

NSIC Big Data Course

Big Data Course

Chapter 1.

Introduction to Big Data

Big Data Course

Chapter Description

Objectives:

- ✓ We will first explore the human journey with data and processing data throughout our history
 - We will gain perspective on how humans have continued to develop new ways to store and process data
 - Major innovations and changes have allowed humans to evolve from agricultural to the information age
- ✓ We are on the cusp of another major paradigm shift and game changer that is once again revolutionizing the way we work with data. At each such junction, humans have been able to catapult to the next evolution through the development of innovative solutions that allows us to harvest the benefits
 - Big Data and the processing of it is set to provide revolutionary gains to those who can harness the difficult to access values hidden within

Contents of Chapter:

1. Big Data Overview and Background
2. Current Trends in Big Data

Unit 1.

Big Data Overview and Background

- | 1.1. Data in human history
- | 1.2. Data is transforming the world
- | 1.3. Why the need for Big Data
- | 1.4. Change of the processing Big Data

What is data?

When did human start using it?

Fact or information, especially when examined and used to find out things or to make decisions

-Oxford Learners Dictionaries-



Journey with data and information

| 150,000 years ago

- ▶ Humans have been keeping "records" and collecting "data" effectively since prehistorical times.

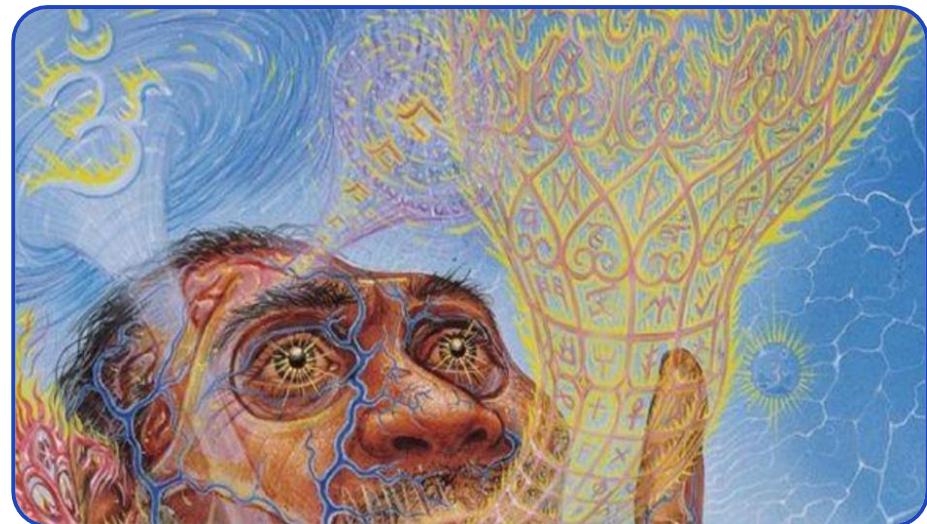
Imagine

Storytelling

Myths

Collective Society

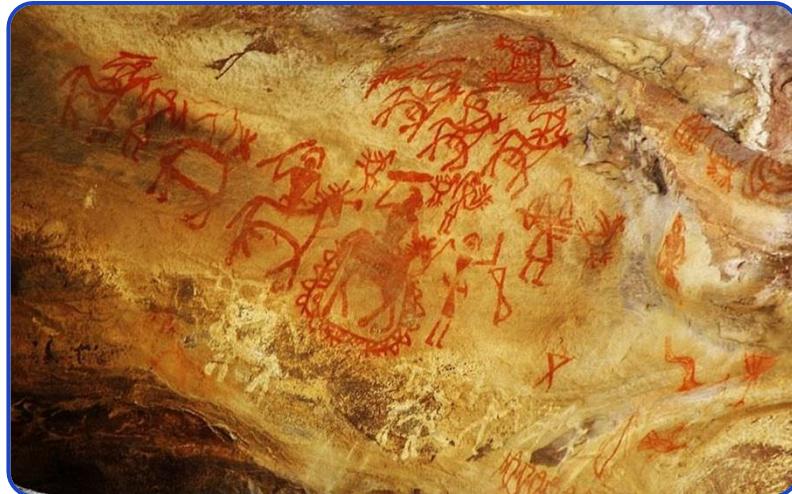
Cooperation



Journey with data and information

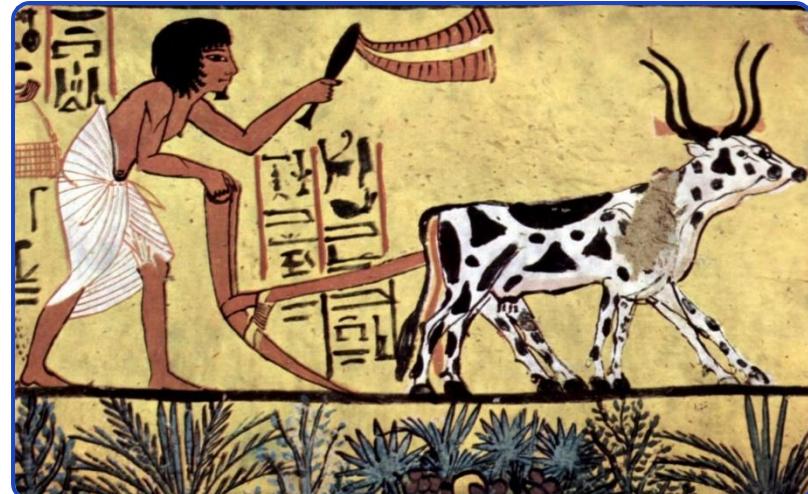
| 150,000 ~ 40,000 years ago

- ▶ Prehistoric Cave Paintings



| 40,000 ~ 12,000 years ago

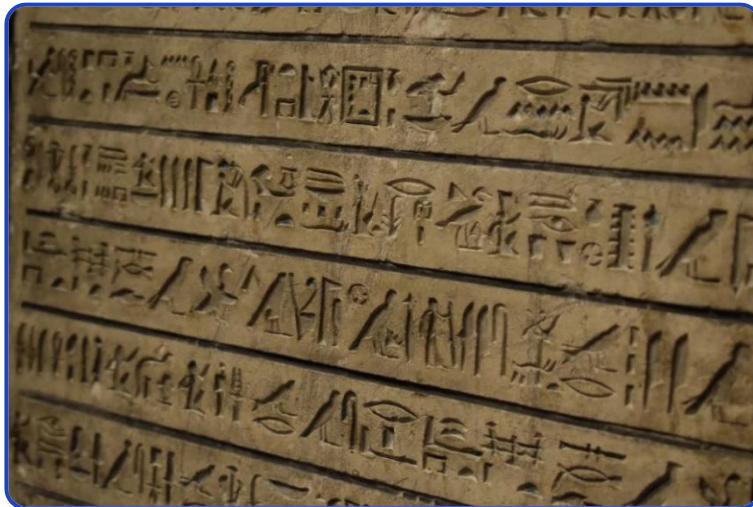
- ▶ Agriculture revolution by Records



Ancient Tools for Record Keeping & Data Processing

| 12,000 ~ 5,000 years ago

▶ Stone Tablets



▶ Ancient Abacus



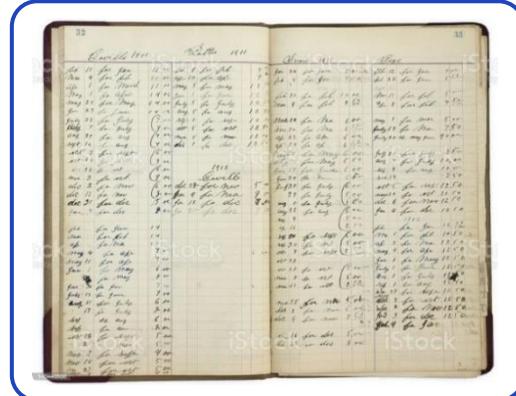
Industrial Revolution

| 250 years ago

▶ Ledgers



▶ Accounting Record

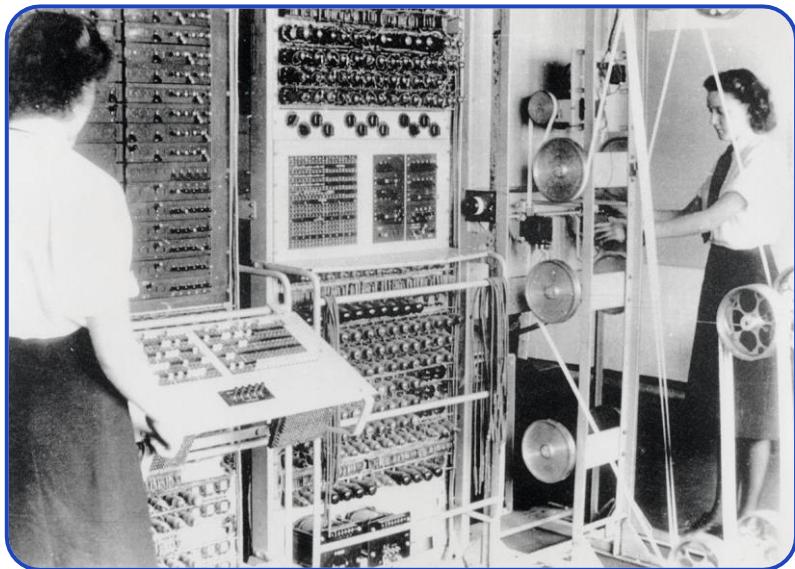


▶ File Record



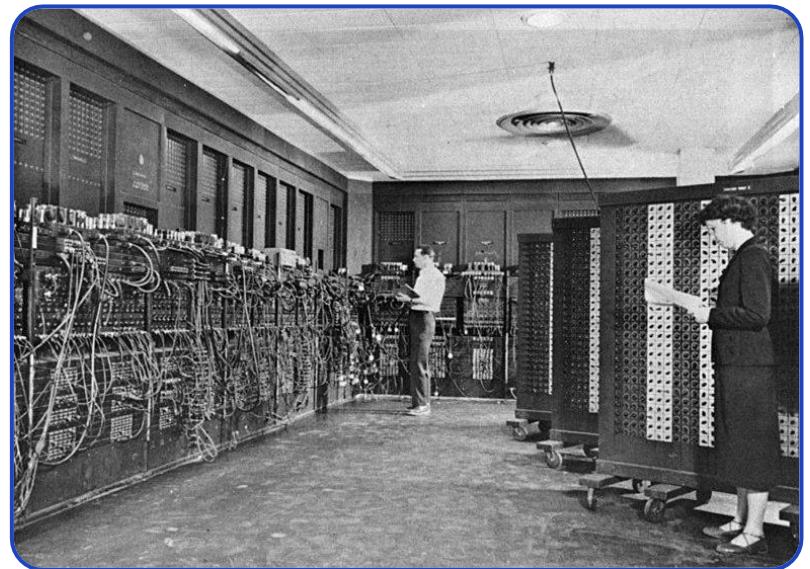
Invention of Computers

| 250 ~ 80 years ago



「Colossus」

The world's first electronic digital programmable computer. It was used to for cryptanalysis.



「Eniac」

The first truly digital computer. It was used to calculate artillery firing tables by US Army.

The Information Age

| Present : We live in a connected World!

- ▶ Data and information technology is driving human progress in all aspect of life.
- ▶ The amount of data being generated is unfathomable.



Why Big Data came

Massive Data Centers

- ▶ Provides lower cost of computing and massive amounts of data for analysis

IoT

- ▶ Generate huge amounts of data
- ▶ Smart phones are everywhere

Cloud Computing

- ▶ Provides a near-infinite amount of scalability
- ▶ Allows for real-time processing of Big Data

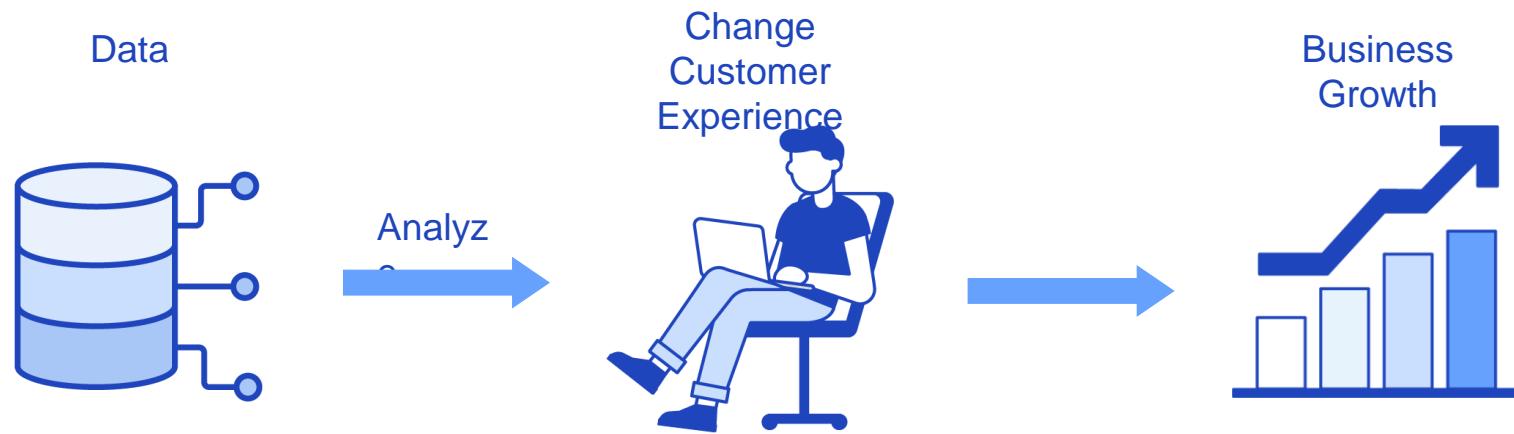


Unit 1.

Big Data Overview and Background

- | 1.1. Data in human history
- | 1.2. Data is transforming the world
- | 1.3. Why the need for Big Data
- | 1.4. Change of the processing Big Data

How is data becoming new fuel?



Big Data Use Cases

Case Study 1



Search
Engine

Case Study 2



e-commerce

Case Study 3

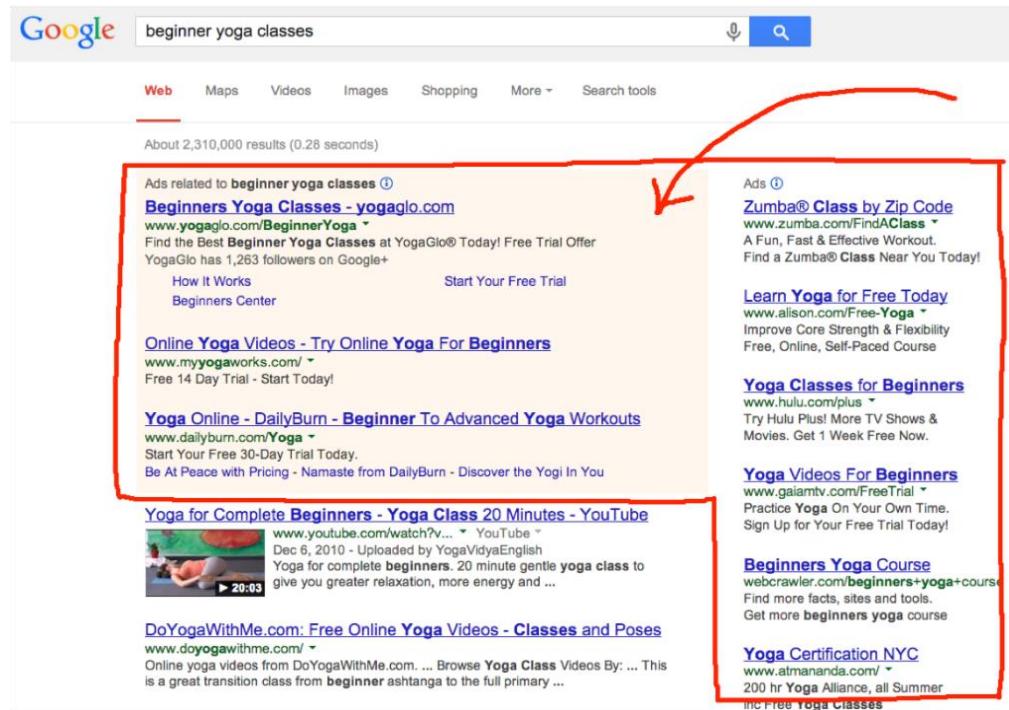


Contents business

[Case study 1] Search Engine

Google Ads

- Allows businesses to reach people as they search for keywords or browse websites with themes related to that business



How much did Google earn?

Quiz



In fiscal 2020, How much did Google earn?

How much did Google earn?

Answer



In fiscal 2020, How much did Google earn?

**181.69 Billion
Dollars**

[Case study 2] E-Commerce

Amazon uses data to sell for your needs

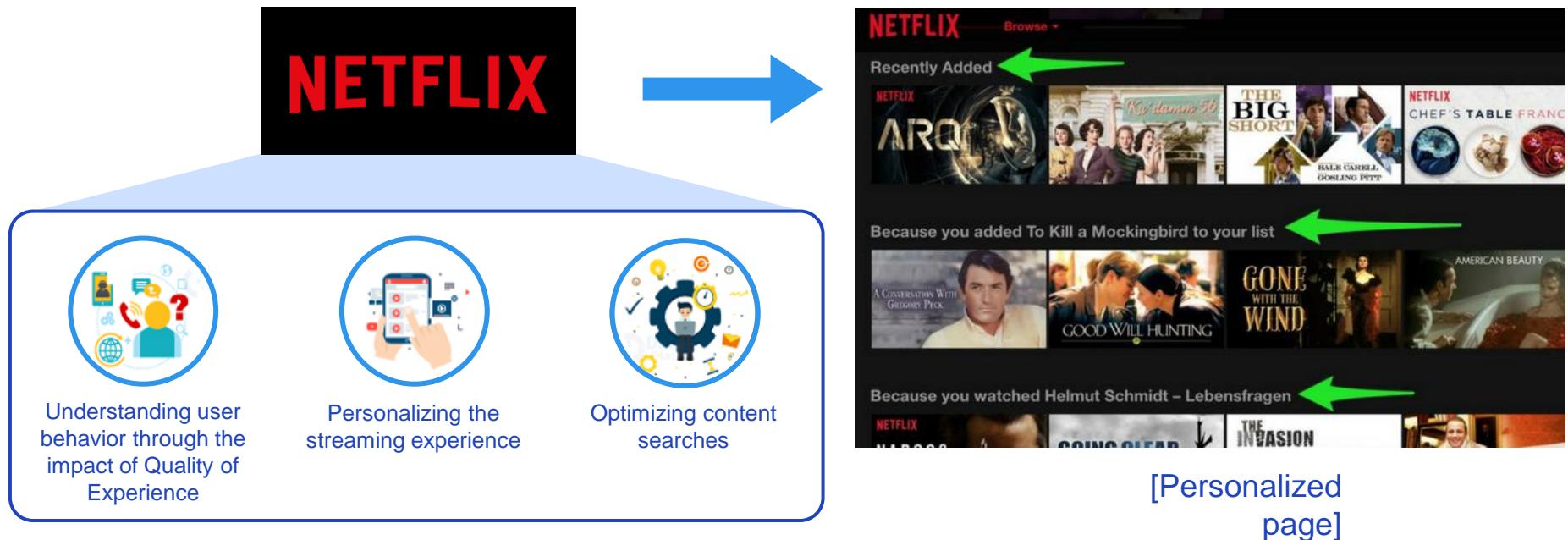
- ▶ Amazon currently uses item-to-item collaborative filtering, which scales to massive data sets and produces high-quality recommendations in real time
- ▶ 35% of sales from product recommendations



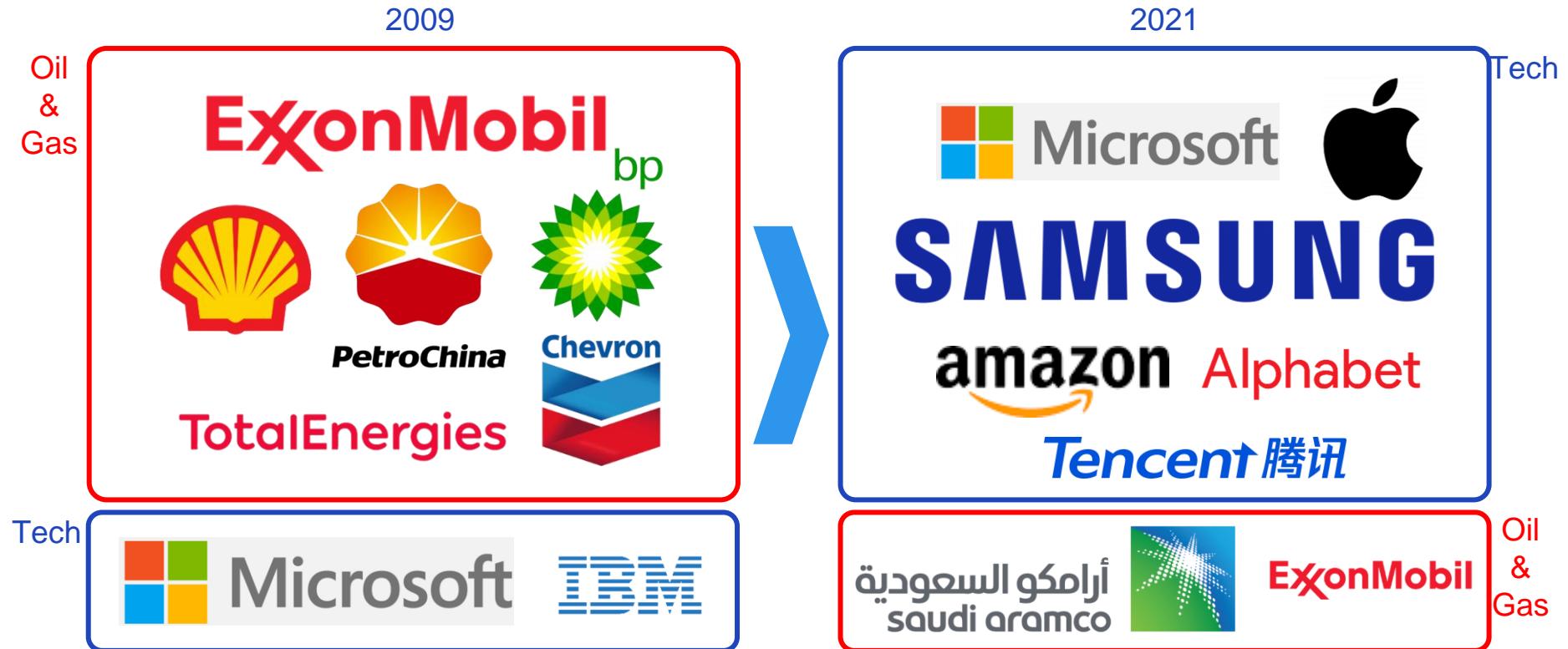
[Case study 3] Contents Business

| Video Recommendations of Netflix

- ▶ 75% of streaming video from recommendations



Changing of the Guard – Market Capitalization



Unit 1.

Big Data Overview and Background

- | 1.1. Data in human history
- | 1.2. Data is transforming the world
- | 1.3. Why the need for Big Data
- | 1.4. Change of the processing Big Data

What is Big Data?

[Discussion
]

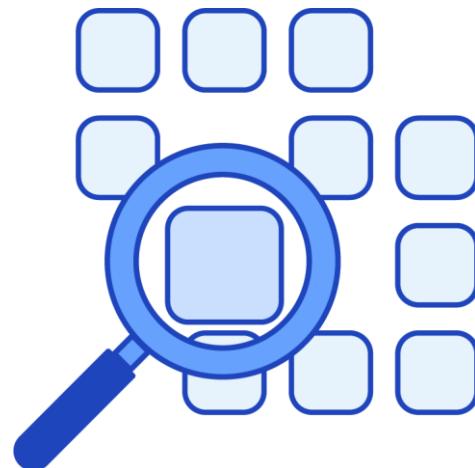
Collect data from SNS,
Blogs, Chats, Web?

A way to predict the
future?

Data Mining?

A tool to help increase sales
and revenue?

A way to better
understand clients?



The meaning of Big data

Extremely large data sets that may be analyzed computationally to reveal patterns, trends, and associations, especially relating to human behavior and interactions.

- Oxford Languages -

Big data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation.

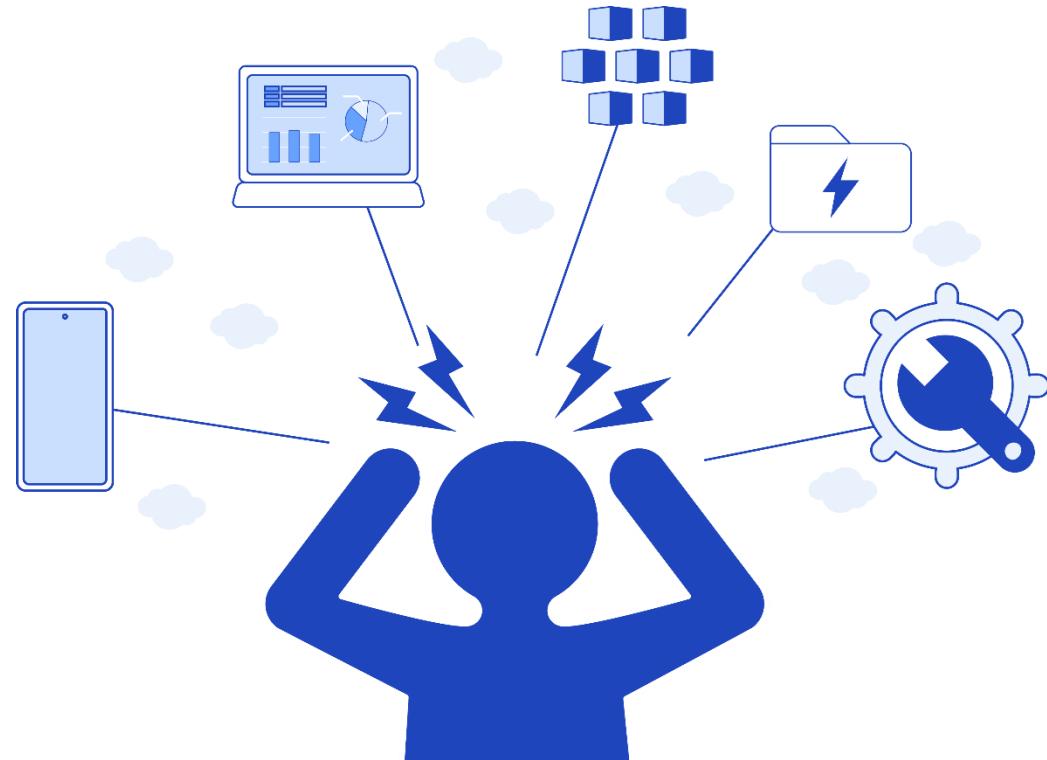
- Gartner Glossary -

Understanding Big Data

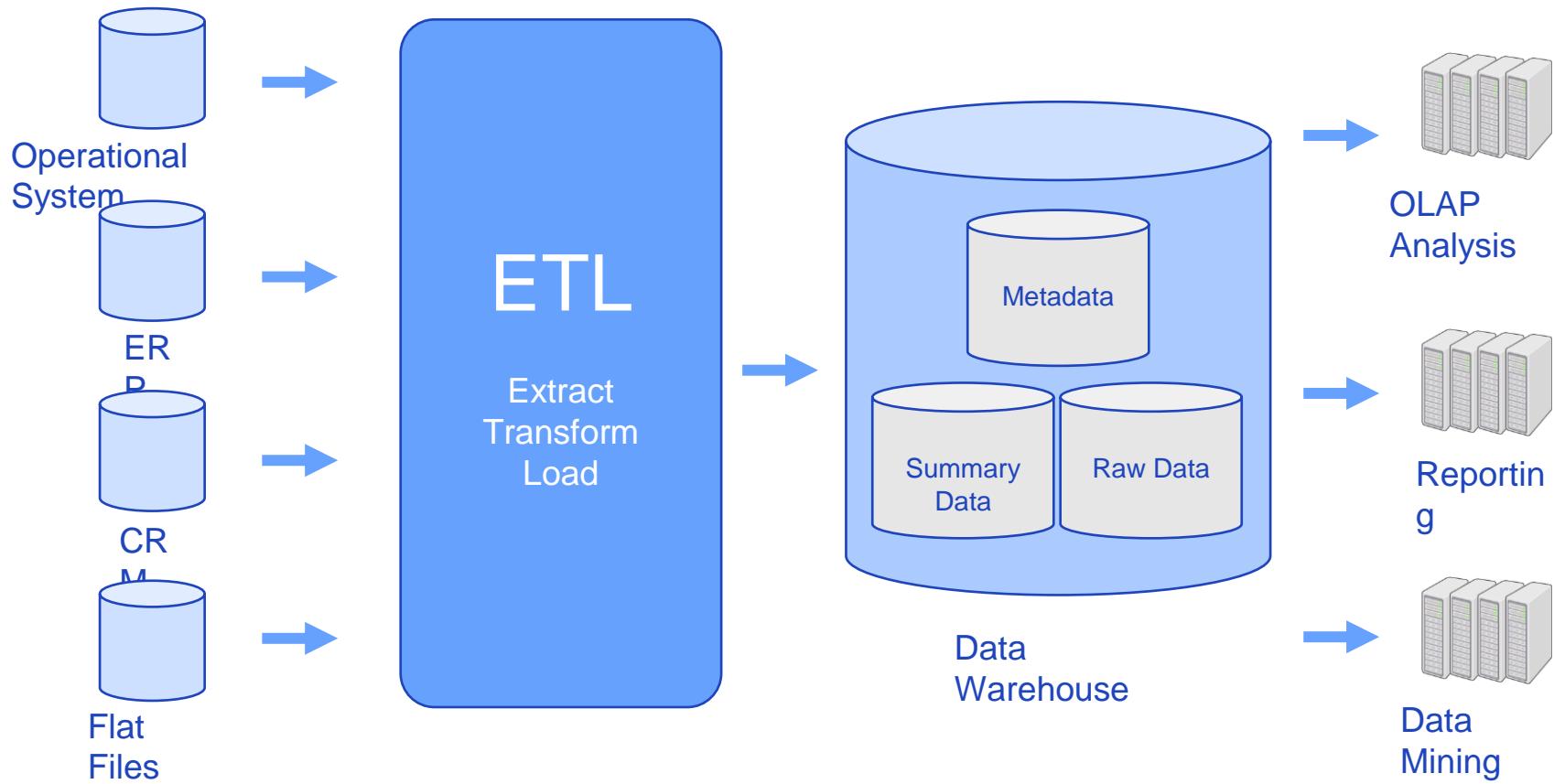
- What is Big Data? ☐ Why Big Data?
 - Trying to answer what Big Data is might be the wrong question
 - The best way to understand what Big Data is to ask- "Why did Big Data come to being?"
 - What pain points is it trying to solve?

What were the Pain Points?

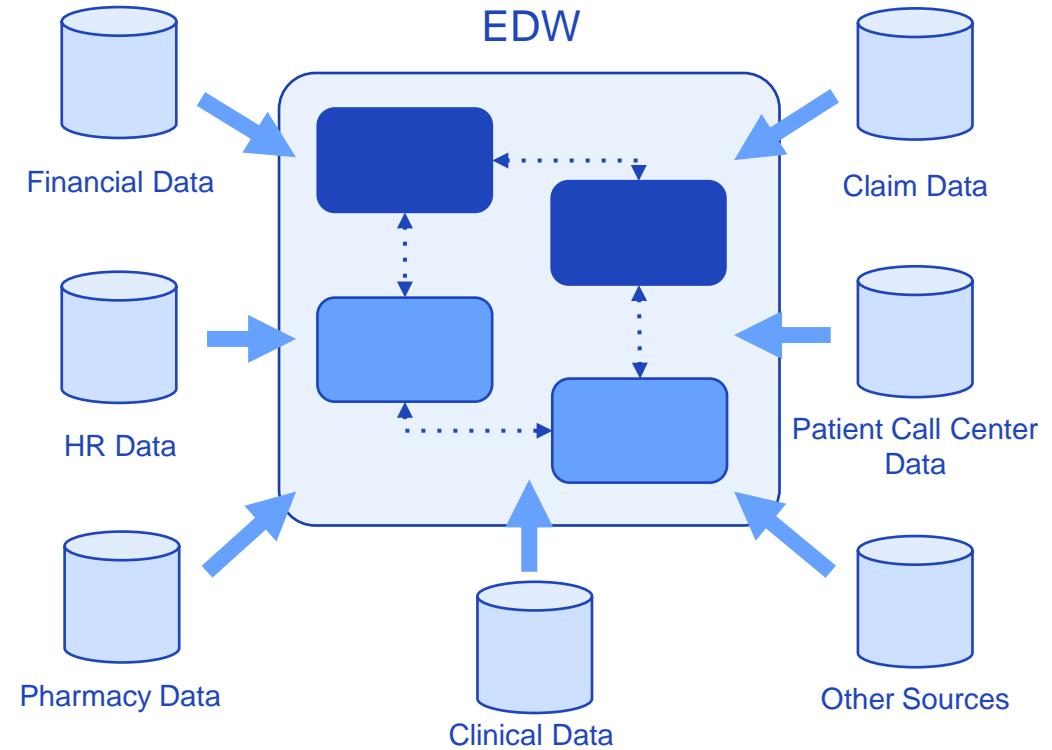
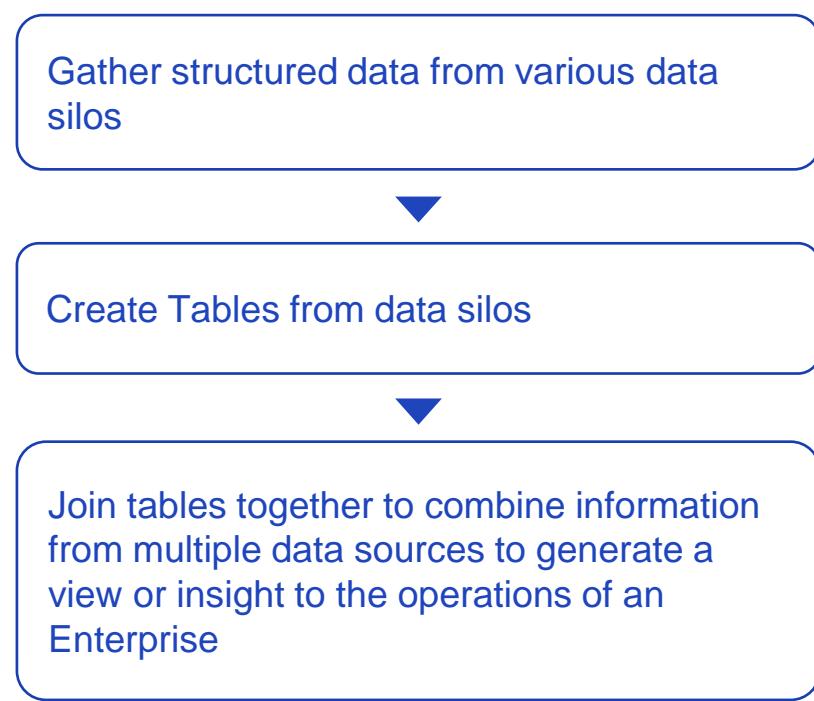
- | Main pain points of data field
 - ▶ Storage Media
 - ▶ Scalability
 - ▶ Network Connectivity
 - ▶ Sub-optimal data transformation
 - ▶ Data Security



Data Processing Before Big Data



Data Silos and Enterprise Data Warehouse



Relational Database Management Systems

Structured data

- ▶ It adheres to a pre-defined data model and is therefore straightforward to analyze.
- ▶ It conforms to a tabular format with relationship between the different rows and columns.



The Dot Com Era Matures

- | The World Wide Web becomes widespread and Internet companies



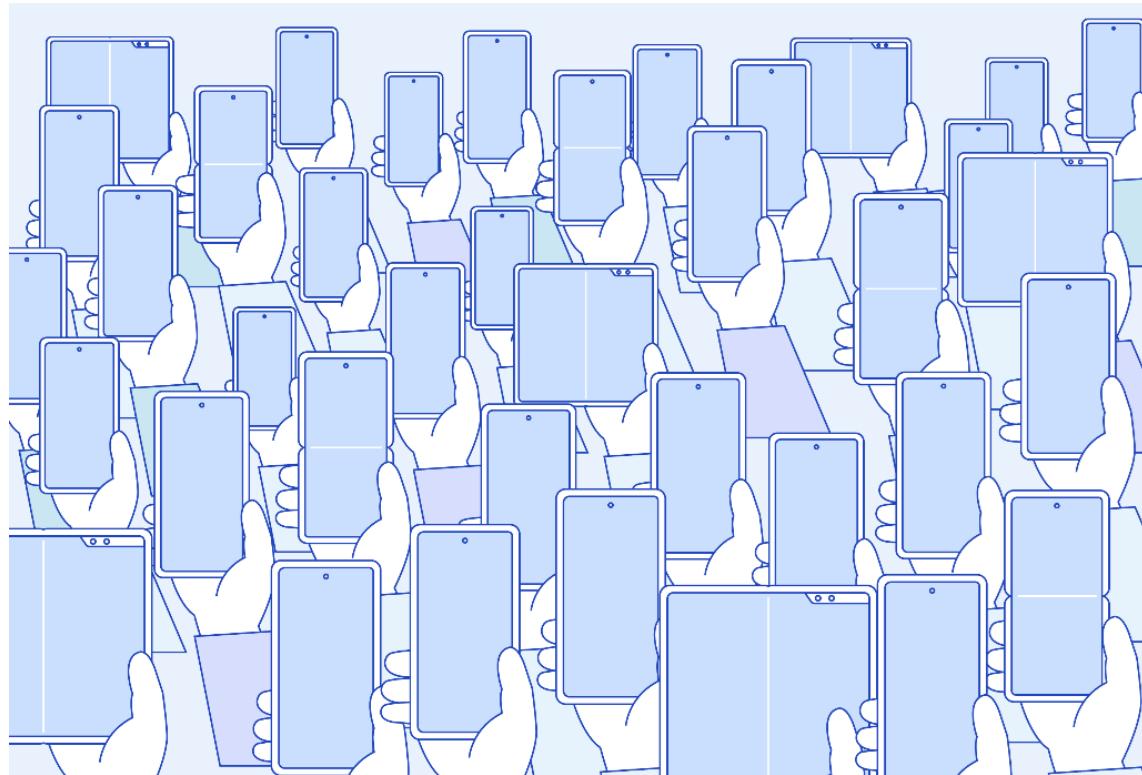
Top Websites make different types of data

- ▶ **Text and Hypertext:** a scrolling box, Hypertext, Synchronized with audio, Transcribed, Searchable tracks, Captions for movies
- ▶ **Video :** standard video, virtual reality panoramas and virtual reality objects
- ▶ **Animation**
- ▶ **Graphics:** 2D Bitmapped Graphics, 2D vector-based graphics, 3D images
- ▶ **Audio:** Digitized Sound, MIDI

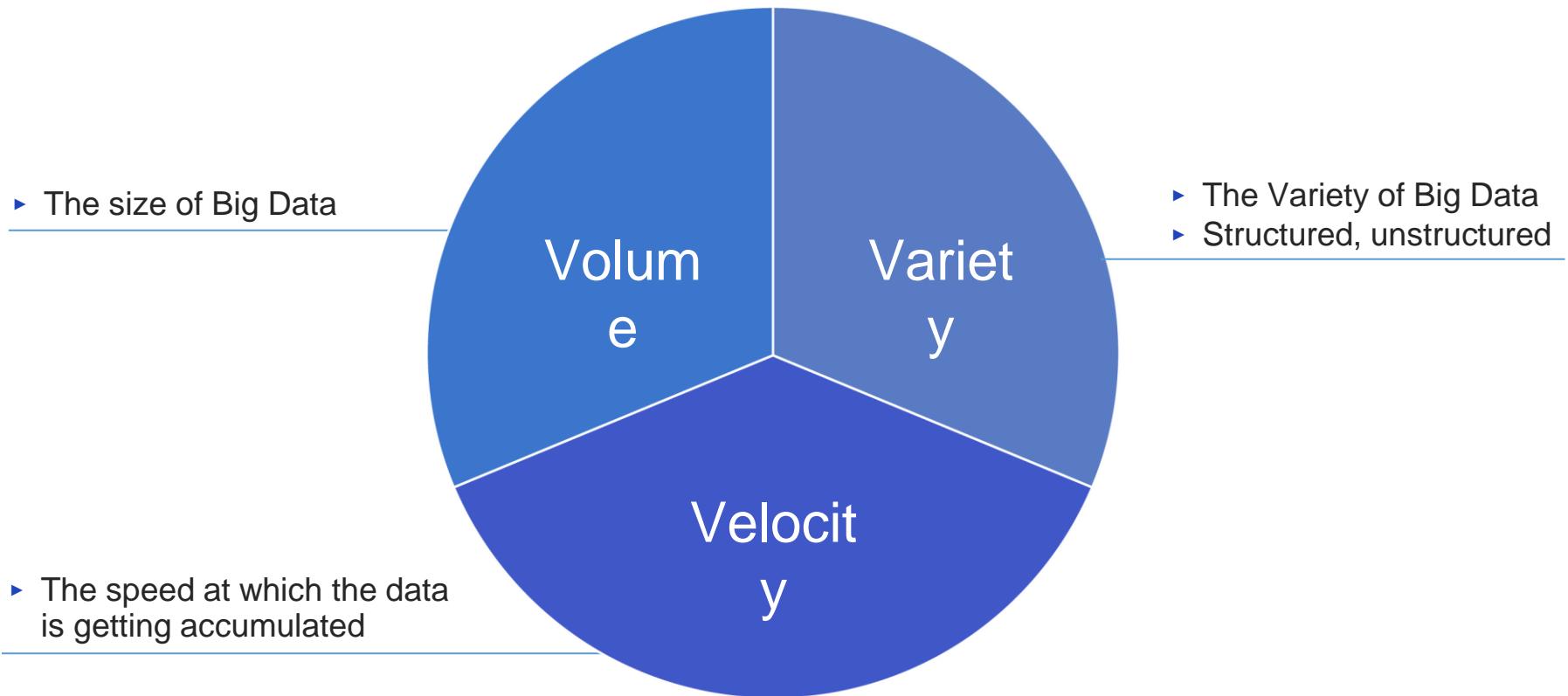


Smartphones Everywhere

| The amount of mobile data is also blowing up



Characteristics of big data



Quantitative Increase of Data - Volume

How Much Data Is Created Every Day

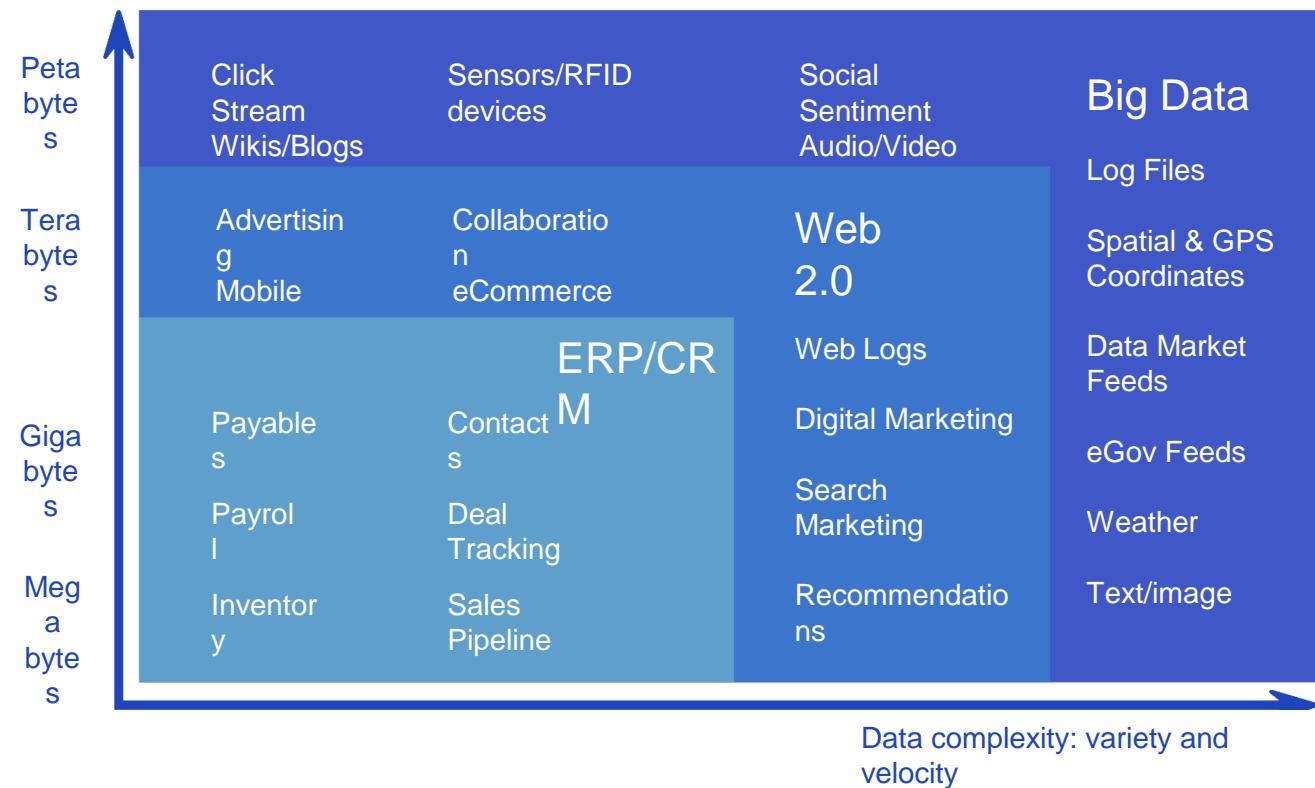
- ▶ In 2020, people created 1.7 MB of data every second.
- ▶ By 2022, 70% of the globe's GDP will have undergone digitization.
- ▶ In 2021, 68% of Instagram users view photos from brands.
- ▶ By 2025, 200+ zettabytes of data will be in cloud storage around the globe.
- ▶ In 2020, users sent around 500,000 Tweets per day.
- ▶ By the end of 2020, 44 zettabytes will make up the entire digital universe.
- ▶ Every day, 306.4 billion emails are sent, and 500 million Tweets are made.

Data is created every day the current estimate stands at 1.145 trillion MB per day.

Changing Data Profile - Variety

Changing Data Profile

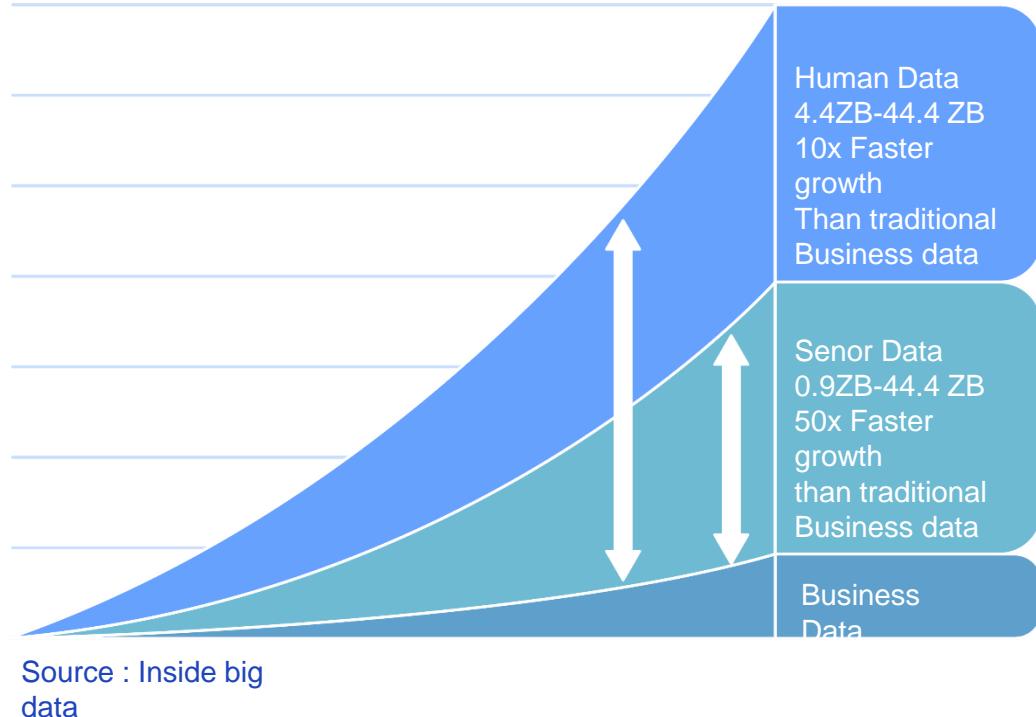
- ▶ Transaction Data
- ▶ Scientific Data
- ▶ Sensor Data
- ▶ Social Media Data
- ▶ Enterprise Data
- ▶ Public Data



Data Deluge - Velocity

Increasing speed of Data Generated

- ▶ Human and machine-generated data is experiencing an overall 10x faster growth rate than traditional business data
- ▶ Machine data is increasing even more rapidly at 50x that growth rate



The pain point is that traditional data processing through RDBMS / EDW was no longer viable

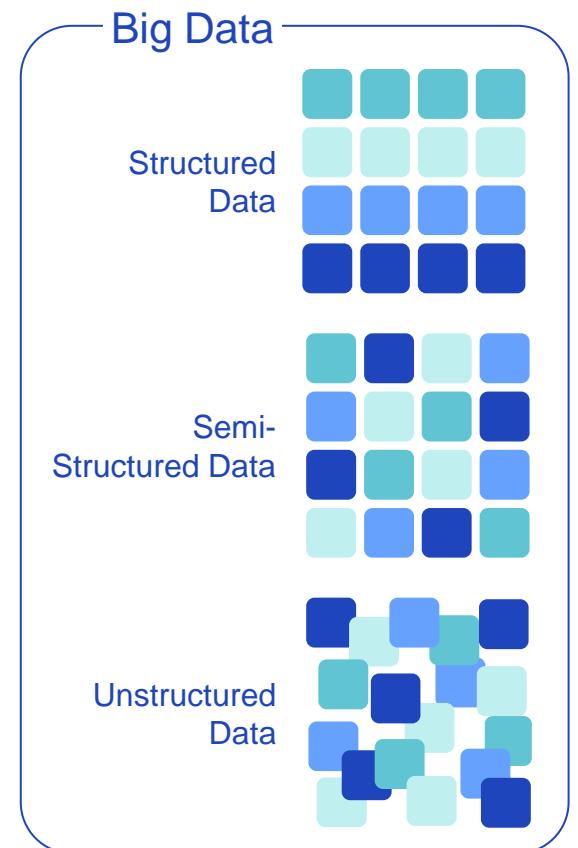
Variety disturbs uniform structure

Structured, unstructured, semi-structured the new norm

- ▶ RDBMS systems unable to process semi-structured and unstructured data
- ▶ Structured Data : Addresses, Dates, Numbers(Phone, Zip Codes, etc.), Text, Most CRM Data
- ▶ Semi-Structured Data : E-Mails, Markup Languages, EDI, XML
- ▶ Unstructured Data

Unstructured data makes up 80% and more of enterprise data

- ▶ Structured data is traditionally easier for Big Data applications to digest, but today's data analytics solutions are making great strides in the unstructured data area



Volume reduces performance

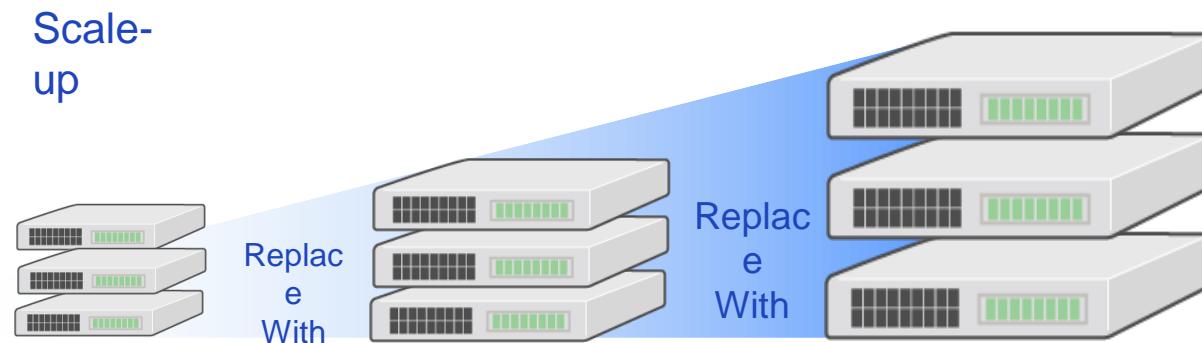
| Performance according to scale

- ▶ RDBMS systems were never designed to handle truly large amounts of data
- ▶ They work best when the size of the data is in the Gigabyte or at most Terabyte range
- ▶ Enterprises were facing Petabytes of data to process
- ▶ Accessing and processing performance gets worse when it comes to Big data

Velocity makes need of Scalability

Scalability

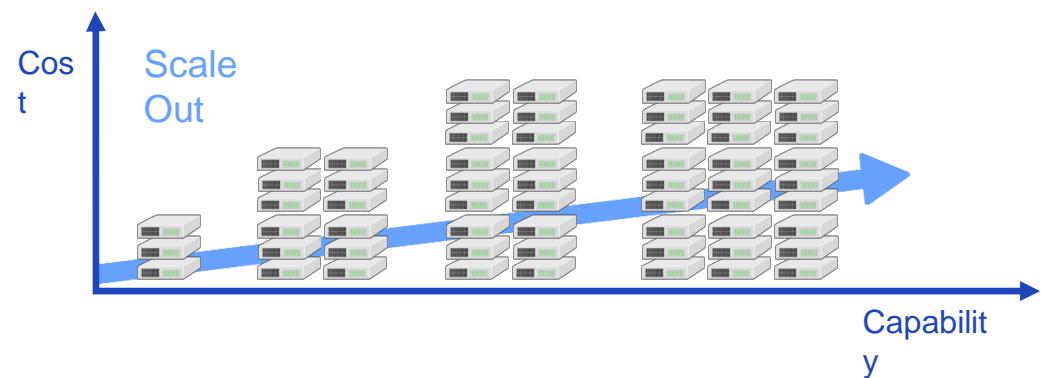
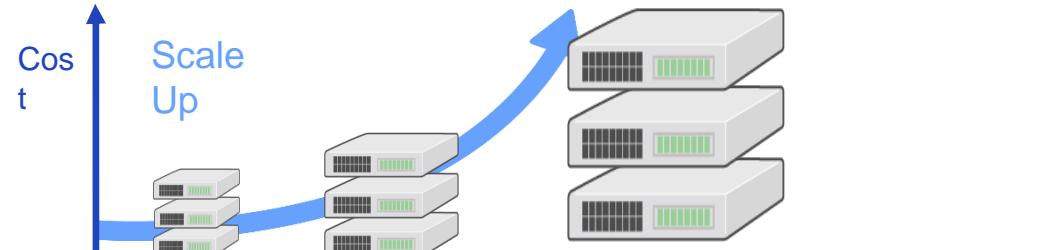
- ▶ To keep up with the growing speed of the data generated, Big Data requires scalable storage systems
- ▶ This was not an issue in traditional EDW processing: typically older and less relevant data is removed to make room for newer data rather than scaling
- ▶ RDBMS is highly reliant on expensive hardware , so it's possible but not suitable to scale.



Velocity generates exponential cost

Cost

- ▶ RDBMS systems are considered to be Scale-Up systems : each generation needs to be replaced with ever more expensive and complicated hardware systems
- ▶ Scale-up system is significantly less cost effective when it comes to the range of 100TB to 1PB of data stored
- ▶ In contrast, scale-out system maintains a constant cost-effectiveness regardless of the size of the data



Unit 1.

Big Data Overview and Background

- | 1.1. Data in human history
- | 1.2. Data is transforming the world
- | 1.3. Why the need for Big Data
- | 1.4. Change of the processing Big Data

Role of a data engineer

| An engineer is like a chef

- ▶ Engineers take current readily available ingredients to produce the best products
- ▶ A product or system often can not support the most expensive ingredients available
- ▶ New ingredients are constantly becoming available or becoming available at reasonable cost
- ▶ Compromises must be made on ingredient choices and then solutions must be engineered to overcome any shortcomings inherent in those choices



How to deal with Big Data

Requirements of Big Data Platform

- ▶ The cost should be reasonable and manageable initially and as the system grows
- ▶ Ability to store and process massive amounts of data (petabytes++) in reasonable time
- ▶ As data deluge is not slowing, be able to grow as storage and process requirements increase in future
- ▶ Data cannot be lost. The system must be fault-tolerant.

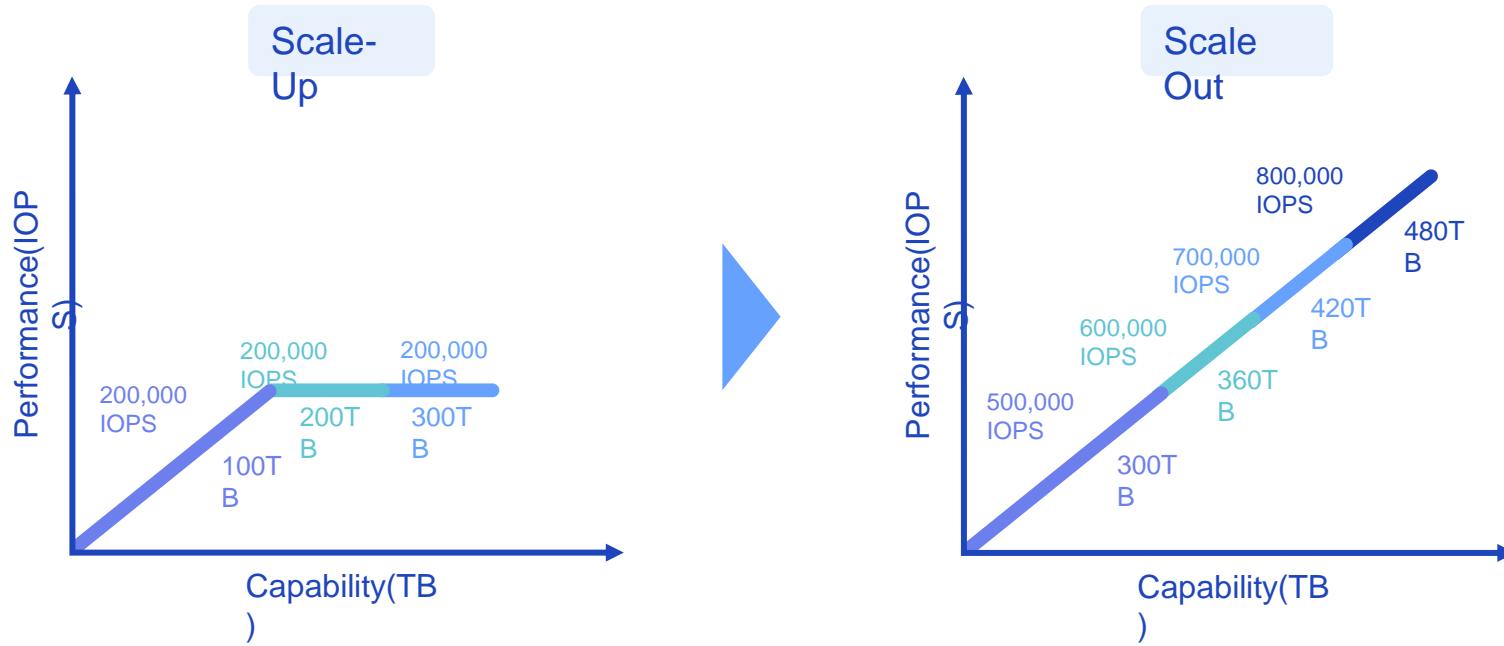
The solution is parallel distributed architecture



Cost Efficiency

Using economical hardware

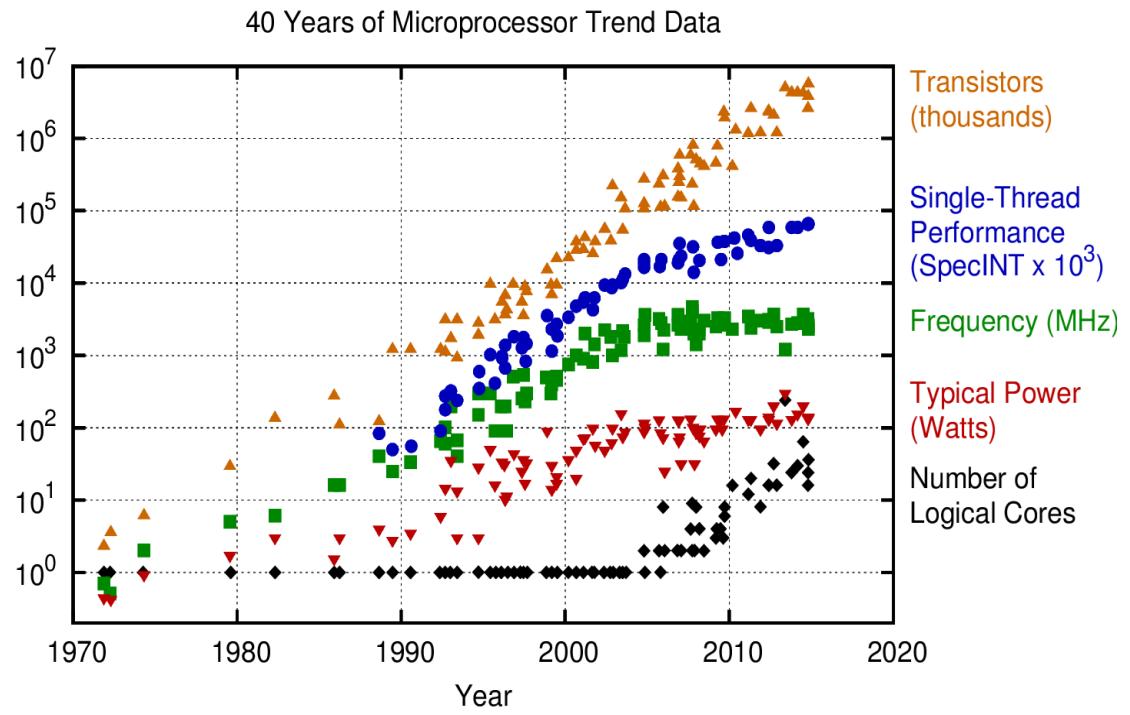
- Transition from Scale-Up to Scale-Out is only way to support storage capacity growth in a cost-effective manner



Performance

Faster and Multi-core Processor

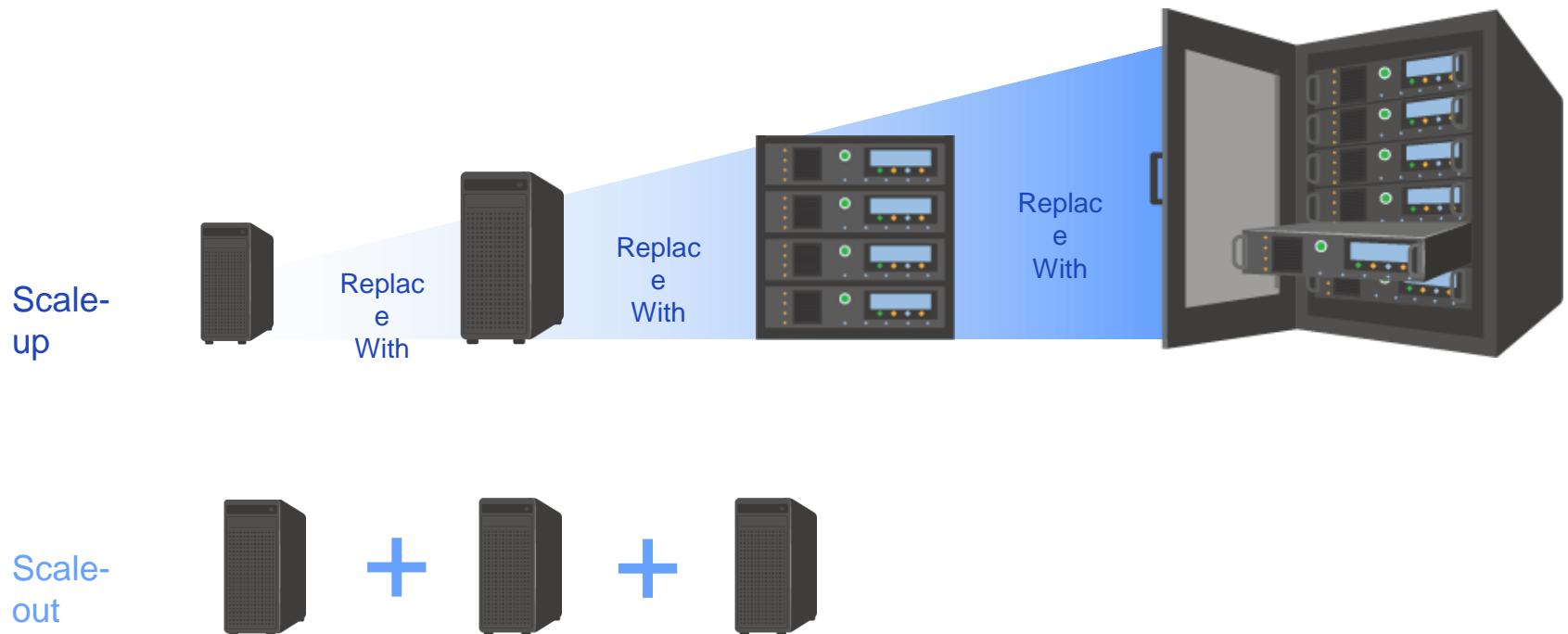
- ▶ Frequency no longer increasing due to physical limitation
- ▶ This relates to the fact that single thread performance stops increasing at the same time
- ▶ As an alternatives, multiple logical cores are introduced in processor



Scalability

[Discussion]
]

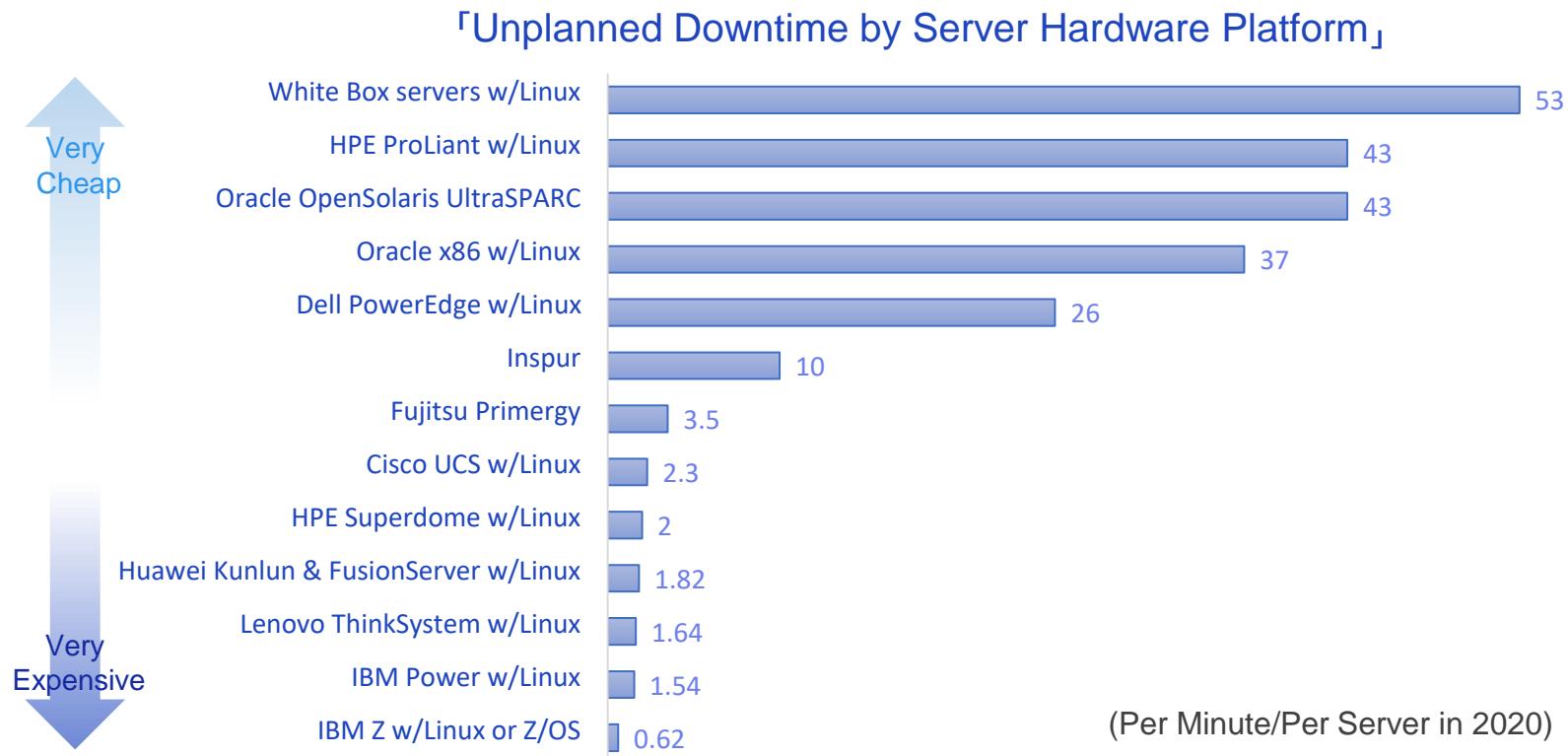
- Why do you think scaling storage moves to Scale-out from Scale-up?



Availability (1/2)

[Discussion]
]

| Which of these servers will you use to construct a Big Data architecture?



Availability (2/2)

| Carrier class servers to Commodity class servers

- ▶ Unplanned downtime by servers can vary dramatically.
- ▶ The lower downtime the more expensive the hardware cost.
- ▶ When using commodity class servers as distributed environment, systems need to be able to accommodate data loss.

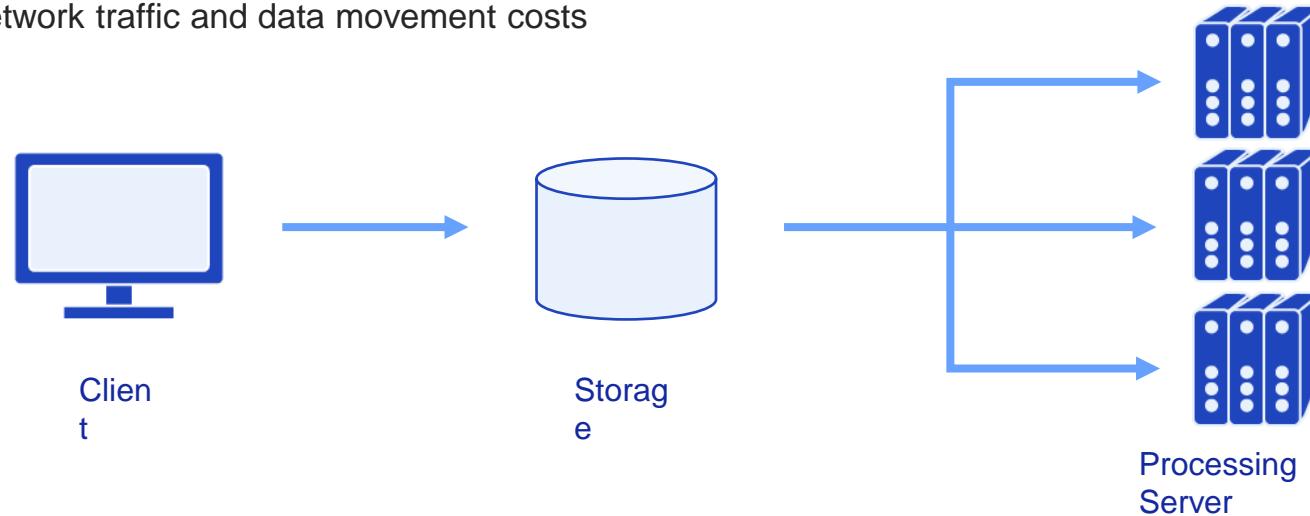
| Client-Server architecture to Parallel Cluster architecture

- ▶ Store the data with the processors
- ▶ Replicate the data for fault-tolerance and increased availability

Client-Server vs. Parallel Cluster of Server (1/3)

| Traditional Architecture – Client-Server

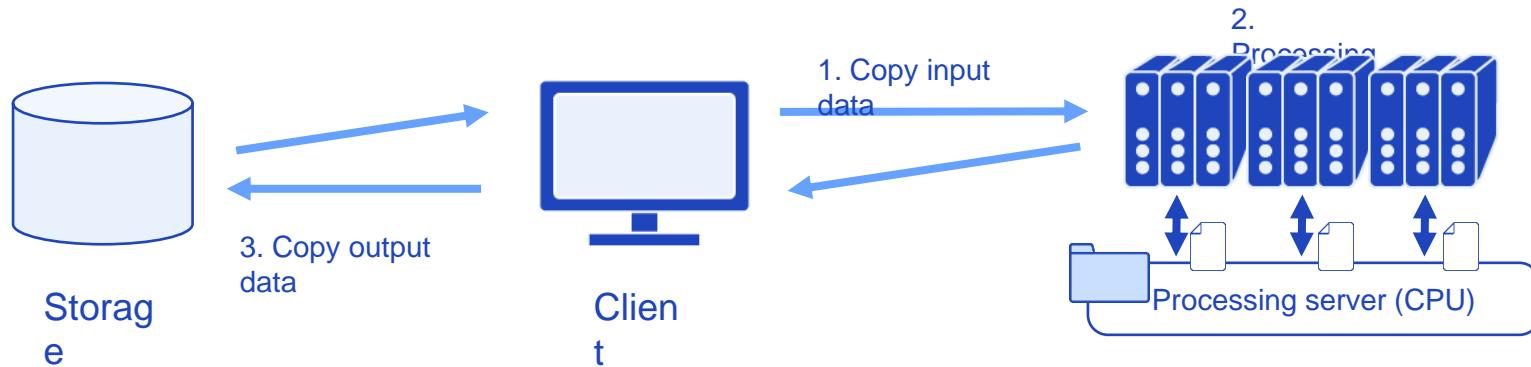
- ▶ One or more client computers connected to a central server over a network or internet connection
- ▶ Centralized management of data
- ▶ Client requests and pulls data for processing
- ▶ Incurs high network traffic and data movement costs



Client-Server vs. Parallel Cluster of Server (2/3)

Typical processing pattern of Client-Server

- ▶ Phase 1: copy input data from disk to processing server
- ▶ Phase 2: perform processing with data
- ▶ Phase 3: copy output data back to storage



Client-Server vs. Parallel Cluster of Server (3/3)

Parallel Cluster of Server

- ▶ Data is replicated on multiple servers
- ▶ Data is processed where the data is stored
- ▶ Reduce data I/O cost



Distributed Parallel Processing

Summary

- ▶ Create a distributed parallel processing cluster architecture where all processors are servers
- ▶ Use economical commodity hardware
- ▶ Install disk-based storage generously on all servers and process the data where the data is located
- ▶ Replicate the data multiple times in case of disk or server failure
- ▶ Manage fault-tolerance and increased performance using software based solutions rather than hardware based ones

Unit 2.

Current Trends in Big Data

Introduction to Big Data

Unit 2.

Current Trends in Big Data

- | 2.1. Edge-To-AI : Emerging automation tools
- | 2.2. Emerging role of Public Cloud Services in Big Data

Digital Transformation

- What is DT(Digital Transformation)?
 - It refers to an organization's efforts to incorporate new technologies, processes, and cultures for a common purpose
 - It refers to an organization's efforts to incorporate new technologies, processes, and cultures for a common purpose
 - The goal may be to improve the customer experience or accelerate innovation, or to find a way to survive the industry change caused by digital disruption
- Convergence accelerates Digital Transformation



The Core of 4th Industrial Revolution

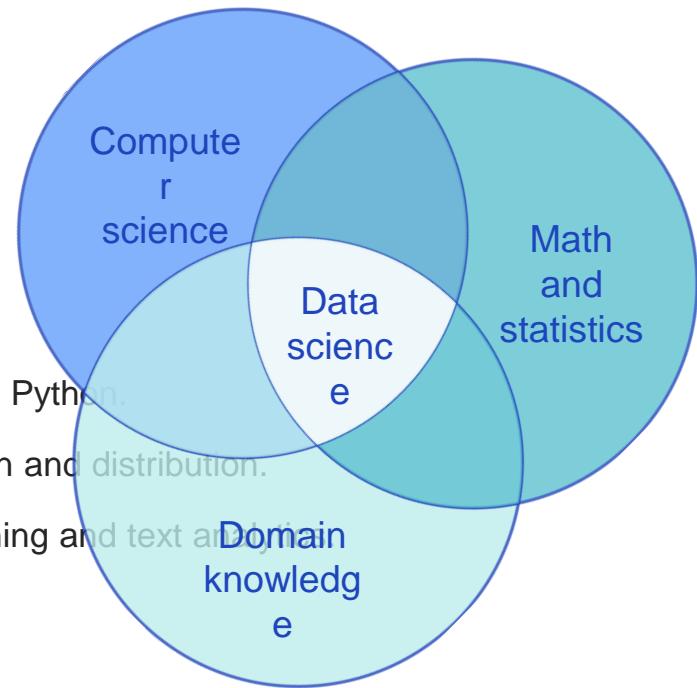
- IoT (Internet of Things)
 - It refers to a technology or environment in which sensors are attached to objects to exchange data in real time through the Internet, etc
- CPS (Cyber-Physical System)
 - A system that integrates real-time physical systems such as robots and medical devices, software in cyberspace, and the surrounding environment in real time
- Big Data
 - It refers to various types of data generated in the digital environment, and it refers to large-scale data with a large scale and a fast generation cycle
- AI
 - A field of computer science and information technology that enables computers to imitate human-specific intelligent behaviors such as thinking, learning, and self-development

Data Scientists vs Data Engineers

- Data Scientists focus on using cleaned data to create AI models using MLOps
- Growing use of AI on Edge: using AI algorithms to run at the edge of the network
 - Performing various operations at the edge where the data is collected, rather than back at the server
 - Feature extraction
 - Data preprocessing (outlier removal, data validation)
 - Model and algorithm application
- Data Engineers focuses on Data pipelining with cloud or on-premise cluster
 - Cloud cluster is the distribution of public cloud services to physical locations, while the operation, governance, updates and evolution of the services are the responsibility of the originating public cloud provider
 - Data preprocessing (ETL)
 - Collect large amounts of unstructured data and transform it into a more useful format.
 - Data collection and exploration

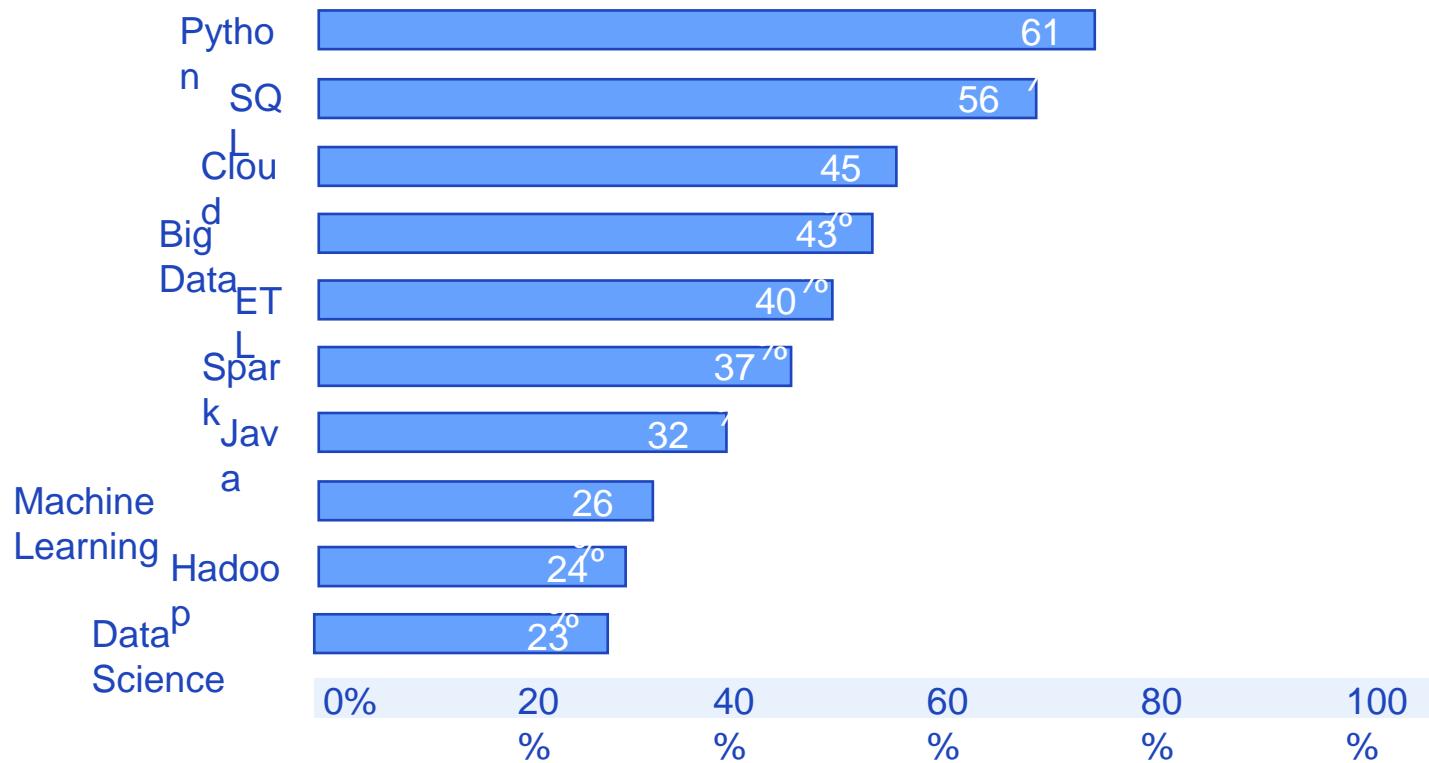
knowledge and Skills for Data Scientist

- Data Scientists Skill set
 - Computer science for Analytics
 - Math and statistics for Data Modeling
 - Domain knowledge for Insight
- Works
 - Use data-driven technologies to solve business-related problems.
 - Work with a variety of programming languages including SAS, R and Python.
 - Have a solid understanding of statistics, including statistical validation and distribution.
 - Based on analytics techniques such as machine learning, deep learning and text analysis.



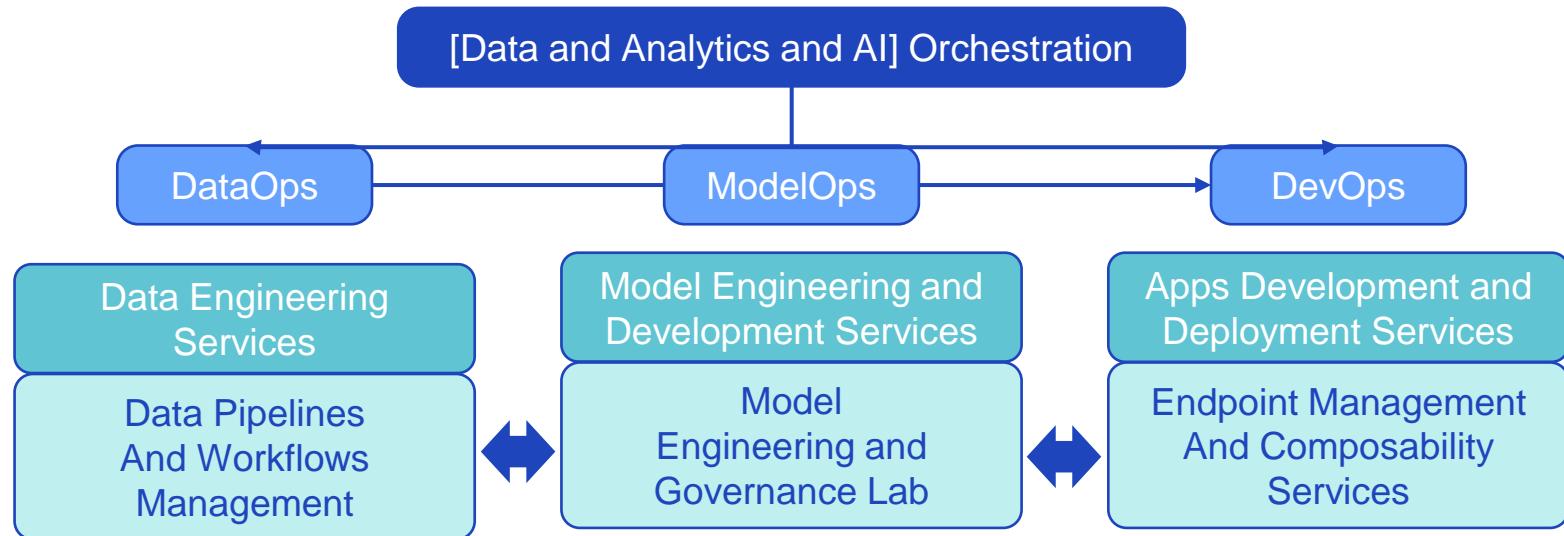
knowledge and Skills for Data Engineer

- Top 10 Skills for Data Engineers in 2021



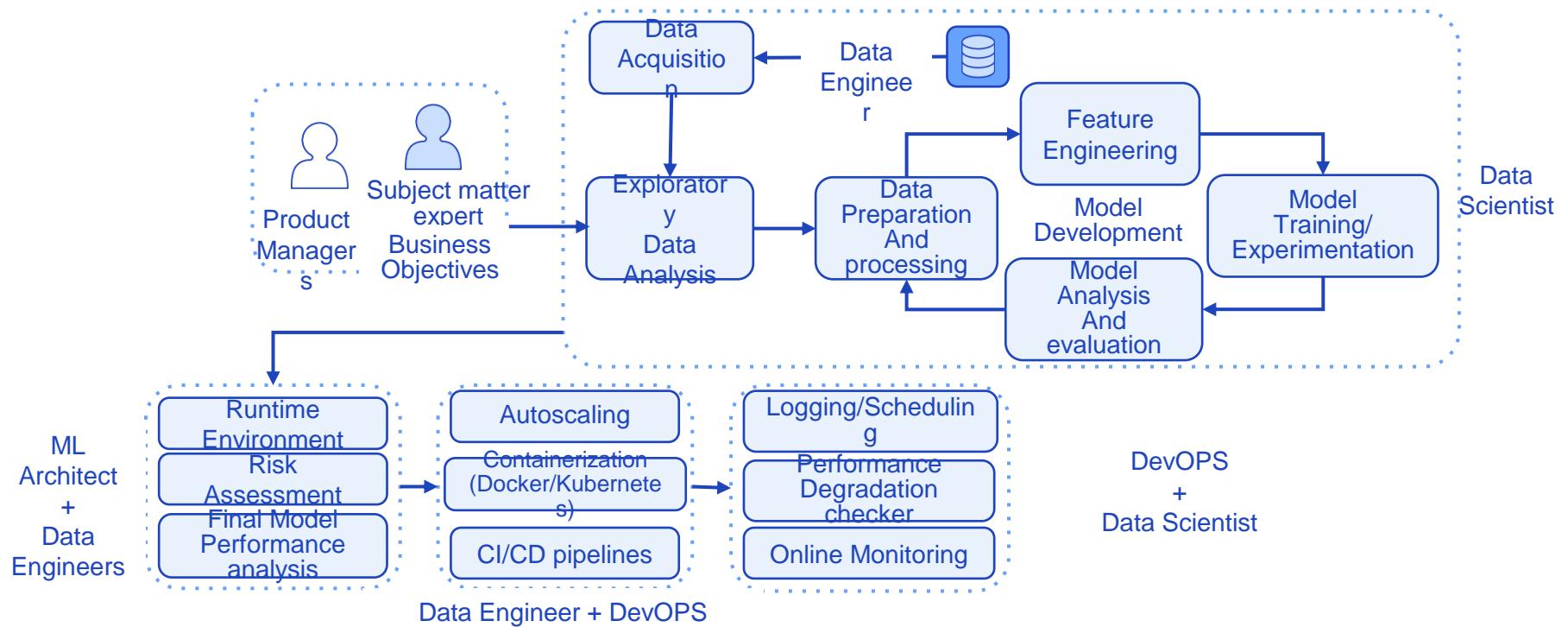
Edge Computing to AI (1/2)

- What is XOps?
 - The goal of XOps (data, machine learning, model, platform) is to achieve efficiencies and economies of scale using DevOps best practices — and to ensure reliability, reusability and repeatability while reducing the duplication of technology and processes and enabling automation. - Gartner -



Edge Computing to AI (2/2)

- Citizen Data Scientists (CDS) and Citizen Data Engineers (CDE) combine both roles
 - Trend is to develop automation tools to assist and facilitate the combining of DataOps, MLOps and DevOps



Unit 2.

Current Trends in Big Data

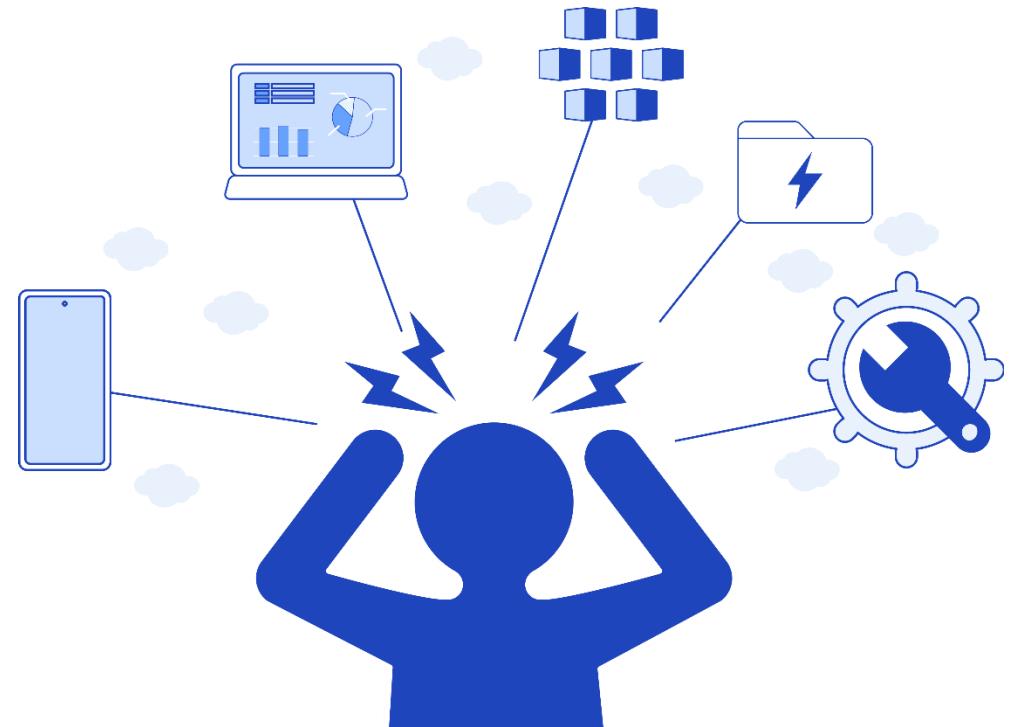
- | 2.1. Edge-To-AI : Emerging automation tools
- | 2.2. Emerging role of Public Cloud Services in Big Data

Cloud Computing

- What is Cloud Computing
 - In this architecture, the data is mostly resident on servers 'somewhere on the Internet' and the application runs on both the 'cloud servers' and the user's browser. -Eric Schmidt –
- Why was Google thinking about that?
 - A very successful service to keep computer users' applications and data on the web through the industry's most advanced web services (Google Docs, Google Notes, etc.).
 - Through this, it is predicted that all software will exist on the Internet in the future.
 - Presenting the concept of “SaaS”
 - Accumulation of Google experience to solve the worldwide connected load
- Modern business in the cloud is the new normal.

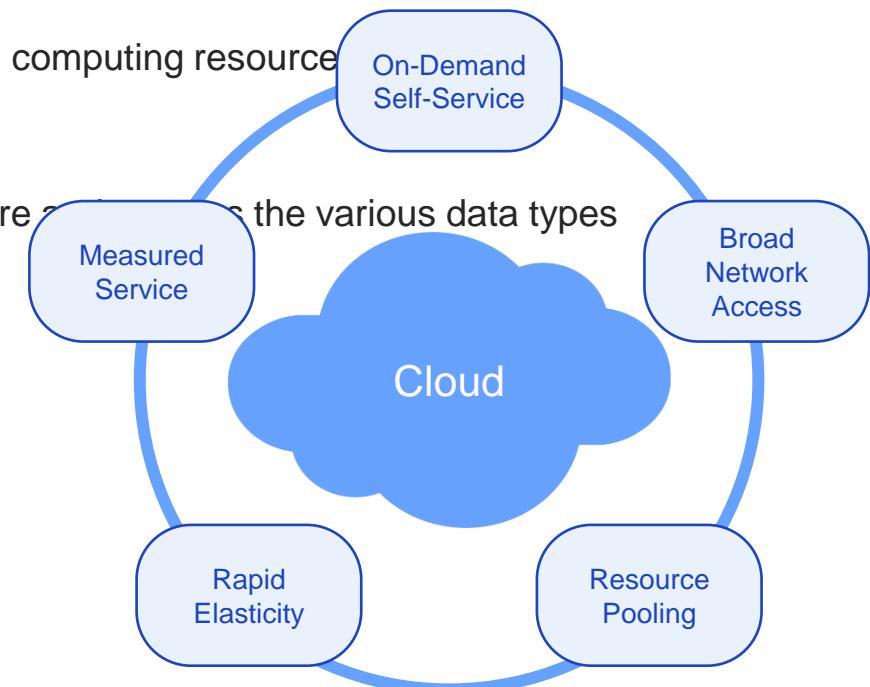
Limitations of Traditional Architecture

- On-premise computing
 - Storage and compute collocation
 - Inefficient cluster operation due to multi-tenancy
 - Inefficient distribution of resources
 - Unable to support auto-scaling according to load
 - Physical Data Cluster Security

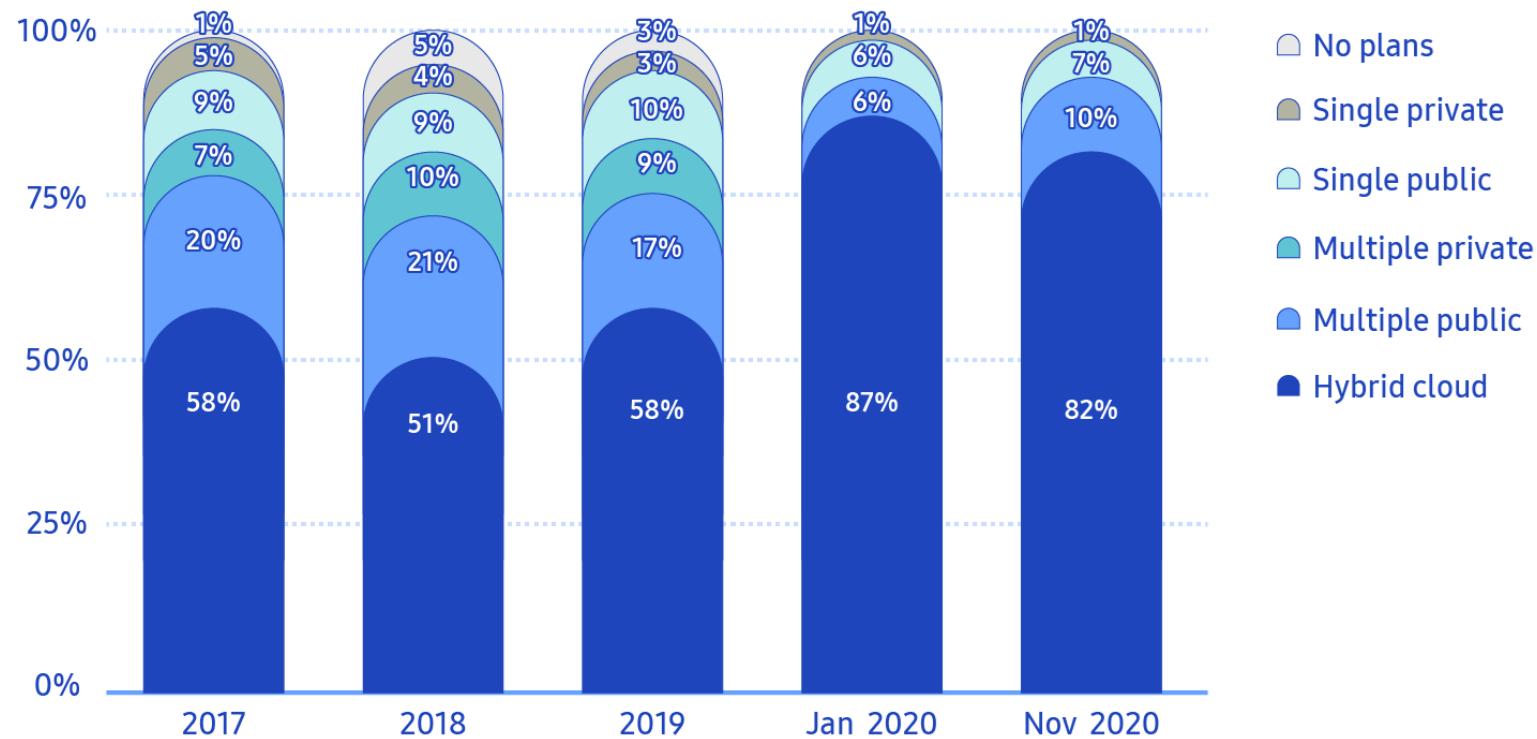


Key aspects of the Public Cloud

- Able to support auto-scaling according to load
- High-speed network allows storage to be separated from computing resources
 - Able to scale them individually for each requirements
- Object store(S3, ADLS) provides a useful method for storing the various data types
- The container technology
- Reliable security enhancements for data clusters

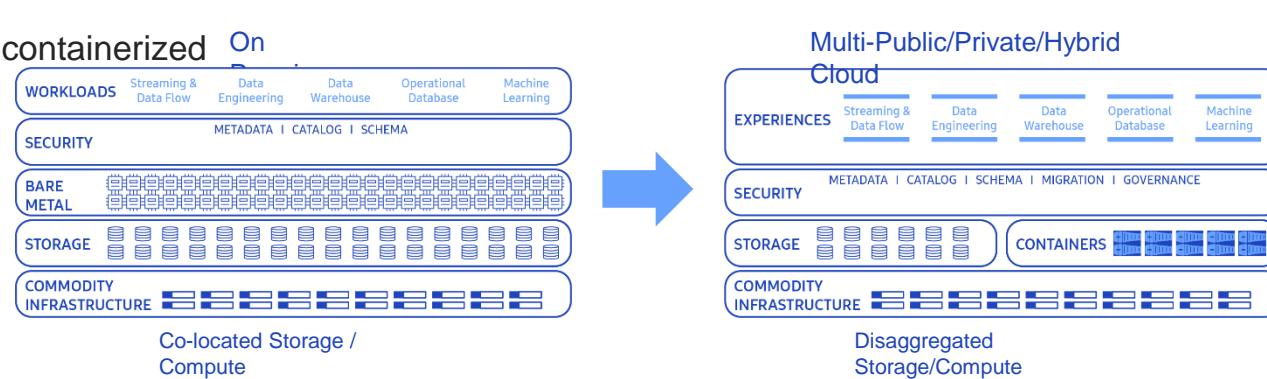


Enterprise cloud strategy worldwide



Evolution of Architecture (1/2)

- Multiple Public Cloud
 - Multiple cloud refers to the deployment of two or more clouds of the same type provided by different vendors.
- Private Cloud
 - A private cloud is a cloud environment dedicated to end users, typically located within the user's firewall.
- Hybrid Cloud
 - A hybrid cloud integrates some level of workload portability, orchestration, and management across two or more environments (public and private)
 - Multi-tenant, containerized



Evolution of Architecture (2/2)

- Important advantages with Hybrid cloud
 - Disaggregate the software stack –storage, compute, security and governance
 - More options for deploying computational and storage resources
 - Customize the deployment resources using on-premise servers, containers, virtual machines, or cloud resources
- Workload Isolation
 - When deploying a cluster to your cloud infrastructure, you can temporarily terminate your compute cluster to avoid unnecessary costs.
 - While keeping data from other applications available for continued use
 - Computing clusters can help resolve resource contention
 - Long-running or resource-intensive workloads can be isolated to run in a dedicated compute cluster

How to build a Hybrid cloud?

- Traditional hybrid cloud architecture
 - Hybrid cloud was originally literally connecting a private cloud environment to a public cloud environment as an iteration of large and complex middleware
 - You can build your own private cloud or use a pre-packaged cloud infrastructure, you also need a public cloud like the one below

 Alibaba Cloud



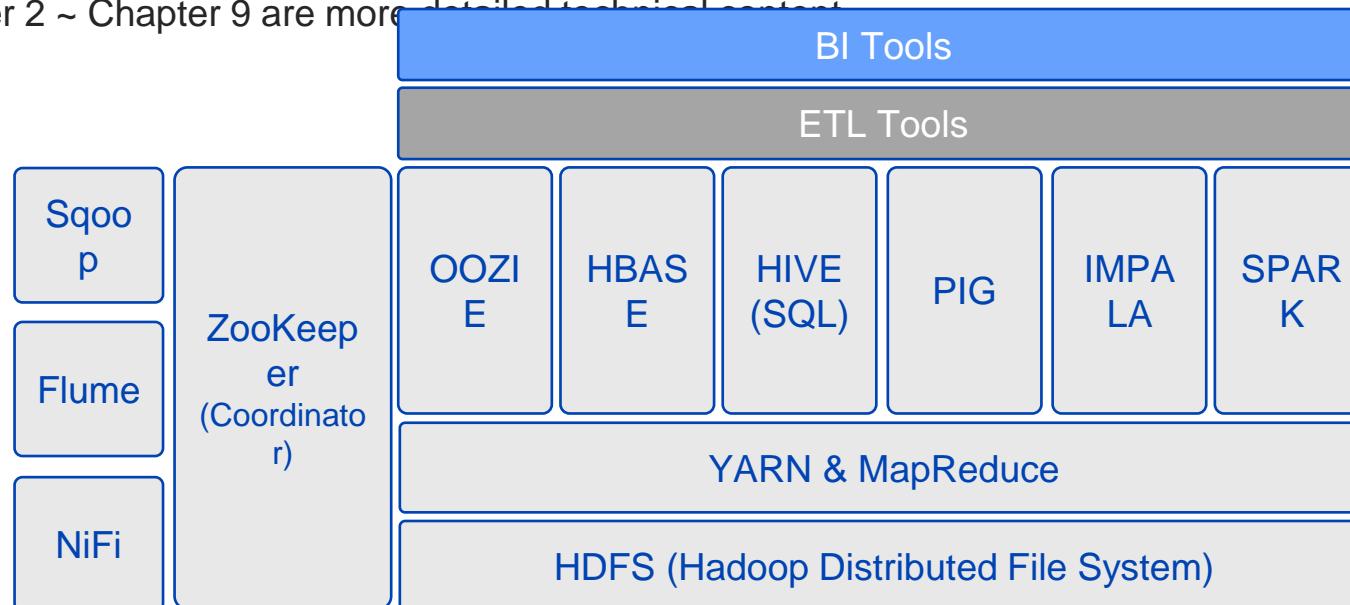
 Google Cloud



- Modern hybrid cloud architecture
 - All IT environments run the same operating system and manage everything through a unified platform, while allowing the universality of applications to extend to sub-environments
 - All hardware requirements are abstracted using the same operating system, and the orchestration platform abstracts all application requirements.

What is Next?

- Chapter 1 was an introduction to Big Data
 - Big Data Overview and Background
 - Current Trends in Big Data
- Chapter 2 ~ Chapter 9 are more detailed technical content



Chapter 2.

Fundamentals of Big Data

Big Data Course

Chapter Description

Objectives:

- ✓ We will learn how Hadoop and its Ecosystem tools come together to solve the Big Data challenge
 - Hadoop solves the problem of storing and processing massive amounts of data in a cost-effective, scalable, and fault-tolerant way through a paradigm shift from hardware to software dependency
 - Hadoop, at its core, provides HDFS for data storage and Yarn/MapReduce for data processing
 - The Hadoop Ecosystem provides a rich set of tools to ingest data from various data sources, pre-process and transform the data in preparation for querying, perform both interactive and massive batch queries on petabytes of data, generate AI models and infer answers, and finally generate intuitive and user-friendly visualizations and dashboards to help understand the results.
- ✓ Big Data is constantly evolving. We shall explore how Cloud Services are playing an important role and how the industry is adopting the Cloud as an alternative to Hadoop.

Contents of Chapter:

1. Big Data Processing
2. Hadoop Core & Eco system overview
3. Hadoop Architecture for Big Data

Unit 1.

Big Data Processing

| Fundamentals of Big Data

Unit 1.

Big Data Processing

- | 1.1. Big Data Application and Processing
- | 1.2. Big Data on the Public Cloud

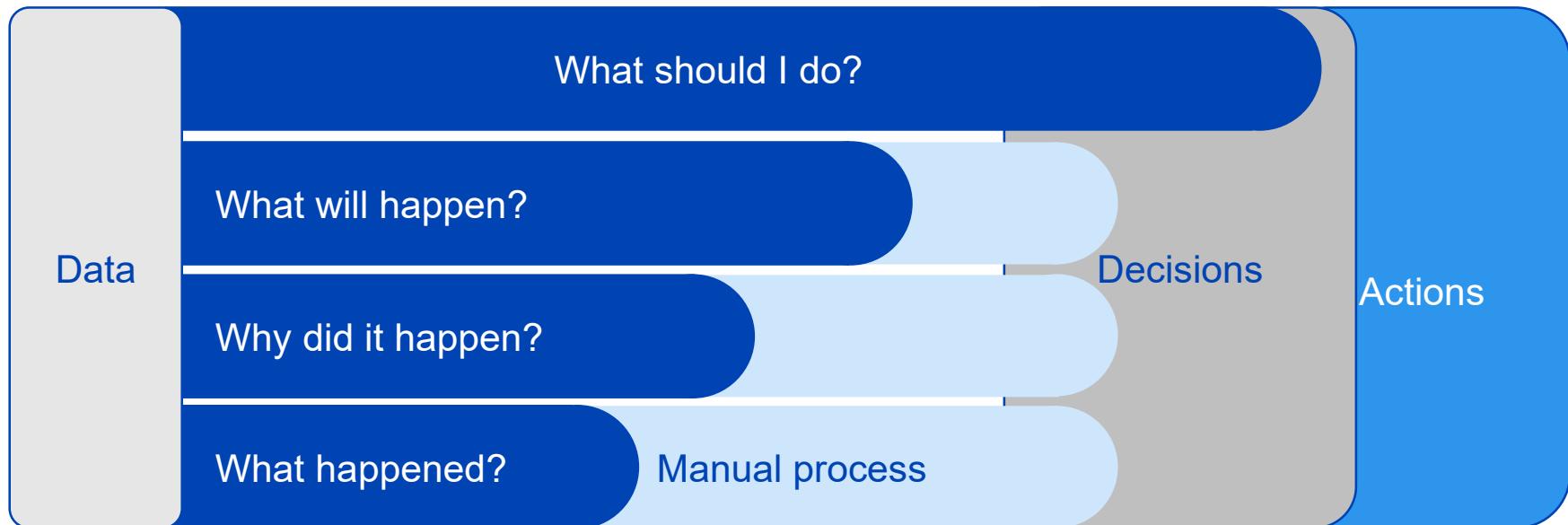
A Goal of Big Data Processing and Analysis

- Comparing Predictive Analytics and Prescriptive Analytics

Predictive Analytics	Prescriptive Analytics
<ul style="list-style-type: none">✓ Learns from the past✓ Looks at past data and designs statistical models or machine learning models to predict the future	<ul style="list-style-type: none">✓ Takes actions based on the past✓ Takes those predictions about the future and turns them into actions or policies

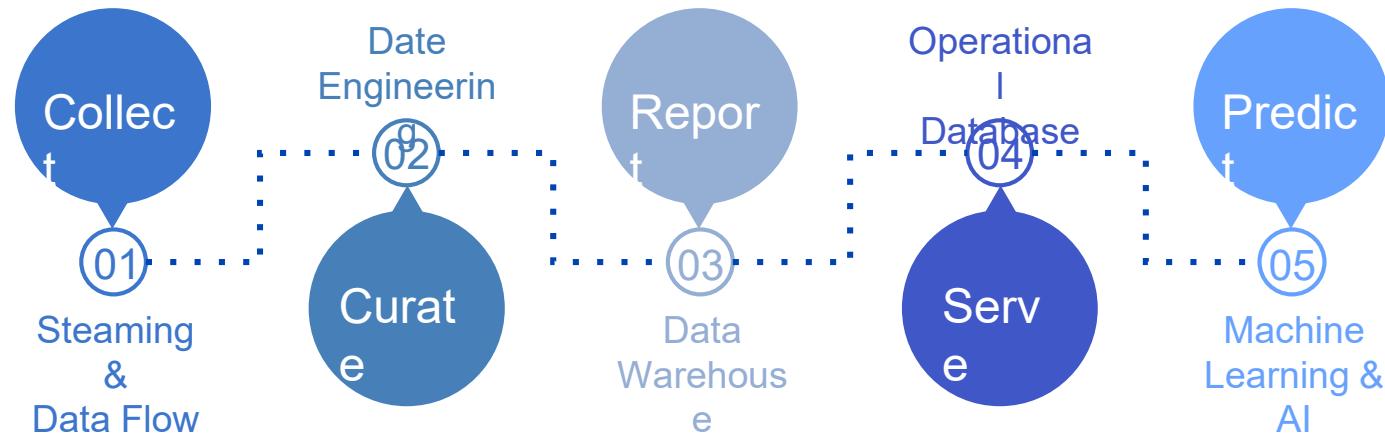
A Goal of Big Data Processing and Analysis

- From Predictive Analytics to Prescriptive Analytics



Big Data Pipeline

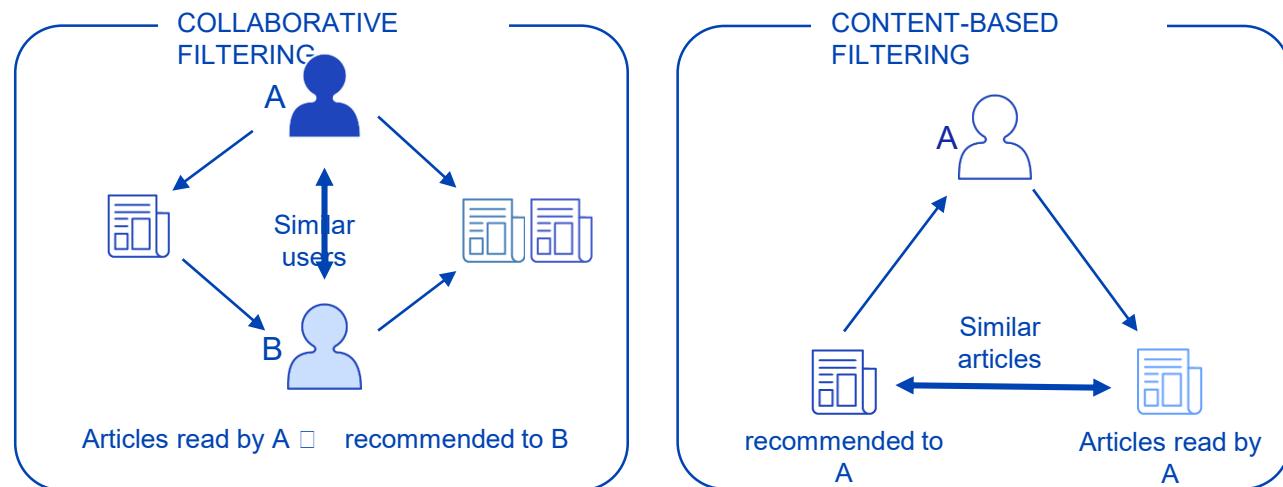
- Services customized for specific steps in the data lifecycle
- Emphasize productivity and ease of use
- Auto-scale compute resources to match changing demands
- Isolate compute resources to maintain workload performance



Big Data Use Case - Recommendation

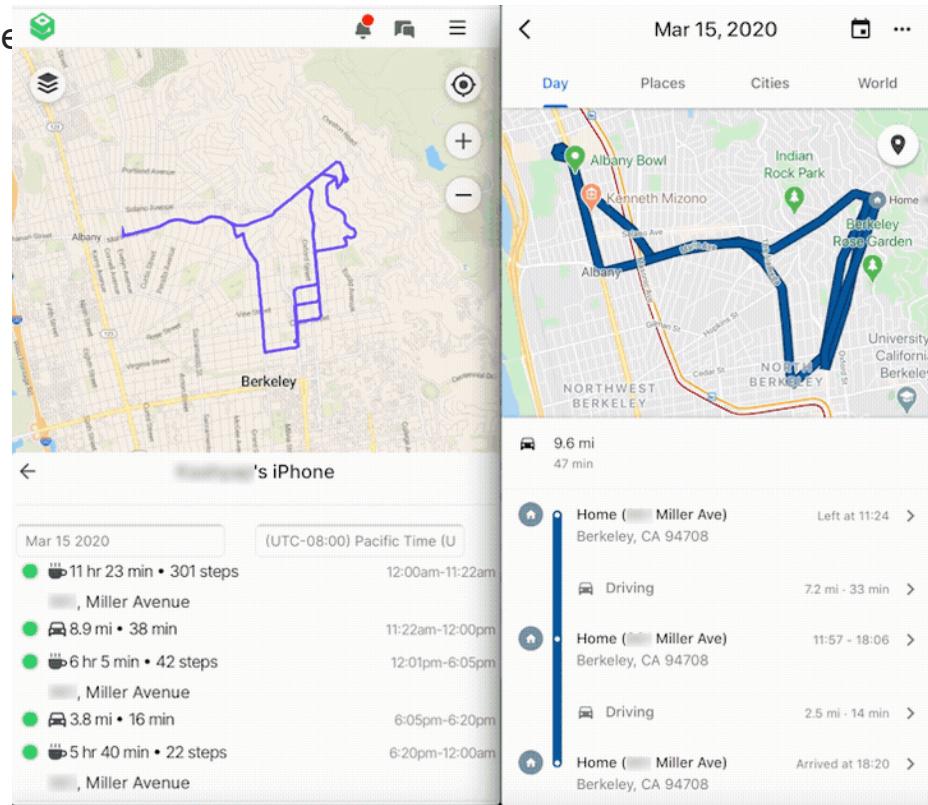
- Predict user interests and deliver customized recommendations
- Highly applicable across multiple industries
 - On-line shopping
 - Streaming Services
- Collaborative Filtering and Content Filtering are main techniques deployed

- Applicable companies
 - Netflix
 - Amazon



Big Data Use Case - Customized Service

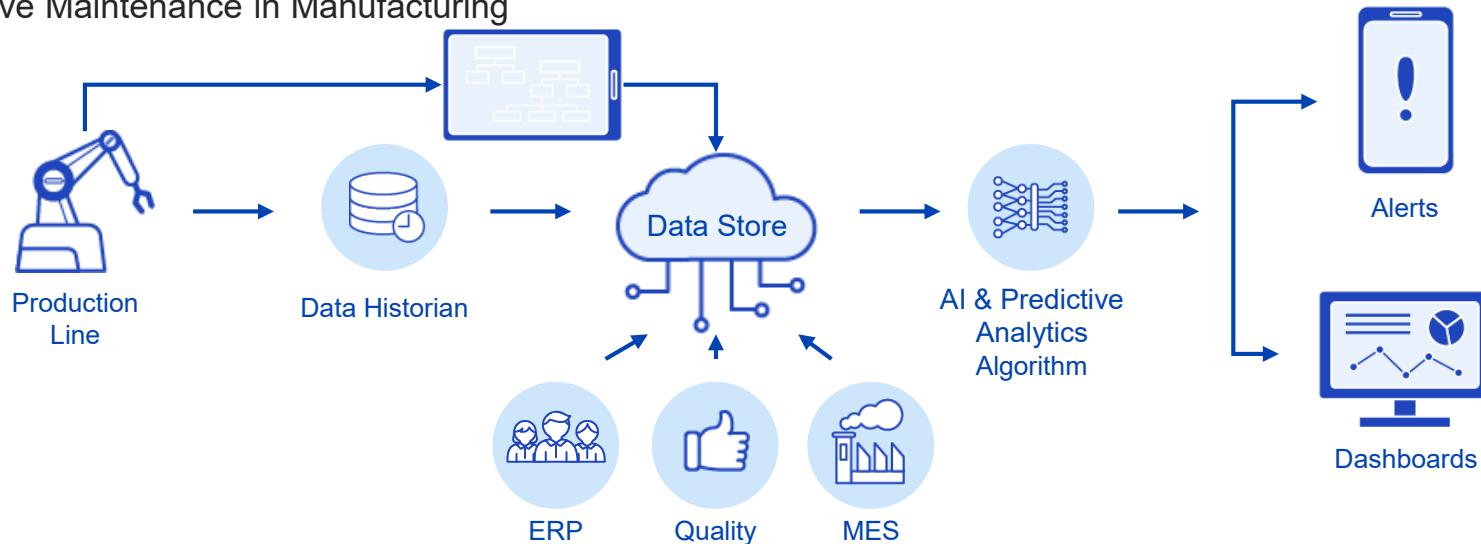
- Provide customized service based on 360 degree
 - Location
 - Gender
 - Wealth
 - Interests
- Applicable Services
 - Google Maps Timeline
 - Google and YouTube ads



<https://hypertrack.com/blog/content/images/2020/04/Comaprison.gif>

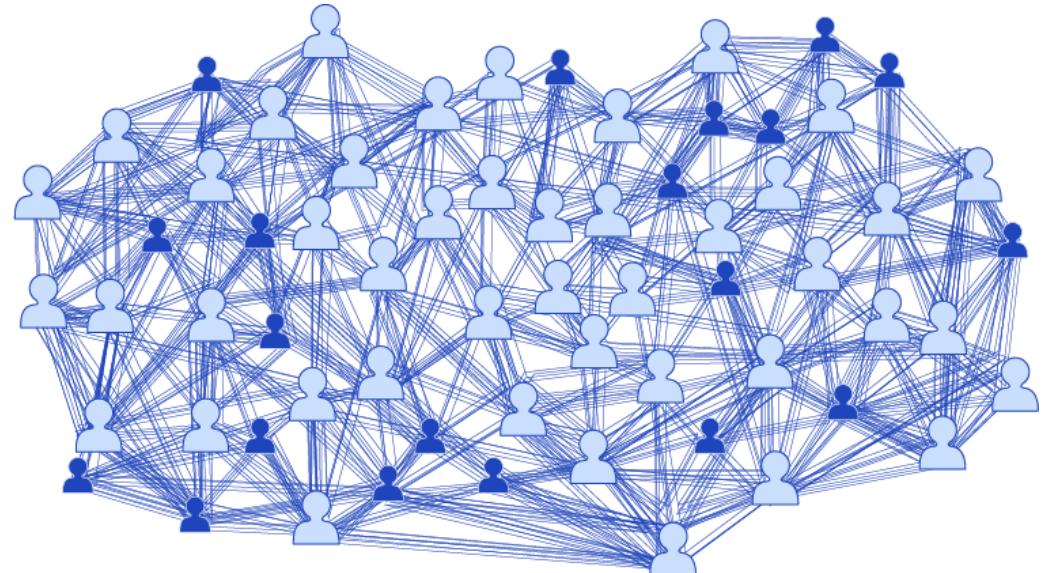
Big Data Use Case - Predictions

- Predict future profits or trends by identifying specific patterns from relevant from accumulated data
- Used in various industries
 - Product development
 - Risk and security management
 - Predictive Maintenance in Manufacturing



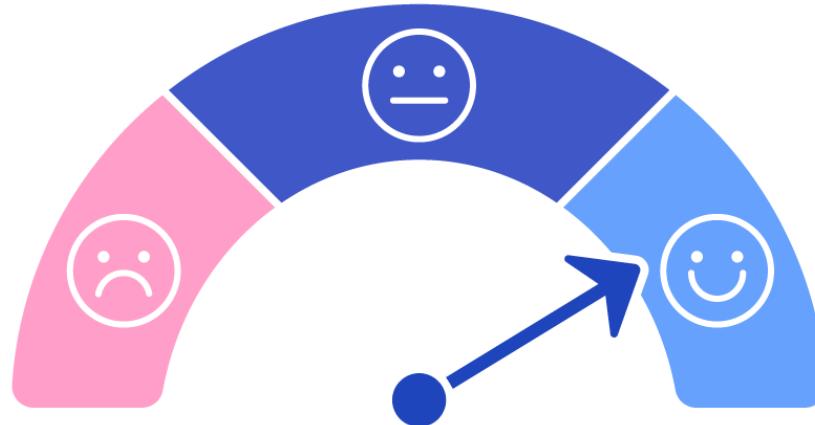
Big Data Use Case - Associative Analysis

- Association
 - Create machine learning models for analyzing events that occur together
 - Walmart - Customers who purchase baby diapers on weekends tend to purchase beer as well
 - Investment - Investors who purchase stocks in A and B tend to purchase C as well
- Graph Analytics
 - Finds connections between nodes and edges
 - Nodes and users, edges as relations
- Applicable Industries
 - On-line and off-line shopping
 - Social Network Services



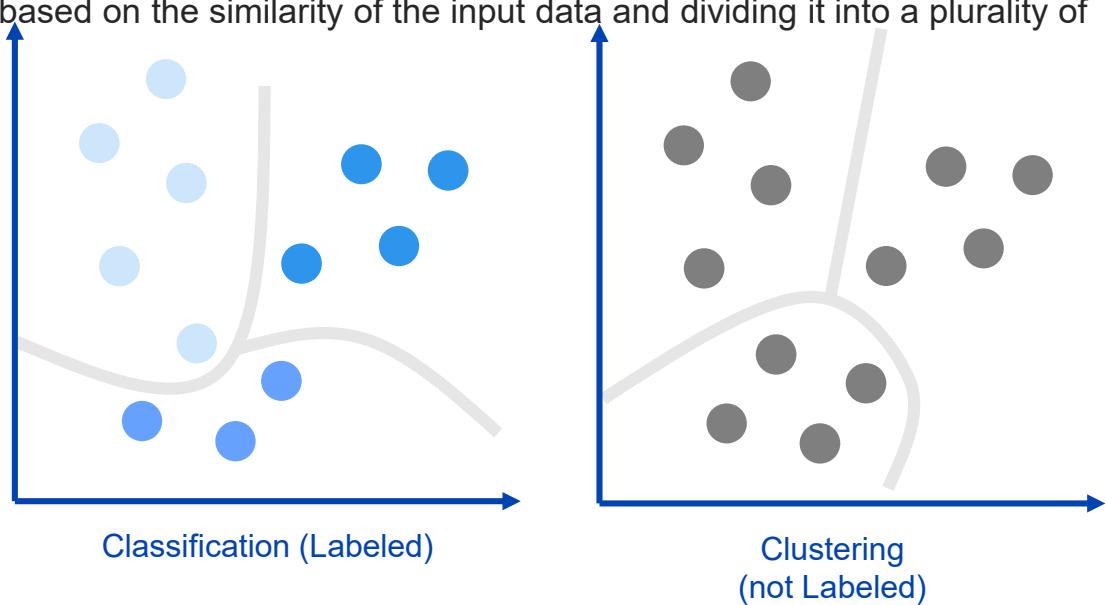
Big Data Use Case - Sentimental Analysis

- Natural Language Processing
- Voice of Customer (VoC)
 - Analyze VoC that may be stored in various formats such as text, audio, etc.
 - Analyze subjective impressions, emotions, attitudes, and individual opinions on a topic from unstructured data such as SNS and product reviews
- Applicable Use-Cases
 - Marketing
 - Customer service
 - Clinical service



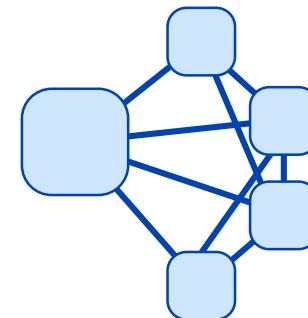
Big Data Use Case – Classification/Clustering

- Classification
 - Classifying a given data set into several predefined categories
- Clustering
 - Identifying the distribution characteristics based on the similarity of the input data and dividing it into a plurality of arbitrary groups.
- Applicable Use cases
 - Inappropriate content sanctions using Facebook AI
 - Pharmaceutical Companies



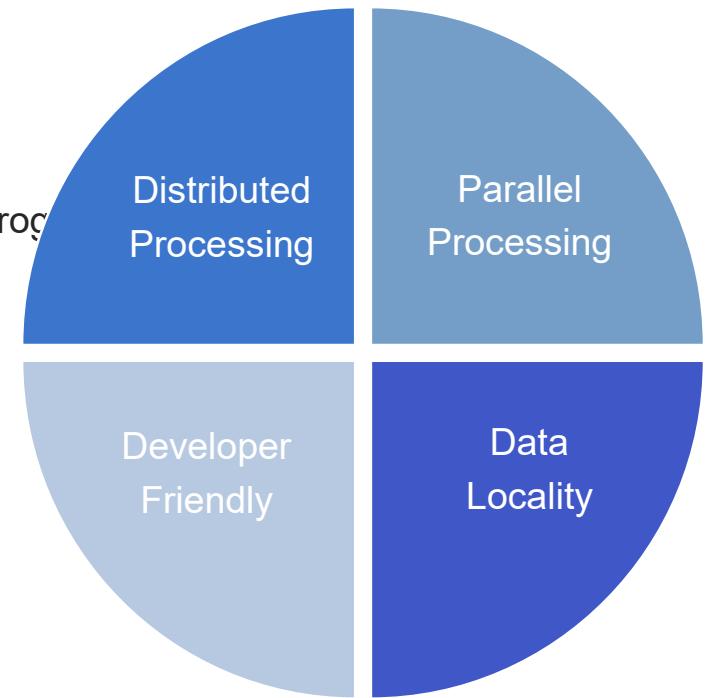
MapReduce: Core Big Data Compute Engine

- MapReduce as a programming model for Big Data
- Record-oriented data processing (key and value)
- Consists of two developer-created phases
 - Map
 - Reduce
- In between Map and Reduce is the shuffle and sort
 - Sends data from the Mapper to the Reducers



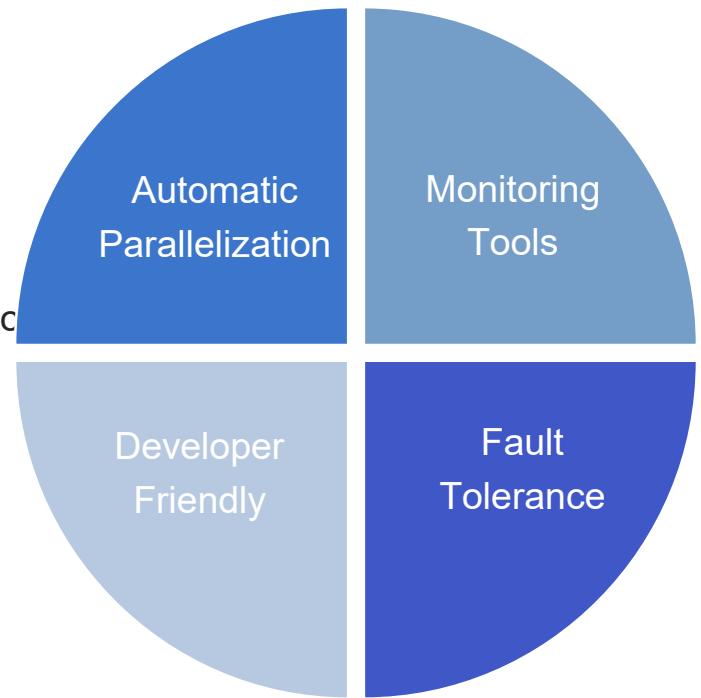
Features of MapReduce

- MapReduce is a distributed parallel processing method.
 - Distribute work across multiple slave nodes
- Run on nodes that have data available
- An environment provided so that developers can focus on their program
 - Automatic data distribution
 - Possible local data processing
 - Move data on demand
 - Supported languages - Java and other languages

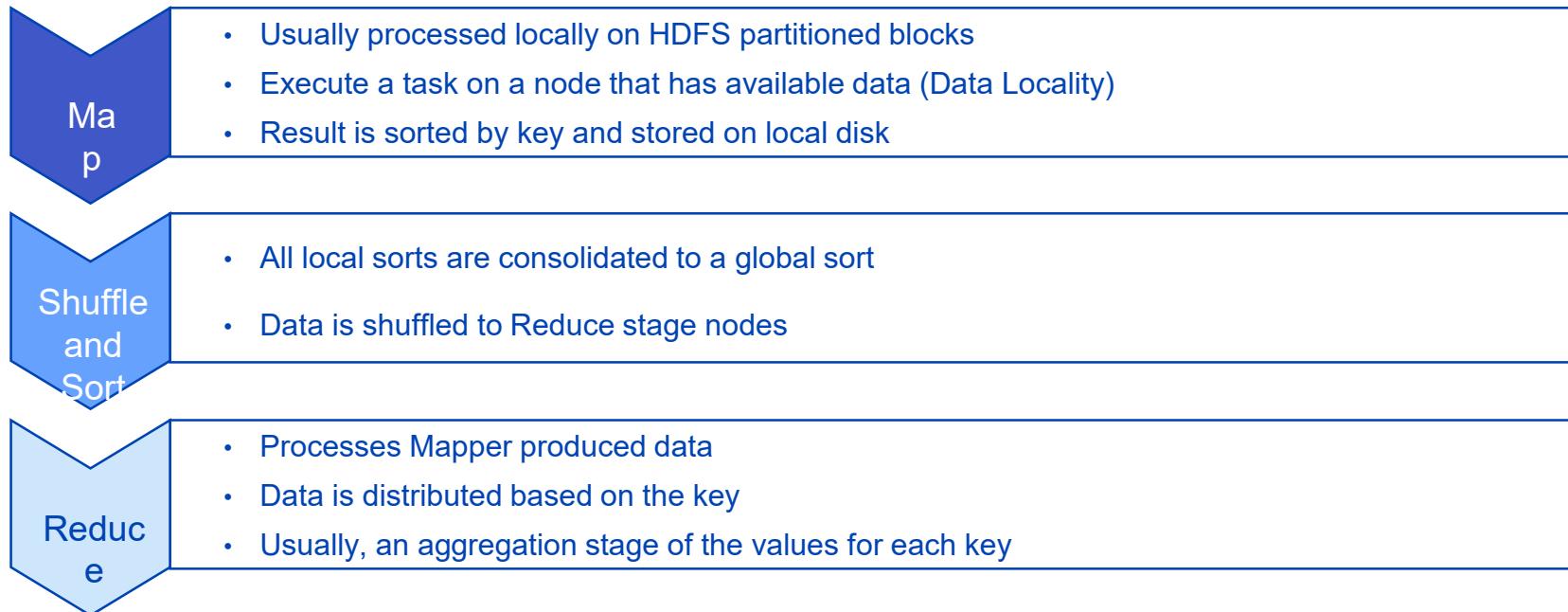


Advantages of MapReduce

- Automatic parallelization and distribution
- Fault-tolerance
- Status and monitoring tools
- A clean abstraction for programmers
- MapReduce abstracts all the ‘housekeeping’ away from the developer

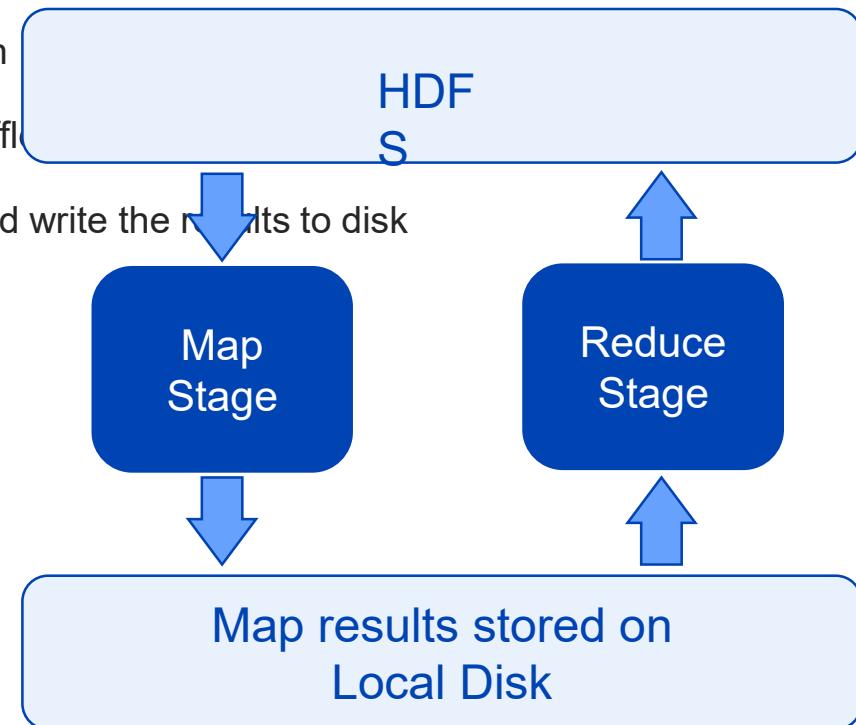


Three Stages of MapReduce



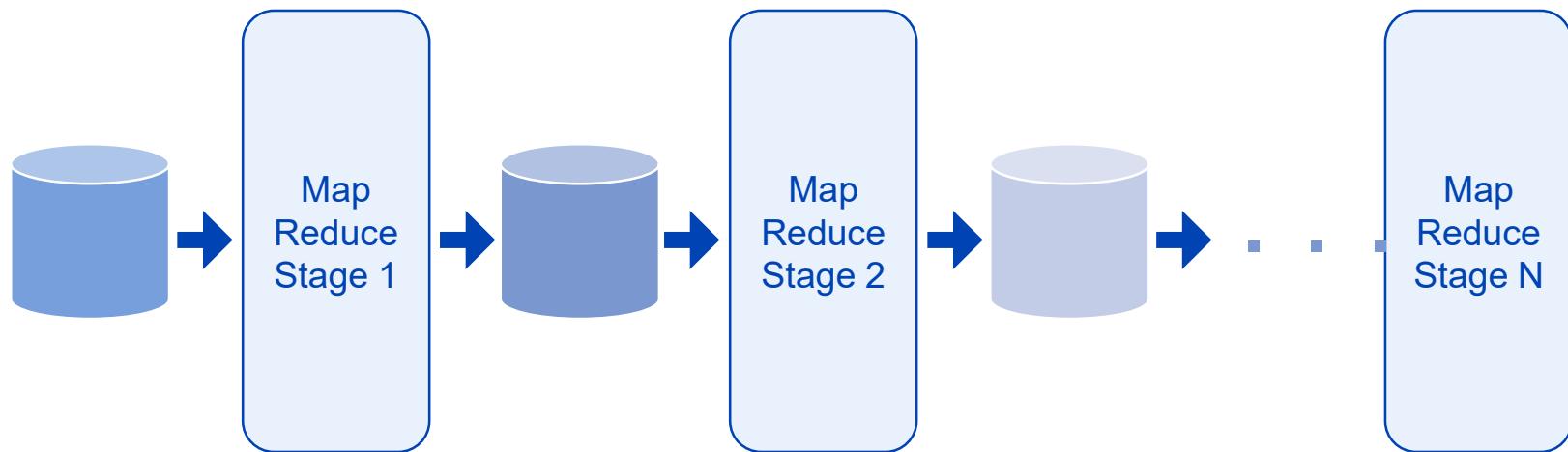
MapReduce Disk Access Pattern

- Mappers read data from disk and perform computation
- After all the mappers have completed, the data is shuffled
- The Reducers read the data, perform the reduction and write the results to disk



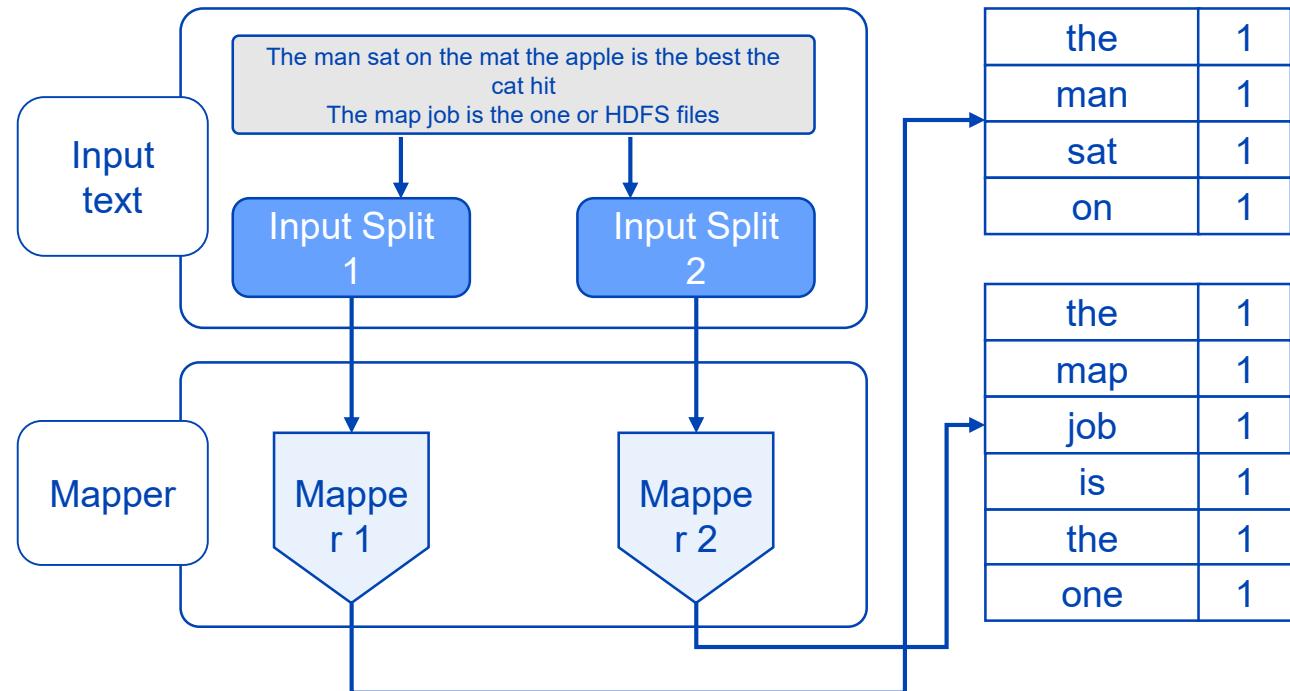
Multiple MapReduce Cycles

- Usually to get any interesting result, a job has to go through multiple Map and Reduce cycles
- In each of the map reduce stage, we already saw disk read and write between mapper and reducer
- In between the stages, each of the immediate results is written to disk and read write back by the next stage
- Disk I/O is very EXPENSIVE



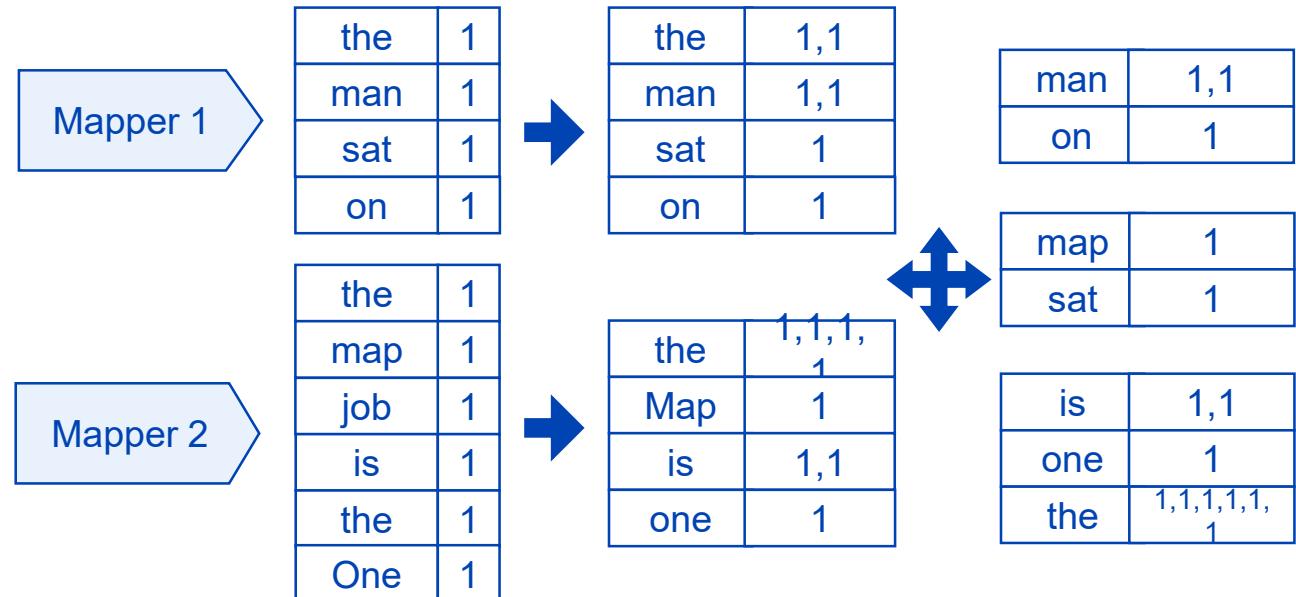
MapReduce - Word Count

- Map Stage
 - Read text from input source
 - Create key:value pair
 - Key is each word
 - Value is constant 1



MapReduce - Word Count

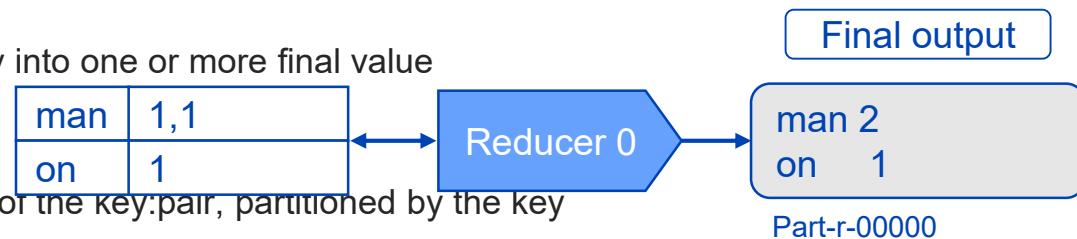
- Shuffle Sort Stage
 - The result of each Mapper is sorted by the key and stored on local disk
 - Optionally, for better performance, as the keys are sorted, they may be aggregated



MapReduce - Word Count

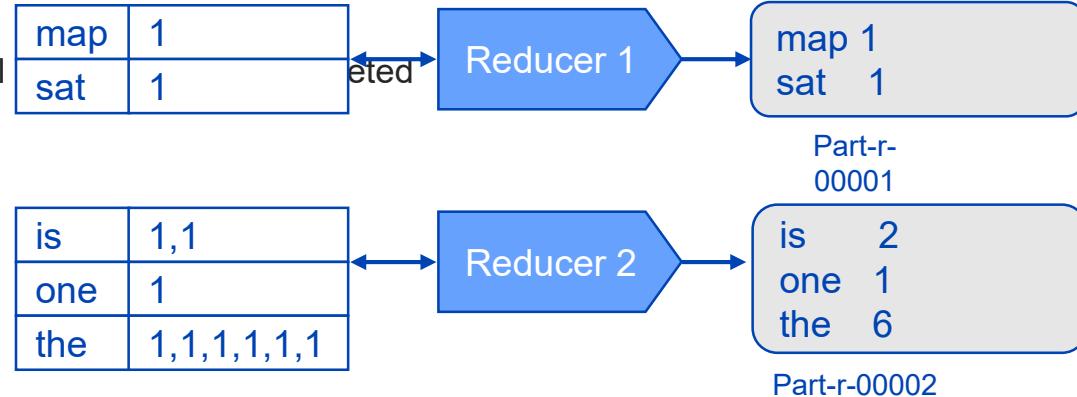
- Reducer Stage

- Combine intermediate values for each key into one or more final value
- Each reducer can run in parallel,
- Each reducer works on a different subset of the key:value, partitioned by the key



- Bottleneck

- The reduce step can only proceed after all mappers have completed.
- Slowest mapper dictates speed



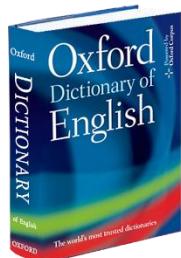
Unit 1.

