Copyright Notice

Data Storage and Queries

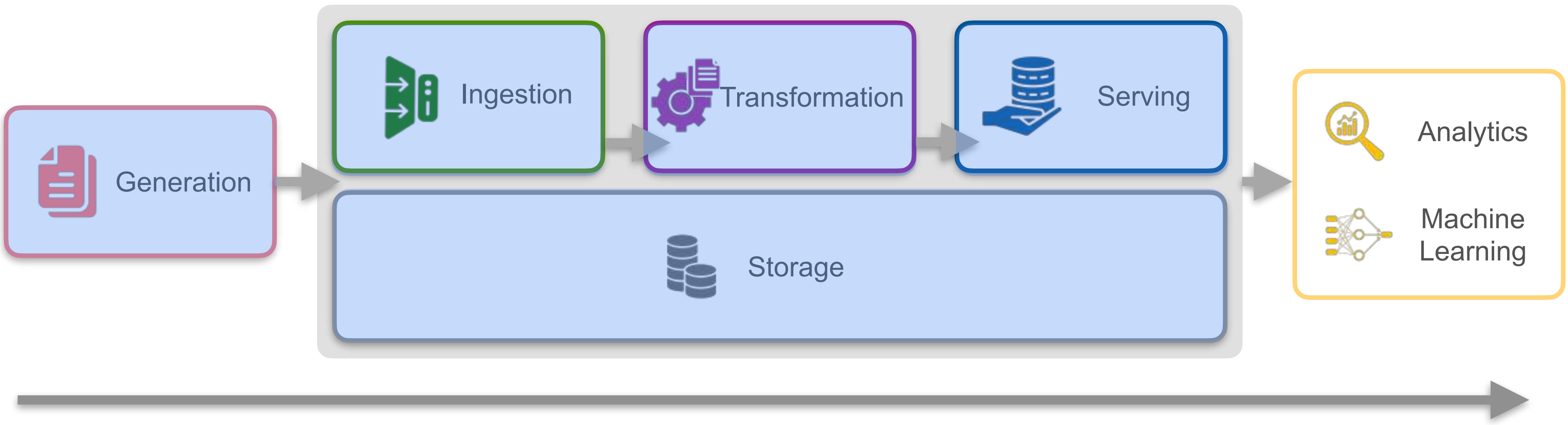
Storage Systems

DeepLearning.AI

Storage Systems

Course 3 Overview

Storage



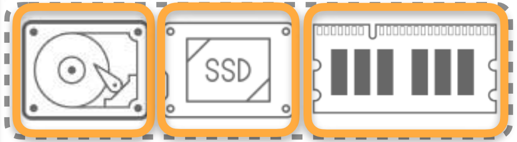
Storage solution considerations:

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

Storage Hierarchy

Raw Ingredients

Physical components



Processes

Networking
Serialization
CPU
Compression
Caching

Storage Hierarchy

Management system:

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

OLTP Systems

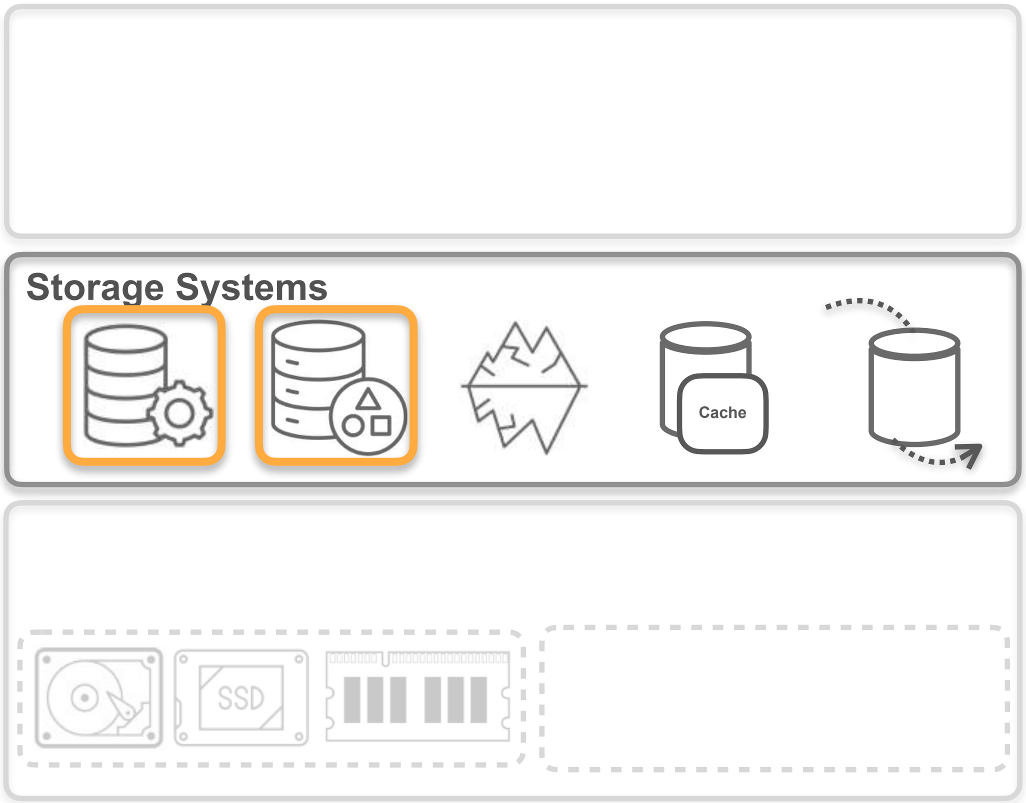
Online Transactional Processing Systems

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

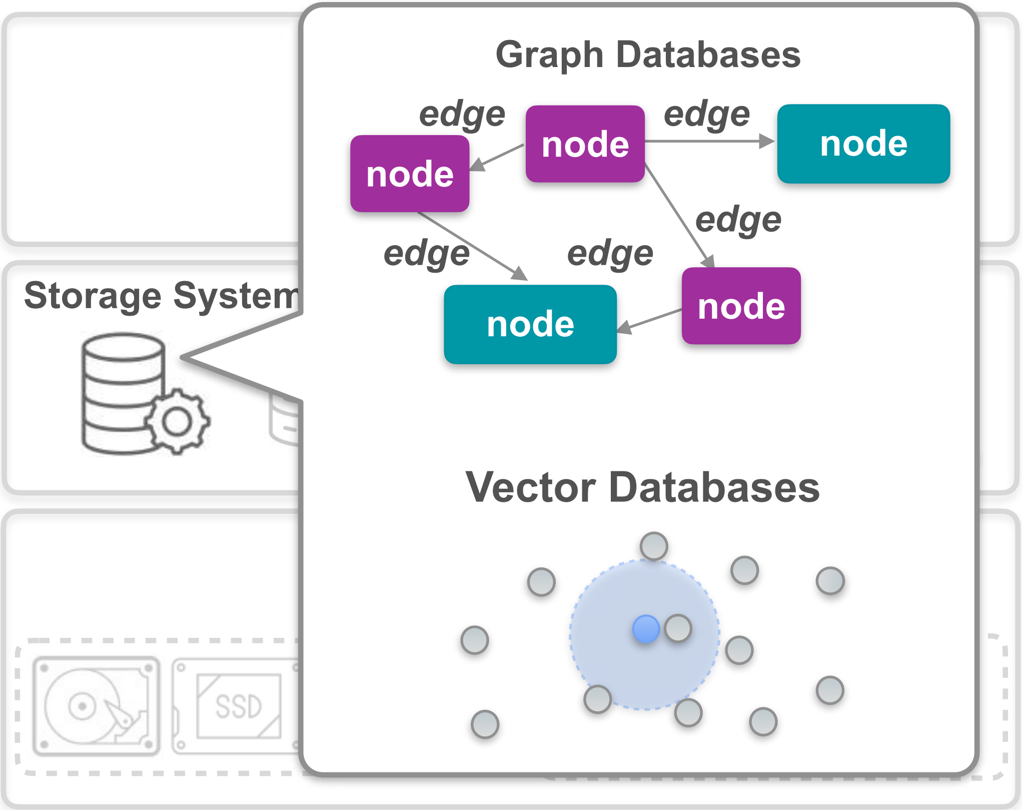
Online Transactional Processing Systems

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)

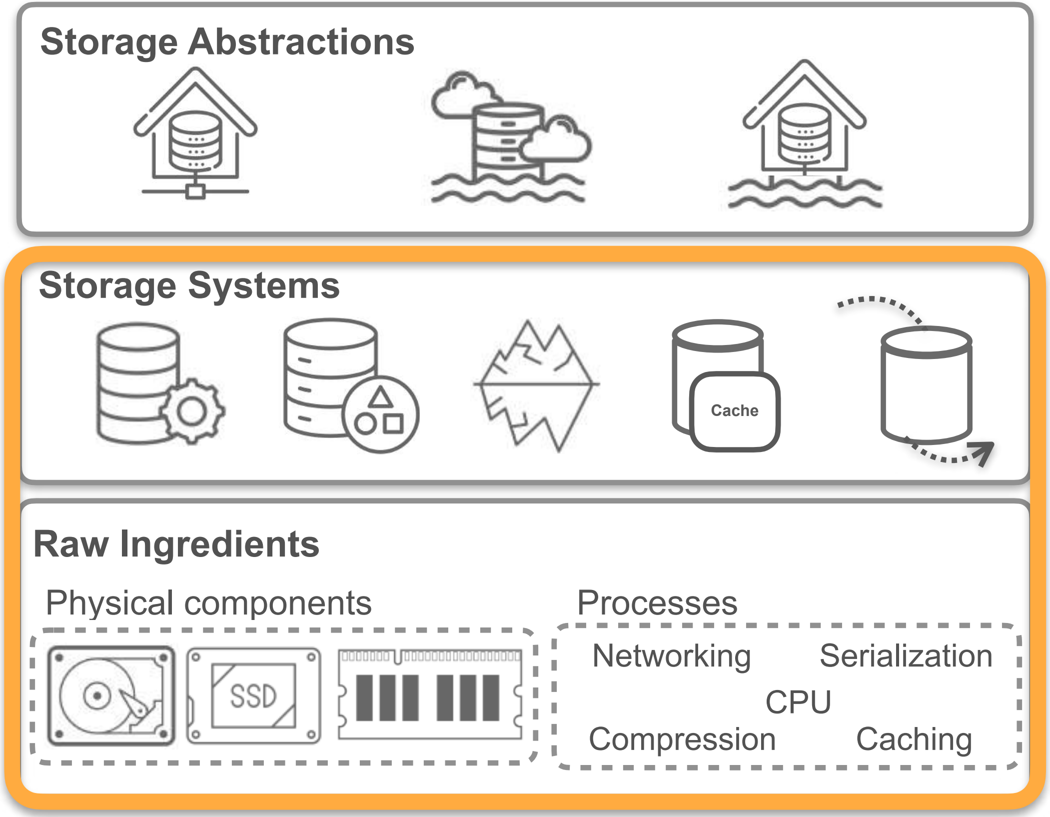


Storage Hierarchy

Course 3 Week 1

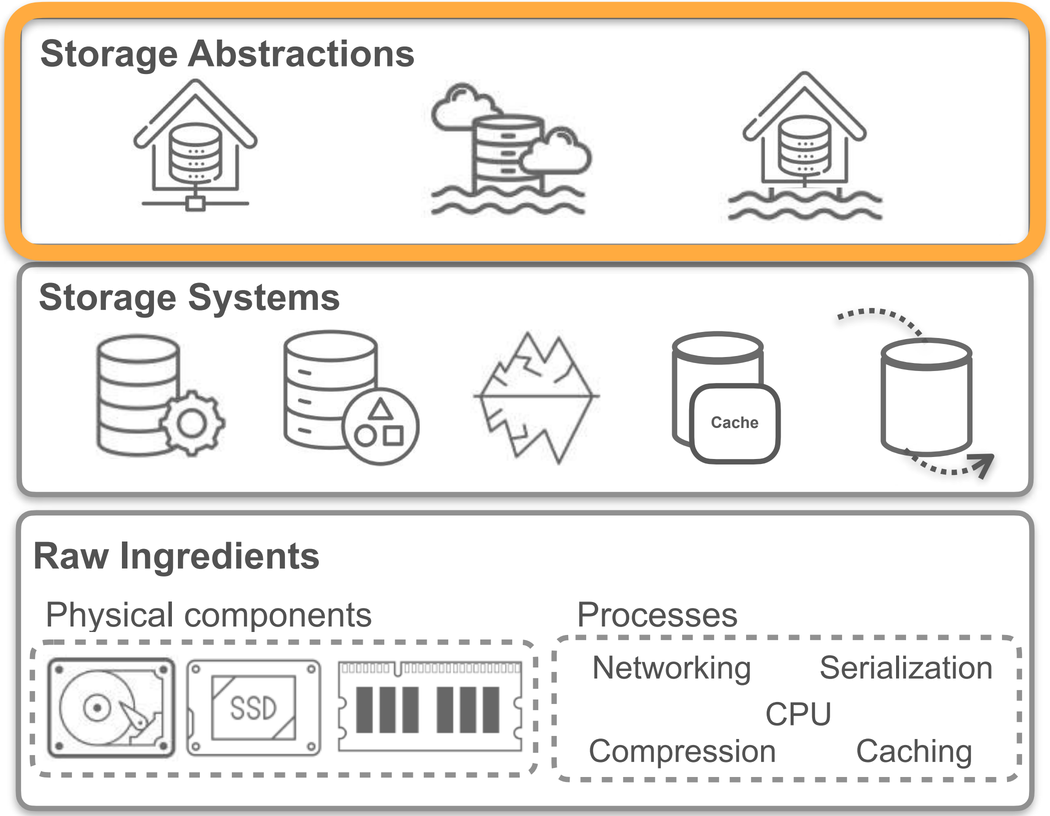
Trade-offs between storage cost and performance

- Cloud storage paradigms (block, object and file storage)
- Data storage in databases
  - Row vs column-oriented databases
  - Graph and vector databases
- Characteristics of physical components
- Serialization and compression



Course 3 Week 2

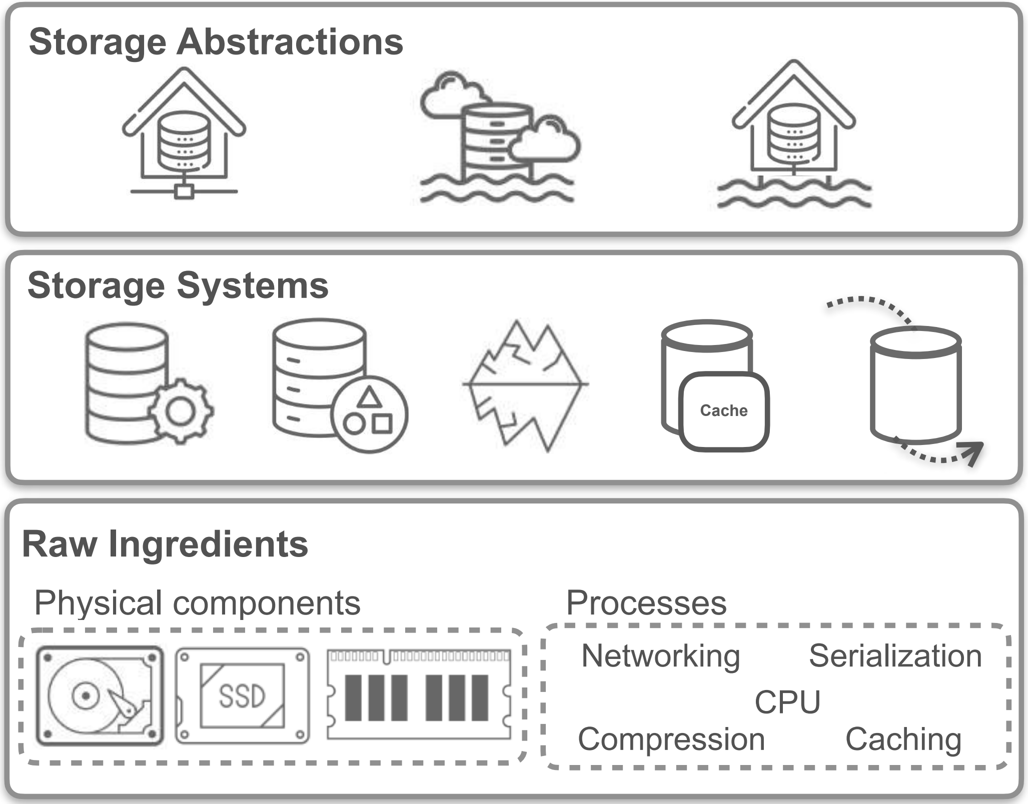
How to choose the appropriate abstractions for storing your data



Course 3 Week 3

Queries

- How queries work
- How different storage solutions affect query performance
- Techniques for improving query performance



Storage Systems

Raw Ingredients:
Physical Components of Data Storage

Raw Storage Ingredients

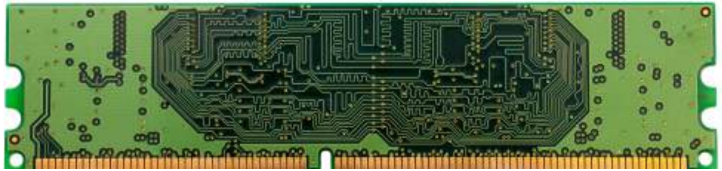
Persistent Storage Medium

Magnetic disk

Solid-state storage

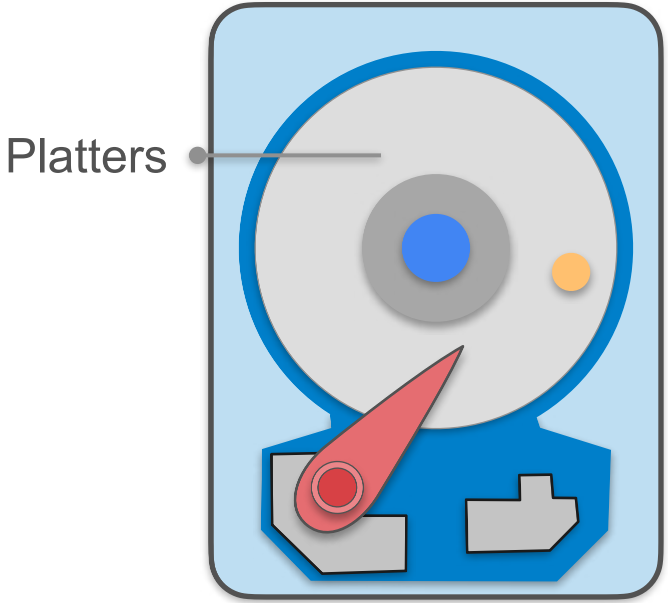
Volatile Memory

RAM



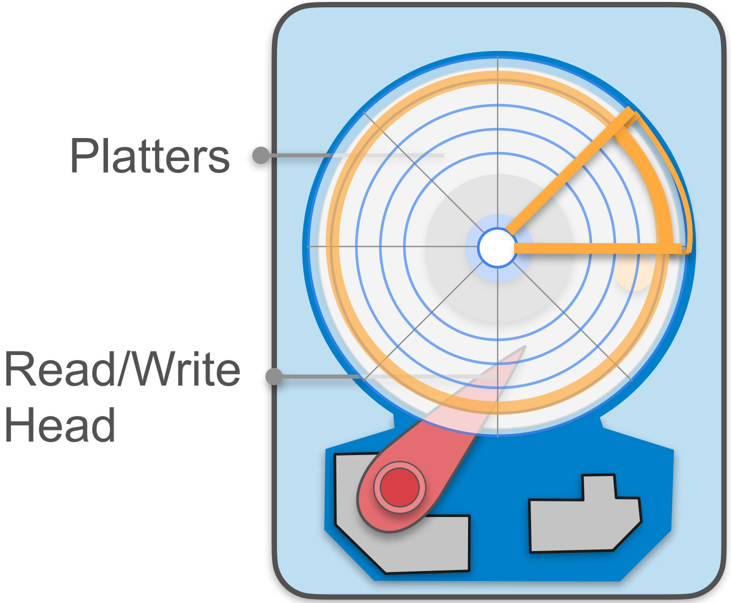
CPU cache

Magnetic Disks



Hard Disk Drives (HDDs)

Magnetic Disks



Hard Disk Drives (HDDs)

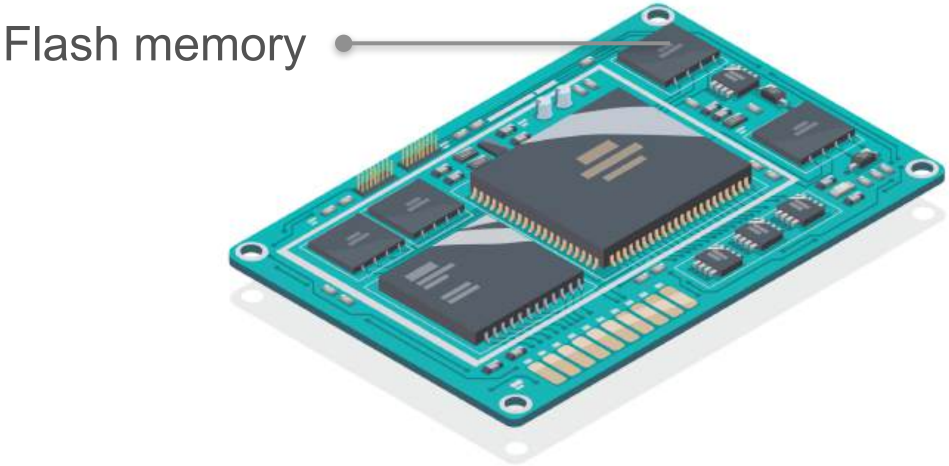
Track + Sector

= Address

Write:
Encode binary data by changing the magnetic field

Read:
Converts magnetic field into binary data

Solid State Drives



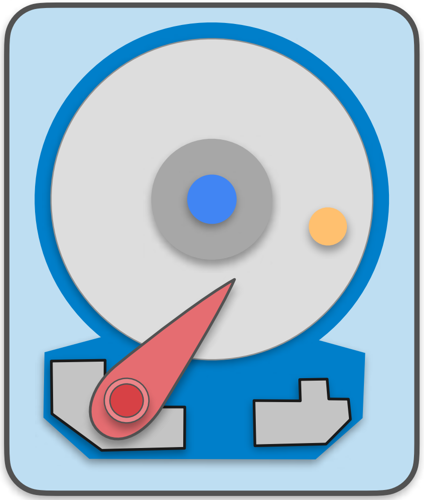
Solid-State Drives (SSDs)

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

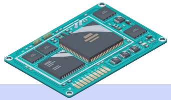
Performance Comparison

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

Solid-State Drives (SSDs)

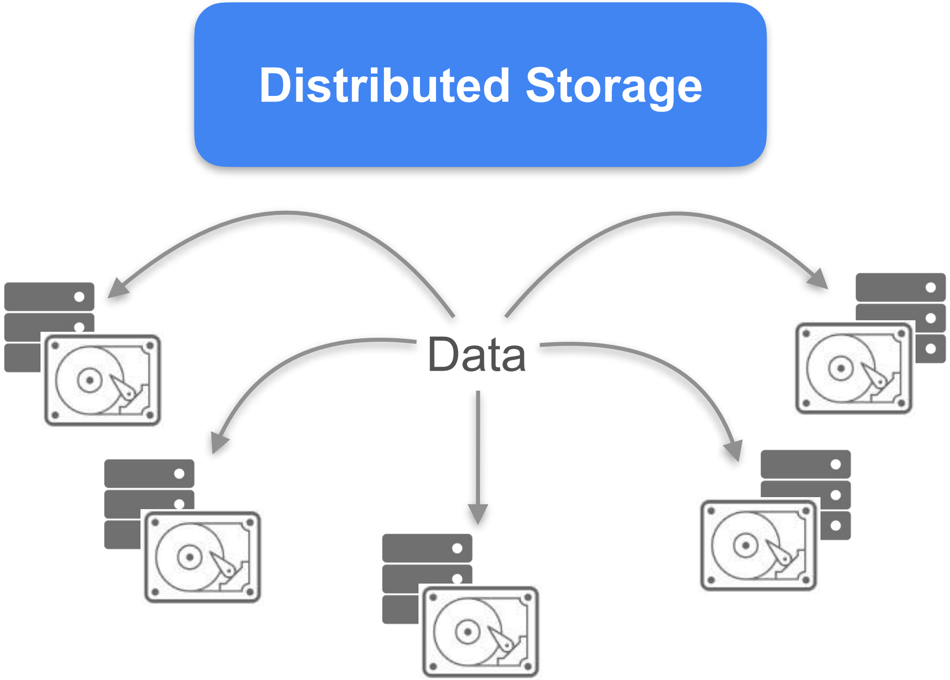
Electrical charges

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

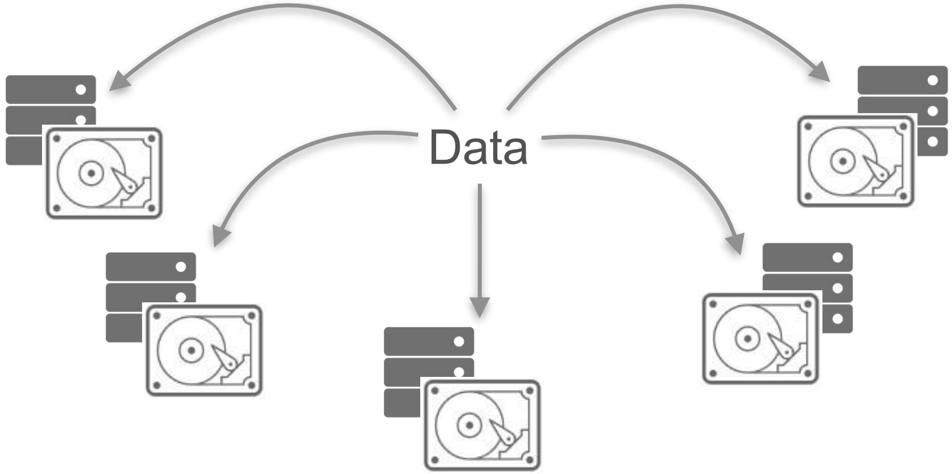
Improving Performance



Data transfer speed limited by network performance

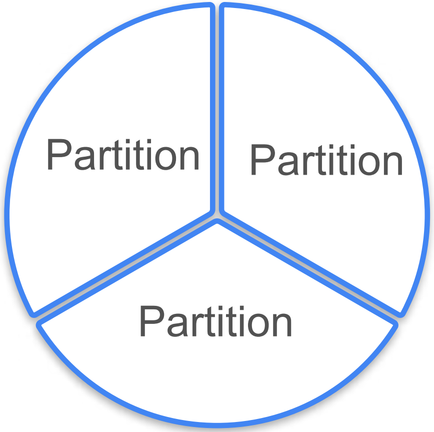
Improving Performance

Distributed Storage



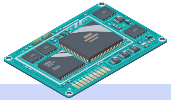
Data transfer speed limited by network performance

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

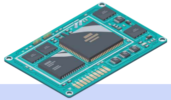
***Note: these metrics can vary*


	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*

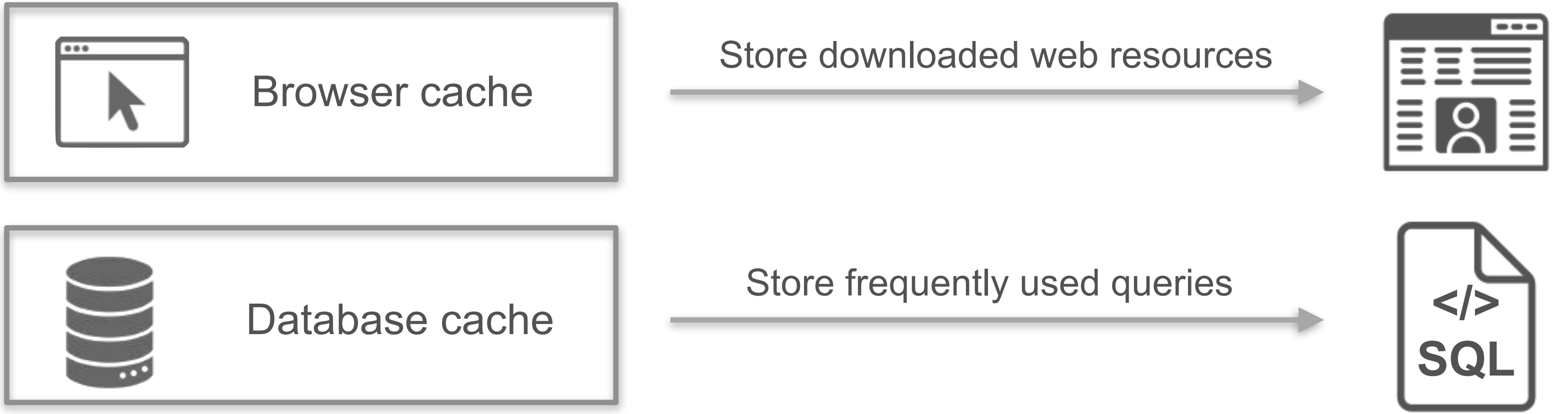



	Magnetic Disk	**SSD**	**RAM** (Random Access Memory)	**CPU Cache**
Latency	4 milliseconds	0.1 milliseconds	0.1 microseconds	1 nanosecond
IOPS (Input/output operations per second)	Hundreds	Tens of thousands	Millions	/
Data Transfer Speed (number of bytes read/ written from disk to memory in a second)	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

