CS4262/5462 Machine Learning Systems
Data Systems for AI

Yao LU

2025-26 Semester 2

National University of Singapore
School of Computing

Data systems are becoming critical for AI

eeNews Europe

IBM to Acquire Confluent in $11B Deal to Build Smart Data Platform for Enterprise AI

IBM to Acquire Confluent in $11B Deal to Build Smart Data Platform for Enterprise AI ... IBM has announced its intent to acquire Confluent for US...

8 Dec 2025

Deloitte

Where is the value of AI in M&A: why multi-agent systems needs modern data architecture

Financial institutions and M&A professionals today face an explosion of data, faster deal cycles and heightened complexity in...

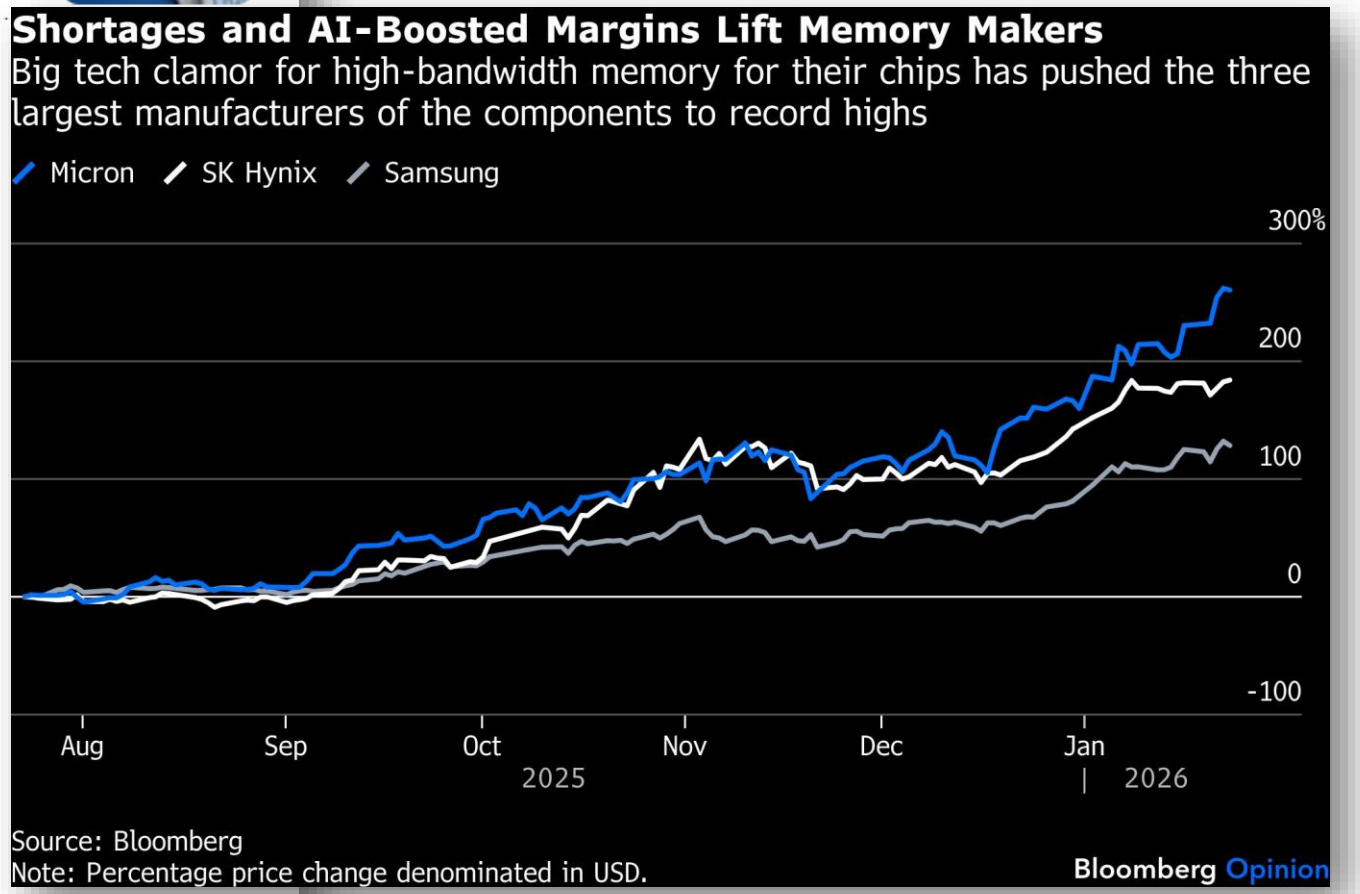
22 Jul 2025

CRN

Databricks Bolsters Lakebase Database 'Vision' With Latest Acquisition

Databrick's purchase of startup Mooncake will accelerate the ability of the Lakebase OLTP database to provide data for AI agents and...

2 Oct 2025



Overview

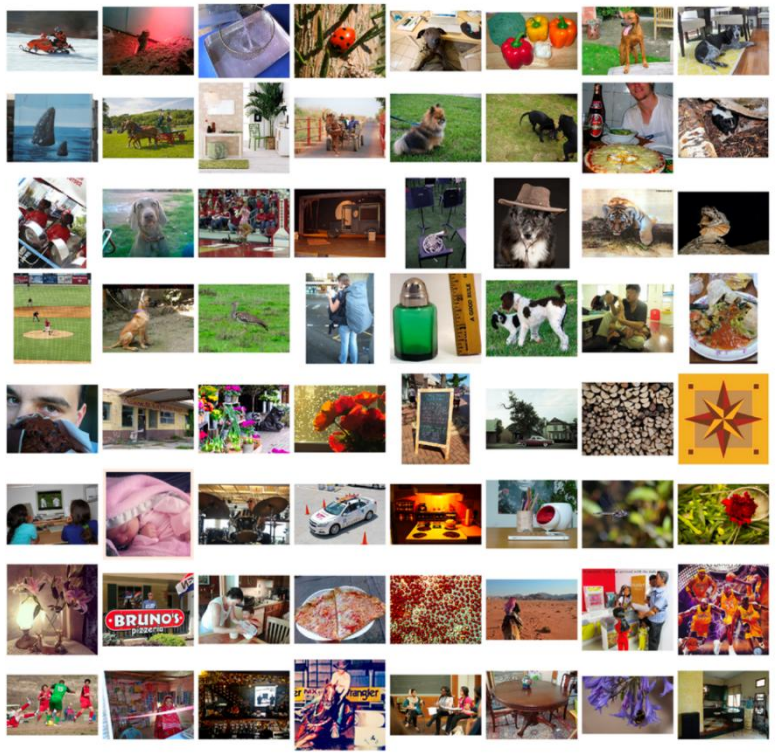
- Vector databases
  - Main-memory vector index
  - Generalized vector DBs vs Specialized vector DBs

- Analytics systems
  - Data lakes and warehouses
  - Case studies

Recent vector databases

Motivation 1: vector embedding

Unstructured Data



Vectors

0.03, 0.12, 0.01, ...

0.01, 0.01, 0.02, ...

0.02, 0.02, 0.03, ...

0.04, 0.11, 0.05, ...

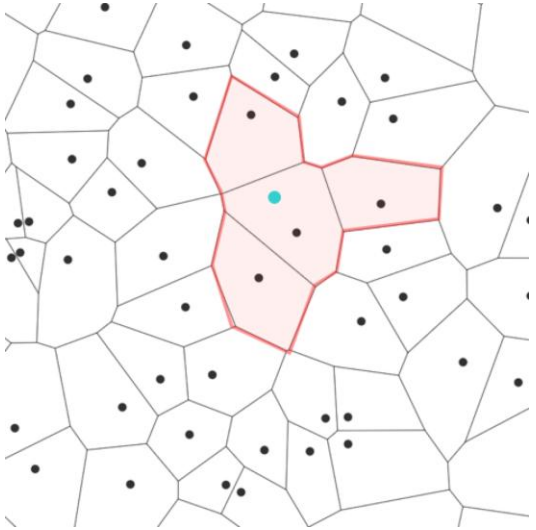
⋮

0.11, 0.13, 0.01, ...

0.06, 0.01, 0.02, ...

0.02, 0.02, 0.03, ...

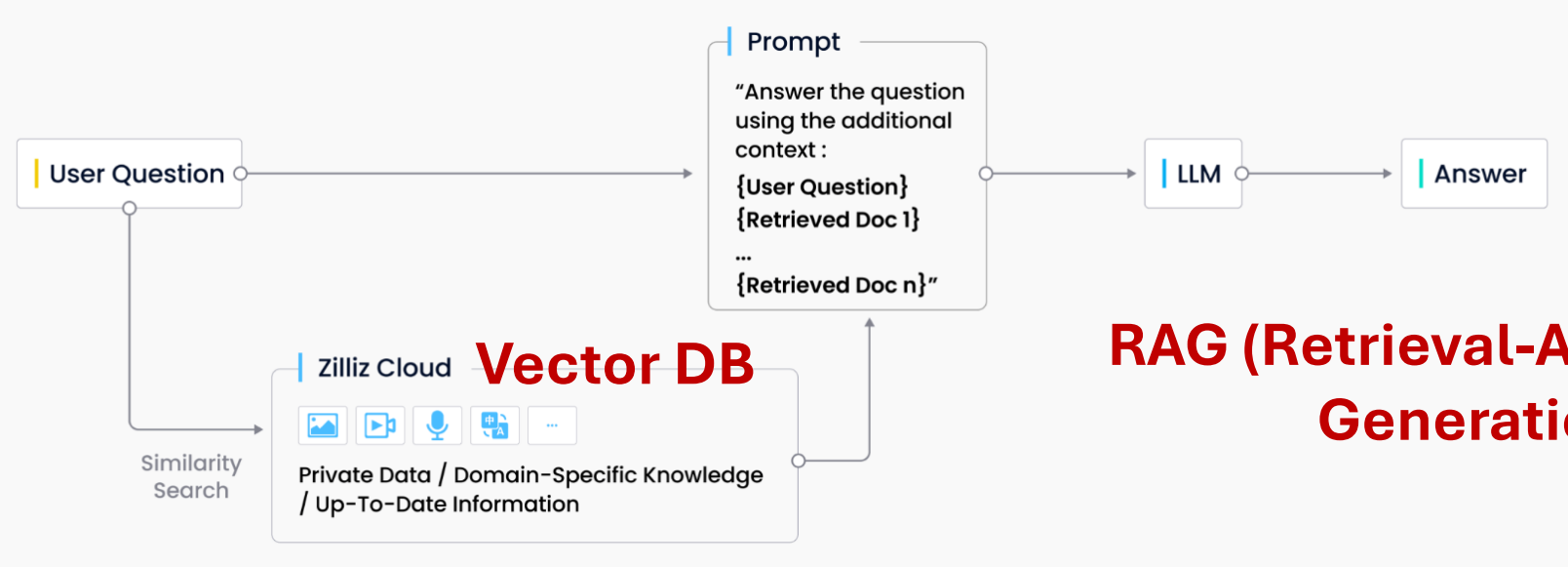
Analytics in Vector Space



Known as “vector embedding” (due to deep learning)

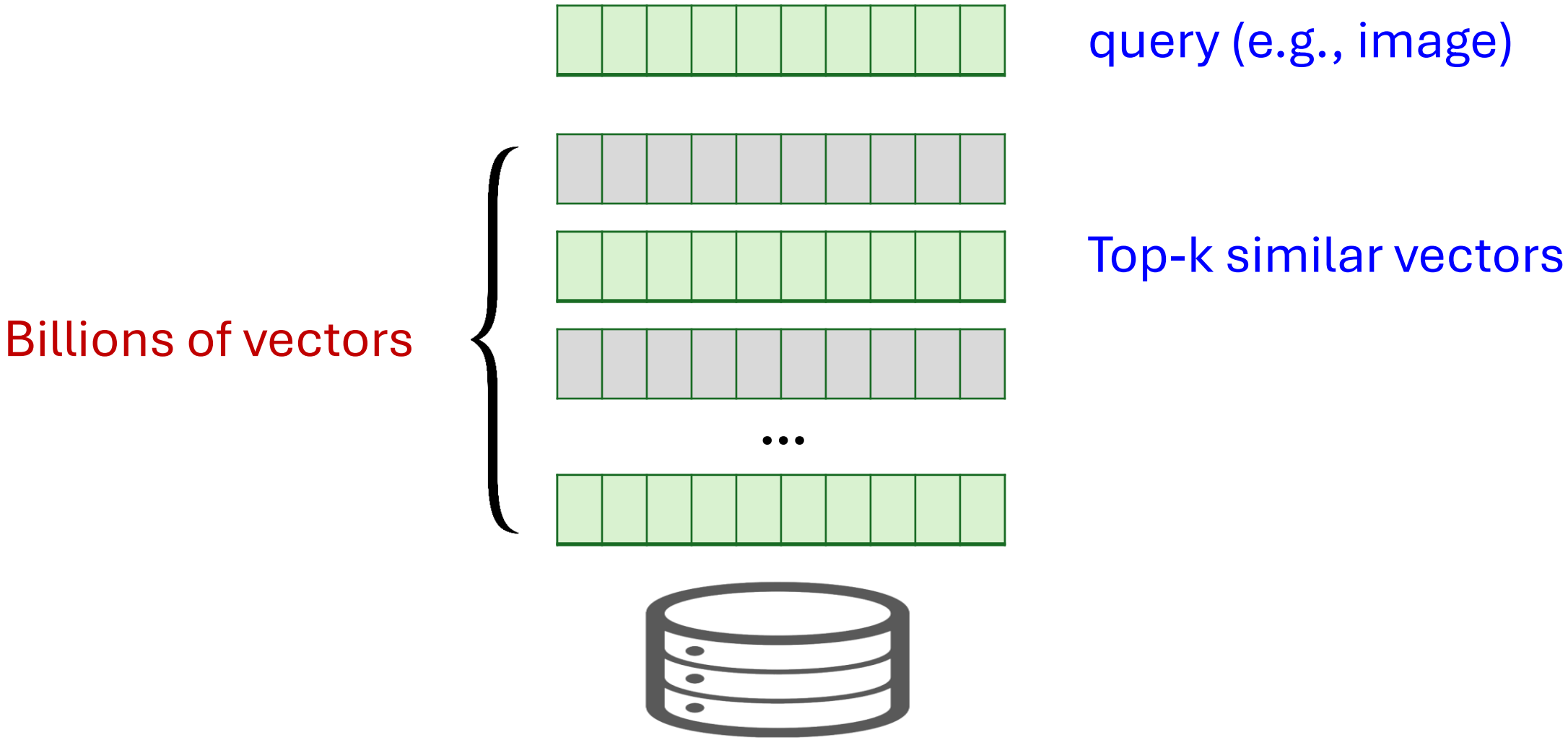
Motivation 2: large language models

- Vector DBs & RAGs address many critical limitations of LLMs
  - Hallucination: incorrect or fabricated answer
  - Lacking domain-specific knowledge
  - Up-to-date information



https://zilliz.com/use-cases/llm-retrieval-augmented-generation

Key operator in vector DBs: vector similarity search

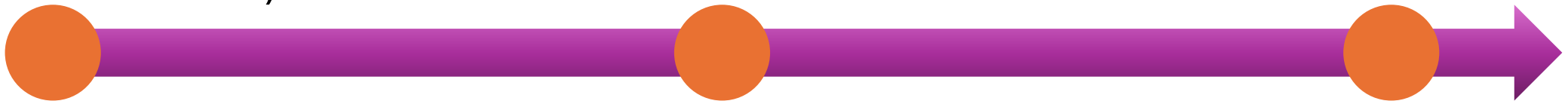


Evolution of vector data(base)

1999
(Content-based information retrieval)

2013
(Embedding)

2023
(LLM)



Similarity Search in High Dimensions via Hashing

Aristides Gionis * Piotr Indyk† Rajeev Motwani‡
Department of Computer Science
Stanford University
Stanford, CA 94305
{gionis,indyk,rajeev}@cs.stanford.edu

Locality-sensitive hash

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Retrieval

OpenAI

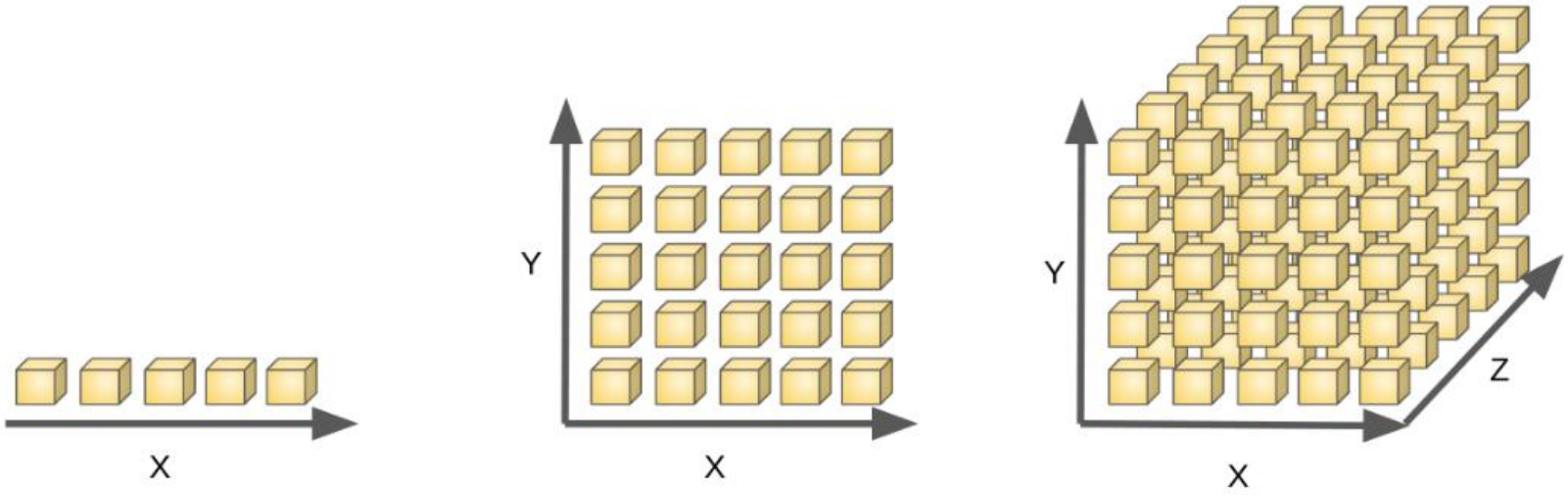
The open-source retrieval plugin enables ChatGPT to access personal or organizational information sources (with permission). It allows users to obtain the most relevant document snippets from their data sources, such as files, notes, emails or public documentation, by asking questions or expressing needs in natural language.

As an open-source and self-hosted solution, developers can deploy their own version of the plugin and register it with ChatGPT. The plugin leverages OpenAI embeddings and allows developers to choose a vector database (Milvus, Pinecone, Qdrant, Redis, Weaviate or Zilliz) for indexing and searching documents. Information sources can be synchronized with the database using webhooks.

Why are vector DBs challenging?

- Easy to get started, but very challenging to achieve high performance, accuracy, and efficiency
- Three unique properties that contribute to the challenges of vector DBs
  - Property P1: Curse of Dimensionality
  - Property P2: Approximation
  - Property P3: Advanced Vector Data Analytics

P1: Curse of dimensionality

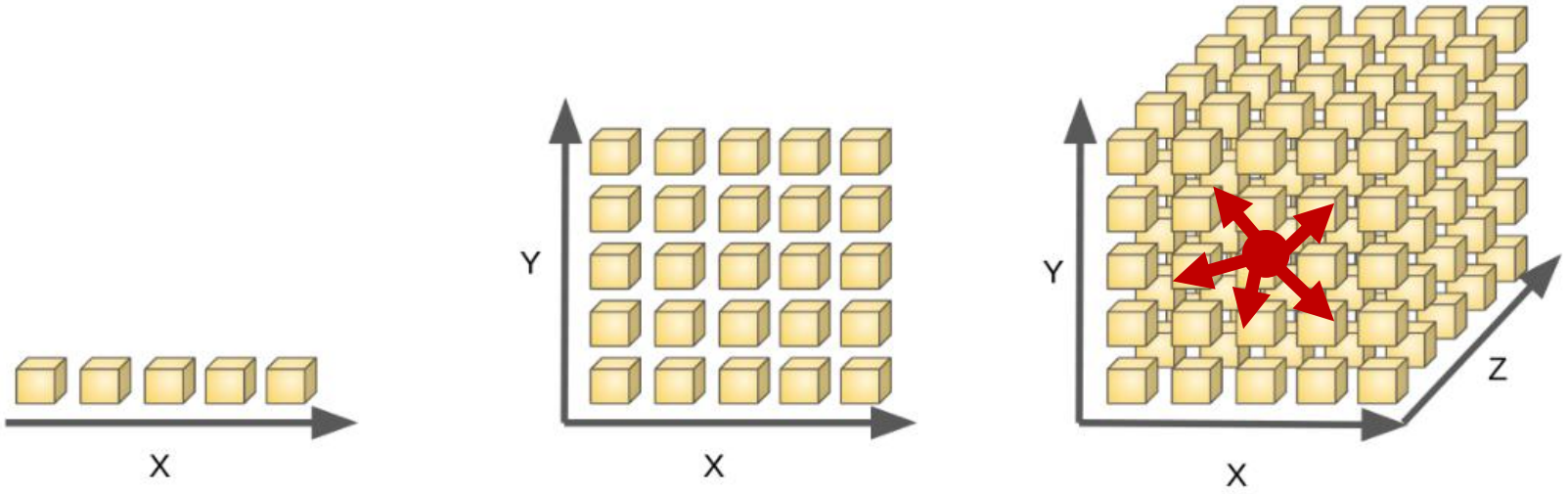


All techniques fail on high-d space (for exact answers)
➔ approximate answer!

