Fundamentals of Data Engineering

Scenario

Every stakeholder interacts with the data ecosystem differently: Business executives want results and trends. Data scientists need structured data to build models. Developers need usable interfaces (APIs).Customers want value and privacy. You, the data engineer, ensure the whole data flow works efficiently and ethically.

Data Engineer

Wasting time & resources!

System Requirements

Functional Requirements	Nonfunctional Requirements

Technology Choices

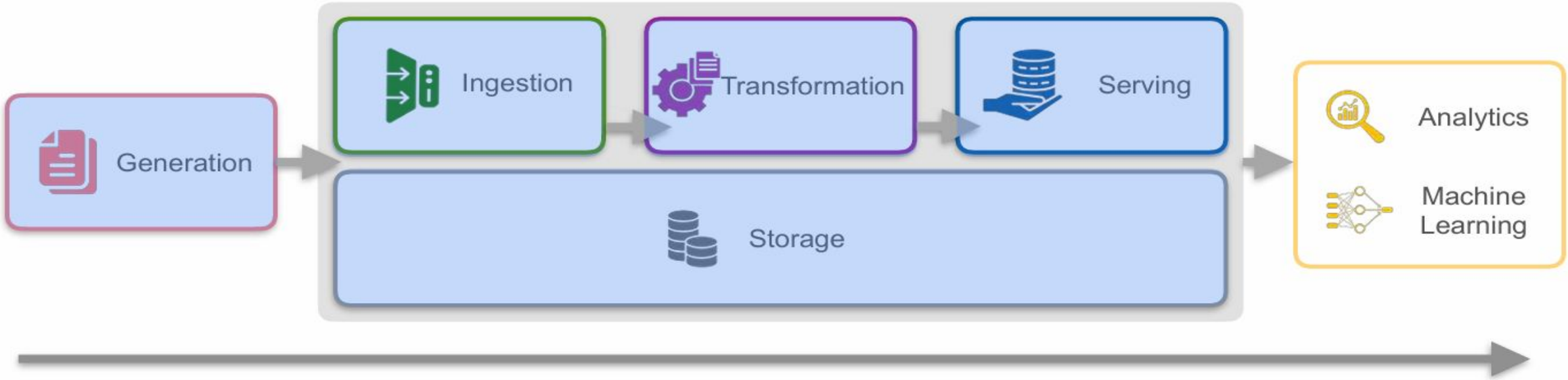
- Ingestion tool
- Storage system
- Transformation tool

Module-4

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

1. Storage Systems,
2. Cloud Storage Options: Block, Object and File storage,
3. Storage Tiers, Distributed Storage Systems,
4. Row vs Column Storage,
5. Graph Databases,
6. Vector Databases,
7. Neo4j Graph Database,
8. Storage Abstractions,
9. Data Warehouse,
10. Modern Cloud Data Warehouses,
11. Data Lakes, Date Lakehouse Implementations,

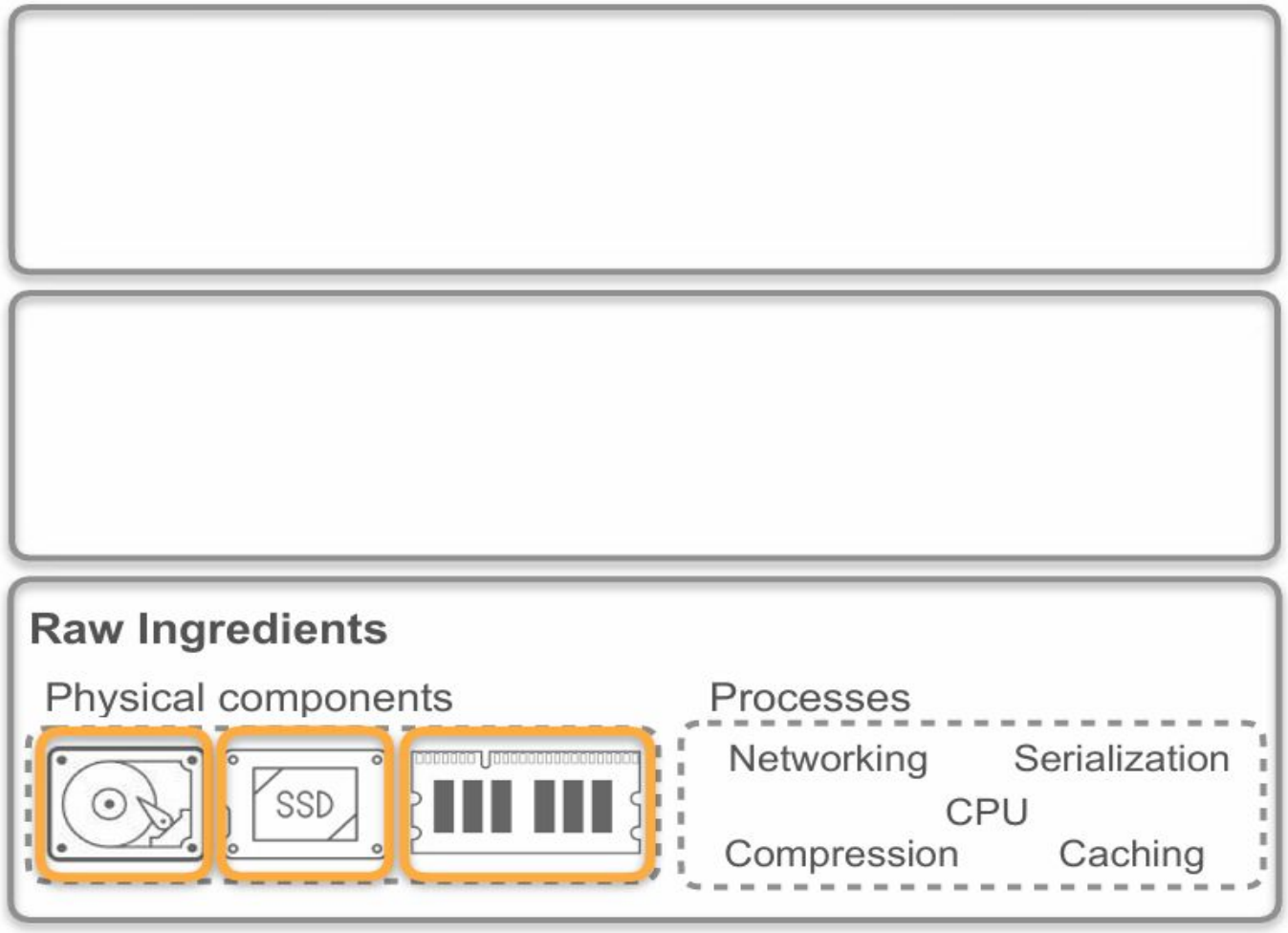
Storage



Storage solution considerations:

- Data type
- Data format
- Data size
- Access and update pattern

Storage Hierarchy



Storage Hierarchy

Management system:

Organizes data in the raw components and allows you to interact with the stored data

OLTP Systems

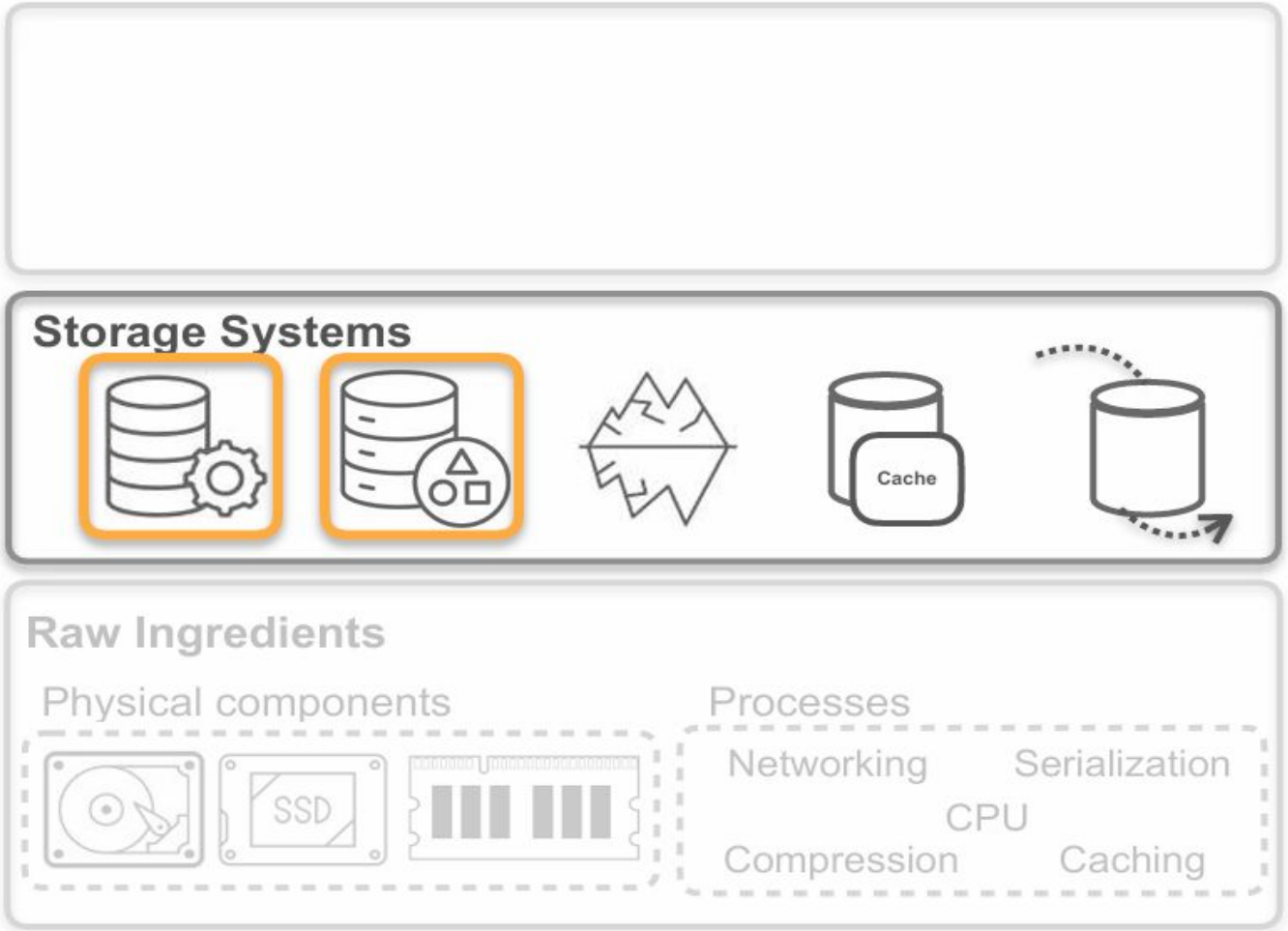
Online Transactional Processing Systems

Focus on performing read and write queries with low latency

OLAP Systems

Online Analytical Processing Systems

Focus on applying analytical activities on data (e.g. aggregation, summarization)



Storage Hierarchy

Management system:

Organizes data in the raw components and allows you to interact with the stored data

OLTP Systems

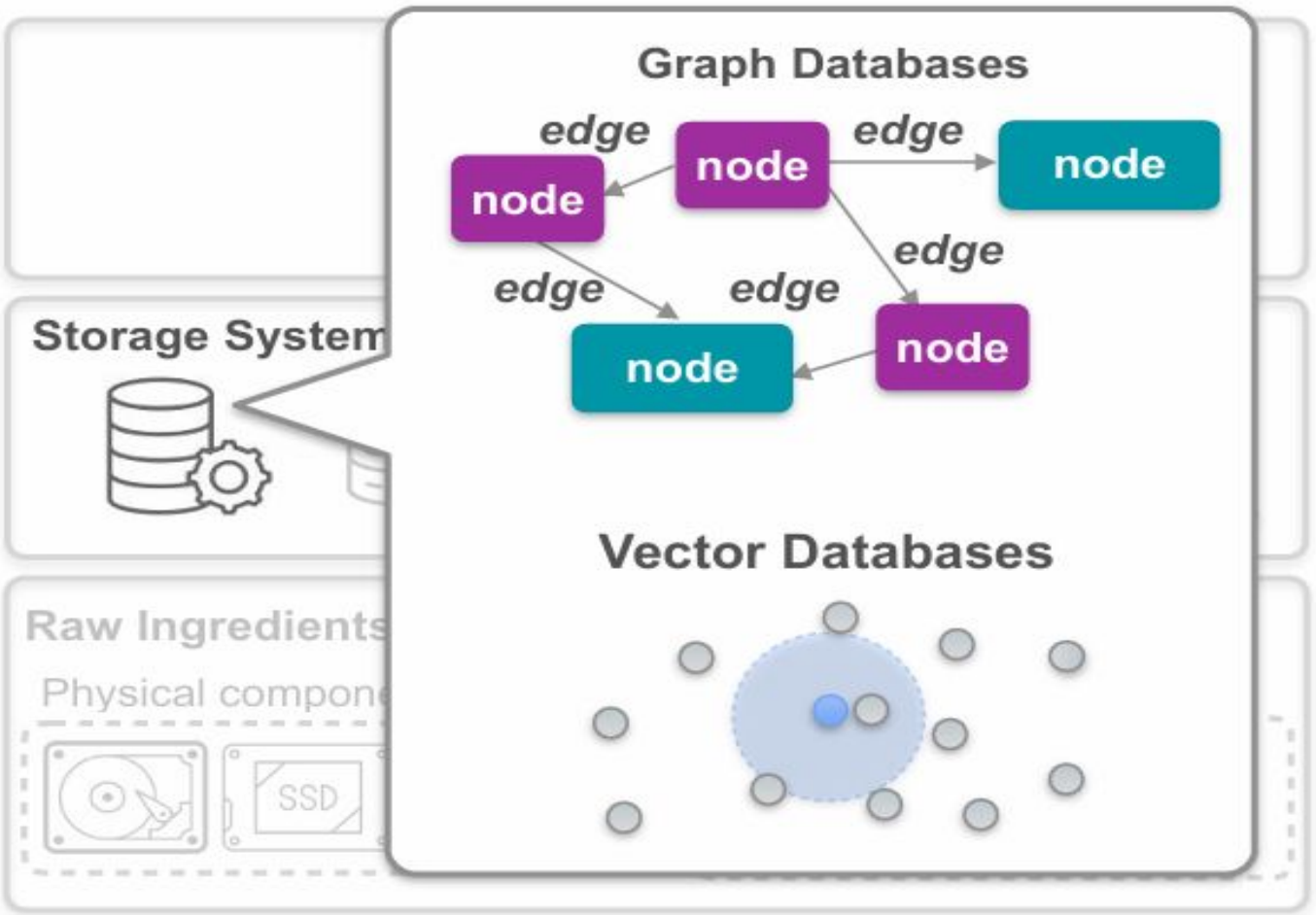
Online Transactional Processing Systems

Focus on performing read and write queries with low latency

OLAP Systems

Online Analytical Processing Systems

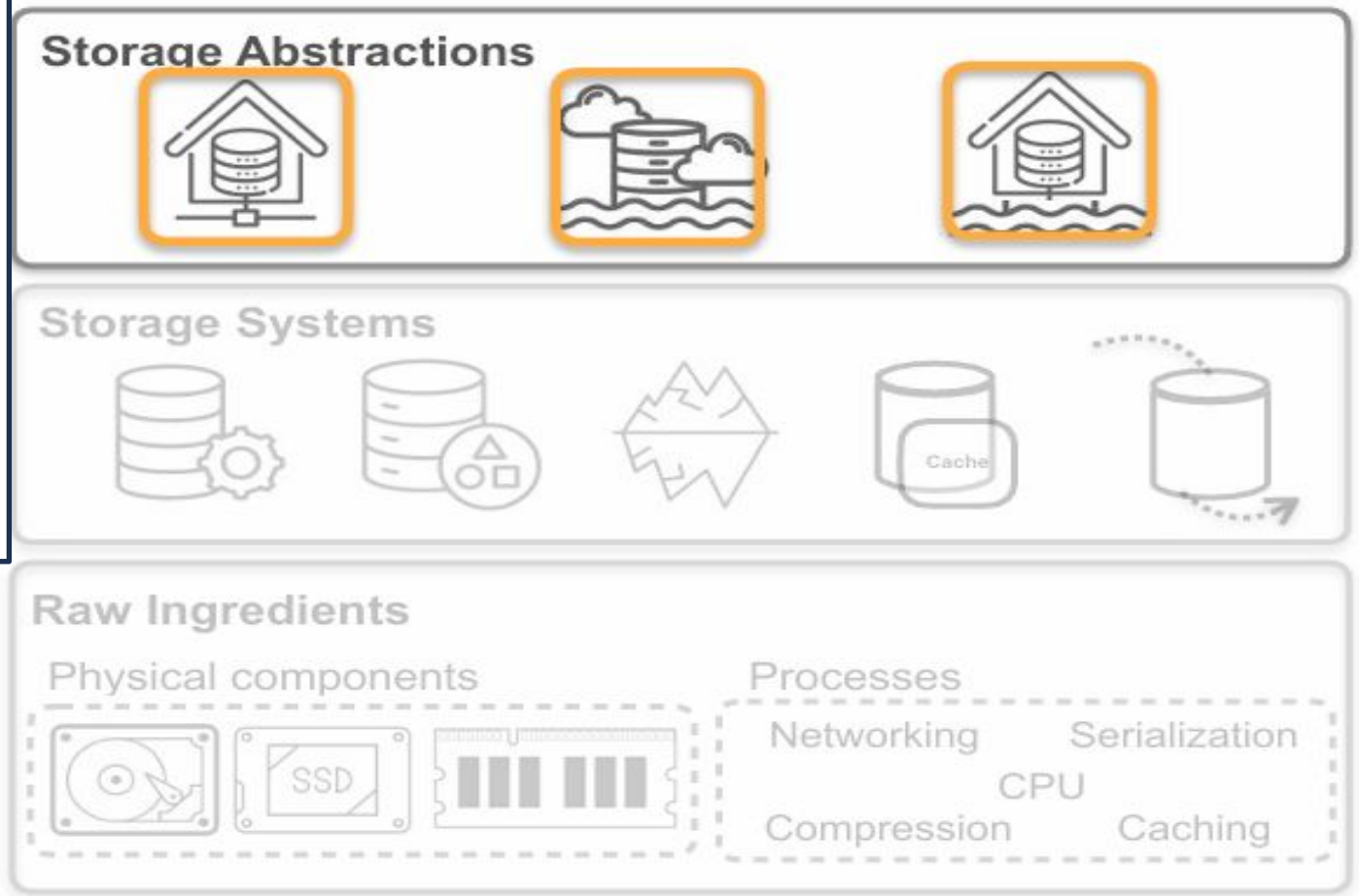
Focus on applying analytical activities on data (e.g. aggregation, summarization)



Sto

Raw Ingredients (bottom layer)
Physical components: HDDs, SSDs, RAM, networking, serialization, compression.Data engineers rarely interact directly with them.

Storage Systems (middle layer)Built on raw ingredients.Managed systems: databases, object storage, OLTP vs. OLAP systems.Also includes specialized systems: graph databases, vector databases (for ML).
Storage
Abstractions (top layer)Cloud-level concepts: data warehouses, data lakes, data lakehouses. Organize systems for large-scale ingestion, transformation, and serving.



RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Raw Ingredients: Physical Components of Data Storage

Raw Storage Ingredients

Persistent Storage Medium

Magnetic disk

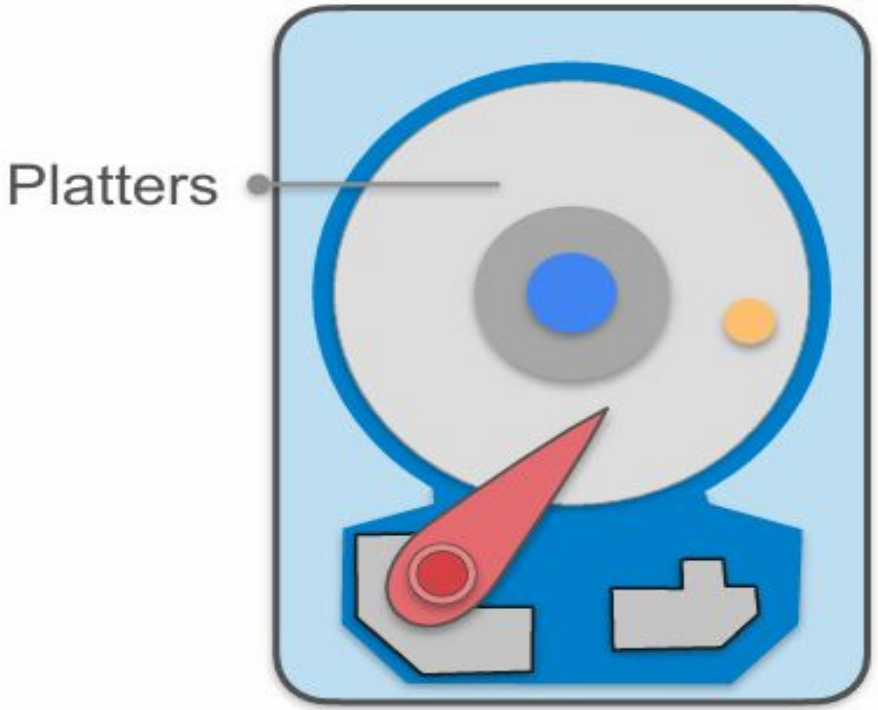
Solid-state storage

Volatile Memory

RAM

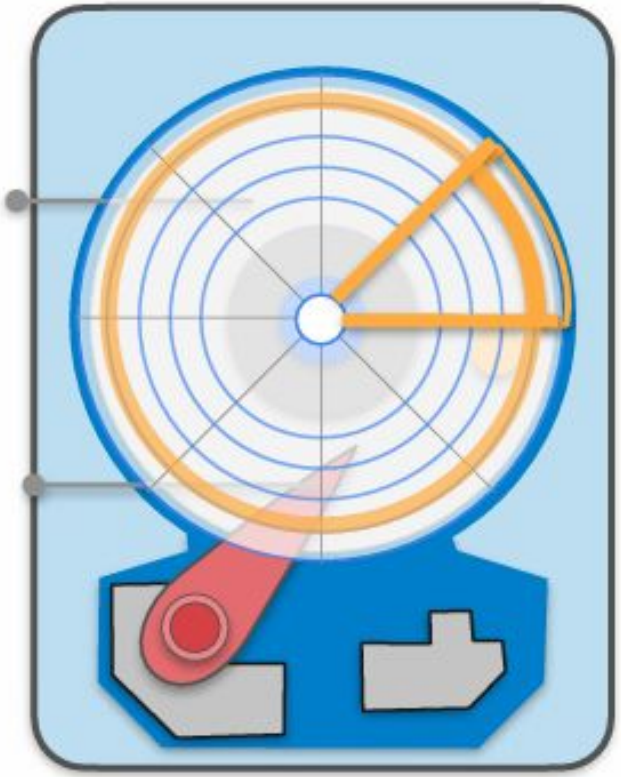
CPU cache

Magnetic Disks



Hard Disk Drives (HDDs)

Magnetic Disks



Hard Disk Drives (HDDs)

Track + Sector

= Address

Write:
Encode binary data by changing the magnetic field

Read:
Converts magnetic field into binary data

Solid State Drives

Solid-State Drives (SSDs)

SSDs read and write data much faster

Performance Comparison

	Magnetic Disk	SSD
Latency	4 milliseconds	
IOPS (Input/output operations per second)	Hundreds	

Commercial magnetic disk drive:
Rotates at 7200 revs/min

Latency
(Data access time)

Seek time **Rotational latency**

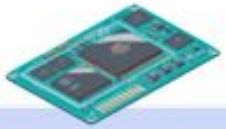
Performance Comparison

	Magnetic Disk	SSD
Latency	4 milliseconds	0.1 milliseconds
IOPS (Input/output operations per second)	Hundreds	Tens of thousands

Solid-State Drives (SSDs)

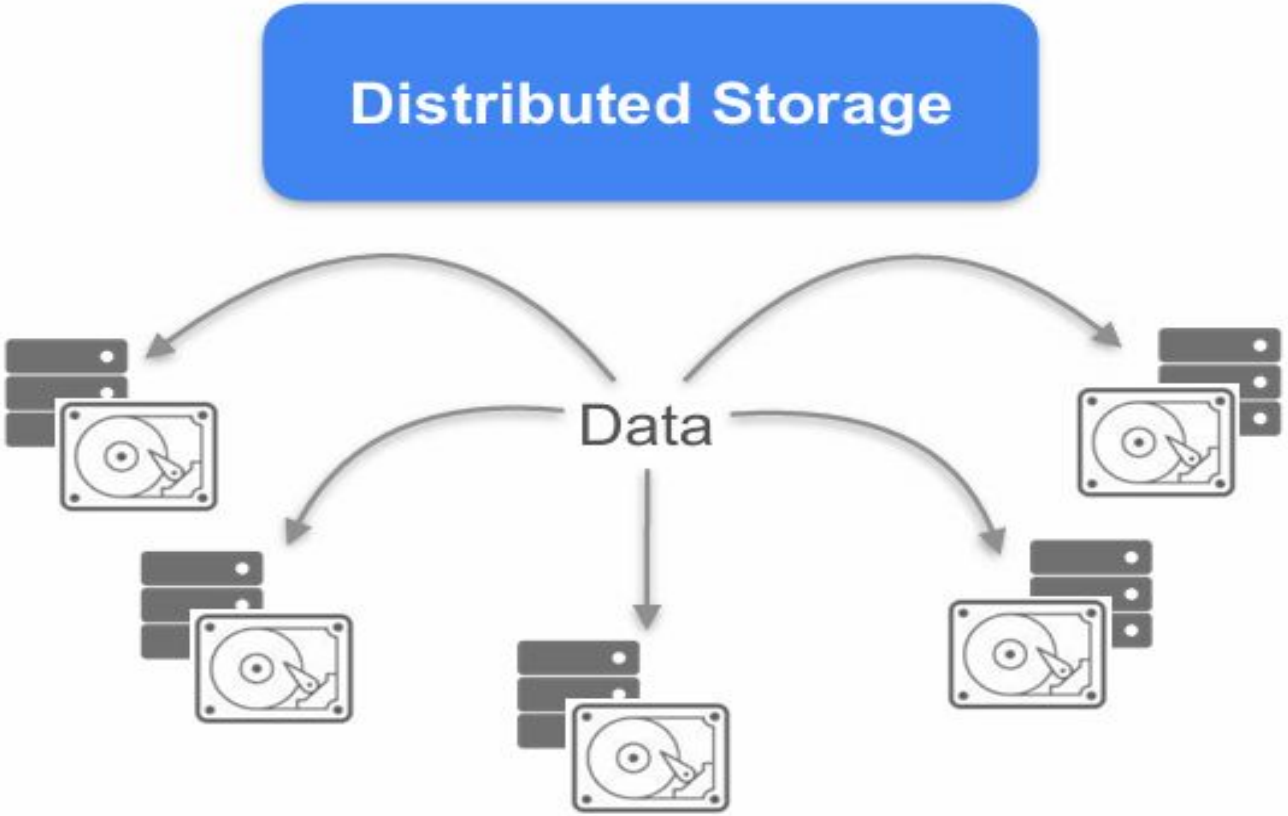
Electrical charges

Performance Comparison



	Magnetic Disk	SSD
Latency	4 milliseconds	0.1 milliseconds
IOPS (Input/output operations per second)	Hundreds	Tens of thousands
Data Transfer Speed (number of bytes read/ written from disk to memory in a second)	Up to 300 MB/s	4 GB/s

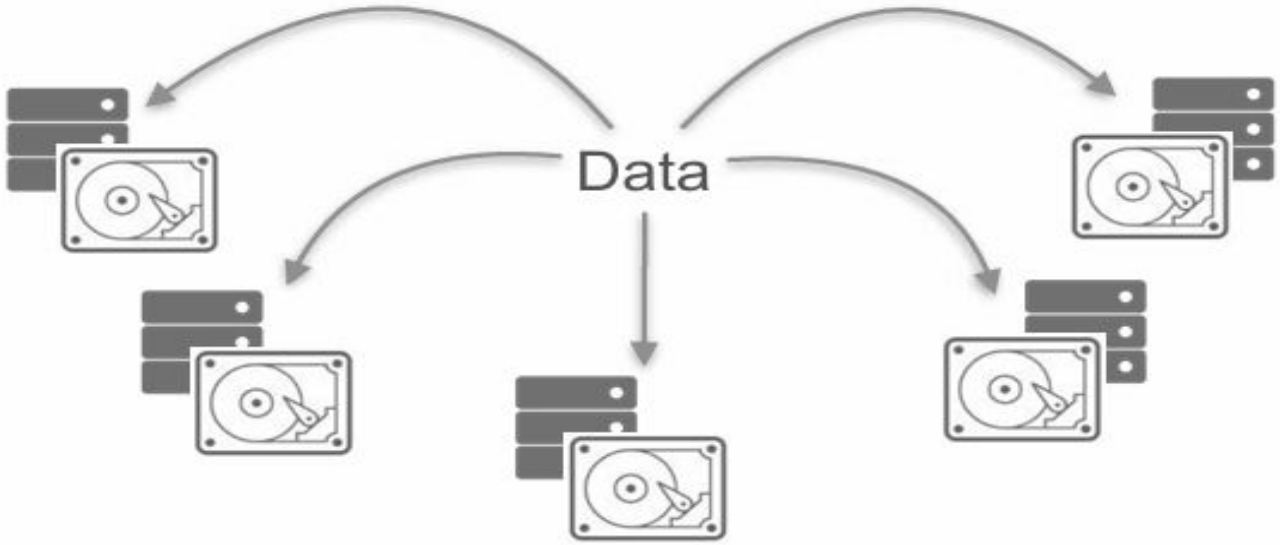
Improving Performance



Data transfer speed limited by network performance

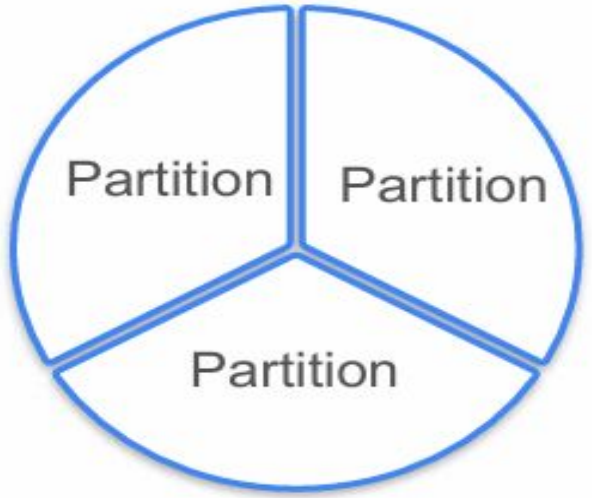
Improving Performance

Distributed Storage



Data transfer speed limited by network performance

Partitioning



Slicing SSDs into partitions

Performance Comparison

	Magnetic Disk	SSD
Latency	4 milliseconds	0.1 milliseconds
IOPS (Input/output operations per second)	Hundreds	Tens of thousands
Data Transfer Speed (number of bytes read/ written from disk to memory in a second)	Up to 300 MB/s	4 GB/s
Cost	$0.03–0.06/GB	$0.08–0.10/GB

2-3 times cheaper

Volatile Memory Ingredients

***Note: these metrics can vary*

	Magnetic Disk	SSD	RAM (Random Access Memory)	CPU Cache
Latency	4 milliseconds	0.1 milliseconds	0.1 microseconds	
IOPS (Input/output operations per second)	Hundreds	Tens of thousands	Millions	
Data Transfer Speed (number of bytes read/ written from disk to memory in a second)	Up to 300 MB/s	4 GB/s	100 GB/s	
Cost	$0.03–0.06/GB	$0.08–0.10/GB	> $3/GB	

30-50 times more expensive

Volatile Memory Ingredients

***Note: these metrics can vary*

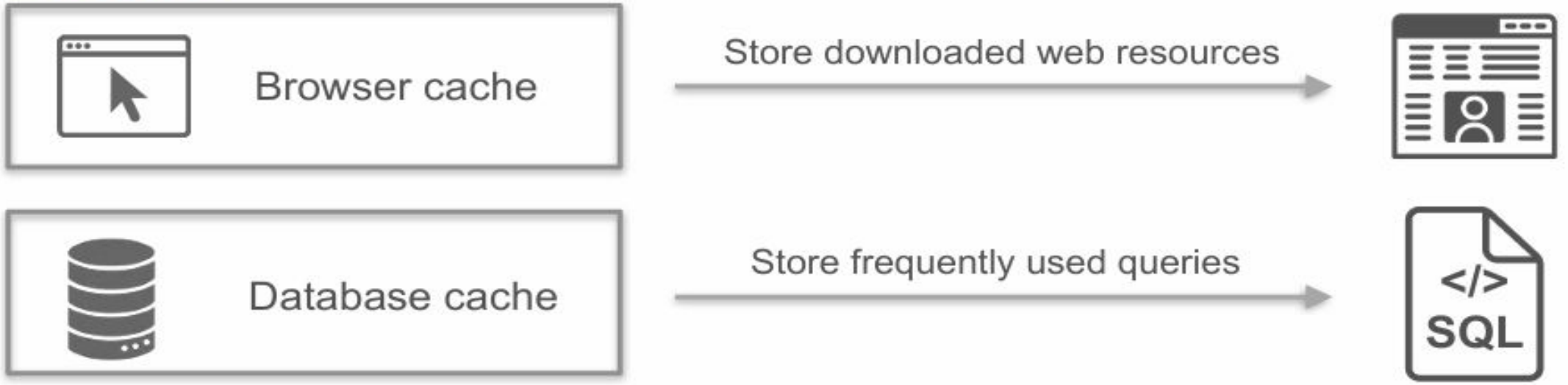



	Magnetic Disk	SSD	RAM (Random Access Memory)	CPU Cache
Latency	4 milliseconds	0.1 milliseconds	0.1 microseconds	1 nanosecond
IOPS (Input/output operations per second)	Hundreds	Tens of thousands	Millions	/
Data Transfer Speed (number of bytes read/ written from disk to memory in a second)	Up to 300 MB/s	4 GB/s	100 GB/s	1 TB/s
Cost	$0.03–0.06/GB	$0.08–0.10/GB	> $3/GB	/

CPU Cache Use Cases

- CPU caching
- Store frequently and recently accessed data in a fast access layer

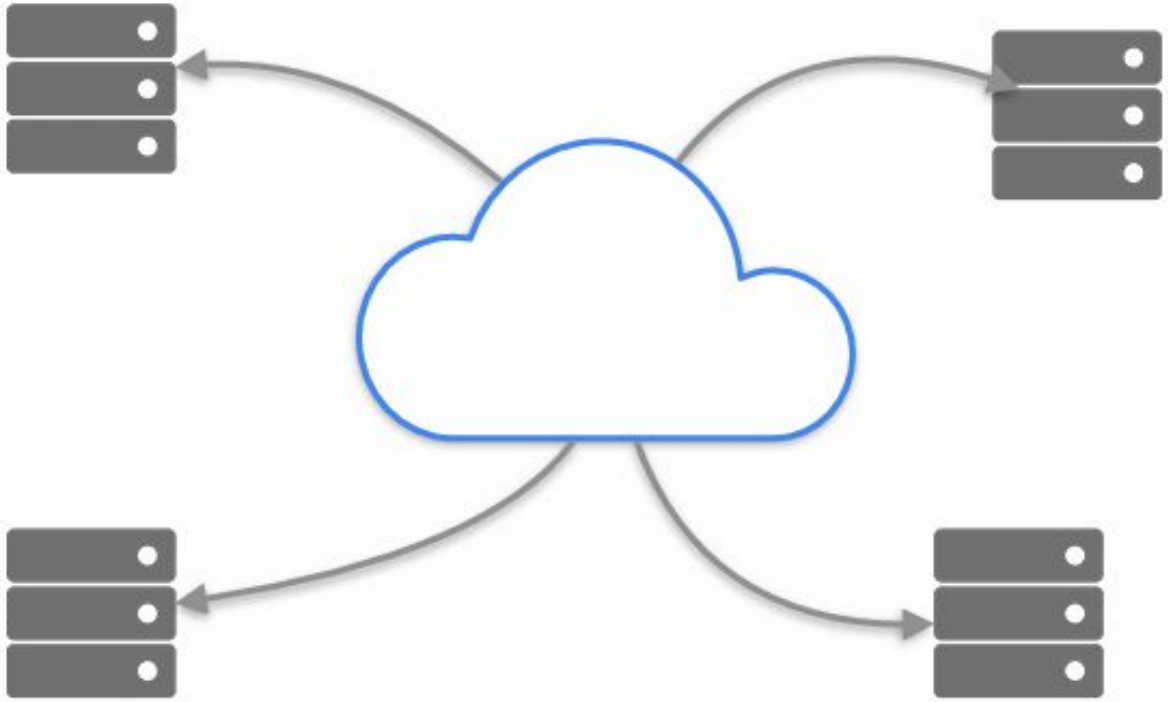
Examples



RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Raw Ingredients: Processes Required for Data Storage

Networking and CPU — "Raw Ingredients" of Storage Systems



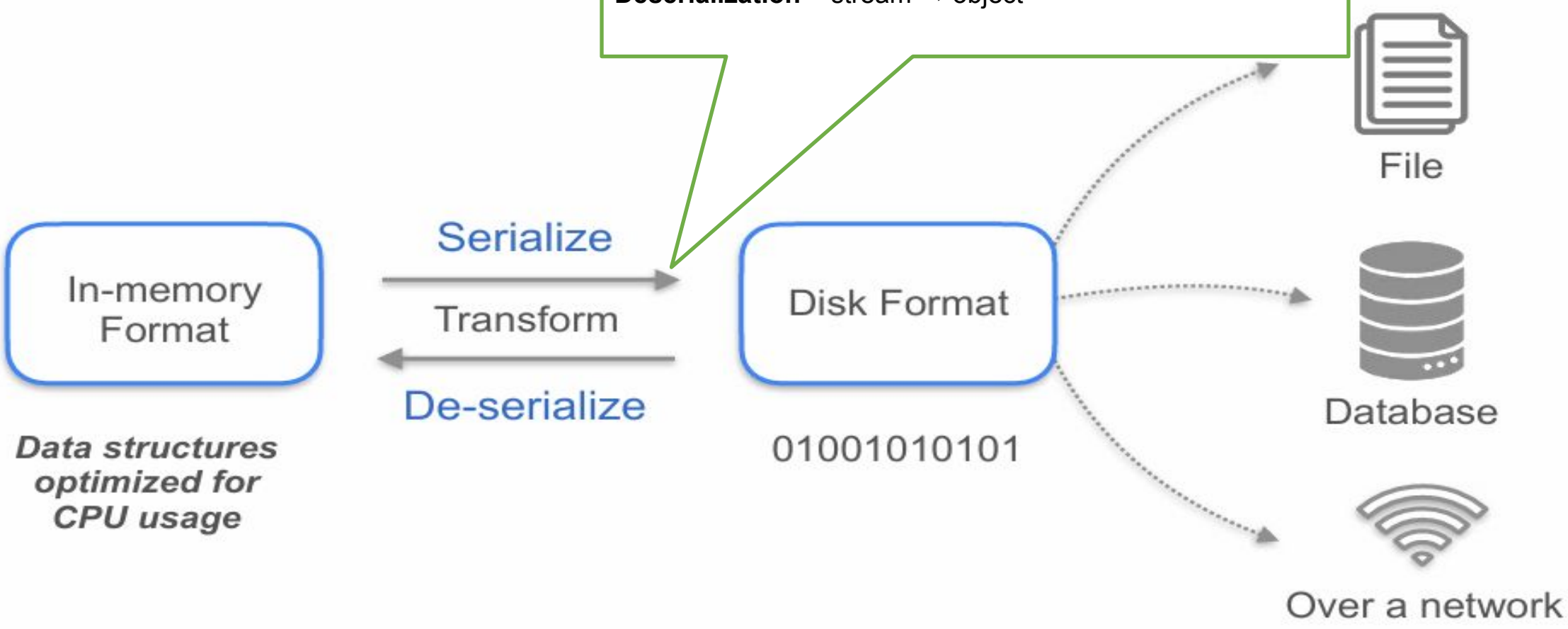
Data in memory (RAM) is stored in formats optimized for CPU access, not for long-term storage or network transfer.

Enhance:

- read and write performance
- data durability
- data availability

Serialization

Serialization = object → stream (bytes, JSON, XML, etc.)
Deserialization = stream → object



Serialization

Data is stored **row by row**, making it efficient for **transactional workloads (OLTP)** where entire records are frequently read or written

Order ID	Price	Product SKU	Quantity	Customer ID
1	40	45865	10	67t
2	23	90234	14	56t
3	45	12558	12	87q
4	50	45682	13	98q

Transactional operations

Physical Storage

Row-Based Serialization

bytes representing the 1st object	bytes representing the 2nd object	...	bytes representing the last object

Column-Based Serialization

Serialization

Order ID	Price	Product SKU	Quantity	Customer ID
1	40	45865	10	67t
2	23	90234	14	56t
3	45	12558	12	87q
4	50	45682	13	98q

Analytical queries

Physical Storage

Row-Based Serialization

bytes representing the 1st row	bytes representing the 2nd row	...	bytes representing the last row

Column-Based Serialization

bytes representing the 1st key	bytes representing the 2nd key	...	bytes representing the last key

Serialization Formats

Human-Readable Textual Formats

Binary Formats

Serialization Formats

Human-Readable Textual Formats

- Row-based format
- Prone to error (no defined schema)
- Adding new rows or columns requires manual handling

Binary Formats

Serialization Formats

Human-Readable Textual Formats

- Row-based format
- Prone to error (no defined schema)
- Adding new rows or columns requires manual handling

- Extensible markup language
- Viewed as a legacy format
- Slow to serialize and deserialize

Binary Formats

Serialization Formats

Human-Readable Textual Formats

- Row-based format
- Prone to error (no defined schema)
- Adding new rows or columns requires manual handling

- Extensible markup language
- Viewed as a legacy format
- Slow to serialize and deserialize

- Used for plain-text object serialization
- Viewed as new standard for data exchange over APIs

Binary Formats

Serialization Formats

Human-Readable Textual Formats

- Row-based format
- Prone to error (no defined schema)
- Adding new rows or columns requires manual handling

- Extensible markup language
- Viewed as a legacy format
- Slow to serialize and deserialize

- Used for plain-text object serialization
- Viewed as new standard for data exchange over APIs

Binary Formats

- Column-based format
- For efficient storage and big data processing

Serialization Formats

Human-Readable Textual Formats

- Row-based format
- Prone to error (no defined schema)
- Adding new rows or columns requires manual handling

- Extensible markup language
- Viewed as a legacy format
- Slow to serialize and deserialize

- Used for plain-text object serialization
- Viewed as new standard for data exchange over APIs

Binary Formats

- Column-based format
- For efficient storage and big data processing

- Row-based format
- Uses a schema to define its data structure
- Supports schema evolution

Serialization Formats

Human-Readable Textual Formats

- Row-based format
- Prone to error (no defined schema)
- Adding new rows or columns requires manual handling

- Extensible markup language
- Viewed as a legacy format
- Slow to serialize and deserialize

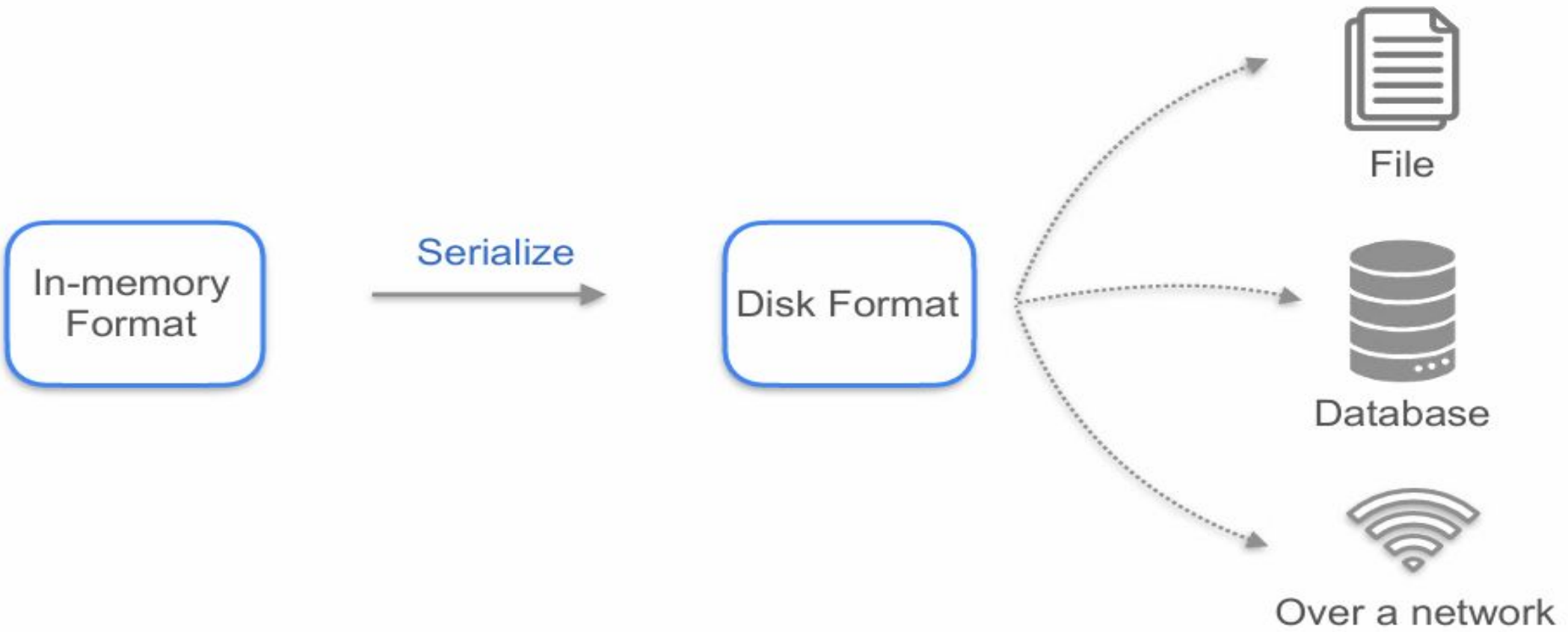
- Used for plain-text object serialization
- Viewed as new standard for data exchange over APIs

Binary Formats

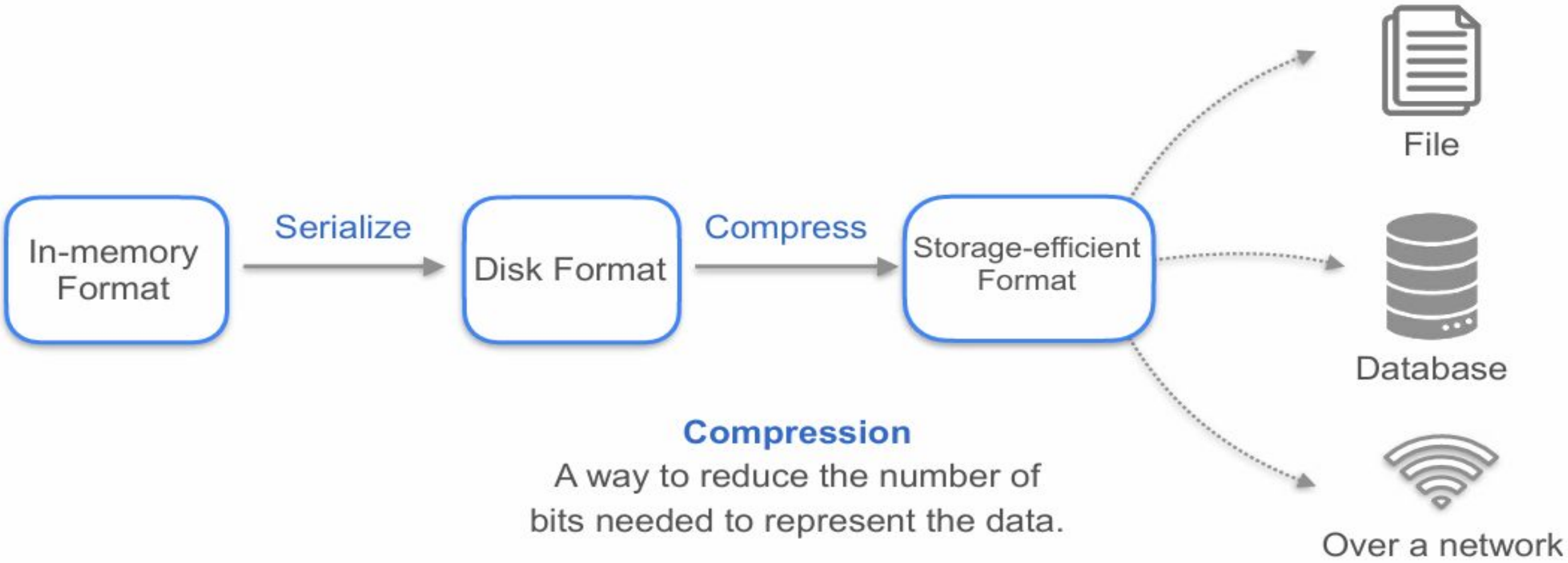
- Column-based format
- For efficient storage and big data processing

- Row-based format
- Uses a schema to define its data structure
- Supports schema evolution

Serialization



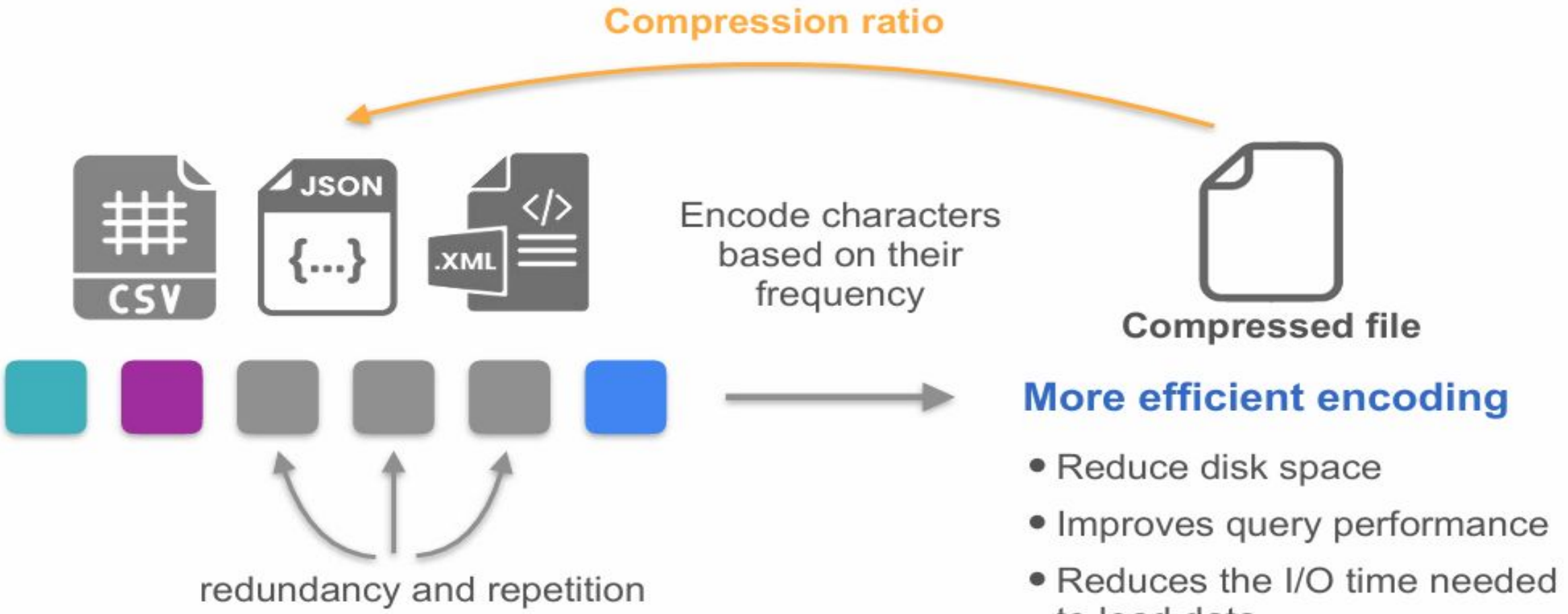
Serialization



Compression

A way to reduce the number of bits needed to represent the data.

