Big Data

&

NoSQL DataBase

Big Data & NoSQL DataBase

- NoSQL (Not Only SQL)
- Refers to databases that do not follow the traditional relational model based on SQL.
- They are designed to handle large volumes of unstructured or semi-structured data.

Big Data & NoSQL DataBase

With the growth of the internet and the explosion of data, traditional databases showed limitations in managing massive volumes of data, leading to the rise of NoSQL databases.

Types of NoSQL Databases

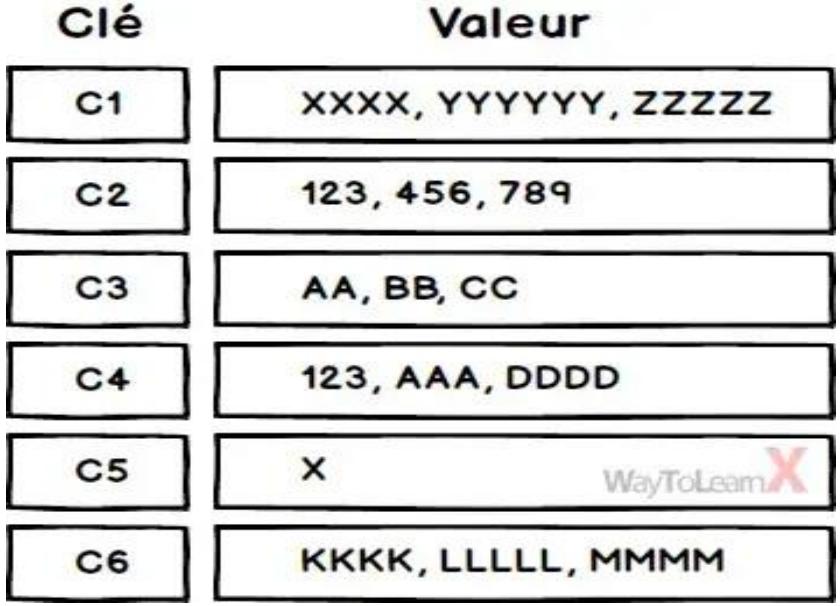
1. **Document-Oriented DataBase**

   JSON ou XML documents

   - Exemples : MongoDB, Couchbase.

2. **Key-Value DataBase**

   - Exemples : Redis, DynamoDB.



Types of NoSQL Databases

3. Column-Oriented Databases

- Store data in columns rather than rows,
- Examples : HBase, Apache Cassandra.

4. Graph Databases

Stores and queries relationships between data in the form of graphs.

- Examples : Neo4j, Amazon Neptune

Document-Oriented Databases: JSON Format

- JSON (JavaScript Object Notation)
- is a lightweight, human-readable data format
- used for exchanging data between a server and a client.

Document-Oriented Databases: JSON Format


```
{
    "nom": "John Doe",
    "age": 30,
    "estEtudiant": false,
    "adresse": {
        "rue": "123 rue Principale",
        "ville": "Ville",
        "codePostal": "12345"
    },
    "numerosDeTelephone": [
        "555-1234",
        "555-5678"
    ]
}
```

Document-Oriented Databases: JSON Format

JavaScript :

```
// Sérialisation
var objet = { nom: "John", age: 30 };
var jsonTexte = JSON.stringify(objet);

// Désérialisation
var objetDeserialise = JSON.parse(jsonTexte);
```

Document-Oriented Databases: JSON Format

In Python :

```
import json

# Sérialisation
objet = {"nom": "John", "age": 30}
json_texte = json.dumps(objet)

# Désérialisation
objet_deserialise = json.loads(json_texte)
```

Document-Oriented Databases

MongoDB Overview

- MongoDB is designed for speed using BSON (Binary JSON) documents.

- It encourages denormalization: instead of foreign keys across tables like SQL, data is duplicated where needed

Table *Cinema*

ID_film	Nom_du_film	Annee_de_sortie	Note_IMDB
1	Jaws	1975	8.0
2	Star Wars	1977	8.6
3	Don't look up	2021	7.2
4	Titanic	1997	7.8
5	Le dernier Samouraï	2003	7.7

Table *Acteurs*

Nom_Acteur	Date_de_naissance	ID_film
Roy Scheider	10/11/1932	1
Mark Hamill	25/09/1951	2
Meryl Streep	22/09/1949	3
Tom Cruise	03/07/1962	5
Tom Hanks	15/09/1977	NULL
Timothée Chamalet	27/12/1995	3

La base de Données « MongoDB »

MongoDB is built for distributed cloud environments with features like :

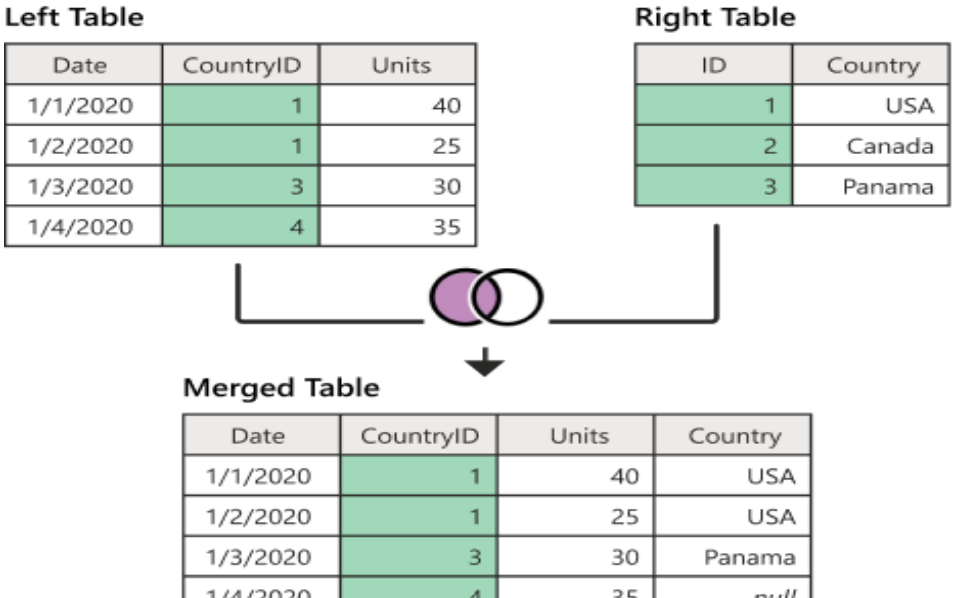
- data replication,
- high availability,
- and horizontal scalability.
- It is schemaless by design, unlike SQL databases.

- **MongoDB: Schemaless Flexibility**
- Schemaless can be a drawback in analytics, but schema rules can be imposed on collections.

MongoDB allows easier schema changes compared to large SQL systems.

MongoDB Advantages: Performance

- 1. Read speed: Avoids slow SQL joins by embedding data.
- 2. Flexibility: Easy to add fields without modifying a global schema.
- 3. Aggregation: Transform and group data for analysis.



Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	4	35

Right Table

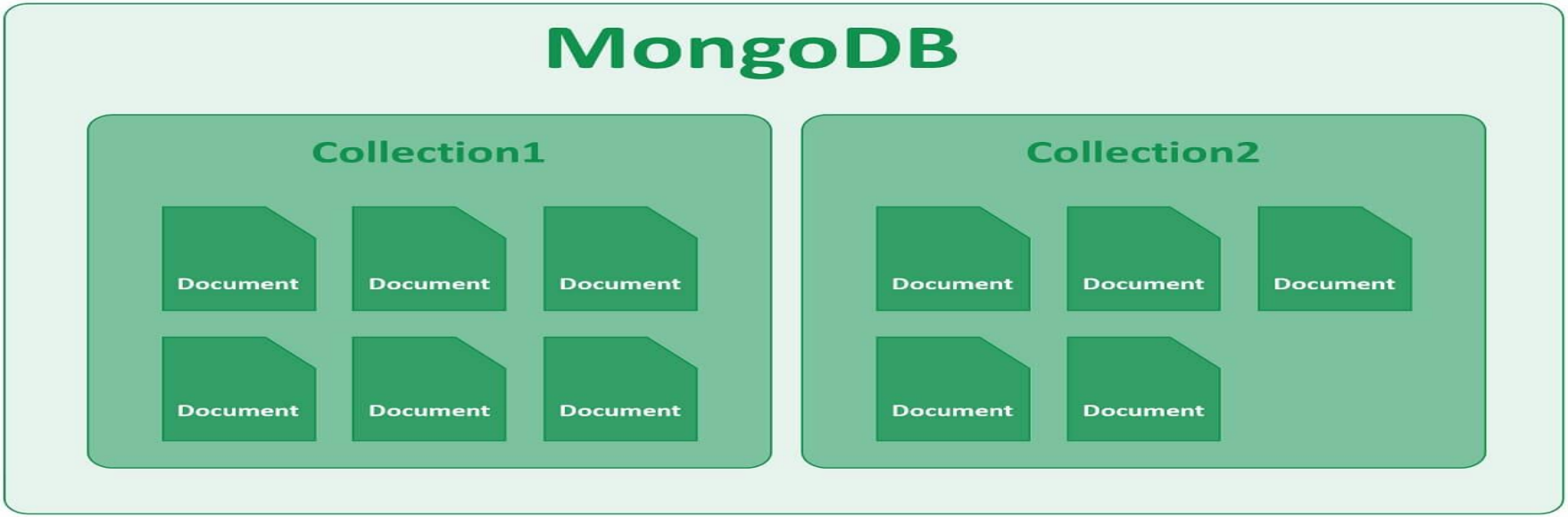
ID	Country
1	USA
2	Canada
3	Panama

Merged Table

Date	CountryID	Units	Country
1/1/2020	1	40	USA
1/2/2020	1	25	USA
1/3/2020	3	30	Panama
1/4/2020	4	35	null

MongoDB Advantages: Text & Geo Search

- 4. Text search: Indexes support keyword search across documents (e.g. blog posts).
- 5. GPS data: Geospatial indexes for location-based queries.