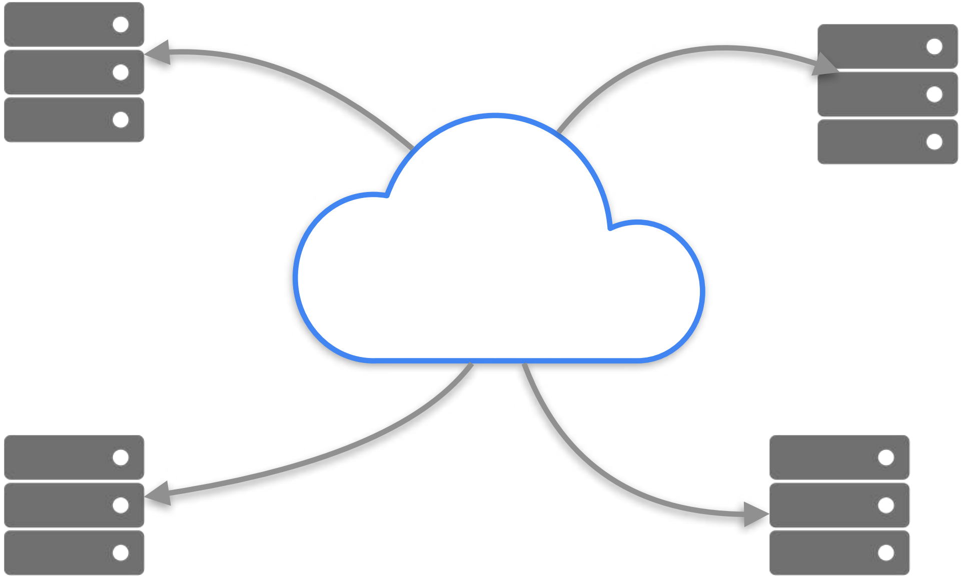
Examples



Storage Systems

Raw Ingredients:
Processes Required for Data Storage

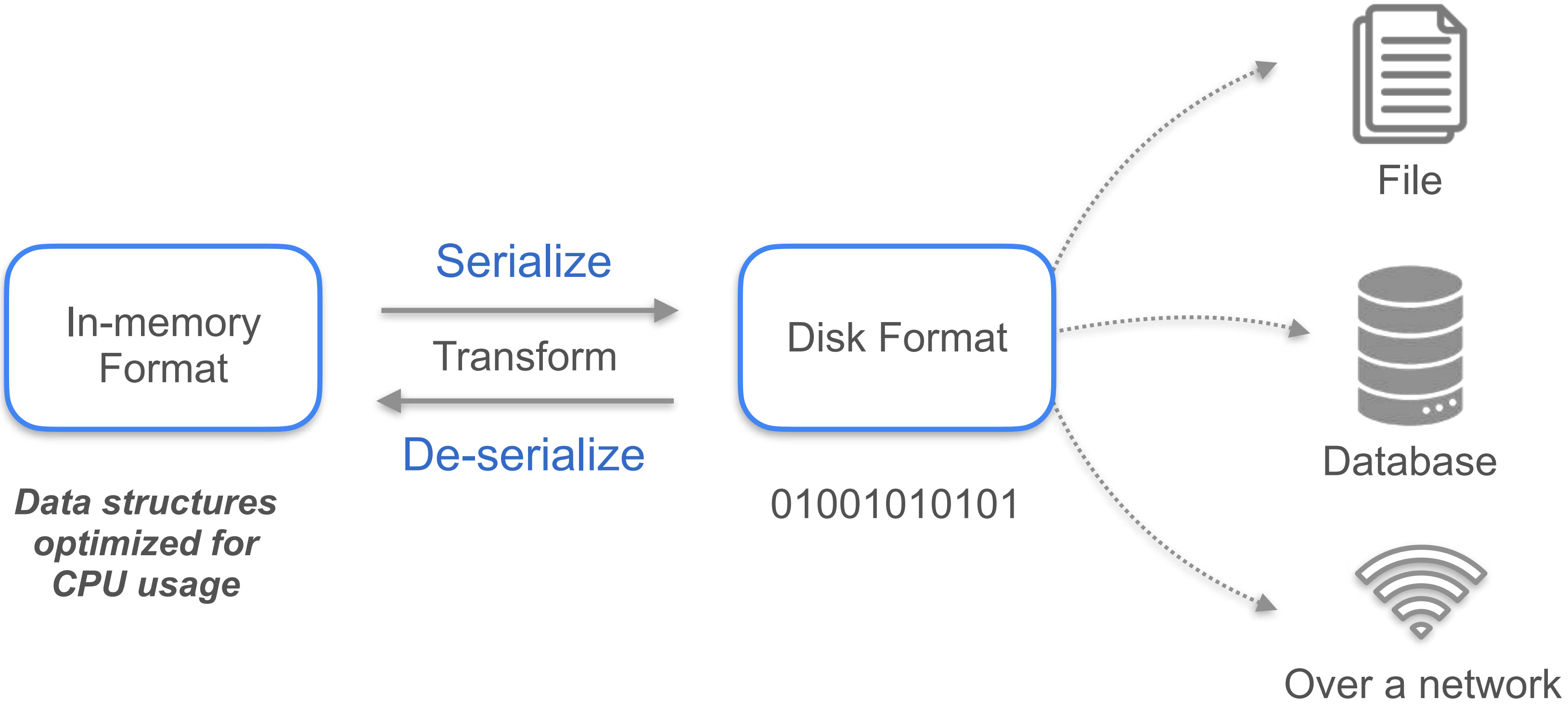
Networking and CPU — "Raw Ingredients" of Storage Systems



Enhance:

- read and write performance
- data durability
- data availability

Serialization

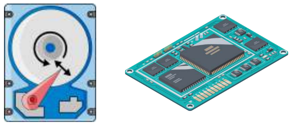


Serialization

Transactional operations

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

Physical Storage



Row-Based Serialization

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

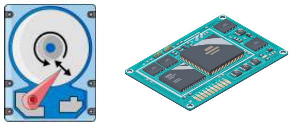
Column-Based Serialization

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

Serialization Formats

Human-Readable Textual Formats

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

Serialization Formats

Human-Readable Textual Formats

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

Serialization Formats

Binary Formats

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

Serialization Formats

Binary Formats

- 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

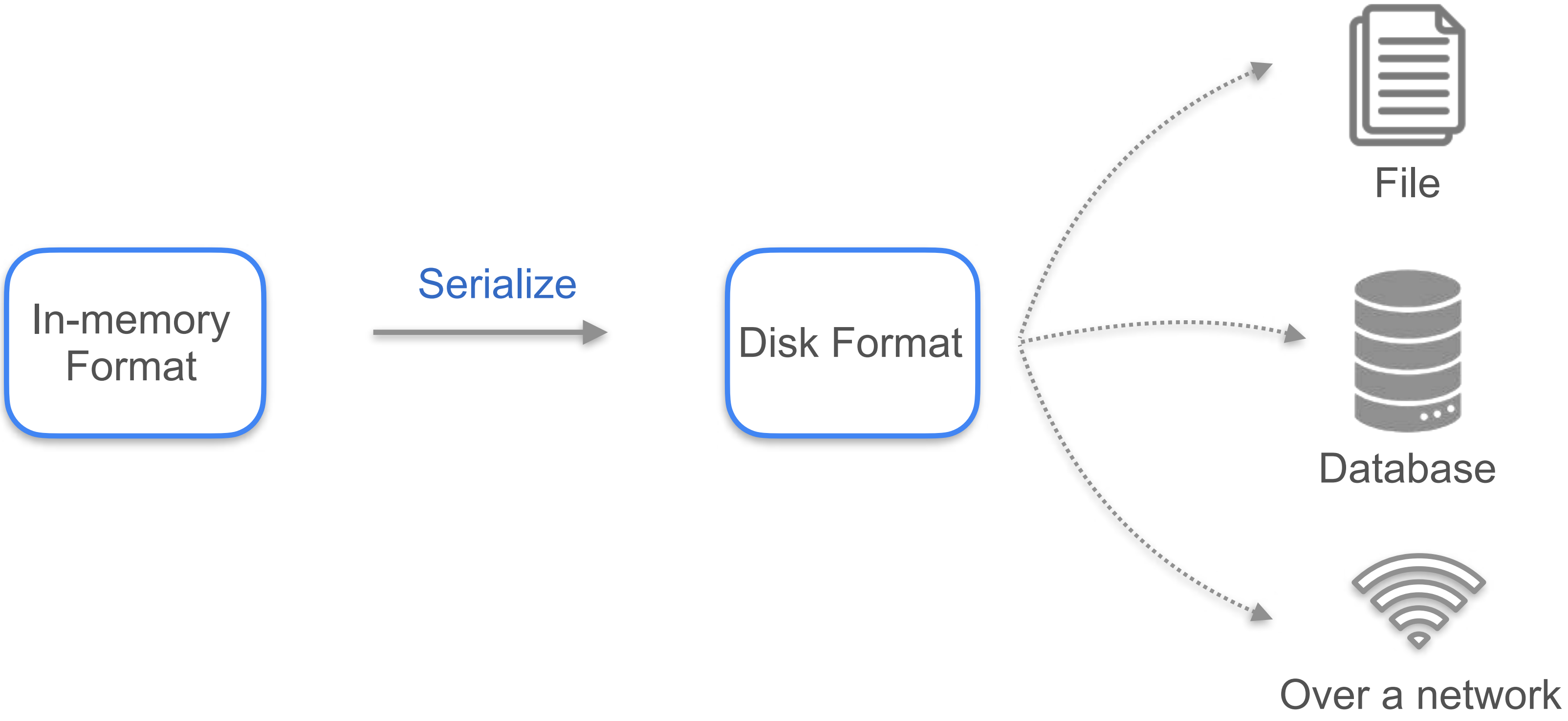
- Used for plain-text object serialization
- Viewed as new standard for data exchange over APIs

Binary Formats

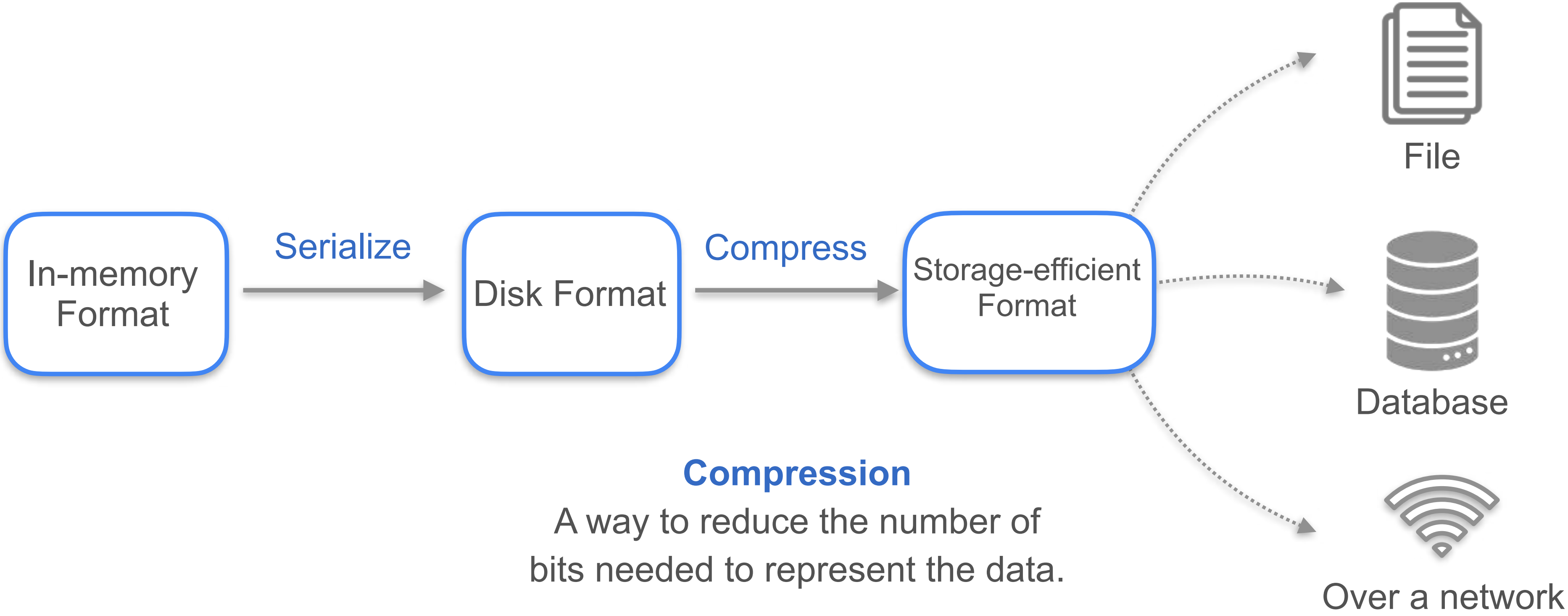
- Column-based format
- For efficient storage and big data processing

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

Serialization



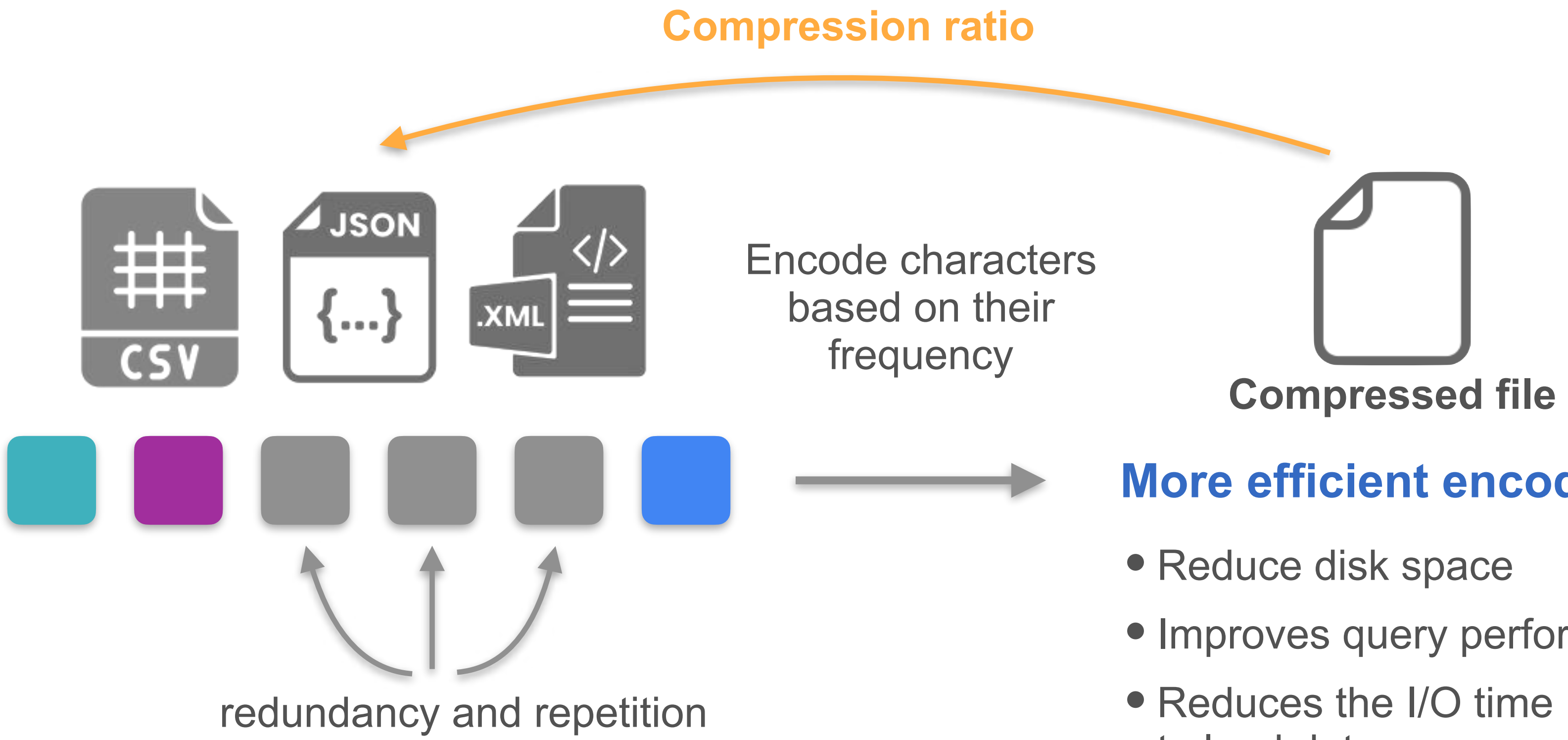
Serialization



Compression

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

Compression



More efficient encoding

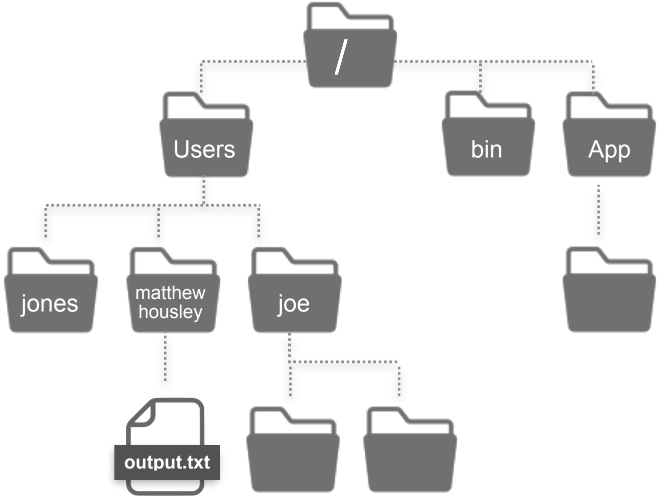
- Reduce disk space
- Improves query performance
- Reduces the I/O time needed to load data

DeepLearning.AI

Storage Systems

Cloud Storage Options: Block, Object and File storage

File Storage



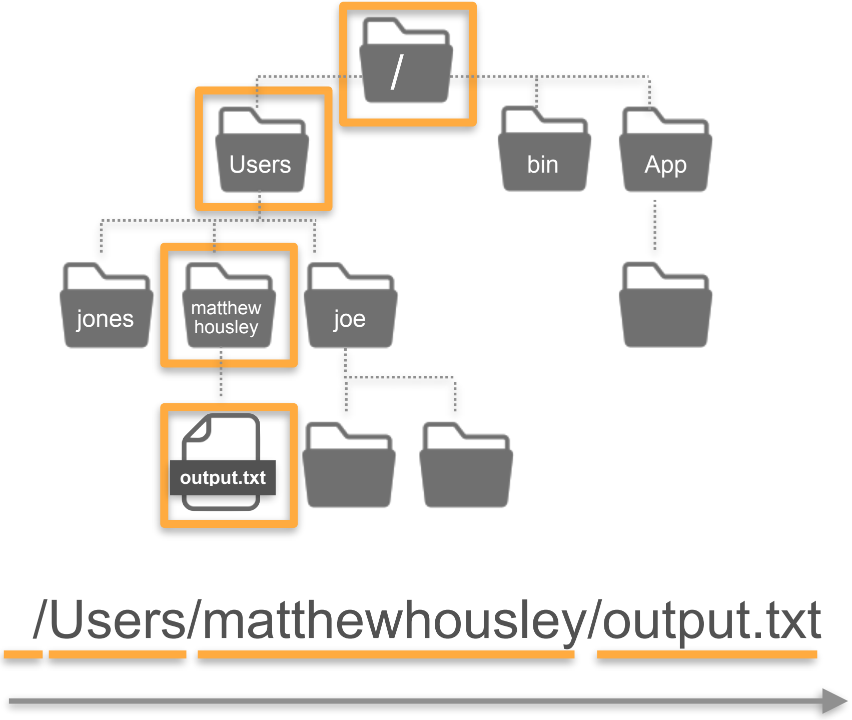
File Storage

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



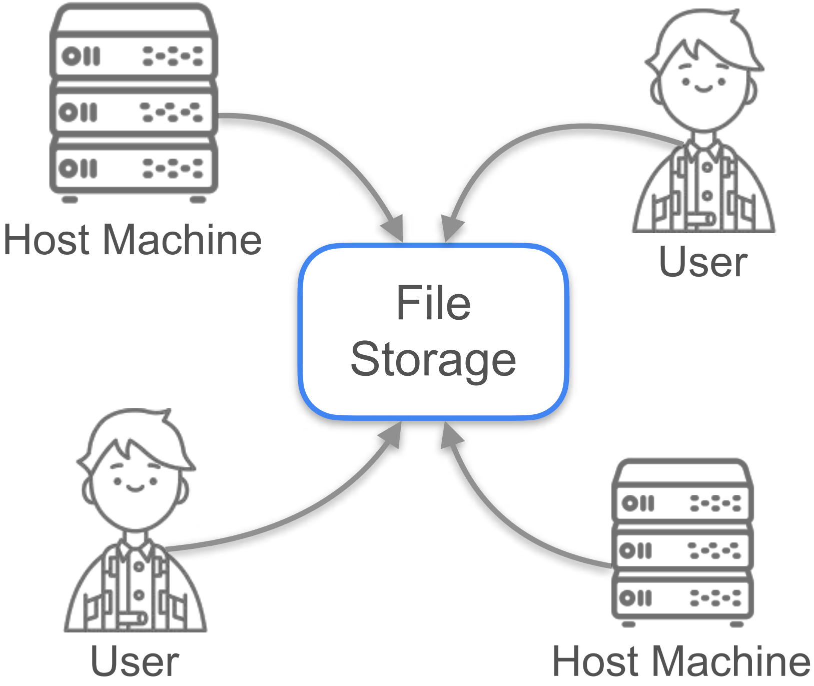
File Storage

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

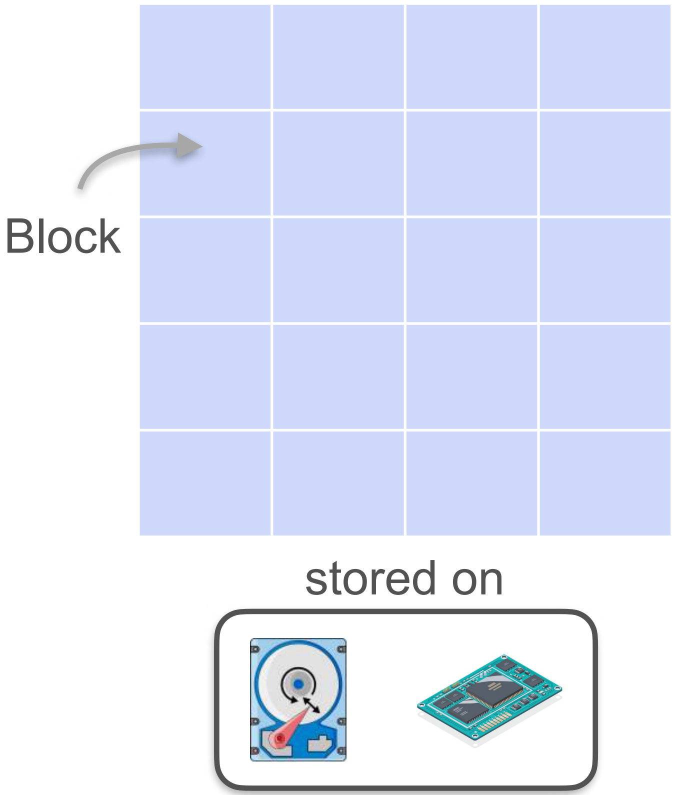


Cloud File Storage Service

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

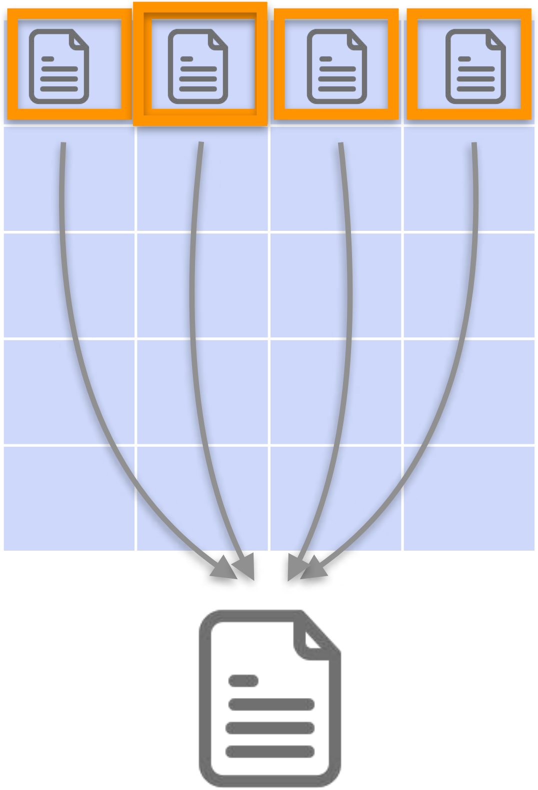


Block Storage

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

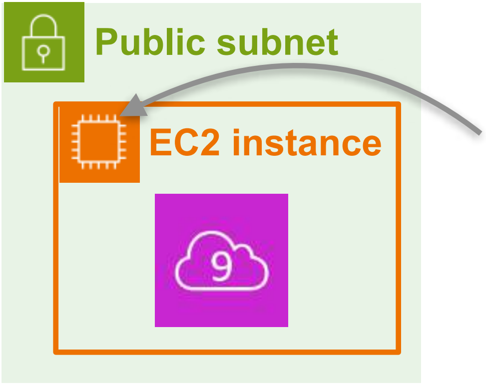


Lookup Table

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



Attach a root storage device backed by a block storage volume

Default storage for EC2

Amazon Elastic Block Store (EBS)

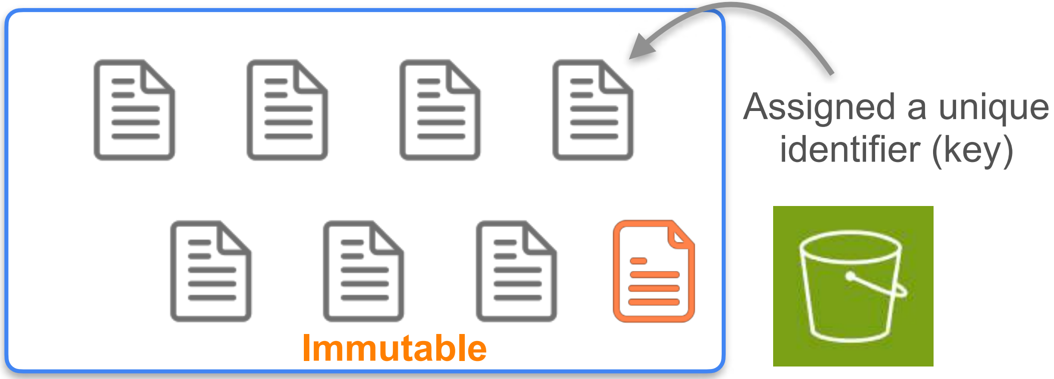
1. SSD for latency-sensitive workloads
2. Magnetic disks to store infrequently-accessed data

Object Storage

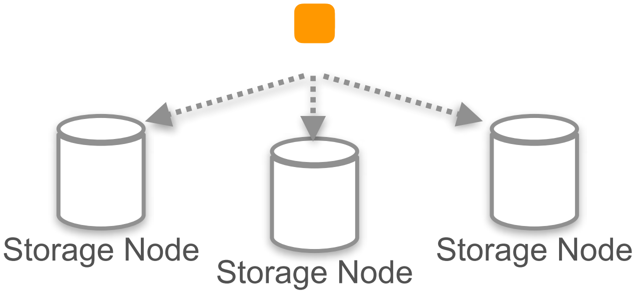
Object Storage Stores immutable files as data objects in a flat structure

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



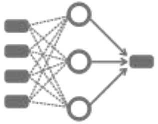
s3://o'reilly-data-engineering-book/data-example.json

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	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

DeepLearning.AI

Storage Systems

Storage Tiers –
Hot, Warm, & Cold Data

Storage Tiers

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

Storage Tiers

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

Storage Tiers

		Warm Storage	
Access Frequency		Less frequent	
Example		Regular reports and analyses	
Storage Medium		Magnetic disks or hybrid storage systems	
Storage Cost		Medium	
Retrieval Cost		Medium	

Storage Tiers

			Cold Storage
Access Frequency			Infrequent
Example			Archive
Storage Medium			Low-cost magnetic disks
Storage Cost			Low
Retrieval Cost			High

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

Amazon S3

Access Frequency

Hot Storage
- S3 Express One Zone
- S3 Standard

Warm Storage
- S3 Standard-IA
- S3 One Zone-IA

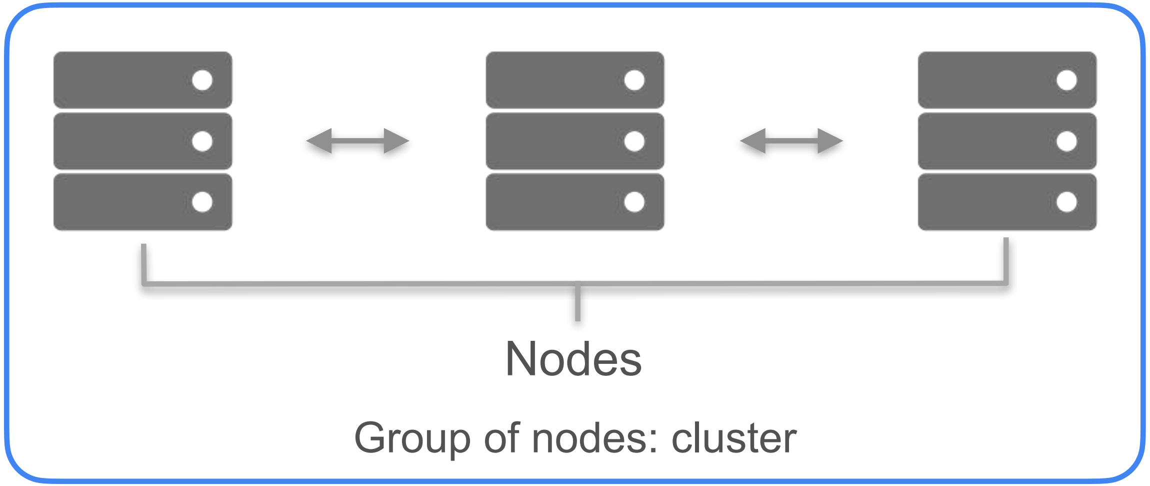
Cold Storage
- S3 Glacier Flexible Retrieval
- S3 Glacier Deep Archive
- S3 Glacier Instant Retrieval