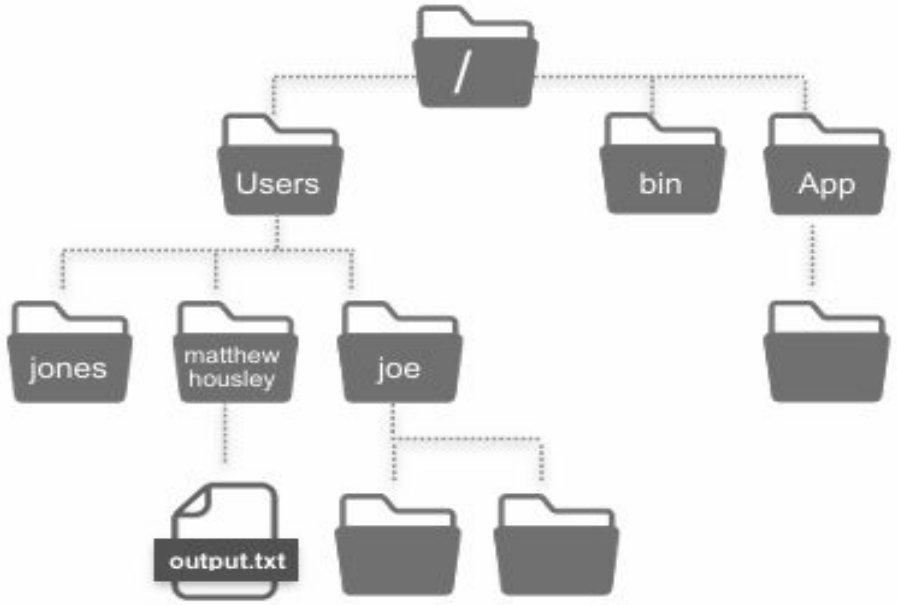
Compression



RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Cloud Storage Options: Block, Object and File storage

File Storage



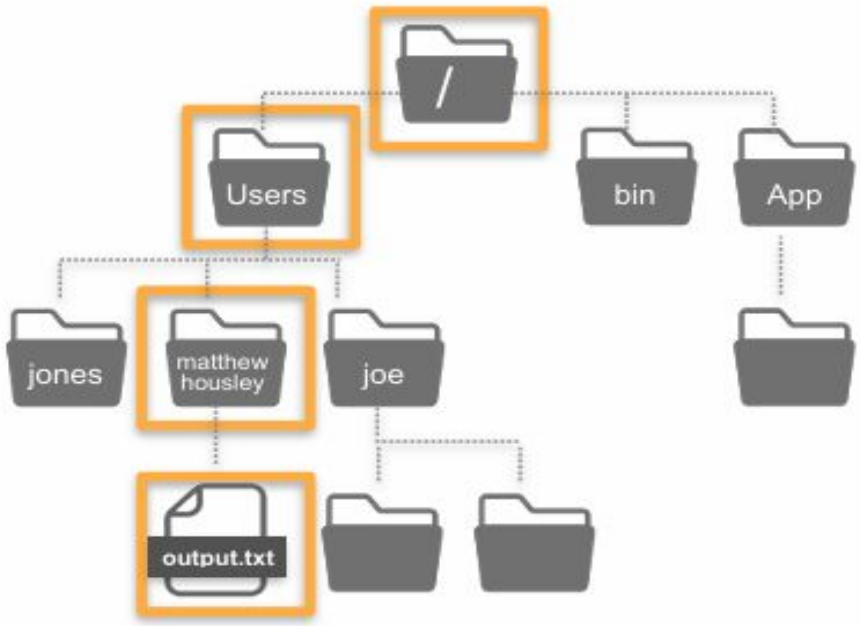
File Storage

Organizes files into a directory tree

Each directory contains metadata about its files and subfolders :

- Name
- Owner
- Last modified date
- Permissions
- Pointer to the actual entity

File Storage



/Users/matthewhousley/output.txt

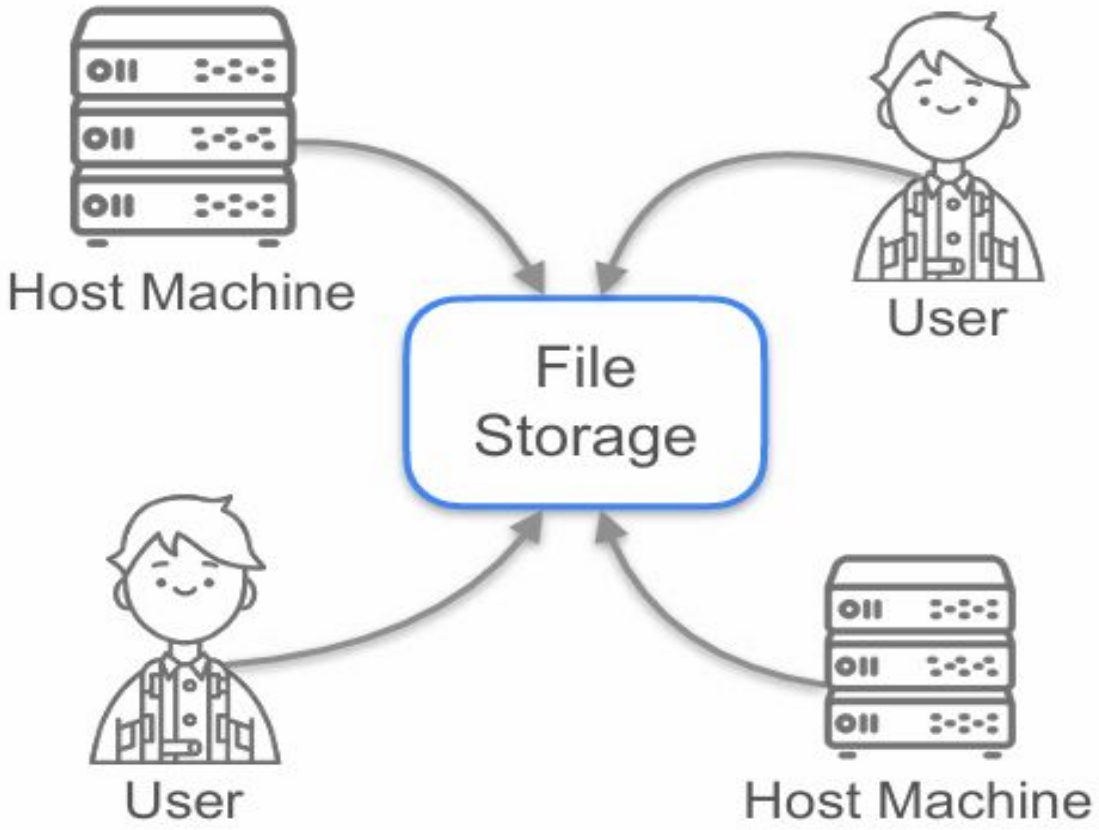
File Storage

Organizes files into a directory tree

Each directory contains metadata about its files and subfolders :

- Name
- Owner
- Last modified date
- Permissions
- Pointer to the actual entity

File Storage Use Cases

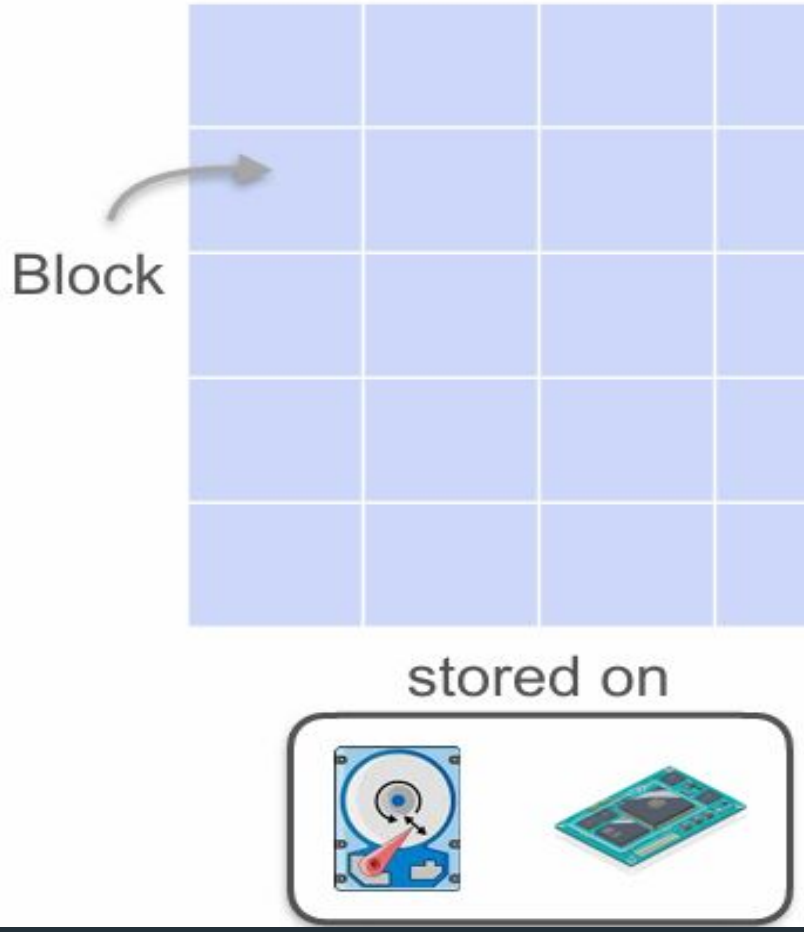


Cloud File Storage Service

Amazon Elastic File System (EFS)

- Provides you access to shared files over a network
- Networking, scaling, and configuration are handled by the cloud vendor

Block Storage

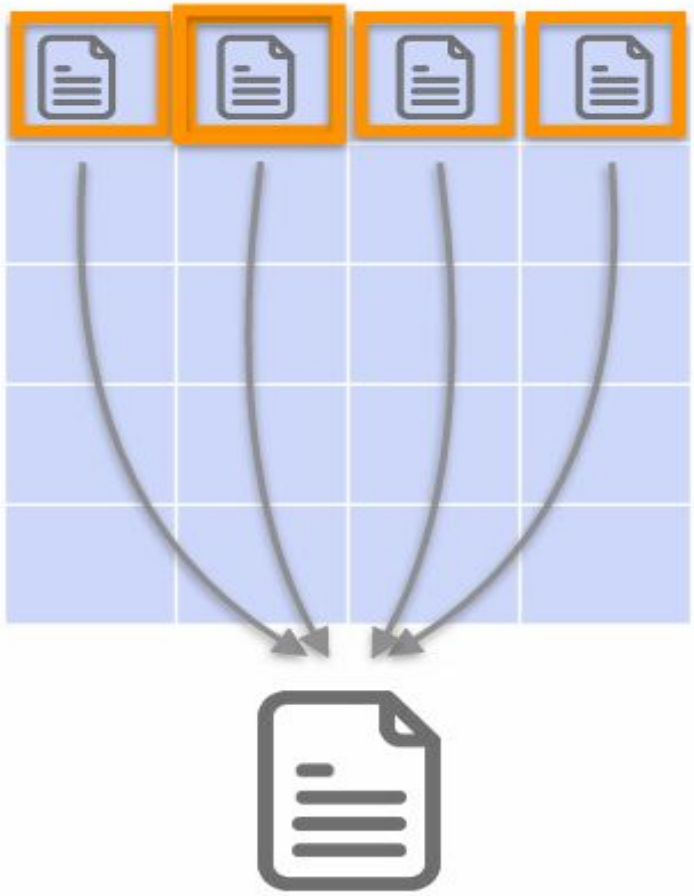


Block Storage

Divides files into small, fixed-size blocks of data and stores them on disk

- Each block has a unique identifier
- You can efficiently retrieve and modify data in individual blocks
- You can distribute blocks of data across multiple storage disks
  - Higher scalability
  - Stronger data durability

Block Storage



Lookup Table

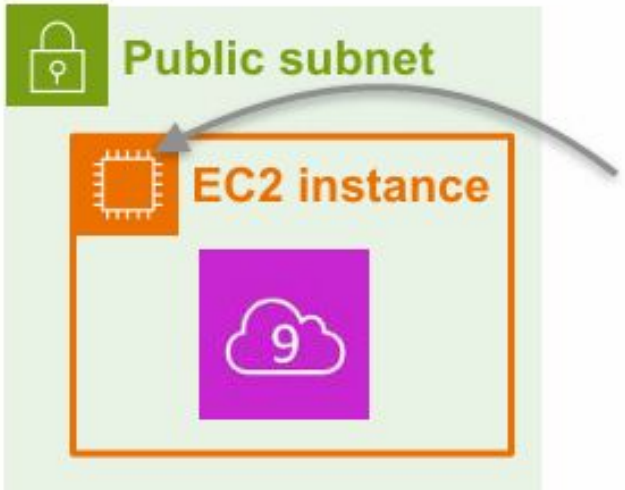
File Piece	Block Identifier
First piece	1232
Second piece	1234
Third piece	1236
Fourth piece	1238

Block Storage Use Cases

- Ideal for frequent access and modification
- Enables OLTP systems to perform small and frequent read and write operations with low latency
- Provides persistent storage for virtual machines

Default storage for EC2

Amazon Elastic Block Store (EBS)



Attach a root storage device backed by a block storage volume

1. SSD for latency-sensitive workloads
2. Magnetic disks to store infrequently-accessed data

Object Storage

Object Storage Stores immutable files as data objects in a flat structure

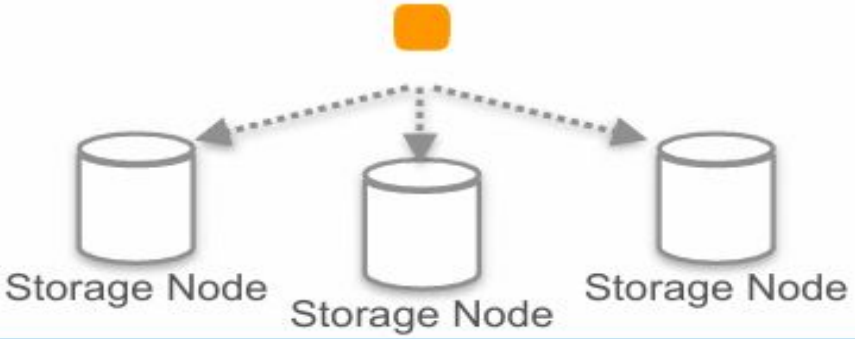
Object Storage

Object Storage Stores immutable files as data objects in a flat structure

- Each object is immutable
- To update the file you have to re-write the entire object
- Can scale horizontally and support performant parallel operations

s3://o'reilly-data-engineering-book/data-example.json

The Bucket Object key



Object Storage Use Cases

Ideal for…	Not ideal for…

• Storage layer of cloud data warehouses or data lakes
• Storing data needed in OLAP systems
• Machine learning pipelines
  • Raw text
  • Images
  • Videos
  • Audio
	• Not good at supporting transactional workloads

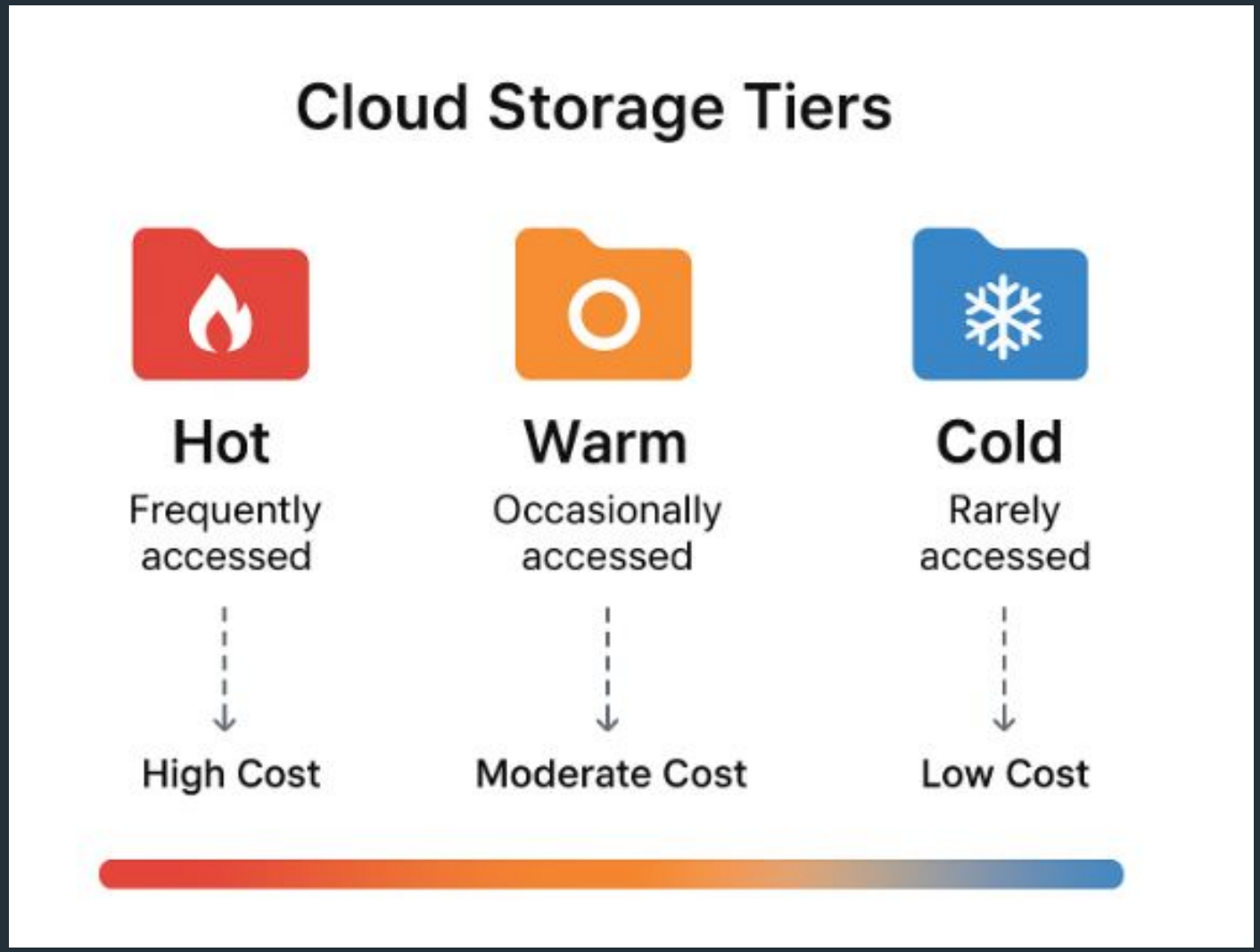
Cloud Storage Options

File storage → easy file sharing.
Block storage → OLTP, frequent read/write.
Object storage → OLAP, big data, massive scalability.

File Storage	Block Storage	Object Storage
• Supports data sharing • Easy to manage with low performance and scalability requirements	• Supports transactional workloads • Allows frequent read and write operations with low latency	• Supports analytical queries on massive datasets • Offers high scalability and parallel data processing

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Storage Tiers – Hot, Warm, & Cold Data


Cloud Storage Tiers
Hot
Frequently accessed
High Cost
Warm
Occasionally accessed
Moderate Cost
Cold
Rarely accessed
Low Cost

Storage Tiers

	Hot Storage	Warm Storage	Cold Storage
Access Frequency			
Example			
Storage Medium			
Storage Cost			
Retrieval Cost			

Storage Tiers

	Hot Storage	Warm Storage	Cold Storage
Access Frequency	Very frequent		
Example	Product recommendation application		
Storage Medium	SSD & Memory		
Storage Cost	High		
Retrieval Cost	Low		

Storage Tiers

	Hot Storage	Warm Storage	Cold Storage
Access Frequency	Very frequent	Less frequent	
Example	Product recommendation application	Regular reports and analyses	
Storage Medium	SSD & Memory	Magnetic disks or hybrid storage systems	
Storage Cost	High	Medium	
Retrieval Cost	Low	Medium	

Storage Tiers

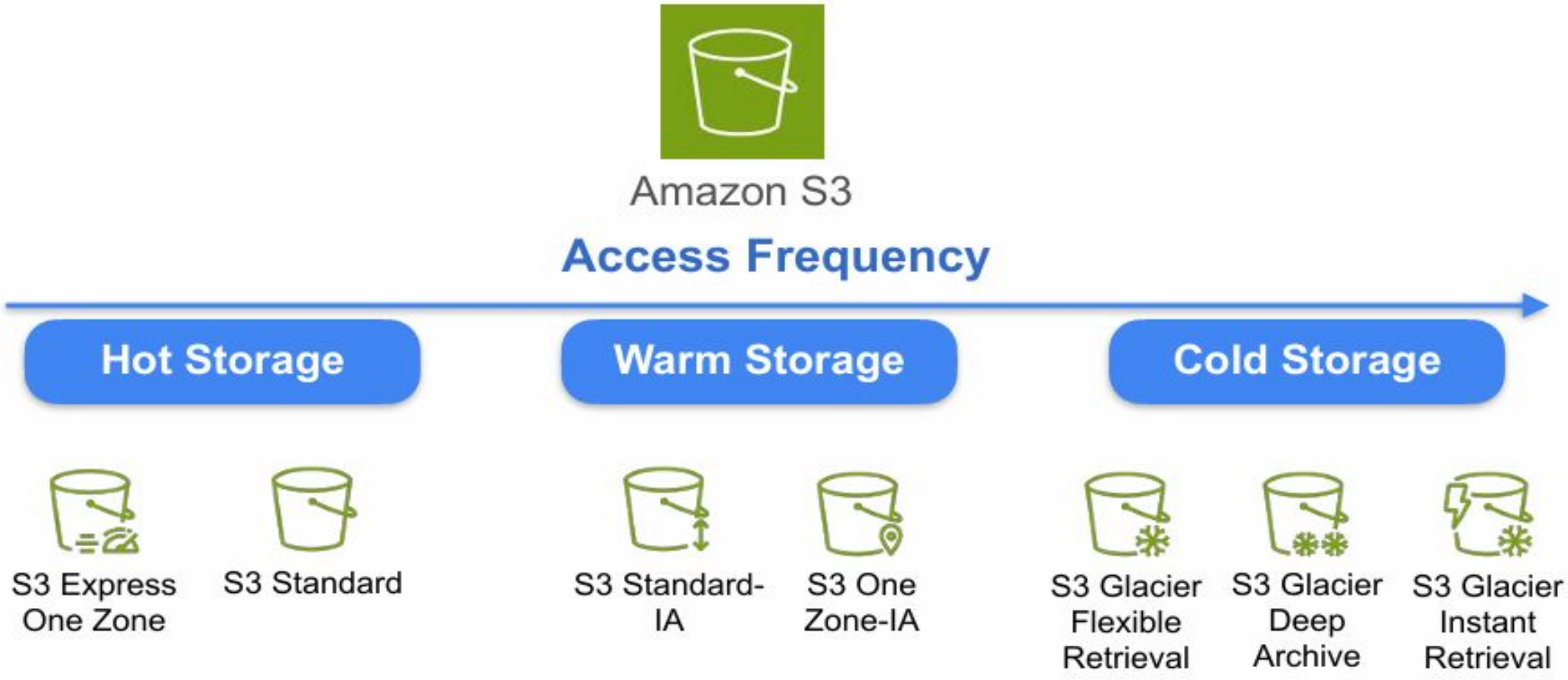
Access Frequency	Very frequent	Less frequent	Infrequent
Example	Product recommendation application	Regular reports and analyses	Archive
Storage Medium	SSD & Memory	Magnetic disks or hybrid storage systems	Low-cost magnetic disks
Storage Cost	High	Medium	Low
Retrieval Cost	Low	Medium	High

Storage cost decreases as retrieval speed decreases (hot → warm → cold).
Optimal design often uses a combination of tiers:
Hot for real-time access
Warm for periodic access
Cold for archival purposes

Storage Tiers

	Hot Storage	Warm Storage	Cold Storage
Access Frequency	Very frequent	Less frequent	Infrequent
Example	Product recommendation application	Regular reports and analyses	Archive
Storage Medium	SSD & Memory	Magnetic disks or hybrid storage systems	Low-cost magnetic disks
Storage Cost	High	Medium	Low
Retrieval Cost	Low	Medium	High

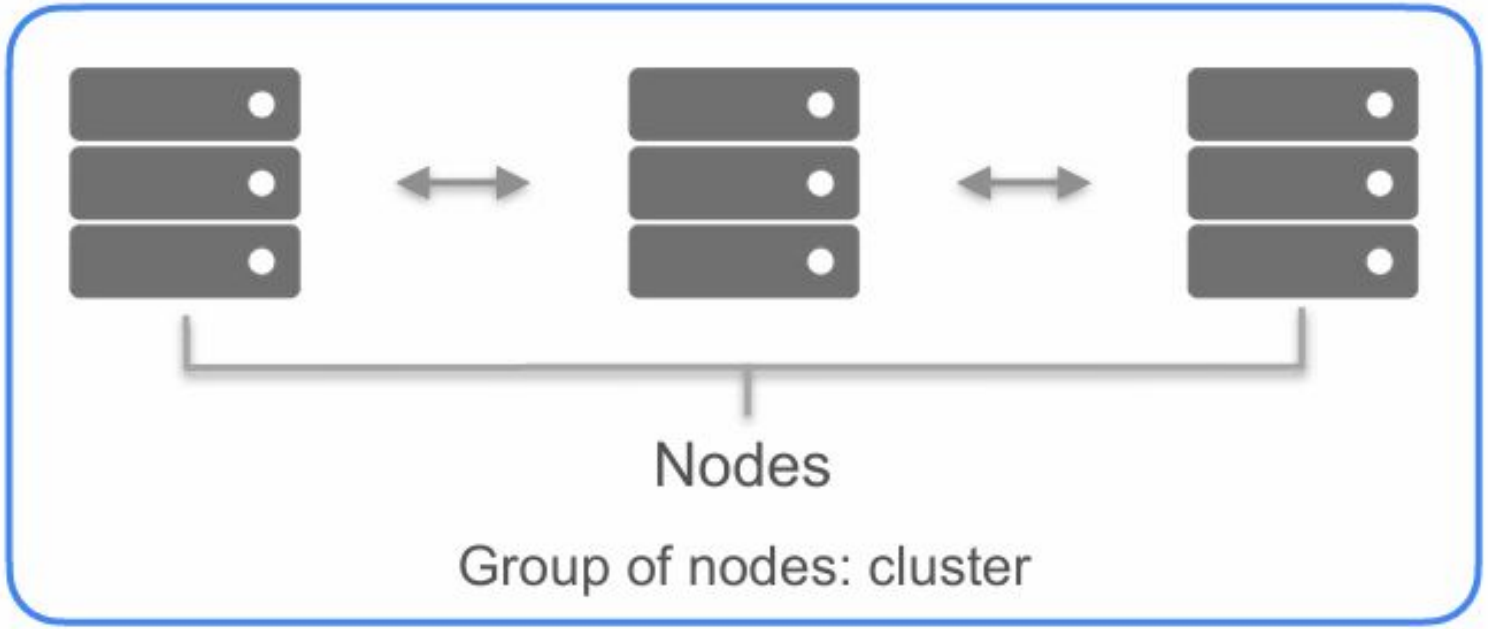
AWS Storage Tiers



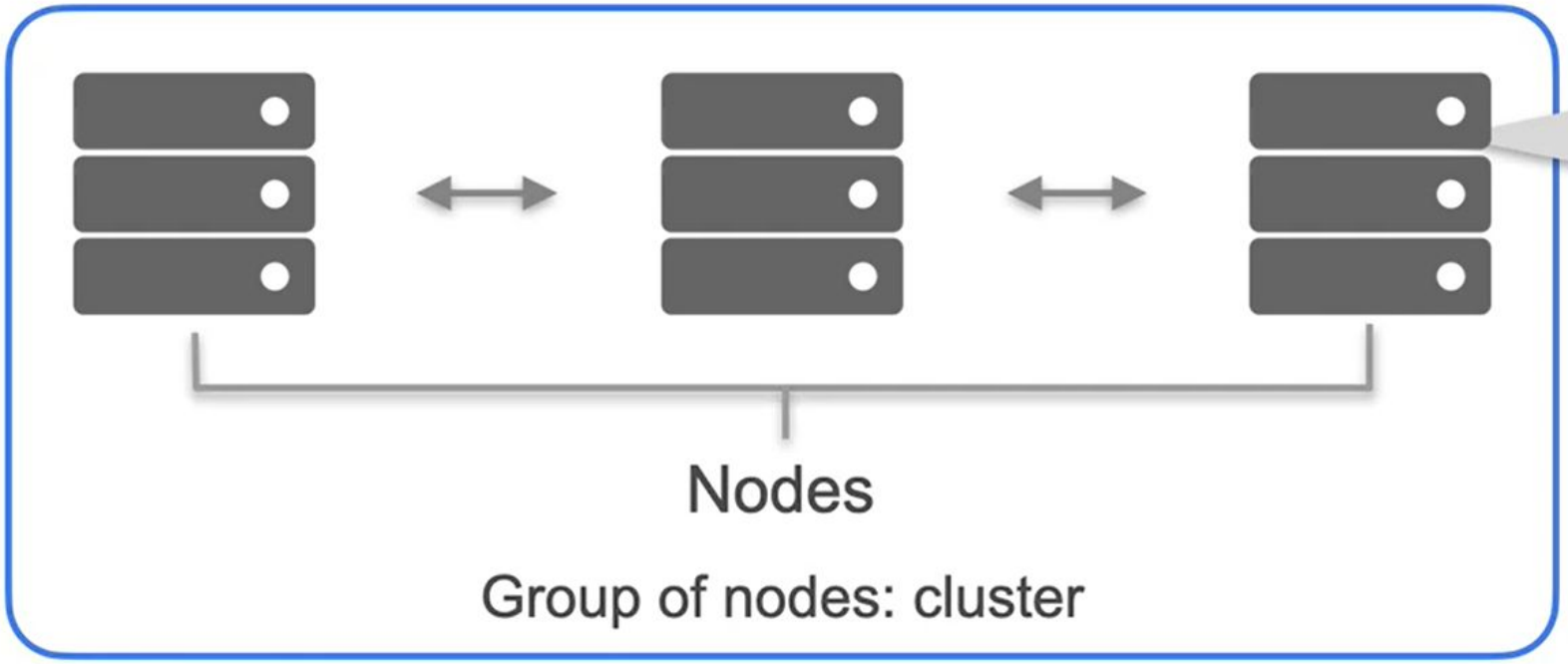
RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

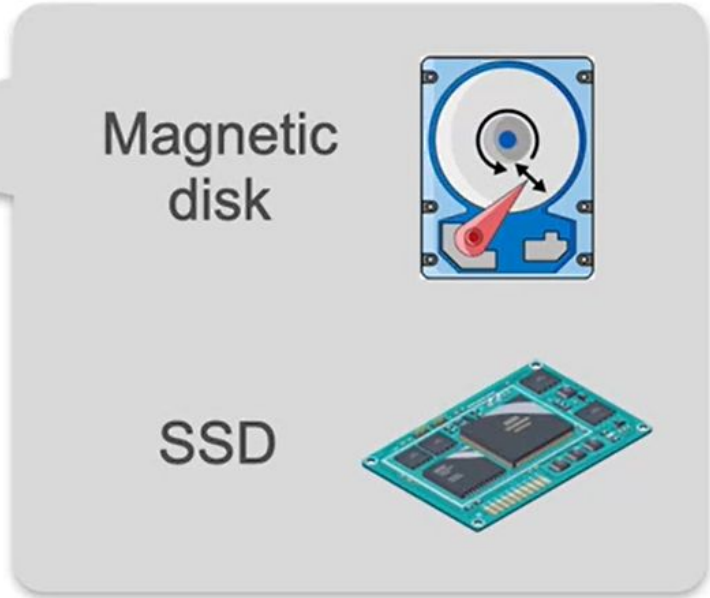
Distributed Storage Systems

How Distributed Storage Systems Work



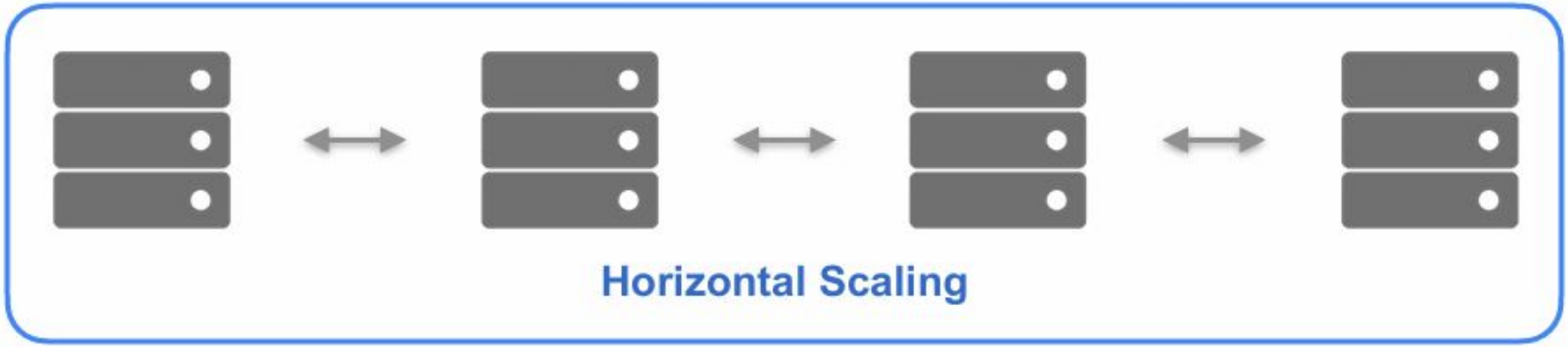
How Distributed Storage Systems Work



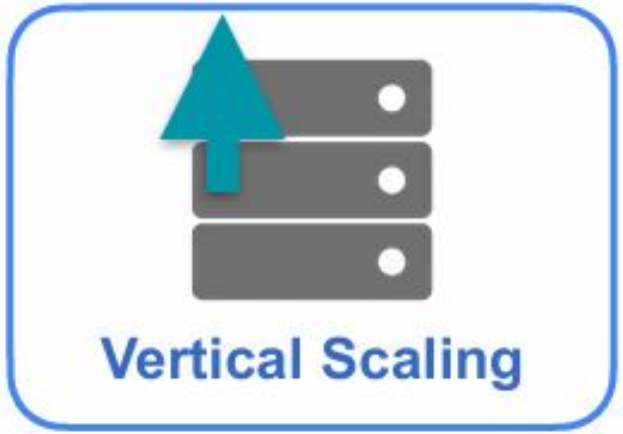


Total Capacity = 

How Distributed Storage Systems Work

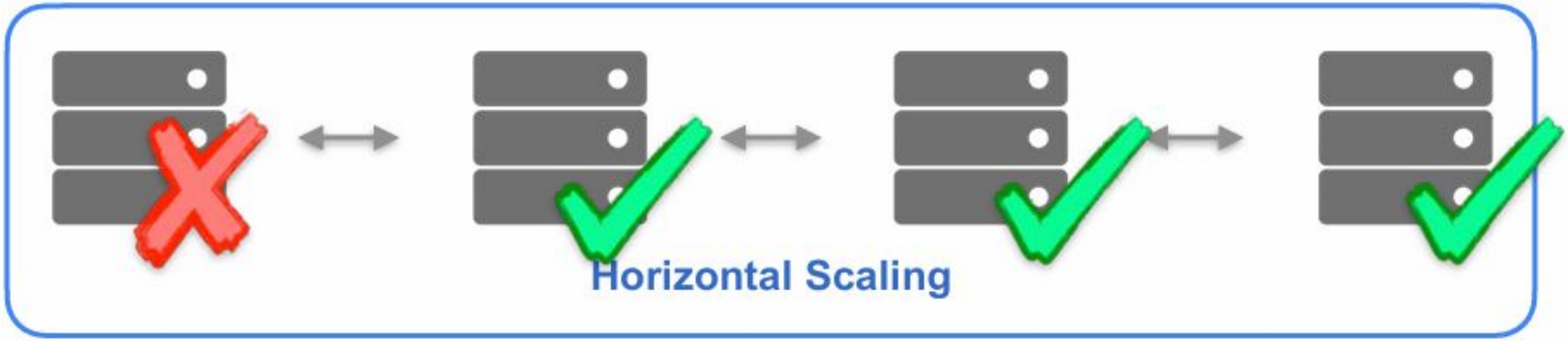


Single Machine Storage Architecture



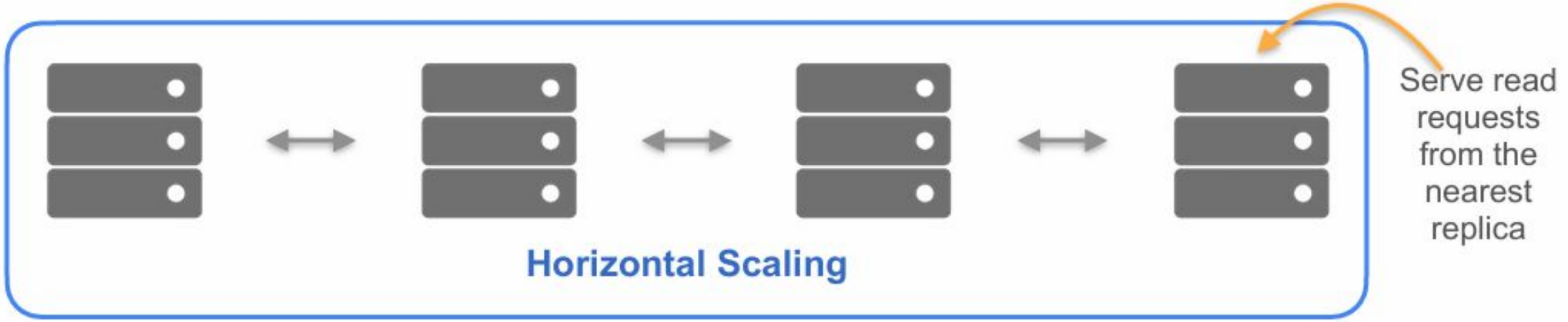
Vertical scaling: Scale up a single server. Simple but limited.
Horizontal scaling: Scale out across multiple servers. Complex but highly scalable.

How Distributed Storage Systems Work



- Higher fault tolerance and data durability
- High availability

How Distributed Storage Systems Work



- Higher fault tolerance and data durability
- High availability
- Process many read and write operations in parallel
- Fast data access

Advantages of Distributed Storage Systems

Distributed Storage Architecture

Object Storage

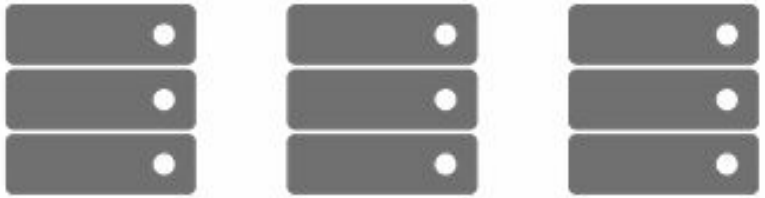
Cloud
Data Warehouse

Methods for Distributing Data

- Replication
- Partitioning

Methods for Distributing Data

Replication

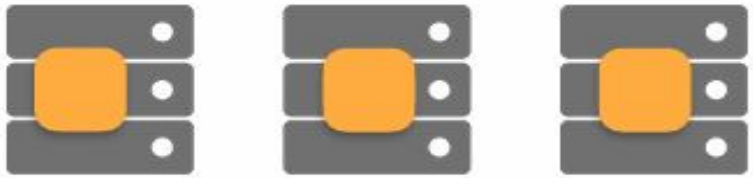


High availability and performance

Partitioning

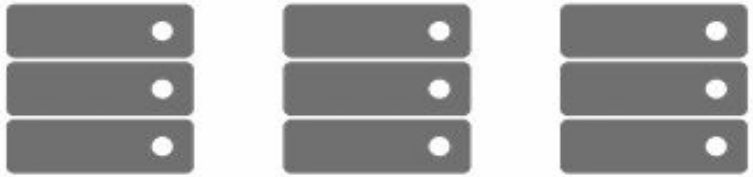
Methods for Distributing Data

Replication



High availability and performance

Partitioning



Sharding

Methods for Distributing Data

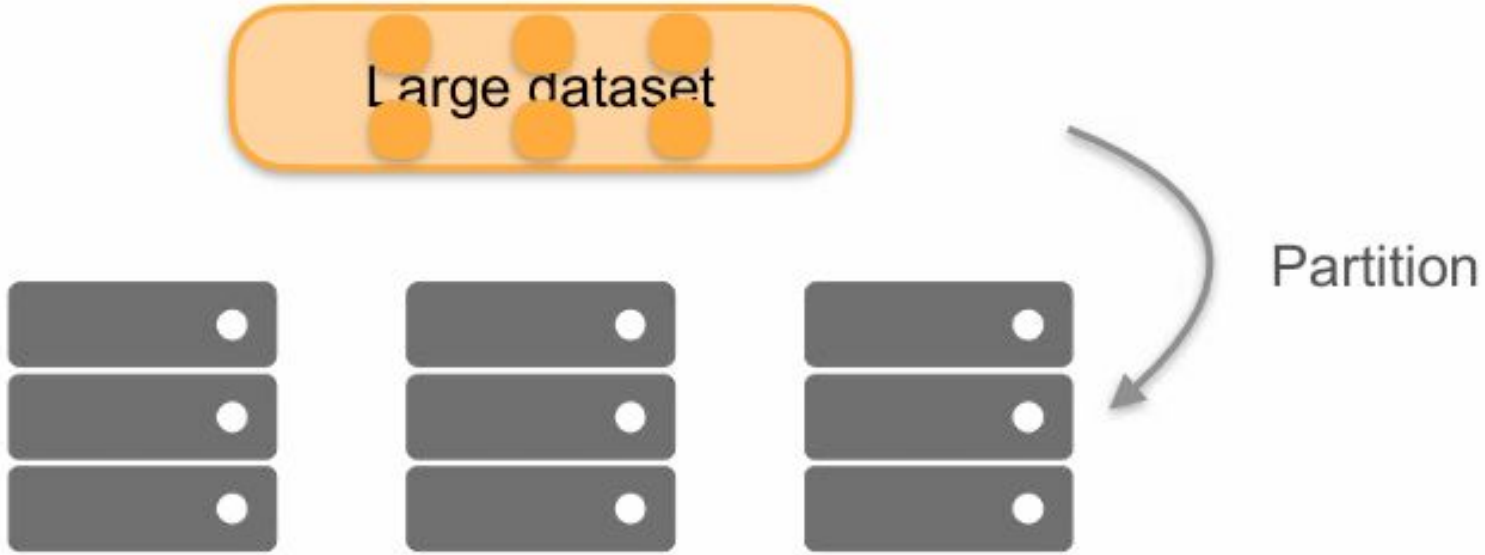
Replication

High availability and performance

Partitioning

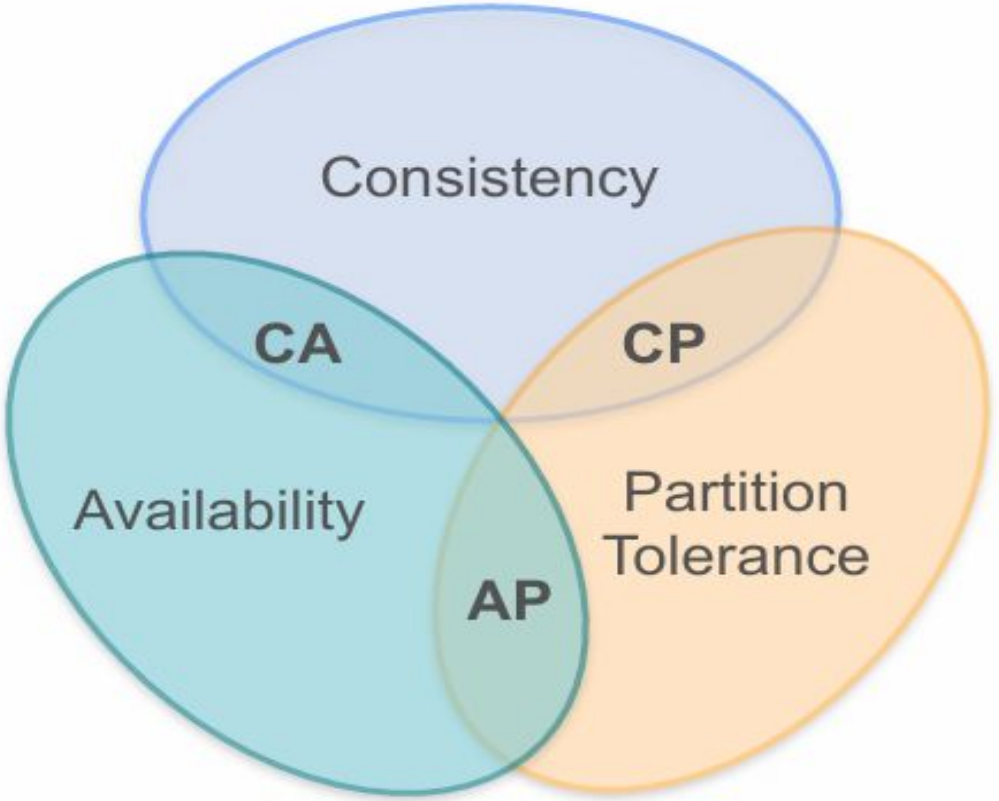
Sharding

Methods for Distributing Data



Distributed Storage Considerations – CAP Theorem

The CAP theorem

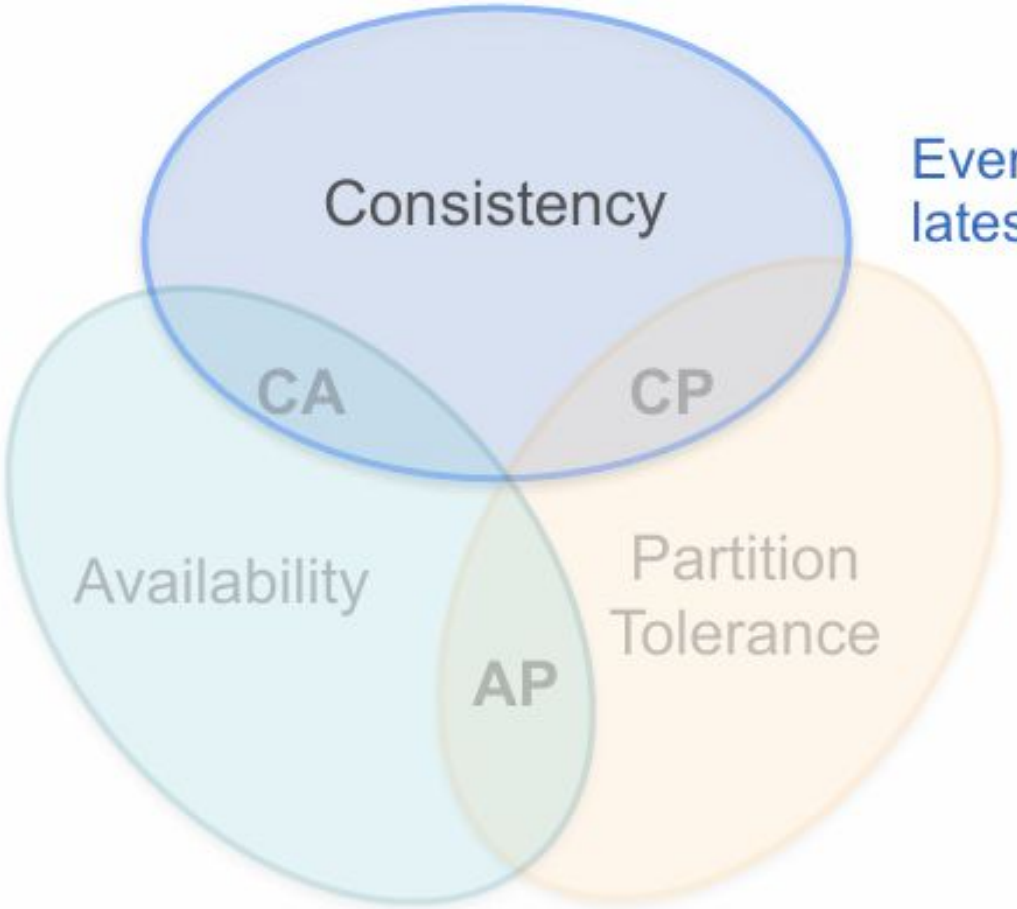


Distributed Storage Considerations – CAP Theorem

ACID

Consistency

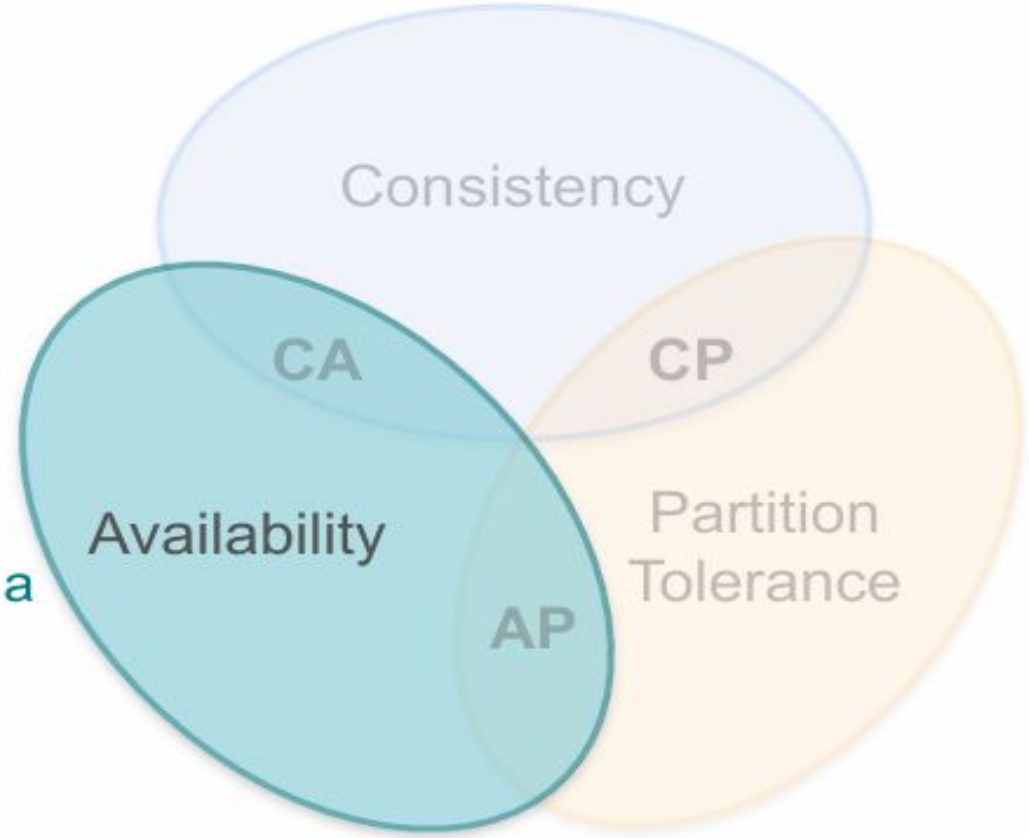
Any change to data must follow the set of rules defined by database schema