Practice: Blog Site Example

- Develop a blog site with user comments.
- Use both SQL and MongoDB:
- - SQL: 3 tables (user, article, comment)
- - MongoDB: 3 collections
- Compare text search performance.

Key-Value Stores

- least complex.➔ attractive
- It uses very simple functions to store (PUT), obtain (GET), and delete data.

2012-01-01	09:00	San Jose	Men's Clothing	214.05	Amex
2012-01-01	09:00	Fort Worth	Women's Clothing	153.57	Visa
2012-01-01	09:00	San Diego	Music	66.08	Cash
2012-01-01	09:00	Pittsburgh	Pet Supplies	493.51	Discover
2012-01-01	09:00	Omaha	Children's Clothing	235.63	MasterCa
2012-01-01	09:00	Stockton	Men's Clothing	247.18	MasterCa
2012-01-01	09:00	Austin	Cameras	379.6	Visa
2012-01-01	09:00	New York	Consumer Electronics	296.8	Cash
2012-01-02	15:20	Lincoln	Cameras	242.2	Discover
2012-01-02	15:20	Madison	Baby	254.15	MasterCa
2012-01-02	15:20	Wichita	Cameras	446.66	Amex
2012-01-02	15:20	Irvine	Computers	9.23	Discover
2012-01-02	15:20	Anaheim	Cameras	3.64	Visa
2012-01-02	15:21	Birmingham	Music	156.68	Cash

Key-Value Stores

Use Case: Web Session

- Sessions can include information about:
  - ❑ user
  - ❑ profile,
  - ❑ recommendations,
  - ❑ targeted promotions,
  - ❑ And discounts.
- Each user session has a unique identifier.
- Session data is only queried by a primary key.
- Fast key-value sessions are ideal in this context.

Key-Value Stores

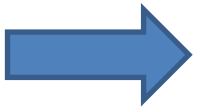
- Calculate total sales per store for the current year

2012-01-01	09:00	San Jose	Men's Clothing	214.05	Amex
2012-01-01	09:00	Fort Worth	Women's Clothing	153.57	Visa
2012-01-01	09:00	San Diego	Music	66.08	Cash
2012-01-01	09:00	Pittsburgh	Pet Supplies	493.51	Discover
2012-01-01	09:00	Omaha	Children's Clothing	235.63	MasterC
2012-01-01	09:00	Stockton	Men's Clothing	247.18	MasterC
2012-01-01	09:00	Austin	Cameras	379.6	Visa
2012-01-01	09:00	New York	Consumer Electronics	296.8	Cash
2012-01-02	15:20	Lincoln	Cameras	242.2	Discover
2012-01-02	15:20	Madison	Baby	254.15	MasterC
2012-01-02	15:20	Wichita	Cameras	446.66	Amex
2012-01-02	15:20	Irvine	Computers	9.23	Discover
2012-01-02	15:20	Anaheim	Cameras	3.64	Visa
2012-01-02	15:21	Birmingham	Music	156.68	Cash

Key-Value Stores

- In traditional programming, we would create hash tables in the form <key-value>.
- For each entry, enter the city and the sale price. If we find an entry with a city already entered, we group them together by summing the sales.

San Jose	214.05
Fort Worth	153.57
San Diego	66.08
New York	55.60
San Jose	100.00



San Jose	31400.05
Fort Worth	15300.57
San Diego	66000.08
New York	55000.60

Key-Value Stores

- Millions of rows of records
- Memory size issues
- Sequential processing

➔Problem?

San Jose	214.05
Fort Worth	153.57
San Diego	66.08
New York	55.60
San Jose	100.00

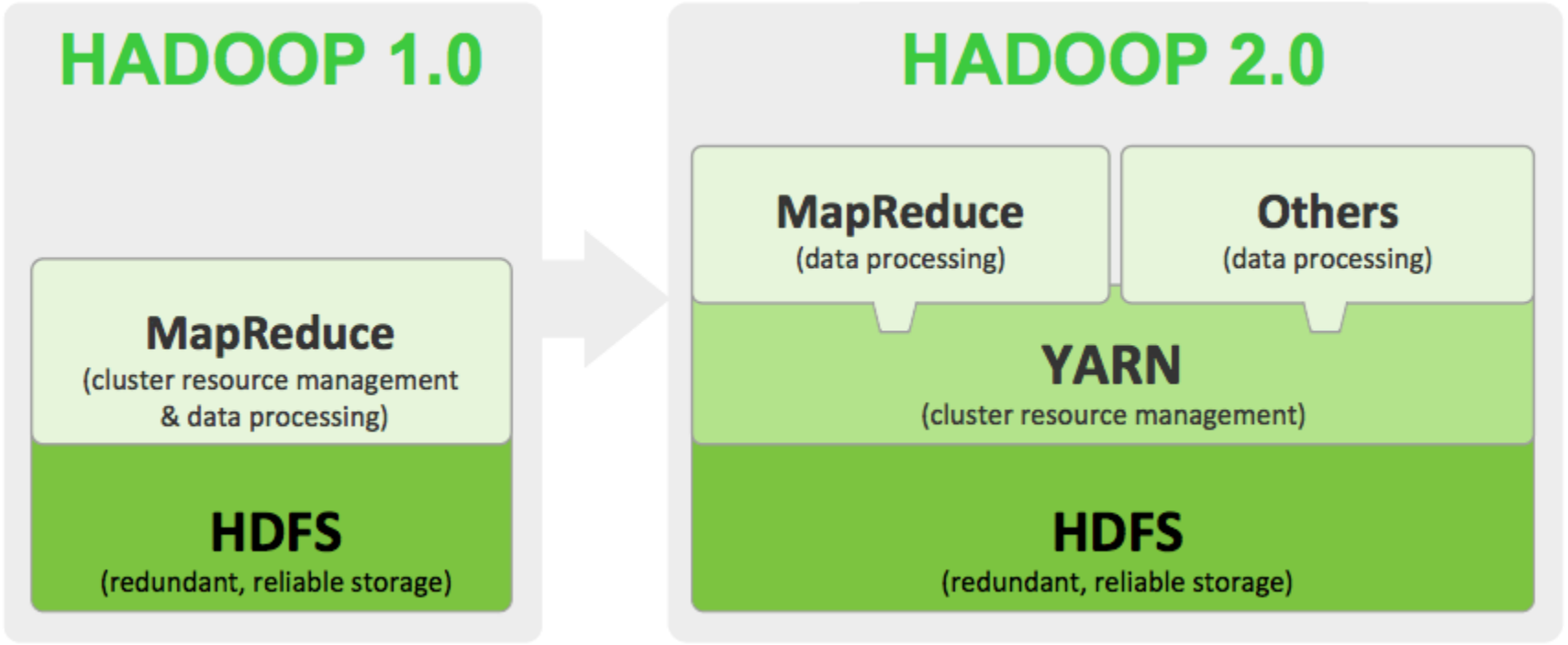
San Jose	31400.05
Fort Worth	15300.57
San Diego	66000.08
New York	55000.60