Big Data Processing

- | 1.1. Processing Big Data On-Premise
- | 1.2. Big Data on the Public Cloud

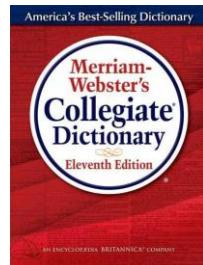
What is Cloud Computing?

- “Cloud Computing” definition of dictionary



“The practice of using a network of remote servers hosted on the Internet to store, manage, and process data, rather than a local server or a personal computer.”

“The practice of storing regularly used computer data on multiple servers that can be accessed through the Internet.”



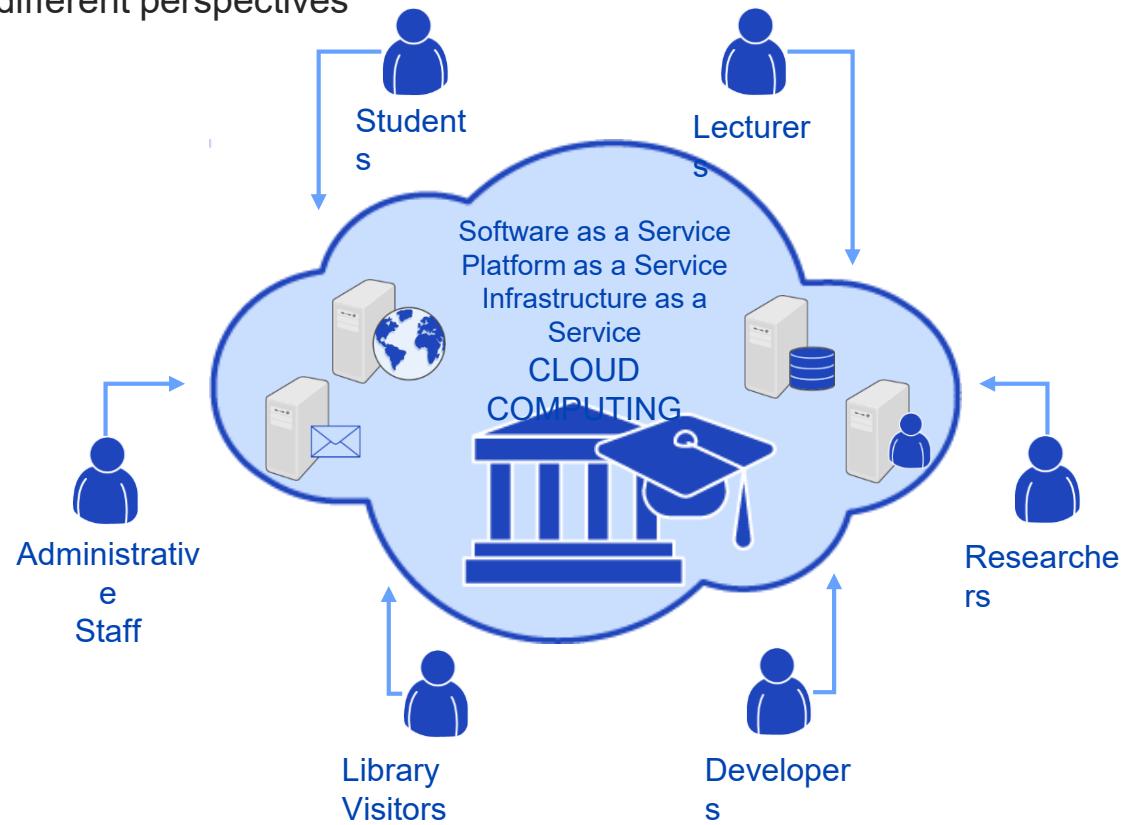
What is Cloud Computing?

[Discussion
]



Cloud Computing Perspectives

- What Cloud Computing means from different perspectives
 - End-User
 - Application Developer
 - Service Provider
 - IT Infrastructure Manager
 - CIO
 - CFO



Evolution of Cloud Computing

- Advances and Changes in Cloud Technology

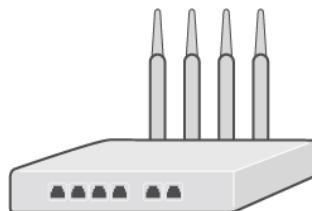


Stage	Characteristics
Grid Computing	Solving large problems with parallel computing Made mainstream by Global Alliance
Utility Computing	Computing resources offered as a metered service Late 1990s
Software as a Service	Subscription-based software accessed over the Internet Gained momentum after 2001
Cloud Computing	Next-generation datacenters with virtualization technology Full stack of service - IaaS, PaaS, & SaaS

Key Enabling Technologies

- Ubiquitous fast wide-area networks
- Powerful and inexpensive servers
- High-performance virtualization technology

Broadband WAN



Servers



Virtualization

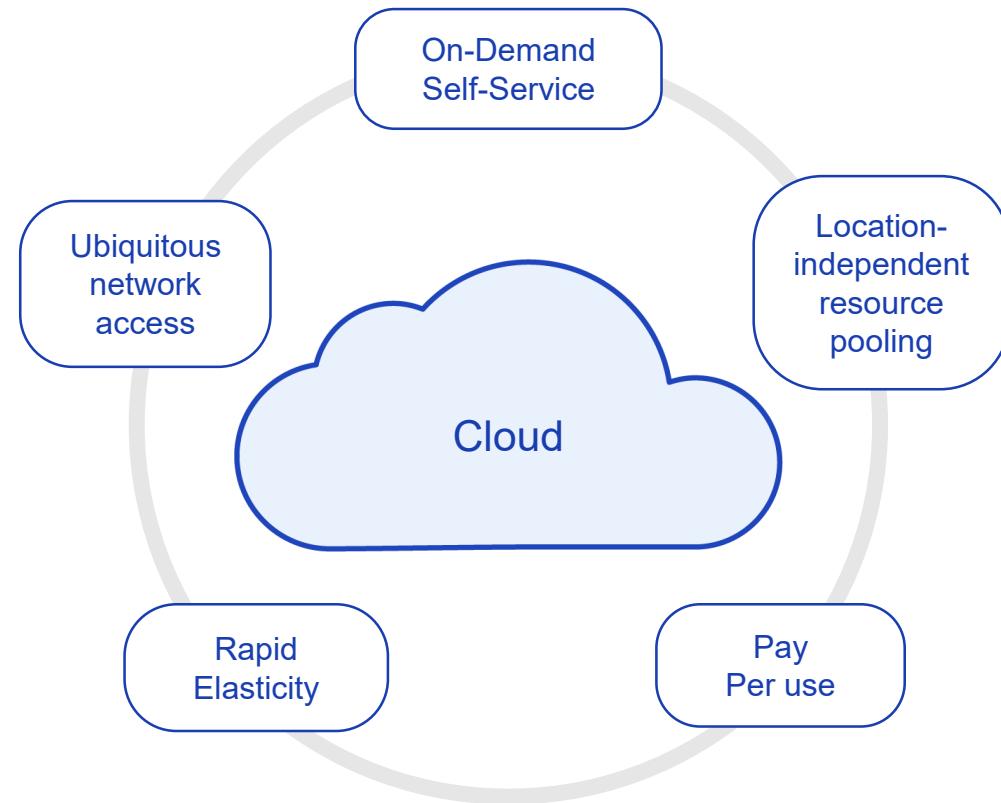


HYPERVISOR



Key Characteristics of Cloud Service

- On-demand self-service
- Ubiquitous network access
- Location-independent resource pooling
- Rapid elasticity
- Pay per use

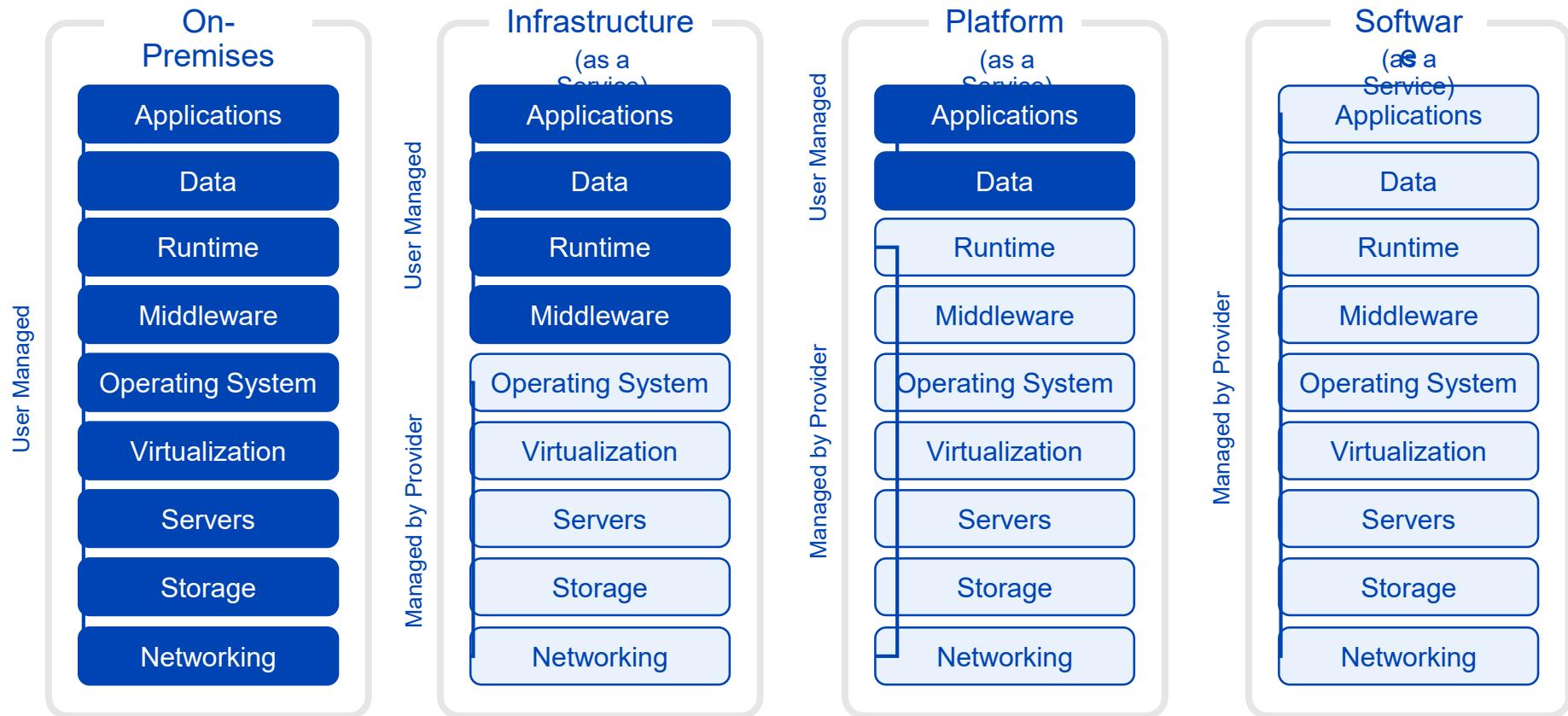


Cloud Service and Consumption Model

- Comparing SaaS, PaaS and IaaS

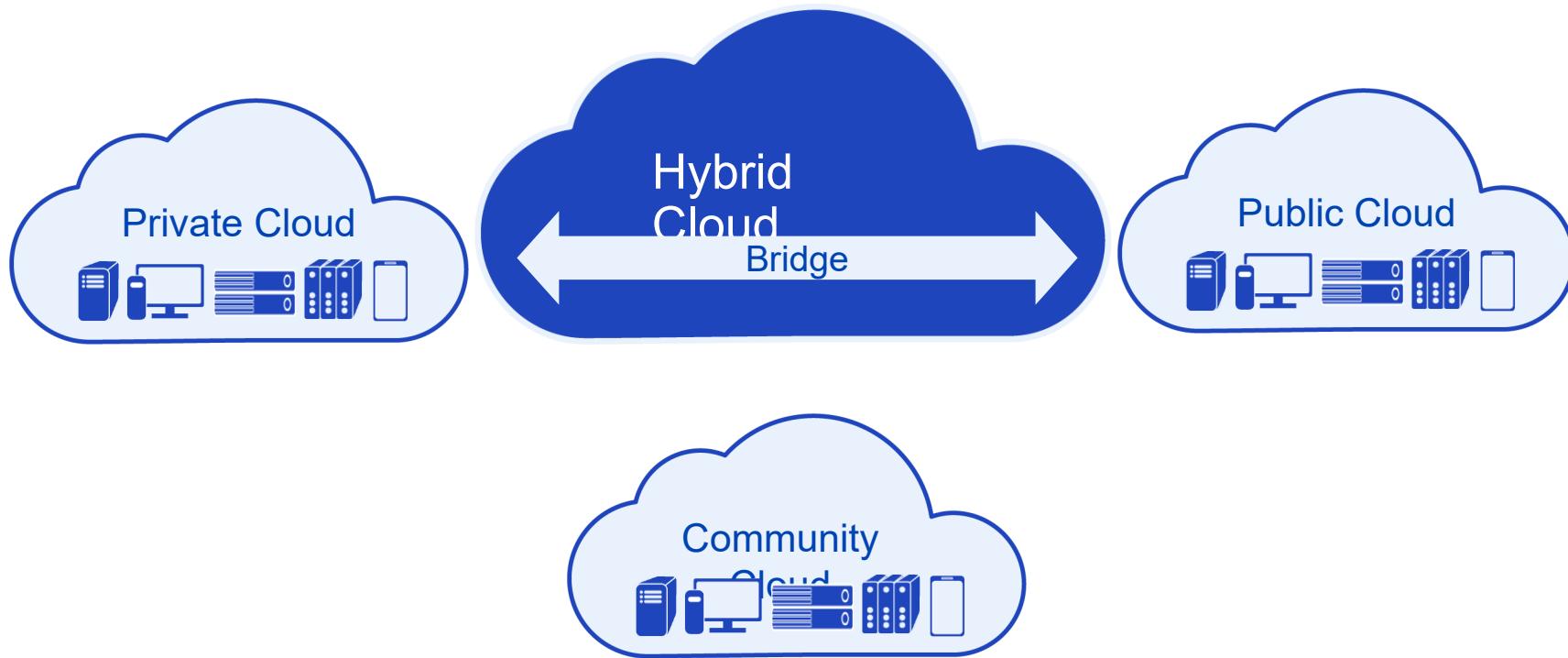
Service Model	Consumption Model	Description
Software as a Service (SaaS)	Directly consume service	End-User Applications delivered as a service
Platform as a Service (PaaS)	Develop and build on the platform	Application platform or middleware provided as a service
Infrastructure as a Service (IaaS)	Infrastructure on-demand	Computing, storage, or other IT infrastructure provided as a service

Service Model Division of Responsibility

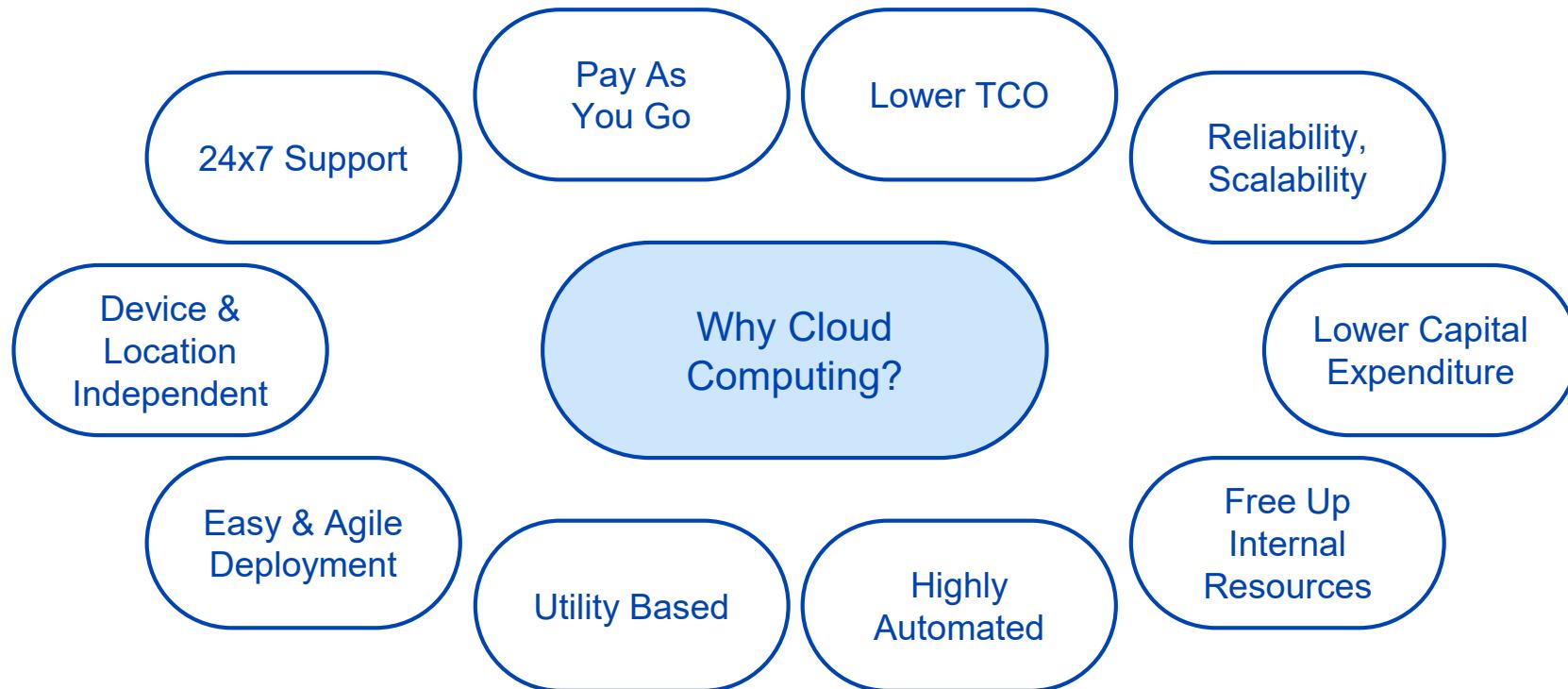


Cloud Deployment Models

- Types of Cloud

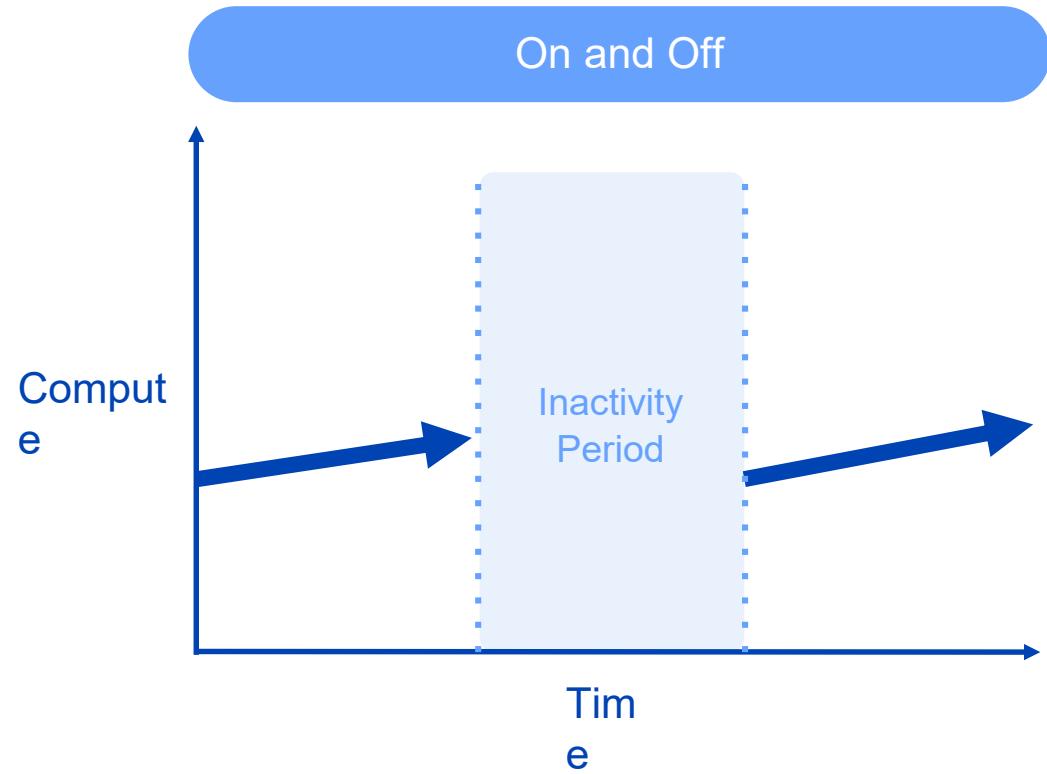


Why Cloud Computing?



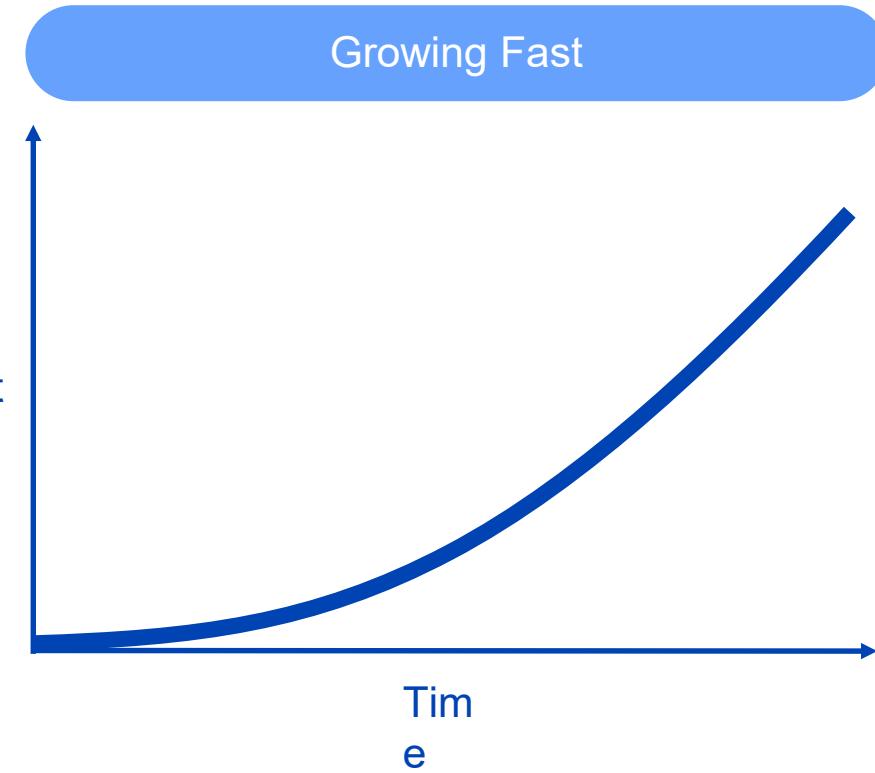
Workload Patterns in Cloud Computing

- Typical compute pattern
 - On & off workloads (e.g. batch job)
 - Wasted Capacity
 - Time to market can be cumbersome



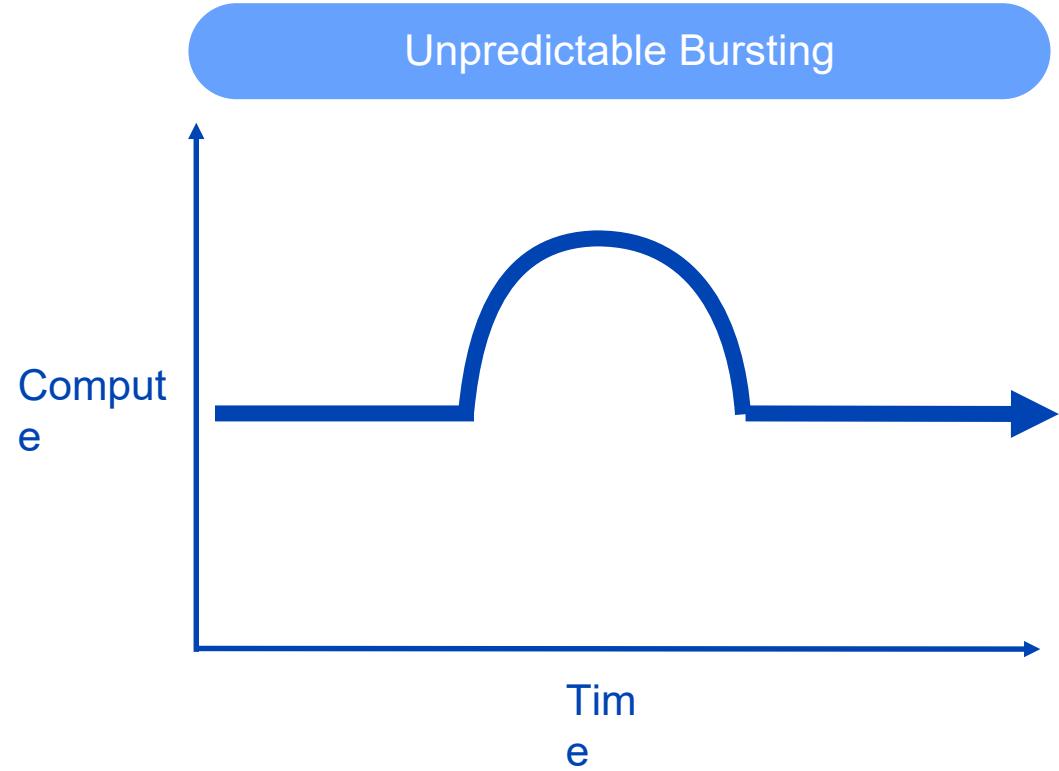
Workload Patterns in Cloud Computing

- Rapidly growing company
 - Major challenge for IT dept. to keep up with growth
 - Potential loss of business opportunity
 - Potential customer service problems



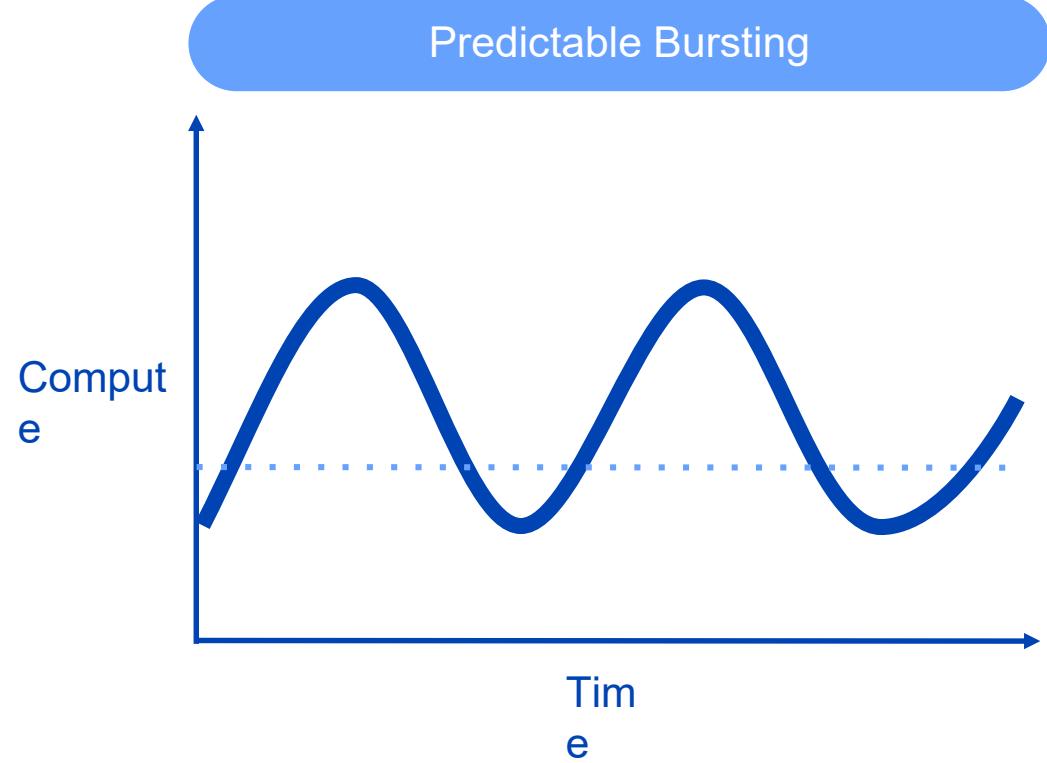
Workload Patterns in Cloud Computing

- Unexpected Events
 - Unexpected peak in demand
 - Loss of business opportunity
 - Wasted capacity if demand wanes

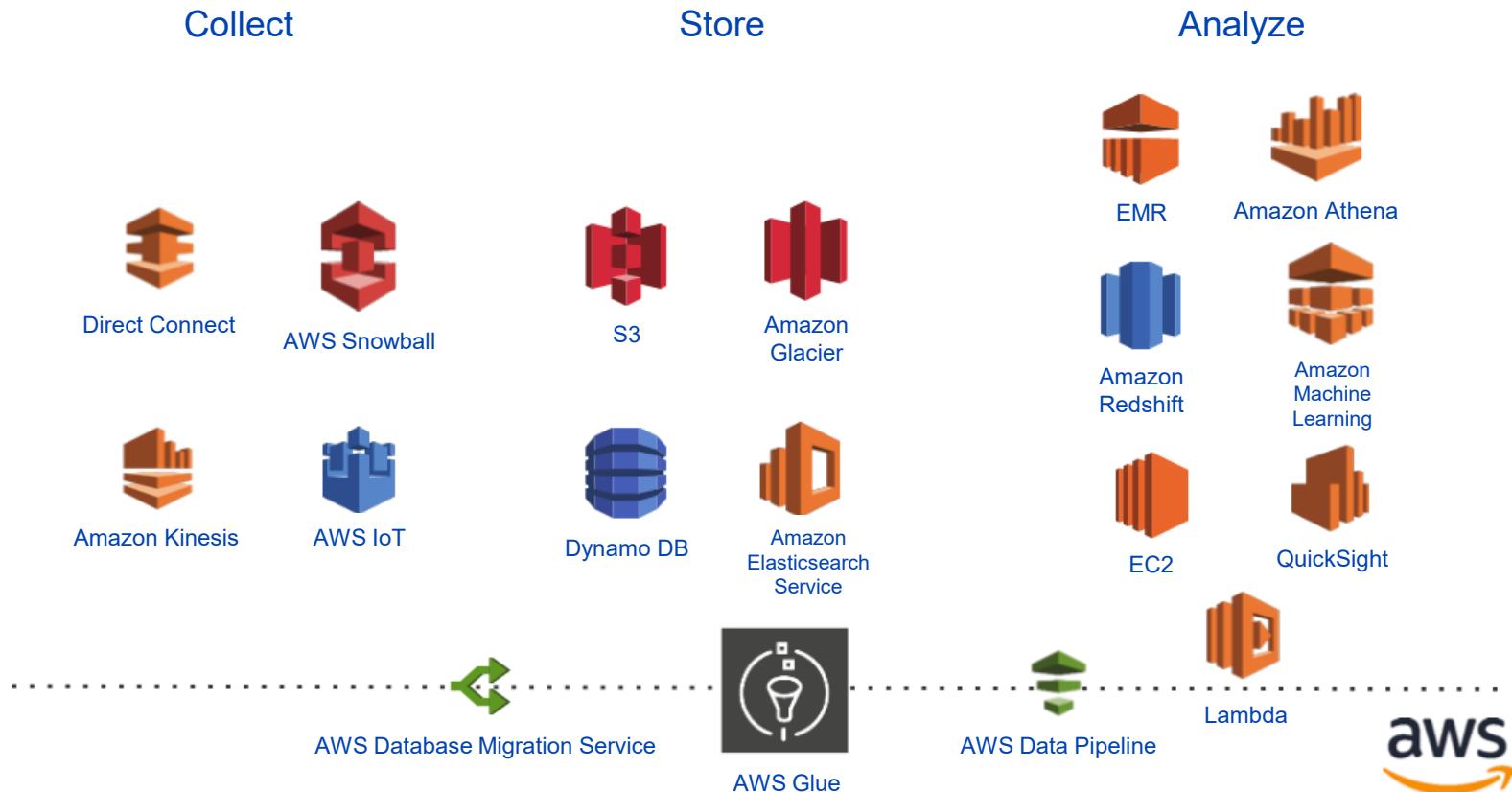


Workload Patterns in Cloud Computing

- Seasonal Peak Computing
 - Seasonal peaks and troughs
 - Provisioning dilemma
 - Wasted capacity or Loss of business

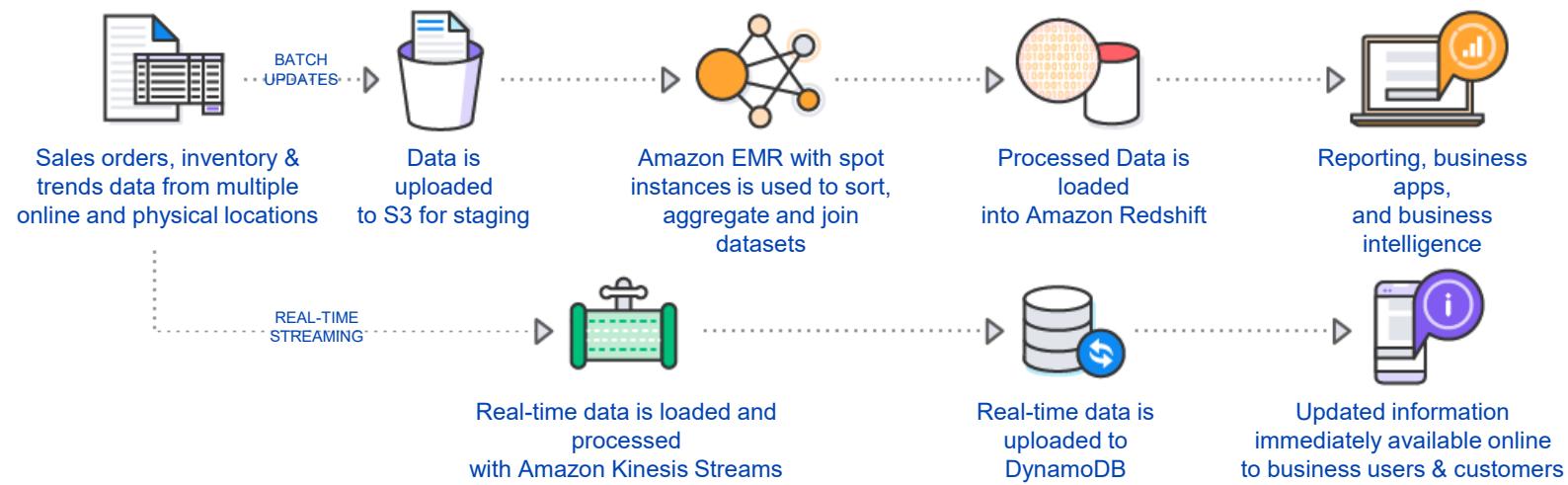


Big Data Services on Amazon AWS



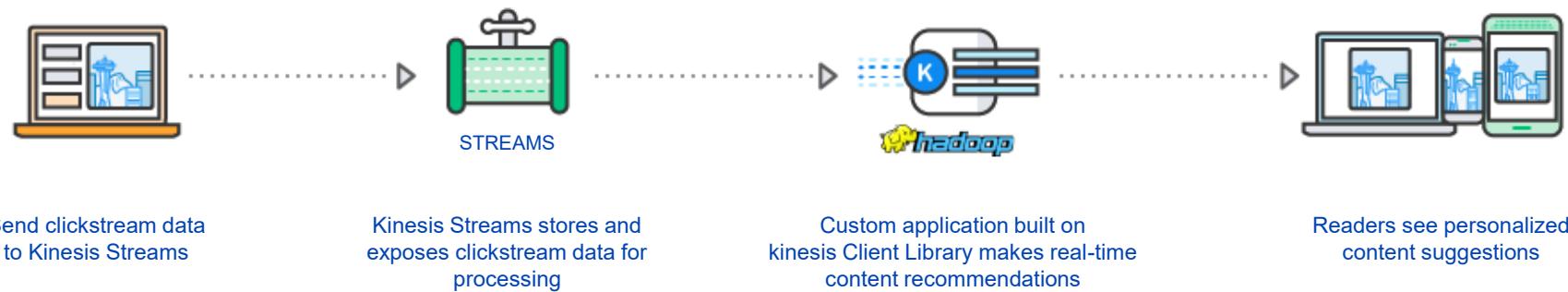
Big Data on Amazon AWS

- On-demand Big Data analytics



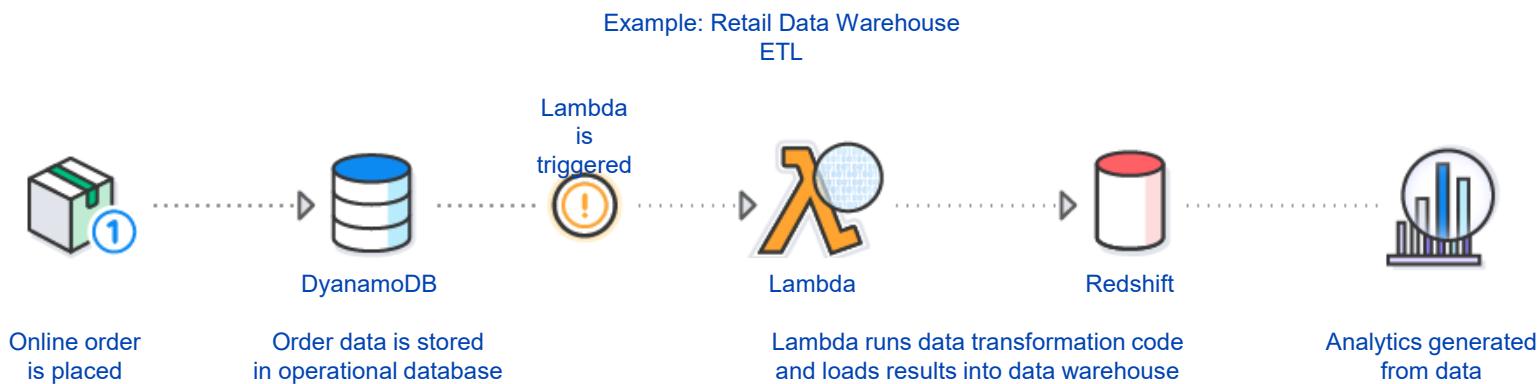
Big Data on Amazon AWS

- Click-Stream Analysis



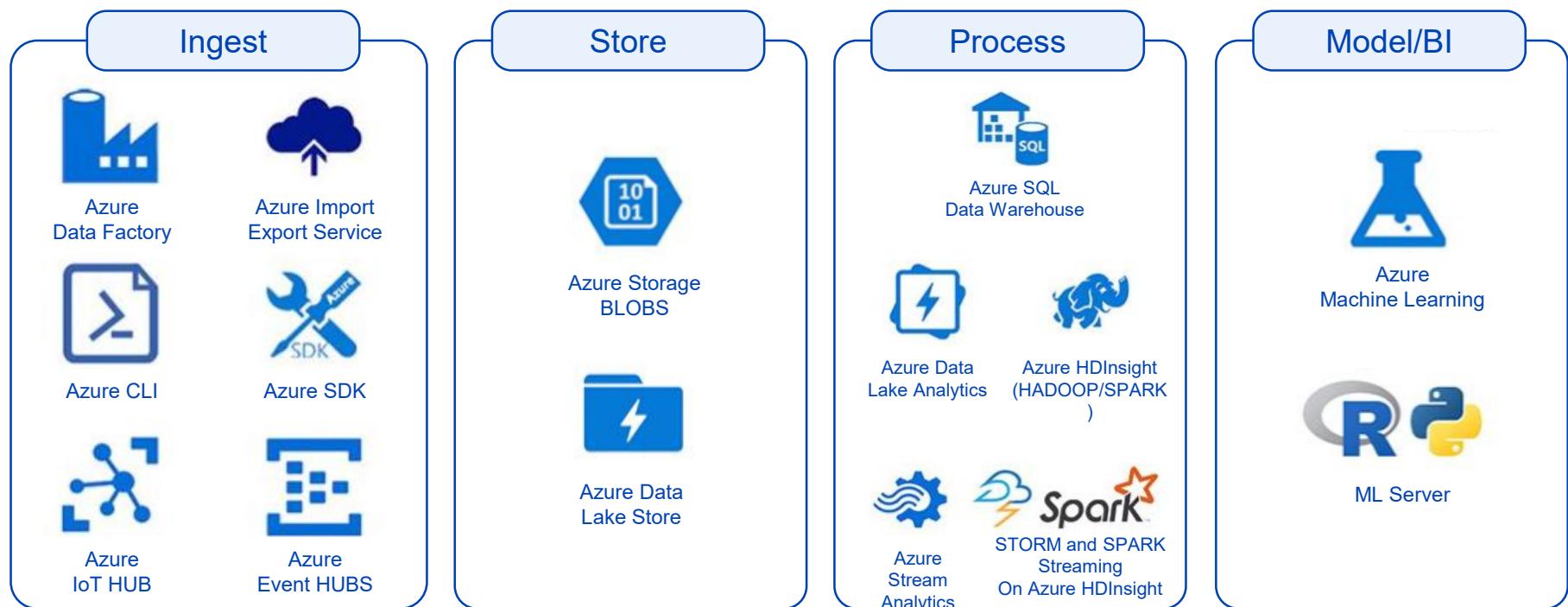
Big Data on Amazon AWS

- Event-driven Extract, Transform and Load



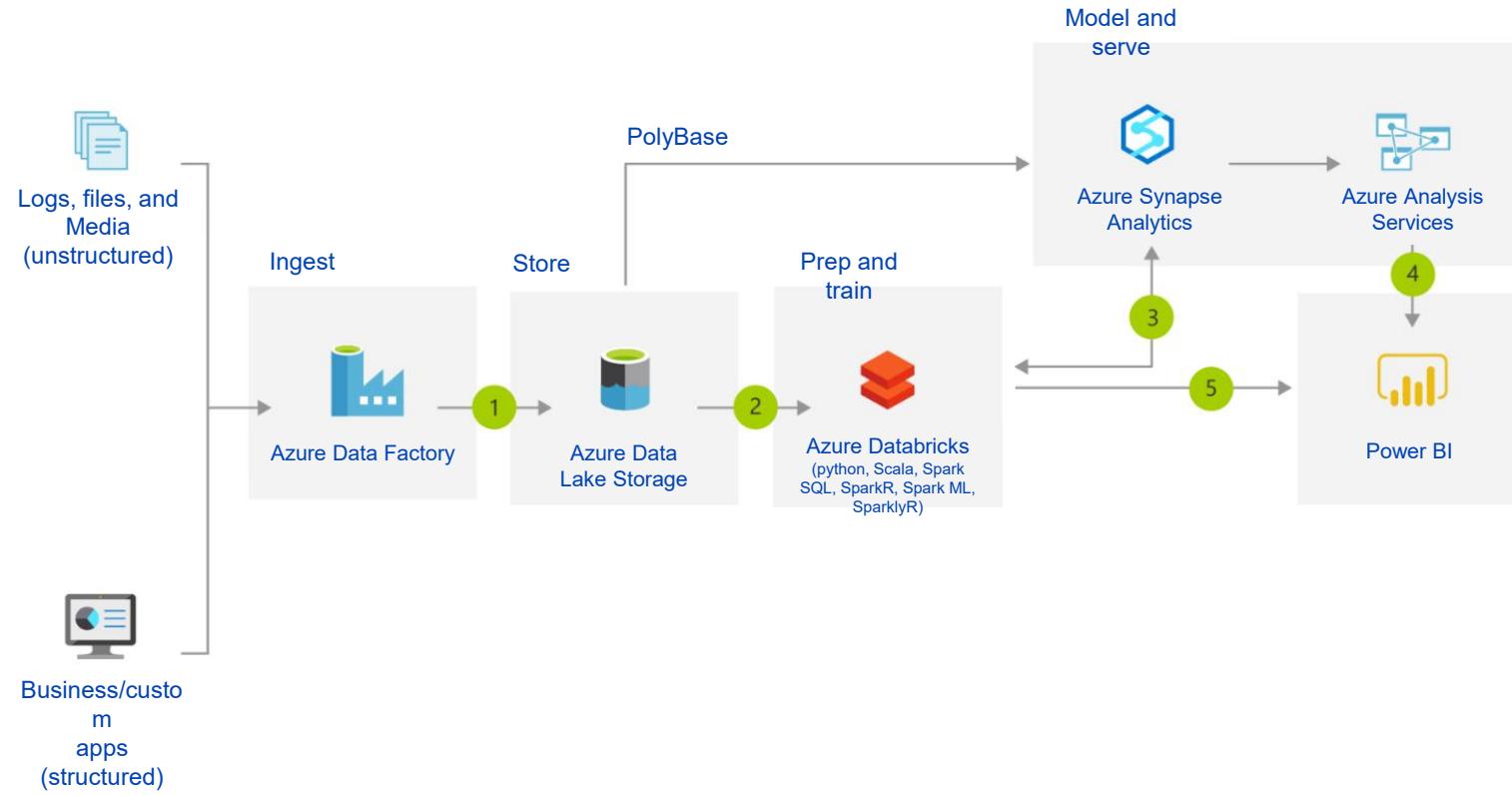
Big Data on Microsoft Azure

- Big Data Tools in Microsoft Azure



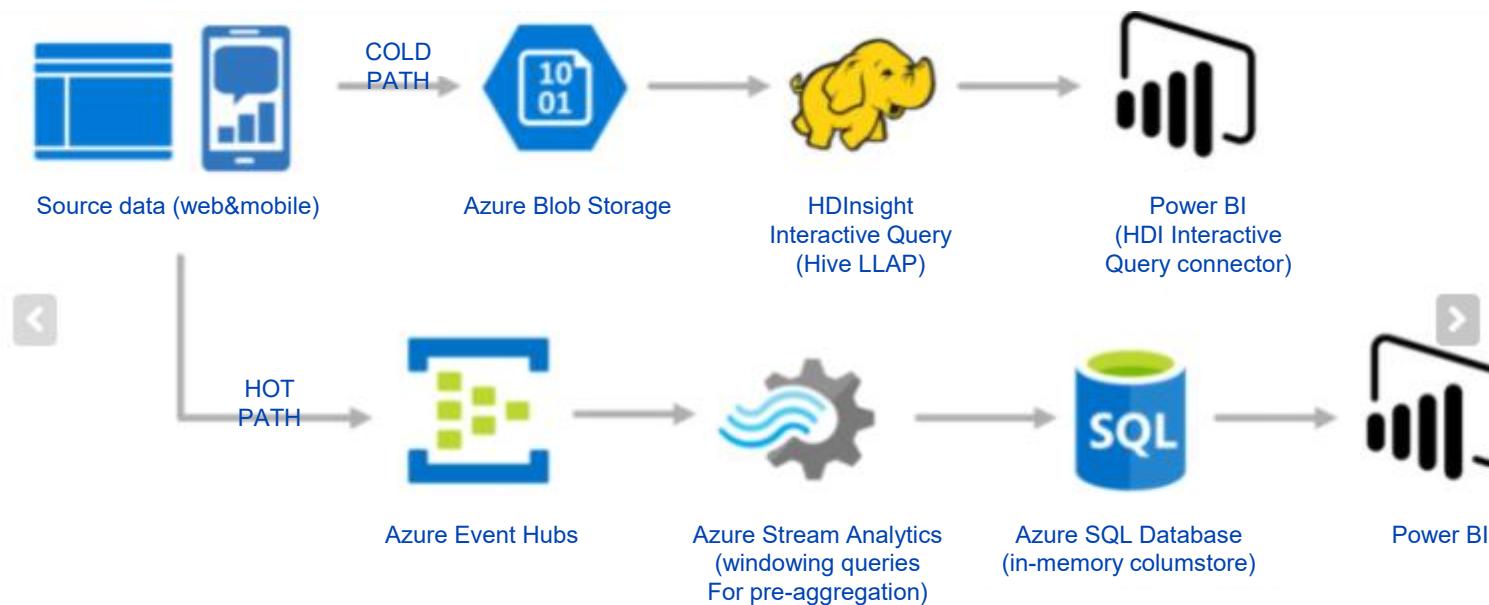
Big Data on Microsoft Azure

- Pre-processing and Data Analytics



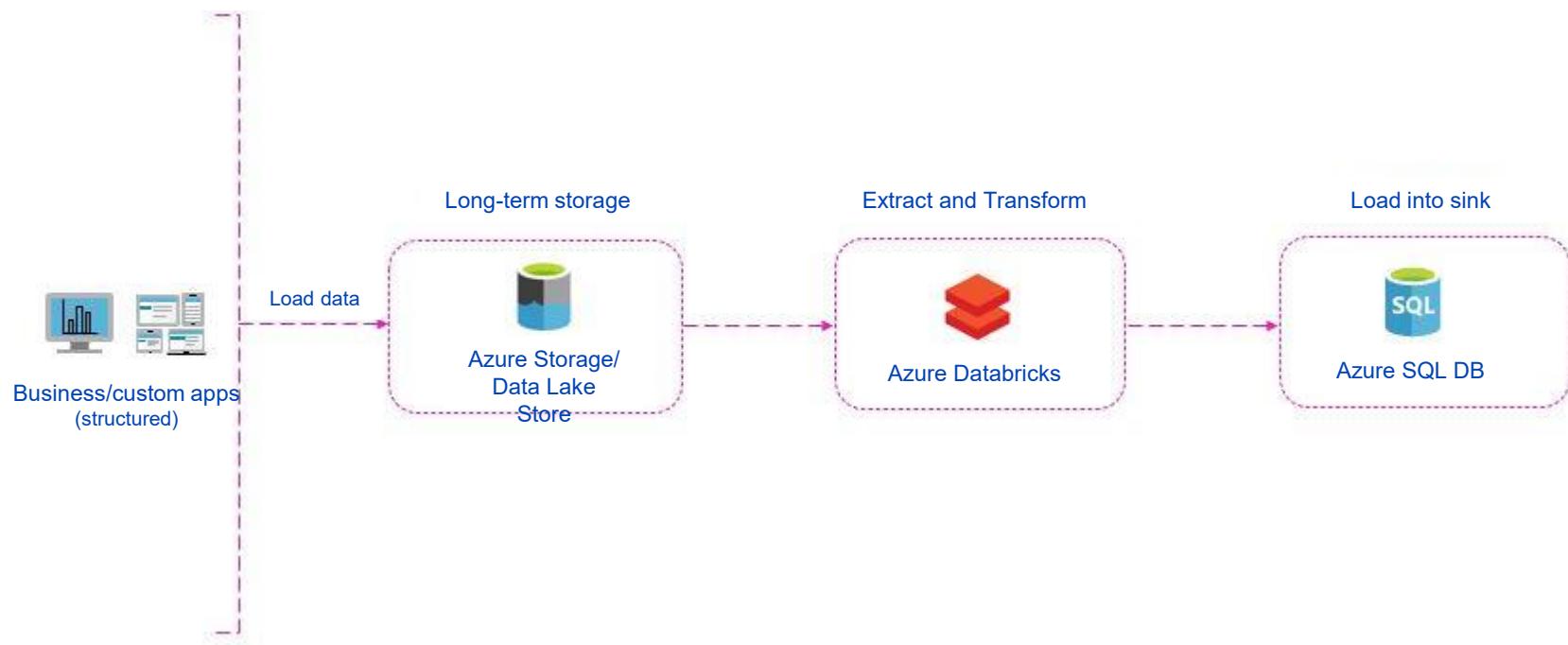
Big Data on Microsoft Azure

- Simultaneous Real-time and Batch processing



Big Data on Microsoft Azure

- Extract Transform and Load on MS Azure



[Demo]

AWS Instructor Demo



Unit 2.

Hadoop Core & Eco system overview

| Fundamentals of Big Data

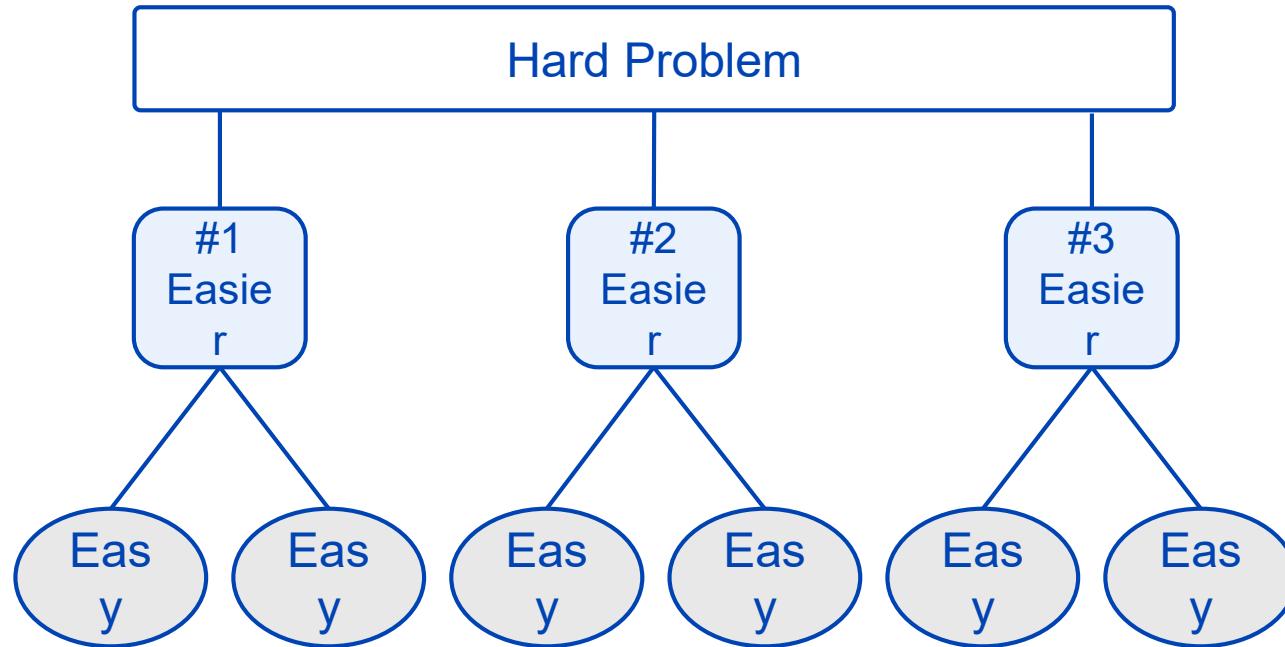
Unit 2.

Hadoop Core & Eco system overview

- | 2.1 Apache Hadoop & Spark platform overview
- | 2.2. Bigdata pipeline overview

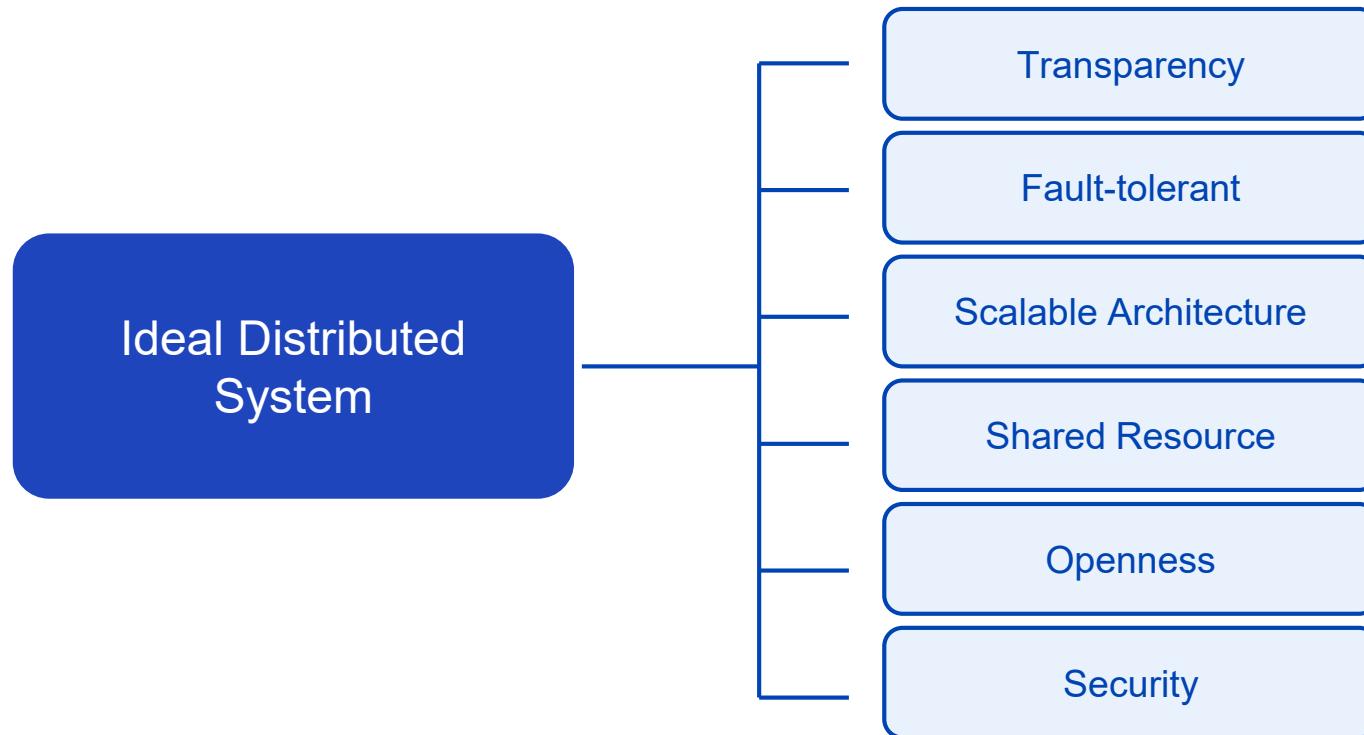
Distributed Parallel Processing for Big Data

- Solve big problems by breaking them down to smaller problems



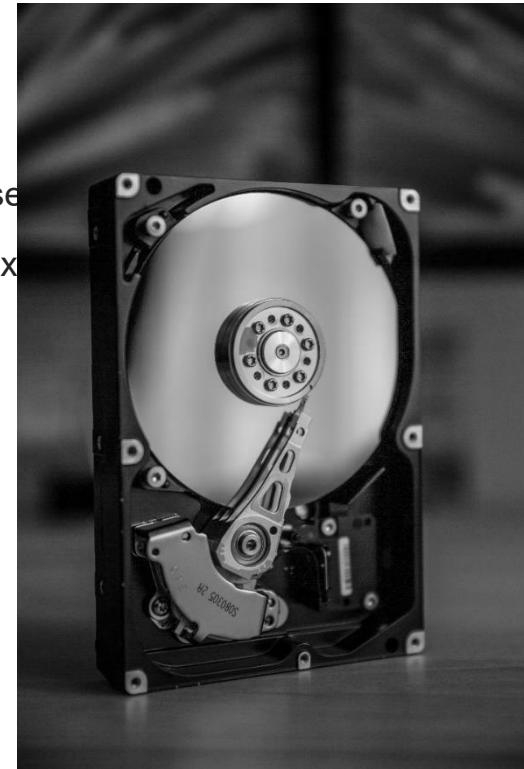
Challenges of Distributed Systems

- Which characteristics should must an Ideal Distributed System have



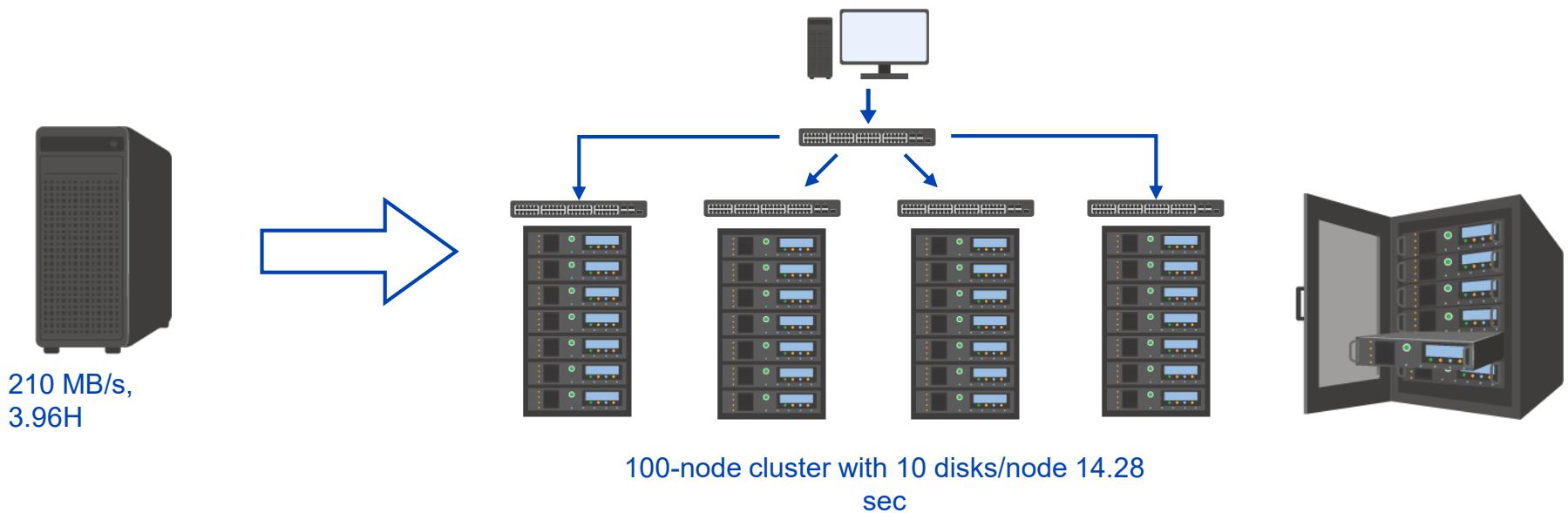
The Disk Bottleneck

- Disk Performance stagnate
 - Disk throughput in 1997->2015: 16.6 MB/sec -> 210 MB/sec
 - While disk prices have been dropping exponentially, disk performance has seen little improvement
 - Reading 1 terabyte of at an average speed of 100 MB/sec would take approximately 10 hours



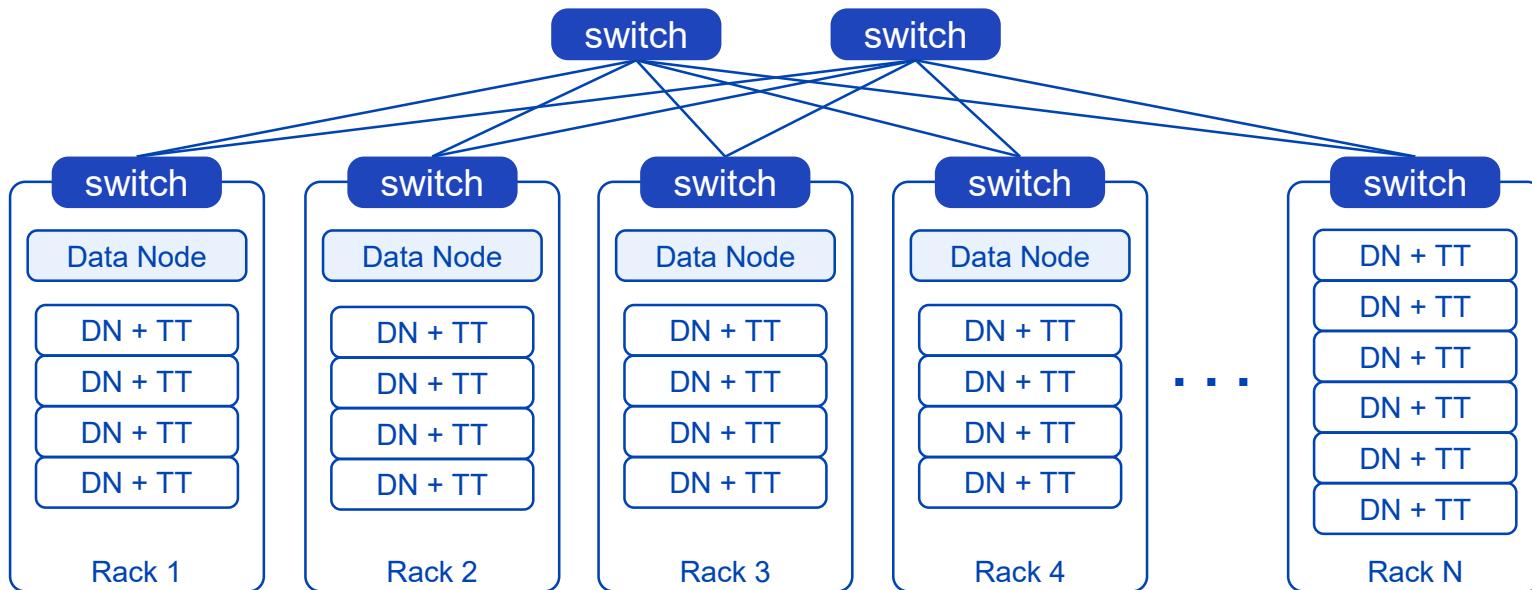
Optimal Parallel Disk Read

- A single disk to parallel disk architecture
 - Transfer rate of one disk at 210 MB/sec -> 4hours for 3TB
 - 1000 disk in parallel cluster can transfer 3TB in 15 secs
 - In this case, 100 nodes with 10 disks per node worked parallelized.

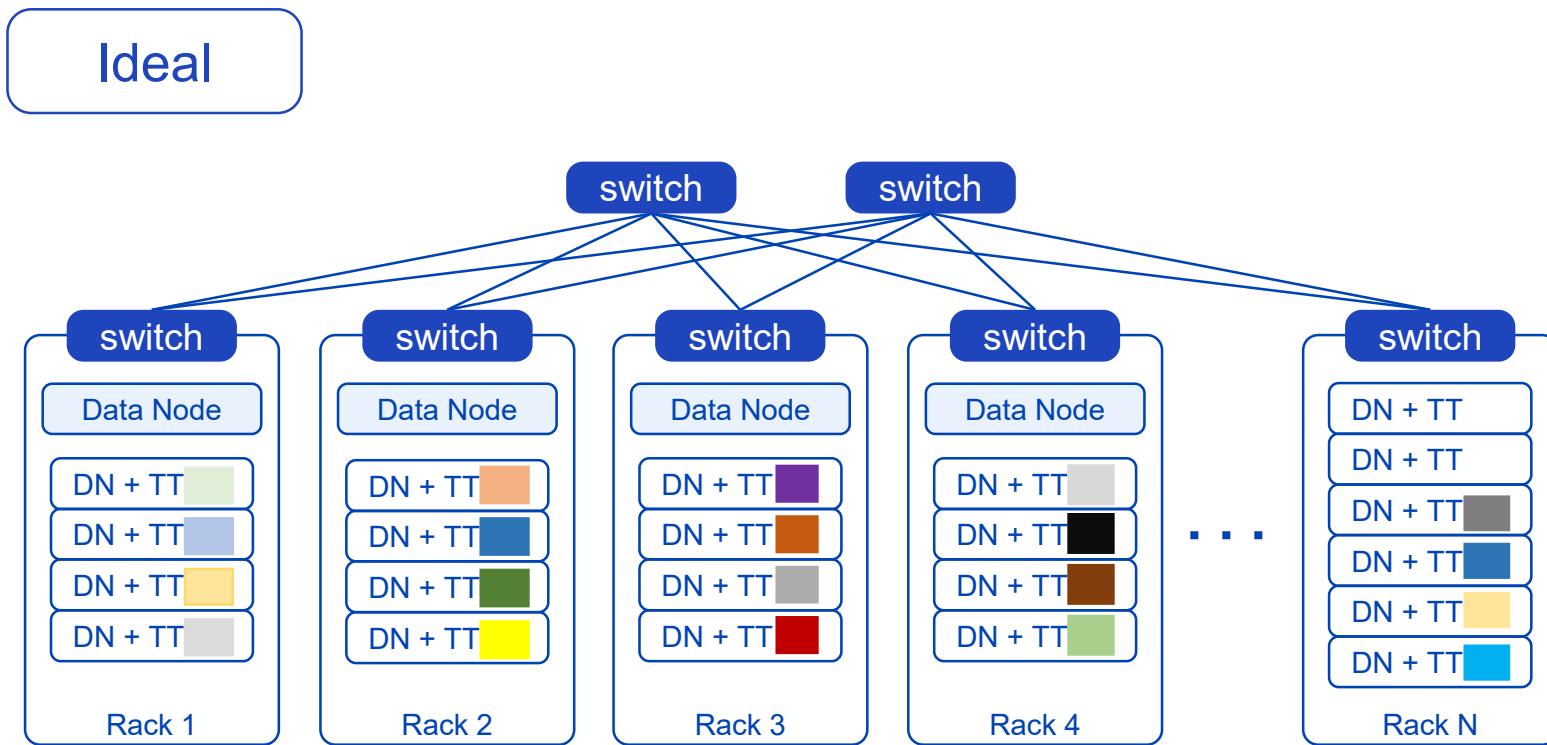


Ideal .vs. Realistic Disk Access Pattern

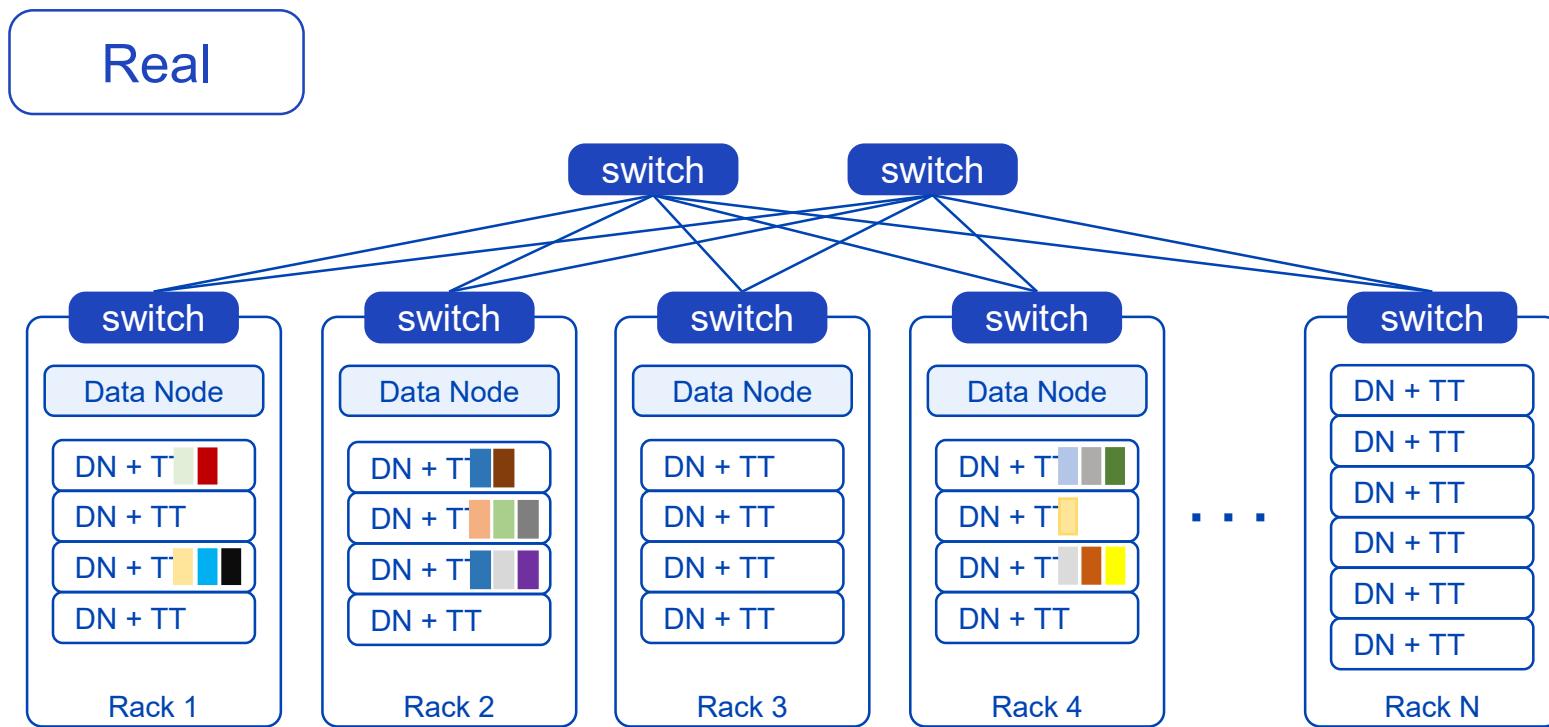
- A more realistic perspective of partitioned block distribution
 - Partitioned data blocks are never distributed in an ideal fashion
 - Full concurrent reads could only occur if each data block was on separate server



Ideal .vs. Realistic Disk Access Pattern



Ideal .vs. Realistic Disk Access Pattern

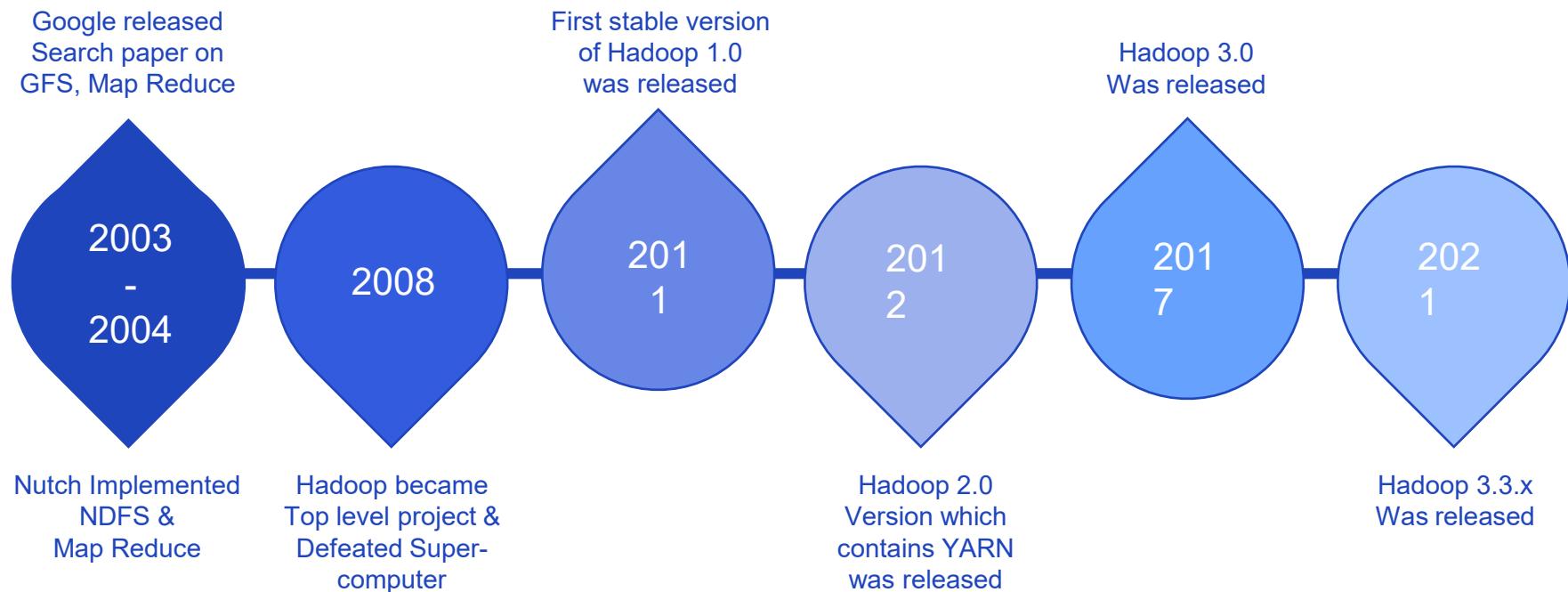


Hadoop solves the challenges

- A new distributed computing paradigm
 - Process the data where it is stored
 - Use software based daemons instead of hardware to solve the challenges of distributed computing
 - Master-Slave architecture where slaves can be added to scale out the platform
 - Provide a platform where developers and users can concentrate on processing the data without worrying about housekeeping tasks related to the platform

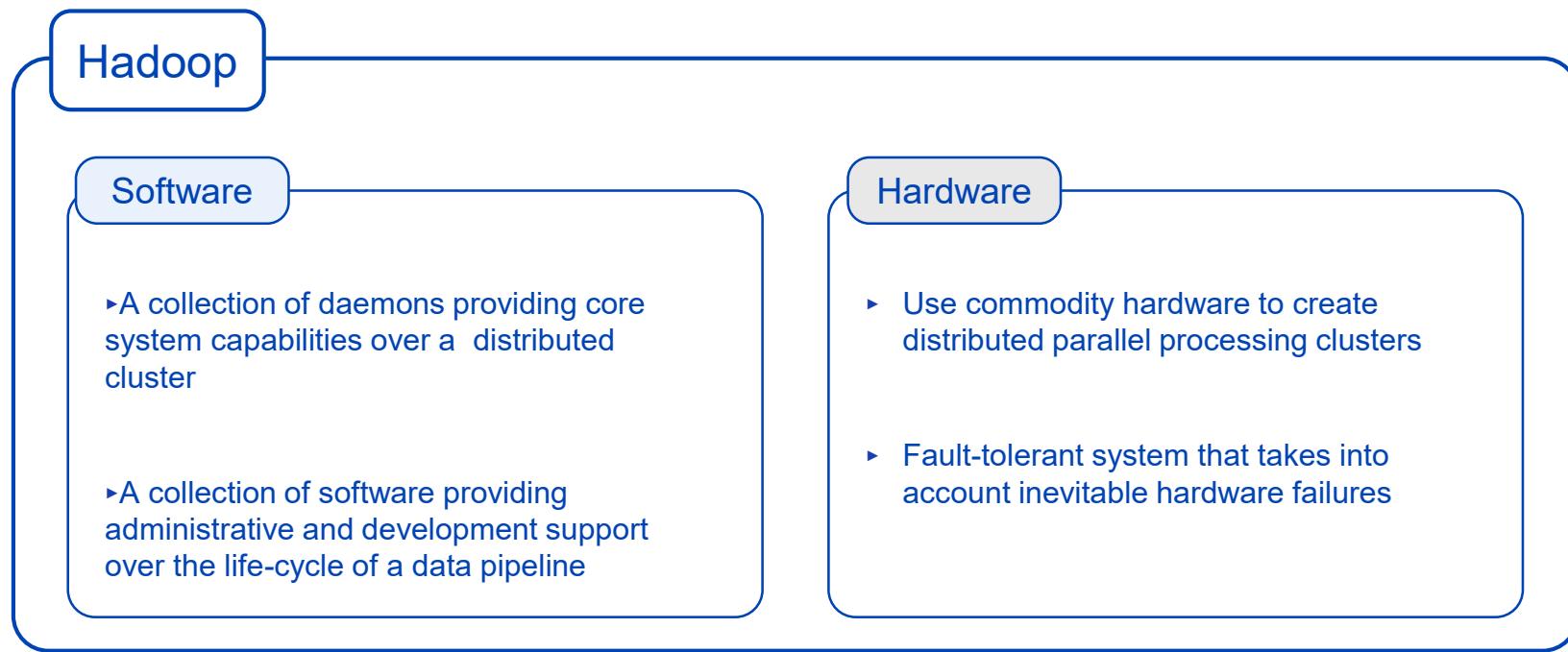
The Origins of Hadoop

- History of Hadoop in brief points.



What is Hadoop?

- A collection of software and hardware to process Big Data



What value does Hadoop provide

- Distributed data storage and processing platform
 - Support better decision making by providing real-time monitoring of business data points
 - Provide advanced data analysis
 - Organizations can fully leverage their data
 - Run on commodity instead of custom architecture

Key features of Hadoop



Flexible



Fault Tolerance



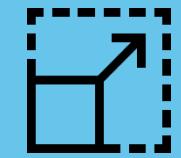
Robust



Fast



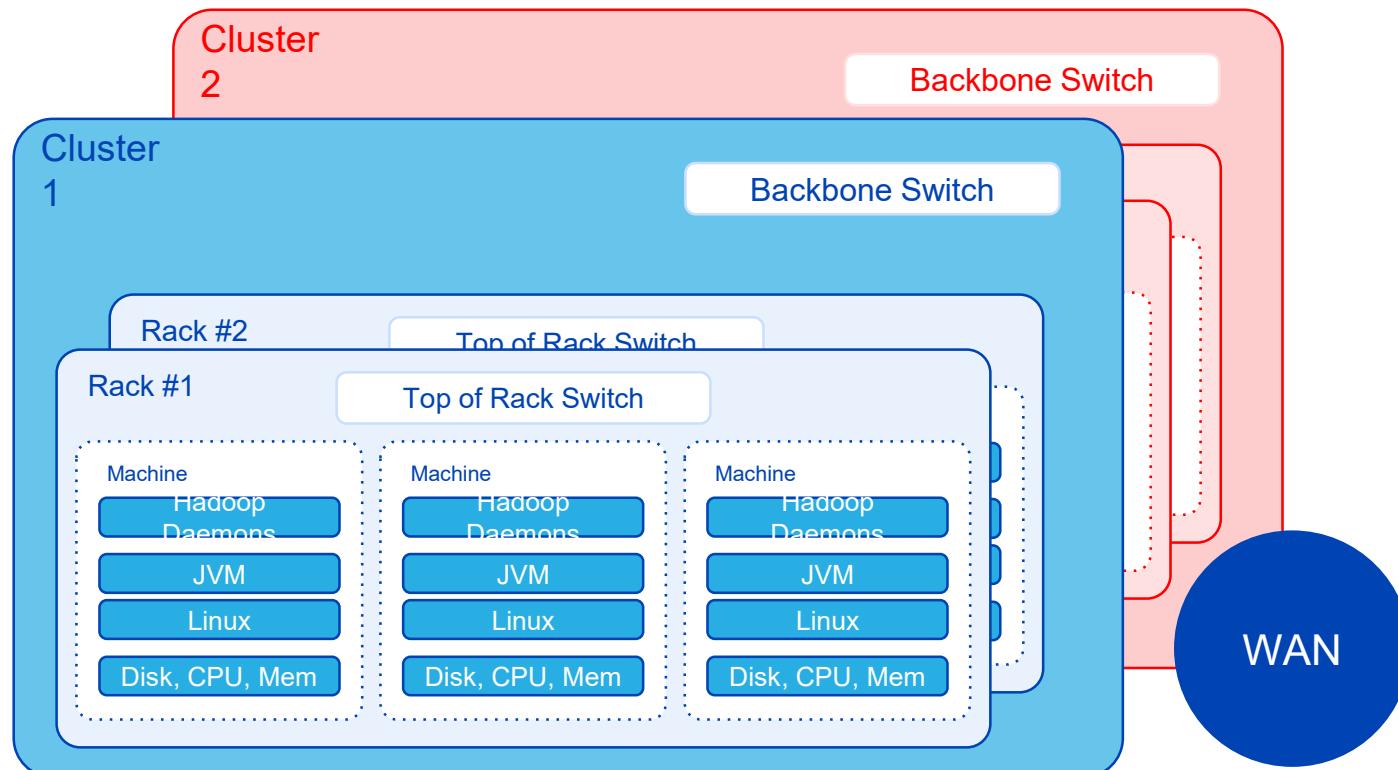
Cost Effective



Scalable

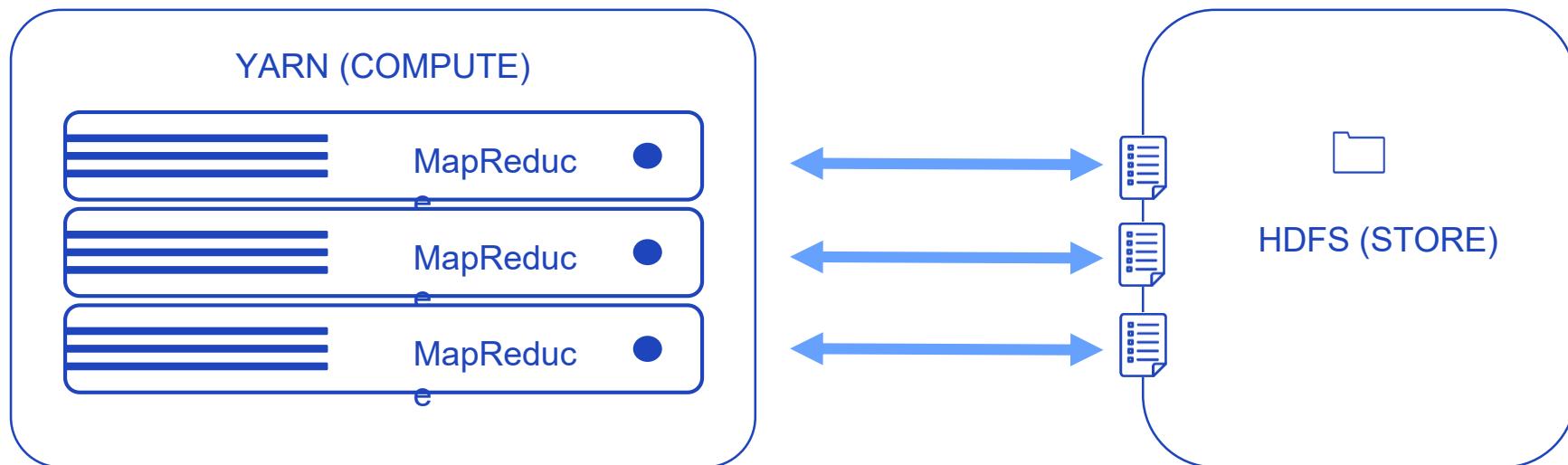
Hadoop Cluster

- A collection of software and hardware to process Big Data



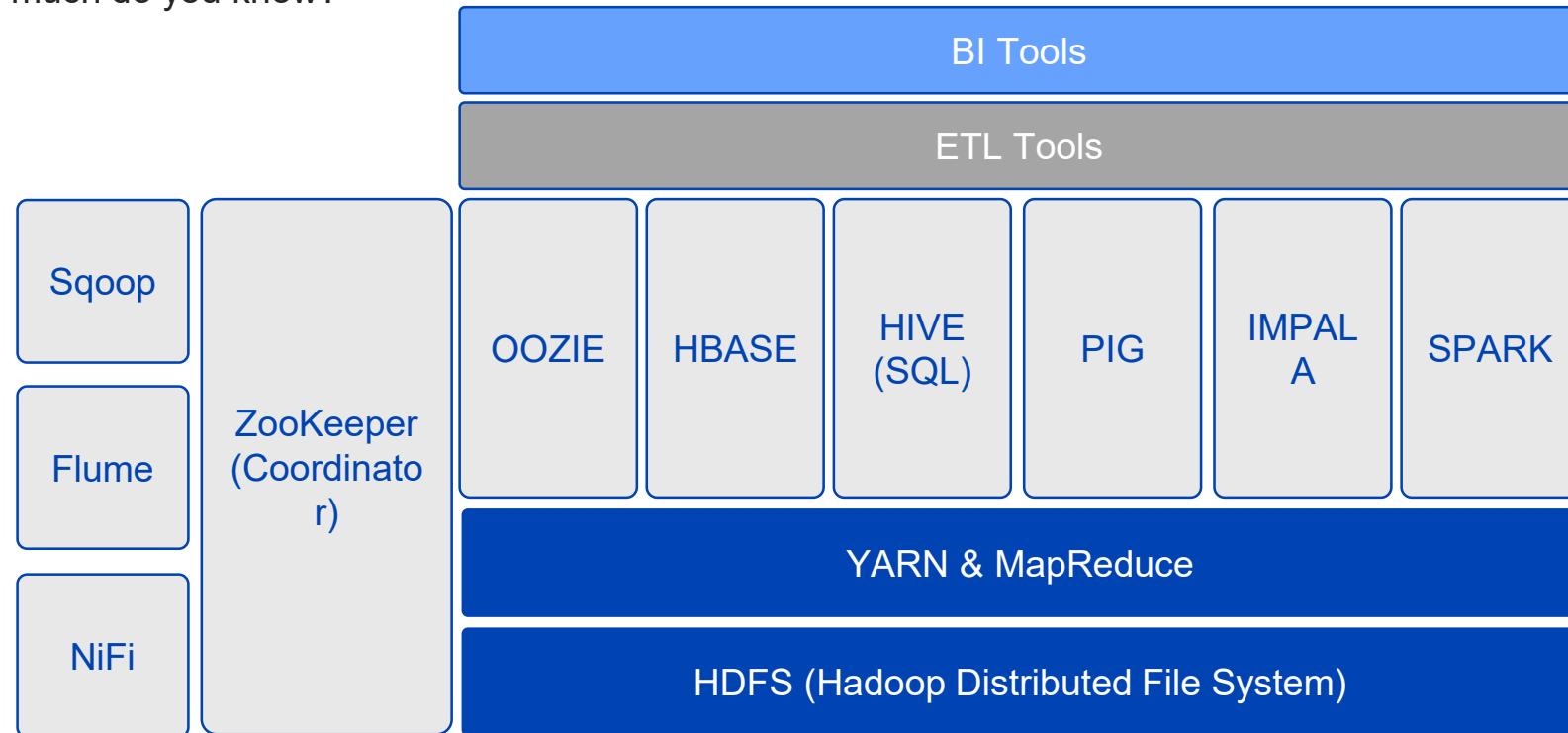
Core Components of Hadoop

- Distributed data storage and processing platform
 - Store component - HDFS
 - Compute component - MapReduce or Yarn



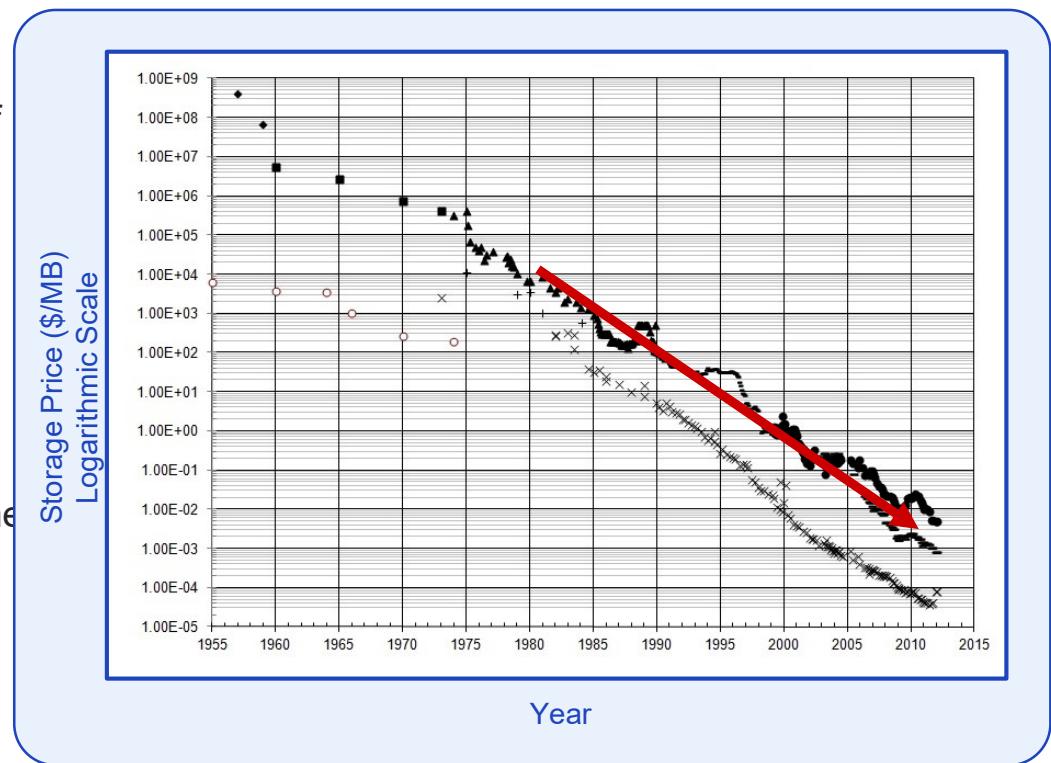
Hadoop Ecosystem

- How much do you know?



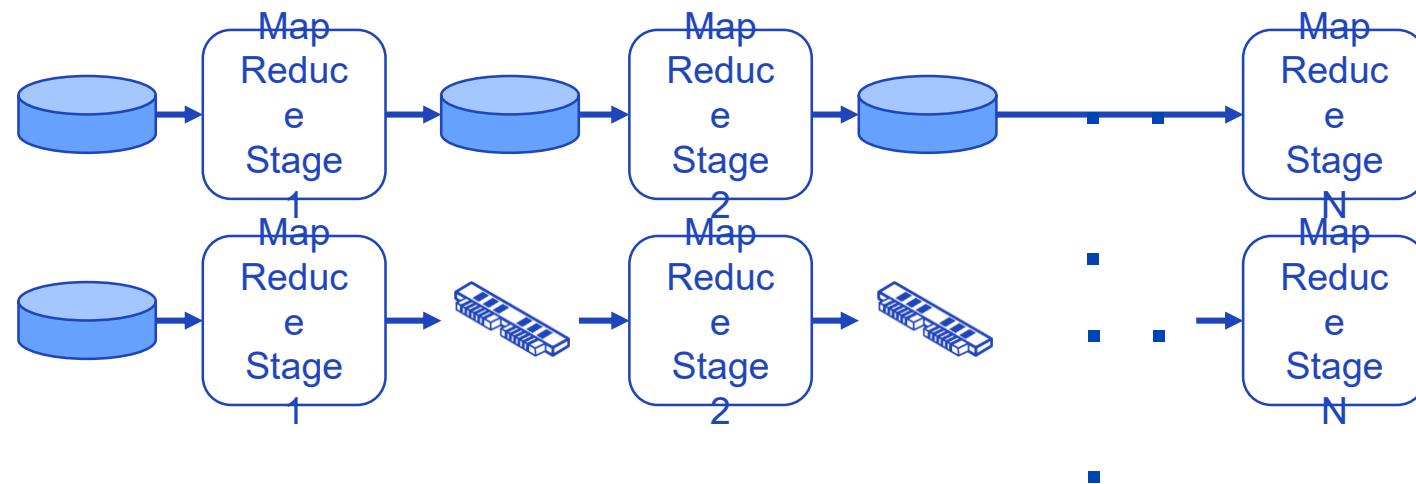
Spark Platform - Opportunity Taken

- Memory cost continues to drop
- New architectural design making heavy use of
- In-memory Computing is born
 - In-memory databases
 - In-memory platforms such as Apache Spark
- Orders of magnitude better performance
- 64 bit operating systems to access far more memory



MapReduce .vs. Apache Spark

- Read and write to memory instead of hard disk
- Perform Disk I/O only at the beginning of computation to load the data into memory
- Distribute the data over all the memory resources in the cluster
- Typically 10~100 times faster



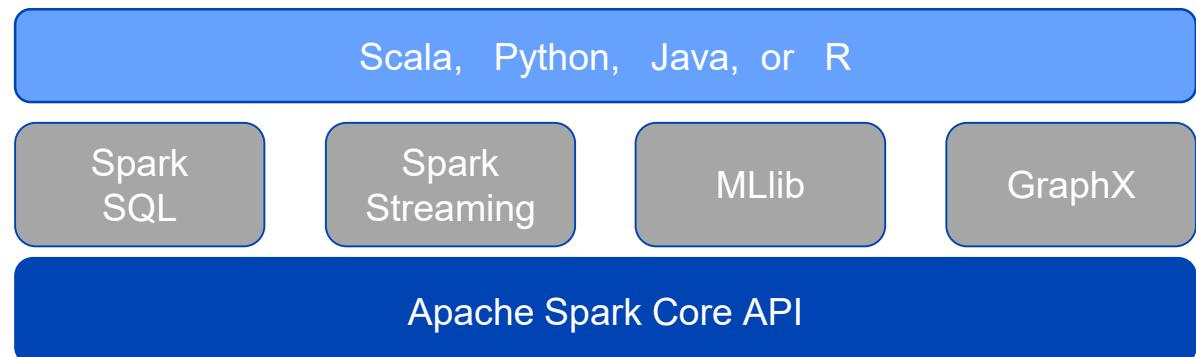
Next Generation Compute for Hadoop

- In-Memory Data Computing
 - 10 to 100x faster than Hadoop MapReduce
- Can integrate with Hadoop and its ecosystems
 - HDFS
 - Amazon S3, HBase, Hive, Cassandra
- Easier Programming API
 - Very powerful high-level API
 - Data pre-processing
 - Good in complex multi-stage applications
 - Machine Learning
 - Interactive query
- Provides Real-time data processing

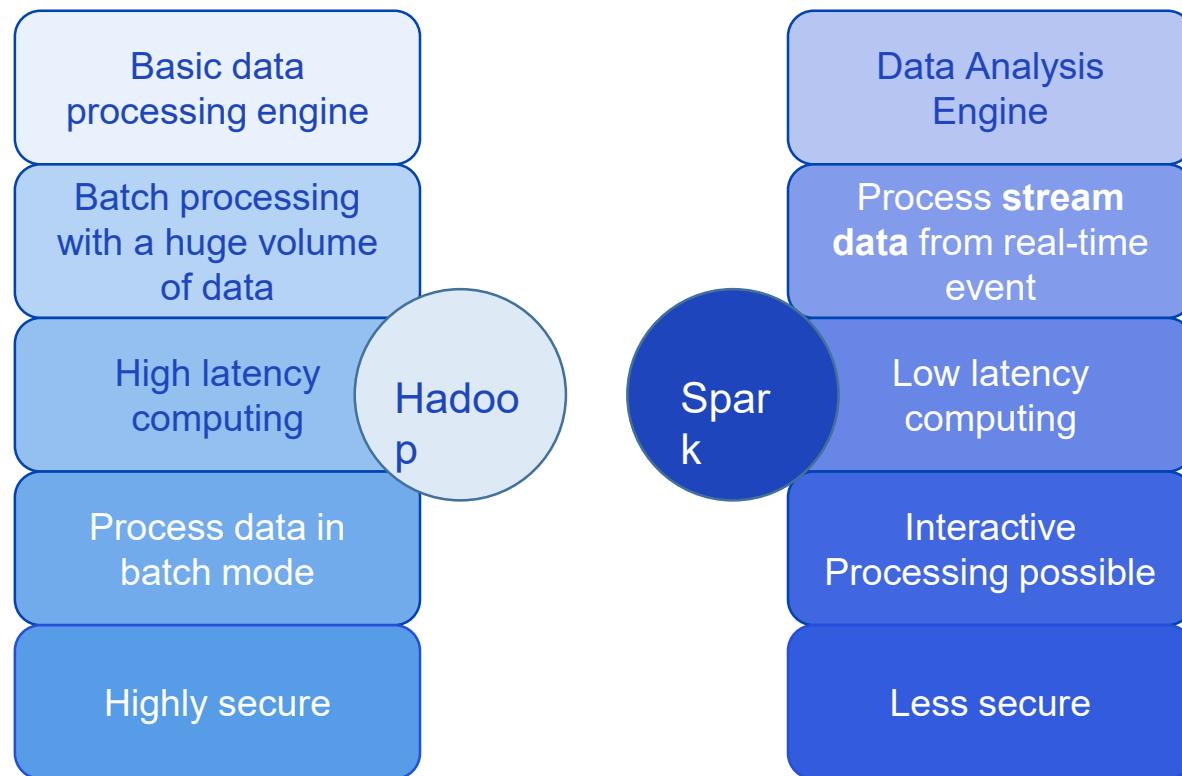


Apache Spark Framework

- Apache Spark provides multiple programming APIs that address different areas
 - Core API
 - Good for processing unstructured and semi-structured data
 - Spark SQL - DataFrame API
 - Good for processing structured data
 - Spark Streaming - Dstream and Structured Streaming API
 - Good for processing real-time data
 - ML and Mllib API
 - Spark machine learning
 - GraphX API
 - Process nodes and edges



Pros and Cons of Hadoop and Spark

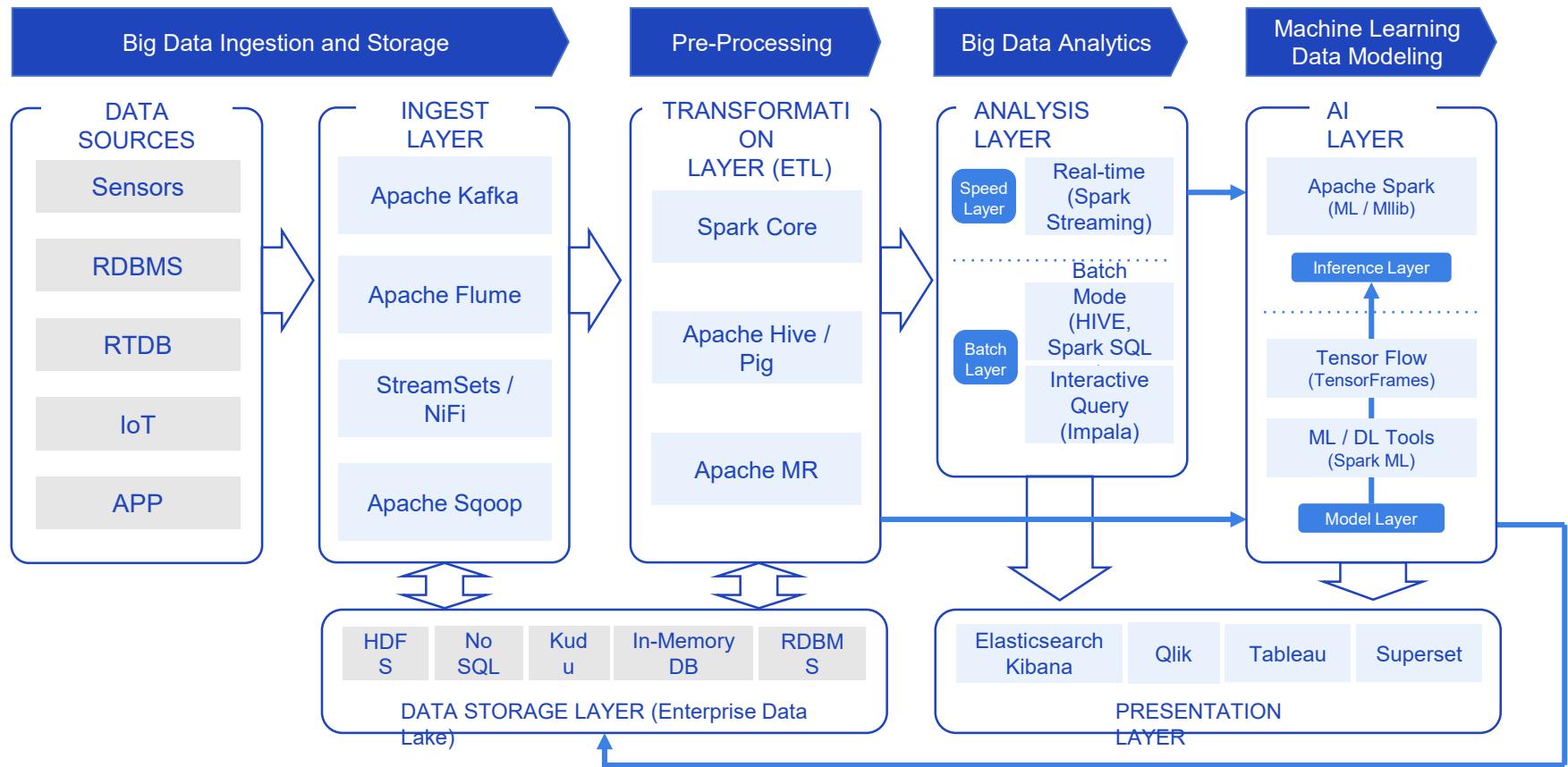


Unit 2.

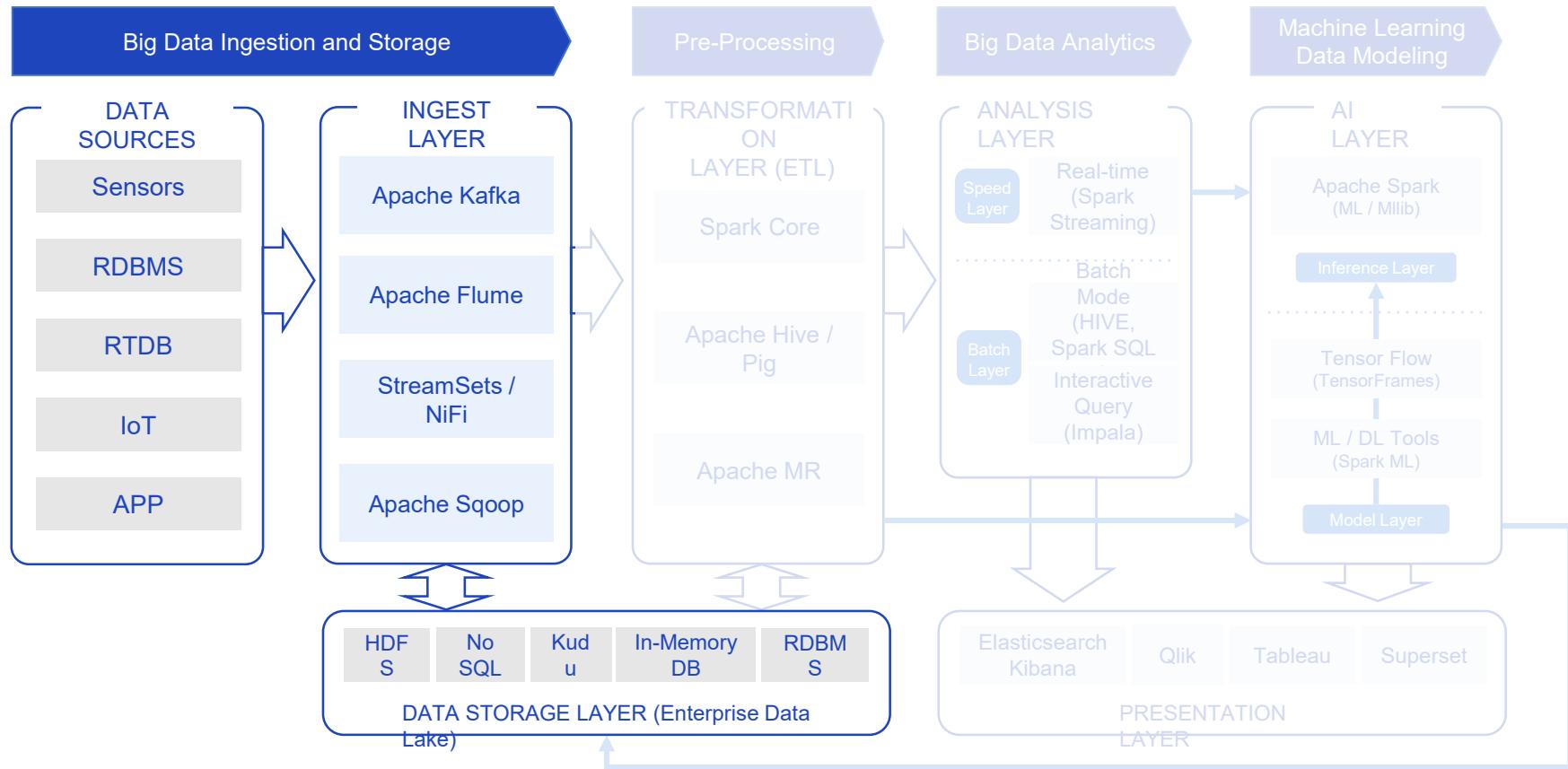
Hadoop Core & Eco system overview

- | 2.1 Apache Hadoop & Spark platform overview
- | 2.2. Bigdata pipeline overview

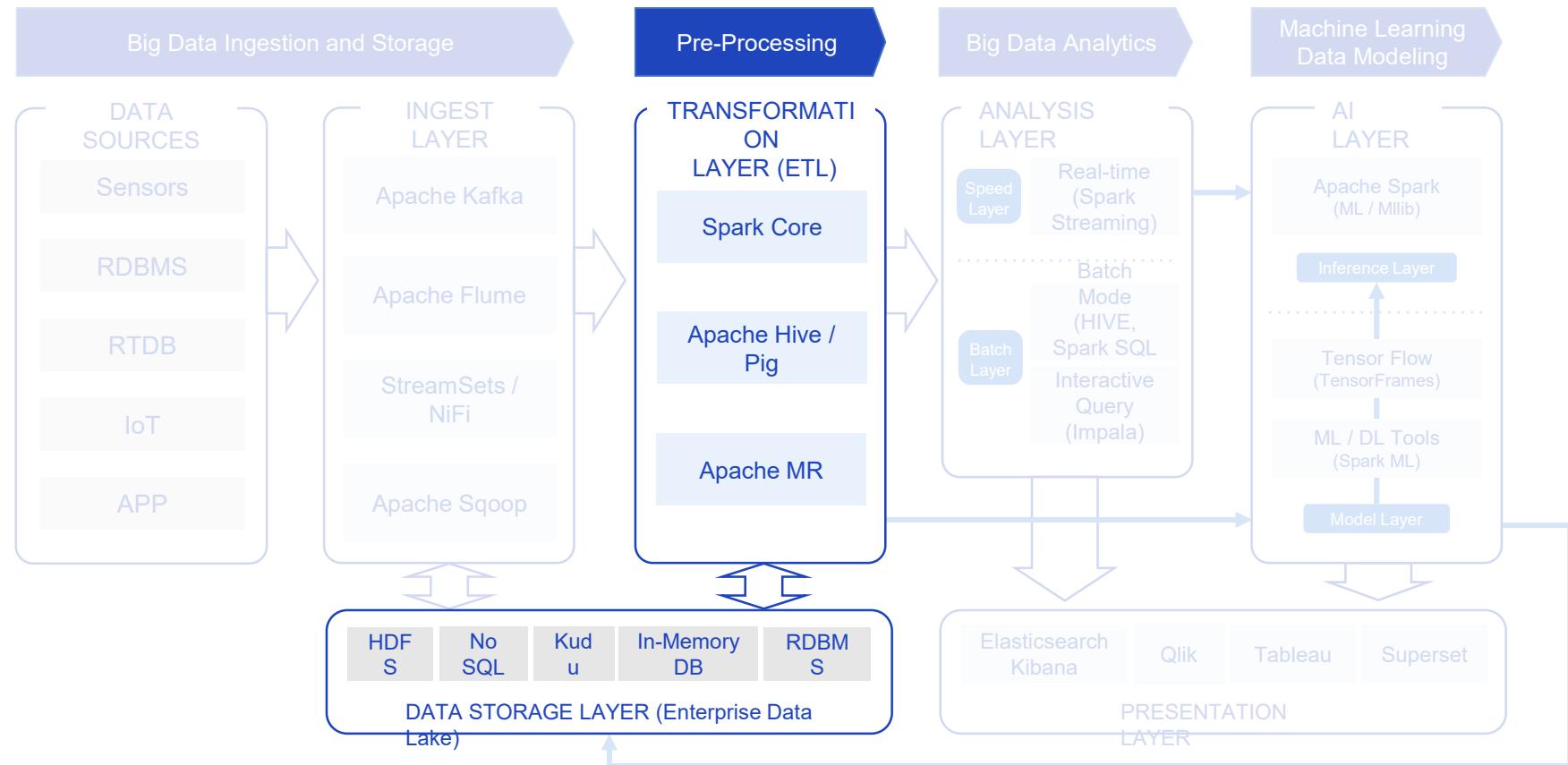
The Data Pipeline for Big Data



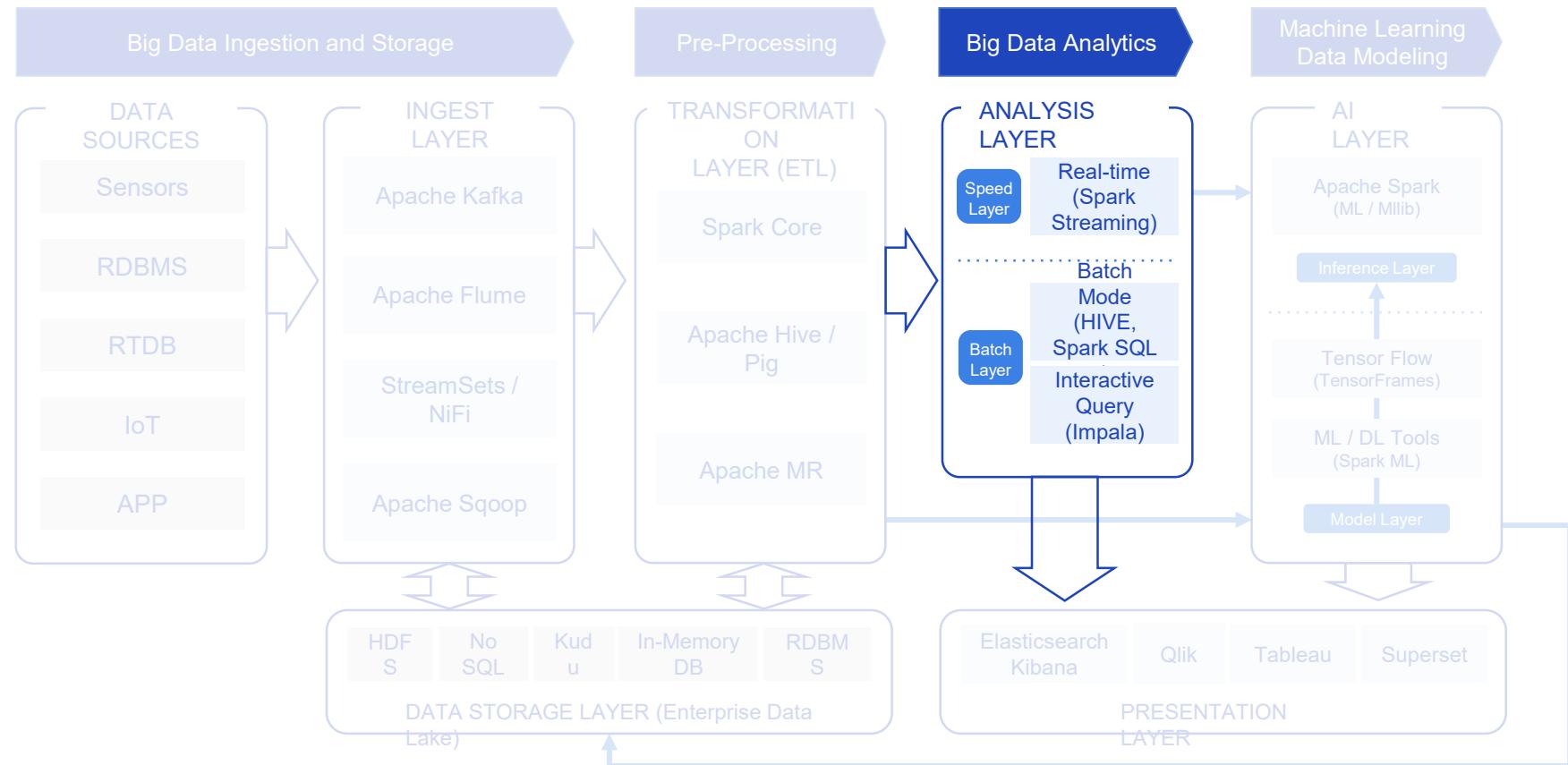
Gathering and Storing Data



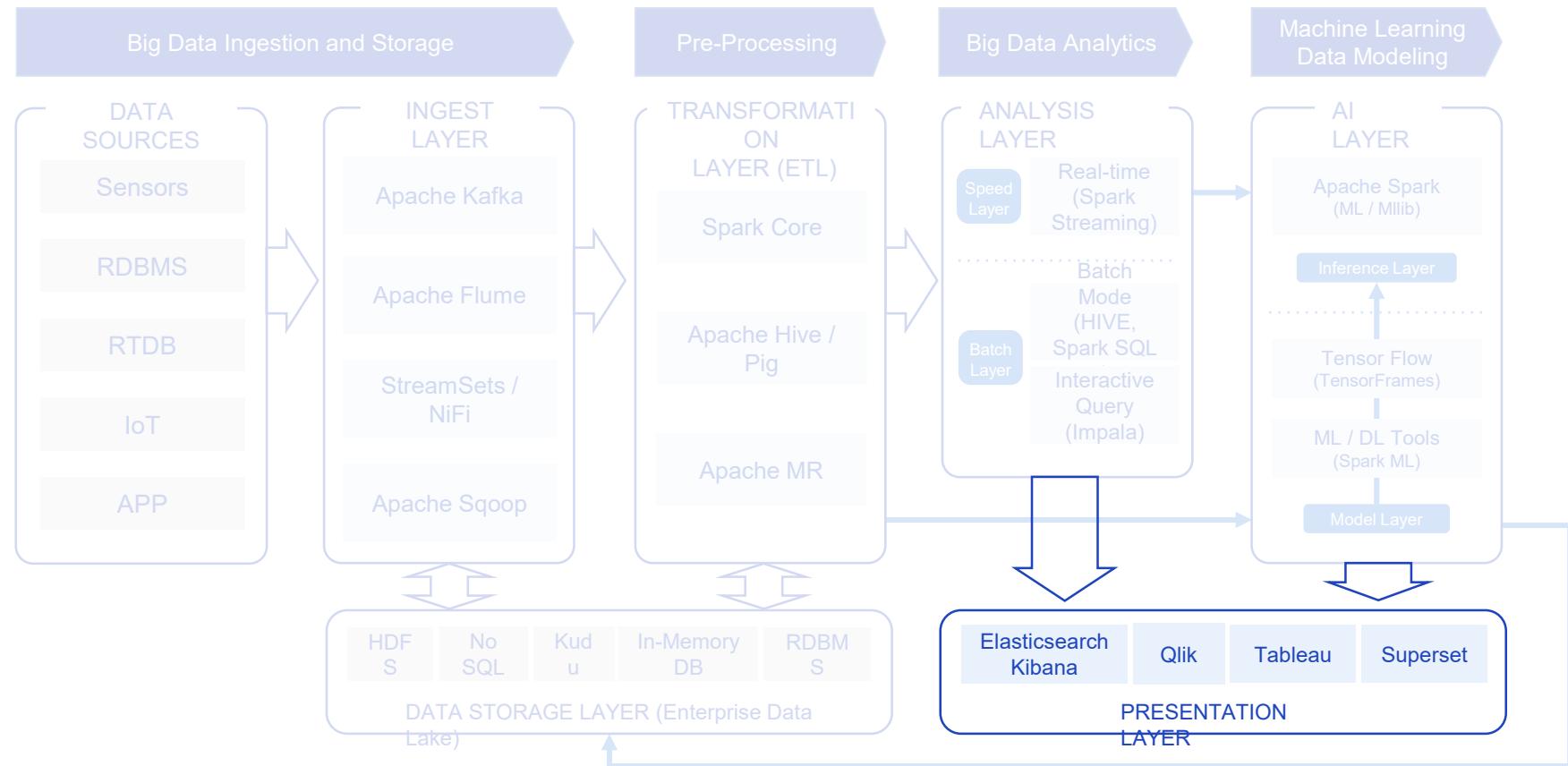
Transforming Your Data



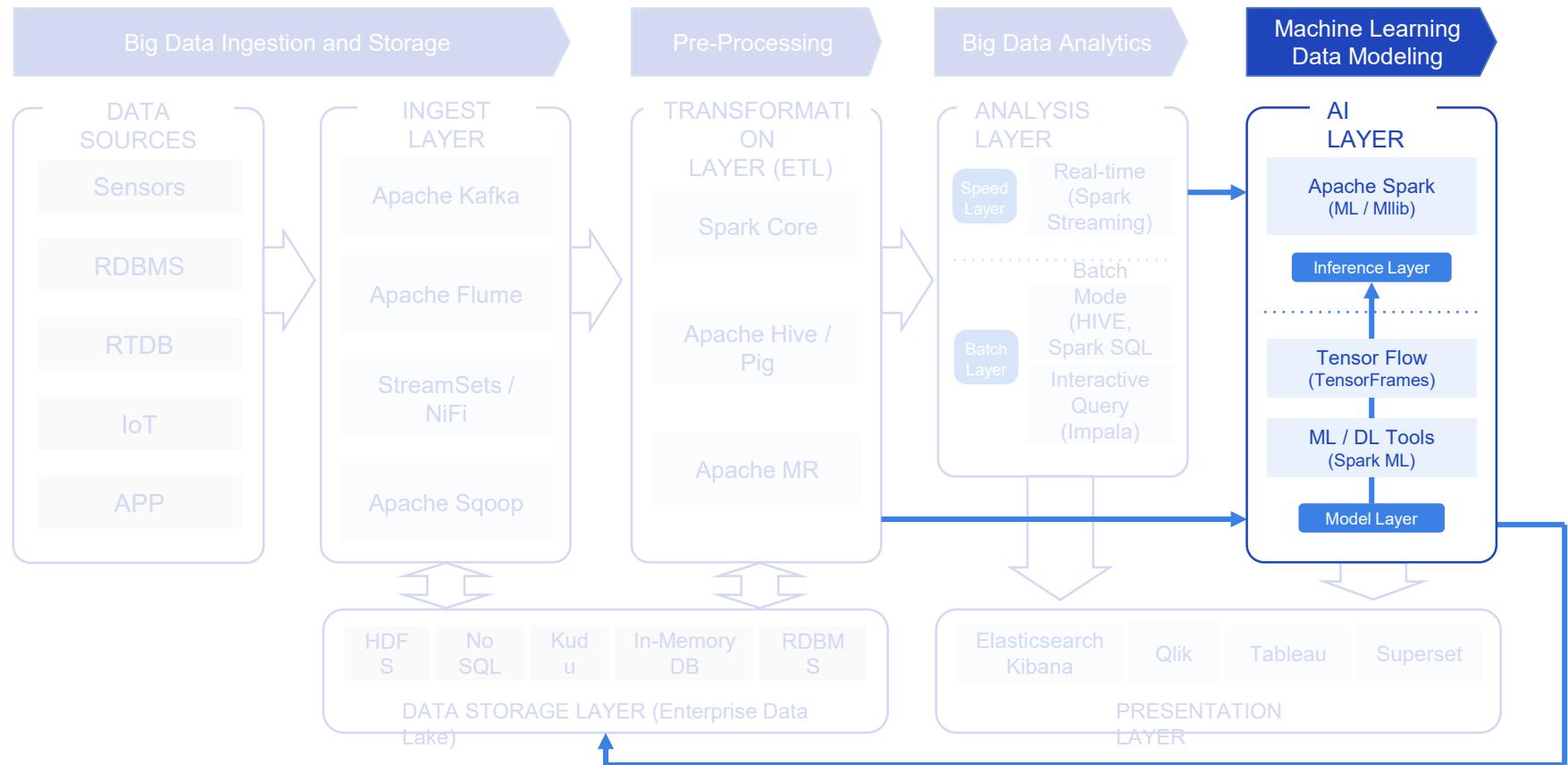
Querying Your Data



Presenting Your Data



Modeling Your Data with AI



Review Questions (1 of 2)

1. As the first platform for big data, what is the name of the collection of software and hardware that processes big data?