DeepLearning.AI

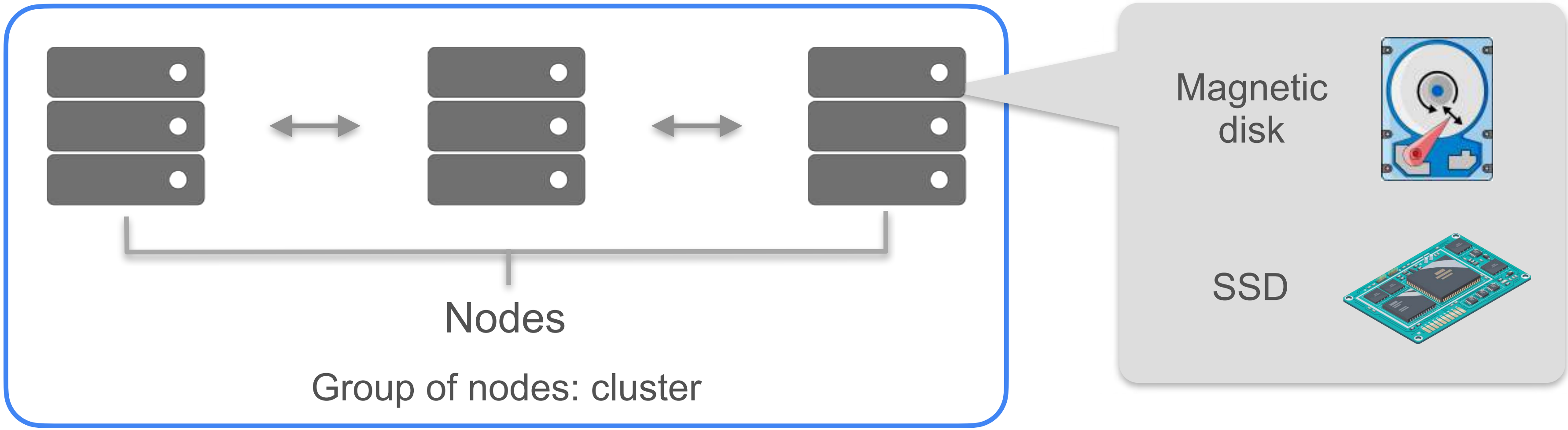
Storage Systems

Distributed Storage Systems

How Distributed Storage Systems Work

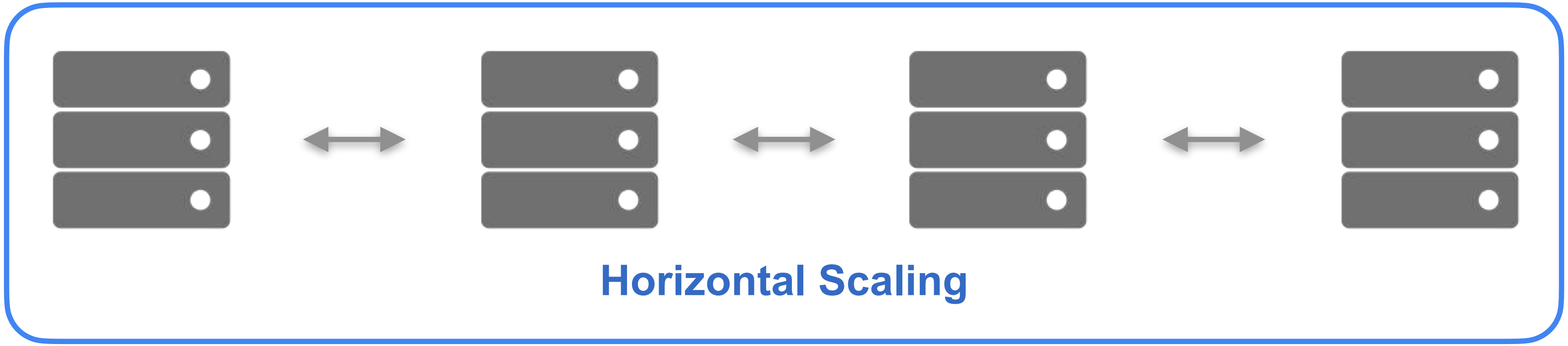


How Distributed Storage Systems Work

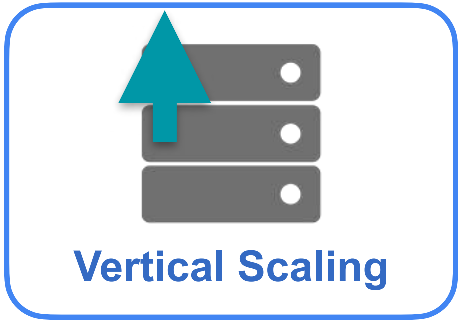


Total Capacity = + +

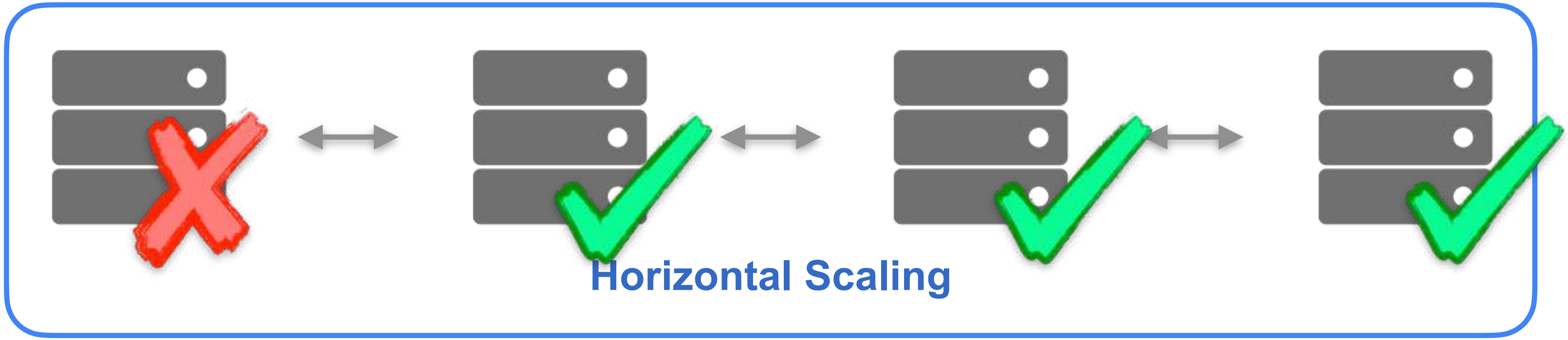
How Distributed Storage Systems Work



Single Machine Storage Architecture

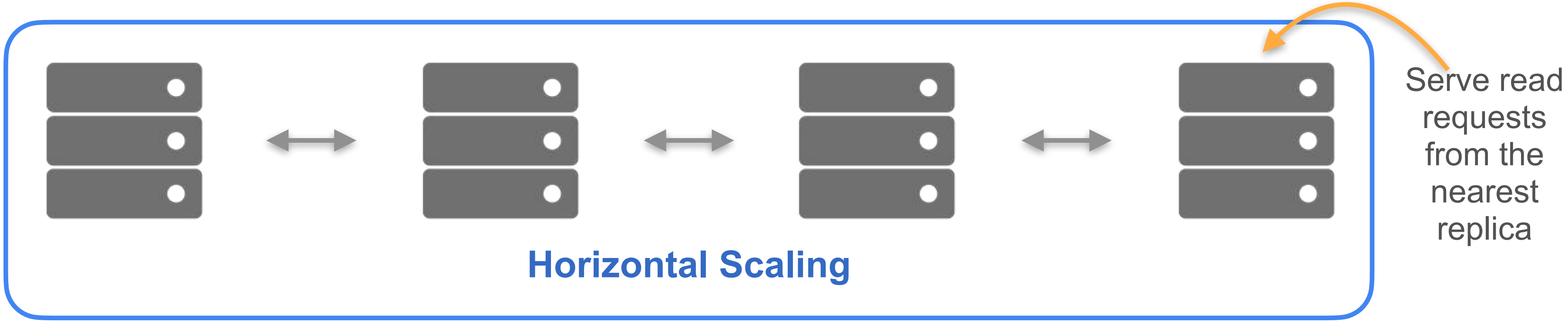


How Distributed Storage Systems Work

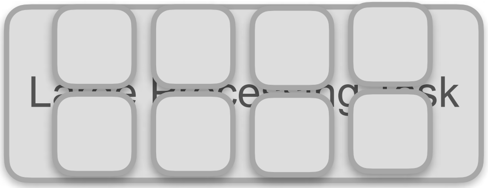


- Higher fault tolerance and data durability
- High availability

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

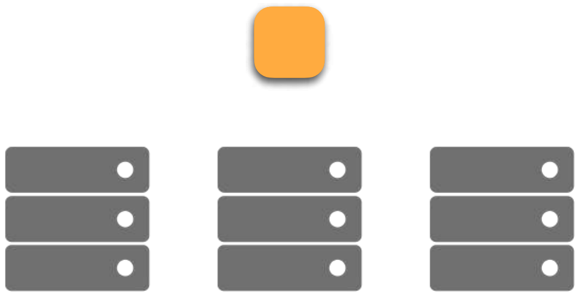
Methods for Distributing Data

Replication

Partitioning

Methods for Distributing Data

Replication



High availability and performance

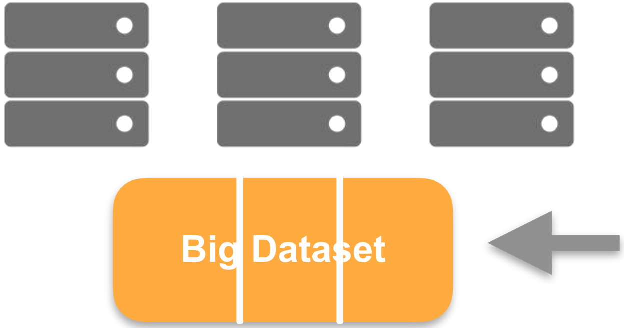
Partitioning

Methods for Distributing Data

Replication

High availability and performance

Partitioning



Sharding

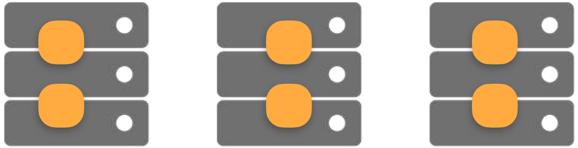
partition or shard

Methods for Distributing Data

Replication

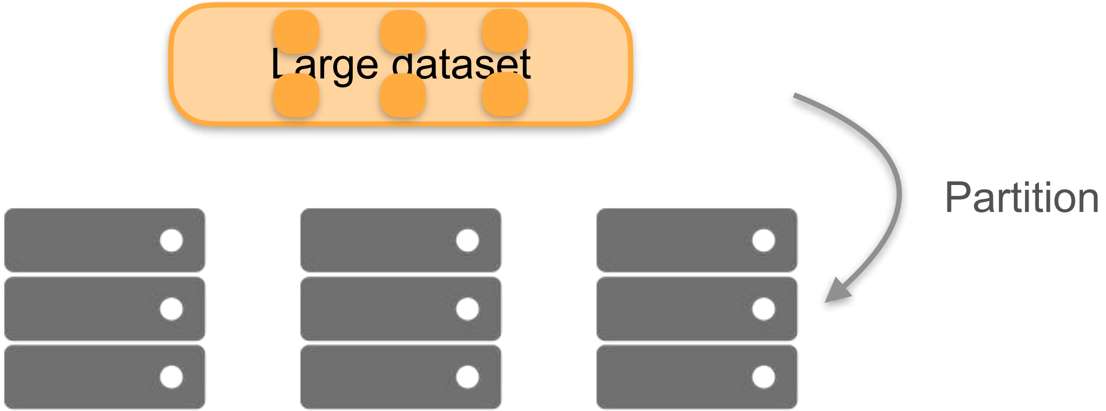
High availability and performance

Partitioning

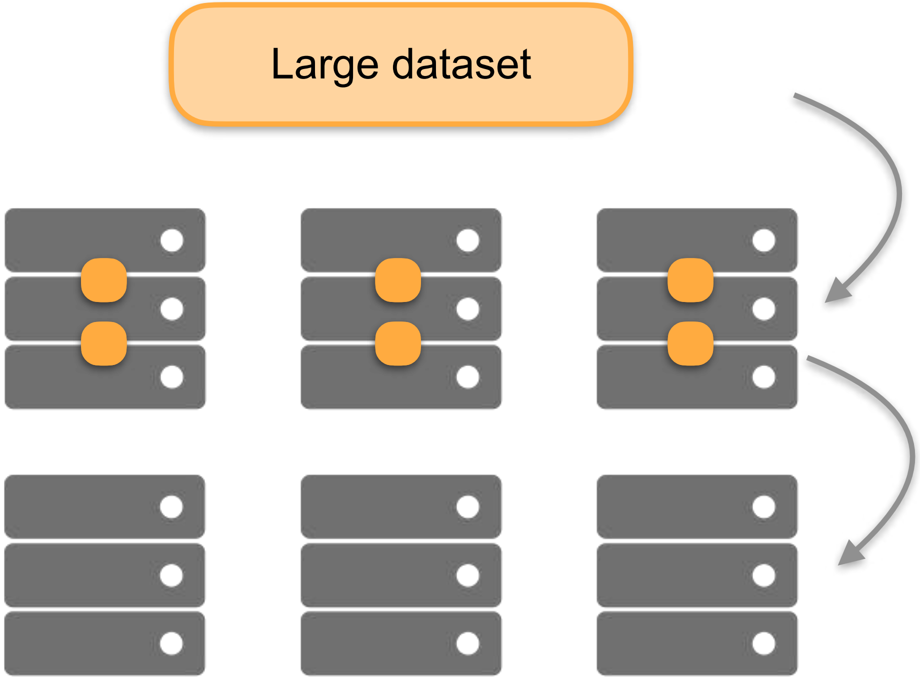


Sharding

Methods for Distributing Data



Methods for Distributing Data



Partition

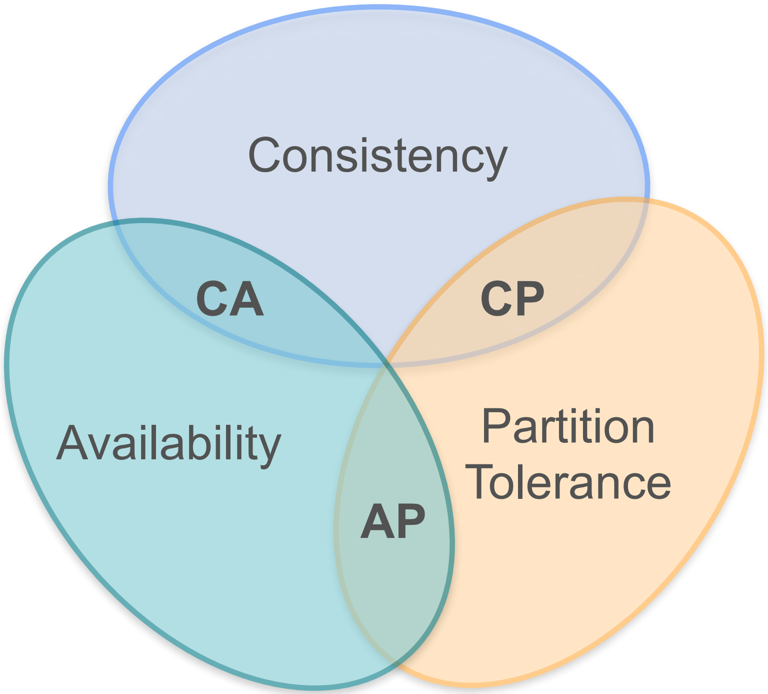
Replication

Takes time

- Wait for the update to complete
- Access “sort of” recent data

Distributed Storage Considerations – CAP Theorem

The CAP theorem

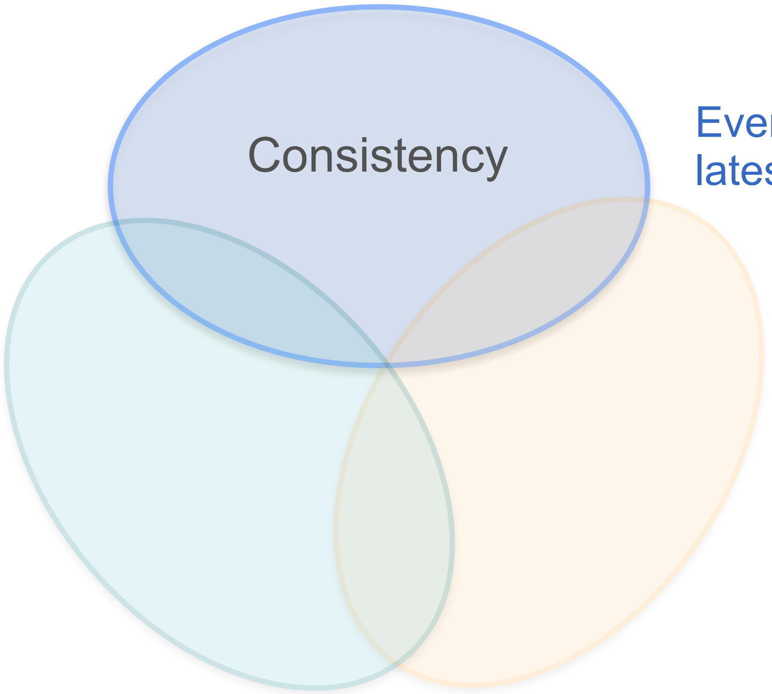


Distributed Storage Considerations – CAP Theorem

ACID

Consistency

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

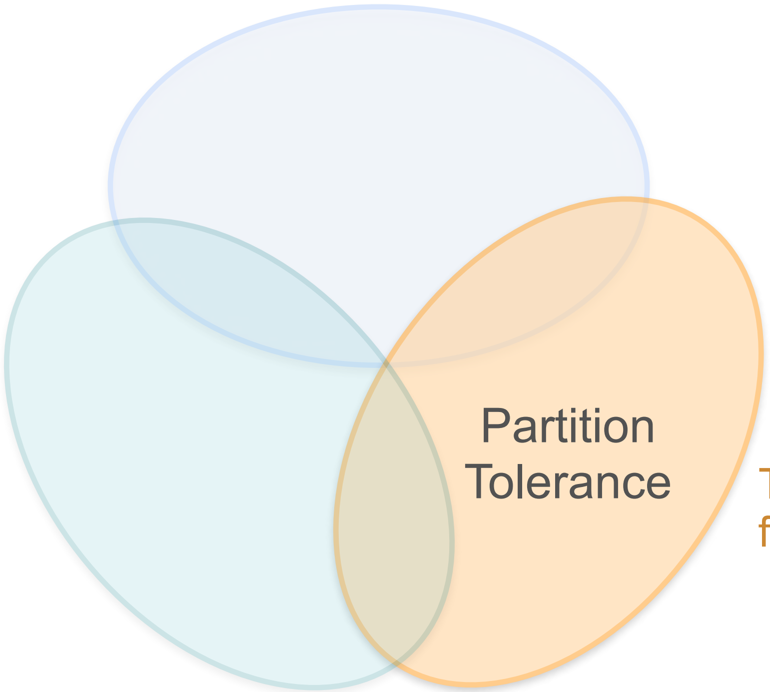


Every read reflects the latest write operation

Distributed Storage Considerations – CAP Theorem

Every request will receive a response

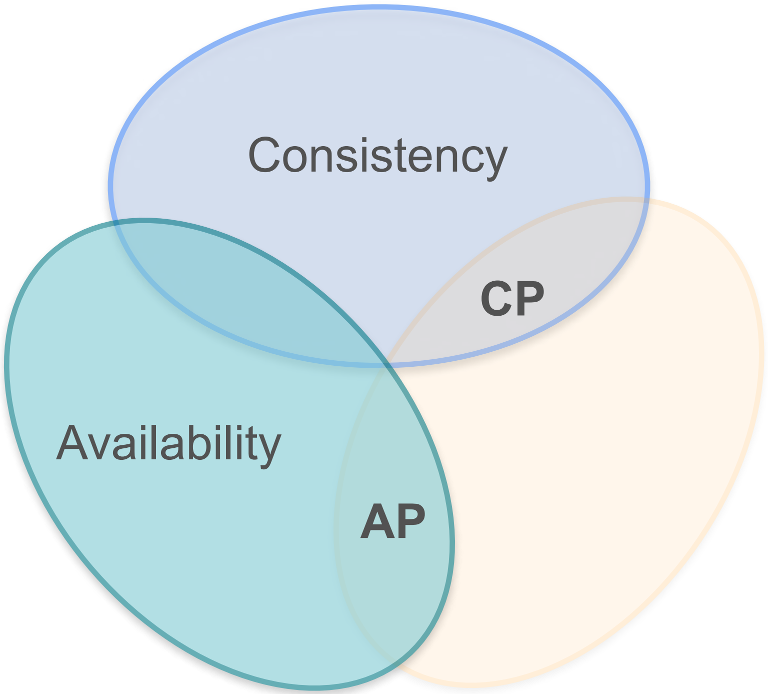
Distributed Storage Considerations – CAP Theorem



The system continues to function

Distributed Storage Considerations – CAP Theorem

The CAP theorem

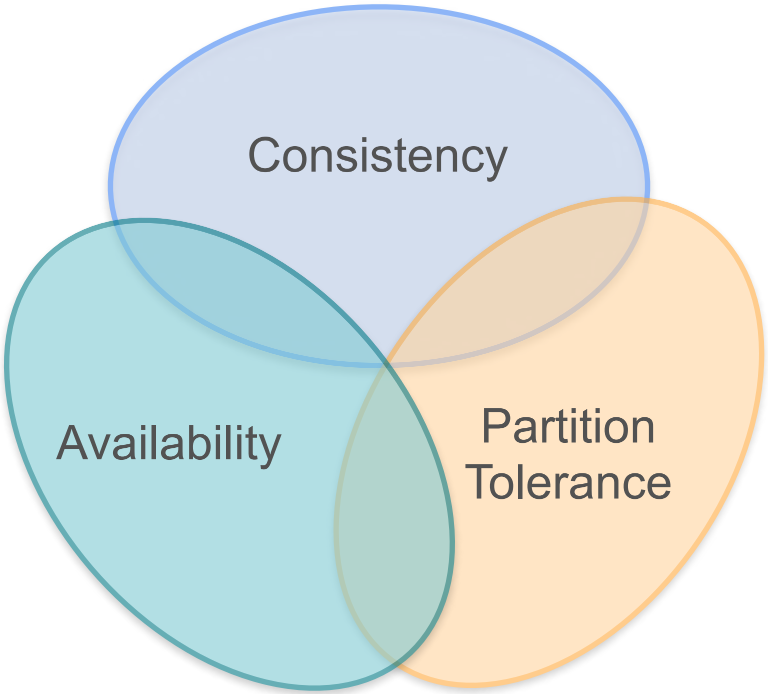


Distributed Storage Considerations – CAP Theorem

Scenario:
Accessing a node that's still being updated

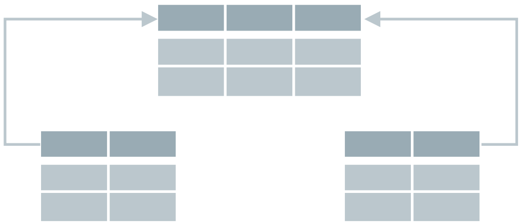
Option 1:
Cancel the request

Option 2:
Proceed with the read operation

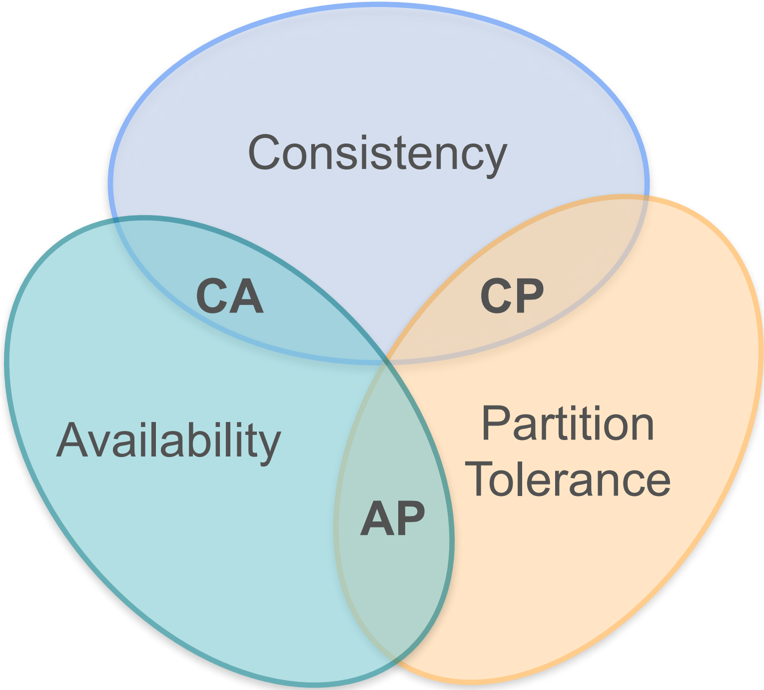


Distributed storage considerations – ACID vs BASE

RDBMS



ACID
compliant



NoSQL Databases

BASE
principles

Distributed storage considerations – ACID vs BASE

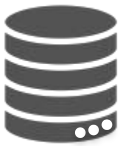
ACID compliant	BASE principles
Atomicity	**B**asically **A**vailable
Consistency	**S**oft-state
Isolation	**E**ventual Consistency
Durability	

