

# HCIA-Big Data V3.0 Training Materials



# Contents

---

1.	Chapter 1 Big Data Development Trend and Kunpeng Big Data Solution .....	4
2.	Chapter 2 HDFS and ZooKeeper .....	70
3.	Chapter 3 Hive - Distributed Data Warehouse .....	116
4.	Chapter 4 HBase Technical Principles .....	144
5.	Chapter 5 MapReduce and YARN Technical Principles .....	195
6.	Chapter 6 Spark - An In-Memory Distributed Computing Engine .....	231
7.	Chapter 7 Flink, Stream and Batch Processing in a Single Engine .....	274
8.	Chapter 8 Flume - Massive Log Aggregation .....	339
9.	Chapter 9 Loader Data Conversion .....	379
10.	Chapter 10 Kafka Distributed Publish-Subscribe Messaging System .....	404
11.	Chapter 11 LDAP and Kerberos .....	441
12.	Chapter 12 Elasticsearch - Distributed Search Engine .....	489
13.	Chapter 13 Redis In-Memory Database .....	519
14.	Chapter 14 Huawei Big Data Solution .....	560

# Revision Record

Do Not Print this Page

Course Code	Product	Product Version	Course Version
H13-711	MRS		V3.0

Author/ID	Date	Reviewer/ID	New/ Update
Chen Ao/cwx860206	2020.04.09	Guan Wenzheng/gwx511452	New
Chen Ao/cwx860206	2020.04.15	Yan Hua/ywx416015	New

# Chapter 1 Big Data Development Trend and Kunpeng Big Data Solution



# Foreword

---

- This chapter consists of two parts.
  - The first part mainly describes what is big data and the opportunities and challenges we face in the age of big data.
  - To keep up with the trend and help partners to improve computing capabilities and data governance capabilities in intelligent transformation, Huawei proposes the strategy of Kunpeng ecosystem. Therefore, the second part describes the Huawei Kunpeng Big Data solution, including the Kunpeng server based on the Kunpeng chipset and HUAWEI CLOUD Kunpeng cloud services. In addition, this part briefly describes the common public cloud services related to big data and data analysis and processing in HUAWEI CLOUD Stack 8.0, and introduces the advantages and application scenarios of HUAWEI CLOUD MRS.

# Objectives

- After completing this course, you will be able to:
  - Understand what is big data, its four Vs, and opportunities and challenges in the big data era.
  - Master the development trend and application of big data technologies.
  - Understand Huawei Kunpeng Solution.
  - Understand Huawei Big Data Solution.

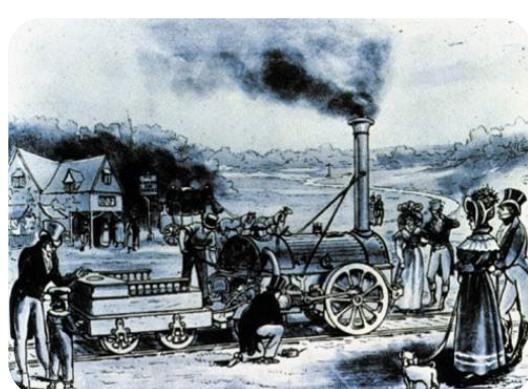
# Contents

## 1. Challenges and Opportunities in the Big Data Era

- Big Data Era
  - Big Data Application Fields
  - Big Data Computing Tasks
  - Challenges and Opportunities for Enterprises

## 2. Huawei Kunpeng Big Data Solution

# Ushering in the 4th Industrial Revolution and Embracing the Intelligent Era



**Steam Age**  
1760s - 1840s  
United Kingdom



**Electricity Age**  
1860s - 1920s  
Europe and America

Computer and communication



**Information Age**  
1940s - 2010s  
United States



**Age of Intelligence**  
(?)

Cloud computing,  
big data, IoT, and AI

# Moving From Data Management to Data Operations

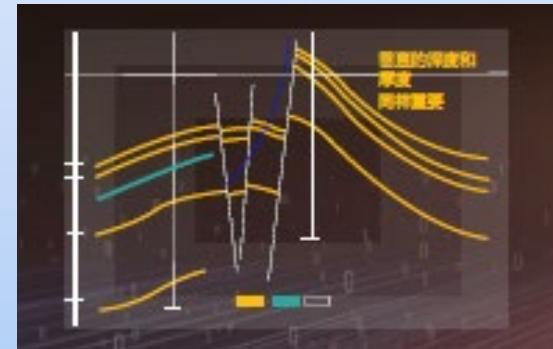
## Data Drives Experience



**120 million** data labels  
every day

Data determines  
user experience.

## Data Drives Decision-Making



**50 years** of oilfield data  
analytics

Data analytics determines  
oil extraction efficiency.

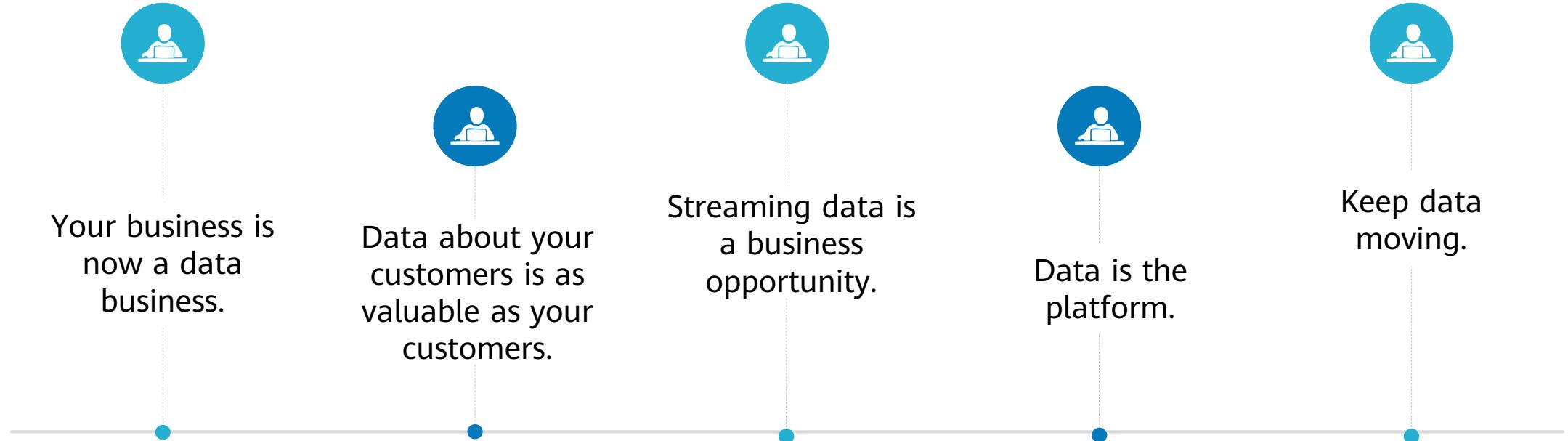
## Data Drives Processes



**450,000** collisions of data  
every day

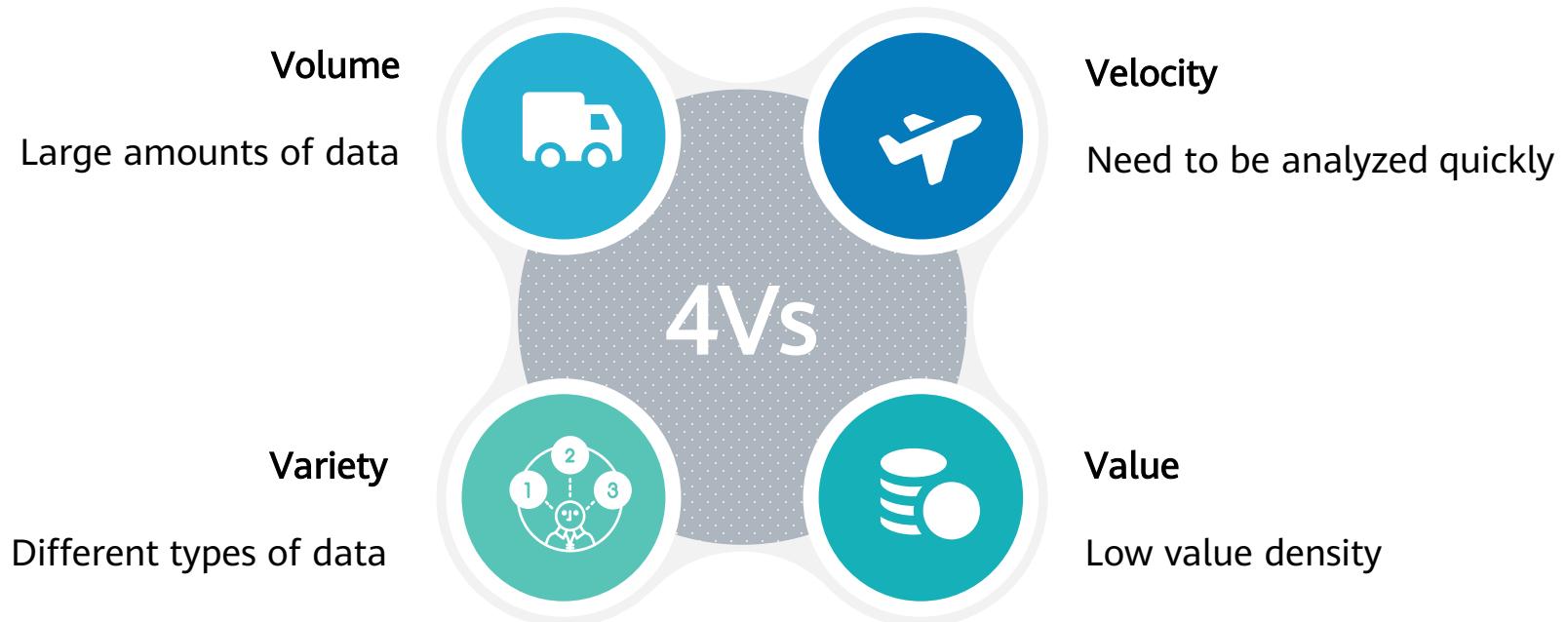
Data flow determines  
process simplicity.

# Everything Is Data and Data Is Everything



# Big Data Era

- Definition on Wikipedia:
  - Big data refers to data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process data within a tolerable elapsed time.



# Big Data Processing VS. Traditional Data Processing

- From database to big data
  - "Fishing in the pond" VS "Fishing in the ocean"  
("fish" indicates the data to be processed)



	Big Data Processing	Traditional Data Processing
Data scale	Large (in GB, TB, or PB)	Small (in MB)
Data type	Various data types (structured, semi-structured, and non-structured data)	Single data type (mostly structured data)
Relationship between mode and data	Modes are set after data is generated. Modes evolve when data increases.	Modes are set before data is generated.
Object to be processed	"Fish in the ocean". "Some fishes" are used to determine whether other types of fish exist.	"Fish in the pond"
Processing tool	No size fits all.	One size fits all.

# Contents

## 1. Challenges and Opportunities in the Big Data Era

- Big Data Era
- Big Data Application Fields
- Big Data Computing Tasks
- Challenges and Opportunities for Enterprises

## 2. Huawei Kunpeng Big Data Solution

# Big Data Era Leading the Future

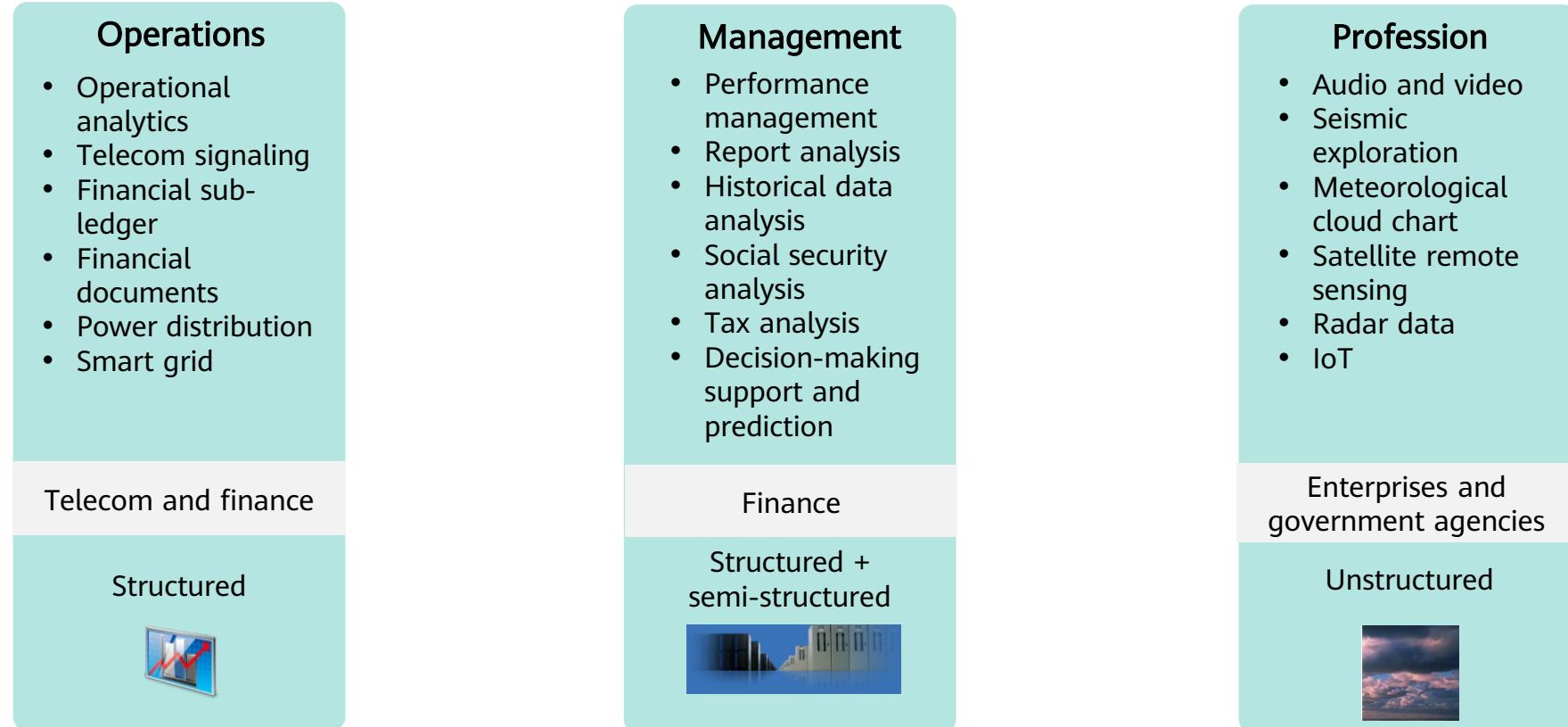
Data has penetrated into every inch of the industry and business domain.

**Discerning essences (services), forecasting trends, and guiding the future are the core of the big data era.**

**With a clear future target, seize every opportunity to harness big data to secure future success.**



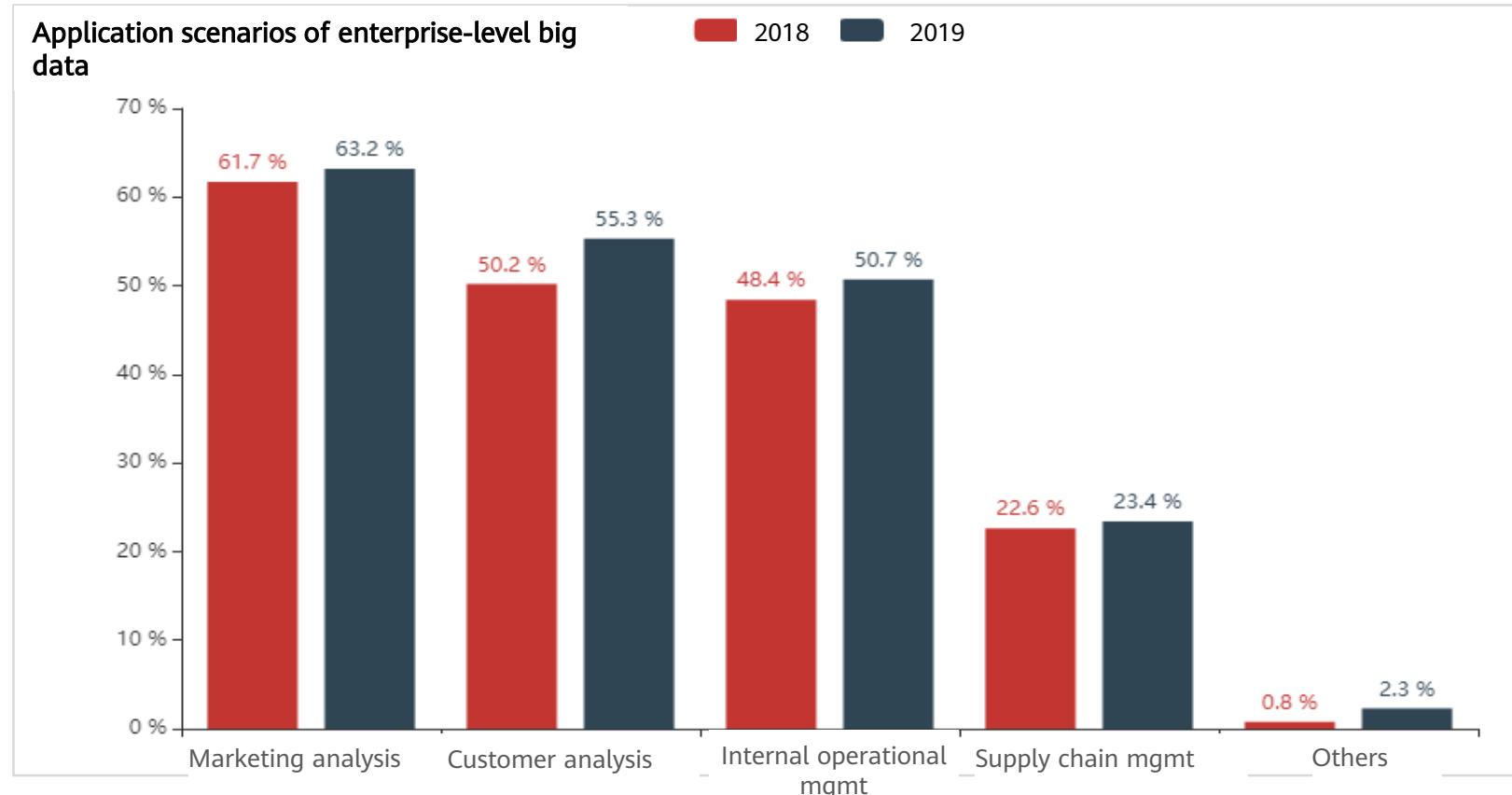
# Application Scenarios of Enterprise - Level Big Data Platforms (1)



With strong appeals for data analytics in telecom carriers, financial institutions, and governments, the Internet has adopted new technologies to process big data of low value density.

# Application Scenarios of Enterprise - Level Big Data Platforms (2)

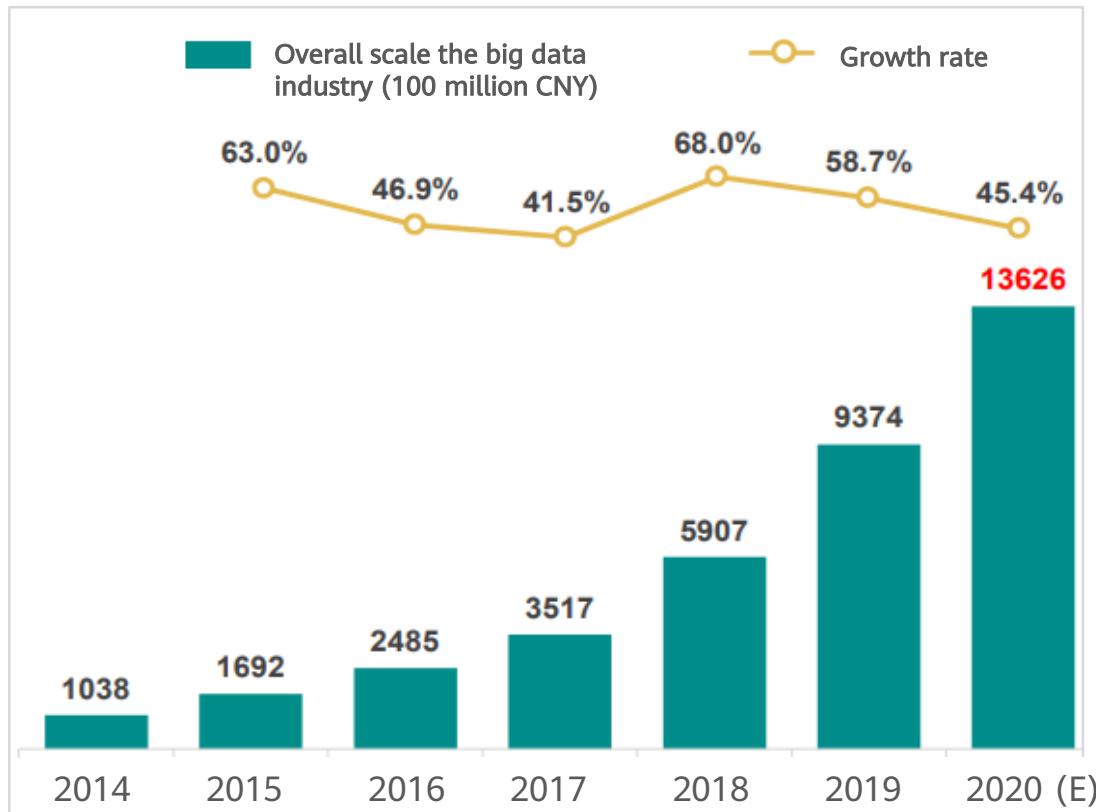
- Marketing analysis, customer analysis, and internal operational management are the top three application scenarios of enterprise big data.



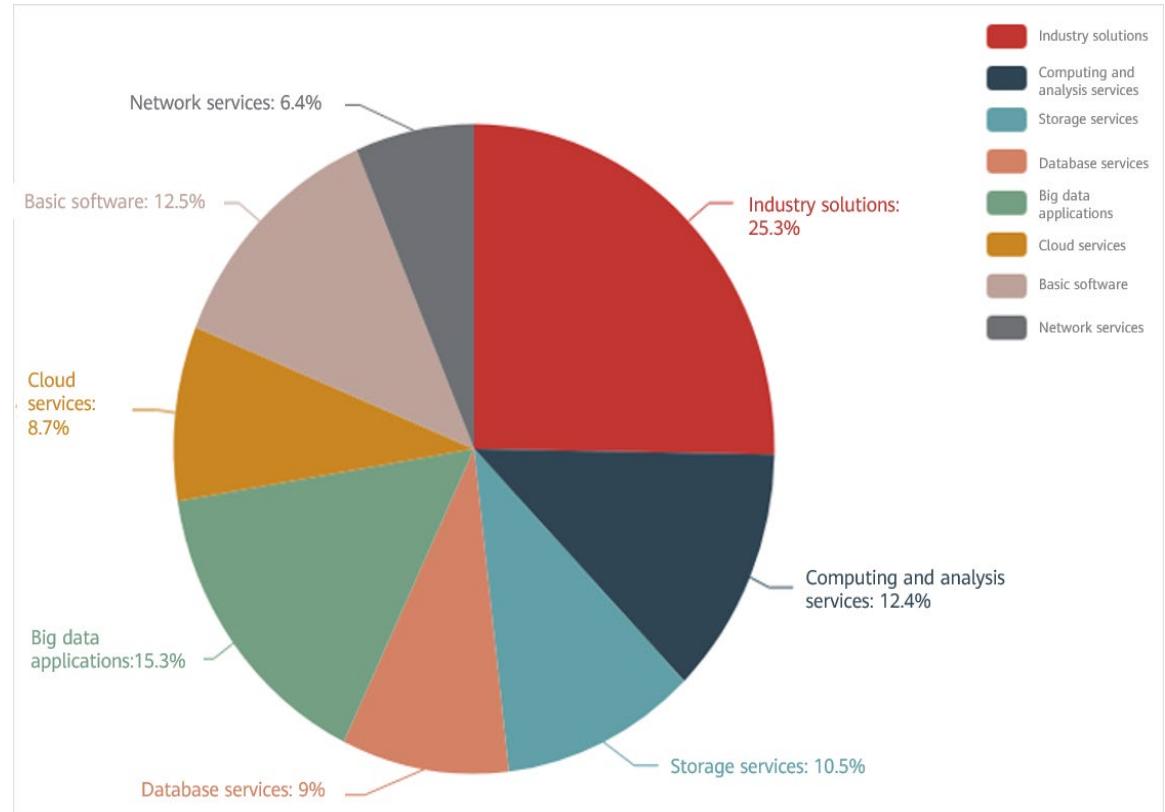
# Big Data Market Analytics

- It is predicted that the overall scale of the big data industry will exceed CNY 1 trillion by the end of 2020, in which industry-specific solutions and big data applications account for the largest proportion.

- Overview of the big data industry in China



- Market scale proportion of big data market segments

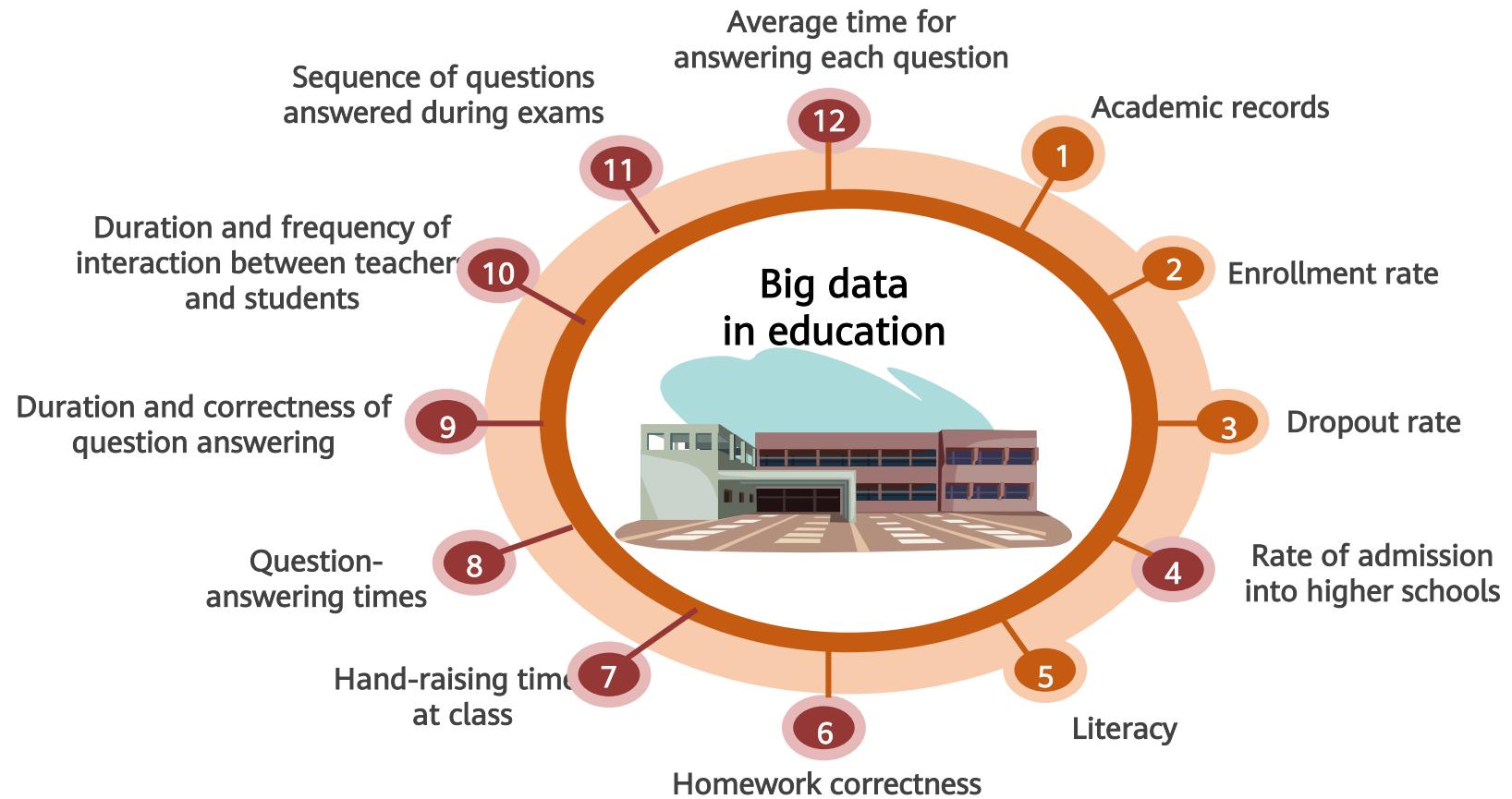


# Big Data Application Scenarios - Finance



# Big Data Application Scenarios - Education

Big data analytics has now been widely applied to the education field.



# Big Data Application Scenarios - Traffic Planning

## Traffic planning: multi-dimensional analysis of crowds

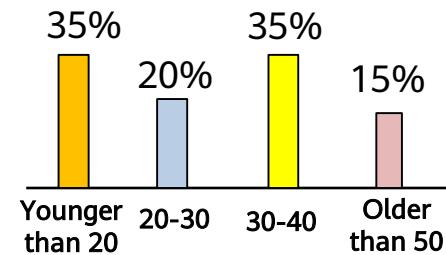


Traffic prediction based on crowd analysis

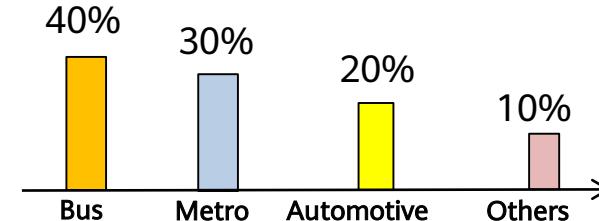
Areas where the people flow ever exceeded the specified threshold

- North gate of the Workers' Stadium: > 500 people/ hour
- Sanlitun: > 800 people/hour
- Beijing Workers' Stadium: > 800 people/hour

Analysis by crowd



Analysis by transportation method



Road network planning



Bus network planning

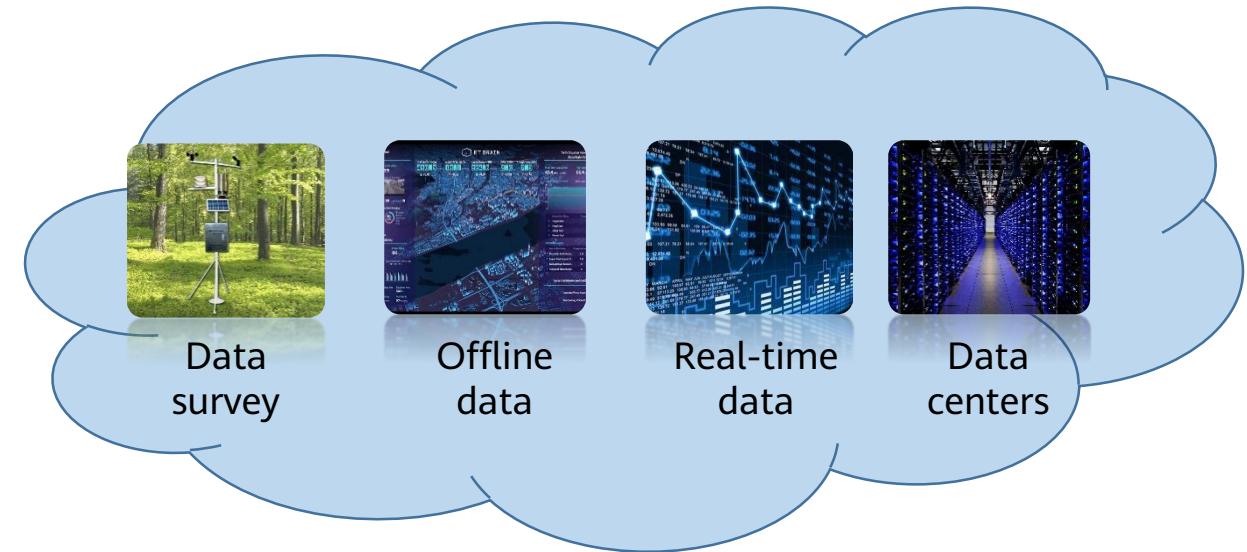


# Big Data Application Scenarios - Clean Energy

Clean energy powers China's Qinghai province for nine consecutive days.

Coal consumption ↓ 800,000 tons

CO<sub>2</sub> emission ↓ 1.44 million tons



Intelligence + Data help fight for clear water and blue sky.

# Contents

## 1. Challenges and Opportunities in the Big Data Era

- Big Data Era
- Big Data Application Fields
- Big Data Computing Tasks
- Challenges and Opportunities for Enterprises

## 2. Huawei Kunpeng Big Data Solution

# I/O-intensive Tasks

- I/O-intensive tasks are tasks involving network, disk, memory, and I/O.
- Characteristics: The CPU usage is low, and most latency is caused by I/O wait (CPU and memory computing is far quicker than I/O processing).
- More I/O-intensive tasks indicate higher CPU efficiency. However, there is a limit. Most applications are I/O-intensive, such as web applications.
- During the execution of I/O-intensive tasks, 99% of the time is spent on I/O wait. Therefore, top priority is to improve the network transmission and read/write efficiency.

# CPU-intensive Tasks

- Characteristics: A large number of computing tasks are performed, including Pi calculation and decoding HD videos, which consumes CPU resources.
- CPU-intensive tasks can be completed in parallel. However, more tasks mean longer duration for switching tasks, and task processing on CPU will be less efficient. Hence, to put the best of CPU performance, keep the number of parallel CPU-intensive tasks equal to the number of CPU cores.
- CPU-intensive tasks mainly consume CPU resources. Therefore, the code running efficiency is critically important.

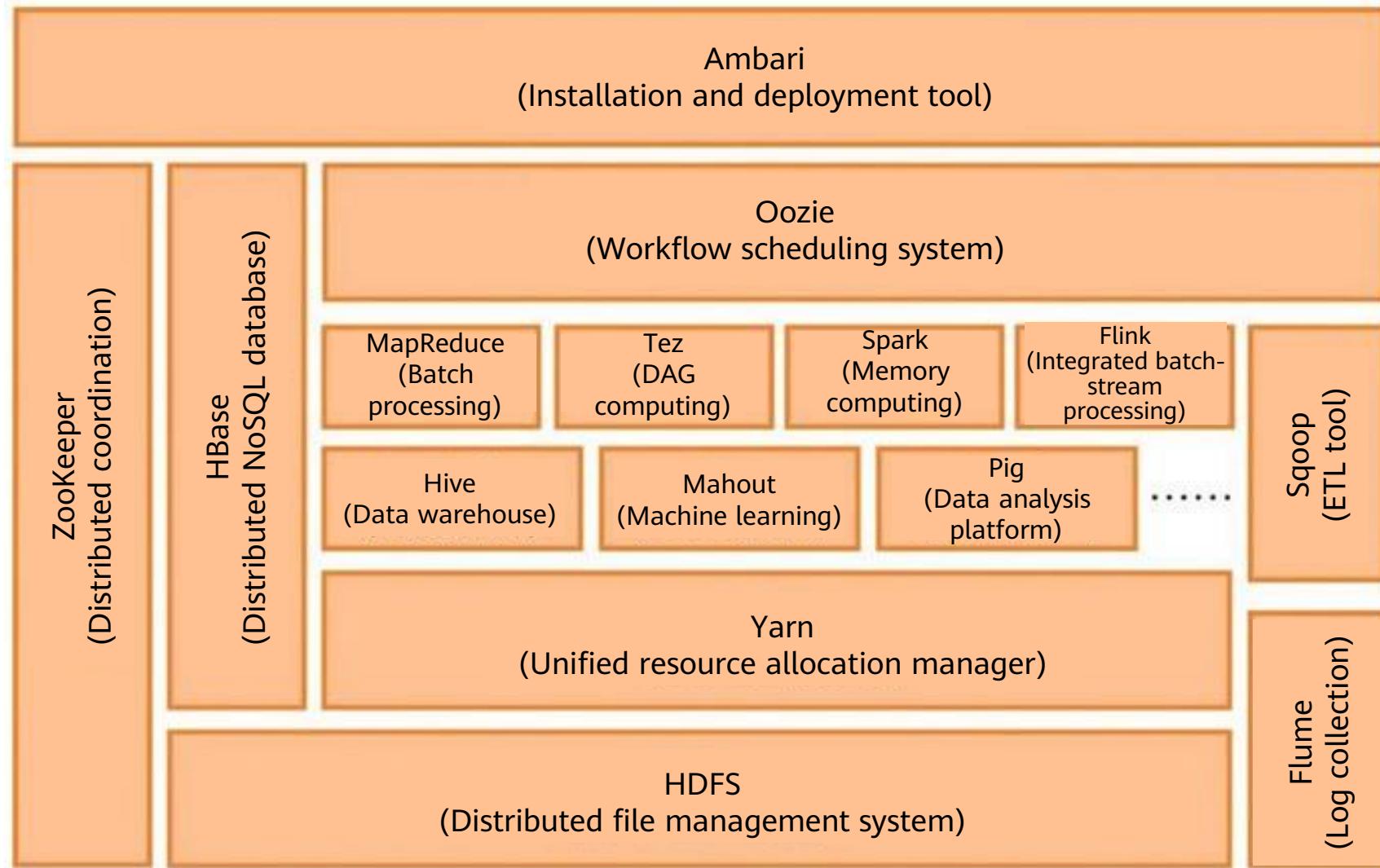
# Data-intensive Tasks

- Unlike CPU-intensive applications where a single computing task occupies a large number of computing nodes, data-intensive applications have the following characteristics:
  - A large number of independent data analysis and processing tasks run on different nodes of a loosely coupled computer cluster system.
  - High I/O throughput is required by massive volumes of data.
  - Most data-intensive applications have a data-flow-driven process.
- Typical applications of data-intensive computing:
  - Log analysis of Web applications
  - Software as a service (SaaS) applications
  - Business intelligence applications for large enterprises

# Major Computing Modes

- Batch processing computing
  - Allows you to process a large amount of data in batches. Major technologies: MapReduce and Spark
- Stream computing
  - Allows you to calculate and process stream data in real time. Major technologies: Spark, Storm, Flink, Flume, and DStream
- Graph computing
  - Allows you to process large volumes of graph structure data. Major technologies: GraphX, Gelly, Giraph, and PowerGraph
- Query and analytics computing
  - Allows you to manage, query, and analyze a large amount of stored data. Major technologies: Hive, Impala, Dremel, and Cassandra

# Hadoop Big Data Ecosystem



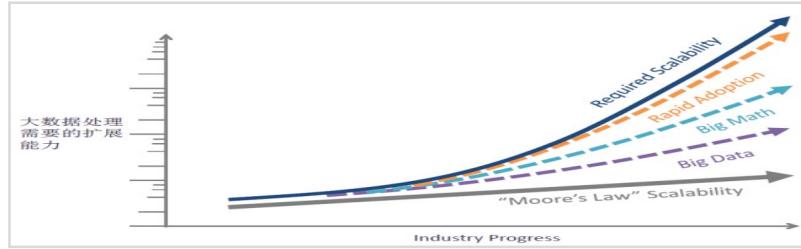
# Contents

## 1. Challenges and Opportunities in the Big Data Era

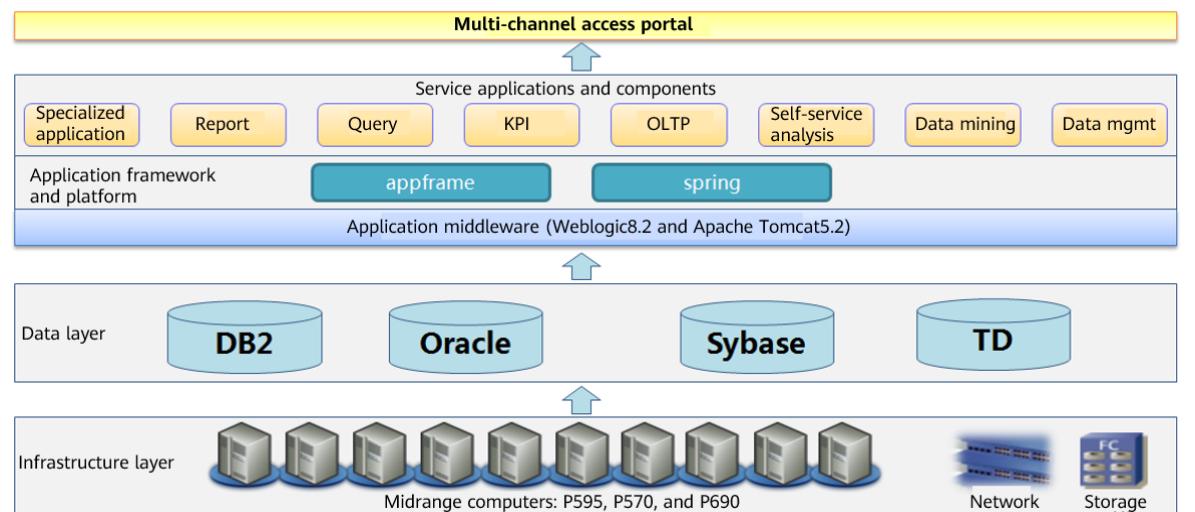
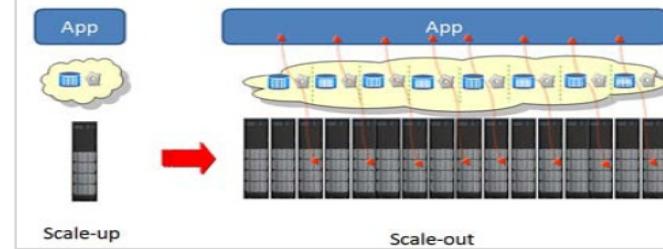
- Big Data Era
- Big Data Application Fields
- Big Data Computing Tasks
- Challenges and Opportunities for Enterprises

## 2. Huawei Kunpeng Big Data Solution

# Traditional Data Processing Are Facing Great Challenges



↑  
Gap between data scalability  
requirements and hardware  
performance  
↓



Traditional framework: midrange computer + disk array + commercial data warehouse

- High cost of mass data storage
- Insufficient performance of batch data processing
- Missing capability of stream data processing
- Limited scalability
- Single data source
- External value-added data assets

# Challenge 1: Few Business Departments Have Clear Big Data Requirements

- Many enterprise business departments have no idea about the values and application scenarios of big data, and therefore have no accurate requirements for big data. In addition, the enterprise decision-makers are worried that establishing a big data department may yield little profits and therefore even delete a large amount of historical data with potential values.



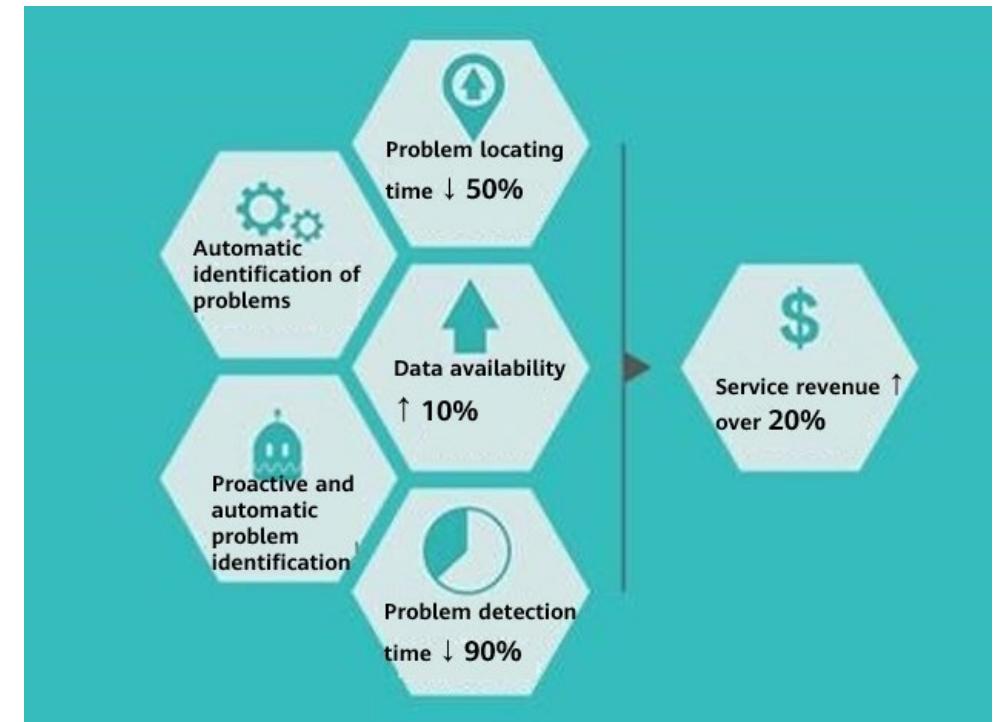
# Challenge 2: Data Silos Within Enterprises

- The greatest challenge for enterprises to implement the big data strategy is that different types of data are scattered in different departments. As a result, data in the same enterprise cannot be efficiently shared, and the value of big data cannot be brought into full play.



# Challenge 3: Poor Data Availability and Quality

- Many large- and medium-sized enterprises generate large volumes of data every day. However, many of them fail to pay enough attention to data preprocessing. As a result, data is not processed in a standard way. In the big data preprocessing phase, data needs to be extracted and converted into data types that can be easily processed, and cleaned by removing noisy data. According to Sybase, with the availability of high-quality data improved by 10%, the profits of enterprises would be improved by 20% consequently..



# Challenge 4: Unsatisfactory Data Management Technologies and Architecture

- Traditional databases are not suitable for processing PB-scale data.
- It is difficult for traditional database systems to process semi-structured and unstructured data.
- The O&M of massive volumes of data requires data stability, high concurrency, and server load reduction.



# Challenge 5: Data Security Risks

- A rapid spread of the Internet increases the chance of breaching the privacy of individuals and also leads to more crime methods that are difficult to be tracked and prevented.
- It is a key issue to ensure user information security in this big data era. In addition, the increasing amount of big data poses higher requirements on the physical security of data storage, and therefore higher requirements on multi-copy and DR mechanisms.



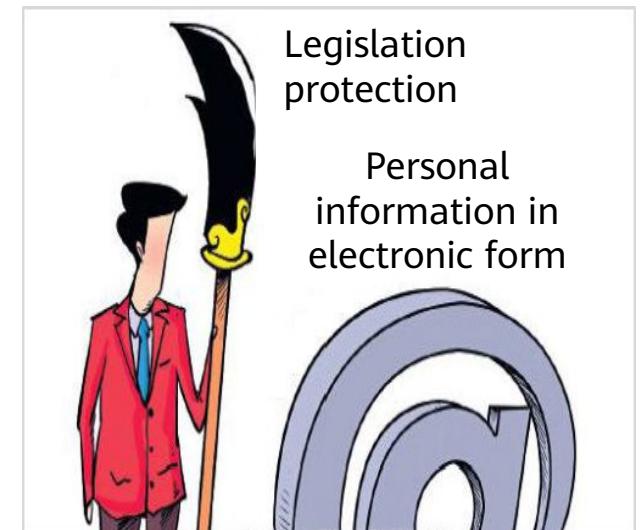
# Challenge 6: Lack of Big Data Talent

- Each step of big data construction must be completed by professionals. Therefore, it is necessary to develop a professional team that understands big data, knows much about administration, and has experience in big data applications. Hundreds of thousands of big data-related jobs are added each year around the world. In the future, there will be a big data talent gap of more than 1 million. Therefore, universities and enterprises make joint efforts to explore and develop big data talent.



# Challenge 7: Trade-off Between Data Openness and Privacy

- As big data applications become increasingly important, data resource openness and sharing have become the key to maintaining advantages against competitors. However, opening up data inevitably risks exposing some users' private information. Hence, it is a major challenge in this big data era to effectively protect citizens' and enterprises' privacy while promoting data openness, application, and sharing, and gradually strengthen privacy legislation.



# Standing Out in the Competition Using Big Data

- Big data can bring a huge commercial value, and it is believed to raise a revolution that is well matched with the computer revolution in 20th century. Big data is affecting commercial and economic fields and so on. It boosts a new blue ocean and hastens generation of new economic growth points, becoming a focus during the competition among enterprises.



# Opportunity 1: Big Data Mining Becomes the Core of Business Analysis

- The focus of big data has gradually shifted from storage and transmission to data mining and application, which will have a profound impact on enterprises' business models. Big data can directly bring enterprises profits and incomparable competitive advantages through positive feedback.
  - On the one hand, big data technologies can effectively help enterprises integrate, mine, and analyze the large volumes of data they have, build a systematic data system, and improve the enterprise structure and management mechanism.
  - On the other hand, with the increase of personalized requirements of consumers, big data is gradually applied in a wide range of fields and is shifting the development paths and business models of most enterprises.

# Opportunity 2: Big Data Bolsters the Application of Information Technologies

- Big data processing and analysis is a support point of application of next-generation information technologies.
  - Mobile Internet, IoT, social networking, digital home, and e-commerce are the application forms of next-generation information technologies. These technologies continuously aggregate generated information and process, analyze, and optimize data from different sources in a unified and comprehensive manner, feedback or cross-feedback of results to various applications further improves user experience and creates huge business, economic, and social values.
  - Big data has the power to drive social transformation. However, to unleash this power, stricter data governance, insightful data analysis, and an environment that stimulates management innovation are required.

# Opportunity 3: Big Data Is a New Engine for Continuous Growth of the Information Industry

- Big data, with its tremendous business value and market demands, becomes a new engine that drives the continuous growth of the information industry.
  - With the increasing recognition of big data's value by industry users, market requirements will burst, and new technologies, products, services, and business models will emerge continuously in the big data market.
  - Big data drives the development of a new market with high growth for the information industry: In the field of hardware and integrated devices, big data faces challenges such as effective storage, fast read/write, and real-time analysis. This will have an important impact on the chip and storage industry and also give rise to the market of integrated data storage and processing servers and in-memory computing. In the software and service field, the value of big data brings urgent requirements for quick data processing and analysis, which will lead to unprecedented prosperity of the data mining and business intelligence markets.

# Quiz

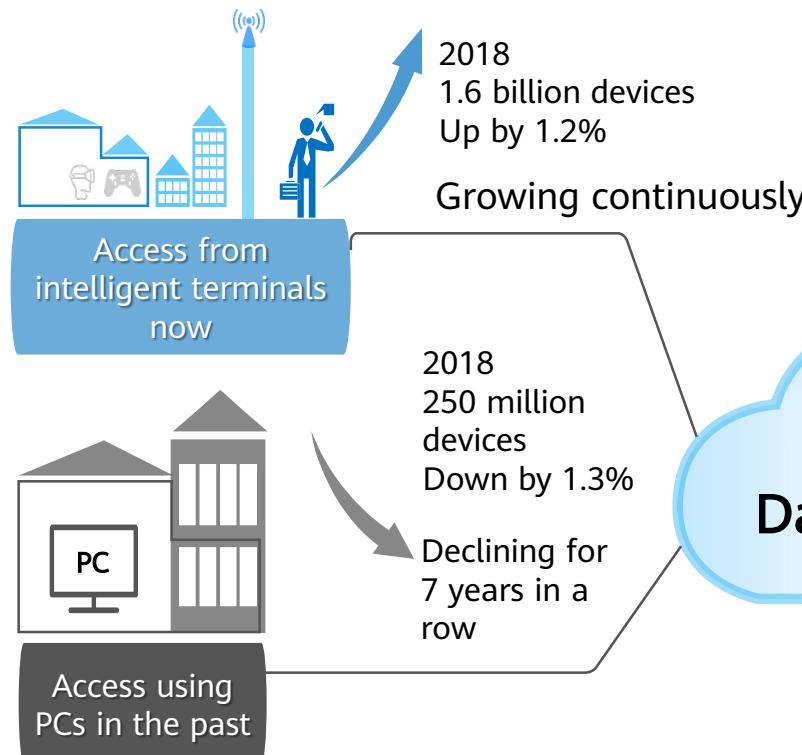
1. Where is big data from? What are the characteristics of big data?
2. Which fields can big data be applied to?
3. What challenges does big data face?

# Contents

1. Opportunities and Challenges in the Big Data Era
2. **Huawei Kunpeng Big Data Solution**
  - Introduction to Kunpeng
  - Kunpeng Big Data Solution

# Internet of Everything - Massive Volumes of Data Requires High Computing Power

Smart mobile devices are replacing traditional PCs.



Transition from traditional PCs to intelligent mobile terminals



Demand for new computing power



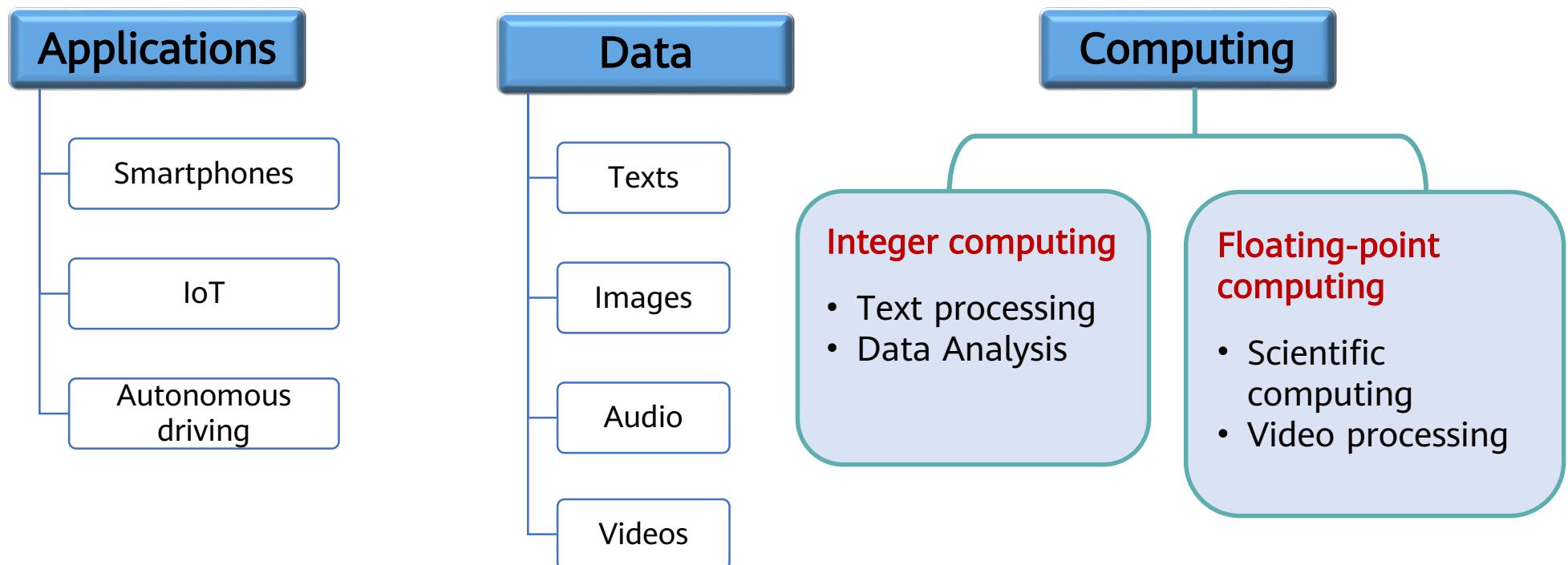
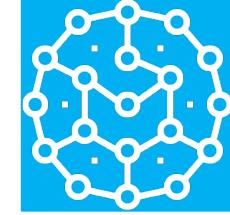
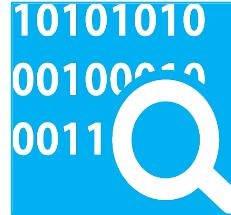
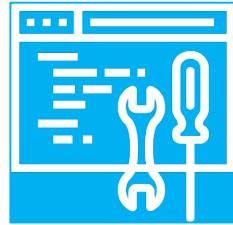
Massive volumes of data

The world is moving towards an era of all things connected

The number of connected devices worldwide exceeded 23 billion in 2018.

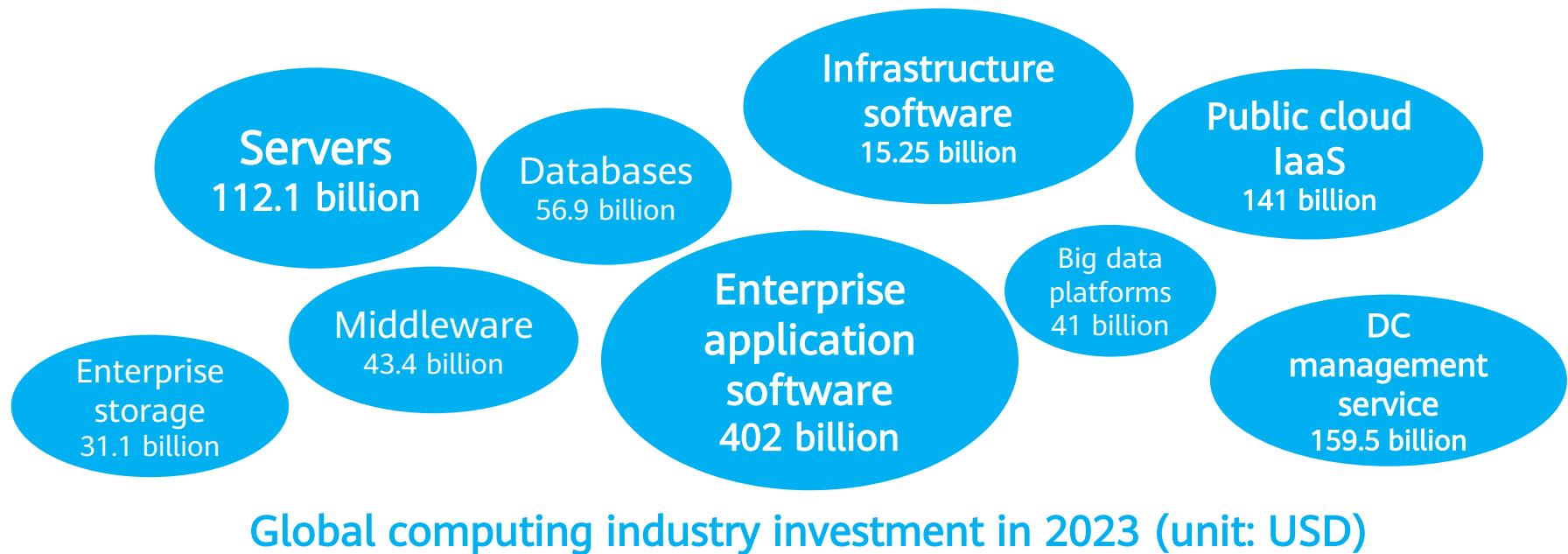


# Application and Data Diversity Requires a New Computing Architecture

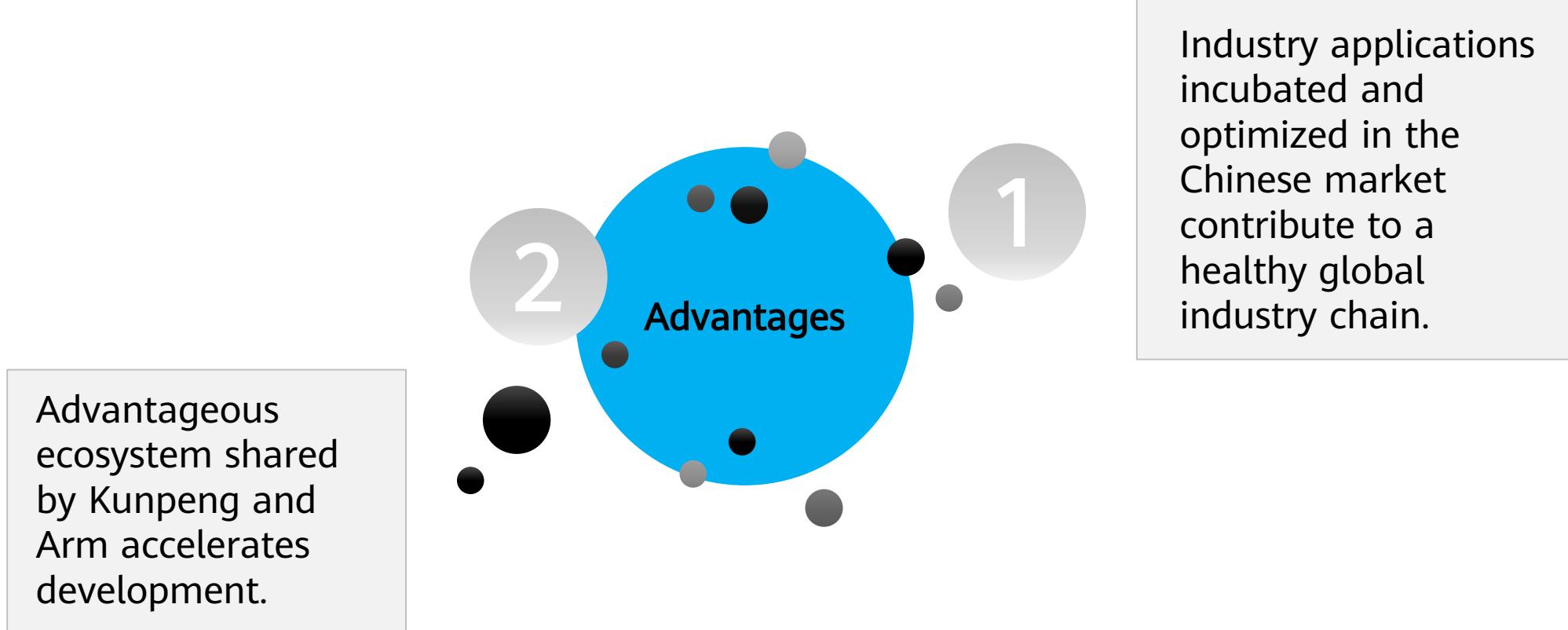


# Over Trillions of Dollars of Computing Market Space

- As the ICT industry landscape is being reshaped by new applications, technologies, computing architectures, tens of billions of connections, and explosive data growth, a new computing industry chain is taking shape, creating new vendors and hardware and software systems:
  - Hardware: servers, components, and enterprise storage devices
  - Software: operating systems, virtualization software, databases, middleware, big data platforms, enterprise application software, cloud services, and data center management services

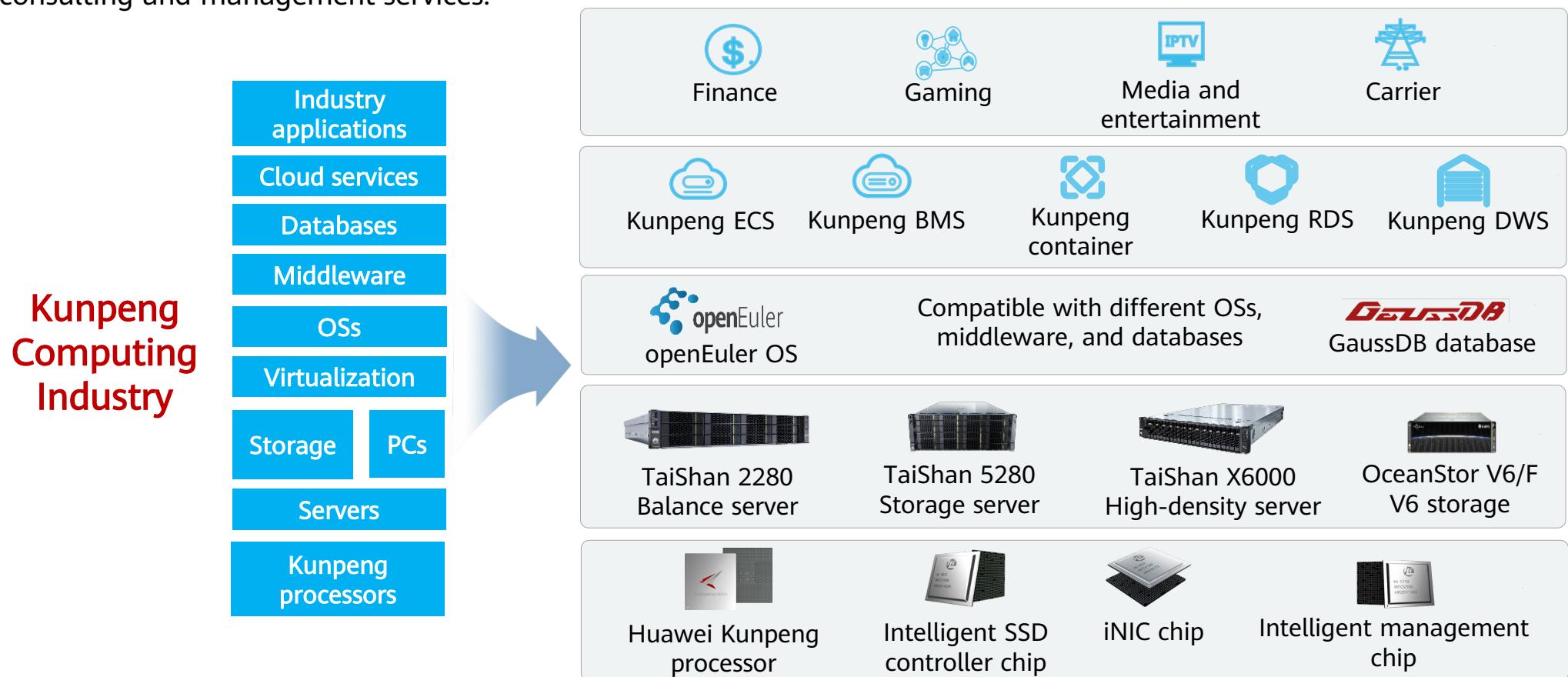


# Advantages of the Kunpeng Computing Industry



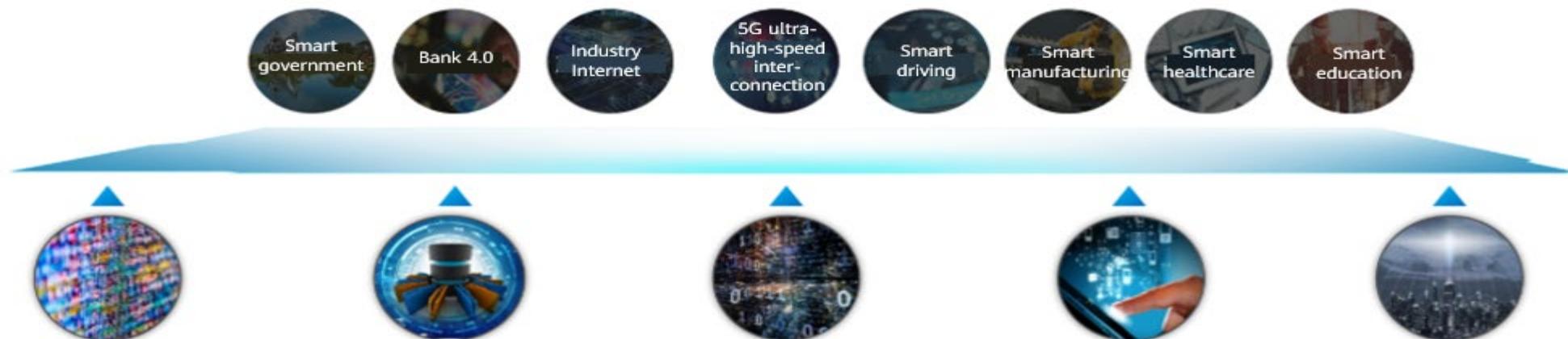
# Overall Architecture of the Kunpeng Computing Industry

- Based on Kunpeng processors, the Kunpeng computing industry covers full-stack IT infrastructure, industry applications, and services, including PCs, servers, storage devices, OSs, middleware, virtualization software, databases, cloud services, and consulting and management services.



# Typical Applications

- Driven by technologies such as 5G, AI, cloud computing, and big data, various industries have raised such requirements for computing platforms as device-cloud synergy, intelligent processing of massive volumes of diverse data, and real-time analysis. The powerful computing base provided by Kunpeng processors will play an important role in the digital transformation of industries.



## Big data

- More CPU cores
- Higher performance
- Encryption/Decryption engine

## Distributed storage

- Compression time reduced
- Higher IOPS
- Compression/Decompression engine

## Databases

- Higher performance
- RoCE low-latency network
- NUMA optimization algorithm

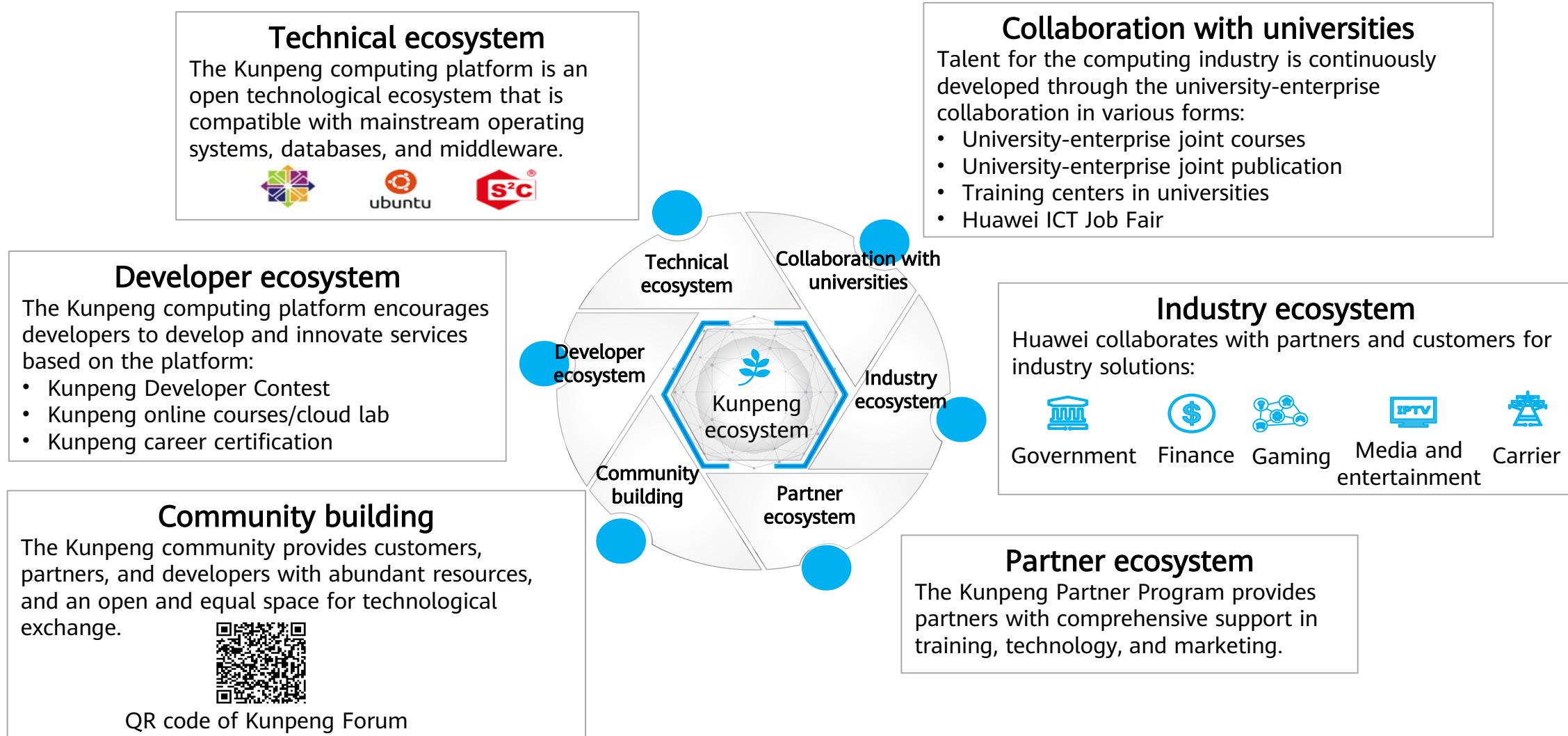
## Native applications

- Zero performance loss with natively homogeneous Arm
- Ecosystem with 5 million native applications

## Cloud services

- Higher memory bandwidth of BMSs
- Higher multi-core integer performance of ECSs
- Hybrid deployment of Kubernetes containers

# Panorama of Kunpeng Computing Industry Ecosystems



# Build the Computing Capability of the Entire System Based on Huawei Kunpeng Processors

## Efficient computing

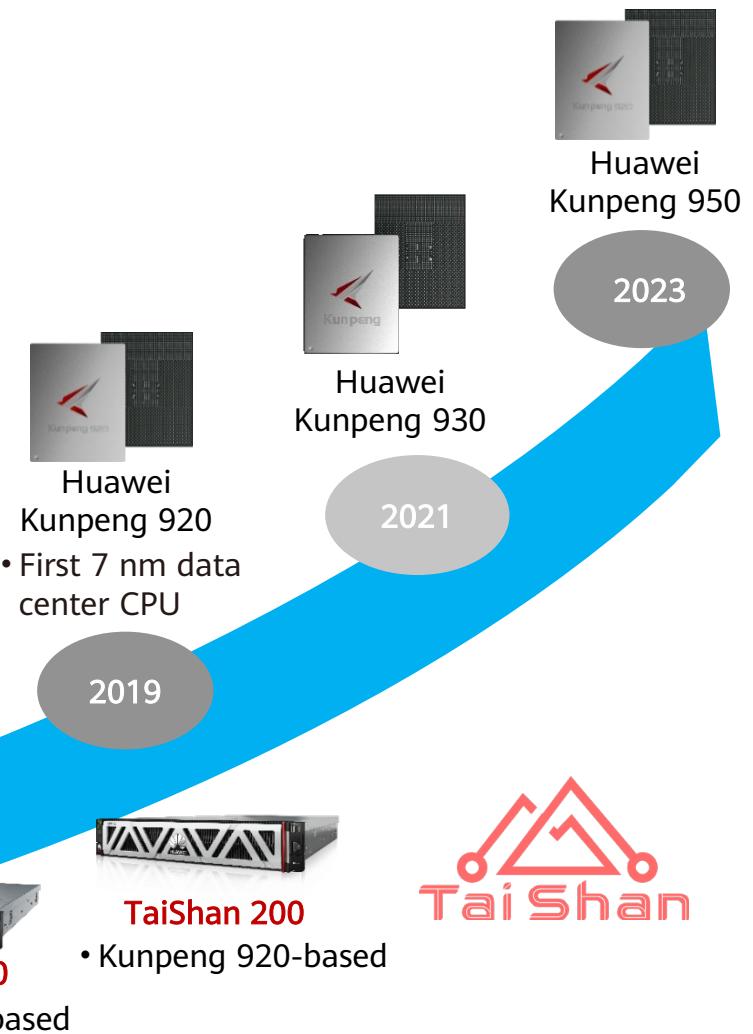
- Arm-compliant high-performance Kunpeng processors, TaiShan servers, and solutions efficiently improve the computing capabilities of data centers.

## Safe and reliable

- Kunpeng processors use Huawei-developed cores, and TaiShan servers use Huawei-developed computing chips.
- 17 years of computing innovation guarantees high quality.

## Open ecosystem

- Open platform supports mainstream hardware and software; Build a Kunpeng ecosystem, and establish a new smart computing base with developers, partners, and other industry organizations.



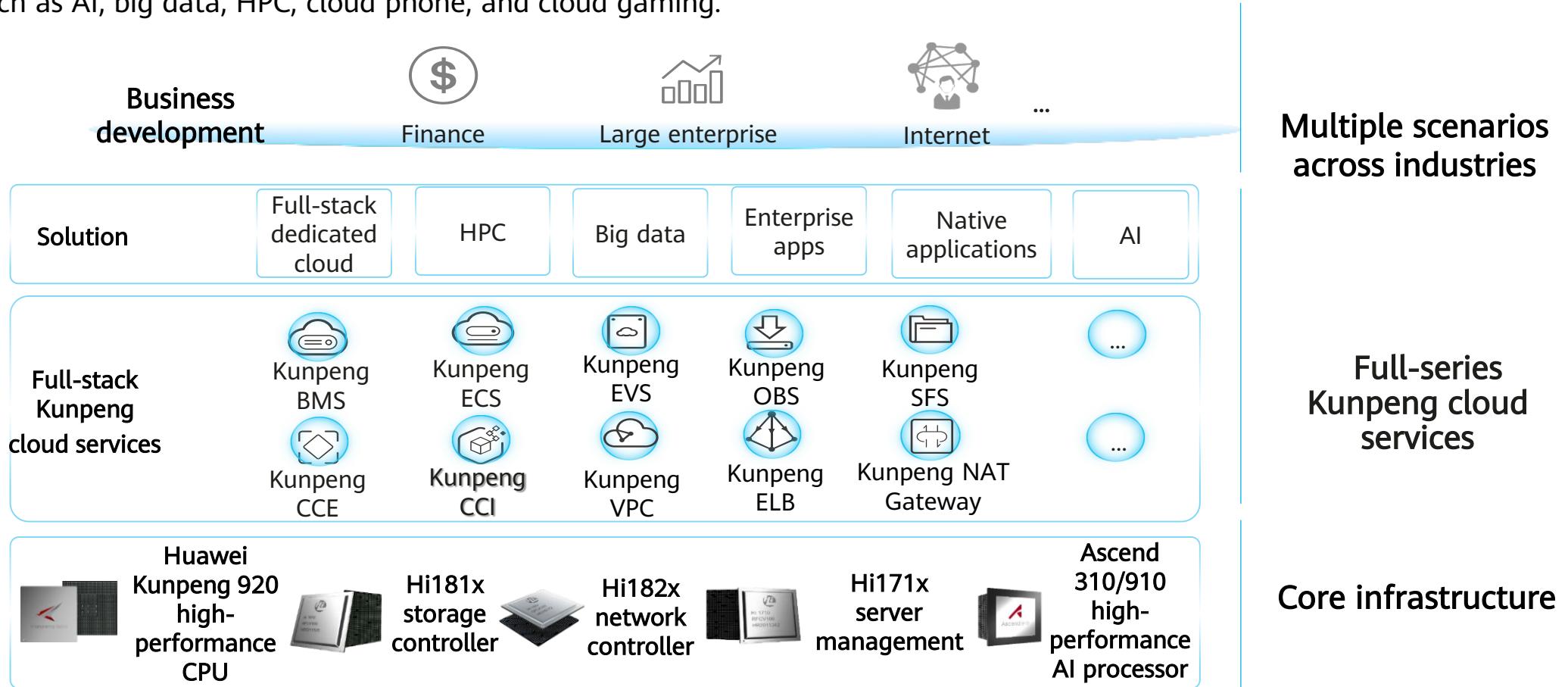
# OSs Compatible with the Kunpeng Ecosystem

	Community Edition	Commercial Edition
China-made OSs	 <b>openEuler</b> ...   <b>NeoKylin</b> <b>KYLIN</b>	 <b>Emindsoft</b>  <b>BC-Linux</b>  <b>LINX-TECH.</b>  <b>湖南麒麟 KYLINSEC</b> ...
Non-China-made OSs	 <b>CentOS</b>  <b>debian</b> ...  <b>SUSE</b> ...	 <b>ubuntu</b>



# Overview of HUAWEI CLOUD Kunpeng Cloud Services

Based on Kunpeng processors and other diverse infrastructures, HUAWEI CLOUD Kunpeng cloud services cover bare metal servers (BMSs), virtual machines, and containers. The services feature multi-core and high concurrency and are suitable for scenarios such as AI, big data, HPC, cloud phone, and cloud gaming.



# HUAWEI CLOUD Kunpeng Cloud Services Support A Wide Range of Scenarios

Transaction Processing	Big Data Analytics	Databases	Scientific Computing	Cloud Services	Storage	Mobile Native Apps
OLTP	OLAP	MySQL	CAE/CFD	Front-end web	Block storage	Cloud gaming
Web servers	Offline analysis	Redis	CAD/EDA	Data cache	Object storage	Game development and testing
Email	AI training	Gbase	Life science	Search	File	Terminal emulation
App service	AI inference	Oracle	Molecular Dynamics			Mobile office
ERP		Kingbase	Energy			Run on Arm servers.
CRM		Dameng	Meteorology			
		SAP	Defense/Security			

- Open-source software can run on Huawei Kunpeng platforms, and commercial software is gradually improved.
- Applications that support high concurrency deliver excellent performance.

# Contents

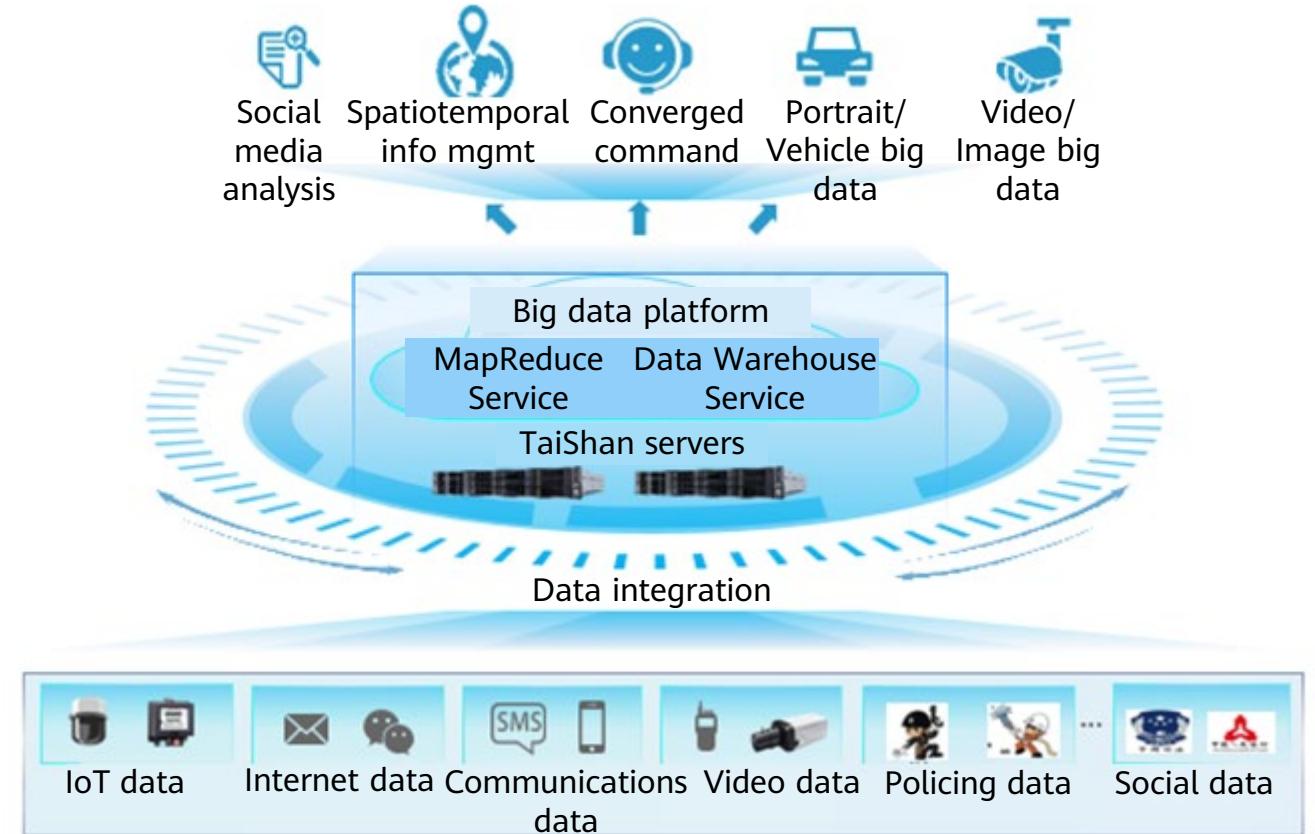
1. Opportunities and Challenges in the Big Data Era
2. **Huawei Kunpeng Big Data Solution**
  - Introduction to Kunpeng
  - Kunpeng Big Data Solution

# Huawei Big Data Solution

- Kunpeng Big Data Solution
  - Huawei's secure and controllable Kunpeng Big Data Solution provides one-stop high-performance big data computing and data security capabilities. This solution aims to resolve basic problems such as data security, efficiency, and energy consumption during intelligent big data construction in the public safety industry.
- BigData Pro
  - This solution adopts the public cloud architecture with **storage-compute decoupling, ultimate scalability, and highest possible efficiency**. The highly scalable Kunpeng computing power is used as the computing resource and Object Storage Service (OBS) that supports **native multi-protocols** is used as the storage pool. The resource utilization of big data clusters can be greatly improved, and the big data cost can be **halved**.
  - Drew on HUAWEI CLOUD's extensive experience, Huawei big data solution provides you with high-performance and highly-reliable infrastructure resources and AI training and inference platforms for big data services, witnessing your success in digitization and intelligentization.

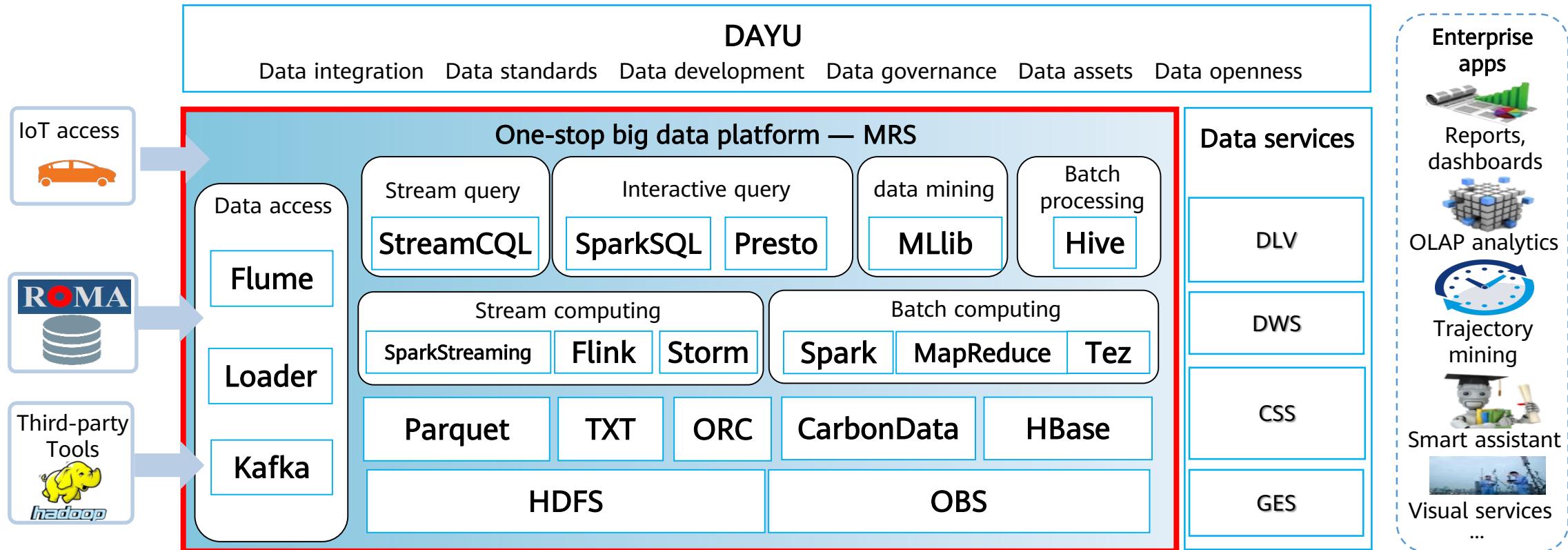
# Advantages of Huawei Big Data Solution

- High security
  - Controllable servers and big data platforms
  - Chip-level data encryption, ensuring data security
- High performance
  - 30% higher performance than common servers of the same level
  - Ultimate computing power and high-concurrency application scenario optimization
  - Support for big data clusters with 5000+ nodes
- High openness
  - Compatible with the Arm ecosystem and support for mainstream hardware and software
  - OpenLabs for services such as software development, application migration, and compatibility certification



# HUAWEI CLOUD Big Data Services

- One-stop service for data development, test, and application



- **100% compatibility** with open-source ecosystems, 3rd-party components managed as plug-ins, one-stop enterprise platform
- Storage-compute decoupling + Kunpeng optimization for **better performance**

# HUAWEI CLOUD MRS Overview

- MRS is a HUAWEI CLOUD service that is used to deploy and manage the Hadoop system and enables one-click Hadoop cluster deployment.
- MRS provides enterprise-level big data clusters on the cloud. Tenants can fully control clusters and easily run big data components such as Hadoop, Spark, HBase, Kafka, and Storm. MRS is fully compatible with open source APIs, and incorporates advantages of HUAWEI CLOUD computing and storage and big data industry experience to provide customers with a full-stack big data platform featuring high performance, low cost, flexibility, and ease-of-use. In addition, the platform can be customized based on service requirements to help enterprises quickly build a massive data processing system and discover new value points and business opportunities by analyzing and mining massive amounts of data in either real time or non-real time.