1. What is the name of the distributed file system that handles large data sets running on commodity hardware?

Review Answers (1 of 2)

1. As the first platform for big data, what is the name of the collection of software and hardware that processes big data?

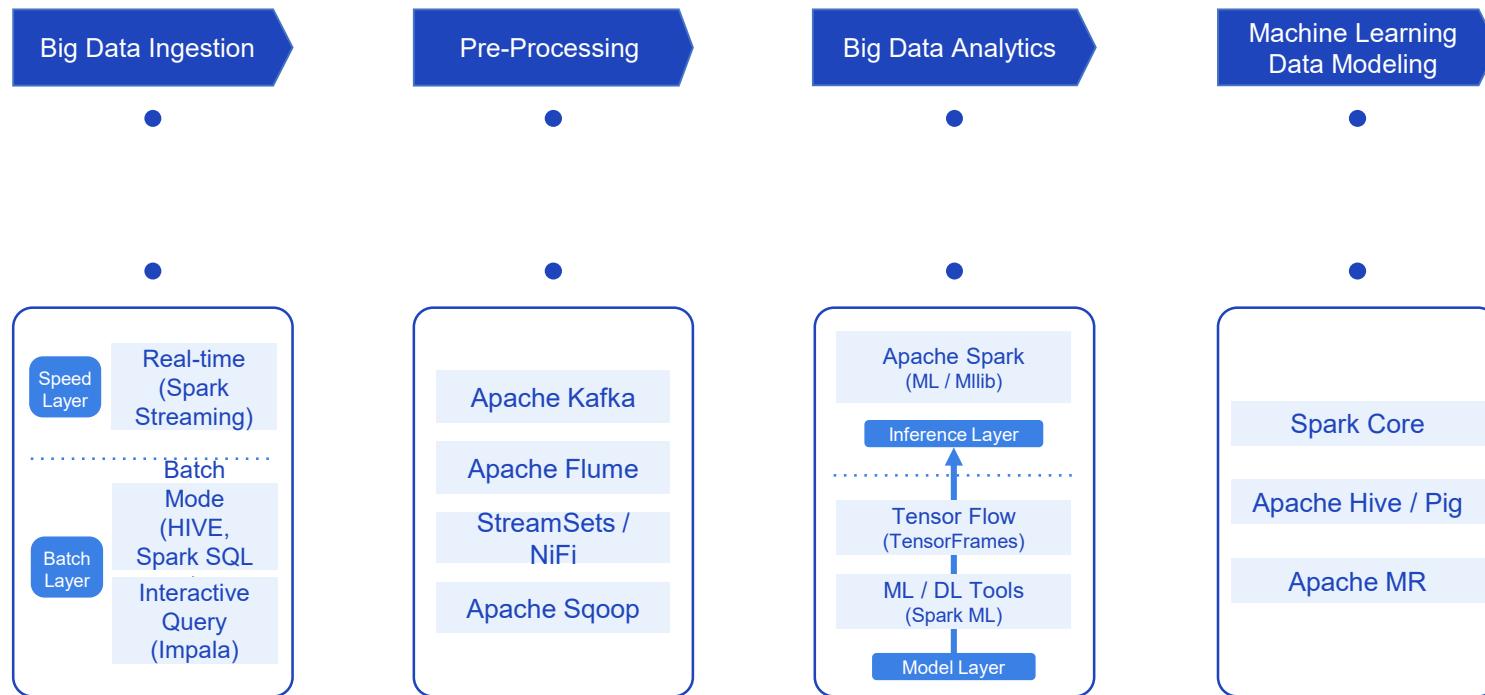
Hadoop

1. What is the name of the distributed file system that handles large data sets running on commodity hardware?

HDFS (Hadoop Distributed File System)

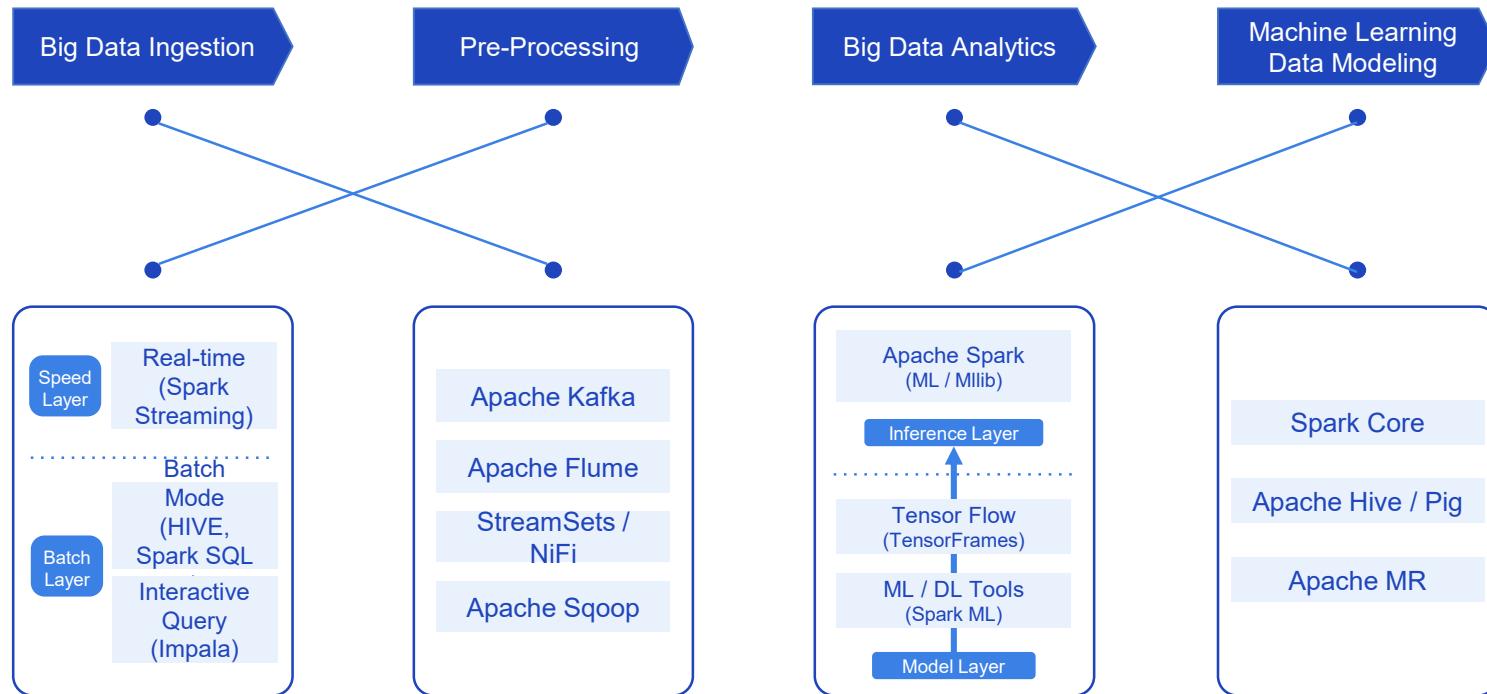
Review Questions (2 of 2)

3. Connect the tools corresponding to the big data pipeline below



Review Answers (2 of 2)

3. Connect the tools corresponding to the big data pipeline below



Unit 3.

Hadoop Architecture for Big Data

| Fundamentals of Big Data

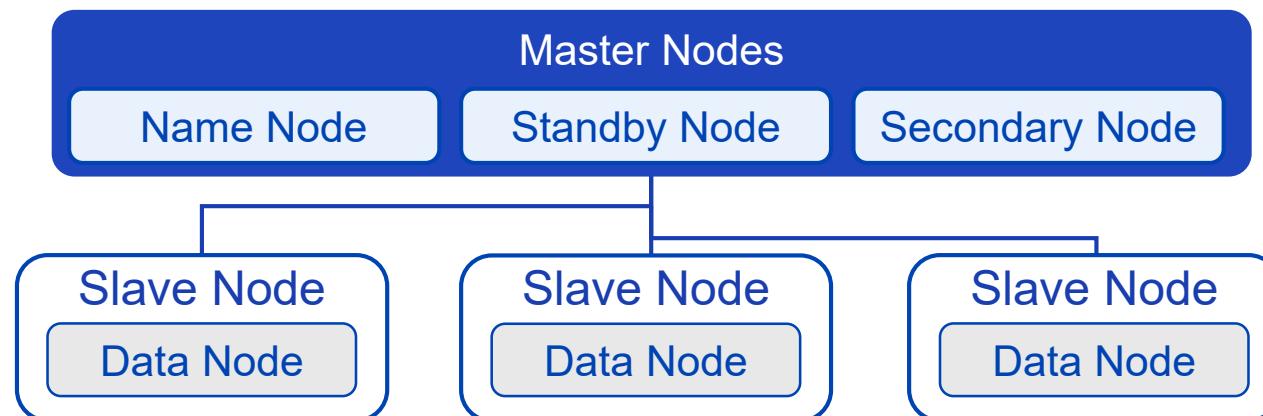
Unit 3

Hadoop Architecture for Big Data

- | 3.1. Hadoop Distributed File System Storage
- | 3.2. Yarn Resource Manager and Compute Architecture

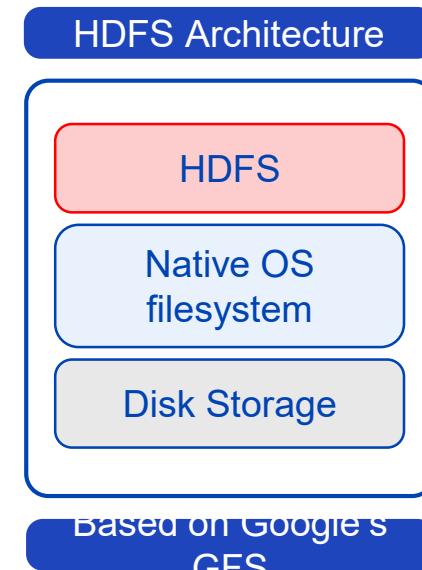
HDFS Architecture Overview

- HDFS follows the Master – Slave architecture
- Master Nodes
 - Responsible for managing the work and keeping metadata records
- Worker / Slave Nodes
 - Performs actual data read and write

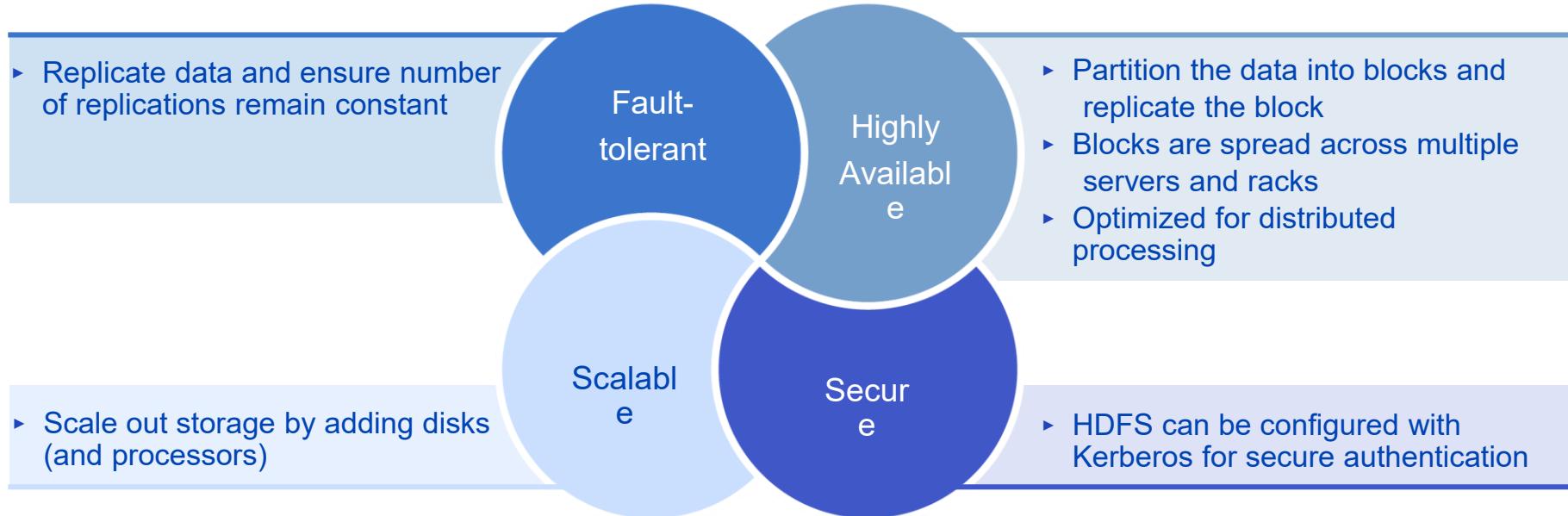


HDFS Basic Concepts

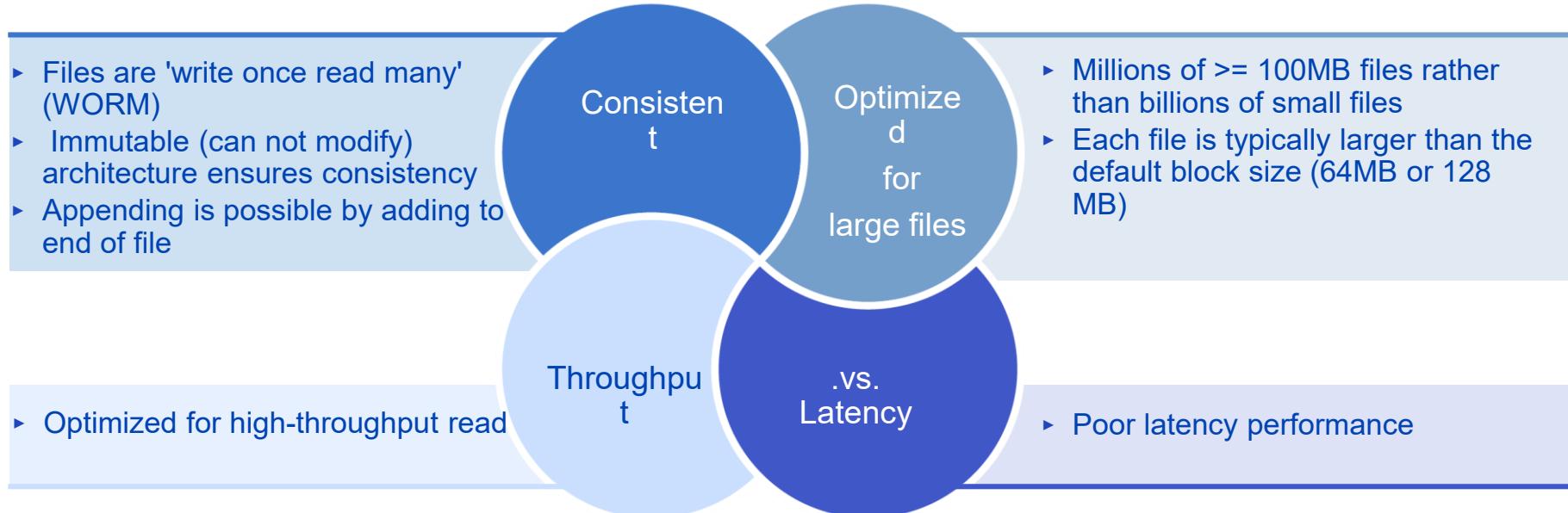
- Operates on top of system file system (typically Linux)
 - Linux ext3, ext4, or xfs
- Emulates a Linux file system with files and directories
 - Has ACL (access control list) similar to Linux
- Operates through Java applications (daemons)
- Based on Google File System (GFS)
- Uses scalable industry standard hardware
- Data is partitioned into blocks and distributed over multiple servers during write operation



HDFS Features (1/2)

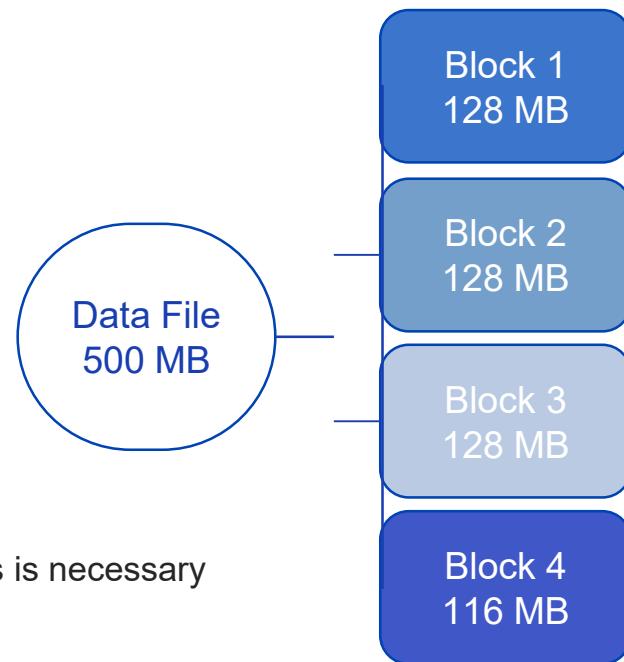


HDFS Features (2/2)



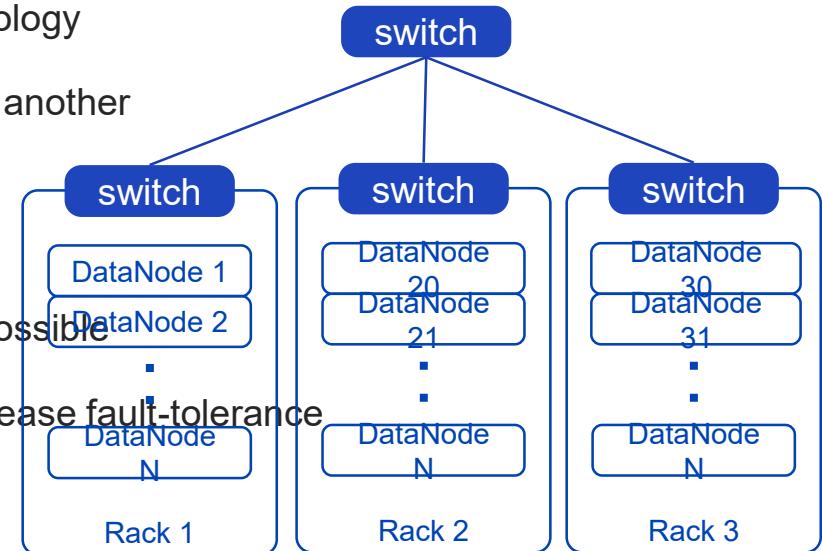
Partitioning Large Files

- Large files are partitioned into smaller blocks and replicated
 - Increased availability
 - Concurrent access for increased throughput
- Block size is configurable
 - Vanilla Hadoop → 64 MB
 - Cloudera Hadoop → 128 MB
- Blocks are not fix-sized slots
 - All blocks in a file are the same size, except the last block
 - As can be seen in Block 4, HDFS only uses as much disk space as is necessary



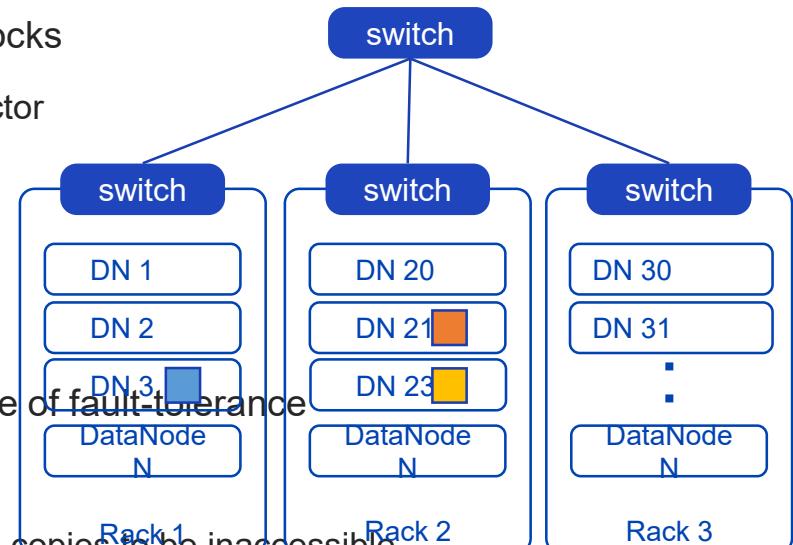
HDFS is Rack-Aware

- HDFS can be configured to be aware of the cluster rack topology
- If so configured, HDFS knows how “close” hosts are to one another
 - Closest: would be on the same host
 - Closer: On the same rack
- Clients read data blocks from the “closest” host whenever possible
- HDFS uses the rack topology during write operations to increase fault-tolerance



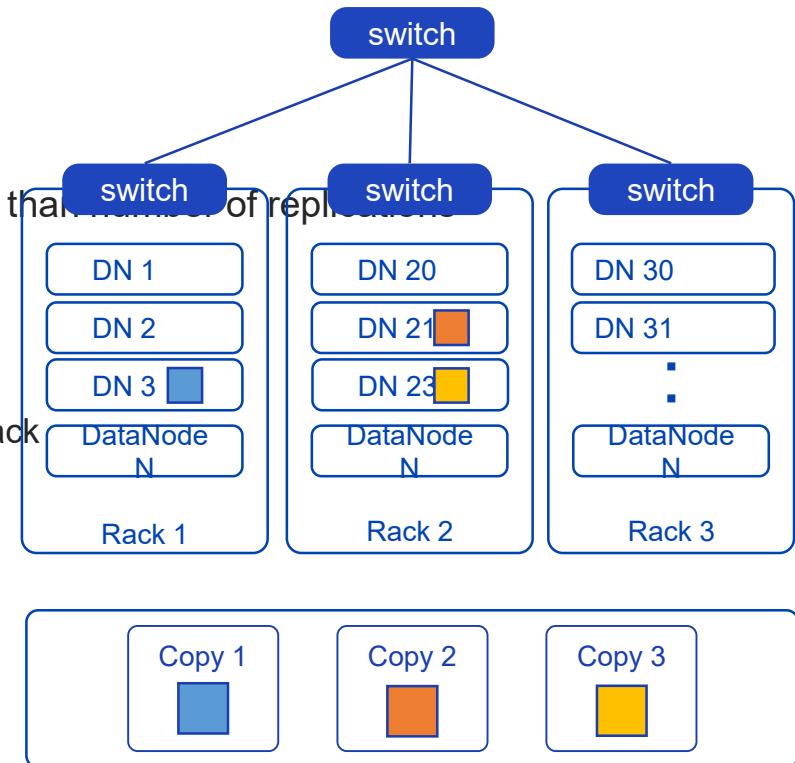
Replication and Rack-Awareness

- HDFS partitions large files into blocks and replicates these blocks
 - The number of replicated blocks is configurable – replication factor
 - Default replication factor is 3
 - Increases availability - 3 locations to initiate read
 - Increases reliability – Can suffer 66% loss and still recover
- The location of the replicated blocks, greatly affects the degree of fault-tolerance
 - All blocks on the same disk – effectively no increased reliability
 - All blocks on the same rack – a rack switch failure can cause all copies to be inaccessible



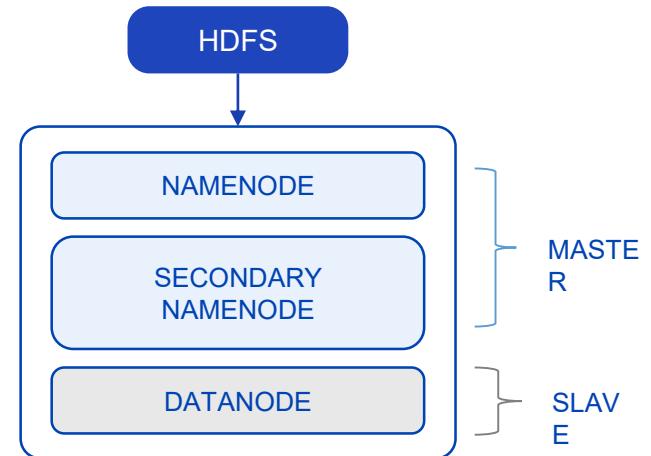
Rack-Awareness Policies

- Not more than one replica is placed on one node
- Not more than two replicas are placed on the same rack
- The number of racks used for block replication should be less than or equal to the number of replicas
- Ex: Replication factor = 3
 - Use less than 3 racks – so 1 or 2 racks
 - No more than 2 replicas on the same rack so can not be on 1 rack
 - Place one replica on 1 rack
 - Place remaining 2 replicas on second rack



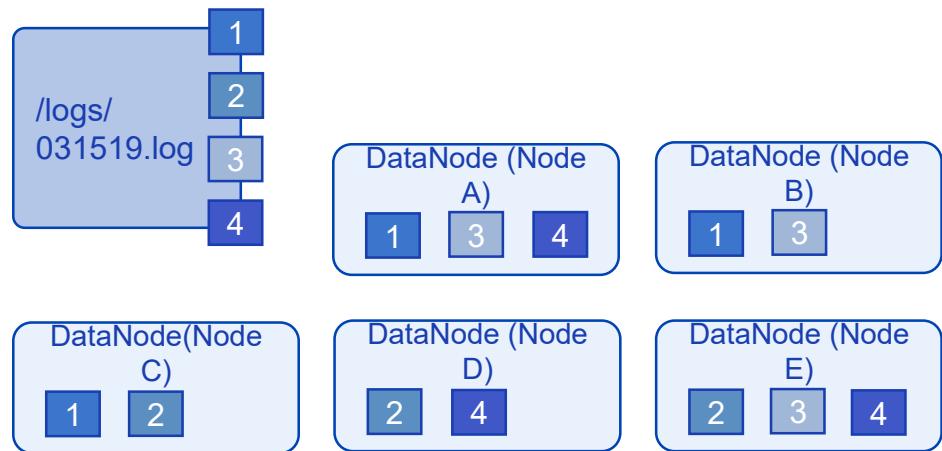
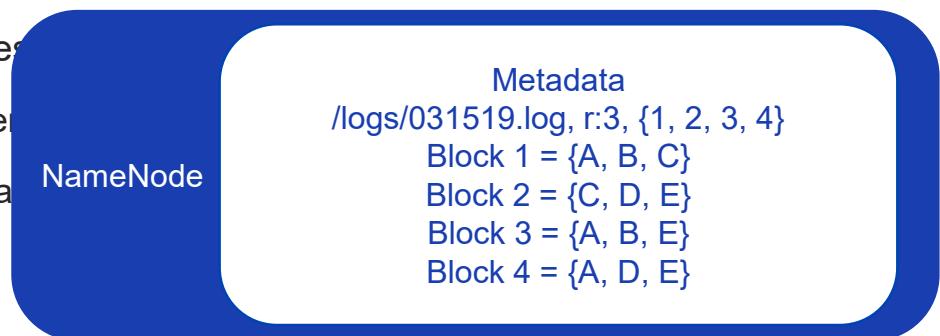
HDFS Components

- Master / Slave daemons
 - Daemons are long lived processes that are typically activated when the server is turned on
- Name Node
 - Active master daemon
 - Determines and maintains how the blocks of data are distributed across the DataNodes
 - Does not participate in the actual read / write operation
- Secondary / Standby Node
 - Passive master daemon
 - Responsible for keeping durable,
the meta data that the Name Node hosts in memory
- DataNode
 - Reads and writes the blocks of data



HDFS NameNode

- The NameNode is responsible for keeping the namespace metadata
 - Namespace metadata consists of association between files and blocks
 - The metadata is stored in memory for faster performance
- The metadata is also stored on disk for durability
 - Stored in file fsimage
 - fsimage does not store the block location
 - Changes to the metadata is logged in edits log file



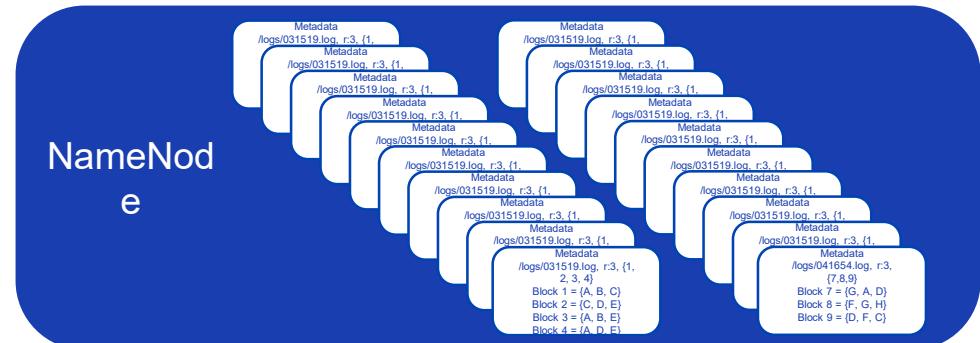
3.1 Hadoop Distributed File System

NameNode Memory Allocation

- When a NameNode is running, all metadata is held in RAM fast response

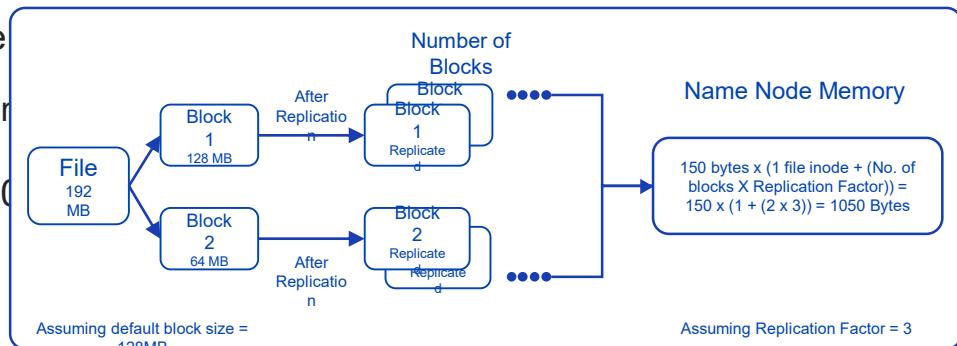
- NameNodes have 1 GB default Java Heap Size

- NameNode stores the following metadata
 - File Info - filename, ownership, permissions, etc.
 - Block Info - for each block that is part of the file, the block name and location
 - Each item uses between 150 to 250 bytes of memory

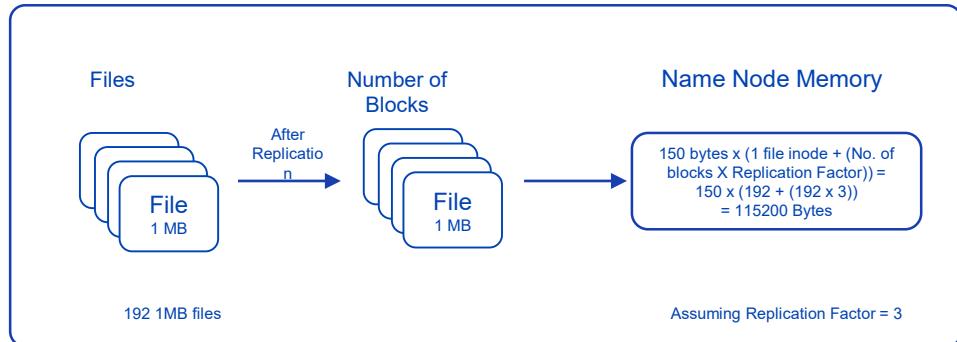


NameNode Small File Problem

- HDFS suffers from small file problem
- This is when a large portion of the files in HDFS are small.
 - HDFS NameNode may run out of Java memory before the data is replicated.
- Example: 1 GB of data stored as a single file .vs. 1000 small files of 1 MB each.
- Single 1 GB file with block size of 128 MB
 - 1 File Info and 8 Block Info
 - Total of 8 items in memory
- 1000 x 1 MB files with block size of 128 MB
 - 1000 File Info and 1000 Block Info
 - Total of 2000 items in memory



Scenario 2 (192 small files, 1MiB each):



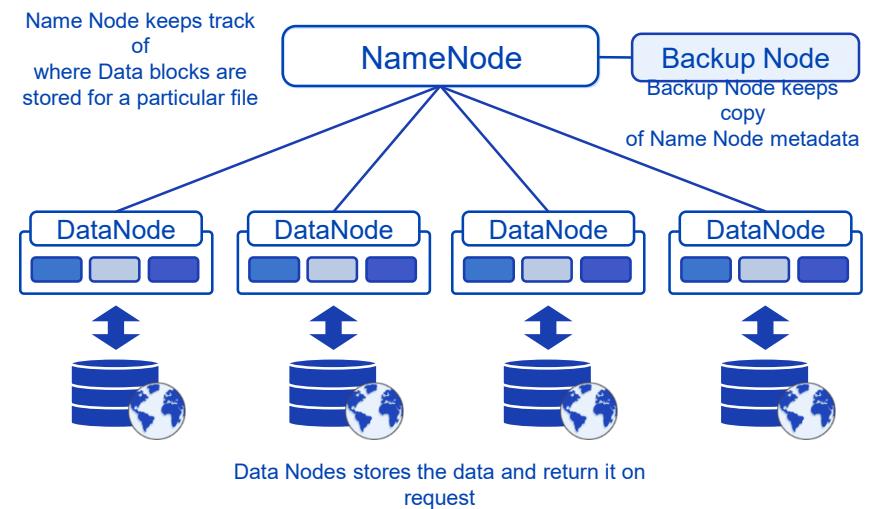
HDFS Secondary NameNode

- The Secondary NameNode, despite its name, is not a failover daemon for the NameNode
- Its primary role is to checkpoint the namespace metadata
 - Get a copy of the latest fsimage and edits log file from NameNode
 - Merge the changes in edits log file into fsimage
 - Send the new copy of fsimage to NameNode
 - The NameNode updates its fsimage and resets edits log file



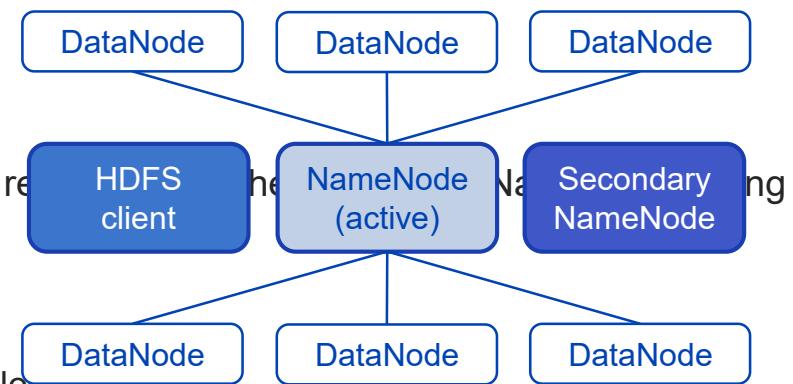
HDFS DataNodes

- DataNodes are the actual workers responsible for writing and reading data blocks
- When writing blocks, DataNodes communicate with each other to replicate the block by the replication factor
 - Blocks are copied in a pipeline fashion
 - The NameNode is not part of the write process.
 - Its only role is to map blocks to datanodes
- The blocks are save as files in the underlying filesystem
 - Location to save blocks cans be configured
 - Blocks are named blk_xxxxxxxxxx
- DataNodes do not know to which file a block belongs to



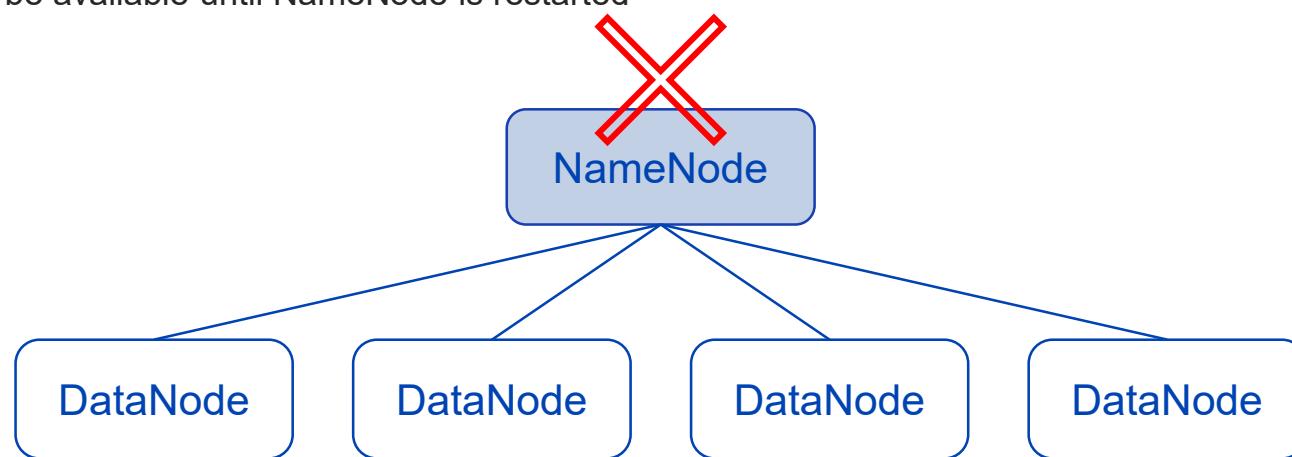
HDFS in non-High Availability Mode

- HDFS can be deployed in High Availability (HA) mode or not
- In non-HA mode, the following daemons service HDFS
 - NameNode (Master)
 - Secondary NameNode (Master)
 - DataNode (Worker)
- In non-HA mode, if the NameNode fails, the system has to be restarted to take the place of NameNode
 - On restart, the new NameNode reads the fsimage file
 - All the datanodes send block location information to the NameNode

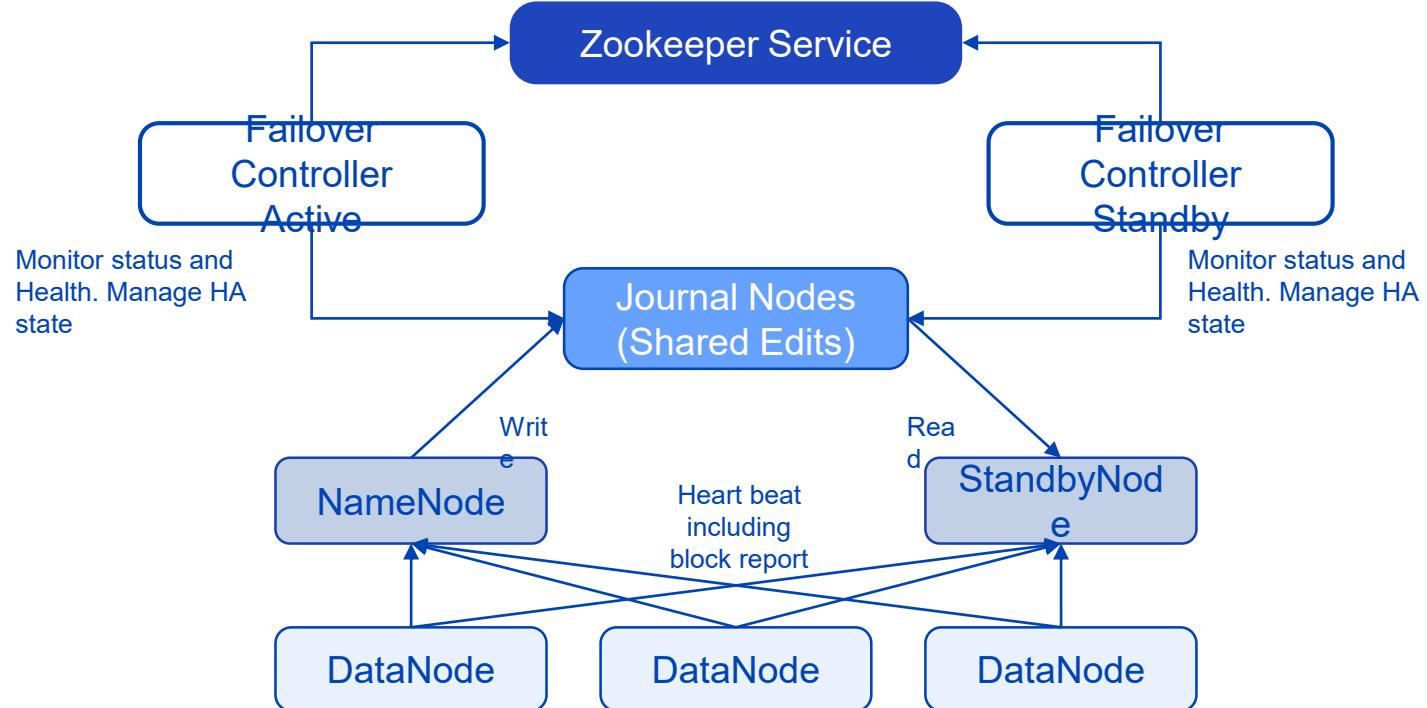


Single Point of Failure (SPOF)

- A Hadoop cluster has a single active NameNode
 - If at any point, there were actually multiple active namenodes, the cluster would suffer from split-brain
- If the active NameNode fails, the entire HDFS service will become inaccessible
 - This is a single point of failure
 - HDFS will not be available until NameNode is restarted



HDFS in High Availability Mode



HDFS in High Availability Mode

Discussion

Explain the difference between HA Mode and Non-HA Mode using keywords

Automatic failover

SPOF

Name Node

Standby Node

Secondary Node

Passive master daemon

DataNode Failure and Recovery

- DataNodes send heartbeats to the NameNode periodically
 - Frequency can be configured with default of 3 seconds
- If heartbeat is not received from a DataNode, it will progress through several stages and finally declared dead
 - Initially declared stale after 30 seconds
 - Declared dead after 10.5 minutes
- The NameNode excludes stale DataNodes from participating in new writes
- After declared dead, all blocks on the dead DataNode is considered lost
 - This will trigger the block under-replicated routine and blocks on the dead datanode will be re-replicated on other datanodes
 - If a DataNode rejoins the cluster after a period of being dead, the NameNode ensures that that blocks are not over-replicated by removing them

HDFS File Permissions

- HDFS Files and directories have permission very similar to Linux
 - Linux has read (r), write (w), and execute (x) permission for owner, group and others
 - Each permission is expressed as a bit – 1 to allow and 0 to deny
 - HDFS also sets permissions for owner, group and others
- However, in HDFS, there is no such thing as execute (x) permission for a file or directory
 - For directories, execute (x) permission allows access to subdirectories
 - For files, this permission is ignored
- HDFS can also be set with ACL enabled instead
 - ACL allows enforcement with much finer granularity than owner, group and others

```
$ hdfs dfs -ls .
drwxr-xr-x - owner group 0 2016-04-02 22:10 /user/owner/test
-rw-rw-r--   owner group 110 2016-04-22 22:15 /user/owner/test/t.txt
```

Accessing HDFS from Command Line

- Although the data blocks for a file is stored in the Linux file system, we can not access it directly
- From the command line use hdfs dfs -<subcommand> <options> <parameters>
- List of commonly used subcommands
 - ls – list the contents of a hdfs director
 - cp – copy a hdfs file
 - mv – move a hdfs file
 - mkdir – make an hdfs directory
 - put – copy a local Linux file to hdfs
 - get – copy a hdfs file to a Linux file

```
$ hdfs dfs –put input.txt input.txt
$ hdfs dfs –ls /
$ hdfs dfs –rm /reports/sales.txt
```

Copy Data Between HDFS and Linux

- `hdfs dfs <-put> / <-get>`



HDFS Filesystem Shell Commands

Command	Description
ls	lists the contents of folders
du	shows the disk usage
count	counts the number of directories, files, and bytes in a path
chgrp, chown, chmod	change file and directory permissions
stat	print statistics about a file or directory
cat, text	display the contents of files
tail	show the last 1KB of a file's contents
get, copyToLocal	identical commands that copy a file from HDFS to the local file system
put, copyFromLocal	identical commands that copy a file from the local file system into HDFS
getmerge	get a collection of files and merges them into a single file

HDFS Filesystem Shell Commands

Command	Description
mv	move a file in HDFS
cp	copy a file to another location in HDFS
mkdir	make a new directory in HDFS
rm	remove a file (to the Trash folder)
rm -R	remove folders and recursively remove any files and subfolders(to the Trash folder)
test	checks if a file exists
touch	writes a timestamp into a new file
expunge	empties the user's Trash folder

Accessing HDFS from HUE

- Hadoop User Experience (HUE) is a GUI based tool for accessing HDFS
 - Create, move, rename, modify, upload, download, and delete directories and files
 - View text file contents

The screenshot shows the HUE File Browser interface. The URL in the address bar is `/user/hue`. The main area displays a list of files and directories under the `pig` folder. A context menu is open over the `pig` directory, listing options: `Rename`, `Move`, `Copy`, `Change permissions`, `Move to trash`, and `Delete forever`. The `Rename` option is highlighted with a red border. The table below lists the files and their details:

Name	Size	User	Group	Permissions	Date
..		hdfs	supergroup	drwxr-xr-x	September 25, 2014 12:44 PM
.		hue	hue	drwxr-xr-x	September 11, 2014 04:57 PM
.Trash		hue	hue	drwxr-xr-x	September 11, 2014 04:57 PM
oozie		hue	hue	drwxrwxrwt	March 04, 2014 11:46 PM
pig		hue	hue	drwxrwxrwt	March 13, 2014 10:15 PM

NameNode Web User Interface

- Get HDFS service health status and reports
- DataNode capacity and health status
- File metadata information – block ID, location, etc

The screenshot shows a web browser window with the URL `localhost:50070/dfshealth.html`. The title bar says "All Applications" and "NameNode Information". The main content area is titled "Overview 'localhost:9000' (active)". It displays various system statistics:

Started:	Sun Apr 06 15:52:11 IST 2014
Version:	2.3.0, r1567123
Compiled:	2014-02-11T13:40Z by jenkins from branch-2.3.0
Cluster ID:	CID-5edbd0ds-c69f425b-bc7-a662ac5d45dc
Block Pool ID:	BP-1127673761-127.0.1.1-1336692397591

Below this is a "Summary" section with the following details:

- Security is off.
- Safemode is off.
- 15 files and directories, 17 blocks = 52 total filesystem object(s).
- Heap Memory used 34.01 MB of 88.5 MB Max Heap Memory is 889 MB.
- Non Heap Memory used 40.67 MB of 40.69 MB Committed Non Heap Memory Max Non Heap Memory is -1 B.

At the bottom, there is a "Configured Capacity" field with a value of 93.54 GB.

Unit 3

Hadoop Architecture for Big Data

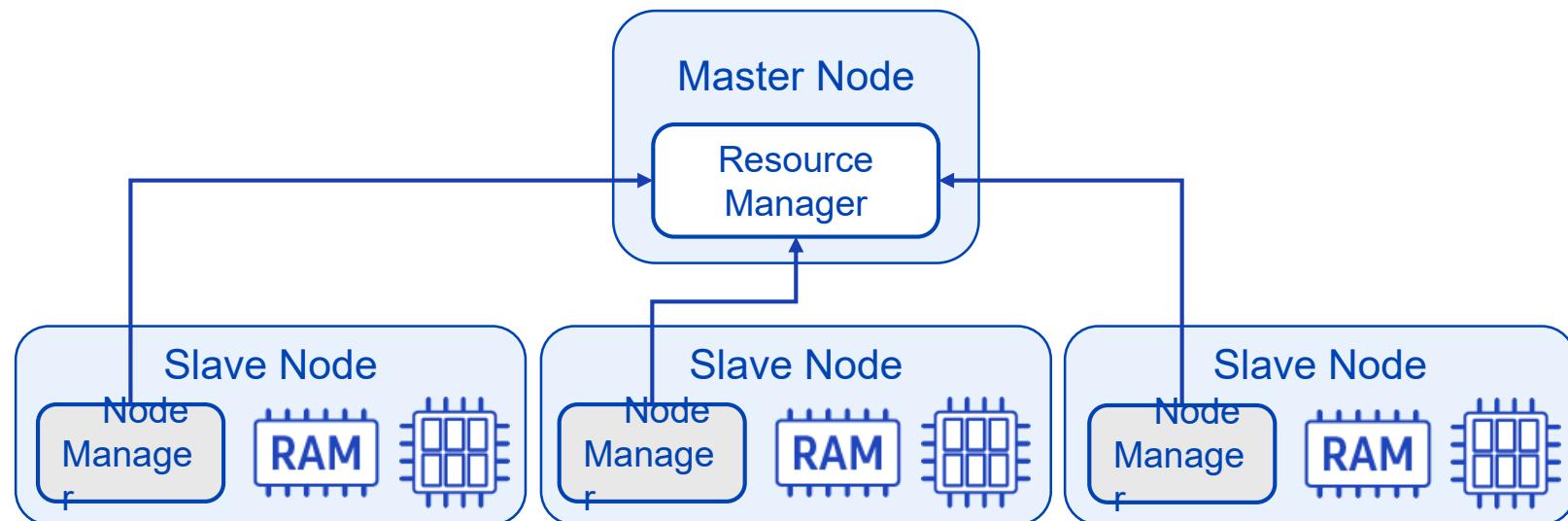
- | 3.1. Hadoop Distributed File System Storage
- | 3.2. Yarn Resource Manager and Compute Architecture

What is YARN?

- A platform for managing resources in a Hadoop cluster
- Yet Another Resource Negotiator
- Master / Slave architecture
- A framework for distributed processing applications on cluster to execute using cluster resources
 - MapReduce v2
 - Spark
 - Impala
 - Search
 - SQL queries
 - Machine Learning
 - Streaming processing

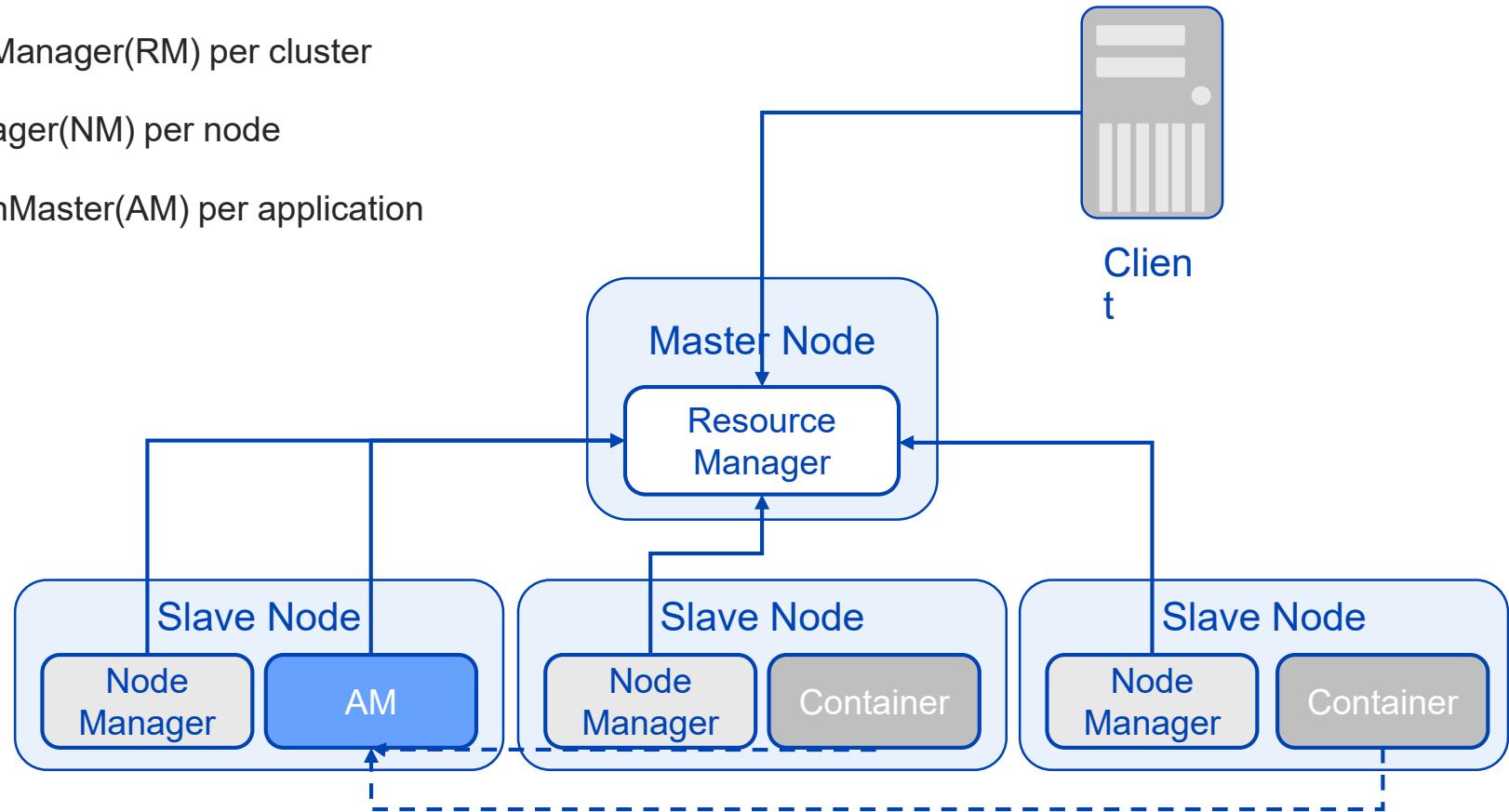
Why YARN?

- YARN allows you to run diverse workloads on the same Hadoop cluster
- Allows dynamically sharing cluster memory and CPU resources between applications
- Increase cluster utilization



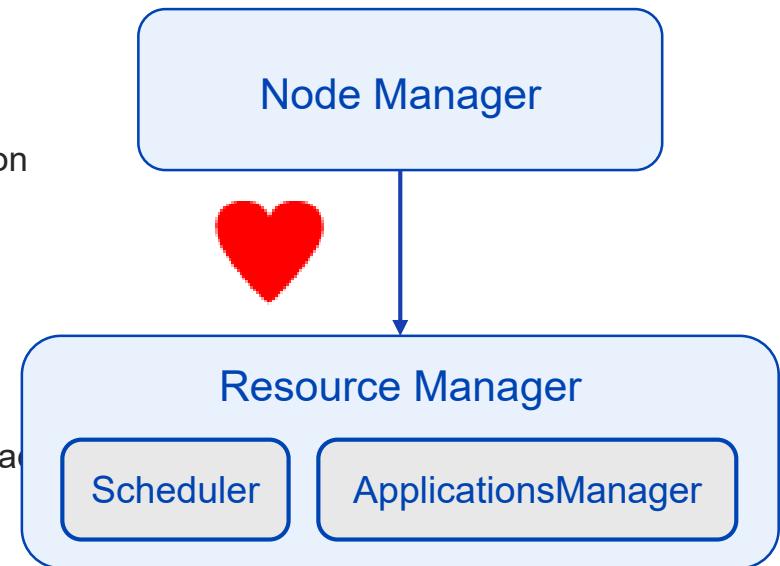
YARN Daemons

- ResourceManager(RM) per cluster
- NodeManager(NM) per node
- ApplicationMaster(AM) per application



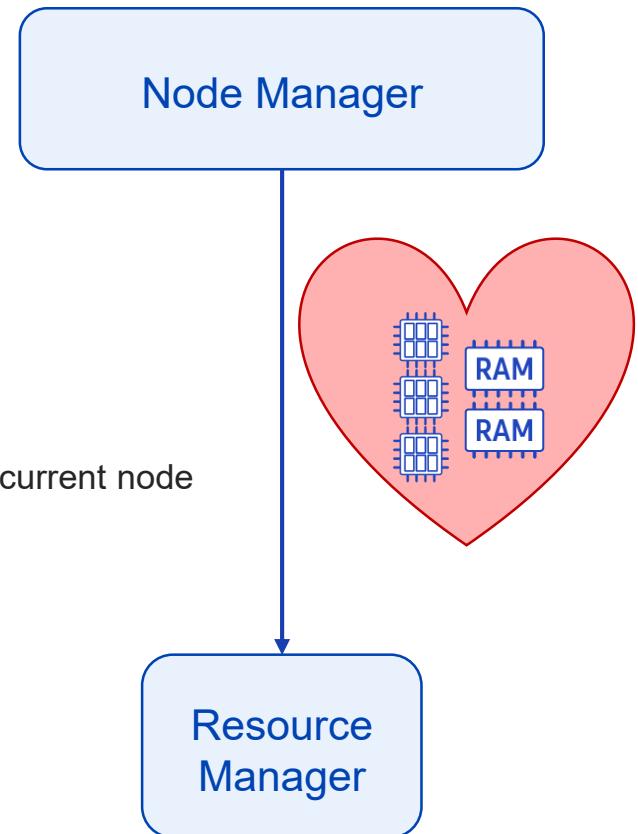
YARN ResourceManager

- Master daemon
- Two main components: Scheduler and ApplicationsManager
- Scheduler:
 - Responsible for allocating resources to various running application
 - Pluggable policy – CapacityScheduler, FairScheduler
- ApplicationsManager
 - Responsible for accepting job submissions
 - Negotiates container for per-application ApplicationMaster and tracks its progress
- Tracks heartbeat from NodeManagers



YARN Node Manager

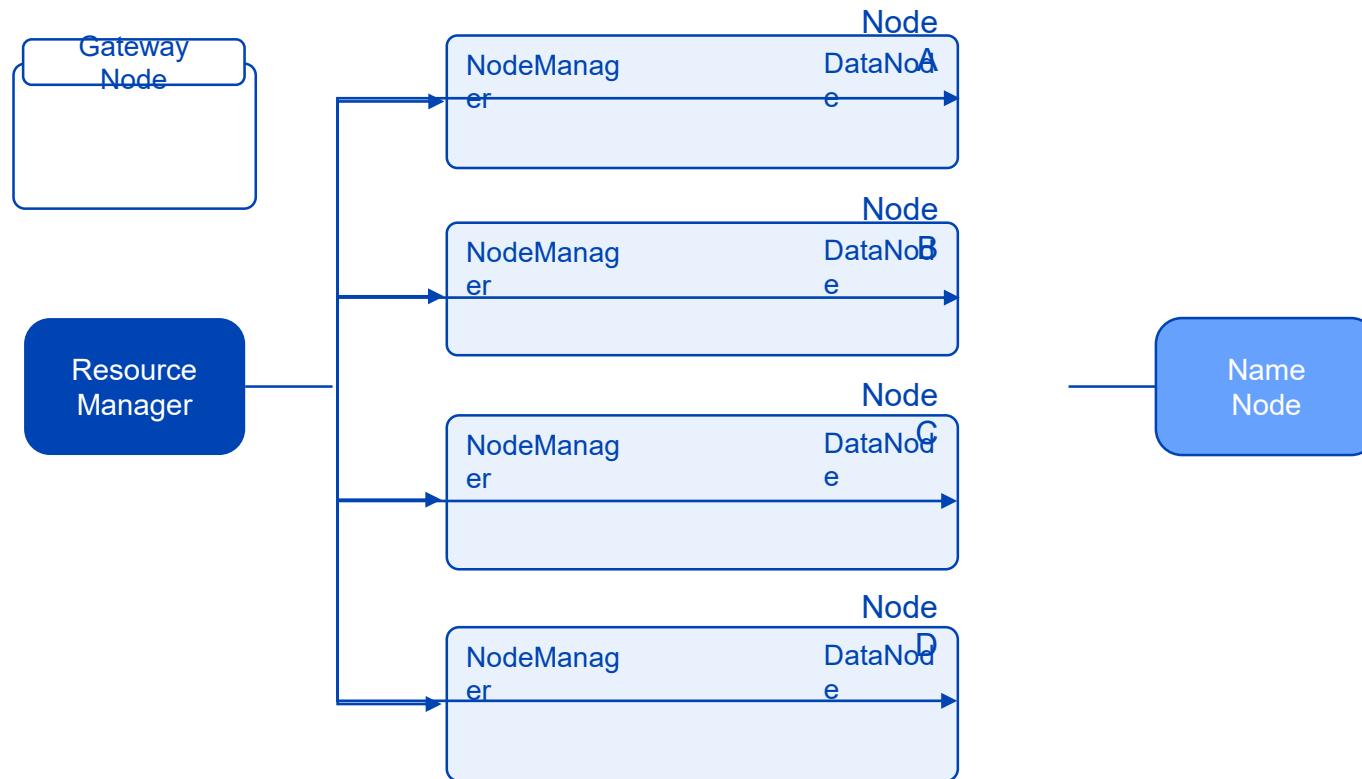
- Worker daemon
- Runs on a worker node and usually along with DataNode
- Responsible for launching and managing containers for application
 - Containers execute tasks as specified by the per application AppMaster
- Monitors the health and resources of the node it is running on
 - Sends heartbeat to ResourceManager with health and resource status of current node



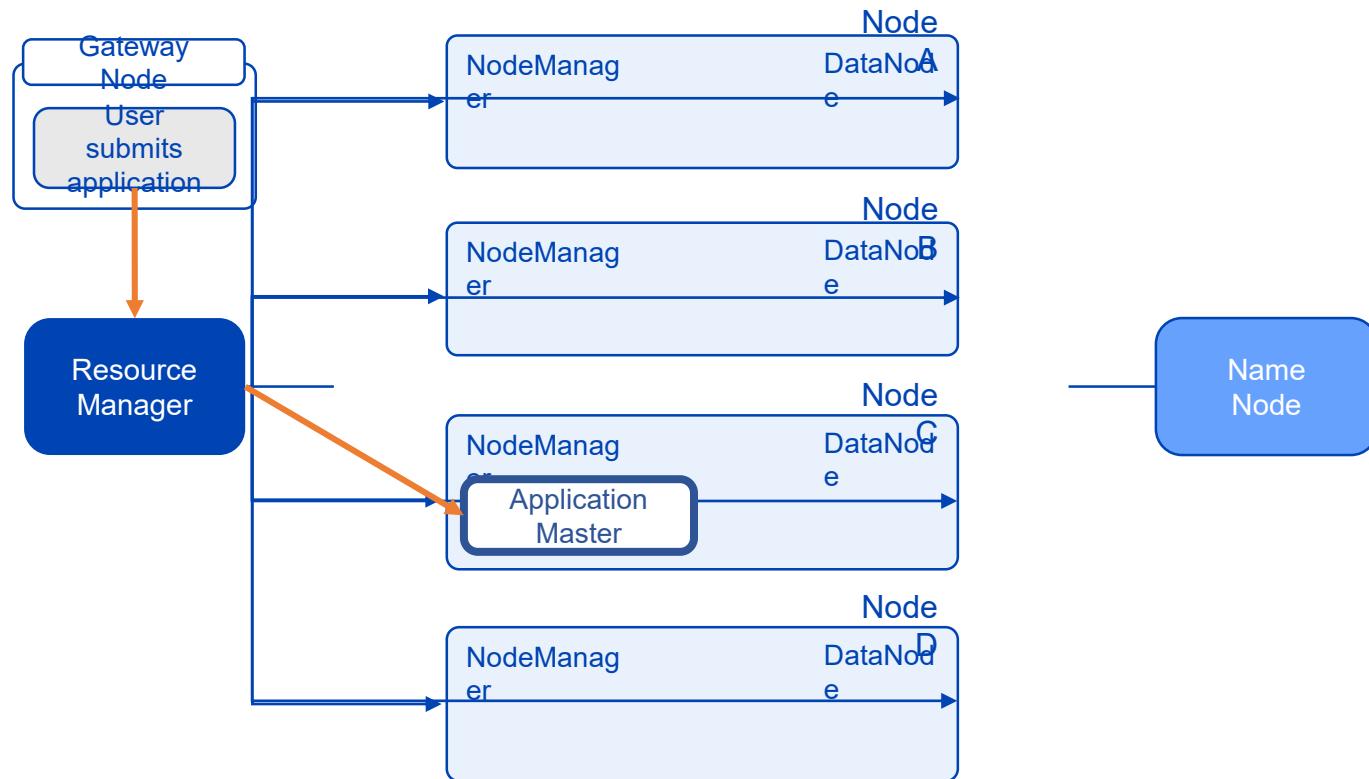
YARN Fault-Tolerance

- There are many daemons involved in running an application
- How does YARN handle exceptions?
- Tasks on NodeManager exits with exceptions or stops sending heartbeats
 - ApplicationMaster reattempts the task on a new container on another node
 - Re-attempts up to 4 times
- ApplicationMaster stops sending heartbeats to YARN
 - ResourceManger starts a new ApplicationMaster and reattempts the whole application
 - Re-attempts up to 2 times
- ResourceManager dies
 - It is possible to run ResourceManager in HA mode
 - In HA mode, a standy by ResourceManger will failover automatically

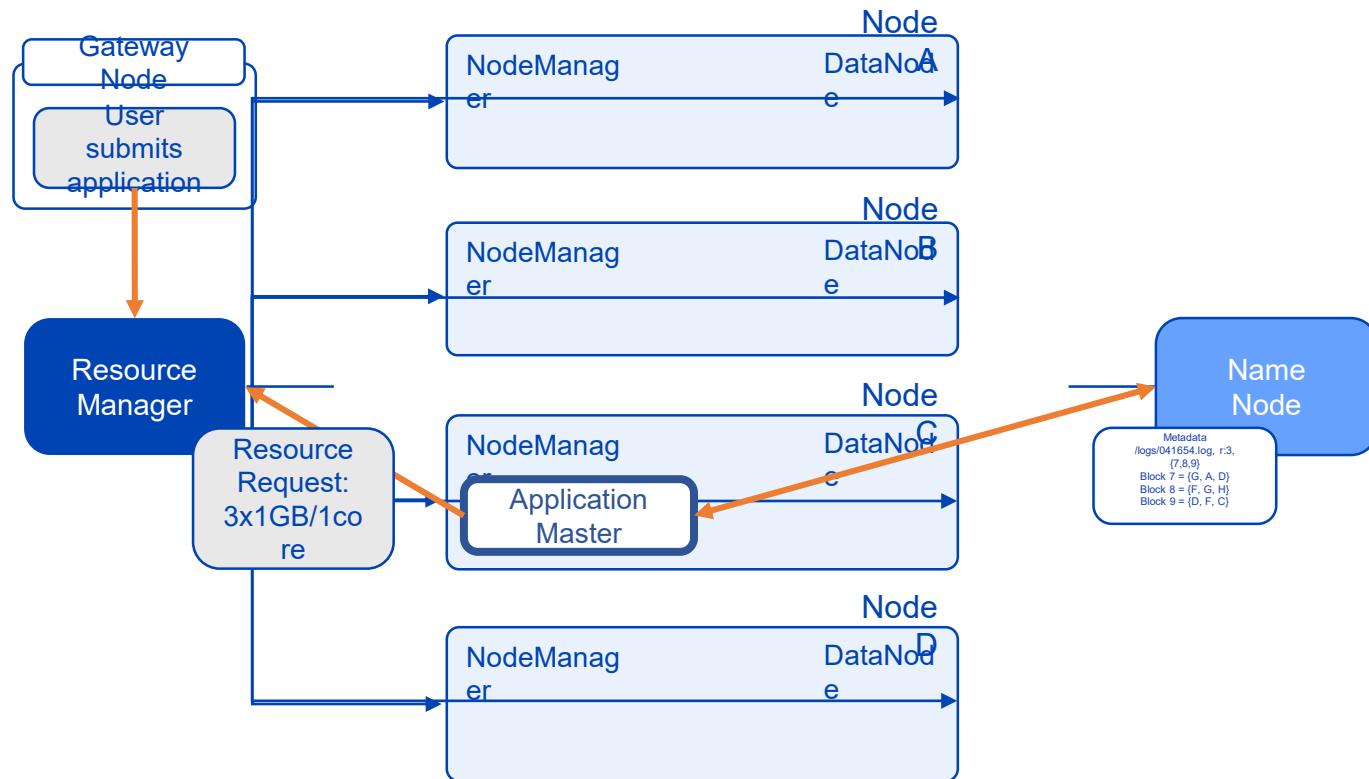
Running an Application in YARN



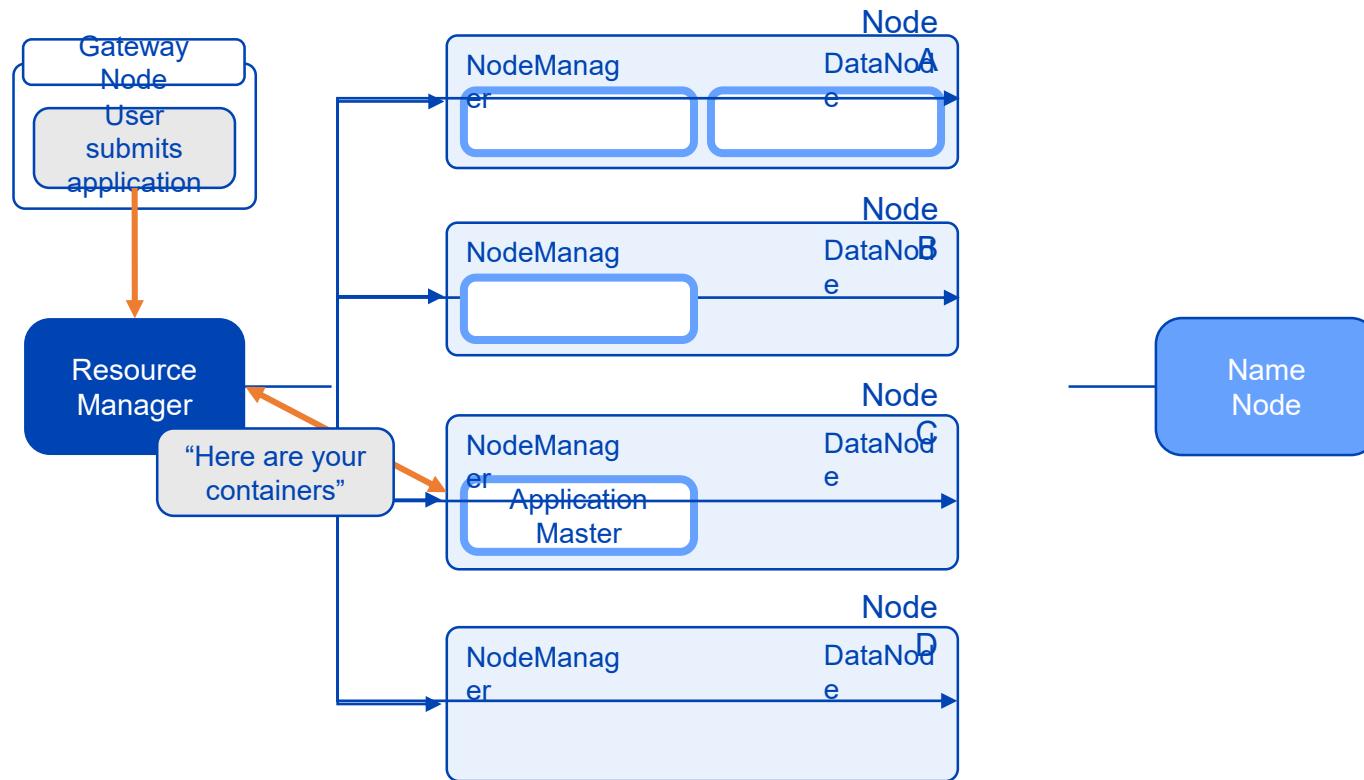
Running an Application in YARN



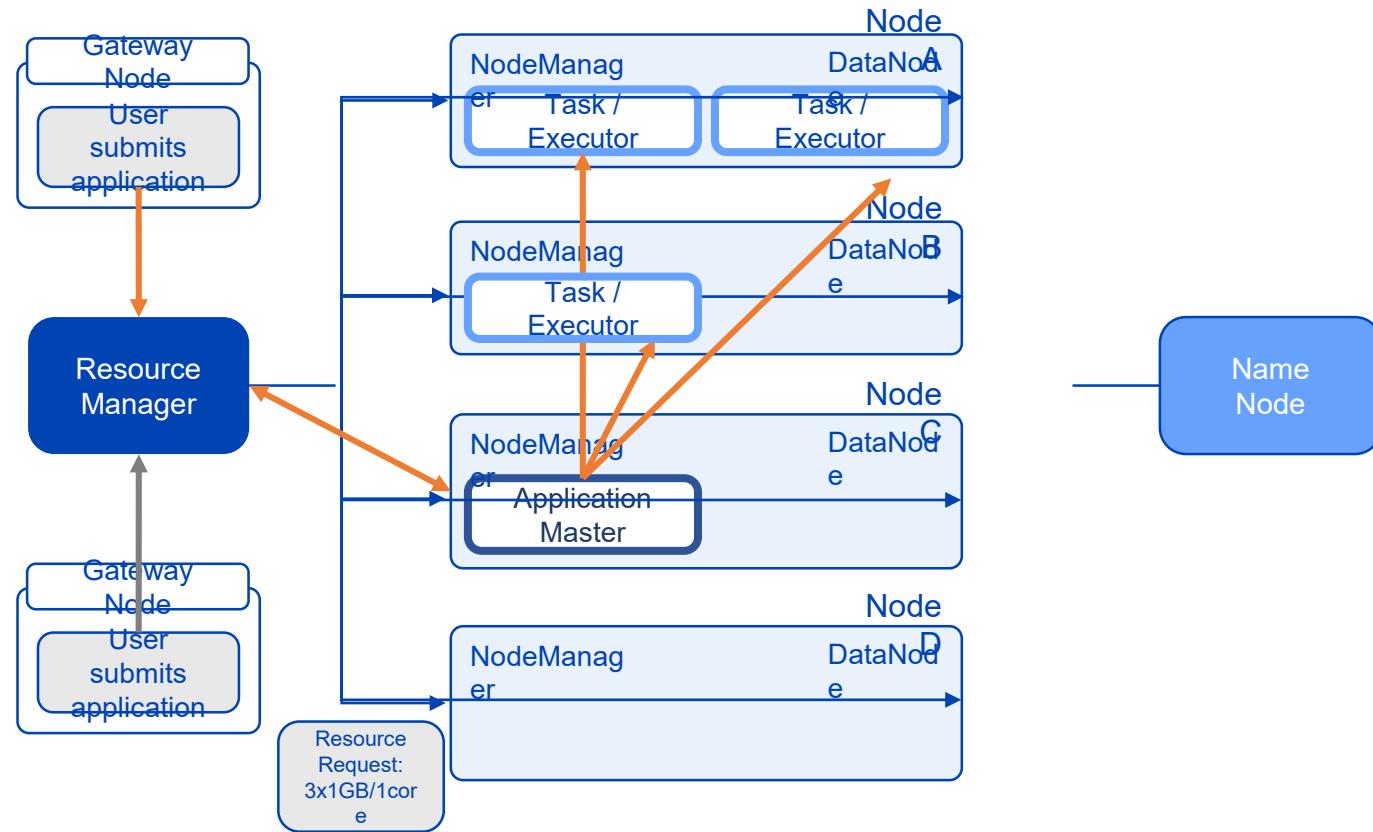
Running an Application in YARN



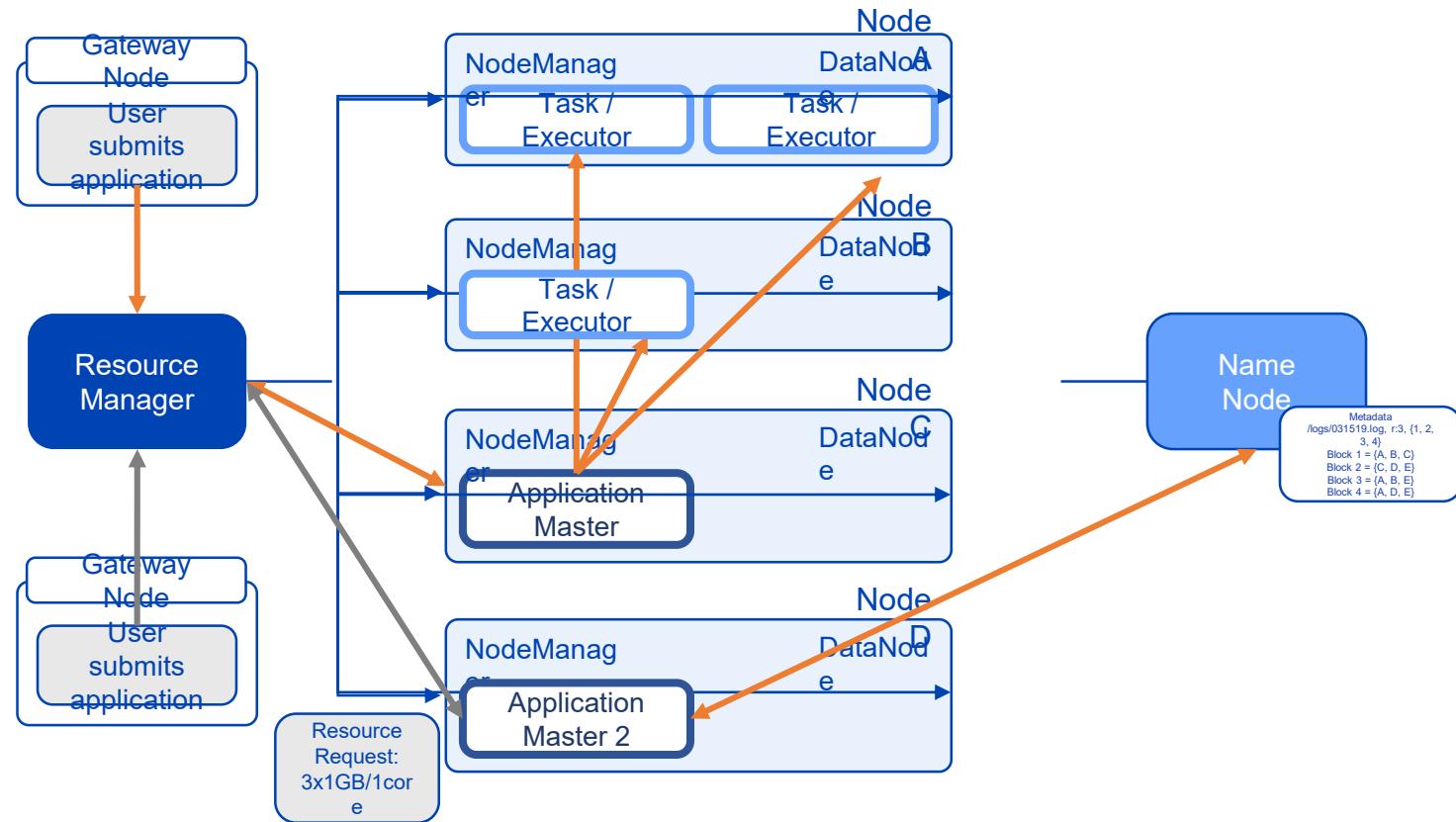
Running an Application in YARN



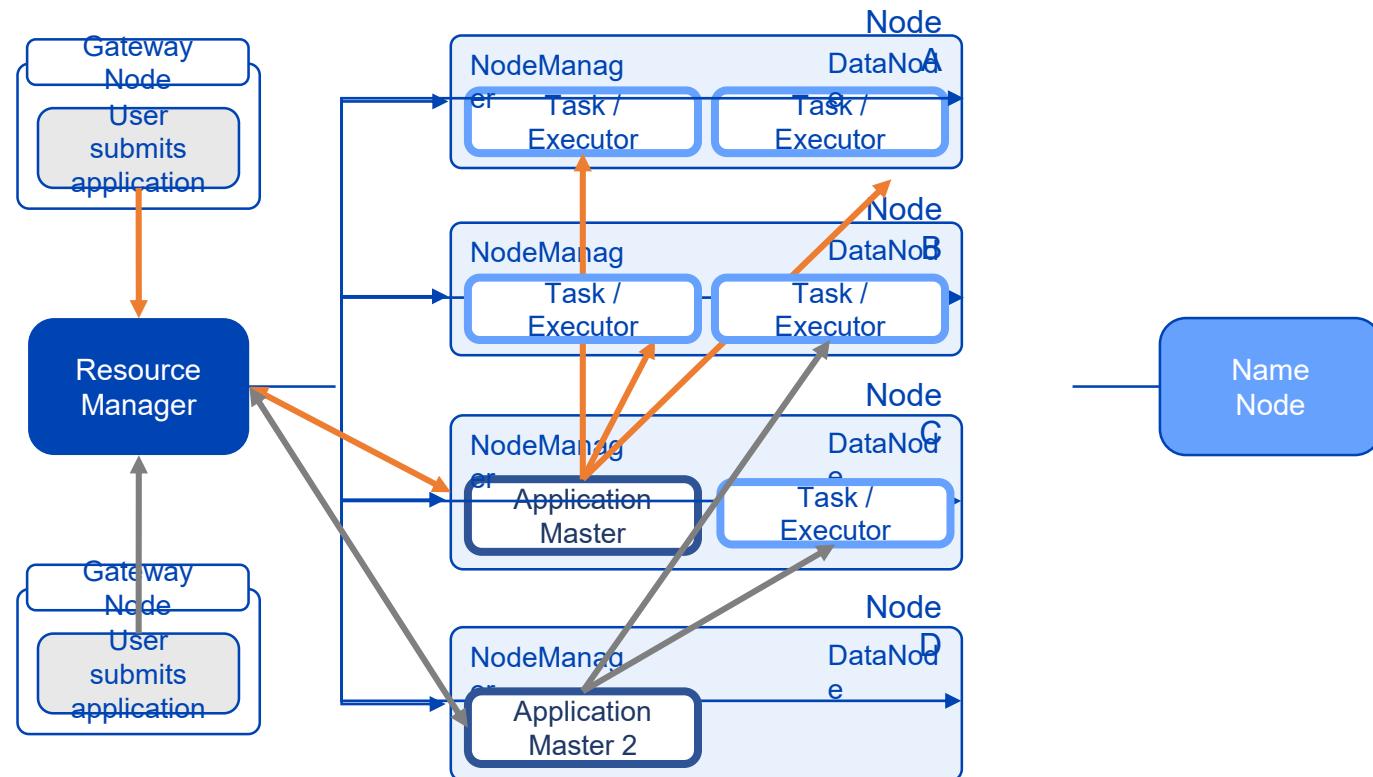
Running an Application in YARN



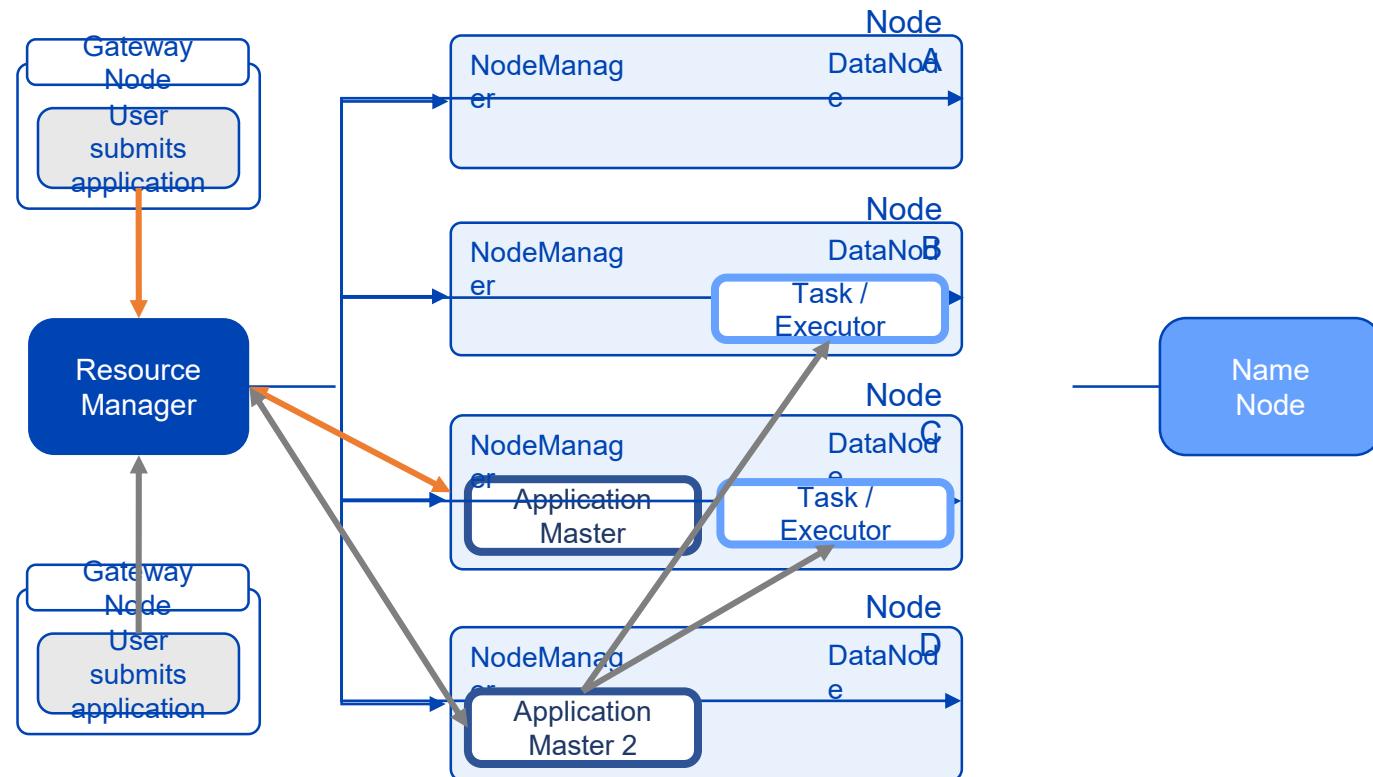
Running an Application in YARN



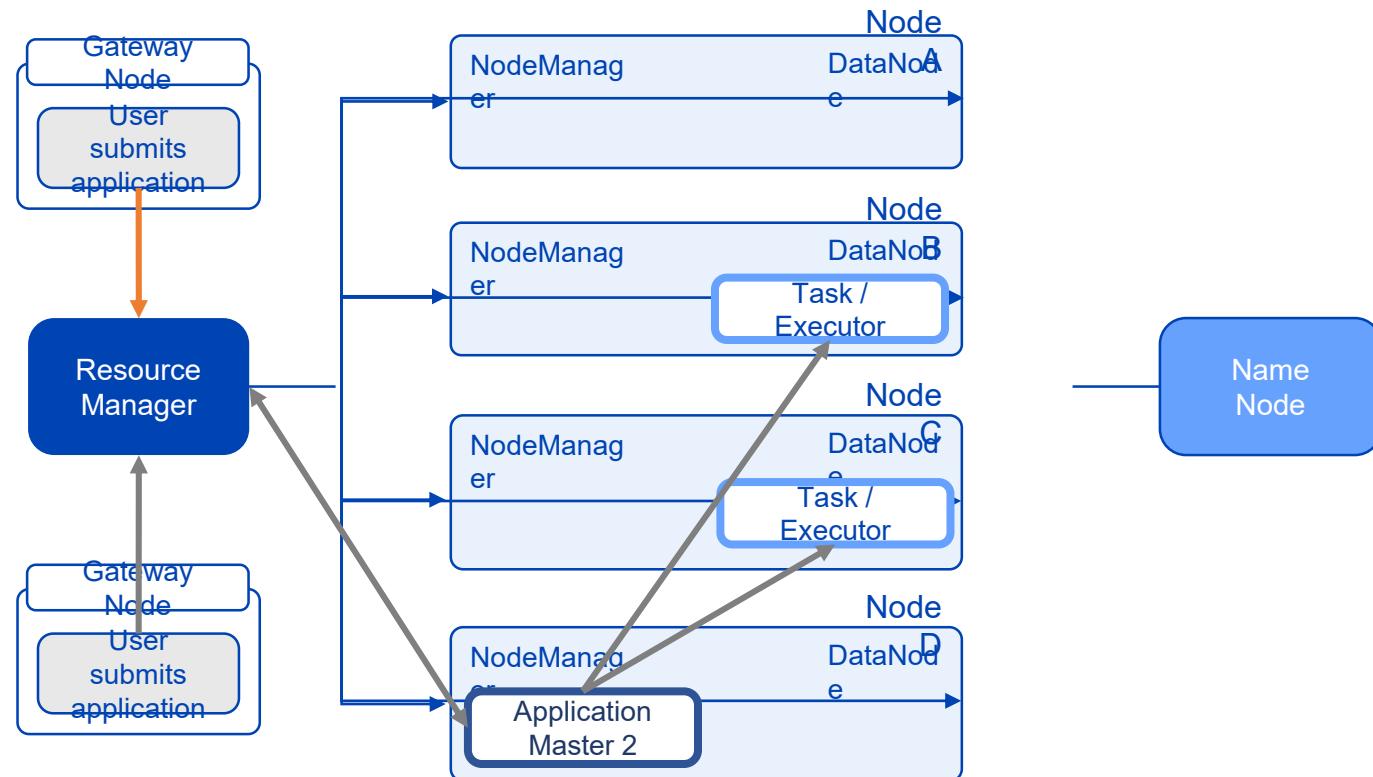
Running an Application in YARN



Running an Application in YARN



Running an Application in YARN



YARN Command Line

- Basic syntax: `yarn <subcommand> <args>`
- `yarn application –list`
 - To view all applications managed by YARN currently running on the cluster
- `yarn application –status <app-id>`
 - To display the status of an individual application
- `yarn application –kill <app-id>`
 - To kill a running application identified by app-id

YARN Application Web UI

- Provides browser-based GUI to view the status of applications running of YARN
- View log files

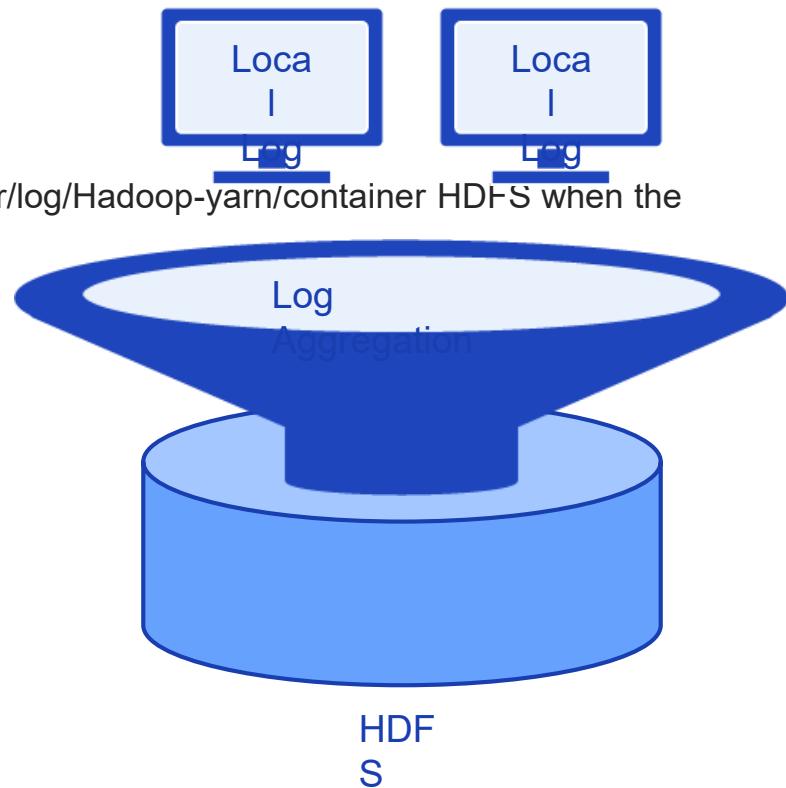
The screenshot shows a web browser window with the URL `localhost:8088/cluster/apps`. The title bar says "All Applications". The page features a yellow elephant logo and the word "hadoop". On the left, there's a sidebar with a tree menu under "Cluster" and sections for "About Nodes", "Applications" (with a list of states: NEW, NEW SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED), "Scheduler", and "Tools". The main area has a table titled "Cluster Metrics" with various statistics. Below that is a search bar and a table listing two application entries. The first entry is for "application_1412793406652_0002" with status ACCEPTED. The second entry is for "application_1412793406652_0001" with status FINISHED. At the bottom, it says "Showing 1 to 2 of 2 entries".

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
2	0	1	1	1	250 MB	2.20 GB	0 B	1	0	0	0	0

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1412793406652_0002	root	QuasiMonteCarlo	MAPREDUCE	default	Wed, 08 Oct 2014 18:51:01 GMT	N/A	ACCEPTED	UNDEFINED		UNASSIGNED
application_1412793406652_0001	root	QuasiMonteCarlo	MAPREDUCE	default	Wed, 08 Oct 2014 18:45:11 GMT	Wed, 08 Oct 2014 18:47:56 GMT	FINISHED	SUCCEEDED		History

YARN Application Log Aggregation

- YARN provides application log aggregation services
 - Recommended (enabled by default in Cloudera distribution)
- When YARN log aggregation is available:
 - Container log files are moved from NodeManager hosts' /var/log/Hadoop-yarn/container HDFS when the application completes
 - Default HDFS directory /tmp/logs
- Aggregated YARN application log retention
 - CM default : 7 days



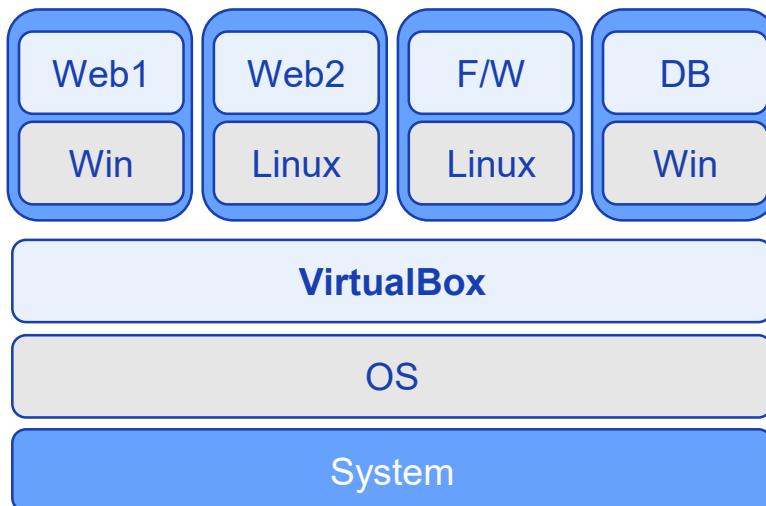
[Lab1]

Starting VirtualBox



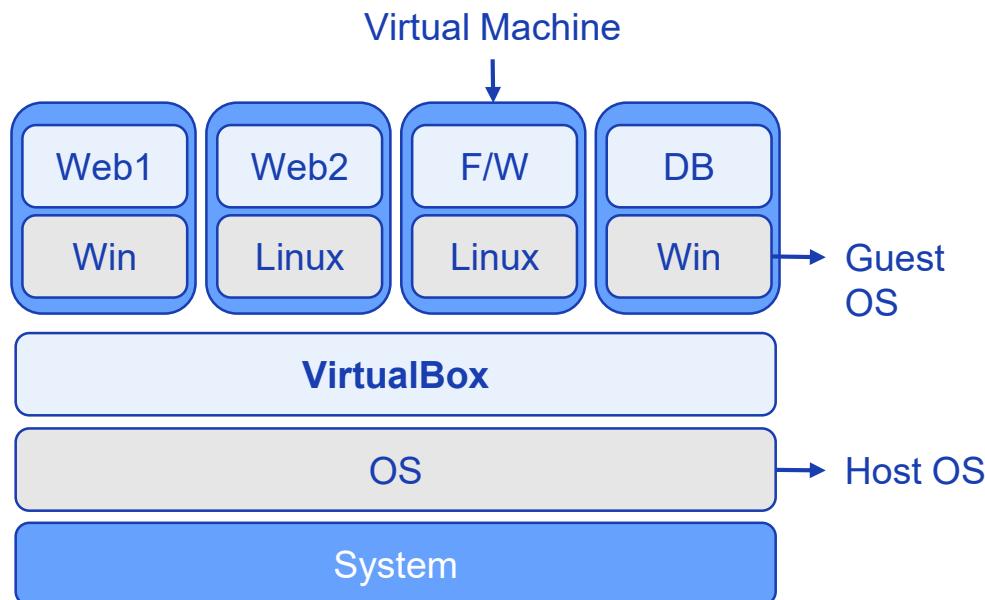
What is VirtualBox?

- A cross-platform virtualization application
 - runs on Windows, Linux, Mac OS X
 - allows multiple machines to run on one system



Basic Terminology

- Host OS
- Guest OS
- Virtual machine (VM)

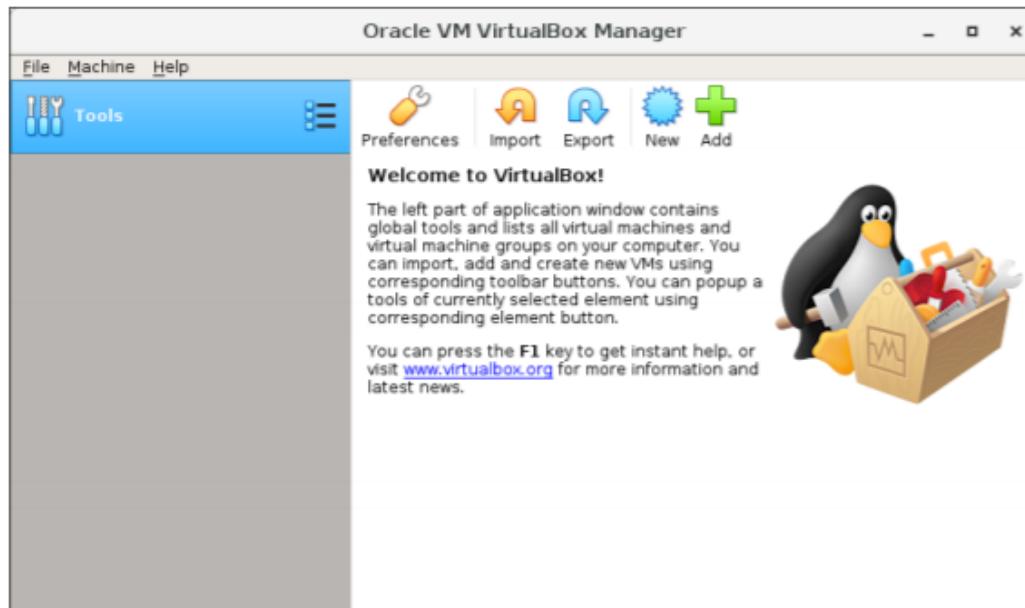


Starting VirtualBox

- Double-click the VirtualBox icon on the Windows desktop



- VirtualBox Manager is displayed:



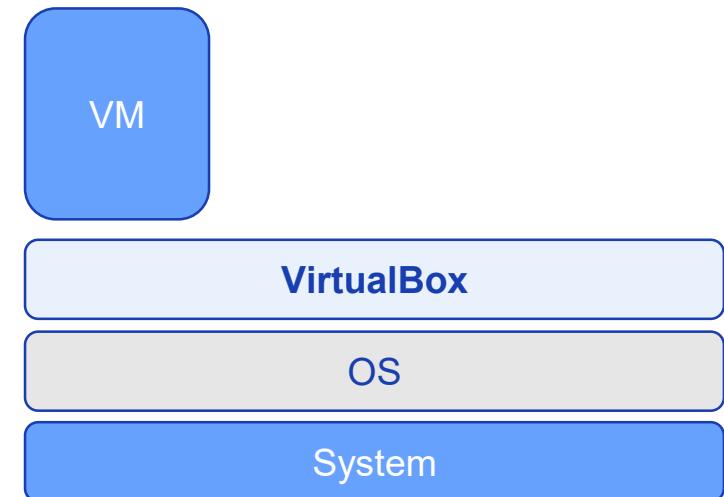
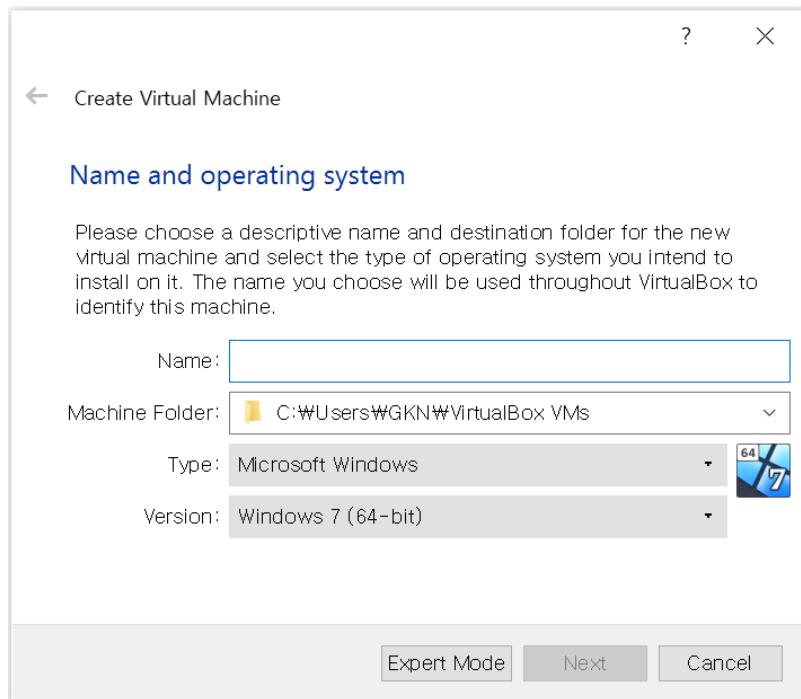
VirtualBox

OS

System

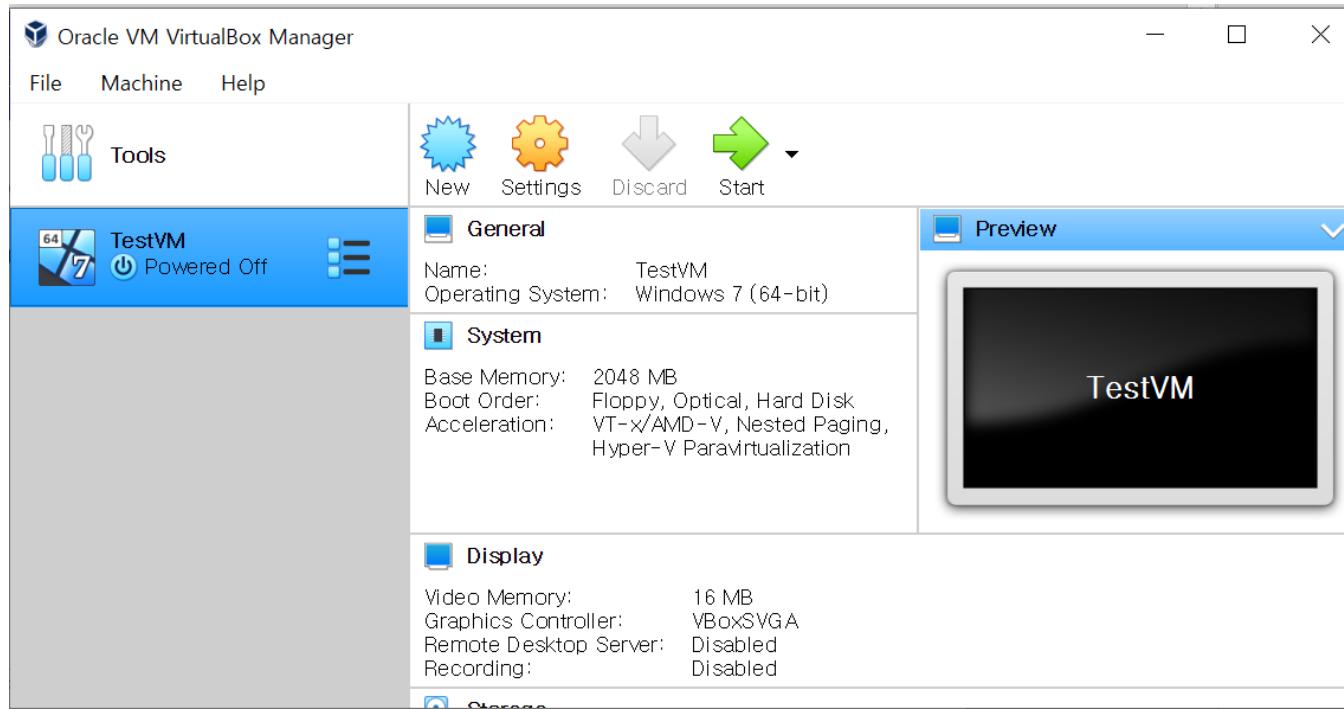
Creating VM

- Click New in the VirtaulBox Manager Window
- A wizard is shown, to guide you through setting up a new VM



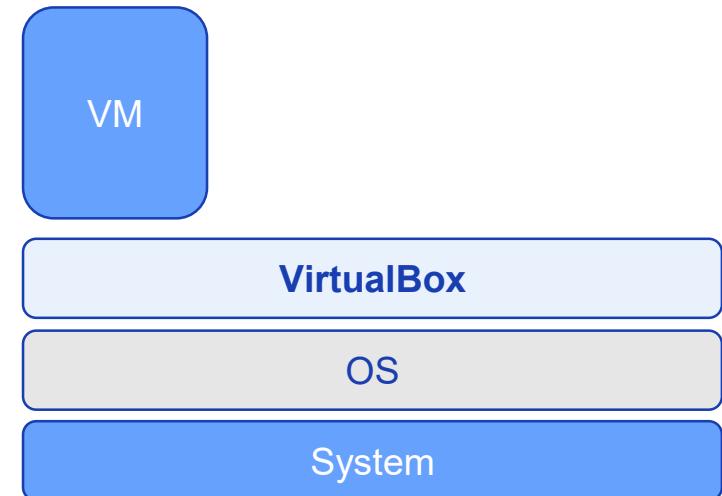
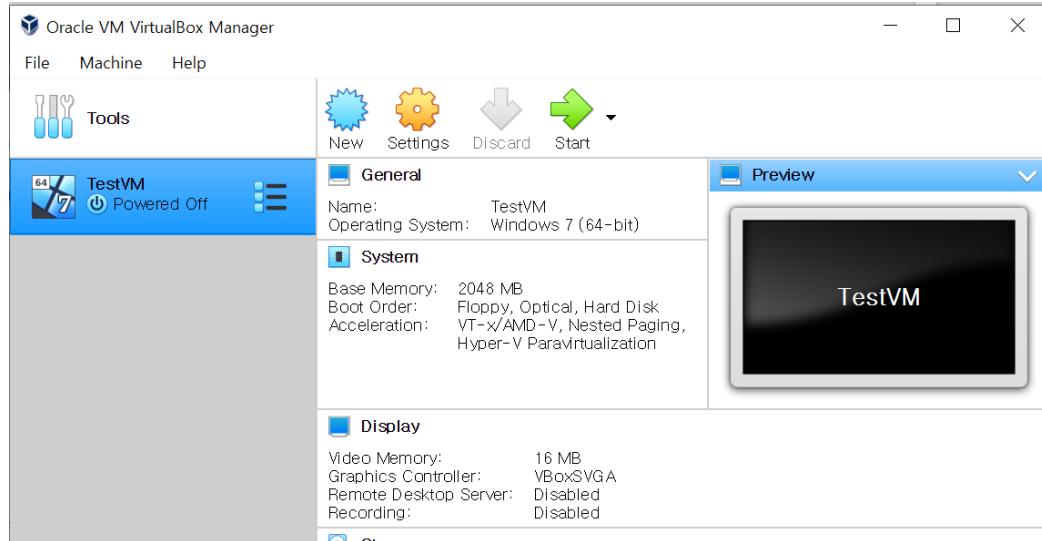
Checking VM information

- Check VM information in the VirtualBox Manager Window



Starting VM

- Several options:
 - Double-click on the VM's entry in the list
 - Select the VM's entry in the list, and click **Start**

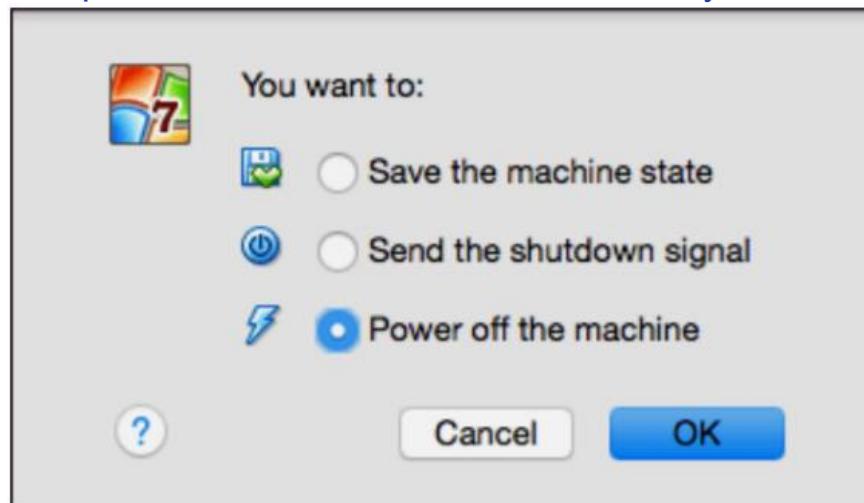


Closing VM

- Click on the Close button of the virtual machine window

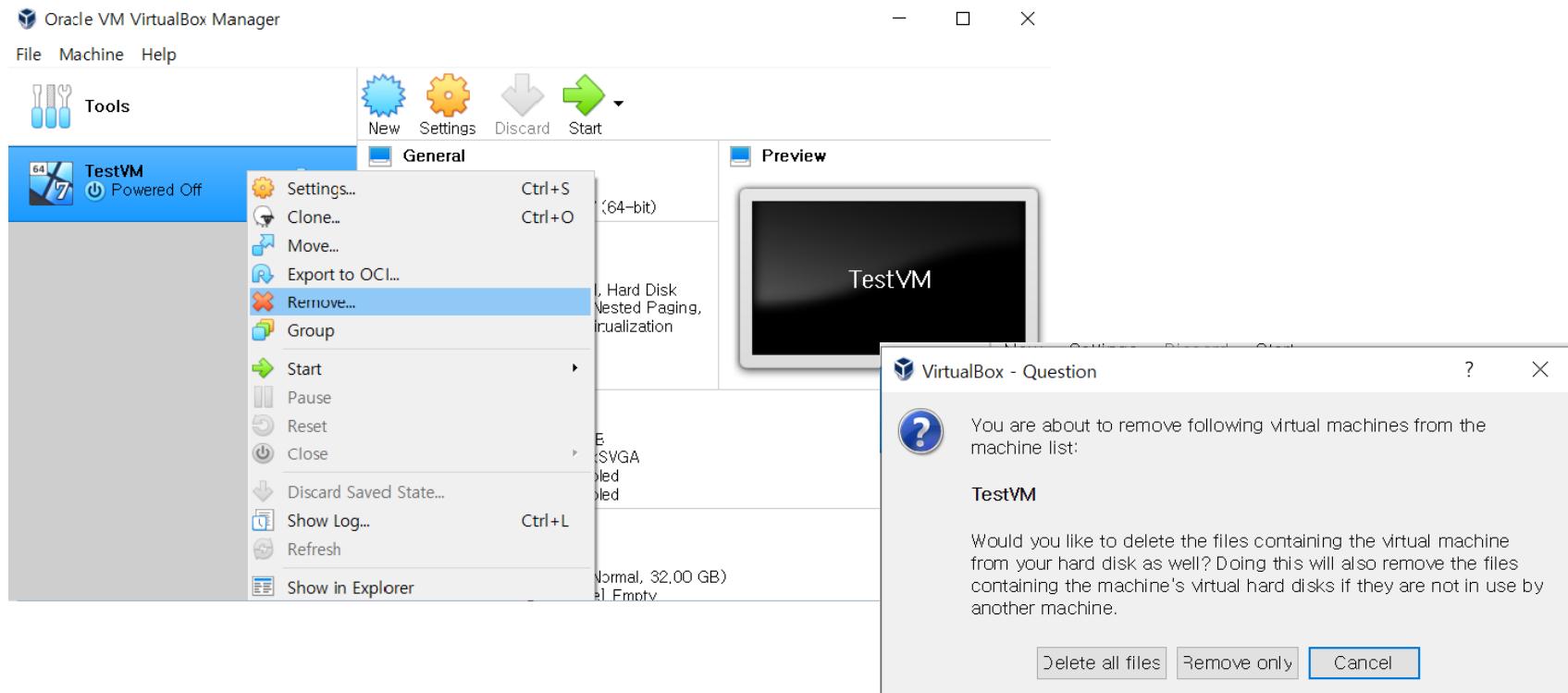


which option is the same as when we normally turn off our PC?



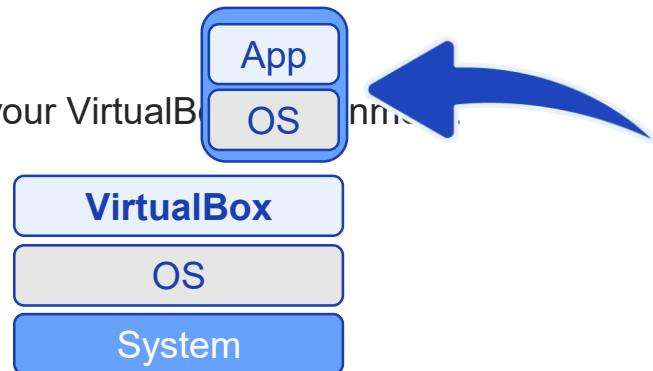
Removing VM

- Right-click on the VM and select Remove

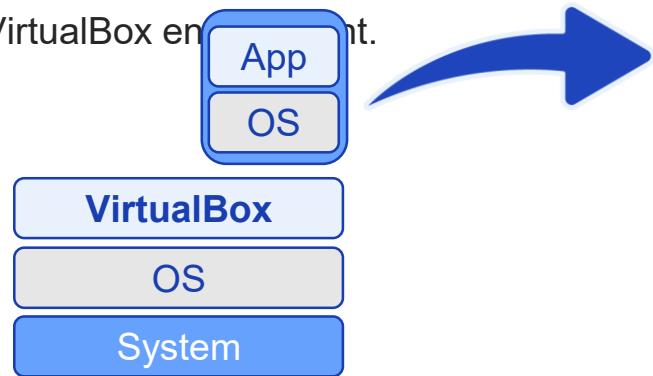


Importing & Exporting

- Importing enables you to easily import virtual machine images to your VirtualBox environment.
 - Don't need to create VM
 - Don't need to install OS and Apps.
 - Just import a well-made VM

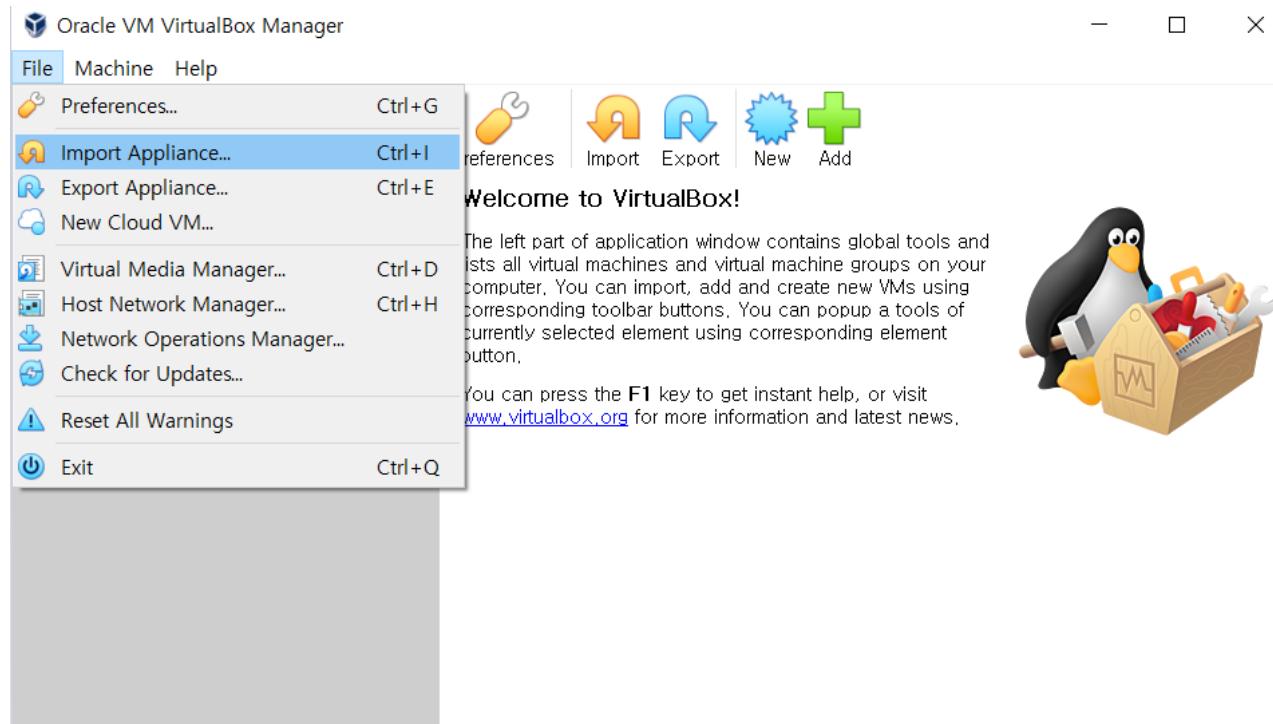


- Exporting enables you to easily export virtual machine from your VirtualBox environment.
 - Can export your VM and move to other PC or environments.



Importing & Exporting

- Importing and Exporting Virtual Machines



[Lab2] Working with HDFS



[Lab3]

Working with YARN/MapReduce



Chapter 3.

Big Data Ingestion

| Big Data Course

Chapter Description

Objectives:

- ✓ We will learn how to use various tools for data ingestion into a Hadoop platform.
 - Data sources can be classified as streaming or non-streaming data.
 - Another major classification is structured, unstructured or semi-structured data.
 - The tools we will learn are optimized for some subset of the above categories
 - We will learn Apache Sqoop, Apache Flume and Apache NiFi for this purpose
- ✓ When ingesting streaming data, it is very important to have a system that will buffer the incoming flow of data.
 - Sometimes the flows can be trickles (just a little bit of data) and sometimes, it might be a flood.
 - Apache Kafka allows us to store and buffer data between Producers and Consumers

Contents of Chapter:

1. Data Migration from EDW
2. Streaming Real-Time Data Sources
3. Creating Data Pipelines with Apache NiFi
4. Apache Kafka

Unit 1.

Data Migration from EDW

Big Data Ingestion

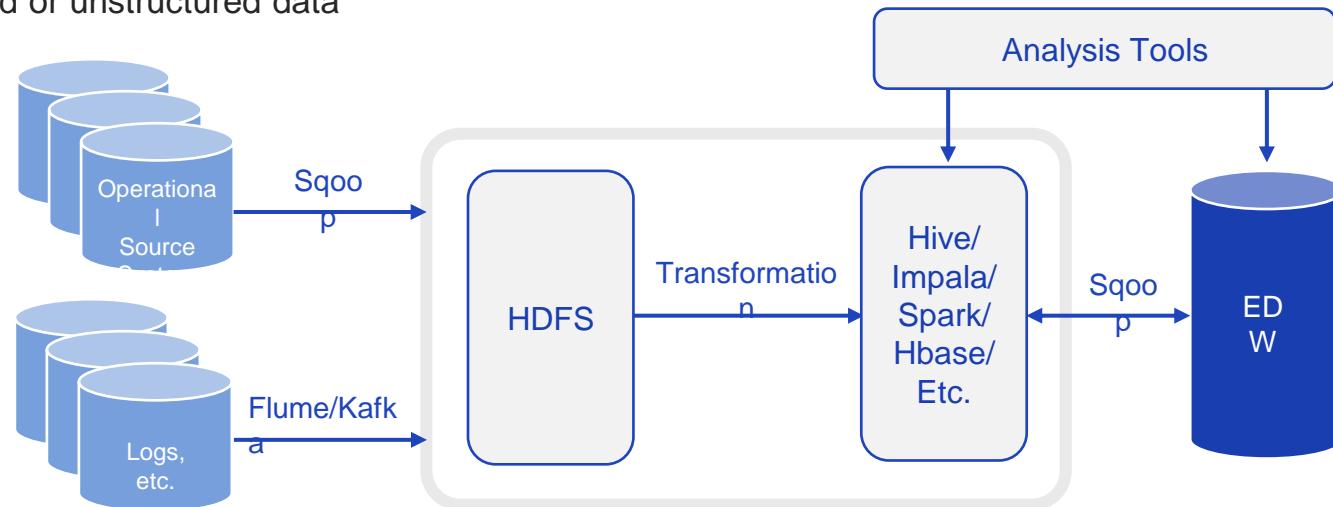
Unit 1.

Data Migration from EDW

- | 1.1. Apache Sqoop Architecture
- | 1.2. Import and Export
- | 1.3. Integration with Hive

Data Migration from RDBMS

- One of the most common data ingestion projects related Big Data is to migrate EDW data to Hadoop
 - Typically EDW data is not saved long term
 - Enterprises will delete older data to make room for the newer "hot" data
 - Data migration to Hadoop allows Enterprises to retain and analyze older data as well as join with other semi-structured or unstructured data



What is Apache Sqoop? (1/2)

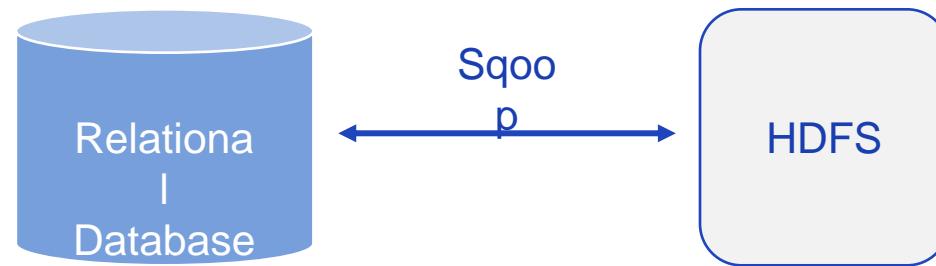
□Quiz□

Apache Sqoop is a tool optimized for
transferring data

between F and

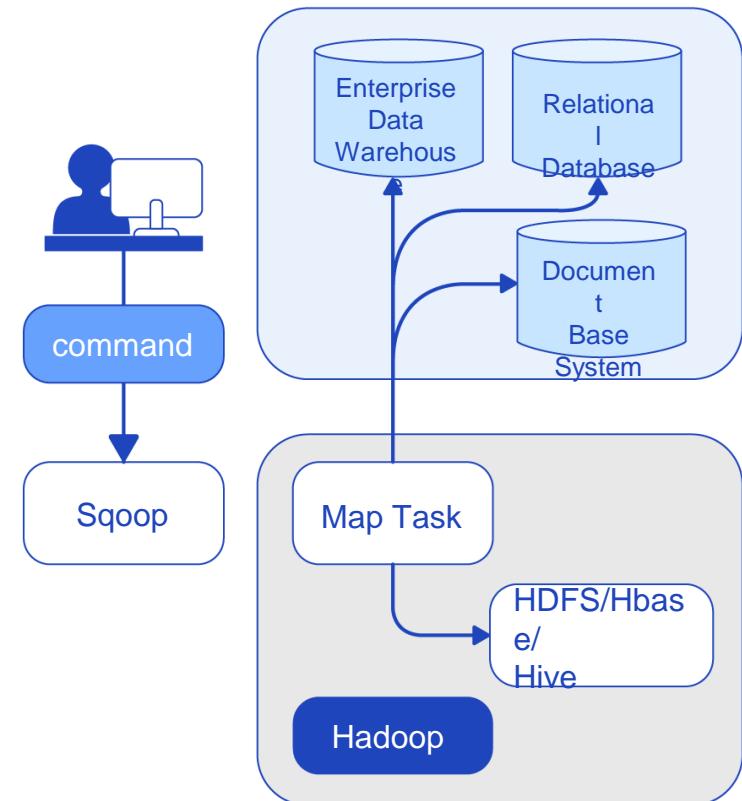
What is Apache Sqoop? (2/2)

- Apache project
 - The name is a contraction of SQL-to-Hadoop
- Sqoop exchanges data between a database and Hadoop
 - Import all tables, a single table, a partial table into HDFS
 - Import in various data formats
 - Export data from HDFS to Database



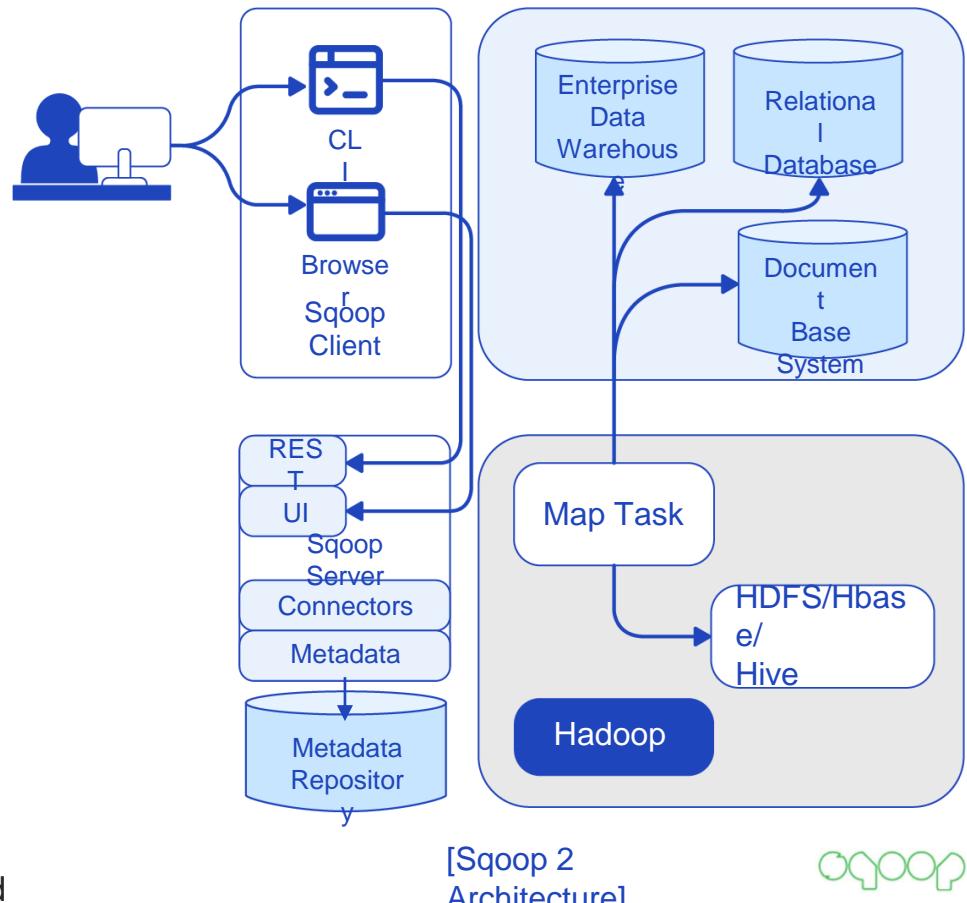
Sqoop 1 Architecture

- Sqoop 1 is a client side application
- Sqoop commands generate and execute MapReduce Job
- The MapReduce job is executed to transfer data between the database and Hadoop
 - Only has Map job – no Reduce job
 - By default the job is partitioned into 4 mappers
 - Each mapper brings the data in parallel
- Sqoop uses JDBC based connectors and interfaces to fetch the metadata for table schema
 - Users provide authorization information

[Sqoop 1
Architecture]

Sqoop 2 Architecture

- Sqoop 2 is a client-server application
- Sqoop 2 exposes REST API as web service that can be used by other systems
- Users communicate and authenticate against a Sqoop Server
 - No need for users to have direct access to the underlying database
- Sqoop Servers manage a metadata repository
 - Includes connections, jobs, submission records
- Sqoop Server manages the connectors and does not have to be JDBC bases
 - Allows not-JDBC compliant databases to be reached



Anatomy of a Sqoop Command

- Sqoop commands are most often issued from the command line
 - Sqoop followed by a subcommand and its list of arguments
- Basic syntax :
 - `sqoop <SUBCOMMAND> <ARGS>`
- Most commonly used subcommands
 - `list-databases`
 - `list-tables`
 - `import-all-tables`
 - `import`
 - `export`
 - `eval`
 - `job`

Common Subcommand Arguments

- Subcommand argument syntax:
 - Two dashes "--" followed by name of argument
 - A space followed by value of the argument
- Common subcommands include:
 - --connect <JDBC connection string>
JDBC connection string syntax:
`jdbc:<database_engine_name>://<server_host_name>:<port>/<database_name>`
 - --user <user name to access database>

```
--connect jdbc:mysql://mysqlserver/mydatabase
--user myuser
--password mypassword
```

Sqoop Usage Examples

- List all databases

```
$ sqoop list-databases --username myuser -P \
--connect jdbc:mysql://dbserver.example.com
```

- List all tables in test database

```
$ sqoop list-tables --username myuser -P \
--connect jdbc:mysql://dbserver.example.com/test
```

- Import all tables from test database

```
$ sqoop import-all-tables \
--username myuser --password mypassword \
--connect jdbc:mysql://dbserver.example.com/test
```

Unit 1.