➔Solution : Map-Reduce of Hadoop

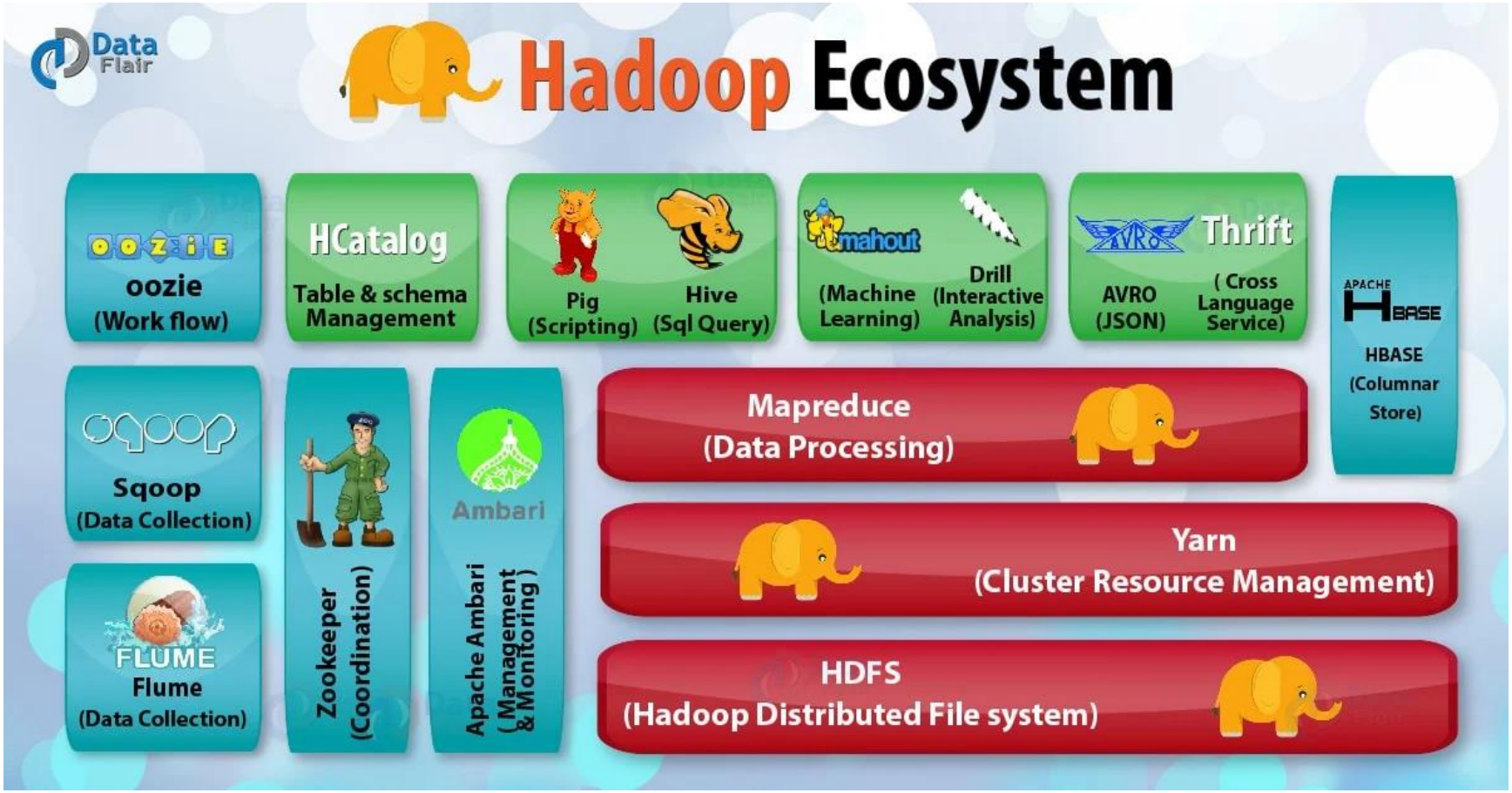
Key-Value Stores

- Hadoop Map-ReduceA programming technique with two main functions:
- Map: transforms the input into KEY/VALUE pairs
- REDUCE: aggregates the values for each key
- OPEN SOURCE (Apache Foundation) written in JAVA
- Inspired by Google publications (2004):
  1. Google Map Reduce
  2. Google Filesystem

Key-Value Stores

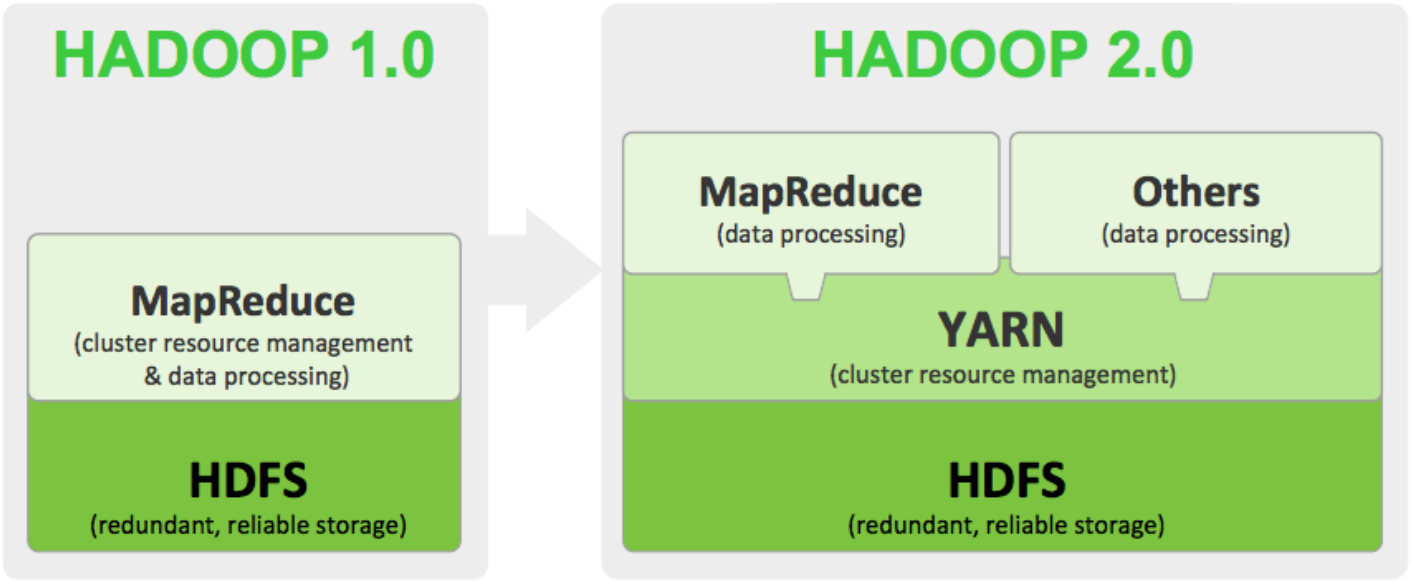


Key-Value Stores



Key-Value Stores

- **MapReduce is a programming model available in Hadoop environments**
- **Used to access big data stored in the Hadoop File System (HDFS)**



Key-Value Stores

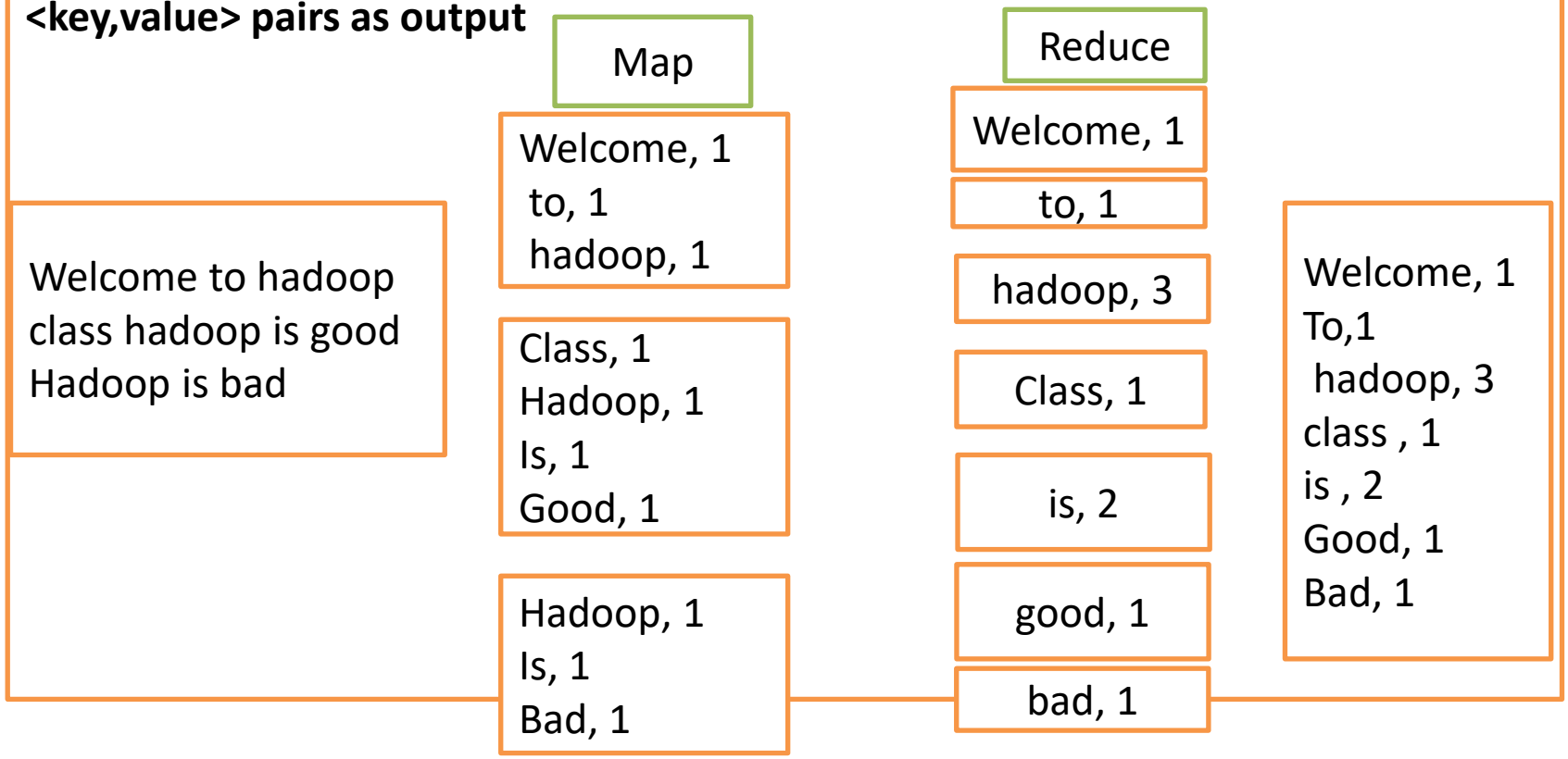
- MapReduce aggregates data from multiple servers and returns a result to the application.
- With MapReduce, rather than sending the data to the application or algorithms,
- The algorithms are executed on the server where the data already resides, which speeds up processing.

