100 Snowflake Cost Optimization Techniques

ALL ABOUT

COST AND PERFORMANCE OPTIMIZATION

IN SNOWFLAKE

100 Snowflake Cost Optimization Techniques

ALL ABOUT

COST AND PERFORMANCE OPTIMIZATION

IN SNOWFLAKE

Best Ways to Benefit from this Course

* Free Trial Snowflake Account
* VSCode Project Setup + GitHub
* 10 Sections w/ Introductions
* Eventual Hands-On + End-Review (slides)
* Quizzes to Test Your Knowledge
* Captions + Playback Speed
* Q&A + Money Back Guarantee
* Review

Introduction to Virtual Warehouses

Tip #1: Larger Virtual Warehouses May Actually Cost You Less

Tip #2: Auto-Suspend Any Warehouse After One Minute

Tip #3: Any Resumed Warehouse Will Cost You at Least One Minute

Tip #4: Never Auto-Suspend Any Warehouse After Less Than One Minute

Tip #5: X-Small Warehouses Could Be Powerful Enough

Tip #6: Resized Warehouses are for More Complex Queries

Tip #7: Multi-Cluster Warehouses are for Multiple Users and Concurrency

Tip #8: Multi-Cluster Warehouses Should Always Have Min Clusters 1

Tip #9: Use Economy Scaling Policy To Save Money

Tip #10: When to Use Snowpark-Optimized Warehouses

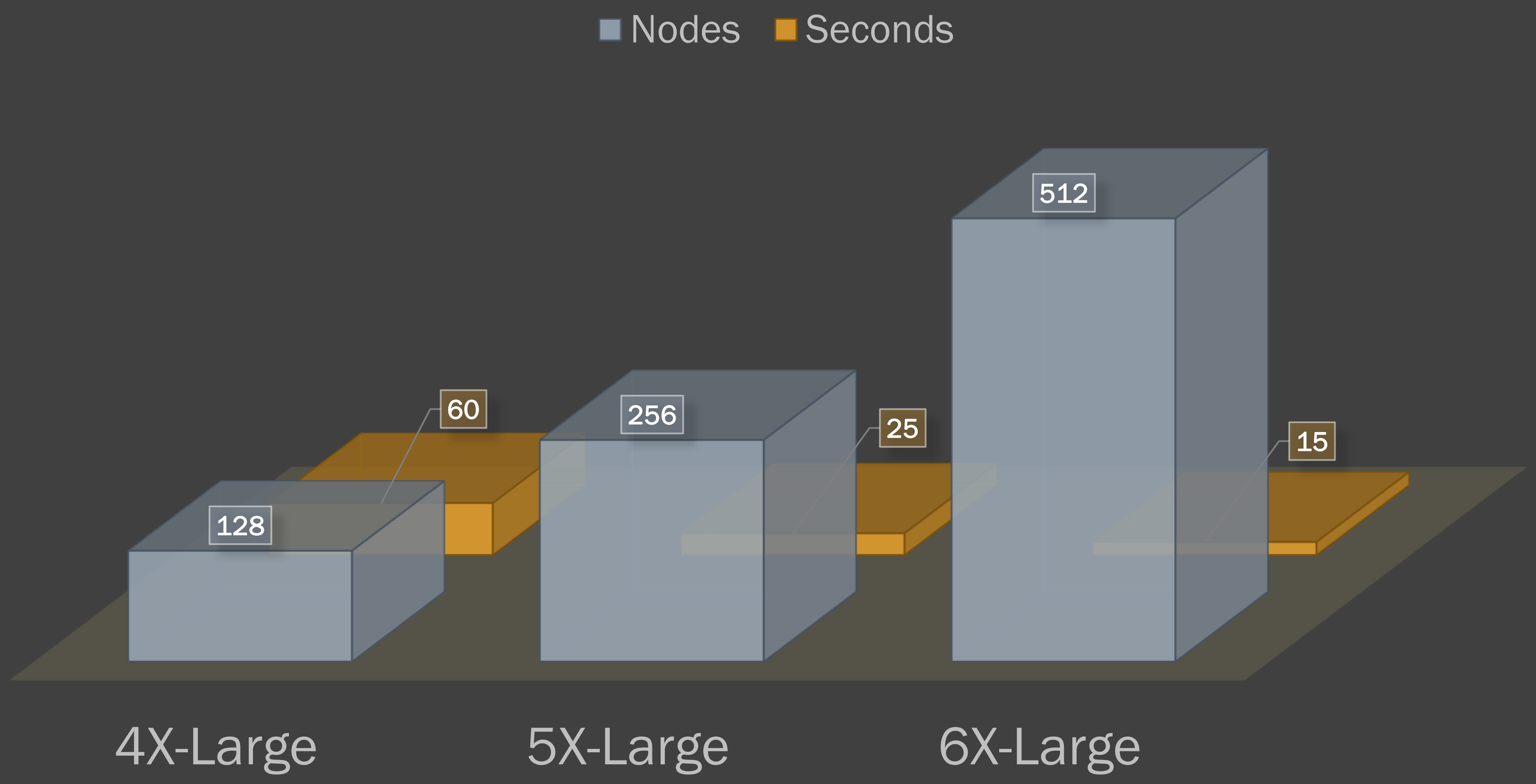
Larger Warehouses

May Actually

Cost You Less

Tip #1

WAREHOUSE SIZE
VS QUERY EXECUTION TIME



Cost

4X-Large: 128 Nodes x 60 secs = 128 x 5c = **$6.40**

5X-Large: 256 Nodes x 30 secs = 256 x 2.5c = **$6.40**

6X-Large: 512 Nodes x 15 secs = 512 x 1.25c = **$6.40**

at $3/credit = $3/hour/node = 5c/minute/node

Checkpoints

* virtual warehouses bring 80-90% of all Snowflake costs!
* *cost effective* = ~same cost, but using more resources
* double in size WH + query twice faster → ~same cost!
* bigger warehouses can actually save you money!
* at some point, bigger warehouses will no longer execute so fast
* small warehouses could take forever in very complex queries

Auto-Suspend

Any Warehouse

After One Minute

Tip #2

Cost

query execution time: 4 min/hour = 96 min/day x 5c = **$4.80**

1 min auto-suspend: 4 min/hour = 96 min/day x 5c = **$4.80**

5 min auto-suspend: 20 min/hour = 480 min/day x 5c = **$24**

for one query taking one min every 15 min

w/ X-Small warehouse → $3/credit = 5c/min (running!)

Checkpoints

* idle warehouses cost money (the same!)
* reducing auto-suspend to the min will save you big!
* creating a warehouse will auto-resume it
* can set INITIALLY_SUSPENDED to True only by code
* avoid auto-suspend by executing more queries together

Resumed Warehouses Will Cost You at Least One Minute

Tip #3

Cost

suspend after 10 secs: 192 nodes x 1 min x 5c = $9.60

suspend-resume in 10 secs: 192 nodes x 2 min x 5c = $19.20

for 3 clusters w/ 3X-Large WH each = 3 x 64 nodes = 192 nodes

Checkpoints

* you are charged at least one minute for any WH resume

* min auto-suspend (recommended) is one minute

* default auto-suspend is 5 minutes → change it to 1 min

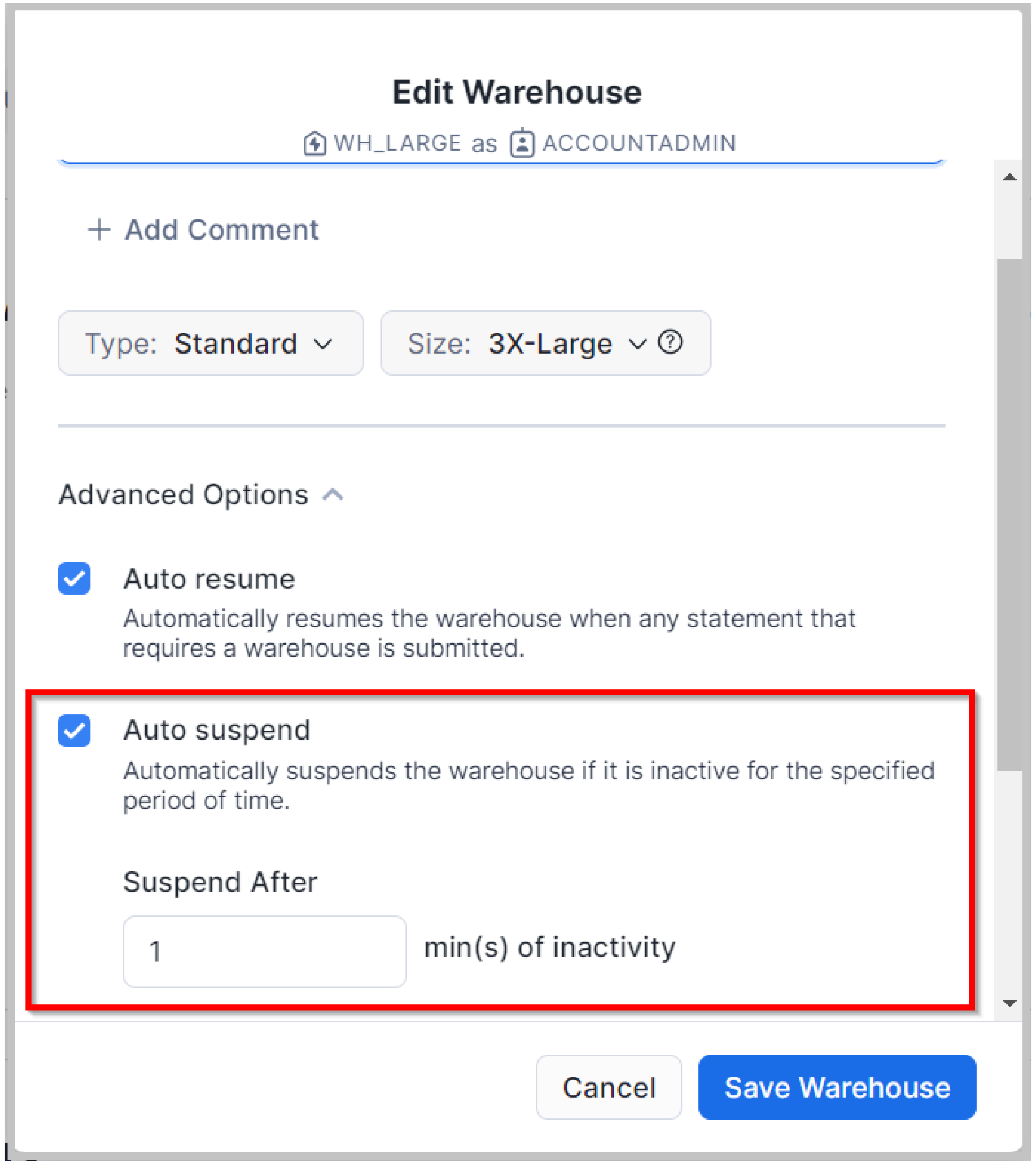
* a background task checks every minute for auto-suspend

* the one-minute period restarts when warehouse resumed

* suspend+resume within a minute may charge you twice

Never Auto-Suspend Any Warehouse After Less Than a Minute

Tip #4


Edit Warehouse
WH_LARGE as ACCOUNTADMIN
+ Add Comment
Type: Standard
Size: 3X-Large
Advanced Options
Auto resume
Automatically resumes the warehouse when any statement that requires a warehouse is submitted.
Auto suspend
Automatically suspends the warehouse if it is inactive for the specified period of time.
Suspend After
1
min(s) of inactivity
Cancel
Save Warehouse

Checkpoints

* in Snowsight → min auto-suspend is one minute (recommended)
* in SQL → min auto-suspend is 0 seconds (60 recommended)
* auto-suspend zero means you always keep it alive
* a background task checks every minute for auto-suspend
* the one-minute period restarts when warehouse resumed
* suspend+resume within a minute may charge you twice

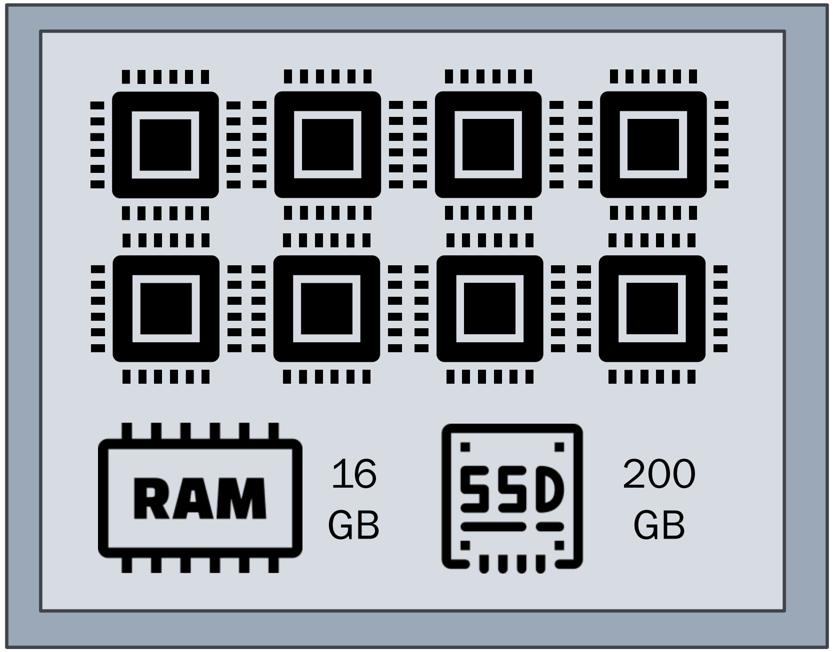
X-Small Warehouses

Could Be

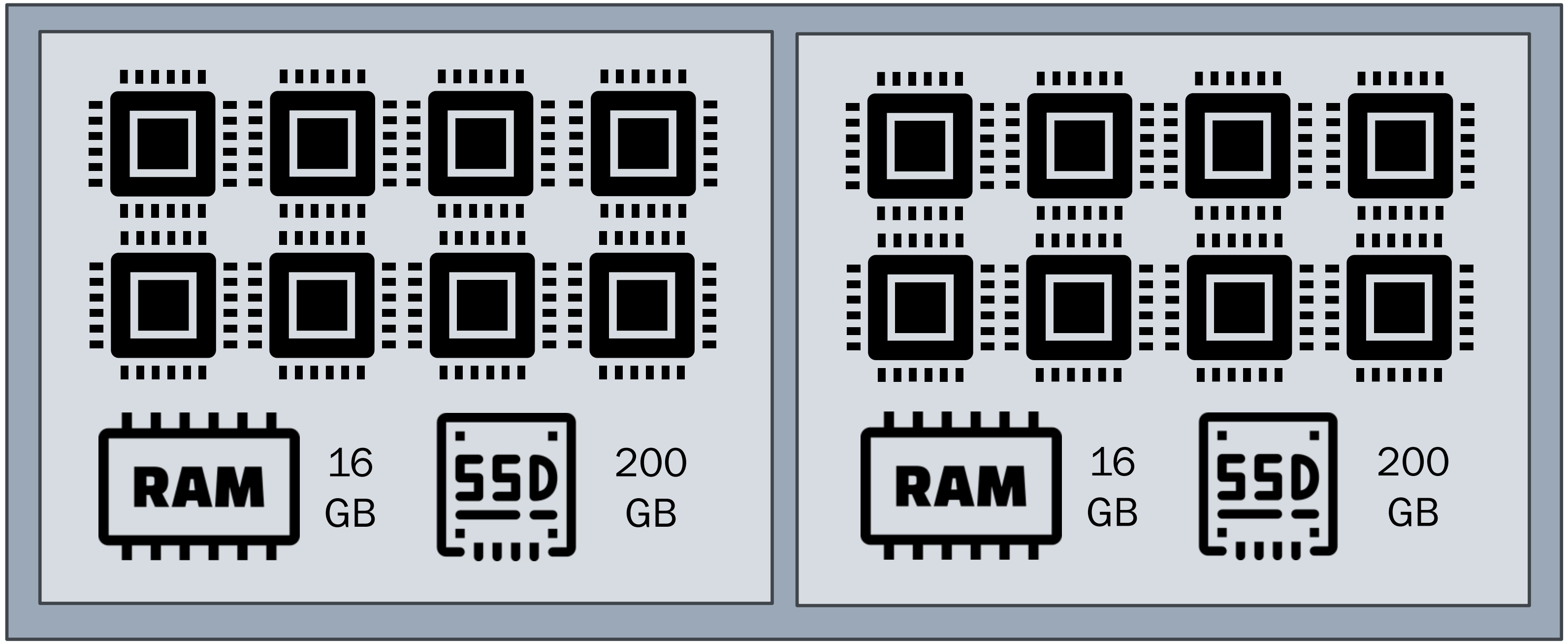
Poweful Enough

Tip #5

X-Small Warehouse (1 node)



Small Warehouse (2 nodes)



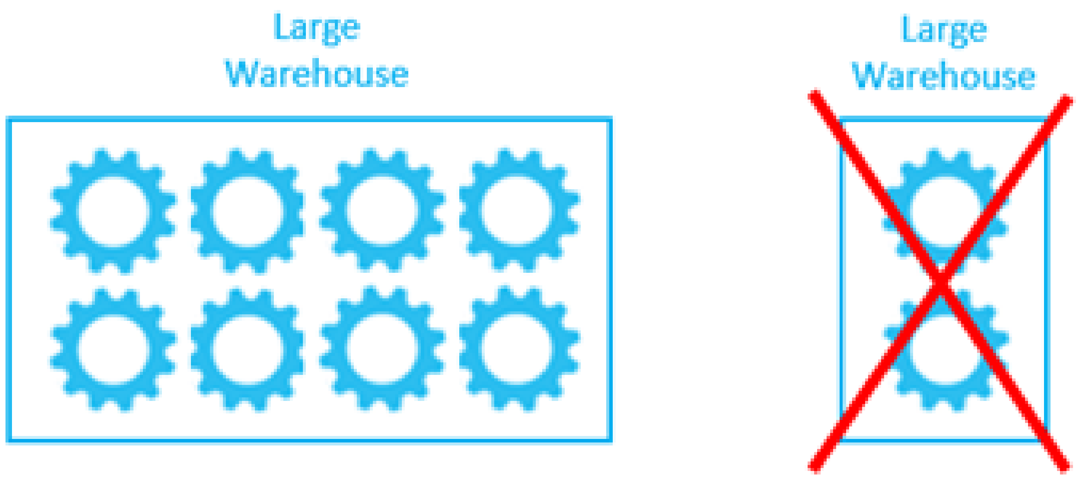
Checkpoints

* a "node" is way more than a one-CPU VM
* a single "node" is rather a powerful server w/ 8-vCPUs
* do not overspend, as X-Small could be enough
* assumed 16 GB RAM + 200 GB SSD
* for AWS: c5d.2xlarge EC2 instance (except for 5XL and 6XL)
* but only Snowflake knows what's in a node (CPU, RAM, SSD)

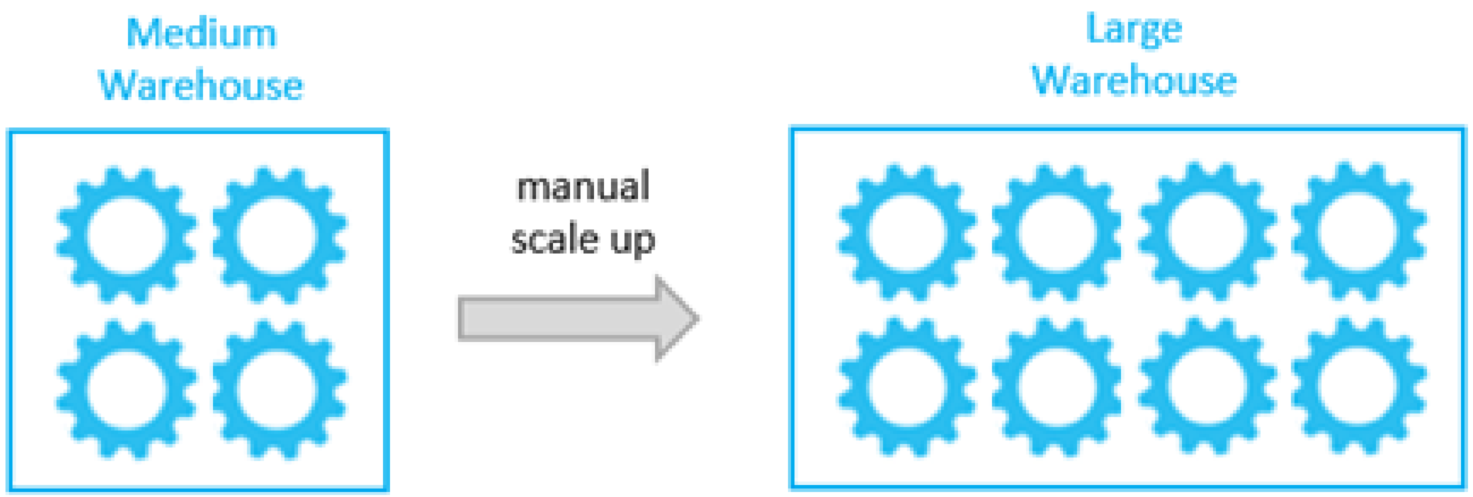
Resized Warehouses are for More Complex Queries

Tip #6

Fix Warehouse Size



Manual Scale



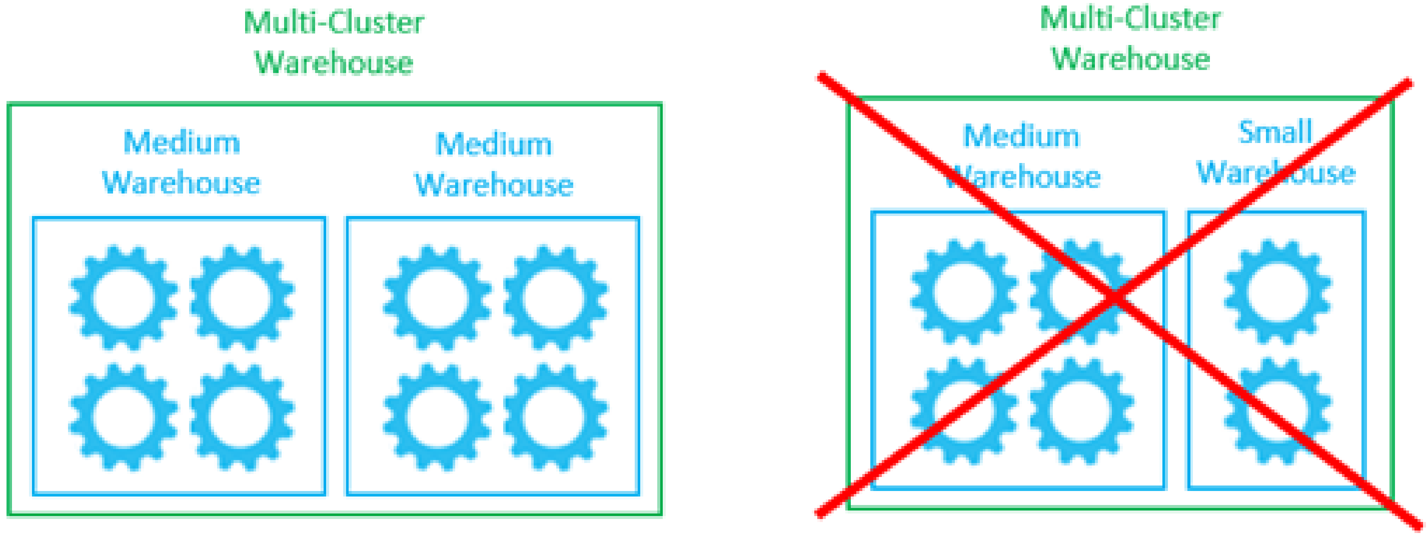
Checkpoints

* bigger sizes could help more complex queries execute faster
* bigger warehouse size could cost you less (as seen before)
* no concurrency improvement by resizing
* resizing a warehouse is always a manual process
* one query can be executed by only one warehouse
* cannot have different sizes in a multi-cluster warehouse

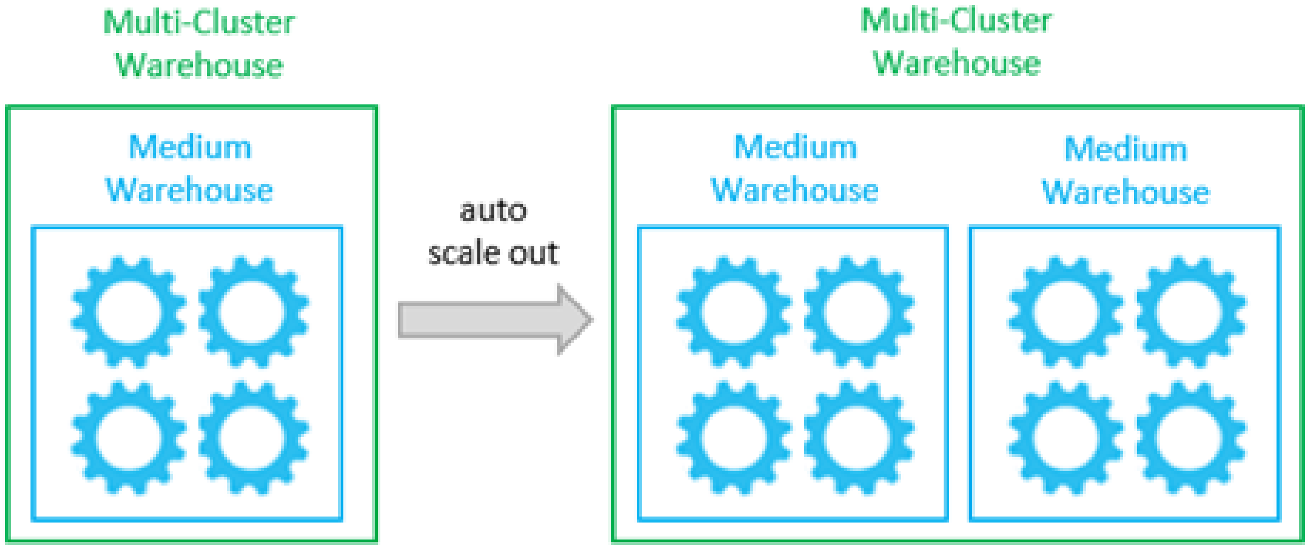
Multi-Cluster Warehouses are for Multiple Users and Concurrency

Tip #7

Fix Cluster Size



Auto Scale


Multi-Cluster Warehouse
Medium Warehouse
auto scale out
Multi-Cluster Warehouse
Medium Warehouse
Medium Warehouse

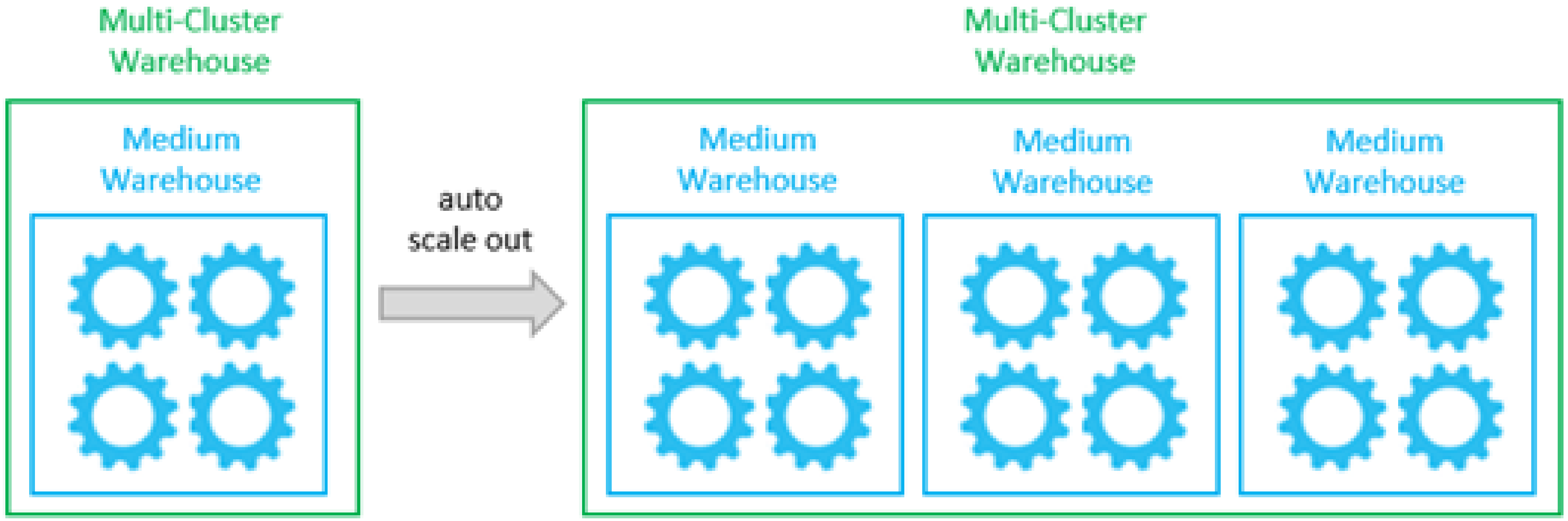
Checkpoints

* multi-cluster warehouses improve concurrent access
* multi-cluster warehouses do not help with complex queries
* multi-clustering = main reason you should have Enterprise Edition
* cannot have different sizes in a multi-cluster warehouse
* could be avoided by assigning diff users/queries to diff warehouses
* support auto-scaling (between 1..10 clusters)
* one query can be executed by only one warehouse

Multi-Cluster Warehouses Should Always Have Min Clusters 1

Tip #8

Auto Scale


Multi-Cluster
Warehouse
Medium
Warehouse
auto
scale out
Multi-Cluster
Warehouse
Medium
Warehouse
Medium
Warehouse
Medium
Warehouse

Checkpoints

* do not over-provision w/ min cluster count > 1

* unnecessary costs

* you may have one user running queries w/ very low latency (no queuing)

* min cluster count = max cluster count → no auto-scale

Use Economy Scaling Policy To Save Money

Tip #9

Multi-Cluster Warehouses

* *scale modes*

- **maximized** → min=max clusters
- **auto-scale** → between min < max clusters

* *scaling policies*

- **standard** (def) → for end-user live queries (performance is top priority)
- **economy** → for lower priority batch jobs (to optimize cost)

Standard Scaling Policy

* *auto-resume*: after 20 secs when new queued req + prior WH at its capacity

* *auto-suspend*: after 2-3 checks every min, if queries on least loaded WH can be distributed elsewhere

* avoid query queuing without saving credits

* for end-user live queries, when performance is a priority

Economy Scaling Policy

* *auto-resume*: when crt queue load could keep new WH busy for min 6 mins

* *auto-suspend*: after 5-6 checks every min, if queries on least loaded WH can be distributed elsewhere

* save credits, query queuing may happen

* for lower priority batch jobs, to maximize throughput + optimize cost

Checkpoints

* standard scaling policy (def) = will avoid queuing, but save no credits

* economy scaling policy = will save credits, but have query queuing

* auto-scale in (WH shutdown) could take 5-6 or 2-3 minutes!

