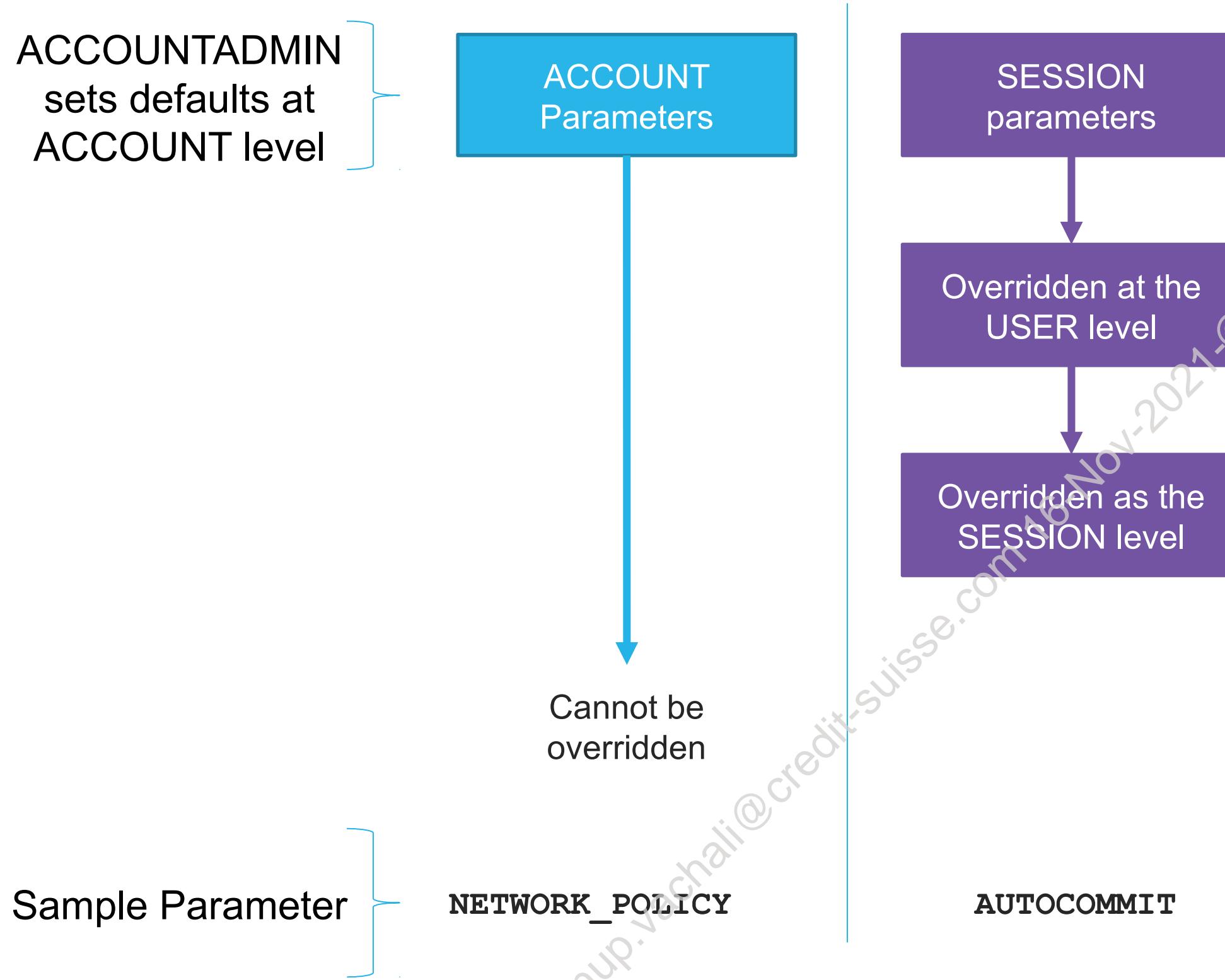
- All parameters can be set at the account level
 - Defaults set by ACCOUNTADMIN (some by SECURITYADMIN as well)
- Most parameters can be set at multiple levels
- The "lowest" level wins
 - A parameter set at the table level takes precedence over the same parameter set at the schema level



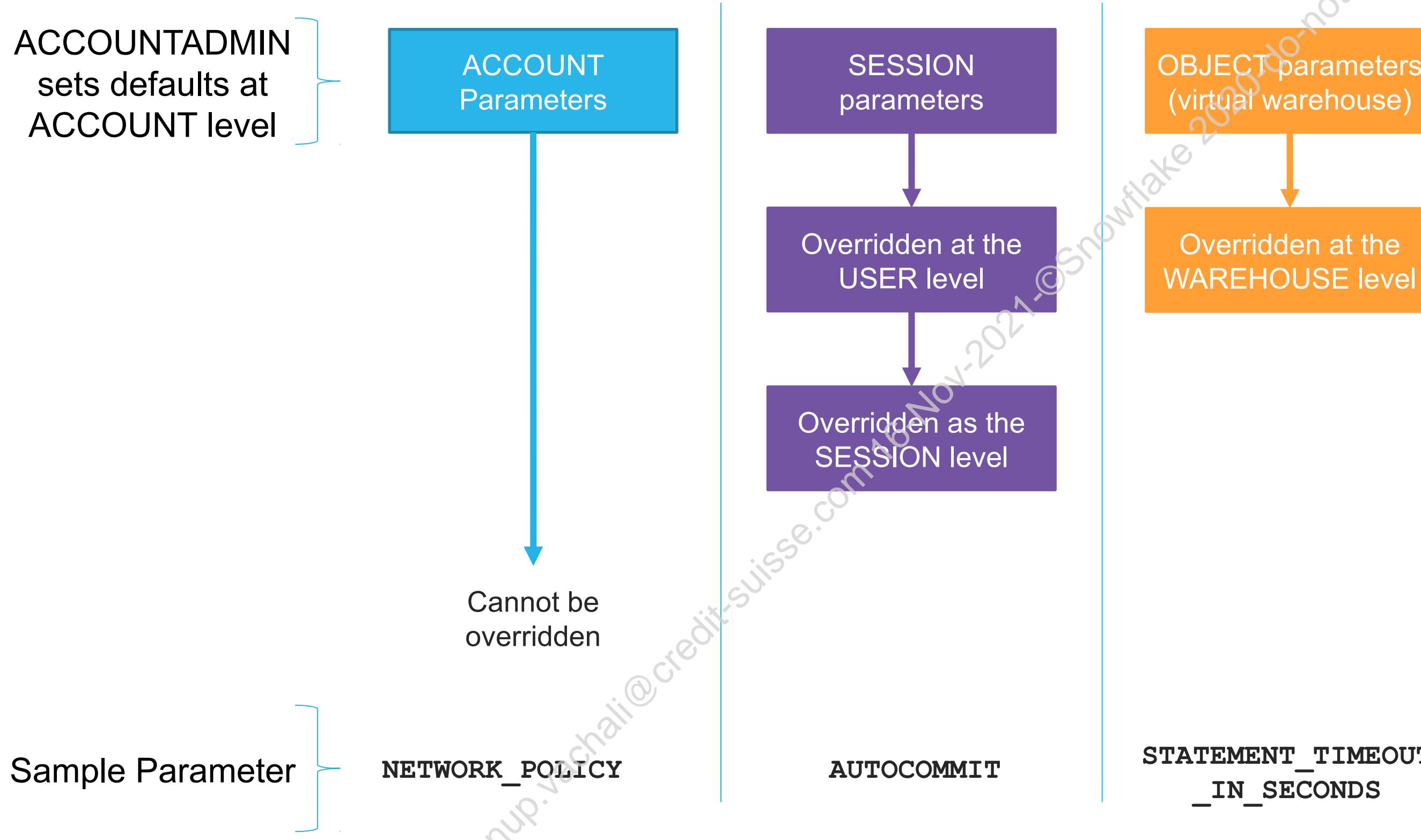
MULTI-LEVEL PARAMETERS



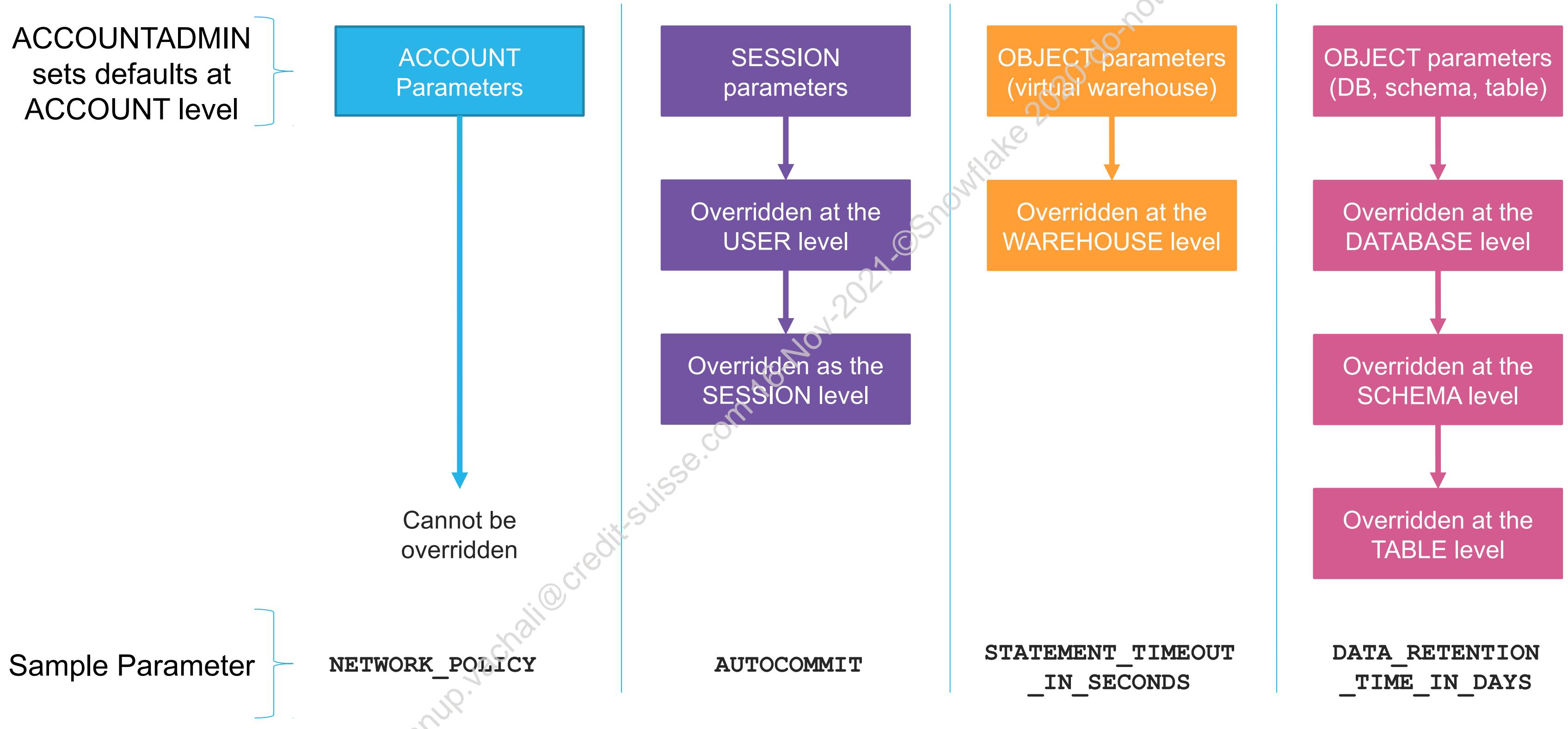
MULTI-LEVEL PARAMETERS



MULTI-LEVEL PARAMETERS



MULTI-LEVEL PARAMETERS



EXAMPLE: TIME TRAVEL SETTING

DATA_RETENTION_TIME_IN_DAYS

ACCOUNT (7 days)

DATABASE_A (default)

SCHEMA1 (30 days)

A_TBL_1 (90 days)



A_TBL_2 (default)



DATABASE_B (5 days)

SCHEMA1 (default)

B_TBL_1 (default)

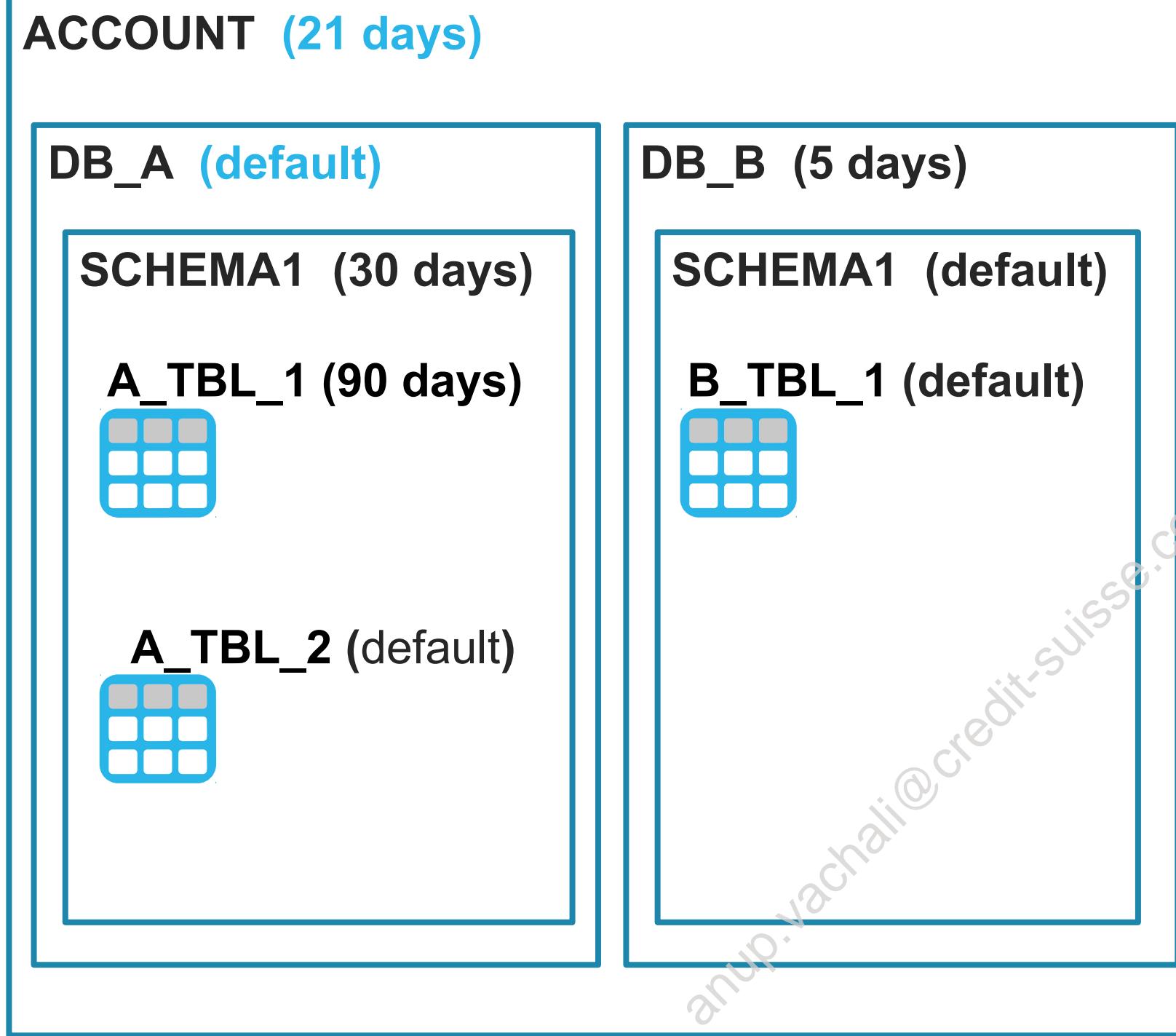


- DATA_RETENTION_TIME_IN_DAYS can be set at the account, database, schema, or table level
- If set at multiple levels, the setting closest to the object will be used
- In the example at left –
 - What will the Time Travel setting be for A_TBL_2? For B_TBL_1?



EXAMPLE: TIME TRAVEL SETTING

DATA_RETENTION_TIME_IN_DAYS



- Changing a default value will affect all objects below it that are using the default
- In this example, if you change the account-level parameter to 21 days, only the setting for DB_A will be affected
 - A_TBL_2 inherits its value from Schema1
 - B_TBL_1 Inherits its value from DB_B



VIEWING AND SETTING PARAMETERS

```
SHOW PARAMETERS FOR { ACCOUNT | WAREHOUSE | USER | SESSION | DATABASE | SCHEMA | TABLE };
```

26 SHOW PARAMETERS FOR SESSION;

27

Results Data Preview Open History

✓ Query ID SQL 32ms 69 rows

Filter result... Columns ▾ ↔

Row	key	value	default	level	description	type
1	ABORT_DETACHE...	false	false		If true, Snowflake ...	BOOLEAN
2	AUTOCOMMIT	true	true		The autocommit p...	BOOLEAN

```
ALTER { ACCOUNT | WAREHOUSE | USER | SESSION | TABLE | DATABASE | SCHEMA }  
SET <parameter> = <value>;
```

```
ALTER { ACCOUNT | WAREHOUSE | USER | SESSION | TABLE | DATABASE | SCHEMA } UNSET <parameter>;
```



LAB EXERCISE: 4

Explore Account Security

10 minutes

- Network Security
- Account-Level Parameters

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



ROLE-BASED ACCESS CONTROL

anup.vachali@credit-suisse.com 16-Nov-2021 @Snowflake 2020-do-not-copy



MODULE AGENDA

- Review of Key Concepts
- Designing Roles
- Use SCIM to Synchronize Users and Roles

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



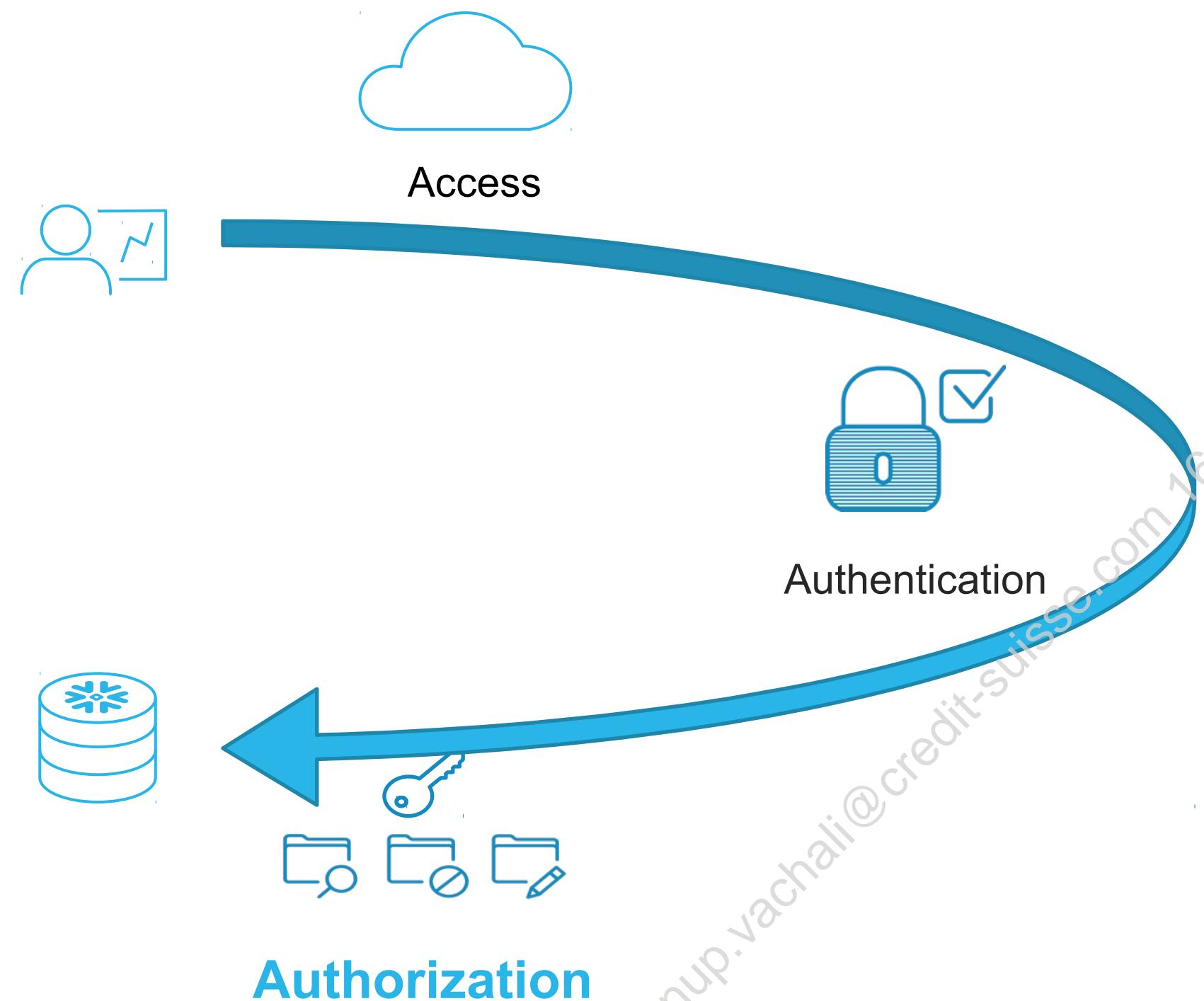
REVIEW OF KEY CONCEPTS

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



AUTHORIZATION

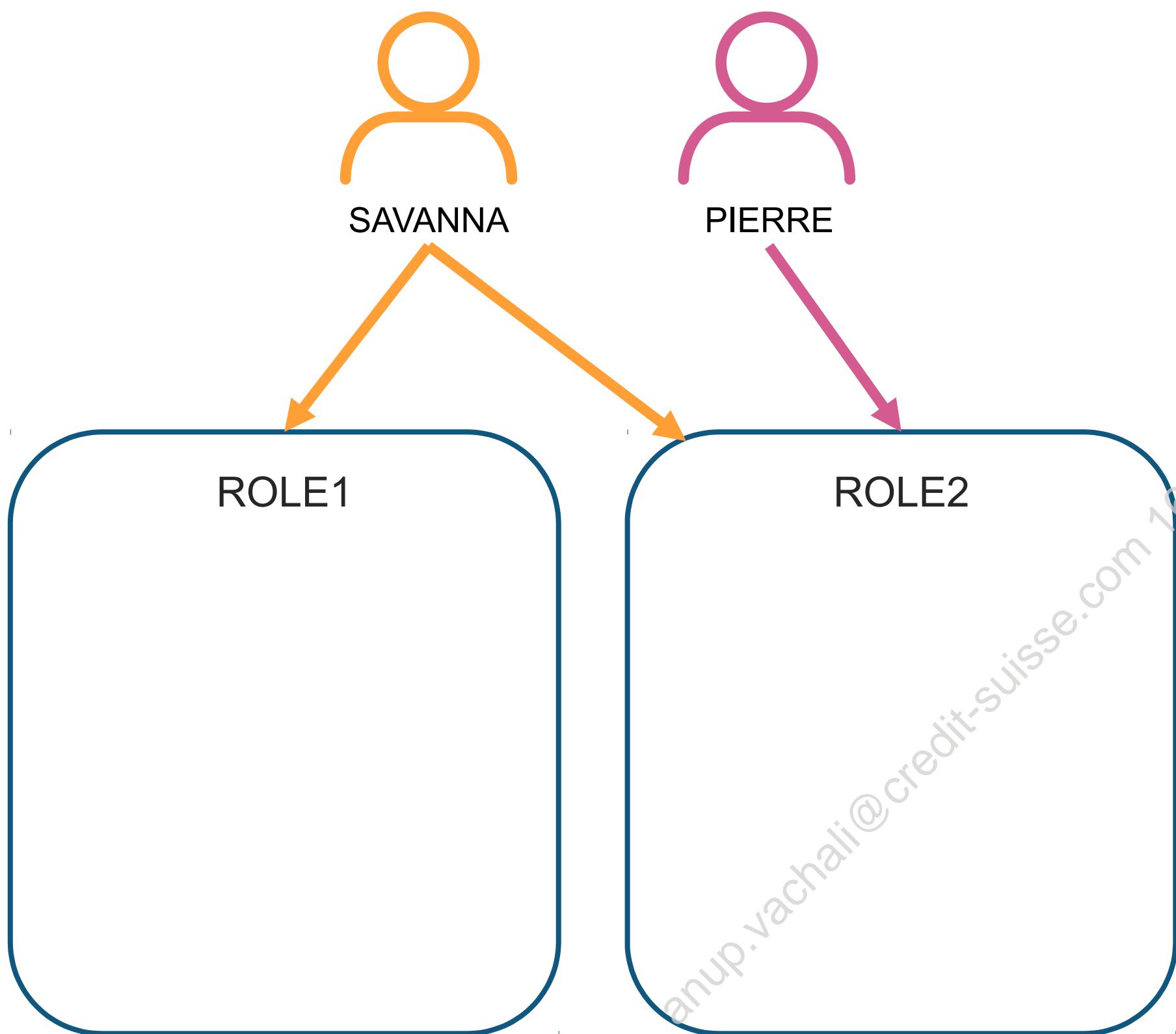
WHO CAN DO WHAT?



- Authorization is defined and enforced through Snowflake's role-based access control



USERS AND ROLES

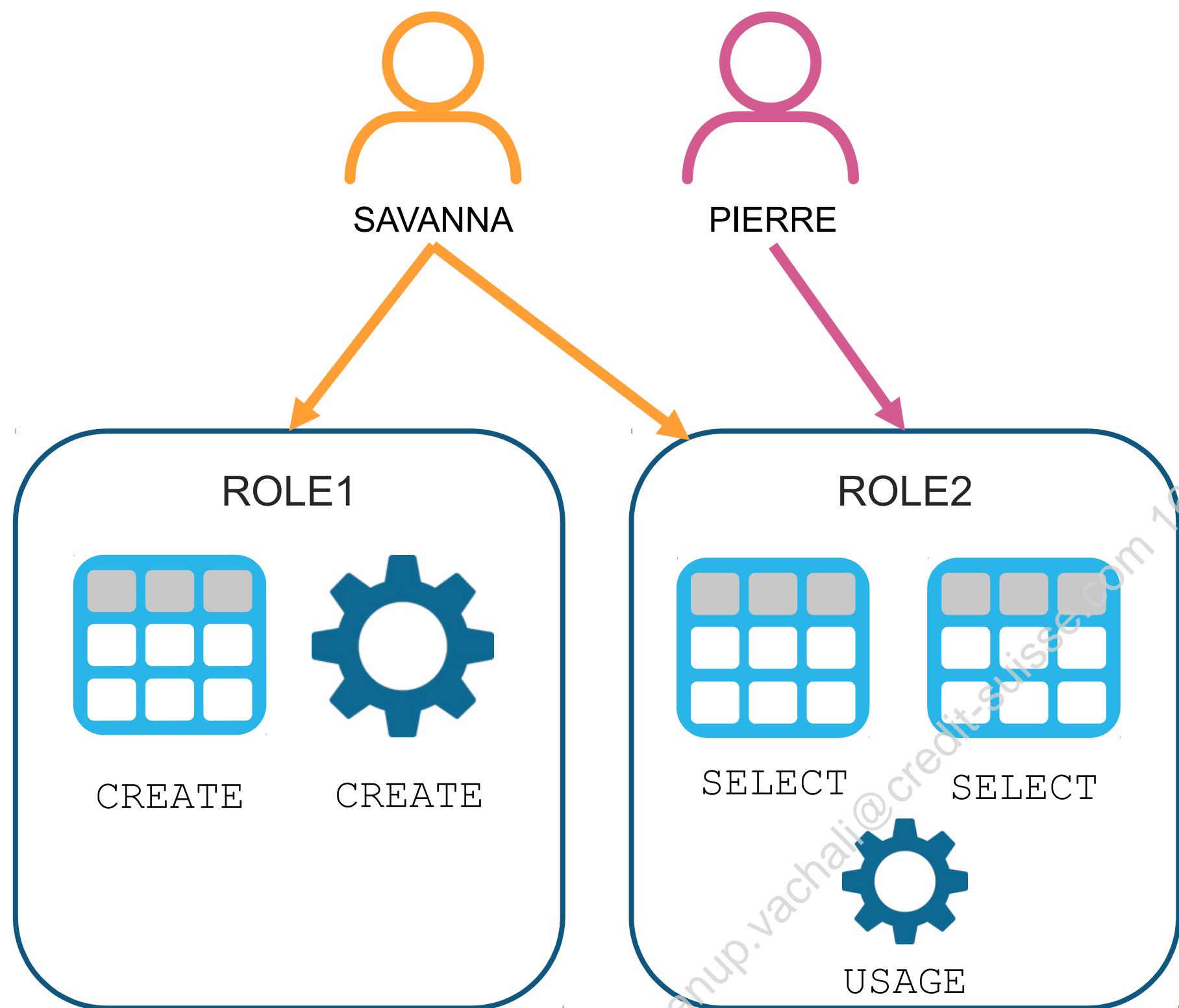


- **Users** are granted access to one or more roles by the **SECURITYADMIN**

```
GRANT ROLE1 TO USER SAVANNA;  
GRANT ROLE2 TO USER SAVANNA;  
GRANT ROLE2 TO USER PIERRE;
```



USERS AND ROLES

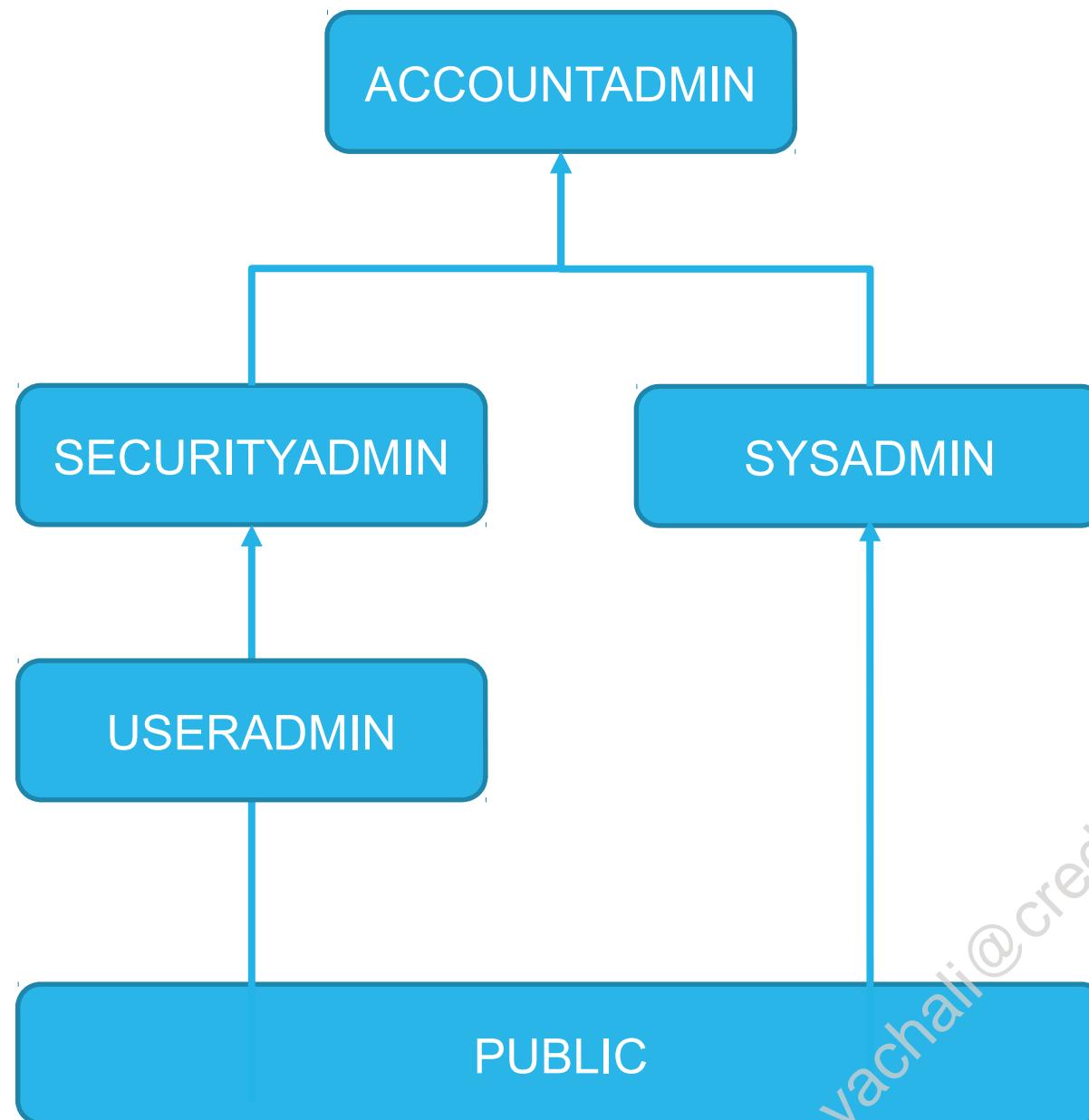


- **Users** are granted access to one or more roles by the **SECURITYADMIN**
- Privileges on objects are granted to **roles** by the **SYSADMIN**
 - Create table
 - Use warehouse
 - Select from view
- Privileges are not granted to users
 - Except for the right to use a role



SYSTEM ROLES

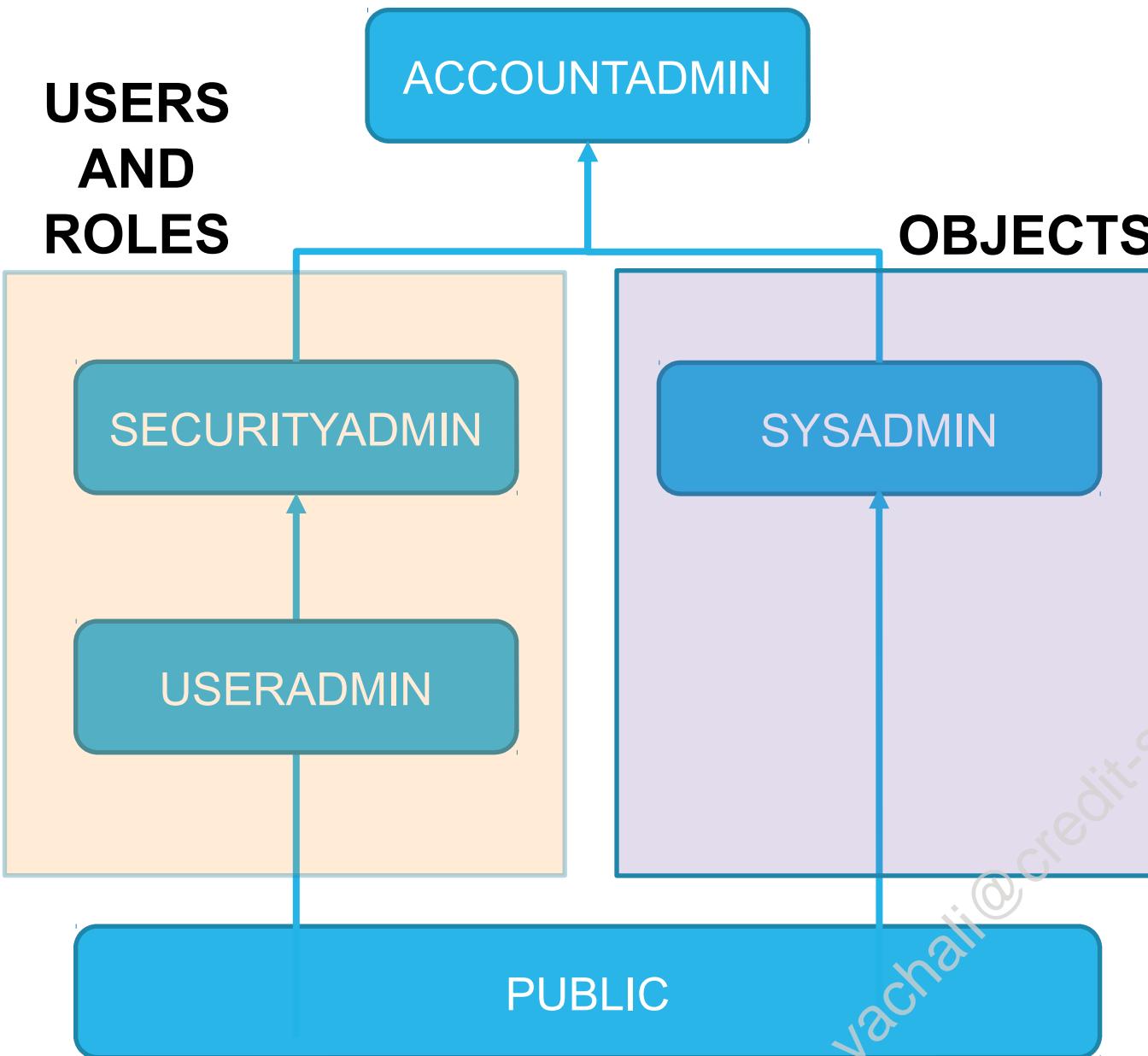
ROLES ARE SIMILAR TO GROUPS



- **ACCOUNTADMIN** – administers account-level parameters and features
- **SYSADMIN** – creates objects, assigns privileges to roles
- **SECURITYADMIN** – creates users and roles, managed grants
 - **USERADMIN** – can also be used to create users and roles, but cannot manage grants
- **PUBLIC** – can log into the system but has no other privileges



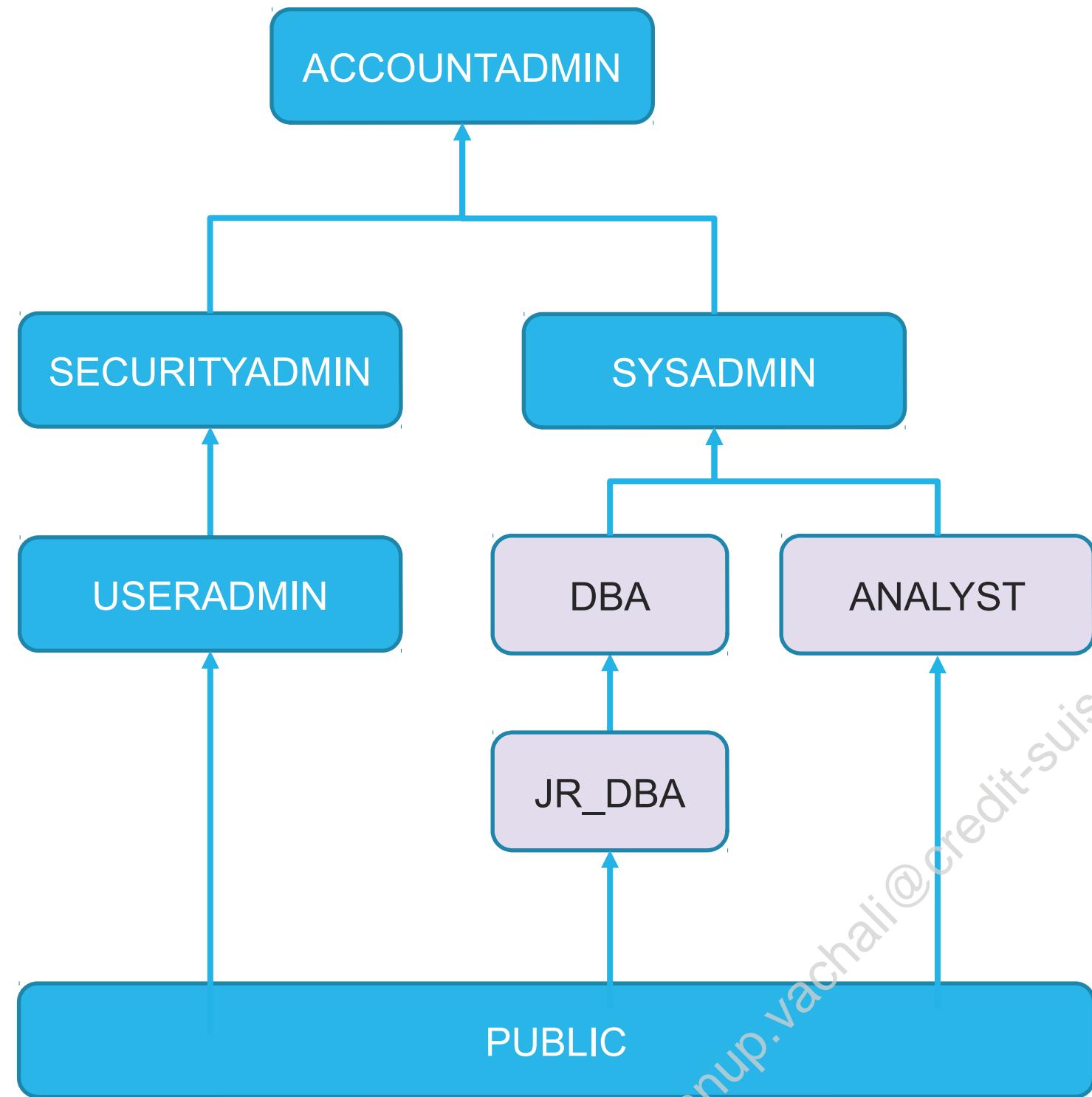
DIVISION OF RESPONSIBILITY



- System roles separate user and role management from object management
- **SECURITYADMIN** and **USERADMIN** do not have access to data



CUSTOM ROLES AND INHERITANCE



- SECURITYADMIN creates custom roles for the account, and places them in the SYSADMIN hierarchy
- Privileges roll UP the hierarchy
 - When someone uses the DBA role, they get the privileges of both DBA and JR_DBA



OWNERSHIP

- When a user creates an object, that object is owned by the **role** that was used to create it
- All users who have access to the creating role are co-owners of the object
- An object owner can do anything with it
 - If a role creates a schema, then the role can create tables in the schema (without specifically being granted the `CREATE TABLE` privilege)
 - The owning role can grant privileges on that object to other roles

DBA

```
CREATE TABLE great_data (col1 INT);  
GRANT SELECT ON TABLE great_data TO ROLE JR_DBA;
```



MANAGED ACCESS SCHEMAS

CENTRALIZE OR LOCK DOWN PRIVILEGE MANAGEMENT FOR OBJECTS

- Designed to centralize management of grants for objects

Regular Schemas	Managed Access Schemas
<p>Object owners can grant access to their objects, including the right to further grant access to others</p> <pre>GRANT SELECT ON ALL TABLES IN SCHEMA <X> TO ROLE <R> WITH GRANT OPTION;</pre>	<p>Only the schema owner, SECURITYADMIN, or a custom role with MANAGE GRANTS privileges can grant access to the objects in the schema</p>
<p>Code example:</p> <pre>USE DATABASE myDB; CREATE SCHEMA mySchema;</pre>	<p>Code example:</p> <pre>USE DATABASE myDB; CREATE SCHEMA mySchema WITH MANAGED ACCESS;</pre>

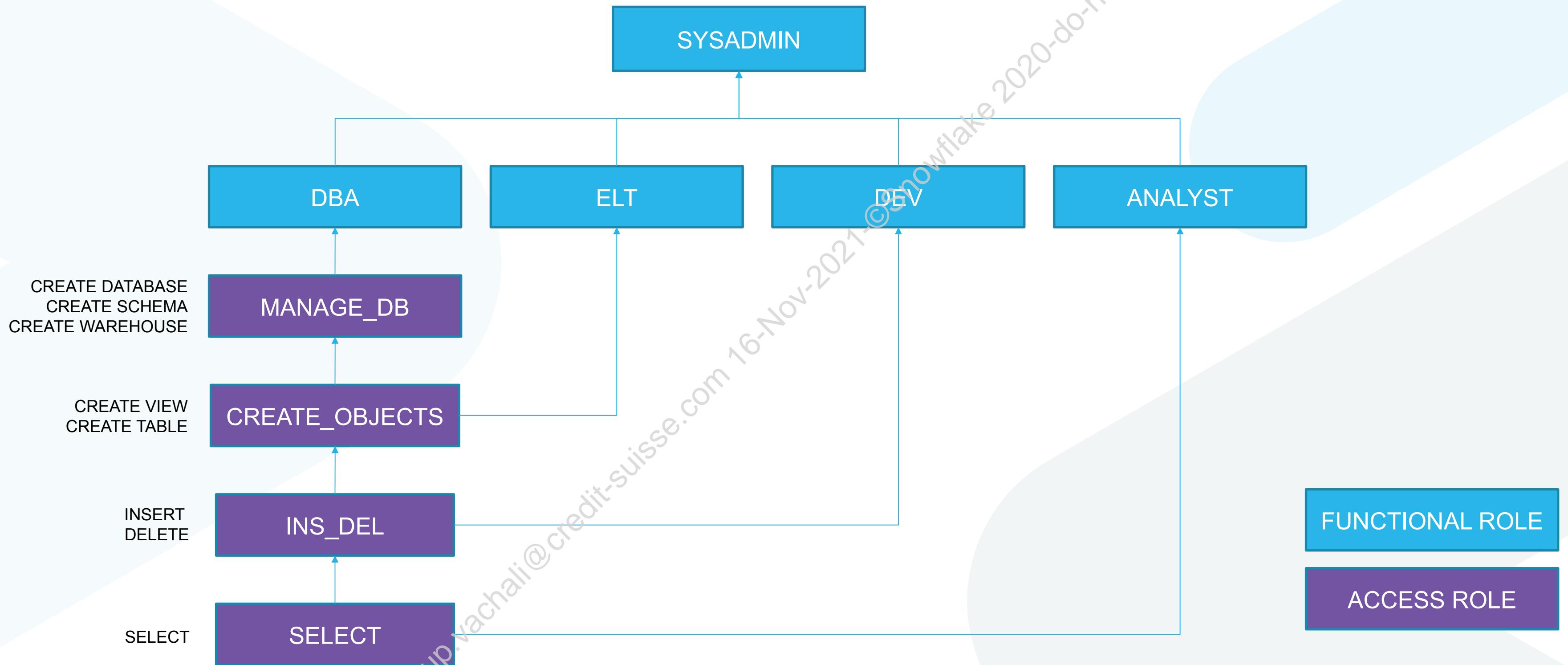


WHAT YOU REALLY NEED TO KNOW

- 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



LAB EXERCISE: PRE-CAP



LAB EXERCISE: 5

Explore Users, Roles, and Privileges

25 minutes

- Create and Configure Roles
- Verify the Current Hierarchy
- Create and Grant Privileges on Warehouses
- Explore and Test Ownership and Privileges
- Explore and Test Revoking Privileges
- Clean Up



USE SCIM TO SYNCHRONIZE USERS AND ROLES



anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy

WHY SCIM?

PROBLEM

- Need to maintain users and roles in Snowflake, as well as users and groups in Active Directory or another Identity Provider (IdP)
- Snowflake users and roles must be manually created and maintained



WHY SCIM?

PROBLEM

- Need to maintain users and roles in Snowflake, as well as users and groups in Active Directory or another Identity Provider (IdP)
- Snowflake users and roles must be manually created and maintained



SOLUTION

- Use SCIM to integrate with IdPs
- User and group information is pushed to Snowflake users and roles



WHAT IS SCIM?

SYSTEM FOR CROSS-DOMAIN IDENTITY
MANAGEMENT



- SCIM is an IETF (Internet Engineering Task Force) standard
- Provides:
 - Standard schema definitions for User and Group
 - Standard REST-based protocol for request-response
 - Standard operations
 - Create, Read, Update, Delete and Search



SCIM IN SNOWFLAKE



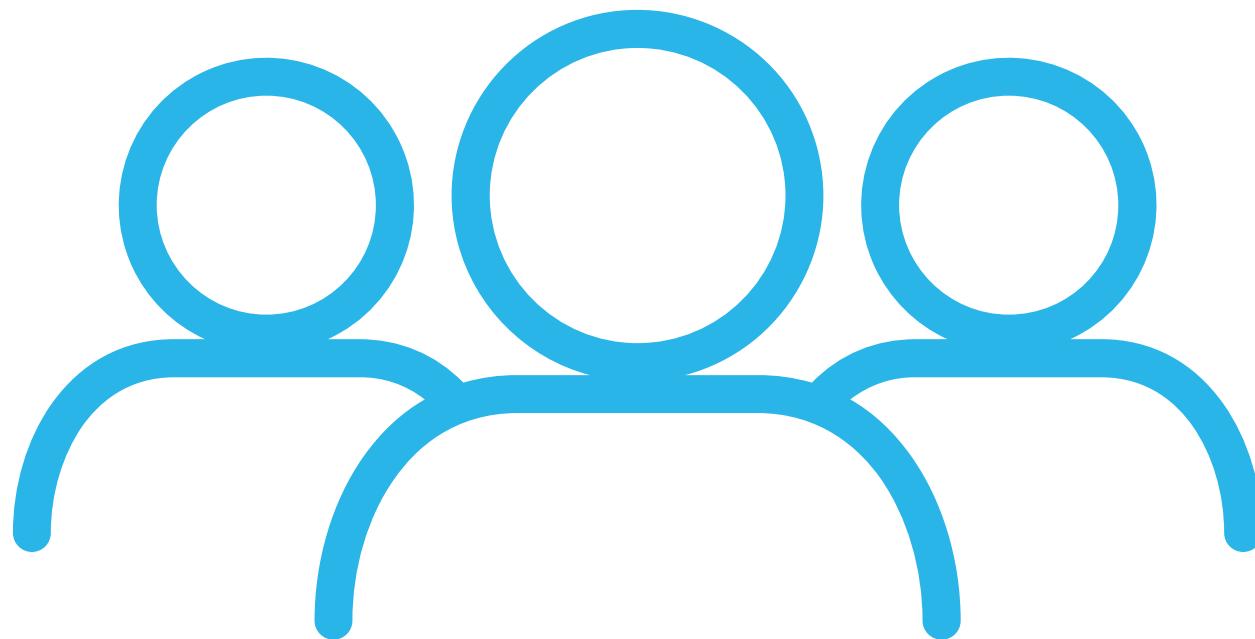
anup.vachali@credit-suisse.com 16-Nov-2021 © Snowflake 2020 - do not copy

- Snowflake supports SCIM 2.0
- Connectors available for:
 - Okta
 - Microsoft Azure Active Directory
- Use custom applications to integrate with other identify providers
- Identify providers provision, manage, and synchronize users and groups to users and roles in Snowflake



MANAGING USERS

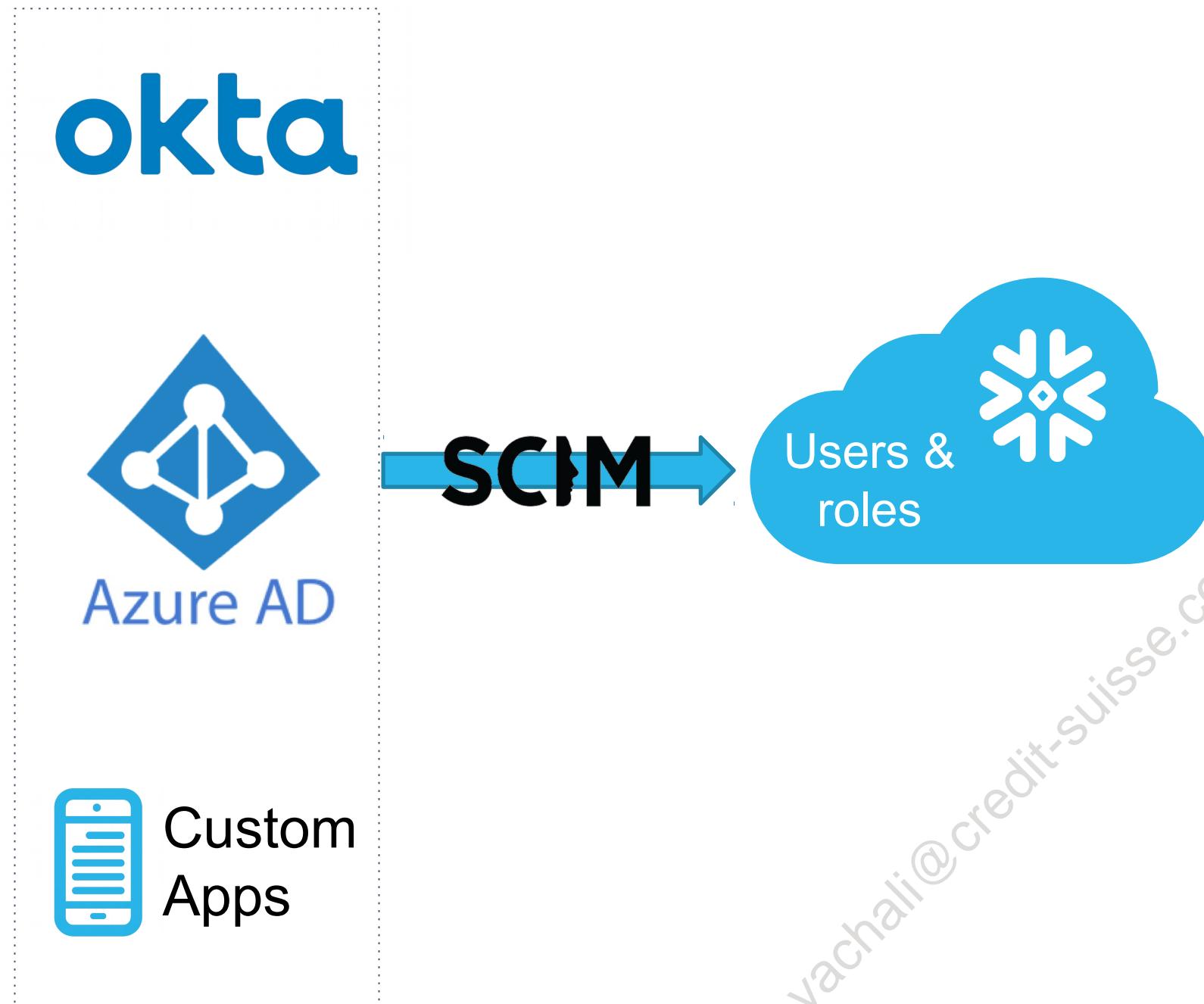
THROUGH SCIM



- Create and activate users
- Update user attributes
- Deactivate users
- Pass user attributes in REST API requests
 - Updated in Snowflake almost immediately



MANAGING ROLES AND GROUPS



- Direct one-to-one mapping of groups to Snowflake roles
 - Nested groups are not supported
- Pass in role attributes via REST API requests
- Import roles from:
 - Okta
 - Azure AD
 - Custom applications



SUMMARY

- SCIM REST APIs integrate with IdPs to bring in users and roles from external systems
- Direct connectors for Okta and Azure Active Directory
- Custom programs can be written for other IdPs
- Some restrictions:
 - Does not support nested role/group mapping
 - Users and roles created in Snowflake outside of SCIM cannot be sync-ed back to IdPs
 - SCIM Access Token must be regenerated and updated in the SCIM client every 6 months:

```
SELECT SYSTEM$GENERATE_SCIM_ACCESS_TOKEN( '<role_name>' ) ;
```



WORK WITH DATA

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy

LOAD DATA

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



MODULE AGENDA

- Overview
- Bulk Loading
- Bulk Loading Recommendations
- Transforming Data on Load
- Validation and Error Handling
- Continuous Data Loading



anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy

OVERVIEW

anup.vachali@credit-suisse.com 16-Nov-2021-©Snowflake 2020-do-not-copy

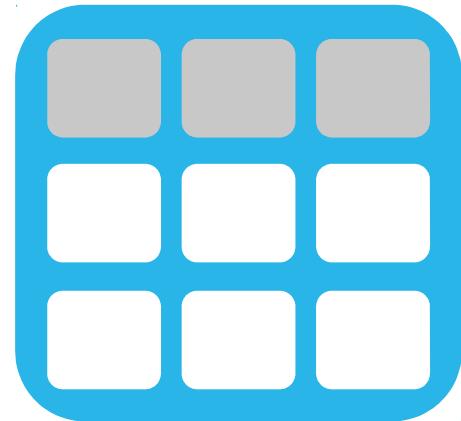


DATA LOADING OPTIONS

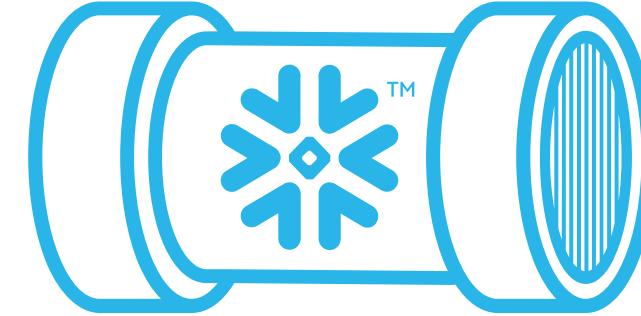
USING NATIVE SNOWFLAKE FUNCTIONALITY

- Bulk loading using COPY INTO
- Continuous loading using Snowpipe

```
COPY INTO <table> . . .
```



```
CREATE PIPE <pipe>  
AS COPY INTO...
```



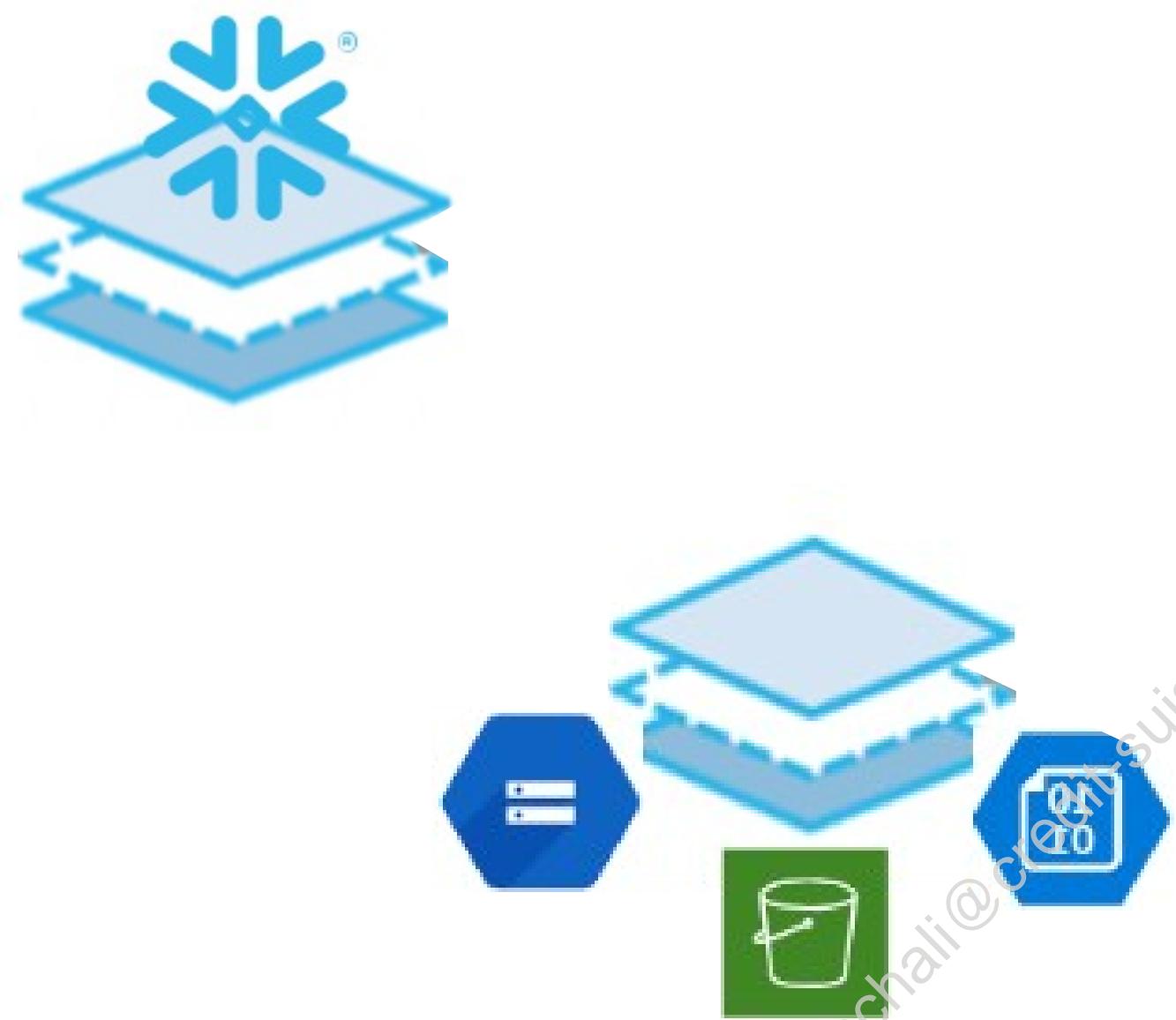
DATA LOADING OPTIONS

USING THIRD-PARTY TOOLS



DATA LOADING OBJECTS

STAGE



- Identifies where the input files are stored
- Internal stages are managed by Snowflake, and use storage in your Snowflake account
- External stages are managed by you, and use storage in your cloud provider account

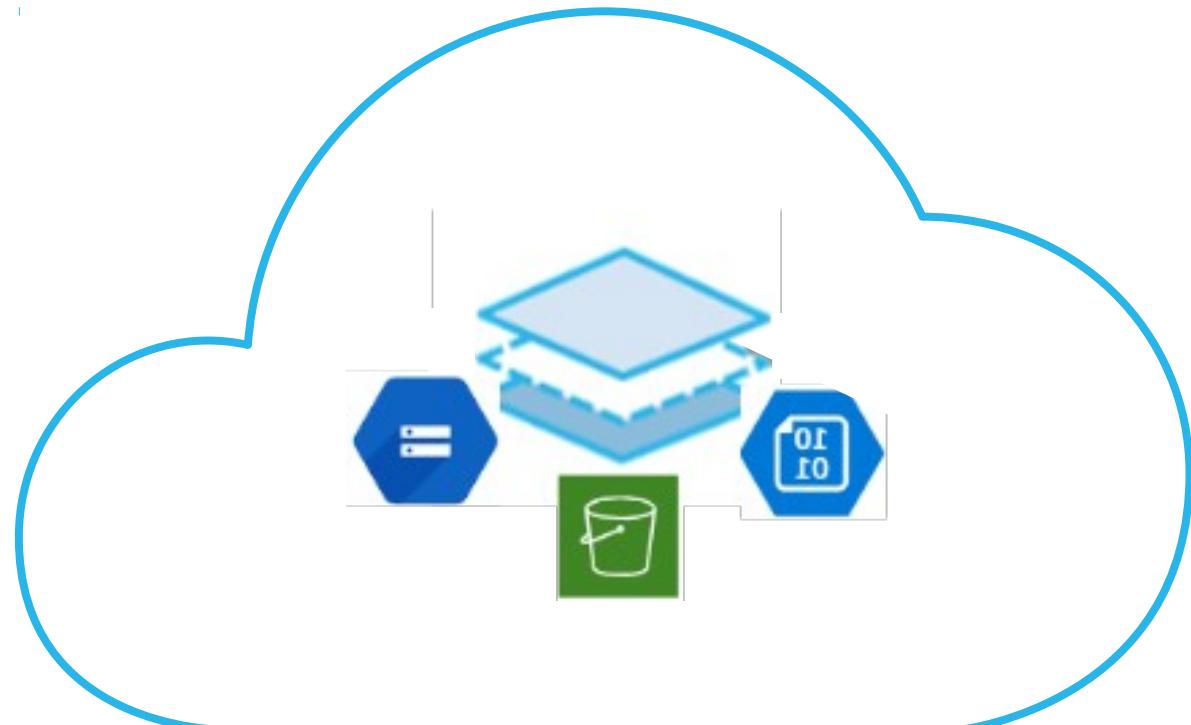


EXTERNAL NAMED STAGES

- External stages are always named stages
- You must provide information required to access the location

```
CREATE STAGE my_external_stage  
  URL = '<location>'  
  CREDENTIALS = (<credentials>)  
  ENCRYPTION = (<encryption info>);
```

- Can be located in a different region, or on a different cloud provider, from the Snowflake account
 - External stages may incur data transfer charges from cloud provider



TYPES OF INTERNAL STAGES

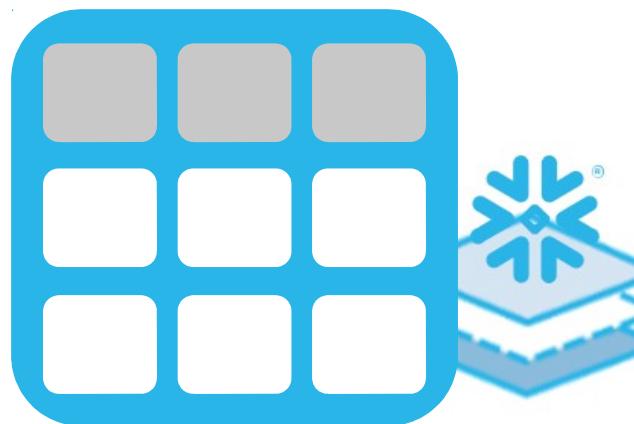


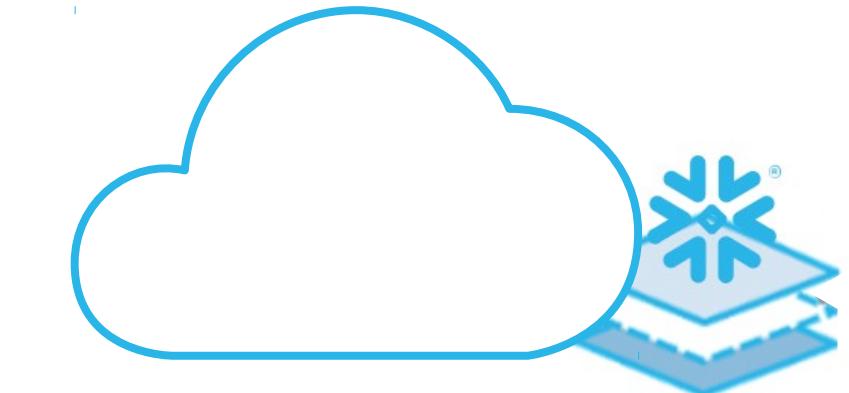
Table Stage

@%<TABLE_NAME>



User Stage

@~



Named Stage

@<STAGE_NAME>

Created automatically

CREATE STAGE my_stage;

DATA LOADING OBJECTS

FILE FORMAT

```
Header line  
Header line  
Col1|Col2|Col3|Col4  
123|ABC|987|FED  
234|BCD|876|  
345|CDE|765|DCB
```

CREATE FILE FORMAT pipe_delimited

```
TYPE = CSV  
FIELD_DELIMITER = '|'  
SKIP_HEADER=3  
EMPTY_FIELD_AS_NULL = TRUE;
```

- Describes the structure of the input files
 - Base type, plus optional modifiers
- Base types available:
 - CSV
 - JSON
 - Parquet
 - XML
 - Avro
 - ORC
- Modify default settings for the chosen base type, if required



SPECIFYING THE FILE FORMAT

- A file format must be provided for every load command
- Two places the file format can be specified
 - Can be included in the COPY INTO command (takes precedence)

```
COPY INTO...FILE_FORMAT = my_file_format;
```



- Can be defined as part of a table or named stage

```
CREATE STAGE...FILE_FORMAT = my_file_format;
```



```
CREATE TABLE...STAGE_FILE_FORMAT = my_file_format;
```

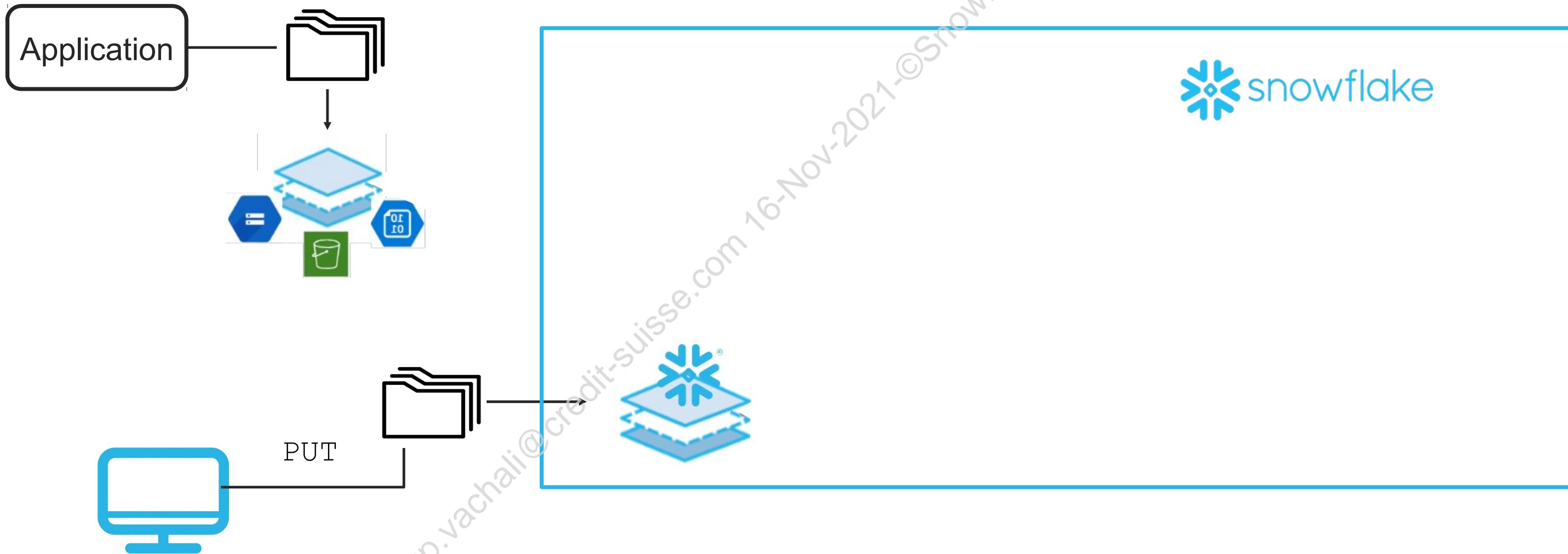
BULK LOADING

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



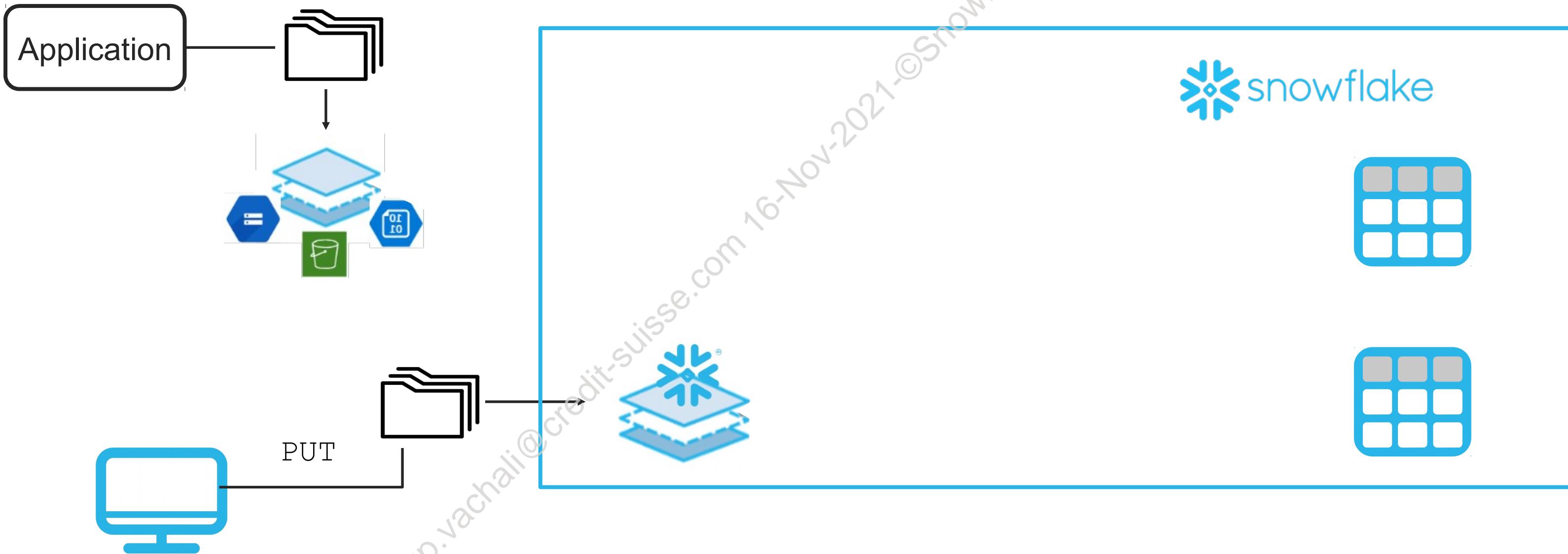
BASIC WORKFLOW

1. Get data files into a stage



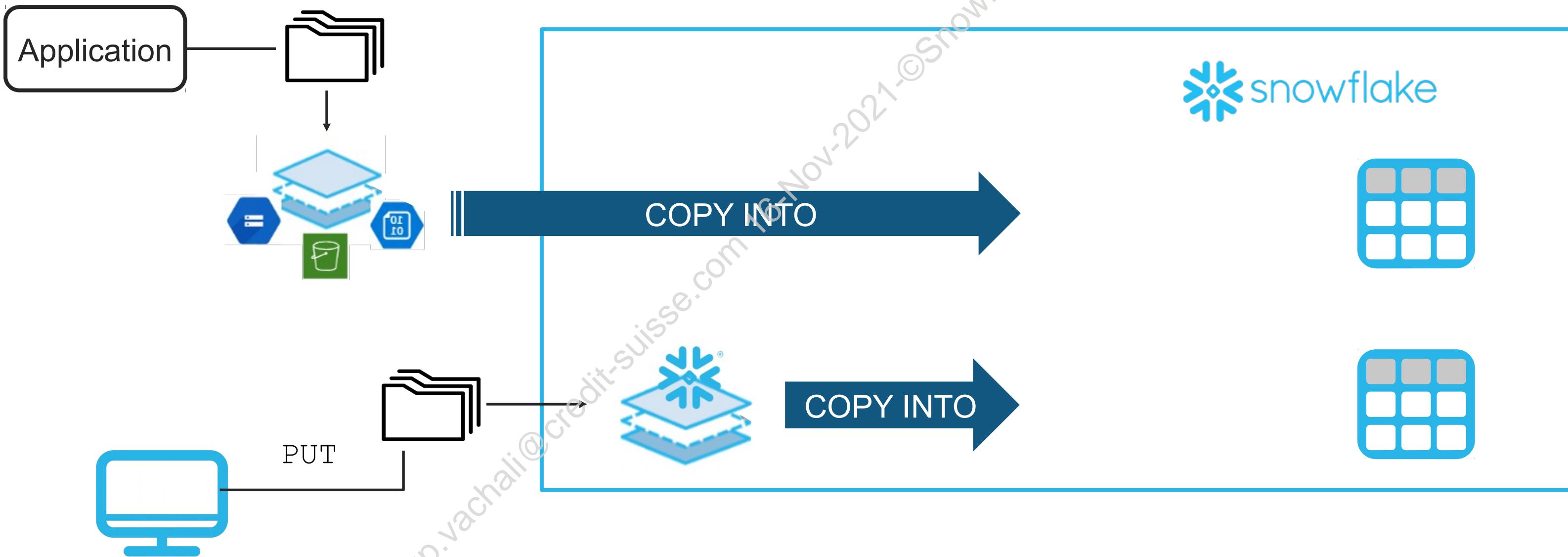
BASIC WORKFLOW

2. Create the target table, if it does not already exist



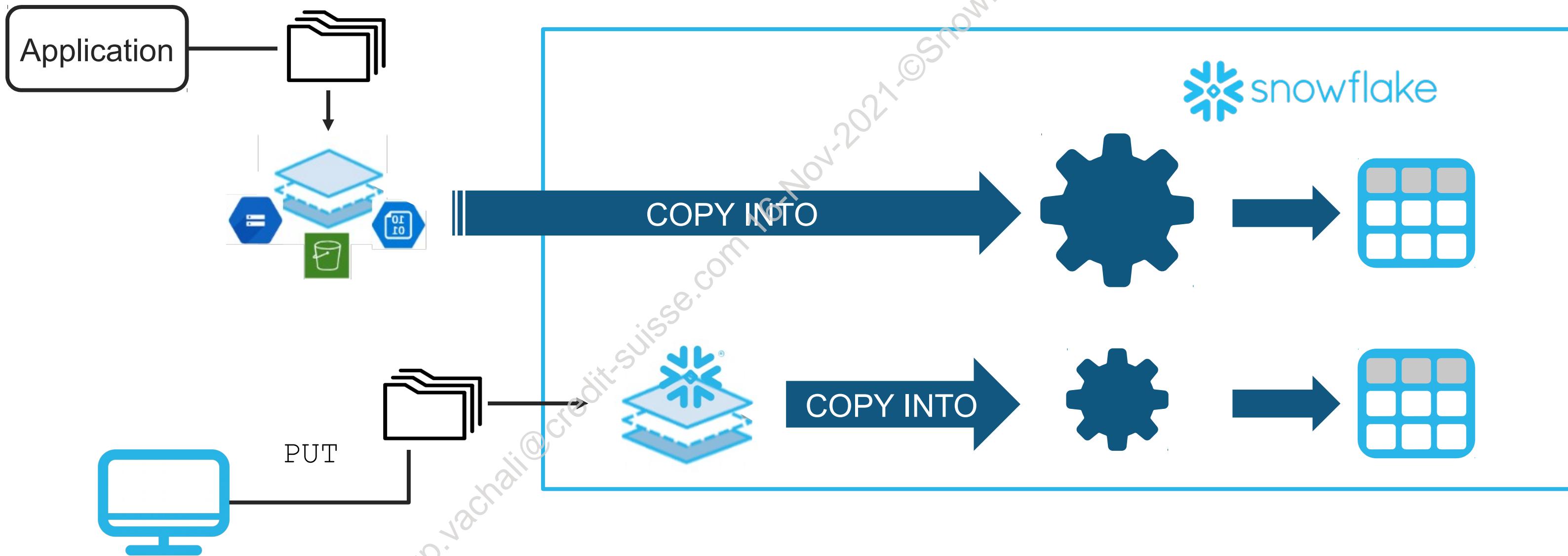
BASIC WORKFLOW

3. Run COPY INTO to load the data



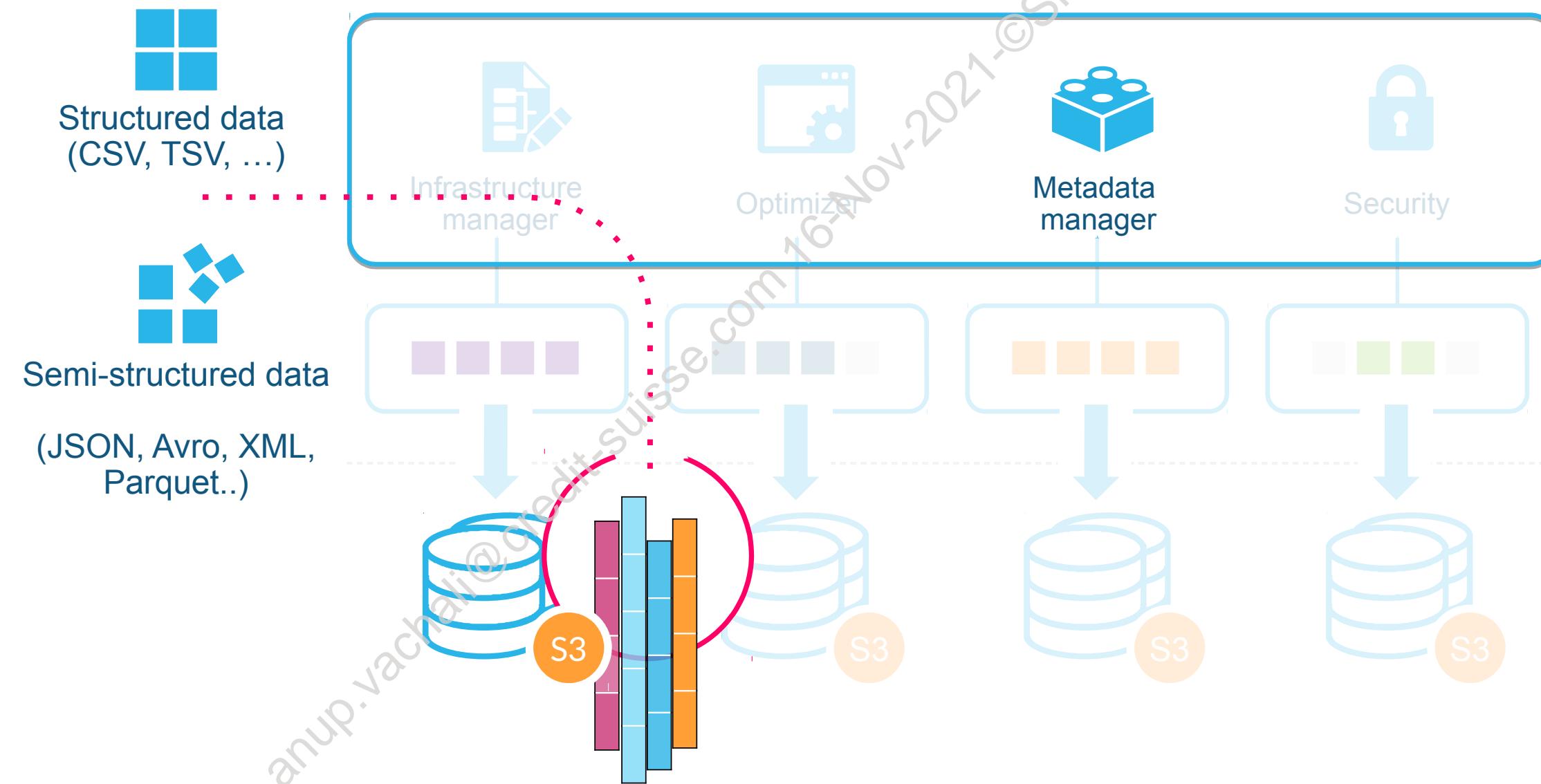
BASIC WORKFLOW

4. Snowflake uses the designated virtual warehouse to load the data using your file format

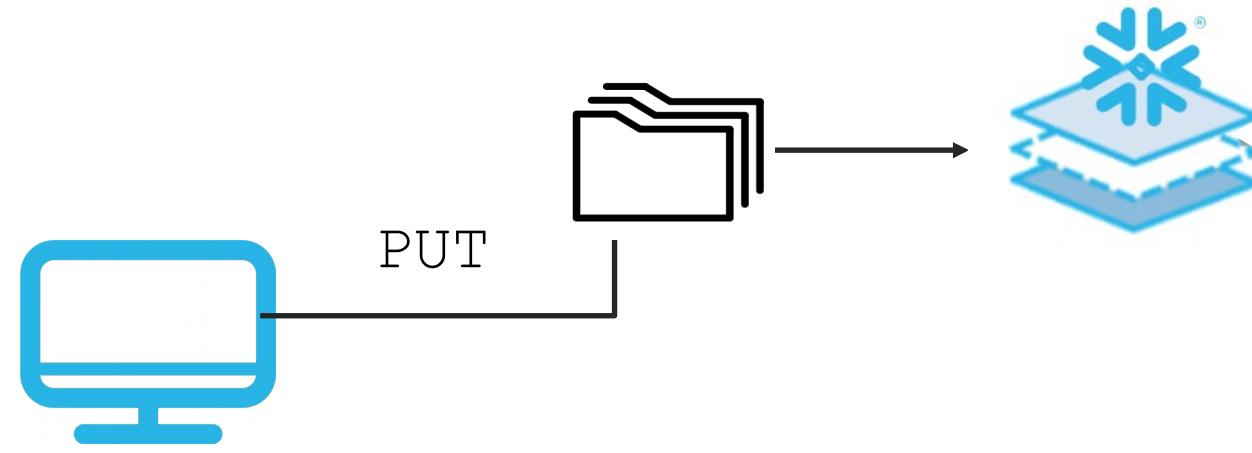


BASIC WORKFLOW

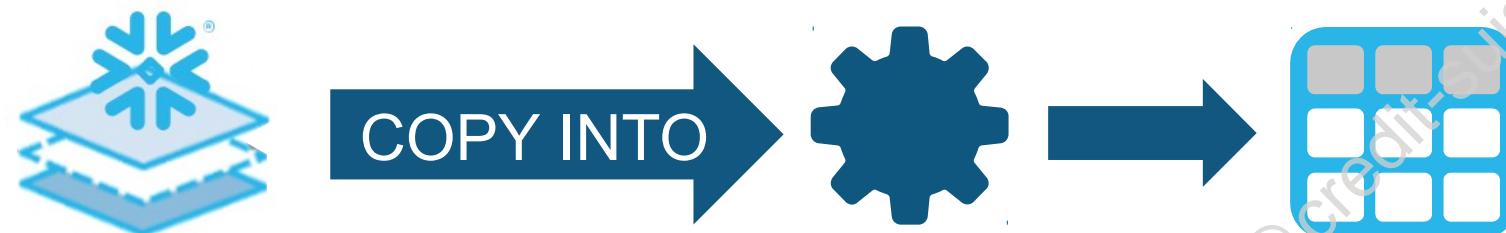
- As data is loaded it is compressed and encrypted, and metadata is collected/updated for each micro-partition and table