All vector DBs return approximate answer

https://medium.com/@soumiksanku08/curse-of-dimensionality-293d0d16fe2a

P1: Curse of dimensionality



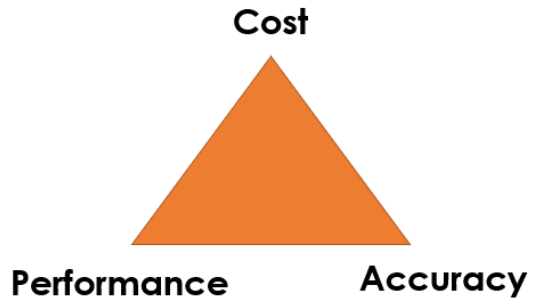
Querying in high-dimensional space
➔ no locality ➔ hard to leverage tiered storage

Almost all vector DBs only use DRAM ➔ too expensive (in the cloud)

https://medium.com/@soumiksanku08/curse-of-dimensionality-293d0d16fe2a

P2: Approximation

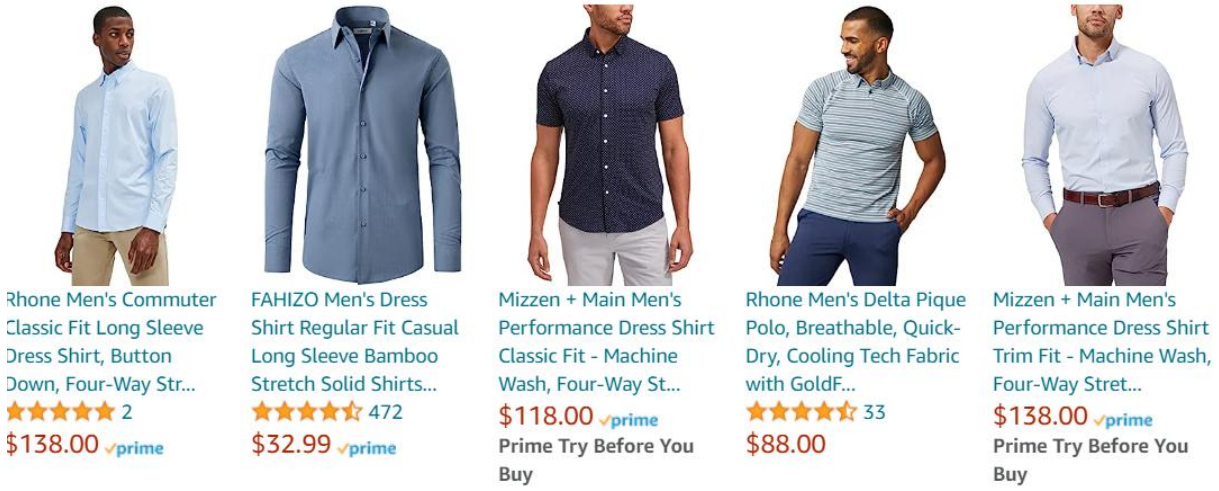
- Approximation introduces a new design tradeoff (in addition to performance and cost)
  - Complicates system design
  - Different from traditional databases
  - Approximate query processing is still not the mainstream
- Many vector indexes are developed with different tradeoffs
- How to make the right tradeoff between CAP?
  - Existing databases rely on users' manual selection



P2: Approximation

- Need to consider approximation from the ground-up
  - Index type selection
  - Index parameter tuning
  - Caching
    - If there's a cache miss, do you want to return current results or go to disk to compute accurate answer?
  - Compression
  - Consistency
  - Visibility
  - …

P3: Advanced query processing



Finding the T-shirts similar to a given image vector that also cost less than $100 and have text descriptions containing specific keywords

- Query is more than pure vector search
- Can contain filters (attribute filters / range filters), and other non-vector data (e.g., relational / document / graph / spatial data)
- Necessary to support advanced RAGs

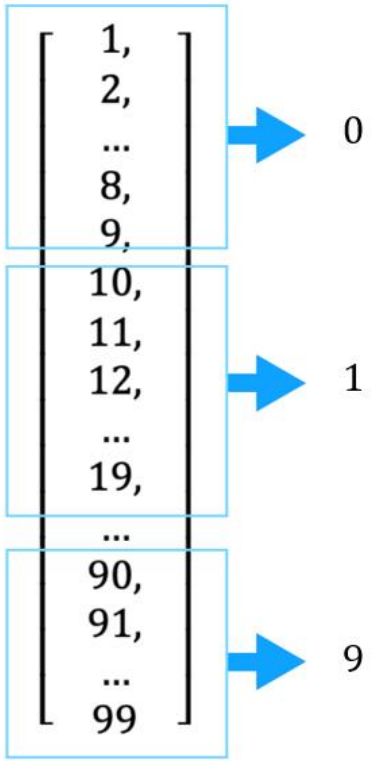
Vector indexes (main memory)

- Quantization-based indexes
  - E.g., IVF_FLAT, IVF_PQ
- Graph-based indexes
  - E.g., NSW, HNSW
- Tree-based indexes
- Hash-based indexes

Widely used in vector DBs

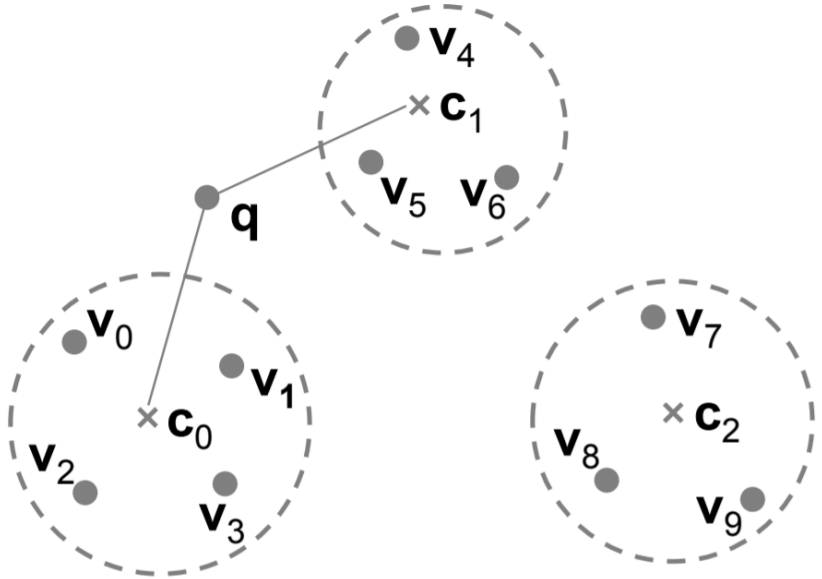
Quantization

- What's quantization?
  - A way of approximation
- Let's look at quantization in 1-dimensional space
  - $Q(x) = \left\lfloor \frac{x}{10} \right\rfloor$, where *x* is an input value
  - input = 3, $Q(3) = \left\lfloor \frac{3}{10} \right\rfloor = \lfloor 0.3 \rfloor = 0$
  - input = 3, $Q(91) = \left\lfloor \frac{91}{10} \right\rfloor = \lfloor 9.1 \rfloor = 9$
  - Those 99 integers can be quantized into a smaller set of 10 buckets



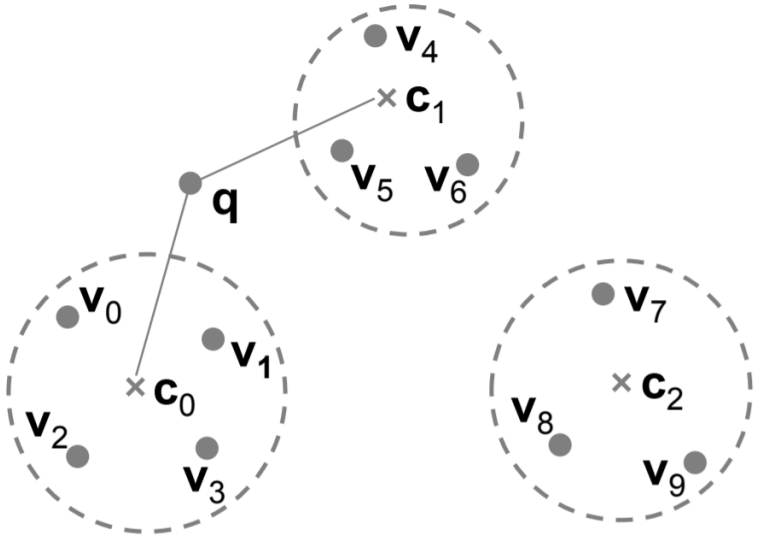
Quantization

- What's quantization in high-dimensional space?
  - It's basically clustering, e.g., k-means



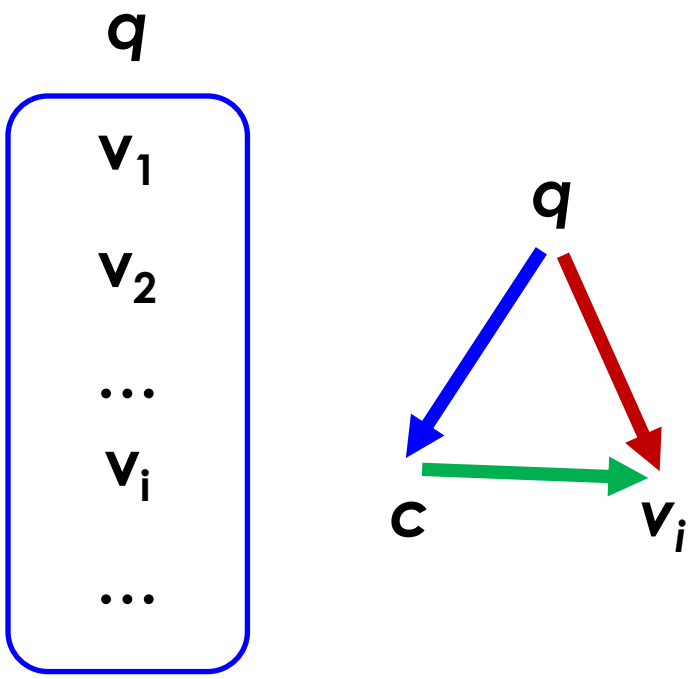
IVF_FLAT

- Index phase
  - Cluster *n* vectors into *K* clusters (quantization)
  - Centroids: $c_0 \ldots c_{K-1}$
- Search phase
  - Given a query *q*, find the closest *u* clusters based on centroids
    - *u*: user-defined parameter
  - Only scan the vectors in the *u* clusters



IVF_FLAT

- Question: how to quickly compute the similarity between **q** and a vector $\mathbf{v}_i$ in a cluster?
- Naïve approach
  - A for-loop to compute dist($\mathbf{q}$,$\mathbf{v}_i$)
  - *d* steps (where *d* is dimensionality, e.g., *d* = 1000)
- Better solutions?
  - Remember, we know the centroid **c**
  - We can pre-compute the distance of dist($\mathbf{c}$,$\mathbf{v}_i$)
- Then dist($\mathbf{q}$,$\mathbf{v}_i$) = dist($\mathbf{q}$,$\mathbf{c}$) + dist($\mathbf{c}$,$\mathbf{v}_i$) (approx.)
  - Only need 1 step to compute distance for all v_i



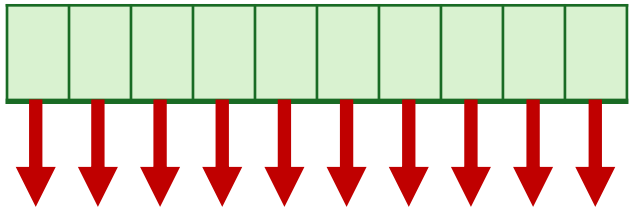
Compression

- How to reduce the space overhead of IVF_FLAT?
  - Compression
- Example
  - Youtube-8M data includes 1.4 billion vectors
  - Each vector takes 1024 dimensions (each float takes 32 bits)
  - 5.6TB space (memory!)

https://dl.acm.org/doi/10.14778/3424573.3424580

Compression: basic idea

- Instead of using 32 bits to represent a float number
- Use L bits (e.g., $L = 8$)
- Think of 1-d quantization
- Every float number in a vector is quantized into $[0 \ldots 2^L-1]$
- The 1.4billion vectors will take 1.4TB space (if $L = 8$)

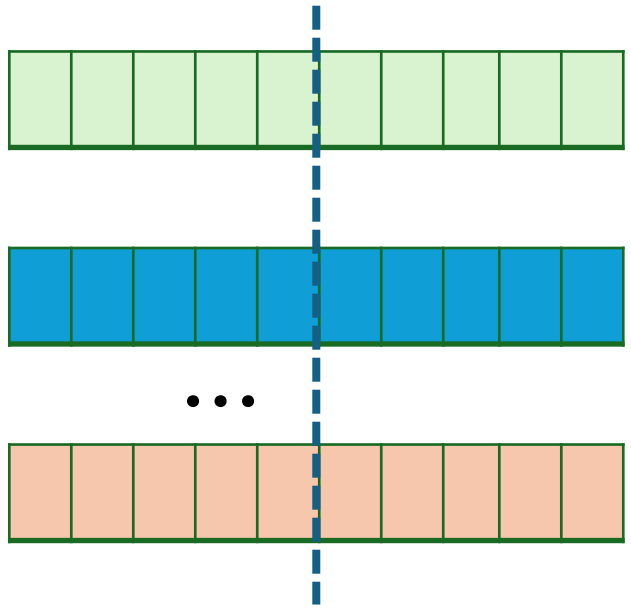


vector

Every float number is mapped to [0…255] (8 bits per number)

Compression: product quantization (PQ)

- How to further reduce the space overhead?
- Product quantization (PQ)
  - Key idea: compress between multiple dimensions
  - Every vector is partitioned into *M* subvectors, e.g., *M* = 8
  - Every subvector is compressed using *L* bits (e.g., *L* = 8)

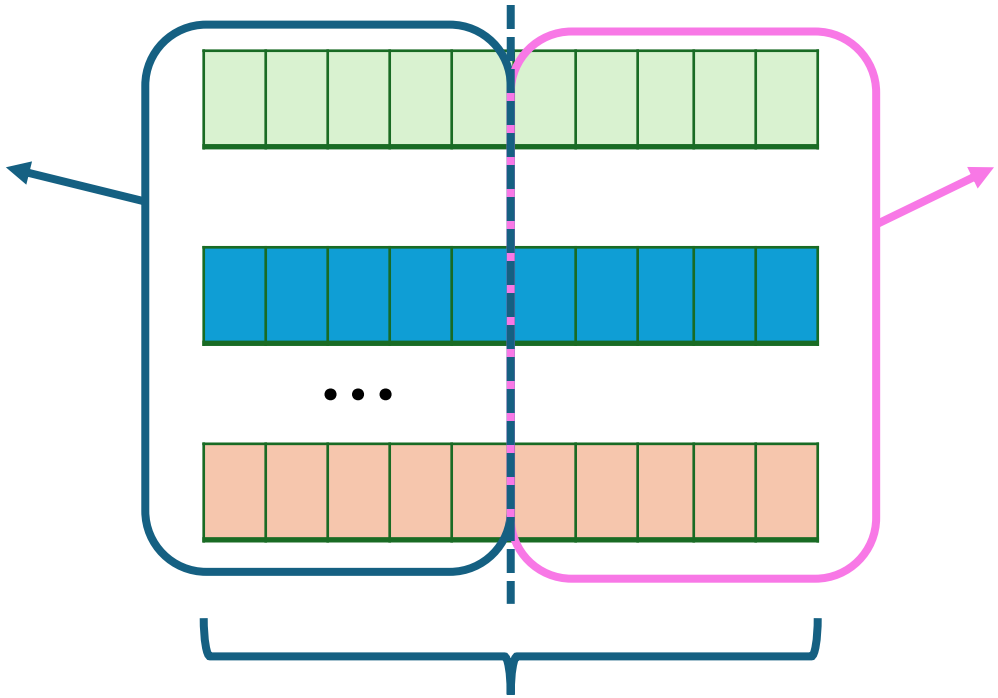


Compression: product quantization (PQ)

- How to compress subvectors?
- Each vector v_i is partitioned into *M* subvectors $v_i^0 \ldots v_i^{M-1}$
  - *M* subspace
- All the vectors in the same subspace are compressed together using high-dimensional quantization (clustering)
  - All $v_0^0, v_1^0, v_2^0 \ldots, v_{n-1}^0$ are compressed together
  - All $v_0^1, v_1^1, v_2^1 \ldots, v_{n-1}^1$ are compressed together
  - Every subvector is represented using the centroid ID

Compression: product quantization (PQ)

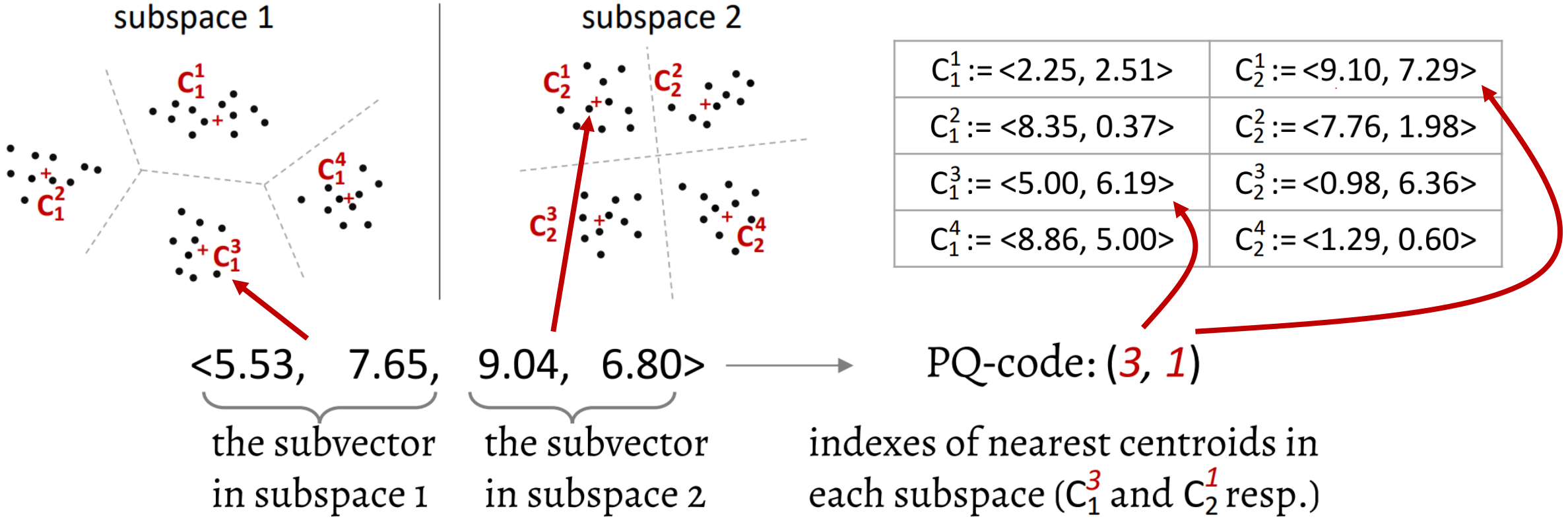
K-means clustering (every subvector is encoded using the centroid ID)



K-means clustering (every subvector is encoded using the centroid ID)

Every vector is split into 2 parts: head and tail vector
All the head vectors will be compressed together
All the tail vectors will be compressed together

Compression: product quantization (PQ)



Original vector is compressed as <5.00, 6.19, 9.10, 7.29>

https://dl.acm.org/doi/10.14778/3424573.3424580

Compression: product quantization (PQ)

- Each vector is compressed using M*L bits
  - E.g., M = 8, L = 8
- Regardless of the dimensionality
  - But the parameters can be tuned based on dimensionality
- Example: Consider the 1.4 billion vectors again
  - Each vector will take 8*8 bits (M = 8, L = 8), i.e., 8 bytes
  - The 1.4billion vectors will take 11.2GB space

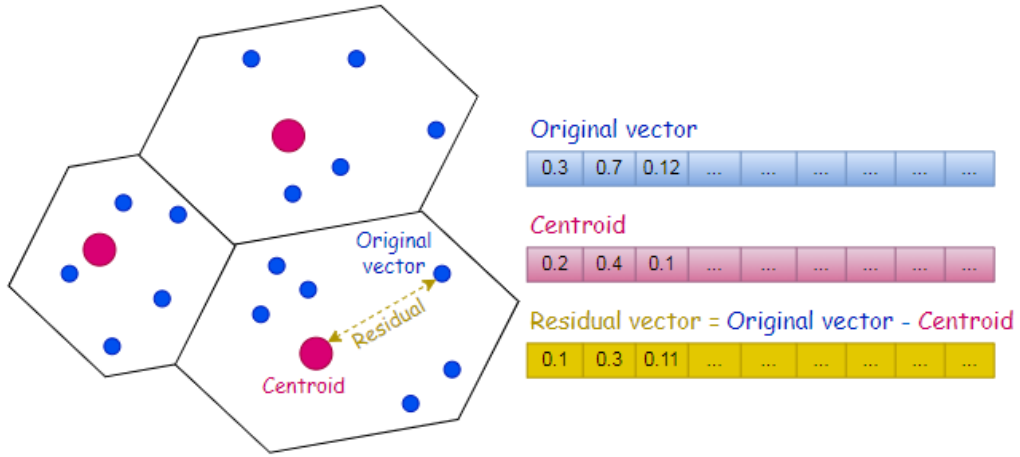
Compression: product quantization (PQ)

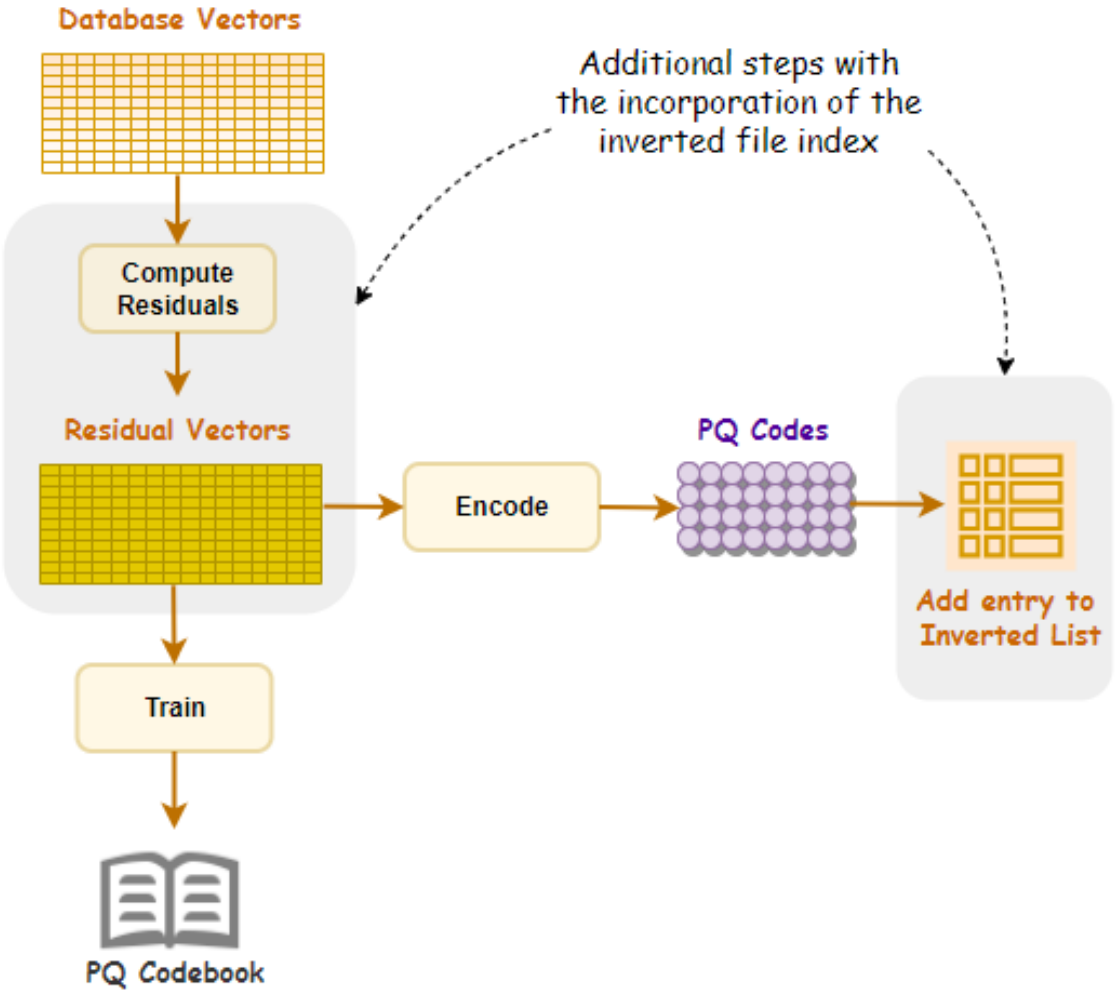
- What's the tradeoff? Space vs. accuracy

- Another benefit of PQ: Fast distance computation
  - All the distance in subspace can be precomputed
  - Example:
    - Vector X → PQ code (3, 1)
    - Vector Y → PQ code (1, 5)
    - Dist(X,Y) = dist(3,1) + dist(1,5), where each part can be precomputed

IVF_PQ

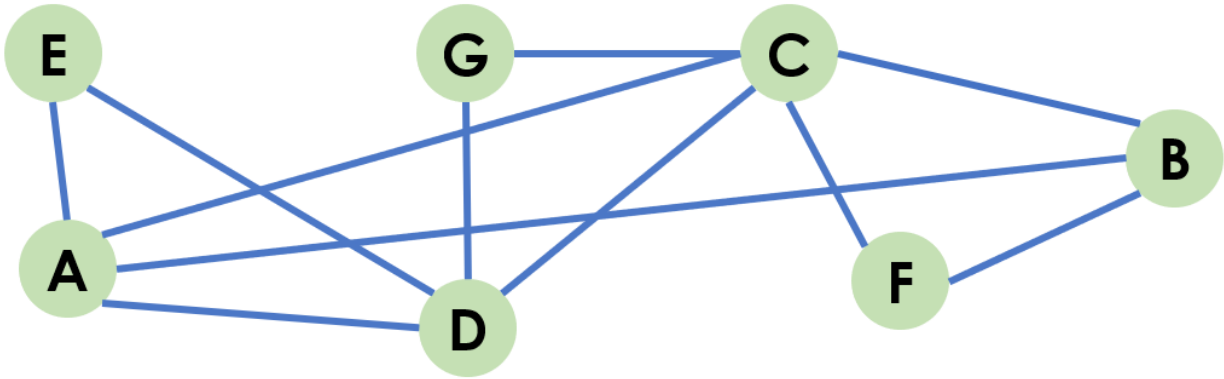
- Similar as IVF_FLAT
- Difference is that
  - Each cluster applies PQ
  - using residual vectors
- Search process is the same