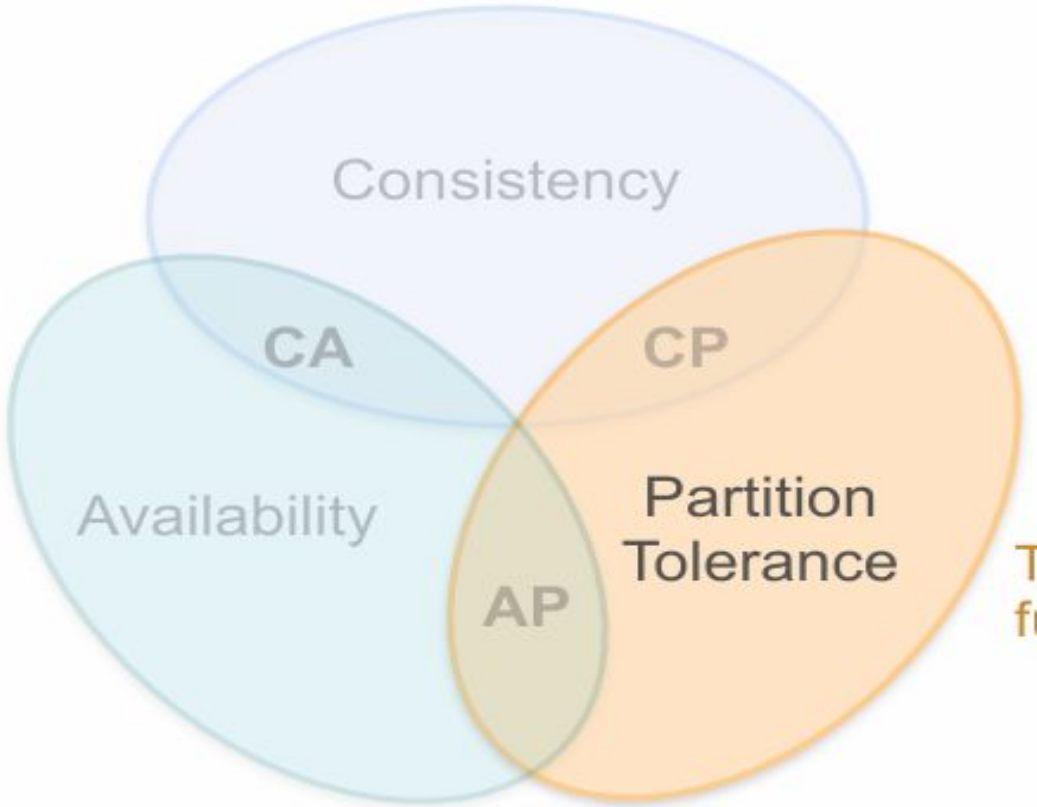
Every read reflects the latest write operation

Distributed Storage Considerations – CAP Theorem

Every request will receive a response



Distributed Storage Considerations – CAP Theorem

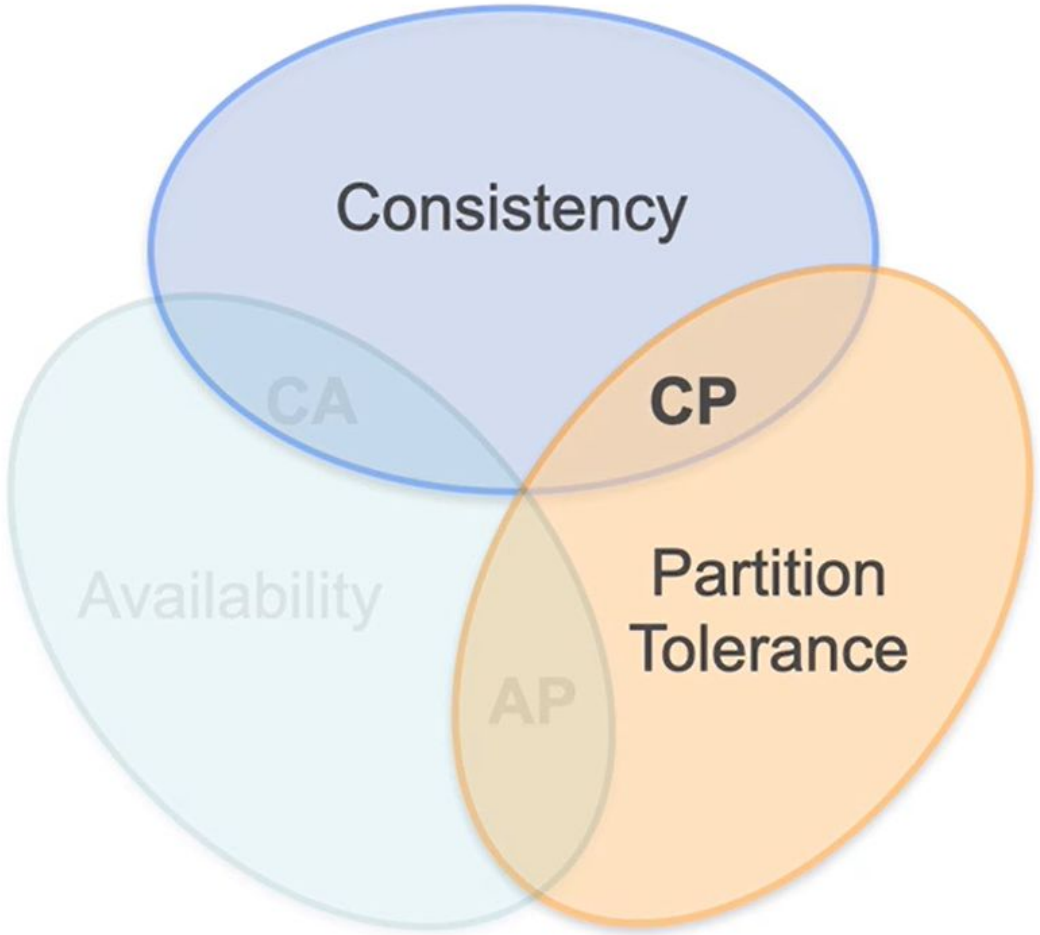


The system continues to function

Distributed Storage Considerations – CAP Theorem

Scenario:
Accessing a node that's still being updated

Option 1:
Cancel the request

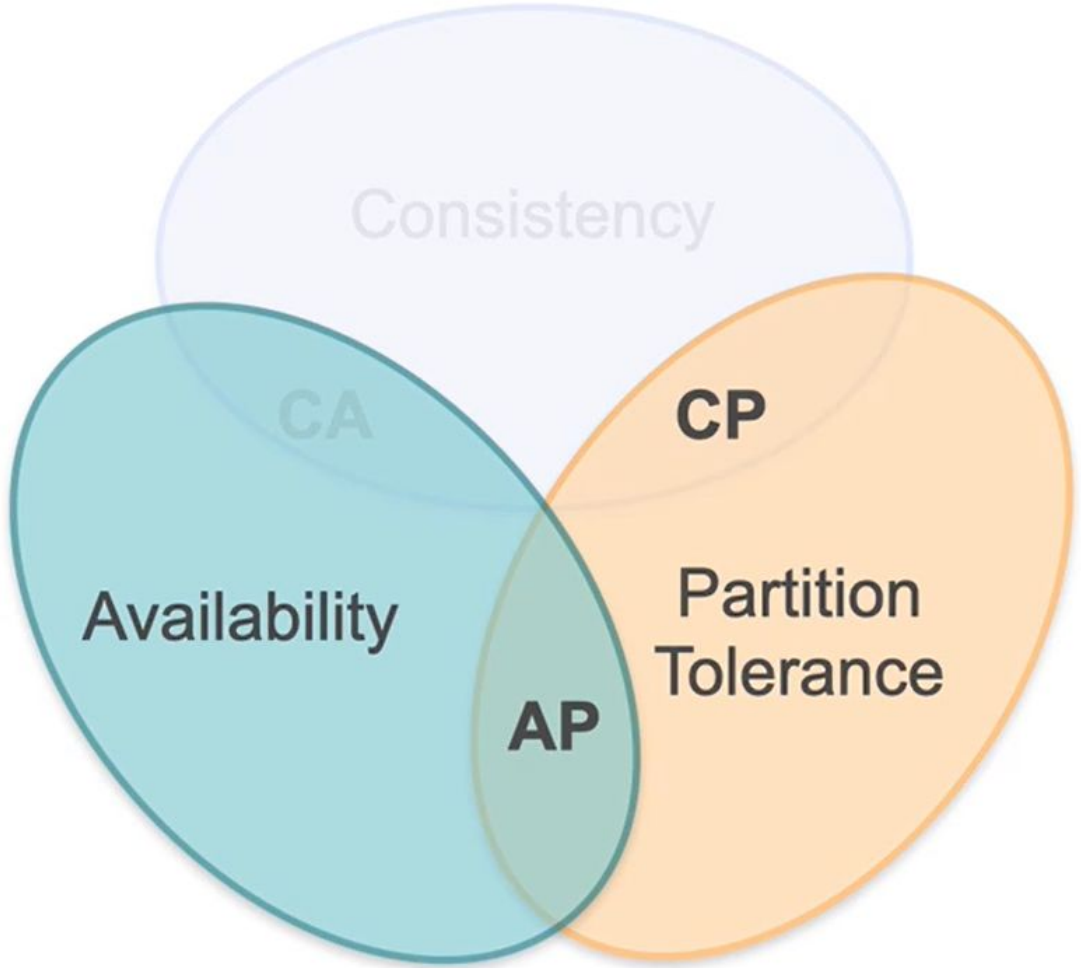


Distributed Storage Considerations – CAP Theorem

Scenario:
Accessing a node that's still being updated

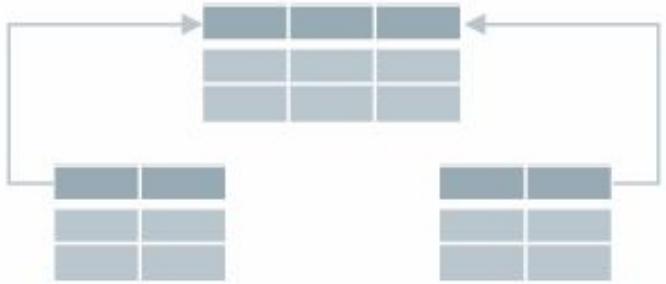
Option 1:
Cancel the request

Option 2:
Proceed with the read operation

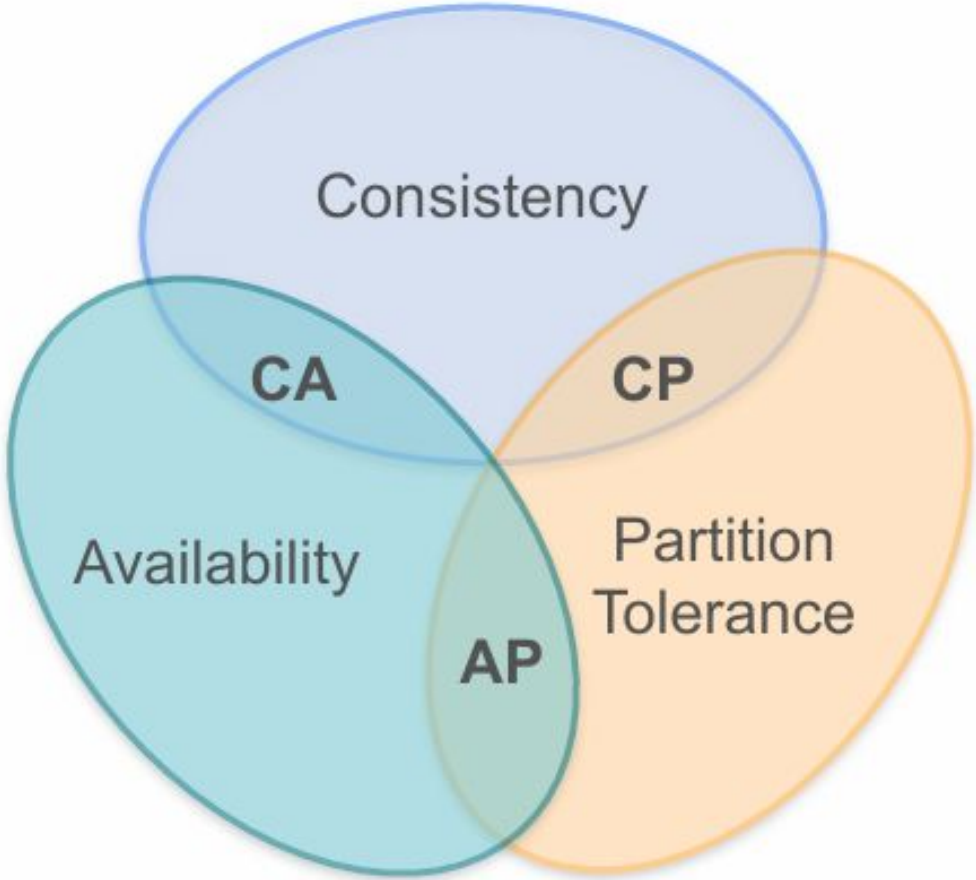


Distributed storage considerations – ACID vs BASE

RDBMS



ACID
compliant



NoSQL Databases

BASE
principles

Distributed storage considerations – ACID vs BASE

ACID
compliant

Atomicity

Consistency

Isolation

Durability

BASE
principles

Basically **A**vailable

Soft-state

Eventual Consistency

Main database (strong consistency):
Think of it as the **master copy** of your sales data.
Read replicas (eventual consistency):
These are **copies of the main database**.
They update from the main database **after a short delay**.

Scenario

Course 1

Data Scientist

Main database instance
(Strong consistency)

Read-replica of the prod database

- Ingest
- Transform
- Store
- Serve

Read Replicas in RDS
(Eventual consistency)

- Track changes in main database
- Update their own data

Amazon RDS

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Comparing Cloud Storage Options

- Object Storage
- File Storage
- Block Storage
- Memory

Object Storage

File Storage

Block Storage

Memory

Flat Structure

Immutability

Object Storage

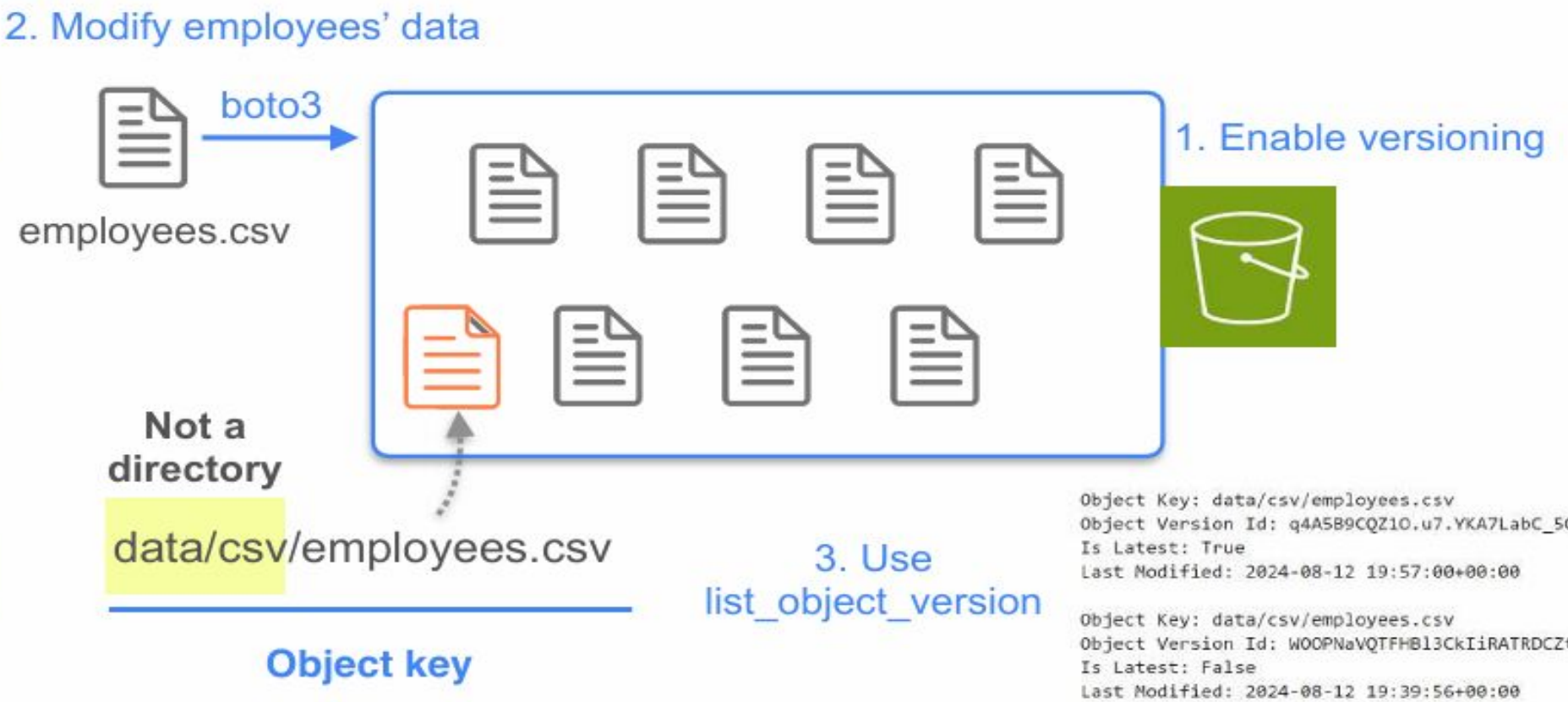
File Storage

Block Storage

Memory

Flat Structure

Immutability



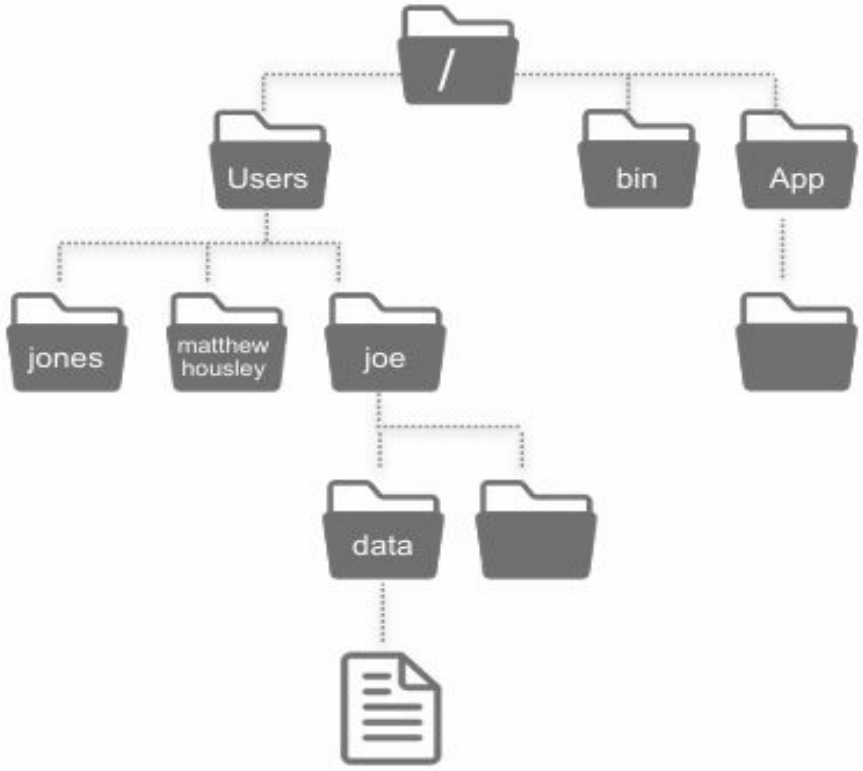
Object Storage

File Storage

Block Storage

Memory

1. Navigate to the “data” directory
2. Explore the directory content and metadata
3. Explore how the data is modified in place



A directory

data/employees.csv

Object Storage

File Storage

Block Storage

Memory

1. Connect to the server
2. Send a file to the server

Server that emulates the behavior of block storage

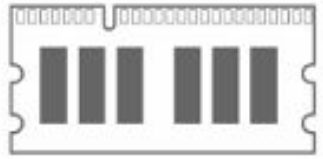
Object Storage

File Storage

Block Storage

Memory

Transferring data from memory is faster than transferring data from disk.



Use the cache-pandas package:

- provides the “timed_LRU_cache” decorator to easily cache in memory pandas DataFrames generated by functions.

```
@timed_lru_cache(seconds=100, maxsize=None)
def read_csv_to_memory(path: str) -> pd.DataFrame:
    """Read CSV function with a cache decorator."""
    return pd.read_csv(path)
```

Compare the time it takes to read the file for the first time with the time it takes to read the same data stored in memory.

Object Storage

File Storage

Block Storage

Memory

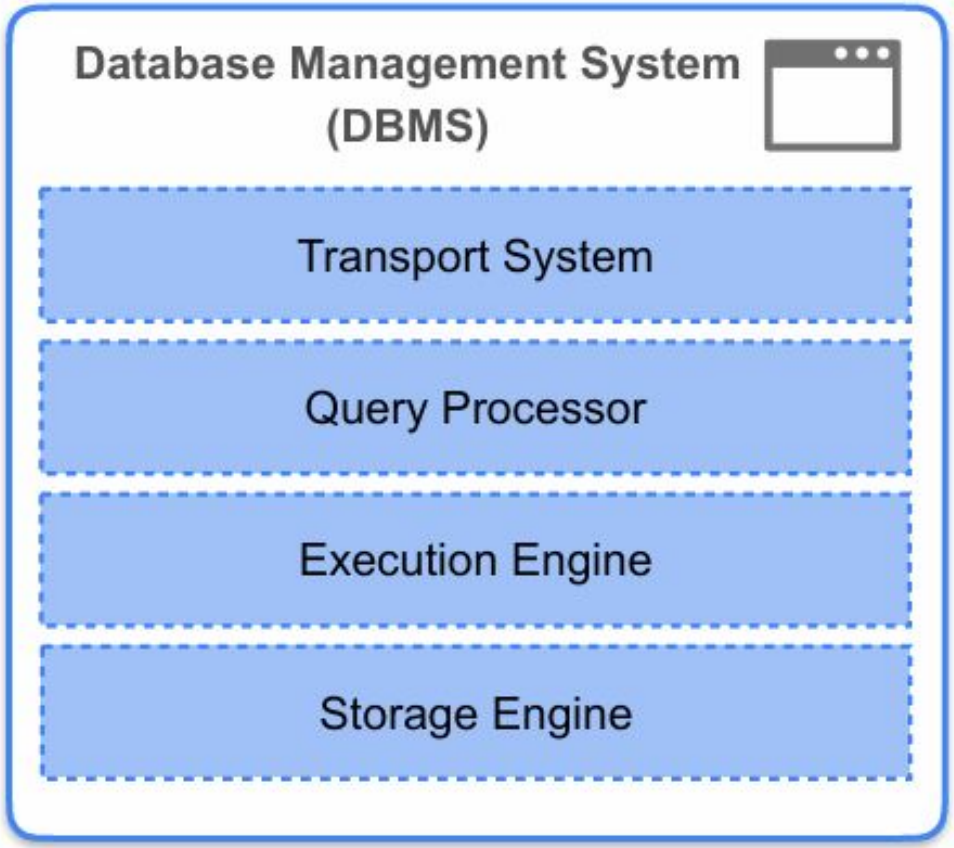
Monitor your memory storage capacity using htop command:

```
  0[|                                                    1.3%] Tasks: 51, 104 thr, 68 kthr; 1 running
  1[                                                     0.0%] Load average: 0.06 0.01 0.00
Mem[||||||||||||||||||||||||||||||||||||||||||||||||||| 773M/1.86G] Uptime: 01:36:48
Swp[|||||||                                            39.2M/488M]

  Main   I/O

    PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
   9600 ec2-user   20   0  218M  4388  3284 R  0.7  0.2  0:00.08 htop
      1 root       20   0  166M 15960  9756 S  0.0  0.8  0:01.59 /usr/lib/systemd/systemd --switched-root --system --deserialize=32
    788 root       20   0 52944 17620 16364 S  0.0  0.9  0:00.72 /usr/lib/systemd/systemd-journald
   1107 root       20   0 30880 10384  7572 S  0.0  0.5  0:00.07 /usr/lib/systemd/systemd-udevd
   1387 systemd-re 20   0 21212 13876 10608 S  0.0  0.7  0:00.11 /usr/lib/systemd/systemd-resolved
   1392 root       16  -4 20224  2300  1632 S  0.0  0.1  0:00.03 /sbin/auditd
   1393 root       16  -4 20224  2300  1632 S  0.0  0.1  0:00.00 /sbin/auditd
   1424 dbus       20   0  8380  3932  3344 S  0.0  0.2  0:00.02 /usr/bin/dbus-broker-launch --scope system --audit
   1425 dbus       20   0  5264  2904  2404 S  0.0  0.1  0:00.19 dbus-broker --log 4 --controller 9 --machine-id ec20b71fb5d5e6c4dae675b053431827 --max
   1426 root       20   0 15296  6444  5612 S  0.0  0.3  0:00.01 /usr/bin/systemd-inhibit --what=handle-suspend-key:handle-hibernate-key --who=noah --w
   1428 root       20   0 81344  3024  2796 S  0.0  0.2  0:00.15 /usr/sbin/irqbalance --foreground
   1429 libstorage 20   0  2756  2072  1908 S  0.0  0.1  0:00.03 /usr/bin/lsmd -d
   1431 root       20   0  161M  6600  5396 S  0.0  0.3  0:37.39 /usr/sbin/rngd -f -x pkcs11 -x nist
   1433 root       20   0 15800  7640  6672 S  0.0  0.4  0:00.02 /usr/lib/systemd/systemd-homed
   1434 root       20   0 17744  9604  7520 S  0.0  0.5  0:00.16 /usr/lib/systemd/systemd-logind
   1436 systemd-ne 20   0  230M  9220  7956 S  0.0  0.5  0:00.08 /usr/lib/systemd/systemd-networkd
   1440 root       20   0 81344  3024  2796 S  0.0  0.2  0:00.00 /usr/sbin/irqbalance --foreground
   1459 root       20   0  2668  1108  1020 S  0.0  0.1  0:00.00 /usr/sbin/acpid -f
   1460 root       20   0  161M  6600  5396 S  0.0  0.3  0:18.47 /usr/sbin/rngd -f -x pkcs11 -x nist
   1461 root       20   0  161M  6600  5396 S  0.0  0.3  0:18.87 /usr/sbin/rngd -f -x pkcs11 -x nist
   1468 root       20   0  274M  3396  2632 S  0.0  0.2  0:00.00 /usr/sbin/gssproxy -D
   1470 root       20   0  274M  3396  2632 S  0.0  0.2  0:00.00 /usr/sbin/gssproxy -D
   1471 root       20   0  274M  3396  2632 S  0.0  0.2  0:00.00 /usr/sbin/gssproxy -D
   1472 root       20   0  274M  3396  2632 S  0.0  0.2  0:00.00 /usr/sbin/gssproxy -D
   1473 root       20   0  274M  3396  2632 S  0.0  0.2  0:00.00 /usr/sbin/gssproxy -D
   1474 root       20   0  274M  3396  2632 S  0.0  0.2  0:00.00 /usr/sbin/gssproxy -D
   1475 root       20   0 1776M 20824  8204 S  0.0  1.1  0:05.19 /usr/bin/containerd
   1516 root       20   0 1776M 20824  8204 S  0.0  1.1  0:01.46 /usr/bin/containerd
   1523 root       20   0 1776M 20824  8204 S  0.0  1.1  0:00.00 /usr/bin/containerd
   1524 root       20   0 1776M 20824  8204 S  0.0  1.1  0:00.46 /usr/bin/containerd
   1534 root       20   0 1776M 20824  8204 S  0.0  1.1  0:00.00 /usr/bin/containerd
F1Help  F2Setup F3Search F4Filter F5Tree  F6SortBy F7Nice -  F8Nice +  F9Kill  F10Quit
```

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

How Databases Store Data

Database Management System



Database Management System

Storage Engine

- Serialization
- Arrangement of data on disk
- Indexing

Modern Storage Engines

- Support the performance characteristics of SSDs
- Handle modern data types and structures
- Offer robust columnar storage support

Average price of products purchased in the USA

```
SELECT AVG(price)
FROM my_table
WHERE country  = "USA"
```

Order ID	Price	Product SKU	Quantity	Customer ID	Store ID	Country
1	40	458650	10	67t	3	Canada
2	23	902348	14	56t	3	Canada
3	45	1255893	12	87q	4	Canada
4	50	456829	13	98q	1	USA
5	34	568298	12	98q	1	USA
6	44	563783	4	67t	1	USA
7	22	234589	5	56u	2	Brazil
8	30	267895	12	78y	3	Canada
9	60	545659	14	13t	5	Mexico

.....

Average price of products purchased in the USA

```
SELECT AVG(price)
FROM my_table
WHERE country  = "USA"
```

Order ID	Price	Product SKU	Quantity	Customer ID	Store ID	Country
1	40	458650	10	67t	3	Canada
2	23	902348	14	56t	3	Canada
3	45	1255893	12	87q	4	Canada
4	50	456829	13	98q	1	USA
5	34	568298	12	98q	1	USA
6	44	563783	4	67t	1	USA
7	22	234589	5	56u	2	Brazil
8	30	267895	12	78y	3	Canada
9	60	545659	14	13t	5	Mexico
.....						

Index

A data structure that helps you efficiently locate data

scan all rows

Average price of products purchased in the USA

```
SELECT AVG(price)
FROM my_table
WHERE country  = "USA"
```

Order ID	Price	Product SKU	Quantity	Customer ID	Store ID	Country
1	40	458650	10	67t	3	Canada
2	23	902348	14	56t	3	Canada
3	45	1255893	12	87q	4	Canada
4	50	456829	13	98q	1	USA
5	34	568298	12	98q	1	USA
6	44	563783	4	67t	1	USA
7	22	234589	5	56u	2	Brazil
8	30	267895	12	78y	3	Canada
9	60	545659	14	13t	5	Mexico

Scanning all rows: O(n)

Binary search on rows: O(log n)

Index

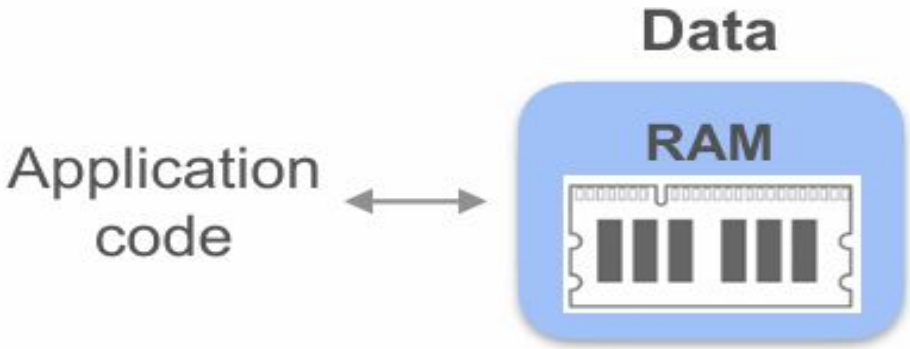
A data structure that helps you efficiently locate data

Use binary search to locate the USA rows

Index table

Country	Row Address
Brazil	###
Canada	###
Canada	###
Canada	###
Canada	###
Mexico	###
USA	###
USA	###
USA	###

In-Memory Storage Systems



- Excellent transfer speed and low latency
- Volatile
- Used to present data for ultra-fast retrieval:
  - Caching applications
  - Real-time bidding
  - Gaming leaderboards

1. Memcached

- Key-value store to cache database query results or API calls
- Used when it's acceptable for data to be lost

2. Redis

- Key-value store that supports more complex data types
- Supports high-performance applications that can tolerate minor data loss

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Row vs Column Storage

Row-Oriented Storage

Order ID	Price	Product SKU	Quantity	Customer ID
1	40	45865	10	67t
2	23	90234	14	56t
3	45	12558	12	87q
4	50	45682	13	98q

Stores data row by row

Physical Storage

1	40	45865	10	67t	2	23	90234	14	56t	…	4	50	45682	13	98q

bytes representing the 1st row | bytes representing the 2nd row | bytes representing the last row

Row-Oriented Storage

Order ID	Price	Product SKU	Quantity	Customer ID
1	40	45865	10	67t
2	23	90234	14	56t
3	45	12558	12	87q
4	50	45682	13	98q

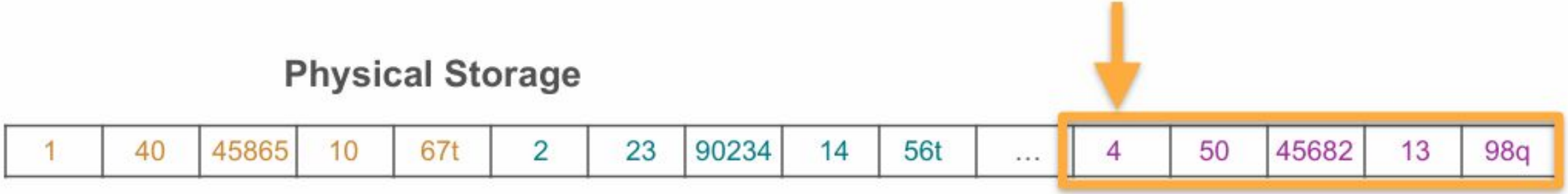
← Locate this order

Row Storage is perfect for OLTP

Perform read and write operations with low latency

↓ Stores data row by row

Physical Storage



Row-Oriented Storage

Order ID	Price	Product SKU	Quantity	Customer ID
1	40	45865	10	67t
2	23	90234	14	56t
3	45	12558	12	87q
4	50	45682	13	98q

Stores data row by row

Analytical queries focus on summarizing or aggregating columns

- Total revenue?
- Most popular product?
- Average quantity?

Physical Storage

1	40	45865	10	67t	2	23	90234	14	56t	…	4	50	45682	13	98q

Row-Oriented Storage

Order ID	Price	Product SKU	Quantity	Customer ID	...
1	40	45865	10	67t	...
2	23	90234	14	56t	...
3	45	12558	12	87q	...
4	50	45682	13	98q	...
...	...	...	...	...	...

```
SELECT SUM(price)
FROM my_table
```

1 million rows 30 columns 100 bytes per entry

Physical Storage

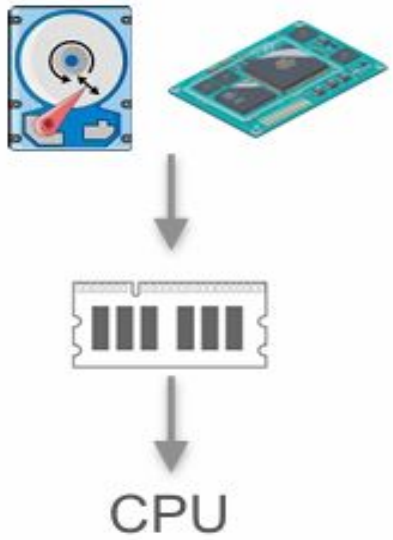
1	40	45865	10	67t	2	23	90234	14	56t	...	4	50	45682	13	98q	...

Row-Oriented Storage

Order ID	Price	Product SKU	Quantity	Customer ID	...
1	40	45865	10	67t	...
2	23	90234	14	56t	...
3	45	12558	12	87q	...
4	50	45682	13	98q	...
...	...	...	...	...	...

1 million rows 30 columns 100 bytes per entry

```
SELECT SUM(price)
FROM my_table
```



Physical Storage

1	40	45865	10	67t	2	23	90234	14	56t	...	4	50	45682	13	98q

Row-Oriented Storage

Order ID	Price	Product SKU	Quantity	Customer ID	...
1	40	45865	10	67t	...
2	23	90234	14	56t	...
3	45	12558	12	87q	...
4	50	45682	13	98q	...
...	...	...	...	...	...

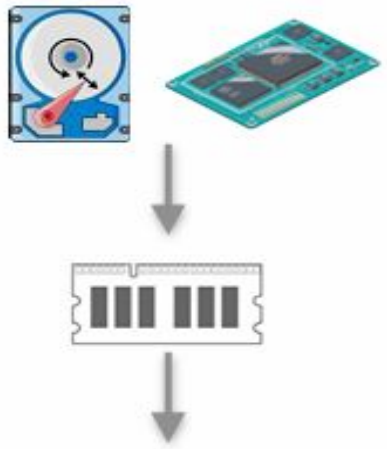
Total size=rows×columns×size per entry

1 million rows x 30 columns x 100 bytes per entry = 3 GB

Data transfer speed: 200 MB/s

Total transfer time? $\frac{\text{3GB or 3000 MB}}{\text{200 MB/s}}$ = 15 s

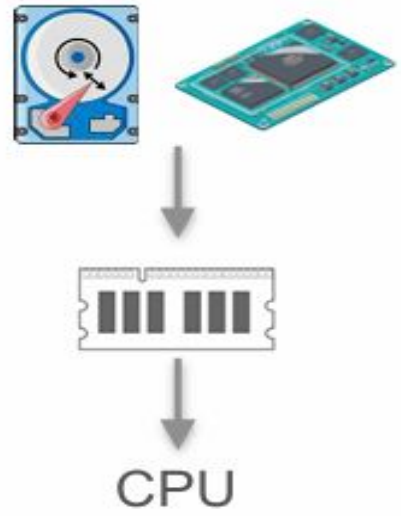
```
SELECT SUM(price)
FROM my_table
```



Row-Oriented Storage

Order ID	Price	Product SKU	Quantity	Customer ID	...
1	40	45865	10	67t	...
2	23	90234	14	56t	...
3	45	12558	12	87q	...
4	50	45682	13	98q	...
...	...	...	...	...	...

```
SELECT SUM(price)
FROM my_table
```



1 billion rows x 30 columns x 100 bytes per entry = 3000 GB

Data transfer speed: 200 MB/s

Total transfer time? $\frac{3000\ GB}{200\ MB/s}$ = 4 hours !

Column-Oriented Storage

Order ID	Price	Product SKU	Quantity	Customer ID	…
1	40	45865	10	67t	…
2	23	90234	14	56t	…
3	45	12558	12	87q	…
4	50	45682	13	98q	…
…	…	…	…	…	…

Stores data
Column by column

Physical Storage

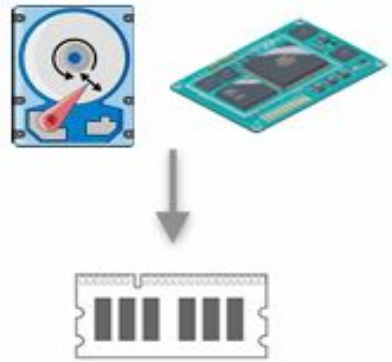
1	2	3	4	40	23	45	50	45865	90234	12558	45682	…

bytes representing 1st column bytes representing 2nd column bytes representing 3rd column

Column-Oriented Storage — Suitable for OLAP systems!

Order ID	Price	Product SKU	Quantity	Customer ID	...
1	40	45865	10	67t	...
2	23	90234	14	56t	...
3	45	12558	12	87q	...
4	50	45682	13	98q	...
...	...	...	...	...	...

```
SELECT SUM(price)
FROM my_table
```



1 billion rows × 30 columns × 100 bytes per entry = 100 GB

Data transfer speed: 200 MB/s

Total transfer time? $\frac{\text{100 GB or 100,000 MB}}{\text{200 MB/s}}$ = 8.33 minutes

Row-oriented Storage

Transfer 1 billion rows from disk to memory

4 hours

Column-Oriented Storage

Order ID	Price	Product SKU	Quantity	Customer ID	…
1	40	45865	10	67t	…
2	23	90234	14	56t	…
3	45	12558	12	87q	…
4	50	45682	13	98q	…
…	…	…	…	…	…

Terrible for transactional workloads!