* avoid scale mode *maximized* → prefer *auto-scale* between min..max

* avoid MAX_CLUSTER_COUNT 10 → keep 5 ok

* MAX_CONCURRENCY_LEVEL (def 8) = queries in parallel on a WH

When to Use Snowpark-Optimized Warehouses

Tip #10

Costs

Type of Virtual Warehouse	X-Small	Small	Medium	Large	X-Large	2X-Large	3X-Large	4X-Large	5X-Large	6X-Large
Standard	1	2	4	8	16	32	64	128	256	512
Snowpark-optimized	n/a	n/a	6	12	24	48	96	192	384	768

Checkpoints

* they have 16x more memory than Standard WHs + cost 33% more

* they are more for Data Science and ML experiments

* for one stored proc deployed by Snowpark in a single-node WH

* well-suited for model training in ML (not model serving!)

* no GPUs → see rather Snowpark Container Services w/ NVIDIA

Introduction to Compute Workloads

Tip #11: Use Resource Monitors

Tip #12: Use Account-Level Budgets

Tip #13: Prevent Never-Ending Queries

Tip #14: Manually Kill Running Queries

Tip #15: Reduce Warehouse Sizes

Tip #16: Consolidate All Warehouses

Tip #17: Use Parallel Jobs for Batch Transformations

Tip #18: Avoid Checking Too Much on Metadata

Tip #19: Charts for Warehouse Monitoring

Tip #20: Revisit the Main Traps with Warehouses

Use

Resource Monitors

Tip #11

CREATE RESOURCE MONITOR

* *levels*: account / warehouse ← SET RESOURCE_MONITOR = ...

* with *credit_quota* ← auto-reset monthly by default

* *triggers* (on N percent do) notify / suspend / suspend_immediate

* *frequency* = <u>monthly</u> / weekly / daily / yearly / never

* *start_timestamp* (= immediately) *... end_timestamp*

* *notify_users* = ("CRISTISCU", ...) ← all ACCOUNTADMIN by def

Checkpoints

* specify one monitor for the whole account

* specify one monitor for each virtual warehouse (1-N)

* enable notifications for each account admin

* watch all warehouses with no res mon assigned

* suspend immediate if long running queries expected

* schedule, for batch queries w/ periodic runs

Use

Account-Level

Budgets

Tip #12

Checkpoints

* *account budget*
  - PuPr, only for paying accounts, must enable/activate first
  - max monthly spending limit → auto-send emails when exceeding
  - SNOWFLAKE.LOCAL.ACCOUNT_ROOT_BUDGET class
* *custom budgets*
  - on group of resources, in db.schema, max 100
  - dbs, schemas, tables, mat views, WHs, pipes, tasks, serverless
  - SNOWFLAKE.CORE.BUDGET class

Prevent Never-Ending Queries

Tip #13

Checkpoints

* a running query is canceled by the system after 2 days
* STATEMENT_TIMEOUT_IN_SECONDS = sess/WH/acct
  - def 2 days (172,800 secs), 0 to 604800 (7 days)
  - max overall time = queue + locked + exec + compile + ...
  - also for queries w/o WH (DDL/DML: CLONE, CREATE...)
* STATEMENT_QUEUED_TIMEOUT_IN_SECONDS = def 0

Manually Kill Running Queries

Tip #14

Checkpoints

* *Cancel* button in Snowsight, *X* button in Query History
* SYSTEM$CANCEL_QUERY with last_query_id()
* CTRL+C from SnowSQL
* only own queries (not necessarily ACCOUNTADMIN)
* NOT with: close browser, abort session, suspend WH
* could also manually stop WH when query ends, to save money!

Reduce

Warehouse Sizes

Tip #15

Checkpoints

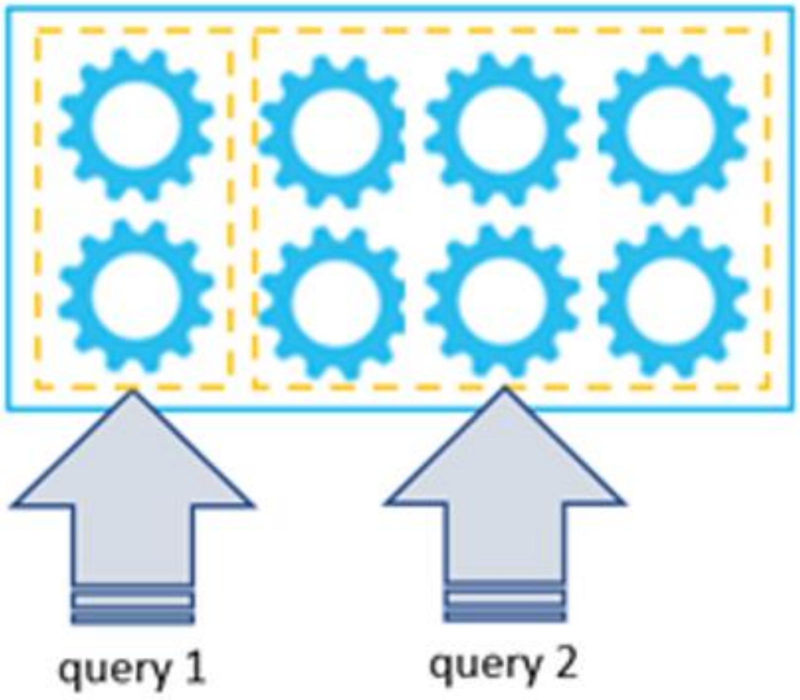
* start with the smallest 1-node X-Small warehouse
* start with MIN_CLUSTER_COUNT 1 in multi-cluster WH
* as a user, connect with an X-Small default WH
* upsize only when need for larger WH is proven
* use Snowpark-optimized only for mem-intensive workloads
* drop expensive warehouses, not frequently used
* cancel queries taking too long w/ current warehouse

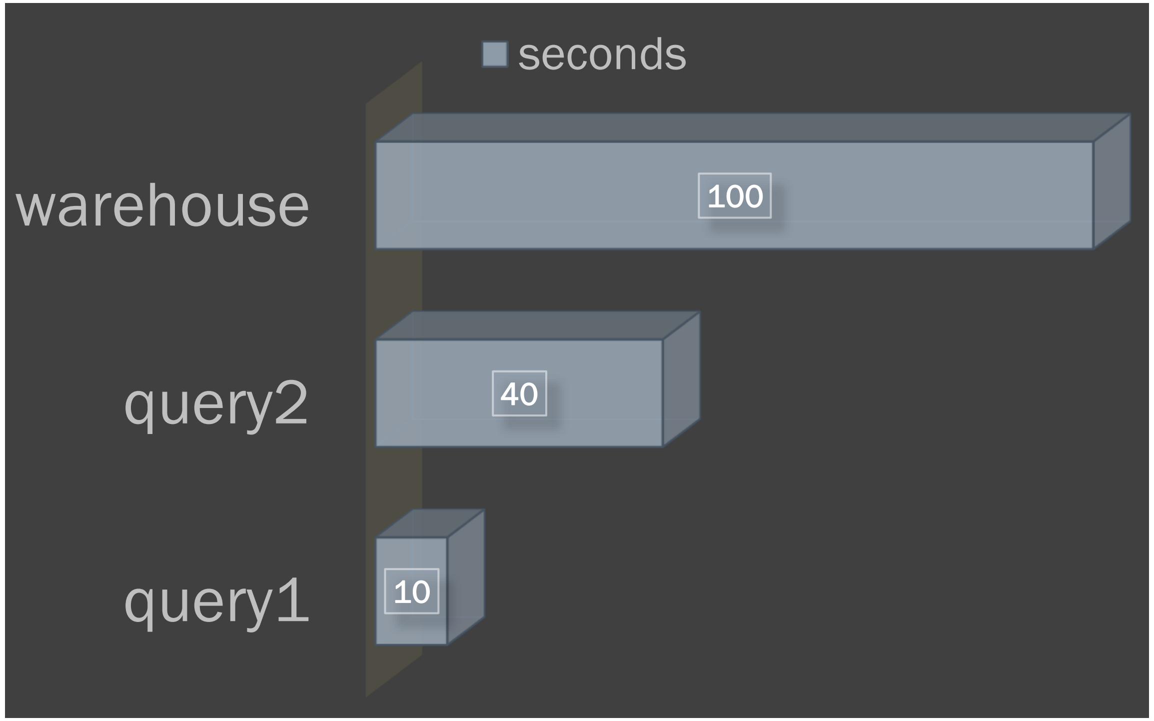
Consolidate

All Warehouses

Tip #16

Query Execution





Warehouse Load/Utilization

* *query1 load*: 10/100/2 = 5% (could run 20 such queries in 100 secs)

* *query2 load*: 40/100/2 = 20% (could run 5 such queries in 100 secs)

* *warehouse load/utilization*: 0.05 + 0.2 = 25%

query1 (10 secs) + query2 (40 secs)

warehouse max_concurrency_level = 2

warehouse up: 40 secs (queries) + 60 secs (auto-suspend) = 100 secs

System Views

* WAREHOUSE_METERING_HISTORY
    - *start_time .. end_time + warehouse_name*
    - ***credits_used***
* QUERY_HISTORY
    - *start_time .. end_time + warehouse_name*
    - ***execution_time*** + *query_load_percent*

Checkpoints

* consolidation = reduce # of *underutilized* warehouses (cost money!)

* *overutilized* warehouses (w/ queuing) have no cost impact

* *query latency* (queuing) vs *warehouse utilization* ← compromise!

* *query load* vs *warehouse load* (+ MAX_CONCURRENCY_LEVEL)

* redirect queries from some WHs to other WHs ← see utilization heatmap

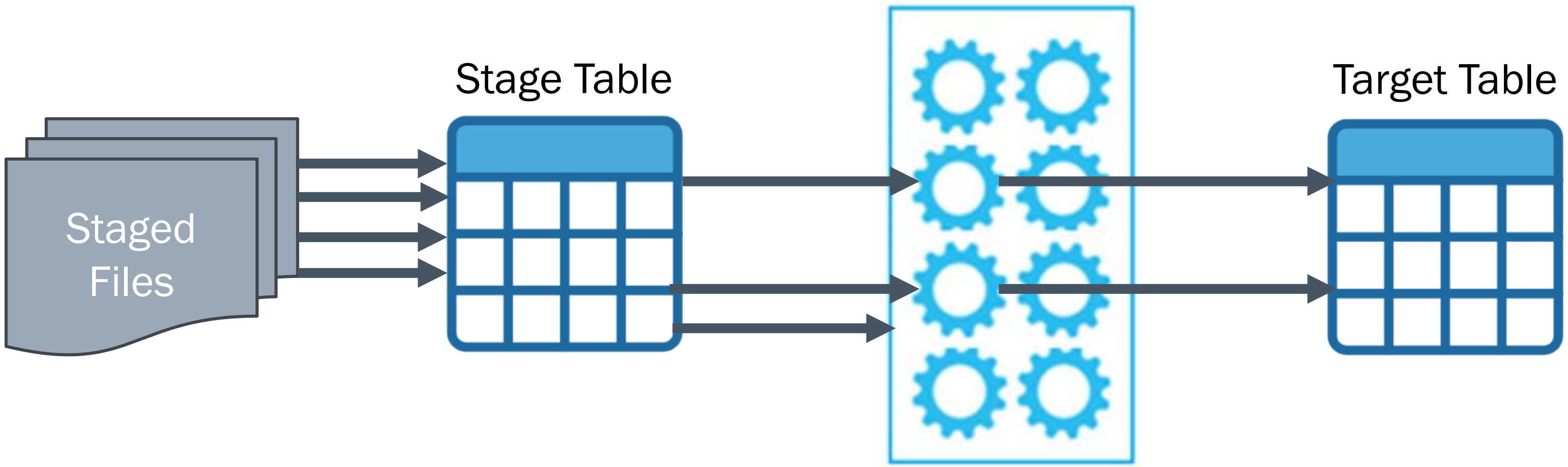
* group similar workloads in the same WH

* *workload type* (*util. target*): ETL (95%), interactive (80%), dashboards (60%)

Use Parallel Jobs for Batch Transformations

Tip #17

Data Transformation Pipeline



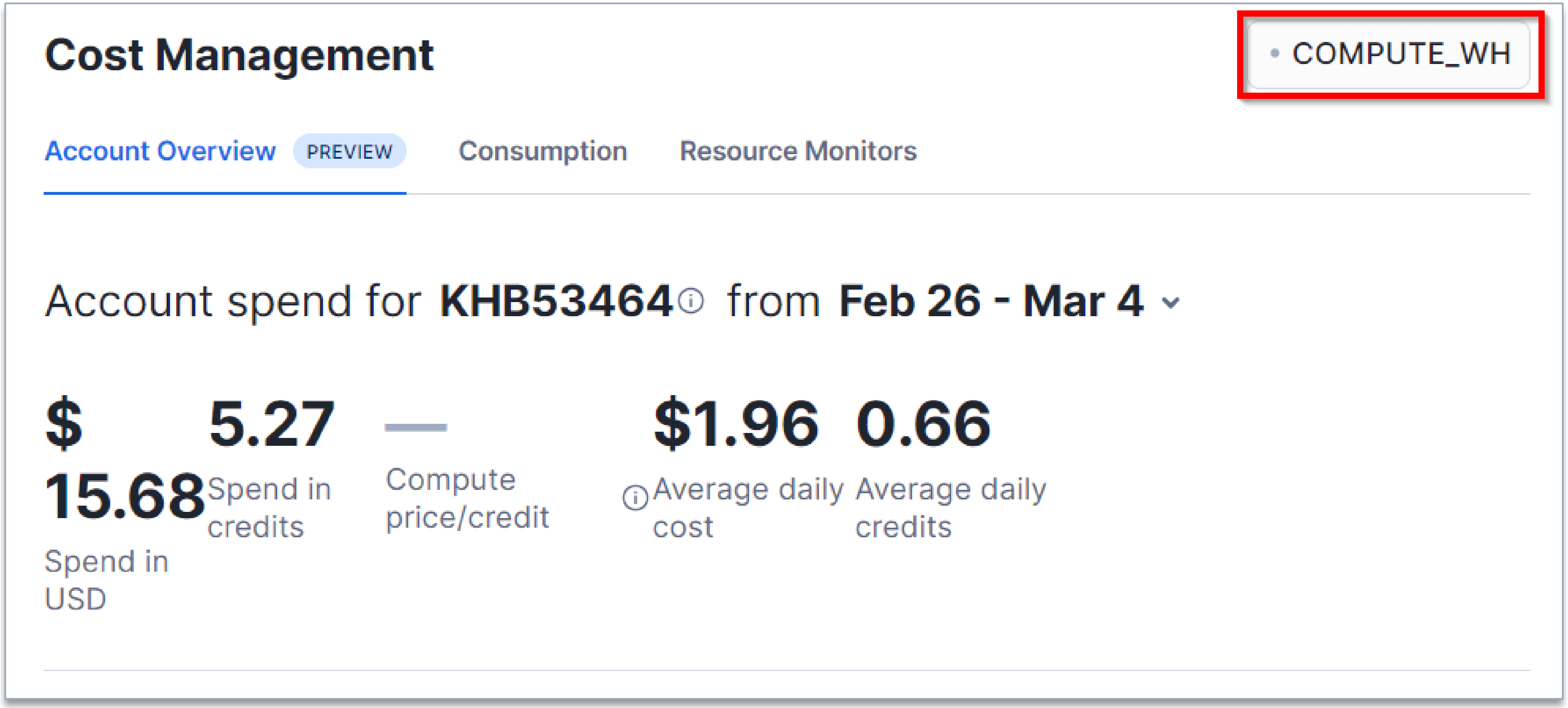
Checkpoints

* ~parallel ingestion into table from staged files
* split intensive MERGE statements w/ filters
* send multiple SQL jobs in parallel on diff connections
* max concurrency on 1-node X-Small WH is 10
* increase WH size for better distribution
* multi-cluster WH could also make sense for >> opers

Avoid Checking Too Much on Metadata

Tip #18

Snowsight


Cost Management
COMPUTE_WH
Account Overview
PREVIEW
Consumption
Resource Monitors
Account spend for KHB53464 from Feb 26 - Mar 4
$ 15.68
Spend in USD
5.27
Spend in credits
—
Compute price/credit
$1.96
Average daily cost
0.66
Average daily credits

Checkpoints

* *Snowsight*
  - Snowflake's admin console will incur a cost!
  - check Snowsight screens w/ a WH
  - auto-refresh Query History every N seconds

* *querying metadata*
  - querying Information Schema does not require a WH
  - querying Account Usage schema requires a WH
  - many Account Usage queries are rather slow (ex: *grants_to_roles*)
  - many Account Usage views could return a lot of data (ex: *query_history*)
  - many columns return JSON data, that needs flattening (ex: *access_history*)

Charts for Warehouse Monitoring

Tip #19

Checkpoints

* overall compute: warehouses / cloud services / serverless
* WAREHOUSE_METERING_HISTORY view
* top warehouses by cost → credits used
* utilization heatmaps → warehouse loads
* queries → running/provisioning/blocked/queued
* total spending per user / query type ...
* timeline charts

Revisit the Main Traps with Warehouses

Tip #20

Checkpoints

* no predef. resource monitors + not through UI → no proactive approach to overspending
* AUTO_SUSPEND by def 600 (10 minutes) in code + 5 min in UI
* billed per second, but at least 1 min + AUTO_SUSPEND 1 min at least (can be set lower!)
* manual 2+ resume+suspend pairs in the first minute → billed for 2+ minutes!
* INITIALLY_SUSPENDED by def FALSE and no UI access → bill 1 min no matter what
* MIN_CLUSTER_COUNT could be set to >1 → should be always 1
* STATEMENT_TIMEOUT_IN_SECONDS 2 days by def + hidden Cancel btn in Query History
* STATEMENT_QUEUED_TIMEOUT_IN_SECONDS 0 by def (no timeout)

Introduction to Snowflake Accounts

Tip #21: What to Choose for a Free Trial Account

Tip #22: When to Use a Free Trial Account

Tip #23: Understand Price Tables for Warehouse Compute Services

Tip #24: Understand Price Tables for Cloud and Serverless Services

Tip #25: Understand Price Tables for Storage and Data Transfer

Tip #26: Use the Account Overview Interface in Snowsight

Tip #27: Use Organization Accounts

Tip #28: Limit Warehouse Changes with Access Control

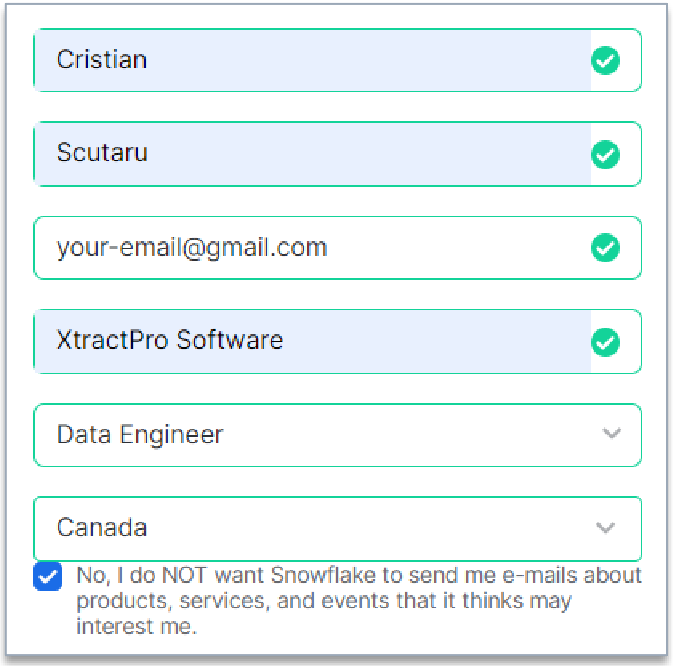
Tip #29: Adjust Default Values of Account-Level Parameters

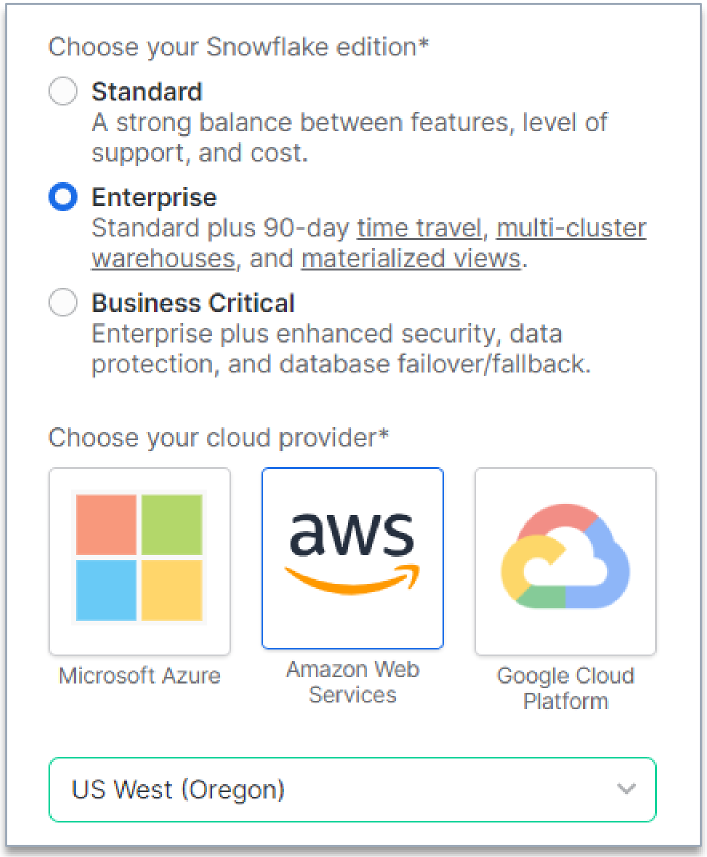
Tip #30: Careful with Reader Accounts

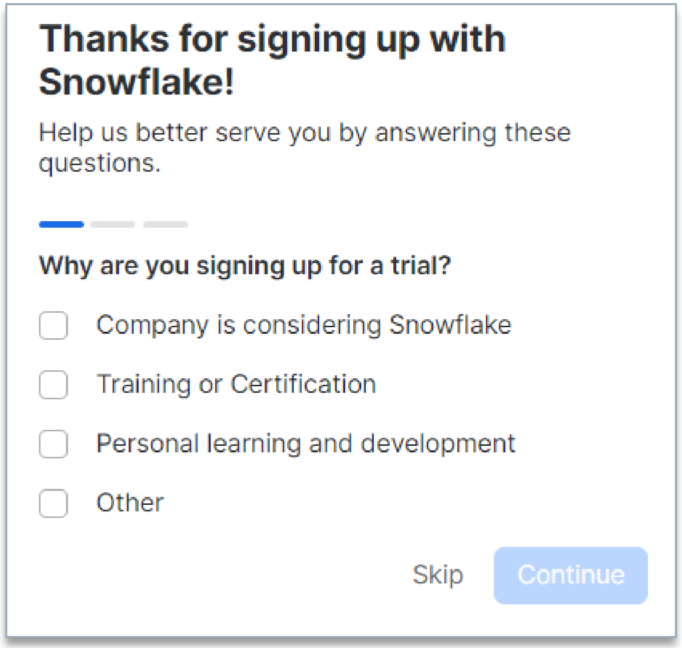
What to Choose for a Free Trial Account

Tip #21

signup.snowflake.com







Checkpoints

* no credit card required / no obligations whatsoever
* email address could be reused after a while
* no need for a corporate email address
* default Enterprise edition could be the most appealing
* AWS is frequently used first as provider for new PuPr features
* region closest to where you are / where your data is
* can skip the last survey questions

When to Use

a Free Trial Account

Tip #22

Limitations

* running out of email addresses
* some new features (Budgets) not on free trial accounts
* native apps sharing requires accounts within org
* no Snowflake customer service
* you'll need to upload/reinstall all after one month
* you lose all access after one month

Checkpoints

* trial of native apps from Snowflake Marketplace
* experiments on large public datasets
* prototyping + trying new Snowflake features
* introducing a new potential client to Snowflake
* prepare for + write a new Udemy course

Understand Price Tables for Virtual Warehouse Compute Services

Tip #23

VW Compute Pricing

* Standard Edition = $2/credit/node = $2/h for X-Small WH

* Enterprise Edition = $3/credit/node = $3/h for X-Small WH

* Business Critical Edition = $4/credit/node = $4/h for X-Small WH

in AWS/US region, for active 1-Node WH

Small (2-Nodes) is 2x more expensive, Medium (4-Nodes) 4x etc.

Snowflake Pricing

* Compute = w/ credits, for active **VW/SPCS**, per second, minimum 1 min

* Cloud Services = 4.4 credits/h, not billed if < 10% of daily VW credits

* Serverless Features = on managed compute (+cloud services), per second

* Data Sharing = pay for all Reader Accounts, not for other Snowflake accounts

* Storage = avg TB/mo (snapshots of data), compressed, diff for hybrid tables!