Graph-based vector index

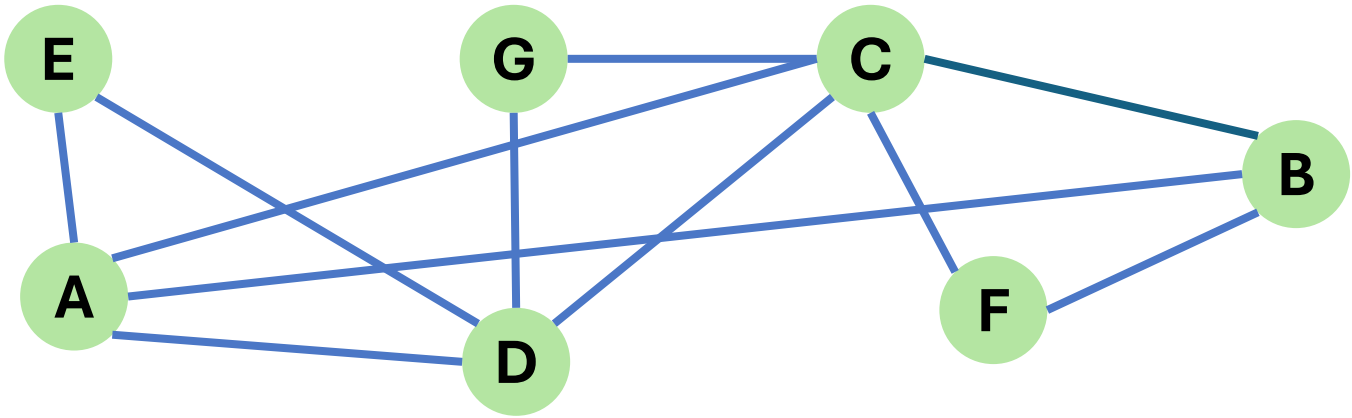
- Key ideas
  - For each vector, pre-compute the nearest neighbors
  - Connect them using a graph
  - Convert vector search problem to graph traversal problem



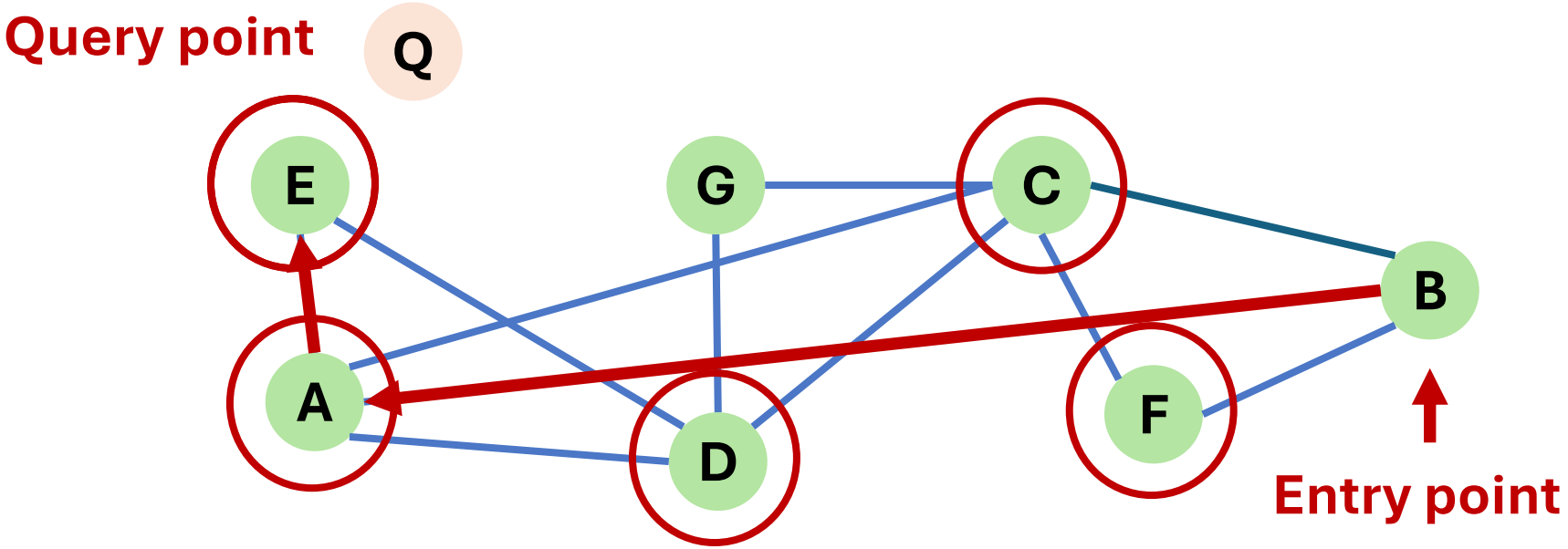
Graph-based vector index

- Navigable Small Worlds (NSW)
  - Add new vertices to the index
  - For each new vertex (vector), find the closest *m* neighbors seen so far and connect with them
  - Balance: index construction time & query performance

***m* = 2**

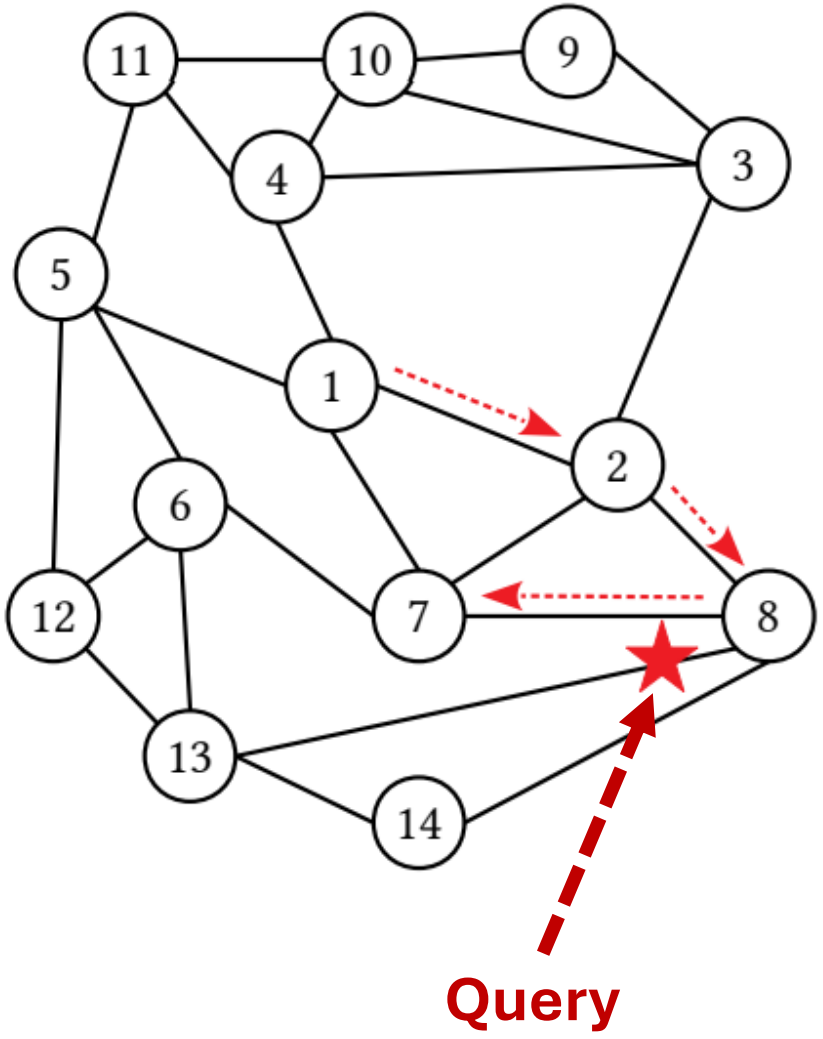


Graph-based vector index



Can be extended to *k*NN by maintaining a result set and a candidate set

Terminate if the max distance in the result set < min distance in the candidate set



Candidate set

Result set

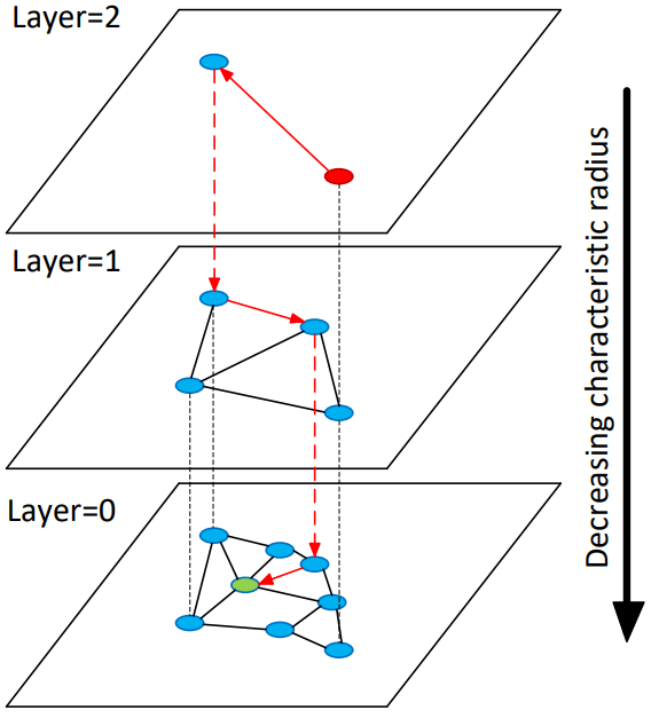
Initialization	*q:*	1
	topk:	∅
	visited:	1
Iteration 1	*q:*	2 7 4 5
	topk:	1
	visited:	1 2 4 5 7
Iteration 2	*q:*	8 7 3 4 5
	topk:	1 2
	visited:	1 2 3 4 5 7 8
Iteration 3	*q:*	7 3 4 5 14 13
	topk:	1 2 8
	visited:	1 2 3 4 5 7 8 13 14
Iteration 4	*q:*	3 4 5 6 14 13
	topk:	7 2 8
	visited:	1 2 3 4 5 6 7 8 13 14

K = 3

https://ieeexplore.ieee.org/abstract/document/9101583

Graph-based vector index

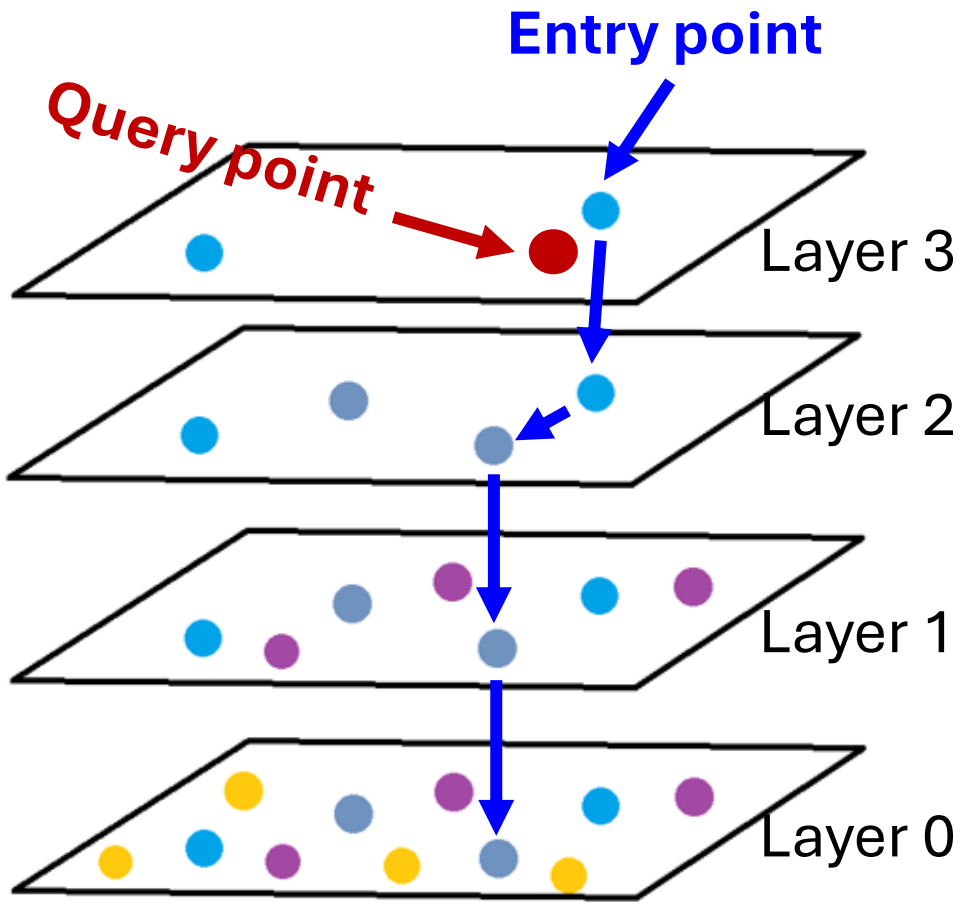
- Hierarchical Navigable Small Worlds (HNSW)
  - Skip list + NSW
  - Multi-layered NSW
  - Address the “bad” entry point issue
    - If the entry point is not selected properly, the search path is long



https://arxiv.org/ftp/arxiv/papers/1603/1603.09320.pdf

Graph-based vector index

- Every layer is an NSW on the sampled vertices
- Find the nearest vector in each layer, which will serve as the entry point for the next layer
- What if I choose a bad entry point from the top layer?
  - Slow, but acceptable
  - As the num of points is small in top layer



Entry point for layer 0 (much closer to query than a randomly selected one)

Overview

- Vector databases
  - Main-memory vector index
  - Generalized vector DBs vs Specialized vector DBs

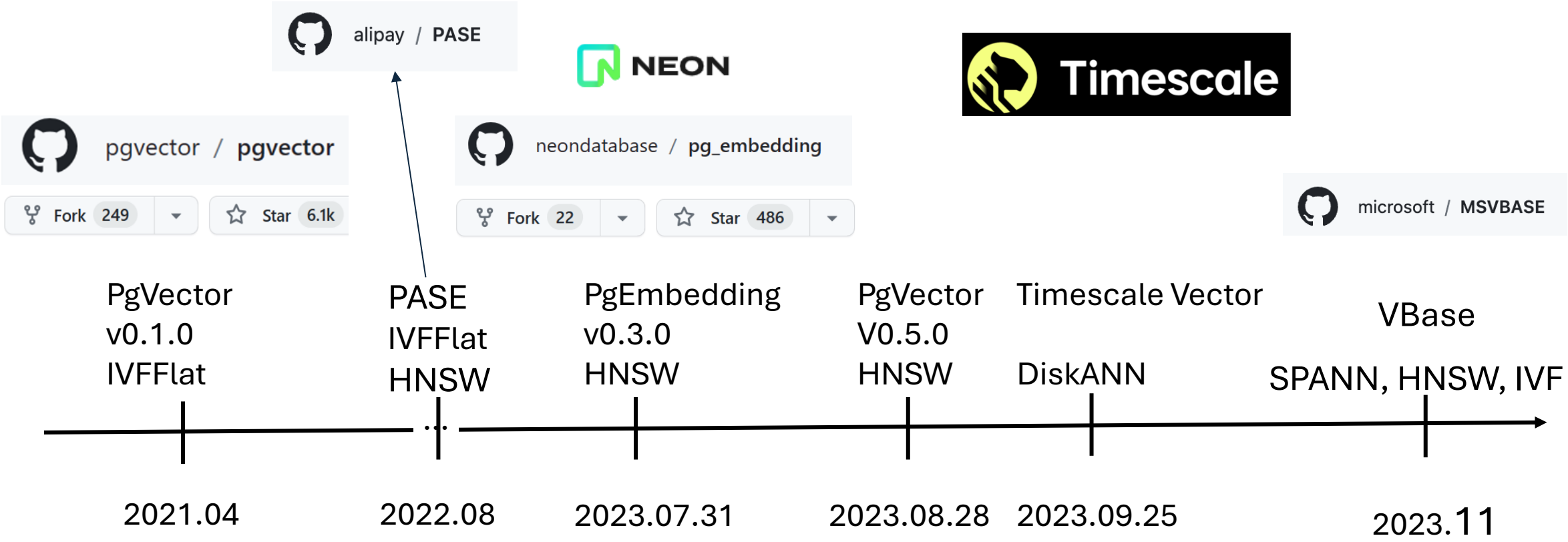
- Analytics systems
  - Data lakes and warehouses
  - Case studies

Vector databases: specialized vs. generalized

- **Specialized vector databases**
  - Explicitly designed for vector data

- **Generalized vector databases**
  - Support vector search within relational databases
  - One-size-fits-all

Vector search in PostgreSQL



How We Made PostgreSQL a Better Vector Database

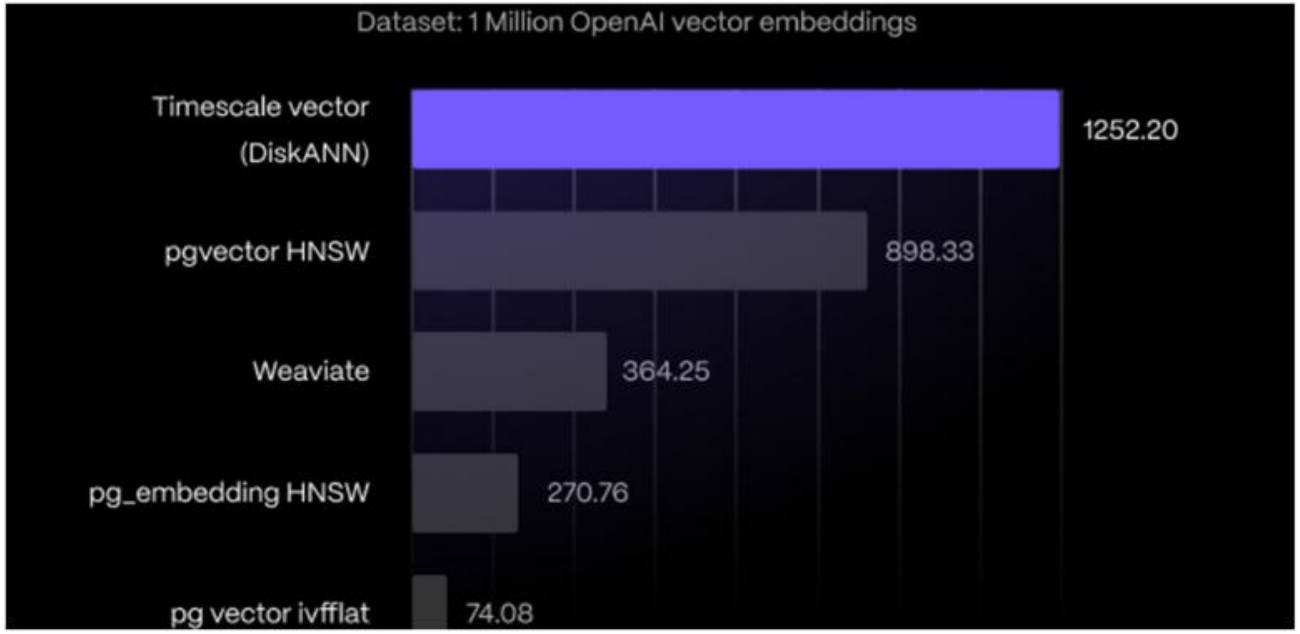
Try Timescale Vector

PostgreSQL++ for AI Applications

Get started for free

Blog Categories

All posts
AI
Announcements
Cloud
Developer Q&A
Engineering
General
Grafana
Observability
PostgreSQL
Product Updates



Introducing Timescale Vector, PostgreSQL++ for production AI applications. Timescale Vector enhances pgvector with faster search, higher recall, and more efficient time-based filtering, making PostgreSQL your new

25 Sep 2023
24 min read

AI

Contributors
Avthar Sewrathan
Matvey Arye
Samuel Gichohi
Maheedhar PV

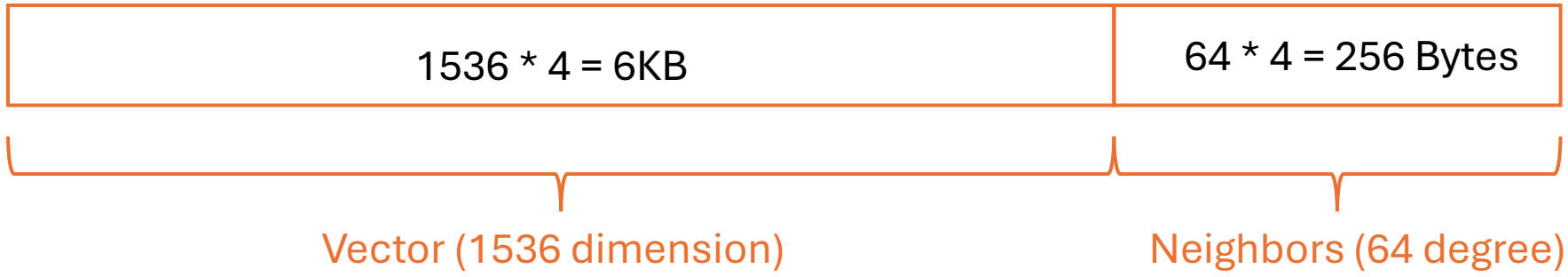
Share

How We Made PostgreSQL a Better Vector Database (timescale.com)

Timescale-vector

[1]Jayaram Subramanya, Suhas, et al. "Diskann: Fast accurate billion-point nearest neighbor search on a single node." Advances in Neural Information Processing Systems 32 (2019).

- ➔ Inspired by DiskANN[1] (Optimized for disk)
  - ◆ On-disk data layout
    - Cluster each node vector with its neighboring links
  - ◆ Single layer graph can further augment the cache's efficiency
    - Unlike HNSW's hierarchical structure



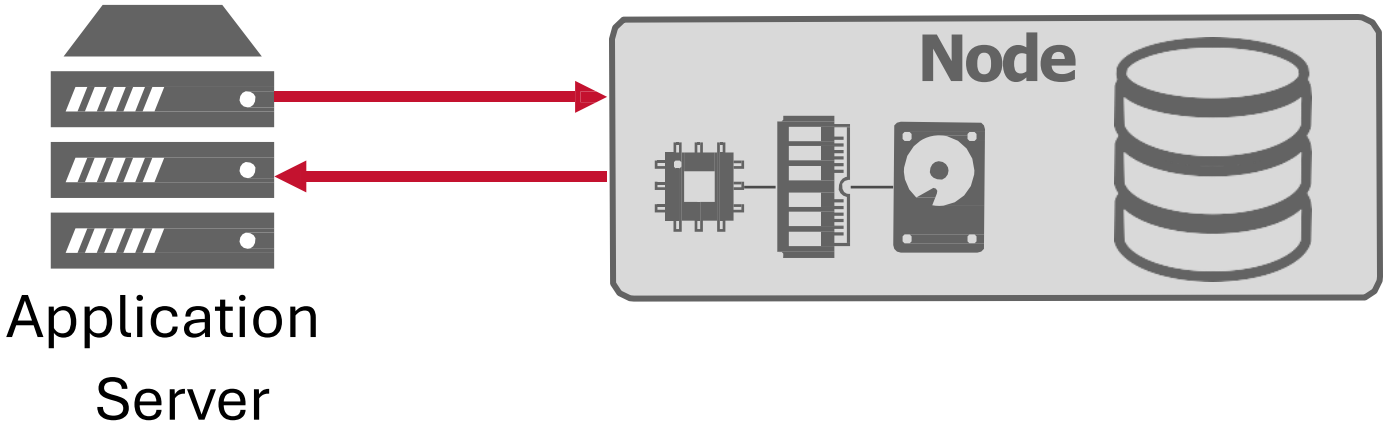
Overview

- Vector databases
  - Main-memory vector index
  - Generalized vector DBs vs Specialized vector DBs

- Analytics systems
  - Data lakes and warehouses
  - Case studies

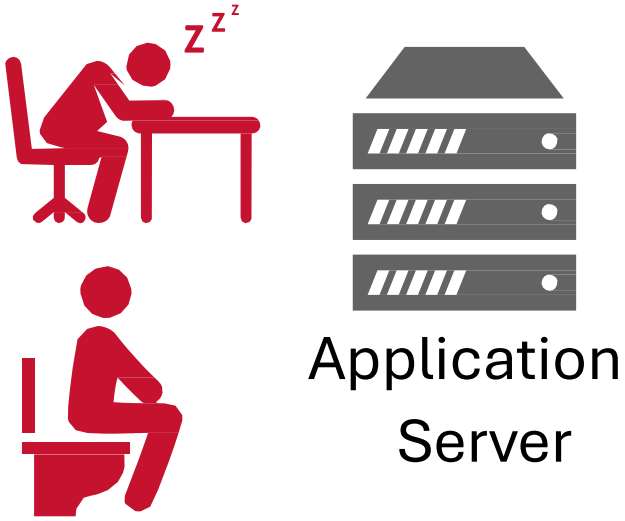
Serverless databases

- Rather than always maintaining compute resources for each customer, a "serverless" DBMS evicts tenants when they become idle.