Stores data Column by column

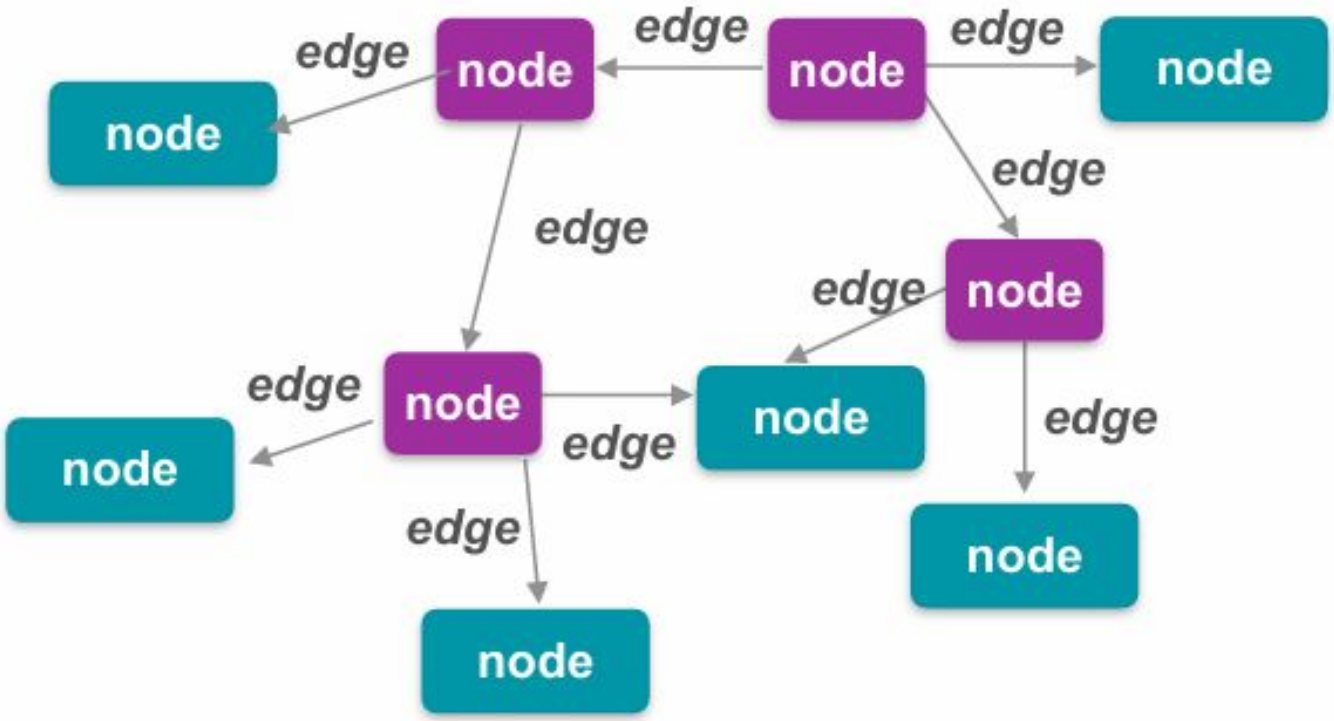
Physical Storage

Deserialize the column, modify it, then write it back to storage

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Graph Databases

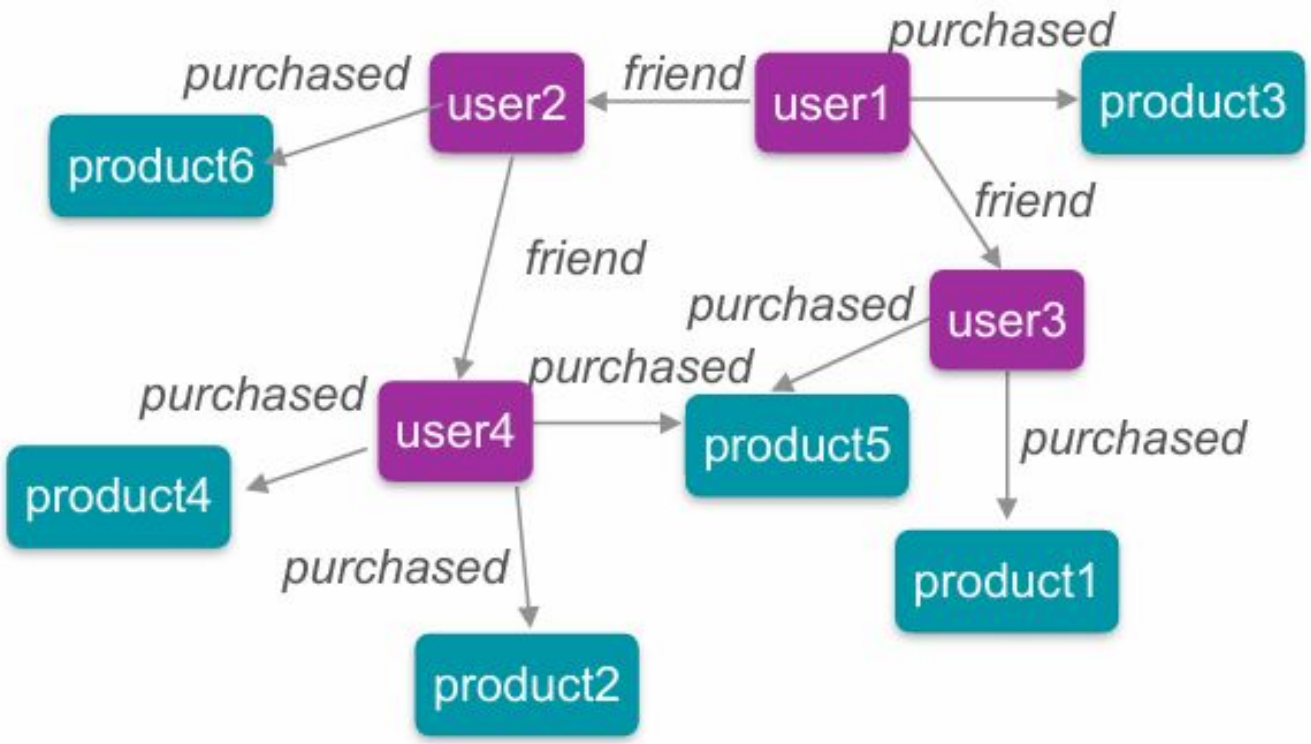
Graph Database



- Nodes represent data items
- Edges represent connection between the data items
- Graph databases model complex connections between data entities

Graph Database

Relationships are first-class citizens



Relational database

purchase

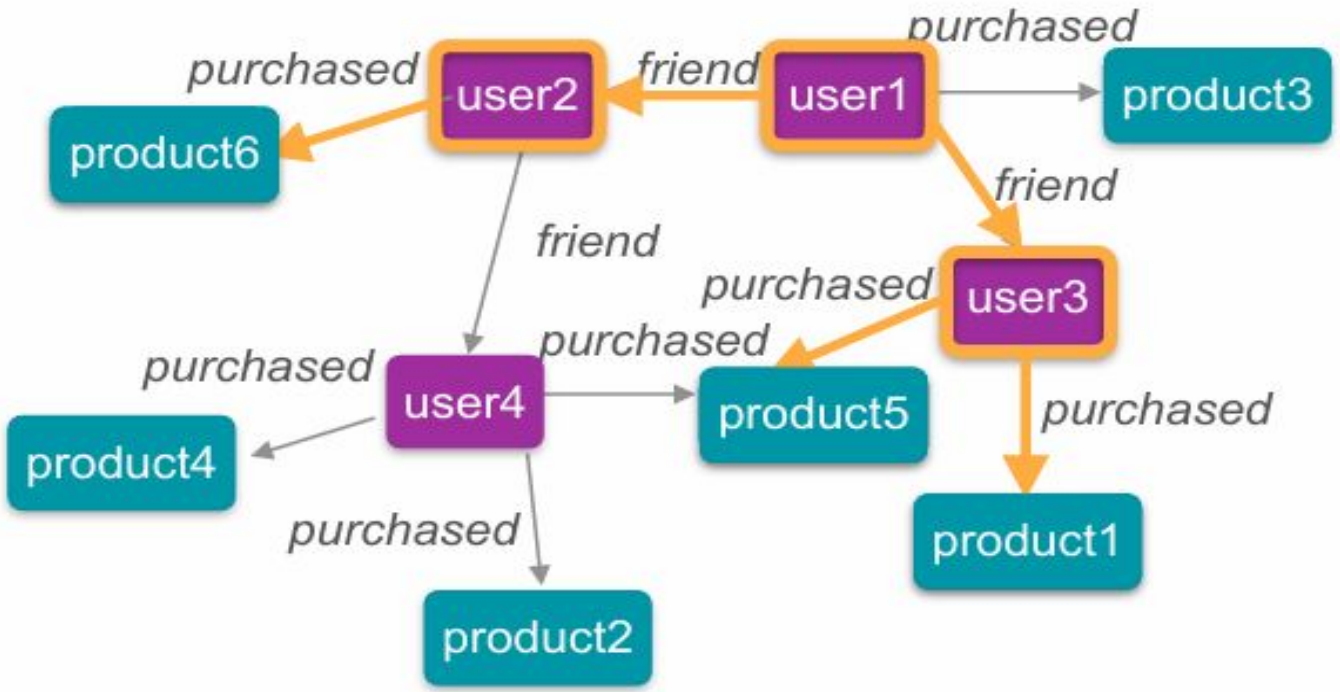
user	product
user1	product3
user2	product6
user3	product1
user3	product5
user4	product5
user4	product4
user4	product2

friendship

user	friend
user1	user3
user1	user2
user2	user4

Querying Data

Traverse the graph to query relationships



Recommendations for user1

Relational database

purchase

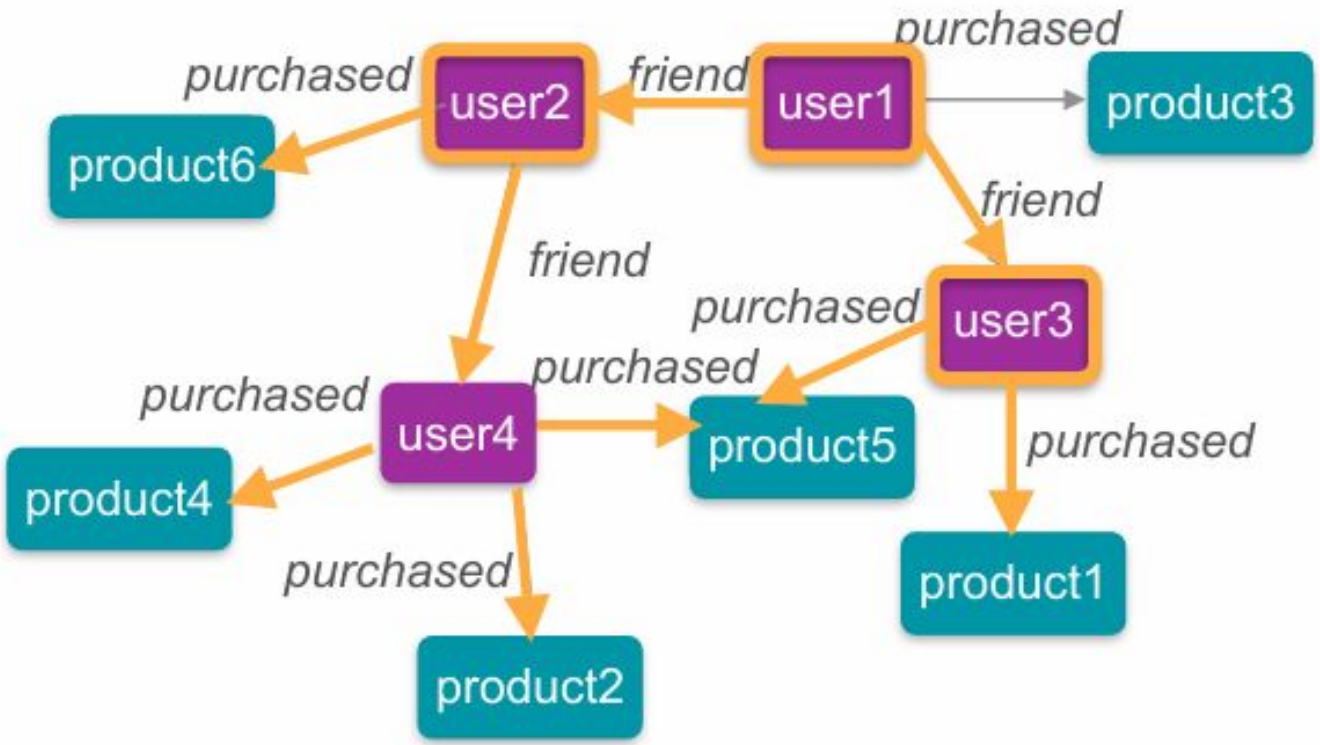
user	product
user1	product3
user2	product6
user3	product1
user3	product5
user4	product5
user4	product4
user4	product2

friendship

user	friend
user1	user3
user1	user2
user2	user4

Querying Data

Traverse the graph to query relationships



Recommendations for user1: product1, product5, product6

Relational database

purchase

user	product
user1	product3
user2	product6
user3	product1
user3	product5
user4	product5
user4	product4
user4	product2

friendship

user	friend
user1	user3
user1	user2
user2	user4

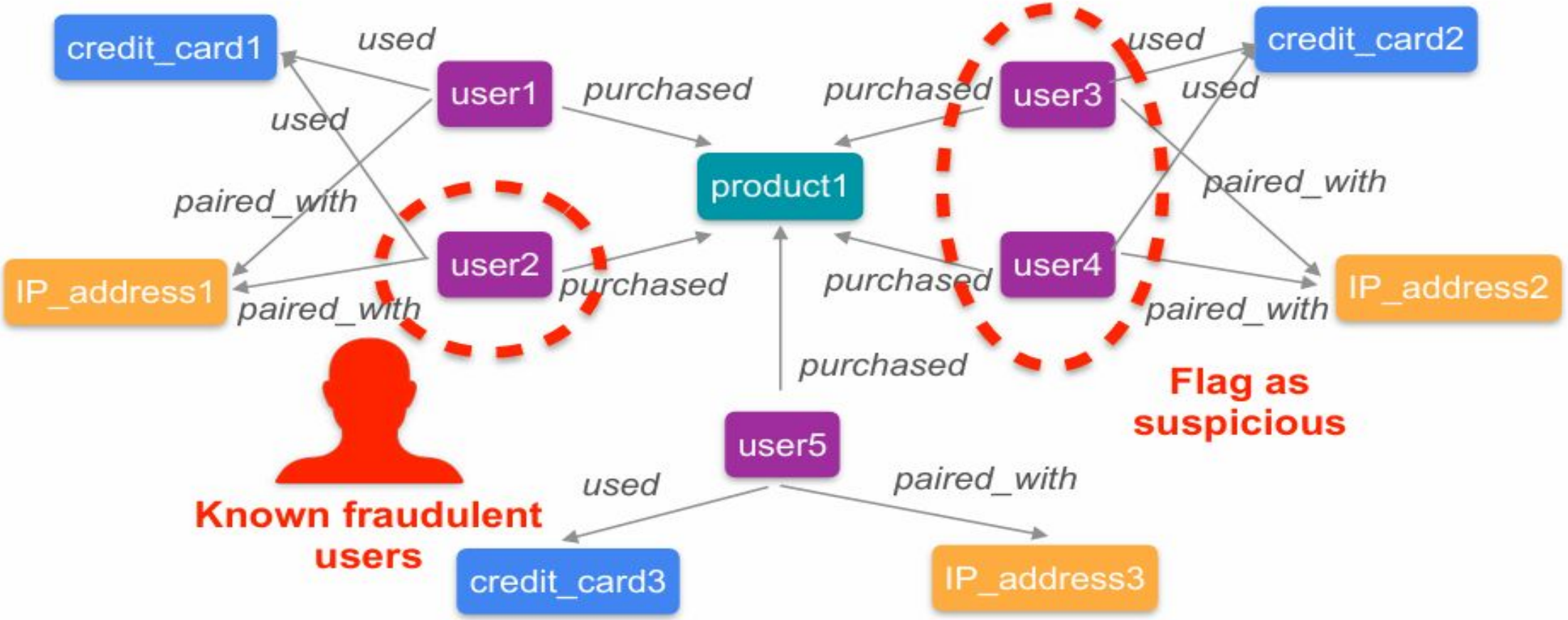
Less efficient in querying complex relationships!

```
SELECT DISTINCT purchase.product
FROM friendship
JOIN purchase  ON friendship.friend = purchase.user
WHERE friendship.user = 'user1'
```

Graph Database - Use Cases

- Recommending products
- Modeling social networks
- Representing network and IT operations
- Simulating supply chains logistics
- Tracing data lineage

Use Case - Fraud Detection

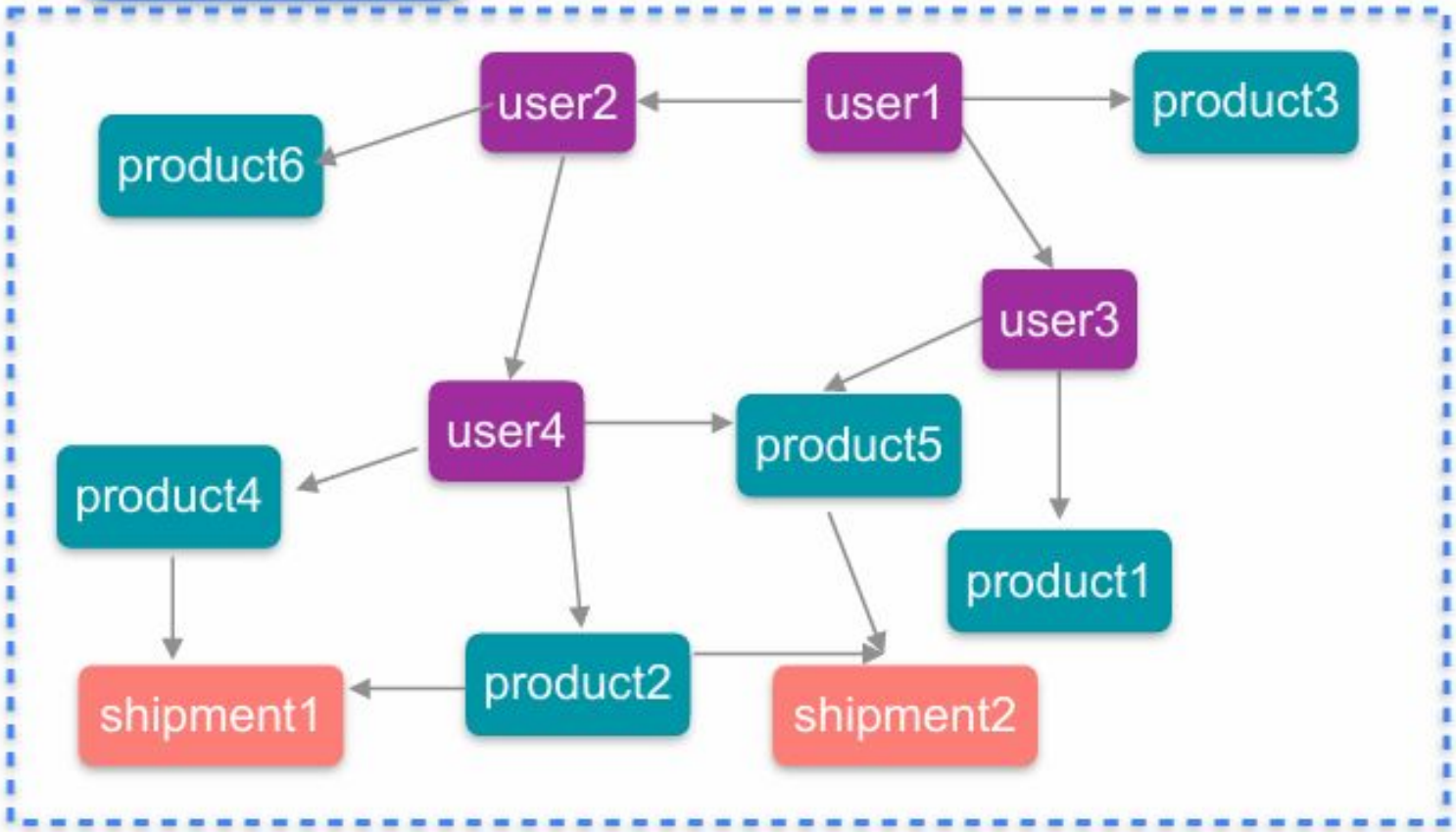


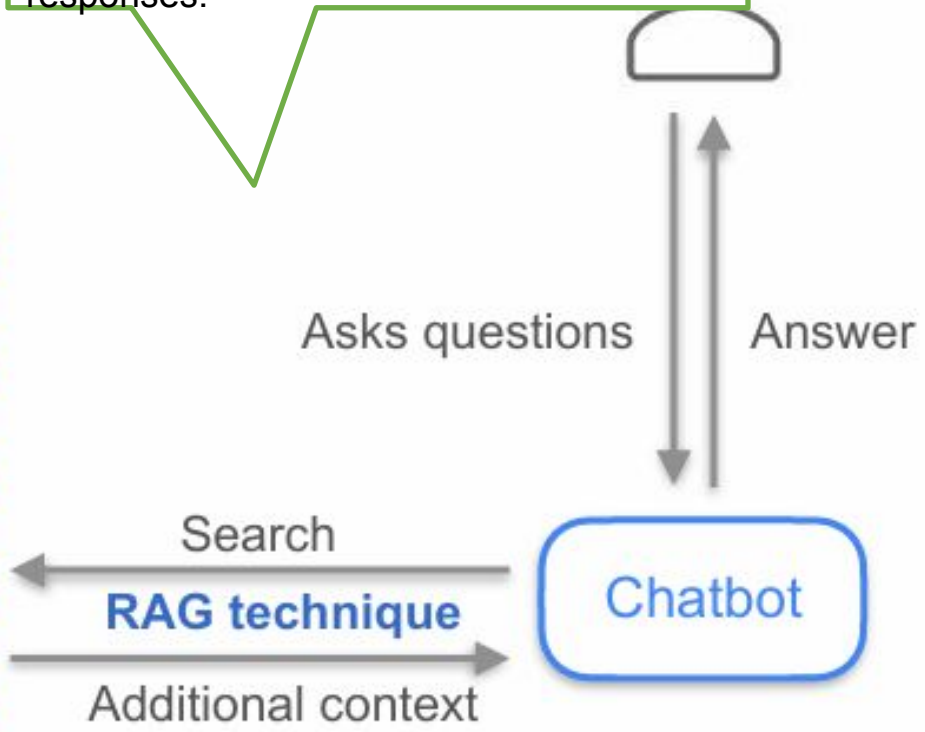
Use Case - Knowledge Graph

A knowledge graph connects e-commerce entities like customers, products, and orders, enabling chatbots to find relevant relationships quickly. Using RAG, the LLM accesses this fresh, company-specific data to provide accurate and up-to-date responses.

Knowledge Graph

Connect diverse data from disparate sources





Graph Databases

Examples of Graph Databases

neo4j

ArangoDB

Amazon Neptune

Examples of Graph Query Language

Cypher

Gremlin

SparQL

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Vector Databases

Vector data

Consists of numerical values arranged in an array

0.5 0.5 0.7 0.5 0.3 0.1 0.7 0.2
0.4 0.8 0.9 0.1 0.3 0.1 0.4 0.2
0.3 0.5 0.7 0.8 0.3 0.1 0.6 0.2

Vector embeddings

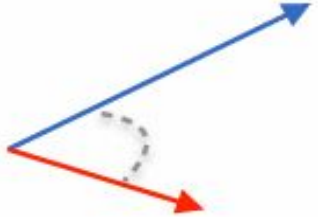
Capture semantic meaning of an item, like a text document or image

- Can convert an entire database of docs or text into embeddings
- Embeddings help you more efficiently find and retrieve similar items
- Example: Finding similar text
  - Compute embeddings for the query item
  - Database returns similar vectors (based on closeness)

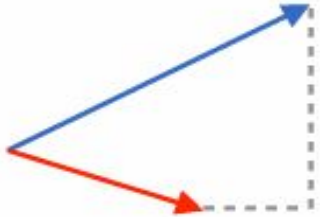
Distance Metric

Vector database uses a distance metric to find similar vectors

Euclidean distance



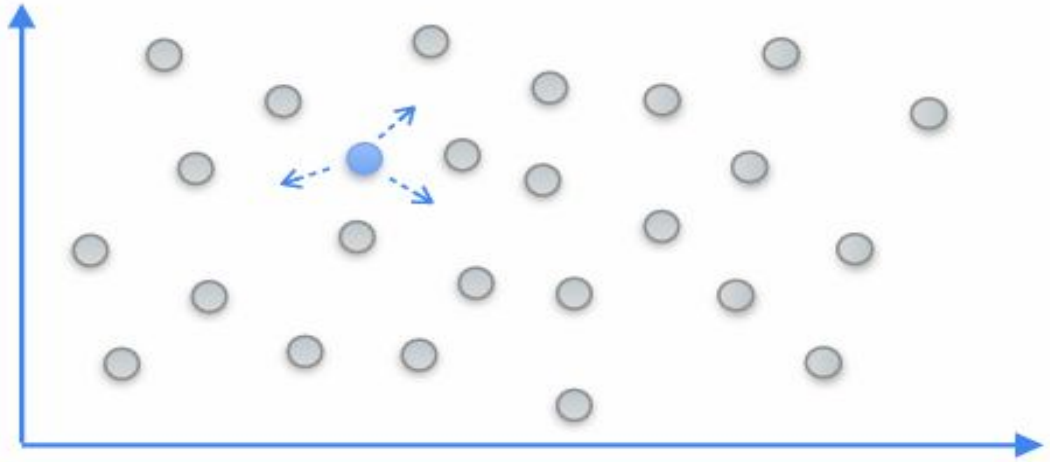
Cosine distance



Manhattan distance

Similarity Search - Popular Algorithm

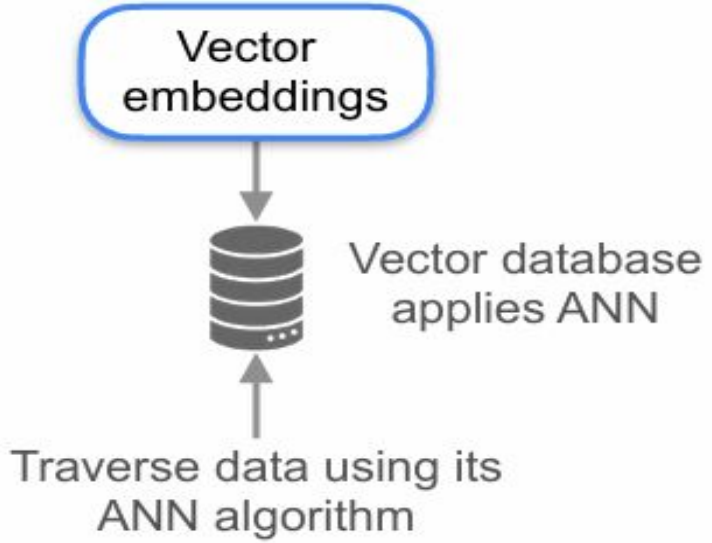
K-nearest neighbors (KNN)



- Calculates distance to all vector embeddings
- Becomes inefficient when the data size increases
- Suffers from the curse of dimensionality

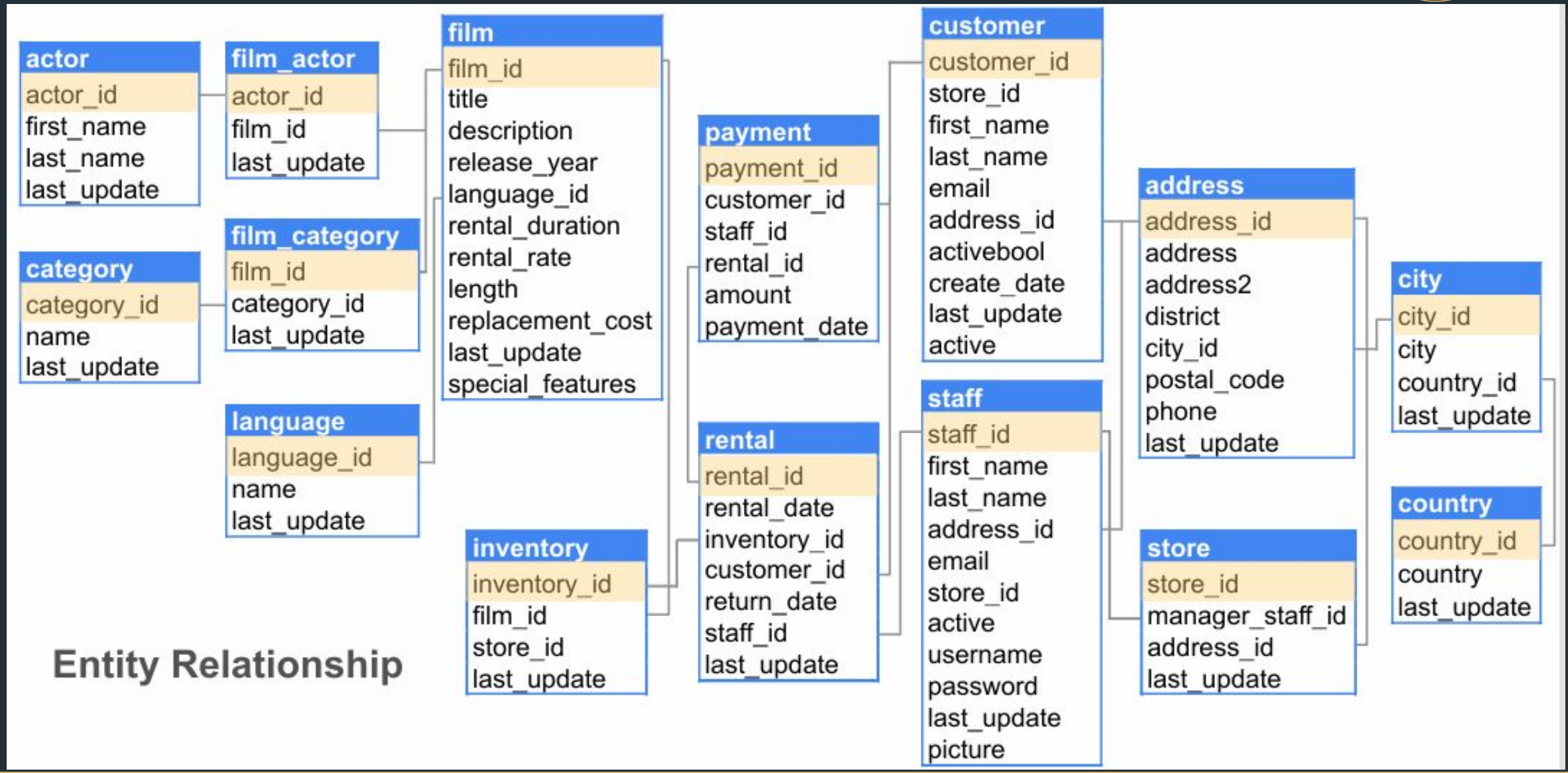
ANN (Approximate Nearest Neighbors)

- Find a good guess for the nearest neighbors
- More efficient than K-NN
- Vector databases are built to support ANN algorithms

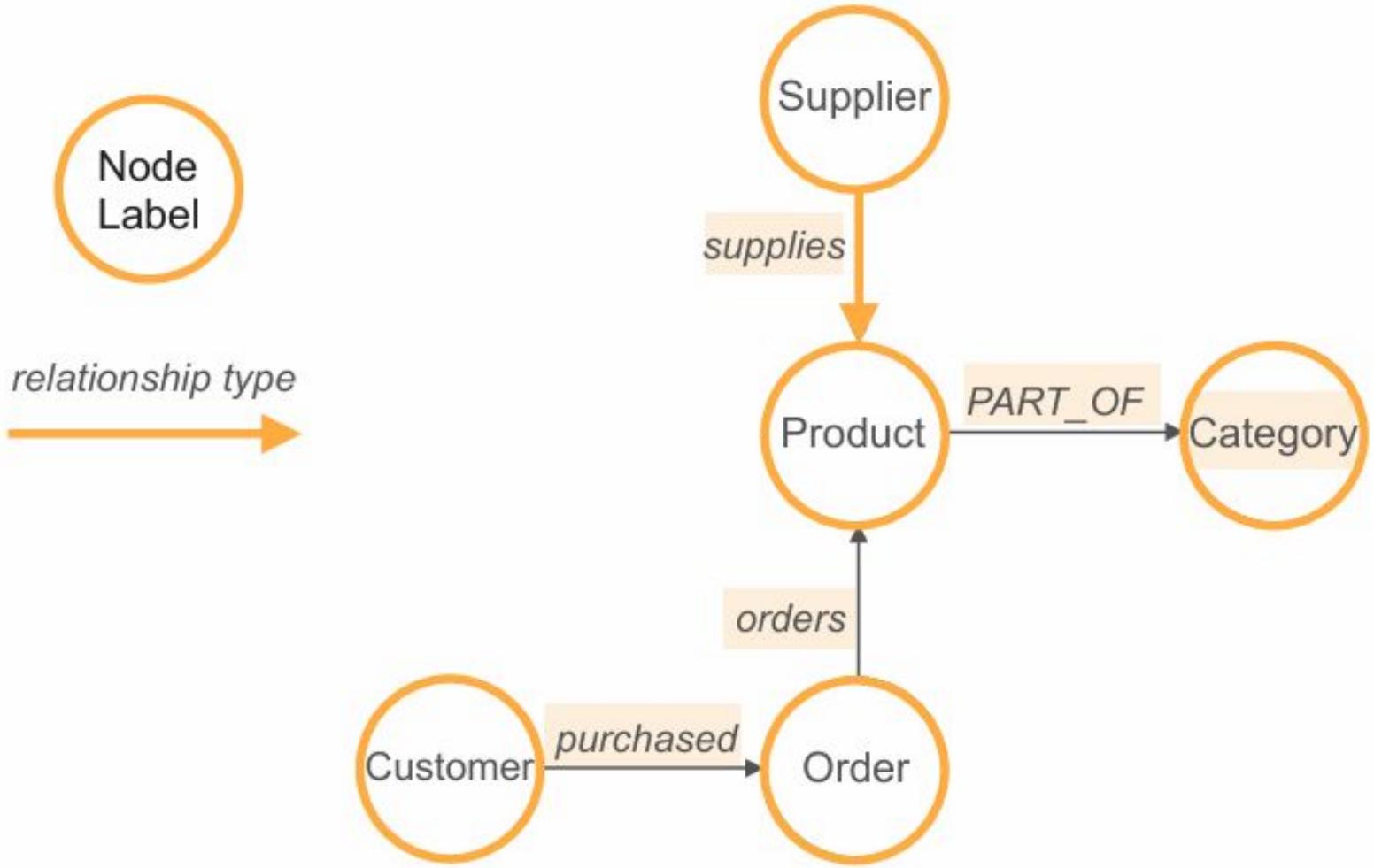


RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

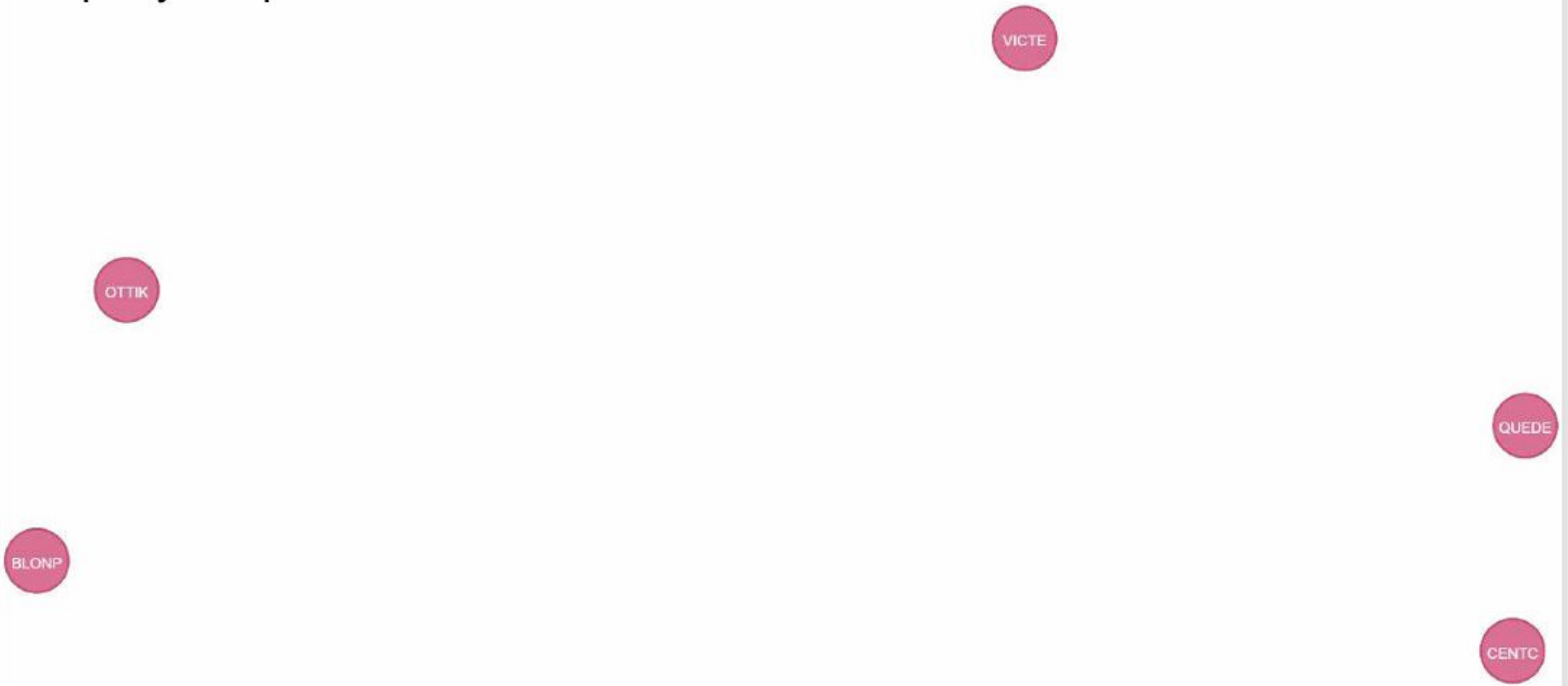
Neo4j Graph Database & Cypher Query Language


actor
actor_id
first_name
last_name
last_update
film_actor
actor_id
film_id
last_update
film
film_id
title
description
release_year
language_id
rental_duration
rental_rate
length
replacement_cost
last_update
special_features
customer
customer_id
store_id
first_name
last_name
email
address_id
activebool
create_date
last_update
active
category
category_id
name
last_update
film_category
film_id
category_id
last_update
payment
payment_id
customer_id
staff_id
rental_id
amount
payment_date
address
address_id
address
address2
district
city_id
postal_code
phone
last_update
city
city_id
city
country_id
last_update
language
language_id
name
last_update
rental
rental_id
rental_date
inventory_id
customer_id
return_date
staff_id
last_update
staff
staff_id
first_name
last_name
address_id
email
store_id
active
username
password
last_update
picture
inventory
inventory_id
film_id
store_id
last_update
store
store_id
manager_staff_id
address_id
last_update
country
country_id
country
last_update
Entity Relationship

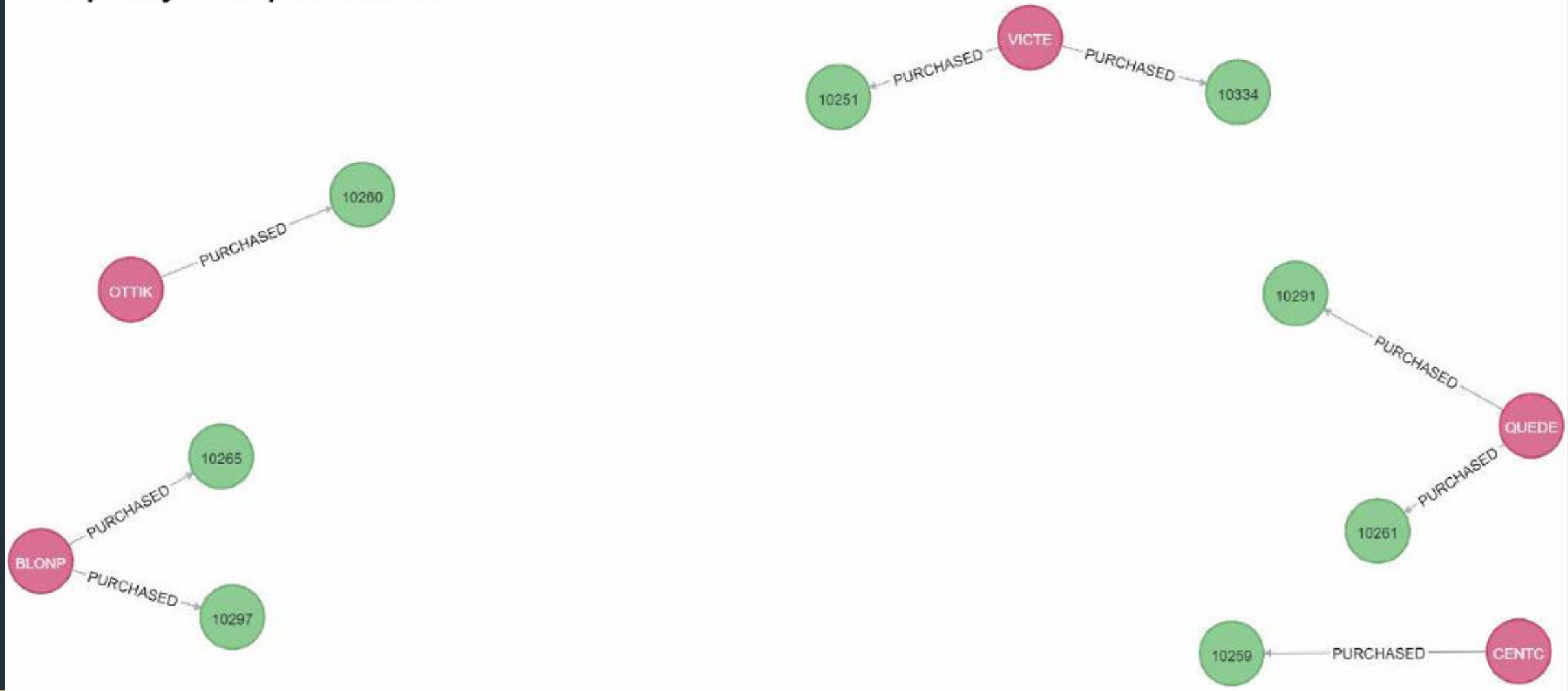
Property Graph Model



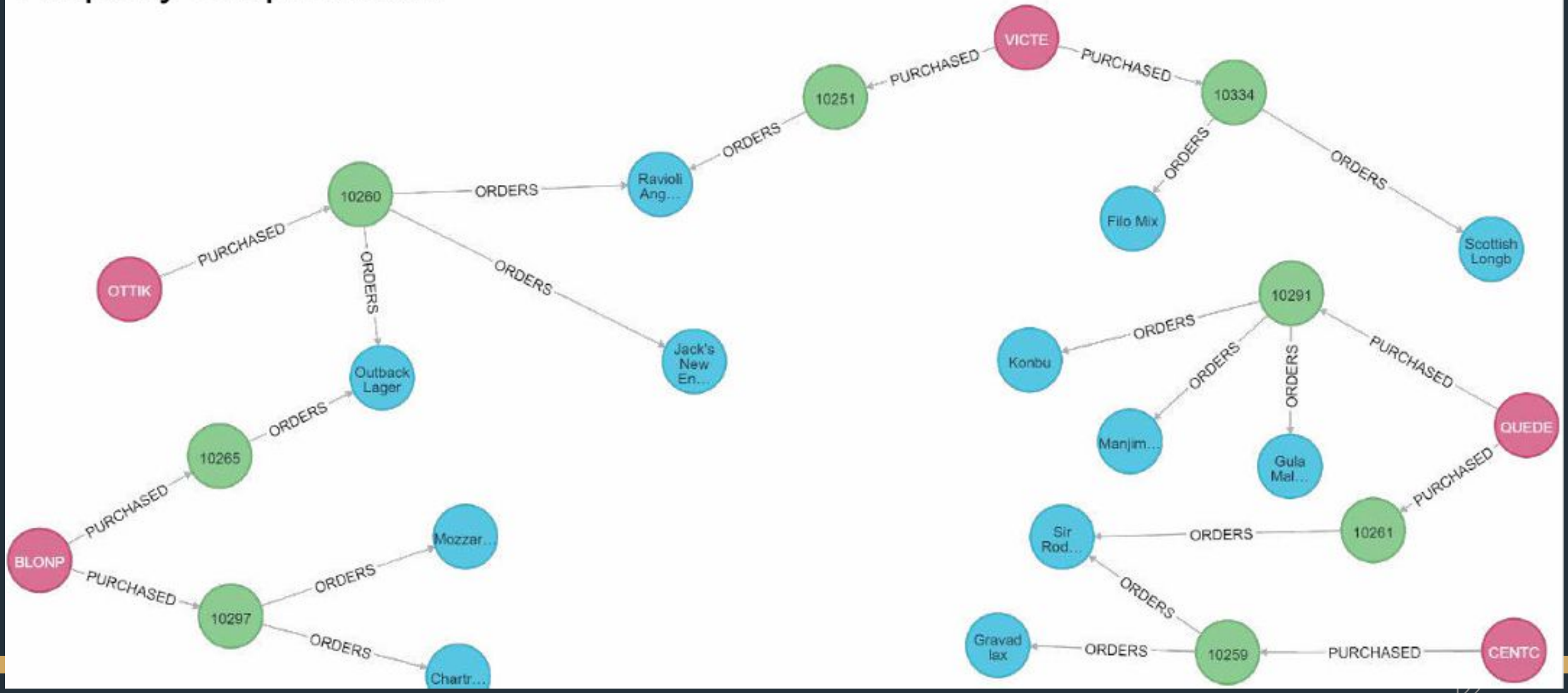
Property Graph Model



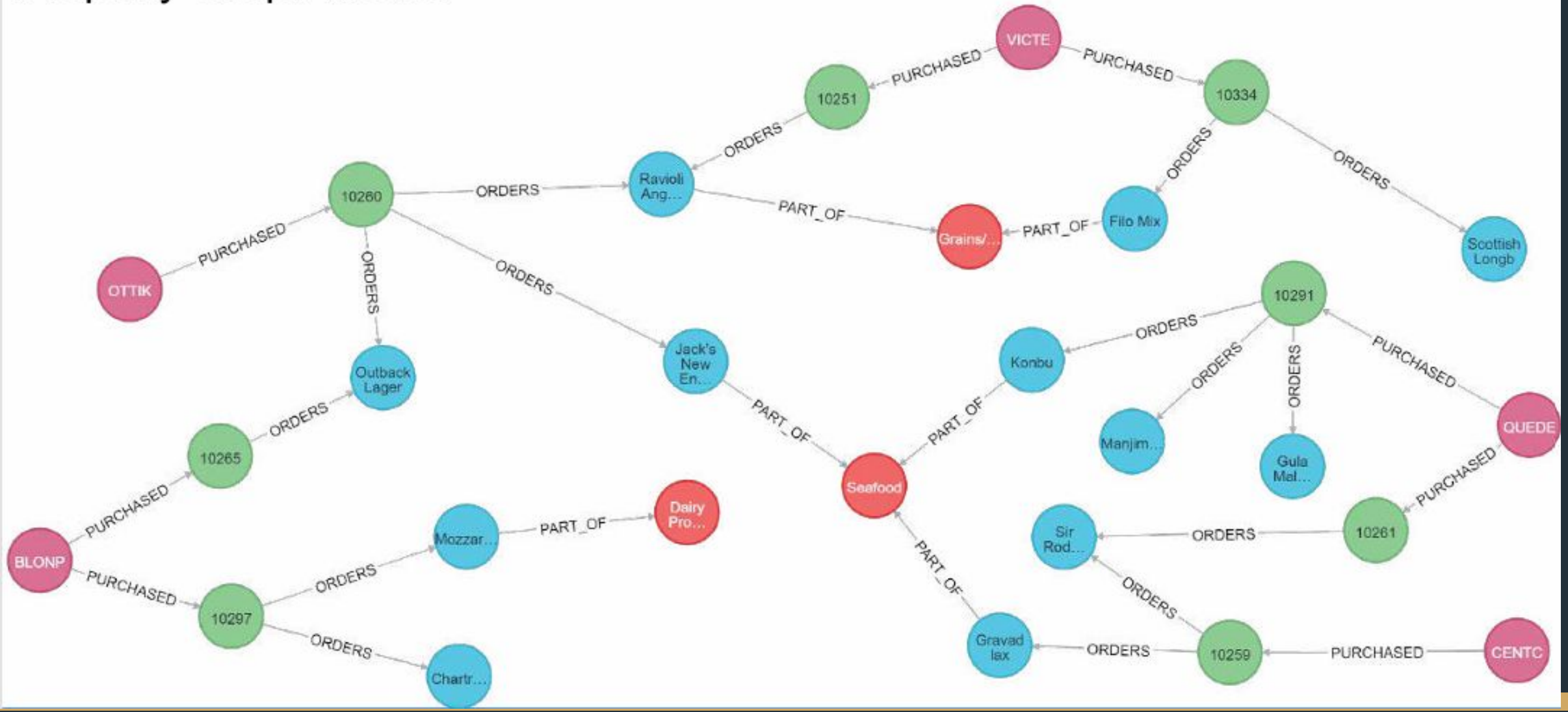
Property Graph Model



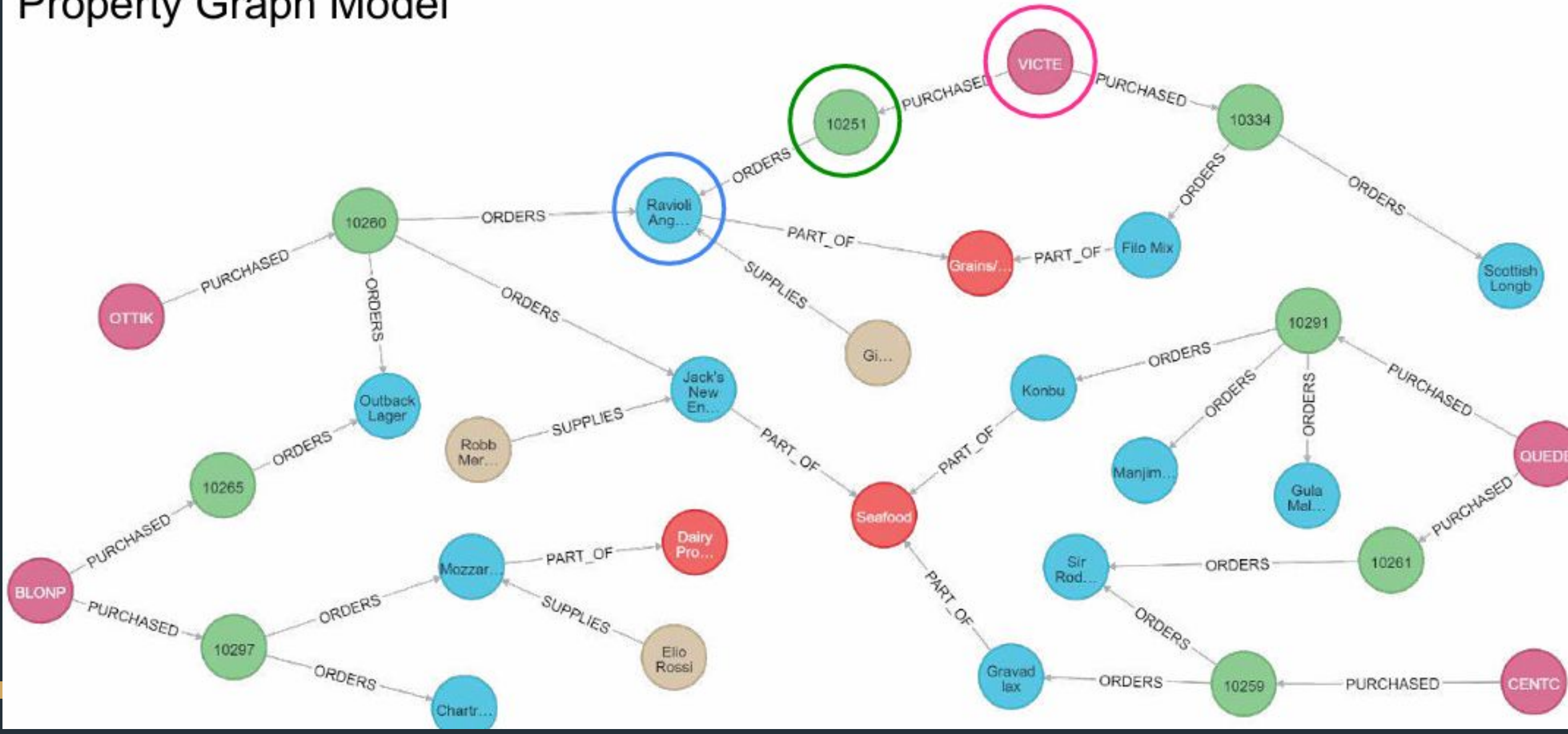
Property Graph Model



Property Graph Model



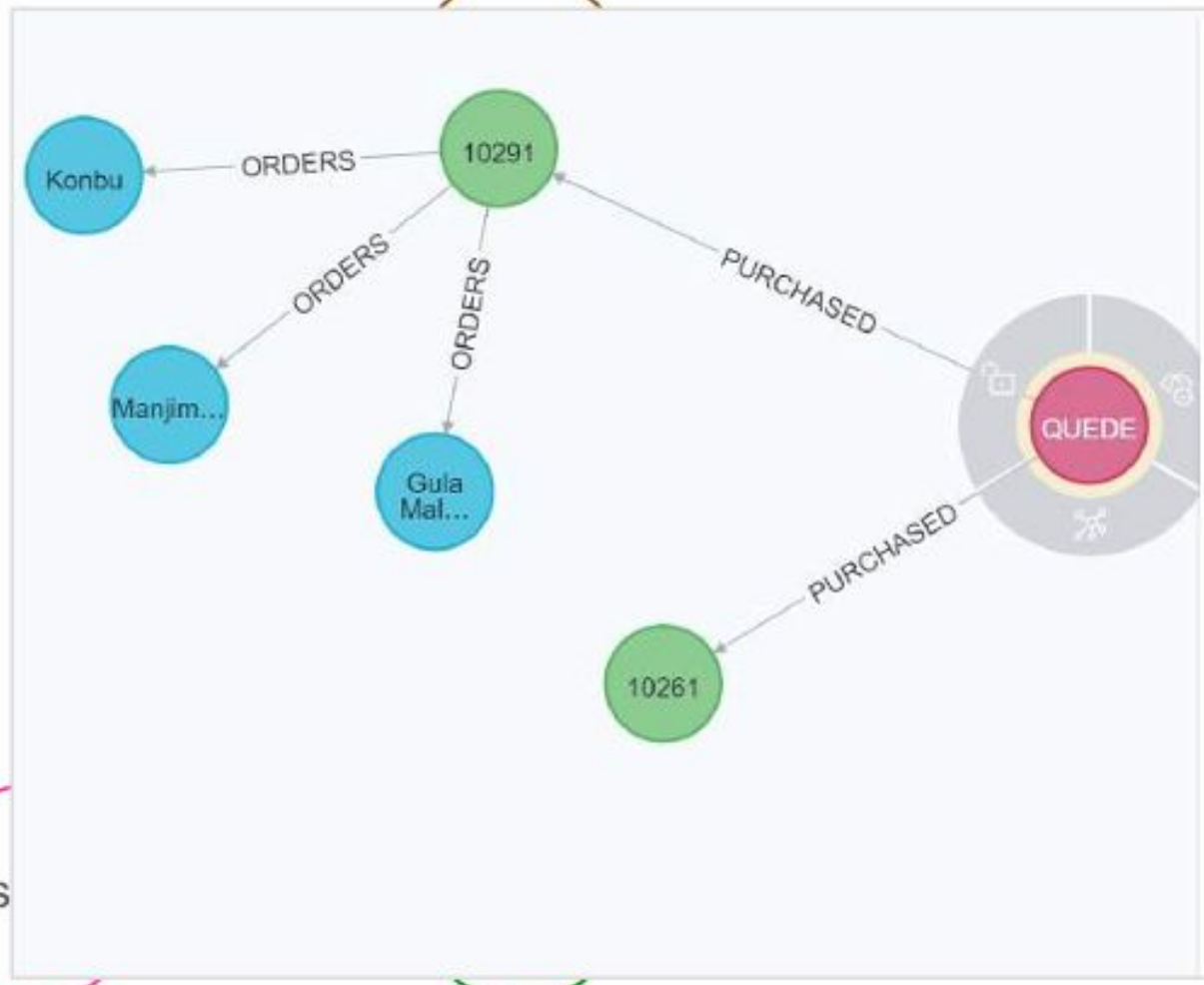
Property Graph Model



Property Graph Model

Customer Properties

address
city
companyName
contactName
contactTitle
country
customerID

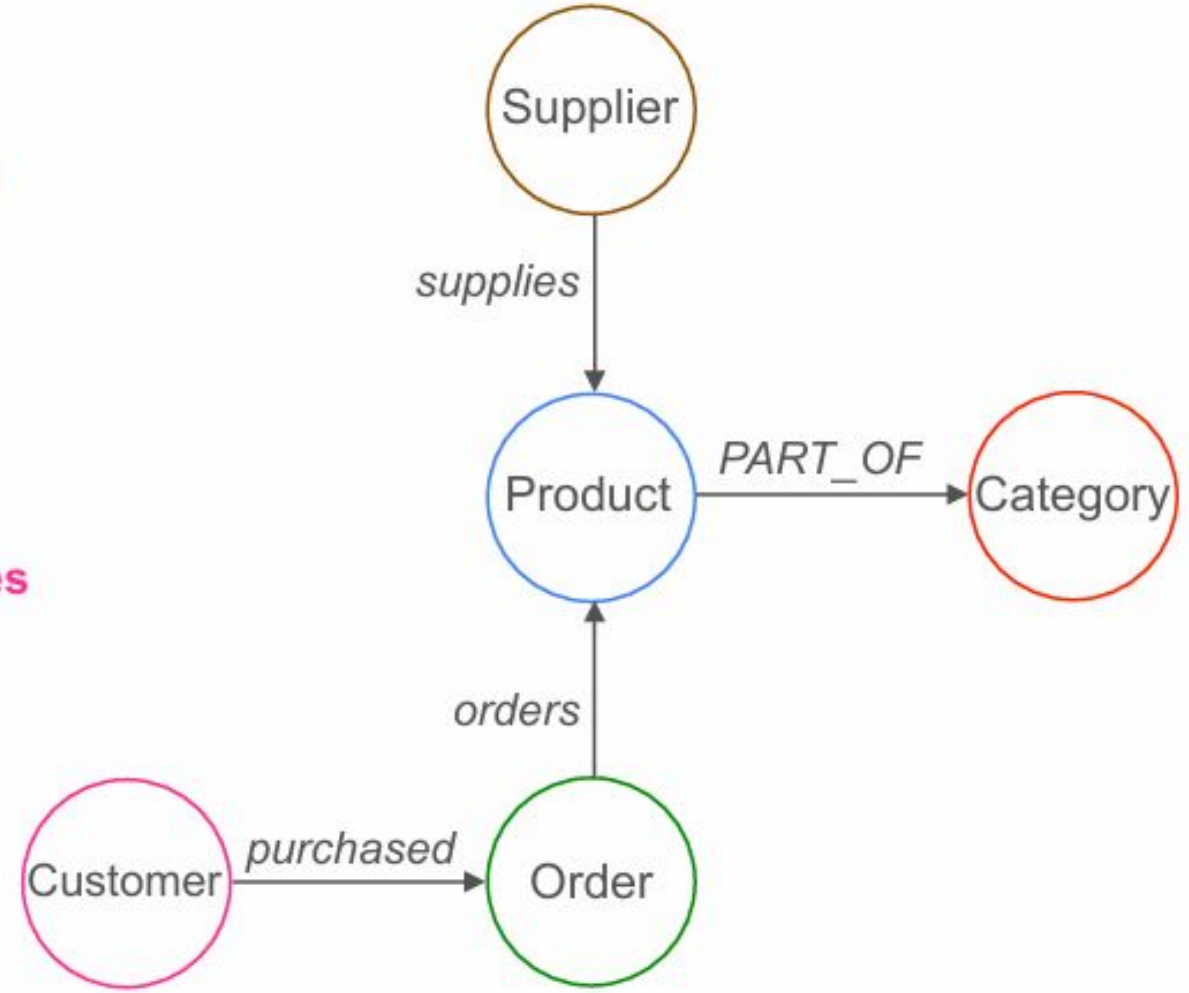


Node properties

Customer

<elementId>	4:fe762cfa-5074-4140-af0c-da262df58724:577
<id>	577
address	Rua da Panificadora, 12
city	Rio de Janeiro
companyNam e	Que Delícia
contactName	Bernardo Batista
contactTitle	Accounting Manager
country	Brazil
customerID	QUEDE
fax	(21) 555-4545
phone	(21) 555-4252
postalCode	02389-673
region	RJ