EXAMPLE FROM LOCAL STORAGE



```
PUT FILE:///data/data.csv @my_stage;  
PUT FILE:///data/*sales* @my_stage;
```

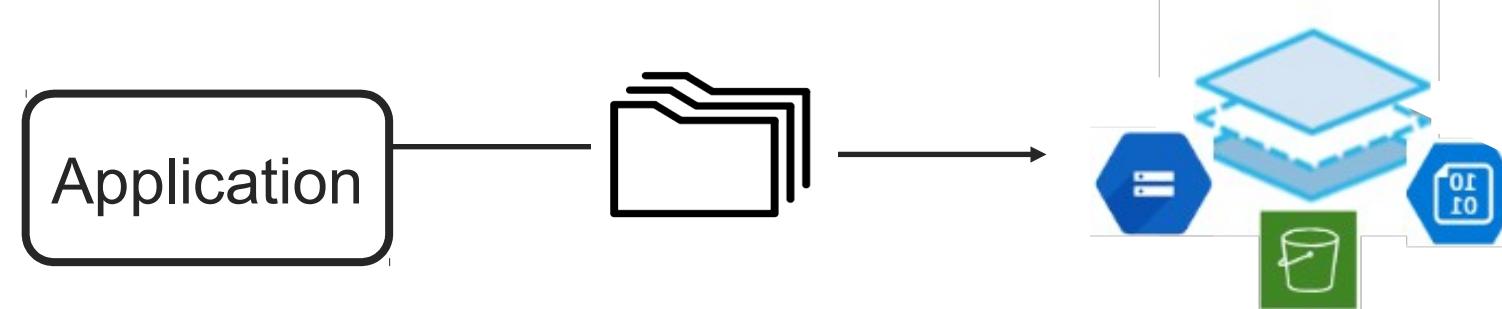


```
COPY INTO my_table FROM @my_stage;
```

- Table, stage, and file format must all exist before the COPY INTO is started
- Compression during the PUT uses local resources. The local host needs sufficient memory, and space in /tmp, or the PUT will fail



EXAMPLE FROM CLOUD STORAGE



Application transfers files to the external stage



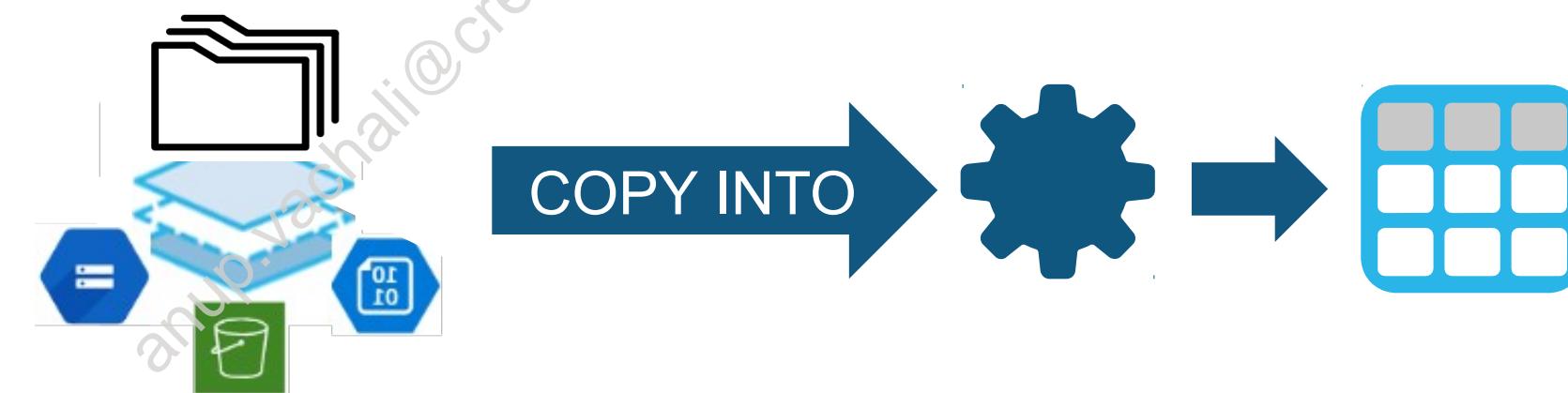
COPY INTO my_table FROM @my_stage;

- Table, stage, and file format must all exist before the `COPY INTO` is started



WHAT FILES ARE LOADED?

- Each stage tracks metadata about which files have been loaded to which tables, and the status of the operation
 - If you TRUNCATE or DROP a table, stage metadata for that table is removed
- File names are hashed with other file metadata
 - If you replace a file with one of the same name, Snowflake will treat it as a new file
- If you do not specify a list of files to load, Snowflake will attempt to load all files that have not already been successfully loaded to the target table



COPY COMMAND PARAMETERS

Optional items in **blue**

```
COPY INTO { [<namespace>].<table name> FROM { <stage> | <external location> }

[FILES = ( '<file name>' [ , '<file name>' ] [ , ... ] )]
[PATTERN = '<regex pattern>']
[FILE_FORMAT = ( { FORMAT_NAME = '[<namespace>.]<format name>' |
                   TYPE = { CSV | JSON | AVRO | ORC | PARQUET | XML }
                   [ <format type options> ] } )]
[VALIDATION_MODE = RETURN_<n>_ROWS | RETURN_ERRORS | RETURN_ALL_ERRORS]
[<copy options>]
```



COPY COMMAND OPTIONS

Default values are shown in **blue**

```
ON_ERROR =
{ CONTINUE | SKIP_FILE | SKIP_FILE_<num> | SKIP_FILE_<num>% | ABORT STATEMENT }
```



```
SIZE_LIMIT = <num bytes>
```



```
PURGE = TRUE | FALSE
```



```
RETURN_FAILED_ONLY = TRUE | FALSE
```



```
ENFORCE_LENGTH = TRUE | FALSE
```



```
TRUNCATECOLUMNS = TRUE | FALSE
```



```
FORCE = TRUE | FALSE
```



```
LOAD_UNCERTAIN_FILES = TRUE | FALSE
```



BULK LOADING RECOMMENDATIONS



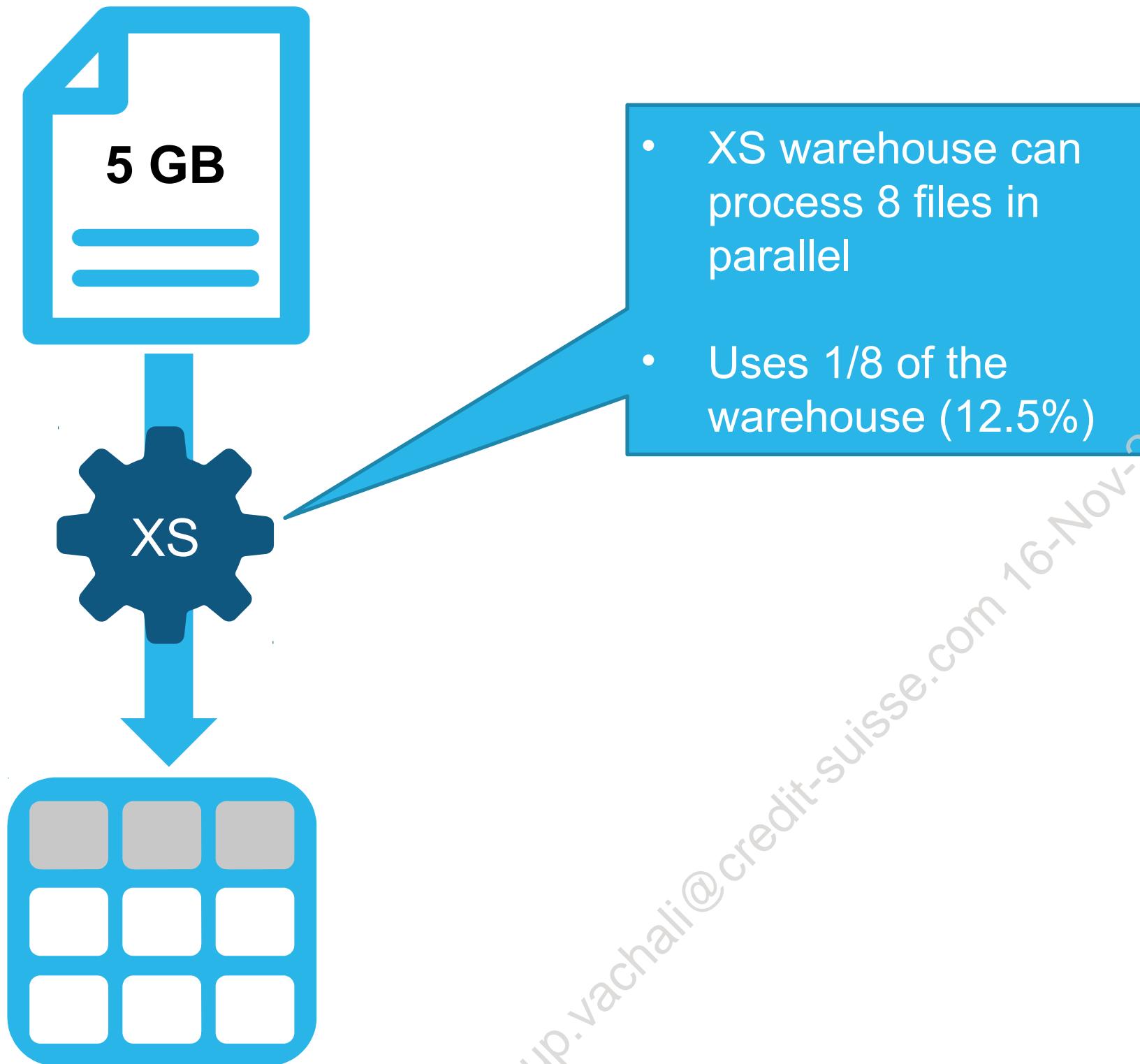
SPLIT LARGE FILES

- Split large single files into multiple files (100 - 250 MB compressed)
- Size warehouse based on the number of files to ingest

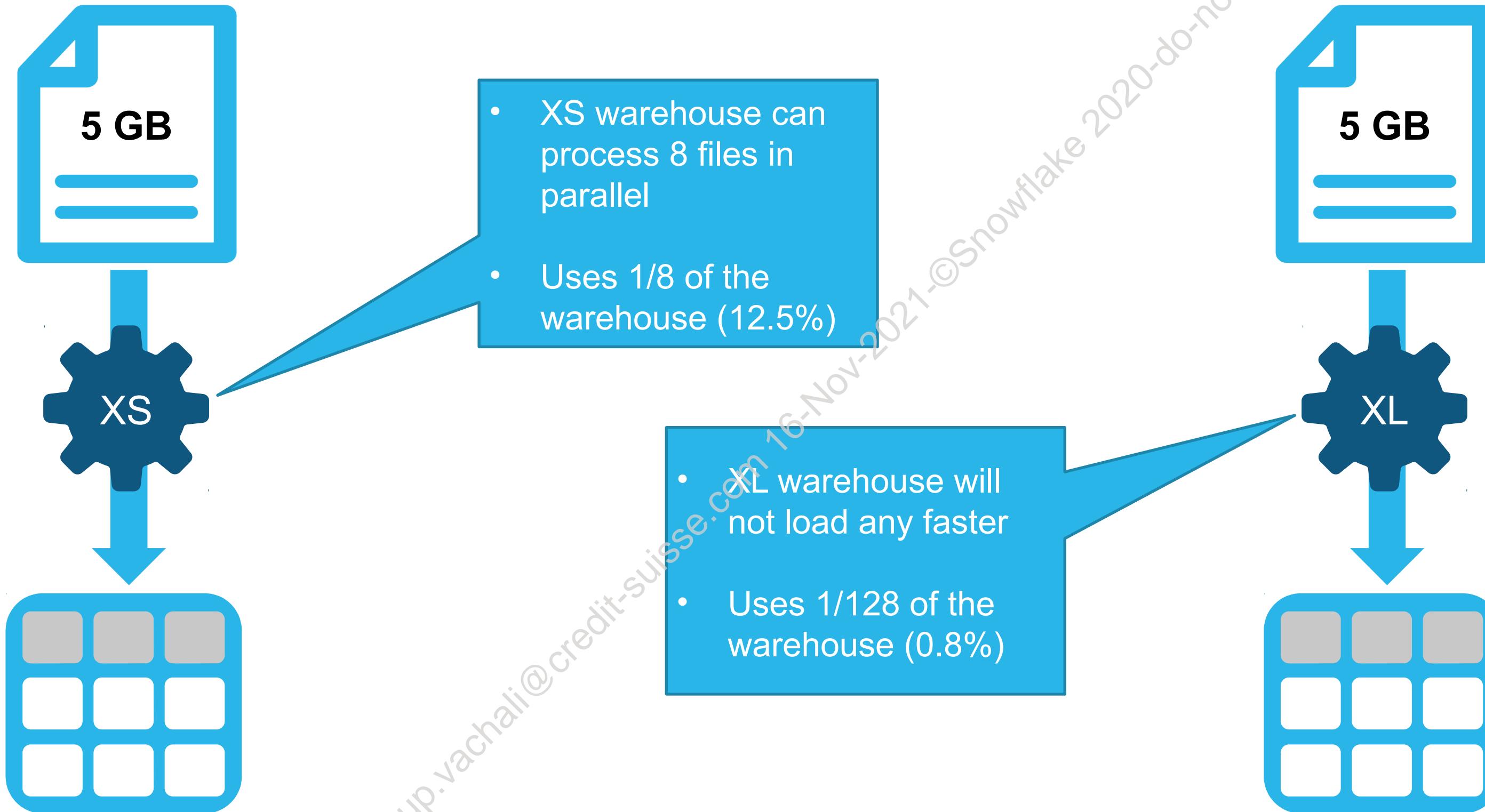
WH Size	# of servers	Concurrent tasks
XS	1	8
S	2	16
M	4	32
L	8	64
XL	16	128



SERIAL COPY: SINGLE FILE

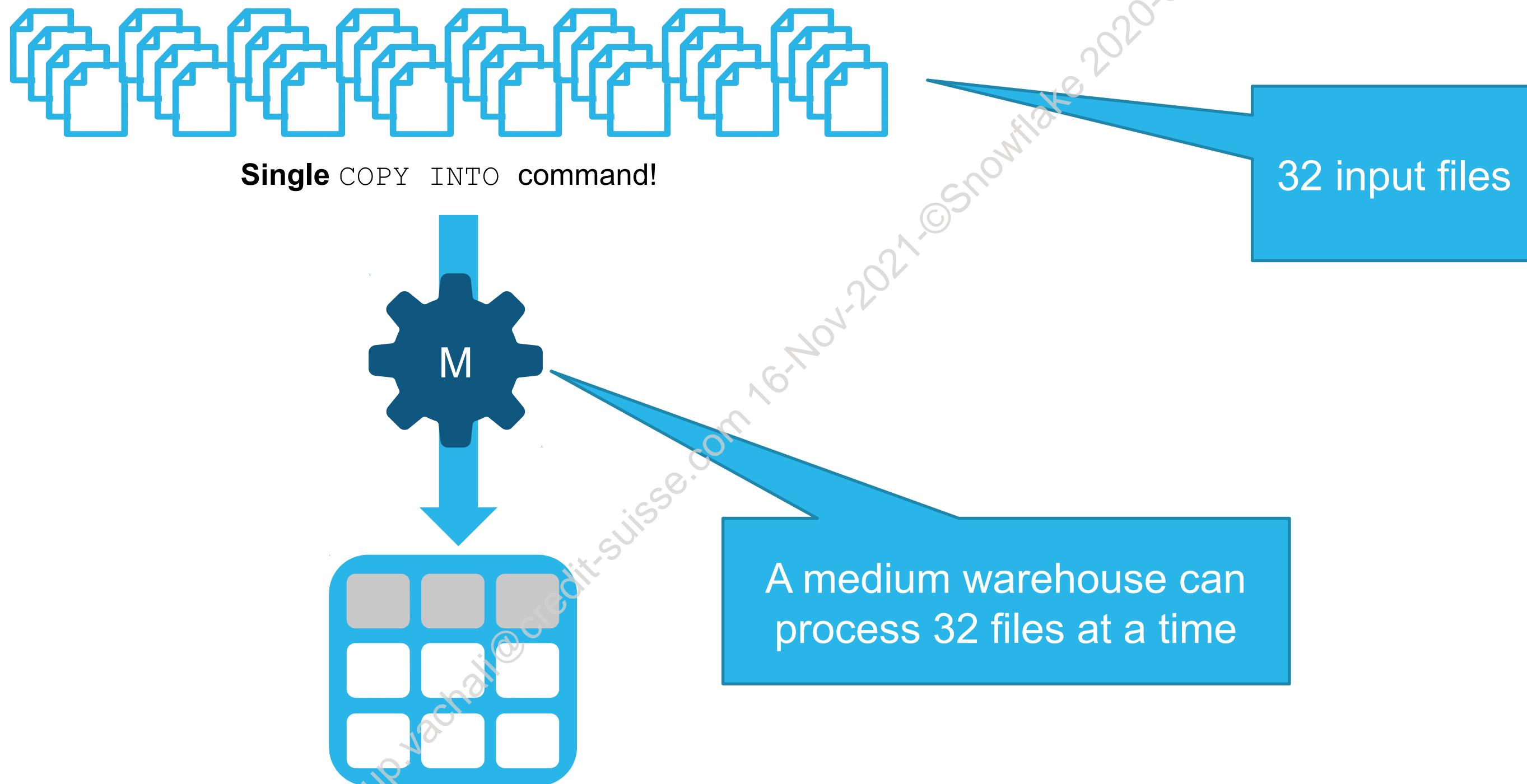


SERIAL COPY: SINGLE FILE



PARALLEL COPY: MULTIPLE FILES

SIZE WAREHOUSE BASED ON NUMBER OF FILES TO INGEST



SPECIFYING A SUBSET OF FILES TO LOAD



Single Location with Many Files
Must scan more files – slower and
uses more cloud compute



Many Locations with Fewer Files
Faster and uses less cloud compute

FILE ORGANIZATION

- Organize data in logical paths using identifiers
 - Example: /mybucket/application/2018/09/05/
- Narrow the path to the most granular level for best performance
- Listing individual files may provide the best performance if you are only loading a few files
 - COPY INTO ... **FILES=(`/2018/09/05/custdata` , `/2018/09/06/custdata` , `...`)**



TRANSFORMING DATA ON LOAD

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



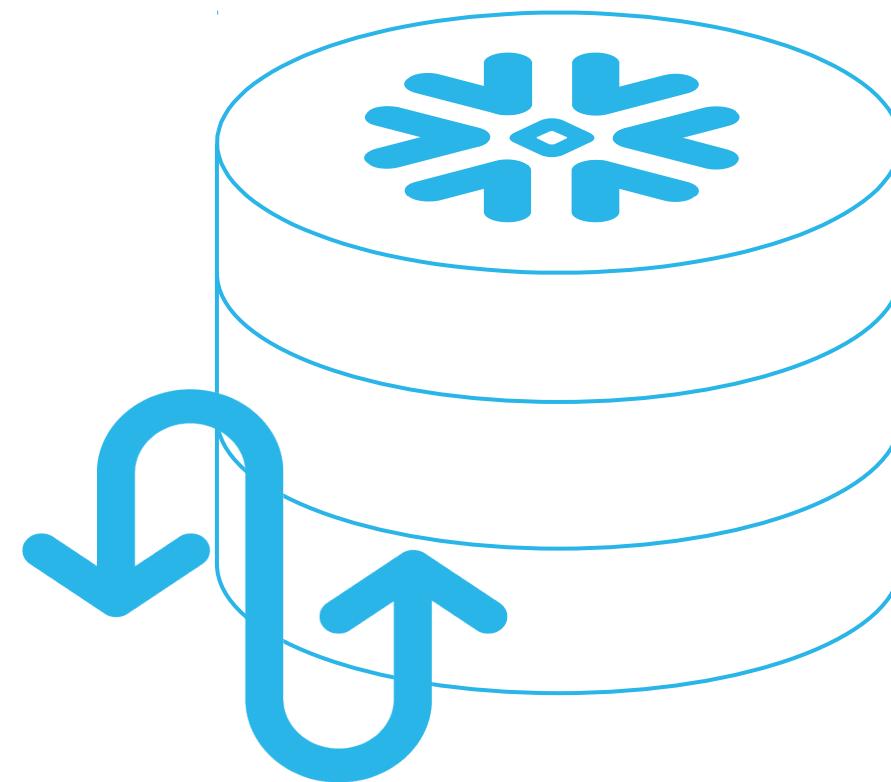
COPY SUPPORTS SIMPLE TRANSFORMATION

Can be done during COPY INTO:

- Column reordering and omission
- CAST using a SELECT statement
- SEQUENCE columns
- CURRENT_TIMESTAMP() and other column functions

Must be done after load:

- Joins
- Filters
- Aggregations



TRANSFORMATION EXAMPLES

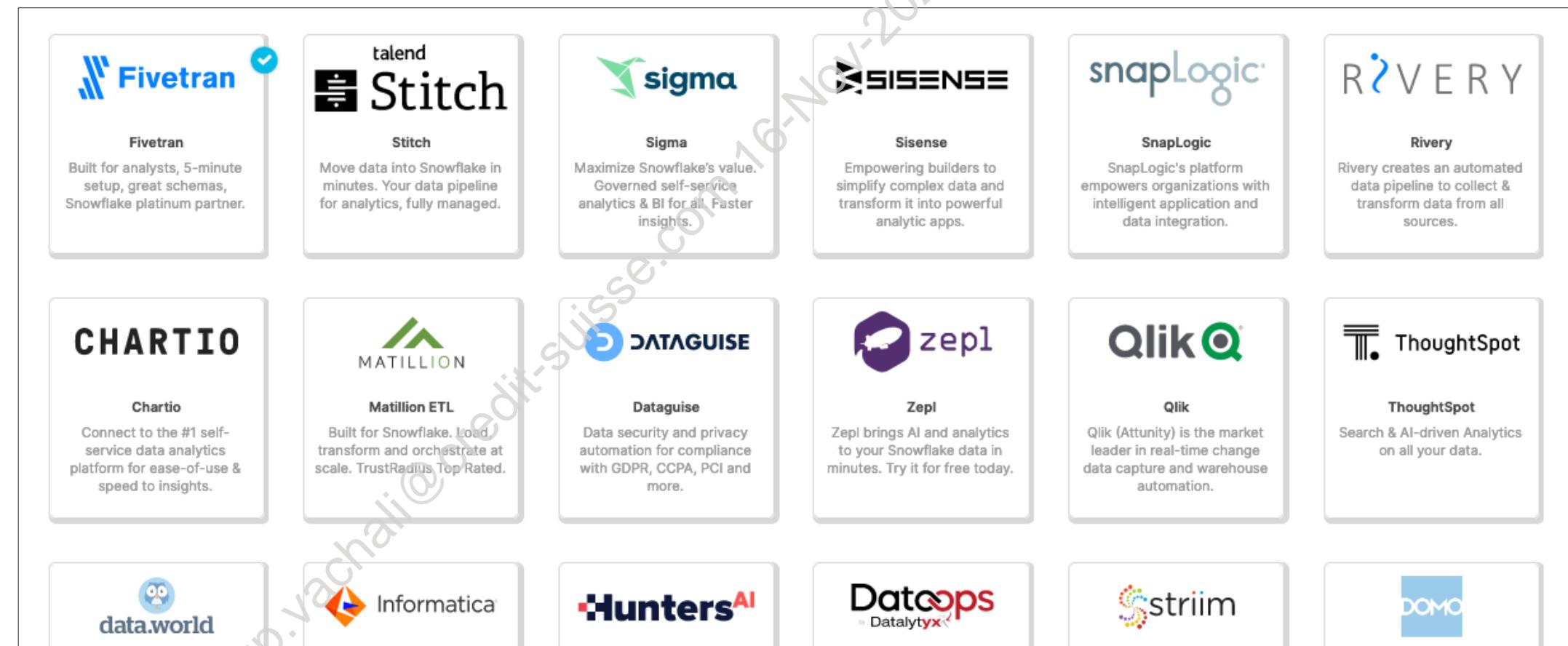
```
COPY INTO home_sales (city, zip, sale_date, price)
FROM (SELECT SUBSTR(t.$2,4), t.$1, t.$5, t.$4 FROM @my_stage t)
FILE_FORMAT = (format_name = mycsvformat);
```

```
COPY INTO CASTTB(col1, col2, col3)
FROM (SELECT
      TO_BINARY(t.$1, 'utf-8'),
      TO_DECIMAL (1.$2, '99.9', 9, 5),
      TO_TIMESTAMP_NTZ(t.$3)
FROM @~/datafile.csv.gz t)
FILE_FORMAT = (type = csv)
```



COMPLEX TRANSFORMATIONS

- Transform the data after loading using Snowflake (ELT)
- Consider using a 3rd party tool
 - See Snowflake documentation for a full list of production-ready solutions



VALIDATION AND ERROR HANDLING



ERROR HANDLING

Use the ON_ERROR option to control how errors in the data load are handled

Supported Values	Notes
CONTINUE	Continue loading the file.
SKIP_FILE	Skip file if any errors encountered in the file.
SKIP_FILE_<num> (e.g. SKIP_FILE_10)	Skip file when the number of errors in the file is equal to or exceeds the specified number.
SKIP_FILE_<num>% (e.g. SKIP_FILE_10%)	Skip file when the percentage of errors in the file exceeds the specified percentage.
ABORT_STATEMENT	Abort the COPY statement if any error is encountered.



VALIDATE BEFORE LOAD

Execute the COPY command in validation mode using VALIDATION_MODE

```
COPY INTO my_table  
FROM @my_stage/mylife.csv.gz  
VALIDATION_MODE=return_all_errors;  
  
SET qid=LAST_QUERY_ID();  
  
SELECT rejected_record FROM TABLE(result_scan($qid));
```



VALIDATE AFTER LOAD

Validates the files loaded in a past execution of the COPY INTO and returns all errors encountered during the load.

```
SELECT * FROM table(VALIDATE(mytable, job_id => '<query_id>'));
```

ERROR	FILE	LINE	CHARACTER
Error parsing JSON: unterminated string	tables/530049/bad.json.gz	120	[NULL]
Error parsing JSON: misplaced colon	tables/530049/bad.json.gz	214	18
Error parsing JSON: unknown keyword "tru"	tables/530049/bad.json.gz	1467	[NULL]
Error parsing JSON: unknown keyword "stat"	tables/530049/bad.json.gz	1469	13
Error parsing JSON: unknown keyword "ok"	tables/530049/bad.json.gz	1469	20
Error parsing JSON: invalid character outside of a string: '\\'	tables/530049/bad.json.gz	1469	21
Error parsing JSON: misplaced }	tables/530049/bad.json.gz	1470	3



MONITORING LOAD STATUS

- Monitor the status of each COPY command run on the History tab page of the Snowflake UI
- Use the INFORMATION_SCHEMA.LOAD_HISTORY view to retrieve the history of data loaded into tables sing the COPY command

```
SELECT file_name, last_load_time, status  
FROM information_schema.load_history  
WHERE SCHEMA_NAME=current_schema() AND TABLE_NAME='customer' AND  
LAST_LOAD_TIME > '2021-02-01 12:00:00 -800';
```

FILE_NAME	LAST_LOAD_TIME	STATUS ↑	FIRST_ERROR_MESSAGE
s3://snowflakeed/load/TCPH_...	2021-02-03 09:14:22.982 -0...	LOADED	NULL
s3://snowflakeed/load/TCPH_...	2021-02-03 09:14:22.982 -0...	LOADED	NULL
s3://snowflakeed/load/TCPH_...	2021-02-03 09:14:22.982 -0...	LOADED	NULL



LAB EXERCISE: 6

Loading, Transforming and Validating Data

45 minutes

- Loading Structured Data
- Loading Data and File Sizes
- Loading Semi-Structured Data
- Loading Fixed Format Data Detect file format problems with **VALIDATION_MODE**
- Load data with various **ON_ERROR** settings
- Reload data with Clean Data



CONTINUOUS DATA LOADING



BULK VS CONTINUOUS

BULK (COPY command)

- Migration from traditional data sources
 - Bulk loading, Backfill processing
- Transaction boundary control
 - BEGIN / START TRANSACTION / COMMIT / ROLLBACK
- Highly customizable for required performance throughput

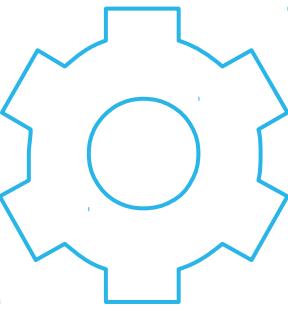
CONTINUOUS (Snowpipe)

- Ingestion from modern data sources
 - Continuously generated data is available for analysis in seconds
- No scheduling (with auto-ingest)
- Serverless model with no user-managed virtual warehouse needed



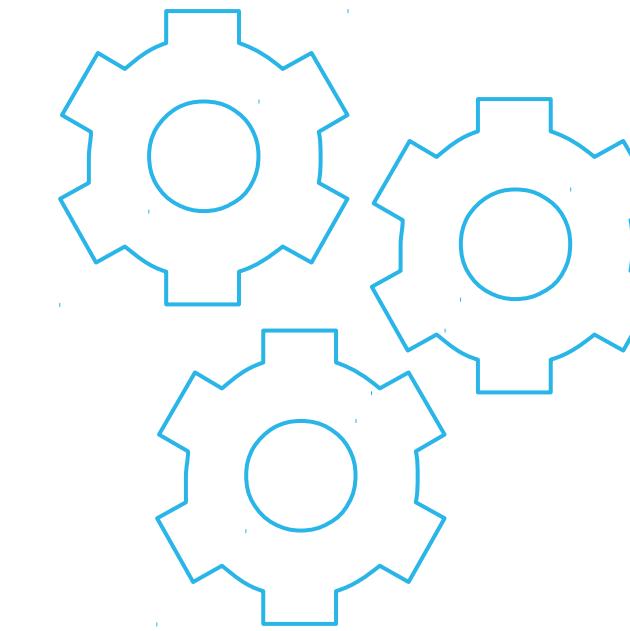
REST VS AUTO_INGEST

REST



- Calls the REST API endpoint
- Passes a list of files in the stage
- Works with internal or external stages

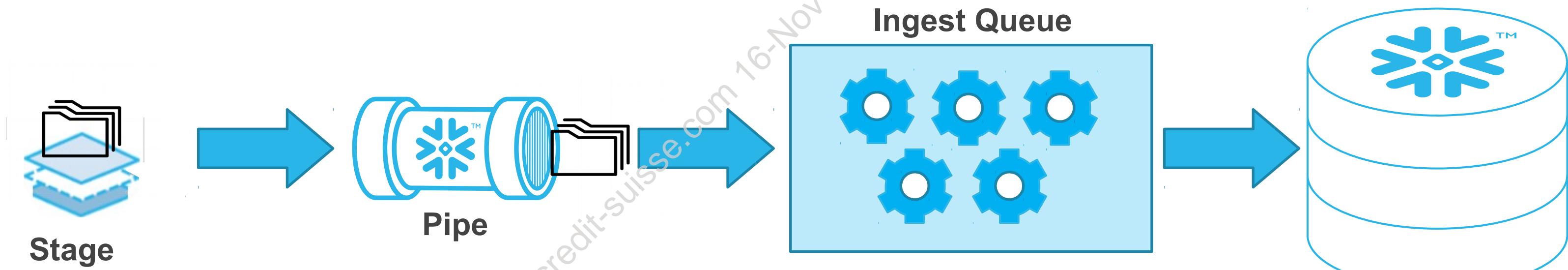
AUTO_INGEST



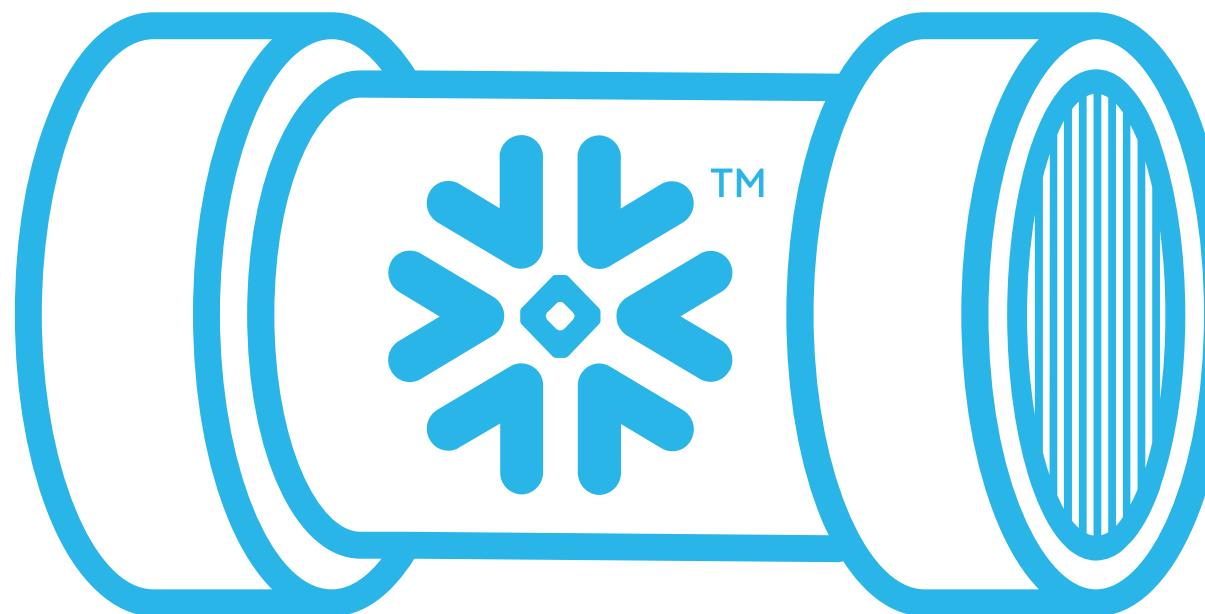
- Receives notifications for new files
- “Wakes up” and processes new files
- Works only with external stages

SNOWPIPE

- For continuous data ingestion (typically < 60 seconds 100 – 250 MB)
- Uses Snowflake serverless compute resources (no virtual warehouse needed)



WHAT IS A PIPE?

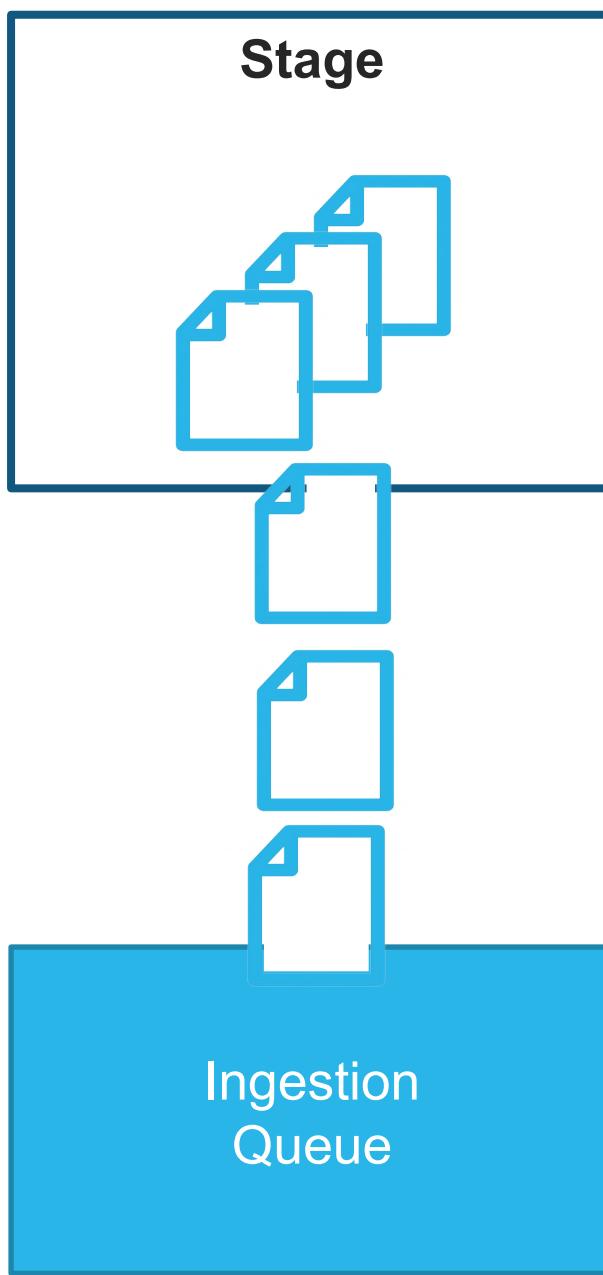


- A pipe is a named definition of a COPY INTO command:

```
CREATE PIPE LOAD_JSON AS  
COPY INTO my_table FROM @my_stage  
FILE_FORMAT = my_json_format;
```

- Three ways to "activate" a pipe:
 - REST API to pass a list of new files
 - Automatic notification when new files appear in the stage (AUTO_INGEST)
 - Manual (ALTER PIPE REFRESH)

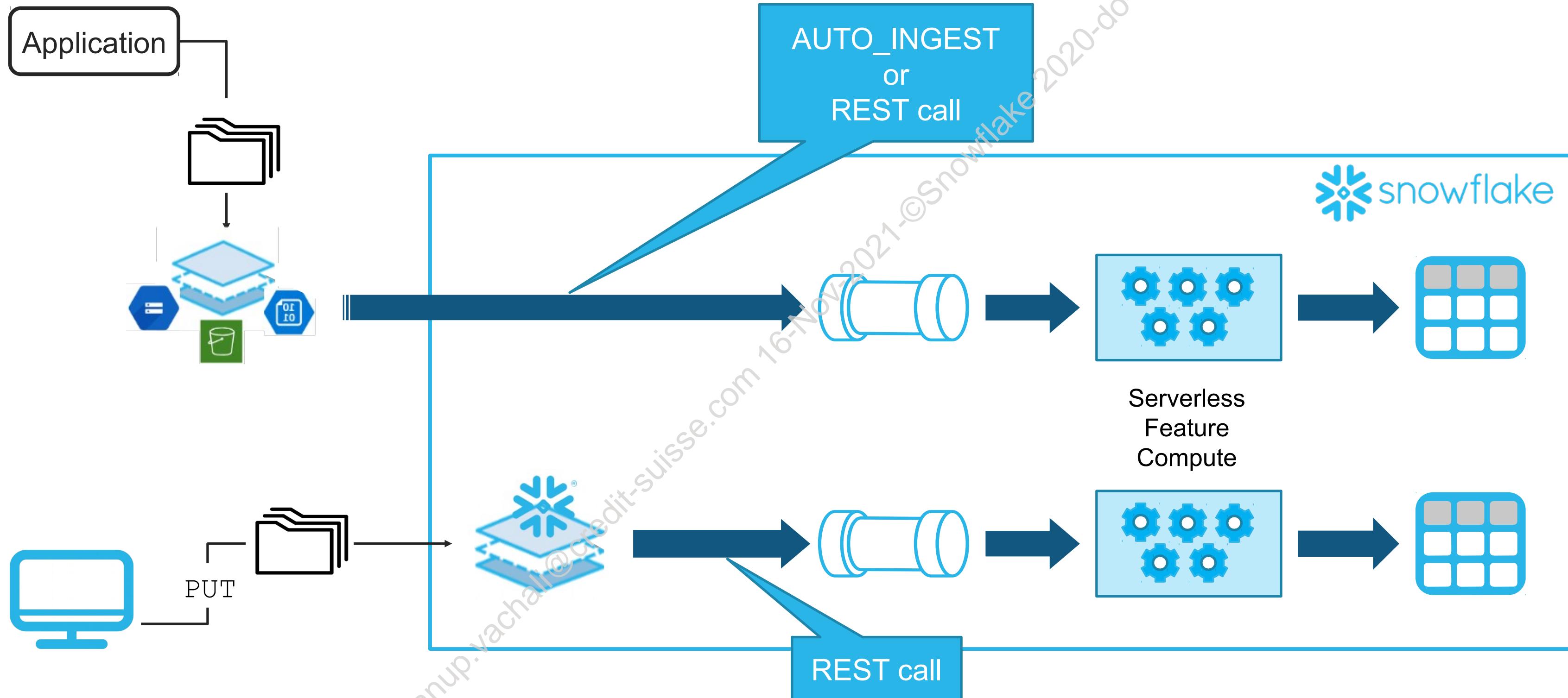
FILE LOAD ORDER



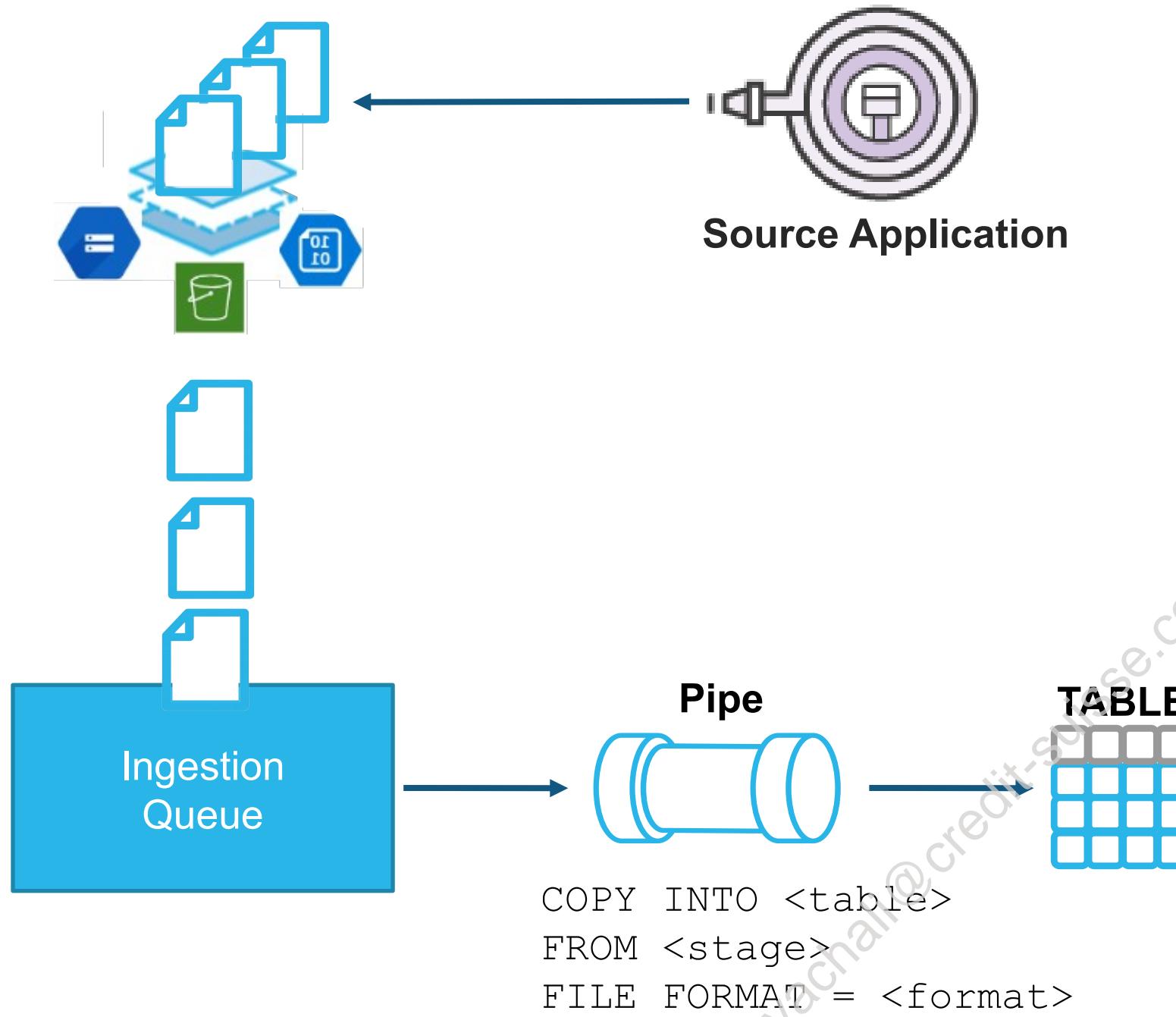
- Staged files are moved into an ingest queue
- Files that appear in the stage later are appended to the queue
- Multiple processes pull from the queue
- Older files are generally loaded first, but files are not guaranteed to be loaded in the same order they are staged



INGEST WITH SNOWPIPE



AUTO_INGEST PROCESS



- Source application loads stage
- Files moved from stage to ingestion queue
- Pipe contains a COPY INTO statement
 - Source stage for data files
 - Target table
- Loads data into tables continuously from an ingestion queue
- Can be paused/resumed, return status



REST VS AUTO_INGEST

REST

```
CREATE PIPE mypipe  
AS COPY INTO mytable  
FROM @mystage;
```

AUTO_INGEST

```
CREATE PIPE mypipe  
AUTO_INGEST=TRUE  
AS COPY INTO mytable  
FROM @mystage;
```



REST VS AUTO_INGEST

REST

1. Create table to hold ingested data

AUTO_INGEST

1. Create table to hold ingested data



REST VS AUTO_INGEST

REST

1. Create table to hold ingested data
2. Create pipe

AUTO_INGEST

1. Create table to hold ingested data
2. Create pipe with **AUTO_INGEST=TRUE**



REST VS AUTO_INGEST

REST

1. Create table to hold ingested data
2. Create pipe

AUTO_INGEST

1. Create table to hold ingested data
2. Create pipe with AUTO_INGEST=TRUE
3. Configure notifications on cloud



REST VS AUTO_INGEST

REST

1. Create table to hold ingested data
2. Create pipe
3. Load data to stage

AUTO_INGEST

1. Create table to hold ingested data
2. Create pipe with AUTO_INGEST=TRUE
3. Configure notifications on cloud
4. Load data to stage



REST VS AUTO_INGEST

REST

1. Create table to hold ingested data
2. Create pipe
3. Load data to stage
4. Make REST API call

AUTO_INGEST

1. Create table to hold ingested data
2. Create pipe with AUTO_INGEST=TRUE
3. Configure notifications on cloud
4. Load data to stage
5. Notification is sent to load data



SECURITY CONSIDERATIONS

- Permissions required for the role executing Snowpipe:

Object	Privilege
Pipe	OWNERSHIP
Named stage	USAGE, READ
File format	USAGE
Target database	USAGE
Target schema	USAGE
Target table	INSERT, SELECT



SNOWPIPE INFORMATION

- Size files between 100-250MB (compressed)
- Stage files no more than once per minute
 - Overhead to manage files increases in relation to the number of files queued
- Snowpipe uses serverless compute
 - Flexible compute optimizes based on load
 - Don't need to worry about suspending a warehouse
 - Charged per-second, per-core
 - Utilization cost of 0.06 credits per 1000 files notified via REST calls or auto-ingest



LAB EXERCISE: 7

Using Snowflake Pipes

15 minutes

- Set up a Basic Snowpipe
- Place Data into Stage and Load into Trips Table

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



QUERY DATA

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



MODULE AGENDA

- Overview
- Estimating and Sampling
- Querying Hierarchical Data
- Using Window Functions
- Querying Semi-Structured Data

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



ESTIMATION AND SAMPLING

anup.vachali@credit-suisse.com 16 Nov 2021 ©Snowflake 2020-do-not-copy



ESTIMATION VS EXACT

MEDIAN (x)

- Slower
- Scalable
- Exact

```
35 select median(ss_sales_price), ss_store_sk  
36 from snowflake_sample_data.tpcds_sf10tcl.store_sales  
37 group by ss_store_sk;
```

Results Data Preview

Query ID SQL 18m59s

APPROX_PERCENTILE(x)

- State-of-the-art percentile estimation functions
- Much faster than MEDIAN
- Uses a constant amount of space regardless of the size of the input

```
42 select approx_percentile(ss_sales_price, 0.5), ss_store_sk  
43 from snowflake_sample_data.tpcds_sf10tcl.store_sales  
44 group by ss_store_sk;  
45  
46 select median(ss_sales_price)
```

Results Data Preview

Query ID SQL 3m2s 751 rows



FREQUENCY ESTIMATION

APPROX_TOP_K (<expr> [, <k> [, <counters>]])

Returns an approximation of frequent values in the input

APPROX_TOP_K_ACCUMULATE (<expr> , <counters>)

Skips the final estimation step and returns the Space-Saving state at the end of an aggregation

APPROX_TOP_K_COMBINE (<state> [, <counters>])

Combines (i.e. merges) input states into a single output state

APPROX_TOP_K_ESTIMATE (<state> [, <k>])

Computes a cardinality estimate of a Space-Saving state produced by APPROX_TOP_K_ACCUMULATE and APPROX_TOP_K_COMBINE



SIMILARITY ESTIMATION

APPROXIMATE_JACCARD_INDEX ([distinct] <expr> [, . . .])

Returns an estimation of the similarity (Jaccard index) of inputs based on their MinHash states.

MINHASH (<k> , [distinct] expr+)

Returns a MinHash state containing an array of size k constructed by applying k number of different hash functions to the input rows and keeping the minimum of each hash function. This MinHash state can then be input to the APPROXIMATE_SIMILARITY function to estimate the similarity with one or more other MinHash states.

MINHASH_COMBINE ([distinct] <state>])

Combines input MinHash states into a single MinHash output state. This Minhash state can then be input to the APPROXIMATE_SIMILARITY function to estimate the similarity with other MinHash states.

This allows use cases in which MINHASH is run over horizontal rowsets of the same table, producing a MinHash state for each rowset. These states can then be combined using MINHASH_COMBINE, producing the same output state as a single run of MINHASH over the entire table.



SAMPLING IN SQL

REGULAR FUNCTIONS

```
SELECT * FROM lineitem WHERE  
MOD(ABS(RANDOM()), 100) < 20;
```

- No knowledge of sampling
- Just executes a set of functions
- random() generator is done for each row and is an expensive operation

SAMPLING CLAUSE

```
SELECT * FROM lineitem  
SAMPLE (20);
```

- Knowledge of sampling by database
- Built-in optimization
- Each row has a x% chance of being in final sample (In this example, it would be a 20% chance)



SAMPLE CLAUSE PARAMETERS

- Sampling probability
 - Probability, or specific number of rows
- Sample Method (optional)
 - ROW or BERNOULLI (default): Accurate but reads every row
 - BLOCK or SYSTEM: Recommended for large data sets. Faster but inaccurate for small tables
- Sampling Seed (optional)
 - Makes the results deterministic



SAMPLING EXAMPLES

```
SELECT * FROM lineitem SAMPLE (10);
```

```
SELECT * FROM lineitem SAMPLE BLOCK (10);
```

```
SELECT * FROM lineitem SAMPLE (5) SEED(4444);
```

```
SELECT * from lineitem SAMPLE (5) JOIN orders SAMPLE (5) on  
(l_orderkey = o_orderkey)
```

```
SELECT AGV(a) FROM (SELECT a FROM lineitem) SAMPLE (5);
```

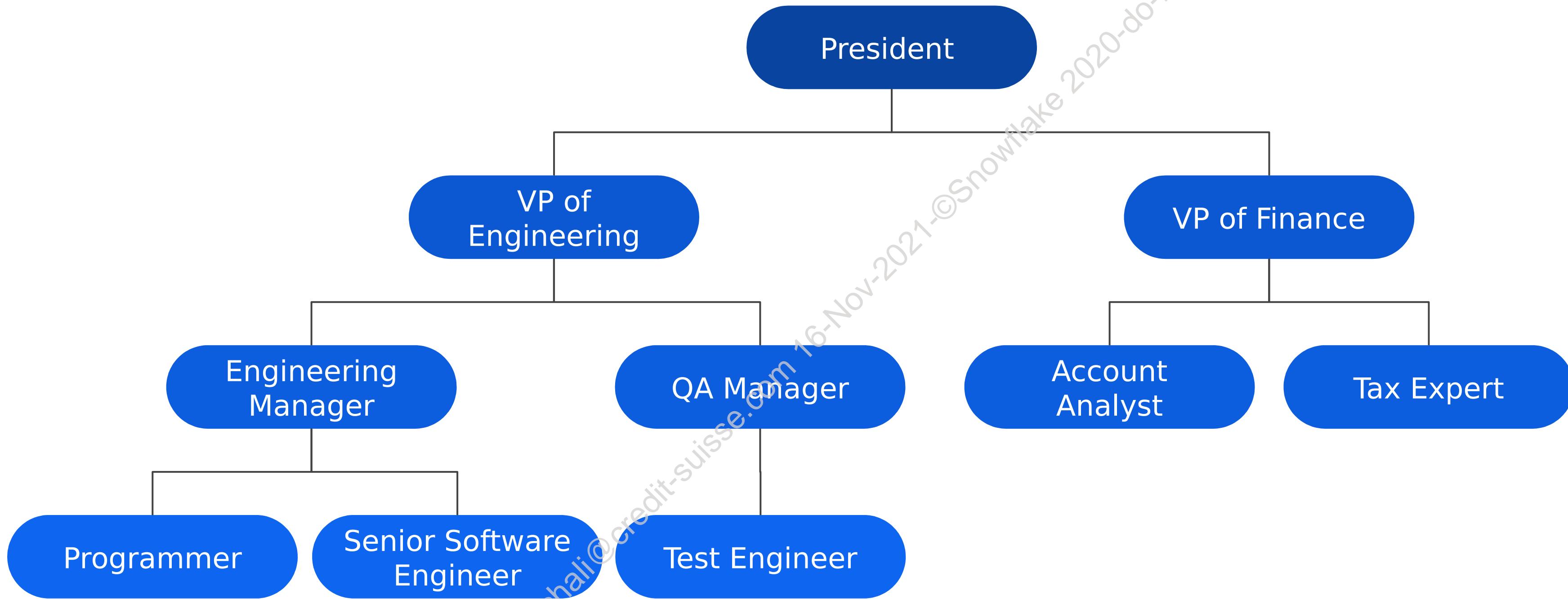


QUERY HIERARCHICAL DATA

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



REPRESENTING HIERARCHICAL DATA



QUERYING HIERARCHICAL DATA

Title	Emp_ID	Mgr_ID
President	0	NULL
VP Engineering	1	0
VP Finance	2	0
Engineering Mgr	3	1
QA Manager	4	1
Account Analyst	5	2
Tax Expert	6	2
Programmer	7	3
Software Engineer	8	3

- How do you find:
 - The total number of reports to the VP of Engineering?
 - The reporting chain for the Tax Expert?
 - The maximum depth in the hierarchy?
- Difficult to do with standard SQL:
 - Requires multiple self-joins (one per level of the hierarchy)
 - If data changes, the query needs to change

Need for hierarchical (recursive) queries!



HIERARCHICAL QUERIES IN SNOWFLAKE

RECURSIVE WITH (SQL Standard)

```
WITH hierarchy AS (
    SELECT *, 1 level, '/' || title chain
    FROM employees
    WHERE emp_id = 0
    UNION ALL
    SELECT e.*,
        h.level + 1,
        chain || '/' || e.title
    FROM
        employees e JOIN hierarchy h
        ON (e.mgr_id = h.emp_id)
)
SELECT * FROM hierarchy;
```

CONNECT BY (Oracle Standard)

```
SELECT*, level,
    sys_connect_by_path(title, '/') chain
FROM employees
CONNECT BY prior emp_id = mgr_id
START WITH emp_id = 0;
```

*Compute the depth in the hierarchy
How is the hierarchy connected?*

*Compute the path in the hierarchy
Where does the hierarchy start?*



RESULT OF THE HIERARCHICAL QUERY EXAMPLE

Title	Emp_ID	Mgr_ID	Level	Chain
President	0	NULL	1	/President
VP Engineering	1	0	2	/President/VP Engineering
VP Finance	2	0	2	/President/VP Finance
Engineering Manager	3	1	3	/President/VP Engineering/Engineering Manager
QA Manager	4	1	3	/President/VP Engineering/QA Manager
Account Analyst	5	2	3	/President/VP Finance/Account Analyst
Tax Expert	6	2	3	/President/VP Finance/Tax Expert
Programmer	7	3	4	/President/VP Engineering/Engineering Manager/Programmer
Software Engineer	8	3	4	/President/VP Engineering/Engineering Manager/Software Engineer
Test Engineer	9	4	4	/President/VP Engineering/QA Manager/Test Engineer



LAB EXERCISE: 8

High-Performance Functions

25 minutes

- Sampling Functions
- Hyperloglog Approximate Count Functions
- Percentile Estimation Functions
- Collations
- Recursive With and Connect By

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



USE WINDOW FUNCTIONS

anup.vachali@credit-suisse.com 16-Nov-2021-©Snowflake 2020-do-not-copy



WHAT IS WINDOWING?

- Taking related rows of data (“windows”) and comparing one data | window in order to understand their relationship
- Method of finding trends in data
- Use cases include:



5-day moving average
for stock prices



Cumulative sales

ID	Val
1	21
2	0
3	13
4	356
5	2
6	8

A diagram illustrating a window in a data table. A blue L-shaped arrow labeled "window" points to the last five rows (ID 3 to 7). A blue arrow labeled "data point" points to the value 356 in the row where ID is 4.

CUMULATIVE VS. SLIDING

- Cumulative – adds rows with each iteration
- Sliding – removes and adds rows with each iteration
- Functions include AVG, COUNT, MIN, MAX, STDDEV, SUM and more

Cumulative

ID	Val
1	21
2	0
3	13
4	356
5	2
6	8

ID	Val
1	21
2	0
3	13
4	356
5	2
6	8

ID	Val
1	21
2	0
3	13
4	356
5	2
6	8

Sliding

ID	Val
1	21
2	0
3	13
4	356
5	2
6	8

ID	Val
1	21
2	0
3	13
4	356
5	2
6	8

ID	Val
1	21
2	0
3	13
4	356
5	2
6	8



THE OVER CLAUSE

- Three-part syntax:
 - PARTITION BY: defines the window
 - ORDER BY: orders the rows within the window before applying the function (optional)
 - ROWS BETWEEN: defines the window frame within the window

Defines the window function

```
SELECT month, week, SUM(registrations)
  (PARTITION BY month
   ORDER BY month, week
   ROWS BETWEEN <start> AND <finish>)

  FROM registrations_table;
```



ROWS BETWEEN

ROWS BETWEEN <start> AND <finish>

- <start> and <finish> can be one of:
 - current row
 - <num> preceding
 - unbounded preceding (to beginning of window)
 - <num> following
 - unbounded following (to end of window)

- Example:

```
...  
PARTITION BY month  
ORDER BY month, week  
ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING
```



month	week	registrations
01	week 1	10
01	week 2	15
01	week 3	6
01	week 4	67
02	week 1	25
02	week 2	100
02	week 3	47
02	week 4	72
03	week 1	15
03	week 2	55
03	week 3	3
03	week 4	12

window

25

25+100=125

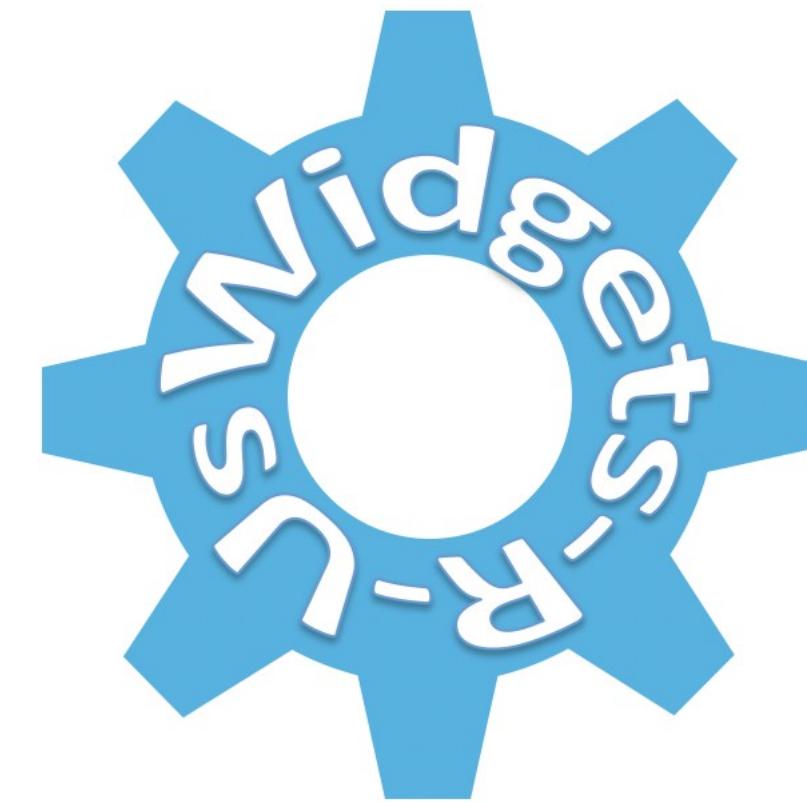
25+100+47=172

25+100+47+72=244



CUMULATIVE WINDOW SCENARIO

- Scenario: Online Widget Company
 - Every day the company monitors cumulative sales on an hourly basis from 8 am – 3 pm
- Need to define:
 - Window Function
 - Window
 - Window Frame



WINDOW FUNCTION

- Define the aggregation to perform
- In this scenario:
 - Every day the company monitors **cumulative sales** on an hourly basis from 8 am – 3 pm
 - Window function will be **SUM**

```
SELECT date, hour, SUM(sales) AS c_sales...
```

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



WINDOW

- Partition the entire data set by some value
- In this scenario:
 - **Every day** the company monitors cumulative sales on an hourly basis from 8 am – 3 pm
- Partition (window) by **date**

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



WINDOW FRAME

- Define the rows within the window to operate on
- In this scenario:
 - Every day the company monitors **cumulative sales** on an hourly basis from 8 am – 3 pm

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



WINDOW FRAME

- Define the rows within the window to operate on
- In this scenario:
 - Every day the company monitors **cumulative sales** on an hourly basis from 8 am – 3 pm
 - **ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING**

Current row indicator

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



WINDOW FRAME

- Define the rows within the window to operate on
- In this scenario:
 - Every day the company monitors **cumulative sales** on an hourly basis from 8 am – 3 pm
 - **ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING**



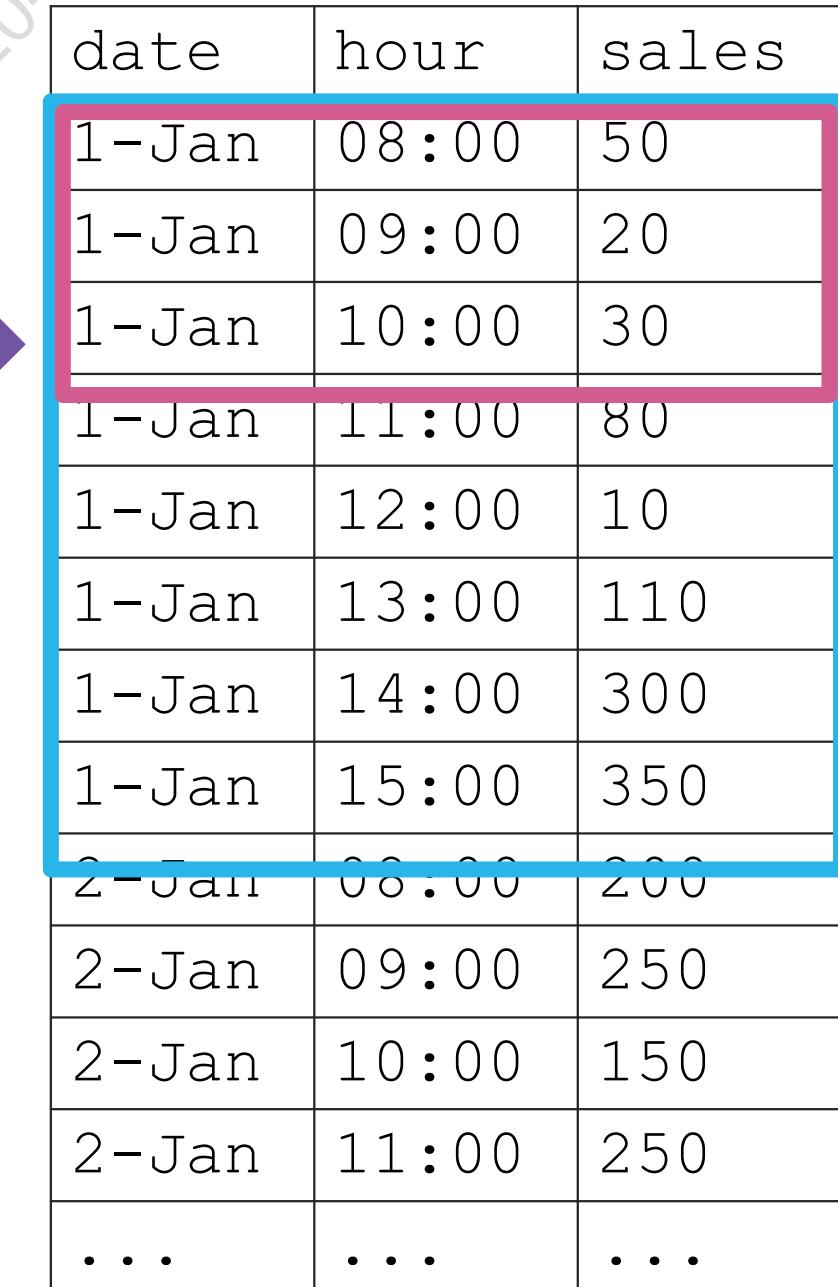
The diagram illustrates a window frame over a dataset of daily sales. A purple arrow points from the text "ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING" to the table. The table has three columns: date, hour, and sales. The rows are grouped by date. A blue border highlights the current row (1-Jan, 08:00, 50). A pink border highlights the unbounded preceding rows (all rows from 1-Jan, 08:00 to 1-Jan, 09:00). A blue border also highlights the current row (2-Jan, 08:00, 200).

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



WINDOW FRAME

- Define the rows within the window to operate on
- In this scenario:
 - Every day the company monitors **cumulative sales** on an hourly basis from 8 am – 3 pm
 - **ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING**



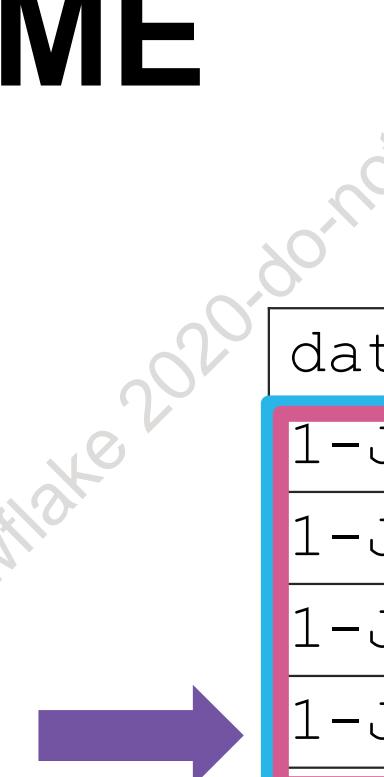
The diagram illustrates a window frame over a dataset of daily sales. A purple arrow points from the text "ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING" to a specific row in the table. This row, for 1-Jan at 10:00, is highlighted with a red border. The entire row is also highlighted with a blue border. The table has columns for date, hour, and sales.

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



WINDOW FRAME

- Define the rows within the window to operate on
- In this scenario:
 - Every day the company monitors **cumulative sales** on an hourly basis from 8 am – 3 pm
 - **ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING**



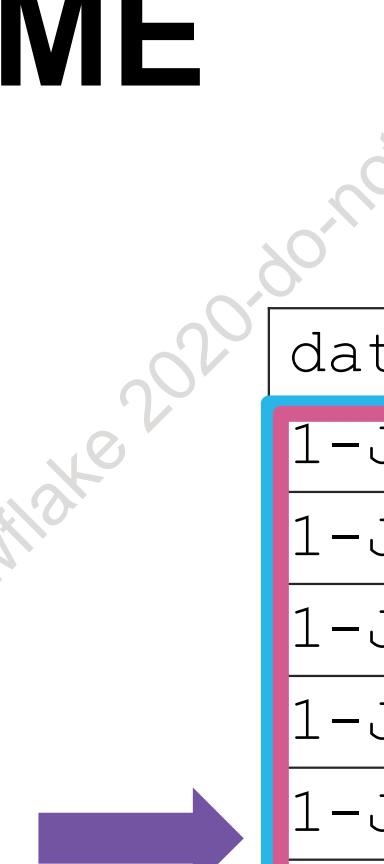
The diagram illustrates a window frame over a dataset of daily sales. A purple arrow points from the text "ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING" to a specific row in the table. This row, for 1-Jan at 11:00, is highlighted with a pink border. Above it, the first four rows (1-Jan 08:00 to 11:00) are highlighted with a blue border, representing the cumulative sum up to the current row. The table has columns: date, hour, and sales.

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



WINDOW FRAME

- Define the rows within the window to operate on
- In this scenario:
 - Every day the company monitors **cumulative sales** on an hourly basis from 8 am – 3 pm
 - **ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING**



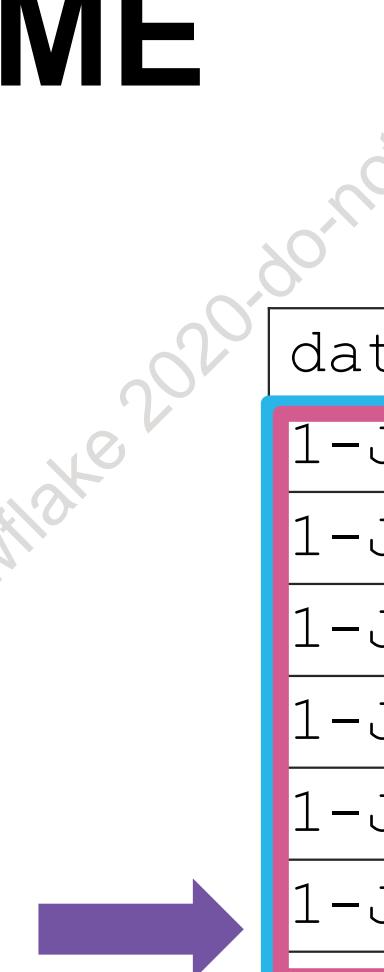
The diagram illustrates a window frame over a dataset of hourly sales. A purple arrow points from the text "ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING" to the table. The table has three columns: date, hour, and sales. The rows are grouped into two main sections by date. The first section for 1-Jan shows sales from 08:00 to 12:00, with a cumulative total of 110 by 13:00. The second section for 2-Jan starts at 08:00 with a value of 200.