# Advantages of MRS (1)

- High performance
  - Leverages Huawei-developed CarbonData storage technology which allows one data set to apply to multiple scenarios.
  - Supports such features as multi-level indexing, dictionary encoding, pre-aggregation, dynamic partitioning, and quasi-real-time data query. This improves I/O scanning and computing performance and returns analysis results of tens of billions of data records in seconds.
  - Supports Huawei-developed enhanced scheduler *Superior*, which breaks the scale bottleneck of a single cluster and is capable of scheduling over 10,000 nodes in a cluster.
  - Optimizes software and hardware based on Kunpeng processors to fully release hardware computing power and achieve cost-effectiveness.

# Advantages of MRS (2)

- Easy O&M
  - Provides a visualized big data cluster management platform, improving O&M efficiency.
  - Supports rolling patch upgrade and provides visualized patch release information.
  - Supports one-click patch installation without manual intervention, ensuring long-term stability of user clusters.
  - Delivers high availability (HA) and real-time SMS and email notification on all nodes.

The screenshot displays the MRS Manager interface, specifically the 'Nodes' tab. On the left, a sidebar shows 'mrs\_zqfa' and navigation links for Dashboard, Nodes, Components, Alarms, Patches, Files, Jobs, and Tenants. Below this is a 'Configure Task Node' button. The main area is divided into two sections: 'Node Group' and 'Node Type'. Under 'Node Group', there are two expanded groups: 'master\_node\_default\_group' (Master) and 'core\_node\_analysis\_group' (Analysis Core). On the right, the 'Service Summary' section lists four services: DBService, Flink, Flume, and HBase, all marked as 'Good'. The 'Cluster Host Health Status' section features a large green circular gauge indicating a 'Good' status, with a small red segment labeled 'Bad' and a grey segment labeled 'Unknown'.

# Advantages of MRS (3)

- High security
  - With Kerberos authentication, MRS provides role-based access control (RBAC) and sound audit functions.
  - MRS is a one-stop big data platform that allows different physical isolation modes to be set up for customers in the public resource area and dedicated resource area of HUAWEI CLOUD as well as HCS Online in the customer's equipment room.
  - A cluster supports multiple logical tenants. Permission isolation enables the computing, storage, and table resources of the cluster to be divided based on tenants.

The image shows two side-by-side screenshots of a cloud management interface. On the left, under 'Kerberos Authentication', a toggle switch is turned on, and the 'Username' field is filled with 'admin'. Below it, the 'Password' and 'Confirm Password' fields both contain masked dots. On the right, the 'Tenant Management' tab is selected in a navigation bar with 'Resource Pool' and 'Dynamic Resource Plan' options. Under 'Create Tenant', 'QueueA' and 'QueueB' are highlighted with red boxes. In the main pane, a table shows tenant details:

Summary	Resource	Service Association	User
Description: QueueA			
Basic Information			
Name	Leaf Tenant	Space Quota	
QueueA	Yes	1024 MB	

# Advantages of MRS (4)

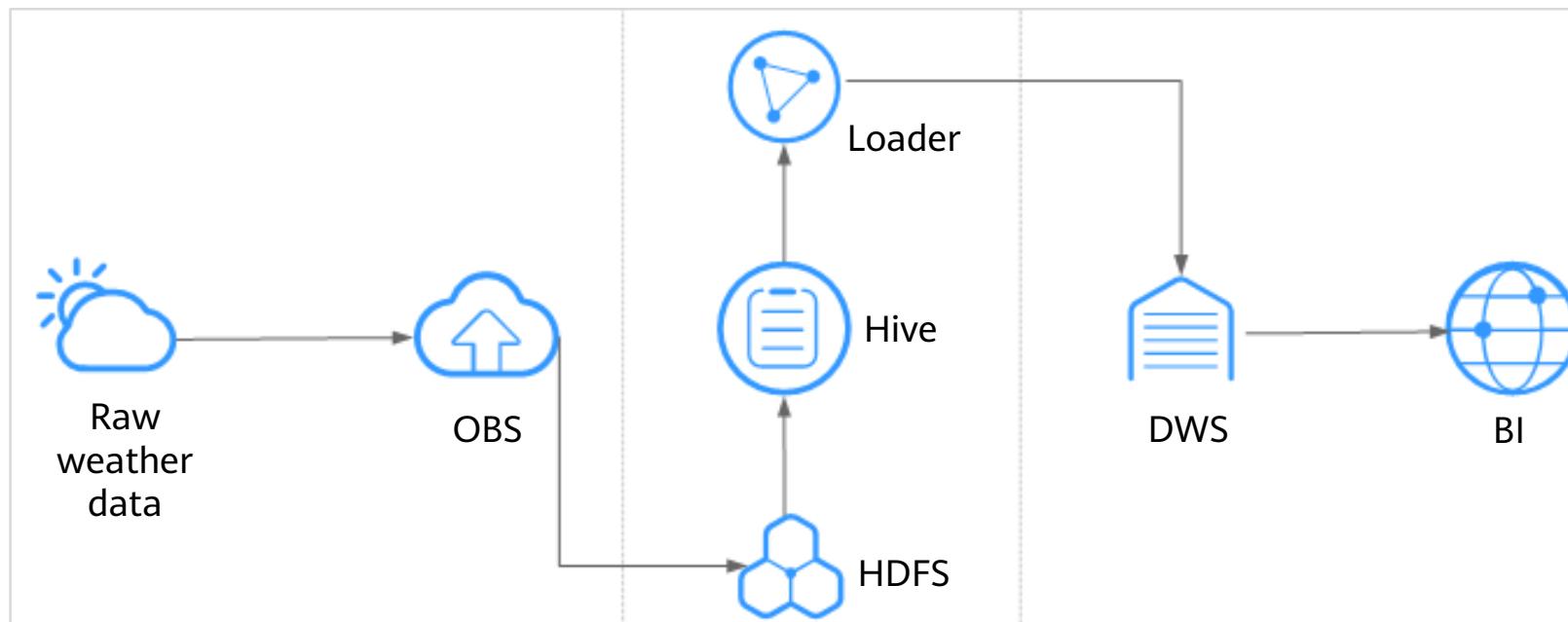
- Cost-effectiveness
  - Provides various computing and storage choices based on diverse cloud infrastructure.
  - Separates computing from storage, delivering low-cost massive data storage solutions.
  - Supports flexible configuration of node and disk specifications
  - Supports on-demand scale up or down of cluster
  - Supports temporary clusters, which are automatically deleted after job execution.
  - Supports custom policies and auto scaling of clusters, releasing idle resources on the big data platform for customers.

Cluster Node	Node Type	Billing Mode	Instance Specifications	Instance Count
	Master ⓘ	Pay-per-use	General computing-plus ⓘ 4 vCPUs   16 GB   c3ne.xlarge.4 System Disk High I/O 100 GB x 1 Data Disk High I/O 200 GB x 1	2 Cluster HA <input checked="" type="checkbox"/>
	Analysis Core ⓘ	Pay-per-use	General computing-plus ⓘ 4 vCPUs   16 GB   c3ne.xlarge.4 System Disk High I/O 100 GB x 1 Data Disk High I/O 100 GB x 1	<input type="button" value="-"/> 3 <input type="button" value="+"/>
	Analysis Task ⓘ	Pay-per-use		<input type="button" value="+"/>

Pay-per-use Node Price **¥8.868/hour**  
This price is an estimate and may differ from the final price. [Pricing details](#)

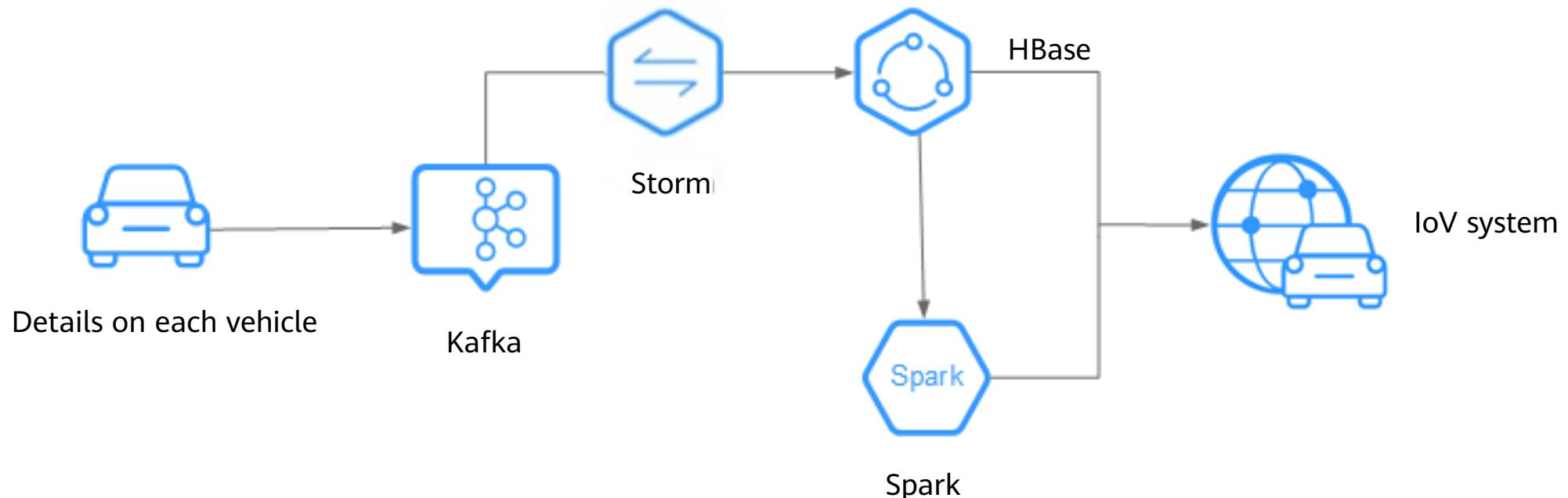
# Application Scenarios of MRS (1)

- Offline analysis of massive volumes of data
  - Low cost: OBS offers cost-effective storage.
  - Mass data analysis: Hive analyzes TB/PB-scale data.
  - Visualized data import and export tool: Loader exports data to Data Warehouse Service (DWS) for business intelligence (BI) analysis.



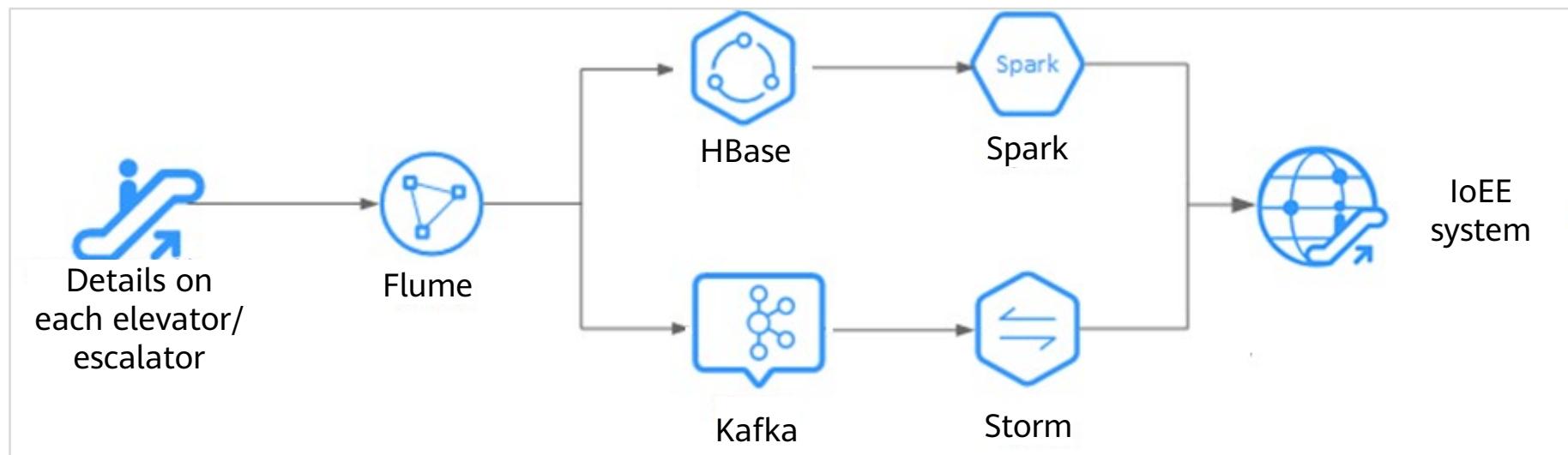
# Application Scenarios of MRS (2)

- Large-scale data storage
  - Real time: Kafka accesses massive amounts of vehicle messages in real time.
  - Massive data storage: HBase stores massive volumes of data and supports data queries in milliseconds.
  - Distributed data query: Spark analyzes and queries massive volumes of data.



# Application Scenarios of MRS (3)

- Low-latency real-time data analysis
  - Real-time data ingestion: Flume implements real-time data ingestion and provides various data collection and storage access methods.
  - Data source access: Kafka accesses data of tens of thousands of elevators and escalators in real time.



# Summary

---

- This chapter describes the opportunities and challenges in the big data era and Huawei Kunpeng big data solution. In this booming big data era, every business is a data business. On the one hand, big data analytics technologies have been fully applied in a wealth of fields, such as finance, education, government and public security, transportation planning, and clean energy. On the other hand, the development of big data also faces many challenges. To address these challenges, Huawei proposes the Kunpeng strategy. Based on Huawei Kunpeng processors and TaiShan servers, Huawei continuously improves the computing power and develops the Huawei Kunpeng computing industry. Based on this, Huawei develops the Huawei Kunpeng ecosystem. In the big data field, particularly, Huawei proposes multiple public cloud services to help partners complete intelligent transformation faster and better.

# Recommendations

---

- Huawei Cloud Official Web Link:
  - <https://www.huaweicloud.com/intl/en-us/>
- Huawei MRS Documentation:
  - <https://www.huaweicloud.com/intl/en-us/product/mrs.html>
- Huawei TALENT ONLINE:
  - <https://e.huawei.com/en/talent/#/>
- OpenEuler community:
  - <https://openeuler.org/en/>

# Thank you.

把数字世界带入每个人、每个家庭、  
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and  
organization for a fully connected,  
intelligent world.

Copyright©2020 Huawei Technologies Co., Ltd.  
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.



# Revision Record

Do Not Print this Page

Course Code	Product	Product Version	Course Version
H13-711	MRS		V3.0

Author/ID	Date	Reviewer/ID	New/ Update
Wang Mengde/wwx711842	2020.03.12	Huang Haoyang/hwx690472	Update
Wang Mengde/wwx711842	2020.08.26	Chen Ao/cwx860206	New

# Chapter 2 HDFS and ZooKeeper



# Foreword

---

- This course describes the big data distributed storage system HDFS and the ZooKeeper distributed service framework that resolves some frequently-encountered data management problems in distributed services. This chapter lays a solid foundation for subsequent component learning.

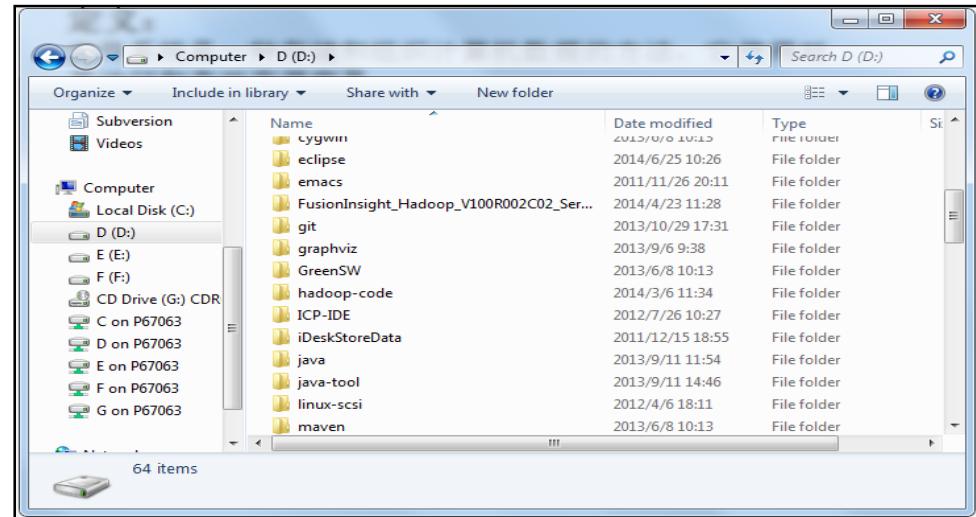
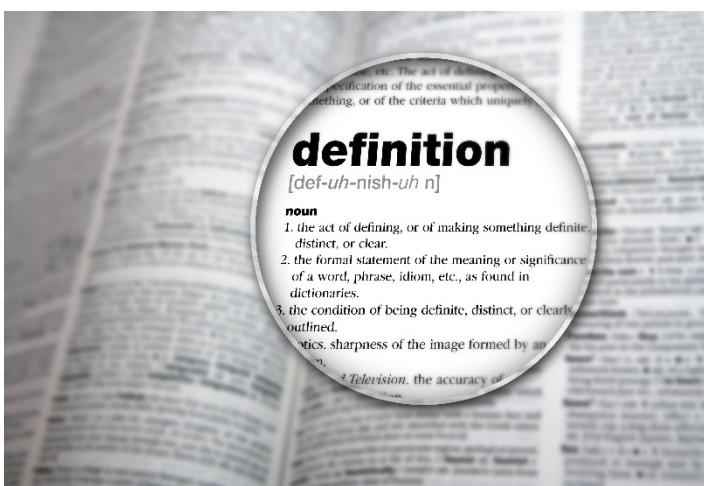
# Objectives

- Upon completion of this course, you will be able to learn:
  - HDFS-related concepts
  - HDFS application scenarios
  - HDFS system architecture
  - HDFS Key features
  - ZooKeeper-related concepts
  - ZooKeeper usage

# Contents

- 1. HDFS Overview and Application Scenarios**
2. HDFS-related Concepts
3. HDFS Architecture
4. Key Features
5. HDFS Data Read/Write Process
6. ZooKeeper Overview
7. ZooKeeper Architecture

# Dictionary and File System

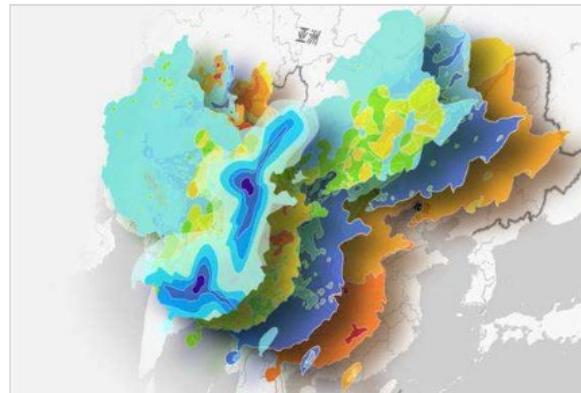


Dictionary	File System
Radical Index (1) Radical directory (2) Word index (3) Stroke index for Rare characters	File name Metadata
Dictionary body	Data block

# HDFS Overview

- Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware.
- HDFS has a high fault tolerance capability and is deployed on cost-effective hardware.
- HDFS provides high-throughput access to application data and applies to applications with large data sets.
- HDFS loses some Portable Operating System Interface of UNIX (POSIX) requirements to implement streaming access to file system data.
- HDFS was originally built as the foundation for the Apache Nutch Web search engine project.
- HDFS is a part of the Apache Hadoop Core project.

# HDFS Application Scenario Example

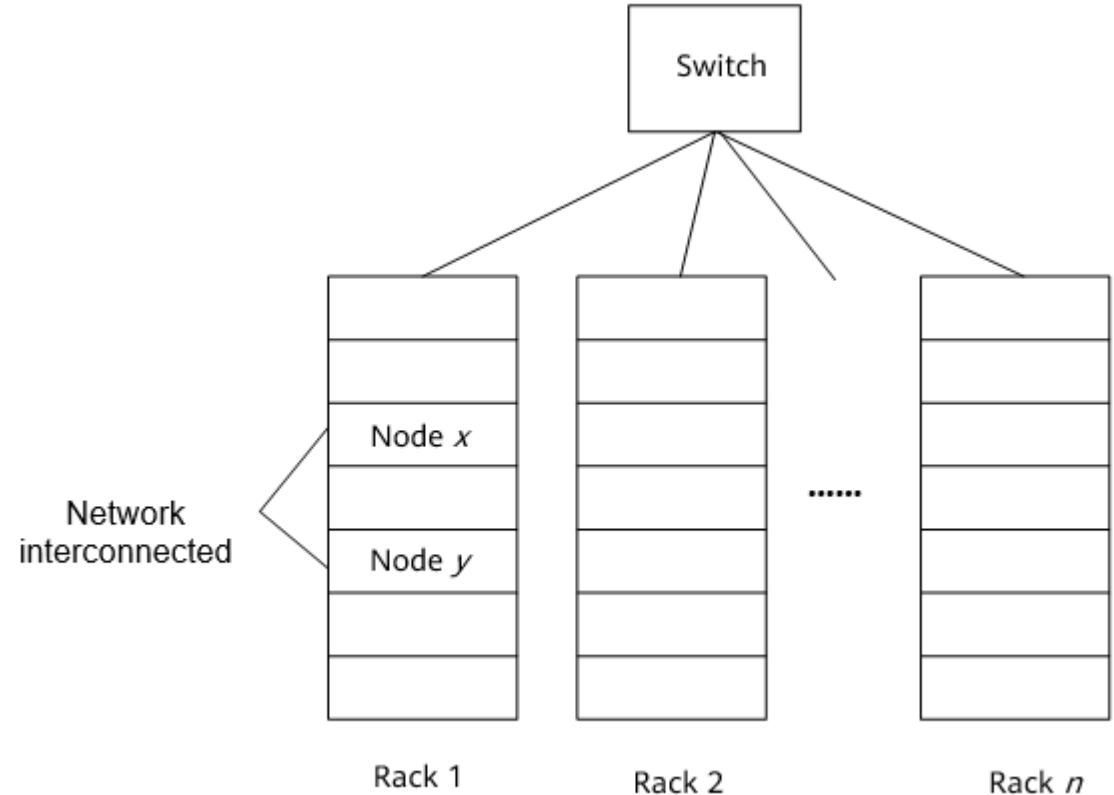


# Contents

1. HDFS Overview and Application Scenarios
- 2. HDFS-related Concepts**
3. HDFS Architecture
4. Key Features
5. HDFS Data Read/Write Process
6. ZooKeeper Overview
7. ZooKeeper Architecture

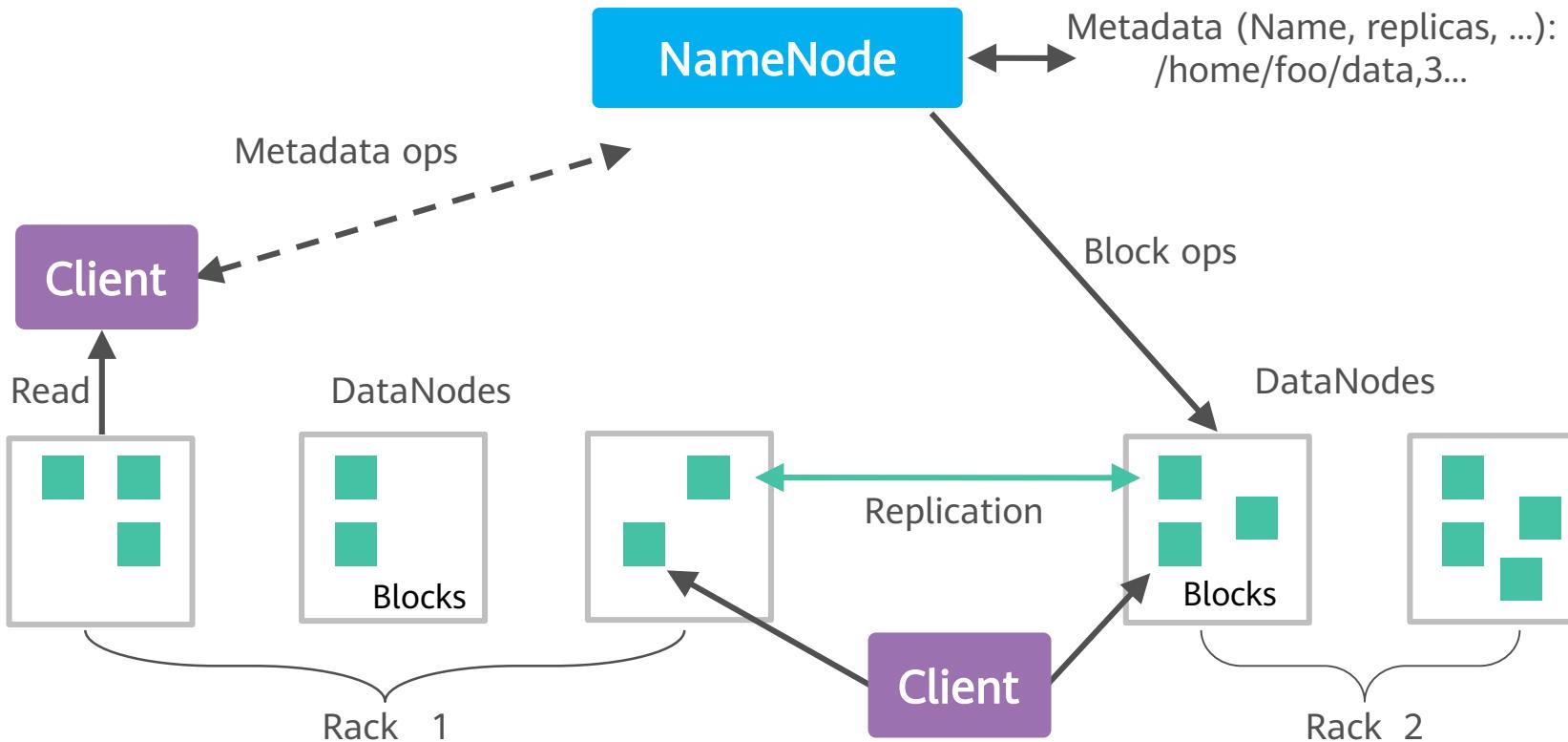
# Computer Cluster Structure

- The distributed file system stores files on multiple computer nodes. Thousands of computer nodes form a computer cluster.
- Currently, the computer cluster used by the distributed file system consists of common hardware, which greatly reduces the hardware overhead.



# Basic System Architecture

## HDFS Architecture



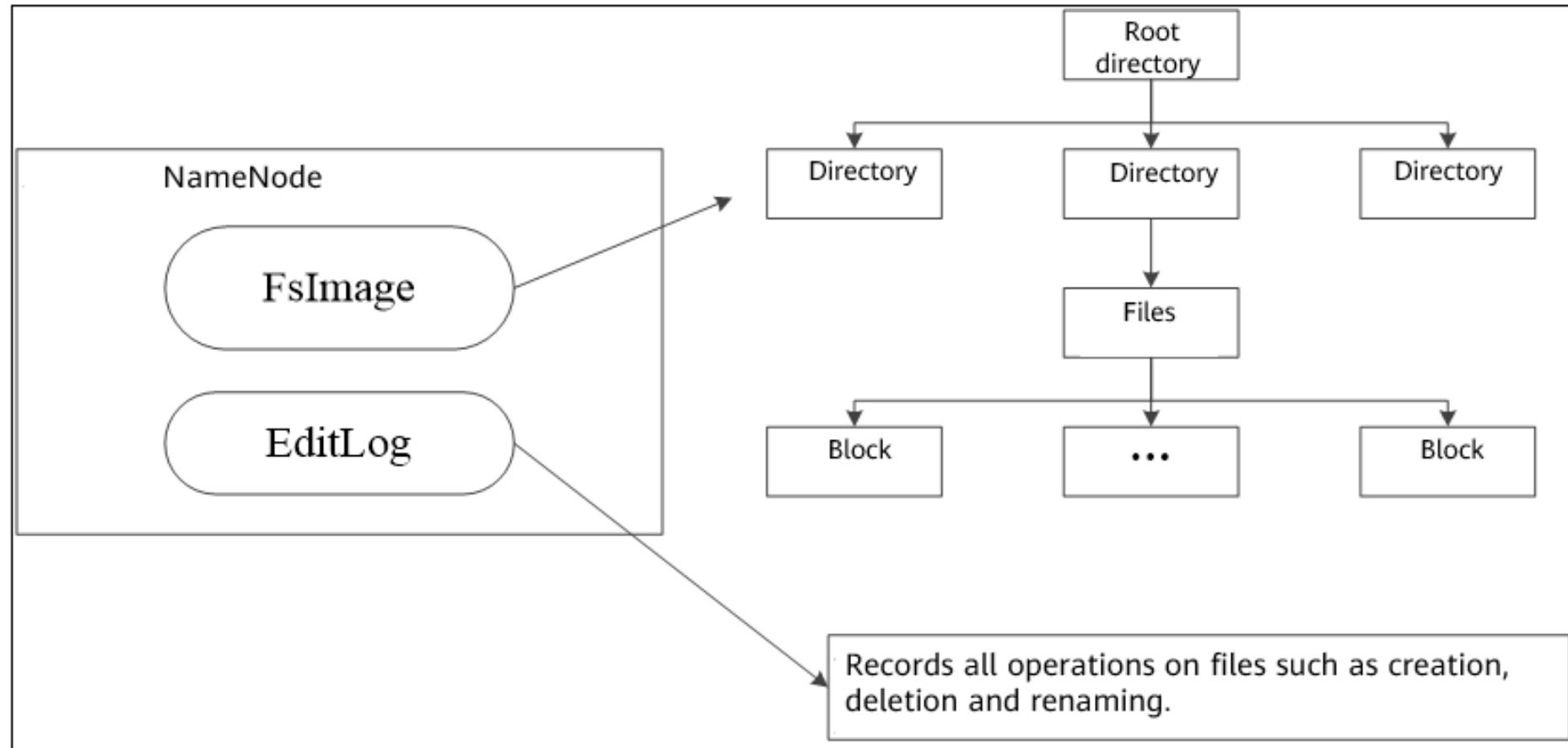
# Block

- The default size of an HDFS block is 128 MB. A file is divided into multiple blocks, which are used as the storage unit.
- The block size is much larger than that of a common file system, minimizing the addressing overhead.
- The abstract block concept brings the following obvious benefits:
  - Supporting large-scale file storage
  - Simplifying system design
  - Applicable to data backup

# NameNode and DataNode (1)

NameNode	DataNode
Stores metadata.	Stores file content.
Metadata is stored in the memory.	The file content is stored in the disk.
Saves the mapping between files, blocks, and DataNodes.	Maintains the mapping between block IDs and local files on DataNodes.

# NameNode and DataNode (2)



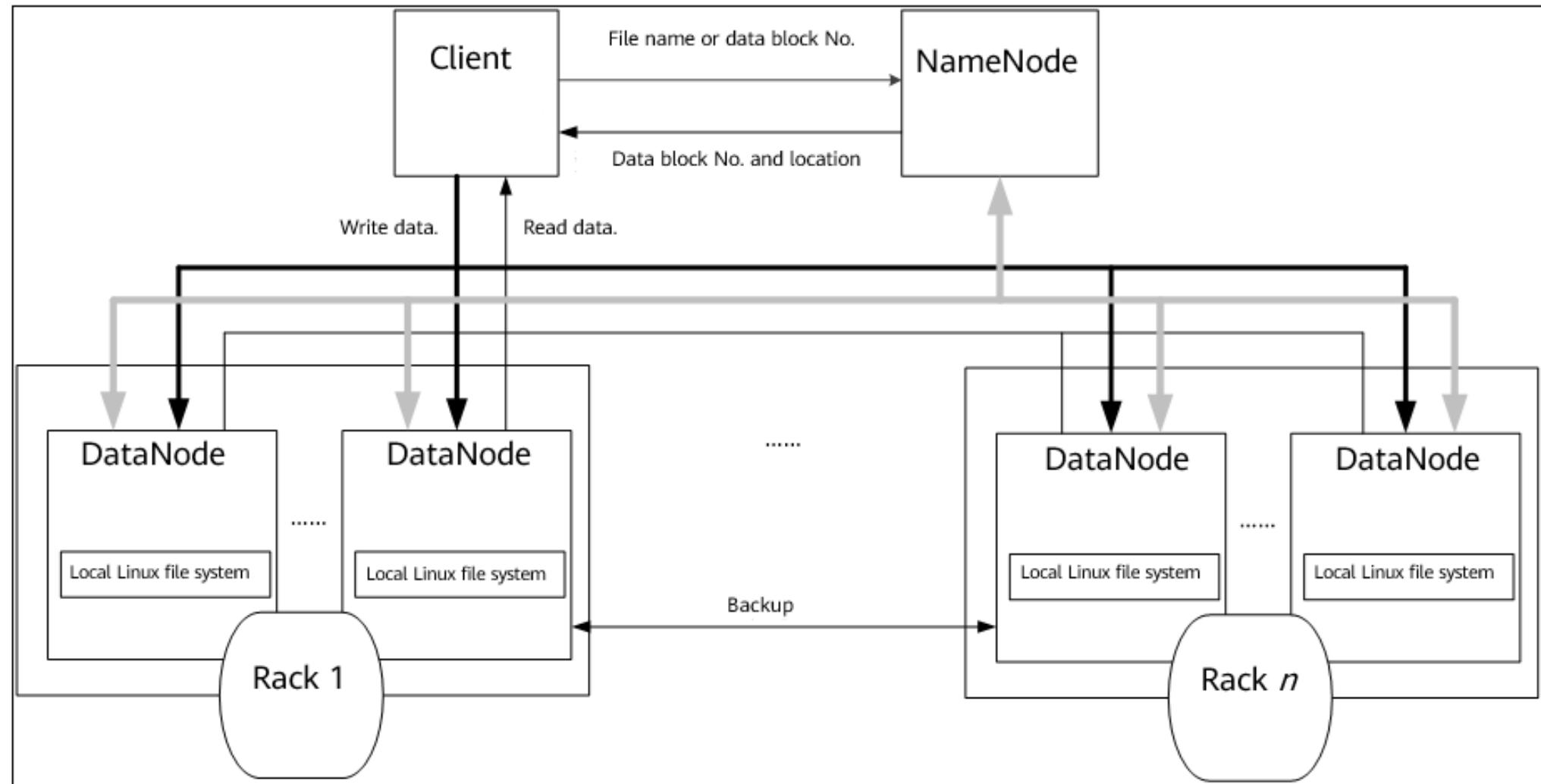
# DataNodes

- DataNodes are working nodes that store and read data in HDFS. DataNodes store and retrieve data based on the scheduling of the clients or NameNodes, and periodically send the list of stored blocks to the NameNodes.
- Data on each DataNode is stored in the local Linux file system of the node.

# Contents

1. HDFS Overview and Application Scenarios
2. HDFS-related Concepts
- 3. HDFS Architecture**
4. Key Features
5. HDFS Data Read/Write Process
6. ZooKeeper Overview
7. ZooKeeper Architecture

# HDFS Architecture Overview



# HDFS Namespace Management

- The HDFS namespace contains directories, files, and blocks.
- HDFS uses the traditional hierarchical file system. Therefore, users can create and delete directories and files, move files between directories, and rename files in the same way as using a common file system.
- NameNode maintains the file system namespace. Any changes to the file system namespace or its properties are recorded by the NameNode.

# Communication Protocol

- HDFS is a distributed file system deployed on a cluster. Therefore, a large amount of data needs to be transmitted over the network.
  - All HDFS communication protocols are based on the TCP/IP protocol.
  - The client initiates a TCP connection to the NameNode through a configurable port and uses the client protocol to interact with the NameNode.
  - The NameNode and the DataNode interact with each other by using the DataNode protocol.
  - The interaction between the client and the DataNode is implemented through the Remote Procedure Call (RPC). In design, the NameNode does not initiate an RPC request, but responds to RPC requests from the client and DataNode.

# Client

- The client is the most commonly used method for users to operate HDFS. HDFS provides a client during deployment.
- The HDFS client is a library that contains HDFS file system interfaces that hide most of the complexity of HDFS implementation.
- Strictly speaking, the client is not a part of HDFS.
- The client supports common operations such as opening, reading, and writing, and provides a command line mode similar to Shell to access data in HDFS.
- HDFS also provides Java APIs as client programming interfaces for applications to access the file system.

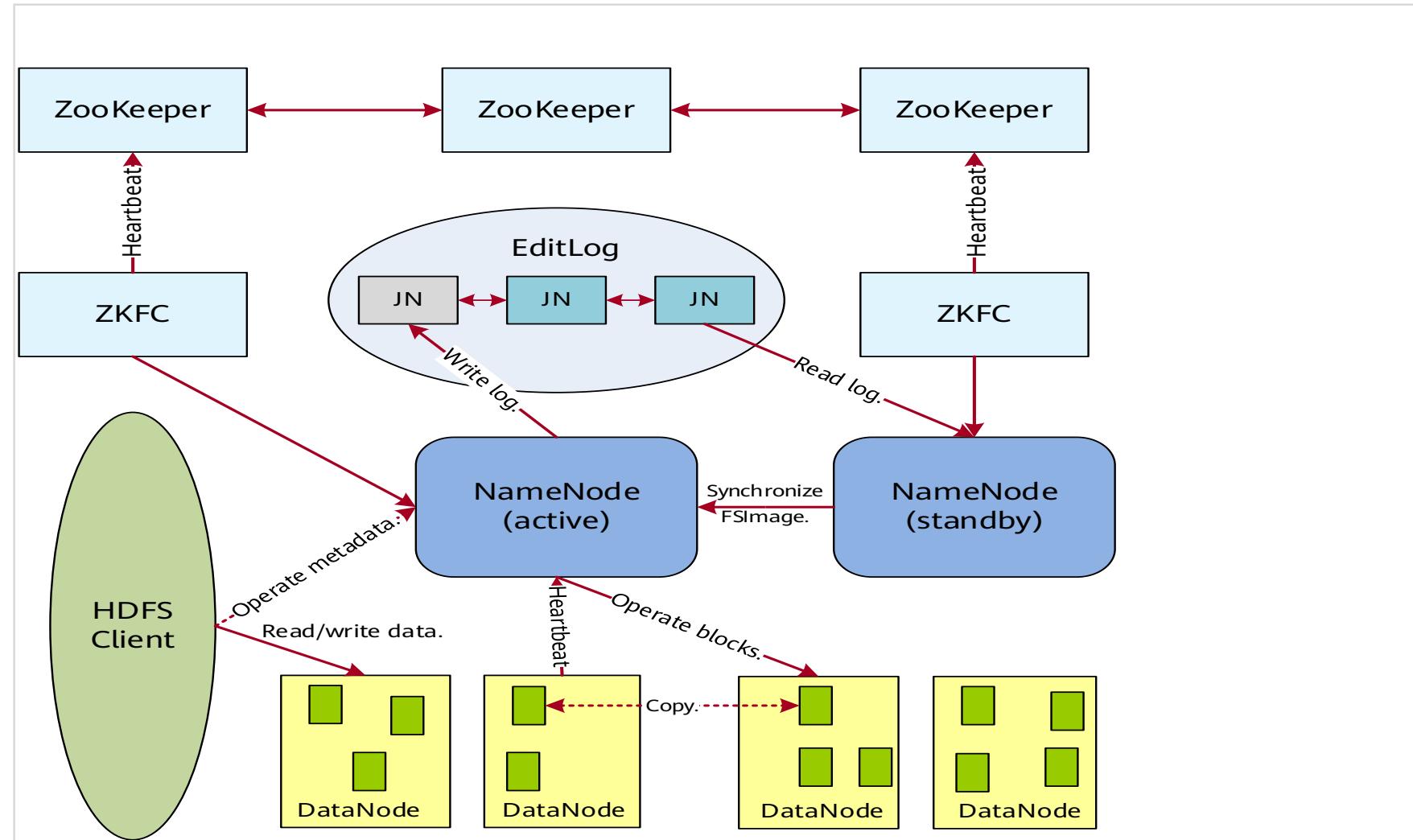
# Disadvantages of the HDFS Single-NameNode Architecture

- Only one NameNode is set for HDFS, which greatly simplifies the system design but also brings some obvious limitations. The details are as follows:
  - **Namespace limitation:** NameNodes are stored in the memory. Therefore, the number of objects (files and blocks) that can be contained in a NameNode is limited by the memory size.
  - **Performance bottleneck:** The throughput of the entire distributed file system is limited by the throughput of a single NameNode.
  - **Isolation:** Because there is only one NameNode and one namespace in the cluster, different applications cannot be isolated.
  - **Cluster availability:** Once the only NameNode is faulty, the entire cluster becomes unavailable.

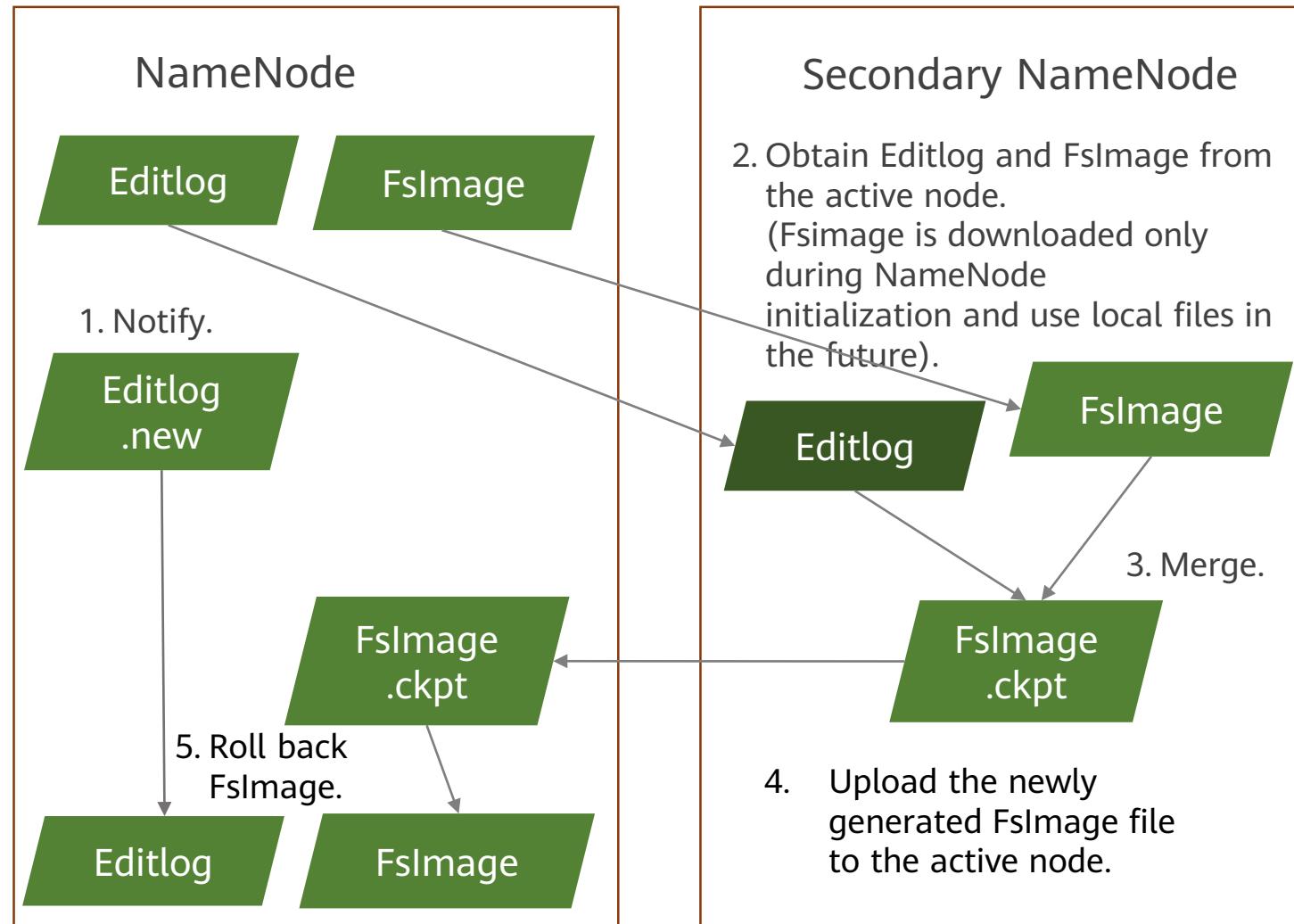
# Contents

1. HDFS Overview and Application Scenarios
2. HDFS-related Concepts
3. HDFS Architecture
- 4. Key Features**
5. HDFS Data Read/Write Process
6. ZooKeeper Overview
7. ZooKeeper Architecture

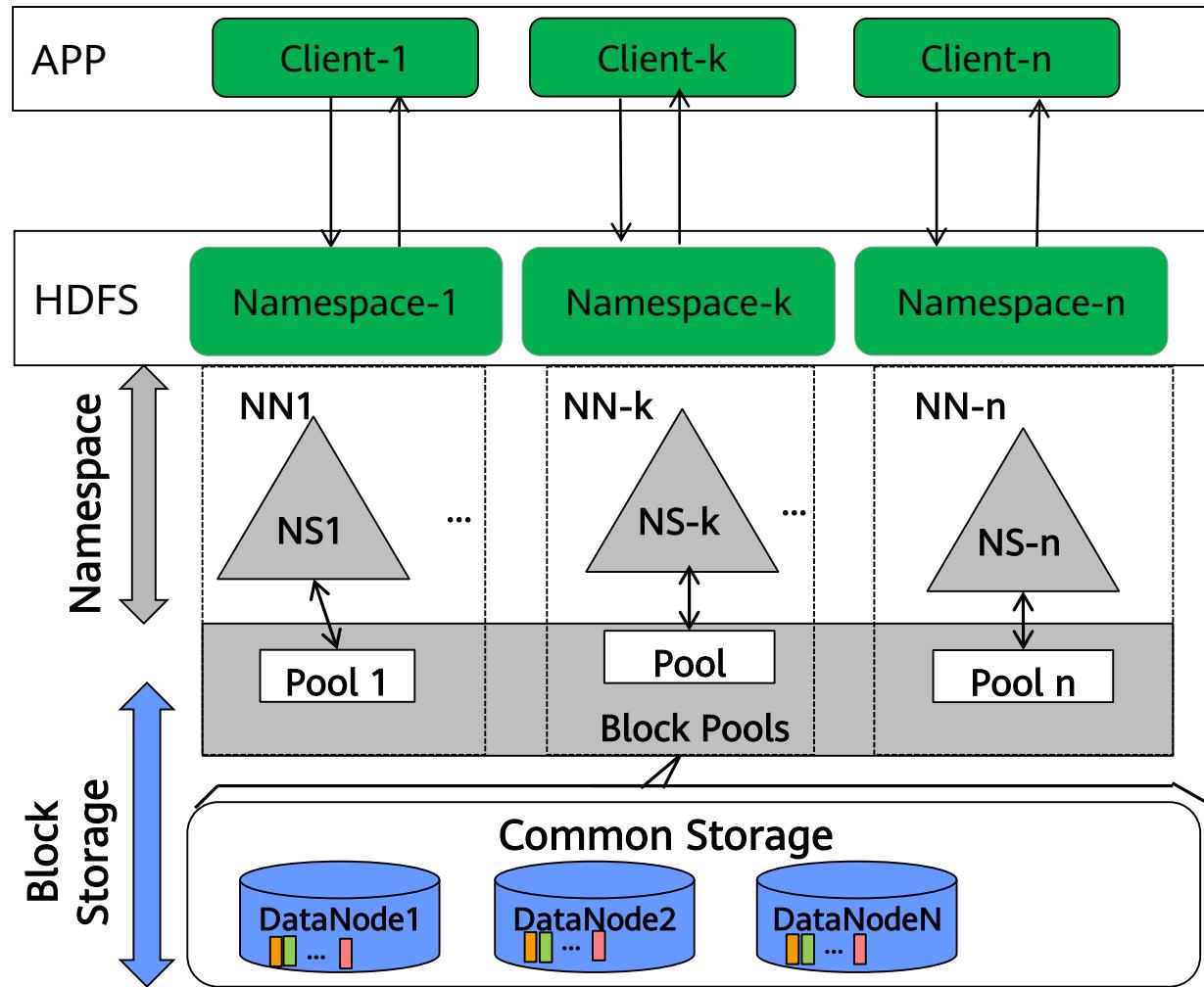
# HDFS High Availability (HA)



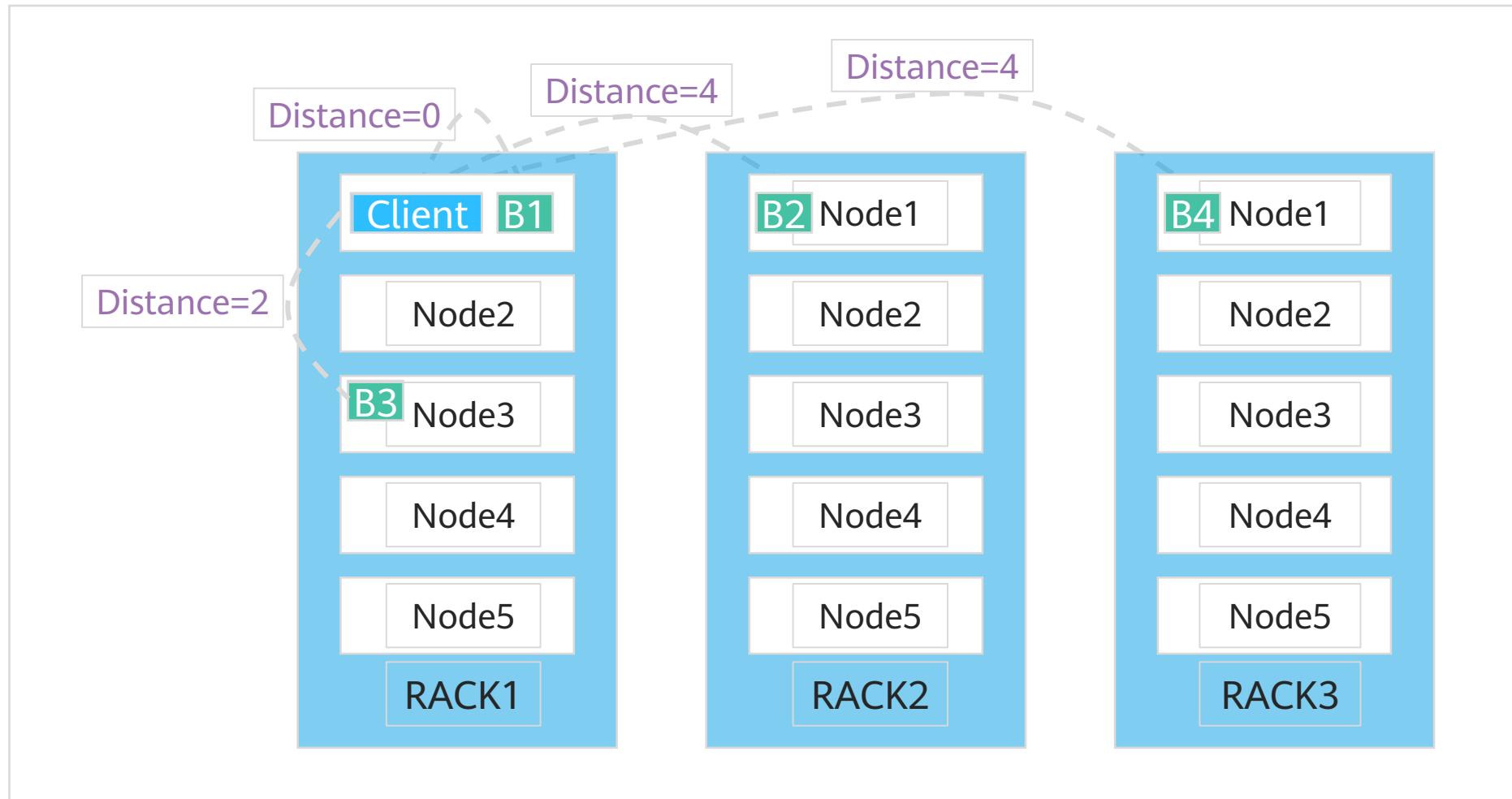
# Metadata Persistence



# HDFS Federation



# Data Replica Mechanism



# HDFS Data Integrity Assurance

- HDFS aims to ensure the integrity of storage data and ensures the reliability of components.
- Rebuilding the replica data of failed data disks
  - When the DataNode fails to report data to the NameNode periodically, the NameNode initiates the replica rebuilding action to restore the lost replicas.
- Cluster data balancing:
  - The HDFS architecture designs the data balancing mechanism, which ensures that data is evenly distributed on each DataNode.
- Metadata reliability:
  - The log mechanism is used to operate metadata, and metadata is stored on the active and standby NameNodes.
  - The snapshot mechanism implements the common snapshot mechanism of file systems, ensuring that data can be restored in a timely manner in the case of mis-operations.
- Security mode:
  - HDFS provides a unique security mode mechanism to prevent faults from spreading when DataNodes or disks are faulty.

# Other Key Design Points of the HDFS Architecture

- Space reclamation mechanism:
  - Supports the recycle bin mechanism and dynamic setting of the number of copies.
- Data organization:
  - Data is stored by data block in the HDFS of the operating system.
- Access mode:
  - Provides HDFS data accessing through Java APIs, HTTP or SHELL modes.

# Common Shell Commands

Type	Command	Description
dfs	-cat	Displays file content.
	-ls	Displays the directory list.
	-rm	Delete a file.
	-put	Uploads the directory or file to HDFS.
	-get	Downloads directories or files to the local host from HDFS.
	-mkdir	Creates a directory.
	-chmod/-chown	Changes the group of the file.
	...	...
dfsadmin	-safemode	Performs security mode operations.
	-report	Reports service status.

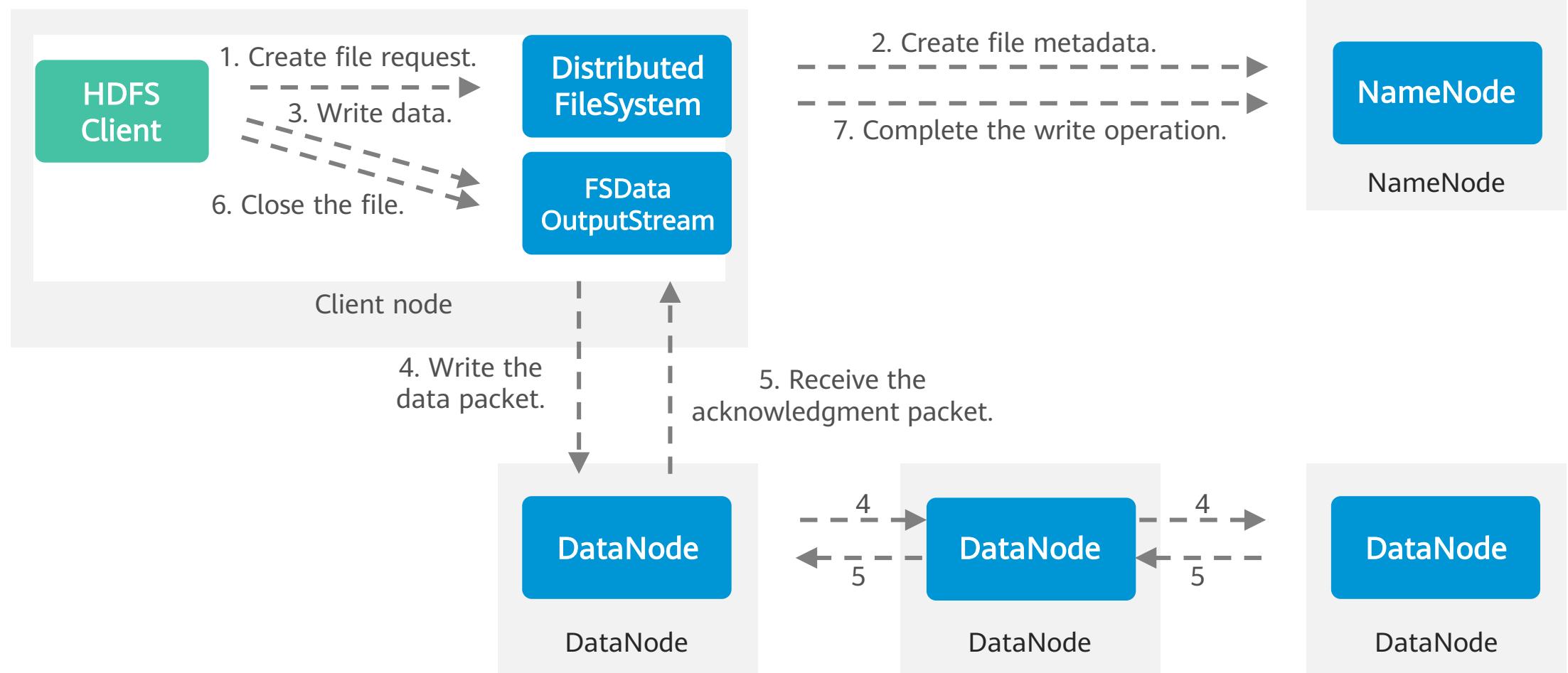
# New Features of HDFS 3.0

- Erasure Code (EC) in HDFS is supported.
- Union based on the HDFS router is supported.
- Multiple NameNodes are supported.
- Disk balancers are added to DataNodes for load balancing.

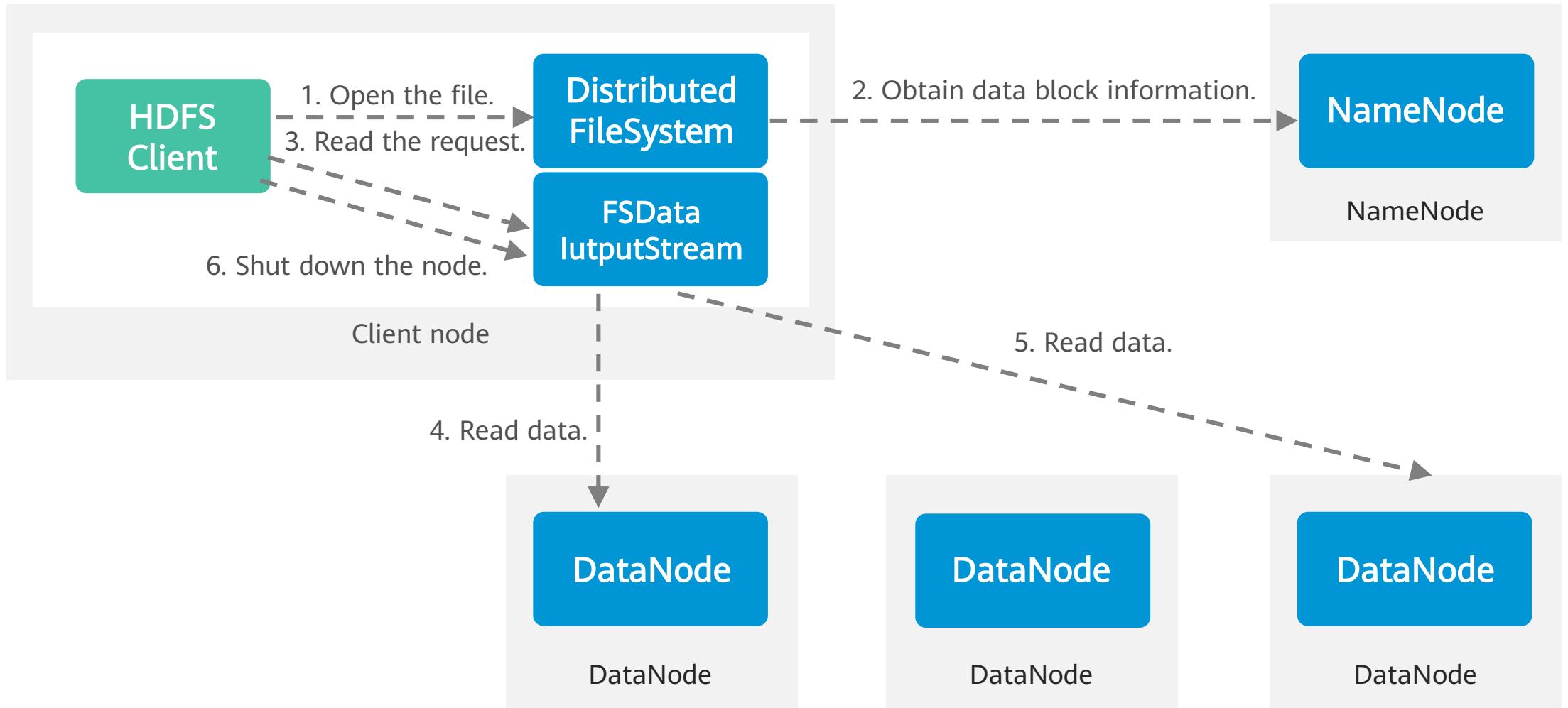
# Contents

1. HDFS Overview and Application Scenarios
2. HDFS-related Concepts
3. HDFS Architecture
4. Key Features
- 5. HDFS Data Read/Write Process**
6. ZooKeeper Overview
7. ZooKeeper Architecture

# HDFS Data Write Process



# HDFS Data Read Process



# Contents

1. HDFS Overview and Application Scenarios
2. HDFS-related Concepts
3. HDFS Architecture
4. Key Features
5. HDFS Data Read/Write Process
- 6. ZooKeeper Overview**
7. ZooKeeper Architecture

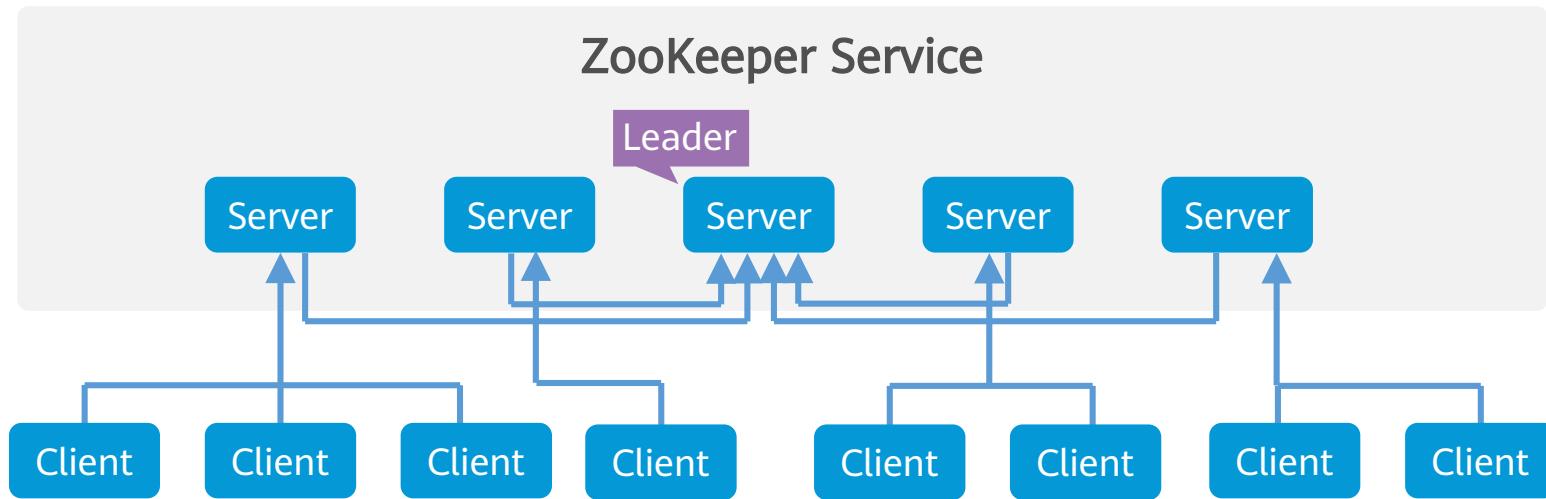
# ZooKeeper Overview

- The ZooKeeper distributed service framework is used to solve some data management problems that are frequently encountered in distributed applications and provide distributed and highly available coordination service capabilities.
- In security mode, ZooKeeper depends on Kerberos and LdapServer for security authentication, but in non-security mode, ZooKeeper does not depend on them any more. As a bottom-layer component, ZooKeeper is widely used and depended by upper-layer components, such as Kafka, HDFS, HBase and Storm.

# Contents

1. HDFS Overview and Application Scenarios
2. HDFS-related Concepts
3. HDFS Architecture
4. Key Features
5. HDFS Data Read/Write Process
6. ZooKeeper Overview
- 7. ZooKeeper Architecture**

# ZooKeeper Service Architecture - Model



- The ZooKeeper cluster consists of a group of server nodes. In this group, there is only one leader node, and other nodes are followers.
- The leader is elected during the startup.
- ZooKeeper uses the custom atomic message protocol to ensure data consistency among nodes in the entire system.
- After receiving a data change request, the leader node writes the data to the disk and then to the memory.

# ZooKeeper Service Architecture - DR Capability

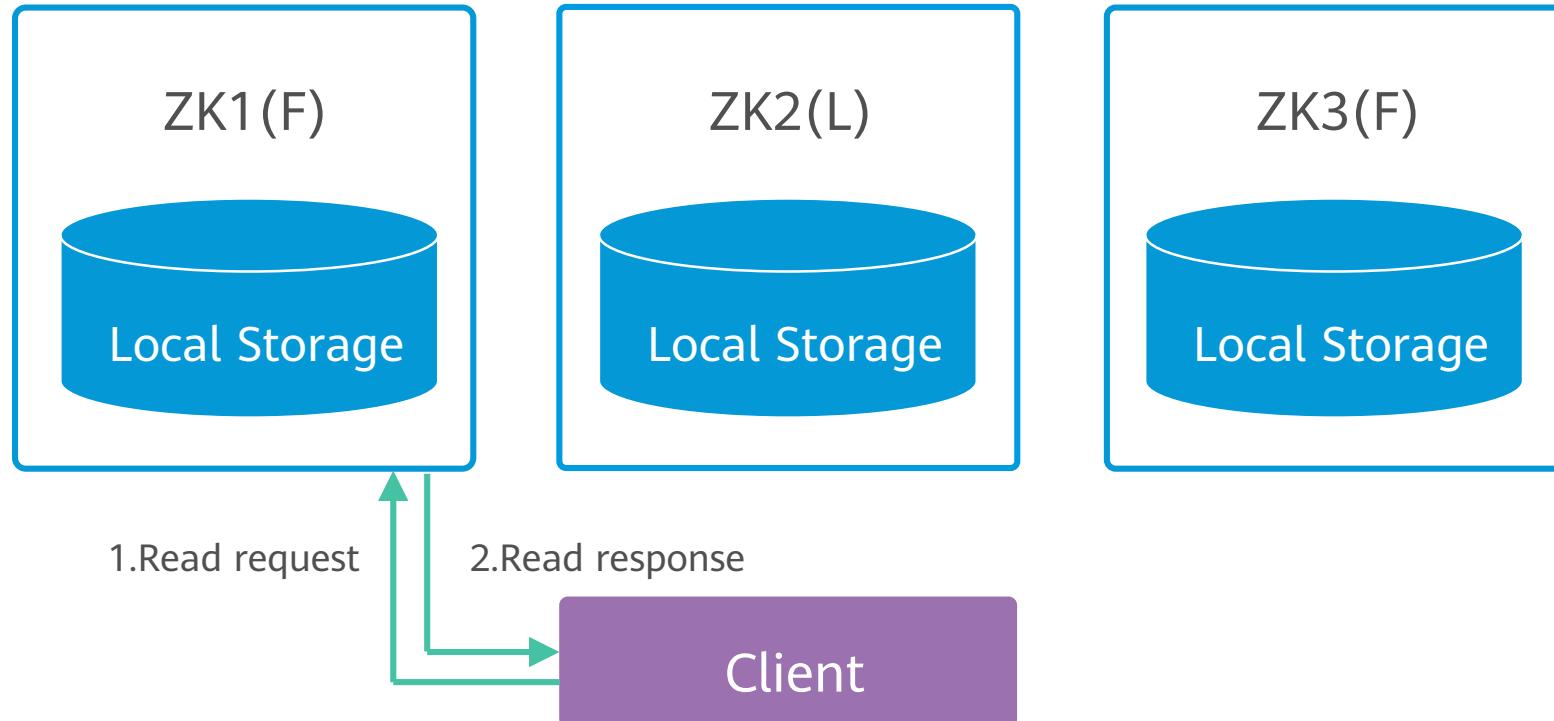
- If the ZooKeeper can complete the election, it can provide services for external systems.
  - During ZooKeeper election, if an instance obtains more than half of the votes, the instance becomes the leader.
- For a service with  $n$  instances,  $n$  may be an odd or even number.
  - When  $n$  is an odd number, assume:  $n = 2x + 1$ ; then the node needs to obtain  $x+1$  votes to become the leader, and the DR capability is  $x$ .
  - When  $n$  is an even number, that:  $n = 2x + 2$ ; then the node needs to obtain  $x+2$  (more than half) votes to become the leader, and the DR capability is  $x$ .

# Key Features of ZooKeeper

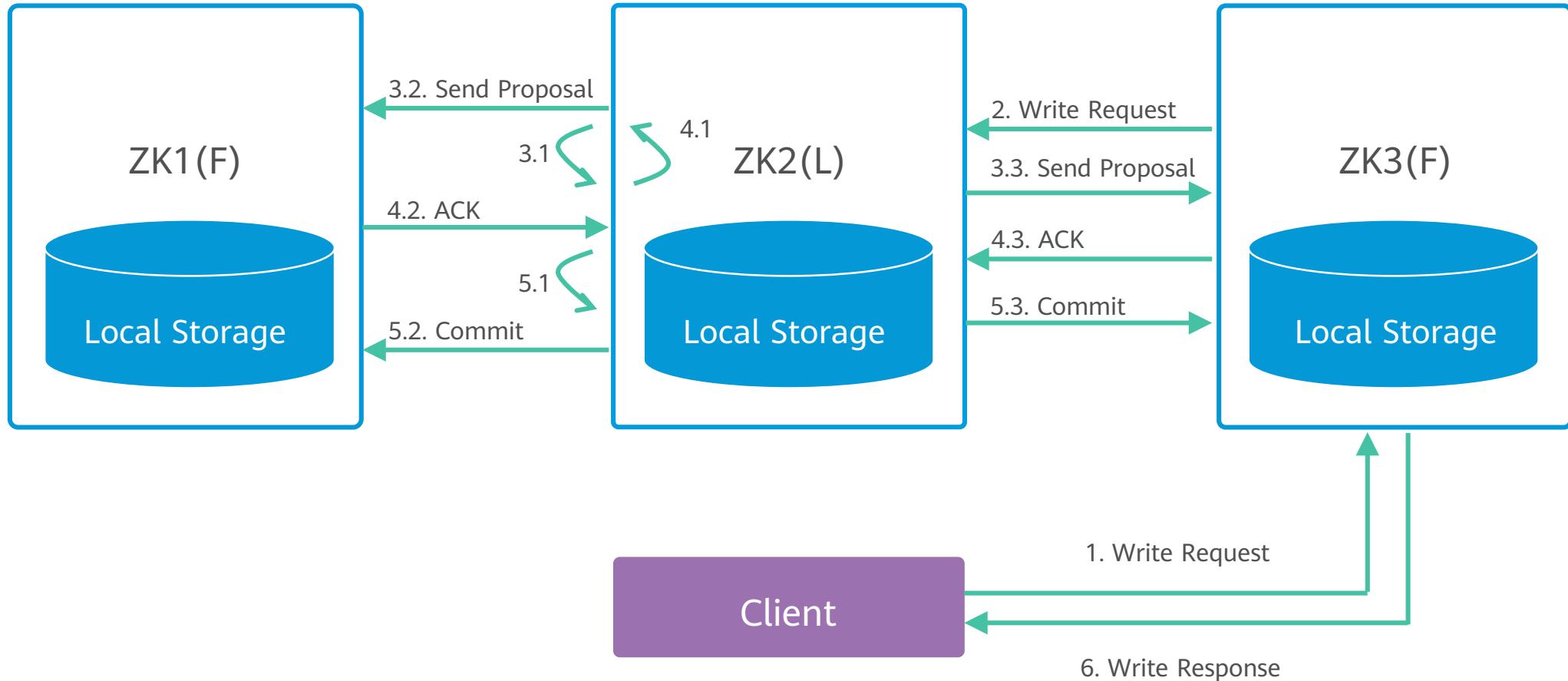
- Eventual consistency: All servers are displayed in the same view.
- Real-time capability: Clients can obtain server updates and failures within a specified period of time.
- Reliability: A message will be received by all servers.
- Wait-free: Slow or faulty clients cannot intervene the requests of rapid clients so that the requests of each client can be processed effectively.
- Atomicity: Data update either succeeds or fails. There are no intermediate states.
- Sequence consistency: Updates sent by the client are applied in the sequence in which they are sent.

# Read Function of ZooKeeper

- According to the consistency of ZooKeeper, the client obtains the same view regardless of the server connected to the client. Therefore, read operations can be performed between the client and any node.



# Write Function of ZooKeeper



# Commands for ZooKeeper Clients

- To invoke a ZooKeeper client, run the following command:

```
zkCli.sh -server 172.16.0.1:24002
```

- To create a node: **create /node**
- To list subnodes: **ls /node**
- To create node data: **set /node data**
- To obtain node data: **get /node**
- To delete a node: **delete /node**
- To delete a node and all its subnodes: **deleteall /node**

# Summary

---

- The distributed file system is an effective solution for large-scale data storage in the big data era. The open-source HDFS implements GFS and distributed storage of massive data by using a computer cluster formed by inexpensive hardware.
- HDFS is compatible with inexpensive hardware devices, stream data read and write, large data sets, simple file models, and powerful cross-platform compatibility. However, HDFS has its own limitations. For example, it is not suitable for low-latency data access, cannot efficiently store a large number of small files, and does not support multi-user write and arbitrary file modification.
- "Block" is the core concept of HDFS. A large file is split into multiple blocks. HDFS adopts the abstract block concept, supports large-scale file storage, simplifies system design, and is suitable for data backup.
- The ZooKeeper distributed service framework is used to solve some data management problems that are frequently encountered in distributed applications and provide distributed and highly available coordination service capabilities.

# Quiz

1. Why is it recommended that the number of ZooKeepers be deployed in an odd number?
2. Why is the HDFS data block size larger than the disk block size?
3. Can HDFS data be read when it is written?

# Recommendations

---

- Huawei Cloud Official Web Link:
  - <https://www.huaweicloud.com/intl/en-us/>
- Huawei MRS Documentation:
  - <https://www.huaweicloud.com/intl/en-us/product/mrs.html>
- Huawei TALENT ONLINE:
  - <https://e.huawei.com/en/talent/#/>

Thank you.



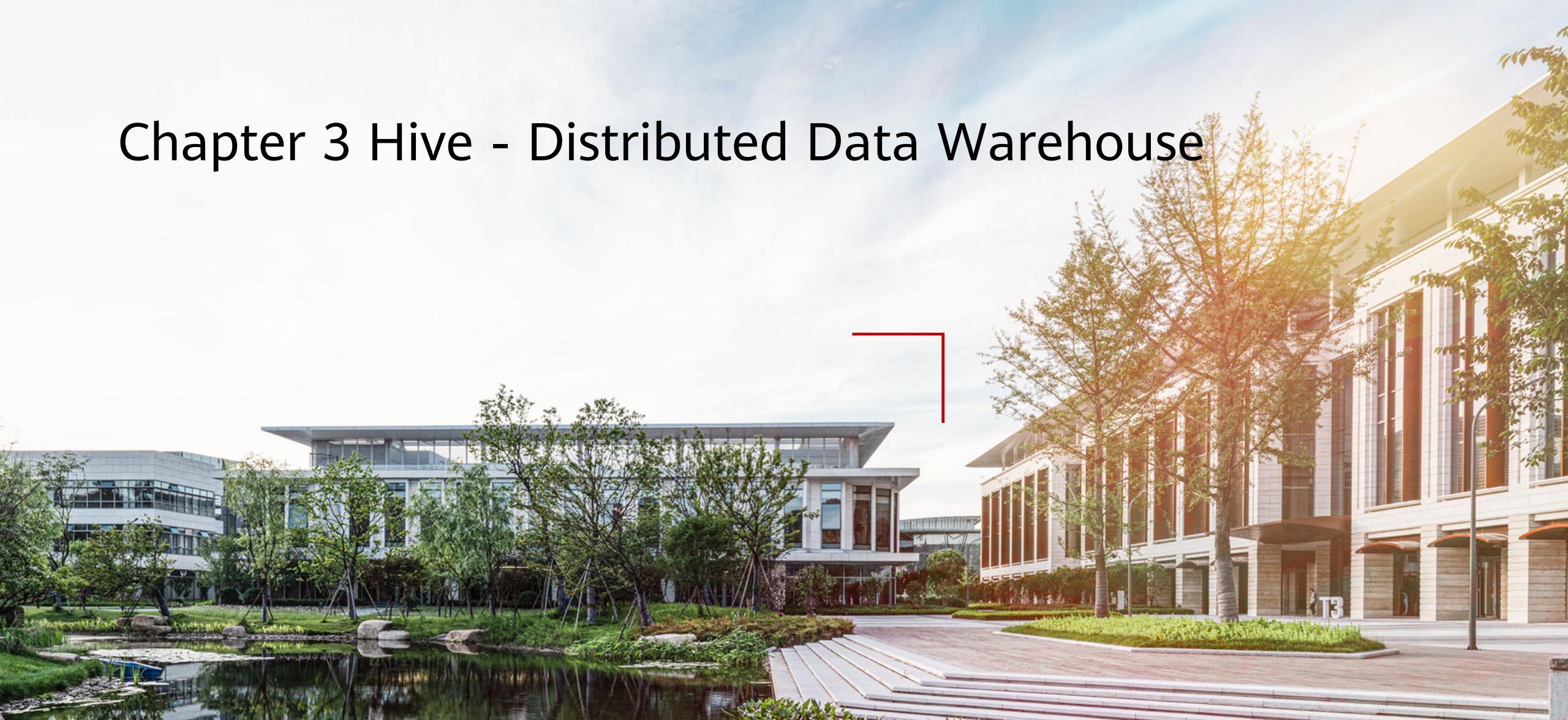
# Revision Record

Do Not Print this Page

Course Code	Product	Product Version	Course Version
H13-711	MRS		V3.0

Author/ID	Date	Reviewer/ID	New/ Update
Wang Mengde/wwx711842	2020.03.12	Huang Haoyang/hwx690472	Update
Wang Mengde/wwx711842	2020.08.29	Huang Haoyang/hwx690472	New

# Chapter 3 Hive - Distributed Data Warehouse



# Foreword

---

- The Apache Hive data warehouse software helps read, write, and manage large data sets that reside in distributed storage by using SQL. Structures can be projected onto stored data. The command line tool and JDBC driver are provided to connect users to Hive.

# Objectives

- Upon completion of this course, you will be able to learn:
  - Hive application scenarios and basic principles
  - Hive architecture and running process
  - Hive SQL statements

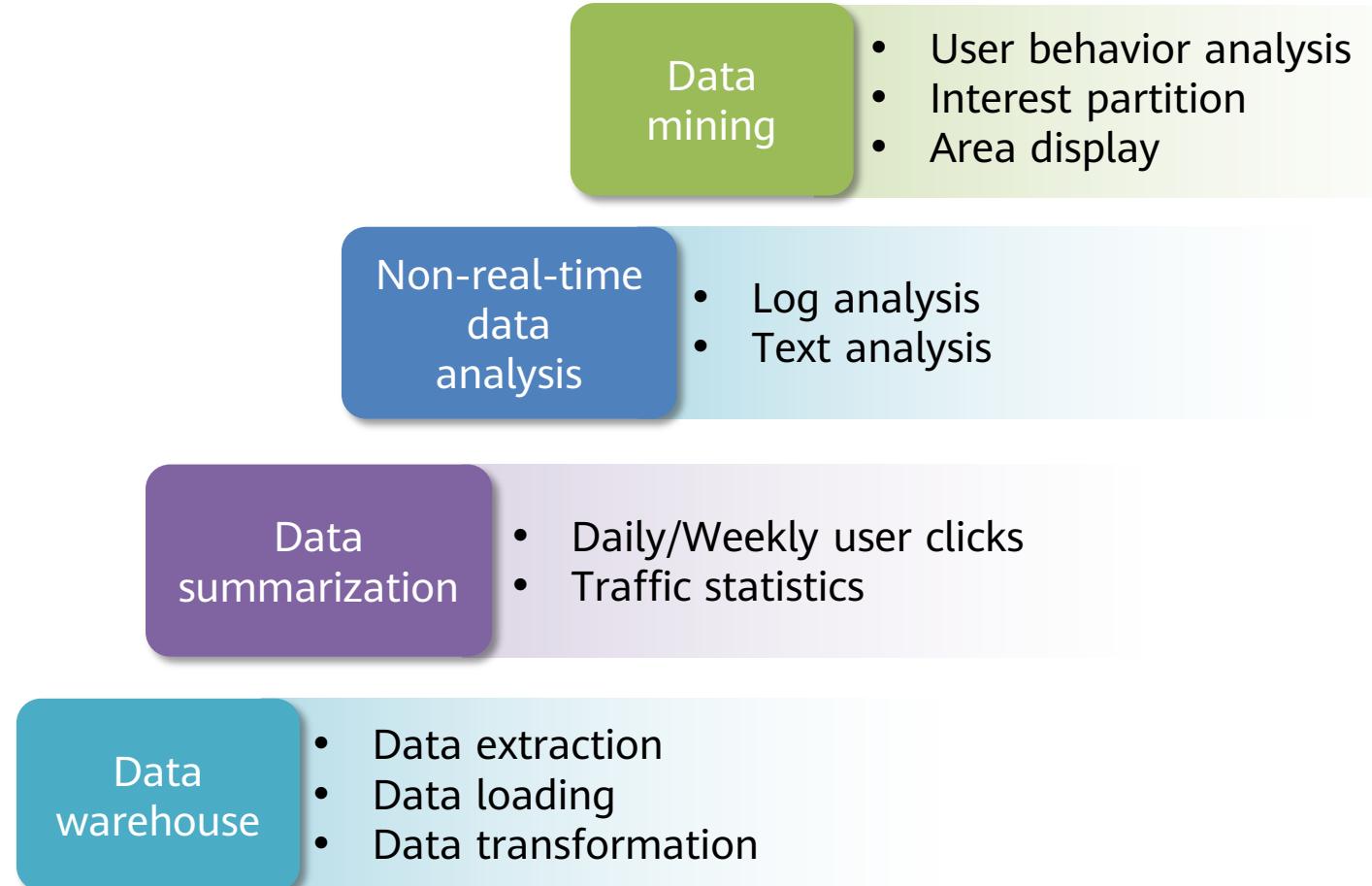
# Contents

- 1. Hive Overview**
2. Hive Functions and Architecture
3. Basic Hive Operations

# Introduction to Hive

- Hive is a data warehouse tool running on Hadoop and supports PB-level distributed data query and management.
- Hive features:
  - Supporting flexible extraction, transformation, and load (ETL)
  - Supporting multiple computing engines, such as Tez and Spark
  - Supporting direct access to HDFS files and HBase
  - Easy-to-use and easy-to-program

# Application Scenarios of Hive



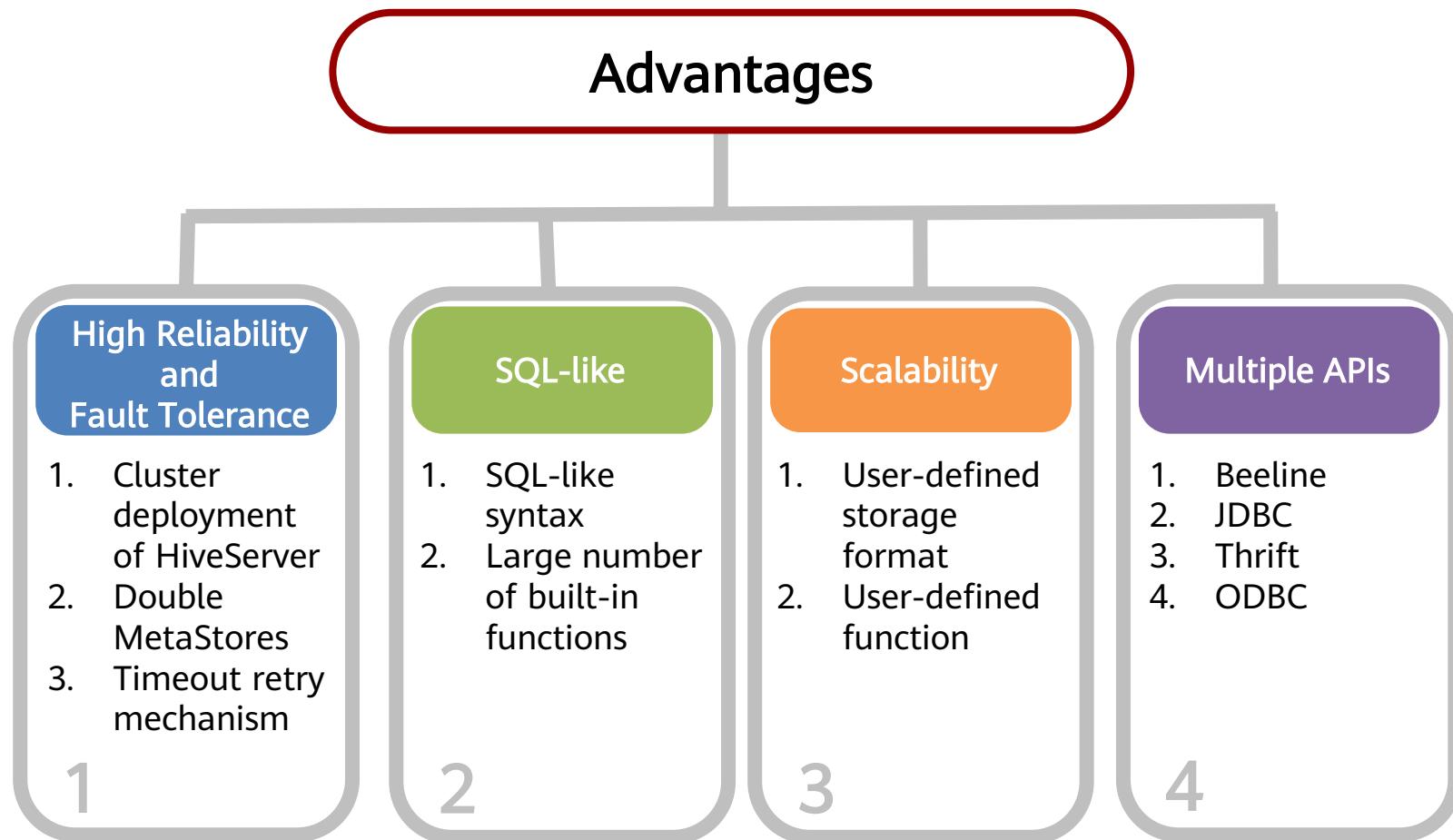
# Comparison Between Hive and Traditional Data Warehouses (1)

	Hive	Conventional Data Warehouse
Storage	HDFS is used to store data. Theoretically, infinite expansion is possible.	Clusters are used to store data, which have a capacity upper limit. With the increase of capacity, the computing speed decreases sharply. Therefore, data warehouses are applicable only to commercial applications with small data volumes.
Execution engine	Tez (default)	You can select more efficient algorithms to perform queries, or take more optimization measures to speed up the queries.
Usage Method	HQL (SQL-like)	SQL
Flexibility	Metadata storage is independent of data storage, decoupling metadata and data.	Low flexibility. Data can be used for limited purposes.
Analysis speed	Computing depends on the cluster scale and the cluster is easy to expand. In the case of a large amount of data, computing is much faster than that of a common data warehouse.	When the data volume is small, the data processing speed is high. When the data volume is large, the speed decreases sharply.