Property Graph Model



Product Properties
productID
productName
unitPrice
unitsInStock
unitesOnOrder

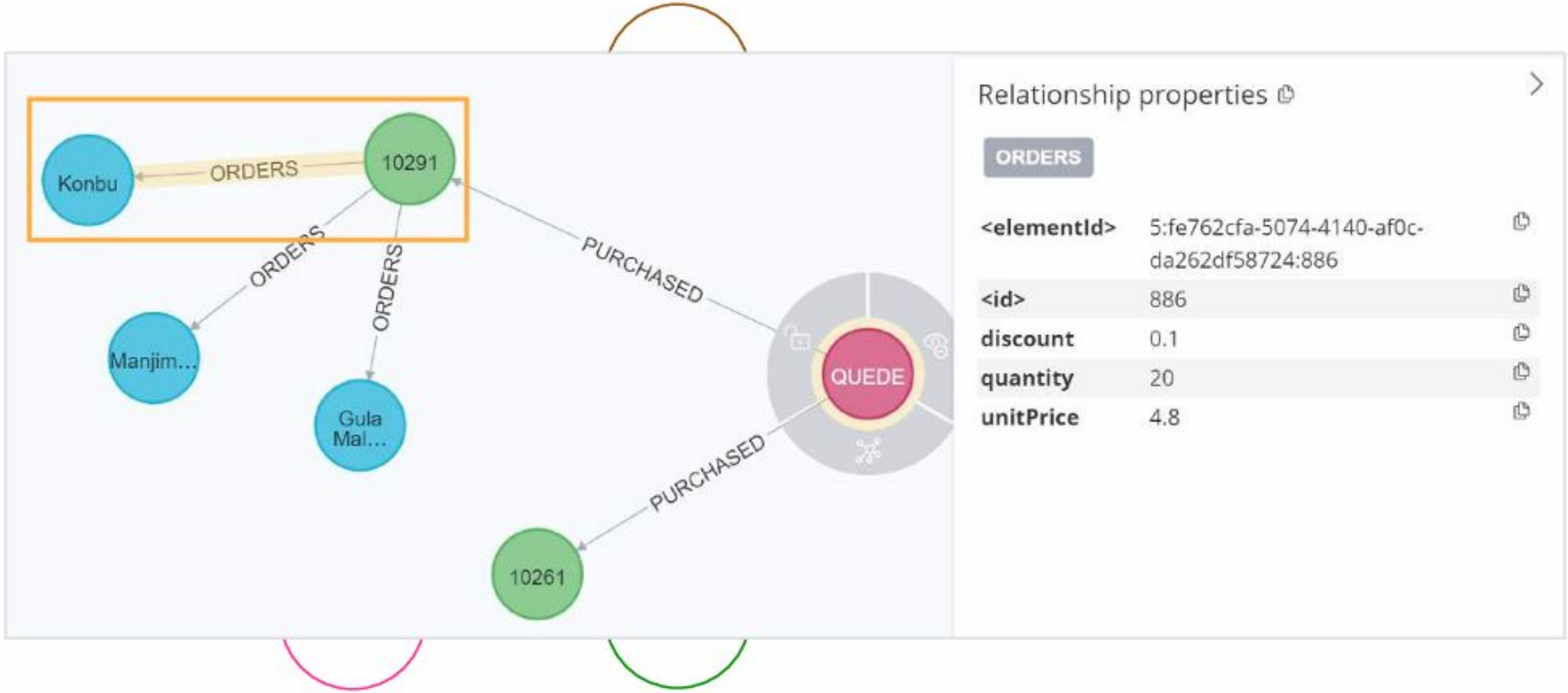
Customer Properties
address
city
companyName
contactName
contactTitle
country
customerID
…..

Supplier Properties
address
city
contactName
fax
region
supplierID
postalCode

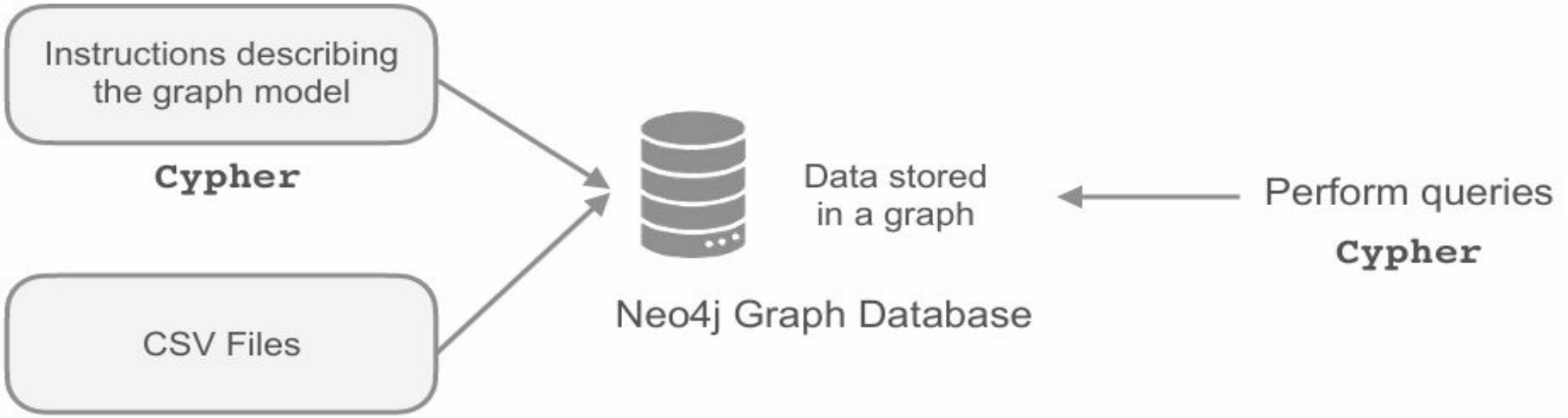
Category Properties
categoryName

Order Properties
freight
orderDate
orderID
requiredDate
shipAddress
……

Property Graph Model



Creating a Graph Database



- In the next video, we'll go through some queries examples.
- In the lab, you'll also practice CRUD operations.

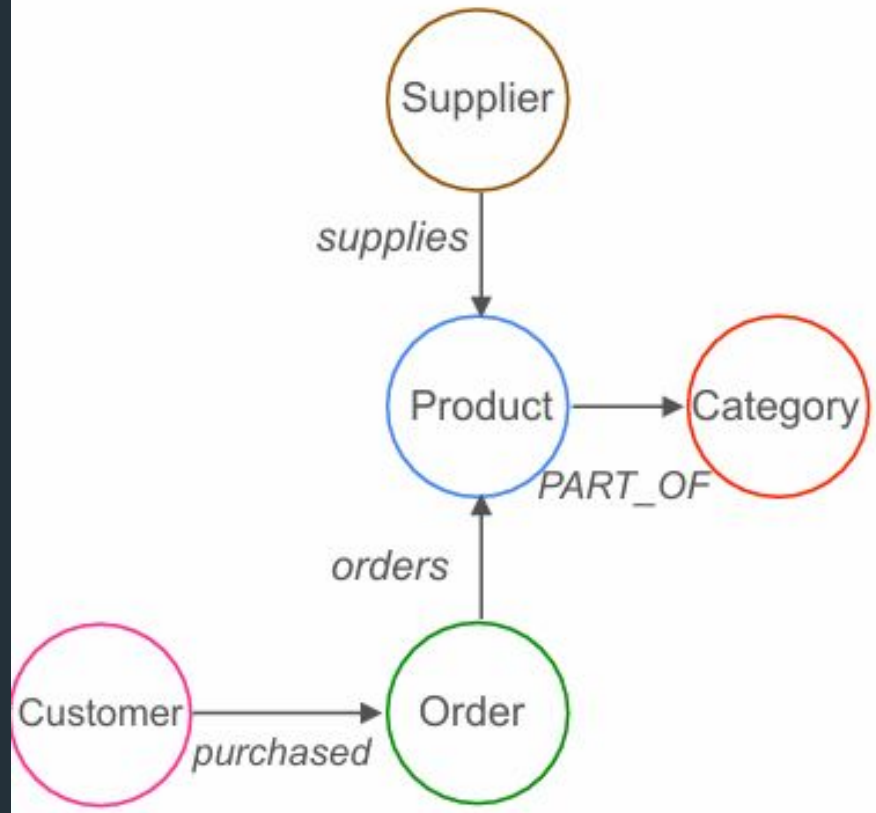
RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Neo4j Graph Database & Cypher Query Language

MATCH Statement

```
MATCH pattern RETURN result
```

node	( )

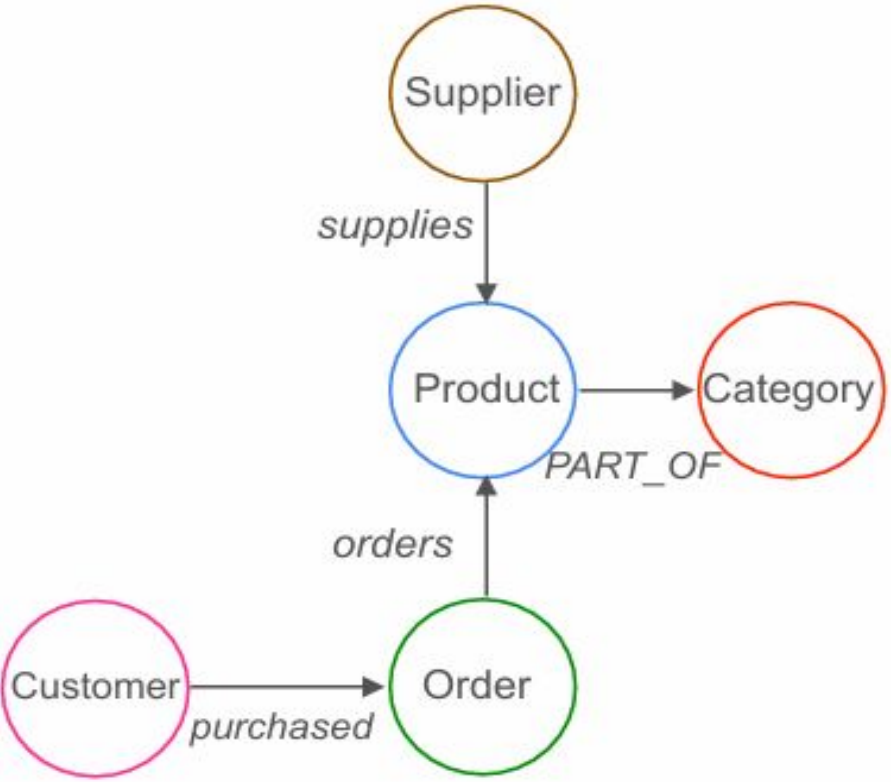


Retrieve all nodes

```
neo4j$
```

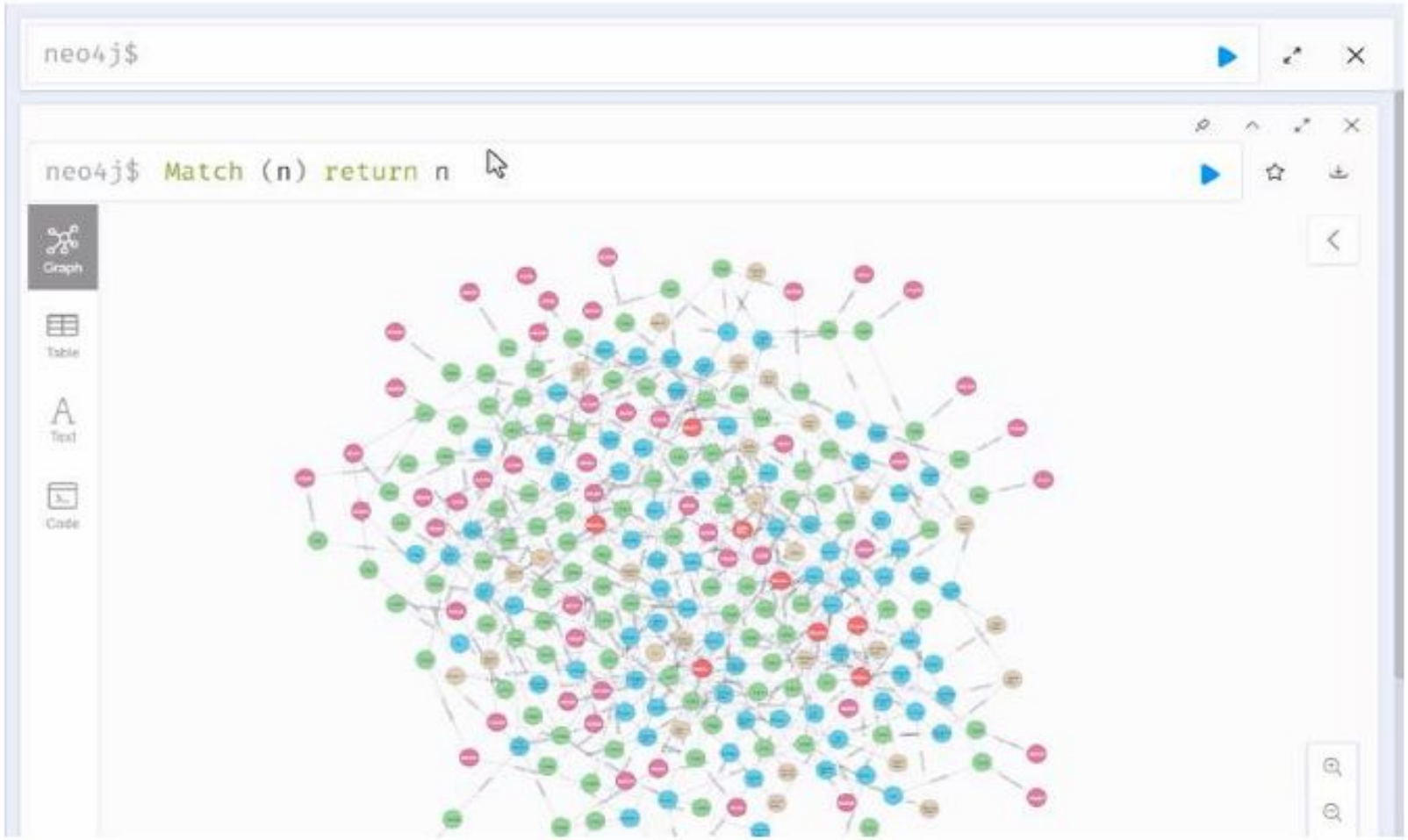
```
MATCH pattern RETURN result
```

node	()



MATCH Statement

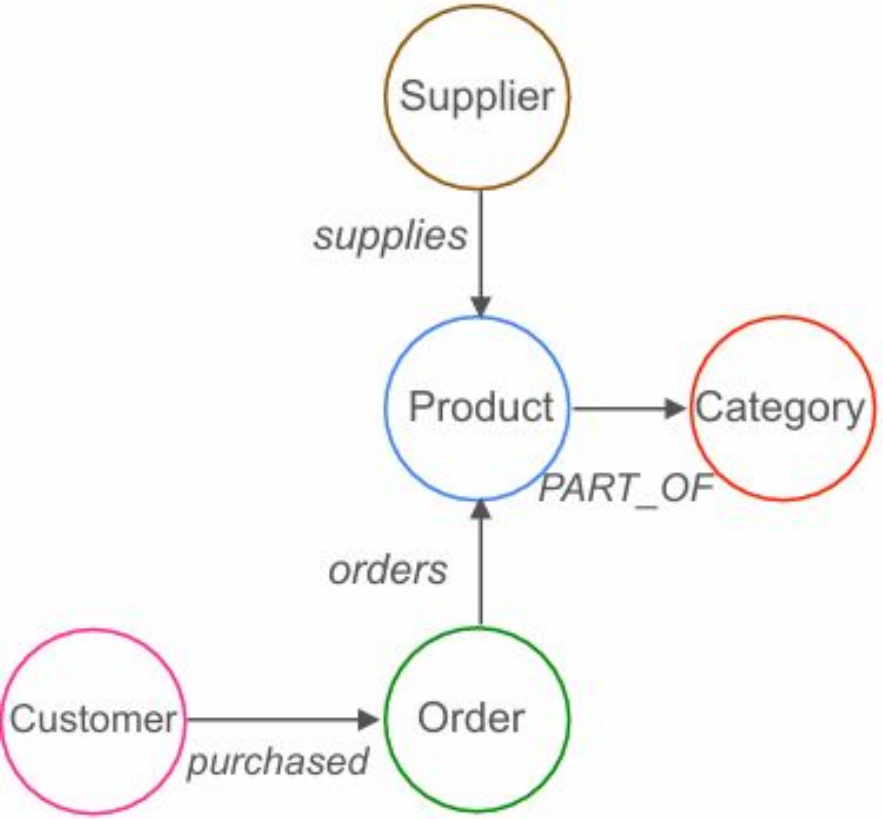
Get the total number of nodes



MATCH Statement

```
MATCH pattern RETURN result
```

node	( )



Explore the node labels using the `labels` function

```
neo4j$
```

```
neo4j$ Match (n) return count(n)
```

count(n)
265

Started streaming 1 records after 1 ms and completed after 1 ms.

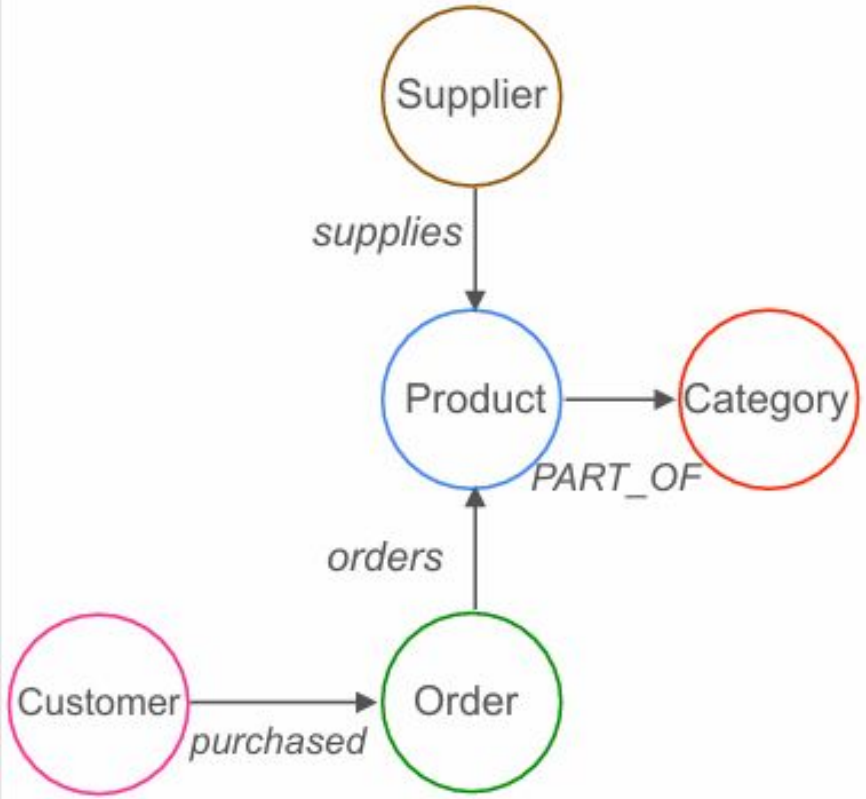
```
neo4j$ Match (n) return n
```

MATCH Statement

```
MATCH pattern RETURN result
```

node	( )

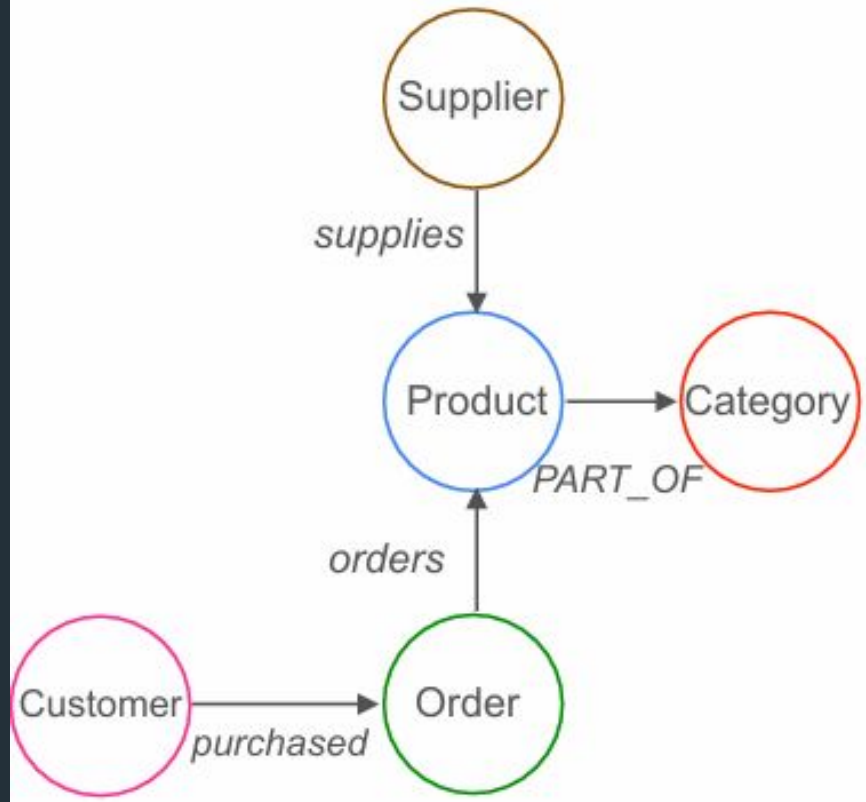
Specify the label of the node



MATCH Statement

```
MATCH pattern RETURN result
```

node	( )



Explore the properties of each order node using the `Properties` function

```
neo4j$
```

```
neo4j$ Match (n:Order) return count(n)
```

count(n)
99

Started streaming 1 records after 2 ms and completed after 10 ms.

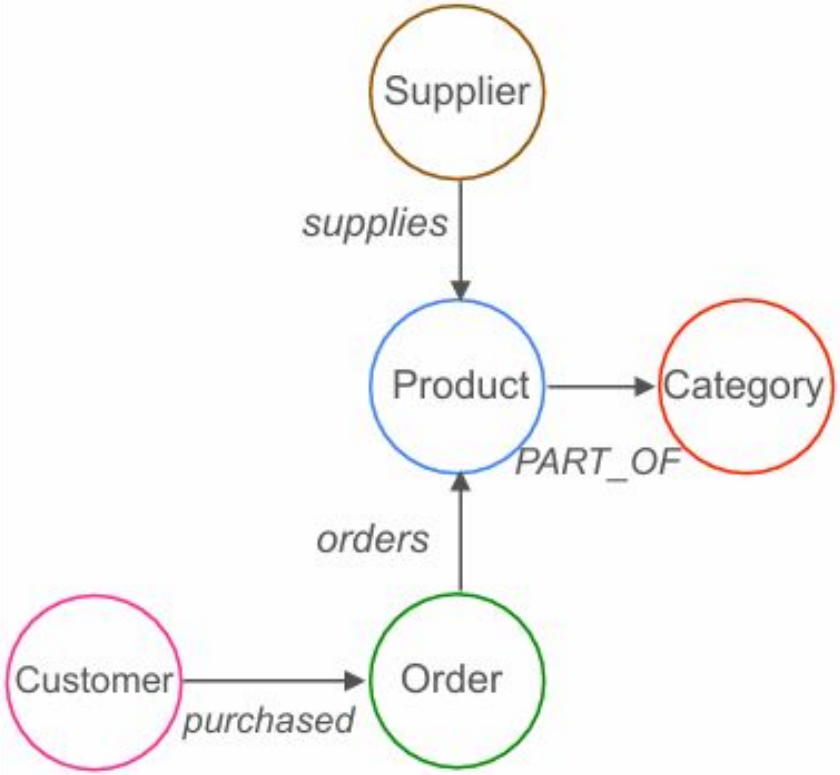
```
neo4j$ Match (n) return distinct labels(n)
```

labels(n)

MATCH Statement

```
MATCH pattern RETURN result
```

node	()



Explore the properties of each order node using the `Properties` function

```
neo4j$
neo4j$ Match (n:Order) return Properties(n)
```

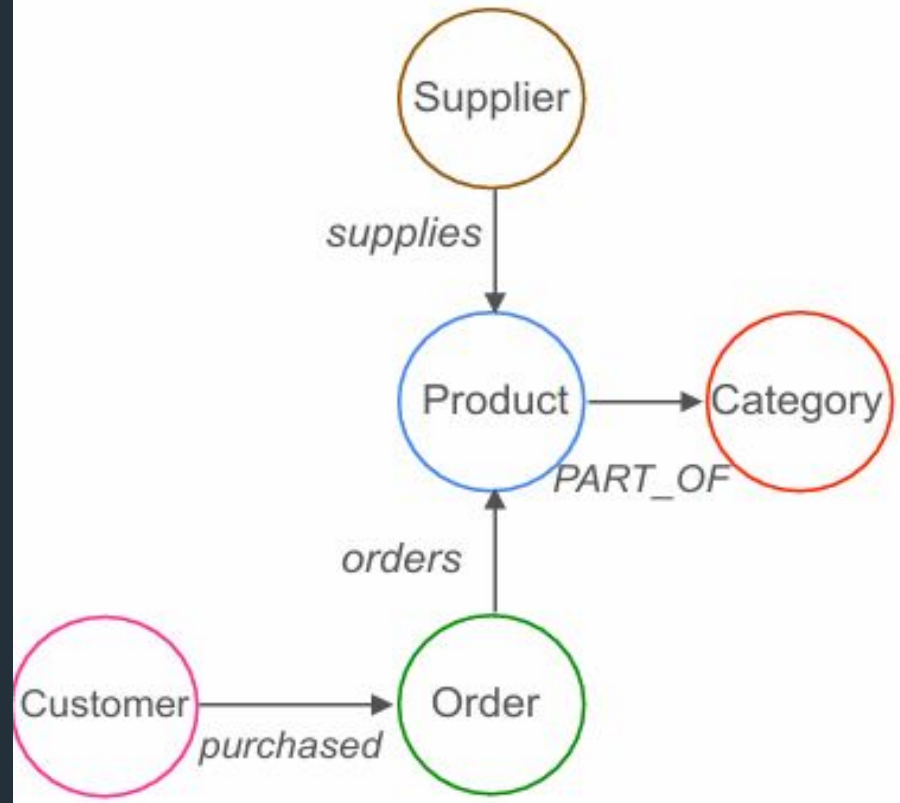
Properties(n)


```
{
  "shipCity": "Reims",
  "orderID": "10248",
  "shippedDate": "00:00.0",
  "orderDate": "00:00.0",
  "shipRegion": "NULL",
  "freight": "32.38",
  "shipName": "Vins et alcools Chevalier",
  "shipCountry": "France",
  "shipAddress": "59 rue de l'Abbaye",
  "requiredDate": "00:00.0",
  "shipPostalCode": "51100"
}
```

MATCH Statement

`MATCH pattern RETURN result`

node	( )
relationship	[ ]
path	(source node)-[r]->(target node)



Count all the directed paths

```
neo4j$
neo4j$ Match (n:Order) return Properties(n) limit 1
```

Properties(n)

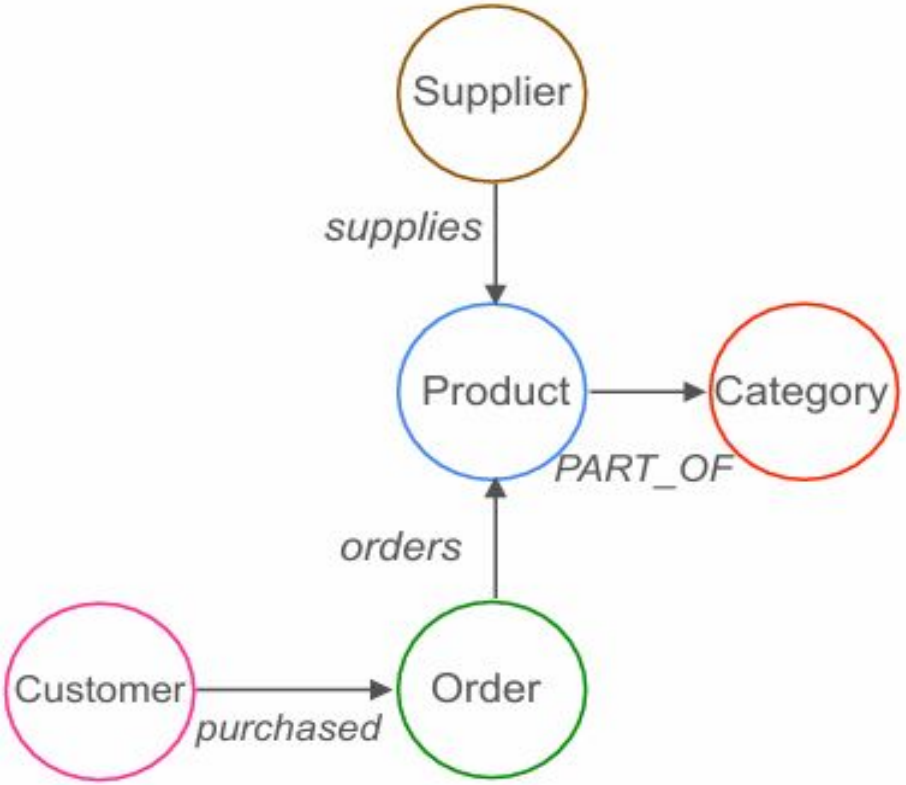

```
{
  "shipCity": "Reims",
  "orderID": "10248",
  "shippedDate": "00:00.0",
  "orderDate": "00:00.0",
  "shipRegion": "NULL",
  "freight": "32.38",
  "shipName": "Vins et alcools Chevalier",
  "shipCountry": "France",
  "shipAddress": "59 rue de l'Abbaye",
  "requiredDate": "00:00.0",
  "shipPostalCode": "51100"
}
```


Started streaming 1 records after 2 ms and completed after 8 ms.

MATCH Statement

MATCH *pattern* RETURN *result*

node	( )
relationship	[ ]
path	(source node)-[r]->(target node)



Return the types of relationships

```
neo4j$
```

```
neo4j$ Match ()-[r]→() return count(r)
```

count(r)
518

Started streaming 1 records in less than 1 ms and completed after 5 ms.

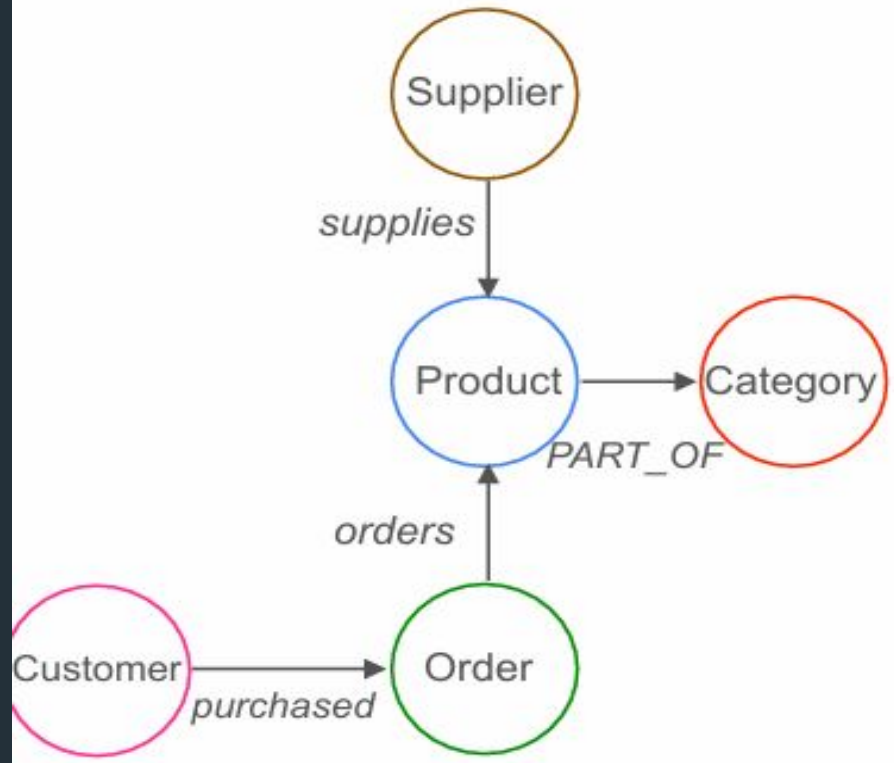
```
neo4j$ Match (n:Order) return Properties(n) limit 1
```

Properties(n)

MATCH Statement

MATCH *pattern* RETURN *result*

node	( )
relationship	[ ]
path	(source node)-[r]->(target node)



Specify the type of the relationship

```
neo4j$
```

```
neo4j$ Match ()-[r]→() return distinct type(r)
```

	type(r)
1	"SUPPLIES"
2	"PART_OF"
3	"PURCHASED"
4	"ORDERS"

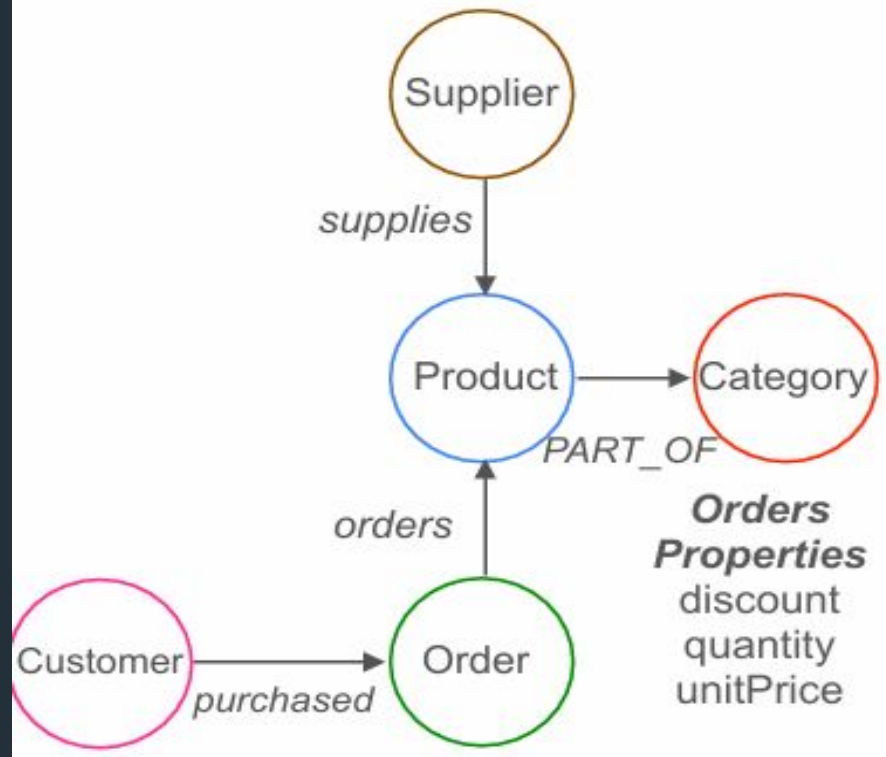
Started streaming 4 records after 1 ms and completed after 21 ms.

```
neo4j$ Match (n:Order) return Properties(n) limit 1
```

MATCH Statement

MATCH *pattern* RETURN *result*

node	( )
relationship	[ ]
path	(source node)-[r]->(target node)



Return the properties of a relationship

```
neo4j$ Match ()-[r:ORDERS]→()
```

```
neo4j$ Match ()-[r]→() return distinct type(r)
```

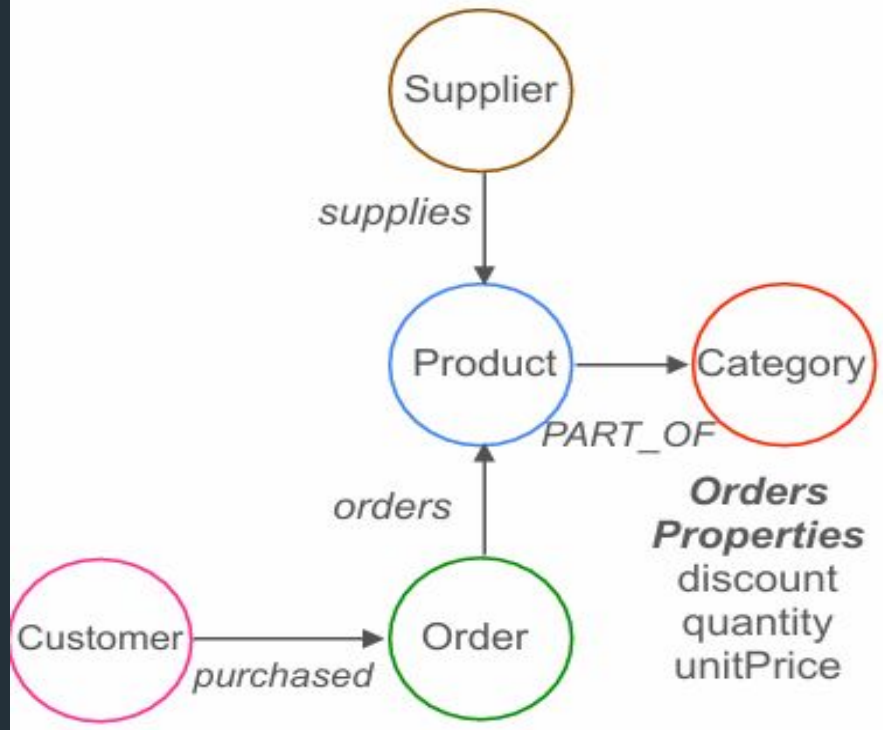
	type(r)
1	"SUPPLIES"
2	"PART_OF"
3	"PURCHASED"
4	"ORDERS"

Started streaming 4 records after 1 ms and completed after 21 ms.

```
neo4j$ Match (n:Order) return Properties(n) limit 1
```

`MATCH pattern RETURN result`

node	( )
relationship	[ ]
path	(source node)-[ ]->(target node)



MATCH Statement

Return the properties of a relationship

```
neo4j$
neo4j$ Match ()-[r:ORDERS]→() return AVG(r.quantity*r.unitPrice)
```

AVG(r.quantity*r.unitPrice)
502.4645283018867

Started streaming 1 records after 1 ms and completed after 25 ms.

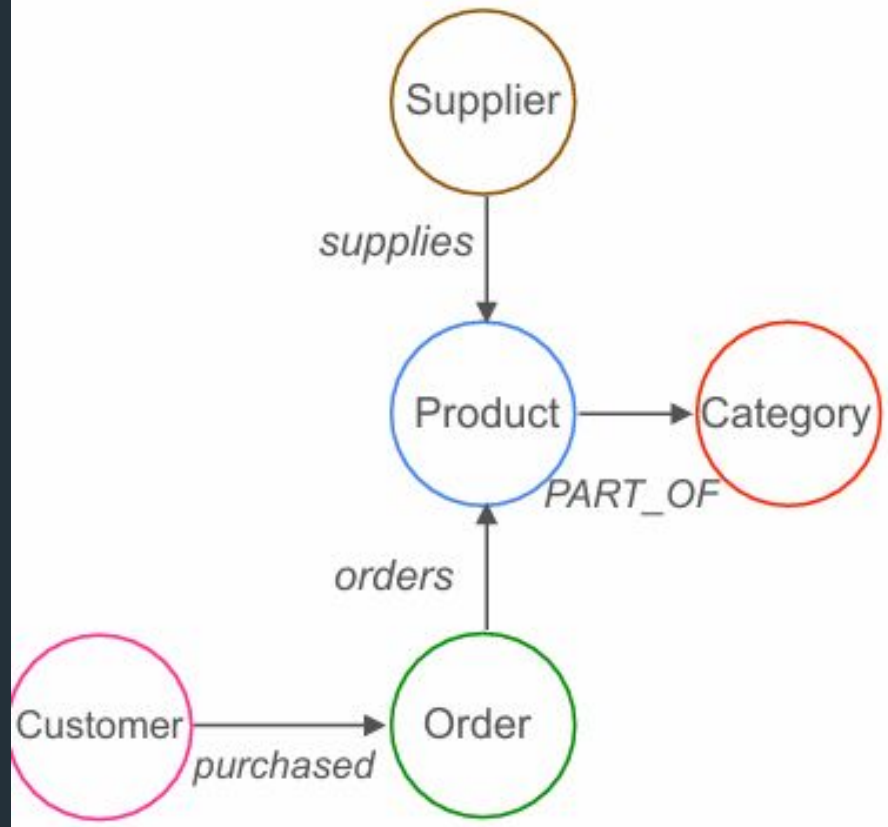
```
neo4j$ Match ()-[r]→() return distinct type(r)
```

type(r)

MATCH Statement

`MATCH pattern RETURN result`

node	( )
relationship	[ ]
path	(source node)-[ ]->(target node)



Get the average price for all orders grouped by product category

```
neo4j$
```

```
neo4j$ Match ()-[r:ORDERS]→() return AVG(r.quantity*r.unitPrice) as a…
```

average_price
502.4645283018867

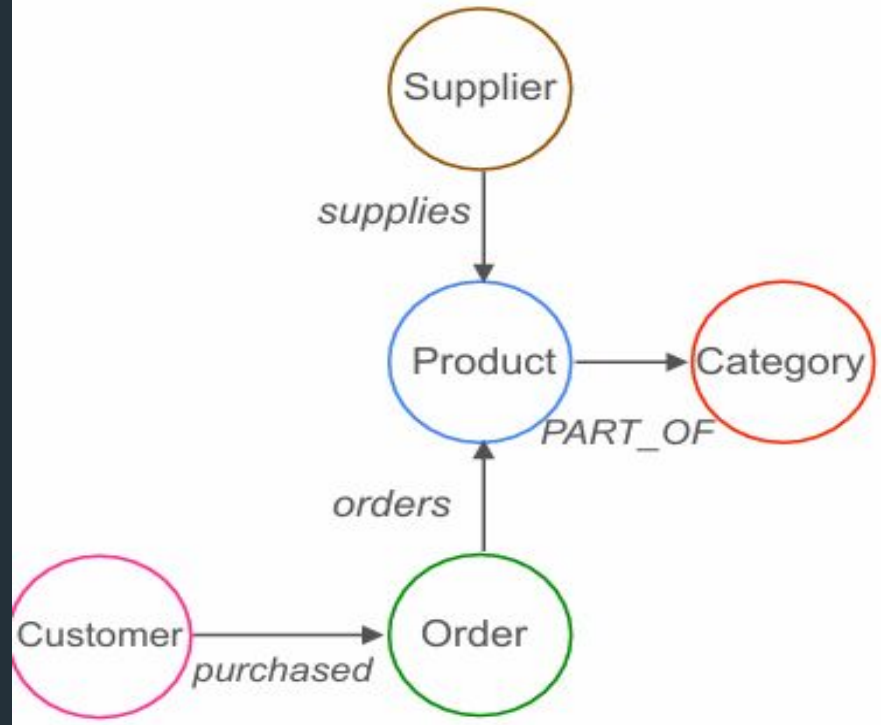
Started streaming 1 records in less than 1 ms and completed after 25 ms.

```
neo4j$ Match ()-[r]→() return distinct type(r)
```

type(r)

MATCH *pattern* RETURN *result*

node	( )
relationship	[ ]
path	(source node)-[ ]->(target node)



MATCH Statement

Get the average price for all orders grouped by product category

```
neo4j$ Match ()-[r:ORDERS]→()-[part:PART_OF]→(c:Category) return
       AVG(r.quantity*r.unitPrice) as average_price
```

```
neo4j$ Match ()-[r:ORDERS]→() return AVG(r.quantity*r.unitPrice) as a…
```

	average_price
1	502.4645283018867

Started streaming 1 records in less than 1 ms and completed after 25 ms.