Data Migration from EDW

- | 1.1. Apache Sqoop Architecture
- | 1.2. Import and Export
- | 1.3. Integration with Hive

Import a Single Table

- Use import subcommand to import a single table
 - Use --table to specify the database table

```
$ sqoop import \
--table sometable \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword
```

- The following default arguments apply
 - The table is stored as a folder in the user's HDFS home directory
 - Saved as comma-delimited text file format

Specify the Save Location

- Save in location other than default HDFS directory
 - By default the table data is stored in a subdirectory of same name as table in the user's HDFS home directory
- We can specify an alternate location to save the data
 - Use --target-dir argument

```
$ sqoop import \
--table sometable \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--target-dir /some/other/directory
```

Specify a Base Directory Location

- When importing several tables, we can specify the base directory location
 - Each table will be saved as a subdirectory of table name
 - Use --warehouse-dir argument

```
$ sqoop import-all-tables \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--warehouse-dir /some/directory/path
```

Qualifying Import Records

- Import only specified columns
 - Use --columns with a comma delimited list of columns

```
$ sqoop import \
--table sometable \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--columns "fname, lname, address, state, zip"
```

- Import only matching rows
 - Use --where with a condition string

```
$ sqoop import \
--table sometable \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--where " state='CA' "
```

Specifying Text Delimiter

- Sqoop saves as comma-delimited text file by default
- We can specify other delimiters
 - Use `--fields-terminated-by` argument with delimiter character
 - For example, maybe we want to use tabs as the delimiter

```
$ sqoop import \
--table sometable \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--fields-terminated-by '\t'
```

Specifying Data Format

- Sqoop saves as comma-delimited text file by default
- We can specify other data formats
 - Use --as-avrodatafile to save as Avro data format
 - Use --as-sequencefile to save as SequenceFiles data format
 - Use --as-textfile to save as text format (default)
 - Use --as-parquetfile to save as Parquet data format

```
$ sqoop import \
--table sometable \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--as-parquetfile
```

Compression Options

- Sqoop supports various compression codecs
 - Use --compress or -z to enable compression
- Default compression is gzip
 - Specify different compression codec with --compression-codec

```
$ sqoop import \
--table sometable \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--compress --compression-codec \
org.apache.hadoop.io.compress.SnappyCodec
```

Hadoop Compression Codecs

Compression Format	Hadoop Compression Codec
DEFLATE	org.apache.hadoop.io.compress.DefaultCodec
Gzip	org.apache.hadoop.io.compress.GzipCodec
Bzip2	org.apache.hadoop.io.compress.BZip2Codec
LZO	com.hadoop.compression.lzo.LzopCodec
Snappy	org.apache.hadoop.io.compress.SnappyCodec
LZ4	org.apache.hadoop.io.compress.Lz4Codec
Zstandard	org.apache.hadoop.io.compress.ZstandardCodec

Incremental Import Append Mode

- Sqoop provides an incremental mode
 - Retrieves only rows newer than some previously imported rows and append them
- Use --incremental and specify append mode
- Use --check-column to specify the column when determining rows to append
- Append rows for which the check-column value is greater than --last-value

- Often this will be the primary key column

```
$ sqoop import \
--table sometable \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--incremental append \
--check-column some_id --last-value 245
```

Incremental Import LastModified

- Sqoop provides an incremental mode
 - Retrieve only rows newer than some previously imported rows
- Use --incremental and specify lastmodified mode
- Use --check-column to specify the column when determining rows to append
- Append rows for which the check-column value is greater than --last-value

- This must be a timestamp or date type column

```
$ sqoop import \
--table sometable \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--incremental lastmodified \
--check-column some_timestamp \
--last-value "2021-06-26 18:19:25"
```

Import Based on Free-form Query

- Sqoop can import the result set of an arbitrary SQL query
- No need to use --table, --columns, and --where arguments
 - Retrieve only rows newer than some previously imported rows
- Use --query and specify the SQL query
- SQL query must include \$CONDITIONS as a WHERE condition
- If results will be retrieved in parallel by multiple mappers,
 - Must include --split-by and specify the column to use in partitioning the dataset for retrieval
- All free-form query imports must specify the --target-dir to store the results

```
$ sqoop import  
  --connect jdbc:mysql://dbhost/somedatabase \  
  --username somename --password somepassword \  
  --query 'SELECT a.* , b.* from a JOIN b on (a.id == b.id) WHERE $CONDITIONS' \  
  --split-by a.id --target-dir /user/myuser/results
```

Export Data to RDBMS

- Use export subcommand to export data in HDFS to a single table in RDBMS
 - Use --export-dir to specify the HDFS source path
 - Use --table to specify the RDBMS table to populate
- Exports the contents of the entire directory to the table
- Primarily used to populating a newly created empty table

```
$ sqoop export \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--table sometable \
--export-dir /path/to/data/directory
```

Update Existing RDBMS Table

- Use export subcommand to export data in HDFS to a single table in RDBMS
- Use --update-key to specify a column or list of columns to match rows
- The sqoop command is converted to UPDATE commands on the database

```
$ sqoop export \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--table sometable \
--export-dir /path/to/data/directory \
--update-key some_column
```

Update or Insert Existing RDBMS Table

- Specify whether to allow insert or only updates with --update-mode <mode>
 - updateonly (default) only updates matching rows
 - allowinsert update as well as insert any non-matching rows

```
$ sqoop export \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--table sometable \
--export-dir /path/to/data/directory \
--update-key some_column \
--update-mode allowinsert
```

Running a Simple SQL Query

- Use eval subcommand to run an SQL query on the database
 - Use --query to specify the SQL query
 - The SQL query can be any query including DDL, DML and SQL

```
$ sqoop eval \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--query "SELECT * from MYTABLE LIMIT 10"
```

```
$ sqoop eval \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--query "DESCRIBE MYTABLE"
```

Creating Sqoop Jobs

- Use job subcommand to create and work with saved jobs
- Saved jobs remember the parameters used to specify a job
- Re-execute by invoking the job id
- job subcommand arguments
 - --create <job id> to save a job
 - --delete <job id> to delete a saved job
 - --exec <job id> to execute a saved job named <job id>
 - --show <job id> to show the parameters of a saved job
 - --list list all saved jobs
- Use the following generic syntax:
`-- sqoop job <JOB ARGS> -- <SUBCOMMAND> <SUBCOMMAND ARGS>`

Example Sqoop Job to Import Table

```
$ sqoop job --create myjob -- import \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--table sometable
```

```
$ sqoop job --list
Available jobs:
    myjob
```

```
$ sqoop job --exec myjob
```

Unit 1.

Data Migration from EDW

- | 1.1. Apache Sqoop Architecture
- | 1.2. Import and Export
- | 1.3. Integration with Hive

Importing Data Into Hive Table

- Use --hive-import argument to import into a hive table
- Use --hive-table <table name> to set the name of the hive table
- Without any option, Sqoop will create the Hive table and import the data
 - Use --hive-overwrite to overwrite existing data in the Hive table
 - Use --create-hive-table to check if hive table already exists and fail if so

```
$ sqoop import \
--table sometable \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--hive-import --hive-table myhivetable
```

Creating Hive Table Metastore

- Sometime we want to just create the Hive table definition without actually loading the data yet
 - Multiple departments are going to analyze a dataset to be imported from RDBMS
 - Some will use Apache Spark to create data models
 - Others will want to use Hive for batch processing with other datasets
- Use `create-hive-table` subcommand to simply create the Hive metastore

```
$ sqoop create-hive-table \
--table sometable \
--connect jdbc:mysql://dbhost/somedatabase \
--username somename --password somepassword \
--hive-table myhivetab
```

[Lab1]

Data Ingestion with Sqoop for RDBMS (MariaDB)



Unit 2.

Streaming Real-Time Sources

Big Data Ingestion

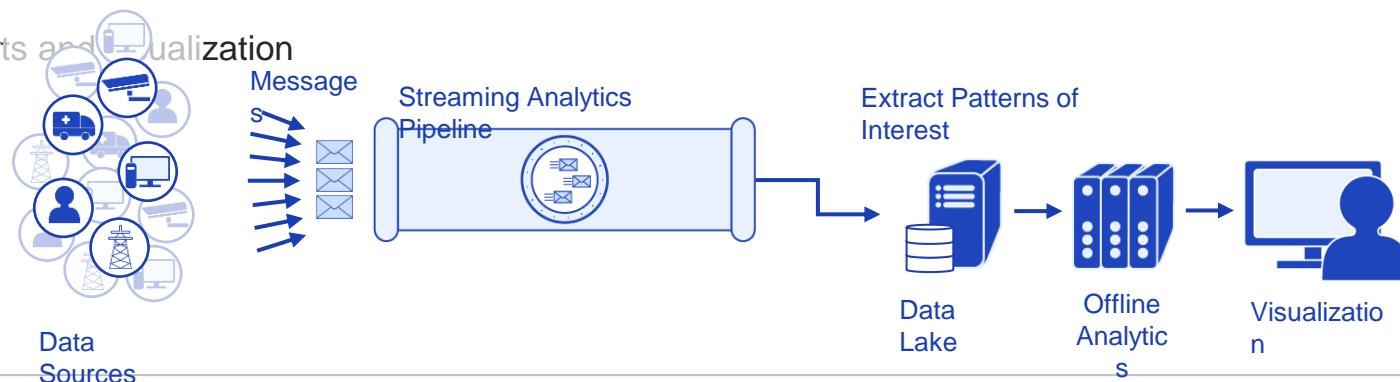
Unit 2.

Streaming Real-Time Sources

- | 2.1. Anatomy of Real-Time Streaming
- | 2.2. Apache Flume

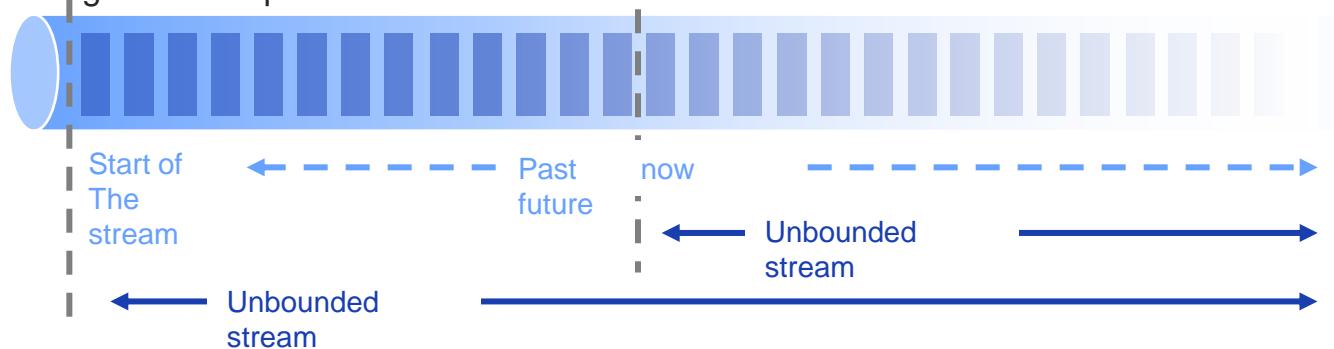
Real-Time Streaming Data - Fundamentals

- Capture and ingest data from data sources that produce data in continuous streams
 - Ex: Sensor data, smart phones, web site logs, smart devices, self-driving cars, etc.
 - Handle bursts of data without loosing information
- Store and categorize the data for long-term usage
- Process the data in real-time
 - Extract, transform and clean-up the data
 - Query the data in real-time
 - Produce reports and visualization



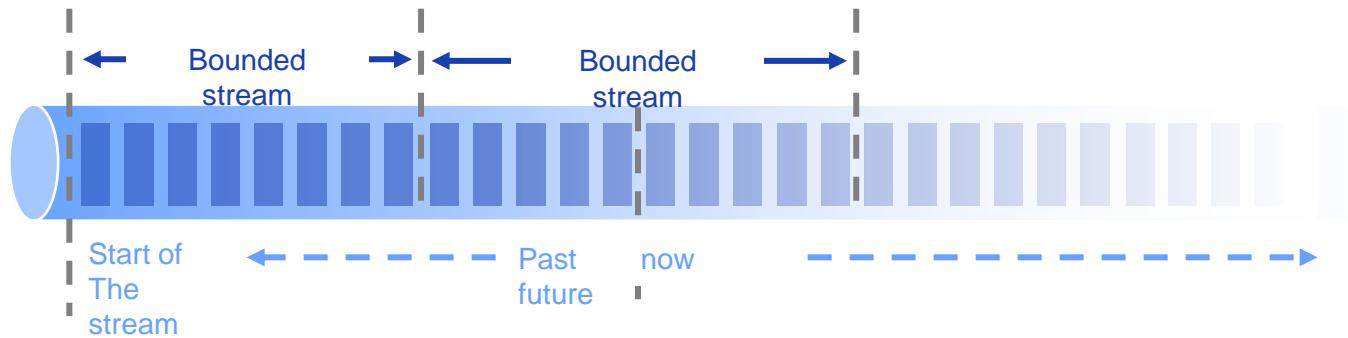
Unbounded Data Streams

- Unbounded data stream has some starting point
 - May be very far in the past and not of any consequence in processing current incoming data
- Incoming data stream has no defined end
 - The incoming data stream is not expected to terminate at any time
- Unbounded streams must be ingested in real-time and in specific order
 - It is not possible to wait until the entire stream is read - it may never end
 - Events must be ingested and processed at the time the event occurred



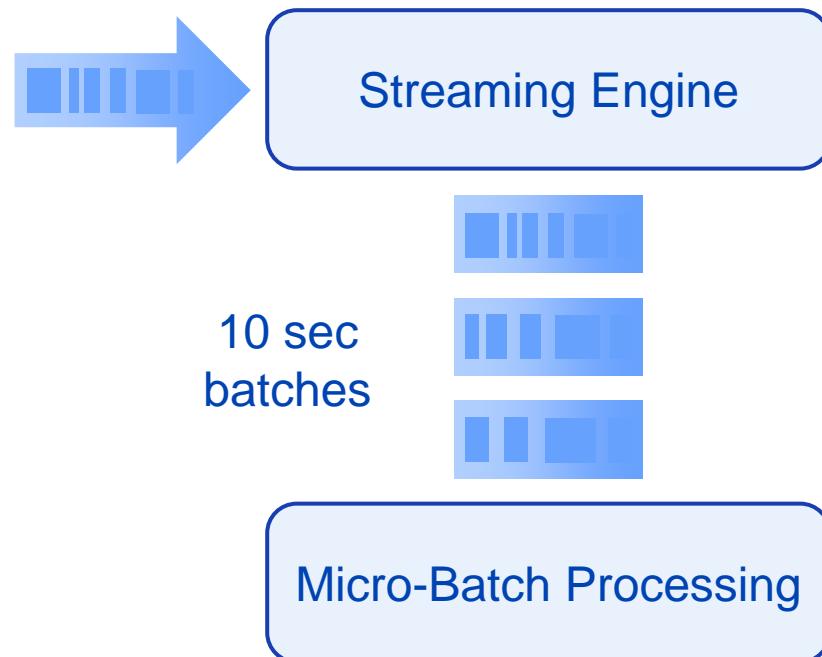
Bounded Data Streams

- Bounded data stream has a starting point and an ending point
- Possible to ingest and store the entire dataset before processing
- Ingesting and processing events in order is not as strict
 - Data may be re-sorted to match the desired query since it has been saved



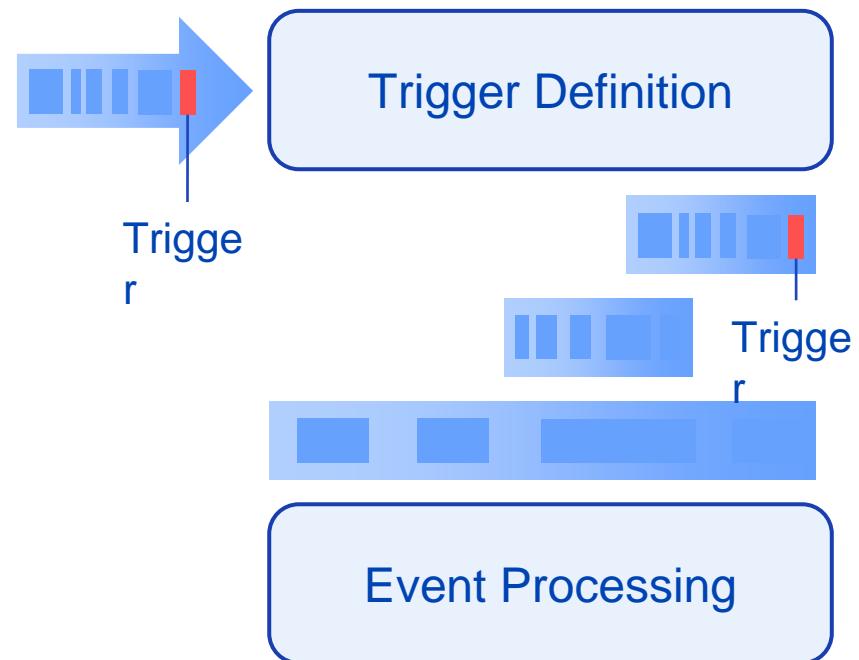
Micro-Batch Processing

- Data arrives from source randomly
 - Size of data is random
 - Arrival time is random
- Group together a sequence of data
 - Usually based on time
 - Ex: Group together messages that have arrived in the last 10 seconds
- The grouped sequence of data is referred to as a micro-batch
- Processing sequences of micro-batches is micro-batch processing



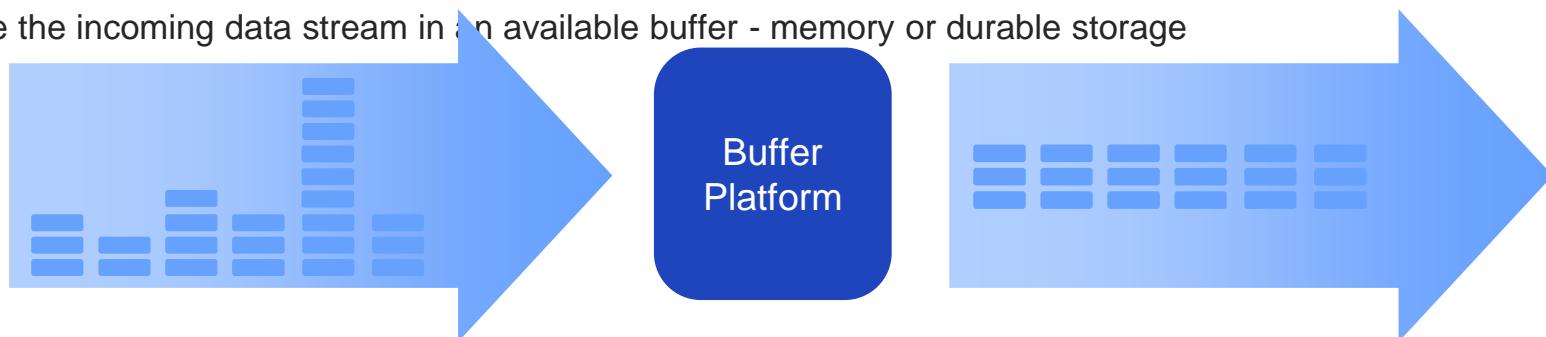
Event-Based Triggered Processing

- Data arrives from source randomly
 - Size of data is random
 - Arrival time is random
- Create and setup triggers
 - Rules that activate based on incoming data messages and events
 - Rules will be highly dependent on use case scenario
- The data processed upon triggering will depend on the rules set on the trigger
- Processing when a trigger is activated is event based processing



Buffering Incoming Data

- Upstream
 - A data source from where we are receiving real-time streaming data
- Downstream
 - A data destination to which we are sending real-time streaming data
 - Often, we perform some type of processing before we send it downstream
- Back Pressure and Buffering
 - Throttling the upstream data source until normal operation is achieved
 - Queue the incoming data stream in an available buffer - memory or durable storage



Unit 2.

Streaming Real-Time Sources

- | 2.1. Anatomy of Real-Time Streaming
- | 2.2. Apache Flume

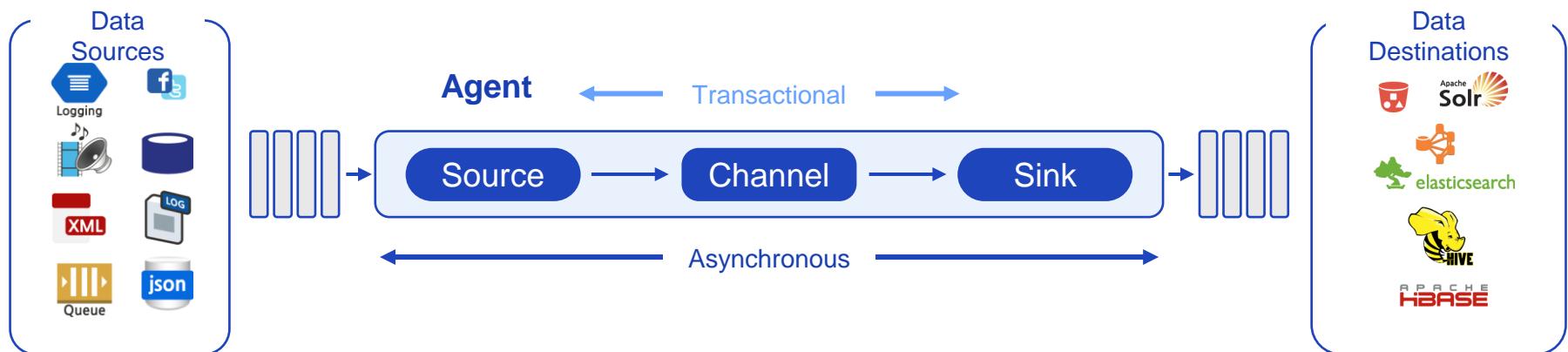
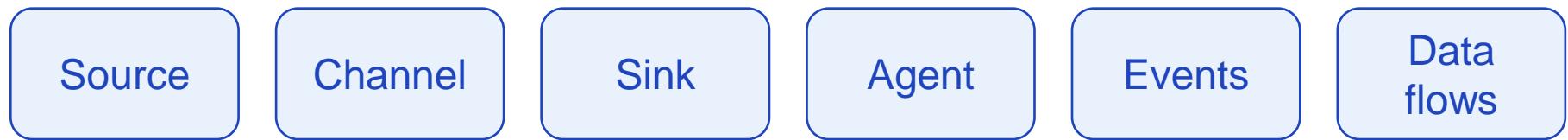
What is Apache Flume

- High-performance system for collection and aggregation of streaming event data
 - Distributed
 - Reliable
 - Available
- Originally developed by Cloudera
 - Donated to Apache Software Foundation in 2011
 - Became a top-level Apache project in 2012
 - Flume OG (old generation) → Flume NG (new generation)



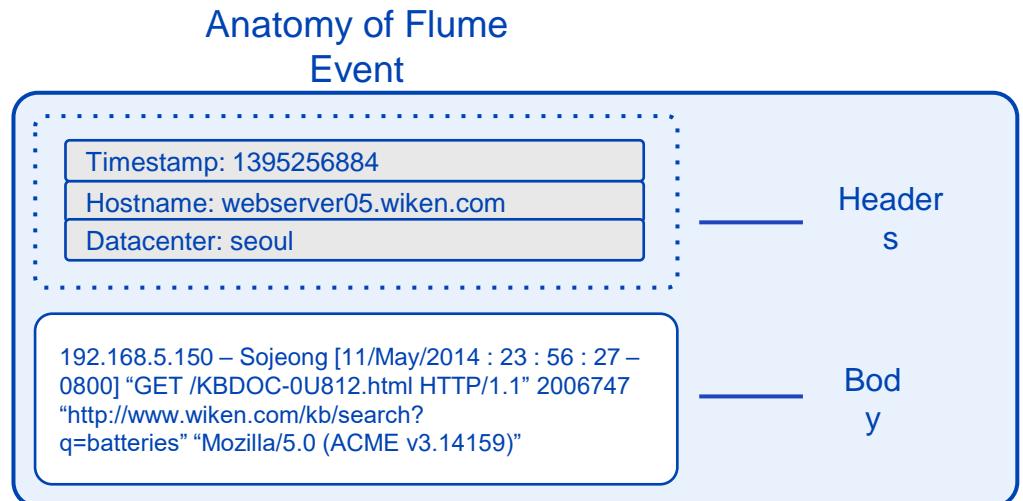
Apache Flume Components

- 6 Main Components



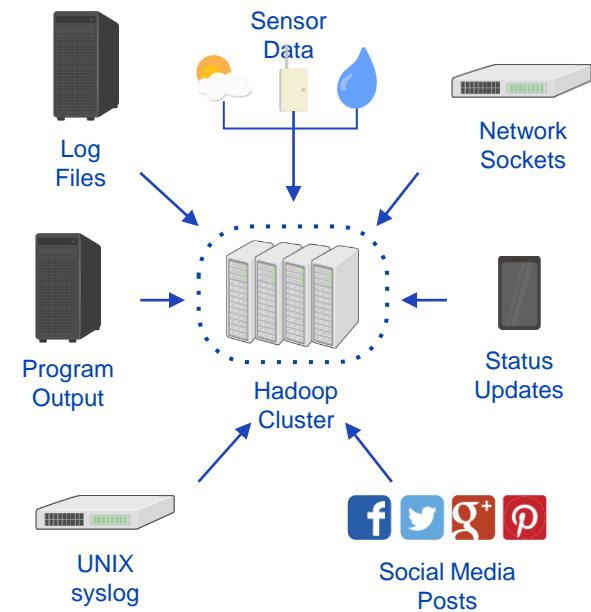
Apache Flume - Event

- A Flume Event is defined as a unit of data flow having a byte payload (body) and an optional set of string attributes (headers).
- Fundamental unit of data transported by Flume from source to destination
- Headers are collection of string key-value pairs
- Keys are unique across the collection
- Headers can be used for contextual routing



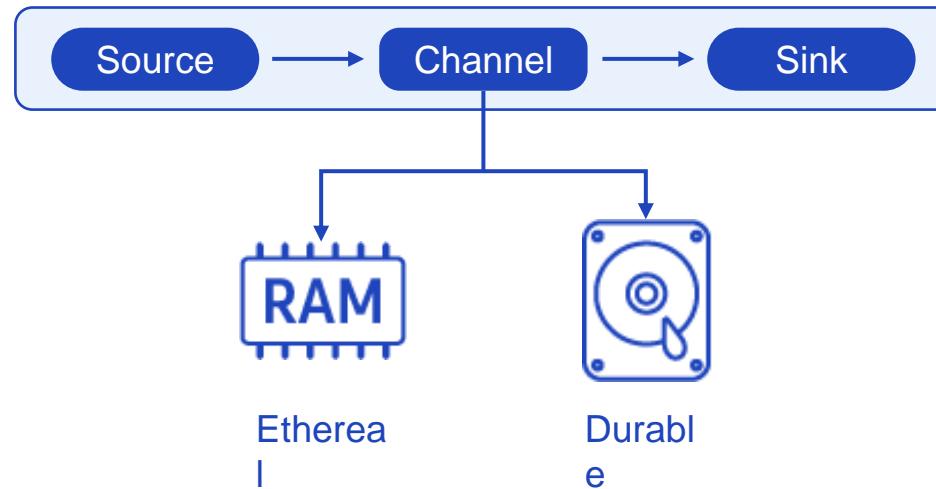
Apache Flume - Source

- An active component that receives Events from an external location or mechanism and places the Event in one or more Channels
- Requires at least one Channel to function
- Common Sources
 - Syslog - Captures messages from the UNIX syslog daemon
 - Netcat - Captures data written to a socket on a TCP port
 - Exec - Executes a UNIX program and reads events from the standard output
 - Spooldir - Extracts events from files appearing in a specified local directory
 - HTTP Source - Retrieves events from HTTP requests
 - Kafka - Retrieves messages from a Kafka topic



Apache Flume - Channel

- A passive component that buffers incoming Events until they are drained by Sinks
- Channels are transactional
 - A failed data transfer to a downstream agent will roll back and retry
- Can work with many different types of Sources and Sinks
- Common Channels
 - Memory
 - Disk



Apache Flume - Built-In Channel

- Type of Channel determines level of persistence

Memory Channel

Stores event in RAM

Provides high throughput but loss of power or overflow may lead to data loss

JDBC Channel

Embedded database with ACID property

Atomicity, Consistency, Isolation, Durability

File Channel

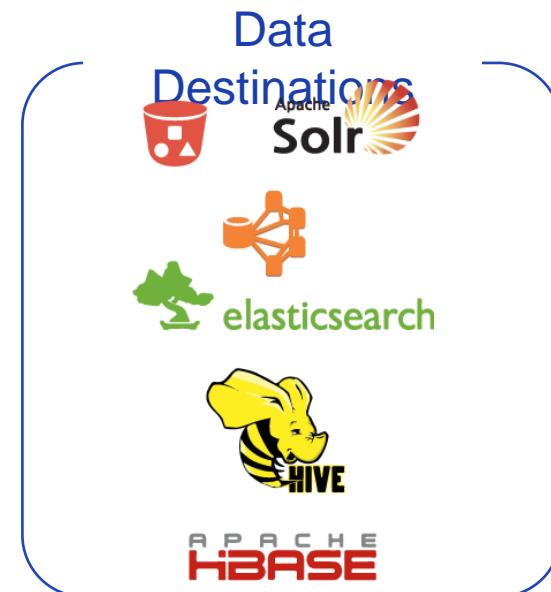
Stores event in durable local disk
A persistent channel that guarantees that when a transaction is committed, no data will be lost due to subsequent crash or power loss
Write-ahead logging

Kafka Channel

Takes advantage of Kafka's scalable, reliable and highly available architecture

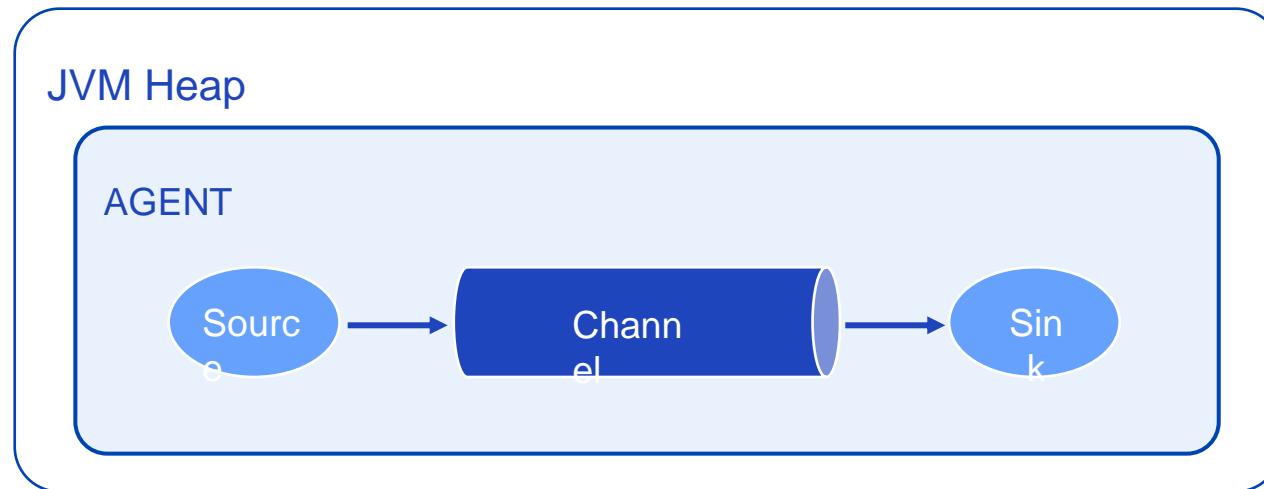
Apache Flume - Sink

- An active component that removes Events from a Channel and transmits them to their next hop or destination
- Requires exactly one Channel to function
- Often used Flume Sinks
 - Logger – Logs event to INFO level using SLF4J
 - IRC – Sends event to specified Internet Relay Chat
 - HDFS – Writes event to a specified directory and file in HDFS
 - Kafka – Sends event as a message to a Kafka topic
 - HBaseSink – Stores event in HBase



Apache Flume - Agent

- A Java process that runs in a JVM
- Hosts the components through which events flow from an external source to the next destination (hop)
- Fundamental part of a Flume flow
- Provides configuration, life-cycle management, and monitoring support for hosted components



Apache Flume - Data Flows

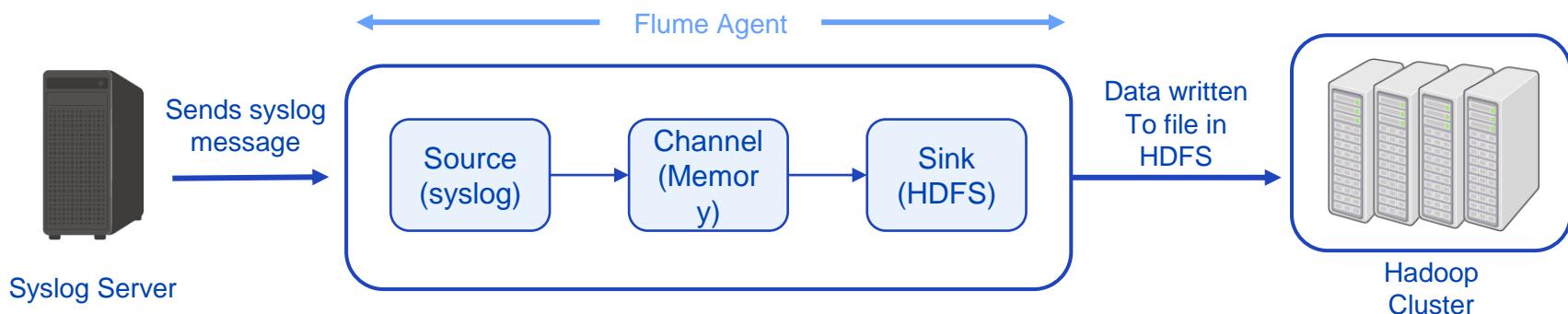
- Apache Flume defines a dataflow by identifying the Source, Channel and Sink
 - A text-based configuration file
 - Each Source, Channel and Sink is further configured with relevant parameters
 - The configuration file is read by an Agent which then hosts each of the components to create the data flow



<https://flume.apache.org/releases/content/1.9.0/FlumeUserGuide.html>

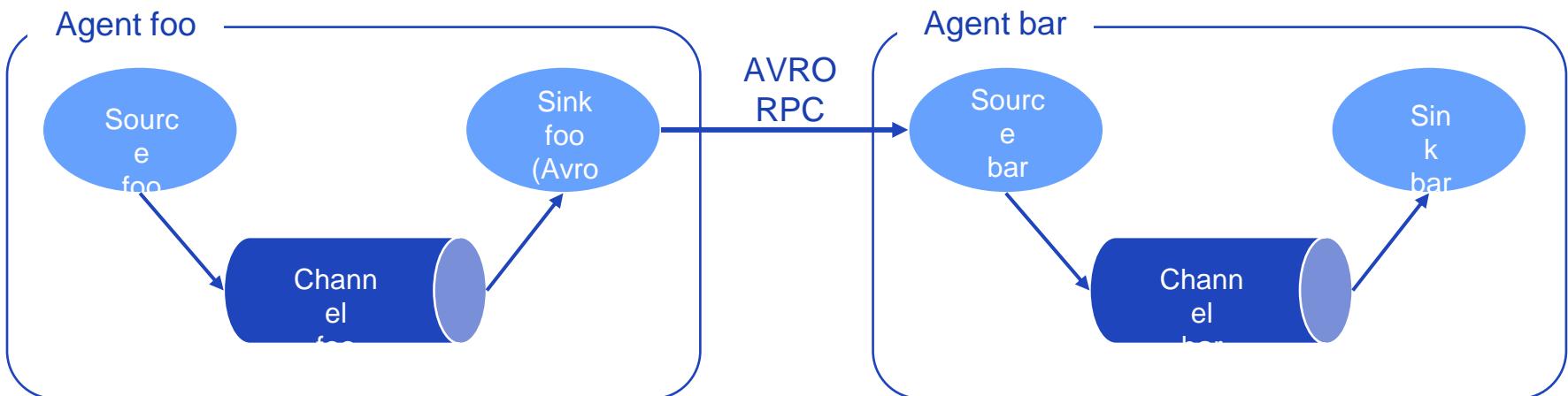
A Simple Data Flow Example

- Capturing syslog data and saving to HDFS
 - Server running syslog daemon logs messages
 - Flume agent configured with Syslog source captures the message and stores it as a Flume event
 - Syslog source pushes the event to a memory channel, where it is buffered in memory
 - Flume agent configured with HDFS sink pulls the event from the memory buffer and writes the message to HDFS



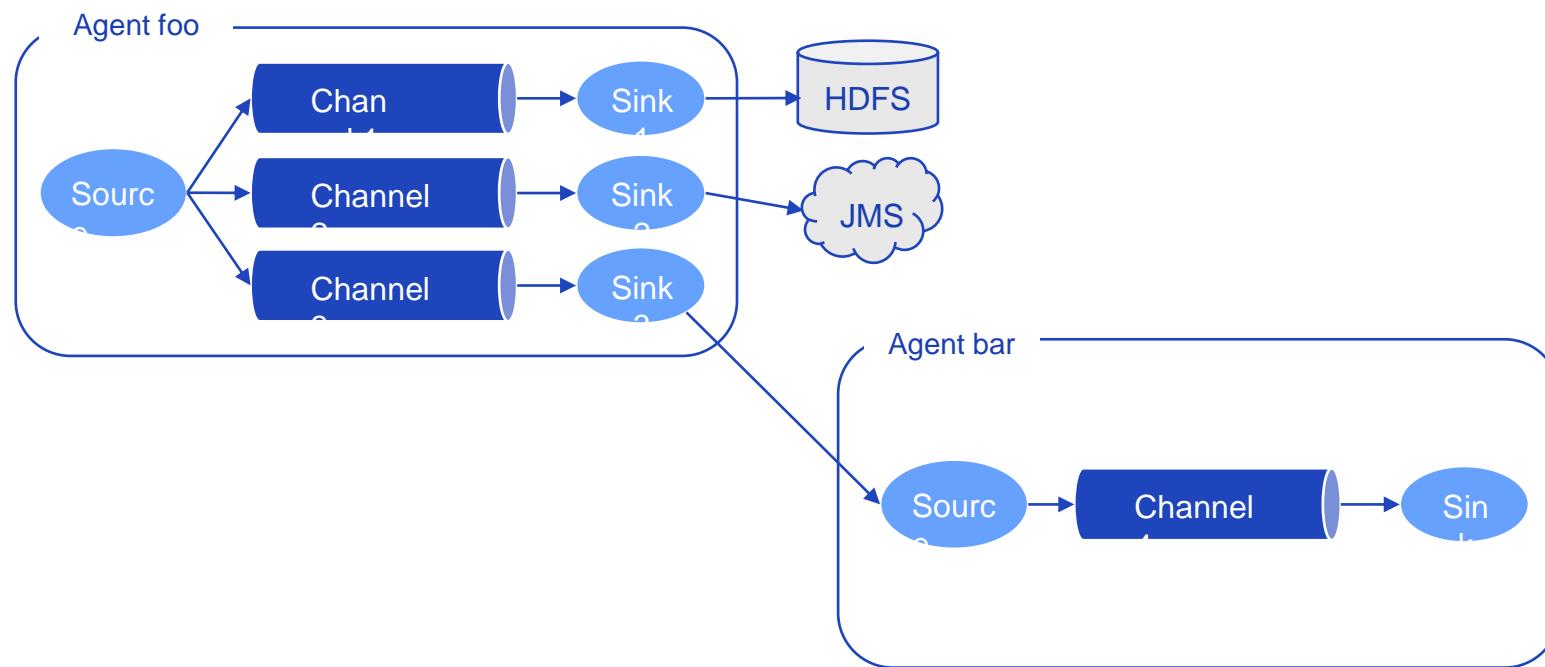
Complex Data Flows - Multi Agent Flow

- Apache Flume supports multi-agent flows
 - Sink of previous agent and Source of current hop must be Avro type
 - The Sink from previous agent points to hostname and port of destination agent



Complex Data Flows - Multiplexing Flow

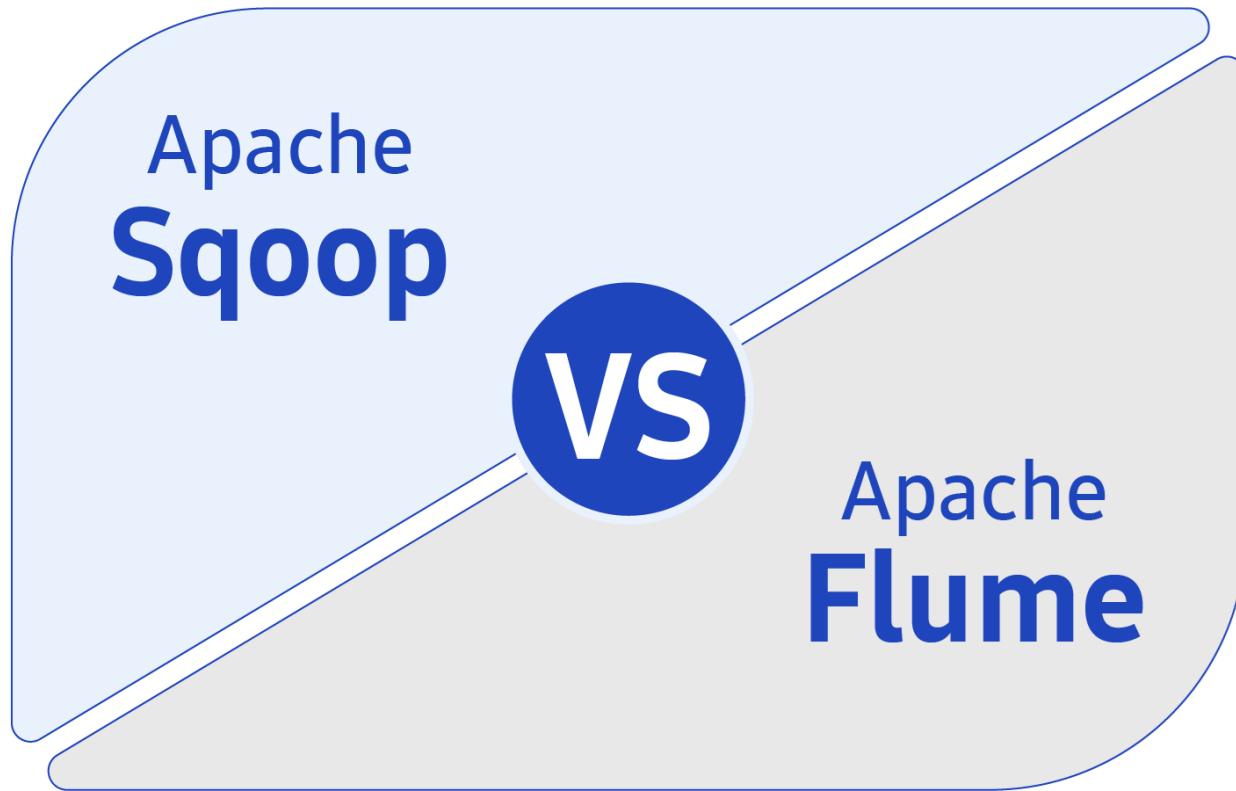
- Apache Flume supports multiplexing the event flow to one or more destinations



Apache Flume vs Sqoop

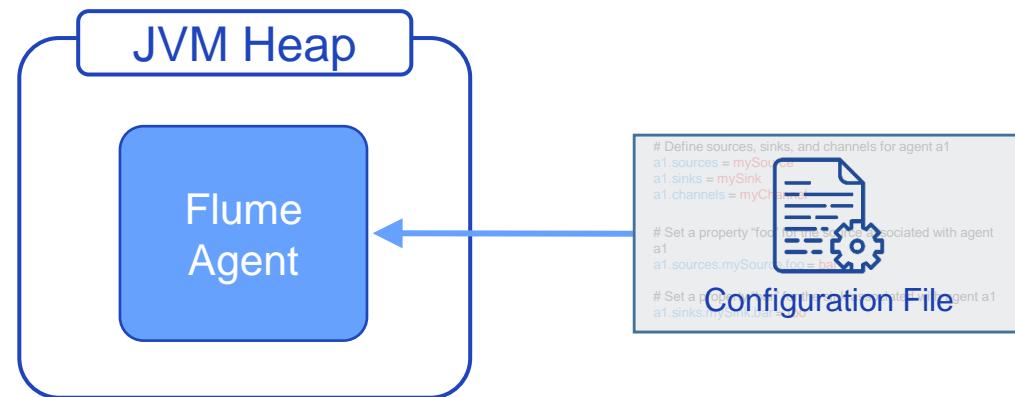
[Discussion]

- Which one is better?



Setting up an Agent

- Configure Flume Agents through a local configuration file
 - Text file that follows the Java properties file format
 - Configuration file includes properties of each Source, Channel and Sink
 - Includes how they are connected together to form a data flow
- Java properties file format
 - # Comment line
 - key1 = value1
 - key2 = multi-line \
 value



Apache Flume Configuration File

- Each component in the flow is given a name, type and set of properties specific to the type selected
- Hierarchical name based configuration
 - agent1.channels.myChannel.type = FILE
 - agent1.source.mySource.type = netcat

```
# Define sources, sinks, and channels for agent a1
a1.sources = mySource
a1.sinks = mySink
a1.channels = myChannel

# Set a property "foo" for the source associated with agent
# a1
a1.sources.mySource.foo = bar

# Set a property "bar" for the sink associated with agent a1
a1.sinks.mySink.bar = foo
```



Example Configuration File

- Name of configuration file is example.conf
- Single agent → a1
- a1 has a source that listens for data on port 44444
- A sink that logs event data and also outputs to the console
- A channel that buffers event data in memory
- Defines the data flow from Source to Sink through the **memory Channel**

```
# example.conf: A single-node Flume configuration

# Name the components on this agent
a1.sources = mySource
a1.sinks = mySink
a1.channels = myChannel

# Describe/configure the source
a1.sources.mySource.type = netcat
a1.sources.mySource.bind = localhost
a1.sources.mySource.port = 44444

# Describe the sink
a1.sinks.mySink.type = logger

# Use a channel which buffers events in memory
a1.channels.myChannel.type = memory
a1.channels.myChannel.capacity = 1000
a1.channels.myChannel.transactionCapacity = 100

# Bind the source and sink to the channel
a1.sources.mySource.channels = myChannel
a1.sinks.mySink.channel = myChannel
```

Starting a Flume Agent

- Agents are started using a shell script called flume-ng
- Need to specify:
 - Agent name using --name <agent name>
 - Configuration file using --conf-file <name of configuration file>
 - Apache Flume configuration directory using --conf <path to Flume configuration directory>
- Start the sample configuration from previous slide with:

```
$ sudo flume-ng agent \
  --conf /etc/flume-ng/conf \
  --conf-file example.conf \
  --name a1 \
  --Dflume.root.logger=INFO, console
```

[Lab2] Data Ingestion with Apache Flume



Unit 3

Creating Data Pipelines with Apache NiFi

| 3.1. Apache NiFi Fundamentals

| 3.2. Apache NiFi User Interface

| 3.3. Apache NiFi Processors

| 3.4. Apache NiFi Connections

| 3.5. Creating Data Flows

| 3.6. Process Group

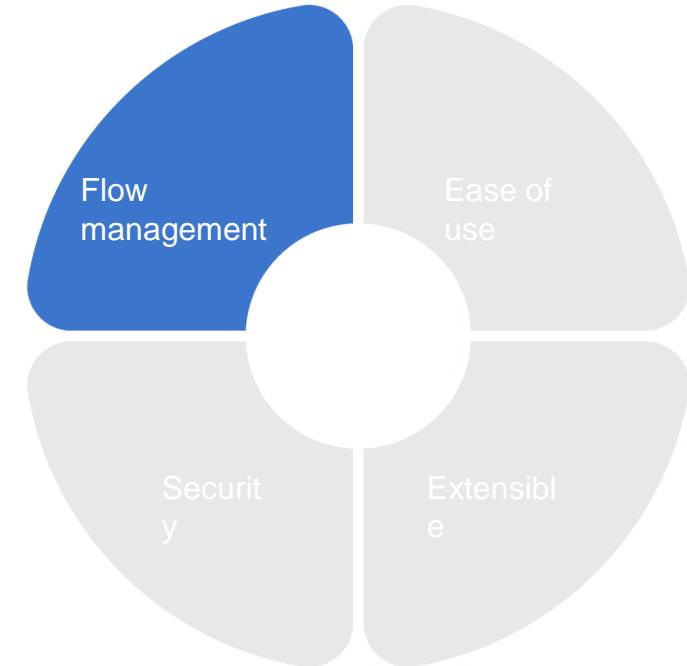
What is Apache NiFi?

- Tool built to help automate the flow of data between systems
 - Create automated and managed data flow between systems
- Provides a web-based graphical user interface
 - Create, configure, and monitor data flows
 - No programming necessary



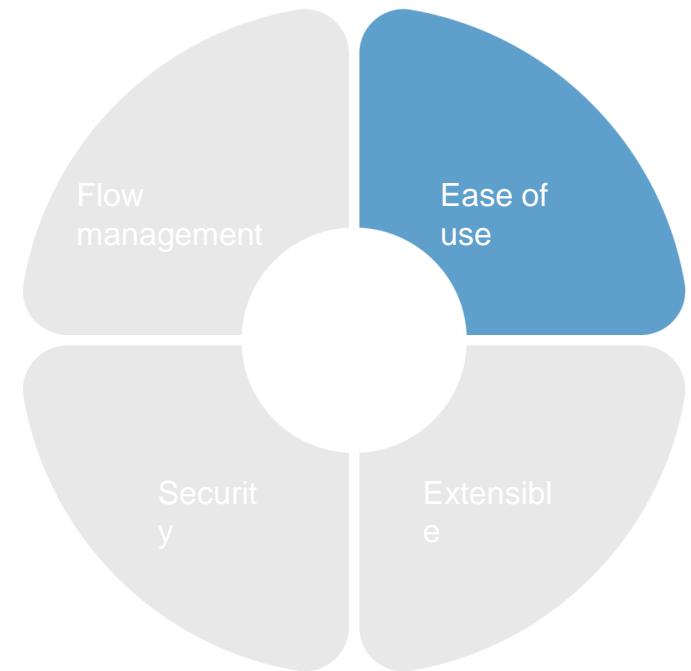
Apache NiFi Features - Flow Management

- Guaranteed delivery
 - Persistent write-ahead log
 - Content repository
- Buffering with Back Pressure
 - All connections provide buffering
 - Set and configure queue to activate back pressure
- Prioritized Queuing
 - Set prioritization schemes for how data is retrieved from queues
 - Based on timestamps, size, or other custom schemes
- Flow Specific Quality of Service
 - No loss of data
 - Throughput .vs. latency



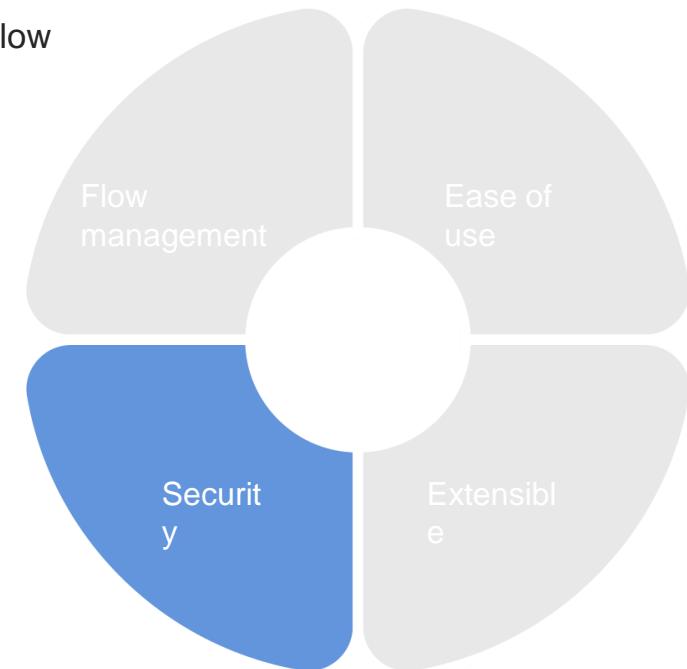
Apache NiFi Features - Ease of Use

- Visual Command and Control
 - Browser based Graphical User Interface
 - Click and drag creation of data flow definitions
- Flow Templates
 - Define and reuse data flows
 - Publish, share and collaborate
- Data Provenance
 - Automatically keep track of data flow lineage



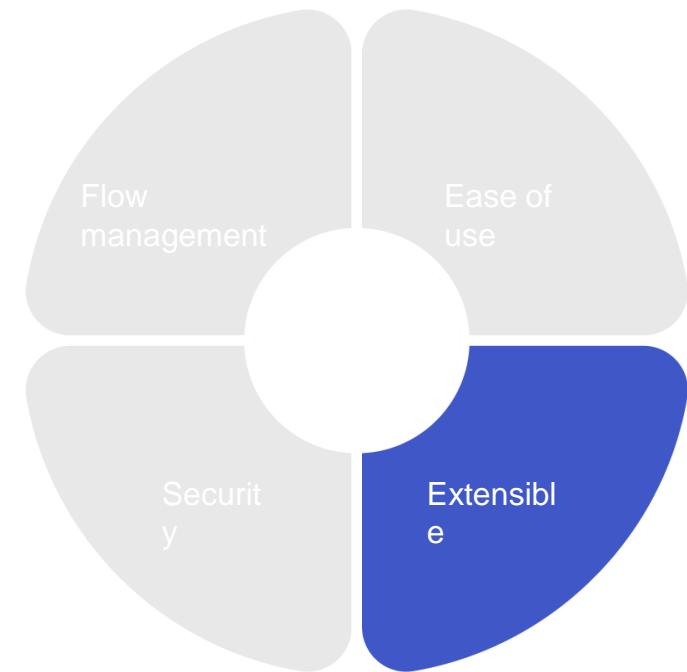
Apache NiFi Features - Security

- Encryption
 - Employs 2-way SSL encryption for data as it travels through the data flow
 - Encrypt and decrypt using shared keys
- Security Policies
 - 2-Way SSL Authentication and pluggable authorization
- Multi-tenant Authorization
 - Allows team within organization to be aware of other flows
 - Can control authorization by and between groups



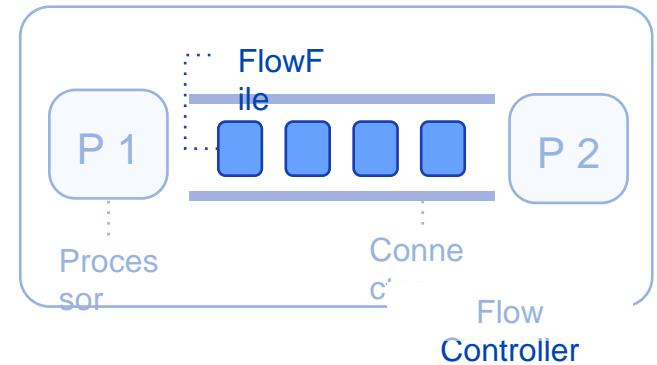
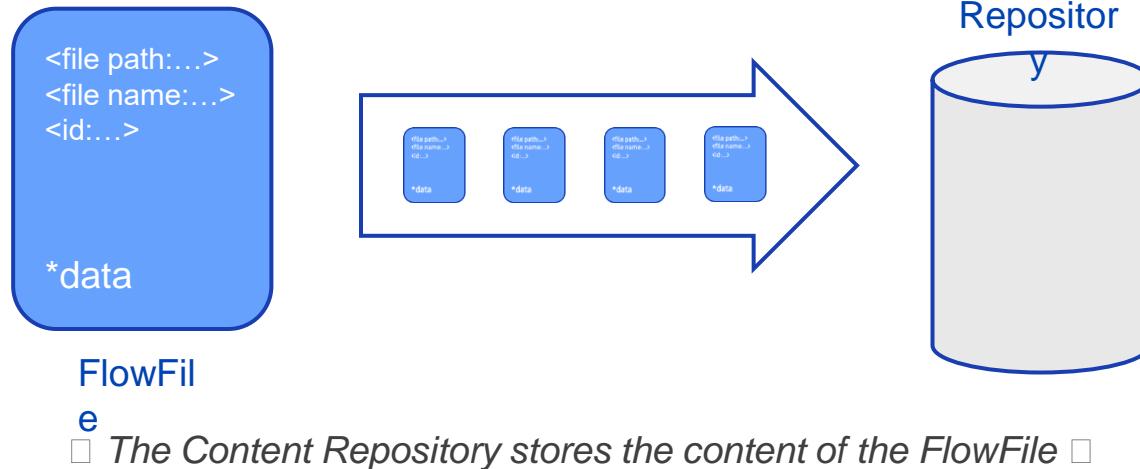
Apache NiFi Features - Extensible Architecture

- Extensions
 - Rich set of built-in processors - over 280 as of v1.12
 - Use API to extend and create custom processors
- Scalable
 - NiFi can run as a cluster
 - Increase number of NiFi workers
- Scale Up and Down
 - Increase parallelism by increasing number of concurrent threads



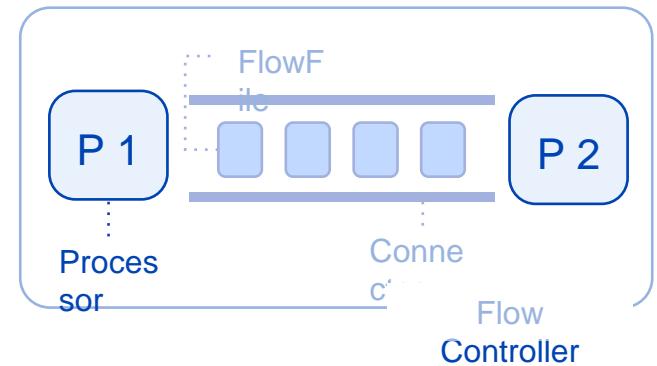
Core Concepts of NiFi (1/5)

- FlowFile
 - Represents each object moving through the system
 - Consists of key/value pair attribute strings
 - Pointer that references the actual content of the flow



Core Concepts of NiFi (2/5)

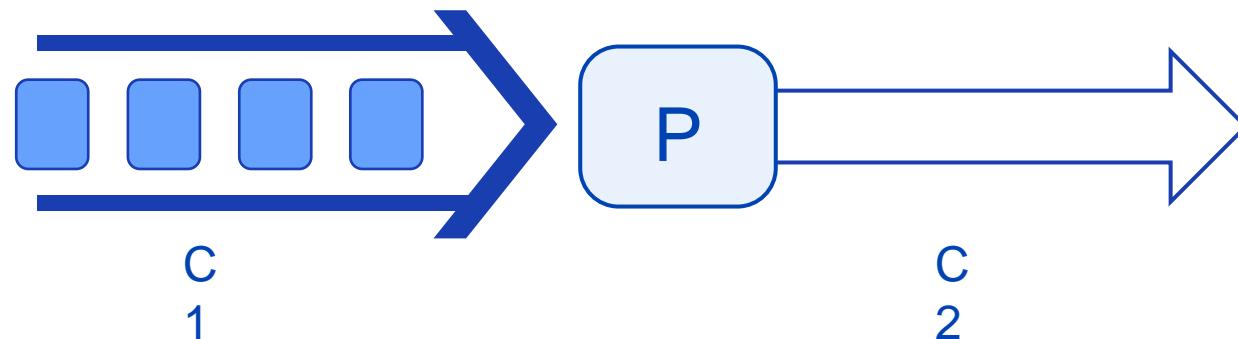
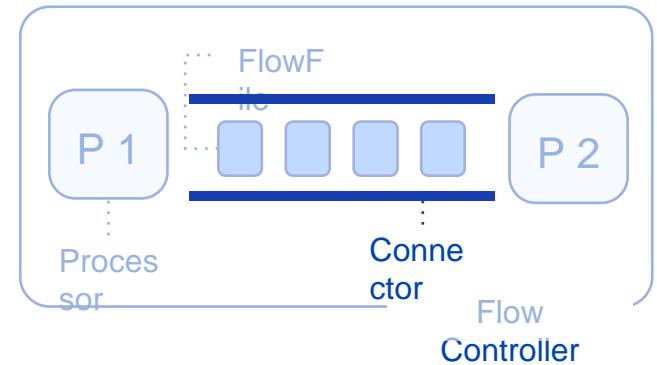
- FlowFile Processor
 - Processors do the actual work
 - Processors can perform routing, transformation and/or mediation between systems
 - Has access to the attributes of a given FlowFile.



□ Three different kinds of processors □

Core Concepts of NiFi (3/5)

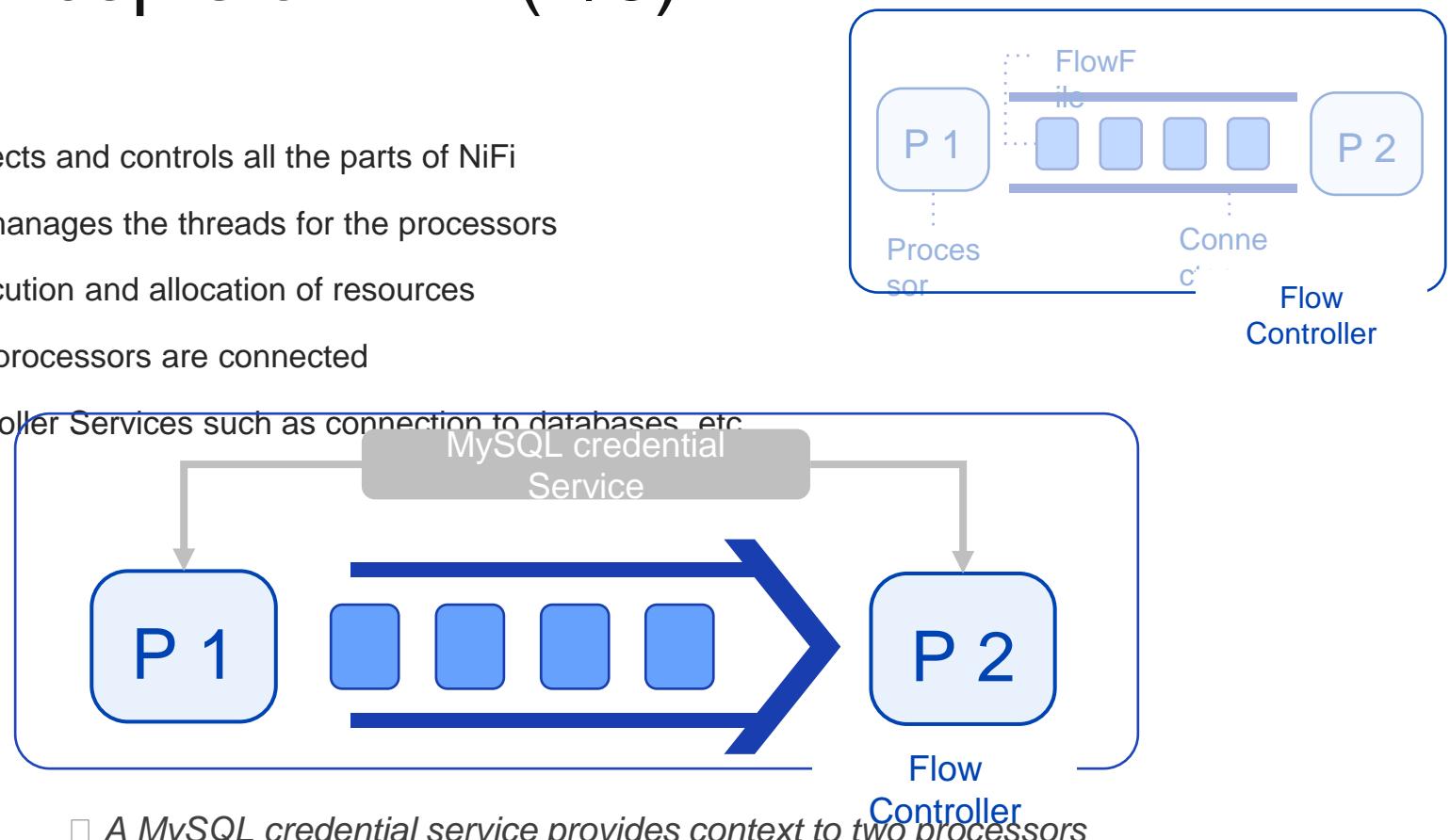
- Connection
 - Provides linkage between processors.
 - Acts as queues, allowing various processor of different rates to interoperate



□ Various capacities for different connectors. Here we have capacity $C_1 >$ capacity C_2 □

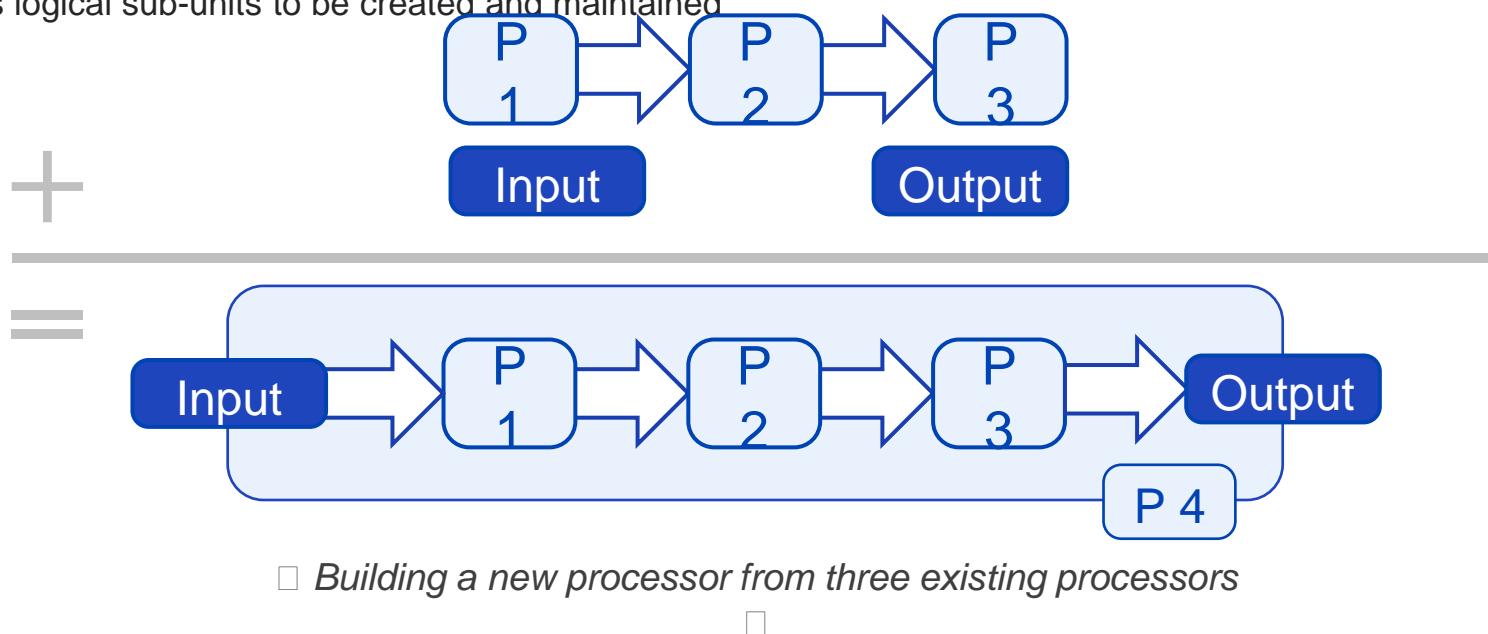
Core Concepts of NiFi (4/5)

- Flow Controller
 - Glue that connects and controls all the parts of NiFi
 - Allocates and manages the threads for the processors
 - Schedules execution and allocation of resources
 - Maintains how processors are connected
 - Manages Controller Services such as connection to databases, etc.



Core Concepts of NiFi (5/5)

- Process Group
 - A specific set of processes and their connections
 - Can receive and send data via its input and output ports
 - Allows logical sub-units to be created and maintained



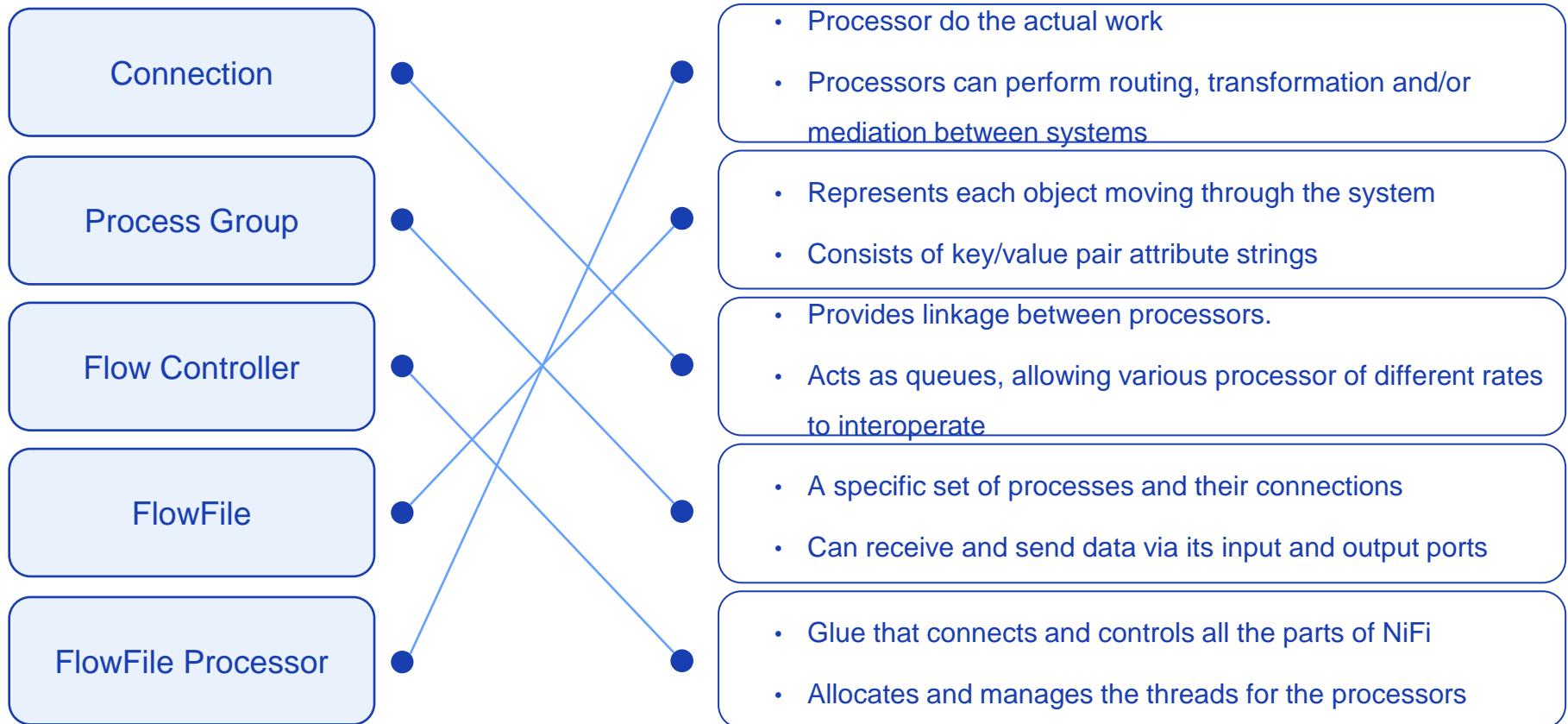
Core Concepts of NiFi

[Quiz]

- | | | |
|--------------------|---|--|
| Connection | • | <ul style="list-style-type: none">Processor do the actual workProcessors can perform routing, transformation and/or mediation between systems |
| Process Group | • | <ul style="list-style-type: none">Represents each object moving through the systemConsists of key/value pair attribute strings |
| Flow Controller | • | <ul style="list-style-type: none">Provides linkage between processors.Acts as queues, allowing various processor of different rates to interoperate |
| FlowFile | • | <ul style="list-style-type: none">A specific set of processes and their connectionsCan receive and send data via its input and output ports |
| FlowFile Processor | • | <ul style="list-style-type: none">Glue that connects and controls all the parts of NiFiAllocates and manages the threads for the processors |

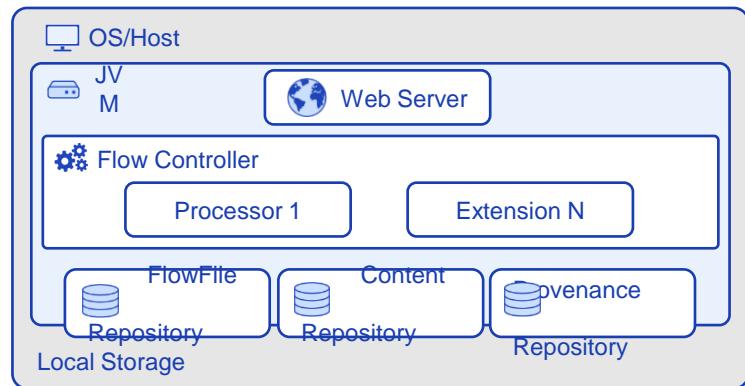
Core Concepts of NiFi

[Quiz]



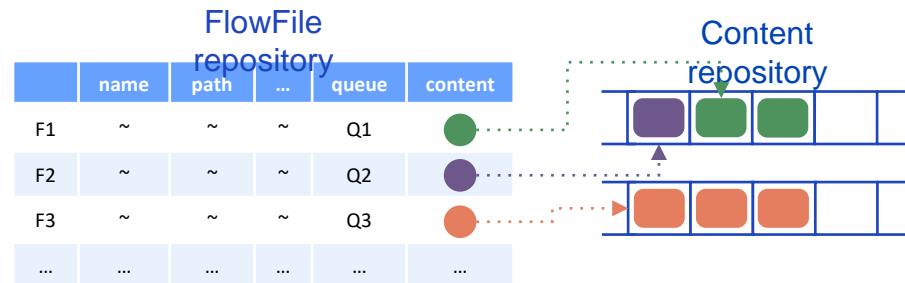
Apache NiFi Architecture (1/4)

- NiFi executes within a JVM on a host operating system
- Web Server
 - Hosts NiFi's HTTP based command and control API
- Flow Controller
 - Brains of the operation
 - Provides threads for extension to run on
 - Manages and schedules resource allocation to extensions
- Extensions
 - Processors, Controller Services, Reporting Tasks, Prioritizers along with Custom User Interfaces can be extended using the extension API
 - All built-in elements can be considered extensions in this context



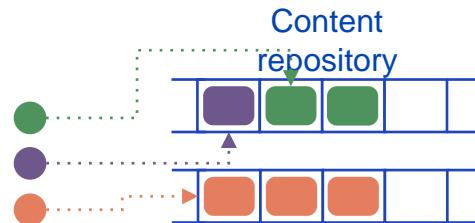
Apache NiFi Architecture (2/4)

- FlowFile Repository
 - Repository where NiFi keeps track of the state of currently active FlowFiles
 - Implementation is pluggable
 - Default approach is persistent Write-Ahead Log



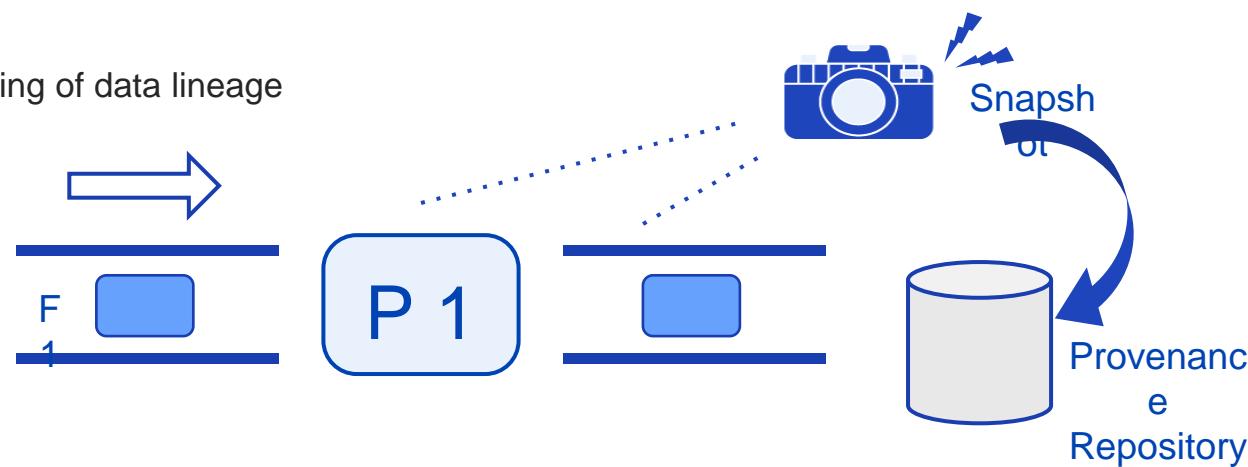
Apache NiFi Architecture (3/4)

- Content Repository
 - Actual content bytes of a FlowFile
 - Implementation is pluggable
 - Default stores blocks of data in the file system



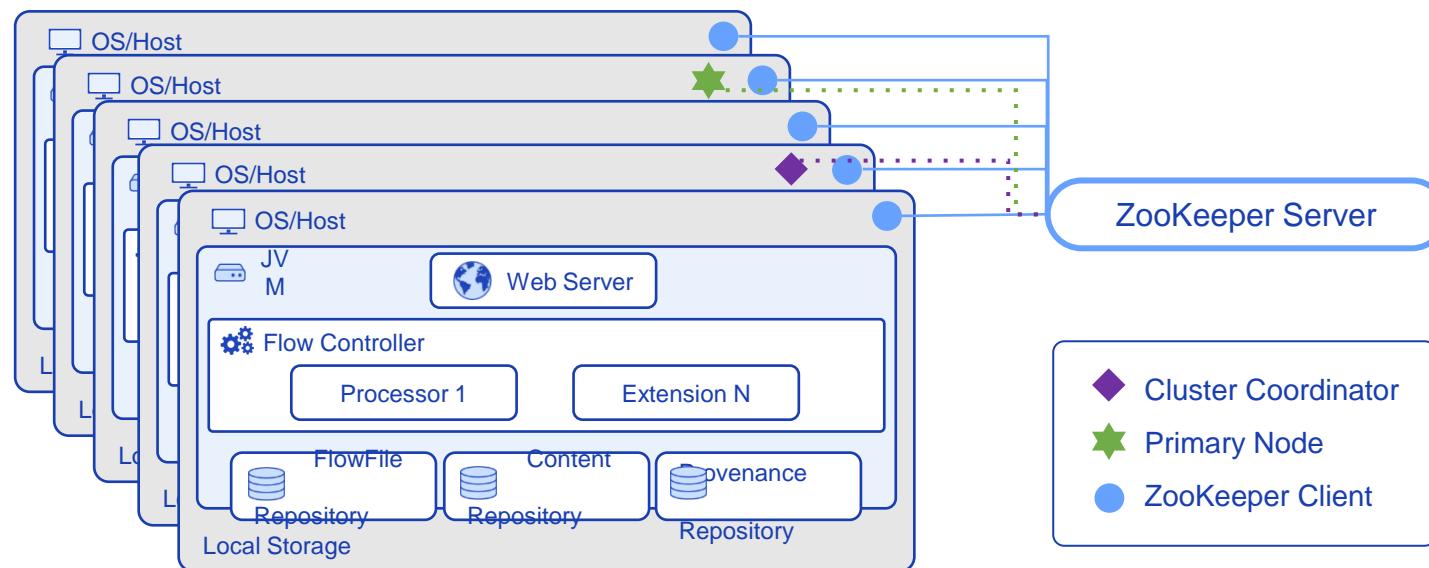
Apache NiFi Architecture (4/4)

- Provenance Repository
 - Whenever a FlowFile is modified, NiFi tracks the change
 - Takes a snapshot of the FlowFile and its context at this point
 - Provenance Repository records and stores these snapshots
 - Enables tracking of data lineage



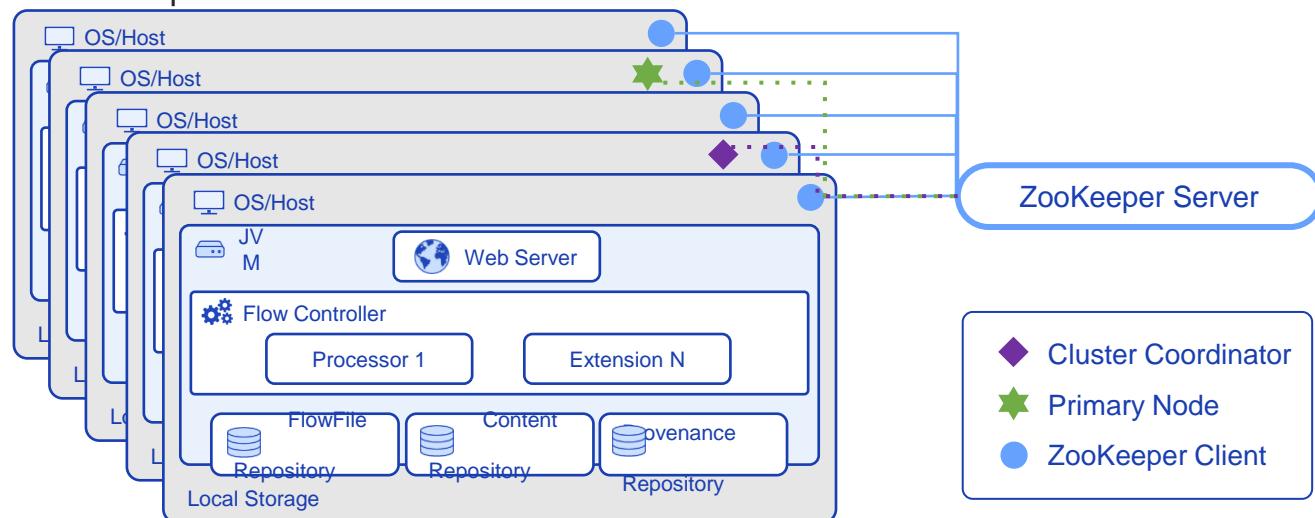
Apache NiFi on a Cluster (1/2)

- Each node in the NiFi cluster performs the same task on a different part of the data
- Apache Zookeeper elects one node as the Cluster Coordinator
 - All nodes in the cluster send heartbeats and status reports to the Cluster Coordinator
 - Responsible to disconnecting bad nodes and connecting new nodes



Apache NiFi on a Cluster (2/2)

- Users may interact with any node to create, modify and manage dataflows
 - Any change is replicated to all nodes automatically
- Apache Zookeeper elects one node as the Primary Node
 - Some processors require a single node to process
 - Primary node handles such requests



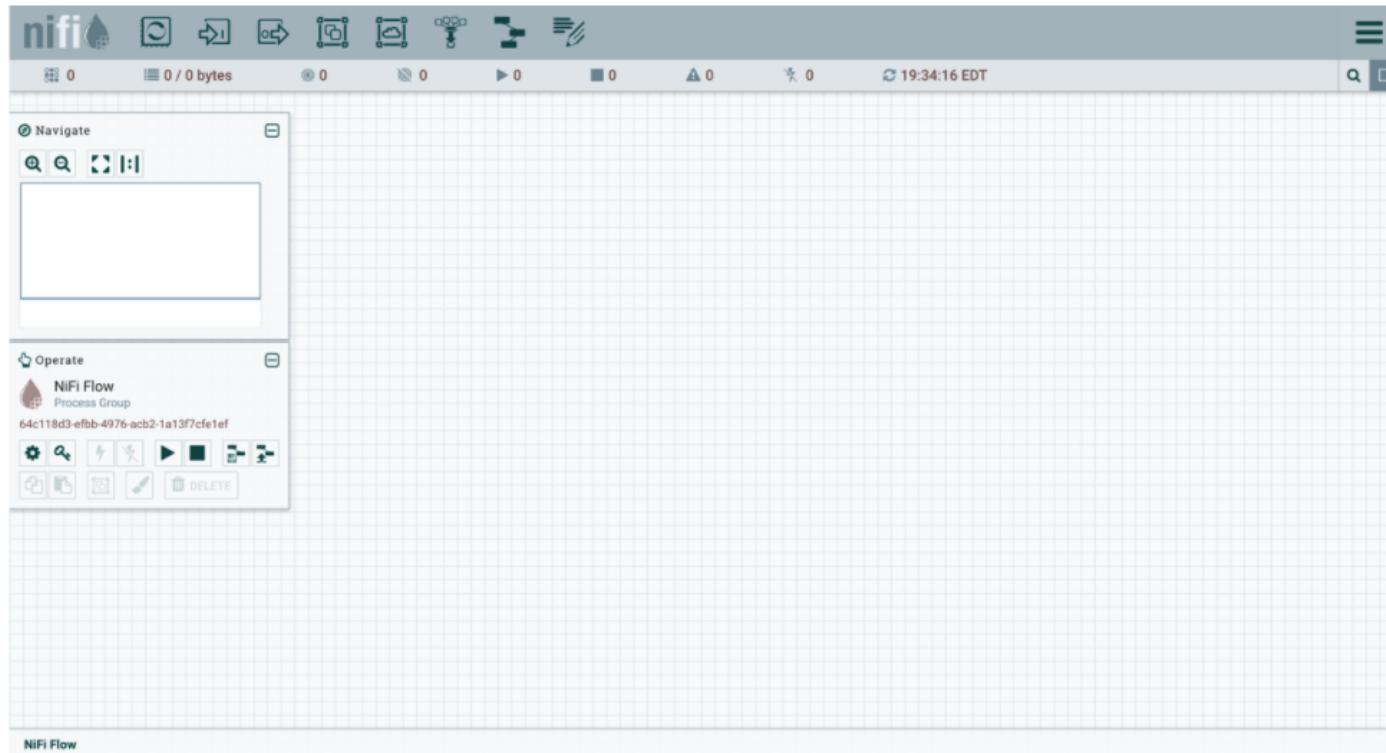
Unit 3

Creating Data Pipelines with Apache NiFi

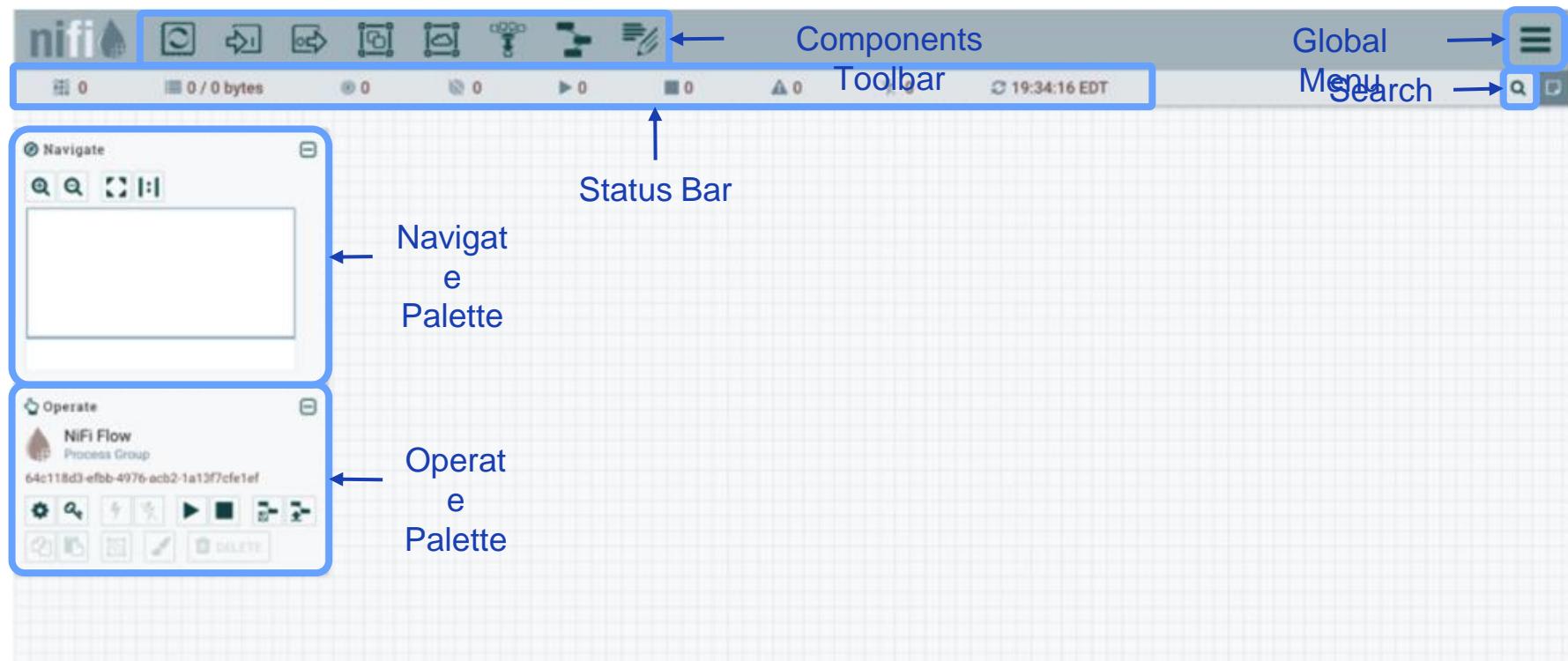
- | 3.1. Apache NiFi Fundamentals
- | 3.2. Apache NiFi User Interface
- | 3.3. Apache NiFi Processors
- | 3.4. Apache NiFi Connections
- | 3.5. Creating Data Flows
- | 3.6. Process Group

Apache NiFi Data Flow Canvas (1/2)

- Access the NiFi user interface from port 8080 on any node in the NiFi Cluster

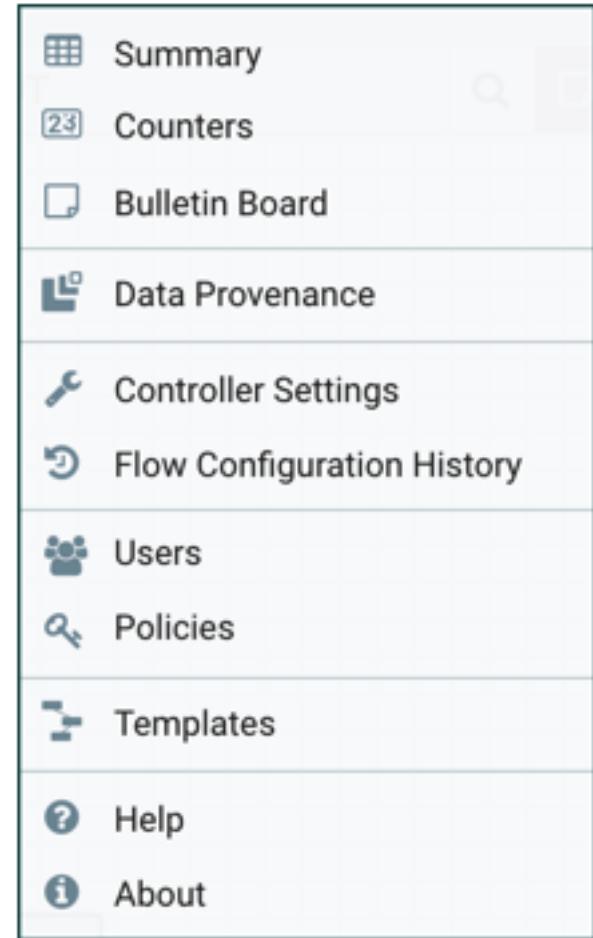


Apache NiFi Data Flow Canvas (2/2)



NiFi Canvas - Global Menu

- Provides menu to access global status of data flows
- Review data lineage through Provenance reports
- Manage and access Controller Settings
- Manage users and create security policies
- Manage and review data flow Templates
- Access NiFi documentation



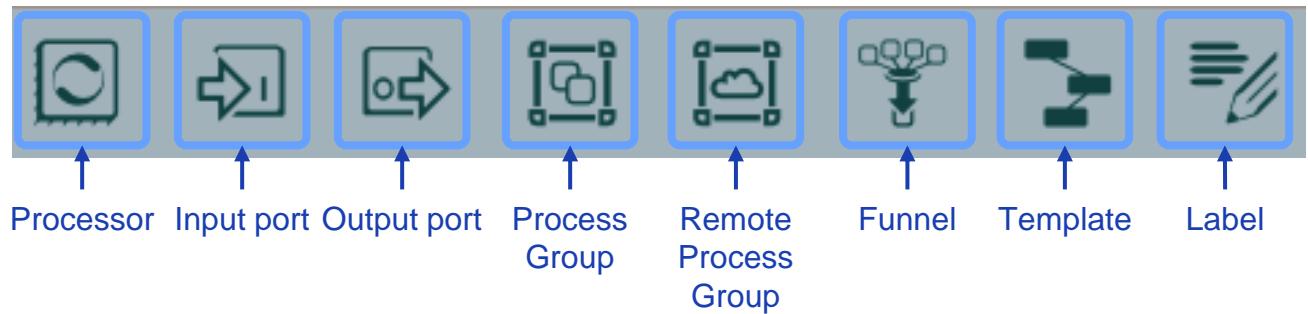
NiFi Canvas - Search

- Quickly find components by name, type, property values, etc.

The screenshot shows the Apache NiFi User Interface with a search bar at the top right containing the text "generate". Below the search bar, the results are displayed under the heading "Processors". The first result is "Attr Generate Small Files", which is listed as a "Parent: FlowFile Attributes Lab" and is "Versioned: -". It has two properties: "Name: Attr Generate Small Files" and "Type: GenerateFlowFile". The "Property description" for this component states: "If true, each FlowFile that is generated will be unique. If false, a random value will be generated and all FlowFiles will get the same content but this offers much higher throughput". Another property, "Property name: generate-ff-custom-text", is also listed with its "Property description": "If Data Format is text and if Unique FlowFiles is false, then this custom text will be used as content of the generated FlowFiles and the File Size will be ignored. Finally, if Expression Language is used, evaluation will be performed only once per batch of generated FlowFiles".

NiFi Canvas - Components Toolbar

- Click and drag components to the canvas to instantiate a new component
 - Processor
 - Input Port
 - Output Port
 - Process Group
 - Remote Process Group
 - Funnel
 - Template
 - Label

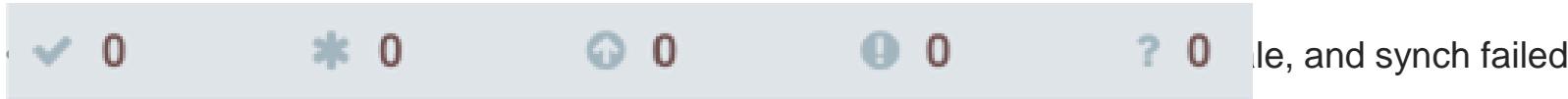
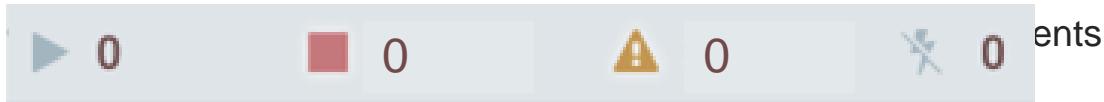
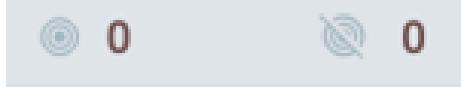


NiFi Canvas - Status Bar

- Number of active threads and Total queued data

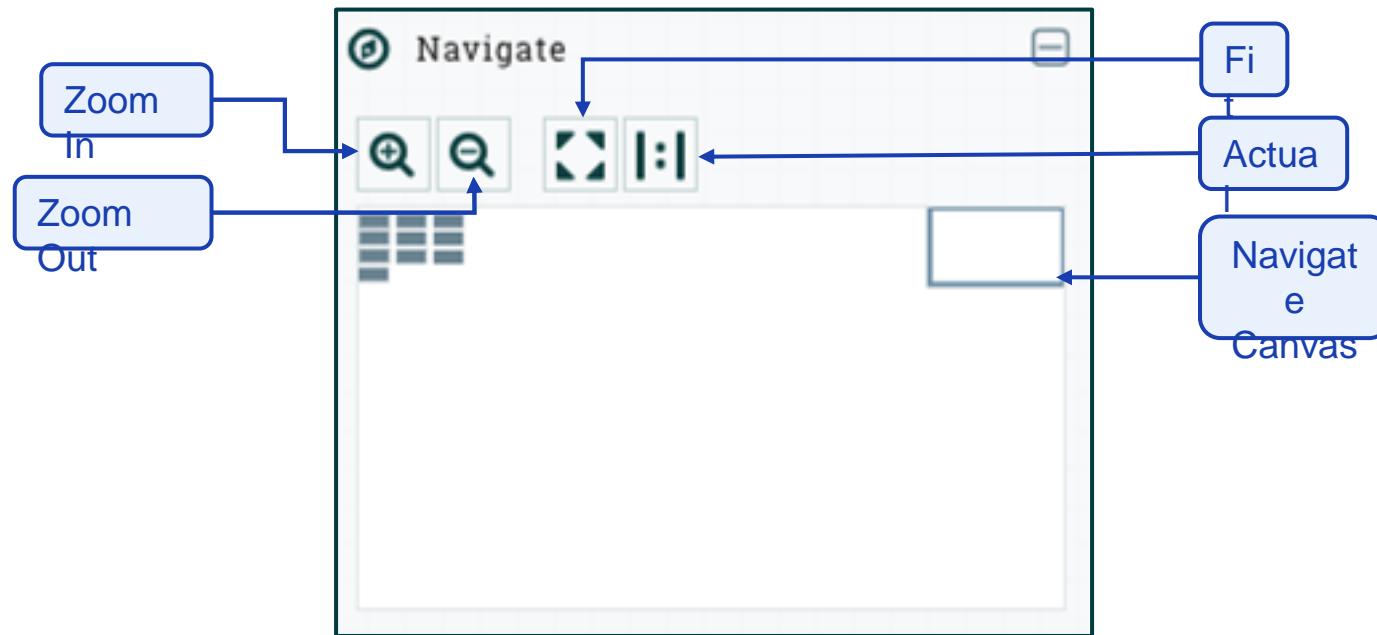


- Transmitting and Not transmitting Remote Processor Groups



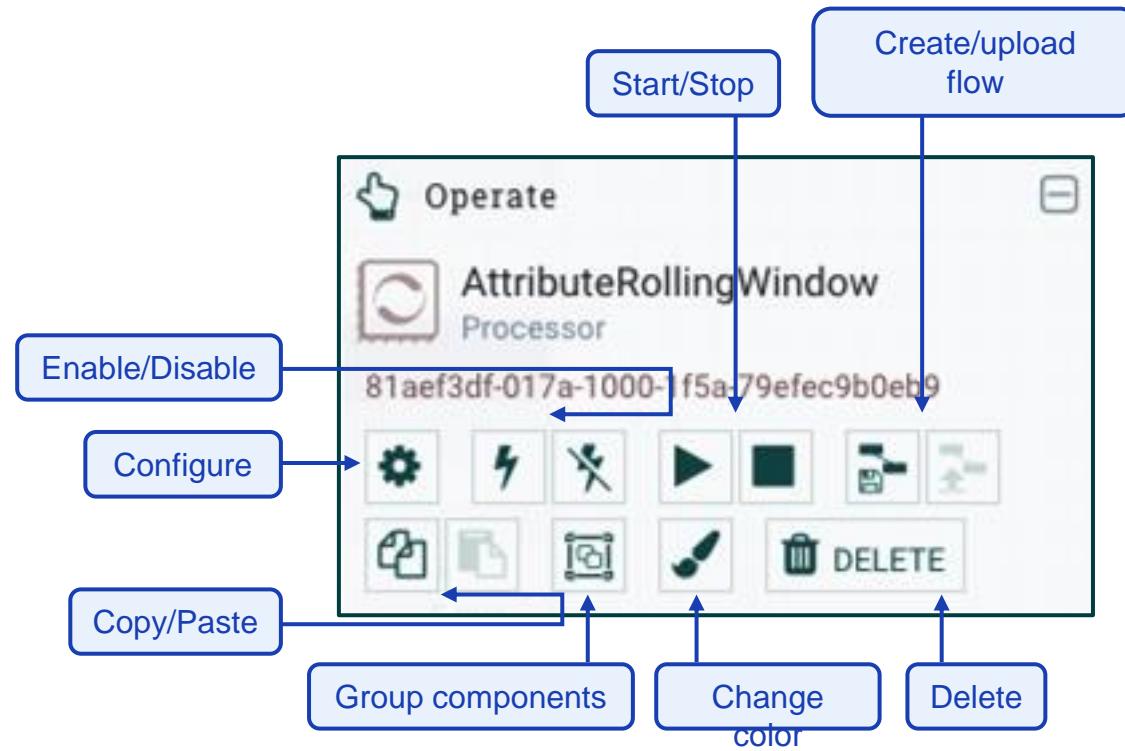
NiFi Canvas - Navigate Palette

- Navigation Palette Anatomy



NiFi Canvas - Operate Palette

- Operation Palette Anatomy



[Lab3]

Making your first dataflow



Unit 3

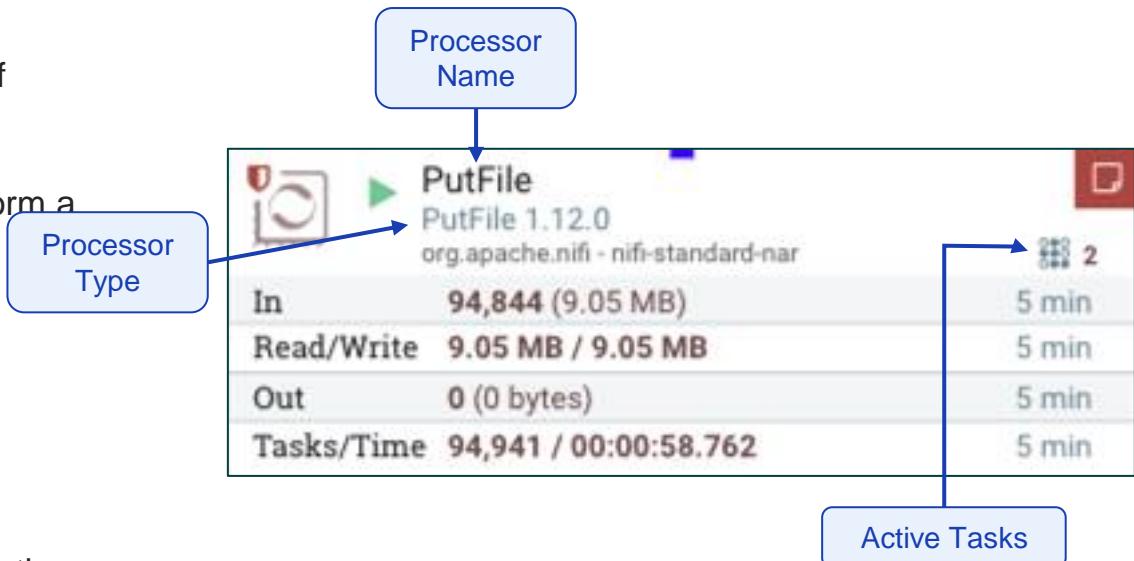
Creating Data Pipelines with Apache NiFi

- | 3.1. Apache NiFi Fundamentals
- | 3.2. Apache NiFi User Interface
- | 3.3. Apache NiFi Processors

- | 3.4. Apache NiFi Connections
- | 3.5. Creating Data Flows
- | 3.6. Process Group

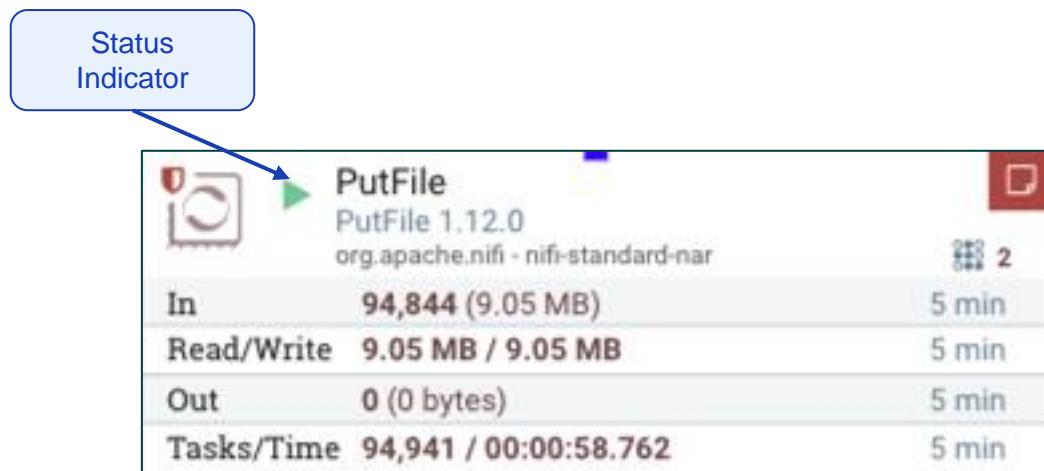
Anatomy of a Processor (1/4)

- Processor Type
 - NiFi provides a wide range of Processors of various categories
 - Each type of Processor is designed to perform a single specific task
- Processor Name
 - The user-defined name of the Processor
- Active Tasks
 - Number of task Processor is currently executing
 - Each task represents a single thread working on a single FlowFile
 - Configured and constrained by "Concurrent Tasks" and setting



Anatomy of a Processor (2/4)

- Status Indicator
 - Shows the current Status of the Processor.
- The following indicators are possible
 - Processor is currently running
 - Processor is valid but not running
 - Processor is enabled but not currently valid.
There are setting that must be fixed before it can run
 - Processor is not running and cannot start until it has been enabled



PutFile		
	PutFile 1.12.0	
org.apache.nifi - nifi-standard-nar		
In	94,844 (9.05 MB)	5 min
Read/Write	9.05 MB / 9.05 MB	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	94,941 / 00:00:58.762	5 min

Anatomy of a Processor (3/4)

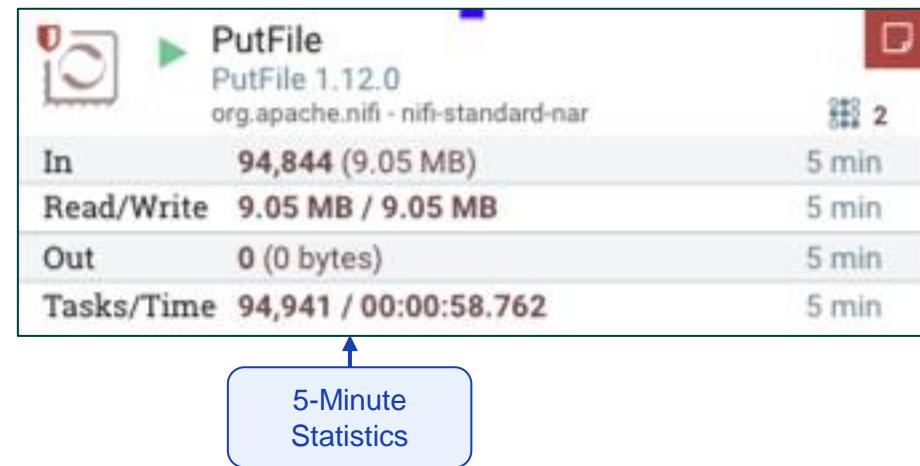
- Bulletin Indicator
 - When Processor logs some event, it generates a bulletin in the UI
 - Users can configure which bulletins should be displayed by updating the "Bulletin Level" configuration setting
 - Default is WARN
 - Available levels are DEBUG, INFO, WARN, ERROR and NONE
 - Icon is present only if there is a bulletin present
 - Hover mouse over icon to view the bulletin message



PutFile		PutFile 1.12.0	org.apache.nifi - nifi-standard-nar	2
In	94,844 (9.05 MB)			5 min
Read/Write	9.05 MB / 9.05 MB			5 min
Out	0 (0 bytes)			5 min
Tasks/Time	94,941 / 00:00:58.762			5 min

Anatomy of a Processor (4/4)

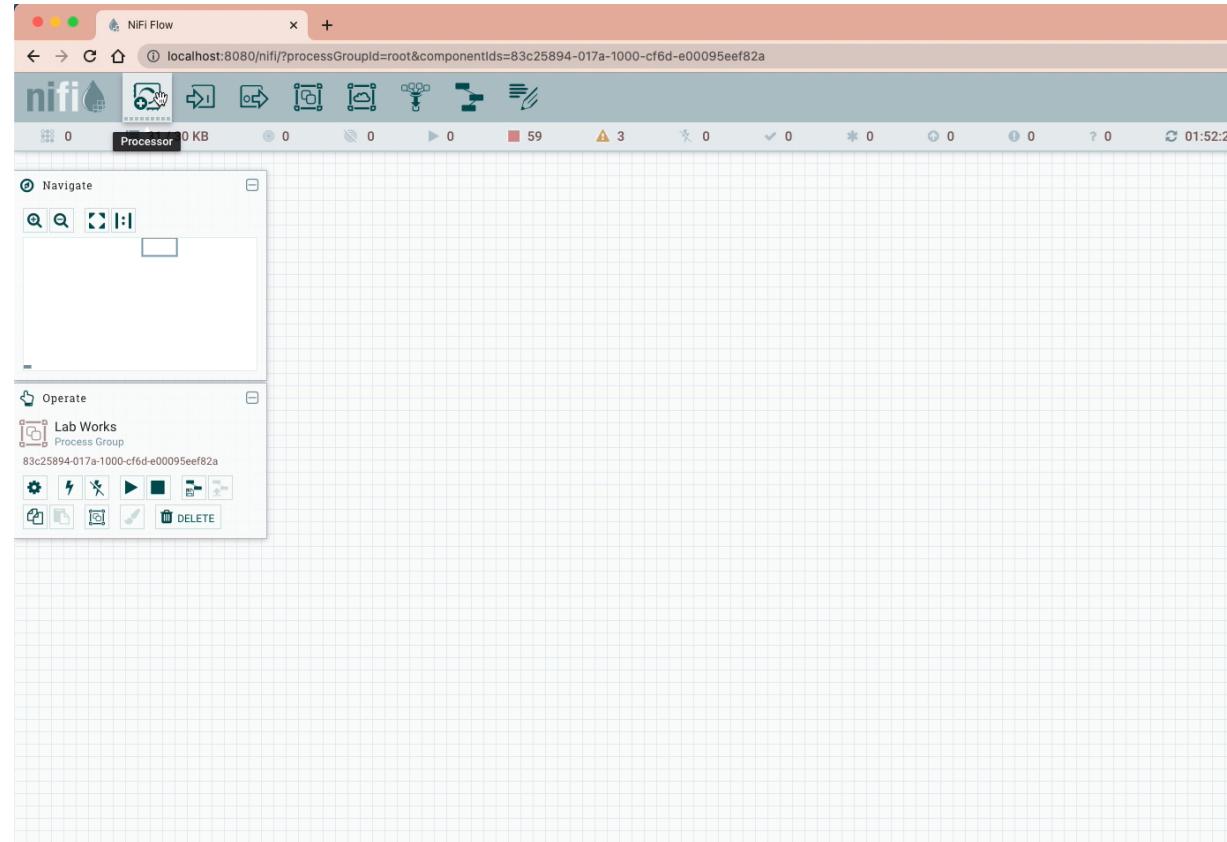
- 5 min Statistics
 - I/O and compute statistics over last 5 mins
- In
 - Count and Size of FlowFiles pulled from queues of the incoming Connection
- Read/Write
 - Total size of FlowFile content that Processor has read and written to disk
- Out
 - Count and Size of FlowFiles sent to its outbound connections
- Tasks/Time
 - Number of times triggered for tasks and amount of time taken



Adding a Processor to the Canvas

[Video]

- Processor do the actual work
- Processors can perform routing, transformation and/or mediation between systems
- Has access to the attributes of a FlowFile.



3.3. Apache NiFi Processors

- Filter by category
- Filter by Name

Selecting a Processor

Add Processor

Source: all groups ▾

Displaying 288 of 288

Type	Version	Tags
AttributeRollingWindow	1.12.0	rolling, data science, Attribute ...
AttributesToCSV	1.12.0	flowfile, csv, attributes
AttributesToJson	1.12.0	flowfile, json, attributes
Base64EncodeContent	1.12.0	encode, base64
CalculateRecordStats	1.12.0	stats, record, metrics
CaptureChangeMySQL	1.12.0	cdc, jdbc, mysql, sql
CompareFuzzyHash	1.12.0	fuzzy-hashing, hashing, cyber...
CompressContent	1.12.0	lzma, deflate, decompress, co...
ConnectWebSocket	1.12.0	subscribe, consume, listen, We...
ConsumeAMQP	1.12.0	receive, amqp, rabbit, get, cons...
ConsumeAzureEventHub	1.12.0	cloud, streaming, streams, eve...
ConsumeEWS	1.12.0	EWS, Exchange, Email, Consu...

Filter:

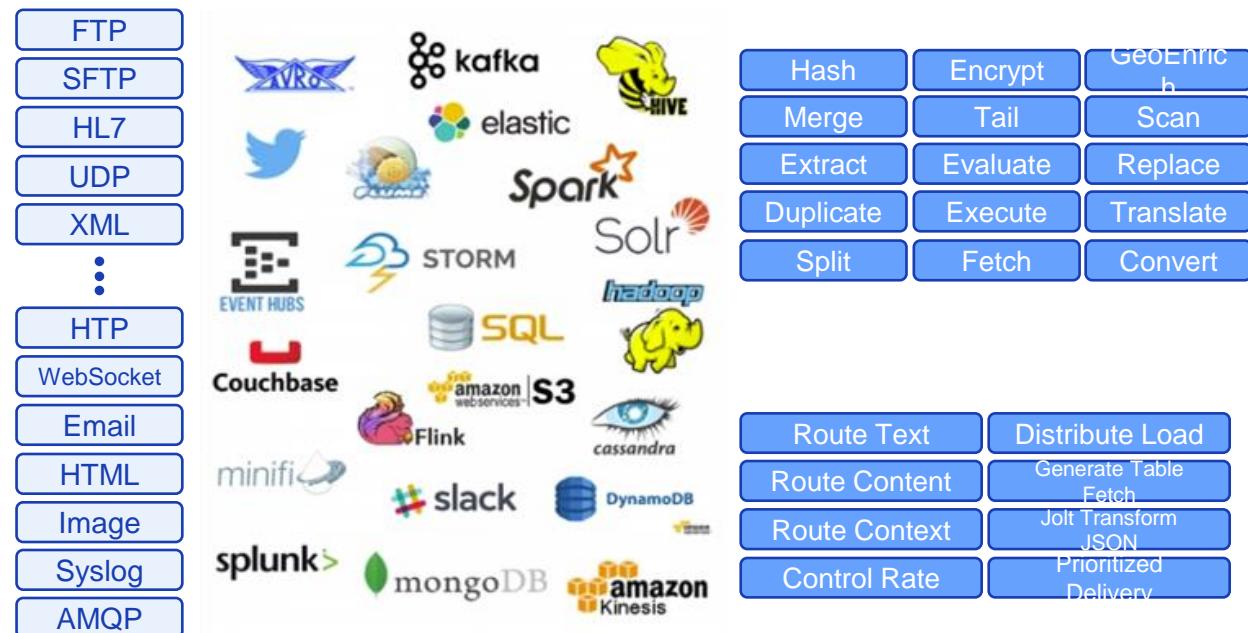
AttributeRollingWindow 1.12.0 org.apache.nifi - nifi-stateful-analysis-nar

Track a Rolling Window based on evaluating an Expression Language expression on each FlowFile and add that value to the processor's state. Each FlowFile will be emitted with the count of FlowFiles and total aggregate value of values processed in the current time window.

CANCEL ADD

Types of Processors

- Data Transformation
- Routing and Mediation
- Database Access
- Attribute Extraction
- System Interaction
- Data Ingestion
- Data Egress / Sending Data
- Splitting and Aggregation
- HTTP
- Amazon Web Services



Data Transformation

- Processors for Data Transformation

Processor	Description
CompressContent	Compress or Decompress Content
ConvertCharacterSet	Convert the character set used to encode the content from one character set to another
EncryptContent	Encrypt or Decrypt Content
ReplaceText	Use Regular Expressions to modify textual Content
TransformXml	Apply an XSLT transform to XML Content

Routing and Mediation

- Processors for Routing and Mediation

Processor	Description
ControlRate	Throttle the rate at which data can flow through one part of the flow
DetectDuplicate	Monitor for duplicate FlowFiles, based on some user-defined criteria
DistributeLoad	Load balance or sample data
MonitorActivity	Sends a notification of period of no activity or resumption of activity
RouteOnAttribute	Route a FlowFile based on an attribute
RouteOnContent	Search Content of a FlowFile and route to matching Relationship
ScanContent	Search Content of a FlowFile and match against user-defined dictionary. If Content contains a match, route to desired Relationship.

Database Access

- Processors for Database Access

Processor	Description
ConvertJSONToSQL	Convert a JSON document into a SQL INSERT or UPDATE command that can then be passed to the PutSQL Processor
ExecuteSQL	Executes a user-defined SQL SELECT command, writing the results to a FlowFile
PutSQL	Updates a database by executing the SQL DDM statement defined by the FlowFile's content
SelectHiveQL	Executes a user-defined HiveQL SELECT command against an Apache Hive database, writing the results to a FlowFile
PutHiveQL	Updates a Hive database by executing the HiveQL DDM statement defined by the FlowFile's content

Attribute Extraction

- Processors for Attributes Extraction

Processor	Description
HashAttribute	Performs a hashing function against the concatenation of a user-defined list of existing Attributes
HashContent	Performs a hashing function against the content of a FlowFile and adds the hash value as an Attribute
IdentifyMimeType	Evaluates the content of a FlowFile and detect MIME Types, such as images, word processor documents, text, and compression formats
UpdateAttribute	Adds or updates any number of user-defined Attributes to a FlowFile. Attributes can be static values, dynamic values, or even conditionally added or updated based on user-defined rules
ExtractText	User supplies one or more Regular Expressions that are then evaluated against the textual content of the FlowFile, and the values that are extracted are then added as user-named Attributes

System Interaction

- Processors for System Interaction

Processor	Description
ExecuteProcess	Runs a user-defined Operating System command. The StdOut of the command becomes the content of the outbound FlowFile
ExecuteStreamCommand	Runs a user-defined Operating System command. The contents of the incoming FlowFile are optionally streamed to the StdIn of the process. The StdOut becomes the content of the outbound FlowFile.

Data Ingestion (1/3)

- Processors for Data Ingestion

Processor	Description
GetFile	Streams the contents of a file from a local disk into NiFi and then deletes the original file
GetFTP	Downloads the contents of a remote file via FTP into NiFi and then deletes the original file
GetHTTP	Downloads the contents of a remote HTTP- or HTTPS-based URL into NiFi
ListenHTTP	Starts an HTTP (or HTTPS) Server and listens for incoming connections. For any incoming POST request, the contents of the request are written out as a FlowFile, and a 200 response is returned
ListenUDP	Listens for incoming UDP packets and creates a FlowFile per packet or per bundle of packets (depending on configuration) and emits the FlowFile to the 'success' relationship
GetHDFS	Monitors a user-specified directory in HDFS. Whenever a new file enters HDFS, it is copied into NiFi and deleted from HDFS

Data Ingestion (2/3)

- Processors for Data Ingestion

Processor	Description
ListHDFS	ListHDFS monitors a user-specified directory in HDFS and emits a FlowFile containing the filename for each file that it encounters
FetchHDFS	Receive a list of files from ListHDFS and fetch the actual content. Files are not deleted
FetchS3Object	Fetches the contents of an object from the Amazon Web Services (AWS) Simple Storage Service (S3). The outbound FlowFile contains the contents received from S3
GetKafka	Fetches messages from Apache Kafka and emit as outgoing FlowFile
GetMongo	Executes a user-specified query against MongoDB and writes the contents to a new FlowFile
GetTwitter	Register and listen to Twitter, creating a FlowFile for each tweet that is received

Data Egress / Sending Data

- Processors for Data Egress / Sending Data

Processor	Description
PutEmail	Sends an E-mail to the configured recipients. The content of the FlowFile is optionally sent as an attachment
PutFile	Writes the contents of a FlowFile to a directory on the local file system
PutFTP	Copies the contents of a FlowFile to a remote FTP Server
PutSQL	Executes the contents of a FlowFile as a SQL DDL Statement (INSERT, UPDATE, or DELETE). The contents of the FlowFile must be a valid SQL statement
PutKafka	Sends the contents of a FlowFile as a message to Apache Kafka
PutMongo	Sends the contents of a FlowFile to Mongo as an INSERT or an UPDATE

Splitting and Aggregation

- Processors for Splitting and Aggregation

Processor	Description
SplitText	SplitText takes in a single text FlowFile and splits it into 1 or more FlowFiles
SplitJson	Allows the user to split a JSON array objects
SplitXml	Allows the user to split an XML message into many FlowFiles
UnpackContent	Unpacks different types of archive formats, such as ZIP and TAR

HTTP

- Processors for HTTP

Processor	Description
GetHTTP	Downloads the contents of a remote HTTP- or HTTPS-based URL into NiFi
InvokeHTTP	Performs an HTTP Request that is configured by the user. This Processor is much more versatile than the GetHTTP and PostHTTP but requires a bit more configuration. This Processor cannot be used as a Source Processor and is required to have incoming FlowFiles in order to be triggered to perform its task
PostHTTP	Performs an HTTP POST request, sending the contents of the FlowFile as the body of the message. This is often used in conjunction with ListenHTTP in order to transfer data between two different instances of NiFi in cases where Site-to-Site cannot be used

Amazon Web Services

- Processors for Amazon Web Service

Processor	Description
FetchS3Object	Fetches the content of an object stored in Amazon Simple Storage Service (S3). The content that is retrieved from S3 is then written as the content of the outgoing FlowFile
PutS3Object	Writes the contents of a FlowFile to an Amazon S3 object using the configured credentials, key, and bucket name
PutSNS	Sends the contents of a FlowFile as a notification to the Amazon Simple Notification Service (SNS)
GetSQS	Pulls a message from the Amazon Simple Queuing Service (SQS) and writes the contents of the message to the content of the FlowFile
PutSQS	Sends the contents of a FlowFile as a message to the Amazon Simple Queuing Service (SQS)
DeleteSQS	Deletes a message from the Amazon Simple Queuing Service (SQS)

Unit 3

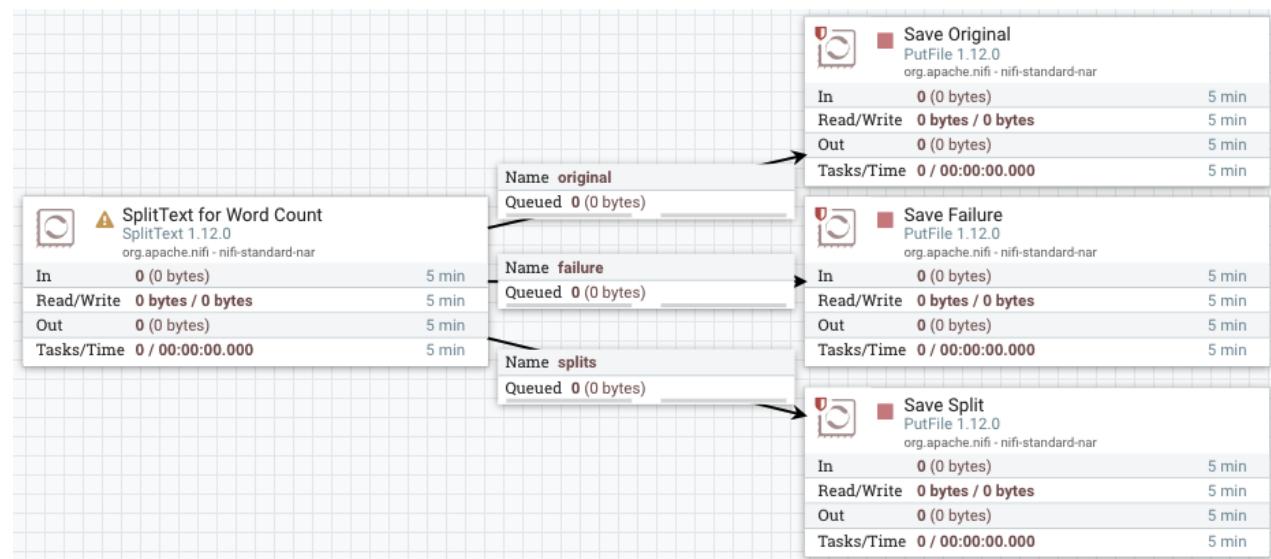
Creating Data Pipelines with Apache NiFi

- | 3.1. Apache NiFi Fundamentals
- | 3.2. Apache NiFi User Interface
- | 3.3. Apache NiFi Processors

- | 3.4. Apache NiFi Connections
- | 3.5. Creating Data Flows
- | 3.6. Process Group

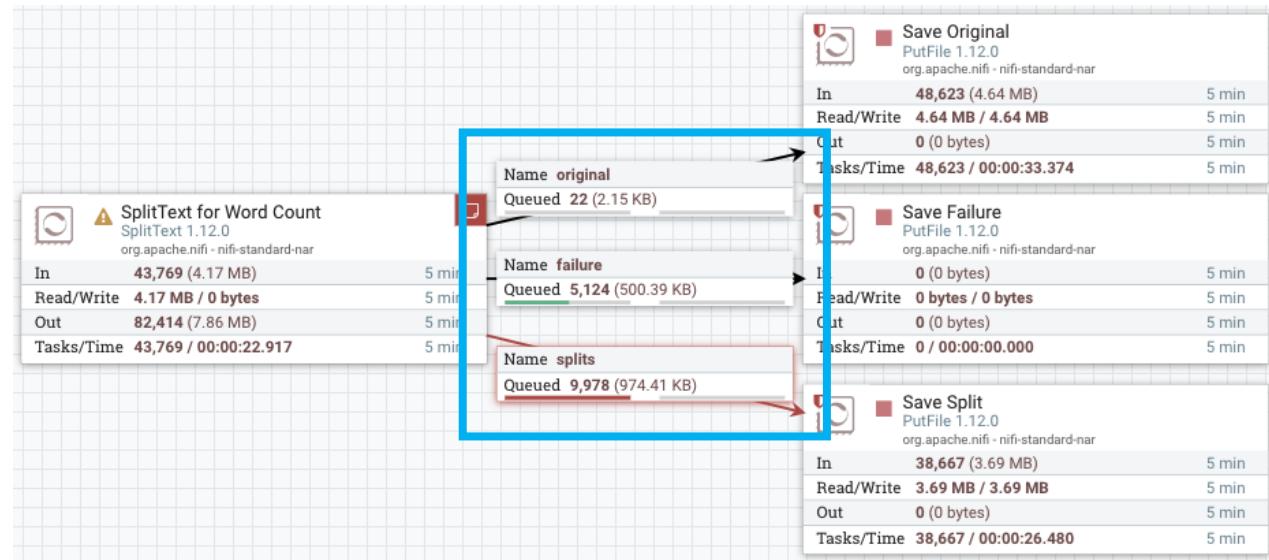
Connections and Relationships

- Processors can have zero or more Relationships defined
 - Named to indicate the result of processing a FlowFile
 - Ex: success, failure, not.found, permission.denied, etc
- Connections allow FlowFiles to travel from one Processor to the next Processor downstream
 - Each connection consists of one or more relationships
 - Allows data to be routed to different Processors depending on the processing outcome



Buffered Connections

- Connections acts as buffers and queue the incoming FlowFiles until processed by the downstream processor



Buffer Queue Detail

- Right click on a connection to bring up the context menu
 - Select List Queue to get a list of FlowFiles currently queued in the buffer
 - View or download the contents of the FlowFile
 - Review FlowFile history ad lineage with the Provenance option

Name original
Queued 22 (2.15 KB)

Name failure
Queued 5,124 (500.39 KB)

Name splits
Queued 9,978 (974.41 KB)

Configure
View status history
List queue
Go to source
Go to destination
Bring to front
Create template
Empty queue
Delete

Tasks/Time 0 / 00:00:00.0

original

Displaying 22 of 22 (2.15 KB)

Position	UUID	Filename	File Size	Queued Duration	Lineage Duration	Penalized
1	9655b31d-aear-4...	9655b31d-aear-4...	100.00 bytes	00:10:41.506	00:10:49.347	No
2	e2bc8672-202b-4...	e2bc8672-202b-4...	100.00 bytes	00:10:41.506	00:10:49.346	No
3	0e5d4d80-978c-4...	0e5d4d80-978c-4...	100.00 bytes	00:10:41.505	00:10:49.346	No
4	83efd081-4248-4...	83efd081-4248-4...	100.00 bytes	00:10:41.505	00:10:49.346	No
5	544e0043-169b-4...	544e0043-169b-4...	100.00 bytes	00:10:41.495	00:10:49.345	No
6	76dc352-719a-4...	76dc352-719a-4...	100.00 bytes	00:10:41.494	00:10:49.345	No
7	583fb894-7caa-4...	583fb894-7caa-4...	100.00 bytes	00:10:41.493	00:10:49.345	No

FlowFile Details

- View detailed information on the FlowFile
- All FlowFiles include common Attributes such as filename, uuid, path, etc

FlowFile

DETAILS	ATTRIBUTES
FlowFile Details	Content Claim
UUID 9655b31d-aear-4c2e-ba3a-0bf9a5a00784	Container default
Filename 9655b31d-aear-4c2e-ba3a-0bf9a5a00784	Section 10
File Size 100 bytes	Identifier 1625736467520-10
Queue Position No value set	Offset 534100
Queued Duration 00:11:33.074	Size 100 bytes
Lineage Duration 00:11:40.915	DOWNLOAD VIEW
Penalized No	

OK

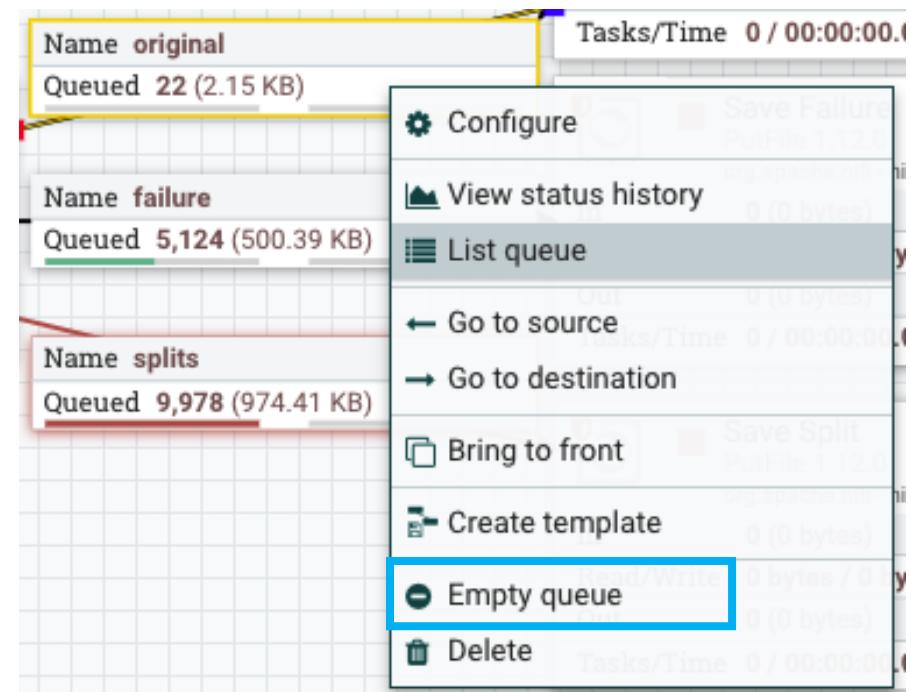
FlowFile

DETAILS	ATTRIBUTES
	Attribute Values
	filename 9655b31d-aear-4c2e-ba3a-0bf9a5a00784
	fragment.count 1
	fragment.identifier 32894c0e-4a85-422c-8792-b1bcff1bc403
	path ./
	uuid 9655b31d-aear-4c2e-ba3a-0bf9a5a00784

OK

Emptying the Queue

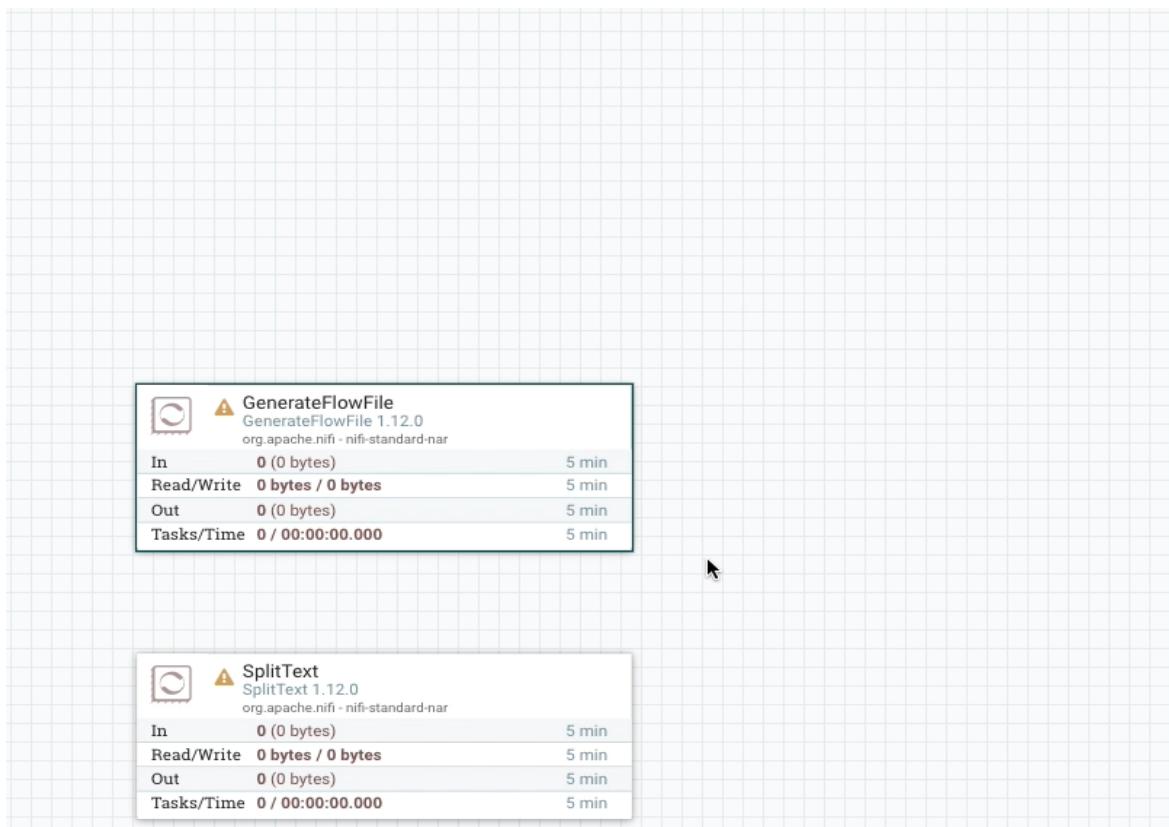
- How to empty the Connection queue
 - Right click on the connection to bring up the context menu
 - Select the Empty Queue option
 - Processors can only be deleted if they are not connected to a Connection
 - Connections can only be deleted if the queue is empty



Connecting Components

[Video]

- Connect two Processor with Connections
- Hover mouse over center of a Proc  or until the Connection icon appears
- Click and drag Connection icon to the downstream Processor



Configure Connection Details Tab

- Provides information about the source and destination components
- Select the Relationships for which the connection which transport FlowFiles
- Multiple Connections can exist for the same Relationship
 - FlowFiles will be cloned and forwarded to each Connection



Configure Connection Settings Tab (1/4)

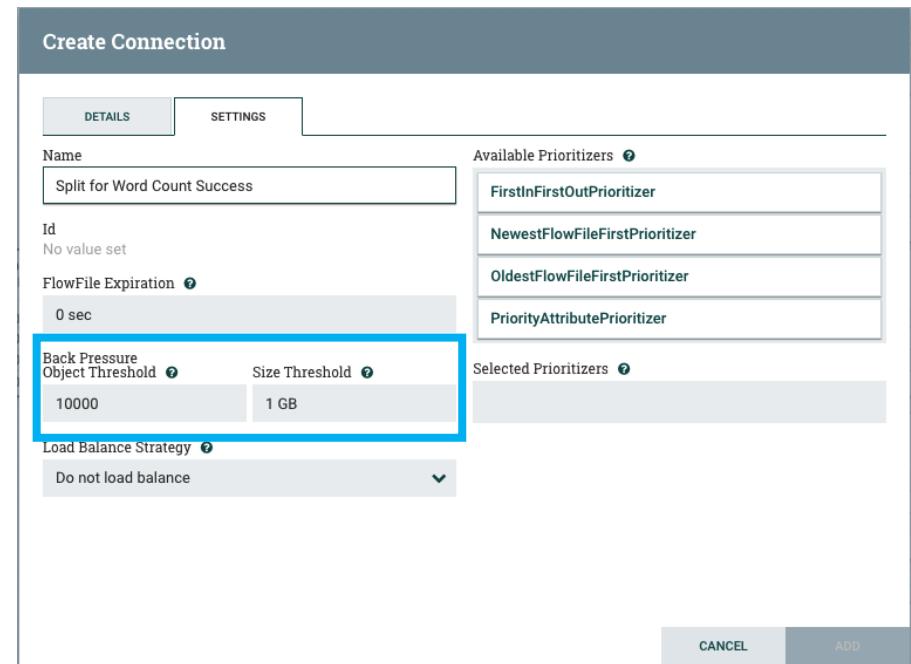
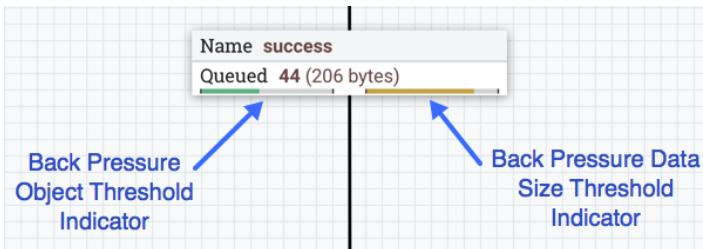
- Name of Connection
- FlowFile Expiration
 - Amount of time that a FlowFile can remain in the queue before it is deleted
 - If set, the connection will have an expiration indicator



The screenshot shows the "Create Connection" dialog box. The "SETTINGS" tab is selected. The "FlowFile Expiration" field is highlighted with a blue box and contains "0 sec". Other settings include "Back Pressure Object Threshold" at 10000, "Size Threshold" at 1 GB, and "Load Balance Strategy" set to "Do not load balance". Available prioritizers listed on the right are FirstInFirstOutPrioritizer, NewestFlowFileFirstPrioritizer, OldestFlowFileFirstPrioritizer, and PriorityAttributePrioritizer.