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

Lab Walkthrough

Comparing Cloud Storage Options

Object Storage

File Storage

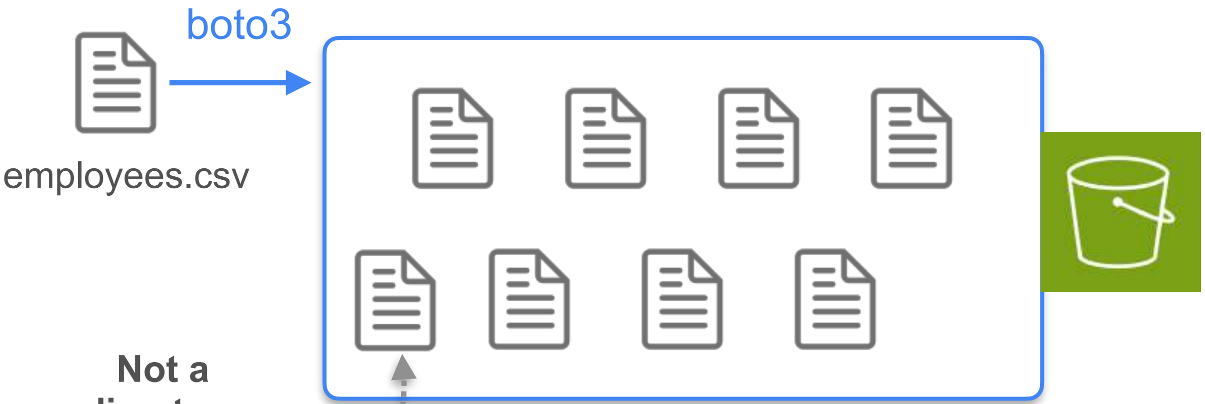
Block Storage

Memory

Object Storage

Flat Structure

Immutability



Not a directory

data/csv/employees.csv

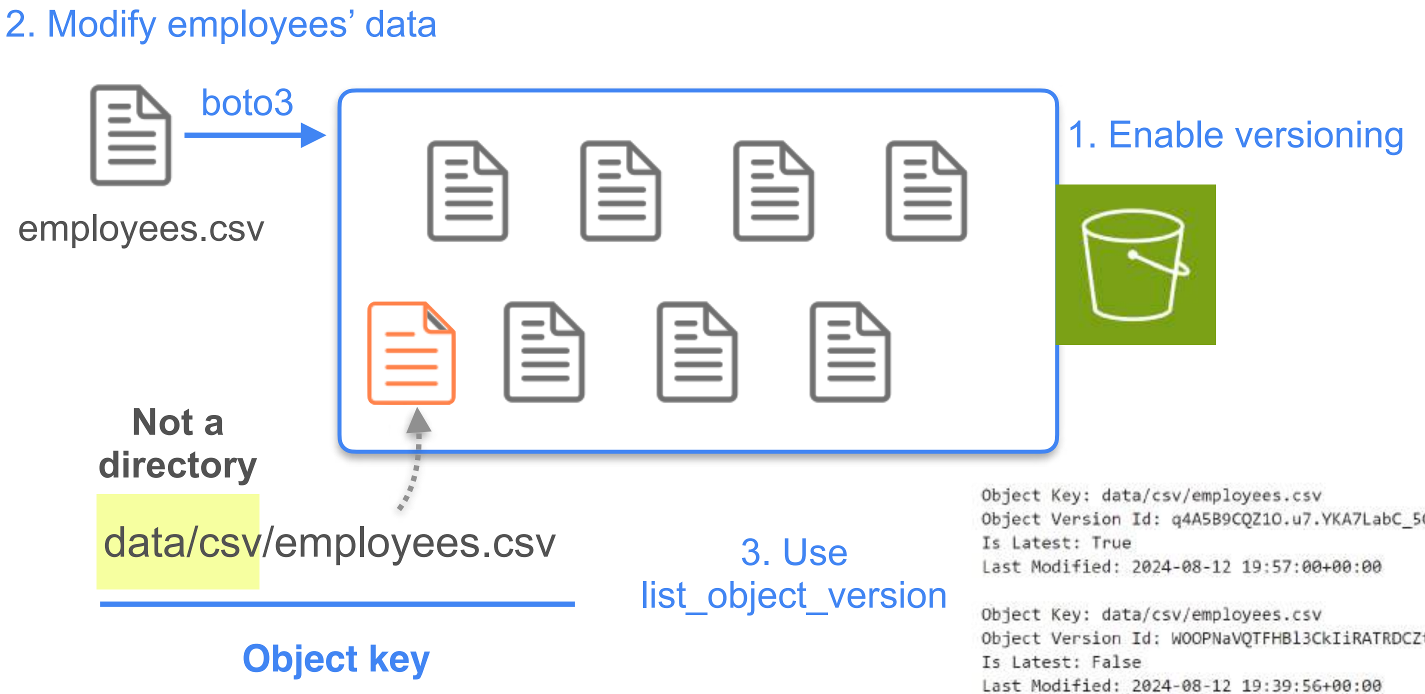
Object key

data/year=2020/sales.csv
data/year=2021/sales.csv
data/year=2022/sales.csv

Object Storage

Flat Structure

Immutability



3. Use list_object_version

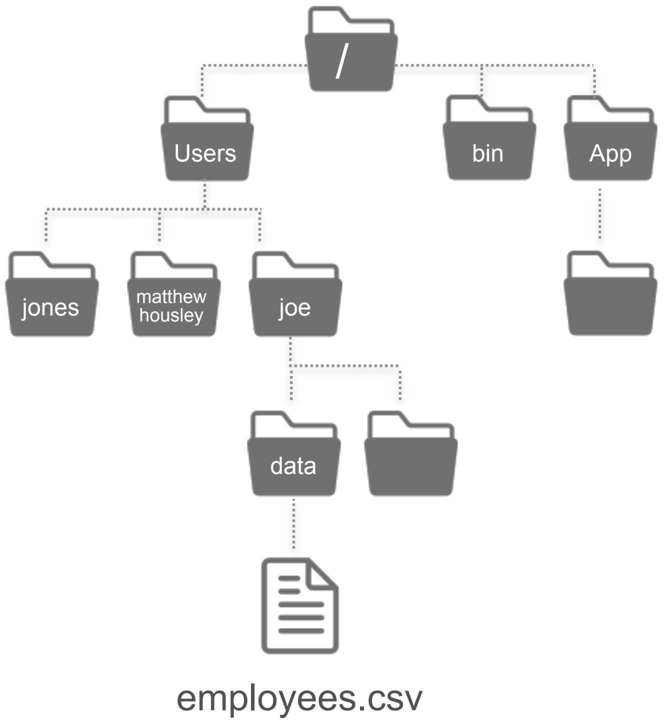
```
Object Key: data/csv/employees.csv
Object Version Id: q4A5B9CQZ1O.u7.YKA7LabC_5GcBA.uZ
Is Latest: True
Last Modified: 2024-08-12 19:57:00+00:00

Object Key: data/csv/employees.csv
Object Version Id: WOOPNaVQTFHBl3CkIiRATRDCZt3OWq9k
Is Latest: False
Last Modified: 2024-08-12 19:39:56+00:00
```

File Storage

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

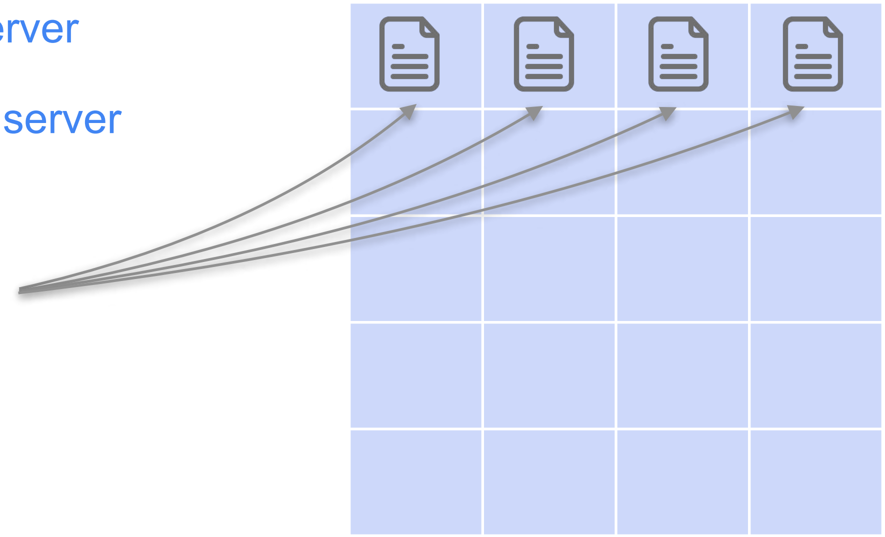
A directory

data/employees.csv



Block Storage

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



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

Use the cache-pandas package:

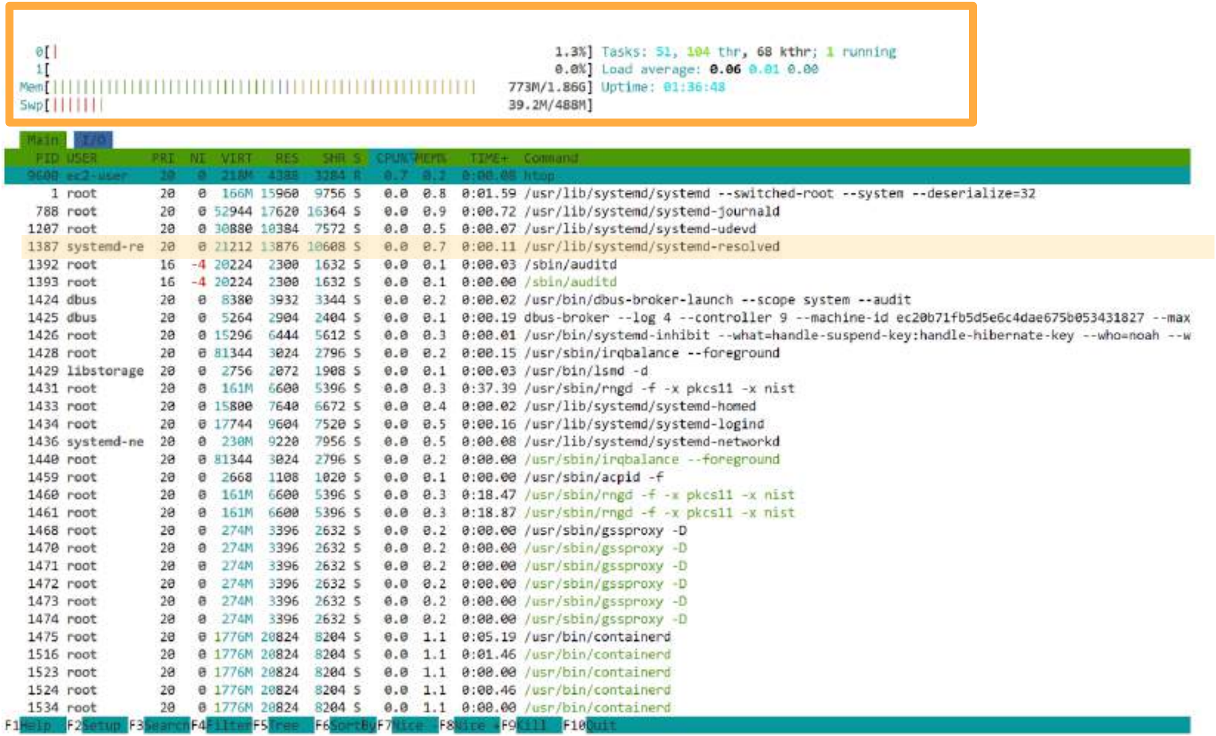
- provides the “timed_LRU_cache” decorator to easily cache in memory pandas DataFrames generated by functions.

```python
@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.

Memory

Monitor your memory storage capacity using htop command:



Memory

Object Storage

File Storage

Block Storage

Memory

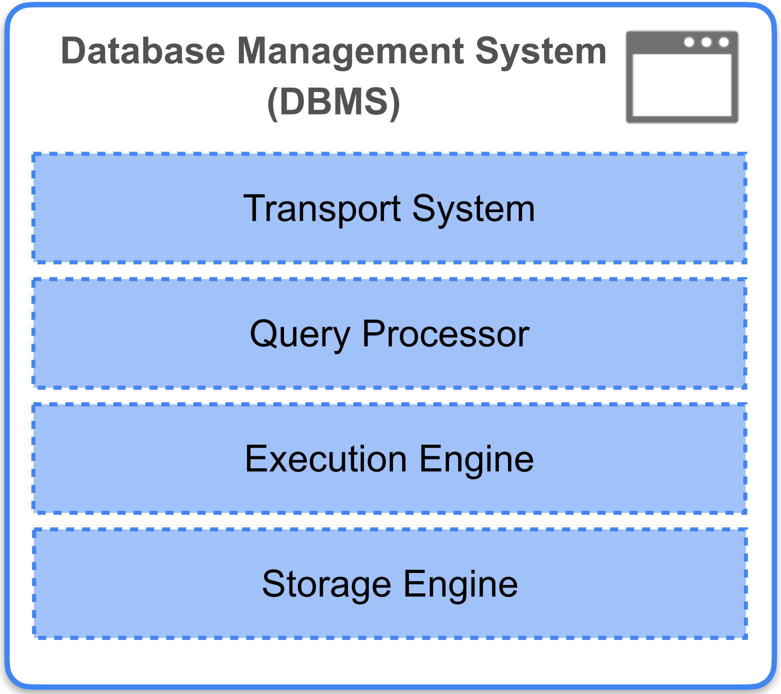
Explore the features of these storage options.

DeepLearning.AI

Storage Systems

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"
```

Index

A data structure that helps you efficiently locate data

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

.....

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

Index table

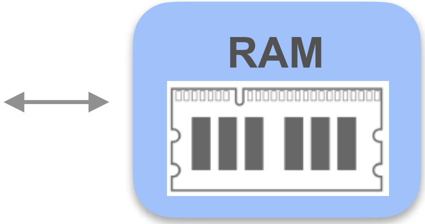
Country	Row Address
Brazil	###
Canada	###
Canada	###
Canada	###
Canada	###
Mexico	###
USA	###
USA	###
USA	###

Use binary search to locate the USA rows

In-Memory Storage Systems

Data

Application code



- 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

Storage Systems

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

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

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	…

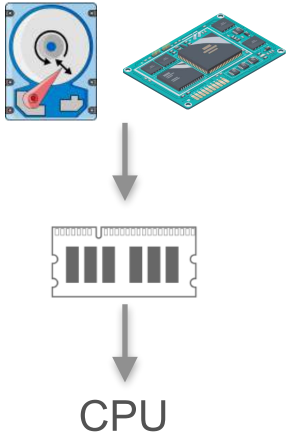
DeepLearning.AI

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
```



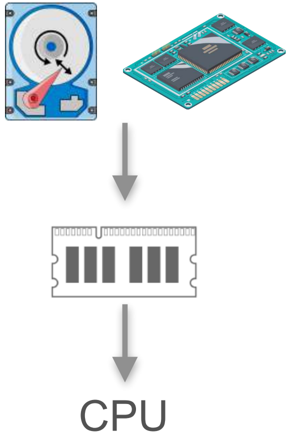
Physical Storage

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	...
...	...	...	...	...	...

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



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

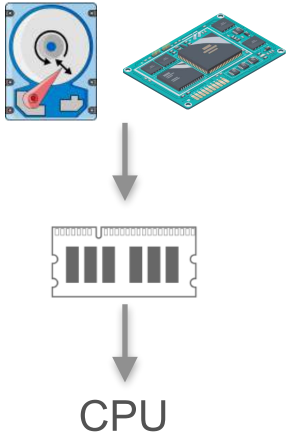
Data transfer speed: 200 MB/s

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

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? $\frac{3000\text{ GB}}{200\text{ MB/s}}$ = 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

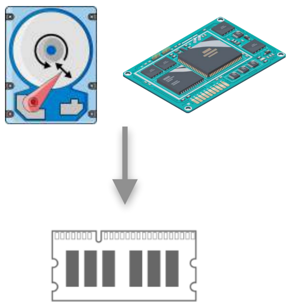
1	2	3	4	40	23	45	50	45865	90234	12558	45682	…

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? $\dfrac{\text{100 GB or 100,000 MB}}{\text{200 MB/s}}$ = 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

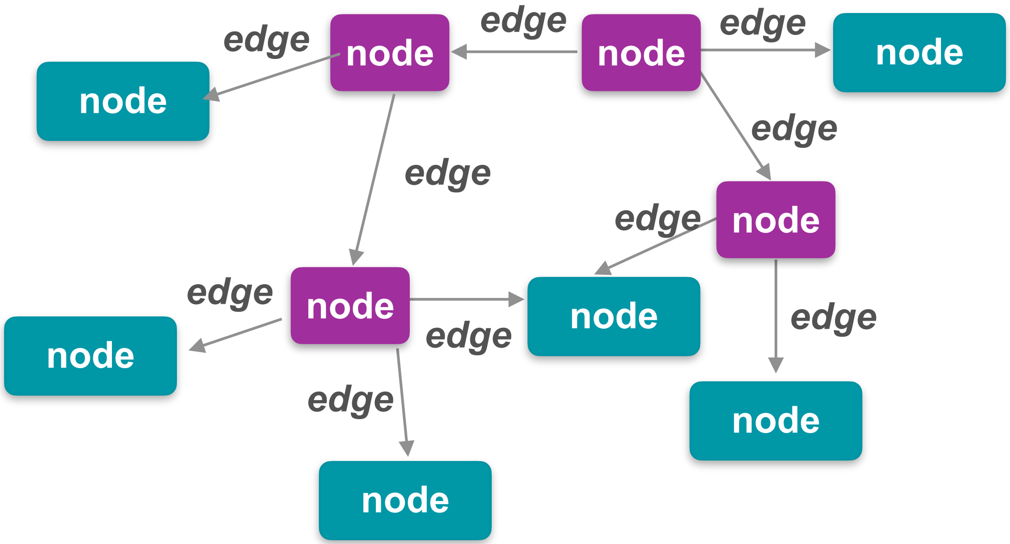
Deserialize the column, modify it, then write it back to storage

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

Storage Systems

Graph Databases

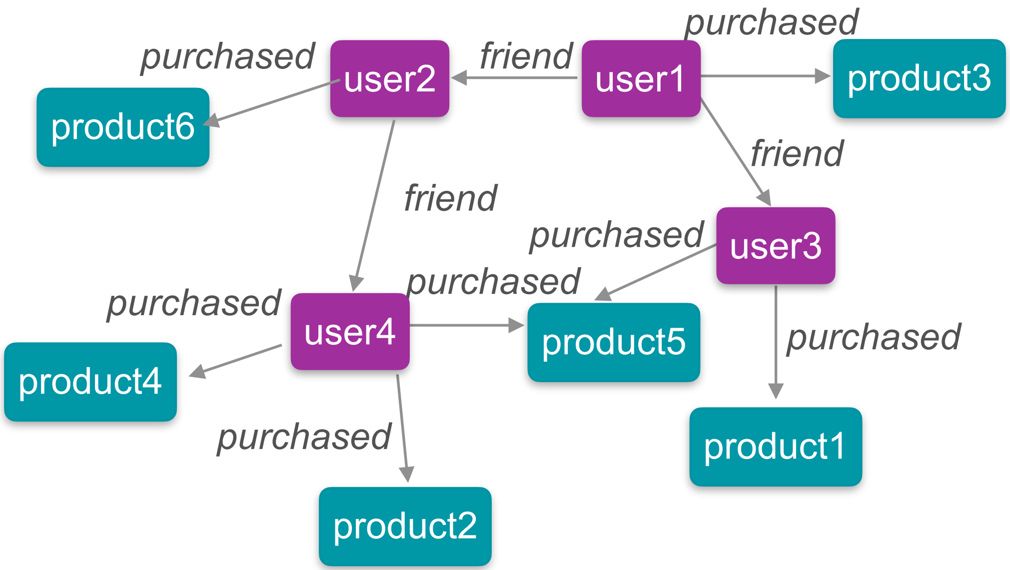
Graph Database



- 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

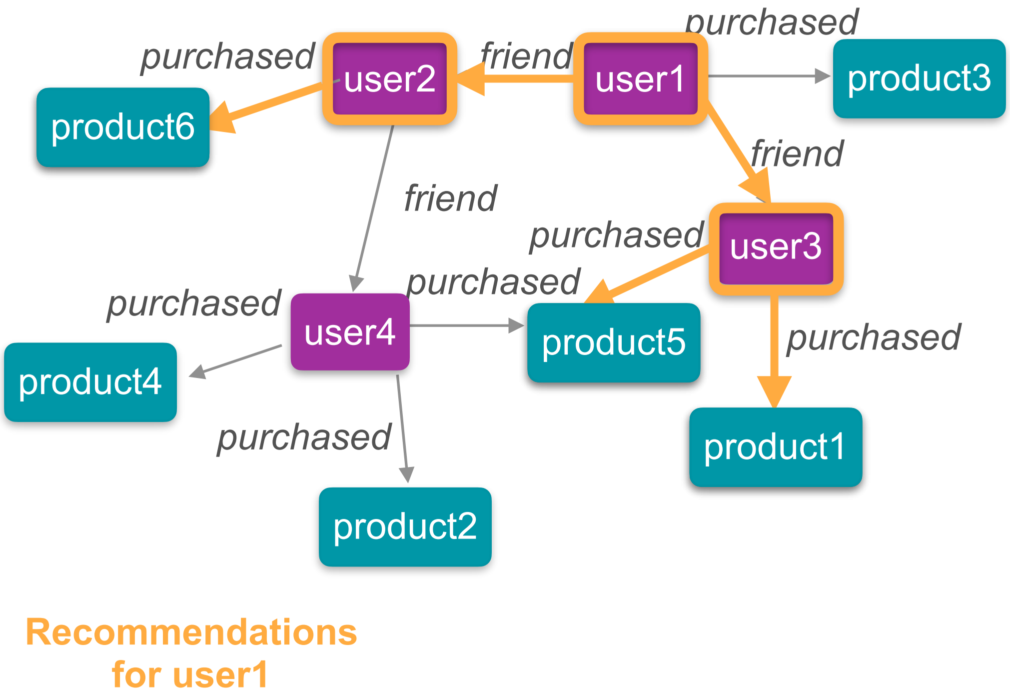
user	product
user1	product3
user2	product6
user3	product1
user3	product5
user4	product5
user4	product4
user4	product2

friendship

user	friend
user1	user3
user1	user2
user2	user4

Querying Data

Traverse the graph to query relationships

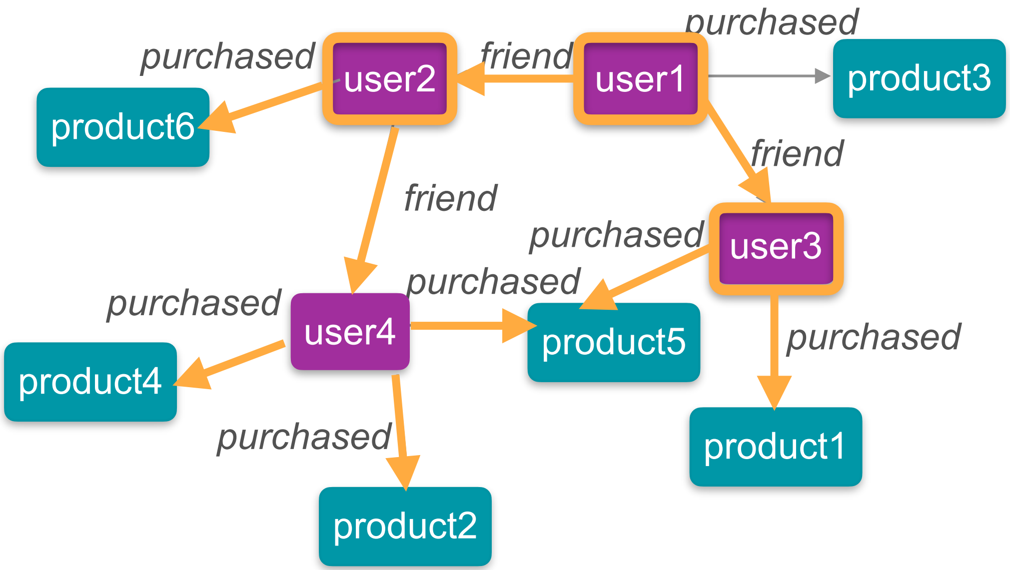


user	product
user1	product3
user2	product6
user3	product1
user3	product5
user4	product5
user4	product4
user4	product2

user	friend
user1	user3
user1	user2
user2	user4

Querying Data

Traverse the graph to query relationships



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

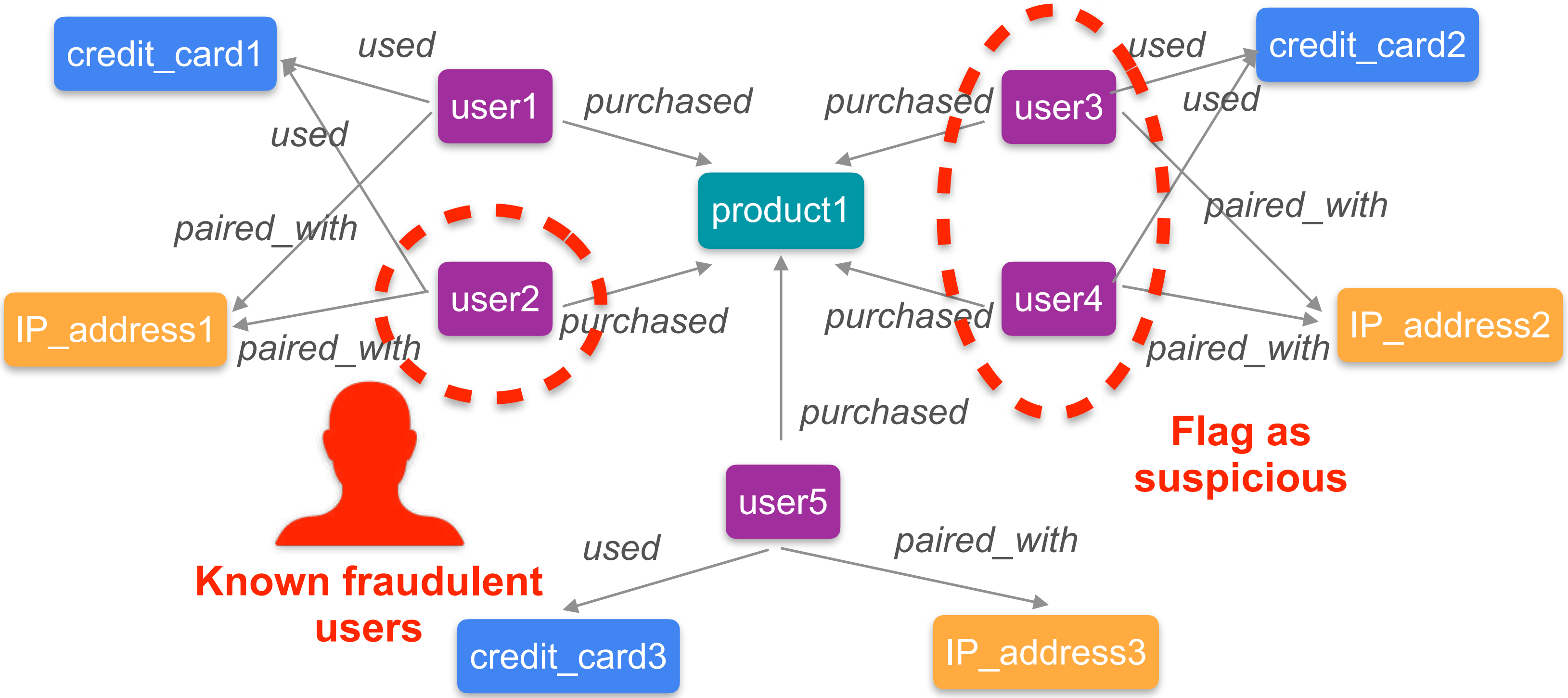
Less efficient in querying complex relationships!

```
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

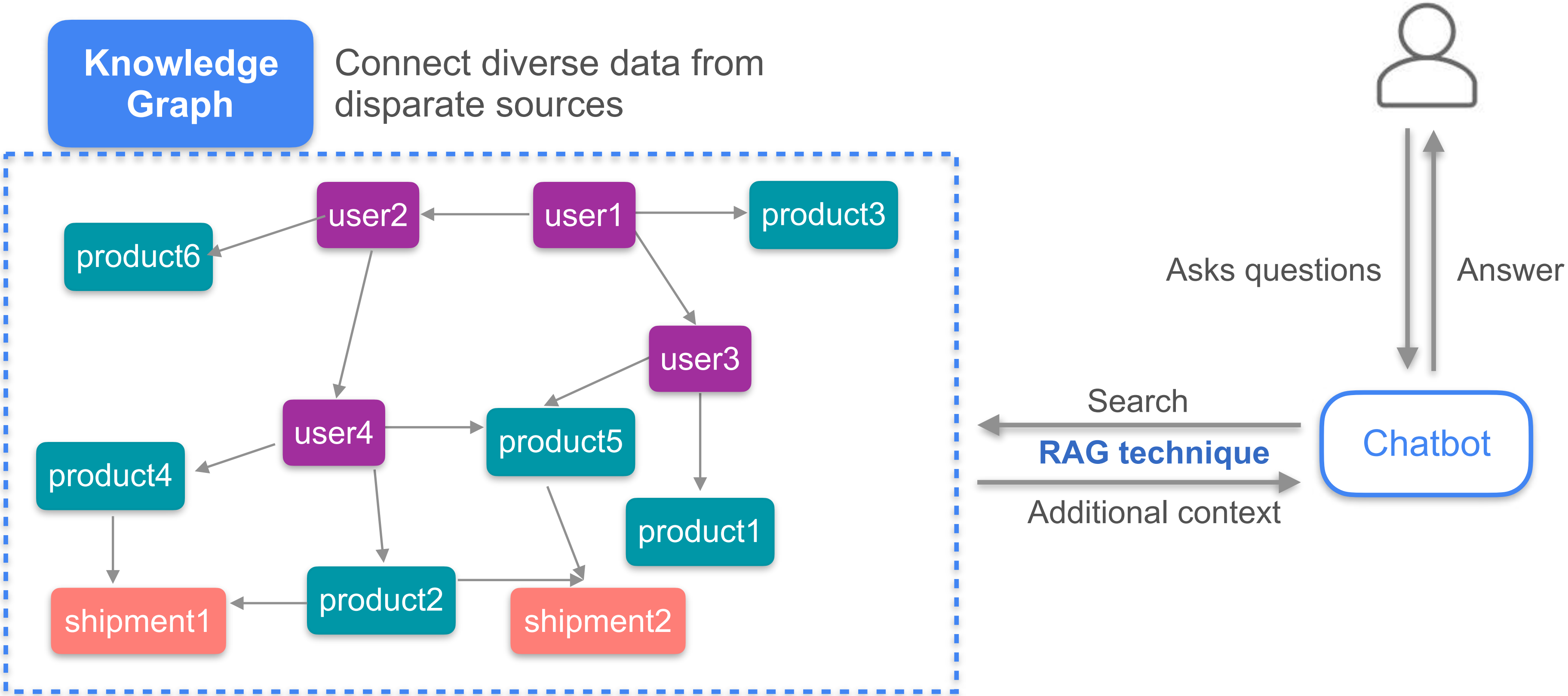
Use Case - Fraud Detection



Use Case - Knowledge Graph

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

DeepLearning.AI

Storage Systems

Vector Databases

Vector data

Consists of numerical values arranged in an array

Image

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

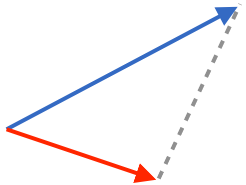
Capture semantic meaning of an item, like a text document or image

Original Data → ML Model → Vector embedding $\xrightarrow{\text{store}}$ Vector database

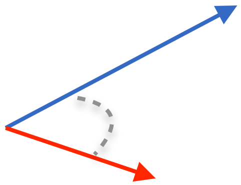
- Can convert an entire database of docs or text into embeddings
- Embeddings help you more efficiently find and retrieve similar items
- Example: Finding similar text
    - Compute embeddings for the query item
    - Database returns similar vectors (based on closeness)

Distance Metric

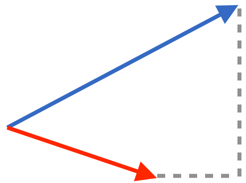
Vector database uses a distance metric to find similar vectors