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



WINDOW FRAME

- Define the rows within the window to operate on
- In this scenario:
 - Every day the company monitors **cumulative sales** on an hourly basis from 8 am – 3 pm
 - **ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING**



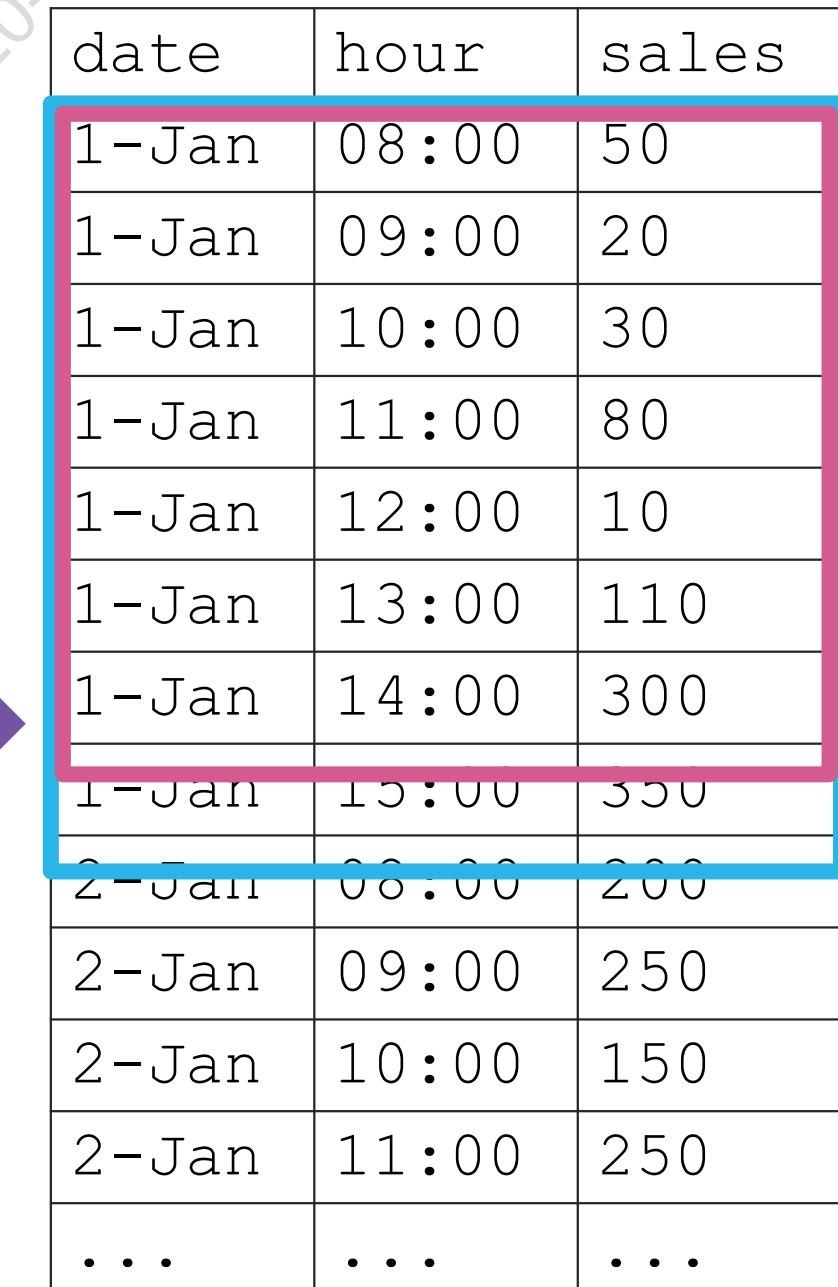
The diagram illustrates a window frame over a dataset of daily sales. A purple arrow points from the text "ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING" to a specific row in the table. This row, for 1-Jan at 13:00, is highlighted with a pink border. The entire row is also highlighted with a blue border. The table has columns for date, hour, and sales.

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



WINDOW FRAME

- Define the rows within the window to operate on
- In this scenario:
 - Every day the company monitors **cumulative sales** on an hourly basis from 8 am – 3 pm
 - **ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING**



The diagram illustrates a window frame over a dataset of daily sales. A purple arrow points from the text "ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING" to a specific row in the table. This row, for 1-Jan at 14:00, is highlighted with a blue border. The entire row is also highlighted with a pink border. The table has columns: date, hour, and sales. The data shows cumulative sales from 8:00 to 14:00 on January 1st, followed by data for January 2nd.

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



WINDOW FRAME

- Define the rows within the window to operate on
- In this scenario:
 - Every day the company monitors **cumulative sales** on an hourly basis from 8 am – 3 pm
 - **ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING**

(continues to end of window)



date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



WINDOW FRAME

- Define the rows within the window to operate on
- In this scenario:
 - Every day the company monitors **cumulative sales** on an hourly basis from 8 am – 3 pm
 - **ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING**



date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



WINDOW FRAME

- Define the rows within the window to operate on
- In this scenario:
 - Every day the company monitors **cumulative sales** on an hourly basis from 8 am – 3 pm
 - **ROWS BETWEEN CURRENT ROW AND UNBOUNDED PRECEDING**

The diagram illustrates a window frame over a dataset of daily sales. A purple arrow points from the text '(continues to end of window)' to a specific row in the table, which is highlighted with a red border. This row represents the current row being processed. The window frame is defined by a blue border around the rows from 2-Jan 08:00 to 2-Jan 11:00. The text '(continues to end of window)' indicates that the window frame continues beyond the visible rows to the end of the data.

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



CUMULATIVE WINDOW FRAME EXAMPLE

```
SELECT date, hour,  
       SUM(sales) as c_sales  
  
OVER  (PARTITION BY date  
        ORDER BY date, hour  
        ROWS BETWEEN CURRENT ROW  
        AND UNBOUNDED PRECEDING)  
  
FROM order_totals_log;
```

Window function

Window

Window frame

order_totals_log

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...

Query output

date	hour	c_sales
1-Jan	08:00	50
1-Jan	09:00	
1-Jan	10:00	
1-Jan	11:00	
1-Jan	12:00	
1-Jan	13:00	
1-Jan	14:00	
1-Jan	15:00	
2-Jan	08:00	
2-Jan	09:00	
2-Jan	10:00	
2-Jan	11:00	
...



CUMULATIVE WINDOW FRAME EXAMPLE

```
SELECT date, hour,  
       SUM(sales) as c_sales  
  
OVER (PARTITION BY date  
      ORDER BY date, hour  
      ROWS BETWEEN CURRENT ROW  
      AND UNBOUNDED PRECEDING)  
  
FROM order_totals_log;
```

order_totals_log

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...

Query output

date	hour	c_sales
1-Jan	08:00	50
1-Jan	09:00	70
1-Jan	10:00	
1-Jan	11:00	
1-Jan	12:00	
1-Jan	13:00	
1-Jan	14:00	
1-Jan	15:00	
2-Jan	08:00	
2-Jan	09:00	
2-Jan	10:00	
2-Jan	11:00	
...



CUMULATIVE WINDOW FRAME EXAMPLE

```
SELECT date, hour,  
       SUM(sales) as c_sales  
  
OVER (PARTITION BY date  
      ORDER BY date, hour  
      ROWS BETWEEN CURRENT ROW  
      AND UNBOUNDED PRECEDING)  
  
FROM order_totals_log;
```

order_totals_log

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...

Query output

date	hour	c_sales
1-Jan	08:00	50
1-Jan	09:00	70
1-Jan	10:00	100
1-Jan	11:00	
1-Jan	12:00	
1-Jan	13:00	
1-Jan	14:00	
1-Jan	15:00	
2-Jan	08:00	
2-Jan	09:00	
2-Jan	10:00	
2-Jan	11:00	
...



CUMULATIVE WINDOW FRAME EXAMPLE

```
SELECT date, hour,  
       SUM(sales) as c_sales  
  
OVER (PARTITION BY date  
      ORDER BY date, hour  
      ROWS BETWEEN CURRENT ROW  
      AND UNBOUNDED PRECEDING)  
  
FROM order_totals_log;
```

order_totals_log

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...

Query output

date	hour	c_sales
1-Jan	08:00	50
1-Jan	09:00	70
1-Jan	10:00	100
1-Jan	11:00	180
1-Jan	12:00	
1-Jan	13:00	
1-Jan	14:00	
1-Jan	15:00	
2-Jan	08:00	
2-Jan	09:00	
2-Jan	10:00	
2-Jan	11:00	
...



CUMULATIVE WINDOW FRAME EXAMPLE

```
SELECT date, hour,  
       SUM(sales) as c_sales  
  
OVER (PARTITION BY date  
      ORDER BY date, hour  
      ROWS BETWEEN CURRENT ROW  
      AND UNBOUNDED PRECEDING)  
  
FROM order_totals_log;
```

order_totals_log

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...

(continues to
end of window)

Query output

Date	hour	c_sales
1-Jan	08:00	50
1-Jan	09:00	70
1-Jan	10:00	100
1-Jan	11:00	180
1-Jan	12:00	190
1-Jan	13:00	300
1-Jan	14:00	600
1-Jan	15:00	950
2-Jan	08:00	
2-Jan	09:00	
2-Jan	10:00	
2-Jan	11:00	
...



CUMULATIVE WINDOW FRAME EXAMPLE

```
SELECT date, hour,  
       SUM(sales) as c_sales  
  
OVER (PARTITION BY date  
      ORDER BY date, hour  
      ROWS BETWEEN CURRENT ROW  
      AND UNBOUNDED PRECEDING)  
  
FROM order_totals_log;
```

order_totals_log

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
...
2-Jan	08:00	250
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...

(continues to
end of window)

Query output

date	hour	c_sales
1-Jan	08:00	50
1-Jan	09:00	70
1-Jan	10:00	100
1-Jan	11:00	180
1-Jan	12:00	190
1-Jan	13:00	300
1-Jan	14:00	600
1-Jan	15:00	950
2-Jan	08:00	200
2-Jan	09:00	
2-Jan	10:00	
2-Jan	11:00	
...



SLIDING WINDOW SCENARIO

- Scenario: Online Widget Company
 - At the end of each day, the company evaluates that day's sales by averaging the sales in three-hour blocks from 8 am – 3 pm
- Need to define:
 - Window Function
 - Window
 - Window Frame



WINDOW FUNCTION

- Define the aggregation to perform on the rows in the window frame
- In this scenario:
 - Every day the company monitors **average sales** on an hourly basis from 8 am – 3 pm
 - Window function will be **AVERAGE**

```
SELECT date, hour, AVERAGE(sales) AS c_sales...
```

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



WINDOW

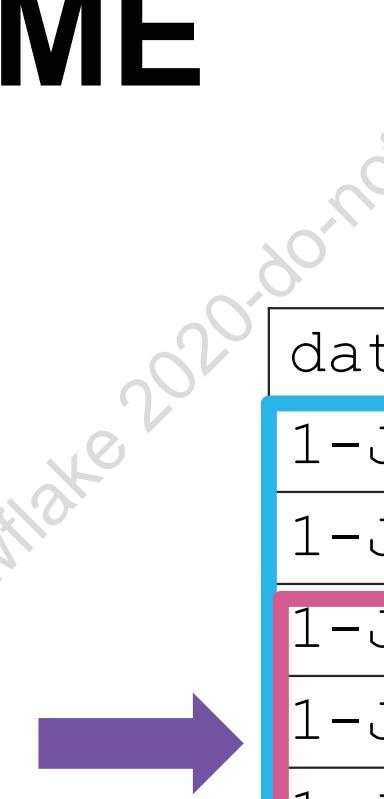
- Partition the entire data set by some value
- In this scenario:
 - **At the end of each day**, the company evaluates that day's sales by averaging the sales in three-hour blocks from 8 am – 3 pm
- Partition (window) by **date**

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



WINDOW FRAME

- Define the rows within the window to operate on
- In this scenario:
 - At the end of each day, the company evaluates that day's sales by averaging the sales in **three-hour blocks from 8 am – 3 pm**
 - This will be a sliding window, with three-hour blocks moving down through the window
 - **ROWS BETWEEN ONE PRECEDING AND ONE FOLLOWING**



date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



SLIDING WINDOW FRAME EXAMPLE

```
SELECT date, hour,  
      AVERAGE(sales) as a_sales  
  
OVER  (PARTITION BY date  
       ORDER BY date, hour  
       ROWS BETWEEN ONE PRECEDING  
         AND ONE FOLLOWING)  
  
FROM order_totals_log;
```

Window function

Window

Window frame

order_totals_log

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...

Query output

date	hour	a_sales
1-Jan	08:00	35.0
1-Jan	09:00	
1-Jan	10:00	
1-Jan	11:00	
1-Jan	12:00	
1-Jan	13:00	
1-Jan	14:00	
1-Jan	15:00	
2-Jan	08:00	
2-Jan	09:00	
2-Jan	10:00	
2-Jan	11:00	
...



SLIDING WINDOW FRAME EXAMPLE

```
SELECT date, hour,  
       AVERAGE(sales) as a_sales  
  
OVER (PARTITION BY date  
      ORDER BY date, hour  
      ROWS BETWEEN ONE PRECEDING  
      AND ONE FOLLOWING)  
  
FROM order_totals_log;
```

order_totals_log

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



Query output

date	hour	a_sales
1-Jan	08:00	35.0
1-Jan	09:00	33.3
1-Jan	10:00	
1-Jan	11:00	
1-Jan	12:00	
1-Jan	13:00	
1-Jan	14:00	
1-Jan	15:00	
2-Jan	08:00	
2-Jan	09:00	
2-Jan	10:00	
2-Jan	11:00	
...



SLIDING WINDOW FRAME EXAMPLE

```
SELECT date, hour,  
       AVERAGE(sales) as a_sales  
  
OVER (PARTITION BY date  
      ORDER BY date, hour  
      ROWS BETWEEN ONE PRECEDING  
      AND ONE FOLLOWING)  
  
FROM order_totals_log;
```

order_totals_log

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...

Query output

date	hour	a_sales
1-Jan	08:00	35.0
1-Jan	09:00	33.3
1-Jan	10:00	43.3
1-Jan	11:00	
1-Jan	12:00	
1-Jan	13:00	
1-Jan	14:00	
1-Jan	15:00	
2-Jan	08:00	
2-Jan	09:00	
2-Jan	10:00	
2-Jan	11:00	
...



SLIDING WINDOW FRAME EXAMPLE

```
SELECT date, hour,  
       AVERAGE(sales) as a_sales  
  
OVER (PARTITION BY date  
      ORDER BY date, hour  
      ROWS BETWEEN ONE PRECEDING  
      AND ONE FOLLOWING)  
  
FROM order_totals_log;
```

order_totals_log

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...

Query output

date	hour	a_sales
1-Jan	08:00	35.0
1-Jan	09:00	33.3
1-Jan	10:00	43.3
1-Jan	11:00	40.0
1-Jan	12:00	
1-Jan	13:00	
1-Jan	14:00	
1-Jan	15:00	
2-Jan	08:00	
2-Jan	09:00	
2-Jan	10:00	
2-Jan	11:00	
...



SLIDING WINDOW FRAME EXAMPLE

```
SELECT date, hour,  
       AVERAGE(sales) as a_sales  
  
OVER (PARTITION BY date  
      ORDER BY date, hour  
      ROWS BETWEEN ONE PRECEDING  
      AND ONE FOLLOWING)  
  
FROM order_totals_log;
```

order_totals_log

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...

Query output

Date	hour	a_sales
1-Jan	08:00	35.0
1-Jan	09:00	33.3
1-Jan	10:00	43.3
1-Jan	11:00	40.0
1-Jan	12:00	66.7
1-Jan	13:00	
1-Jan	14:00	
1-Jan	15:00	
2-Jan	08:00	
2-Jan	09:00	
2-Jan	10:00	
2-Jan	11:00	
...



SLIDING WINDOW FRAME EXAMPLE

```
SELECT date, hour,  
       AVERAGE(sales) as a_sales  
  
OVER (PARTITION BY date  
      ORDER BY date, hour  
      ROWS BETWEEN ONE PRECEDING  
      AND ONE FOLLOWING)  
  
FROM order_totals_log;
```

order_totals_log

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



Query output

Date	hour	a_sales
1-Jan	08:00	35.0
1-Jan	09:00	33.3
1-Jan	10:00	43.3
1-Jan	11:00	40.0
1-Jan	12:00	66.7
1-Jan	13:00	140.0
1-Jan	14:00	
1-Jan	15:00	
2-Jan	08:00	
2-Jan	09:00	
2-Jan	10:00	
2-Jan	11:00	
...



SLIDING WINDOW FRAME EXAMPLE

```
SELECT date, hour,  
       AVERAGE(sales) as a_sales  
  
OVER (PARTITION BY date  
      ORDER BY date, hour  
      ROWS BETWEEN ONE PRECEDING  
      AND ONE FOLLOWING)  
  
FROM order_totals_log;
```

order_totals_log

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...

(continues to
end of window)

Query output

Date	hour	a_sales
1-Jan	08:00	35.0
1-Jan	09:00	33.3
1-Jan	10:00	43.3
1-Jan	11:00	40.0
1-Jan	12:00	66.7
1-Jan	13:00	140.0
1-Jan	14:00	253.3
1-Jan	15:00	
2-Jan	08:00	
2-Jan	09:00	
2-Jan	10:00	
2-Jan	11:00	
...



SLIDING WINDOW FRAME EXAMPLE

```
SELECT date, hour,  
       AVERAGE(sales) as a_sales  
  
OVER  (PARTITION BY date  
        ORDER BY date, hour  
        ROWS BETWEEN ONE PRECEDING  
        AND ONE FOLLOWING)  
  
FROM order_totals_log;
```

order_totals_log

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
2-Jan	08:00	200
2-Jan	09:00	250
2-Jan	10:00	150
2-Jan	11:00	250
...



Query output

Date	hour	a_sales
1-Jan	08:00	35.0
1-Jan	09:00	33.3
1-Jan	10:00	43.3
1-Jan	11:00	40.0
1-Jan	12:00	66.7
1-Jan	13:00	140.0
1-Jan	14:00	253.3
1-Jan	15:00	325.0
2-Jan	08:00	
2-Jan	09:00	
2-Jan	10:00	
2-Jan	11:00	
...



SLIDING WINDOW FRAME EXAMPLE

```
SELECT date, hour,  
       AVERAGE(sales) as a_sales  
  
OVER  (PARTITION BY date  
        ORDER BY date, hour  
        ROWS BETWEEN ONE PRECEDING  
        AND ONE FOLLOWING)  
  
FROM order_totals_log;
```

order_totals_log

date	hour	sales
1-Jan	08:00	50
1-Jan	09:00	20
1-Jan	10:00	30
1-Jan	11:00	80
1-Jan	12:00	10
1-Jan	13:00	110
1-Jan	14:00	300
1-Jan	15:00	350
...
2-Jan	08:00	250
2-Jan	09:00	150
2-Jan	10:00	250
2-Jan	11:00	250
...

(starts new
window)

Query output

date	hour	a_sales
1-Jan	08:00	35.0
1-Jan	09:00	33.3
1-Jan	10:00	43.3
1-Jan	11:00	40.0
1-Jan	12:00	86.6
1-Jan	13:00	140.0
1-Jan	14:00	253.3
1-Jan	15:00	325.0
2-Jan	08:00	225.0
2-Jan	09:00	
2-Jan	10:00	
2-Jan	11:00	
...

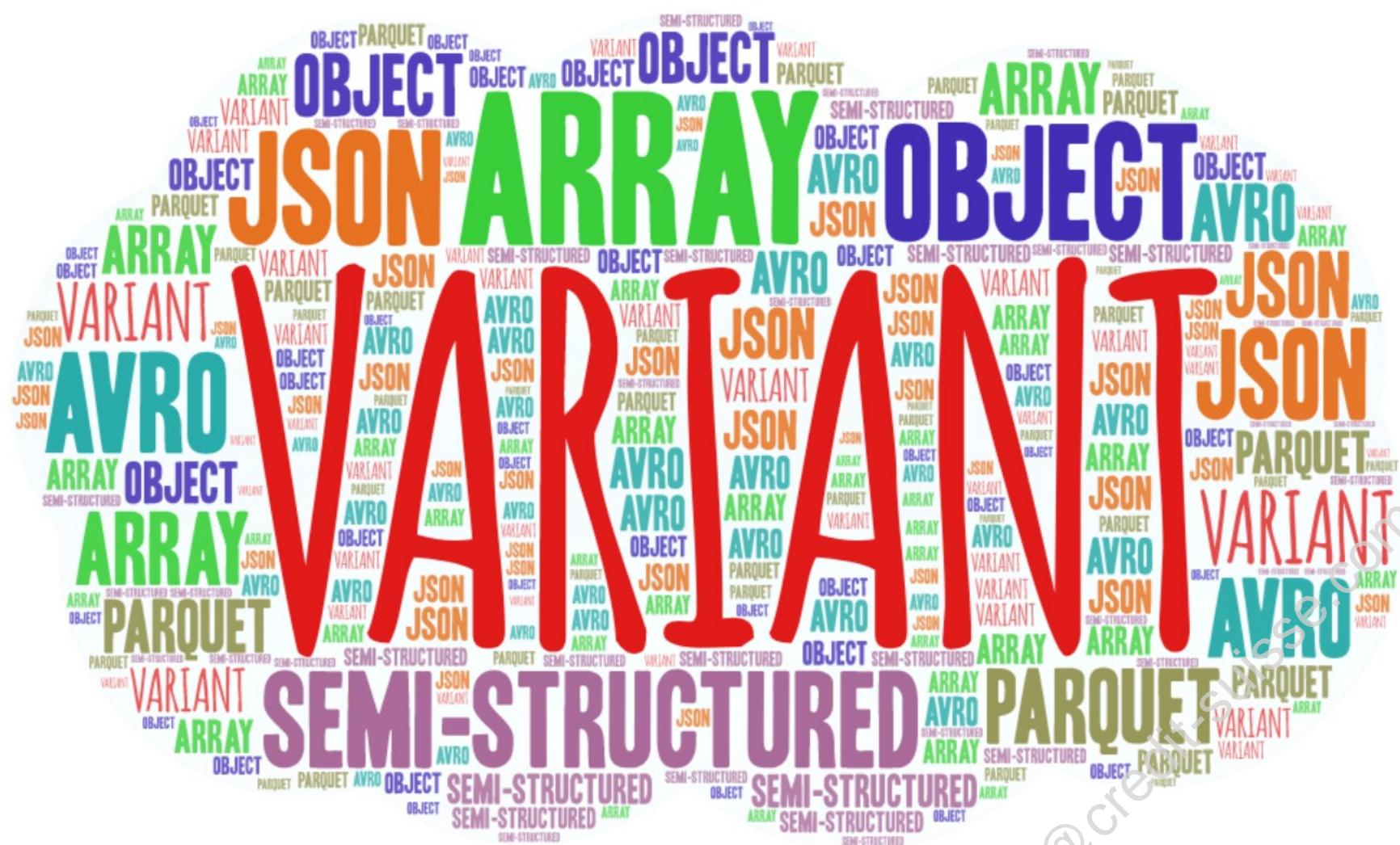


QUERY SEMI-STRUCTURED DATA

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



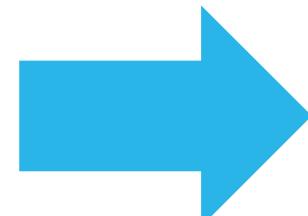
SEMI-STRUCTURED DATA TYPES



- VARIANT – holds values of standard SQL type, arrays, and objects
 - Non-native values (e.g., dates and timestamps) are stored as strings in a variant column
 - Operations could be slower than when stored in a relational column as the corresponding data type
- OBJECT – a collection of key-value pairs
 - The value is a VARIANT
- ARRAY – Arrays of varying sizes
 - The value is a VARIANT

QUERY RICH HIERARCHICAL STRUCTURES IN SQL

{"data": {"observations": [{"air": { "dew-point": 1.9, "dew-point-quality-code": "1", "elevation": 42.4, "lat": 46.7, "lon": -121.5, "temp": 10.2, "temp-quality-code": "1" }, {"atmosphere": { "dew-point": 3.0, "dew-point-quality-code": "1", "elevation": 1271.6, "lat": 46.7, "lon": -121.5, "temp": 14.1, "temp-quality-code": "1" }, {"cloud": { "dew-point": 24.6, "dew-point-quality-code": "1", "elevation": 1602.6, "lat": 46.7, "lon": -121.5, "temp": 24.6, "temp-quality-code": "1" }, {"precipitation": { "dew-point": 22.4, "dew-point-quality-code": "1", "elevation": 9.1, "lat": 46.7, "lon": -121.5, "temp": 28.5, "temp-quality-code": "1" }, {"wind": { "dew-point": -2.6, "dew-point-quality-code": "1", "elevation": 763.0, "lat": 46.7, "lon": -121.5, "temp": 0.8, "temp-quality-code": "1" } }, {"atmosphere": { "dew-point": 5.8, "dew-point-quality-code": "1", "elevation": 232.0, "lat": 46.7, "lon": -121.5, "temp": 9.6, "temp-quality-code": "1" }, {"cloud": { "dew-point": -10.3, "dew-point-quality-code": "1", "elevation": 9.1, "lat": 46.7, "lon": -121.5, "temp": -6.5, "temp-quality-code": "1" }, {"precipitation": { "dew-point": -8.7, "dew-point-quality-code": "1", "elevation": 1790.0, "lat": 46.7, "lon": -121.5, "temp": 24.1, "temp-quality-code": "1" }, {"wind": { "dew-point": -16.4, "dew-point-quality-code": "1", "elevation": 191.1, "lat": 46.7, "lon": -121.5, "temp": -7.0, "temp-quality-code": "1" } }, {"atmosphere": { "dew-point": 20.7, "dew-point-quality-code": "1", "elevation": 27.4, "lat": 46.7, "lon": -121.5, "temp": 26.3, "temp-quality-code": "1" }] }}
--



ELEVATION	TEMP	DEW_POINT	WIND_SPEED
42.4	23	16	21
1271.6	-7.8	-10.6	62
1602.6	21	17	0
9.1	12.2	10.6	15
763	-1	-3	21
232	23	16	26
9.1	28	22	31
1790	21	9	0
191.1	26	10	21
27.4	2	-3	57



ACCESSING VALUES IN NESTED DATA

Access a value in a VARIANT column for a particular key

Colon - column:key

```
SELECT v:station.elev  
AS elevation  
FROM  
weather.isd_2019_total  
...
```

ELEVATION
=====
763

Brackets - column['key']

```
SELECT v['station.elev']  
AS elevation  
FROM  
weather.isd_2019_total  
...
```

ELEVATION
=====
763



CASTING DATA FROM VARIANTS

- VARIANTs are often just strings, arrays or numbers
- Without CASTing, they remain VARIANT object types
- Cast them to SQL data types using the :: operator

```
SELECT
    v:station.elev::NUMBER(4,2)
    AS elevation
FROM
    weather.isd_2019_total dt,
    LATERAL FLATTEN(input=> dt.v:data)
LIMIT 5;
```

↑ Row	ELEVATION	TEMP
1	7	8.9
2	5	6.9
3	145.4	1
4	39.5	32.6
5	819.5	25.7



QUERY RICH HIERARCHICAL STRUCTURES IN SQL

```
{"data": {"observations": [ { "air": { "dew-point": 1.9, "dew-point-quality-code": "1"...,  
[ { "air": { "dew-point": 1.9, "dew-point-quality-code": "1", "temp": 10.2, "temp-quality-code": "1" }, "atm...  
  
SELECT  
    v:station.elev AS elevation,  
    v:data.observations[0].air.temp::FLOAT  
        AS temp,  
    v:data.observations[0].air."dew-point)::FLOAT  
        AS dew_point,  
    v:data.observations[0].wind."speed-rate"::INT  
        AS wind_speed  
FROM  
    weather.isd_2019_total  
LIMIT 10;
```

ELEVATION	TEMP	DEW_POINT	WIND_SPEED
42.4	23	16	21
1271.6	-7.8	-10.6	62
1602.6	21	17	0
9.1	12.2	10.6	15
763	-1	-3	21
232	23	16	26
9.1	28	22	31
1790	21	9	0
191.1	26	10	21
27.4	2	-3	57



FLATTENING NESTED DATA

- VARIANTs may contain nested elements: arrays or objects that contain the data
- FLATTEN() explodes compound values into multiple rows
 - Takes a VARIANT, OBJECT, or ARRAY column
 - Often used with a LATERAL join (to refer to columns from other tables, views, or table functions)

```
LATERAL FLATTEN(input=> expression [ options ] )
```

- input => designates the expression or column that will be unnested into rows



COLUMNS RETURNED BY FLATTEN

SEQ	Unique sequence number associated with input record; not guaranteed to be gap-free or ordered any particular way
KEY	For maps or objects, this column contains the key to the exploded value
PATH	Path to the element within a data structure that needs to be flattened
INDEX	Index of the element, if it is an array; otherwise NULL
VALUE	Value of the element of the flattened array or object
THIS	The element being flattened (useful in recursive flattening)



FLATTEN DATA

- Use FLATTEN() to output rows for each column

```
SELECT key, path, index, value  
FROM TABLE(FLATTEN  
  (input=> PARSE_JSON(' [1,2,3,4]')));
```

KEY	PATH	INDEX	VALUE
NULL	[0]	0	1
NULL	[1]	1	2
NULL	[2]	2	3
NULL	[3]	3	4

Input is an array (a collection of VALUES), so the KEY column is NULL



FLATTEN VARIANT COLUMNS

Table t1 (VARIANT column, v):

```
{ "clouds" : 41,  
  "humidity": 21,  
  "temp": {  
    "day" : 79.30,  
    "eve" : 81.20,  
    "max" : 81.20,  
    "min" : 68.23,  
    "morn" : 75.28,  
    "night" : 68.23}  
}
```

```
SELECT key, value  
FROM t1 t,  
TABLE (FLATTEN (input=> t1.v));  
  
KEY          VALUE  
clouds       41  
humidity     21  
temp         { "day": 79.30, "eve": 81.20, "m
```

Value of “temp” is
a nested object



FLATTEN RECURSIVELY

Table t1 (VARIANT column, v):

```
{ "clouds" : 41,  
  "humidity": 21,  
  "temp": {  
    "day" : 79.30,  
    "eve" : 81.20,  
    "max" : 81.20,  
    "min" : 68.23,  
    "morn" : 75.28,  
    "night" : 68.23}  
}
```

“temp” object is
flattened

```
SELECT key, value  
FROM t1,  
TABLE (FLATTEN (input=> t1.v,  
                RECURSIVE => TRUE));
```

KEY	VALUE
clouds	41
humidity	21
temp	{ "day": 79.30, "eve": 81.20, "m
day	79.3
eve	81.2
max	81.2
min	68.23
morn	75.28
night	68.23



FLATTEN WITH LATERAL JOIN

- Use a LATERAL JOIN to access columns in other parts of the query

```
SELECT v, f.key AS time_of_day, f.value AS temp  
FROM t1,  
LATERAL FLATTEN(input=> t1.v:temp) f;
```

The “V” column is from the source table

V	TIME_OF_DAY	TEMP
{"clouds":41, "humidity":21, "temp":{"day":79.30}}	day	79.3
{"clouds":41, "humidity":21, "temp":{"day":79.30}}	eve	81.2
{"clouds":41, "humidity":21, "temp":{"day":79.30}}	max	81.2
{"clouds":41, "humidity":21, "temp":{"day":79.30}}	min	68.23
{"clouds":41, "humidity":21, "temp":{"day":79.30}}	morn	75.28
{"clouds":41, "humidity":21, "temp":{"day":79.30}}	night	68.23



LAB EXERCISE: 9

Query Semi-Structured and Structured Data Together

20 minutes

- Query the Weather Data
- Query Semi-Structured Data
- Create a View for Production
- Create a Joined View

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



WORK WITH DATA LAKES

anup.vachali@credit-suisse.com 16 Nov 2021 ©Snowflake 2020-do-not-copy



MODULE AGENDA

- Overview
- External Tables

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



OVERVIEW

anup.vachali@credit-suisse.com 16-Nov-2021-©Snowflake 2020-do-not-copy



ACCESS METHODS FOR EXTERNAL DATA

Slowest

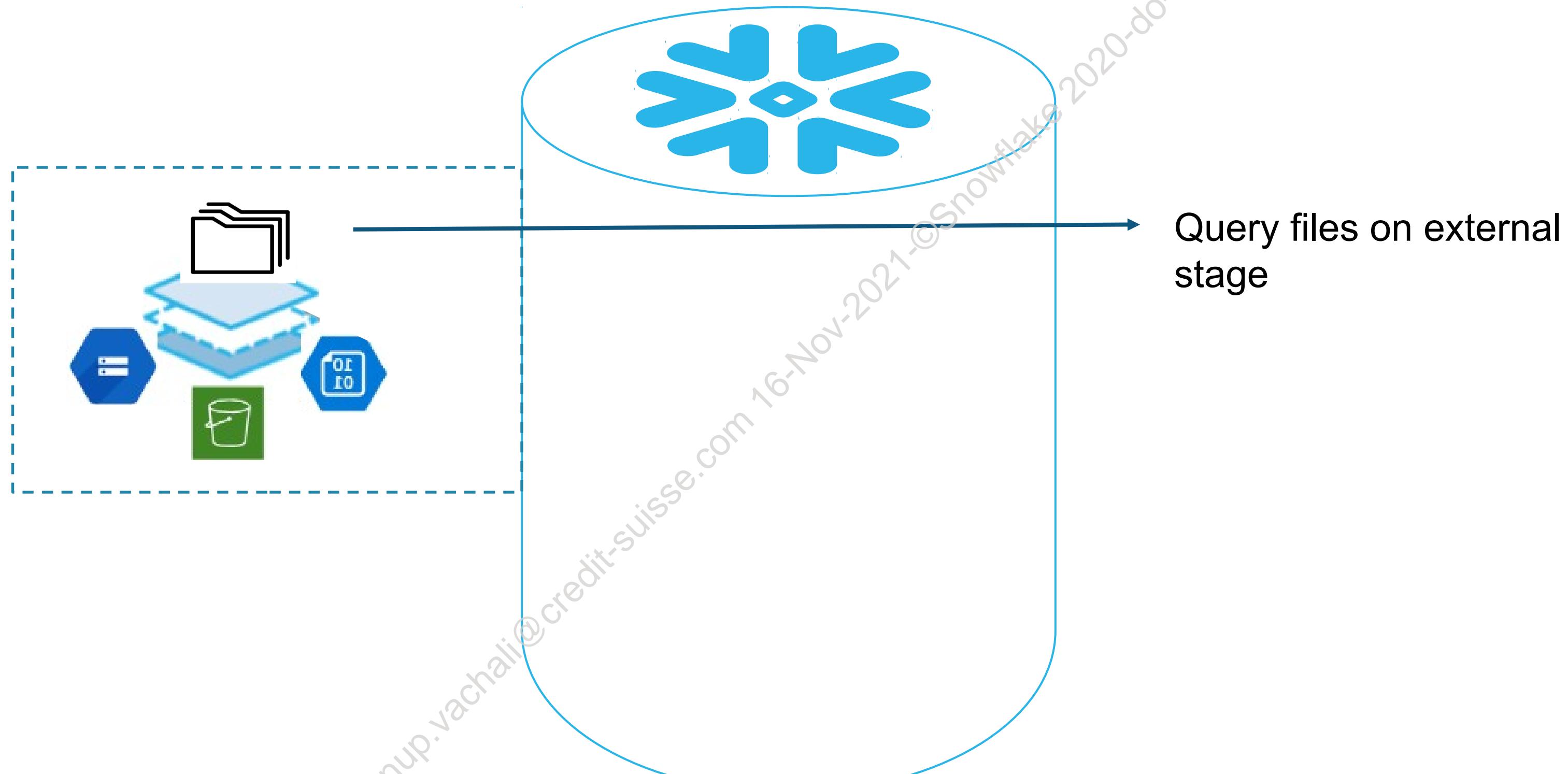


Type	Data Location	Micro-partition Pruning	Schema
External Data	External to Snowflake	None	Schema at query time
External Table	External to Snowflake	Coarse, based on file path	Can define external table and schema using views on table
External Table + MV	External to Snowflake; view result set is materialized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

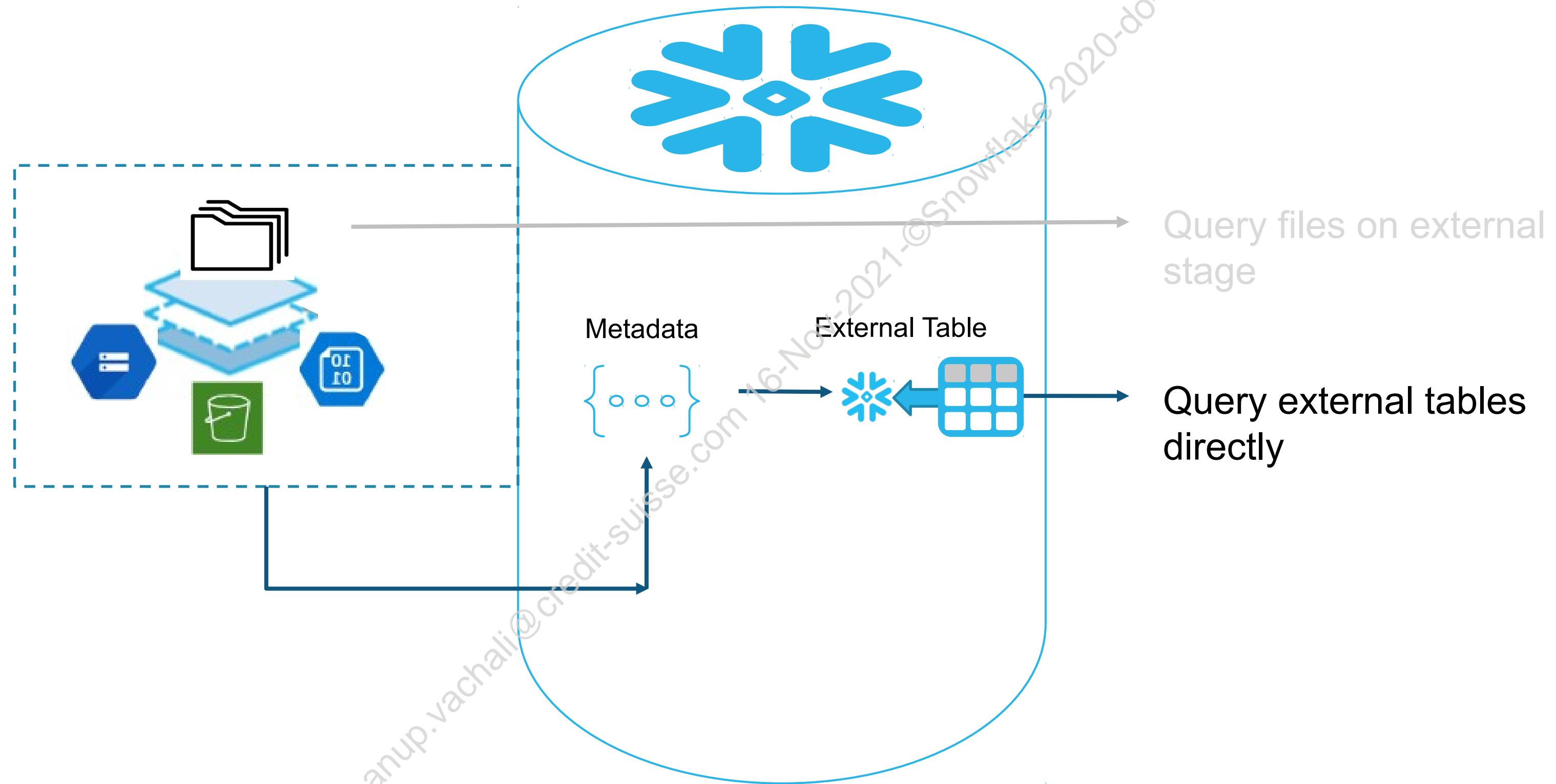
Fastest



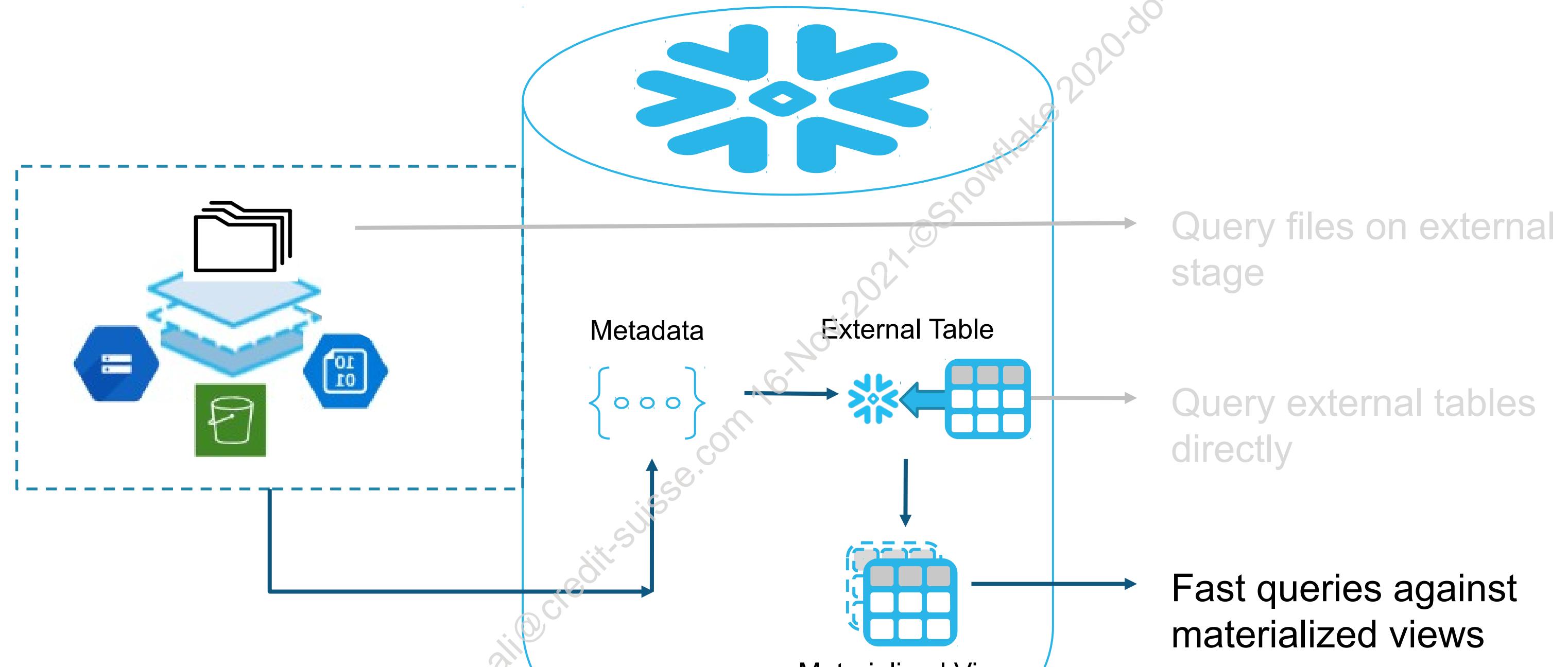
EXTERNAL DATA: QUERY DIRECTLY



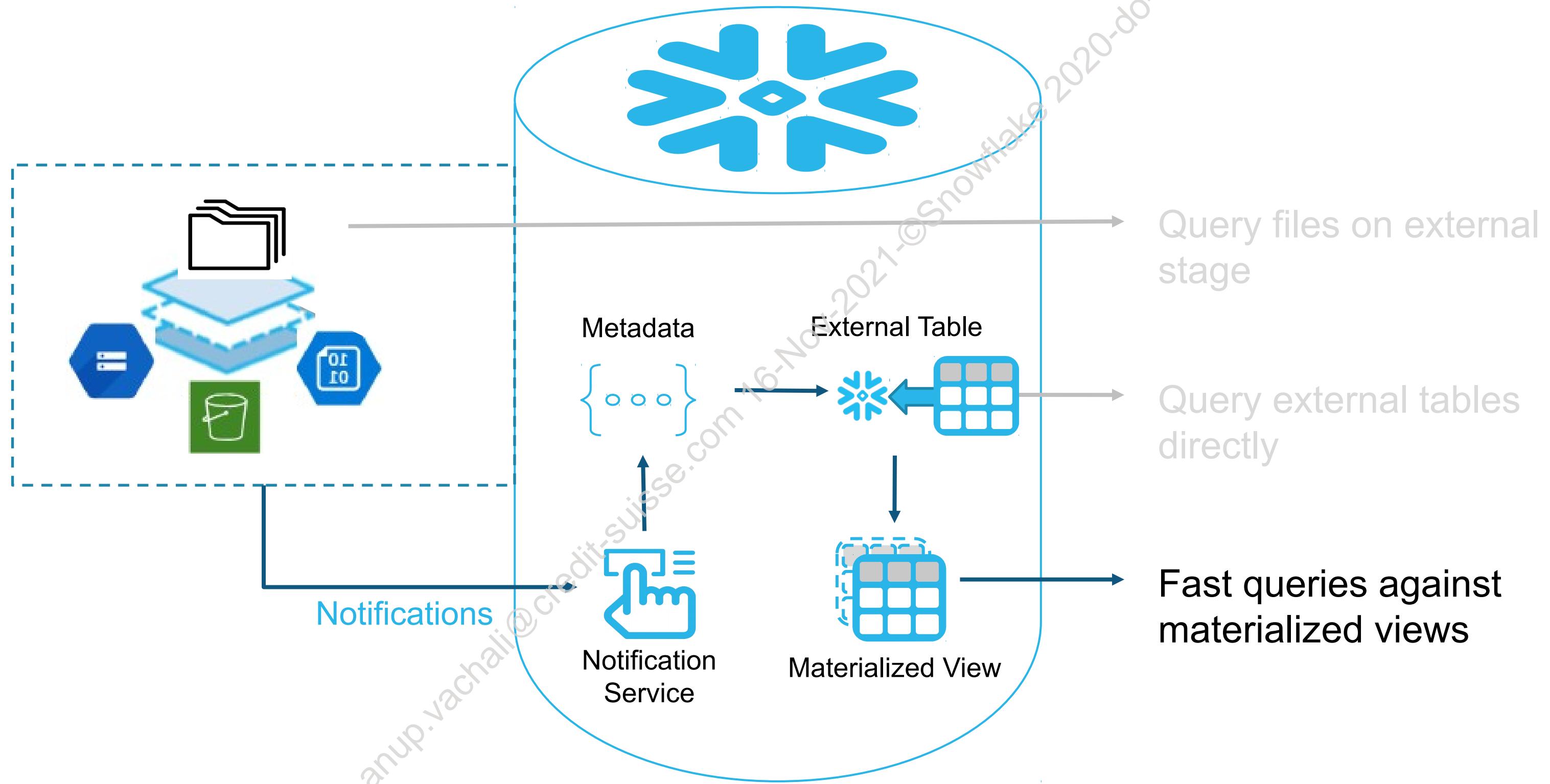
EXTERNAL DATA: EXTERNAL TABLE



EXTERNAL DATA: EXTERNAL TABLE + MV



UPDATING DATA LAKE INFORMATION



EXTERNAL TABLES

anup.vachali@credit-suisse.com 16-Nov-2021-©Snowflake 2020-do-not-copy



CREATE EXTERNAL TABLE

- Create stage where external table files reside

```
CREATE STAGE my_external_stage  
URL='<path>'  
...;
```

- Create external table, optionally with partitioning

```
CREATE EXTERNAL TABLE my_external_table  
LOCATION = @my_external_stage  
PARTITIONED BY (  
    order_date STRING AS split_part(METADATA$FILENAME, '/', 5),  
    storeID STRING AS split_part(METADATA$FILENAME, '/', 6)  
FILE_FORMAT = (FORMAT_NAME = parquet)  
ON_ERROR = SKIP_FILE
```



EXTERNAL TABLE PARTITIONING

```
CREATE EXTERNAL TABLE my_external_table
LOCATION = @my_external_stage
PARTITIONED BY (
    order_date_part STRING AS split_part (METADATA$FILENAME, '/', 5),
    storeID_part STRING AS split_part (METADATA$FILENAME, '/', 6)
FILE_FORMAT = (FORMAT_NAME = parquet)
ON_ERROR = SKIP_FILE
```

5th field of file name, with '/' as the field delimiter

6th field of file name, with '/' as the field delimiter

File name: s3://my_bucket/usa/**2018-05-24**/**123**/sales000.csv

Field: 1 2 3 4 5 6 7

QUERY EXTERNAL TABLE

CSV FILE

SELECT

 cust_id, prod_id, store_id, order_date, total_amt

FROM my_external_table

WHERE

storeId_part = 234 AND

order_date_part > '2020-06-30' AND

order_date_part < '2020-12-31';

Rough query "pruning" in
WHERE clause, which
identifies files to use



MATERIALIZED VIEW ON EXTERNAL TABLE

FOR IMPROVED QUERY PERFORMANCE

```
CREATE MATERIALIZED VIEW US_ext_orders AS  
SELECT cust_id, prod_id, store_id, order_date, total_amt  
FROM my_external_table  
WHERE  
    storeId_part < 5000 AND  
    order_date_part > '2020-06-30' AND  
    order_date_part < '2020-12-31';
```

```
SELECT * FROM US_ext_orders  
WHERE prod_id = 'A22354';
```



SECURITY

- All security features available for regular tables (RBAC, privileges) are also available for external tables
- All encryption options supported by external stage are supported by external tables
- There is separate privilege to create external tables

GRANT CREATE EXTERNAL TABLE

```
ON EXTERNAL_SCHEMA  
TO ROLE project_admin;
```



LAB EXERCISE: 10

Work with External Tables

35 minutes

- Unloading Data to Cloud Storage as a Data Lake
- Unload Date to Cloud Storage Using Different Virtual Warehouses
- Executing Queries Against External Tables
- Working with External Tables
- Create a Partitioned External Table



OPTIMIZE QUERY PERFORMANCE

anup.vachali@credit-suisse.com Nov-2021 ©Snowflake 2020-do-not-copy



MODULE AGENDA

- Explain and the Query Profile
- Automatic Clustering
- Search Optimization
- Materialized Views
- Optimize Warehouse Utilization

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



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

Status	Query ID	SQL Text
Running	0199e4e7...	with cross_items as (... DPL)
✓	0199e4e7...	use snowflake_sampl... DPL
✗	0199e3a9...	grant role intern to us... DPL
✓	0199e3a9...	show users; DPL

Worksheet Result Pane

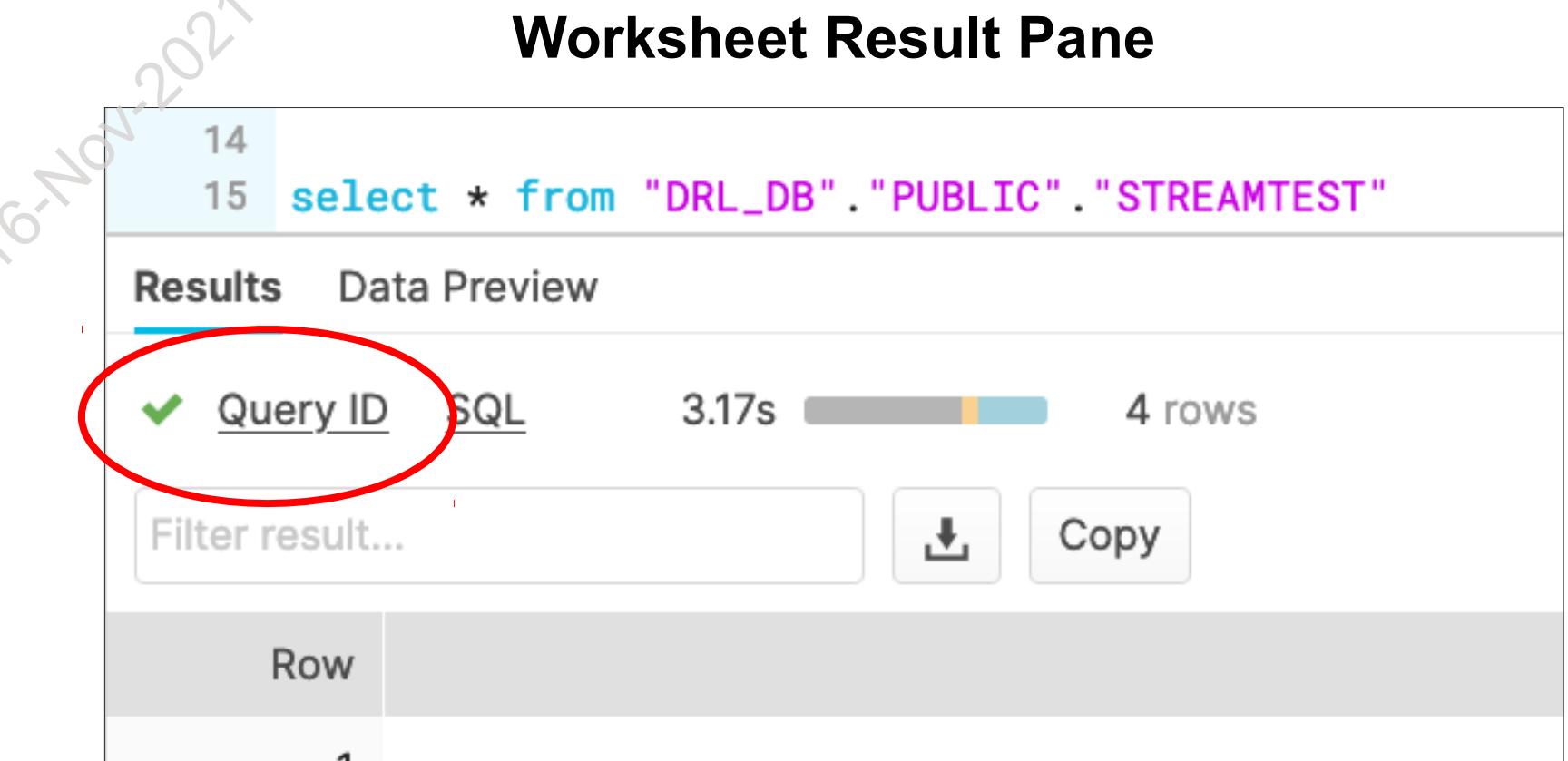
14
15 `select * from "DRL_DB"."PUBLIC"."STREAMTEST"`

Results Data Preview

✓ [Query ID](#) SQL 3.17s 4 rows

Filter result... Row 1

Copy



QUERY PROFILE

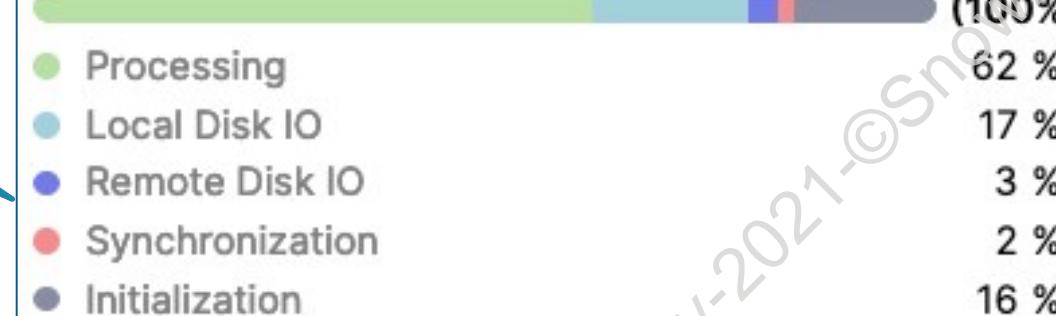
GREAT STARTING POINT FOR EVALUATING PERFORMANCE

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)



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_DATE...	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_DATE...	NULL	NULL	NULL
1	6	5	TableScan	SNOWFLAKE_S...	SS	SS_SOLD_DATE_SK	722313	1840	12017734006784

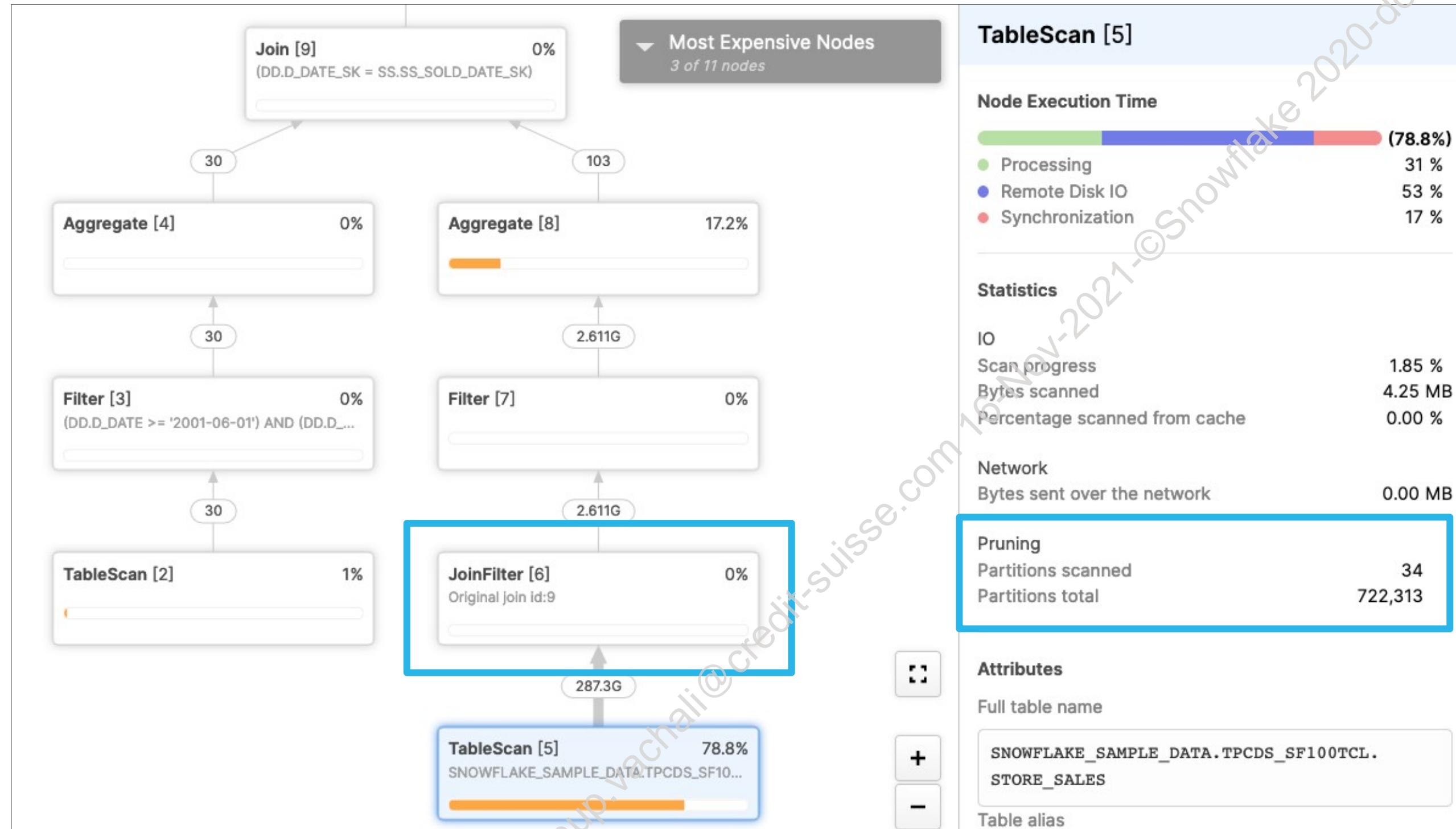


STATIC PRUNING SHOWN IN RESULTS

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

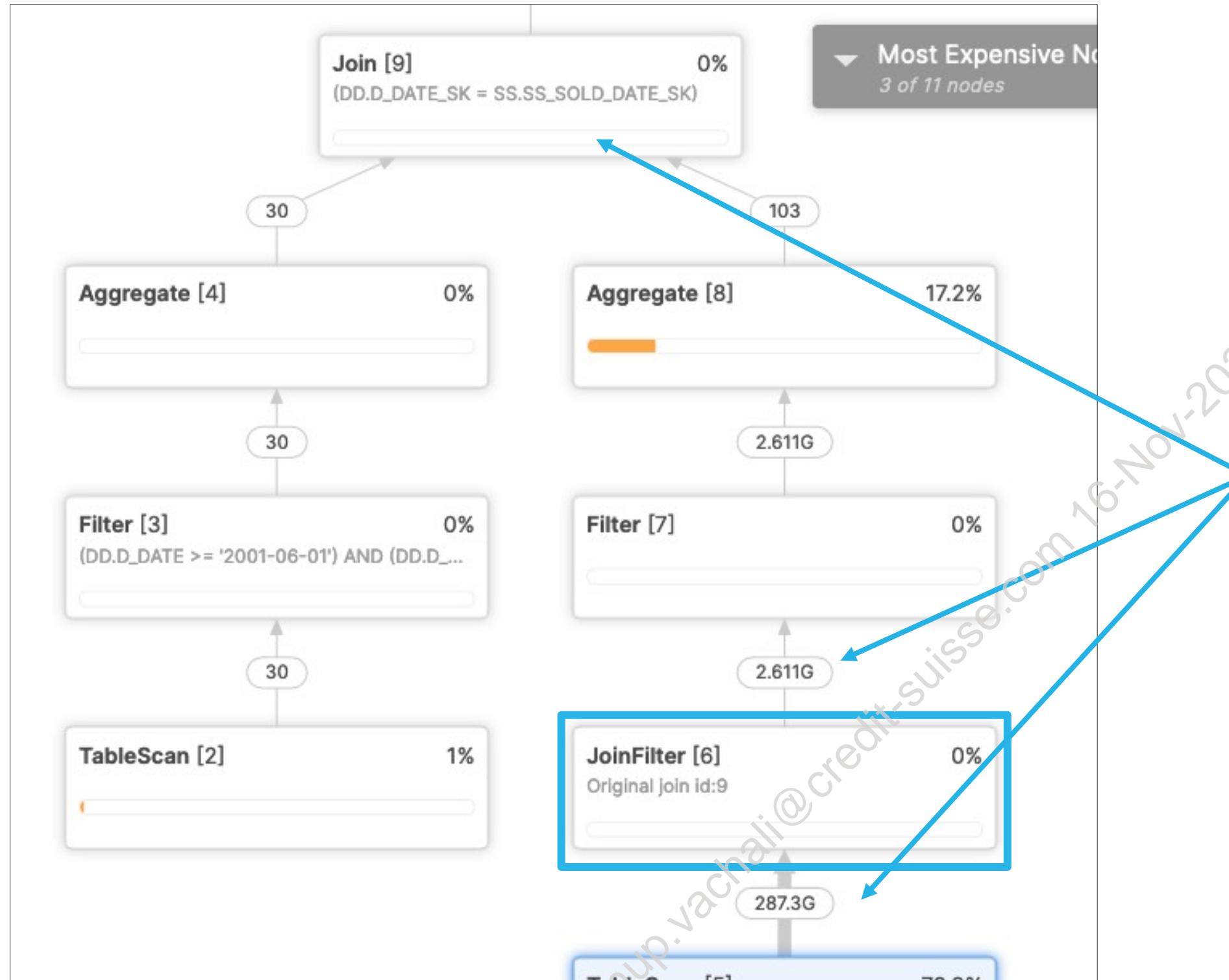


QUERY MAY PRUNE FURTHER AT RUN TIME



- 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

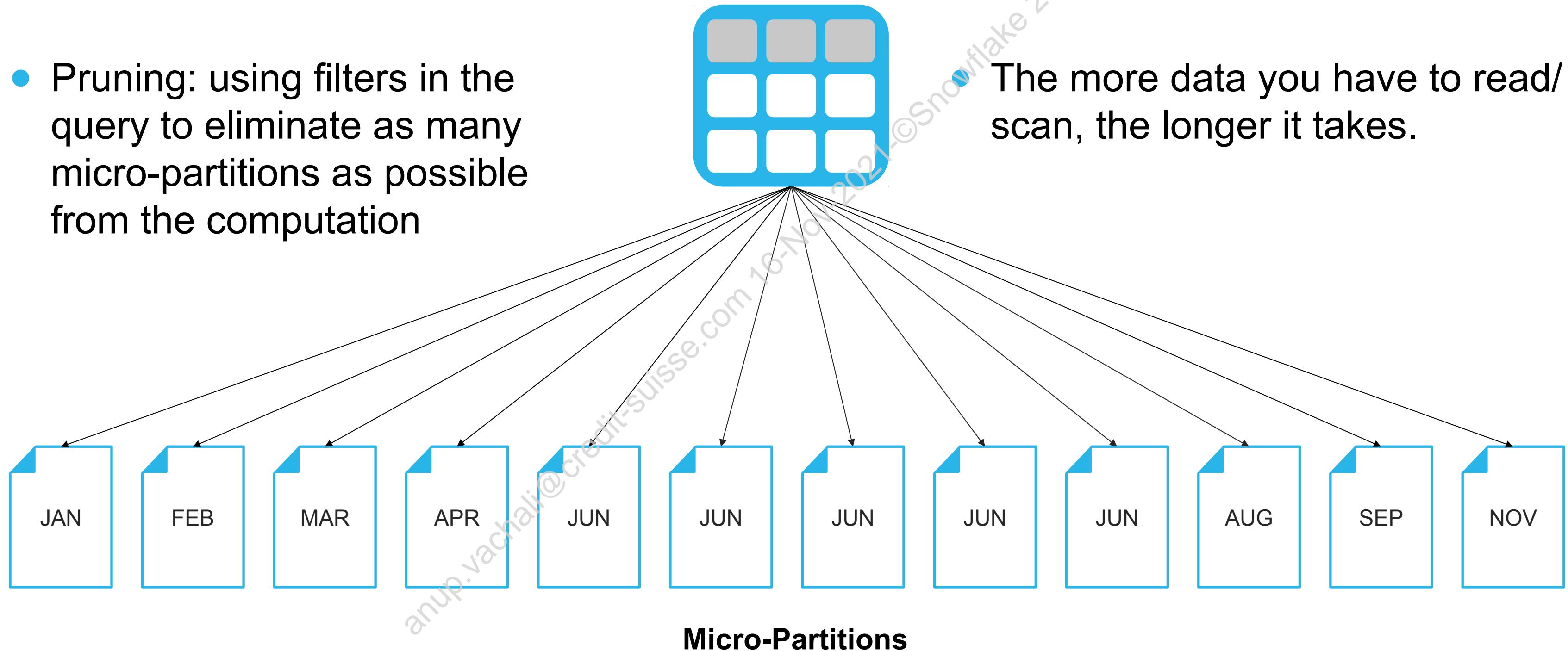
anup.vachali@credit-suisse.com 16-Nov-2021-©Snowflake 2020-do-not-copy



WHAT IS QUERY PRUNING?