# Comparison Between Hive and Traditional Data Warehouses (2)

	Hive	Conventional Data Warehouse
<b>Index</b>	Low efficiency	High efficiency
<b>Ease of use</b>	Self-developed application models are needed, featuring high flexibility but delivering low usability.	A set of mature report solutions is integrated to facilitate data analysis.
<b>Reliability</b>	Data is stored in HDFS, implementing high data reliability and fault tolerance.	The reliability is low. If a query fails, you start the task again. The data fault tolerance depends on hardware RAID.
<b>Environment dependency</b>	Low dependency on hardware, applicable to common machines	Highly dependent on high-performance business servers
<b>Price</b>	Open-source product, free of charge	Expensive in commercial use

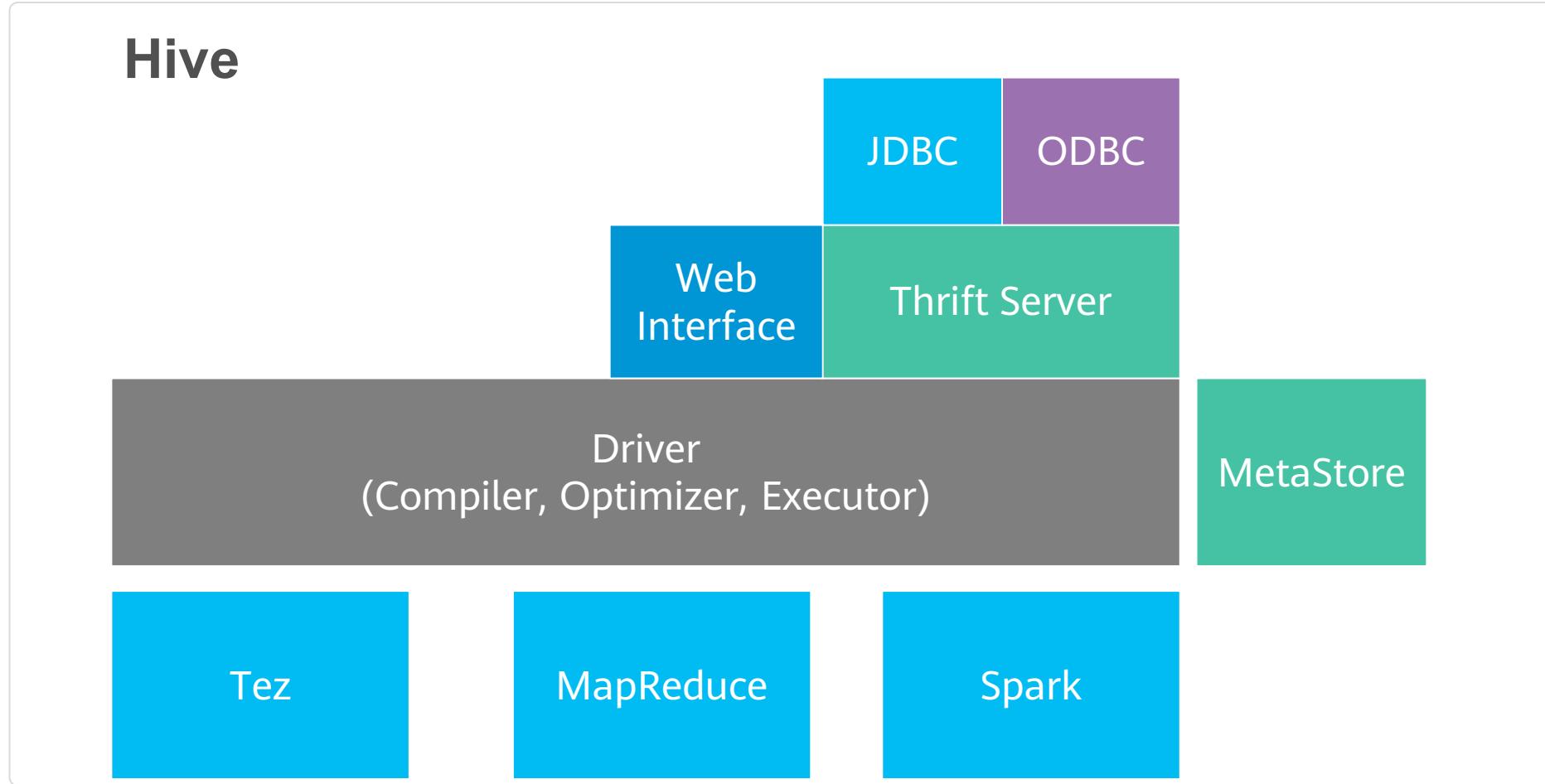
# Advantages of Hive



# Contents

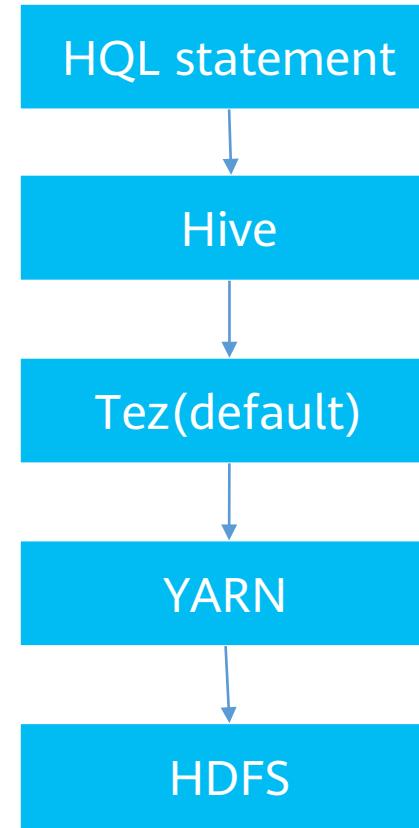
1. Hive Overview
- 2. Hive Functions and Architecture**
3. Basic Hive Operations

# Hive Architecture

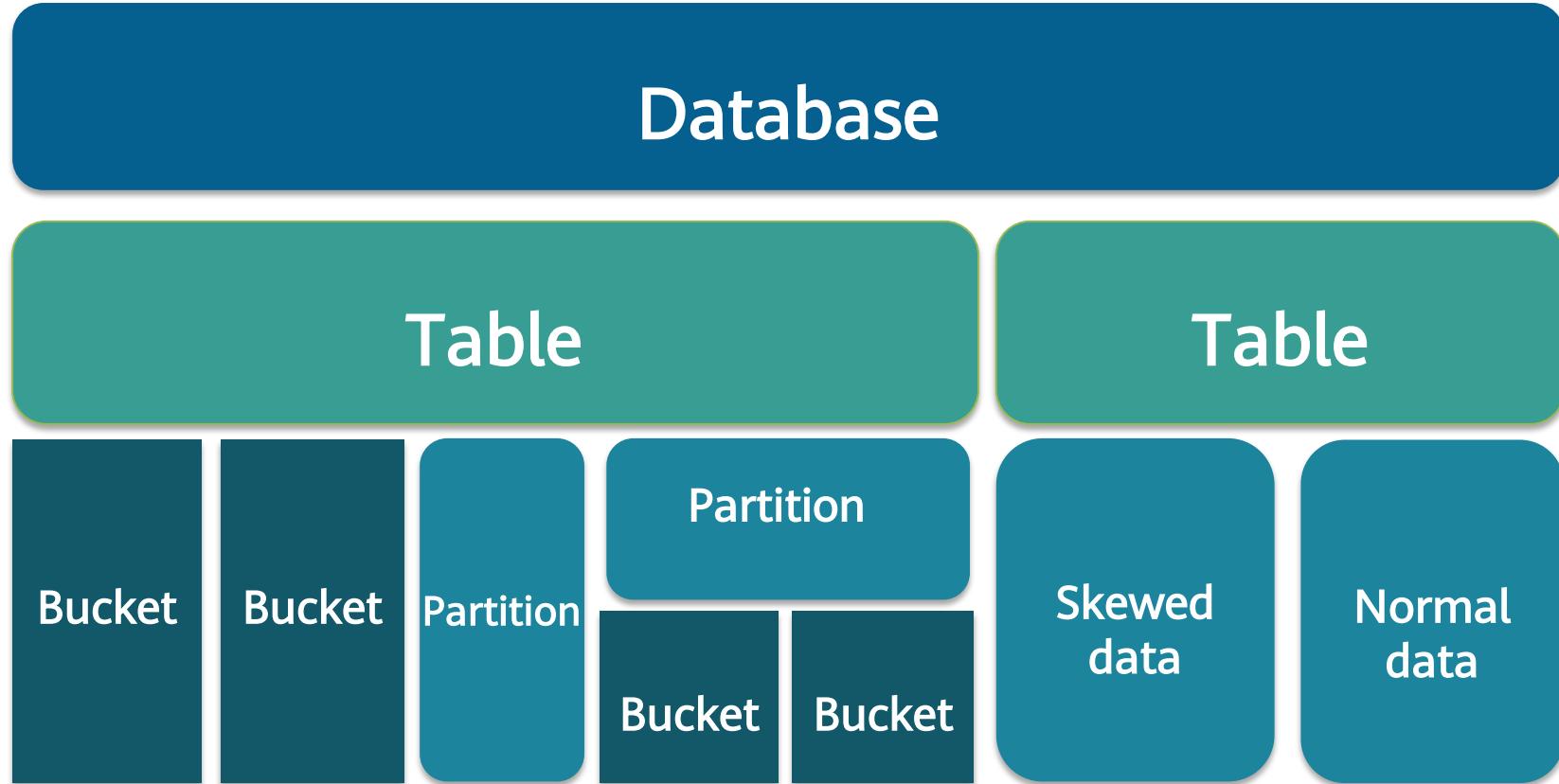


# Hive Running Process

- The client submits the HQL command.
- Tez executes the query.
- YARN allocates resources to applications in the cluster and enables authorization for Hive jobs in the YARN queue.
- Hive updates data in HDFS or Hive warehouse based on the table type.
- Hive returns the query result through the JDBC connection.



# Data Storage Model of Hive



# Partition and Bucket

- Partition: Data tables can be partitioned based on the value of a certain field.
  - Each partition is a directory.
  - The number of partitions is not fixed.
  - Partitions or buckets can be created in a partition.
- Data can be stored in different buckets.
  - Each bucket is a file.
  - The number of buckets is specified when creating a table. The buckets can be sorted.
  - Data is hashed based on the value of a field and then stored in a bucket.

# Managed Table and External Table

- Hive can create managed tables and external tables.
  - By default, a managed table is created, and Hive moves data to the data warehouse directory.
  - When an external table is created, Hive accesses data outside the warehouse directory.
  - If all processing is performed by Hive, you are advised to use managed tables.
  - If you want to use Hive and other tools to process the same data set, you are advised to use external tables.

	Managed Table	External Table
CREATE/LOAD	Data is moved to the repository directory.	The data location is not moved.
DROP	The metadata and data are deleted together.	Only the metadata is deleted.

# Functions Supported by Hive

- Built-in Hive Functions
  - Mathematical functions, such as `round()`, `floor()`, `abs()`, and `rand()`.
  - Date functions, such as `to_date()`, `month()`, and `day()`.
  - String functions, such as `trim()`, `length()`, and `substr()`.
- User-Defined Function (UDF)

# Contents

1. Hive Overview
2. Hive Functions and Architecture
- 3. Basic Hive Operations**

# Hive Usage

- Running HiveServer2 and Beeline:

```
$ $HIVE_HOME/bin/hiveserver2
```

```
$ $HIVE_HOME/bin/beeline -u jdbc:hive2://$HS2_HOST:$HS2_PORT
```

- Running Hcatalog:

```
$ $HIVE_HOME/hcatalog/sbin/hcat_server.sh
```

- Running WebHCat (Templeton):

```
$ $HIVE_HOME/hcatalog/sbin/webhcat_server.sh
```

# Hive SQL Overview

- DDL-Data Definition Language:
  - Creates tables, modifies tables, deletes tables, partitions, and data types.
- DML-Data Management Language:
  - Imports and exports data.
- DQL-Data Query Language:
  - Performs simple queries.
  - Performs complex queries such as **Group by**, **Order by** and **Join**.

# DDL Operations

-- Create a table:

```
hive> CREATE TABLE pokes (foo INT, bar STRING);
```

```
hive> CREATE TABLE invites (foo INT, bar STRING) PARTITIONED BY (ds STRING);
```

-- Browse the table:

```
hive> SHOW TABLES;
```

-- Describe a table:

```
hive> DESCRIBE invites;
```

-- Modify a table:

```
hive> ALTER TABLE events RENAME TO 3koobecaf;
hive> ALTER TABLE pokes ADD COLUMNS (new_col INT);
```

# DML Operations

-- Load data to a table:

```
hive> LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE INTO TABLE pokes;
```

```
hive> LOAD DATA LOCAL INPATH './examples/files/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');
```

-- Export data to HDFS:

```
EXPORT TABLE invites TO '/department';
```

# DQL Operations (1)

## --SELECTS and FILTERS:

```
hive> SELECT a.foo FROM invites a WHERE a.ds='2008-08-15';
```

```
hive> INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out' SELECT a.* FROM invites a WHERE a.ds='2008-08-15';
```

## --GROUP BY:

```
hive> FROM invites a INSERT OVERWRITE TABLE events SELECT a.bar, count(*) WHERE a.foo > 0 GROUP BY a.bar;  
hive> INSERT OVERWRITE TABLE events SELECT a.bar, count(*) FROM invites a WHERE a.foo > 0 GROUP BY a.bar;
```

# DQL Operations (2)

## --MULTITABLE INSERT:

```
FROM src
  INSERT OVERWRITE TABLE dest1 SELECT src.* WHERE src.key < 100
  INSERT OVERWRITE TABLE dest2 SELECT src.key, src.value WHERE src.key >= 100 and src.key < 200;
```

## --JOIN:

```
hive> FROM pokes t1 JOIN invites t2 ON (t1.bar = t2.bar) INSERT OVERWRITE TABLE events SELECT t1.bar, t1.foo, t2.foo;
```

## --STREAMING:

```
hive> FROM invites a INSERT OVERWRITE TABLE events SELECT TRANSFORM(a.foo, a.bar) AS (oof, rab) USING
  '/bin/cat' WHERE a.ds > '2008-08-09';
```

# Summary

- This course introduces Hive application scenarios, basic principles, Hive architecture, running process, and common Hive SQL statements.

# Quiz

1. (Multiple-choice) Which of the following scenarios are applicable to Hive? ( )
  - A. Online real-time data analysis
  - B. Data mining (including user behavior analysis, region of interest, and regional display)
  - C. Data summary (daily/weekly user clicks and click ranking)
  - D. Non-real-time analysis (log analysis and statistical analysis)
2. (Single-choice) Which of the following statements about basic Hive SQL operations is correct? ( )
  - A. You need to use the keyword "external" to create an external table and specify the keyword "internal" to create a normal table.
  - B. The location information must be specified when an external table is created.
  - C. When data is loaded to Hive, the source data must be a path in HDFS.
  - D. Column separators can be specified when a table is created.

# Recommendations

---

- Huawei Cloud Official Web Link:
  - <https://www.huaweicloud.com/intl/en-us/>
- Huawei MRS Documentation:
  - <https://www.huaweicloud.com/intl/en-us/product/mrs.html>
- Huawei TALENT ONLINE:
  - <https://e.huawei.com/en/talent/#/>

Thank you.



# Revision Record

Do Not Print this Page

Course Code	Product	Product Version	Course Version
H13-711	MRS		V3.0

Author/ID	Date	Reviewer/ID	New/ Update
Wang Mengde/wwx711842	2020.03.12	Huang Haoyang/hwx690472	Update
Wang Mengde/wwx711842	2020.08.29	Huang Haoyang/hwx690472	New

# Chapter 4 HBase Technical Principles



# Foreword

---

- This course describes the non-relational distributed database called HBase in the Hadoop open-source community, which can meet the requirements of large-scale and real-time data processing applications.

# Objectives

- On completion of this course, you will be able to be familiar with:
  - HBase system architecture and related concepts
  - HBase key processes and prominent features
  - HBase performance tuning
  - Basic shell operations of HBase

# Contents

- 1. Introduction to HBase**
2. HBase Related Concepts
3. HBase Architecture
4. HBase Key Processes
5. HBase Highlights
6. HBase Performance Tuning
7. Common HBase Shell Commands

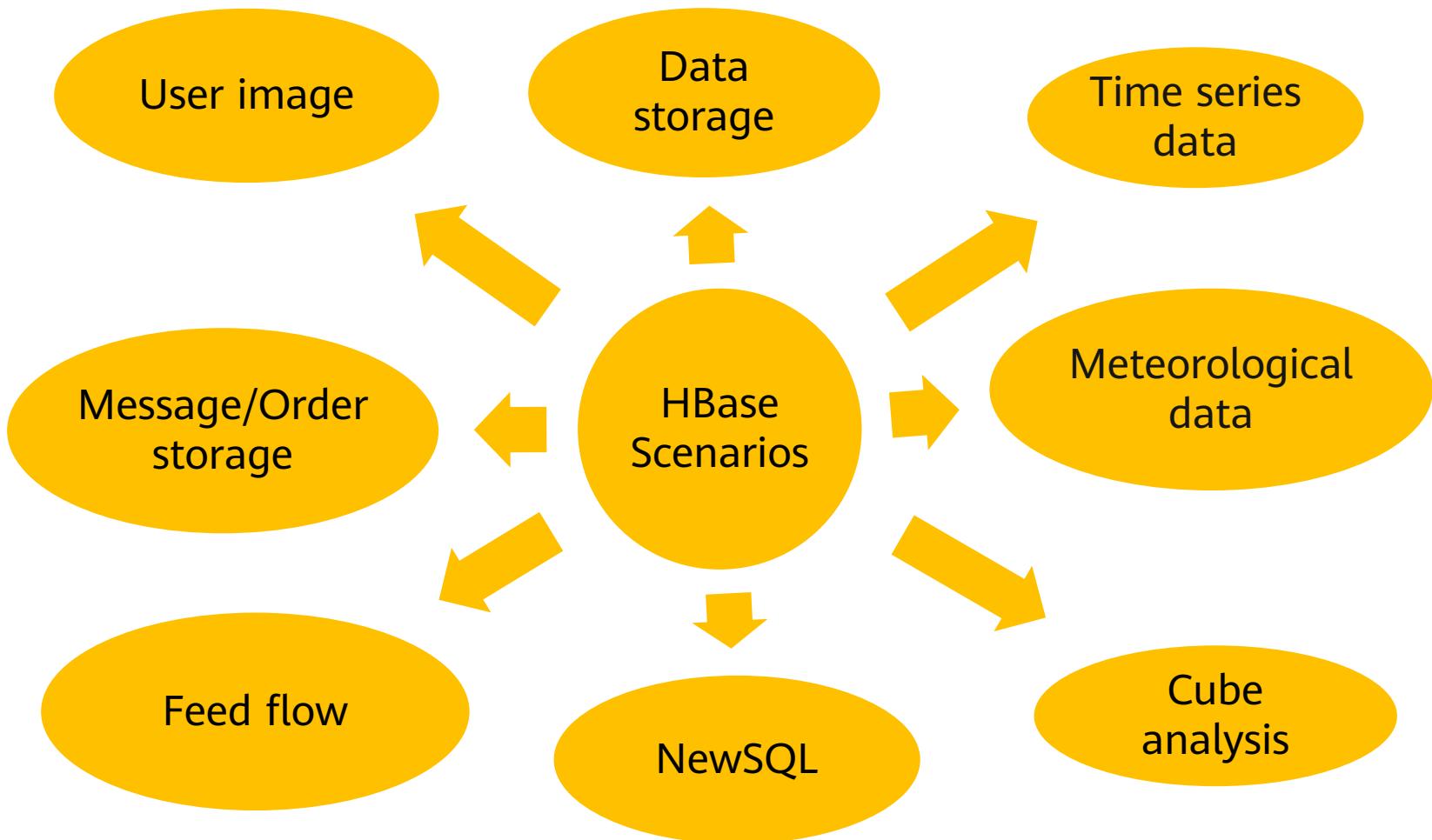
# Introduction to HBase

- HBase is a column-based distributed storage system that features high reliability, performance, and scalability.
  - HBase is suitable for storing data in a big table (the table can store billions of rows and millions of columns) and allows real-time data access.
  - Hadoop HDFS (Hadoop Distributed File System) is used as the file storage system to provide a distributed database system that supports real-time read and write operations.
  - HBase uses ZooKeeper as the collaboration service.

# Comparison Between HBase and RDB

- HBase differs from traditional relational databases in the following aspects:
  - Data indexing: A relational database can build multiple complex indexes for different columns to improve data access performance. HBase has only one index, that is, the row key. All access methods in HBase can be accessed by using the row key or row key scanning, ensuring the proper system running.
  - Data maintenance: In the relational database, the latest current value is used to replace the original value in the record during the update operation. The original value does not exist after being overwritten. When an update operation is performed in HBase, a new version is generated with the original version retained.
  - Scalability: It is difficult to implement horizontal expansion of relational databases, and the space for vertical expansion is limited. On the contrary, distributed databases, such as HBase and BigTable, are developed to implement flexible horizontal expansion. They can easily implement performance scaling by adding or reducing hardware in a cluster.

# HBase Application Scenario



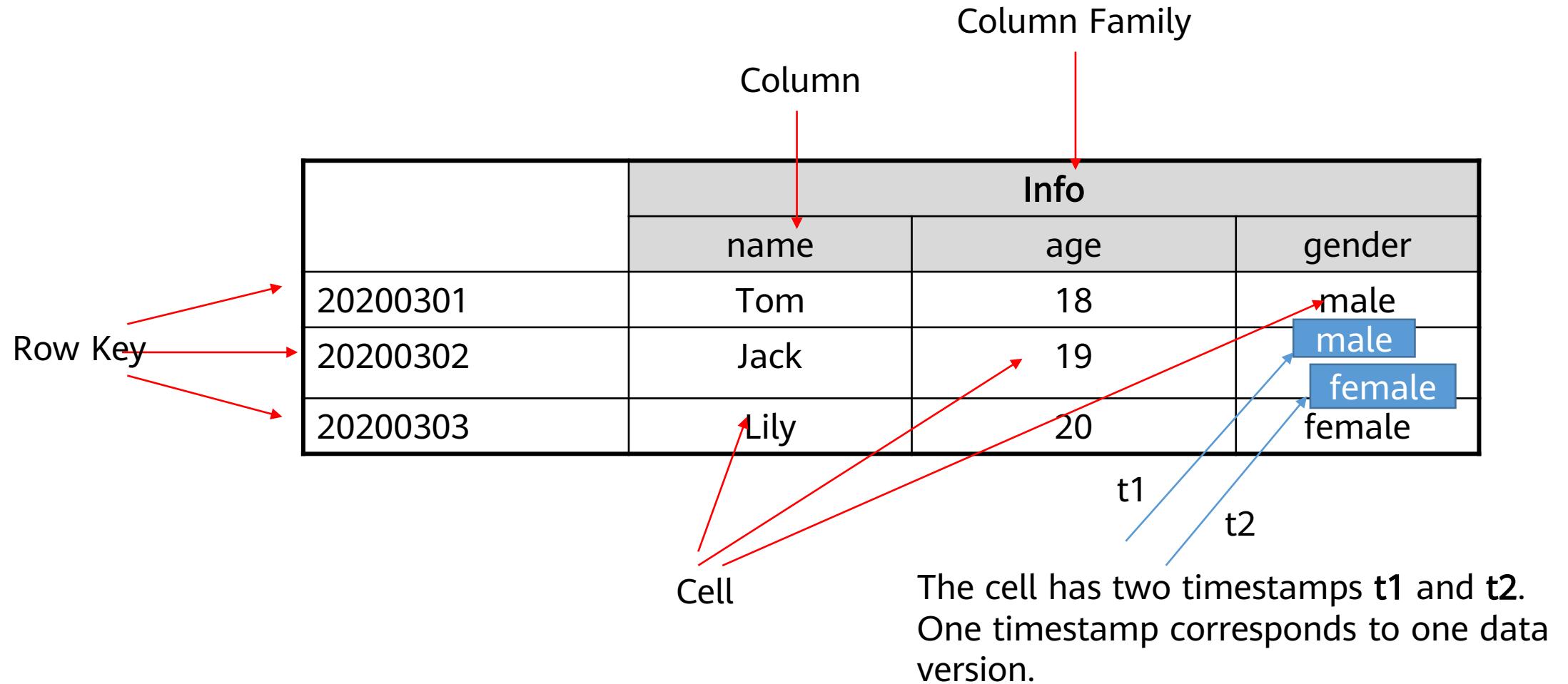
# Contents

1. Introduction to HBase
- 2. HBase Data Model**
3. HBase Architecture
4. HBase Key Processes
5. HBase Highlights
6. HBase Performance Tuning
7. Common HBase Shell Commands

# Data Model

- Simply, applications store data in HBase as tables.
- A table consists of rows and columns. All columns belong to a column family.
- The intersection of a row and a column is called a cell, and the cell is versioned. The contents of the cell are an indivisible byte array.
- The row key of a table is also a byte array, so anything can be saved, either as a string or as a number.
- HBase tables are sorted by key. The sorting mode is byte. All tables must have a primary key.

# HBase Table Structure (1)



# HBase Table Structure (2)

- Table: HBase uses tables to organize data. A table consists of rows and columns. A column is divided into several column families.
- Row: Each HBase table consists of multiple rows, and each row is identified by a row key.
- Column family: An HBase table is divided into multiple column families, which are basic access control units.
- Column qualifier: Data in a column family is located by column qualifiers (or columns).
- Cell: In an HBase table, a cell is determined by the row, column family, and column qualifier. Data stored in a cell has no data type and is considered as a byte array `byte[]`.
- Timestamp: Each cell stores multiple versions of the same data. These versions are indexed using timestamps.

# Conceptual View of Data Storage

- There is a table named **webtable** that contains two column families: **contents** and **anchor**. In this example, **anchor** has two columns (**anchor:aa.com** and **anchor:bb.com**), and **contents** has only one column (**contents:html**).

Row Key	Time Stamp	ColumnFamily contents	ColumnFamily anchor
"com.cnn.www"	t9		anchor:aa.com= "CNN"
"com.cnn.www"	t8		anchor:bb.com= "CNN.com"
"com.cnn.www"	t6	contents:html=<html>..."	
"com.cnn.www"	t5	contents:html=<html>..."	
"com.cnn.www"	t3	contents:html=<html>..."	

# Physical View of Data Storage

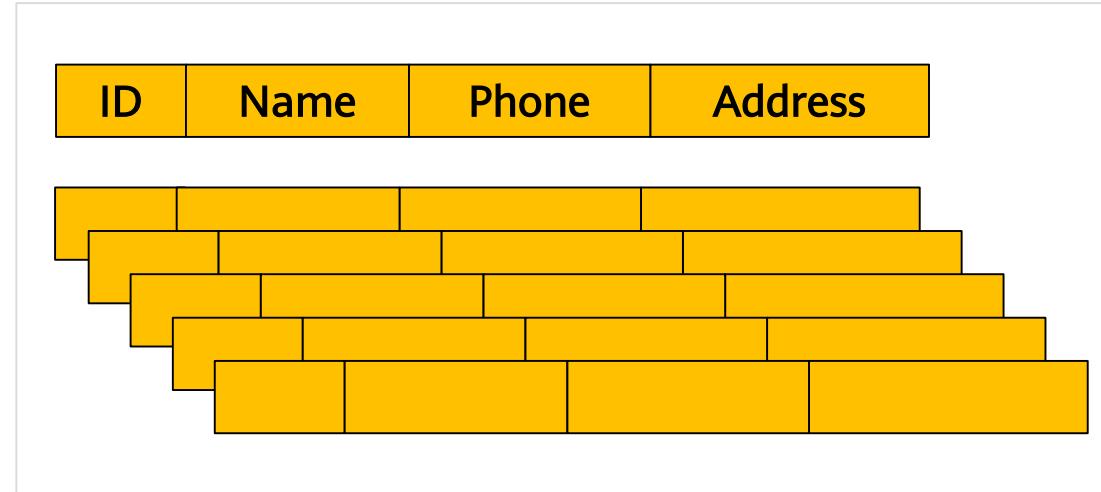
- Although in the conceptual view, a table can be considered as a collection of sparse rows. Physically, however, it differentiates column family storage. New columns can be added to a column family without being declared.

Row Key	Time Stamp	ColumnFamily anchor
"com.cnn.www"	t9	anchor:aa.com= "CNN"
"com.cnn.www"	t8	anchor:bb.com= "CNN.com"

Row Key	Time Stamp	ColumnFamily contents
"com.cnn.www"	t6	contents:html="<html>..."
"com.cnn.www"	t5	contents:html="<html>..."
"com.cnn.www"	t3	contents:html="<html>..."

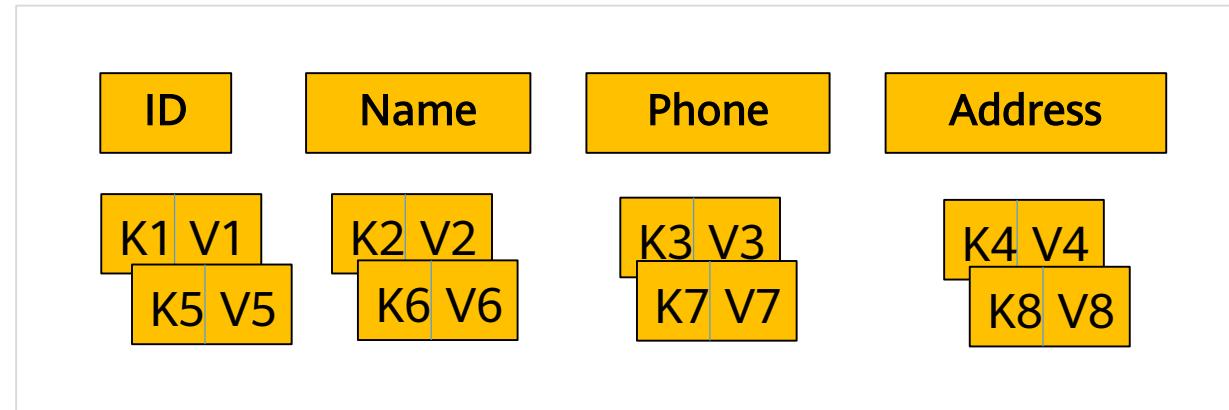
# Row-based Storage

- Row-based storage refers to data stored by rows in an underlying file system. Generally, a fixed amount of space is allocated to each row.
  - Advantages: Data can be added, modified, or read by row.
  - Disadvantage: Some unnecessary data is obtained when data in a column is queried.



# Column-based Storage

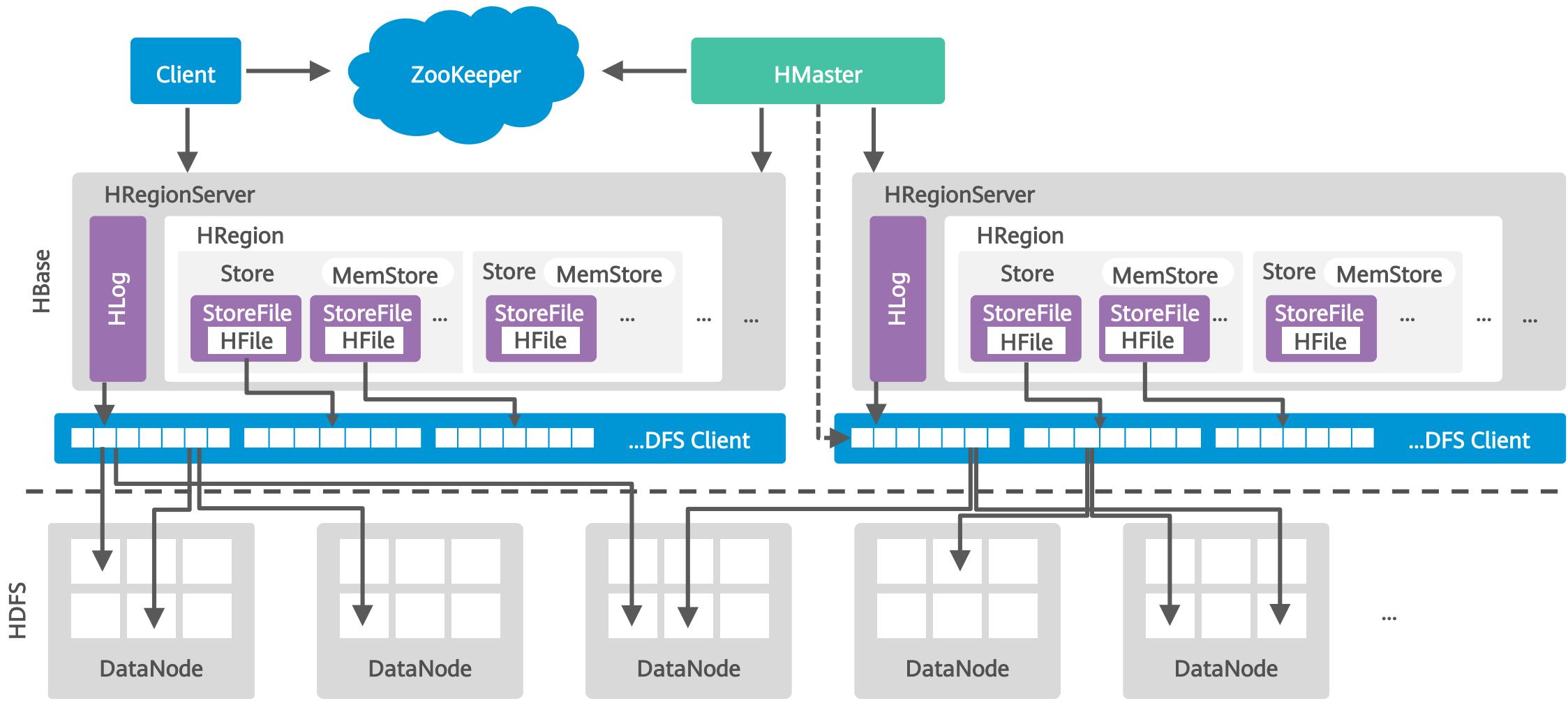
- Column-based storage refers to data stored by columns in an underlying file system.
  - Advantage: Data can be read or calculated by column.
  - Disadvantage: When a row is read, multiple I/O operations may be required.



# Contents

1. Introduction to HBase
2. HBase Related Concepts
- 3. HBase Architecture**
4. HBase Key Processes
5. HBase Highlights
6. HBase Performance Tuning
7. Common HBase Shell Commands

# HBase Architecture (1)



# HBase Architecture (2)

- The HBase architecture consists of the following functional components:
  - Library functions (linking to each client)
  - HMaster
  - HRegionServer

# HBase Architecture (3)

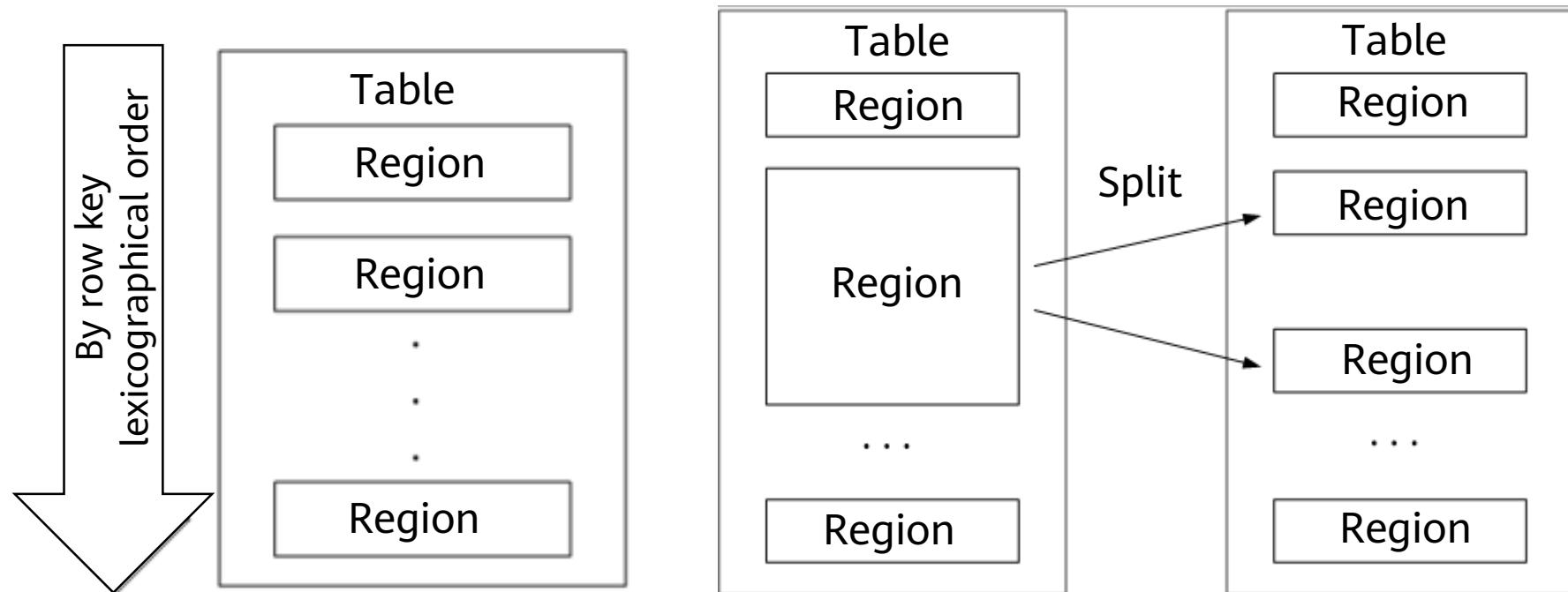
- The HMaster server manages and maintains the partition information in the HBase table, maintains the HRegionServer list, allocates regions, and balances loads.
- HRegionServer stores and maintains the allocated regions and process read and write requests from clients.
- The client does not directly read data from HMaster. Instead, the client directly reads data from HRegionServer after obtaining the storage location of the region.
- The client does not depend on the HMaster. Instead, the client obtains the region location through ZooKeeper. Most clients do not even communicate with the HMaster. This design reduces the load of the HMaster.

# HBase Architecture (4)

- Table (HBase table)
  - Region (Regions for the table)
    - Store (Store per ColumnFamily for each Region for the table)
      - MemStore (MemStore for each Store for each Region for the table)
      - StoreFile (StoreFiles for each Store for each Region for the table)
      - ~ Block (Blocks within a StoreFile within a Store for each Region for the table)

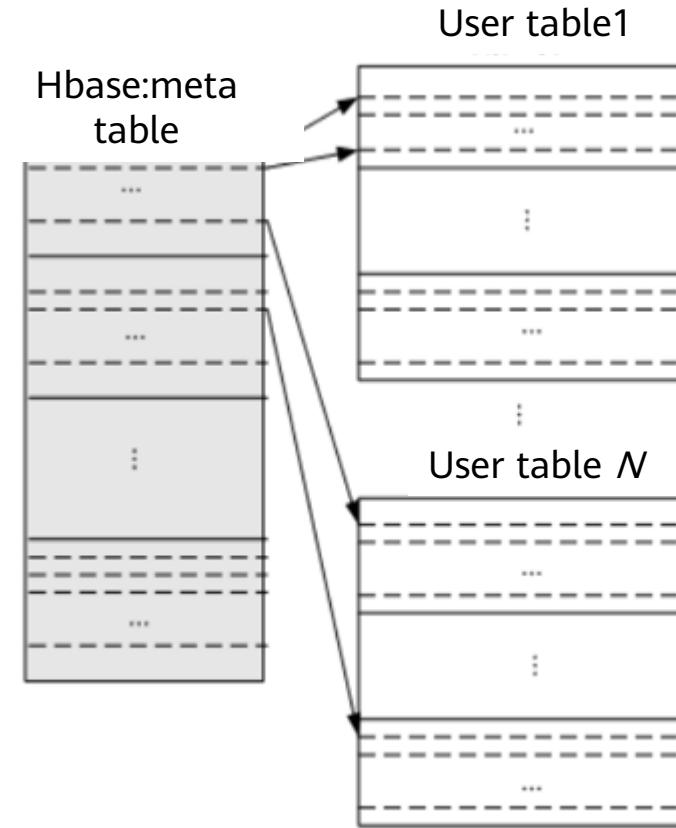
# Table and Region

- In normal cases, an HBase table has only one region. As the data volume increases, the HBase table is split into multiple regions.
- The region splitting operation is fast because the region still reads the original storage file after the splitting. The region reads the new file only after the storage file is asynchronously written to an independent file.



# Region Positioning (1)

- Region is classified into Meta Region and User Region.
- Meta Region records the routing information of each User Region.
- To read and write region data routing information, perform the following steps:
  - Find the Meta Region address.
  - Find the User Region address based on Meta Region.



## Region Positioning (2)

- To speed up access, the **hbase:meta** table is saved in memory.
- Assume that each row (a mapping entry) in the **hbase:meta** table occupies about 1 KB in the memory, and the maximum size of each region is 128 MB.
- In the two-layer structure,  $2^{17}$  (128 MB/1 KB) regions can be saved.

# Client

- The client contains the interface for accessing HBase and maintains the location information of the accessed regions in the cache to accelerate subsequent data access.
- The client queries the **hbase:meta** table first, and determines the location of the region. After the required region is located, the client directly accesses the corresponding region (without passing through the HMaster) and initiates a read/write request.



# HMaster HA

- ZooKeeper can help elect an HMaster node as the primary management node of the cluster and ensure that there is only one HMaster node running at any time, preventing single point of failures (SPOFs) of the HMaster node.

# HMaster

- The HMaster server manages tables and regions by performing the following operations:
  - Manages users' operations on tables, such as adding, deleting, modifying, and querying.
  - Implements load balancing between different HRegionServers.
  - Adjusts the distribution of regions after they are split or merged.
  - Migrates the regions on the faulty HRegionServers.

# HRegionServer

- HRegionServer is the core module of HBase. It provide the following main functions:
  - Maintains the regions allocated.
  - Responds to users' read and write requests.

# Contents

1. Introduction to HBase
2. HBase Related Concepts
3. HBase Architecture
- 4. HBase Key Processes**
5. HBase Highlights
6. HBase Performance Tuning
7. Common HBase Shell Commands

# Data Read and Write Process

- When you write data, the data is allocated to the corresponding HRegionServer for execution.
- Your data is first written to MemStore and HLog.
- The **commit()** invocation returns the data to the client only after the operation is written to HLog.
- When you read data, the HRegionServer first accesses MemStore cache. If the MemStore cache cannot be found, the HRegionServer searches StoreFile on the disk.

# Cache Refreshing

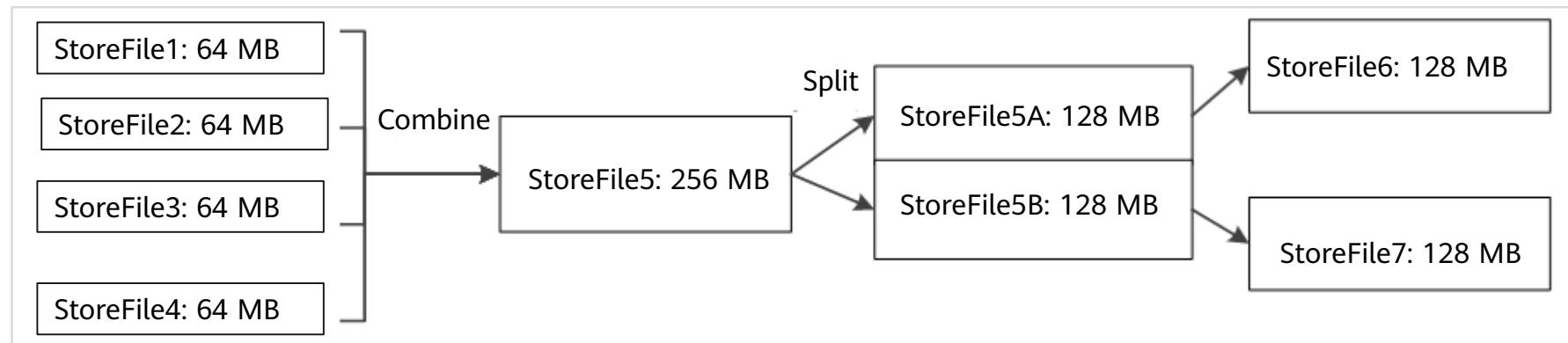
- The system periodically writes the content in the MemStore cache to the StoreFile file in the disk, clears the cache, and writes a tag in the HLog.
- A new StoreFile file is generated each time data is written. Therefore, each Store contains multiple StoreFile files.
- Each HRegionServer has its own HLog file. Each time the HRegionServer is started, the HLog file is checked to confirm the latest startup. Check whether a new write operation is performed after the cache is refreshed. If an update is detected, the data is written to MemStore and then to StoreFile. At last, the old HLog file is deleted, and HRegionServer provides services for you.

# Merging StoreFiles

- A new StoreFile is generated each time data is flushed, affecting the search speed due to the large number of StoreFiles.
- The `Store.compact()` function is used to combine multiple StoreFiles into one.
- The merge operation is started only when the number of StoreFiles reaches a threshold because the merge operation consumes a large number of resources.

# Store Implementation

- Store is the core of a HRegionServer.
- Multiple StoreFiles are combined into one Store.
- When the size of a single StoreFile is too large, splitting is triggered. One parent region is split into two sub-regions.



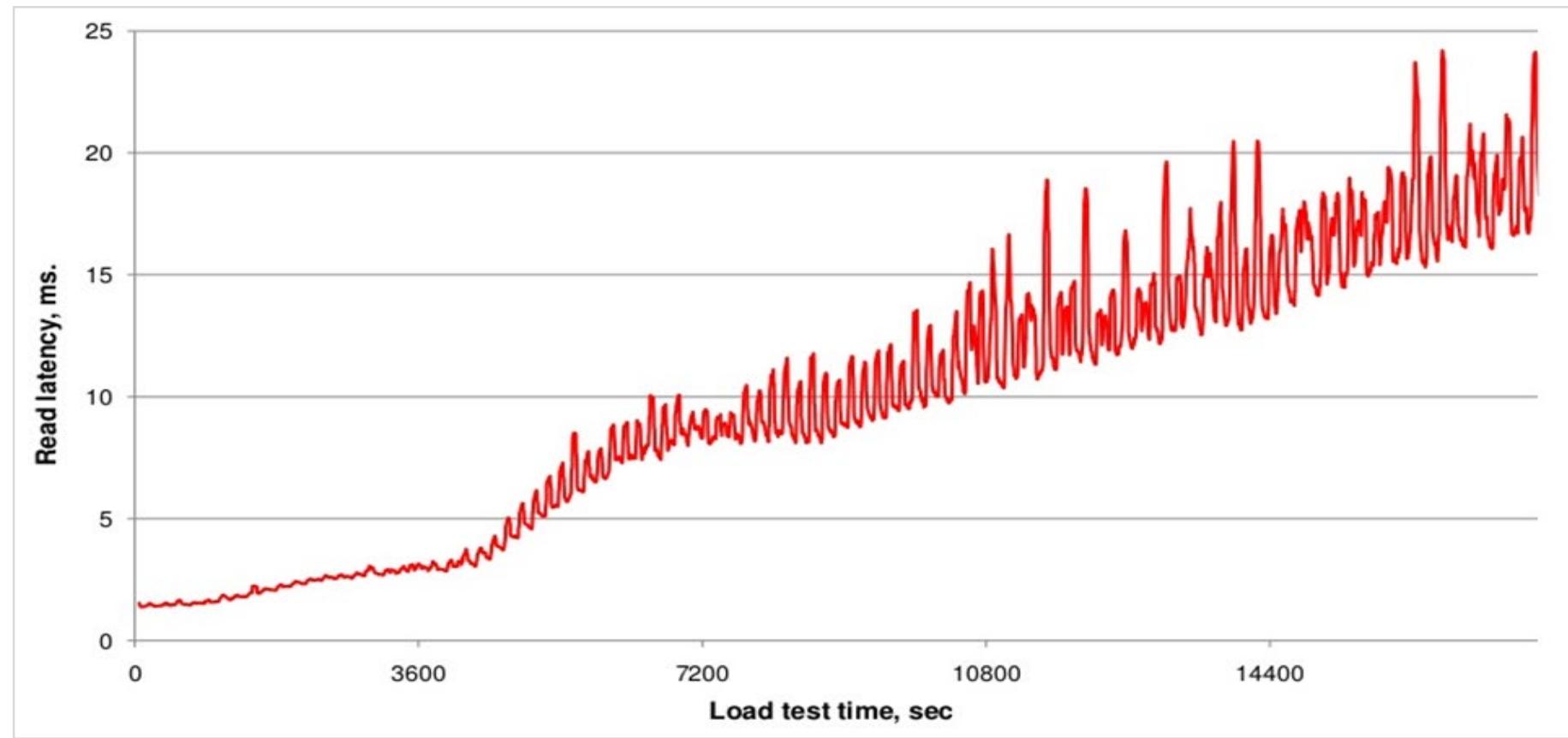
# HLog Implementation

- In a distributed environment, you need to consider system errors. HBase uses HLog to ensure system recovery.
- The HBase system configures an HLog file for each HRegionServer, which is a write-ahead log (WAL).
- The updated data can be written to the MemStore cache only after the data is written to logs. In addition, the cached data can be written to the disk only after the logs corresponding to the data cached in the MemStore are written to the disk.

# Contents

1. Introduction to HBase
2. HBase Related Concepts
3. HBase Architecture
4. HBase Key Processes
- 5. HBase Highlights**
6. HBase Performance Tuning
7. Common HBase Shell Commands

# Impact of Multiple HFiles

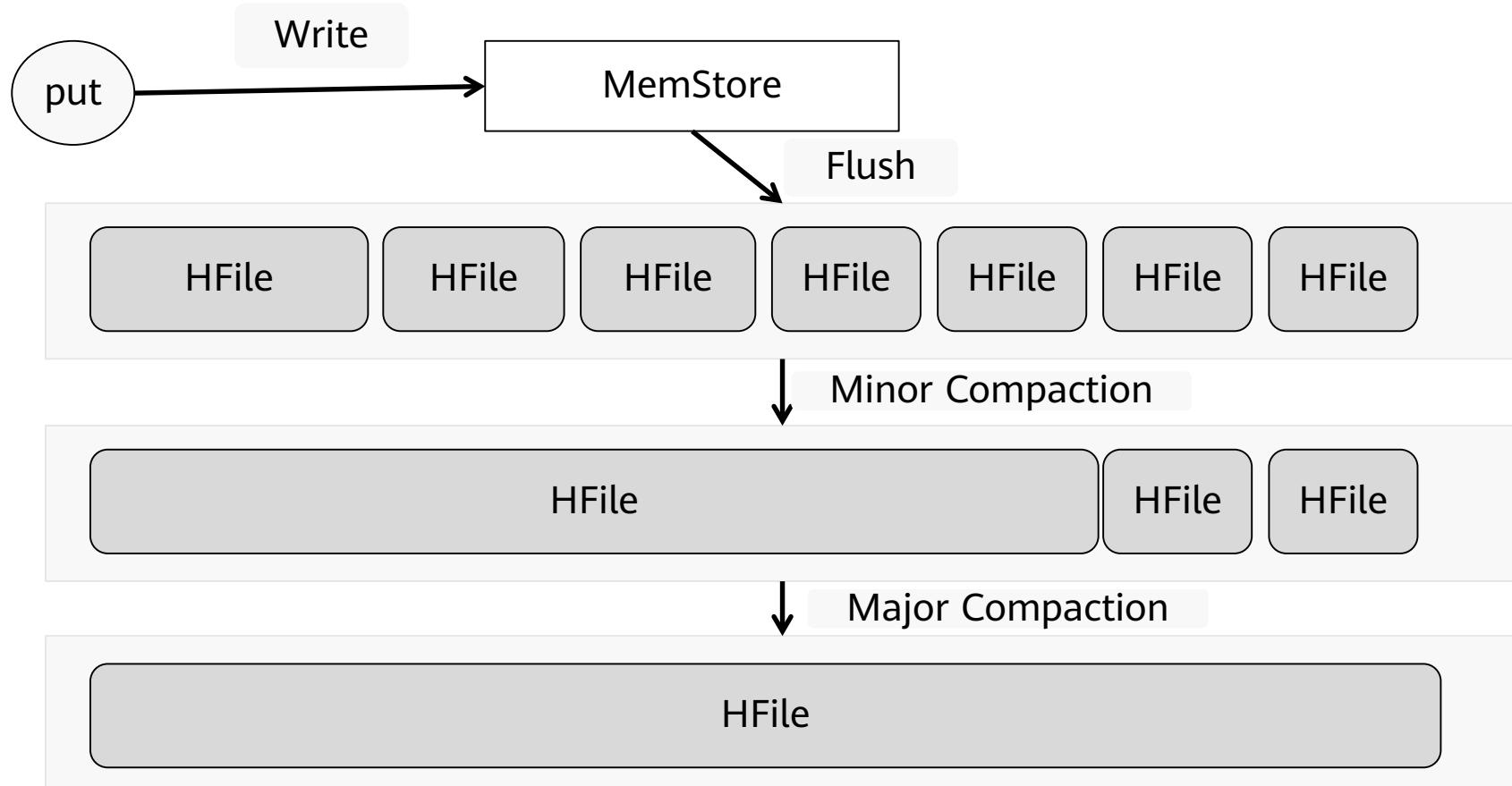


- The read latency prolongs as the number of HFiles increases.

# Compaction (1)

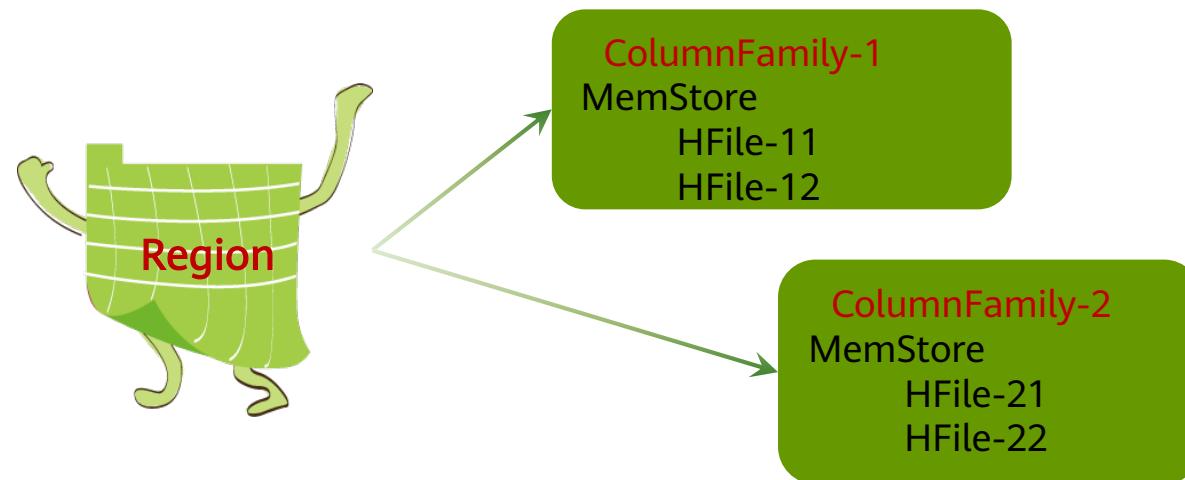
- Compaction is used to reduce the number of small files (HFiles) in the same column family of the same region to improve the read performance.
- Compaction is classified into minor compaction and major compaction.
  - Minor: indicates small-scale compaction. There are limits on the minimum and maximum number of files. Generally, small files in a continuous time range are merged.
  - Major: indicates the compaction of all HFile files under the column family of the region.
  - Minor compaction complies with a certain algorithm when selecting files.

# Compaction - 2



# OpenScanner

- In the OpenScanner process, two different scanners are created to read HFile and MemStore data.
  - The scanner corresponding to HFile is StoreFileScanner.
  - The scanner corresponding to MemStore is MemStoreScanner.



# BloomFilter

- BloomFilter is used to optimize some random read scenarios, that is, the **Get** scenario. It can be used to quickly determine whether a piece of user data exists in a large data set (most data in the data set cannot be loaded to the memory).
- BloomFilter has possibility of misjudgment when determining whether a piece of data exists. However, the judgment result of "The data xxxx does not exist" is reliable.
- BloomFilter's data in HBase is stored in HFiles.

# Contents

1. Introduction to HBase
2. HBase Related Concepts
3. HBase Architecture
4. HBase Key Processes
5. HBase Highlights
- 6. HBase Performance Tuning**
7. Common HBase Shell Commands

# Row Key

- Row keys are stored in **alphabetical** order. Therefore, when designing row keys, you need to fully use the sorting feature to store the data that is frequently read together and the data that may be accessed recently.
- For example, if the data that is recently written to the HBase table is most likely to be accessed, you can use the timestamp as a part of the row key. Because the data is sorted in alphabetical order, you can use `Long.MAX_VALUE - timestamp` as the row key, in this way, newly written data can be quickly hit when being read.

# Creating HBase Secondary Index (1)

- HBase has only one index for row keys.
- There are three methods for accessing rows in the HBase table:
  - Access through a single rowkey
  - Access through a row key interval
  - Full table scan

# Creating HBase Secondary Index (2)

- Hindex Secondary Index
  - Hindex is a Java-based HBase secondary index developed by Huawei and is compatible with Apache HBase 0.94.8. The current features are as follows:
    - Multiple table indexes
    - Multiple column indexes
    - Index based on some column values

# Contents

1. Introduction to HBase
2. HBase Related Concepts
3. HBase Architecture
4. HBase Key Processes
5. HBase Highlights
6. HBase Performance Tuning
7. Common HBase Shell Commands

# Common HBase Shell Commands

- **create**: creating Hive tables
- **list**: listing all tables in HBase
- **put**: adding data to a specified cell in a table, row, or column
- **scan**: browsing information about a table
- **get**: obtaining the value of a cell based on the table name, row, column, timestamp, time range, and version number
- **enable/disable**: enabling or disabling a table
- **drop**: deleting a table

# Summary

- This course describes the knowledge about the HBase database. HBase is an open source implementation of BigTable. Similar to BigTable, HBase supports a large amount of data and distributed concurrent data processing. It is easy to expand, supporting dynamic scaling, and is applicable to inexpensive devices.
- Additionally, this course describes the differences between the conceptual view and physical view of HBase data. HBase is a mapping table that stores data in a sparse, multi-dimensional, and persistent manner. It uses row keys, column keys, and timestamps for indexing, and each value is an unexplained string.

# Quiz

1. (Single-choice) Which of the following is type used to store data in HBase? ( )
  - A. Int
  - B. Long
  - C. String
  - D. Byte[]

# Quiz

2. (Single-choice) What is the smallest storage unit of HBase? ( )
- A. Region
  - B. Column Family
  - C. Column
  - D. Cell

# Recommendations

---

- Huawei Cloud Official Web Link:
  - <https://www.huaweicloud.com/intl/en-us/>
- Huawei MRS Documentation:
  - <https://www.huaweicloud.com/intl/en-us/product/mrs.html>
- Huawei TALENT ONLINE:
  - <https://e.huawei.com/en/talent/#/>

Thank you.



# Revision Record

Do Not Print this Page

Course Code	Product	Product Version	Course Version
H13-711	MRS		V3.0

Author/ID	Date	Reviewer/ID	New/ Update
Chen Ao/cwx860206	2020.04.01	Zhang Xinqi/zwx701844	Update
Chen Ao/cwx860206	2020.08.26	Yan Hua/ywx416015	New

# Chapter 5 MapReduce and YARN Technical Principles



# Foreword

---

- This course describes MapReduce and YARN. MapReduce is the most famous computing framework for batch processing and offline processing in the big data field. In this course, you will learn its principles, processes, and application scenarios. YARN is the component responsible for unified resource management and scheduling in the Hadoop cluster. In this course, you will learn its definition, functions and architecture, HA solution, and fault tolerance mechanism, and how to use YARN to allocate resources. In addition, we will briefly introduce the enhanced features that Huawei provides for these components.

# Objectives

- Upon completion of this course, you will be able to:
  - Be familiar with the concept of MapReduce and YARN.
  - Understand the application scenarios and principles of MapReduce.
  - Understand functions and architectures of MapReduce and YARN.
  - Be familiar with features of YARN.

# Contents

---

- 1. Introduction to MapReduce and YARN**
2. Functions and Architectures of MapReduce and YARN
3. Resource Management and Task Scheduling of YARN
4. Enhanced Features

# MapReduce Overview

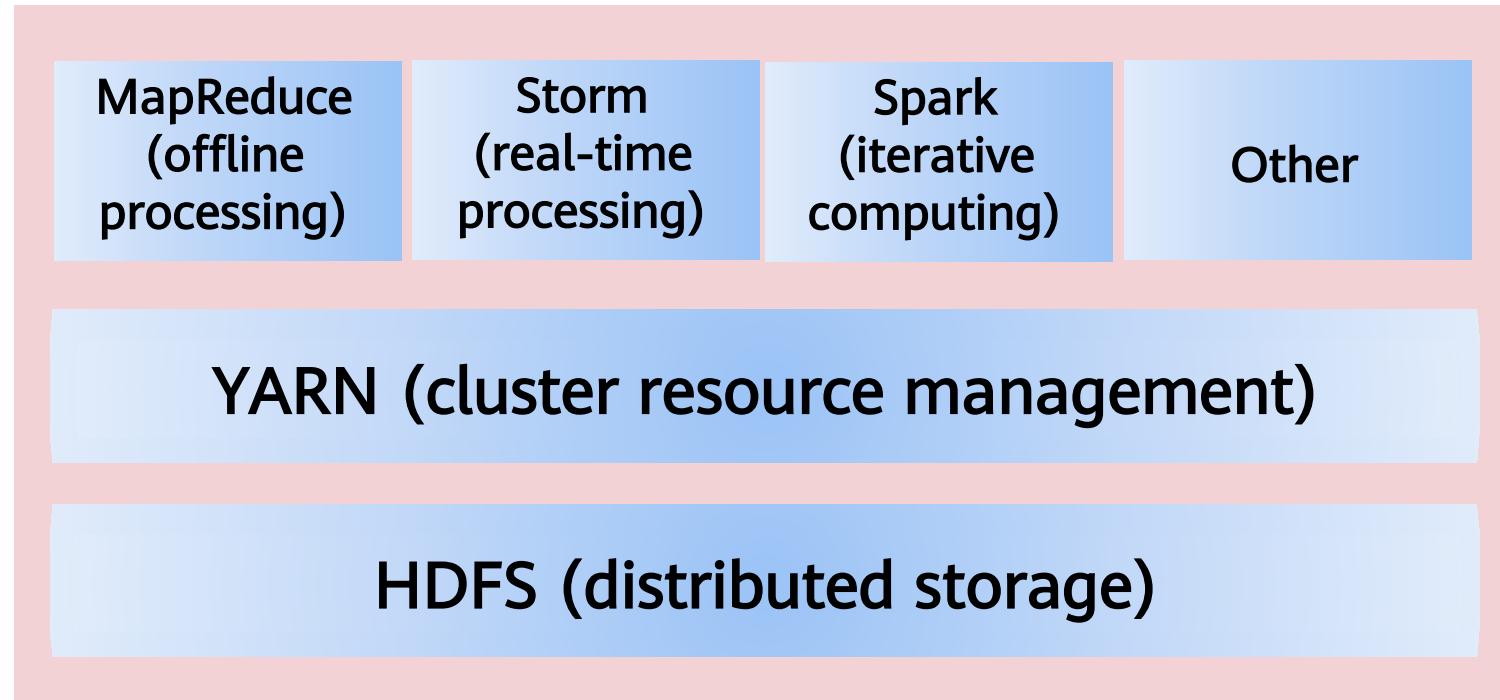
- MapReduce is designed and developed based on the MapReduce thesis released by Google. Based on the "divide-and-conquer" algorithm, MapReduce is used for parallel computing and offline computing of large-scale datasets (larger than 1 TB). It has the following features:
  - Highly abstract programming ideas: Programmers only need to describe what to do, and the execution framework will process the work accordingly.
  - Outstanding scalability: Cluster capabilities can be improved by adding nodes.
  - High fault tolerance: Cluster availability and fault tolerance are improved using policies, such as computing or data migration.

# Resource Scheduling and Management

- Hadoop 1.0 has only HDFS and MapReduce. Resource scheduling is implemented with MRv1, thus, the scheduling has the following defects:
  - The Master node has SPOFs. The fault recovery depends on the periodic checkpoint, which does not ensure the reliability. When a fault occurs, users determine whether to perform recalculation.
  - Job scheduling and resource scheduling are not distinguished. When MapReduce is running, a large number of concurrent jobs exist in the environment. Therefore, diversified and efficient scheduling policies are important.
  - Resource isolation and the security are not mentioned. When a large number of jobs are concurrently executed without considering resource isolation and security, the major issues in Hadoop 1.0 are how to ensure that a single job does not occupy too many resources and that users' programs are secure for the system.
- Therefore, Hadoop 2.0 introduces the YARN framework to better schedule and allocate cluster resources to overcome the shortcomings of Hadoop 1.0 and meet the diversified requirements of programming paradigms.

# YARN Overview

- Apache Hadoop Yet Another Resource Negotiator (YARN) is a resource management system. It provides unified resource management and scheduling for upper-layer applications, remarkably improving cluster resource utilization, unified resource management, and data sharing.



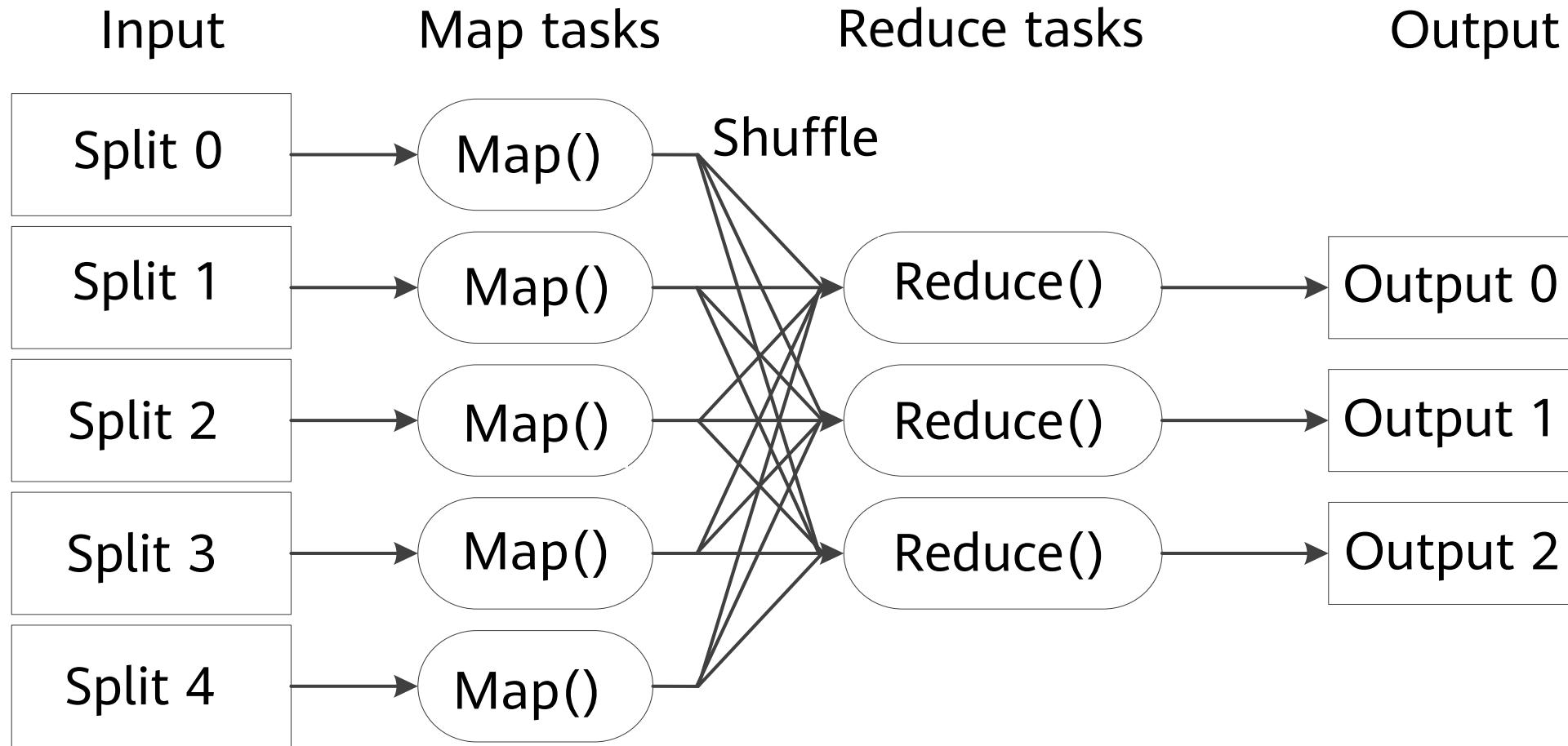
# Contents

1. Introduction to MapReduce and YARN
- 2. Functions and Architectures of MapReduce and YARN**
3. Resource Management and Task Scheduling of YARN
4. Enhanced Features

# MapReduce Process

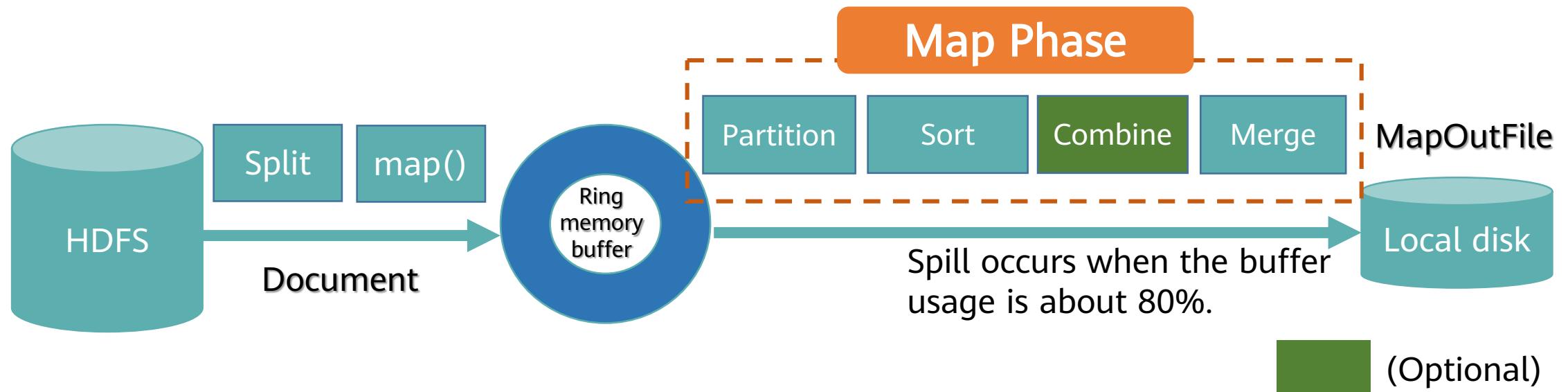
- As the name implies, the MapReduce calculation process can be divided into two phases: Map and Reduce. The output in the Map phase is the input in the Reduce phase.
- MapReduce can be understood as a process of summarizing a large amount of disordered data based on certain features and processing the data to obtain the final result.
  - In the Map phase, keys and values (features) are extracted from each piece of uncorrelated data.
  - In the Reduce phase, data is organized by keys followed by several values. These values are correlated. On this basis, further processing may be performed to obtain a result.

# MapReduce Workflow



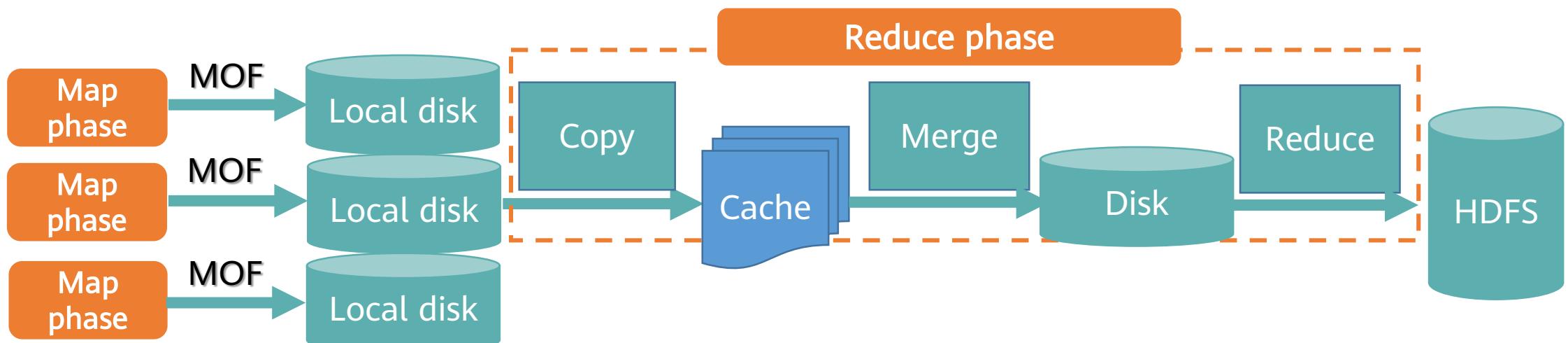
# Map Phase

- Before jobs are submitted, the files to be processed are split. By default, the MapReduce framework regards a block as a split. Client applications can redefine the mapping between blocks and splits.
- In the Map phase, data is stored in a ring memory buffer. When the data in the buffer reaches about 80%, spill occurs. In this case, the data in the buffer needs to be written to local disks.



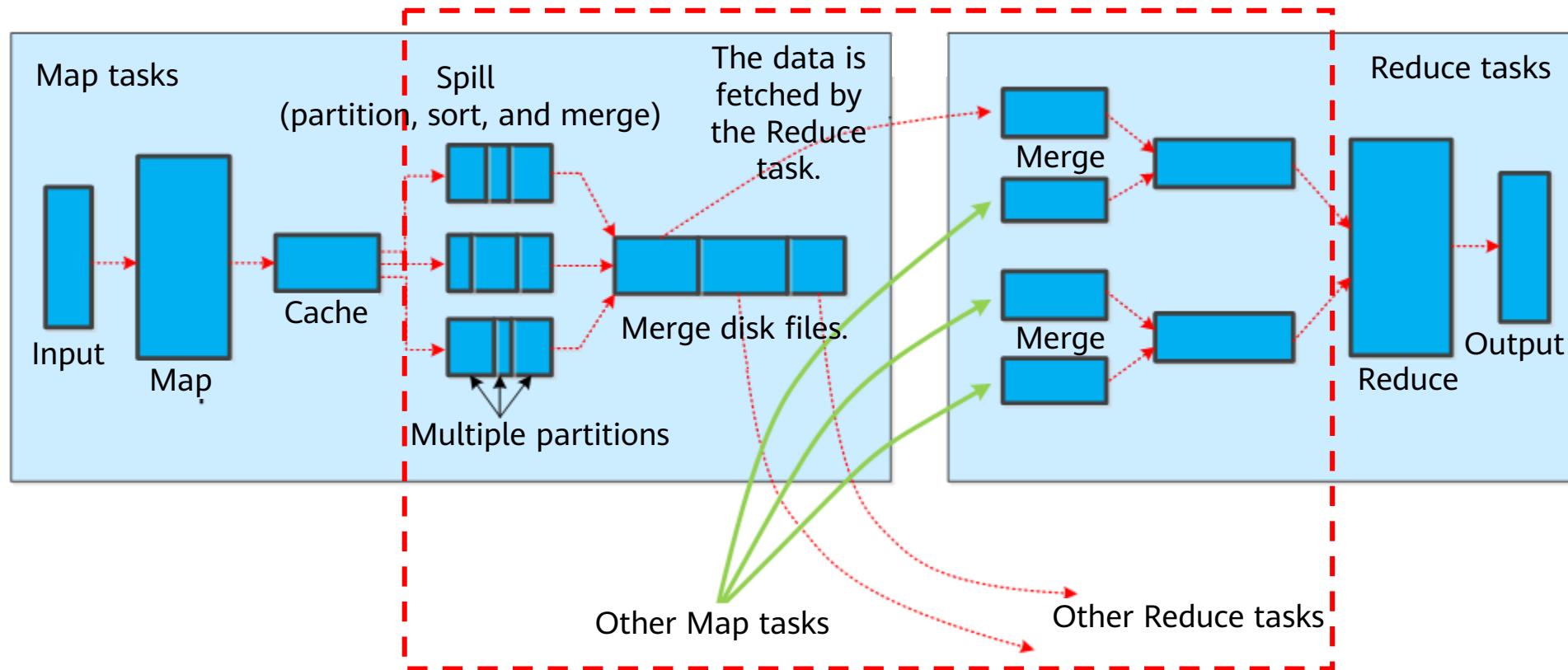
# Reduce Phase

- The MOF files are sorted. If the amount of data received by Reduce tasks is small, the data is directly stored in the buffer. As the number of files in the buffer increases, the MapReduce background thread merges the files into a large ordered file. Many intermediate files are generated during the merge operation. The last merge result is directly outputted to the Reduce function defined by the user.
- When the data volume is small, the data does not need to be spilled to the disk. Instead, the data is merged in the cache and then output to Reduce.

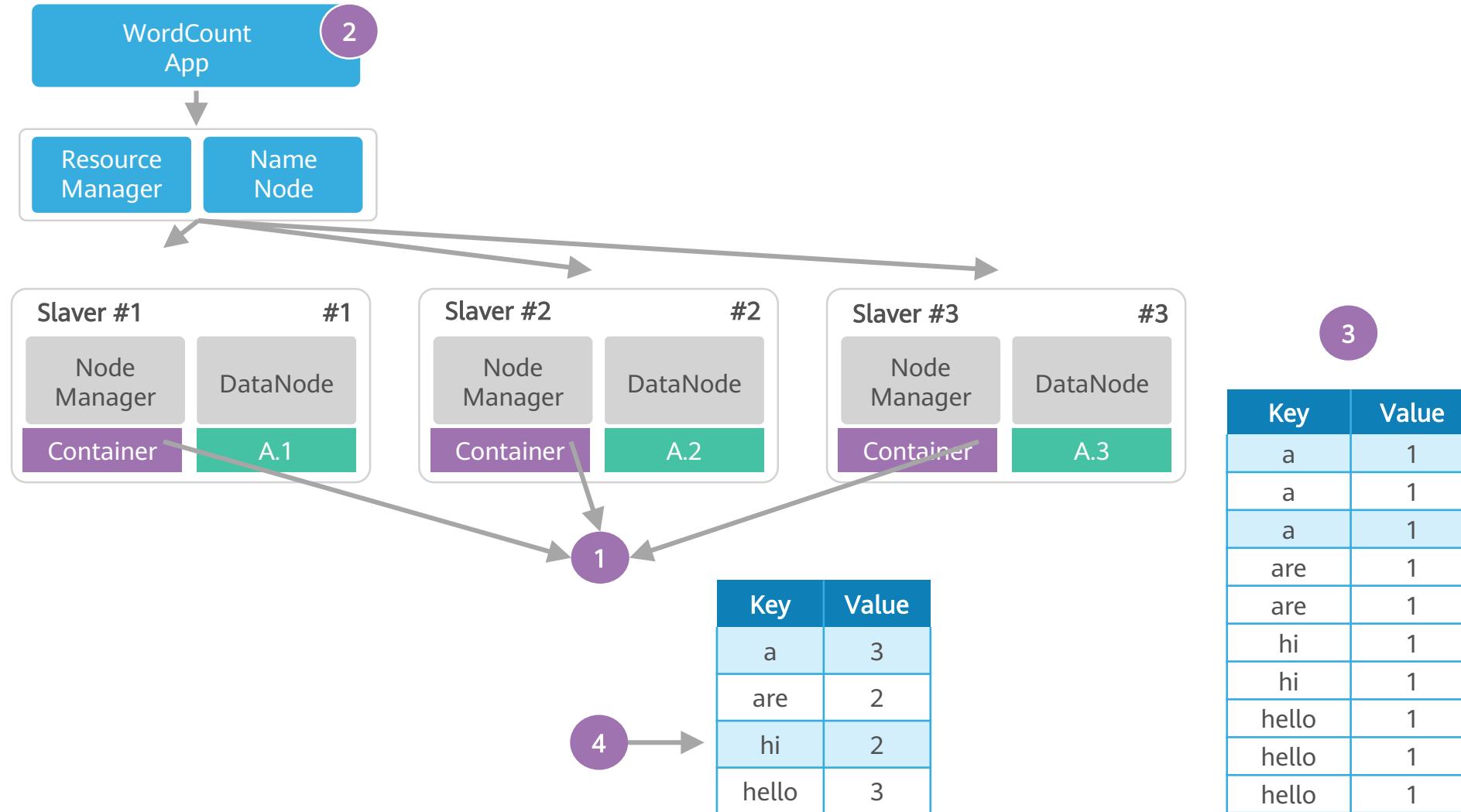


# Shuffle Process

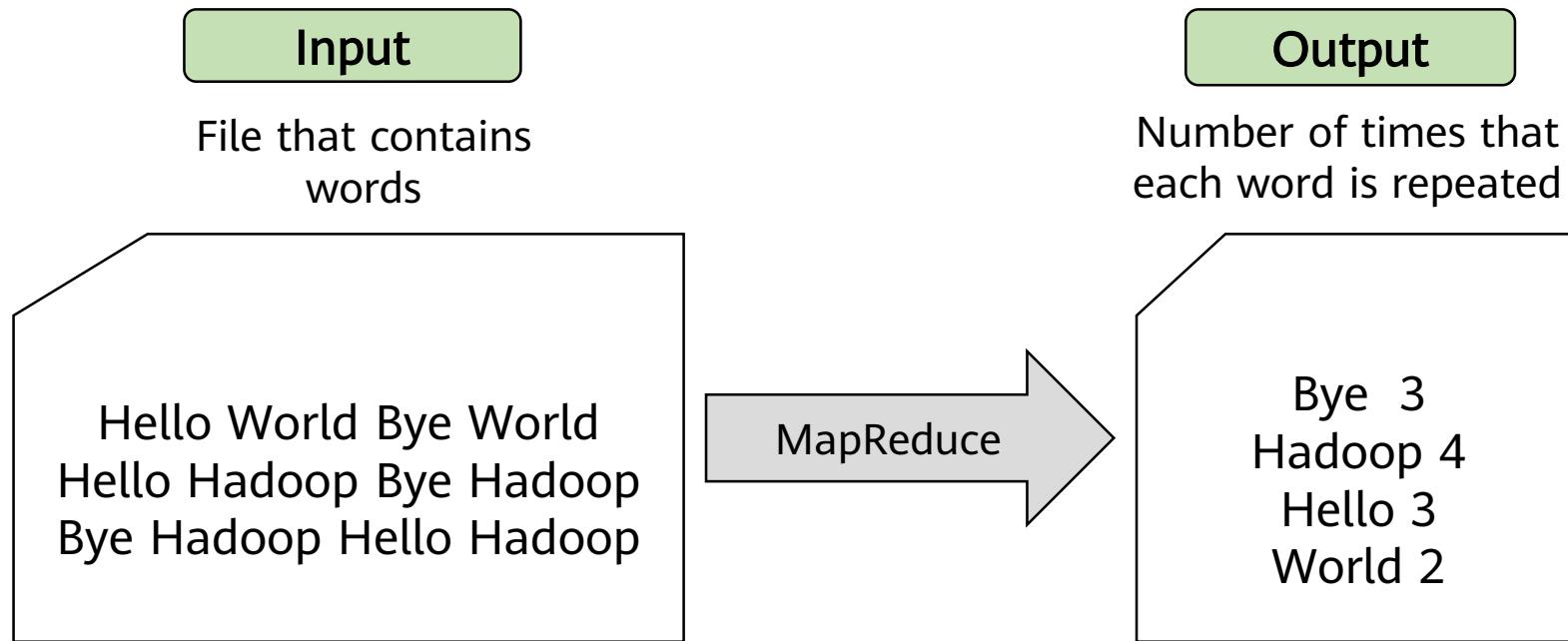
- Shuffle is the **intermediate data transfer** process between the Map phase and Reduce phase. The Shuffle process includes obtaining MOF files from the Map tasks of Reduce tasks and sorting and merging MOF files.