Euclidean distance



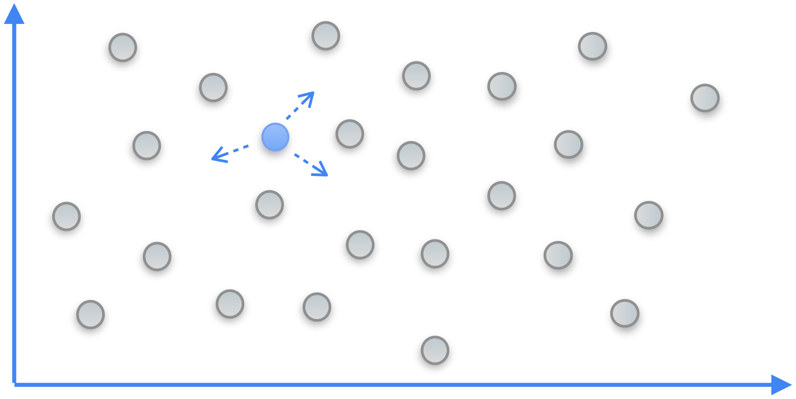
Cosine distance



Manhattan distance

Similarity Search - Popular Algorithm

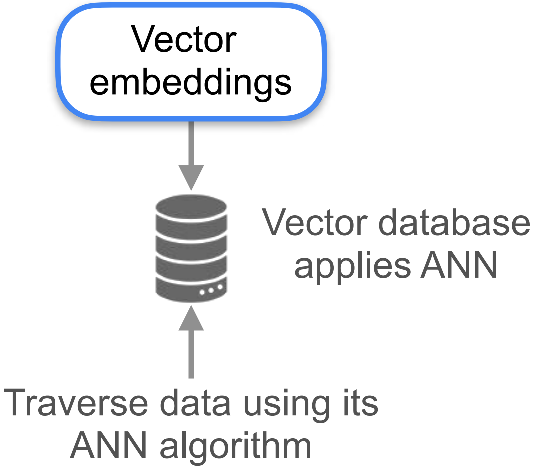
K-nearest neighbors (KNN)



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

ANN (Approximate Nearest Neighbors)

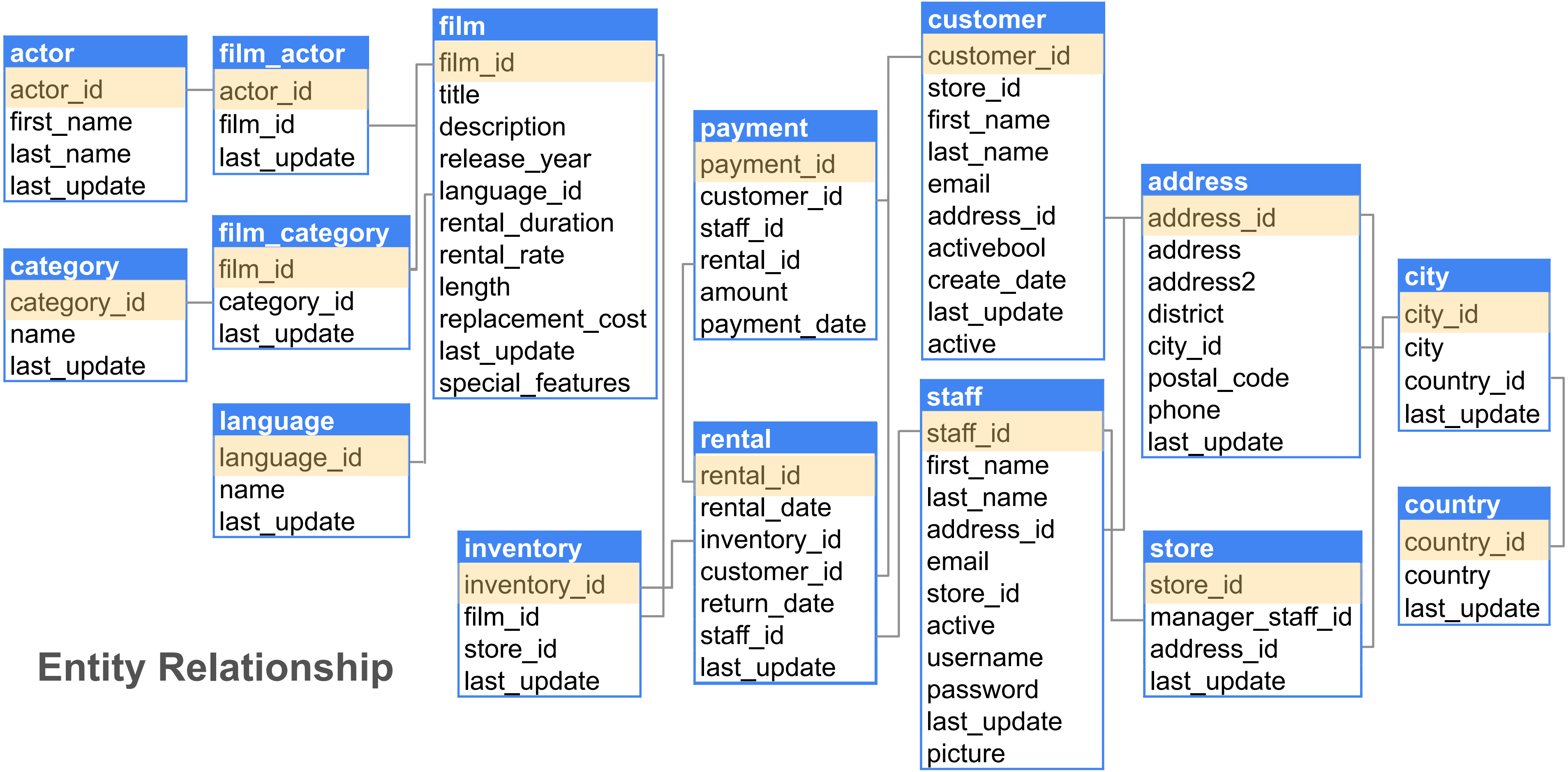
- Find a good guess for the nearest neighbors
- More efficient than K-NN
- Vector databases are built to support ANN algorithms



DeepLearning.AI

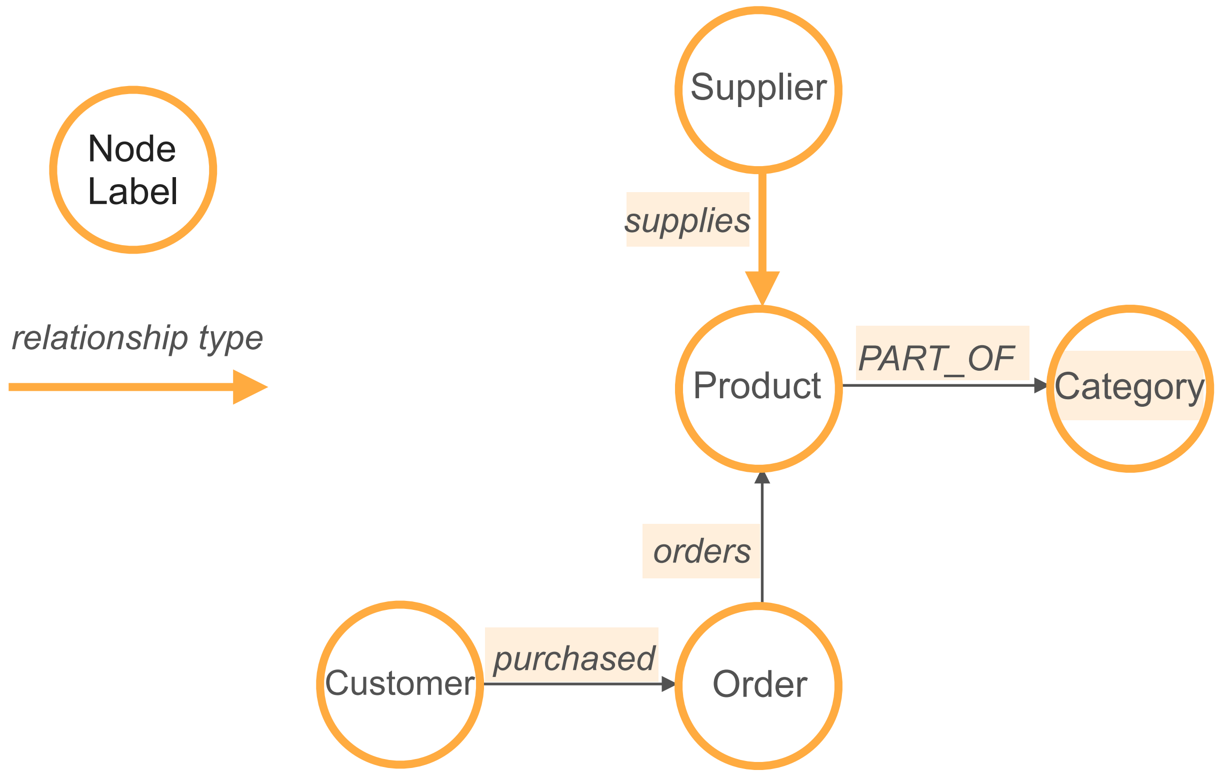
Storage Systems

Neo4j Graph Database & Cypher Query Language (Part 1)

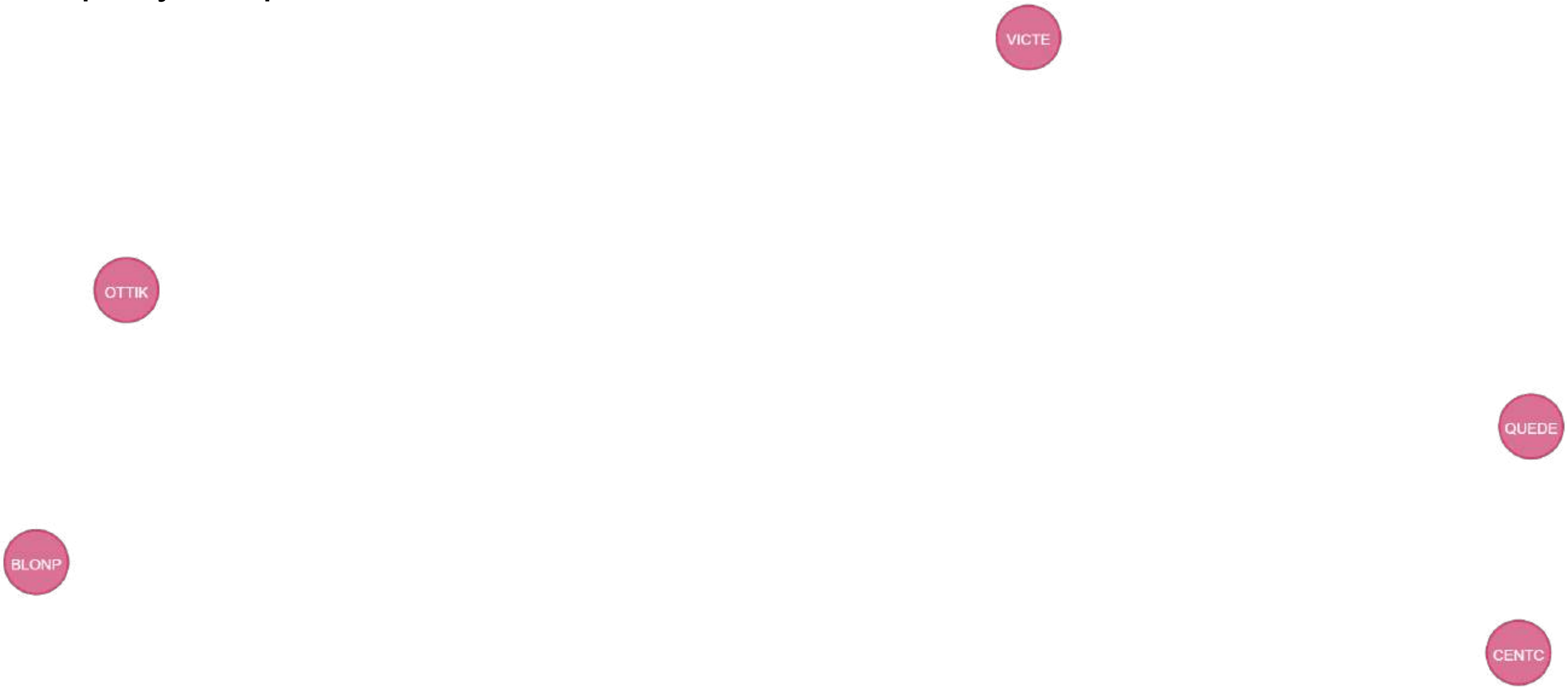


Entity Relationship

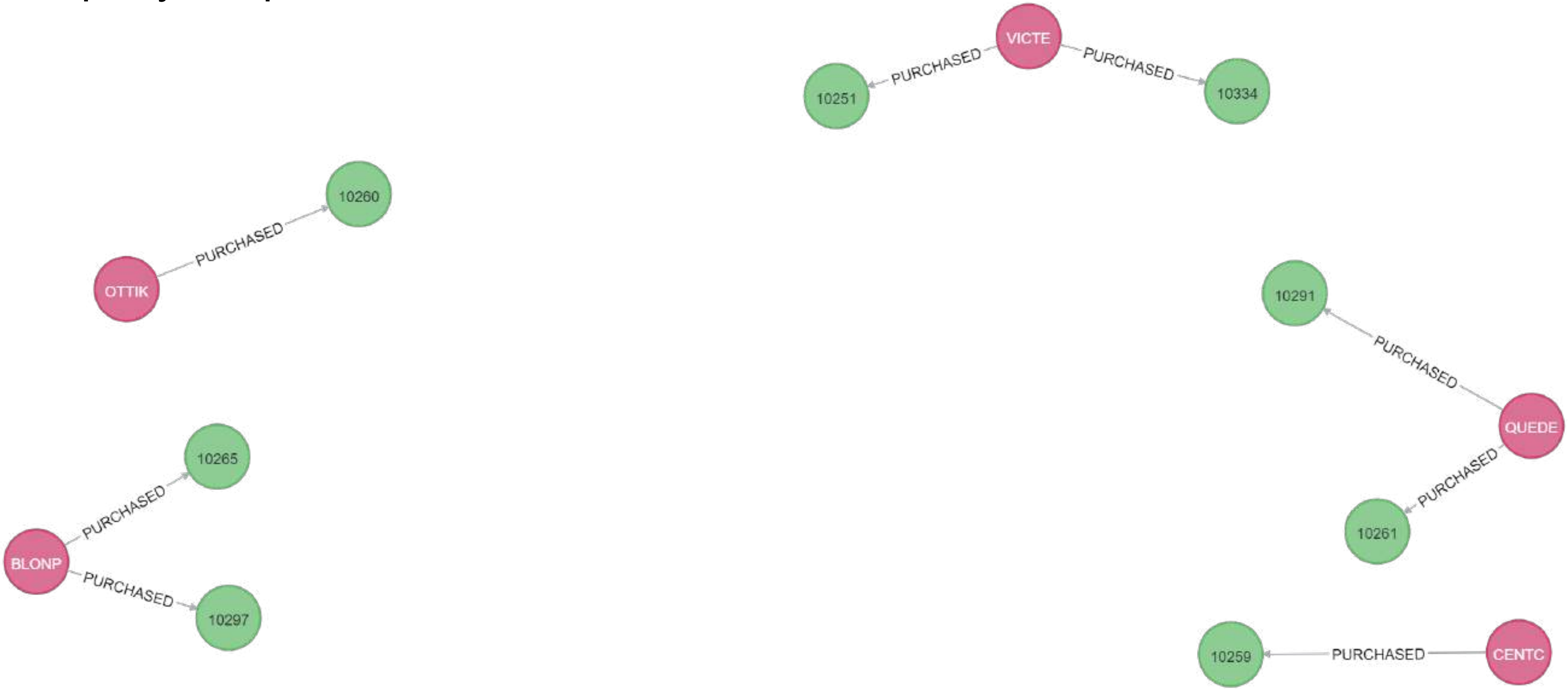
Property Graph Model



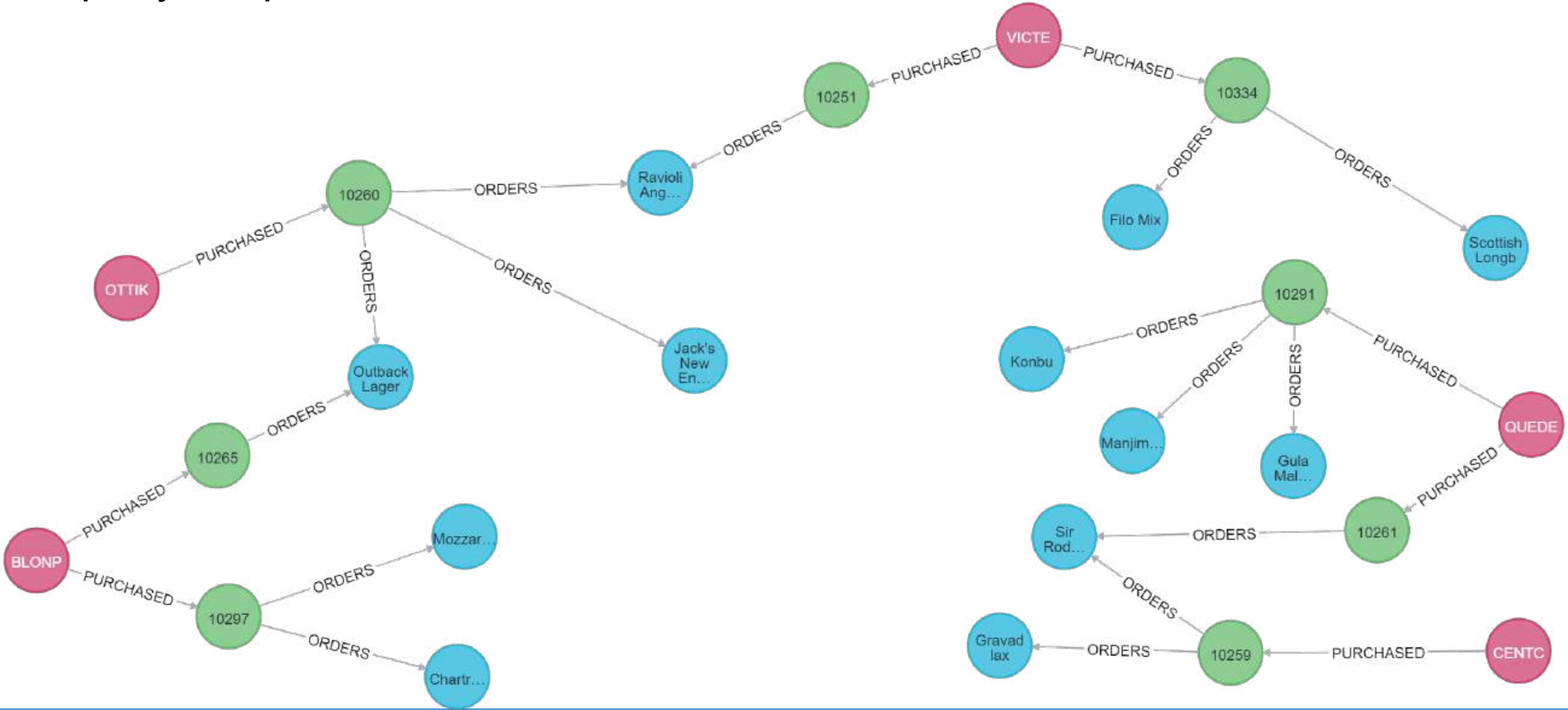
Property Graph Model



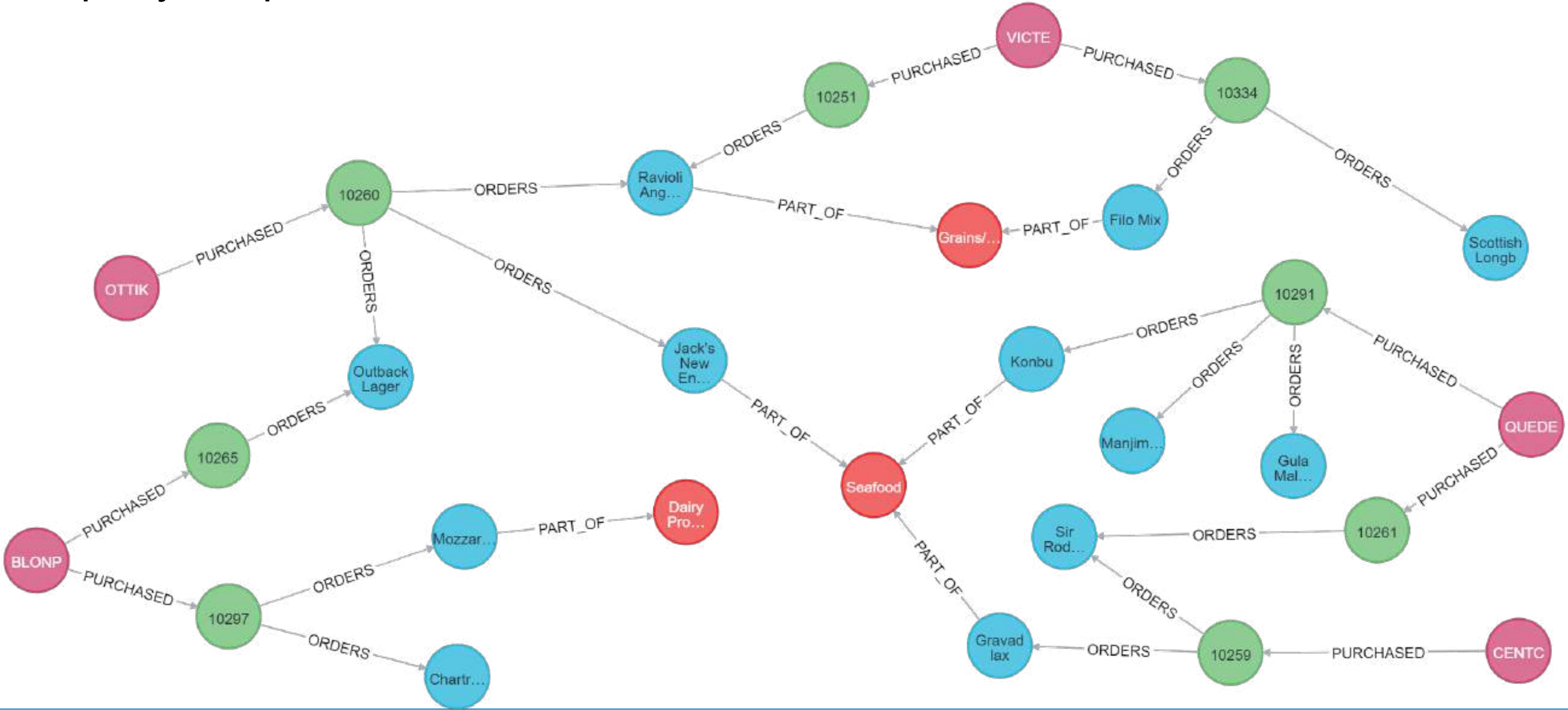
Property Graph Model



Property Graph Model

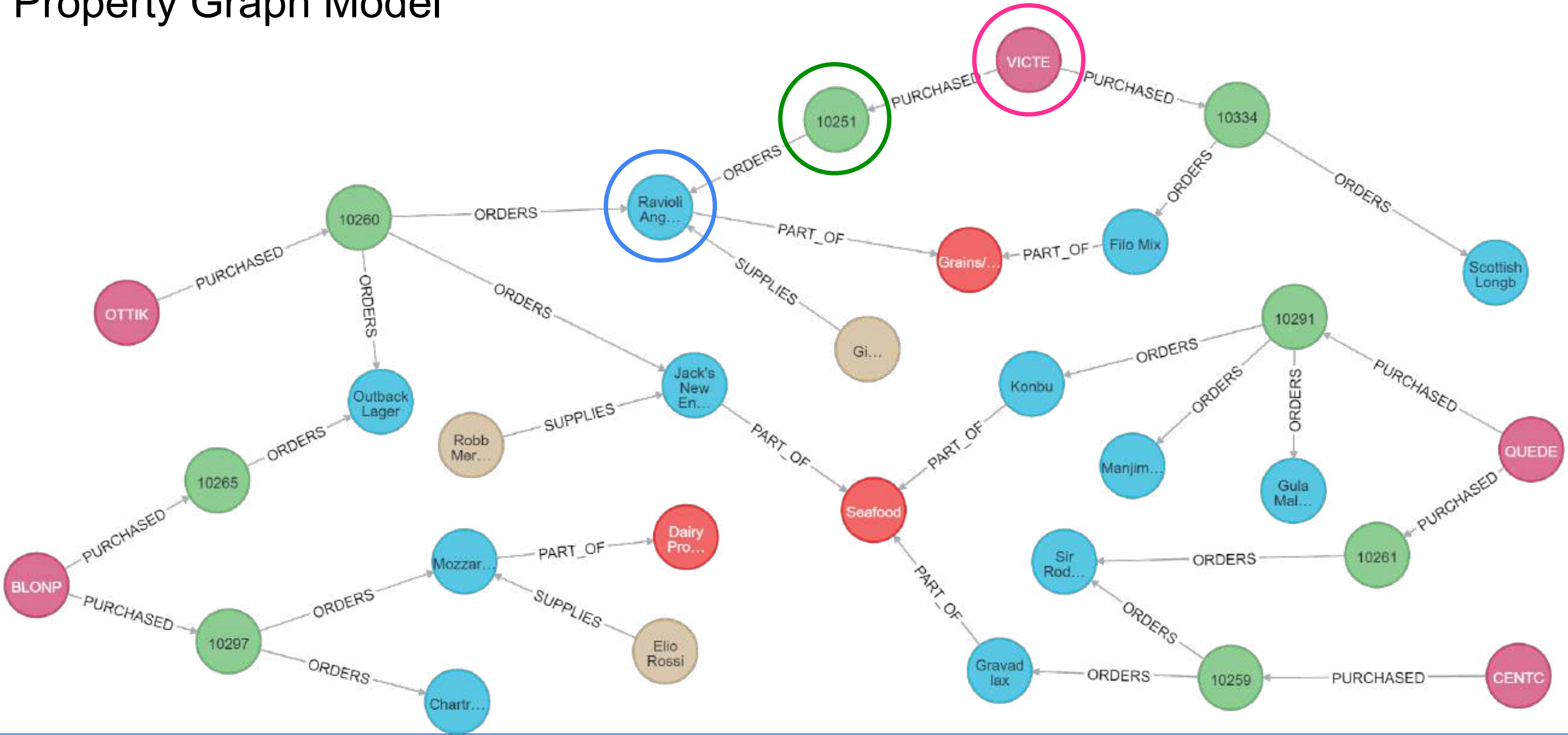


Property Graph Model

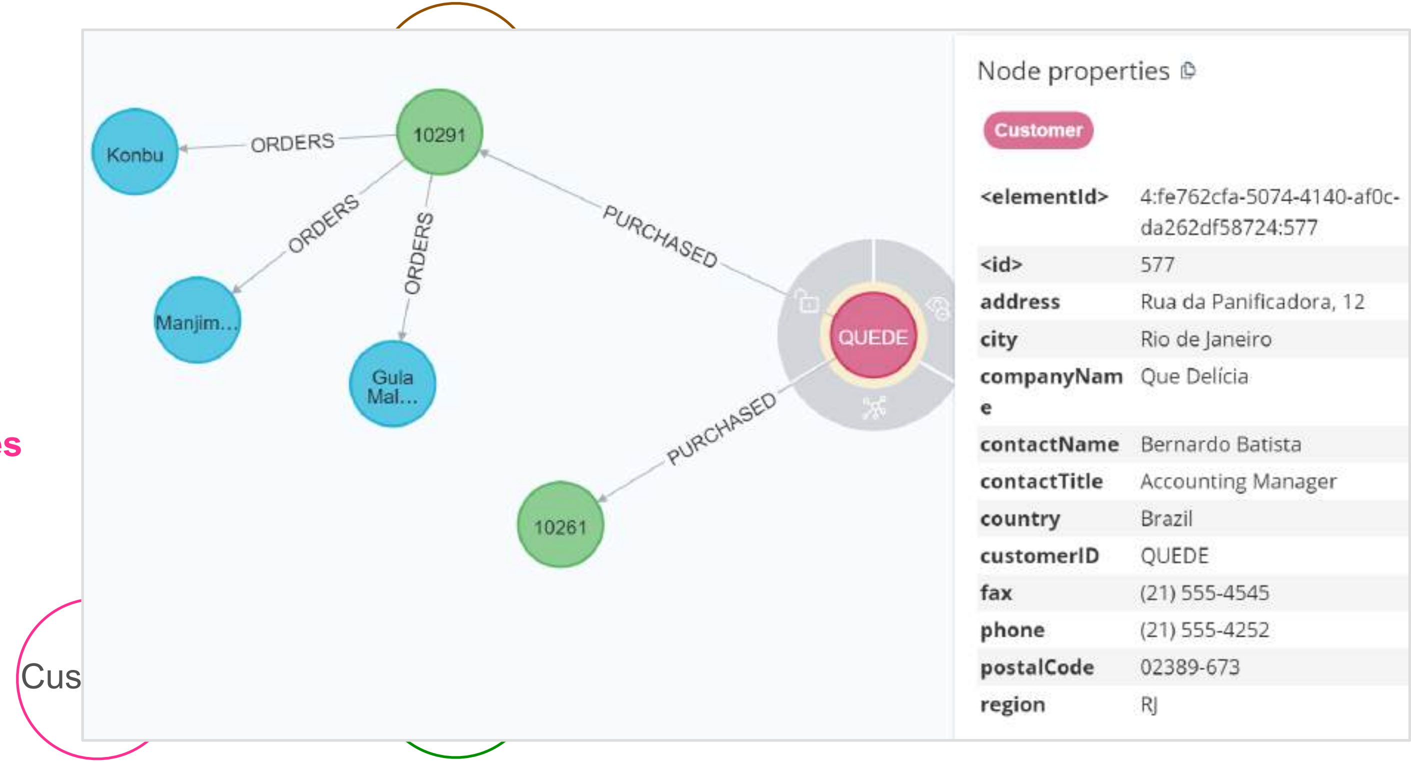


DeepLearning.AI

Property Graph Model



Property Graph Model



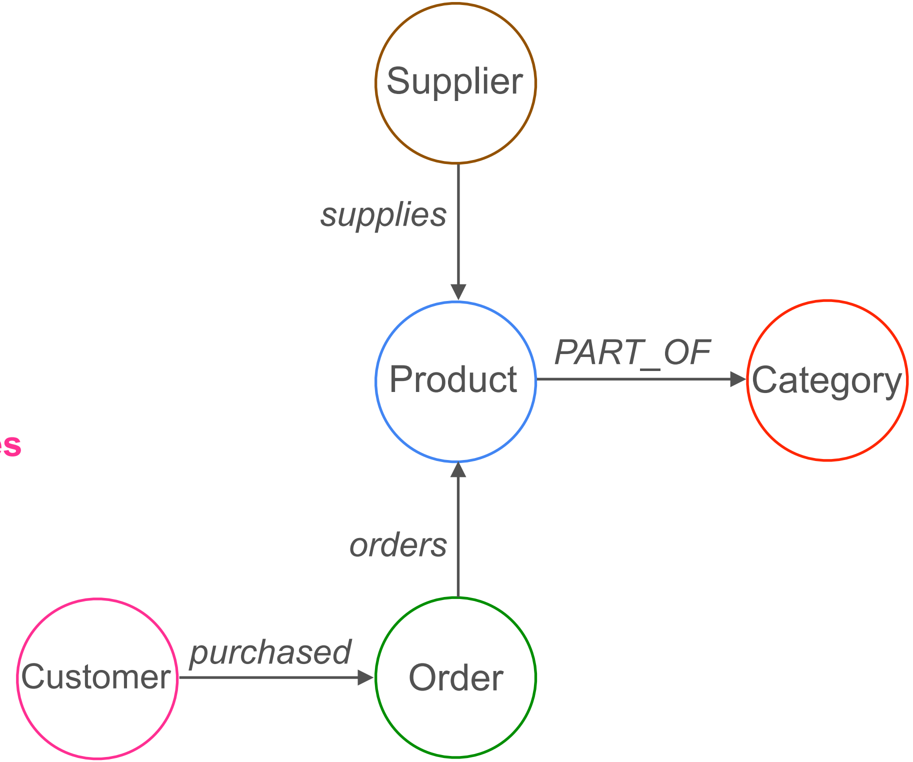
Customer Properties

address
city
companyName
contactName
contactTitle
country
customerID
.....