`SELECT * FROM sales;`

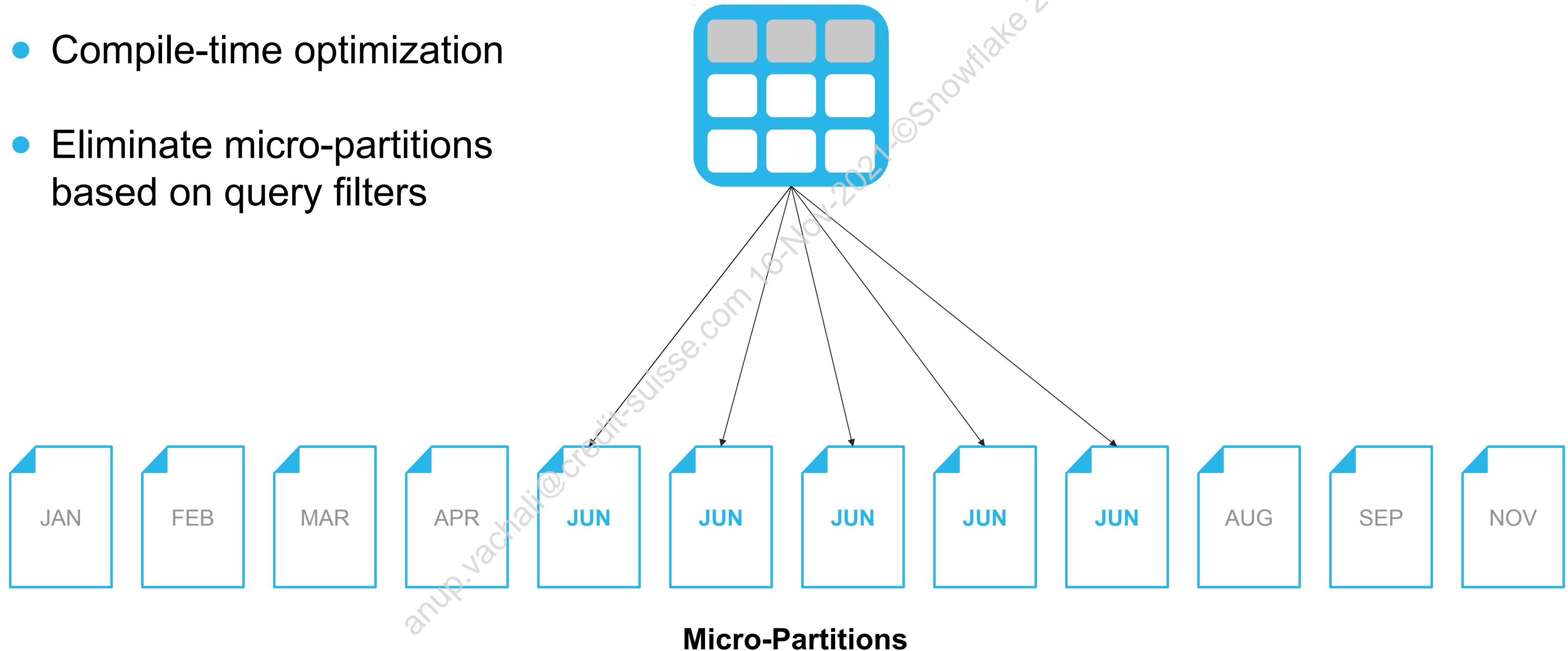
- Pruning: using filters in the query to eliminate as many micro-partitions as possible from the computation



STATIC QUERY PRUNING

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

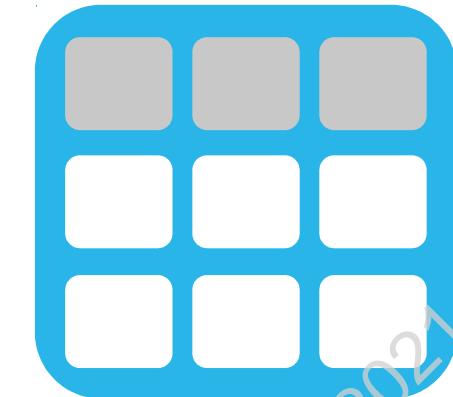
- Compile-time optimization
- Eliminate micro-partitions based on query filters



DYNAMIC QUERY PRUNING

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

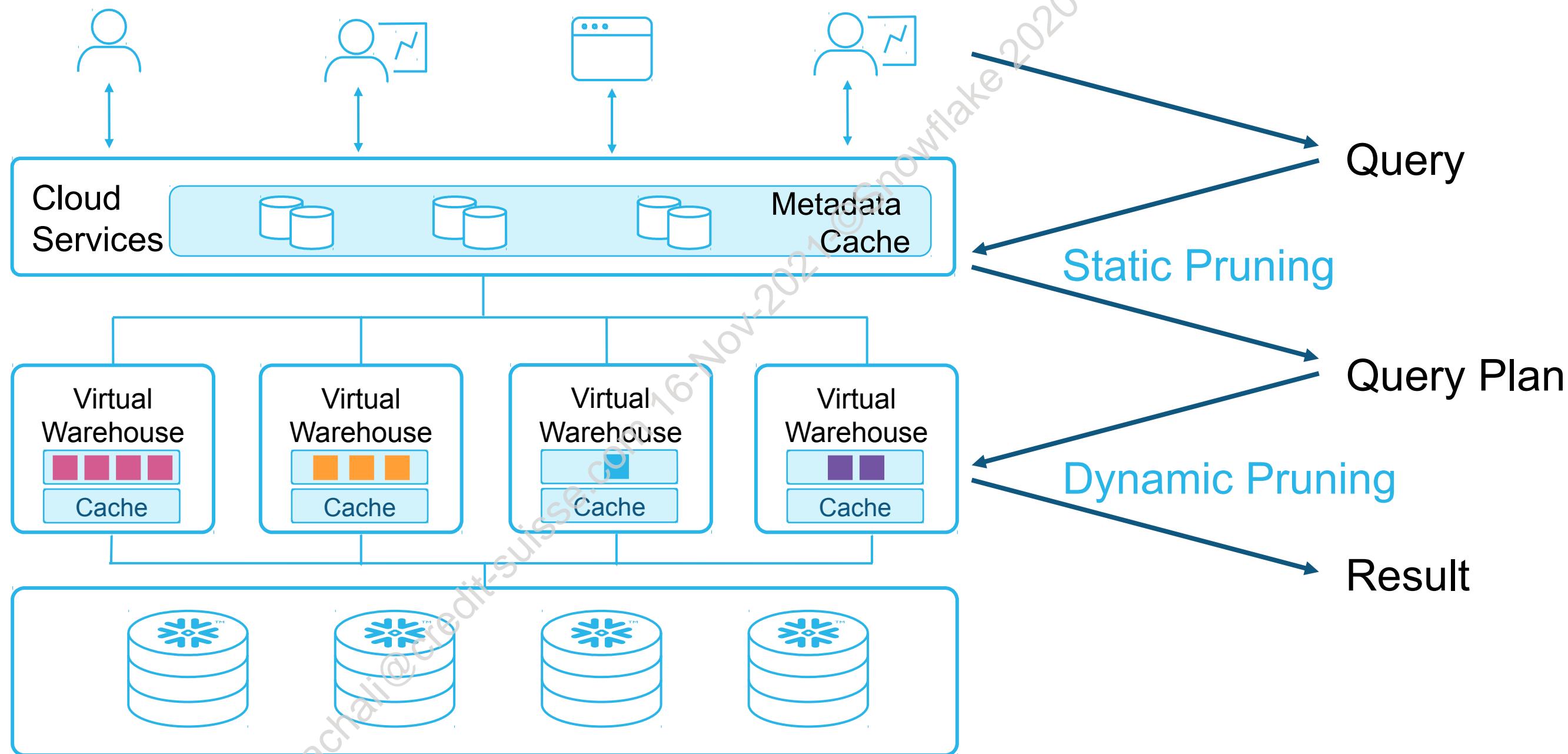
- Run-time optimization
- Push down JOIN filter to remove tuples identified as not possibly matching the JOIN condition



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

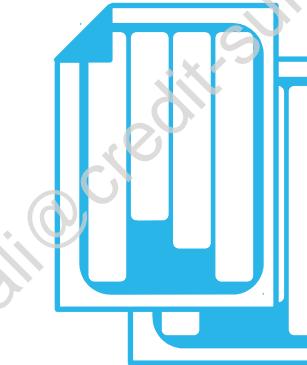
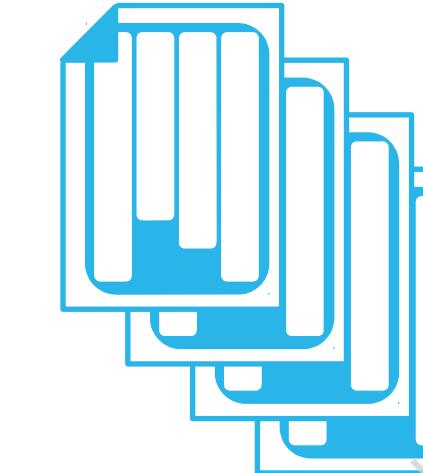
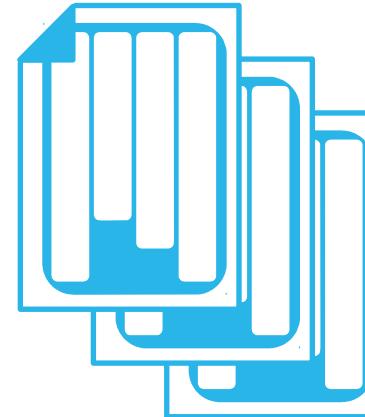
January - December

Records Contain
order dates that span

January - December

Records Contain
order dates that span

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

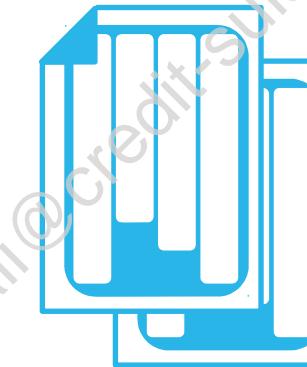
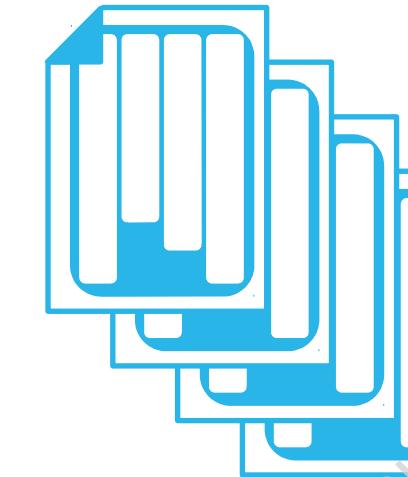
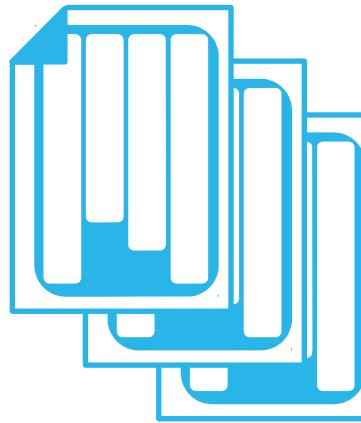
Records Contain
order dates that span

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
```

Poorly clustered for filters on tested column

```
"partition_depth_histogram": {  
    "00000": 0,  
    "00001    "00002    "00003    "00004    "00005    "00006    "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 ($\text{MAX} - \text{MIN}$)

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 ($\text{MAX} - \text{MIN}$)

Clustering key = **AGE**

Micropartition 1: Wide on age column

Sam	18	USA
Trevor	78	Canada



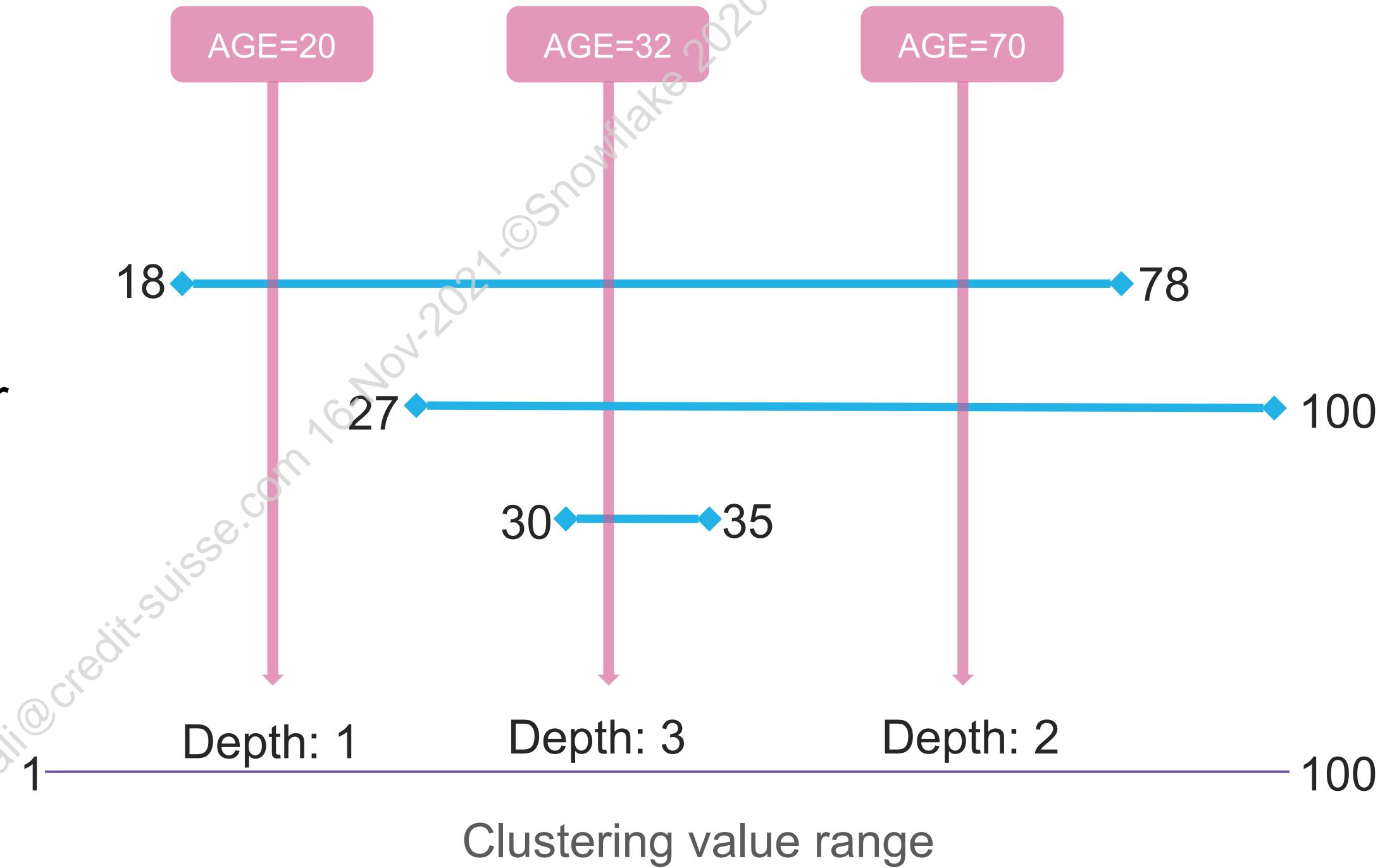
Micropartition 2: Narrow on age column

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);
```

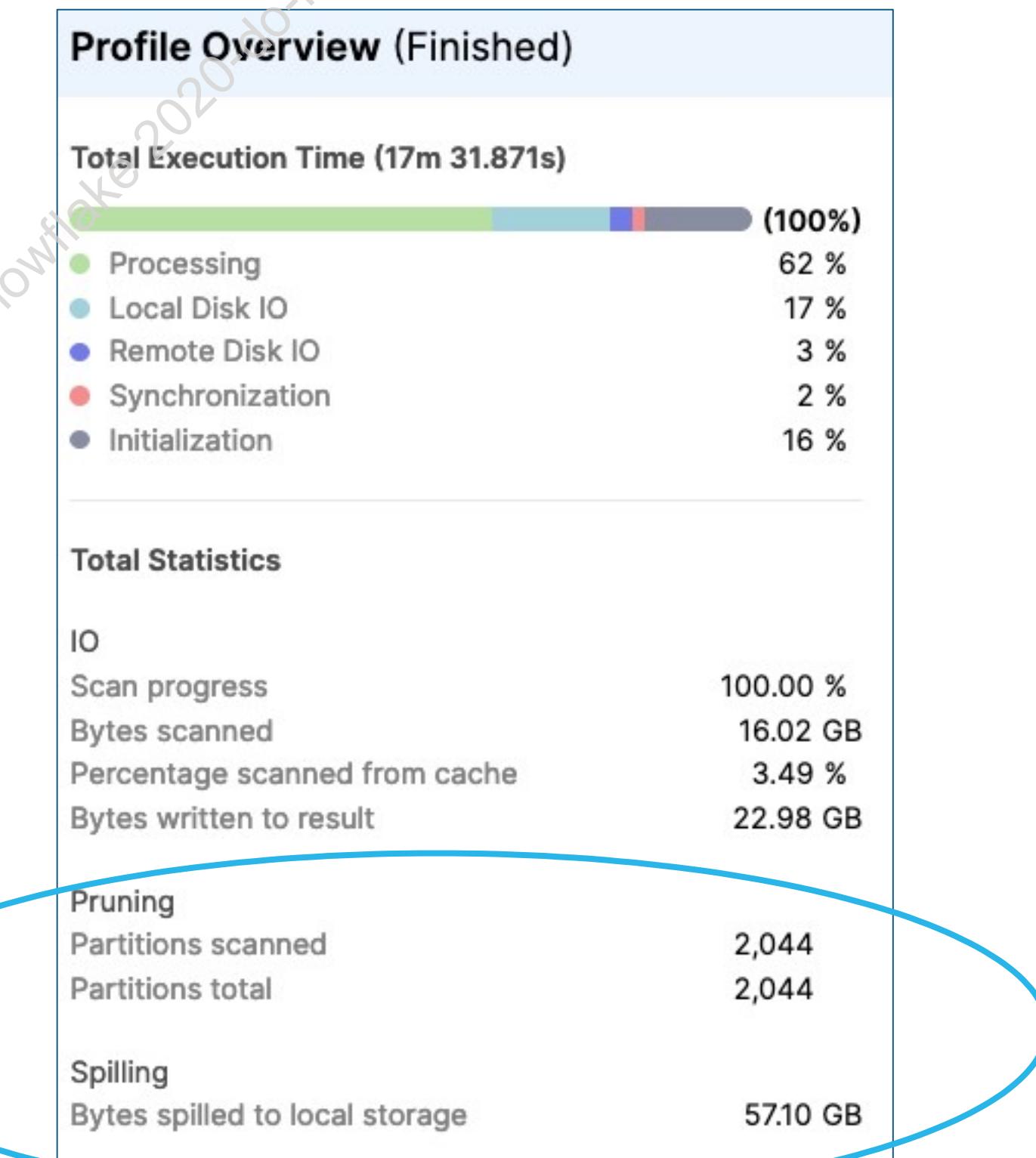
```
ALTER TABLE t2  
CLUSTER BY (SUBSTRING(c2, 5, 2), TO_DATE(c1));
```



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



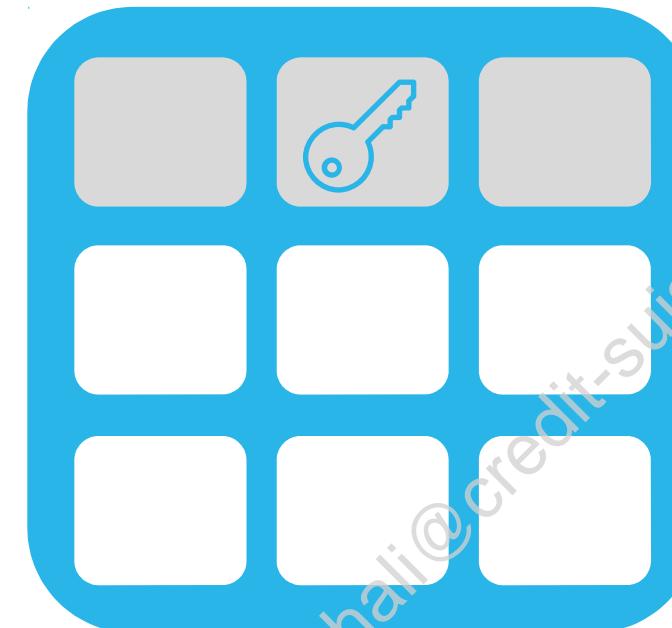
METHODOLOGY FOR EXPLICIT CLUSTERING

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

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
(STATUS, STORE_NO)

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

Clustered by
(STORE_NO, STATUS)



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

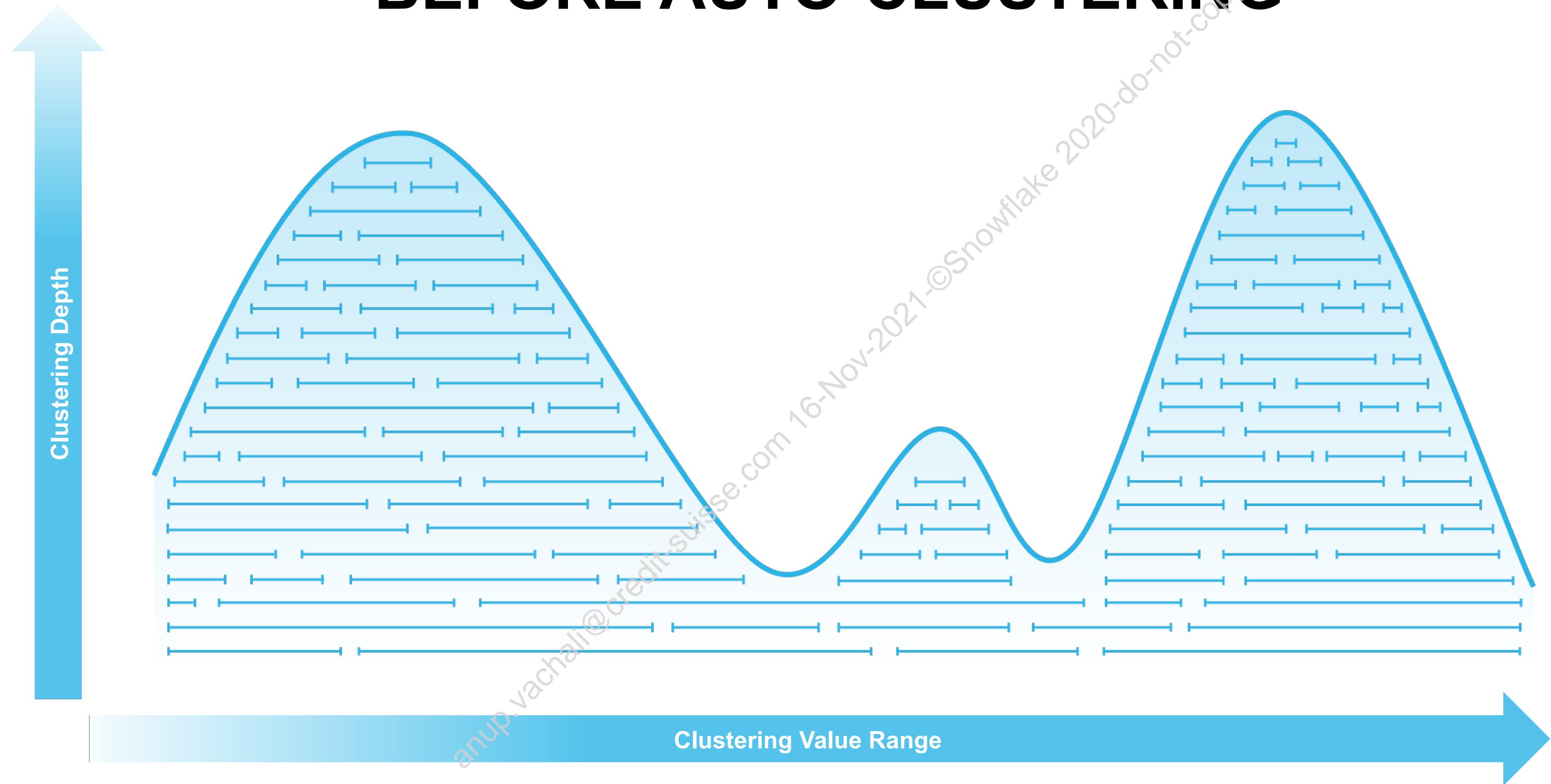
Reduce **Worst** Clustering Depth

below an acceptable threshold to get

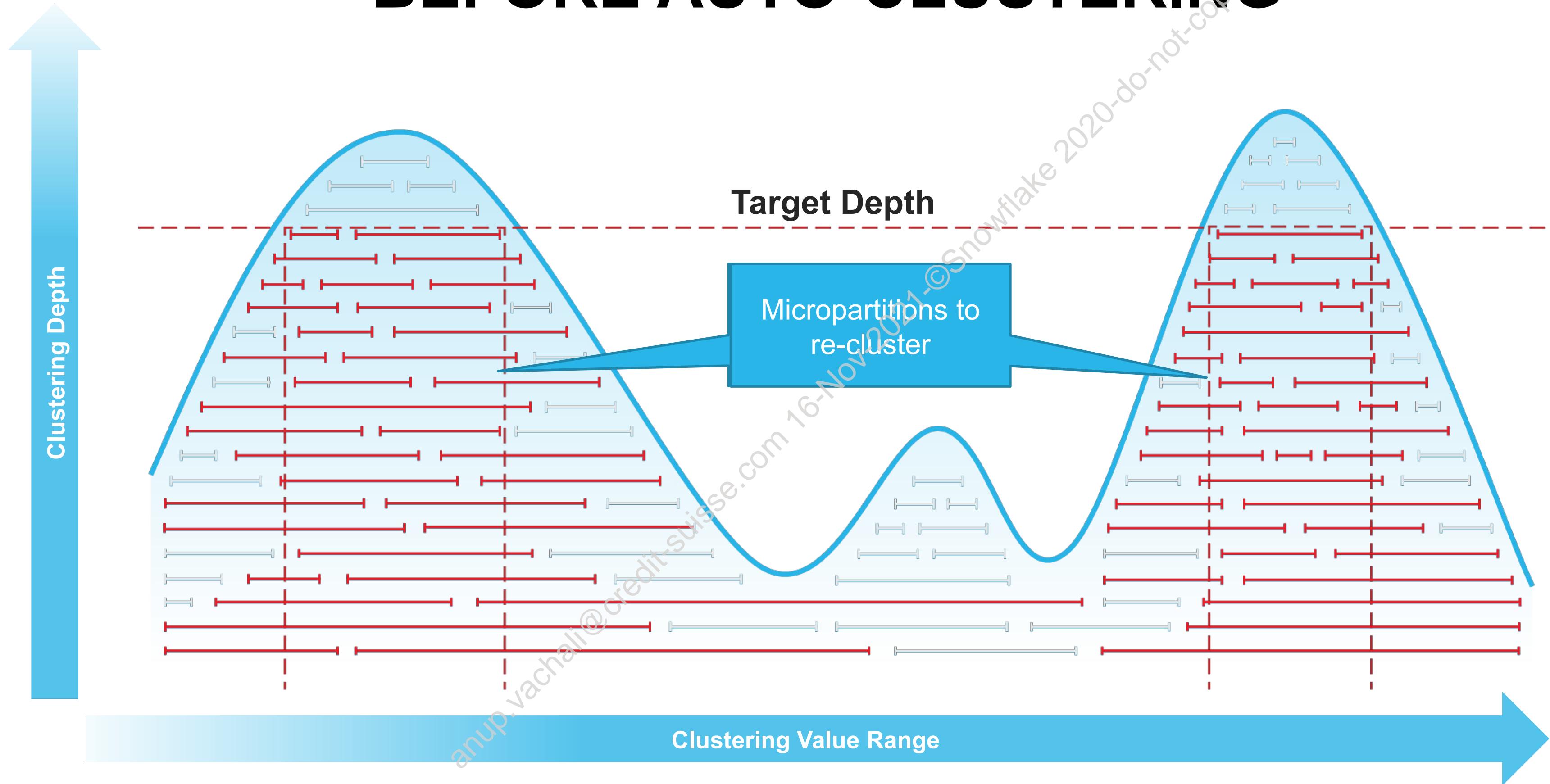
Predictable Query Performance



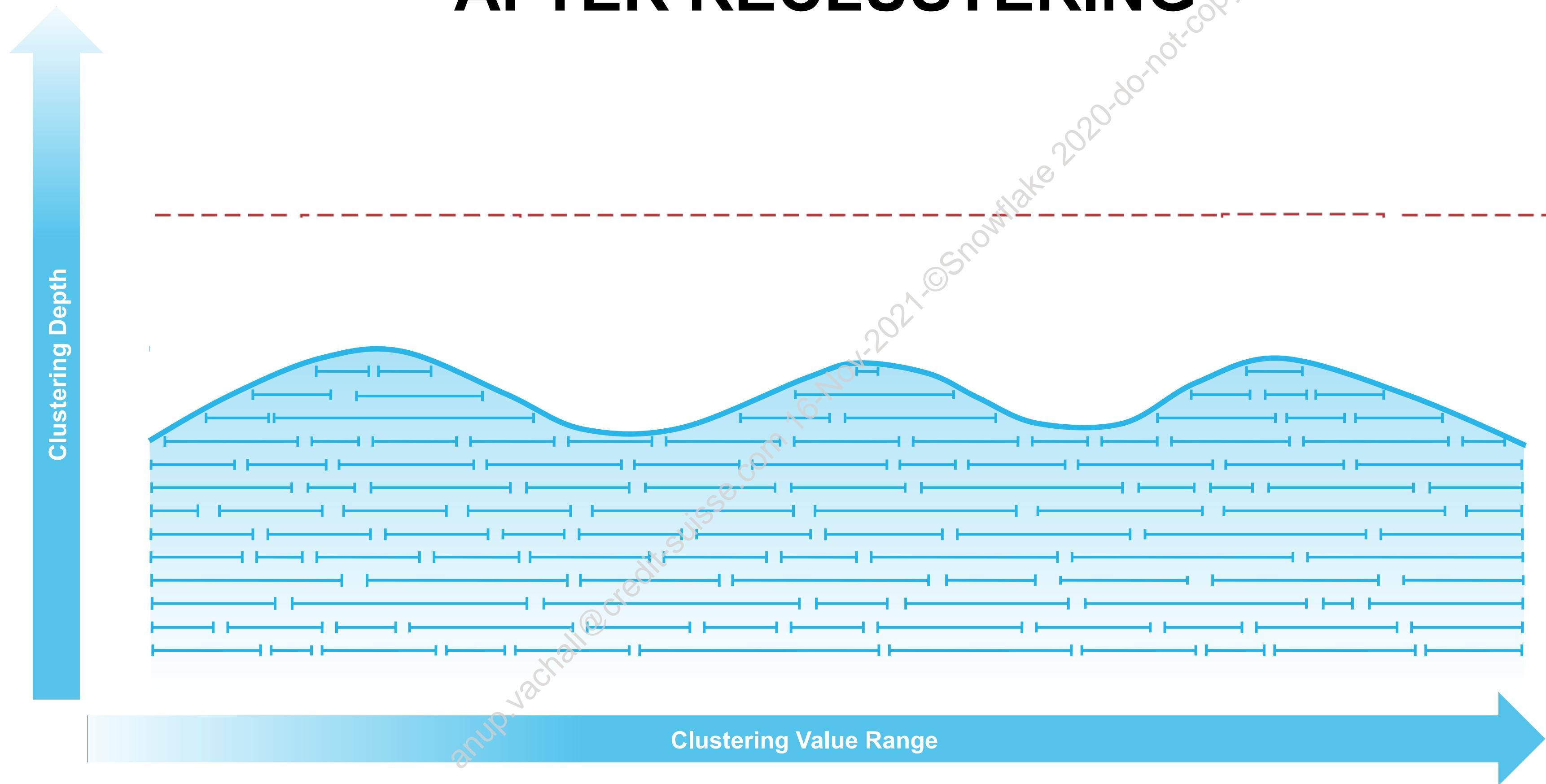
BEFORE AUTO-CLUSTERING



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

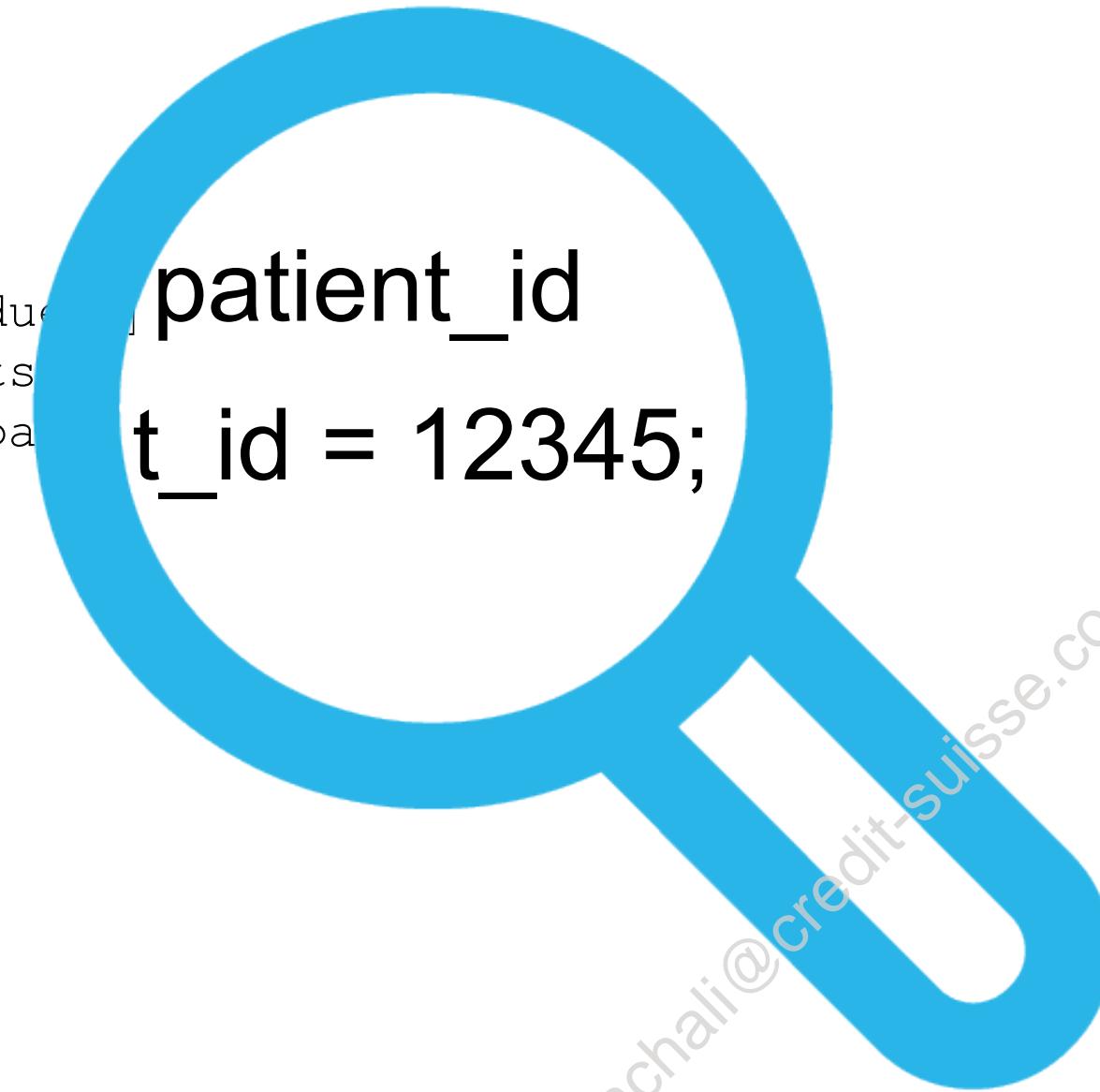
anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



OVERVIEW

WHEN TO USE SEARCH OPTIMIZATION

```
SELECT bal_due  
FROM patients  
WHERE pa
```

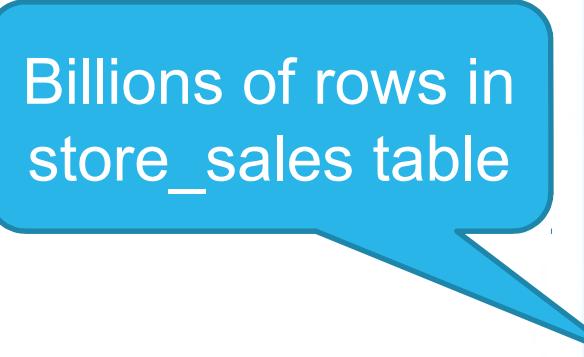


- Equality searches
 - Also known as point query or fast lookup query
 - Returns only a small % of rows
- Tune tables with frequent selective queries using Search Optimization

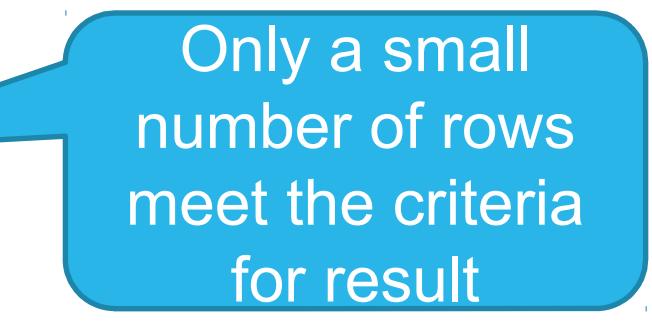
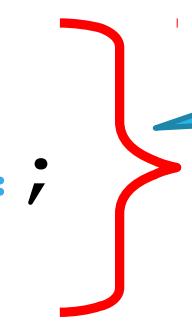


SELECTIVE QUERY EXAMPLE

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



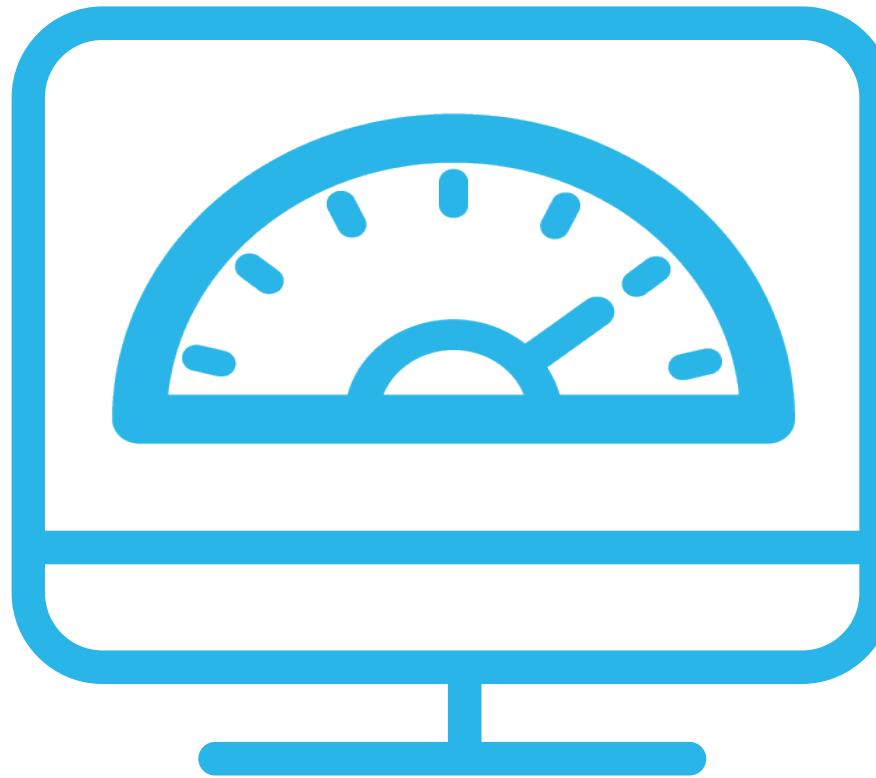
```
-- Find the sales ticket, items sold and sale date  
-- for one customer  
  
SELECT ss_ticket_number, ss_sold_date_sk, ss_item_sk  
FROM store_sales  
WHERE  
ss_customer_sk = 3229284;
```



Billions of rows in store_sales table

Only a small number of rows meet the criteria for result

CREATE A SEARCH PATH



- Add table to the Search Optimization service:

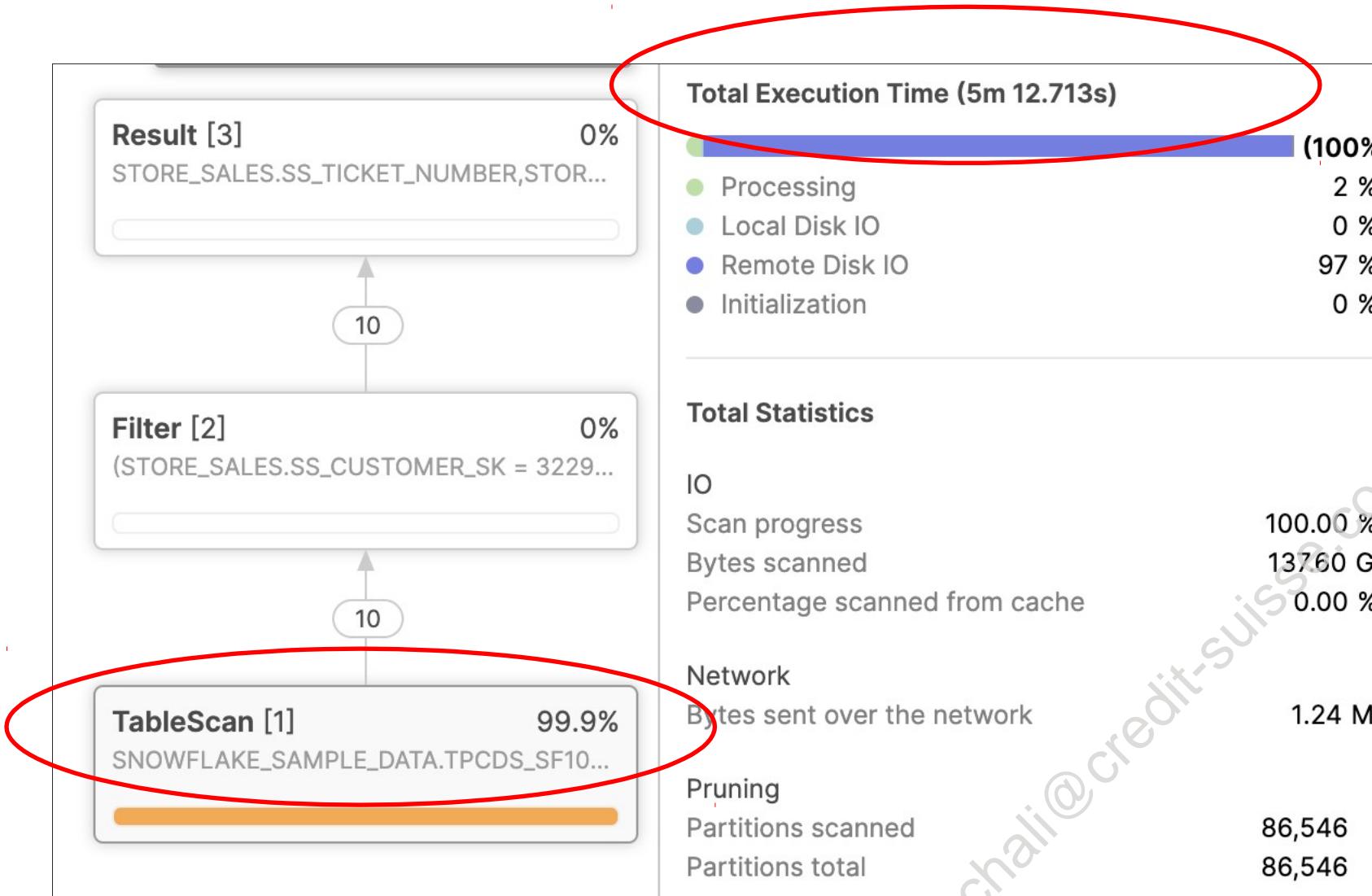
```
ALTER TABLE store_sales  
ADD SEARCH OPTIMIZATION;
```

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

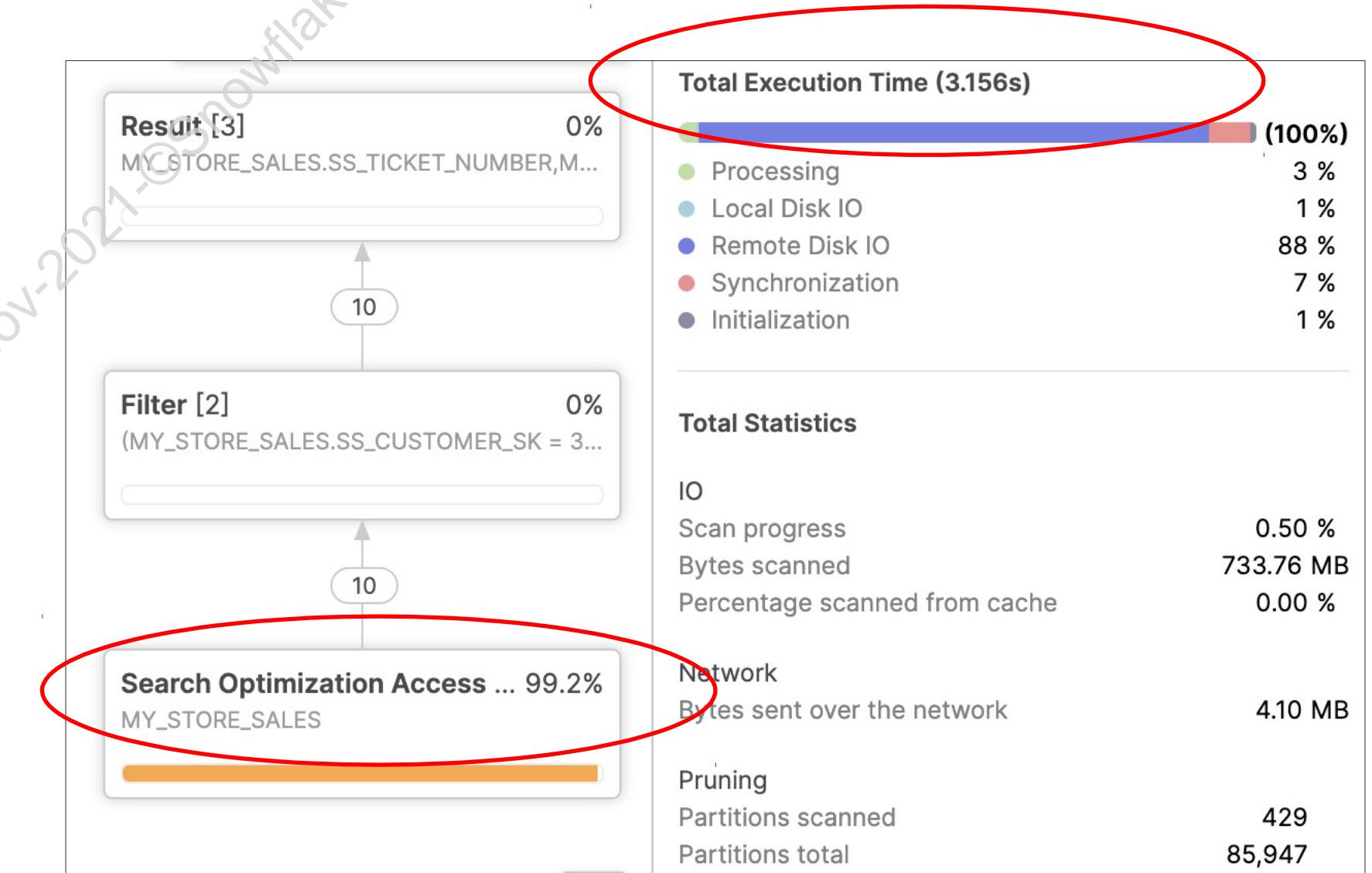


PERFORMANCE BENEFIT

Without search optimization

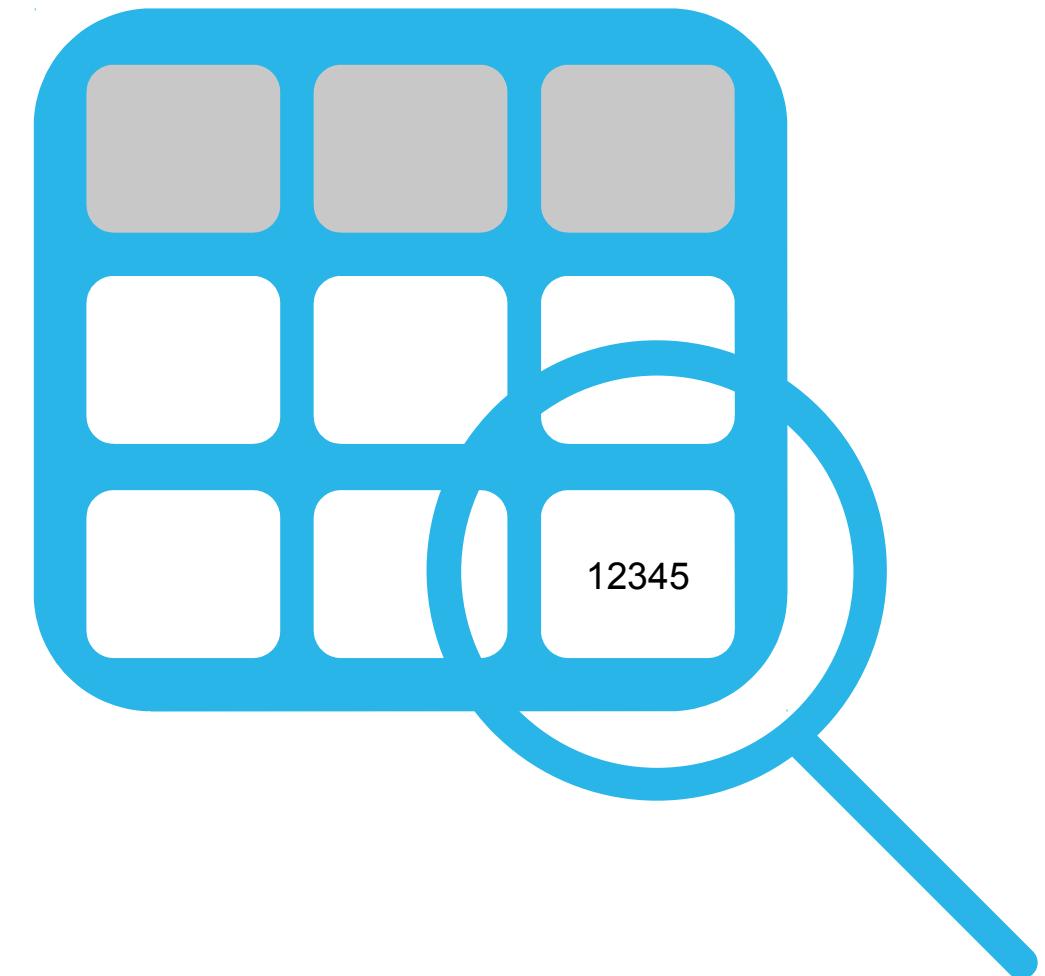


With search optimization



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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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

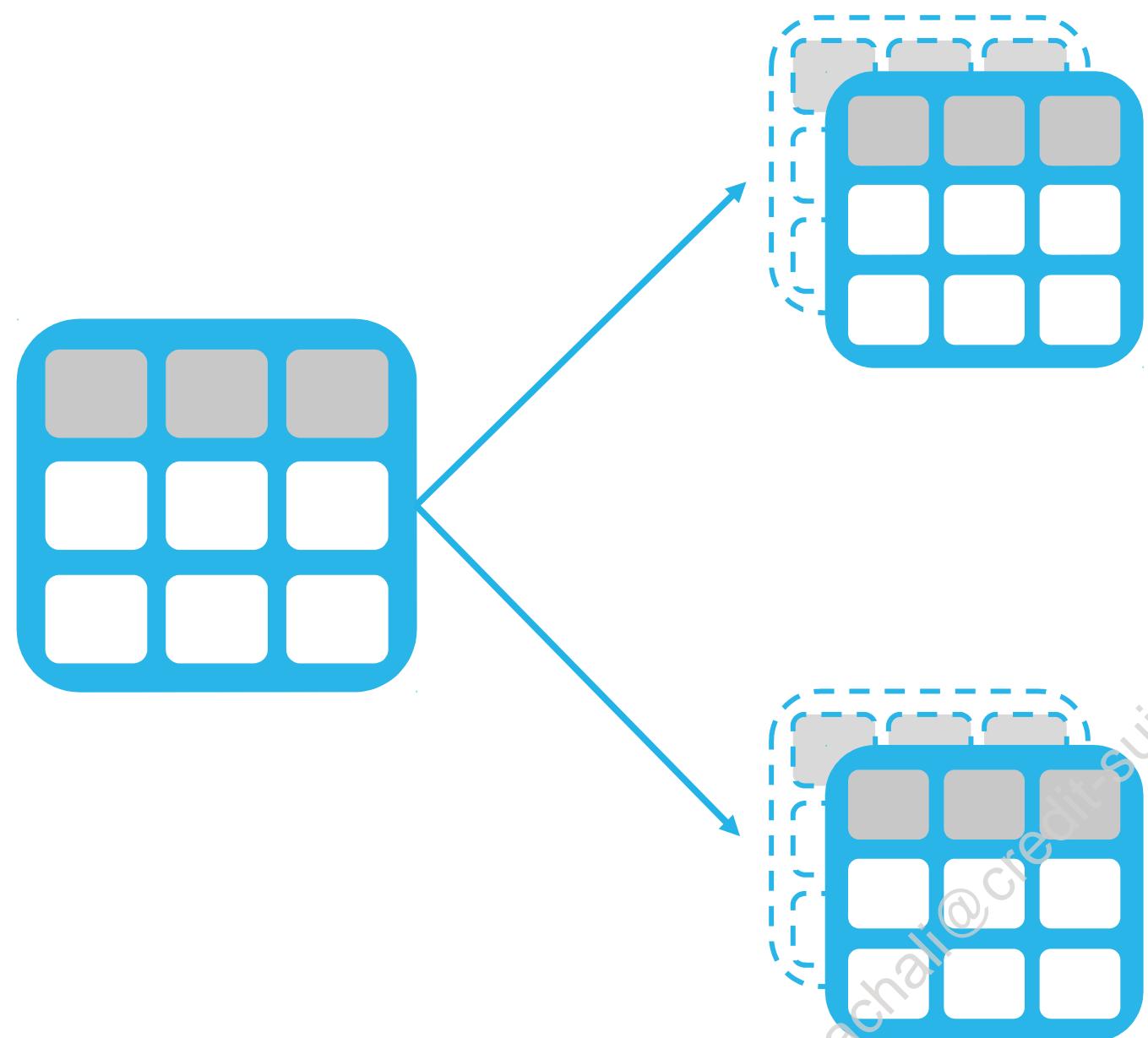


MATERIALIZED VIEWS

anup.vachali@credit-suisse.com 16 Nov 2021 ©Snowflake 2020-do-not-copy



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 MATERIALIZED VIEW mv2...
```

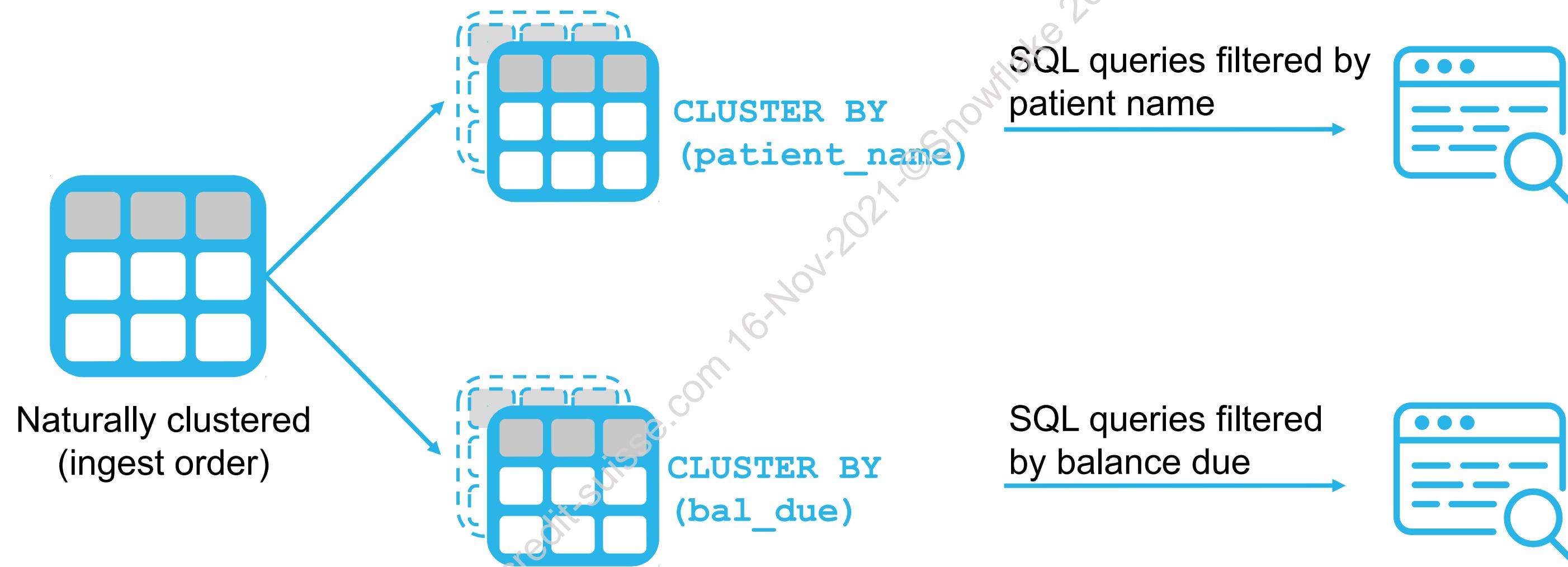
```
CLUSTER BY (bal_due)
```

```
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:

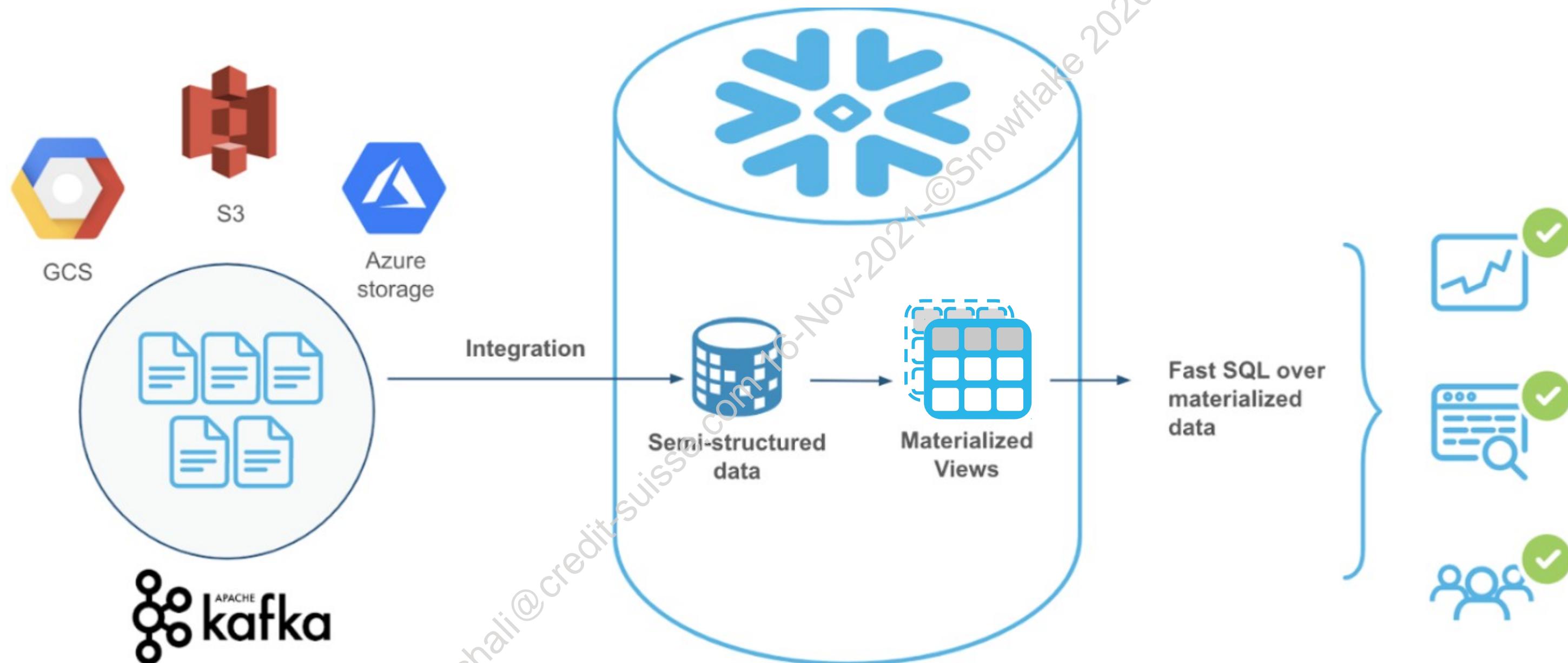
- Create a materialized view with the temporal data extracted into distinct columns, and cluster on those columns

```
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?

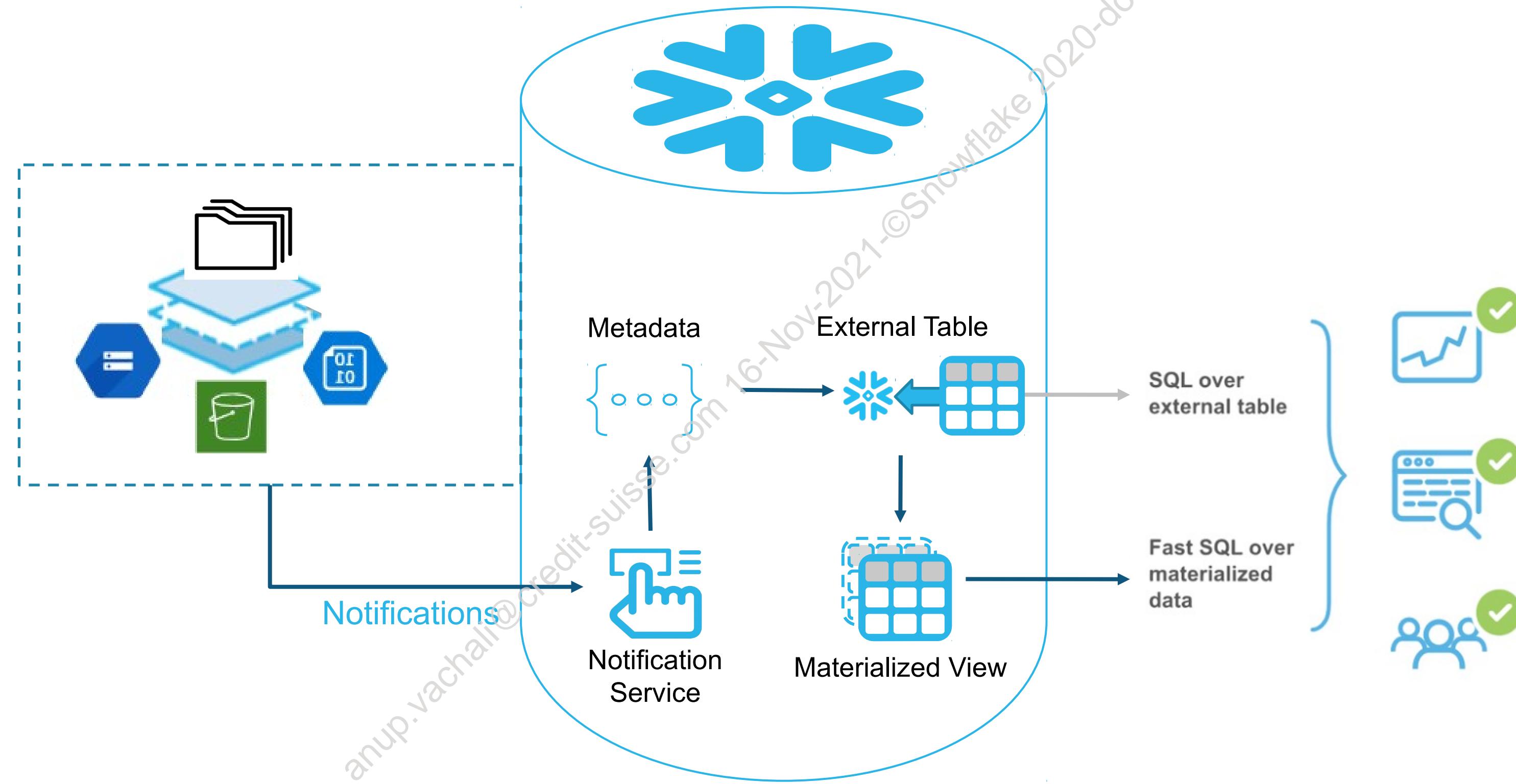
SOLUTION

- 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



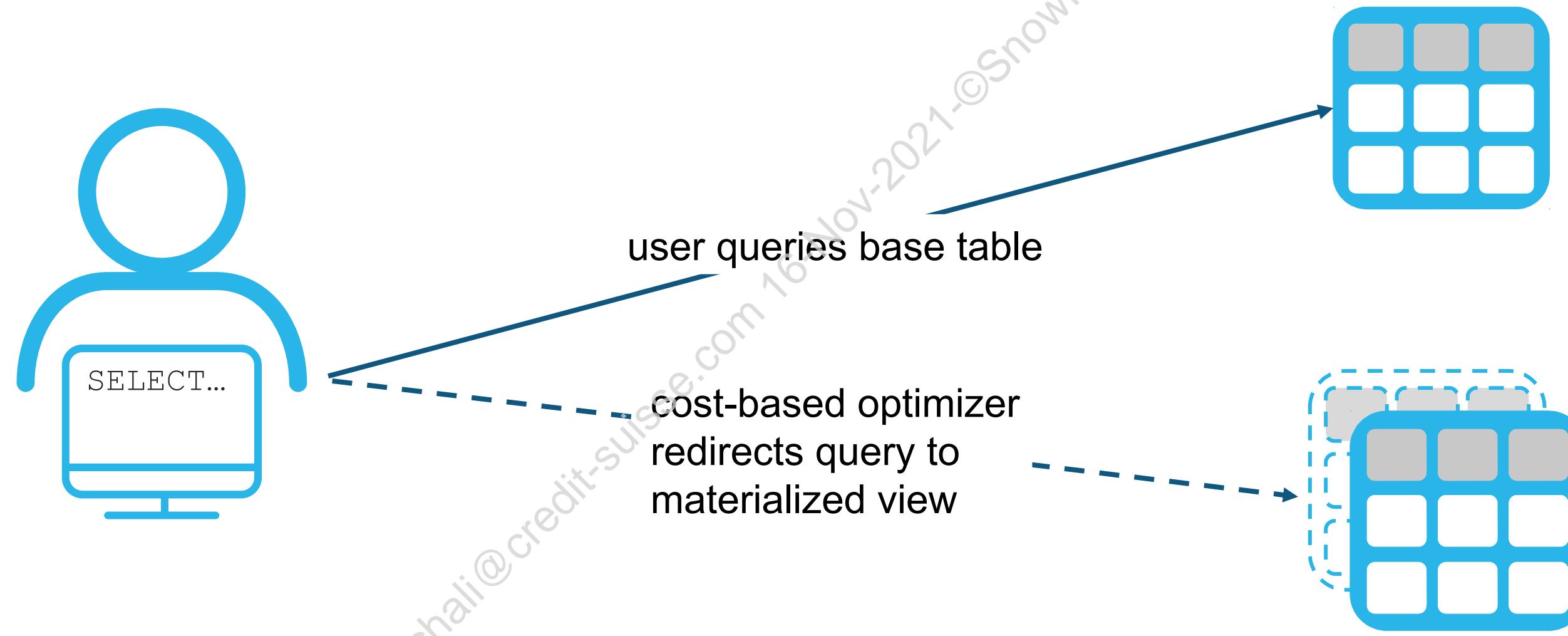
SCENARIO 3

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

15 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

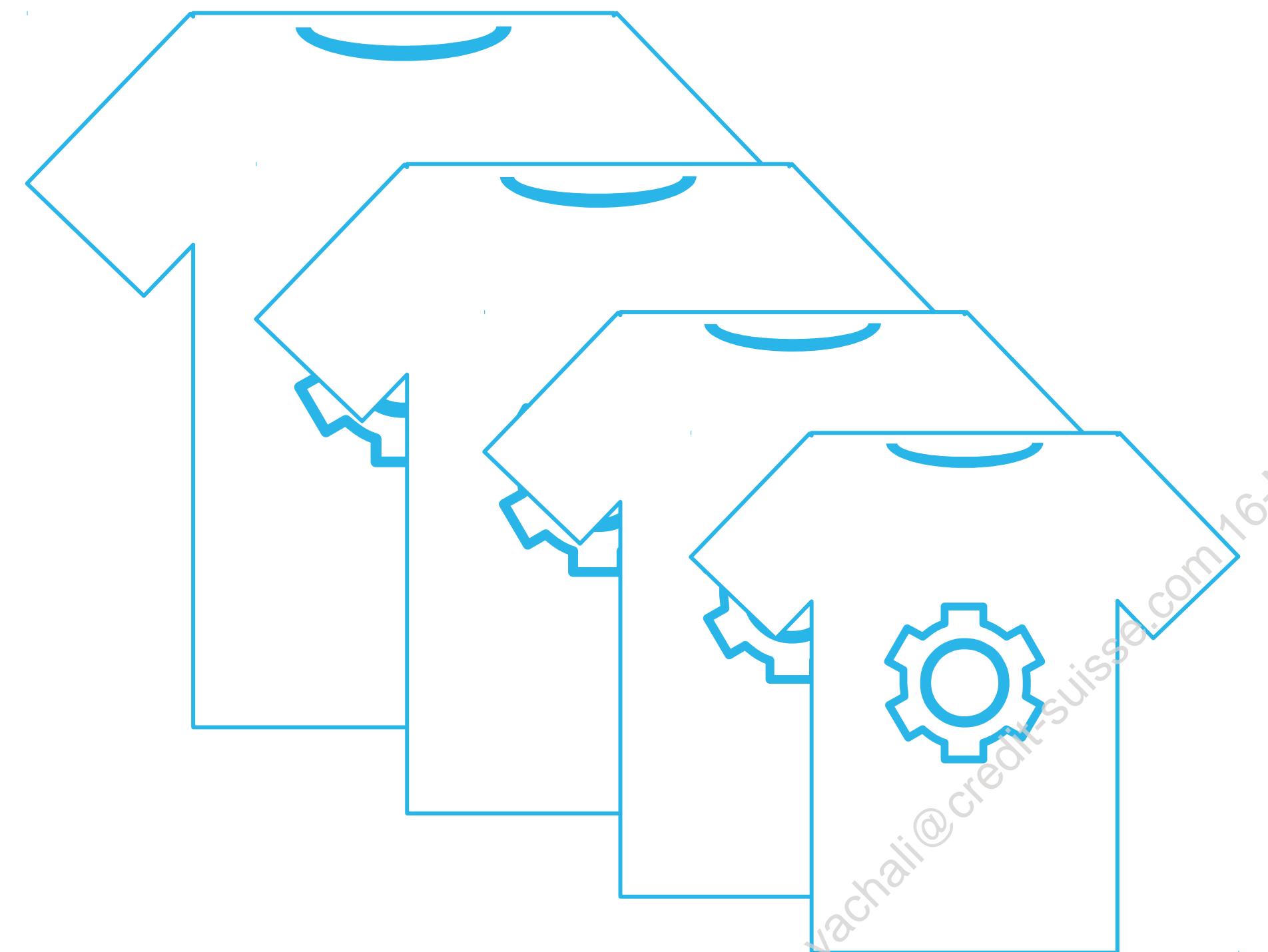


OPTIMIZE WAREHOUSE UTILIZATION

anup.vachali@credit-suisse.com Nov-2021 ©Snowflake 2020-do-not-copy



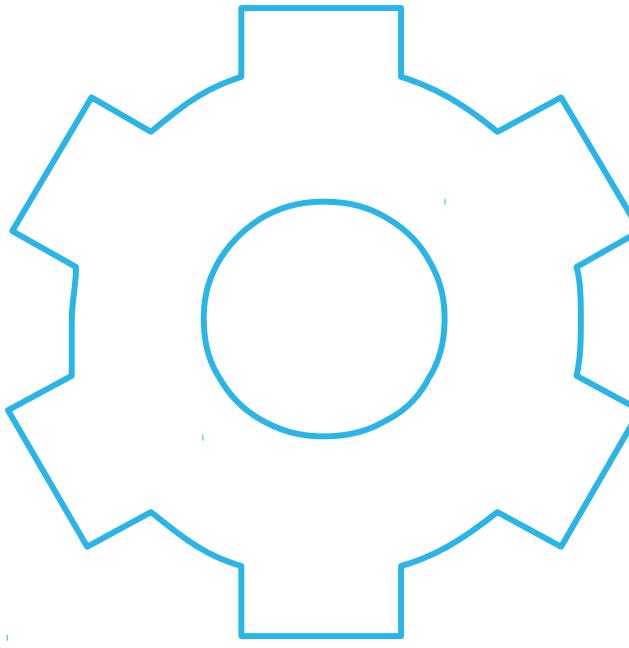
VIRTUAL WAREHOUSES



- 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

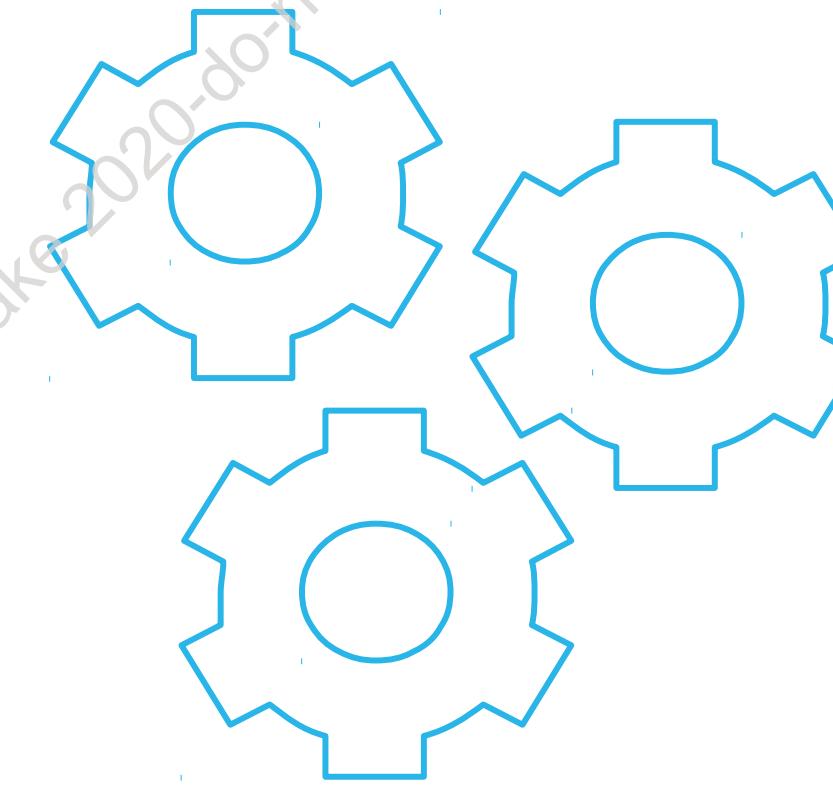


VIRTUAL WAREHOUSE TYPES



Standard

- Will only ever have a single compute cluster

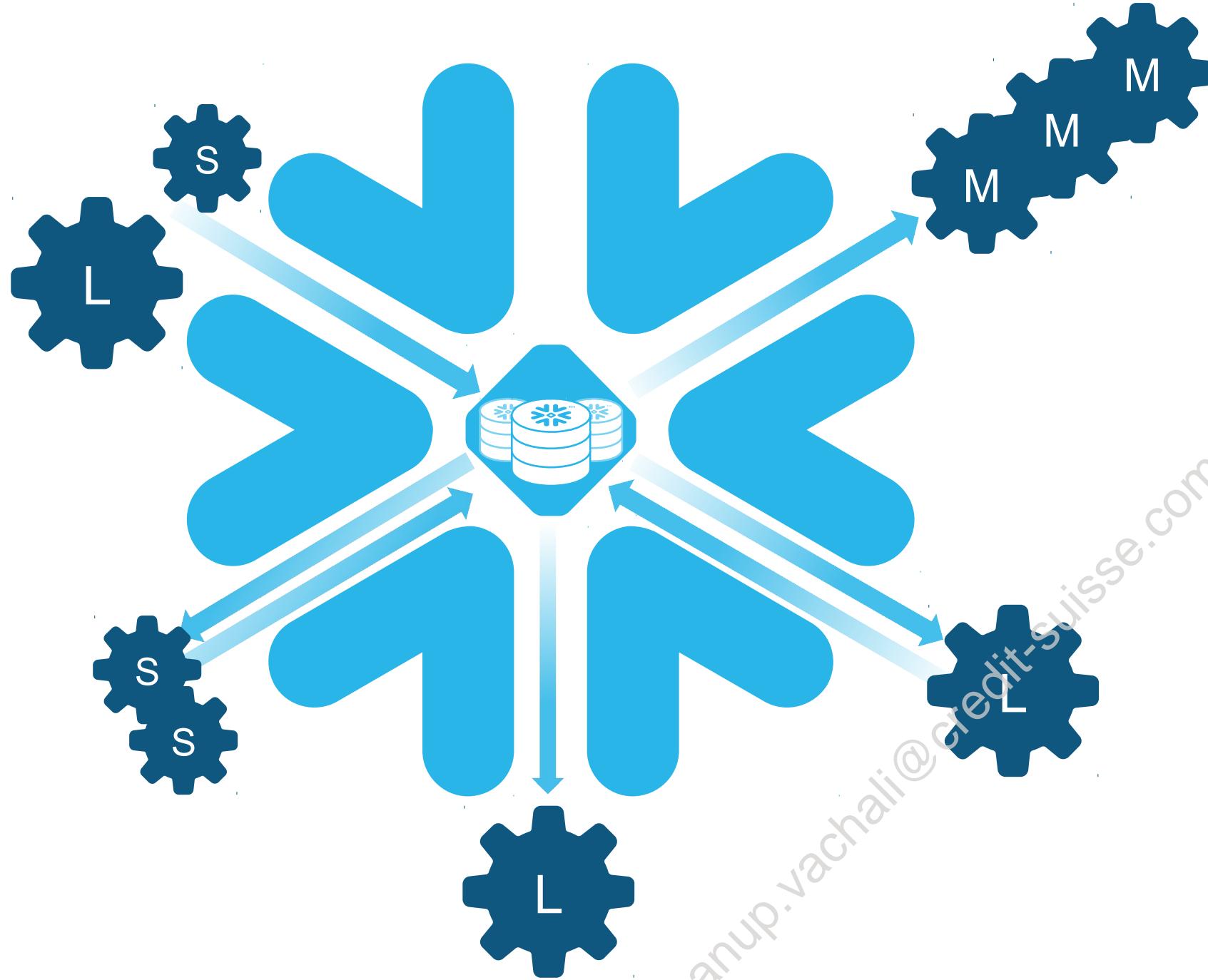


Multi-Cluster Warehouse (MCW)

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

SCALE ACROSS

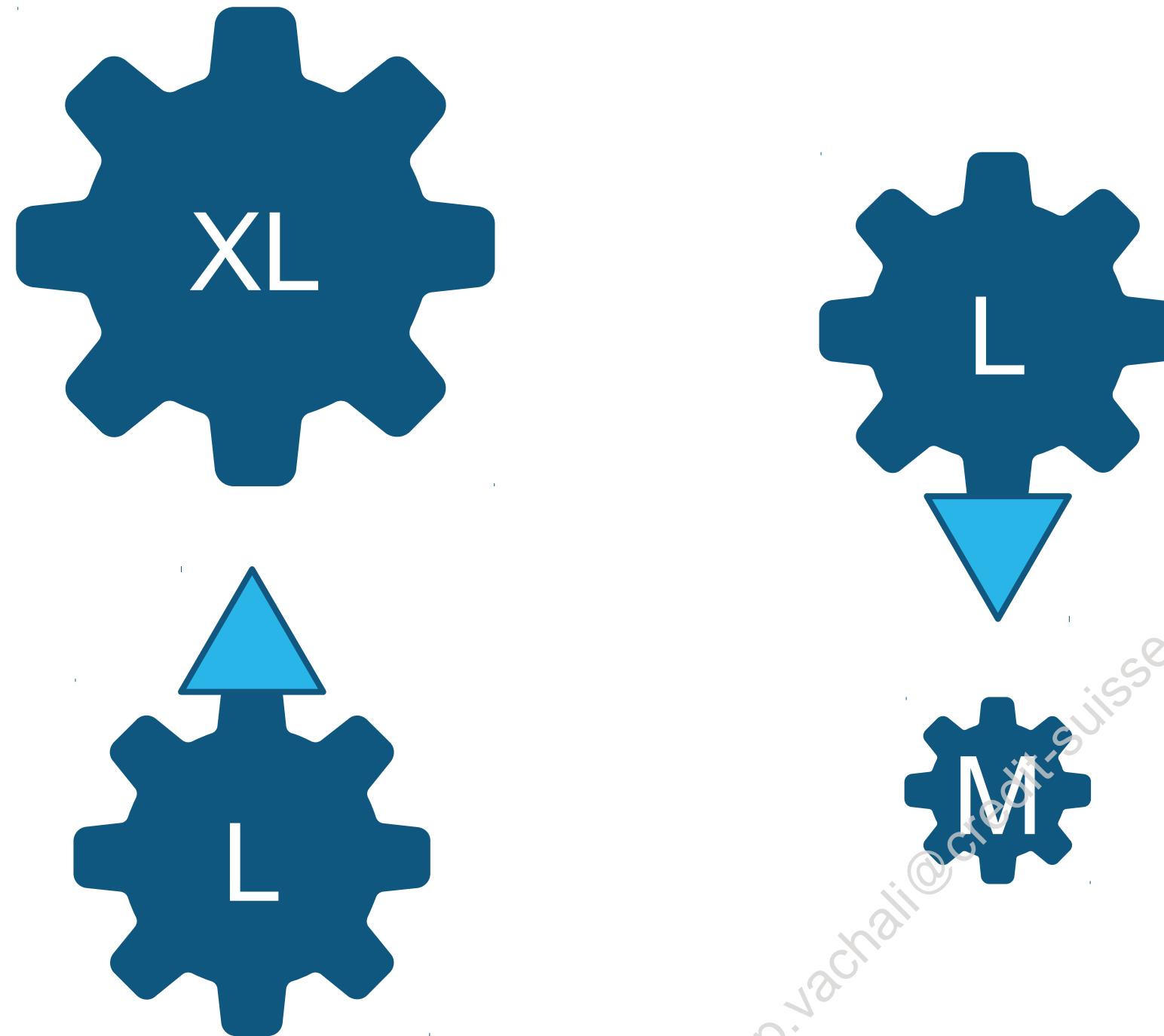
ELIMINATE RESOURCE CONTENTION



- 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



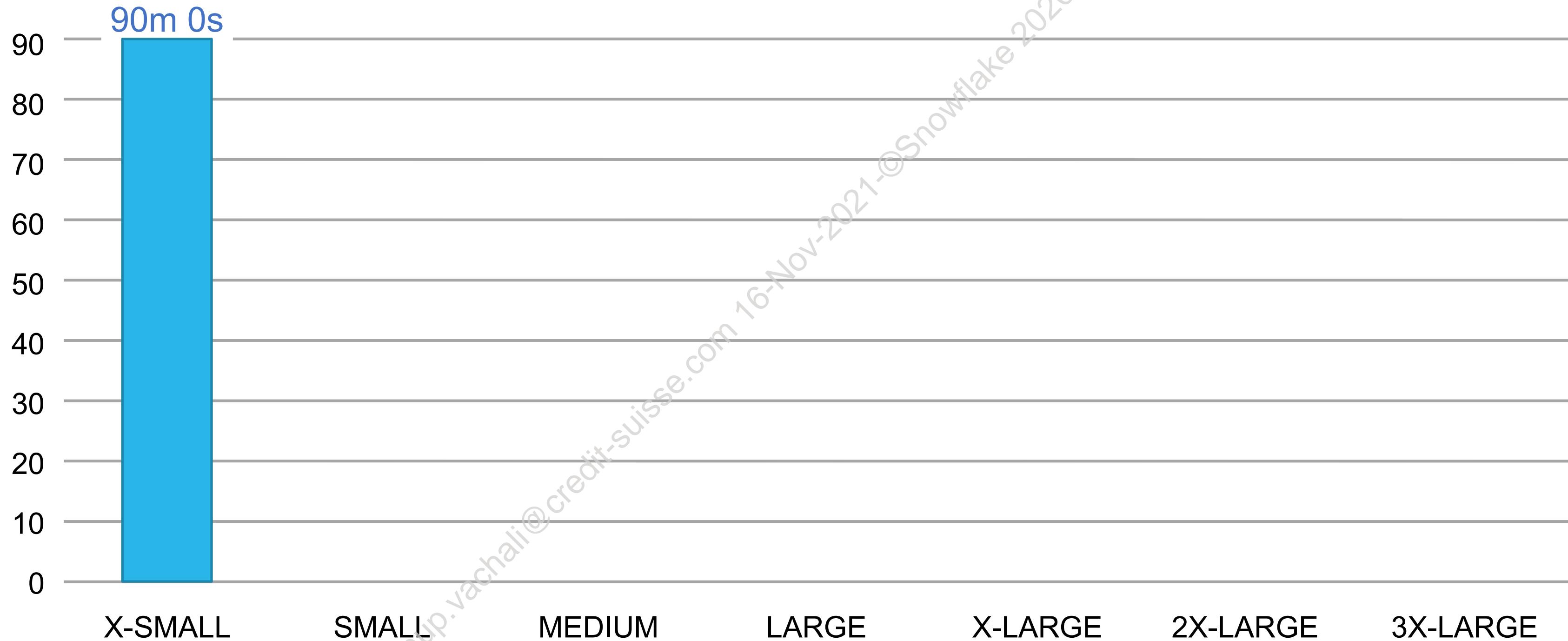
SCALE UP (OR DOWN)