Key-Value Stores

- A Hadoop cluster of 20,000 servers (standard and inexpensive servers) with 256 MB data blocks can process approximately 5 TB of data.
- With MapReduce, you can therefore reduce processing time compared to sequential processing of such a large dataset.
- Google's cluster contains 10,000,000 servers.

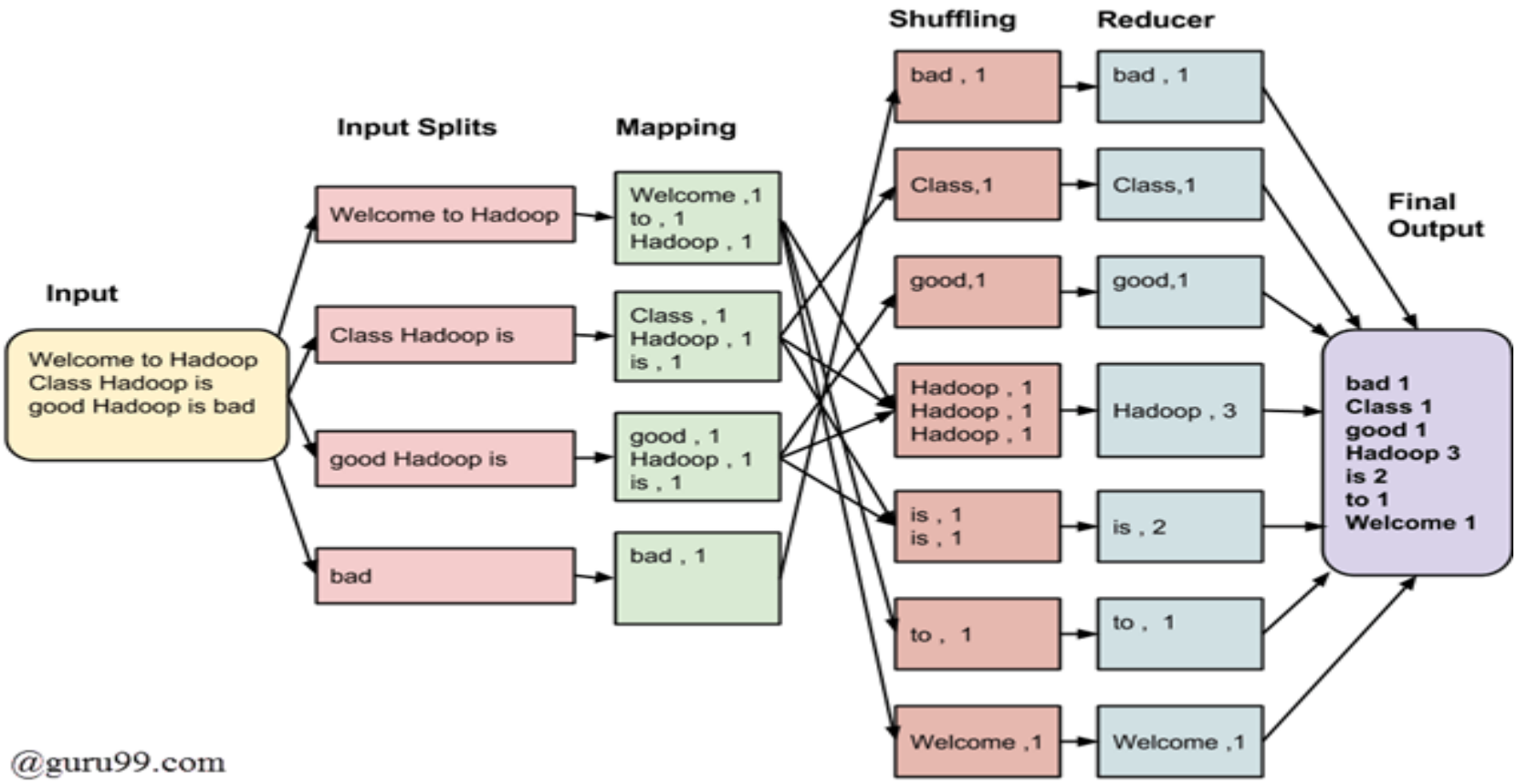
Key-Value Stores

MapReduce – (ex: WORD COUNT)
Map transforms disk entries into <key,value> pairs
Reduce transforms the inputs into <key,value> pairs and generates one of the <key,value> pairs as output



Key-Value Stores

MapReduce :**WORD COUNT**



Key-Value Stores

- **MapReduce :WORD COUNT**
- **programme: MAP**

public class WordCountMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

private final IntWritable one = new IntWritable(1); private Text word = new Text();

public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

String line = value.toString();

StringTokenizer itr = new StringTokenizer(line.toLowerCase());

while(itr.hasMoreTokens())

{ word.set(itr.nextToken()); output.collect(word, **one**); } } }

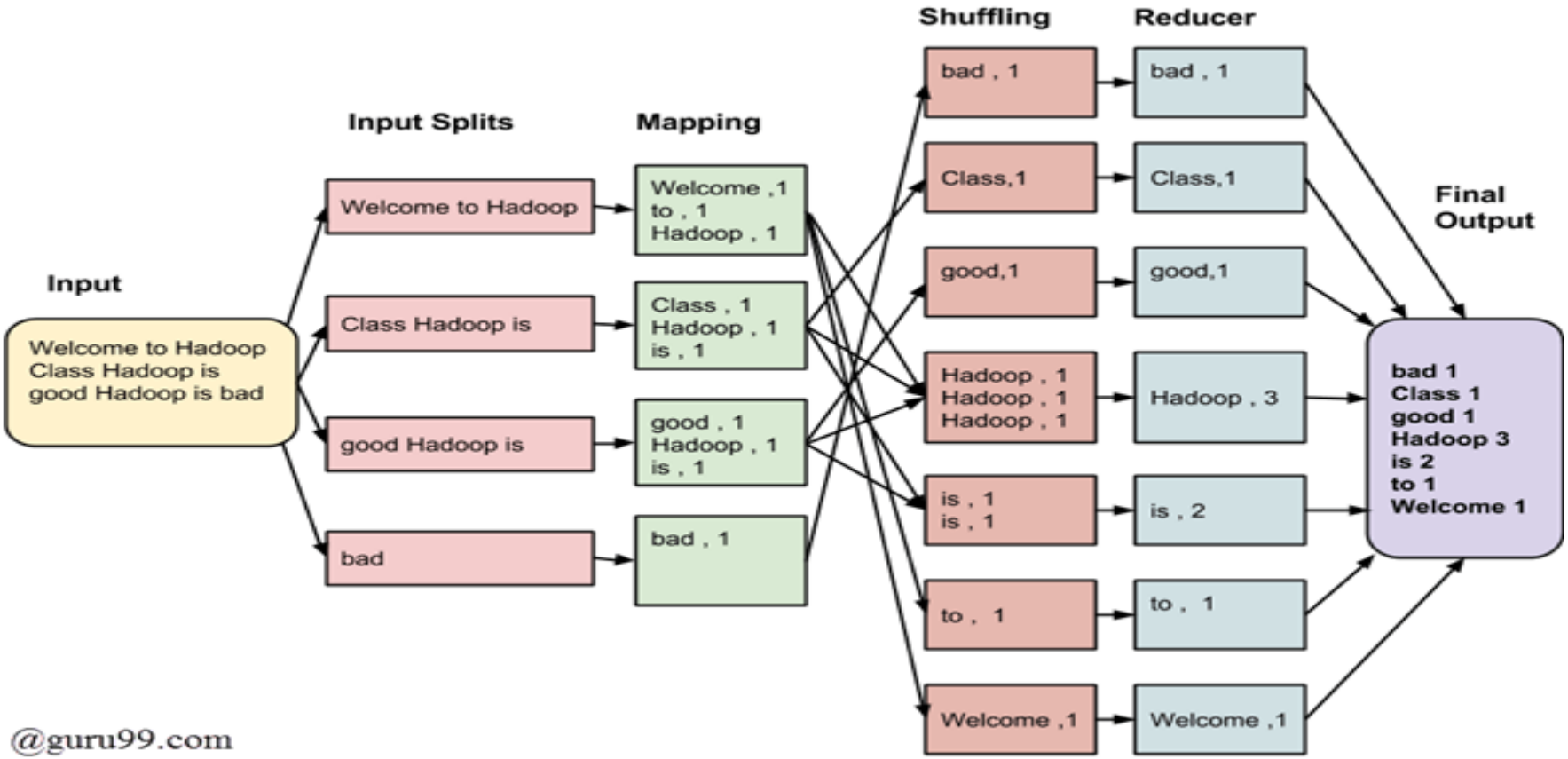
Key-Value Stores

REDUCER Code

```
public class WordCountReducer extends MapReduceBase implements Reducer<Text,
IntWritable, Text, IntWritable> {

   public void reduce(Text key, Iterator<IntWritable> values,  OutputCollector
<Text,  IntWritable> output, Reporter reporter) throws IOException {

 int sum = 0;
      while (values.hasNext()) {
         // replace ValueType with the real type of your value
         IntWritable value = (IntWritable) values.next();
         sum += value.get(); // process value
      }
      output.collect(key, new IntWritable(sum));
   }}
```