Property Graph Model

Product Properties
productID
productName
unitPrice
unitsInStock
unitesOnOrder

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

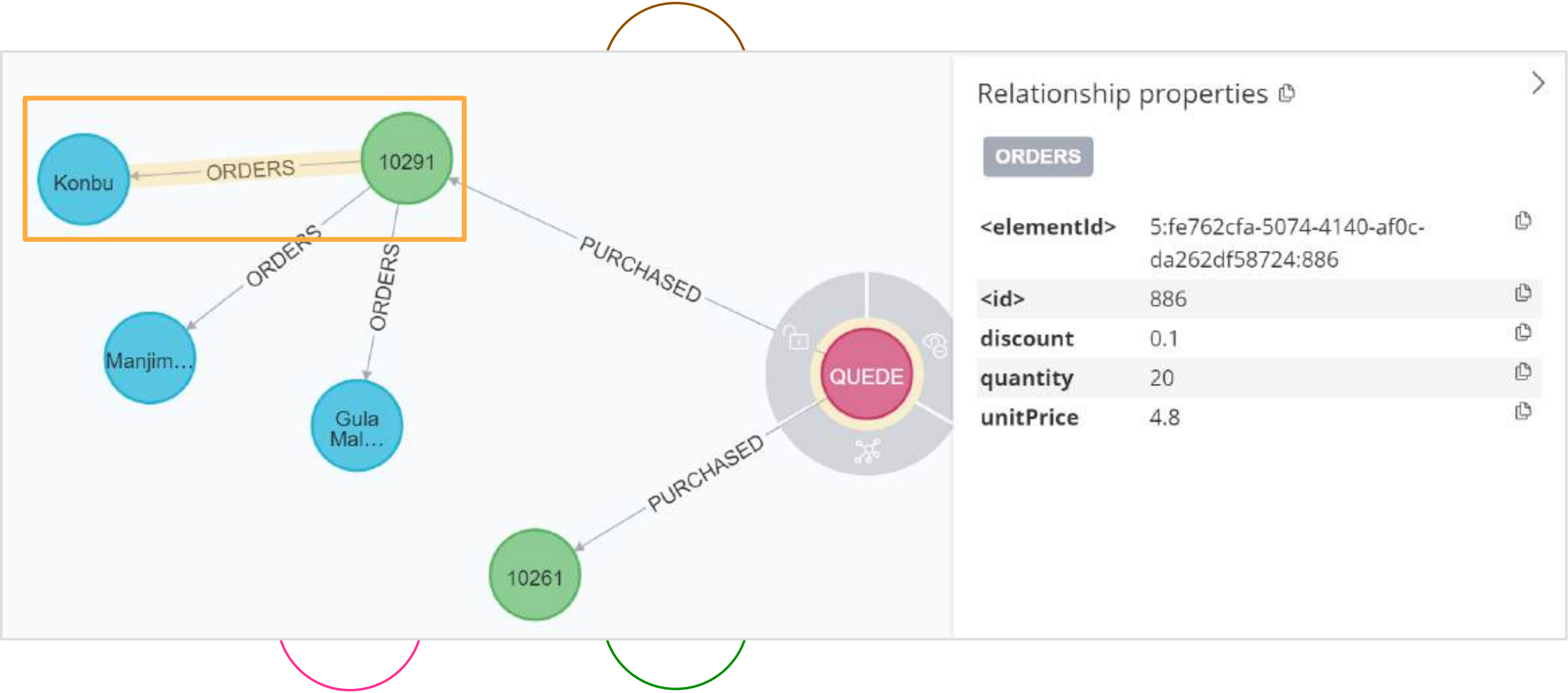


Supplier Properties
address
city
contactName
fax
region
supplierID
postalCode

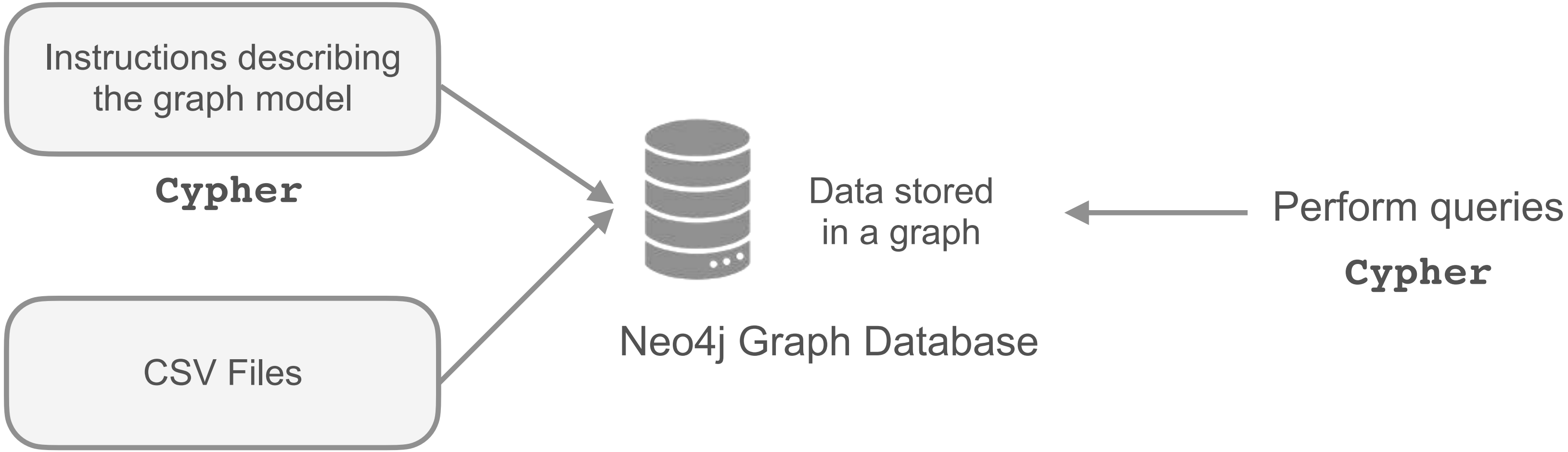
Category Properties
categoryName

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.

DeepLearning.AI

Storage Systems

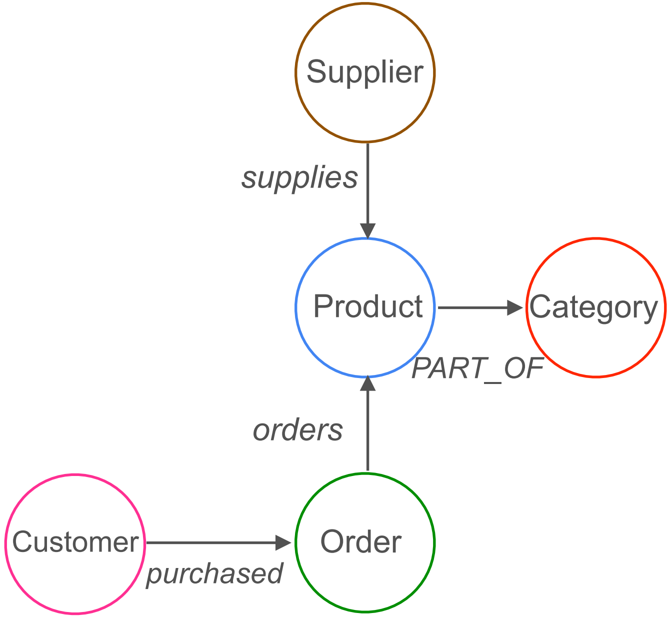
Neo4j Graph Database & Cypher Query Language (Part 2)

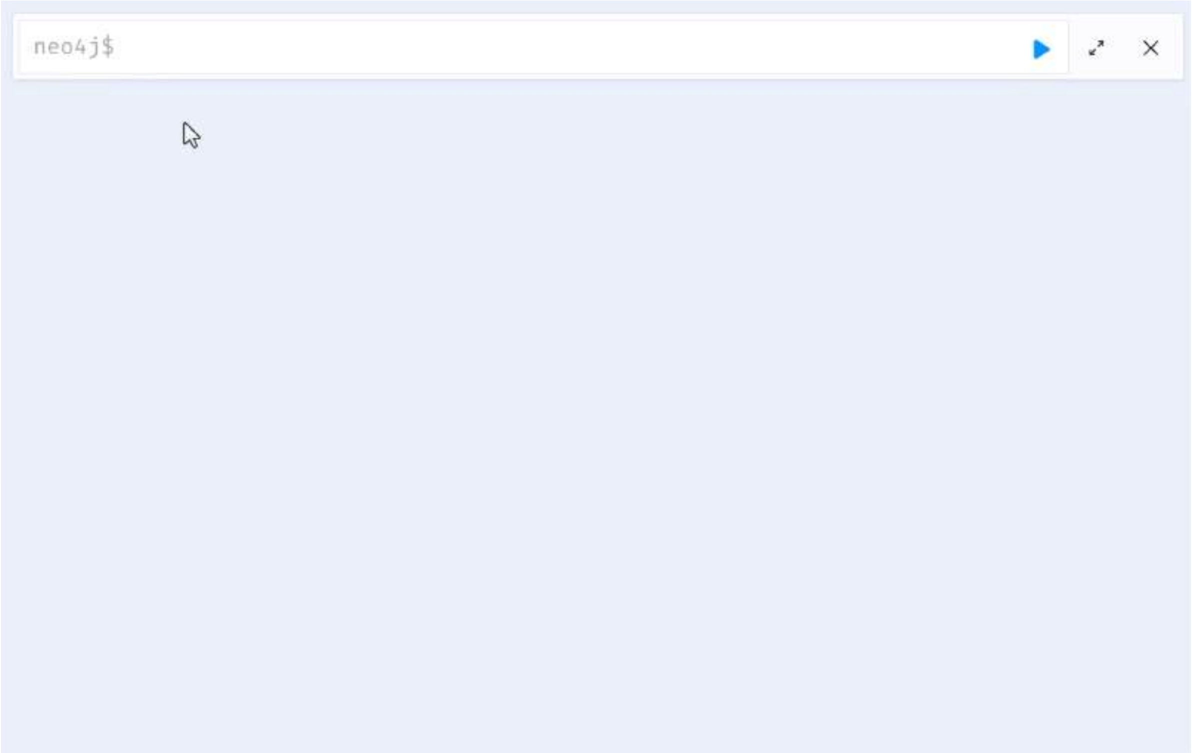
MATCH Statement

Retrieve all nodes

```
MATCH pattern RETURN result
```

node	( )



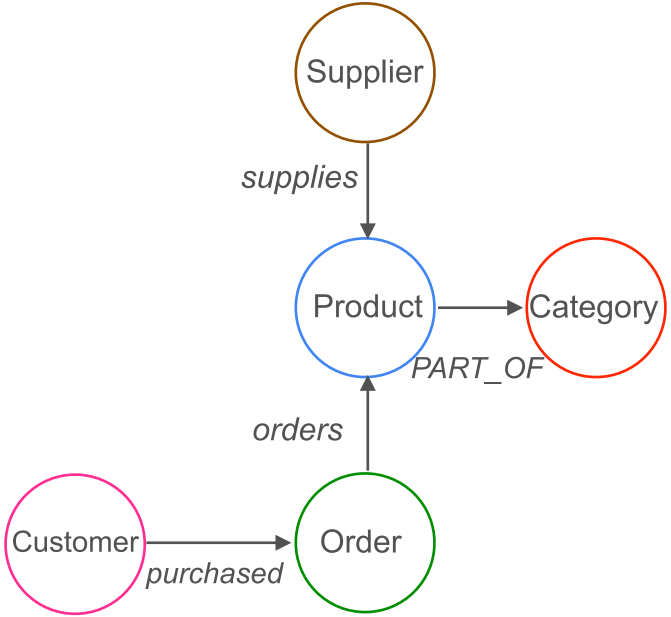


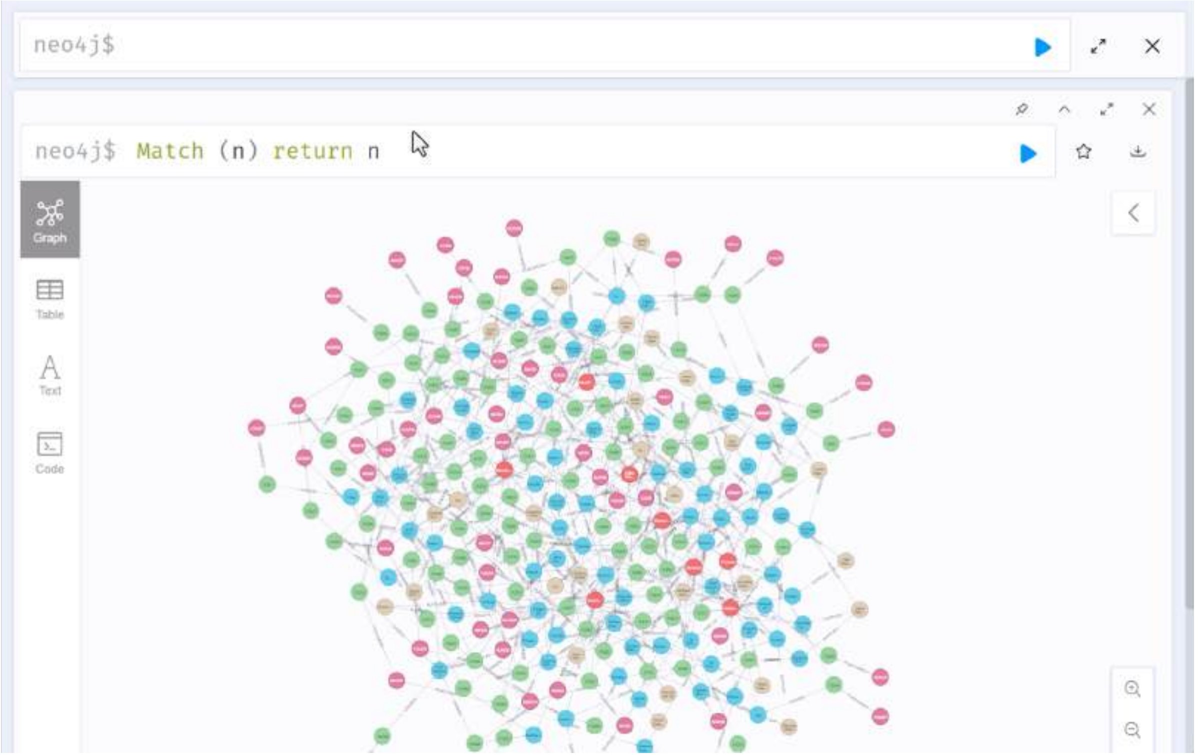
`MATCH pattern RETURN result`

node	( )

MATCH Statement

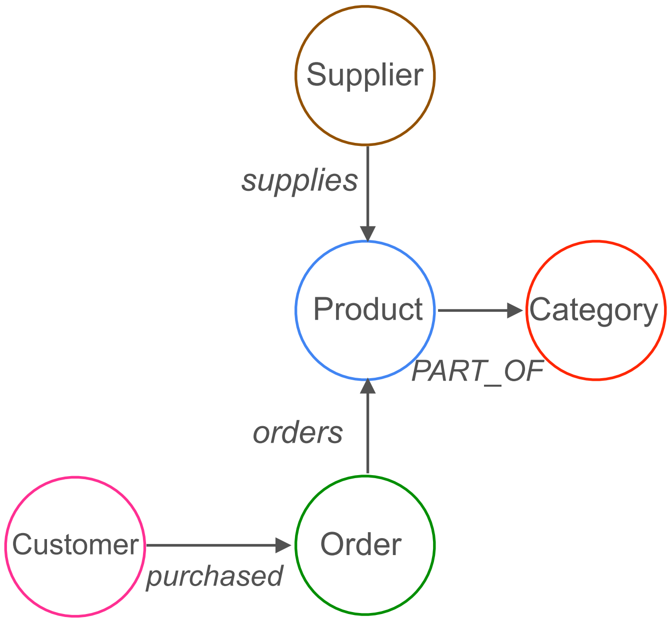
Get the total number of nodes





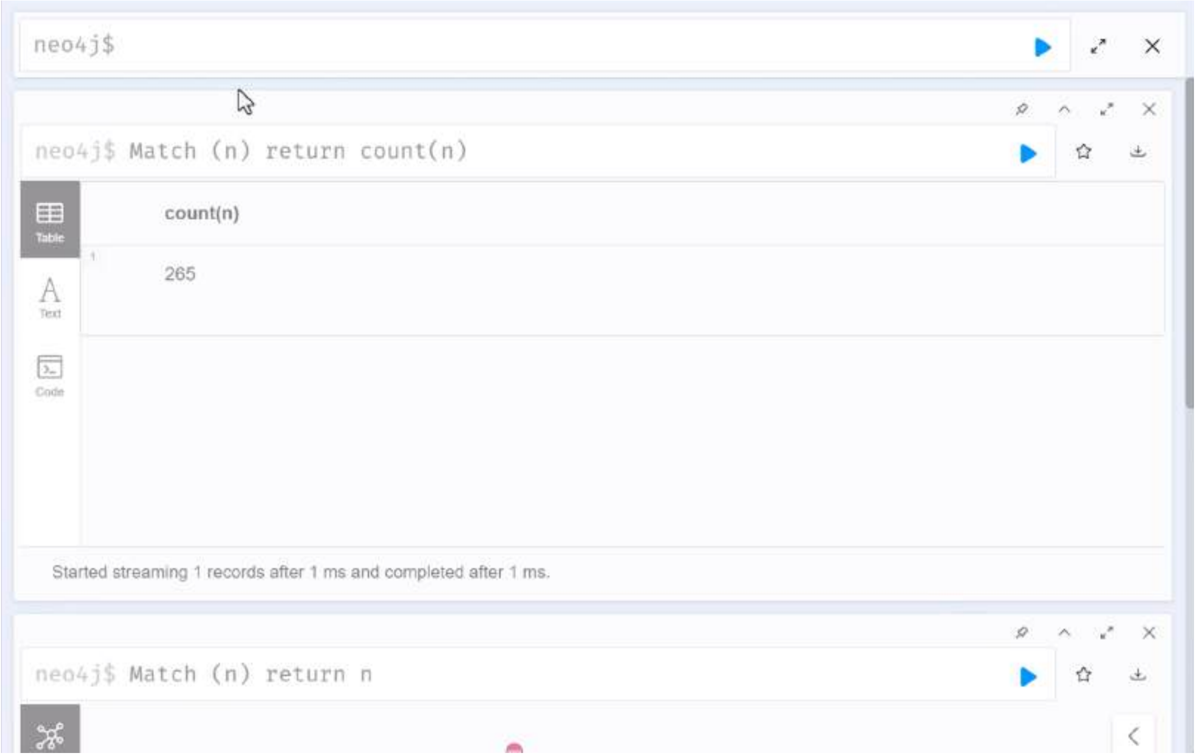
MATCH *pattern* RETURN *result*

node	( )



MATCH Statement

Explore the node labels using the `labels` function

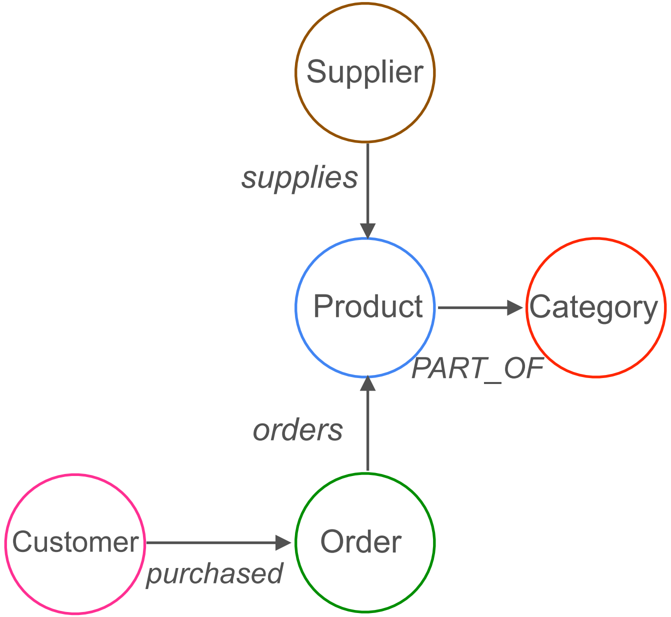


MATCH Statement

Specify the label of the node

`MATCH` *`pattern`* `RETURN` *`result`*

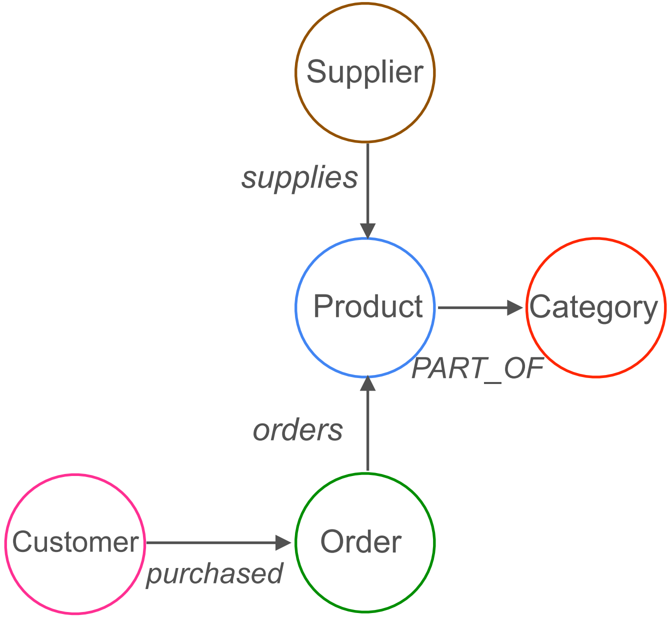
node	( )



MATCH Statement

`MATCH` *`pattern`* `RETURN` *`result`*

node	( )

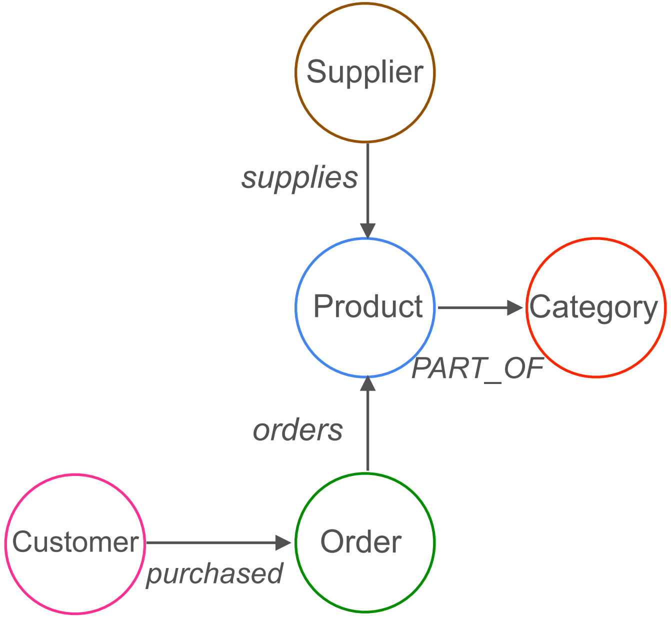


Explore the properties of each order node using the `Properties` function

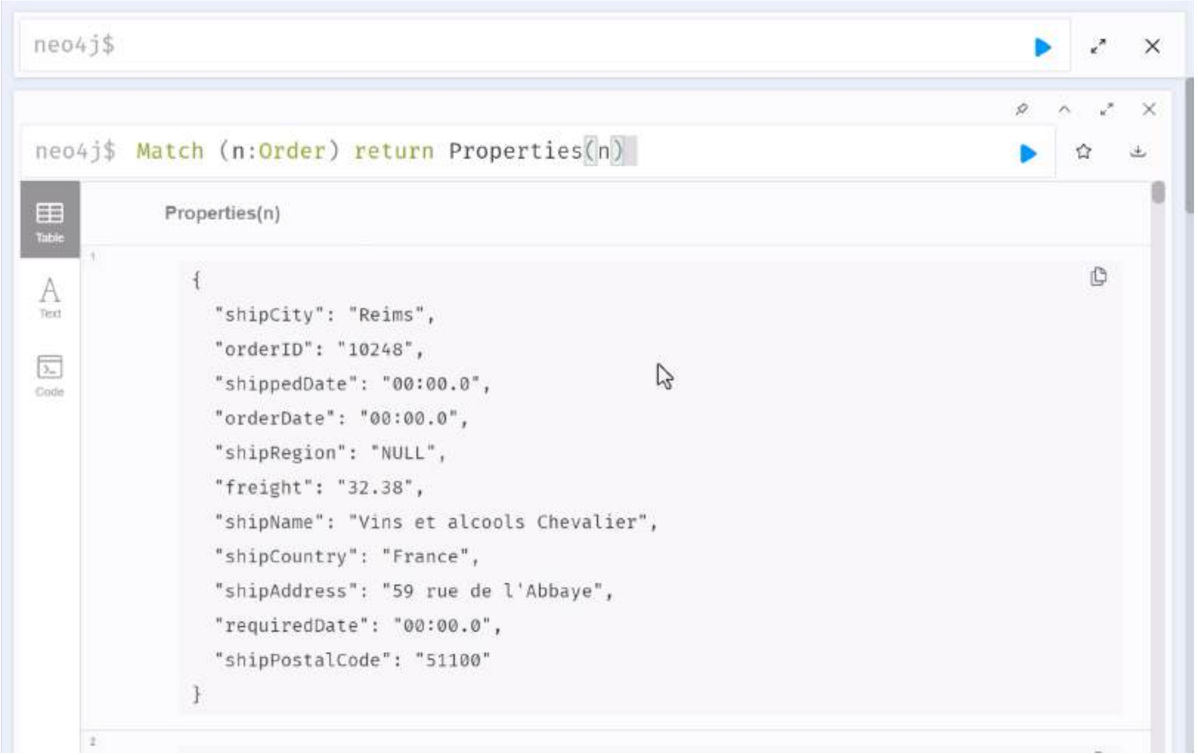
MATCH Statement

MATCH *pattern* RETURN *result*

node	( )