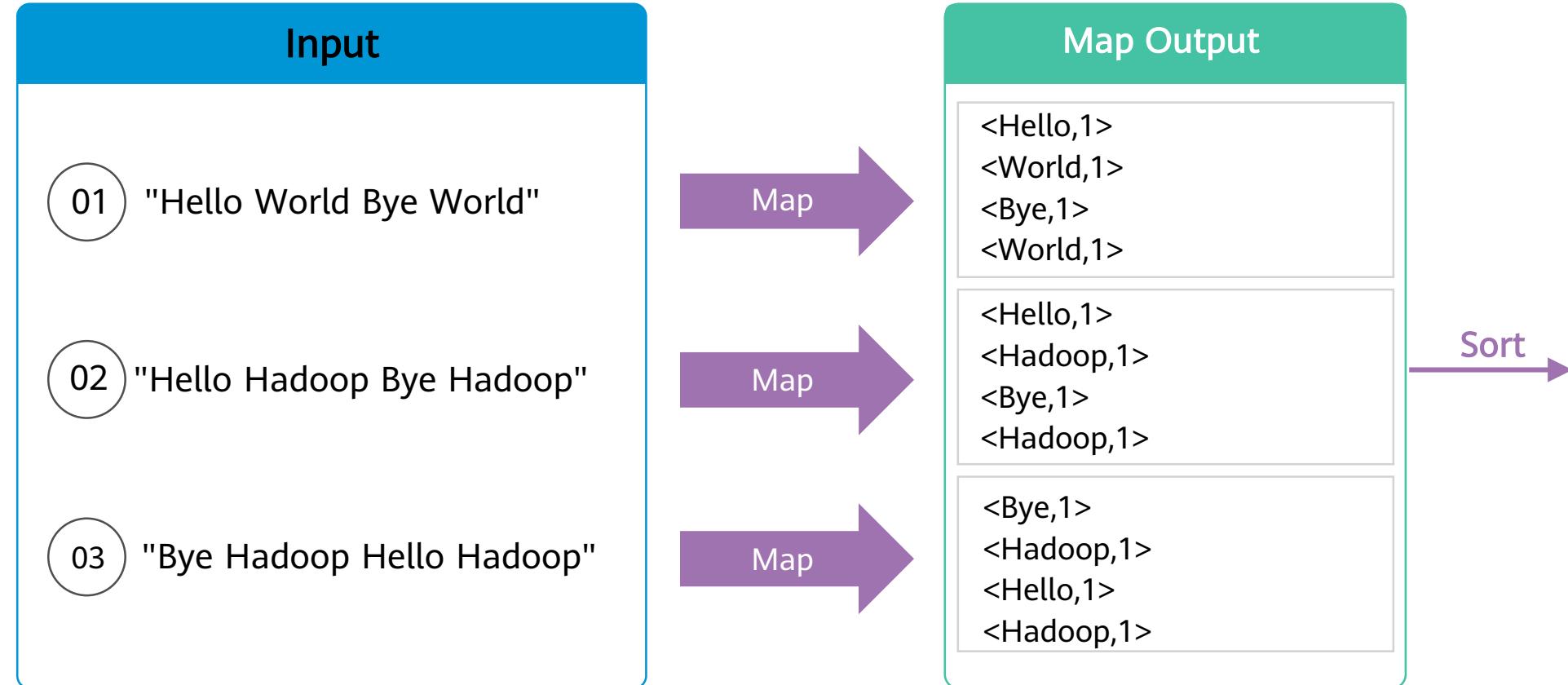
# WordCount Examples for Typical Programs



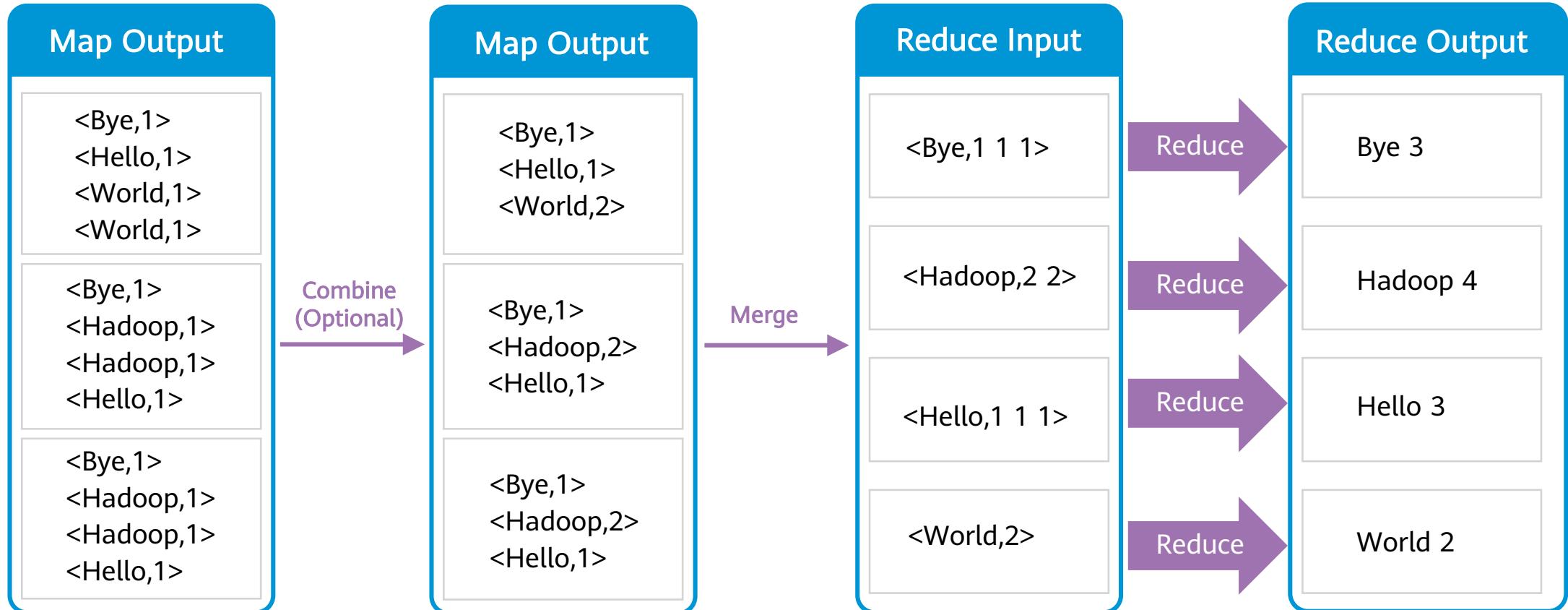
# Functions of WordCount



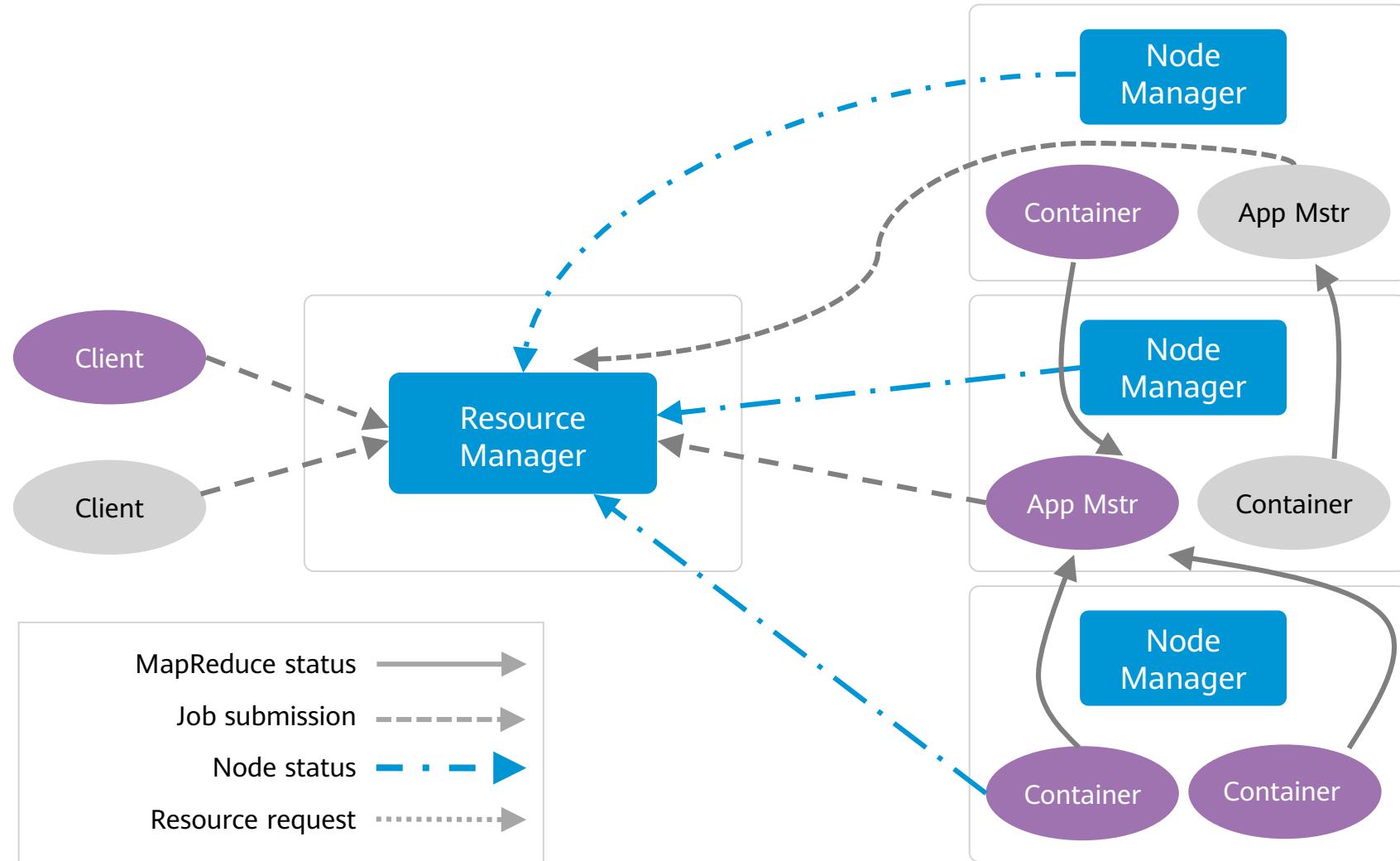
# Map Process of WordCount



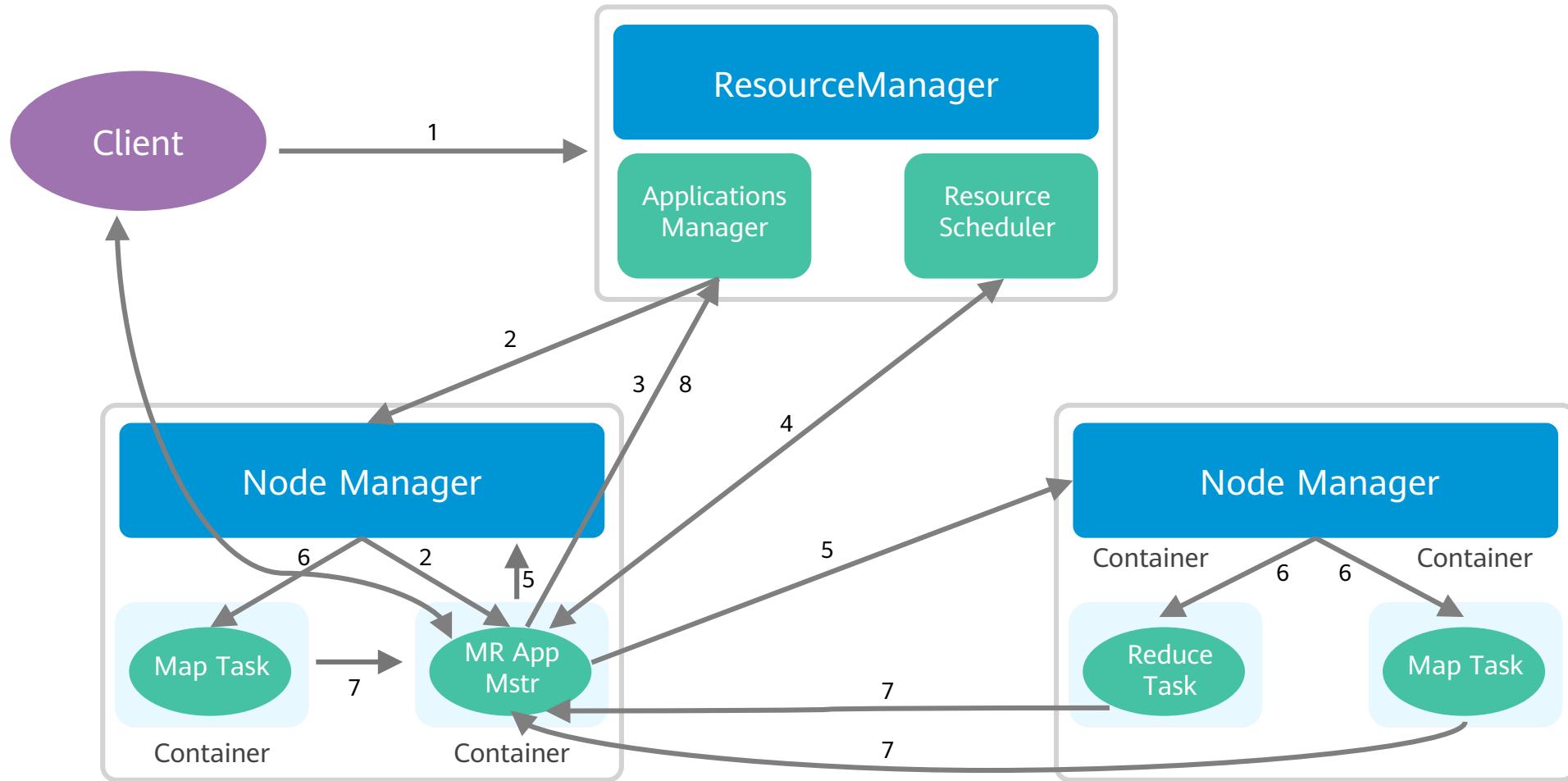
# Reduce Process of WordCount



# YARN Architecture

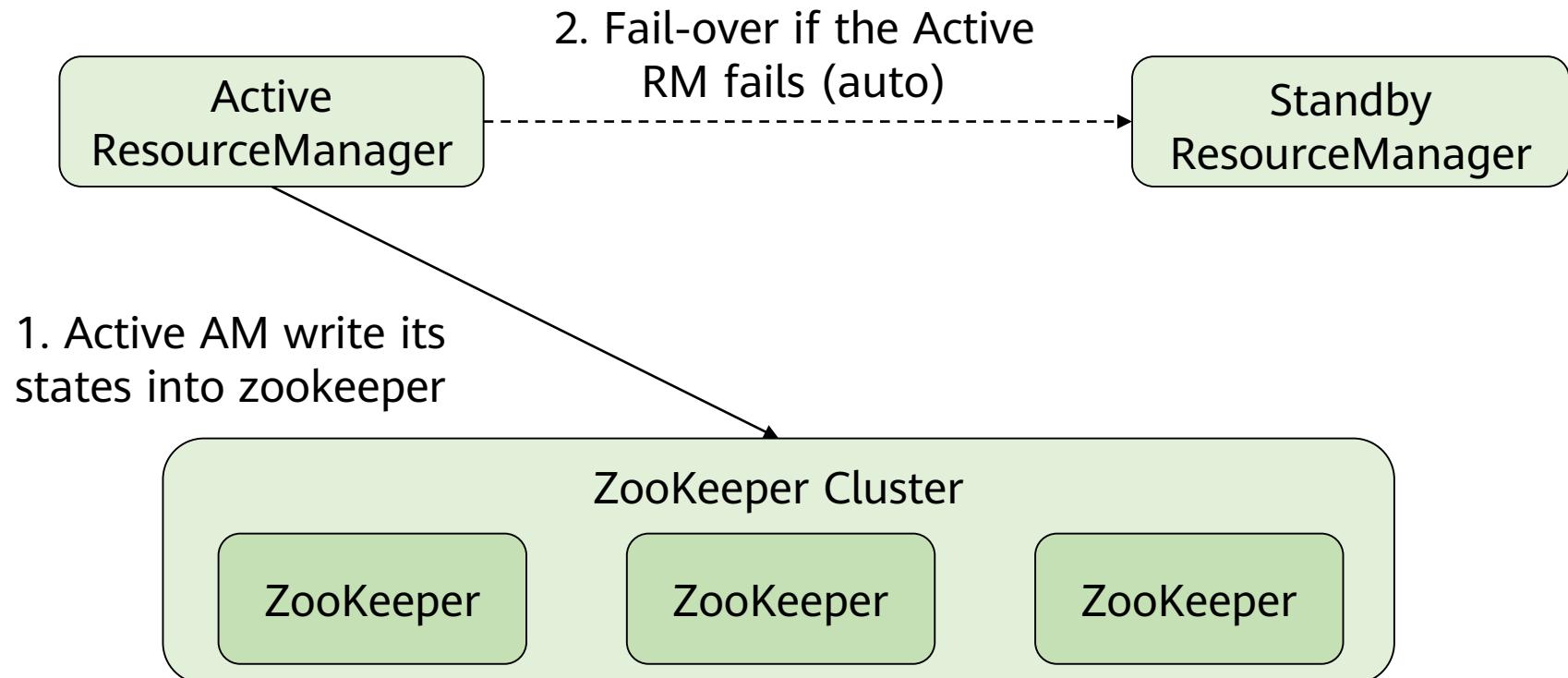


# MapReduce on YARN Task Scheduling Process

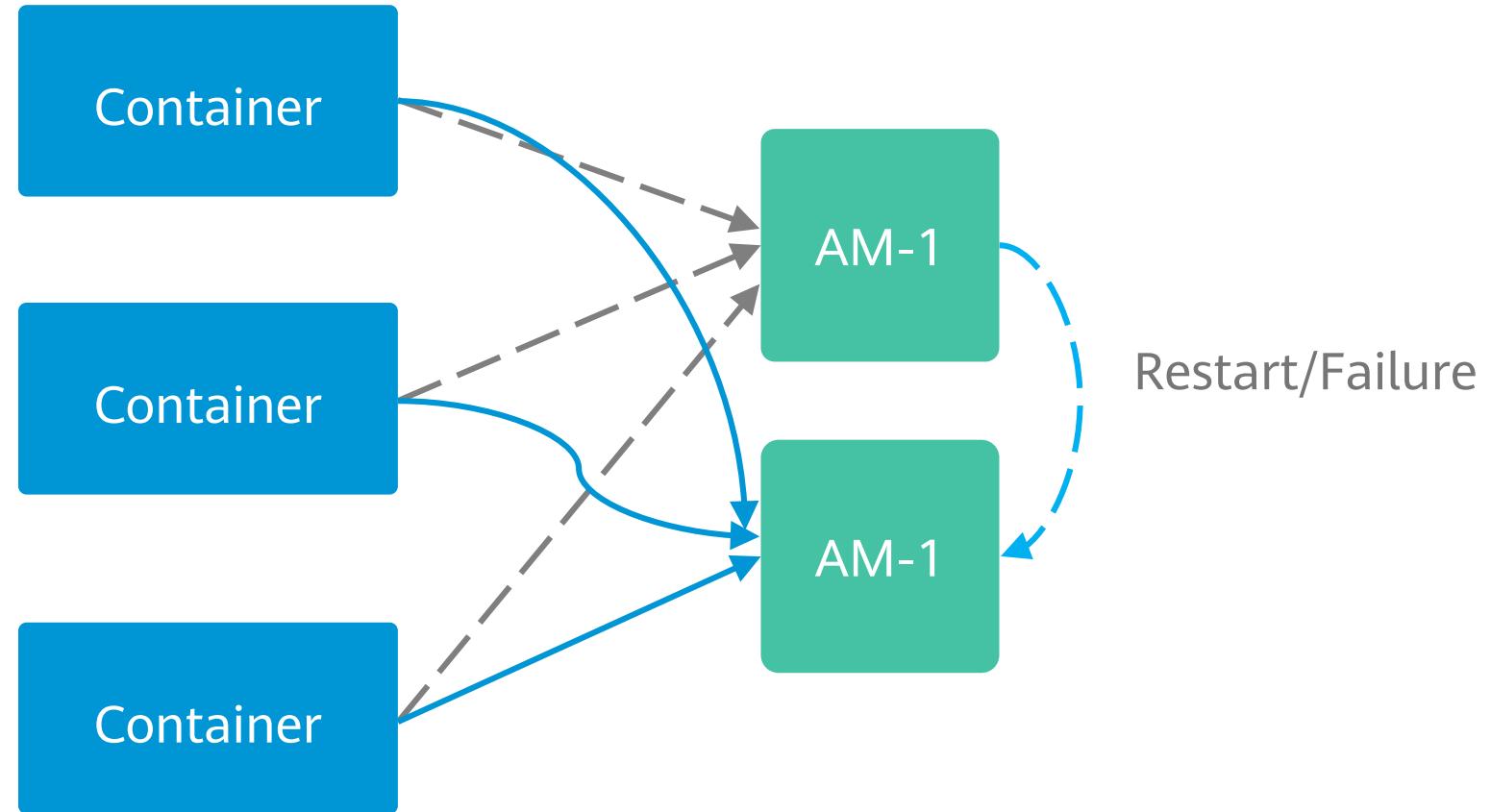


# YARN HA Solution

- ResourceManager in YARN is responsible for resource management and task scheduling of the entire cluster. The YARN HA solution uses redundant ResourceManager nodes to solve SPOFs.



# YARN APPMaster Fault Tolerance Mechanism



# Contents

1. Introduction to MapReduce and YARN
2. Functions and Architectures of MapReduce and YARN
- 3. Resource Management and Task Scheduling of YARN**
4. Enhanced Features

# Resource Management

- Available memory and CPU resources allocated to each NodeManager can be configured through the following parameters (on the YARN service configuration page):
  - **YARN.nodemanager.resource.memory-mb**: size of physical memory that YARN can allocate to a container
  - **YARN.nodemanager.vmem-pmem-ratio**: ratio of the virtual memory to the physical memory
  - **YARN.nodemanager.resource.cpu-vcore**: number of vCores that can be allocated to the container
- In Hadoop 3.x, the YARN resource model has been promoted to support user-defined countable resource types, not just CPU and memory.
  - Common countable resource types include GPU resources, software licenses, and locally-attached storage in addition to CPU and memory, but do not include ports and labels.

# YARN Resource Schedulers

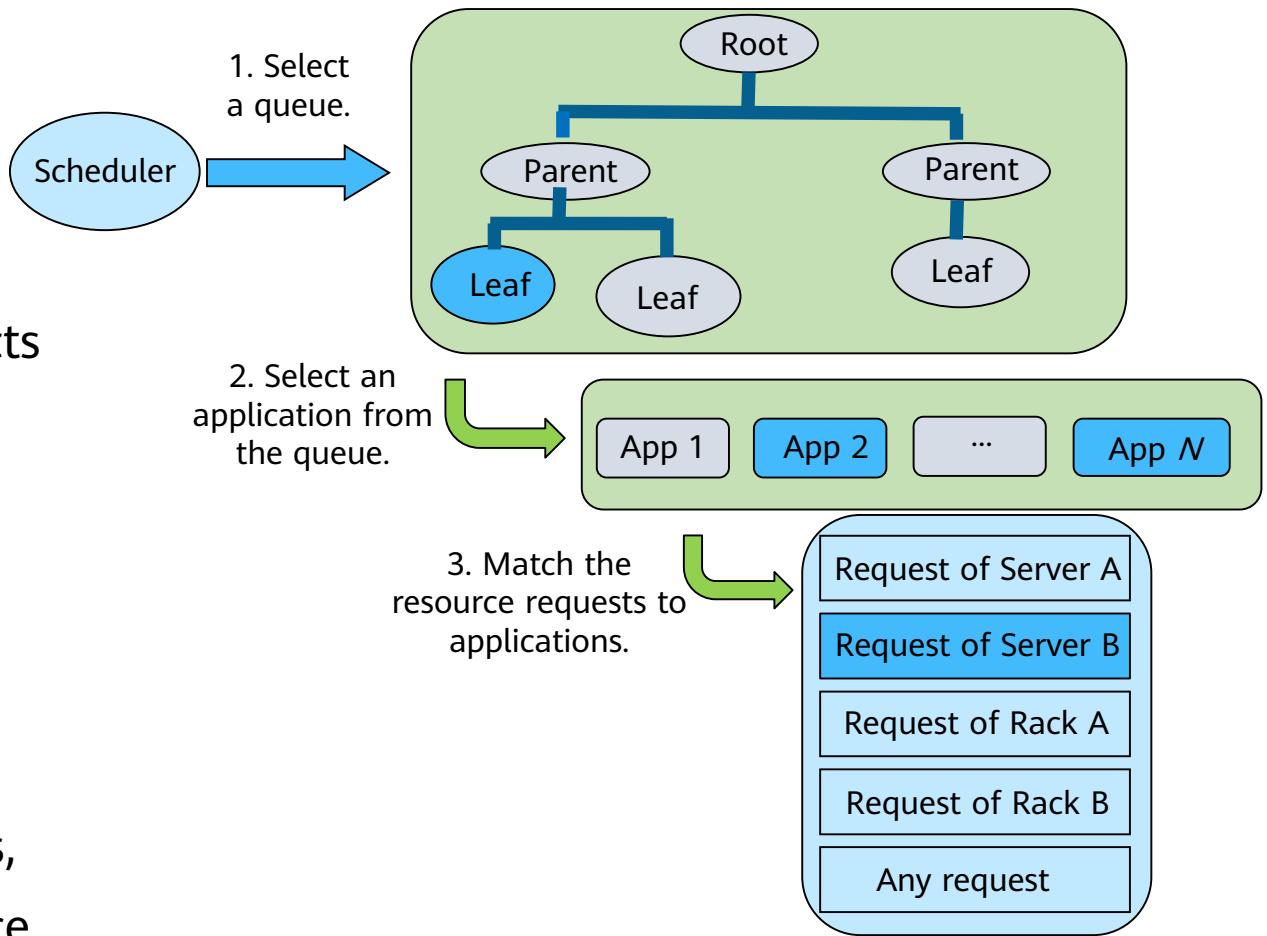
- In YARN, a scheduler allocates resources to applications. In YARN, three schedulers are available based on different policies:
  - FIFO scheduler: Applications are arranged into a queue based on the submission sequence. This queue is a first-in-first-out (FIFO) queue. During resource allocation, resources are first allocated to the application on the top of the queue. After the application on the top of the queue meets the requirements, resources are allocated to the next application. The rest may be deduced by analogy.
  - Capacity scheduler: Multiple organizations are allowed to share the entire cluster. Each organization can obtain some computing capabilities of the cluster. A dedicated queue is allocated to each organization, and certain cluster resources are allocated to each queue. Multiple queues are configured to provide services for multiple organizations. In addition, resources in a queue can be divided so that multiple members in an organization can share the queue resources. In a queue, the FIFO policy is used for resource scheduling.
  - The resources are allocated ‘fairly’ to each application. (The fairness can be configured.)

# Capacity Scheduler Overview

- Capacity scheduler enables Hadoop applications to run in a shared, multi-tenant cluster while maximizing the throughput and utilization of the cluster.
- Capacity scheduler allocates resources by queue. Upper and lower limits are specified for the resource usage of each queue. Each user can set the upper limit of resources that can be used. Administrators can restrict the resource used by a queue, user, or job. Job priorities can be set but resource preemption is not supported.
- In Hadoop 3.x, OrgQueue expands the capacity scheduler and provides a programmatic way to change the queue configuration through the REST API. In this way, the administrator can automatically configure and manage the queue in the **administer\_queue** ACL of the queue.

# Resource Allocation Model

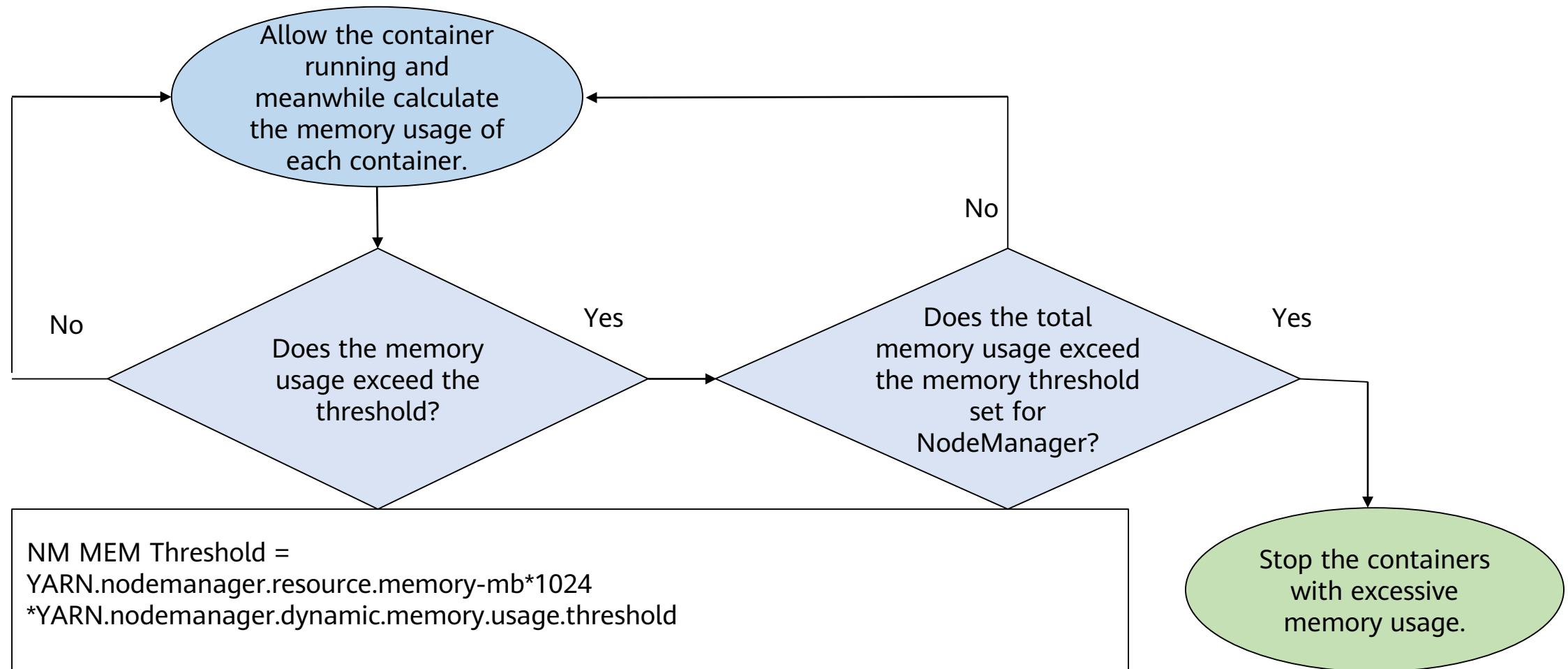
- A scheduler maintains queue information. Users can submit applications to one or more queues.
- Each time NodeManager sends a heartbeat message, the scheduler selects a queue based on the rules, selects an application from the queue, and attempts to allocate resources to the application.
- The scheduler responds to the requests for local resources, intra-rack resources, and resources on any server in sequence.



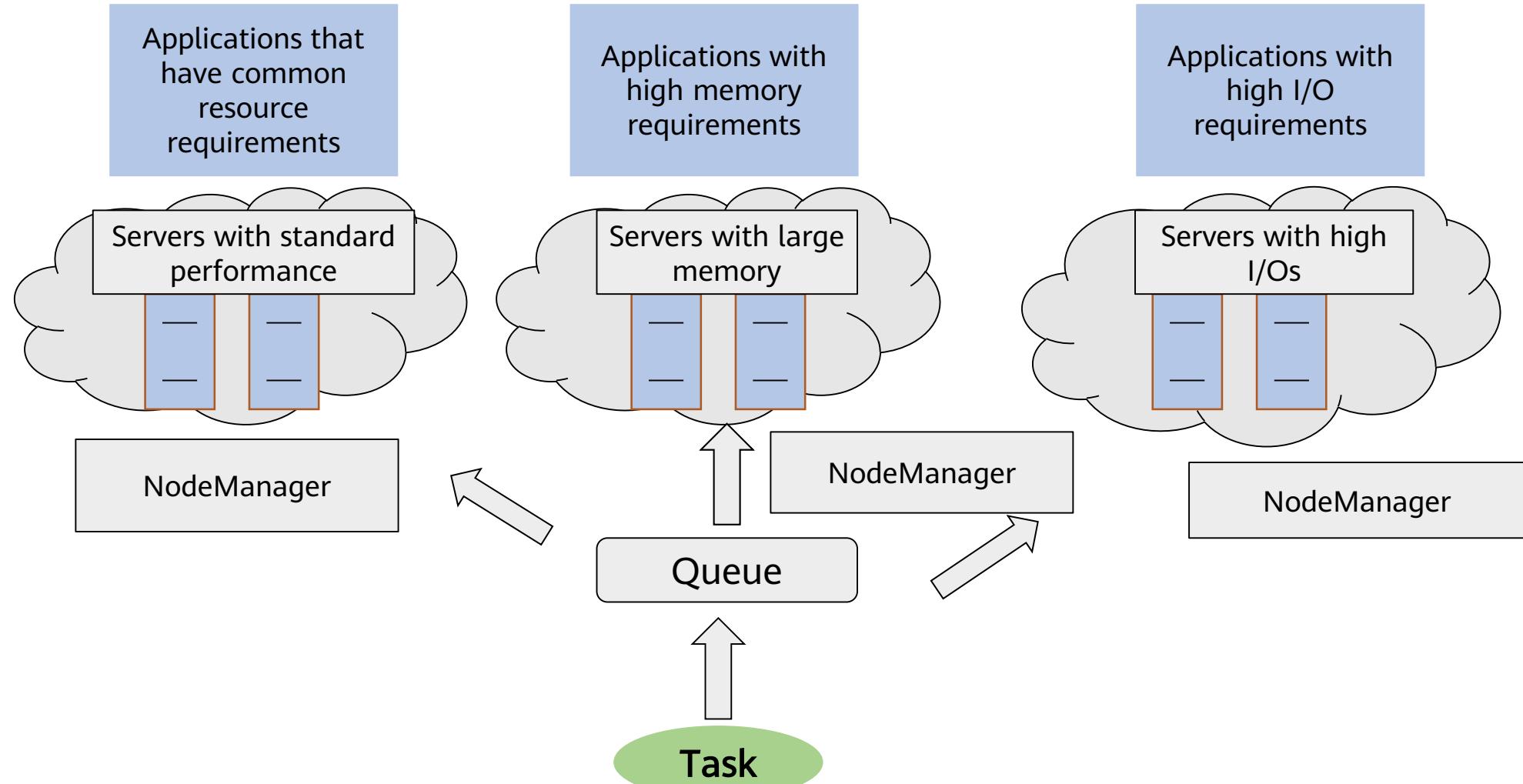
# Contents

1. Introduction to MapReduce and YARN
2. Functions and Architectures of MapReduce and YARN
3. Resource Management and Task Scheduling of YARN
- 4. Enhanced Features**

# Enhanced Feature - Dynamic Memory Management of YARN



# Enhanced Feature - Label-based YARN Scheduling



# Summary

- This course describes the application scenarios and basic architectures of MapReduce and YARN, principles of YARN resource management and task scheduling, as well as enhanced features of YARN in Huawei MRS clusters.

# Quiz

1. (Multiple-choice) What are highlights of MapReduce? ( )
  - A. Easy to program
  - B. Outstanding scalability
  - C. Real-time computing
  - D. High fault tolerance

# Quiz

2. (Single-choice) What is an abstraction of YARN resources? ( )
  - A. Memory
  - B. CPU
  - C. Container
  - D. Disk space

# Quiz

3. (Single-choice) What does MapReduce apply to? (    )
- A. Iterative computing
  - B. Offline computing
  - C. Real-time interactive computing
  - D. Stream computing

# Quiz

4. (Multiple-choice) What are the features of the capacity scheduler? ( )
- A. Capacity assurance
  - B. Flexibility
  - C. Multi-leasing
  - D. Dynamic update of configuration files

# Recommendations

---

- Huawei Cloud Official Web Link:
  - <https://www.huaweicloud.com/intl/en-us/>
- Huawei MRS Documentation:
  - <https://www.huaweicloud.com/intl/en-us/product/mrs.html>
- Huawei TALENT ONLINE:
  - <https://e.huawei.com/en/talent/#/>

Thank you.



# Chapter 6 Spark - An In-Memory Distributed Computing Engine



# Foreword

---

- This course describes the basic concepts of Spark and the similarities and differences between the Resilient Distributed Dataset (RDD), DataSet, and DataFrame data structures in Spark. Additionally, you can understand the features of Spark SQL, Spark Streaming, and Structured Streaming.

# Objectives

- Upon completion of this course, you will be able to:
  - Understand application scenarios and master highlights of Spark.
  - Master the programming capability and technical framework of Spark.

# Contents

- 1. Spark Overview**
2. Spark Data Structure
3. Spark Principles and Architecture

# Introduction to Spark

- Apache Spark was developed in the UC Berkeley AMP lab in 2009.
- It is a fast, versatile, and scalable memory-based big data computing engine.
- As a one-stop solution, Apache Spark integrates batch processing, real-time streaming, interactive query, graph programming, and machine learning.

# Application Scenarios of Spark

- Batch processing can be used for extracting, transforming, and loading (ETL).
- Machine learning can be used to automatically determine whether e-Bay buyers comments are positive or negative.
- Interactive analysis can be used to query the Hive warehouse.
- Streaming processing can be used for real-time businesses such as page-click stream analysis, recommendation systems, and public opinion analysis.

# Highlights of Spark

**Lightweight**

Spark core code has 30,000 lines.

**Fast**

Delay for small datasets reaches the sub-second level.

**Flexible**

Spark offers different levels of flexibility.

**Smart**

Spark smartly uses existing big data components.

# Spark and MapReduce

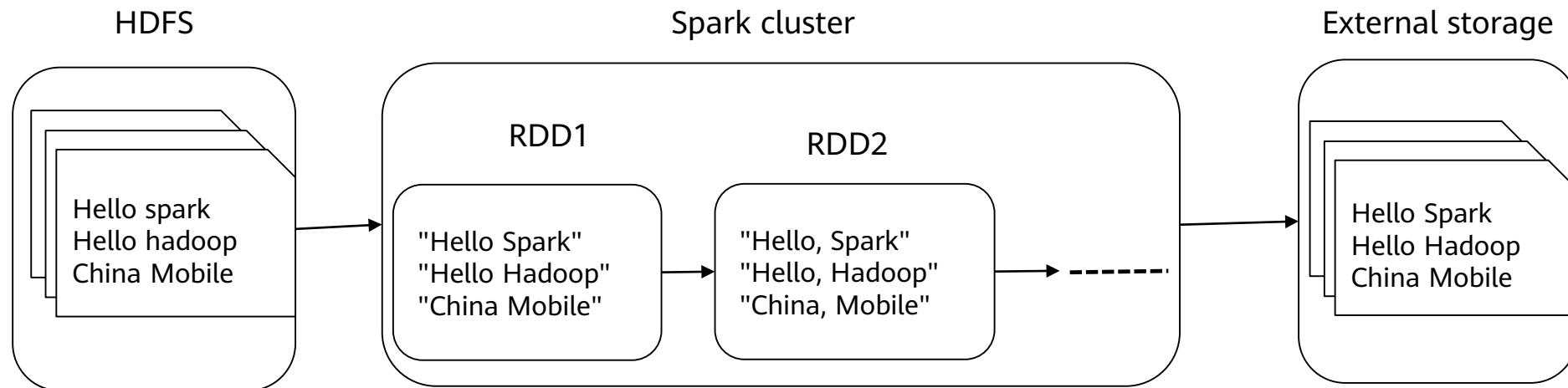
	Hadoop	Spark	Spark
Data size	102.5 TB	102 TB	1,000 TB
Consumed time (mins)	72	23	234
Nodes	2,100	206	190
Cores	50,400	6,592	6,080
Rate	1.4 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min
Daytona Gray Sort	Yes	Yes	Yes

# Contents

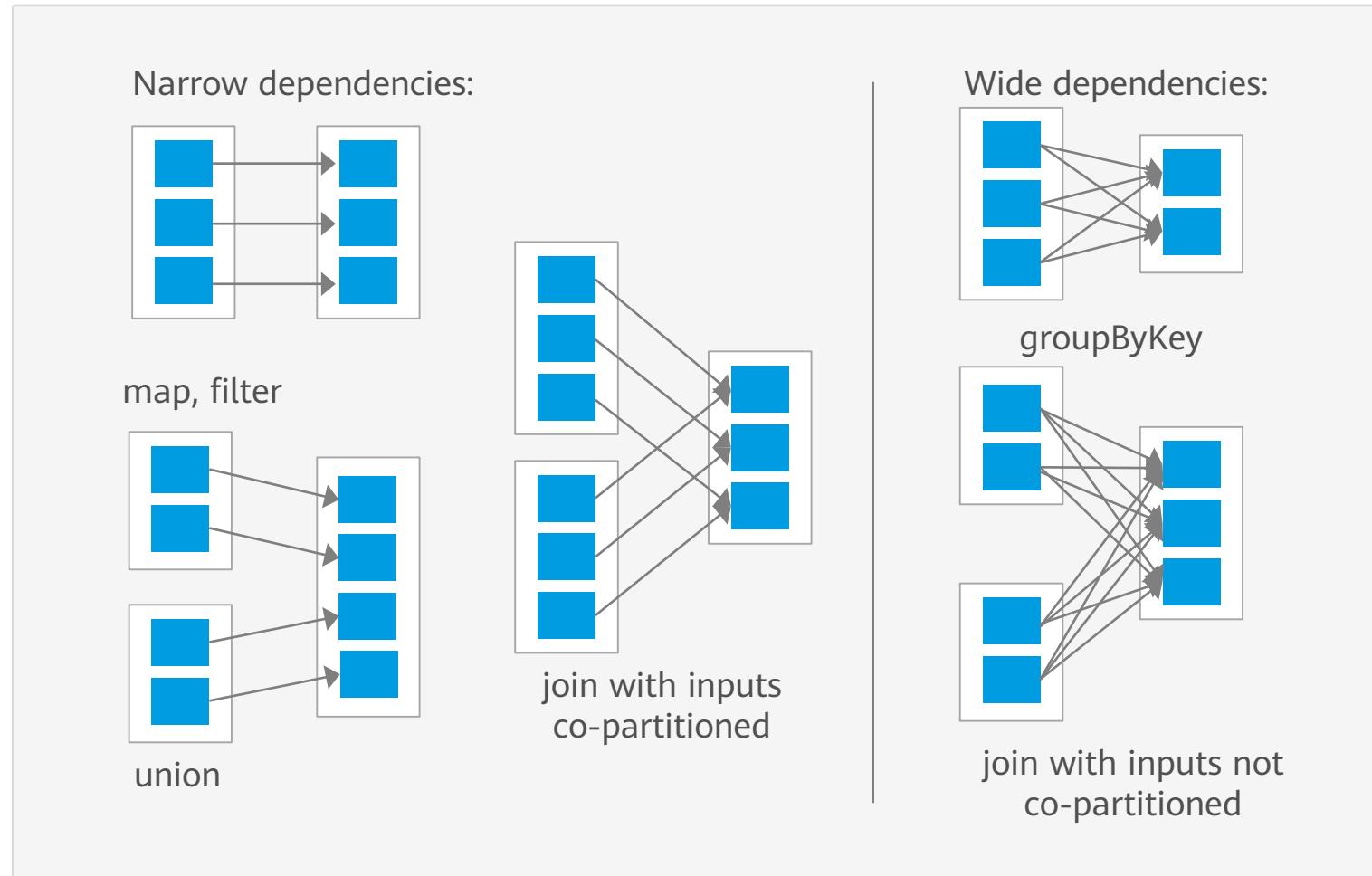
1. Spark Overview
- 2. Spark Data Structure**
3. Spark Principles and Architecture

# RDD - Core Concept of Spark

- Resilient Distributed Datasets (RDDs) are elastic, read-only, and partitioned distributed datasets.
- RDDs are stored in the memory by default and are written to disks when the memory is insufficient.
- RDD data is stored in clusters as partitions.
- RDDs have a lineage mechanism, which allows for rapid data recovery when data loss occurs.



# RDD Dependencies



# Differences Between Wide and Narrow Dependencies

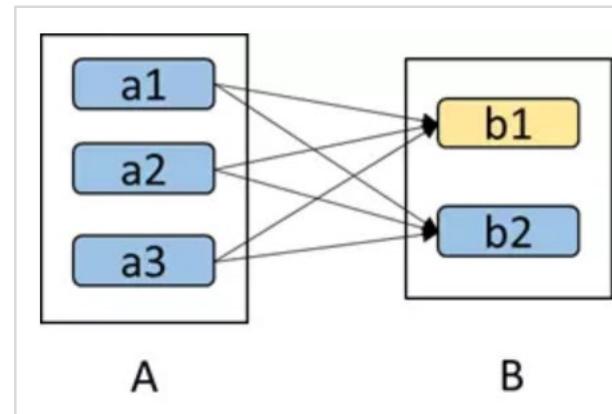
## - Operator

- Narrow dependency indicates that each partition of a parent RDD can be used by at most one partition of a child RDD, for example, map, filter, and union.
- Wide dependency indicates that the partitions of multiple child RDDs depend on the partition of the same parent RDD, for example, groupByKey, reduceByKey, and sortByKey.

# Differences Between Wide and Narrow Dependencies

## - Fault Tolerance

- If a node is faulty:
  - Narrow dependency: Only the parent RDD partition corresponding to the child RDD partition needs to be recalculated.
  - Wide dependency: In extreme cases, all parent RDD partitions need to be recalculated.
- As shown in the following figure, if the b1 partition is lost, a1, a2, and a3 need to be recalculated.

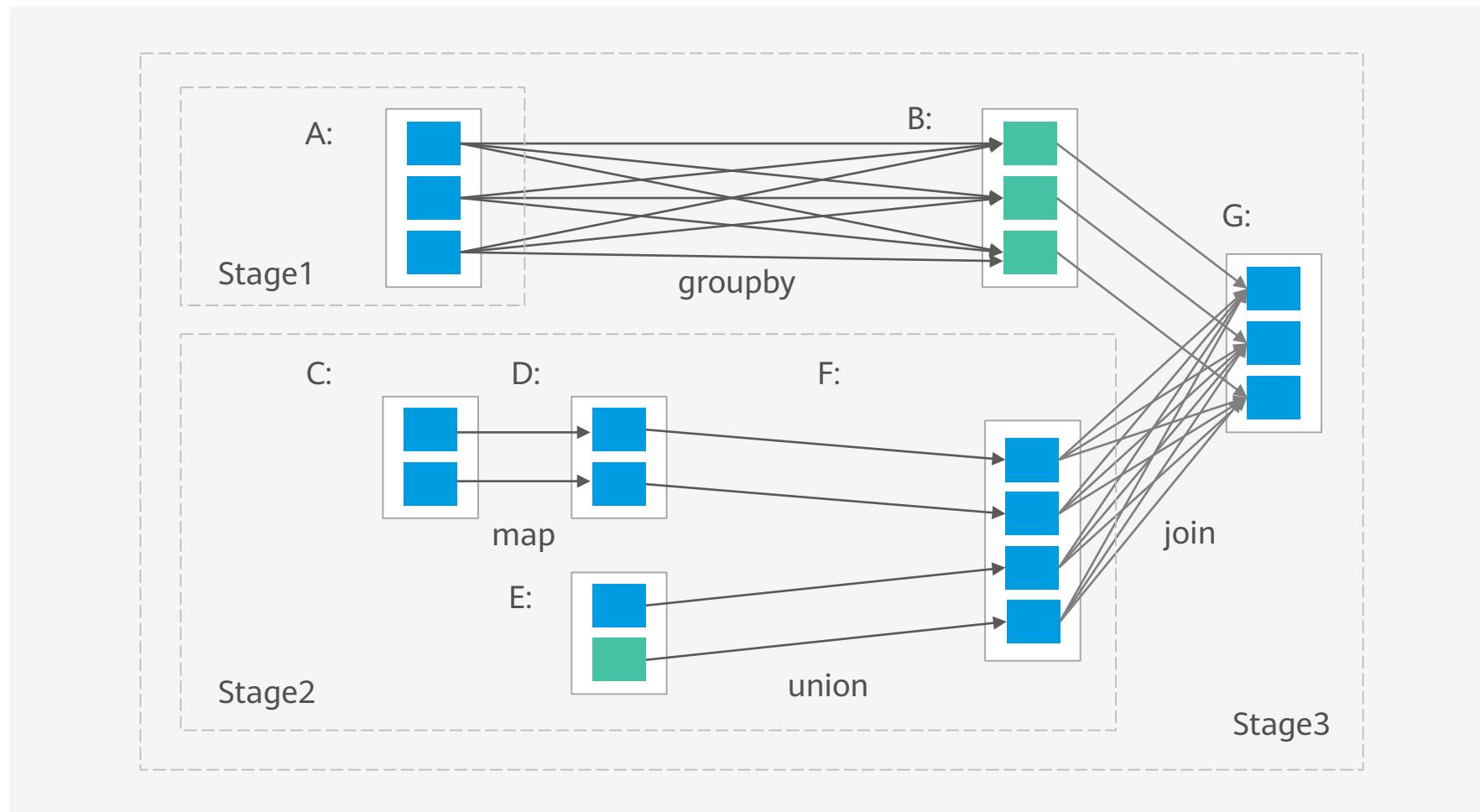


# Differences Between Wide and Narrow Dependencies

## - Data Transmission

- Wide dependency usually corresponds to shuffle operations. During the running process, the partition of the same parent RDD needs to be transferred to different child RDD partitions, which may involve data transmission between multiple nodes.
- The partition of each parent RDD on which narrow dependency exists is transferred to only one child RDD partition. Generally, the conversion can be completed on one node.

# Stage Division of RDD



# RDD Operation Type

- Spark operations can be classified into creation, conversion, control, and behavior operations.
  - Creation operation: used to create an RDD. An RDD is created through a memory collection and an external storage system or by a transformation operation.
  - Transformation operation: An RDD is transformed into a new RDD through certain operations. The transformation operation of the RDD is a lazy operation, which only defines a new RDD but does not execute it immediately.
  - Control operation: RDD persistence is performed. An RDD can be stored in the disk or memory based on different storage policies. For example, the cache API caches the RDD in the memory by default.
  - Action operation: An operation that can trigger Spark running. Action operations in Spark are classified into two types. One is to output the calculation result, and the other is to save the RDD to an external file system or database.

# Creation Operation

- Currently, there are two types of basic RDDs:
  - Parallel collection: An existing collection is collected and computed in parallel.
  - External storage: A function is executed on each record in a file. The file system must be HDFS or any storage system supported by Hadoop.
- The two types of RDD can be operated in the same way to obtain a series of extensions such as sub-RDD and form a lineage diagram.

# Control Operation

- Spark can store RDD in the memory or disk file system persistently. The RDD in memory can greatly improve iterative computing and data sharing between computing models. Generally, 60% of the memory of the execution node is used to cache data, and the remaining 40% is used to run tasks. In Spark, **persist** and **cache** operations are used for data persistency. Cache is a special case of **persist()**.

# Transformation Operation - Transformation Operator

Transformation	Description
map(func)	Uses the <b>func</b> method to generate a new RDD for each element in the RDD that invokes <b>map</b> .
filter(func)	<b>func</b> is used for each element of an RDD that invokes <b>filter</b> and then an RDD with elements containing <b>func</b> (the value is <b>true</b> ) is returned.
reduceByKey(func, [numTasks])	It is similar to <b>groupByKey</b> . However, the value of each key is calculated based on the provided <b>func</b> to obtain a new value.
join(otherDataset, [numTasks])	If the data set is <b>(K, V)</b> and the associated data set is <b>(K, W)</b> , then <b>(K, (V, W))</b> is returned. <b>leftOuterJoin</b> , <b>rightOuterJoin</b> , and <b>fullOuterJoin</b> are supported.

# Action Operation - Action Operator

Action	Description
reduce(func)	Aggregates elements in a dataset based on functions.
collect()	Used to encapsulate the <b>filter</b> result or a small enough result and return an array.
count()	Collects statistics on the number of elements in an RDD.
first()	Obtains the first element of a dataset.
take(n)	Obtains the top elements of a dataset and returns an array.
saveAsTextFile(path)	This API is used to write the dataset to a text file or HDFS. Spark converts each record into a row and writes the row to the file.

# DataFrame

- Similar to RDD, DataFrame is also an invariable, elastic, and distributed dataset. In addition to data, it records the data structure information, which is known as schema. The schema is similar to a two-dimensional table.
- The query plan of DataFrame can be optimized by Spark Catalyst Optimizer. Spark is easy to use, which improves high-quality execution efficiency. Programs written by DataFrame can be converted into efficient forms for execution.

# DataSet

- DataFrame is a special case of DataSet (**DataFrame=Dataset[Row]**). Therefore, you can use the `as` method to convert DataFrame to DataSet. **Row** is a common type. All table structure information is represented by row.
- DataSet is a strongly typed dataset, for example, **Dataset[Car]** and **Dataset[Person]**.

# Differences Between DataFrame, DataSet, and RDD

- Assume that there are two lines of data in an RDD, which are displayed as follows:

1, Allen, 23
2, Bobby, 35

- The data in DataFrame is displayed as follows:

ID:String	Name:String	Age:int
1	Allen	23
2	Bobby	35

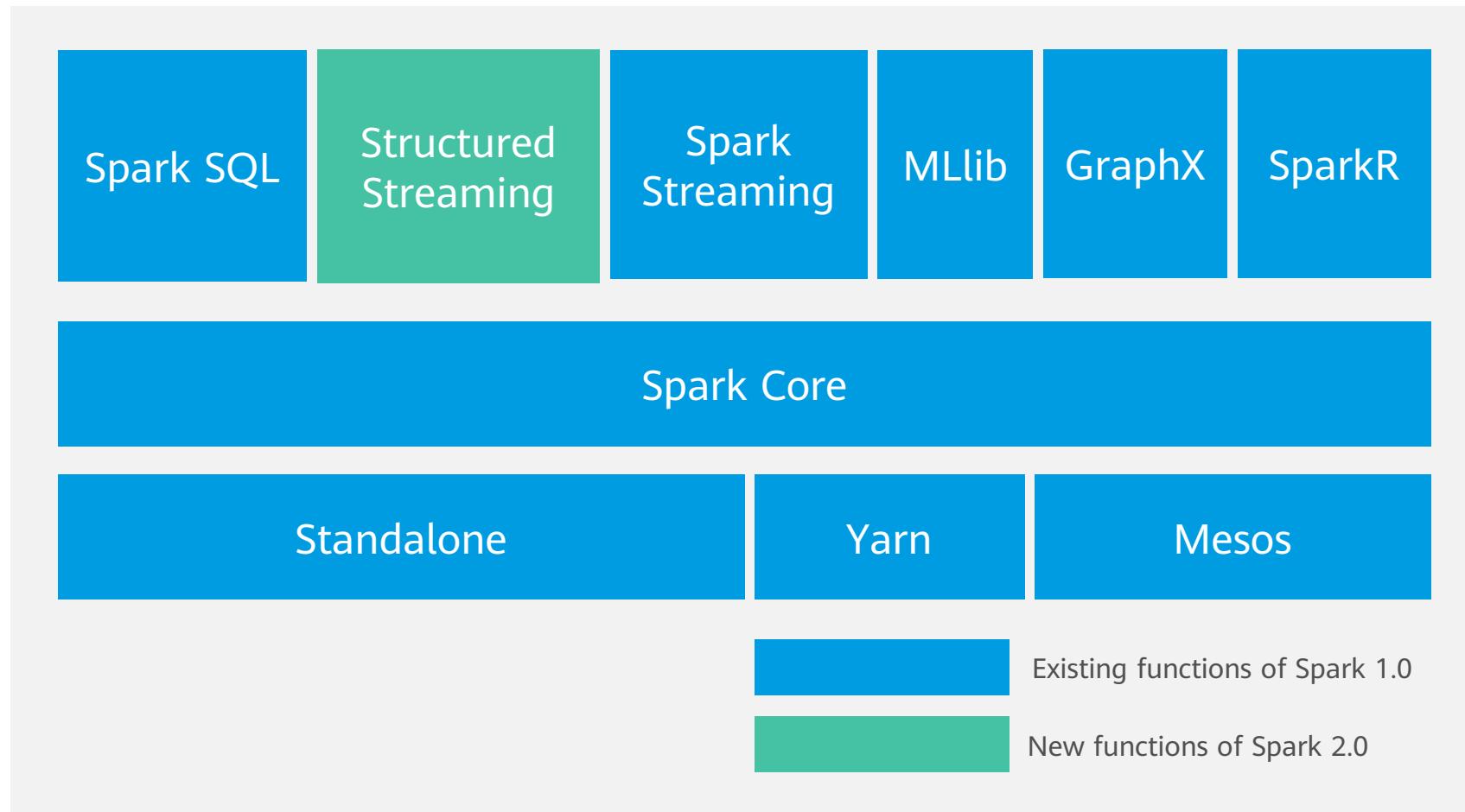
- The data in DataSet is displayed as follows:

<b>value:People[age: bigint, id: bigint, name:string]</b>
People(id=1, name="Allen", age=23)
People(id=1, name="Bobby", age=35) [283]

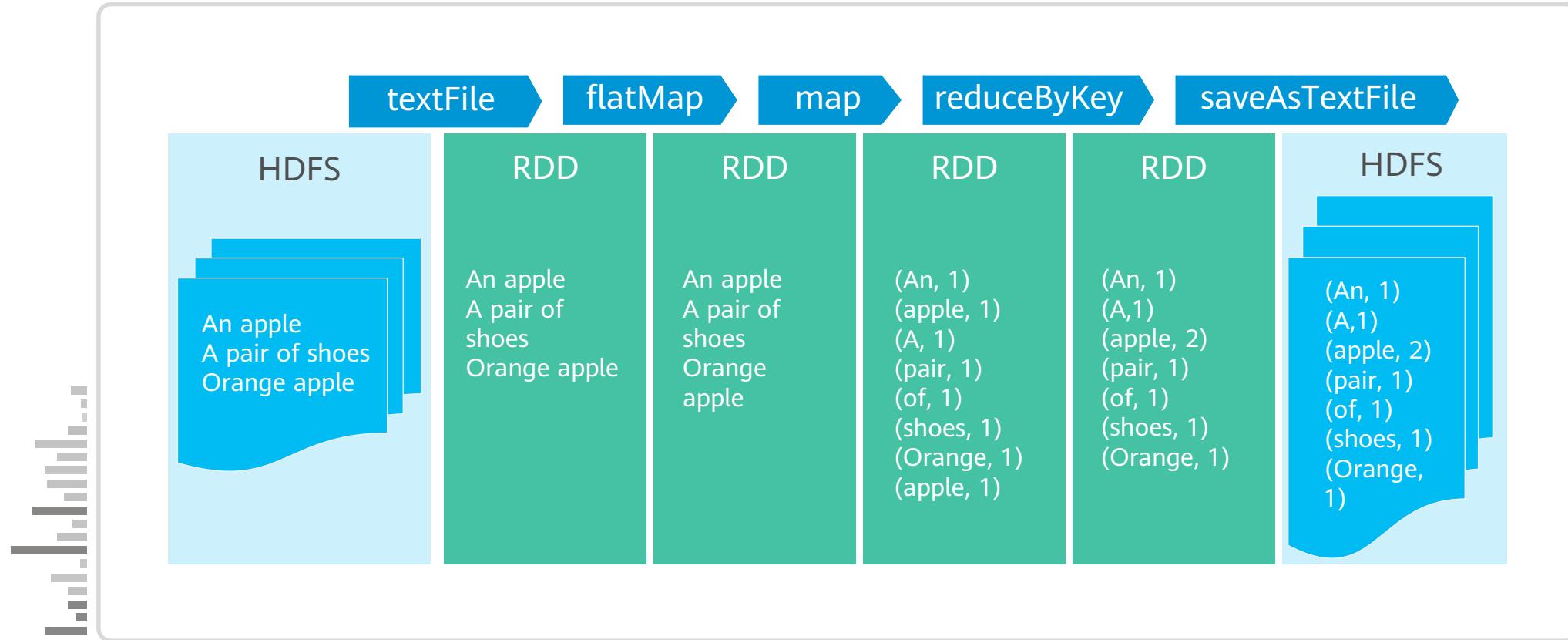
# Contents

1. Spark Overview
2. Spark Data Structure
3. **Spark Principles and Architecture**
  - Spark Core
    - Spark SQL and DataSet
    - Spark Structured Streaming
    - Spark Streaming

# Spark System Architecture



# Typical Case - WordCount



# WordCount

```
object WordCount
{
    def main (args: Array[String]): Unit = {
        //Configuring the Spark application name.
        val conf = new
        SparkConf().setAppName("WordCount")
        val sc: SparkContext = new SparkContext(conf)
        val textFile = sc.textFile("hdfs://...")
        val counts = textFile.flatMap(line => line.split(" "))
            .map(word => (word, 1))
            .reduceByKey(_ + _)
        counts.saveAsTextFile("hdfs://...")
    }
}
```

Create a SparkContext object and set the application name to Wordcount.

Load the text file on HDFS to obtain the RDD data set.

Invoke RDD transformation for calculations. Split the text file by spaces, set the occurrence frequency of each word to 1, and then combine the counts by the same key. This operation is performed on each executor.

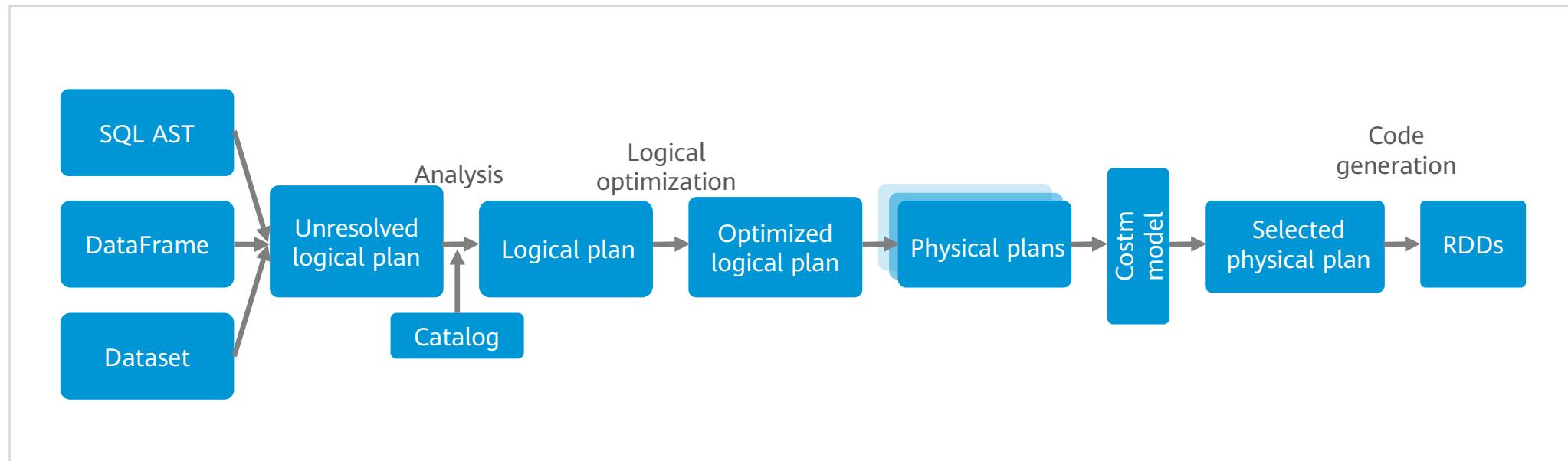
Invoke the **saveAsTextFile** action and save the result. This line triggers the actual task execution.

# Contents

1. Spark Overview
2. Spark Data Structure
3. **Spark Principles and Architecture**
  - Spark Core
  - **Spark SQL and DataSet**
  - Spark Structured Streaming
  - Spark Streaming

# Spark SQL Overview

- Spark SQL is the module used in Spark for structured data processing. In Spark applications, you can seamlessly use SQL statements or DataFrame APIs to query structured data.



# Spark SQL and Hive

- Differences:
  - The execution engine of Spark SQL is Spark Core, and the default execution engine of Hive is MapReduce.
  - The execution speed of Spark SQL is 10 to 100 times faster than Hive.
  - Spark SQL does not support buckets, but Hive supports.
- Relationships:
  - Spark SQL depends on the metadata of Hive.
  - Spark SQL is compatible with most syntax and functions of Hive.
  - Spark SQL can use the custom functions of Hive.

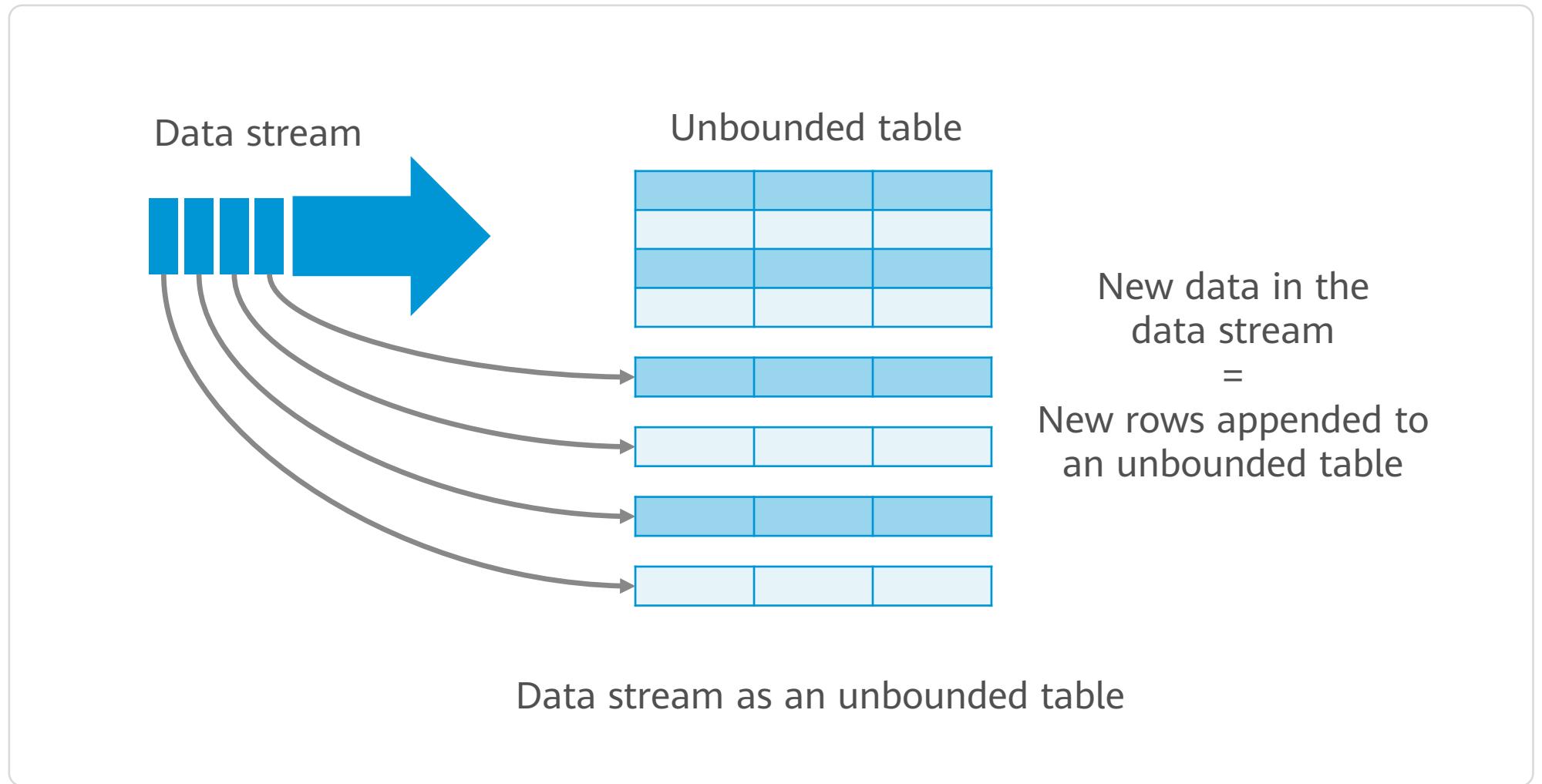
# Contents

1. Spark Overview
2. Spark Data Structure
3. **Spark Principles and Architecture**
  - Spark Core
  - Spark SQL and DataSet
  - **Spark Structured Streaming**
  - Spark Streaming

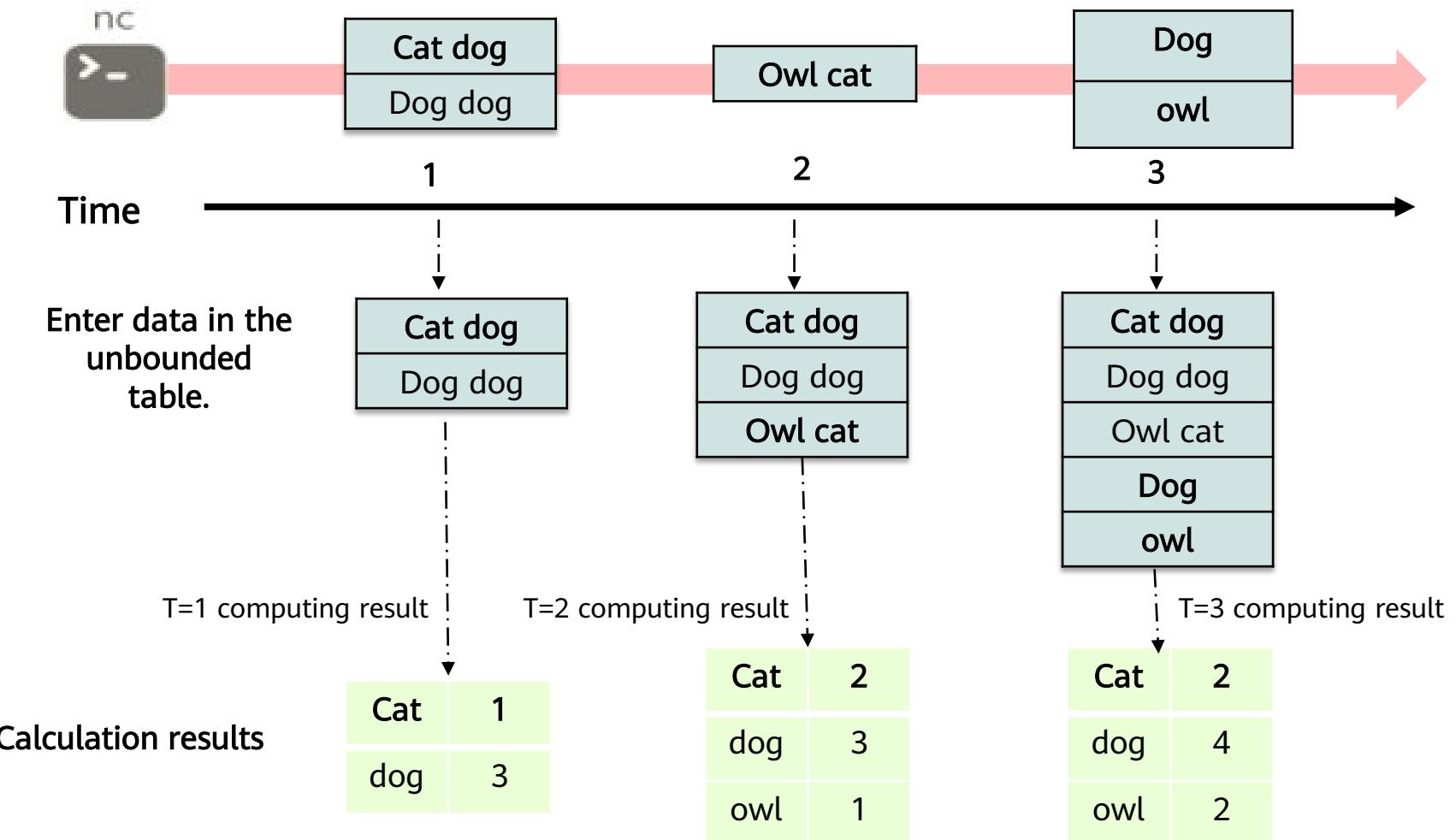
# Structured Streaming Overview (1)

- Structured Streaming is a streaming data-processing engine built on the Spark SQL engine. You can compile a streaming computing process like using static RDD data. When streaming data is continuously generated, Spark SQL will process the data incrementally and continuously, and update the results to the result set.

# Structured Streaming Overview (2)



# Example Programming Model of Structured Streaming

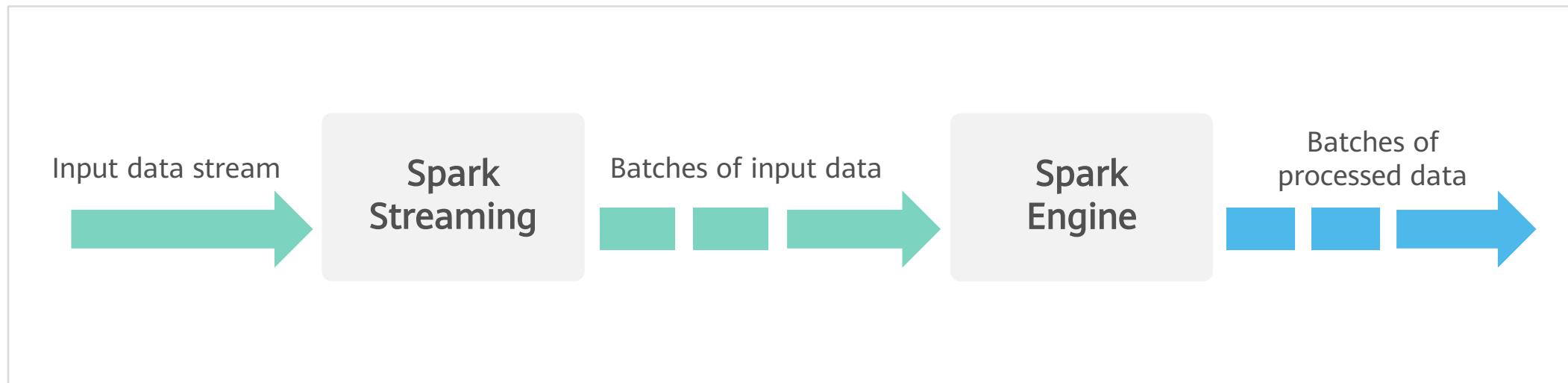


# Contents

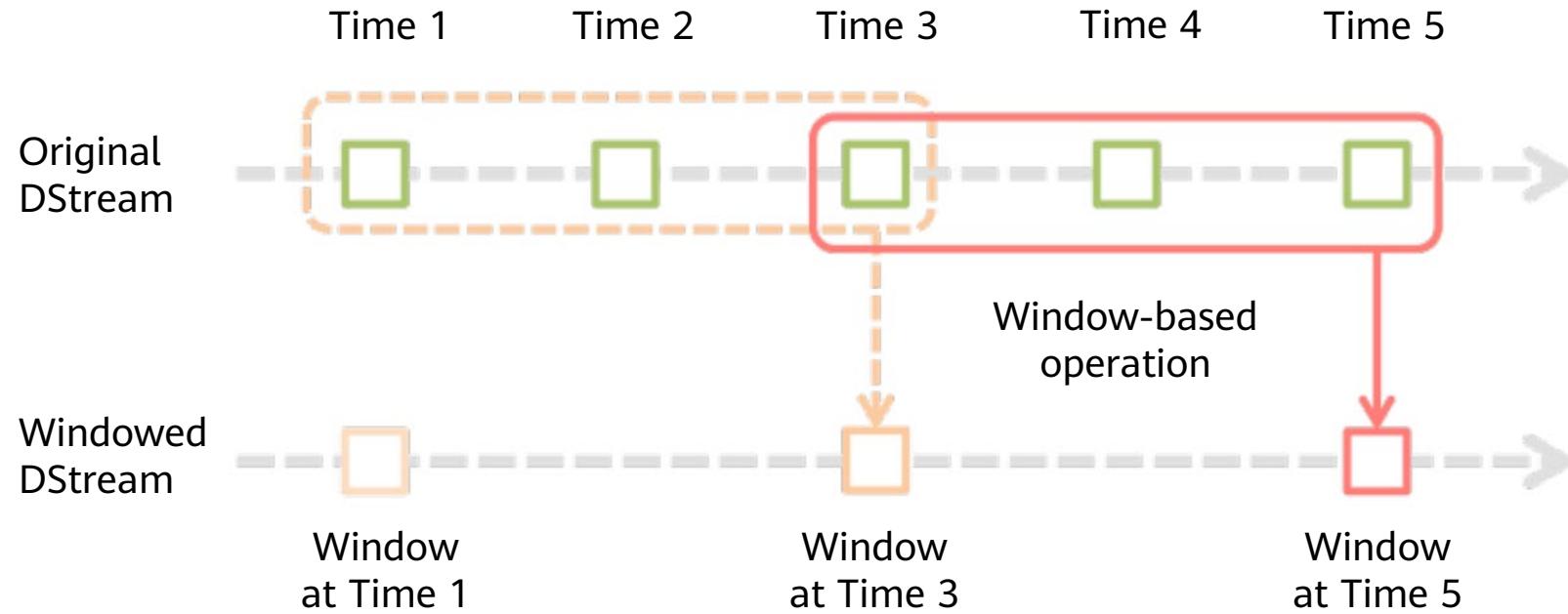
1. Spark Overview
2. Spark Data Structure
3. **Spark Principles and Architecture**
  - Spark Core
  - Spark SQL and DataSet
  - Spark Structured Streaming
  - **Spark Streaming**

# Spark Streaming Overview

- The basic principle of Spark Streaming is to split real-time input data streams by time slice (in seconds), and then use the Spark engine to process data of each time slice in a way, which is similar to batch processing.



# Window Interval and Sliding Interval



- The window slides on the Dstream. The RDDs that fall within the window are merged and operated to generate a window-based RDD.
  - **Window length:** indicates the duration of a window.
  - **Sliding window interval:** indicates the interval for performing window operations.

# Comparison Between Spark Streaming and Storm

Item	Storm	Spark Streaming
Real-time computing model	Real time. A piece of data is processed once it is requested.	Quasi real time. Data within a time range is collected and processed as an RDD.
Real-time computing delay	In milliseconds	In seconds
Throughput	Low	High
Transaction mechanism	Supported and perfect	Supported but not perfect
Fault tolerance	High for ZooKeeper and Acker	Normal for Checkpoint and WAL
Dynamic adjustment parallelism	Supported	Not supported

# Summary

- This course describes the basic concepts and technical architecture of Spark, including SparkSQL, Structured Streaming, and Spark Streaming.

# Quiz

1. What are the features of Spark?
2. What are the advantages of Spark in comparison with MapReduce?
3. What are the differences between wide dependencies and narrow dependencies of Spark?
4. What are the application scenarios of Spark?

# Quiz

5. RDD operators are classified into: \_\_\_\_\_ and \_\_\_\_\_.
6. The \_\_\_\_\_ module is the core module of Spark.
7. RDD dependency types include \_\_\_\_\_ and \_\_\_\_\_.

# Recommendations

---

- Huawei Cloud Official Web Link:
  - <https://www.huaweicloud.com/intl/en-us/>
- Huawei MRS Documentation:
  - <https://www.huaweicloud.com/intl/en-us/product/mrs.html>
- Huawei TALENT ONLINE:
  - <https://e.huawei.com/en/talent/#/>

# Thank you.

把数字世界带入每个人、每个家庭、  
每个组织，构建万物互联的智能世界。  
Bring digital to every person, home, and  
organization for a fully connected,  
intelligent world.

Copyright©2020 Huawei Technologies Co., Ltd.  
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.



# Chapter 7 Flink, Stream and Batch Processing in a Single Engine



# Objectives

On completion of this course, you will be able to:

- Master the core technologies and architecture of Flink.
- Understand the time and window mechanisms of Flink.
- Know the fault tolerance mechanism of Flink.

# Contents

- 1. Principles and Architecture of Flink**
2. Flink Time & Window
3. Flink Watermark
4. Fault Tolerance of Flink

# Overview

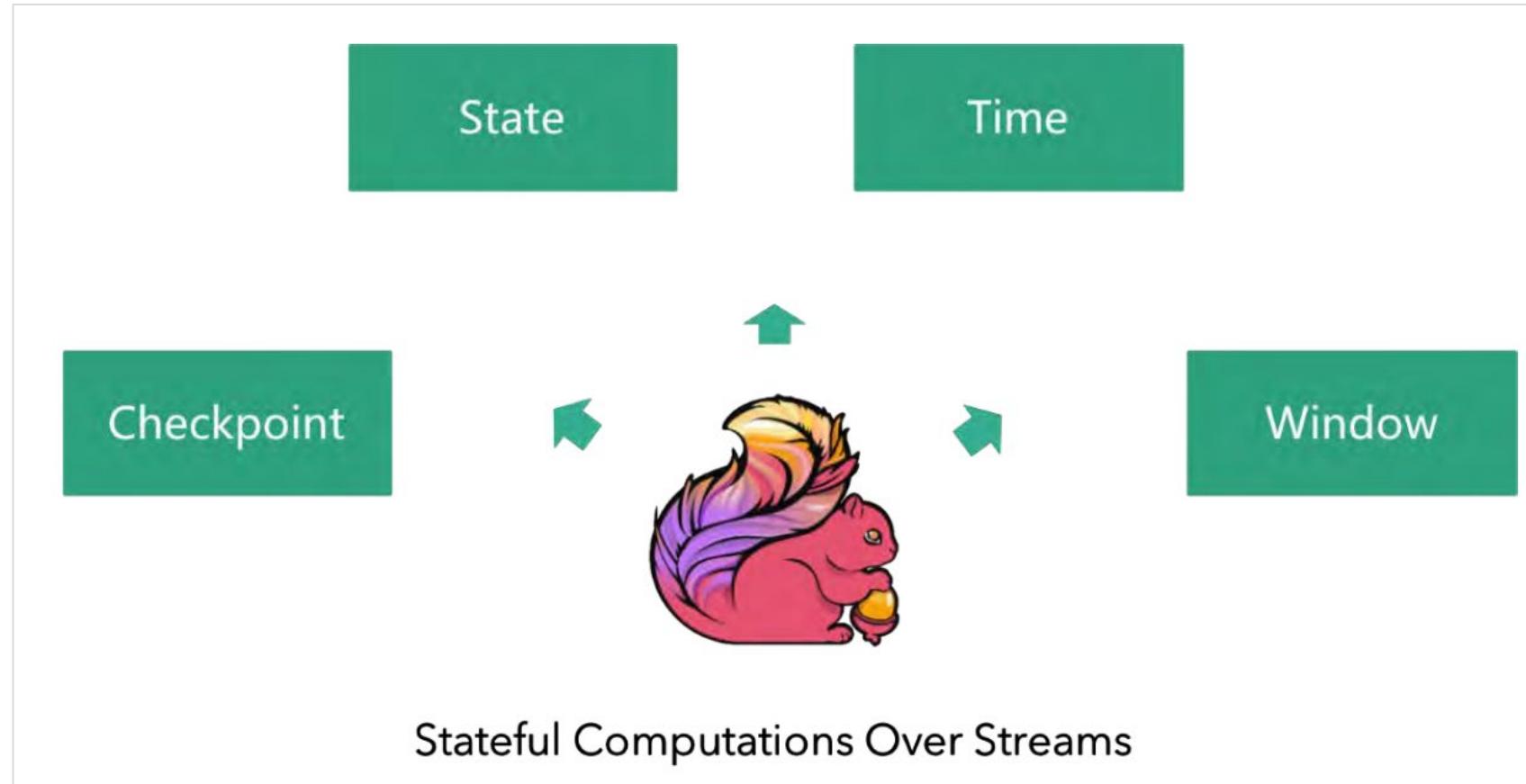
- Apache Flink is an open-source stream processing framework for distributed, high-performance stream processing applications. Flink not only supports real-time computing with both high throughput and exactly-once semantics, but also provides batch data processing.
- Compared with other data processing engines in the market, Flink and Spark support both stream processing and batch processing. From the perspective of the technical philosophy, Spark simulates stream computing based on batch computing. Flink is the opposite. It simulates batch computing based on stream computing.



Flink

# Key Concepts

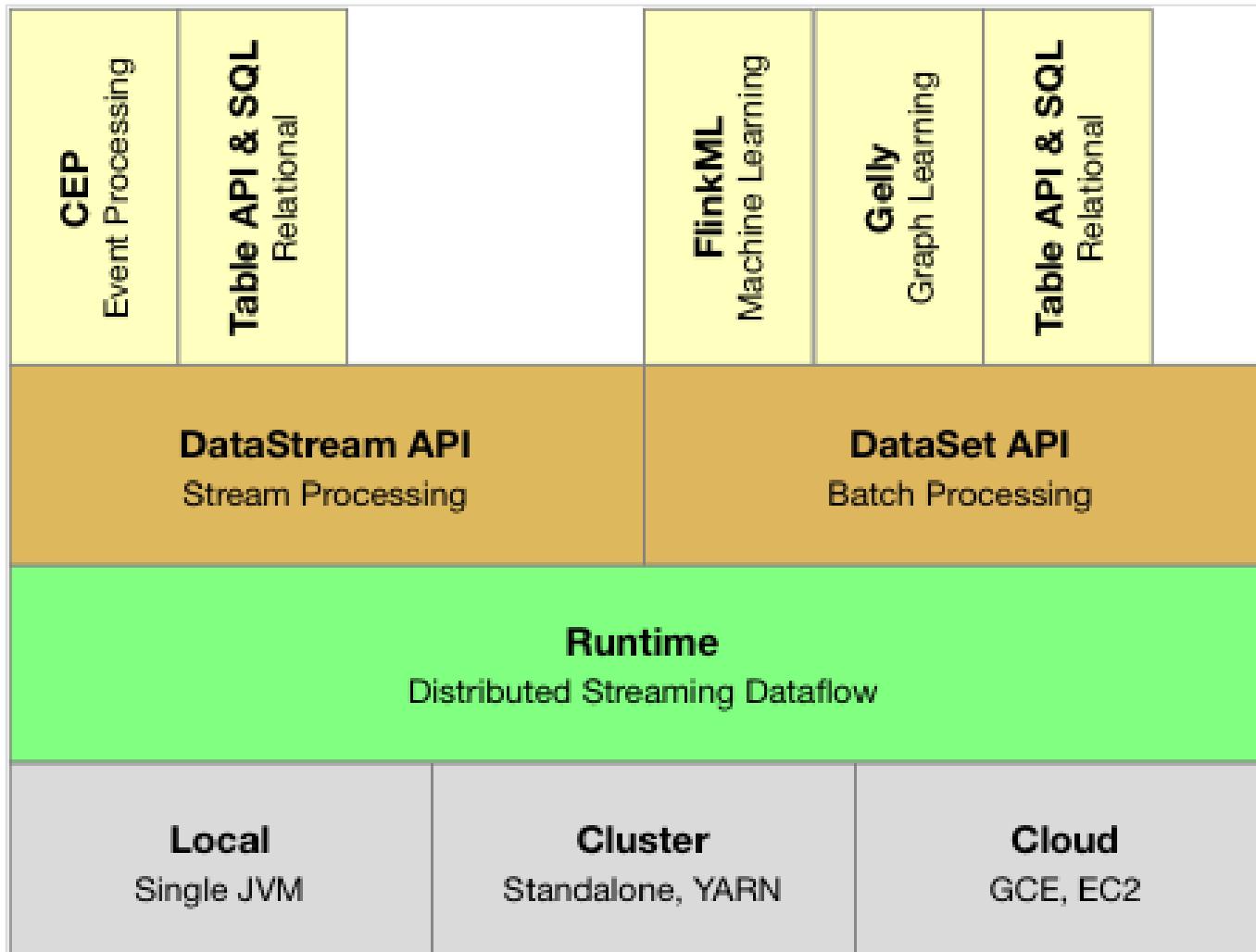
- Continuous processing of streaming data, event time, stateful stream processing, and state snapshots



# Core Ideas

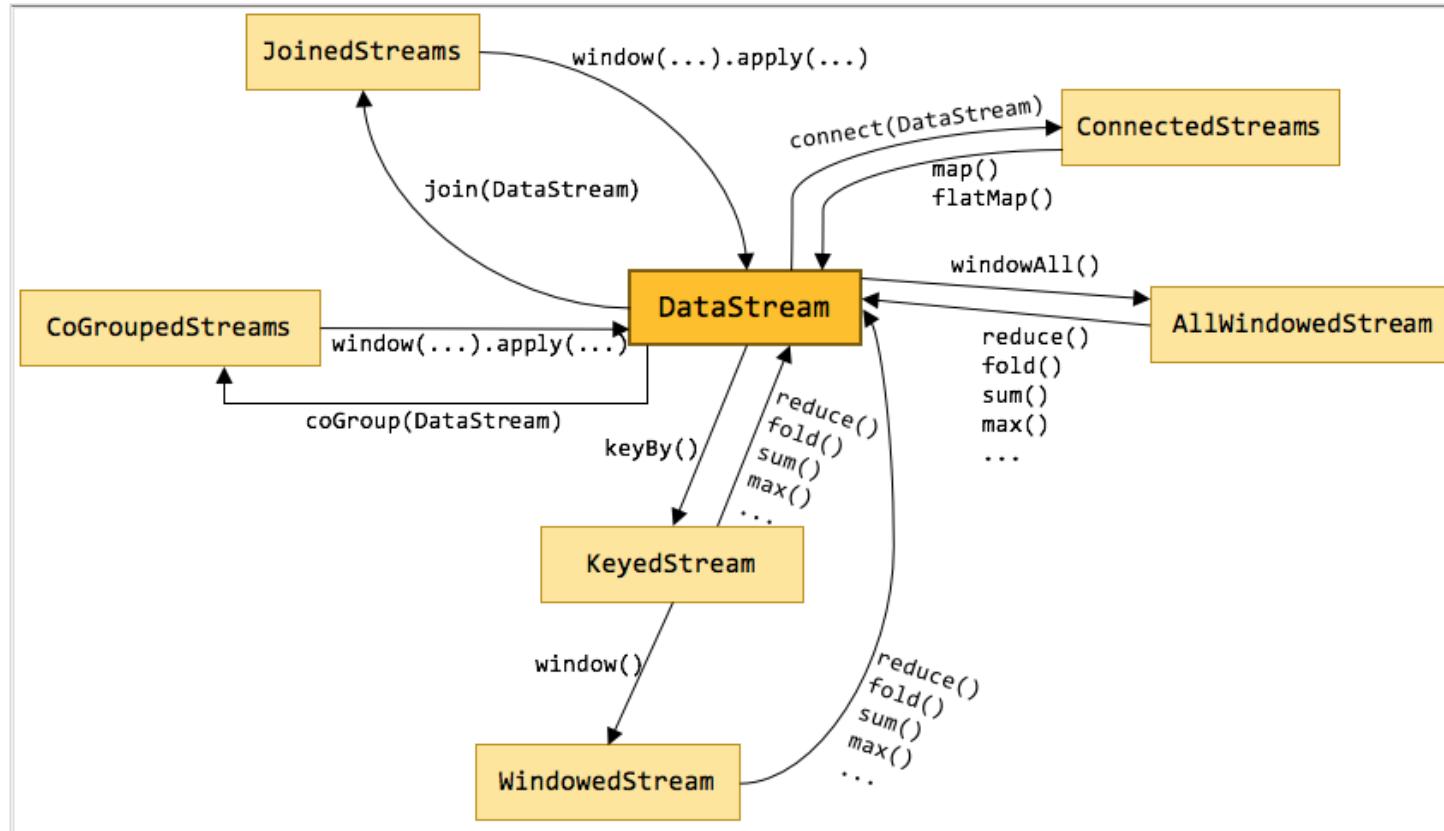
- The biggest difference between Flink and other stream computing engines is state management.
- Flink provides built-in state management. You can store states in Flink instead of storing them in an external system. Doing this can:
  - Reduce the dependency of the computing engine on external systems, simplifying deployment and O&M.
  - Significantly improve performance.

# Overall Architecture of Flink Runtime



# Key Concept - DataStream

- DataStream: represents streaming data. You can think of DataStream as immutable collections of data that can contain duplicates. The number of elements in DataStream are unlimited.

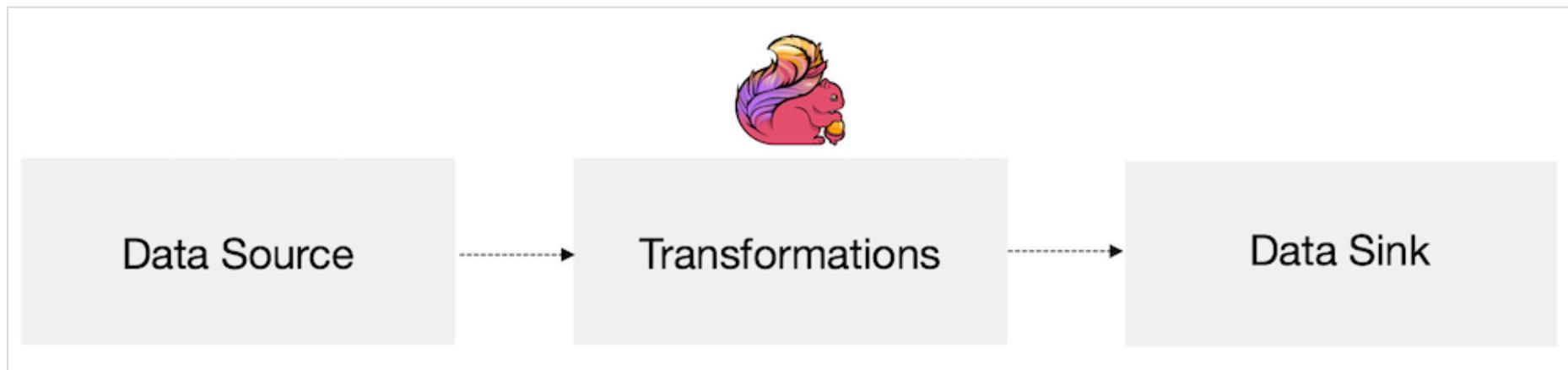


# Key Concept - DataSet

- DataSet programs in Flink are regular programs that implement transformations on data sets (for example, filtering, mapping, joining, and grouping). The data sets are initially created by reading files or from local collections. Results are returned via sinks, which can write the data to (distributed) files, or to standard output (for example, the command line terminal).