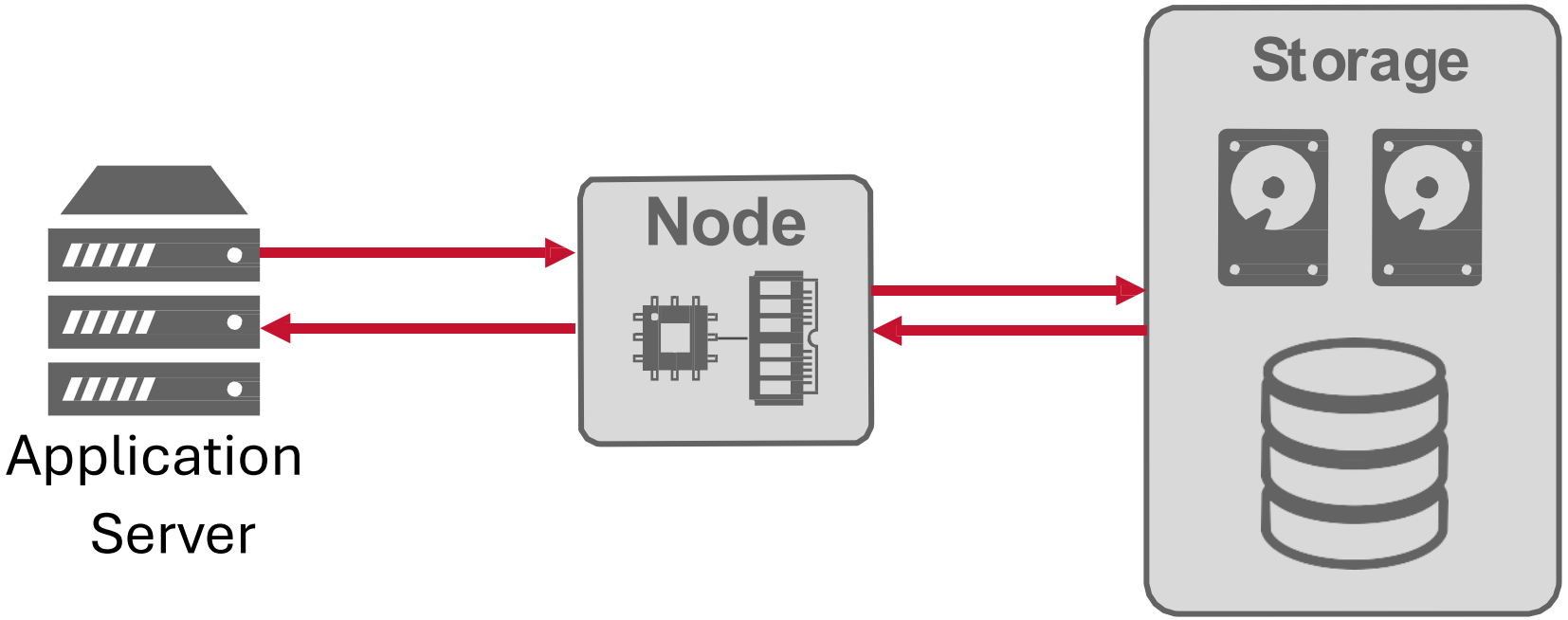
Serverless databases

- Rather than always maintaining compute resources for each customer, a "serverless" DBMS evicts tenants when they become idle.



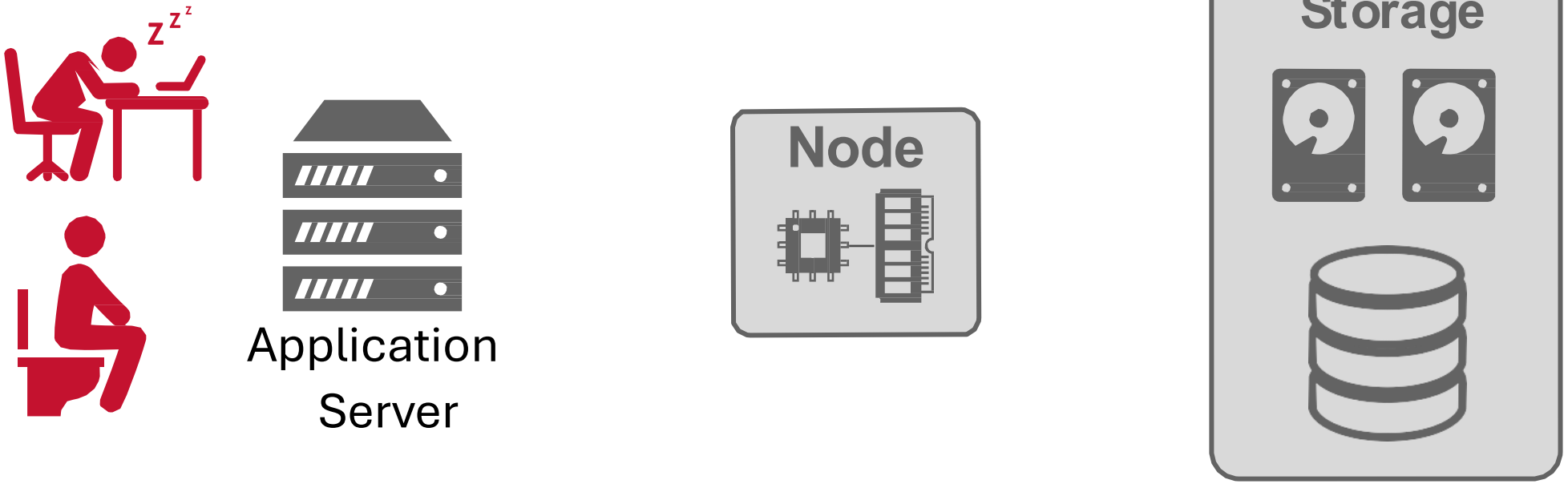
Serverless databases

- Rather than always maintaining compute resources for each customer, a "serverless" DBMS evicts tenants when they become idle.



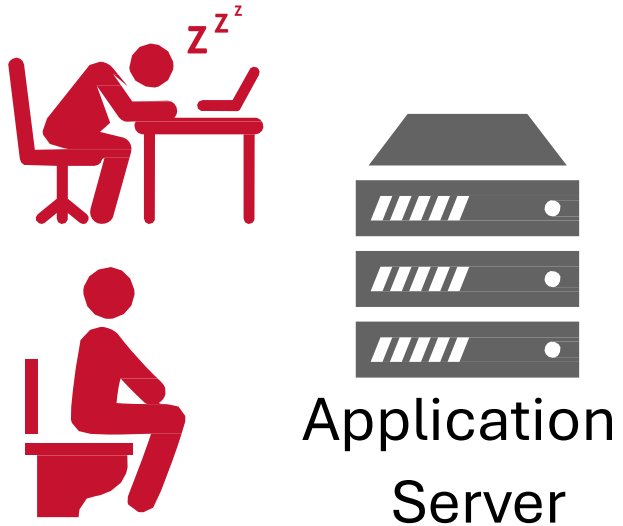
Serverless databases

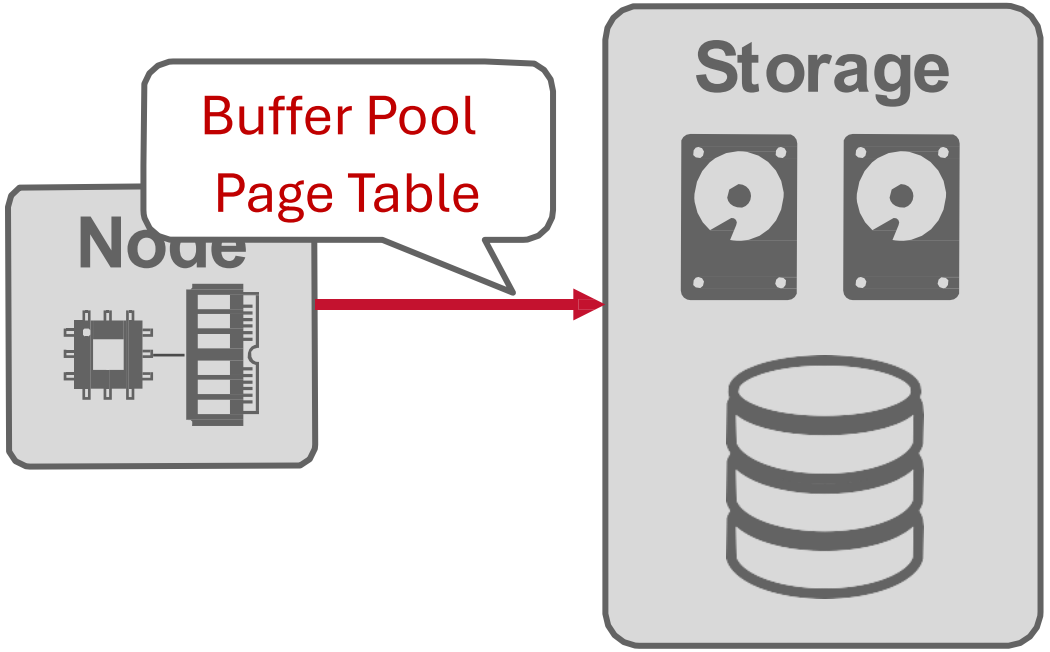
- Rather than always maintaining compute resources for each customer, a "serverless" DBMS evicts tenants when they become idle.



Serverless databases

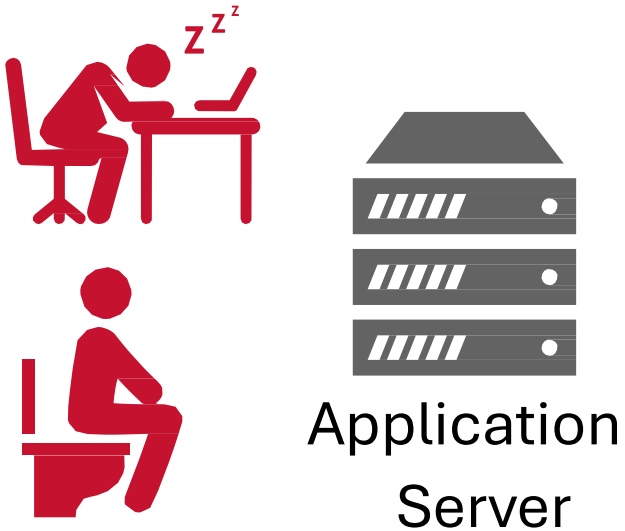
- Rather than always maintaining compute resources for each customer, a "serverless" DBMS evicts tenants when they become idle.

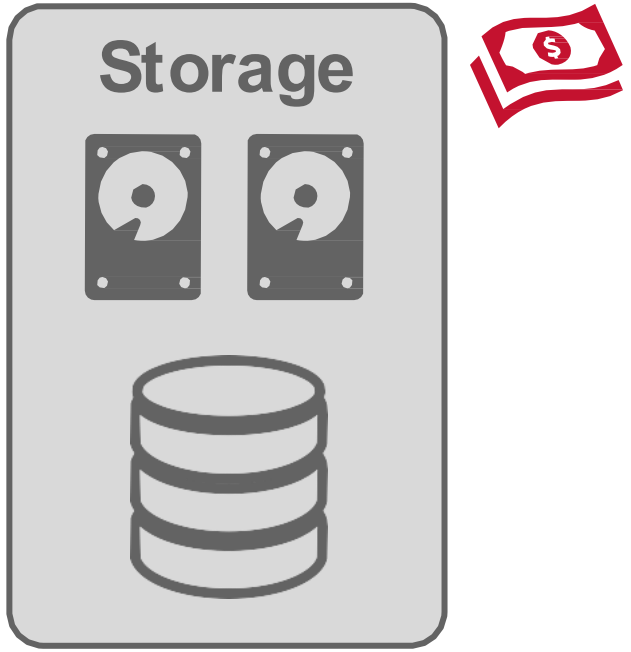




Serverless databases

- Rather than always maintaining compute resources for each customer, a "serverless" DBMS evicts tenants when they become idle.

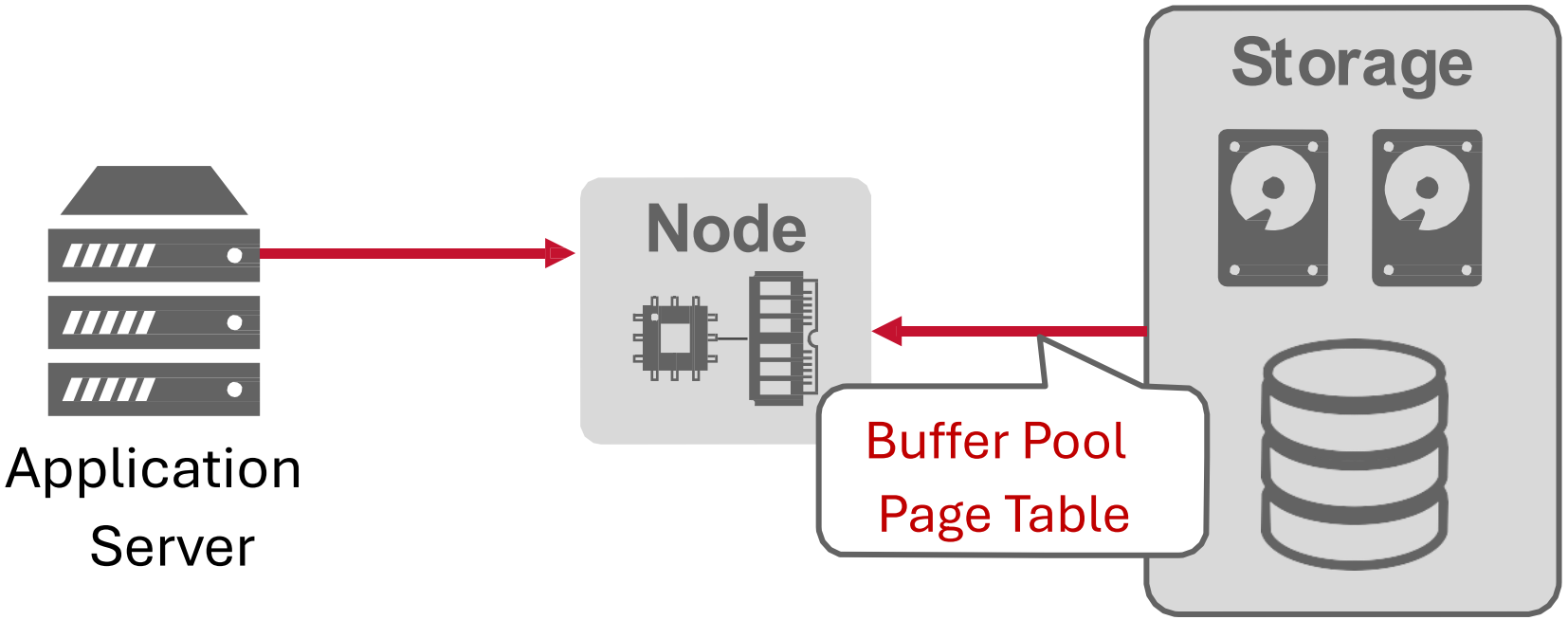




Serverless databases

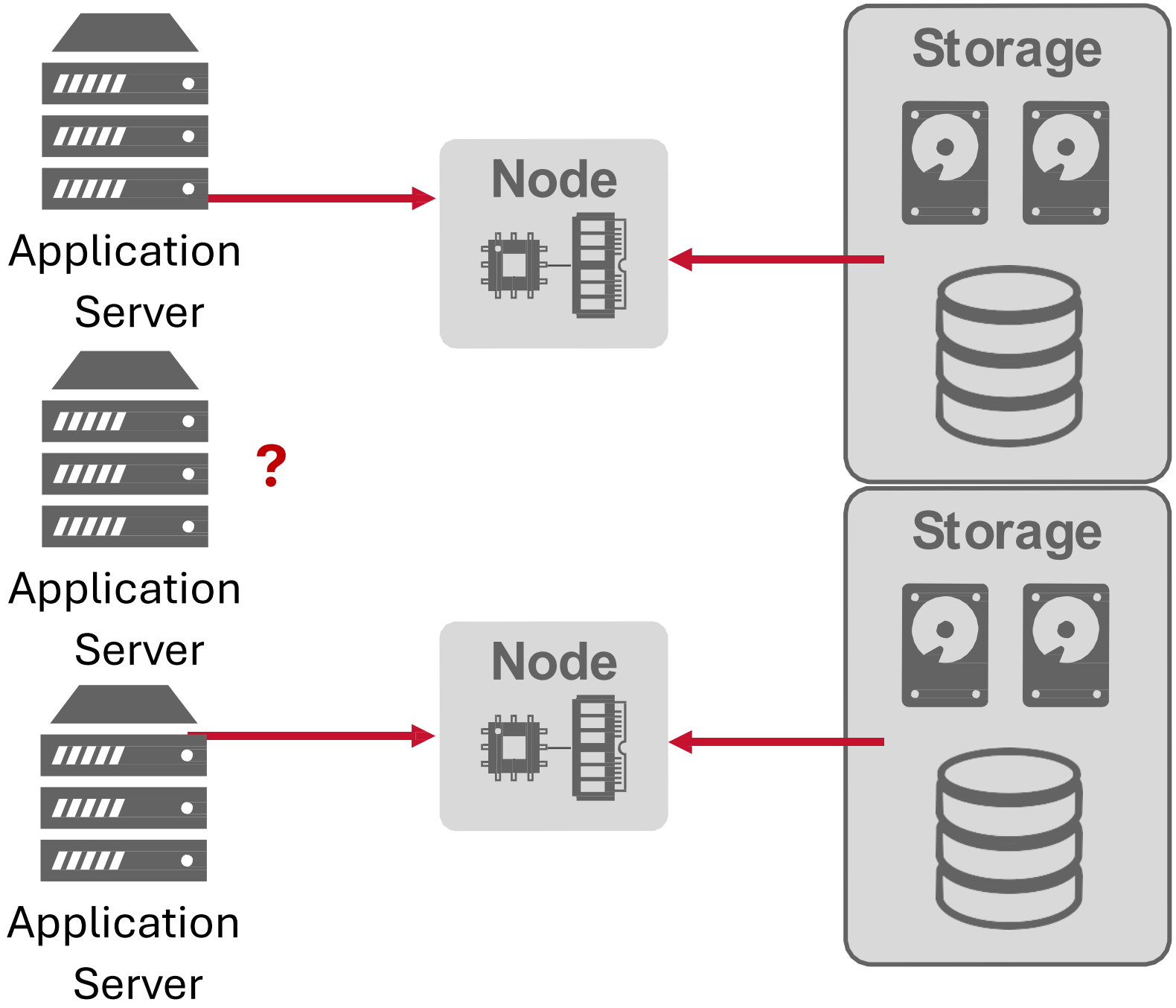
- Rather than always maintaining compute resources for each customer, a "serverless" DBMS evicts tenants when they become idle.

planetscale
CockroachDB
NEON
amazon
fauna
Microsoft SQL Azure



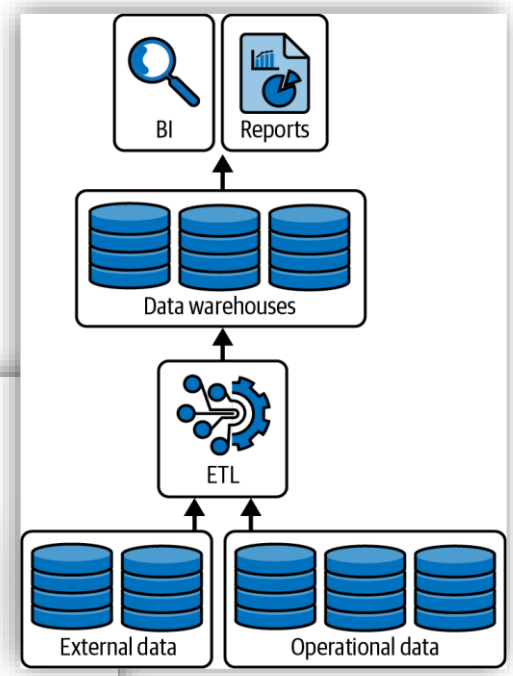
Overbooking?

- Sell more than have.



Data warehouses

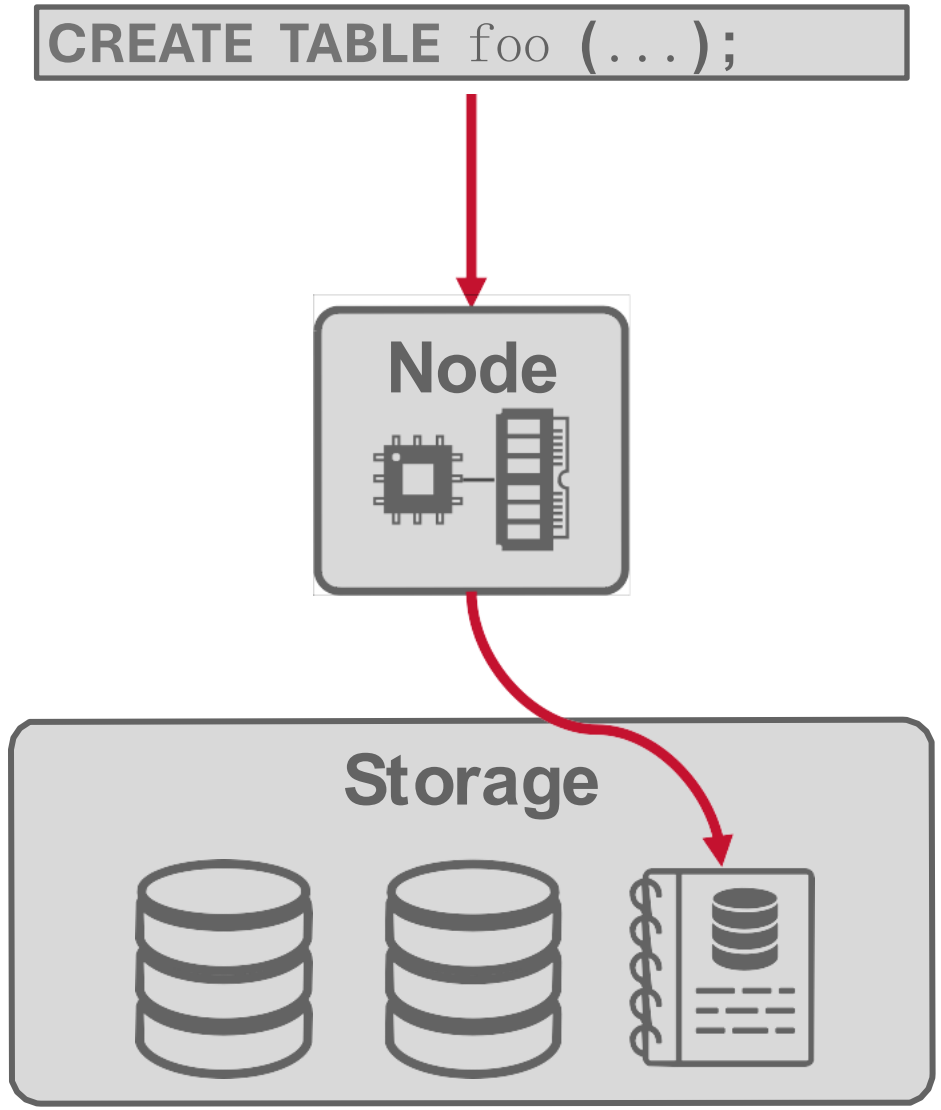
- A data management system that stores current and historical data from multiple sources in a business friendly manner for easier insights and reporting. Typically used for business intelligence (BI).
  - ACID transactions
  - Management features (backup and recovery controls, gated controls, etc.)
  - Performance optimizations (indexes, partitioning, etc.)
  - Limited support for ML and unstructured data.



https://www.future-processing.com/blog/top-8-data-warehouse-solutions/

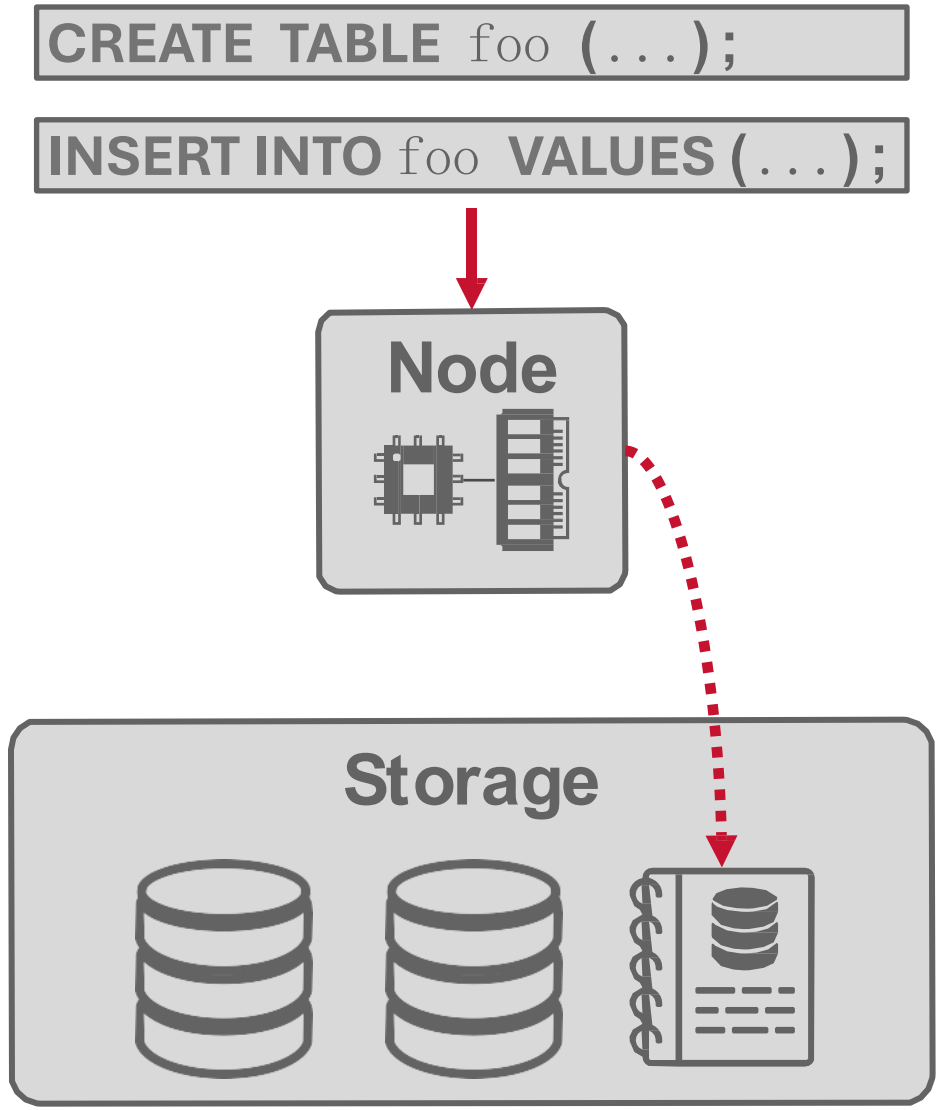
Data lakes

- Repository for storing large amounts of structured, semi-structured, and unstructured data without having to define a schema or ingest the data into proprietary internal formats.