Configure Connection Settings Tab (2/4)

- Back Pressure Object Threshold
 - Maximum number of FlowFiles in queue before back pressure is triggered
- Back Pressure Size Threshold
 - Maximum total size of all the FlowFiles in the queue before back pressure is triggered



Configure Connection Settings Tab (3/4)

- Load may be balanced by nodes in the NiFi cluster
 - By default, do not load balance
 - Partition by a particular attribute and send all FlowFiles with the same attribute to a single node
 - Distribute the FlowFiles by round robin
 - Send all FlowFiles to a single node

Load Balance Strategy ?

Attribute Name ?

- Compl Partition by attribute ▼

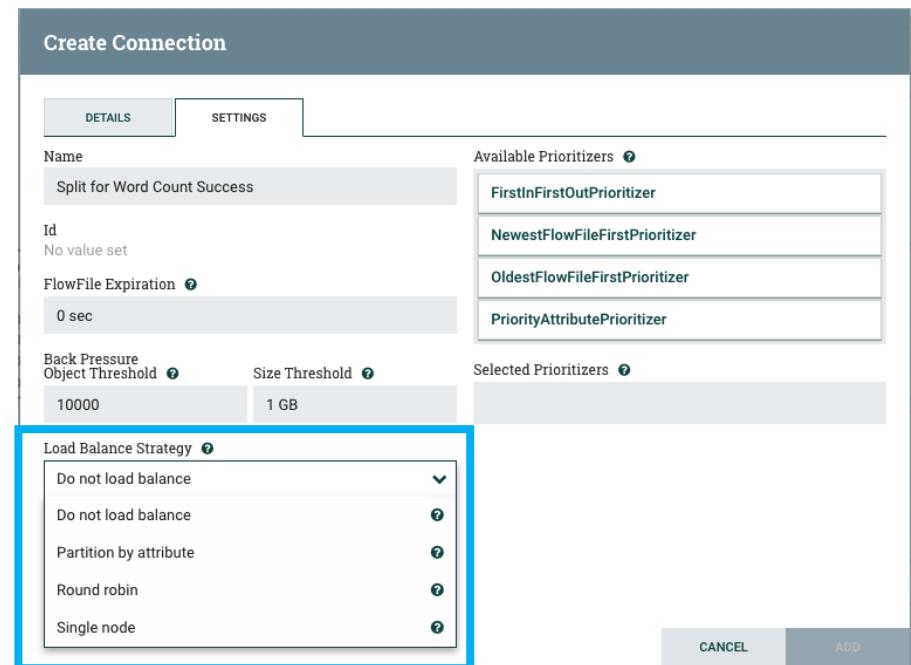
Load Balance Compression ?

Do not compress ▼

Do not compress ?

Compress attributes only ?

Compress attributes and content ?



Configure Connection Settings Tab (4/4)

- Prioritizers allow configuration of which FlowFiles will be processed first
- Click and drag the prioritizers to the selected box
- More than one prioritizer can be selected and priority is applied top down

Available Prioritizers ?

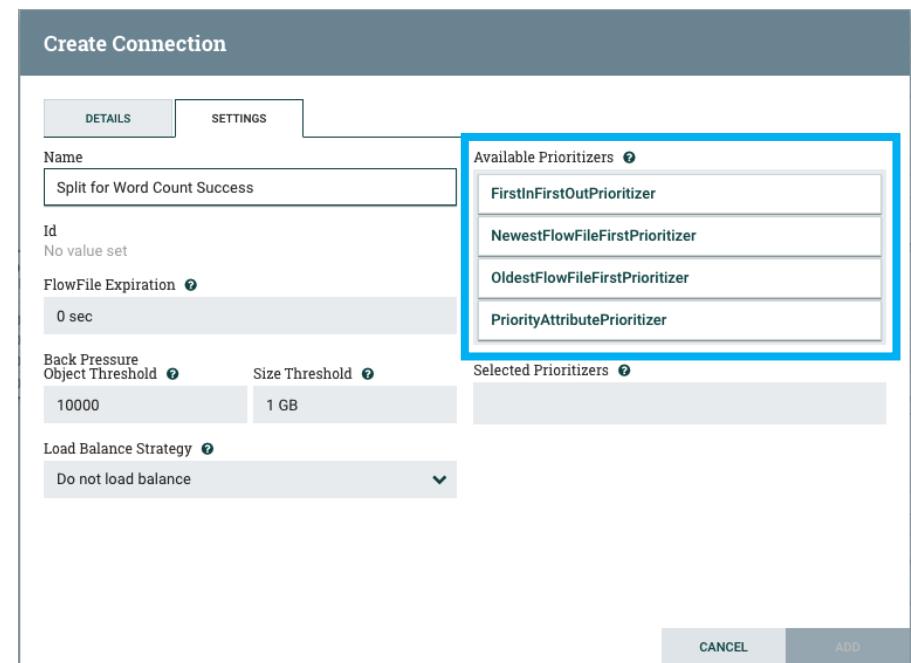
NewestFlowFileFirstPrioritizer

FirstInFirstOutPrioritizer

Selected Prioritizers ?

PriorityAttributePrioritizer

OldestFlowFileFirstPrioritizer



[Lab4]

Creating Connections



Unit 3

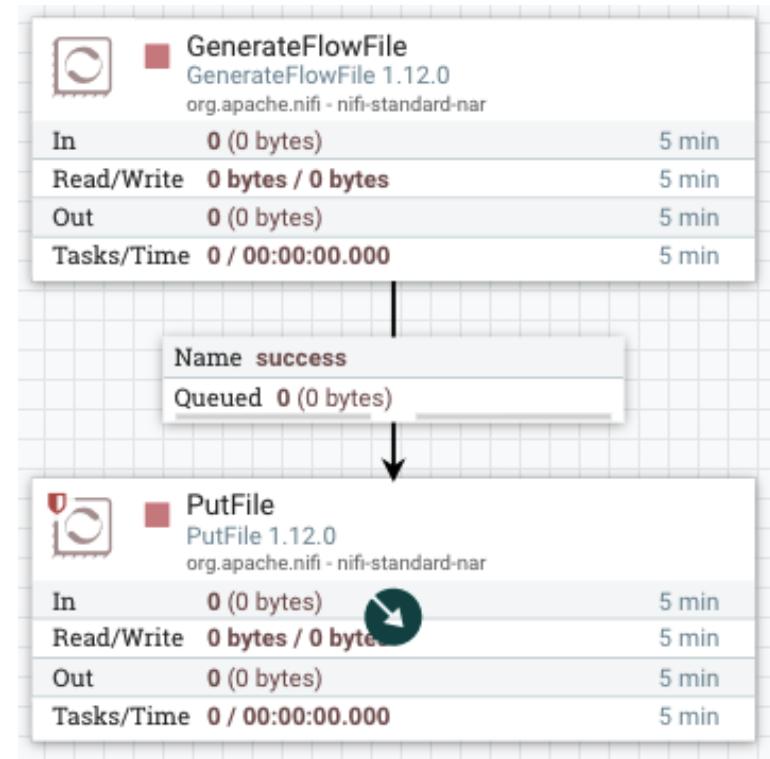
Creating Data Pipelines with Apache NiFi

- | 3.1. Apache NiFi Fundamentals
- | 3.2. Apache NiFi User Interface
- | 3.3. Apache NiFi Processors

- | 3.4. Apache NiFi Connections
- | 3.5. Creating Data Flows
- | 3.6. Process Group

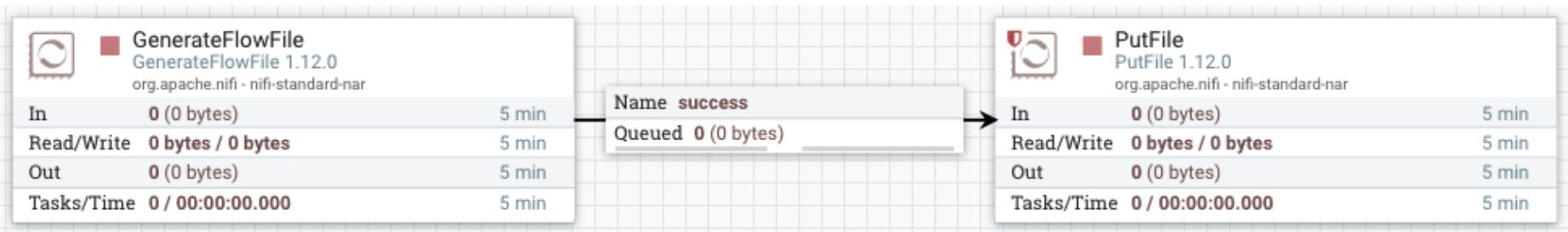
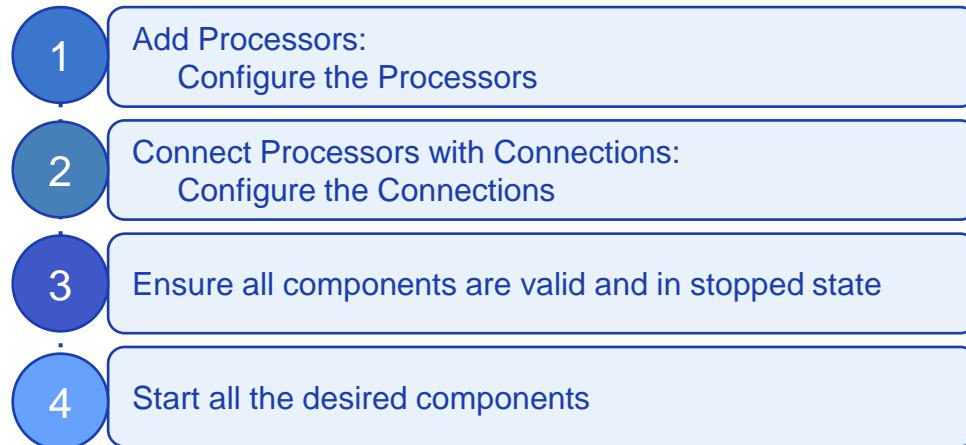
Anatomy of a Data Flow

- Two Processors connected by a Connection represents the most minimal Data Flow
- All processors must be in a valid state 
 - There should not be any yellow warning icons
- All Processor Relationships must be either automatically terminated or connected via a Connection 
- All Processors within the Data Flow must be enabled 
- All Processors within the Data Flow must be stopped
- When all the above conditions are met, select all the components and click  Start



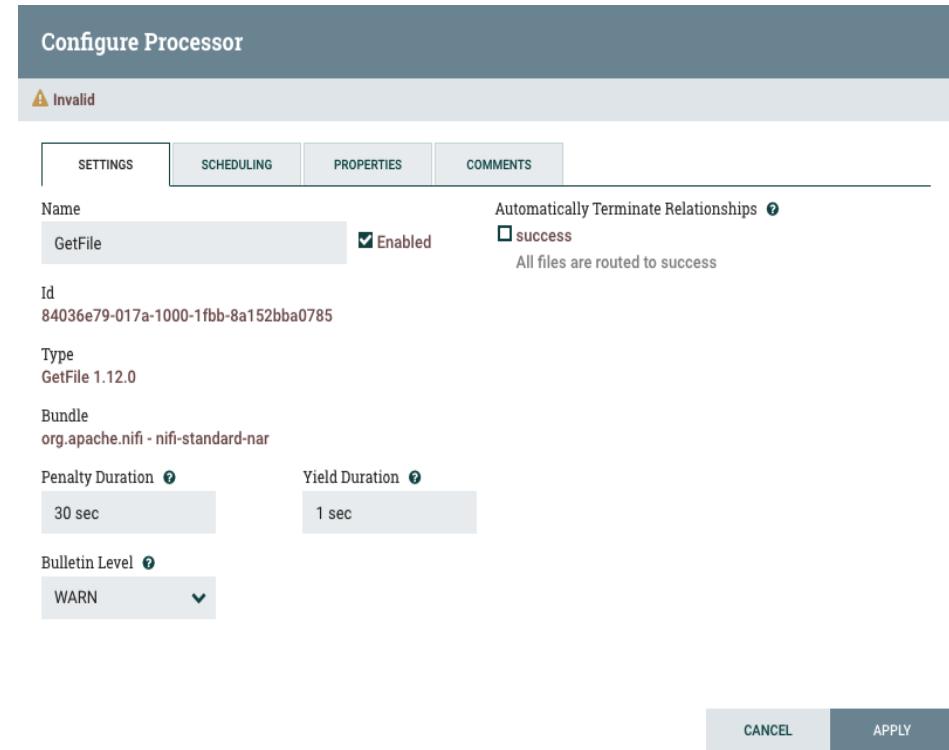
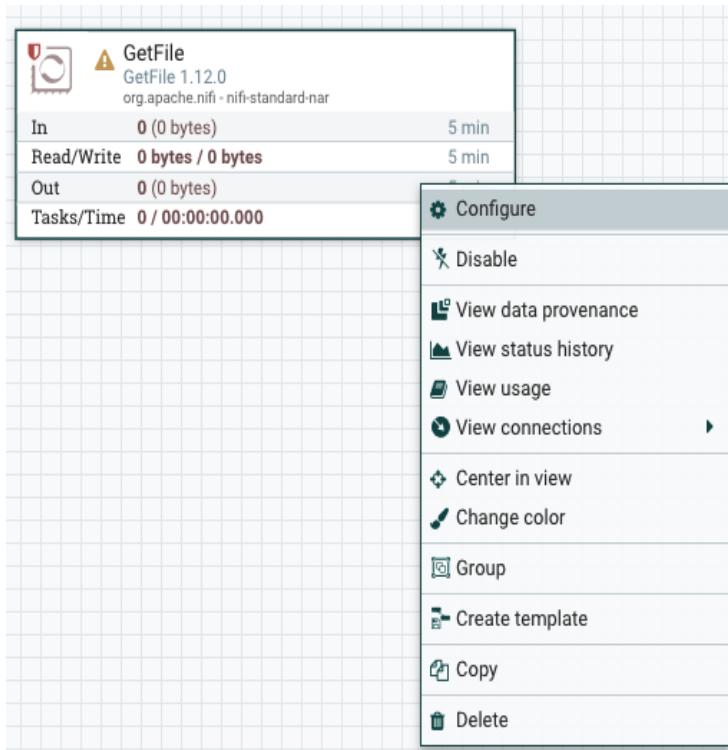
Putting a Data Flow Together

- Even the most complicated data flows have four basic steps



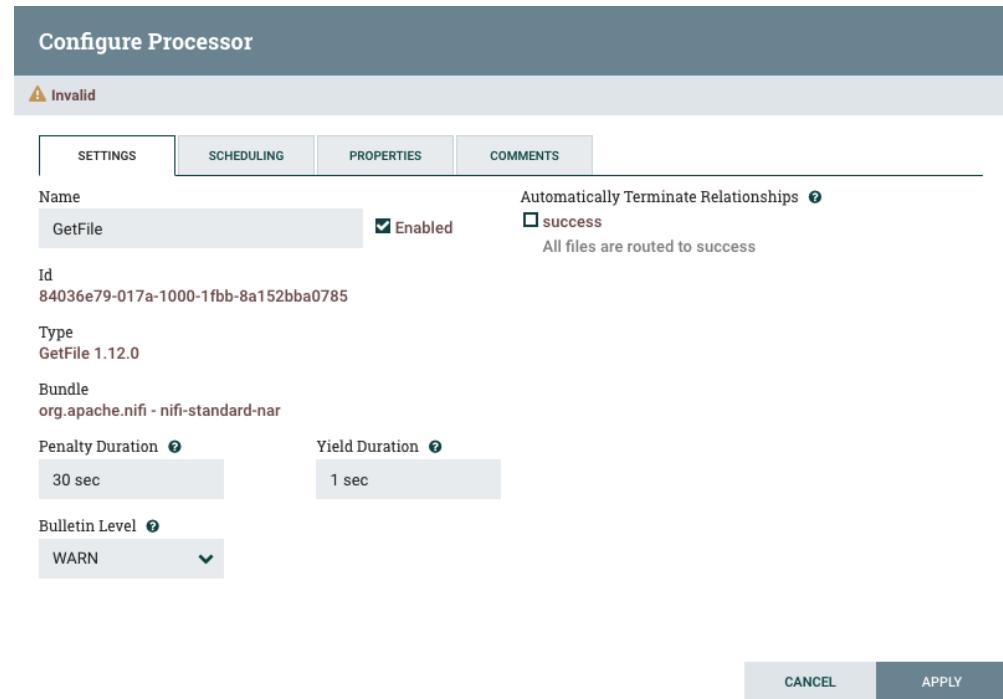
Configuring a Processor

- Right click on processor and select Configure



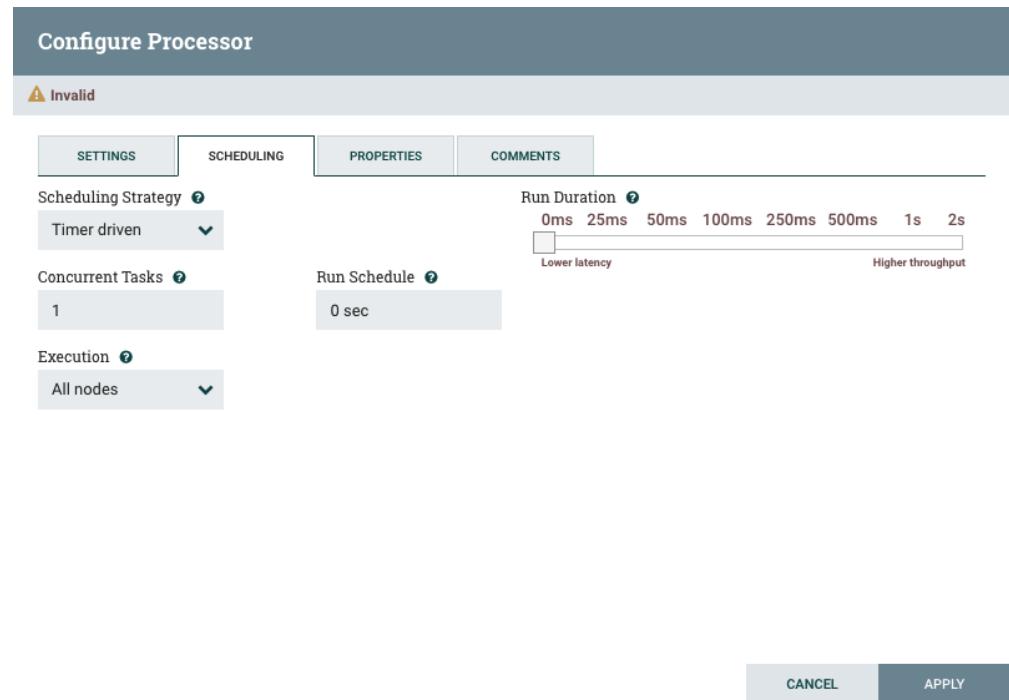
SETTINGS Tab Menu

- Name the Processor
- Enable/Disable
- Processor execution timing
 - Penalty Duration
 - Yield Duration
- Processors generate logs at various levels - DEBUG, INFO, ERROR, etc
 - Bulletins can appear on processors
 - Set the level of log for which a bulletin will be generated
- Connections to downstream processor are available for different conditions
 - SUCCESS, FAILURE, SPLIT, etc
 - Automatically terminate connections



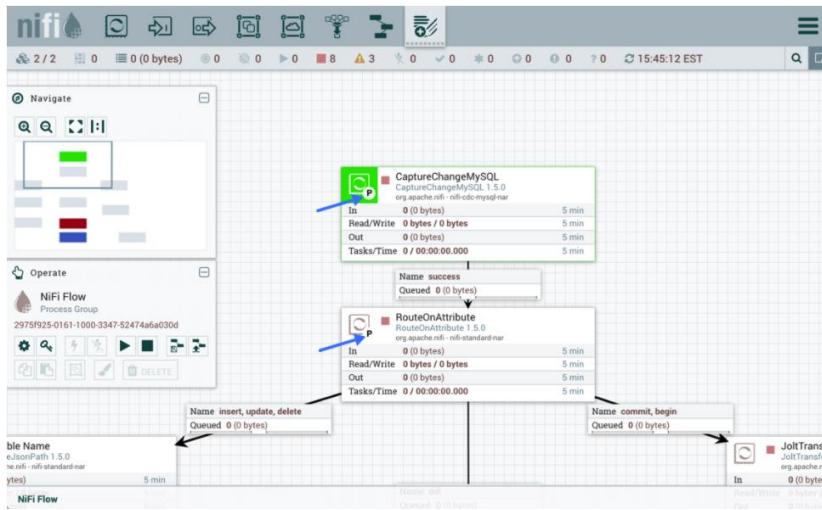
SCHEDULING Tab Menu

- Scheduling Strategy to activate Processors
 - Timer driven - simple timer based interval
 - Event driven - FlowFiles entering the connection to this Processor will trigger an event
 - CRON driven - set up a CRON schedule
- Concurrent Tasks controls the number of concurrent threads activated by the Processor
 - Each thread processes a single FlowFile
- Run Schedule depends on the scheduling strategy used
- Execution allows users to set execution of Processor on all nodes or just the Primary node
- Run Duration controls the amount of time the Processor will run once activated



Identify Primary Node Processors

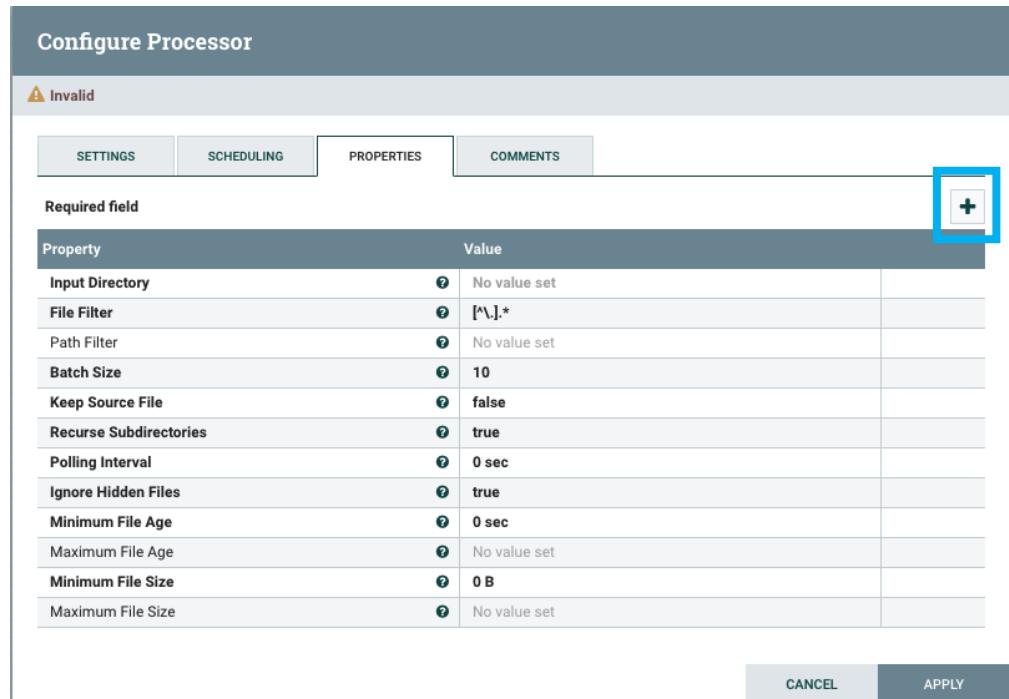
- Processors where only the Primary Node will execute will be indicated by a "P" next to the processor icon
- We can also identify them from the Summary page from the Global Menu



NiFi Summary										
Processors		Input Ports		Output Ports		Remote Process Groups		Connections		Process Groups
Name	Type	Process Group	Run Status	In / Size 5 min	Read / Write 5 min	Out / Size 5 min	Tasks / Time 5 min			
1 P CaptureChangeMySQL	CaptureChangeMySQL	NiFi Flow	Stopped	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000			
2 CaptureChangeMySQL	CaptureChangeMySQL	PG1	Invalid	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000			
3 EnforceOrder	EnforceOrder	PG1	Stopped	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000			
4 EnforceOrder	EnforceOrder	PG1	Stopped	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000			
5 Get Table Name	EvaluateJsonPath	PG1	Stopped	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000			
6 Get Table Name	EvaluateJsonPath	NiFi Flow	Stopped	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000			
7 JoltTransformJSON	JoltTransformJSON	PG1	Stopped	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000			
8 JoltTransformJSON	JoltTransformJSON	NiFi Flow	Stopped	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000			
9 LogAttribute	LogAttribute	PG1	Invalid	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000			
10 LogAttribute	LogAttribute	NiFi Flow	Invalid	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000			
11 LogAttribute	LogAttribute	PG1	Invalid	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000			
12 PutDatabaseRecord	PutDatabaseRecord	PG1	Invalid	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000			
13 PutDatabaseRecord	PutDatabaseRecord	NiFi Flow	Invalid	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000			
14 RouteOnAttribute	RouteOnAttribute	NiFi Flow	Stopped	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000			
15 RouteOnAttribute	RouteOnAttribute	PG1	Stopped	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000			

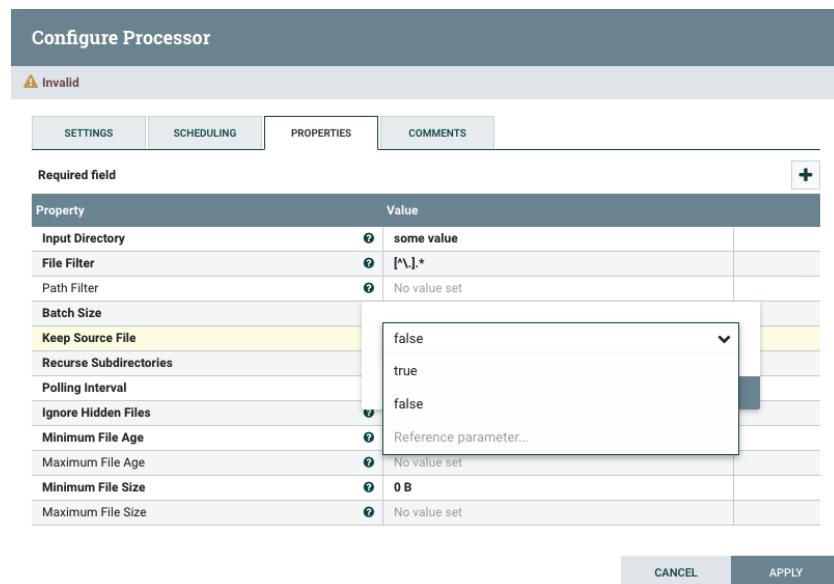
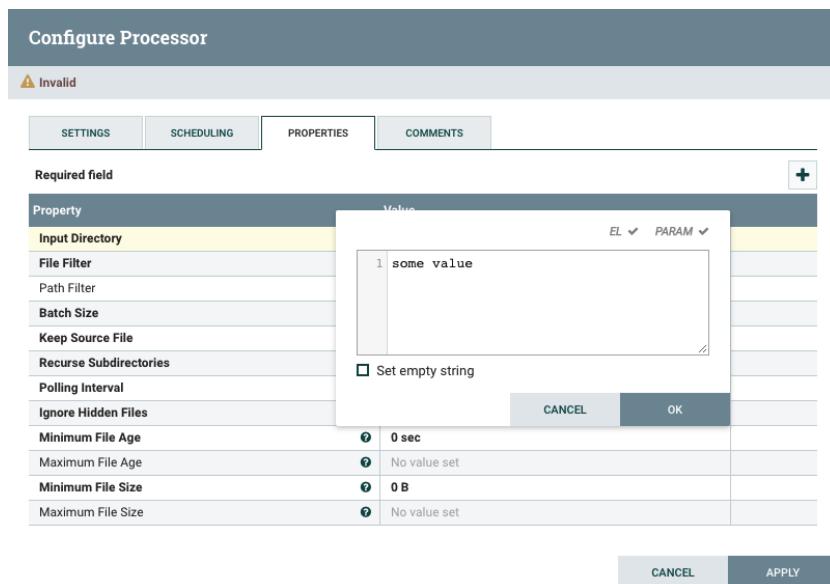
PROPERTIES Tab Menu

- Every Processor will have different properties
- Configure Processor-specific behavior
 - A GetFile Processor for example has settings for location of directory, file filter, hidden files, etc
- All properties are in Key - Value pairs
 - Click to modify Values
-  Users can also set custom properties using the sign
 - Any custom properties will be accessible by downstream processors as attributes



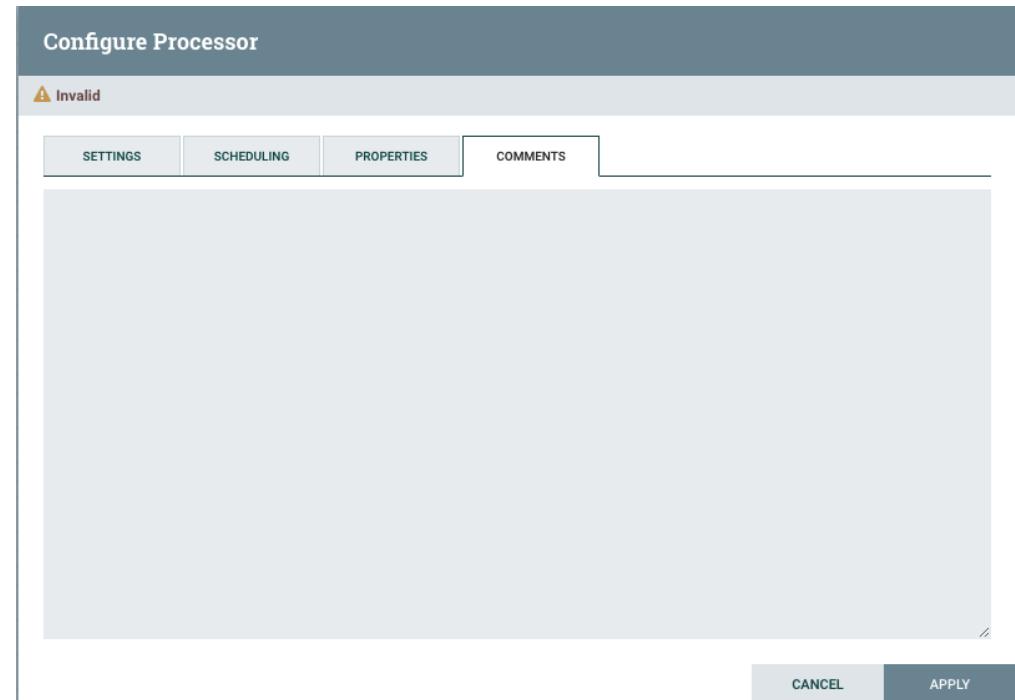
Entering Property Values

- Some properties are directly entered in by the user
- Some will have pop-up windows to select the property



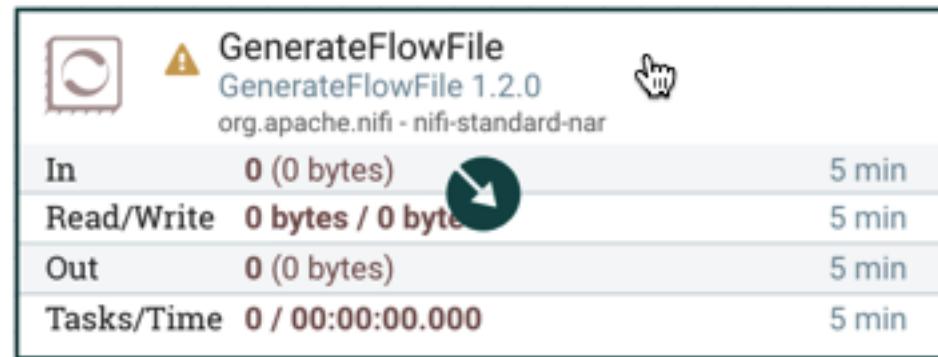
COMMENTS Tab Menu

- Users can add any free form comments appropriate for the current data flow and Processor
- Use is optional



Connect the Components

- Connect two Processor with Connections
- Hover mouse over center of a Proc  until the Connection icon appears
- Click and drag Connection icon to the downstream Processor



Configure Connection Details Tab

- Select the Relationships for which the connection which transport FlowFiles



Configure Connection Settings Tab

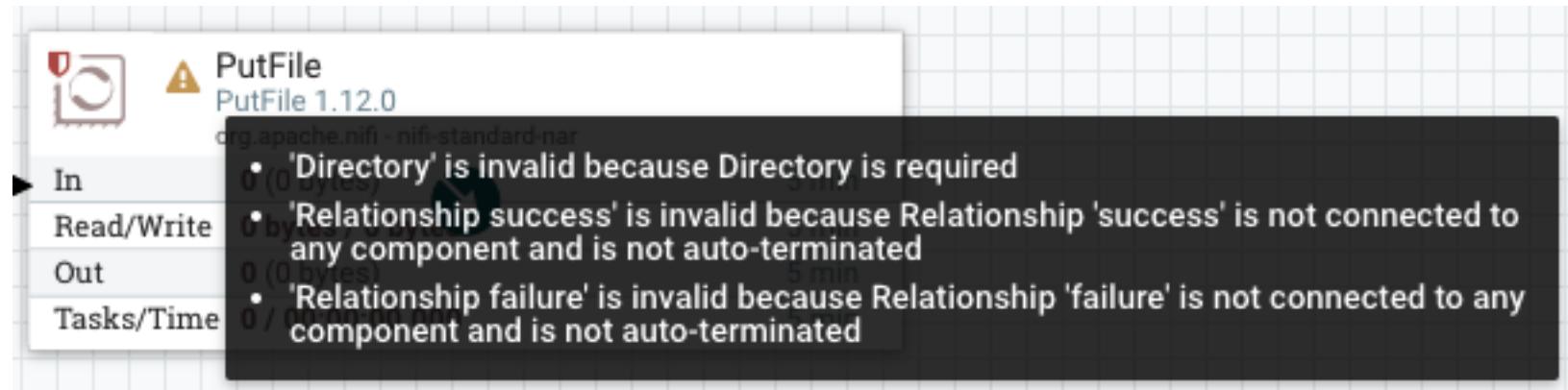
- Name of Connection
- FlowFile Expiration
 - How long can FlowFiles hang around?
- Back Pressure Policy
 - When do we tell upstream to stop sending FlowFiles?
- Load Balance Policy
 - How do we distribute the FlowFiles over the cluster?
- Prioritizer Policy
 - Which FlowFile gets priority service?

Create Connection

DETAILS		SETTINGS	
Name	Split for Word Count Success		
Id	No value set		
FlowFile Expiration	0 sec		
Back Pressure Object Threshold	10000	Size Threshold	1 GB
Load Balance Strategy	Do not load balance		
Available Prioritizers			
FirstInFirstOutPrioritizer			
NewestFlowFileFirstPrioritizer			
OldestFlowFileFirstPrioritizer			
PriorityAttributePrioritizer			
Selected Prioritizers			
CANCEL ADD			

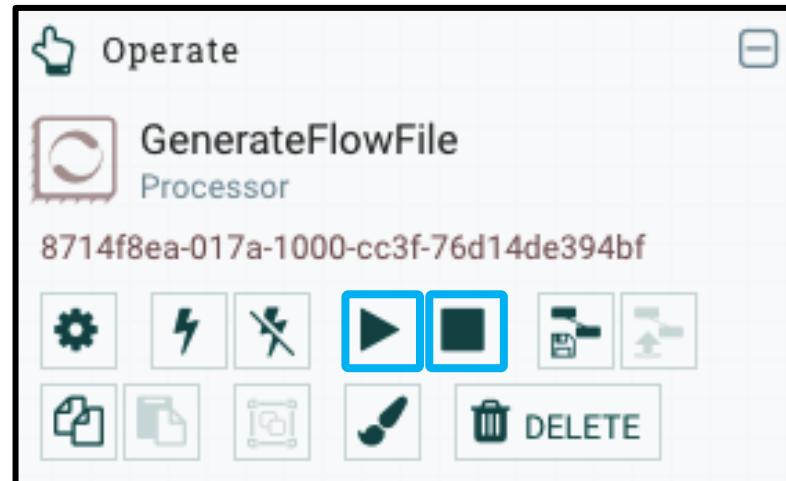
Processor Validation

- Before starting a Processor, it must be validated
 - A yellow  warning status indicator for invalid processors
 - Hovering over the indicator icon will display the validation error
- Once the error is fixed, the processor will change to stopped state



Starting and Stopping Processors

- A Processor must be enabled before it can be started
- A valid Processor in a stopped state can be started
- Start a Processor by
 - Right click on Processor and select  button
 - Select Processor and click  button
- Once started, the Processor will start processing FlowFiles
- Any running Processor can be stopped
 - All currently running tasks will complete
 - Will stop scheduling new tasks



	 GenerateFlowFile GenerateFlowFile 1.12.0 org.apache.nifi - nifi-standard-nar
In	0 (0 bytes) 5 min
Read/Write	0 bytes / 976.56 KB 5 min
Out	10,000 (976.56 KB) 5 min
Tasks/Time	10,000 / 00:00:03.395 5 min



Editing and Debugging Data Flows

- Editing a Processor configuration
 - The Processor must be stopped before editing is possible
- Removing a Processor
 - A Processor that is currently connected to a Connection can not be deleted
- Removing a Connection
 - The Connection buffer queue must be empty before deletion is possible
- Start one processor at a time to debug
 - Only start the next downstream Processor after current hop Processor functionality has been verified
 - Examine FlowFiles generated from current hop for attributes and content

Stateful Processors

- Most Processors are "stateless"
 - They do not keep states or operating information regarding the Processor
- Some Processors, however, must keep states
 - TailFile reads the "tail" of a file or list of files, ingestin data from the file as it is written to the file
 - TailFile stores the location of the last read, files that have been completely read, etc
- Right-click for View state option
- This brings up the current states stored in the processor

The screenshot shows the Apache NiFi interface. On the left, a card displays the TailFile processor details:

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

On the right, a context menu is open with the following options:

- Configure
- Start
- Disable
- View data provenance
- View status history
- View state** (highlighted in grey)
- View usage
- View connections
- Center in view
- Change color
- Create template
- Copy
- Delete

A blue arrow points from the "View state" option in the context menu to a modal window titled "View State" at the bottom left. This modal shows a table of current states:

Key	Value
file.0.checksum	2798311963
file.0.filename	/opt/nifi/nifi-current/logs/nifi-app.log
file.0.length	26527
file.0.position	26527
file.0.timestamp	1625840342000

Removing States from Processor

- While debugging a data flow, users often have to re-run them
- If the data flow includes stateful Processors, they need to be reset
 - If states are not reset, the Processor might not process any data
 - For example, in the TailFile processor, there may not be any new "tail" information generated by the system
 - Because the state has not been reset, none of the older data will be considered for reading

Displaying 5 of 5

Key ▲	Value
file.0.checksum	2798311963
file.0.filename	/opt/nifi/nifi-current/logs/nifi-app.log
file.0.length	26527
file.0.position	26527
file.0.timestamp	1625840342000

[Clear state](#)

Unit 3

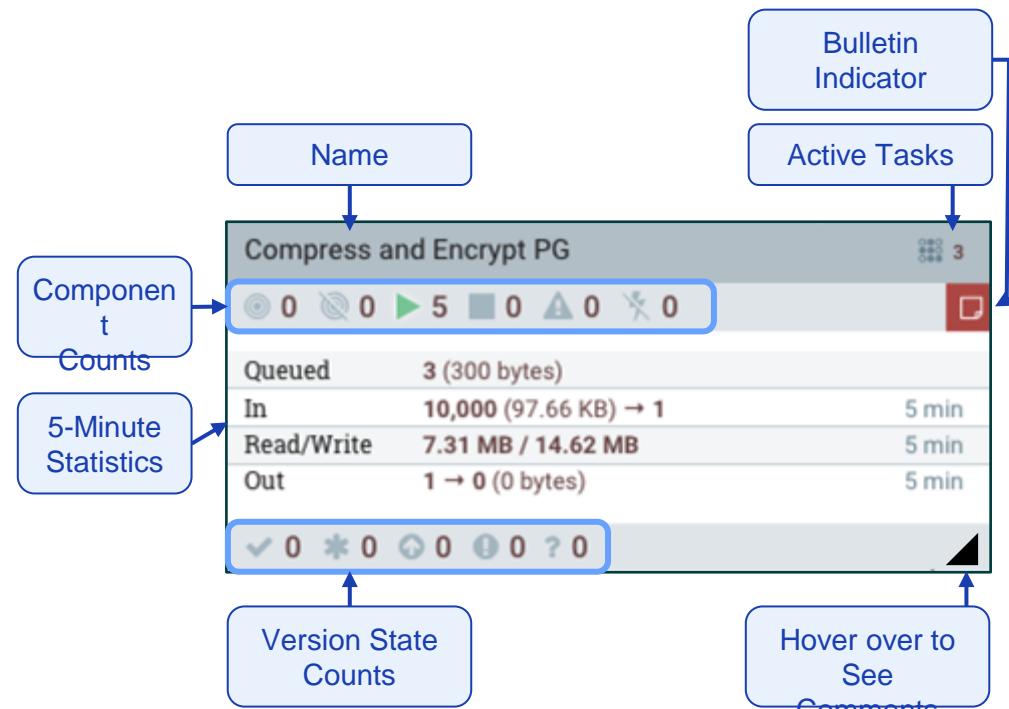
Creating Data Pipelines with Apache NiFi

- | 3.1. Apache NiFi Fundamentals
- | 3.2. Apache NiFi User Interface
- | 3.3. Apache NiFi Processors

- | 3.4. Apache NiFi Connections
- | 3.5. Creating Data Flows
- | 3.6. Process Group

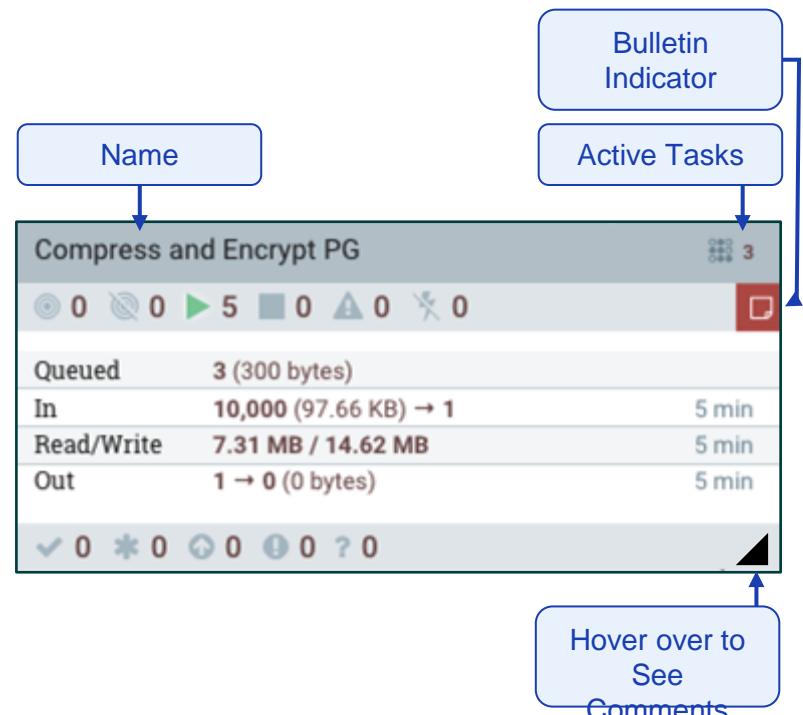
What is a Process Group?

- Process Group provides a way for grouping components
 - Contains a set of Processors, Connections
 - May contain Input and Output Ports
- Create and organize logical constructs
 - Very similar to functions in a program
 - Can be copied and reused with parameter changes
- Another perspective is as sub-folders
 - Subdivide the canvas by users, groups, projects, etc
 - Nesting is possible for even more control
- Security and Version Control
 - Available at Process Group Level



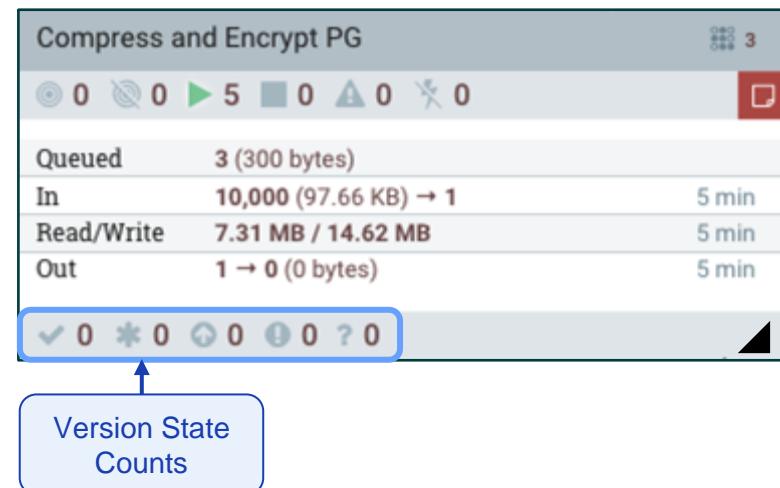
Anatomy of a Process Group (1/4)

- Name
 - User-defined friendly name of Process Group
- Active Tasks
 - Total number of tasks currently executing by all components within the Process Group
- Bulletin Indicator
 - When any child component of the Process Group emits a bulletin, it is propagated up to the Process Group
- Comments
 - Comments can be added when the Process Group is created
 - Change comments from the Processor Group configure menu



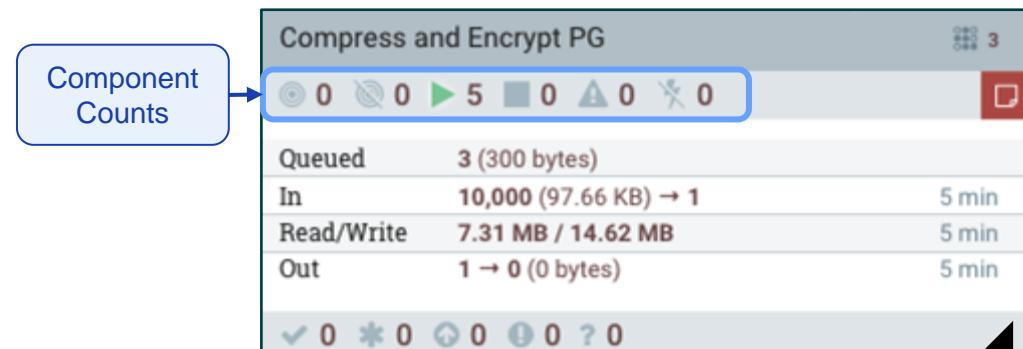
Anatomy of a Process Group (2/4)

- Version State Count
 - Version control of dataflows is available at the Process Group Level
- Following indicators possible:
 - Version up to date
 - Locally modified
 - Stale
 - Locally modified and stale
 - Sync Failure



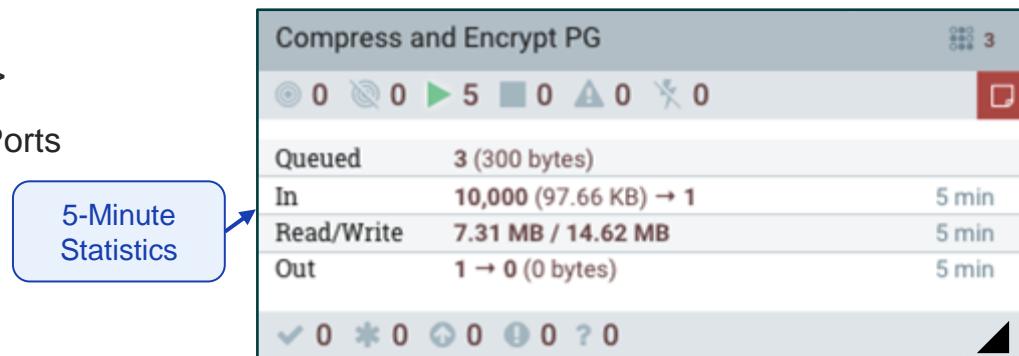
Anatomy of a Process Group (3/4)

- Component Counts
 - Total count, by current state, of all the components in the Process Group
- Following indicators possible:
 -  Transmitting Ports
 -  Non-transmitting Ports
 -  Running Components
 -  Stopped Components
 -  Invalid Components
 -  Disabled Components



Anatomy of a Process Group (4/4)

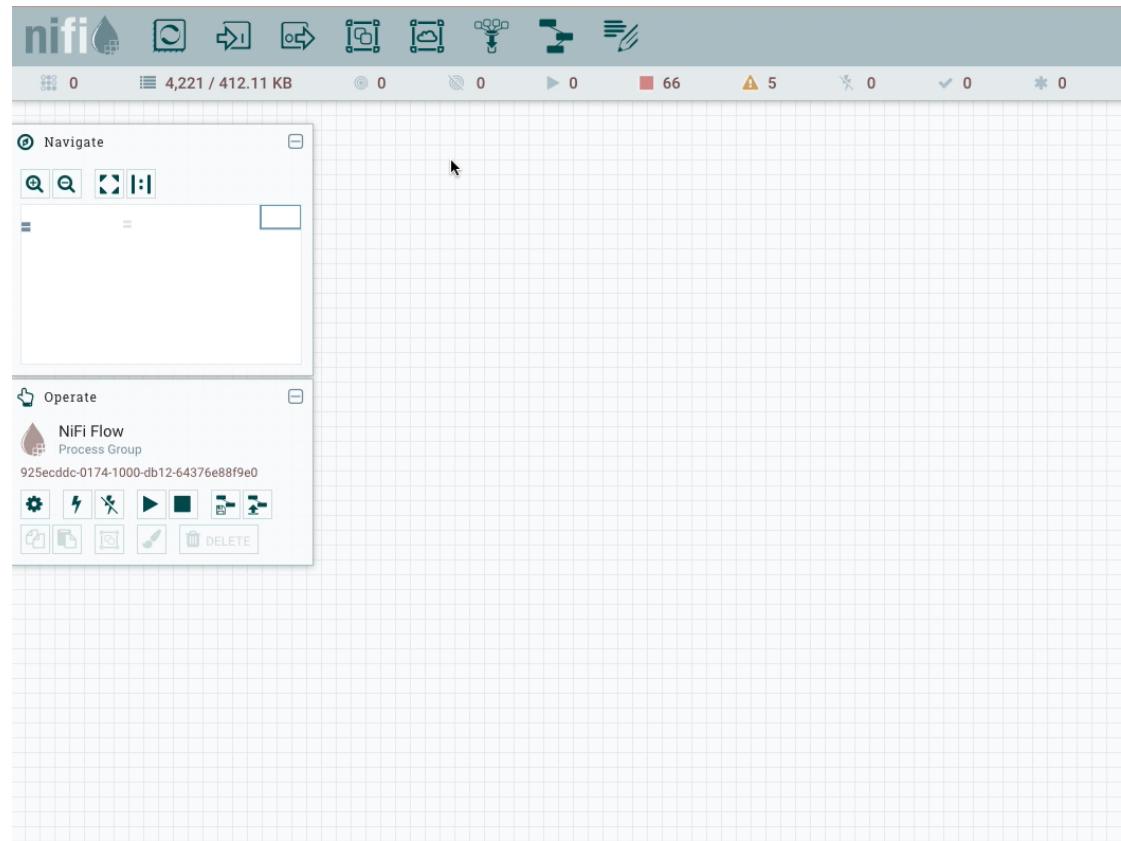
- 5-minute aggregated status of all components in the Process Group (PG)
- Queued - <count>(<size>)
 - Total count FlowFiles enqueued in the PG
- In - <count>(<size>) → <number of Input Ports>
 - FlowFiles transferred into the PG through Input Ports
- Read/Write - <size>/<size>
 - Total size of FlowFile content that Process Group has read and written to disk
- Out - <number of Input Ports> <count>(<size>)
 - FlowFiles transferred out of the PG through Output Ports



Create Empty Process Group

[Video]

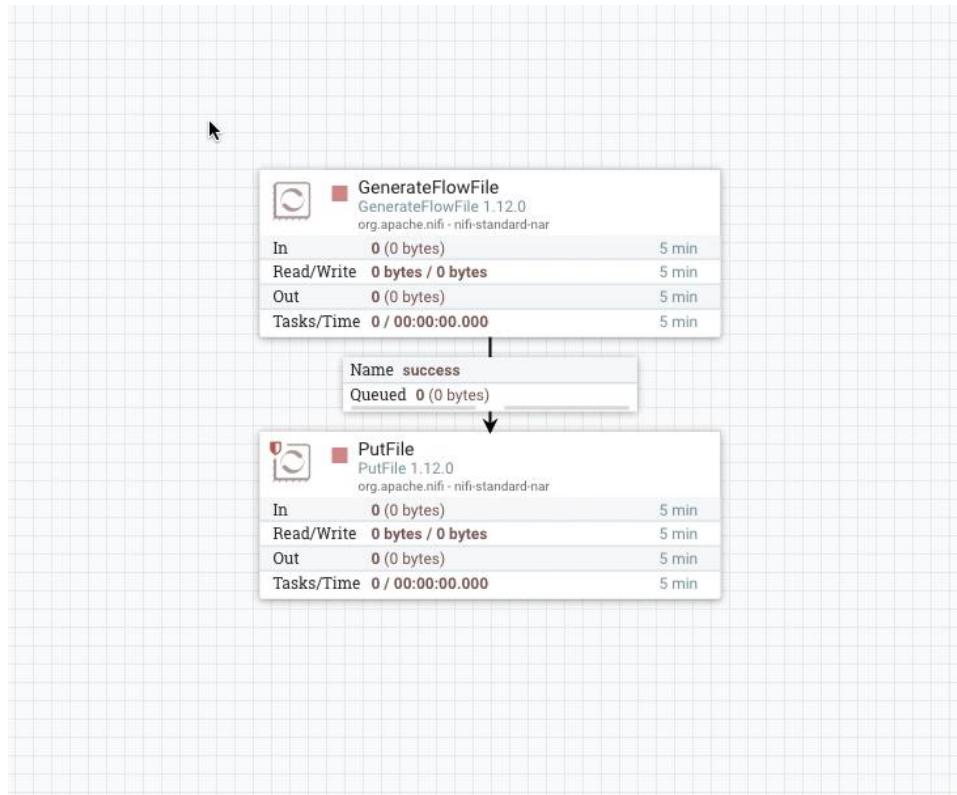
- Creating an empty Process Group
 - Click and drag the Process Group icon to the canvas
 - Name the Process Group



Create Process Group from Components

[Video]

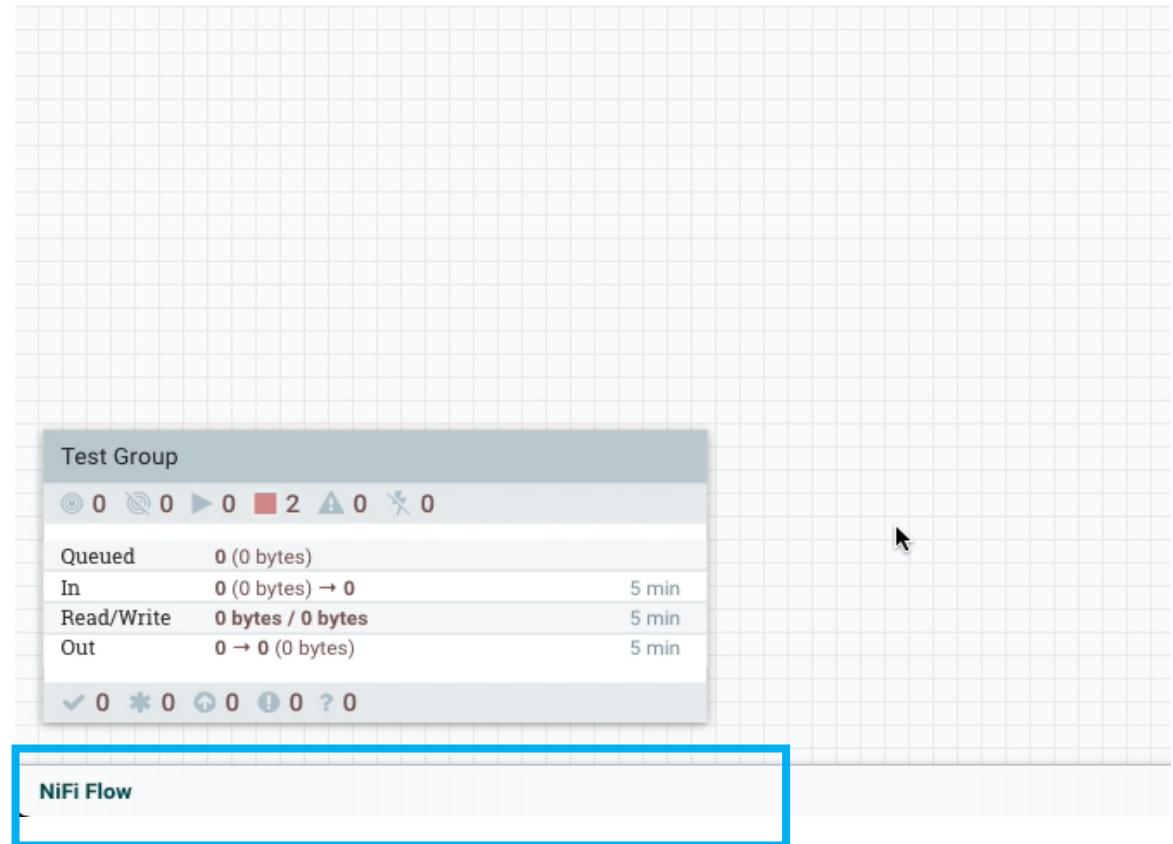
- Select the components to add to the Process Group
- Right-click and select Group
- Enter name for new Process Group



Managing Nested Processor Groups

[Video]

- It is possible to create Process Groups inside Process Group
- Double-click a Processor Group to enter it
 - When you enter a Processor Group, NiFi will leave breadcrumbs for you to find you way back home



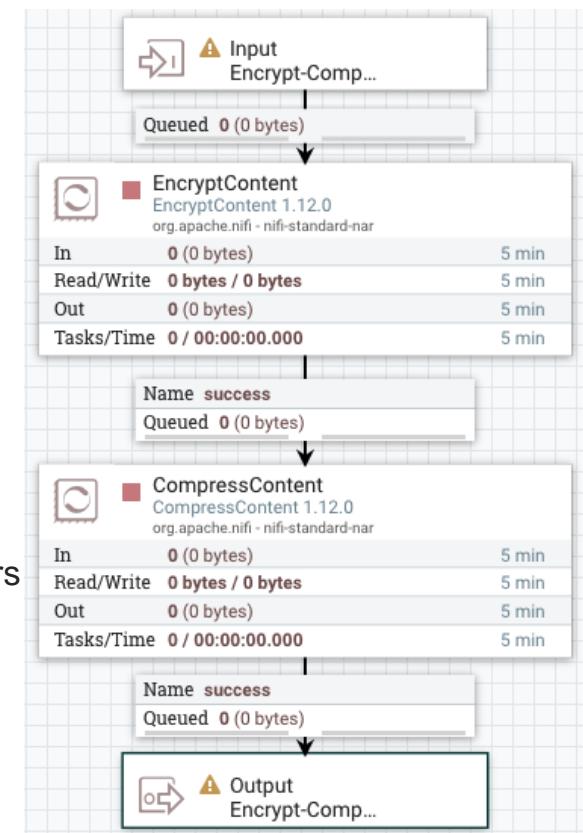
Input Port and Output Port

- Input Ports provide a mechanism for transferring data into a Process Group
- Output Ports provide a mechanism to transfer data out of the Process Group
- All Ports must have a unique name within the Processor Group that it belongs to
- Data is exchanged by and between the current Process Group and a component in parent group
- If an Input Port or Output Port is created on the root canvas, it provides a mechanism for site-to-site communication
 - Receive data through Input Port from a remote and separate NiFi cluster
 - Send data through Output Port to a remote NiFi cluster



Adding Input and Output Ports

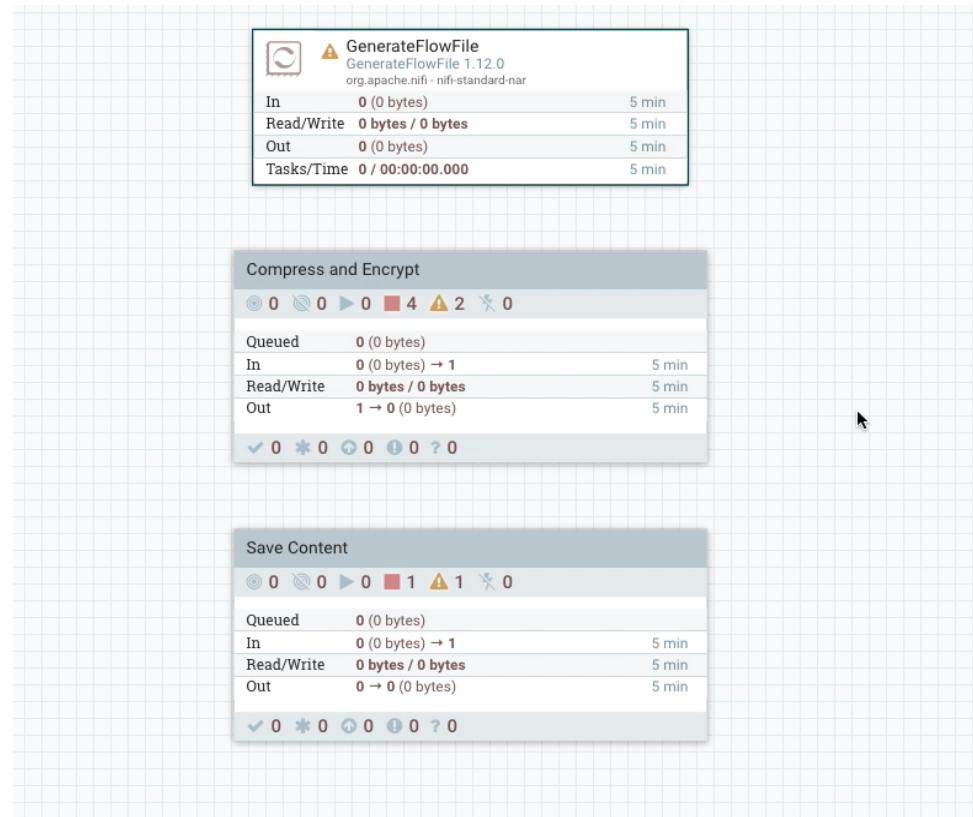
- Click and drag either Input Port or Output Port to the Processor Group
- Provide a unique name for the Port within the PG
- Connect the Ports to components within the PG
- Connecting an Input Port
 - Hover the mouse over the Input Port until the connection icon appears
 - Connect to desired Processor
- Connecting an Output Port
 - Hover the mouse over the output component until the connection icon appears
 - Connect to the Output Port
 - Set the Relationship for the Connection



Putting It All Together

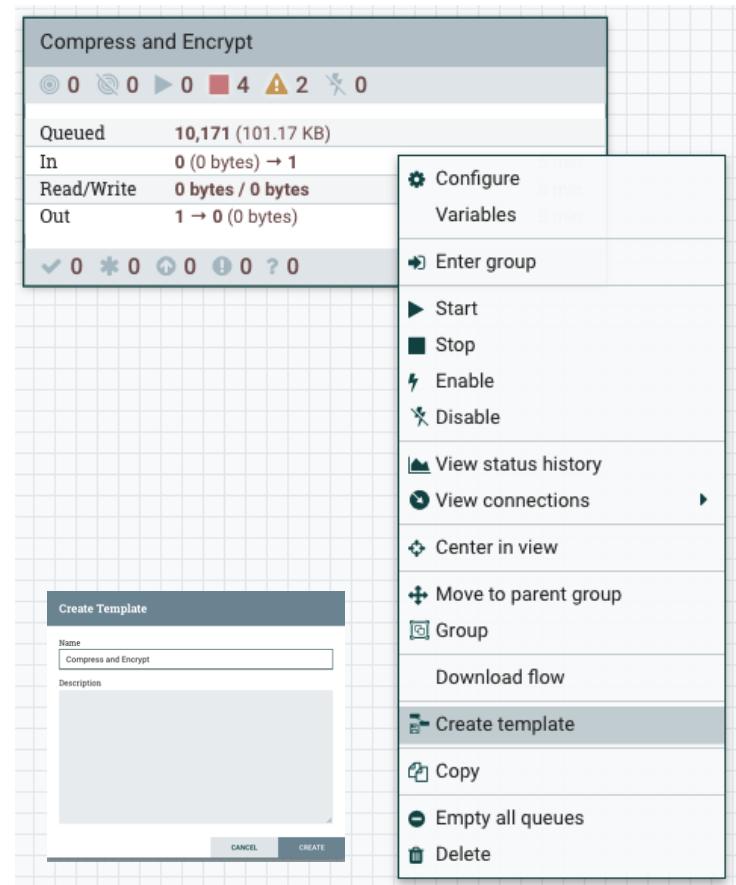
[Video]

- We will demonstrate creating a data flow that utilizes the Input Port and Output Port
- The ports can connect to other Processors or other Processor Groups
 - The Processor or Processor Group must be in the parent group



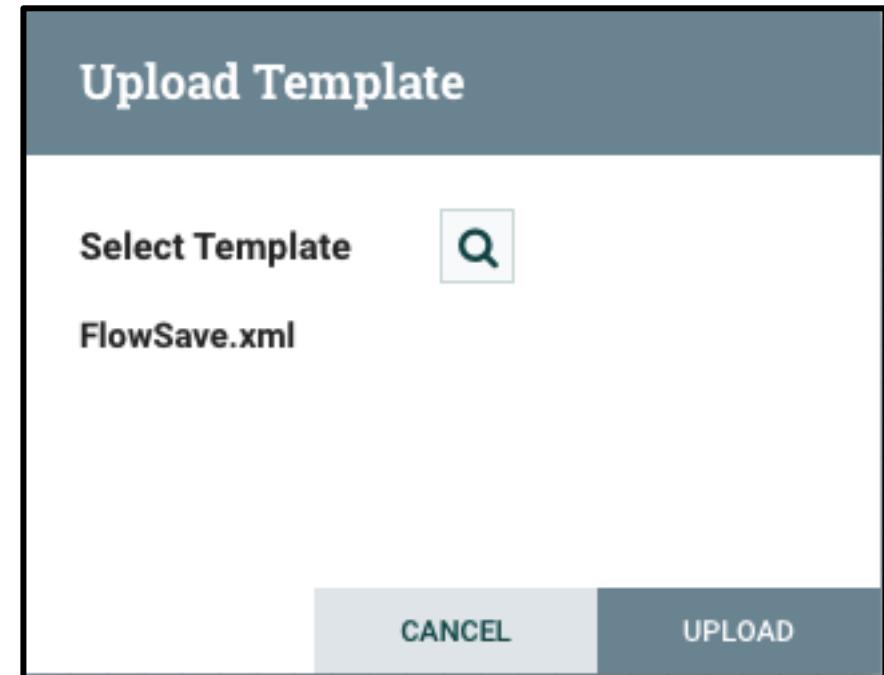
Creating Templates

- The Compress and Encrypt Process Group performs a data flow function that we might often utilize
- We can use Templates to save data flows
 - Avoid repeatedly creating the same data flow
- Templates can include:
 - Processors, Funnels
 - Input and Output Ports
 - Process Group and Remote Process Group
- Select the components that are part of the template
 - Right click and Create template
 - OR, click the Create Template button from the Operate Palette
 - Provide a name for the template



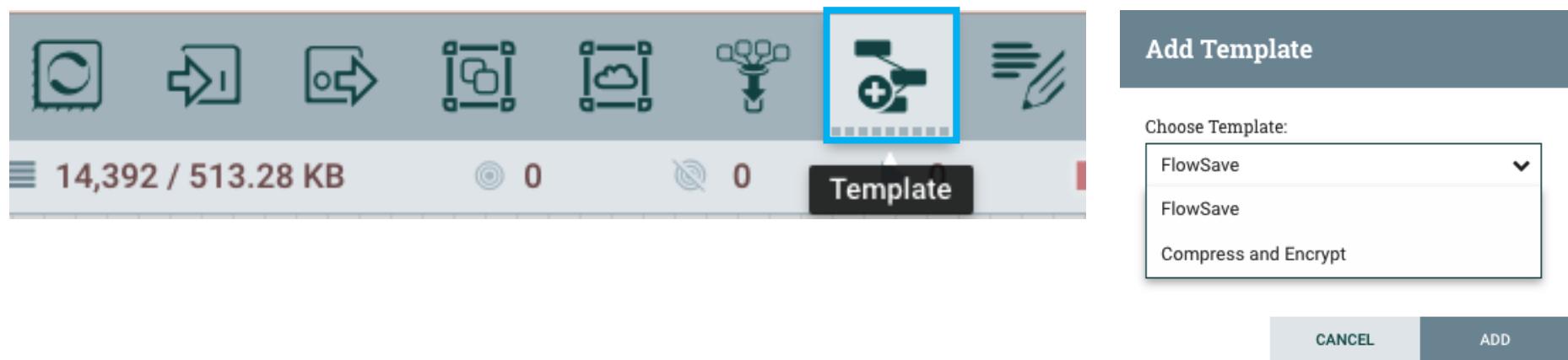
Importing Templates

- NiFi Templates are saved as XML files
- NiFi can import XML templates into current instance of NiFi
- From the Operate Palette, click the Upload Template button 
- Click on the  to browse for templates
- Select the desired template and upload



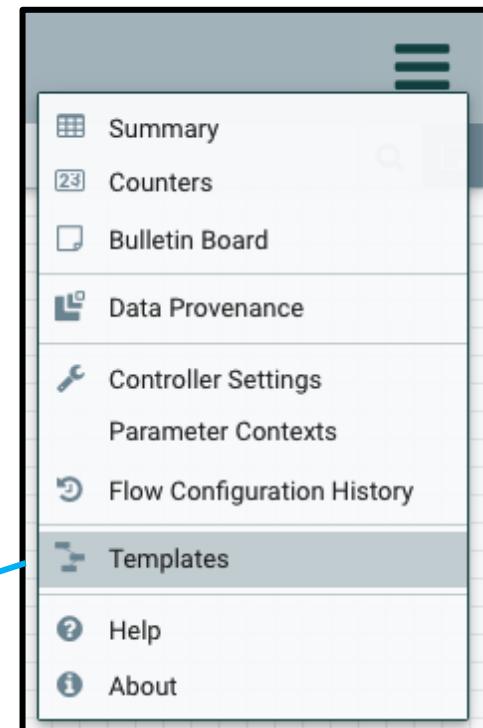
Adding a Template to Data Flow

- Click and drag the Template icon from the Components Toolbar
- Choose the desired template and add



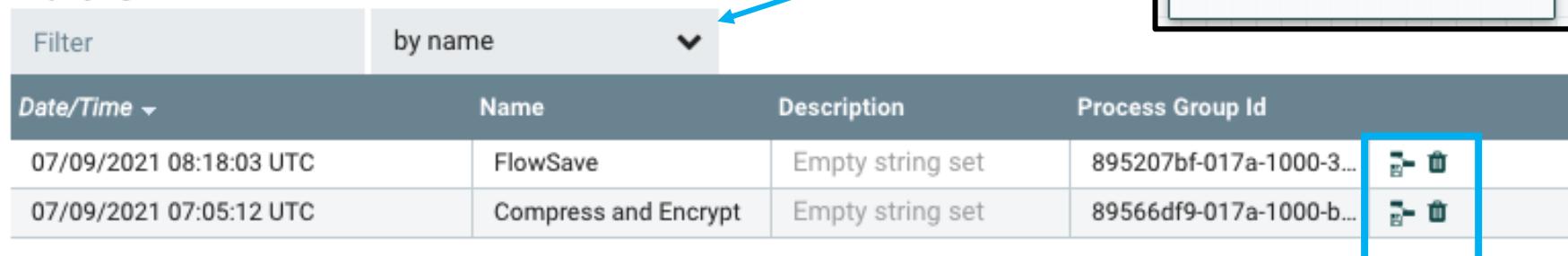
Manage Templates

- From the Global Menu, select Templates
- The NiFi Templates Manager window opens
- From this menu, you may download  a template as XML file 
- Delete a template



NiFi Templates

Displaying 2 of 2



Date/Time	Name	Description	Process Group Id	
07/09/2021 08:18:03 UTC	FlowSave	Empty string set	895207bf-017a-1000-3...	 
07/09/2021 07:05:12 UTC	Compress and Encrypt	Empty string set	89566df9-017a-1000-b...	 

[Lab5]

Navigating Data Flows



[Lab6]

Creating and Using Templates



[Lab7]

Using Processor Groups



[Lab8]

Setting Back Pressure on your Connections



[Lab9]

Working with Hadoop in NiFi



Unit 4.

Apache Kafka

Big Data Ingestion

Unit 4

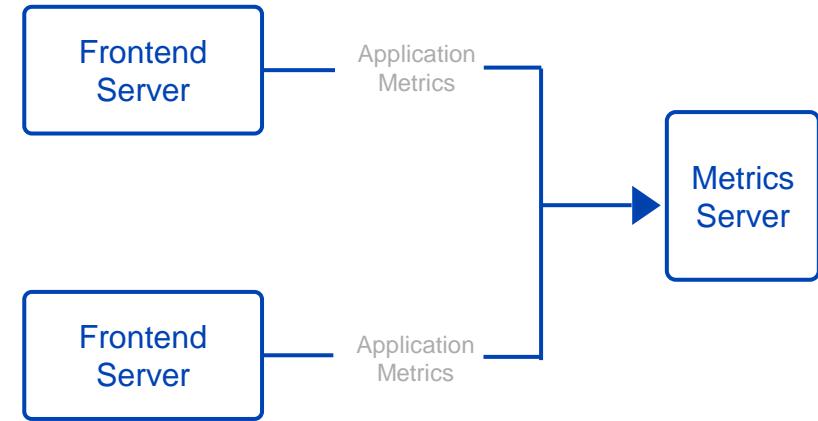
Apache Kafka

| 4.1. Apache Kafka Fundamentals

| 4.2. Apache Kafka Architecture

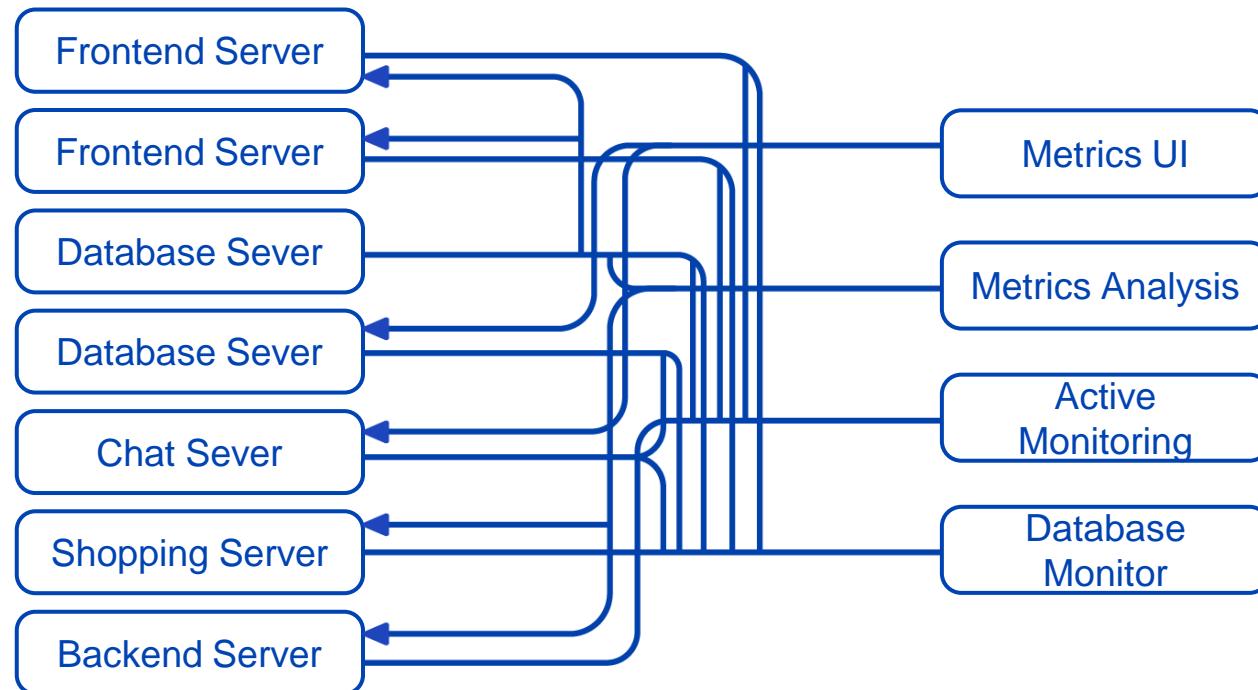
Data Transfer from Message Queue (1/2)

- Moving from Monolithic to Microservice based development
 - Microservices communicate with each other through message queues
- Message Queue - a communication method used by a process or process instance to exchange data with each other
- Microservices are often categorized and producers of messages while other are consumers of messages
 - Some services may be both producer and consumer
- Other use cases for message queues
 - Batch processing to process large amounts of data (Big Data)
 - Chat services
 - Asynchronous data processing



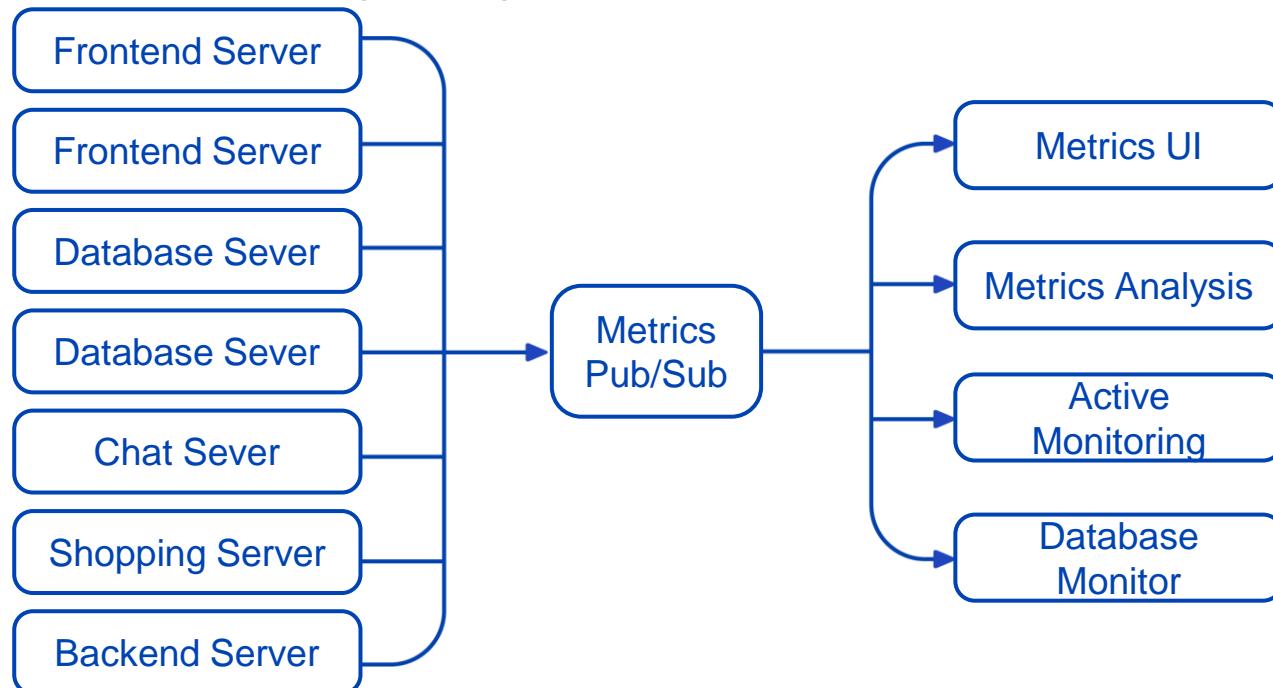
Data Transfer from Message Queue (2/2)

- As needs and requirements expand, soon you end up with a complicated and unmanageable tangle of communications



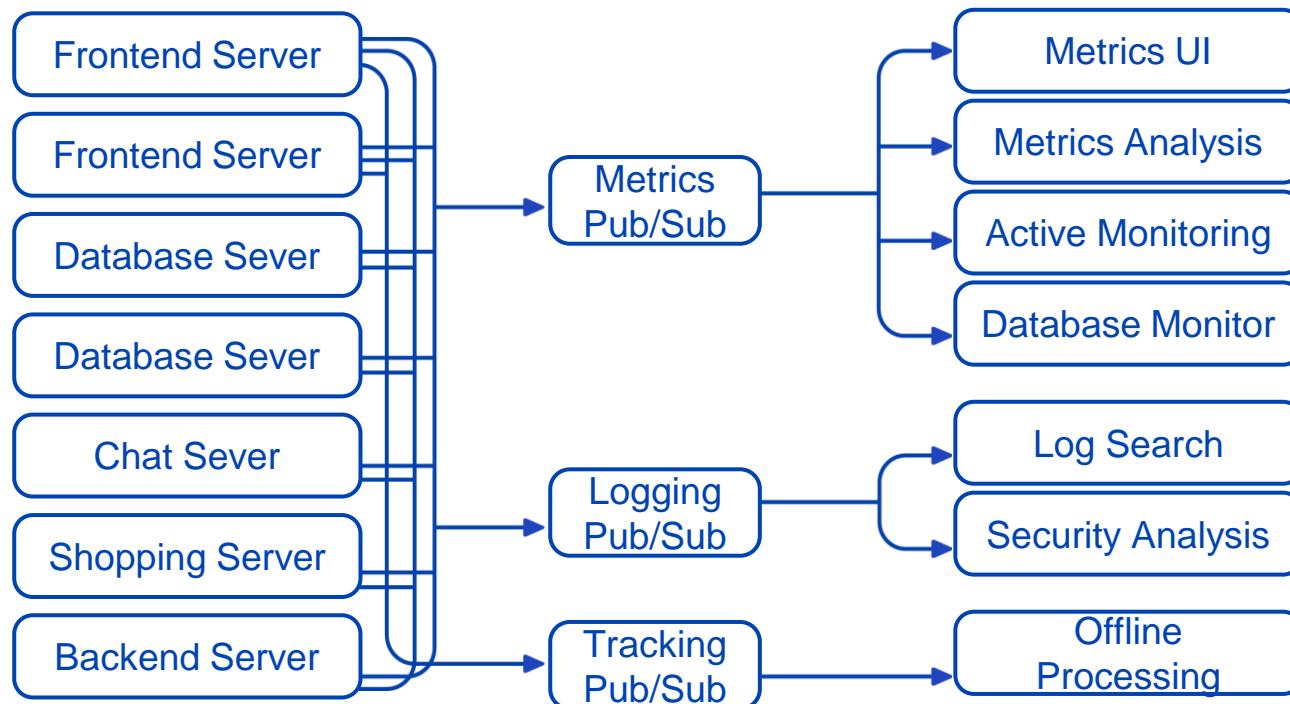
A Case for a Better Pub/Sub System (1/2)

- Replace the myriad connections with a publish-subscribe system
- Typically, a set of publishers pushing messages to a broker in the middle and subscribers pulling messages



A Case for a Better Pub/Sub System (2/2)

- A publish-subscribe system is much better but soon you start getting other members of your organization setting up their own pub/sub systems as well



Kafka – Messaging system (1/2)

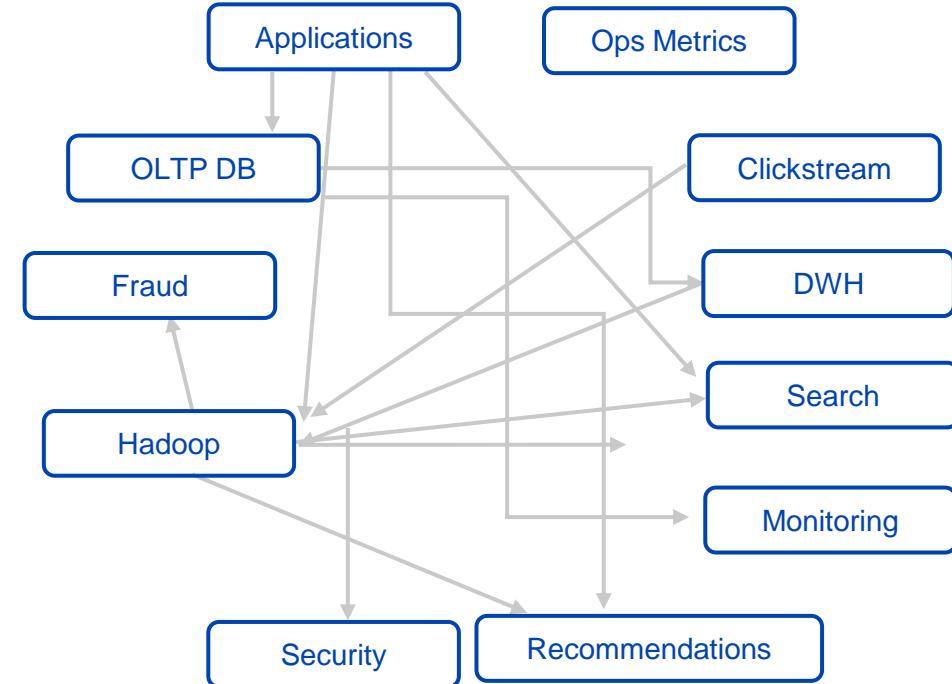
- Messaging systems generally have 2 models
 - Queueing – a pool of consumers read from servers and each record goes to one of them
 - Publish/Subscribe – Record is broadcast to all consumers
- Consumer group generalizes these two models
 - As in a queue, processing can be divided among the consumer processes in the consumer group
 - As with pub/sub, messages can be broadcast to consumer groups
 - This gives Kafka the advantages of both systems

Kafka – Messaging system (2/2)

- In addition to having both queue and pub/sub properties, Kafka's model allows stronger ordering guarantees
- Traditional queue retains records in order,
 - With parallel access, and asynchronous communication, that order may not be preserved
- Kafka's partition fixes this problem
 - A topic can be split into multiple partitions for parallel access
 - Within a consumer group, only one consumer process may access a particular partition (“exclusive” consumer to ensure order)

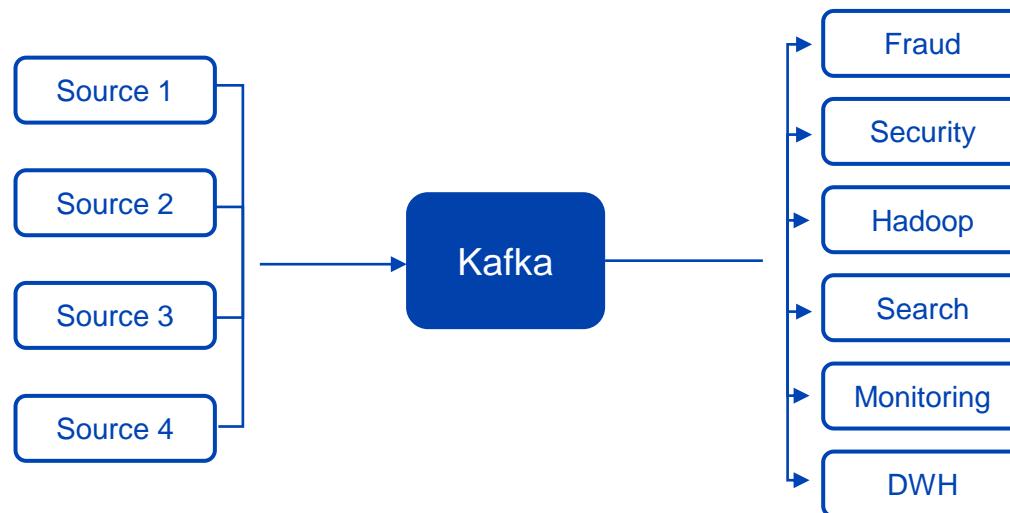
Typical Underlying Data Flow

- Before the introduction of Kafka
 - Increased complexity of data integration (hardware, operating system, failures, etc.)
 - Different data pipeline connection structure
 - Great effort required for expansion
- For most systems, however, the underlying data flow is much more complicated and involves much more than just Hadoop



Typical Underlying Data Flow using Kafka

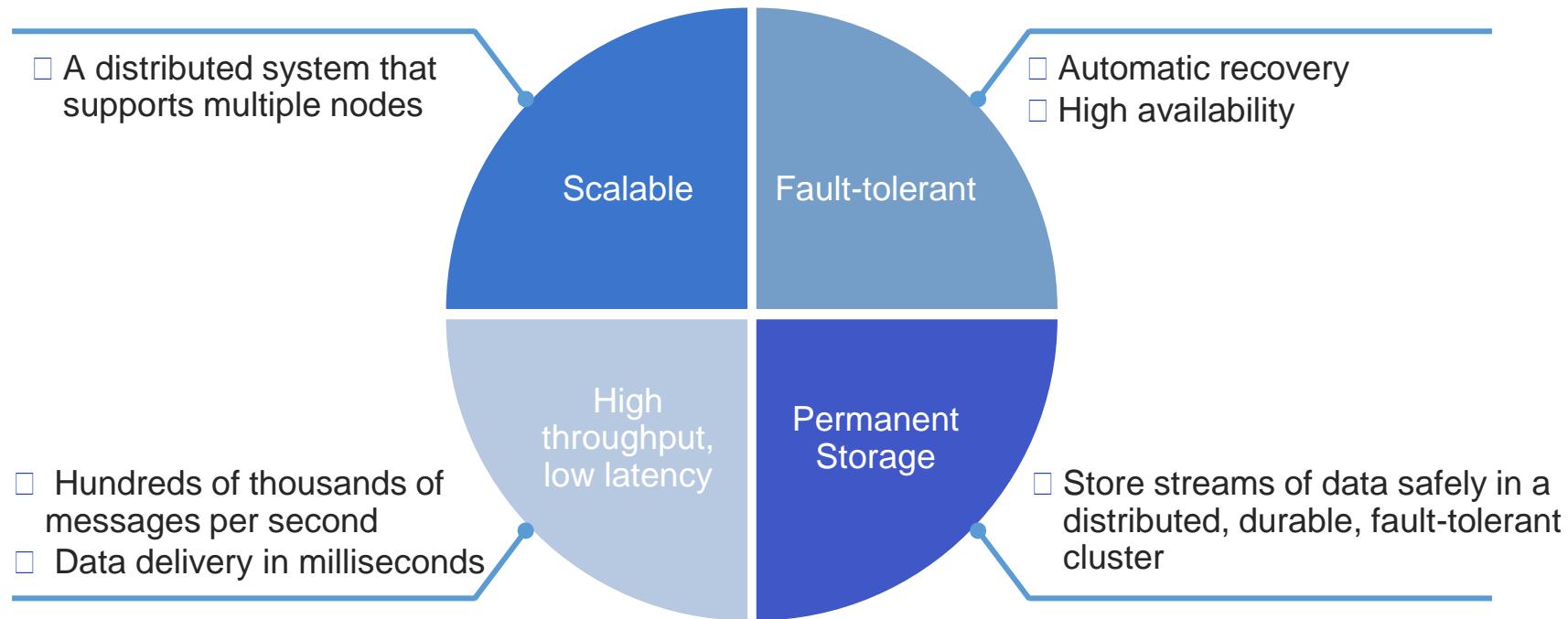
- After the introduction of Kafka
 - Decouples publisher and subscriber for messages
 - Organized by topic to communicate each other



What is Apache Kafka?

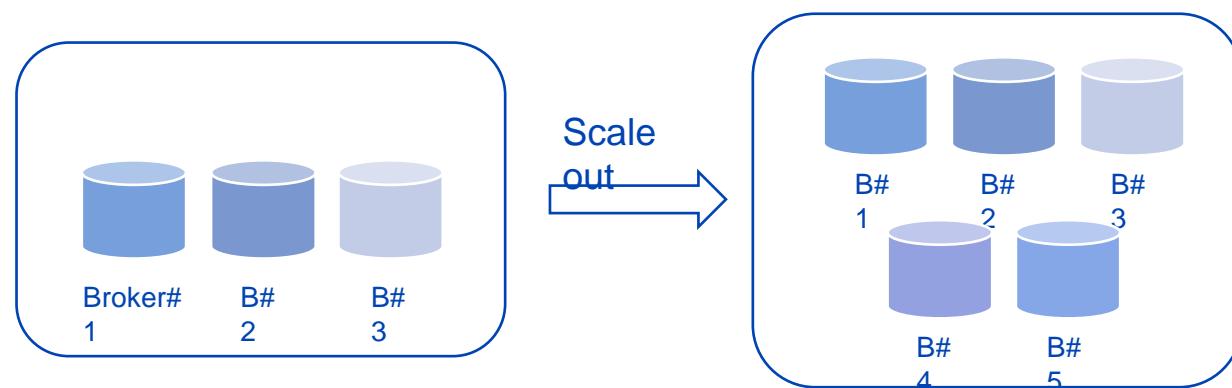
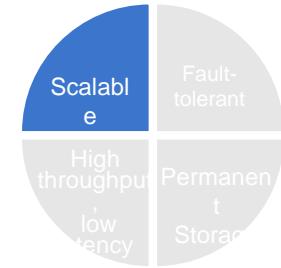
- Apache Kafka is a publish/subscribe messaging system designed to solve the problems of a conventional pub/sub system
- Apache Kafka is a fast, scalable, distributed publish/subscribe messaging system
- Kafka has basic functions of a distributed streaming platform. Kafka can be conveniently used to develop the following applications:
 - Build a real-time streaming data pipeline for secure data transfer between the system and / or the system and the application
 - Build a real-time streaming application that transforms and receives data streams for immediate processing
- Originally created at LinkedIn
 - Donated to Apache Software Foundation in 2011

Why is Kafka suitable for data pipelines?



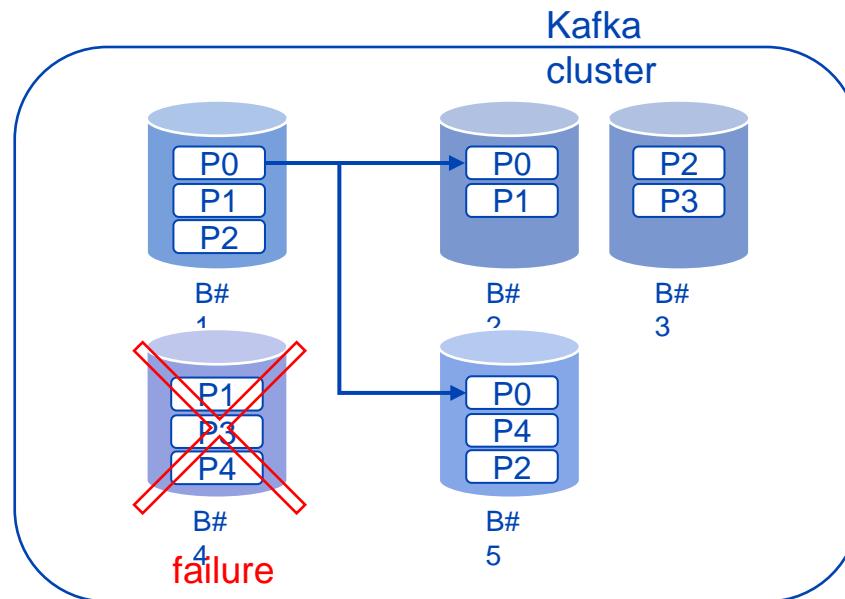
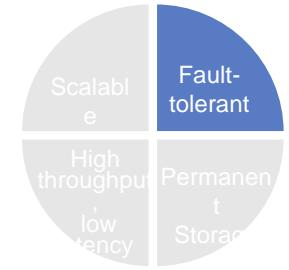
Why use Apache Kafka? (1/4)

- Scalable
 - Continuous scale-in / out is possible according to the load that increases or decreases the amount of data.
 - Start with a small number of servers
 - More and more servers can be added according to the load



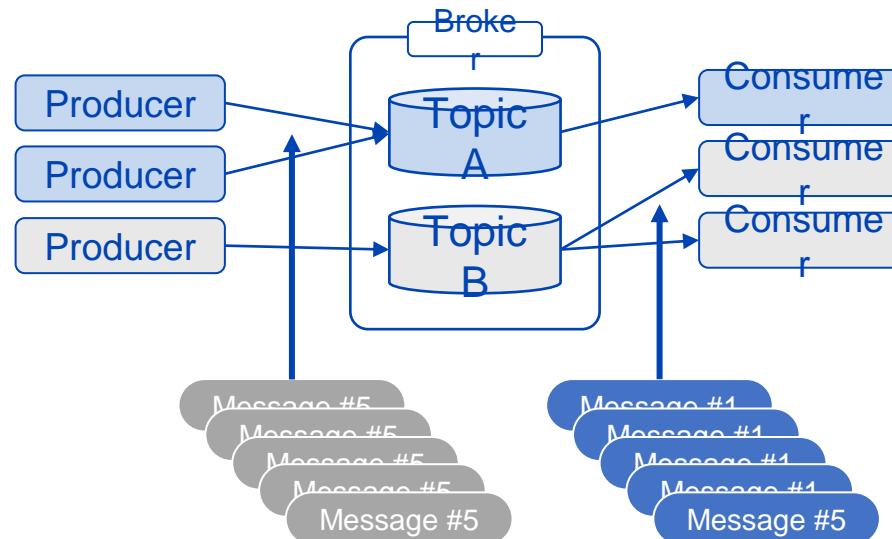
Why use Apache Kafka? (2/4)

- Fault-tolerant
 - High availability
 - Sustainable data processing even if some brokers have problems



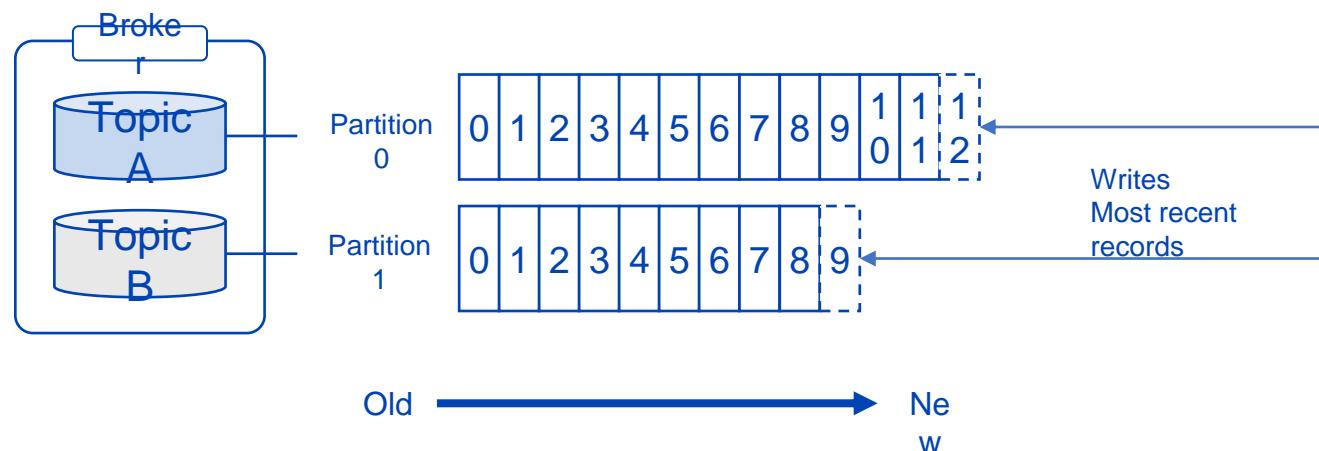
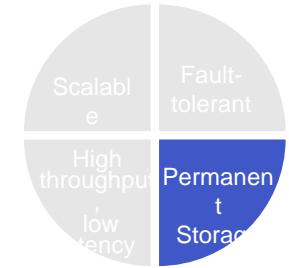
Why use Apache Kafka? (3/4)

- High throughput, low latency
 - Collective (batch) transfer of data as consumers / producers communicate with brokers
 - It is possible to increase the parallel throughput by increasing the number of consumers by the number of partitions.



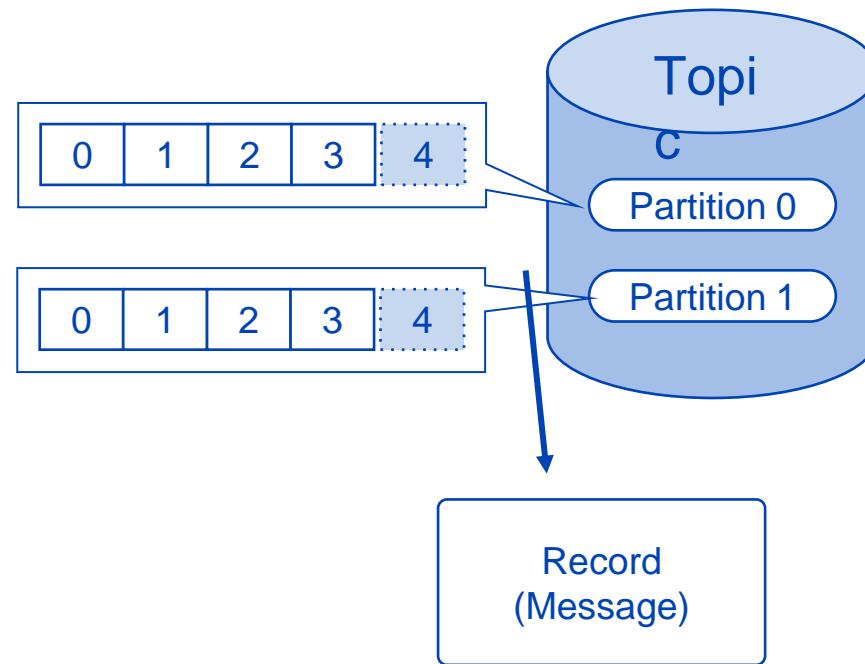
Why use Apache Kafka? (4/4)

- Permanent Storage
 - The messages are stored in the order they were sent and are immutable.
 - The retention period can be set for each topic.



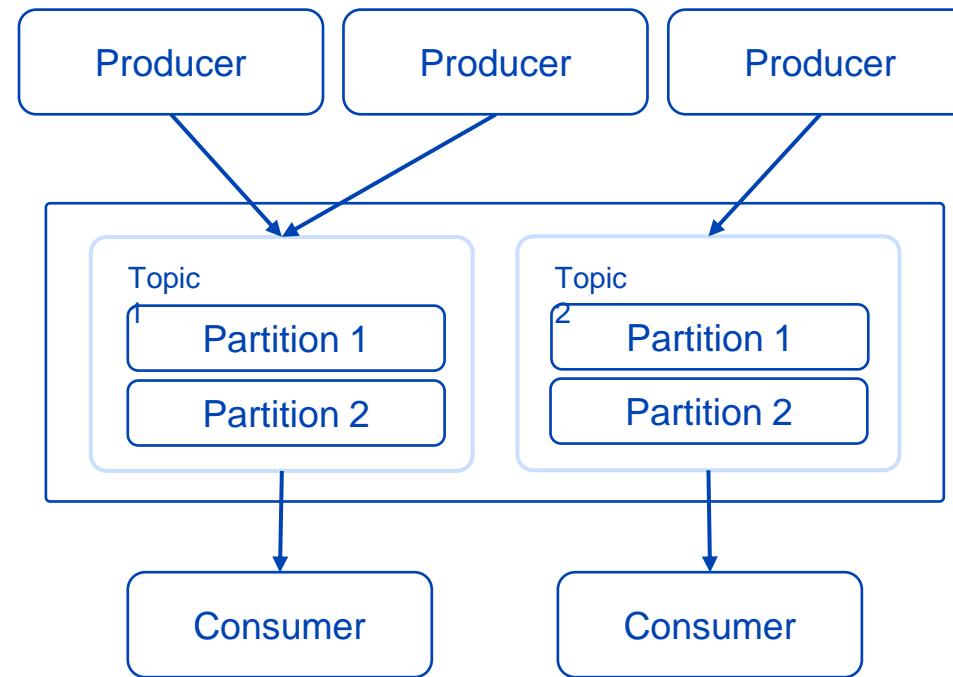
Basics of Kafka

- Messages are organized into topics
- Producers push messages
- Consumers pull messages
- Kafka runs on a cluster
- Nodes in the cluster are brokers



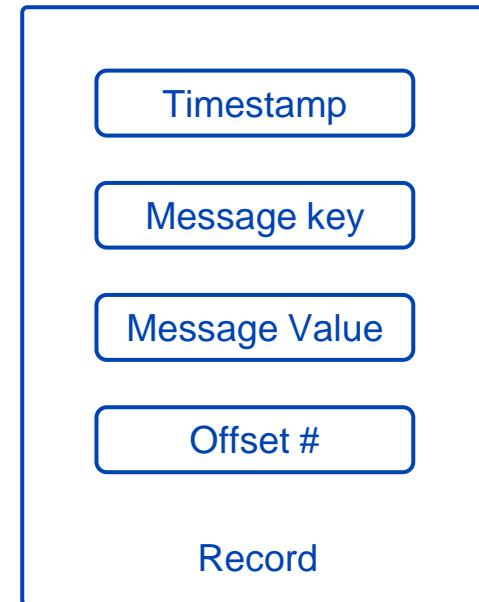
Brokers

- A broker is a 'server' unit on which Kafka is installed.



Messages (1/2)

- Messages are organized into topics
 - Topics can be partitioned
 - Ordering is only guaranteed within a partition
- Messages are variable-size byte arrays
 - Think of them as a row or record in a database
 - Each record consists of a key, value, and a timestamp
- There is no limitation on message size but a few KB's is good
- For efficiency, messages are written in batches
 - Amortize the cost of communication (network, setup, etc)

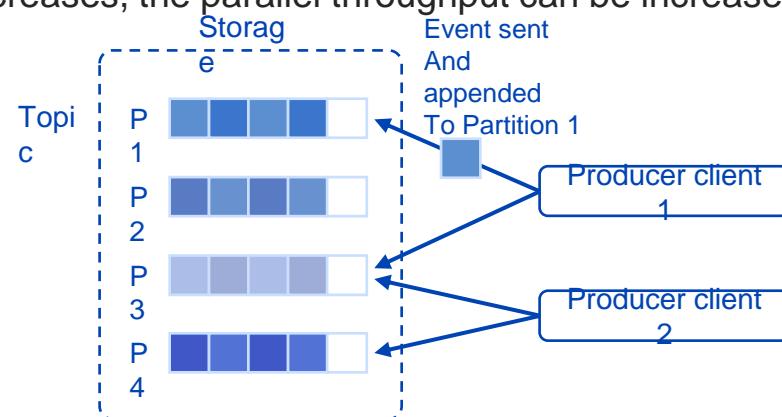


Messages (2/2)

- It is recommended to impose some additional structure to messages in the form of schema
 - JSON, XML are easy to use and human legible
 - Apache Avro provides a serialization mechanism including compression
- A consistent data format is important for efficient decoupling of producers and consumers
- Messages are retained whether they have been read or not
- Kafka uses two parameters to determine message retention
 - Configurable retention period - log.retention.{ms, minutes, hours}
 - Configurable size limit – log.retention.bytes
 - Which ever limit is reached first triggers removal of message

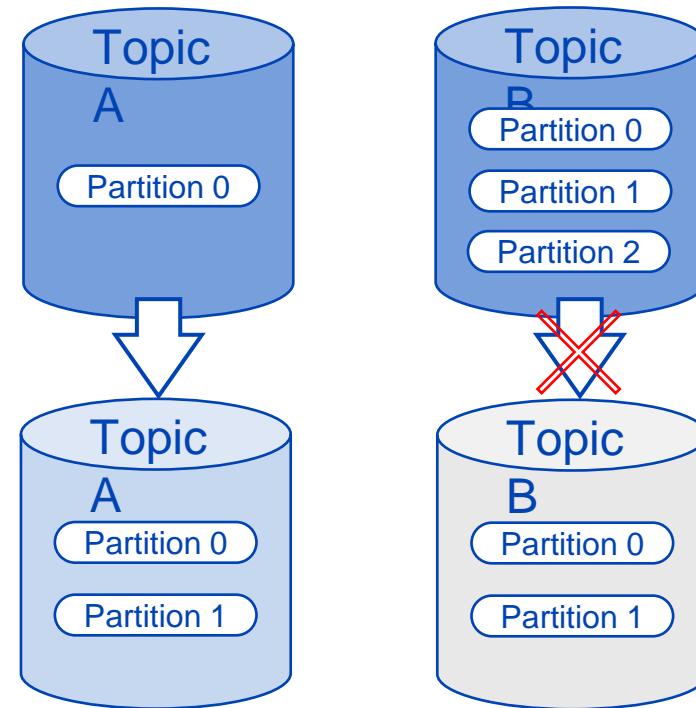
Topics & Partition

- A topic is a category or feed name to which messages are published
- Topics may have multi-subscribers - zero, one, or many consumers
- Topics are split into partitions - an ordered, immutable sequence of records that is continually appended to (structured commit log)
- Records in partitions have a sequential id number – *offset*
- As the number of partitions increases, the parallel throughput can be increased



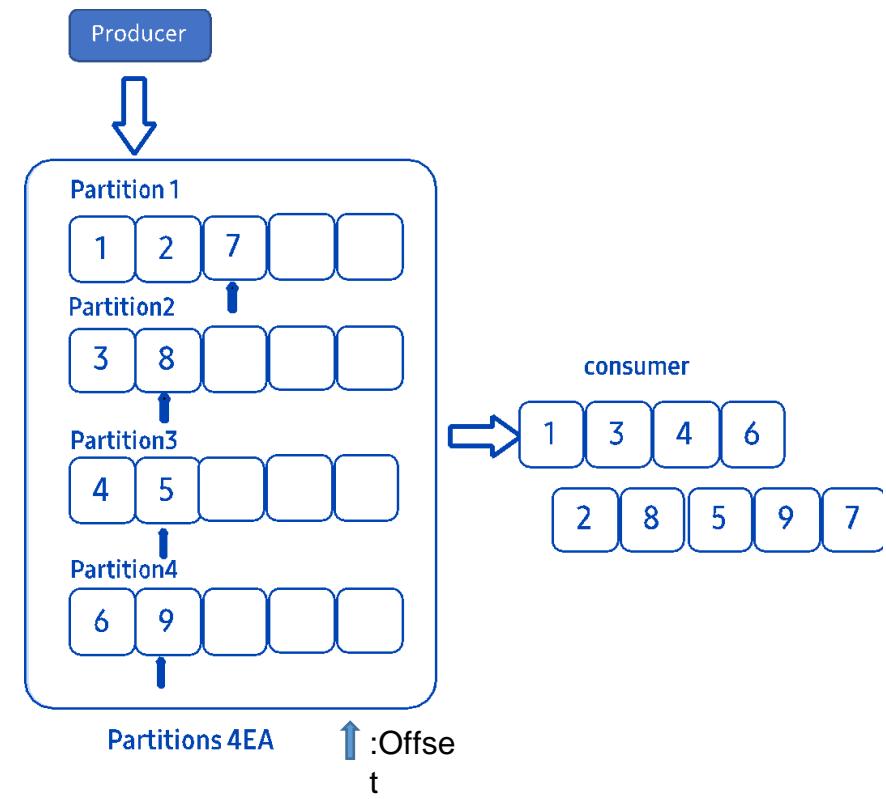
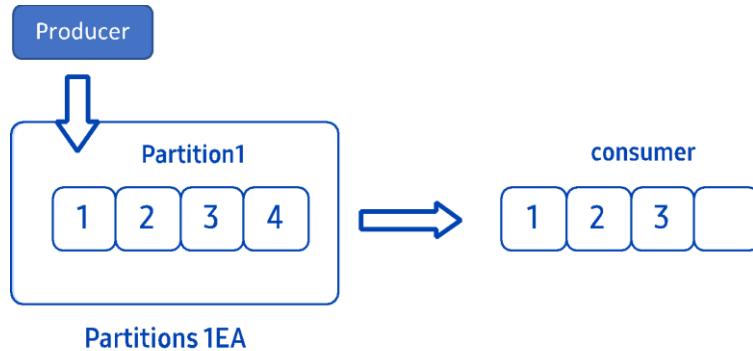
Change the number of partitions

- The number of partitions can be increased, however once expanded, cannot be reduced.