Key-Value Stores

```
import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
```


Input
Welcome to Hadoop
Class Hadoop is
good Hadoop is bad
Input Splits
Welcome to Hadoop
Class Hadoop is
good Hadoop is
bad
Mapping
Welcome ,1
to , 1
Hadoop , 1
Class , 1
Hadoop , 1
is , 1
good , 1
Hadoop , 1
is , 1
bad , 1
Shuffling
bad , 1
Class,1
good,1
Hadoop , 1
Hadoop , 1
Hadoop , 1
is , 1
is , 1
to , 1
Welcome ,1
Reducer
bad , 1
Class,1
good,1
Hadoop , 3
is , 2
to , 1
Welcome ,1
Final Output
bad 1
Class 1
good 1
Hadoop 3
is 2
to 1
Welcome 1

Calculer le total des ventes par magasin pour l'année en cours ?

2012-01-01	09:00	San Jose	Men's Clothing	214.05	Amex
2012-01-01	09:00	Fort Worth	Women's Clothing	153.57	Visa
2012-01-01	09:00	San Diego	Music	66.08	Cash
2012-01-01	09:00	Pittsburgh	Pet Supplies	493.51	Discover
2012-01-01	09:00	Omaha	Children's Clothing	235.63	MasterCard
2012-01-01	09:00	Stockton	Men's Clothing	247.18	MasterCard
2012-01-01	09:00	Austin	Cameras	379.6	Visa
2012-01-01	09:00	New York	Electronics	296.8	Cash
2012-01-02	15:20	Lincoln	Cameras	242.2	Discover
2012-01-02	15:20	Madison	Baby	254.15	MasterCard
2012-01-02	15:20	Wichita	Cameras	446.66	Amex
2012-01-02	15:20	Irvine	Computers	9.23	Discover
2012-01-02	15:20	Anaheim	Cameras	3.64	Visa
2012-01-02	15:21	Birmingham	Music	156.68	Cash

Distributed storage

- Hadoop Distributed File System (HDFS),
- Amazon S3,
- Google Cloud Storage

Hadoop Distributed File System (HDFS)

- is a distributed file system that manages large datasets.
- HDFS is one of the core components of Apache Hadoop (along with MapReduce and YARN).
- Running on commodity hardware

objectives of HDFS

1. Fast recovery from hardware failures

- HDFS can include thousands of servers, so the failure of at least one server is inevitable
- HDFS was designed to detect faults and recover automatically quickly

objectives of HDFS

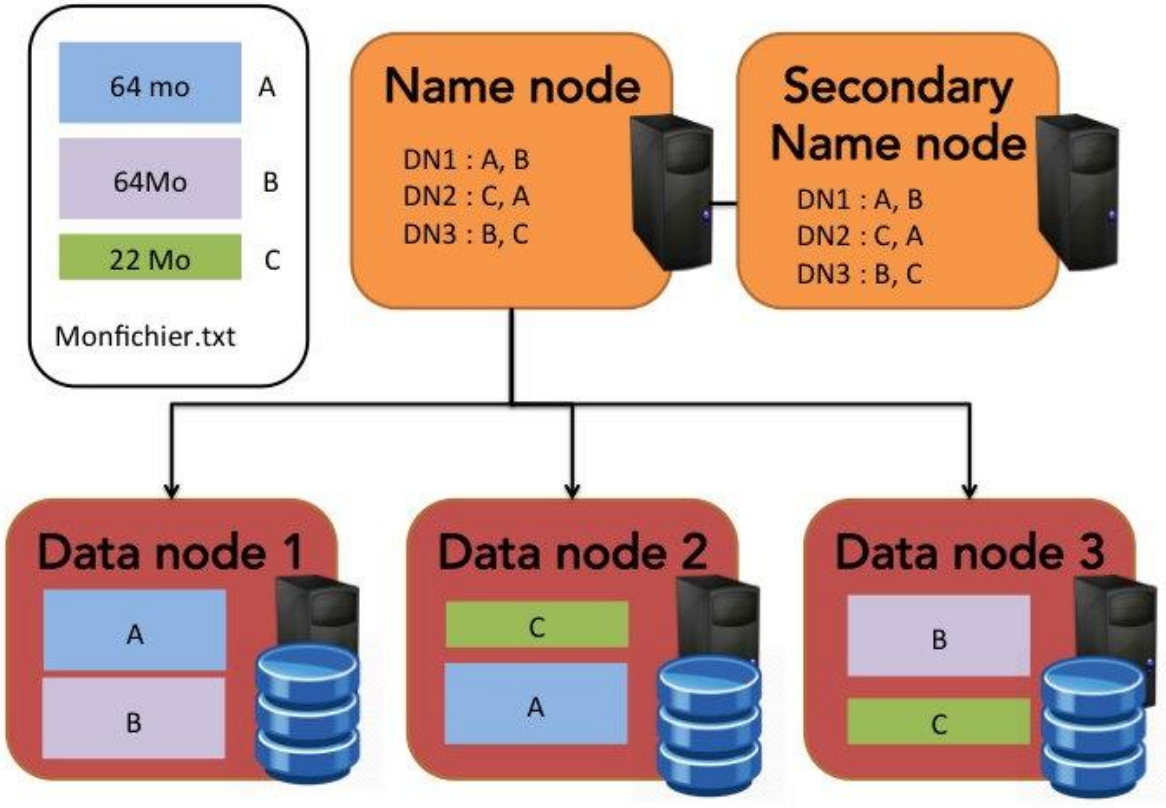
2. Access to streaming data
3. Hosting of large data sets.
4. Portability