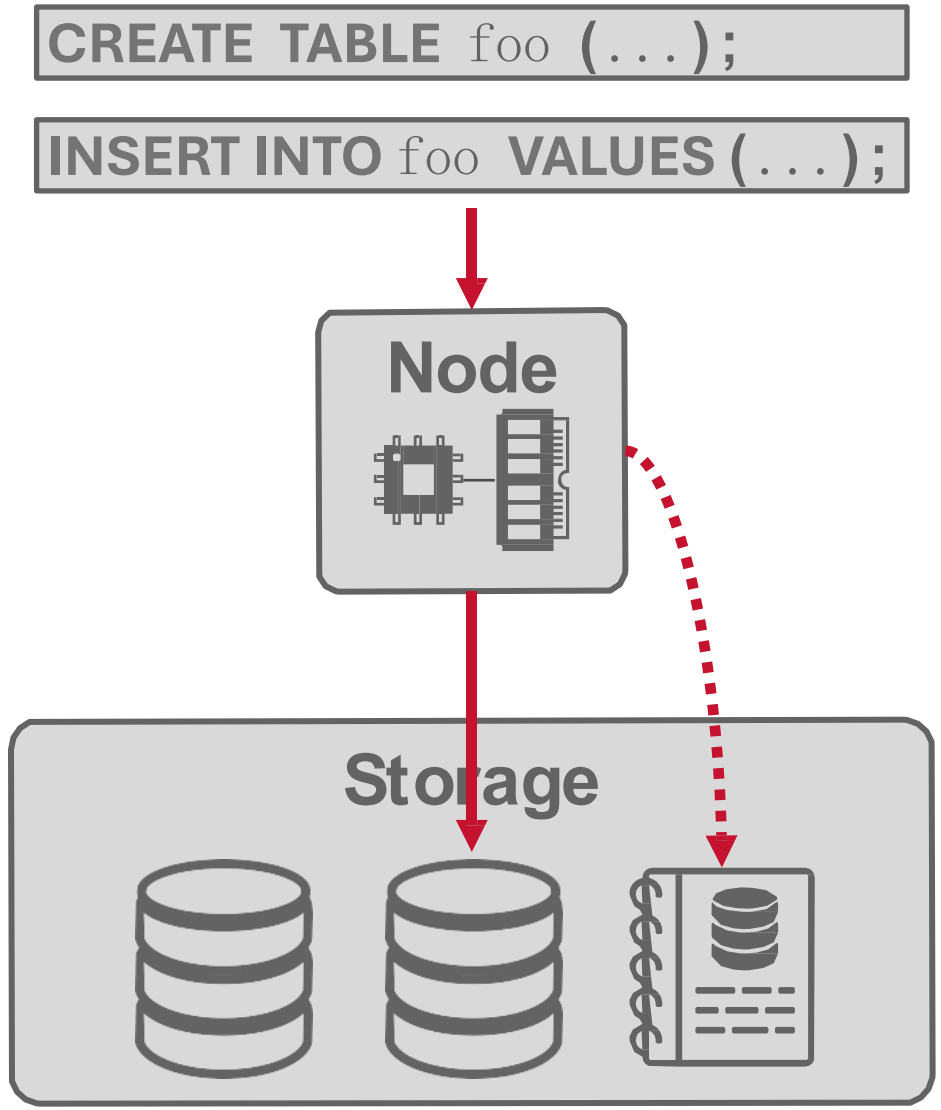
Data lakes

- Repository for storing large amounts of structured, semi-structured, and unstructured data without having to define a schema or ingest the data into proprietary internal formats.



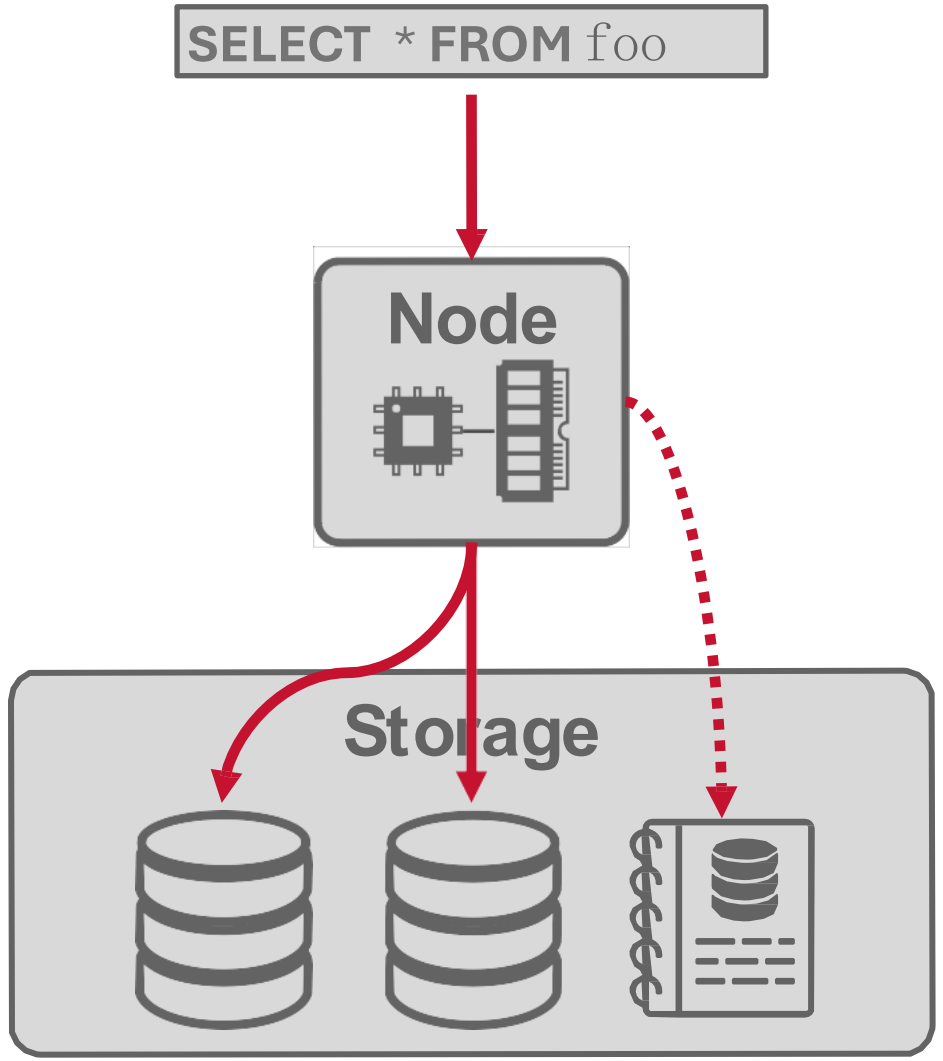
Data lakes

- Repository for storing large amounts of structured, semi-structured, and unstructured data without having to define a schema or ingest the data into proprietary internal formats.



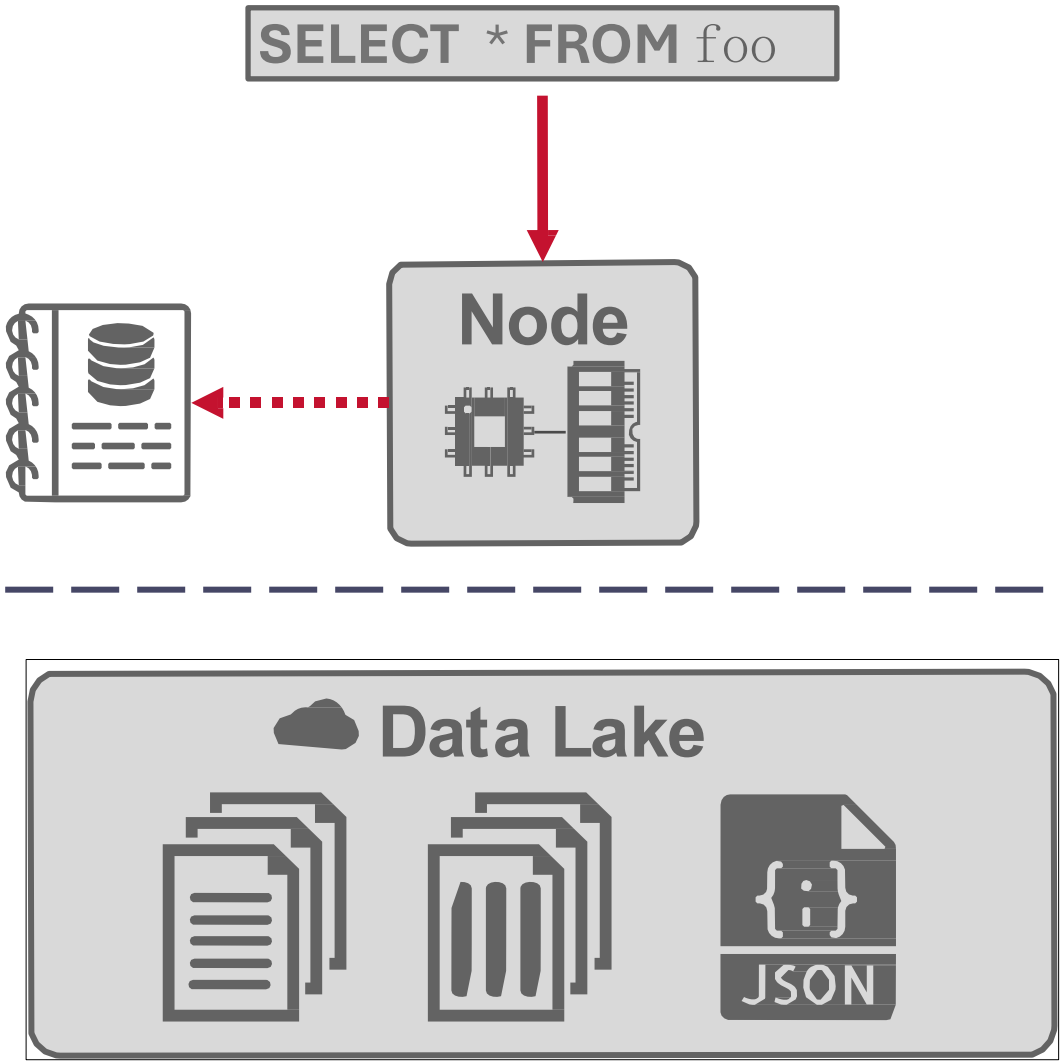
Data lakes

- Repository for storing large amounts of structured, semi-structured, and unstructured data without having to define a schema or ingest the data into proprietary internal formats.



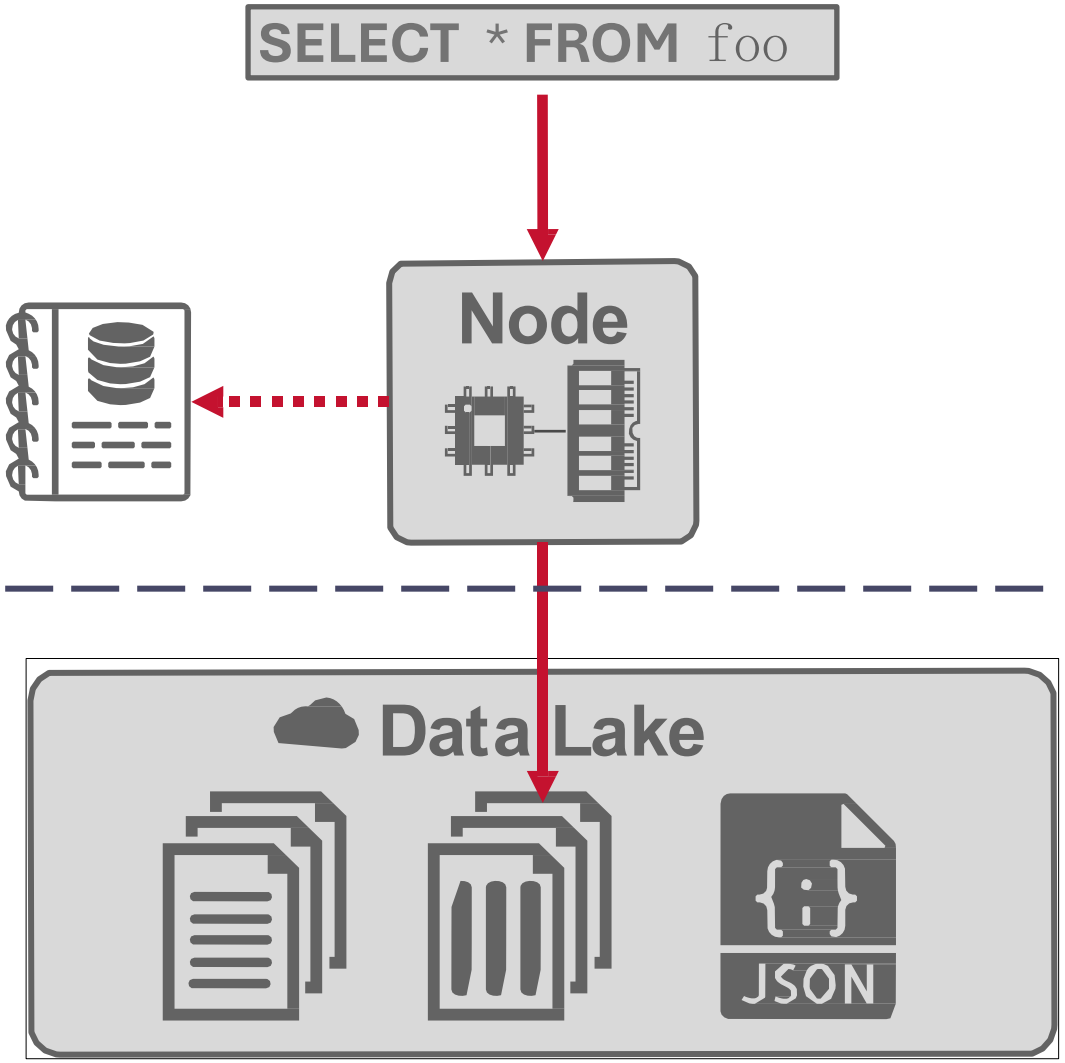
Data lakes

- Repository for storing large amounts of structured, semi-structured, and unstructured data without having to define a schema or ingest the data into proprietary internal formats.



Data lakes

- Repository for storing large amounts of structured, semi-structured, and unstructured data without having to define a schema or ingest the data into proprietary internal formats.



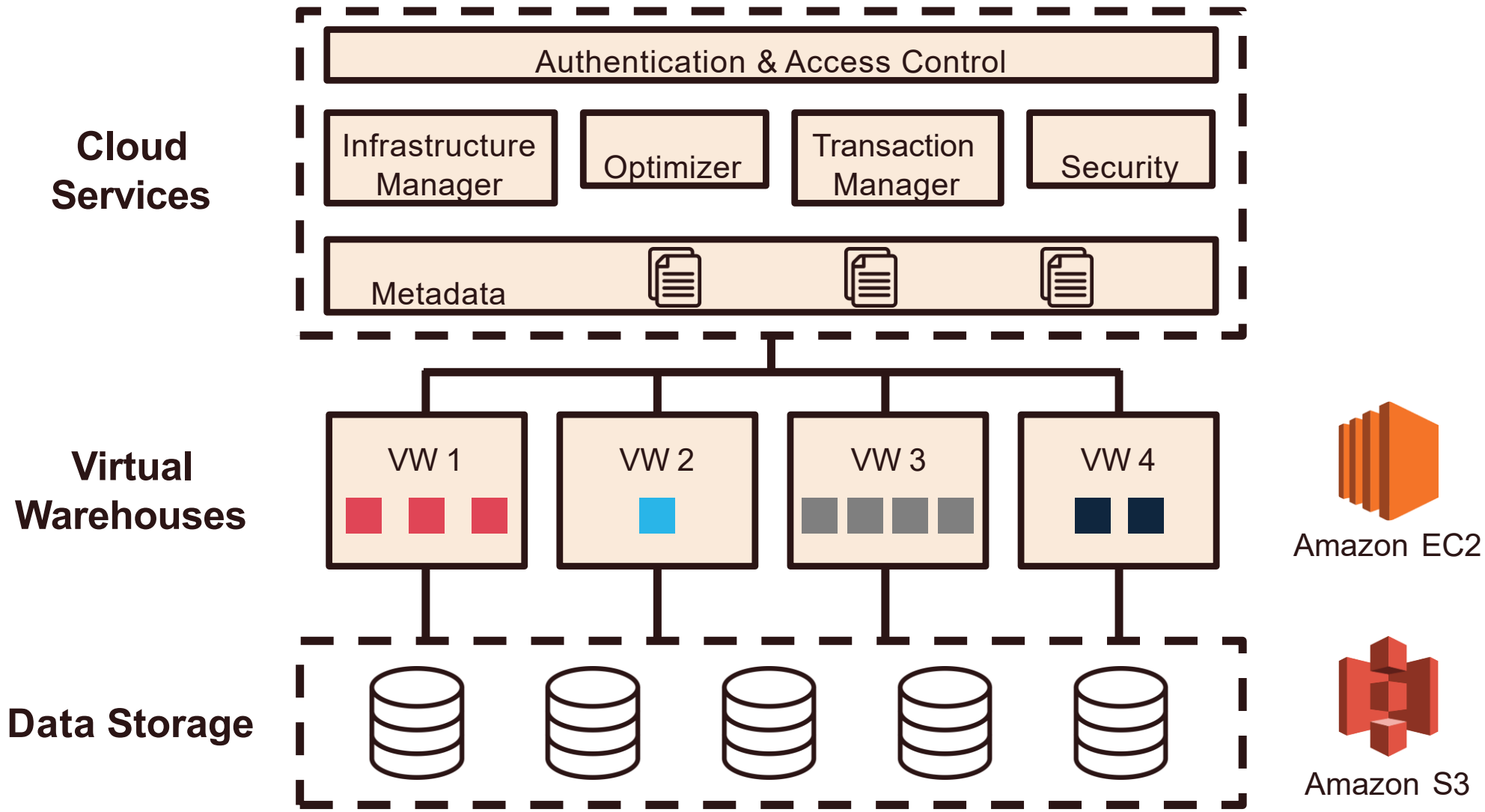
trino amazon REDSHIFT Google Big Query HIVE

databricks snowflake presto

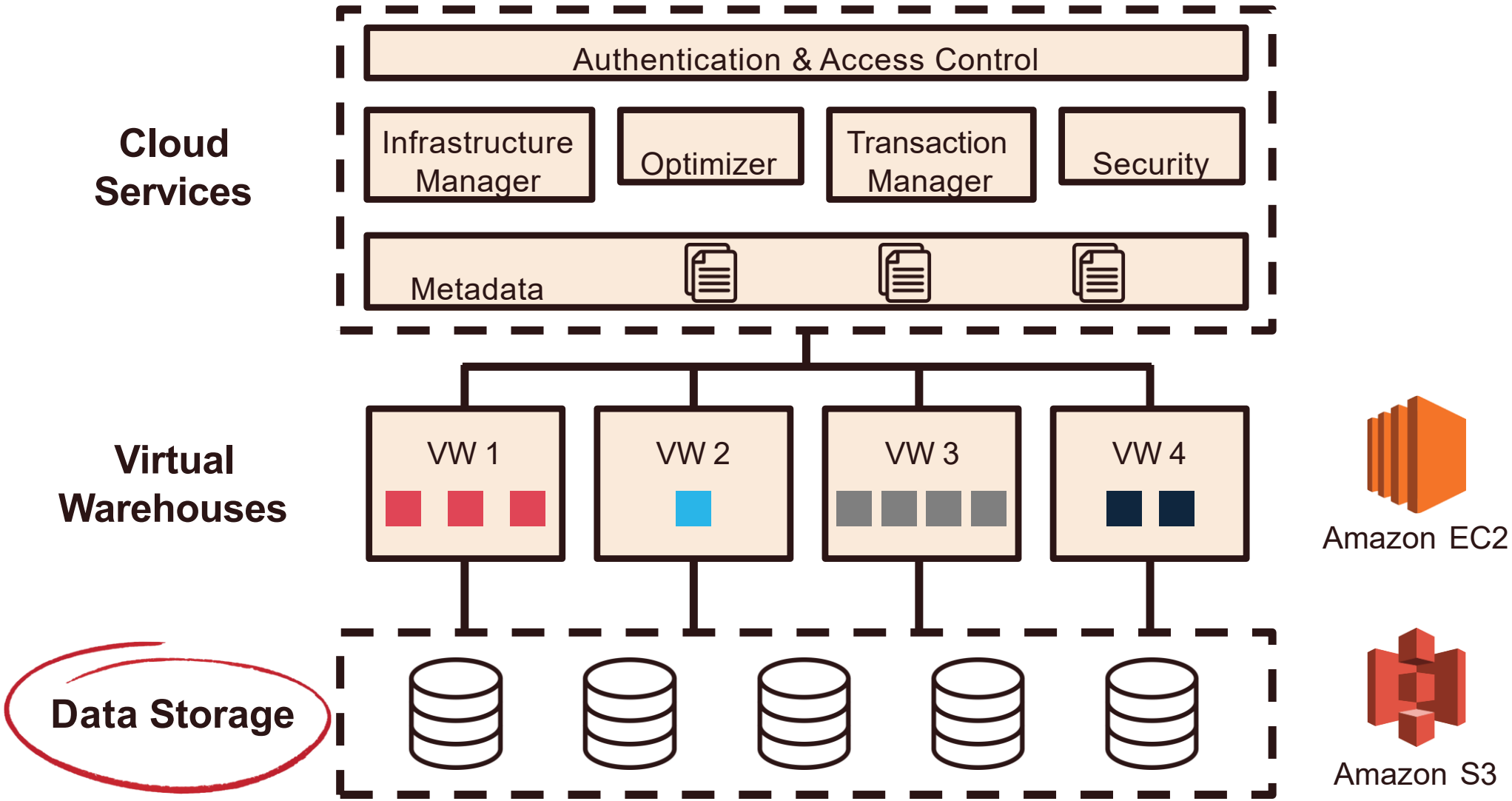
Snowflake

- The Snowflake Elastic Data Warehouse
  - Target analytical queries to support business intelligence (BI)
  - Founded at 2012, growing fast, largest software IPO (2020) ever
- Pure SaaS
  - Nothing to install, always on, always up-to-date
  - Ease of use, only pay for what you use
- Multi-Cloud Support

Snowflake architecture



Snowflake architecture



Database workloads

- Online Transaction Processing (OLTP)
  - Transactions that read/update a small amount of data each time.
  - Each transaction finishes in a short time (e.g., 1 or 0.1 milliseconds)
- Online Analytical Processing (OLAP)
  - Complex queries that read a lot of data to compute joins/aggregates.
    - A query can take up to hours or days.
  - Data is typically either unchanged or insert-only.
    - A query typically does not take locks. The DBMS uses a special way to maintain consistency if there are inserts.
- Hybrid Transaction + Analytical Processing
  - OLTP + OLAP together on the same database instance

Online Transaction Processing (OLTP)

Example: online shopping, stock market transactions, …

```
SELECT *
FROM ShoppingCart
WHERE customerID = …
```

```
INSERT INTO Orders
VALUES (…)
```

```
UPDATE Accounts
SET balance = …
WHERE customerID = …
```

Online Transaction Processing (OLTP)

Example: online shopping, stock market transactions, …

- Simple, short-lived transactions (ms)
- Only touch a small amount of data
- Insert- and update-heavy
- Few table joins
- Skewed access towards recent data
- Queries often predefined

Large number of concurrent operations

```
SELECT *
FROM ShoppingCart
WHERE customerID = …
```

```
INSERT INTO Orders
VALUES (…)
```

```
UPDATE Accounts
SET balance = …
WHERE customerID = …
```

Online Analytical Processing (OLAP)

Example: data analytics, business report, ...

```
with v1 as(
  select i_category, i_brand, cc_name, d_year, d_moy,
         sum(cs_sales_price) sum_sales,
         avg(sum(cs_sales_price)) over
           (partition by i_category, i_brand,
                      cc_name, d_year)
           avg_monthly_sales,
         rank() over
           (partition by i_category, i_brand,
                      cc_name
            order by d_year, d_moy) rn
  from item, catalog_sales, date_dim, call_center
  where cs_item_sk = i_item_sk and
        cs_sold_date_sk = d_date_sk and
        cc_call_center_sk= cs_call_center_sk and
        (
          d_year = 1999 or
          ( d_year = 1999-1 and d_moy =12) or
          ( d_year = 1999+1 and d_moy =1)
        )
  group by i_category, i_brand,
           cc_name , d_year, d_moy),
v2 as(
  select v1.i_category ,v1.d_year, v1.d_moy ,v1.avg_monthly_sales
        ,v1.sum_sales, v1_lag.sum_sales psum, v1_lead.sum_sales nsum
  from v1, v1 v1_lag, v1 v1_lead
  where v1.i_category = v1_lag.i_category and
        v1.i_category = v1_lead.i_category and
        v1.i_brand = v1_lag.i_brand and
        v1.i_brand = v1_lead.i_brand and
        v1.cc_name = v1_lag.cc_name and
        v1.cc_name = v1_lead.cc_name and
        v1.rn = v1_lag.rn + 1 and
        v1.rn = v1_lead.rn - 1)
select  *
from v2
where  d_year = 1999 and
         avg_monthly_sales > 0 and
         case when avg_monthly_sales > 0 then abs(sum_sales - avg_monthly_sales) / avg_mont
order by sum_sales - avg_monthly_sales, 3
limit 100;
```

Online Analytical Processing (OLAP)

Example: data analytics, business report, ...