* Data Transfer = Ingress/Egress, separate for external functions

Checkpoints

* *all compute*

- VW Compute
- Cloud Services
- Serverless Features

* *for the same supported queries*

- Enterprise is 1.5x more expensive than Standard
- Business Critical is 2x more expensive than Standard

Understand Price Tables for Cloud and Serverless Services

Tip #24

Cloud Services

* distributed back-end systems
* SHOW commands
* query compilation
* result caches
* INFORMATION_SCHEMA queries

Serverless Features

* Query Acceleration Service (QAS) - Nx boost of a WH
* Search Optimization Service (SOS) - (also in secondary db)
* Automatic Clustering - for clustered tables (w/ clustered key), initial + recluster
* Materialized Views - for maintenance (also in secondary db), auto-refresh data
* Snowpipe & Snowpipe Streaming - w/ fixed credits/files or h
* Serverless Tasks - 1.5x Managed Tasks (w/ WH)
* Hybrid Tables - for maintenance+requests, indexes, R/W data
* Replication - only in Business Critical for objects other than dbs

Checkpoints

* *cloud services*

- charged only when > 10% x daily VW consumption (rare!)
- could pile up for very frequently used metadata queries (~10k times)

* *serverless features*

- Snowflake-managed vs user-managed resources
- charged w/ credits, ~WH, depending on the service
- billed per second, not when idle (like for the user-managed resources)
- Snowflake-managed compute + cloud service credits

Understand Price Tables for Storage and Data Transfer

Tip #25

Checkpoints

* *storage*

- small fraction, in average TB/mo, for compressed data
- On-Demand / Capacity
- Hybrid Tables storage billed separately → GB/mo

* *data transfer*

- very small fraction, in TB transferred, ~the cloud provider
- for data out only (data in free)
- free when within same region/continent/provider
- from COPY, replication, external functions

Use the Account Overview Interface in Snowsight

Tip #26

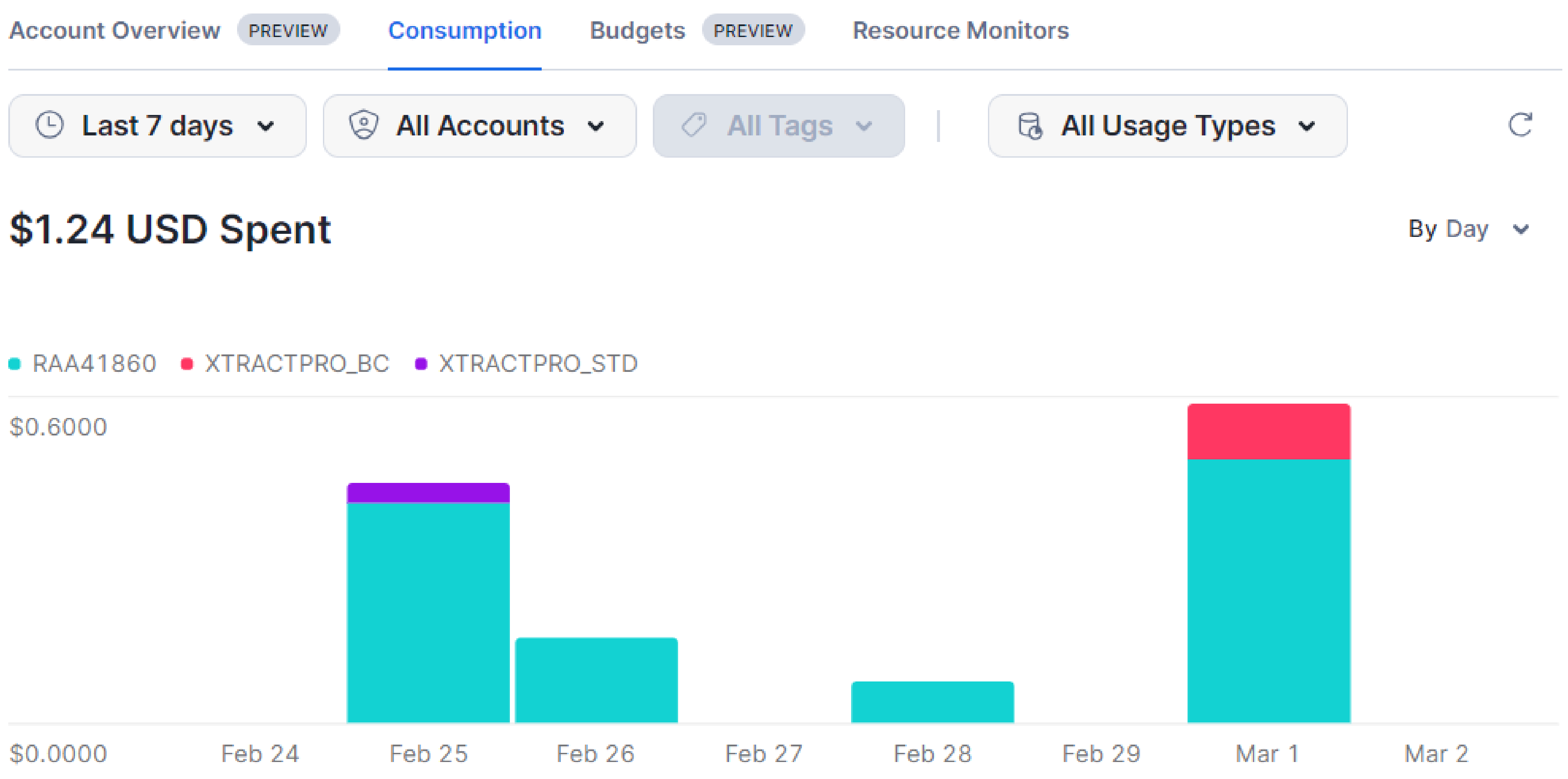
Cost Management interface

Account Overview PREVIEW | Consumption | Budgets PREVIEW | Resource Monitors

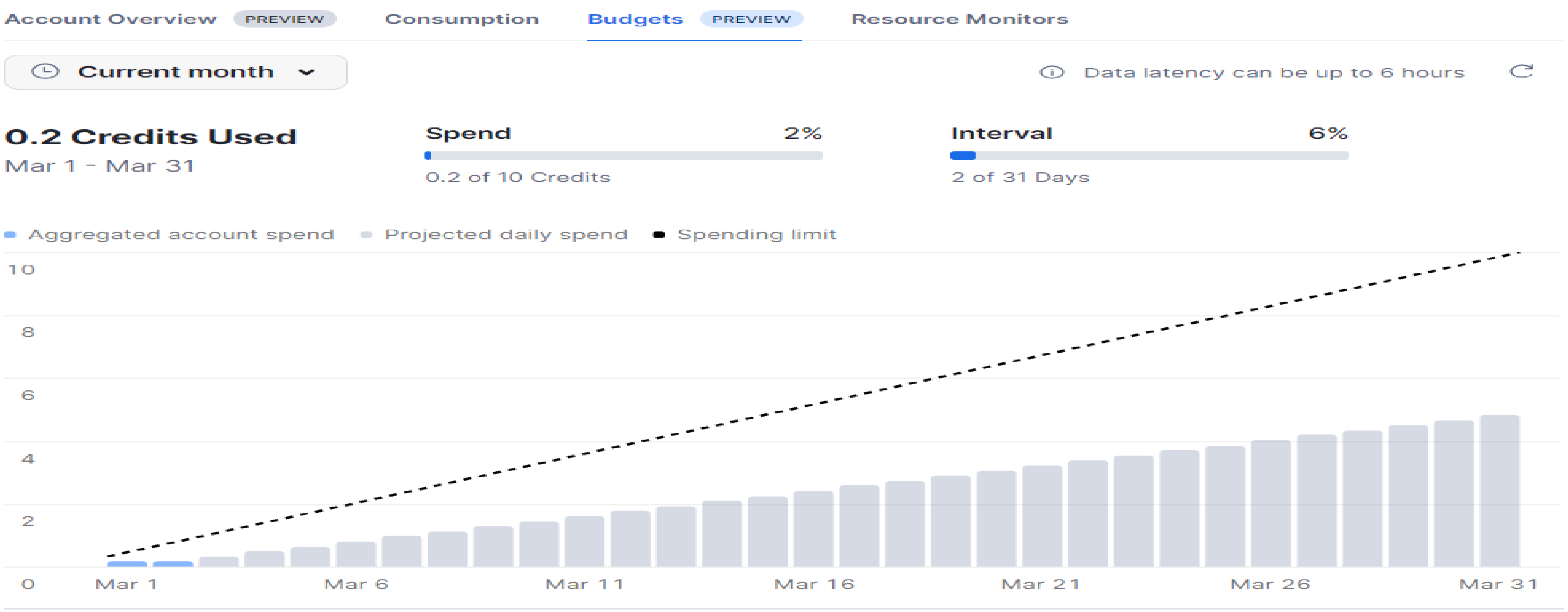
Account spend for **RAA41860** from **Feb 24 - Mar 2**

$1.11	**0.32**	**$3**	**$0.14**	**0.04**
Spend in USD	Spend in credits	Compute price/credit	Average daily cost	Average daily credits

Consumption



Budgets



Account Overview
PREVIEW
Consumption
Budgets
PREVIEW
Resource Monitors
Current month
Data latency can be up to 6 hours
0.2 Credits Used
Mar 1 - Mar 31
Spend
2%
0.2 of 10 Credits
Interval
6%
2 of 31 Days
Aggregated account spend
Projected daily spend
Spending limit
10
8
6
4
2
0
Mar 1
Mar 6
Mar 11
Mar 16
Mar 21
Mar 26
Mar 31

Checkpoints

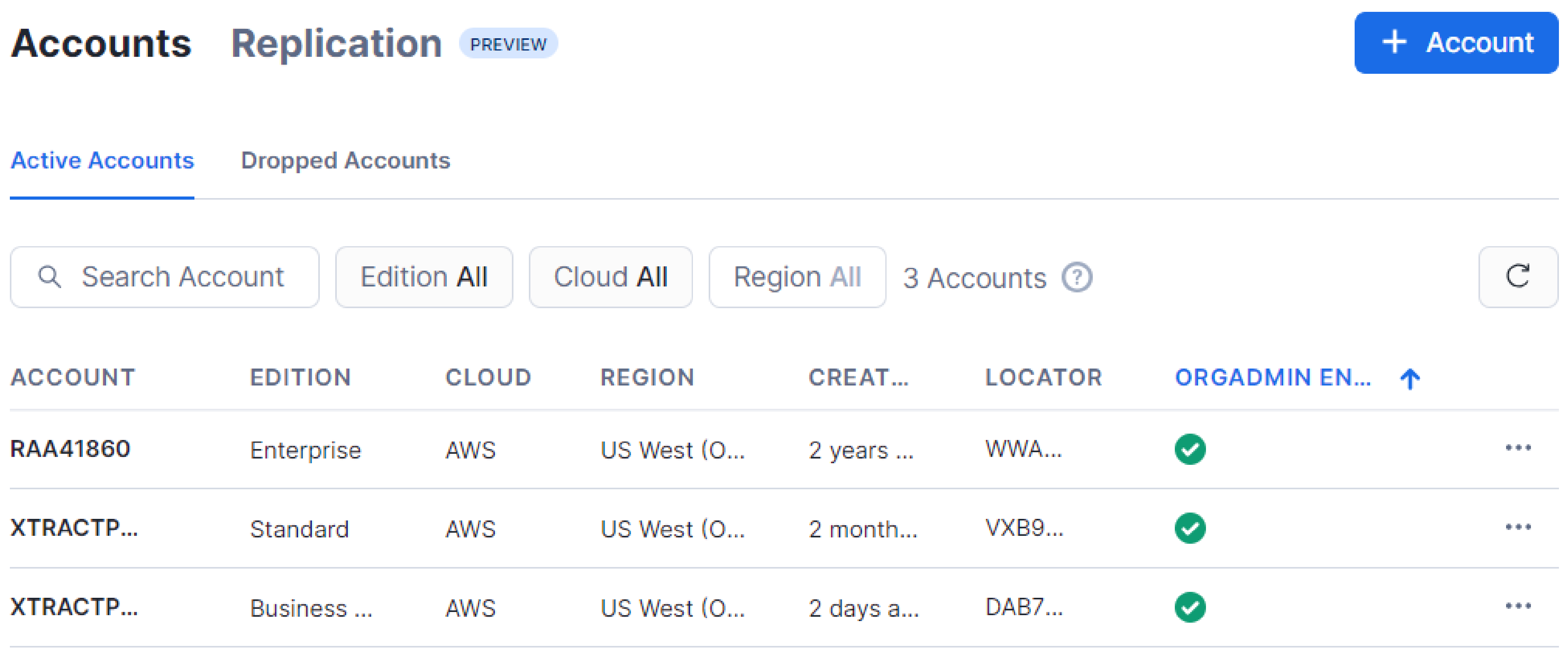
* **Cost Management** - Admin interface, w/ date range
  - Account Overview - PuPr
  - Consumption - per date range/acct, C/S/DT
  - Budgets - PuPr, only paid accounts
  - Resource Monitors -

* Account Overview Insights
  - ***Account budget for current month*** - w/ budget(s)
  - ***Top warehouses by cost*** - w/ SQL
  - ***Top databases by storage*** - w/ SQL
  - ***Most expensive queries*** - total/avg exec time + #

Use Organization Accounts

Tip #27

Accounts interface



Accounts interface

* Create New Account
  - *edition*: Standard/Enterprise/Business Critical
  - *cloud/region*: AWS/Azure/GCP
* accounts: active/dropped ← 3..90 days grace period
  - manage URLs: locator/org-acct
  - enable/disable ORGADMIN

Checkpoints

* create new related paid Snowflake accounts
* create Standard account for basic query experiments
* private share data & native apps within org accounts
* test new PuPr features available only in paid accounts
* requires higher level of security than free trial accounts
* SHOW ORGANIZATION ACCOUNTS ← w/ ORGADMIN

Limit Warehouse Changes with Access Control

Tip #28

Checkpoints

* USAGE ← execute/abort queries (w/ eventual auto-resume)

* OPERATE ← execute/abort queries + view crt/past queries + suspend/resume

* MODIFY ← alter properties (including size) + assign to resource monitors

* MONITOR ← view current/past queries + usage statistics

* APPLYBUDGET ← add/remove from a budget

* OWNERSHIP ← full control (for a single role)

* ALL ← all privileges but OWNERSHIP

Adjust Default Values of Account-Level Parameters

Tip #29

Checkpoints

* *time travel* ← cannot be disabled for an account!

- MIN_DATA_RETENTION_TIME_IN_DAYS - (def 0), can force TT, does not alter current settings
- DATA_RETENTION_TIME_IN_DAYS - (def 1) 0..90 Ent / 0..1 Std, set def for database/schema/permanent tables, but cannot force disable w/ 0

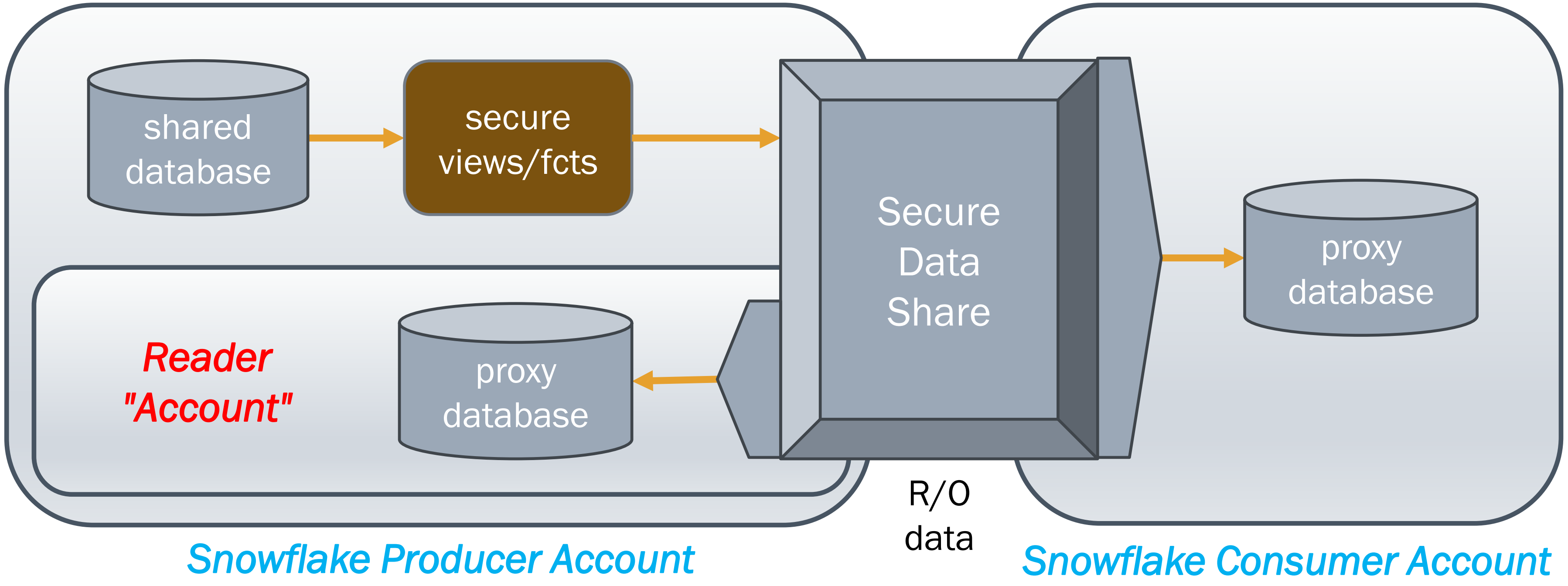
* *replication*

- INITIAL_REPLICATION_SIZE_LIMIT_IN_TB – (def 10) max estimated size limit for initial replication of a primary db to a sec db, to prevent accounts w/ sec dbs from accidentally incurring large replication charges

Careful with Reader Accounts

Tip #30

Data Share w/ Reader Account



Checkpoints

* Reader accounts are not separate Snowflake accounts
* Reader accounts consume YOUR credits!
* monitor closely all your Reader accounts
* set resource monitors w/ budget alerts
* "local" admins could create/manage WHs there!

Introduction to Snowflake Editions

Tip #31: When to Choose Enterprise over Standard Edition

Tip #32: How to Avoid Multi-Cluster Warehouses

Tip #33: When to Use Incremental Materializations

Tip #34: How to Emulate Materialized Views

Tip #35: The Case for Extended Time Travel

Tip #36: Use Standard Edition Account for Analytics

Tip #37: Use Separate Standard Edition Account for Common Queries

Tip #38: How to Reduce Costs to Zero for an Inactive Paid Account

Tip #39: When to Choose the Business Critical Edition

Tip #40: When to Choose the Virtual Private Snowflake (VPS) Edition

When to Choose Enterprise over Standard Edition

Tip #31

Esential New Features

* **Multi-Cluster Virtual Warehouses** → w/ auto-scaling out for concurrency
* Query Acceleration Service → boost WH size Nx, for eligible queries
* Search Optimization Service → for point lookup queries
* Materialized Views → w/ auto-maintenance of results
* Extended Time Travel → max 90 days (from 1) for permanent tables

Other New Features

* Data Masking Policies → column-level security in tables/views
* Row Access Policies → row-level security for query rows
* Object Tagging → to track sensitive data + resource usage
* Classification → to auto-tag potentially sensitive data
* User Access Auditing → in ACCESS_HISTORY view
* Periodic Rekeying of Encrypted Data → increased protection
* Early Access to Weekly New Releases → 12-hour

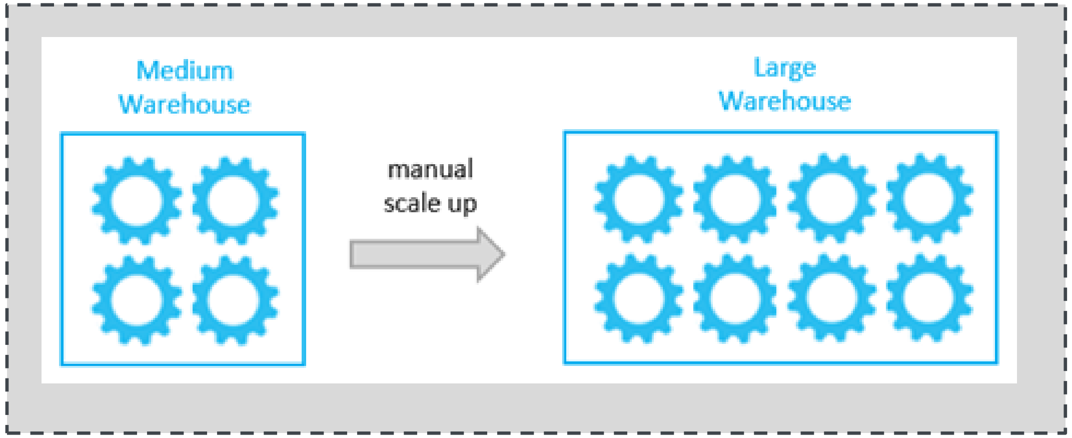
Checkpoints

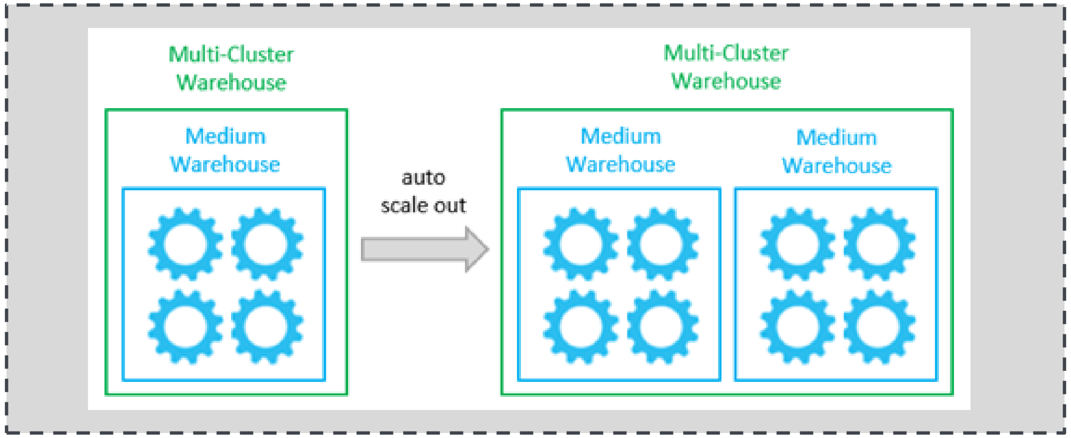
* SMB → large enterprise/org
* 1.5x credit cost
* higher number of users → multi-cluster WHs (concurrency)
* QAS + SOS + MVs → improved query performance
* 90 days TT → better data protection
* column/row policies, obj tagging, classif → better data gov
* rekey, user audit → better security

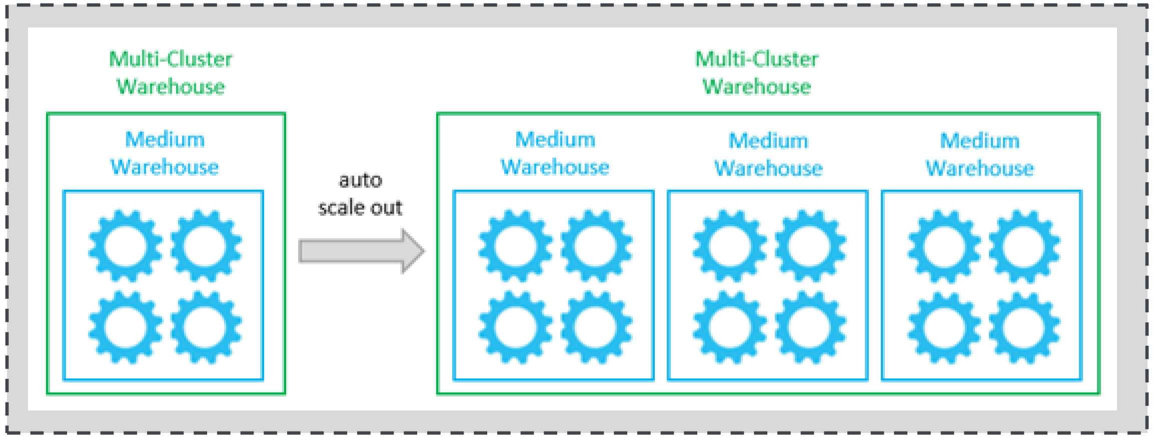
How to Avoid Multi-Cluster Warehouses

Tip #32

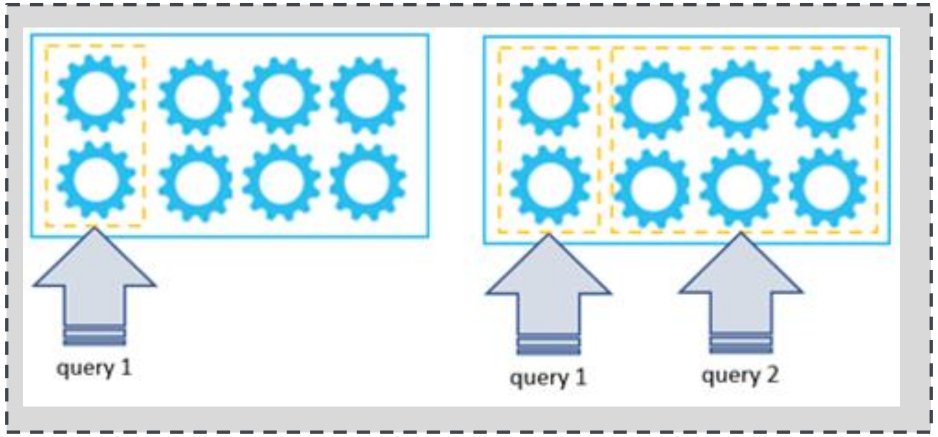
Scalability

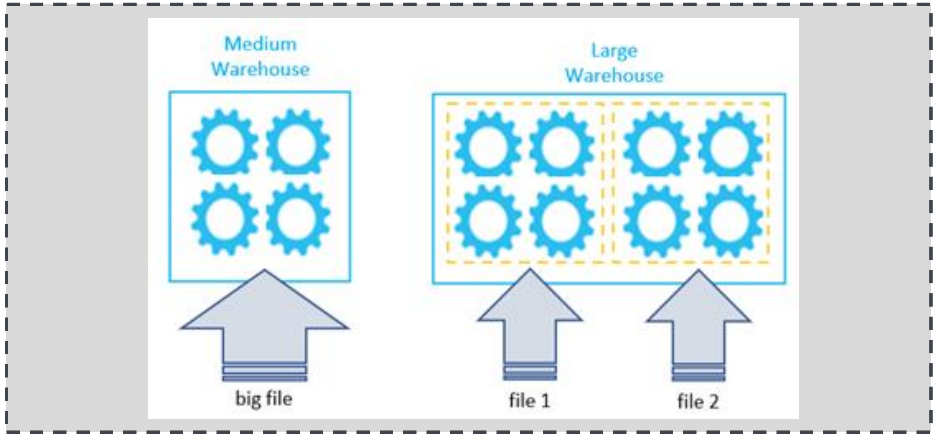


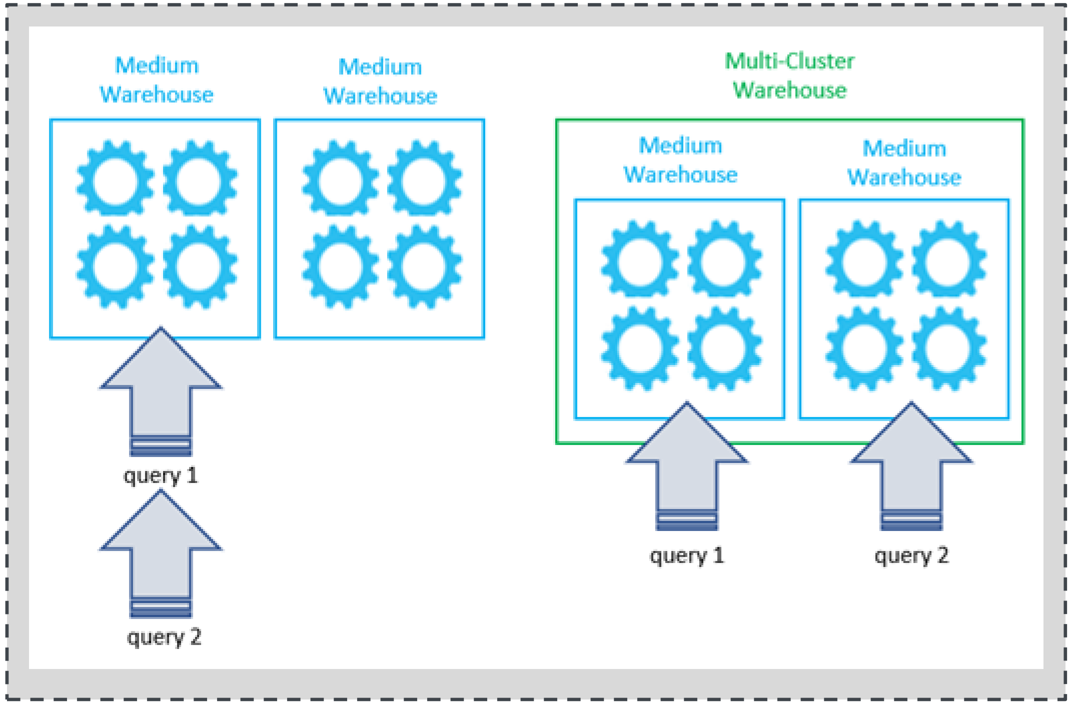




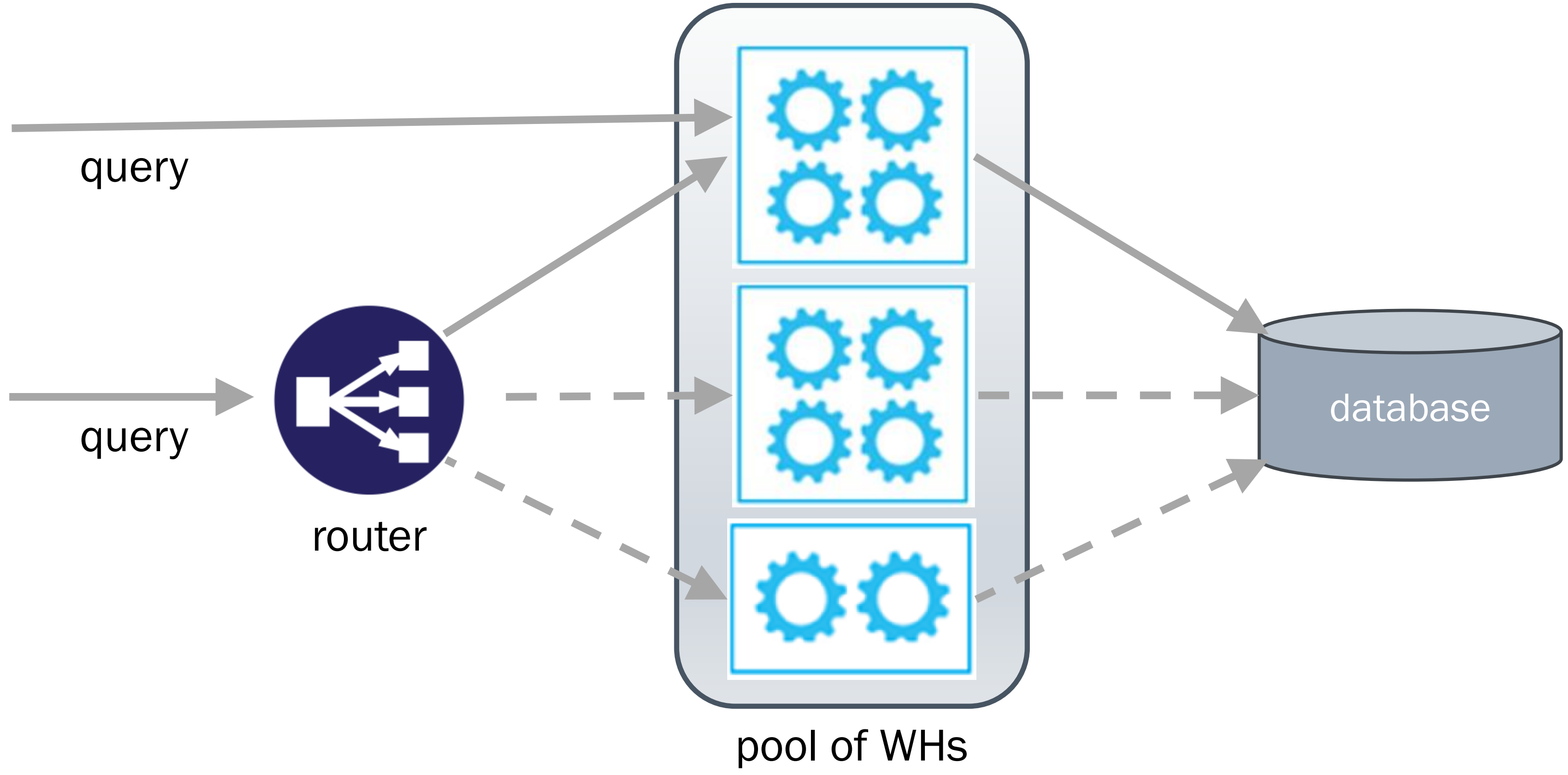
Concurrency







Custom Auto-Router



Checkpoints

* multi-cluster WHs (Enterprise+ only) improve concurrent access (avoid queuing)

* *custom auto-router* =~ load balancer, on standard WHs

* auto-scale out/in to standard overflow WHs (manual scale up/down normally)

* could reroute if M+ queries on current WH (use MAX_CONCURRENCY_LEVEL)

* auto-route work on load spikes (to existing/dynamic WHs w/ same/different sizes)

* reroute to pool of WHs (10+!) after STATEMENT_QUEUED_TIMEOUT_IN_SECONDS expires

* could use default auto-suspend times (multi-cluster could use longer time!)