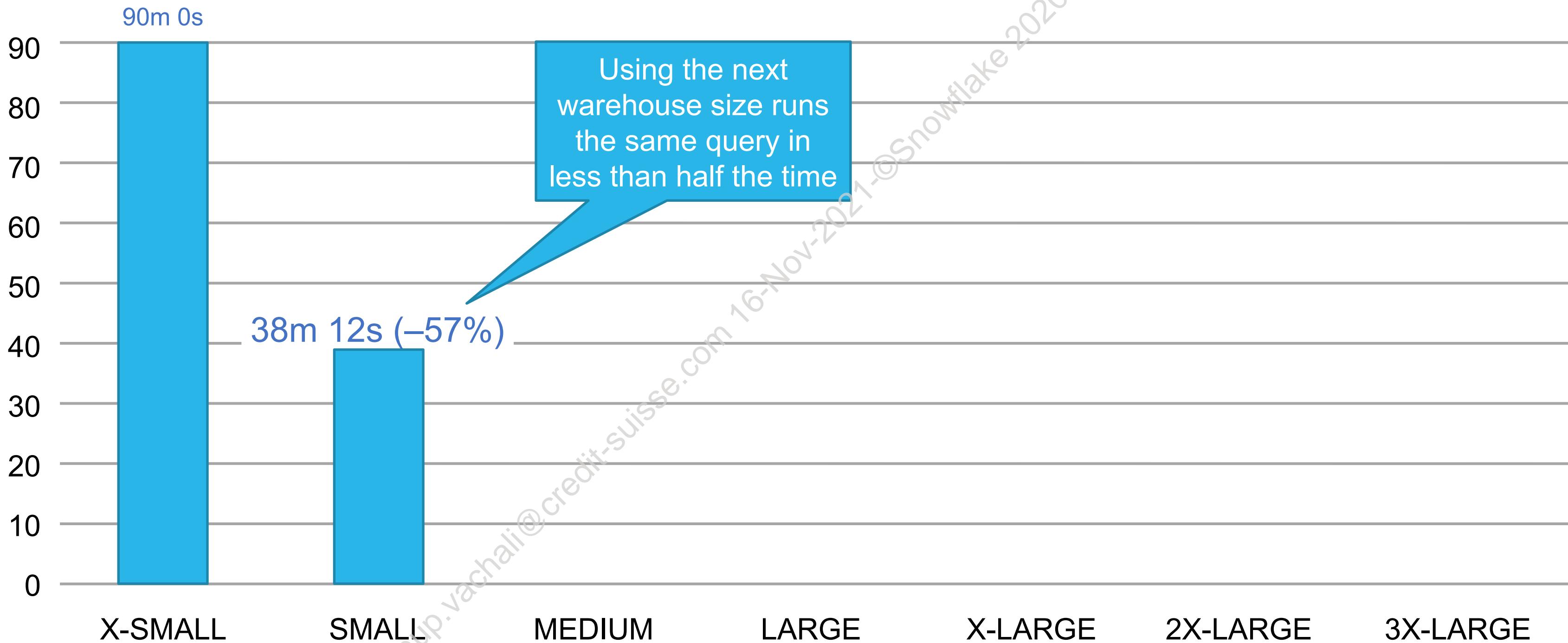
- Scale up to increase performance
- Make sure the warehouse is the right size for the workload
 - Each size up doubles the resources (and doubles the cost per second)
- Scaling is generally linear, to a point
 - Smaller warehouses are not necessarily more cost-efficient
- Find the sweet spot



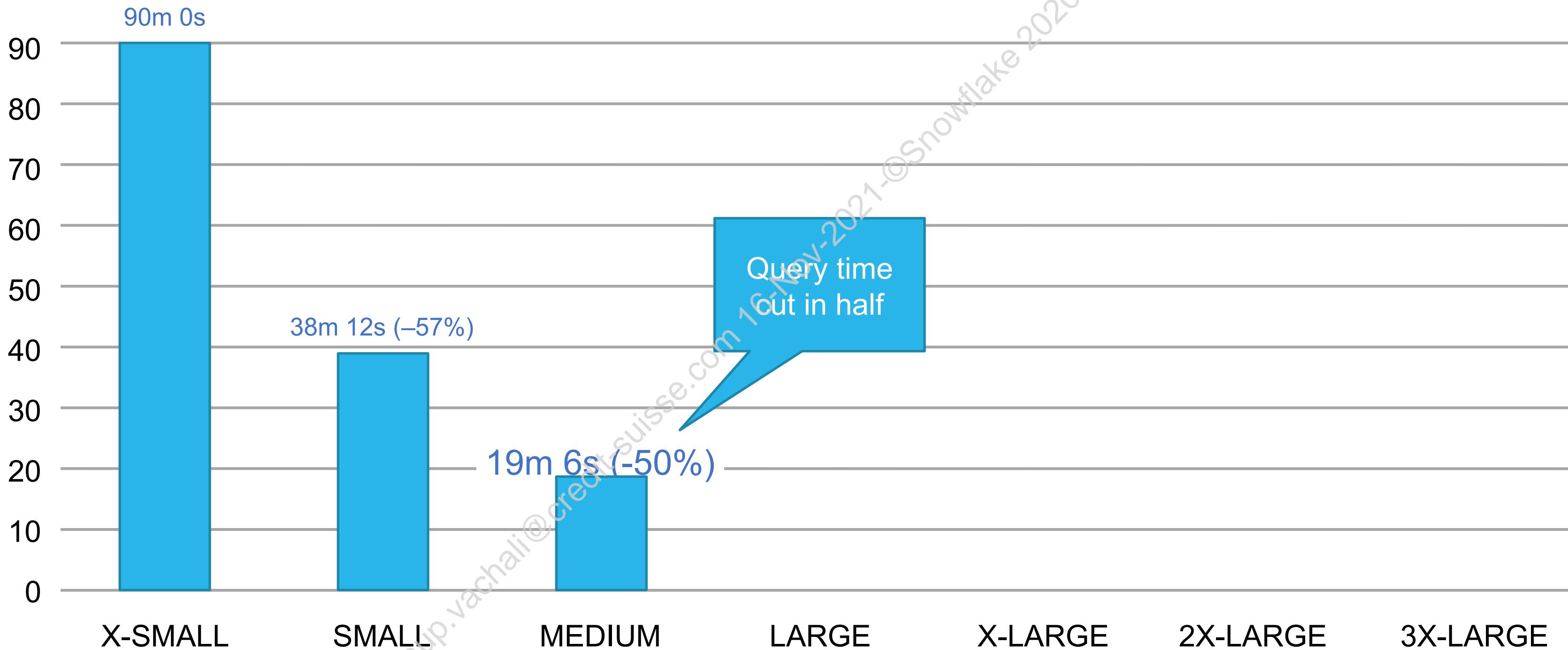
QUERY TIME BY WAREHOUSE SIZE



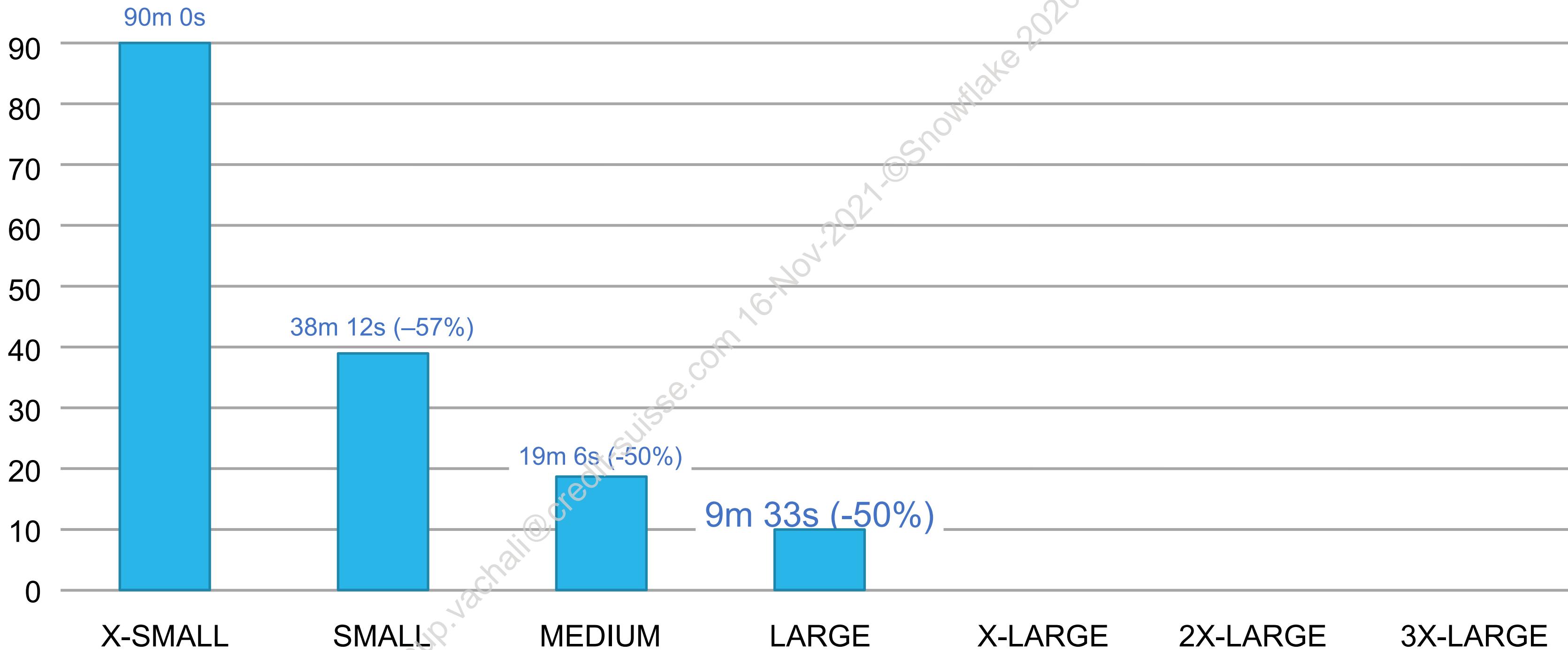
QUERY TIME BY WAREHOUSE SIZE



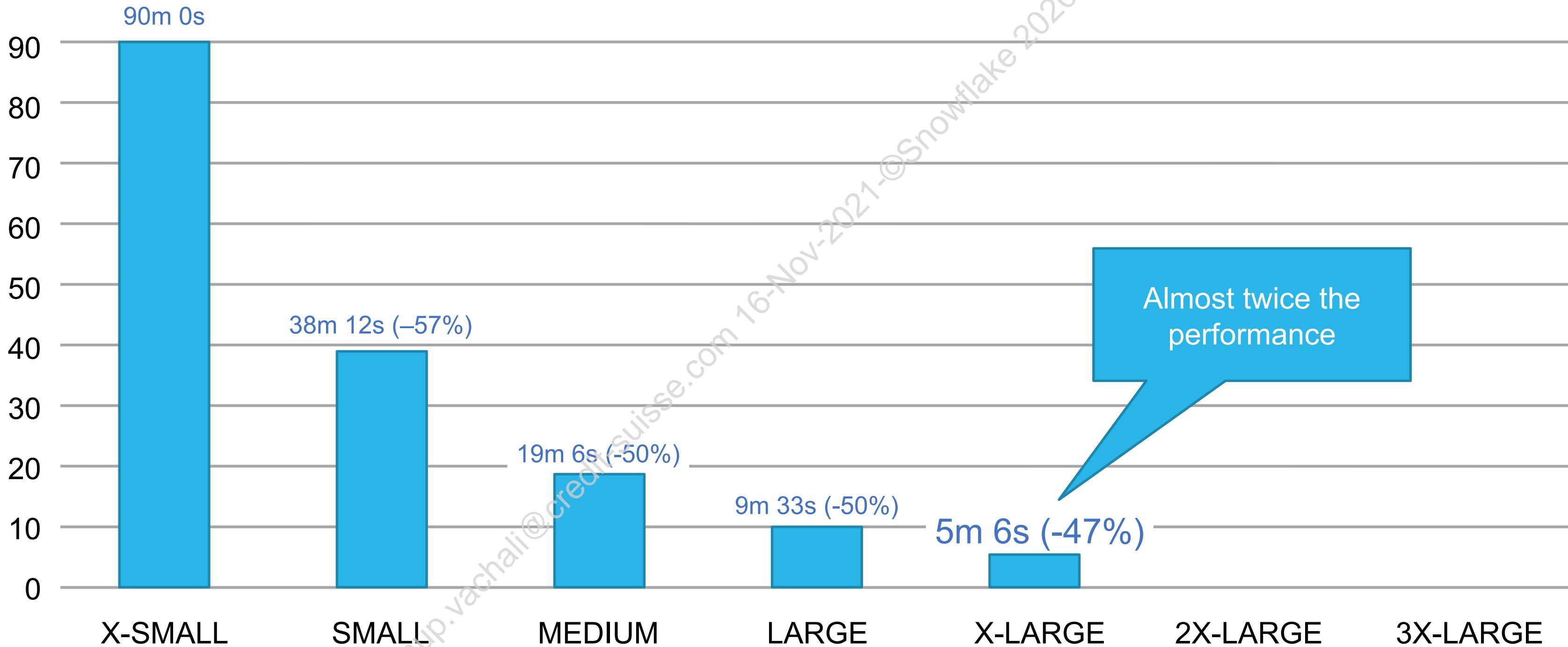
QUERY TIME BY WAREHOUSE SIZE



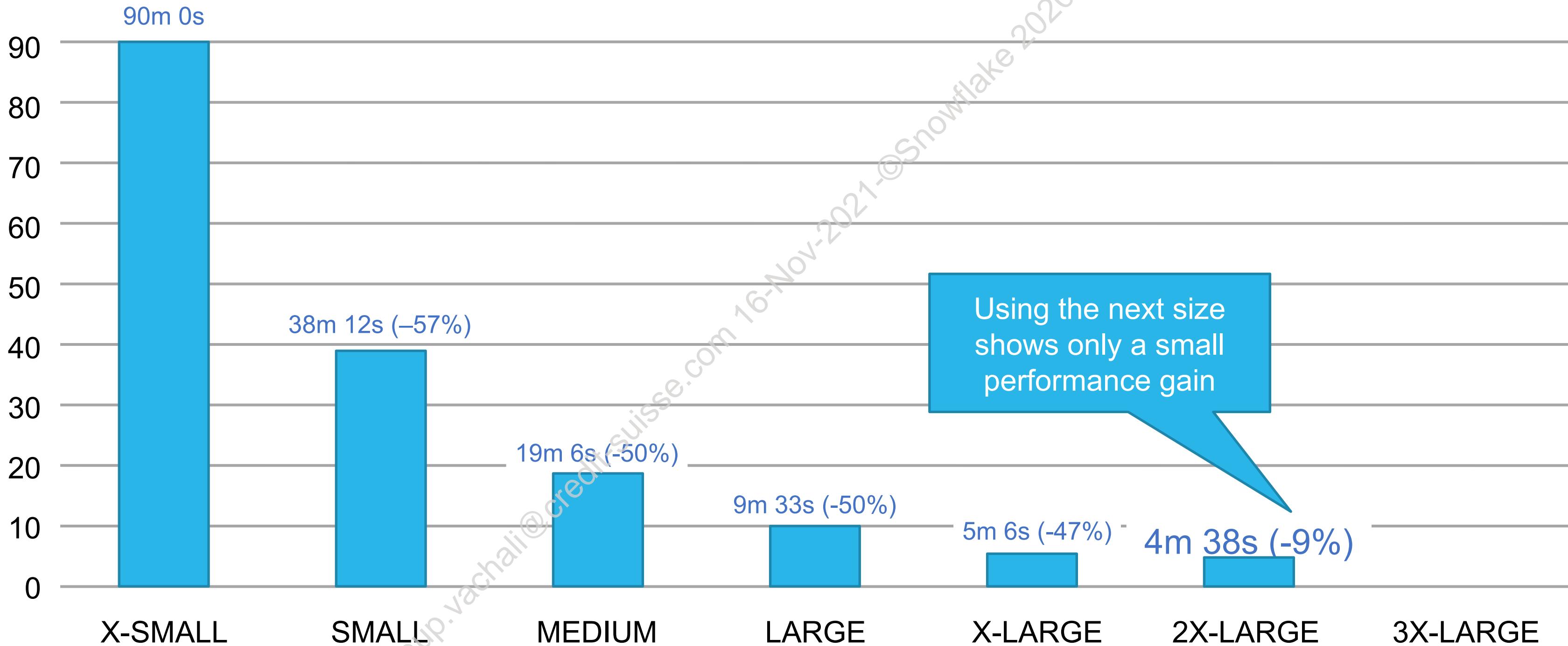
QUERY TIME BY WAREHOUSE SIZE



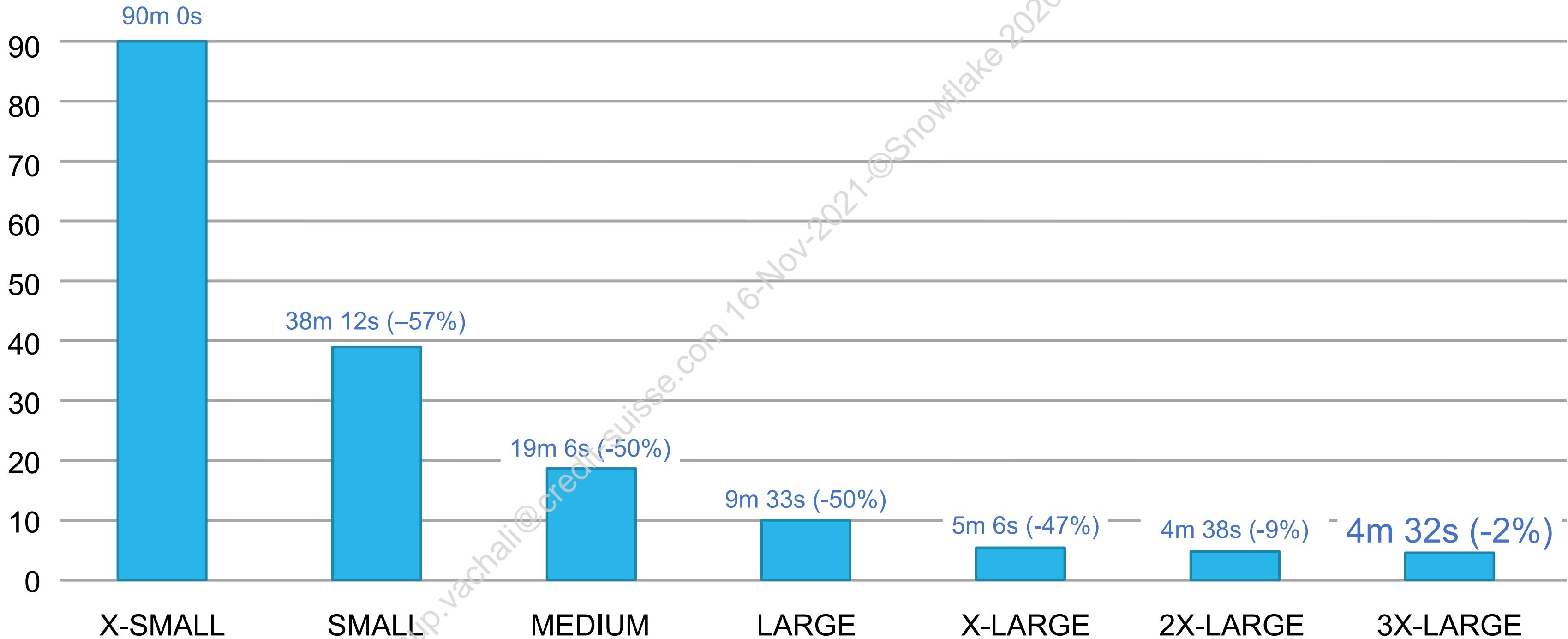
QUERY TIME BY WAREHOUSE SIZE



QUERY TIME BY WAREHOUSE SIZE

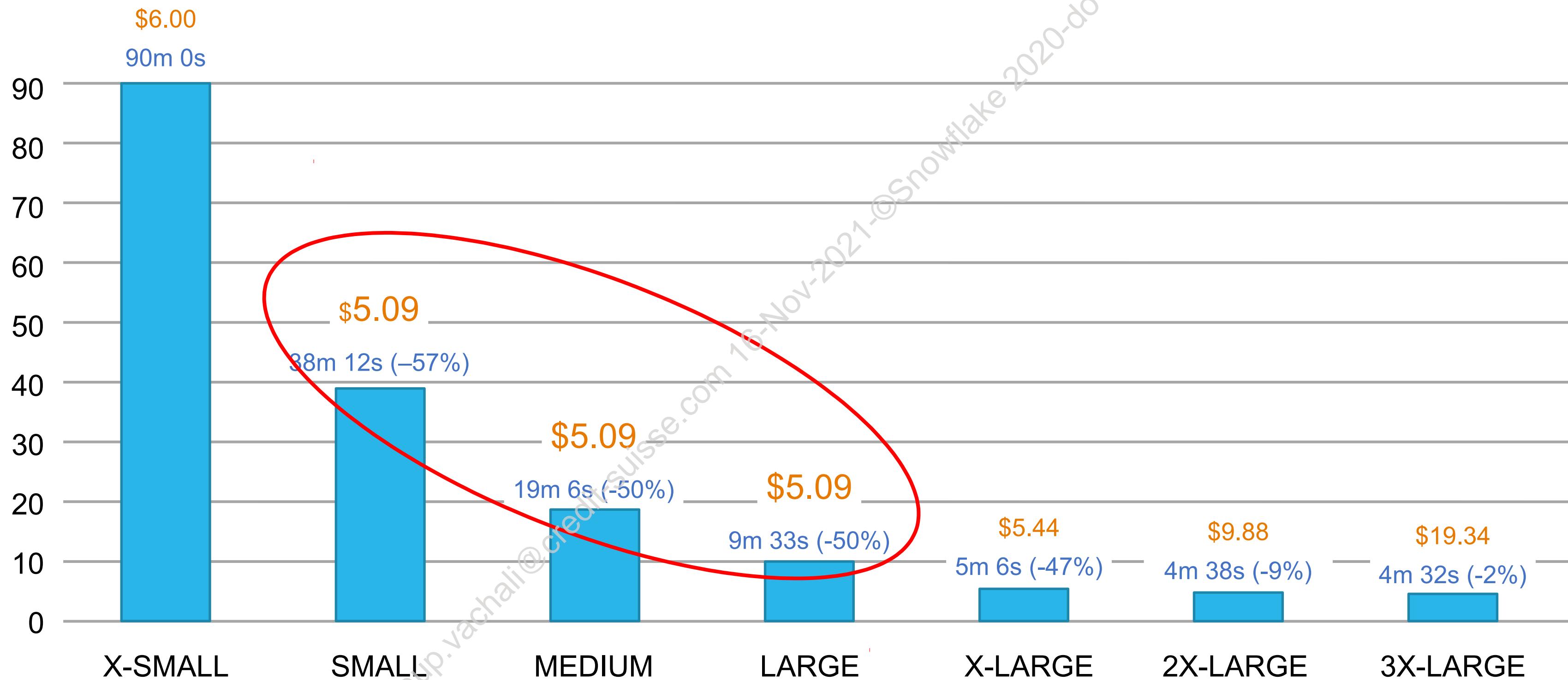


QUERY TIME BY WAREHOUSE SIZE



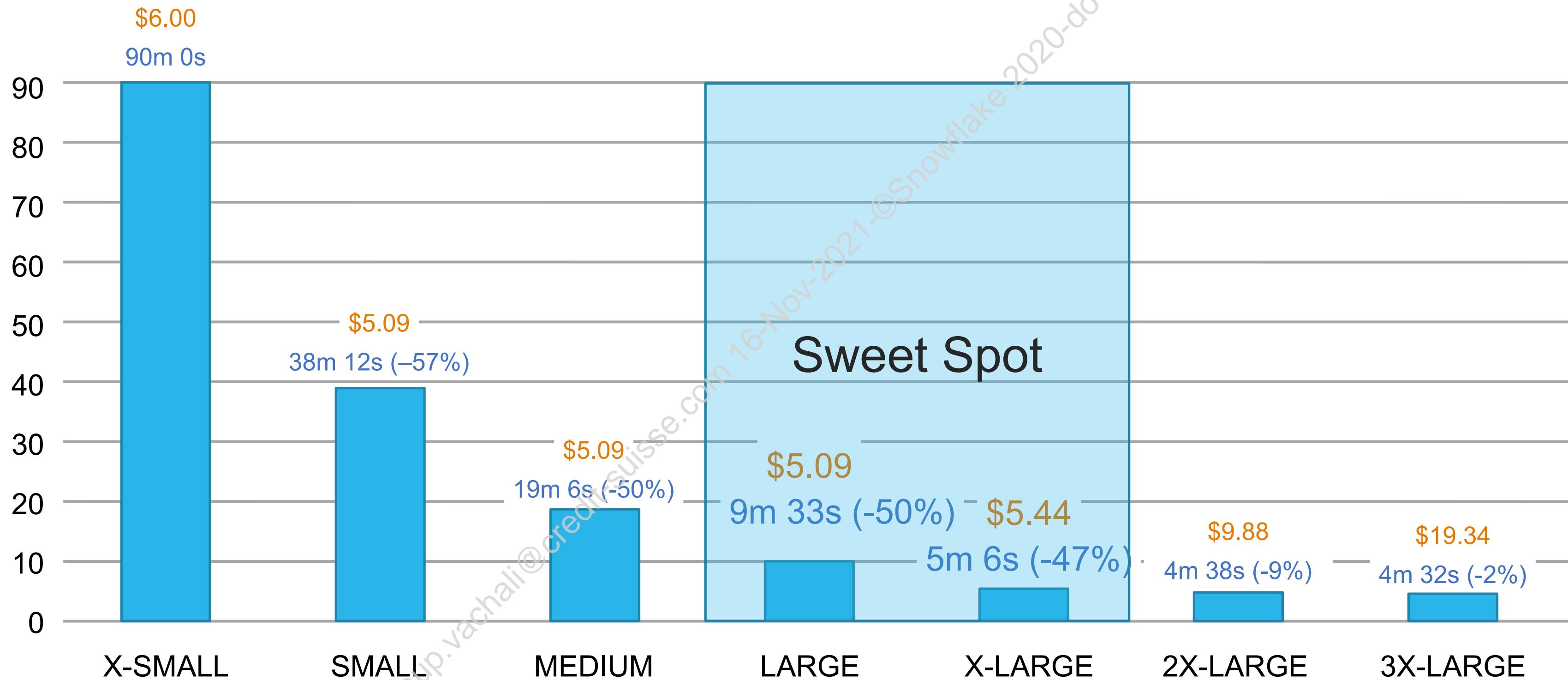
COST PER QUERY

CALCULATED USING \$4.00 PER CREDIT



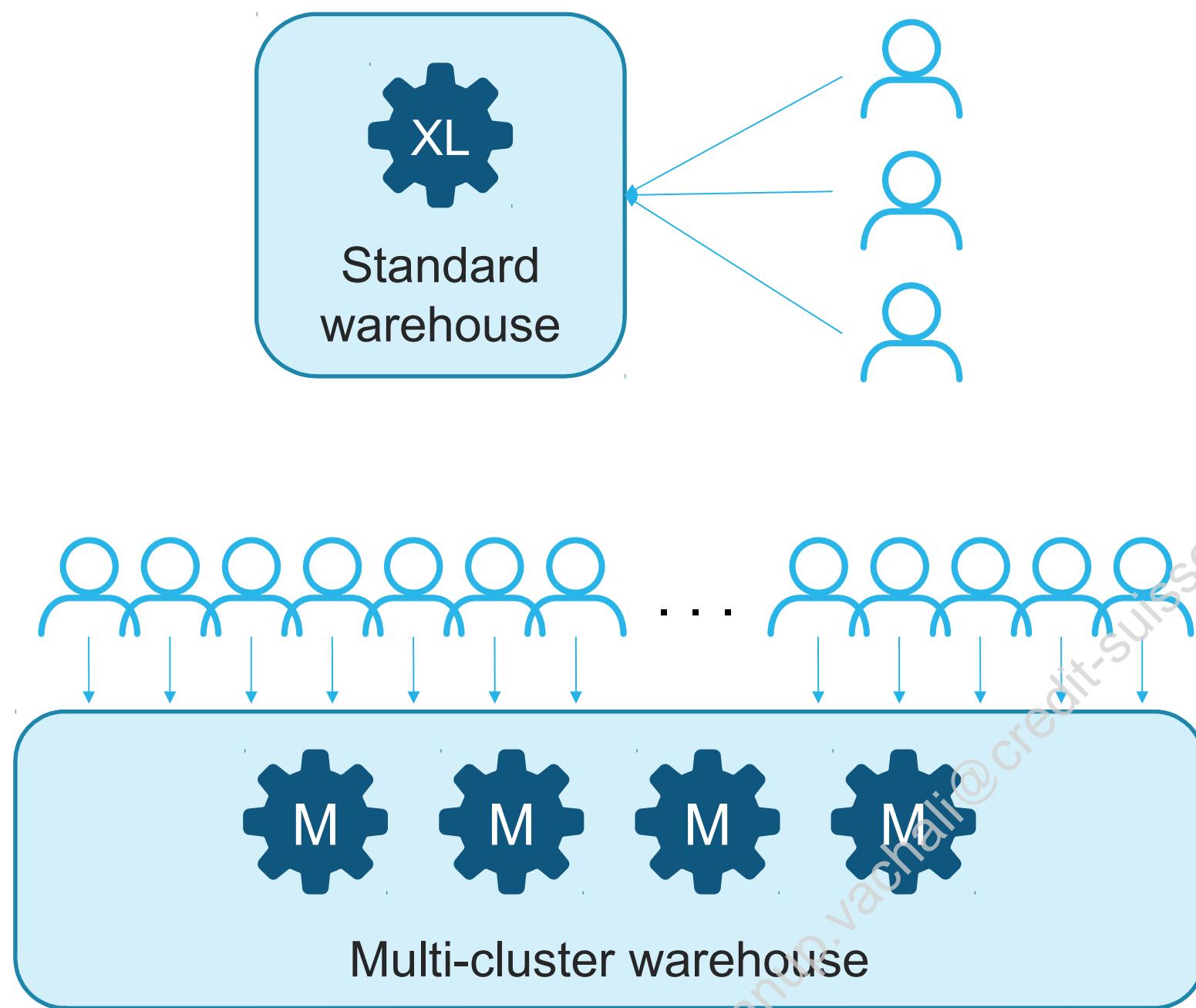
COST PER QUERY

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



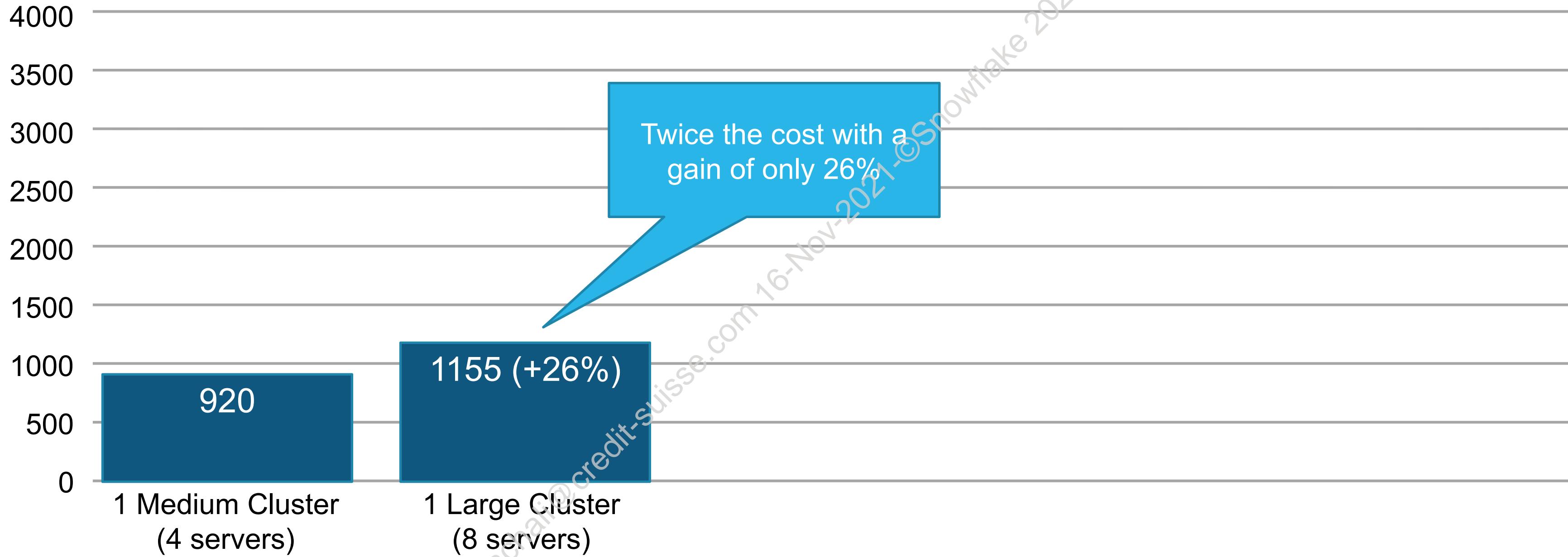
SCALE UP VS SCALE OUT

RECORDS PROCESSED BY 50 CONCURRENT USERS



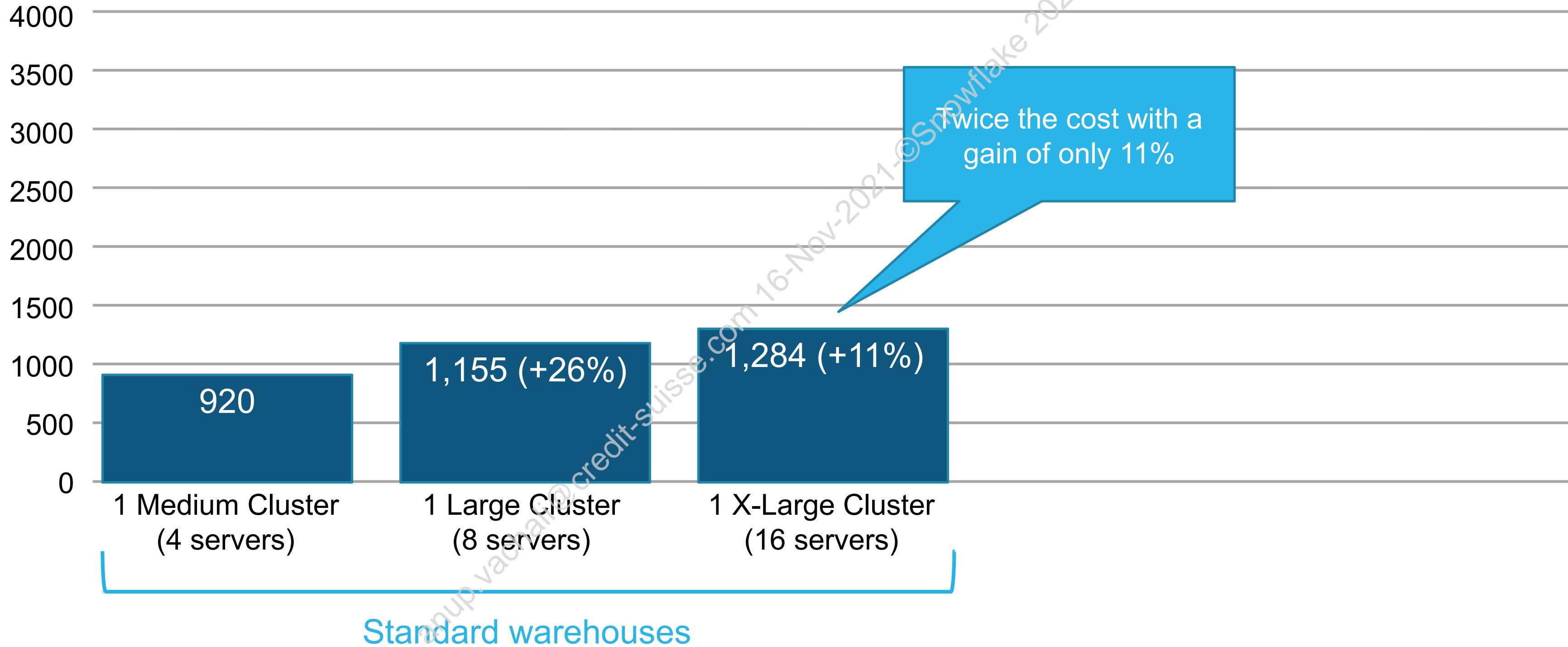
SCALE UP VS SCALE OUT

RECORDS PROCESSED BY 50 CONCURRENT USERS



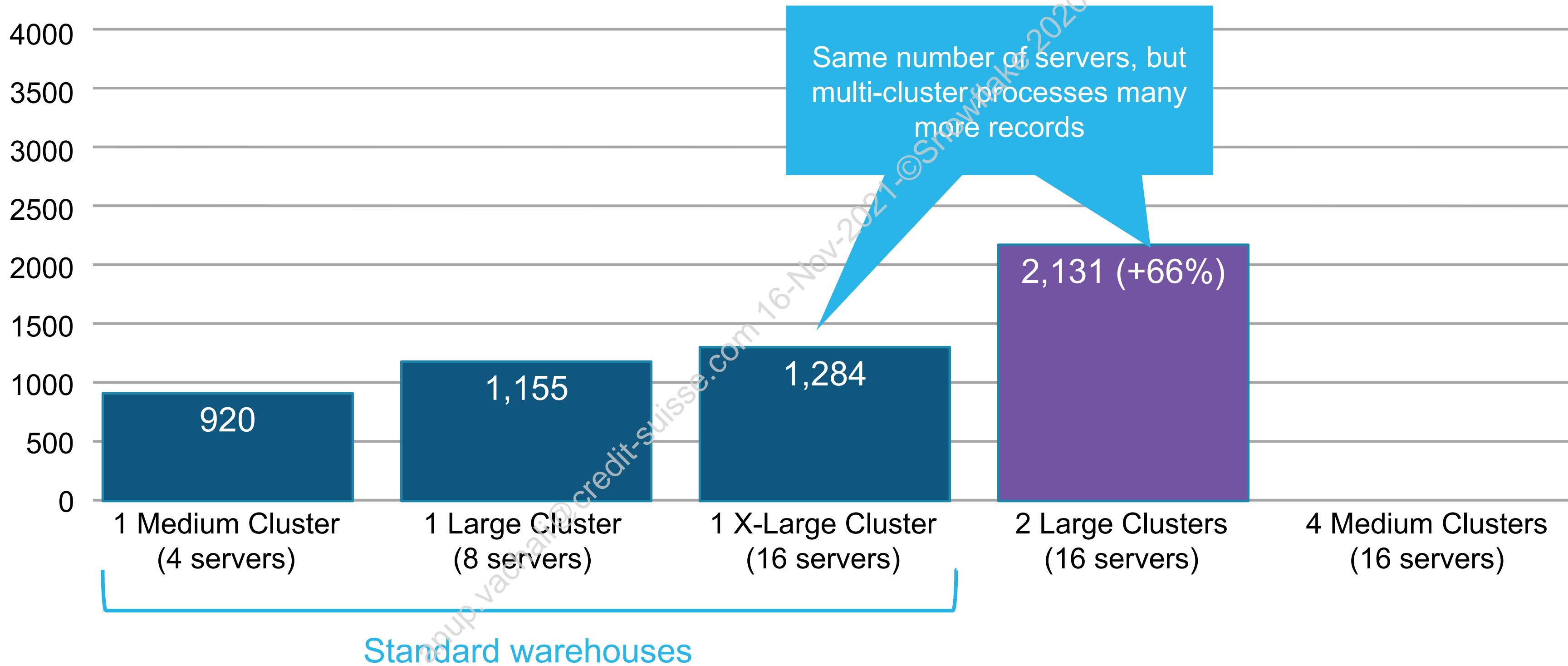
SCALE UP VS SCALE OUT

RECORDS PROCESSED BY 50 CONCURRENT USERS



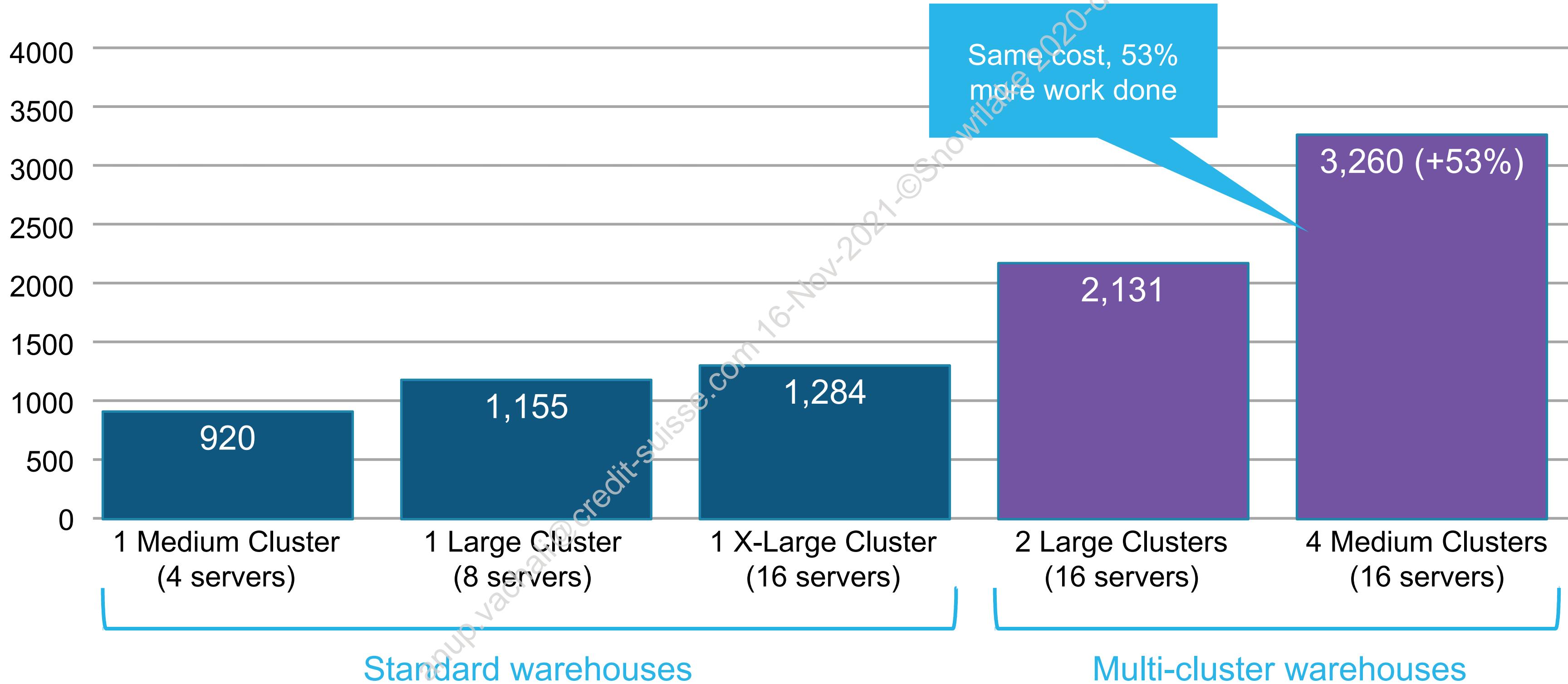
SCALE UP VS SCALE OUT

RECORDS PROCESSED BY 50 CONCURRENT USERS



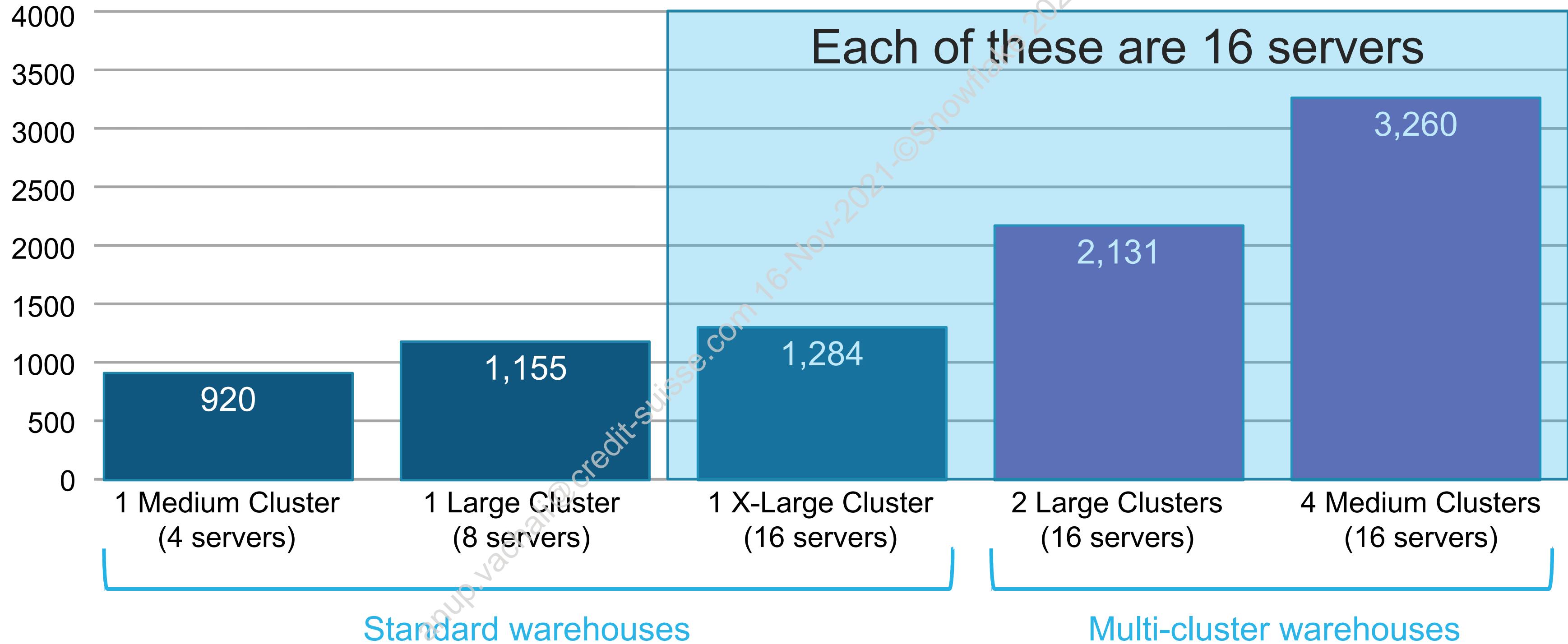
SCALE UP VS SCALE OUT

RECORDS PROCESSED BY 50 CONCURRENT USERS



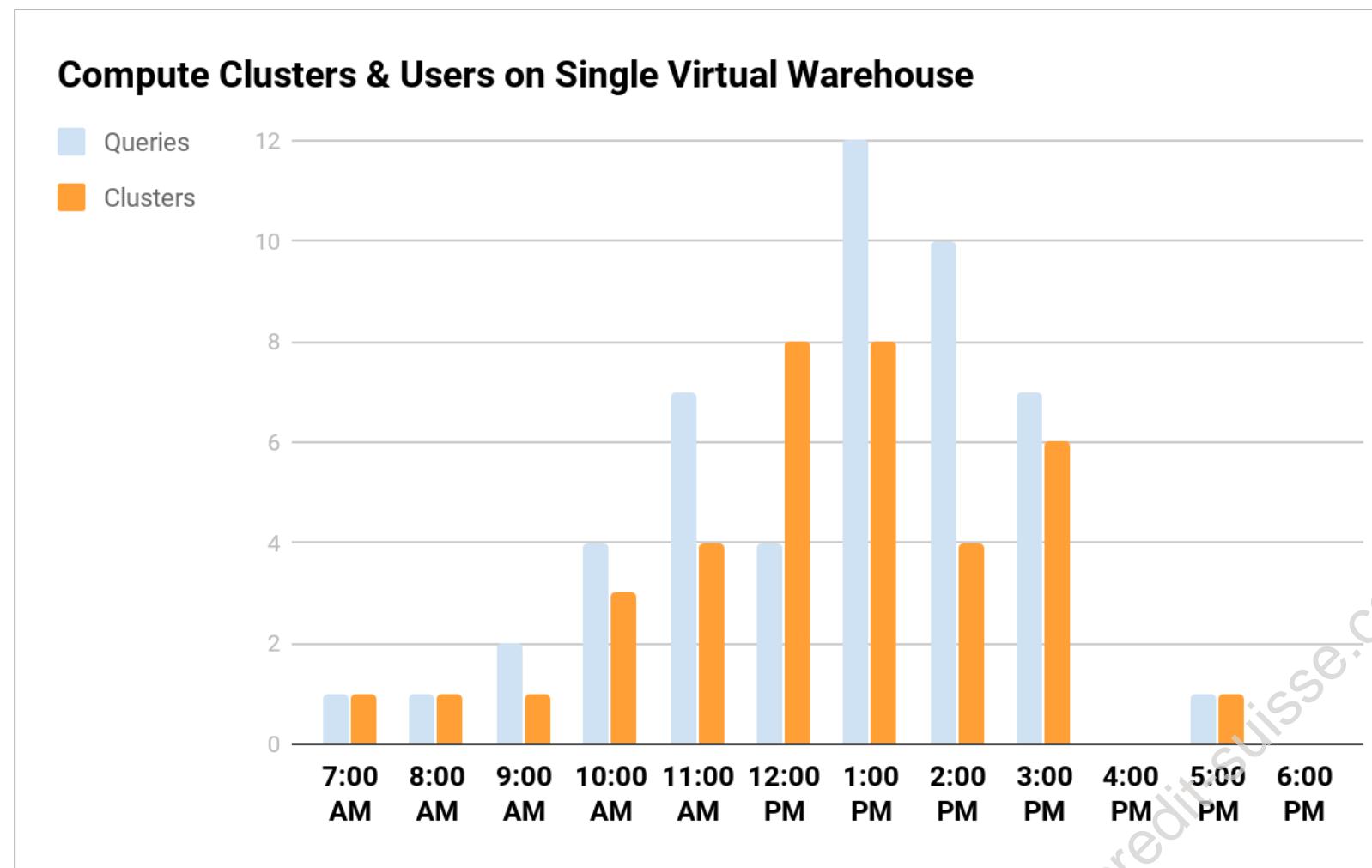
SCALE UP VS SCALE OUT

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



LAB EXERCISE: 13

Three Vectors of Scaling a Virtual Warehouse

20 minutes

- Create Warehouses
- Explore AUTO-SUSPEND and AUTO-RESUME
- Sizing Warehouses Up and Down
- Scaling Warehouses Out and Back



REDUCE QUEUEING AND SPILLING



QUERY QUEUEING



Overload
the warehouse is busy



Blocked (rare)
waiting on locks to be released



Provisioning
waiting on servers



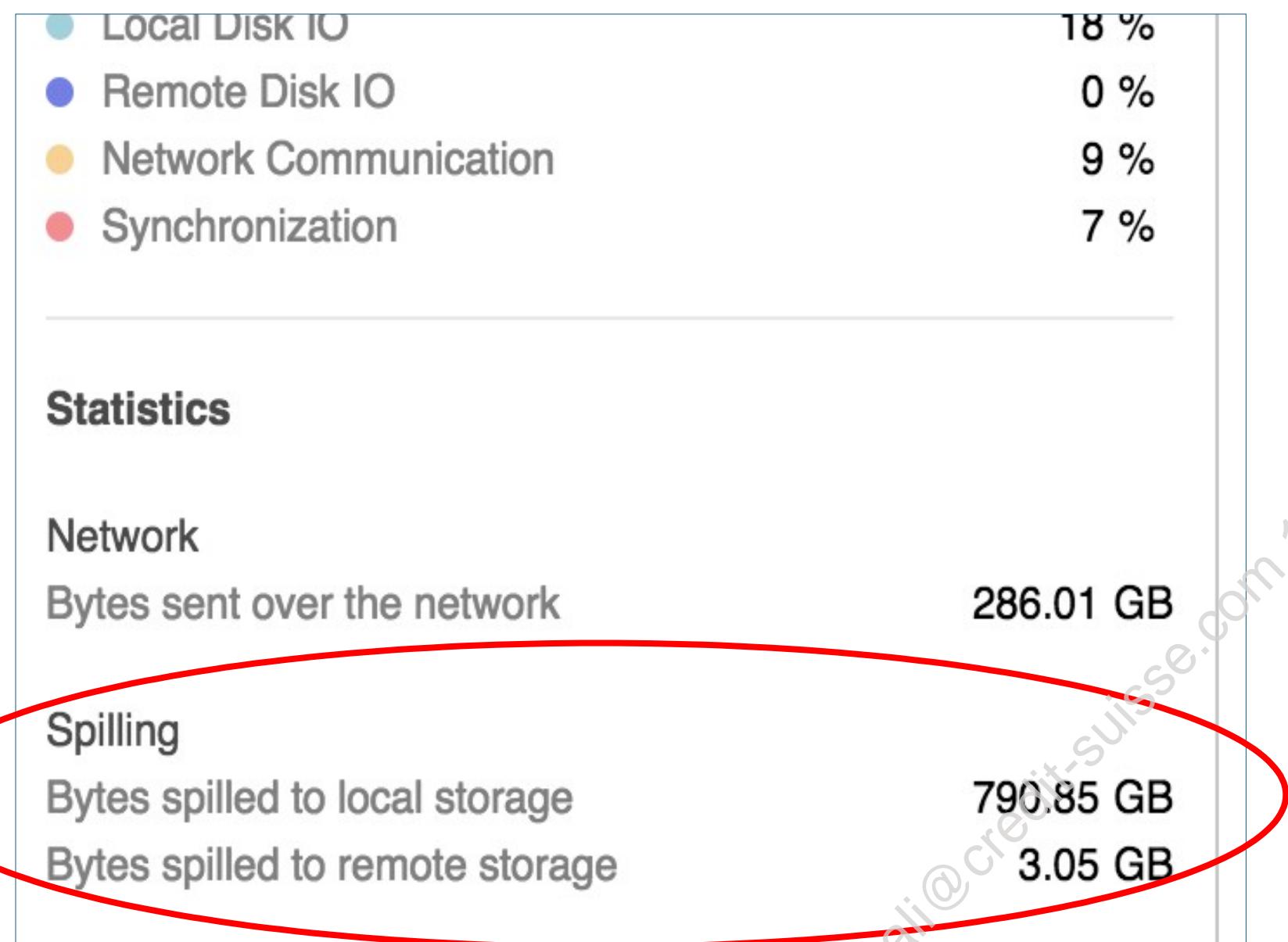
Repair time (rare)
self healing

RESOLVING QUERY QUEUEING

- Check virtual warehouse size
- For many concurrent queries, use an auto-scaling multi-cluster warehouse
- Check for long-running queries
- Check parameter values:
 - STATEMENT_TIMEOUT_IN_SECONDS
 - STATEMENT_QUEUED_TIMEOUT_IN_SECONDS



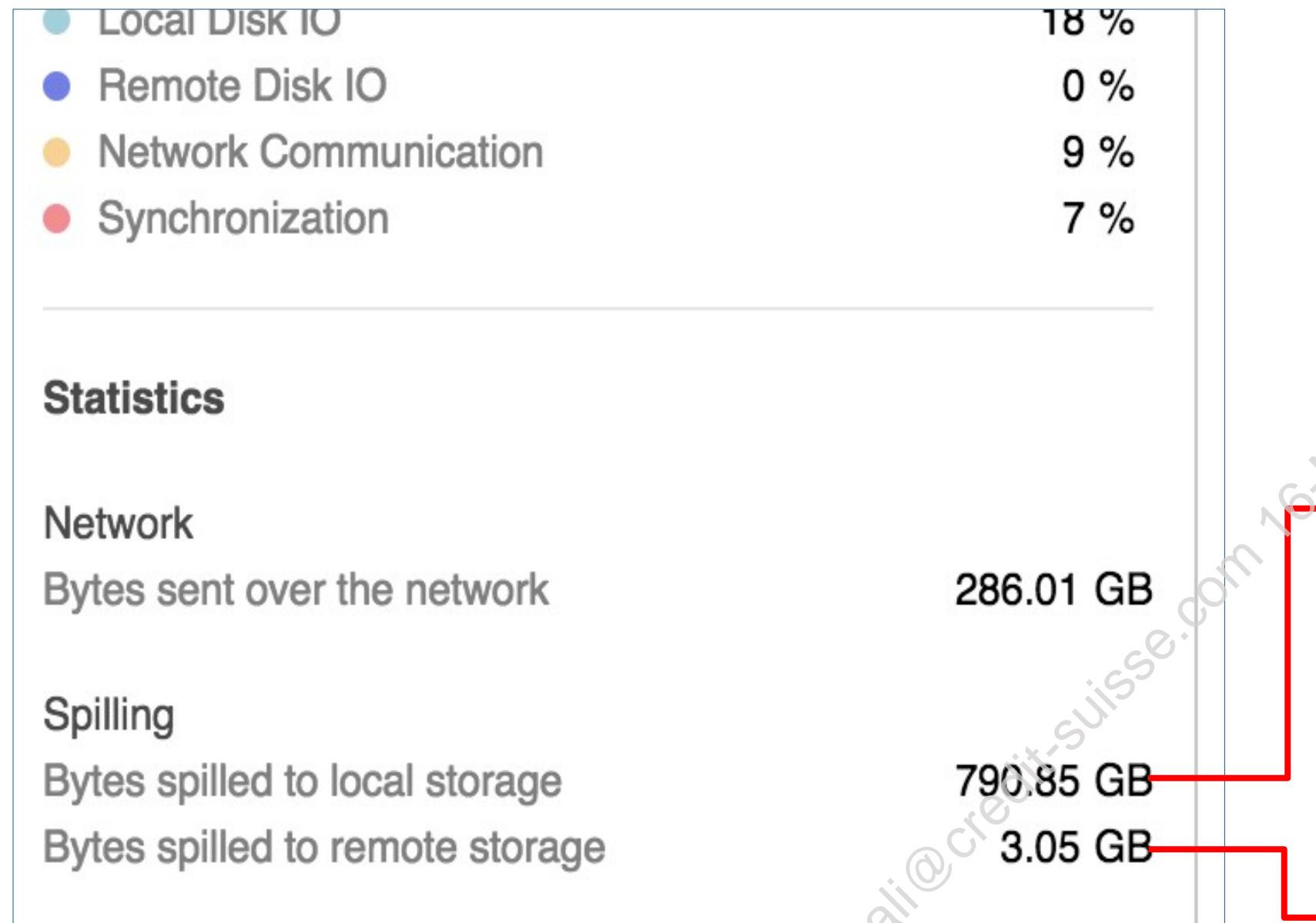
SPILLING TO DISK



- When Snowflake cannot fit an operation in memory, it starts spilling data first to disk, and then to remote storage
- Impacts query performance
- Look at the profile overview to see if a query is spilling to disk



LOCAL VS REMOTE SPILLING



- To find out more, click on a node in the Query Profile
 - Node of interest is most likely one where the query is spending significant time
- Spilled to local storage
 - Ran out of memory, spilled to local disk
 - Some performance impact
- Spilled to remote storage
 - Ran out of memory and local disk, spilled to remote storage
 - Significant performance impact



RESOLVE SPILLING

If you are spilling to local or remote storage, there are a few things you can try to address the problem

- Check ORDER BY or GROUP BY clauses
- Check your JOIN filters and WHERE clauses
- Try a larger warehouse



LAB EXERCISE: 14

Performance Analysis Toolkit and Tuning Metrics

20 minutes

- Spilling to local storage and remote storage
- Explore a Large GROUP BY
- Explore WHERE clause without clustering
- Explore WHERE clause with clustering
- Rogue Query Syndrome from Join Explosion
- Tune timeout parameter to manage rogue queries
- Using Query tags to identify and track queries for monitoring



SHARE DATA

anup.vachali@credit-suisse.com 16 Nov 2021 ©Snowflake 2020-do-not-copy



MODULE AGENDA

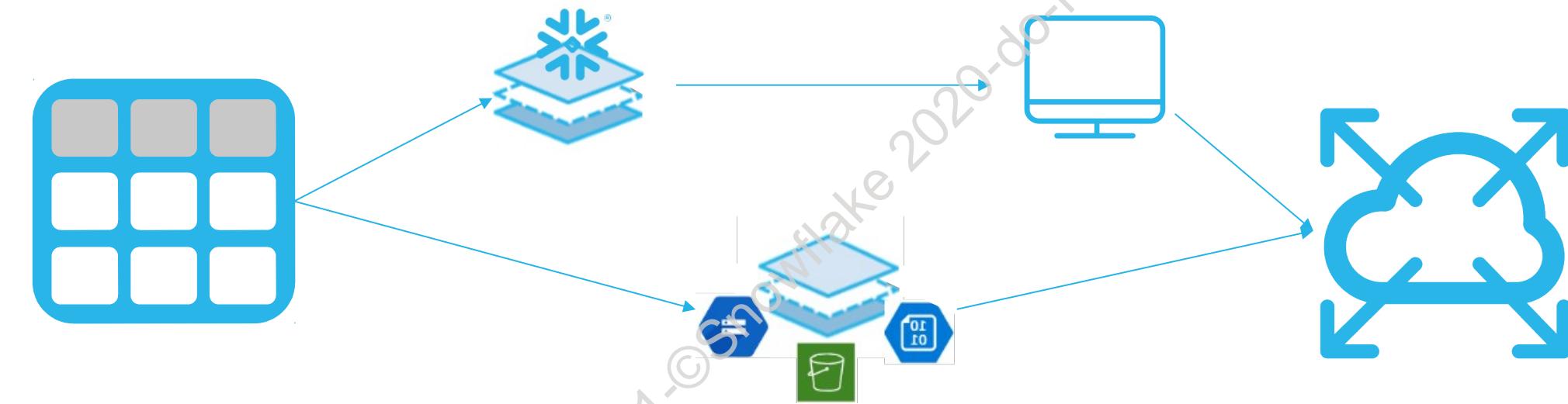
- Unload Structured Data
- Unload Options
- Unload Semi-Structured Data
- Share Data

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy

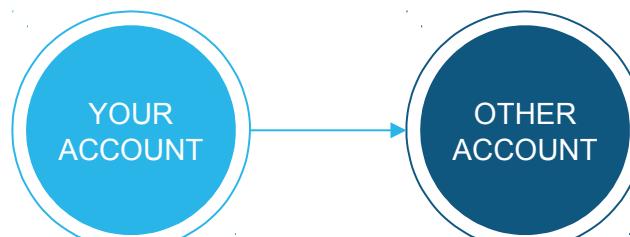


WAYS TO SHARE DATA

**UNLOAD AND SHARE
("THE HARD WAY")**

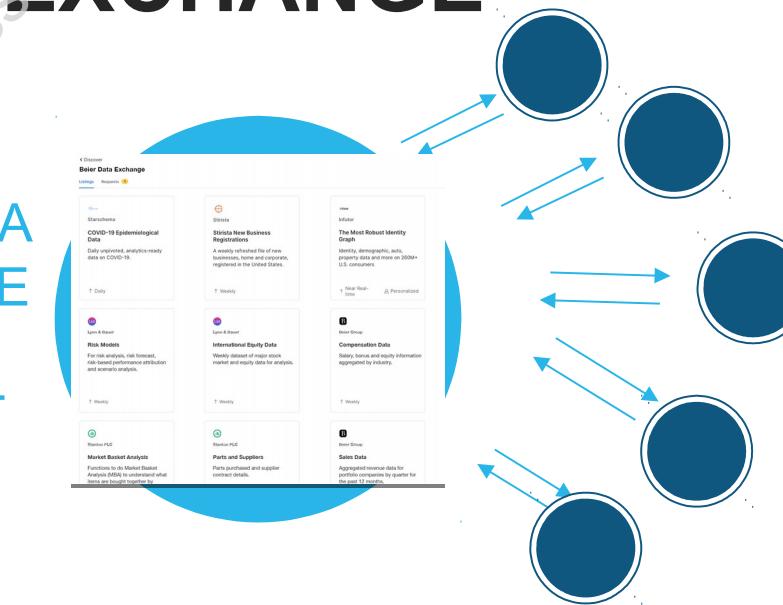


**SECURE DIRECT
DATA SHARING**

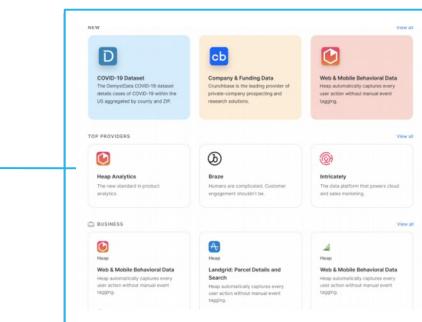


**SNOWFLAKE DATA
EXCHANGE**

**YOUR DATA
EXCHANGE
IN YOUR
ACCOUNT**



**SNOWFLAKE DATA
MARKETPLACE**



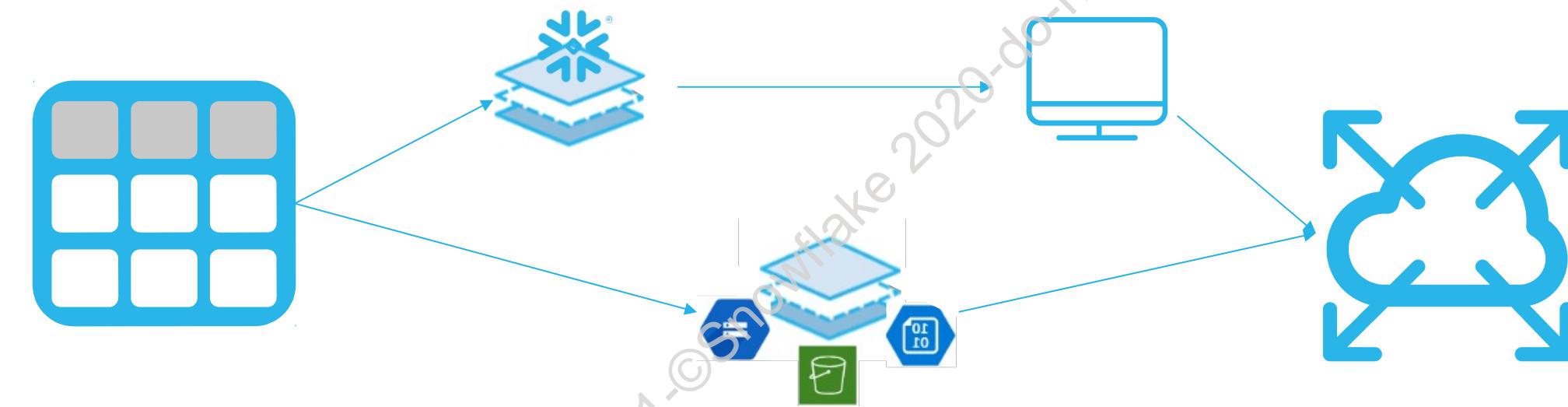
UNLOAD STRUCTURED DATA

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy

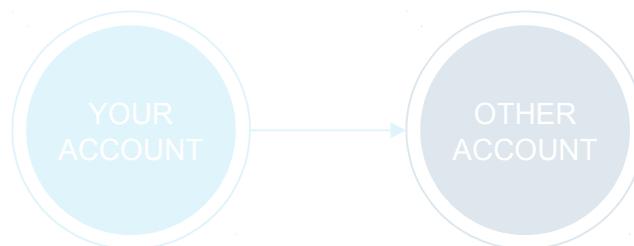


WAYS TO SHARE DATA

**UNLOAD AND SHARE
("THE HARD WAY")**



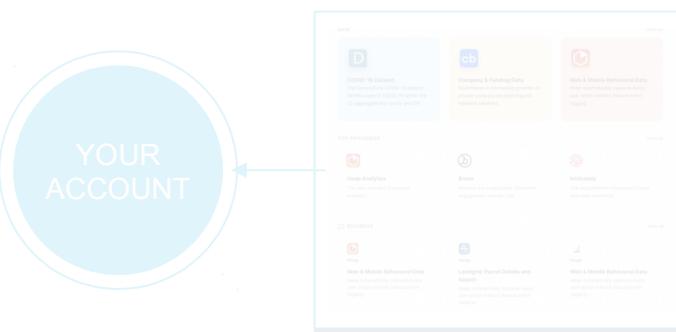
**SECURE DIRECT
DATA SHARING**



**SNOWFLAKE DATA
EXCHANGE**

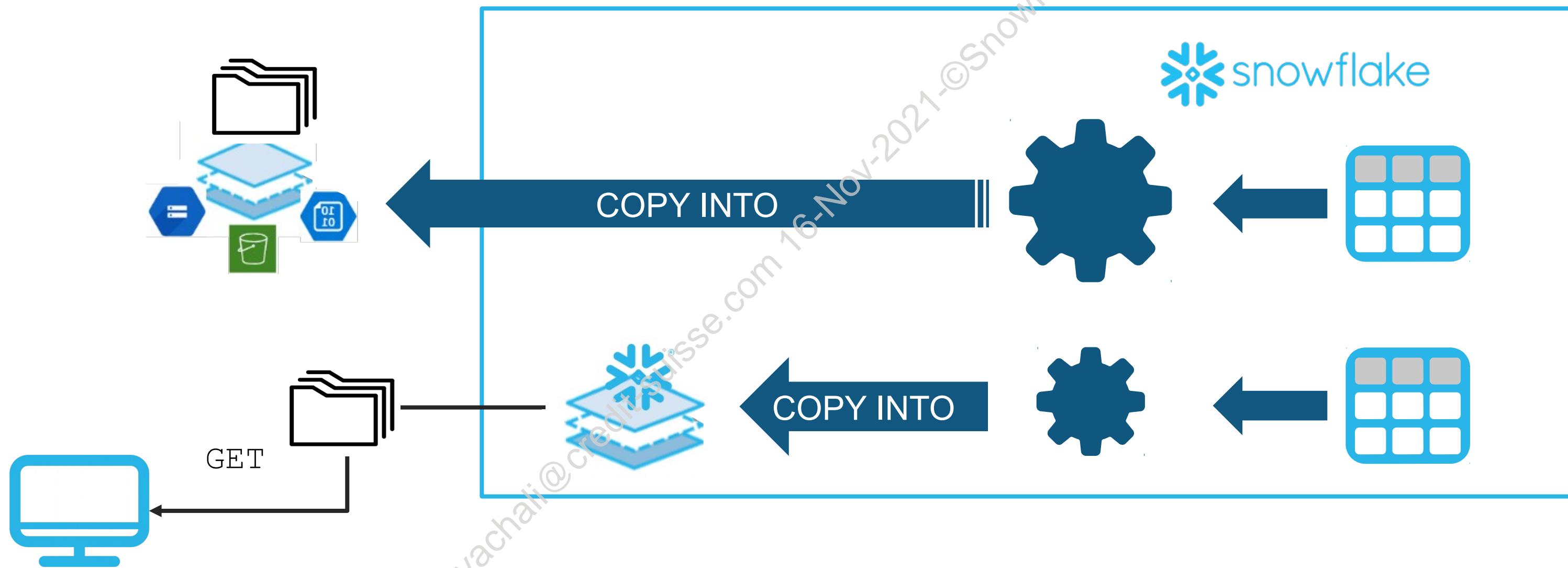


**SNOWFLAKE DATA
MARKETPLACE**



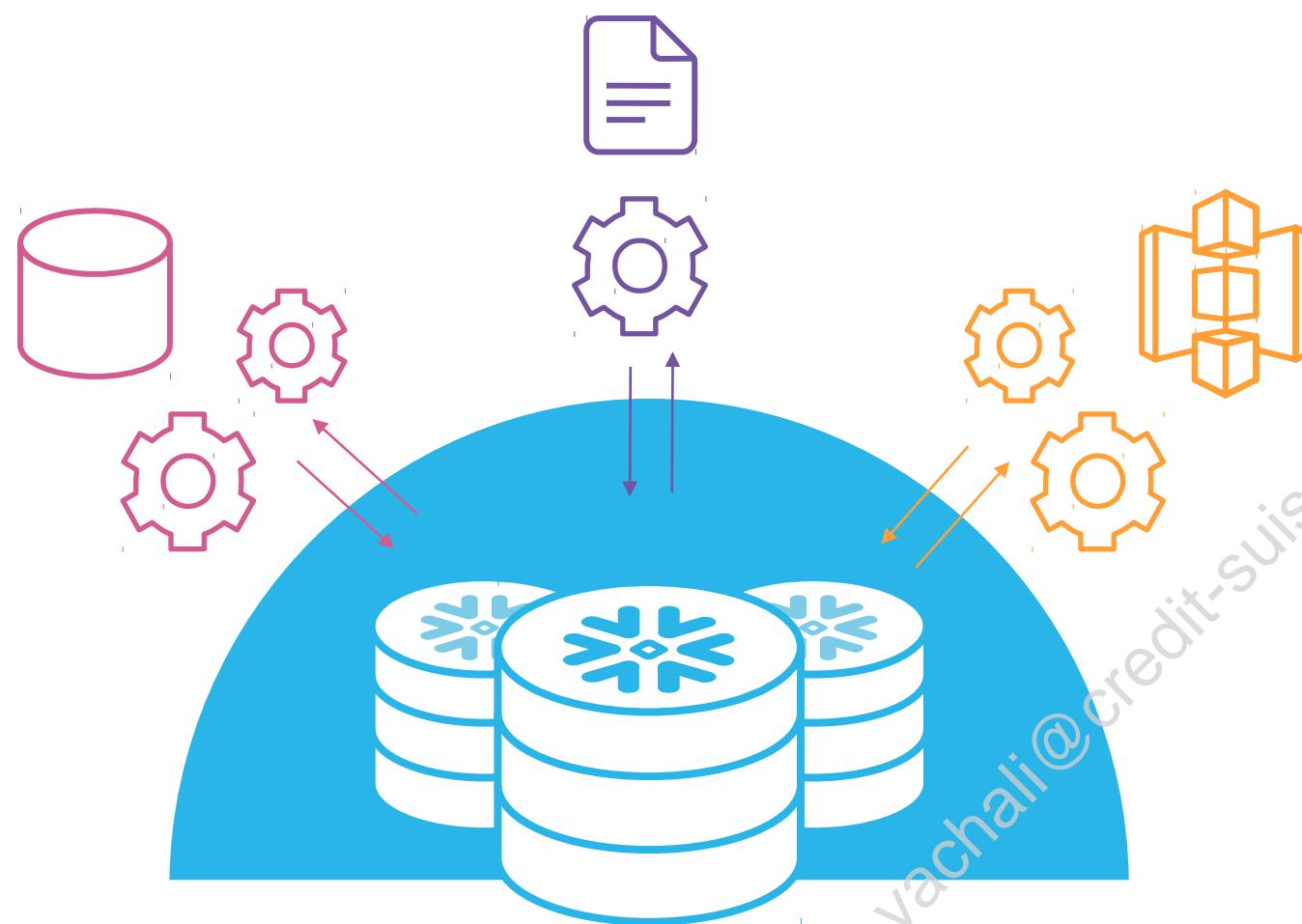
UNLOADING DATA

- The same process as loading, but in reverse



DATA UNLOADING

CONSIDERATIONS



- File formats
- File paths and names
- File splitting



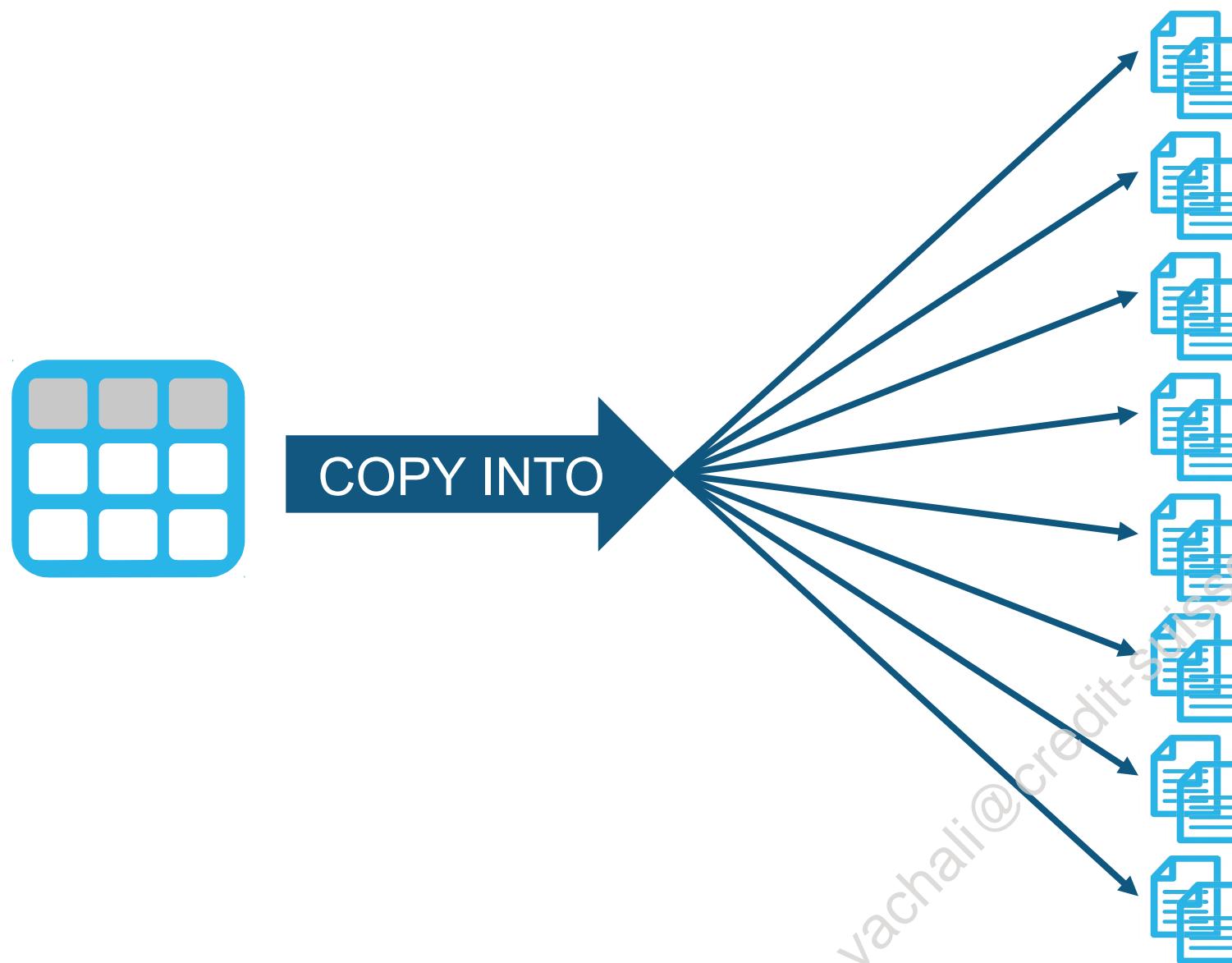
FILE FORMATS

Unload data into:

- Any flat, delimited plain text format (CSV, TSV, etc.)
- JSON
 - If unloading relational data to JSON, must convert the data using functions
- Parquet
 - Use a SELECT statement to unload a table to Parquet as multiple columns



UNLOAD AN ENTIRE TABLE



```
COPY INTO @mystage FROM mytable;
```

- Default file format is gzipped CSV
- Can be changed with file formats
- By default, creates multiple output files to increase performance



UNLOAD WITH SELECT

- SELECT queries in COPY statements support the full syntax of Snowflake SQL queries

```
COPY INTO @my_stage  
FROM (SELECT column1, column2 FROM my_table)  
FILE_FORMAT = (FORMAT_NAME = 'my_format');
```

- JOIN clauses enable unloading data from multiple tables

```
COPY INTO @my_stage  
FROM (SELECT name, id1  
      FROM my_table1  
      JOIN my_table2 ON id1 = id2)  
FILE_FORMAT = (FORMAT_NAME = 'my_format');
```



FILE PATHS AND NAMES

- To set the file path for the files, add the path after the stage name
 - In this example, the files will be copied into the folder “CSVFiles”.

```
COPY INTO @mystage/CSVFiles/ FROM mytable
```

- Default files names have the form data_<x>_<y>_<z>
- To change the file name prefix for the files, add the desired prefix after the folder name

```
COPY INTO @mystage/CSVFiles/Cust FROM mytable
```



FILE PATHS AND NAMES

- To set the file path for the files, add the path after INTO
 - In this example, the files will be copied into the folder CSVFiles

```
COPY INTO @mystage/CSVFiles/ FROM mytable
```

Terminating '/'
defines this as a
folder name

- Default files names have the form data_<x>_<y>_<z>

- To change the file name prefix for the files, add the desired prefix after the folder name

```
COPY INTO @mystage/CSVFiles/Cust FROM mytable
```

No terminating '/'
defines this as a
file name prefix

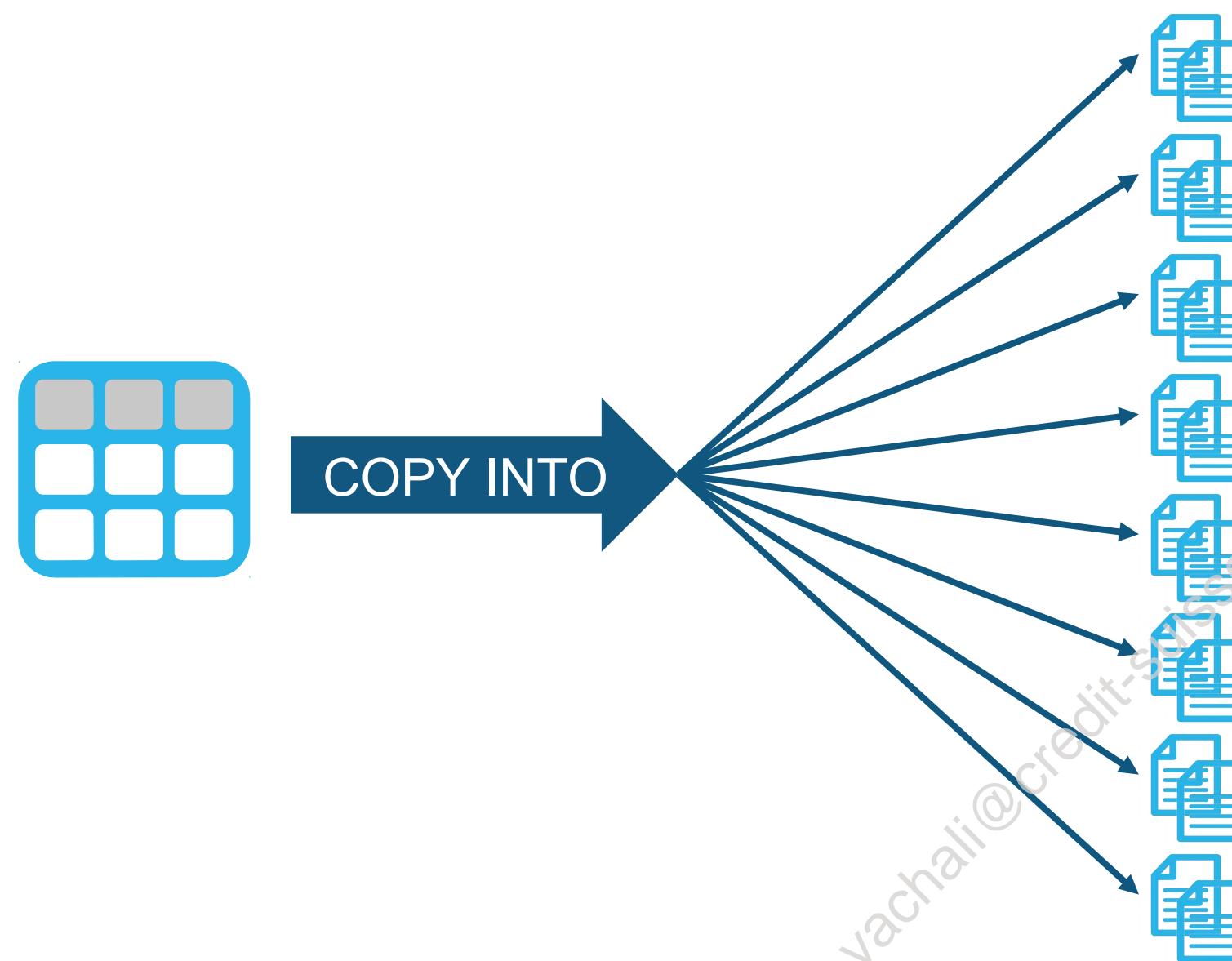


UNLOAD OPTIONS

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



FILE SPLITTING



- By default, multiple files will be created
 - At least one for each thread used to create files
- Default files size is about 16 MB (compressed)
- Use the options `SINGLE` and `MAX_FILE_SIZE` to change default behavior



MAX_FILE_SIZE

```
COPY INTO @mystage/ FROM mytable  
MAX_FILE_SIZE=2000000000
```

- Set `MAX_FILE_SIZE` to create files larger or smaller than the default (about 16 MB)
 - Maximum size that can be set is 5 GB
- The unload will create multiple files, no larger than the `MAX_FILE_SIZE` specified
- Several factors will determine the actual file size; the files will not all be the same size



SINGLE

```
COPY INTO @mystage/ FROM mytable  
SINGLE=TRUE MAX_FILE_SIZE=5000000000;
```

- Set **SINGLE=TRUE** to create a single output file, rather than many
 - Must combine with **MAX_FILE_SIZE** if the data unload is larger than the default (16 MB)
- This option will impact performance and compute usage
 - Only one thread of the warehouse will be used; all others will be idle
 - Use an XSmall warehouse (unless the unload is a complex query)
- If **SINGLE** is specified and the output is greater than **MAX_FILE_SIZE**, the unload will fail with an error



PERFORMANCE IMPACT OF SINGLE

```
COPY INTO @my_stage/stdunload/  
FROM tpch_sf100.orders;
```

Profile Overview Finished		
Total Execution Time (1m 43.72s)		
Total Statistics		
DML		
Number of rows unloaded	150,000,000	
Number of bytes unloaded		4.55 GB
IO		
Scan progress	100.00 %	
Bytes scanned		4.33 GB
Percentage scanned from cache	0.00 %	
Pruning		
Partitions scanned	291	
Partitions total		291

```
COPY INTO @mystage/singleunload/  
FROM tpch_sf100.orders  
SINGLE=TRUE MAX_FILE_SIZE=5000000000;
```

Profile Overview Finished		
Total Execution Time (8m 45.564s)		
Total Statistics		
DML		
Number of rows unloaded	150,000,000	
Number of bytes unloaded		4.60 GB
IO		
Scan progress	100.00 %	
Bytes scanned		4.33 GB
Percentage scanned from cache	0.00 %	
Pruning		
Partitions scanned	291	
Partitions total		291
Spilling		
Bytes spilled to local storage		4.61 GB



DETAILED_OUTPUT

```
COPY INTO @mystage/ FROM mytable  
DETAILED_OUTPUT=TRUE;
```

- By default, output from a COPY INTO is a single row describing the entire unload operation
- Set DETAILED_OUTPUT to TRUE to output a row for every file:

Row	FILE_NAME	FILE_SIZE	ROW_COUNT
1	data_0_2_0.csv.gz	16804599	516357
2	data_0_2_1.csv.gz	16808470	516490
3	data_0_2_2.csv.gz	16804358	516386
4	data_0_2_3.csv.gz	16804158	516379



OVERWRITE

```
COPY INTO @mystage/ FROM mytable  
OVERWRITE=TRUE;
```

- Specifies whether the COPY INTO should overwrite existing files of the same name in the stage, if any exist

X	Query ID	SQL	170ms	
Files already existing at the unload destination: @fred/stdunload/myfiles/. Use overwrite option to force unloading.				