# Flink Program

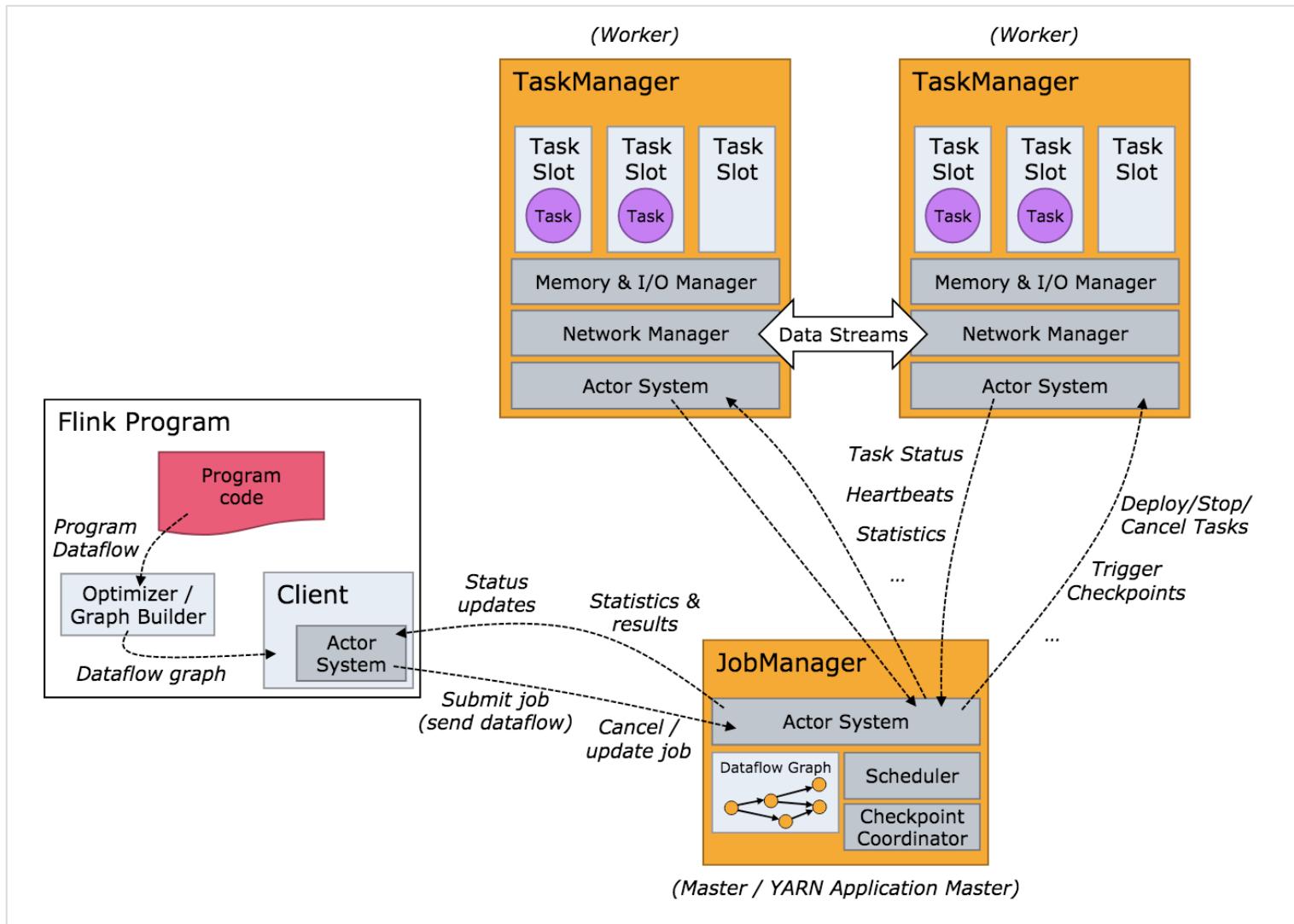
- Flink programs consist of three parts: Source, Transformation, and Sink. Source is responsible for reading data from data sources such as HDFS, Kafka, and text. Transformation is responsible for data transformation operations. Sink is responsible for final data outputs (such as HDFS, Kafka, and text). Data that flows between parts is called stream.



# Flink Data Sources

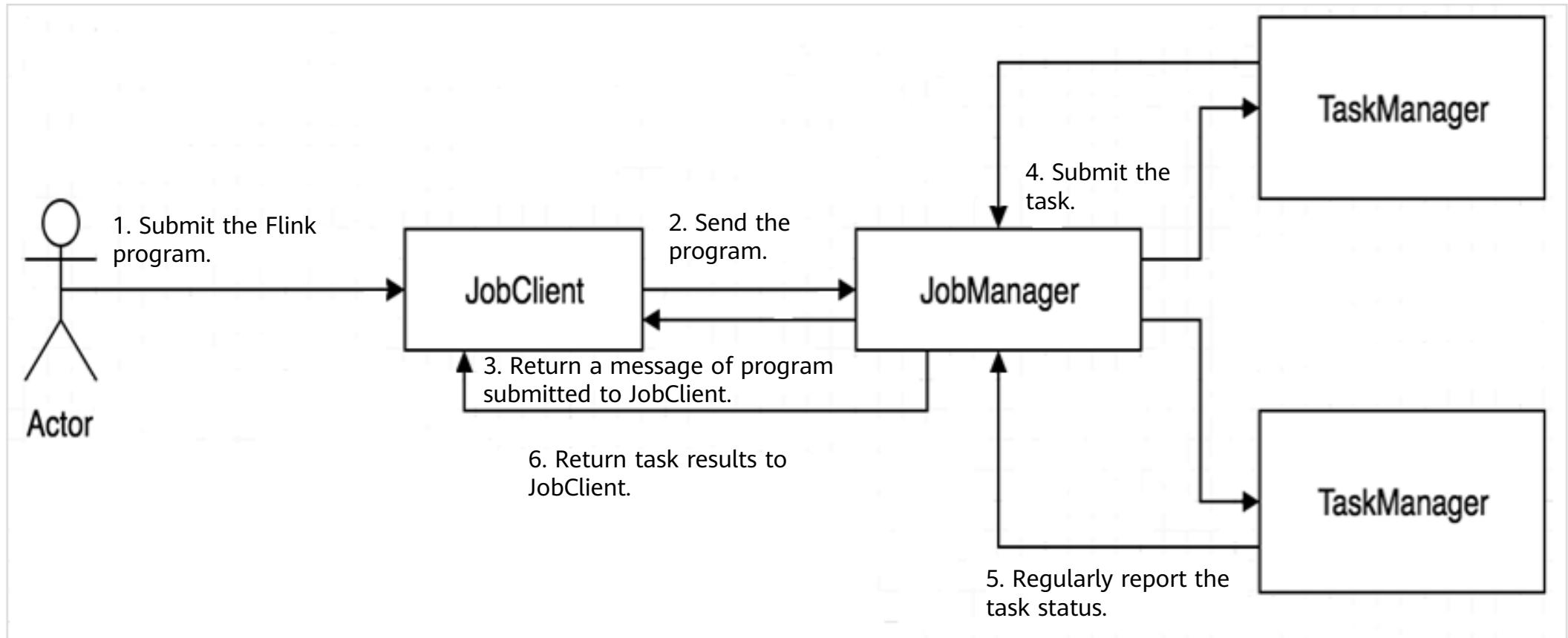
- Batch processing
  - Files
    - HDFS, Local file system, MapR file system
    - Text, CSV, Avro, Hadoop input formats
  - JDBC
  - HBase
  - Collections
- Stream processing
  - Files
  - JDBC
  - Socket streams
  - Kafka
  - RabbitMQ
  - Flume
  - Collections
  - Implement your own
    - `SourceFunction.collect`

# Flink Program Running Diagram



# Flink Job Running Process (1)

- A user submits a Flink program to JobClient. JobClient processes, parses, and optimizes the program, and then submits the program to JobManager. TaskManager runs the task.



# Flink Job Running Process (2)

- JobClient is a bridge between Flink and JobManager. It mainly receives, parses, and optimizes program execution plans, and submits the plans to JobManager. There are three types of operators in Flink.
  - Source Operator: data source operations, such as file, socket, and Kafka.
  - Transformation Operator: data transformation operations, such as map, flatMap, and reduce.
  - Sink Operator: data storage operations. For example, data is stored in HDFS, MySQL, and Kafka.

# A Complete Flink Program (1)

```
public class Example {  
    public static void main(String[] args) throws Exception {  
        final StreamExecutionEnvironment env  
            =StreamExecutionEnvironment.getExecutionEnvironment();  
        DataStream<Person> flintstones = env.fromElements(  
            new Person("Fred", 35), new Person("Wilma", 35), new Person("Pebbles", 2));  
        DataStream<Person> adults = flintstones.filter(new FilterFunction<Person>() {  
            @Override  
            public boolean filter(Person person) throws Exception {return person.age >= 18;}  
        });  
        adults.print();  
        env.execute();  
    }  
}
```

# A Complete Flink Program (2)

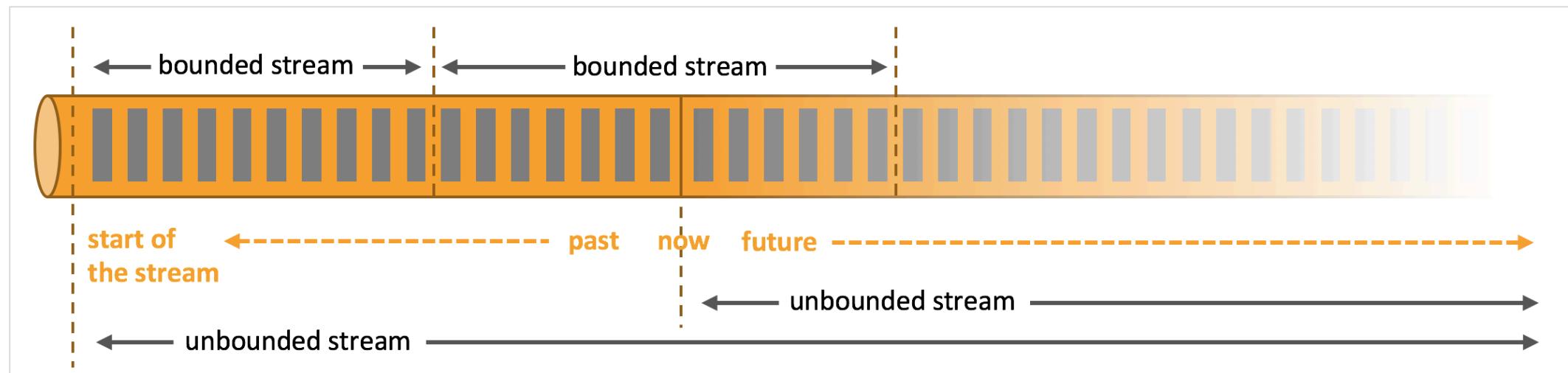
```
public static class Person {  
    public String name;  
    public Integer age;  
    public Person() {};  
    public Person(String name, Integer age) {  
        this.name = name;  
        this.age = age;  
    };  
    public String toString() {  
        return this.name.toString() + ": age " + this.age.toString();  
    };  
};
```

# Data Processing

- Apache Flink supports both batch and stream processing that can be used to create a number of event-based applications.
  - Flink is first of all a pure stream computing engine with data stream as its basic data model. A stream can be infinite and borderless, which describes stream processing in the general sense, or can be a finite stream with boundaries, as in the case of batch processing. Flink thus uses a single architecture to support both stream and batch processing.
  - Flink has the additional strength of supporting stateful computing. Processing is called stateless when the result of processing an event or piece of data is only related to the content of that event itself. Alternatively, if the result is related to previously processed events, it is called stateful processing.

# Bounded Stream and Unbounded Stream

- Unbounded stream: Only the start of a stream is defined. Data sources generate data endlessly. The data of unbounded streams must be processed continuously. That is, the data must be processed immediately after being read. You cannot wait until all data arrives before processing, because the input is infinite and cannot be completed at any time. Processing unbounded stream data typically requires ingesting events in a particular sequence, such as the sequence of the events that occurred, so that the integrity of the results can be inferred.
- Bounded stream: Both the start and end of a stream are defined. Bounded streams can be computed after all data is read. All data in a bounded stream can be sorted, without ingesting events in an order. Bounded stream processing is often referred to as batch processing.



# Batch Processing Example (1)

- Batch processing is a very special case of stream processing. Instead of defining a sliding or tumbling window over the data and producing results every time the window slides, a global window is defined, with all records belonging to the same window. For example, a simple Flink program that counts visitors in a website every hour, grouped by region continuously, is the following:

```
val counts = visits
    .keyBy("region")
    .timeWindow(Time.hours(1))
    .sum("visits")
```

# Batch Processing Example (2)

- If you know that the input data is bounded, you can implement batch processing using the following code:

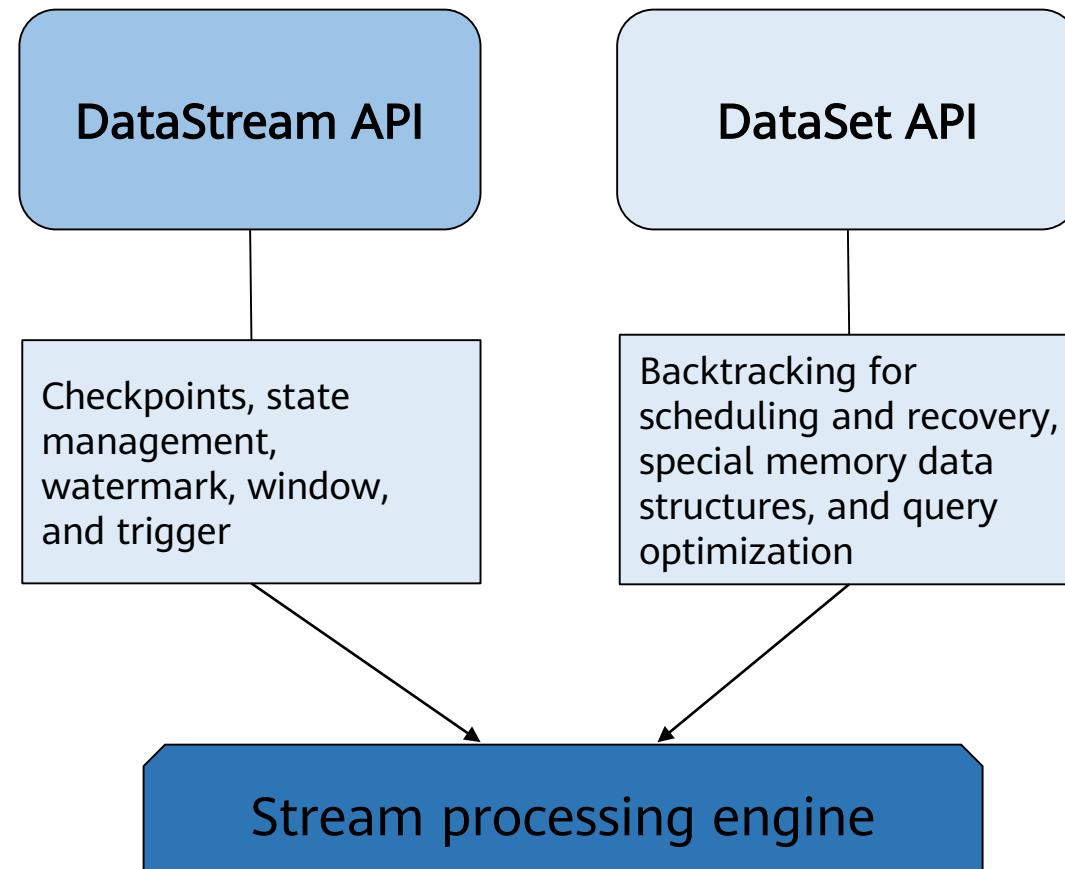
```
val counts = visits
    .keyBy("region")
    .window(GlobalWindows.create)
    .trigger(EndOfTimeTrigger.create)
    .sum("visits")
```

- If the input data is bounded, the result of the following code is the same as that of the preceding code:

```
val counts = visits
    .groupBy("region")
    .sum("visits")
```

# Flink Batch Processing Model

- Flink uses a bottom-layer engine to support both stream and batch processing.



# Stream and Batch Processing Mechanisms

- The two sets of Flink mechanisms correspond to their respective APIs (DataStream API and DataSet API). When creating a Flink job, you cannot combine the two sets of mechanisms to use all Flink functions at the same time.
- Flink supports two types of relational APIs: Table API and SQL. Both of these APIs are for unified batch and stream processing, which means that relational APIs execute queries with the same semantics and produce the same results on unbounded data streams and bounded data streams.
  - The Table API and SQL APIs are becoming the main APIs for analytical use cases.
  - The DataStream API is the primary API for data-driven applications and pipelines.

# Contents

1. Principles and Architecture of Flink
- 2. Flink Time & Window**
3. Flink Watermark
4. Fault Tolerance of Flink

# Time Background

- In stream processor programming, time processing is very critical. For example, event stream data (such as server log data, web page click data, and transaction data) is continuously generated. In this case, you need to use keys to group events and count the events corresponding to each key at a specified interval. This is our known "big data" application.

# Time Classification in Stream Processing

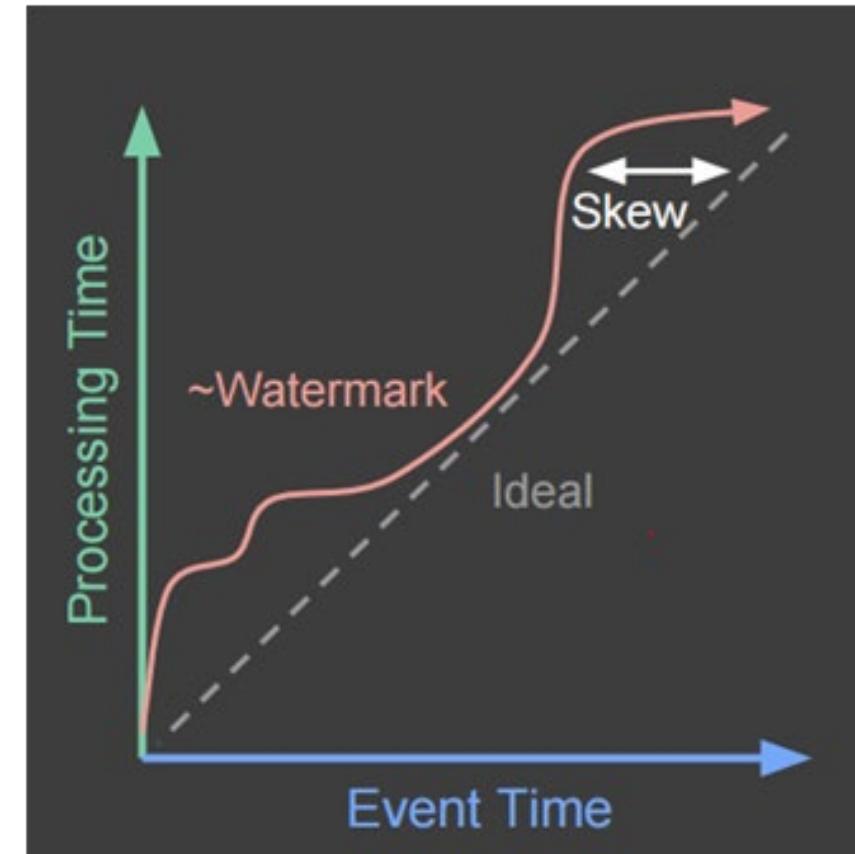
- During stream processing, the system time (processing time) is often used as the time of an event. However, the system time (processing time) is the time that is imposed on an event. Due to reasons such as network delay, the processing time cannot reflect the sequence of events.
- In actual cases, the time of each event can be classified into the following types:
  - Event time: time when an event occurs
  - Ingestion time: time when an event arrives at the stream processing system
  - Processing time: time when an event is processed by the system

# Three Time Examples

- For example, if a log enters Flink at 10:00:00.123 on November 12, 2019 and reaches Window at 10:00:01.234 on November 12, 2019, the log content is as follows:
  - 2019-11-02 18:37:15.624 INFO Fail over to rm2
    - 2019-11-02 18:37:15.624 is the event time.
    - 2019-11-12 10:00:00.123 is the ingestion time.
    - 2019-11-12 10:00:01.234 is the processing time.

# Differences Between Three Time

- In the actual situation, the sequence of events is different from the system time. These differences are caused by network delay and processing time. See the following figure.
- The horizontal coordinate indicates the event time, and the vertical coordinate indicates the processing time. Ideally, the coordinate formed by the event time and processing time should form a line with a tilt angle of 45 degrees. However, the processing time is later than the event time. As a result, the sequence of events is inconsistent.



# Time Semantics Supported by Flink

Processing Time	Event Time (Row Time)
Time of the real world	Time when data is generated
Local time when the data node is processed	Timestamp carried in a record
Simple processing	Complex processing
Uncertain results (cannot be reproduced)	Certain results (reproducible)

# Window Overview

- Streaming system is a data processing engine designed to process infinite data sets. Infinite data sets refer to constantly growing data sets. Window is a method for splitting infinite data sets into finite blocks for processing.
- Windows are at the heart of processing infinite streams. Windows split the stream into "buckets" of finite size, over which we can apply computations.

# Window Types

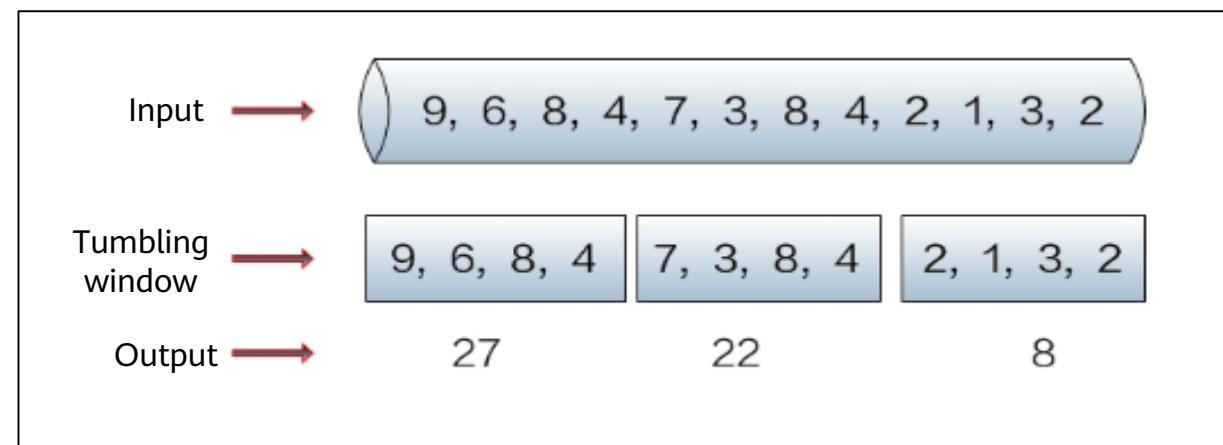
- Windows can be classified into two types:
  - Count Window: Data-driven window is generated based on the specified number of data records, which is irrelevant to time.
  - Time Window: Time-driven window is generated based on time.
  - Apache Flink is a distributed computing framework that supports infinite stream processing. In Flink, Window can divide infinite streams into finite streams.

# Time Window Types

- According to the window implementation principles, Time Window can be classified into tumbling window, sliding window, and session window.

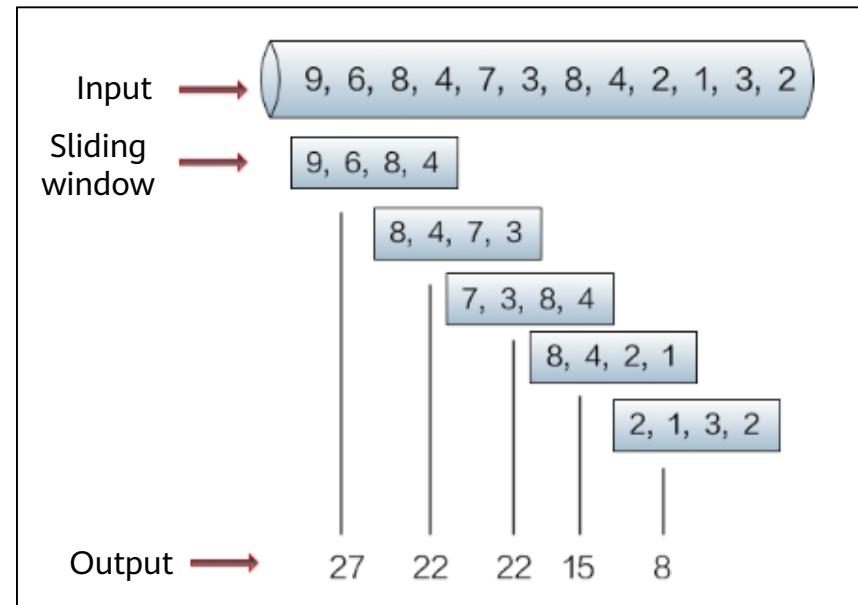
# Tumbling Window

- The data is sliced according to a fixed window length. Characteristics: The time is aligned, the window length is fixed, and the windows do not overlap.
- Application scenario: BI statistics collection (aggregation calculation in each time period)
- For example, assume that you want to count the values emitted by a sensor. A tumbling window of 1 minute collects the values of the last minute, and emits their sum at the end of the minute. See the following figure.



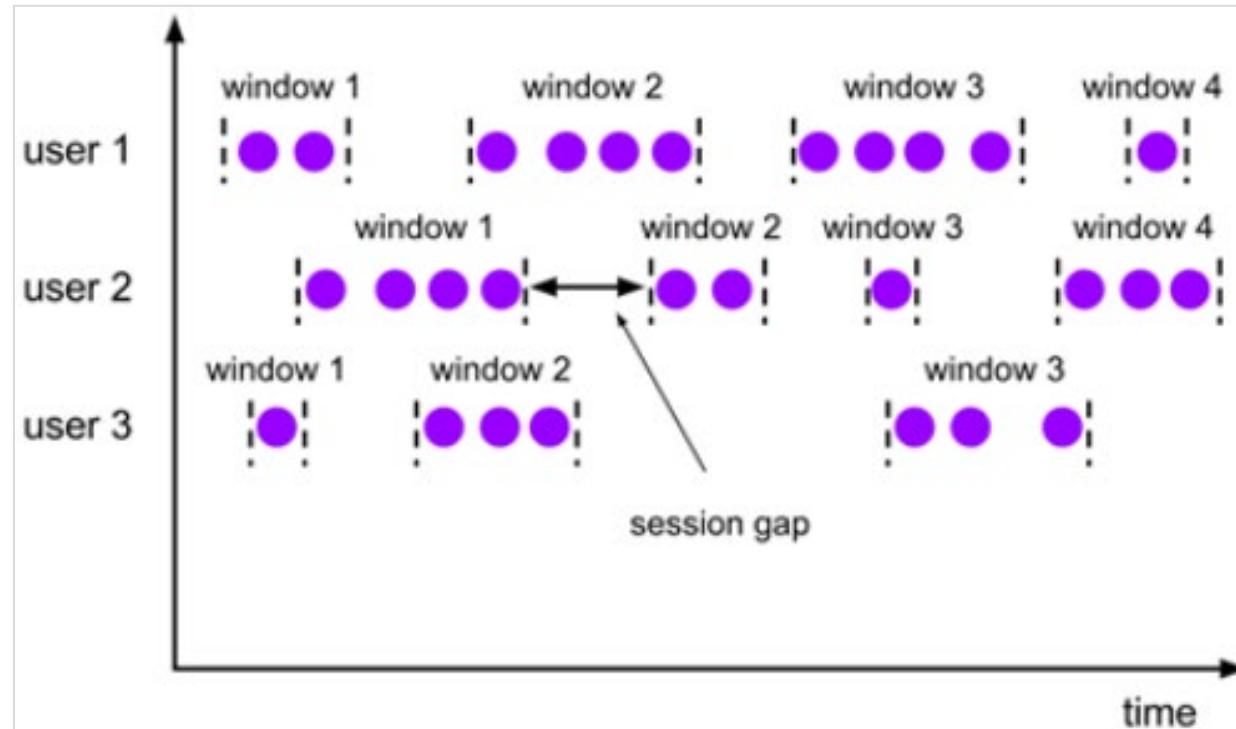
# Sliding Window

- The sliding window is a more generalized form of a fixed window. It consists of a fixed window length and a sliding interval. Characteristics: The time is aligned, the window length is fixed, and the windows can be overlapping.
- Application scenario: Collect statistics on the failure rate of an interface over the past 5 minutes to determine whether to report an alarm.
- A sliding window of 1 minute that slides every half minute counts the values of the last minute, emitting the count every half minute. See the following figure.



# Session Window

- A session window consists of a series of events and a timeout interval of a specified duration. It is similar to a web application session. That is, a new window is generated if no new data is received within a period of time. Characteristics: The time is not aligned.



# Code Definition

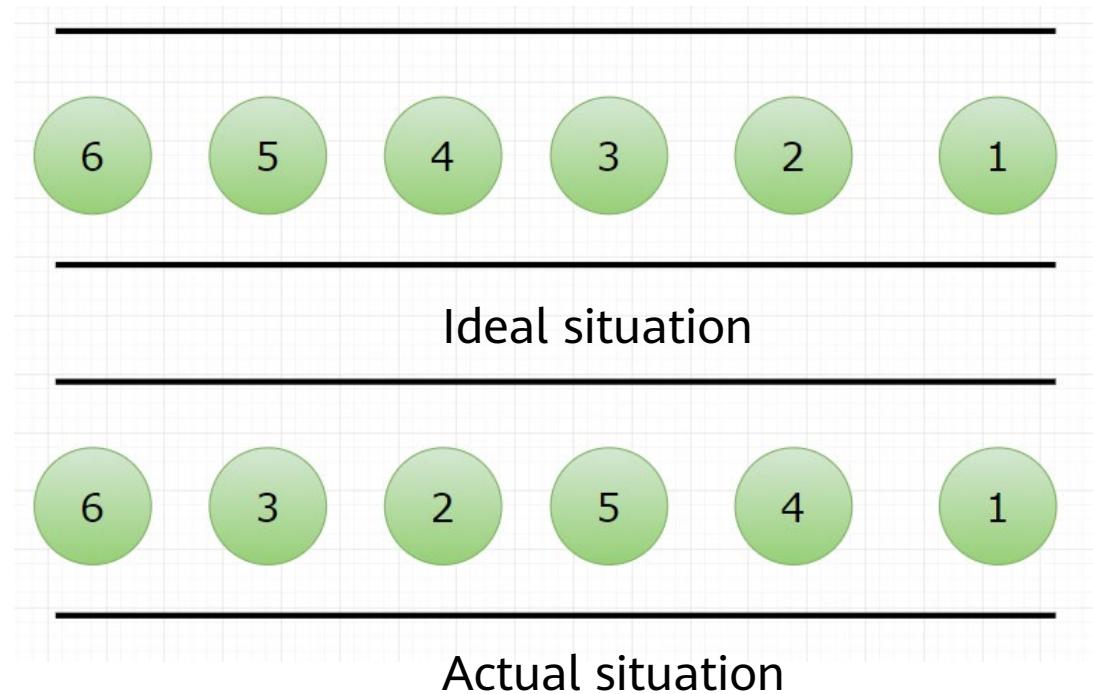
- A tumbling time window of 1 minute can be defined in Flink simply as:
  - `stream.timeWindow(Time.minutes(1));`
- A sliding time window of 1 minute that slides every 30 seconds can be defined as simply as:
  - `stream.timeWindow(Time.minutes(1),Time.seconds(30));`

# Contents

1. Principles and Architecture of Flink
2. Flink Time & Window
- 3. Flink Watermark**
4. Fault Tolerance of Flink

# Out-of-Order Problem

- There is a process from event generation to stream processing through the source and then to the operator. In most cases, data is sent to the operator based on the time when the event is generated. However, out-of-order data may be generated, such as, events received by Flink are not sorted strictly based on the event time due to network problems.



# Out-of-Order Example

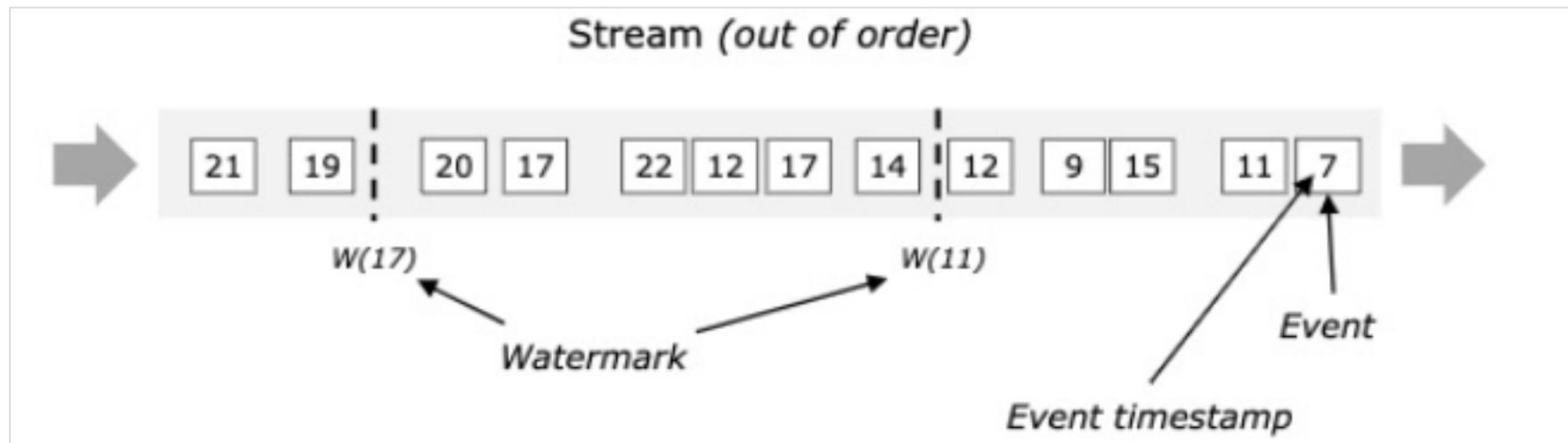
- An app records all user clicks and sends logs back. If the network condition is poor, the logs are saved locally and sent back later. User A performs operations on the app at 11:02 and user B performs operations on the app at 11:03. However, the network of user A is unstable and log backhaul is delayed. As a result, the server receives the message from user B at 11:03 and then the message from user A at 11:02.

# Why Are Watermarks Needed?

- For infinite data sets, there is no effective way to determine data integrity. Therefore, watermark is a concept based on event time to describe the integrity of data streams. If events are measured by processing time, everything is ordered and perfect. Therefore, watermarks are not required. In other words, event time causes disorder. Watermarks are used to solve the disorder problem. The so-called disorder is actually an event delay. For a delayed element, it is impossible to wait indefinitely. A mechanism must be provided to ensure that the window is triggered for calculation after a specific time. This special mechanism is watermark, which tells operators that delayed messages should no longer be received.

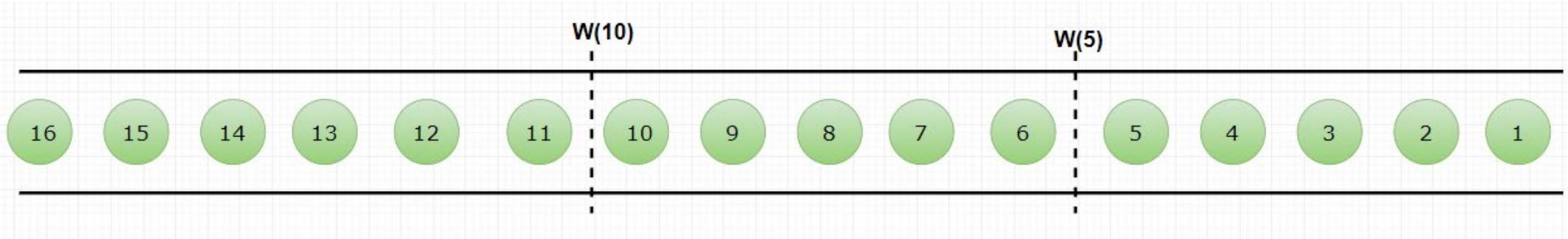
# Watermark Principles (1)

- How does Flink ensure that all data has been processed when the event time-based window is destroyed? This is a watermark does. A watermark is a monotonically increasing timestamp  $t$ . **Watermark( $t$ )** indicates that all data whose timestamp is less than or equal to  $t$  has arrived, and data whose timestamp is less than or equal to  $t$  will not be received in the future. Therefore, the window can be safely triggered and destroyed.

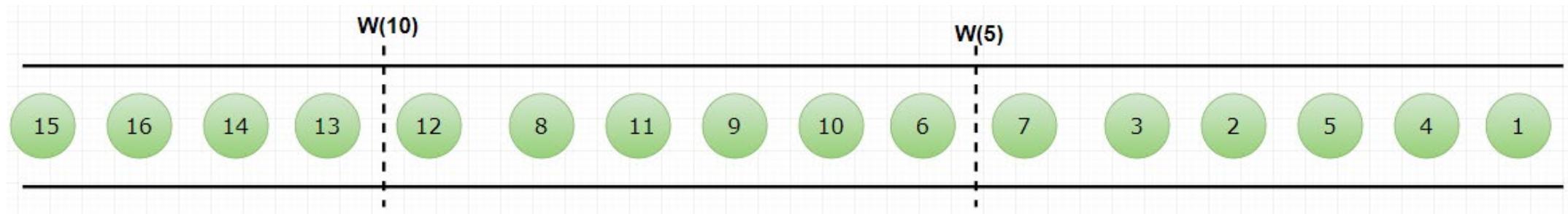


# Watermark Principles (2)

- The following figure shows the watermark of ordered streams (Watermark is set to 0).



- The following figure shows the watermark of out-of-order streams (Watermark is set to 2).



# Delayed Data

- Watermark is a means of coping with out-of-order data, but in the real world we cannot obtain a perfect watermark value - either it cannot be obtained, or it is too costly. Therefore, in actual use, we will use an approximate **Watermark(t)** value, but there is still a low probability that the data before the timestamp **t** will be received. In Flink, the data is defined as late elements. Similarly, we can specify the maximum latency allowed in the window (0 by default), which can be set using the following code:

```
input .keyBy(<key selector>)
    .window(<window assigner>)
    .allowedLateness(<time>)
    .<windowed transformation>(<window function>);
```

# Delayed Data Processing Mechanism

- Delayed events are special out-of-order events. Different from common out-of-order events, their out-of-order degree exceeds that watermarks can predict. As a result, the window is closed before they arrive.
- In this case, you can use any of the following methods to solve the problem:
  - Reactivate the closed windows and recalculate to correct the results.
  - Collect the delayed events and process them separately.
  - Consider delay events as error messages and discard them.
- By default, Flink discards the third method and uses Side Output and Allowed Lateness instead.

# Side Output Mechanism

- In the Side Output mechanism, a delayed event can be placed in an independent data stream, which can be used as a by-product of the window calculation result so that users can obtain and perform special processing on the delay event.

# Getting Delayed Data as a Side Output

- After allowedLateness is set, delayed data can also trigger the window for output. Using the side output mechanism of Flink, we can obtain this delayed data in the following way:

```
final OutputTag<T> lateOutputTag = new OutputTag<T>("late-date"){};  
  
DataStream input =...;  
SingleOutputStreamOperator<T> result = input  
    .keyBy(<key selector>)  
    .window(<window assigner>)  
    .allowedLateness(<time>)  
    .<windowed transformation>(<>window function>);  
DataStream<T> lateStream = result.getSideOutput(lateOutputTag);
```

# Allowed Lateness Mechanism

- Allowed Lateness allows you to specify a maximum allowed lateness. After the window is closed, Flink keeps the state of windows until their allowed lateness expires. During this period, the delayed events are not discarded, but window recalculation is triggered by default. Keeping the window state requires extra memory. If the ProcessWindowFunction API is used for window calculation, each delayed event may trigger a full calculation of the window, which costs a lot. Therefore, the allowed lateness should not be too long, and the number of delayed events should not be too many.

# Contents

1. Principles and Architecture of Flink
2. Flink Time & Window
3. Flink Watermark
- 4. Fault Tolerance of Flink**

# Checkpoint

- How does Flink ensure exactly-once? It uses a feature called checkpoint to reset the system to the correct state in the event of a failure. Flink state storage depends on the checkpoint mechanism. Checkpoint periodically creates distributed snapshots to back up the state in the program.

# Checkpoint Mechanism

- Apache Flink offers a lightweight fault tolerance mechanism based on distributed checkpoints. A checkpoint is an automatic, asynchronous snapshot of task/operator state. Flink generates checkpoint barriers at intervals on the input data set and uses barriers to divide the data during the interval into the corresponding checkpoints. When an application error occurs, the states of all operators can be restored from the previous snapshot to ensure data consistency.
- For applications with small state, these snapshots are very light-weight and can be drawn frequently without impacting the performance much. During checkpointing, the state is stored at a configurable place (such as the JobManager node or HDFS).

# Checkpoint Configuration (1)

- By default, checkpointing is disabled. To enable checkpointing, call `enableCheckpointing(n)` on the `StreamExecutionEnvironment`, where *n* is the checkpoint interval in milliseconds.

```
env.enableCheckpointing(1000) // Enable checkpointing and set the checkpoint  
interval to 1000 ms. If the state is large, you are advised to increase the value.
```

# Checkpoint Configuration (2)

- Exactly-once or at-least-once
  - Exactly-once ensures end-to-end data consistency, prevents data loss and duplicates, and delivers poor Flink performance.
  - At-least-once applies to scenarios that have high requirements on the latency and throughput but low requirements on data consistency.
- Exactly-once is used by default. You can use the `setCheckpointingMode()` method to set the semantic mode.

```
env.getCheckpointConfig().setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE)
```

# Checkpoint Configuration (3)

- Checkpoint Timeout
  - Specifies the timeout period of checkpoint execution. Once the threshold is reached, Flink interrupts the checkpoint process and regards it as timeout.
  - This metric can be set using the `setCheckpointTimeout` method. The default value is 10 minutes.

```
env.getCheckpointConfig().setCheckpointingTimeout(60000)
```

# Checkpoint Configuration (4)

- Minimum time between checkpoints
  - When checkpoints end up frequently taking longer than the base interval because state grew larger than planned, the system is constantly taking checkpoints. That can mean that too many computing resources are constantly tied up in checkpointing, thereby affecting the overall application performance. To prevent such a situation, applications can define a minimum duration between two checkpoints:

```
env.getCheckpointConfig().setMinPauseBetweenCheckpoints(500)
```

# Checkpoint Configuration (5)

- Number of concurrent checkpoints
  - Specifies the number of checkpoints that can be executed at the same time. By default, the system will not trigger another checkpoint while one is still in progress. If you configure multiple checkpoints, the system can trigger multiple checkpoints at the same time.

```
env.getCheckpointConfig().setMaxConcurrentCheckpoints(500)
```

# Checkpoint Configuration (6)

- Externalized checkpoints
  - You can configure periodic checkpoints to be persisted externally. Externalized checkpoints write their meta data out to persistent storage and are not automatically cleaned up when the job fails. This way, you will have a checkpoint around to resume from if your job fails.

```
env.getCheckpointConfig().enableExternalizedCheckpoints(ExternalizedCheckpointCleanup.RETAIN_ON_CANCELLATION)
```

# How Do I Restore Job Data?

- The checkpoint can be retained in an external media when a job is cancelled. Flink also has another mechanism, savepoint, to restore job data.
- Savepoints are a special implementation of checkpoints. The underlying layer uses the checkpoint mechanism. Savepoints are manually triggered by running commands and results are saved to a specified storage path so that the system can be restored to the original computing status during system upgrade and maintenance and end-to-end exactly-once semantics can be ensured.

# Savepoint and Checkpoint

	<b>Checkpoint</b>	<b>Savepoint</b>
<b>Triggering and management</b>	Automatically triggered and managed by Flink	Manually triggered and managed by users
<b>Purpose</b>	Quickly restores tasks from failures, for example, timeout due to network jitter.	Backs up data as planned, for example, by modifying code or adjusting concurrency.
<b>Features</b>	<ul style="list-style-type: none"><li>• Lightweight</li><li>• Automatic recovery from failures</li><li>• State is cleared by default after a job is stopped.</li></ul>	<ul style="list-style-type: none"><li>• Persistent</li><li>• Stored in a standard format and allows code or configuration changes.</li><li>• Manually restores data from savepoints.</li></ul>

# State Storage Method - MemoryStateBackend

- Constructor
  - `MemoryStateBackend(int maxStateSize, boolean asynchronousSnapshots)`
  - Storage Methods
    - State: TaskManager memory
    - Checkpoint: JobManager memory
  - Capacity Limit
    - The default value of `maxStateSize` for a single state is 5 MB.
    - $\text{maxStateSize} \leq \text{akka.framesize}$  (Default value: 10 MB)
    - The total size cannot exceed the memory of JobManager.
  - The `MemoryStateBackend` is encouraged for local testing and jobs that hold little state, such as ETL.

# State Storage Method - FsStateBackend

- Constructor
  - `FsStateBackend(URI checkpointDataUri ,boolean asynchronousSnapshots)`
  - Storage Methods
    - State: TaskManager memory
    - Checkpoint: external file storage system (local or HDFS)
  - Capacity Limit
    - The amount of state on a single TaskManager cannot exceed its memory.
    - The total size cannot exceed the configured file system capacity.
  - The FsStateBackend is encouraged for jobs with large state, such as aggregation at the minute-window level and join, and jobs requiring all high-availability setups. It can be used in production scenarios.

# State Storage Method - RocksDBStateBackend

- Constructor
  - `RocksDBStateBackend(URI checkpointDataUri, boolean enableIncrementalCheckpointing)`
  - Storage Methods
    - State: KV database on the TaskManager (used memory + disk)
    - Checkpoint: external file storage system (local or HDFS)
  - Capacity Limit
    - The amount of state on a single TaskManager cannot exceed the memory and disk size of the TaskManager.
    - The maximum size of a key is 2 GB.
  - The total size cannot exceed the configured file system capacity.
- The RocksDBStateBackend is encouraged for jobs with very large state, for example, aggregation at the day-window level, jobs requiring high-availability setups, and jobs that do not require high read/write performance. It can be used in production scenarios.

# Summary

- This course explains the architecture and technical principles of Flink and the running process of Flink programs. The focus is on the difference between Flink stream processing and batch processing. In the long run, the DataStream API should contain the DataSet API through bounded data streams.

# Quiz

1. What are the four key concepts of Flink?
2. What are the two types of APIs for Flink stream processing and batch processing?

# Recommendations

---

- Huawei Cloud Official Web Link:
  - <https://www.huaweicloud.com/intl/en-us/>
- Huawei MRS Documentation:
  - <https://www.huaweicloud.com/intl/en-us/product/mrs.html>
- Huawei TALENT ONLINE:
  - <https://e.huawei.com/en/talent/#/>

# Thank you.

把数字世界带入每个人、每个家庭、  
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and  
organization for a fully connected,  
intelligent world.

Copyright©2020 Huawei Technologies Co., Ltd.  
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.



# Revision Record

Do Not Print this Page

Course Code	Product	Product Version	Course Version
H13-711	MRS		V3.0

Author/ID	Date	Reviewer/ID	New/ Update
Mao Jian/mwx711840	2020.03.30	Wang Yongjie/wwx769662	Update
Mao Jian/mwx711840	2020.08.26	Fu Zheng/fw277898	New

# Chapter 8 Flume - Massive Log Aggregation



# Foreword

---

- Flume is an open-source, distributed, reliable, and highly available massive log aggregation system. It supports custom data transmitters for collecting data. It roughly processes data and writes data to data receivers.

# Objectives

- Upon completion of this course, you will be able to:
  - Know what is Flume.
  - Understand what Flume can do.
  - Know the system architecture of Flume.
  - Grasp key features of Flume.
  - Master Flume applications.

# Contents

- 1. Overview and Architecture**
2. Key Features
3. Applications

# What Is Flume?

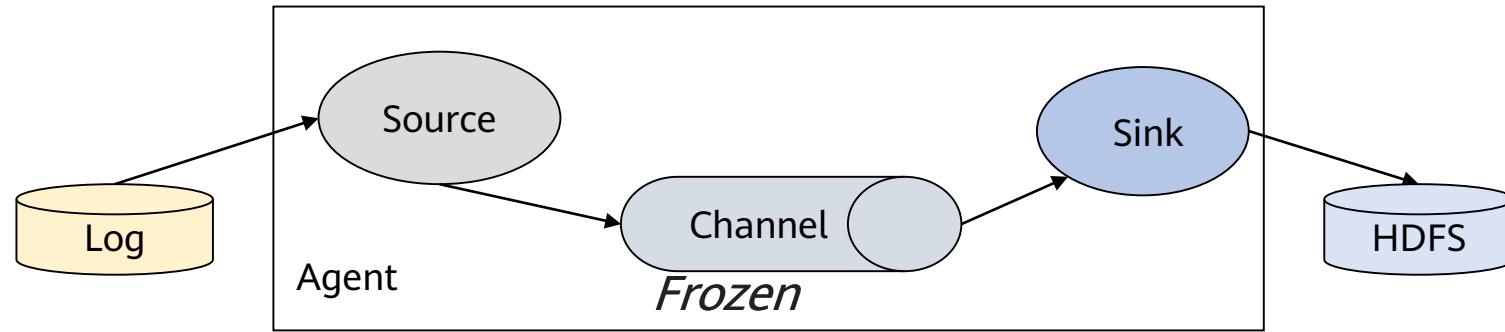
- Flume is a stream log collection tool. It roughly processes data and writes data to data receivers. Flume collects data from local files (spooling directory source), real-time logs (taildir and exec), REST messages, Thrift, Avro, Syslog, Kafka, and other data sources.

# What Can Flume Do?

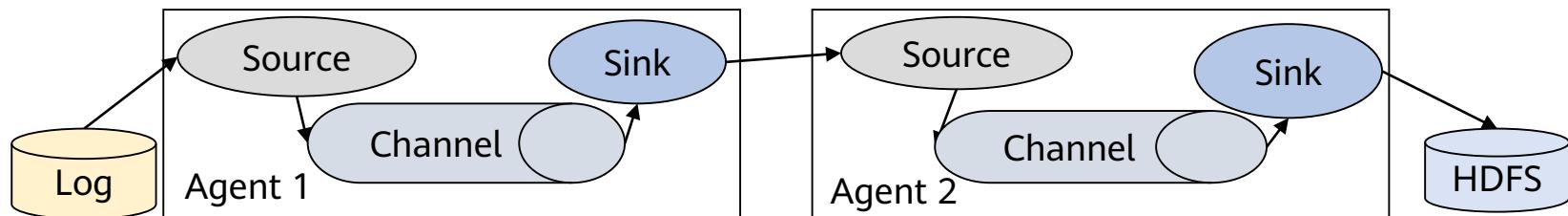
- Collect log information from a fixed directory to a destination (HDFS, HBase, or Kafka).
- Collect log information (taildir) to the destination in real time.
- Support cascading (connecting multiple Flumes) and data conflation.
- Support custom data collection tasks based on users.

# Flume Agent Architecture (1)

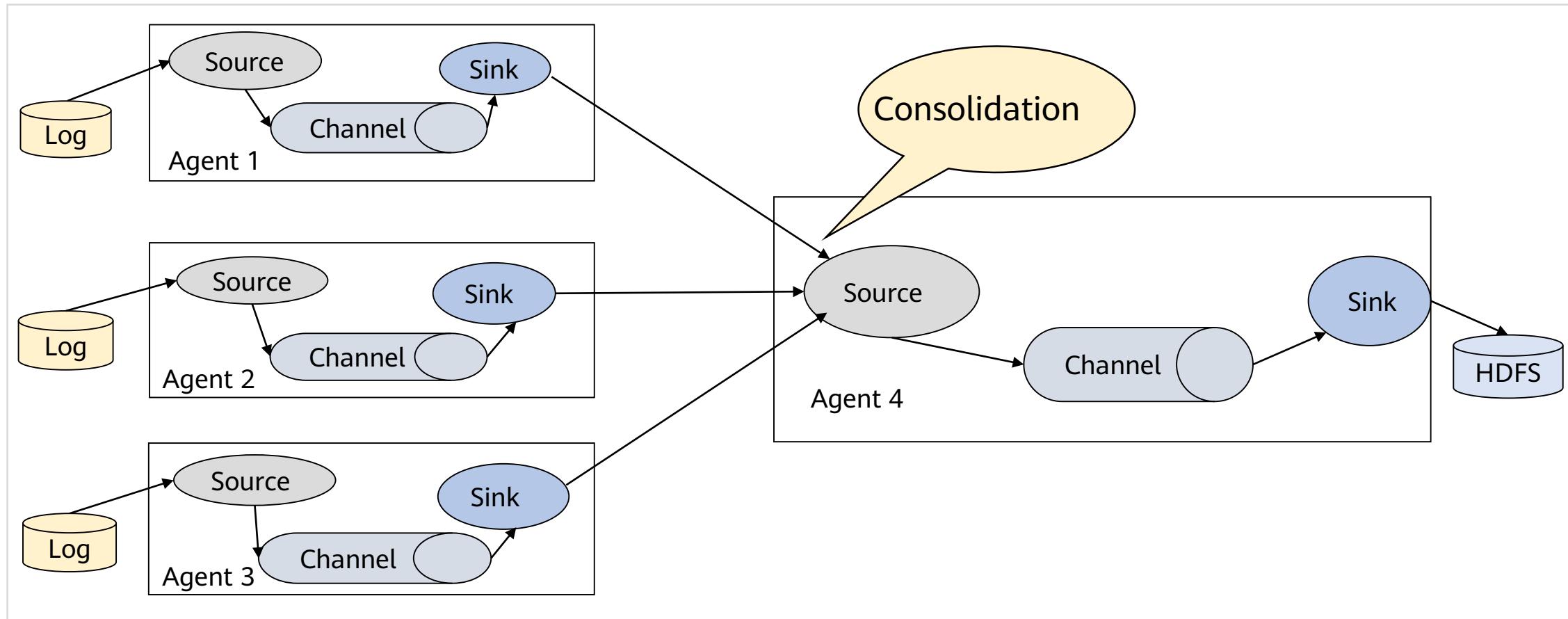
- Infrastructure: Flume can directly collect data with an agent, which is mainly for data collection in a cluster.



- Multi-agent architecture: Flume can connect multiple agents to collect raw data and store them in the final storage system. This architecture is used to import data from outside the cluster to the cluster.

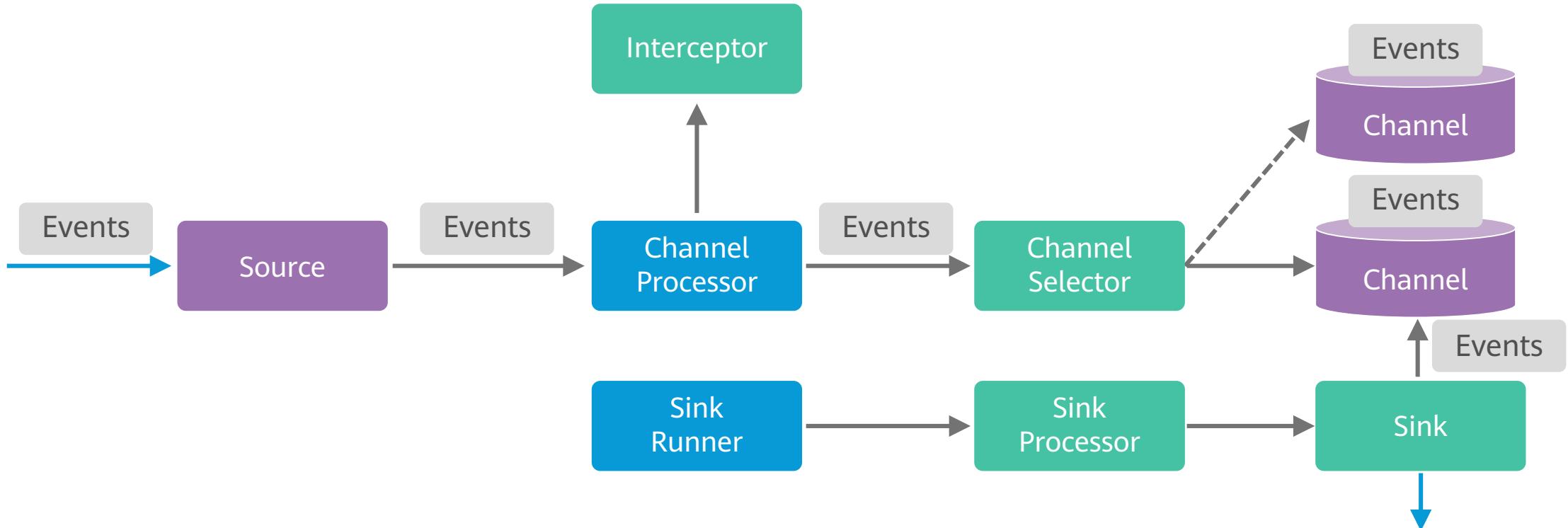


# Flume Multi-Agent Consolidation



- You can configure multiple level-1 agents and point them to the source of an agent using Flume. The source of the level-2 agent consolidates the received events and sends the consolidated events into a single channel. The events in the channel are consumed by a sink and then pushed to the destination.

# Flume Agent Principles



# Basic Concepts - Source (1)

- A source receives events or generates events using a special mechanism, and places events to one or more channels. There are two types of sources: driver-based source and polling source.
  - Driver-based source: External systems proactively send data to Flume, driving Flume to receive data.
  - Polling source: Flume periodically obtains data.
- A source must be associated with at least one channel.

# Basic Concepts - Source (2)

Source Type	Description
exec source	Executes a command or script and uses the output of the execution result as the data source.
avro source	Provides an Avro-based server, which is bound to a port and waits for the data sent from the Avro client.
thrift source	Same as Avro, but the transmission protocol is Thrift.
http source	Supports sending data using the POST request of HTTP.
syslog source	Collects syslogs.
spooling directory source	Collects local static files.
jms source	Obtains data from the message queue.
Kafka source	Obtains data from Kafka.

# Basic Concepts - Channel (1)

- A channel is located between a source and a sink. A channel functions as a queue and is used for caching temporary events. When a sink successfully sends events to the channel of the next hop or the final destination, the events are removed from the channel.
- The persistency of channels varies with the channel types:
  - Memory channel: The memory in this channel type is not persistent.
  - File channel: It is implemented based on write-ahead logs (WALs).
  - JDBC channel: It is implemented based on the embedded database.
- Channels support transactions and provide weak sequence assurance. They can connect to any number of sources and sinks.

# Basic Concepts - Channel (2)

- Memory channel: Messages are stored in the memory, which has high throughput but does not ensure reliability. Data may be lost.
- File channel: Data is permanently stored. The configuration is complex. You need to configure the data directory and checkpoint directory. A checkpoint directory must be configured for each file channel.
- JDBC channel: A built-in Derby database makes events persistent with high reliability. This channel type can replace the file channel with the persistence feature.

# Basic Concepts - Sink

- A sink sends events to the next hop or the final destination and then removes the events from the channel.
- A sink must work with a specific channel.

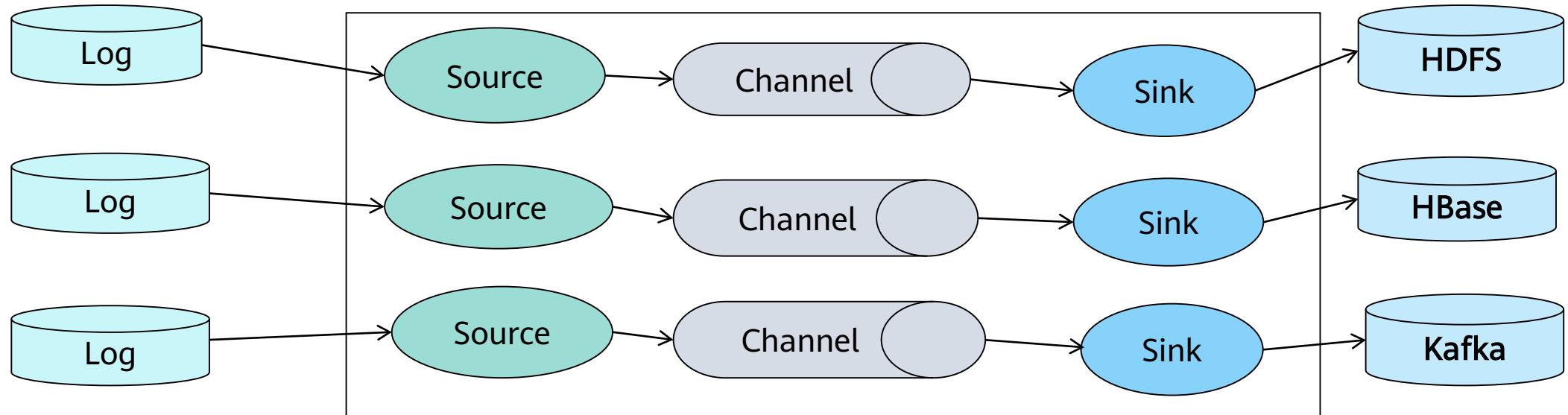
Sink Type	Description
HDFS sink	Writes data to HDFS.
Avro sink	Sends data to Flume of the next hop using the Avro protocol.
Thrift sink	Same as Avro, but the transmission protocol is Thrift.
File roll sink	Saves data to the local file system.
HBase sink	Writes data to HBase.
Kafka sink	Writes data to Kafka.
MorphlineSolr sink	Writes data to Solr.

# Contents

1. Overview and Architecture
- 2. Key Features**
3. Applications

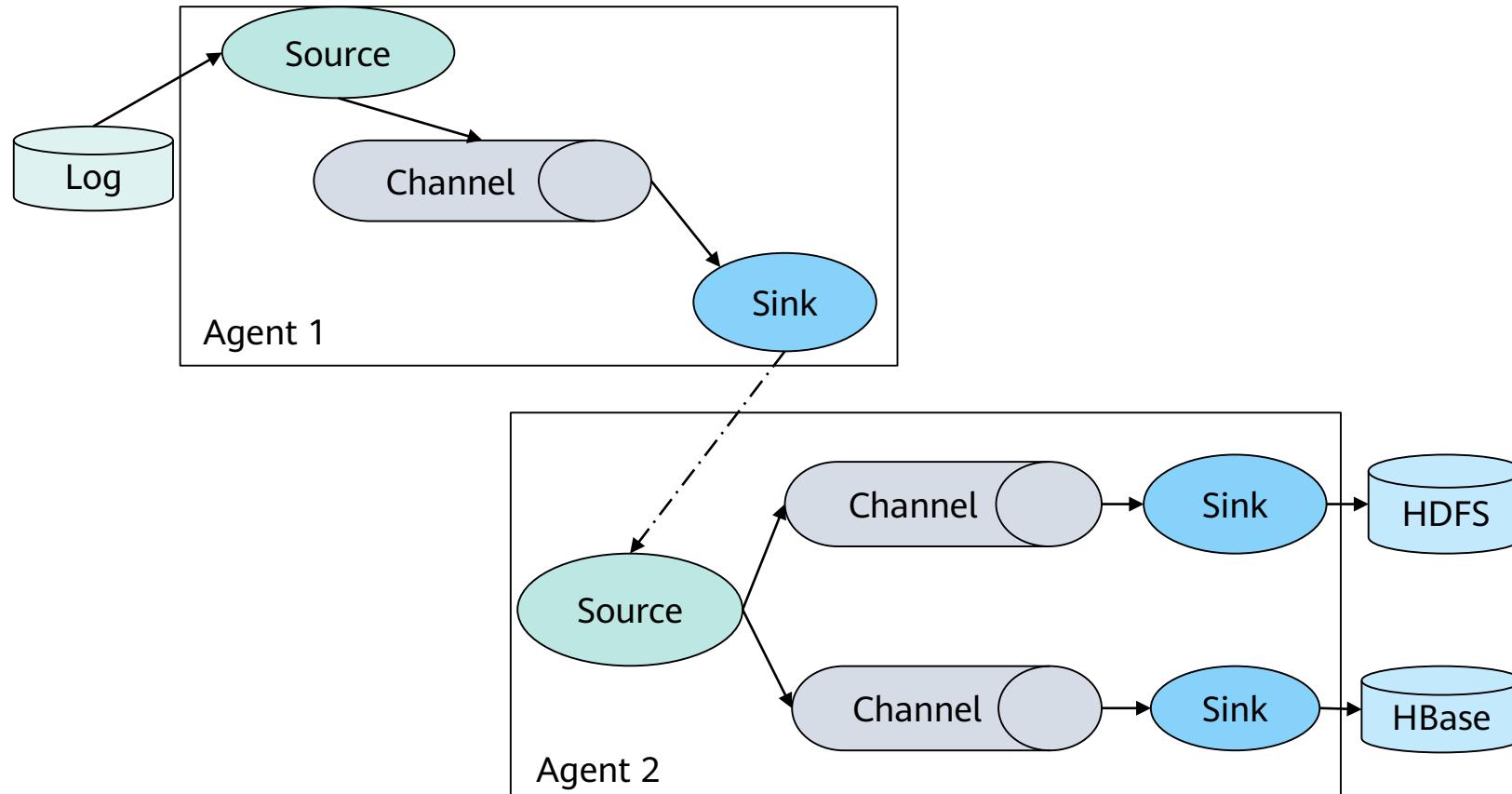
# Log File Collection

- Flume collects log files in a cluster and archives them in HDFS, HBase, or Kafka used for data analysis and cleansing for upper-layer applications.



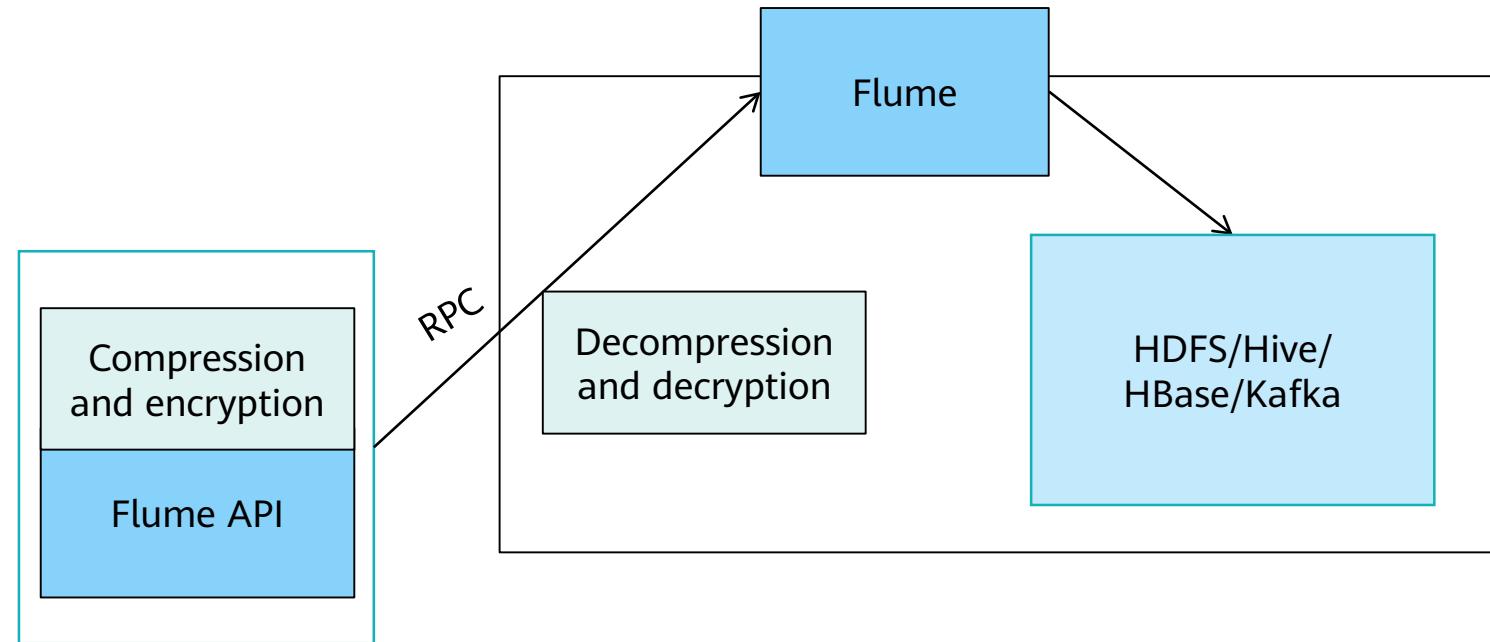
# Multi-level Cascading and Multi-channel Replication

- Flume supports cascading of multiple Flume agents and data replication within the cascading nodes.

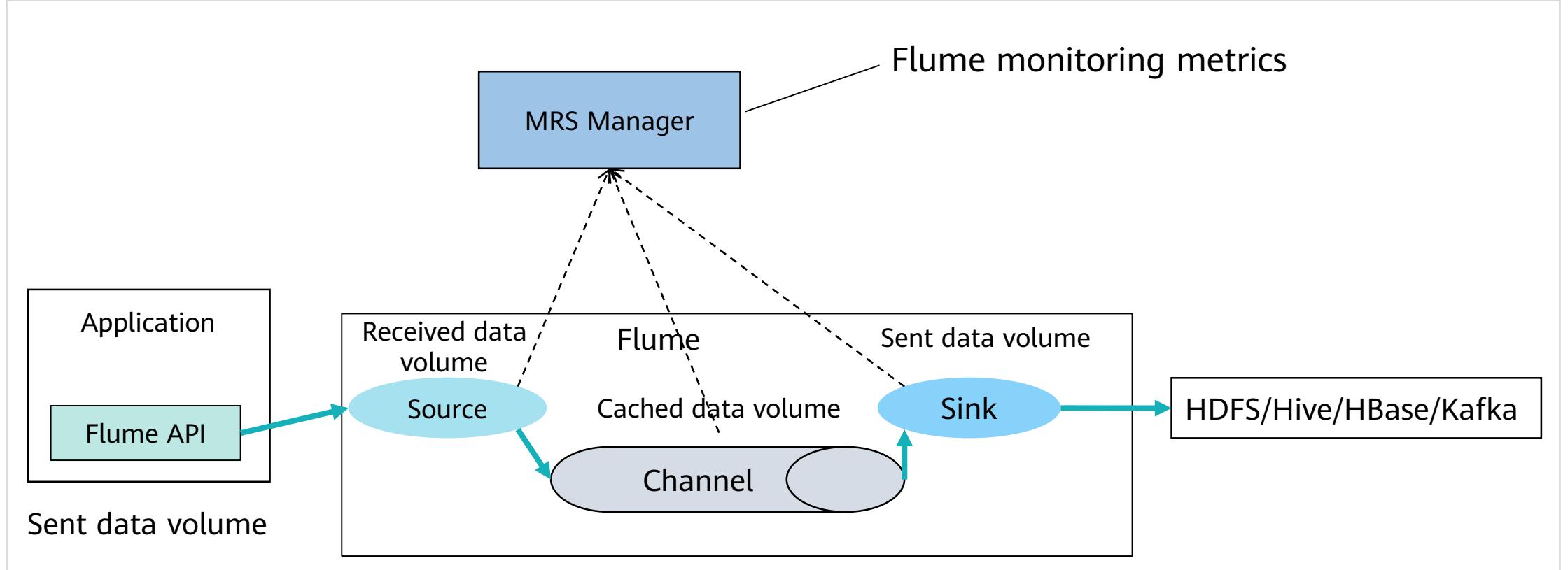


# Cascading Message Compression and Encryption

- Data transmission between cascaded Flume agents can be compressed and encrypted, improving data transmission efficiency and security.

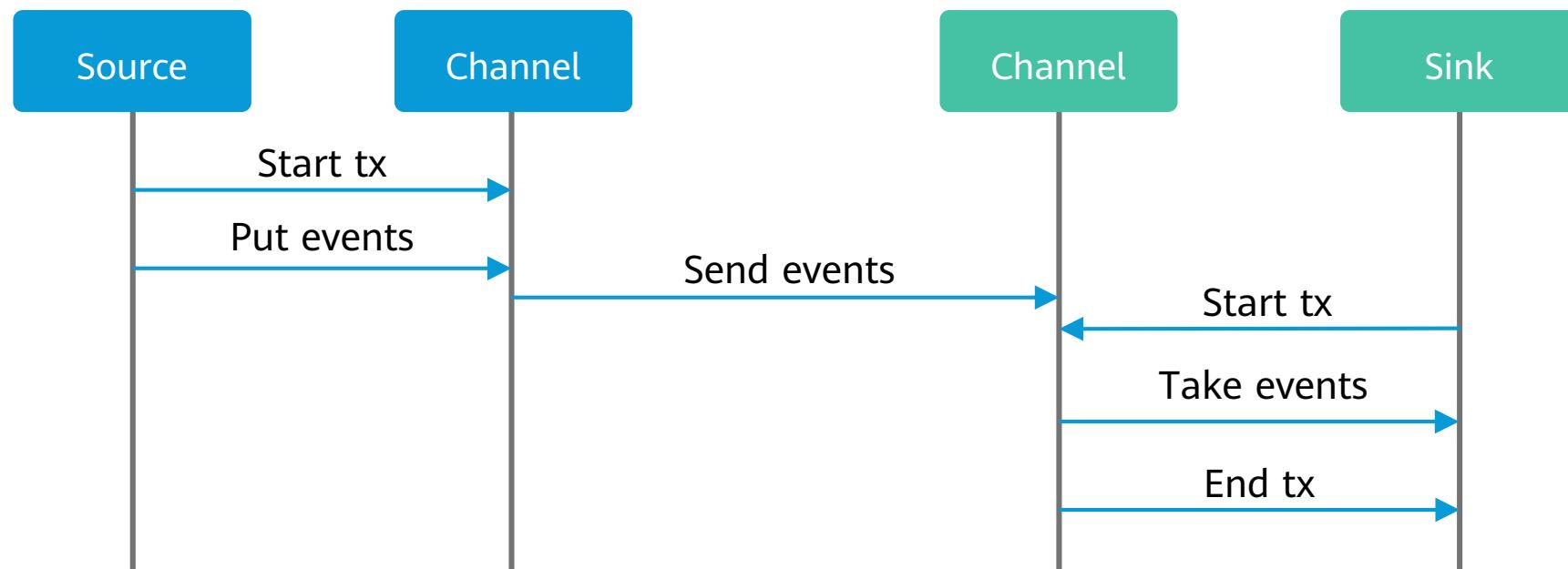


# Data Monitoring



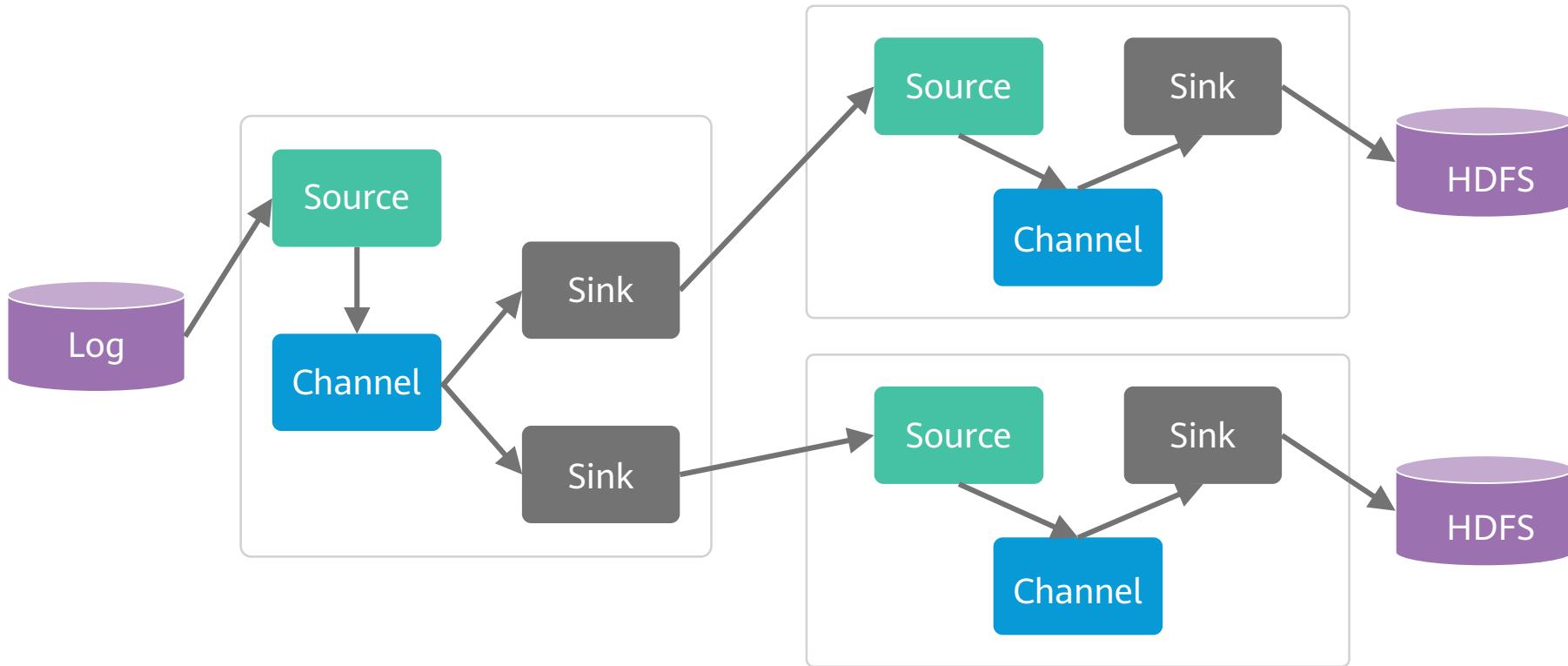
# Transmission Reliability

- During data transmission, Flume uses the transaction management mode to ensure data completeness and keeps transmission reliable. In addition, if the data is cached in the file channel, data will not be lost when the process or agent is restarted.



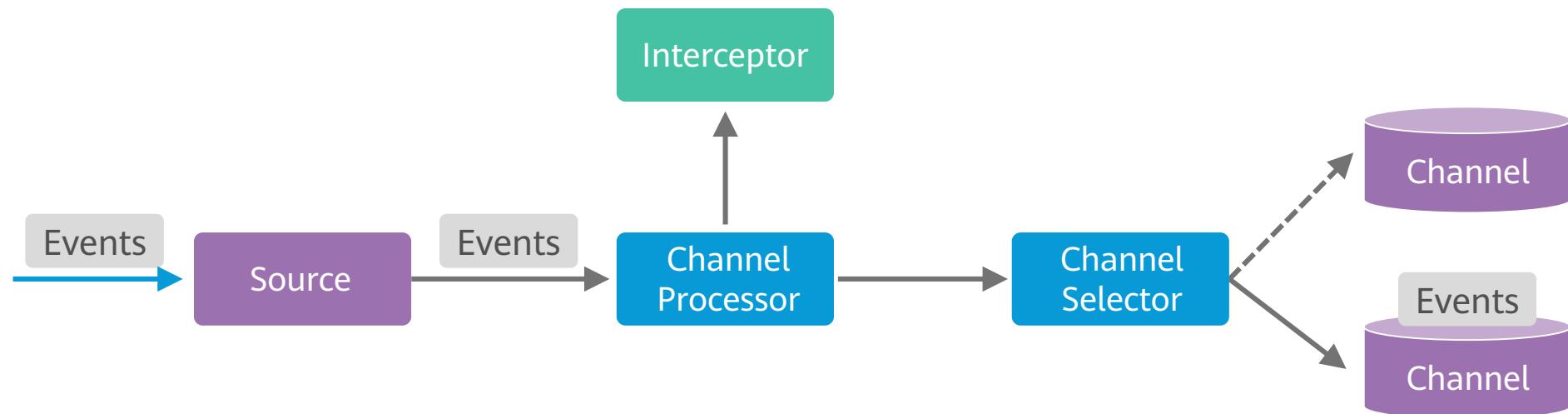
# Failover

- Data can be automatically switched to another channel for transmission when the next-hop Flume agent is faulty or data receiving is abnormal during Flume data transmission.



# Data Filter During Data Transmission

- During data transmission, Flume roughly filters and cleans data to delete unnecessary data. If data to be filtered is complex, users need to develop filter plugins based on their data characteristics. Flume supports third-party filter plugins.



# Contents

1. Overview and Architecture
2. Key Features
- 3. Applications**

# Flume Operation Example 1 (1)

- Description
  - This example shows how Flume ingests logs generated by applications (such as e-banking systems) in a cluster to HDFS.
- Prepare data.
  - Create a log directory named `mkdir /tmp/log_test` on a node in the cluster.
  - Use this directory as the monitoring directory.
- Download the Flume client.
  - Log in to MRS Manager. On the Clusters page, choose Services > Flume > Download Client.

# Flume Operation Example 1 (2)

- Install the Flume client.

- Decompress the client.

```
tar -xvf MRS_Flume_Client.tar
tar -xvf MRS_Flume_ClientConfig.tar
cd /tmp/MRS-client/MRS_Flume_ClientConfig/Flume
tar -xvf FusionInsight-Flume-1.6.0.tar.gz
```

- Install the client.

```
./install.sh -d /opt/FlumeClient -f hostIP -c
flume/conf/client.properties.properties
```

# Flume Operation Example 1 (3)

- Configure the Flume source.

```
server.sources = a1
server.channels = ch1
server.sinks = s1
# the source configuration of a1
server.sources.a1.type = spooldir
server.sources.a1.spoolDir = /tmp/log_test
server.sources.a1.fileSuffix = .COMPLETED
server.sources.a1.deletePolicy = never
server.sources.a1.trackerDir = .flumespool
server.sources.a1.ignorePattern = ^$ 
server.sources.a1.batchSize = 1000
server.sources.a1.inputCharset = UTF-8
server.sources.a1.deserializer = LINE
server.sources.a1.selector.type = replicating
server.sources.a1.fileHeaderKey = file
server.sources.a1.fileHeader = false
server.sources.a1.channels = ch1
```

# Flume Operation Example 1 (4)

- Configure the Flume channel.

```
# the channel configuration of ch1
server.channels.ch1.type = memory
server.channels.ch1.capacity = 10000
server.channels.ch1.transactionCapacity = 1000
server.channels.ch1.channlefullcount = 10
server.channels.ch1.keep-alive = 3
server.channels.ch1.byteCapacityBufferPercentage = 20
```

# Flume Operation Example 1 (5)

- Configure the Flume sink.

```
server.sinks.s1.type = hdfs
server.sinks.s1.hdfs.path = /tmp/flume_avro
server.sinks.s1.hdfs.filePrefix = over_%{basename}
server.sinks.s1.hdfs.inUseSuffix = .tmp
server.sinks.s1.hdfs.rollInterval = 30
server.sinks.s1.hdfs.rollSize = 1024
server.sinks.s1.hdfs.rollCount = 10
server.sinks.s1.hdfs.batchSize = 1000
server.sinks.s1.hdfs.fileType = DataStream
server.sinks.s1.hdfs.maxOpenFiles = 5000
server.sinks.s1.hdfs.writeFormat = Writable
server.sinks.s1.hdfs.callTimeout = 10000
server.sinks.s1.hdfs.threadsPoolSize = 10
server.sinks.s1.hdfs.failcount = 10
server.sinks.s1.hdfs.fileCloseByEndEvent = true
server.sinks.s1.channel = ch1
```

# Flume Operation Example 1 (6)

- Name the configuration file of the Flume agent properties.properties.
- Upload the configuration file.

Parameter	Value
Flume->Flume	
flume.config.file	properties.pr <input type="button" value="Upload File"/> <input type="button" value="Download File"/>
FLUME_LOG_MAX_FILE_SIZE	50MB
FLUME_LOGFILE_MAXBACKUPI...	20
GC_OPTS	-Xms2G -Xmx4G - XX:CMSFullGCsBeforeCompaction=1 - XX:+UseConcMarkSweepGC - XX:+CMSParallelRemarkEnabled - XX:+UseCMSCompactAtFullCollection

# Flume Operation Example 1 (7)

- Produce data in the /tmp/log\_test directory.

```
mv /var/log/log.11 /tmp/log_test
```

- Check whether HDFS has data obtained from the sink.

```
hdfs dfs -ls /tmp/flume_avro
```

- In this case, log.11 is renamed log.11. COMPLETED by Flume, indicating that collection is successful.

# Flume Operation Example 2 (1)

- Description
  - This example shows how Flume ingests clickstream logs to Kafka in real time for subsequent analysis and processing.
- Prepare data.
  - Create a log directory named /tmp/log\_click on a node in the cluster.
  - Ingest data to Kafka topic\_1028.

# Flume Operation Example 2 (2)

- Configure the Flume source.

```
server.sources = a1
server.channels = ch1
server.sinks = s1
# the source configuration of a1
server.sources.a1.type = spooldir
server.sources.a1.spoolDir = /tmp/log_click
server.sources.a1.fileSuffix = .COMPLETED
server.sources.a1.deletePolicy = never
server.sources.a1.trackerDir = .flumespool
server.sources.a1.ignorePattern = ^$ 
server.sources.a1.batchSize = 1000
server.sources.a1.inputCharset = UTF-8
server.sources.a1.selector.type = replicating
jserver.sources.a1.basenameHeaderKey = basename
server.sources.a1.deserializer.maxBatchLine = 1
server.sources.a1.deserializer.maxLineLength = 2048
server.sources.a1.channels = ch1
```

# Flume Operation Example 2 (3)

- Configure the Flume channel.

```
# the channel configuration of ch1
server.channels.ch1.type = memory
server.channels.ch1.capacity = 10000
server.channels.ch1.transactionCapacity = 1000
server.channels.ch1.channlefullcount = 10
server.channels.ch1.keep-alive = 3
server.channels.ch1.byteCapacityBufferPercentage = 20
```

# Flume Operation Example 2 (4)

- Configure the Flume sink.

```
# the sink configuration of s1
server.sinks.s1.type = org.apache.flume.sink.kafka.KafkaSink
server.sinks.s1.kafka.topic = topic_1028
server.sinks.s1.flumeBatchSize = 1000
server.sinks.s1.kafka.producer.type = sync
server.sinks.s1.kafka.bootstrap.servers = 192.168.225.15:21007
server.sinks.s1.kafka.security.protocol = SASL_PLAINTEXT
server.sinks.s1.requiredAcks = 0
server.sinks.s1.channel = ch1
```

# Flume Operation Example 2 (5)

- Upload the configuration file to Flume.
- Run the Kafka command to view the data ingested from Kafka topic\_1028.

```
fi02host01:/opt/hadoopclient_0810/Kafka/kafka # bin/kafka-console-consumer.sh \
--topic topic_1028 --bootstrap-server 192.168.225.15:21007 \
--new-consumer --consumer.config config/consumer.properties
{"movie":"914","rate":"3","timeStamp":"978301968","uid":"1"}
{"movie":"594","rate":"4","timeStamp":"978302268","uid":"1"}
{"movie":"1197","rate":"3","timeStamp":"978302268","uid":"1"}
 {"movie":"2355","rate":"5","timeStamp":"978824291","uid":"1"}
 {"movie":"661","rate":"3","timeStamp":"978302109","uid":"1"}
 {"movie":"2804","rate":"5","timeStamp":"978300719","uid":"1"}
 {"movie":"3408","rate":"4","timeStamp":"978300275","uid":"1"}
 {"movie":"1193","rate":"5","timeStamp":"978300760","uid":"1"}
 {"movie":"1287","rate":"5","timeStamp":"978302039","uid":"1"}
```

# Summary

- This chapter introduces functions and application scenarios of Flume and gives details on its basic concepts, features, reliabilities, and configurations. After learning this chapter, you will be able to understand the functions, application scenarios, and configurations, and usage of Flume.

# Quiz

1. What is Flume? What is it used for?
2. What are the key features of Flume?
3. What are the functions of a source, channel, and sink?

# Recommendations

---

- Huawei Cloud Official Web Link:
  - <https://www.huaweicloud.com/intl/en-us/>
- Huawei MRS Documentation:
  - <https://www.huaweicloud.com/intl/en-us/product/mrs.html>
- Huawei TALENT ONLINE:
  - <https://e.huawei.com/en/talent/#/>

# Thank you.

把数字世界带入每个人、每个家庭、  
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and  
organization for a fully connected,  
intelligent world.

Copyright©2020 Huawei Technologies Co., Ltd.  
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.