When to Use Incremental Materializations

Tip #33

Materialized Views

* Enterprise+, storage + compute costs (10 credits/h) → when refreshed
* act more like tables, as they store R/O data → result never outdated
* base table data changes → back incremental update
* based on one single table (self-joins not allowed)
* can be secured (not optimized) + can have a (different) cluster key
* has limited SQL syntax, no UDFs

Checkpoints

* when query result doesn't change often (base table not freq updated)
* when results are asked often (less complex query executed here)
* when original query consumes lots of resources
* to improve performance on an external query
* when lots of aggregations returned
* when few rows constantly returned only from a very large table
* need a different cluster key than the base table

How to Emulate Materialized Views

Tip #34

Checkpoints

* use a table instead, instantly/periodically updated

* use a task to periodically update the table

* task + CHANGE_TRACKING enabled on the source table

* task + STREAM ← legacy mechanism for CDC

* use a Snowflake *dynamic table* instead ← new!

The Case for Extended Time Travel

Tip #35

Time Travel Use Cases

** looking back in time*

- SELECT … FROM … AT (TIMESTAMP => timestamp) …
- SELECT … FROM … AT (OFFSET => rel_time_diff) …
- SELECT … FROM … AT (STREAM => 'name') …
- SELECT … FROM … AT/BEFORE (STATEMENT => query_id) …

** in zero-copy cloning*

- CREATE DATABASE/SCHEMA/TABLE target CLONE source AT/BEFORE (…)

** restoring dropped objects*

- DROP DATABASE/SCHEMA/TABLE name
- SHOW DATABASES/SCHEMAS/TABLES HISTORY […] ← dropped
- UNDROP DATABASE/SCHEMA/TABLE name ← restore dropped object

Checkpoints

* *time travel*
    - max 90 days for permanent tables in Enterprise Edition
    - max 1 day for permanent tables in Standard Edition
    - max 1 day for transient/temporary tables (in any edition)

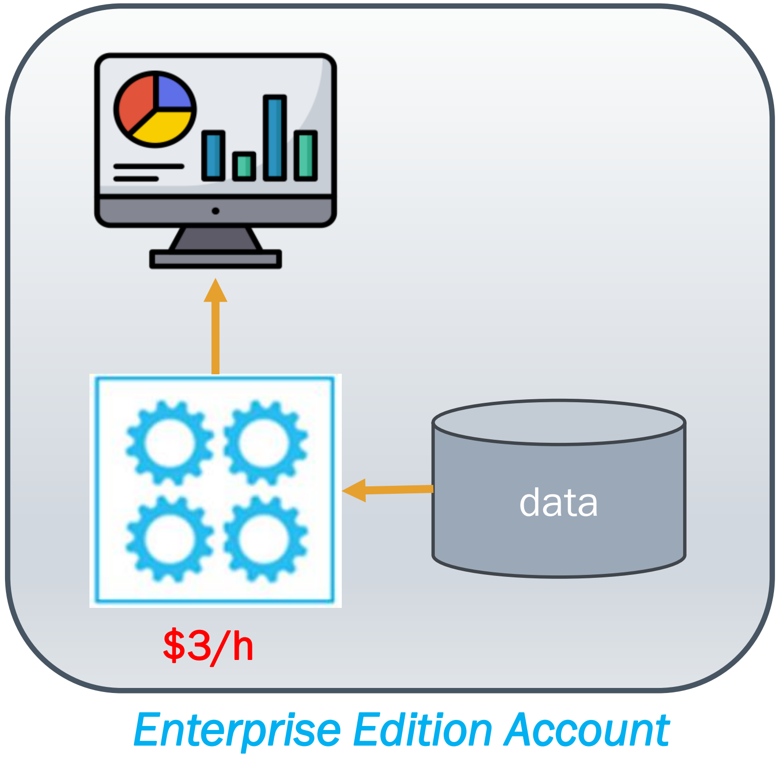
* for DATABASE → SCHEMA → TABLE

* DATA_RETENTION_TIME_IN_DAYS → in CREATE … / ALTER … SET …
    - set to 0 to disable → but cannot permanently disable in account!

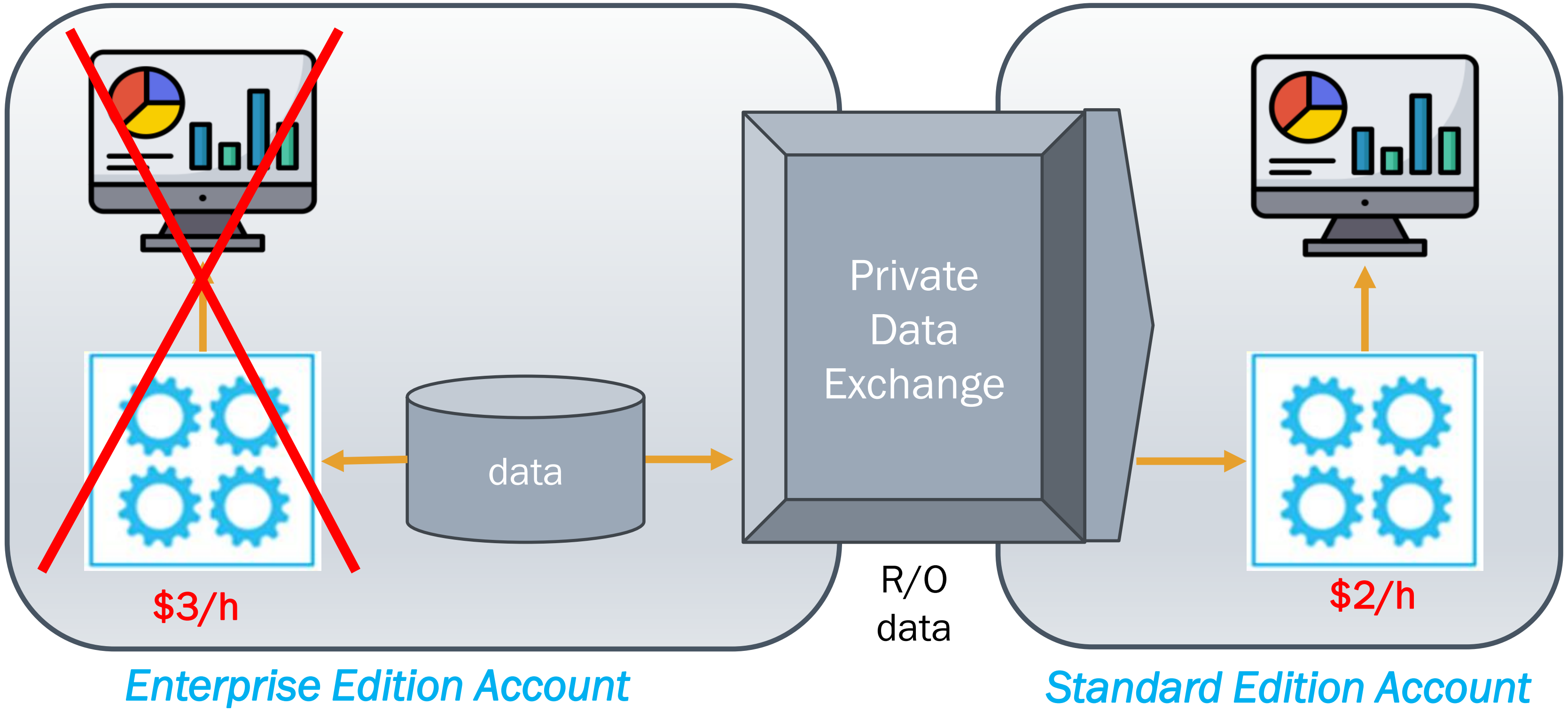
Use Standard Edition Account for Analytics

Tip #36

Analytics in Enterprise Edition



Analytics in Standard Edition



Checkpoints

* w/ private data share Enterprise → Standard accounts
* analytics in Standard Edition consume just $2/h credits
* save 30% on compute costs!
* *limitations*: multi-cluster WHs, mat views, QA, SO, clustering
* multi-cluster warehouses could be avoided
* keep data w/ mat views + clustering in Enterprise

Use Separate Standard Edition Account for Common Queries

Tip #37

Checkpoints

* any Organization must also have one Standard account
* for higher edition account → create also one Standard
* if no special features needed → run query on Standard
* *credit cost*: $2 Standard, $3 Enterprise, $4 Business
* no need for a multi-cluster warehouse if you're alone

How to Reduce Costs to Zero for an Inactive Paid Account

Tip #38

Checkpoints

* do not share it with other people
* keep one single 1-Node X-Small WH w/ auto-suspend 1 minute
* associate your user connection w/ this WH
* do not run long queries (Cancel them if they already take too long)
* do not have active/scheduled tasks running (better drop them all)
* do not leave any serverless features running

Checkpoints (2)

* drop any potential dangerous object after experiments, even if inactive (WHs, tasks)

* don’t check too much on metadata (in screen or by SQL)

* remove any app installed from the Marketplace after testing

* create explicit transitory tables w/ no time travel

* change default days for time travel to 0 at the account level

* do not store/copy/keep/transfer large data

When to Choose the Business Critical Edition

Tip #39

Enhanced Data Protection

* Failover/failback Snowflake accounts → w/ redir conns
* Tri-Secret Secure w/ CMKs
* AWS PrivateLink support
* Ext Funcs through AWS API Gateway Private Endpoints

Support for Regulations

* PHI Data Support (for HIPAA and HITRUST CSF)
  - for orgs w/ extremely sensitive data, as in healthcare
  - required signed business agreement w/ Snowflake
* PCI DSS Compliance (payment card industry)
* Support for public sector (FedRAMP and ITAR)
* Support for IRAP - Protected (P) data (in Asia Pacific)

Checkpoints

* former ESD (Enterprise for Sensitive Data)
* Enterprise+, 2x compute credit cost over Standard
* enhanced security and data protection
* failover/failback between Snowflake accounts
* for business continuity/disaster recovery
* support for regulations (PHI/HIPAA, PCI, US Federal)
* combine w/ Standard/Enterprise for reg queries → 50% cost savings!

When to Choose the Virtual Private Snowflake (VPS) Edition

Tip #40

New Features

* completely separate Snowflake environment

* isolated from all other Snowflake accounts

* dedicated virtual servers (w/ in-memory encryption key)

* pool of compute resources (used in VWs)

* dedicated metadata store

* secure data share (w/ Marketplace/Data Exchange) disabled by def

* call support to enable data sharing with non-VPS customers

Credit Cost

* Standard: $2.00 - $3.10
* Enterprise: $3.00 - $4.65
* Business Critical: $4.00 - $6.20
* VPS: $6.00 - $9.30

Checkpoints

* Business Critical+, 2x credit cost of Enterprise ($6+ in AWS/US)

* completely isolated/separated account/env → highest level of security

* dedicated servers, metadata, pool of VW resources

* secure data share disabled by def, but can enable

* 24-hour early access to weekly new releases

* for orgs w/ strictest reqs: financial inst., large ent. w/ highly sensitive data

Introduction to Query Monitoring

Tip #41: Monitor Longest Running Queries

Tip #42: Interpret Query History

Tip #43: More Charts for Query Monitoring

Tip #44: Use Query Tags

Tip #45: Reduce Frequency of Simple Queries

Tip #46: Reduce Frequency of Metadata Queries

Tip #47: Reduce Frequency of SHOW Commands

Tip #48: Clone Less Frequently

Tip #49: Change Query Schedules

Tip #50: Parallel over Sequential Transfer and Processing

Monitor

Longest Running

Queries

Tip #41

Checkpoints

* from **snowflake.account_usage.QUERY_HISTORY** view
* top N w/ longest individual *execution_time*
* top N w/ longest AVG exec time, by *type/WH size/user/...*
* top N query types w/ most consumed *credits*
* top N w/ longest *total* execution time overall
* top N most *frequently executed* queries

Interpret

Query History

Tip #42

Checkpoints

* Monitoring - Query History ← in Snowsights (14 days)
* ACCOUNT_USAGE.QUERY_HISTORY view (1 year)
  - *total_elapsed_time* = compilation + execution + queued…
    - queued_provisioning/repair/overload_time
    - transaction_blocked_time
  - *transferred bytes* = …
* INFORMATION_SCHEMA.QUERY_HISTORY[_BY…] functions (7 days)
  - filter by session/user/warehouse
  - less info

More Charts for Query Monitoring

Tip #43

Checkpoints

* histogram of queries duration - in seconds + log scale

* zoom into top-3 longest queries in detail

* time-histograms of aggregate queries duration - in seconds

* day-hour histogram

* longest and most frequent queries - with log scales

Use

Query Tags

Tip #44

Checkpoints

* QUERY_TAG = session-level param, 2k chars max, can use JSON
  - ALTER SESSION SET QUERY_TAG = '...' ← w/ SQL
  - session_parameters={'QUERY_TAG': '...'}) ← w/ Python Connector
  - session.query_tag = '...' ← w/ Snowpark
* filter QUERY_TAG column in QUERY_HISTORY
* alternative: end query w/ '-- comment'
* not a session variable (no SET QUERY_TAG = ...)
* no object tagging (no CREATE TAG ...)

Reduce Frequency of Simple Queries

Tip #45

Checkpoints

* ~10K calls/day → significant cloud service usage
* for SELECT 1 / seq1.NEXTVAL / CURRENT_SESSION()
* session ID does not change while a connection is open
* could save CURRENT_SESSION() value after first call
* JDBC SnowflakeConnection.*getSessionId*() uses cache!
* could use a dashboard chart, to monitor

Reduce Frequency of Metadata Queries

Tip #46

Checkpoints

* ~10K calls/day → significant cloud service usage

* query INFORMATION_SCHEMA → uses cloud services

* query ACCOUNT_USAGE → uses a virtual warehouse

* ACCOUNT_USAGE has more data → 1y, for account

* could use a dashboard chart, to monitor

Reduce Frequency of SHOW Commands

Tip #47

SHOW for RBAC

SHOW ROLES → builtin+custom roles

SHOW GRANTS TO ROLE … → assigned roles+privs

SHOW USERS → users

SHOW GRANTS TO USER … → assigned roles

SHOW for ERDs

SHOW TABLES → tables

SHOW COLUMNS → table columns

SHOW PRIMARY KEYS → PRIMARY KEY constraints

SHOW UNIQUE KEYS → UNIQUE constraints

SHOW IMPORTED KEYS → FOREIGN KEY constraints

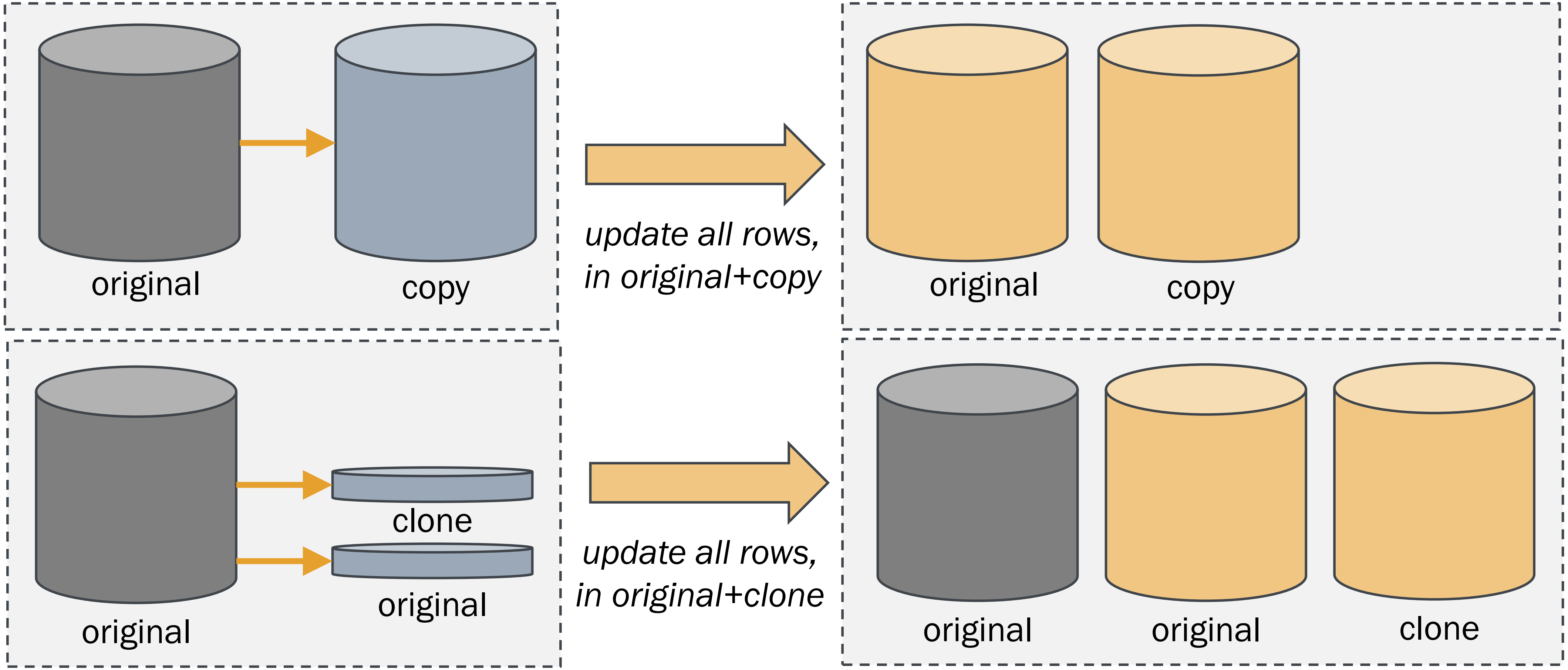
Checkpoints

* SHOW cmds → consume only cloud service resources
* avoid SHOW cmds at a high frequency
* check third-party tools, on top of Snowflake
* check partner tools
* could use a dashboard chart, to monitor

Clone Less Frequently

Tip #48

Changing Most Rows in Common



Checkpoints

* extensive DDL (cloning) → significant cloud serv usage
* DDL opers (like cloning) use only cloud compute serv

* do not frequently create/drop large schemas/tables
* do not frequently clone databases for backup
* avoid cloning if you intend to change most rows in common
* updating most rows in common will increase total storage space

Change Query Schedules

Tip #49

Cost

* *task*
  - scheduled every 2 mins, runs for max 2 secs each time
  - (24h x 60 mins / 2 mins) x 2 secs = 1440 secs = 24min/day
  - 24min/day for a related WH → 0.5 credits/day = $1.5/day
* *warehouse*
  - X-Small, 1 credit/h ($3/credit), auto-suspended after 1 min inactivity
  - resumed by the task every 2 mins = 30mins/h = 12h/day
  - 24h x 30 mins = 720min = 12h → $36/day

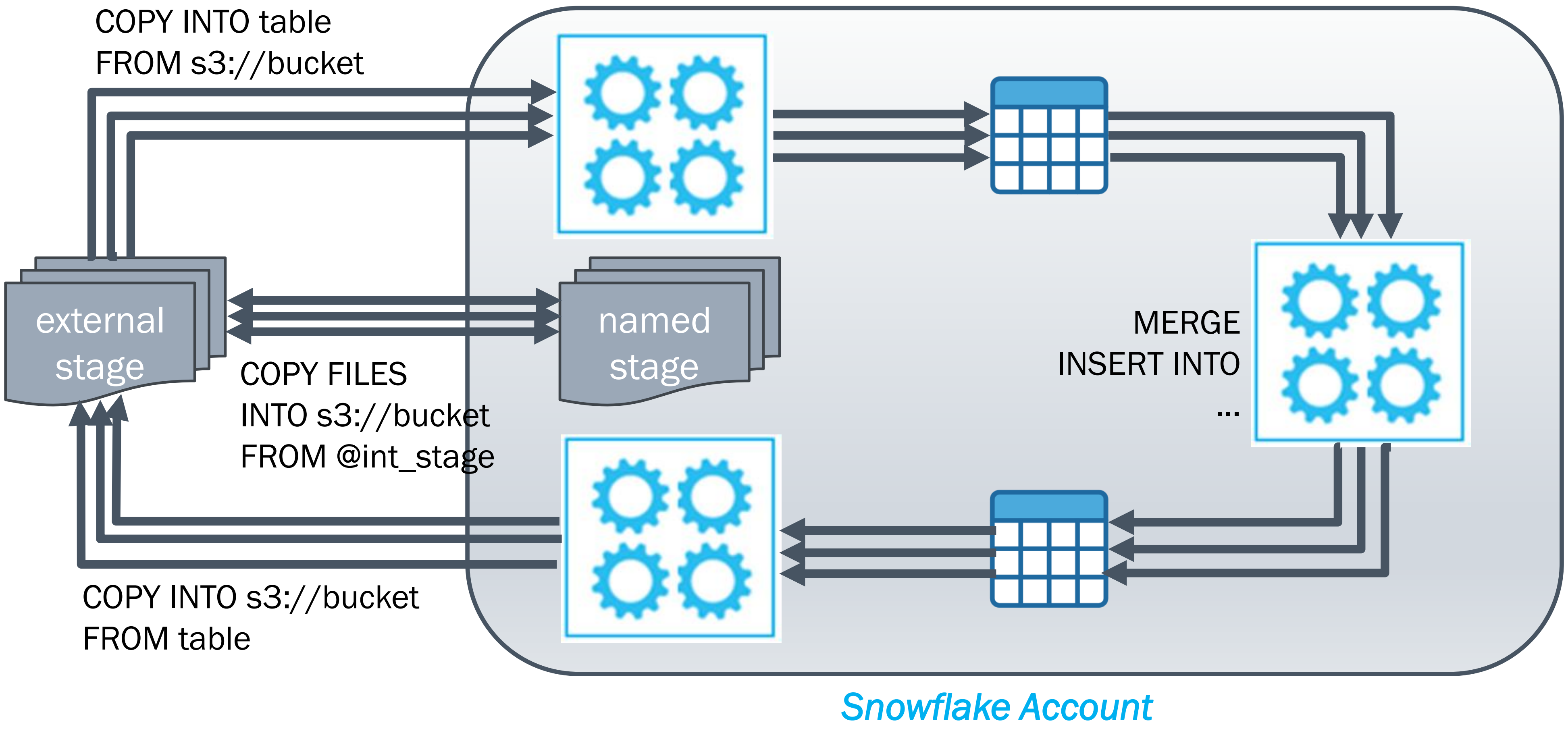
Checkpoints

* frequently scheduled tasks not ok w/ managed WHs
* you may pay more for the WH idle, suspending (1 min)
* *solution*: try using a WH already up, w/ other queries
* adjust eventually the schedule, to find such a WH up
* avoid up-down pattern → exec more queries together
* consider a serverless task (no WH, but 1.x the price)

Parallel over Sequential Transfer and Processing

Tip #50

Parallel Processing



Checkpoints

* use parallel query processing → max_concurrency_level at WH
* COPY INTO table from multiple split input files
* COPY INTO @stage w/ multiple split output files
* COPY FILES w/ multiple files, from stage to stage
* MERGE over individual INSERT/UPDATE/DELETE
* INSERT INTO multiple tables w/ parallel processing
* batch transformations w/ multiple SQL connections

Introduction to Query Optimization

Tip #51: Use the Query Profile

Tip #52: Use the Explain Statement

Tip #53: Use Data Caching

Tip #54: Queries on Data Lakes

Tip #55: Use Vectorized Python UDFs

Tip #56: Use Batch Commands to Prevent Transaction Locks

Tip #57: Reduce Query Complexity and Compilation Time

Tip #58: Check for Cross Joins and Exploding Joins

Tip #59: Process Only New or Updated Data

Tip #60: Remote Spillage Optimization

Use the Query Profile

Tip #51

Checkpoints

* generates graph of the actual execution plan
* can use for running queries as well
* based on the generated query ID
* carefull w/ queries executed from cache
* great for TableScan and Join steps ← most costly
* evaluate partitions scanned/total + bytes scanned
* check number of rows returned by a Join → exploding joins?

Use the Explain Statement

Tip #52

Checkpoints

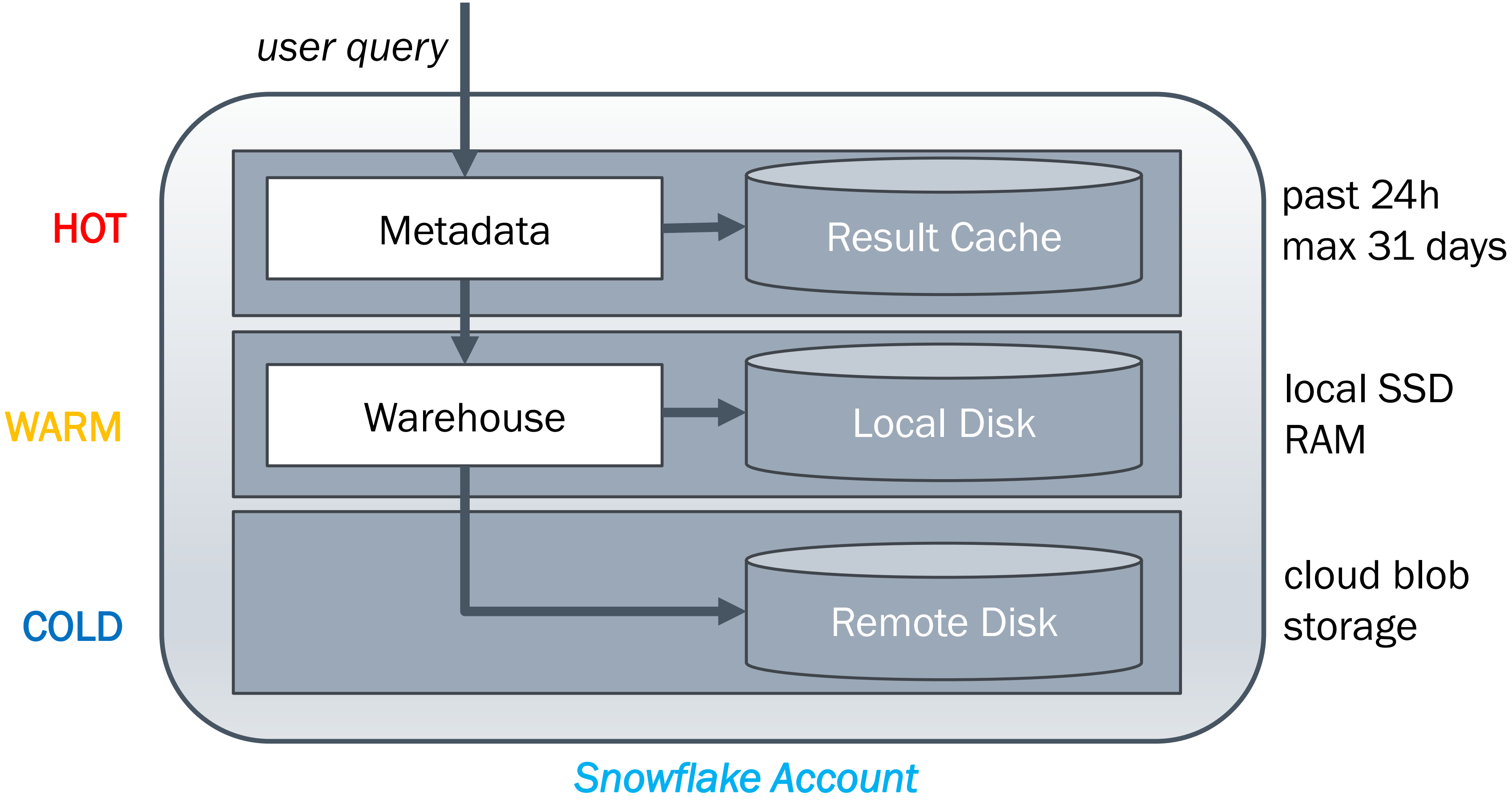
* get the logical execution plan without running the query
* estimates micro-partitions to scan and total bytes
* USING TEXT/JSON → hierarchy of logical steps
* TableScan steps could use lots of data
* consider columns used for Automatic Clustering
* see when mat views could be scanned in the back
* could use as query syntax check

Use

Data Caching

Tip #53

Query Result Caching



Checkpoints

HOT

- result cache on
- cache hit → SELECT * FROM TABLE(RESULT_SCAN(LAST_QUERY_ID()))

WARM

- result cache off or no cache hit
- warehouse up and result still in the warehouse (in RAM or local disks)
- get query result from warehouse cache (BYTES_SCANNED > 0)

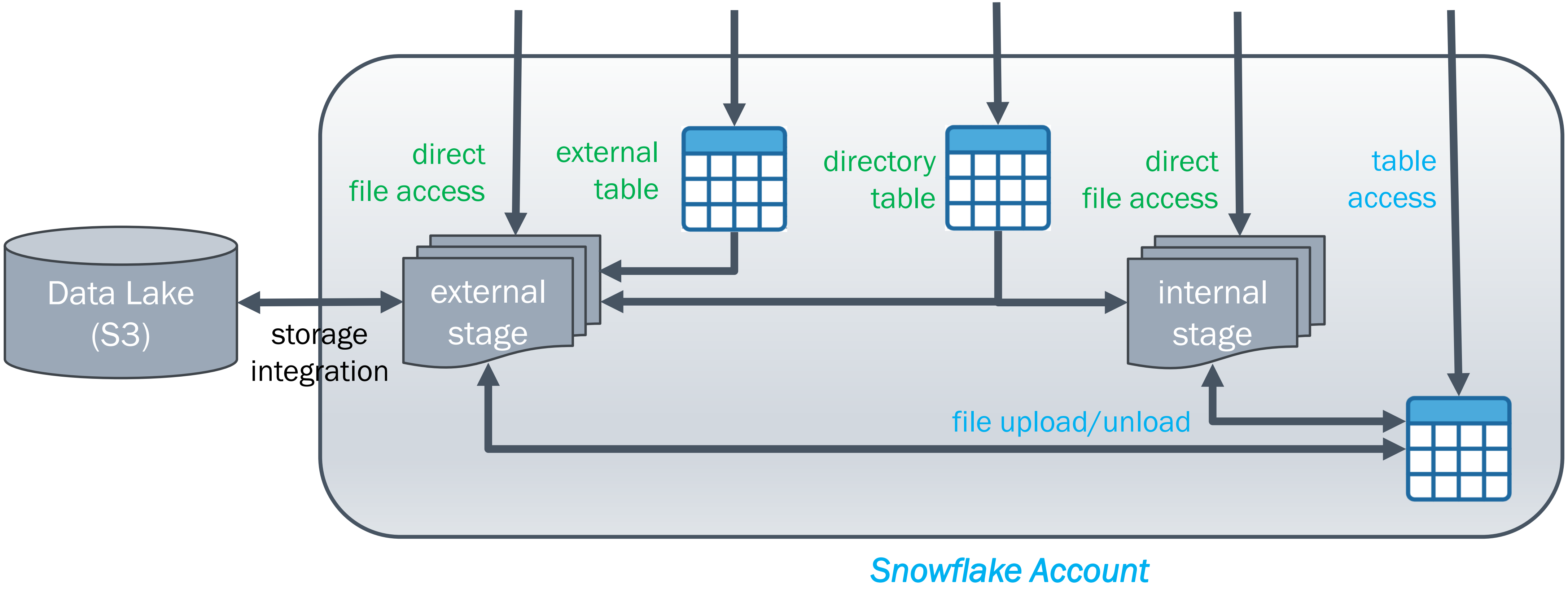
COLD

- no warehouse cache (warehouse suspended, not up)
- result cache off (ALTER SESSION SET USE_CACHED_RESULT = false) or no cache hit
- must run query (could also be a new query)

Queries on Data Lakes

Tip #54

Data Warehouse vs Data Lake



Checkpoints

* could be better to leave files in stage and query from there

* *data warehouse*

- *file upload/unload* → between stage files and tables (partitions!)
- *table access* → typical SQL on stored Snowflake data

* *data lake*

- *direct file access* → on internal/external stage files ($1, $2...)
  - JSON in VARIANT is more optimized!
  - Parquet vs CSV: 50x+ times faster!
- *external tables* → on external stage (w/ column defs!)
- *directory tables* → for unstructured data (refs only in the database!)