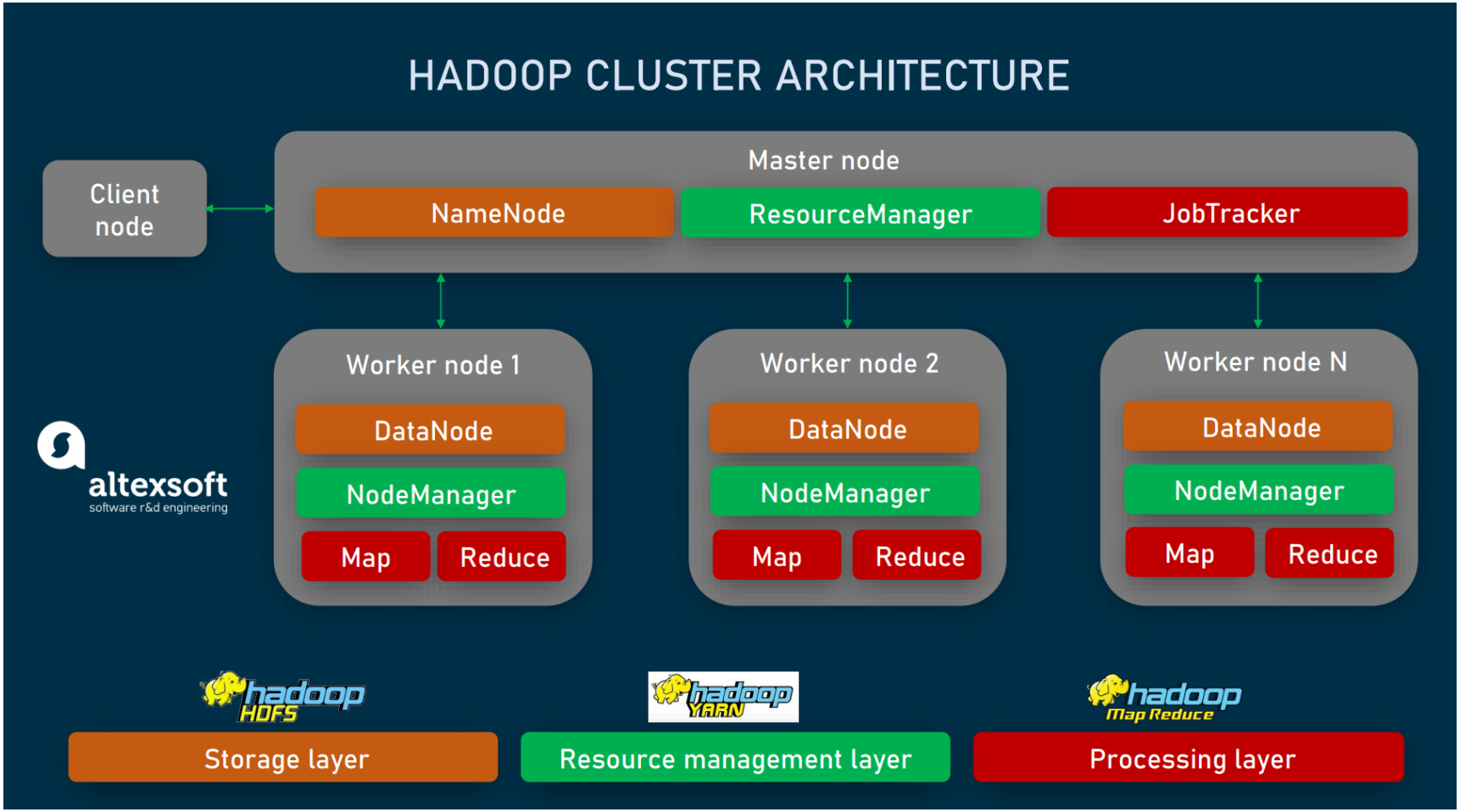
HDFS Architecture

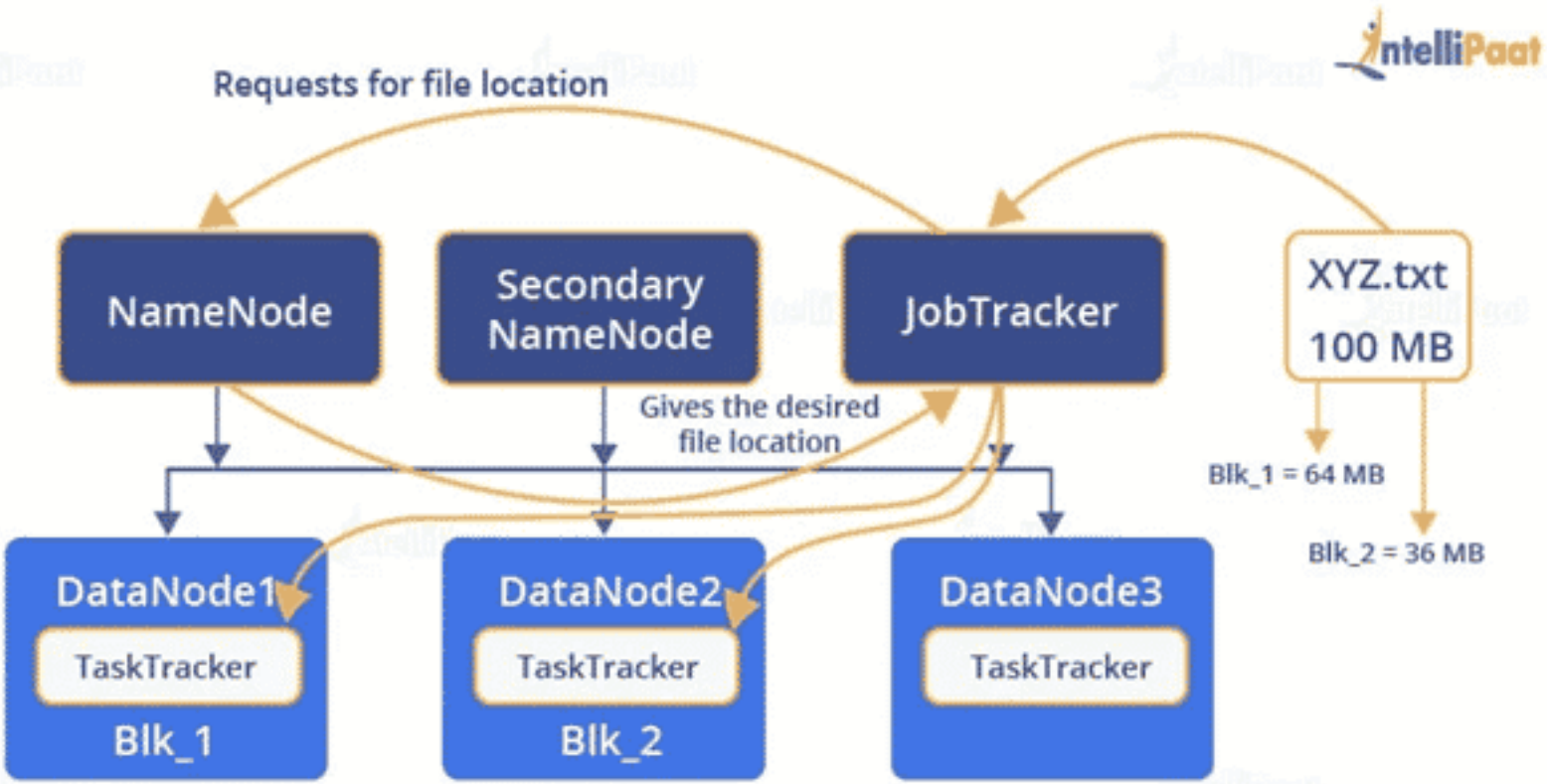
- A cluster of multiple machines
- Master/slave technique
- Data is stored on datanodes (slaves)
- Metadata on data blocks is managed by the namenode (master)

HDFS Architecture

- Each data file is broken down into blocks.
- 64 MB by default.
- With replication principle.




HADOOP CLUSTER ARCHITECTURE
Client node
Master node
NameNode
ResourceManager
JobTracker
Worker node 1
DataNode
NodeManager
Map
Reduce
Worker node 2
DataNode
NodeManager
Map
Reduce
Worker node N
DataNode
NodeManager
Map
Reduce
altexsoft
software r&d engineering
hadoop HDFS
Storage layer
hadoop YARN
Resource management layer
hadoop Map Reduce
Processing layer


IntelliPaat
Requests for file location
NameNode
Secondary NameNode
JobTracker
XYZ.txt
100 MB
Gives the desired file location
Blk_1 = 64 MB
Blk_2 = 36 MB
DataNode1
TaskTracker
Blk_1
DataNode2
TaskTracker
Blk_2
DataNode3
TaskTracker

Column-oriented database

- Are designed to optimize queries that analyze data across multiple rows.
- Ex: Calculate the total sales for the San Diego store? Then for the Cameras category?

2012-01-01	09:00	San Jose	Men's Clothing	214.05	Amex
2012-01-01	09:00	Fort Worth	Women's Clothing	153.57	Visa
2012-01-01	09:00	**San Diego**	Music	66.08	Cash
2012-01-01	09:00	Pittsburgh	Pet Supplies	493.51	Discover
2012-01-01	09:00	Omaha	Children's Clothing	235.63	MasterCard
2012-01-01	09:00	Stockton	Men's Clothing	247.18	MasterCard
2012-01-01	09:00	Austin	**Cameras**	379.6	Visa
2012-01-01	09:00	New York	Electronics	296.8	Cash
2012-01-02	15:20	Lincoln	**Cameras**	242.2	Discover
2012-01-02	15:20	**San Diego**	Baby	254.15	MasterCard
2012-01-02	15:20	Wichita	Cameras	446.66	Amex
2012-01-02	15:20	Irvine	Computers	9.23	Discover
2012-01-02	15:20	Anaheim	**Cameras**	3.64	Visa
2012-01-02	15:21	**San Diego**	Music	156.68	Cash

Column-oriented database

- The most expensive operations involving hard drives are search functions.
- Data is stored as follows: bytes on the hard drive (or RAM).
- A byte can contain one line (or several lines).

2012-01-01	09:00	San Jose	Men's Clothing	214.05	Amex
2012-01-01	09:00	Fort Worth	Women's Clothing	153.57	Visa
2012-01-01	09:00	**San Diego**	Music	66.08	Cash
2012-01-01	09:00	Pittsburgh	Pet Supplies	493.51	Discover
2012-01-01	09:00	Omaha	Children's Clothing	235.63	MasterCard
2012-01-01	09:00	Stockton	Men's Clothing	247.18	MasterCard
2012-01-01	09:00	Austin	**Cameras**	379.6	Visa
2012-01-01	09:00	New York	Electronics	296.8	Cash
2012-01-02	15:20	Lincoln	**Cameras**	242.2	Discover
2012-01-02	15:20	**San Diego**	Baby	254.15	MasterCard
2012-01-02	15:20	Wichita	Cameras	446.66	Amex
2012-01-02	15:20	Irvine	Computers	9.23	Discover
2012-01-02	15:20	Anaheim	**Cameras**	3.64	Visa
2012-01-02	15:21	**San Diego**	Music	156.68	Cash

Column-oriented database

- The most expensive operations involving hard drives are search functions ➔ requires indexing
- Data is stored as follows: bytes on the hard drive (or RAM)
- A byte can contain one line (or several)

ID, Date, Time, State, Category, amount, card
AAA1, 2012-01-01, 09:00, **San Jose**, Music, 214.05, Visa
AAA2, 2012-01-01, 09:00, Fort Worth, Women's Clothing, 153.57, Visa
AAA3, 2012-01-01, 09:00, San Diego**,** Music, 66.08, Cash
AAA4 2012-01-01 09:00, **San Jose**, Pet Supplies, 493.51, Visa