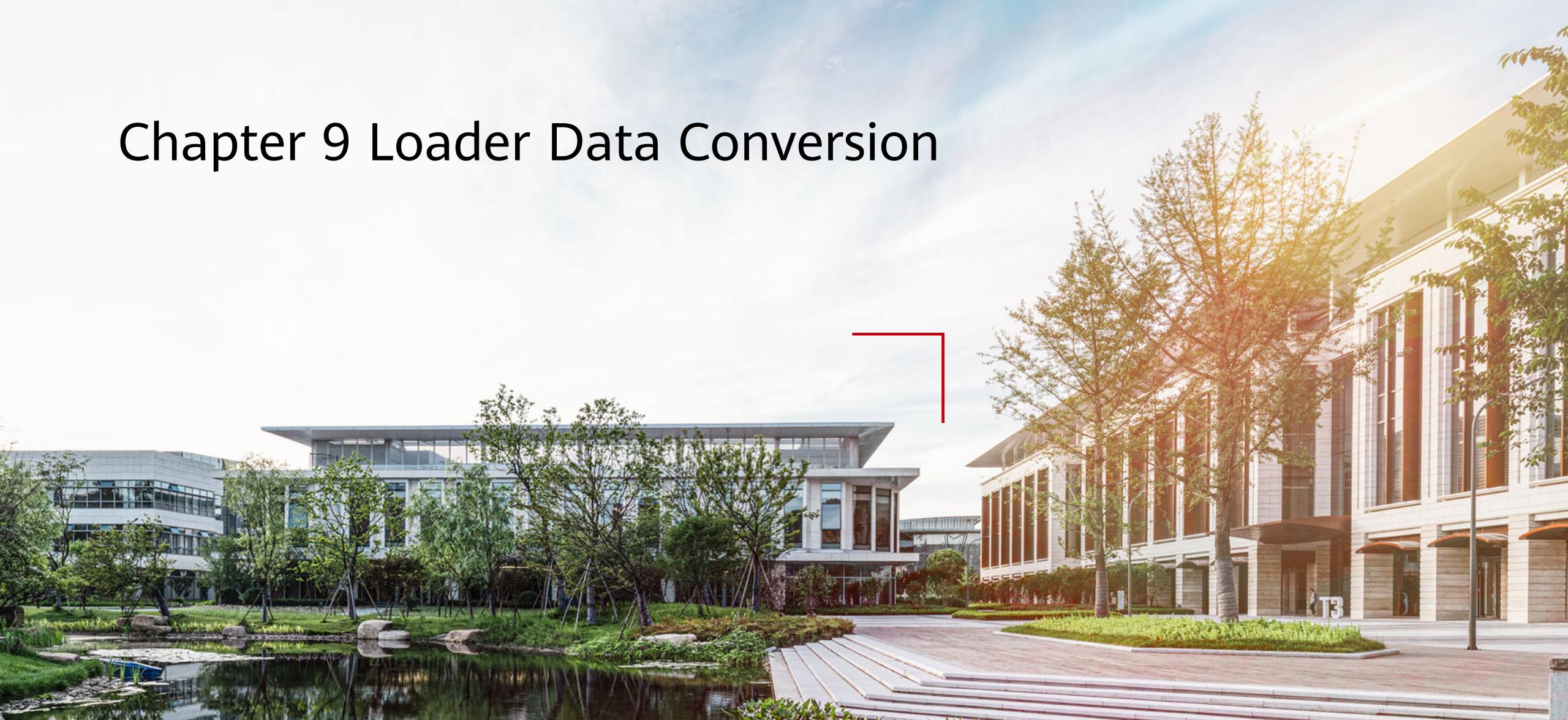
# Revision Record

Do Not Print this Page

Course Code	Product	Product Version	Course Version
H13-711	MRS		V3.0

Author/ID	Date	Reviewer/ID	New/ Update
Mao Jian/mwx711840	2020.03.31	Wang Yongjie/wwx769662	Update
Mao Jian/mwx711840	2020.07.31	Yan Hua/ywx416015	New

# Chapter 9 Loader Data Conversion



# Foreword

---

- Loader is used for efficient data import and export between the big data platform and structured data storage (such as relational databases). Based on the open-source Sqoop 1.99.x, Loader functions have been enhanced.

# Objectives

- On completion of this course, you will be able to:
  - Know what is Loader
  - Know what Loader can do
  - Understand the main features of Loader
  - Understand the system architecture of Loader
  - Manage Loader jobs
  - Monitor Loader jobs

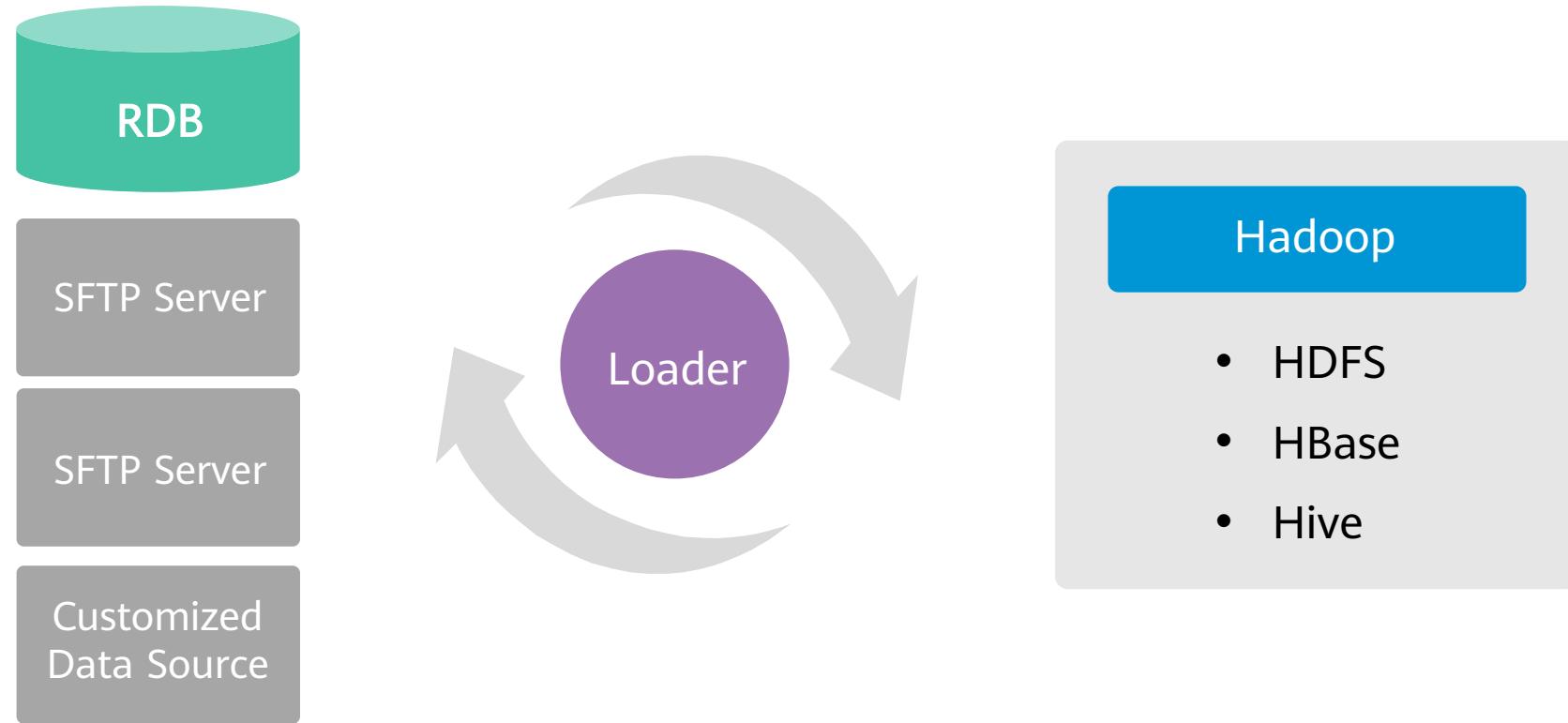
# Contents

- 1. Introduction to Loader**
2. Loader Job Management

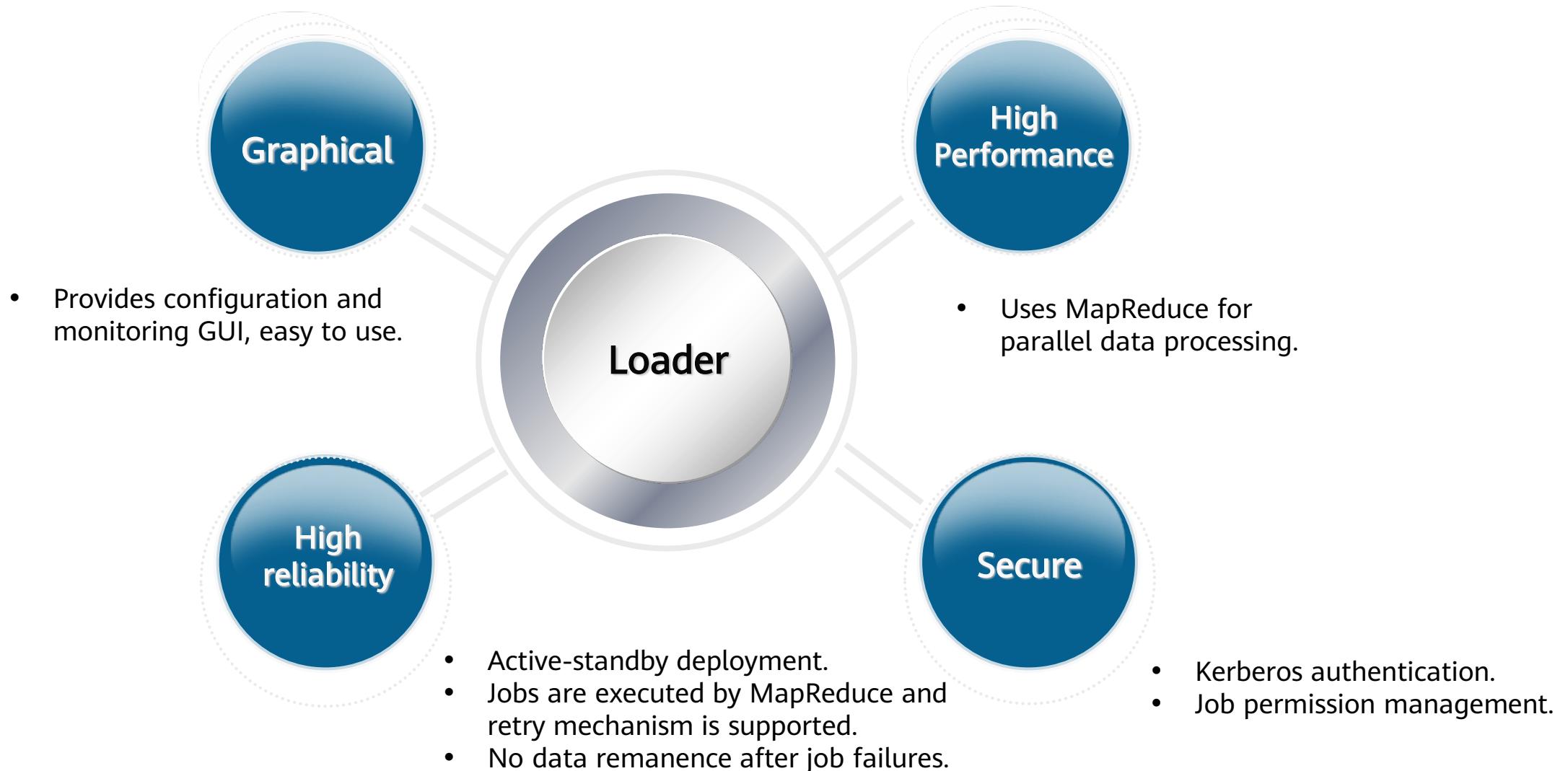
# Loader Definition

- Loader is a data loading tool used to exchange data and files between the big data platform and relational databases/file systems. Loader provides a visualized wizard-based interface for job configuration and management. It also provides a job scheduler to periodically execute Loader jobs. On the user interface, you can specify multiple data sources, configure data cleansing and conversion procedures, and configure the cluster storage system.

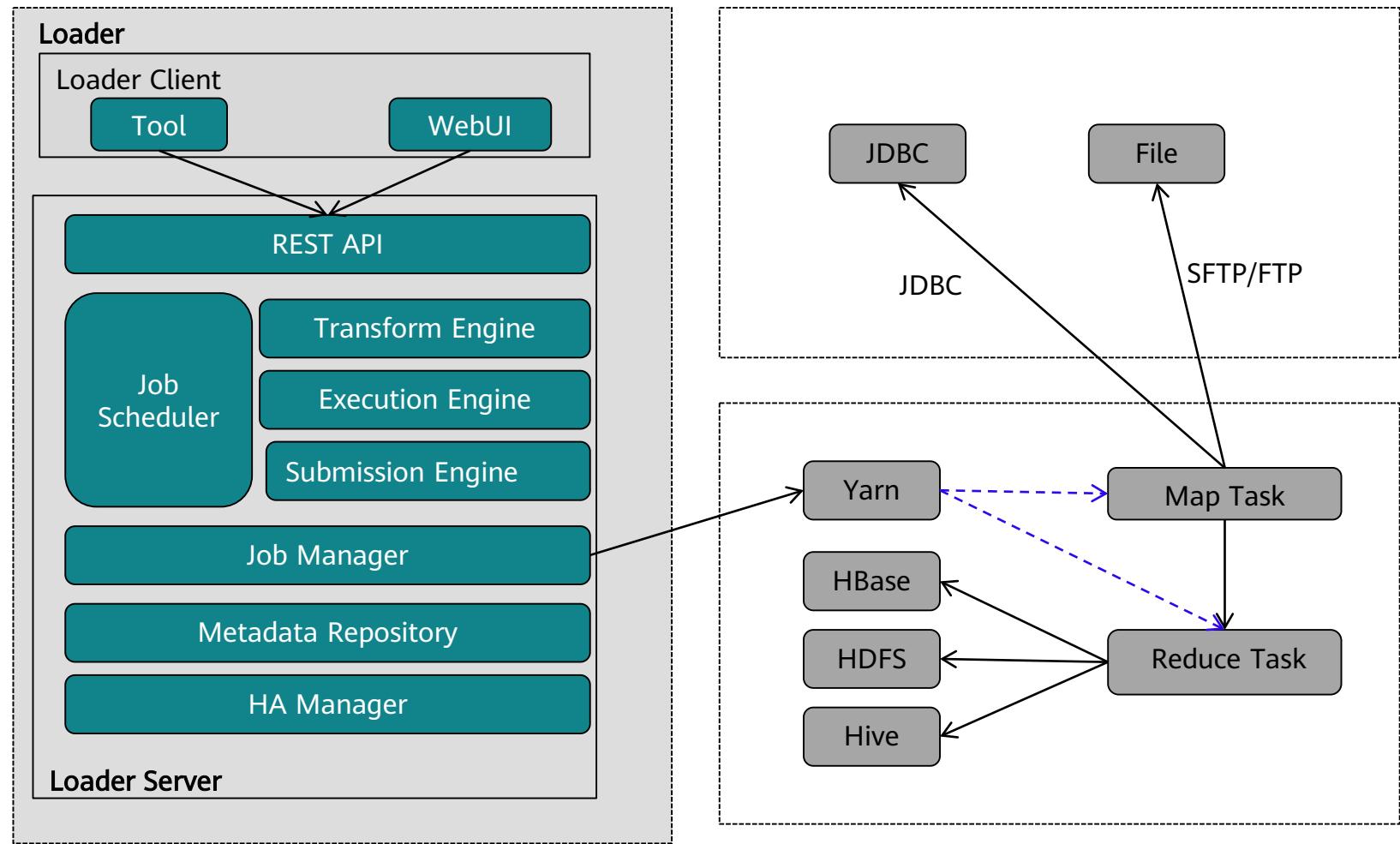
# Application Scenarios



# Loader Features



# Loader Modular Architecture



# Loader Modules - Description

Term	Description
Loader Client	Provides a web user interface (WebUI) and a command-line interface (CLI).
Loader Server	Processes operation requests sent from the client, manages connectors and metadata, submits MapReduce jobs, and monitors MapReduce job statuses.
REST API	Provides RESTful APIs (HTTP + JSON) to process requests from the client.
Job Scheduler	A simple job scheduling module. It supports periodical execution of Loader jobs.
Transform Engine	Processes data transformation. Supports field combination, string cutting, string reversing, and other data transformations.
Execution Engine	Execution engine of Loader jobs. It provides detailed processing logic of MapReduce jobs.
Submission Engine	Submission engine of Loader jobs. It supports the submission of jobs to MapReduce.
Job Manager	Manages Loader jobs, including creating, querying, updating, deleting, activating, deactivating, starting, and stopping jobs.
Metadata Repository	Stores and manages data about Loader connectors, transformation procedures, and jobs.
HA Manager	Manages the active/standby status of Loader servers. Two Loader servers are deployed in active/standby mode.

# Contents

1. Introduction to Loader
2. Loader Job Management

# Loader Service Status

- Choose Services > Service Loader, and go to the Service Status page.

The screenshot shows the Service Status page for the Service Loader. At the top, there is a navigation bar with tabs: Dashboard, Services, Hosts, and Alarms. Below the navigation bar, the breadcrumb path shows Service Loader > Service Status. There are four tabs in the main content area: Service Status (which is selected and underlined), Instance, Service Configuration, and Resources. Below these tabs are three buttons: Start Service, Stop Service, and Download Client. The main content area is titled "Loader Summary" and contains three data rows: Health Status (Good), Configuration Status (Synchronized), and Version (2.0.0). The "Configuration Status" row is highlighted with a light gray background.

Health Status	Good
Configuration Status	Synchronized
Version	2.0.0

# Loader Job Management Page

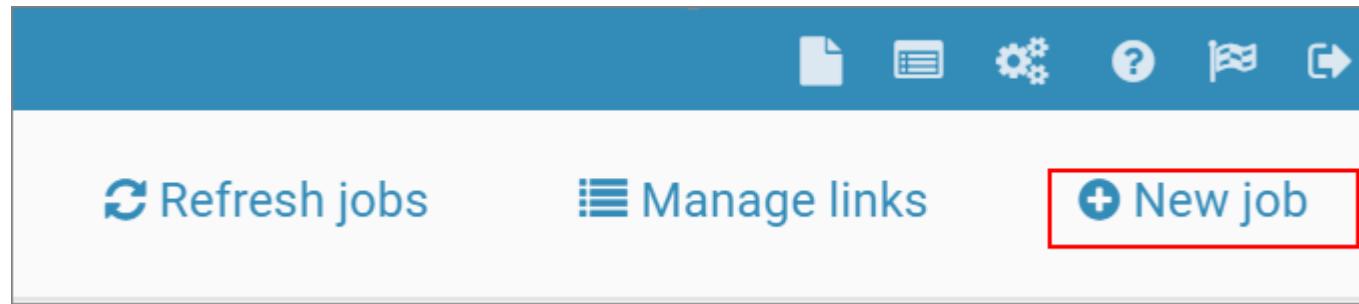
- Log in to MRS Manager and choose **Services > Hue**. Click **Hue (Active)** to access the Hue page.
- Choose **Data Browsers > Sqoop**. The Sqoop job page is displayed.
- A job is used to describe the process of extracting, transforming, and loading data from a data source to a destination. It includes the data source location, data source attributes, data transformation rules, and target attributes.

# Loader Job Management - Job Conversion Rules

- Loader conversion operators:
  - Long integer to time conversion: Implements the conversion between long integer values and date types.
  - Null value conversion: Replaces a null value with a specified value.
  - Constant field adding: Generates a constant field.
  - Random value conversion: Generates a random number field.
  - Concatenation conversion: Combines existing fields to generate new fields.
  - Delimiter conversion: Uses specified delimiters to separate existing fields to obtain new fields.
  - Modulo conversion: Generates new fields by performing modulo operation on existing fields.
  - Character string cutting: Specifies the start and end positions to cut an existing character string field and generate a new field.

# Creating a Loader Job - Basic Information

- Open the Sqoop page and click **New Job**.



- Configure basic job information

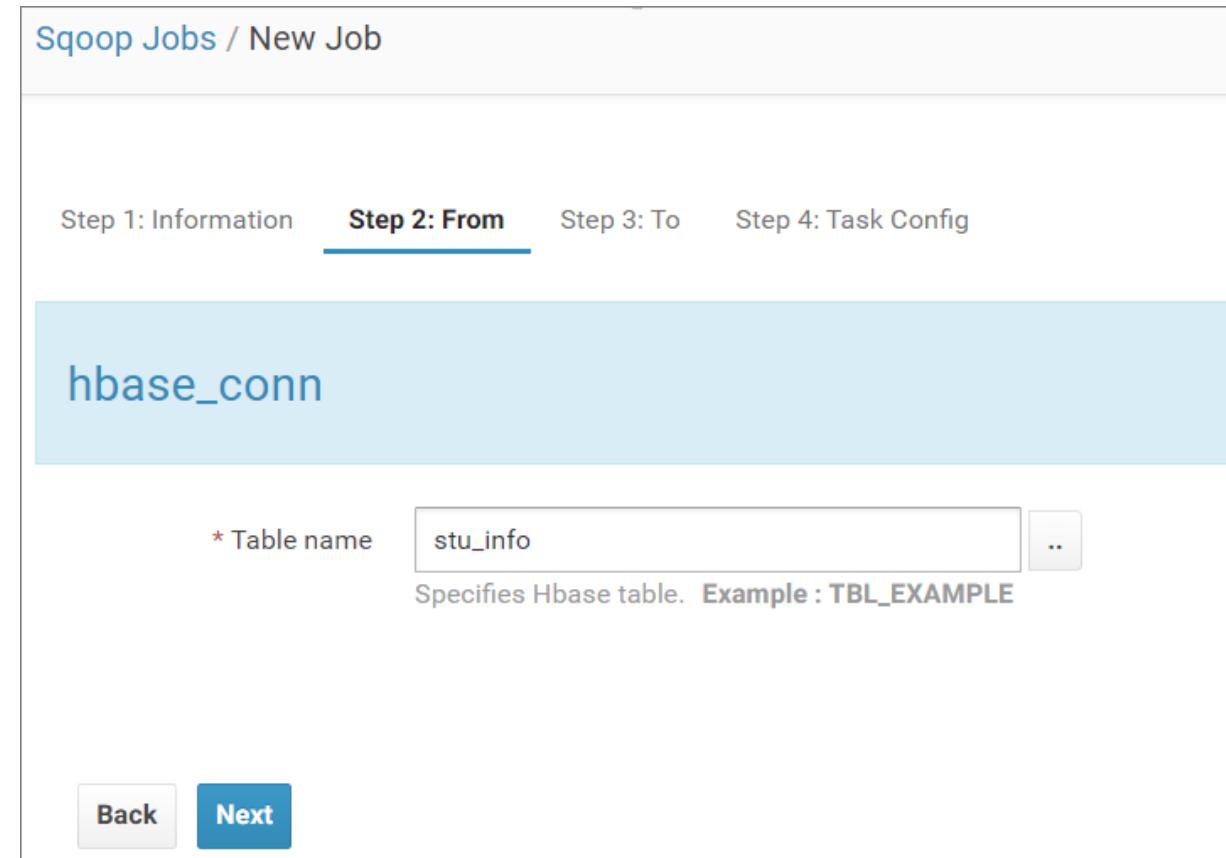
A screenshot of a configuration form for a new job. It contains three fields:

- \* Name: A text input field containing "job\_hbase\_to\_hdfs".
- \* From link: A dropdown menu showing "hbase\_conn" as the selected option.
- \* To link: A dropdown menu showing "hdfs\_conn" as the selected option.

The entire form is enclosed in a light gray border.

# Creating a Loader Job - Configuring HBase Information

- HBase\_conn table name configuration



# Creating a Loader Job - Configuring HDFS Information

Step 1: Information Step 2: From **Step 3: To** Step 4: Task Config

### hdfs\_conn

\* Output directory  HDFS directory where transferred data will be written to. Example : /user/sqoop/output

\* File format  File format that should be used for transferred data. Example : CSV\_FILE

Compression codec  Compression codec that should be used to compress transferred data. Example : SNAPPY

Overwrite   If set to false, job will fail if output directory already exists. If set to true then imported data will

[Show Senior Parameter](#)

[Back](#) [Next](#)

# Creating a Loader Job - Task Configuration

Step 1: Information   Step 2: From   Step 3: To   **Step 4: Task Config**

## Task Config

Extractors  Number of extractors when retrieving data. Example : 3

Show Senior Parameter

[Back](#) [Save](#) **Save and execute**

# Monitoring Job Execution Statuses

- View the execution statuses of all jobs.
  - Go to the Sqoop job management page.
  - All the current jobs and the last execution statuses of the jobs are displayed.
  - Select a job and click the button above the list or in the **Operation** column on the right to perform the operation.

# Monitoring Job Execution Statuses - Job History

- View the historical execution records of a specified job.
  - Select a job and click **History** in the **Operation** column. The job history page is displayed.
  - The page displays the start time, running time (in seconds), status, failure cause, number of read rows and files, number of written rows and files, number of skipped rows and files, dirty data link, and MapReduce log link of each job execution.

# Client Scripts

- In addition to the GUI, Loader provides a complete set of shell scripts. These scripts can be used to add, delete, query, and modify data sources and jobs, start and stop jobs, view job statuses, and check whether jobs are running.
- These scripts are listed as follows:
  - lt-ctl: job control tool, which is used to query job statuses, start and stop jobs, and check whether jobs are running.
  - lt-ucj: job management tool, which is used to query, create, modify, and delete jobs.
  - lt-ucc: data source management tool, which is used to query, create, modify, and delete data source connection information.

# Summary

- This chapter describes the Loader about its main functions, features, job management, and job monitoring.

# Quiz

1. (T or F) True or false: MRS Loader supports only data import and export between relational databases and Hadoop's HDFS or HBase.
2. (T or F) True or false: Conversion steps must be configured for a Loader job.
3. (Multiple-choice) Which of the following statements are correct? ( )
  - A. No data remanence of the original file is left if a job fails after running for a period of time.
  - B. Dirty data refers to the data that does not comply with the conversion rules.
  - C. Loader client scripts can only be used to submit jobs.
  - D. After a human-machine account is created, all Loader jobs can be operated.

# Quiz

4. Which of the following statements are correct? ( )
- A. After Loader submits a job to MapReduce for execution, the job execution will fail if Loader is faulty.
  - B. After Loader submits a job to MapReduce for execution, the job execution will be retried if a Mapper fails to execute the job.
  - C. If a job execution fails, you need to manually clear the data remanence.
  - D. After Loader submits a job to MapReduce for execution, it cannot submit another job before the preceding job completes.

# Recommendations

---

- Huawei Cloud Official Web Link:
  - <https://www.huaweicloud.com/intl/en-us/>
- Huawei MRS Documentation:
  - <https://www.huaweicloud.com/intl/en-us/product/mrs.html>
- Huawei TALENT ONLINE:
  - <https://e.huawei.com/en/talent/#/>

# Thank you.

把数字世界带入每个人、每个家庭、  
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and  
organization for a fully connected,  
intelligent world.

Copyright©2020 Huawei Technologies Co., Ltd.  
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.



# Chapter 10 Kafka - Distributed Publish-Subscribe Messaging System



# Foreword

---

- This chapter describes the basic concepts, architecture, and functions of Kafka. It is important to know how Kafka ensures reliability for data storage and transmission and how historical data is processed.

# Objectives

- On completion of this course, you will be able to know:
  - Basic concepts of the message system
  - Kafka system architecture

# Contents

- 1. Introduction**
2. Architecture and Functions
3. Data Management

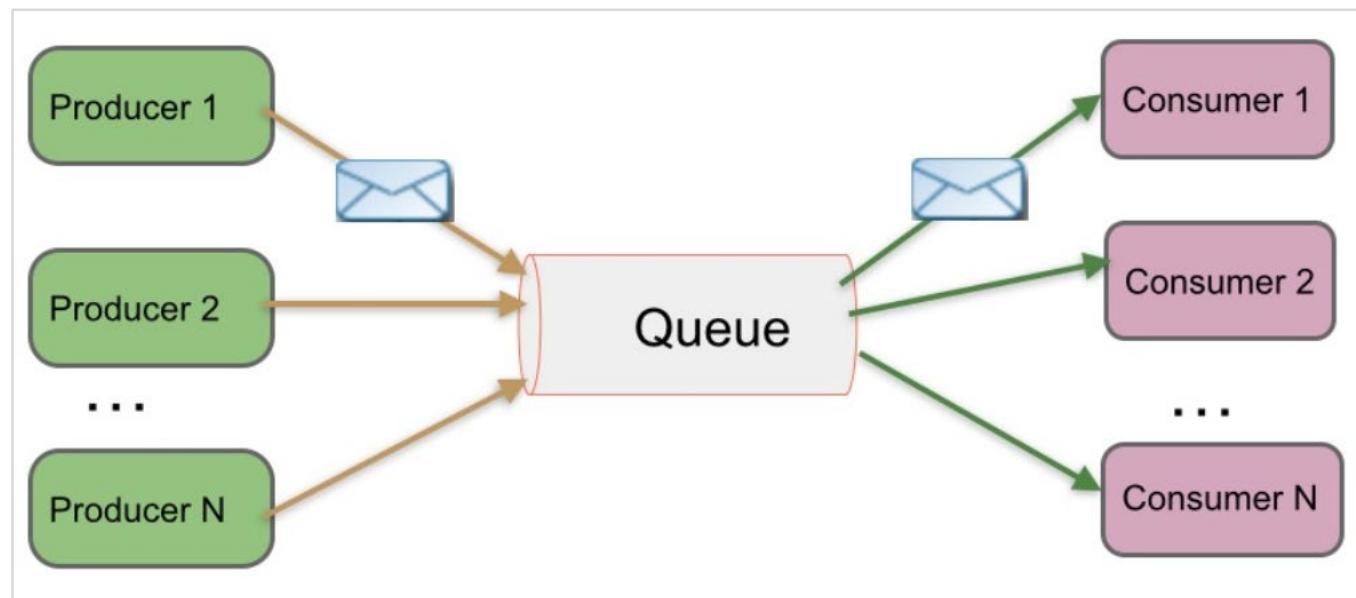
# Introduction

- Kafka is a distributed, partitioned, replicated, and ZooKeeper-based messaging system. It supports multi-subscribers and was originally developed by LinkedIn.
- Main application scenarios: the log collection system and message system
- Robust message queue is a foundation of distributed messaging, in which messages are queued asynchronously between client applications and the messaging system. Two types of messaging patterns are available: point to point messaging, and publish-subscribe (pub-sub) messaging. Most of the messaging patterns follow pub-sub. Kafka implements a pub-sub messaging pattern.



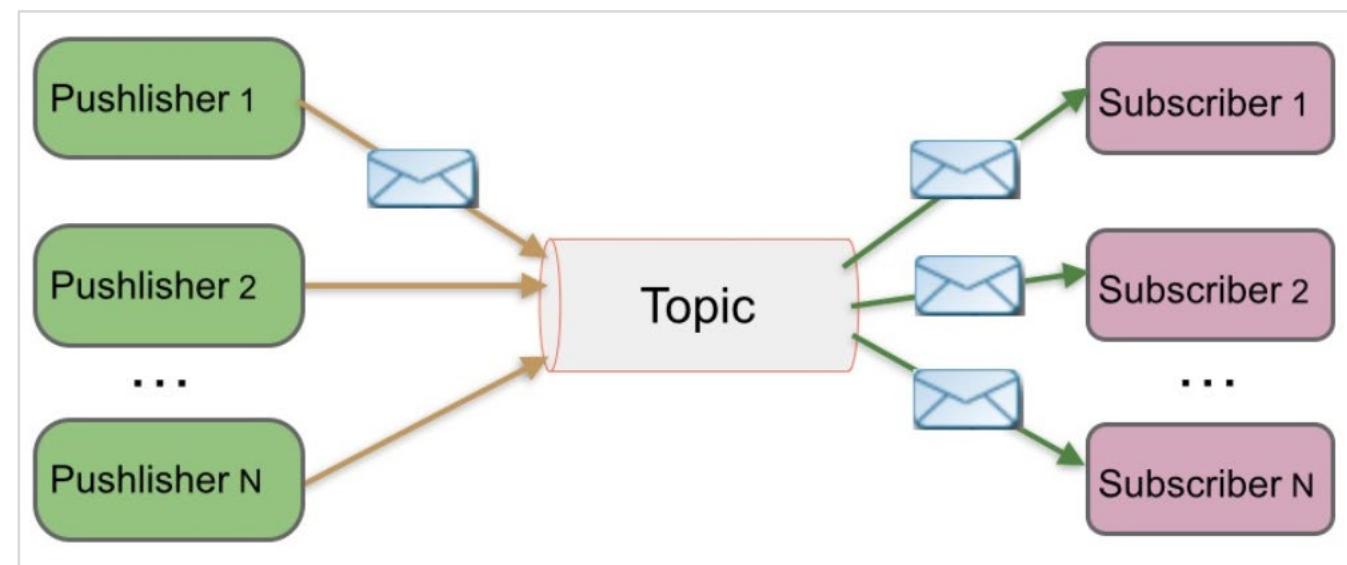
# Point-to-Point Messaging

- In a point-to-point messaging system, messages are persisted in a queue. In this case, one or more consumers consume the messages in the queue. However, a message can be consumed by a maximum of one consumer only. Once a consumer reads a message in the queue, it disappears from that queue. This pattern ensures the data processing sequence even when multiple consumers consume data at the same time.



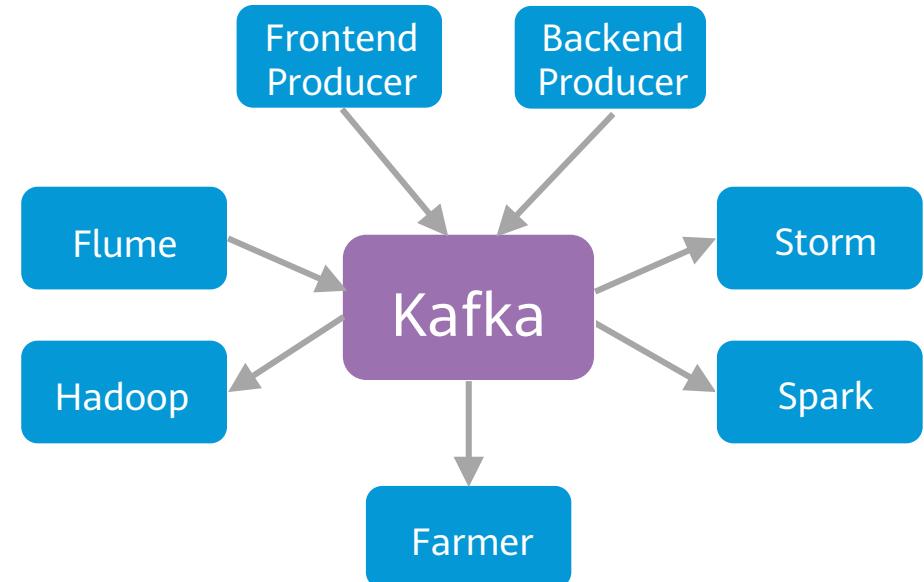
# Publish-Subscribe Messaging

- In the publish-subscribe messaging system, messages are persisted in a topic. Unlike point-to-point messaging system, a consumer can subscribe to one or more topics and consume all the messages in those topics. One message can be consumed by more than one consumer. A consumed message is not deleted immediately. In the publish-subscribe messaging system, message producers are called publishers and message consumers are called subscribers.



# Kafka Features

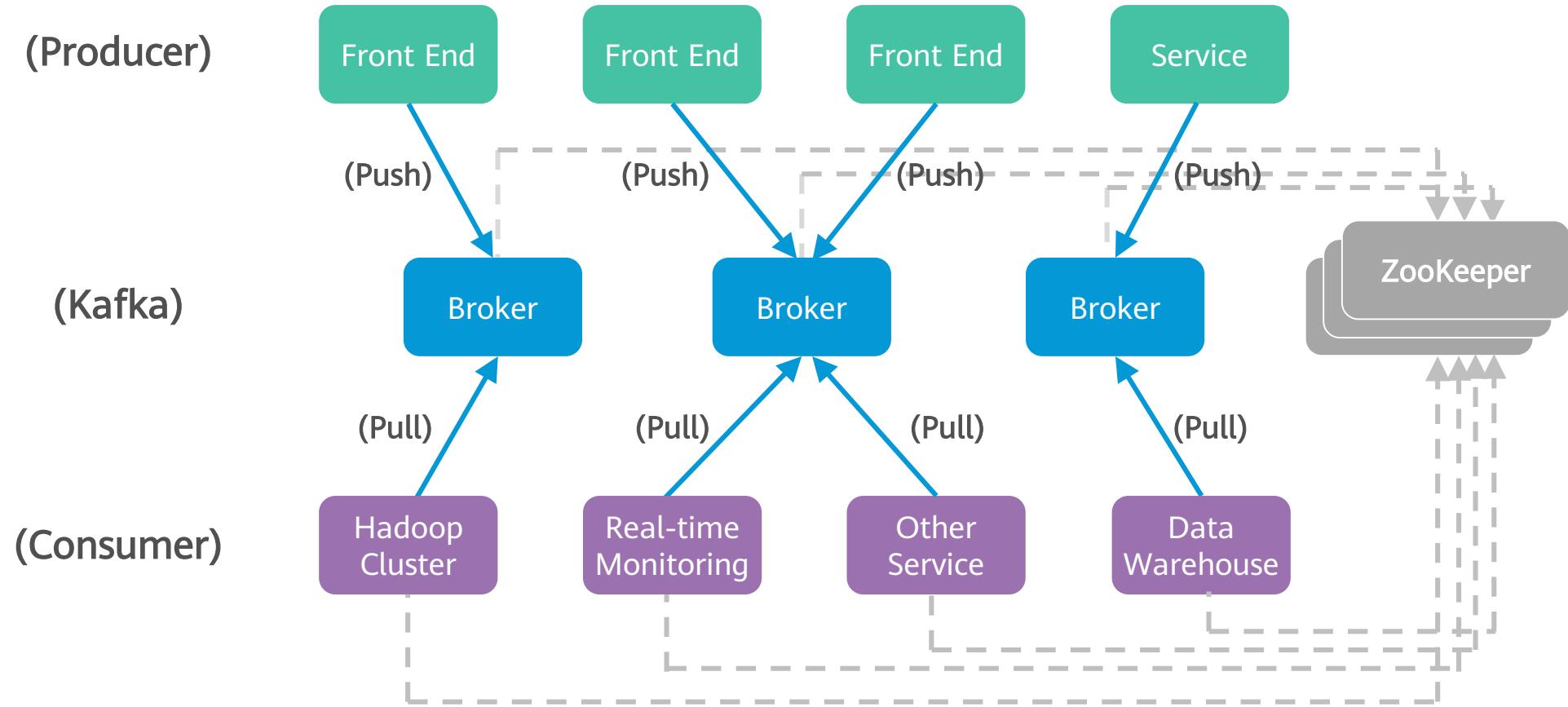
- Kafka can persist messages in a time complexity of  $O(1)$ , and can maintain data access performance in constant time even in the face of terabytes of data.
- Kafka provides high throughput. Even on cheap commercial machines, a single-node system can transmit 100,000 messages per second.
- Kafka supports message partitioning and distributed consumption, and ensures that messages are transmitted in sequence in each partition.
- Kafka supports offline and real-time data processing.
- Kafka supports scale-out.



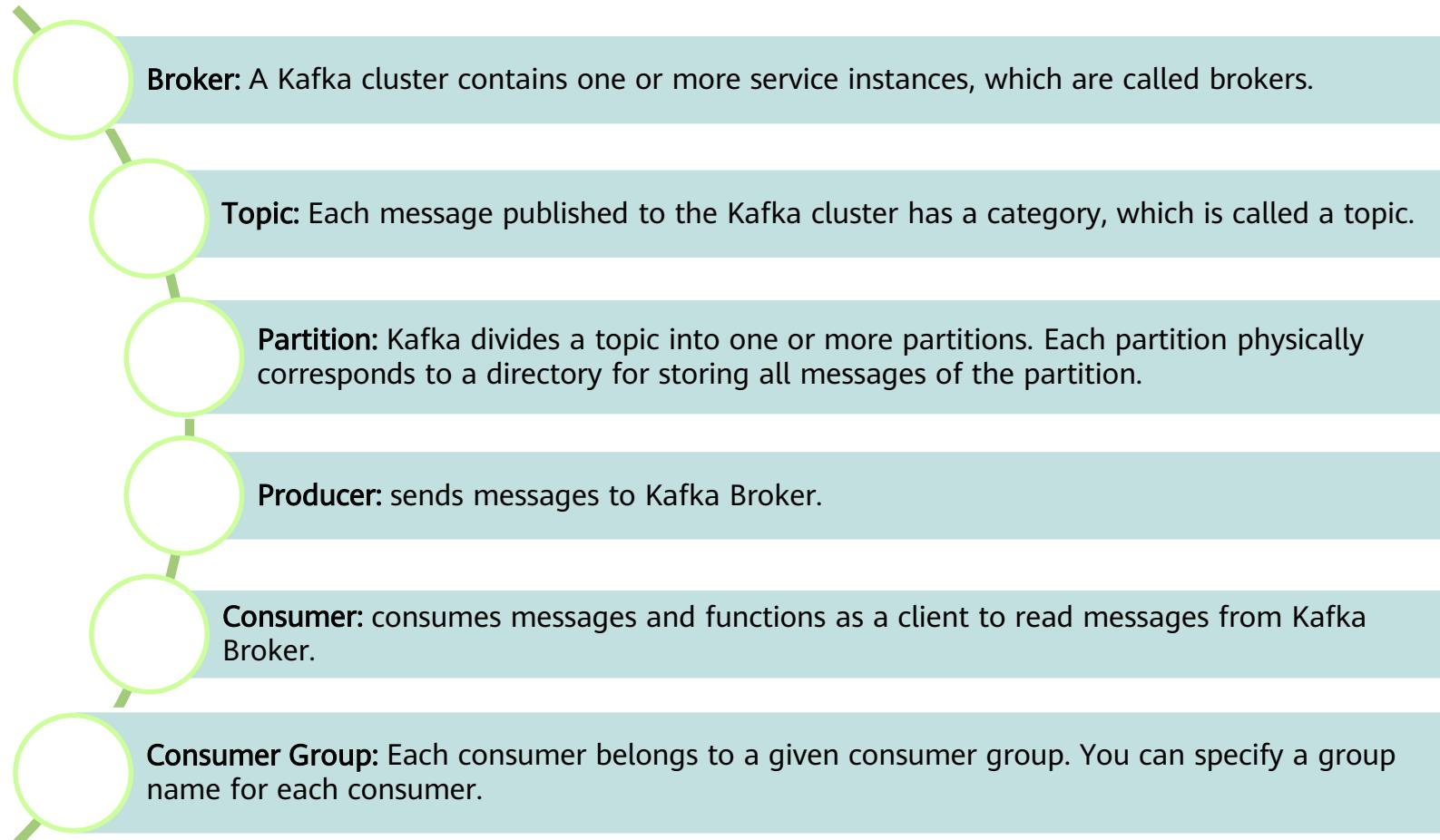
# Contents

1. Introduction
- 2. Architecture and Functions**
3. Data Management

# Kafka Topology

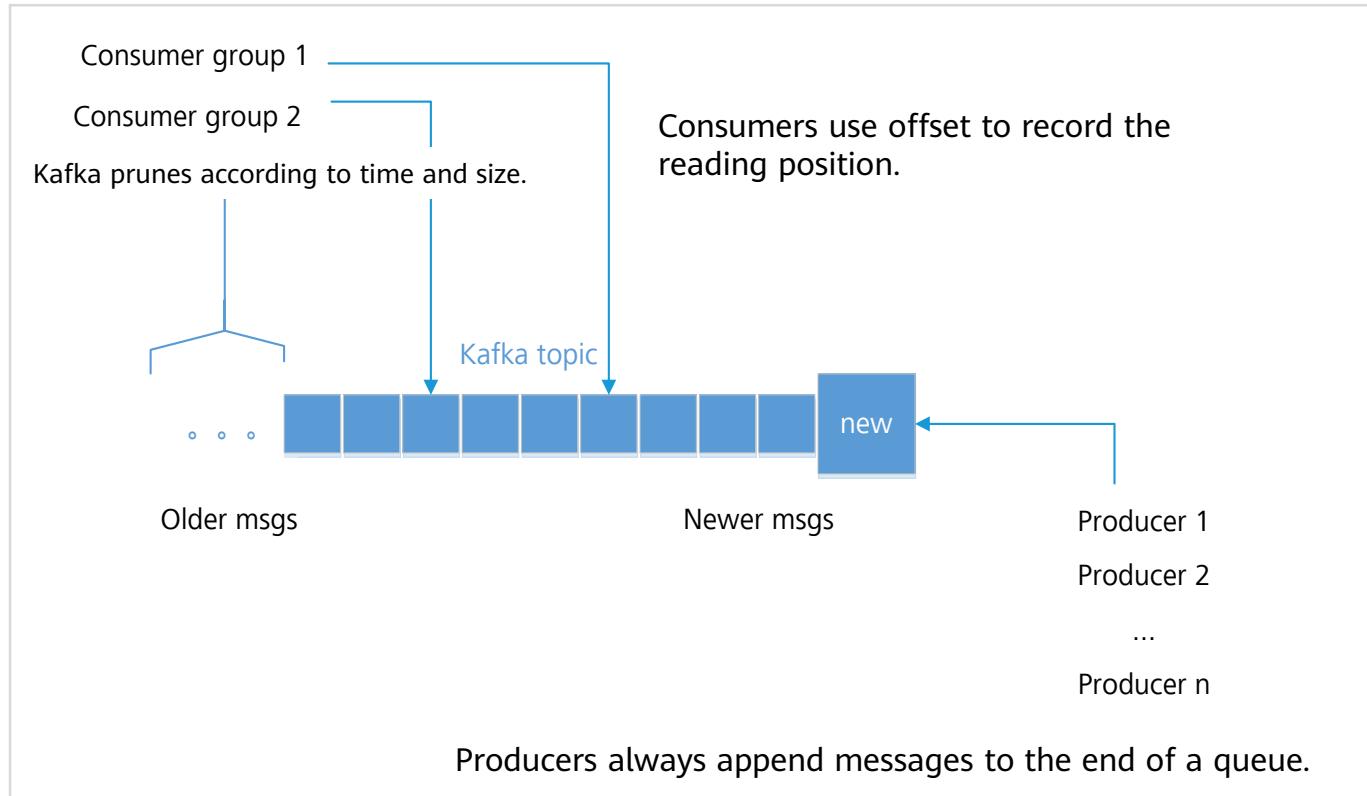


# Kafka Basic Concepts



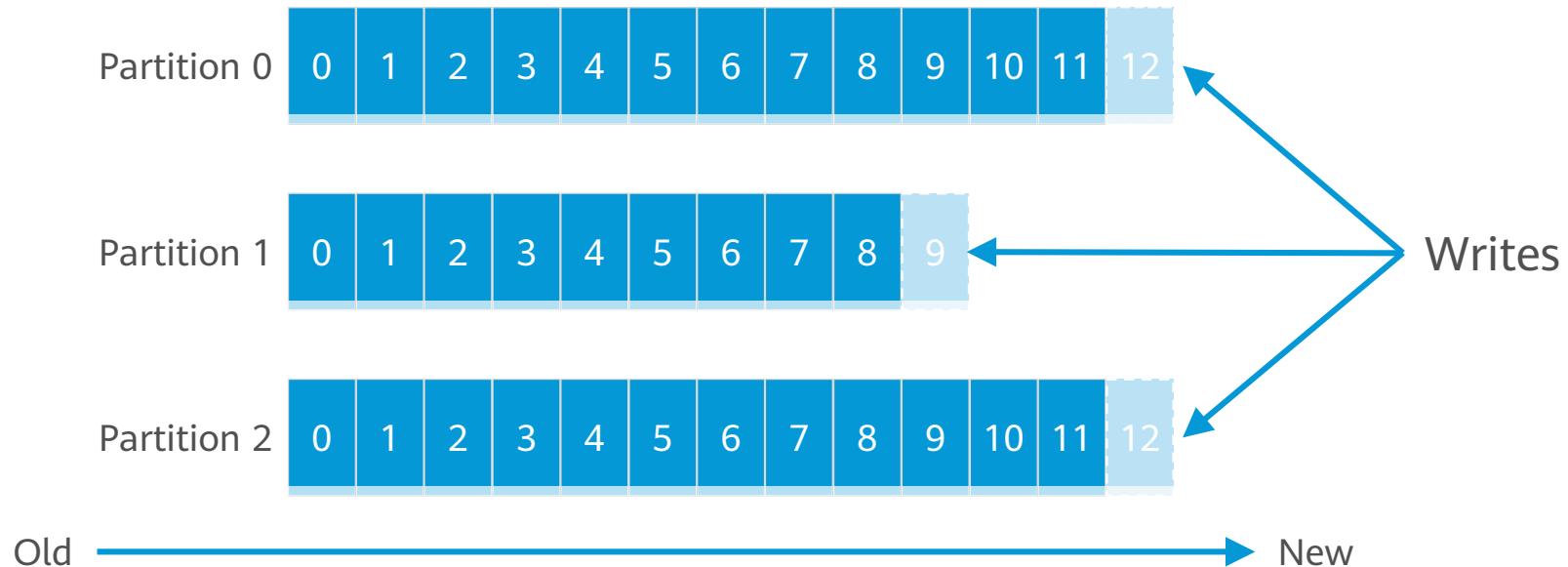
# Kafka Topics

- Each message published to Kafka belongs to a category, which is called a topic. A topic can also be interpreted as a message queue. For example, weather can be regarded as a topic (queue) that stores daily temperature information.



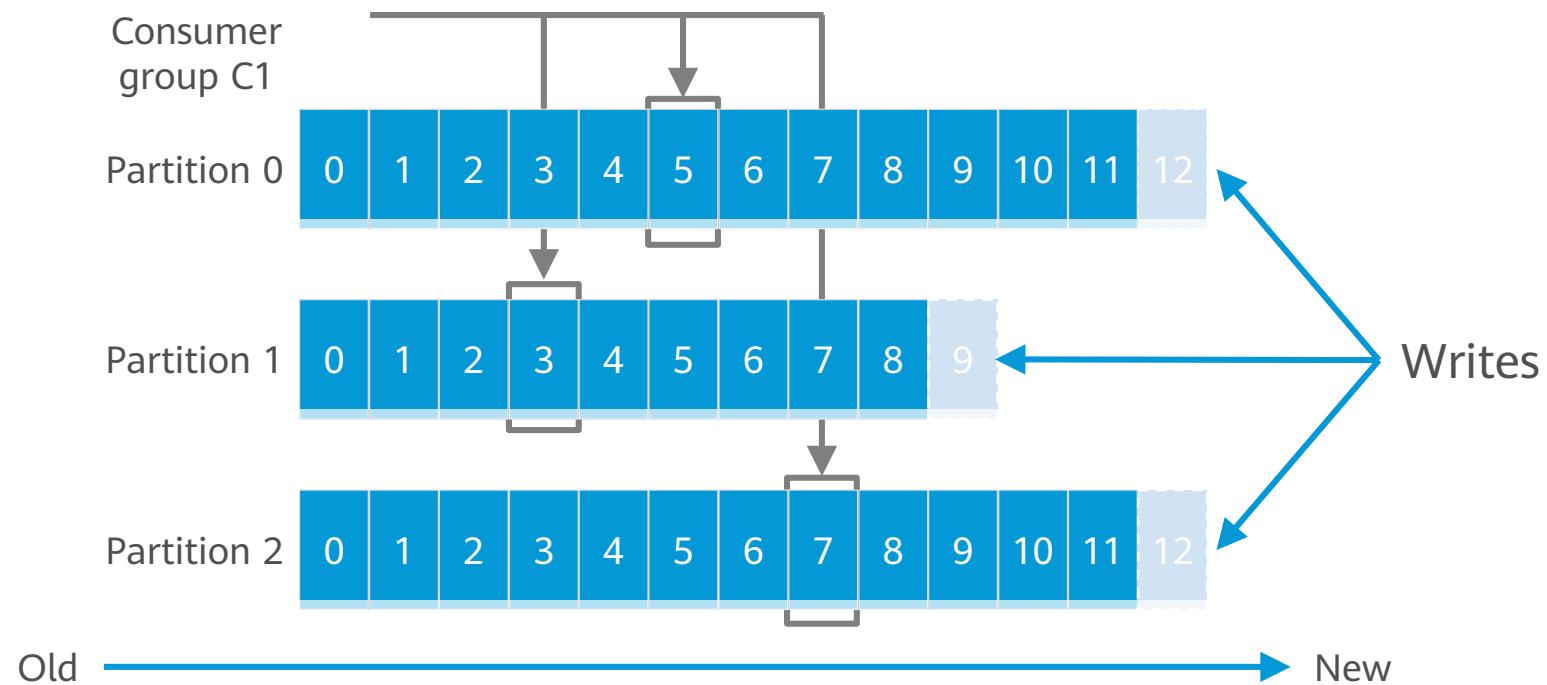
# Kafka Partition

- To improve the throughput of Kafka, each topic is physically divided into one or more partitions. Each partition is an ordered and immutable sequence of messages. **Each partition physically corresponds to a directory for storing all messages and index files of the partition.**



# Kafka Partition Offset

- The position of each message in a log file is called offset, which is a long integer that uniquely identifies a message. Consumers use offsets, partitions, and topics to track records.

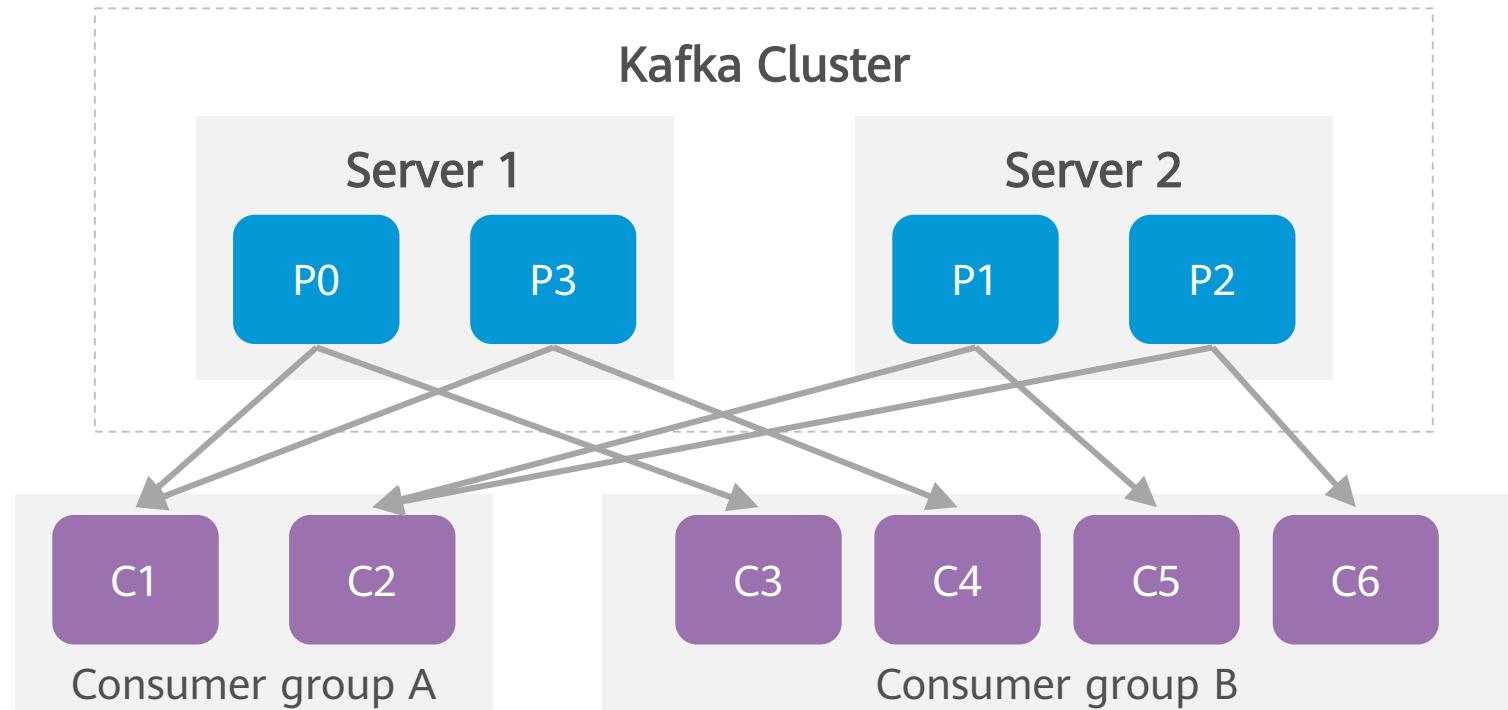


# Kafka Offset Storage Mechanism

- After a consumer reads the message from the broker, the consumer can commit a transaction, which saves the offset of the read message of the partition in Kafka. When the consumer reads the partition again, the reading starts from the next message.
- This feature ensures that the same consumer does not consume data repeatedly from Kafka.
- Offsets of consumer groups are stored in the `_consumer_offsets` directory.
  - Formula: `Math.abs(groupId.hashCode()) % 50`
  - Go to the `kafka-logs` directory and you will find multiple sub-directories. This is because kafka generates 50 `_consumer_offsets-n` directories by default.

# Kafka Consumer Group

- Each consumer belongs to a consumer group. Each message can be consumed by multiple consumer groups but only one consumer in a consumer group. That is, data is shared between groups, but exclusive within a group.



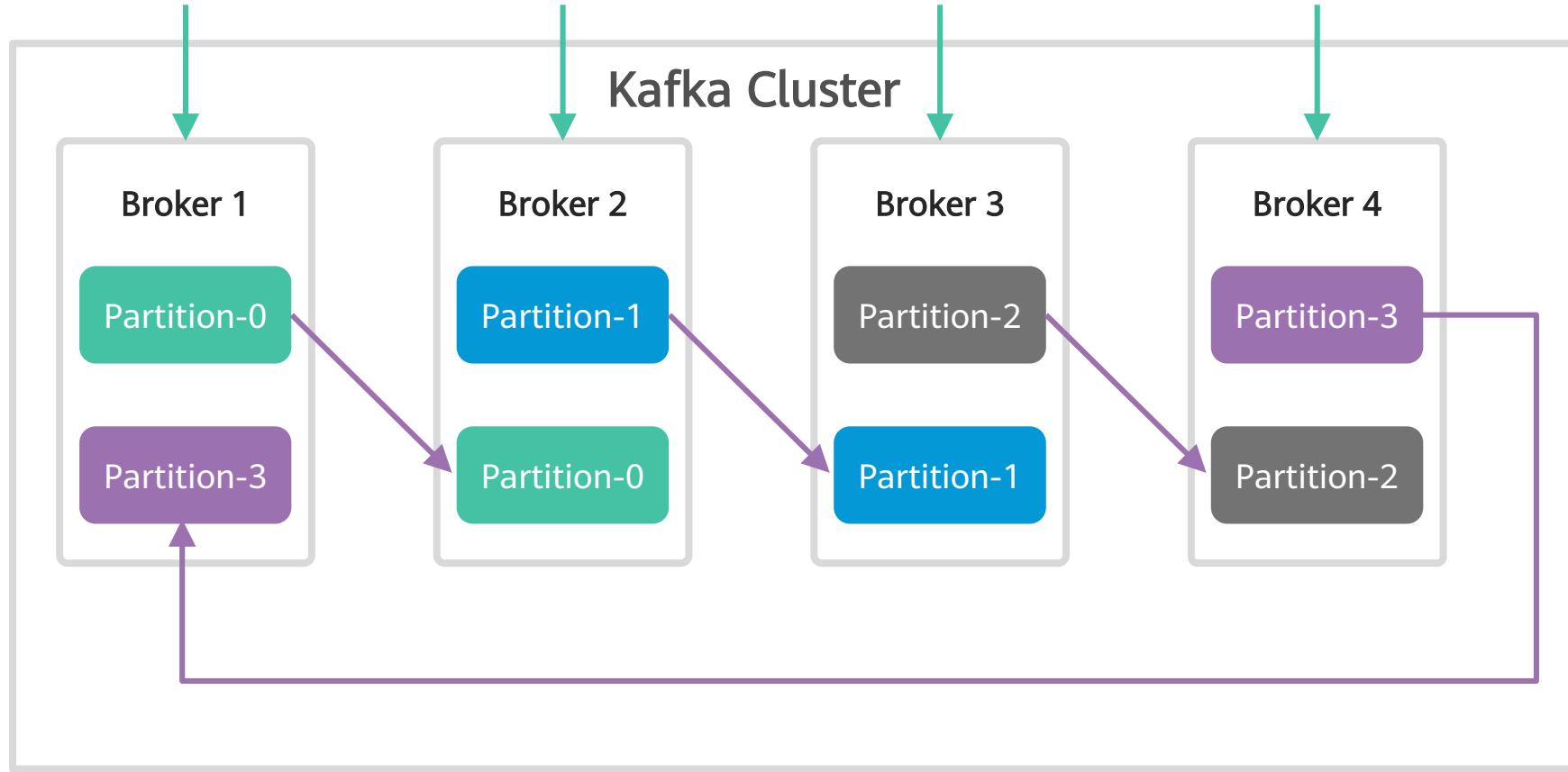
# Other Important Concepts

- Replica:
  - Refers to a replica of a partition, which guarantees high availability of partitions.
- Leader:
  - A role in a replica. Producers and consumers only interact with the leader.
- Follower:
  - A role in a replica, which replicates data from the leader.
- Controller:
  - A server in a Kafka cluster, which is used for leader election and failovers.

# Contents

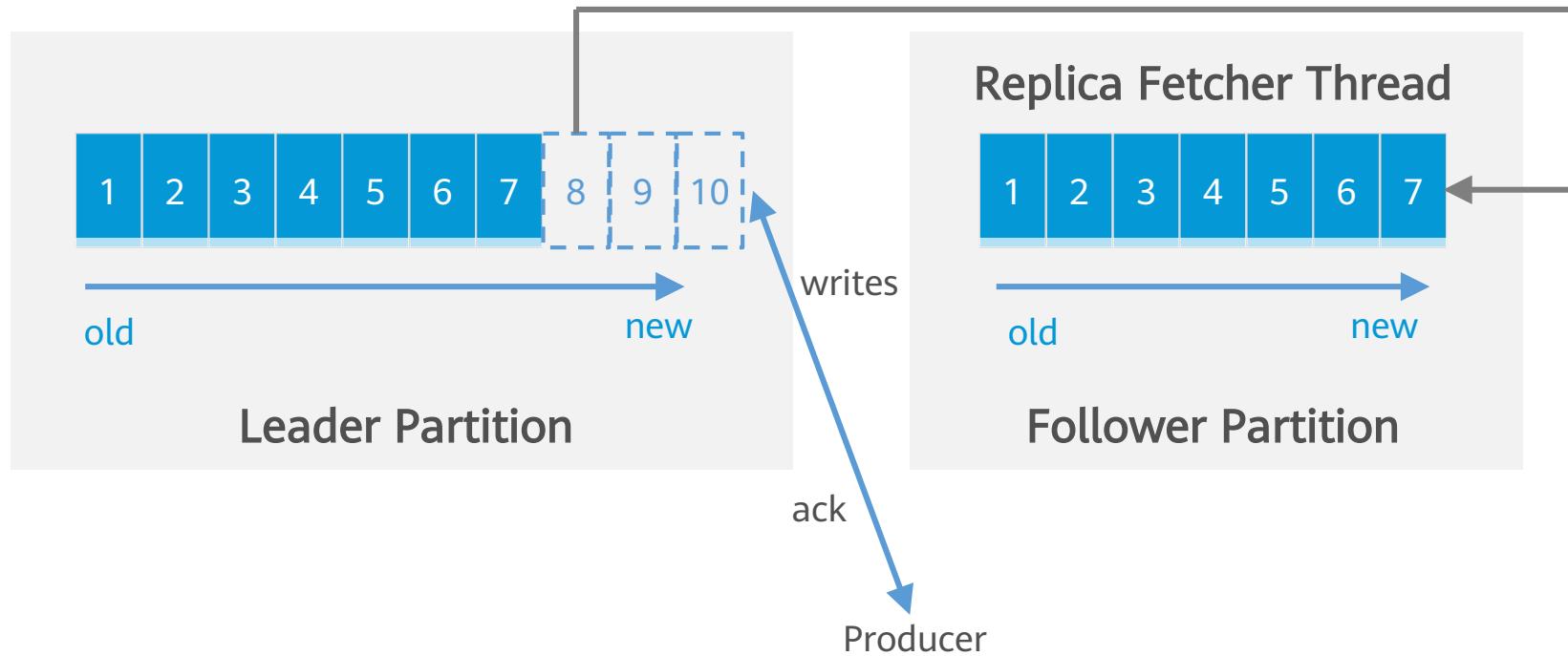
1. Introduction
2. Architecture and Functions
3. Data Management
  - Data Storage Reliability
  - Data Transmission Reliability
  - Old Data Processing Methods

# Kafka Partition Replica



# Kafka Partition Replica

## Follower Pulls Data from Leader



# Kafka HA

- A partition may have multiple replicas (equivalent to **default.replication.factor=N** in the **server.properties** configuration).
- If there is no replica, once the broker breaks down, all the partition data on the broker cannot be consumed, and the producer cannot store data in the partition.
- After replication is introduced, a partition may have multiple replicas. One of these replicas is elected to act as the leader. Producers and consumers interact with only the leader, and the rest of the replicas act as the followers to copy messages from the leader.

# Leader Failover (1)

- A new leader needs to be elected in case of the failure of an existing one. When a new leader is elected, **the new leader must have all the messages committed by the old leader.**
- According to the write process, all replicas in ISR (in-sync replicas) have fully caught up with the leader. Only replicas in ISR can be elected as the leader.
- For  $f+1$  replicas, a partition can tolerate the number of  $f$  failures without losing committed messages.

# Leader Failover (2)

- If all replicas do not work, there are two solutions:
  - Wait for a replica in the ISR to come back to life and choose this replica as the leader. This can ensure that no data is lost, but may take a long time.
  - Choose the first replica (not necessarily in the ISR) that comes back to life as the leader. This cannot ensure that no data is lost, but the unavailability time is relatively short.

# Contents

1. Introduction
2. Architecture and Functions
3. Data Management
  - Data Storage Reliability
  - Data Transmission Reliability
  - Old Data Processing Methods

# Kafka Data Reliability

- All Kafka messages are persisted on the disk. When you configure Kafka, you need to set the replication for a topic partition, which ensures data reliability.
- How data reliability is ensured during message delivery?

# Message Delivery Semantics

- There are three data delivery modes:
  - At Most Once
    - Messages may be lost.
    - Messages are never redelivered or reprocessed.
  - At Least Once
    - Messages are never lost.
    - Messages may be redelivered and reprocessed.
  - Exactly Once
    - Messages are never lost.
    - Messages are processed only once.

# Reliability Assurance - Idempotency

- An idempotent operation is an operation that is performed multiple times and has the same impact as an operation that is performed only once.
- Principles:
  - Each batch of messages sent to Kafka will contain a sequence number that the broker will use to deduplicate data.
  - The sequence number is made persistent to the replica log. Therefore, even if the leader of the partition fails, other brokers take over the leader. The new leader can still determine whether the resent message is duplicate.
  - The overhead of this mechanism is very low: each batch of messages has only a few additional fields.

# Reliability Assurance - Acks Mechanism

- The producer needs the acknowledgement (ACK) signal sent by the server after receiving the data. This configuration refers to the number of such ACK signals that the producer needs. This configuration actually represents the availability of data backup. The following settings are common options:
  - acks=0: Zero indicates that the producer will not wait for any acknowledgment from the server at all. The record will be immediately added to the socket buffer and considered sent. There is no guarantee that the server has successfully received the record in this case, and the retry configuration will not take effect (as the client will not generally know of any failures). The offset given back for each record will always be set to -1.
  - acks=1: This means that the leader will write the record to its local log but will respond without awaiting full acknowledgement from all followers. In this case, if the leader fails immediately after acknowledging the record but before the followers have replicated it, then the record will be lost.
  - acks=all: This means that the leader will wait for the full set of in-sync replicas to acknowledge the record. This guarantees that the record will not be lost as long as at least one in-sync replica remains alive. This is the strongest available guarantee.

# Contents

1. Introduction
2. Architecture and Functions
3. Data Management
  - Data Storage Reliability
  - Data Transmission Reliability
  - Old Data Processing Methods

# Old Data Processing Methods

- In Kafka, each partition of a topic is sub-divided into segments, making it easier for periodical clearing or deletion of consumed files to free up space.

```
-rw----- 1 omm wheel 10485760 Jun 13 13:44 00000000000000000000.index  
-rw----- 1 omm wheel 1081187 Jun 13 13:45 00000000000000000000.log
```

- For traditional message queues, messages that have been consumed are deleted. However, **the Kafka cluster retains all messages regardless of whether they have been consumed**. Due to disk restrictions, it is impossible to permanently retain all data (actually unnecessary). Therefore, Kafka needs to process old data.
- Configure the Kafka server properties file:

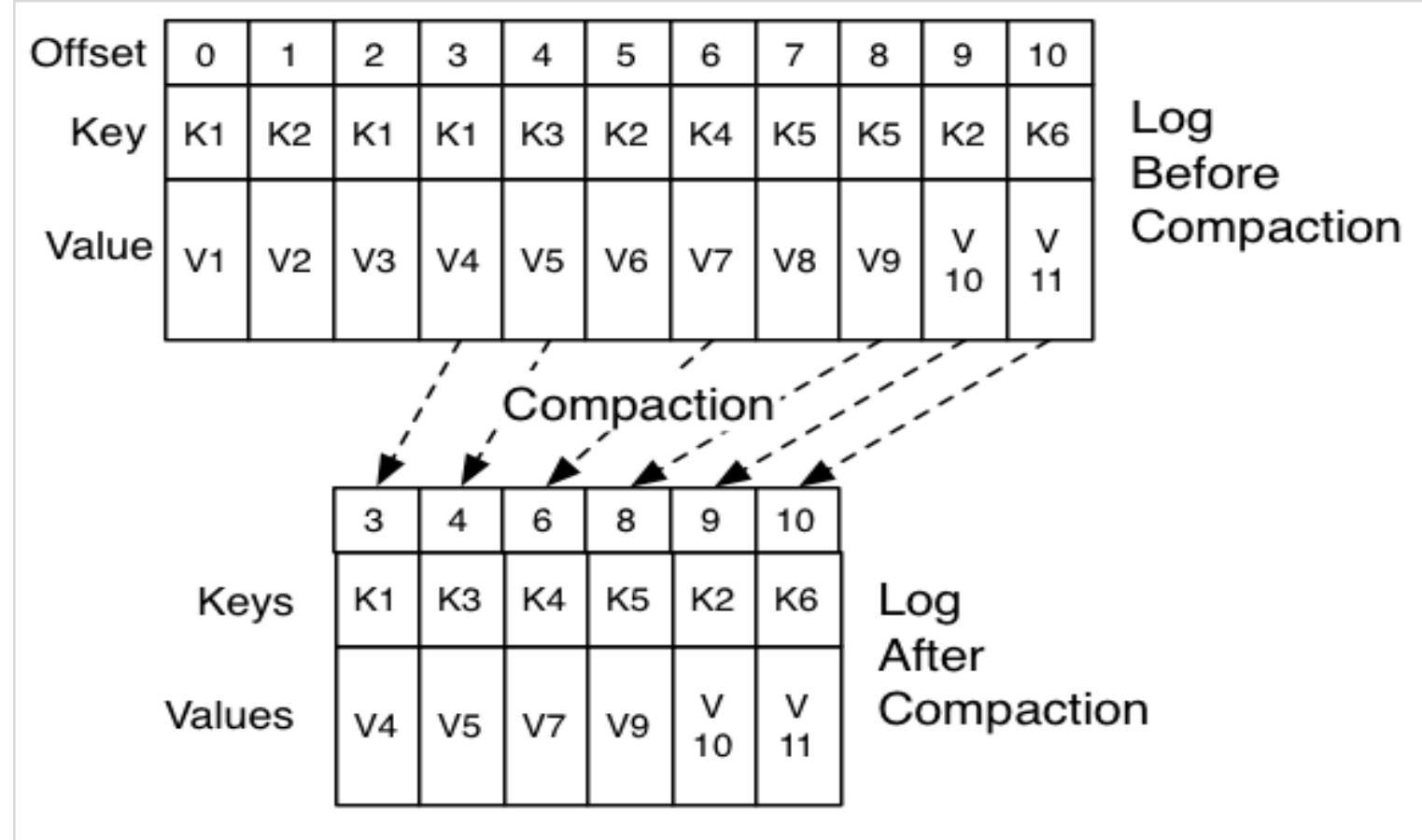
```
$KAFKA_HOME/config/server.properties
```

# Kafka Log Cleanup

- Log cleanup policies: **delete** and **compact**.
- Threshold for deleting logs: retention time limit and size of all logs in a partition.

Configuration	Default Value	Description	Range
log.cleanup.policy	delete	Log segments will be deleted when they reach the time limit (beyond the retention time). This can take either the value <b>delete</b> or <b>compact</b> .	delete or compact
log.retention.hours	168	Maximum period to keep a log segment before it is deleted. Unit: hour	1 - 2147483647
log.retention.bytes	-1	Maximum size of log data in a partition. By default, the value is not restricted. Unit: byte.	-1 - 9223372036854775807

# Kafka Log Compact



# Summary

- This chapter describes the basic concepts of the message system, and Kafka application scenarios, system architecture, and data management.

# Quiz

1. Which of the following are characteristics of Kafka?( )
  - A. High throughput
  - B. Distributed
  - C. Message persistence
  - D. Random message reading
2. Which of the following components does the Kafka cluster depend on during its running?( )
  - A. HDFS
  - B. Zookeeper
  - C. HBase
  - D. Spark

# Recommendations

---

- Huawei Cloud Official Web Link:
  - <https://www.huaweicloud.com/intl/en-us/>
- Huawei MRS Documentation:
  - <https://www.huaweicloud.com/intl/en-us/product/mrs.html>
- Huawei TALENT ONLINE:
  - <https://e.huawei.com/en/talent/#/>

# Thank you.

把数字世界带入每个人、每个家庭、  
每个组织，构建万物互联的智能世界。  
Bring digital to every person, home, and  
organization for a fully connected,  
intelligent world.

Copyright©2020 Huawei Technologies Co., Ltd.  
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.



# Revision Record

Do Not Print this Page

Course Code	Product	Product Version	Course Version
H13-711	MRS		V3.0

Author/ID	Date	Reviewer/ID	New/ Update
Huang Haoyang/hwx690472	2020.03.15	Wang Mengde/wwx711842	Update
Huang Haoyang/hwx690472	2020.09.04	Yan Hua/ywx416015	New

# Chapter 11 LDAP and Kerberos



# Foreword

---

- The in-depth development of big data open-source technologies cannot be achieved without the support of underlying platform technologies such as Hadoop. Big data, featuring vastness, diversity, timeliness, and accuracy, requires a basic platform as a support with stable performance and information security. To manage the access control permission of data and resources in the cluster, Huawei big data platform implements a highly reliable cluster security mode based on LDAP and Kerberos and provides an integrated security authentication.

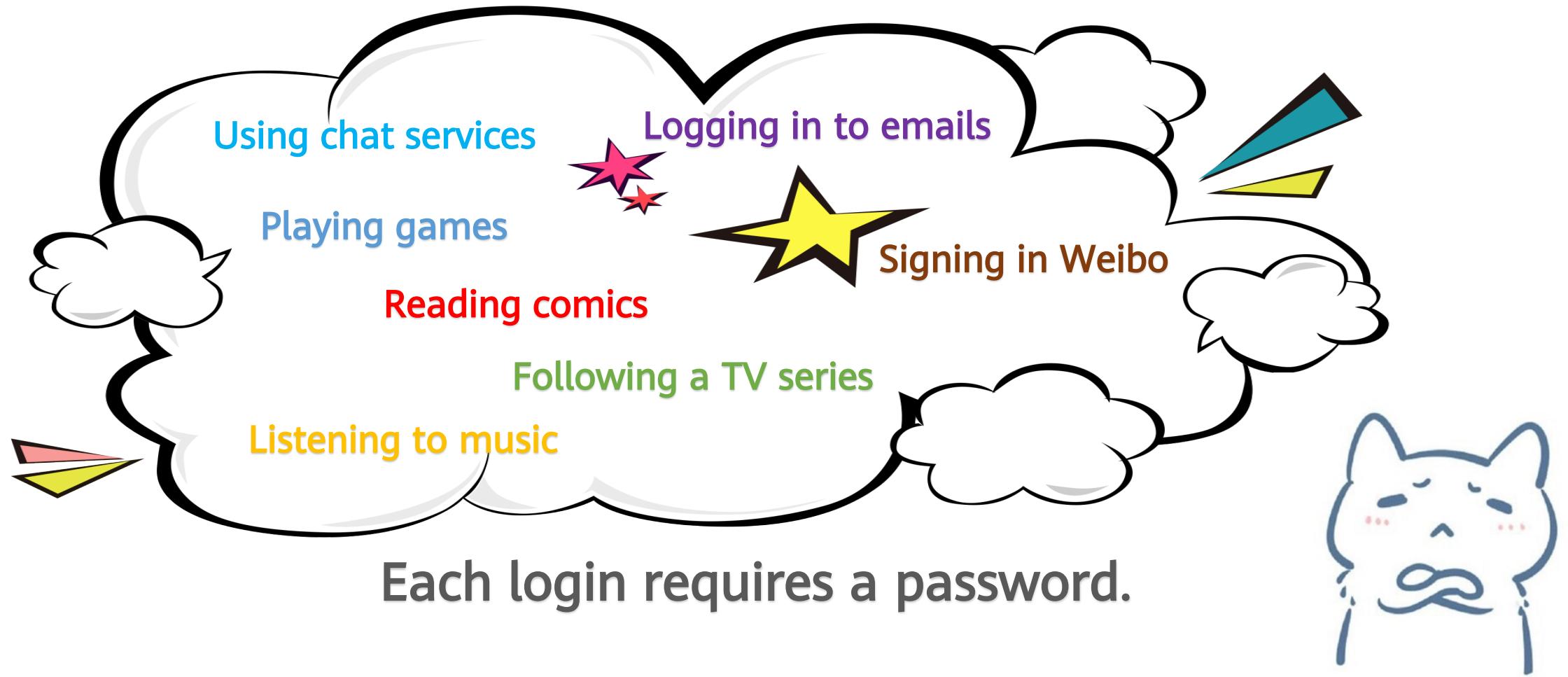
# Objectives

- On completion of this course, you will be able to know:
  - Basic principles of Identity and Access Management (IAM)
  - Basic principles of the directory service and LDAP
  - Basic principles of SSO (Single Sign-On) and Kerberos
  - Scenario architecture of Huawei big data security authentication

# Contents

- 1. IAM (Identity and Access Management)**
2. Directory Services and Basic Principles of LDAP
3. SSO and Basic Principles of Kerberos
4. Scenario Architecture of Huawei Big Data Security Authentication

# Too Many Passwords to Remember



# IAM (Identity and Access Management)

- IAM is similar to the passing rules of the amusement park, where guests can use a pass (key) to play authorized programs.
- On the open-source big data platform, users may need to use many open-source components at the same time. Therefore, identity authentication and access permission of each component are involved. The unified authentication service can better manage user identity authentication and session.

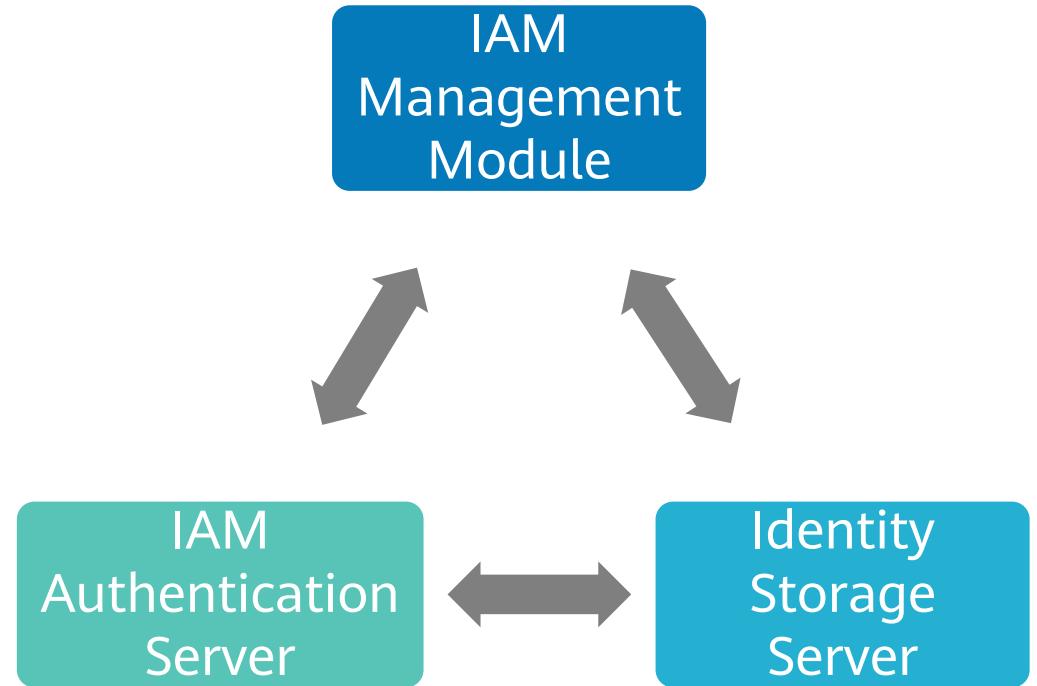


# IAM System

- On the big data platform, IAM systems manage users, roles, and organizations of various open source component application systems in a unified manner, implementing cross-domain single sign-on (SSO) and unified identity authentication between application systems. IAM systems have the following features:
  - User management
  - User authentication
  - SSO
  - Hierarchical management
  - Permission management
  - Session management
  - Good OS compatibility

# Structure of IAM Systems

- Most IAM systems used by enterprises consist of an IAM management module, IAM authentication server, and identity storage server.
- Huawei big data solutions implement permission management through WebUI to control access to data and resources in clusters. The structure is based on OpenLDAP for managing and storing identity authentication, and the Kerberos technologies for unified identity authentication.

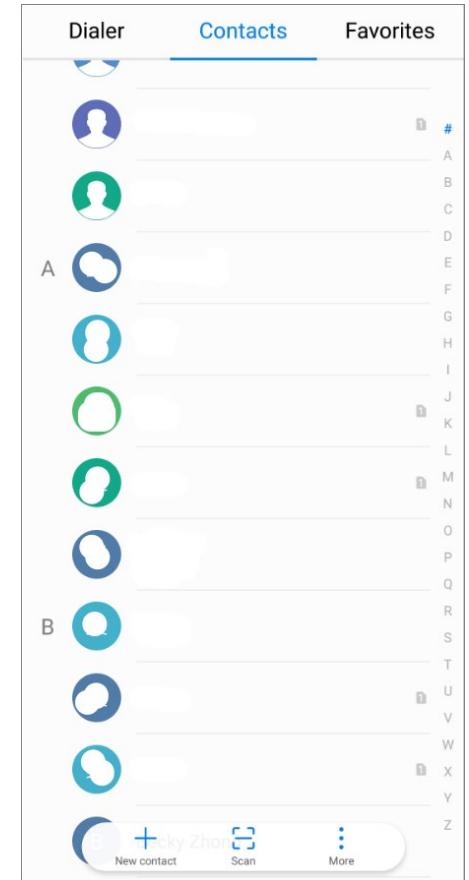


# Contents

1. IAM (Identity and Access Management)
- 2. Directory Services and Basic Principles of LDAP**
3. SSO and Basic Principles of Kerberos
4. Scenario Architecture of Huawei Big Data Security Authentication

# Directory Services

- Directory service is an optimized service for querying, browsing, and searching. Similar to a file system, a directory service stores and traverses data in a tree-like structure.
- The most common directory service in daily life is the address book on the mobile phone. The address book consists of names, addresses, and phone numbers in alphabetical order, helping users quickly locate contacts and their information.
- Q: What is the relationship between LDAP and directory services?



# LDAP Overview

- LDAP stands for Lightweight Directory Access Protocol. It is a protocol for implementing centralized account management architecture based on X.500 protocols.
- LDAP has the following characteristics:
  - LDAP runs over TCP/IP or other connection-oriented transfer services.
  - LDAP is an IETF (Internet Engineering Task Force) standard track protocol and is specified in *RFC 4510 on Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map*.

# LDAP Server: Directory Service System

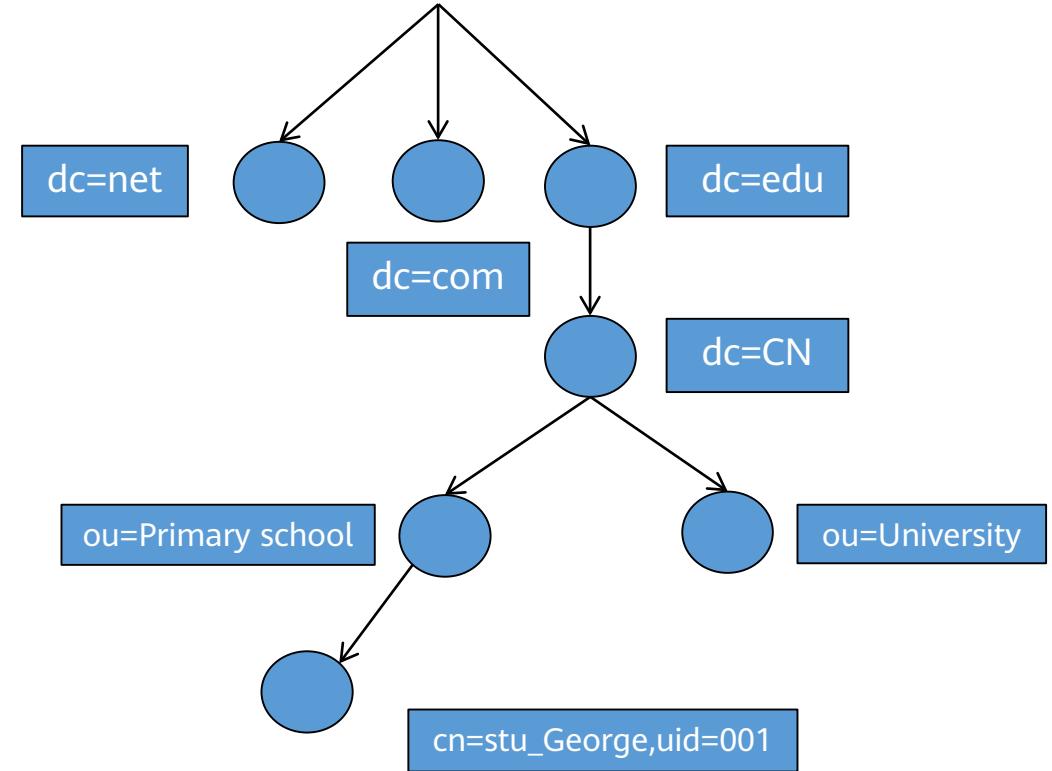
- For Huawei big data platform, LDAP server functions as a directory service system to implement centralized account management.
- As a directory service system, LDAP server consists of a directory database and a set of access protocols.
  - LDAP server is based on the open source OpenLDAP.
  - LDAP server uses Berkeley DB as the default back-end database.
  - LDAP server is an open source implementation of the LDAP protocol.

# LDAP Server Organizational Model

- The LDAP server organizational model is also known as the LDAP server directory tree. It has the following features:
  - Information in the LDAP server directory is organized and stored in a tree structure.
  - Each node in the LDAP server directory tree is called an entry and has a unique distinguished name (dn).
  - The root of the LDAP server directory tree defines the domain component (dc).

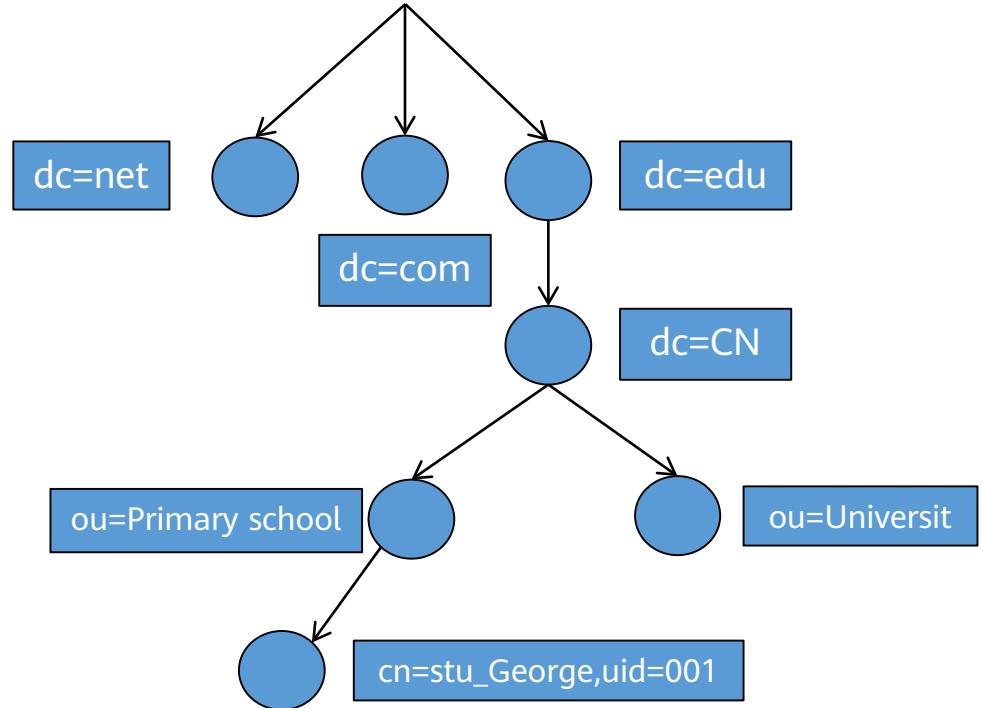
# LDAP Server Organizational Model

- LDAP server organizational model is also called LDAP server directory tree or LDAP server naming model. It has the following features:
  - Information in the LDAP server directory is organized and stored in a tree structure.
  - Each node in the LDAP server directory tree is called an entry and has a unique distinguished name (dn).
  - The root of the LDAP server directory tree usually defines the domain component (dc).