Use Vectorized Python UDFs

Tip #55

Checkpoints

* row-by-row processing → batches of rows (better performance!)

* non-vectorized UDF = scalars → UDF → scalar return

* vectorized UDF = scalars → pandas DataFrames → UDF → pandas arrays/Series → scalar returns

@vectorized(input=pandas.DataFrame)

optional max_batch_size

Use Batch Commands to Prevent Transaction Locks

Tip #56

Checkpoints

* UPDATE/MERGE → transaction lock on a table
  - other cmds cannot exec on table until lock released
  - queries consume cloud serv credits waiting for lock
* *solution*: use a batch cmd instead of single updates
  - use multi-row batch INSERT in temporary stage table
  - use task to periodically UPDATE/MERGE into target table

Reduce Query Complexity and Compilation Time

Tip #57

Cost

* *object dependencies query*
  - compilation: 19 secs
  - execution: 5.6 secs
  - rows returned: 0
* *data lineage query*
  - compilation: 1.2 secs
  - execution: 4.2 secs
  - rows returned:1.4K

Checkpoints

* *complex queries*

- can consume significant cloud services compute resources
- will have high compilation times
- see *compilation_time* vs *execution_time* in QUERY_HISTORY

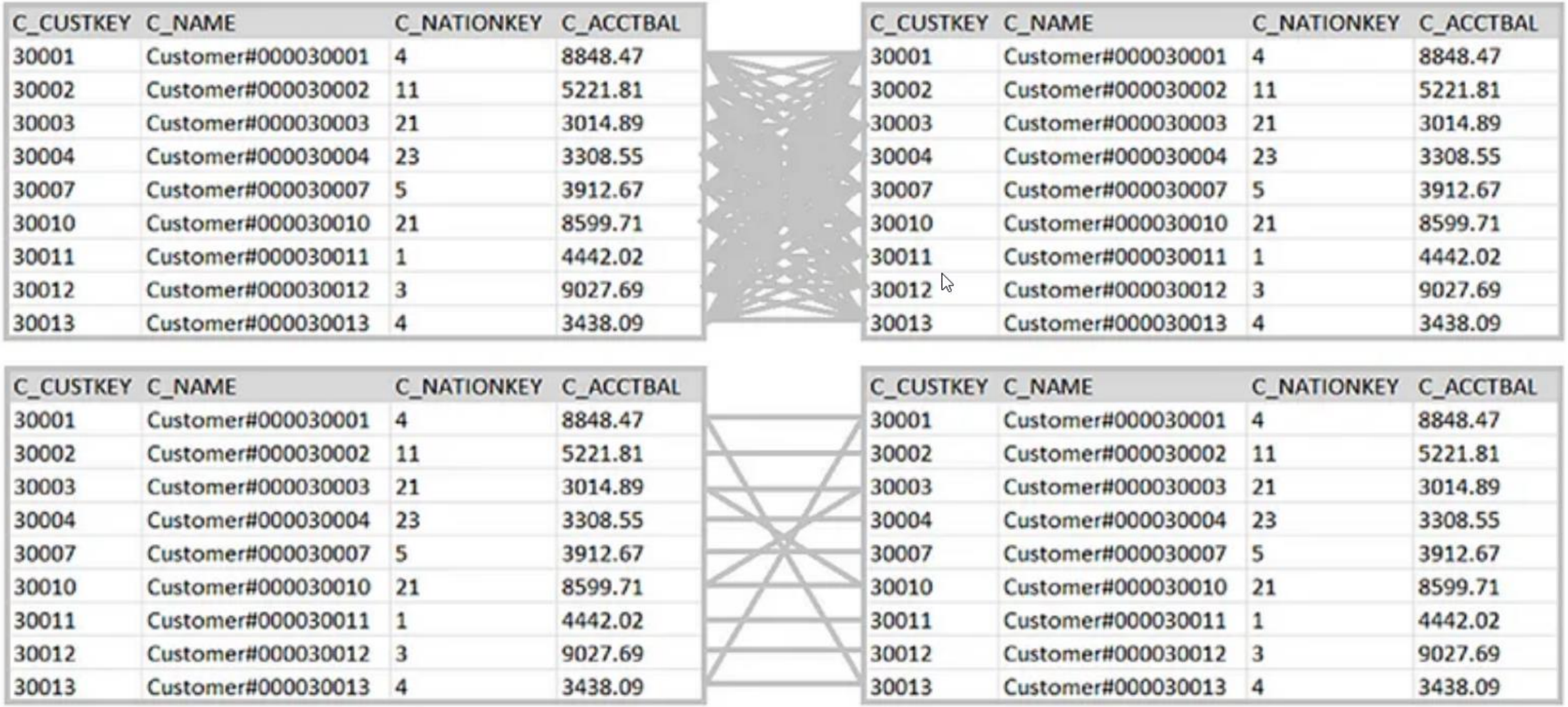
* *examples*

- queries with lots of joins
- queries with cartesian products
- queries with IN operator and large lists
- very large queries

Check for Cross Joins and Exploding Joins

Tip #58

Cross Joins vs Exploding Joins



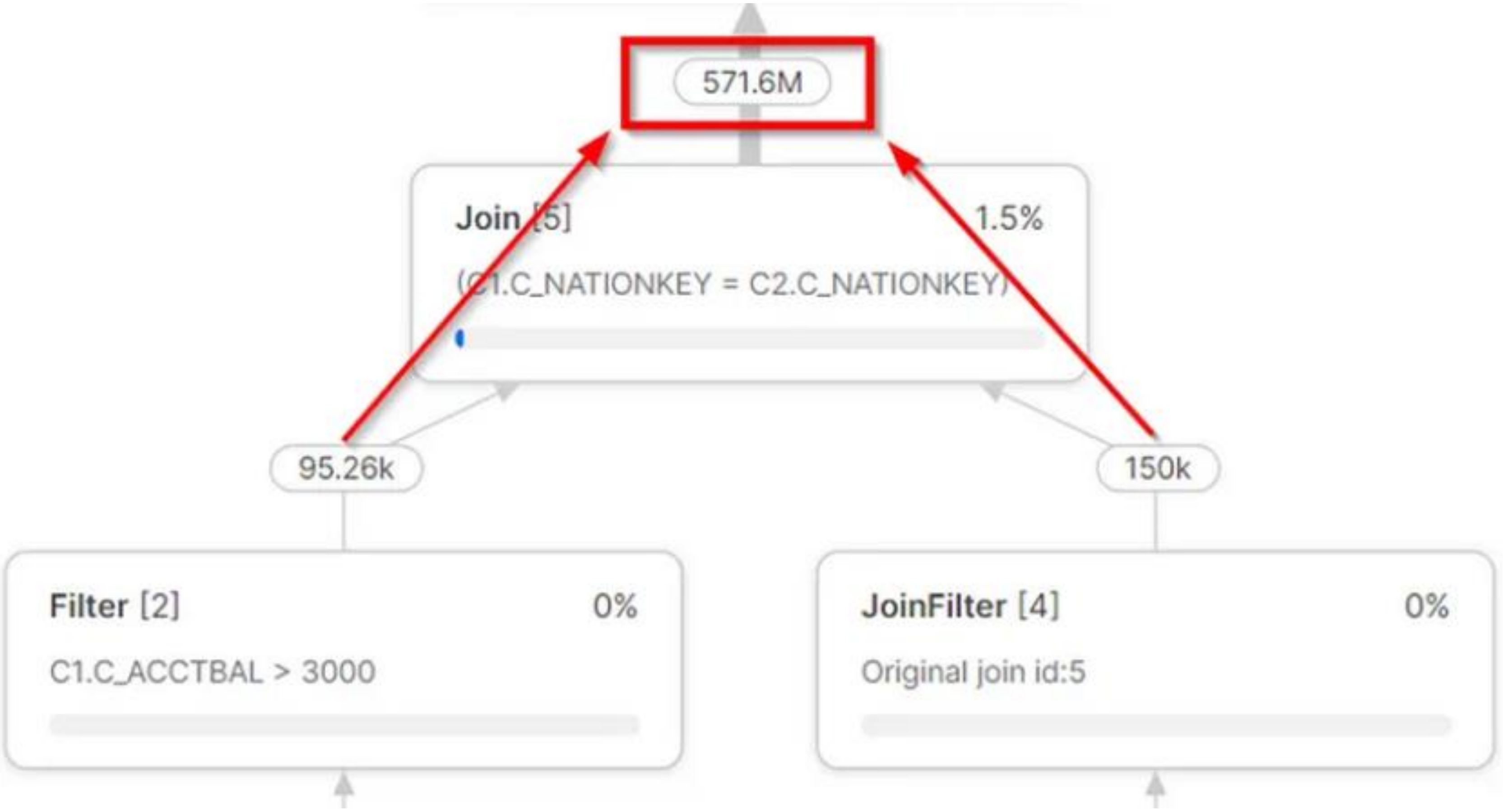
C_CUSTKEY	C_NAME	C_NATIONKEY	C_ACCTBAL
30001	Customer#000030001	4	8848.47
30002	Customer#000030002	11	5221.81
30003	Customer#000030003	21	3014.89
30004	Customer#000030004	23	3308.55
30007	Customer#000030007	5	3912.67
30010	Customer#000030010	21	8599.71
30011	Customer#000030011	1	4442.02
30012	Customer#000030012	3	9027.69
30013	Customer#000030013	4	3438.09

C_CUSTKEY	C_NAME	C_NATIONKEY	C_ACCTBAL
30001	Customer#000030001	4	8848.47
30002	Customer#000030002	11	5221.81
30003	Customer#000030003	21	3014.89
30004	Customer#000030004	23	3308.55
30007	Customer#000030007	5	3912.67
30010	Customer#000030010	21	8599.71
30011	Customer#000030011	1	4442.02
30012	Customer#000030012	3	9027.69
30013	Customer#000030013	4	3438.09

C_CUSTKEY	C_NAME	C_NATIONKEY	C_ACCTBAL
30001	Customer#000030001	4	8848.47
30002	Customer#000030002	11	5221.81
30003	Customer#000030003	21	3014.89
30004	Customer#000030004	23	3308.55
30007	Customer#000030007	5	3912.67
30010	Customer#000030010	21	8599.71
30011	Customer#000030011	1	4442.02
30012	Customer#000030012	3	9027.69
30013	Customer#000030013	4	3438.09

C_CUSTKEY	C_NAME	C_NATIONKEY	C_ACCTBAL
30001	Customer#000030001	4	8848.47
30002	Customer#000030002	11	5221.81
30003	Customer#000030003	21	3014.89
30004	Customer#000030004	23	3308.55
30007	Customer#000030007	5	3912.67
30010	Customer#000030010	21	8599.71
30011	Customer#000030011	1	4442.02
30012	Customer#000030012	3	9027.69
30013	Customer#000030013	4	3438.09

Exploding Joins in Query Profile



Joins

* *one-to-one joins*: 95.26K rows

* *cross joins*: 95.26K * 150K = **9B+ rows** (cancelled)

* ***exploding joins***: < 95.26K * 150K = **571.6M rows**

customer → 150K rows

customer w/ c_acctbal > 3000 → 95.26K rows

Checkpoints

* exploding joins = inner join w/ huge number of matches
* ~cross joins/cartesian product → large number of rets
* returns close to N x M rows (N, M = rows of tables)
* more related to many-to-many relationships
* detect by looking at the Query Profile

Process Only New or Updated Data

Tip #59

Checkpoints

* only the changed data gets processed → reduce compute costs

* avoid transferring the whole batch of data, daily → when most data did not change

* CDC (Change Data Capture) → "play" transaction log for new changes only

* *incremental processing*: do not rerun entire batches of data for minor updates

* restart interrupted pipelines from point of interruption → no redundancies

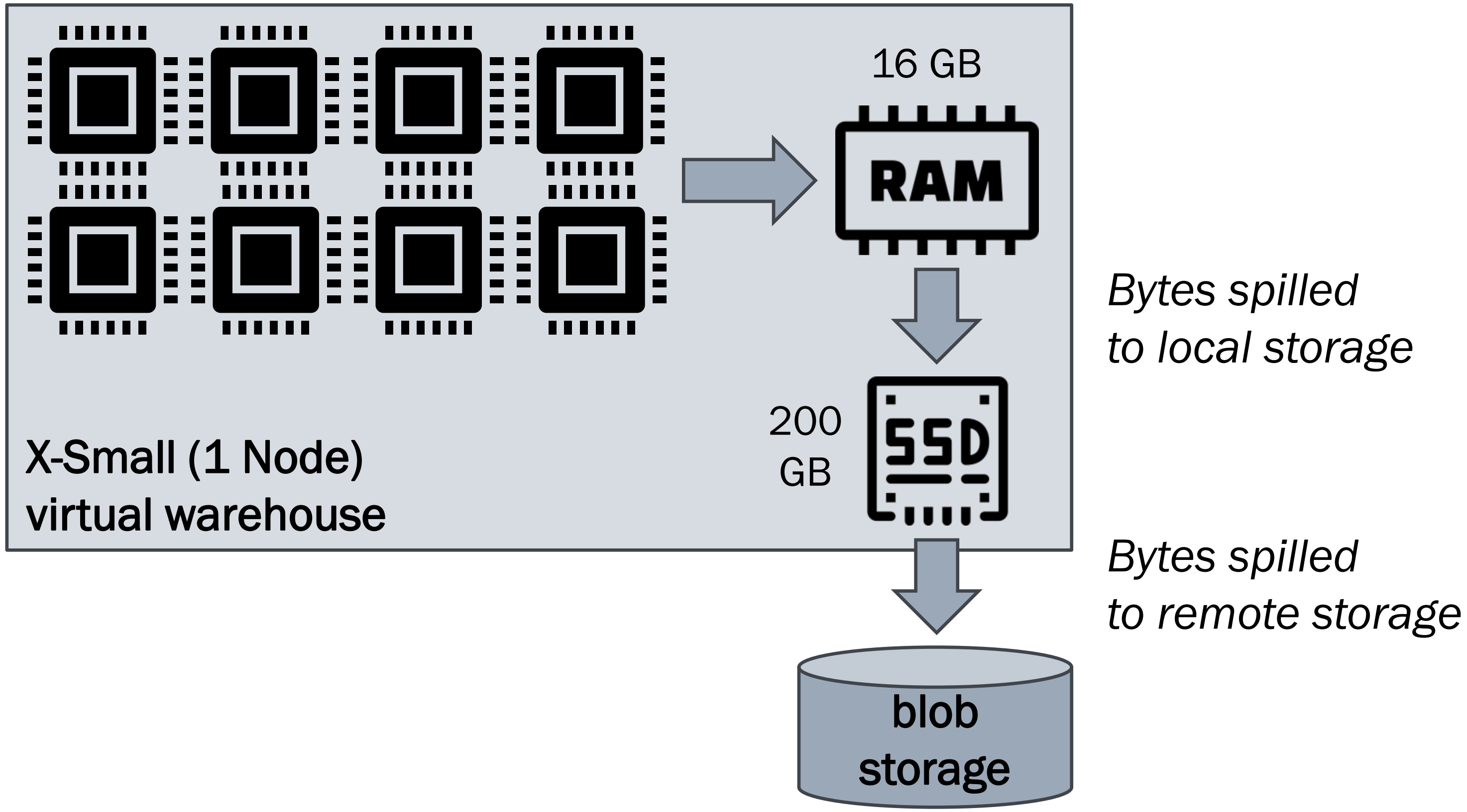
* on failed data uploading → keep successful loaded data, continue w/ failed

* on failed batch data transformations → partition and replay for failed ranges only

Remote Spillage Optimization

Tip #60

Spilling to Local/Remote Storage



Cost

* Large (8 Nodes): 2 mins, 60 GB spilled to local

* Snowpark-optimized Large (16x memory): 1:30m, ~3 GB spilled to local

on 48.6 GB table

Checkpoints

* Bytes spilled to local storage = RAM → SSD (all in WH!)
* Bytes spilled to remote storage = SSD → blob storage (slowest!)
* on SSD = intermediate results for ORDER BY, GROUP BY, PIVOT…
* RAM (fastest + most expensive) >> SSD (disk) >> blob storage (network)
* *solution*
  - use a bigger WH (= 2x RAM+SSD!) - Snowpark-optimized (16x RAM) or QAS as alternative
  - use less data
* check *bytes_spilled_to_local/remote_storage* in QUERY_HISTORY

Introduction to Serverless Features

Tip #61: Monitor the Cost of Automated Jobs

Tip #62: Estimate Cost of Scheduled Tasks

Tip #63: When to Use Serverless Tasks

Tip #64: Replace Snowpipe with Snowpipe Streaming

Tip #65: Estimate Cost of Automatic Clustering on Tables

Tip #66: Estimate Cost of the Query Acceleration Service (QAS)

Tip #67: Estimate Cost of the Search Optimization Service (SOS)

Tip #68: Reduce Materialized Views Maintenance Cost

Tip #69: Reduce Database Replication Cost

Tip #70: Estimate Cost of Hybrid Tables

Monitor the Cost of Automated Jobs

Tip #61

Cost

Table 5: Serverless Feature Credit Table

Feature	Snowflake Credits per Compute-Hour	
	Snowflake-managed compute	Cloud Services
Clustered tables	2	1
Copy Files[10]	2	N/A
Hybrid Tables requests[10]	1	1 In addition: 1 Credit per 30GB read 1 Credit per 7.5GB write
Logging	1.25	1
Materialized views maintenance	10	5
Materialized views maintenance in secondary databases	2	1
Query acceleration	1	1
Replication	2	1
Search optimization service	10	5
Search optimization service in secondary databases	2	1
Serverless tasks	1.2	1
Snowpipe	1.25	N/A but charged 0.06 Snowflake Credits/1000 Files
Snowpipe Streaming	1	N/A but charged at an hourly rate of 0.01 Snowflake Credits per client instance

METERING_HISTORY View

* *start_time* .. *end_time* = hourly credit usage, for 1y

* *service_type* + *name*

* *credits_used_compute* = WH + serverless

* *credits_used_cloud_services* = ~1 credit/h (10 for MVs/SO)

* *credits_used* = *credits_used_compute* + *credits_used_cloud_services*

* *bytes* + *rows* + *files* – for auto-clustering / pipe / snowpipe_streaming

Service Types

* WAREHOUSE_METERING[_READER] – any edition, w/ WH name [+for Reader accounts]

* SERVERLESS_TASK – not user-managed, ~1.5x more expensive (1.2x now)

* AUTO_CLUSTERING – any edition, auto-background maintenance of clustered table/MV (+initial clustering+reclustering), ~2 credits/h

* MATERIALIZED_VIEW – 10 credits/h [2 in sec dbs] + 5 for cloud → storage+compute (per second). when base table changes → MVs updated by serverless back service

* QUERY_ACCELERATION – scale factor to limit cost → N nodes x boost

* SEARCH_OPTIMIZATION – 10 credits/h [2 in sec dbs] + 5 for cloud → storage+compute maintenance (higher cost when large table data changes + deletes + auto-clustering)

Service Types (cont.)

* PIPE + SNOWPIPE_STREAMING — Snowpipe ~1.25 credits/h + 0.06 credits/1000 files. Streaming w/ 2 table+service entries, 1 credit/h + 0.01 credits/h/client.

* HYBRID_TABLE_REQUESTS — 1 credit/h per req + 1 cloud credit per 30GB R/7.5GB W

* COPY_FILES — for COPY INTO between stages, 2 credits/h (no cloud)

* REPLICATION — for dbs/shares (any edition) / failover across accounts (BC) – 2 credits/h

* SNOWPARK_CONTAINERS — SPCS, all editions → costs for storage (image repository+logs+mounting volumes) + compute pool (1+ VM nodes) + data transfer (outbound + cross-AZ)

* AI_SERVICES — compute cost for Cortex LLM functions → per number of tokens (~4 chars) processed by the calls

Checkpoints

* automated jobs → ~70% of all compute cost

* credits per compute-hour

* credits used = Snowflake-managed compute + cloud services

* *cloud services*: for AuthN / infra/metadata management / query parsing+optimization / access control…

* METERING_HISTORY view

Estimate Cost of Scheduled Tasks

Tip #62

Cost

* *managed task*: 1 sec (exec time) x 60 x 24 = <30 min → <**$1.50**/day (estimated)

* *warehouse*: 1 min (up every time) x 60 x 24 = 24 h → $3 x 24h = **$72**/day

user-managed task w/ 1 sec total exec time + executed every minute

X-Small dedicated WH w/ 1 min auto-suspend, at $3/h/node credit (in AWS/US)

Checkpoints

* *user-managed* (w/ WH) vs *serverless* tasks (no WH)
  - USER_TASK_MANAGED_INITIAL_WAREHOUSE_SIZE → def MEDIUM!
* WHEN sql_condition → just cloud service credits!
  - task not triggered at all if WHEN not satisfied
* SCHEDULE → not for child tasks, w/ manual EXECUTE TASK if none
* USER_TASK_TIMEOUT_MS → def 1h (max 1 day)
  - STATEMENT_TIMEOUT_IN_SECONDS → sometimes w/ higher priority, in WH
* information_schema.**TASK_HISTORY**()
* check orphan tasks running but not being used

When to Use Serverless Tasks

Tip #63

Cost

* *serverless task*: 1 sec (exec time) x 60 x 24 = <30 min → <**$1.80**/day

* *managed task*: 1 sec (exec time) x 60 x 24 = <30 min → <**$1.50**/day (estimated)

* *warehouse*: 1 min (up every time) x 60 x 24 = 24 h → $3 x 24h = **$72**/day

serverless task w/ 1 sec total exec time + executed every minute → $3.6/h/node

user-managed task w/ 1 sec total exec time + executed every minute

X-Small dedicated WH w/ 1 min auto-suspend, at $3/h/node credit (in AWS/US)

Checkpoints

* favor serverless tasks for *frequently executed* + *short-running* operations

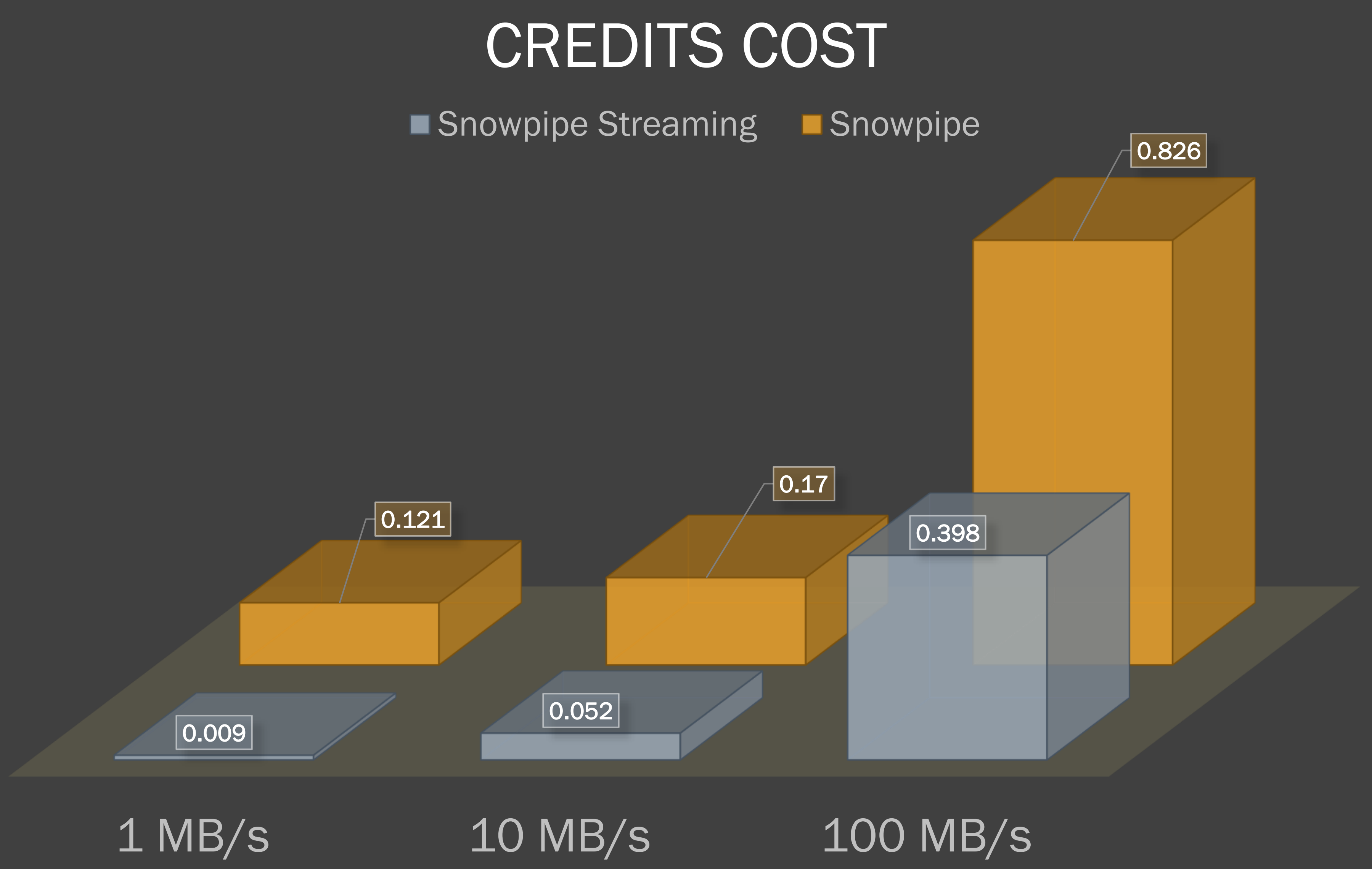
* try grouping more user-managed tasks on the same WH → WH consolidation

* *user-managed* (w/ WH) vs *serverless* tasks (no WH, 1.2x more expensive)
  - USER_TASK_MANAGED_INITIAL_WAREHOUSE_SIZE → def MEDIUM!