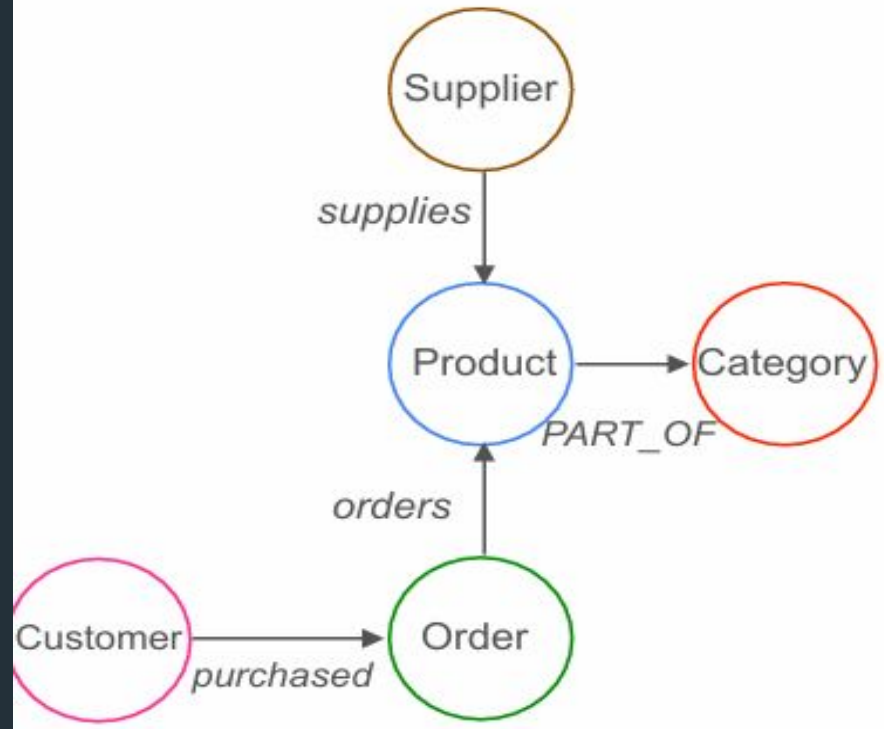
```
neo4j$ Match ()-[r]→() return distinct type(r)
```

type(r)

MATCH Statement

`MATCH pattern RETURN result`

node	( )
relationship	[ ]
path	(source node)-[ ]->(target node)



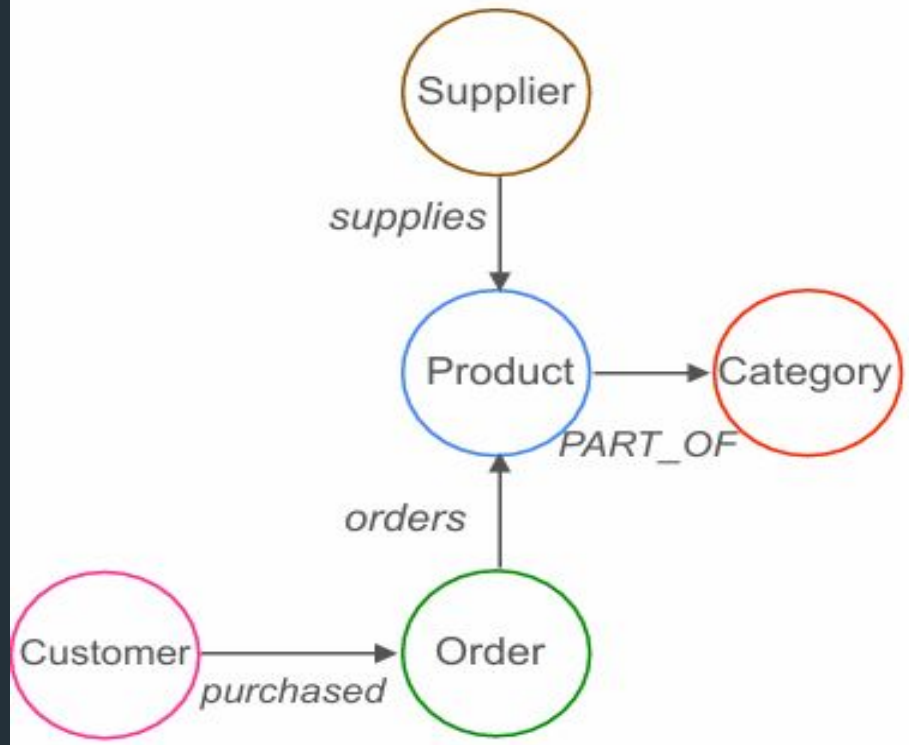
Retrieve the product name and product unit price of all products that belong to category “Meat/Poultry”

```
neo4j$ Match ()-[r:ORDERS]→()-[part:PART_OF]→(c:Category) return c.c…
```

	c.categoryName	average_price
3	"Confections"	531.3666666666664
4	"Dairy Products"	504.31836734693877
5	"Grains/Cereals"	278.67
6	"Meat/Poultry"	766.9142857142859
7	"Produce"	645.6571428571427
8	"Seafood"	434.8282051282051

MATCH *pattern* RETURN *result*

node	( )
relationship	[ ]
path	(source node)-[ ]->(target node)



MATCH Statement

Retrieve the product name and product unit price of all products that belong to category “Meat/Poultry”

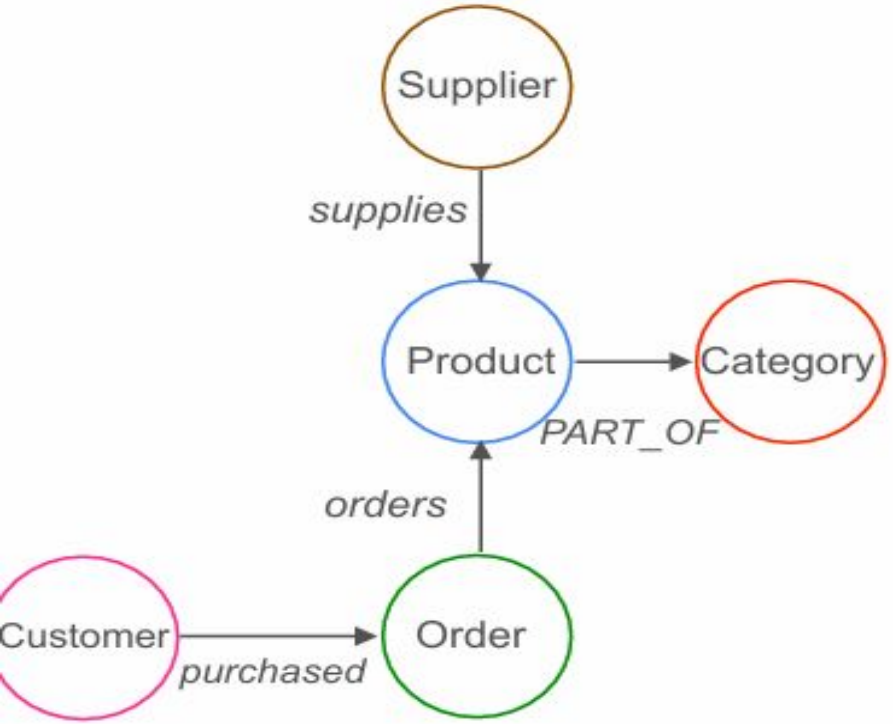
```
neo4j$ Match (p:Product)-[:PART_OF]→(c:Category)
```

```
neo4j$ Match ()-[r:ORDERS]→()-[part:PART_OF]→(c:Category) return c.c…
```

	c.categoryName	average_price
3	"Confections"	531.3666666666664
4	"Dairy Products"	504.31836734693877
5	"Grains/Cereals"	278.67
6	"Meat/Poultry"	766.9142857142859
7	"Produce"	645.6571428571427
8	"Seafood"	434.8282051282051

MATCH *pattern* RETURN *result*

node	( )
relationship	[ ]
path	(source node)-[ ]->(target node)



MATCH Statement

Retrieve the product name and product unit price of all products that belong to category “Meat/Poultry”

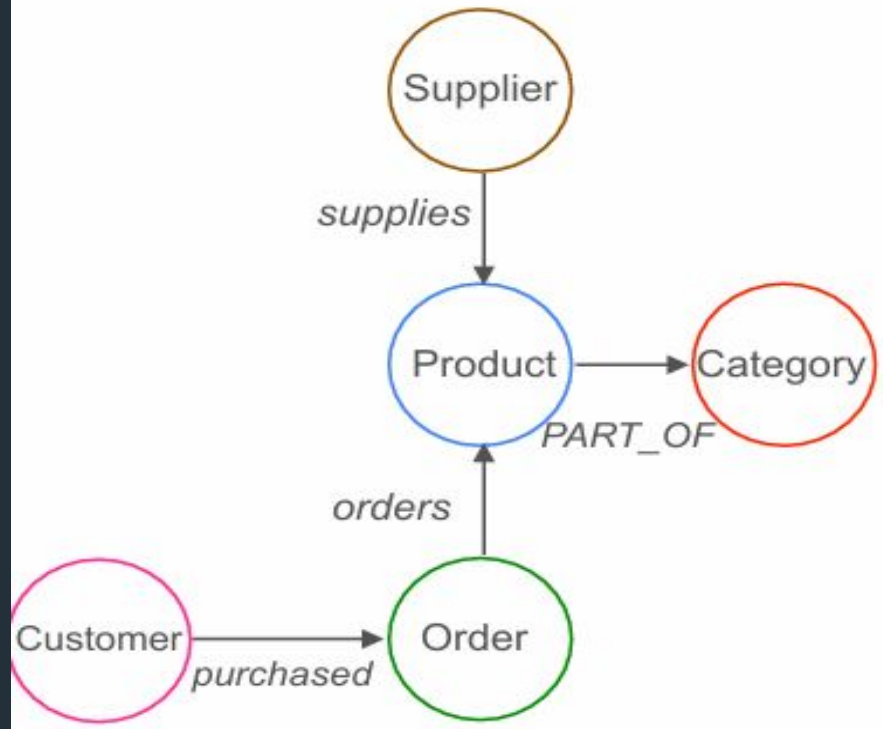
```
1 Match (p:Product)-[:PART_OF]→(c:Category)
2 where c.categoryName="Meat/Poultry"
```

```
neo4j$ Match ()-[r:ORDERS]→()-[part:PART_OF]→(c:Category) return c.c…
```

	c.categoryName	average_price
3	"Confections"	531.3666666666664
4	"Dairy Products"	504.31836734693877
5	"Grains/Cereals"	278.67
6	"Meat/Poultry"	766.9142857142859
7	"Produce"	645.6571428571427
8	"Seafood"	434.8282051282051

MATCH *pattern* RETURN *result*

node	( )
relationship	[ ]
path	(source node)-[ ]->(target node)



MATCH Statement

Retrieve the product name and product unit price of all products that belong to category “Meat/Poultry”

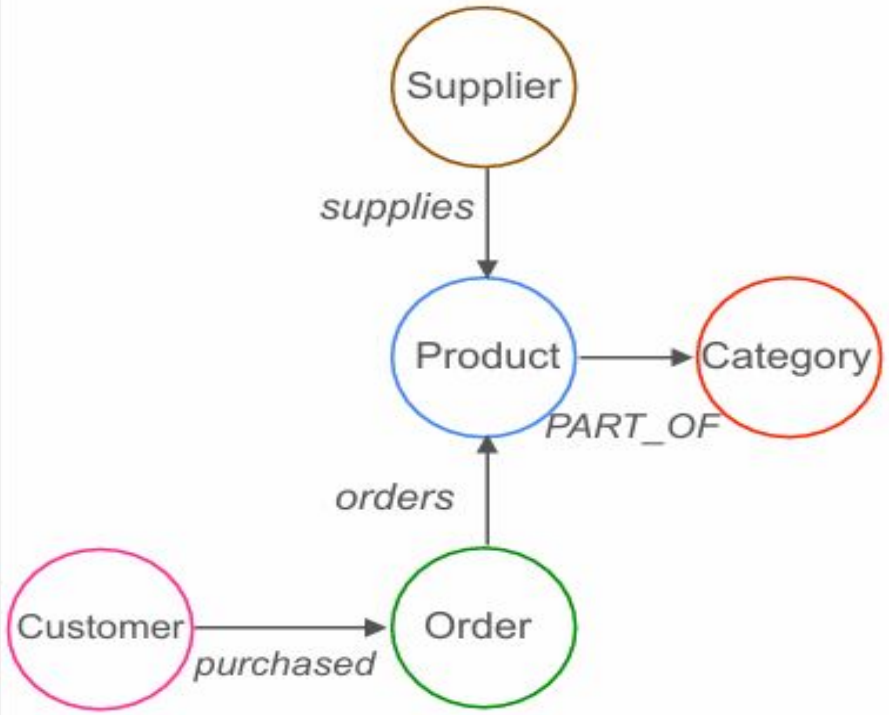
neo4j$

```
1 Match (p:Product)-[:PART_OF]→(c:Category)
2 where c.categoryName="Meat/Poultry"
3 return p.productName, p.unitPrice
```

	p.productName	p.unitPrice
1	"Perth Pasties"	32.8
2	"Tourtière"	7.45
3	"Alice Mutton"	39.0
4	"Pâté chinois"	24.0
6	"Thüringer Rostbratwurst"	123.79
8	"Mishi Kobe Niku"	97.0

MATCH *pattern* RETURN *result*

node	( )
relationship	[ ]
path	(source node)-[ ]->(target node)



MATCH Statement

Retrieve the product name of all products ordered by the customer “QUEDE”

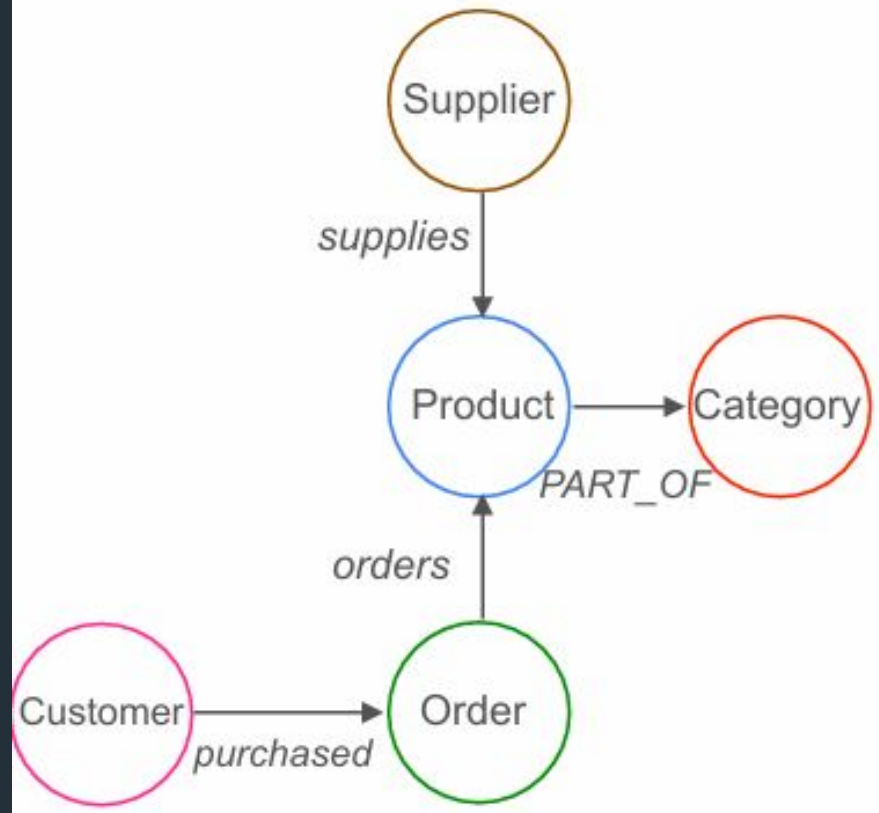
```
neo4j$ Match (p:Product)-[:PART_OF]→(c:Category {categoryName:"Meat/P…
```

	p.productName	p.unitPrice
1	"Perth Pasties"	32.8
2	"Tourtière"	7.45
3	"Alice Mutton"	39.0
4	"Pâté chinois"	24.0
5	"Thüringer Rostbratwurst"	123.79
6	"Mishi Kobe Niku"	97.0

MATCH Statement

`MATCH pattern RETURN result`

node	( )
relationship	[ ]
path	(source node)-[ ]->(target node)



Get the ID of other customers who ordered the same products as “QUEDE”

```
neo4j$ Match (c1:Customer {customerID:"QUEDE"}) -[:PURCHASED]→()-
       [:ORDERS]→(p:Product)
```

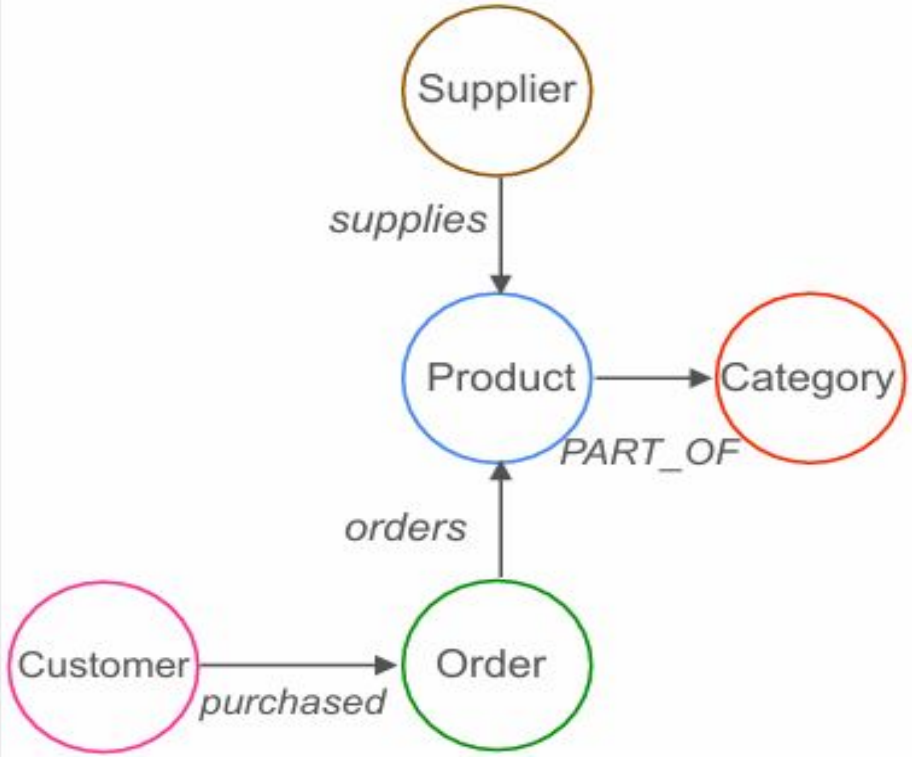
```
neo4j$ Match (c1:Customer {customerID:"QUEDE"}) -[:PURCHASED]→()-[:OR…
```

	p.productName
1	"Steeleye Stout"
2	"Sir Rodney's Scones"
3	"Gula Malacca"
4	"Konbu"
5	"Manjimup Dried Apples"

Started streaming 5 records after 1 ms and completed after 1 ms.

MATCH *pattern* RETURN *result*

node	( )
relationship	[ ]
path	(source node)-[ ]->(target node)



MATCH Statement

Retrieve the orders that contain at most two products

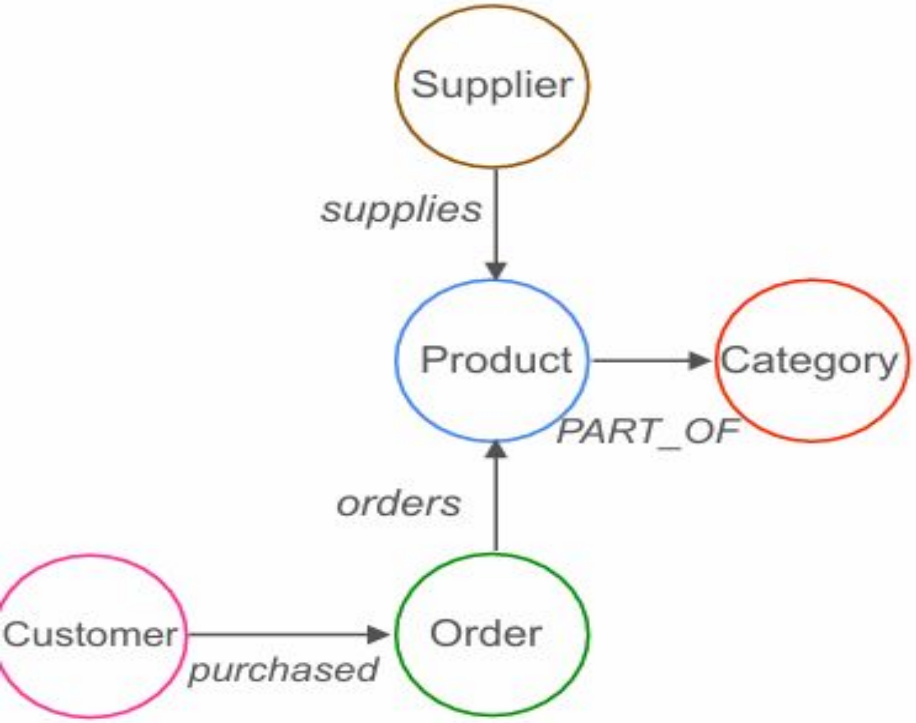
```
neo4j$
```

```
neo4j$ Match (c1:Customer {customerID:"QUEDE"}) -[:PURCHASED]->()-[:OR…
```

	c2.customerID
1	"QUICK"
2	"SAVEA"
3	"ROMEY"
4	"ISLAT"
5	"WARTH"
6	"CENTC"

MATCH *pattern* RETURN *result*

node	( )
relationship	[ ]
path	(source node)-[ ]->(target node)



MATCH Statement

Retrieve the orders that contain at most two products

neo4j$

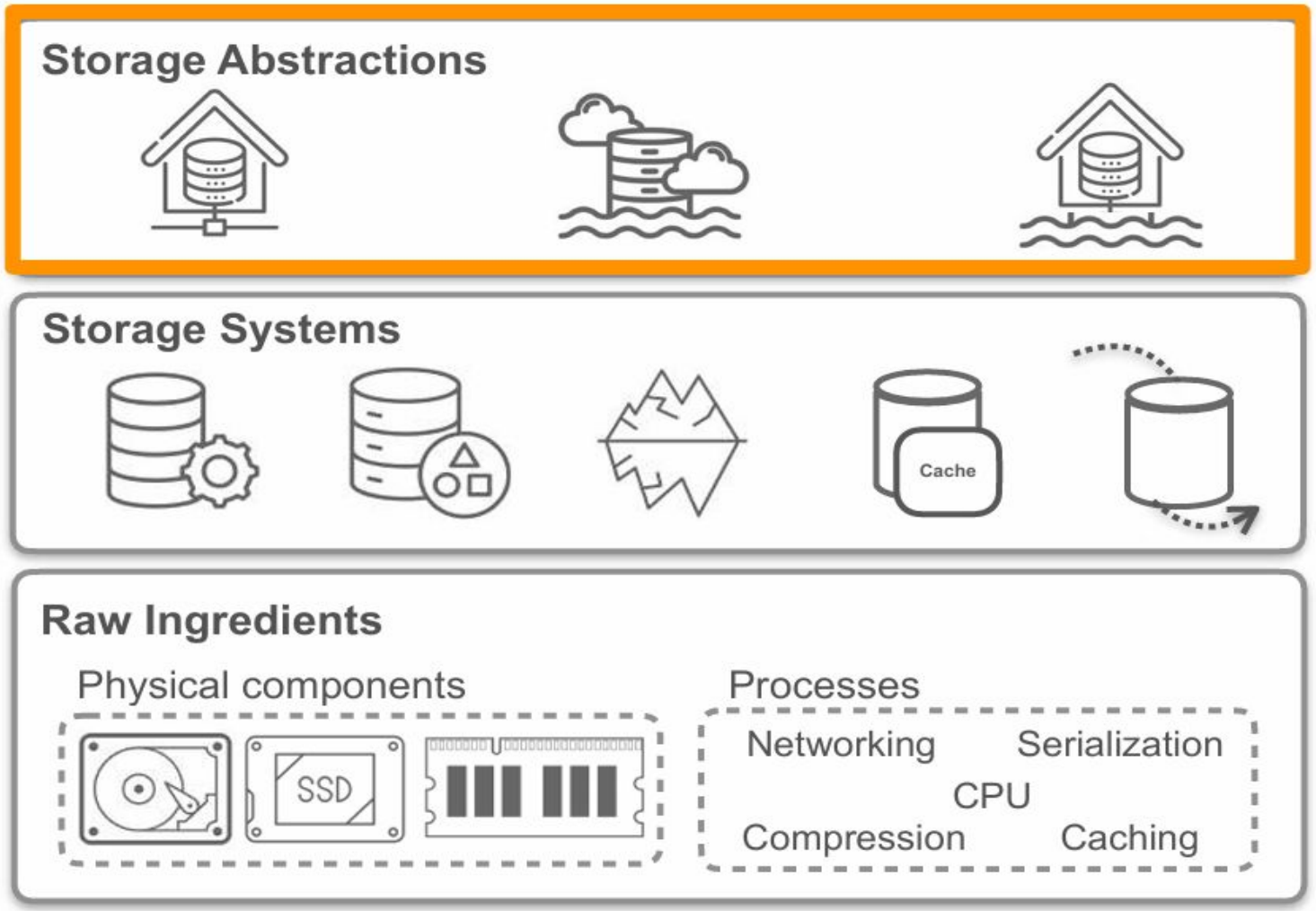
```
1 Match (o:Order)-[:ORDERS]→(p:Product)
2 return o.orderID as ID, count(p) as countProd
```

	ID	countProd
1	"10294"	5
2	"10317"	1
3	"10285"	3
4	"10342"	4
5	"10255"	4
6	"10327"	4

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Storage Abstractions

Storage Hierarchy



Storage Hierarchy

Data Warehouse

Cloud data warehouse

Data Lake

Supports growing storage needs

Data Lakehouse

Combines the advantages of data warehouses and data lakes

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Data Warehouse – Key Architectural ideas

Data Warehouse

Bill Inmon

"Father" of the Data Warehouse

Data Warehouse:

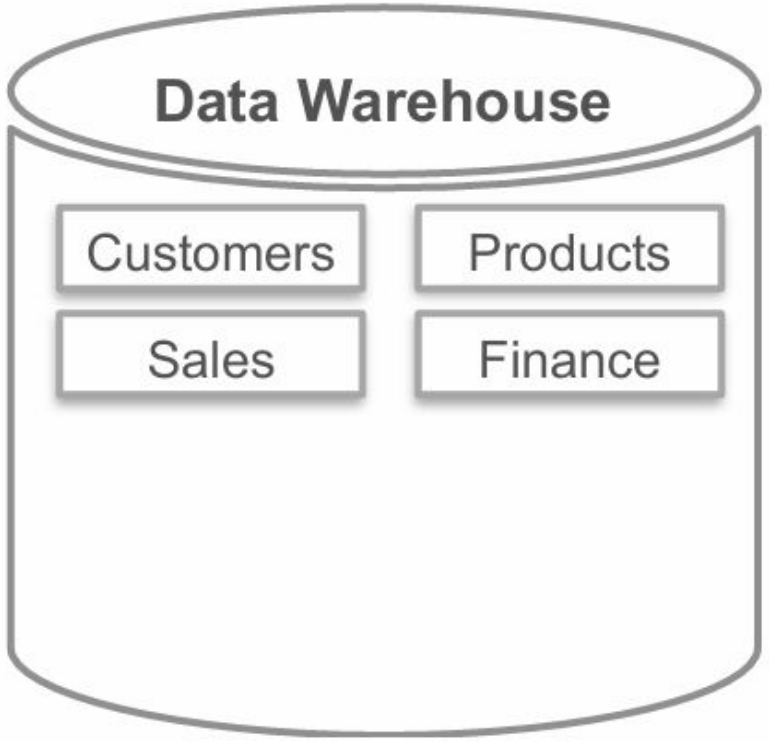
A subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management's decisions.

Data Warehouse

Subject-Oriented

Organizes and stores data around key business domains

(Models data to support decision making)



Data Warehouse

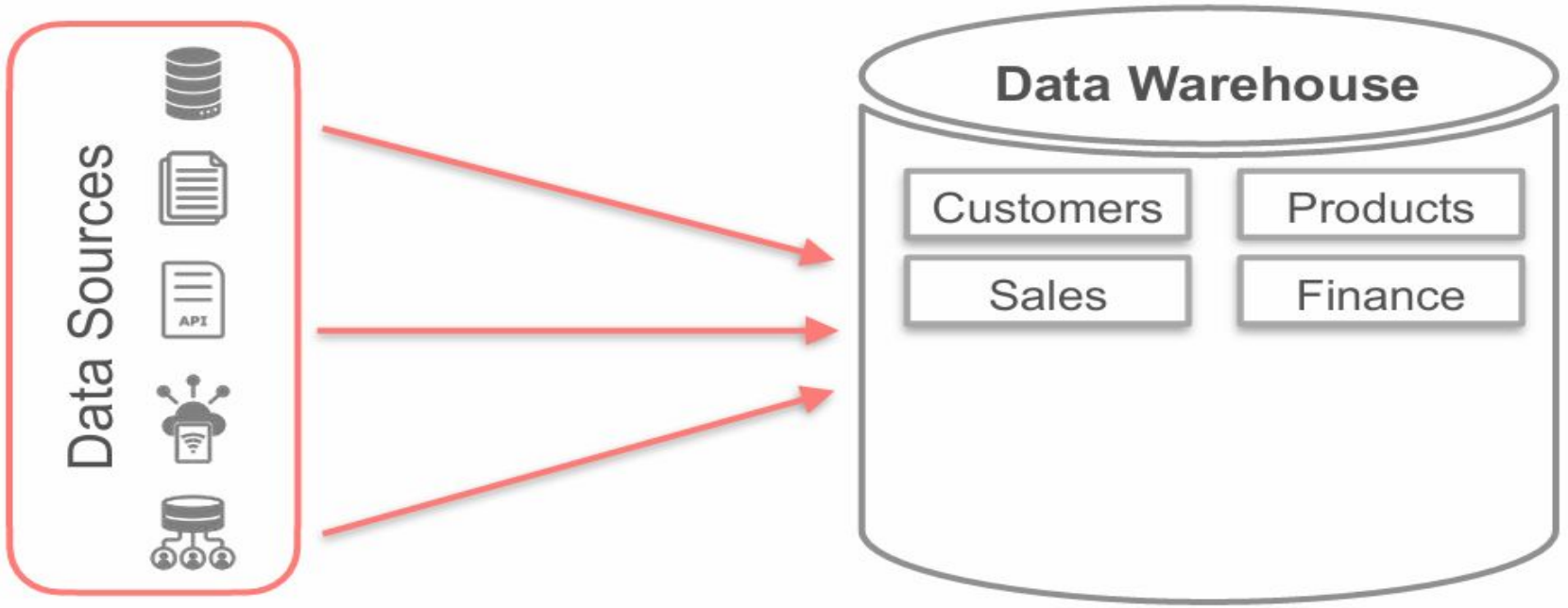
Subject-Oriented

Organizes and stores data around key business domains

(Models data to support decision making)

Integrated

Combines data from different sources into a consistent format



Data Warehouse

Subject-Oriented

Organizes and stores data around key business domains

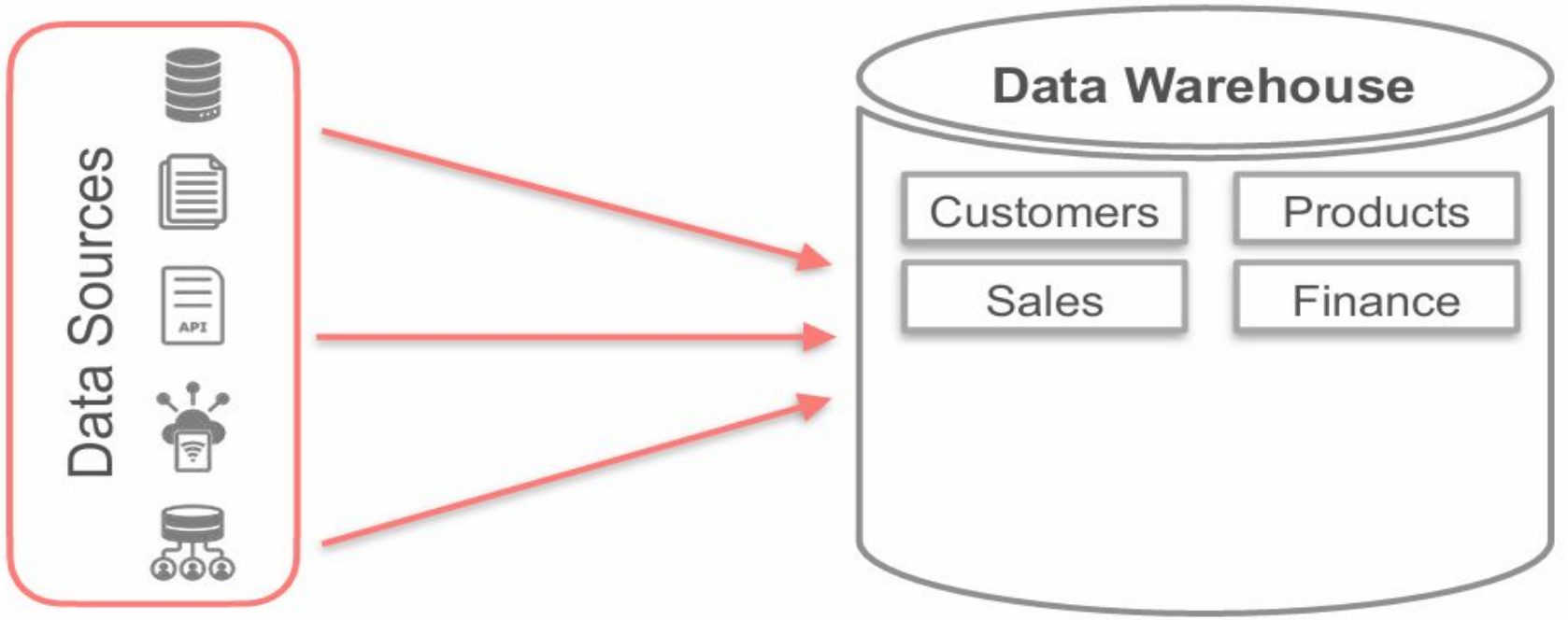
(Models data to support decision making)

Integrated

Combines data from different sources into a consistent format

Nonvolatile

Data is read-only and cannot be deleted or updated



Data Warehouse

Subject-Oriented

Organizes and stores data around key business domains

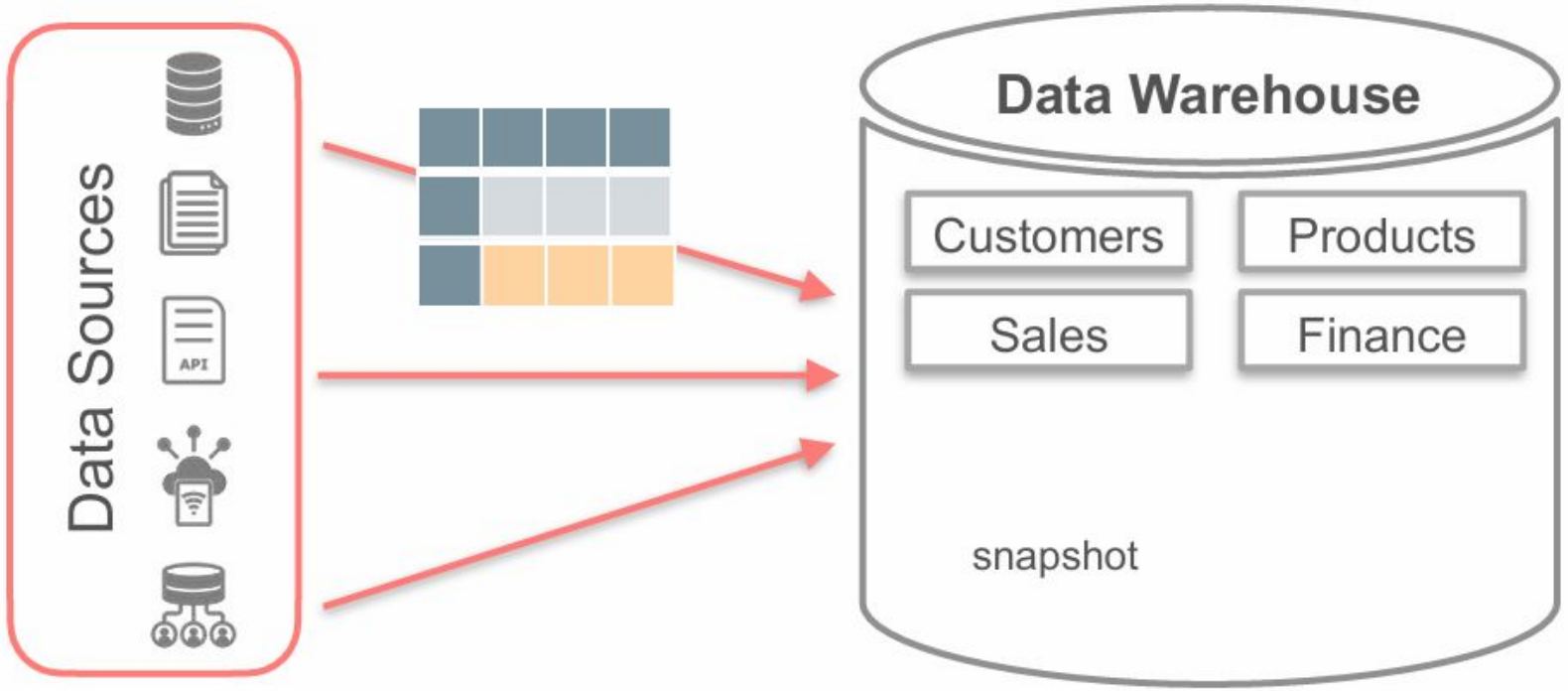
(Models data to support decision making)

Integrated

Combines data from different sources into a consistent format

Nonvolatile

Data is read-only and cannot be deleted or updated



Data Warehouse

Subject-Oriented

Organizes and stores data around key business domains

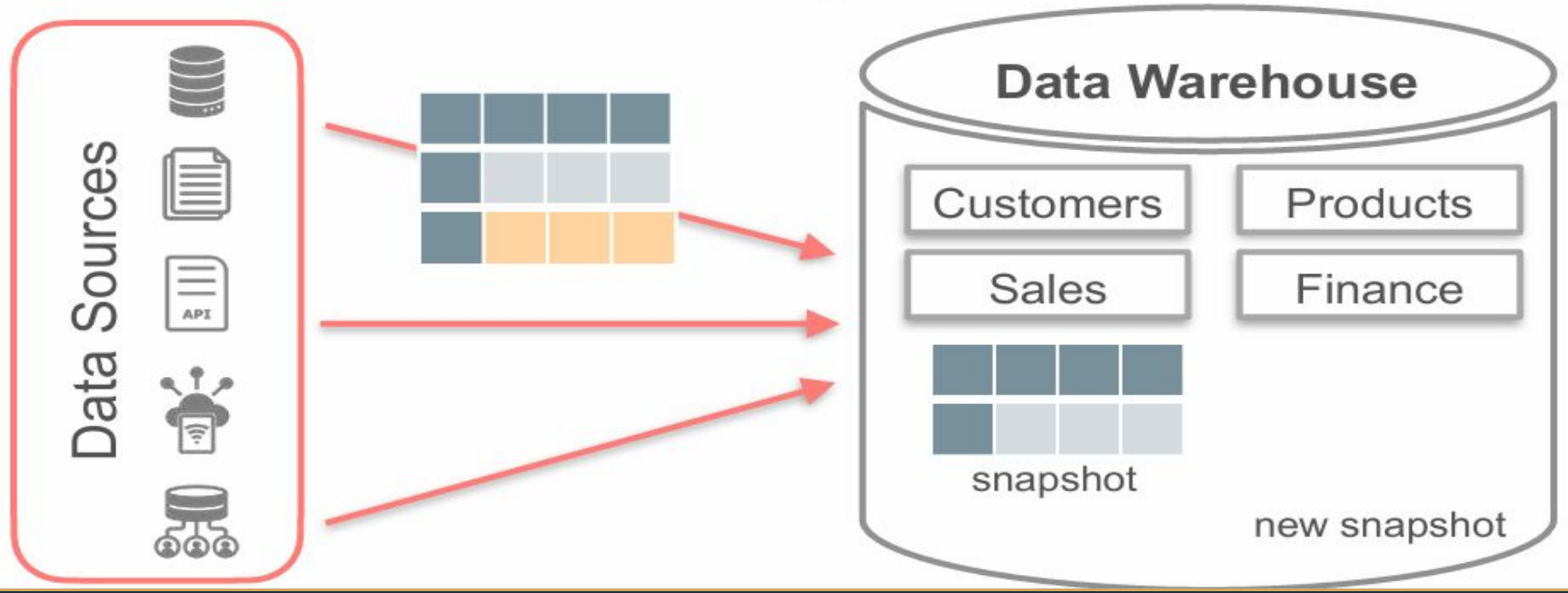
(Models data to support decision making)

Integrated

Combines data from different sources into a consistent format

Nonvolatile

Data is read-only and cannot be deleted or updated



Data Warehouse

Subject-Oriented

Organizes and stores data around key business domains

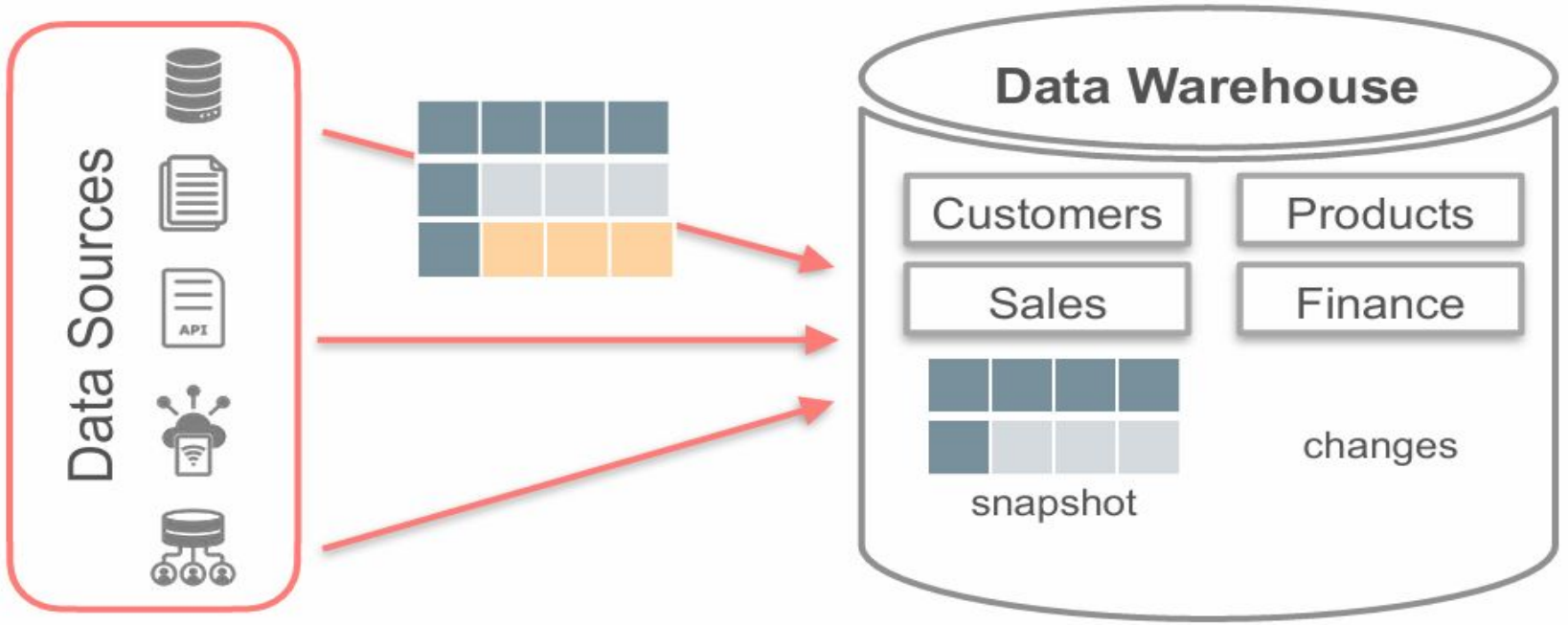
(Models data to support decision making)

Integrated

Combines data from different sources into a consistent format

Nonvolatile

Data is read-only and cannot be deleted or updated



Data Warehouse

Subject-Oriented

Organizes and stores data around key business domains

(Models data to support decision making)

Integrated

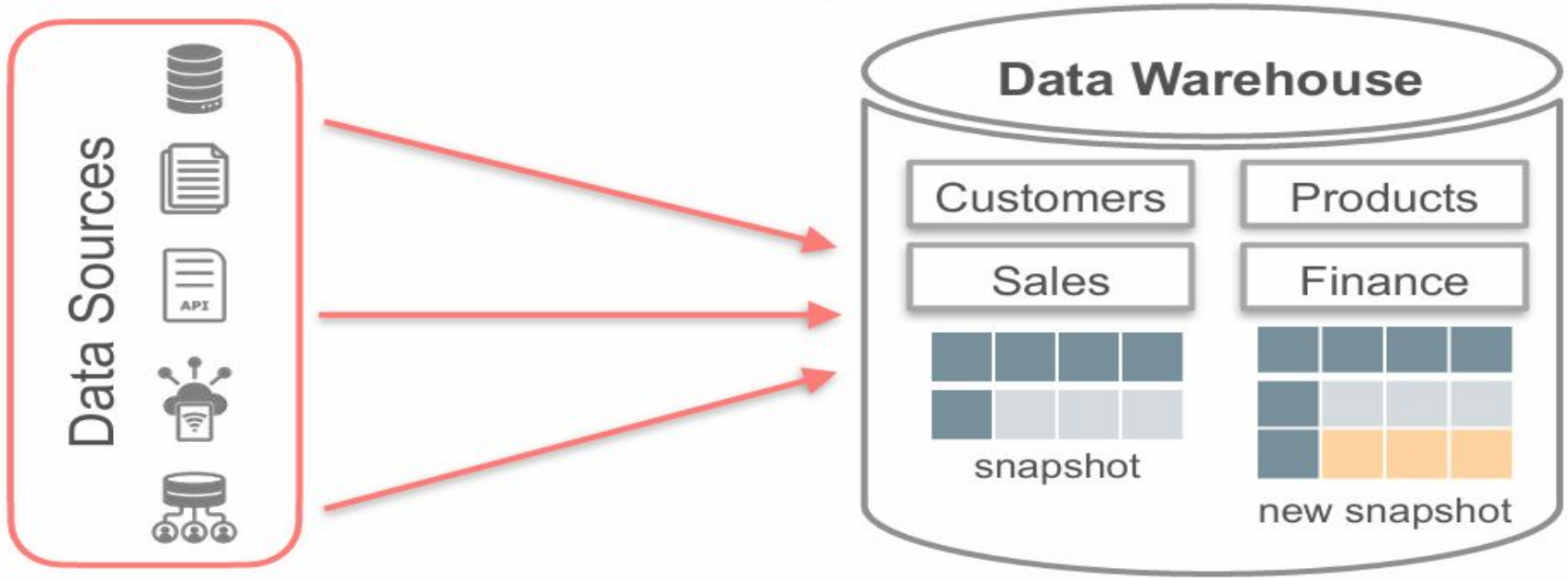
Combines data from different sources into a consistent format

Nonvolatile

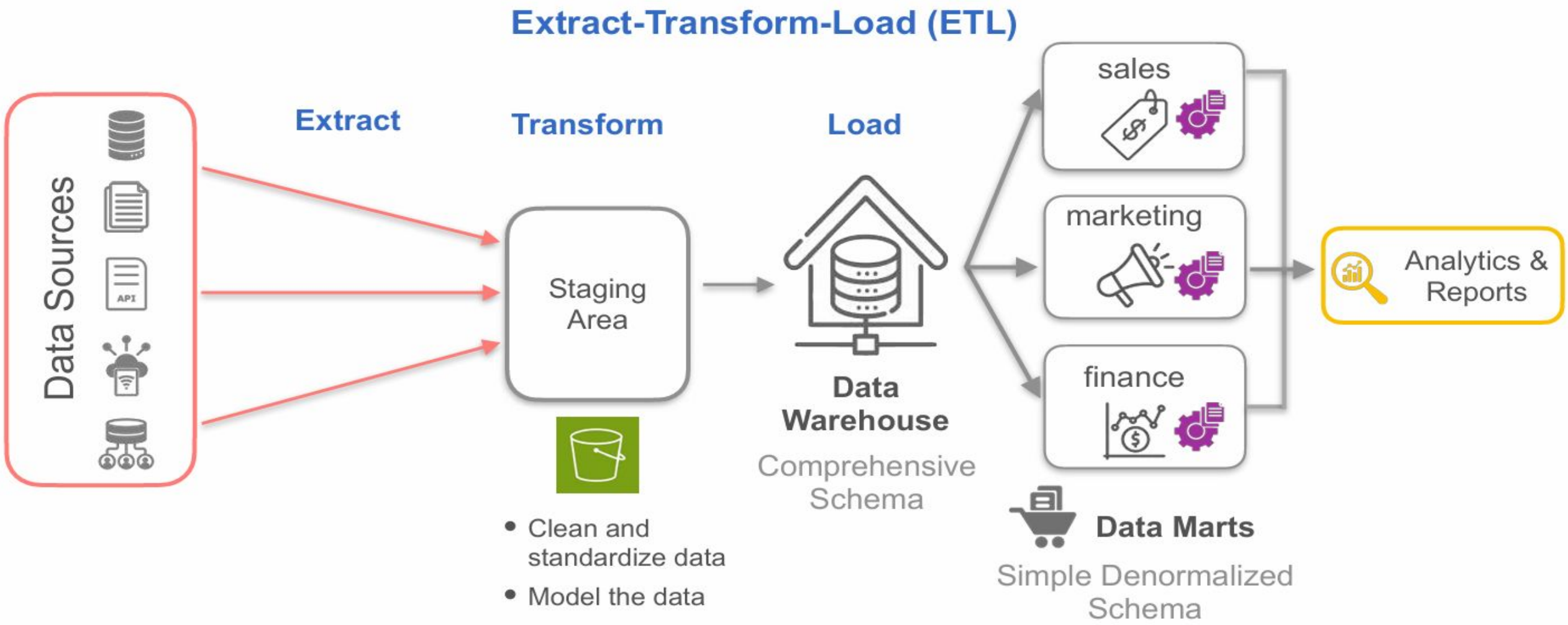
Data is read-only and cannot be deleted or updated