AAA1: 0001, AAA2:0002, AAA3 :0003, AAA4:0004
2012-01-01: 0001, 2012-01-01: 0002, 2012-01-01: 0003, 2012-01-01: 0004
09:00 : 0001, 09:00 : 0002, 09:00 : 0003, 09:00 : 0004
San Jose: 0001, , Fort Worth :0002, San Diego :0003, **San Jose** :0004
Music : 0001, Women's Clothing :0002, Music :0003, Pet Supplies :0004
214.05 : 0001, 153.57 :0002, 66.08 :0003, 493.51 :0004
Visa : 0001, Visa :0002, Cash :0003, Visa :0004

Column-oriented database

- The most expensive operations involving hard drives are search functions ➔ requires indexing
- Data is stored as follows: bytes on the hard drive (or RAM)
- A byte can contain one line (or several)

AAA1: 0001, AAA2:0002, AAA3 :0003, AAA4:0004
2012-01-01: 0001, 2012-01-01: 0002, 2012-01-01: 0003, 2012-01-01: 0004
09:00 : 0001, 09:00 : 0002, 09:00 : 0003, 09:00 : 0004
San Jose: 0001, Fort Worth :0002, San Diego :0003, **San Jose** :0004
Music : 0001, Women's Clothing :0002, Music :0003, Pet Supplies :0004
214.05 : 0001, 153.57 :0002, 66.08 :0003, 493.51 :0004
Visa : 0001, Visa :0002, Cash :0003, Visa :0004

AAA1: 0001; AAA2:0002; AAA3 :0003; AAA4:0004;
2012-01-01: 0001, 0002, 0003, 0004;
09:00 : 0001, 0002, 0003, 0004;
San Jose: **0001, 0004** ; Fort Worth :0002; San Diego :0003;
Music : 0001, 0003 ; Women's Clothing :0002; Pet Supplies :0004;
214.05 : 0001; 153.57 :0002; 66.08 :0003; 493.51 :0004;
Visa : 0001, 0002, 0004; Cash :0003;

Column-oriented database

1. **Efficient storage**

stores column values together, enabling efficient compression and indexing of similar data, thus reducing disk space requirements**.**

2012-01-01	09:00	San Jose	Men's Clothing	214.05	Amex
2012-01-01	09:00	Fort Worth	Women's Clothing	153.57	Visa
2012-01-01	09:00	**San Diego**	Music	66.08	Cash
2012-01-01	09:00	Pittsburgh	Pet Supplies	493.51	Discover
2012-01-01	09:00	Omaha	Children's Clothing	235.63	MasterCard
2012-01-01	09:00	Stockton	Men's Clothing	247.18	MasterCard
2012-01-01	09:00	Austin	**Cameras**	379.6	Visa
2012-01-01	09:00	New York	Electronics	296.8	Cash
2012-01-02	15:20	Lincoln	**Cameras**	242.2	Discover
2012-01-02	15:20	**San Diego**	Baby	254.15	MasterCard
2012-01-02	15:20	Wichita	Cameras	446.66	Amex
2012-01-02	15:20	Irvine	Computers	9.23	Discover
2012-01-02	15:20	Anaheim	**Cameras**	3.64	Visa
2012-01-02	15:21	**San Diego**	Music	156.68	Cash

Column-oriented database

- **Improved performance for analytical queries**
  - **Data analysis**
  - **Aggregations**

Calculate the total sales for the San Jose store?

Calculate the total Music sales for the San Jose store?

AAA1: 0001; AAA2:0002; AAA3 :0003; AAA4:0004;
2012-01-01: 0001, 0002, 0003, 0004;
09:00 : 0001, 0002, 0003, 0004;
San Jose: **0001, 0004** ; Fort Worth :0002; San Diego :0003;
Music : 0001, 0003 ; Women's Clothing :0002; Pet Supplies :0004;
214.05 : 0001; 153.57 :0002; 66.08 :0003; 493.51 :0004;
Visa : 0001, 0002, 0004; Cash :0003;

Column-oriented database

- **Data compression**

AAA1: 0001; AAA2:0002; AAA3 :0003; AAA4:0004;
2012-01-01: 0001, 0002, 0003, 0004;
09:00 : 0001, 0002, 0003, 0004;
San Jose: 0001, 0004 ; Fort Worth :0002; San Diego :0003;
Music : 0001, 0003 ; Women's Clothing :0002; Pet Supplies :0004;
214.05 : 0001; 153.57 :0002; 66.08 :0003; 493.51 :0004;
Visa : 0001, 0002, 0004; Cash :0003;

- **Less suitable for online transactions**
  - **For frequent real-time read-write operations (such as DBMS)**

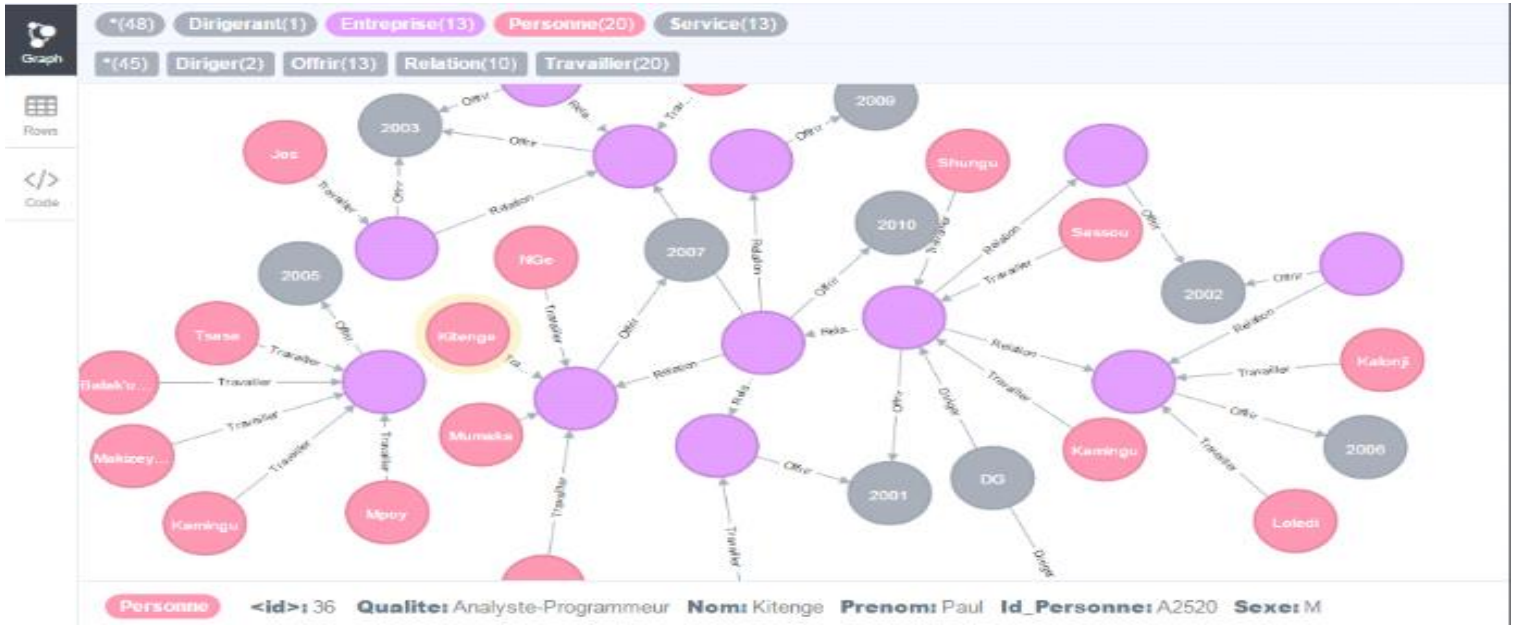
Column-oriented database

- Example of a column-oriented DB system:

1. **Google BigTable**
2. **Apache Hbase (used by Facebook)**
3. **Apache Cassandra (used by Facebook)**
4. **Amazon Redshift**
5. **IBM Informix**
6. **Oracle Database 12c**
7. **Microsoft SQL Server (columnstore)**