- Complex, long-running aggregations
- Large table scans
- Mostly reads with periodic batch inserts
- Often joins multiple tables
- Historical data
- Queries often ad hoc

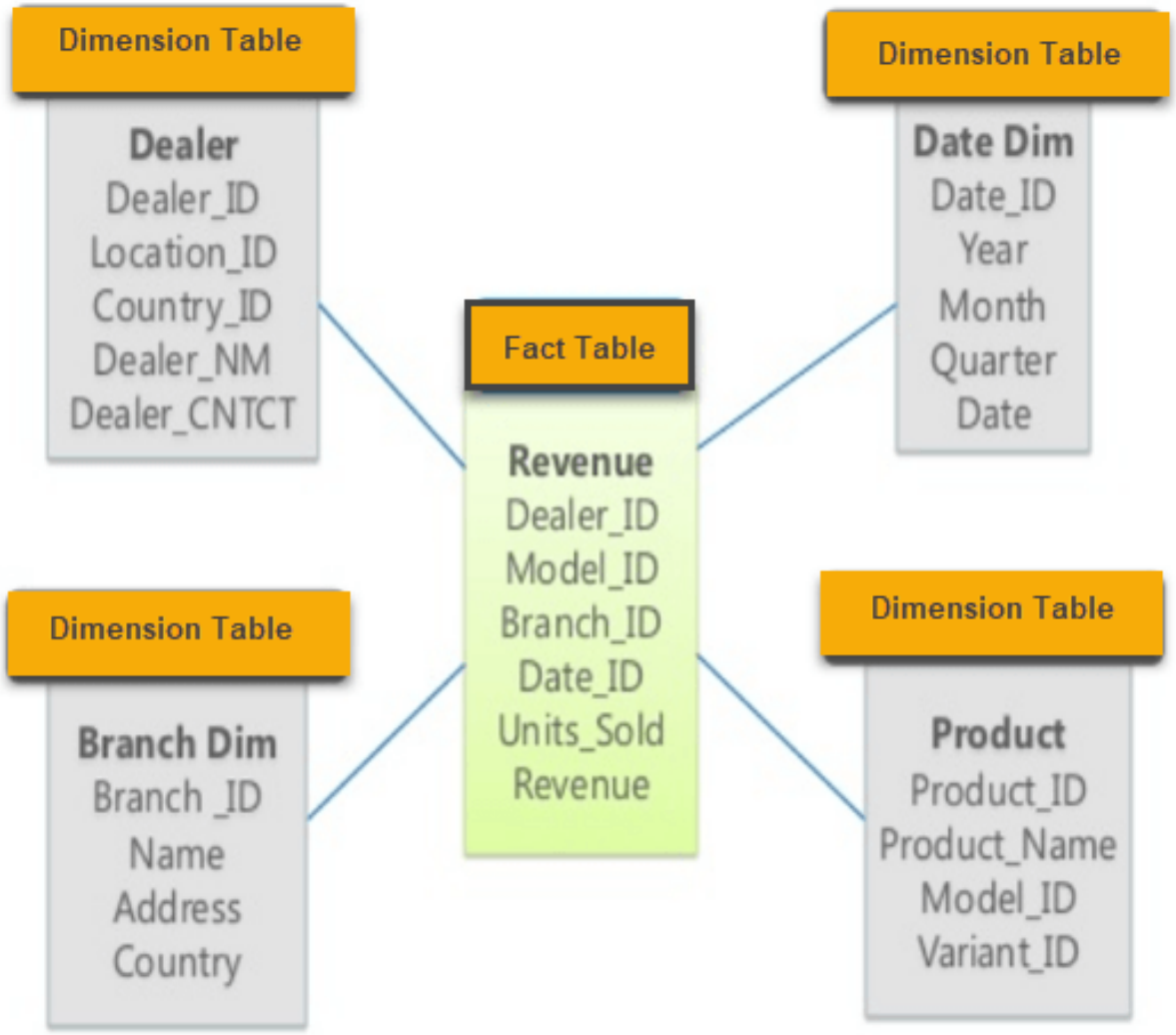
Heavy compute on large volume of data

```
with v1 as(
  select i_category, i_brand, cc_name, d_year, d_moy,
         sum(cs_sales_price) sum_sales,
         avg(sum(cs_sales_price)) over
           (partition by i_category, i_brand,
                      cc_name, d_year)
           avg_monthly_sales,
         rank() over
           (partition by i_category, i_brand,
                      cc_name
            order by d_year, d_moy) rn
  from item, catalog_sales, date_dim, call_center
  where cs_item_sk = i_item_sk and
        cs_sold_date_sk = d_date_sk and
        cc_call_center_sk= cs_call_center_sk and
        (
          d_year = 1999 or
          ( d_year = 1999-1 and d_moy =12) or
          ( d_year = 1999+1 and d_moy =1)
        )
  group by i_category, i_brand,
           cc_name , d_year, d_moy),
v2 as(
  select v1.i_category ,v1.d_year, v1.d_moy ,v1.avg_monthly_sales
         ,v1.sum_sales, v1_lag.sum_sales psum, v1_lead.sum_sales nsum
  from v1, v1 v1_lag, v1 v1_lead
  where v1.i_category = v1_lag.i_category and
        v1.i_category = v1_lead.i_category and
        v1.i_brand = v1_lag.i_brand and
        v1.i_brand = v1_lead.i_brand and
        v1.cc_name = v1_lag.cc_name and
        v1.cc_name = v1_lead.cc_name and
        v1.rn = v1_lag.rn + 1 and
        v1.rn = v1_lead.rn - 1)
select  *
from v2
where  d_year = 1999 and
         avg_monthly_sales > 0 and
         case when avg_monthly_sales > 0 then abs(sum_sales - avg_monthly_sales) / avg_mont
order by sum_sales - avg_monthly_sales, 3
limit 100;
```

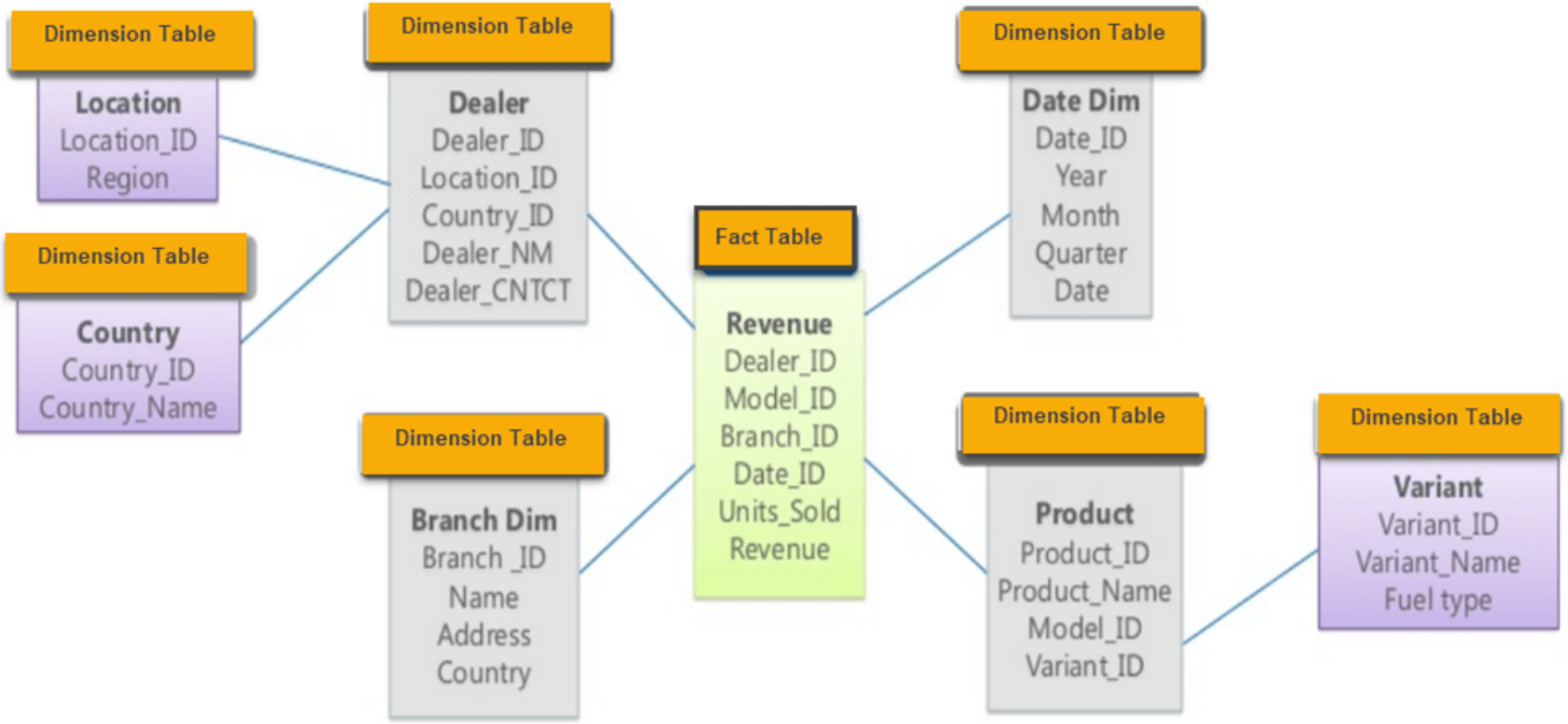
OLAP Schema

- An OLAP database is typically composed of fact tables and dimension tables.
- Fact tables record information about individual events, such as sales, and are usually very large.
  - Example: sales information for a retail store, with one tuple for each item that is sold.
- Dimension tables includes the attributes for describing the data in the fact table.
  - Example: Time and location for each item that is sold
- Fact tables and dimension tables are connected via foreign keys.

Star schema


Dimension Table
Dealer
Dealer_ID
Location_ID
Country_ID
Dealer_NM
Dealer_CNTCT
Dimension Table
Date Dim
Date_ID
Year
Month
Quarter
Date
Fact Table
Revenue
Dealer_ID
Model_ID
Branch_ID
Date_ID
Units_Sold
Revenue
Dimension Table
Branch Dim
Branch _ID
Name
Address
Country
Dimension Table
Product
Product_ID
Product_Name
Model_ID
Variant_ID

Snowflake schema



Observation

- The way we store data will significantly impact the performance of processing queries.

- The relational model does not specify that the DBMS must store all of a tuple's attributes together on a single page.

- This may not be the best layout for OLAP workloads…

Row-store

Table is 2D, but storage is 1D array

id	name	age	balance
105	Alice	18	1000
102	Bob	25	2000
104	Charlie	18	3000
101	David	18	1500
103	Emily	20	2500

Row-store

Table is 2D, but storage is 1D array

id	name	age	balance
105	Alice	18	1000
102	Bob	25	2000
104	Charlie	18	3000
101	David	18	1500
103	Emily	20	2500

Pros **Ideal for OLTP**

- Fast tuple insertion/deletion
- Fast SELECT *

Cons

- **SELECT avg**(balance) **FROM** T **GROUP BY** age
- Reading useless data: wasting I/O

Tuple-by-tuple Storage

105 Alice 18 1000	102 Bob 25 2000	104 Charlie 18 3000	104 David 18 1500	103 Emily 20 2500

Column-store

Table is 2D, but storage is 1D array

id	name	age	balance
105	Alice	18	1000
102	Bob	25	2000
104	Charlie	18	3000
101	David	18	1500
103	Emily	20	2500

Column-by-column Storage

105 102 104 101 103	Alice Bob Charlie David Emily	18 25 18 18 20	1000 2000 3000 1500 2500

id **name** **age** **balance**

Column-store

Table is 2D, but storage is 1D array

id	name	age	balance
105	Alice	18	1000
102	Bob	25	2000
104	Charlie	18	3000
101	David	18	1500
103	Emily	20	2500

Pros **Ideal for OLAP**

- Only scan relevant attributes
- Fast and efficient query processing

Cons

- **INSERT INTO** T **VALUES** (a, b, c, d, …)
- **SELECT * …**
- Extra work in tuple splitting and stitching

Column-by-column Storage

105 102 104 101 103	Alice Bob Charlie David Emily	18 25 18 18 20	1000 2000 3000 1500 2500

id **name** **age** **balance**

Column-store is everywhere

- Becomes popular in the late 2000s
  - Vertica (C-Store), MonetDB, VectorWise
- Every major data warehouse today
  - Teradata, Amazon Redshift, Snowflake, Google BigQuery, ClickHouse, Greenplum …
- Traditional row-stores intending to support OLAP-type queries
  - Oracle 12c, SQL Server, IBM DB2 BLU

Hybrid columnar format

Tables are horizontally portioned into files, where each file is stored in columnar format

Pure Columnar

File 1

name
Alice
David
Bob
Emily
Bob
Emily
Bob
Charlie
Emily

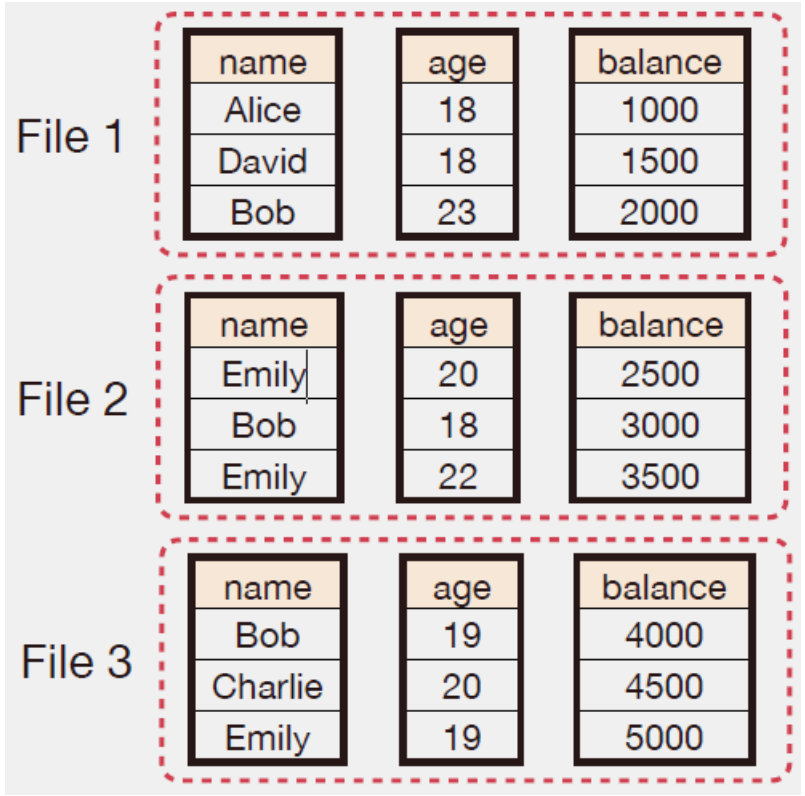
File 2

age
18
18
23
20
18
22
19
20
19

File 3

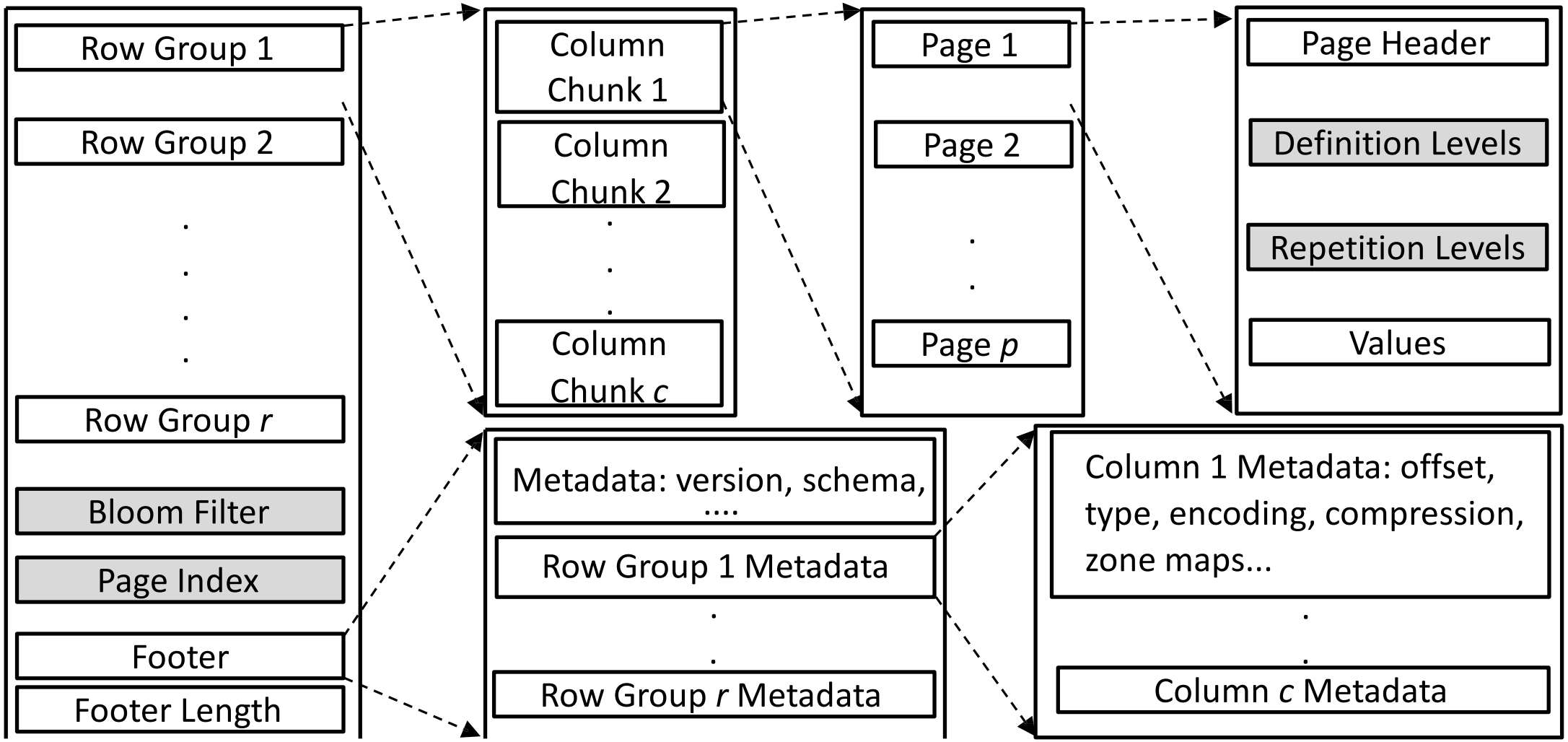
balance
1000
1500
2000
2500
3000
3500
4000
4500
5000

Hybrid Columnar (aka., PAX)



A disk page in the original proposal

Apache Parquet



Columnar compression

- Dictionary Encoding
- Run-length Encoding
- Bit-Packing Encoding
- Bitmap Encoding
- Delta Encoding
- Learned encoding

(The later schemes go to my other cloud DB course)

Naïve Compression

Compress data using a general-purpose algorithm. The scope of compression is only based on the data provided as input.

→ LZO (1996), LZ4 (2011), Snappy (2011), Oracle OZIP (2014), Zstd (2015)

Considerations

→ Computational overhead
→ Compress vs. decompress speed.

Naïve compression

The DBMS must decompress data first before it can be read and (potentially) modified.

→ Repeated compression and decompression will be the bottleneck.

These schemes also do not consider the high-level meaning or semantics of the data.

Dictionary encoding

Replace frequent values with smaller fixed-length codes and then maintain a mapping (dictionary) from the codes to the original values

→ Typically, one code per attribute value.

→ Most widely used native compression scheme in DBMSs.

The ideal dictionary scheme supports fast encoding and decoding for both point and range queries.

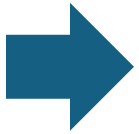
Dictionary encoding example

```
SELECT * FROM users
 WHERE name = 'Andy'
```

```
SELECT * FROM users
 WHERE name = 30
```

Original Data

name
Andrea
Prashanth
Andy
Matt
Prashanth



Compressed Data

name
10
20
30
40
20

value	*code*
Andrea	10
Prashanth	20
Andy	30
Matt	40

Dictionary

Dictionary encoding: order preserving

The encoded values need to support the same collation as the original values.

```
SELECT * FROM users
 WHERE name LIKE 'And%'
```

```
SELECT * FROM users
 WHERE name BETWEEN 10 AND 20
```

Original Data

name
Andrea
Prashanth
Andy
Matt
Prashanth

Compressed Data

name
10
40
20
30
40

value	*code*
Andrea	10
Andy	20
Matt	30
Prashanth	40

Sorted Dictionary

Dictionary: order preserving

```
SELECT name FROM users
 WHERE name LIKE 'And%'
```

Still must perform scan on column

```
SELECT DISTINCT name
  FROM users
 WHERE name LIKE 'And%'
```

Only need to access dictionary

Original Data

name
Andrea
Prashanth
Andy
Matt
Prashanth

Compressed Data

name
10
40
20
30
40

value	*code*
Andrea	10
Andy	20
Matt	30
Prashanth	40

Sorted Dictionary

Run-length encoding

Compress the same continuous values in a single column into triplets:

→ The value of the attribute.