* information_schema table functions
  - SERVERLESS_TASK_HISTORY() → for serverless tasks
  - TASK_HISTORY() → for user-managed tasks
  - WAREHOUSE_METERING_HISTORY() → for warehouses

Replace Snowpipe with Snowpipe Streaming

Tip #64


CREDITS COST
Snowpipe Streaming
Snowpipe
0.826
0.121
0.17
0.398
0.009
0.052
1 MB/s
10 MB/s
100 MB/s

Checkpoints

* Snowpipe + Snowpipe REST API
  - old legacy service + API for streaming in small batches
* Snowpipe Streaming API
  - more cost-effective data ingestion + save on storage costs
  - high-throughput + low-latency streaming + low cost
  - automated service (no VW)
  - uses managed-compute resources + fixed credit charge per file

Estimate Cost of Automatic Clustering on Tables

Tip #65

Checkpoints

* *Automatic Clustering* = serverless, in any edition
* no clustering keys → no Automatic Clustering costs
* Clustered Tables serverless price = 2 credits/hour
* AUTOMATIC_CLUSTERING_HISTORY view/function
* AUTOMATIC_CLUSTERING Snowflake WH
* avoid table fragmentation → check load+query patterns

Estimate Cost of the Query Acceleration Service (QAS)

Tip #66

Cost

w/o QAS: 8min x 5c/min = **40c**

w/ QAS: 40sec x 5c/min x 14 nodes =< **47c**

*X-Small WH (1 node) = **$3**/h = 5c/min*

*QAS 14x scale factor =~ **$3**/h/node = 5c/min/node*

Programming QAS

* ENABLE_QUERY_ACCELERATION - in WH, to enable
* QUERY_ACCELERATION_MAX_SCALE_FACTOR - ctrls boost
* SYSTEM$ESTIMATE_QUERY_ACCELERATION - w/ qID
* QUERY_ACCELERATION_ELIGIBLE - system view
* QUERY_HISTORY - system view, w/ data
* QUERY_ACCELERATION_HISTORY - in info schema, for cost

Checkpoints

* Query Acceleration Service (QAS) = serverless, on WH
* can accelerate parts of a WH query workload
* parts of query processing → shared compute resources
* billed per second, no minimum
* no cost if enabled + no eligible queries
* in some cases could cost less than scalling up the WH

Estimate Cost of the Search Optimization Service (SOS)

Tip #67

Search Optimization Service

* for some lookup/analytical queries/views

* for queries w/ filters/joins, running for seconds+

* not on the primary cluster key

* at least one column must have 100K+ distinct values

* for numbers, date, string, variant, geography, binary etc (not float, geometry)

* filtering predicates: EQUALITY, SUBSTRING, GEO → keep a search access path

* for IN, reg exp, substrings, JSON searches, geospatial queries, and/or

* not for external/dynamic tables, MVs, casts…

Checkpoints

* Enterprise+, 10 credits/h + 5 credits/h cloud (2+1 in sec. dbs)
* avoid in high churn tables
* avoid when data changes a lot in the table
* enable only on specific columns, w/ the right predicate
* disable once you no longer have benefiting queries
* check actual query improvements
* pair the service with smaller warehouses to lower the cost

Reduce Materialized Views Maintenance Cost

Tip #68

Checkpoints

* MV cost = 10 credits/h + 5 credits/h cloud (2+1 in sec. dbs)

* there are no tools to estimate costs of maintaining MVs (serverless)

* *what may trigger MV maintenance*
  - any change of micro-partitions in the base table
  - any changes w/ DML statements on the base table
  - reclustering of the base table

* cost proportional to: number of MVs + clustered MVs

* suspend/resume MVs → only defers costs, not reducing them

* *best practice*: start by creating only a few MVs + monitor the costs over time

Reduce Database Replication Cost

Tip #69

Replication in Snowflake

* Database Replication = ~db backup, for all editions
    ◦ w/ replication groups, across multiple accounts
* Share Replication = for all editions
* Account Replication = Business Critical only, same org
    ◦ w/ failover/failback ← w/ failover groups
* Cross-Cloud Auto-Fulfillment = shared listing → other regions

Checkpoints

* *db replication compute costs* = 2 credits/h managed + 1 credit/h cloud

* standard costs in both source+target accounts!

* initial+refresh replication → lower refresh frequency

* avoid transfer between regions → data transfer costs

* avoid very large databases, or w/ lots of data changes

* refresh granular objects → use a replication group

* DATABASE_REPLICATION_USAGE_HISTORY view → CREDITS_USED

Estimate Cost

of Hybrid Tables

Tip #70

Hybrid Tables

* Snowflake Unistore = OLTP (transactional) + OLAP (analytical)

* fast writes as well → stores data also by row

* required primary keys

* optional foreign keys → referential integrity enforced

* allows indexes

* row locking

Checkpoints

* Standard+, PuPr in some AWS paid accounts (currently)
* *consumption modes*
  - **compute** = for WHs, same credits as for standard tables
  - **storage** = fix monthly rate per GB, more expensive
  - **requests** = for serverless resources on the row storage clusters
* *cost per request*
  - 1 credit/h managed + 1 credit/h cloud
  - 1 credit per 30 GB read + 1 credit per 7.5 GB write

Introduction to Data Storage

Tip #71: Use On-Demand When You Don't Know Your Spending Pattern

Tip #72: Copy and Keep Less Data

Tip #73: Lower Data Retention with No Time Travel

Tip #74: Estimate Storage Cost of the Fail-Safe

Tip #75: Use Transient or Temporary Tables

Tip #76: Use Zero-Copy Cloning

Tip #77: Clone Less Data

Tip #78: Ensure Tables Are Clustered Correctly

Tip #79: Drop Unused Tables and Other Objects

Tip #80: Remove Old Files from Stage Areas

Use On-Demand Storage When You Don’t Know Your Spending Pattern

Tip #71

Data Storage Costs

OPTIMIZED STORAGE

Snowflake charges a monthly fee for data stored in the platform. Calculated using the average amount of storage used per month, after compression, for data ingested into Snowflake.

Download Snowflake Pricing Guide

On-Demand Storage

Pay for usage month-to-month.

$23 / per TB / per month ($USD)

AWS, US West (Oregon)

Capacity Storage

Discounted storage paid upfront.

$23 / per TB / per month ($USD)

AWS, US West (Oregon)

Capacity Storage Pricing

Table 3(a): Standard Storage Pricing

Cloud Provider	Region	On Demand Storage Pricing (TB/mo)	Capacity Storage Pricing by ACV Range (TB/mo)						
			Tier 1	Tier 2	Tier 3	Tier 4	Tier 5	Tier 6	Tier 7
			USD $0 - $1,199,999	USD $1,200,000 - $2,999,999	USD $3,000,000 - $4,999,999	USD $5,000,000 - $9,999,999	USD $10,000,000 - $19,999,999	USD $20,000,000 - $39,999,999	USD $40,000,000+
AWS	US East (Northern Virginia)	$23.00	$23.00	$21.47	$19.94	$18.40	$16.86	$15.34	$13.80
AWS	US West (Oregon)	$23.00	$23.00	$21.47	$19.94	$18.40	$16.86	$15.34	$13.80

Checkpoints

* On-Demand (PAYG) vs Capacity (prepaid) Storage Pricing
  ◦ US $23/TB/mo → lowest for AWS/Azure US ($20 for GCP Central US)
  ◦ storage billing → average stored per day
* Capacity Storage → less only for $2M+ accounts
  ◦ storage price → per TB/mo (from highest for On-Demand)
  ◦ long-term contract for small discounts → could overspend
  ◦ auto-switched to On-Demand if underspent
  ◦ use eventually once you know your spending patterns

Copy and Keep Less Data

Tip #72

Cost

* table data storage: 2B rows (65GB)

* Time Travel: 1 day (def)

* Fail-safe: 7 days (fixed)

* one clone: 2B rows (shared)

copied from tpcds_sf100tcl.store_sales (288B rows, 10TB) in 11 mins

TABLE_STORAGE_METRICS

* in INFORMATION_SCHEMA → w/ 1-2h latency

* Active bytes = live data that can be queried

* Bytes in Time Travel = deleted, but still kept for the retention period

* Bytes in Fail-safe = deleted, past the Time Travel, but still kept for 7 days

* Bytes retained for clones = deleted, but still kept as clone refs

Retention Policies

* do not use Snowflake as an archiving system
  - backup/store raw data cheap in Amazon S3 Glacier
* ex: retention policy: 5 → 3 years
  - less bytes scanned in queries
  - 36% storage cost saving

Checkpoints

* copy less, clone more → for dev/test/prod envs
* use transient/temporary tables for no Fail-safe
* disable Time Travel
* cloned tables will transparently keep shared table data
* create huge table + delete right away → still expensive!
* limit period of retention policy

Lower Data Retention with No Time Travel

Tip #73

Checkpoints

* *permanent table*
  - 0..1 days (def 1) Time Travel for Standard Edition
  - 0..90 days (def 1) Time Travel for Enterprise Edition
* *temporary/transient table*
  - 0..1 days (def 1) Time Travel

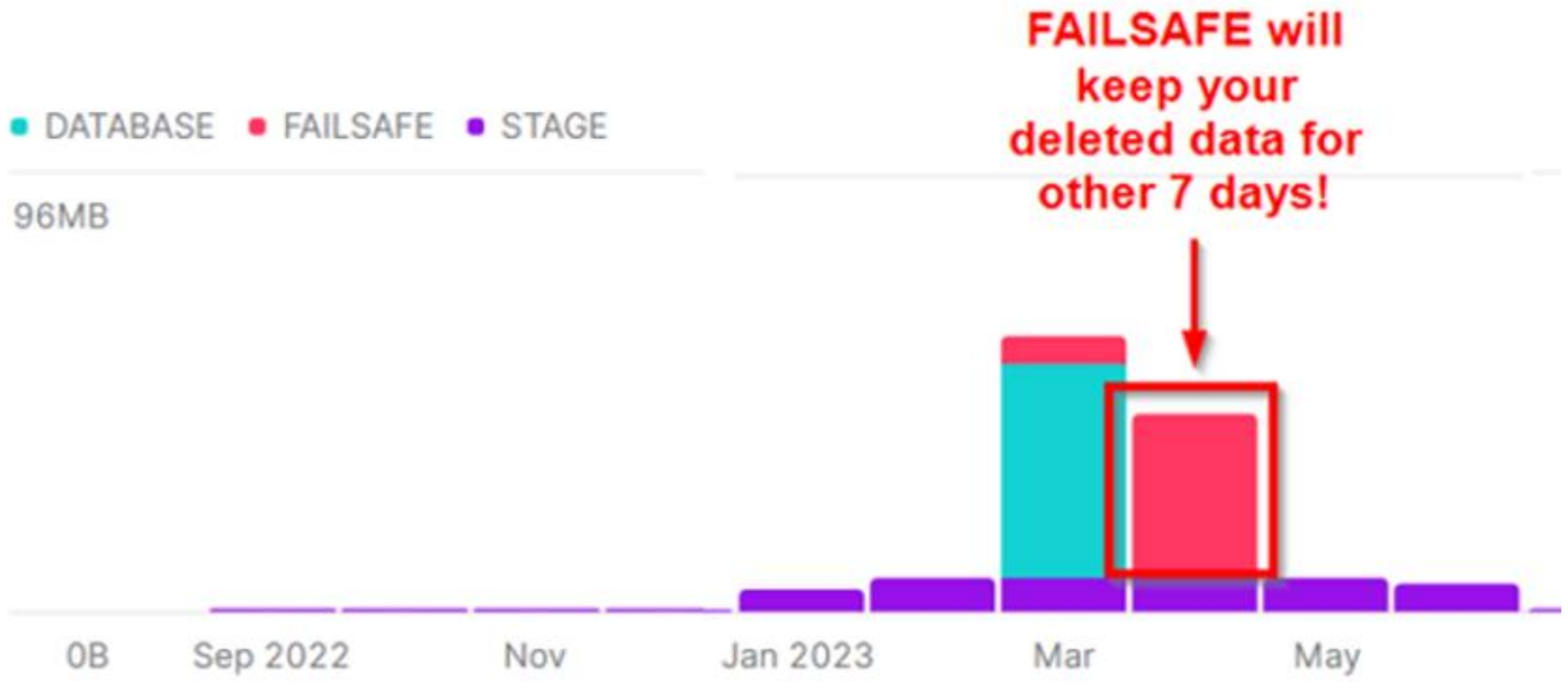
Time Travel may remain there after you drop a table

set DATA_RETENTION_TIME_IN_DAYS = 0 to disable

Estimate Storage Cost of the Fail-Safe

Tip #74

Fail-Safe for Large Dropped Table



Cost

* 1TB table dropped after 1 day → $23 / 30 days ~= 77c

* Fail safe for dropped table → $23 x 7/30 days = $5.37

cost On-Demand storage: US **$23**/TB/mo

Checkpoints

* permanent table : 7 days Fail-safe

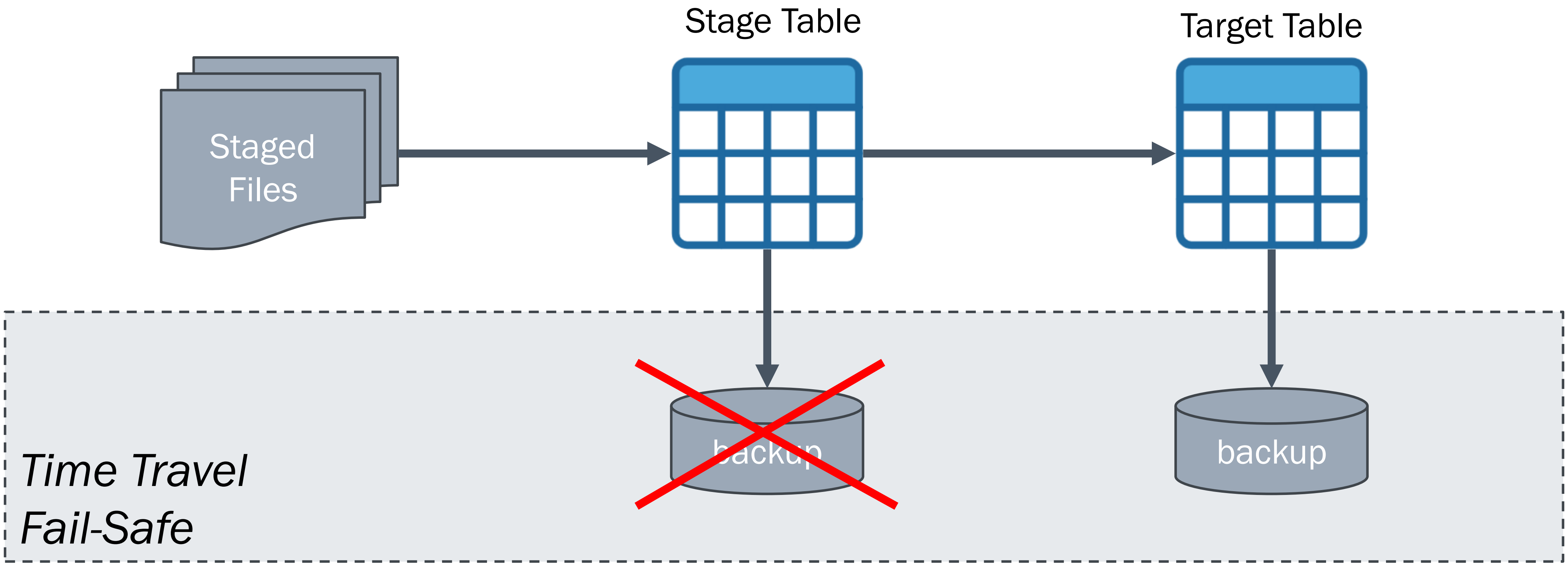
* temporary/transient table: no Fail safe

* drop a large table → *all* its data moves into Fail safe!

Use Transient or Temporary Tables

Tip #75

ELT Data Pipeline



Checkpoints

* temporary table = auto-deleted at the end of session

* transient table = shared, req explicit drop

* max time travel 1 day (def 1) → save on storage

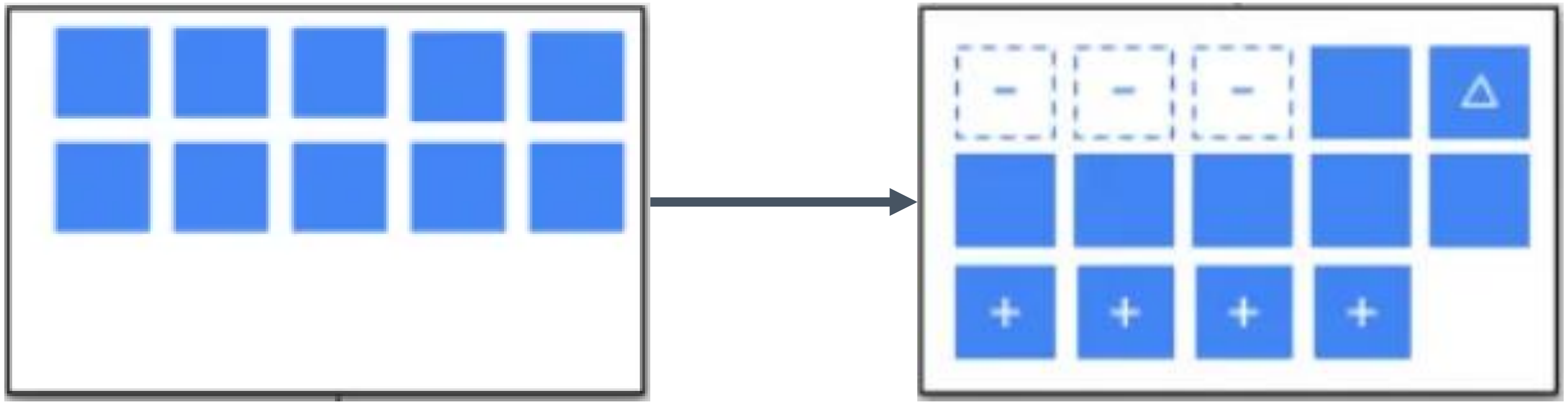
* no fail safe (else 7 days) → save on storage

* for transitory data (no need for backup)

Use Zero-Copy Cloning

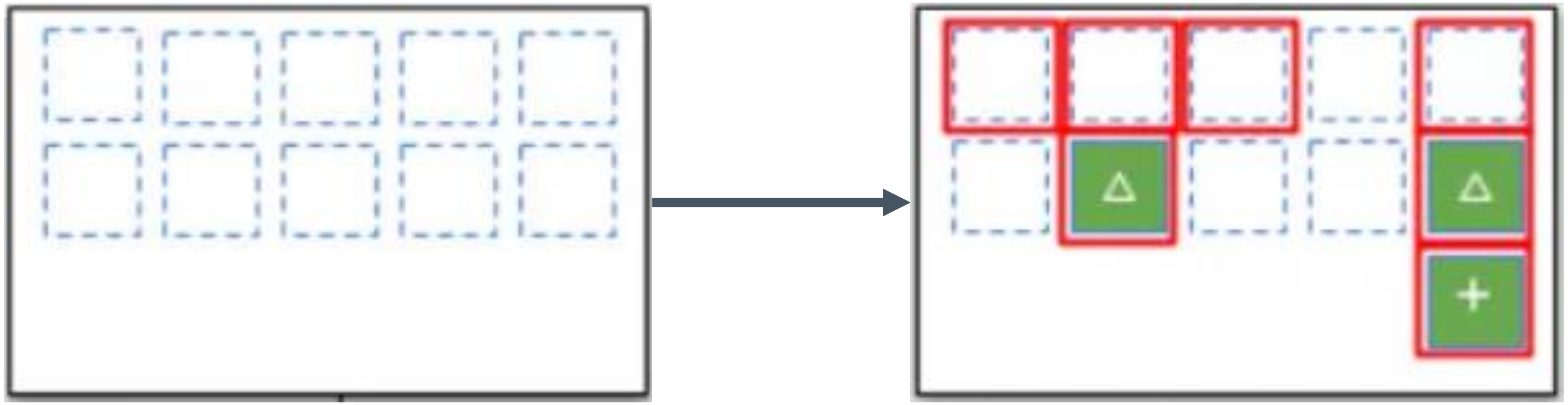
Tip #76

Table Copy



Storage Cost for Copied Table = Original Data + New Data - Deleted Data

Table Clone



Storage Cost for Cloned Table = Changed Data + New Data

Checkpoints

* copied table cost = orig data + INSERTs - DELETEs
* cloned table cost = UPDATEs + INSERTs

* clones will save on storage (keep data references)
* clones will save on compute (to duplicate data)
* clones will avoid redundancies → less error prone

Clone

Less Data

Tip #77

Checkpoints

* extensive DDL (cloning) → significant cloud serv usage
* DDL opers (like cloning) use only cloud compute serv

* clone more granular objects
* clone only individual tables rather than entire schemas

Ensure Tables Are Clustered Correctly

Tip #78

CLUSTERING_INFORMATION

* *cluster_by_keys*, *total_partition_count* - just info

* *total_constant_partition_count* - higher is better (reclustered)

* *average_overlaps* - lower is better (<1), higher not well-clustered

* average_depth - lower is better (CLUSTERING_DEPTH)

* *partition_depth_histogram*
  - 0, 1, 2, 3, ..., 16, 32, 64, 128, ...
  - most w/ depth 1 or on top is best
  - worst if most at the bottom

* *clustering_errors* - max 10 recent reclustering errors [...]

Checkpoints

* load data → natural auto-clustering

* evaluate for natural auto-clustering (no key → no cost!)

* evaluate for a clustering key (auto-clustering cost!)

* identify + evaluate for candidate columns

* load data according to current/new clustering key

* use queries that benefit from current table clustering

Drop Unused Tables and Other Objects

Tip #79

Storage Usage History

* DATABASE_STORAGE_USAGE_HISTORY
  - *average_database_bytes*
  - *average_failsafe_bytes*
  - *average_hybrid_table_storage_bytes*
* STAGE_STORAGE_USAGE_HISTORY
  - *average_stage_bytes*

Table Storage Metrics

* *active_bytes*

* *time_travel_bytes*

* *failsafe_bytes*

* *retained_for_clone_bytes*

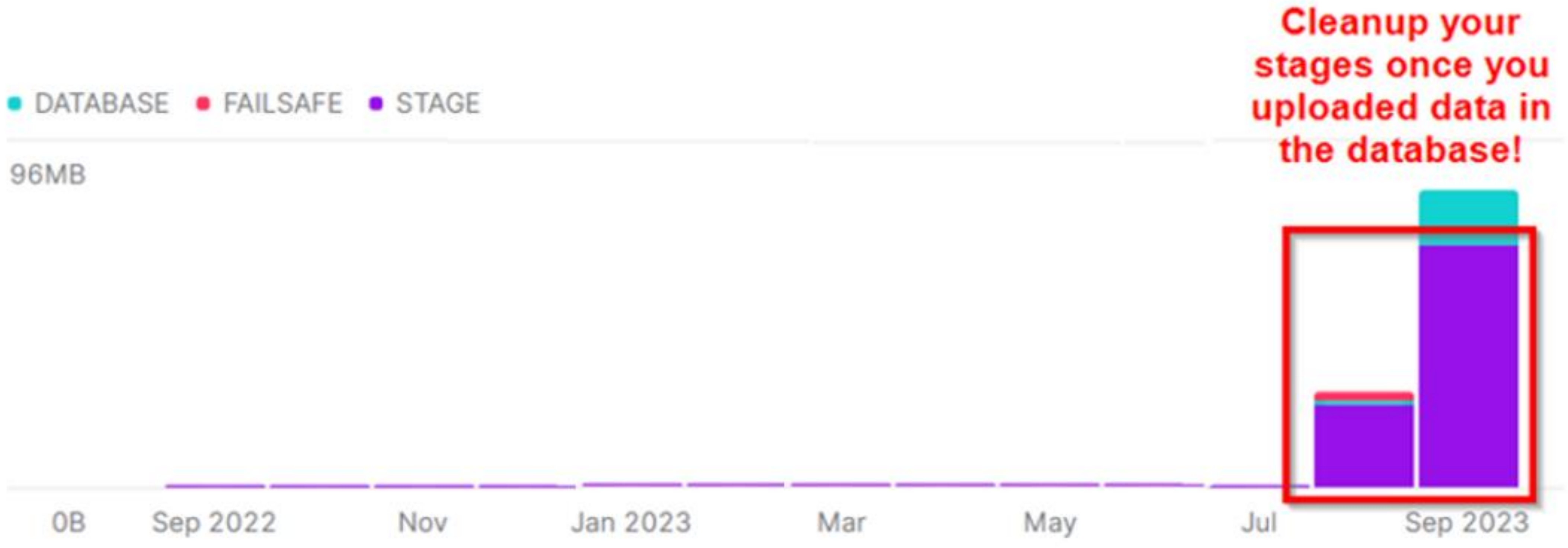
Checkpoints

* *storage*
  * db active + time travel + fail-safe + retained for clones + hybrid tables + stages
* drop any unused objects → recreate them from scripts!
* *unused tables* = not modified/changed/accessed lately
* TABLES view
  * *last_altered* → by DDL/DML operation
* ACCOUNT_HISTORY view
  * *base_objects_accessed* → JSON array, must flatten!
  * *query_start_time* → last accessed time

Remove Old Files from Stage Areas

Tip #80

Storage Wasted by Old Stage Files



Checkpoints

* delete files no longer required from stages
* external+internal stages (named/table/user)
* ACCOUNT_USAGE.STORAGE_USAGE.stage_bytes

SHOW STAGES IN ACCOUNT → all stage names

LIST @mystage → all files

REMOVE @mystage / @%mytable / @~

Introduction to Data Transfer

Tip #81: Data In is Free, Data Out is Expensive

Tip #82: Choose the Same Provider and Region Where Your Data Is

Tip #83: External Access Integrations vs External Functions

Tip #84: Use Data Compression

Tip #85: Use Batch Transfer with Path Partitioning

Tip #86: Use Bulk Loads instead of Single-Row Inserts

Tip #87: Use Parallel Data Uploading

Tip #88: Design Cost-Effective Data Pipelines

Tip #89: Use External Tables in a Data Lake

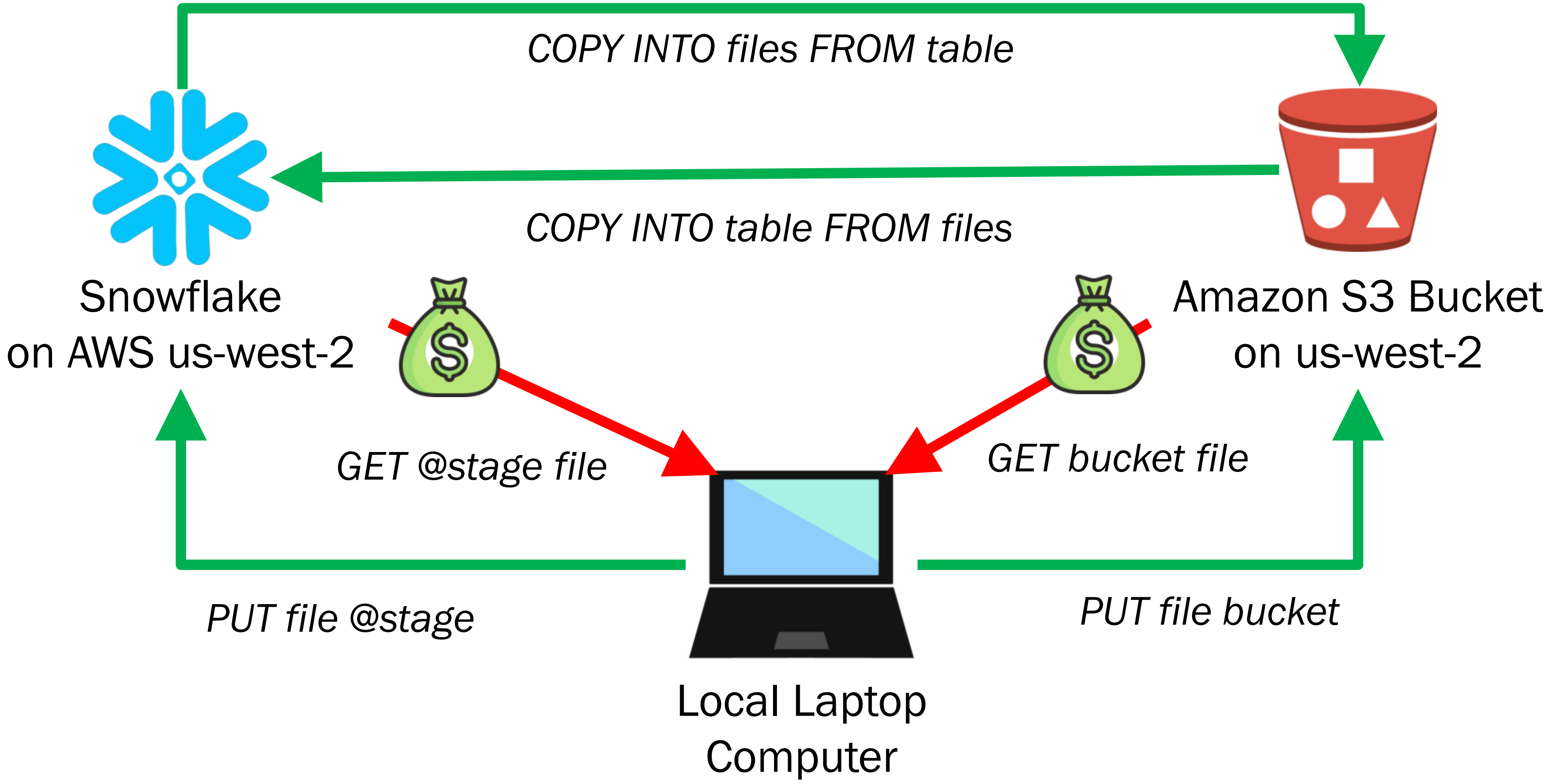
Tip #90: Query Parquet Files instead of CSV