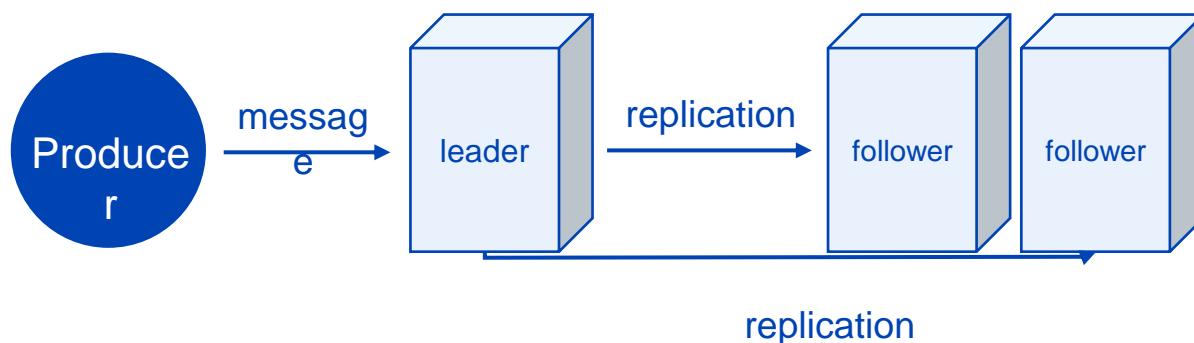
Message order

- Messages are ordered within a partition
 - There is no ordering amongst partitions
- A special case is when there is only a single partition
 - Order of messages for the entire Topic is now preserved



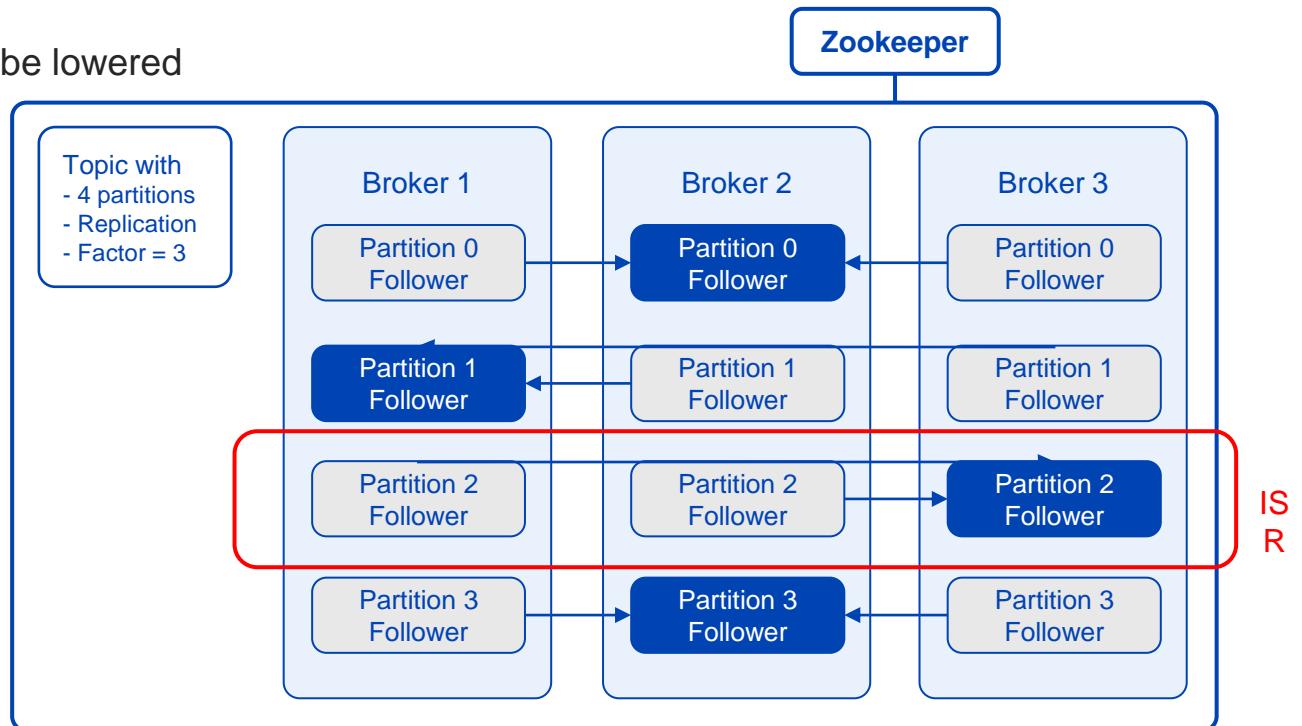
Topics Replication (1/2)

- Topics can and should be replicated.
- The unit of replication is the partition
- Each partition in a topic has 1 leader and 0 or more replicas
- A replica (follower) is deemed to be “in-sync” if
 - The replica can communicate with Zookeeper
 - The replica is not “too far” behind the leader (configurable)



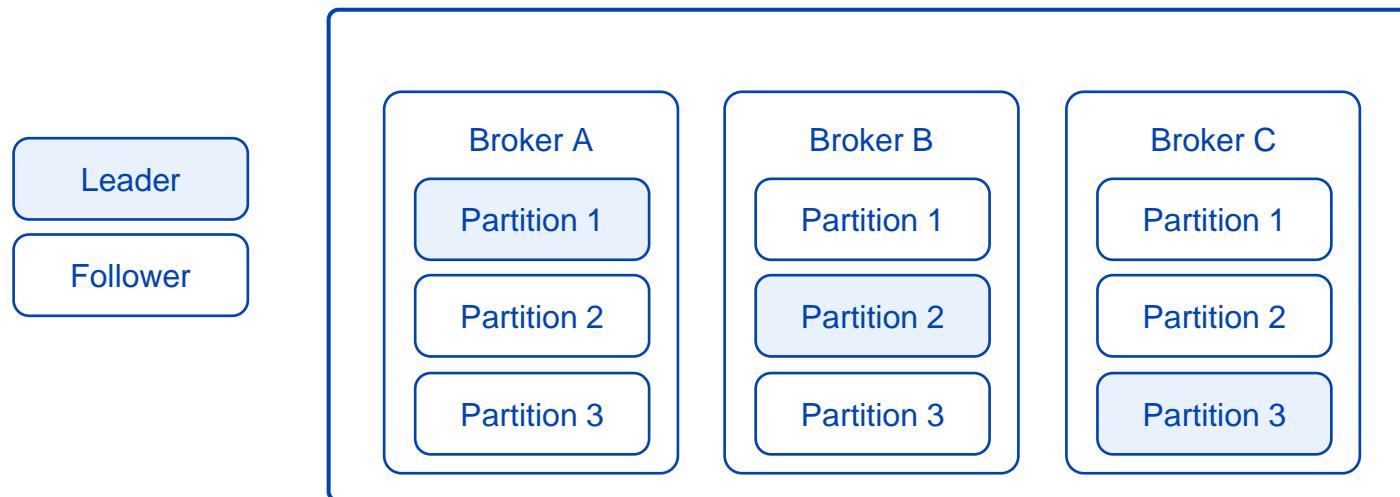
Topics Replication (2/2)

- ISR(In-Sync Replicas)
 - The group of in-sync replicas for a partition
- The Replication factor cannot be lowered



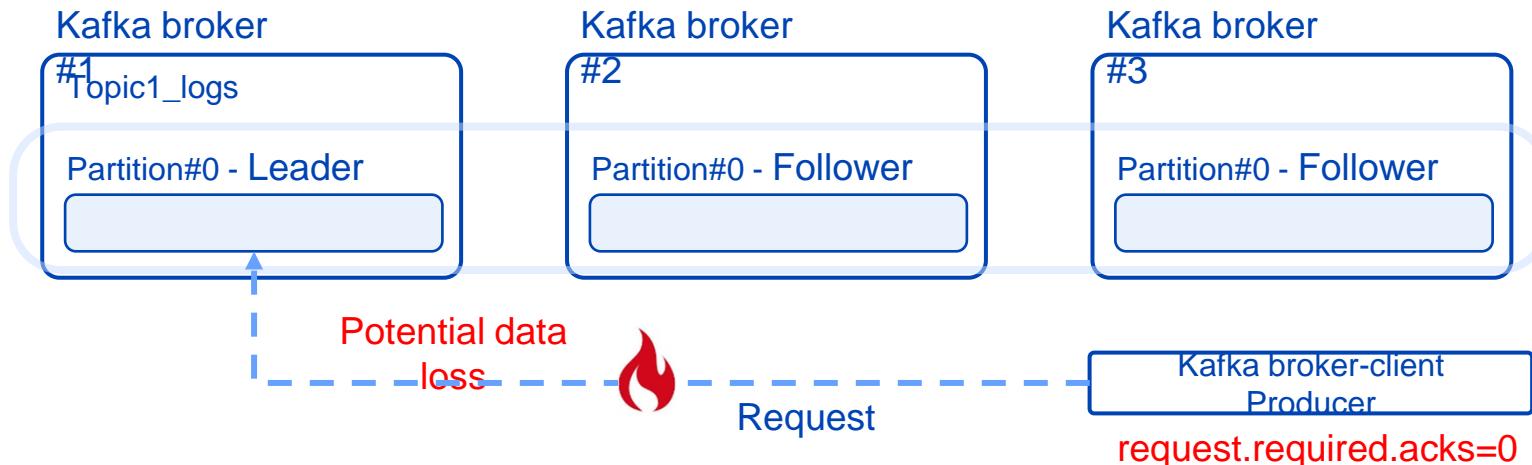
Distribution

- The partitions of the log are distributed over the brokers
- The leader of a partition handles all read and write request for that partition while followers passively replicate the leader
- If leader fails, one of the followers automatically becomes leader
- Each broker acts as a follower on some partitions and followers on others



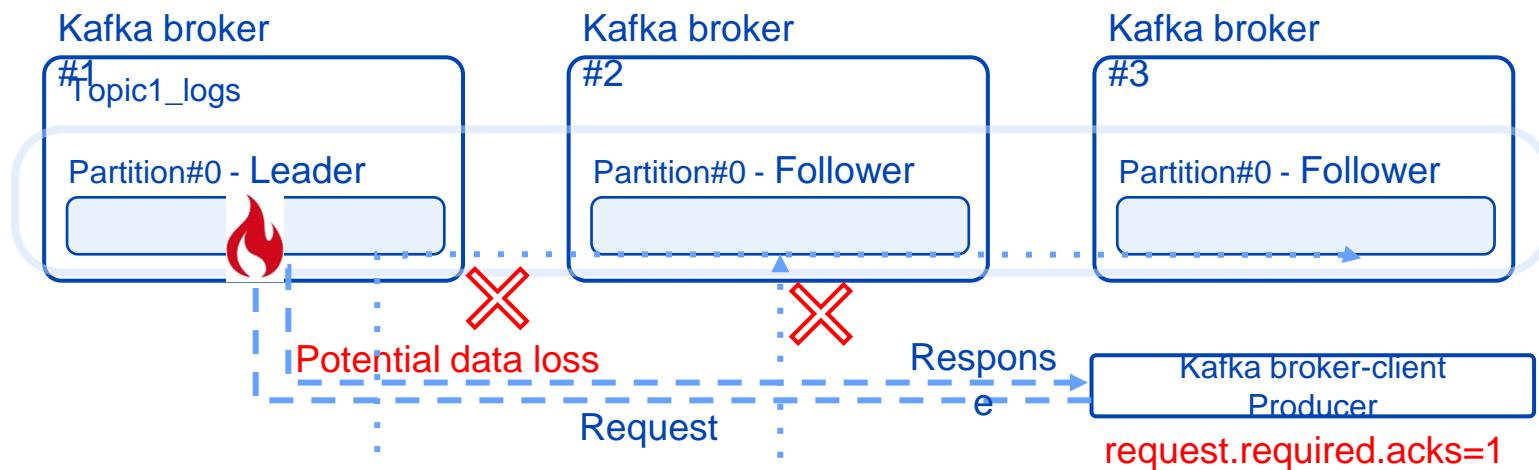
Topics Replication Ack (1/3)

- Durability can be configured with the producer configuration `request.required.acks`
 - 0 The producer never waits for an ack
 - 1 The producer gets an ack after the leader replica has received the data
 - -1 The producer gets an ack after all ISRs receive the data
- `request.required.acks = 0`



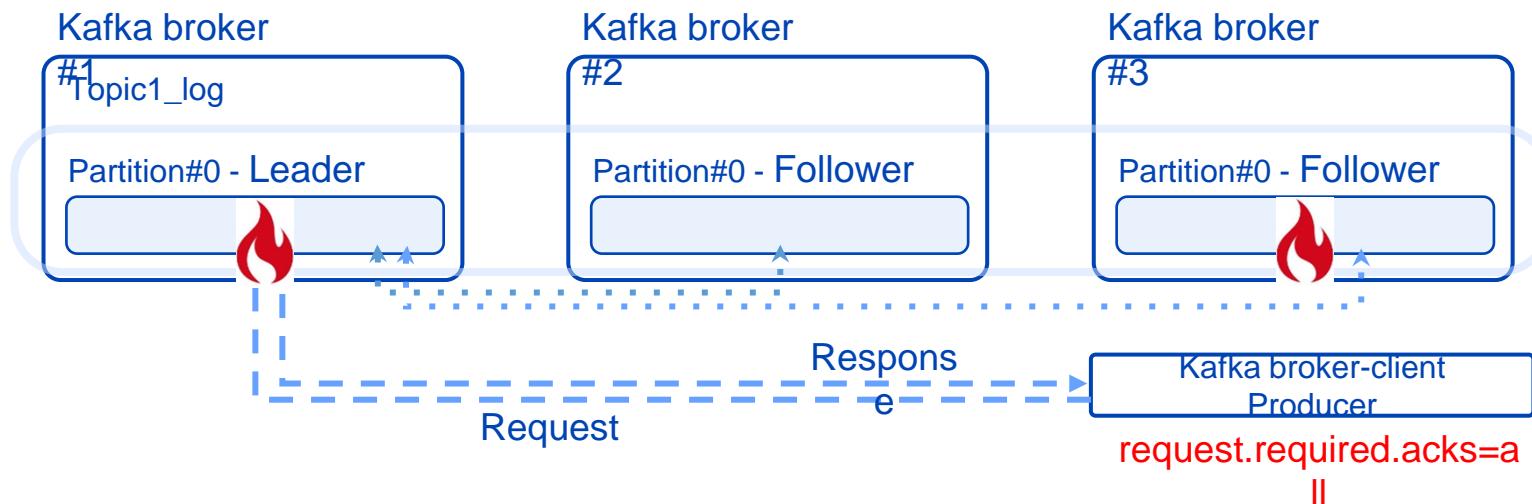
Topics Replication Ack (2/3)

- `request.required.acks = 1`



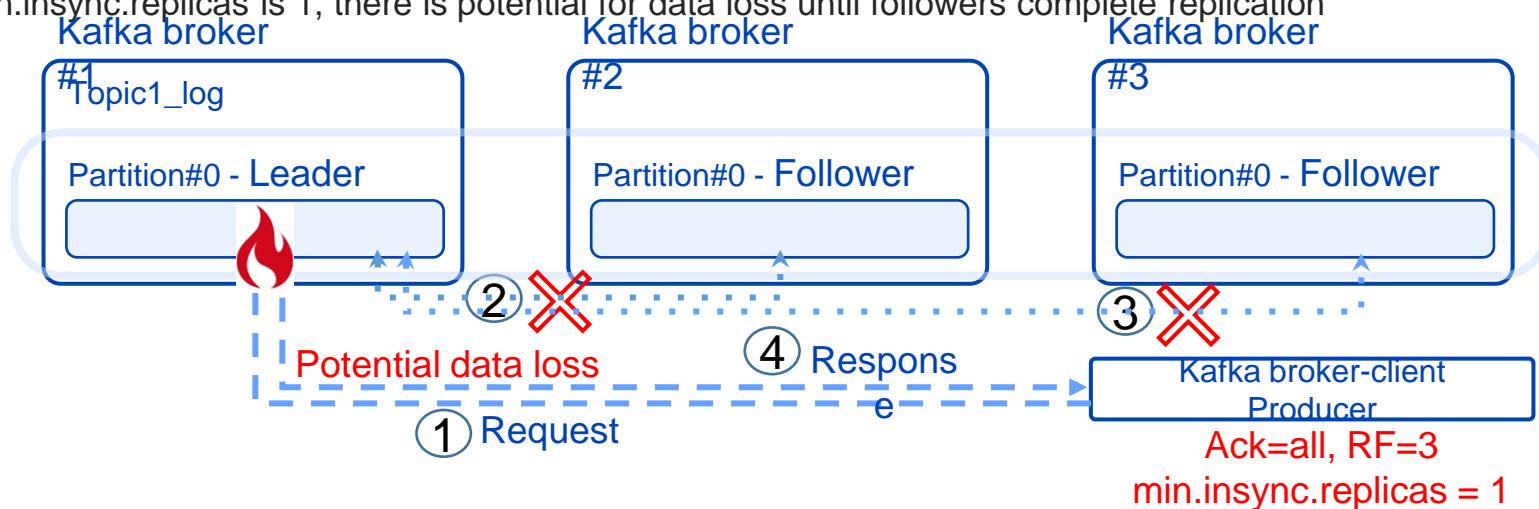
Topics Replication Ack (3/3)

- `request.required.acks = All (-1)`
- With RF = N, can handle N-1 losses



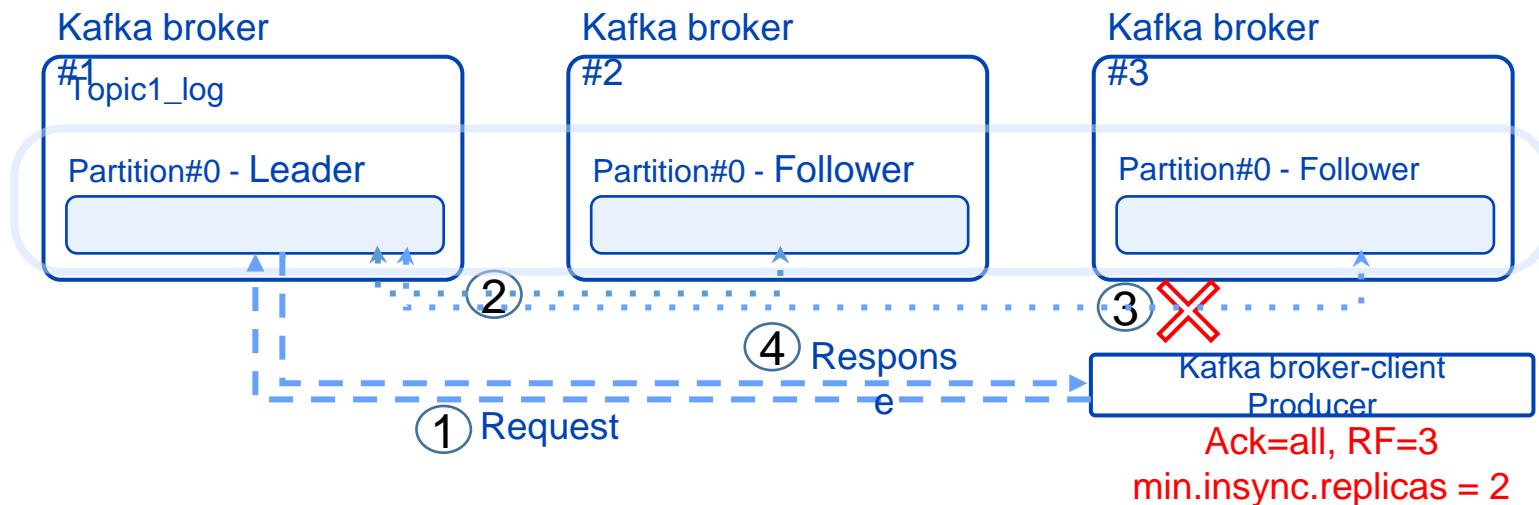
Minimum In-Sync Replica (1/2)

- The minimum number of in-sync replicas specify how many replicas that are needed to be available for the producer to successfully send records to a partition.
 - request.required.acks =all, using the min.insync.replicas property
 - The number of replicas in your topic is specified by you when creating the topic.
 - If min.insync.replicas is 1, there is potential for data loss until followers complete replication



Minimum In-Sync Replica (2/2)

- Ack = All (-1)
- Replication Factor = 3
- min.insync.replicas=2



Topics Naming

- Do not mix uppercase and lowercase letters
 - More prone to human error
- Do not mix period and underscore
 - Mixing periods and underscores can cause conflicts with named used in Kafka.
- Kafka cluster administrator to establish and disseminate rules
 - Set rules such as <team name>-<application name>-<usage>-<environment>

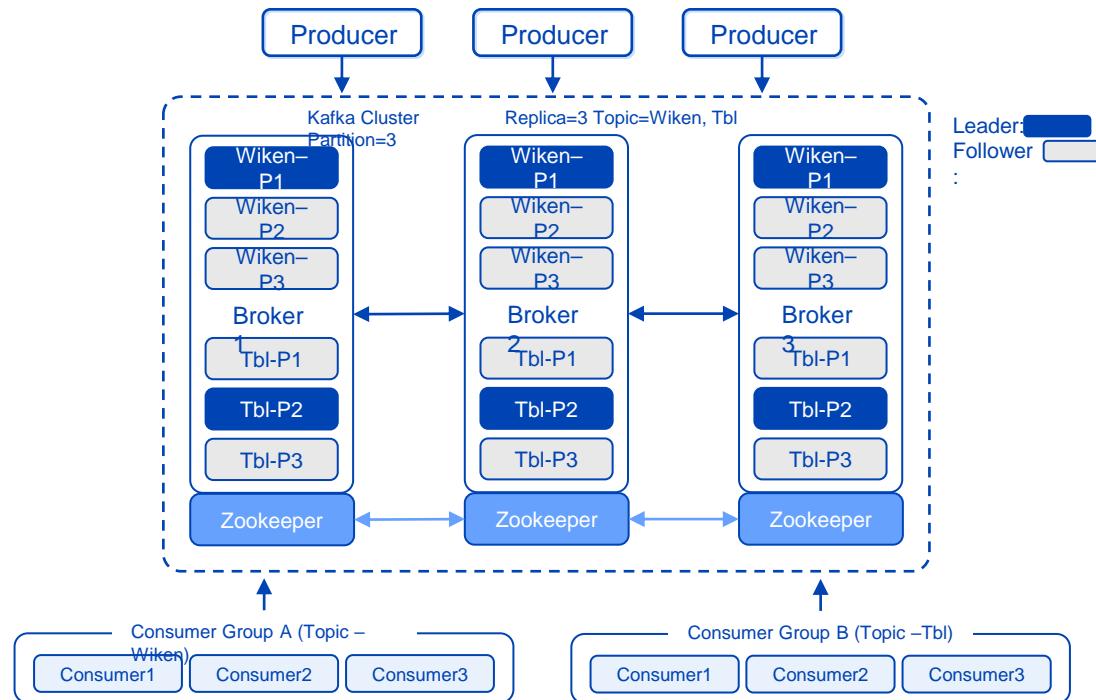
Unit 4

Apache Kafka

- | 4.1. Apache Kafka Fundamentals
- | 4.2. Producers and Consumers

Kafka Architecture Overview

- Kafka clusters consist of one or more servers running the Kafka broker daemon (Brokers)
- Kafka depends on Apache Zookeeper service for coordination

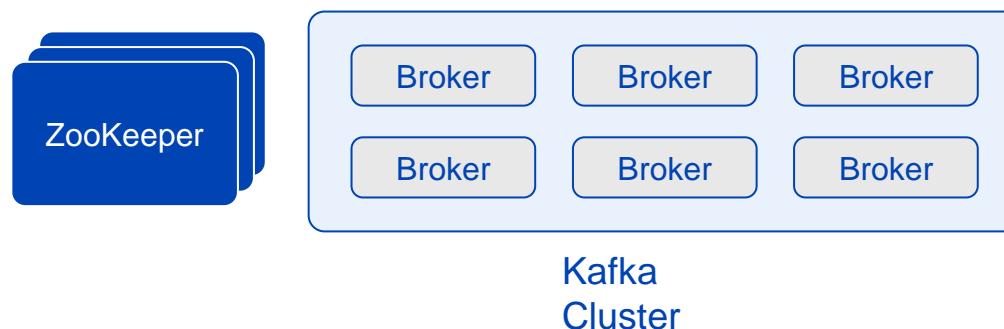


Kafka Brokers

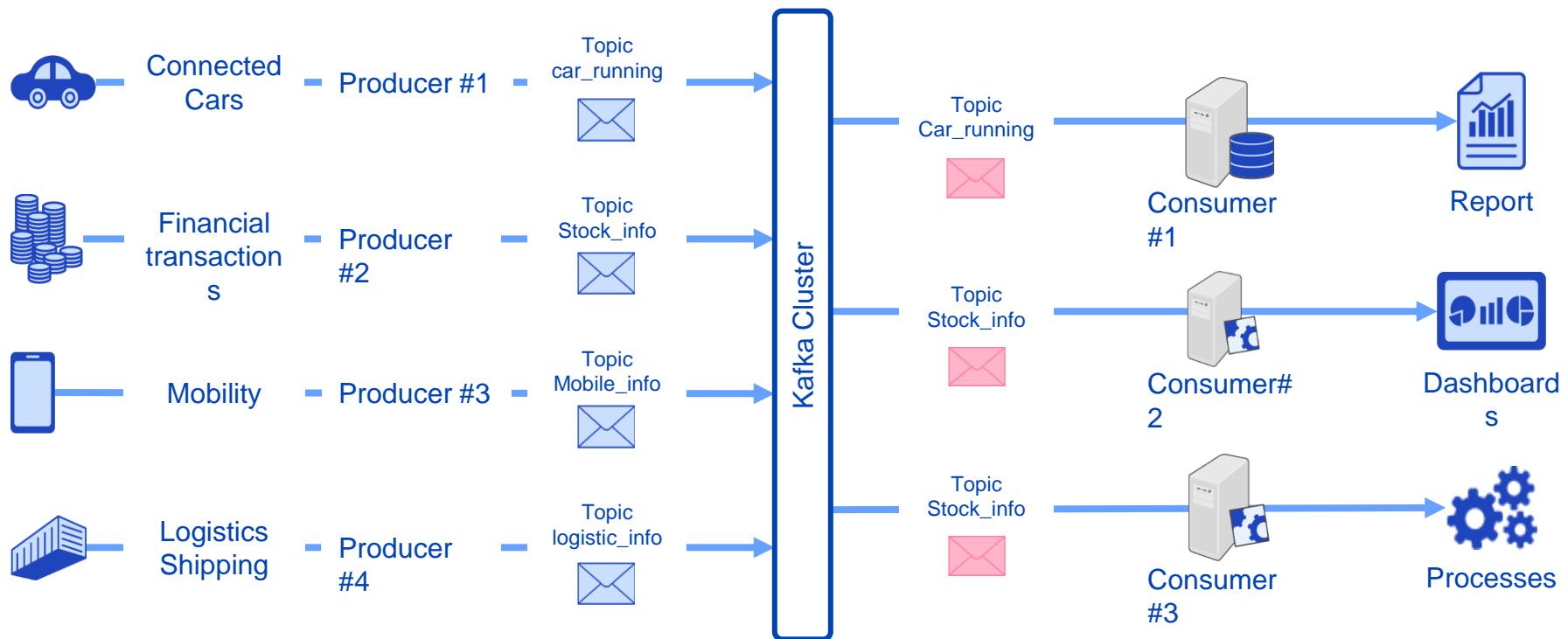
- Brokers are also called Kafka servers.
- It plays an important role in receiving the message generated by the producer, managing the offset, and responding to the request to read the message from the consumer.
- This broker can have multiple brokers in one cluster.
 - As the number of brokers increases, the throughput within a unit time can increase, so it can respond to large amounts of data.
- In Kafka, the broker recommends creating at least three.
 - This is the minimum number to keep your data safe.
 - There is one special broker, which is the Controller.
 - Zookeeper chooses one of the brokers as its controller.

Apache Zookeeper

- Apache Zookeeper is a coordination service for distributed applications
- Kafka uses Zookeeper to maintain metadata of brokers that are members of the cluster.
- Kafka uses ZooKeeper to detect the addition or removal of brokers and consumers, as well as to trigger rebalancing in consumers when brokers or consumers are added or removed.
- Kafka uses Zookeeper to keep track of brokers running in the cluster
- Zookeeper also detects the addition or removal of consumers

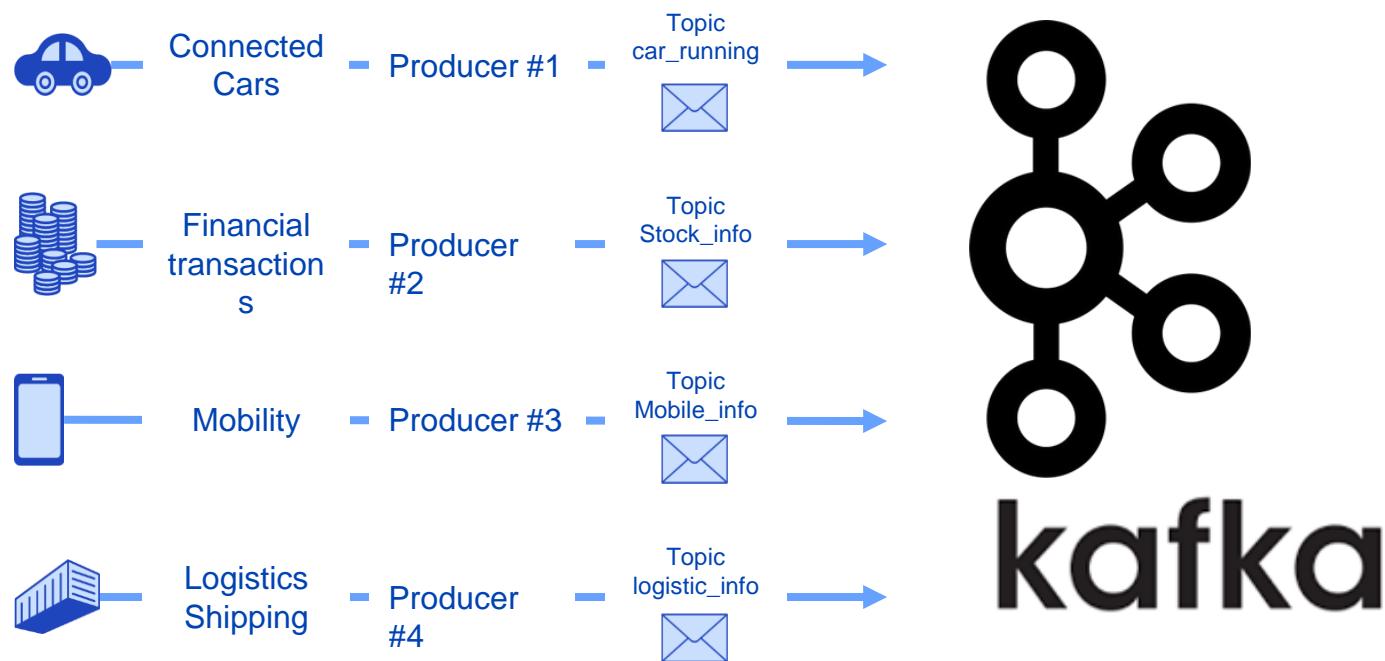


Producers and Consumers



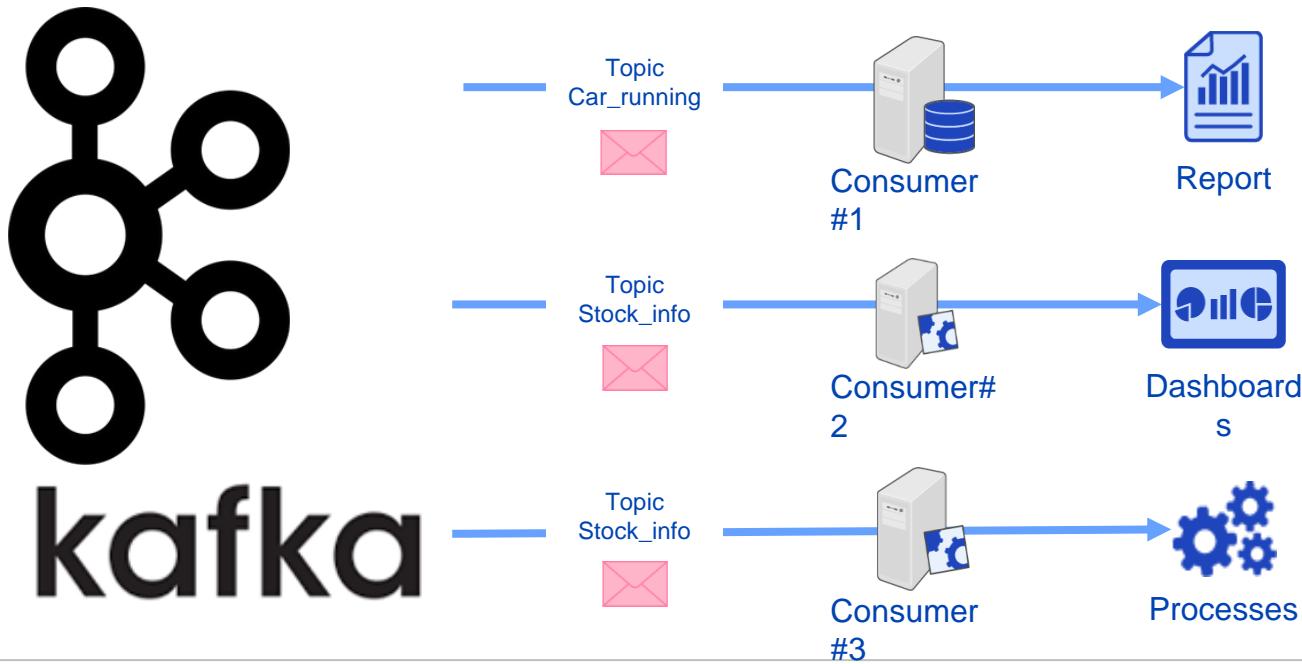
Producers

- This is the Kafka client of generating data and pushing it into a Kafka Cluster.
- Processes that write messages to a particular topic on Kafka
- Producers communicate with Broker and not with Consumers



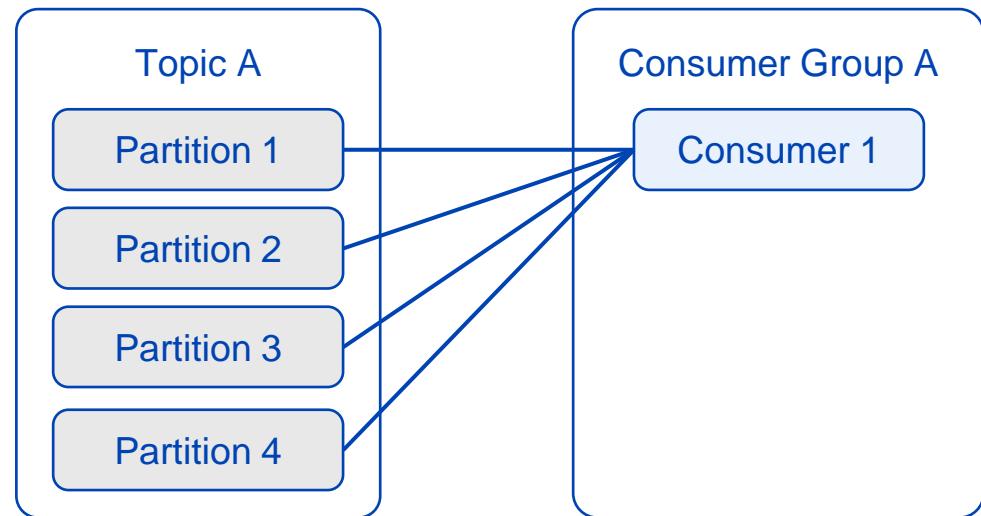
Consumers

- Consumers are Kafka clients that fetch messages from a topic
 - They pull the broker for messages
- the Consumer directly transmits the Message from the Kafka where the Partition is located.



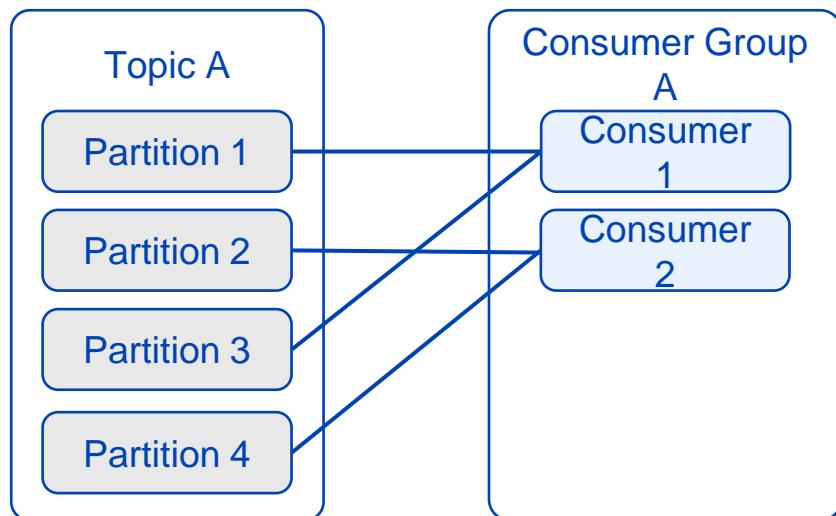
Consumer Group

- Consumers label themselves as being part of a Consumer Group
- Consumers are associated with a particular partition while Consumer Groups are linked to an entire topic.
 - Consumer Groups consume topics
 - Consumer consumes partition.
- Purpose of a Consumer Group
 - Increase topic read throughput (Scale out)
 - Rebalancing
 - Offset sharing

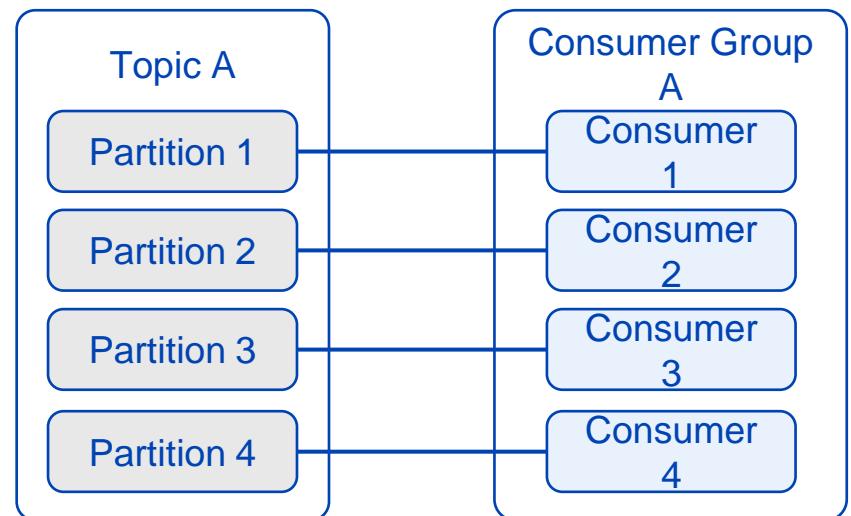


Consumer Group – Scale out

- Increase topic read throughput (Scale out)



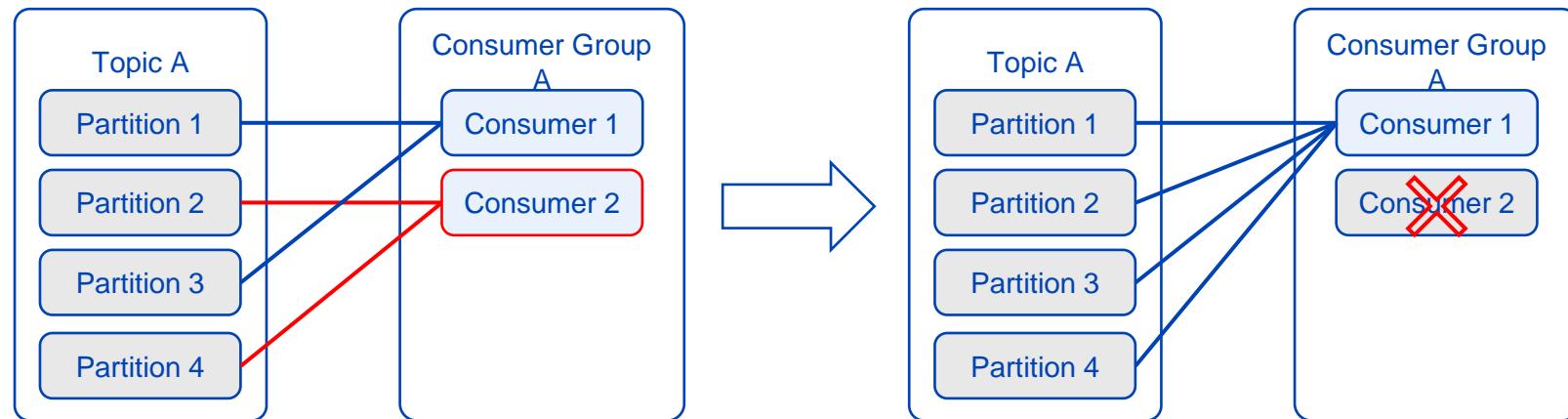
Add :
Consumer2



Add :
Consumer2,3,4

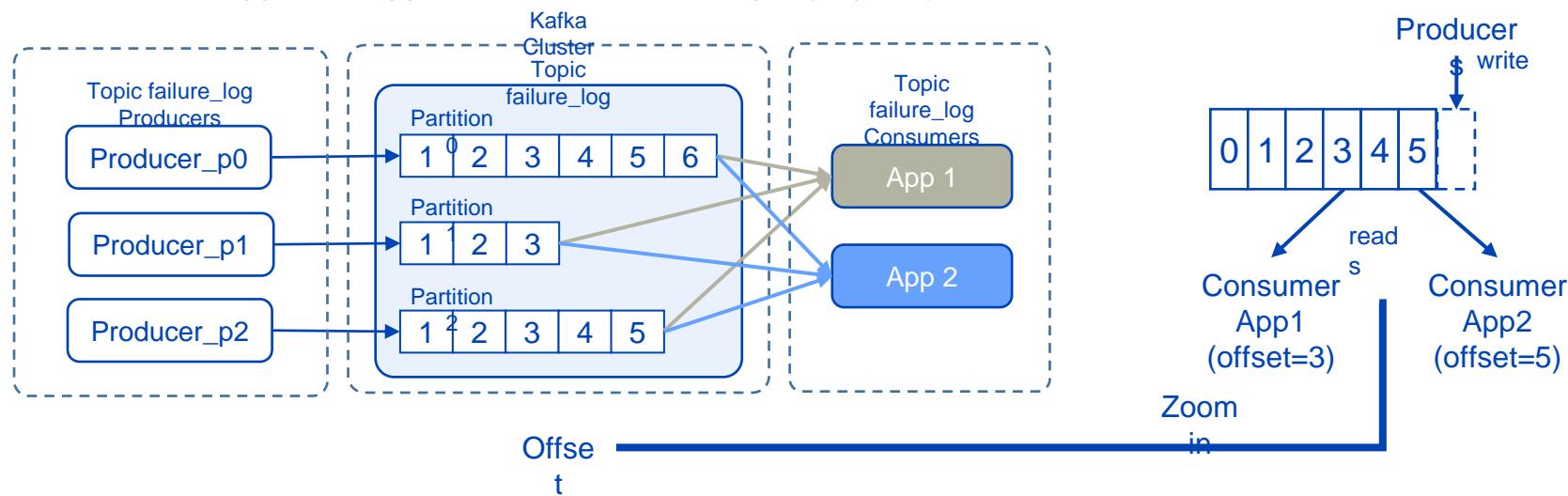
Consumer Group - Rebalancing

- In the previous page, where 4 partitions were handled by one consumer, a bottleneck occurred and one(three) consumer was added.
 - After additional consumers are added, division of partition ownership is performed.
 - This process is called rebalancing.
- This can also occur if a consumer becomes unavailable, such as in a failure situation.



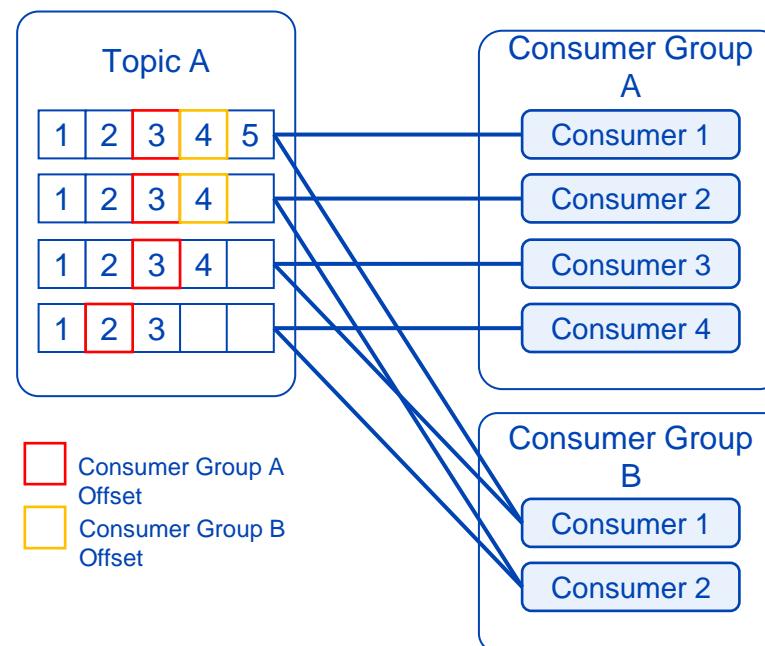
Consumer Group - Offset Sharing

- Offset Sharing
 - Kafka has an offset for each record.
 - Each consumer and consumer group tracks the current offset independently of each other
- Multiple consumer groups can subscribe to the same topic at the same time
 - Consumer app1 and app2 can subscribe one topic (Topic A) at the same time.



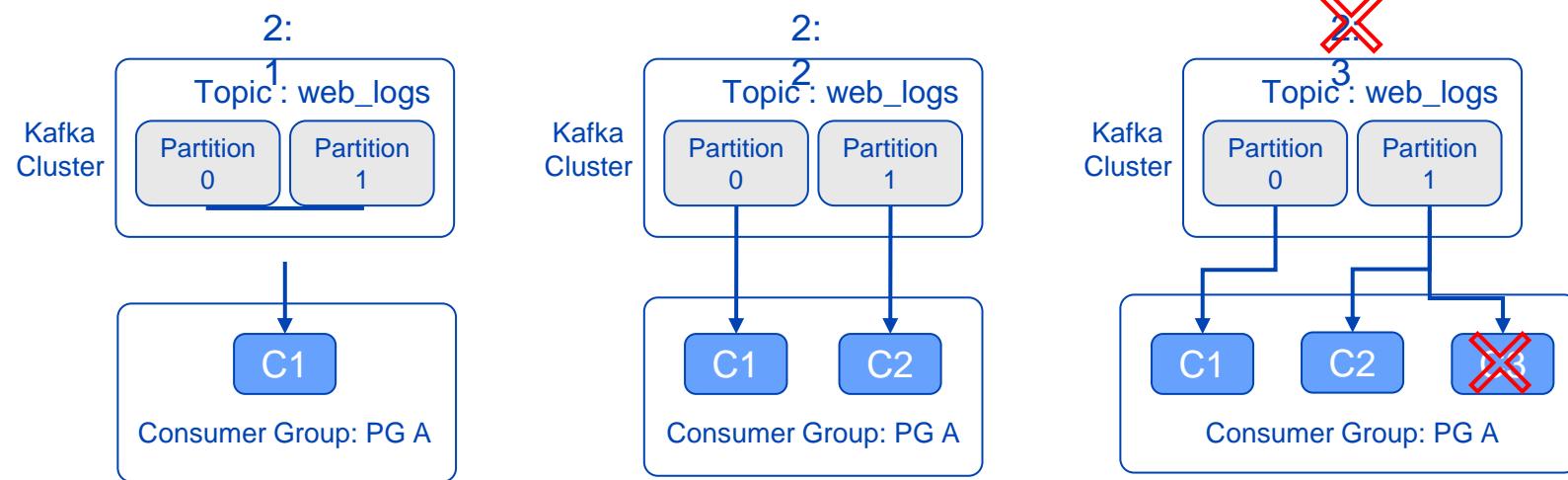
Consumer Group

- Using Multi-Consumer Group
- Consumer Group A and Group B store and manage separately the offset information.
 - Consumer Group A offset – 4, 4, 3, 2
 - Consumer Group B offset – 3, 3, 3, 2



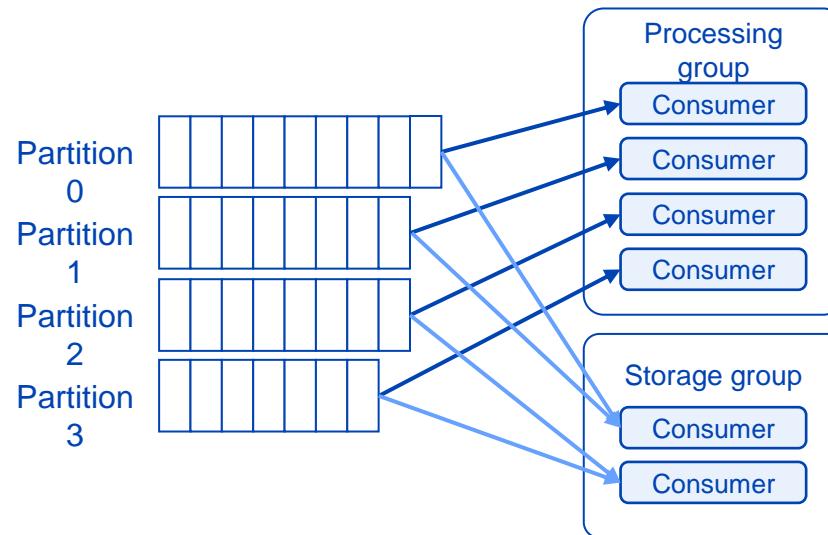
Increasing Consumer Throughput

- Additional consumers can be added to scale consumer group processing
- Consumer instances that belong to the same consumer group can be in separate processes or on separate machines



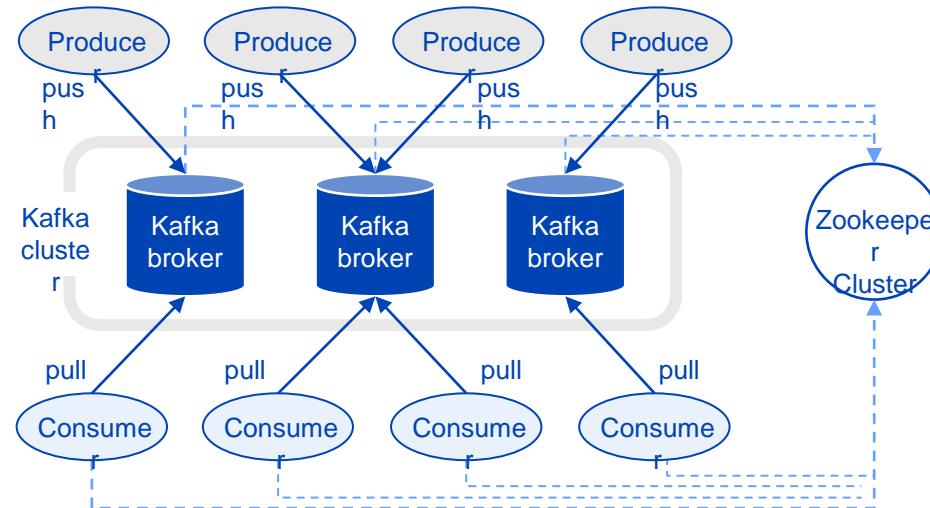
Multiple Consumer Groups

- Kafka can scale to large number of consumer groups and consumers
- A partition in a Topic is delivered to only one instance of a consumer within a consumer group
- Each message published to a topic is delivered to one consumer instance within each subscribing consumer group



Kafka – Storage System

- Producers' records are persisted to disk in a commit log
- Consumers are decoupled from producers and messages are retained, effectively acting as a storage system
- Kafka stores messages to disk and makes N copies for fault tolerance. Therefore, it can be used as an efficient storage system by itself.



Kafka – Stream Processing

- Kafka can be used for real-time data processing purposes.
- Kafka's stream processor can read and process a continuous data stream from an input topic and output the result to the topic stream again.
- Kafka provides several APIs to facilitate all of this
 - Producer and Consumer API
 - Streams API for powerful transformation

Available Producers and Consumers

- Tools available in Kafka
 - Command-line producers and consumers
 - Producer and Consumer Java APIs
- Large number of third-party APIs
 - Available in many languages such as Python, PHP, C/C++, .NET, Ruby
- Integration with other tools and projects
 - Apache Flume
 - Apache Spark
 - Amazon AWS
- Kafka's ecosystem is growing very fast

Kafka Topics from the command line

- Kafka include several command line tools to explore and experiment with Kafka
- kafka-topics creates a Kafka topic from the command line
 - Name of topic
 - Bootstrap-server connection string

```
$ kafka-topics --create \  
  --bootstrap-server localhost:9092 \  
  --replication-factor 3 \  
  --partitions 3 \  
  --topic topic1_logs
```

Kafka Topics from the command line

- Topics List

```
$ kafka-topics --list \  
--bootstrap-server localhost:9092
```

- The details of a topic

```
$ kafka-topics --describe \  
--bootstrap-server localhost:9092 \  
--topic topic1_logs
```

Create a Kafka Producer

- kafka-console-producer tool to create a producer that will take your keystrokes from the console and create messages from them
- Specify one or more brokers in the -- broker-list option

```
$ kafka-console-producer \
  --broker-list brokerhost1:9092, brokerhost2:9093 \
  -- topic topic1_logs
```

Create a consumer

- kafka-console-consumer tool will create a consumer
- Recall that Zookeeper is used to keep track of brokers as well as consumers
- This tool requires a Zookeeper connection string

```
$ kafka-console-consumer \  
  --bootstrap-server localhost:9092 \  
  --topic topic1_logs \  
  --from-beginning
```

Produce messages

- It is often convenient to use UNIX pipes or redirects to push data into the producer just created
- Here, we output a text file with cat and pipe it in to the producer
 - Each line in the file becomes a message

```
$ kafka-console-producer \  
  --broker-list brokerhost1:9092, brokerhost2:9093 \  
  -- topic topic1_log
```

(Type the some messages)

Hello everyone.

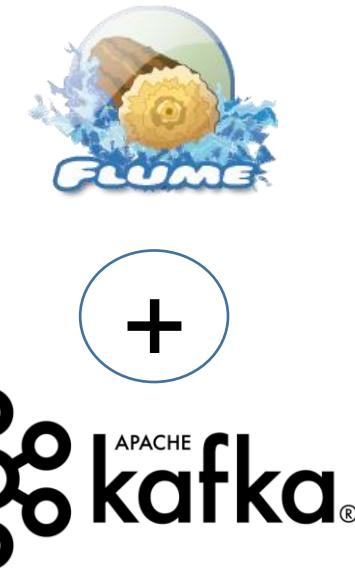
I hope you are enjoying kafka.

Flume or Kafka?

- Flume is efficient at moving data from a single source into Hadoop
 - Offers sinks that write to HDFS, HBase, Kudu table, or a Solr index
 - Easily configured to support common scenarios, without writing code
 - Can also process and transform data during the ingest process
- Kafka is a publish-subscribe messaging system
 - Offers more flexibility than Flume for connecting multiple systems
 - Provides better durability and fault tolerance than Flume
 - Often requires writing code for producers and/or consumers
 - Has no direct support for processing messages or loading into Hadoop

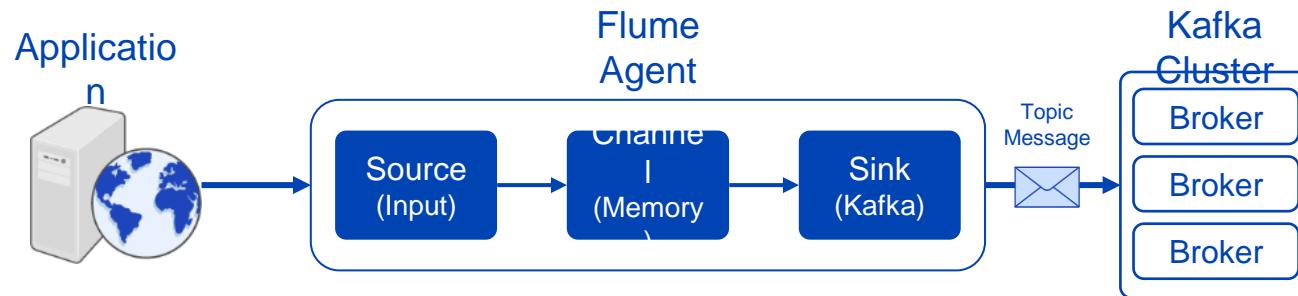
Flafka = Flume + Kafka

- Both systems have pros and cons
- Integrate them into a complementary solution
 - Take advantage of the strengths of Kafka
 - Take advantage of the ease-of-use of Flume
- Flume provides Kafka Source, Kafka Channel, & Kafka Sink
 - This allows using Flume agents to receive to send messages to Kafka



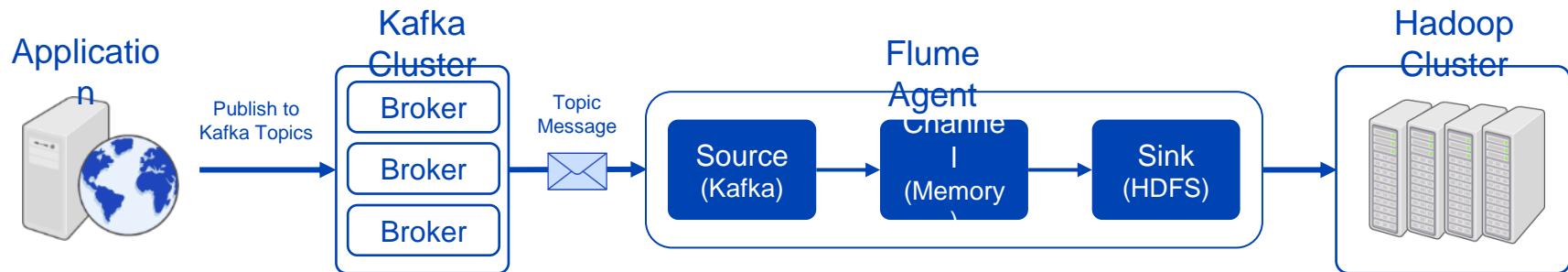
Flume as Kafka Producer

- Configuring Flume with Kafka Sink allows Flume to act as a Kafka Producer
 - Application sends data through a Flume dataflow
 - Event data flows through the memory channel
 - The Kafka sink publishes these events to a Kafka Topic



Flume as Kafka Consumer

- Configuring Flume with Kafka Source allows Flume to act as a Kafka Consumer
 - Kafka Source is configured to read desired Topic from Kafka
 - The data is then buffered through the memory channel
 - Finally, the sink writes this data into HDFS



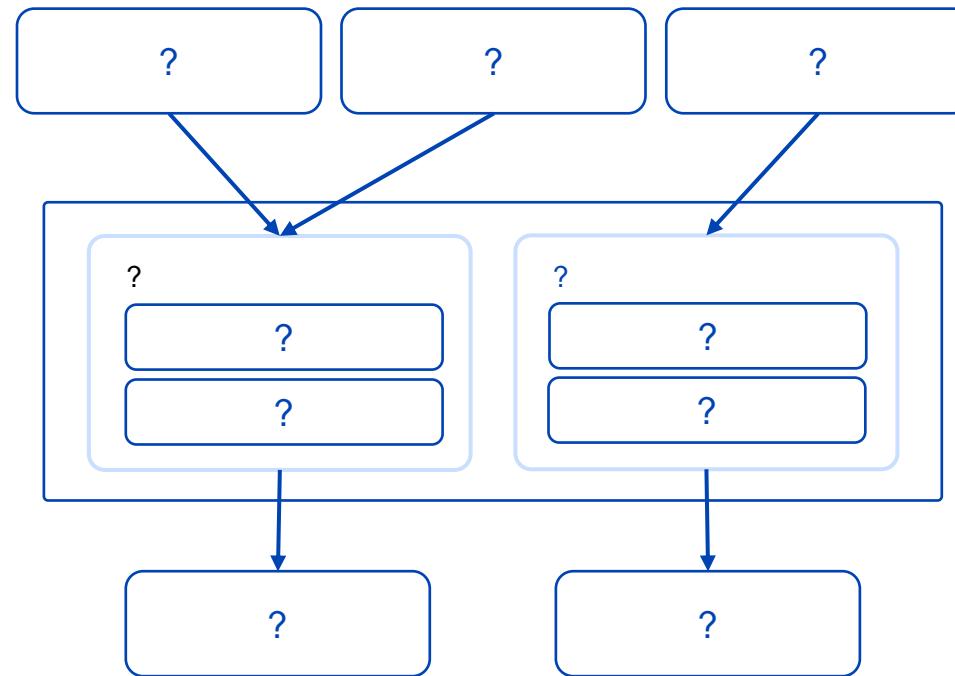
Flume using Kafka Channel

- A Kafka channel can be used with any Flume source or sink
 - Kafka provide much better scalability, reliability, availability and fault-tolerance compared to memory or file channels
- The Kafka channel can be used for multiple scenarios:
 - With Flume source and sink - it provides a reliable and highly available channel for events
 - With Flume source and interceptor but no sink - it allows writing Flume events into a Kafka topic, for use by other apps
 - With Flume sink, but no source - it is a low-latency, fault tolerant way to send events from Kafka to Flume sinks such as HDFS or Hbase.
- Key properties of the Kafka channel
 - Type: component type name
 - Kafka.bootstrap.servers: List of brokers in the Kafka cluster used by the channel
 - Kafka.topic: Kafka topic which the channel will use. The default value is flume-channel.
 - Kafka.consumer.group.id: Consumer group ID the channel uses to register with Kafka.

Apache Kafka Review

[Quiz/Discussion]

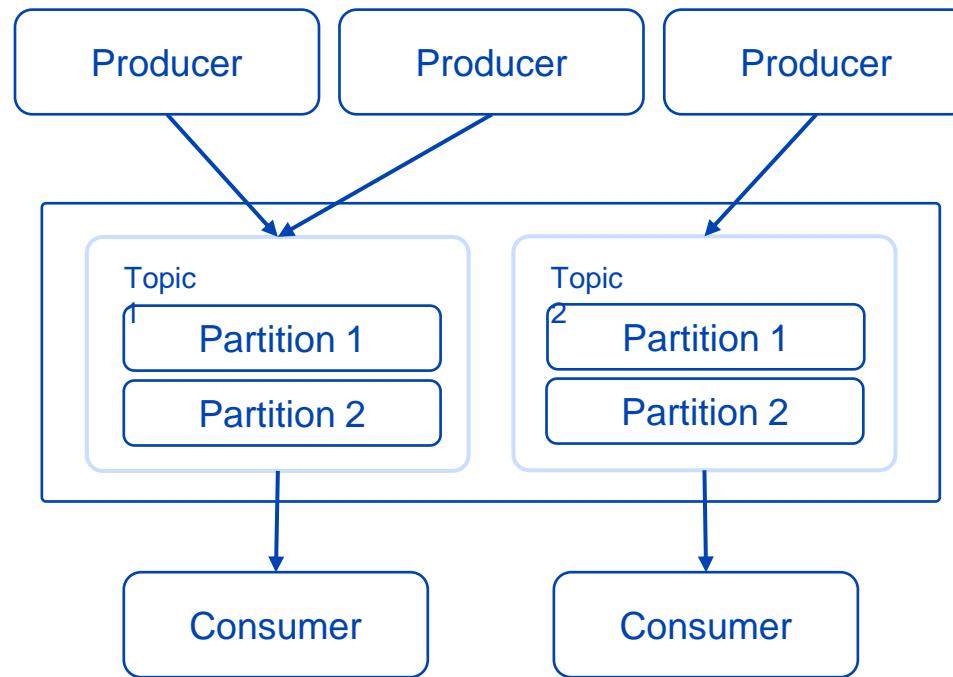
- How was it look like and what was in it?



Apache Kafka Solution

[Quiz/Discussion]

- How was it look like and what was in it?



[Lab10]

Creating a Kafka Topic, Producer, and Consumer



[Lab11]

Sending Messages from Flume to Kafka



Chapter 4.

Big Data Storage

Big Data Course

Chapter Description

Objectives:

- ✓ We will learn how to use other storage devices beyond HDFS
 - HDFS EC allows us to achieve the same level of fault-tolerance at about half the disk usage overhead
 - Kudu allows applications with both high throughput and fast latency characteristics
 - Moving beyond, on premise storage, there are many public cloud storage options available
- ✓ We will learn what NoSQL is and how it differs from conventional relational database management systems
 - Apache HBase is one of the earliest NoSQL engines and tightly integrated with Hadoop
 - NoSQL engines are broadly categorized as "strongly consistent" or "highly available"
 - We will learn the basics of Apache Cassandra, a primarily a highly available NoSQL database
 - For strongly consistent, we will learn the basics of Apache MongoDB

Contents of Chapter:

1. Data Storage Alternatives
2. NoSQL

Unit 1.

Data Storage Alternatives

Big Data Storage

Unit 1.

Data Storage Alternatives

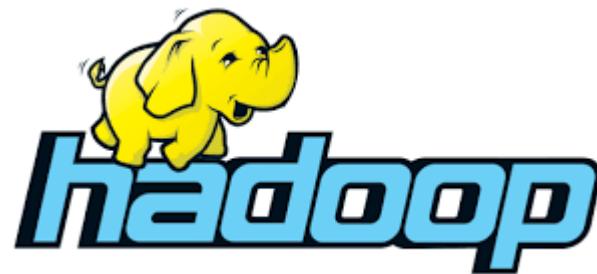
- | 1.1. On-Premise Storage
- | 1.2. Public Cloud Storage

When Hadoop was Young

- When Hadoop was introduced as a platform and methodology for working with and processing massive amounts of data, it did so by introducing two core functionalities
 - Hadoop Distributed File System for storage
 - MapReduce managed by YARN for compute
- At about the same time, Web 2.0, was well underway, and needed a new database model to handle the changing requirements
 - Second generation web sites that emphasized user-generated content
 - Shared content amongst participants and the general public
 - Ease of use features
 - NoSQL, a term was first coined in 1998, was proposed as a substitute for RDBMS to handle Web 2.0
 - Current NoSQL specifications matured around 2009

Hadoop HDFS

- Good at:
 - Efficiently scanning large amounts of data
 - Accumulating data with high throughput
 - Multiple SQL Options
 - All processing engines
- Not so good:
 - Single Row Access is problematic
 - Mutation is problematic
 - “Fast Data” access is problematic



Apache HBase - the Hadoop NoSQL

- Good at:
 - Efficiently finding and writing individual rows
 - Accumulating data with high throughput
- Not so good:
 - Scans are problematic
 - High cardinality access is problematic
 - SQL support is so/so due to the above



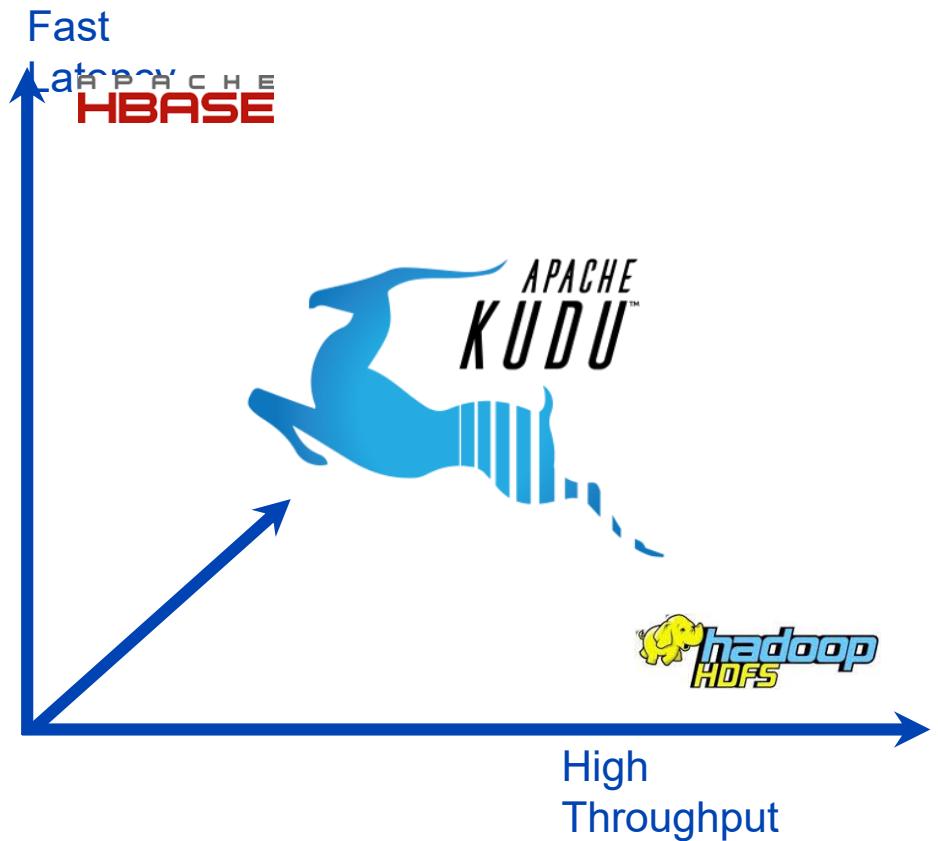
Hadoop Matures

- Hadoop matures and new use cases emerge
- Traditionally applications separated the processing of historical data and real-time data
 - Batch process historical data saved on HDFS storage
 - Access and modify random access datapoints in real-time using stream processing engines with data stored on NoSQL's such as HBase
- Lambda Architecture
 - Many use-cases require the analysis of historical data to be joined with up to date current streaming data
 - Lambda architecture proposes two separate applications and systems to process this



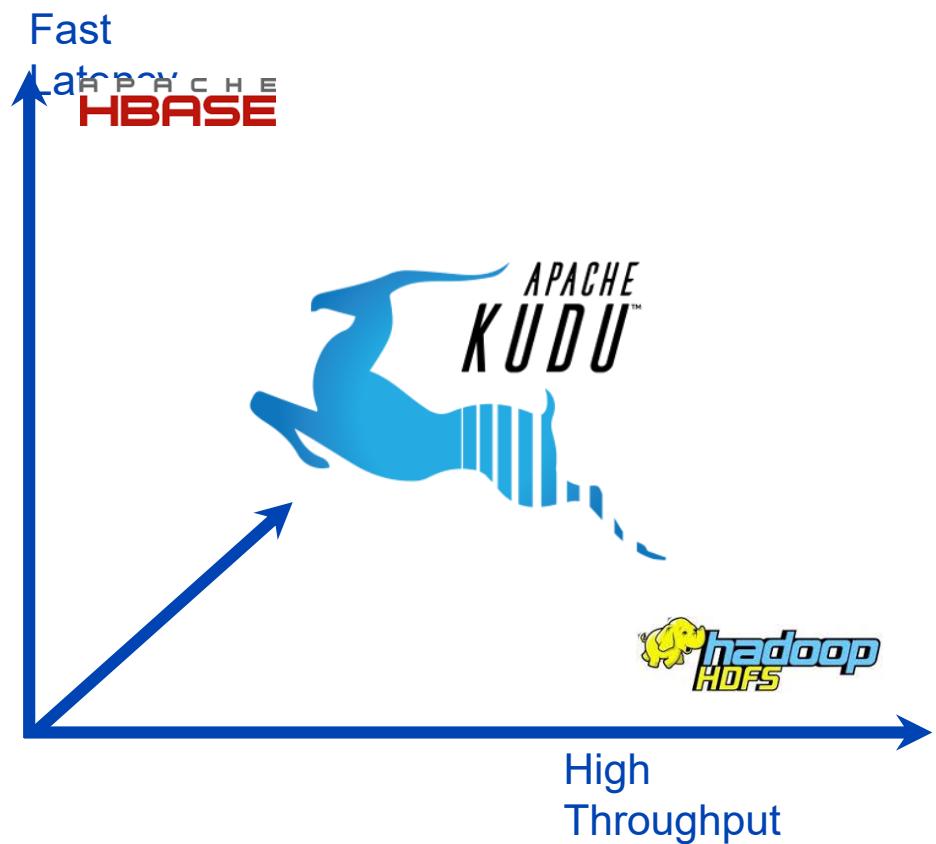
Kudu Design Goals to Meet the Gap (1/2)

- High throughput for big scans
- Low-latency for random accesses
- High CPU performance to better take advantage of RAM and Flash
- High IO efficiency
 - True column store with type-specific encodings
 - Efficient analytics when only certain columns are accessed
- Expressive and evolvable data model
- Architecture that supports multi-data center operation



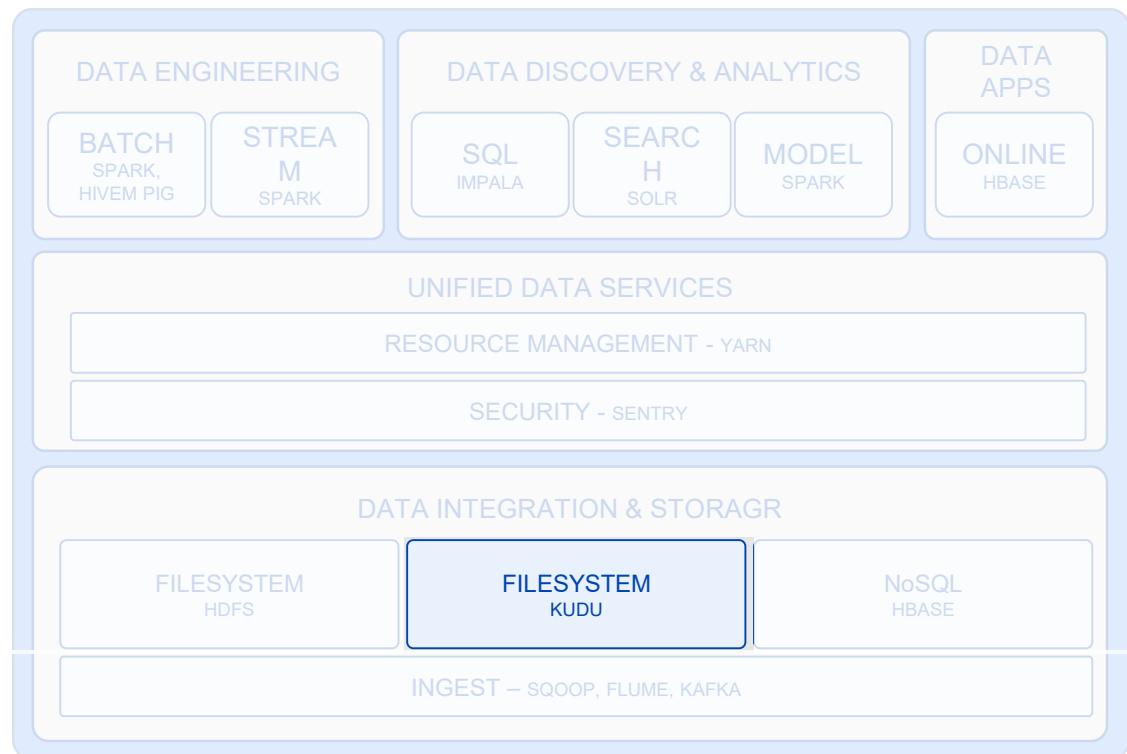
Kudu Design Goals to Meet the Gap (2/2)

- Fast processing of OLAP workloads
- Strong performance for running sequential and random workload simultaneously
- Integration with MapReduce, Spark and other Hadoop ecosystem components
- Tight integration with Apache Impala
- Strong but flexible consistency model
 - Choose consistency requirement of a per-request basis
- Highly available and Fault-tolerant
 - Continue to provide read/writes in the face of faults



Kudu's Place in the Hadoop Ecosystem

- Updatable column store for Hadoop
- Apache open source

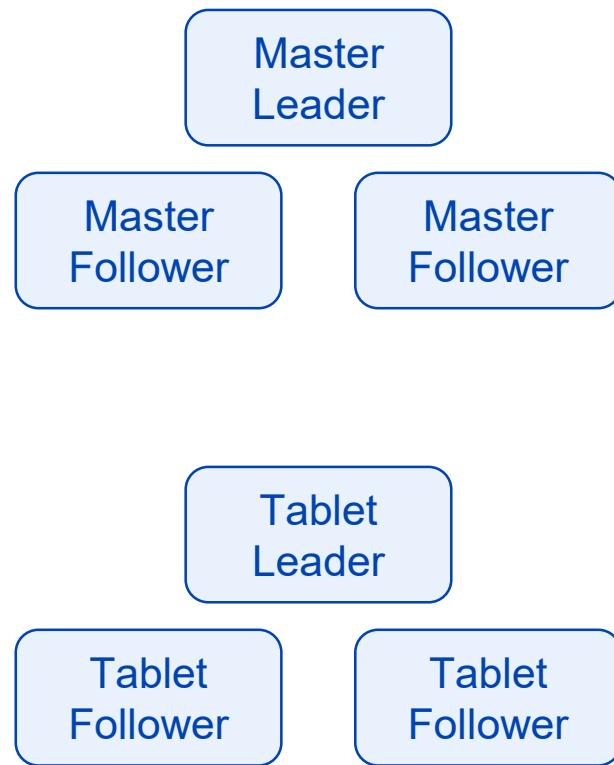


What Kudu is NOT

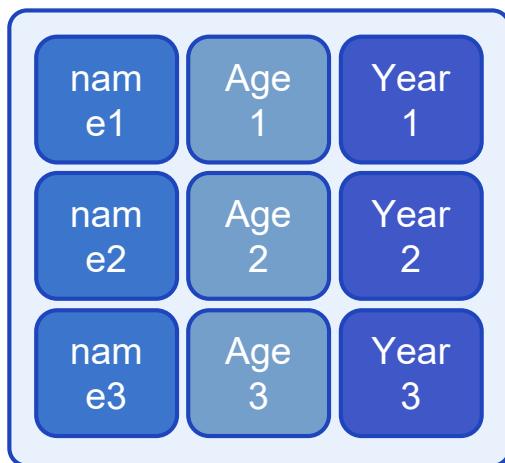
- Not a SQL interface itself
 - It's just the storage layer
 - Bring Your Own SQL (e.g. Impala or Spark)
- Not an application that runs on HDFS
 - It's an alternative, native Hadoop storage engine
 - Colocation with HDFS is expected
- Not a replacement for HDFS or HBase
 - Select the right storage for the right use case
 - Cloudera will support and invest in all three

Kudu Basic Design

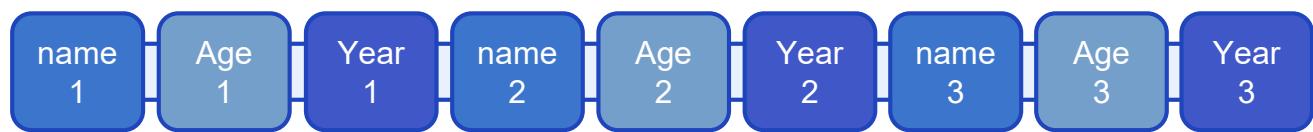
- Kudu is a columnar data store manager
 - Stores data by column that is strongly typed
- Kudu cluster stores tables that look just like SQL database tables
 - A simple as a binary key and value pair
 - Complex with hundreds of strongly typed attributes
- Master Slave Architecture
 - Master servers responsible for metadata
 - Tablet servers responsible for actual data
- Leader Follower Architecture
 - Leader amongst multiple Master servers
 - Tablet servers can be leaders or followers



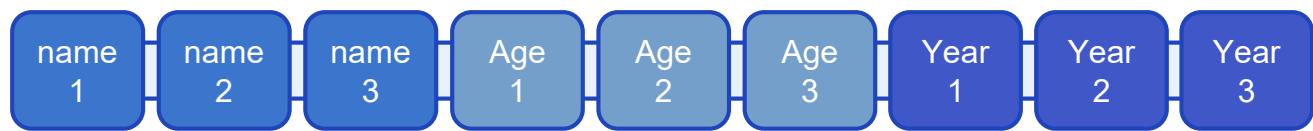
Column Data Storage



Row-Based Storage

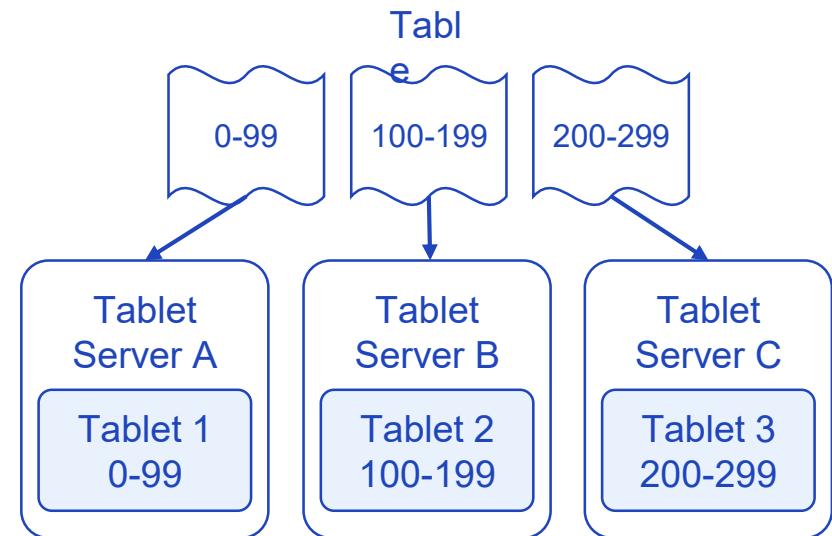


Columnar Storage



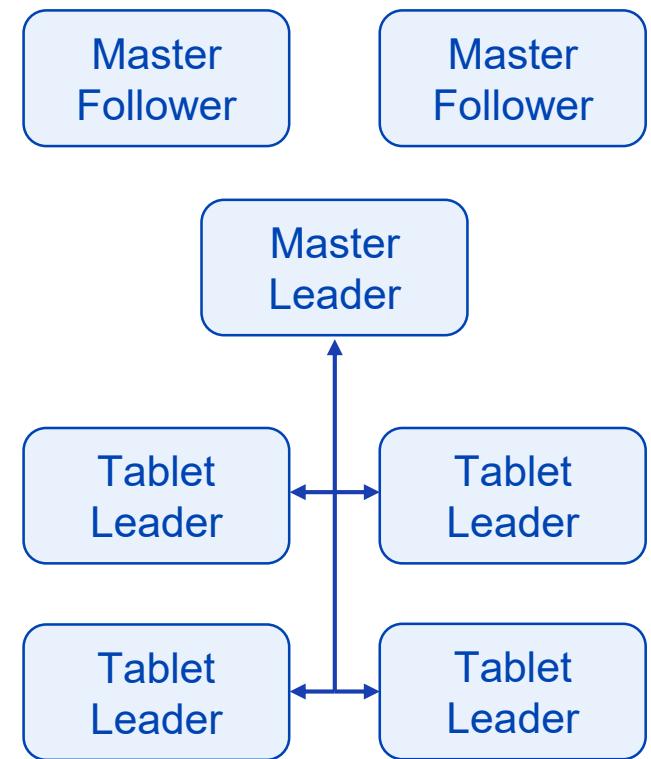
Tables and Tablets

- Every table must have a PRIMARY KEY
 - Single column like a unique identifier or complex compound key
- A table is split into segments called tablets
 - Table is horizontally partitioned into tablets
 - A tablet is a contiguous segment of a table
- Rows are assigned to tablets based on partitioning strategy
 - Partitioned based on one or more partition keys
 - Partition key column must be one of the primary key columns
 - Default is no partitioning
- Goal of partitioning strategy is to distribute reads and writes amongst the tablet servers
- Tablets are stored on local disks
 - Not HDFS



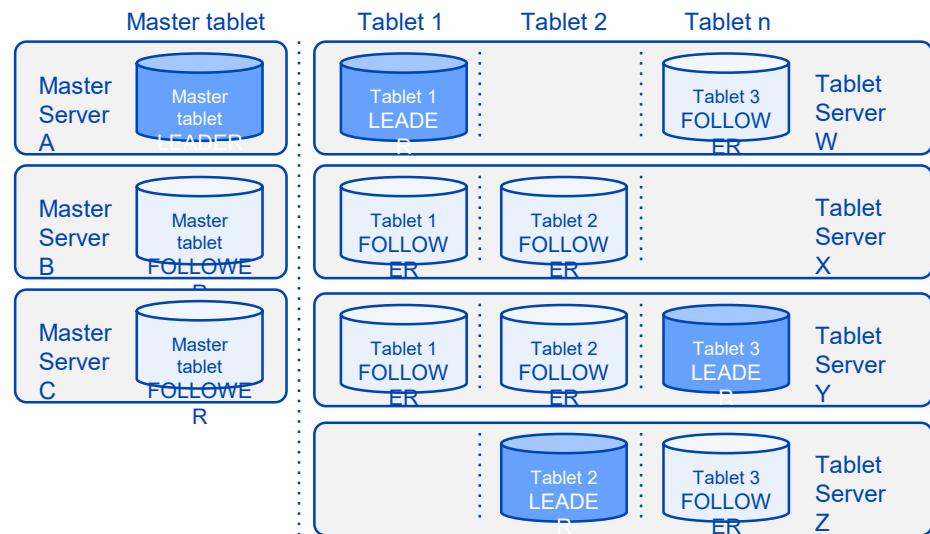
Master Slave Architecture

- Master keeps track of all the tablets, tablet servers, and the Catalog Table
- Catalog Table is the central location for the Kudu Metadata
 - Stores information about tables and tablets
 - The Catalog Table is a special table and can not be accessed directly
 - Accessible via metadata operations exposed in the client API
- A Kudu cluster can have multiple Masters
 - Provides fault-tolerance in case of failure
 - However, there can only be one active master
 - If Leader Master fails, a new one is elected using Raft Consensus Algorithm
- A Tablet Server stores and serves tablets for clients
 - For any given table, one tablet server acts as leader and others act as follower replicas of the tablet



Leader Follower Architecture

- Each tablet has multiple replicas
 - Typically 3 or 5 (odd number) replicas
 - Each replica is served by a tablet server
- One of the tablet servers, serves as Leader
 - Only leaders service write requests
- The rest serve as Followers
 - Leaders and Followers, each service read requests
- Leaders are elected using Raft Consensus Algorithm
- Raft Consensus Algorithm is also used for write operations
 - Leader sends write requests to all the followers
 - Once write is persisted by a majority of followers, Leader can acknowledge client
 - Guarantees fault-tolerance and consistency
 - For N replicas, $(N-1)/2$ faulty replicas can be tolerated



Raft Consensus Algorithm

[Video]

- Objective is to reach distributed consensus
 - Consensus among many nodes that are distributed across a cluster
 - What is the value of column "name" in row 3 stored in tablet servers A, B and C?

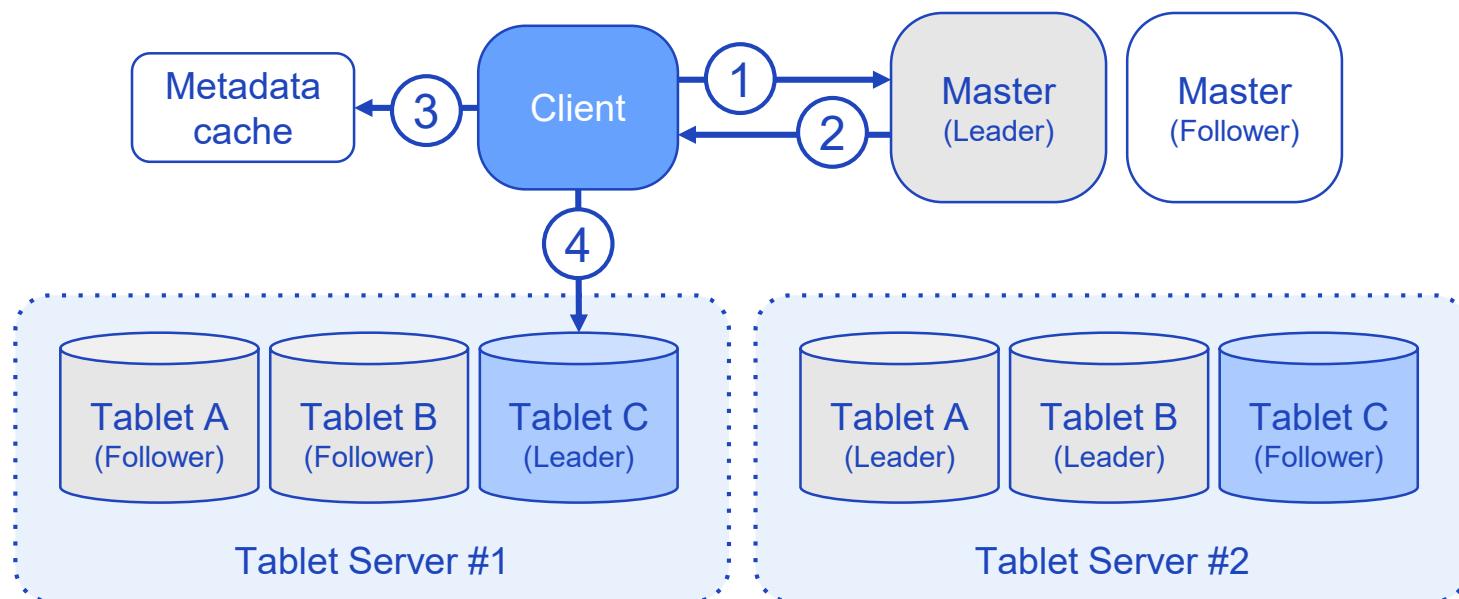
So What is Distributed Consensus?

Let's start with an example...

Continue ➔

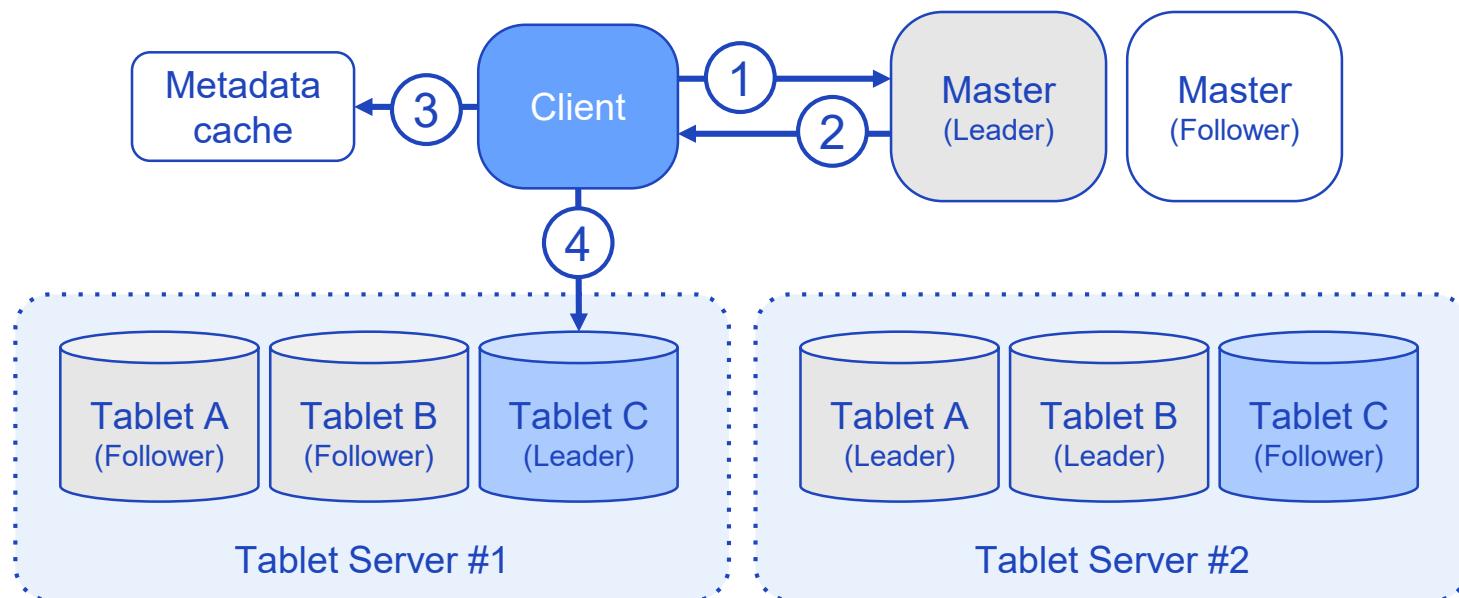
Client Read/Write Access Pattern (1/4)

- Step 1 Client contacts Master for location of data for read/write access



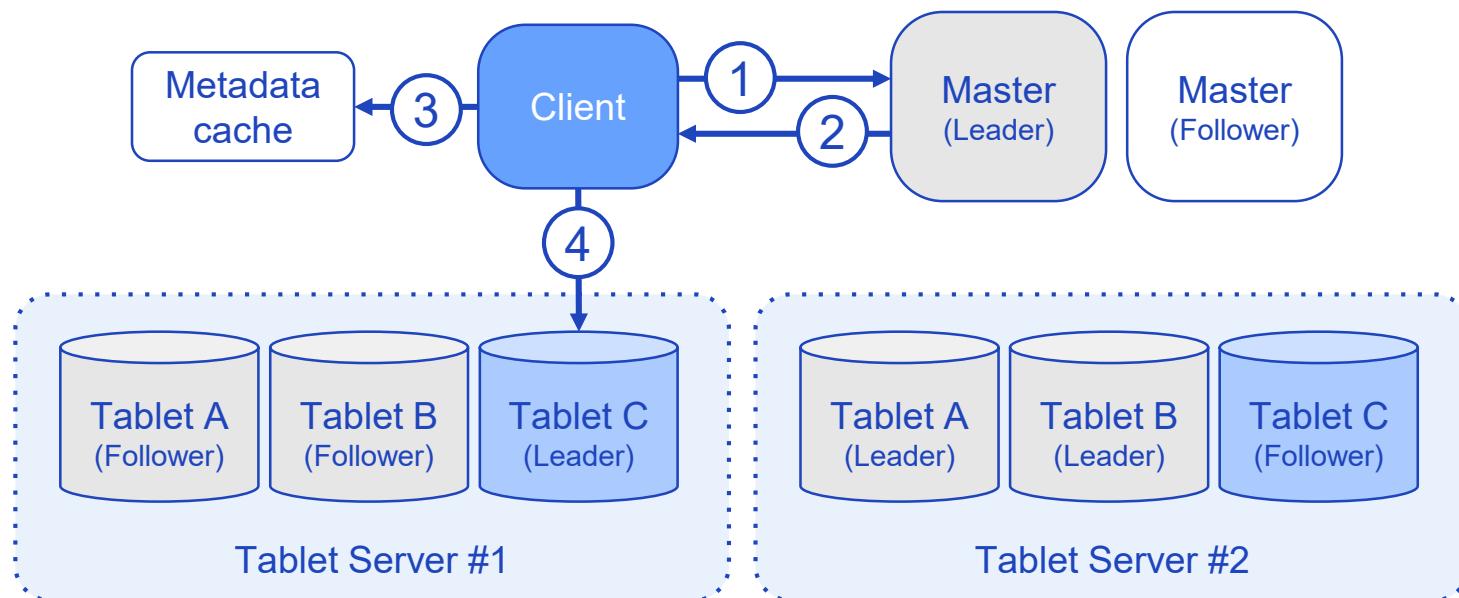
Client Read/Write Access Pattern (2/4)

- Step ② Master provides location of tablet and the Tablet Server storing the data



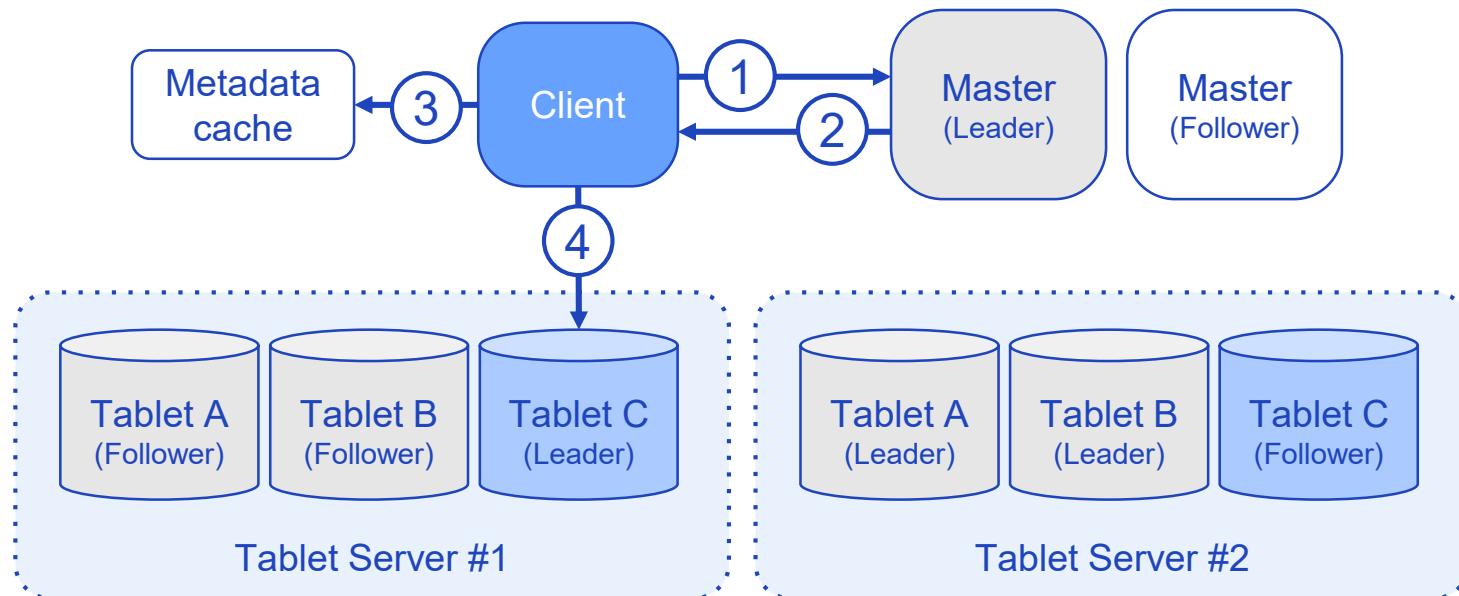
Client Read/Write Access Pattern (3/4)

- Step 3: Client caches the Metadata for all recently accessed tablets



Client Read/Write Access Pattern (4/4)

- Step 4: Client directly contacts Tablet Leader for the tablet to be accessed and begins read/write operations



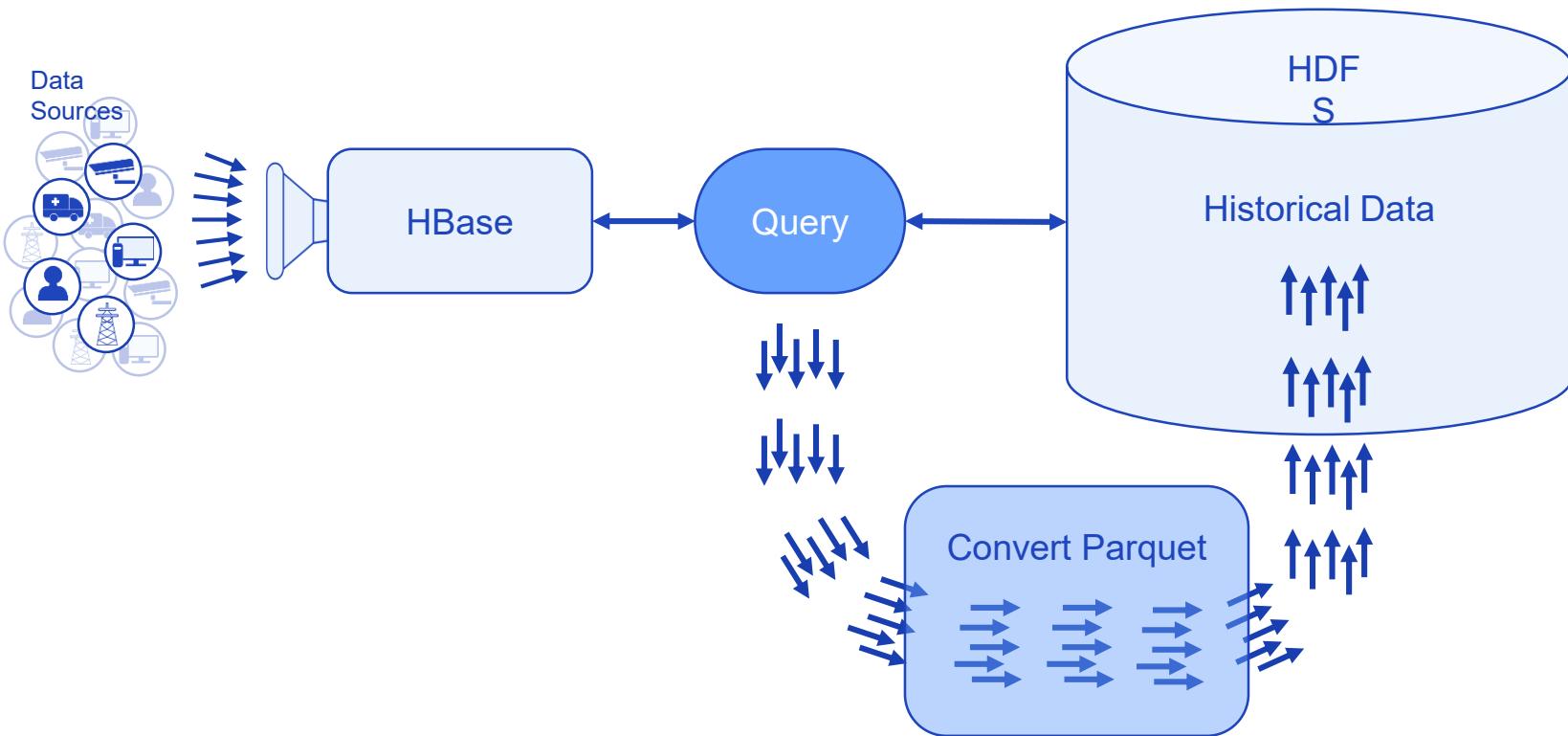
Fault Tolerance (1/2)

- Transient FOLLOWER failure:
 - Leader can still achieve majority
 - Restart follower tablet server within 5 min and it will rejoin transparently
- Transient LEADER failure:
 - Followers expect to hear a heartbeat from their leader every 1.5 seconds
 - 3 missed heartbeats: leader election!
 - New LEADER is elected from remaining nodes within a few seconds
 - Restart within 5 min and it rejoins as a FOLLOWER
- N replicas handle $(N-1)/2$ failures

Fault Tolerance (2/2)

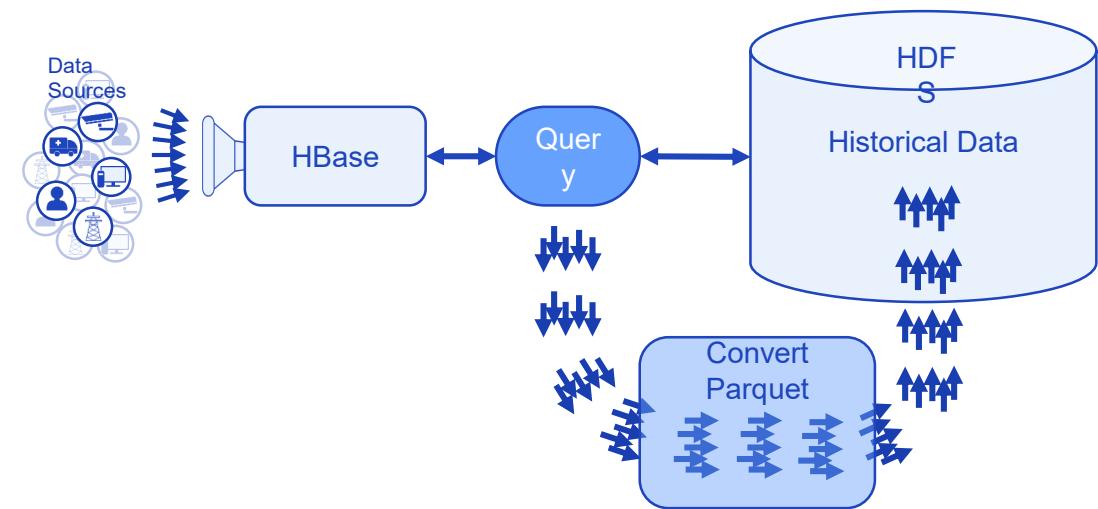
- Permanent failure:
 - Leader notices that a follower has been dead for 5 minutes
 - Evicts that follower
 - Master selects a new replica
 - Leader copies the data over to the new one, which joins as a new FOLLOWER

Real-time Analytics Example

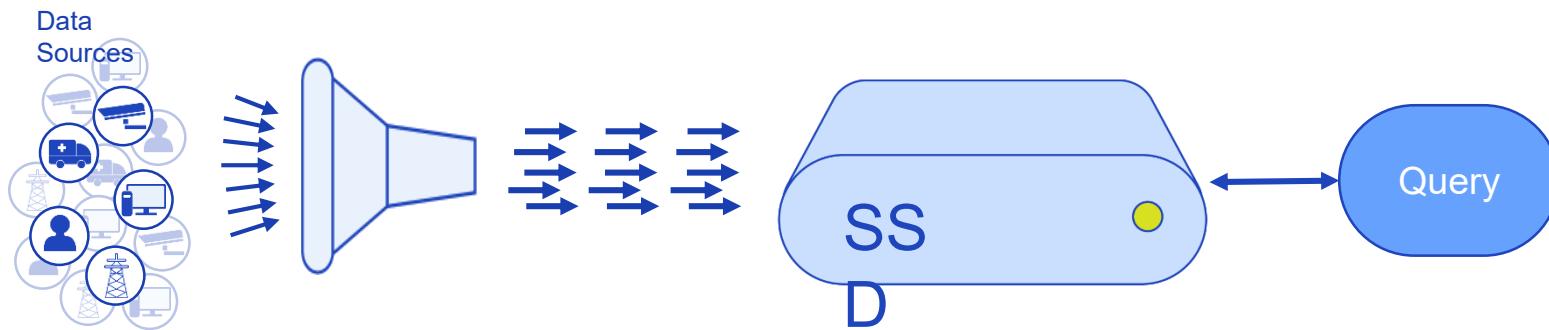


Real-time Analytics Issues

- Considerations
 - How do I handle failure during this process?
 - How often do I reorganize data streaming into a format appropriate for reporting?
 - When reporting, how do I see data that has not yet been reorganized?
 - How do I ensure that important jobs aren't interrupted by maintenance?

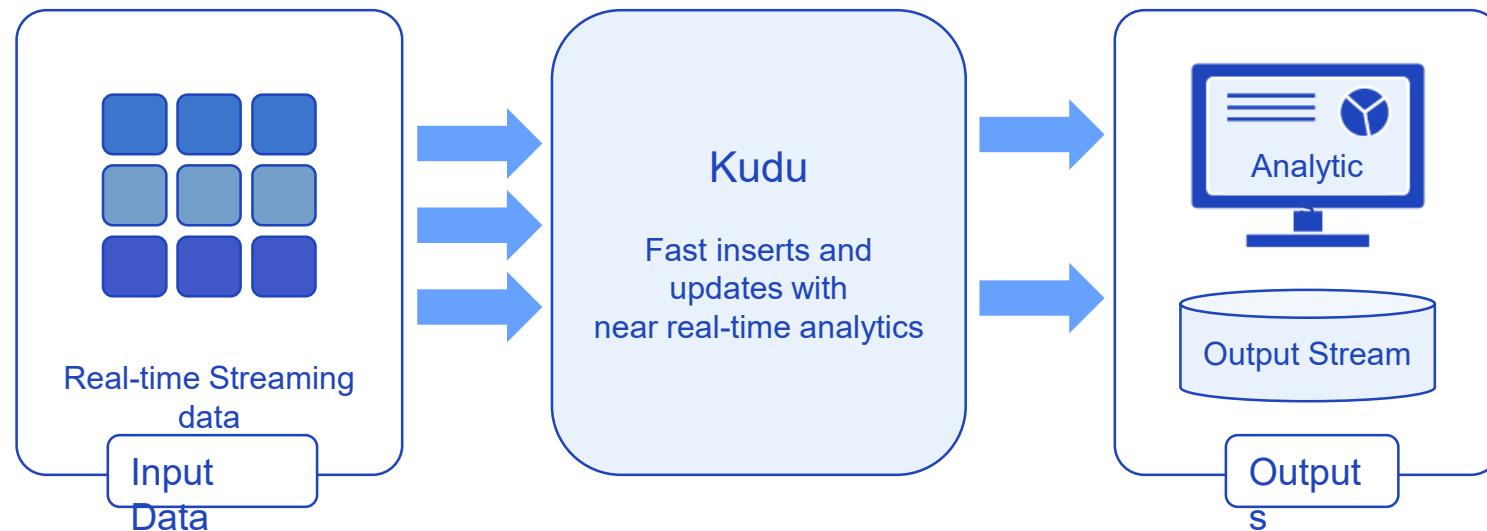


Real-time Analytics with Kudu



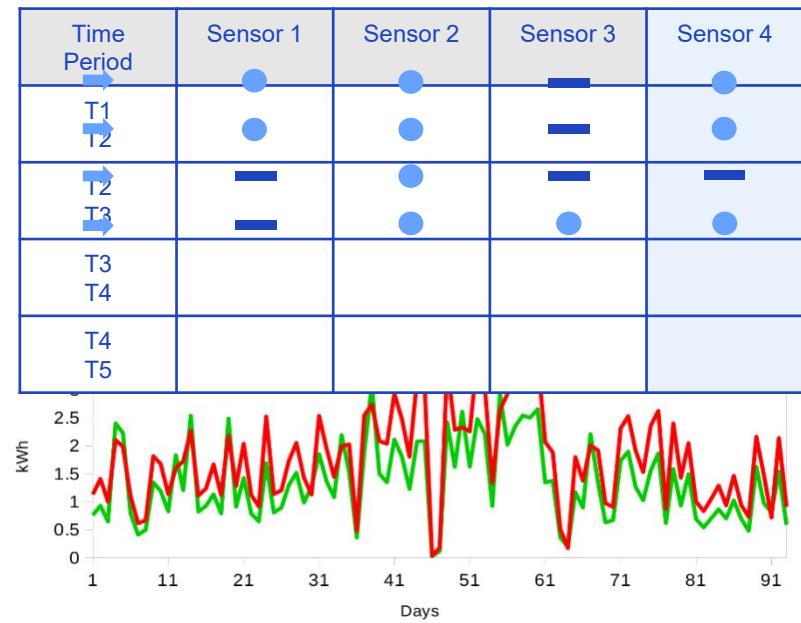
Kudu Use Case - Streaming Input

- Streaming Input with near real-time availability
 - New data arrives rapidly and constantly
 - Requires fast inserts and updates
 - Efficient columnar scans to enable real-time analytics



Kudu Use Case - Time Series Applications

- Time-series application with widely varying access patterns
 - Data points are organized and keyed according to the time they occur
 - Investigate performance of metrics over time
 - Predict future behavior based on past data
 - Analytic scans of past data along with real-time inserts and updates
 - Ex: Time-series customer data stores purchase click-stream history to predict future purchases



Kudu Use Case - Predictive Modeling

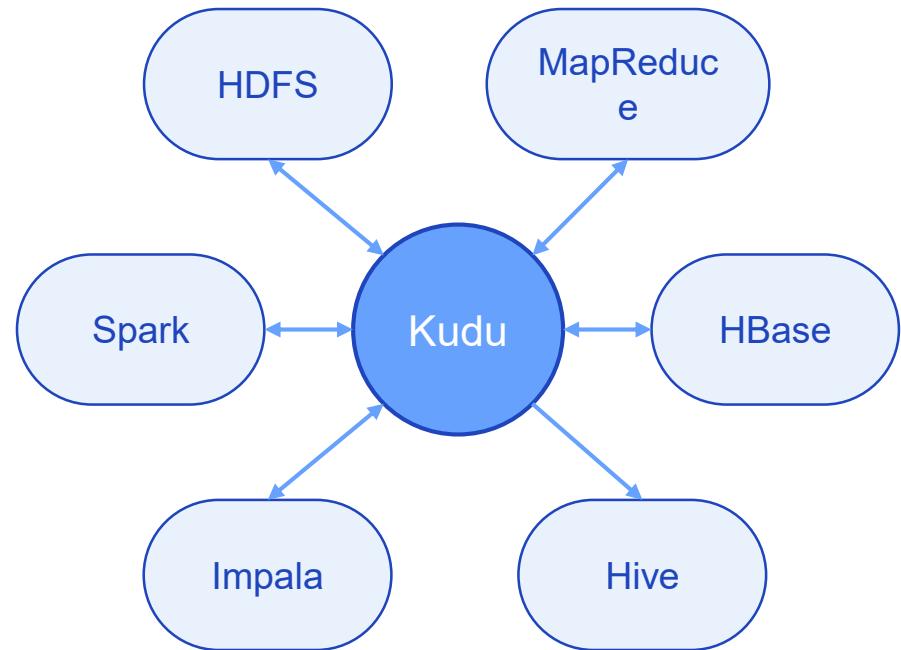
- Predictive Modeling
 - Develop predictive learning models from large data sets
 - Model and data requires frequent updates and modification
 - Data Scientist may want to change one or factors in the model to observe changes over time
 - Updating in HDFS requires the data to be completely rewritten
 - In Kudu, these updates and changes can occur in near real time

Predictive
Modeling
Identifying right customer and taking actions accordingly



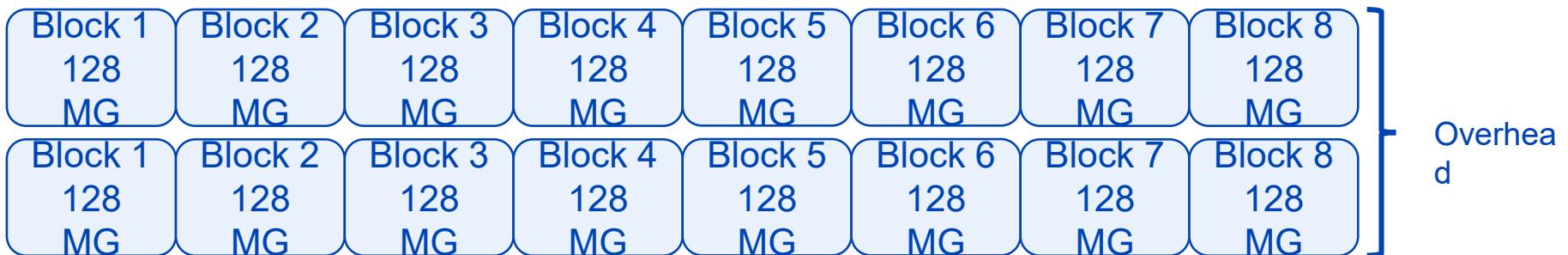
Kudu Use Case - Integrate Legacy Systems

- Combining data in Kudu with legacy systems
 - Companies generate data from multiple sources
 - Stored in a variety of systems and formats
 - Stored in traditional RDBMS
 - Stored in flexible NoSQL
 - Stored in high scan HDFS
 - All above types of data can be integrated and queried using Impala / Kudu combination



Why Erasure Coding Storage?

- Hadoop replication is expensive
 - Replication factor of 3 means 200% overhead or 33% efficiency
 - Provides 66.67% fault-tolerance - amount of data that can be lost and still recover
 - Durability of 2 - number of simultaneous failures that can be survived
- Replicated blocks for warm and cold data are rarely accessed



Example EC Storage Using XOR Coding

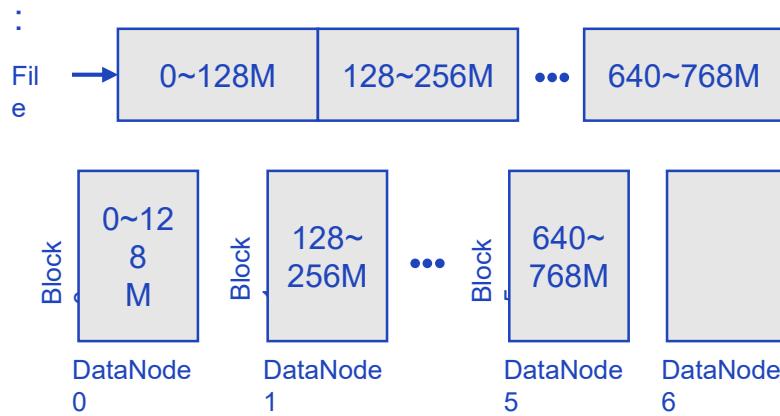
- Exclusive OR coding example
- Replication: grey bits are overhead bits
 - Requires 2 extra bits overhead or 100% overhead or 50% efficiency
 - Achieve durability of 1
- Exclusive OR
 - Requires 1 extra bit or 50% overhead or 67% efficiency
 - Achieve durability of 1

Replication	0	0	1	1	2 bits overhead
XOR Coding	0	\oplus	1	1	1 bit overhead

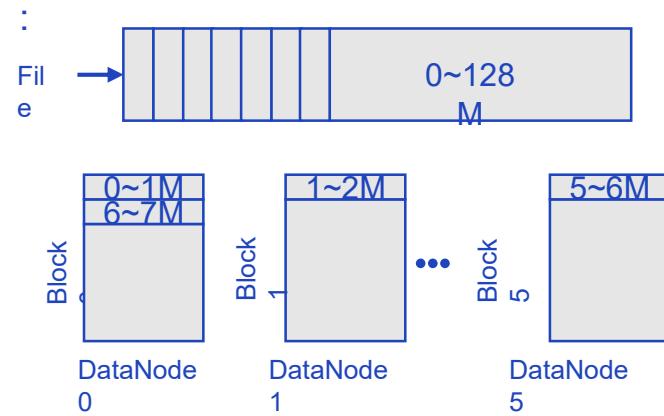
Striping Data Blocks

- HDFS data blocks without EC Storage are stored as contiguous blocks
 - The entire block is stored together
- For EC Storage, each of the data blocks is further divided into stripes and stored in different datanodes
- Basic unit of striped data is called a cell
 - For example, a 64 KB cell

Block Layout

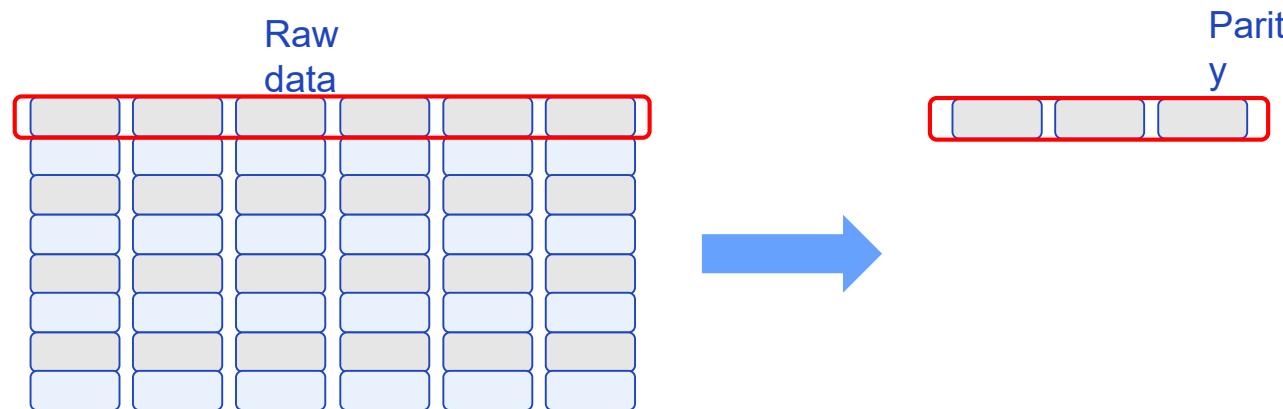


Block Layout



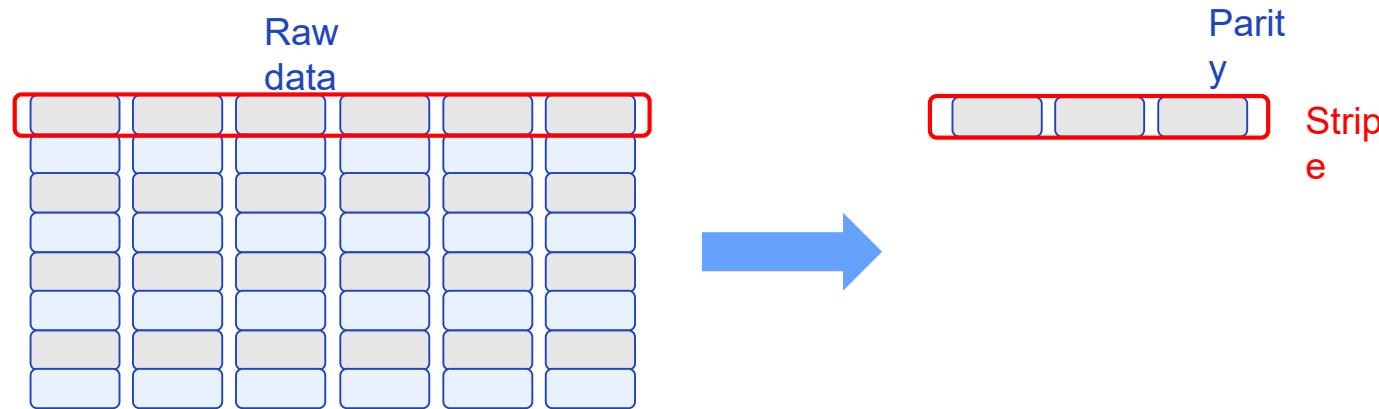
Striped Data with Parity Blocks

- Codecs used in HDFS EC storage define a data and parity block configuration
- The Reed-Solomon (6, 3) codec
 - 6 data cells will be used to calculate 3 parity cells



Striped Data with Parity Blocks

- Most of the codecs used in HDFS EC storage define a data and parity block configuration
- The Reed-Solomon (6, 3) codec
 - 6 data cells will be used to calculate 3 parity cells
- The 6 data cells and 3 parity cells is called a stripe or EC group



Comparison of Codec Algorithms

- Different codec algorithms provide different durability and efficiency

Data Durability = How many simultaneous failures can be tolerated?

Storage Efficiency = How much portion of storage is for useful data?

	Data Durability	Storage Efficiency
Single Replica	0	100%
3-way Replication	2	33%
XOR with 6 data cells	1	86%
RS (6,3)	3	67%
RS (10,4)	4	71%

Using EC Storage

- Each of the cells in EC stripe must be on a different datanode
 - For Reed-Solomon (6,3), this means a minimum of 9 datanodes configured in the cluster
- Erasure Coded files are spread across racks for rack-tolerance if available
 - Consequence is that when reading and writing striped data, most operations will be off-rack
- To use HDFS EC storage, configure or select one of the default erasure coding policies
- Erasure Coding Policy defines
 - The EC Schema - The codec algorithm and number of data and parity blocks in an EC group
 - Size of the striping cell
 - Default policy is "RS-6-3-1024K" - Reed Solomon codec with 6 data and 3 parity cells where each cell is 1024KB
- EC Policy is applied as the directory level
 - Any existing data in the directory will not be converted
 - Only new data stored to the directory will be EC coded

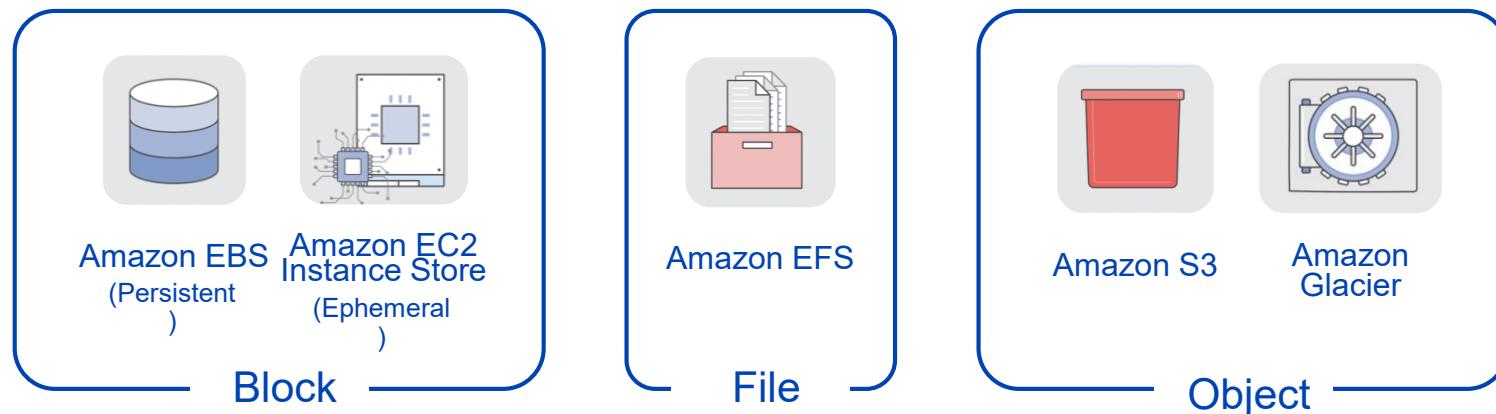
Unit 1.

Data Storage Alternatives

- | 1.1. On-Premise Storage
- | 1.2. Public Cloud Storage

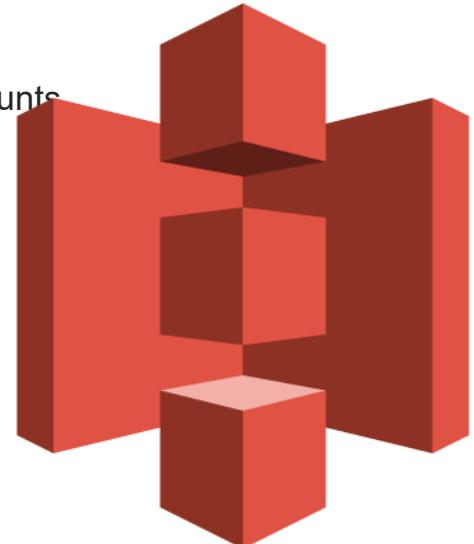
AWS Storage Services Overview

- Storage types
 - Block Storage - A storage type in which user data is stored and accessed in block units on a volume on local disk or SAN storage
 - File Storage - A storage type (NAS) that accesses storage composed of file systems in file units using a network-based protocol.
 - Object Storage - A virtual container that stores encapsulated data and attributes, metadata, and object IDs.



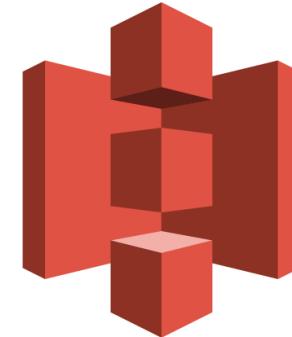
Amazon S3

- Simple Storage Service
- A simple web services interface
- Buckets
 - To upload data (photos, videos, documents, etc.) to S3, first create an S3 bucket in one AWS Region. Upload objects to this bucket
 - The bucket name is globally unique and the namespace is shared by all AWS accounts
- Permission
- Managing public access to buckets
 - ACL
 - Bucket policies



Use case with Amazon S3

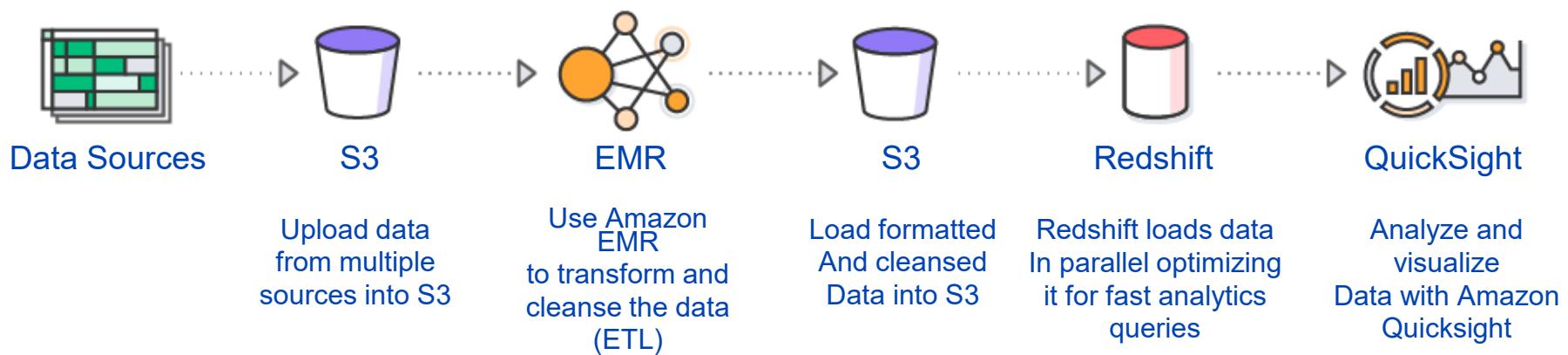
- Content Storage & Distribution
- Big Data Analytics
- Backup & Archiving
- Disaster Recovery
- Static website hosting
- Cloud-native application data
- Build a Data Warehouse



Amazon
S3

Data warehousing with Amazon S3

- Storing data using S3
- Perform data transformation operations (ETL) and analyzes via EMR
- Leveraging the redshift platform for business intelligence applications



Buckets

- A bucket is a container for objects stored in S3
- All objects in S3 are contained in a bucket
- Buckets organize the S3 namespace at highest level
- Identify the account responsible
- Used for access control



Objects

- Objects are the fundamental entities stored in S3
- Objects consist of object data and metadata
 - The data portion is opaque and visible
 - Metadata consists of properties of the object and is stored in key-value pairs
 - Some example default metadata
 - date last modified
 - HTTP metadata such as Content-Type
 - You can view the meta programmatically or from console
- Each object is uniquely identified within a bucket by a key and a version ID

Keys

- A key is the unique identifier for an object within a bucket
- The combination of bucket, key, version ID uniquely identifies every object
- This combination is used as the web service endpoint

`https://mybucket.s3.us-west-2.amazonaws.com/myobjectkey`



Bucket name Region name

Key name

Regions

- You can choose the geographical AWS Region where S3 will store the buckets you created
- Setting regions allows you to optimize latency and costs
- Objects stored in a region never leave the region unless you explicitly transfer them to another region

AWS Global Infrastructure



S3 Bucket naming rules

- Bucket names must be between 3 and 63 characters long.
- Use only lowercase letters, numbers, dots, and hyphens (-)
- The name must start and end with a letter or number.
- Do not use IP address format
- The name is unique within the partition.
 - A partition is a Regions group
 - AWS has three partitions
 - aws (Standard)
 - aws-cn (China)
 - aws-us-gov (AWS Govcloud in US Regions)

Accessing a bucket

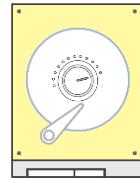
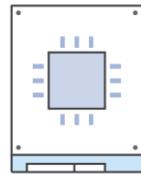
- Virtual-hosted-style access
 - <https://bucket-name.s3.Region.amazonaws.com/key name>
- Path-style access
 - <https://s3.Region.amazonaws.com/bucket-name/key name>
- Accessing a bucket through S3 access points
 - <https://AccessPointName-AccountId.s3-accesspoint.region.amazonaws.com>
- Accessing a bucket using S3://
 - <S3://bucket-name/key-name>

Deleting a bucket

- Deleting with S3 bucket versioning enabled permanently deletes all versions of all objects in the bucket.
- Considerations Before Deleting
 - Bucket names are unique. Deleting a bucket makes its name available to other AWS users.
 - Deleting a bucket that contains objects permanently deletes all objects in the bucket, including objects that have been converted to the S3 Glacier storage class.
 - If your bucket is receiving log data from Elastic Load Balancing (ELB), we recommend that you stop shipping ELB logs to the bucket before you delete the bucket.

S3 Storage classes

- Storage classes designed for your use case



S3 Standard

- ✓ Big data analysis
- ✓ Content distribution
- ✓ Static website hosting

Standard - IA

- ✓ Backup & archive
- ✓ Disaster recovery
- ✓ File sync & share
- ✓ Long-retained data

Amazon Glacier

- ✓ Long term archives
- ✓ Digital preservation
- ✓ Magnetic tape replacement

Azure Storage Services

- Microsoft Azure provides scalable and durable cloud storage, backup, and recovery solutions for all your data, large and small.
- It provides storage that is highly available, secure, durable, scalable and redundant for object store for data objects, disk storage for Azure VMs, a file system service for the cloud, a message store, and NoSQL store.
- Whether it is images, audio, video, logs, configuration files, or sensor data be stored in a way that can be easily accessible for analysis purposes.
- Core storage Services
 - Azure Blobs
 - Azure Files
 - Azure Queues
 - Azure Tables
 - Azure Disks
- Each service is accessed through a storage account

Microsoft Azure
Blob Storage



Storage Account overview

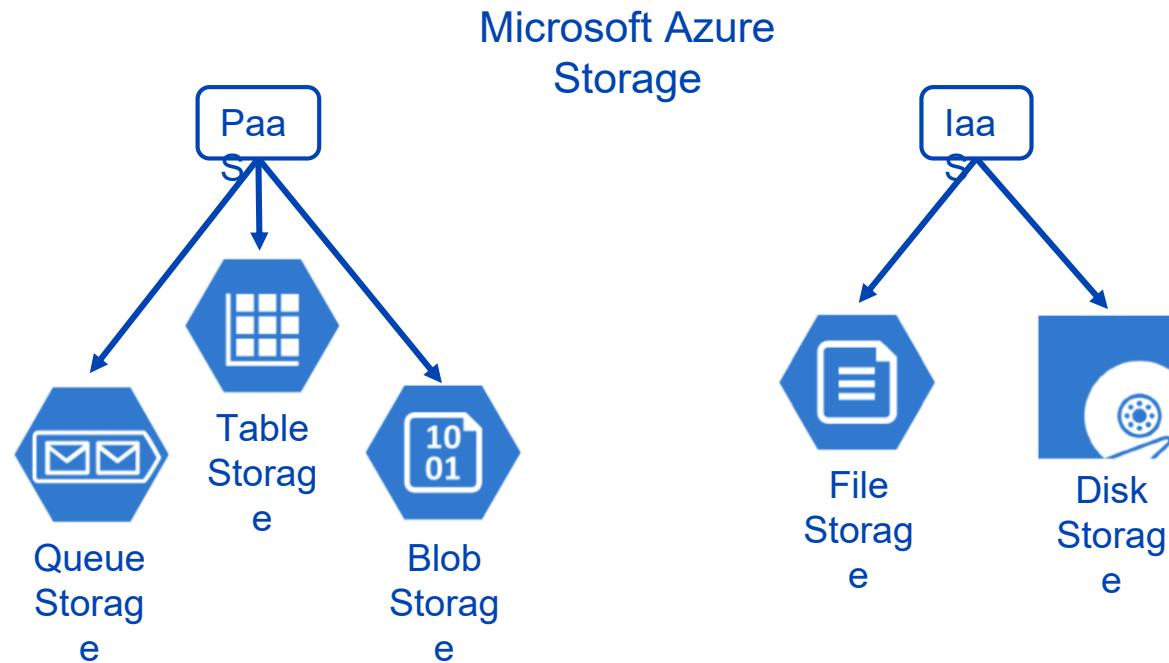
- An Azure storage account contains all Azure Storage data objects such as blobs, files, queues, tables, and disks.
- A storage account provides a unique namespace for Azure Storage data.

- Types of storage accounts

Storage account type	Supported Service	Replication method	Deployment model
General – purpose V2	Blob, File, Queue, Table, Disk, and Data Lake Gen2	LRS, GRS, RA-GRS, ZRS, GZRS, RA-GZRS	Resource Manager
General – purpose V1	Blob, File, Queue, Table and Disk	LRS, GRS, RA-GRS	Resource Manager, Classic
BlockBlobStorage	Blob (block blobs and append blobs only)	LRS, ZRS	Resource Manager
FileStorage	File only	LRS, ZRS	Resource Manager
BlobStorage	Blob (block blobs and append blobs only)	LRS, GRS, RA-GRS	Resource Manager

Core storage services

- Core storage services provide massively scalable object storage for data objects, disk storage for Azure virtual machines (VMs), file system services for the cloud, messaging storage for messaging, and NoSQL storage.

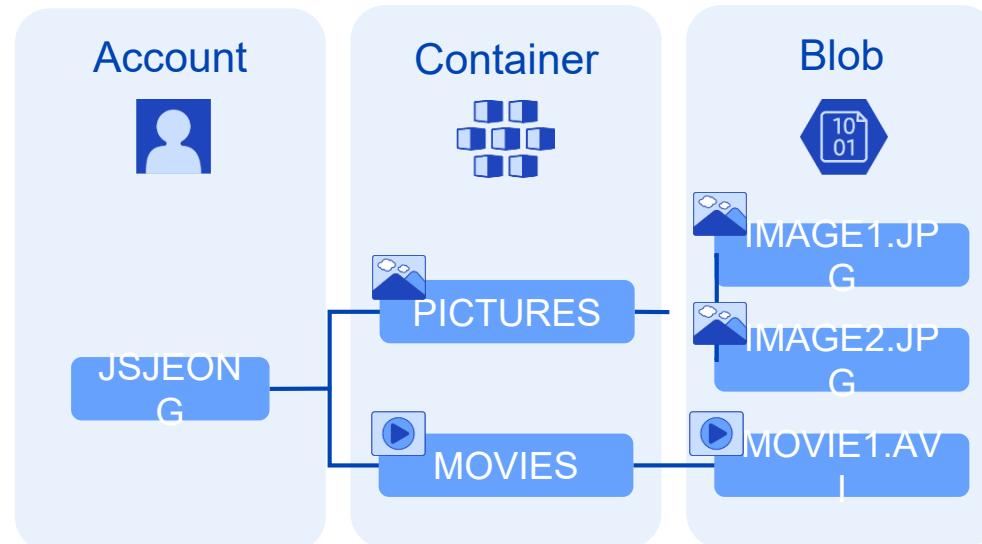


Azure Blob Storage

- Azure Blob Storage is Microsoft's object storage for cloud
- Optimized for storing massive amounts of unstructured data
- Use Cases:
 - Serve images or documents directly to your browser
 - Store files for distributed access
 - Streaming video and audio
 - write to log file
 - Storing data for backup/restore, disaster recovery and archival
 - Storing data for analytics on-premises or by Azure hosted services
- Accessible via Using Azure storage REST API, Powershell, CLI, Azure Storage client lib

Blob Storage Resources

- Blob storage offers three types of resources
 - Storage account
 - Container in the storage account
 - A blob in a container



Storage Resources – Storage Account

- Storage accounts provides a unique namespace
 - Every object stored has an address that includes the unique storage account name
 - Address is combination of account name and blob endpoint



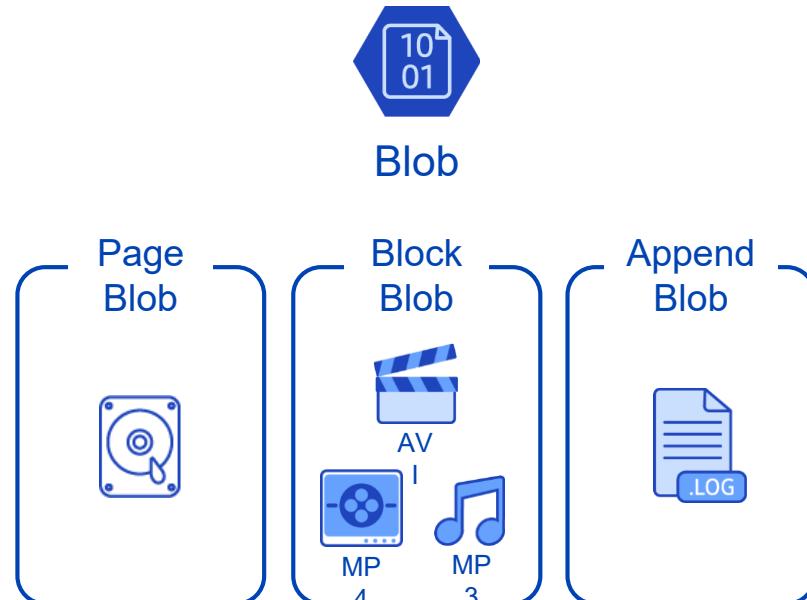
Storage
Account
s

Storage Resources – Containers

- Containers organizes a set of blobs – similar to directory
 - Can include unlimited number of containers
 - Naming – lowercase letters and numbers
 - You can not create sub-containers but you can create blobs with "/" in them to emulate sub-folders

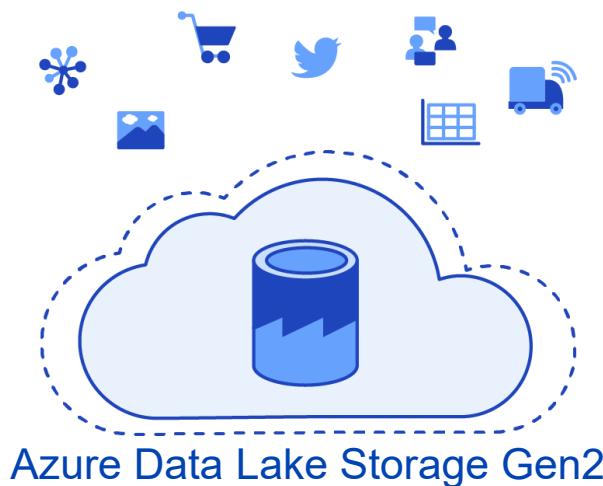
Storage Resources – Blobs

- Block blobs
 - Store text and binary data
 - Made up of blocks that can be managed separately
 - Each block can store 4.75 TiB of data
- Append blobs
 - Similar to block blobs but optimized for append ops
 - Ideal for storing logging data
- Page blobs
 - Stores random access files 512-byte to 8 TiB pages
 - Ideal for frequent random read/write operations
 - Basis for Azure IaaS disks



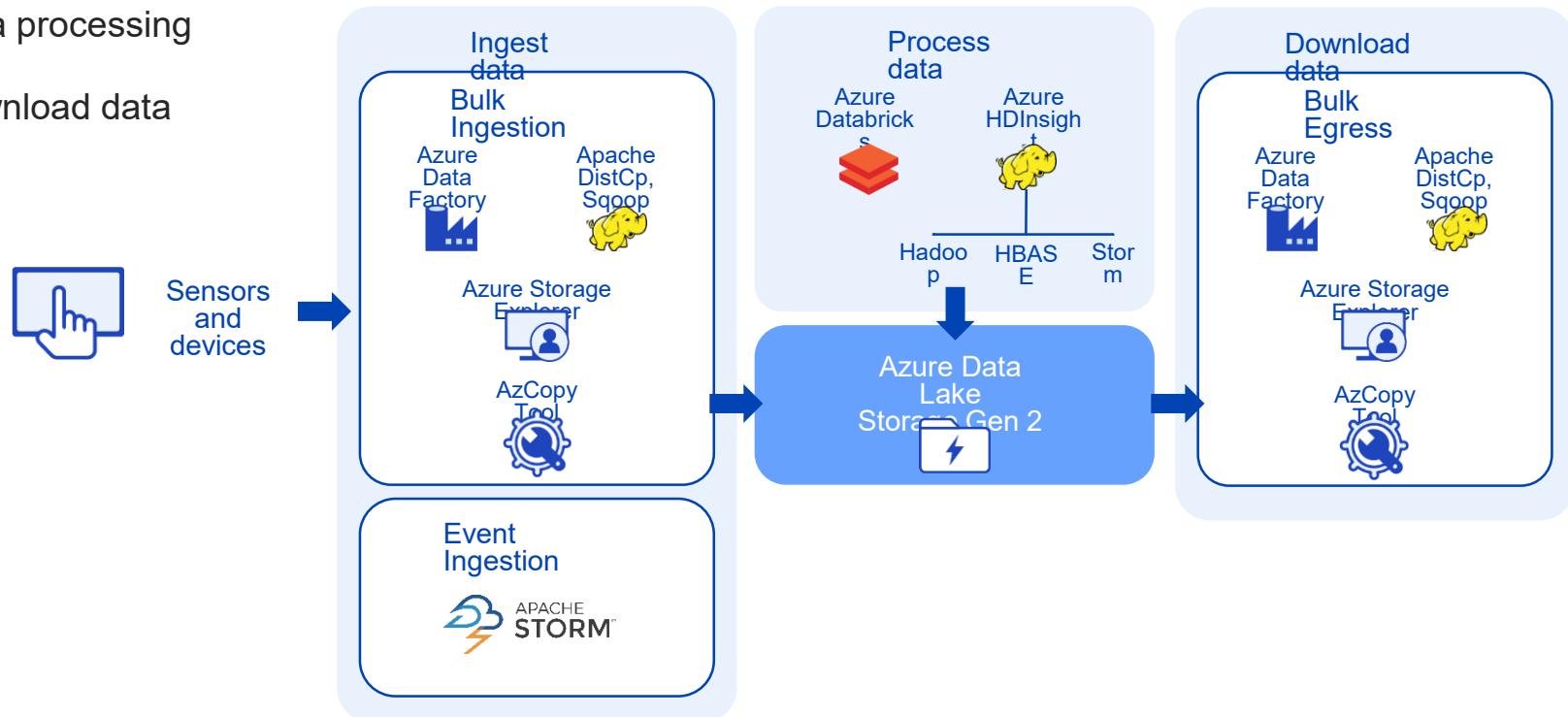
Azure Data Lake Storage Gen2

- Azure Data Lake Storage is a comprehensive, scalable, and cost-effective data lake solution for big data analytics built into Azure.
- Built on top of Azure Blob Storage
 - This integration enables the analytic performance, tiering and data lifecycle management capabilities of Blob Storage, and the high availability, security and durability features of Azure Storage



ADLS Gen2 for Big data requirements

- Collect large amounts of data in real-time or in batches to the data store
- Data processing
- Download data



Key features of Data Lake Storage Gen2

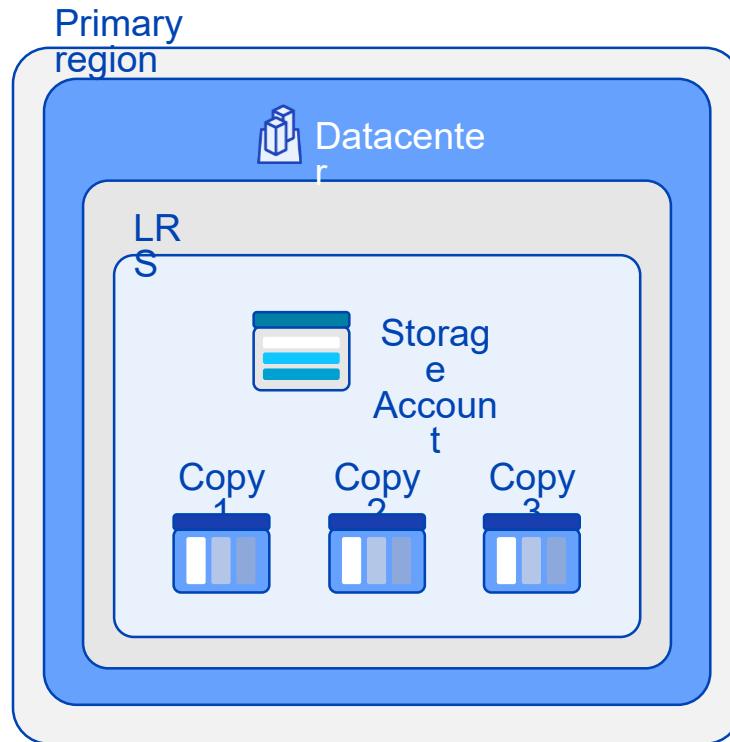
- Hadoop compatible access
 - HDFS
 - Azure HDInsight, Azure Databricks, and Azure Synapse Analytics
- A superset of POSIX permission
- Cost effective
- Optimized driver
- Scalability
- Supported Azure service integrations and open source platforms

Blob Storage Service

- Provides object storage that can be used to store and serve unstructured data
- This data can range from App and Web data to images, files, Big Data from IoT and, of course, backups and archives
- Petabytes data storage
- The types of Blob storage durability
 - Local Redundant Storage (LRS)
 - Zone Redundant Storage (ZRS)
 - Geo Replicated Storage (GRS)
 - Read Access Geo Replicated Storage (RA-GRS)

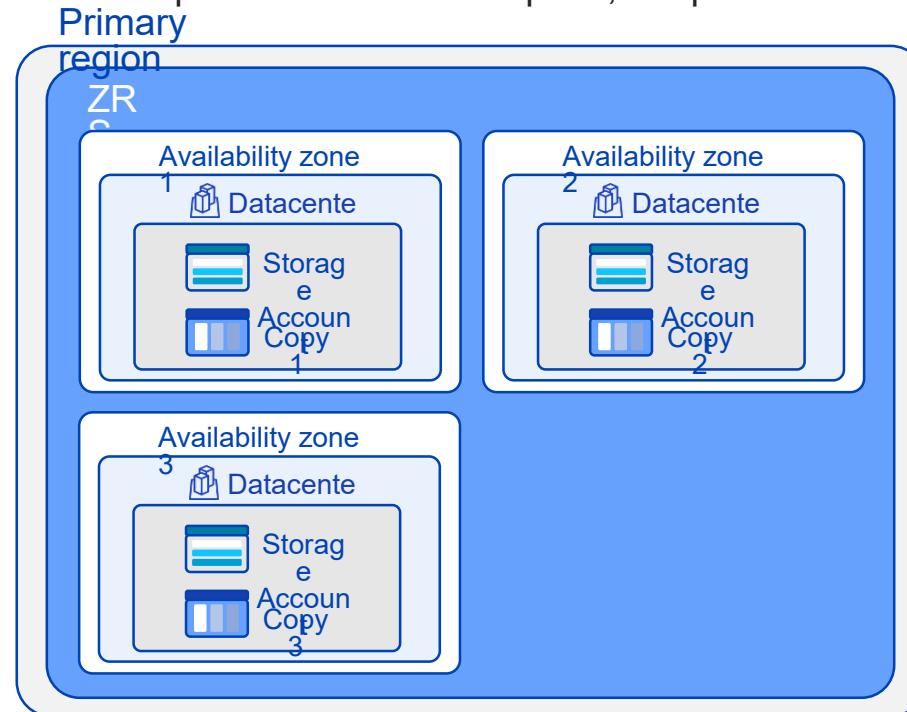
LRS(Local Redundant Storage)

- LRS storage stores three copies of data in a single region to protect against disk, node, and rack failures.