INCLUDE_QUERY_ID

```
COPY INTO @mystage/ FROM mytable  
INCLUDE_QUERY_ID=TRUE;
```

- Set `INCLUDE_QUERY_ID=TRUE` to have the file name include the query ID of the `COPY` statement used to produce the files
- Helps ensure that concurrent `COPY` statements do not accidentally overwrite unloaded files
- Identifies the source transaction for a file
- Not supported in conjunction with the `SINGLE=TRUE` or `OVERWRITE=TRUE` options



LIST

- List the files stored in a stage

```
LIST @%monthly_sales_agg;
```

name	size	md5	last_modified
data	144	055c5ee1d38271580a6a925ad1a69d45	Sun, 2 Dec 2018 17:48:22 GMT
data_0_0_0.csv	320	aefac11b7e3b543809988ea3f90924c5	Sun, 2 Dec 2018 18:15:06 GMT
data_0_0_0.csv.gz	144	0f01461eab63268987260ae6622c3ec6	Sun, 2 Dec 2018 18:15:19 GMT
data_0_0_0.json.gz	208	72e0f0d577b590233cda48464184204b	Sat, 1 Dec 2018 16:44:43 GMT
monthly_sales_dec_0_0_0.json.gz	208	37888eaf7698a6ac9eee178eaa509e57	Sat, 1 Dec 2018 16:47:26 GMT



REMOVE

- Removes files that have been stored in an internal stage

```
REMOVE @%monthly_sales_agg PATTERN='.*data.*';
```

name	result
data_0_0_0.json.gz	removed
data_0_0_0.csv.gz	removed
data	removed
data_0_0_0.csv	removed

```
LIST @%monthly_sales_agg;
```

name	size	md5	last_modified
monthly_sales_dec_0_0_0.json.gz	208	37888eaf7698a6ac9eee178eaa509e57	Sat, 1 Dec 2018 16:47:26 GMT



LAB EXERCISE: 15

Unload Structured Data

40 minutes

Tasks:

- Unload a pipe-delimited file
- Unload part of a table
- JOIN and unload

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



UNLOAD SEMI-STRUCTURED DATA

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



FILE FORMATS FOR UNLOAD

- JSON:
 - Data must be unloaded as a VARIANT data type
 - Must unload from a VARIANT column or build a VARIANT using a function such as `object_construct()` as part of the unload
- Parquet:
 - Use a SELECT statement to unload a table to Parquet as multiple columns



CONVERT RELATIONAL TO JSON

- Unloading structured data to JSON format requires the `object_construct()` function

```
SELECT empid, SUM(amount), month  
FROM monthly_sales  
GROUP BY empid, month;
```

EMPID	SUM(AMOUNT)	MONTH
111	10400	Jan
112	39500	Jan
111	8000	Feb
112	90700	Feb
111	11000	Mar
112	12000	Mar
111	18000	Apr
112	5300	Apr
100	10500	Jan
101	8500	Jan
101	1000	Feb

```
SELECT OBJECT_CONSTRUCT(*) FROM  
(SELECT empid, SUM(amount), month  
FROM monthly_sales  
GROUP BY empid, month);
```

OBJECT_CONSTRUCT(*)
{
"AMOUNT": 10400,
"EMPID": 111,
"MONTH": "Jan"
}
{
"AMOUNT": 39500,
"EMPID": 112,
"MONTH": "Jan"
}
{



UNLOAD RELATIONAL TO SEMI-STRUCTURED

```
COPY INTO @%monthly_sales/december_sales
FROM
  (SELECT OBJECT_CONSTRUCT(*) FROM
    (SELECT empty, SUM(amount, month
      FROM monthly_sales
      GROUP BY empid, month))
FILE_FORMAT = (TYPE = JSON);

GET @%monthly_sales file:///Users/jsmith/
```

```
{"AMOUNT":10400,"EMPID":111,"MONTH":"Jan"}
{"AMOUNT":39500,"EMPID":112,"MONTH":"Jan"}
{"AMOUNT":8000,"EMPID":111,"MONTH":"Feb"}
{"AMOUNT":90700,"EMPID":112,"MONTH":"Feb"}
{"AMOUNT":11000,"EMPID":111,"MONTH":"Mar"}
{"AMOUNT":12000,"EMPID":112,"MONTH":"Mar"}
{"AMOUNT":18000,"EMPID":111,"MONTH":"Apr"}
{"AMOUNT":5300,"EMPID":112,"MONTH":"Apr"}
{"AMOUNT":10500,"EMPID":100,"MONTH":"Jan"}
{"AMOUNT":8500,"EMPID":101,"MONTH":"Jan"}
{"AMOUNT":1000,"EMPID":101,"MONTH":"Feb"}
{"AMOUNT":200,"EMPID":113,"MONTH":"Mar"}
{"AMOUNT":1000,"EMPID":102,"MONTH":"Mar"}
{"AMOUNT":600,"EMPID":115,"MONTH":"Mar"}
{"AMOUNT":1000,"EMPID":104,"MONTH":"Feb"}
{"AMOUNT":9000,"EMPID":105,"MONTH":"Mar"}
{"AMOUNT":7500,"EMPID":105,"MONTH":"Jan"}
 {"AMOUNT":7500,"EMPID":106,"MONTH":"Mar"}
 {"AMOUNT":500,"EMPID":100,"MONTH":"Mar"}
 {"AMOUNT":5500,"EMPID":114,"MONTH":"Mar"}
 {"AMOUNT":500,"EMPID":104,"MONTH":"Jan"}
 {"AMOUNT":10500,"EMPID":106,"MONTH":"Jan"}
 {"AMOUNT":5500,"EMPID":102,"MONTH":"Jan"}
 {"AMOUNT":2500,"EMPID":103,"MONTH":"Jan"}  
monthly_sales_dec_0_0_0.json (END)
```



EXAMPLE: EXPORTING USER ACCESS LOGS

- History logs are available via functions and views
- Results can be further filtered using SQL
- Return (for example) login activity
 - Up to 7 days back with functions
 - Up to 365 days back with views
- Export through JDBC or as JSON for use in SIEM
- All supplied drivers/connectors also have extended logging

The screenshot shows the Snowflake UI with the following schema structure:

- SNOWFLAKE**: ACCOUNT_USAGE (No Tables in this Schema)
- Views**:
 - COLUMNS
 - COPY_HISTORY** (highlighted)
 - DATABASES
 - DATABASE_STORAGE_USAGE_HIST...
 - DATA_TRANSFER_HISTORY** (highlighted)
 - FILE_FORMATS
 - FUNCTIONS
 - LOAD_HISTORY** (highlighted)
 - LOGIN_HISTORY** (highlighted)
 - PIPE_USAGE_HISTORY** (highlighted)
 - QUERY_HISTORY** (highlighted)



EXAMPLE: EXPORTING USER ACCESS LOGS

```
COPY INTO @mystage/login_history
FROM
  (SELECT object_construct(*)
   FROM
     (SELECT event_timestamp, event_type, user_name, reported_client_type, error_message
      FROM TABLE(information_schema.login_history(
        dateadd('hour', -167, current_timestamp())))
      WHERE error_code IS NOT NULL
      ORDER BY event_timestamp))
FILE_FORMAT = (type = 'JSON');
```

```
{"ERROR_MESSAGE": "PASSWORD_EXPIRED", "EVENT_TIMESTAMP": "2020-05-29 12:7:52.009 -0700", "EVENT_TYPE": "LOGIN..."}, {"ERROR_MESSAGE": "INCORRECT_USERNAME_PASSWORD", "EVENT_TIMESTAMP": "2020-06-03 05:06:56.240 -0700", "EVENT_TYPE": "..."}, {"ERROR_MESSAGE": "INCORRECT_USERNAME_PASSWORD", "EVENT_TIMESTAMP": "2020-06-03 05:06:59.830 -0700", "EVENT_TYPE": "..."}, {"ERROR_MESSAGE": "INCORRECT_USERNAME_PASSWORD", "EVENT_TIMESTAMP": "2020-06-03 05:07:01.030 -0700", "EVENT_TYPE": "..."};
```



LAB EXERCISE: 16

Unload Semi-Structured Data

30 minutes

Tasks:

- Unload Semi-Structured JSON Data
- Unload Semi-Structured JSON Data to a Dynamic Path
- Unload Structured Data to a JSON File
- Unload Semi-Structured Parquet Data



SNOWFLAKE PRIVATE DATA SHARING

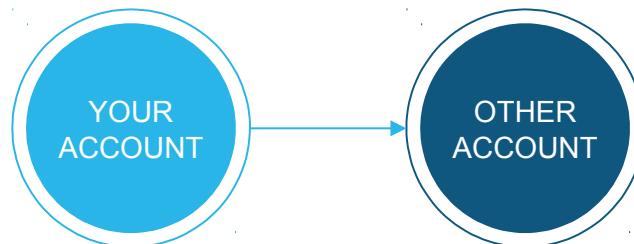


WAYS TO SHARE DATA

UNLOAD AND SHARE
("THE HARD WAY")

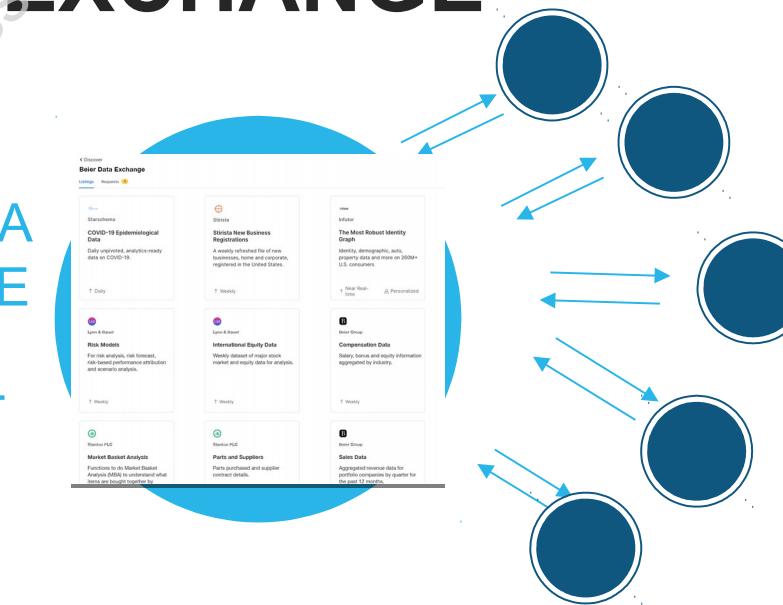


SECURE DIRECT DATA SHARING

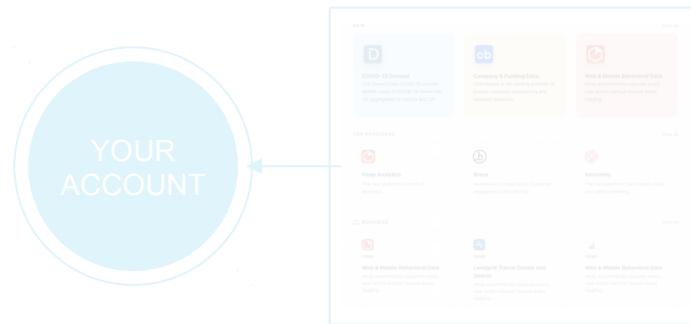


SNOWFLAKE DATA EXCHANGE

YOUR DATA
EXCHANGE
IN YOUR
ACCOUNT

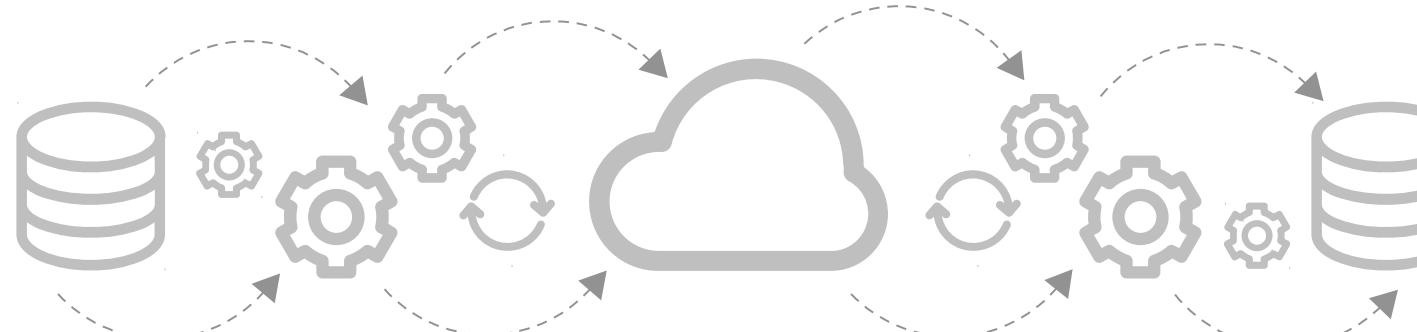


SNOWFLAKE DATA MARKETPLACE



TRADITIONAL WAYS OF SHARING DATA

Traditional Methods



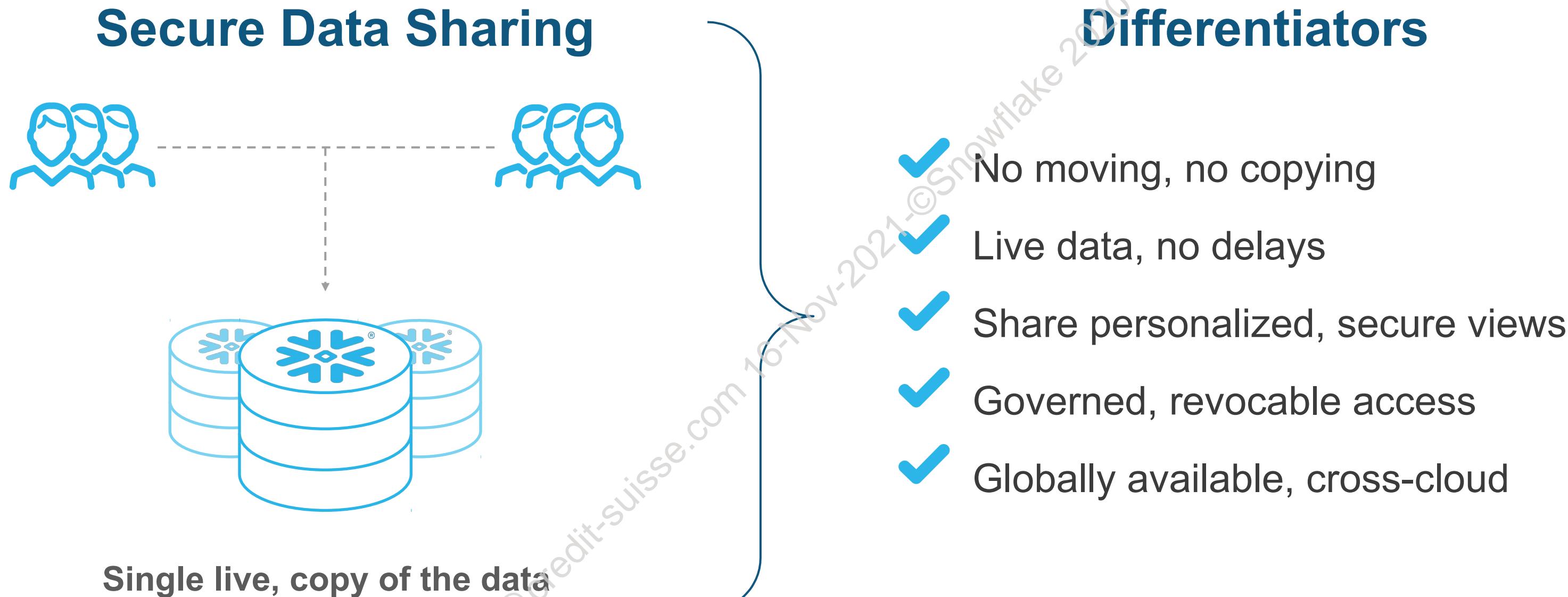
- Copying files in FTP/ cloud buckets
- Building, Maintaining & Calling APIs
- ETL pipelines
- Data marts

Gaps

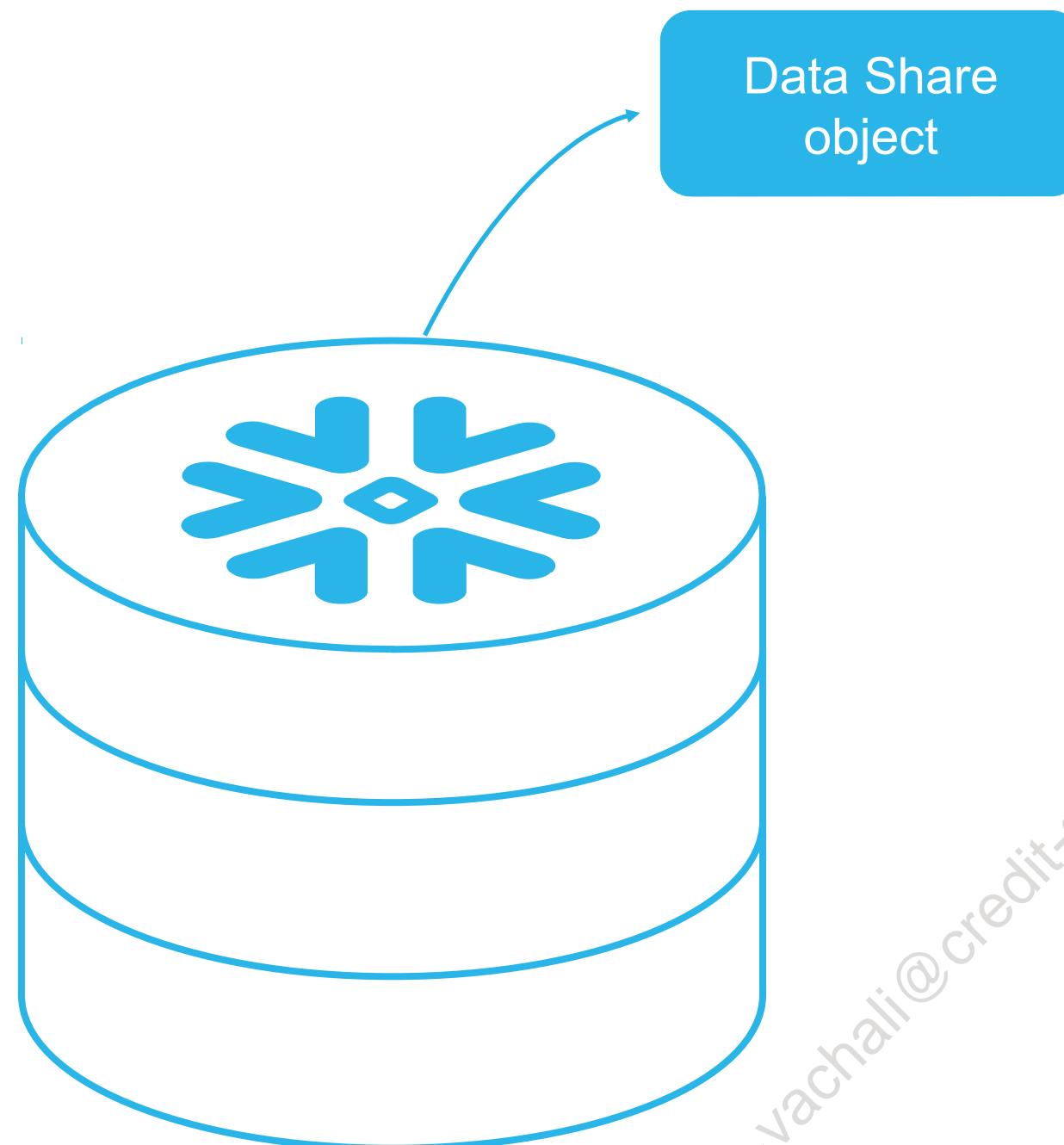
- ✖ Copy and move data
- ✖ Costly to maintain
- ✖ Data is delayed
- ✖ Error-prone
- ✖ Unsecure, once data is moved



SECURE DIRECT DATA SHARING



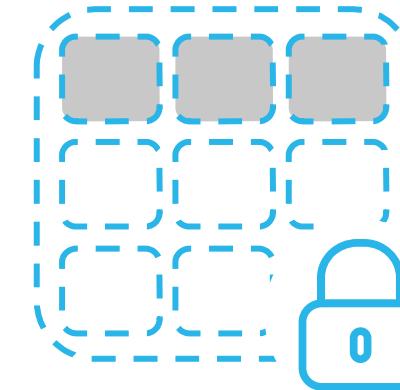
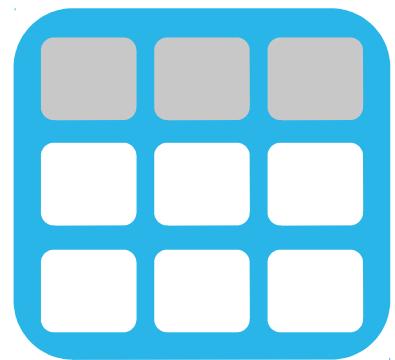
WHAT IS A SHARE?



- A named Snowflake object that encapsulates all the information required to share objects within a database
- Share Contains:
 - Privileges that grant access to the database, schema, or specific objects
 - Data Consumer or Reader account name



WHAT CAN YOU SHARE?



- Tables
- Secure Views
- Secure UDFs

WHO CAN YOU SHARE WITH?



Data Consumers



Reader Accounts

Data Consumers

- Have their own Snowflake account
- Pay for their own compute

Reader Accounts

- You create a Snowflake account for them
- You pay for any compute they use
- You pay for any storage they use (if you give them access to storage)



SHARE TYPES

Outbound Shares

- Created by the provider account
- Must be created by ACCOUNTADMIN (or role with appropriate privileges)
- Can share with an unlimited number of accounts

Inbound Shares

- Accessible to data consumers or reader accounts
- To consume, create a dummy database to hold the share
- Must be consumed by ACCOUNTADMIN (or role with appropriate privileges)
- Can consume an unlimited number of shares
- Reader accounts can only consume shares from the account that created it



SHARE LIMITATIONS

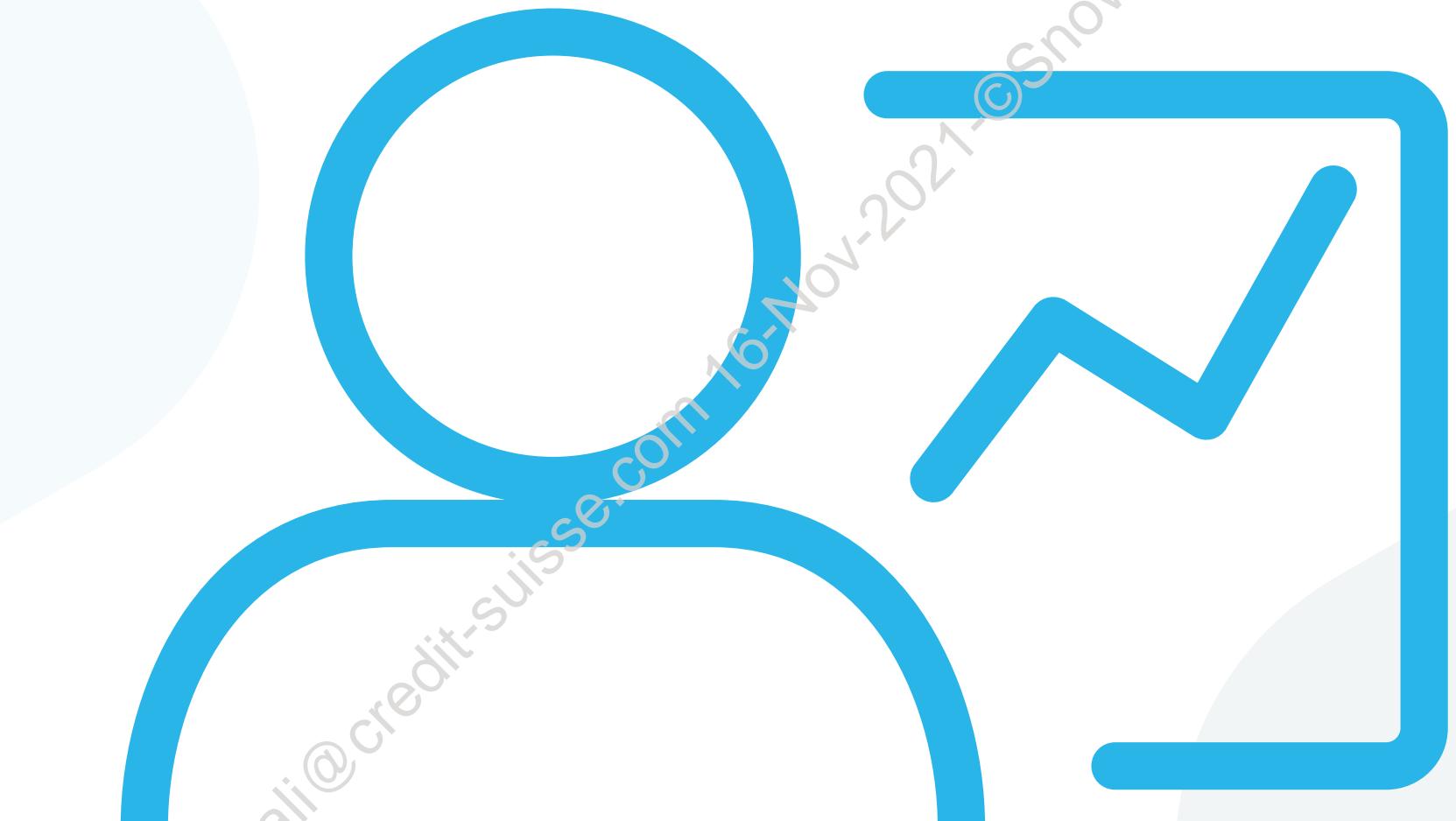
- New objects in shared database must be explicitly added to the share to be available
- Shares cannot be re-shared or cloned by the consumer/reader account
- Time travel information on the share is not available to consumers
- Shares cannot span multiple databases
- Shared cannot span regions
- Shares are always read-only
- Only tables, secure views, and secure UDFs can be shared



INSTRUCTOR DEMO

Data Sharing and Reader Accounts

15 minutes



anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



LAB EXERCISE: 17

Data Sharing

30 minutes

- Set up
- Basic Data Sharing
- Create Database on Consumer Account from Data Provider
- Use a Share as a Data Consumer
- Remove Objects
- Explore Secure User Defined Functions for Protecting Shared Data



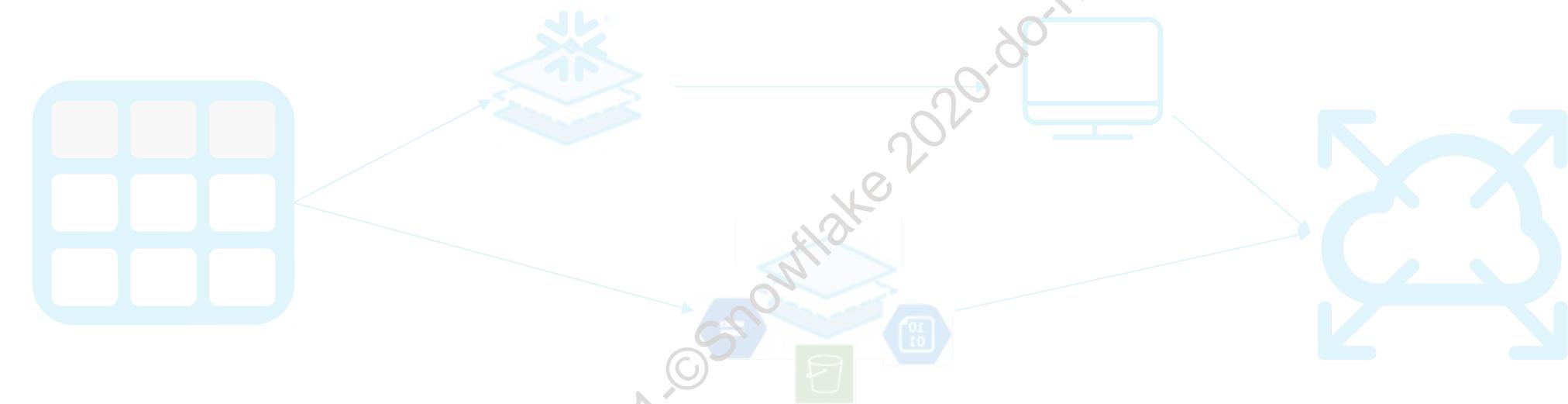
THE DATA MARKETPLACE

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy

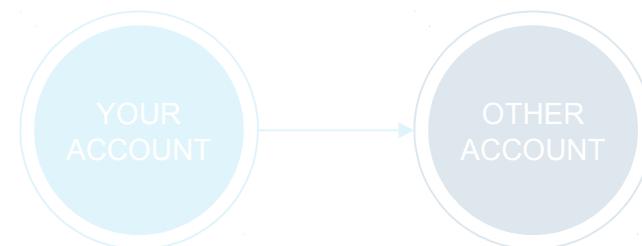


WAYS TO SHARE DATA

UNLOAD AND SHARE
("THE HARD WAY")



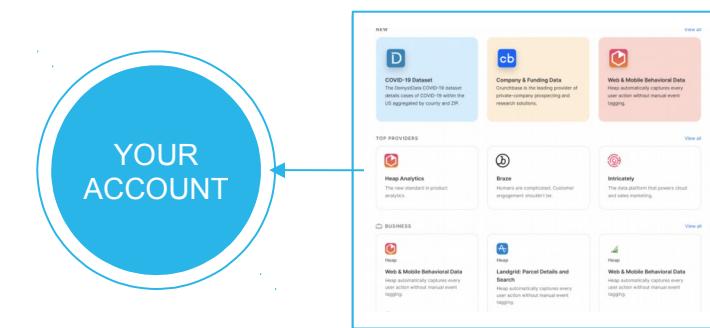
SECURE DIRECT
DATA SHARING



SNOWFLAKE DATA
EXCHANGE



SNOWFLAKE DATA
MARKETPLACE



SNOWFLAKE DATA MARKETPLACE

Snowflake is the only solution for accessing live, personalized data globally

The screenshot shows the Snowflake Data Marketplace interface. It includes sections for 'NEW' (Property Assessor, AbiliTec: Resolve Fragmented PII, Web & Mobile Behavioral Data), 'RECENTLY VIEWED' (AgilOne Customer Data Plat..., The Truth in B2B Intent Data, US Commercial Property Val...), and 'TOP PROVIDERS' (Heap Analytics, Braze, Intricately). Each item has a small icon, a title, a brief description, and a 'View all' link.

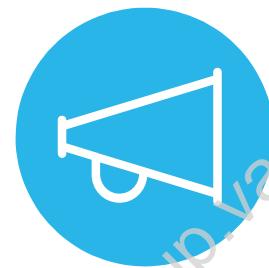
Live, ready-to-query data;
no copying or moving

Only data marketplace
with personalized data

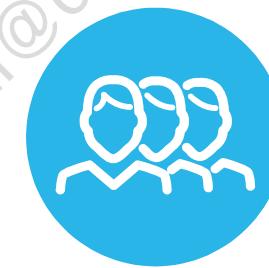
Globally available, across
clouds



Financial



Marketing



Demographic



Macroeconomic



Government



Healthcare



Business



SECURE DATA AND MONITOR USAGE

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake2020 do-not-copy

DATA GOVERNANCE

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



MODULE AGENDA

- Secure Views and UDFs
- Dynamic Data Masking
- External Tokenization

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy

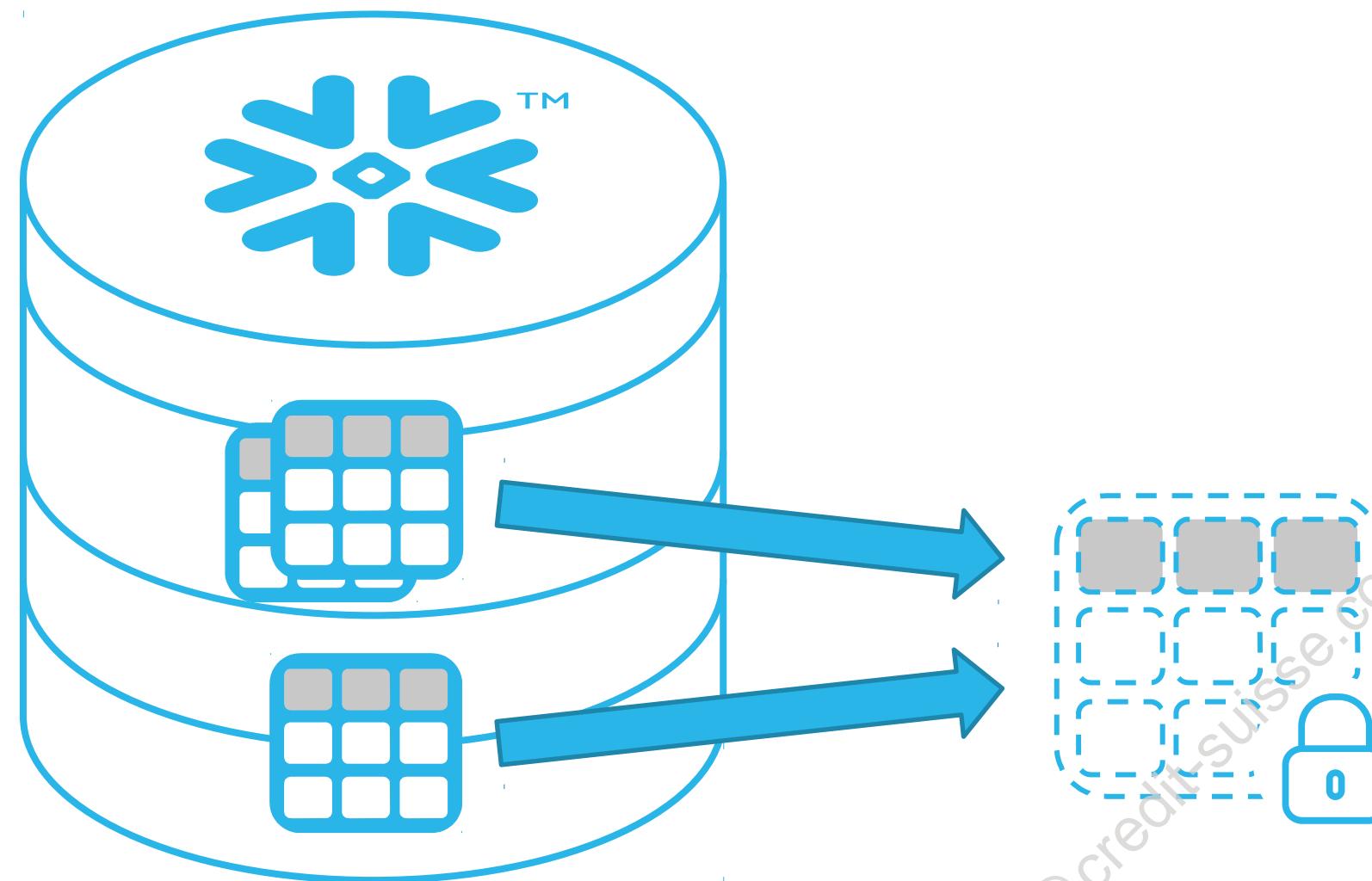


SECURE VIEWS AND UDFS

anup.vachali@credit-suisse.com 16-Nov-2021-©Snowflake 2020-do-not-copy



STANDARD VIEW VS SECURE VIEW



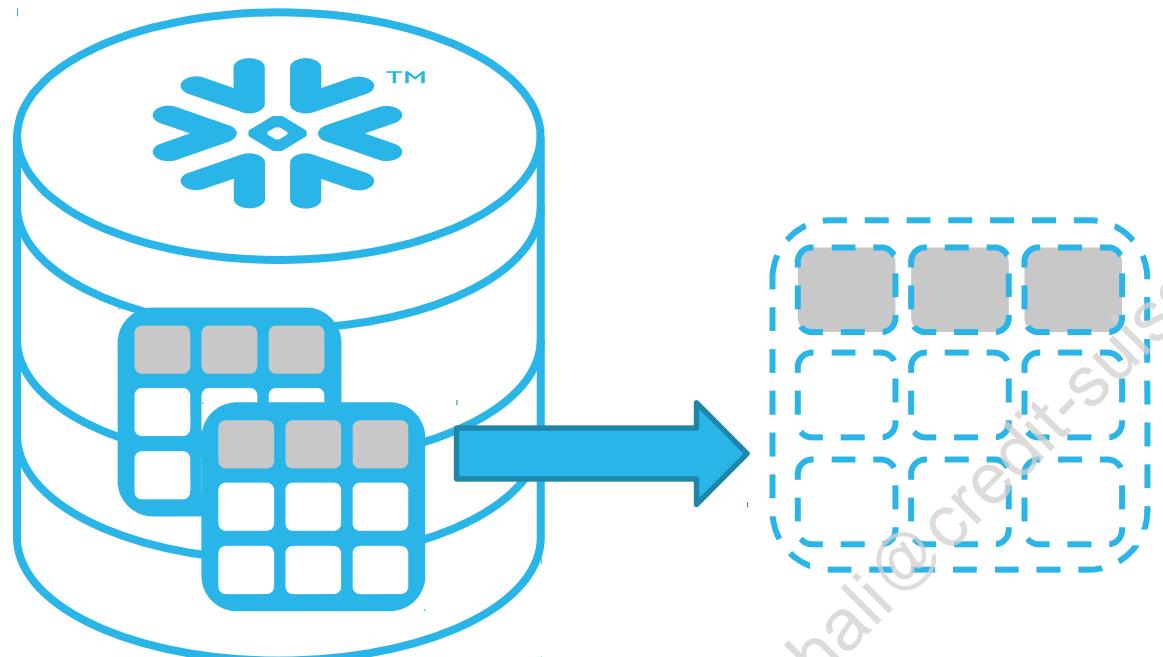
- Standard views are used to provide abstraction
 - `get_ddl` will show the view definition
- Secure views are used to expose data securely
 - Underlying logic, including `get_ddl`, is hidden from users



CREATE A VIEW

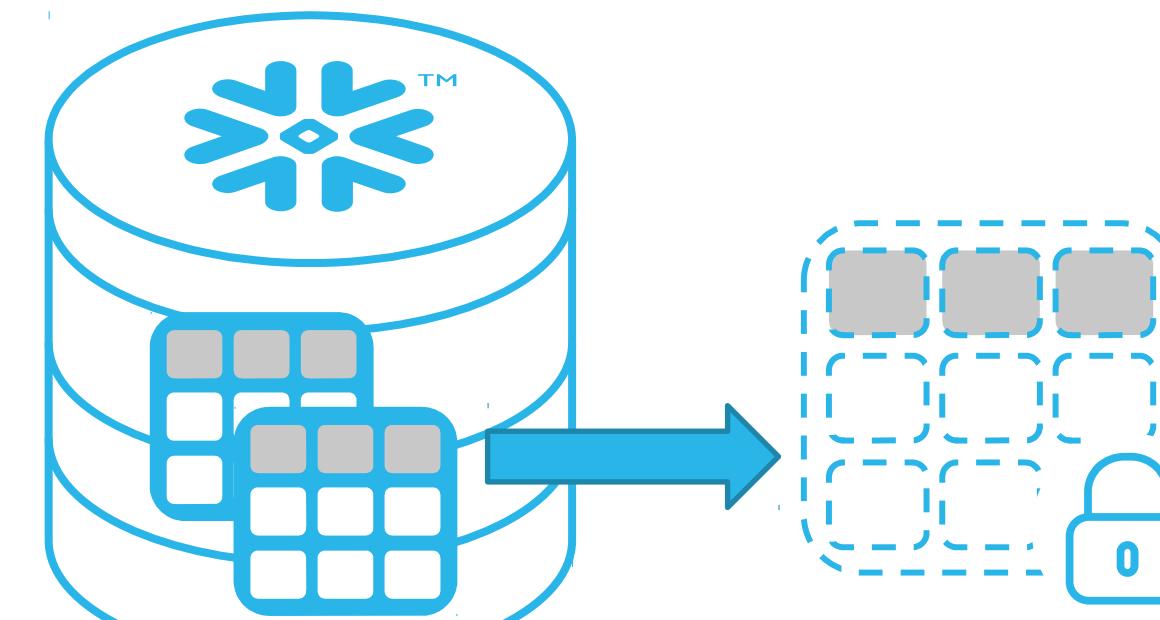
Standard Views

```
CREATE VIEW <name>  
AS  
<view definition>
```



Secure Views

```
CREATE SECURE VIEW <name>  
AS  
<view definition>
```



VIEWS AND QUERY LOGIC

- SHOW VIEWS (or get_ddl) will show the underlying query to anyone who can use the view
- Secure views do not show any of the view logic, except to owners

name	text	is_secure
ORDER_STATUS	<code>create view ORDER_STATUS AS SELECT o_custkey, o_orderke...</code>	false
SECURE_PATIENT_INFO		true



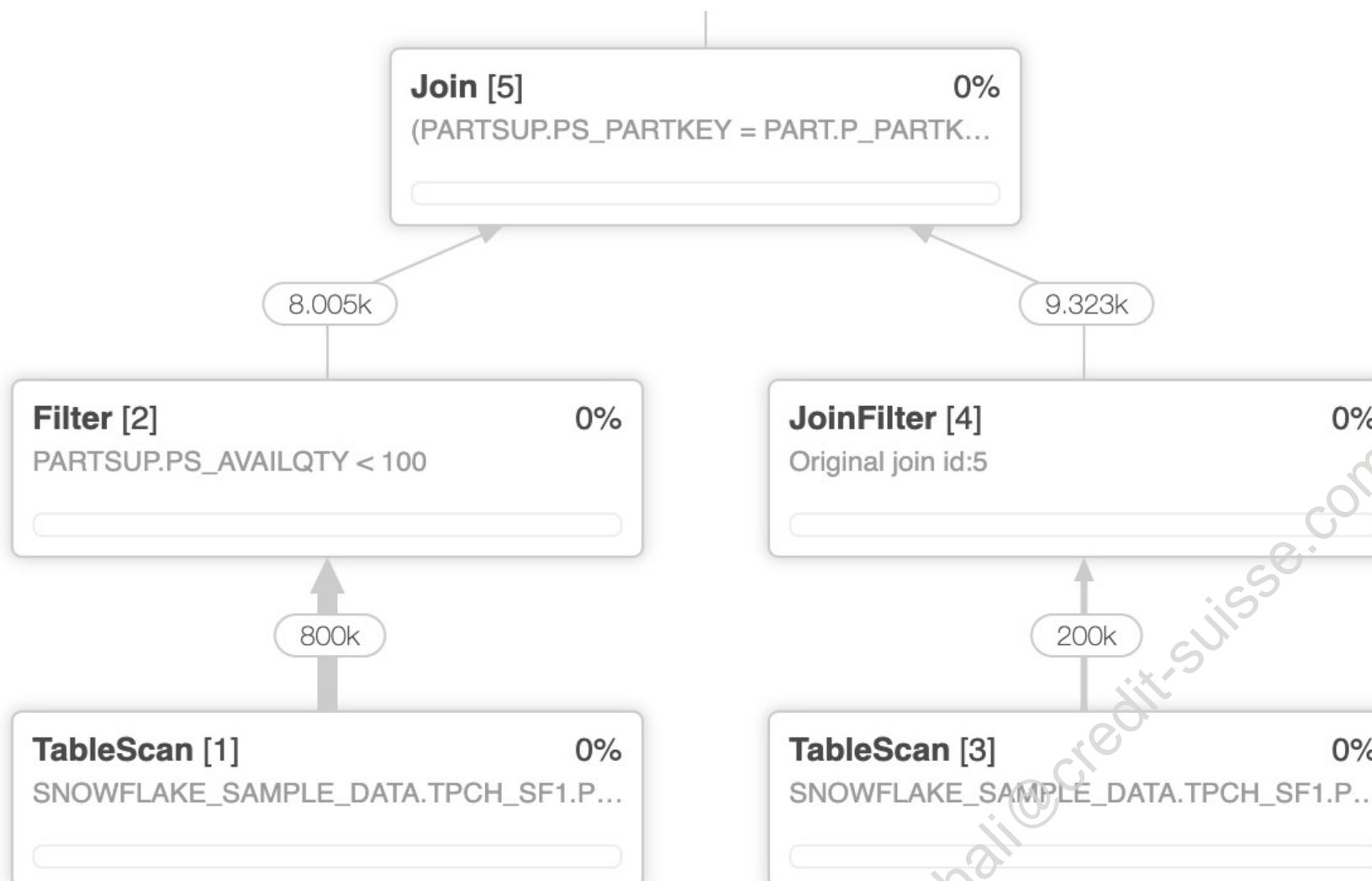
IEWS AND THE QUERY PROFILE

Standard View

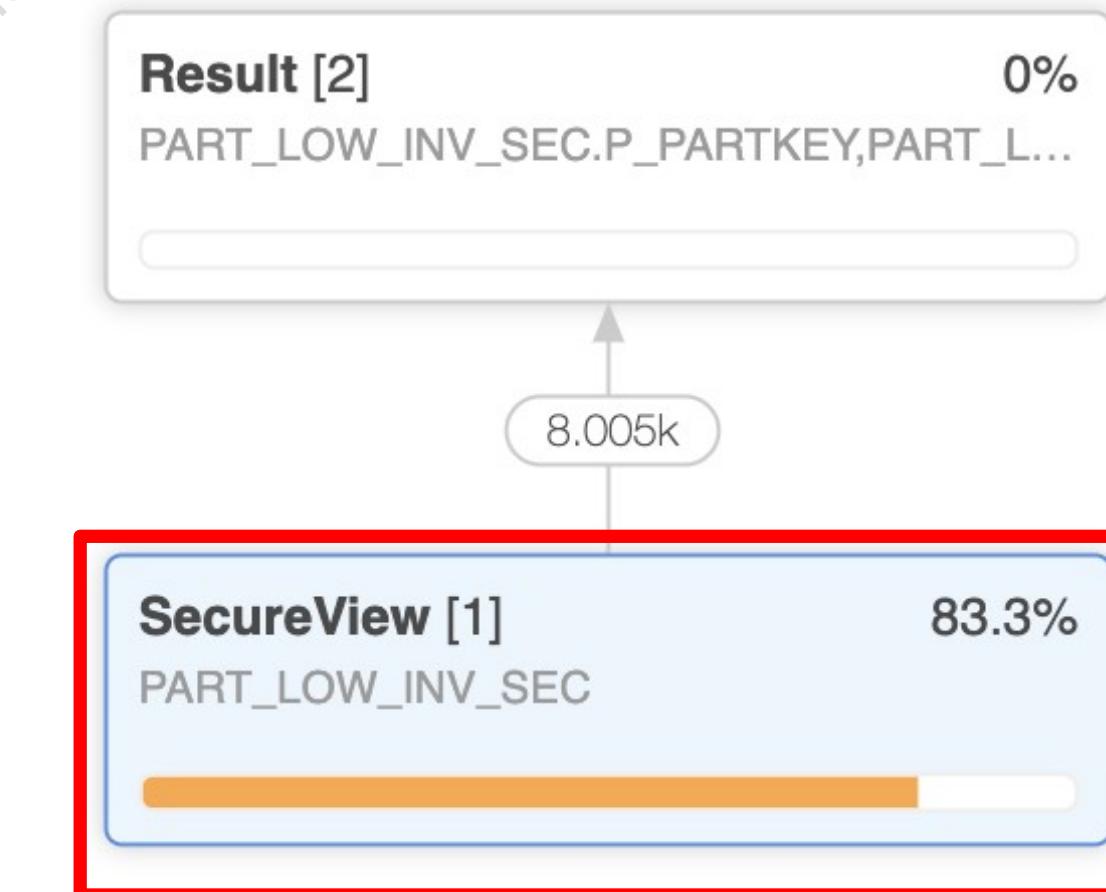


IEWS AND THE QUERY PROFILE

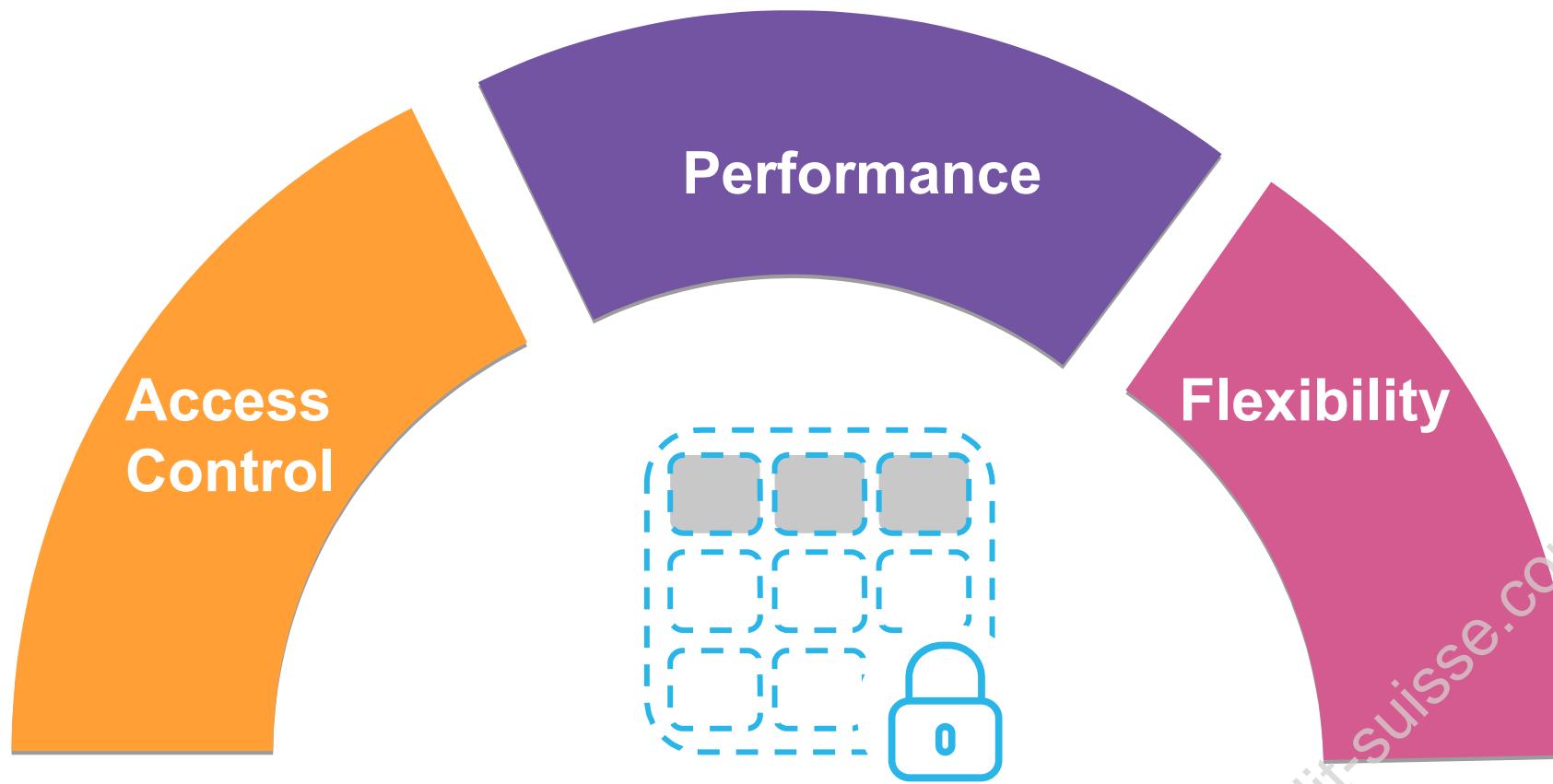
Standard View



Secure View



SECURE VIEW CONSIDERATIONS



- Limit owner privileges on secure view
- Data privacy takes precedence over query performance
- It's easy to convert between a secure view and a standard view

```
ALTER VIEW <name>
{ SET | UNSET } SECURE;
```



SECURE UDFs

```
CREATE SECURE FUNCTION  
get_salary(name VARCHAR)  
RETURNS NUMBER(10,2)  
AS  
'SELECT salary  
FROM payroll p  
WHERE p.emp_name = name';
```

- Allow logic to be shared
- Protect raw data from view or export
- Protect against exposing function's underlying logic
- Can convert existing UDF to secure

```
ALTER FUNCTION <name>  
{SET | UNSET} SECURE;
```



DYNAMIC DATA MASKING

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



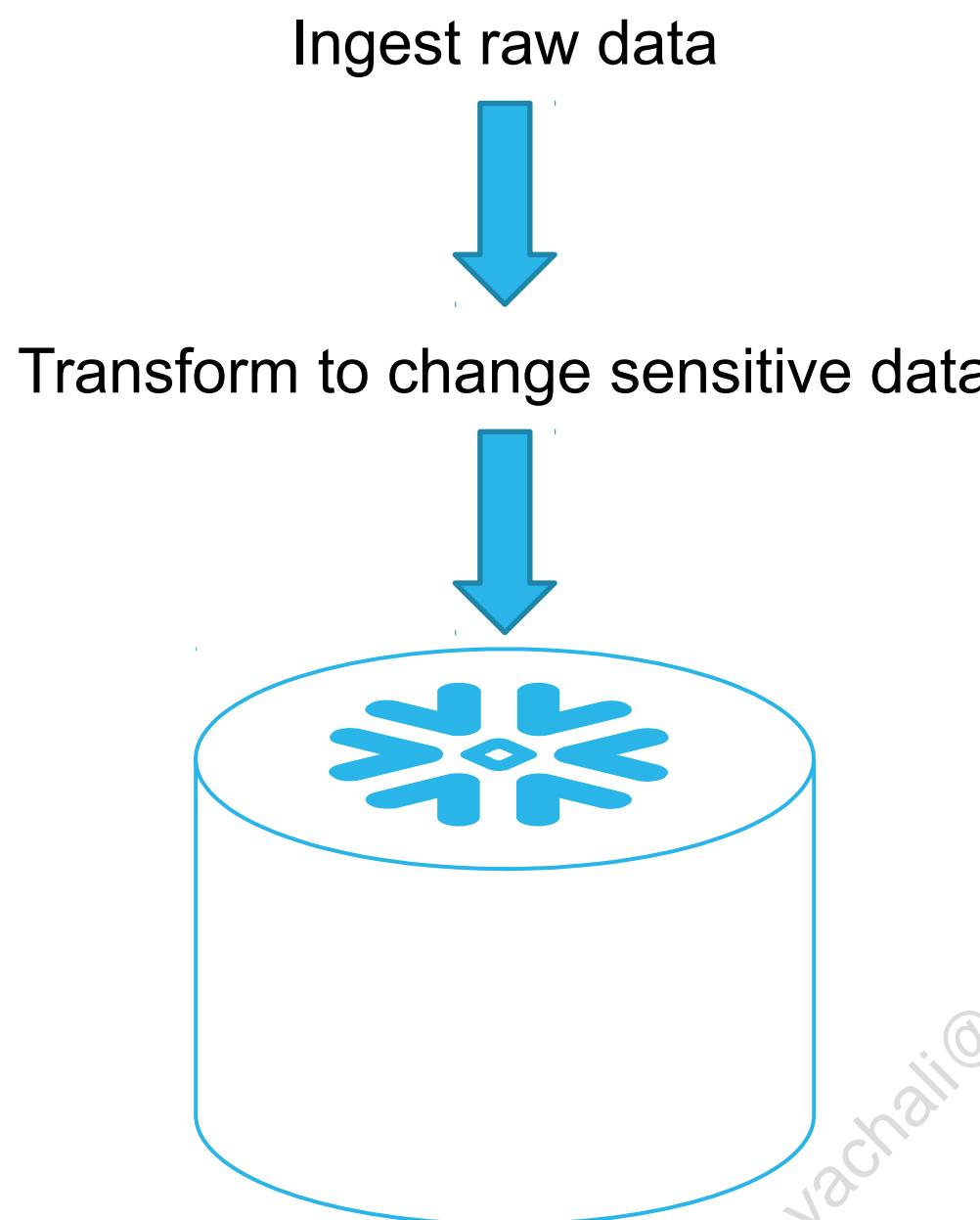
WHY DYNAMIC DATA MASKING?

- Protecting access to PII and PHI data at the column level is required by many compliance directives
- Role-based access controls limit access only at the object level, not column level
- Secure views may involve additional management overhead, and the secure view's owner has access to the data
- Using data masking in conjunction with RBAC ensures a granular level of access control by limiting what data can be viewed by all roles and users
 - Snowflake supports using Dynamic Data Masking on tables and views



IN-PLACE DATA MASKING

SENSITIVE DATA PHYSICALLY CHANGED

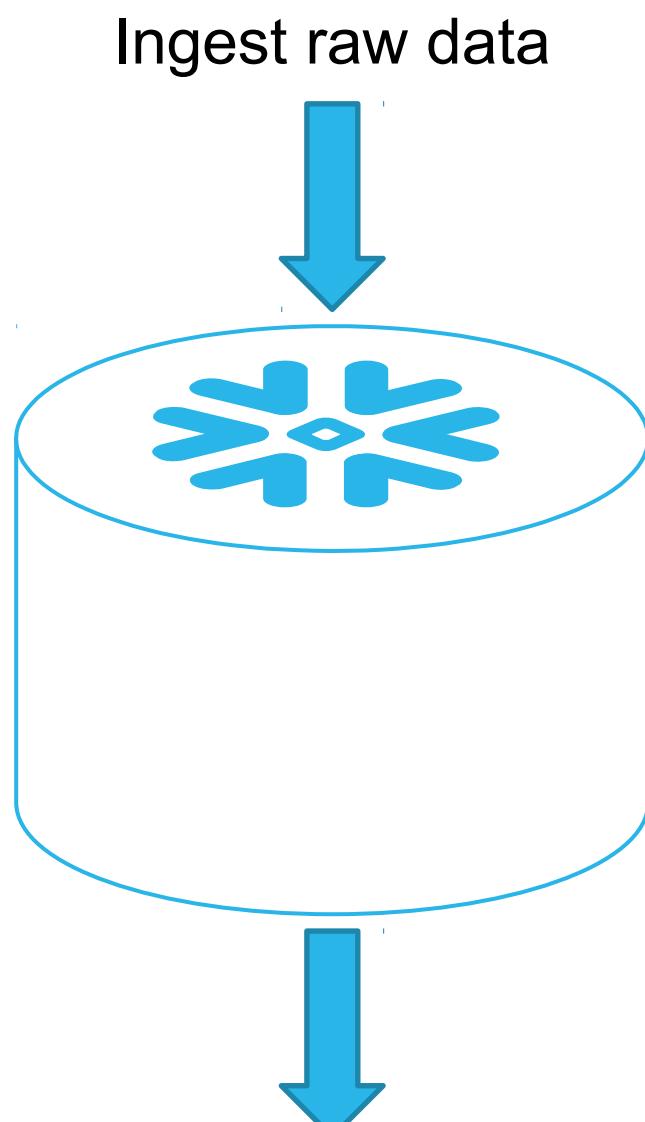


- Typically used to mask data that is copied from production to non-production (for example, for testing)
- Cannot be applied to production databases, as doing so effectively corrupts the data
- Time-consuming to create and test scripts
- Potentially time-consuming to run script when data is moved out of production
- What if you forget?



DYNAMIC DATA MASKING

MASK SENSITIVE DATA WHEN NEEDED

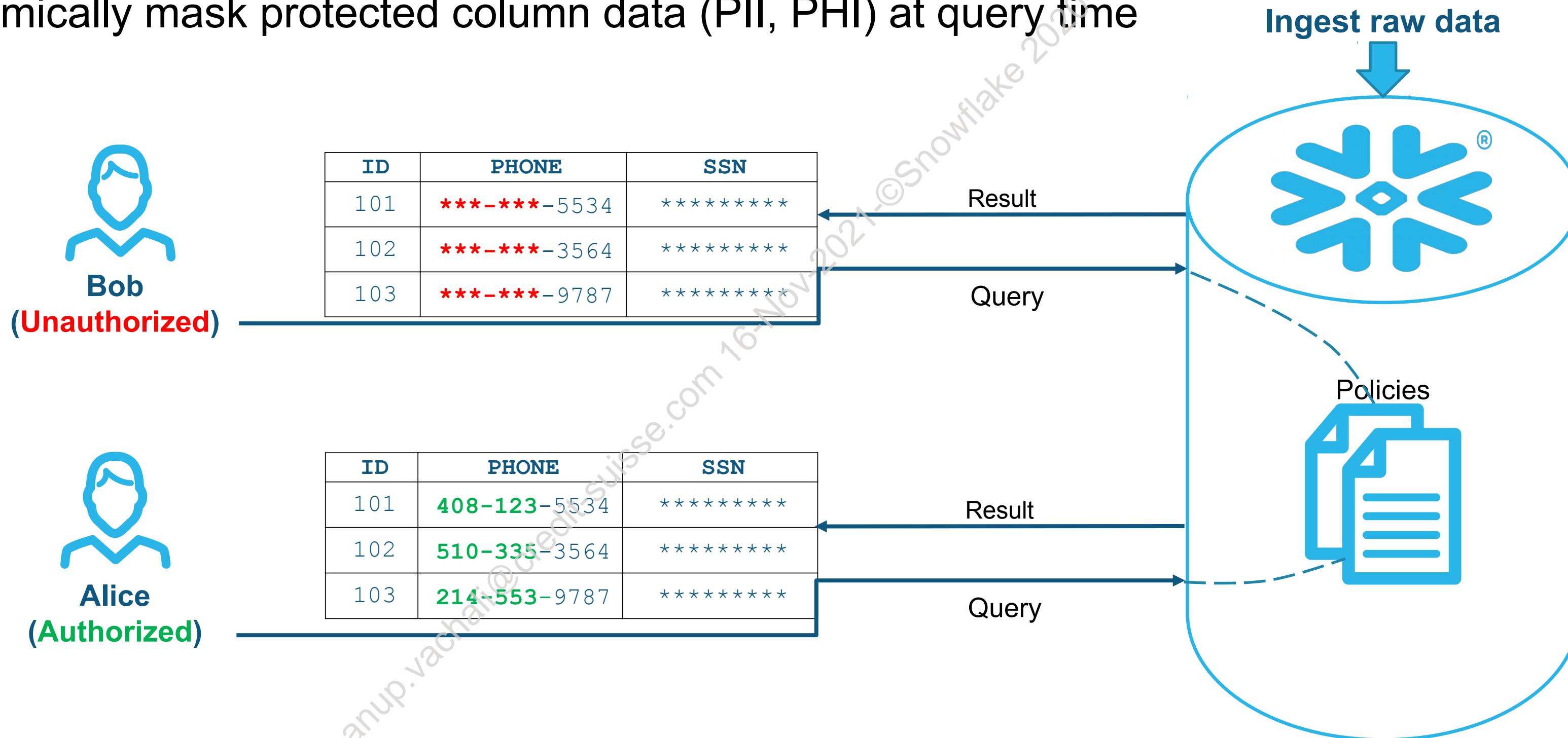


- Can be deployed in both production and non-production databases
- Removes the burden of having to modify data when creating new non-production databases
- Can be applied on all environments
- Can mask based on user or role



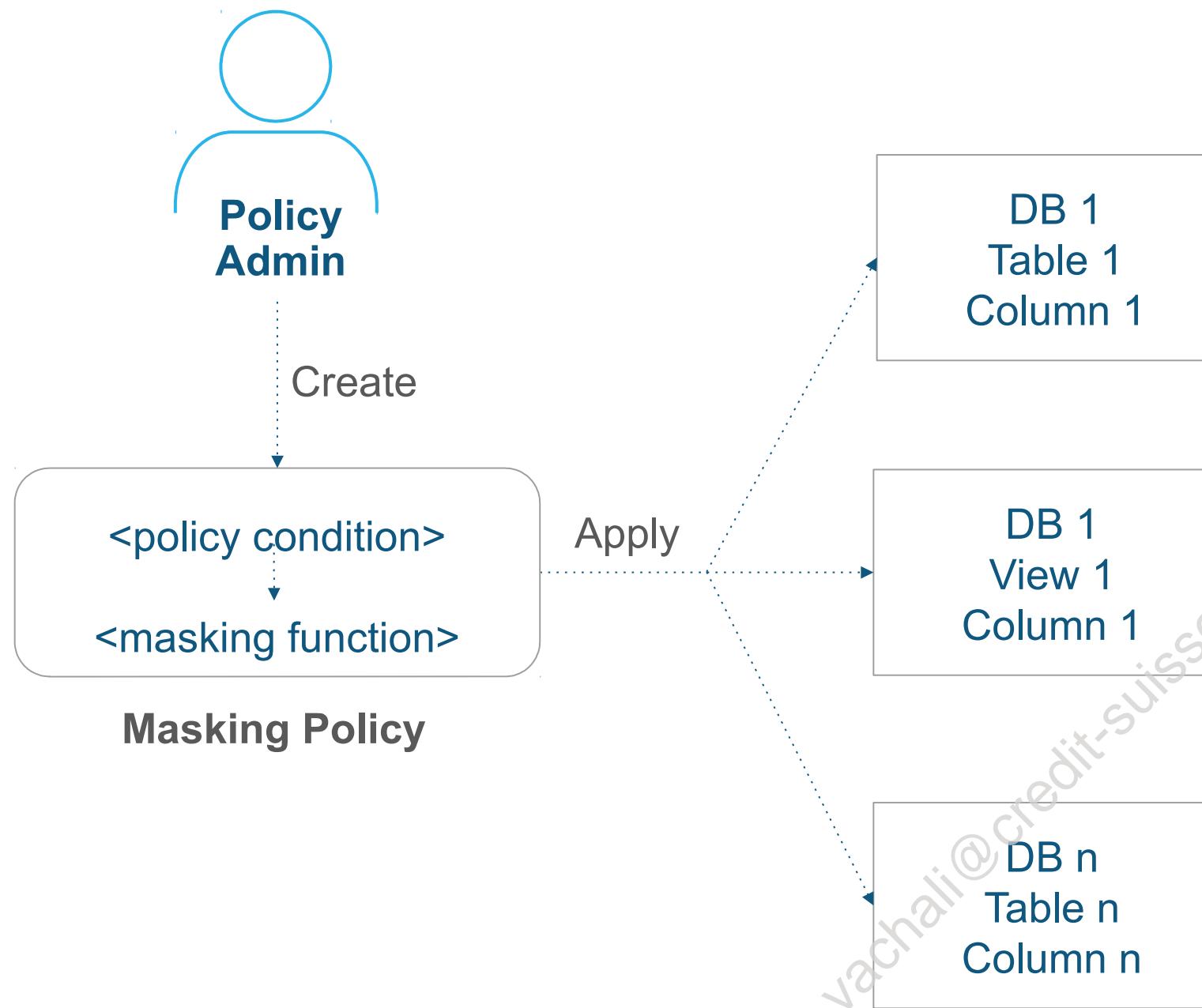
HOW DYNAMIC DATA MASKING WORKS

- Dynamically mask protected column data (PII, PHI) at query time



MASKING POLICIES

ENTERPRISE-EDITION FEATURE



- Contain condition(s), and the masking function to apply under those conditions
- Can be applied to one or more table, view, or external table columns
- Use nested policy execution for views - policy on table executed before policy on view(s)
- Support all data types, shared data, and streams
- Carry over policy associations when cloned



MASKING POLICY EXAMPLE 1

- Only the ANALYST role can see the email address

Raw value in the masked column

```
CREATE MASKING POLICY email_mask AS  
(val STRING) RETURNS STRING ->  
CASE  
    WHEN current_role() IN ('ANALYST') THEN val  
    ELSE '*****'  
END;
```

Other available functions
include `current_user()`
and `current_account()`



MASKING POLICY EXAMPLE 1

- Only the ANALYST role can see the email address

```
CREATE MASKING POLICY email_mask AS
(val STRING) RETURNS STRING ->
CASE
WHEN current_role() IN ('ANALYST') THEN val
ELSE '*****'
END;
```

SQL expression
that transforms the
data

Data type of the
column you are
masking

Data type to
be used for
the mask



MASKING POLICY EXAMPLE 2

- Use Conditional Expression Functions, Context Functions, and UDFs to write the SQL expression

```
CREATE MASKING POLICY mask_ssn AS
(val STRING) RETURNS STRING ->
CASE
    WHEN current_role() IN ('SECURE_ROLE') THEN val
    WHEN current_role() IN ('SUPPORT') THEN
        regex_replace(val, '^.{6}', '*****')
    WHEN current_role() IN ('INSTRUCTOR') THEN mask_udf(val)
    ELSE '*****'
END;
```



APPLY MASKING POLICY

- Once the masking policy exists, it can be applied to columns in multiple tables or views:

```
ALTER TABLE taxpayer MODIFY COLUMN email SET MASKING POLICY email_mask;
```

```
ALTER TABLE orders MODIFY COLUMN cust_email SET MASKING POLICY email_mask;
```

```
ALTER VIEW prospects MODIFY COLUMN email SET MASKING POLICY email_mask;
```

- Multiple policies can be applied to a single table or view:

```
ALTER TABLE taxpayer MODIFY COLUMN email SET MASKING POLICY email_mask;
```

```
ALTER TABLE taxpayer MODIFY COLUMN ssn SET MASKING POLICY ssn_mask;
```

```
ALTER TABLE taxpayer MODIFY COLUMN address SET MASKING POLICY addr_mask;
```

- Masks can be removed if needed:

```
ALTER TABLE taxpayer MODIFY COLUMN email UNSET MASKING POLICY;
```



DYNAMIC DATA MASKING – USAGE EXAMPLE

- SSN Masking policy:

```
create or replace masking policy ssn_mask as
  (val string) returns string ->
  case
    when current_role() in ('SYSADMIN') then val
    when current_role() in ('TRAINING_ROLE') then repeat('*', length(val)-4) || right(val, 4)
    else '*** REDACTED ***'
  end;
```

- Applying SSN masking policy to table columns:

```
alter table taxpayer_dependents modify column taxpayer_ssn set masking policy ssn_mask;
alter table taxpayer_dependents modify column dependent_ssn set masking policy ssn_mask;
```



DYNAMIC DATA MASKING – USAGE EXAMPLE

- Query output using role SYSADMIN:

DEPENDENT_SSN	TAXPAYER_SSN	DEP_RELATIONSHIP
123456798	678465237	Spouse
332129001	678465237	Daughter

- Query output using role TRAINING_ROLE:

DEPENDENT_SSN	TAXPAYER_SSN	DEP_RELATIONSHIP
*****6798	*****5237	Spouse
*****9001	*****5237	Daughter

- Query output using role PUBLIC:

DEPENDENT_SSN	TAXPAYER_SSN	DEP_RELATIONSHIP
*** REDACTED ***	*** REDACTED ***	Spouse
*** REDACTED ***	*** REDACTED ***	Daughter



SHOW MASKING POLICIES

- Lists masking policy information including the creation date, database and schema names, and owner.
- Only returns rows for:
 - The masking policy owner OR
 - A role with the APPLY privilege on the masking policy
- Examples of usage:
 - `show masking policies;` -- **Current database only**
 - `show masking policies like '%email%';`
 - `show masking policies in account;` -- **All databases in account**
 - `show masking policies like '%email%' in database taxpayer_db;`



DESCRIBE MASKING POLICIES

```
DESCRIBE masking policy test_email_mask;
```

Sample output:

Row	name	signature	return_type	body
1	TEST_EMAIL_MASK	(VAL VARCHAR)	VARCHAR(16777216)	case when current_role() in ('SYSA...')