Data In is Free,

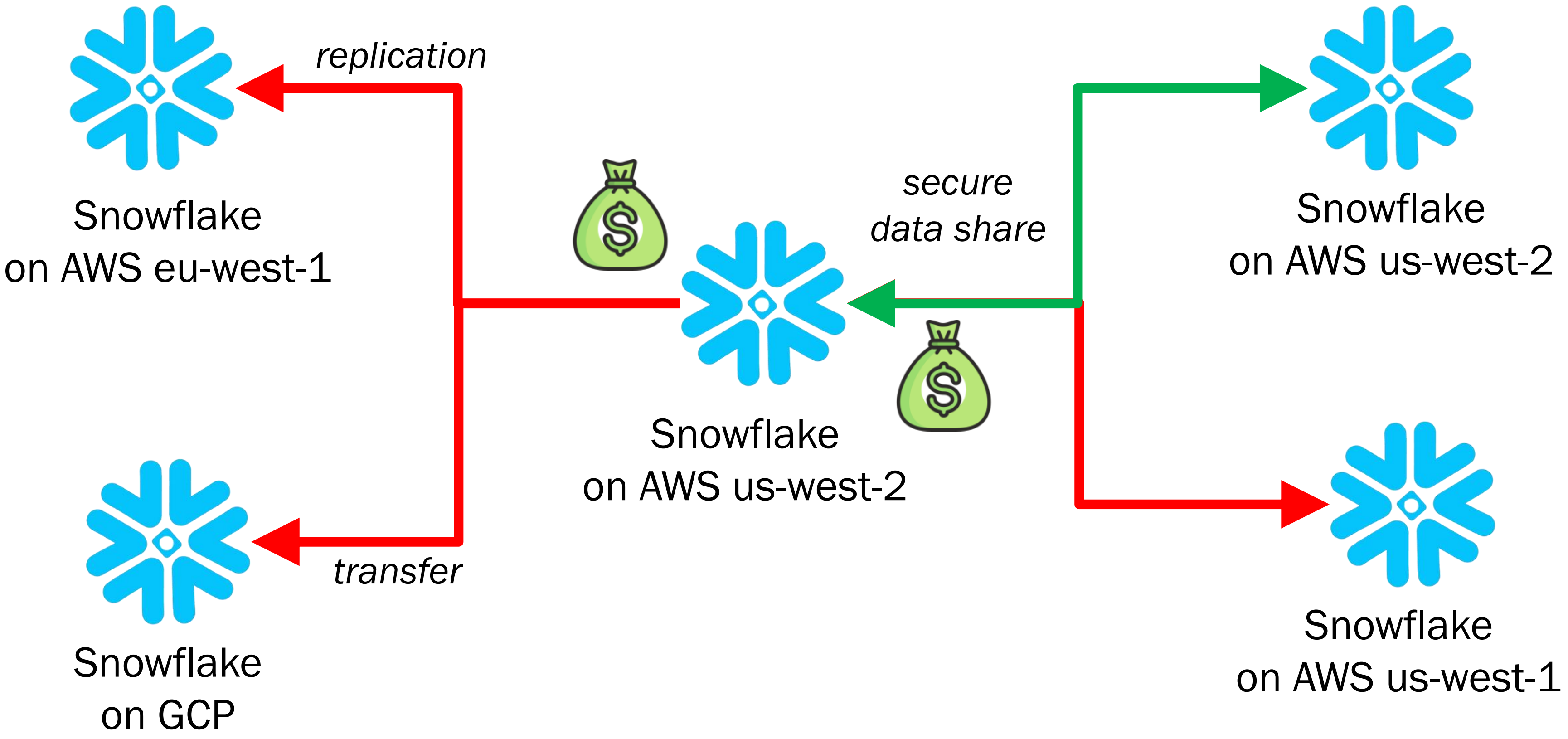
Data Out is Expensive

Tip #81

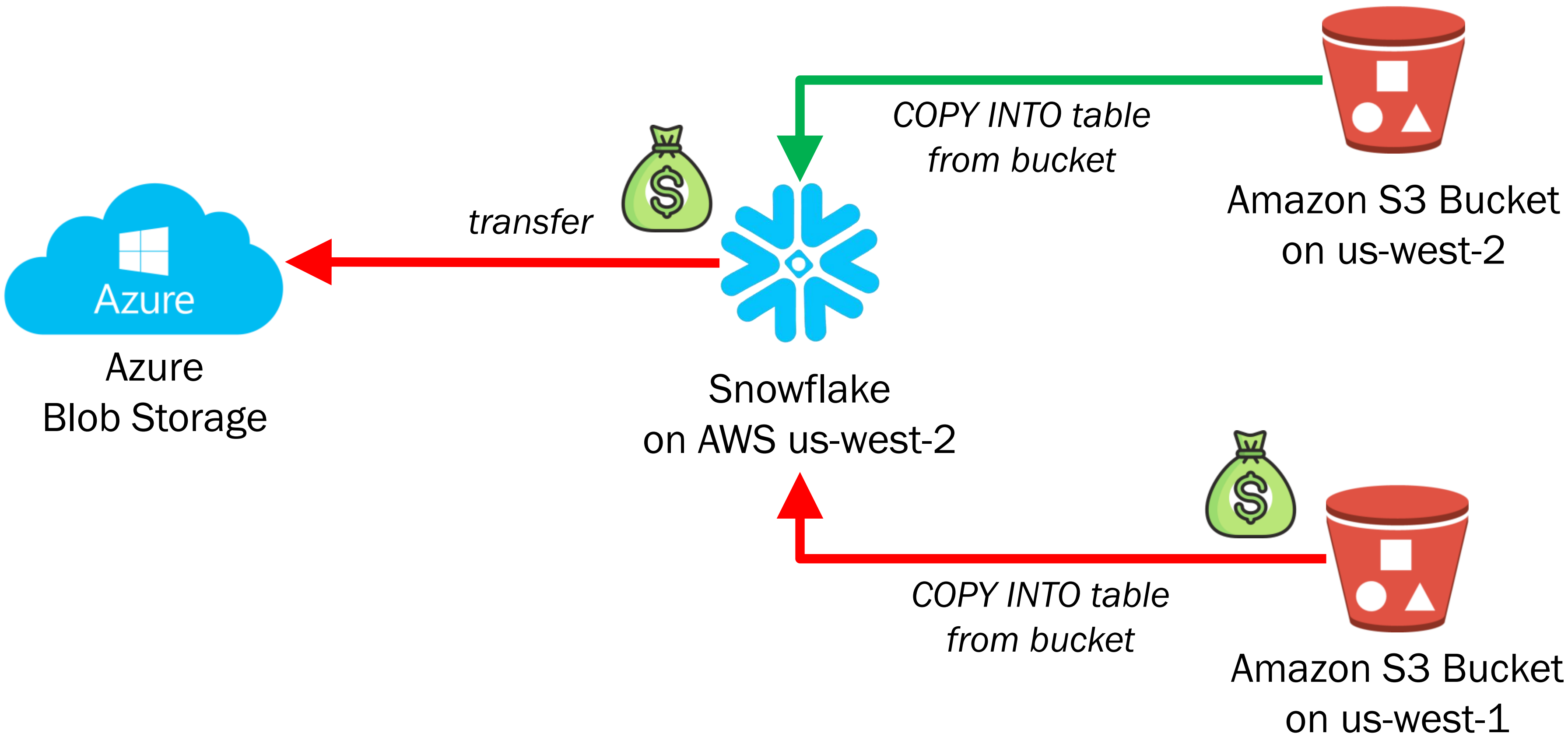
Local Data Transfer Costs



Snowflake-to-Snowflake Costs



Internet Data Transfer Costs



DATA_TRANSFER_HISTORY

	TFX	TRANSFER_TYPE	BYTES_TRANSFERRED
1	aws (us-west-2) → (aws) us-west-2	EXTERNAL_FUNCTION	2808091
2	aws (us-west-2) → (aws) us-east-2	EXTERNAL_FUNCTION	27797
3	aws (us-west-2) → (internet) internet	EXTERNAL_ACCESS	98058
4	aws (us-west-2) → (aws) us-west-2	COPY	101584

Checkpoints

* Snowflake (us-west-1) → S3 bucket (us-west-2) - $ to Snowflake

* Snowflake (us-west-1) ← S3 bucket (us-west-2) - $ to AWS

* Snowflake → local - $ to Snowflake

* AWS → local - $ to AWS

Choose the Same Provider and Region Where Your Data Is

Tip #82

Data Transfer Pricing

* AWS: diff region $20+ / diff prov $90+

* Azure: same cont $20+ / diff cont $50+ / diff prov $87.50+

* GCP: same cont $10+ / diff cont $80+ / diff prov $120+

pricing is per TB of data transferred

within the same region+provider is FREE

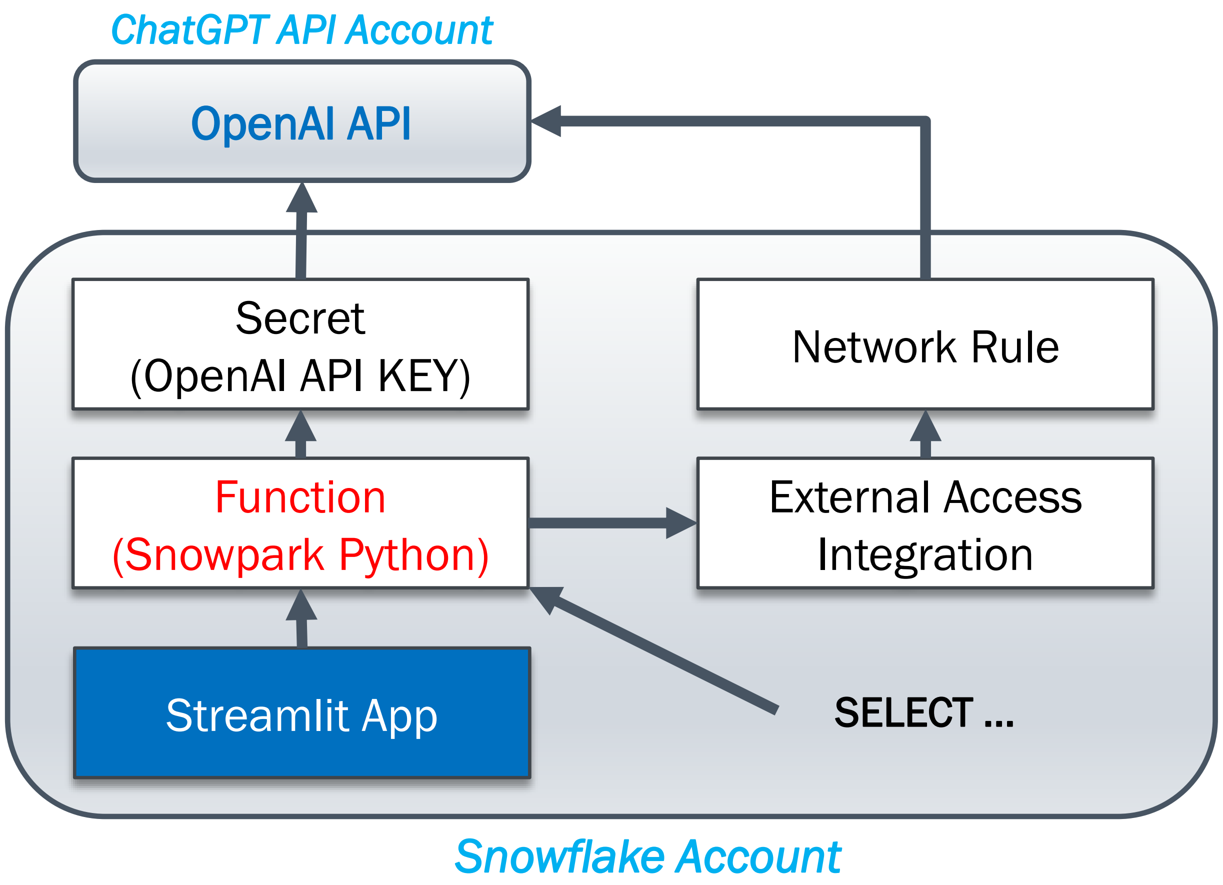
Checkpoints

* no cost to tfx data in (upload)
* no cost to tfx data out within the same region+provider
* moderate cost to tfx to another region, same provider
* moderate cost to tfx within the same continent+provider
* higher cost to tfx to another continent, same provider
* highest cost to tfx to another provider

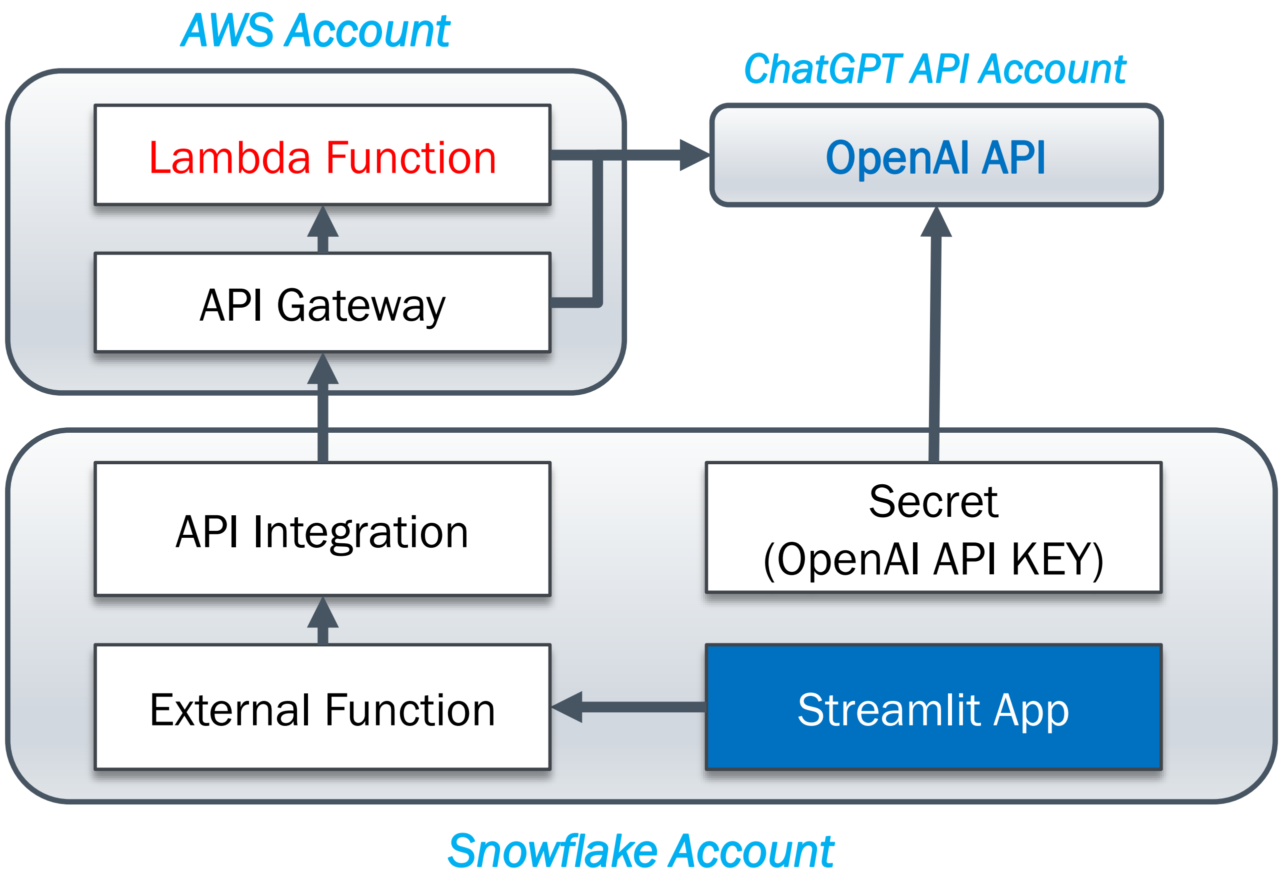
External Access Integrations vs External Functions

Tip #83

External Access Integration



External Function



Checkpoints

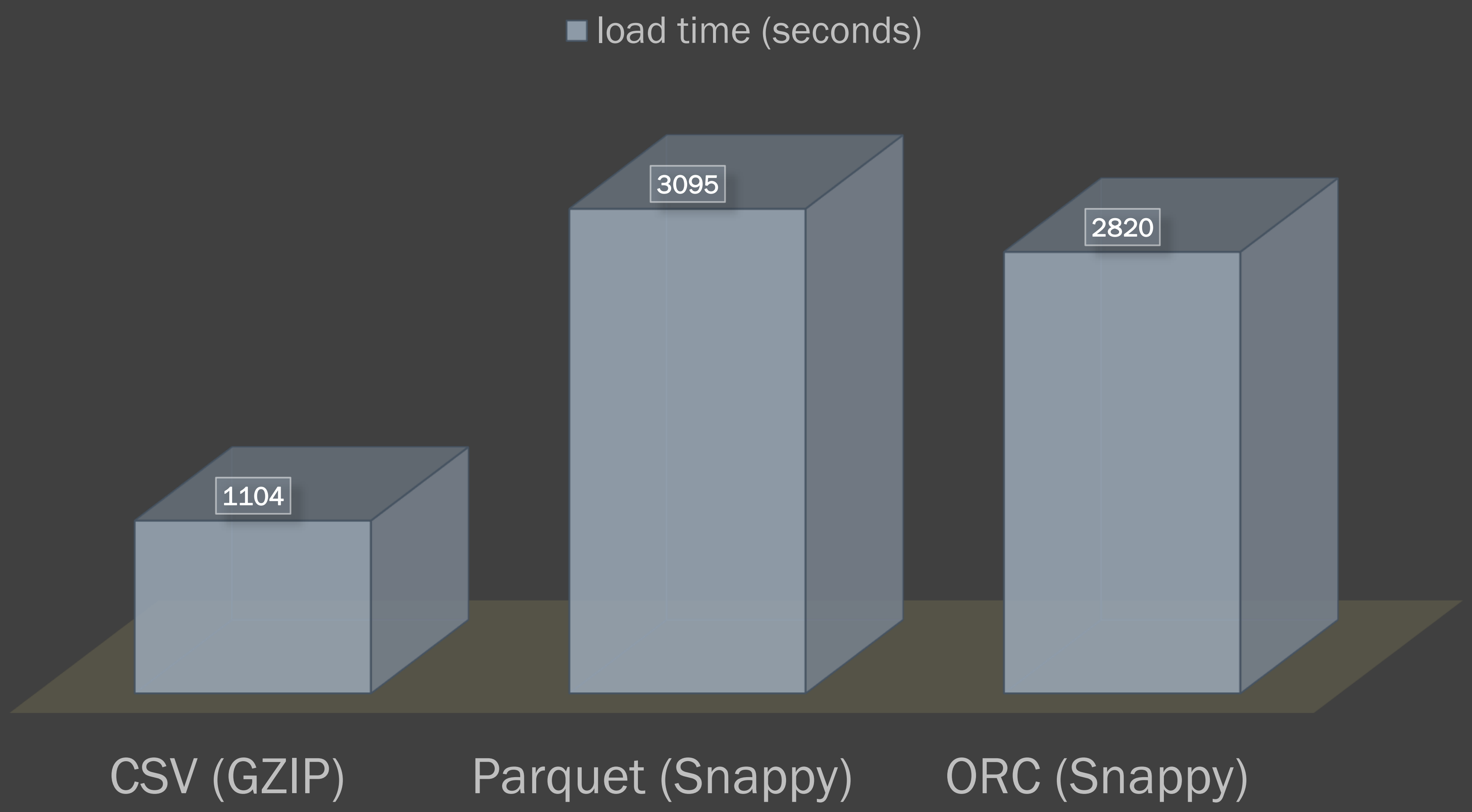
* *External Access Integration*
  - business logic mostly within Snowflake, in the WH
  - use when you do not transfer a lot of data w/ the remote API
  - easier to implement and deploy
* *External Function*
  - business logic mostly remote, in another cloud service
  - use when you may transfer a lot of data w/ the remote API
  - more complex to implement and deploy

Use

Data Compression

Tip #84

LOADING STRUCTURED DATA



Checkpoints

* Snowflake storage
    * min 3:1 compression ratio → billed only compressed storage!
    * 4.7 TB uncompressed (28.8B rows) → stored as 1.3 TB
* loading compressed data
    * less data transfer cost + less time (cheaper COPY INTO)
    * from CSV (GZIP) >> faster than Parquet/ORC (Snappy)
    * keep data files 100-250 MB in size compressed
    * COMPRESSION = GZIP/BZ2/BROTLI/ZSTD/DEFLATE/RAW_DEFLATE

Use Batch Transfer with Path Partitioning

Tip #85

Checkpoints

* partition staged data files w/ logical granular paths
* COPY INTO must list S3 folder files → improve performance!
* ex: s3://bucket/app2/loc1/2016/07/01/14/
* copy any fraction of data w/ a single command
* could copy data by hour/day/month etc
* create subfolders of 10-15 mins increments per hour

Use Bulk Loads instead of Single-Row Inserts

Tip #86

Checkpoints

* COPY INTO (batch/bulk) vs INSERT (single row)

* OLTP vs OLAP

* batch/bulk copy consumes less cloud res + faster

* batch/bulk loads can be done in parallel

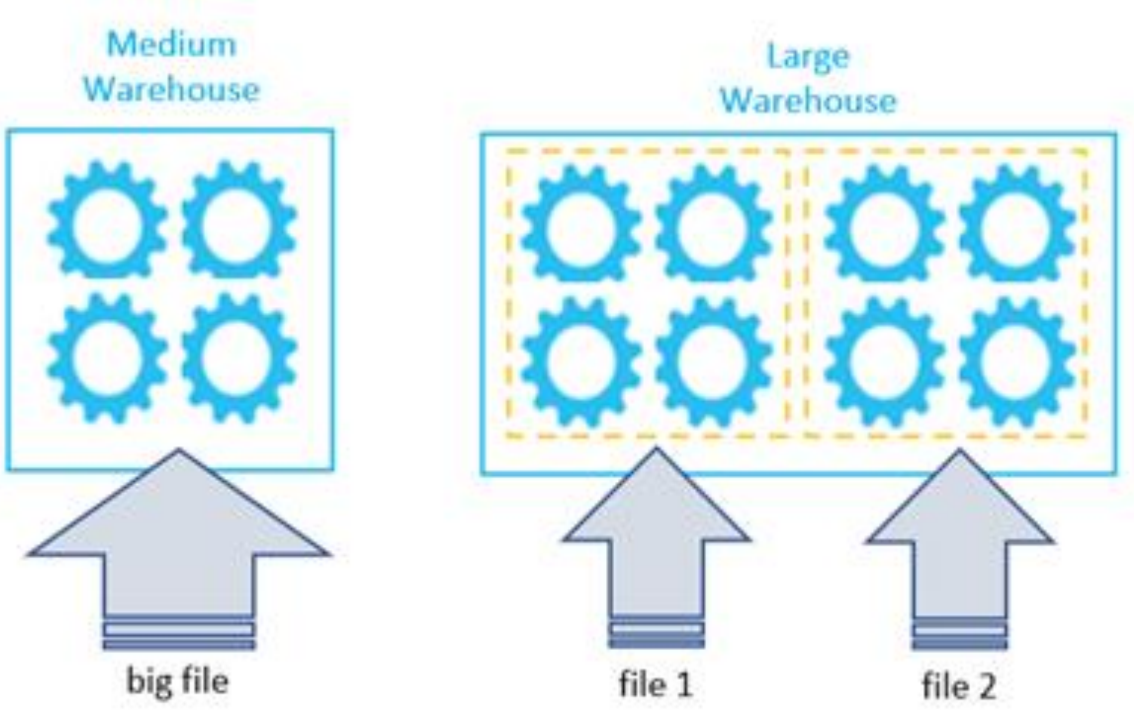
* not for hybrid tables (Snowflake Unistore)

Use Parallel

Data Uploading

Tip #87

Parallel Data Uploading



Checkpoints

* small WH + one file → ok in most cases

* big WH + one file → may take longer if large file

* big WH + single file → could waste idle nodes

* big WH + parallel loading → faster for ~100 files

```
COPY INTO mytable FROM @stage/folder/ FILES=('file1.csv', ...)
```

Design Cost-Effective Data Pipelines

Tip #88

Checkpoints

* check that data-loading processes don’t run more often than consuming data pipelines

* reduce sync frequency of your data ingestion tool + evaluate how often to update data

* upload data to S3 + use as external stage for data loading to Snowflake

* check related costs when running all type of workloads in Snowflake

* it could be more cost-effective to transform S3 data w/ AWS Glue, then load it in Snowflake

* incremental processing: do not rerun entire batches of data for minor updates

* restart interrupted pipelines from the point of interruption, w/ no redundant computation

Use

External Tables

in a Data Lake

Tip #89

Checkpoints

* to query in-place data lake files, from ext stages, R/O
* slower than Snowflake tables → use mat view
* metadata$filename + metadata$file_row_number
* value → single VARIANT column, use for column defs (or INFER_SCHEMA)
* partitioning always recommended → PARTITION BY col1, ...
* manual/auto-refresh metadata → w/ event notifs, may incur cost
* PIPE_USAGE_HISTORY → estimate charge

Query Parquet Files instead of CSV

Tip #90

Parquet vs CSV

* 80%+ less data stored
* 30%+ faster queries
* 99% less data scanned
* 99%+ cost savings

Dataset	Size on Amazon S3	Query Run Time	Data Scanned	Cost
Data stored as CSV files	1 TB	236 seconds	1.15 TB	$5.75
Data stored in Apache Parquet Format	130 GB	6.78 seconds	2.51 GB	$0.01
Savings	87% less when using Parquet	34x faster	99% less data scanned	99.7% savings

Checkpoints

* transform CSV to Parquet when left in the data lake
* less storage because of binary format + compression
* less data scanned because of less storage + columnar
* faster queries because of the columnar format
* huge compute + storage cost savings

Introduction to Snowflake Apps

Tip #91: Estimate Cost Impact of Data Sharing in Snowflake

Tip #92: Estimate Cost Impact of Client and Server (Snowpark) Apps

Tip #93: Estimate Cost Impact of Streamlit in Snowflake and Native Apps

Tip #94: Estimate Cost Impact of Data Science Applications

Tip #95: Check All Connected Applications

Tip #96: Third-Party Apps Saving Money Will Spend Money

Tip #97: Free Marketplace Native Apps Will Cost Money

Tip #98: Keep App Versions Updated

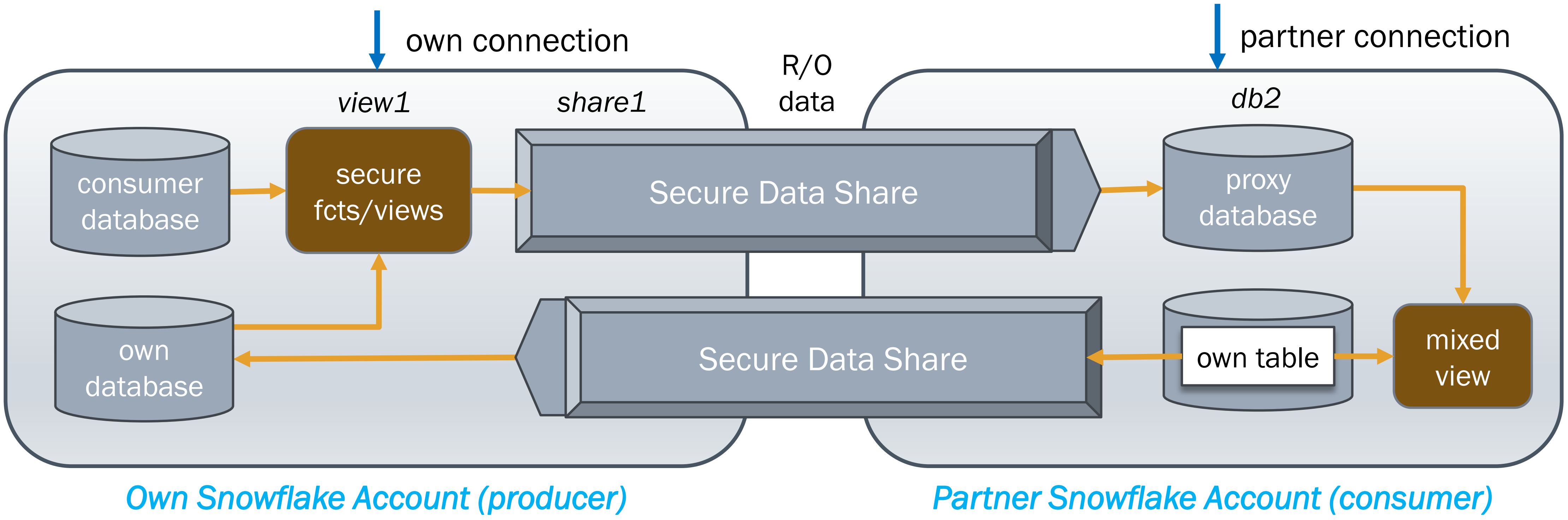
Tip #99: Cache Data in Third-Party Tools