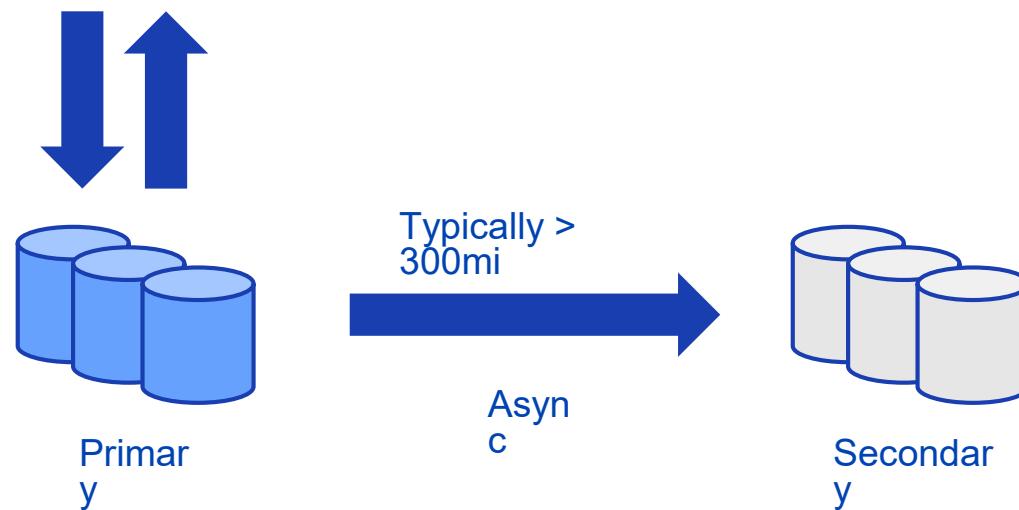
ZRS(Zone Redundant Storage)

- ZRS is similar to LRS, but splits data across two data centers in the same region.
- Makes your data more resilient compared to the default option, Simple LRS.



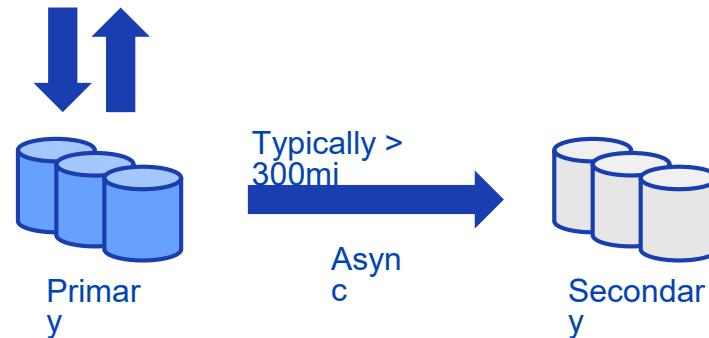
GRS(Geo Redundant Storage)

- GRS protects against major regional disasters by storing six replicas in two different regions
 - For example, Northern Europe and Western Europe
- Replication to the secondary region is asynchronous.



RA-GRS

- RA-GRS: Read Access Geo Replicated Storage
- It provides read-only access to the secondary storage through an accessible endpoint to allow for load-balanced queries



Replication Strategy	LRS	ZRS	GRS	RA-GRS
Whether data is copied to multiple data center	N	Y	Y	Y
Secondary position read service as primary	N	N	N	Y
Number of replica	3	3	6	6

Types of storage

- Standard Storage (default)
 - When deciding on Azure Storage offerings, performance and cost are two important factors
 - Normal, commodity spinning disks
- Premium Storage
 - For high-performance, low latency disk support
 - Stores data on solid state drives (SSDs)
- Cold Storage
 - This service is geared towards infrequent access
 - It is designed for the purpose of archiving data, such as database and VM backups.
 - The API and SDKs between hot and cool storage are identical
 - Cheaper cost. If you access frequently, then the cost for reading data is higher. (Considering storage space & frequency of use)

[Lab1]

AWS S3 and Glacier



[Lab2] AWS S3 Glacier Storage



Unit 2.

NoSQL

Big Data Storage

Unit 2.

NoSQL

- | 2.1. NoSQL Overview
- | 2.2. Apache HBase
- | 2.3. Cassandra
- | 2.4. MongoDB

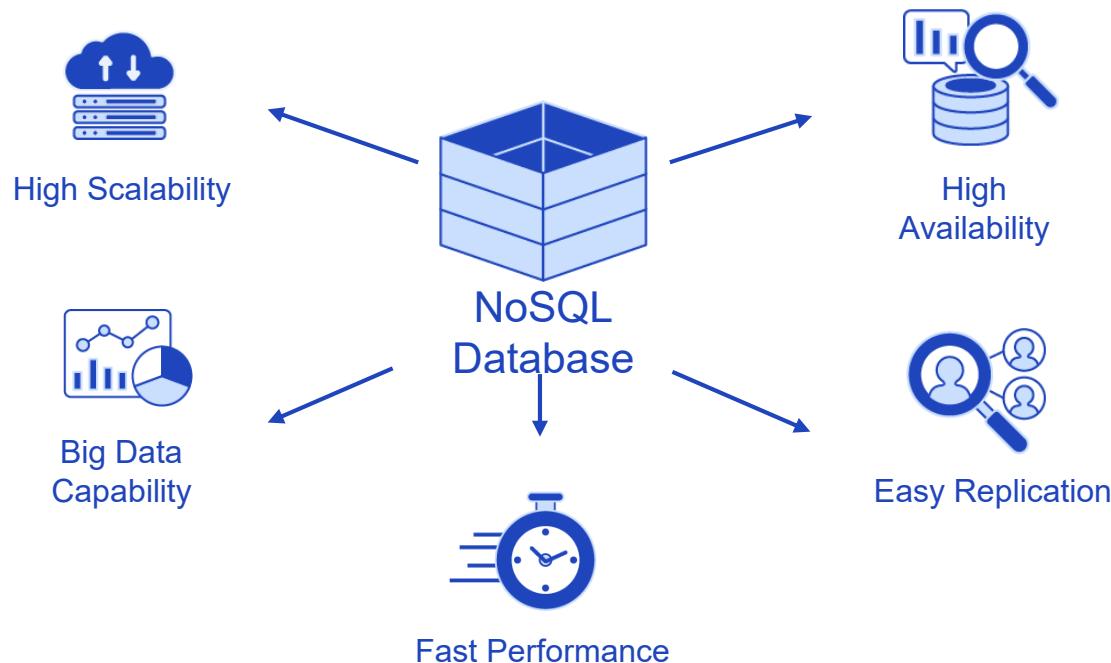
What is NoSQL? (1/2)

- Stands for “No SQL”, “**Not only SQL**”, “Non-relational Operational SQL”
 - RDB that does not support standard SQL interface
- In 2009, Johan Oskarsson used in open source based distributed DB
- Schema less, not support transaction
- Data Storage with non-relational DB schema

NotOnlySQL

What is NoSQL? (2/2)

- Complement for Relational database
- A flexible database management system that can store and process both structured and semi-structured data



Brief History of NoSQL

- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database. (1998)
- Graph database Neo4j is launched (2000)
- Google **BigTable** is launched (2004)
- CouchDB is launched (2005)
- The research paper on **Amazon Dynamo** is released (2007)
- Facebooks open sources the **Cassandra** project (2008)
- The term NoSQL was reintroduced by Johan Oskarsson (2009)

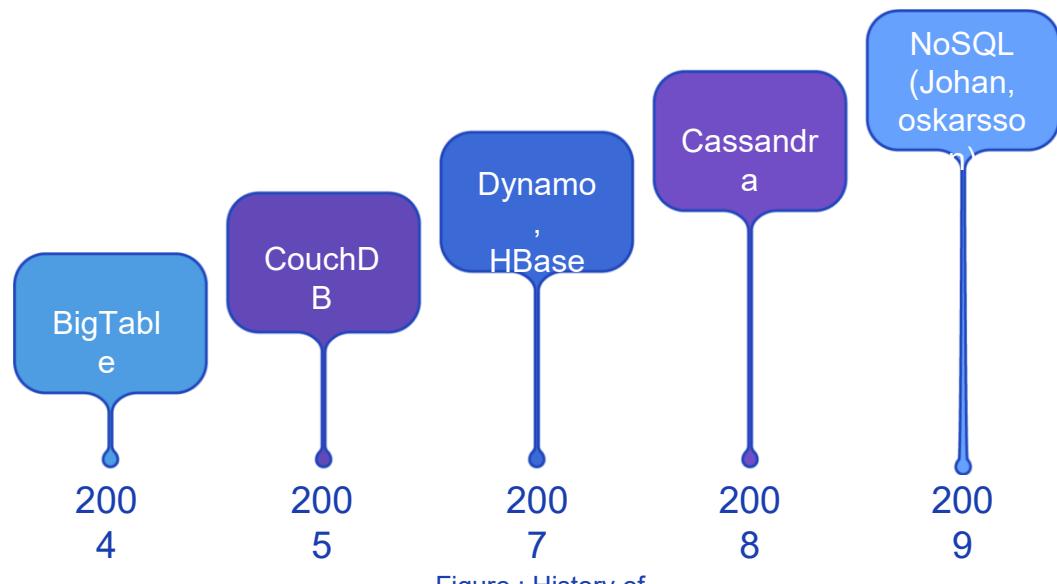


Figure : History of HBase

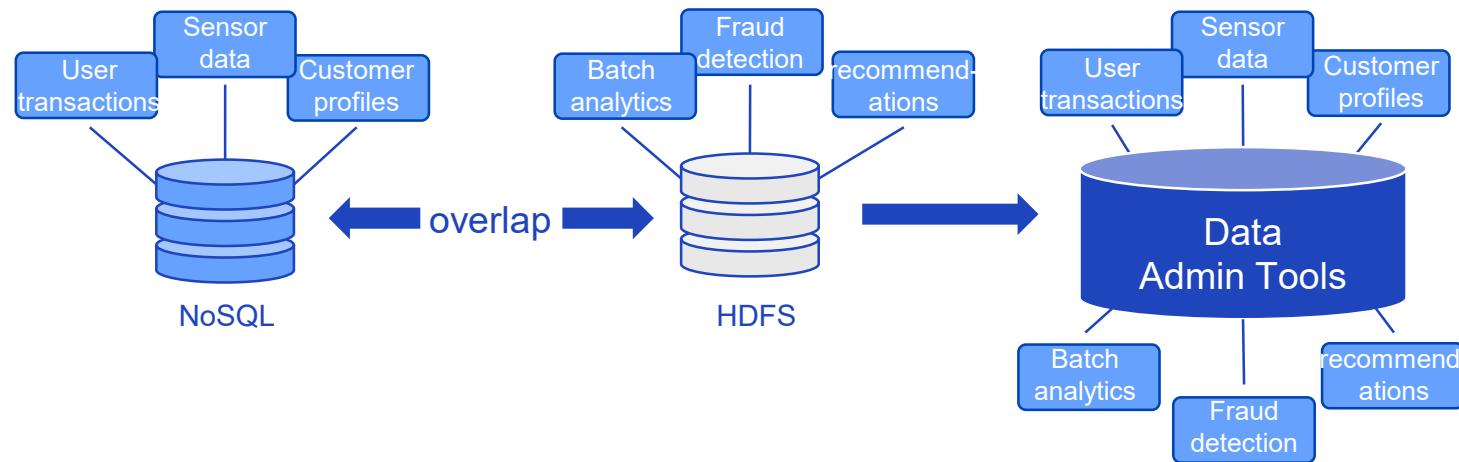
Characteristics of NoSQL (1/2)

- Complex queries are not supported.
 - In general, join operations are not supported.
- Schema flexible response
 - Support not normalizing
- Ready-to-deploy cloud-scale design
 - Works fine in clusters
- No ACID required
 - Write queries work on all databases, and consequently propagate to other distributed servers
 - BASE is instead of ACID.

Characteristics of NoSQL (2/2)

- BASE – Basically Available, Soft state, Eventual consistency
- Basically, Available
 - There is a possibility of a fault, but it does not become a fault of the entire system.
- soft state
 - System state can be changed without input
 - The state of a node is determined through information transmitted from the outside.
- Eventual consistency
 - Eventual consistency is the guarantee of consistency over time.

NoSQL VS HDFS



- Real-time
- Interactive
- Fast reads/writes
- Batch
- Large-scale processing
- Massive compute power

Relational(SQL) VS NoSQL Database (1/2)

- Relational Database
 - Relational databases provide a mechanism for storing and retrieving data through table relationships.
 - A table consists of rows (records, tuples) and columns (fields, items).
 - Organize data in relation to the values of table fields
 - Ex) MySQL, DB2, Oracle, MS-SQL
- NoSQL
 - Easy cloning and simple API support
 - A Non-Relational database (No Tables schema)
 - Ideal for big data and real-time systems
 - Various types of NoSQL databases
 - Ex) HBase, Cassandra, MongoDB

Relational(SQL) VS NoSQL Database (2/2)

	Relational DB	NoSQL
Data Storage	A relational model with rows and columns	Various storage models: document, key-value, column type, graph type.
Schema & flexibility	<ul style="list-style-type: none">• Each record follows a fixed schema• Columns must be predefined• When entering data, a null value is entered even in an empty field.• Once a schema is set, it is difficult to change it,• It costs a lot for changes.	<ul style="list-style-type: none">• Schema is variable• Change in real time when data is input• Empty fields have no storage space
Scalability	<ul style="list-style-type: none">• Scale UP• Requires larger and more expensive servers to expand data.	<ul style="list-style-type: none">• Scale Out• When expanding data, it is possible to increase the number of general servers.
ACID compliant	Compatible with Atomicity, Consistency, Isolation, and Durability	Sacrificing ACID for performance and scalability
Data Type	Structured data	Structured, semi-structured, unstructured
Product	MySQL, Oracle, MS-SQL	MongoDB, HBase, Cassandra

Important Aspect of RDBMS - ACID

- Atomicity
 - Each transaction is atomic. A transactional operation is not partially executed, is performed in full, or is canceled (All or Nothing)
 - Ex) Bank transfer (processing error cannot occur in the middle)
- Consistency
 - All constraints (constraints, triggers, keys) of the data model are maintained before and after transaction execution
- Isolation
 - Transactions do not interfere with each other
- Durability
 - Once a transaction is executed, the result is guaranteed forever.

Advantages of NoSQL over RDBMS

- Big Data support
- Allow continuous availability or splitting
- Data model – flexible schema support
- Support for various data structures
- Supports linear scalability- horizontal scaling
- Support for major developer languages and platforms
- Open Source

Disadvantages of NoSQL over RDBMS

- Lack of standardization support and completeness
- The burden of different management methods for each system
- Increased data storage with redundant storage
- Lack of analytics and BI tools

When to Use NoSQL

- Support for multiple applications requiring different levels of performance, consistency, availability, and scalability
- Systems that produce tens of terabytes or more of data per day
- Systems that require high performance but do not require consistency
- Guaranteed eventual consistency after a delay
- Big data storage and processing
 - Big Data - 3V (Volume, Variety, Velocity)

Features of NoSQL

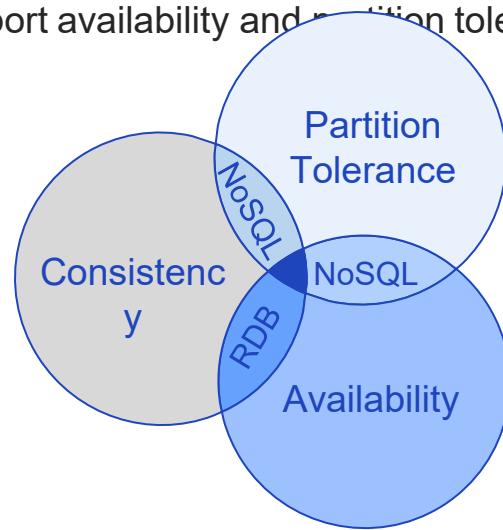
- Non-relational
 - NoSQL database avoids relational model
 - Does not provide a table of fixed column record type
 - No object-relational mapping and data normalization required
 - Efficiently manage large size and semi-structured data
- Schema-free
- Simple API

What is CAP Theorem? (1/2)

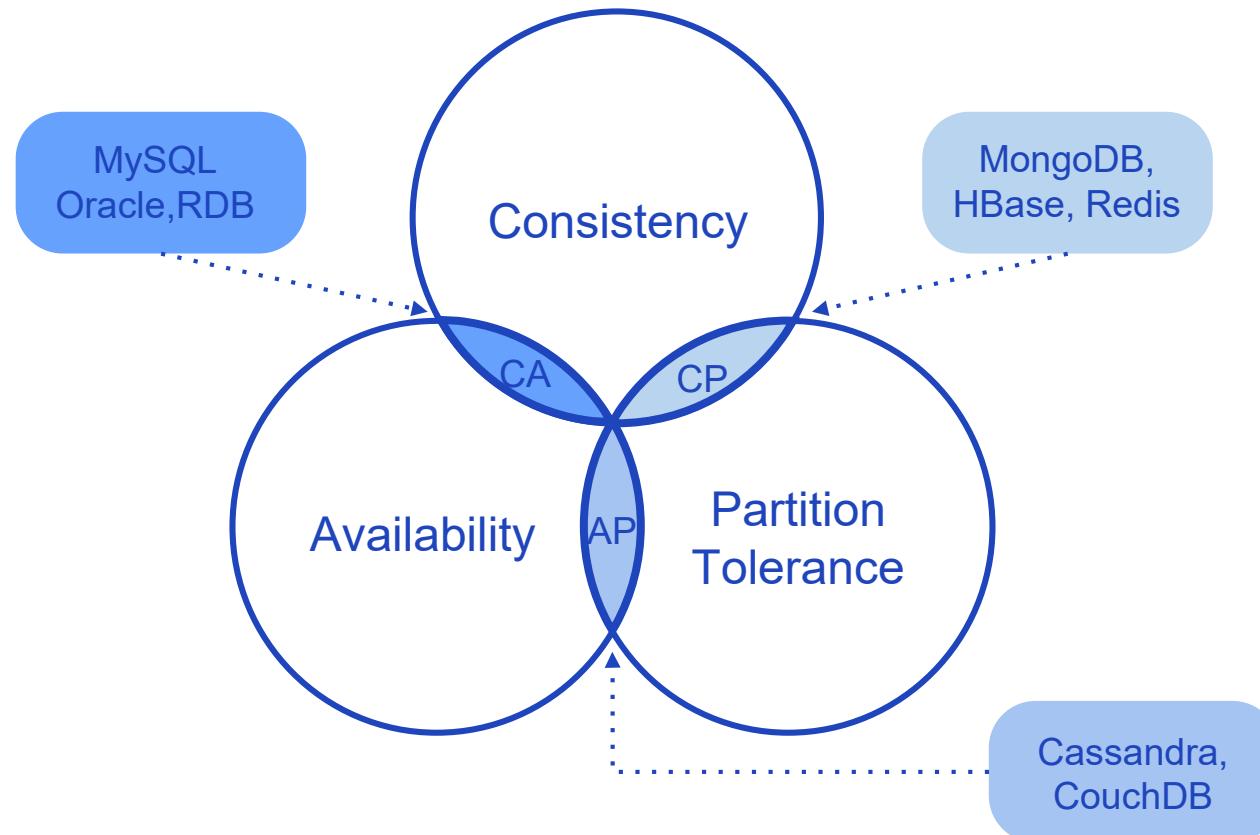
- According to the CAP (C, A, P) theory, no distributed system can simultaneously support the three properties of consistency, availability, and partition tolerance.
 - Proposed by Eric Brewer in 2000.
- Consistency
 - All nodes provide the same data for the same item at the same time.
- Availability
 - All clients can always perform Read/Write operations (ex. 24X7)
- Partition toleration
 - The system works even on physical network partitions (message delivery fails or parts of the system fail)

What is CAP Theorem? (2/2)

- A distributed data system is a balance between consistency, availability, and partition tolerance.
- Every database guarantees only 2 of 3 properties in CAP
- Relational databases generally provide consistency and availability, but not partition tolerance.
 - When a partition occurs, the system hangs
- NoSQL databases generally support availability and partition tolerance.

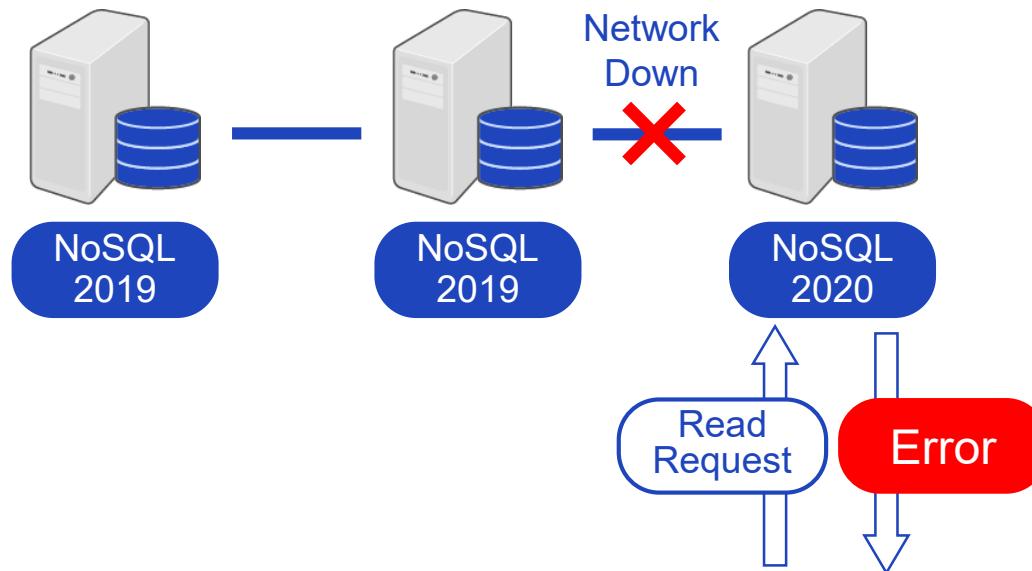


CAP theorem in Database



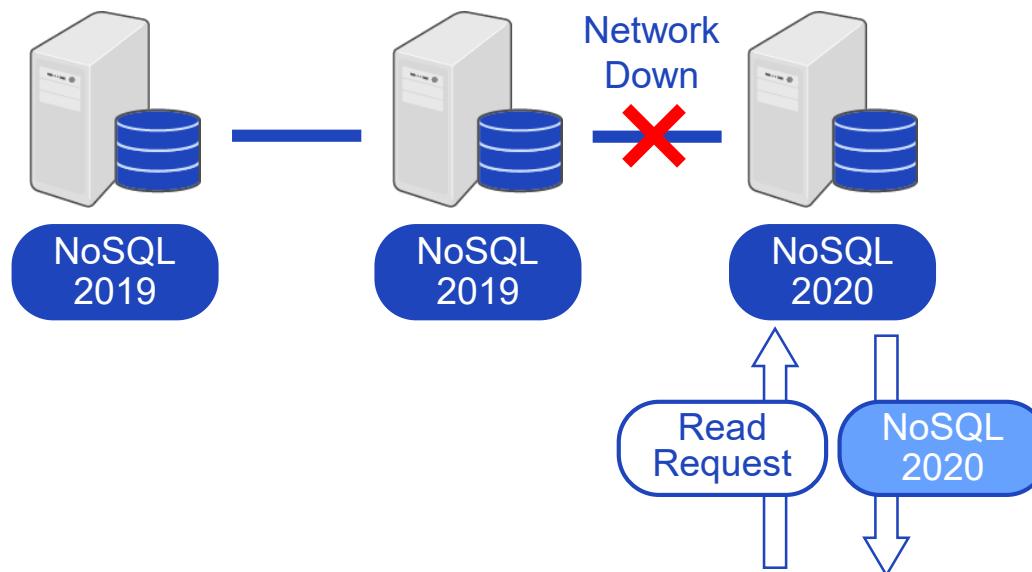
Consistency & Partition-Tolerance

- Ensure consistency and partition tolerance (CP) and give up availability (A)
 - Database unavailable while network is down to ensure consistency
 - There is a risk of some data becoming unavailable



Availability & Partition-Tolerance

- Ensure availability and partition tolerance (AP) and give up consistency (C)
 - When a read request is received, information held by the requested node is transmitted.
 - Because it doesn't check for consistency, Clients can read inconsistent data.
- Ex) DNS domain information



Types of NoSQL DB

Column store type



APACHE
HBASE

Document store type



Key-Value store type



amazon
DynamoDB

Graph store type

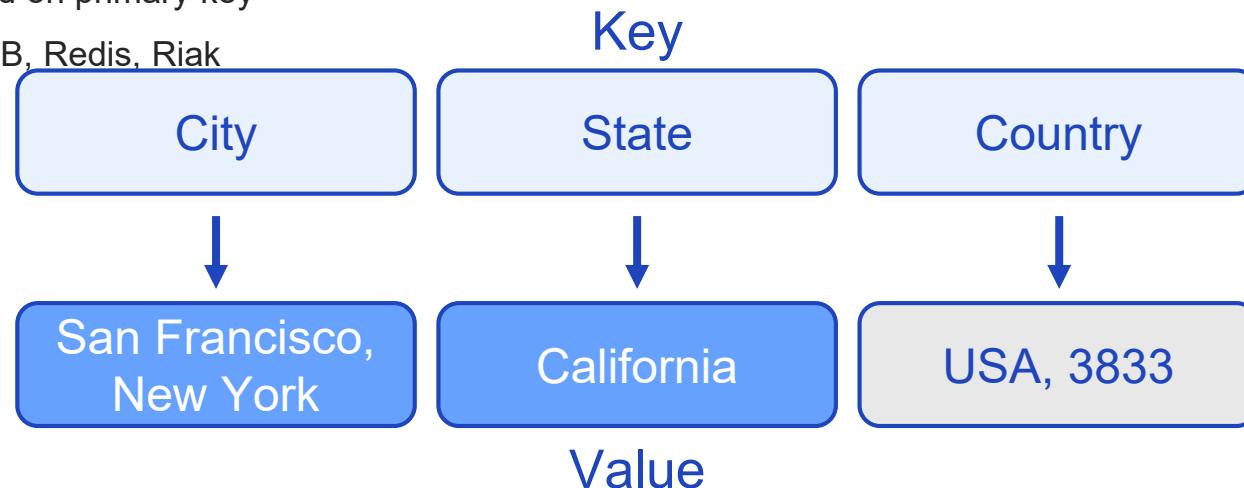


Characteristics by type of NoSQL DB

- Key-value Stores - DynamoDB, Redis, Riak
 - A key-value store associates each data value with a unique key. Most key-value stores only support simple query, insert, and delete operations
 - Useful for random access
- Document Database - MongoDB, CouchDB, Elastic Search
 - Store data in JSON-like structure
- Column Database - Apache Cassandra, HBase, BigTable
 - Organize your data into columns instead of rows
 - Advantages for querying large data sets
- Graph Database - neo4j, Giraph
 - Data into nodes (objects) and edges (relationships)

Key-value Stores

- Data is stored as key-value pairs
 - It stores the data as a hash table where each key is unique, the values are JSON, BLOB (Binary Large Objects), string
 - A form that has one value in a unique key
 - Working based on primary key
 - Ex) DynamoDB, Redis, Riak



Document Database

- Document/XML/Object Store
 - Extended form of Key-Value Store, not suitable for complex queries
 - Suitable for CMS systems, blog platforms, real-time analytics and e-commerce applications
 - A structure that stores data in the Value field corresponding to the Key, and the data type of the stored Value is Document (XML, JSON..)
 - Ex) MongoDB, CouchDB, Elasticsearch

```
{  
  "UUID": "21f7f8de-8051-5b89-  
  "Time": "2011-04-01T13:01:02.4  
  "Server": "A2223E",  
  "Calling Server": "A2113W",  
  "Type": "E100",  
  "Initiating User": "dsalings@spy.net",  
  "Details":  
    {  
      "ip": "10.1.1.23",  
      "API": "InsertDVDQueueItem",  
      "trace": "cleansed",  
      "tags":  
        [  
          "SERVER",  
          "US-West",  
          "API"  
        ]  
    }  
}
```

Document 1

```
{  
  "UUID": "21f7f8de-8051-5b89-  
  "Time": "2011-04-01T13:01:02.4  
  "Server": "A2223E",  
  "Calling Server": "A2113W",  
  "Type": "E100",  
  "Initiating User": "dsalings@spy.net",  
  "Details":  
    {  
      "ip": "10.1.1.23",  
      "API": "InsertDVDQueueItem",  
      "trace": "cleansed",  
      "tags":  
        [  
          "SERVER",  
          "US-West",  
          "API"  
        ]  
    }  
}
```

Document 2

```
{  
  "UUID": "21f7f8de-8051-5b89-  
  "Time": "2011-04-01T13:01:02.4  
  "Server": "A2223E",  
  "Calling Server": "A2113W",  
  "Type": "E100",  
  "Initiating User": "dsalings@spy.net",  
  "Details":  
    {  
      "ip": "10.1.1.23",  
      "API": "InsertDVDQueueItem",  
      "trace": "cleansed",  
      "tags":  
        [  
          "SERVER",  
          "US-West",  
          "API"  
        ]  
    }  
}
```

Document 3

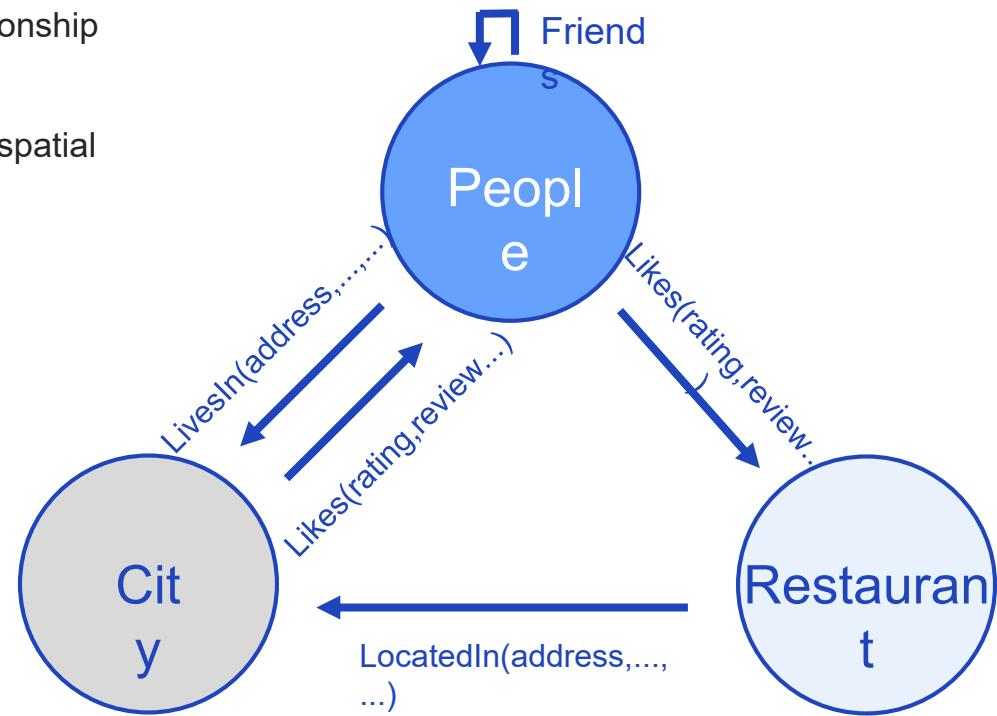
Column Database

- Based on BigTable paper by Google
 - Column data is saved together, as opposed to row data
 - Ideal for aggregate queries such as SUM, COUNT, AVG, MIN, etc., as the data is readily available in columns
 - Ex) HBase, Cassandra, BigTable



Graph Database

- Graph Store
 - Storing data in graph structure
 - Nodes are stored independently, and the relationship between nodes (edges) are stored with data
 - Mainly used for social networks, logistics, and spatial data
 - Ex) Neo4j, OrientDB



Summary for NoSQL Data Store

Data Model	performance	Scalability	Flexibility	Complexity	Functionality
Key-Value store	High	High	High	none	Variable (none)
Column-Oriented Store	High	High	Moderate	Low	minimal
Document-oriented Store	High	Variable High	High	Low	Variable (Low)
Graph Database	variable	variable	High	High	Variable (none)
Relational Database	variable	variable	Low	Moderate	Variable (none)

Unit 2.

NoSQL

- | 2.1. NoSQL Overview
- | 2.2. Apache HBase
- | 2.3. Cassandra
- | 2.4. MongoDB

What is HBase?

- HBase is NoSQL running on top of HDFS.
- HBase is...
 - The representative column store type
 - Highly Available and fault tolerant
 - Highly scalable and high throughput
 - Easy to handle large tables
 - Data with scattered rows with a large number of columns
 - Open source, Apache project
- Features provided by HDFS
 - Fault tolerance
 - Scalability



Hbase Terminology

- Node
 - One server, machine
- Cluster
 - A group of nodes in which multiple nodes are connected and collaborate to perform a task
- Master Node
 - Nodes performing coordination operations
- Worker Nodes
 - A node that performs tasks assigned by the master node.
- Daemon
 - A process or program that runs in the background

HBase Overview (1/2)

- Storing data in HBase table
 - Similar to RDBMS tables, but with key differences
- Tables are stored in HDFS
 - Data is partitioned into HDFS blocks and stored on multiple nodes in the cluster.
- Tables are made up of rows, columns, and column families.
- All rows include Row Key for quick retrieval
- Columns store data in a table
- Each column belongs to a specific column family
- A table contains one or more column families.

HBase Overview (2/2)

- Natively distributed, sorted maps
- Distributed
 - HBase is designed to use multiple machines to store and serve table data.
- Sorted Map
 - Store table data as a map, with contiguous keys stored contiguously on disk
- Multi-dimensional
 - Hundreds or thousands of columns potentially stored in a single column family

HBase Rows

- Row key is similar to primary key of existing RDBMS
- Sort and store rows by row key for quick data retrieval
 - Rows are sorted alphabetically by key
- Row Key is a string of 100 bytes or less
 - Ex) 1, 10, 2, ...

The diagram illustrates the structure of an HBase row. It features a blue header labeled "pinfo". To the left of the table, three vertical brackets are positioned: a top bracket labeled "Column Families", a middle bracket labeled "Column Names", and a bottom bracket labeled "Row s". The table itself has three columns: "Row Key", "fname", and "lname". The first row contains the "Row Key" "hkchang" and values "fname" and "lname". The second row contains the "Row Key" "jhjeong" and values "fname" and "lname". The third row contains the "Row Key" "thjtjeon" and values "fname" and "lname". Three arrows point from the text labels to their corresponding components in the table: one arrow points from "Column Families" to the "pinfo" header; another arrow points from "Column Names" to the "fname" and "lname" columns; and a third arrow points from "Row s" to the "Row Key" column.

pinfo		
Row Key	fname	lname
hkchang	Jean	Choi
jhjeong	Scott	Jeong
thjtjeon	Lonan	Jeon

HBase Columns

- Similar to RDBMS columns, but with different characteristics
- Columns are created when needed (vs. RDBMS)
 - Columns can be added at any time as long as the column family exists
 - A column exists only if the row contains data in that column.
 - A table cell (row-column intersection) is an arbitrary byte array (Arrays)
- Each column in a table belongs to a specific column family.
 - set of columns
- One or more column families exist in a table, written as a table definition
 - Include at least one
- Columns are grouped into column families

k			
			x
	x		

HBase Column Families

- All columns belonging to the same column family have the same prefix
 - Ex) pinfo:fname and pinfo:lname
 - ":" is used as a column name and column family separator
- Tuning and storage settings for each column family
 - Ex) The number of versions of each cell to be saved
- Column family can have multiple columns
 - Columns within a column family are sorted and stored together

HBase Tables

- Columns are referenced using the column family and column name.
- When using separate column families is useful
 - Infrequently accessed data
 - Data using different column family options
 - e.g., compression

		pinfo		pimage
Column Families	Column Names	Row Key	fname	Iname
Row s	hkchoi	Jean	Choi	
	jhjeon	Scott	Jeong	<scott.jpg>
	thjeon	Lonan	Jeon	<lonan.jpg>

Storing Data in Tables

- Data is physically stored on disk in units of column families.

pinfo		
Row Key	fname	Iname
hkchoi	Jean	Choi
jhjeon	Scott	Jeon
jtjeon	Lonan	Jeon

File

- The table is the value, the contents of the cell

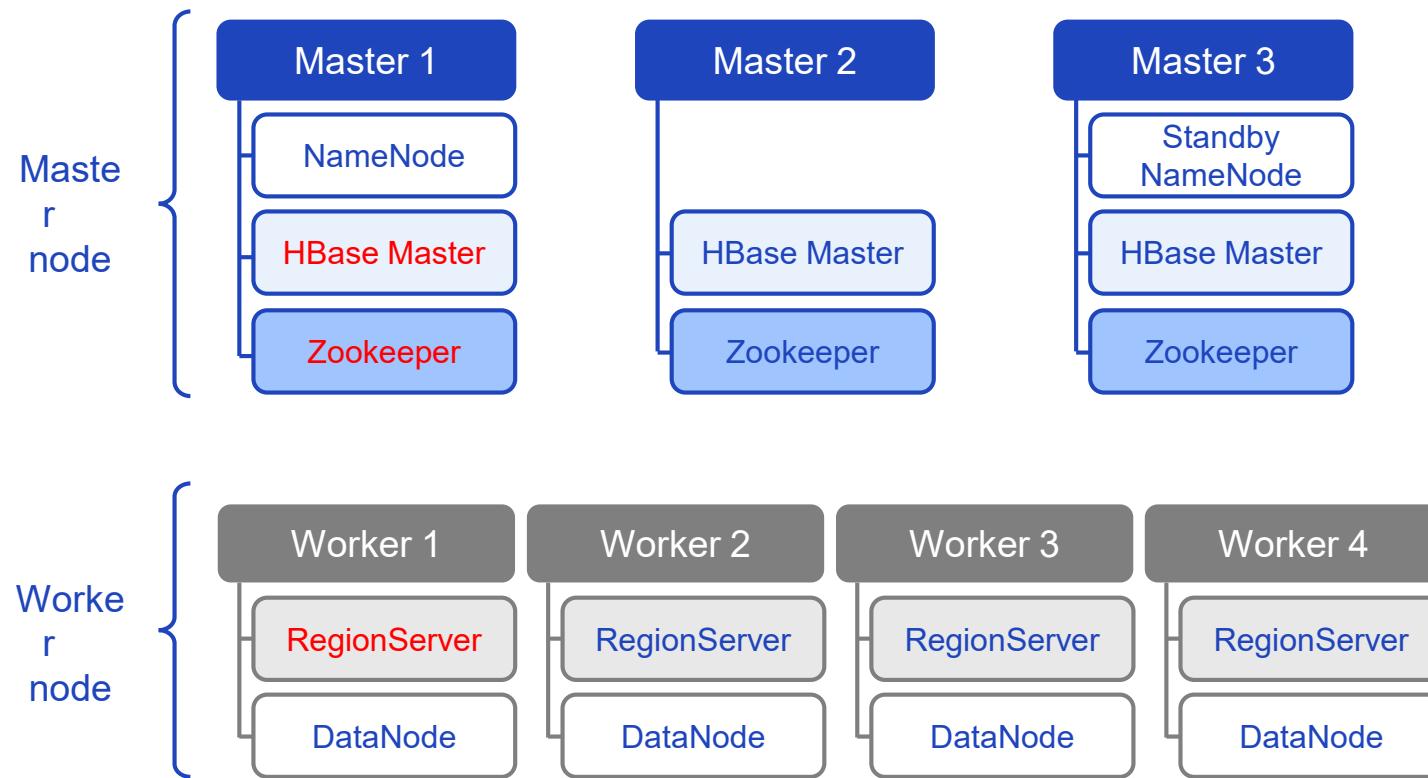
- Row key

Row Key	Column	Timestamp	Cell Value
hkchoi	pinfo:fname	102638498231	Scott
jhjeon	pinfo:Iname	1026753402919	Jeong

HBase Components

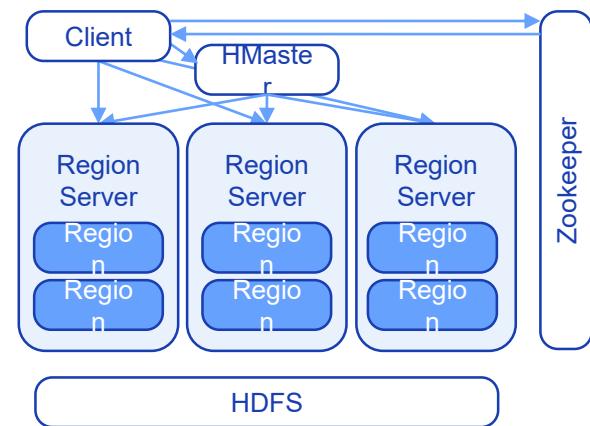
- RegionServer
 - Server servicing and managing Regions
- HBase Master
 - Monitor all RegionServer instances on the cluster interface for all metadata changes
- ZooKeeper
 - Coordination framework used to centrally manage configuration information for HBase
 - Decision support for a single Master
- NameNode
 - Management process to manage HDFS metadata
- DataNode
 - A daemon process that stores and holds HDFS blocks.

Daemon and Service in HBase Cluster



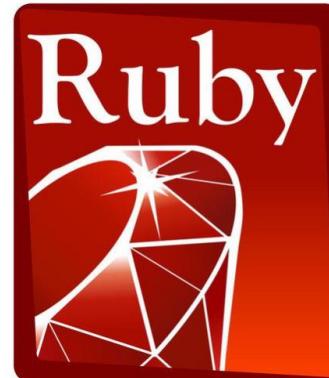
HBase Regions and Region Servers

- HBase table split into Regions
 - Table partition
 - Similar to existing RDBMS sharding and partitioning
- Regions are provided to clients by the RegionServer process.
- RegionServer typically runs on the cluster's working node
- Datanodes usually have several region servers running.
- RegionServer usually serves multiple regions
 - Provided Region belongs to several different tables.
 - It is unlikely that a single RegionServer will serve all regions of a particular table.



HBase Shell basics

- The HBase shell handles interactive commands to pass commands to HBase.
 - Use HDFS
 - Hmaster, region server, region interworking
 - Most of the operations available in the API can be done in the shell.
- HBase shell uses JRuby
 - Wrapping Java client calls in Ruby (Ruby programs in JVM)
 - Use Ruby syntax for commands
 - The way parameters are used is different from other shells.
 - Command parameters are enclosed in single quotes (').



Programming
Language

HBase Shell usage (1/2)

- HBase shell execution

```
$ hbase shell
```

The screenshot shows the Cloudera Manager Instances page for the HBase service. The top navigation bar includes tabs for Status, Instances, Configuration, Commands, Charts Library, Table Statistics, Audits, HBase Web UI, and Quick Links. The status bar indicates the date and time: Feb 18, 8:46 PM PST.

The left sidebar contains filters for STATUS, COMMISSION STATE, MAINTENANCE MODE, RACK, and ROLE GROUP. The ROLE GROUP filter is currently expanded, showing two entries:

Role Type	State	Host	Commission State	Role Group
Master (Active)	Started	quickstart.cloudera	Commissioned	Master Default Group
RegionServer	Started	quickstart.cloudera	Commissioned	RegionServer Default Group

- HBase Basic Command

- help, status, version, whoami
- Exit(terminate) + <Return>

HBase Shell usage (2/2)

- Interactive Ruby Shell

```
[cloudera@quickstart ~] $ hbase shell
20/02/18 20:37:01 INFO Configuration.deprecation: Hadoop.native.lib is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.0-cdh5.13.0, rUnknown, Wed Oct 4 11:16:18 PDT 2017

hbase(main):001:0> version
1.2.0-cdh5.13.0, rUnknown, Wed Oct 4 11:16:18 PDT 2017

hbase(main):002:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000 average load

hbase(main):003:0> exit
[cloudera@quickstart ~]$ hbase shell
20/02/18 20:37:58 INFO Configuration.deprecation: Hadoop.native.lib is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.0-cdh5.13.0, rUnknown, Wed Oct 4 11:16:18 PDT 2017

hbase(main):001:0>
```

Shell command (1/2)

- Shell commands can take parameters
 - <return> after entering the command
 - Unlike the Bash shell, single quote (' ') is used.
 - Multi-parameter calls require curly braces, key name, value in single quotes, and comma (,)

```
Hbase> command 'param1', 'param2'
```

```
hbase(main):001:0> create 'test', 'pinfo'  
0 row(s) in 1.5330 seconds
```

```
=>Hbase::Table - test  
hbase(main):002:0> list 'test'  
TABLE  
test  
1 row(s) in 0.0310 seconds  
  
=> ["test"]
```

Shell command (2/2)

- Table creation and alteration uses Ruby hashes
 - {'key1' => 'value1', 'key2' => 'value2', ... }
 - The “=>” operator is a hash rocket and separates keys and values.
 - Hash must end with another curly brace

```
create 't1', {NAME => 'f1', VERSIONS => 2, BLOCKCACHE => true}
```

- Shell command type
 - General commands
 - Data Definition commands (create, list, drop, alter...)
 - Data Manipulation commands(Put, Get, Scan...)

Shell scripting

- HBase shell runs in interactive and batch mode
 - Write a script with Jruby and pass it to the shell
 - Passed as a parameter when executing hbase shell or from the shell

```
$ hbase shell rubyscript.rb
```

```
hbase> require 'rubyscript.rb'
```

HBase Operations on Table

- Data Definition Command
 - Create
 - List
 - Disable, Enable
 - Describe
 - Alter
 - Exists
 - Drop

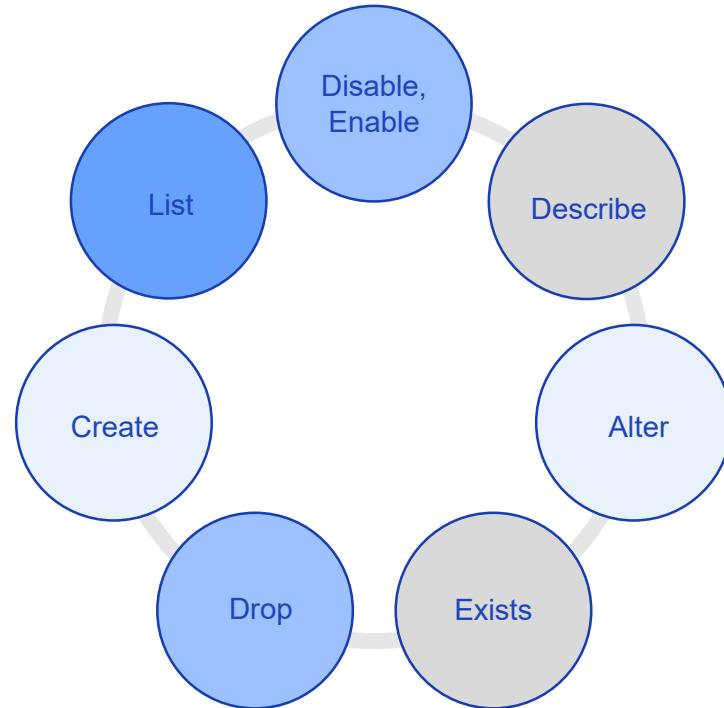
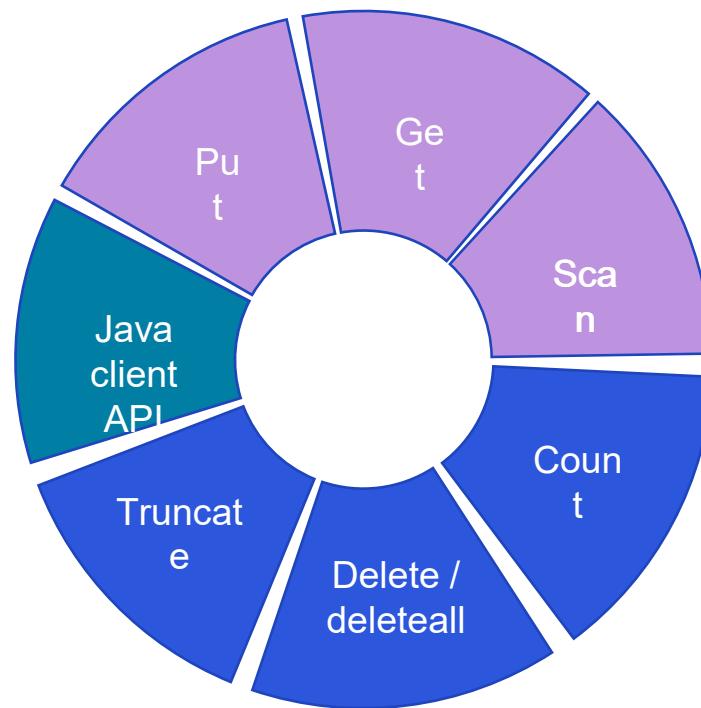


Table creation

- Table name, column family, options, and table configuration
 - Required specification of table and column family when creating a table
 - Create ‘<table name>’, ‘<column family>’
 - At least one column family is required for every table.
 - The column is a simple string or dictionary and must contain the NAME attribute.
- Tables can be grouped through the new namespace feature
 - The concept of “database” and “schema” in relational databases
- Difference from RDBMS
 - No need to create columns or relationships
 - No need to specify key relationships or constraints

Data Manipulation command

- Put
- Get
- Scan
- Count
- Delete / deleteall
- Truncate
- Java client API



Create Data

- Put
 - enter a new row
 - Used to change existing data
 - If the row key exists, it modifies the existing value and timestamp.
 - If a column family is defined to keep only one version of each column value, only keep the new value.
 - If n versions are defined to be kept, then n values are kept, including new values.
 - Updates to a specific column do not change other columns in the row

```
Put 'table name', 'rowkey', 'columnfamily:column', 'value' [,timestamp]
```

```
hbase> put 'test','1','pinfo:name', 'Jean'  
hbase> put 'test','2','position:title', 'Engineer'
```

Read Data

- Get
 - Get the most recent version of the row or column contents corresponding to the row key
 - Specify the exact row key to be fetched

```
get 'table name', 'rowkey' [, options]
```

```
hbase(main):008:0> get 'test', '1'
COLUMN                                CELL
    pinfo:city                      timestamp=1582167226850, value=Seoul
    pinfo:name                       timestamp=1582166931093, value=Jean
    position:title                   timestamp=1582167310073, value=Manager
3 row(s) in 0.0210 seconds
```

```
hbase(main):009:0> get 'test', '2', {COLUMN => 'pinfo:name'}
COLUMN                                CELL
    pinfo:city                      timestamp=1582167455859, value=Scott
1 row(s) in 0.0050 seconds
```

```
hbase(main):010:0> get 'test', '3', {COLUMN => ['pinfo']}
COLUMN                                CELL
    pinfo:name                      timestamp=1582167555280, value=Lonan
1 row(s) in 0.0060 seconds
```

Scan & Count Data

- Scan
 - If you do not know the exact row key
 - If you need access to a row-group
- Full table scan
 - Can be restricted to start line and stop line keys
 - Stop rows are not included in the results, only the previous one is searched.
 - Can be restricted to specific columns and column families

```
scan 'table name' [, options]
get 'table name', 'rowkey' [, options]
```

- Count
`count 'table name'`
Count the number of rows

Delete Data

- delete
 - Delete a specific cell, column, or column family from a table
 - Adding a timestamp deletes that version's column and all previous versions

```
delete 'table name', 'rowkey', 'column family:column name', timestamp
```

```
hbase> delete 'test','1','position:title', 1582167310073
```

- deleteall

```
Deleteall 'table name', 'rowkey'
```

Versions

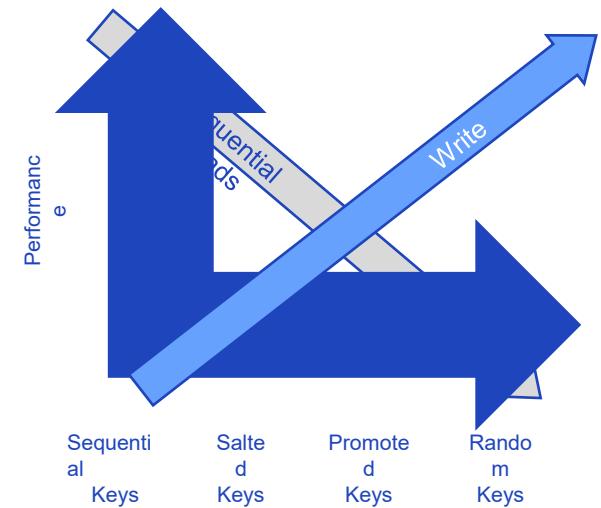
- HBase stores versions of values in each cell
 - The number of versions is determined when setting up the column family.
 - The version is identified by a timestamp of type long, and the current timestamp is used as the default.
 - The default number of cell versions is 3

Row Key	Column	Timestamp	Value
1	pinfo:title	1582172284996	Engineer
1	pinfo:title	1582167455859	Manager
1	pinfo:title	1582167226850	Engineer

- Value
 - Row Key Column family Column Version

Key design tradeoff

- Salted
 - Place a small, calculated hash in front of the real data to randomize the row key
 - For example, <salt><timestamp> instead of just <timestamp>
- Promoted Field Keys
 - A field is moved in front of the incremental or timestamp field
 - For example, <sourceid><timestamp> instead of <timestamp><sourceid>
- Random
 - Process the data using a one-way hash like MD5
 - For example, <md5(timestamp)> instead of just <timestamp>



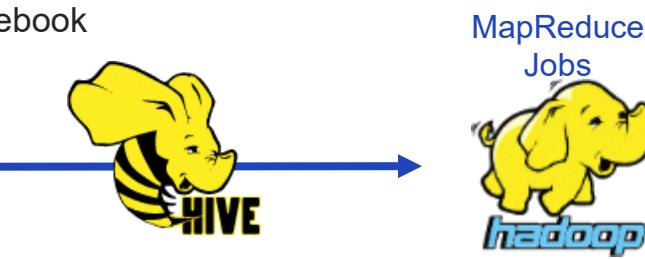
VERSIONS & TTL(Time-To-Live)

- By default, one cell version is kept.
 - HBase automatically inserts a timestamp to each version of the cell.
 - Only the most recently committed cells are kept.
- The TTL attribute is used as an expiration time (sec) for value of cell.
 - The Rows are retained until the user deletes them, but automatically, expired rows are deleted.
 - The TTL attribute can be set for each column family.

Hive and HBase (1/2)

- Hive
 - Used to access HDFS data using SQL-like syntax
 - During Hadoop processing, it is changed to actual Map-Reduce and executed.
 - Using Map-Reduce and Tez engine
 - Data processing / [Analytics Service developed by Facebook](#)

```
SELECT id, pwd, addr, gender, acct_num
      FROM users
     JOIN accounts
       ON accounts.acct_num = users.acct_num
      WHERE addr = 'Seoul'
    ORDER BY id DESC;
```



- HBase code is usually written in java or other languages using thrift
 - Simple analysis using the HBase shell

Hive and HBase (2/2)

- Requirements to use HBase
 - A good programmer who understands the code
 - Understanding how HBase works
 - Human resources who can write and maintain program API code
- How to use it easily with tools
 - How to store and retrieve data without needing to know the details of the HBase API
 - Available via Hive/Impala/presto

Using Hive with HBase

- Hive execution in beeline
 - Create “layers” table as external table

```
$ beeline -u jdbc:hive2://localhost:10000

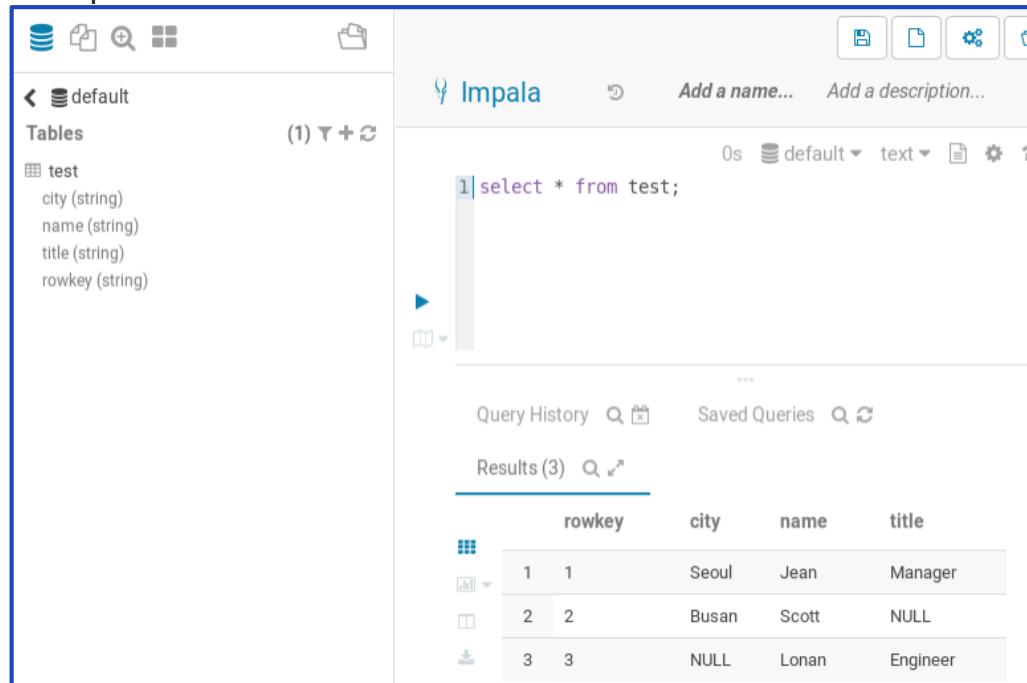
beeline> CREATE EXTERNAL TABLE test
  (rowkey string, name string, city string, title string)
STORED BY
'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping"
= ":key,pinfo:name, pinfo:city, position:title")
TBLPROPERTIES
("hbase.table.name" = "test");

beeline> DESCRIBE test;

Beeline> select * from test;
```

Using Impala with HBase

- Impala execution in Hue Query Editor
 - Impala share metastore with Hive



The screenshot shows the Hue Query Editor interface. On the left, the sidebar displays the 'default' database and a table named 'test' with columns: city (string), name (string), title (string), and rowkey (string). The main area is titled 'Impala' and contains the following query:

```
1|select * from test;
```

Below the query, there are buttons for 'Query History' and 'Saved Queries'. The results section shows three rows of data:

rowkey	city	name	title
1 1	Seoul	Jean	Manager
2 2	Busan	Scott	NULL
3 3	NULL	Lonan	Engineer

Unit 2.

NoSQL

- | 2.1. NoSQL Overview
- | 2.2. Apache HBase
- | 2.3. Cassandra
- | 2.4. MongoDB

Cassandra Definition

Apache Cassandra is an **open source, distributed, decentralized, elastically scalable, highly available, fault-tolerant, tuneably consistent, column-oriented** database that bases its distribution design on Amazon's Dynamo and its data model on Google's Bigtable. Created at Facebook, it is now used at some of the most popular sites on the Web

The Definitive Guide, Eben Hewitt,



Bigtable,
2006



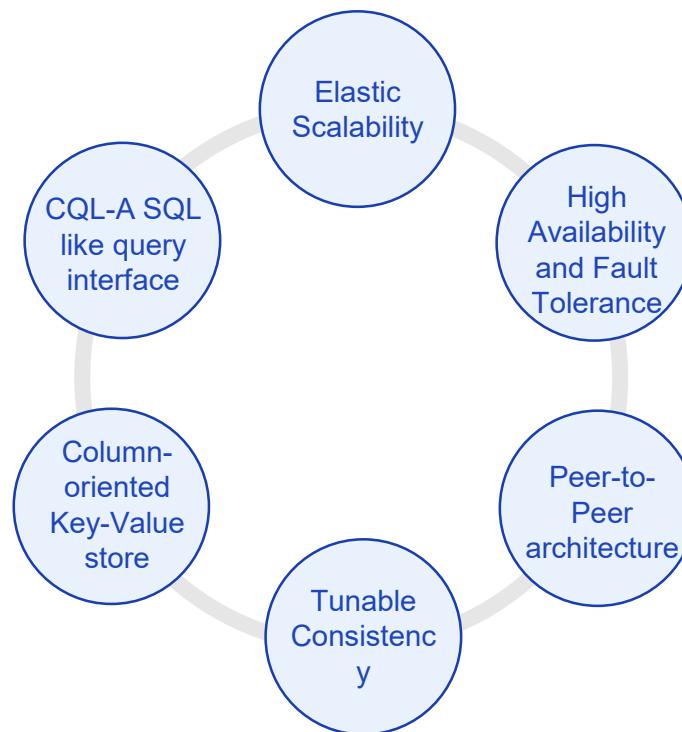
OpenSource,
2008



Dynamo,
2007

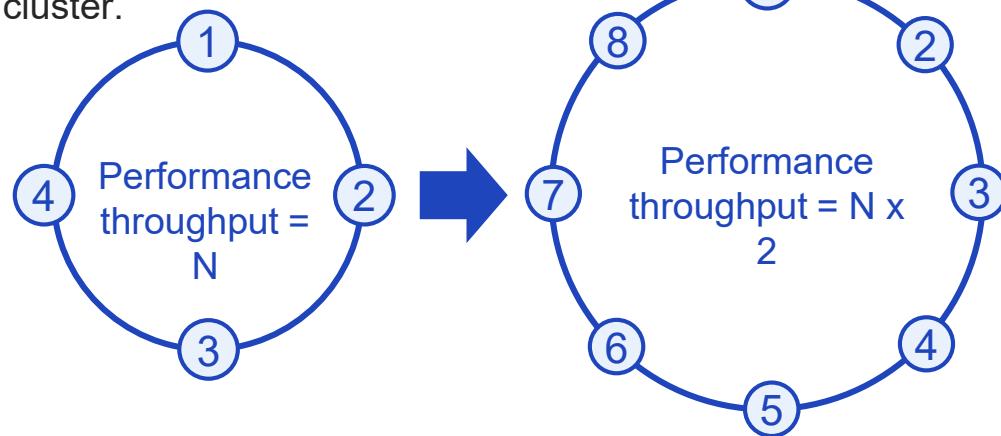
Key Features

- Cassandra is a distributed data store optimized for scalability and high availability.
 - Ring and Gossip protocol
 - Failure response without sharding and master-slave structure

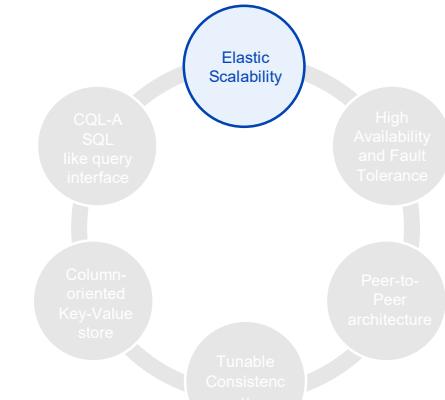


Elastic Scalability

- Cassandra scales out to add systems containing all or part of the data
- Adding nodes increases performance throughput linearly
- It is easy to increase and remove the number of nodes without reconfiguring the entire cluster.

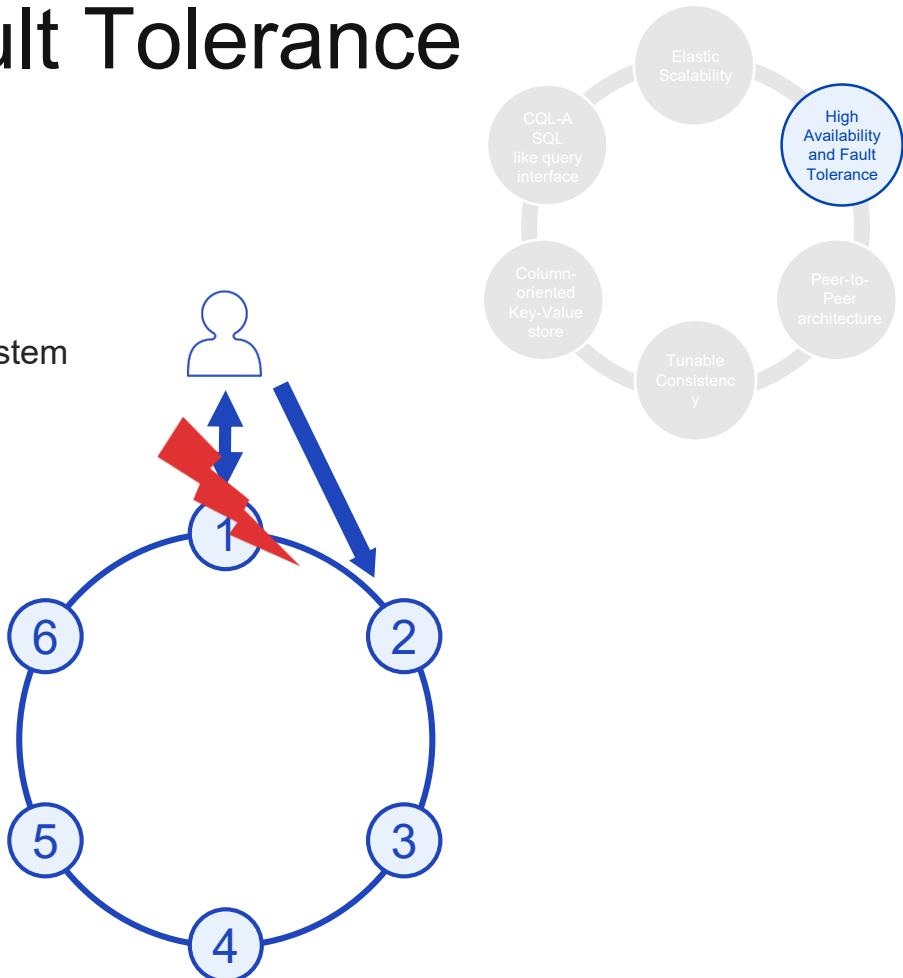


[Linearly scales to terabytes and petabytes of data]



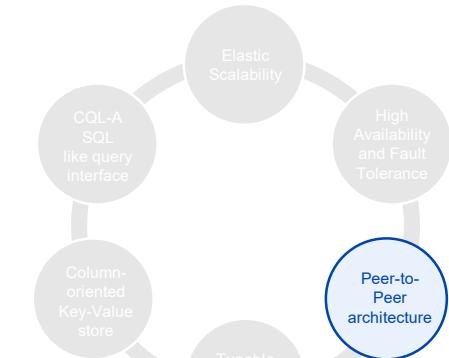
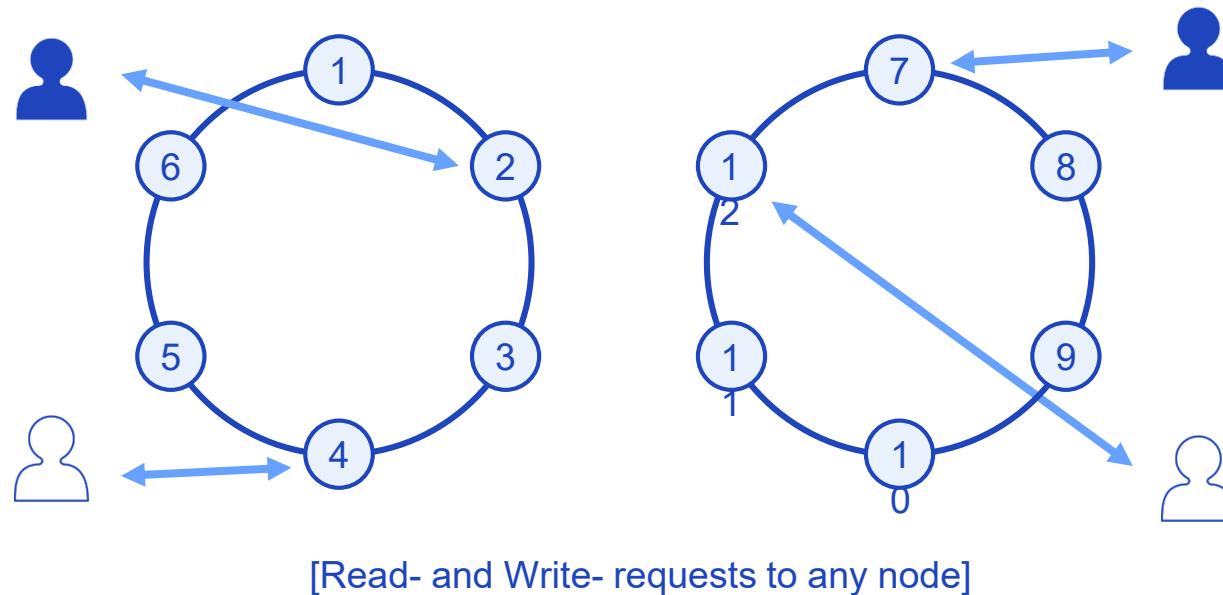
High Availability and Fault Tolerance

- Cassandra High Availability
 - Multiple network computers working in a cluster
 - Node failure awareness
 - Failover by forwarding requests to other parts of the system



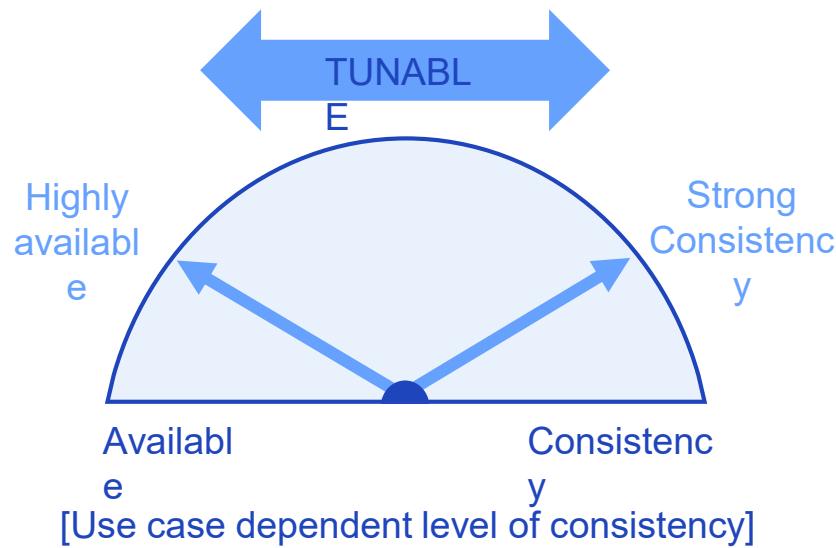
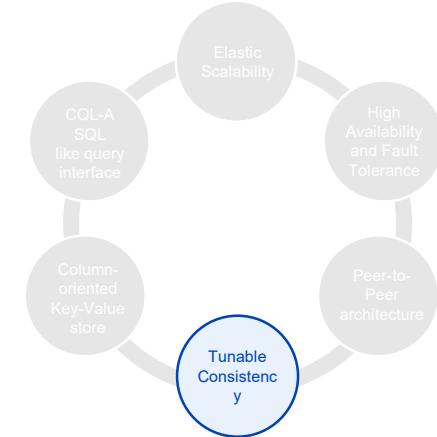
Distributed and Decentralized

- Decentralized: SPOF does not occur
 - No master-slave problem with peer-to-peer structure (protocol "gossip")
 - A single Cassandra cluster can run in geographically dispersed data centers



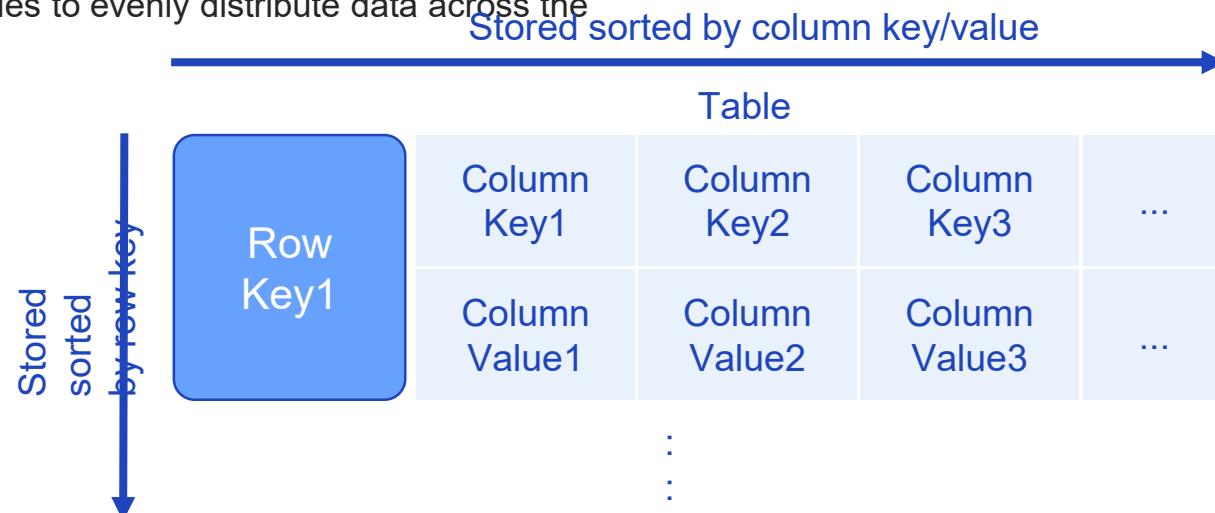
Tunable Consistency

- Choose between Strong / eventual (Available vs Consistency) consistency
- Read (read operation) and Write (write operation) can be adjusted separately
- Slightly sacrificing Consistency to increase overall Availability.
 - Adjust the Consistency Level to balance availability



Column-oriented Store

- Data is stored in multiple hash tables
- A row contains multiple columns, but each row need not have the same number of columns.
- Each row has a unique key, which is used for partitioning
 - Row keys give hash values to evenly distribute data across the cluster.
- No Relations for tables



CQL- An SQL-like query interface

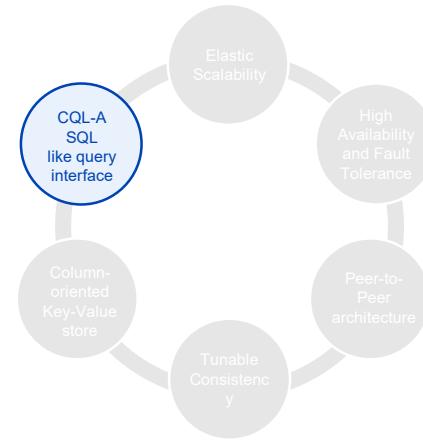
- CQL is the main interface to Cassandra DBMS.
- Maps to the Cassandra storage engine and uses a SQL-like syntax to simplify data modeling

```
CREATE TABLE song_col (
    PRIMARY KEY,
    id uuid
    title text,
    album
    text,
    artist text,
    data blob,
    tags
    set<text>
);
```

```
INSERT INTO song_col (
    id, title,
    artist, album, tags)
VALUES(
    'a3e64f8f...', 'La
    Grange',
    'ZZ Top', 'Tres
    Hombres'
    {'cool',
    'hot'});
```

[“SQL-like” but NOT relational SQL]

```
SELECT *
FROM song_col
WHERE id = 'a3e63f8f...';
```



Where to use Cassandra

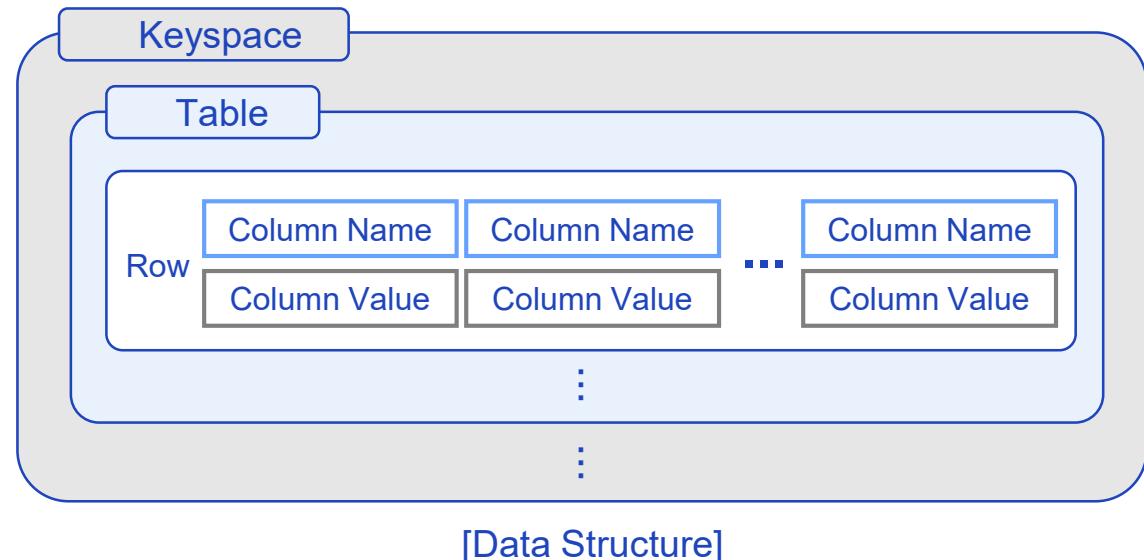
- Simple setup, maintenance, and ability to search code
- Very fast reads/writes operations
- If you don't need multiple secondary indexes
- When various column types are required
- Where NOT to use Cassandra
 - Transaction
 - Stringent Security and Authorization Needs on Data
 - Dynamic Queries on Columns

Cassandra Terminology

- Data Center – A unit that constitutes a cluster with a set of nodes
- Commit log – Save local disk by evacuating failure during write operation
- MemTables – Storing write operation data in temporary memory buffer, flush to SSTable when space is full
- SSTable (Sorted string table) – Immutable, sequential data disk storage
- Gossip – A node periodically exchanges state information about other nodes with which it is connected.
 - Peer to peer protocol, run every second
 - Exchange status messages with up to 3 other nodes
- Bloom filter – An algorithm that tests whether an element is a member of a set

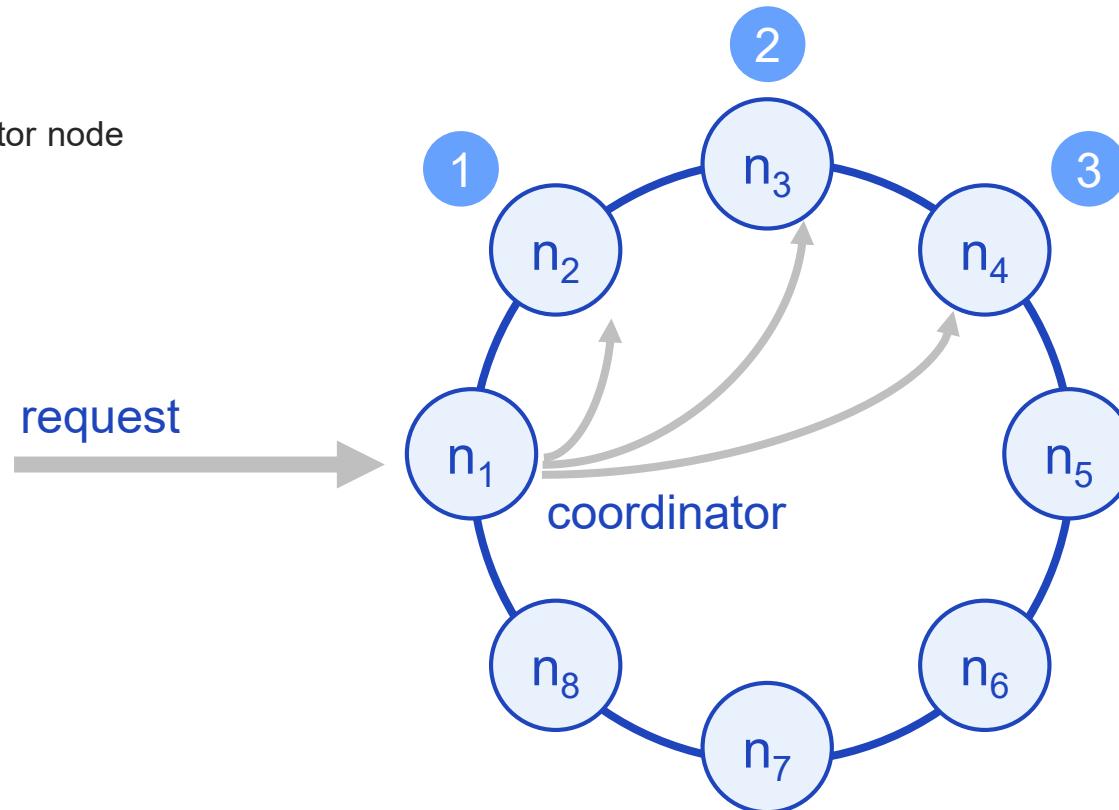
Cassandra Data Structure

- Cassandra data model
 - Keyspace > Table > Row >
column name : column value
 - Keyspace acts as a data container
- Cassandra Cluster
 - Distributed storage of data in each node in the form of a ring
 - The distribution criterion is the partition key
 - Distributed based on data hash value



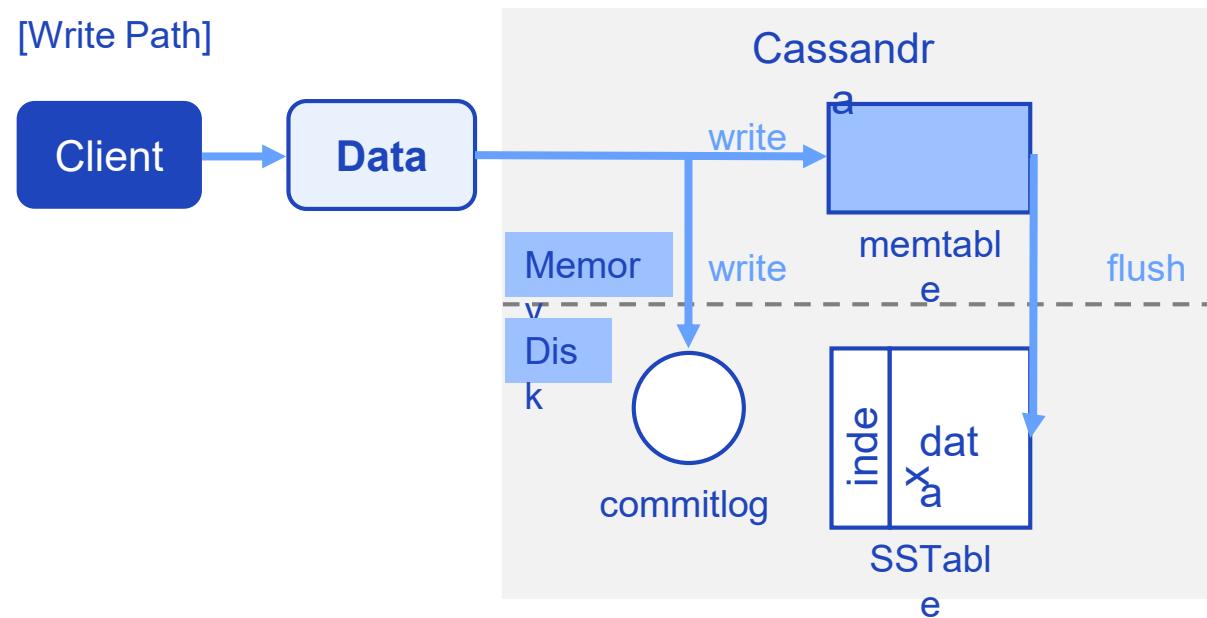
Write/Read data flow

- Coordinator node
 - Handling client requests
 - Any node can be a coordinator node



Write Path (1/2)

- Add to Commit log on disk and save to memTable
- When memTable is full, it is flushed to SSTable through I/O.
- After flushing, the data in the commit log is deleted.



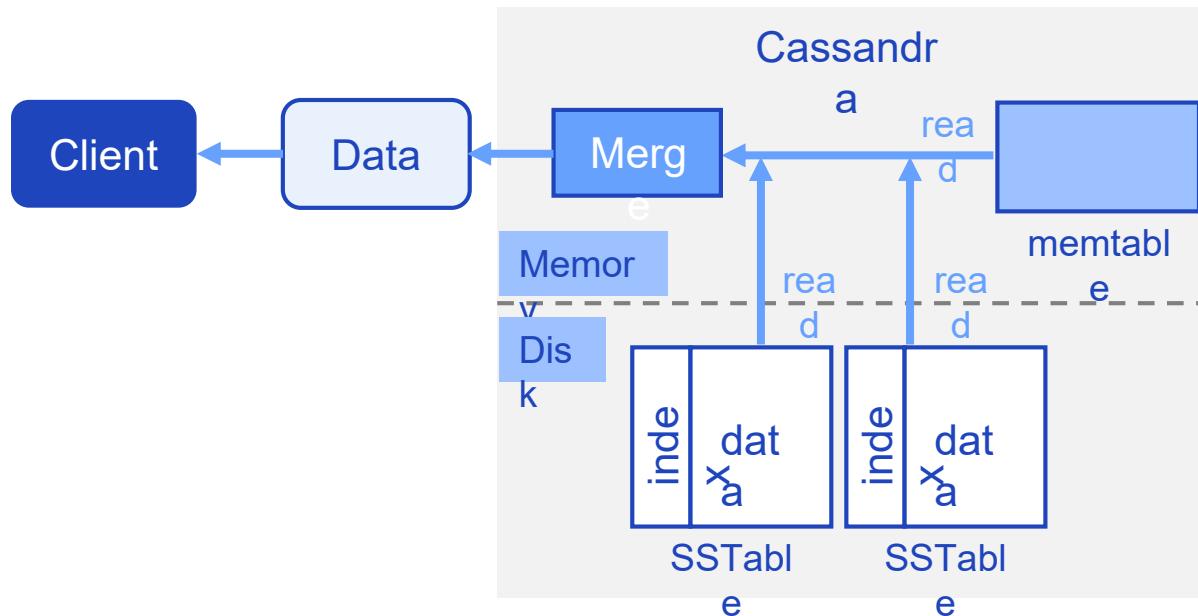
Write Path (2/2)

- Rows are usually stored in SSTable files
- SSTable includes primary index and Bloom filter
 - Primary index: A list of row keys and row start positions in the data file
 - Bloom filter: A subset of the primary index to improve performance
- Write consistency level: The number of replicas that acknowledge successful writes.



Read Path (1/2)

- When a read operation is requested on a node, it is merged from all related SSTables and all unflushed memTables
- The merged values are stored in the write-through row cache.



Read Path (2/2)

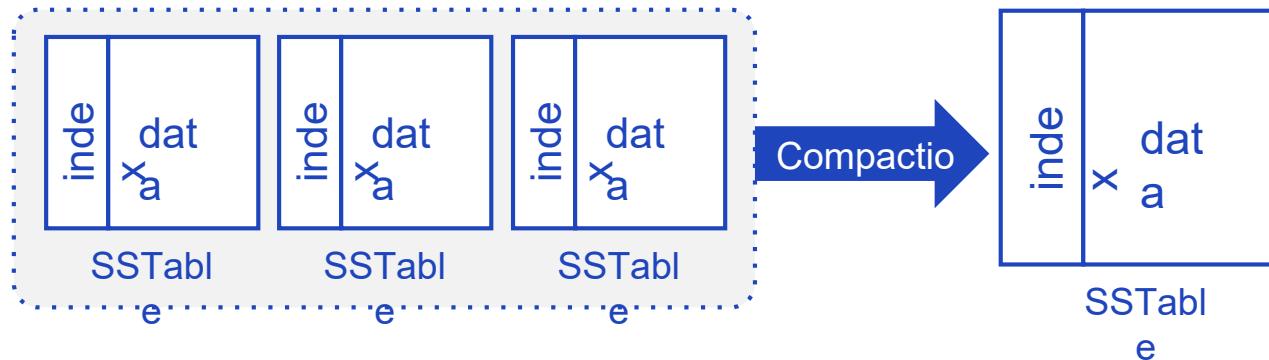
- Read Consistency level: The number of replicas accessed for successful consistent read operations.



- Read Repair
 - Coordinator accesses multiple nodes for data (as much as consistency level)
 - The fastest replica returns data for consistency check through memory comparison.
 - The coordinator checks all remaining replicas in the background
 - Update for inconsistent replicas

Delete Data

- Tombstone
 - Marker of the row indicating the deleted column
 - Marked columns are not deleted immediately, but are deleted when SSTables are compacted over time.
 - Side-effect occurs (a node that recovers after a failure)
- Compaction
 - Merge and sort SSTables to create a new SSTable
 - Reduce the number of searches required to free up space and improve performance
 - Merge key, merge column, delete tombstone, create new index



What is CQL?

- CQL(Cassandra Query Language)
 - CQL is a basic language that leverages the Cassandra database and is similar to standard SQL.
 - Perform CRUD operations using CQL shell (cqlsh)
 - Commands end with a semicolon (;) to separate multiple commands.
 - Identifiers (names) are used to identify keyspaces, tables, columns, and other objects.
 - Names starting with an alphabet
 - SELECT and With keywords are case insensitive
 - (") - a quoted identifier defined by enclosing arbitrary characters in double quotation marks.

Data types

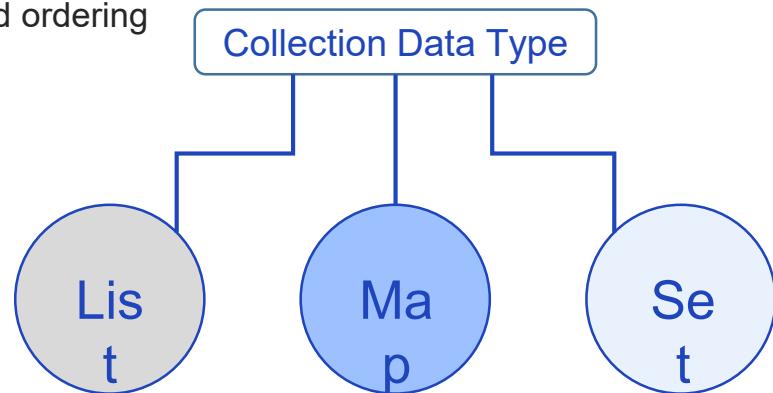
- Cassandra supports various data types
 - Built-in type
 - Collection type
 - User-defined type
 - Tuple type
 - Custom type

Built-in Data Type

Data Type	data	
Boolean	True, false	Use one of T/F
Blob	Binary large objects	written in hexadecimal
ASCII	65(A), 97(a)..	US-ASCII String
Bigint	-2^32 ~ 2^32	Only integers, 64bit
Counter	1, 2, 3....	Increment/decrement counter value (64bit integer)
Timestamp	2020-01-03 01:02:00+0000	Date and time with millisecond, +0000(GMT)
Double	-2^32 ~ 2^32	All real numbers, 64bit
Int	-2^16 ~ 2^16	integers
varchar	string	UTF8 encoded string

Collection Data Type (1/3)

- Map
 - An ordered collection of key-value pairs
 - Store Key in Name and Value in Value
- Set
 - set of one or more non-overlapping ordered elements
- List
 - A set of elements that allows for one or more overlapping and ordering



Collection Data Type (2/3)

- Map

```
Create table ucol1 (
    id      integer PRIMARY KEY,
    name    varchar,
    items   map<text, text>
);
```

```
Insert into ucol1 (id, name, items)
    values(1,'Jason Jeong', {'color' : 'blue', 'movie' : 'star wars'});
```

```
Update ucol1 set items = {'color' : 'green'}
    where id = 1;
```

```
Update ucol1 set items['fruit'] = 'apple' where
    id = 1;
```

Collection Data Type (3/3)

- Set

```
Create table img1 (
    id      integer PRIMARY KEY,
    title   varchar,
    tags    set<text>
);
```

```
Insert into img1 (id, title, tags)
    values(1,'Dog',{'black', 'pet', 'smile'});
```

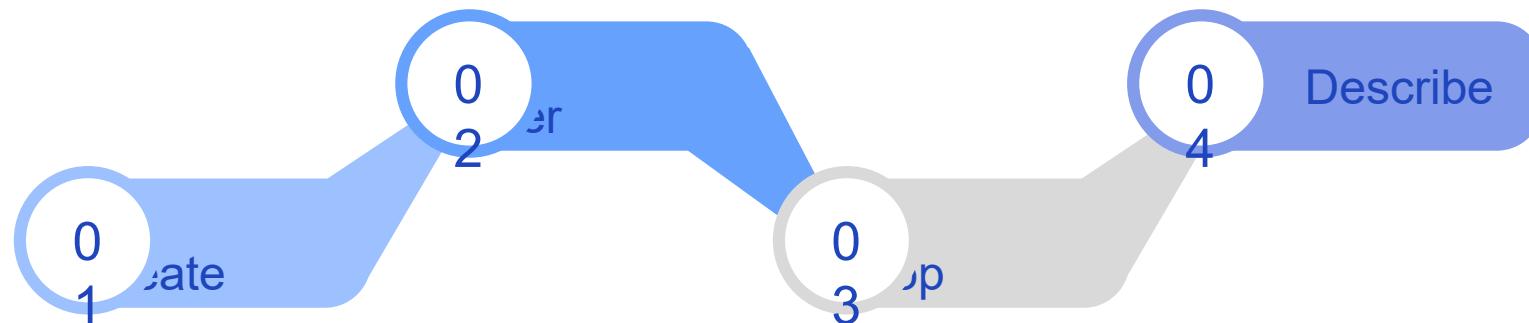
```
Update img1 set tags = {'white', 'pet', 'cute'}
    where id = 1;
```

```
Update img1 set tags = tags + {"gray", 'small'}
    where id = 1;
```

```
Update img1 set tags = tags - {"small"}
    where id = 1;
```

User-defined Data Type(UDT) (1/2)

- User-defined data types



- ```
create_type_statement ::= CREATE TYPE [IF NOT EXISTS] udt_name
 '(' field_definition (',' field_definition)* ')'
 field_definition ::= identifier cql_type
```

## User-defined Data Type(UDT) (2/2)

- Alter\_type\_statement

```
alter_type_statement ::= ALTER TYPE udt_name alter_type_modification
alter_type_modification ::= ADD field_definition |
 RENAME identifier TO identifier (identifier TO identifier)*
```

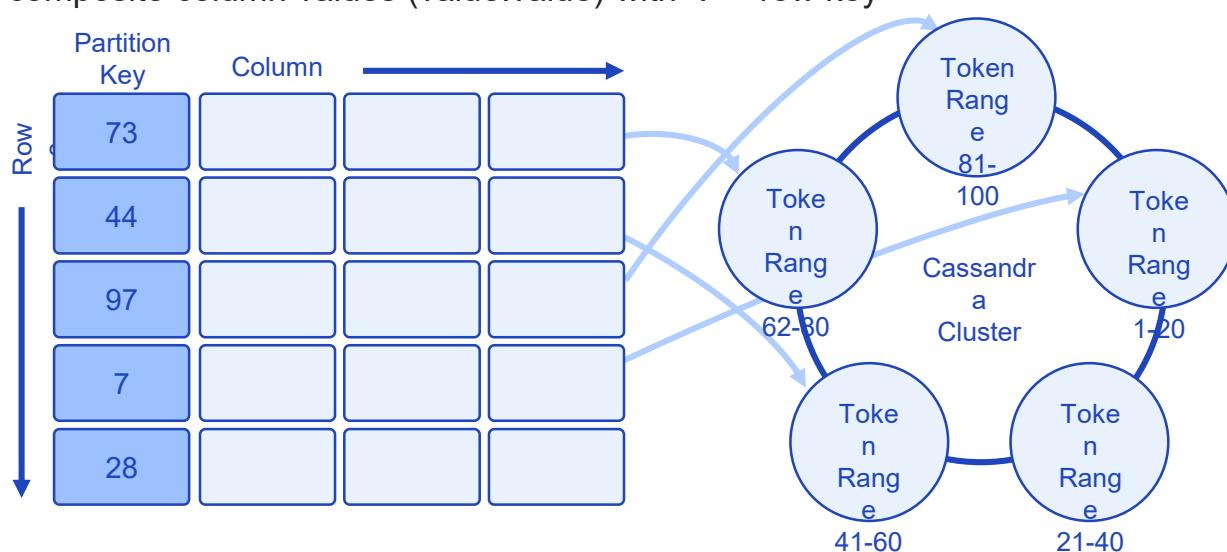
- Drop\_type\_statement

```
drop_type_statement ::= DROP TYPE [IF EXISTS] udt_name
```

- describe\_type\_statement ::= DESCRIBE TYPE udt\_name

# CQL Key (1/2)

- Partition Key
  - Unique key for distributed storage of data
  - When composing a table, one or more must be specified, and several can be specified.
  - Value in single column = row key
  - Combination of composite column values (value:value) with ":" = row key

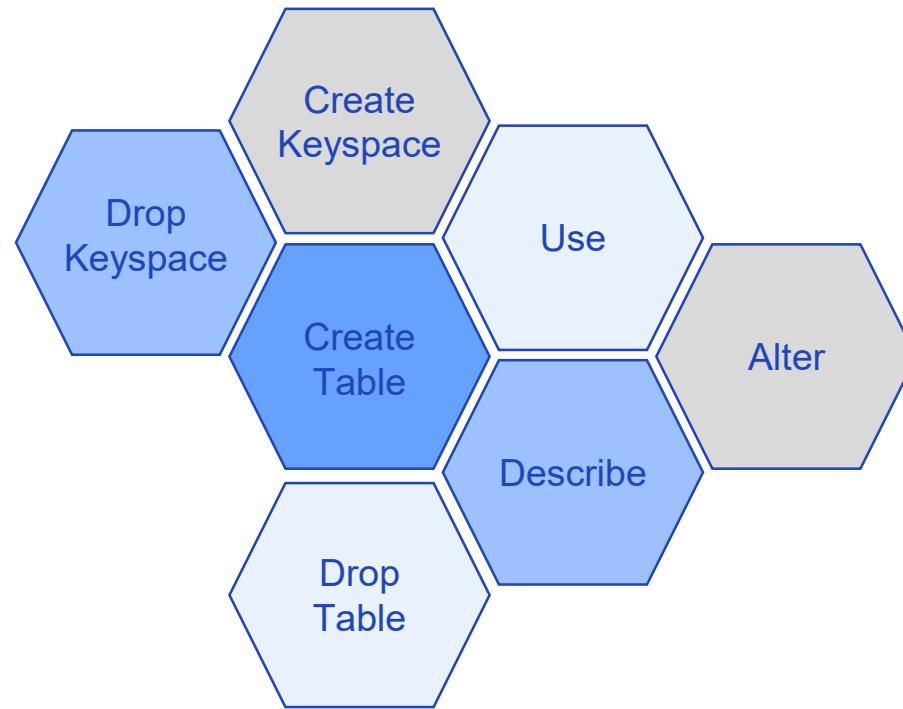


## CQL Key (2/2)

- Cluster Key
  - A key for sorting columns belonging to a row
- Primary Key
  - A column or column group that uniquely determines a row in a table
  - Partition key + cluster-key
  - $\text{PK}(a,b,c) \Rightarrow a / bc$
  - $\text{PK}((a,b),c) \Rightarrow a:b / c$
- Composite key (compound key)
  - A primary key composed of two or more CQL columns
- Composite partition key
  - A partition key composed of two or more CQL columns

# Data definition command

- Define creation, modification, and deletion of keyspaces and tables
  - Create keyspace
  - Use
  - Alter keyspace
  - Drop keyspace
  - Create table
  - Alter table
  - drop table
  - Describe



# Common definitions

- Table name and keyspace

```
keyspace_name ::= name
table_name ::= [keyspace_name '.'] name
name ::= unquoted_name | quoted_name
unquoted_name ::= re('[a-zA-Z_0-9]{1, 48}')
quoted_name ::= "" unquoted_name ""
```

- **CREATE KEYSPACE [ IF NOT EXISTS ]**  
**Keyspace\_name WITH options**

# Keyspace creation

- Keyspace ks\_test
  - Simple strategy - use 2 replicas for the whole cluster

```
>Create keyspace 'ks_test'
With replication = {'class':'SimpleStrategy',
'replication_factor' : 2};
```

- Keyspace ks\_test2
  - Define the number of replicas for each data center

```
>CREATE KEYSPACE ks_test2
WITH replication =
{'class':'NetworkTopologyStrategy', 'DC1':
'1', 'DC2': '2'}
```

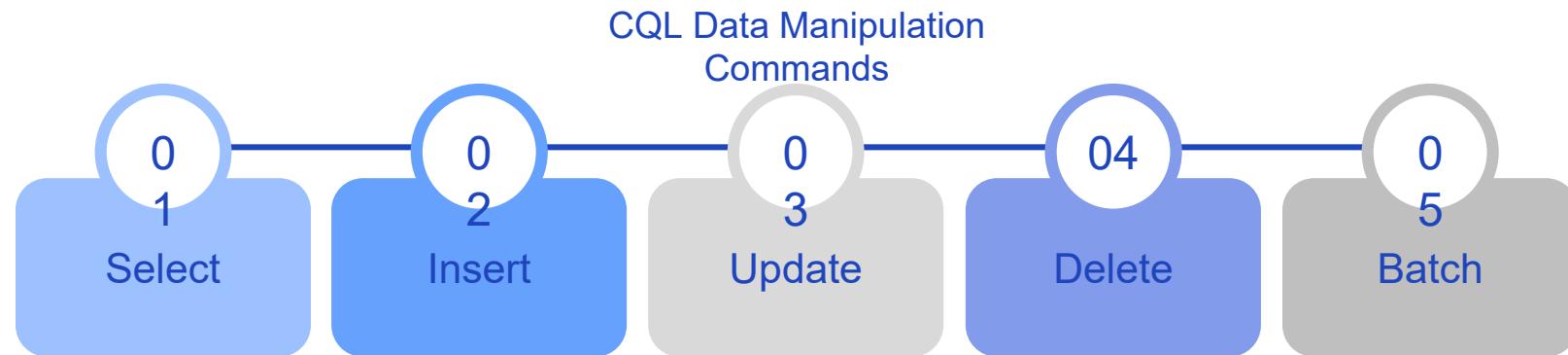
# Create Table

- Create new table

```
create_table_statement ::= CREATE TABLE [IF NOT EXISTS] table_name '('
 column_definition
 (',' column_definition)*
 [',' PRIMARY KEY '(' primary_key ')']
 ')' [WITH table_options]
column_definition ::= column_name cql_type
[STATIC] [PRIMARY KEY]
primary_key ::= partition_key
 [',' clustering_columns]
partition_key ::= column_name
| '(' column_name (',' column_name)* ')'
clustering_columns ::= column_name (',' column_name)*
table_options ::= COMPACT STORAGE [AND table_options]
 | CLUSTERING ORDER BY '(' clustering_order ')'
 [AND table_options] | options
clustering_order ::= column_name (ASC | DESC)
 (',' column_name (ASC | DESC))*
```

# Data manipulation command

- Commands to retrieve, delete, add, update, and batch data



# Cassandra Environment in the Exercise

- Cassandra installation & Operations
- Lab Environment
  - <http://cassandra.apache.org/download>
  - Latest version : 3.11
  - Linux - Centos 6.7, Ubuntu..
  - On the download page, select your operating system and the appropriate distribution
  - Java install required
  - Python 2.7 to use Cqlsh

Unit 2.

# NoSQL

- | 2.1. NoSQL Overview
- | 2.2. Apache HBase
- | 2.3. Cassandra
- | 2.4. MongoDB

# What is MongoDB?

- MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema. MongoDB is developed by MongoDB Inc.
- Document-oriented NoSQL database, written in C++
- 2007.10 Development started, shifted to an open source development model in 2009
- Developed as a component of a PaaS product by 10 gen. (MongoDB Inc)
- mongoDB (from “humongous” DB) is a scalable, high-performance database.



# MongoDB Overview

- Document Database with JSON type data storage structure
  - Store data in the same format as used in programming
  - BSON format (Binary JSON)
- Easy Scalability through Sharding
- High Availability using Replica Set
- Provides APIs in various languages
  - C, C#, C++, Java, JavaScript, Perl, PHP, Python, Ruby, Scala
- Schema-less
  - Each document stored in the database contains various fields.
  - Ideal for unstructured data
  - dynamical typed schema because it is not predefined

# Where to use MongoDB

- RDBMS replacement for web applications
- Semi-structured content management system
- Real-time analysis & high-speed logging (Logging)
- Big Data & Data Hub
- User Data Management
- Who are using MongoDB?



# MongoDB Common Terminology (1/2)

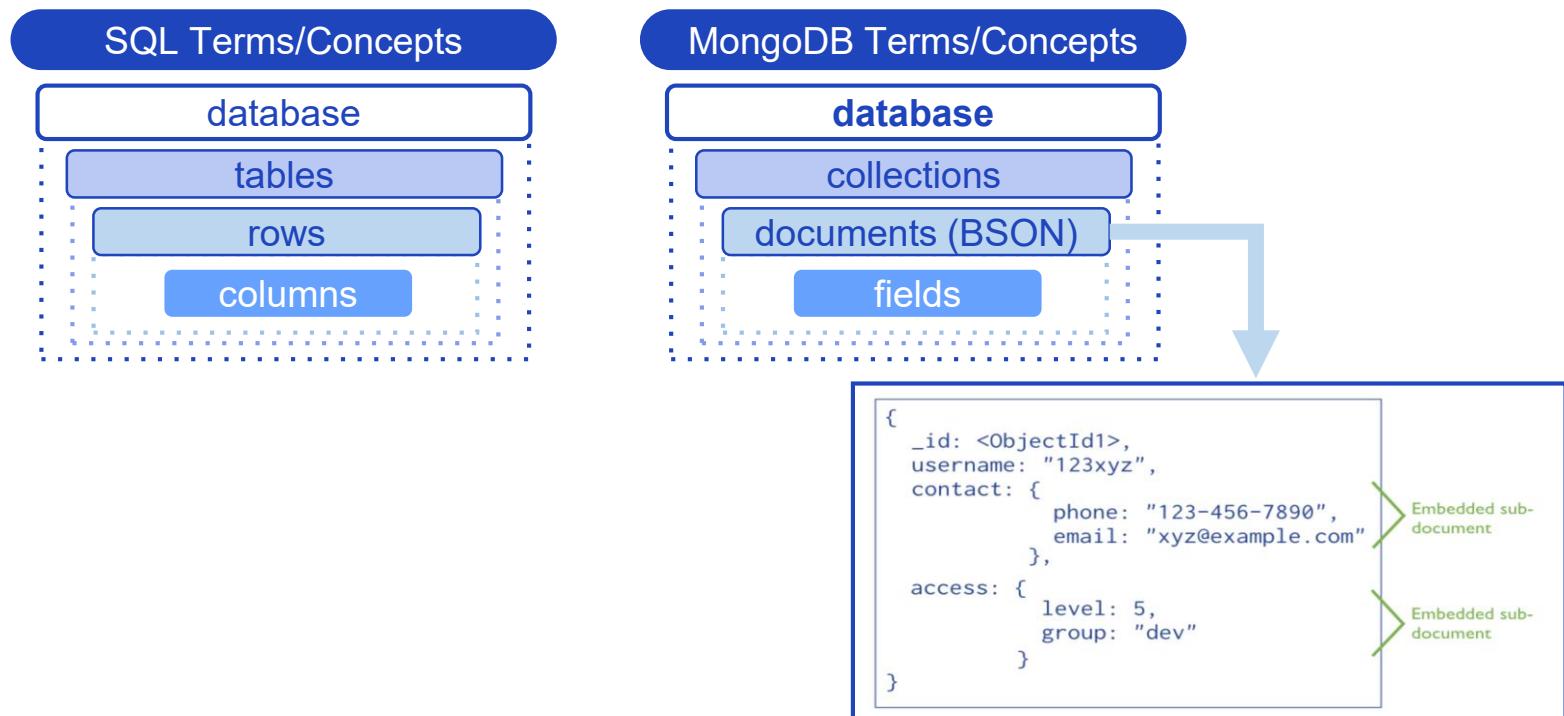
- \_id - Document's primary key
  - Fields required for every document
  - Unique value of Document
  - \_id is composed of time/machine ID/process ID/sequential number to ensure uniqueness
  - Automatically created if missing
- Collection – group documents
  - Table concept in relational database
  - exist in one database
  - Collections are not limited in structure
- Cursor - Pointer to the result set of a query
  - The client can iterate through the cursor to retrieve results

# MongoDB Common Terminology (2/2)

- Database - Container for collections
  - Each DB has its own set of files in the file system
  - MongoDB server stores multiple databases
- Document – Row concept in relational database
  - Records in MongoDB Collection
  - Document consists of field names and values (key-value)
  - Value can be different document, array, or document array
- Field - A name-value pair in a document
  - One or more fields exist in the document
  - Fields are similar to columns in a relational database.
- JSON - JavaScript Object Notation
  - A readable text format to represent structured data
  - Support for many programming languages

# Data model in MongoDB

- [Mongodb : db > collection > document(record) > key:value]



# MongoDB Features (1/3)

- Queries:
  - Supports ad-hoc queries and document-based queries
- Index Support:
  - All fields in the document can be indexed
- Replication:
  - Master-slave replication
- Maintaining Multiple Data Replicas
  - Preventing database hangs



# MongoDB Features (2/3)

- Multiple Servers:
  - The database runs on multiple servers and replicates data to protect the system in case of hardware failure.
- Auto-Sharding:
  - Distributed storage of data in shards (physical partitions) and automatic load balancing with sharding
- MapReduce:
  - Distributed/parallel support through a combination of Map and reduce functions



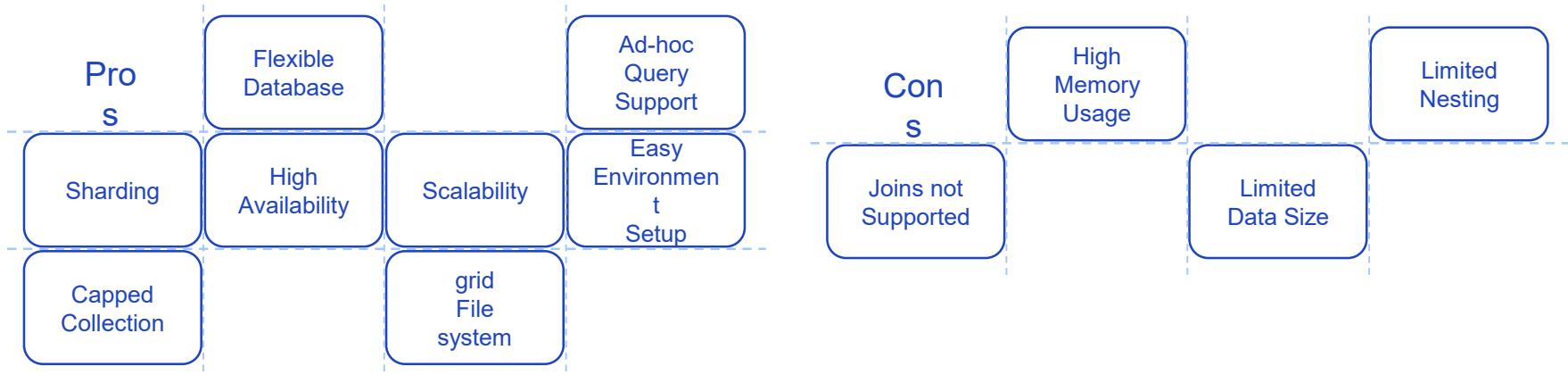
# MongoDB Features (3/3)

- Failure Handling:
  - A large number of replicas ensure data availability in the event of a system failure, data center failure, or network segmentation failure
- GridFS:
  - Split the file into smaller parts and save them as separate documents
- Schema-less:
  - Schema is not defined in advance, and dynamically stores various data when needed.
- Document-Oriented Storage:
  - A document-type database using BSON format, storing one document per row



# Why MongoDB

- Pros & Cons

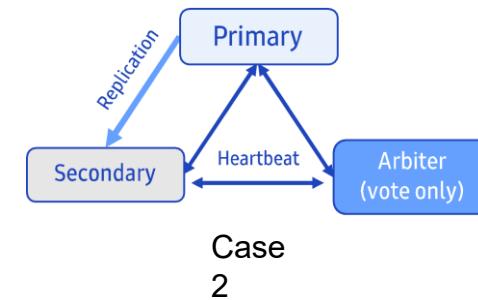
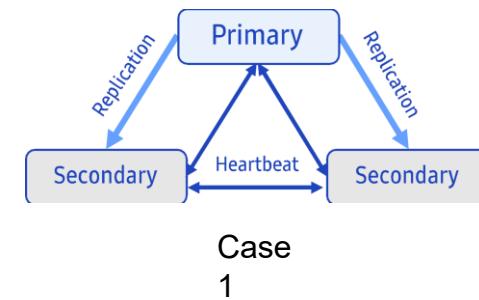
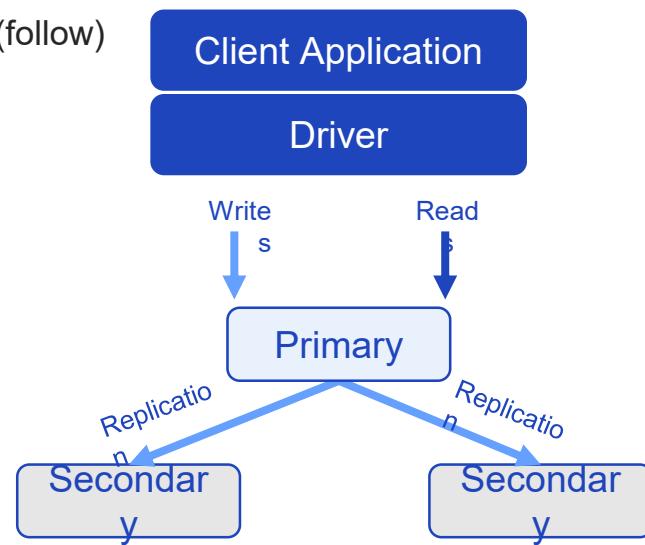


# Ad-Hoc Query

- Supports various query types such as SQL Query of RDBMS
  - Field
  - Range query
  - Regular expression search
  - Exact match
  - User-Defined Functions Made with Java Script
  - Search conditions can be flexibly specified
  - From searching through regular expression, it is possible to check whether a specific value is included in the array data.

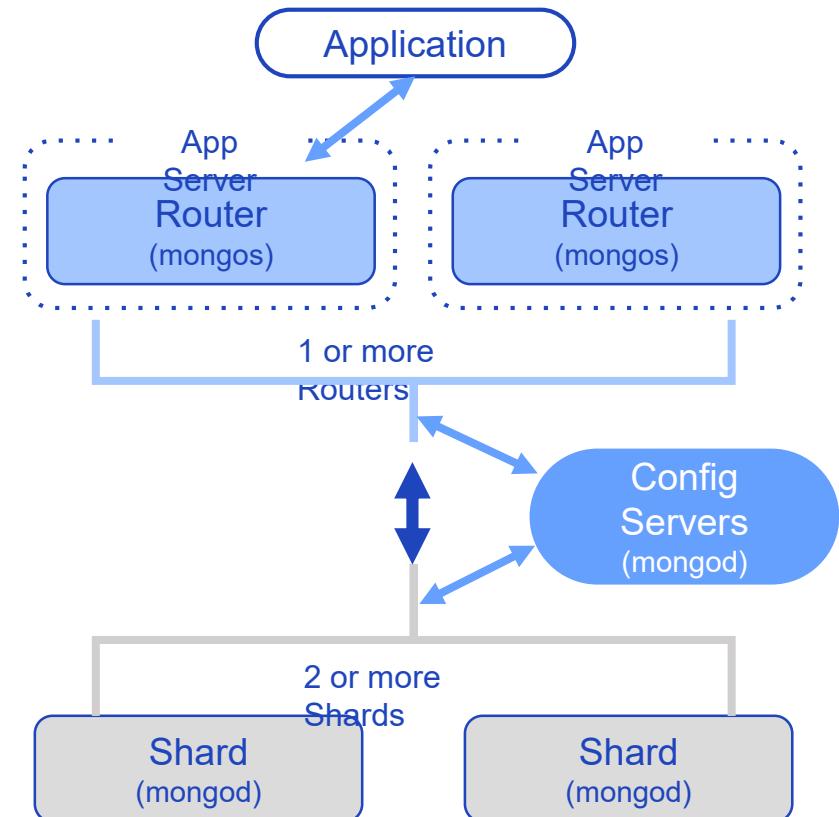
# Replication(Replica set)

- Distributed management of the same data to multiple servers for high availability in case of server and network failure
  - Primary node (Leader)
  - Secondary node (follow)
  - Arbiter node



# Sharding

- Sharding distributes and stores and processes data across multiple servers.
- Shard Cluster
  - Config Server – stores meta information such as data range and shard location information
  - Router Server (mongos) – acts as a proxy to perform queries
  - Shard – Partitioned data subset, can be deployed as a replica set



# Shema-less

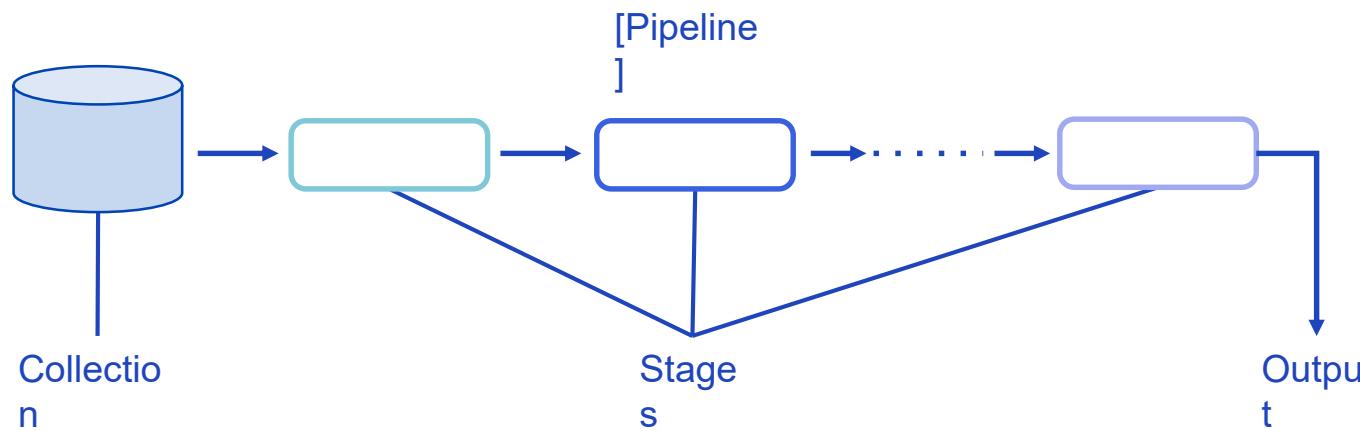
- There is no predefined schema
  - The application determines the data structure
  - Data design can change frequently
  - Store data at any time when you need it without defining the column to be used in advance
  - All data can be stored in one collection without normalization like an RDB table
  - Various types of data can be stored
  - Easy to change data model, expand fields

# Aggregation framework

- Aggregation operations process data records and pass calculated results
  - Process in pipeline order
- Group values from multiple documents together and perform various operations on the grouped data to deliver a single result
- Execution method
  - Aggregation pipeline
  - Aggregation framework is modeled according to the concept of data processing pipeline
  - Map-Reduce function
  - Single purpose aggregation operations
  - Operations that combine documents in a single collection
  - `db.collection.count()`, `db.collection.distinct()`

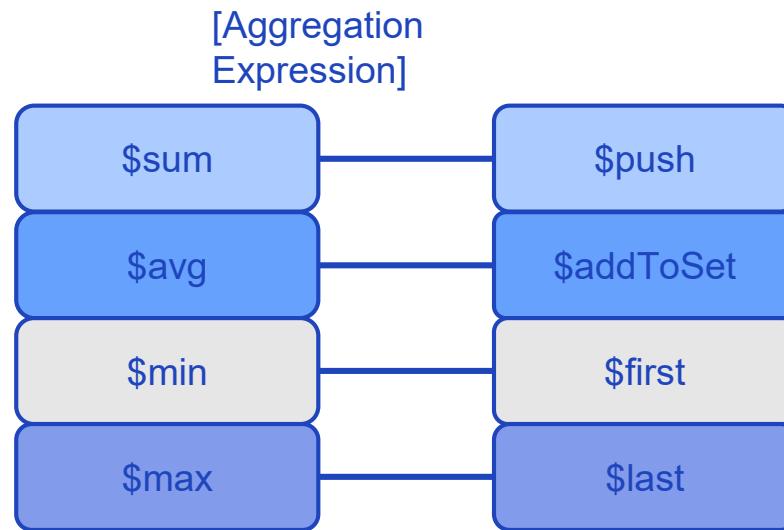
# Aggregation pipeline

- Pipeline
  - Stream the document and make it into multiple states
  - Each stage transforms the document through the pipeline.
  - Some stages create new documents or filter documents.
  - 7 states - \$project, \$match, \$group, \$sort, \$skip & \$limit, \$first & \$last, \$unwind



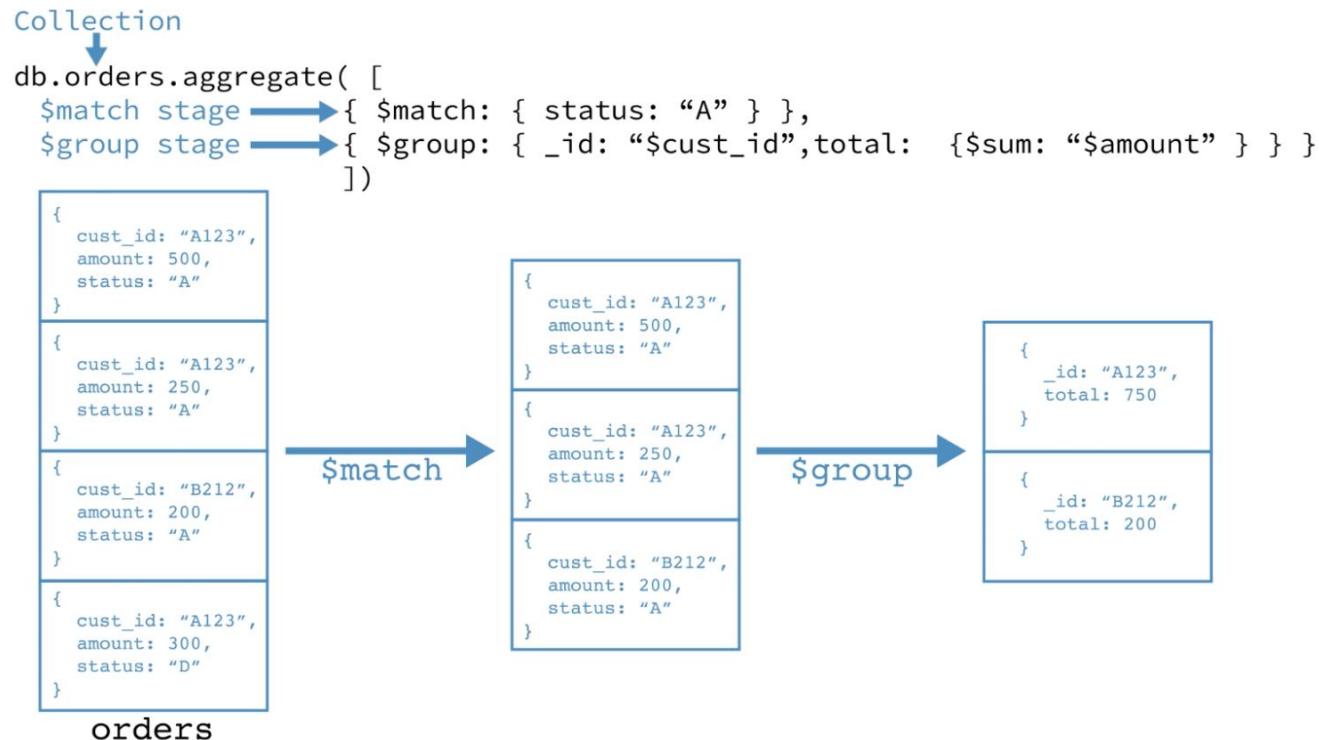
# Aggregation Expression

- Expression
  - Create output document based on input document expression



# Aggregation example

- Aggregation - Combination of Stage (pipeline) and expression



# Indexing

- A data structure for quickly retrieving records from a given table without having to traverse all the records in the table.
  - Supports quick search by arranging the order of data in advance
  - Limit the number of documents that need to be scanned in a collection (vs. scan)
  - Binary Tree Structure
- Only one index is valid per query
- If you need two indexes, use a composite index.
  - In a composite index, the order of the keys is important.
- Default index
  - `_id` creates a unique index on the `_id` field while creating the collection
  - The `_id` index prevents the client from inserting two documents with the same `_id` value.

# Types of index in MongoDB (1/2)

- Single field index
  - User-specified separately in addition to the `_id` index
  - Provides custom ascending/descending index creation for a single field in one document
- Compound index
  - Composite indexes support user-defined indexes on multiple fields
  - The order of each field is important - first sort the first field, then the second
- Multikey index
  - Index with Array and embedded document as field
- Geospatial index
  - Provides a special index to provide efficient querying of geographic coordinate data

## Types of index in MongoDB (2/2)

- Text index
  - Indexes that support searching for strings in collections
  - Text indices contain fields whose values are strings or arrays of string elements
- Hashed index
  - Use a hash data structure rather than a B tree
  - If key data is distributed non-uniformly, hash index values remain consistent.
  - Multi-key hash indexes are not allowed, query by range is not allowed

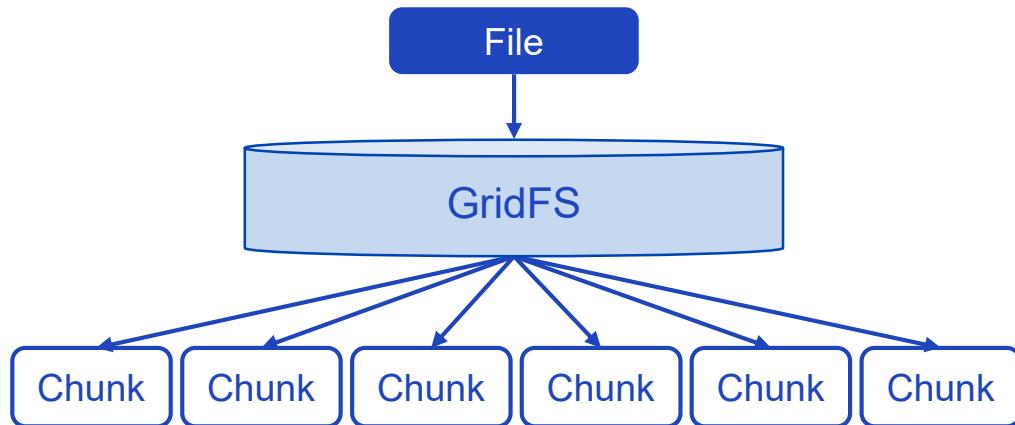
# Capped Collection

- Store data only within the space initially created with a limited size
  - Fixed size collection
  - A collection that is reused again when the first space is used up
  - Suitable for data that is stored and managed only within a certain period, such as log data.
  - Documents are saved in the order in which they are saved, and when full, the oldest document is deleted and a new document is stored (Queue method)
  - Size option required
- Document cannot be deleted or updated
- Document insert speed is fast, but find operation depends on insertion order

```
db.createCollection("capped_collection",
 {capped: true, size: 100000})
```

# Grid File system (1/2)

- Manage files that exceed the BSON document size limit of 16 MB
  - Suitable for large binaries/images, when there is a limit on the number of files stored in the file system
  - Instead of storing the file in a single document, split it into several smaller chunks and store each chunk as a separate document.
  - Inappropriate when updating the contents of an entire file atomically



# Grid File system (2/2)

- By default, GridFS stores files in two collections using prefix ‘fs’.
  - fs.files – file meta information
  - fs.chunks – binary chunks
- GridFS Index
  - Chunks index - use unique and compound indexes for files\_id and n fields
  - Files index – use the file name and uploadDate fields to index the file collection

## [Lab3] Data Access with Hbase



## [Lab4] Data Accessing Using DML commands



## [Lab5] Working with Hbase



## [Lab6] Data Access with Cassandra



# [Lab7]

## Working with Cassandra



Chapter 5.