# LDAP Server Organizational Model (Cont.)

- LDAP server organizational model is also called LDAP server directory tree. It has the following features:
  - Organization unit (ou) can be defined under a domain name. An OU can contain groups of objects.  
In the directory service system, An OU contains information about all user accounts within the organization unit.
  - An OU contains objects. You can specify an object with Common Name (cn) and user ID (uid).
- For example, the dn of the lower left node is:  
`cn=stu_George,uid=001,ou=Primary school,dc=CN,dc=edu`



# LDAP Server: Tree-Structure Database

- The LDAP server directory tree is a tree-structure database and applies to scenarios where data is written once and queried for multiple times. In a traditional relational database, data is recorded in a table. In a directory tree, data is stored in a node. The tree structure can better correspond to the table storage mode. The directory tree has the following features:
  - The domain component (dc) is similar to a database in a relational database.
  - The organization unit (ou) is similar to the set of tables in the database.
  - The user ID (uid) is similar to the primary key in the table.
  - The common name (cn) is similar to the name of the unit data in the table.

# LDAP Server Functional Model

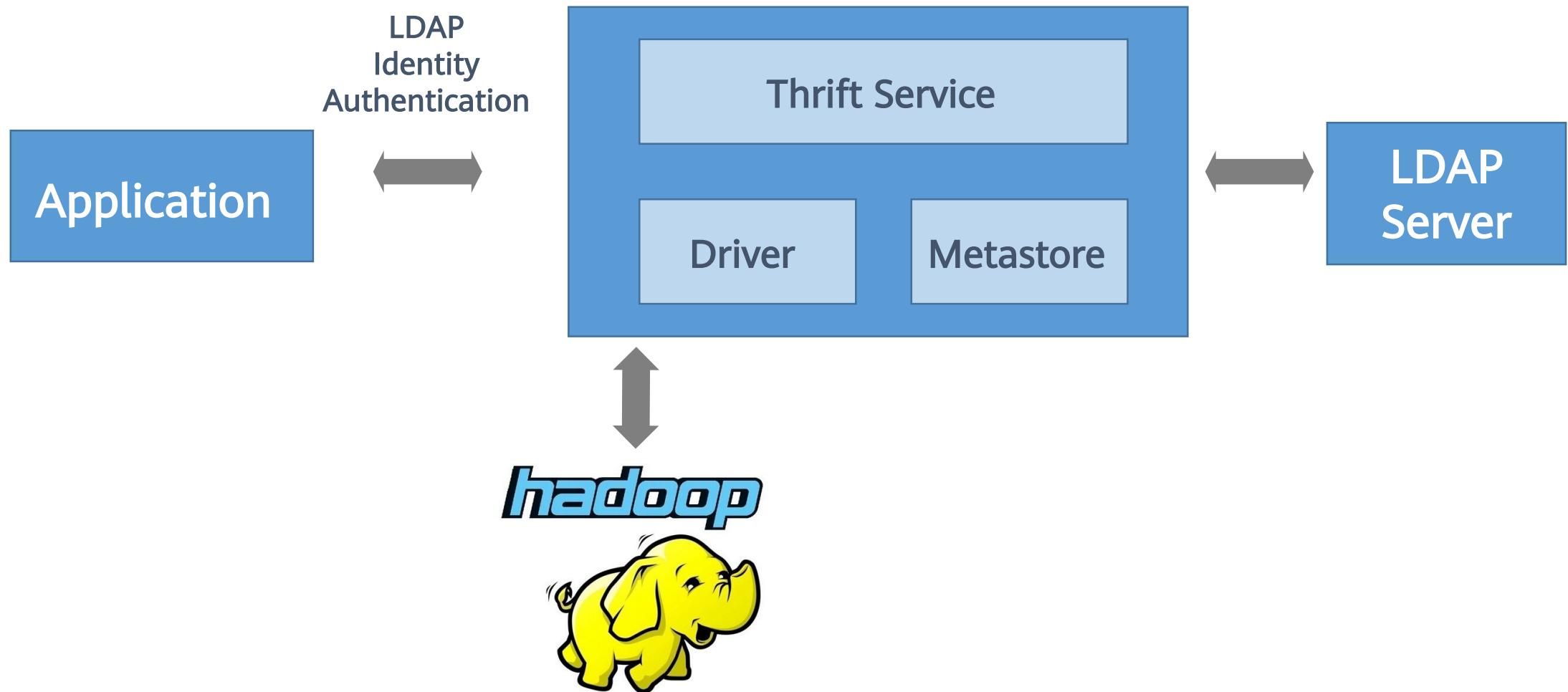
- LDAP servers on Huawei big data platform support 10 types of operations, which can be classified into the following four categories:

No.	Category	Operation
1	Query	search and compare information
2	Update	Add, delete, modify entries, and modify the dn of entries
3	Authentication	Bind and unbind authentication information
4	Others	Abandon and extend services

# LDAP Server Integration Design

- For platform integration design, LDAP and Hive are used as examples. The following three aspects are involved in the architecture design of access authentication:
  - Identity authentication architecture design
  - Identity authentication function design
  - Identity authentication process design

# Identity Authentication Architecture Design



# Identity Authentication Function Design

- LDAP server manages users by using groups and roles to better manage the attributes and permissions of users in different organizations.
- LDAP server groups are used to manage users in a unified manner. If a user is added to a group, the user's dn is added to the member attribute of the group.
- LDAP server roles are used to assign permissions to users. Each user entity has the nsrole attribute, which is used to add a role to the user.

# Identity Authentication Process Design

① **Create** a Hive super administrator.

② **Use** LDAP server to integrate users, user groups and roles.

③ **Hive super administrator** authorizes users in the LDAP server.

④ **The client** can connect to approved services after passing the authentication.

# LDAP Server Advantages

- The directory service system based on LDAP servers provides centralized user account management. It has the following advantages:
  - When facing a large, fast-growing number of users, LDAP server makes it easier for you to manage user accounts, specifically, to create accounts, reclaim accounts, manage permissions, and audit security.
  - LDAP server makes it secure to access different types of systems and databases across multiple layers. All account-related management policies are configured on the server, implementing centralized account maintenance and management.
  - LDAP server fully inherits and utilizes the identity authentication functions of existing account management systems on the platform. It separates account management from access control, thus improves access authentication security of the big data platform.

# Contents

1. IAM (Identity Access and Management)
2. Directory Services and Basic Principles of LDAP
- 3. SSO and Basic Principles of Kerberos**
4. Scenario Architecture of Huawei Big Data Security Authentication

# Single Sign-On (SSO)

- SSO is a part of identity management of the big data platform. With SSO, you only need to log in to one component with credentials, and then you can switch freely between components on the platform.
- SSO has the following features:
  - Delivers convenient services.
  - Improves O&M efficiency.
  - Simplifies application system development.

# Mainstream SSO Approaches

- SSO can be implemented via the following approaches:

Cookies-based approach

Broker-based approach

Agent-based approach

Agent and broker-based approach

Gateway-based approach

Token-based approach

- On Huawei big data platform, Kerberos is used as a trusted third-party authentication service (broker-based). It is a traditional password-based authentication service to implement SSO.

# Introduction to Kerberos

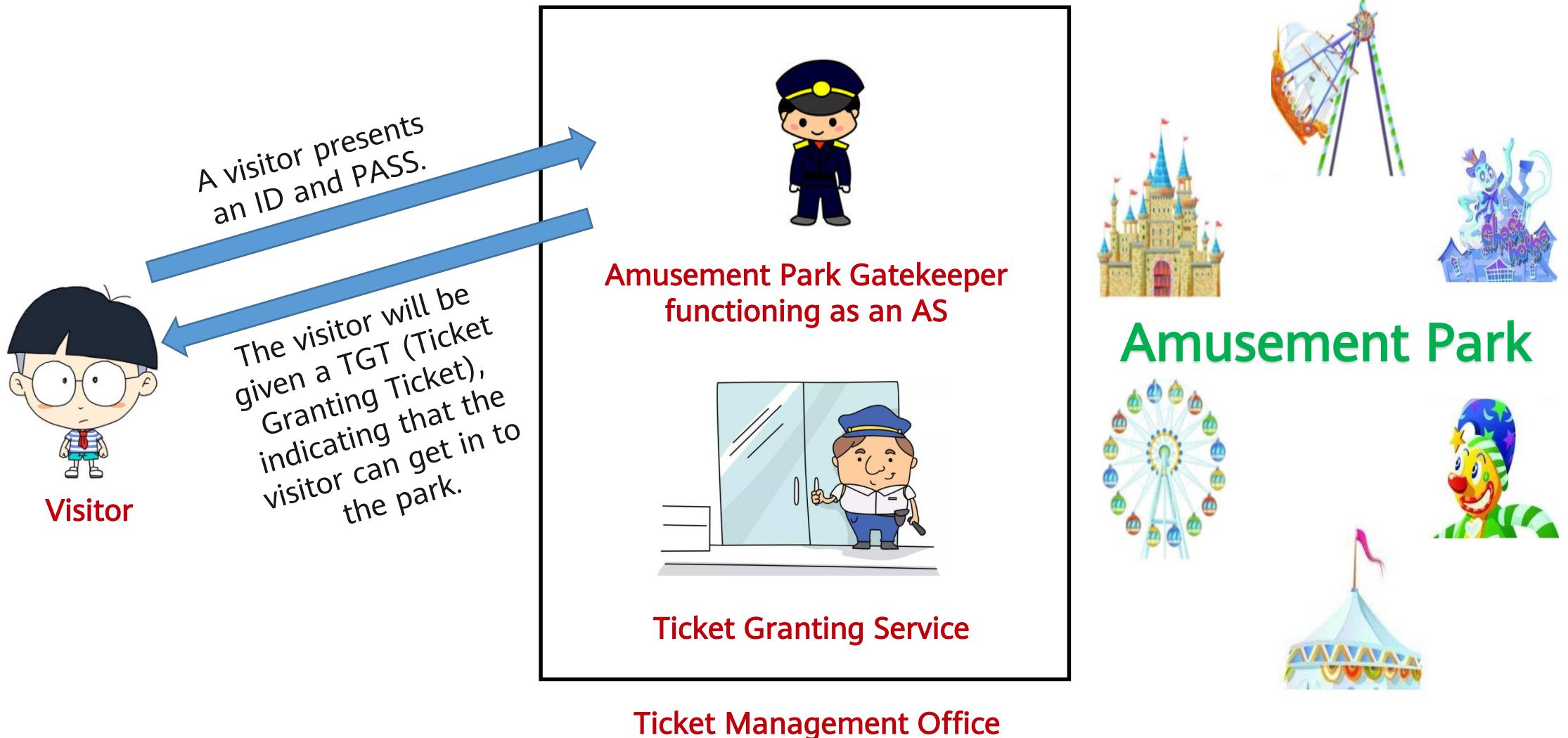
- Kerberos, named after the ferocious three-headed guard dog of Hades from Greek mythology, is now known as an authentication concept. The Kerberos protocol adopts a client-server model and cryptographic algorithms such as Data Encryption Standard (DES) and Advanced Encryption Standard (AES). It provides mutual authentication, so the client and server can verify each other's identity.
- Huawei big data platform uses KrbServer to provide Kerberos functions for all components. Kerberos can be used to prevent eavesdropping and replay attacks, and protect data integrity. It is a key management system that leverages an asymmetric key mechanism.



# Three Core Elements of KrbServer

- KrbServer authentication mechanism uses a ticket (key) that can prove the user identity to protect the cluster from eavesdropping and replay attacks. The three core elements of KrbServer include:
  - Kerberos Client
  - Kerberos KDC Server
  - Key Distribution Center (KDC)

# KrbServer and Amusement Park



# KrbServer and Amusement Park



Visitor

"Excuse me, I want to play the sky wheel."



Sky wheel administrator  
Functioning as the client

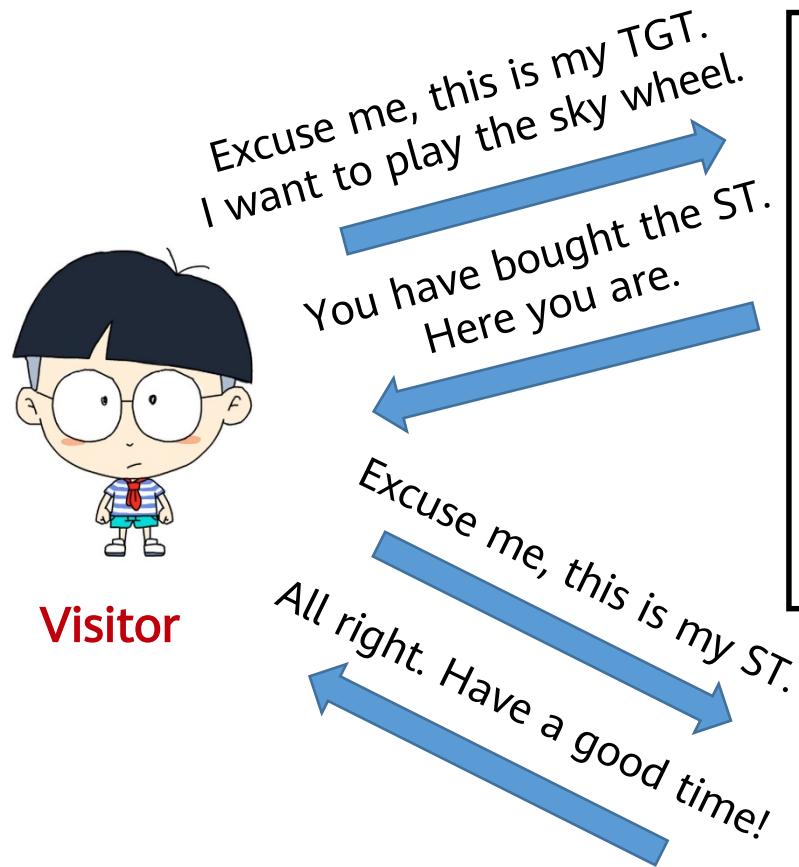
"Can I have your ST (service ticket) for the wheel?"

"Please swipe your TGT on the TGS machine."

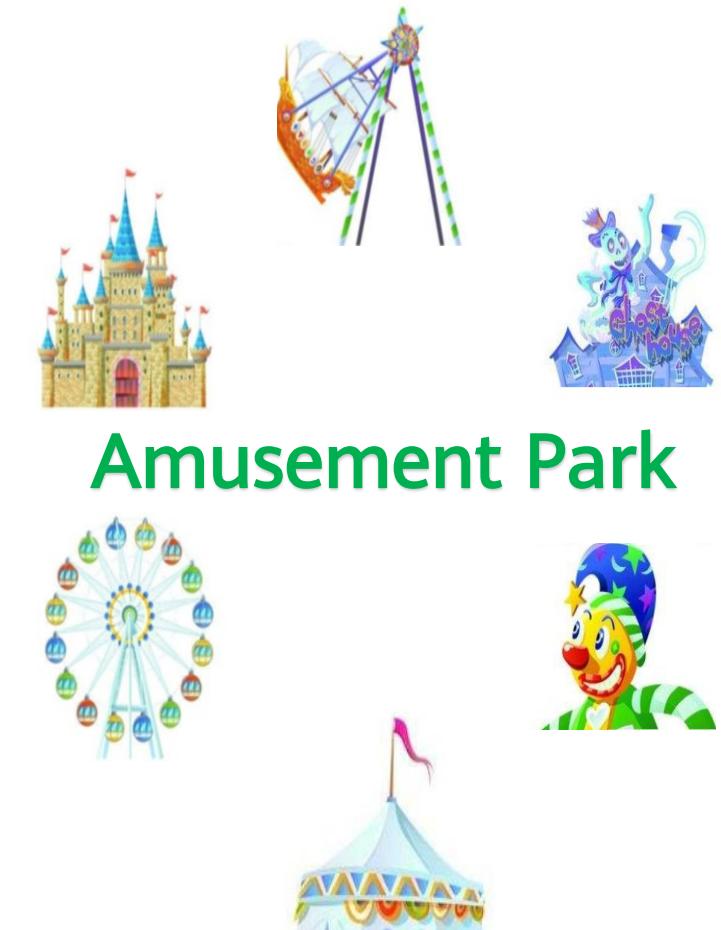
You can get the ST if you have purchased it from TGS.



# KrbServer and Amusement Park



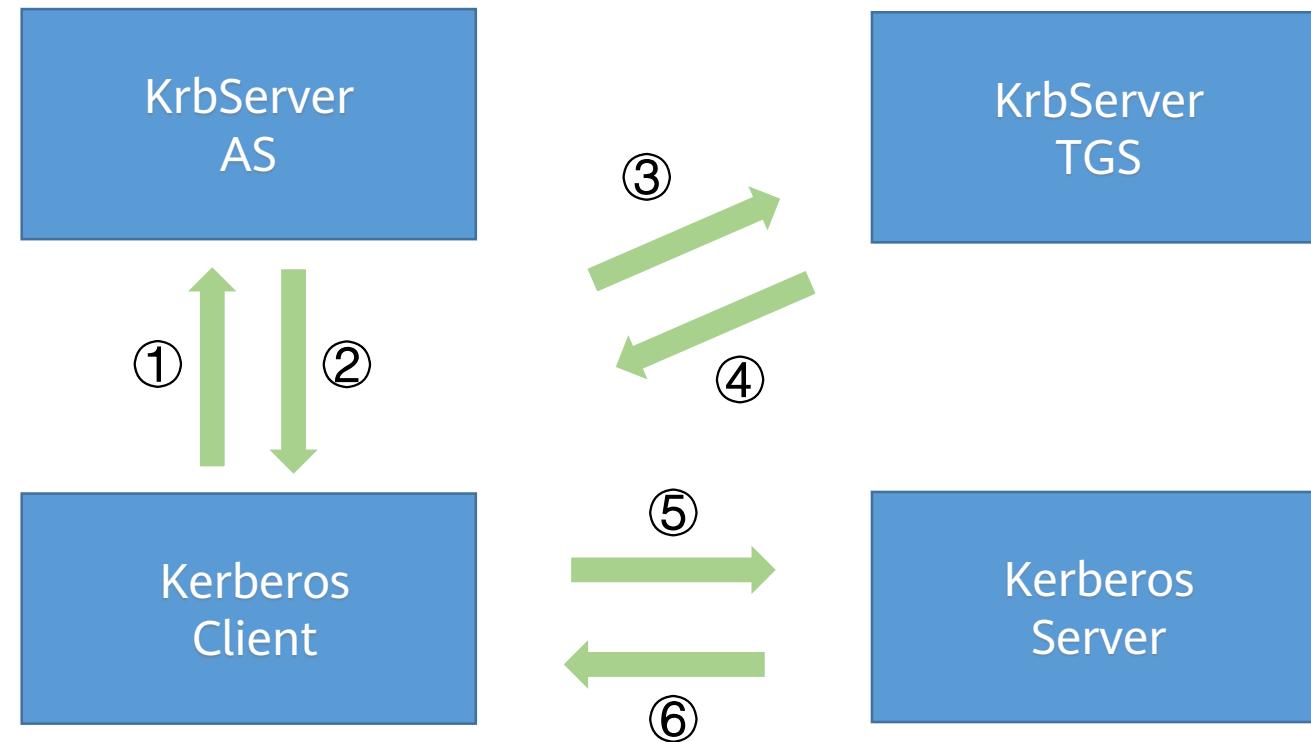
Sky wheel administrator  
functioning as the client



# KrbServer Concepts

Name	Function	Roles in the Amusement Park
Client	Authentication client in the Cluster	Sky wheel administrator
Server	Authentication server in the cluster	Sky wheel
KDC	Key distribution center	Ticket management office
AS	Authentication server	Gatekeeper
TGS	Ticket granting server	Ticket granting center
TGT	Ticket granting ticket	Amusement park key card
ST	Service ticket	Sky wheel ticket

# KrbServer Authentication Process



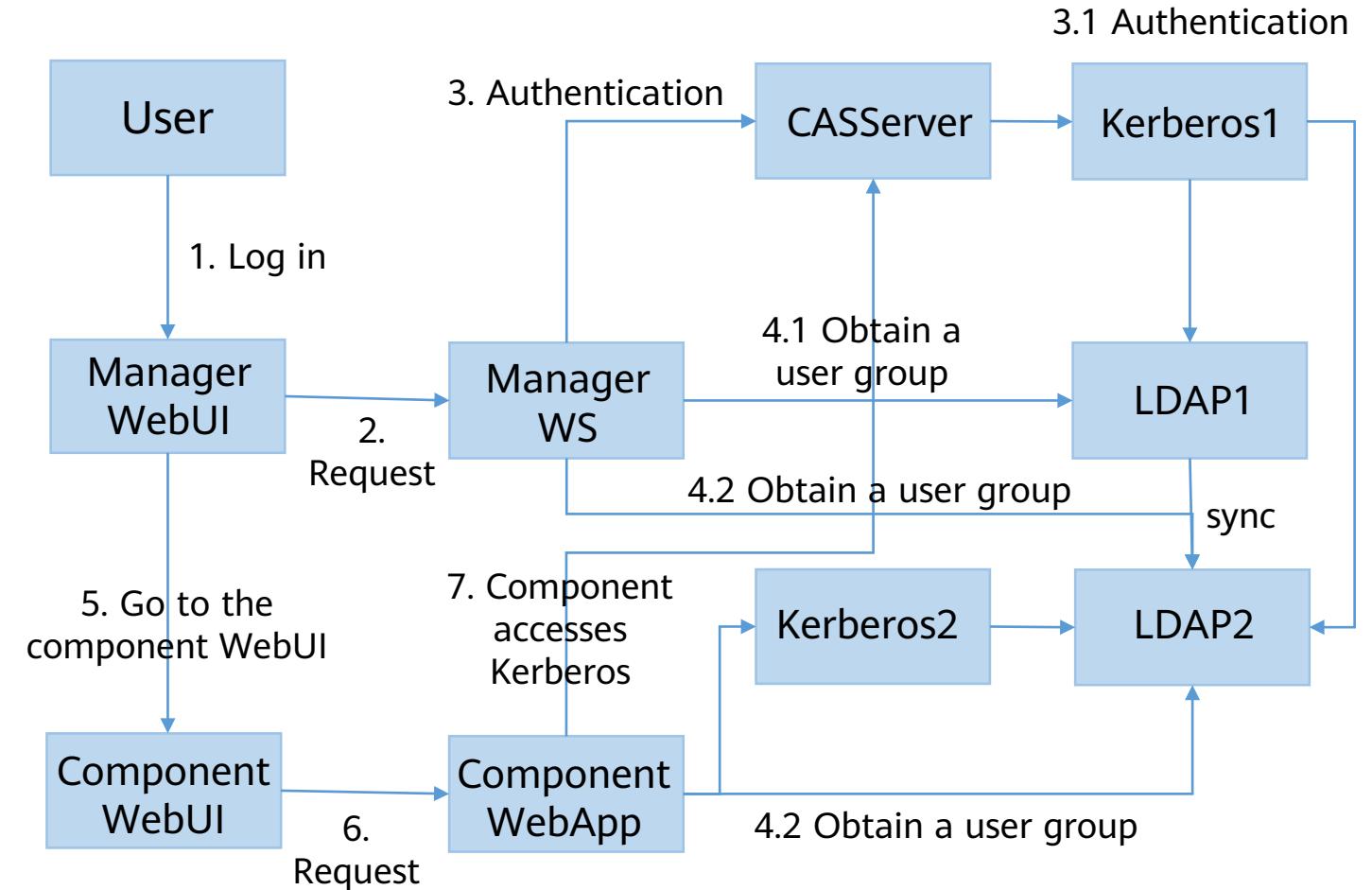
# Contents

---

1. IAM (Identity Access and Management)
2. Directory Services and Basic Principles of LDAP
3. SSO and Basic Principles of Kerberos
- 4. Scenario Architecture of Huawei Big Data Security Authentication**

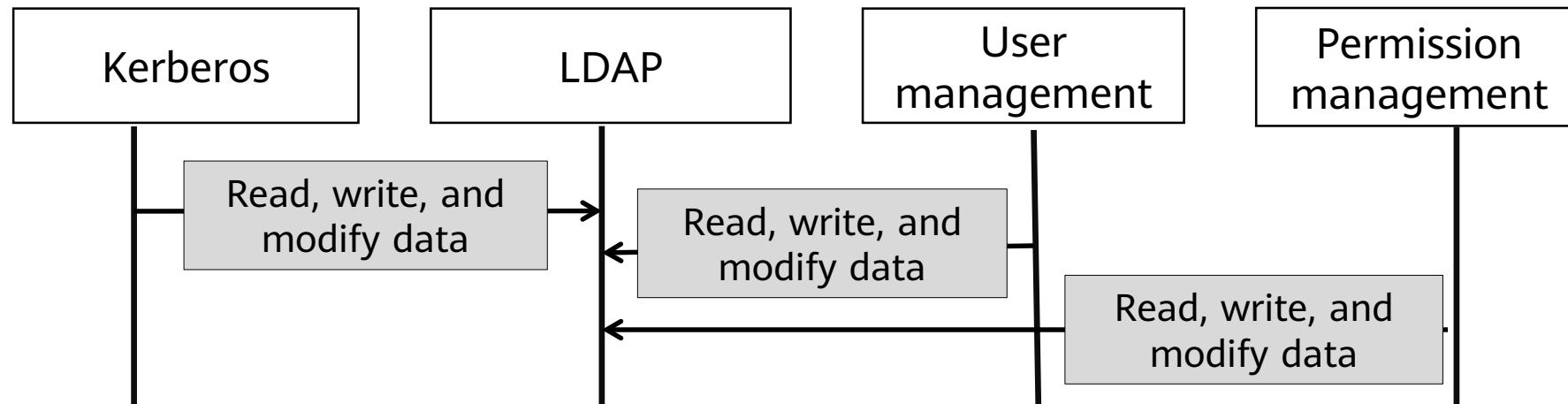
# Architecture of Huawei Big Data Security Authentication

- Step 1, 2, 3, 4:
  - Process of logging in to the Manager WebUI.
- Step 5, 6:
  - Process of logging in to the component UI.
- Step 7:
  - Access between components.
- Data operation mode of Kerberos1 in LDAP:
  - The active and standby instances of LDAP1 and LDAP2 are accessed in load sharing mode. Data can be written only to the active LDAP2 instance. Data can be read on LDAP1 or LDAP2.
- Data operation mode of Kerberos2 in LDAP:
  - Only the active and standby instances of LDAP2 can be accessed. Data can be written only to the active LDAP2 instance.



# Interaction between Kerberos and LDAP Services

- Kerberos serves as an authentication server center and provides unified authentication services for all services in the cluster and secondary development applications of customers.
- LDAP serves as a user data storage center and stores user information in the cluster, including passwords and supplementary information.
- During the unified authentication, all data of Kerberos, including user passwords and supplementary information (such as the user group information), needs to be obtained from LDAP.
- In each authentication, Kerberos needs to obtain user information from LDAP.



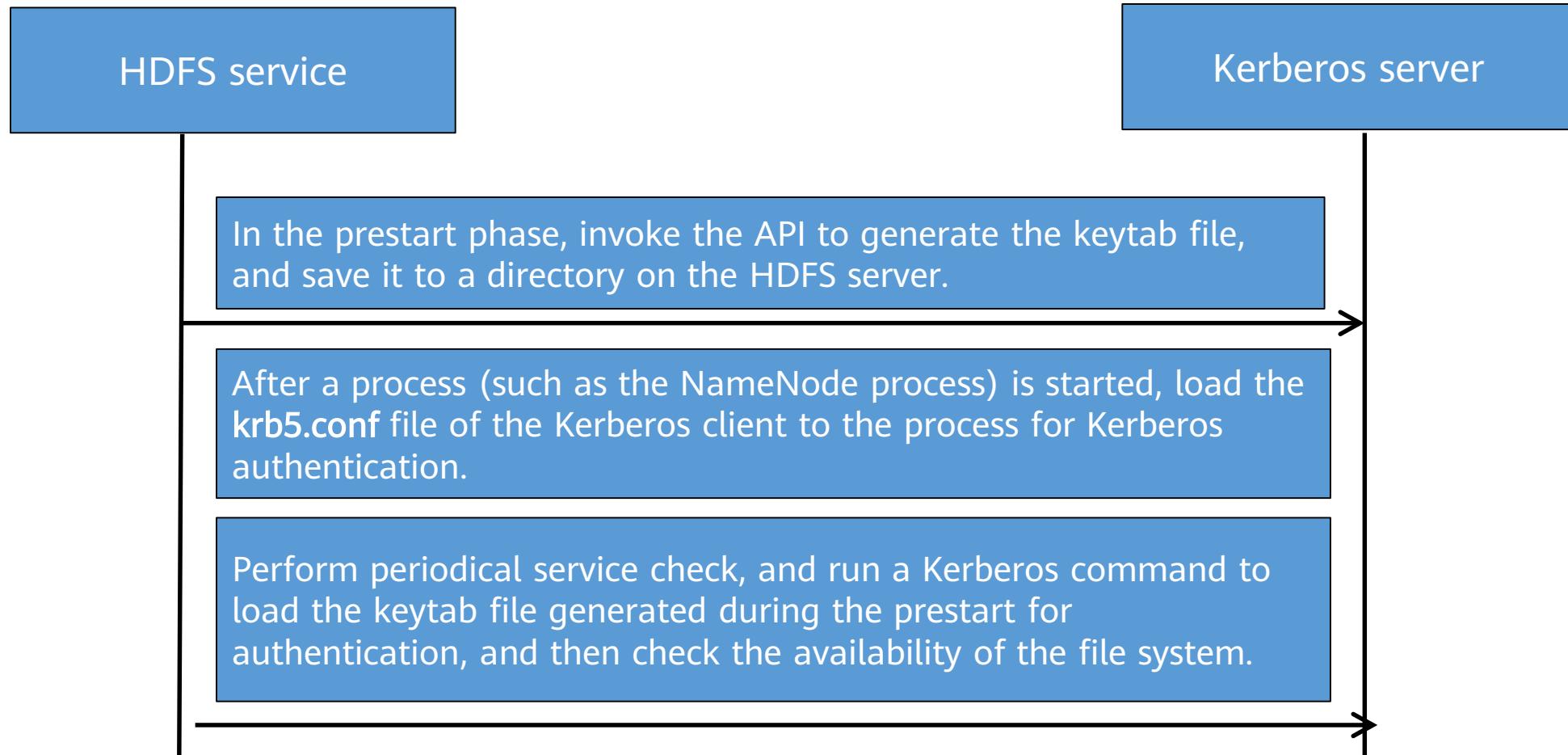
# User Storage

- LDAP provides data storage for users, including the default users (for example, admin users) in the cluster and the users created by a customer (for example, a user created on the UI).
- Two types of key user information are stored: the Kerberos information and the LDAP information.
  - Kerberos information includes usernames and passwords. Kerberos supports the authentication query, that is, password verification.
  - LDAP information refers to additional user information, such as the user type, user group, role, and email address. LDAP supports external authentication, that is, permission identification. For example, whether a user has the permission to access a file directory in the HDFS.

# Service Authentication in the Cluster

- Kerberos service is a basic component module in the cluster. In security mode, all service components depend on the Kerberos service (component). Service components (such as HDFS) must pass the Kerberos authentication when providing external services. If the Kerberos authentication fails, no application service of the service component can be obtained.
- As a Kerberos data storage module, LDAP server does not directly interact with other service components (except for the Kerberos service).
- When a service (such as HDFS) in the cluster is prestarted, the corresponding session key (keytab, which is mainly used for identity authentication of applications) is obtained from Kerberos in advance. When any other service (such as Yarn) needs to add, delete, modify, or query data in HDFS, the corresponding TGT and ST must be obtained for the security access.

# Service Authentication in the Cluster (Cont.)



# Common Role Deployment Mode

- Kerberos service role: Kerberos server and Kerberos admin.
  - Kerberos server supports external authentication, and Kerberos admin supports external user management (such as adding, deleting, and modifying users).
- Kerberos service is deployed in load sharing mode. During the installation, Kerberos service needs to be distributed to the two control nodes in the cluster.
- LDAP server service role: SLAPD server.
- LDAP server service is deployed in active/standby mode. During the installation, LDAP server service needs to be distributed to the two control nodes in the cluster.
- To achieve optimal performance, it is recommended that LDAP server and KrbServer in all clusters be deployed on the same node.

# Kerberos Strengths

- Prevents brute-force attacks. The session key used for authentication is a short term key, which is valid only in one session.
- Prevents replay attacks. Each request is marked with a timestamp.
- Supports mutual authentication. Kerberos supports mutual authentication, which is better than NTLM. Kerberos server returns the timestamp sent by the client to the client for identity verification.
- Provides reliable performance.
  - The KRB\_AS\_REP and KRB\_TGS\_REP messages show that KDC sends the identity information of the client, including the session key and encrypted client identity information (TGT and session ticket), to the client, which stores the information. In this way, the storage pressure of the server and KDC is reduced.
  - The client instead of KDC sends the authenticator and session ticket in a unified manner. This reduces the KDC pressure and prevents unsuccessful authentication caused by the failure of the client and KDC to reach the server simultaneously.

# Kerberos Weaknesses

- In the AS Exchange process, AS uses the master key of the client to encrypt the response to the request sent to the client. Therefore, the master key of the client is still used for encryption and transmission. The TGT granted by the AS to the client has a lifetime. When the lifetime ends, the client must resend the KRB\_AS\_REQ message to the AS. As a result, the master key is transmitted multiple times on the network.
- KDC needs to store a large number of keys, causing high cost of maintaining its account database.
- Kerberos uses timestamps to prevent replay attacks. However, in a distributed system, strict time synchronization is difficult, and incorrect time synchronization causes an authentication failure. This allows attackers to launch attacks by interfering with the time system.

# Parameter Description

Configuration Item	Description
KADMIN_PORT	Port provided by kadmin service for user management
Kdc_ports	Ports of the KDC service instances
Kdc_timeout	Timeout interval of the KDC authentication service
KPASSWD_PORT	Port provided by the kadmin service for password management
LDAP_OPTION_TIMEOUT	Timeout interval for the connection between Kerberos and the back-end LDAP database. If the connection time exceeds the timeout interval, a failure message is returned.
LDAP_SEARCH_TIMEOUT	Timeout interval for Kerberos to query the back-end LDAP database. If the query time exceeds the timeout interval, a failure message is returned.
max_retries	The maximum number of attempts made by the JDK process to connect to KDC for authentication. If the connection attempts exceed the max_retries value, a failure message is returned.

# Common Commands

Commands	Description
ldapsearch	Indicates the command-line tool of LDAP to search for the user information in LDAP.
ldapadd	Indicates the command-line tool of LDAP to add the user information to LDAP.
ldapdelete	Indicates the command-line tool of LDAP to remove entries from the LDAP.
kinit	Indicates the command-line tool of Kerberos to authenticate users. Only authenticated users can run the shell command of each MRS component to complete maintenance tasks.
kdestroy	Indicates the command-line tool of Kerberos to deregister users after tasks of components are completed.
kadmin	Indicates the command-line tool of Kerberos to switch to the Kerberos admin who can obtain and modify Kerberos user information.
kpasswd	Indicates the command-line tool of Kerberos to change the user password.
klist	Indicates the command-line tool of Kerberos to list authenticated users.

# Quiz

1. Which services (components) need to interact with Kerberos that serves as a basic component in security mode? In which processes are these services involved?
2. What are the differences between the following authentication methods: 1) running the kinit command on a client; 2) invoking the secondary development interface (for example, the login interface provided by Hadoop)?

# Summary

- This chapter introduces the security authentication system of Huawei's big data platform, including the basic authentication process (by explaining the protocol), how to productize the security authentication system (from the perspective of big data integration and deployment), and new features developed during the productization process.
- After learning this chapter, you can better understand LDAP, Kerberos, and the security authentication of MRS products so that you can better maintain the products.

# Recommendations

---

- Huawei Cloud Official Web Link:
  - <https://www.huaweicloud.com/intl/en-us/>
- Huawei MRS Documentation:
  - <https://www.huaweicloud.com/intl/en-us/product/mrs.html>
- Huawei TALENT ONLINE:
  - <https://e.huawei.com/en/talent/#/>

# Thank you.

把数字世界带入每个人、每个家庭、  
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and  
organization for a fully connected,  
intelligent world.

Copyright©2020 Huawei Technologies Co., Ltd.  
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.



# Revision Record

Do Not Print this Page

Course Code	Product	Product Version	Course Version
H13-711	MRS		V3.0

Author/ID	Date	Reviewer/ID	New/ Update
Zhang Xinqi/zwx701844	2020.03.12	Chen Xing/cwx525950	New
Zhang Xinqi/zwx701844	2020.08.26	Fu Zheng/fwxfw277898	New

# Chapter 12 Elasticsearch - Distributed Search Engine



# Foreword

---

- When we search for a movie or a book we like, a commodity on the e-commerce website, or a resume, or position on a recruitment website, we will use a search engine. In real life, Elasticsearch is often firstly brought up when talking about the search function during project development.
- In recent years, Elasticsearch has developed rapidly and surpassed its original role as a search engine. It has added the features of data aggregation analysis and visualization. If you need to locate desired content using keywords in millions of documents, Elasticsearch is the best choice.

# Objectives

- Upon completion of this course, you will be able to:
  - Know basic functions and concepts of Elasticsearch.
  - Master application scenarios of Elasticsearch.
  - Understand the system architecture of Elasticsearch.
  - Know the key features of Elasticsearch.

# Contents

---

- 1. Elasticsearch Overview**
2. Elasticsearch System Architecture
3. Elasticsearch Key Features

# Elasticsearch Overview

- Elasticsearch is a high-performance Lucene-based full-text search service. It is a distributed RESTful search and data analysis engine and can also be used as a NoSQL database.
  - Lucene extension
  - Seamless switchover between the prototype environment and production environment
  - Horizontal scaling
  - Support for structured and non-structured data



# Elasticsearch Features

## High performance

- The search results can be obtained immediately, and the inverted index for full-text search is implemented.

## Scalability

- Horizontal scaling is supported. Elasticsearch can run on hundreds or thousands of servers. The prototype environment and production environment can be seamlessly switched.

## Relevance

- Searches results are sorted based on elements (from word frequency or proximate cause to popularity).

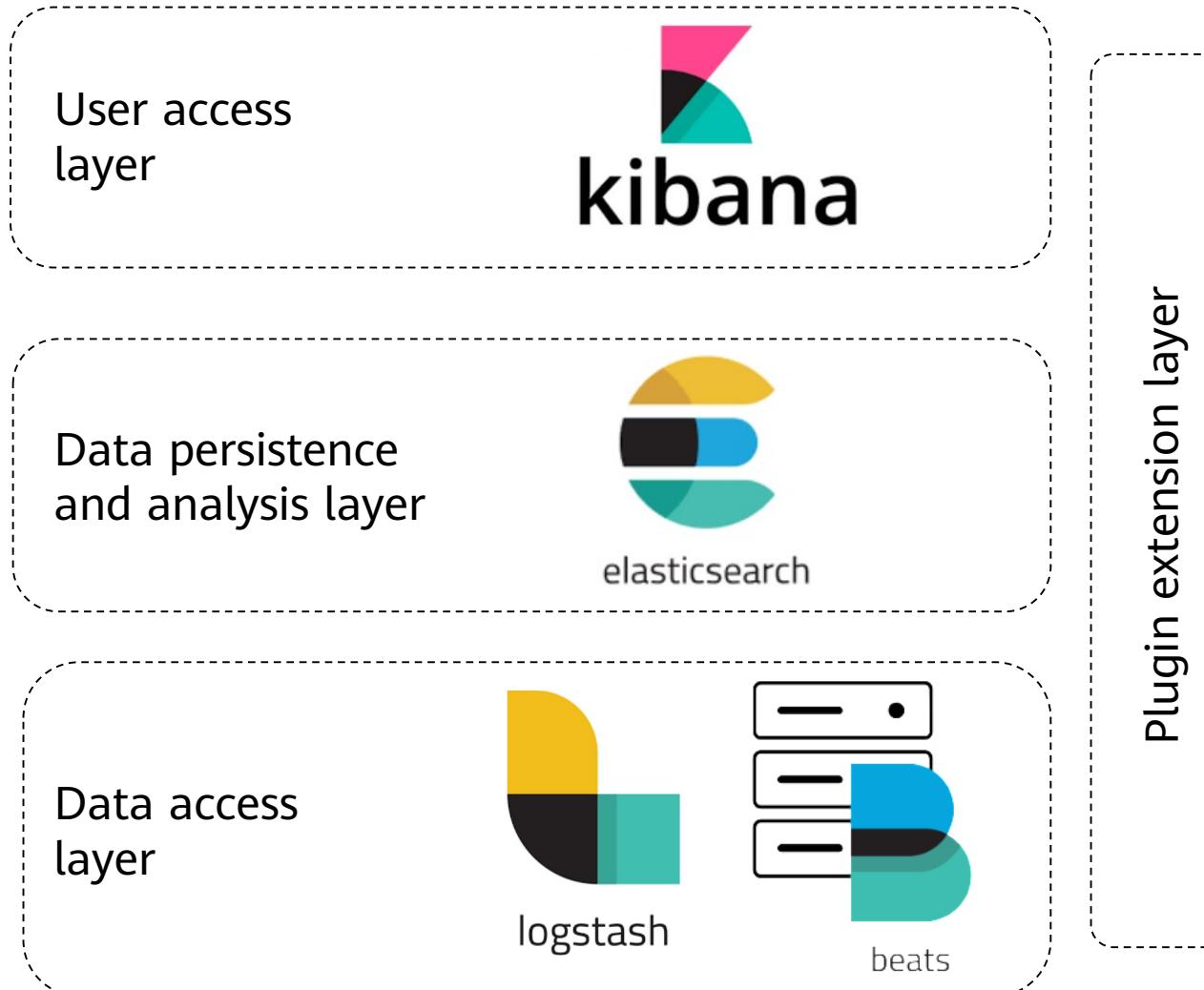
## Reliability

- Faults are automatically detected, hardware faults such as and network segmentation, ensuring the security and availability of your cluster (and data).

# Elasticsearch Application Scenarios

- Elasticsearch is used for log search and analysis, spatiotemporal search, time sequence search, and intelligent search.
  - Complex data types: Structured data, semi-structured data, and unstructured data need to be queried. Elasticsearch can perform a series of operations such as cleansing, word segmentation, and inverted index creation on the preceding data types, and then provide the full-text search capability.
  - Diversified search criteria: Full-text search criteria contain words or phrases.
  - Write and read: The written data can be searched in real time.

# Elasticsearch Ecosystem

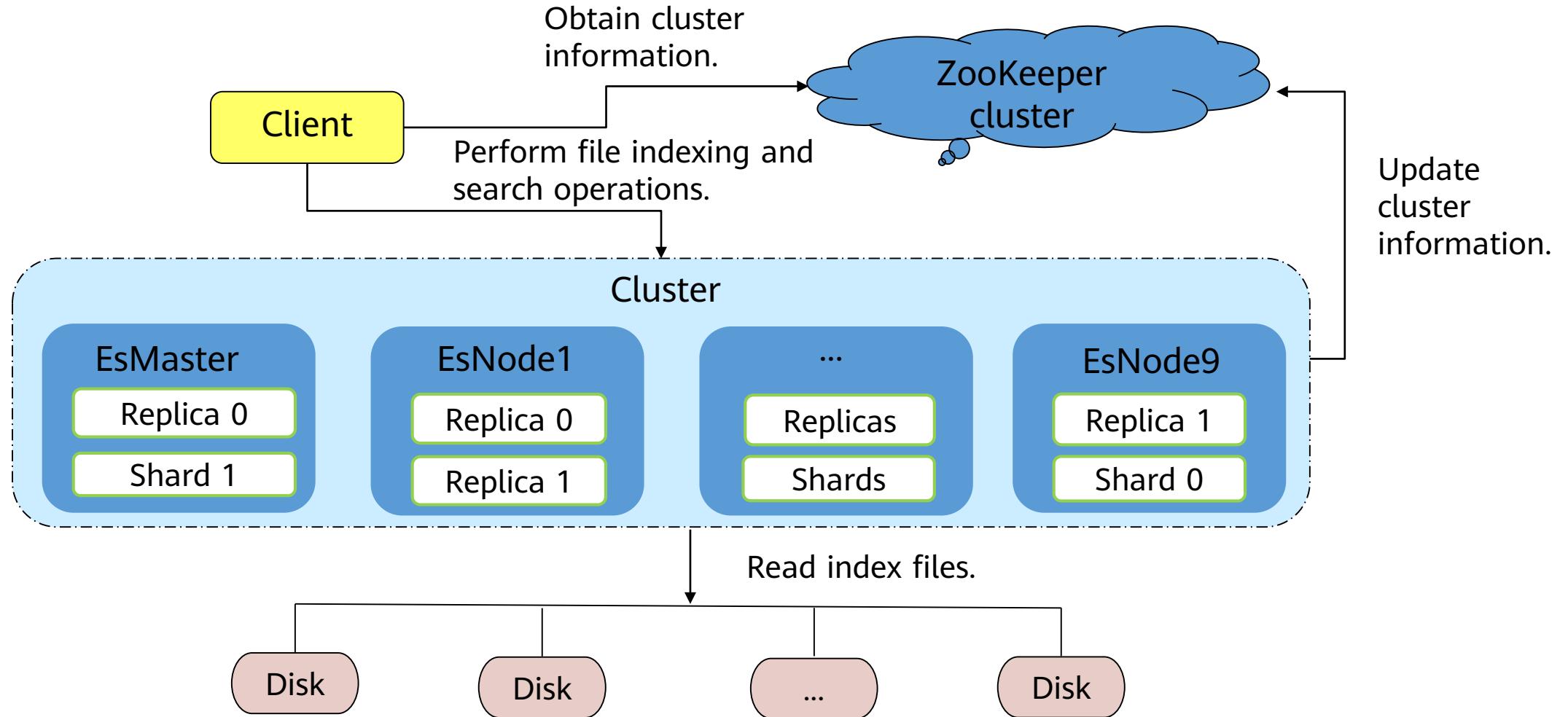


- ELK/ELKB provides a complete set of solutions. They are open-source software and work together to meet diverse requirements.

# Contents

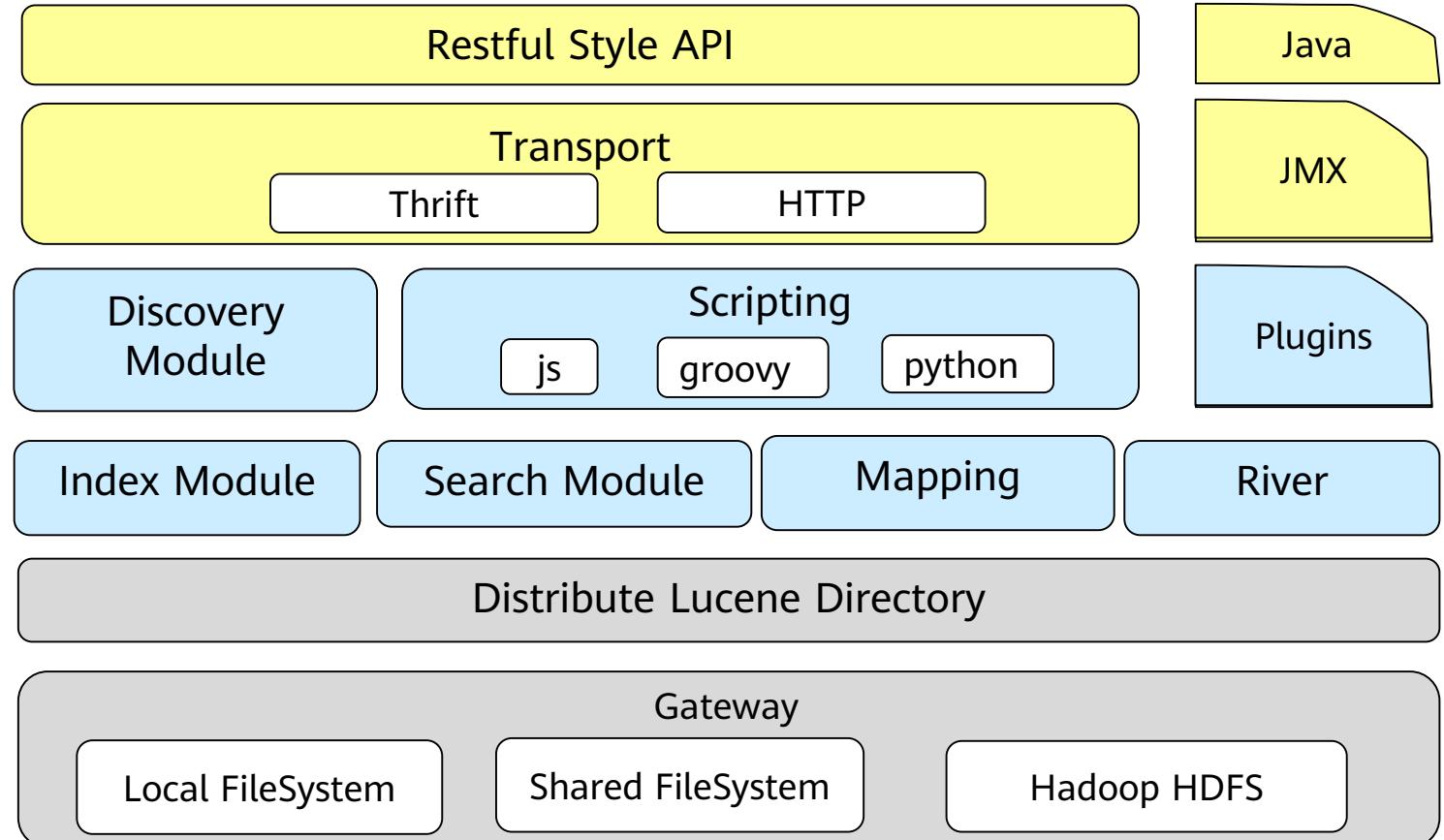
1. Introduction to Elasticsearch
- 2. Elasticsearch System Architecture**
3. Elasticsearch Key Features

# Elasticsearch System Architecture



# Elasticsearch Internal Architecture

- Elasticsearch provides RESTful APIs or APIs related to other languages (such as Java).
- The cluster discovery mechanism is used.
- Script languages are supported.
- The underlying layer is based on Lucene, ensuring absolute independence of Lucene.
- Indexes are stored in local files, shared files, and HDFS.



# Basic Concepts of Elasticsearch (1)

Index

A logical namespace in Elasticsearch

Type

Used to store different types of documents.  
It is deleted in Elasticsearch 7.

Document

A basic unit that can be indexed

Mapping

Used to restrict the field type

# Basic Concepts of Elasticsearch (2)

## Cluster

Cluster. Each cluster contains multiple nodes, one of which is the master node (the rest are slave nodes). The master node can be elected.

## EsNode

Elasticsearch node. A node is an Elasticsearch instance.

## EsMaster

Master node that temporarily manages cluster-level changes, such as creating or deleting indexes, and adding or removing nodes. A master node does not involve in document-level change or search. When the traffic increases, the master node does not affect the cluster performance.

## Shard

Index shard. Elasticsearch splits a complete index into multiple shards and distributes them on different nodes.

## Replica

Index replica. Elasticsearch allows you to set multiple replicas for an index. Replicas can improve the fault tolerance of the system. When a shard on a node is damaged or lost, the data can be recovered from the replica. In addition, replicas can improve the search efficiency of Elasticsearch by automatically balancing the load of search requests.

# Basic Concepts of Elasticsearch (3)

## Recovery

Data recovery or re-distribution. When a node is added to or deleted from the cluster, Elasticsearch redistributes shards based on the load of the node. When a failed node is restarted, data will be recovered.

## Gateway

Mode for storing Elasticsearch index snapshots. By default, Elasticsearch stores indexes in the memory and only makes them persistent on the local hard disk when the memory is full. The gateway stores index snapshots. When an Elasticsearch cluster is disabled and restarted, the cluster reads the index backup data from the gateway. Elasticsearch supports multiple types of gateways, including the default local file system, distributed file system, Hadoop HDFS, and Amazon S3.

## Transport

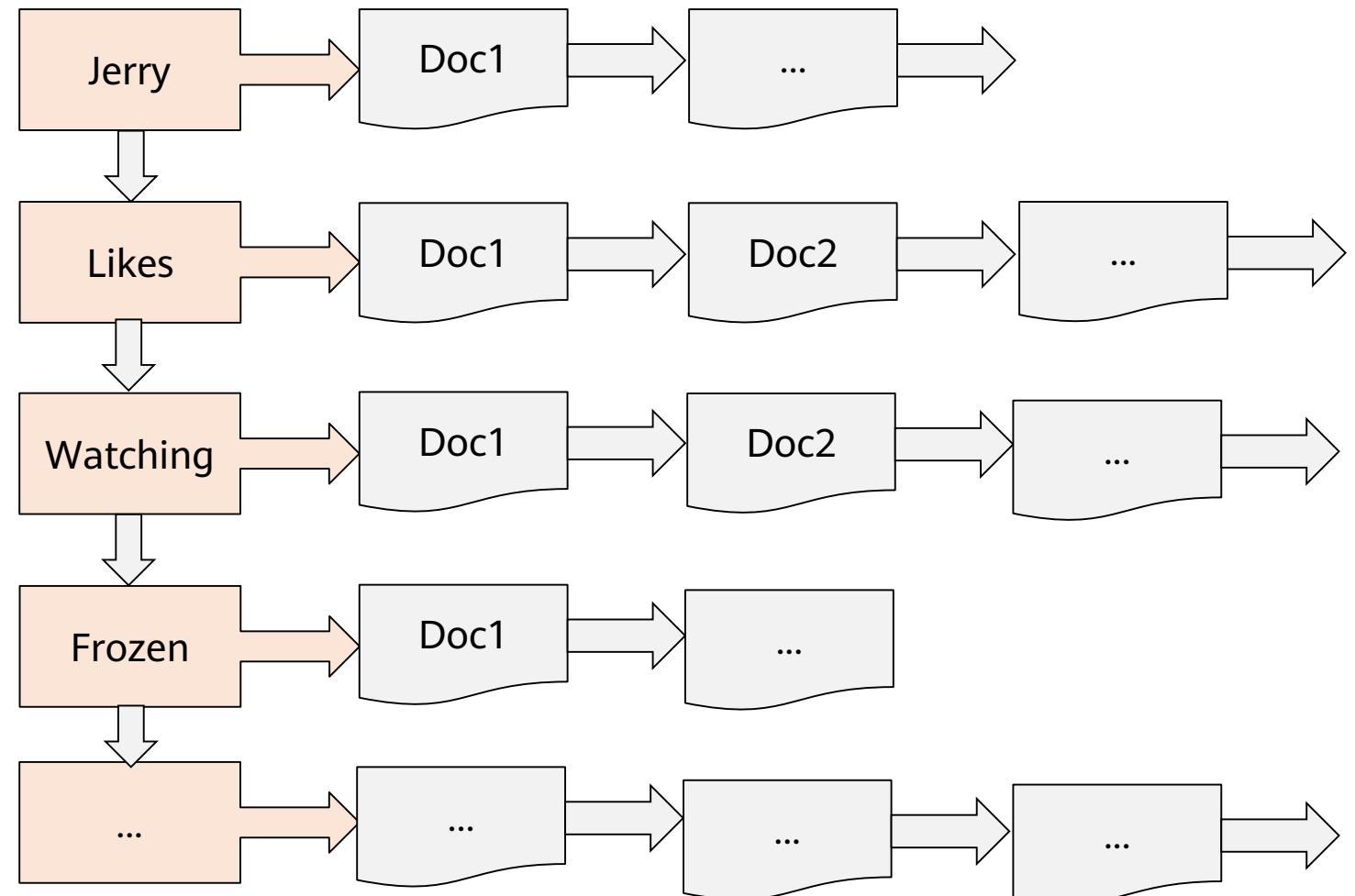
Interaction mode between an Elasticsearch internal node or cluster and the client. By default, internal nodes use the TCP protocol for interaction. In addition, such transmission protocols (integrated using plugins) as the HTTP (JSON format), Thrift, Servlet, Memcached, and ZeroMQ are also supported.

# Contents

1. Elasticsearch Overview
2. Elasticsearch System Architecture
- 3. Elasticsearch Key Features**

# ElasticSearch Inverted Index

- Forward index: Values are searched for based on keys. That is, specific information that meets the search criteria is located based on keys.
- Inverted index: It searches for the key based on the value. In the full-text search, a value is the keyword to be searched for. Corresponding documents are located based on the value.



# Elasticsearch Access APIs

- Elasticsearch can initiate RESTful requests to operate data. The request methods include GET, POST, PUT, DELETE, and HEAD, which allow you to add, delete, modify, and query documents and indexes.

1. View the cluster health status.

```
GET /_cat/health?v&pretty
```

2. Create a small index with only one primary shard and no replicas.

```
PUT /my_temp_index
{
  "settings": {
    "number_of_shards" : 1,
    "number_of_replicas" : 0
  }
}
```

3. Delete multiple indexes.

```
DELETE /index_one,index_two
```

4. Add documents by automatically generated IDs.

```
POST ip:9200/person/man
{
  "name":"111",
  "age":11
}
```

- For example, access Elasticsearch using the cURL client.

Obtain the cluster health status.

```
curl -XGET "http://ip:port/_cluster/health?pretty"
```

# Elasticsearch Routing Algorithm

- Elasticsearch provides two routing algorithms:
  - Default route: `shard=hash (routing)%number_of_primary_shards`. In this routing policy, the number of received shards is limited. During capacity expansion, the number of shards needs to be multiplied (Elasticsearch 6.x). In addition, when creating an index, you need to specify the capacity to be expanded in the future. Elasticsearch 5.x does not support capacity expansion. Elasticsearch 7.x supports expansion freely.
  - Custom route: In this routing mode, the routing can be specified to determine the shard to which a document is written, or search for a specified shard.

# Elasticsearch Balancing Algorithm

- Elasticsearch provides the automatic balancing function.
- Application scenarios: capacity expansion, capacity reduction, and data import
- The algorithms are as follows:
  - $\text{weight\_index}(\text{node}, \text{index}) = \text{indexBalance} * (\text{node.numShards}(\text{index}) - \text{avgShardsPerNode}(\text{index}))$
  - $\text{Weight\_node}(\text{node}, \text{index}) = \text{shardBalance} * (\text{node.numShards}() - \text{avgShardsPerNode})$
  - $\text{weight}(\text{node}, \text{index}) = \text{weight\_index}(\text{node}, \text{index}) + \text{weight\_node}(\text{node}, \text{index})$

# Elasticsearch Capacity Expansion

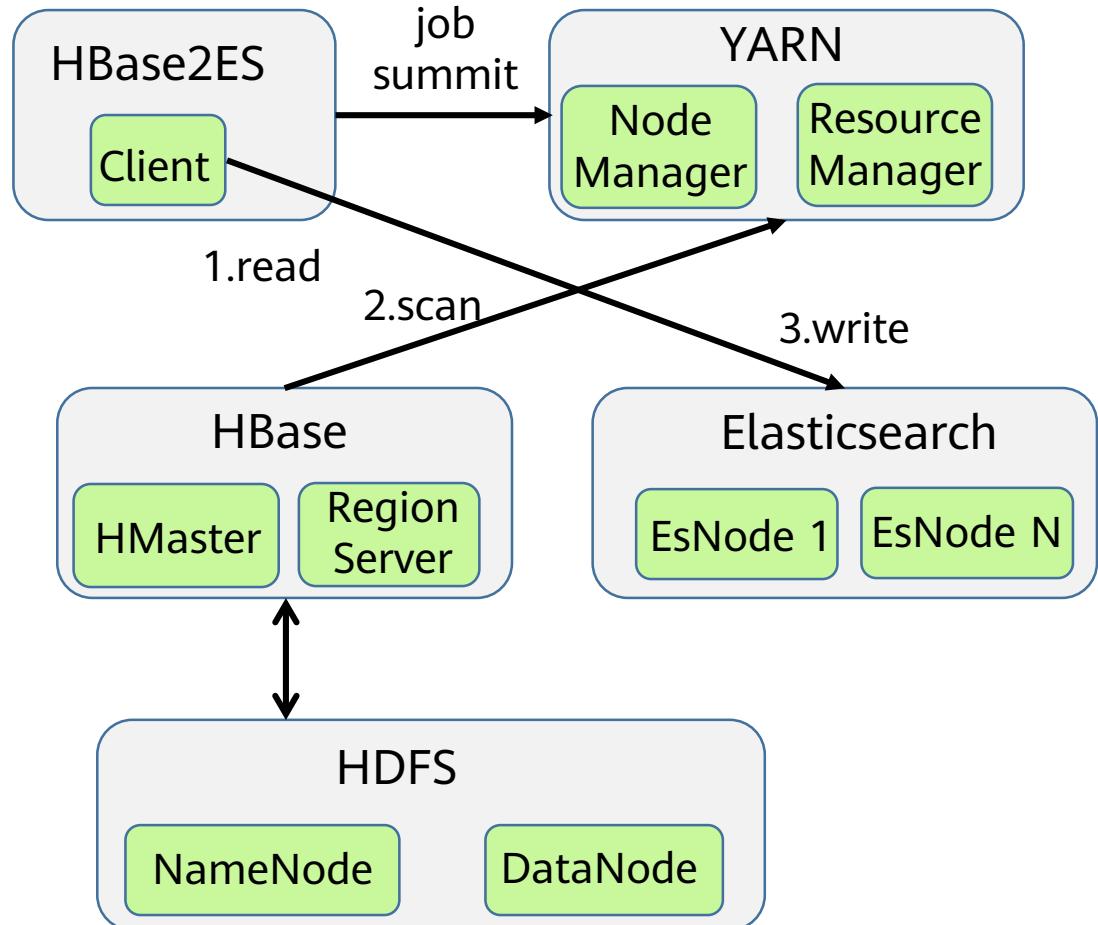
- Scenarios:
  - High physical resource consumption such as high CPU and memory usage of Elasticsearch service nodes, and insufficient disk space
  - Excessive index data volume for one Elasticsearch instance, such as 1 billion data records or 1 TB data
- Capacity expansion mode:
  - Add EsNode instances.
  - Add nodes with EsNode instances.
- After capacity expansion, use the automatic balancing policy.

# Elasticsearch Capacity Reduction

- Scenarios:
  - OS reinstallation on nodes required
  - Reduced amount of cluster data
  - Out-of-service
- Capacity reduction mode:
  - Delete an Elasticsearch instance on the Cloud Search Service (CSS) console.
- Precautions:
  - Ensure that replicas in the shard of the instance to be deleted exist in another instance.
  - Ensure that data in the shard of the instance to be deleted has been migrated to another node.

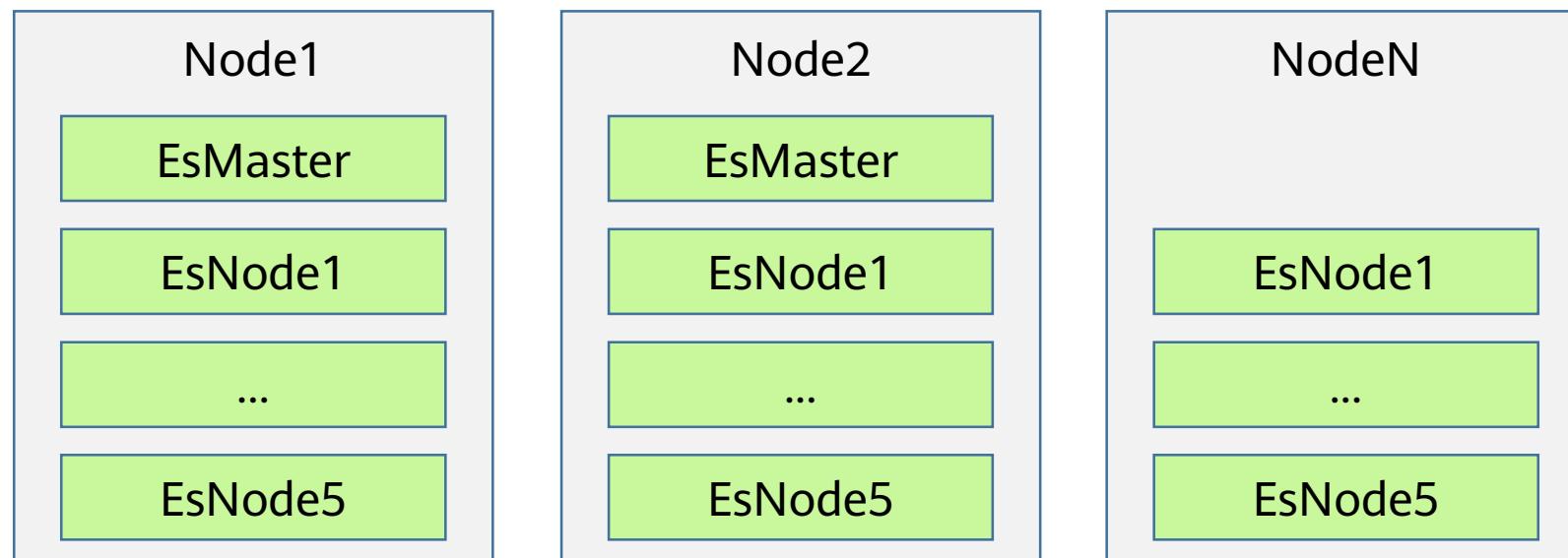
# Elasticsearch Indexing HBase Data

- When Elasticsearch indexes the HBase data, the HBase data is written to HDFS and Elasticsearch creates the corresponding HBase index data. The index ID is mapped to the rowkey of the HBase data, which ensures the unique mapping between each index data record and HBase data and implements full-text search of the HBase data.
- Batch indexing: For data that already exist in HBase, an MR task is submitted to read all data in HBase, and then indexes are created in Elasticsearch.



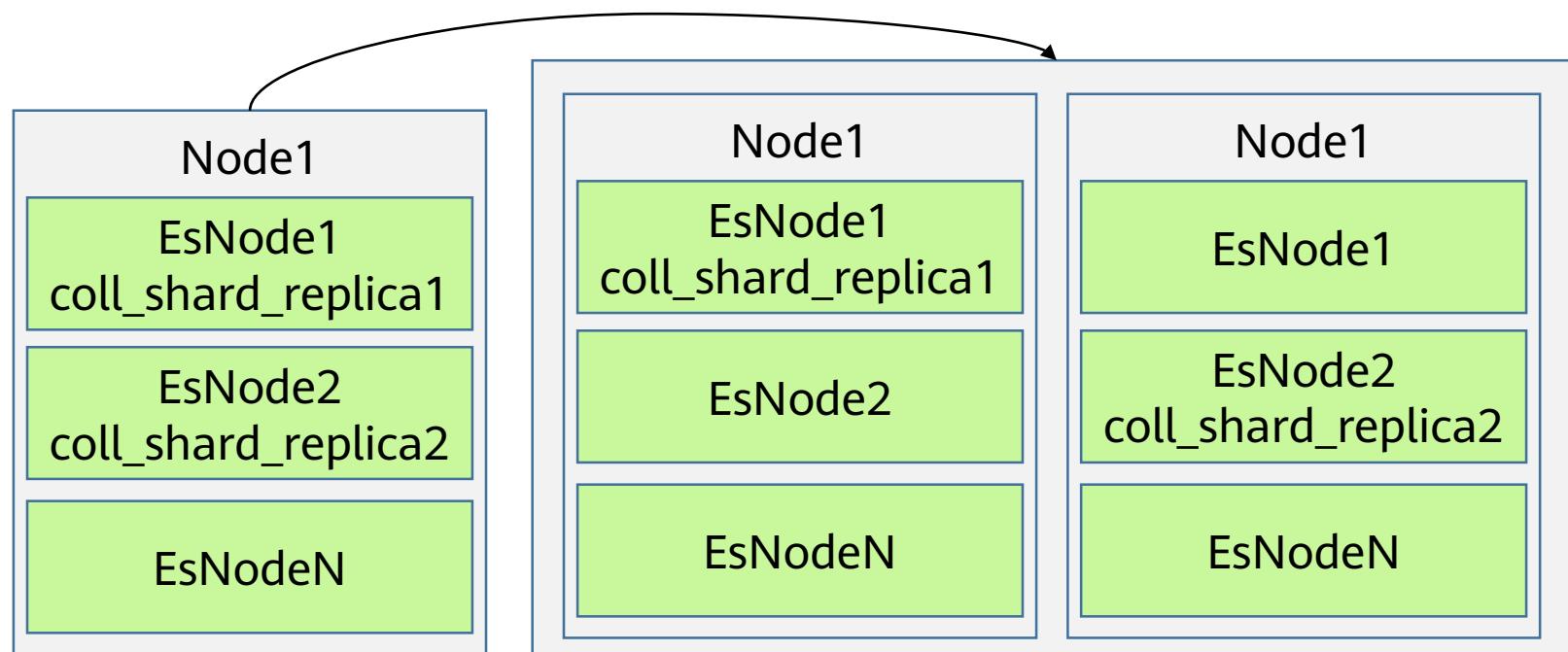
# Elasticsearch Multi-instance Deployment on a Node

- Multiple Elasticsearch instances can be deployed on one node, and differentiated from each other based on the IP address and port number. This method increases the usage of the single-node CPU, memory, and disk, and improves the indexing and search capability of Elasticsearch.



# Elasticsearch Cross-node Replica Allocation Policy

- When multiple instances are deployed on a single node with multiple replicas, if replicas can only be allocated across instances, a single-point failure may occur. To solve this problem, configure parameter `cluster.routing.allocation.same_shard.host` to `true`.



# New Features of Elasticsearch

- HBase full-text indexing
  - After the HBase table and Elasticsearch indexes are mapped, indexes and raw data can be stored in Elasticsearch and HBase, respectively. The HBase2ES tool is used for offline indexing.
- Encryption and authentication
  - Encryption and authentication are supported for a user to access Elasticsearch through a security cluster.

# Quiz

1. (Fill-in-the-blank) What is the basic unit that can be indexed in Elasticsearch? ( )
2. (Multiple-choice) Which type of data can be indexed using Elasticsearch? ( )
  - A. Structured data
  - B. Unstructured data
  - C. Semi-structured data
  - D. All of the above

# Quiz

3. (Single-choice) Which of the following open-source software is used to develop Elasticsearch? ( )
- A. MySQL
  - B. MongoDB
  - C. Memcached
  - D. Lucence

# Summary

- This chapter describes the basic concepts, functions, application scenarios, architecture, and key features of Elasticsearch. Understanding the key concepts and features of ElasticSearch allows you to better develop and use components.

# Recommendations

---

- Huawei Cloud Official Web Link:
  - <https://www.huaweicloud.com/intl/en-us/>
- Huawei MRS Documentation:
  - <https://www.huaweicloud.com/intl/en-us/product/mrs.html>
- Huawei TALENT ONLINE:
  - <https://e.huawei.com/en/talent/#/>

# Thank you.

把数字世界带入每个人、每个家庭、  
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and  
organization for a fully connected,  
intelligent world.

Copyright©2020 Huawei Technologies Co., Ltd.  
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.



# Chapter 13 Redis In-Memory Database



# Objectives

- Upon completion of this course, you will be able to:
  - Know Redis application scenarios.
  - Learn Redis data types.
  - Master Redis optimization methods.
  - Understand Redis service development.

# Contents

- 1. Redis Application Scenarios**
2. Redis Service Process
3. Redis Features and Data Types
4. Redis Optimization
5. Redis Application Cases

# Redis Overview

- Redis is a network-based, high-performance key-value in-memory database.
- Redis is similar to Memcached. Besides, it supports data persistence and diverse data types. It also supports the calculation of the union, intersection, and complement of sets on the server as well as multiple sorting functions.
- Redis has the following features:
  - High performance
  - Low latency
  - Access to diverse data structures
  - Persistence

# Redis Application Scenarios

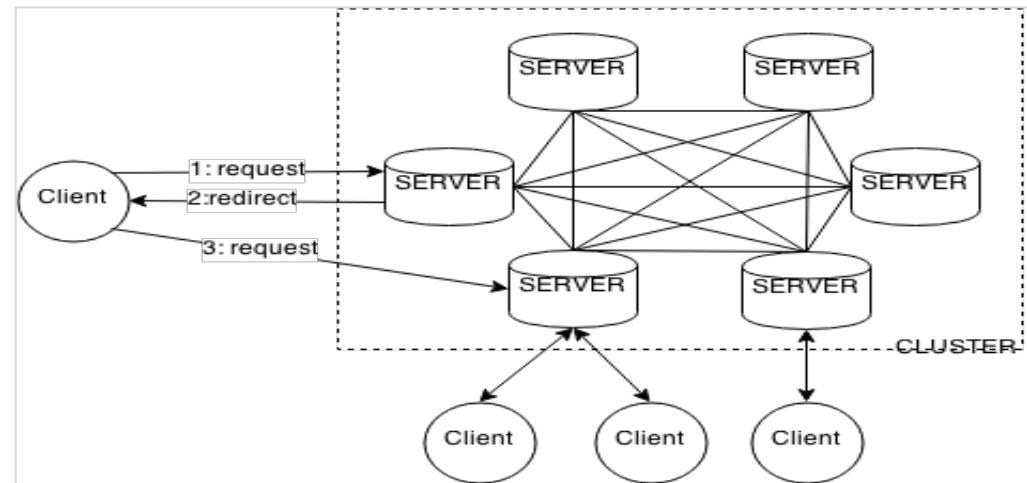
- Redis supports flexible data structures and various data operations and is applicable to the following scenarios:
  - Obtaining the latest  $N$  pieces of data, for example, the latest articles from a certain website.
  - Obtaining the top  $N$  applications from a ranking list. This operation is based on a certain condition, for example, sorting by times that people click "Like", while the preceding operation gives priority to time.
  - Applications that require precise expiration time, for example, user session information.
  - Counter applications, for example, counters that record website access times.
  - Constructing a queue system, for example, a message queue.
  - Cache, for example, table data that is frequently accessed in a cached relational database.
  - Publish/Subscription (pub/sub).
  - SMS verification code (The **expire** parameter is used to set the expiration time of the verification code).

# Contents

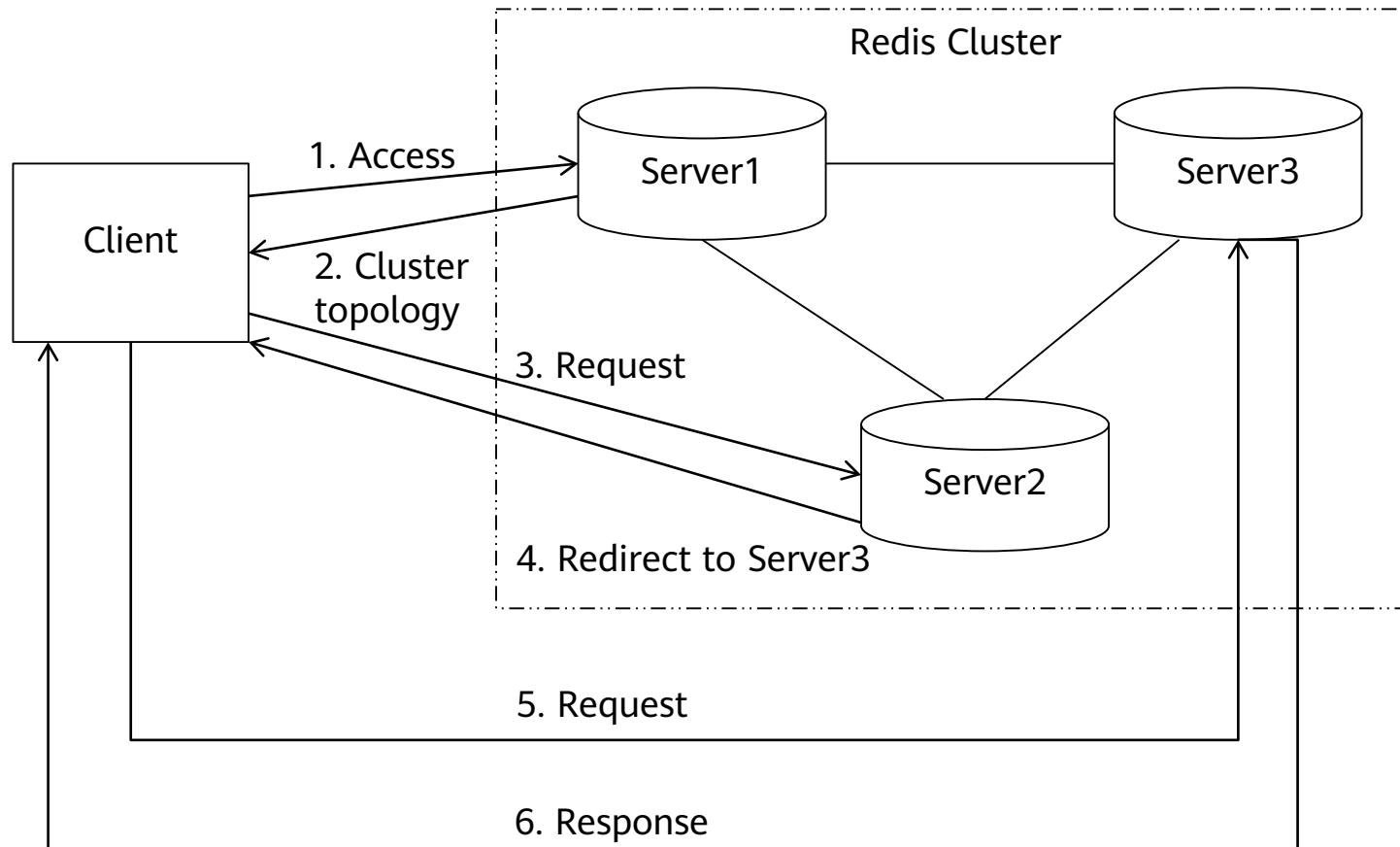
1. Redis Application Scenarios
- 2. Redis Service Process**
3. Redis Features and Data Types
4. Redis Optimization
5. Redis Application Cases

# Redis Architecture

- The Redis architecture is without a central control node. Node status information is exchanged between nodes by using the Gossip protocol.
- Each node maintains the mapping relationship from a key to a server.
- A client can send a service request to any node, and the node redirects the client instead of forwarding the request.
- If the cluster topology changes during the duration after the client sends the first request and before the request is redirected, the second redirection request will be redirected again until a target server is found.



# Redis Data Reading and Writing Processes (1)



# Redis Data Reading and Writing Processes (2)

- Redis data reading and writing processes are as follows:
  1. The client accesses any server node in the cluster and sends the cluster nodes request.
  2. The server node returns the cluster topology, including the cluster node list and the mapping relationship between slots and nodes. The client caches the cluster topology in memory.
  3. The client calculates the slot of the key based on hash  $(KEY)\%16384$  and queries the mapping relationship between slots and nodes, and then accesses the Server2 node that the key belongs to read and write data.
  4. Server2 receives the request sent by the client and checks whether the key exists. If the key does not exist, Server2 informs the client of redirecting the request to the Server3 node. If the key exists, Server2 returns the service operation results.
  5. The client receives the redirection response and sends a reading and writing request to Server3.
  6. Server3 receives the request and processes the request the same way as that in step 4.

# Contents

1. Redis Application Scenarios
2. Redis Service Process
- 3. Redis Features and Data Types**
4. Redis Optimization
5. Redis Application Cases

# Redis Feature - Multiple Databases

- Multiple databases
  - Each database is named in ascending order starting from 0 and cannot be customized.
  - By default, Redis supports 16 databases. Users can change the default number by modifying the **databases** parameter.
  - By default, Redis uses database 0.
  - **SELECT** number: Switches to a desired database by selecting a number.
  - The multiple databases are not completely isolated from each other. For example, the **flushall** command takes effect on all databases.
  - **flushall**: Clears data in all databases of a Redis instance.
  - **flushdb**: Clears data in the current database.

# Basic Commands of Redis

- Obtain the key name that meets rules.
  - Expression of keys: (?,\*,[],\?)
- Check whether a key exists.
  - exists key
- Delete one or more keys.
  - del key
  - del key1 key2
- Obtain data type of the obtained key value.
  - The return value can be of the **string**, **hash**, **list**, **set**, or **zset** type.
- Note: Redis commands are case insensitive.

# Redis Data Type - String

- The string type is the most basic data type in Redis. It can store content in any form, including binary data and even an image (binary content). The maximum capacity for storing a string value is 1 GB.
- Commands
  - `set/get(setnx)`
  - `mset/mget`
  - `incr/decr/incrby/decrby/incrbyfloat`
  - `append`
  - `strlen`

# Redis Data Type - Hash

- A hash value stores the mapping between fields and field values. The fields and field values must be strings. A hash key can store a maximum of  $2^{32}-1$  fields.
- Hash is suitable for storing objects.
- Redis can add or delete fields for any key without affecting other keys.
- Commands:

**hset/hget/hmset/hmget/hgetall(hsetnx)**

**hexists** (Check whether the attribute in the key exists.)

**hincrby** (The hash type does not have the hincr command.)

**hdel**

**hkeys/hvals**

**hlen** (Obtain the number of fields contained in a key.)

# Redis Data Type - List

- List is an ordered string list. The list is implemented using a bidirectional link (linked list). List can also be used as a queue.
- A key of the list type can contain a maximum of  $2^{32}-1$  elements.
- Commands:

**lpush/rpush/lpop/rpop;**

**llen/lrange** (-1 indicates the location of the last element.)

**lrem** (lrem key count value) count has the following three situations:

count > 0: searches from the table header to the table tail and removes the elements whose values are equal to **value**. The number of elements is **count**.

count < 0: searches from the table tail to the table header and removes the elements whose values are equal to **value**. The number of elements is the absolute value of **count**.

count = 0: removes all elements whose values are equal to **value** from the table.

**lindex**: queries the data of a specified corner mark.

**lset**: changes the value of a specified corner mark.

**ltrim**: truncates and retains the specified data.

**linsert**: inserts elements before and after a specified element.

**rpoplpush**: transfers an element from one list to another.

# Redis Data Type - Set

- The elements in a set are not duplicate and are unordered. A key of the set type can store a maximum of  $2^{32}-1$  elements.
- Commands

**sadd/smembers/srem/sismember;**

**sdiff** (difference set)/**sinter** (intersection set)/**sunion** (union set);

**sdiffstore/sinterstore/sunionstore;**

**scard** (Obtains the set length.)/**spop** (Randomly takes an element out of the set and deletes it.);

**srandmember key [count]:**

If **count** is a positive number and less than the set cardinality, the command returns an array containing **count** different elements.

If **count** is greater than or equal to the set cardinality, the entire set is returned.

If **count** is a negative number, the command returns an array. The elements in the array may appear multiple times, and the length of the array is the absolute value of **count**.

# Redis Data Type - Sorted Set

- Each element in the set is associated with a score based on the set type. In this way,  $N$  elements with the highest scores can be easily obtained.
- Commands

**zadd/zscore/zrange/zrevrange/**

**zrangebyscore** (A closed interval is used by default. Users can use "(" to adopt an open interval.)

**zincrby/zcard/zcount** (Obtains the number of elements in a specified score range. A closed interval is used by default. Users can use "(" to adopt an open interval.)

**zrem/zremrangebyrank/zremrangebyscore** (A closed interval is used by default. Users can use "(" to adopt an open interval.)

Extension: **+inf** (positive infinity) **-inf** (negative infinity)

# Setting TTL of a Key in Redis (Using Expire Command)

- In Redis, users can run the **Expire** command to set the time to live (TTL) of a key. After the TTL expires, Redis automatically deletes the key.
  - **Expire**: Sets the TTL (in seconds).
  - **Pexpire**: Sets the TTL (in milliseconds).
  - **ttl/p ttl**: Checks the remaining TTL of the key.
  - **Persist**: Cancels the TTL.
  - **expireat [key]**: Indicates the unix timestamp 1351858600.
  - **pexpireat [key]**: Indicates the unix timestamp 1351858700000. (unit: ms)
- Application scenarios:
  - Time-limited preferential activity information
  - Website data cache (for data that needs to be updated periodically, for example, bonus point rankings)
  - Limiting the frequency to access a website (for example, a maximum of 10 times per minute).

# Redis Pipeline

- The pipeline function of Redis is not available in the command line. However, Redis supports pipelines and can be used on Java clients (jedis).
- Test results:
  - If pipeline is not used, it takes 328 ms to insert 1,000 data records.

```
for (int i = 0; i < 1000; i++) {  
    jedis.set("test"+i, "test"+i);  
}
```

- If pipeline is used, it takes 37 ms to insert 1,000 data records.

```
Pipeline pipelined = jedis.pipelined();  
for (int i = 0; i < 1000; i++) {  
    pipelined.set("test"+i, "test"+i);  
}  
pipelined.sync();
```

# Data Sorting in Redis (Using sort Command)

- The **sort** command can sort the list, set, and ordered set.
  - `sort key [desc] [limit offset count]`
  - by reference key (The reference key can be a character string or a field of the hash type. The format of the hash type is "key name -> field name").
    - If the reference key does not contain an asterisk (\*), the data is not sorted.
    - If the reference key of an element does not exist, the value of the reference key is **0** by default.
- Extended get parameter: The rule of the get parameter is the same as that of the by parameter. `get #` (Returns the value of the element.)
- Extended store parameter
  - Use the store parameter to save the sort result to a specified list.
- Performance optimization:
  - Reduce the number of elements in the key to be sorted as much as possible.
  - Use the limit parameter to obtain only the required data.
  - If there is a large amount of data to be sorted, use the store parameter to cache the result.

# Redis Task Queues

- Task queue: Use **lpush** and **rpop** to implement common task queues.
- Priority queue:
  - **brpop key1 key2 key3 timeout second**

# Redis Persistence

- Redis supports two persistence modes, which can be used separately or together.
  - RDB mode (Default)
  - AOF mode

# Redis Persistence - RDB

- The persistence in Redis Database (RDB) mode is implemented through snapshots. When certain conditions are met, Redis automatically takes snapshots of all data in the memory and stores the data to a disk. By default, the data is stored in the `dump.rdb` file.
- The time when Redis takes snapshots (in the `redis.conf` configuration file) is as follows:
  - `save 900 1`: A snapshot is taken if at least one key is changed within 900 seconds.
  - `save 300 10`
  - `save 60 10000`
- Run the `save` or `bgsave` command to enable Redis to perform snapshot operations.
  - The difference between the two commands is that the `save` command is used by the main process to perform snapshot operations, which block other requests, and the `bgsave` command is used by Redis to execute the `fork` function to copy a subprocess for snapshot operations.

# Redis Persistence - AOF (1)

- Append Only File (AOF) persistence is implemented through log files. By default, AOF is disabled in Redis. Users can enable it by setting the **appendonly** parameter.
  - **appendonly yes**
- Synchronization policies for writing commands of Redis:
  - **appendfsync always**: The command is executed every time.
  - **appendfsync synchronizedsec**: By default, the synchronization is performed every second (recommended, default).
  - **appendfsync no**: The synchronization is performed by the operating system every 30 seconds.

# Redis Persistence - AOF (2)

- Dynamically switch the Redis persistence mode from RDB to AOF (Redis 2.2 or later is supported).
  - CONFIG SET appendonly yes
  - (Optional) CONFIG SET save ""
- Note: When Redis is started, if both RDB persistence and AOF persistence are enabled, the program preferentially uses the AOF mode to restore the data set because the data stored in the AOF mode is the most complete. If the AOF file is lost, the database is empty after the startup.
- Note: To switch the running Redis database from RDB to AOF, users can use the dynamic switchover mode and then modify the configuration file. (Do not modify the configuration file on your own and restart the database. Otherwise, the data in the database is empty.)

# Redis Memory Usage

- 1 million key-value pairs (key ranges from 0 to 999999, and value is **hello world**) on a 32-bit laptop use 100 MB memory.
- If a 64-bit operating system is used, more memory is occupied. This is because the pointer in the 64-bit operating system occupies eight bytes. However, the 64-bit operating system supports larger memory. It is recommended that the 64-bit server be used to run large-scale Redis services.

# Contents

1. Redis Application Scenarios
2. Redis Service Process
3. Redis Features and Data Types
- 4. Redis Optimization**
5. Redis Application Cases

# Redis Optimization (1)

- Simplify key names and values.
  - Key name: The key name should be as simple as possible, but do not use incomprehensible key names just for saving space.
  - Key value: If the number of key values is fixed, the values can be represented by 0 and 1, for example, **male/female** or **right/wrong**.
- If data persistence is not required in service scenarios, disable all data persistence modes to achieve optimal performance.
- Optimize internal coding (only need to have a basic understanding of it).
  - Redis provides two internal coding methods for each data type. Redis can automatically adjust the coding method in different scenarios.
- SLOWLOG [get/reset/len]
  - The commands whose execution time is larger than the value (in microseconds, one second = one million microseconds) specified by **slowlog-log-slower-than** will be recorded.
  - **slowlog-max-len** determines the maximum number of logs that can be saved in **slowlog**.