Click the value in the body column to view the masking policy definition:

Details

```
1 case
2   when current_role() in ('SYSADMIN') then val
3   when current_role() in ('TRAINING_ROLE') then
4     regexp_replace(val, '.+\@', '*****@')
5   else '*** REDACTED ***'
6 end
```



LAB EXERCISE: 18

Using Dynamic Data Masking

20 minutes

- Identifying PII Data
- Creating Masking Policies
- Use Data Masking Policies
- Test Masking Policies

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



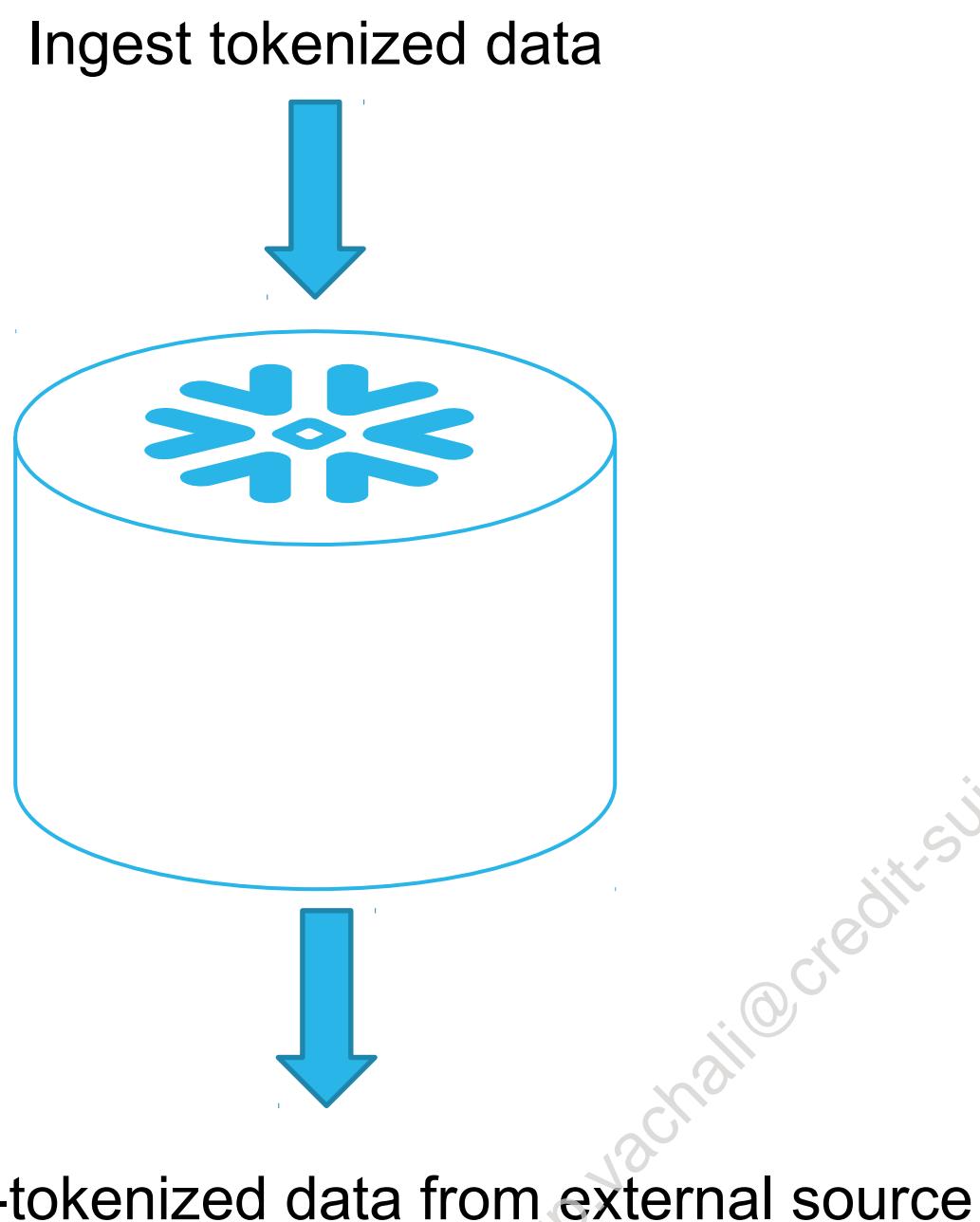
EXTERNAL TOKENIZATION

anup.vachali@credit-suisse.com 16-Nov-2021-©Snowflake 2020-do-not-copy



EXTERNAL TOKENIZATION

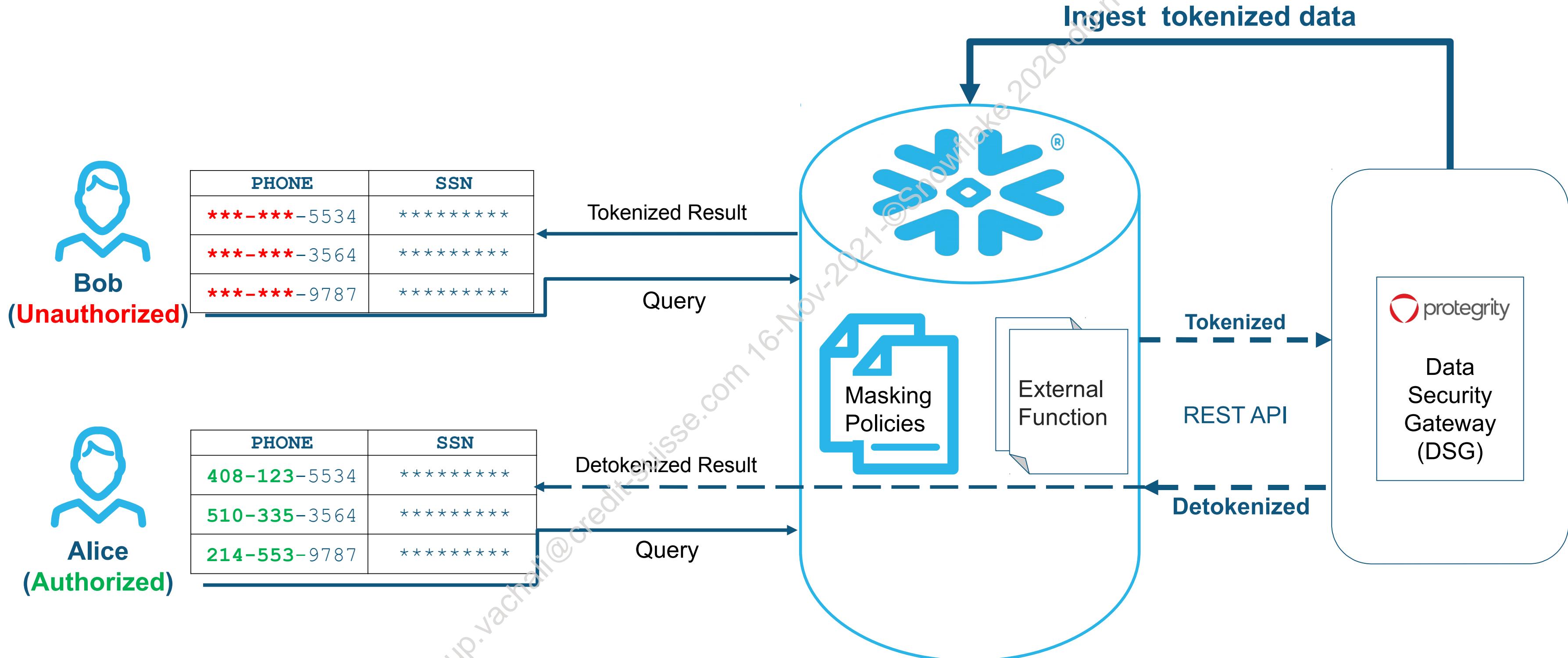
TOKENIZE DATA STORED IN SNOWFLAKE



- Actual data is never stored in Snowflake
- Policies determine who can view the de-tokenized data
- When needed, external function is used to retrieve de-tokenized data from provider



HOW EXTERNAL TOKENIZATION WORKS



DATA MASKING VS TOKENIZATION

	DYNAMIC DATA MASKING	EXTERNAL TOKENIZATION
Data storage	Stored in clear	Stored as tokenized by partner solution
Delivery	Built-in	Through partner integrations (such as Protegrity) and external functions
Query time	User sees masked vs unmasked data based on Snowflake masking policy	User sees tokenized vs de-tokenized data based on partner solution's access policy
Availability	Does not depend on cloud provider; available to all	Currently supported on Amazon AWS – support is planned for Azure and GCP



DATABASE REPLICATION

anup.vachali@credit-suisse.com 16 Nov 2021 ©Snowflake 2020-do-not-copy



MODULE AGENDA

- Overview
- Set Up Replication
- Failover and Failback

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy

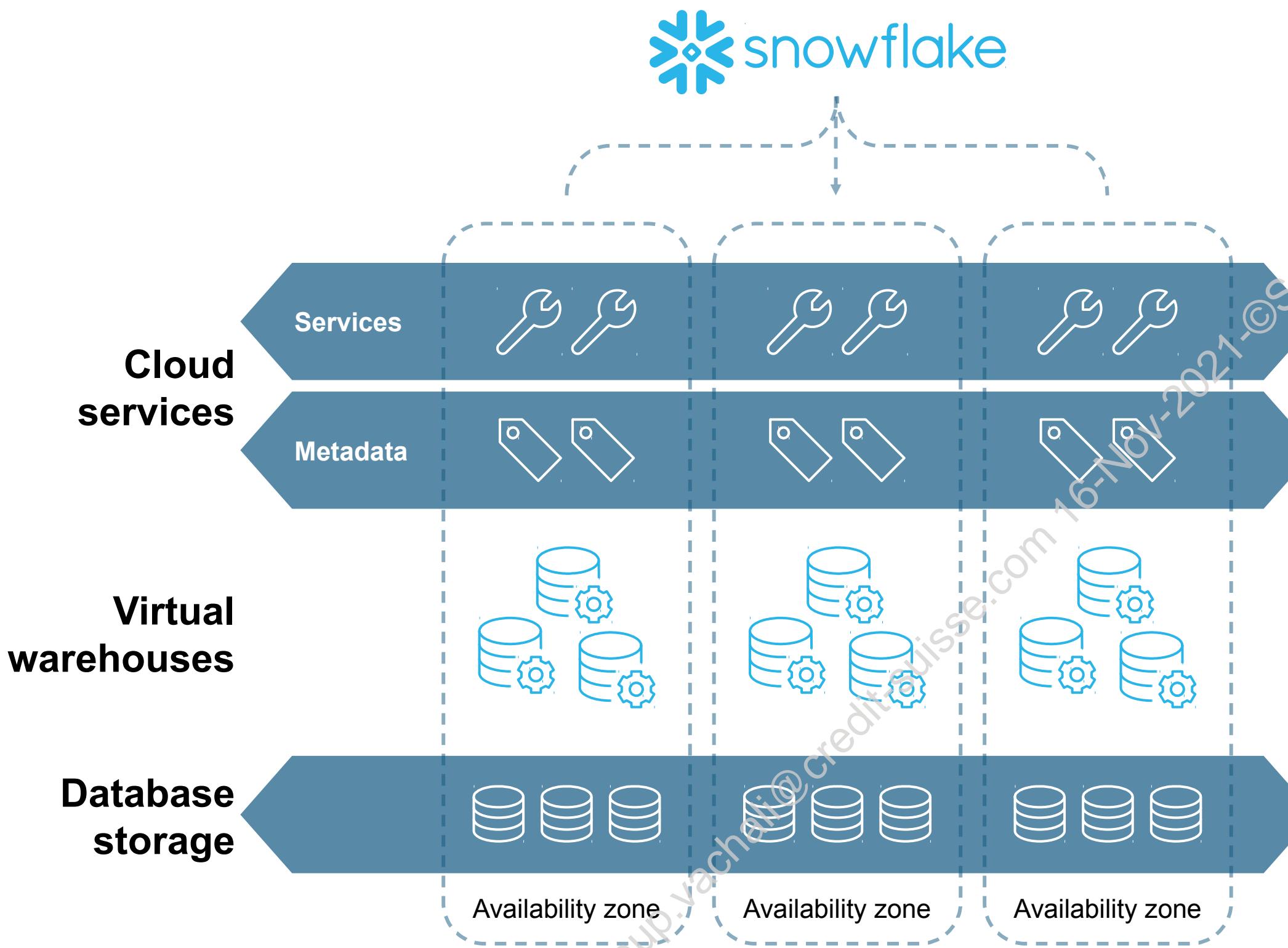


OVERVIEW

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



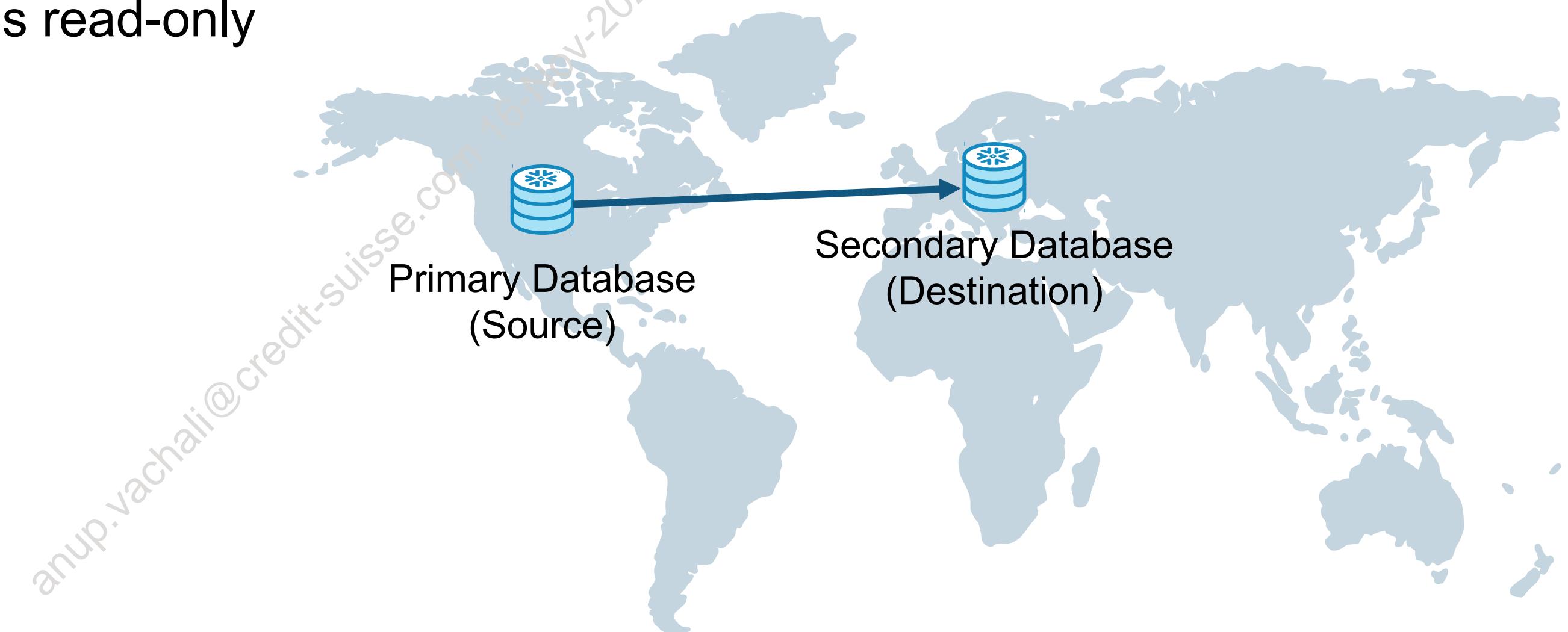
CLOUD PROVIDER CONTINUOUS AVAILABILITY



- Resiliency across multiple availability zones
 - geographic separation
 - built for synchronous replication
 - 11 9's durability
 - 4 9's availability
- Scale-out of all tiers: metadata, compute, storage
- Automatic updates & patches with zero downtime

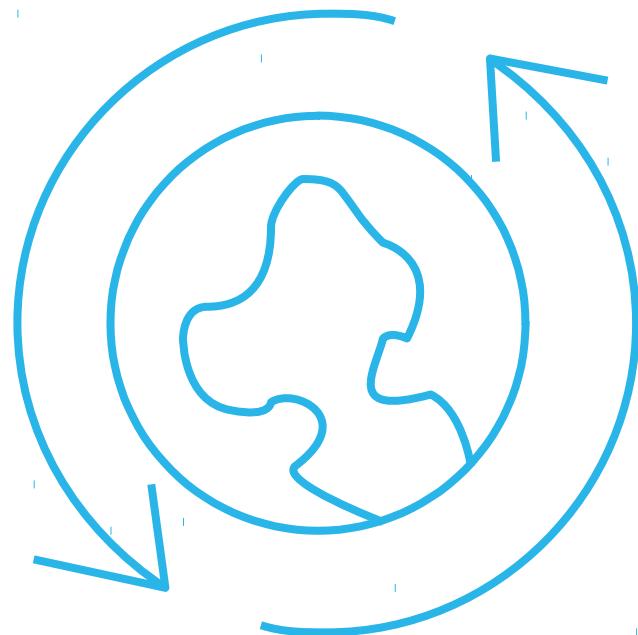
WHAT IS REPLICATION?

- Keeps database objects and stored data synchronized between one or more accounts (within the same organization)
- Unit of replication is a database (permanent or transient)
- Secondary database is read-only



REPLICATION USE CASES

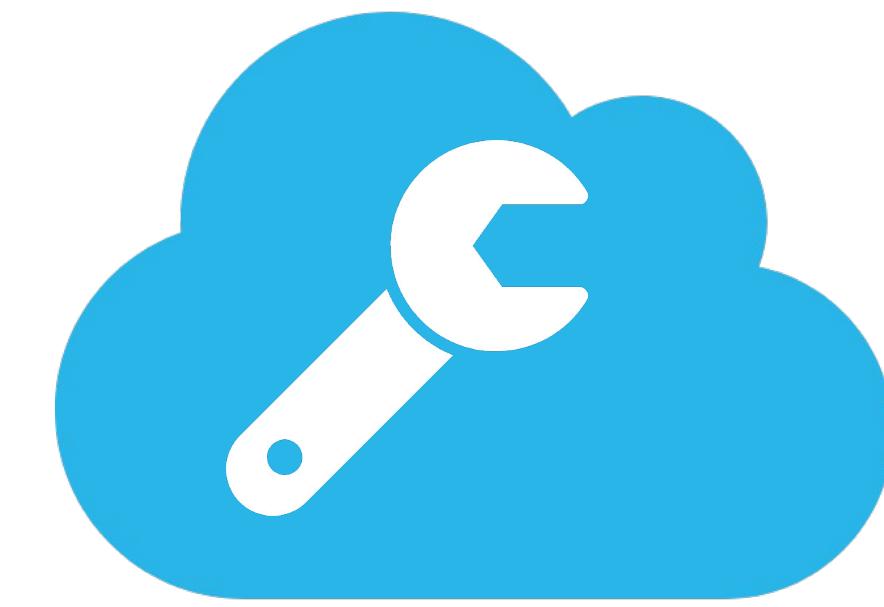
Business Continuity &
Disaster Recovery



Secure Data Sharing
Across Regions / Clouds



Data Portability for
Account Migrations

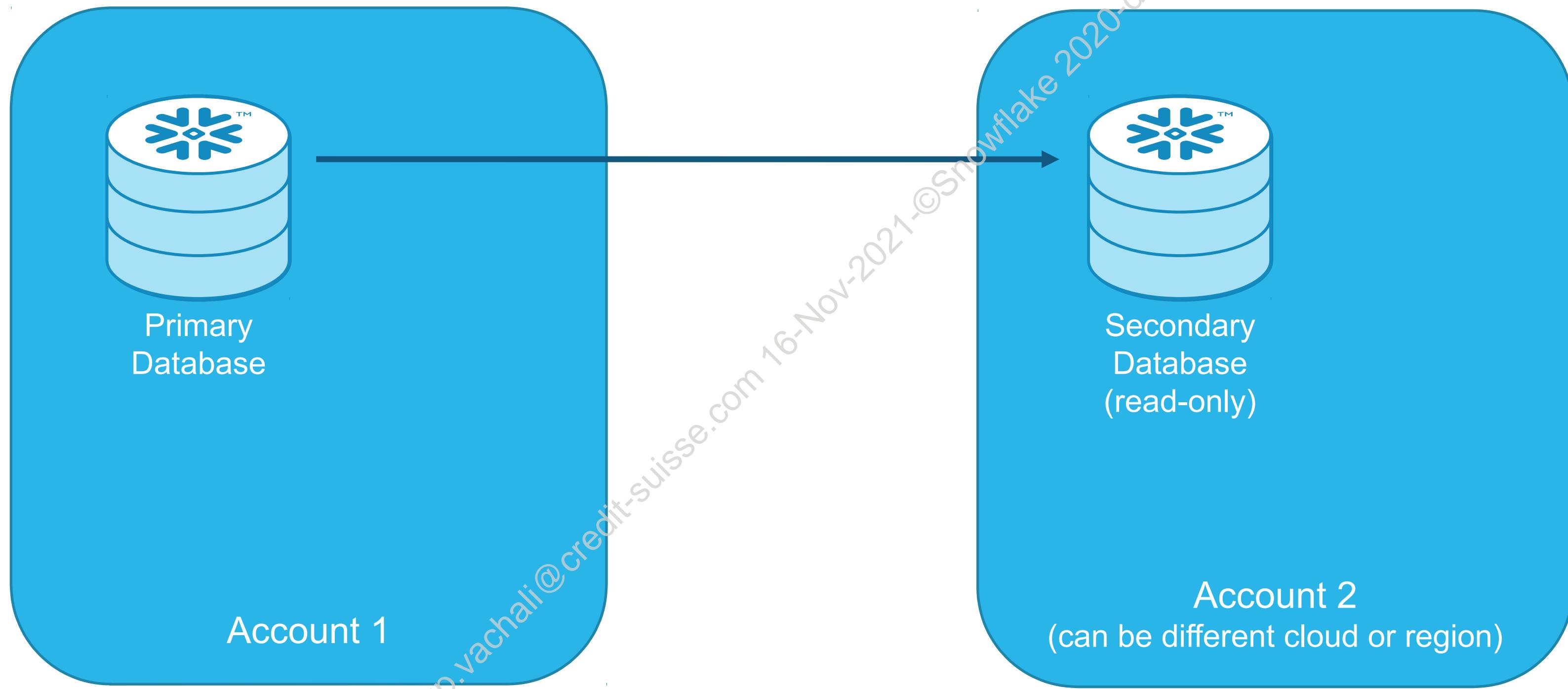


SET UP REPLICATION

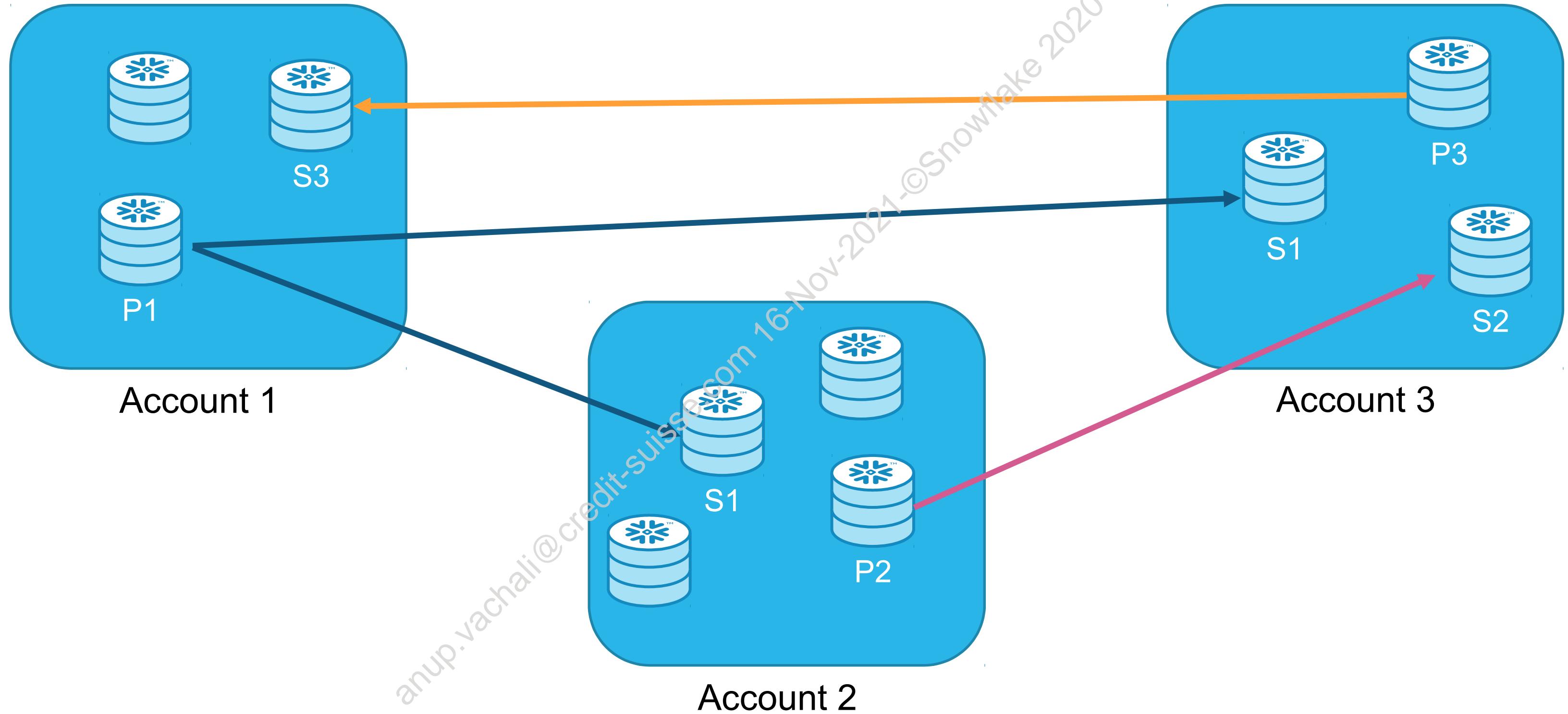
anup.vachali@credit-suisse.com 16-Nov-2021-©Snowflake 2020-do-not-copy



DATABASE REPLICATION



DATABASE REPLICATION



SET UP DATABASE REPLICATION

AWS_US_WEST_2

Account: COMPANYA-1



Research

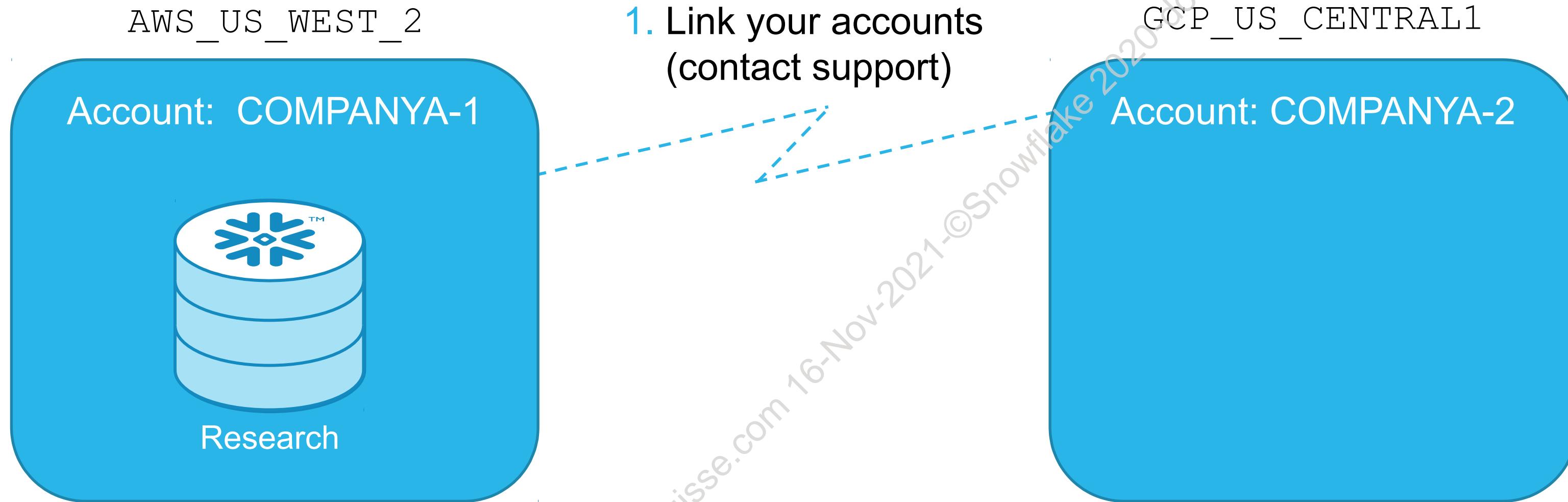
GCP_US_CENTRAL1

Account: COMPANYA-2

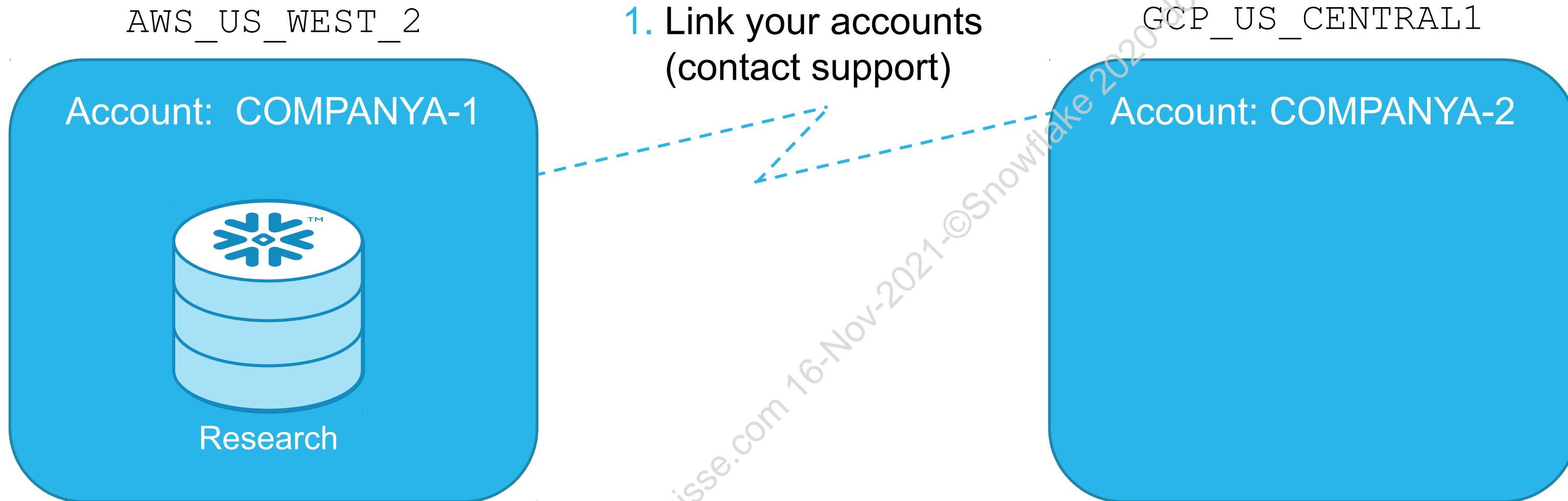
anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020. All rights reserved. Reproduction or distribution is prohibited.



SET UP DATABASE REPLICATION



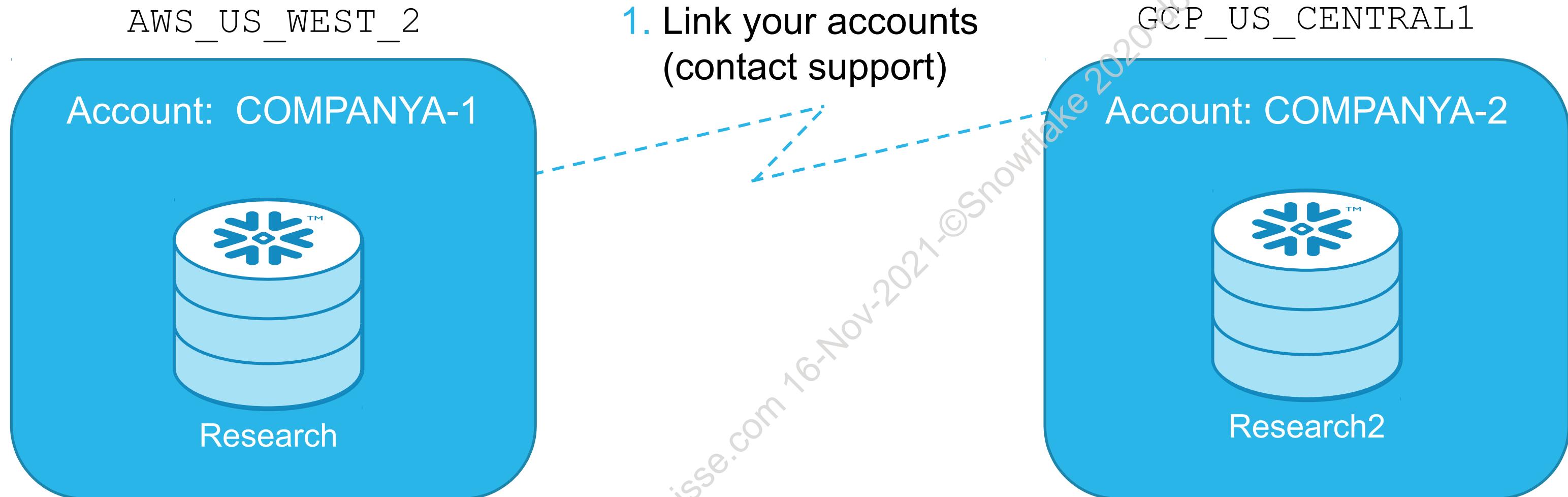
SET UP DATABASE REPLICATION



ALTER DATABASE research **ENABLE REPLICATION**
TO ACCOUNTS gcp_us_central1.companya2;



SET UP DATABASE REPLICATION

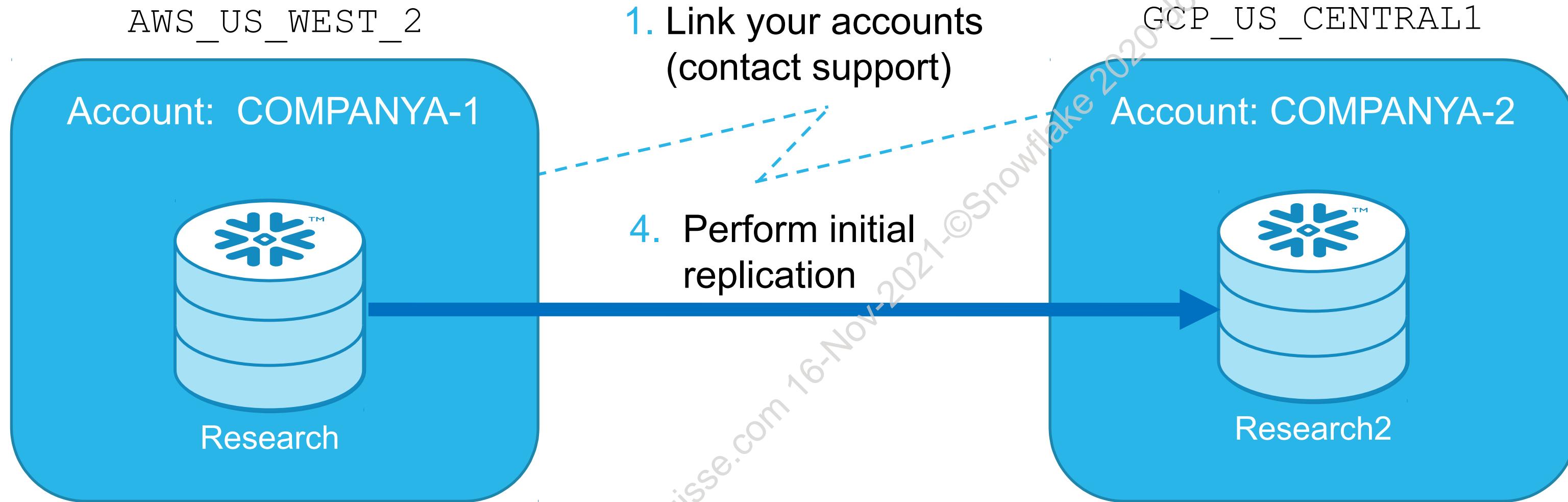


1. Link your accounts
(contact support)
2. Enable replication on primary database
3. Create a secondary database

```
CREATE DATABASE research2 AS REPLICA OF  
aws_us_west2.companya1.research;
```

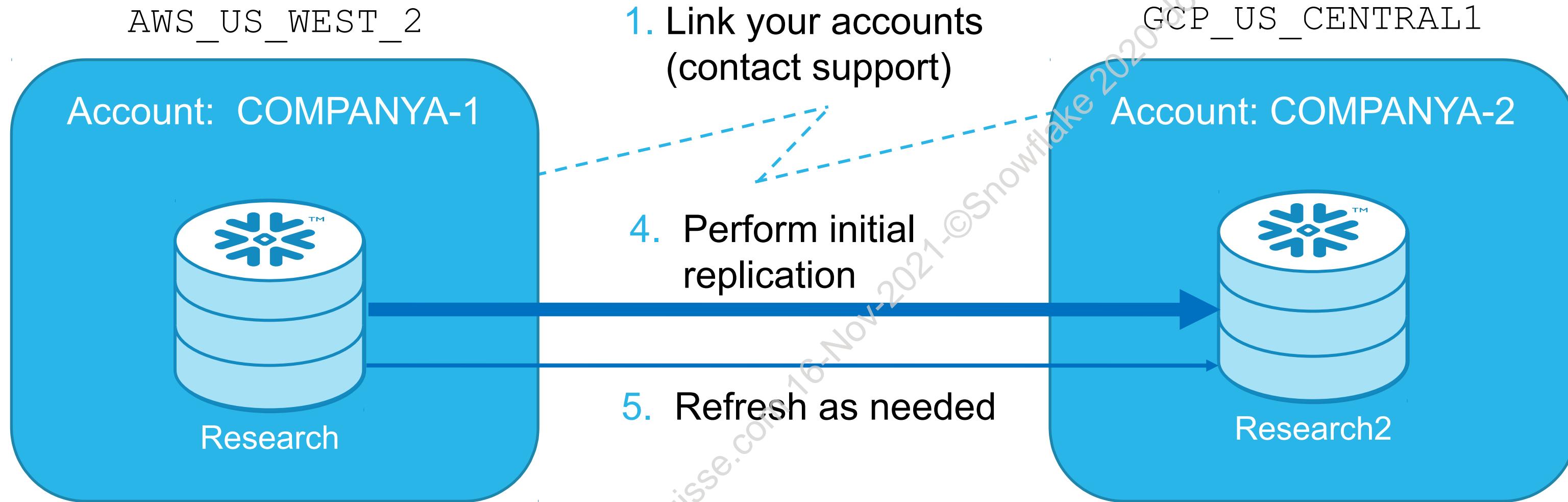


SET UP DATABASE REPLICATION



```
ALTER DATABASE research2 REFRESH;
```

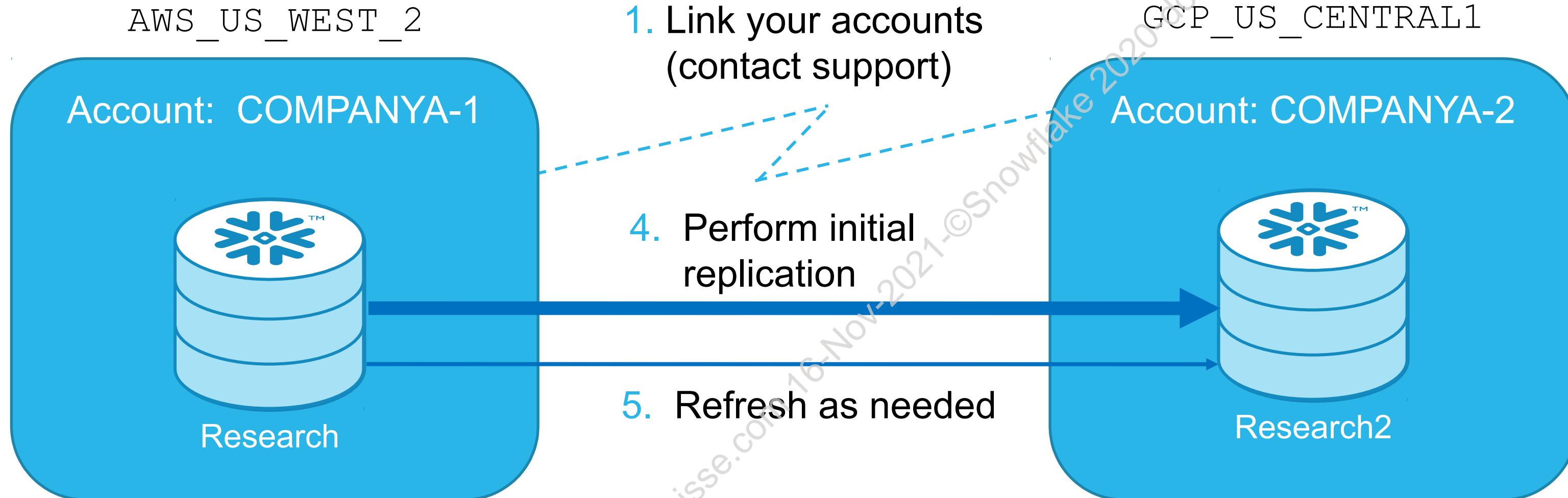
SET UP DATABASE REPLICATION



ALTER DATABASE research2 **REFRESH**;



SET UP DATABASE REPLICATION



2. Enable replication on primary database
6. `ALTER DATABASE <name>
ENABLE FAILOVER TO ACCOUNT <list>`



REFRESH THE SECONDARY DATABASE

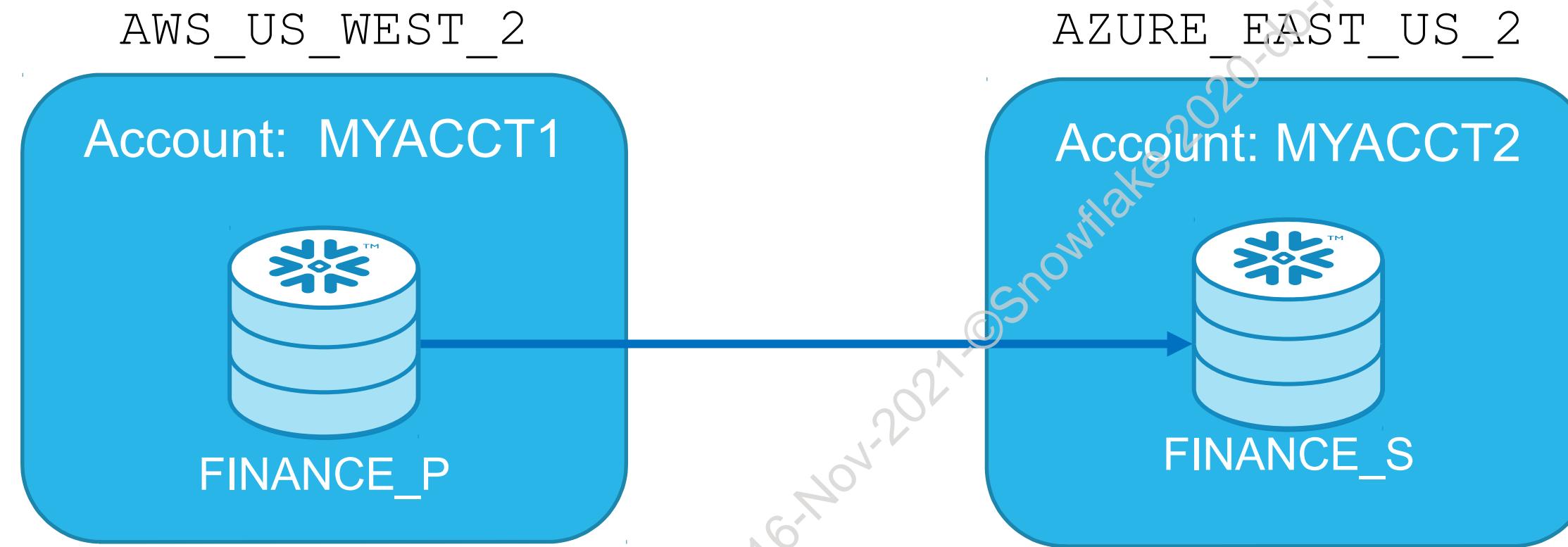
- Refresh manually, or on a schedule using tasks



```
CREATE OR REPLACE TASK refresh_db  
WAREHOUSE = <warehouse name>  
SCHEDULE = <schedule>  
AS  
ALTER DATABASE <name> REFRESH;
```



VERIFY REPLICATION RELATIONSHIP



SHOW REPLICATION DATABASES;

SNOWFLAKE_REGION	ACCOUNT_NAME	NAME	IS_PRIMARY	PRIMARY
AWS_US_WEST_2	MYACCT1	FINANCE_P	Y	ASW_US_WEST_w.MYACCT1.FINANCE_P
AZURE_EAST_US_2	MYACCT2	FINANCE_S	N	ASW_US_WEST_w.MYACCT1.FINANCE_P



HOW OFTEN TO REFRESH?

- Refresh rate depends on your RPO (Recovery Point Objective)

T_1 Secondary database is refreshed

T_2 Changes are happening on the primary database

T_3 Outage on primary database

Any changes made to the primary database during this time will NOT be reflected in the secondary database



SOME REPLICATION CONSIDERATIONS

- May need to adjust STATEMENT_TIMEOUT_IN_SECONDS
 - Use a dedicated warehouse, and set STATEMENT_TIMEOUT_IN_SECONDS on that warehouse
 - Default is 2 days



SOME REPLICATION CONSIDERATIONS

- May need to adjust `STATEMENT_TIMEOUT_IN_SECONDS`
 - Use a dedicated warehouse, and set `STATEMENT_TIMEOUT_IN_SECONDS` on that warehouse
 - Default is 2 days
- When a materialized view is replicated, the view **definition** (not the result) is replicated
 - Set `AUTO_REFRESH_MATERIALIZED_VIEWS_ON_SECONDARY = TRUE` on secondary database to maintain materialized view data



SOME REPLICATION CONSIDERATIONS

- May need to adjust STATEMENT_TIMEOUT_IN_SECONDS
 - Use a dedicated warehouse, and set STATEMENT_TIMEOUT_IN_SECONDS on that warehouse
 - Default is 2 days
- When a materialized view is replicated, the view definition (not the result) is replicated
 - Set AUTO_REFRESH_MATERIALIZED_VIEWS_ON_SECONDARY = TRUE on secondary database to maintain materialized view data
- Cloned objects are replicated physically (copied), rather than logically



SOME REPLICATION CONSIDERATIONS

- May need to adjust `STATEMENT_TIMEOUT_IN_SECONDS`
 - Use a dedicated warehouse, and set `STATEMENT_TIMEOUT_IN_SECONDS` on that warehouse
 - Default is 2 days
- When a materialized view is replicated, the view **definition** (not the result) is replicated
 - Set `AUTO_REFRESH_MATERIALIZED_VIEWS_ON_SECONDARY = TRUE` on secondary database to maintain materialized view data
- Cloned objects are replicated physically (copied), rather than logically
- Time travel queries may return different results on the primary and secondary databases



SOME REPLICATION CONSIDERATIONS

- May need to adjust `STATEMENT_TIMEOUT_IN_SECONDS`
 - Use a dedicated warehouse, and set `STATEMENT_TIMEOUT_IN_SECONDS` on that warehouse
 - Default is 2 days
- When a materialized view is replicated, the view **definition** (not the result) is replicated
 - Set `AUTO_REFRESH_MATERIALIZED_VIEWS_ON_SECONDARY = TRUE` on secondary database to maintain materialized view data
- Cloned objects are replicated physically (copied), rather than logically
- Time travel queries may return ~~different~~ results on the primary and secondary databases
- Replication refreshes incur egress and compute charges



FAILOVER AND FAILBACK

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



DATABASE FAILOVER

BUSINESS CRITICAL EDITION OR ABOVE

- Promote secondary database to primary
 - Secondary database becomes read/write primary
 - Primary database becomes read-only secondary

ALTER DATABASE RESEARCH2 **PRIMARY**

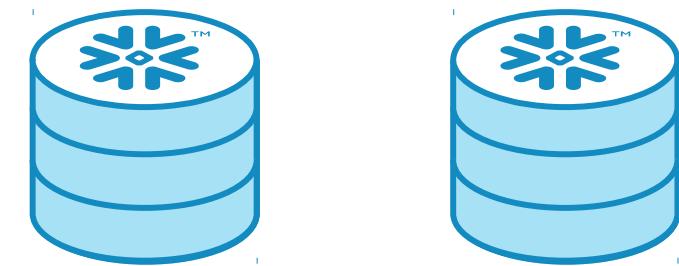


RE-ESTABLISH THE ORIGINAL PRIMARY

BUSINESS CRITICAL EDITION OR ABOVE

Be aware of differences between primary and secondary!

T₁ Secondary database is refreshed

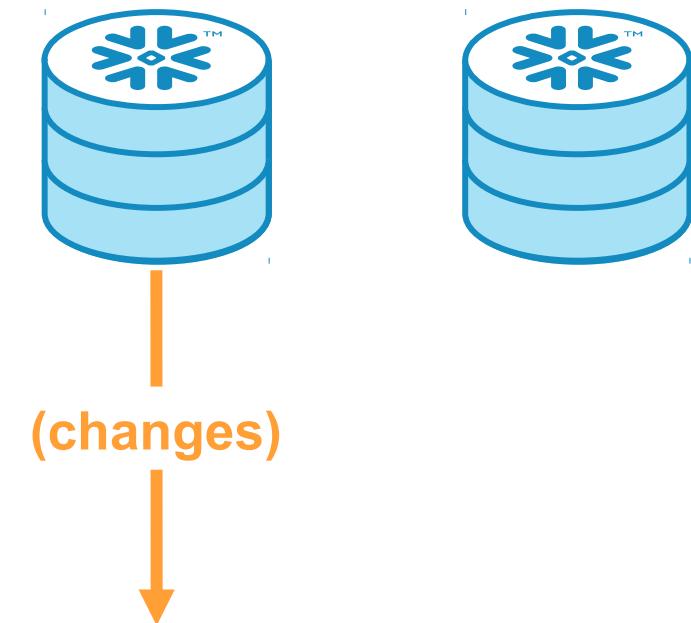


RE-ESTABLISH THE ORIGINAL PRIMARY

BUSINESS CRITICAL EDITION OR ABOVE

Be aware of differences between primary and secondary!

T₁ Secondary database is refreshed



T₂ Changes are happening on the primary database

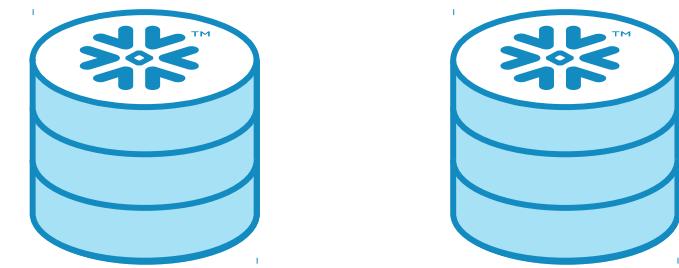


RE-ESTABLISH THE ORIGINAL PRIMARY

BUSINESS CRITICAL EDITION OR ABOVE

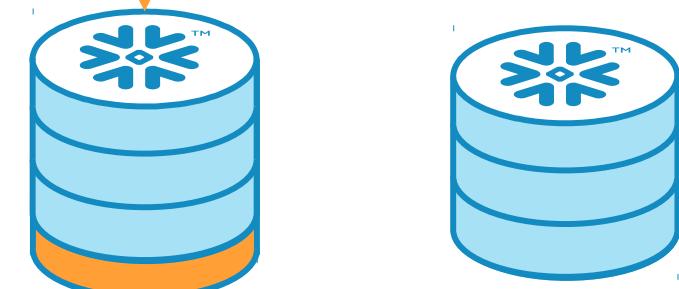
Be aware of differences between primary and secondary!

T₁ Secondary database is refreshed



T₂ Changes are happening on the primary database

(changes)



T₃ Outage on primary database



T₄ Secondary database becomes primary

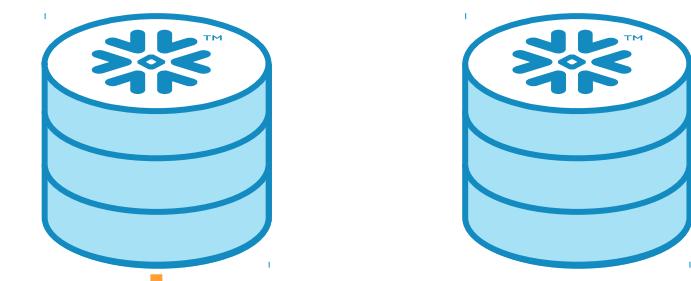


RE-ESTABLISH THE ORIGINAL PRIMARY

BUSINESS CRITICAL EDITION OR ABOVE

Be aware of differences between primary and secondary!

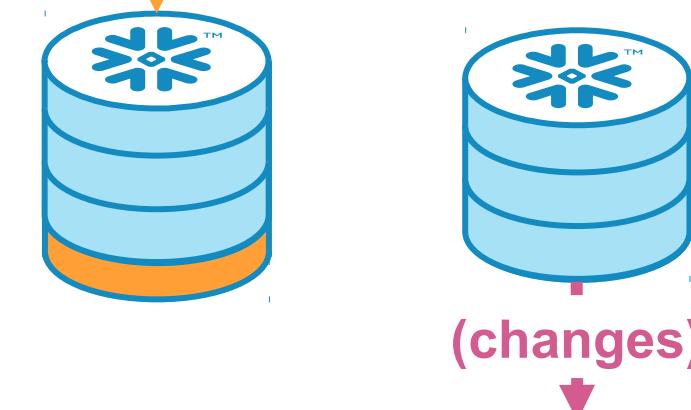
T₁ Secondary database is refreshed



T₂ Changes are happening on the primary database



(changes)



T₃ Outage on primary database



T₄ Secondary database becomes primary



T₅ Changes are happening on secondary (new primary)

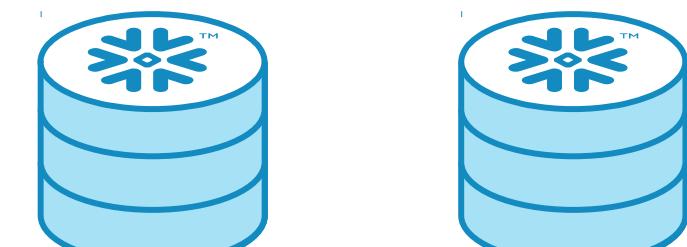


RE-ESTABLISH THE ORIGINAL PRIMARY

BUSINESS CRITICAL EDITION OR ABOVE

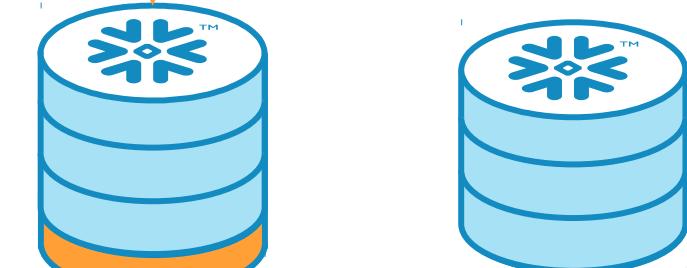
Be aware of differences between primary and secondary!

T₁ Secondary database is refreshed



T₂ Changes are happening on the primary database

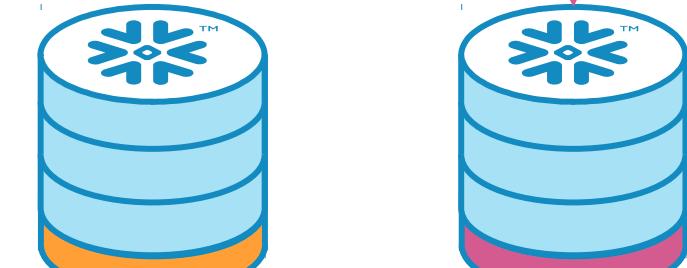
(changes)



T₃ Outage on primary database



T₄ Secondary database becomes primary



T₅ Changes are happening on secondary (new primary)



T₆ Outage on primary database resolved

(changes)



LAB EXERCISE: 19

Database Replication and Disaster Recovery (OPTIONAL)

40 minutes

- Setup Primary a Database to Another Account
- Enable Replication and Failover of Primary Database
- Create Replica of Primary Database
- Monitor Replication Manually
- Schedule Replication Automatically (OPTIONAL)
- Changing Replication Direction

Note: Instructor will grant students ACCOUNTADMIN role for this lab.



MANAGE RESOURCES

anup.vachali@credit-suisse.com 16 Nov 2021 ©Snowflake 2020-do-not-copy



MODULE AGENDA

- Manage Compute
- Manage Storage
- Manage Data Transfer

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



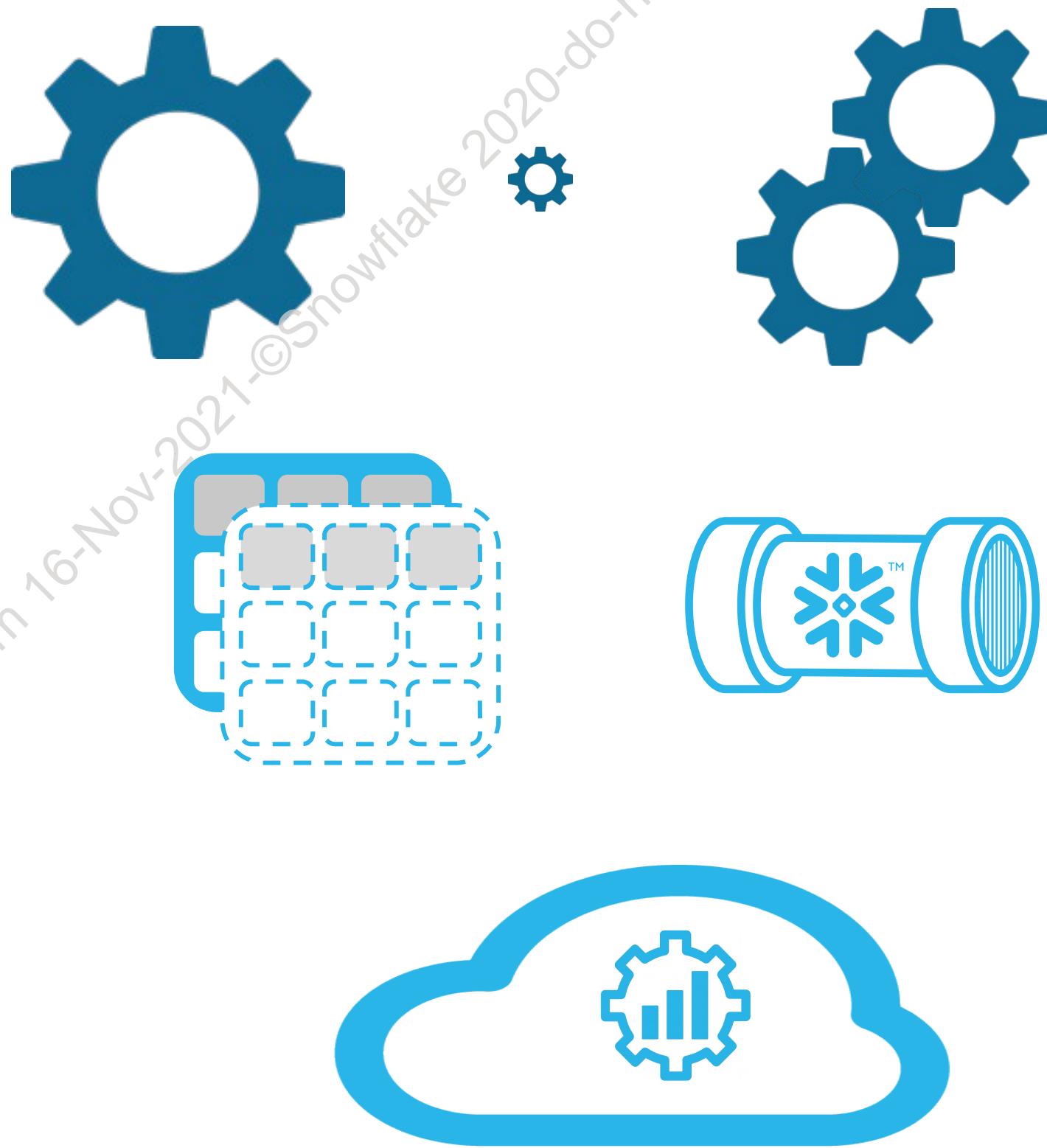
MANAGE COMPUTE

anup.vachali@credit-suisse.com 16-Nov-2021-©Snowflake 2020-do-not-copy



THREE SOURCES OF COMPUTE CHARGES

- Virtual warehouses
 - User-allocated compute
- Serverless features
 - Materialized view maintenance
 - Automated clustering service
 - External table maintenance
 - Database replication
 - Snowpipe
- Cloud compute
 - Cache lookups
 - Other "overhead" operations



VIRTUAL WAREHOUSE COMPUTE

HOW CREDITS ARE CHARGED

- Charged when a virtual warehouse is started up, continues until warehouse is suspended
 - Billed per-second, with a one-minute minimum
- Charges are based on:
 - Number of virtual warehouses
 - How large they are
 - How long they run
- Hourly credits billed for the different sized virtual warehouses:

XS	Small	Medium	Large	X-Large	2X-Large	3X-Large	4X-Large
1	2	4	8	16	32	64	128



QUERY_HISTORY

QUERY_HISTORY	
Columns	Data Type
QUERY_ID	VARCHAR(16777216)
QUERY_TEXT	VARCHAR(16777216)
DATABASE_ID	NUMBER(38,0)
DATABASE_NAME	VARCHAR(16777216)
SCHEMA_ID	NUMBER(38,0)
SCHEMA_NAME	VARCHAR(16777216)
QUERY_TYPE	VARCHAR(16777216)
SESSION_ID	NUMBER(38,0)
USER_NAME	VARCHAR(16777216)
ROLE_NAME	VARCHAR(16777216)
WAREHOUSE_ID	NUMBER(38,0)
WAREHOUSE_NAME	VARCHAR(16777216)
WAREHOUSE_SIZE	VARCHAR(16777216)

- The QUERY_HISTORY view (in SNOWFLAKE.ACCOUNT_USAGE), or the table function of the same name, has a wealth of information for monitoring warehouse efficiency
 - PERCENTAGE_SCANNED_FROM_CACHE
 - PARTITIONS_SCANNED
 - PARTITIONS_TOTAL
 - BYTES_SPILLED_TO_LOCAL_STORAGE
 - BYTES_SPILLED_TO_REMOTE_STORAGE
 - QUEUED_OVERLOAD_TIME



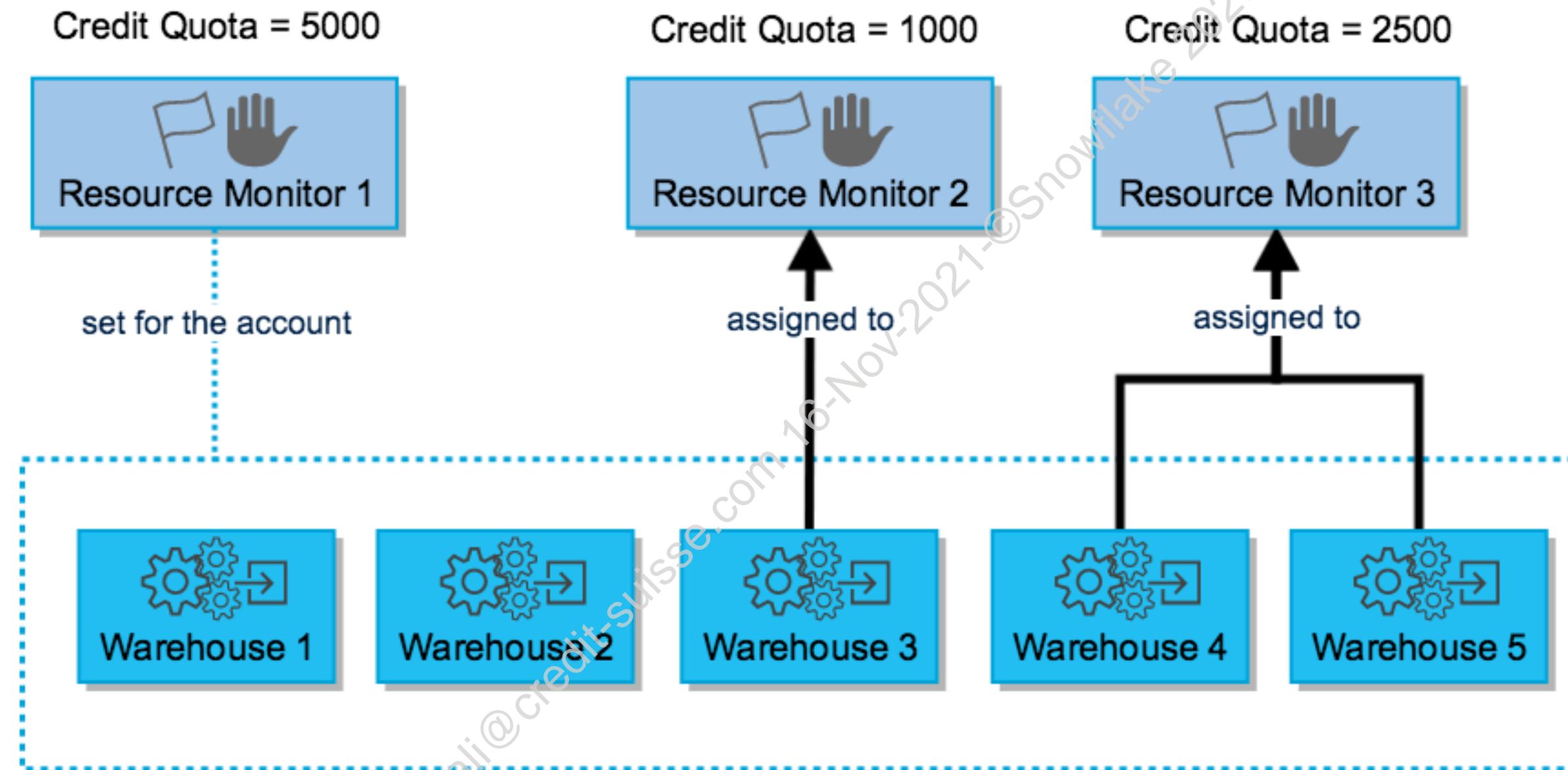
VIRTUAL WAREHOUSE COMPUTE

GENERAL RECOMMENDATIONS

- Auto-suspend virtual warehouses for ETL/ELT and transformation workloads very quickly
 - No benefit from data cache
- Size virtual warehouses for ETL/ELT to maximize parallelism
- Maximize the use of data cache
 - Make sure queries of the same tables share a warehouse
 - For query workloads, set auto-suspend to at least 10 minutes
- Set up resource monitors to control compute usage
- Use multi-cluster warehouses for concurrency
 - Have Snowflake automatically scale out and back



RESOURCE MONITORS



SET UP RESOURCE MONITORS

Edit Resource Monitor

Name My soft limit

Credit Quota 100

Monitor Level ACCOUNT

Schedule Start monitoring on 01.Apr.2019 at 2:00AM and reset monthly.
[Customize](#)

Actions and Notifications
Specify what action to perform when quota is reached.