Time-variant

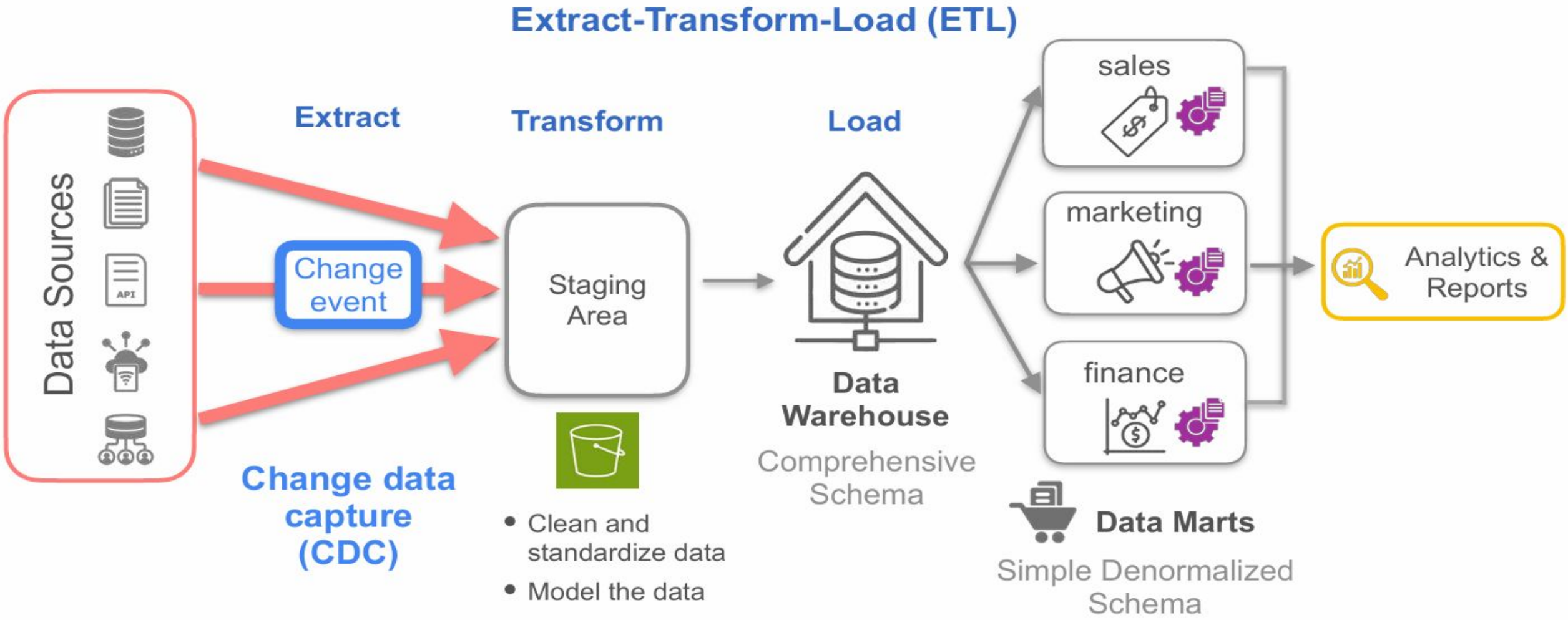
Stores current and historical data (Unlike OLTP systems)



Data Warehouse-Centric Architecture



Change Data Capture

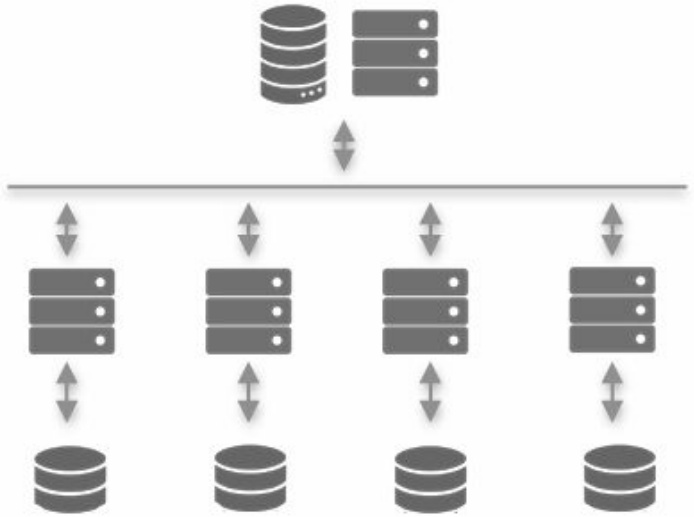


Data Warehouse Implementation

Early Data Warehouses

Big monolithic server

Data Warehouses with Massively Parallel Processing (MPP)



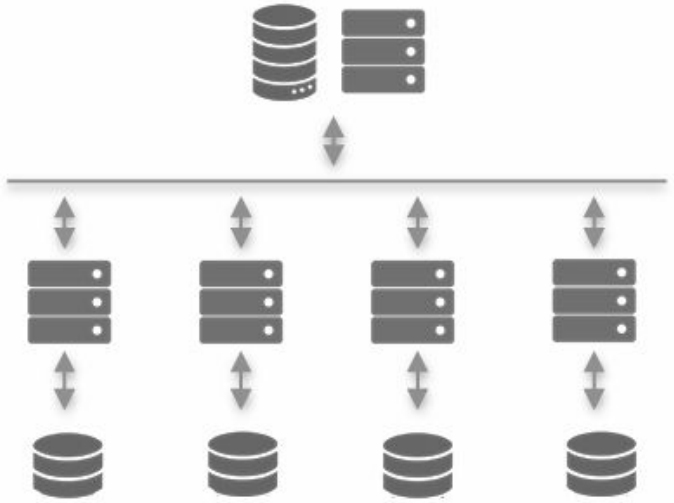
- Scans large amounts of data in parallel
- Complex configurations and requires effort to maintain

Data Warehouse Implementation

Early Data Warehouses

Big monolithic server

Data Warehouses with Massively Parallel Processing (MPP)



- Scans large amounts of data in parallel
- Complex configurations and requires effort to maintain

Modern Cloud Data Warehouses

Amazon Redshift

Google Big Query

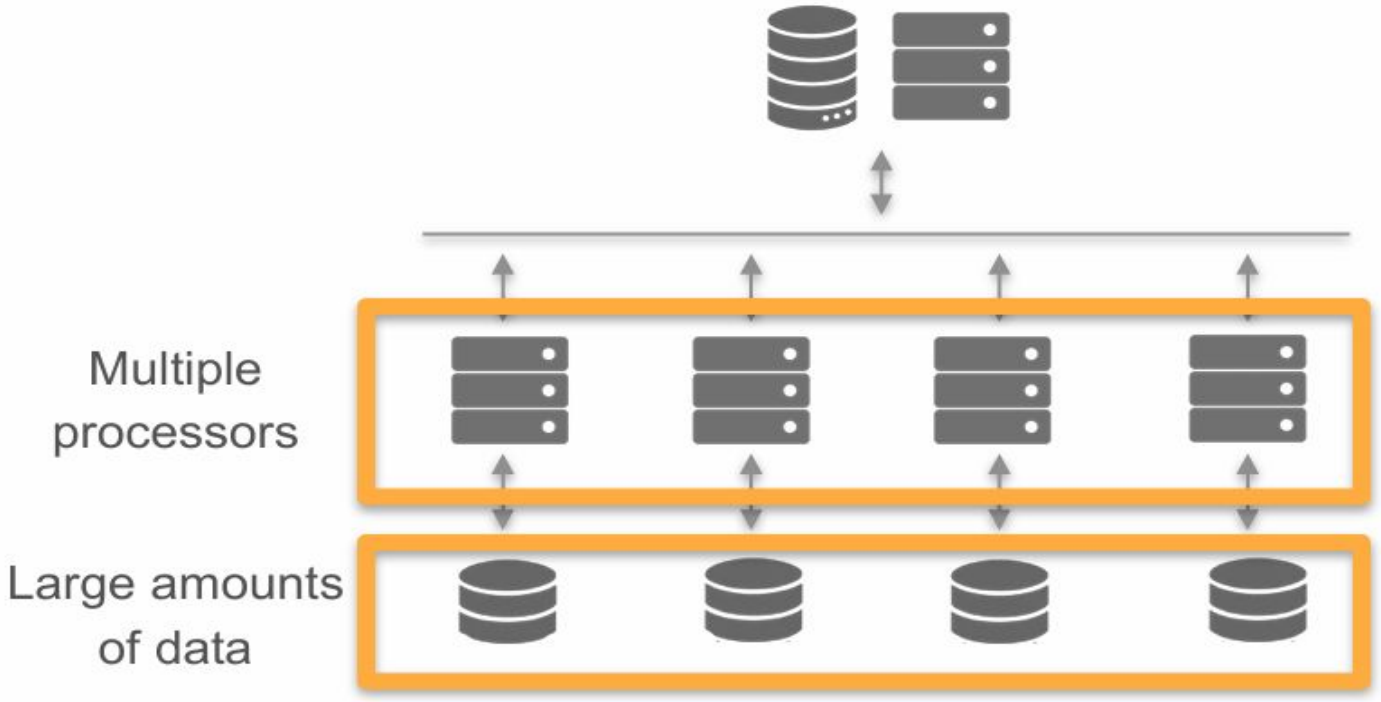
snowflake

- Separates compute from storage
- Expands the capability of MPP systems

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

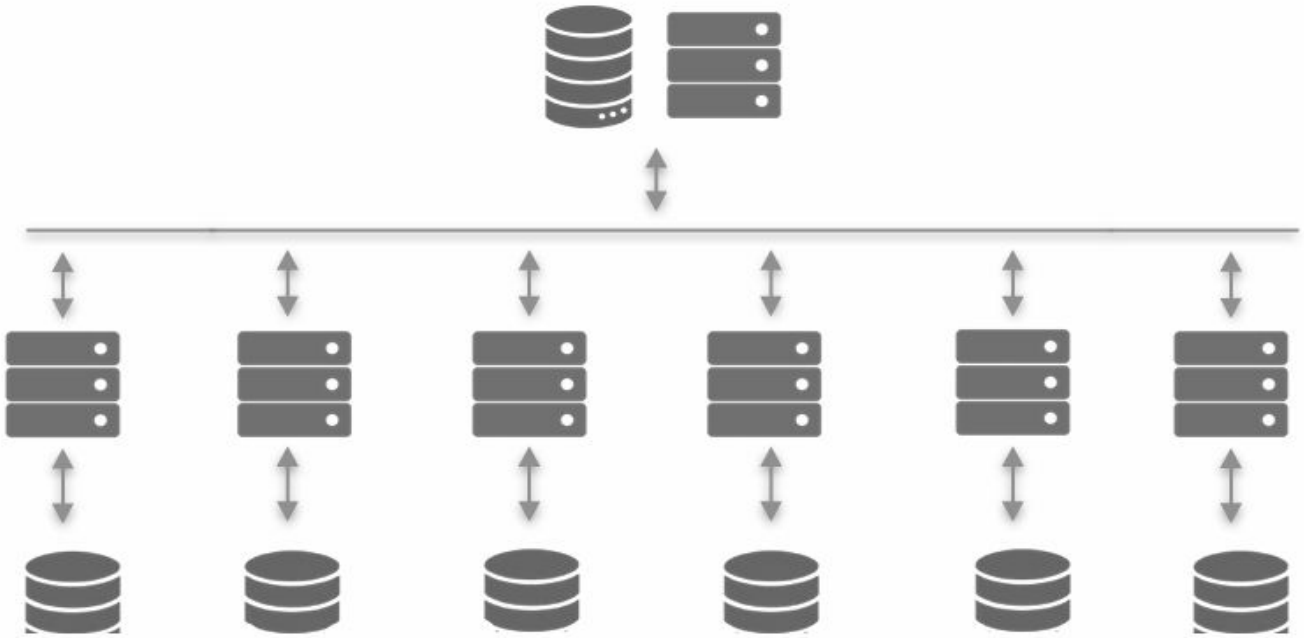
Modern Cloud Data Warehouses

Massively Parallel Processing



Massively Parallel Processing (MPP)

Massively Parallel Processing - Cloud Data Warehouses



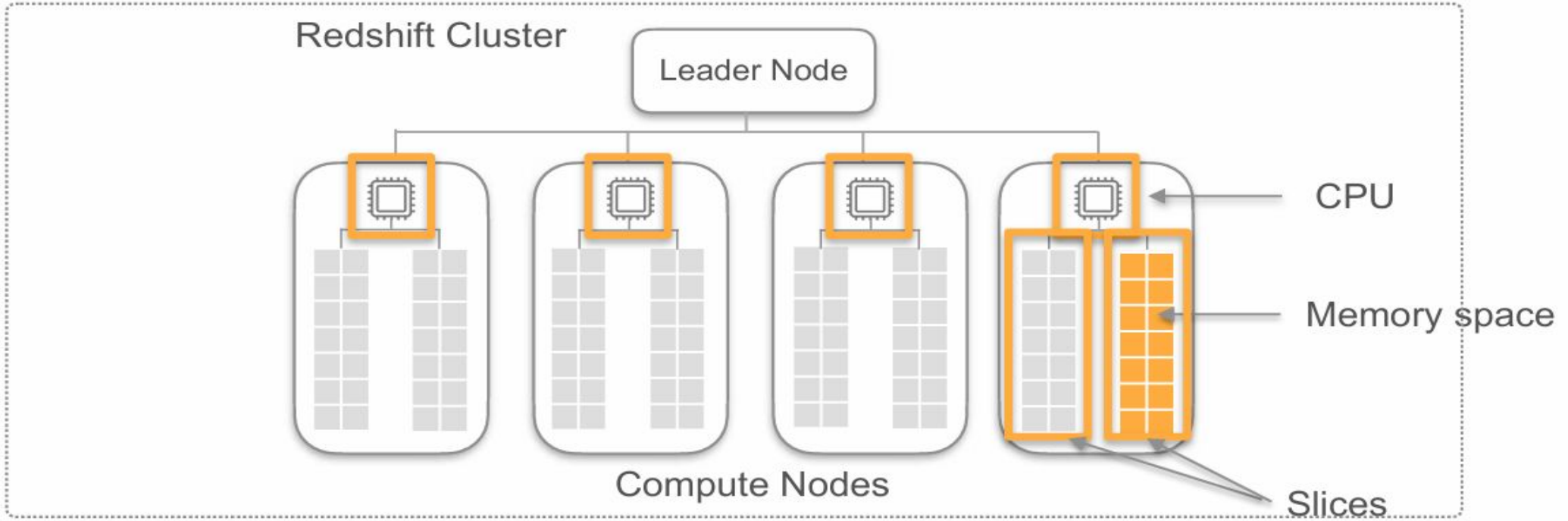
Massively Parallel Processing (MPP)

Amazon Redshift

Google Big Query

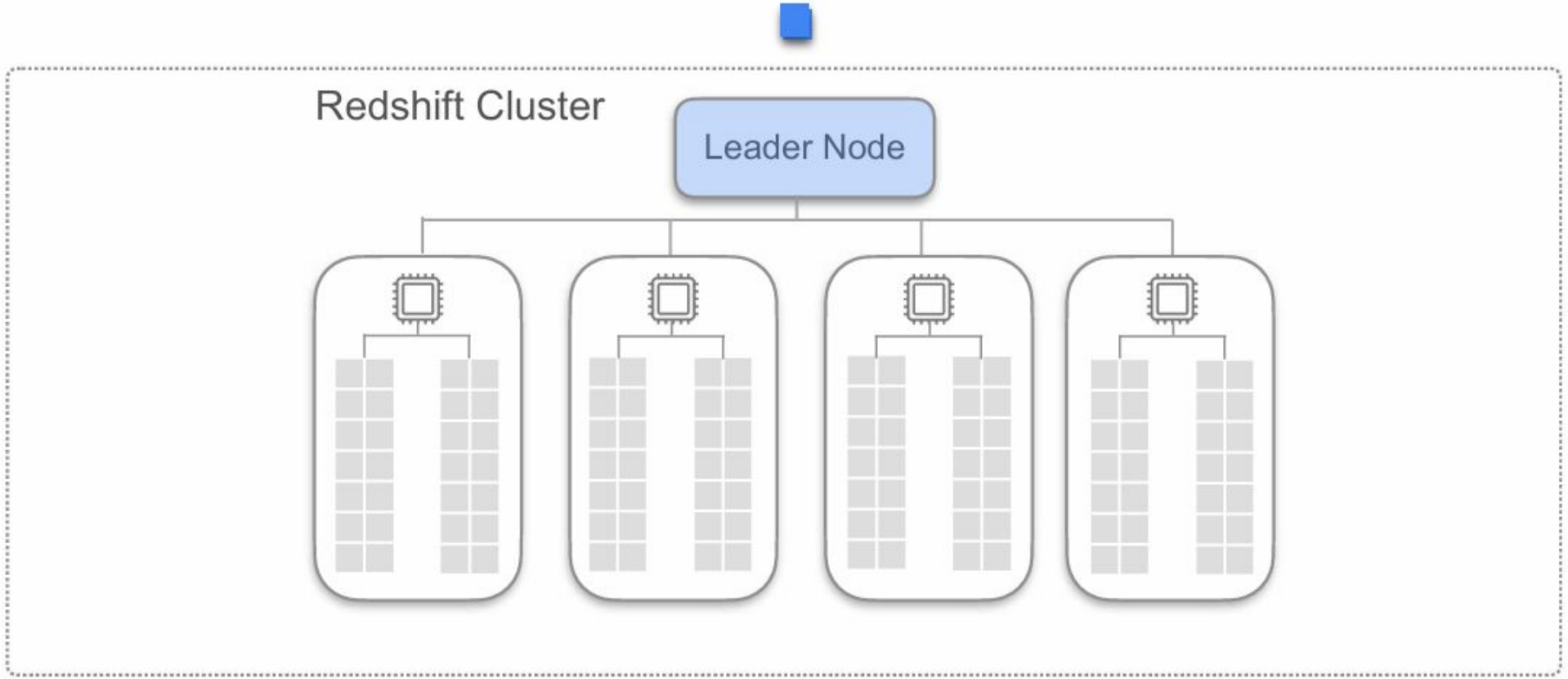
snowflake

MPP Architecture for Amazon Redshift



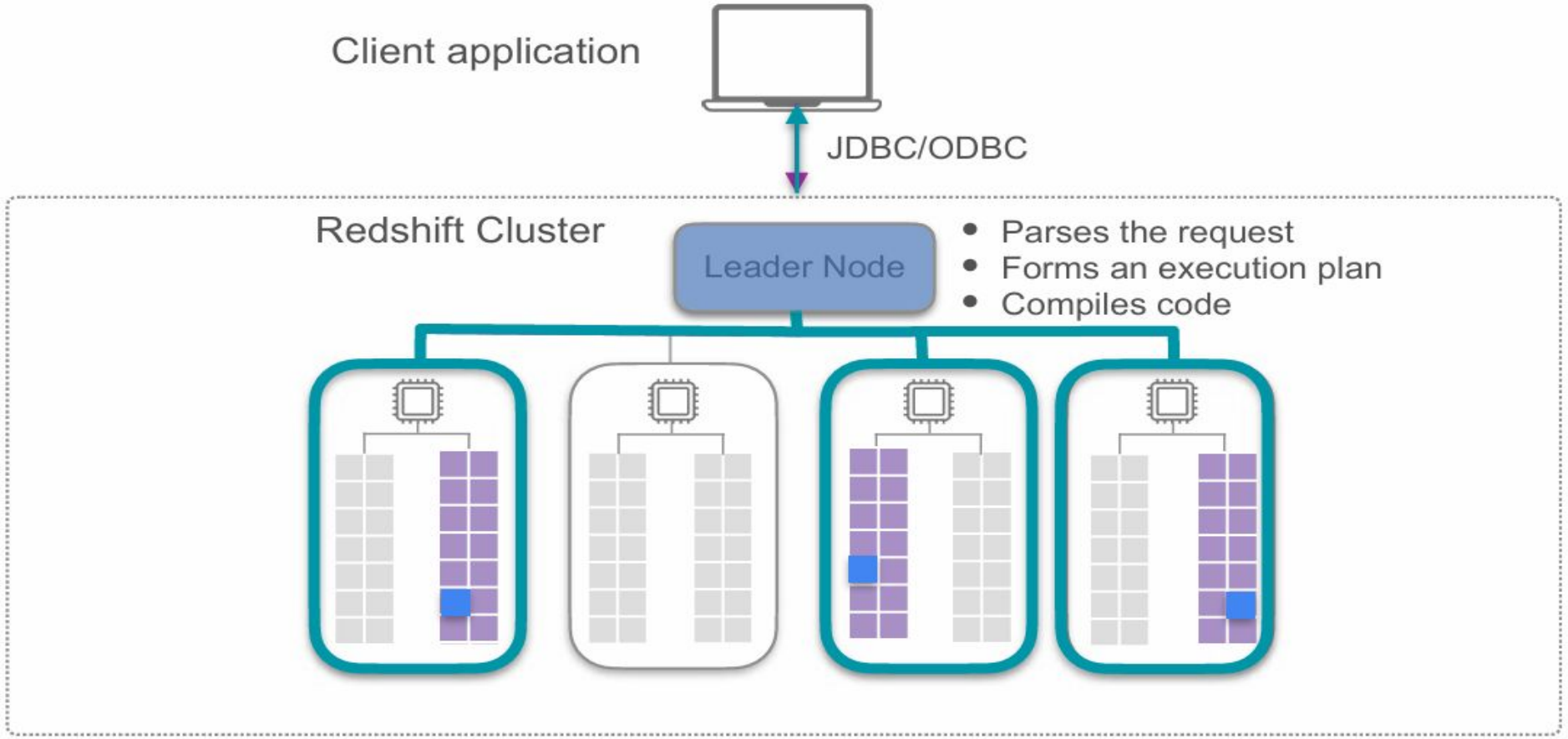
Amazon Redshift

MPP Architecture for Amazon Redshift



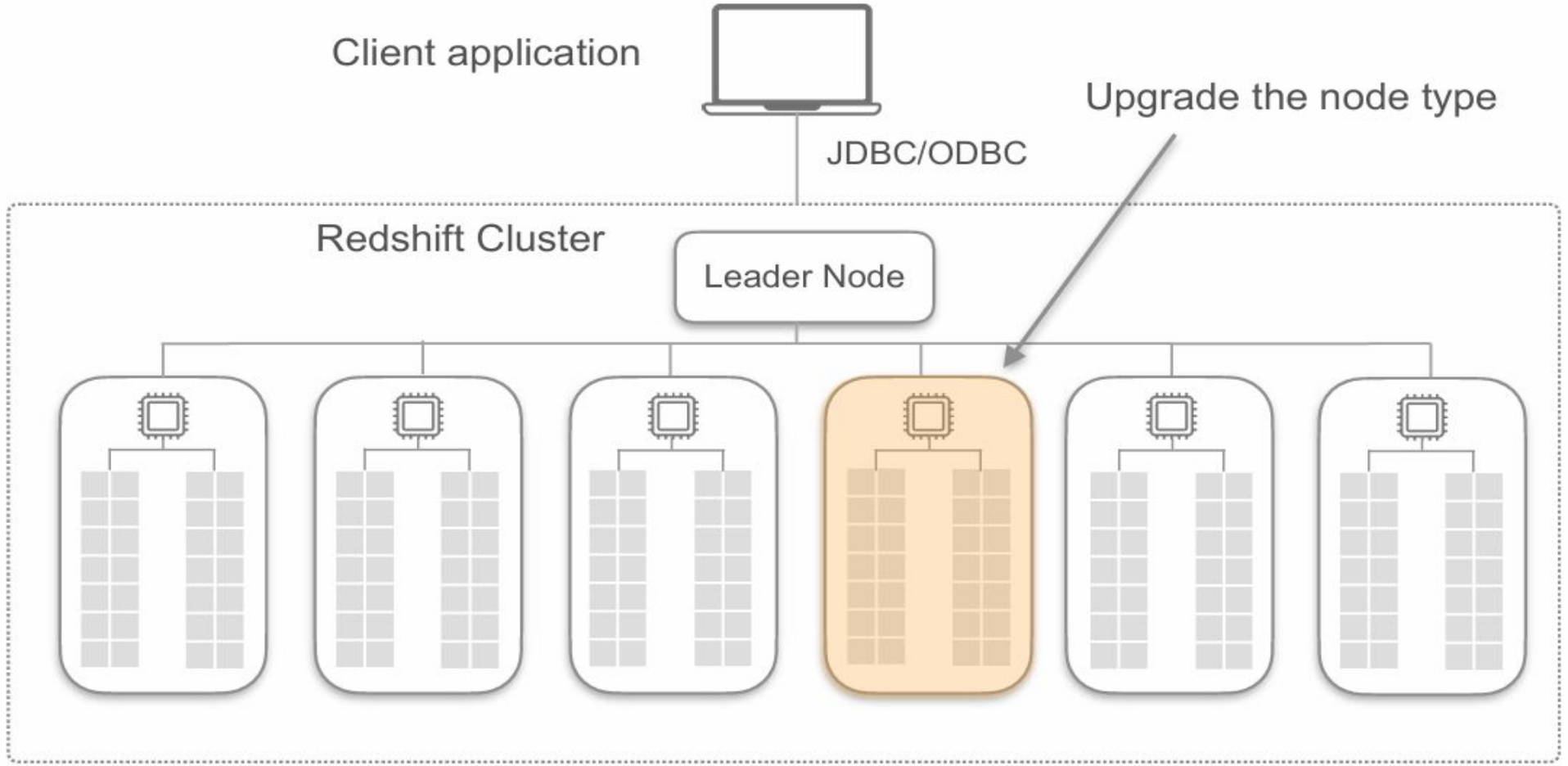
Amazon Redshift

MPP Architecture for Amazon Redshift



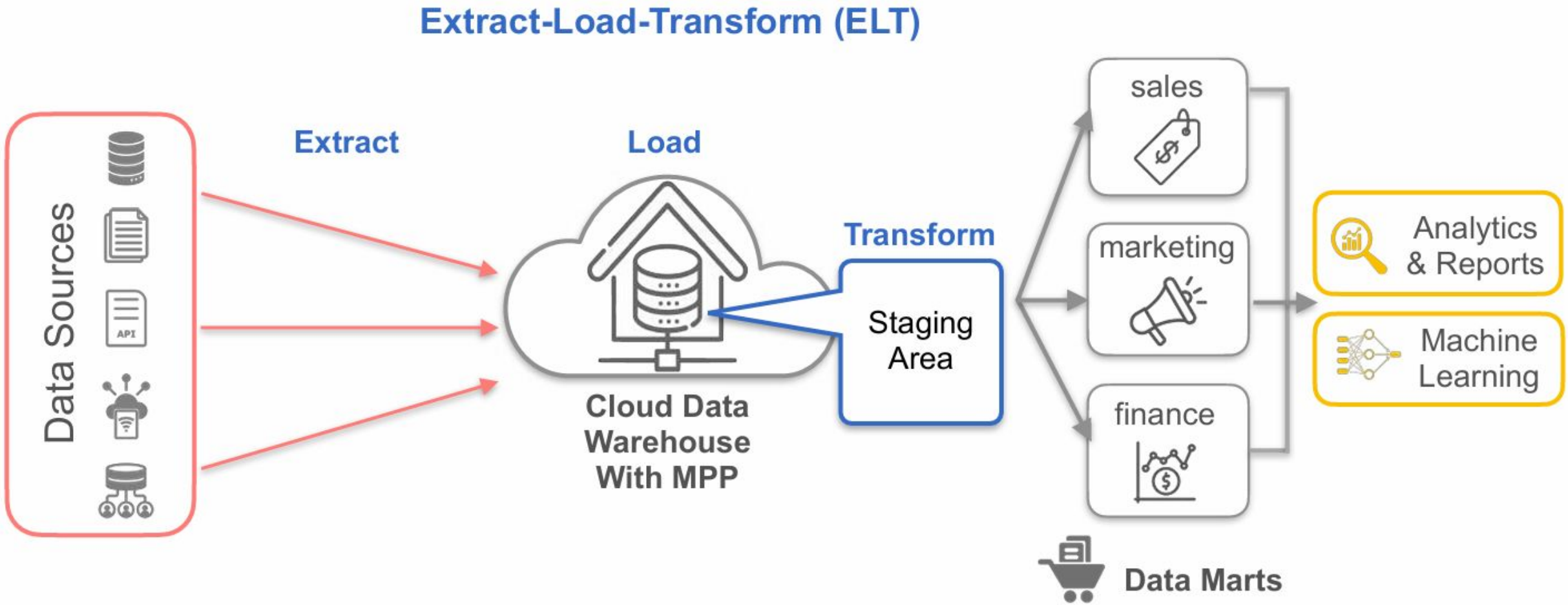
Amazon Redshift

MPP Architecture for Amazon Redshift



Amazon Redshift

Data Warehouse-Centric Architecture



Cloud Data Warehouse

Columnar Architecture

Order ID	Price	Product SKU	Quantity	Customer ID
1	40	458650	10	67t
2	23	902348	14	56t
3	45	1255893	12	87q
4	50	456829	13	98q

Facilitates high performance analytical queries

Separation of Compute and Storage

Compute

Storage

Cloud Data Warehouse

Traditional Data Warehouse	Cloud Data Warehouse
• Stored data is highly structured • Data modeled to enable analytical queries	• Stored data is highly structured • Data modeled to enable analytical queries • High processing from MPP • Columnar Storage • Separation of storage and compute **Efficiently stores and processes data for high-volume analytical workloads**

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Data Lakes – Key Architectural ideas

Data Lake

- Central repository for storing large volumes of data
- No fixed schema or predefined set of transformations
- **Schema-on-read pattern:**
  - Reader determines the schema when reading the data

Data Lake 1.0

Combined different storage and processing technologies

Storage

Processing Tools

Apache Pig

Shortcomings of Data Lake 1.0

Data Swamp

- No proper data management
- No data cataloging
- No data discovery tools
- No guarantee on the data integrity and quality

Shortcomings of Data Lake 1.0

Data Swamp

- No proper data management
- No data cataloging
- No data discovery tools
- No guarantee on the data integrity and quality

Write-only storage

- Data Manipulation Language (DML) operations were painful to implement

delete or update rows
Create a new table

- Difficult to comply with data regulations

Shortcomings of Data Lake 1.0

Data Swamp

- No proper data management
- No data cataloging
- No data discovery tools
- No guarantee on the data integrity and quality

Write-only storage

- Data Manipulation Language (DML) operations were painful to implement

delete or update rows

Create a new table

- Difficult to comply with data regulations

No schema management and data modeling

- Hard to process stored data
- Data not optimized for query operations such as joins

Joins

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Next-Generation Data Lakes

Data Zones

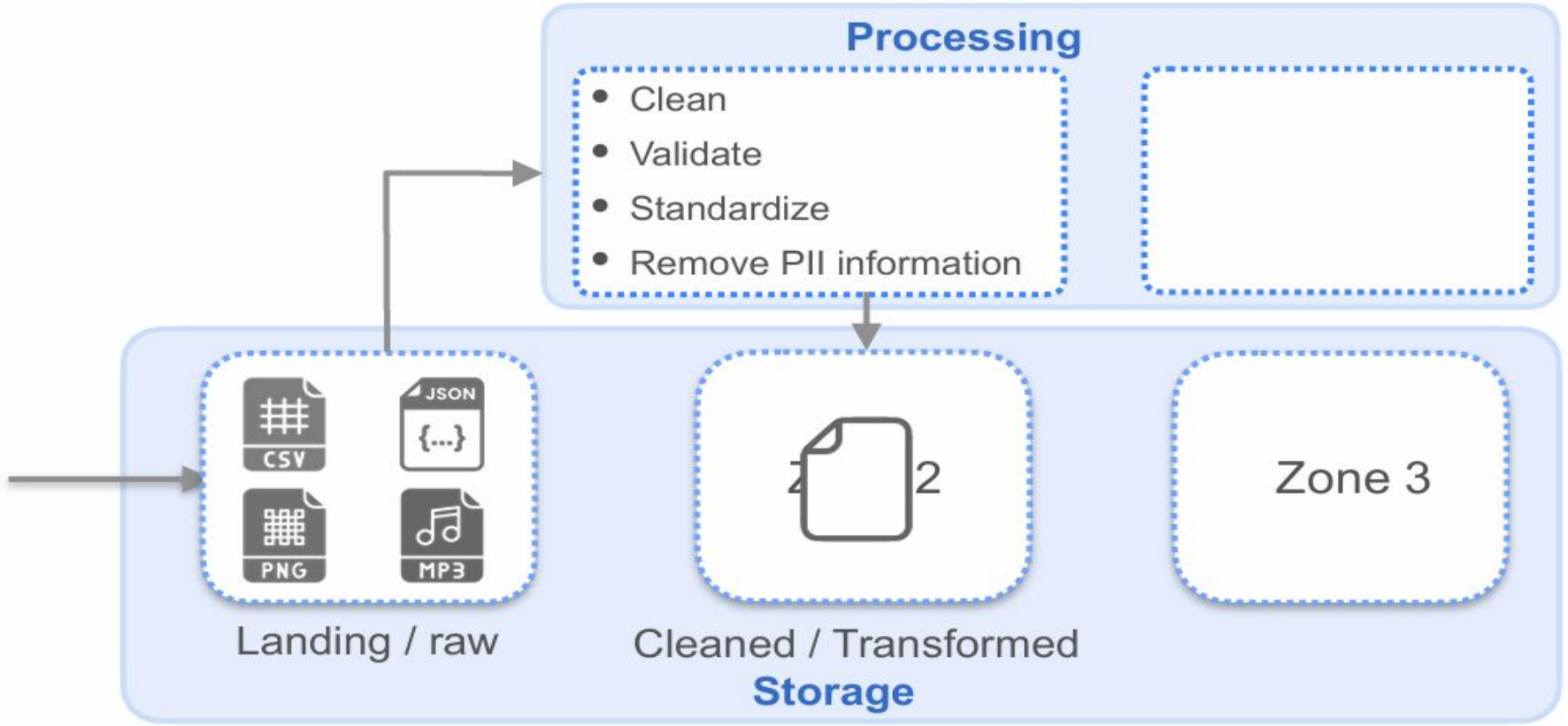
Used to organize data in a data lake, where each zone houses data that has been processed to varying degrees

Data Zones

Used to organize data in a data lake, where each zone houses data that has been processed to varying degrees

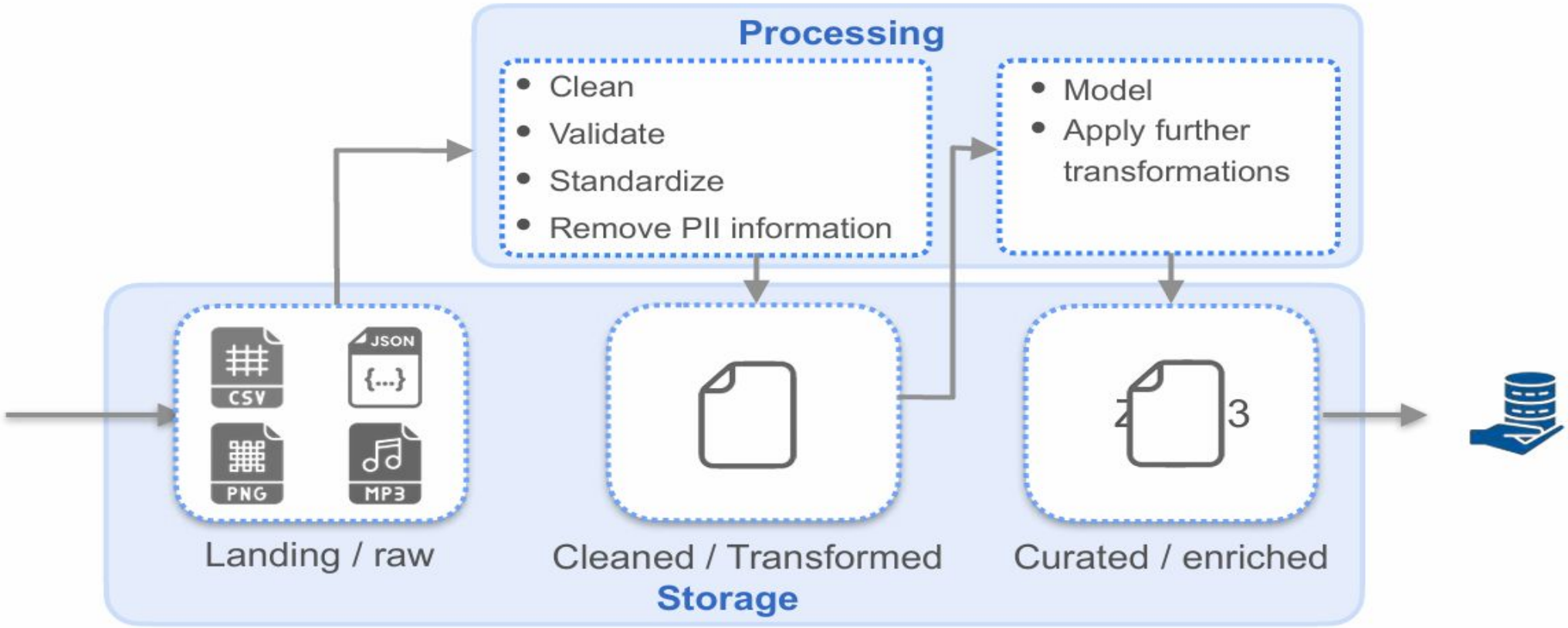
Data Zones

Used to organize data in a data lake, where each zone houses data that has been processed to varying degrees



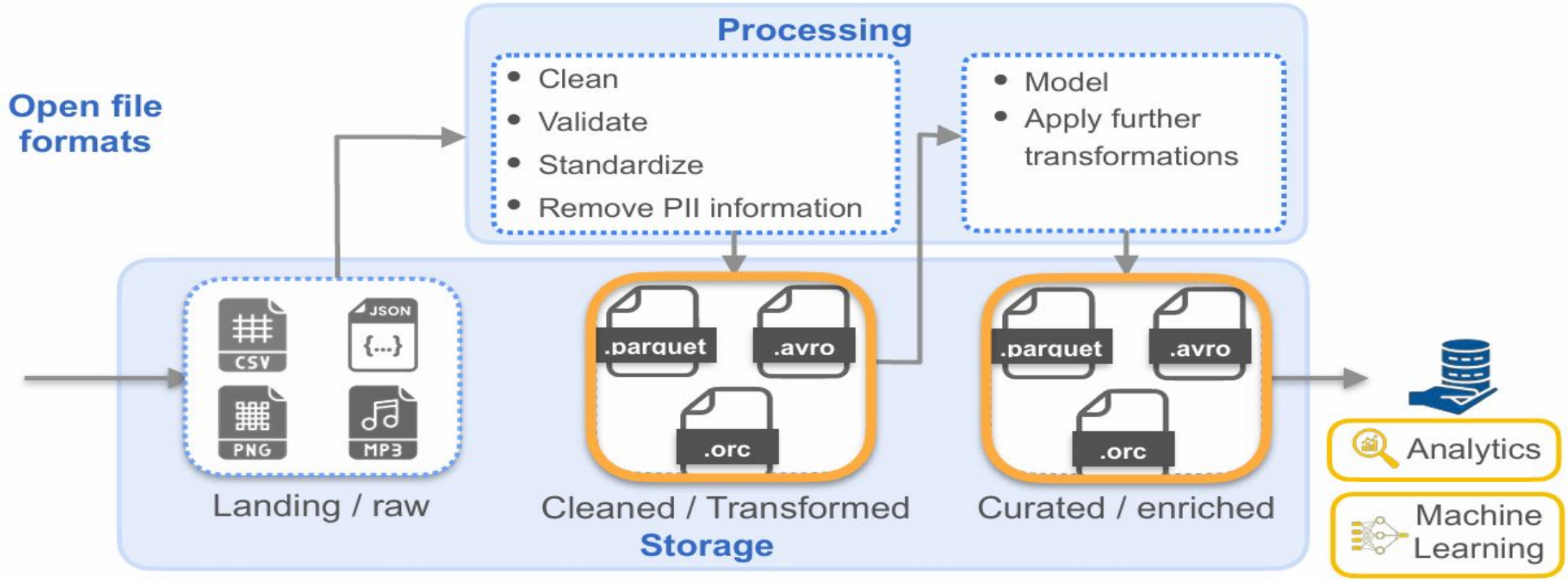
Data Zones

Used to organize data in a data lake, where each zone houses data that has been processed to varying degrees



Data Zones

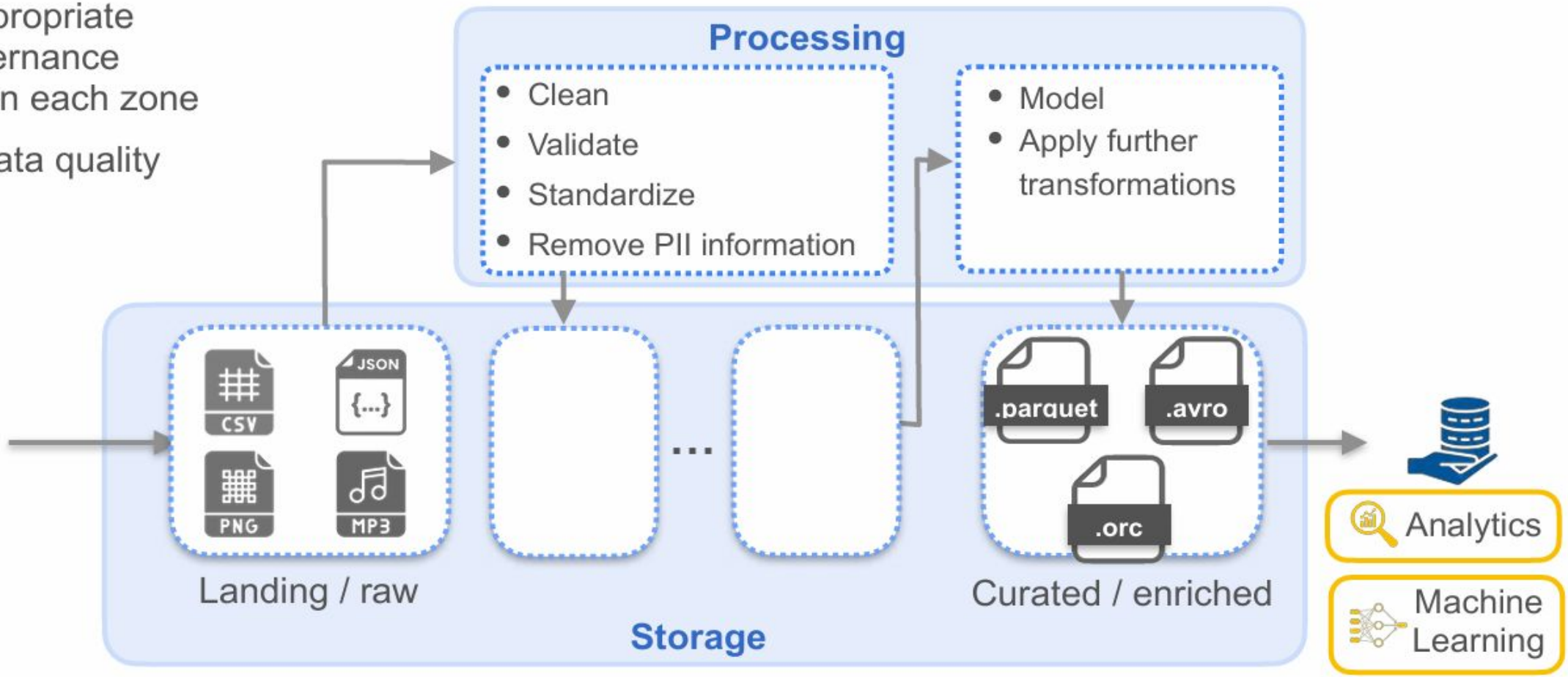
Used to organize data in a data lake, where each zone houses data that has been processed to varying degrees



Data Zones

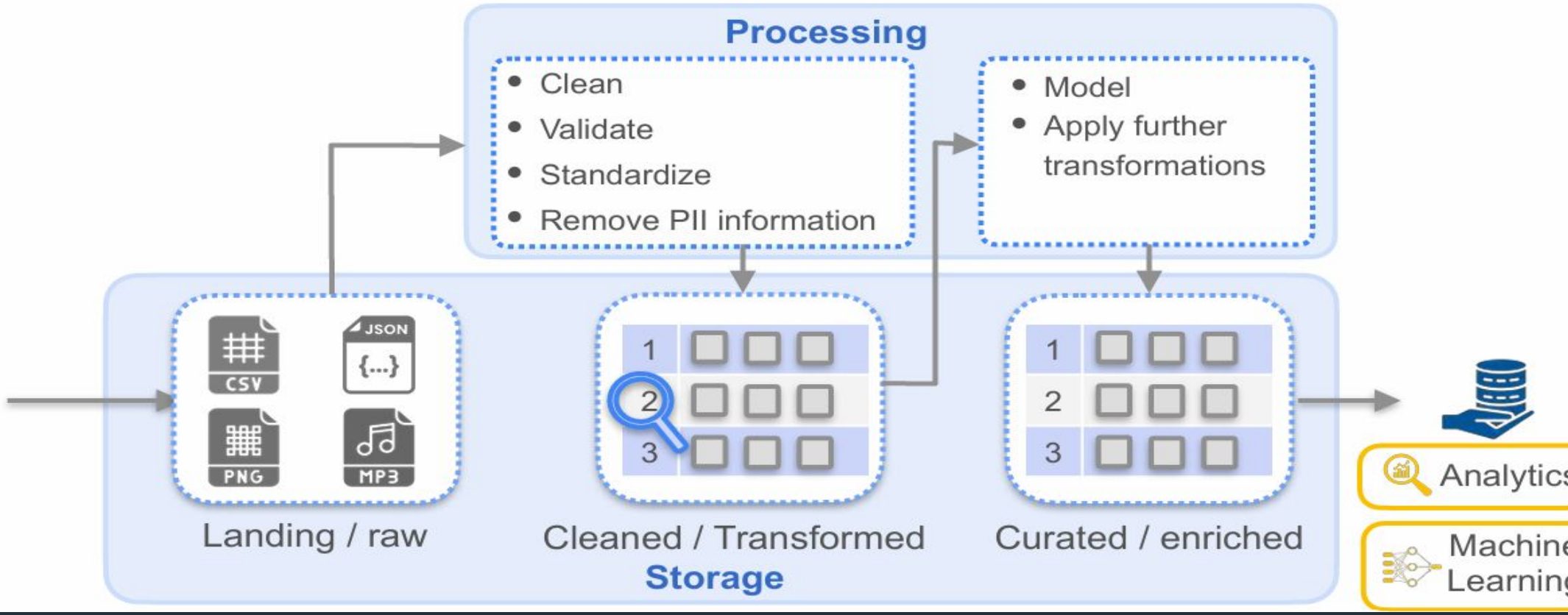
Used to organize data in a data lake, where each zone houses data that has been processed to varying degrees

- Apply appropriate data governance policies on each zone
- Ensure data quality



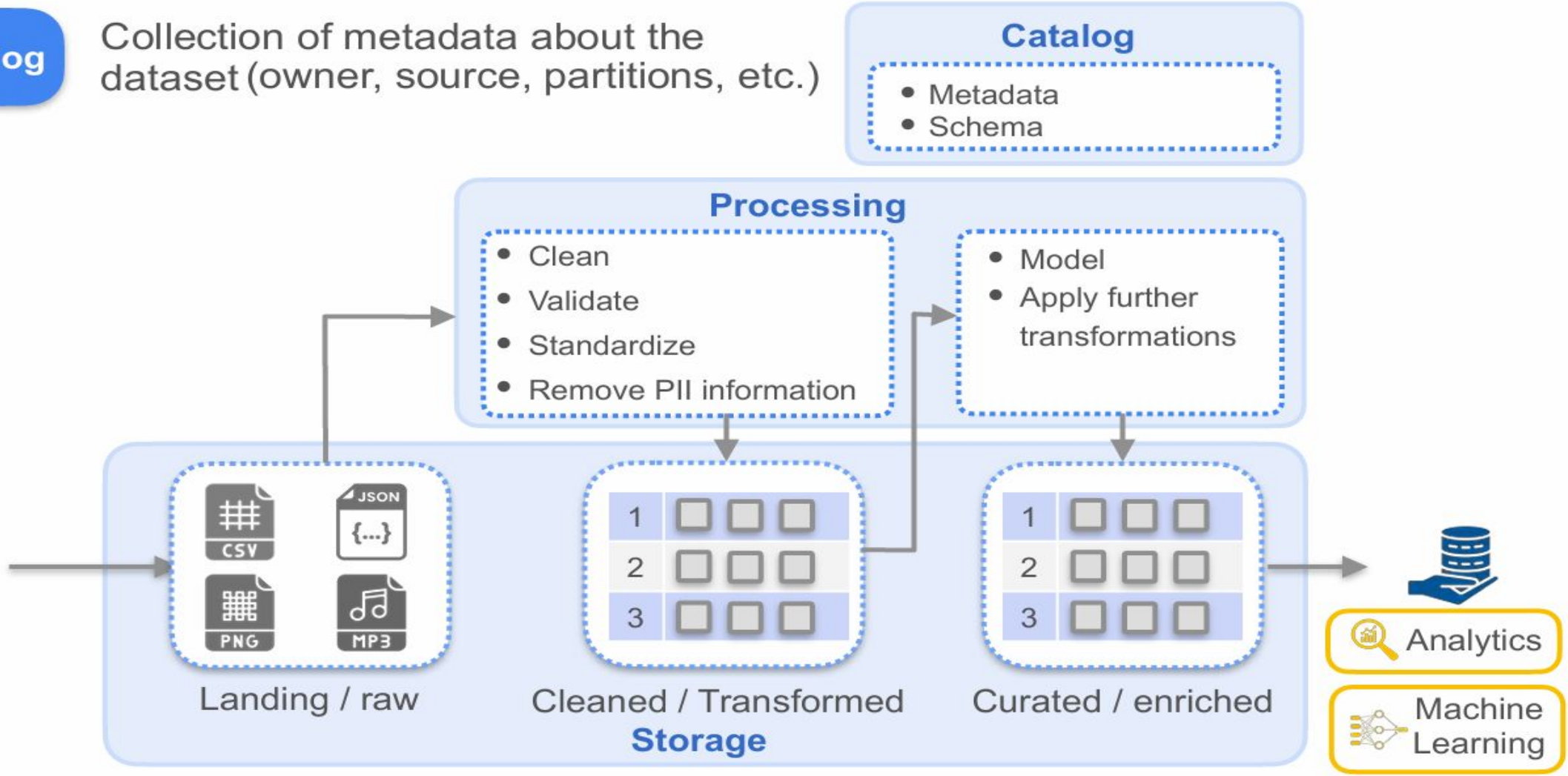
Data Partitioning

Divide a dataset into smaller, more manageable parts based on a set of criteria (e.g. time, date, location recorded in the data)



Data Catalog

Collection of metadata about the dataset (owner, source, partitions, etc.)