# Redis Optimization (2)

- Modify the memory allocation policy of the Linux kernel.
  - Add `vm.overcommit_memory = 1` to `/etc/sysctl.conf` and restart the server.
  - Alternatively, run the `sysctl vm.overcommit_memory=1` command (take effect immediately).

# Redis Optimization (3)

- Disable Transparent Huge Pages (THP).
  - THP may cause memory locks and affect Redis performance. It is recommended that this function be disabled.
    - THP is used to improve memory management performance.
    - THP is not supported in 32-bit RHEL 6.
  - Run the following command as user root:
    - `echo never > /sys/kernel/mm/transparent_hugepage/enabled`
    - Add the command to the `/etc/rc.local` file.

# Redis Optimization (4)

- Modify the maximum number of TCP connections in Linux.
  - This parameter determines the length of a completed queue (after three-way handshake) in the TCP connection. The value must be less than or equal to the `/proc/sys/net/core/somaxconn` value defined by the Linux system. The default value is 511 for Redis and 128 for Linux. When the system has a large number of concurrent requests and the client responds slowly, users can set the value by referring to the two parameters.
  - `echo 511 > /proc/sys/net/core/somaxconn`
  - Note: This parameter does not limit the maximum number of Redis connections. To limit the maximum number of Redis connections, modify the **maxclients** parameter. The default maximum number of connections is 10000.

# Redis Optimization (5)

- Limit the Redis memory size.
  - Run the **info** command of Redis to view the memory usage.
  - If **maxmemory** is not set or set to 0, the memory size is not limited in a 64-bit system, and the maximum memory size is 3 GB in a 32-bit system.
  - Modify **maxmemory** and **maxmemory-policy** in the configuration file.
  - **maxmemory** indicates the maximum memory.
  - **maxmemory-policy** indicates the data clearance policy when the memory is insufficient.
- If the total data volume is not large and the memory is sufficient, users do not need to limit the memory used by Redis. If the data volume is unpredictable and the memory is limited, limit the memory used by Redis to prevent Redis from using the swap partition or prevent OOM errors.
- Note: If the memory is not limited, the swap partition is used after the physical memory is used up. In this case, the performance is low. If the memory is limited, data cannot be added after the specified memory is reached. Otherwise, an OOM error is reported. Users can set **maxmemory-policy** to delete data when the memory is insufficient.

# Redis Optimization (6)

- Redis is a single-thread model. Commands from the client are executed in sequence. Therefore, users can use pipelines or commands to add multiple pieces of data at a time, for example:

set	mset
get	mget
lindex	lrange
hset	hmset
hget	hmget

# Contents

1. Redis Application Scenarios
2. Redis Service Process
3. Redis Features and Data Types
4. Redis Optimization
- 5. Redis Application Cases**

# Application Development Case Analysis

- Business objective
  - Some online recommendation services are demanding of velocity because users send a service request upon obtaining the recommendation result. If the latency is high, user experience will be affected. Therefore, real-time access to the recommendation result must be a concern for online recommendation services.
- Service solution
  - Redis functions as a cache mechanism. This reduces times and data volume that a user reads data from a database and a file system.
  - The calculated recommendation results are cached to Redis. Next time when users obtain the same recommendation results, they can directly read data from the cache.

# Application Development Case Analysis

- Data structure design
  - The user information used in the calculation is stored and obtained using the hash structure. The key is **userinfo-<user id>**, and the fields contain the user attributes, such as the name, gender, age, and hobby.
    - Example: userinfo-19810101,name,zhangsan
    - userinfo-19810101,sex,female
  - Multiple recommendation results (offerings) may exist. To avoid repeated recommendation of an offering, the set result is used for storage and access. The key is designed as **res-<user id>**.
- Data reading and writing principles
  - The MapReduce task periodically imports user data from the backend data storage source (Hbase) to Redis every day.
  - The service system obtains data from Redis. If Redis data fails to be obtained, the service system then obtains data from the backend HBase or by real-time computing, and then writes the data simultaneously to Redis.

# Quiz

1. Which Redis data structure is appropriate when the top  $N$  records for an application need to be obtained?
2. Does the Redis server forward a key operation request that does not belong to its node to the correct node?

# Summary

- This course introduces the application scenarios, features, data types, and service data reading and writing processes of the Redis component. After learning this course, users can select a proper Redis data structure based on the specific service scenario to access Redis data.

# Recommendations

---

- Huawei Cloud Official Web Link:
  - <https://www.huaweicloud.com/intl/en-us/>
- Huawei MRS Documentation:
  - <https://www.huaweicloud.com/intl/en-us/product/mrs.html>
- Huawei TALENT ONLINE:
  - <https://e.huawei.com/en/talent/#/>

# Thank you.

把数字世界带入每个人、每个家庭、  
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and  
organization for a fully connected,  
intelligent world.

Copyright©2020 Huawei Technologies Co., Ltd.  
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.



# Revision Record

Do Not Print this Page

Course Code	Product	Product Version	Course Version
H13-711	MRS		V3.0

Author/ID	Date	Reviewer/ID	New/Update
Rao Lilu/rwx350111	2020.4.15	Ling Huiguo/lwx570550	New
Rao Lilu/rwx350111	2020.09.04	Zhu Lin/zwx832322	New

# Chapter 14 Huawei Big Data Solution



# Foreword

---

- Brand-new hybrid cloud solution of Huawei big data and data mid-end services based on Huawei Kunpeng processors.
- The solution implements cross-cloud seamless synchronization of advanced service capabilities and multi-scenario collaboration, and supports Huawei Kunpeng and Ascend computing capabilities to help governments and enterprises realize refined resource control, cross-cloud hybrid orchestration, collaboration of multiple scenarios, such as combining online development and test with offline deployment, and online training with offline inference. In this case, each enterprise can build its own cloud.
- Huawei's big data services are deployed in 150 countries and regions, serving more than 7,000 customers.
- As cloud-based transformation of government and enterprises develops, hybrid clouds are favored by more and more government and enterprise users. User requirements drive industry evolution and replacement. HUAWEI CLOUD Stack 8.0 is developed from "hybrid resource management" to "refined management and control + hybrid services". It redefines the hybrid cloud.

# Objectives

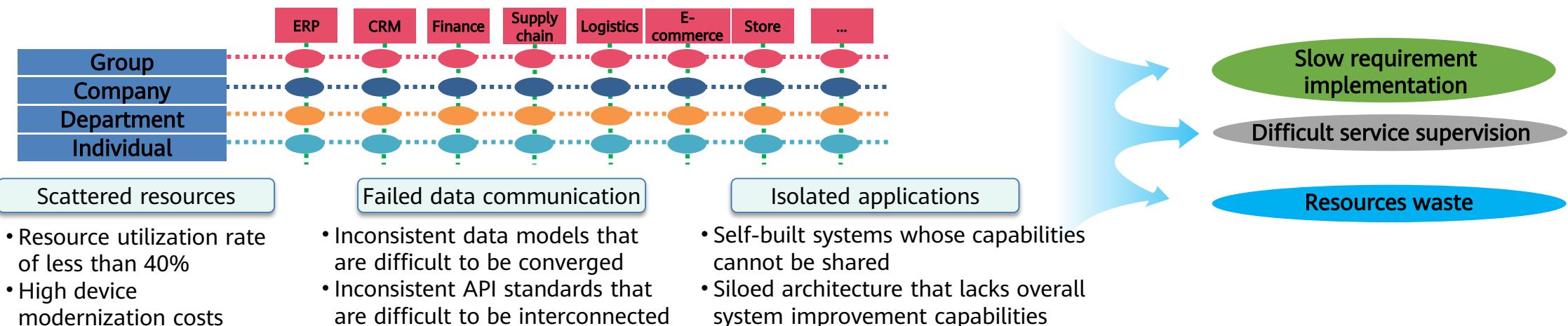
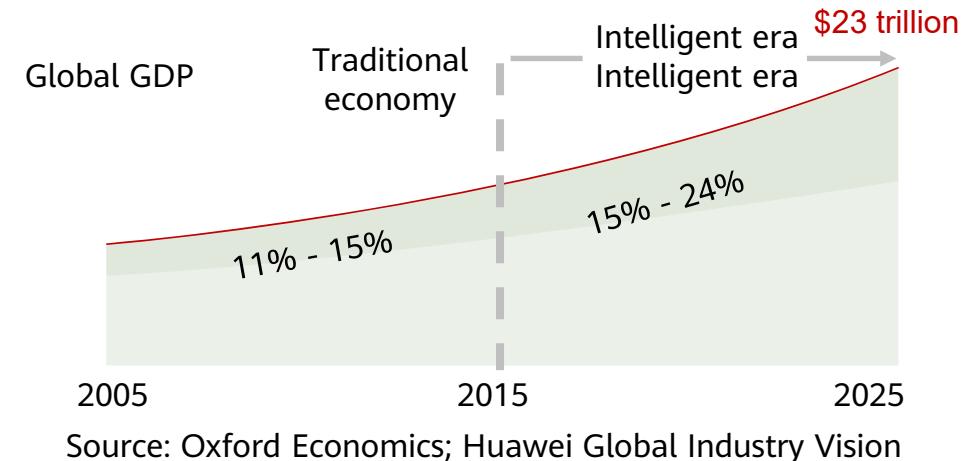
- On completion of this course, you will be able to:
  - Understand the development trend of the ICT industry.
  - Master Huawei big data services.
  - Know Huawei intelligent data lake operations platform.

# Contents

- 1. Development Trend of the ICT Industry**
2. HUAWEI CLOUD Big Data Services
3. HUAWEI CLOUD Intelligent Data Lake Operation Platform

# Changes in the ICT Industry: Bottleneck of System Construction

Changes in production elements: Data becomes a new asset, and intelligence becomes a new productivity.



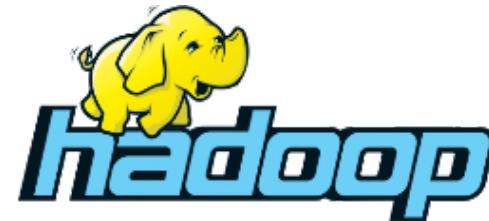
# Changes in Technology Development: Technical Architecture Evolution of the Cloud-based Data Mid-End



## Data warehouse

Advantage: clean data  
Disadvantages: no AI, difficult to cope with future challenges  
Challenges

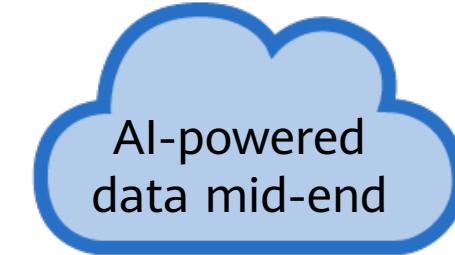
Easy to become a data silo



## Big data platform

Advantage: rich ecosystem  
Disadvantages: Schemaless, manual management, and complex processing

Easy to become a data swamp



## Data mid-end +AI

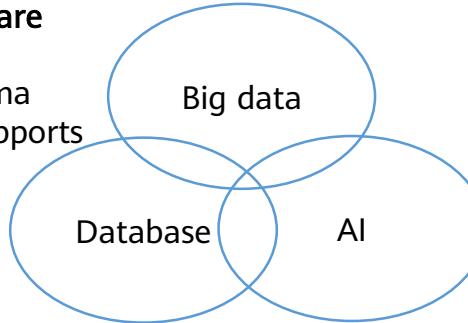
Data intelligence  
Industry intelligence  
Facilitate data monetization and reduce enterprise costs

Cloud-based mid-end data convergence and monetization

# Changes in Technology Development: Gradual Integration of Data and AI Technologies

Trend 1: Data lake and data warehouse technologies are integrated.

Logic: The data lake requires high performance, schema validation, and transactional update capabilities. It supports multiple open source compute engine ecosystems.



Trend 2: AI-based SQL enables intelligent applications.

Logic: Do more with the language you have mastered.

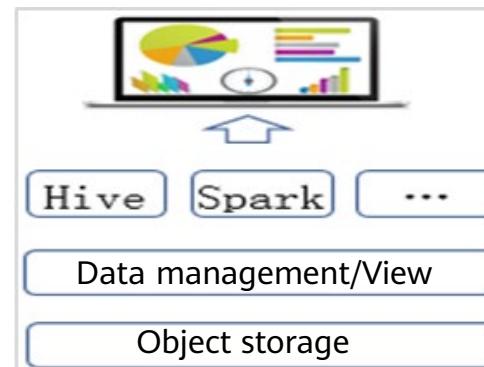


Trend 3: Data management and resource management make up for the weaknesses of the AI industry.  
Logic: The success of AI depends on the understanding of data. Only one resource pool is required to maximize resource utilization.

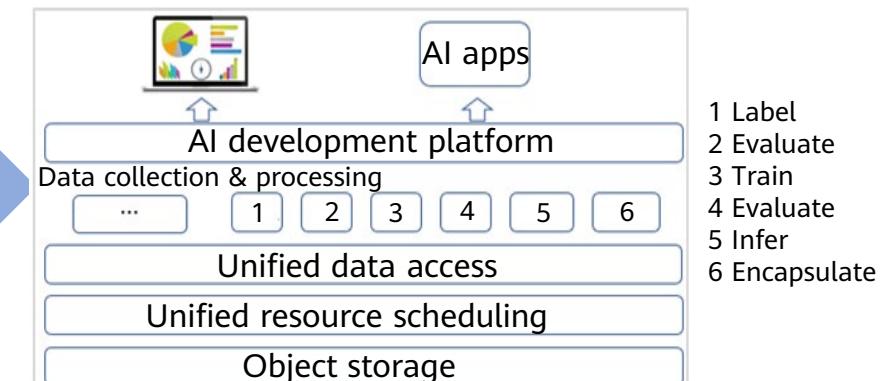
**Big data computing**  
Storage-compute decoupling + brute-force scanning



**Introduction of database**  
ACID, updating, indexing

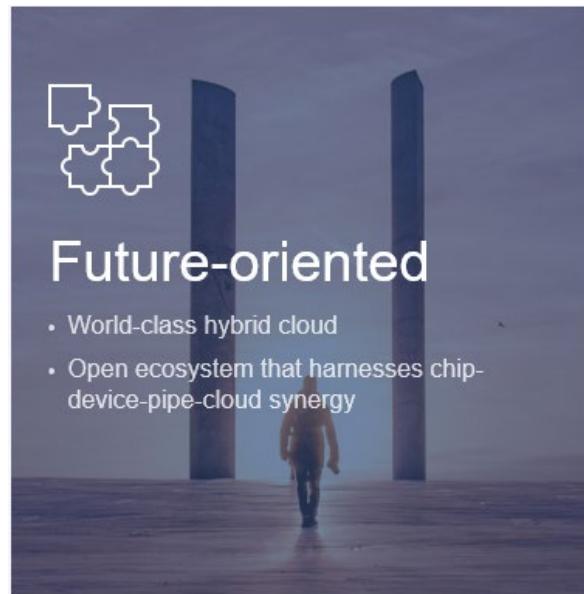


**Data management added to AI development**  
Data preprocessing, feature processing, unified access, and resource scheduling

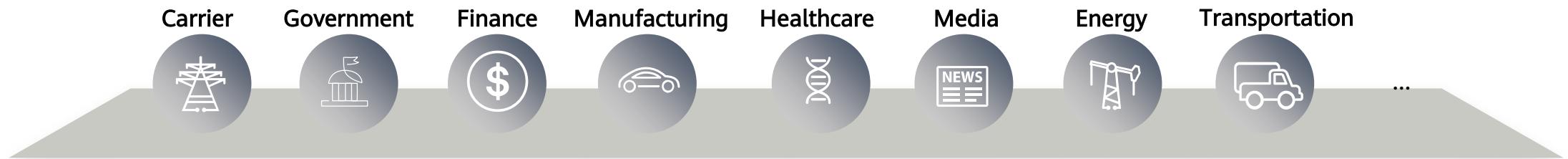


# HUAWEI CLOUD Services

- HUAWEI CLOUD is a cloud service brand of Huawei. It provides customers with more than 30 years of experience in ICT infrastructure and product solutions in the ICT infrastructure field, and is committed to providing stable, reliable, secure, reliable, and sustainable cloud services, as the "fertile soil" of the smart world, we will promote the universal AI that is "affordable, good, and easy to use". As the foundation, HUAWEI CLOUD provides a powerful computing platform and an easy-to-use development platform for Huawei's full-stack all-scenario AI strategy.
- Huawei aims to build an open, cooperative, and win-win cloud ecosystem and helps partners quickly integrate into the local ecosystem. HUAWEI CLOUD adheres to business boundaries, respects data sovereignty, does not monetize customer data, and works with partners for joint innovation to continuously create value for customers and partners. By the end of 2019, HUAWEI CLOUD has launched 200+ cloud services and 190+ solutions, serving many well-known enterprises around the world.



# HUAWEI CLOUD Stack, an Ideal Choice of Hybrid Cloud for Governments and Enterprises



Share 200+ services of 18 categories, 200+ solutions, and big cloud ecosystem



Enterprise-perspective management



Diverse cloud services and ecosystems



Seamless evolution with multi-architecture computing power

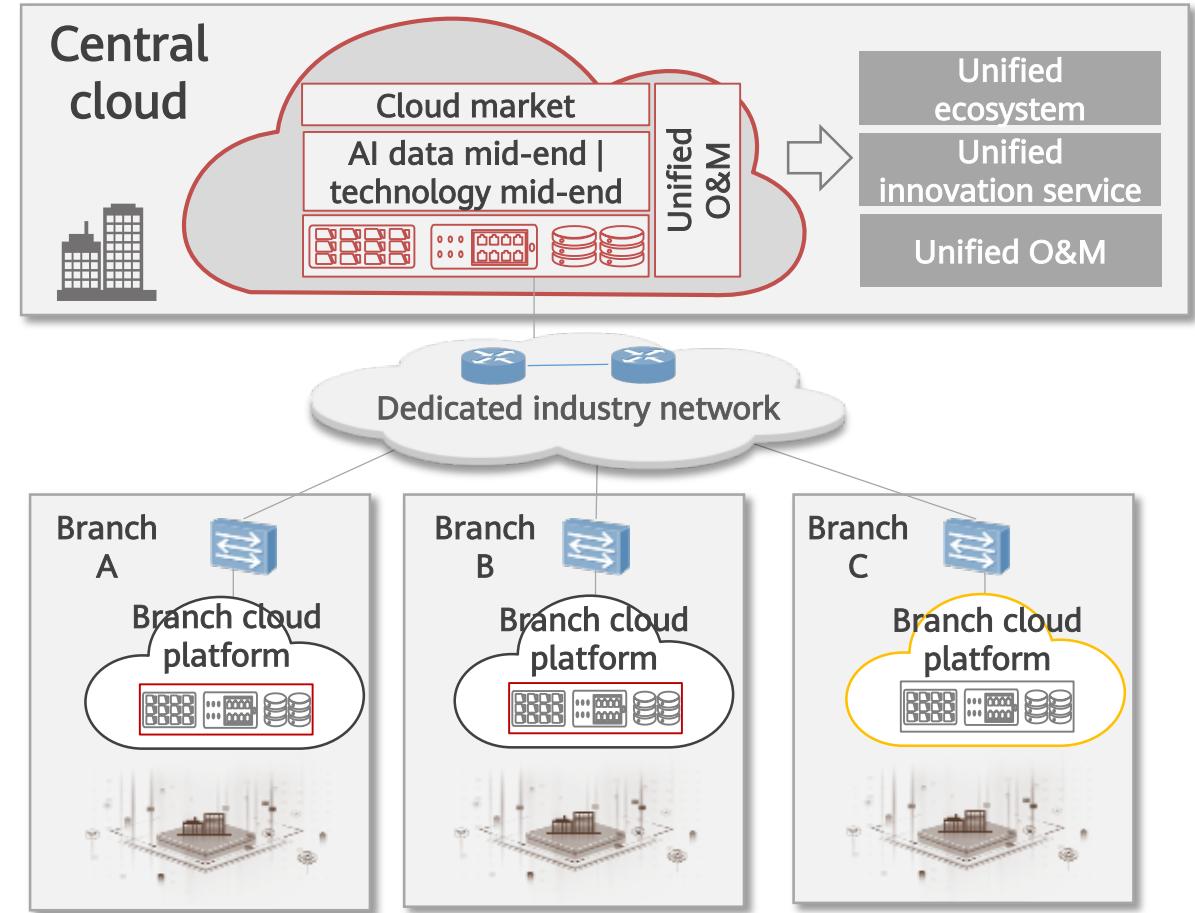
# Industrial Cloud Solution Overview

Scenario:

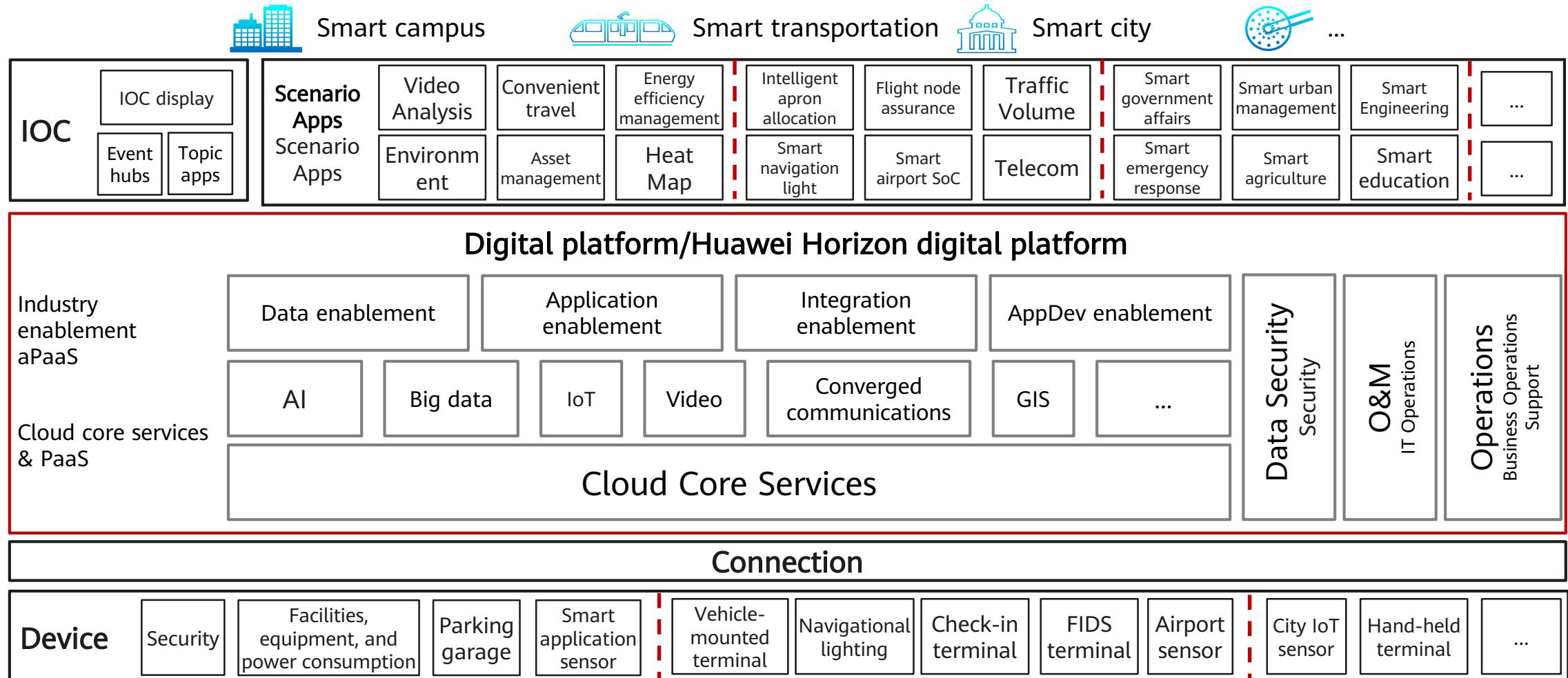
- HUAWEI CLOUD Stack provides a 1+N architecture to accelerate the digital transformation of governments and enterprises. The core objective is to help government and enterprise customers digitize and thereby activate core industry assets accumulated over the years using leading-edge technologies including cloud computing, big data, and AI. Rebuilding core application systems using a distributed architecture helps transform customers' core assets into digital services that can be made available to external systems, leading to improved IT efficiency, accelerated ecosystem growth, and energized innovation.

Advantages:

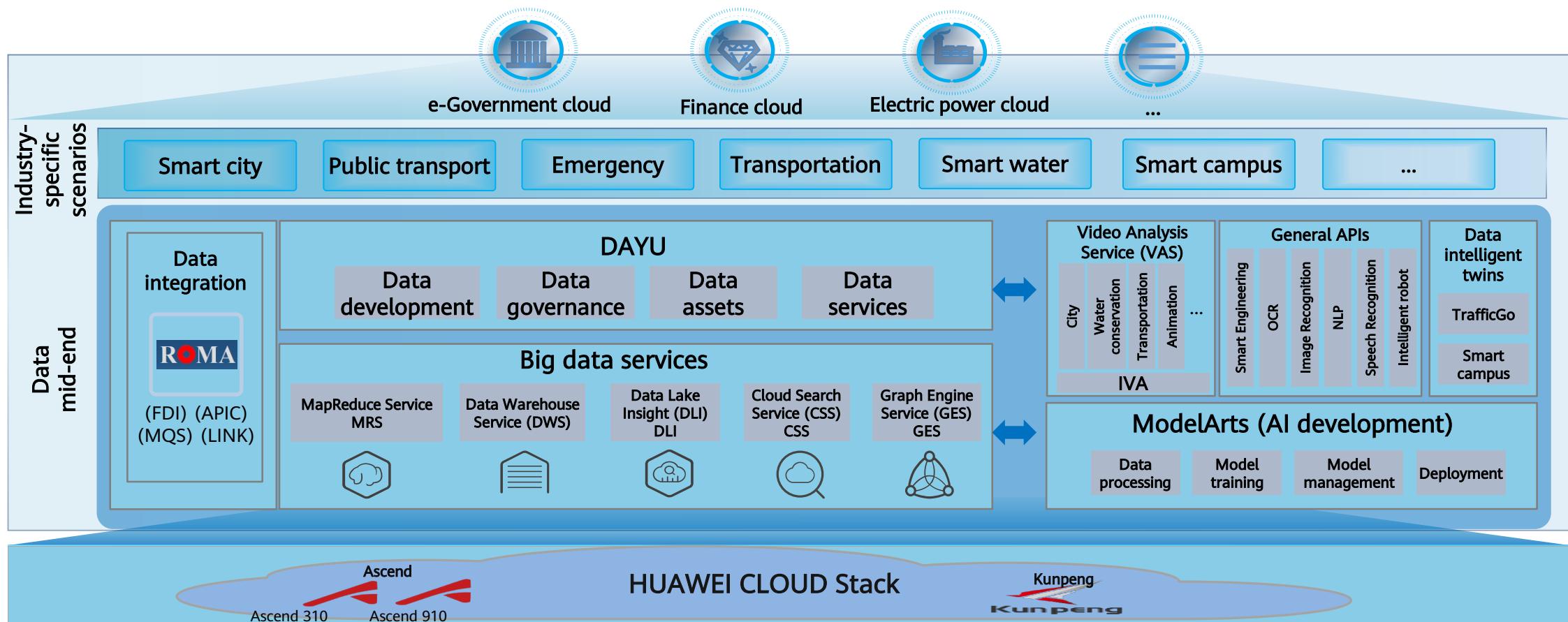
- One cloud for multi-branch, multi-site, multi-DC, and multi-edge scenarios: It helps customers integrate and evolve towards a distributed cloud architecture and enables flexible multi-level management and O&M.
- Four-dimensional, smart collaboration: Collaboration is implemented for resources, data, applications, and management.
- Continuous innovation: Big data, AI, container, microservice, and edge computing are used to transform customers' core application systems and move them to the cloud.
- Open ecosystem: An open architecture supports aggregation of a large industry ecosystem and enables joint innovation of industry applications.



# Scenario-Specific Solutions Based on Huawei Digital Platform



# Huawei Data Mid-End Solution

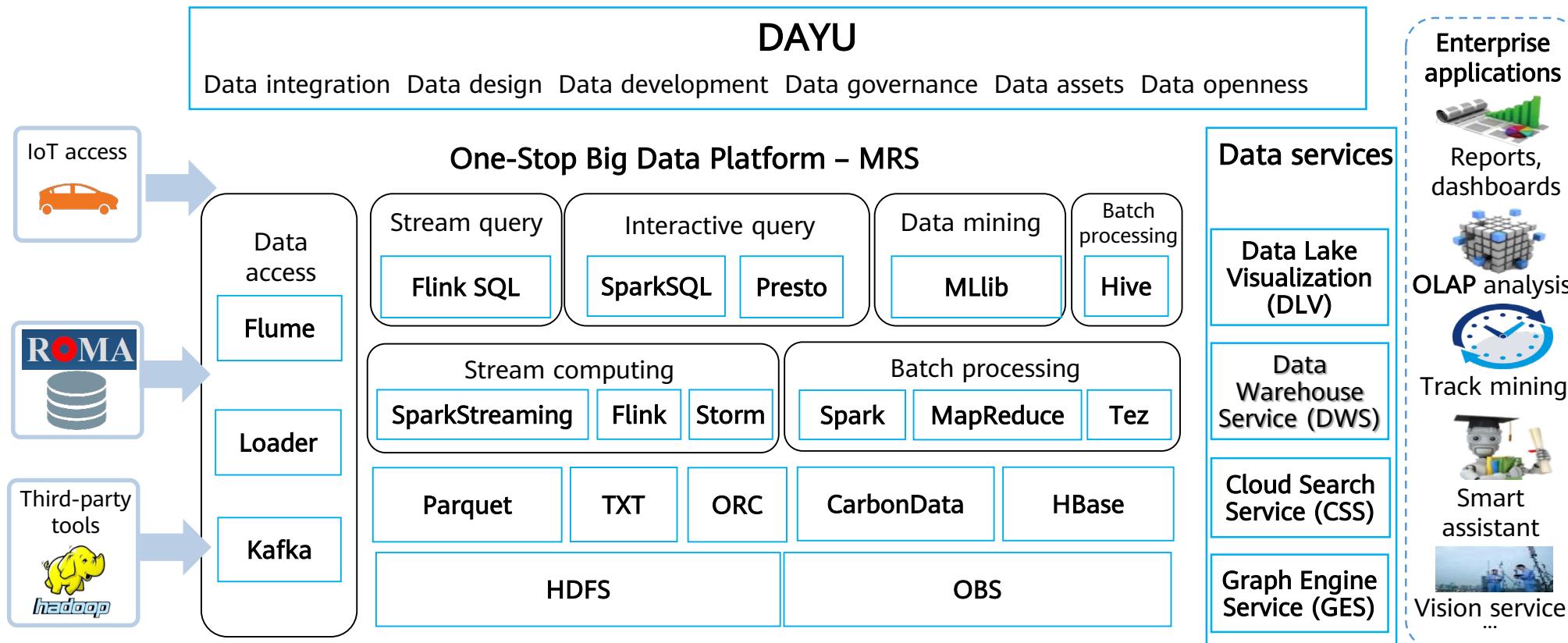


- **One-stop AI + data services**  
Provide an end-to-end, data-driven data management and AI innovation platform for governments, industry clouds, and large enterprises.

- **Guaranteed full-stack business continuity**  
Huawei-developed Kunpeng + Ascend AI chips plus traditional GPU solutions

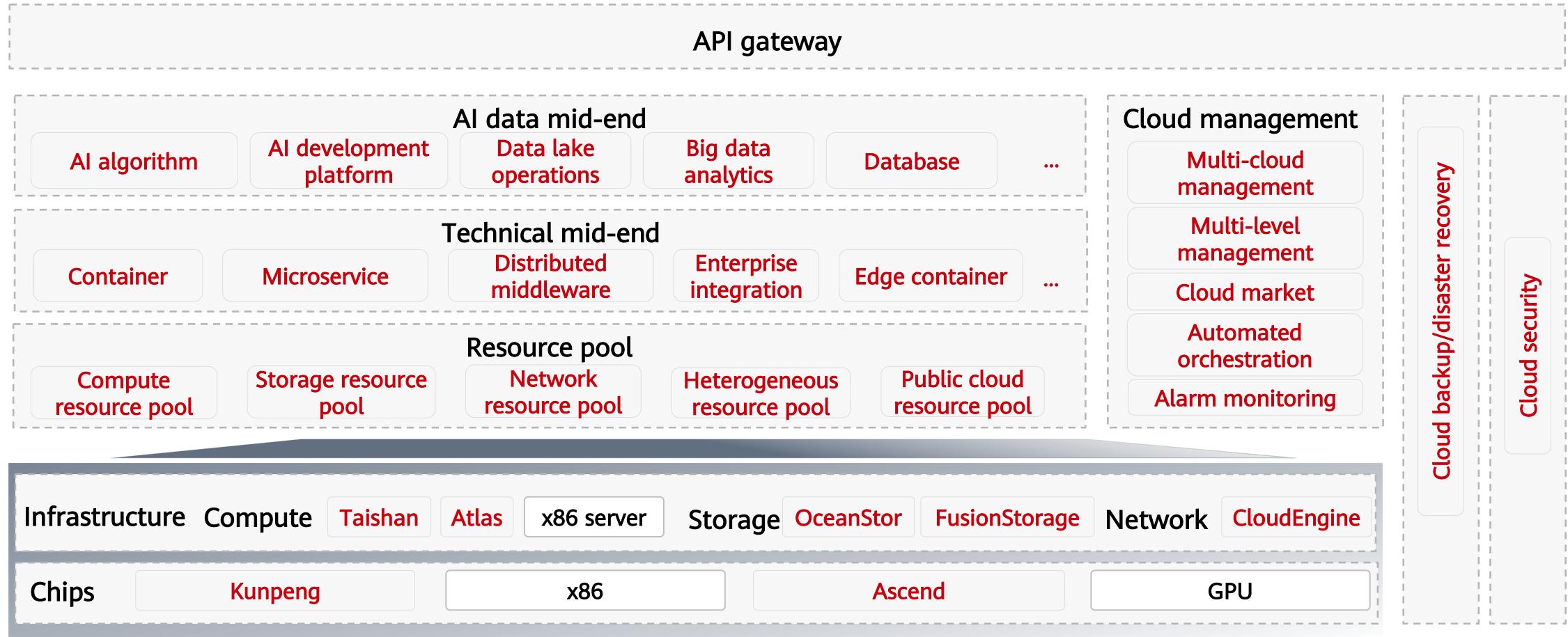
# HUAWEI CLOUD Big Data Services

- One-stop service for data development, testing, and application



- **100% compatibility** with open-source ecosystems, 3rd-party components managed as plugins, one-stop enterprise platform
- Storage-compute decoupling + Kunpeng optimization for **better performance**

# HUAWEI CLOUD Stack 8.0 Functional Architecture

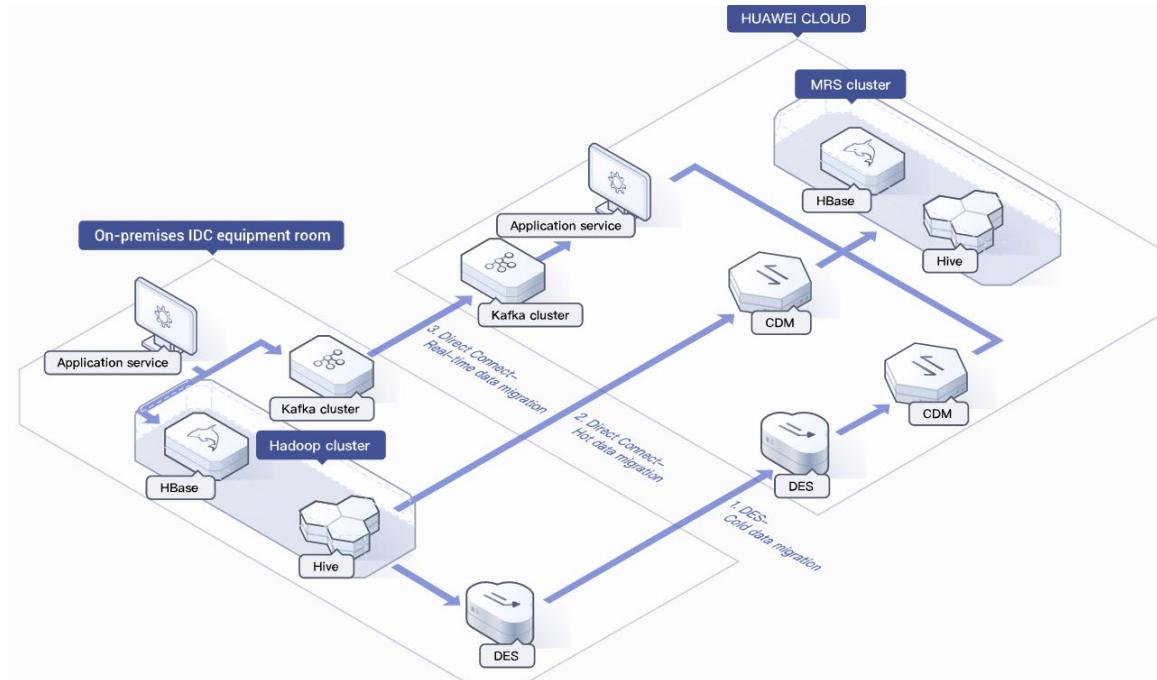


# Contents

1. Development Trend of the ICT Industry
2. **HUAWEI CLOUD Big Data Services**
3. HUAWEI CLOUD Intelligent Data Lake Operation Platform

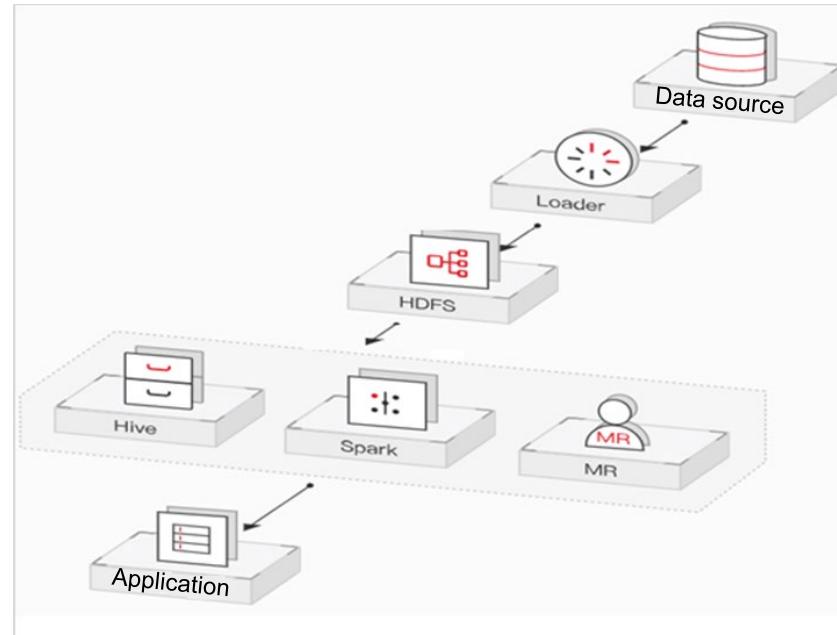
# HUAWEI CLOUD Big Data Services

- **Enterprise-class:** Based on the Huawei FusionInsight enterprise-level big data platform, HUAWEI CLOUD big data services allow deployment of tens of thousands of nodes, isolate resources between different jobs, and provide SLA guarantee for multi-level users.
- **Easy O&M:** You do not need to purchase and maintain hardware. A dedicated enterprise-level cluster management system is provided for you to easily manage the big data platform. Platform exceptions are reported by SMS or email. O&M has never been easier.
- **High security:** Proven by Huawei's professional security teams and Germany PSA security authentication tests, HUAWEI CLOUD big data services are capable of providing secure, cloud-based big data services. Based on Kerberos authentication, HUAWEI CLOUD big data services provide role-based security control and sound audit functions.
- **Low cost:** With diverse cloud infrastructure and rich compute/storage facilities, MRS clusters can be created, scaled in/out when necessary, and destroyed after use, ensuring the lowest cost.



# MRS (Hadoop)

- Based on the Huawei FusionInsight enterprise-level big data platform, MRS (Hadoop) provides enterprise-class scheduling to isolate resources between different jobs, and supports SLA guarantee for multi-level users.
- A dedicated enterprise-level cluster management system is provided for you to easily manage the big data platform. Platform exceptions are reported by alarms.
- Based on Kerberos authentication, MRS (Hadoop) provides highly secure big data service featuring role-based security control and sound audit functions.
- MRS (Hadoop) provides key capabilities such as statistical analysis, data mining, full-text retrieval, and streaming computing and analysis.
- MRS (Hadoop) provides offline data analysis, online query, and real-time computing.



# Storage-Compute Decoupling

- Storage-compute decoupling and scaling, ensuring resource utilization and performance

## Challenges

### Data silos

- Systems were built to be silos
- Inefficient data sharing, hindering global services

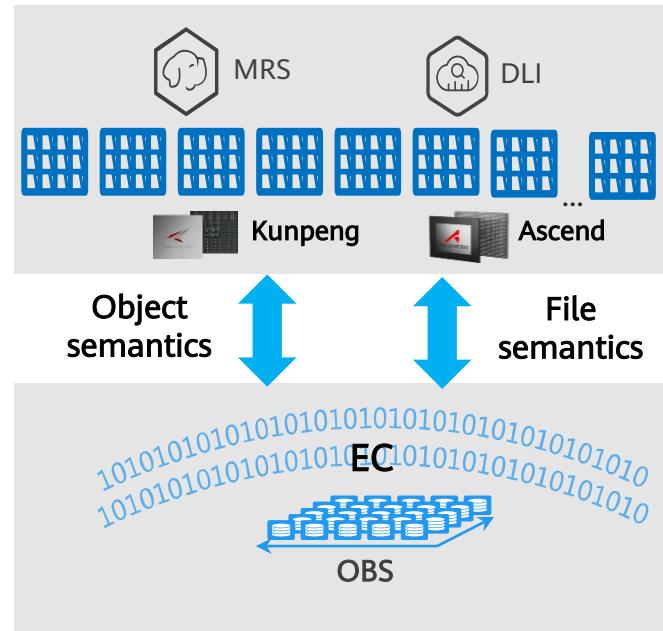
### Rigid capacity expansion

- As long-term storage of data is mandated by China's Cyber Security Law, the three-replica policy of HDFS is faced with a huge cost pressure.
- IT capacities must be flexibly scaled to meet changing demands, e.g. peaks and troughs.

### Low utilization

- Unbalanced storage and compute resources, low efficiency
- Data is kept for long-term auditing and occasional access, compute resource utilization < 20%

## Compute-Storage Decoupling



## Customer Benefits

### Unified storage, no silos

- Raw data all stored in OBS
- Multi-architecture computing and data interaction supported by multi protocols of OBS

### Elastic scalability, higher resource utilization

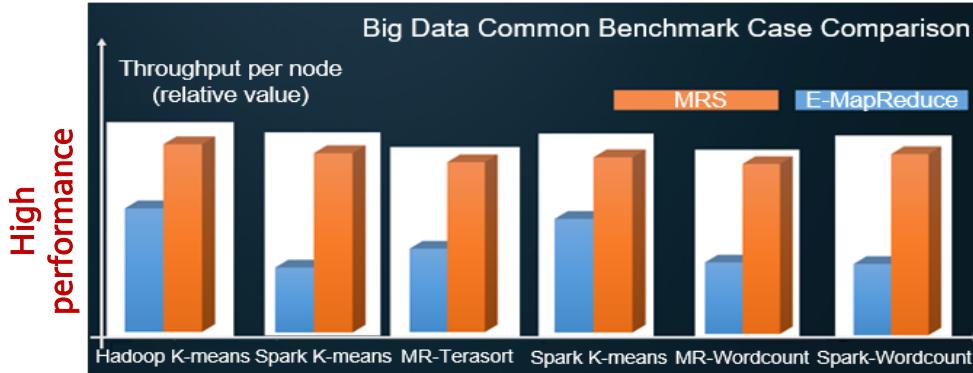
- Compute and storage decoupled and scaled separately
- Compute resource utilization can reach 75%

### Guaranteed performance with storage-compute decoupling + Kunpeng

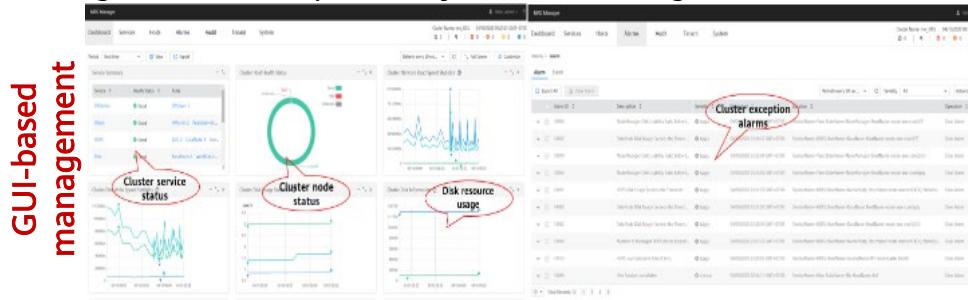
- Kunpeng multi-core optimization, software cache layer, and OBS high-concurrency support
- Doubled cost-efficiency

# Advantages of HUAWEI CLOUD MRS

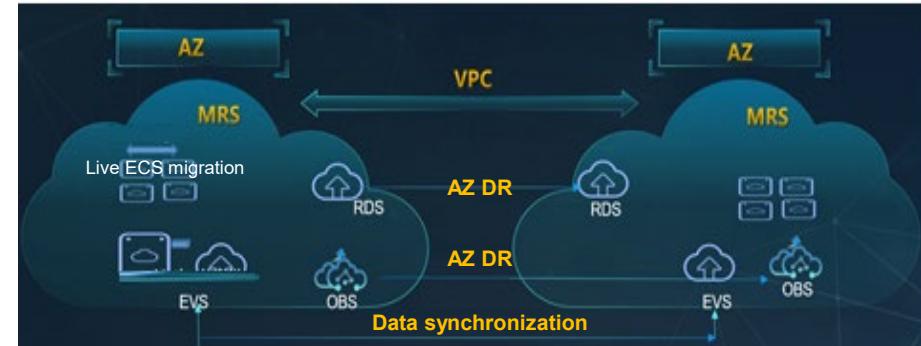
- 100% compatible with open source software, enhanced cost-effectiveness and enterprise-level capabilities



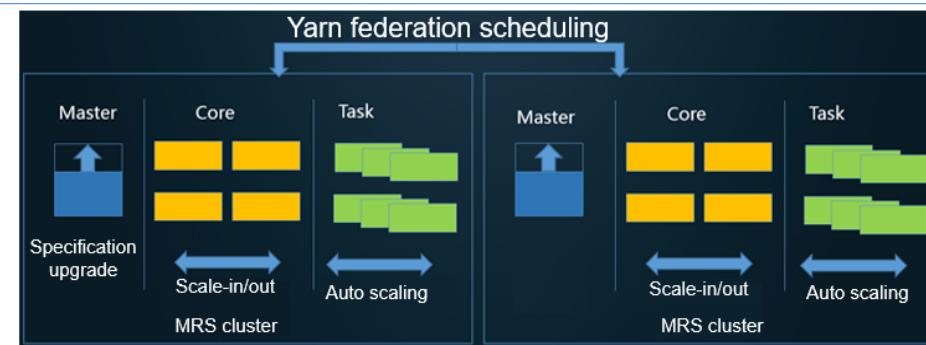
- Advantages of multiple models + Kunpeng
- Software optimization: Spark CBO, CarbonData, and HBase secondary index and tag index, and optimized compression algorithm are improved by 50% on average.



- One-click deployment/capacity expansion; configuration, status management, and alarm monitoring all on a web portal
- Seamless interconnection with legacy management systems via standard SNMP, FTP, and Syslog APIs



- Anti-affinity, live migration of ECSS, and cross-AZ data synchronization and backup
- Storage-compute decoupling (cross-AZ OBS DR and cross-AZ compute resource deployment). External Relational Database Service (RDS) for storing metadata, and cross-AZ reliability

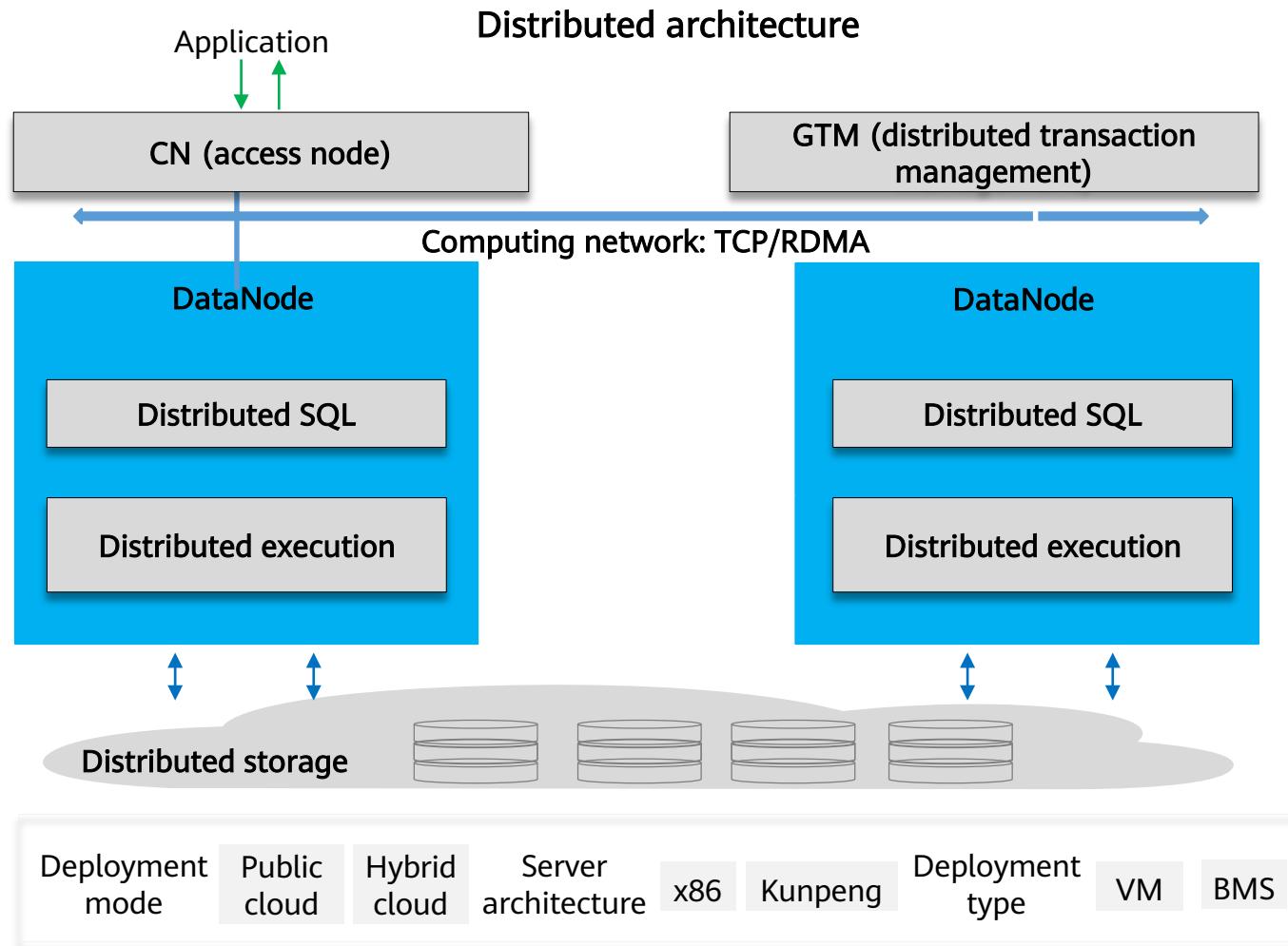


- The smallest cluster has only two nodes (4 vCPUs and 8 GB memory). All nodes can be scaled out. Specification scale-up and node scale-out and scale-in events do not affect services.

High reliability

Auto scaling

# DWS: Cloud-based Data Warehouse Service Based on GaussDB and Fully Compatible with SQL Standards



## Enterprise-Level Distributed Multi-Module Data Warehouse

- Enterprise-level multi-module data warehouse that supports OLAP data analysis and time series flow engine
- Distributed architecture, storage-compute decoupling, and on-demand independent scaling

## Compatible with Standard SQL, Ensuring Transaction ACID

- Compatible with standard SQL 2003
- Ensure transaction ACID and data consistency

## Software and Hardware Collaboration, Improving Performance by 50%

- Support for x86 and Arm servers
- Vertical optimization based on the Kunpeng chip, improving the performance by 50% compared with the same-generation x86 server

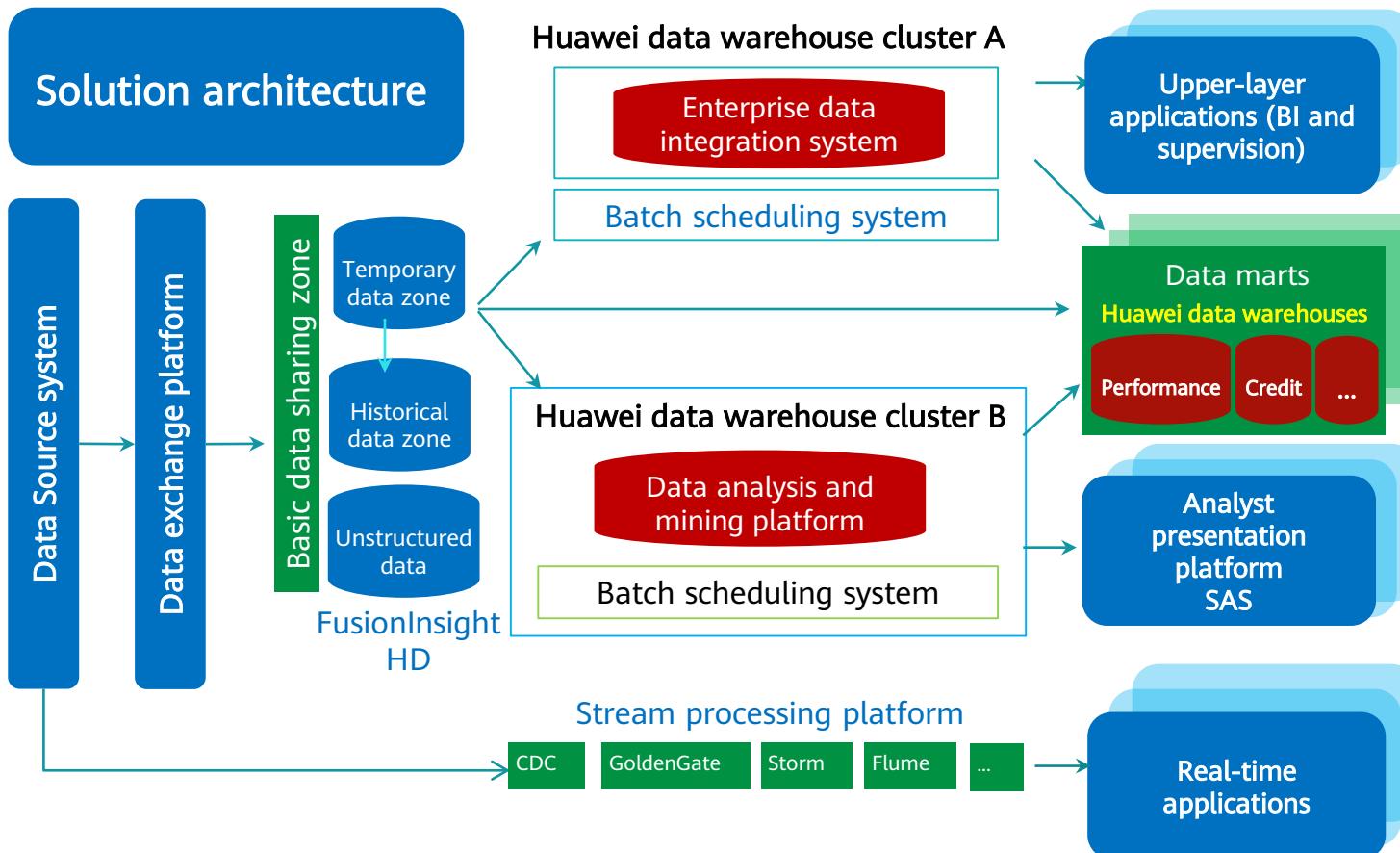
# Core Highlights of DWS

- 
- 01 Performance
    - Fully parallel MPP databases with optimal performance
    - Hybrid row-column storage and vectorized computing
    - High-speed parallel bulk load tool-GDS

**Surpassing similar data warehouse products**
  - 02 Scalability
    - Open architecture, on-demand horizontal expansion, and linear increase of capacity and performance
    - Table-level online capacity expansion, ensuring service continuity
  - 03 Reliability
    - PB-level data, 256 physical nodes, 4,096 logical nodes
    - Multi-layer redundancy, preventing single points of failure in the system
    - Multi-active compute nodes, higher concurrency, reliability, and scalability
    - OBS backup and cross-AZ DR
  - 04 Feasibility
    - Visualized one-stop cluster management, simplified O&M
    - System interoperability and compliance with standard SQL
    - Complete application development and migration tools, comprehensive cluster monitoring and alarm reporting, and automatic incremental backup
  - 05 Security
    - Support for more than 20,000 EDW jobs of a bank
    - Deep integration with Database Security Service (DBSS)
    - HA design and transparent data encryption, ensuring high reliability of data and systems

# Case: PB-Level Financial Data Warehouse

- DWS fully replaces Teradata and Oracle Exadata of bank G, implementing cloud-based transformation on its IT system.
  - Huawei data warehouse has **more than 240 nodes in a single cluster**. There are **more than 700 nodes in 8 clusters** of the bank's data warehouse. The total data volume is about **10 PB**.



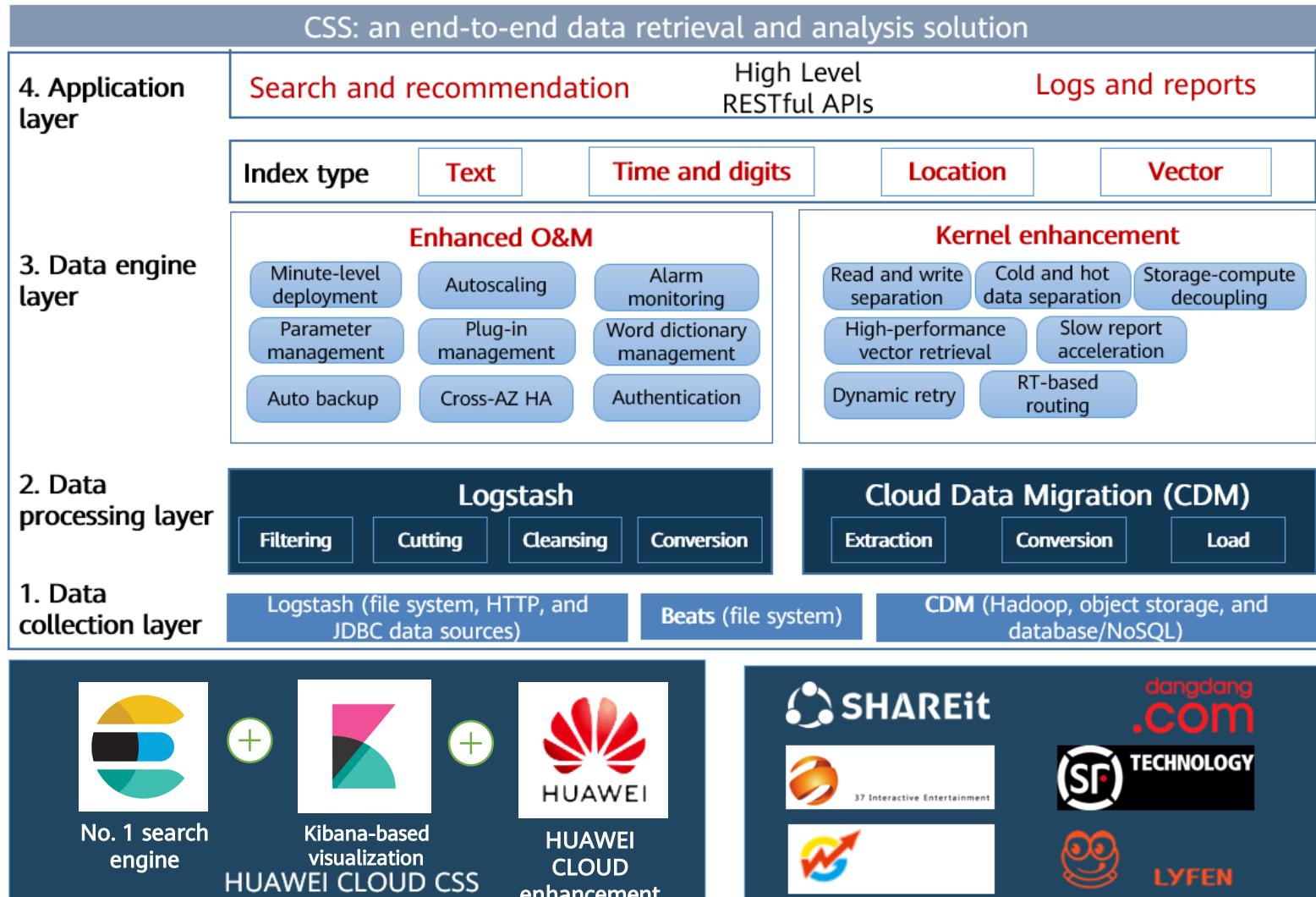
## Customer Challenges

- High scaling cost: The scaling cost of the traditional data warehouse appliance is high, and the downtime is 3 to 7 days, affecting service continuity.
  - Poor timeliness: In the Internet finance era, data sharply expands but analysis performance continuously decreases.
  - Poor usability: There is no solution that integrates data warehouse and big data analytics.

## Customer Benefits

- On-demand scaling: DWS is based on the shared-nothing architecture and can be **quickly scaled on demand**.
  - High performance: Supports high-performance batch processing of tens of thousands of jobs. Compared with Teradata, the performance is **improved by 30%**. The performance increases linearly as the number of nodes increases.
  - Convergent big data analytics platform: DWS interconnects with MRS to support self-service analysis of analysts.

# CSS: An End-to-End Data Retrieval and Analysis Solution



## Industry Background

- Mainstream information retrieval and log analysis engine in the industry
- DB-Engine index, which is the No.7 database in the world
- DB-Engine index, which is the No.1 search engine in the world

## Application Scenarios

- Log analysis and O&M monitoring
- Search, recommendation, and database acceleration

## Advantages

- High performance:** Provides a vector retrieval engine with 10x higher performance than the open source Elasticsearch, supports automatic rollup of time series data, and improves the aggregation analysis capability by 10x.
- Low cost:** Provides Kunpeng computing power, separation of hot and cold data, storage-compute decoupling, and index lifecycle management solutions, reducing the storage cost of cold data by more than 80%.
- High availability:** Services are not interrupted when the cluster specifications are changed and plug-ins and parameters are changed. Supports automatic data backup and delivers 99.9999% data availability.

# Technical Advantages of CSS

Advantages of CSS: high concurrency and stable, low latency in search and recommendation scenarios; low cost and excellent aggregation performance in log and report scenarios.

## Search and Recommendation

- Read and write separation
  - Index file synchronization
- The read and write separation architecture is used to reduce the interference of write operations on read operations. Indexes are synchronized based on index files to prevent index re-creation for copies.

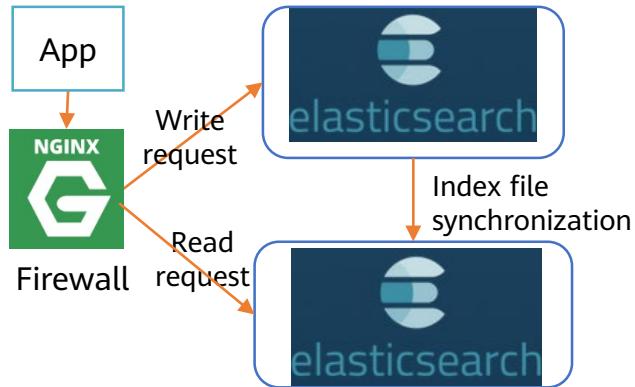
After optimization:

- Write load down by 50%
- Latency-based routing
- Dynamic retry

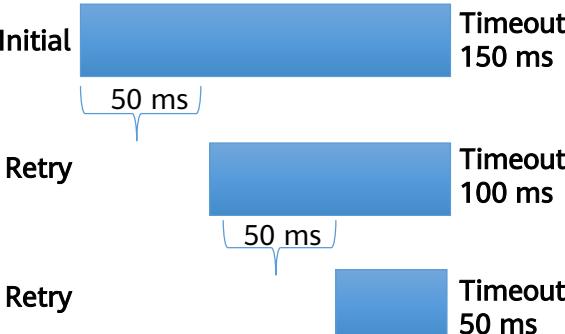
Latency-based routing is used to send requests to nodes with lighter loads. The active timeout interval is set, and automatic retry is performed after the timeout interval expires.

After optimization:

- Latency reduced by 30%
- P99 reduced by 20%, more stable



## Query Timeout + Retry

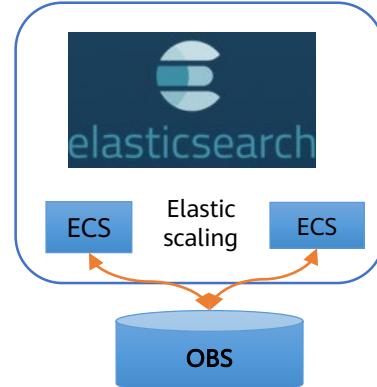


## Logs and Reports

- Storage-compute decoupling

Storage and computing resources are decoupled. Computing resources are used on demand. Storage resources are shared and inexpensive. Data migration is not involved during elastic scaling.

After optimization:

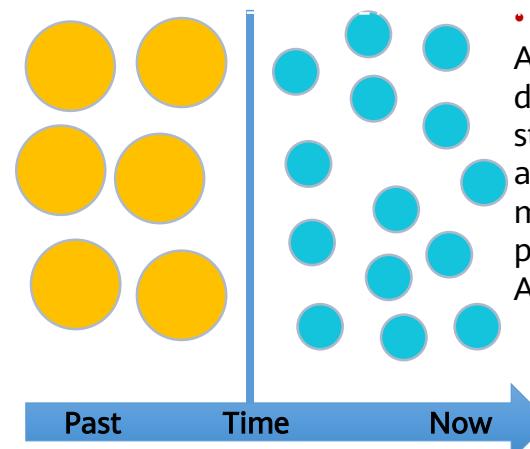


Cost of existing data reduced by 80%

- Automatic roll-up

Aggregates fine-grained detailed data into large-granularity statistics in advance to accelerate aggregation query without modifying existing service programs.

After optimization:



Report performance improved by 100 times

# Case: Internet Customer Video Recommendation

CSS is used to filter recommendation results, greatly improving performance and stability.

The architecture diagram illustrates the flow of data. An 'App' sends a 'Write request' to a 'Search' section, which includes an 'NGINX Firewall'. This leads to a 'Cluster writing' section containing two 'elasticsearch' clusters, each with two 'ECS' nodes and 'Disk' storage. An 'Index file synchronization' arrow points from the 'Cluster writing' section to a 'Cluster reading' section, which also contains two 'elasticsearch' clusters with two 'ECS' nodes and 'Disk' storage.

**Query Timeout + Retry**

Initial	Timeout: 150 ms	50 ms
Retry	Timeout: 100 ms	50 ms
Retry	Timeout: 50 ms	

**Optimization Effect of the Read and Write Synchronization Solution**

A line chart titled 'CPU usage for indexes: 60%' shows CPU usage in 'ms' on the y-axis (0 to 4000) against time on the x-axis. The usage starts at 60% and gradually decreases to 20% over a period of approximately 10 hours. A red arrow points from the initial high usage to the final low usage.

**Challenges:**

- Hundreds of millions of daily active short video recommendation services and thousands of condition combinations for query.
- High concurrency, stable, low latency (P99), and zero service downtime

**Huawei solution: (5,000+ cores in total and 170+ cores in a single cluster)**

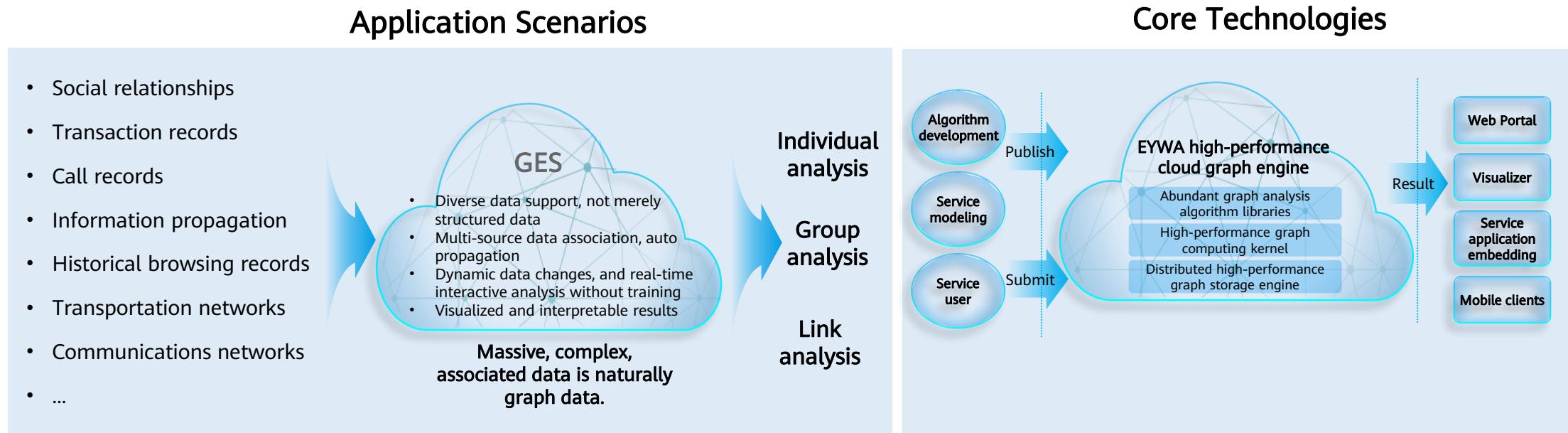
- The solution uses the **read and write separation architecture** to reduce the read and write interference between the CPU and disks.
- The cluster uses the master, data, and client role separation architecture and supports 200 data nodes. One active node and N standby nodes are used to improve service concurrency.
- The active timeout interval is set. After the access times out, the system **proactively retries**. A response is returned immediately after any request is responded. In addition, the firewall **forwards the requests based on the node response latency** to reduce the response delay and P99 jitter.
- Firewalls are deployed at the front-end services to limit the types and size of data. One active node and N standby nodes are deployed to improve cluster availability.

**Customer benefits:**

- The **peak QPS of multi-condition search reaches 8,000+**, and services are running stably.
- The average service response **latency is 40 ms**, and **P99 is within 120 ms**. The service **SLA has reached 99.99%+** since the service was brought online.

# Graph Engine Service (GES)

- GES is a hyper-scale integrated graph analysis and query engine that applies to scenarios such as social apps, enterprise relationship analysis applications, logistics distribution, domain-specific knowledge graph, and risk control.
- GES facilitates querying and analysis of graph-structure data based on various relationships. It uses Huawei's in-house, high-performance graph engine EYWA, which is powered by multiple patented technologies.



## Product Advantages

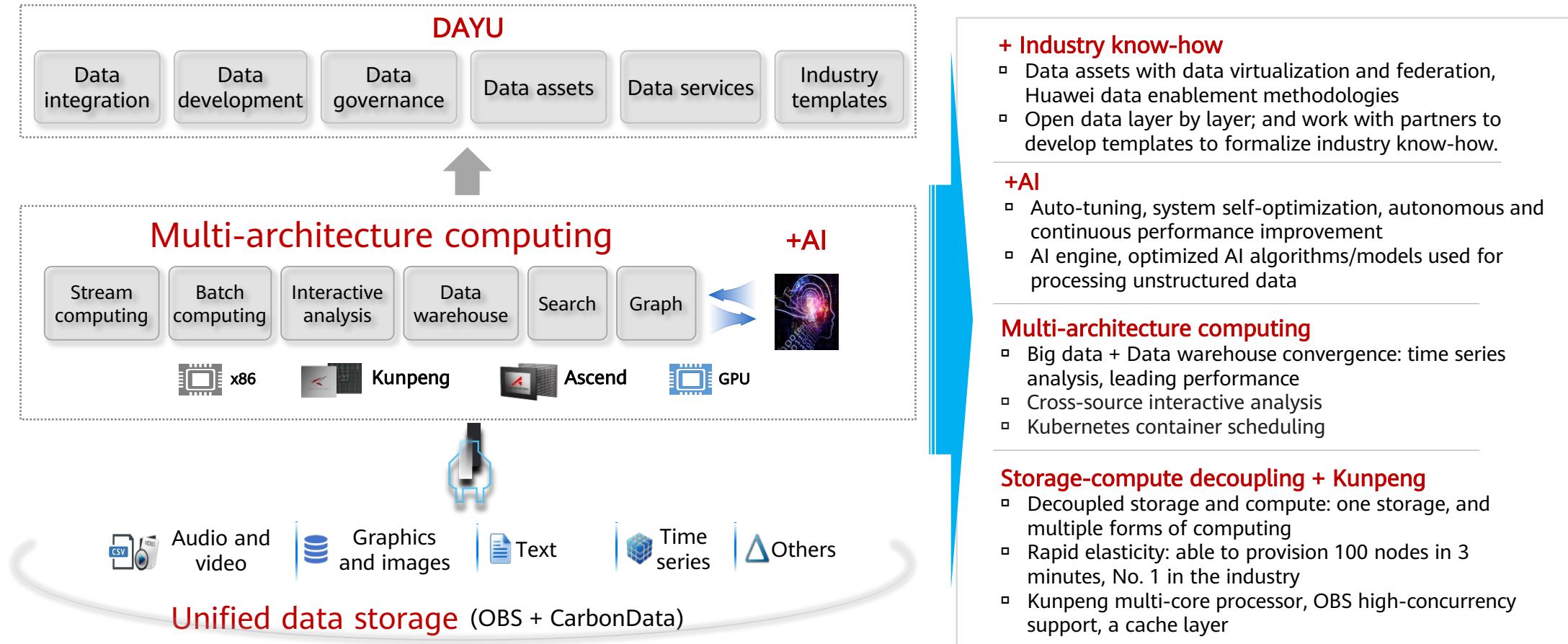


# Contents

1. Development Trend of the ICT Industry
2. HUAWEI CLOUD Big Data Services
- 3. HUAWEI CLOUD Intelligent Data Lake Operation Platform**

# HUAWEI CLOUD Intelligent Data Lake - DAYU

- Understand not only technologies but also data, and implement closed-loop management from data to value



# DAYU - Features of Intelligent Data Lake



## Unified metadata

Metadata is managed in a unified manner. Intelligent connection-based data exploration, automatic data classification, data tagging, and data inheritance are supported.



## Data map

Intelligent data search and display help you quickly find data and comprehensively display data.



## Data asset catalog

Automatically associate technical metadata with business metadata to reveal complex relationships between data.



## Data quality monitoring and improvement

Monitor and improve data quality throughout the lifecycle, and output clean and reliable data.



## E2E data lineage

The E2E data lineage is automatically generated to meet requirements such as data source tracing, impact analysis, and compliance check.



## Unified data standards

Unify metrics, measures, and data models, provide unique data standards, and automatically apply the standards to quality monitoring and improvement.



## Data security management

Data isolation, permission management, and data anonymization ensure data security.

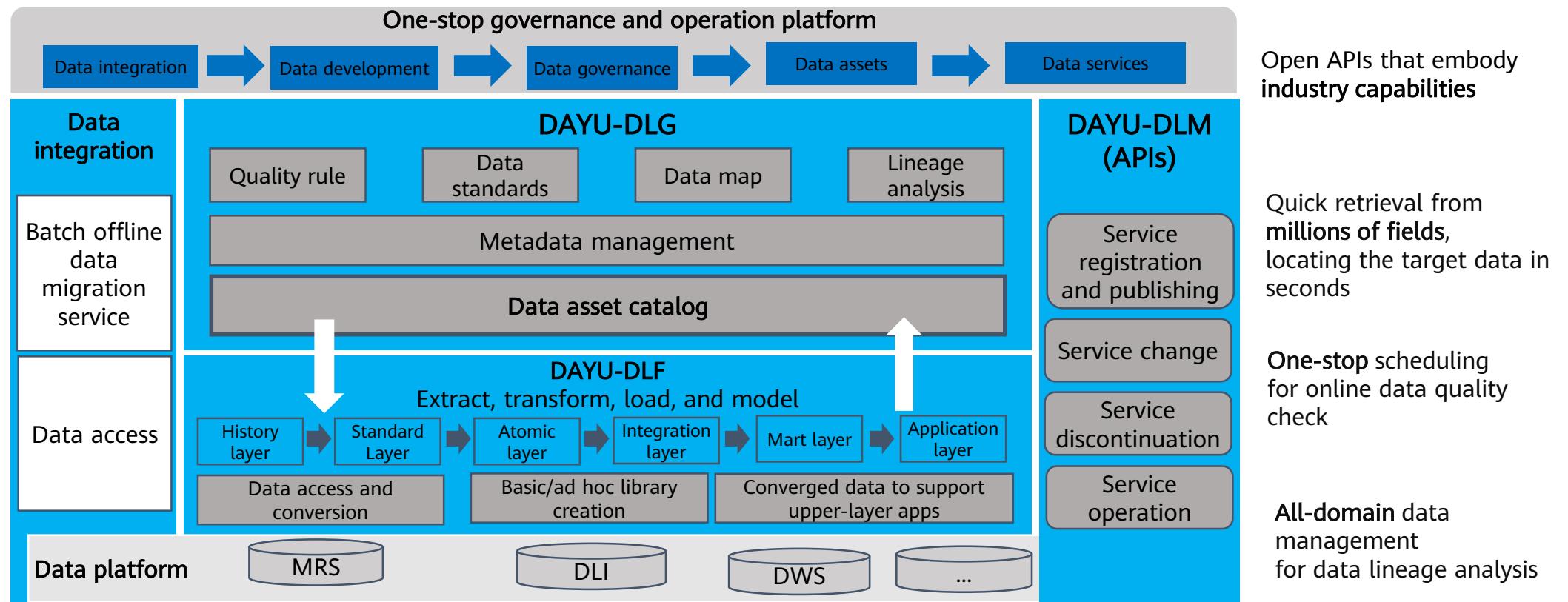


## Periodic task processing and scheduling

Metadata collection and quality tasks can be scheduled and monitored periodically for continuous data value extraction.

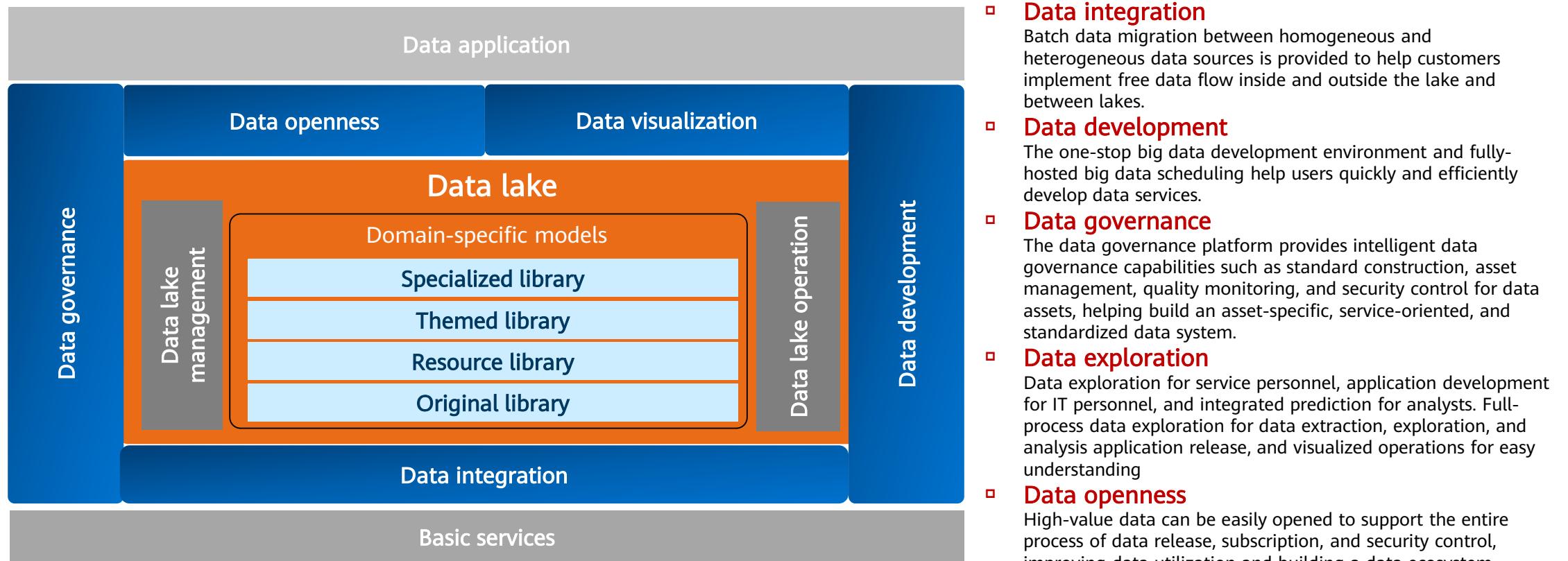
# DAYU - Intelligent Data Lake Operation Platform (1)

- DAYU is a one-stop operation platform that provides data lifecycle management and intelligent data management for customers' digital operations.
- DAYU provides functions such as **data integration**, **data design**, **data development**, **data quality control**, **data asset management**, and **data visualization**, and supports data foundations such as big data storage and big data computing and analysis engine.
- DAYU helps enterprises quickly build an end-to-end intelligent data system from data access to analysis. This system can eliminate data silos, unify data standards, accelerate data monetization, and implement digitization.



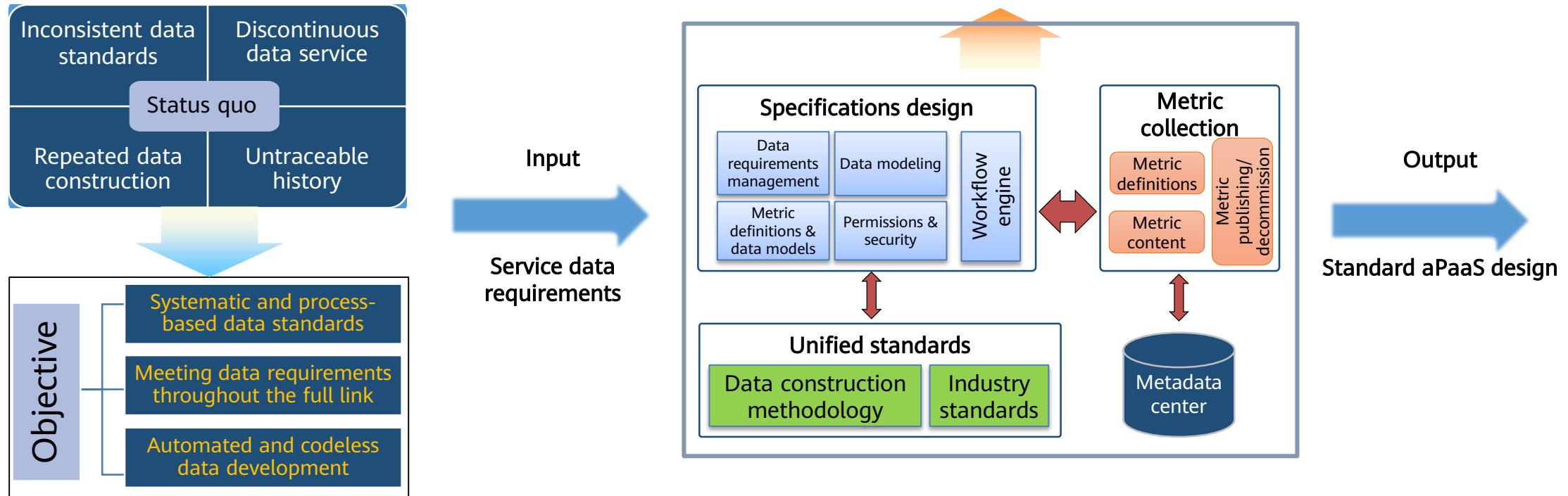
# DAYU - Intelligent Data Lake Operation Platform (2)

- DAYU provides one-stop data asset management, development, exploration, and sharing capabilities based on the enterprise data lake.
- DAYU provides a unified **data integration, governance, and development platform** to seamlessly connect to data foundations such as MRS, DWS, and DLI on HUAWEI CLOUD.
- For government data with complex data sources and various data types, data integration can be implemented without switching multiple tools.



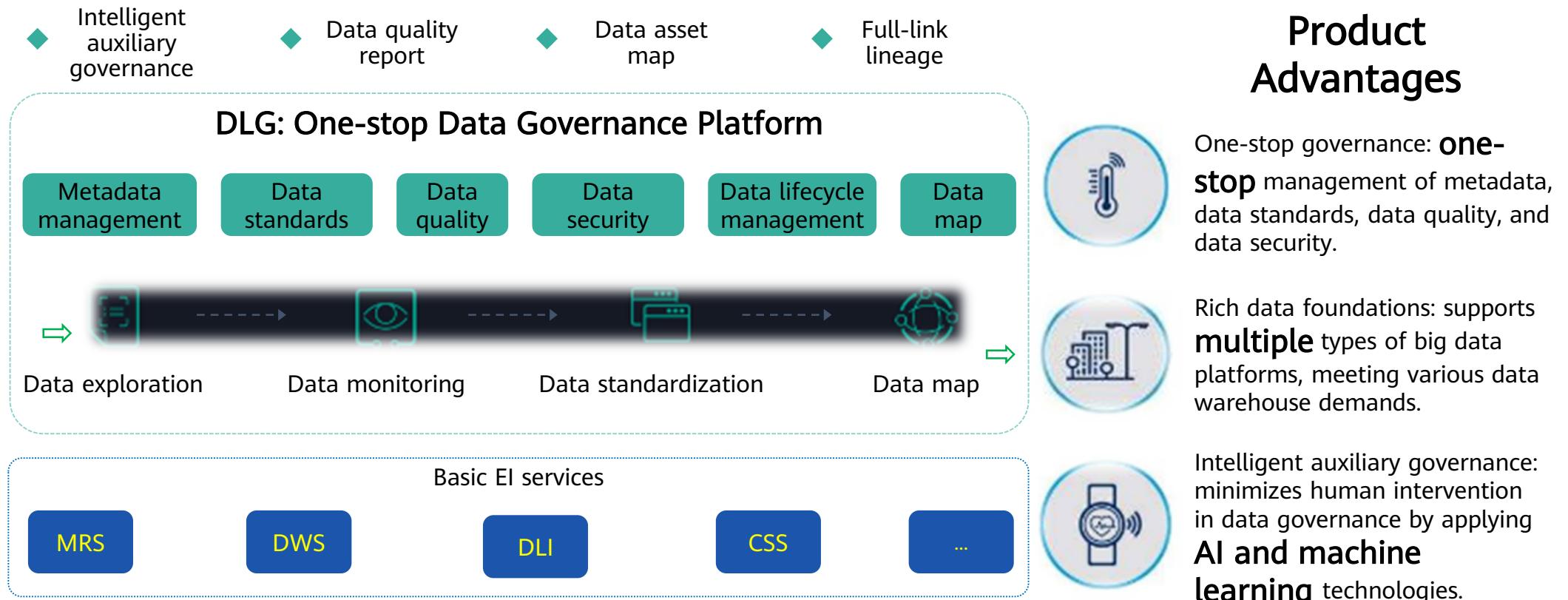
# DAYU - Advantages of Intelligent Data Lake

- GUI-based and simplified operation drawing from Huawei's internal digital operation experience and methodology
- Visualized E2E process, lowering the data governance threshold
- Hierarchical open APIs for self-developed product integration



# DAYU-DLG

- DAYU-DLG data foundation supports various big data core services, such as MRS, DWS, and DLI.
- It offers diverse functions such as unified metadata management, intelligent data analysis, management and implementation of data warehouse standards and business standards, intelligent data quality monitoring, and data quality improvement.



# Summary

- This chapter introduces Huawei big data services and data mid-end services.
- HUAWEI CLOUD Stack 8.0 is a brand-new hybrid cloud solution based on Huawei Kunpeng processors.

# Recommendations

---

- Huawei Cloud Official Web Link:
  - <https://www.huaweicloud.com/intl/en-us/>
- Huawei MRS Documentation:
  - <https://www.huaweicloud.com/intl/en-us/product/mrs.html>
- Huawei TALENT ONLINE:
  - <https://e.huawei.com/en/talent/#/>

# Thank you.

把数字世界带入每个人、每个家庭、  
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and  
organization for a fully connected,  
intelligent world.

Copyright©2020 Huawei Technologies Co., Ltd.  
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.