Suspend and notify when % of credits is used. [?](#)

Suspend immediately and notify when % of credits is used. [?](#)

Notify when % of credits is used. [?](#)

[+Add more notification thresholds](#)

[Show SQL](#)

Customize Schedule

Time Zone Local [▼](#)

Starts Immediately
 Later

5 / 30 / 2019 [▼](#) 11:00PM [▼](#)

Resets Never
 Daily
 Weekly
 Monthly
 Yearly

Ends Never
 On



MANAGE CREDIT CONSUMPTION

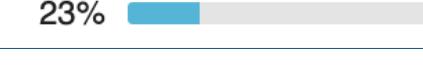
- Control costs and avoid unexpected credit usage
- Set credit quotas for each warehouse and/or the entire account
- Set periodic credit quotas for each team or warehouse
- Act when quotas or pre-defined thresholds are reached

Resource Monitors
Set limits on your account's credit consumption with Resource Monitors. [Learn more](#)

Last refreshed 2:24:37 PM 

 Create Resource Monitor  Edit  Drop

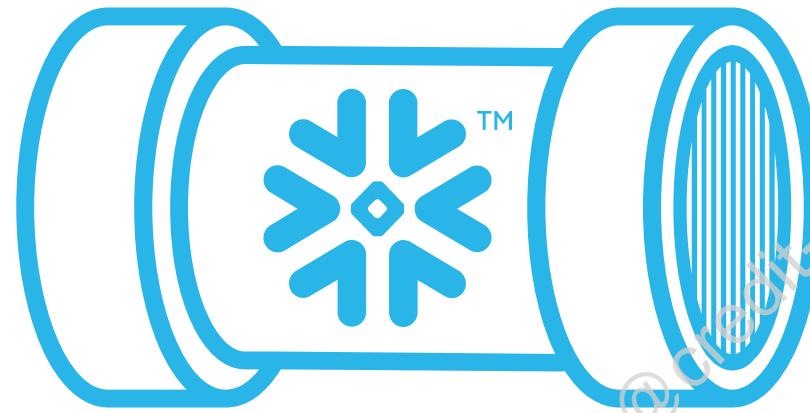
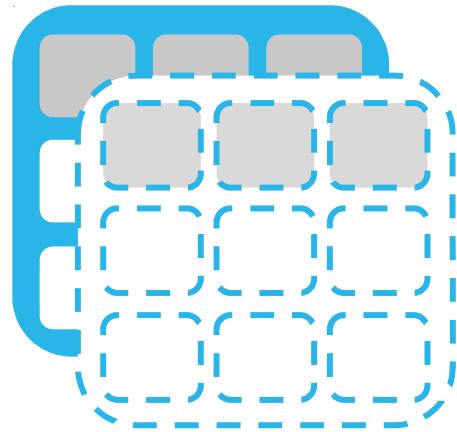
Search Resource Monitors 7 Resource Monitors Columns ▾

Name	↓ Quota Used	Credit Quota	Level	Warehouses	Frequency
MARKETING	91% 	70.00	WAREHOUSE	ALI	NEVER
CK	47% 	1000.00	WAREHOUSE	ALI_MCW	MONTHLY
YBMONITOR2	23% 	500.00	ACCOUNT		MONTHLY



SERVERLESS FEATURES

COMPUTE



- Generally efficient, but should be monitored
- Make sure features like materialized views and clustering keys offset compute costs with performance gains
- Use Snowpipe rather than COPY INTO for continuous loading



CLOUD COMPUTE



- What uses cloud compute?
 - Cache lookups
 - Overhead like filtering files to load
 - Creating and dropping clones
 - Other overhead
- Only charged for cloud compute if it exceeds 10% of overall compute usage
 - Including cloud, serverless, and virtual warehouses



MANAGE STORAGE

anup.vachali@credit-suisse.com 16-Nov-2021-©Snowflake 2020-do-not-copy

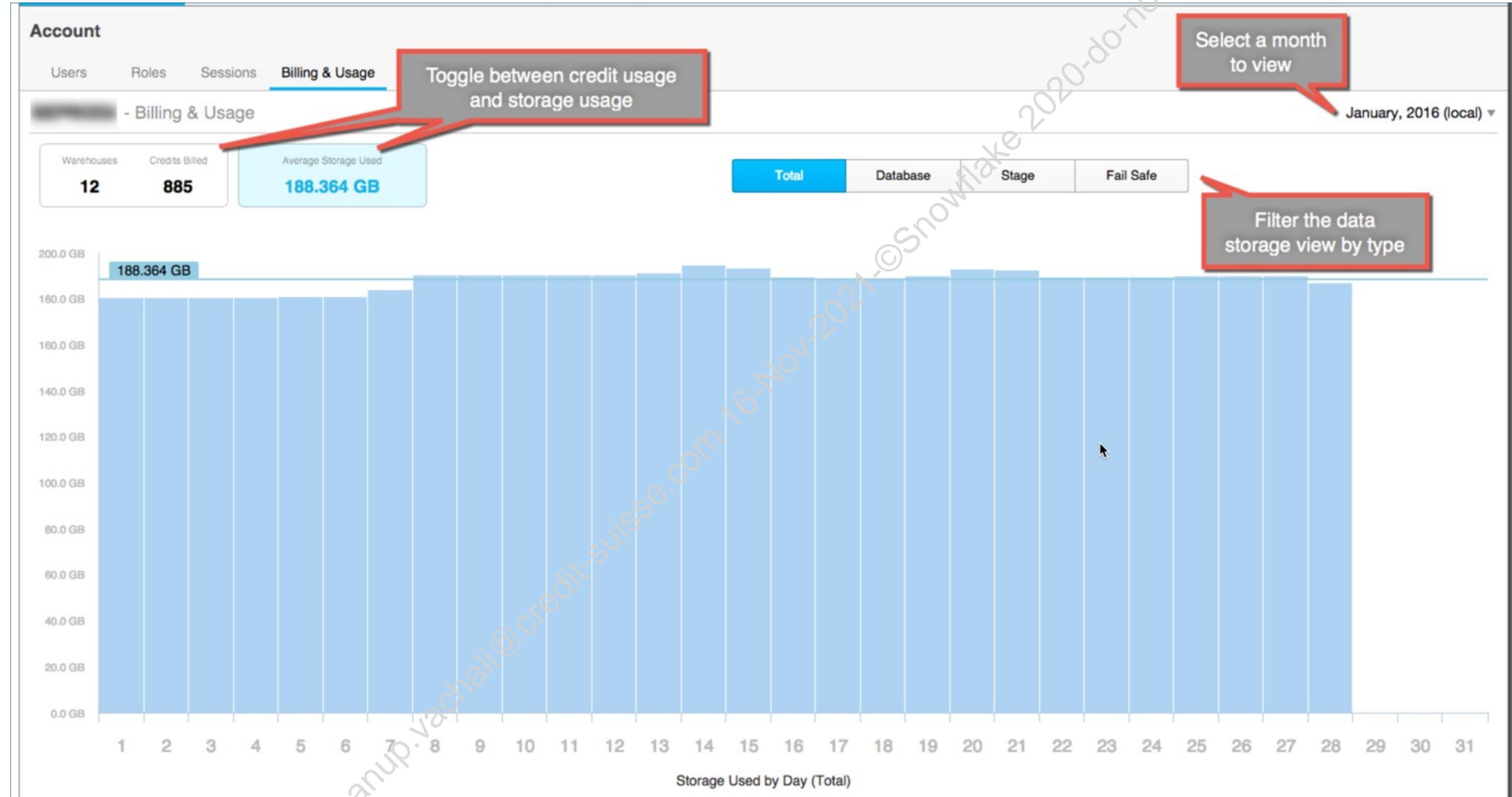


DATA STORAGE COSTS

- Monthly cost is based on a flat rate per TB
 - Daily average
- Cost varies depending on
 - Type of account (capacity or on-demand)
 - Cloud provider (AWS, Azure or GCP)
 - Region (US, EU or APAC)
- See website for pricing details



MONITORING STORAGE



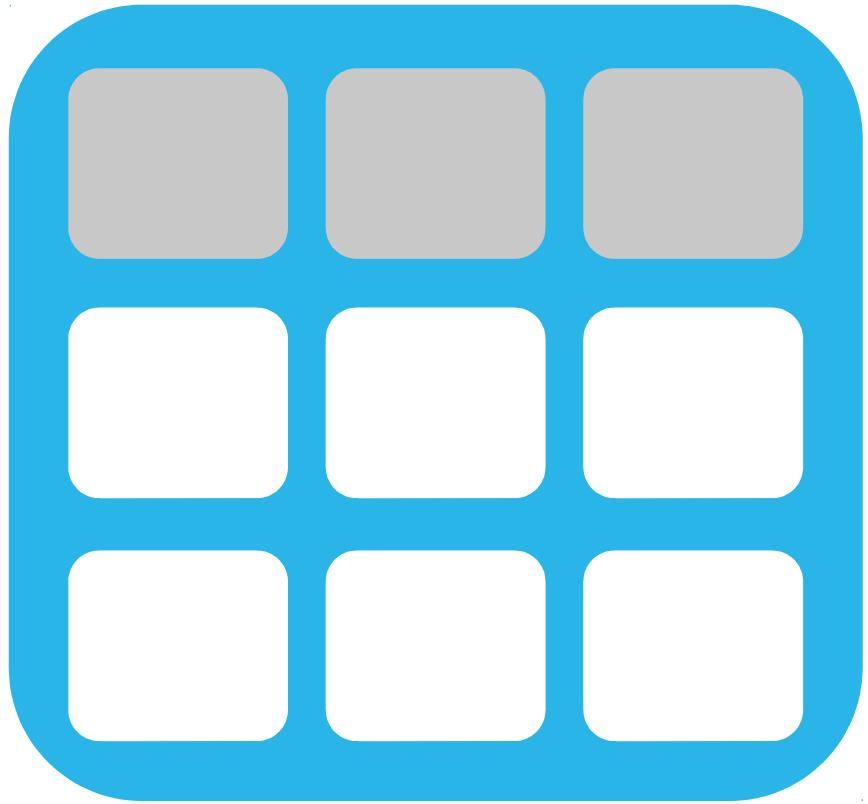
STORAGE USAGE FOR AN ACCOUNT

Available through:

- Snowflake Web Interface, Account -> Billing & Usage
 - Must be ACCOUNTADMIN
- INFORMATION_SCHEMA
 - DATABASE_STORAGE_USAGE_HISTORY
 - STAGE_STORAGE_USAGE_HISTORY
- SNOWFLAKE.ACCOUNT_USAGE
 - STORAGE_USAGE



STORAGE USAGE FOR A TABLE



Available through:

- Snowflake Web Interface
 - Databases -> DB Name -> Tables
- SHOW TABLES
- INFORMATION_SCHEMA
 - TABLE_STORAGE_METRICS
- SNOWFLAKE.ACCOUNT_USAGE
 - STORAGE_USAGE



DATA STORAGE CONSIDERATIONS

- Clean out stages on a regular basis
 - Account is charged for all data stored in tables, schemas, databases, and internal stages
- Manage time travel retention time
 - Storage is charged for data regardless of whether it is in the Active, Time Travel, or Fail-safe state
 - Time travel retention can be set at the account, database, schema, or table level
- For ELT, use temporary/transient tables for staging to save on Fail-safe costs
- Delete transient objects when they are no longer needed



MANAGE DATA TRANSFER

anup.vachali@credit-suisse.com 16Nov2021-©Snowflake 2020-do-not-copy

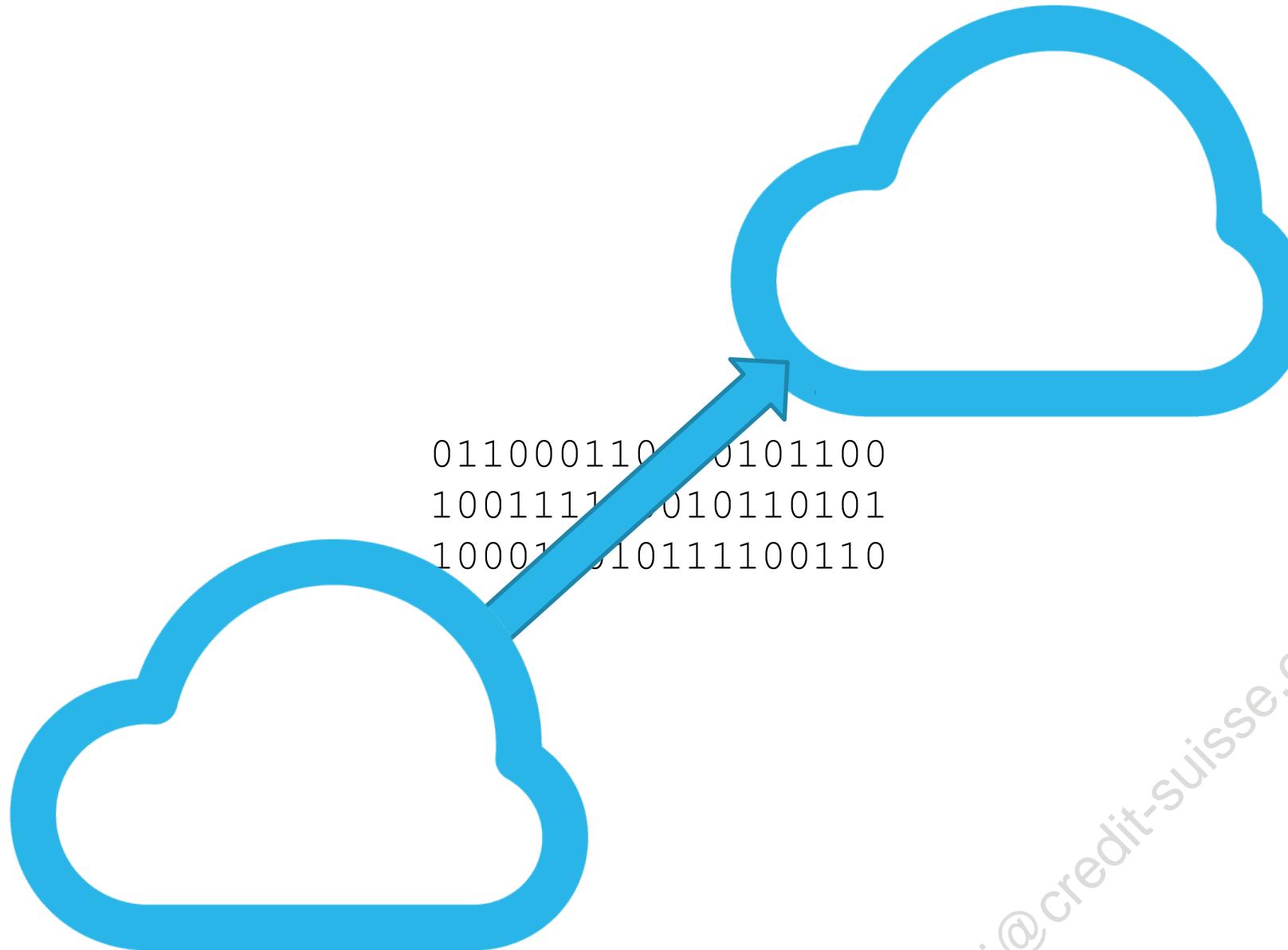


DATA TRANSFER COSTS

- Data transfer charges apply for data unloading or replication:
 - Between regions within the same cloud provider
 - Between clouds (within or across regions)
- Pass-through from cloud provider
 - Costs depend on origin, destination, and provider
- Snowflake does not charge to **load** data from an external stage, but customer's cloud provider might charge a separate egress fee
 - If the stage is located in a different region/on a different cloud from the Snowflake account



MONITOR DATA TRANSFER USAGE



Information available through:

- Snowflake Web Interface, Account -> Billing & Usage
Must be ACCOUNTADMIN
- INFORMATION_SCHEMA
 - DATA_TRANSFER_HISTORY
 - Available for 14 days
- SNOWFLAKE.ACCOUNT_USAGE
 - DATA_TRANSFER_HISTORY
 - Available for 1 year



BUILD A DATA PIPELINE

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy

CLIENTS, CONNECTORS, AND THE ECOSYSTEM

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



MODULE AGENDA

- Ecosystem tools
- SnowSQL
- Python Connector
- Apache Spark Connector

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy

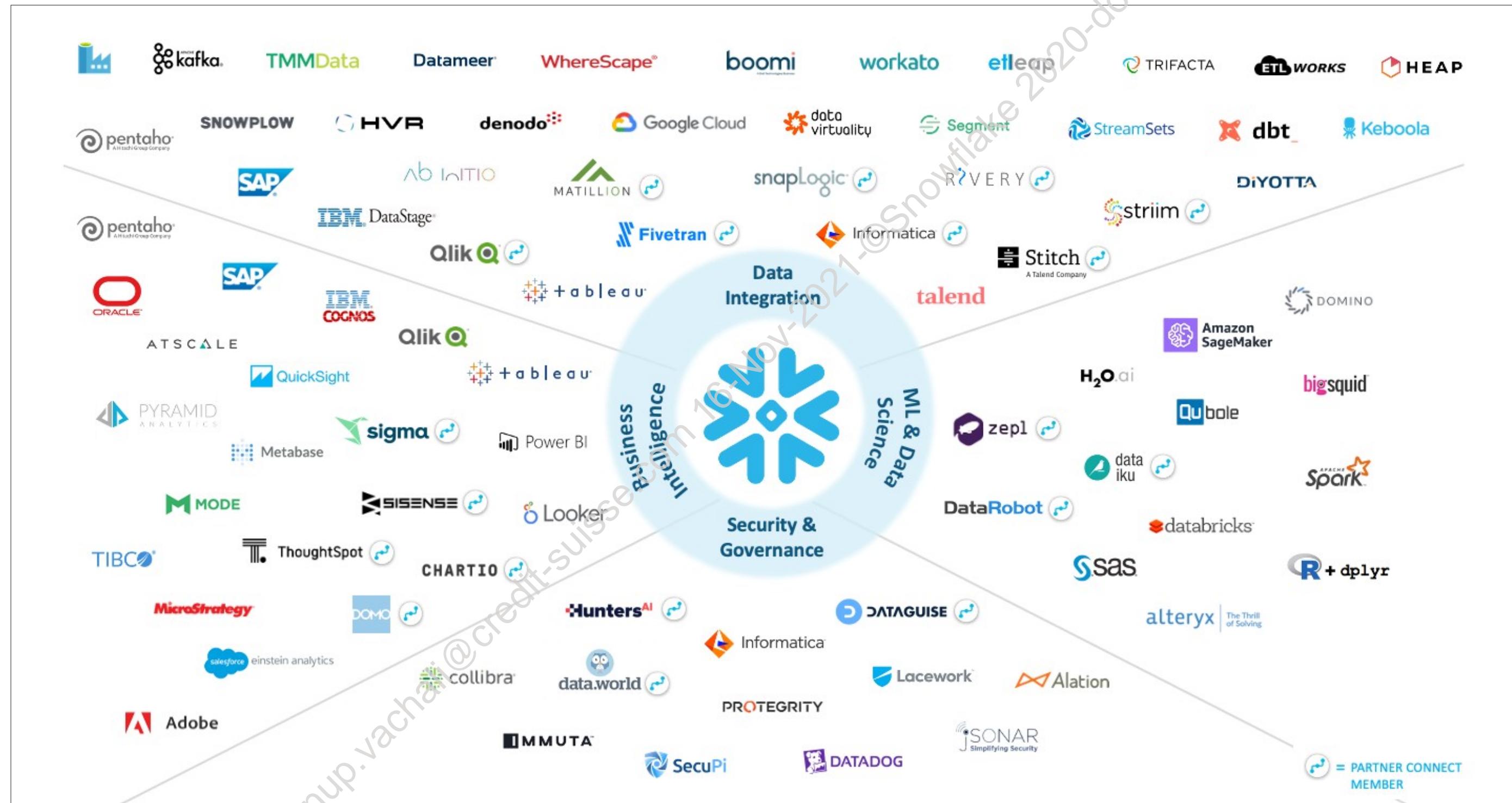


ECOSYSTEM TOOLS

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



SNOWFLAKE ECOSYSTEM



SQL CLIENTS AND CONNECTORS



- Tools (see previous slide) for:
 - Data integration
 - Machine Learning and Data Science
 - Business Intelligence
 - Security and Governance
- SQL Development and Management
 - SQL access to Snowflake
- Native Programmatic Interfaces
 - Connect to Snowflake programmatically

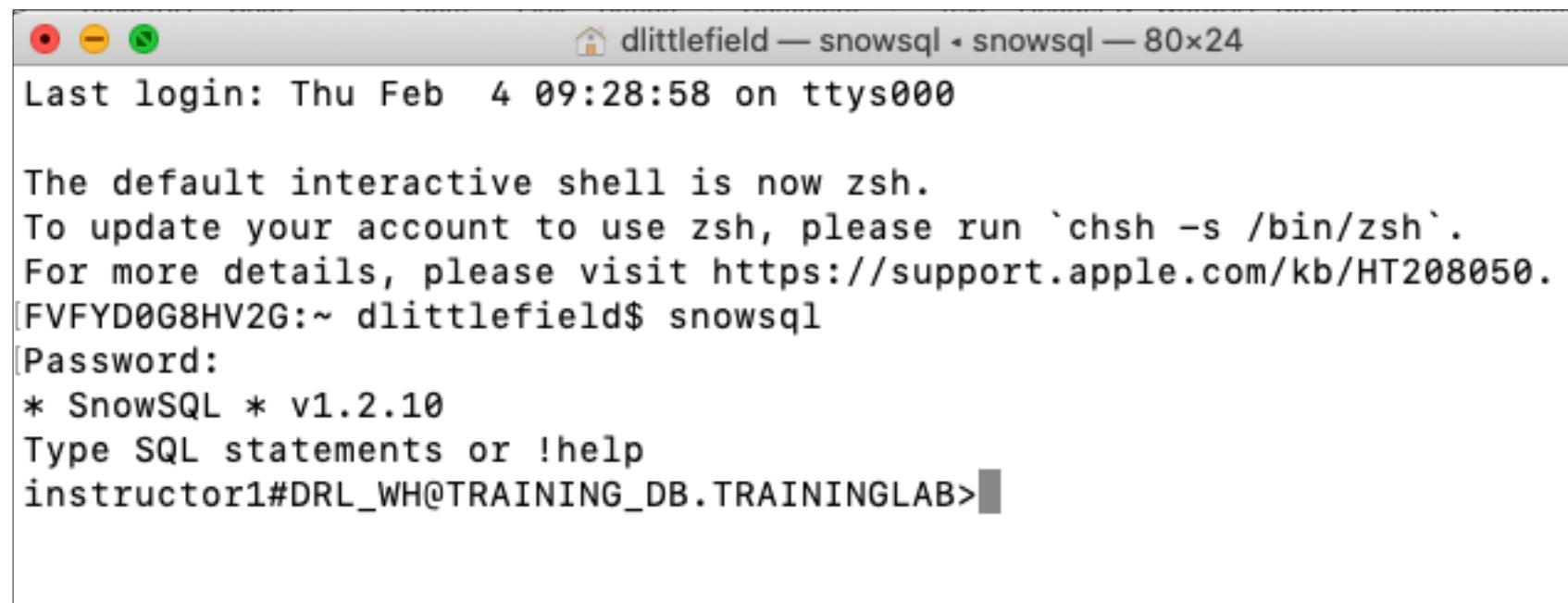
SNOWSQL

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



SNOWSQL

SNOWFLAKE'S CLI



A screenshot of a terminal window titled "dlittlefield — snowsql - snowsql — 80x24". The window shows a login message, system details, and a password prompt. It then displays the SnowSQL version (v1.2.10) and a SQL command being typed.

```
Last login: Thu Feb  4 09:28:58 on ttys000
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
[FVFYD0G8HV2G:~ dlittlefield$ snowsql
Password:
* SnowSQL * v1.2.10
Type SQL statements or !help
instructor1#DRL_WH@TRAINING_DB.TRAININGLAB>
```

```
#If a connection doesn't specify a value, it will default to these
#
#accountname = <account>
#username = <users>
#dbname = <database>
#schemaname = <schema>
#warehousename = <warehouse>
#rolename = <role>
#proxy_host = <defaultproxyhost>
#proxy_port = <defaultproxyport>
```

- Snowflake's command-line client
- Installs hidden directory (.snowsql) in home directory
- Configuration file defines default and alternate connection options



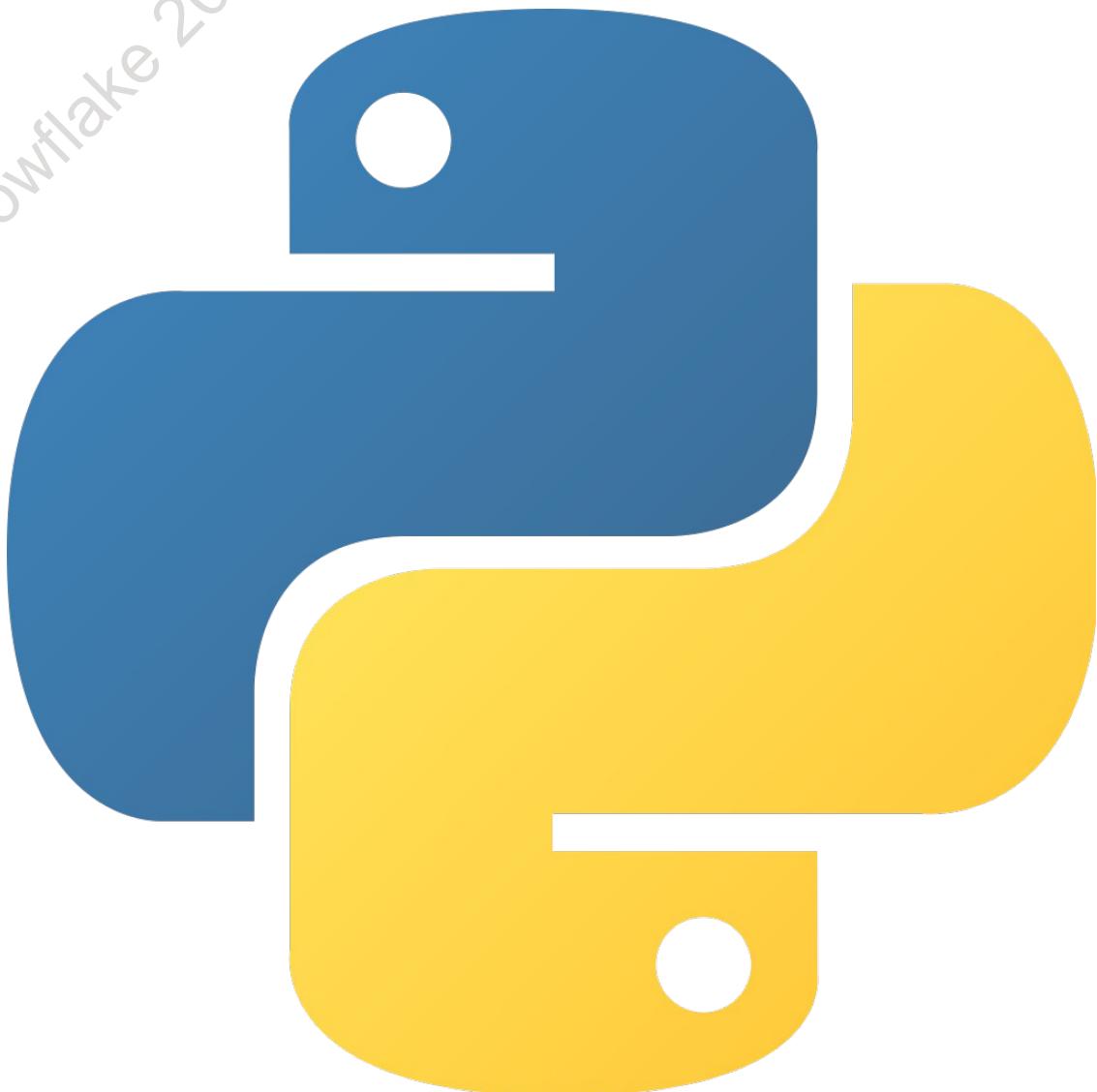
PYTHON CONNECTOR

anup.vachali@credit-suisse.com 16-Nov-2021-©Snowflake 2020-do-not-copy

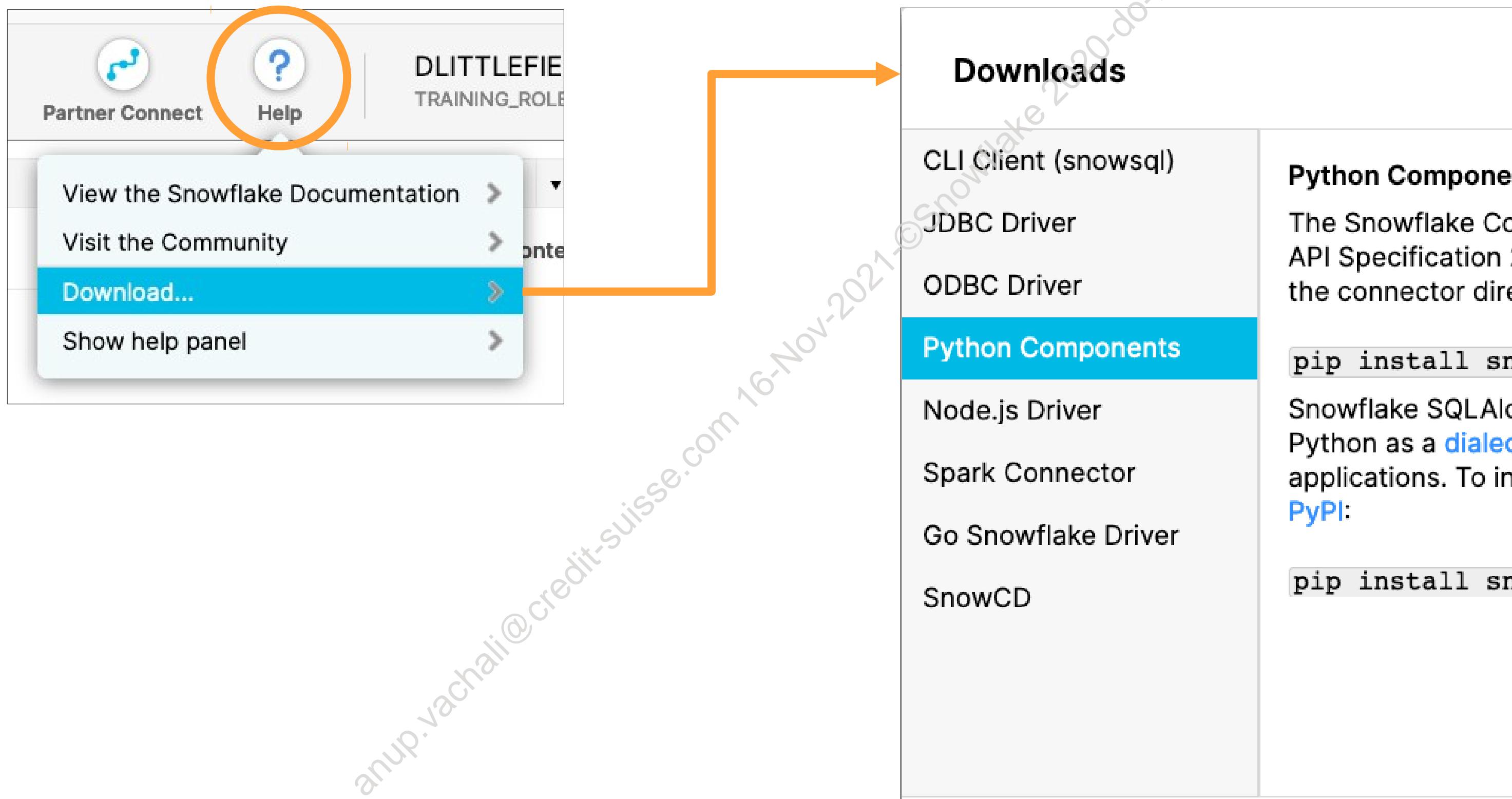


CONNECTOR FOR PYTHON

- Native Python package set up using pip
 - Not dependent on JDBC or ODBC protocols
- Supported in Windows, Mac OS, Linux
 - See documentation for supported versions
- Python API supports:
 - Transaction control
 - Encrypted PUT, GET, and COPY commands
 - Data types DATE and TIMESTAMP
 - Queries, DDL, and DML commands
 - Executing SQL scripts
 - Retrieving column metadata or Snowflake Query IDs



DOWNLOAD THE PYTHON CONNECTOR



APACHE SPARK CONNECTOR



anup.vachali@credit-suisse.com 16 Nov 2021 ©Snowflake 2020-do-not-copy

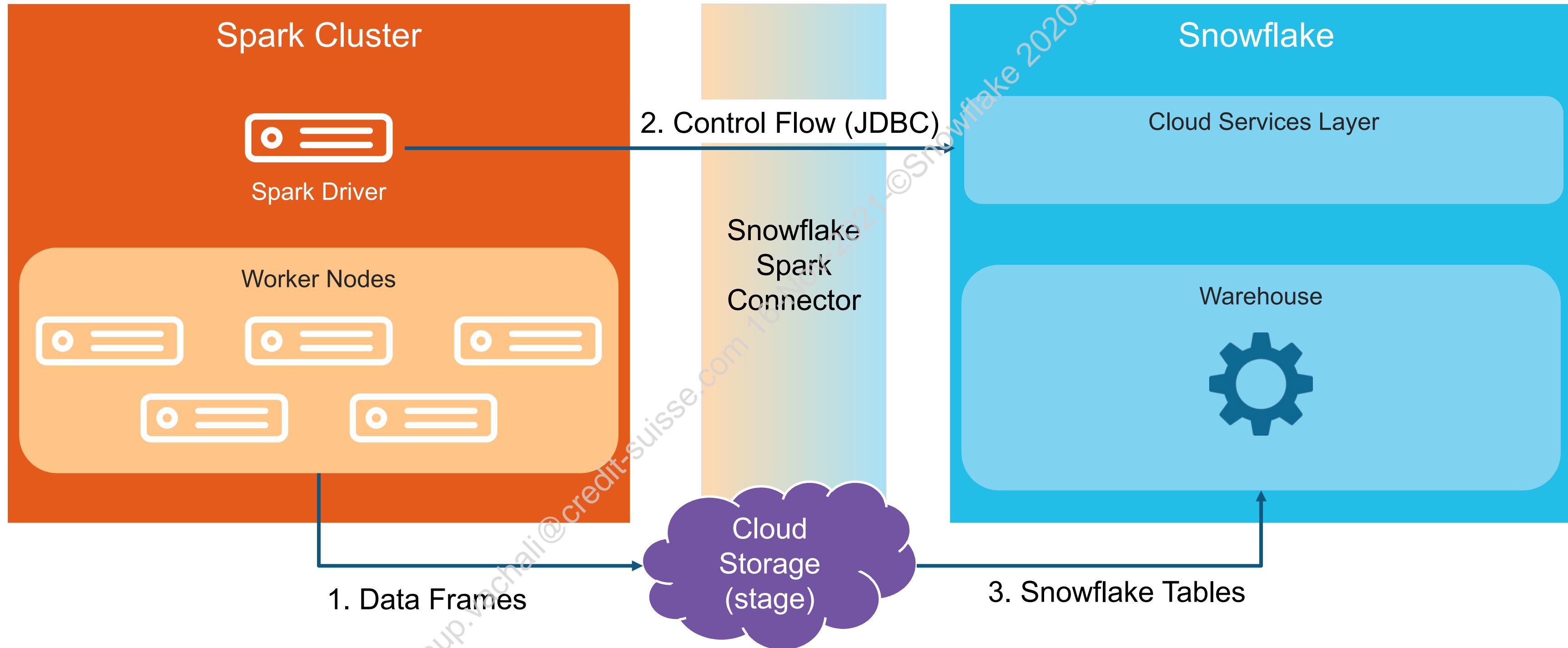
THE SPARK CONNECTOR



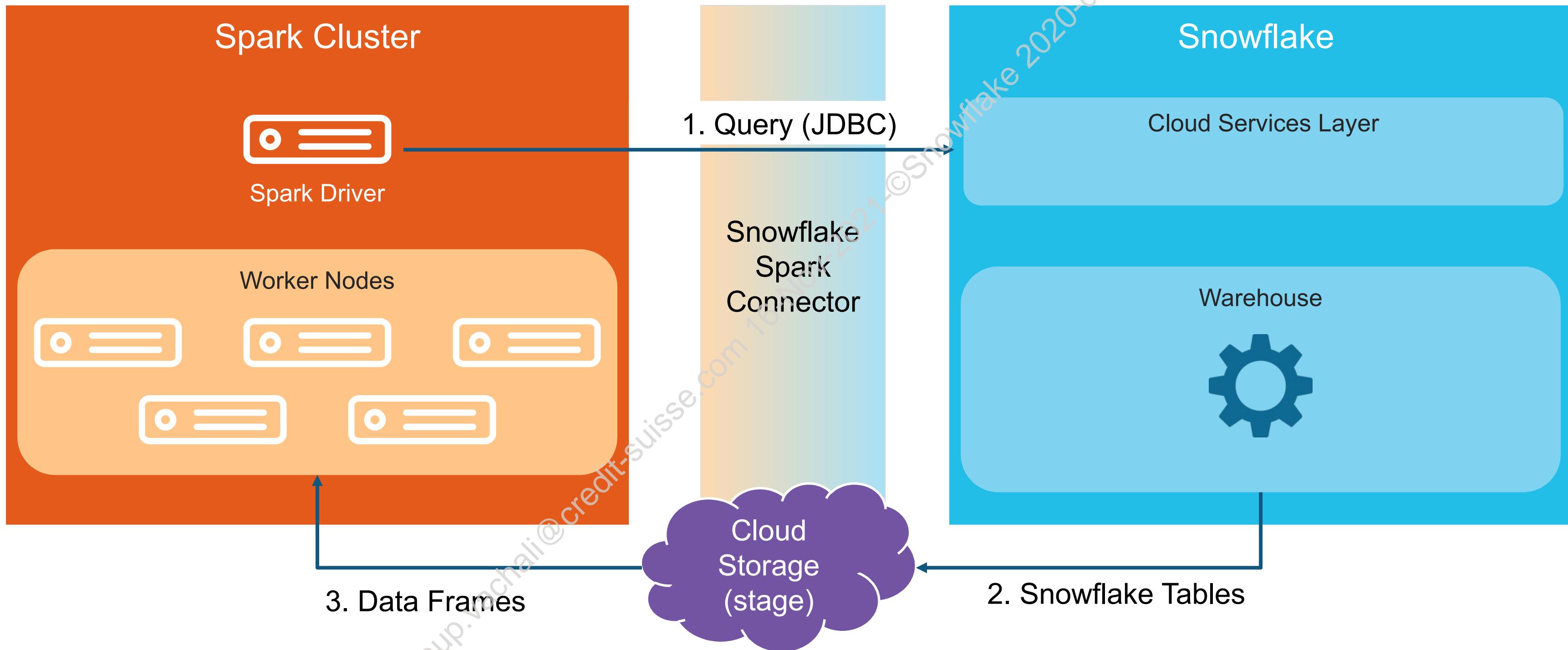
- Allows bi-directional movement of data between a Snowflake cluster and a Spark cluster
- Process data on the Spark cluster, or push processing down to Snowflake
 - Best if processing happens close to the data
- Spark cluster can be self-hosted, or hosted on a 3rd party provider



WRITE FROM SPARK TO SNOWFLAKE



READ FROM SNOWFLAKE TO SPARK



USING STREAMS FOR CHANGE DATA CAPTURE

anup.vachali@credit-suisse.com Nov-2021 ©Snowflake 2020-do-not-copy



MODULE AGENDA

- Overview
- Streams and Offsets
- Using Streams

anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy



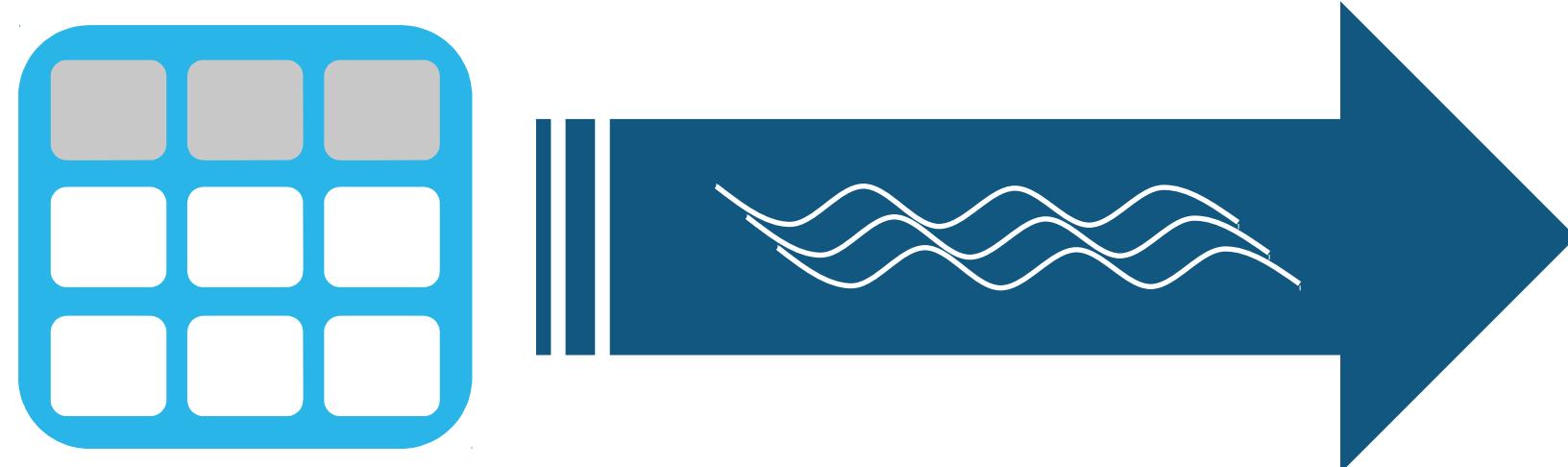
OVERVIEW

anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



STREAMS

CHANGE DATA CAPTURE (CDC)



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

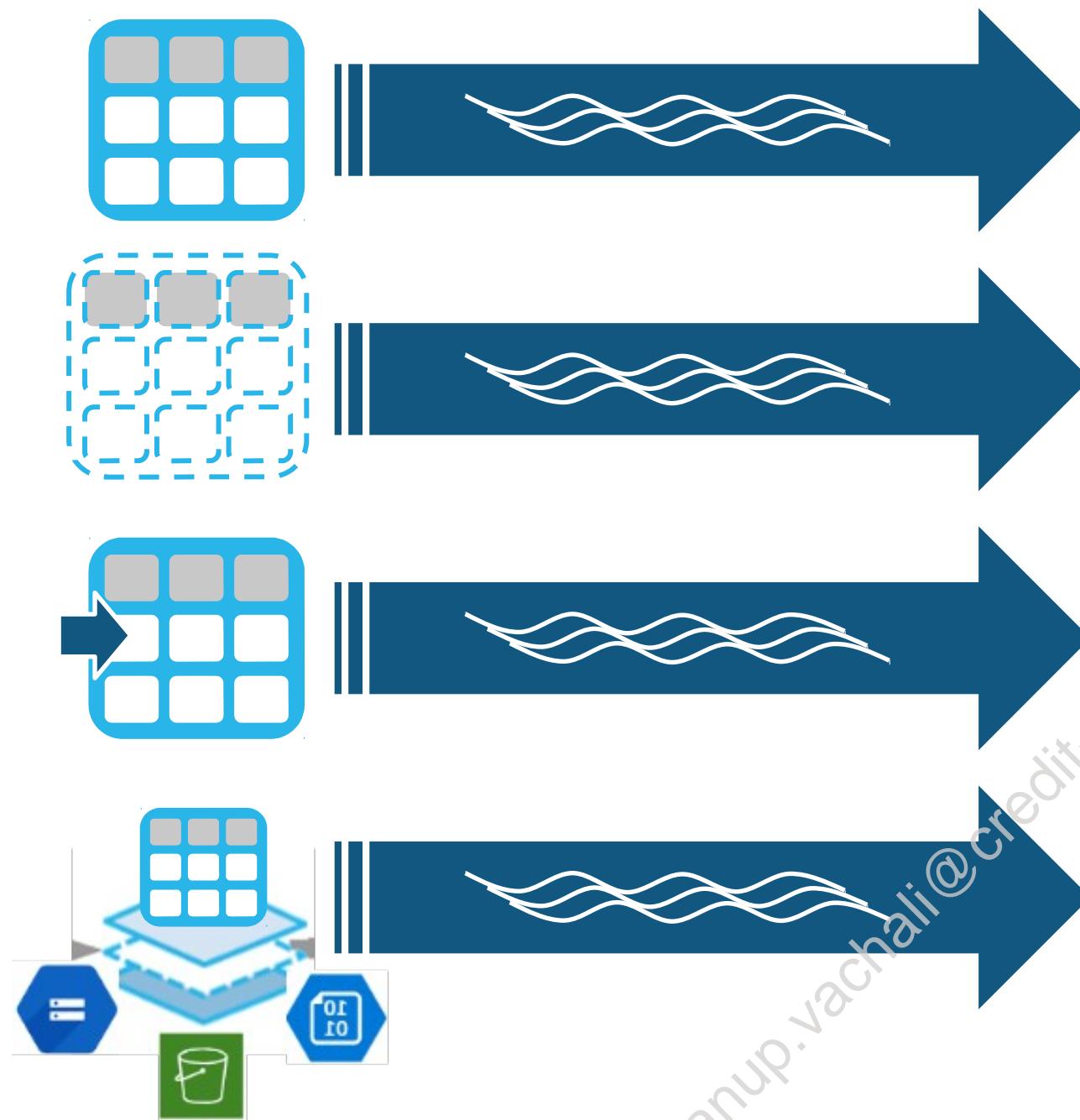
```
CREATE STREAM <stream> ON TABLE <src_table>  
APPEND ONLY = TRUE;
```

- 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 added to:

- Tables
- Views
- Shared Tables
- External Tables

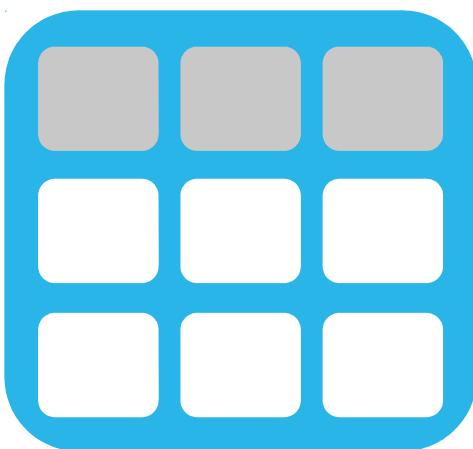


TABLES VS STREAMS

DATA AT REST – DATA IN MOTION

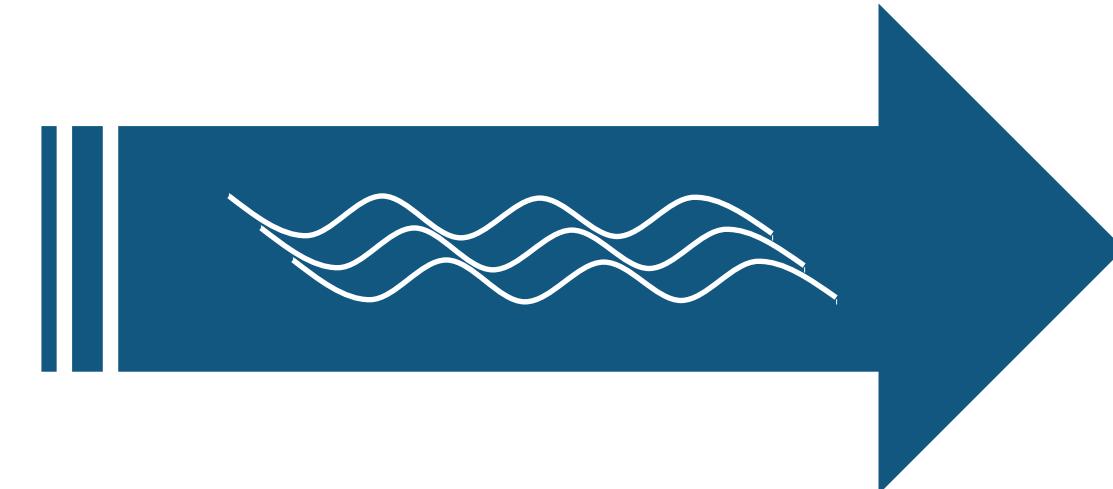
TABLE

- 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

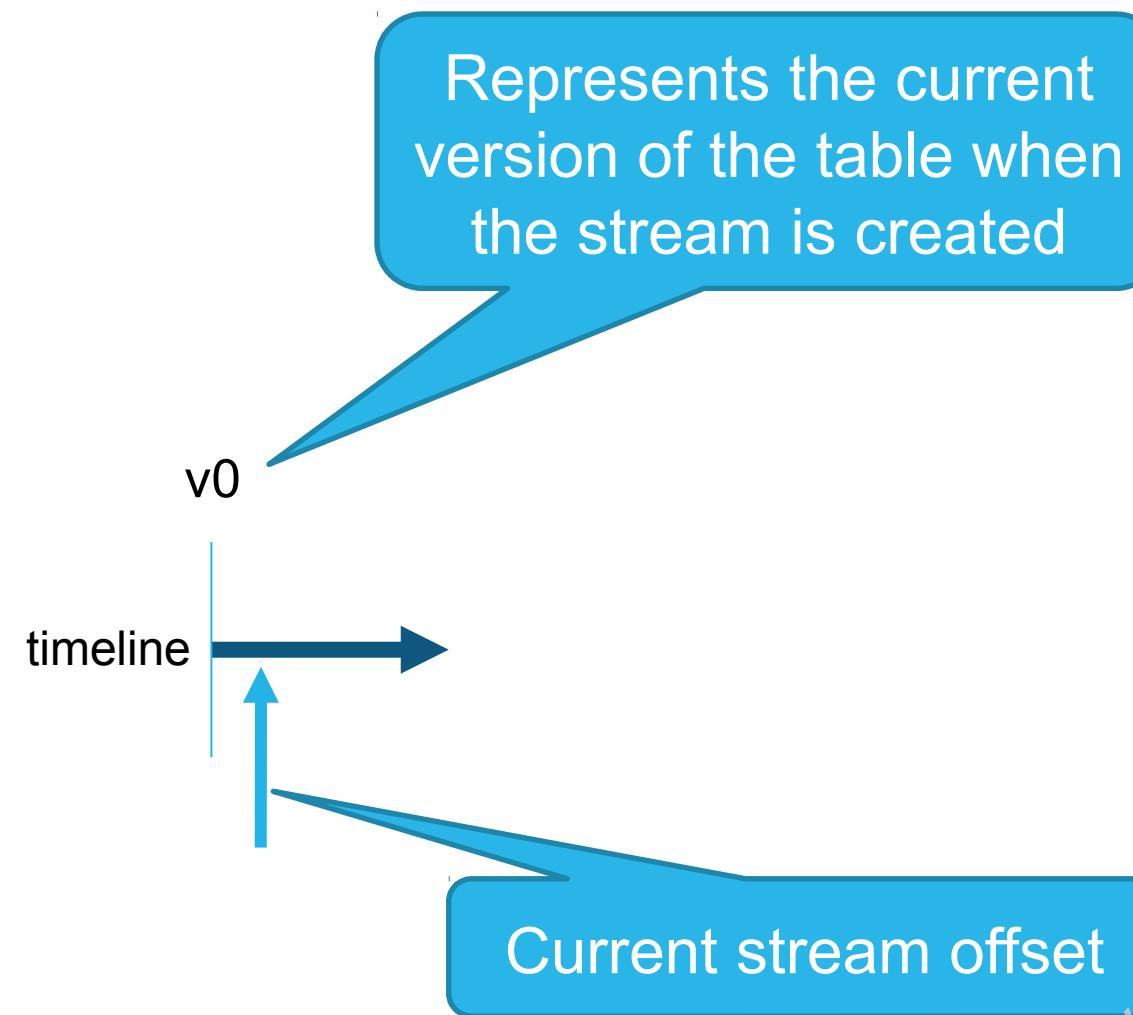
anup.vachali@credit-suisse.com 16 Nov 2021 ©Snowflake 2020-do-not-copy



HOW DOES IT WORK?

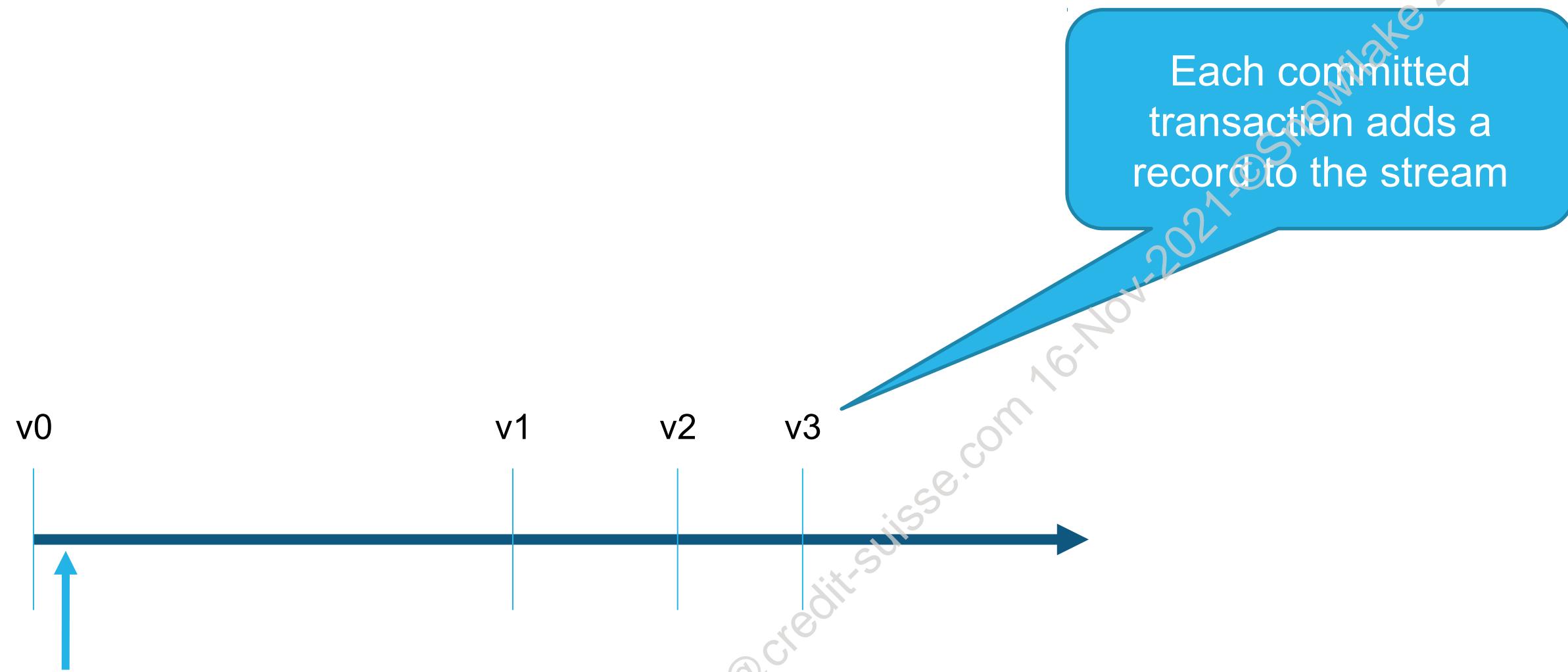
1. Stream is created

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



HOW DOES IT WORK?

- Transactions are committed on the source table

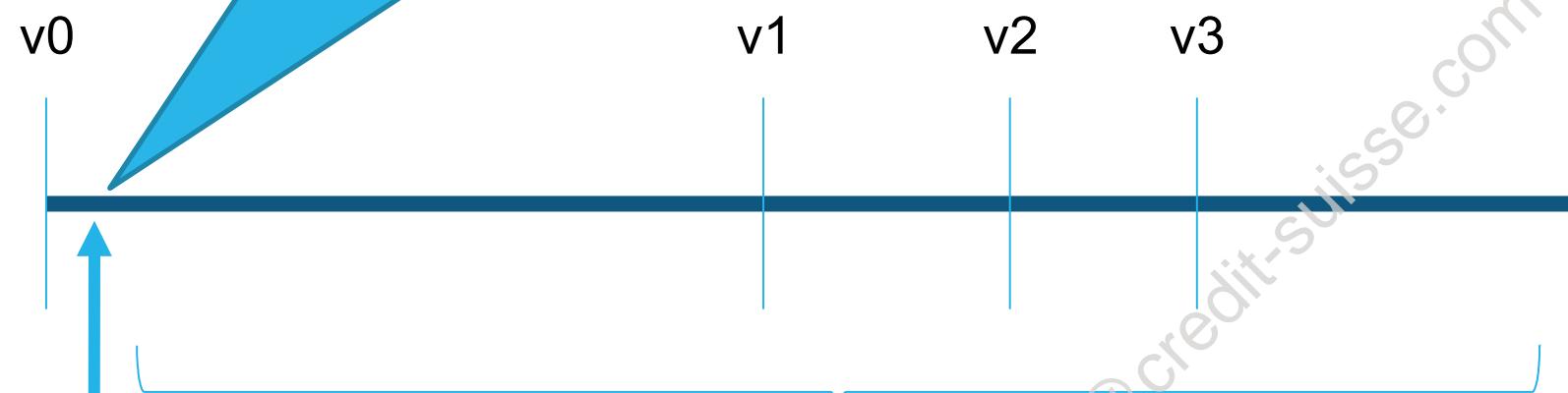


HOW DOES IT WORK?

3. Stream data is queried

```
SELECT * FROM fruit_stream;
```

Querying a stream does not change the offset; information has not been consumed



Stream returns change records after the current offset

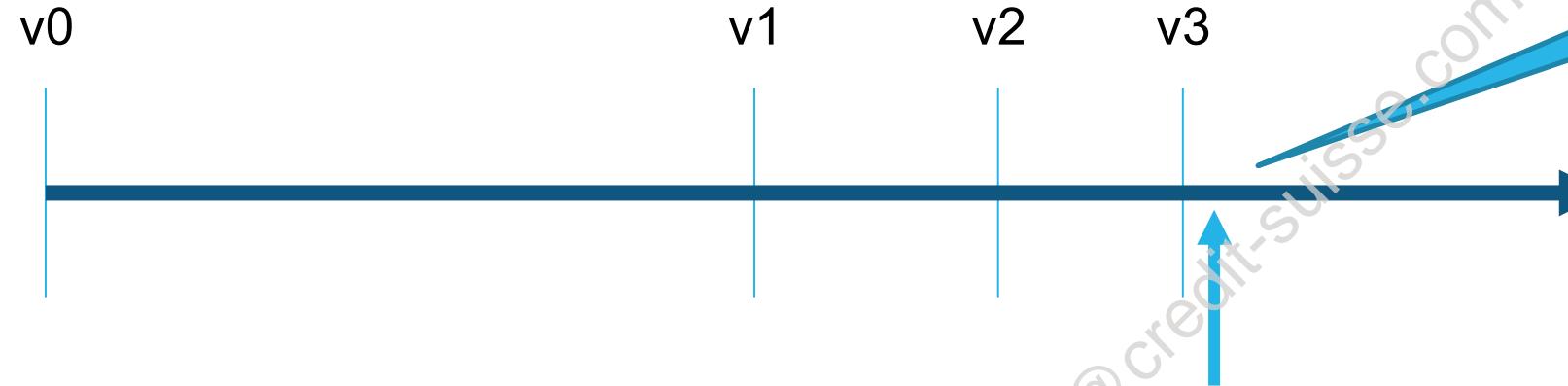


HOW DOES IT WORK?

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

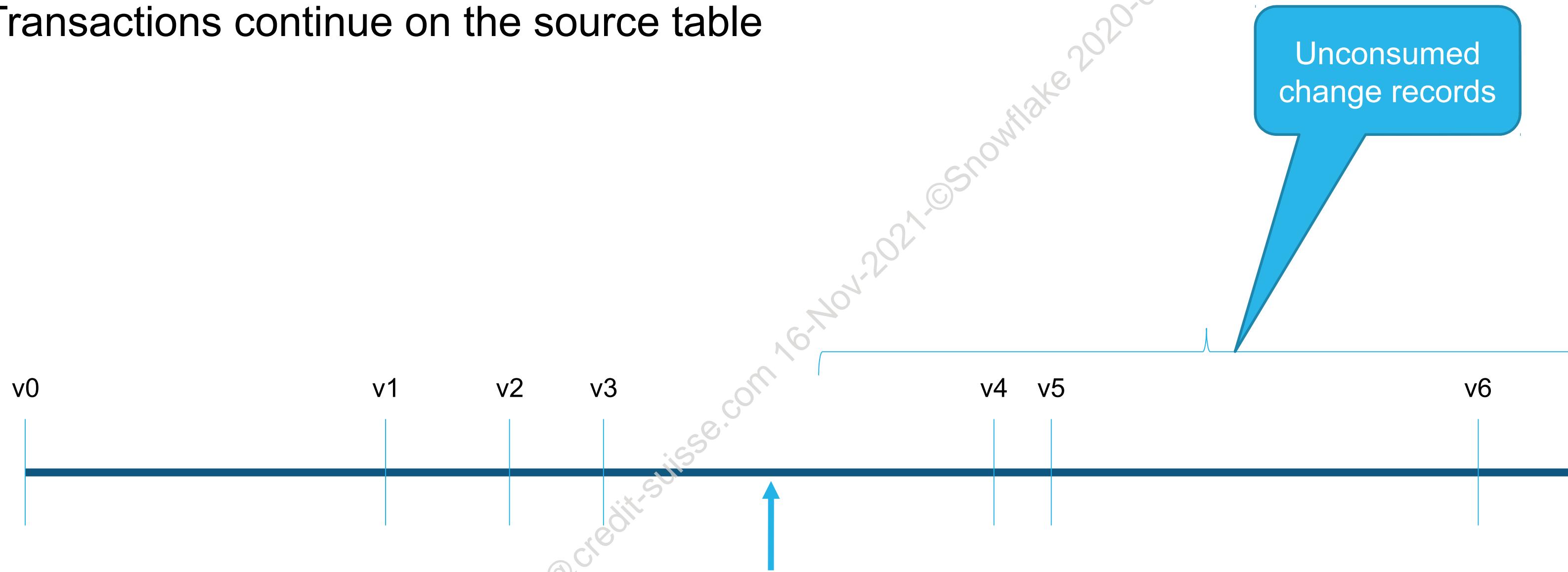
```
INSERT INTO order_history(product_name, quantity)  
SELECT fruit, qty FROM fruit_stream;
```

Consuming the stream
advances the offset



HOW DOES IT WORK

- 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

anup.vachali@credit-suisse.com 16-Nov-2021-©Snowflake 2020-do-not-copy



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

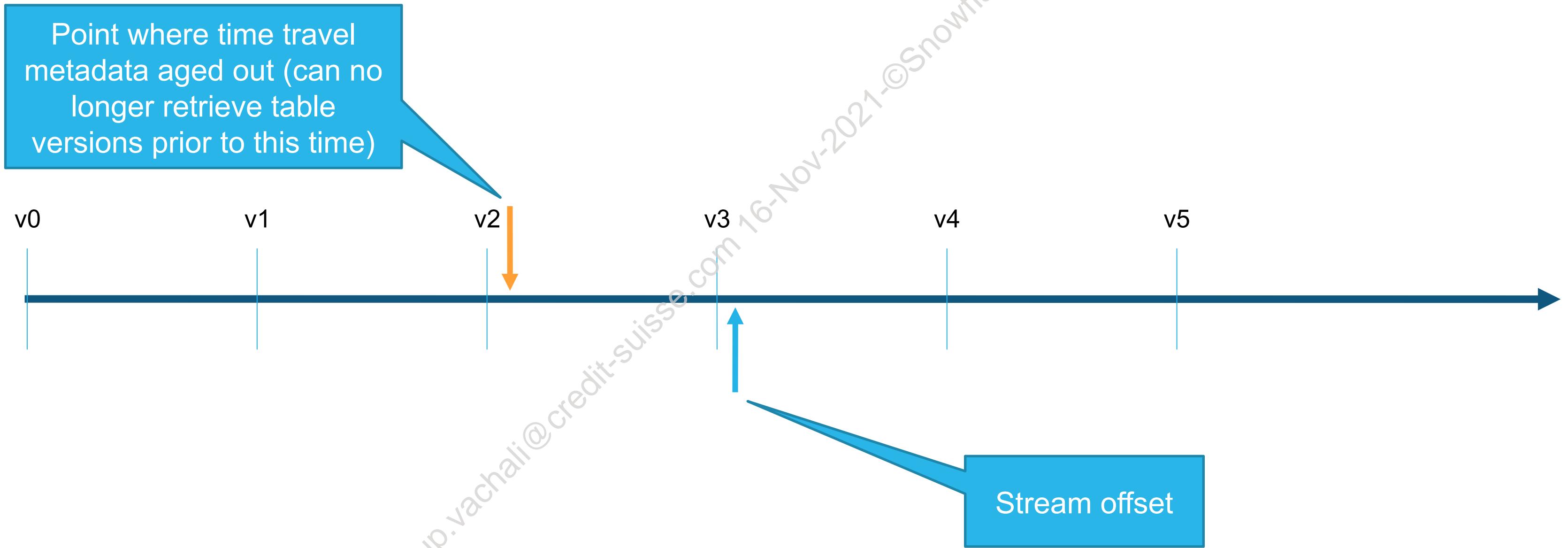
```
SELECT system$stream_get_table_timestamp('standard_stream_streamtest')
```

Row	SYSTEM\$STREAM_GET_TABLE_TIMESTAMP('FRUIT_STREAM')
1	1611242753877



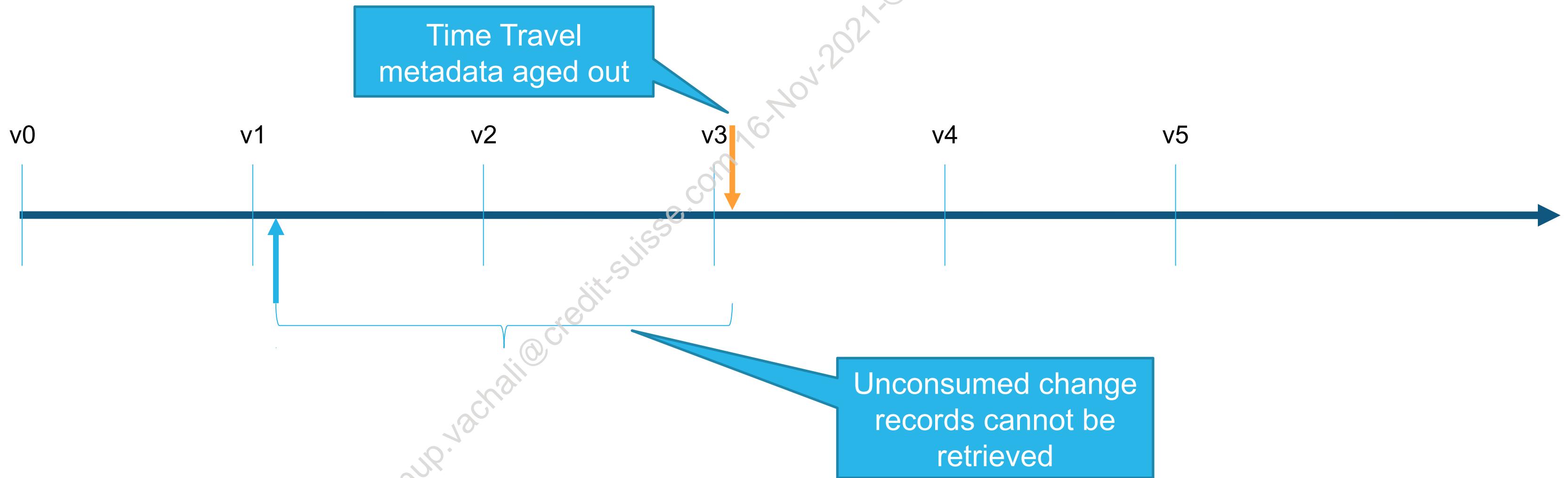
STREAMS AND TIME TRAVEL

- 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

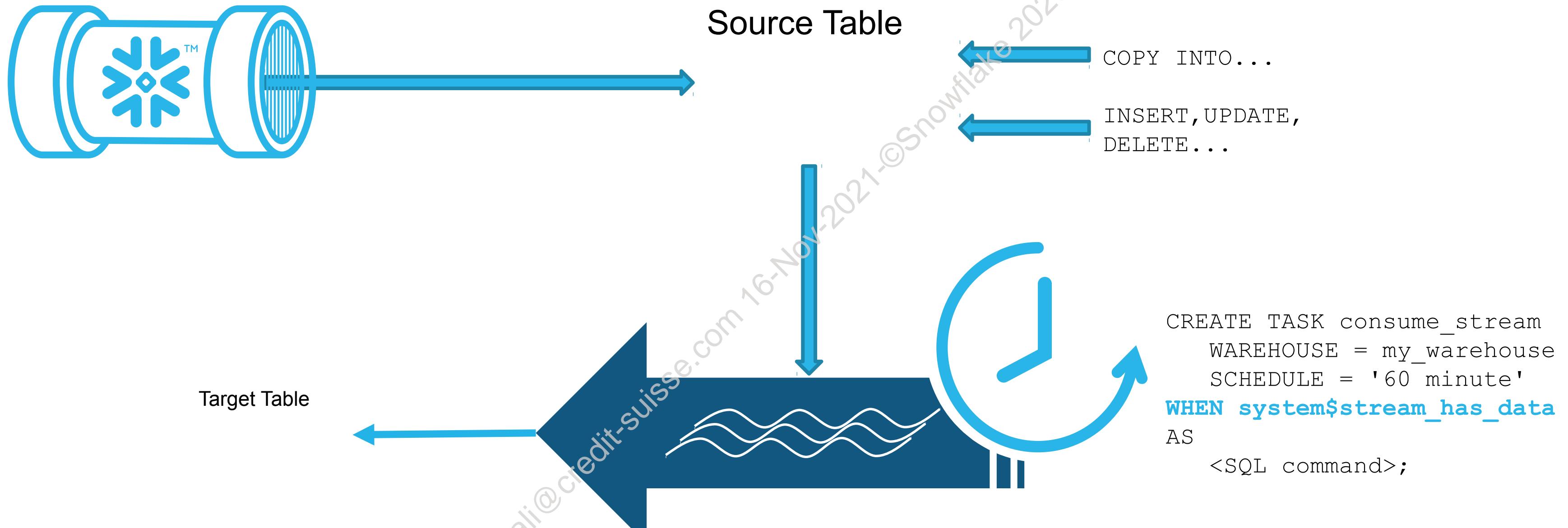


STREAMS AND TIME TRAVEL

- 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



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

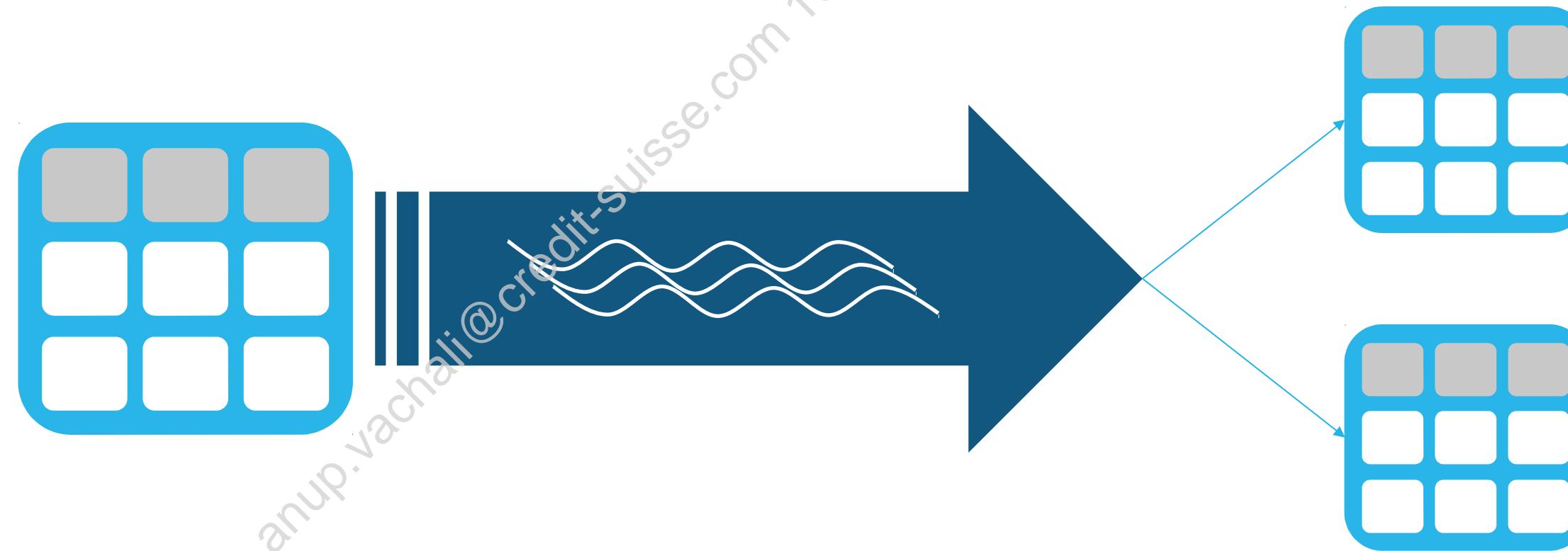
```
BEGIN;
```

```
<update table1 from stream>
```

```
<update table2 from stream>
```

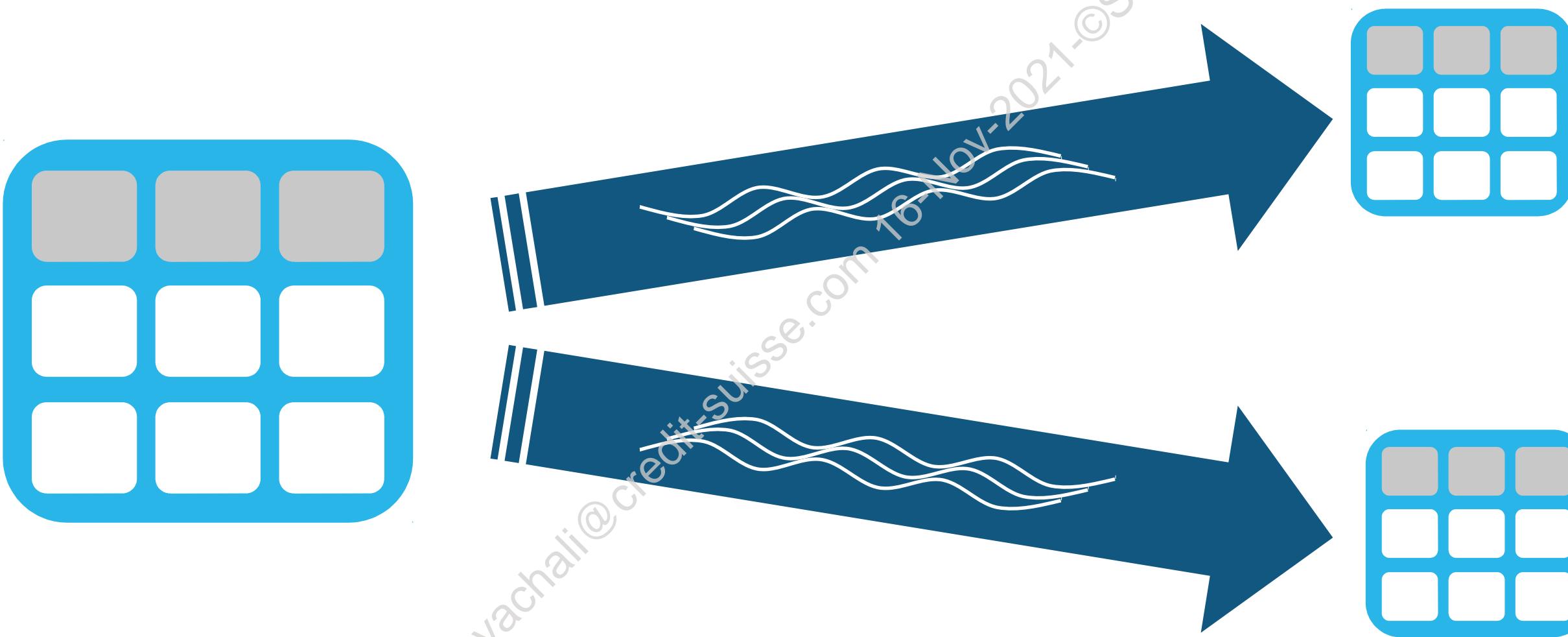
```
...
```

```
COMMIT;
```



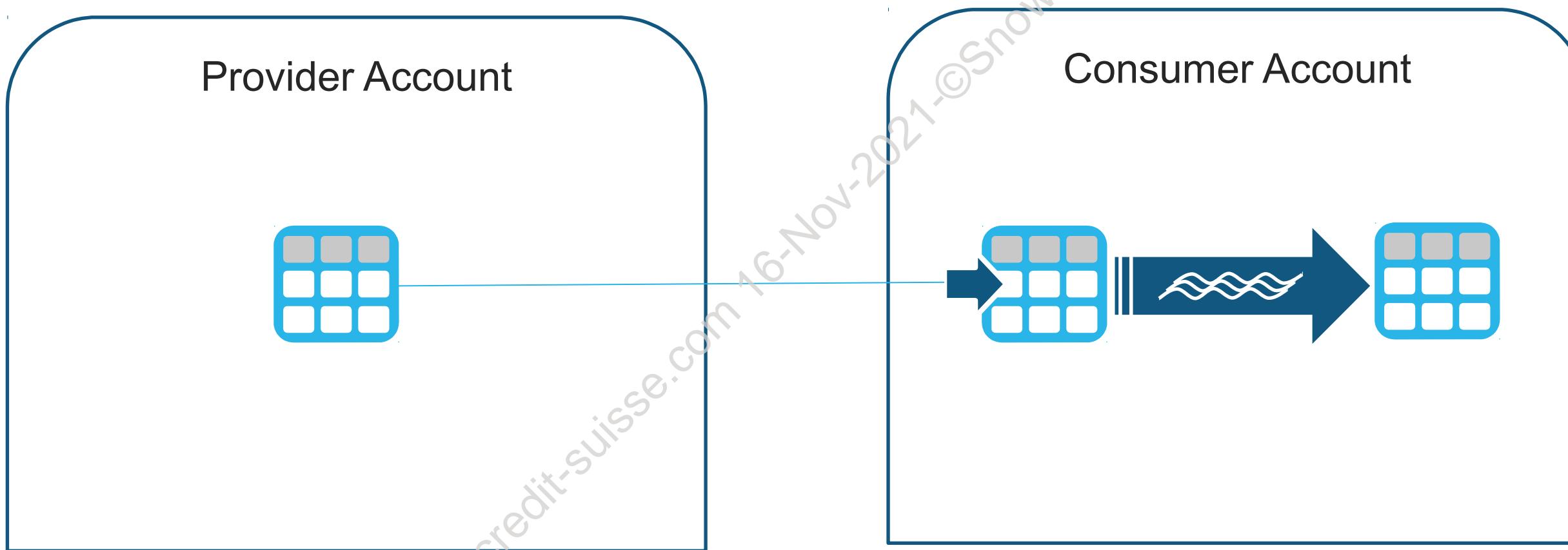
UPDATE MULTIPLE TABLES

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



STREAMS AND SHARES

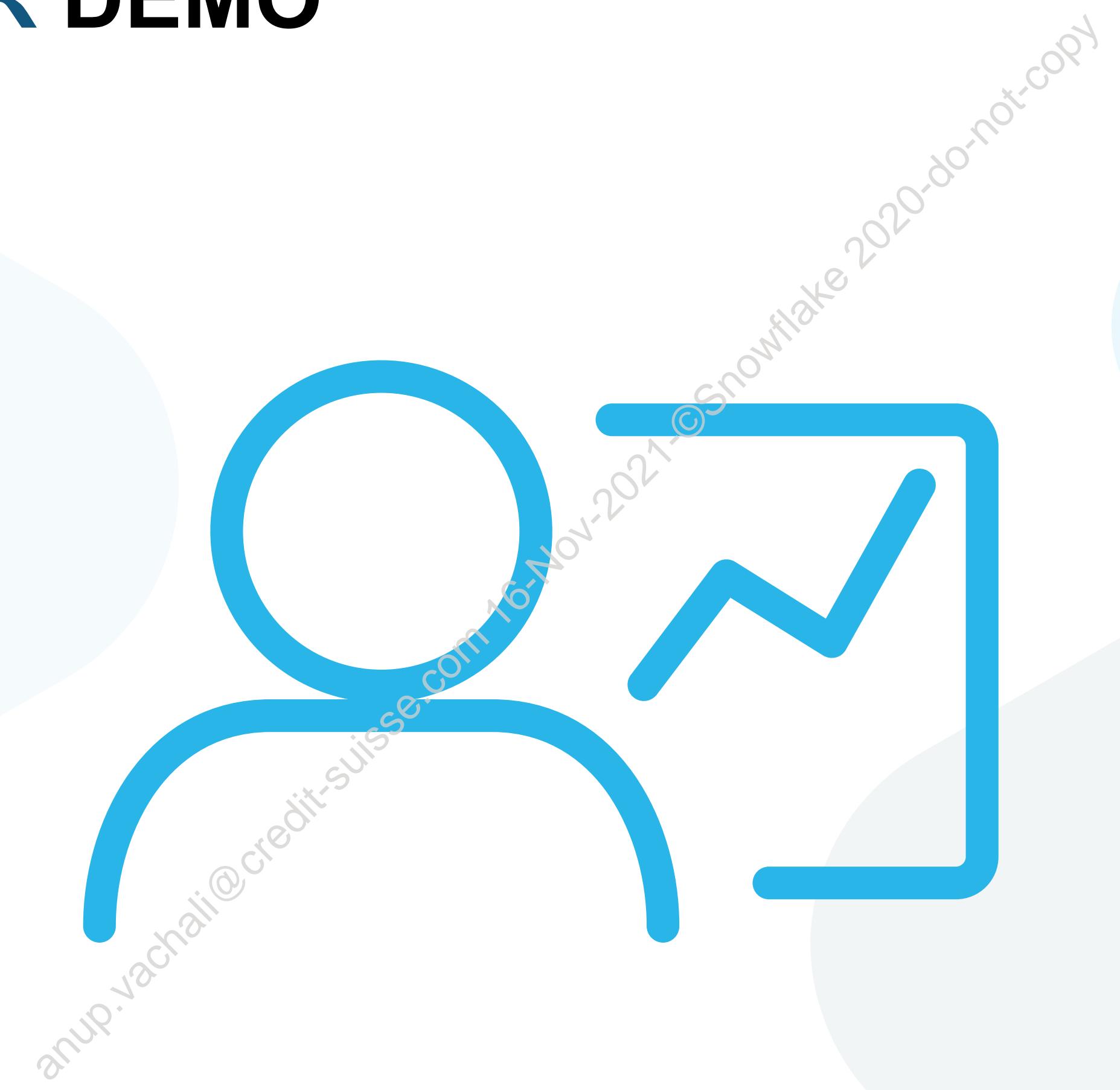
- Consumer accounts can create streams on shares



INSTRUCTOR DEMO

Use Streams

10 minutes



anup.vachali@credit-suisse.com 16-Nov-2021 ©Snowflake 2020-do-not-copy



LAB EXERCISE: 20

Track Table Changes with Streams

25 minutes

- Creating Basic Table Streams
- Querying Table Streams
- Exploring Delta and Append Streams
- Pairing Streams and Tasks



anup.vachali@credit-suisse.com 16-Nov-2021-@Snowflake 2020-do-not-copy

LAB EXERCISE

Fill Out Course Survey

5 minutes

- Available:
 - In course materials at <https://training.snowflake.com>
 - Via email (sent this afternoon)
 - Direct link: <https://www.surveymonkey.com/r/QZJ6LRS>





snowflake®

THANK YOU



anup.vachali@credit-suisse.com
Nov-2021 ©snowflake 2020-do-not-copy



© 2021 Snowflake Computing Inc. All Rights Reserved