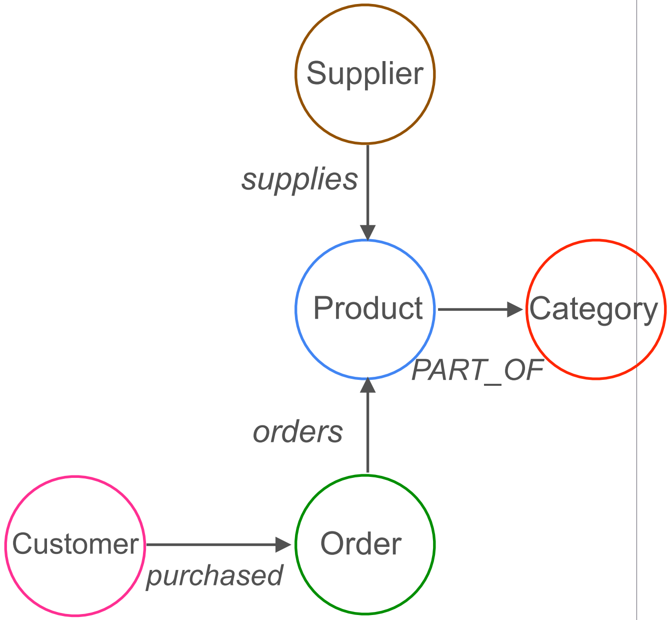


Explore the properties of each order node using the `Properties` function



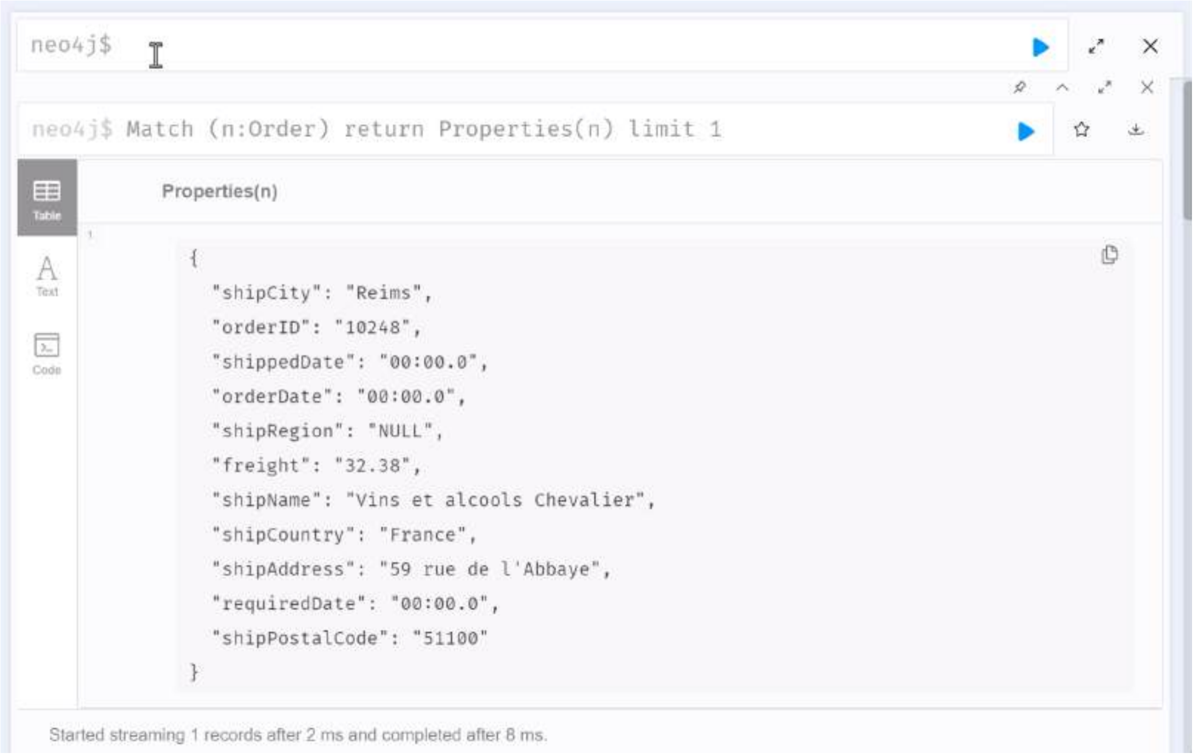
`MATCH` *`pattern`* `RETURN` *`result`*

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



MATCH Statement

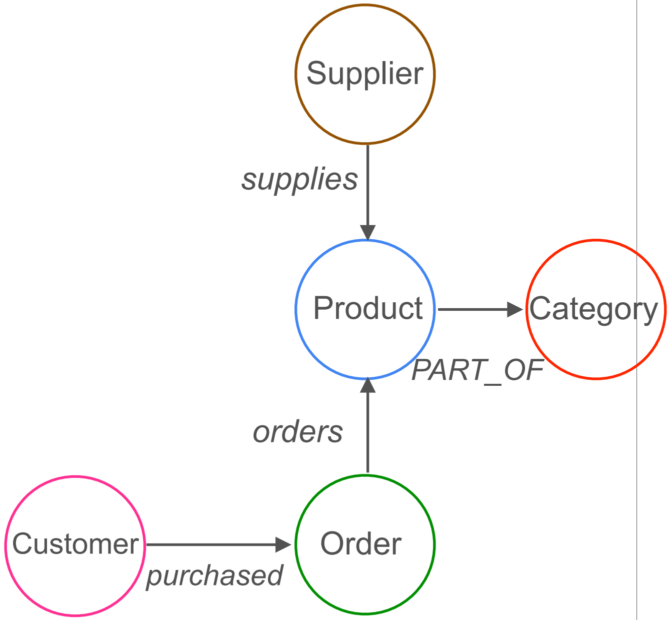
Count all the directed paths



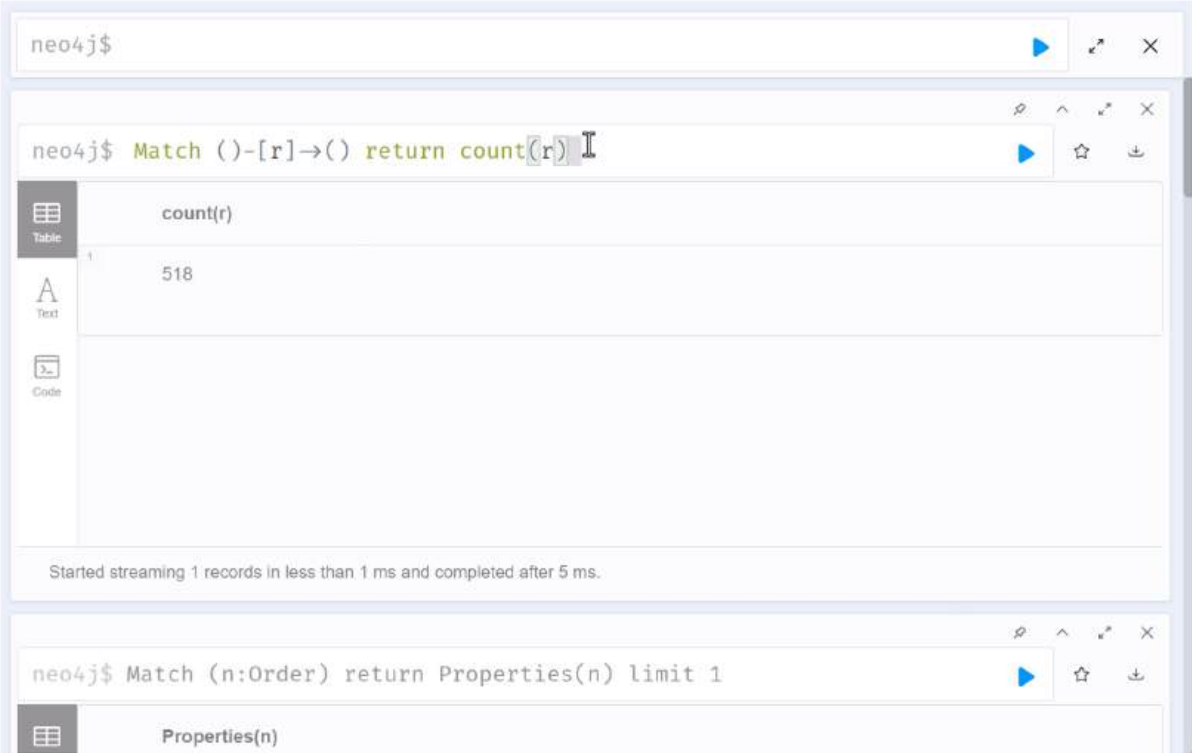
MATCH Statement

MATCH *pattern* RETURN *result*

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



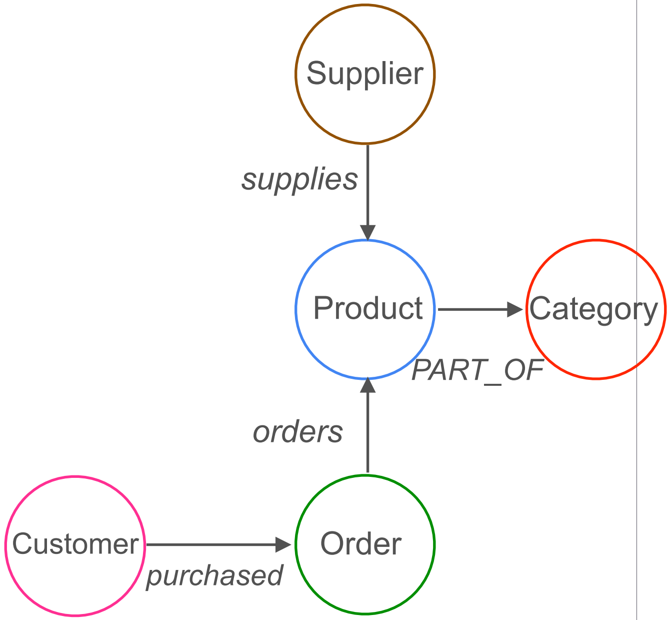
Return the types of relationships



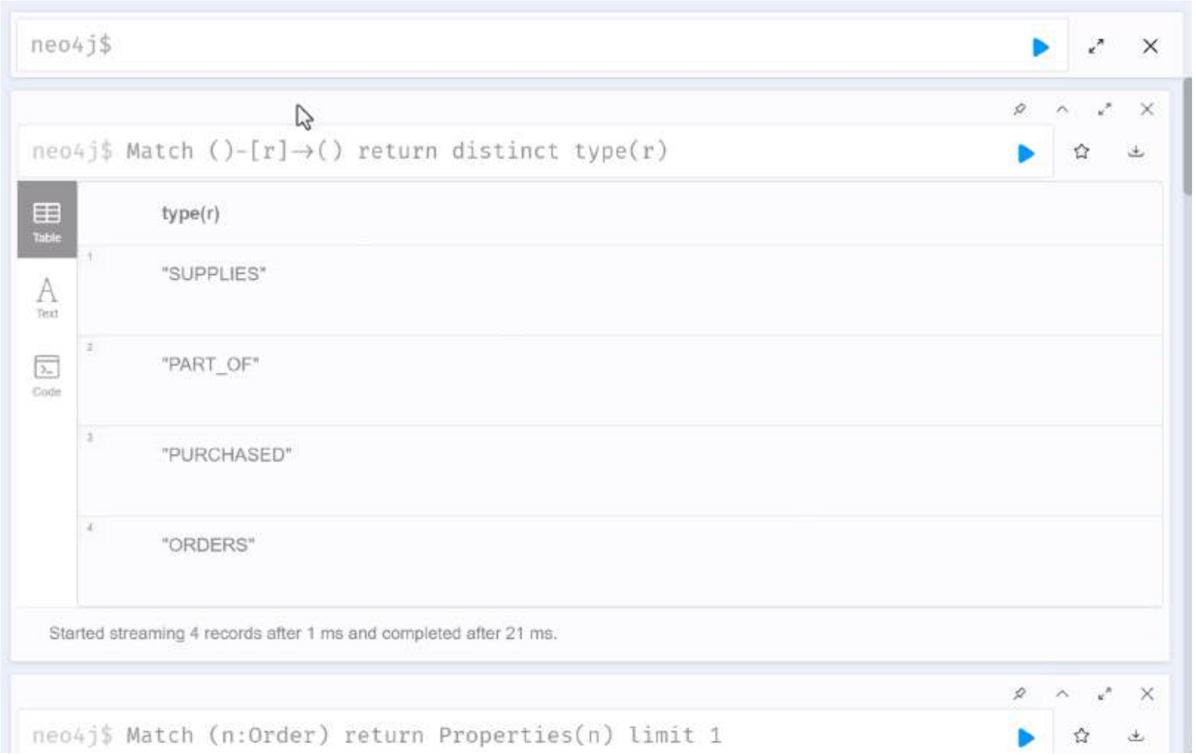
MATCH Statement

MATCH *pattern* RETURN *result*

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



Specify the type of the relationship

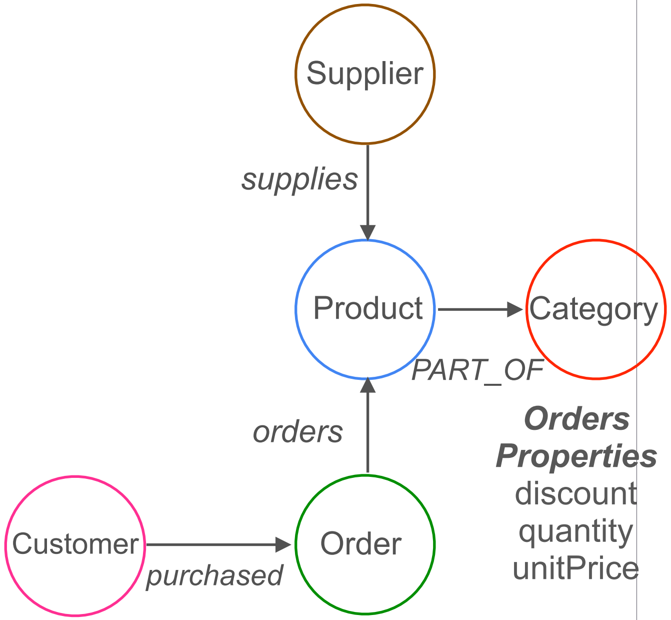


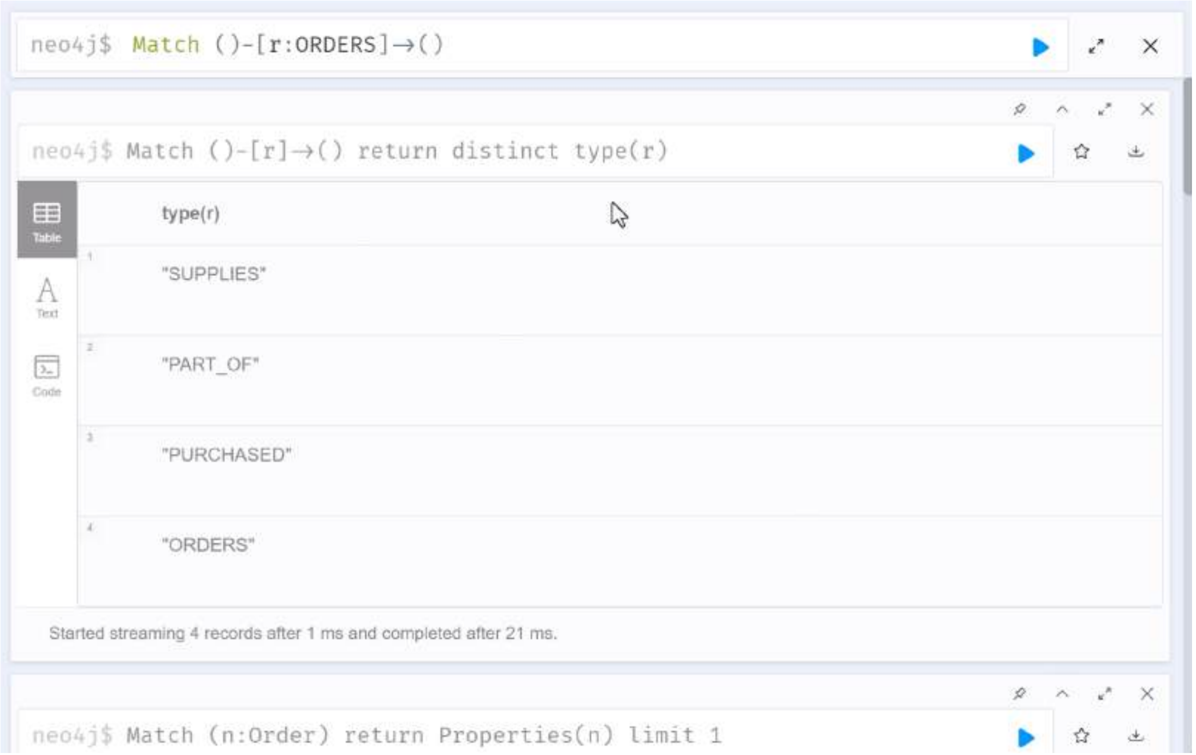
MATCH Statement

Return the properties of a relationship

`MATCH` *`pattern`* `RETURN` *`result`*

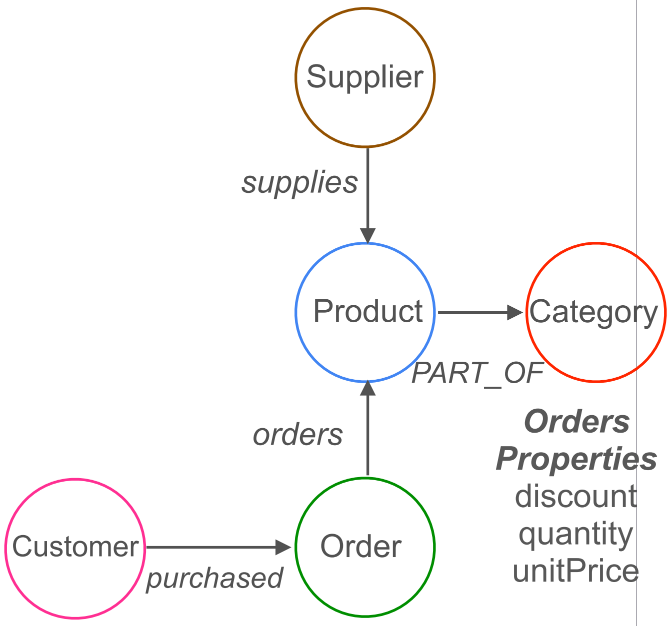
node	( )
relationship	[ ]
path	(source node)-[r]->(target node)





`MATCH` *`pattern`* `RETURN` *`result`*

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



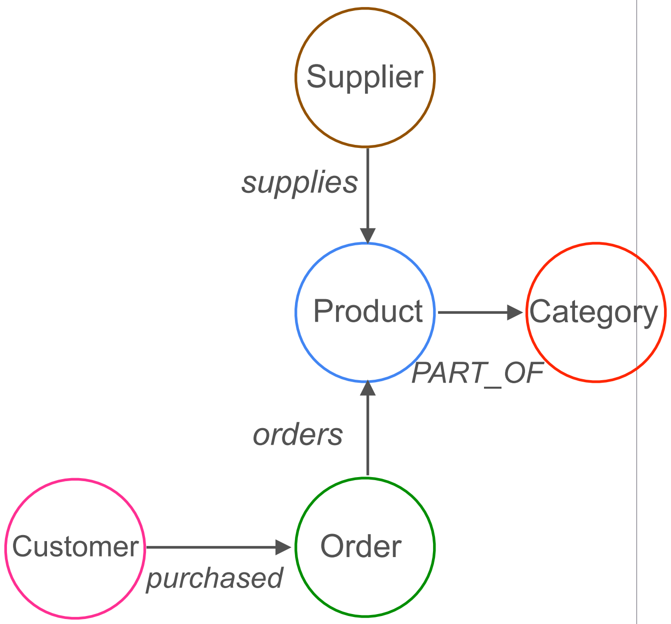
MATCH Statement

Return the properties of a relationship

MATCH Statement

`MATCH pattern RETURN result`

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

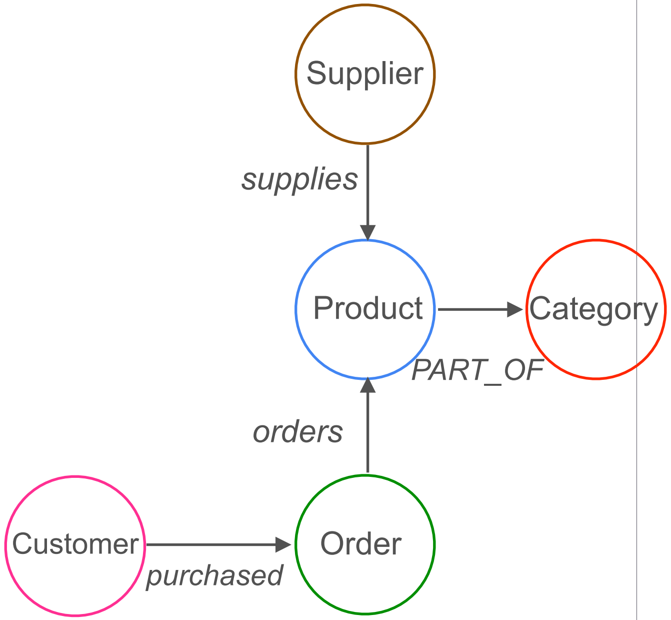


Get the average price for all orders grouped by product category

MATCH Statement

`MATCH` *`pattern`* `RETURN` *`result`*

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

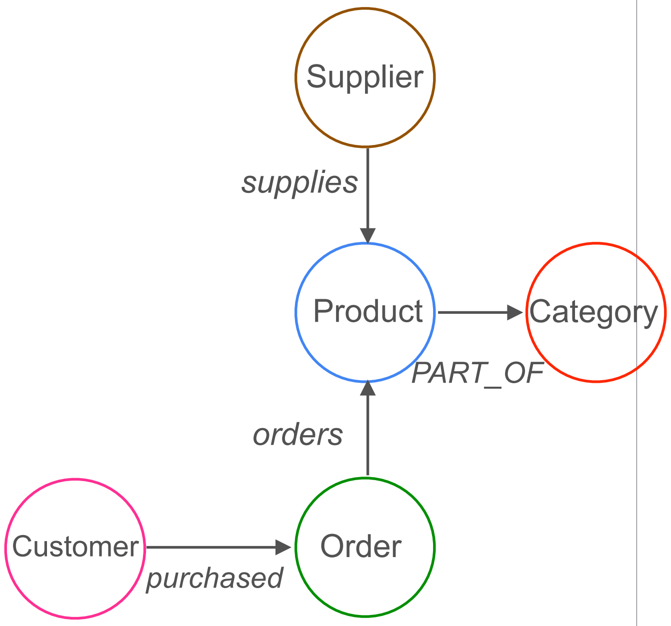


Get the average price for all orders grouped by product category

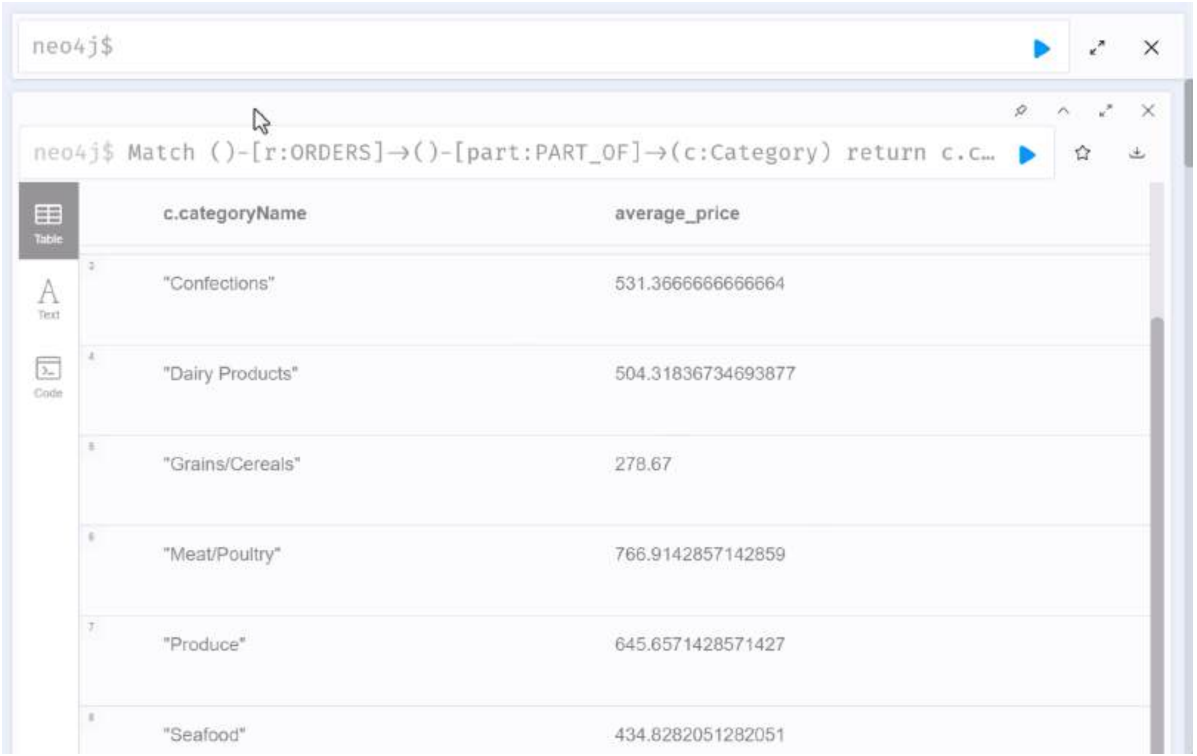
MATCH Statement

MATCH *pattern* RETURN *result*

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



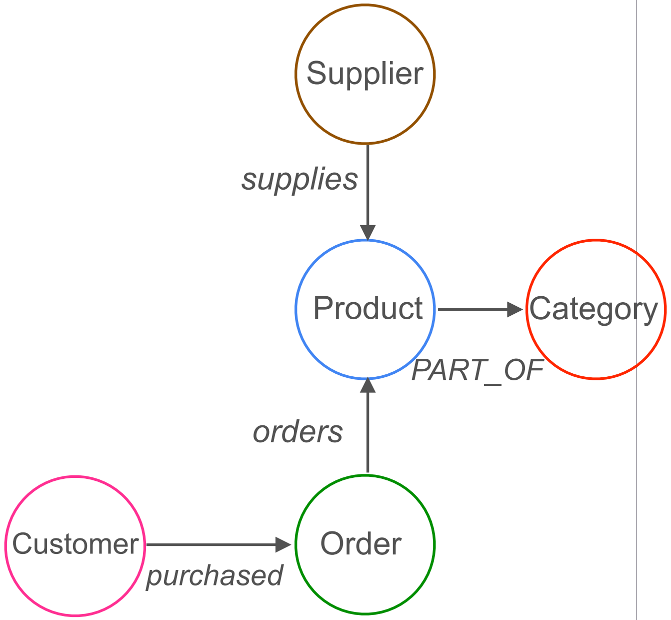
Retrieve the product name and product unit price of all products that belong to category "Meat/Poultry"



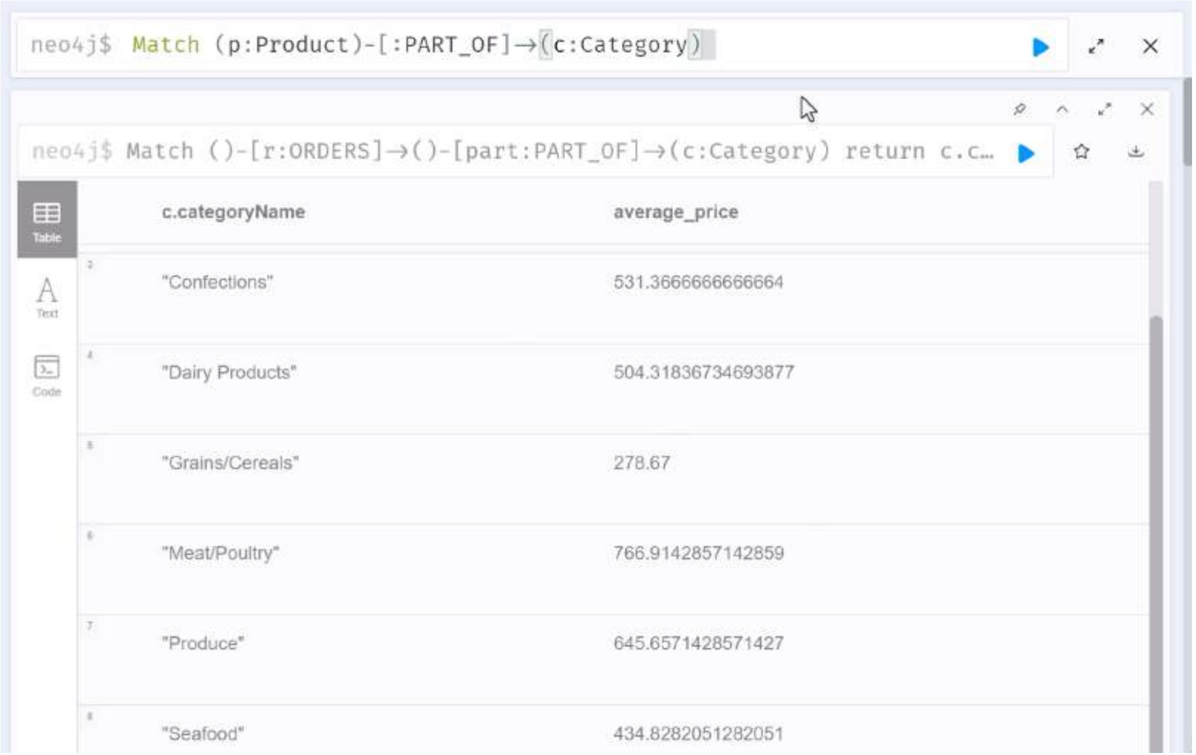
MATCH Statement

MATCH *pattern* RETURN *result*

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



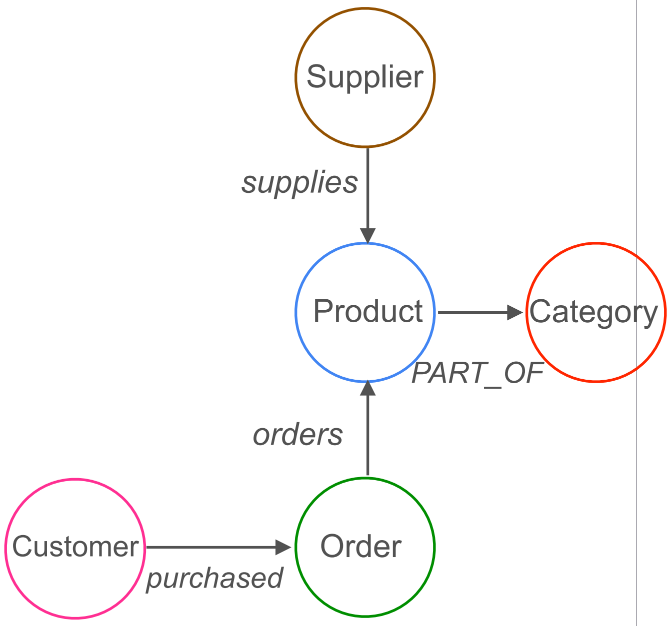
Retrieve the product name and product unit price of all products that belong to category “Meat/Poultry”