Tip #100: Auto-Abort Running Queries from Disconnected Apps

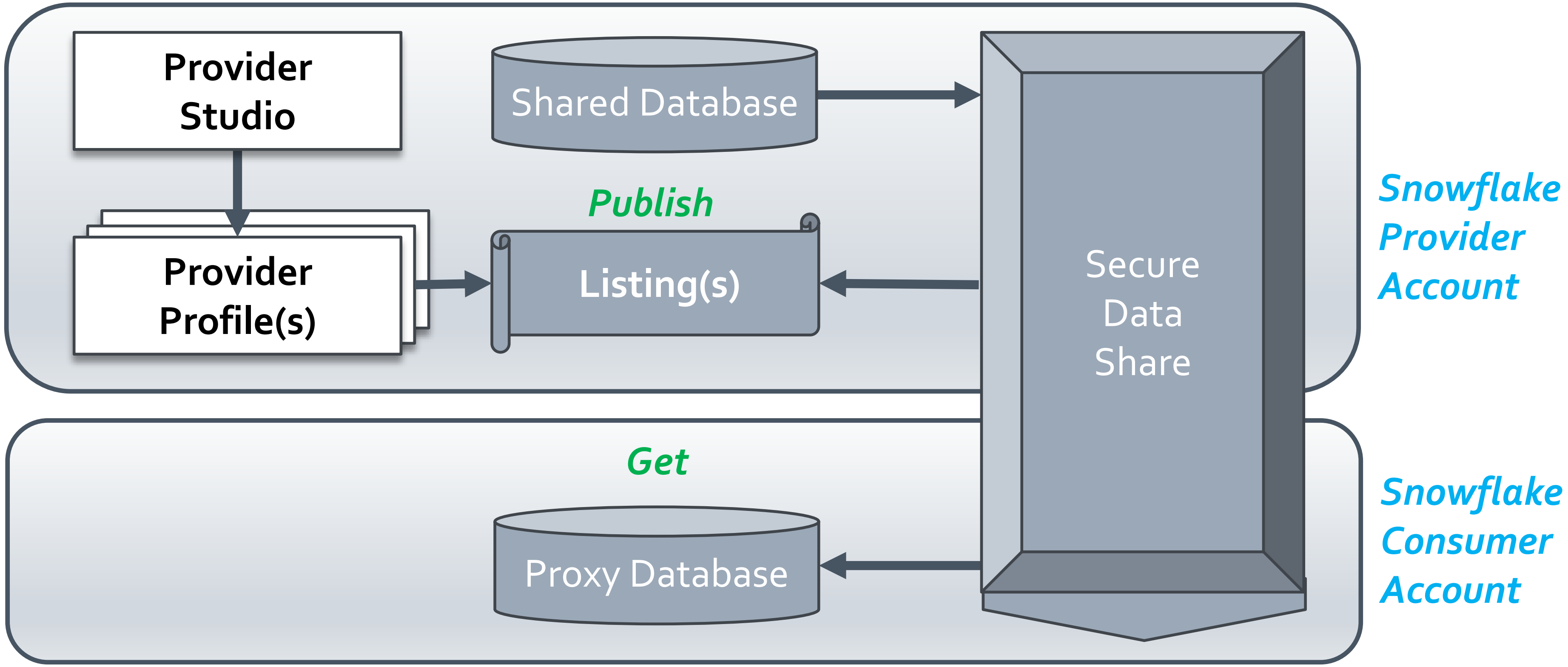
Estimate Cost Impact of Data Sharing in Snowflake

Tip #91

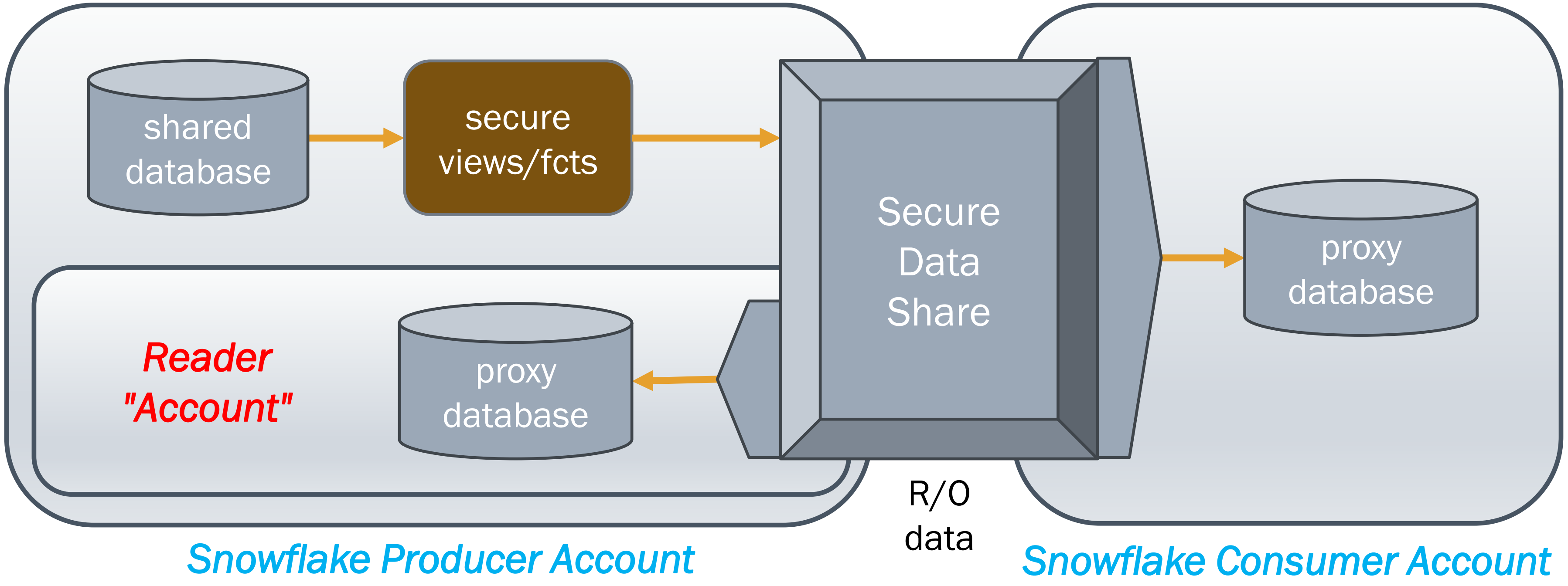
Inbound/Outbound Data Shares



Private Data Exchange



Data Share w/ Reader Account



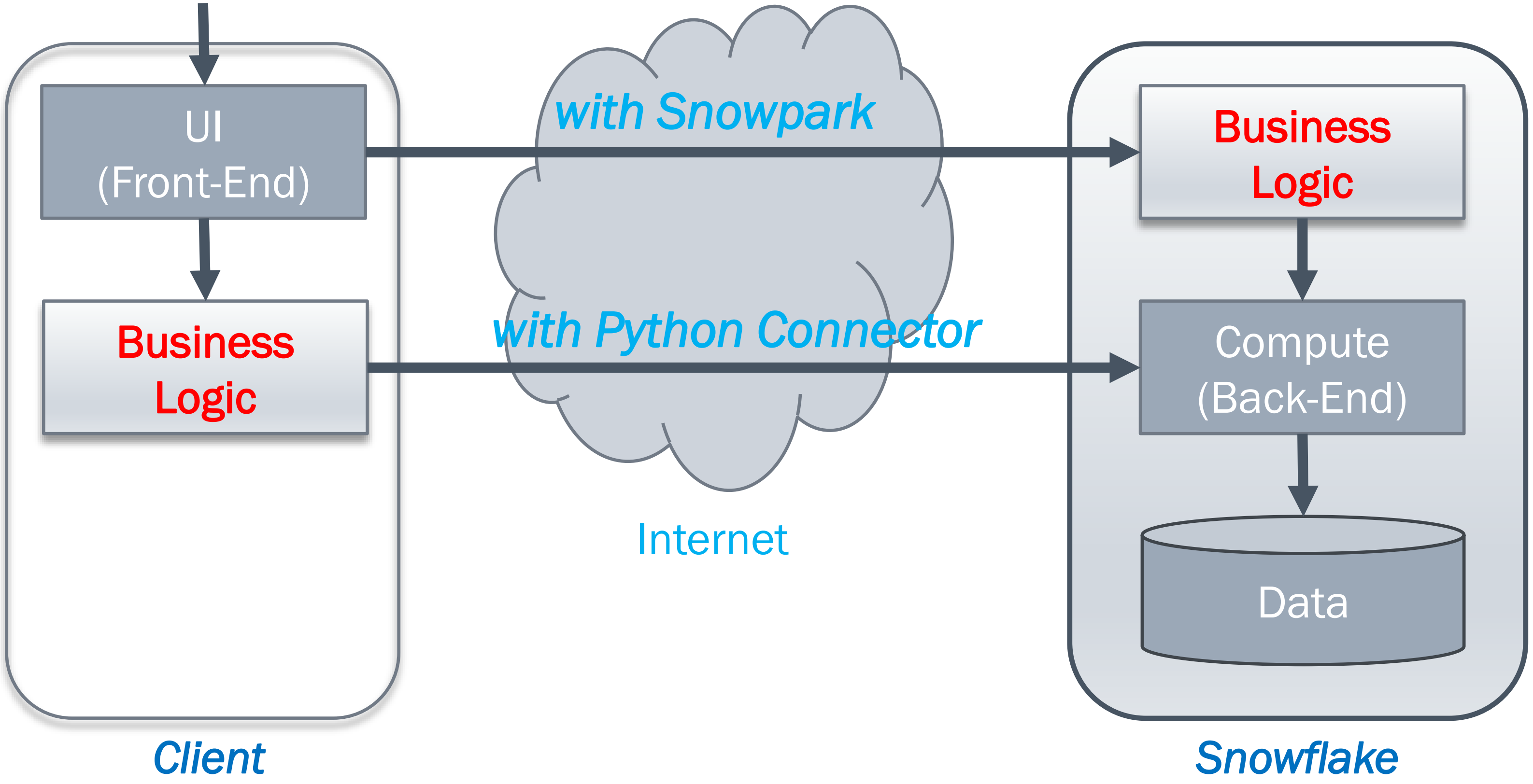
Checkpoints

* data shares
  - if provider → consume some other account compute credits
  - if provider for Reader account → consume OWN credits!
  - if consumer → consume OWN credits
* Marketplace (public) / Data Exchange (private)
  - consumer of shared datasets → consume OWN compute credits
  - ex: SNOWFLAKE_SAMPLE_DATA datasets → some are huge!

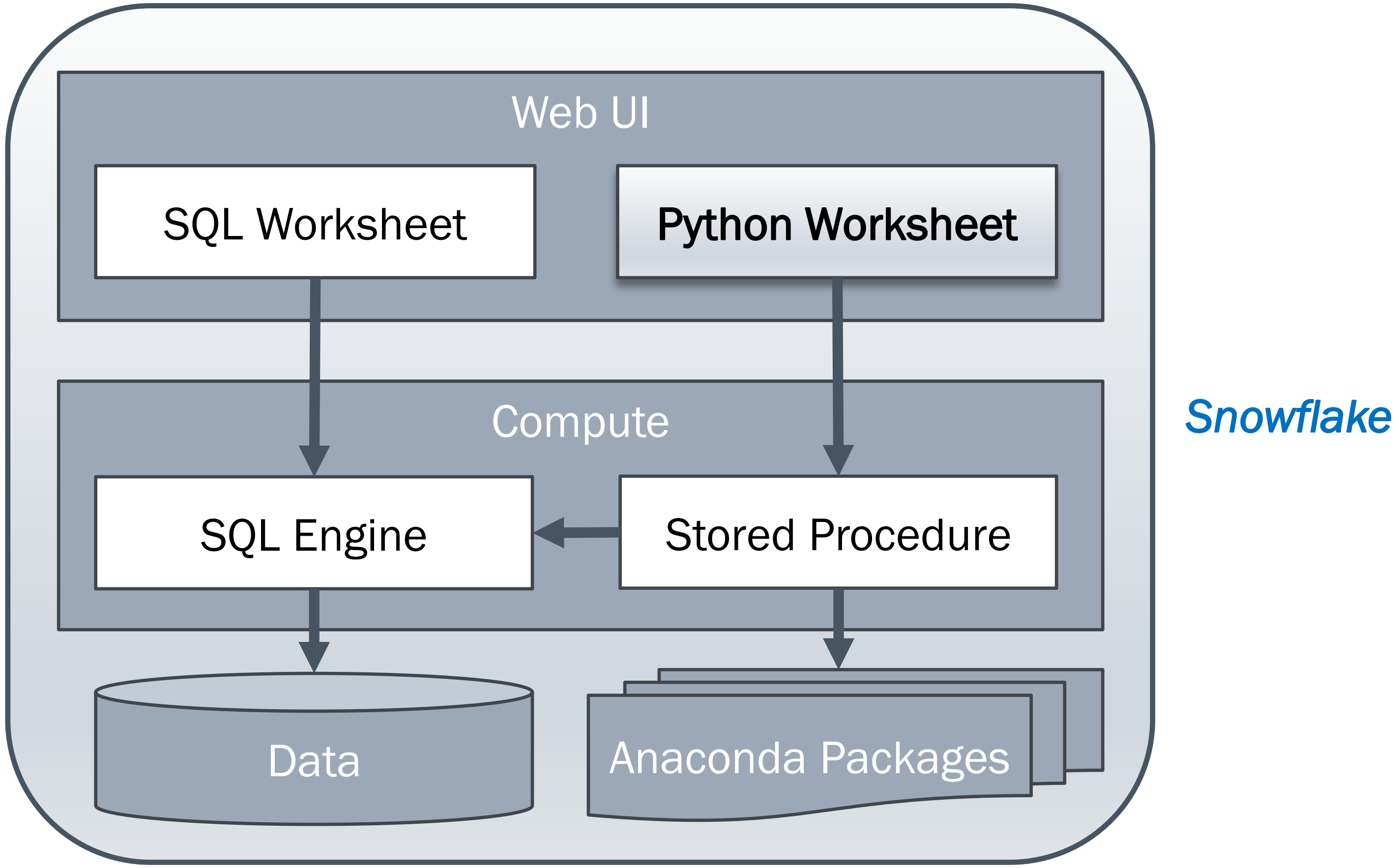
Estimate Cost Impact of Client and Server (Snowpark) Applications

Tip #92

Snowpark vs Python Connector



Python Worksheet



Client and Server Applications

* client apps (w/ client drivers)
  - consume YOUR compute + storage credits
  - plus lots of data transfer, eventually
* server apps (w/ Snowpark, SiS, Container Services)
  - consume **A LOT OF** YOUR compute + storage credits
  - but less data transfer (as they are closer to data)

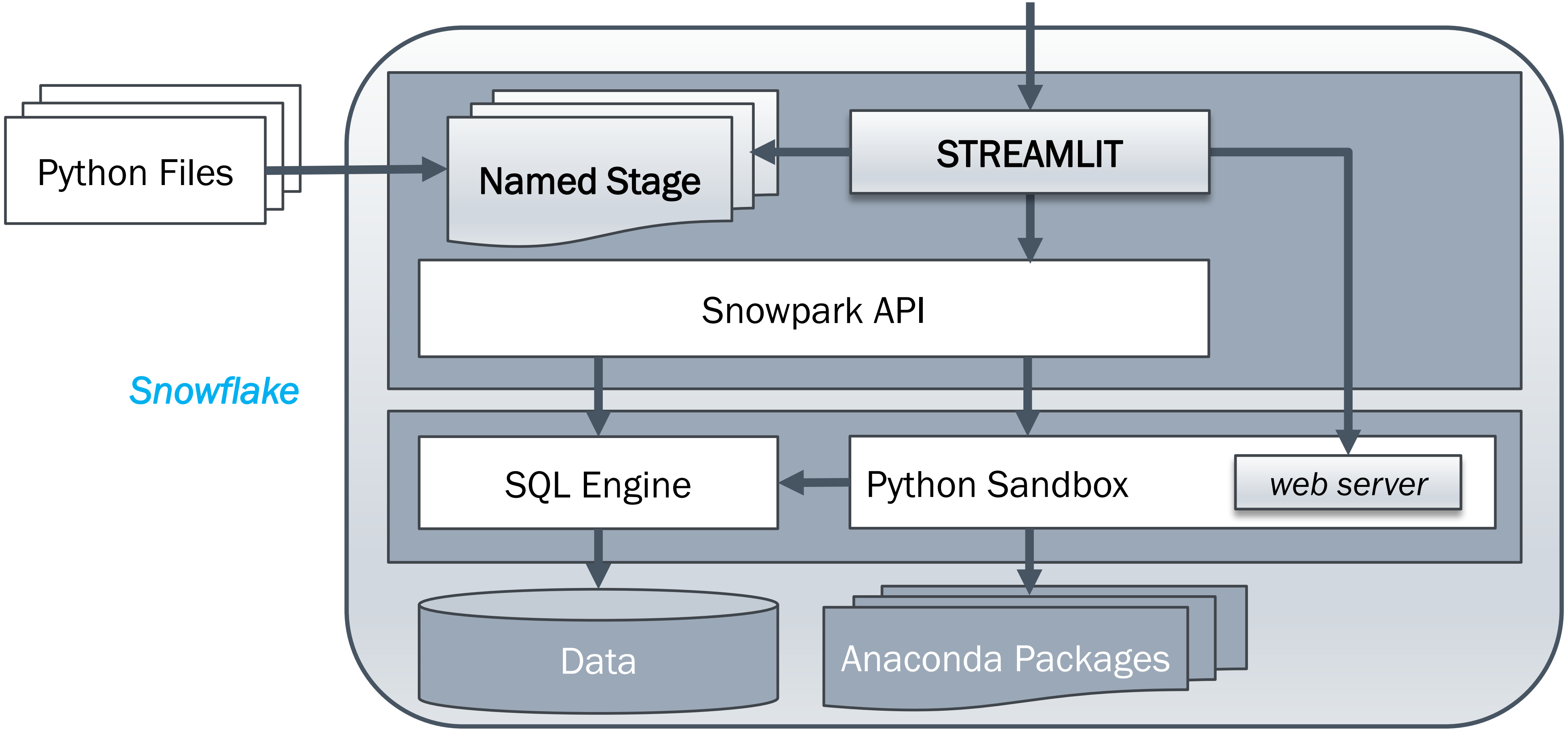
Checkpoints

* Snowpark apps = mostly server-side apps
* unlike client apps, business logic is deployed in WHs
* Snowflake WHs = db servers (SQL) → app servers (Python)
* better rich alternative to Snowflake Scripting
* stored proc wrappers around Python code → in WHs
* efficient if executing code closer to data is justified

Estimate Cost Impact of Streamlit in Snowflake and Native Applications

Tip #93

Streamlit in Snowflake (SiS)



Snowflake Native Apps

* if provider → consume some other account credits

* if consumer → consume OWN compute credits

* *warning*: any FREE native app will cost you money!

* ex: SNOWFLAKE app, w/ ACCOUNT_USAGE schema

* it is your metadata, but expensive to browse!

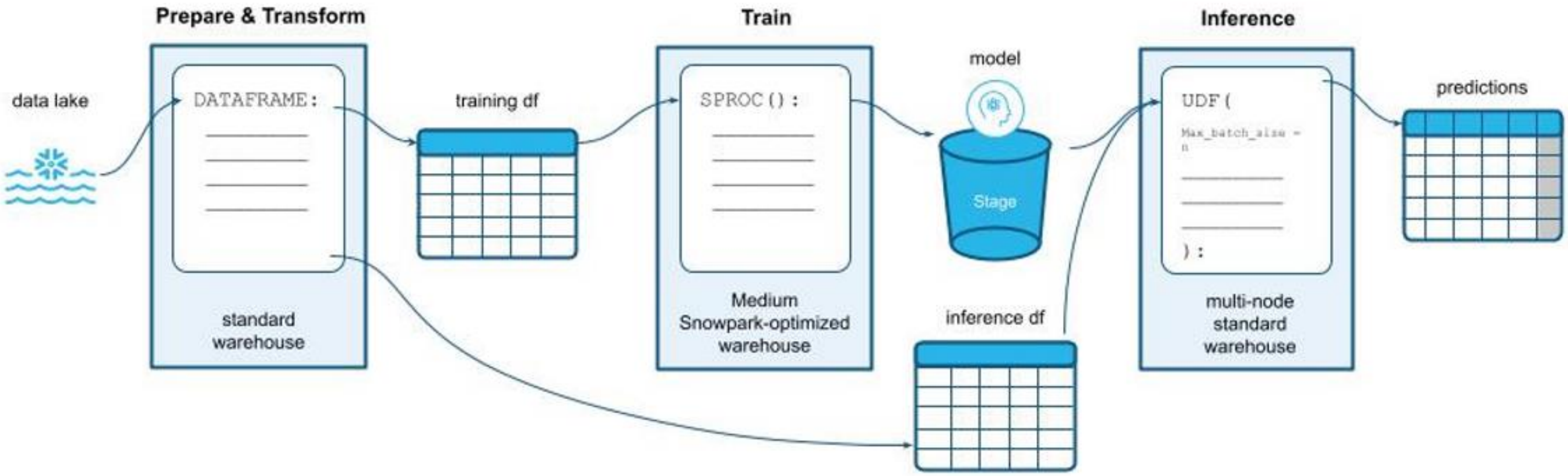
Checkpoints

* Streamlit in Snowflake (SiS) = wrapper around Python code
* STREAMLIT object, deployed+executed as stored proc in WH
* it keeps WH up as long as app used → expensive!
* app goes to sleep when WH auto-suspended
* most Streamlit web apps can be easily deployed as SiS
* alternatives: native apps, Container Services

Estimate Cost Impact of Data Science Applications

Tip #94

Model Training & Serving



https://www.snowflake.com/blog/snowpark-python-feature-engineering-machine-learning/

Checkpoints

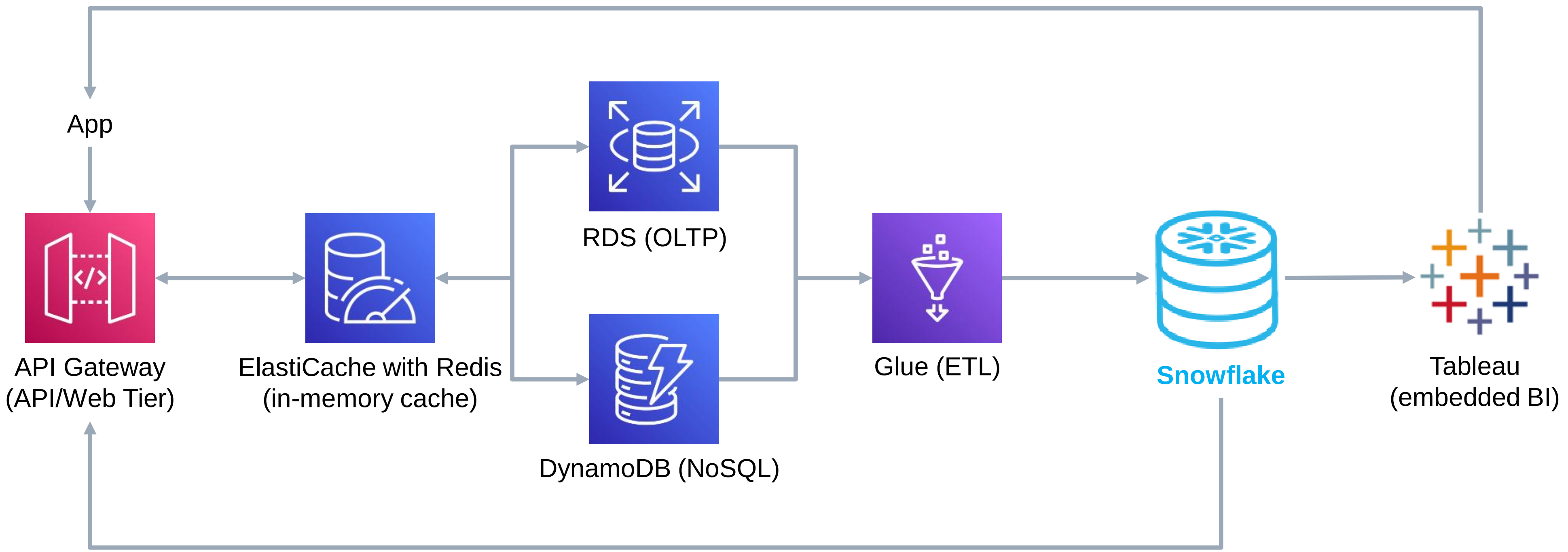
* Snowflake made Data Science/ML a top priority
* model training may require *optimized WHs* (expensive)
* no GPUs → not for NN (but see Container Services w/ NVIDIA)
* model serving w/ *multi-node standard WHs* (also expensive)
* is Snowflake ready for Data Science? → still experimental
* Snowflake Notebooks, Snowflake Cortex etc coming up

Check All

Connected Applications

Tip #95

Embedded Analytics



Checkpoints

* keep an architecture diagram w/ all your integrations
* understand how they all consume data, use caches…
* check all in-house developed applications
* check analytic apps: Tableau, Power BI, Looker…
* check ETL tools: Fivetran, dbt…

Third-Party Apps

Saving Money

Will Spend Money

Tip #96

Checkpoints

* they all install in your account → consume YOUR credits!
* principle: "*spend even more money to save money*"
* claim: "*we save you more than you spend on us*"
* they all rely on best practices and tips from this course
* this proves that Snowflake is not at all a zero-admin DW

Free Marketplace Native Apps Will Cost Money

Tip #97

Checkpoints

* no free Marketplace native app is truly free
* they all install in your account → use YOUR compute credits
* plenty of them are actually unreliable → use w/ much care!
* are better suited for expensive prod account, not test accounts
* will start an expensive STREAMLIT object
* may query metadata w/ intensive queries
* may transparently schedule expensive tasks!

Keep

App Versions

Updated

Tip #98

Checkpoints

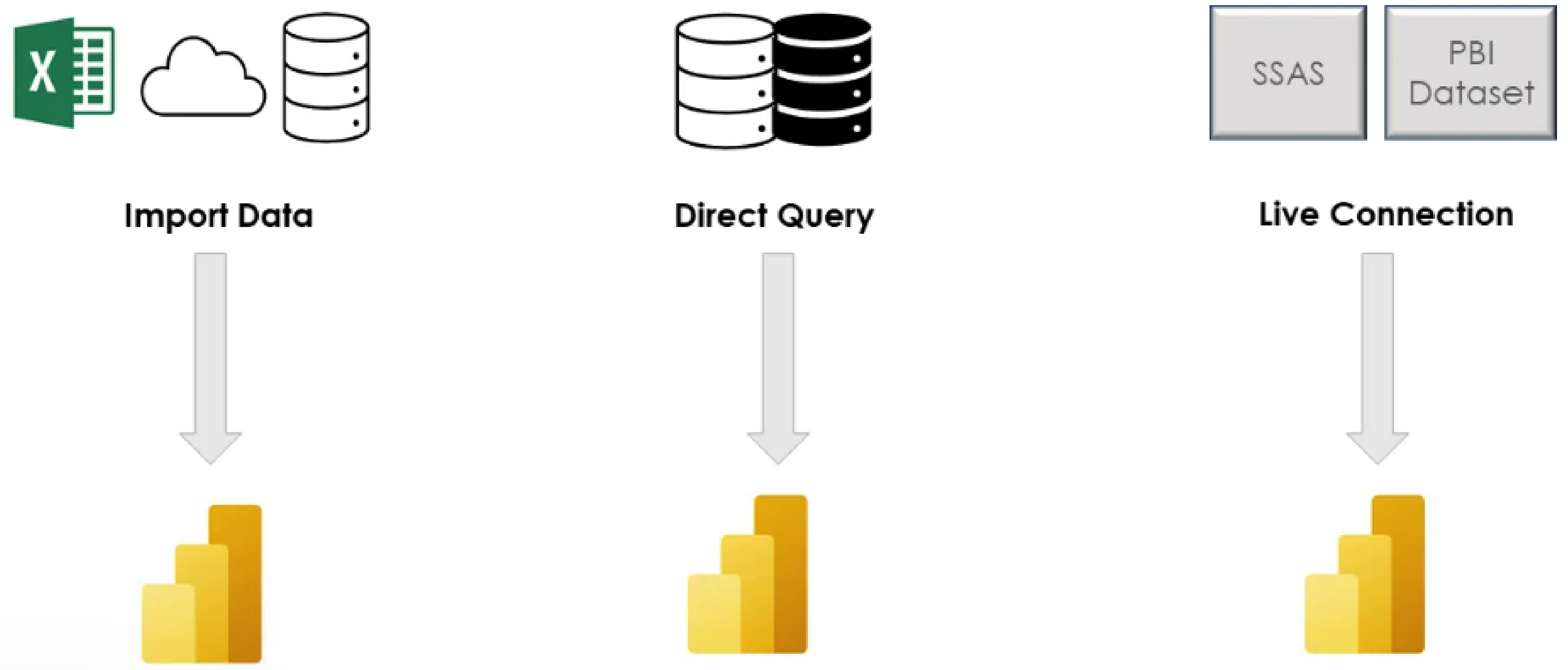
* hard to use latest performance and security updates
* WHs usually have installed older versions → check!
* check and use the latest deployed framework versions
* check and use latest deployed packages & libs
* update your Snowflake client drivers & libraries

Cache Data

in Third-Party Tools

Tip #99

Power BI Connection Types



Checkpoints

* mostly for analytic tools (Tableau, Power BI, Looker)
* avoid querying Snowflake data at every visit
* should combine access to live+cached data
* use data extracts w/ aggregations for dataviz
* avoid refreshing dashboards too frequently
* use query tags to identify intensive operations

Auto-Abort

Running Queries

from Disconnected Apps

Tip #100

Checkpoints

* conn closed on sync queries → all aborted right away
* conn closed/lost (net outage) on async queries
  - all may continue, for max 2 days!
  - STATEMENT_TIMEOUT_IN_SECONDS (account/session/WH level)
  - ABORT_DETACHED_QUERY (session level) → abort after 5 mins max

100 Snowflake Cost Optimization Techniques

100 Snowflake Cost Optimization Techniques