→ The start position in the column segment.

→ The # of elements in the run.

Requires the columns to be sorted intelligently to maximize compression opportunities.

Run-length encoding

Original Data

id	isDead
1	Y
2	Y
3	Y
4	N
6	Y
7	N
8	Y
9	Y

Compressed Data

id	isDead
1	(Y,0,3)
2	(N,3,1)
3	(Y,4,1)
4	(N,5,1)
6	(Y,6,2)
7	
8	
9	

RLE Triplet
- **Value**
- **Offset**
- **Length**

Run-length encoding

Sorted Data

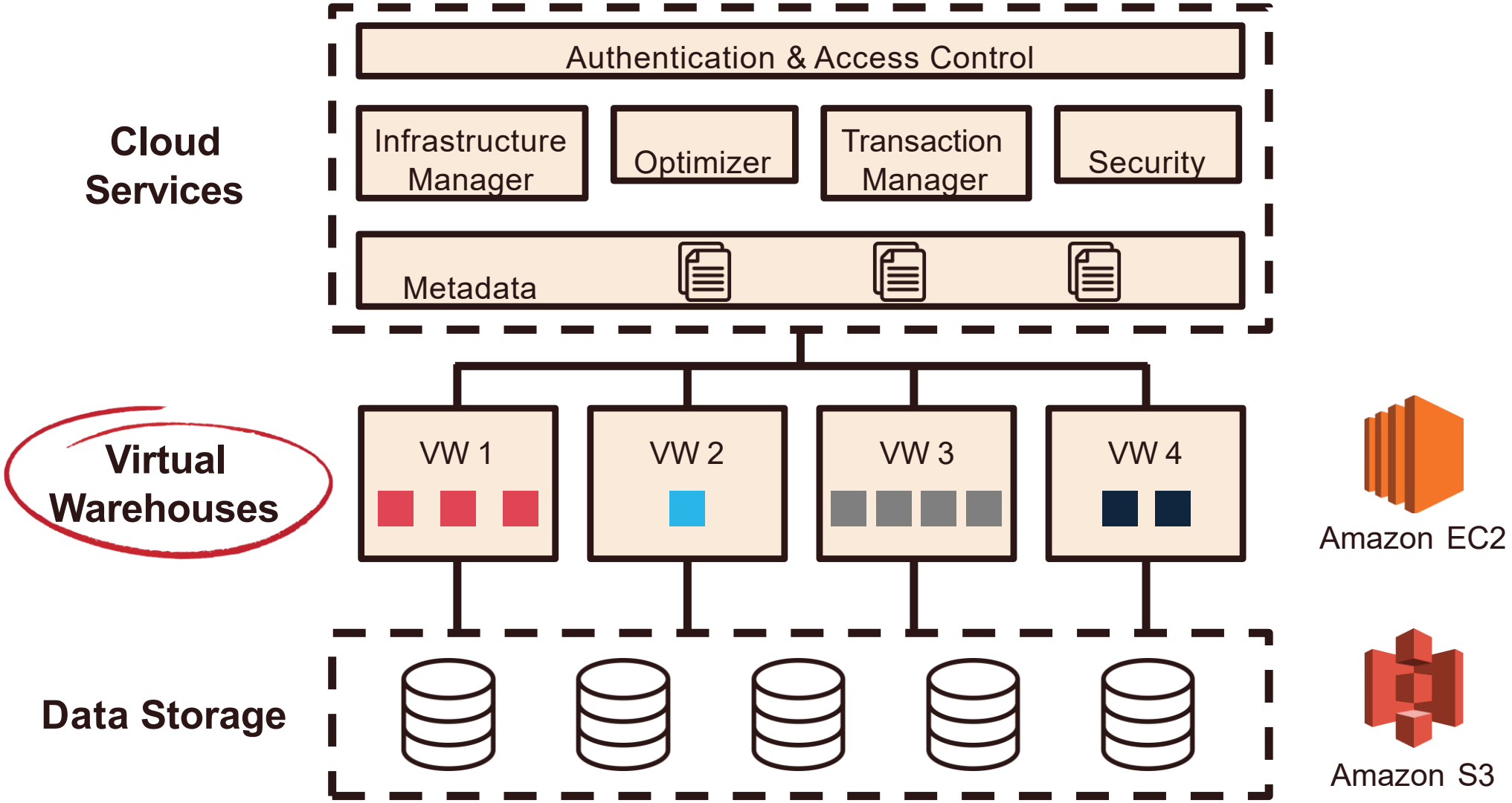
id	isDead
1	Y
2	Y
3	Y
6	Y
8	Y
9	Y
4	N
7	N

Compressed Data

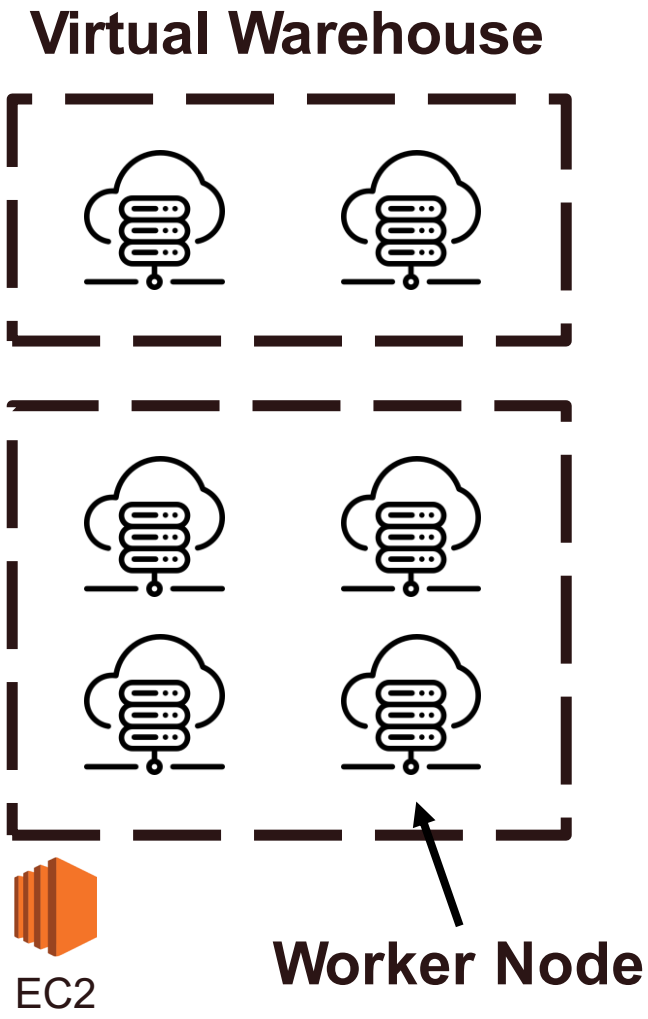
id	isDead
1	(Y,0,6)
2	(N,7,2)
3	
6	
8	
9	
4	
7	

RLE Triplet
- **Value**
- **Offset**
- **Length**

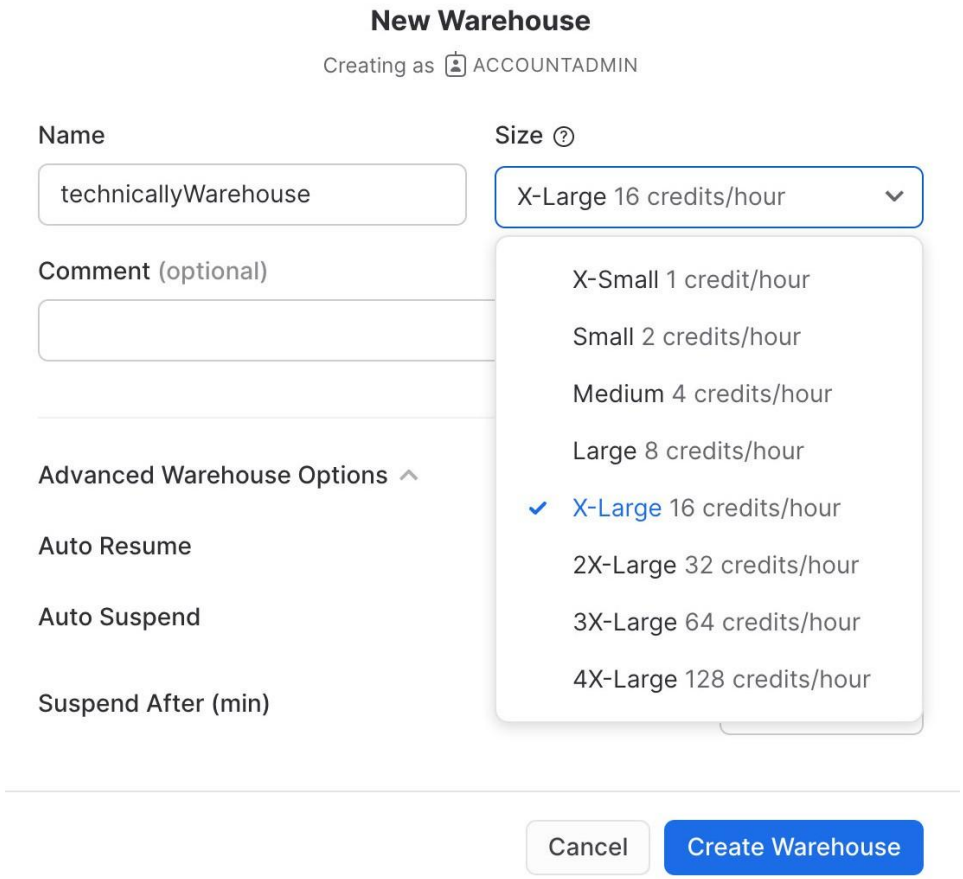
Snowflake architecture



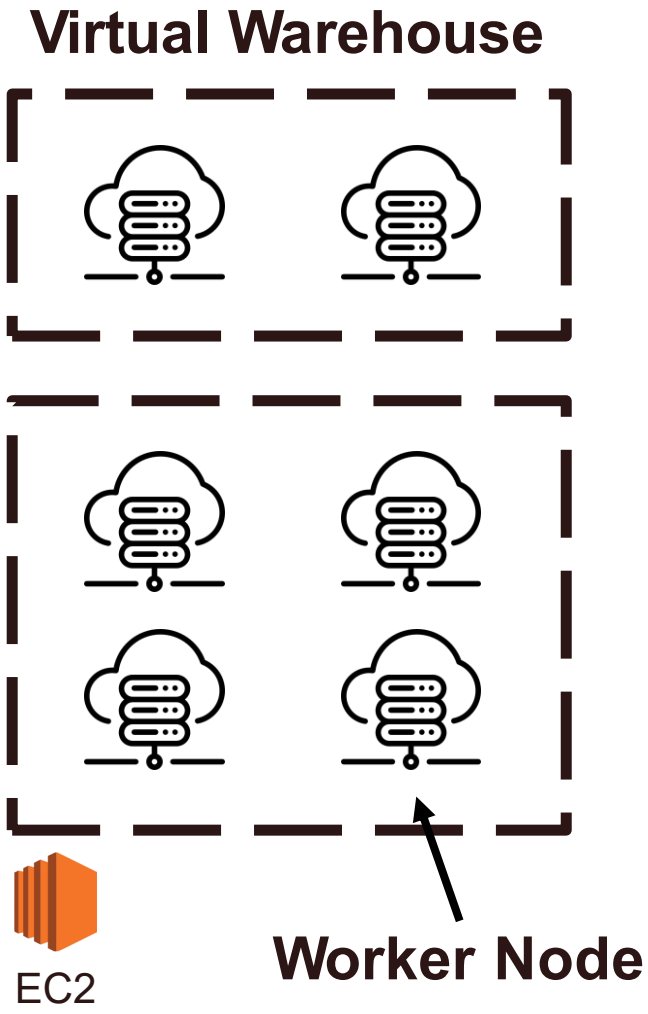
Virtual warehouses: the muscle



- Create, destroy, resize on demand

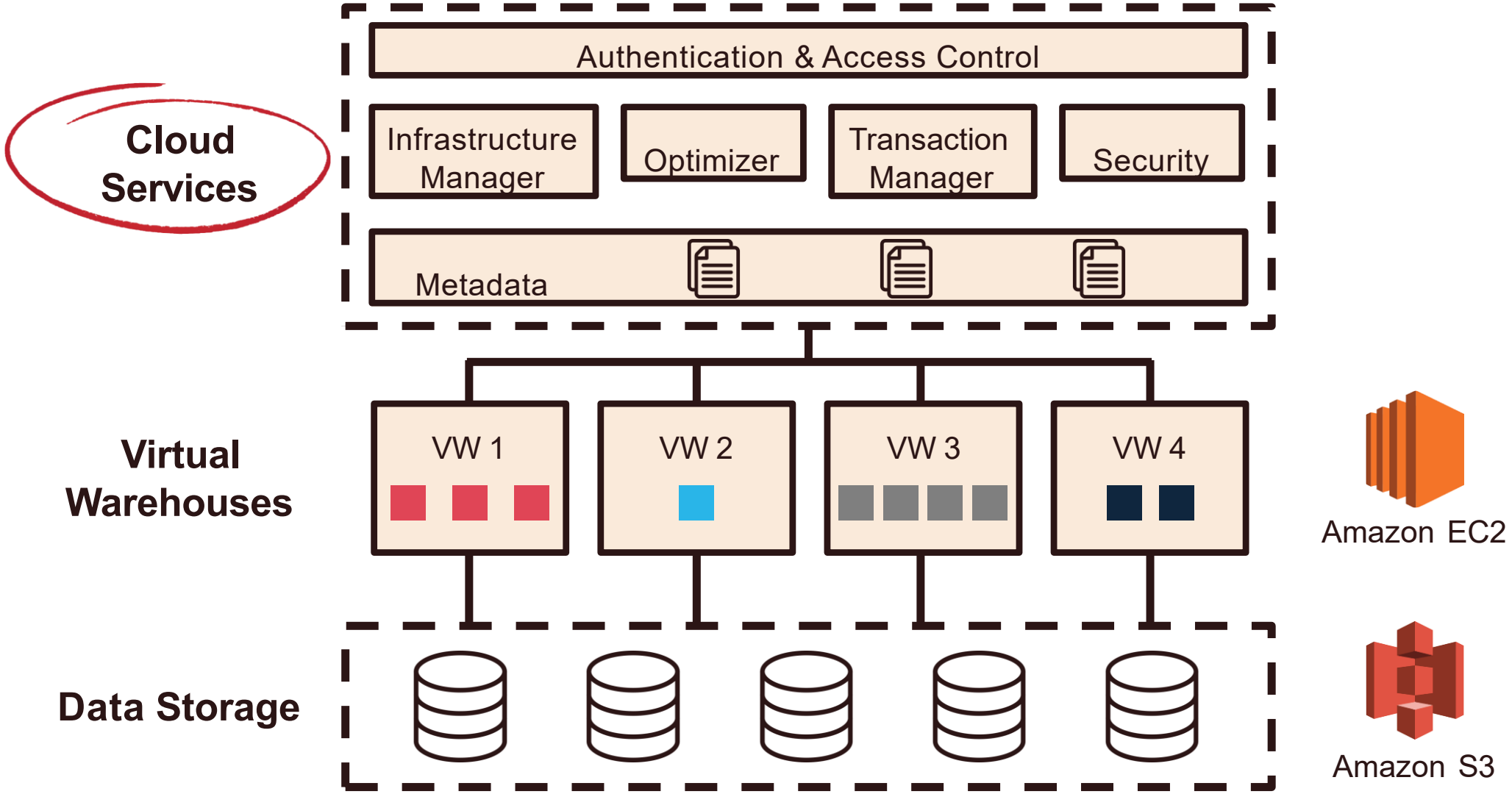


Virtual warehouses: the muscle

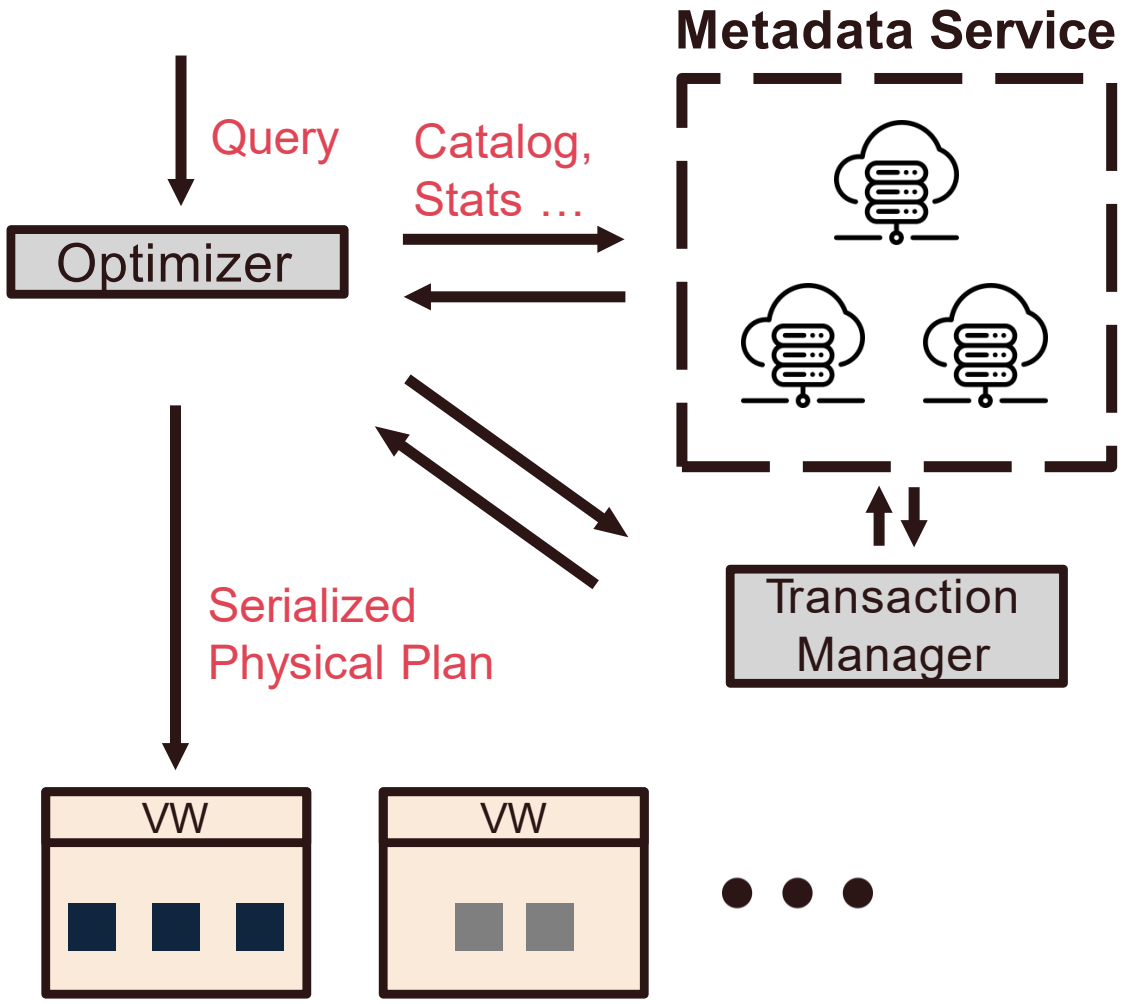


- Create, destroy, resize on demand
- Performance Isolation
  - Shared data, private compute
  - Typical usage pattern
    - Continuously-running VWs for repeating jobs
    - On-demand VWs for ad-hoc tasks
- Ephemeral worker processes
- **Columnar, Vectorized, Push-Based**

Snowflake architecture



Cloud services: the brain



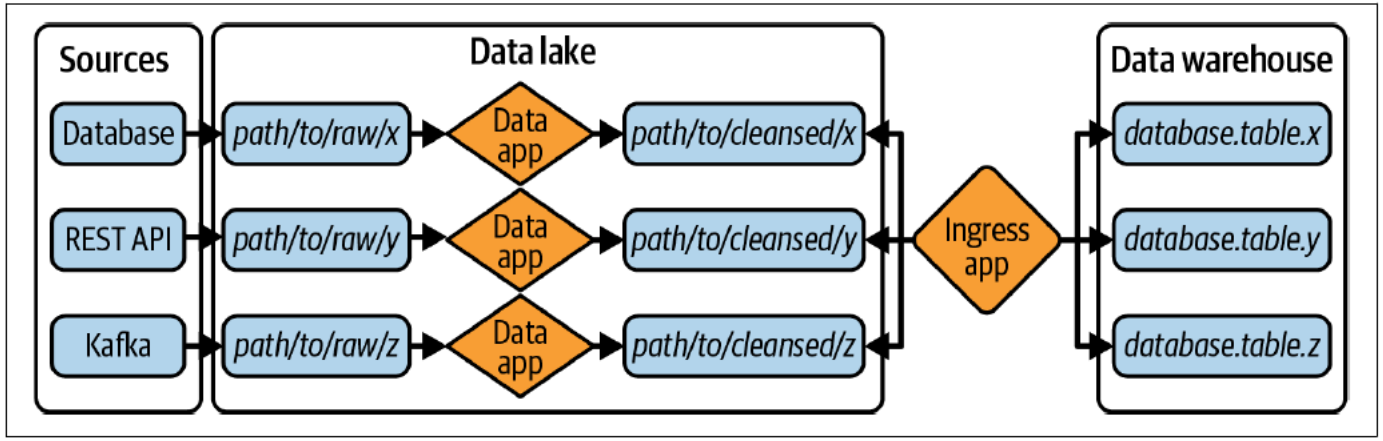
- **Optimizer**
  - Cascade-style
  - Scan set **pruning**
- **Metadata Service**
  - Stand-alone **FoundationDB** cluster for low latency accesses
  - Info needed for query compilation
    - Catalog, Stats
    - Lock status, version info
    - Zone maps
- **Multi-Version Concurrency Control (Snapshot Isolation)**

Snowflake architecture summary

- Disaggregated compute and storage
- Immutable hybrid columnar files in object storage
- Virtual warehouses provide elasticity and performance isolation
- Vectorized push-based execution engine
- Ephemeral storage system for caching intermediate results and persistent files
- Multi-tenant, always-on cloud services
- Separate fast metadata store
- Cascades-style optimizer, zone maps for scan pruning

The dual-tier data architecture

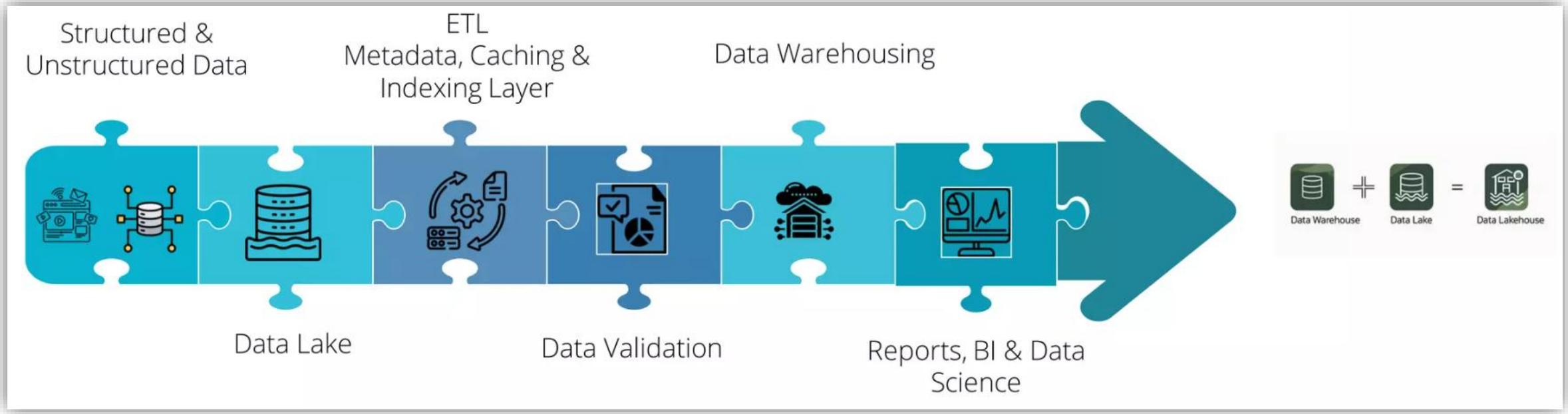
- Repository for storing large amounts of structured, semi-structured, and unstructured data without having to define a schema or ingest the data into proprietary internal formats.
    - Extract operational data from siloed data sources for writing into landing zones (/raw).
    - Read, clean, and transform the data from /raw and write the changes to /cleansed.
    - Read from /cleansed (could do additional joining and normalization) before writing out the warehouse.