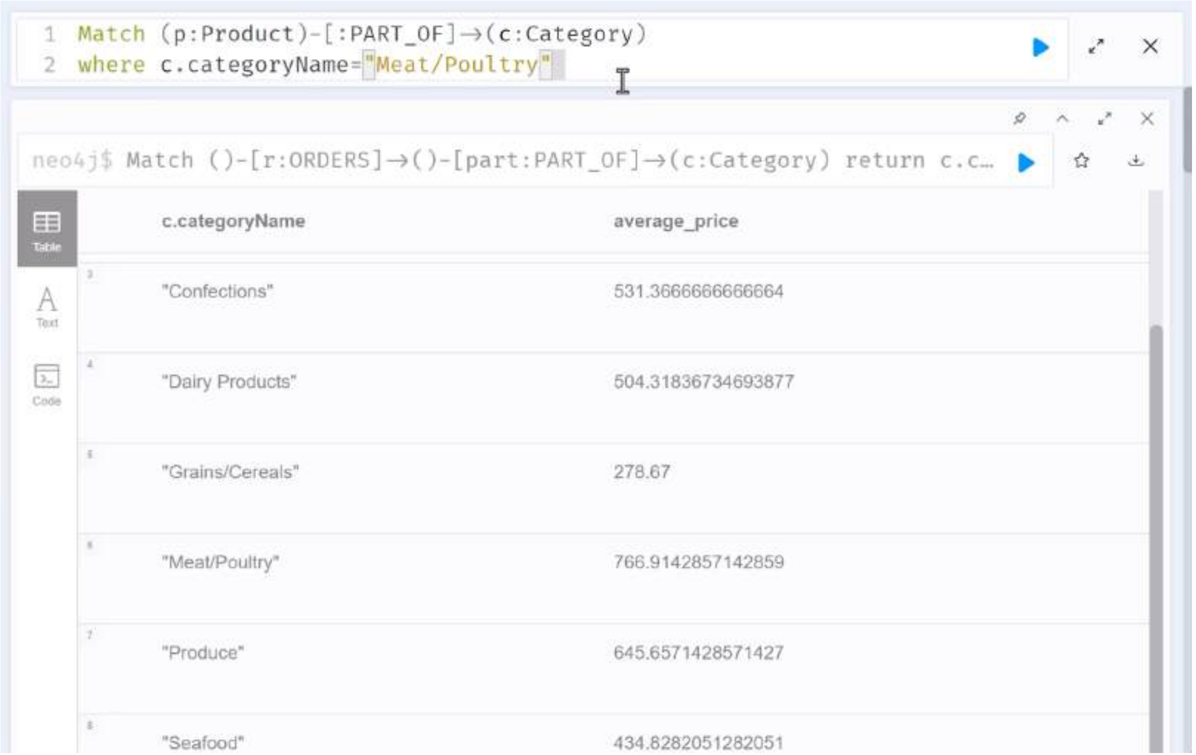
MATCH Statement

`MATCH` *`pattern`* `RETURN` *`result`*

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



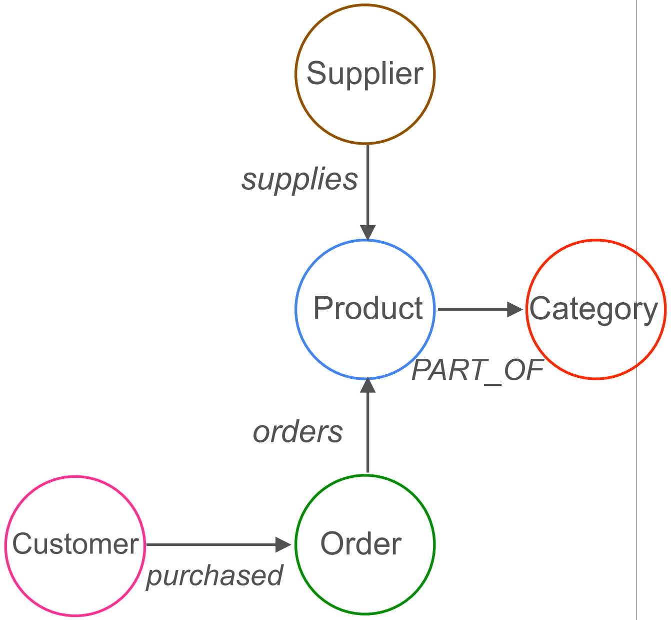
Retrieve the product name and product unit price of all products that belong to category "Meat/Poultry"



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”

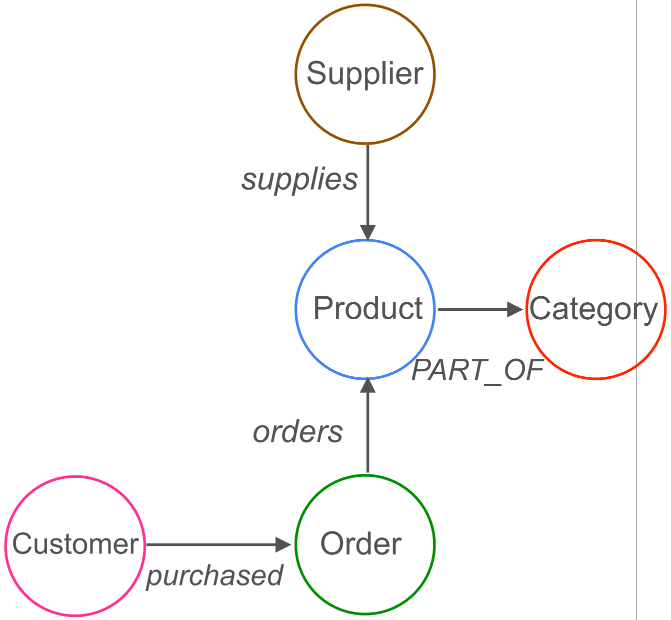

```
Match (p:Product)-[:PART_OF]→(c:Category)
where c.categoryName="Meat/Poultry"
return p.productName, p.unitPrice
```

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

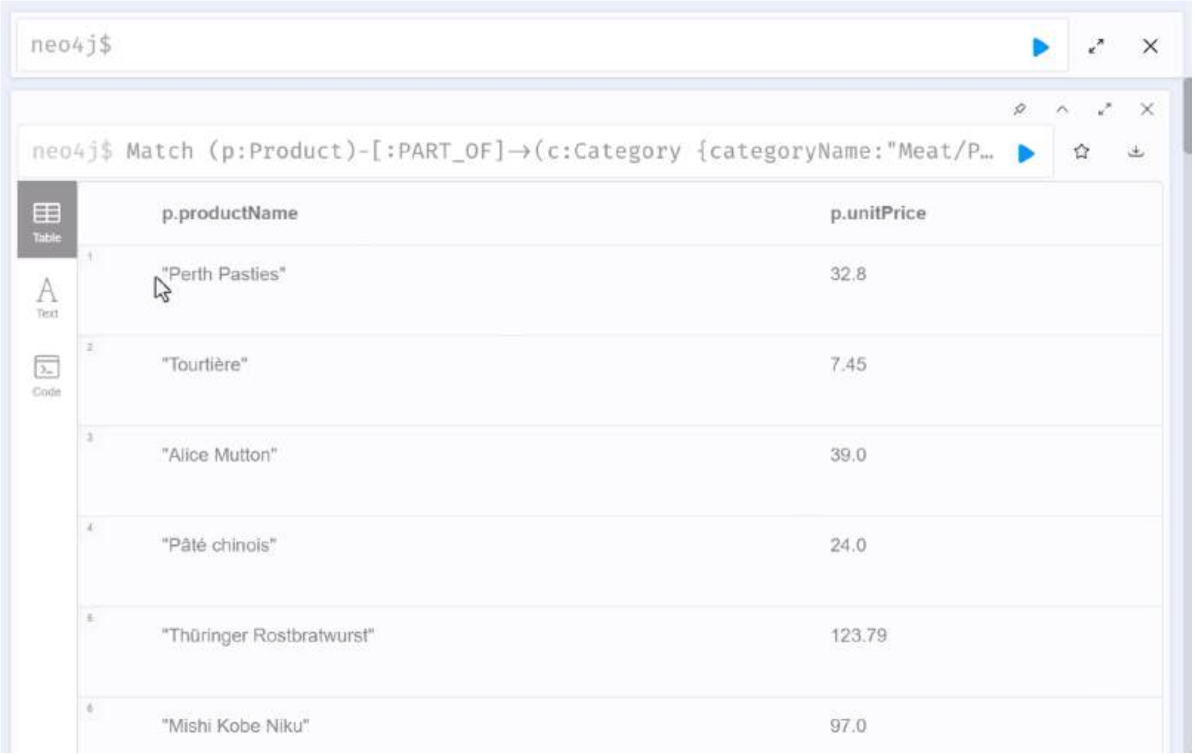
MATCH Statement

`MATCH` *`pattern`* `RETURN` *`result`*

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



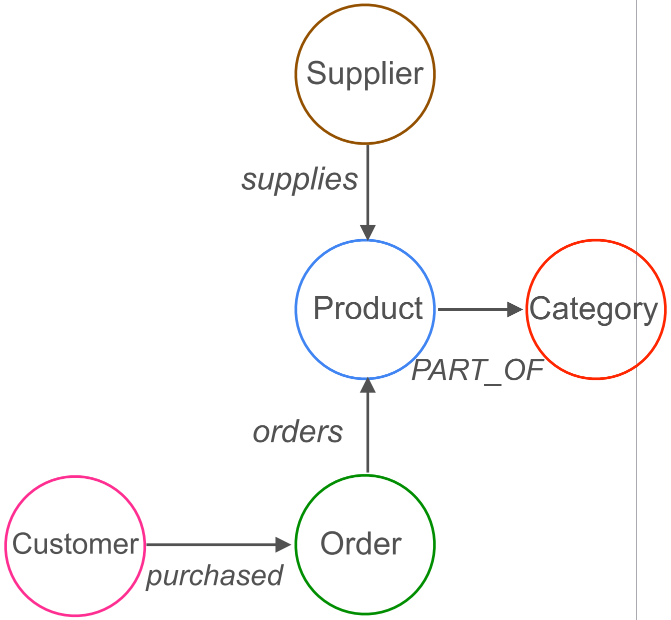
Retrieve the product name of all products ordered by the customer “QUEDE”



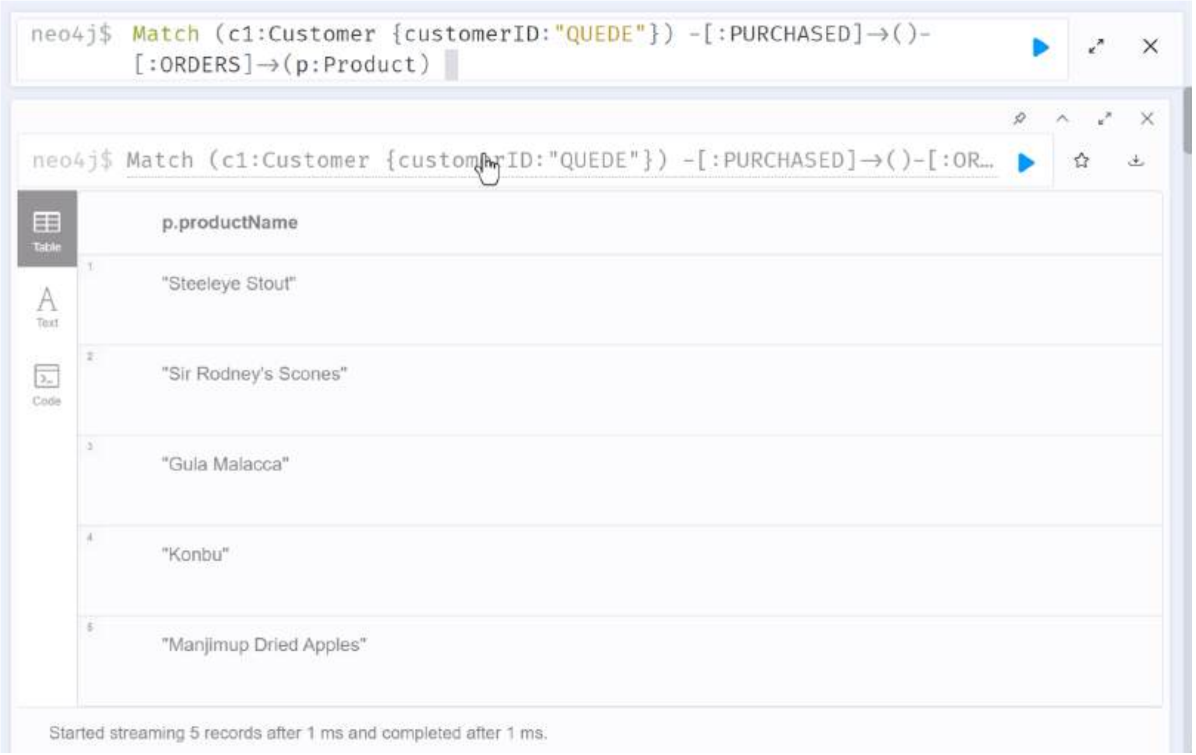
MATCH Statement

MATCH *pattern* RETURN *result*

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



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

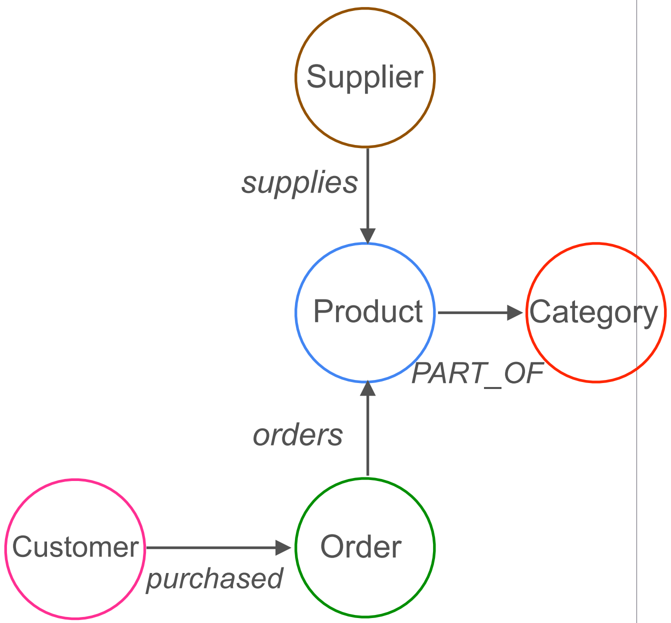


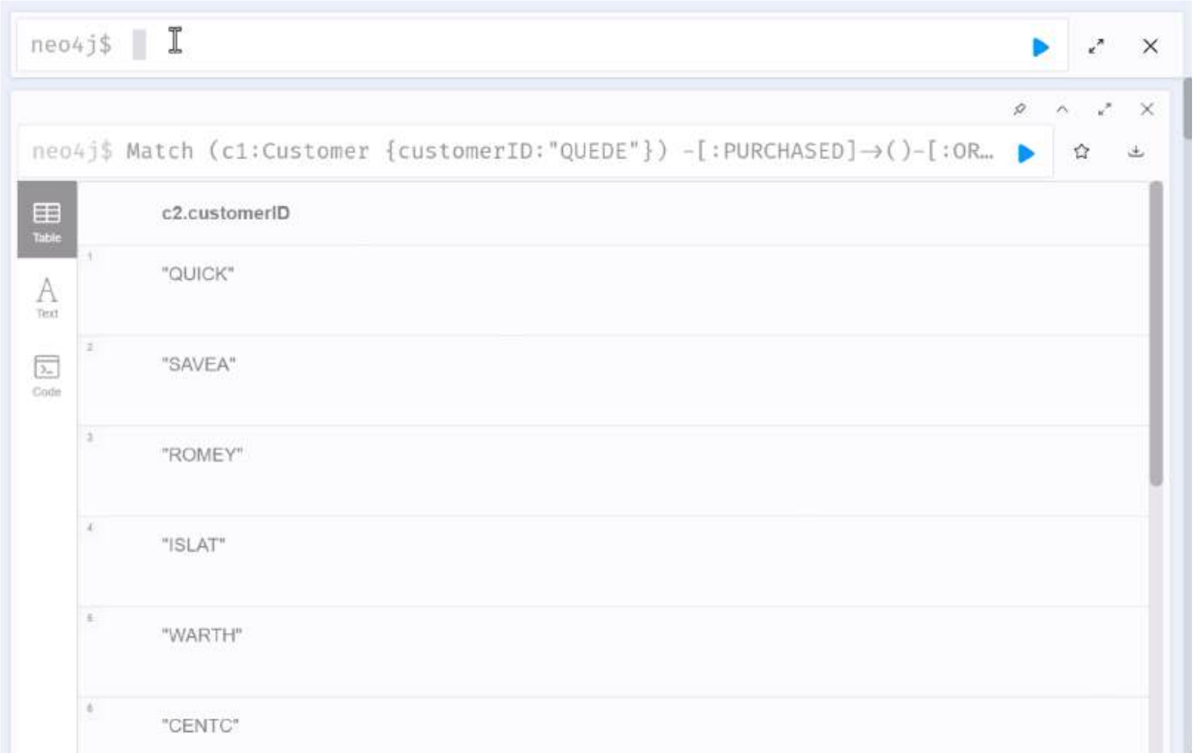
MATCH Statement

Retrieve the orders that contain at most two products

`MATCH` *`pattern`* `RETURN` *`result`*

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

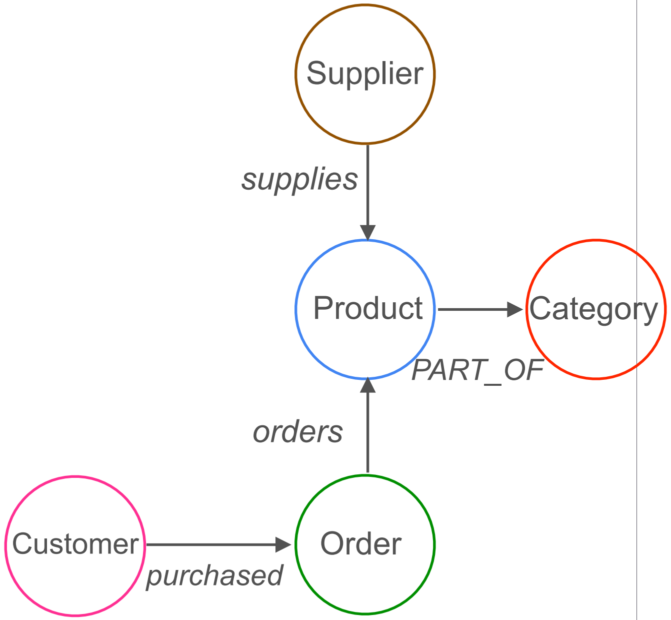




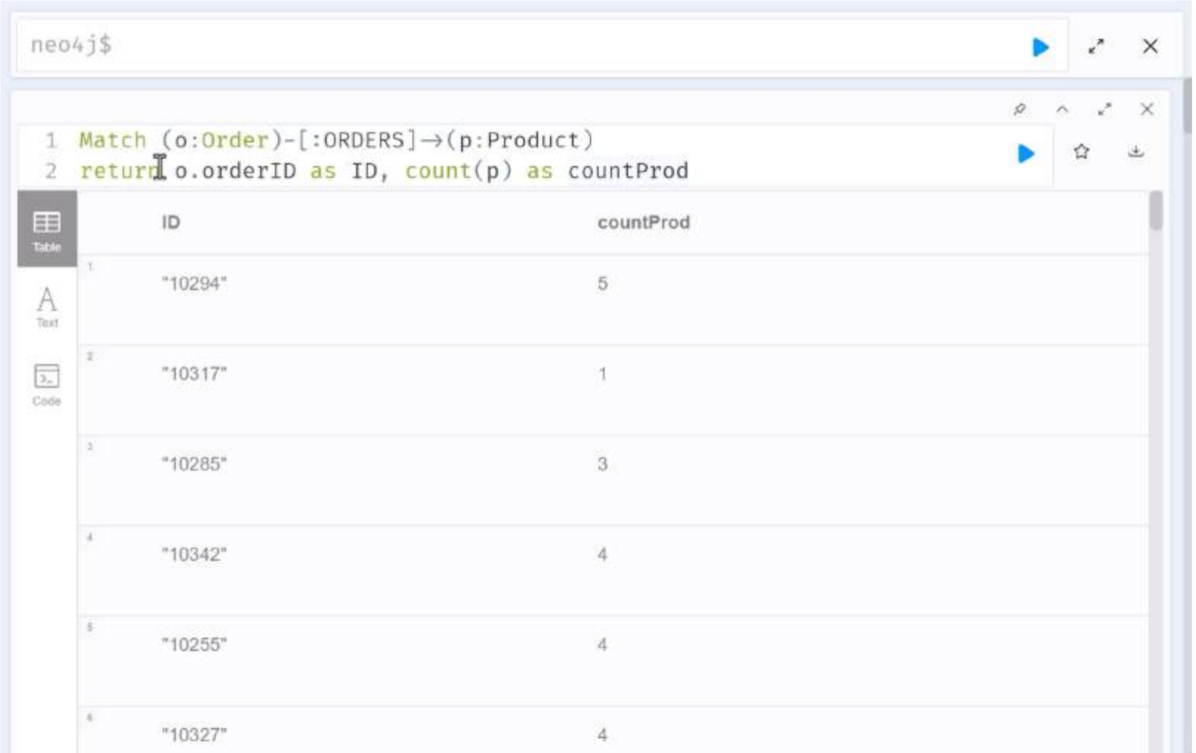
MATCH Statement

MATCH *pattern* RETURN *result*

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



Retrieve the orders that contain at most two products



Storage Systems

Summary

Raw Storage Ingredients

Persistent Storage Medium

Magnetic disk

Solid-state storage

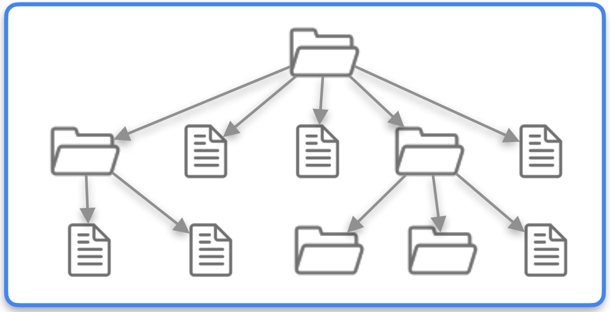
Volatile Memory

RAM

CPU cache

Cloud Storage Options

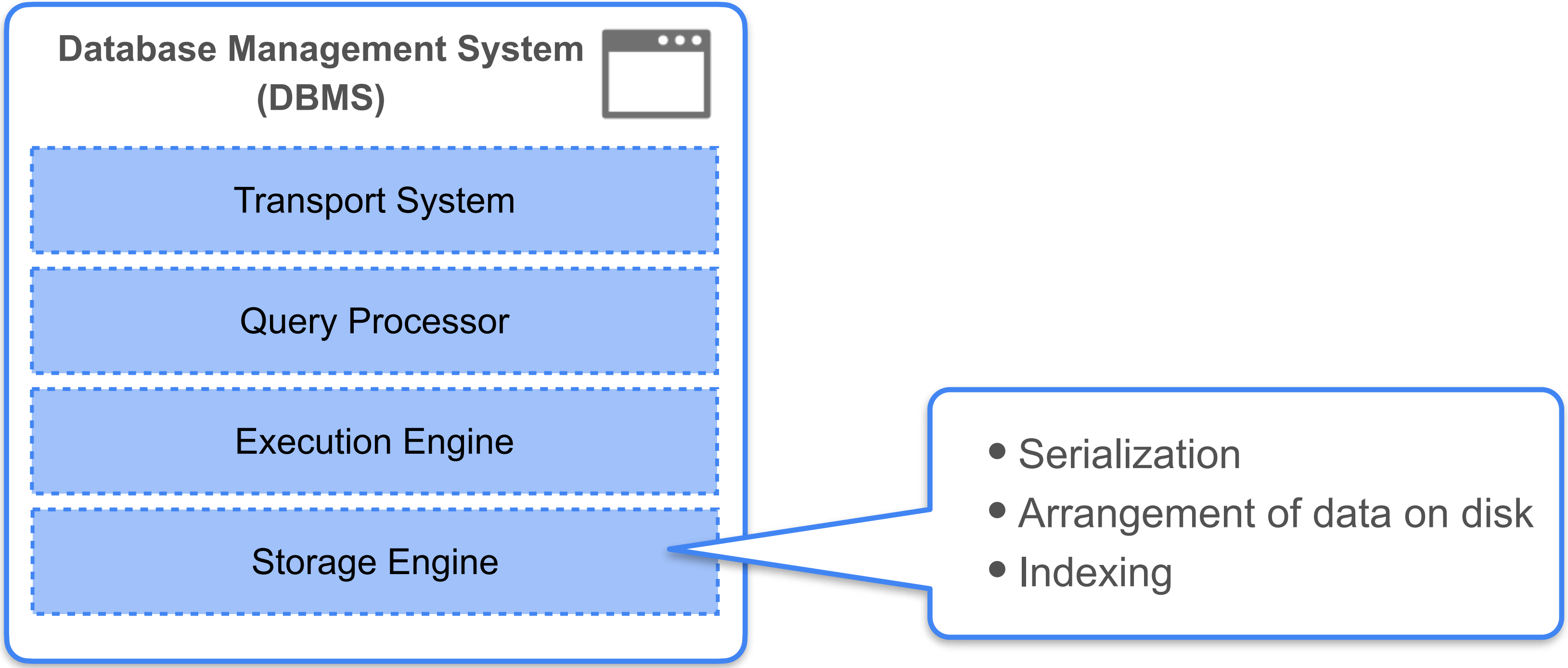
File Storage



Block Storage

Object Storage

Storage in Databases



Row and Columnar Storage

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

Row-oriented storage

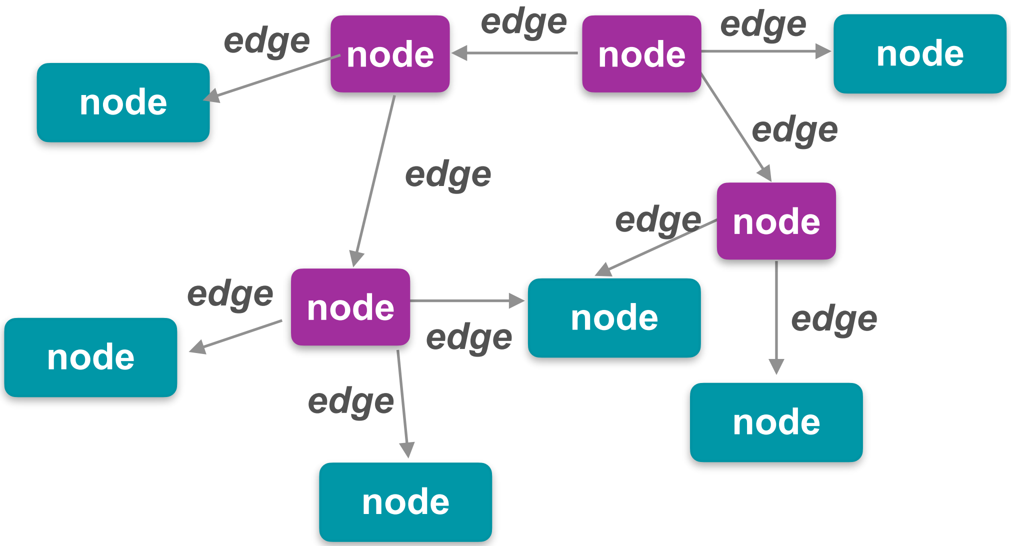
bytes representing the 1st row	bytes representing the 2nd row	…	bytes representing the last row

Column-oriented storage

bytes representing the 1st column	bytes representing the 2nd column	…	bytes representing the last column

Databases

Graph Databases



Cypher

Vector Databases