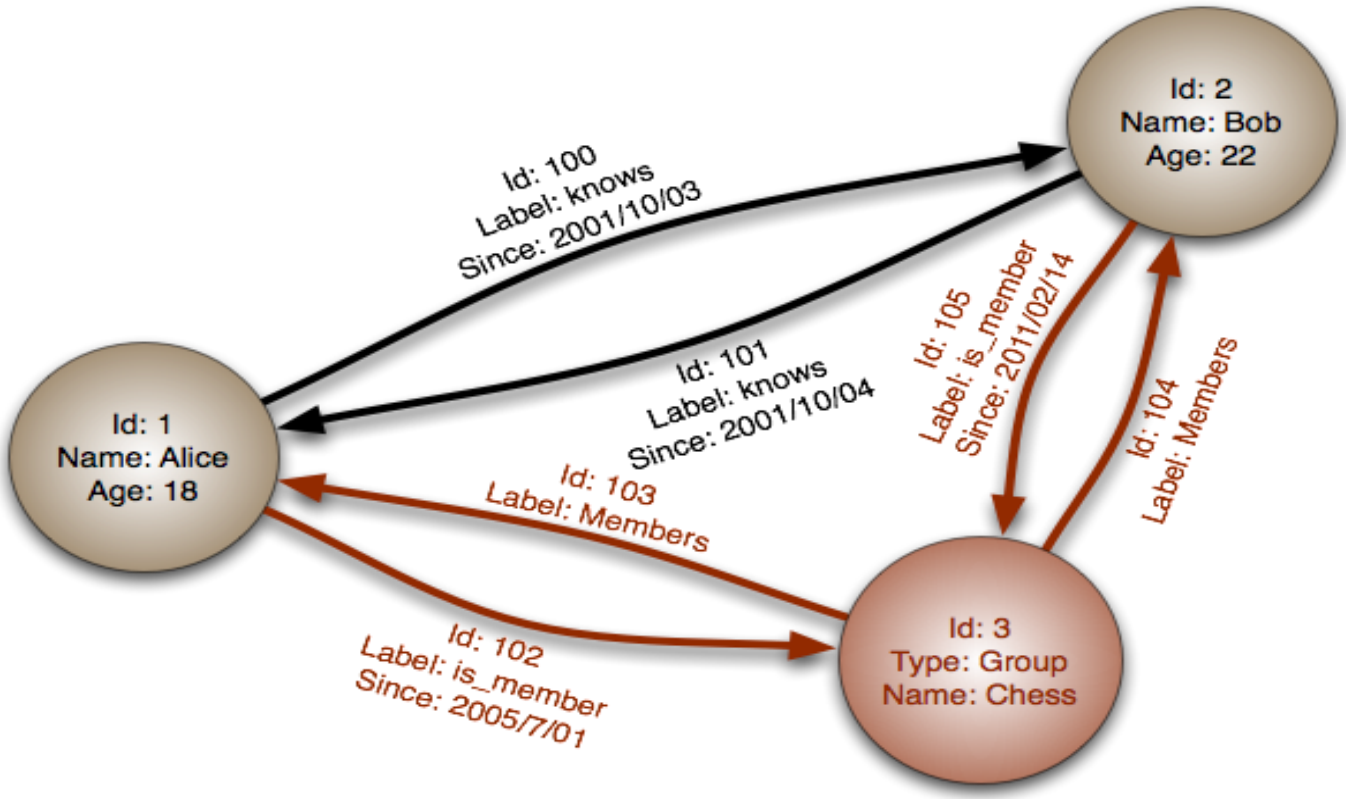
Graph-Oriented Database

optimized for Complex relationships between data



Graph-Oriented Database

- Optimized for path or neighborhood type queries



Graph-Oriented Database

Graph-Oriented Database

Use cases such as:

1. social networks,
2. recommendations,
3. fraud detection,
4. network management (IT, transportation, etc.)

Graph-Oriented Database

- **Main Elements:**
- **Nodes:** Objects or entities (e.g., person, product, etc.)
- **Edges or Relationships:** Relationships between nodes (e.g., "likes," "works_with")
- **Properties:** Attributes attached to nodes or edges

Example: (Alice)-[aime]->(Chocolat)

Graph-Oriented Database

Tool	Description
Neo4j	The most well-known and used, with Cypher
OrientDB	graphe/document
ArangoDB	Multi-model (document + graph)
Amazon Neptune	AWS cloud solution
JanusGraph	Distributed graph database, integrable with Big Data
IBM DB2	**RDF** triplestore, SPARQL Language
Oracle	**RDF** triplestore, SPARQL Language
AllegroGraph	**RDF** triplestore, SPARQL Language
MarkLogic	**RDF** triplestore, SPARQL Language,, XQuery

Graph-Oriented Database

- RDF is a standard data model for the Semantic Web, defined by the W3C.
- It is designed to describe resources (documents, people, concepts, etc.) and their relationships in a structured and machine-readable manner.

- RDF allows data to be automatically linked and interpreted by computer systems.

Graph-Oriented Database

- RDF is based on a simple structure:
- **Subject**: the resource being discussed
- **Predicate**: the property or relationship
- **Object**: the value or other related resource

This is called an RDF triplet:

(Subject) -- (Predicate) --> (Object)

(Alice) -- (hasAFriend) --> (Bob)

Graph-Oriented Database

- RDF
- Example : (Alice) -- (hasAFriend) --> (Bob)
- En Format RDF :

<http://example.org/Alice> <http://example.org/ hasAFriend > <http://example.org/Bob>

- There are several formats for writing RDF data:
- Turtle (.ttl): human-readable
- RDF/XML: XML-based
- JSON-LD: JSON-based

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix schema: <https://schema.org/> .
@prefix ex: <http://example.org/> .

# --- Utilisateurs ---
ex:Alice a foaf:Person ;
   foaf:name "Alice" ;
   foaf:age "25" ;
   foaf:mbox "alice@example.org" ;
   foaf:knows ex:Bob, ex:Charlie .

ex:Bob a foaf:Person ;
   foaf:name "Bob" ;
   foaf:age "28" ;
   foaf:mbox "bob@example.org" ;
   foaf:knows ex:Alice .

ex:Charlie a foaf:Person ;
   foaf:name "Charlie" ;
   foaf:age "24" ;
   foaf:mbox "charlie@example.org" ;
   foaf:knows ex:Alice .

# --- Posts ---
ex:Post1 a schema:CreativeWork ;
   schema:author ex:Alice ;
   schema:text "Salut tout le monde ! C’est mon
premier post." ;
   schema:datePublished "2025-04-27" .

ex:Post2 a schema:CreativeWork ;
```

Graph-Oriented Database

- @prefix foaf: <http://xmlns.com/foaf/0.1/> .
- @prefix ex: <http://example.org/> .

- ex:Alice a foaf:Person ;
- foaf:name "Alice" ;
- foaf:mbox "alice@example.org" .

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

 xmlns:foaf="http://xmlns.com/foaf/0.1/"

 xmlns:ex="http://example.org/">

  <foaf:Person
rdf:about="http://example.org/Alice">
    <foaf:name>Alice</foaf:name>
    <foaf:mbox>alice@example.org</foaf:mbox>
  </foaf:Person>

</rdf:RDF>
```

Graph-Oriented Database

- Cypher de Neo4j

```
CREATE (alice:Person {name: 'Alice'})
CREATE (bob:Person {name: 'Bob'})
CREATE (alice)-[:KNOWS]->(bob)
```

➔Requests: Find all of Alice's friends:

```
MATCH (alice:Person {name: 'Alice'})-[:KNOWS]->(friend)
RETURN friend.name
```

Graph-Oriented Database

//Create Person

```
CREATE (alice:Person {name: 'Alice', age: 25})
CREATE (bob:Person {name: 'Bob', age: 28})
CREATE (charlie:Person {name: 'Charlie', age: 24})
```

//Create Post

```
CREATE (post1:Post {title: 'Découverte du Web
Sémantique', date: '2025-04-27'})
CREATE (post2:Post {title: 'Comment bien coder en
Python', date: '2025-04-26'})
```

//Create Relation

é (KNOWS)

```
CREATE (alice)-[:KNOWS]->(bob)
CREATE (alice)-[:KNOWS]->(charlie)
```

//Create Relation Authored

```
CREATE (alice)-[:AUTHORED]->(post1)
CREATE (bob)-[:AUTHORED]->(post2)
```

```
MATCH (p:Person)
RETURN p.name AS name, p.age AS age
```

```
MATCH (alice:Person {name: 'Alice'})-[:KNOWS]->(friend)
RETURN friend.name AS friend_name
```

Trouver tous les posts publiés par Alice:

```
MATCH (alice:Person {name: 'Alice'})-[:AUTHORED]->
(post:Post)
RETURN post.title AS post_title, post.date AS
publication_date
```

Trouver tous les amis d'Alice et leurs posts:

```
MATCH (alice:Person {name: 'Alice'})-[:KNOWS]->(friend)-
[:AUTHORED]->(post:Post)
RETURN friend.name AS friend_name, post.title AS
post_title
```

Graph-Oriented Database

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:rss="http://purl.org/rss/1.0/"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
   <foaf:Person rdf:about="http://example.net/Paul_Dupont">
       <foaf:name>Paul Dupont</foaf:name>
       <foaf:img rdf:resource="http://example.net/Paul_Dupont.jpg"/>
       <foaf:knows rdf:resource="http://example.net/Pierre_Dumoulin"/>
   </foaf:Person>
   <foaf:Person rdf:about="http://example.net/Pierre_Dumoulin">
       <foaf:name>Pierre Dumoulin</foaf:name>
       <foaf:img rdf:resource="http://example.net/Pierre_Dumoulin.jpg"/>
   </foaf:Person>
   <foaf:Image rdf:about="http://example.net/Paul_Dupont.jpg">
       <dc:description>Photo d'identité de Paul Dupont</dc:description>
   </foaf:Image>
   <foaf:Image rdf:about="http://example.net/Pierre_Dumoulin.jpg">
       <dc:description>Photo d'identité de Pierre Dumoulin</dc:description>
   </foaf:Image>
</rdf:RDF>
```

Base de données Orienté Graphes

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT DISTINCT ?nom ?image ?description
WHERE {
    ?personne rdf:type foaf:Person.
    ?personne foaf:name ?nom.
    ?image rdf:type foaf:Image.
    ?personne foaf:img ?image.
    ?image dc:description ?description
}
```