Separate Data Lakes and Data Warehouses

Data Lake

Low-Cost Storage

Store large amounts of data

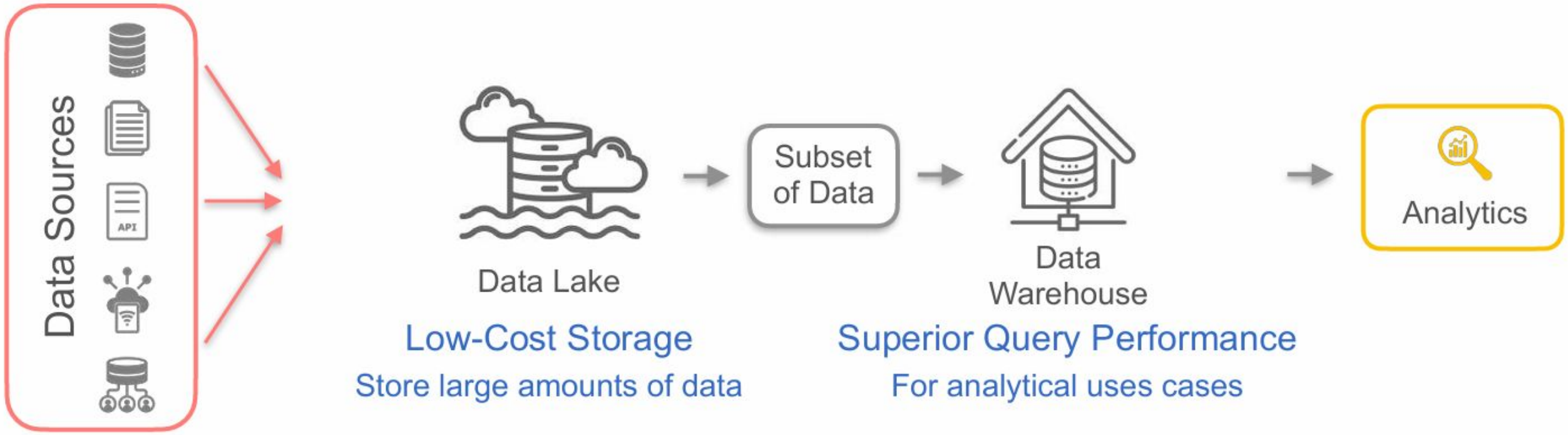
and

Data Warehouse

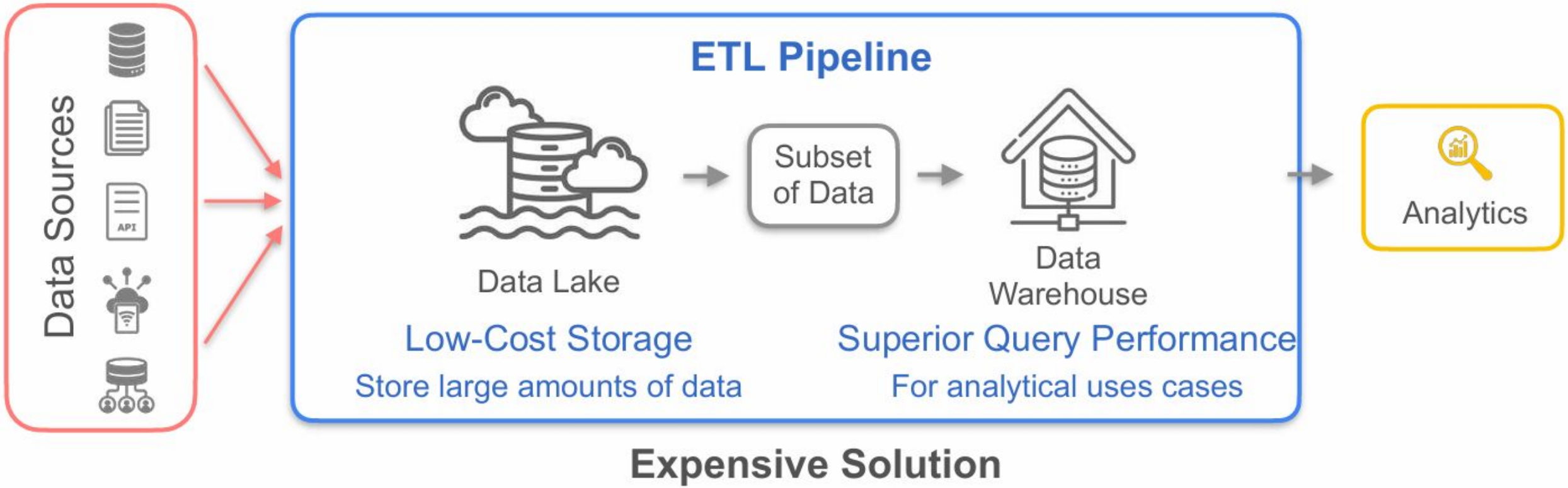
Superior Query Performance

For analytical uses cases

Separate Data Lakes and Data Warehouses

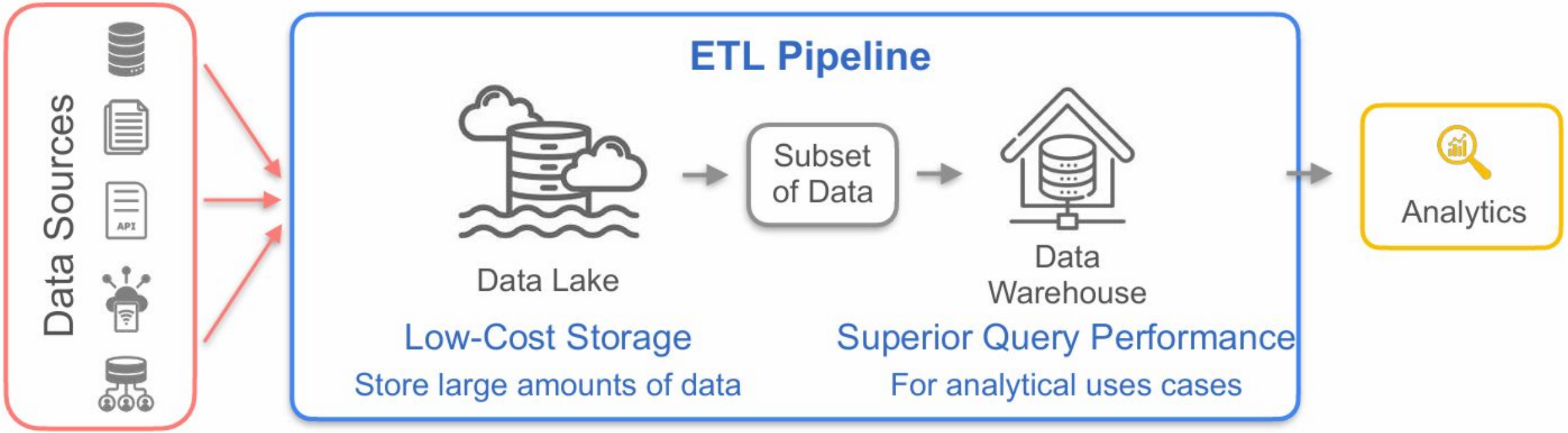


Separate Data Lakes and Data Warehouses



Expensive Solution

Separate Data Lakes and Data Warehouses



Expensive Solution

- Can introduce bugs/failures
- Can cause issues with data quality, duplication, consistency

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Data Lakehouse

Data Lakehouse

Data Lake

- Flexibility
- Low-Cost Storage

+

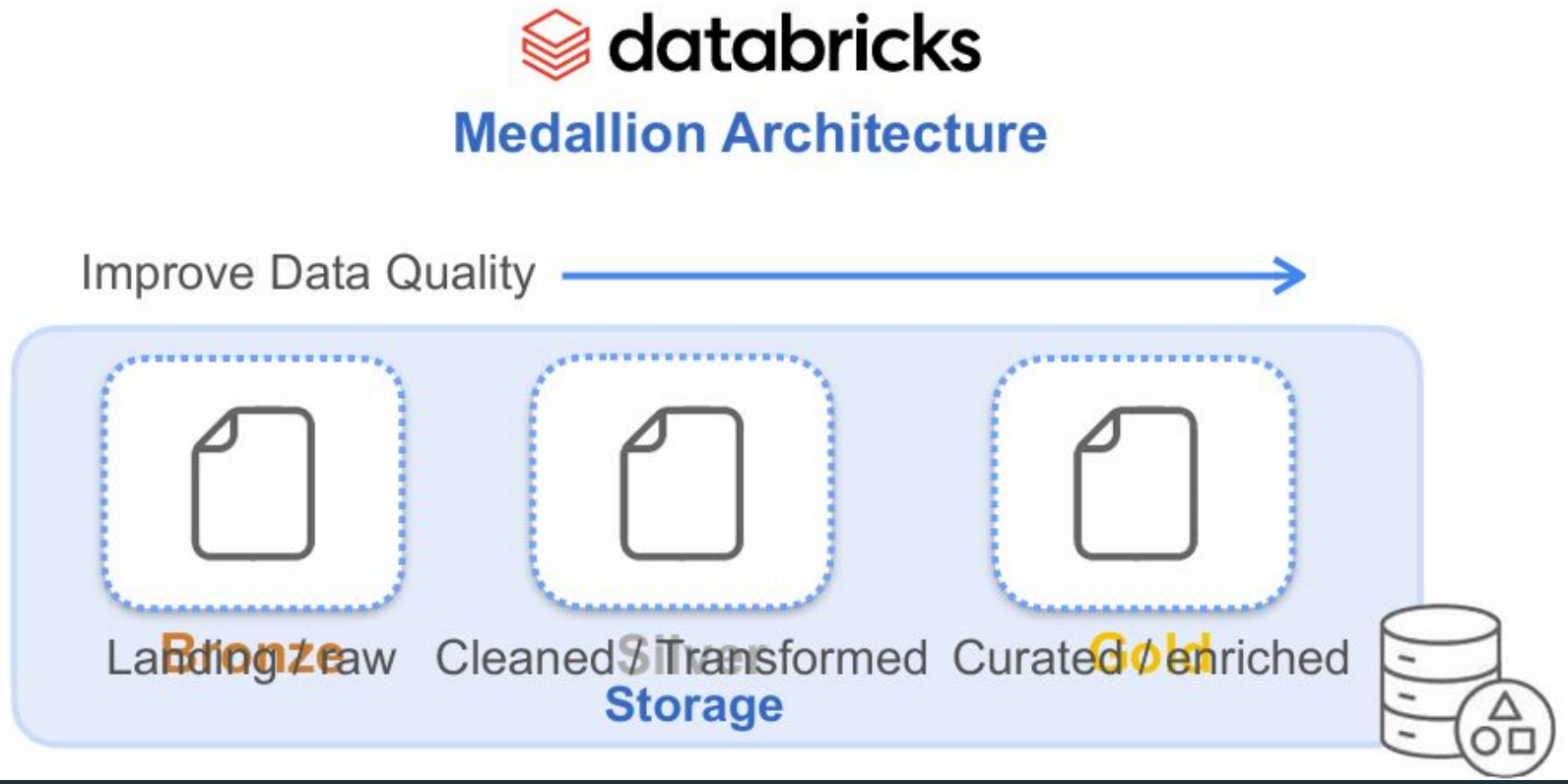
Data Warehouse

- Superior query performance
- Robust data management

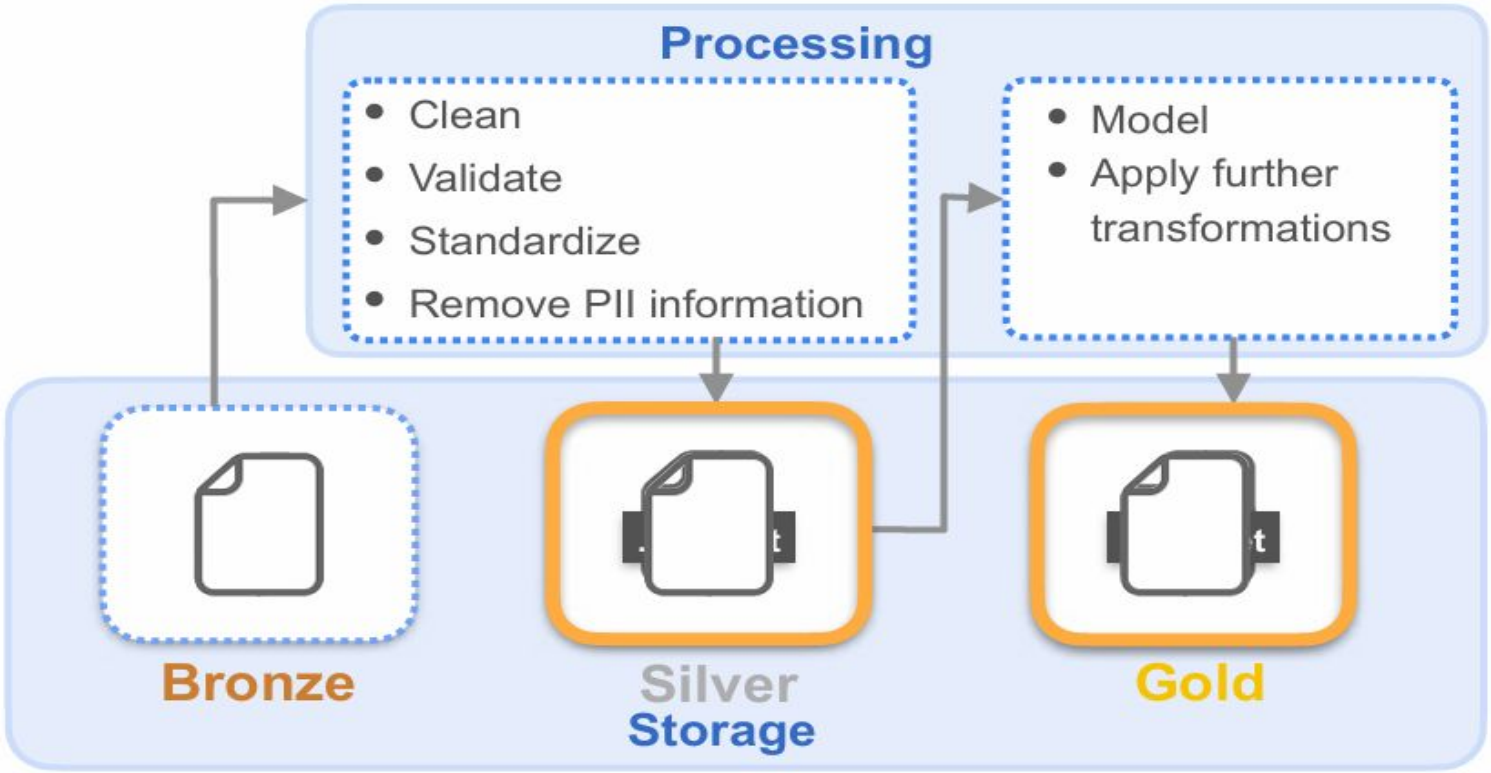
=

Data Lakehouse

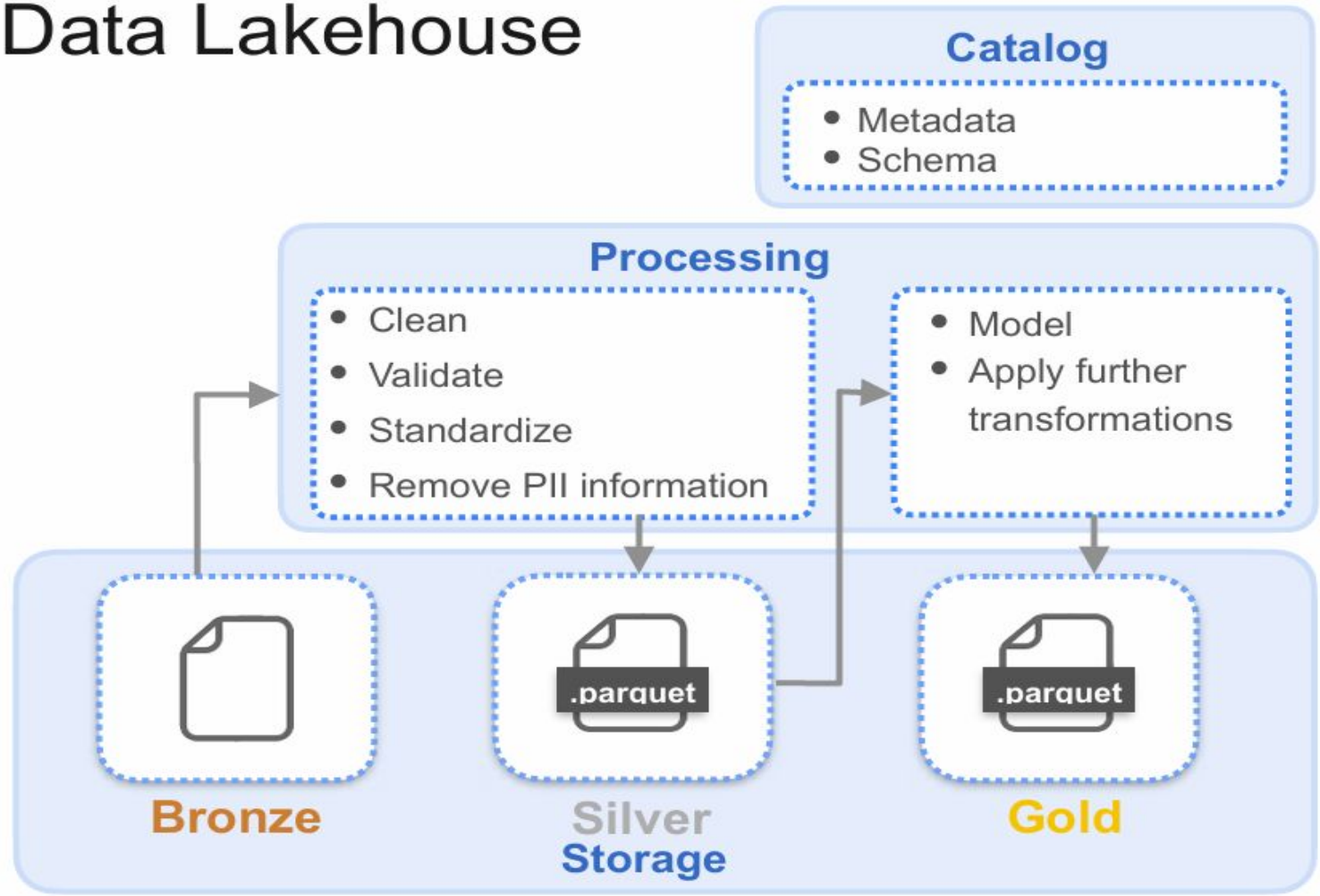
Data Lake Features



Data Lake Features

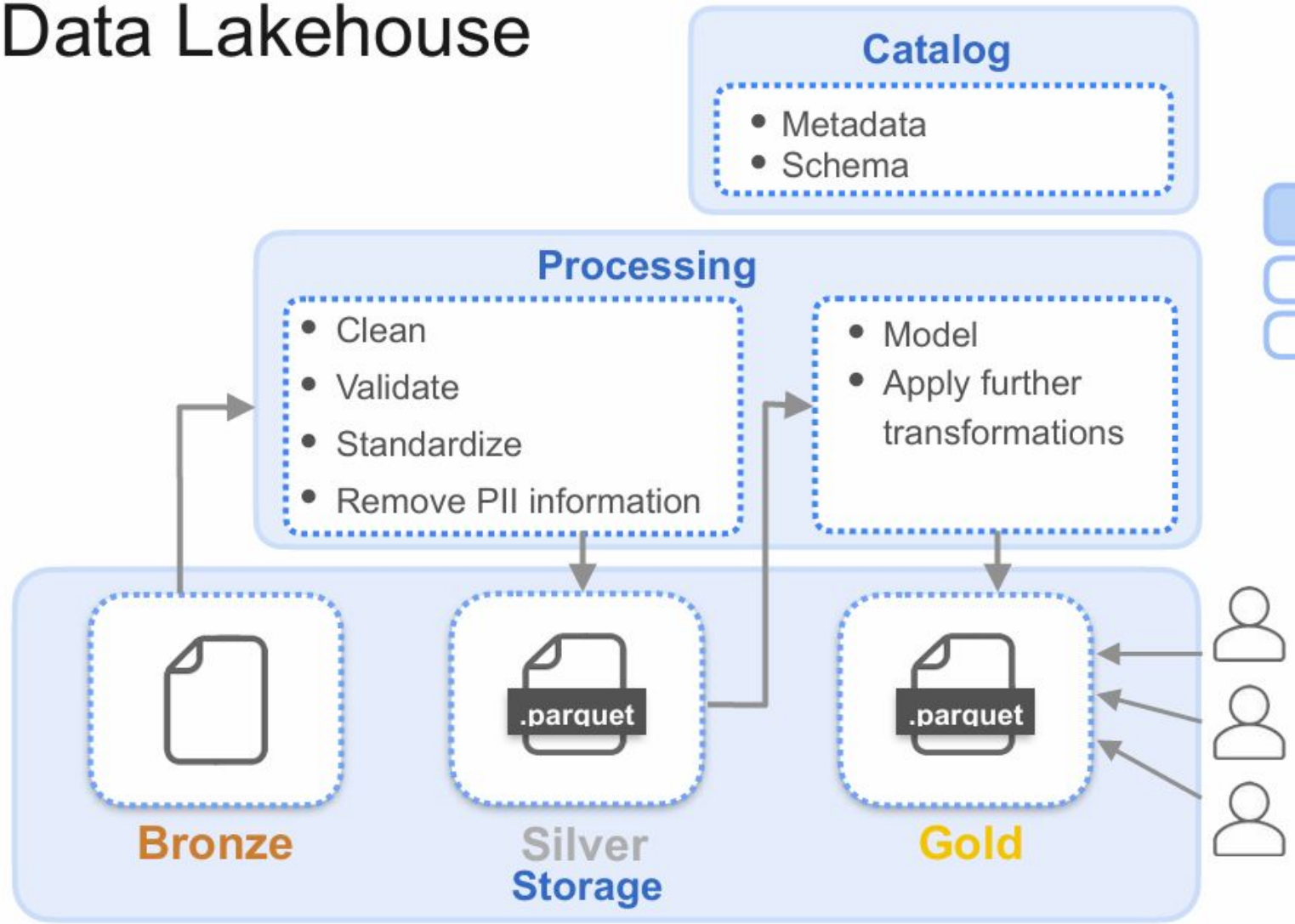


Data Lakehouse



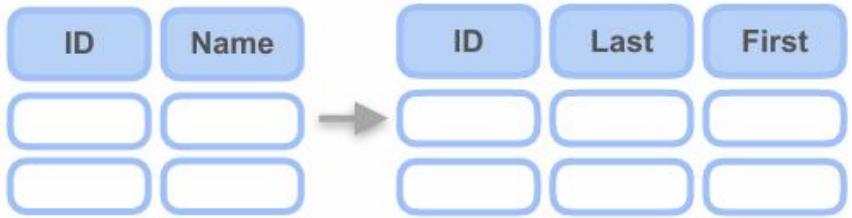
Data Management from Data Warehouses

Data Lakehouse



Data Management from Data Warehouses

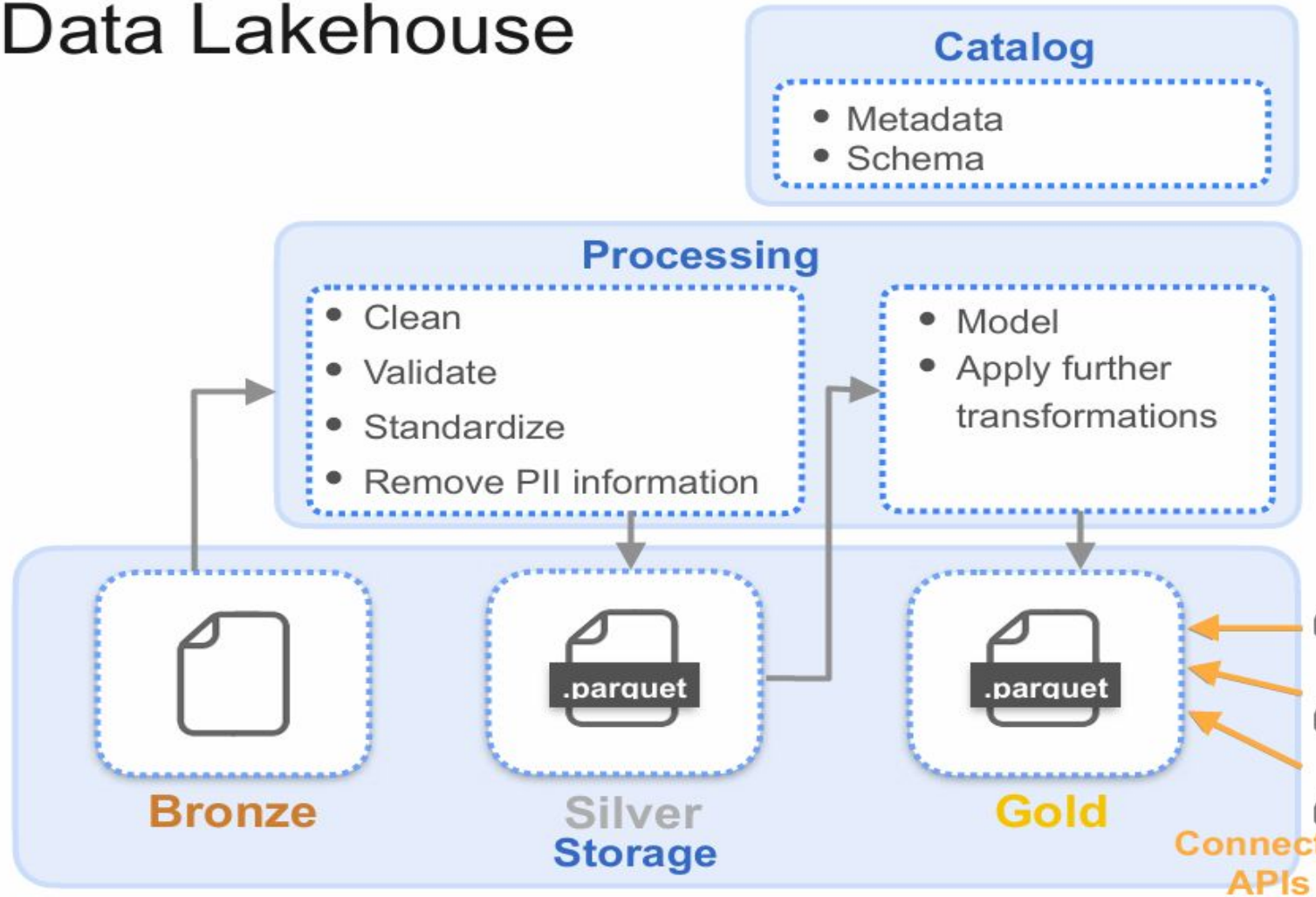
Schema Enforcement



ACID

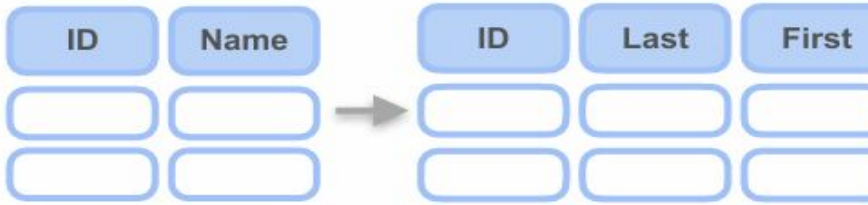
- **A**tomic, **C**onsistent, **I**solated, **D**urable
- Concurrent read, insert, update, delete

Data Lakehouse



Data Management from Data Warehouses

Schema Enforcement



ACID

- Atomic, Consistent, Isolated, Durable
- Concurrent read, insert, update, delete

Data Governance & Security

- Robust access controls, data auditing capabilities, data lineage
- Incremental updates & deletions
- Rollback to or access any version of your historical data

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Date Lakehouse Implementation

Open Table Formats

Open Table Formats

Specialized storage formats that add transactional features to your data lakehouse

- Allows you to update and delete records
- Supports ACID principles

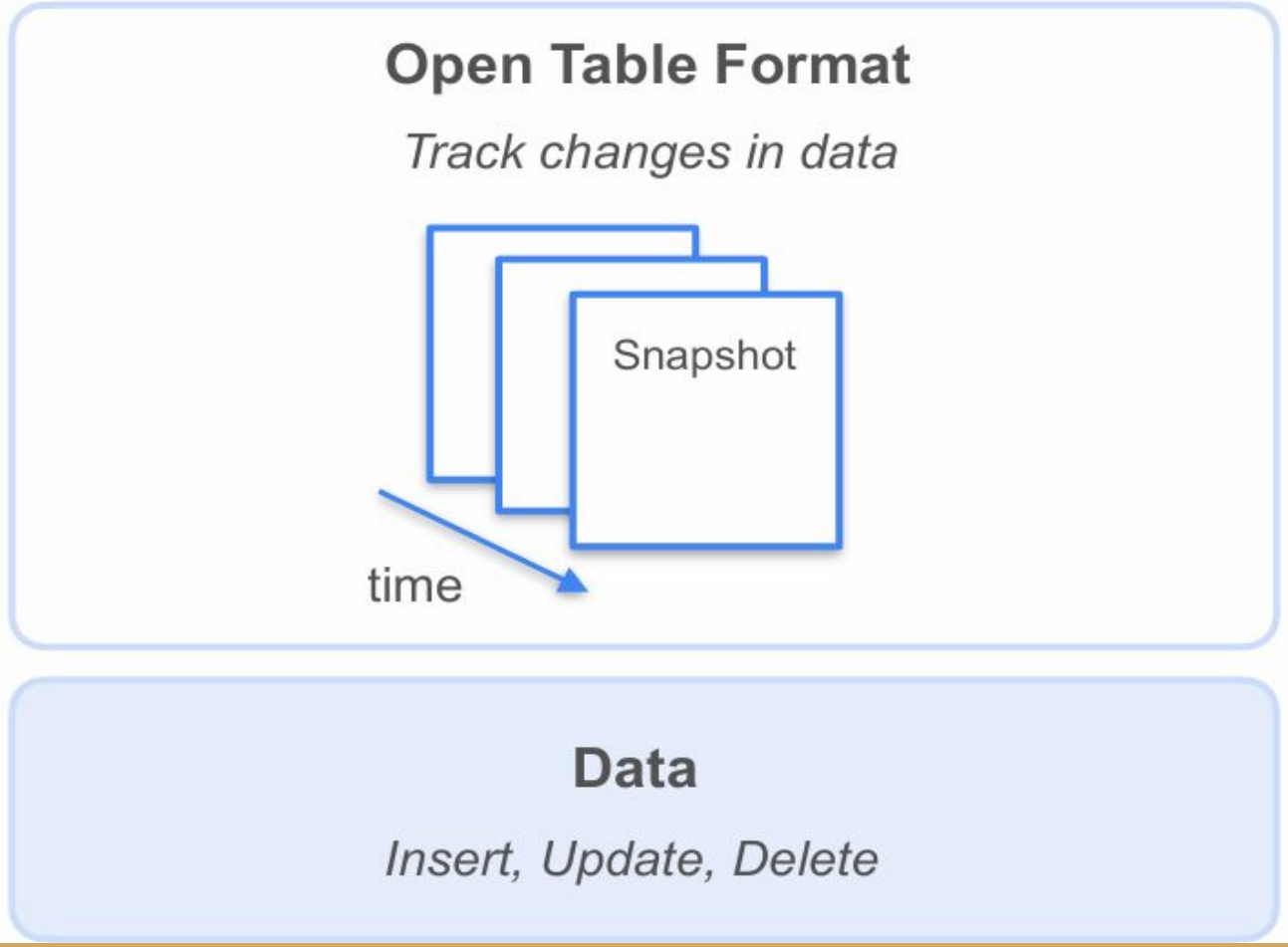
DELTA LAKE

ICEBERG

Apache hudi

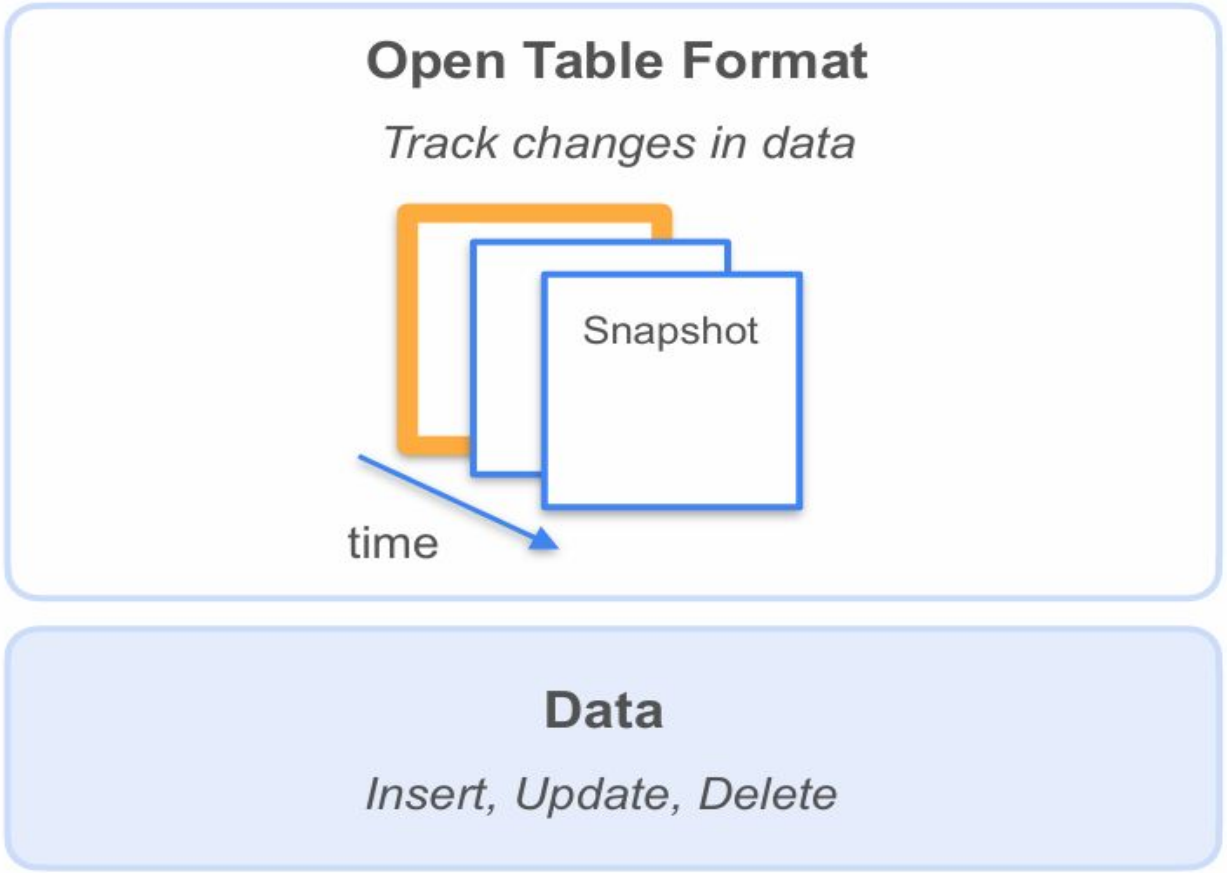
Hadoop
Update
Delete
Incremental

Open Table Formats



Snapshot:
Reflects the state of the data at a given time

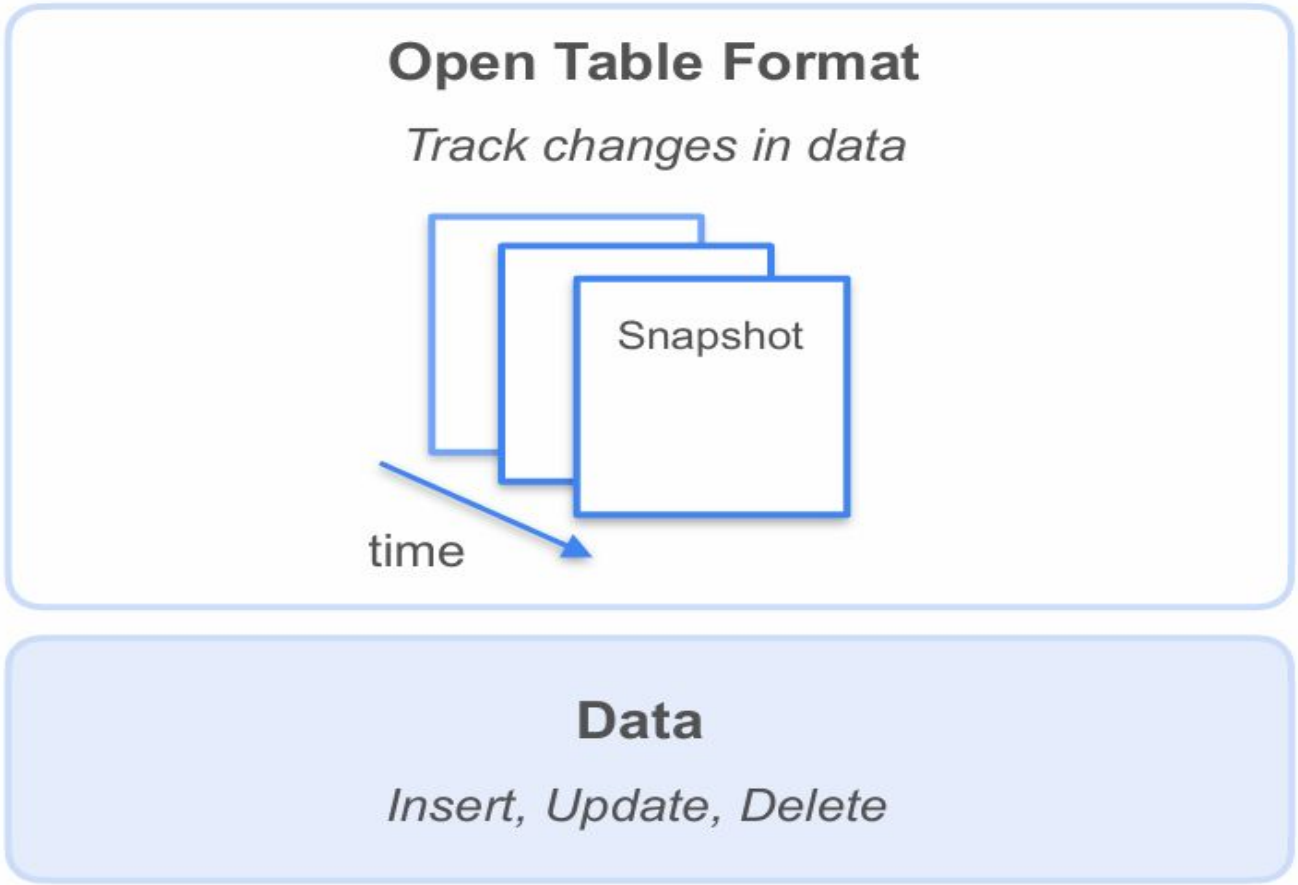
Open Table Formats



Snapshot:
Reflects the state of the data at a given time

Time Travel:
Query any previous version of a table

Open Table Formats

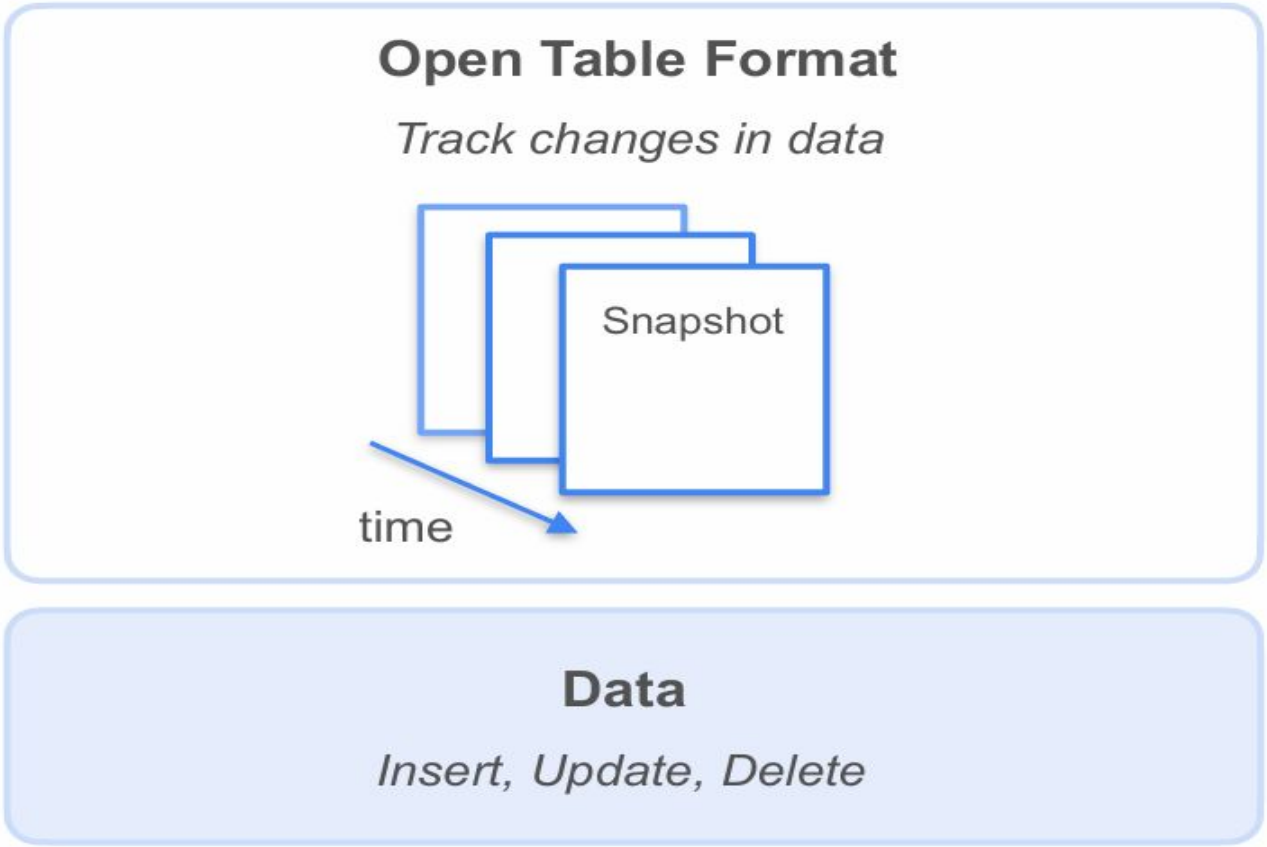


Snapshot:
Reflects the state of the data at a given time

Time Travel:
Query any previous version of a table

Schema & Partition Evolution:
Ability to query the data even if you make changes to the schema or partitioning

Open Table Formats

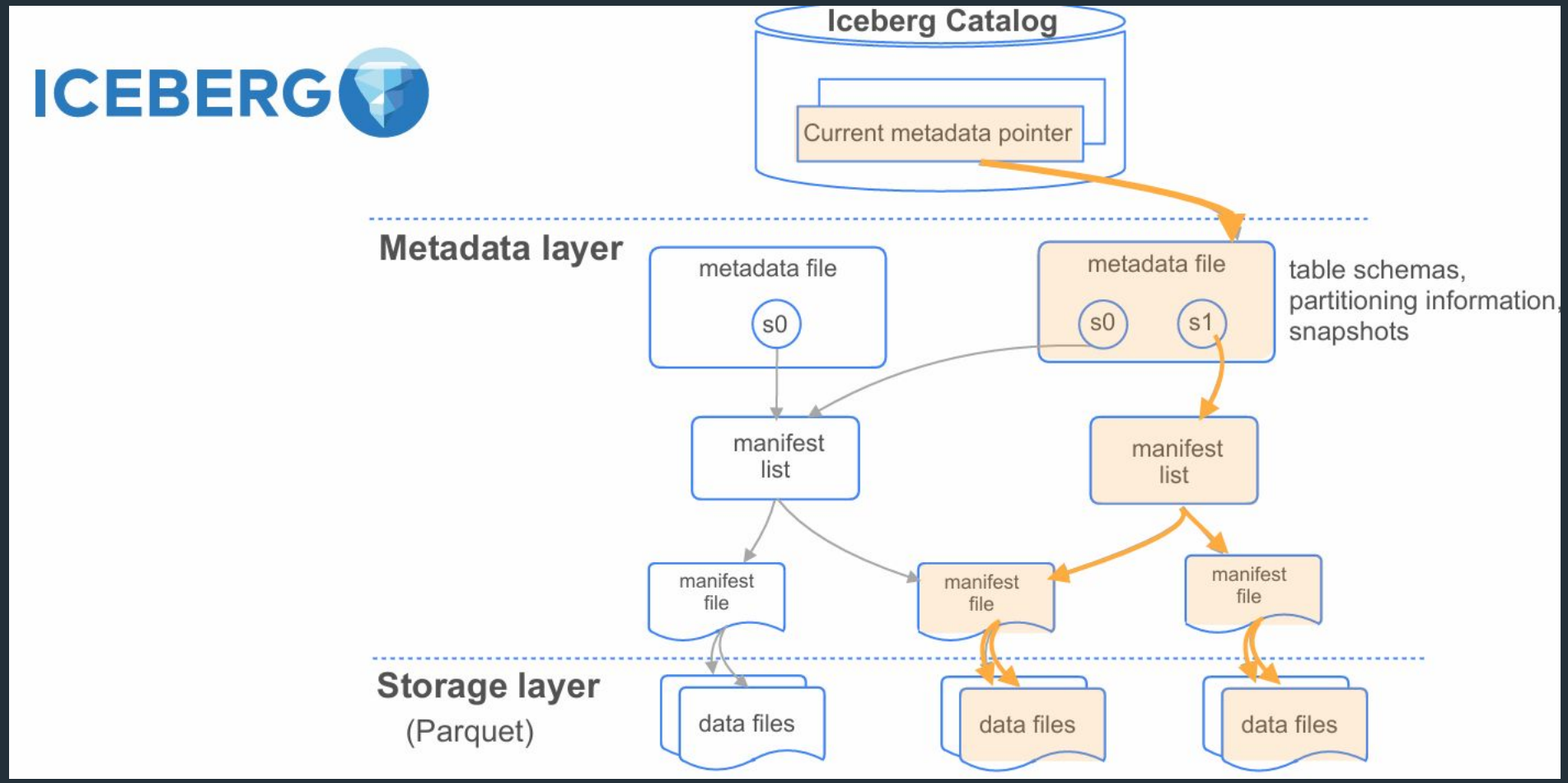


Snapshot:
Reflects the state of the data at a given time

Time Travel:
Query any previous version of a table

Schema & Partition Evolution:
Ability to query the data even if you make changes to the schema or partitioning

Open Source
Different query engines can access the data


ICEBERG
Iceberg Catalog
Current metadata pointer
Metadata layer
metadata file
s0
metadata file
s0
s1
table schemas,
partitioning information,
snapshots
manifest list
manifest list
manifest file
manifest file
manifest file
Storage layer
(Parquet)
data files
data files
data files

Storage Options

Production Database	Data Warehouse		
• Process small amounts of structured data	• Bring together large volumes of structured / semi-structured data • Query current and historical data		
Use case: analytics, reporting	**Use case**: analytics, reporting		

Storage Options

Production Database	Data Warehouse	Data Lake	
• Process small amounts of structured data	• Bring together large volumes of structured / semi-structured data • Query current and historical data	• Process large volumes of structured, semi-structured and unstructured data • Save on storage cost	
Use case: reporting, analytics	**Use case**: reporting, analytics	**Use case**: machine learning	

Storage Options

Production Database	Data Warehouse	Data Lake	Data Lakehouse
• Process small amounts of structured data	• Bring together large volumes of structured / semi-structured data • Query current and historical data	• Process large volumes of structured, semi-structured and unstructured data • Save on storage cost	• Data management & discoverability features • Low latency queries
Use case: reporting, analytics	**Use case**: reporting, analytics	**Use case**: machine learning	**Use case**: machine learning, analytics, reporting

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Lakehouse Architecture on AWS

AWS Lake Formation