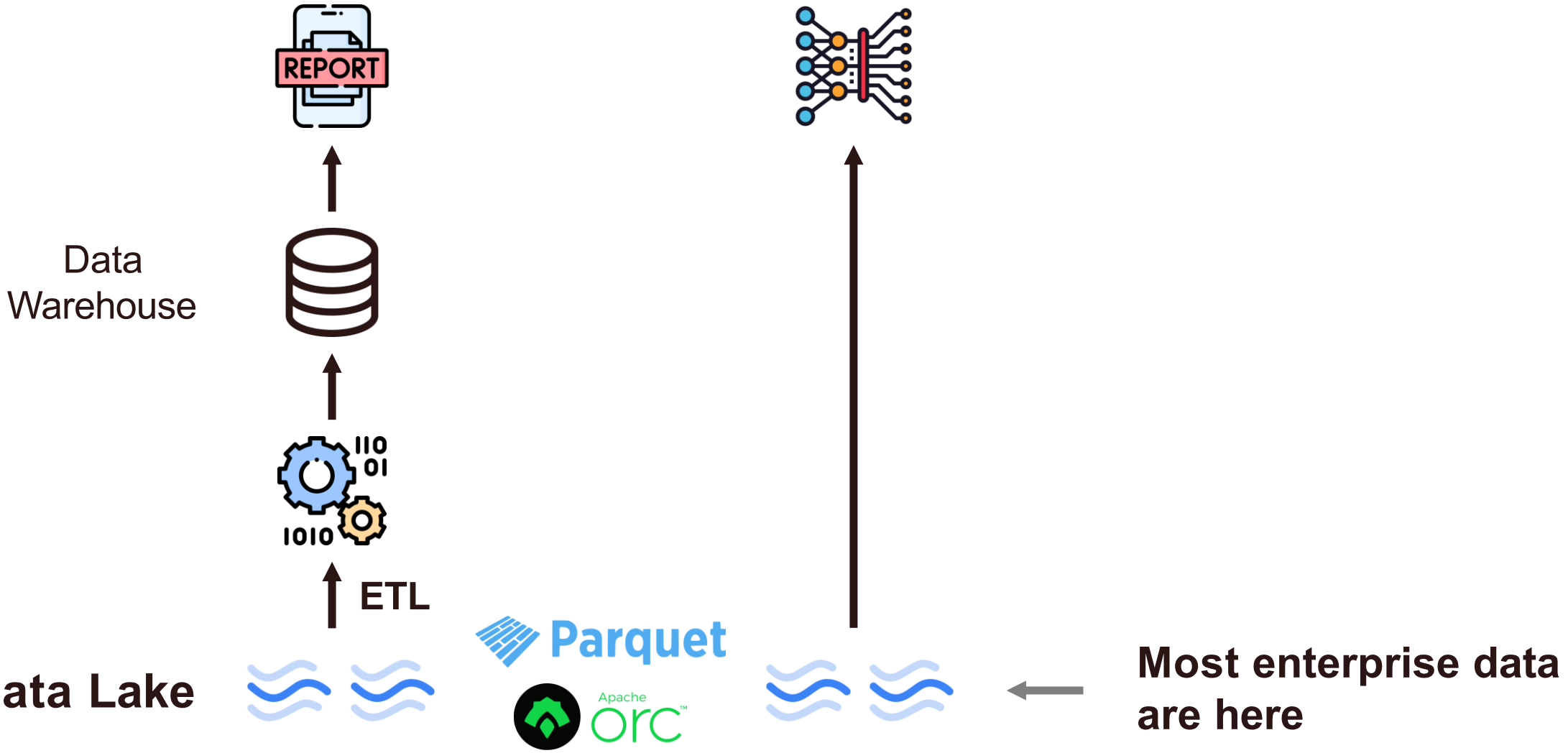
- Complex staging, redundant storage and less efficient

Lakehouse

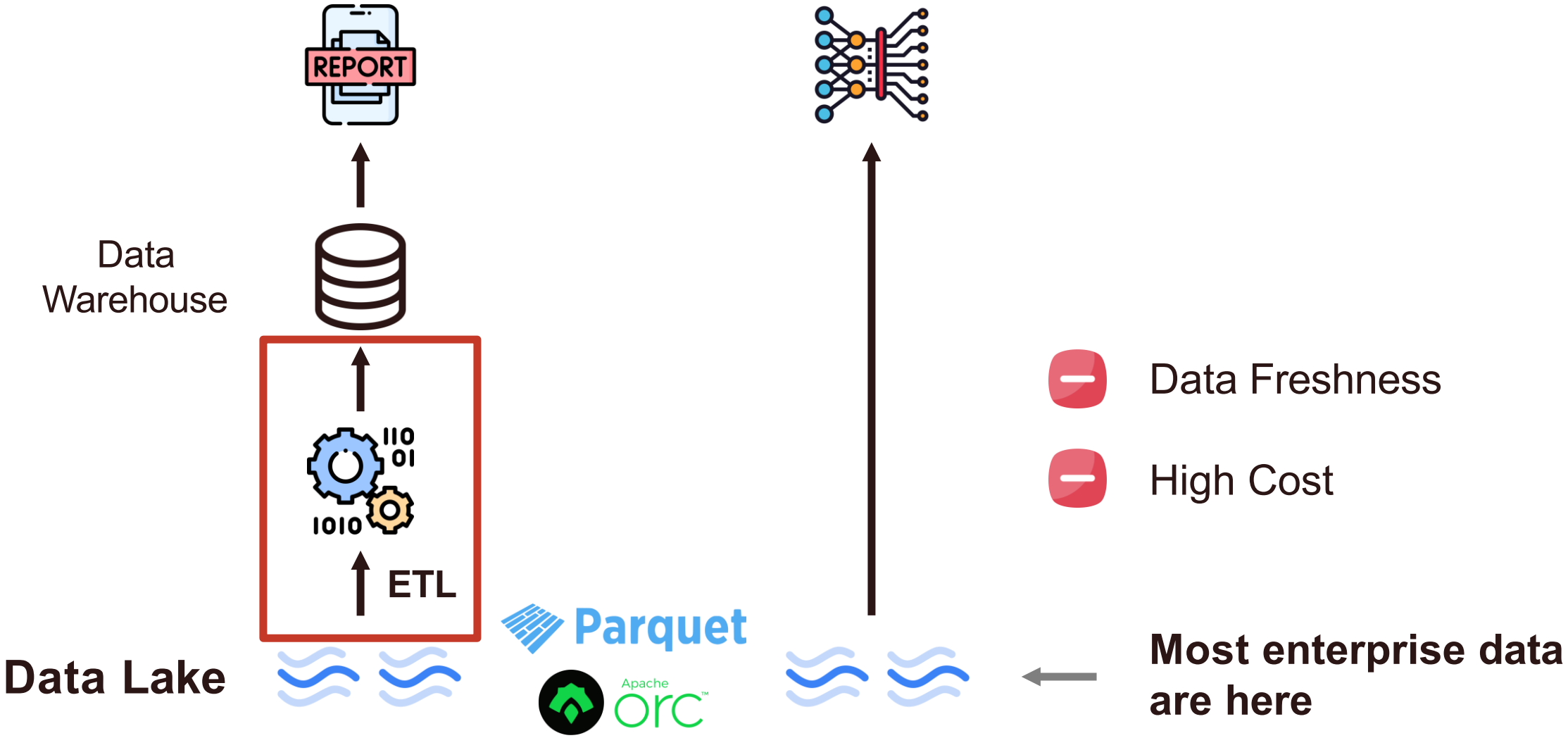
- A combination of data warehouse and data lake for better flexibility, low cost, and ACID transactions.
  - No need to copy data to data lake and warehouse separately.
  - Saves cost of infrastructure and staff.
  - Scalability and resilience.



Lakehouse

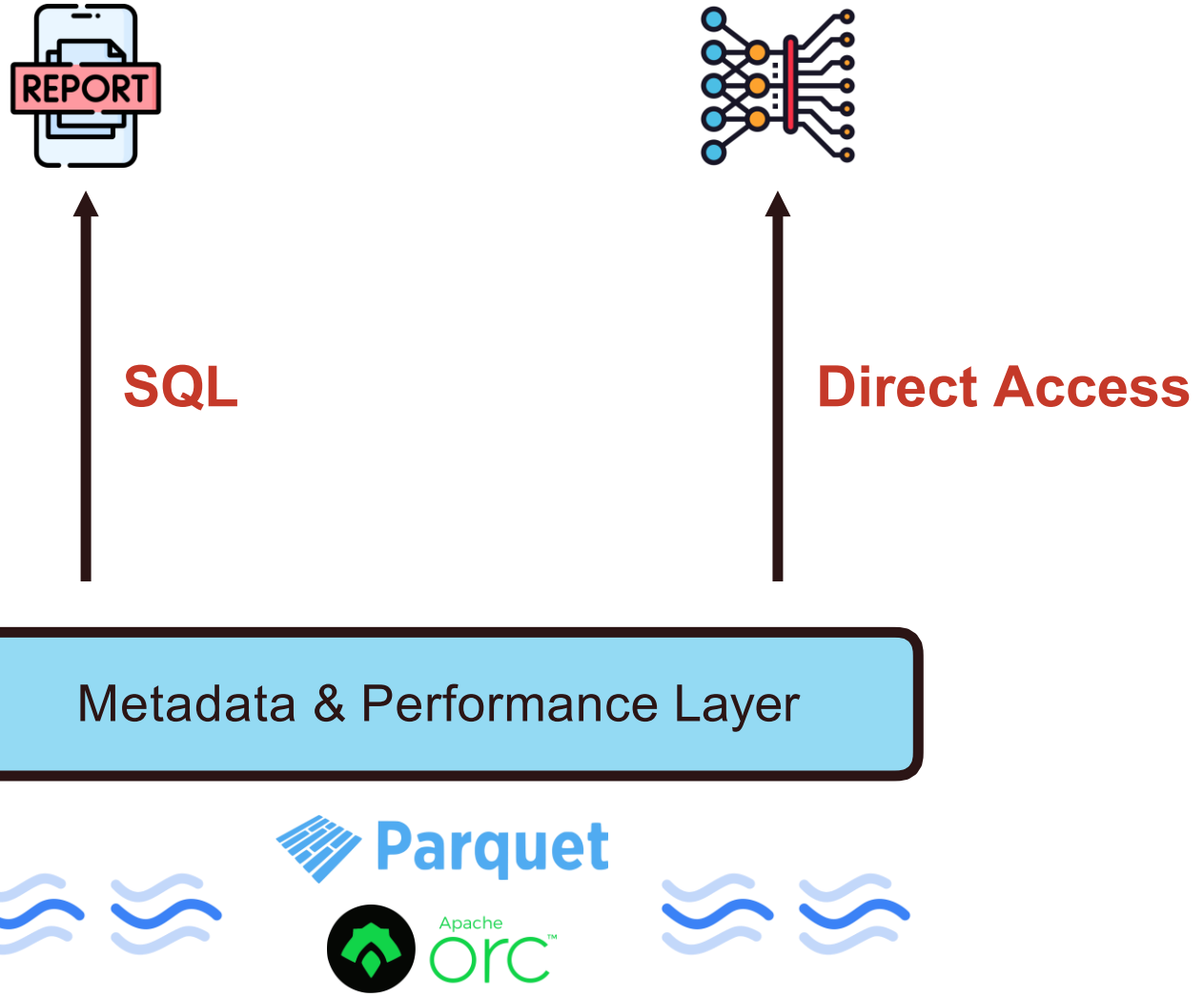

REPORT
Data Warehouse
ETL
Data Lake
Parquet
Apache orc
Most enterprise data are here

Lakehouse


REPORT
Data Warehouse
ETL
Data Lake
Parquet
Apache orc
Data Freshness
High Cost
Most enterprise data are here

Lakehouse

databricks

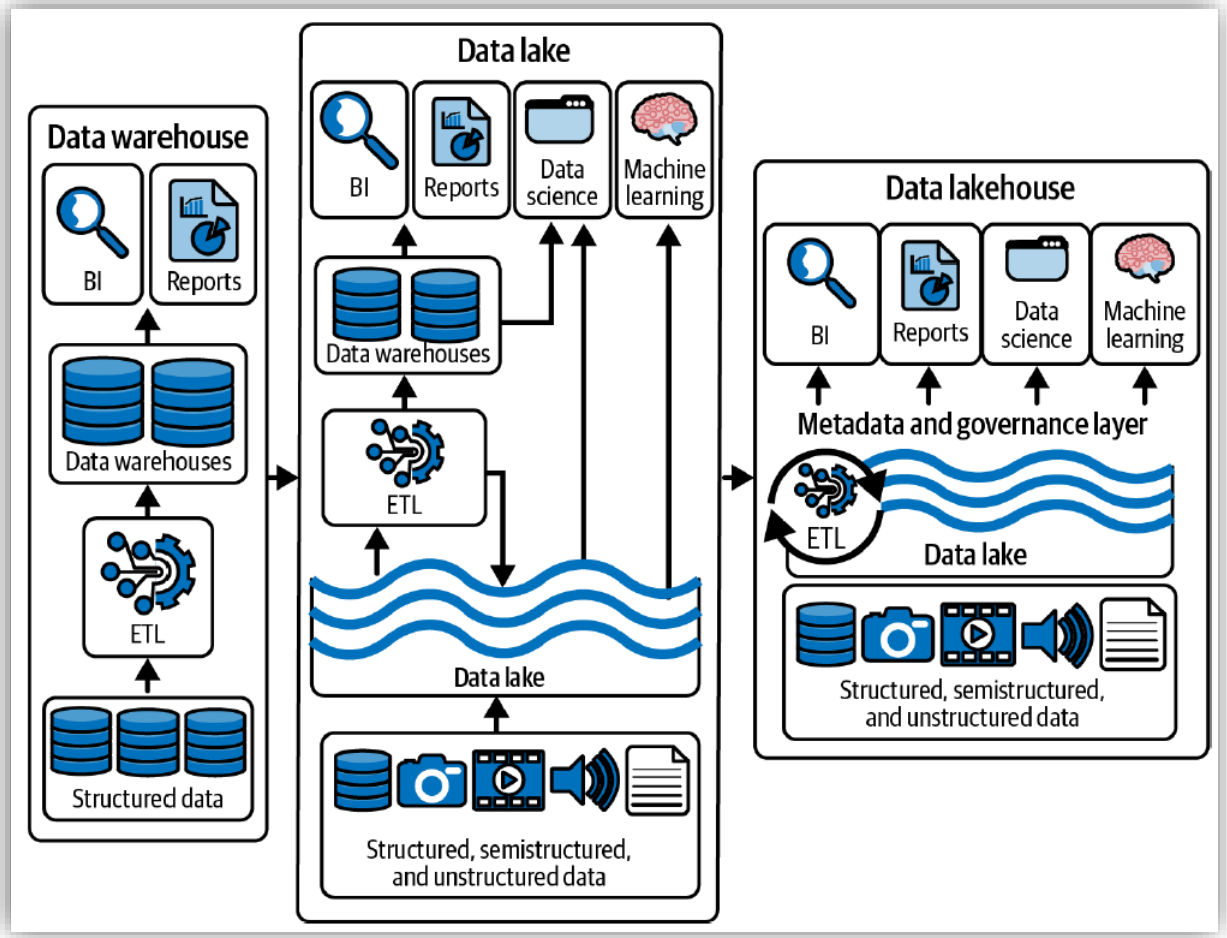


Lakehouse performance optimization

- Zone-maps, indexes, … stored as Delta tables
- Caching hot data in SSD or DRAM
- New vectorized engine: **Photon**
  - Pull-based vectorized query processing
  - Precompiled operator primitives
  - Use position list rather than bitmap for late materialization

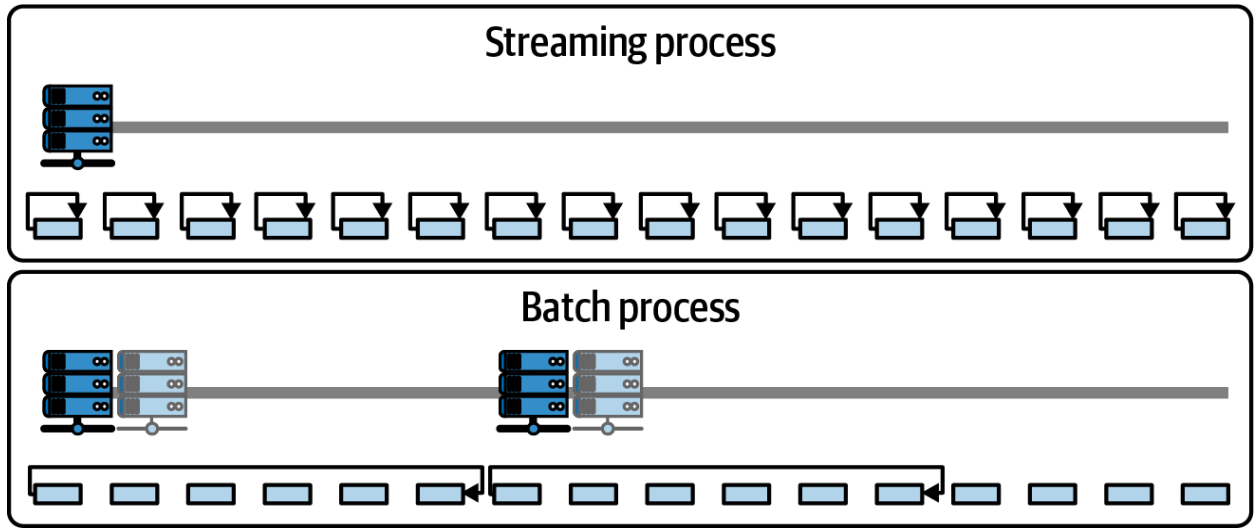
Delta Lake

- A combination of data warehouse and data lake for better flexibility, low cost, and ACID transactions.
  - No need to copy data to data lake and warehouse separately.
  - Saves cost of infrastructure and staff.
  - Scalability and resilience.



Streaming vs. batch processing

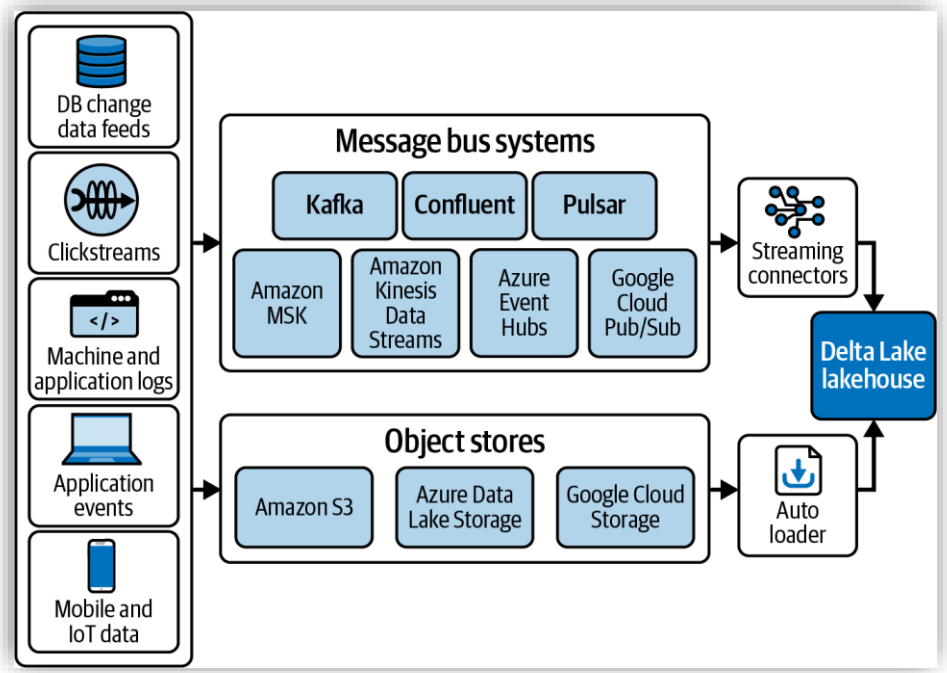
- **Streaming processing**: continuously processes data streaming, enabling instant insights and actions.
- **Batch processing**: deals with large volumes of data in chunks at scheduled intervals.



Streaming processing optimizes for **latency**, while batch processing optimizes for **throughput**.

Streaming vs. batch processing

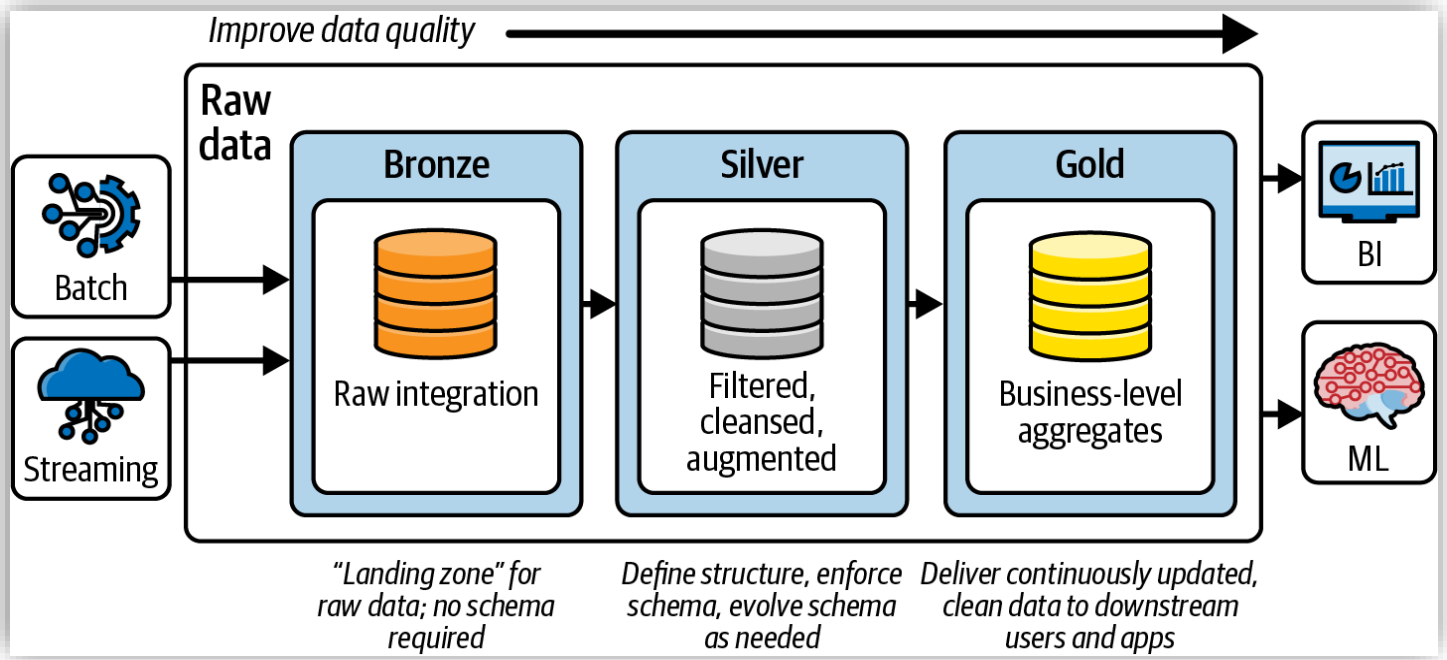
- **Streaming processing**: continuously processes data streaming, enabling instant insights and actions.
- **Batch processing**: deals with large volumes of data in chunks at scheduled intervals.



An example architecture diagram for stream processing applications with a Delta Lake sink from Databricks.

Medallion architecture

- A scheme to progressively refine datasets in the lakehouse.
  - Works for both batch or streaming sources.
  - Bronze: as simple as possible. E.g., Json parsing.
  - Silver: more complex preprocessing. E.g., text extraction from HTMLs.
  - Gold: complex joins and aggregates, w/ external data.



Take home messages

- Cloud = scale, reuse & save costs
- Familiarize your data!
- Analytics is a different, whole new world with AI
- AI coding = ops + ops + ops (data, ml, dev)

Credits and references

- Denny Lee et al. Delta Lake: The Definitive Guide.
- Andy Pavlo, CMU
- Dixin Tang, UT Austin
- Huanchen Zhang, Tsinghua
- Jianguo Wang, Purdue
- Silu Huang, ByteDance