# **Big Data Analytics**

Big Data Course

# Chapter Description

---

## Objectives:

- ✓ We will learn how to use various tools for data analytics in a Hadoop platform.
  - SQL is the language of queries. We will gain a comprehensive understanding of this language.
  - In data analysis, there are different ways that we want to analyze data.
  - Streaming real-time data, batch mode historical data and interactive ad-hoc queries
  - These use-cases often require different tools to process them. We will start with batch-mode and interactive mode in this chapter.
- ✓ We will gain insight to two main architectures to handle concurrent analysis of real-time and historical data
  - We will see why it is important to analyze historical and real-time data together.
  - Lambda architecture develops and deploys two separate systems to process them while Kappa attempts to overcome the difficulty of developing and maintaining two separate systems.

## Contents of Chapter:

1. Introduction to SQL
2. Basic Analytics
3. Advanced Analytics

Unit 1.

# Introduction to SQL

Big Data Analytics

Unit 1

# Introduction to SQL

- | 1.1. Creating tables using DDL
- | 1.2. Manage table data with DML
- | 1.3. Query data with SQL

# What is DB?

- A set of data
- An organically organized set of logically related data
- A set of common data that multiple applications can store and operate integrated information
- A set of information that is integrated and managed for shared use
- Characteristic
  - Realtime accessibility – real-time processing
  - Continuous evolution
  - Concurrent sharing
  - Content reference – referenced by data value

# What is DBMS?

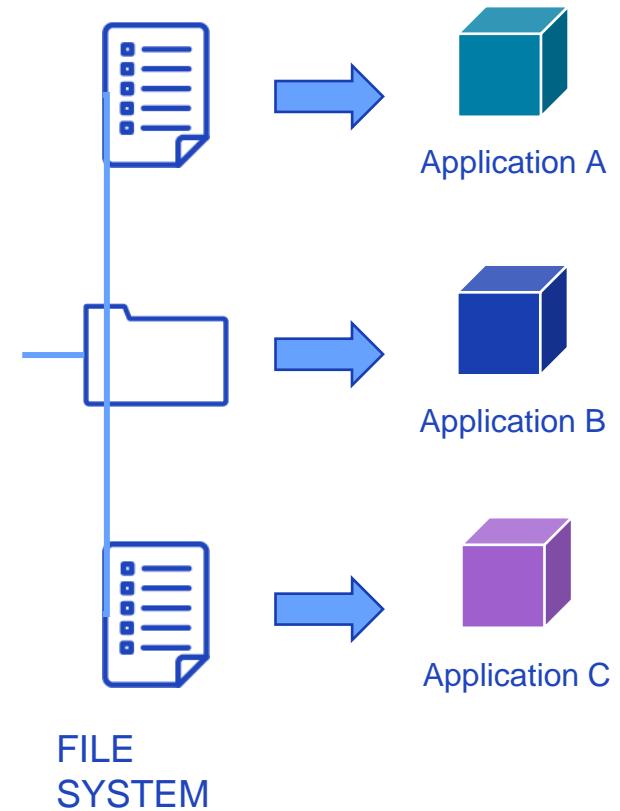
- A DBMS, a database management system, is a program that is made up of various programs that easily create and manage databases.
- Oracle, MySQL, SQLServer, PostgreSQL, MariaDB...
- Since most DBs are created and operated through DBMS, DB and DBMS are sometimes used with the same meaning, but the two are distinctly different expressions, but it is generally acceptable to treat them the same

# Types of database systems

- File system
- Hierarchical DB
- Network DB
- Relational DB
- Object oriented DB
- NoSQL

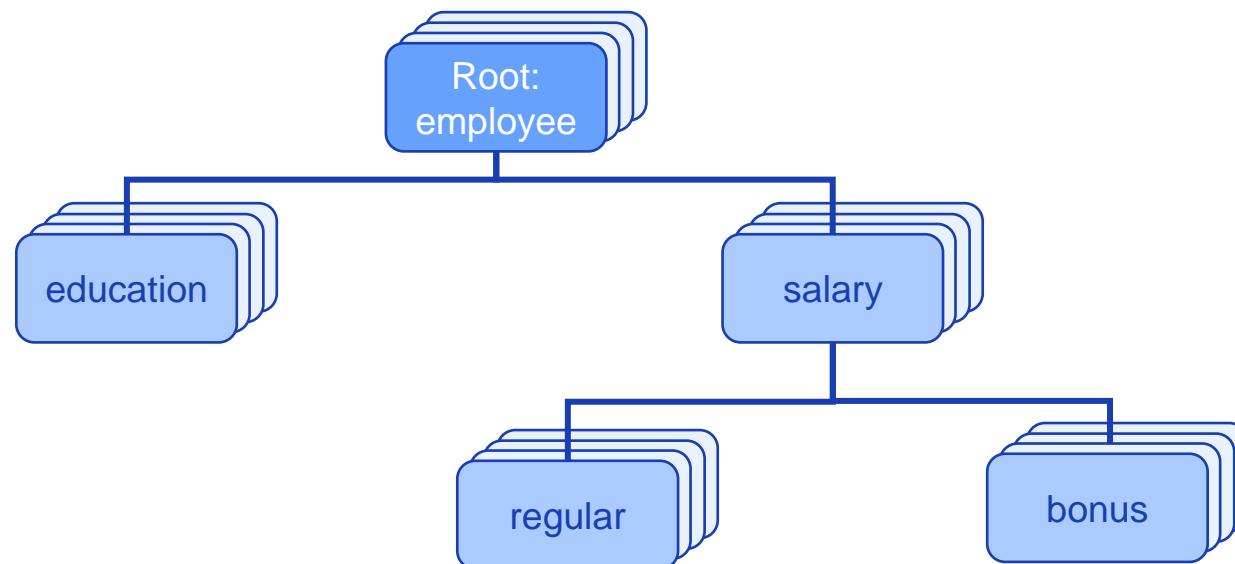
# FILE SYSTEM

- For data processing, they are arranged according to a certain rule and saved as a file on the disk.
  - The initial database starts with the file system
  - A file system is basically a way of arranging files on a storage medium such as a hard disk.
  - When the records stored in the file are read one by one, the necessary processing is repeated until the target record is displayed.
  - Redundant data can be present in a file system.
- In order to process data that exists in the file system, pointers must be used.
- Complex or ad-hoc query functions are not supported because record unit processing must be repeated.



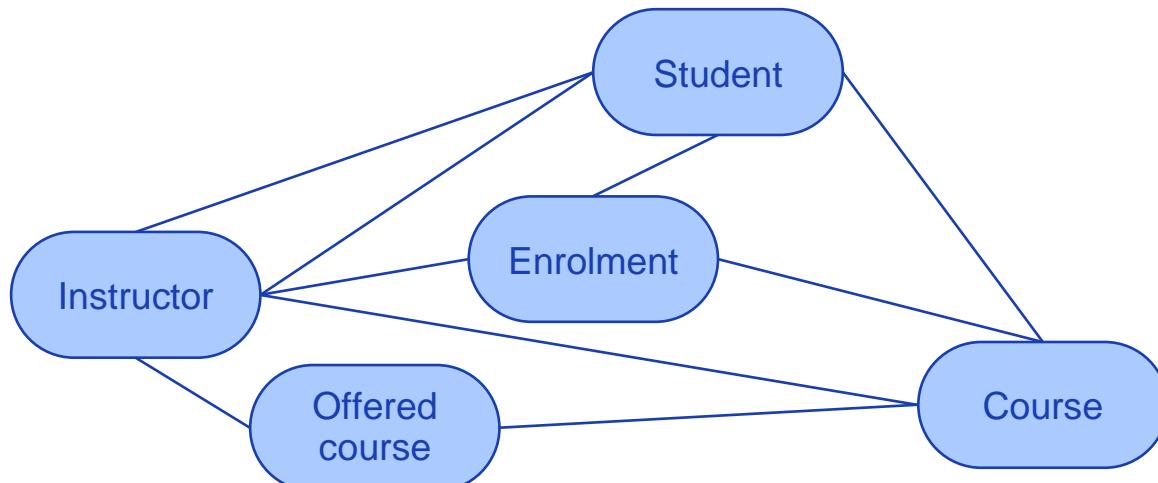
# Hierarchical DB

- Data is hierarchical and structured in hierarchical and subordinate relationships
  - Advantages: Data access speed is fast, and data usage can be easily predicted.
  - Disadvantages: It is not easy to accommodate the process that changes after initial setting as it is composed of a subordinate relationship.



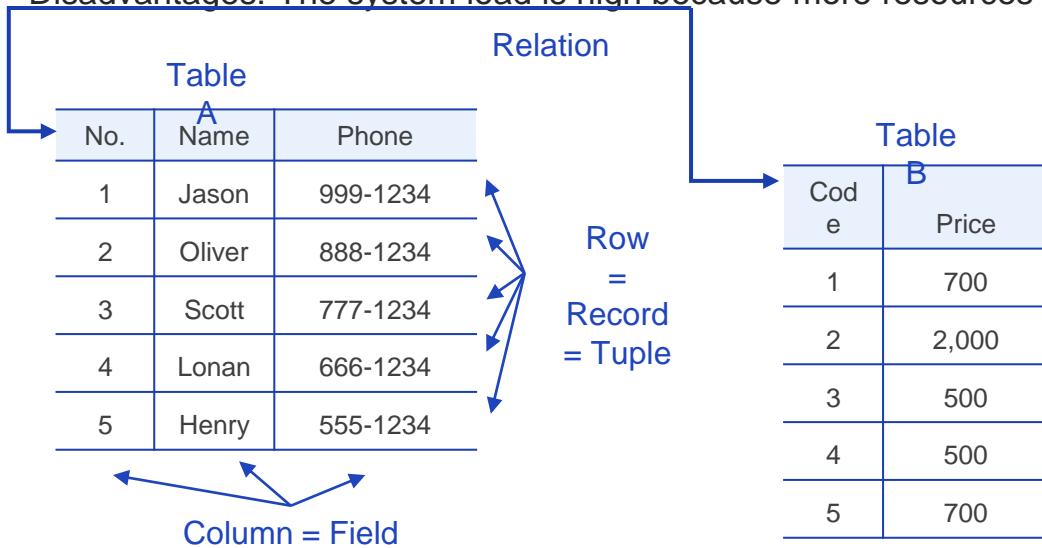
# Network DB

- A data model that logically expresses the data structure in the form of a node on the network, and a system in which each node is configured in an equal relationship (here, node refers to data, not system)
  - Advantages: Solving the disadvantages of hierarchical DB. It can contain various relationships between data entities.
  - Disadvantages: Complex construction and design, and ultimately data dependency



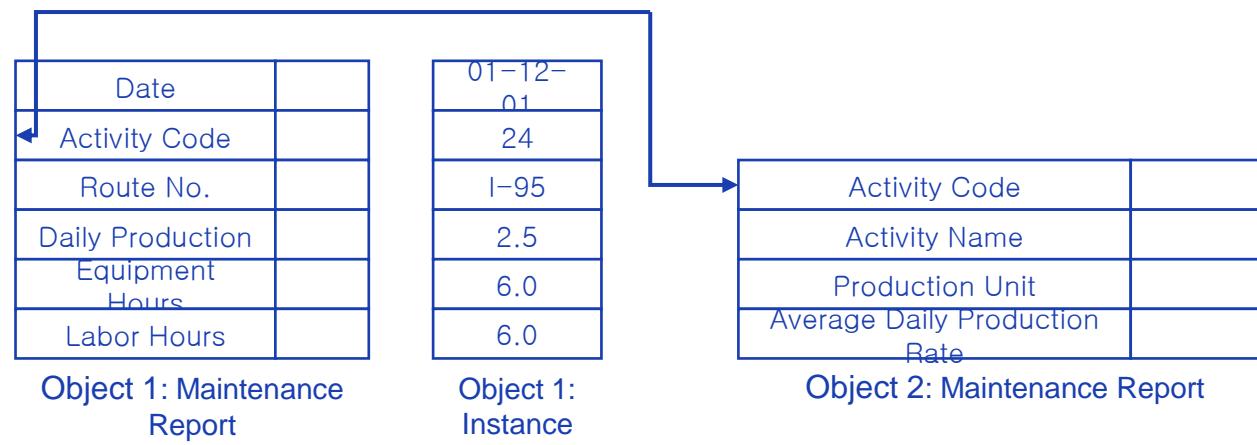
# Relational DB

- A structure in which mathematical and logical relationships are formed in the form of a table, and some of the columns in the table are duplicated with other tables to define the correlation between each table.
  - Advantage: High adaptability to changes in work, easy to use for changing work and convenient maintenance. Therefore, productivity is also improved.
  - Disadvantages: The system load is high because more resources are utilized than other DBMSs.



# Object oriented DB

- It is designed to overcome the basic limitations of processing multimedia data smoothly and only business type data types of RDBMS.
- Object databases are different from relational databases which are table-oriented. Object-relational databases are a hybrid of both approaches.
- UniSQL, Object Store

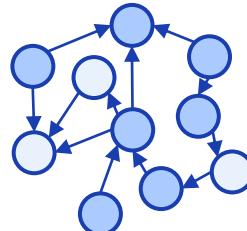


# NoSQL

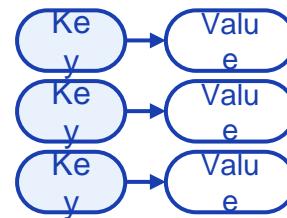
- A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.
- The data structures used by NoSQL databases (e.g. key–value pair, wide column, graph, or document) are different from those used by default in relational databases, making some operations faster in NoSQL.



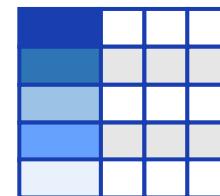
Document



Graph



Key-Value



Wide-column

# Advantages of RDBMS

- SQL (relational) databases have a mature data storage and management model.
- SQL database supports the notion of views that allow users to only see data that they are authorized to view.
- SQL databases support stored procedure SQL which allows database developers to implement a part of the business logic into the database.
- SQL databases have better security models compared to NoSQL databases.

# RDBMS Features (1/2)

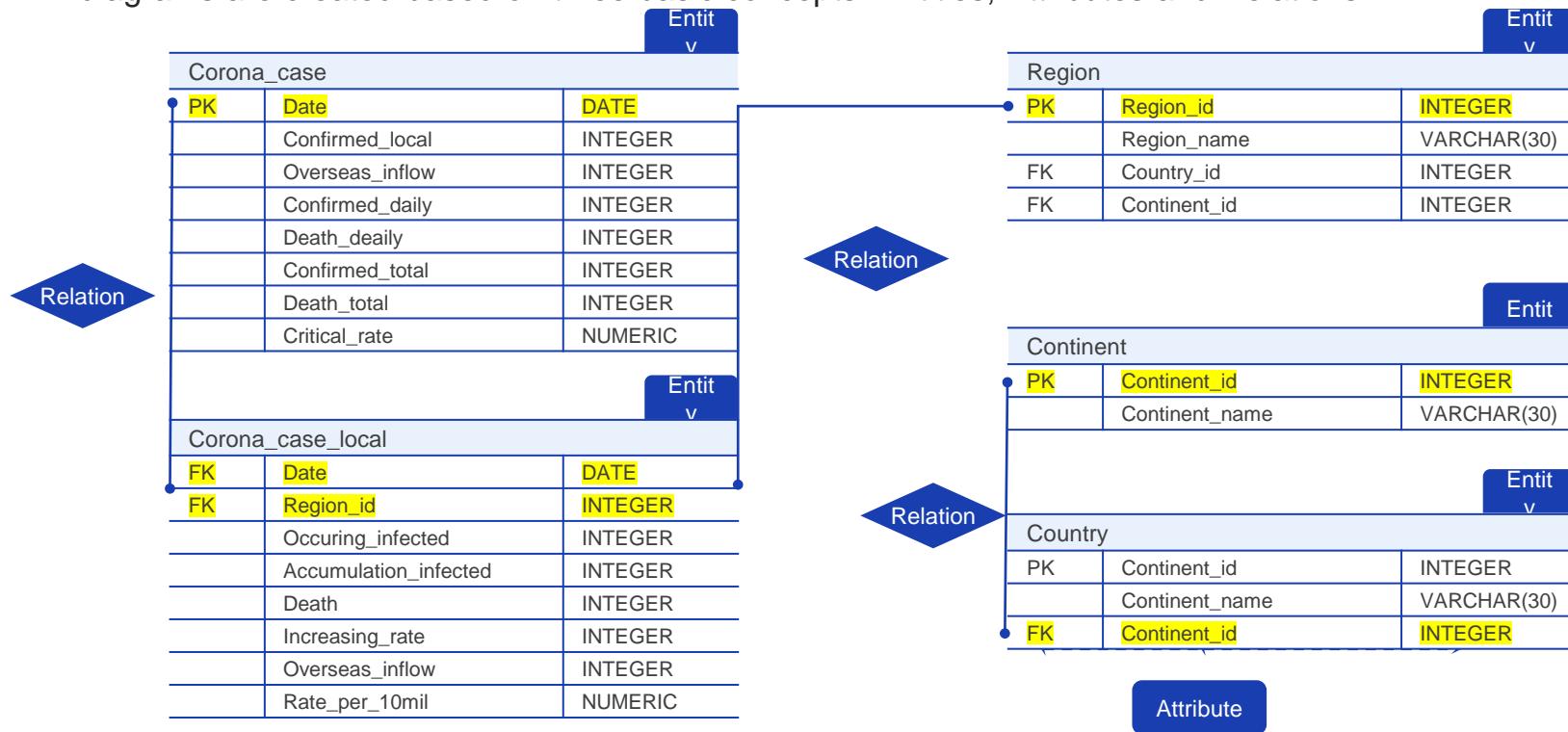
- Transaction, ACID (Atomicity, Consistency, Isolation, Durability) support
- Represent all data as a two-dimensional table
  - A table is a basic data storage unit consisting of row (record, tuple) and column (field, item).
  - set of tables that are related to each other
  - Entity Relationship (ER) model of the blueprint of the database
  - According to the ER model, a database is created, and the database consists of one or more tables.
- Relational
  - Search for desired data by combining multiple tables

## RDBMS Features (2/2)

- DataBase - a database is a kind of data storage.
  - A repository of information organized into simple, regular shapes
  - Like a table in Excel, it is composed of tables, and each table is composed of rows and columns.
  - Each row is called a record, and records are made up of several pieces of information, where the pieces are columns.
- Management System is software that allows you to insert (insert), search (select), modify (update), and delete (delete) records in the DB.
  - The ability to process data
  - Many DBMSs do this by supporting SQL (Structured Query Language).

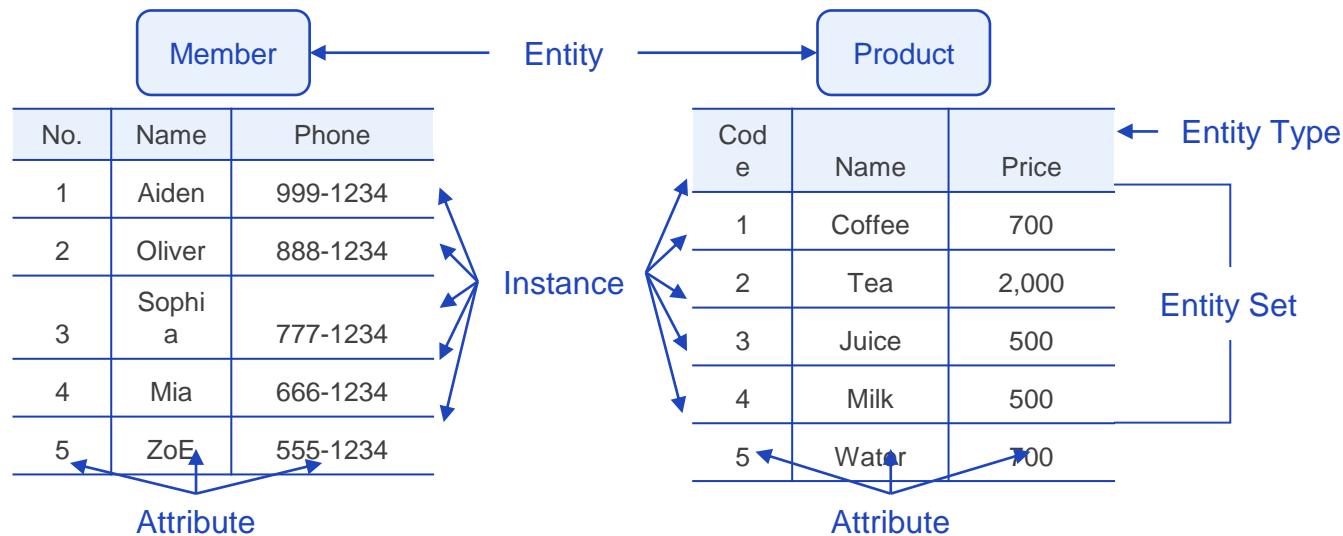
# Database Design (1/4)

- Entity-Relation Model
  - ER diagrams are created based on three basic concepts: Entities, Attributes and Relations



## Database Design (2/4)

- Entity means a set of data
  - It is data that must be stored and managed.
  - It refers to a concept, place, event, etc.
  - It refers to a tangible or intangible object.



# Database Design (3/4)

- Attribute
    - Defined as the smallest data unit that is no longer semantically separated to be managed as an instance
    - Attributes are no longer semantically decoupled but elements that describe entities and components of instances
- Ex) Student number, name, address



| Name    | Color | Year | Displacement | Model |
|---------|-------|------|--------------|-------|
| Porsche | red   | 2021 | 2981         | 911   |

| Power unit               |                            |
|--------------------------|----------------------------|
| Number of cylinders      | <b>6</b>                   |
| Bore                     | <b>91.0 mm</b>             |
| Stroke                   | <b>76.4 mm</b>             |
| Displacement             | <b>2,981 cc</b>            |
| Power (kW)               | <b>288 kW</b>              |
| Power (PS)               | <b>392 PS</b>              |
| RPM point maximum power  | <b>6,500 r/min</b>         |
| Maximum engine speed     | <b>7,500 r/min</b>         |
| Max. torque              | <b>45.9 kg·m</b>           |
| RPM range maximum torque | <b>1,950 - 5,000 r/min</b> |

# Database Design (4/4)

- Relation
  - Two-dimensional table, represented by rows and columns
  - A relation is used to store information about an entity represented in the database.
  - Each relation has a unique name.
  - In general, relations do not have a one-to-one correspondence with the file system.
- Row = record = tuple
  - A collection of values relating to a particular instance of the entity represented by the relation.
  - That is, a tuple represents an instance of the entity represented by the relation.
- Column = attribute
  - One column with name in relation
  - Degree: Number of columns/number of attributes in a relation
  - The minimum order of a valid relation must have 1 -> at least one attribute. The order does not change often.

## DBMS Key (1/4)

- Key
  - Uniquely identifying something
  - A property or set of properties used to identify a particular tuple.

| Region |    |              |
|--------|----|--------------|
| Key    | PK | Region_id    |
|        |    | VARCHAR(30)  |
|        | FK | Country_id   |
|        | FK | Continent_id |

\* PK – Primary key, FK – Foreign key

## DBMS Key (2/4) – Super & Candidate Key

- Superkey
  - A superkey is a single attribute or set of attributes that uniquely identifies a tuple.
  - That is, if a tuple can be identified, all are superkeys.
- A candidate key (or minimal superkey)
  - Is a superkey that can't be reduced to a simpler superkey by removing an attribute

| Customer | No. | Name   | Social Number  | Address         | Phone         |
|----------|-----|--------|----------------|-----------------|---------------|
|          | 1   | Aiden  | 810101-1111111 | Seoul, Korea    | 000-5000-0001 |
|          | 2   | Oliver | 900101-2222222 | London, England | 000-6000-0001 |
|          | 3   | Mia    | 830101-2333333 | Paris, France   | 000-7000-0001 |
|          | 4   | ZoE    | 820101-1444444 | LA, US          | 000-8000-0001 |

## DBMS Key (3/4) – Primary Key (PK)

- A primary key is a key that selects one of the candidate keys to represent it.
- Considerations when selecting a primary key
  - It must have a unique value that identifies the tuple in the relation.
  - NULL values are not allowed.
  - The key value should not change.
  - It should have as few attributes as possible.
  - There should be no problem in using the key in the future.

## DBMS Key (4/4) – Foreign Key (FK)

- A foreign key is a property that refers to the primary key of another relation.
  - A foreign key refers to the primary key of another relation to express the relationship between relations, which is a characteristic of the relational data model.
- Characteristics of foreign keys
  - Represents relationships between relations in a relational data model.
  - A property that refers to the primary key of another relation.
  - Both the referenced (foreign key) and the referenced (primary key) domain have the same domain
  - NULL values and duplicate values are allowed.
  - A foreign key that references its own primary key is also possible.
  - Foreign key can be part of primary key

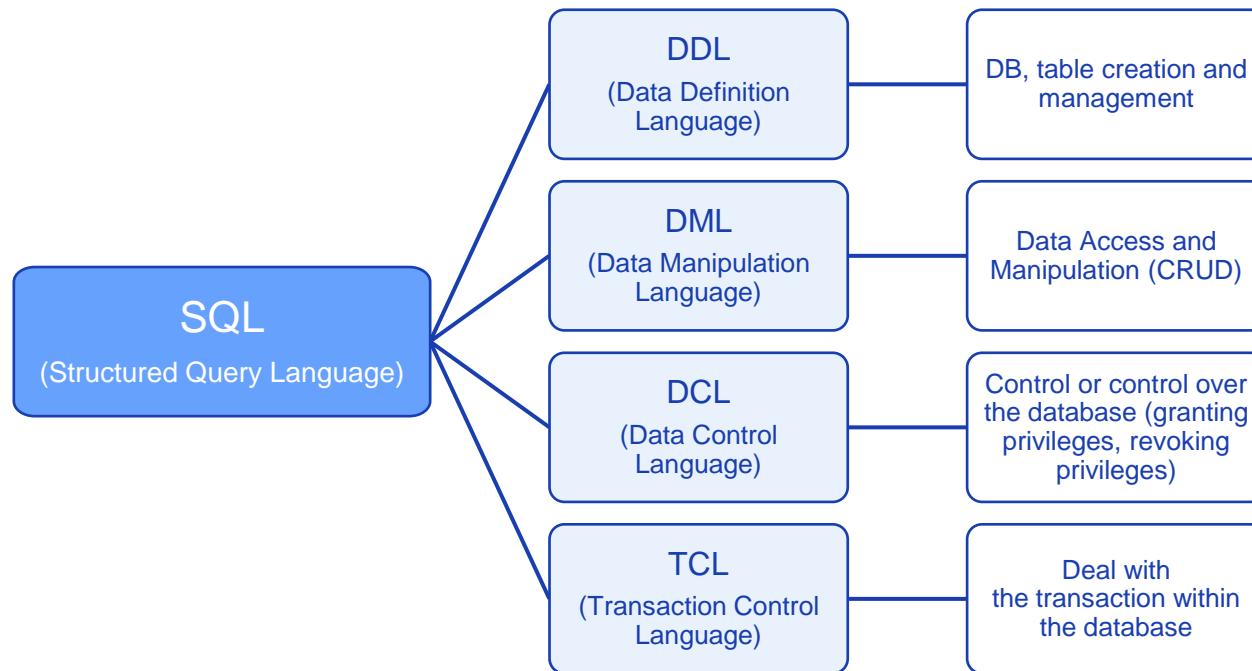
# Database Normalization

- Normalization is in order to reduce data redundancy and improve data integrity
- Database design techniques that reduce data redundancy and eliminate undesirable attributes such as insert, update, and delete anomalies
  - Split a large table into smaller tables and connect them using relationships
  - The purpose of normalization is to remove redundant (repeated) data and ensure that the data is stored logically.  
Ex) 1NF, 2NF, 3NF, BCNF
- Most database systems are normalized databases up to the third canonical form.
- A uniquely identified primary key is a record in a table and cannot be null.
- Foreign key helps to link tables and refer to primary key

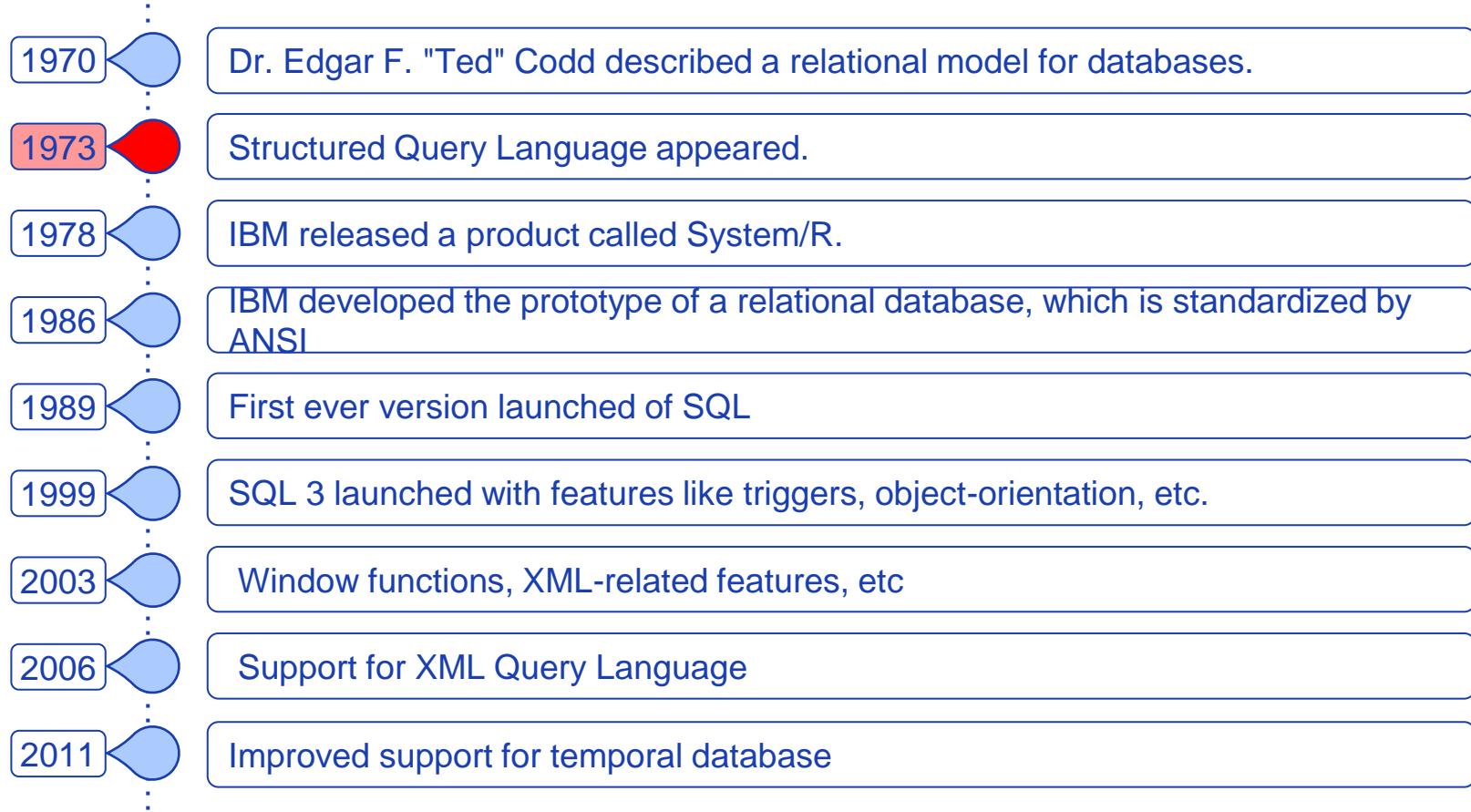
# What is SQL

- Structured Query Language
- A language for managing data in relational database systems.
- Insert, search, update and delete for database records
- High Productivity
- Portability
- Standard SQL – Commands written in standard SQL can be executed in the same way in other DBMSs without modifying the commands written once.

# Types of SQL commands



# History of SQL



# What is SQL used for?

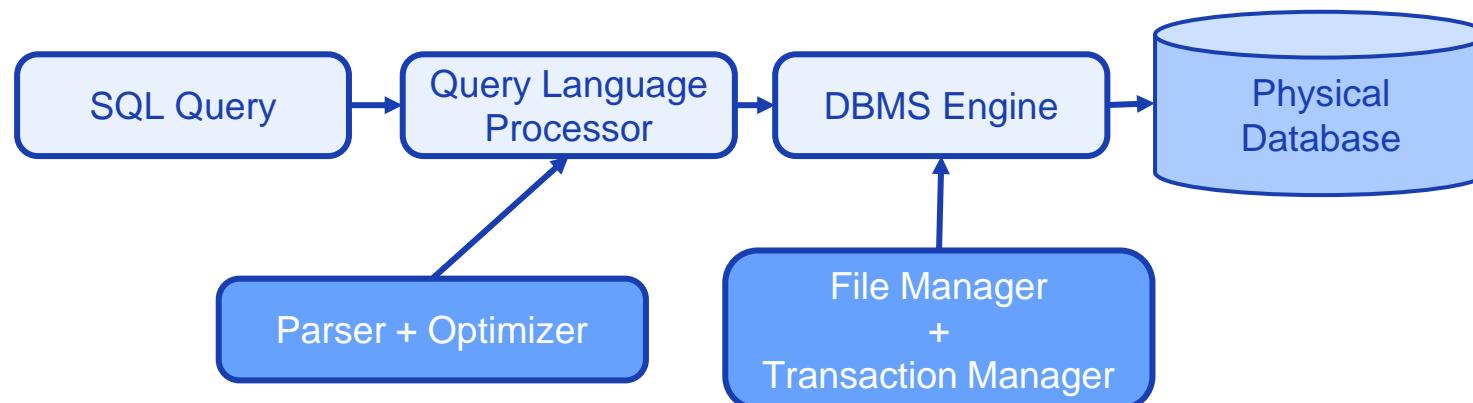
- Help users access data in RDBMS system
- Help to explain data
- It defines the data in the database and manipulates specific data.
- Create and drop databases and tables with the usage of SQL commands
- SQL uses functions in the database, creates views and provides stored procedures
- Set permissions on tables, procedures, and views

# SQL commands

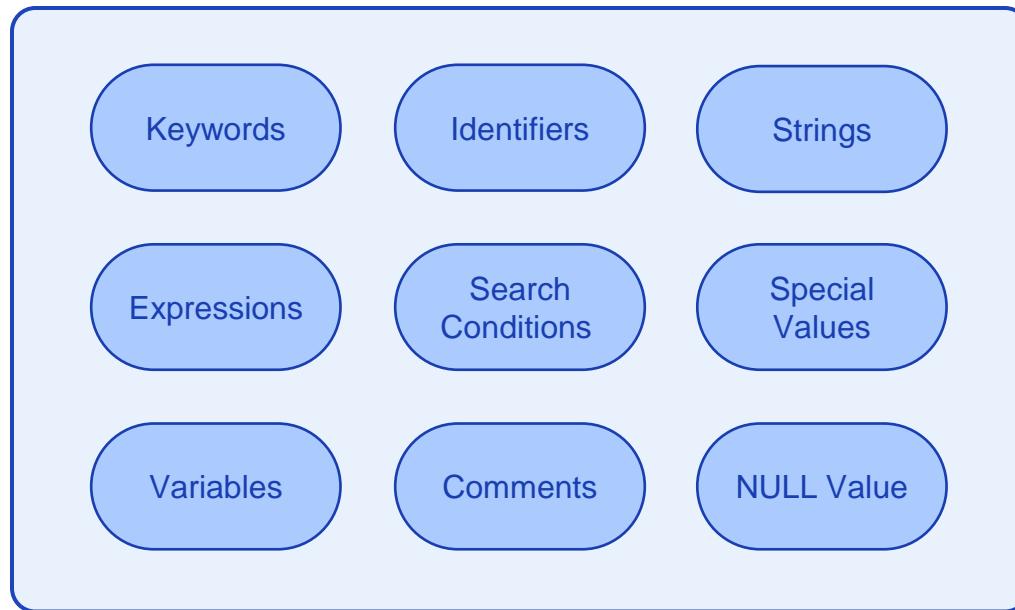
| Commands | Description                                                                |
|----------|----------------------------------------------------------------------------|
| CREATE   | Defines the database structure schema                                      |
| INSERT   | Inserts data into the row of a table                                       |
| UPDATE   | Updates data in a database                                                 |
| DELETE   | Removes one or more rows from a table                                      |
| SELECT   | Selects the attribute based on the condition described by the WHERE clause |
| DROP     | Removes tables and databases                                               |

# SQL Process

- Key components of the SQL process
  - SQL Query Engine
  - Optimization Engines
  - Query Dispatcher
  - Classic Query Engine



# SQL Language elements



# Data Definition Language

- In the context of SQL, data definition or data description language (DDL) is a syntax for creating and modifying database objects such as tables, indices, and users
- DDL statements are similar to a computer programming language for defining data structures, especially database schemas.
- Common examples of DDL statements include CREATE, ALTER, and DROP

# How to create / delete Database

- By executing a simple SQL query in PostgreSQL
  - Using pgAdmin4 application for commands
- CREATE DATABASE syntax

```
CREATE DATABASE IF NOT EXISTS db_name;
```

- Use the command CREATE SCHEMA instead of Database for schema
- DROP DATABASE syntax

```
DROP DATABASE db_name;
```

# How to create Table (1/2)

- CREATE TABLE syntax

```
CREATE TABLE IF NOT EXISTS accounts;
```

- Example

```
CREATE TABLE IF NOT EXISTS accounts (
 user_id serial PRIMARY KEY,
 username VARCHAR (30) UNIQUE NOT NULL,
 password VARCHAR (20) NOT NULL,
 email VARCHAR (50) UNIQUE NOT NULL,
 last_login timestamp
);
```

accounts



user\_id: int4  
username: varchar(30)  
password: varchar(20)  
email: varchar(255)  
last\_login: timestamp(6)

## How to create Table (2/2)

- PostgreSQL includes the following column constraints:
  - NOT NULL – Ensures that values in a column cannot be NULL
  - UNIQUE – Ensures the values in a column unique across the rows within the same table
  - PRIMARY KEY
    - A primary key column uniquely identify rows in a table
    - A table can have one and only one primary key
    - The primary key constraint allows you to define the primary key of a table
  - CHECK – A CHECK constraint ensures the data must satisfy a Boolean expression
  - FOREIGN KEY
    - Ensures values in a column or a group of columns from a table exists in a column or group of columns in another table
    - Unlike the primary key, a table can have many foreign keys

# Change table structure

- Change the schema(Field's attributes) of Table
  - Change for column

```
ALTER TABLE test ADD COLUMN abc BOOLEAN;
```

```
ALTER TABLE test DROP COLUMN abc;
```

```
ALTER TABLE test RENAME COLUMN abc TO new_abc;
```

```
ALTER TABLE test RENAME TO t_table;
```

# CREATE TABLE AS SELECT (CTAS)

- How to create a table through the result of a query using the Select statement
  - Create table new\_table as select statement
  - When copying data

```
CREATE TABLE table_name
AS SELECT * FROM table_name_to_copy;
```

```
CREATE TABLE table_name
AS SELECT * FROM table_name_to_copy
WHERE 1=2;
```

# Data types

- A database data type refers to the format of data storage that can hold a distinct type or range of values.
  - When computer programs store data in variables, each variable must be designated a distinct data type.
- Some common data types are as follows: integers, characters, strings, floating point numbers and arrays.
  - More specific data types are as follows: varchar (variable character) formats, Boolean values, dates and timestamps.
- Common Database Data Types
  - Integer
  - Character String
  - String
  - Floating Point Number
  - Array
  - Varchar
  - Boolean
  - Datetime

# Numeric data types

| Type        | Description                                                                                          |
|-------------|------------------------------------------------------------------------------------------------------|
| TINYINT()   | -128 to 127 normal.<br>0 to 255 UNSIGNED.                                                            |
| SMALLINT()  | -32768 to 32767 normal.<br>0 to 65535 UNSIGNED.                                                      |
| MEDIUMINT() | -8388608 to 8388607 normal.<br>0 to 16777215 UNSIGNED.                                               |
| INT()       | -2147483648 to 2147483647 normal.<br>0 to 4294967295 UNSIGNED.                                       |
| BIGINT()    | -9223372036854775808 to 92233720368547758077 normal.<br>0 to 18446744073709551615 UNSIGNED.          |
| FLOAT       | A small approximate number with a floating decimal point.                                            |
| DOUBLE(,)   | A large number with a floating decimal point.                                                        |
| DECIMAL(,)  | A DOUBLE stored as a string, allowing for a fixed decimal point. Choice for storing currency values. |

# String data types

| Type       | Description                                              |
|------------|----------------------------------------------------------|
| CHAR()     | A fixed section from 0 to 255 characters long.           |
| VARCHAR()  | A variable section from 0 to 255 characters long.        |
| TINYTEXT   | A string with a maximum length of 255 characters.        |
| TEXT       | A string with a maximum length of 65535 characters.      |
| BLOB       | A string with a maximum length of 65535 characters.      |
| MEDIUMTEXT | A string with a maximum length of 16777215 characters.   |
| MEDIUMBLOB | A string with a maximum length of 16777215 characters.   |
| LONGTEXT   | A string with a maximum length of 4294967295 characters. |

# Date & Time data types

| Type      | Description         |
|-----------|---------------------|
| DATE      | YYYY-MM-DD          |
| DATETIME  | YYYY-MM-DD HH:MM:SS |
| TIMESTAMP | YYYYMMDDHHMMSS      |
| TIME      | HH:MM:SS            |

# MariaDB main data types

| Type                 | List                                         |
|----------------------|----------------------------------------------|
| String Types         | varchar(n), char(n), text, string, binary(n) |
| Numeric Types        | tinyint, smallint, int, bigint               |
| Floating-Point Types | float, double, decimal(m,n)                  |
| Date/Time Types      | date, time, datetime, timestamp              |
| Boolean Types        | boolean                                      |

Unit 1

# Introduction to SQL

- | 1.1. Creating tables using DDL
- | 1.2. Manage table data with DML
- | 1.3. Query data with SQL

# Data Manipulation Language

- DML is Data Manipulation Language which is used to manipulate data itself
  - For example: insert, update, delete are instructions in SQL
- It is used to add, retrieve or update the data
- It adds or updates the row of the table. These rows are called as tuple
- BASIC command present in DML are UPDATE, INSERT, SELECT, DELETE, etc

# CRUD (Create, Read, Update, Delete)

- CRUD operations mean
  - C- Create means "Insert the data"
  - R- Read means "Select the data"
  - U- Update means "Update the data"
  - D- Delete means "Delete the data"
- UPSERT - conditionally update when trying to insert
  - If the key exists when inserting

```
INSERT INTO table_name(column_1) values(value_1) ON CONFLICT target action;
```

- `INSERT INTO TABLE_NAME(COLUMN_1) VALUES(VALUE_1) ON CONFLICT TARGET ACTION;`

# UPDATE

- The PostgreSQL UPDATE statement allows you to modify data in a table
- When referencing the contents of another table during UPDATE

- Example

```
UPDATE target_table a SET a.column_1 = expression FROM ref_table b
WHERE a.column_1 = b.column_1;
```

- Modifying existing data in a table

```
UPDATE table_name SET column_1 = value1, column_2 = value2 WHERE condition;
```

# DELETE

- The PostgreSQL DELETE statement allows you to delete one or more rows from a table.
  - When deleting specific data from a table
  - Delete data that meets the conditions

Example

```
DELETE FROM target_table a WHERE conditional expression;
```

# INSERT

- Insert - inserts data into a table
  - Enter only specific columns
  - Example

```
INSERT INTO table_name (column1, column2) values (value1, value2);
```

- When entering in the order of columns included in the table, you can omit the column

```
INSERT INTO table_name values (value1, value2, value3, ...);
```

# SELECT (1/2)

- SELECT is a SQL keyword that tells the database that you want to retrieve data(tuples)

- Example

```
SELECT columns_list FROM table_name;
```

- PostgreSQL evaluates the FROM clause before the SELECT clause in the SELECT statement.
- The SELECT statement has the following clauses:
  - Select distinct rows using DISTINCT operator.
  - Sort rows using ORDER BY clause.
  - Filter rows using WHERE clause.
  - Select a subset of rows from a table using LIMIT or FETCH clause.
  - Group rows into groups using GROUP BY clause.
  - Filter groups using HAVING clause.

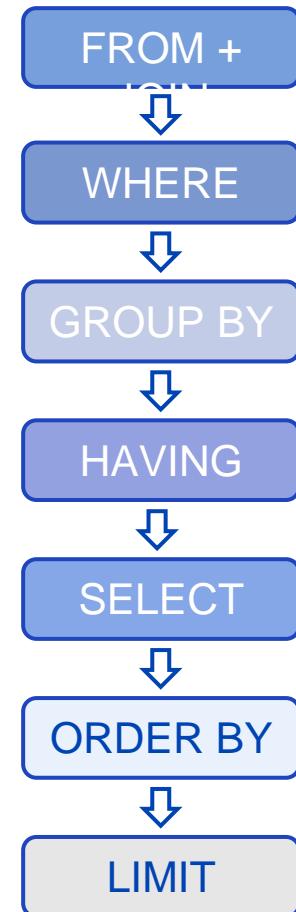
## SELECT (2/2)

- Syntax

```
SELECT [DISTINCT|ALL] { * | fieldExpression [AS newName]}
FROM tableName [alias]
[WHERE condition]
[GROUP BY fieldName(s)]
[HAVING condition]
ORDER BY fieldName(s)
```

# Order of execution of SELECT

- 1. FROM and JOIN - table
- 2. WHERE - condition
- 3. GROUP BY – column or expression (where clause extracted data grouping)
- 4. HAVING – group condition
- 5. SELECT - column name
- 6. DISTINCT
- 7. ORDER By – sort
- 8. LIMIT / OFFSET



# Where clause

- Examples

```
SELECT * FROM `movies` WHERE `category_id` = 2 AND `year_released` = 2008;
```

```
SELECT * FROM `movies` WHERE `category_id` = 1 OR `category_id` = 2;
```

```
SELECT * FROM `movies` WHERE `membership_number` IN (1,2,3);
```

```
SELECT * FROM `movies` WHERE `membership_number` NOT IN (1,2,3);
```

```
SELECT * FROM `movies` WHERE `category_id` <> 1;
```

Unit 1

# Introduction to SQL

- | 1.1. Creating tables using DDL
- | 1.2. Manage table data with DML
- | 1.3. Query data with SQL

# JOIN

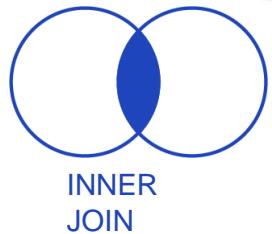
- Join is used to combine columns from one (self-join) or more tables based on the values of the common columns between related tables.
  - The common columns are typically the primary key columns of the first table and foreign key columns of the second table.
- It refers to outputting data by linking or combining two or more tables.
- PostgreSQL supports inner join, left join, right join, full outer join, cross join, natural join, and a special kind of join called self-join.

| a_id<br>[PK] integer | animal<br>character varying (50) |
|----------------------|----------------------------------|
| 10                   | Lion                             |
| 20                   | Elephant                         |
| 30                   | Tiger                            |
| 40                   | Horse                            |

| b_id<br>[PK] integer | animal<br>character varying (50) |
|----------------------|----------------------------------|
| 10                   | Lion                             |
| 20                   | Monkey                           |
| 30                   | Horse                            |
| 50                   | Elephant                         |

# Inner JOIN

- Inner JOIN
  - This method is used when column values between two tables exactly match each other.
  - The condition of JOIN is described in the WHERE clause, and the = operator is used.



```
SELECT a_id, a_animal, b_id, b_animal
FROM set_a
INNER JOIN set_b
ON a_animal = b_animal;
```

| a_id<br>integer | a_animal<br>character varying (50) | b_id<br>integer | b_animal<br>character varying (50) |
|-----------------|------------------------------------|-----------------|------------------------------------|
| 1               | Lion                               | 10              | Lion                               |
| 2               | Elephant                           | 50              | Elephant                           |
| 7               | Horse                              | 30              | Horse                              |

# Outer JOIN – LEFT Outer Join

- An outer join is used to return results by combining rows from two or more tables
- Return every row from one specified table, even if the join condition fails
- LEFT Outer Join
  - Data corresponding to the marked left table is first read, and then the JOIN target data is read from the right table
  - NULL if there is no satisfactory data in the right table

- Example

```
SELECT a_id, a_animal, b_id, b_animal
FROM set_a
LEFT JOIN set_b
ON a_animal = b_animal;
```



| a_id<br>integer | a_animal<br>character varying (50) | b_id<br>integer | b_animal<br>character varying (50) |
|-----------------|------------------------------------|-----------------|------------------------------------|
| 1               | Lion                               | 10              | Lion                               |
| 2               | Elephant                           | 50              | Elephant                           |
| 3               | Tiger                              | [null]          | [null]                             |
| 7               | Horse                              | 30              | Horse                              |

# Outer JOIN – RIGHT Outer Join



- RIGHT Outer Join

- When performing a join, the corresponding data is first read from the right table marked first, and then the JOIN target data is read from the left table
- NULL if there is no satisfactory data in the left table

```
SELECT a_id, a_animal, b_id, b_animal
FROM set_a
RIGHT JOIN set_b
ON a_animal = b_animal;
```

| a_id<br>integer | a_animal<br>character varying (50) | b_id<br>integer | b_animal<br>character varying (50) |
|-----------------|------------------------------------|-----------------|------------------------------------|
| 1               | Lion                               | 10              | Lion                               |
| [null]          | [null]                             | 20              | Monkey                             |
| 7               | Horse                              | 30              | Horse                              |
| 2               | Elephant                           | 50              | Elephant                           |



## Outer JOIN – FULL Outer Join

- FULL Outer Join
  - When performing a join, all data in the left and right tables are read and the result is generated by joining
  - Example

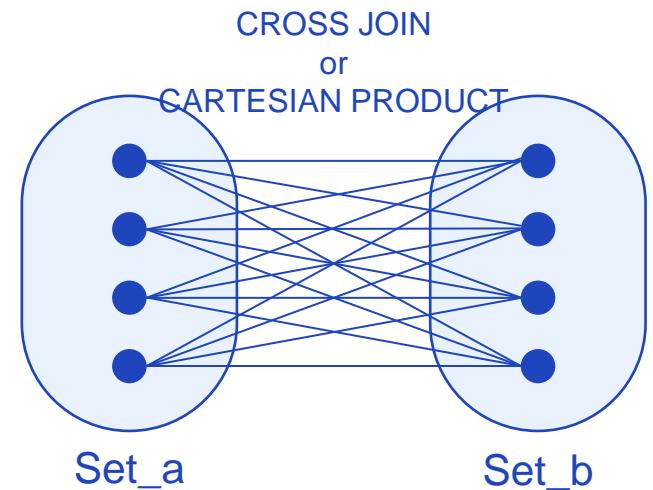
```
SELECT a_id, a_animal, b_id, b_animal
FROM set_a
FULL JOIN set_b
ON a_animal = b_animal;
```

| a_id<br>integer | a_animal<br>character varying (50) | b_id<br>integer | b_animal<br>character varying (50) |
|-----------------|------------------------------------|-----------------|------------------------------------|
| 1               | Lion                               | 10              | Lion                               |
| 2               | Elephant                           | 50              | Elephant                           |
| 3               | Tiger                              | [null]          | [null]                             |
| 7               | Horse                              | 30              | Horse                              |
| [null]          | [null]                             | 20              | Monkey                             |

# Cross JOIN

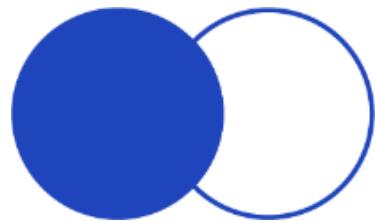
- A cross join is a type of join that returns the Cartesian product of rows from the tables in the join.
- In other words, it combines each row from the first table with each row from the second table.
- For CROSS JOIN results for two tables,  $M * N$  data combinations of both sets.
- Massive load (Caution)

```
SELECT a_id, a_animal, b_id, b_animal
FROM set_a
CROSS JOIN set_b
```

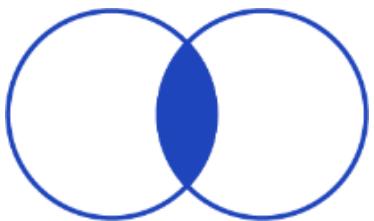


# JOIN Operations

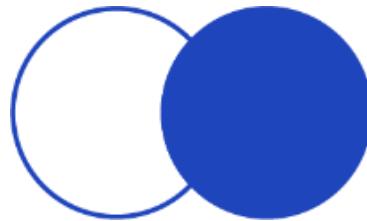
LEFT JOIN b ON a.key = b.key



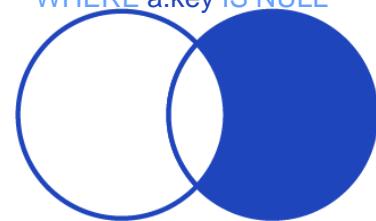
INNER JOIN b ON a.key = b.key



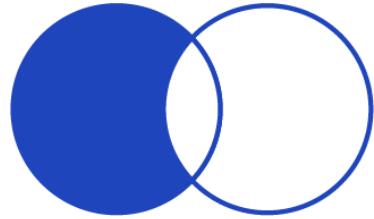
RIGHT JOIN b ON a.key = b.key



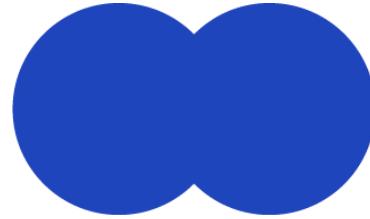
RIGHT JOIN b ON a.key = b.key  
WHERE a.key IS NULL



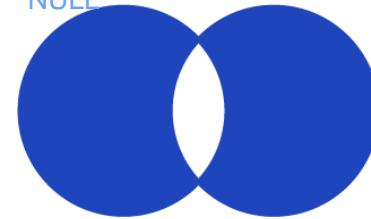
LEFT JOIN b ON a.key = b.key  
WHERE b.key IS NULL



FULL JOIN b ON a.key = b.key



FULL JOIN b ON a.key = b.key  
WHERE a.key IS NULL OR b.key IS NULL



# Comparison and Logical Operators

- To form the condition in the WHERE clause, you use comparison and logical operators

| Operator | Description           |
|----------|-----------------------|
| =        | Equal                 |
| >        | Greater than          |
| <        | Less than             |
| >=       | Greater than or equal |
| <=       | Less than or equal    |
| <>or!=   | Not equal             |
| AND      | Logical operator AND  |
| OR       | Logical operator OR   |

- Priority
  - NOT > AND > OR

# LIKE & Wildcards

- Like & Wildcards powerful tools that help search data matching complex patterns.
- There are a number of wildcards that include the percentage, underscore and charlist (not supported by MySQL ) among others
- The percentage wildcard is used to match any number of characters starting from zero (0) and more.
- The underscore wildcard is used to match exactly one character.
- Escape Keyword
  - The ESCAPE keyword is used to exclude pattern-matching characters when characters such as percentages (%) and underscores (\_) form part of the data

```
SELECT * FROM movies WHERE title LIKE '67#%' ESCAPE '#';
```

```
SELECT * FROM movies WHERE title LIKE '67=%' ESCAPE '=';
```

# Regular Expressions (REGEXP)

- Regular expressions search for data matching complex criteria
- Compared to wildcards, regular expressions retrieve data that matches much more complex criteria
  - Syntax

```
SELECT statement ... WHERE fieldname REGEXP 'pattern';
```

```
SELECT * FROM `movies` WHERE `title` REGEXP '^ [abcd]';
```

# BETWEEN & IN

- BETWEEN - operator that outputs a set that falls within a specific range

- expression BFTWFFN value 1 AND value 2

```
SELECT * FROM price_table WHERE price BETWEEN 200 AND 280;
```

```
SELECT * FROM price_table WHERE price >= 200 AND price <= 280;
```

- IN - an operator that determines whether a specific set or list exists in a specific set (column or list).
  - Use IN operator in the WHERE clause to check if a value matches any value in a list of values.
- value IN (value1, value2, ...)
  - Operator to check if value1 and value2 values exist in the set of column

# GROUP BY

- The PostgreSQL GROUP BY clause is used to divide rows returned by SELECT statement into different groups.
- Perform an aggregation function for each group
- If you create a group with multiple columns, list them with “,”
- **SELECT statements... GROUP BY column\_name1[,column\_name2,...] [HAVING condition];**

```
SELECT col_1, sum(col_2) FROM table GROUP BY col_1
```

- Example

- The SELECT statement used in the GROUP BY clause can contain only column names, aggregate functions, constants, and expressions.

# Aggregate functions

- Aggregate functions perform a calculation on a set of rows and return a single row.
  - AVG() – return the average value.
  - COUNT() – return the number of values.
  - MAX() – return the maximum value.
  - MIN() – return the minimum value.
  - SUM() – return the sum of all or distinct values.

# Single-row Function

- It can be used in SELECT, WHERE, ORDER BY clauses.
- It operates on each row individually to manipulate data values, and returns an operation result for each row.
- Even if multiple arguments are input, only one result is returned.
- Single row functions can be character functions, numeric functions, date functions, and conversion functions.
- Note that these functions are used to manipulate data items.

# Single-row Function

- Case Conversion functions - Accepts character input and returns a character value
  - Functions under the category are UPPER, LOWER and INITCAP
    - UPPER function converts a string to upper case
    - LOWER function converts a string to lower case
    - INITCAP function converts only the initial alphabets of a string to upper case
- Character functions - Accepts character input and returns number or character value.
  - Functions under the category are CONCAT, LENGTH, SUBSTR, INSTR, LPAD, RPAD, TRIM and REPLACE
    - CONCAT function concatenates two string values
    - LENGTH function returns the length of the input string
    - SUBSTR function returns a portion of a string from a given start point to an end point
    - INSTR function returns numeric position of a character or a string in a given string.
    - LPAD and RPAD functions pad the given string upto a specific length with a given character.
    - TRIM function trims the string input from the start or end.

# SUBSTRING( ) function

- The PostgreSQL substring function is used to extract a string containing a specific number of characters from a particular position of a given string.
  - Extracts specific number of characters form specific position

```
SUBSTRING(string [from starting_position] [for length])
```

```
SELECT substring('jsjeong' for 2);
```

- ```
SELECT substring('jsjeong' for 2);
```

```
SELECT substring('jsjeong' from 3 for 7);
```

▷ The result is "jeong"

SUBQUERY

- SELECT statement nested in one SQL statement
- A query in which another query is nested within one query
- JOIN vs. SUBQUERY
 - JOIN : Gathers data from multiple tables.
 - SUBQUERY: Process multiple sql statements into one (all DML possible)

```
SELECT select_list
FROM table condition or View
WHERE condition operator (SELECT select_list
FROM table
WHERE condition);
```

[Lab1]

Start with MariaDB



[Lab2] Working with SQL Tables



[Lab3]

Working with Tables



[Lab4]

Working with Queries



Unit 2.

Basic Analytics

Big Data Analytics

Unit 2

Basic Analytics

- | 2.1. Data preprocessing and Basic data analysis with Apache Pig
- | 2.2. Basic Querying with Apache Hive and Impala

Data Analysis and Processing



Hadoop MapReduce and
Spark

- ✓ Data processing and Analysis with Hadoop
- ✓ Nice data processing engines but it's too difficult for normal user to user



Pig and
Hive

- ✓ Higher-level abstractions for general data processing



Impala

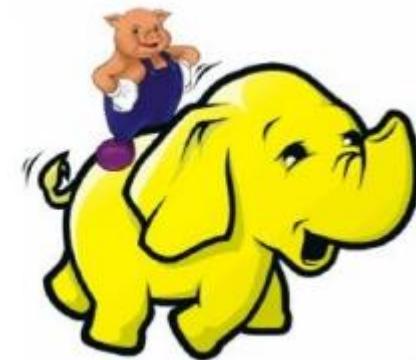
- ✓ Interactive analysis with execution engine

Apache Pig / Hive / Impala

- Pig
 - Alternative to writing java code for big data processing.
 - Offer high-level data processing
 - Especially good at preprocessing data for ETL with unstructured data.
- Apache Hive
 - Another abstraction on top of Hadoop with SQL
 - Hive use a SQL-like language, HiveQL
- Apache Impala
 - A massively parallel SQL engine on Hadoop cluster
 - Uses impala SQL such as HiveQL
 - High-level query language

What is Pig?

- It is an engine for executing programs on top of Hadoop
- Pig is an alternative to writing low-level MapReduce code.
- It provides a language, Pig Latin, to specify these programs
- Use Case
 - Data sampling
 - Used for extract, transform, and load (ETL) processing
 - Extract valuable data from log



Why Pig?

- Pigs eat anything
 - Pig can process any data, structured or unstructured
- Pigs live anywhere
 - Pig can run on any parallel data processing framework
- A data flow language as high level language
- A procedural language and it fits in pipeline paradigm
- Handle structured, unstructured, and semi-structured data
- No need for compilation, on execution pig operator is converted internally into a MapReduce Job

Pig VS MapReduce

- Major differences between Apache Pig and MapReduce

Apache Pig	MapReduce
Apache Pig is a data flow language.	MapReduce is a data processing paradigm.
It is a high level language.	MapReduce is low level and rigid.
Performing a Join operation in Apache Pig is pretty simple.	It is quite difficult in MapReduce to perform a Join operation between datasets.
Any novice programmer with a basic Knowledge of SQL can work conveniently with Apache Pig.	Exposure to Java is must to work With MapReduce.
Apache Pig uses multi-query approach, Thereby reducing the length of the codes to a great extent.	MapReduce will require almost 20 Times more the number of lines To perform the same task.
There is no need for compilation. On Execution, every Apache Pig operator is converted internally into a MapReduce job.	MapReduce jobs have a long compilation process.

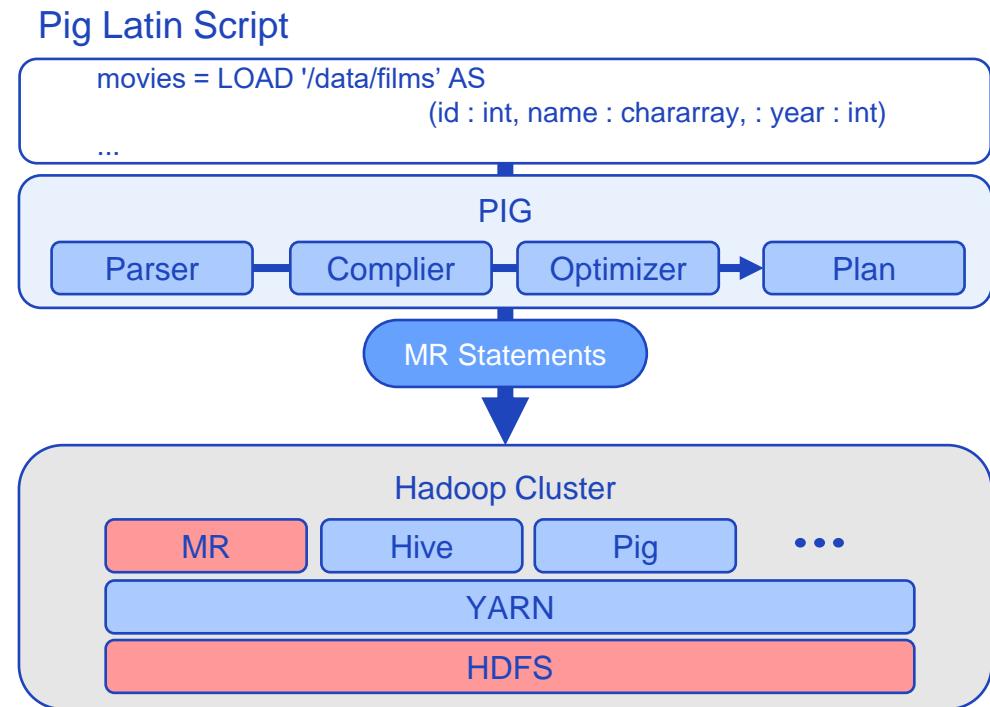
Pig VS SQL

- Major differences between Apache Pig and SQL.

Pig	SQL
Pig Latin is a procedural language.	SQL is declarative language.
In Apache Pig, schema is optional. We can store data with designing a schema (values are stored as \$01, \$02 etc.)	Schema is mandatory in SQL.
The data model in Apache Pig is nested relational.	The data model used in SQL is flat relational.
It provides limited opportunity for Query optimization.	There is more opportunity for query optimization in SQL.

Pig Components and Architecture

- Pig Latin – data flow language
- Grunt – interactive shell where you can type pig Latin statements
- Pig interpreter and execution Engine



Grunt Shell

- Grunt shell is Pig's interactive shell.
- The grunts shell of Apache pig is mainly used to write pig Latin scripts.
- Pig script can be executed with grunt shell which is native shell provided by Apache pig to execute pig queries.
- Useful for ad-hoc data inspection
- Execution is delayed until output is required
- ```
$ pig
grunt> sh ls
grunt> fs -ls
grunt> quit;
```

# Pig Scripts

- A Pig script is simply Pig Latin code stored in a text file (.pig extension)
  - Write all the required Pig Latin statements in a single file.
  - Write all the Pig Latin statements and commands in a single file and save it as . pig file.

~~- Execute the Apache Pig script (sales\_report.pig)~~

```
$ pig
grunt> run sales_report.pig;
grunt> quit;

$ pig sales_report.pig
```

## Pig Latin (1/5)

- Pig Latin is a data flow language
  - the flow of data is expressed as a sequence of statements
- Comments
  - Single line: –
  - Multi-line: /\* \*/
- Identifiers
  - Identifiers are the names assigned to fields and other data structures
  - An identifier must always begin with a letter
  - Keywords (in blue text) are not case-sensitive

```
allsales = LOAD 'sales' AS (name, price) ;
bigsales = FILTER allsales BY price > 999 ;
STORE bigsales INTO 'myreport' ;
```

## Pig Latin (2/5)

- Operators

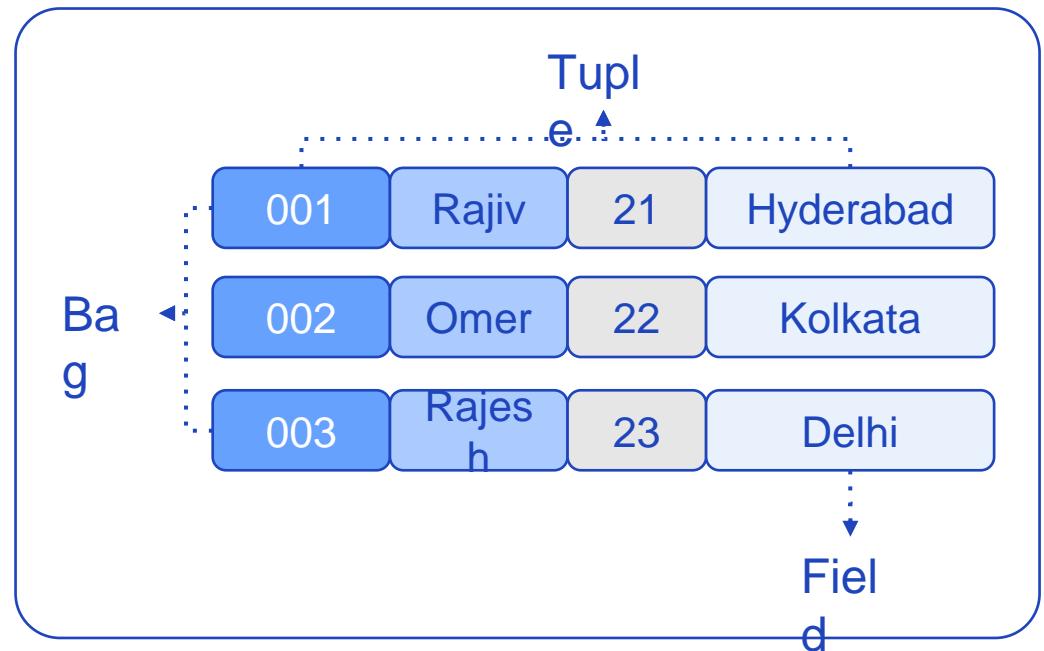
| Arithmetic | Comparison | Null        | Boolean |
|------------|------------|-------------|---------|
| +          | ==         | IS NULL     | AND     |
| -          | !=         | IS NOT NULL | OR      |
| *          | <          |             | NOT     |
| /          | >          |             |         |
| %          | <=         |             |         |
|            | >=         |             |         |

## Pig Latin (3/5)

| Data Type | Description                                      | Example                                                                 |
|-----------|--------------------------------------------------|-------------------------------------------------------------------------|
| int       | Signed 32-bit integer                            | 10                                                                      |
| long      | Signed 64-bit integer                            | Data: 10L or 10l<br>Display: 10L                                        |
| float     | 32-bit floating point                            | Data: 10.5F or 10.5f or 10.5e2f or 10.5E2F<br>Display: 10.5F or 1050.0F |
| double    | 64-bit floating point                            | Data: 10.5 or 10.5e2 or 10.5E2<br>Display: 10.5 or 1050.0               |
| chararray | Character array (string) in Unicode UTF-8 format | hello world                                                             |
| bytearray | Byte array (blob)                                |                                                                         |
| boolean   | boolean                                          | true/false (case insensitive)                                           |
| datetime  | Datetime                                         | 1970-01-01 T00:00:00.000+00:00                                          |

## Pig Latin (4/5)

- Tuples
  - A collection of values is called a tuple
- Bags
  - A collection of tuples is called a bag
  - Tuples within a bag are unordered by default
- Fields
  - A single element of data is called a field



## Pig Latin (5/5)

- A relation is the result of a processing step, simply a bag with an assigned name (alias)
- The name given to a relation is called an alias
- For example, result\_2cols is an alias:

```
grunt> allsales = LOAD 'sales.csv' USING PigStorage(',') AS (name, price);
grunt> describe all_sales;
grunt> dump all_sales;
```

# LOAD

- Data loading
- Pig's default loading function is called PigStorage
  - The name of the function is implicit when calling LOAD
  - PigStorage assumes text format with tab-separated columns
- The path can also refer to a directory (recursively load all files in that directory)

```
grunt> allsales = LOAD 'sales' AS (name, price);
grunt> allsales = LOAD 'sales';
grunt> allsales = LOAD 'sales_200[5-9]' AS (name, price);
grunt> allsales = LOAD 'sales.csv' USING PigStorage(',') AS (name, price);
```

# PigStorage

- A load function that parses a line of input into fields using a character delimiter
  - The default delimiter is a tab
  - Uses a delimited text file format
  - The default delimiter(Tab) can be changed to (,)

```
grunt> result_2cols = load 'mydata.log' AS (name, grade)
grunt> Describe result_2cols;
Result_2cols: {name: bytearray, grade: bytearray}
grunt> Dump
```

- Pigstorage can also be applied when using store() to save the result

## DUMP & STORE

- Command that executes the current code
- **DUMP** : sends output to the screen
- **STORE** : sends output to disk (HDFS)
  - The field delimiter also has a default value(tab)

```
STORE relation_name INTO ' required_directory_path ' [USING function];
```

```
STORE bigsales INTO 'myreport';
```

```
STORE bigsales INTO 'myreport' USING PigStorage(',');
```

# DESCRIBE

- Shows the structure of the data, including names and types

```
grunt> result_2cols = load 'mydata.log' as (name, grade)
grunt> Describe result_2cols;
Result_2cols: {name: bytearray, grade: bytearray}
```

- Defining a Schema

- Define the data type for the relation to be created.

```
grunt> result_2cols = load 'mydata.log' as (name: chararray, grade: int)
grunt> Describe result_2cols;
Result_2cols: {name: chararray, grade: int}
```

# FILTER

- Extracts tuples matching the specified criteria
- Example

```
A = FILTER salaries BY (age > 50 AND salary < 60000) OR zip == '95103'
```

| salarie |     | A        |       |
|---------|-----|----------|-------|
| s       |     | ?        |       |
| Gender  | Age | Salary   | Zip   |
| M       | 66  | 41000.00 | 95103 |
| M       | 58  | 76000.00 | 57701 |
| F       | 40  | 95000.00 | 95102 |
| M       | 45  | 60000.00 | 95105 |
| F       | 28  | 55000.00 | 95103 |

# FOREACH...GENERATE

- The operator works on each record in the data set (as in, “for each record”).
- The result of a FOREACH is a new tuple, typically with a different schema.

A = FOREACH salaries GENERATE age, salary;

| salarie |     |          |       |
|---------|-----|----------|-------|
| Gender  | Age | Salary   | Zip   |
| M       | 66  | 41000.00 | 95103 |
| M       | 58  | 76000.00 | 57701 |
| F       | 40  | 95000.00 | 95102 |
| M       | 45  | 60000.00 | 95105 |
| F       | 28  | 55000.00 | 95103 |



| A   |          |
|-----|----------|
| age | salary   |
| 66  | 41000.00 |
| 58  | 76000.00 |
| 40  | 95000.00 |
| 45  | 60000.00 |
| 28  | 55000.00 |

# FOREACH...GENERATE

- To create fields
  - Create a new field based on price

```
T = FOREACH salaries GENERATE salary * 0.07;
```

```
T = FOREACH salaries GENERATE salary * 0.07 as tax;
```

```
T = FOREACH salaries GENERATE salary * 0.07 as tax: float;
```

# DISTINCT / LIMIT

- Eliminates duplicate records in a bag

```
grunt> result_2cols = load 'mydata.log' as (name: chararray, grade: int)
grunt> unique_2cols = DISTINCT result_2cols;
```

- LIMIT

- Reduce the number of output records

```
grunt> Result = LIMIT Relation_name required number of tuples;
```

```
grunt> unique_2cols = DISTINCT result_2cols;
grunt> limit10_2cols = LIMIT unique_2cols 10;
```

# ORDER BY

- Used to display the contents of a relation in a sorted order based on one or more fields
- Record ordering is random unless specified with ORDER BY
- Given below is the example of the ORDER BY operator (TOP 5)

Sort ascending by grade (ASC: ascending order, DESC: descending order)

```
grunt> result_2cols = load 'mydata.log' as (name: chararray, grade: int)
grunt> sorted_2cols = ORDER result_2cols BY grade DESC;
grunt> top_five = LIMIT sorted_2cols 5;
```

# GROUP

- Grouping data within one or more relations based on a specific key
  - Grouping is possible even for multiple keys
  - Example

```
grunt> result_2cols = load 'mydata.log' as (name: chararray, grade: int)
grunt> group_2cols_by_name = GROUP result_2cols BY name;
```

- GROUP All
  - You can group a relation by all the columns

```
grunt> result_2cols = load 'mydata.log' as (name: chararray, grade: int)
grunt> group_all = GROUP result_2cols All;
```

# UNION

- Used to merge two relations with the same structure
  - Syntax

```
grunt> Relation_name3 = UNION Relation_name1, Relation_name2;
```

```
grunt> data_june = load 'mydata_jun.log' as (name: chararray, grade: int)
grunt> data_july = load 'mydata_jul.log' as (name: chararray, grade: int)
grunt> data_union = UNION data_june, data_july;
```

# Built-in Functions

- Math functions – ABS(amount), SUM(amount), AVG(amount), MIN(amount), MAX(amount)
- String functions – CONCAT(a, b), REPLACE(a, ‘-’, ‘/’)
- Date and time functions

| Function Description               | Example Invocation    | Input | Output               |
|------------------------------------|-----------------------|-------|----------------------|
| Convert to uppercase               | UPPER(country)        | UK    | UK                   |
| Remove leading/trailing spaces     | TRIM(name)            | Bob   | Bob                  |
| Return a random number             | RANDOM()              |       | 0.4816132<br>6652569 |
| Round to closest whole number      | ROUND(price)          | 37.19 | 37                   |
| Return chars between two positions | SUBSTRING(name, 0, 2) | Alice | Al                   |

# Review Questions

[Quiz]

Quiz 1

- What are some of the main components of Pig?

Quiz 2

- What is the difference between MapReduce mode and Local mode?

Quiz 3

- If you don't specify the type of data being loaded, what will Pig use as a default type?

[Lab5]

Using Pig for ETL Processing (Pre-processing)



Unit 2

# Basic Analytics

- | 2.1. Data preprocessing and Basic data analysis with Apache Pig
- | 2.2. Basic Querying with Apache Hive and Impala

# What is HIVE?

- A system for querying and managing structured data built on top of Hadoop
  - Uses Map-Reduce for execution
  - HDFS for storage – but any system that implements Hadoop FS API
- Key Building Principles:
  - Structured data with rich data types (structs, lists and maps)
  - Directly query data from different formats (text/binary) and file formats (Flat/Sequence)
  - SQL as a familiar programming tool and for standard analytics
  - Allow embedded scripts for extensibility and for non standard applications
  - Rich MetaData to allow data discovery and for optimization

# What does Hive provide?

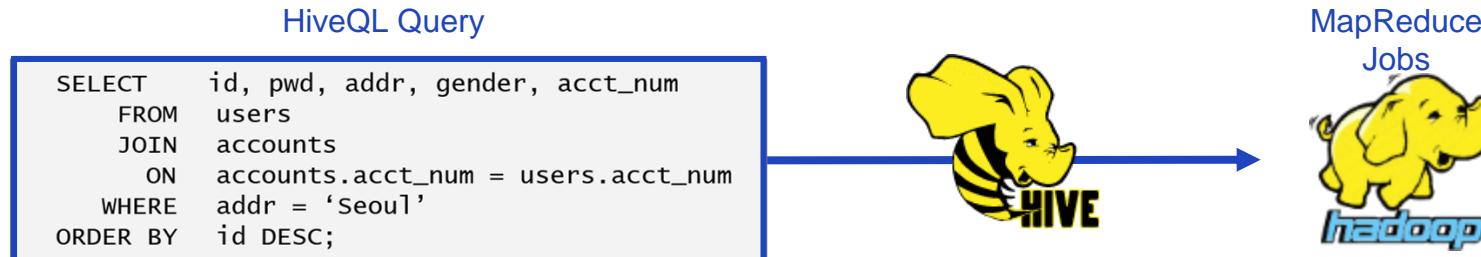
- Hive allows user to query data using HiveQL, a language very similar to standard SQL
- Hive turns HiveQL queries into standard MapReduce
  - Automatically runs the job and displays result
- Hive is NOT a RDBMS
  - Results of queries are converted into MapReduce jobs and can take quite a while to finish
  - You can not modify the data on a row level

# Why Hive?

- MapReduce is very difficult to program.
- Facebook analysts are SQL experts
- Developed to leverage Hadoop's massive data processing capabilities and existing SQL expert experience in-house.

# Apache Hive Characteristics

- A high-level tools that uses an SQL-like syntax to query HDFS
- When processing Hadoop, it is changed to actual Map-Reduce and executed.
- Using Map-Reduce and SPARK engine
- Data processing / analytics Service developed by Facebook
  - Used to access HDFS data using SQL-like syntax
  - Data Warehouse on Hadoop Platform
  - Batch Data preparation / ETL



# Version-specific characteristics

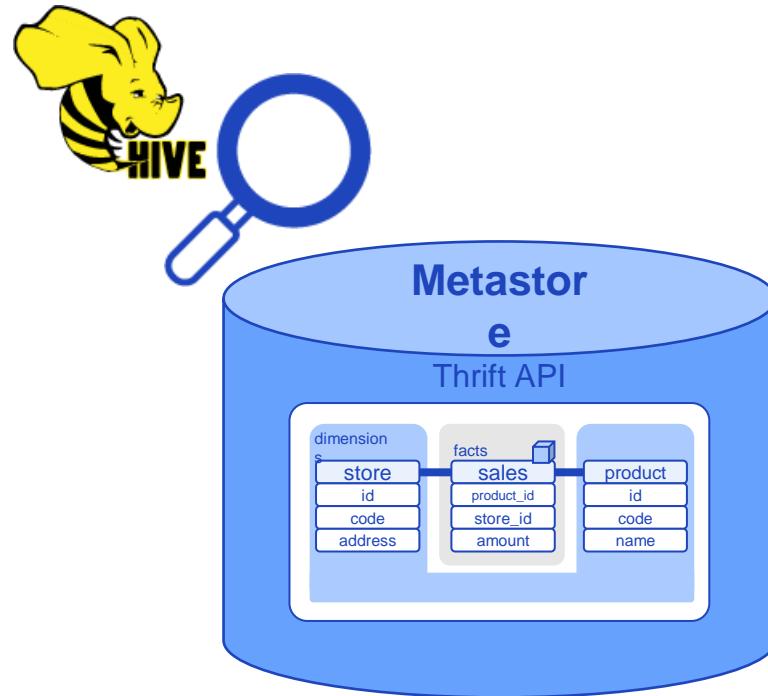
- Hive 1.0
  - It gradually developed starting with version 0.10 in 2012, and version 1.0 was released in February 2015.
  - **MapReduce** processing using SQL
  - Logical representation of file data
  - Aim for batch processing of Big Data
- Hive 2.0
  - In February 2016, version 2.0 was released by improving Hive 1.0. With the advent of LLAP and the default execution engine changed to **TEZ**, performance has been improved.
  - Add Live Long and Process (LLAP) structure
  - Enhanced Spark support

# Version-specific characteristics

- Hive 3.0
  - In May 2018, version 3.0 was released. It has been modified to **deprecate the MapReduce engine** and Hive CLI and process the task using the **TEZ engine and Beeline**.
  - Workload management using roles
  - Enhancing transaction processing
  - Add Materialized View
  - Cache query results to work faster
  - Add table information management database

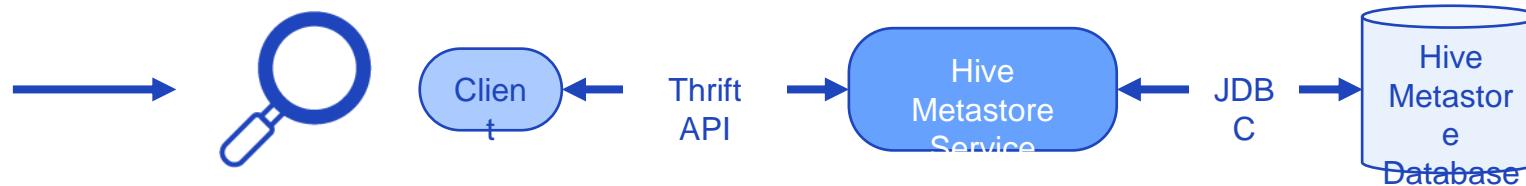
# Apache Hive Metastore

- Hive Metastore
  - Data Schema
  - Database tables and fields
  - Field Type
  - Data location information
  - Data schema is managed in a separate database



# Retrieving data from Apache Hive

- Retrieve data from Hive in two step
  - First step : check the data structures in Metastore  
Retrieve field information, type, and location in a table



- Second step : actual data is from HDFS  
Retrieve real data in HDFS

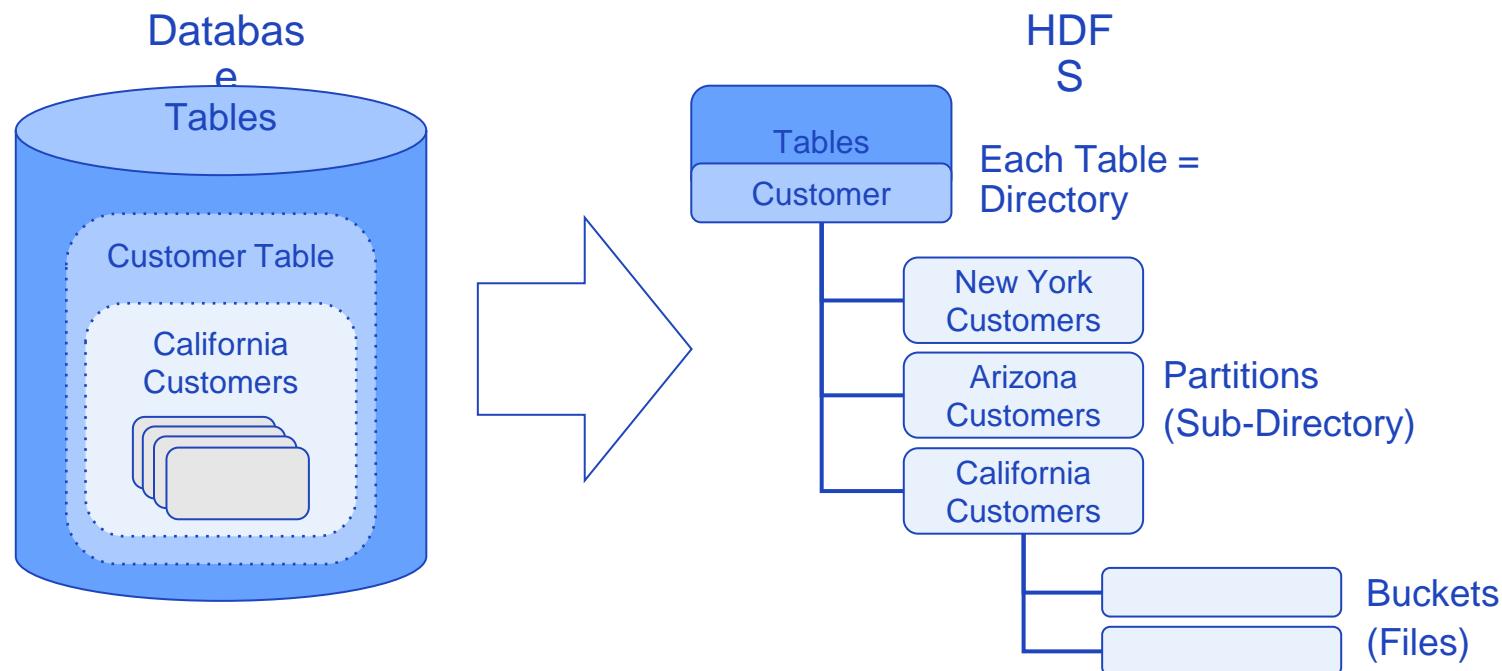


## How Hive works (1/3)

- A Table in Hive is a directory in HDFS
  - All files in the directory are contents of the Table
  - Schema information and how rows and columns are delimited are stored in the Hive Metastore
- Subdirectories within the “Table” directory are user defined partitions
- Metadata (table structure and location for data) is stored in an RDBMS
- Tables are created as either managed or external
  - Managed: if the table is dropped, HDFS data is deleted
  - External : table is dropped, only the table schema is deleted. Data is not deleted

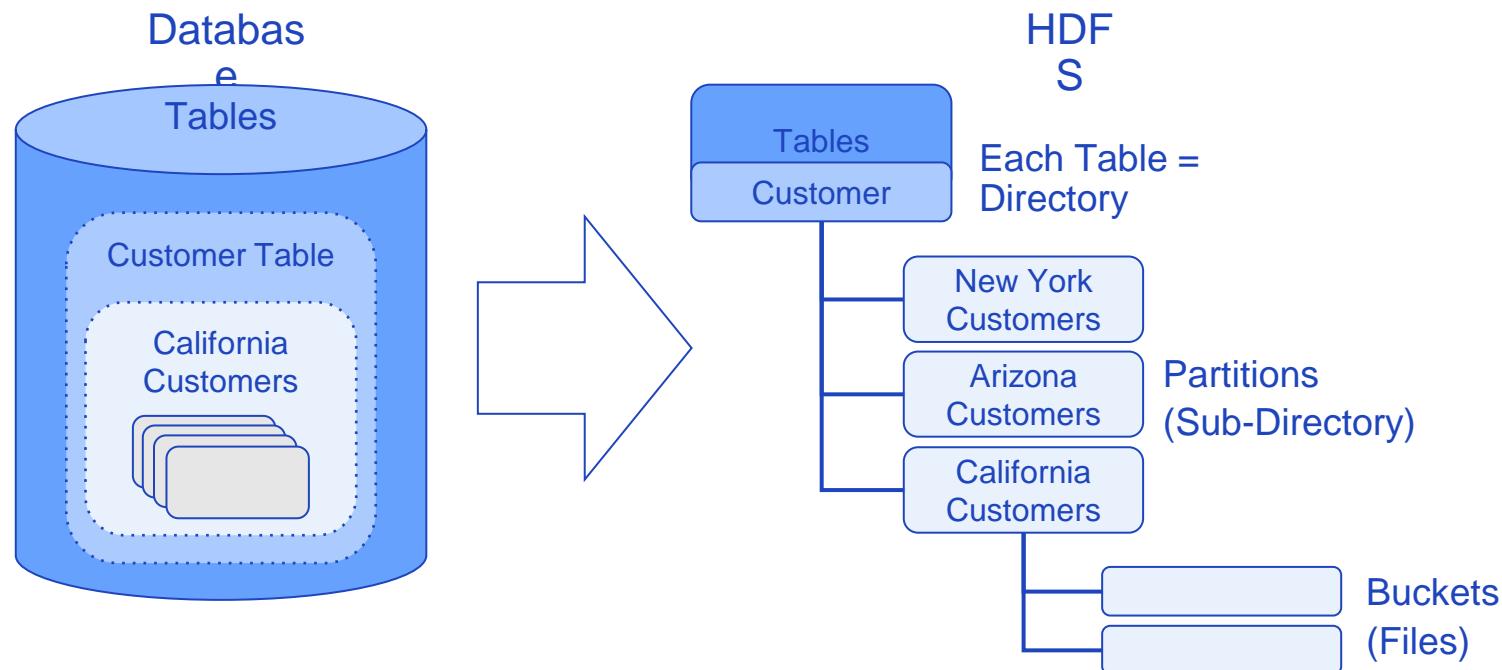
## How Hive works (2/3)

- Hive Data model and HDFS



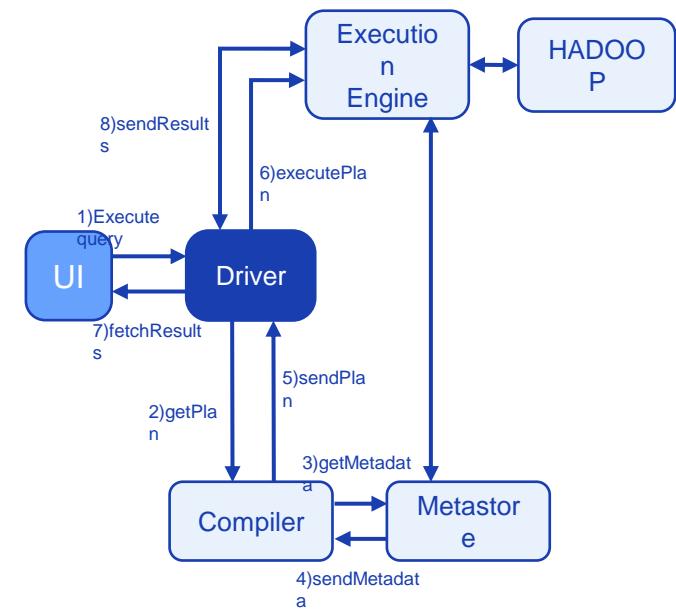
## How Hive works (3/3)

- Hive uses the metastore to determine data schema and location
- The query itself operates on data stored in a filesystem (normally HDFS)



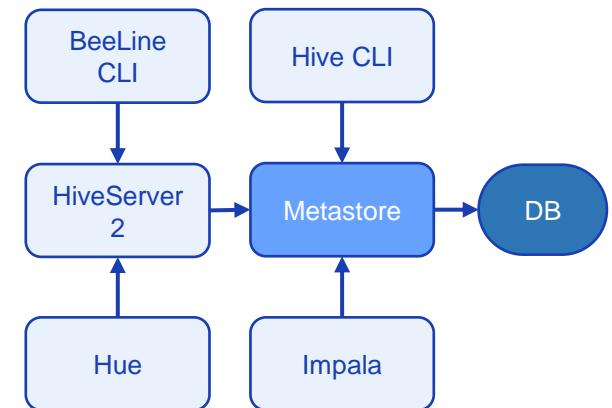
# HIVE Components

- UI
  - User interface through which users submit queries and other actions to the system
  - CLI, Beeline, JDBC, etc.
- Driver
  - Receive query input and process operation
  - Implement user session and provide JDBC/ODBC interface API
- Compiler
  - Refers to the metastore to parse the query and create an execution plan
- Metastore
  - Stores information about db, table, and partition
- Execution Engine



# Hive Service

- metastore
  - The metastore service has a physical database that stores HDFS data structures. The metastore has three execution modes.
  - Embedded: Mode using Derby DB without configuring a separate database. Only one user can access at a time
  - Local: It has a separate database but runs on the same JVM as the hive driver.
  - Remote: Mode that has a separate database and operates independently in a separate JVM
- HiveServer2 (hiveserver2)
  - HiveServer2 provides interworking services with clients developed in other languages.
- beeline
  - This is Hive's command-line interface that works in embedded mode like a normal CLI or accesses the HiveServer2 process with JDBC.



# HIVE CLI

- The Hive Command Line Interface (CLI) is the most basic tool to run Hive queries

| Hive CLI Option                                                              | Description                |
|------------------------------------------------------------------------------|----------------------------|
| hive –e 'select a.col from tab1 a'                                           | Run query                  |
| hive –S –e 'select a.col from tab1 a'                                        | Run query silent mode      |
| hive –e 'select a.col from tab1 a' –hiveconf hive.root.logger=DEBUG, console | Set hive config variables  |
| hive –i initialize.sql                                                       | Use initialization script  |
| hive –f script.sql                                                           | Run non-interactive script |

- Interactive Shell

```
--hive execution
$ hive
hive> select * from product;
```

Show the result

# Beeline

- Beeline is a tool for executing queries by connecting to Hiveserver2 based on SQLLine.
  - Connect to Hive Server 2 using JDBC.
- There are two ways to connect to Hive Server 2

Enter the connection string as a parameter to beeline

```
$ beeline -u jdbc:hive2://localhost:10000 -n jsjeong -p password
jdbc:hive2://localhost:10000> select * from product;
jdbc:hive2://localhost:10000>!quit
```

```
$ beeline
beeline> !connect jdbc:hive2://localhost:10000 -jsjeong password
```

# What is Apache Impala? (1/2)

- Tools for Data Processing and Analytics developed by Cloudera
- Like Hive, HDFS data can be accessed through SQL-like Query statements.
- Unlike Hive, it does not use MR or Spark as Execution Engine.
  - It has its own Query Engine and executes Query through Impala Daemon
  - About 10 to 50 times faster than Hive
  - Simultaneous access to multi-client is provided more efficiently compared to Hive
- Share the Metastore with Hive
  - Tables created in Hive can be used
  - Similarly, tables created in Impala can be used in Hive
- Mainly used for Interactive Query or Data analytics

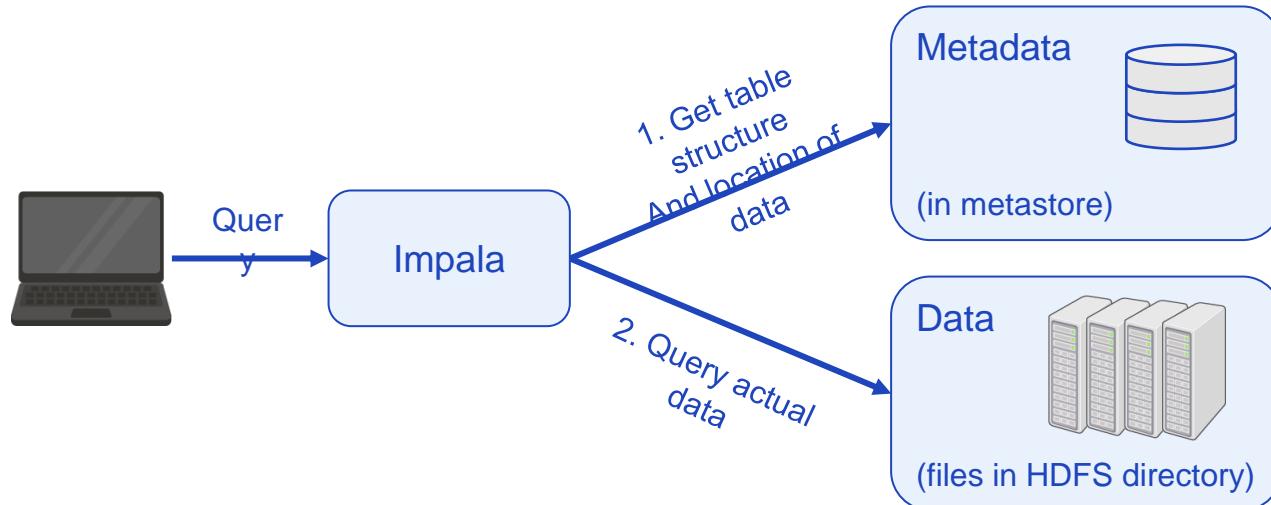


## What is Apache Impala? (2/2)

- Impala is a high-performance SQL engine for vast amounts of data
  - Massively parallel processing (MPP)
- Impala can query data stored in HBase, Kudu, S3, ADLS instead of HDFS
- Read and writes data in common Hadoop file formats
- Directly executes query on the cluster
- Impala Query Language : Impala SQL
  - It is compatible with standard SQL and HiveQL, but there are some differences
  - HiveQL and Impala SQL are very similar

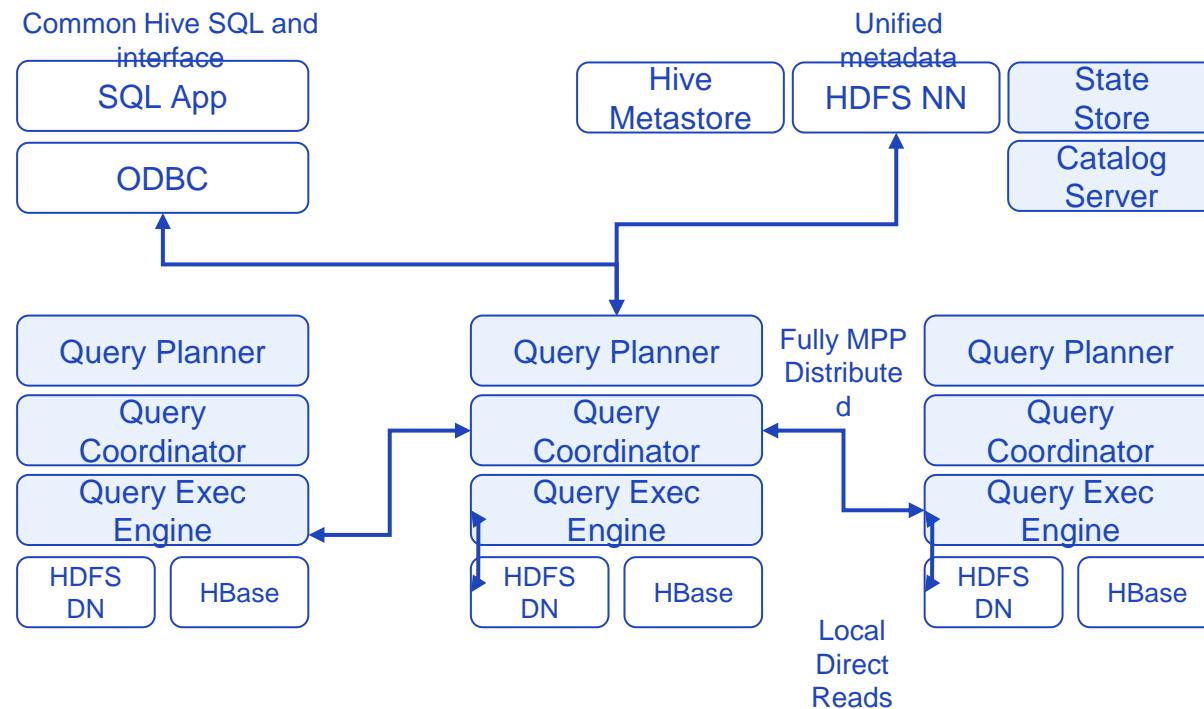
# Impala Query Data

- First, Impala use the metastore to check data schema and location for table
- Then, Query actual table data in HDFS or other storage



# Apache Impala Architecture

- Impala with HDFS



# Apache Impala component (1/2)

- Impala Daemon
  - In general, there is one (or more than one) worker node in the cluster.
  - Execute the query received from the client and send the result
  - Client: Hue Web UI, Impala-shell, JDBC / ODBC, etc.
- Performance improvement factors
  - Working with the HDFS DataNode to which the daemon belongs
  - Utilize local metadata cache
  - Load balancing between Impala daemons in a cluster

## Apache Impala component (2/2)

- Impala State Store
  - One State Store per Cluster
  - Continuously check the status of Impala daemons
- Impala Catalog Server
  - One Catalog Server per Cluster
  - Changes in the results of Query execution in Impala daemon are reflected throughout the cluster
  - Changes can be manually reflected through INVALIDATE METADATA or REFRESH commands
  - The above command is required when changing metadata outside Impala (Table update in E.g. Hive)

# Apache Impala – Impala Shell

- Using the Impala Shell
  - Provides CLI-based Impala-shell Client like Hive Beeline
  - Available on nodes with the Impala daemon installed

```
impala@90d98a18078c:/opt/impala/bin$ impala-shell
Starting Impala Shell without Kerberos authentication
Opened TCP connection to 90d98a18078c:21000
Connected to 90d98a18078c:21000
Server version: impalad version 3.4.0-RELEASE RELEASE (build Cloud not obtain git hash)

```

Welcome to the Impala shell.  
(Impala Shell v3.4.0-RELEASE (9f1c31c) built on Fri Apr 24 14:10:19 PDT 2020)

To see more tips, run the TIP command.

```

[90d98a18078c:21000] default>
```

# Apache Impala – HUE Editor

- Using the HUE editor

The screenshot shows the HUE (Hadoop User Environment) interface for Apache Impala. The top navigation bar includes 'Query' and a search bar. Below the navigation is a toolbar with icons for file operations. On the left, a sidebar shows a 'default' database with tables 'products', 'test', and 'test2'. The main area is divided into two sections: the 'Query Area' (top) and the 'Result Area' (bottom). The 'Query Area' contains the following SQL code:

```
1 INVALIDATE METADATA;
2
3 SELECT * FROM products;
```

The 'Result Area' displays the output of the query as a table:

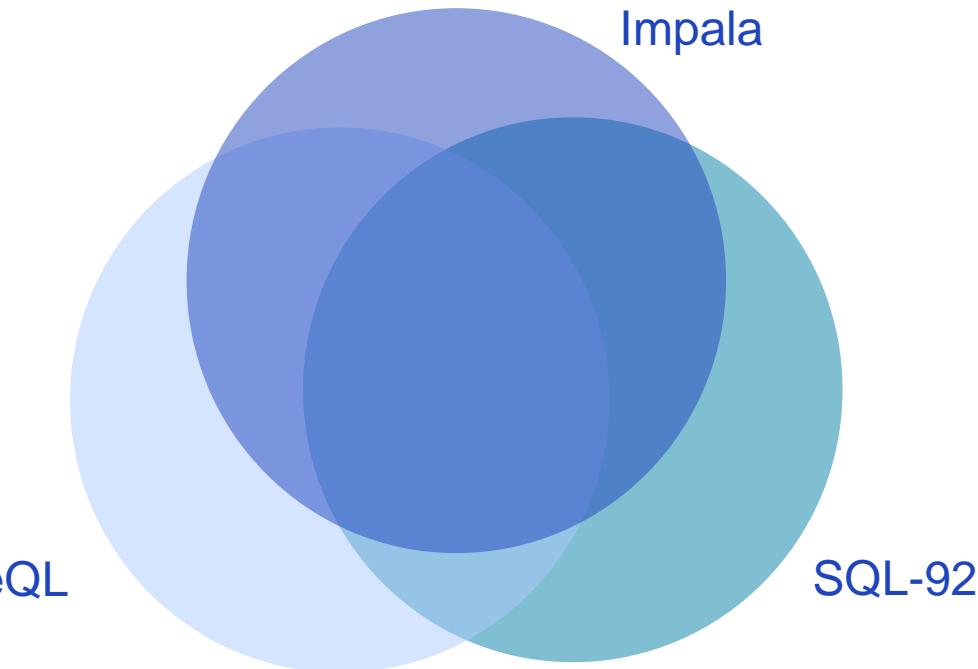
|   | id | name               | price |
|---|----|--------------------|-------|
| 1 | 1  | Blue t-shirts      | 19000 |
| 2 | 2  | Red t-shirts       | 19000 |
| 3 | 3  | Green t-shirts     | 19000 |
| 4 | 4  | Purple t-shirts    | 19000 |
| 5 | 5  | Blue Jeans         | 49000 |
| 6 | 6  | White Shorts       | 39000 |
| 7 | 7  | Black Cotton Pants | 49000 |
| 8 | 8  | Brown Jogger Pants | 49000 |

Query  
Area

Result  
Area

# Query Language for Hive, Impala

- Hive : HiveQL
- Impala: Impala SQL



**We can use almost same query language for Hive and Impala**

# Hive QL

- Hive keywords are not case-sensitive
- Statements are terminated by a semicolon
  - A statement may span multiple lines
- Comments begin with “–”
  - Only supported in Hive scripts (\*.hql, \*.sql)
  - No multi-line comments

## Databases in HDFS

- “default” is name of default database
- Hive and Impala typically use /user/hive/warehouse to store its data
- Hive and Impala support multiple databases
  - Helpful for organization and authorization
- Creating a new database would create a new directory under default directory
- “default” database does not have a separate directory

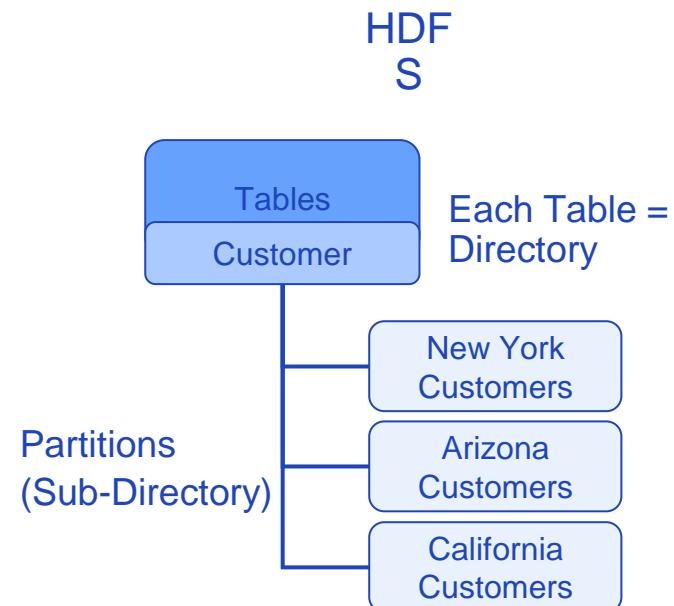
```
$hdfs dfs – ls /user/hive/warehouse
```

Found 2 items

```
drwxrwxrwx – hive supergroup 0 2020-08-10 02:35 /user/hive/warehouse/accounting.db
drwxrwxrwx – hive supergroup 0 2020-10-26 04:55 /user/hive/warehouse/orders
```

# Tables

- Data for Hive and Impala tables is stored on the HDFS
  - Each table maps to a single directory
- A table's directory may contain multiple files
  - Multiple reduces would create multiple files with a directory
  - Subdirectories are not allowed, unless the table is partitioned
- Partitions are subdirectories
- Metastore database is used to map the database schema to the HDFS directory and files.
  - Helps map raw data in HDFS to named columns of specific types



# Database Management

- When creating a new database in hive:
  - A new DB is created in the subfolder /user/hive/warehouse/new\_dbname

```
CREATE DATABASE <DB_NAME>;
```

```
SHOW DATABASES;
```

```
USE DATABASE <DB_NAME>;
```

```
DROP DATABASE <DB_NAME>;
```

# Data Types

- Primitive Data types
  - Numeric Types

| Type     | Description                            |
|----------|----------------------------------------|
| INT      | 4-byte signed integer                  |
| BIGINT   | 8-byte signed integer                  |
| SMALLINT | 2-byte signed integer                  |
| TINYINT  | 1-byte signed integer                  |
| FLOAT    | 4-byte single precision floating point |
| DOUBLE   | 8-byte double precision floating point |
| DECIMAL  | 38-digits number                       |
| NUMERIC  | Same as DECIMAL                        |

# Data Types

- Primitive Data types
  - Date/time, String, Boolean

| Type      | Description                         |
|-----------|-------------------------------------|
| STRING    | VARCHAR() AND CHAR() also available |
| BINARY    | Binary data                         |
| TIMESTAMP | Integer, Float or String            |
| BOOLEAN   | Boolean true or false               |

- Complex Types
  - Array
  - Map
  - Struct
  - Union

# CREATE Table (1/5)

- Managed / External Table
  - Syntax

```
> CREATE [EXTERNAL] TABLE [IF NOT EXISTS]
[<DB_NAME>.]<TBL NAME>
[(<COL_NAME> DATA_TYPE, <COL_NAME> DATA_TYPE, ...)]
[PARTITIONED BY (<COL_NAME> DATA_TYPE)]
[ROW FORMAT <ROW_FORMAT>]
[[STORED AS <FLIE_FORMAT>]
| STORED BY '<storage.handler.class.name>' [WITH SERDEPROPERTIES (...)]]
[LOCATION <HDFS_PATH>]
[TBLPROPERTIES (<property_name>=<property_value>, ...)]
[AS <SELECT_STMT>];
```

## CREATE Table (2/5)

- Create Hive table using products.file
  - Hive managed table : products
  - Table schema

|   | columns | Data types |   | columns | Data types |
|---|---------|------------|---|---------|------------|
| 1 | Id      | Int        | 3 | Price   | int        |
| 2 | name    | string     |   |         |            |

```
CREATE TABLE products (
 id int,
 name string,
 price int
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
LOCATION '/user/jsjeong/products';
```

## CREATE Table (3/5)

- Create Hive table using products.file
  - Hive External table : products
  - Table schema:

|   | columns | Data types |   | columns | Data types |
|---|---------|------------|---|---------|------------|
| 1 | Id      | Int        | 3 | Price   | int        |
| 2 | name    | string     |   |         |            |

```
CREATE EXTERNAL TABLE AB_NYC_2019 (
 id int,
 name string,
 price int
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
LOCATION '/user/jsjeong/products';
```

## CREATE Table (4/5)

- Create a Table with comments, table properties, and alternate location
  - Example

```
CREATE TABLE IF NOT EXISTS mydb.employees (
 name STRING COMMENT 'Employee name',
 salary FLOAT COMMENT 'Employee salary',
 subordinates ARRAY<STRING> COMMENT 'Names of subordinates',
 deductions MAP<STRING, FLOAT> COMMENT 'Keys are deductions names, values are
percentages',
 address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT> COMMENT 'Home
address'
)
COMMENT 'Description of the table'
TBLPROPERTIES ('creator='me', 'created_at'='2021-07-02 10:00:00', ...)
LOCATION '/user/hive/warehouse/mydb.db/employees';
```

## CREATE Table (5/5)

- Copy the schema of a table.
- You can also change the location where table will be created. No other properties can be modified and follows original table.

Example

```
CREATE TABLE IF NOT EXISTS mydb.employees2
LIKE mydb.employees
LOCATION /my/preferred/location;
```

```
hive> USE mydb;
hive> SHOWTABLES;
employees
table1
table2
```

or other



```
hive> USE default;
hive> SHOWTABLES IN mydb;
employees
table1
table2
```

# DESCRIBE Table

- Show the table information.

```
> DESCRIBE <TBL_NAME> // (or) DESC <TBL_NAME>
```

| col_name | data_type | comment |
|----------|-----------|---------|
| id       | int       |         |
| name     | string    |         |
| price    | int       |         |

<Describe products>

- ```
> DESCRIBE FORMATTED <TBL_NAME> // (or) DESC FORMATTED <TBL_NAME>
```

DROP Table & ALTER Table

- Drop table ‘products’
 - In the case of an external table, even if the table is deleted, the directory in HDFS is not deleted

```
DROP TABLE products;
```

- ```
ALTER TABLE products RENAME products_2019;
ALTER TABLE products CHANGE id prod_id int;
ALTER TABLE products ADD COLUMNS(last_modified date);
```

```
ALTER TABLE products REPLACE COLUMNS (
 hours_mins_secs INT COMMENT 'hour, minute, seconds from timestamp',
 severity STRING COMMENT 'The message severity'
 message STRING COMMENT 'The rest of the message'
);
```

# ALTER Table

- Various alter table commands for Partition
  - Add Partitions to the Table

```
ALTER TABLE log_messages ADD IF NOT EXISTS
PARTITION (year = 2021, month = 1, day = 1) LOCATION '/logs/2021/01/01'
PARTITION (year = 2021, month = 1, day = 2) LOCATION '/logs/2021/01/02'
```

```
ALTER TABLE log_messages PARTITION(year = 2021, month = 7, day = 2)
SET LOCATION 's3n://ourbucket/logs/2021/07/02';
```

```
ALTER TABLE log_messages DROP IF EXISTS
PARTITION(year = 2021, month = 7, day = 2);
```

# Query data from Table

- The SELECT statement retrieves records from tables
  - Specify individual columns
  - Column names are case-insensitive

```
SELECT column1, column2, column FROM table_name;
```

```
SELECT * FROM table_name;
```

# Review Questions

[Quiz]

Quiz 1

- What command do you use to switch to another database?

Quiz 2

- When you first start Impala or Hive, what is the name of the selected database?

Quiz 3

- Where is metadata about a table (such as the table's properties and column names) stored?

Quiz 4

- Where is the data for partitioned tables stored?

## [Lab6]

# Running Basic Queries with Hive QL



Unit 3.

# Advanced Analytics

Big Data Analytics

Unit 3

# Advanced Analytics

- | 3.1. Hive and Impala Data Management
- | 3.2. Complex Data and Relational Data Analysis

# Choosing a file format

- Hive and Impala support many different file formats for data storage
  - Syntax

```
CREATE TABLE DB_name.table_name (col_name DATATYPE,...)
ROW FORMAT DELIMITED FIELDS TERMINATED BY char
STORED AS supported_format;
```

- Supported file format
  - Textfile
  - Sequencefile
  - AVRO
  - Parquet
  - Orc (Hive only)

# File Formats

- TEXTFILE
  - Typically comma or tab separated values (CSV, TSV)
- SEQUENCEFILE
  - Flat file consisting of binary key/value pairs
  - Uncompressed
  - Record Compressed - only values are compressed
- Parquet – a columnar storage format
  - Available to any component in the Hadoop ecosystem, regardless of the data processing framework, data model, or programming language.
  - Best combination with impala
- ORC – Optimized Row Columnar
  - Designed to overcome limitations of other file formats
  - Using ORC files improves performance

# Using Parquet format

- Create external table using parquet format
- When you use STORED AS PARQUET, Impala and Hive automatically take care of setting how the records are encoded in the Parquet files.

```
CREATE EXTERNAL TABLE authors_parquet
(
 first_name string,
 last_name string
)
STORED AS PARQUET
```

```
0: jdbc:hive2://> create external table authors_parquet (
 > f_name string,
 > l_name string)
 > STORED AS PARQUET;
OK
No rows affected (0.086 seconds)
```

# Using Parquet format in Impala

- Create a new table to access an existing Parquet file in HDFS
  - In Impala, use LIKE PARQUET to create a table using the schema of an existing parquet file.

```
CREATE EXTERNAL TABLE author_parquet
 LIKE PARQUET '/home/student/authors_parquet/data1.parquet'
 STORED AS PARQUET
 LOCATION '/home/student/authors_parquet/';
```

# HiveQL Select query

- HiveQL Semantics

|                                                      |
|------------------------------------------------------|
| SELECT, LOAD INSERT from query                       |
| Expressions in WHERE and HAVING                      |
| GROUP BY, ORDER BY, SORT BY                          |
| Sub-queries in FROM clause                           |
| GROUP BY, ORDER BY                                   |
| UNION                                                |
| LEFT, RIGHT and FULL INNER/OUTER JOIN                |
| CROSS JOIN, LEFT SEMI JOIN                           |
| Windowing functions (OVER, RANK, etc.)               |
| INTERSECT, EXCEPT, UNION, DISTINDT                   |
| Sub-queries in WHERE (IN, NOT IN, EXISTS/NOT EXISTS) |

# Query data from Table

- The SELECT statement retrieves records from tables
  - Specify individual columns
  - Column names are case-insensitive

```
SELECT column1, column2, columnN FROM table_name;
```

```
SELECT * FROM table_name;
```

## Sort & Distinct the Query

- Use DISTINCT to remove duplicates

```
SELECT DISTINCT name FROM authors;
```

- The ORDER BY sorts the result

- Default order is ascending (ASC)

```
SELECT id, first_name, last_name, email FROM authors
ORDER BY last_name;
```

# Limiting Query Results

- The LIMIT clause sets the maximum number of rows returned

```
SELECT id, first_name, last_name FROM authors
ORDER BY last_name, first_name DESC LIMIT 10;
```

- Table Aliases

```
SELECT a.first_name, a.last_name, a.email, p.title
FROM authors AS a JOIN posts AS p
ON (a.id = p.author_id)
WHERE a.email LIKE '%.net' OR
a.Email LIKE '%.org' LIMIT 10;
```

| id   | first_name | last_name |
|------|------------|-----------|
| 4572 | Willow     | Abbott    |
| 3574 | Vanessa    | Abbott    |
| 6071 | Turner     | Abbott    |
| 2923 | Tiara      | Abbott    |
| 6503 | Stanley    | Abbott    |
| 4169 | Sincere    | Abbott    |
| 1780 | Melany     | Abbott    |
| 3310 | Mario      | Abbott    |
| 9290 | Mario      | Abbott    |
| 4829 | Magdalena  | Abbott    |

10 rows selected (19.667 seconds)

# Filter Results with WHERE

- WHERE clauses restrict rows to those matching specified criteria

```
SELECT * FROM authors WHERE id=9290;
```

- IN operator

```
SELECT * FROM authors
WHERE first_name IN ("Mario", "Malany", "Dorthy", "Tiara", "Willow");
```

- ```
SELECT * FROM authors
WHERE first_name LIKE 'Do%'
AND (birthdate = '1980-04-20' OR birthdate = '1983-02-12');
```

Subqueries in the FROM Clause

- A subquery is another SQL statement included in one SQL statement
- Hive and Impala support subqueries in the FROM clause
 - The subquery used in the FROM clause is called an inline view (dynamic view)
 - The subquery must be name(sub_name)

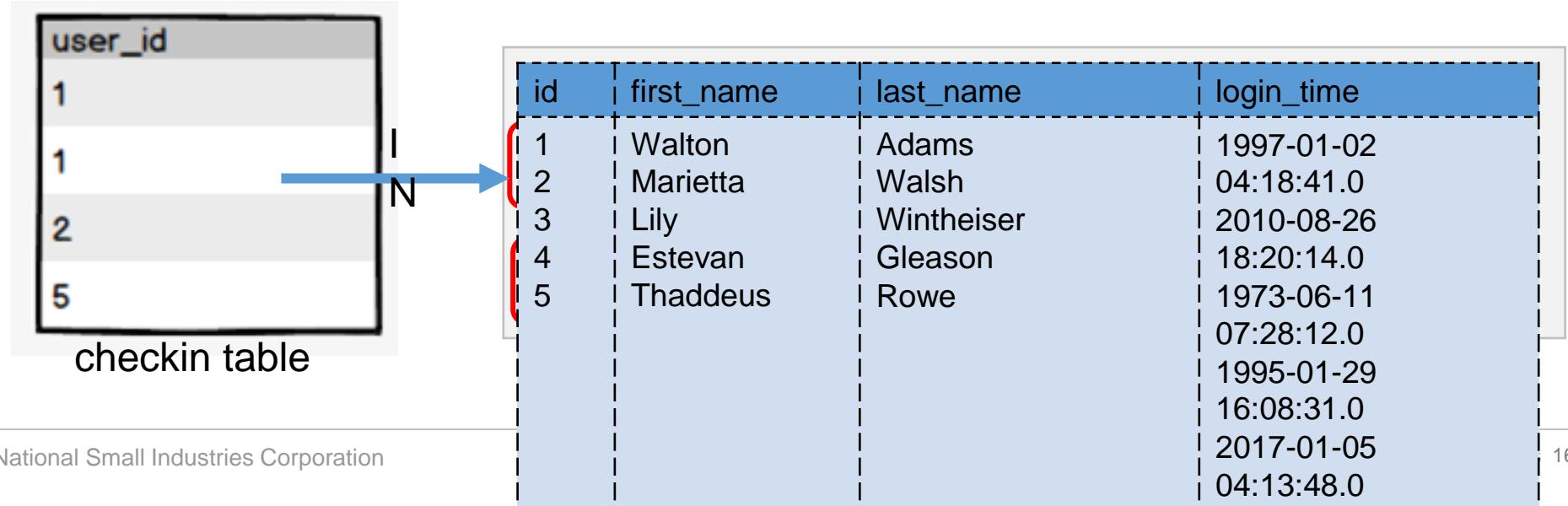


```
SELECT order_item
FROM (SELECT order_item, COUNT(*) AS cnt
      FROM order_table GROUP BY order_item) big_order
WHERE cnt >=10;
```

Subqueries in the WHERE Clause

- Hive and Impala support subqueries in the WHERE clause

```
SELECT u.first_name, u.last_name, login_time FROM users u
WHERE u.id IN ( SELECT c.uid FROM checkin c );
```



Querying and Inserting Data (1/2)

- Input and inquiry are operations of reading and writing data using meta information of Hive table
- The table's meta information stores the location of the actual file and the format of the data
- For input (INSERT),
 - There are two ways to write data to a table with an INSERT statement, and to copy a file to the storage location of the table and to the LOCATION specified when creating the table
 - A partition table consists of a directory divided into partitions and copies files
- For query(SELECT),
 - A query (SELECT) reads a file at the location of LOCATION in the table

Querying and Inserting Data (2/2)

- The Insert and Select methods include
 - A method by reading a file and writing to a table
 - A method by reading data from a table and writing to another table
 - A method by reading a table and writing it to a specified directory
- File to table
 - Write to table with LOAD command
 - Copy the file to the table's LOCATION
- Table to table
 - INSERT statement
 - CREATE TABLE AS SELECT statement
- Table to directory
 - INSERT DIRECTORY statement

Loading Data

- Write to table with LOAD command
 - Execute a command on Hive and Impala CLI
 - Data source can be either a file or directory

Syntax

```
LOAD DATA [LOCAL] INPATH source [OVERWRITE] INTO TABLE table_name [PARTITION  
(partcol1=value1, partcol2=value2 ...)]
```

```
LOAD DATA INPATH '/user/student/data/product.txt' INTO table product;
```

-

```
$ hdfs dfs -mv product.txt /user/hive/warehouse/product/
```

Copy the file to the table's LOCATION (1/2)

- The method of specifying the table's LOCATION information can be specified during CREATE
 - Just copy or move the data to this location
 - Can not specify a local location,
 - ~~Must use a file sharing file system that can be accessed by Hadoop, such as HDFS or S3~~

```
CREATE TABLE product (
    id integer,
    name string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LOCATION '/user/student/data';
```

Copy the file to the table's LOCATION (2/2)

- Using Alter command for location change

```
hive > create table product (
    >   name string
    > )
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
OK
Time taken: 0.089 seconds
hive > desc product;
OK
id          int
name        string
Time taken: 0.058 seconds, Fetched: 2 row(s)
hive > select * from product;
OK [1]
Time taken: 0.206 seconds
hive > ALTER TABLE product SET location '/user/student/data';
OK [3]
Time taken: 0.132 seconds
hive >
>
> select * from product; [4]
OK
1  hadoop
2  hive
3  impala
4  pig
5  sqoop
6  flume
7  kafka
8  nifi
9  hdfs
10 yarn
11 spark
12 flink
Time taken: 0.161 seconds, Fetched: 12 row(s)
hive >
```

```
[student@localhost ch5]$ hdfs dfs -put product.txt data
[student@localhost ch5]$ hdfs dfs -ls data
Found 1 items
[student@localhost ch5]$ hdfs dfs -cat data/product.txt
[2]
1  hadoop
2  hive
3  impala
4  pig
5  sqoop
6  flume
7  kafka
8  nifi
9  hdfs
10 yarn
11 spark
12 flink
[student@localhost ch5]$
```

Query output to a Table

- INSERT statement
 - As a basic data entry method, data from a table or view is entered into another table

~~Basic syntax of the INSERT statement is as follows:~~

```
INSERT OVERWRITE TABLE tablename [PARTITION (partcol1=value1, partcol2=value2 ...)] [IF NOT EXISTS] select_statement FROM statement;
```

```
INSERT INTO TABLE tablename [PARTITION (partcol1=value1, partcol2=value2 ...)] select_statement  
FROM statement;
```

```
INSERT INTO TABLE new_product  
SELECT * FROM product  
WHERE id > 10 and name = 'spark';
```

CREATE TABLE AS SELECT

- CTAS statements populate data while creating tables
 - Known as CREATE TABLE AS SELECT (CTAS)

```
CREATE TABLE new_product
ROW FORMAT DELIMITED FIELDS
TERMINATED BY ','
AS SELECT name, id
FROM product
WHERE id > 5;
```

0: jdbc:hive2://> select * from
new_product;
OK

new_product.name	new_product.id
hdfs	6
yarn	7
sqoop	8
flume	9
kafka	10
nifi	11
hive	12
pig	13
kudu	14
hbase	15

10 rows selected (0.213 seconds)

Writing Output to HDFS in Hive (1/3)

- It reads data from the table and outputs a file to the specified location
 - Set how to save data using ROW FORMAT
 - The basic syntax

```
INSERT OVERWRITE [LOCAL] DIRECTORY directory [ROW FORMAT row_format] [STORED AS  
file_format] SELECT ... FROM ...
```

```
FROM from_statement
```

```
INSERT OVERWRITE [LOCAL] DIRECTORY directory1 select_statement 1
```

```
[INSERT OVERWRITE [LOCAL] DIRECTORY directory2 select_statement2] ... row_format :
```

```
DELIMITED [FIELDS TERMINATED BY char
```

Writing Output to HDFS in Hive (2/3)

- Hive also save output to a directory in HDFS

```
0: jdbc:hive://> insert overwrite directory '/user/student/product/'  
    . . . . . > row format delimited fields terminated by '\t'  
    . . . . . > select * from new_product  
    . . . . . > where id >= 10;
```

- [student@localhost ~]\$ hdfs dfs -ls product
Found 1 items
-rw-r--r-- 1 student student 49 2021-07-25 18:42 product/000000_0
[student@localhost ~]\$ hdfs dfs -cat product/000000_0
kafka 10
nifi 11
hive 12
pig 13
kude 14
hbase 15

Writing Output to HDFS in Hive (3/3)

- The following query produces the same result for product

```
FROM product p
INSERT OVERWRITE DIRECTORY 'product_from'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
SELECT name, id
WHERE id >= 10;
```

```
[student@localhost ~]$ hdfs dfs -ls product
Found 1 items
-rw-r--r--      1 student student          49 2021-07-25 18:42 product/000000_0
[student@localhost ~]$ hdfs dfs -cat product/000000_0
kafka      10
nifi       11
hive       12
pig        13
kude       14
hbase      15
```

Saving to multiple directories

- Extract data to multiple files
 - The result is two directories in HDFS

```
FROM product p
INSERT OVERWRITE DIRECTORY 'product_name'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
SELECT name
WHERE id < 10
INSERT OVERWRITE DIRECTORY 'product_id'
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
SELECT count(id), id
WHERE id >= 10
GROUP BY id;
```

- Hive can generate two directory results using a single query, which is more efficient than using multiple queries.

Create and Delete a View

- Create a view at the time of executing a SELECT statement

- Syntax

```
CREATE VIEW [IF NOT EXISTS] view_name [(column_name [COMMENT column_comment],  
...)]
```

```
[COMMENT table_comment] AS SELECT ...
```

- A view is a temporary storage table and saved query on a table or set of tables.

```
CREATE VIEW author_100 AS  
SELECT * FROM authors  
WHERE id < 101;
```

```
SELECT * FROM author_100;
```

- ```
DROP VIEW author_100;
```

## Managing the Views (1/2)

- List a view

```
SHOW TABLES;
```

- See a view's underlying query

```
DESCRIBE FORMATTED author_100;
```

- 
- SHOW CREATE TABLE author\_100;

```
0: jdbc:hive2://> show create table author_100;
OK
```

createtab\_stmt

```
CREATE VIEW 'author_100' AS select 'authors'.id', 'authors'.first_name', 'authors'.
'last_name', 'authors'.email', where 'authors'.id' < 101
```

## Managing the Views (2/2)

- Change the underlying query

```
ALTER VIEW author_100
AS select id, first_name, email, birthdate from authors;
```

- To rename a view:

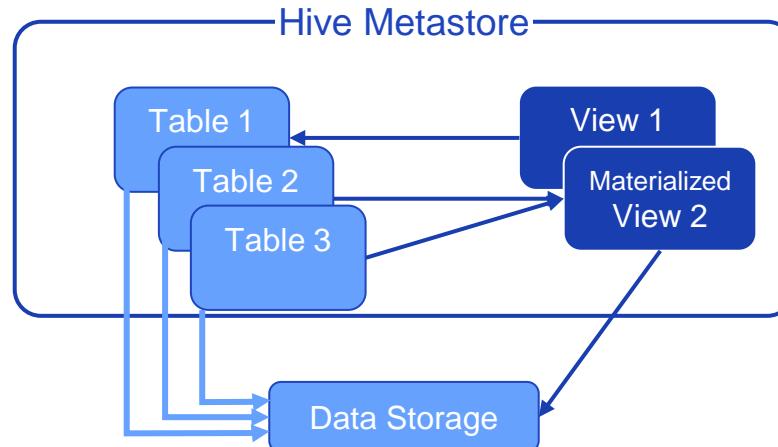
```
ALTER VIEW author_100
RENAME TO author_hundred;
```

- **DROP VIEW** author\_hundred;

# Materialized Views in Hive

- Materialized views persist the results so the query does not run every time
  - Available in Hive 3.0
- In order to enable query rewriting using Materialized views, the global property is needed:

```
SET hive.materializedview.rewriting=true;
```



Unit 3

# Interactive Analytics

- | 3.1. Hive and Impala Data Management
- | 3.2. Complex Data and Relational Data Analysis

## Table Partitioning (1/4)

- Partitions store data separated into directories
  - File-based tables like Hive basically read all the row information in the table, so they slow down when there is a lot of data
  - A partition column can be used like a column in a where condition, so it reduces the data read at the beginning and improves processing speed

- ```
CREATE TABLE author hundred (
    column type,
    ...
)
PARTITIONED BY (column type)
ROW FORMAT DELIMITED FIEDLS TERMINATED BY '\t';
```

Table Partitioning (2/4)

- The product_category is a non-partitioned table

```
CREATE TABLE product_category (
    id integer,
    name string,
    category string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LOCATION '/user/student/data2';
```

Table Partitioning (3/4)

- Product_category's files are stored in single directory, “/user/student/data2”
- All files are scanned for query

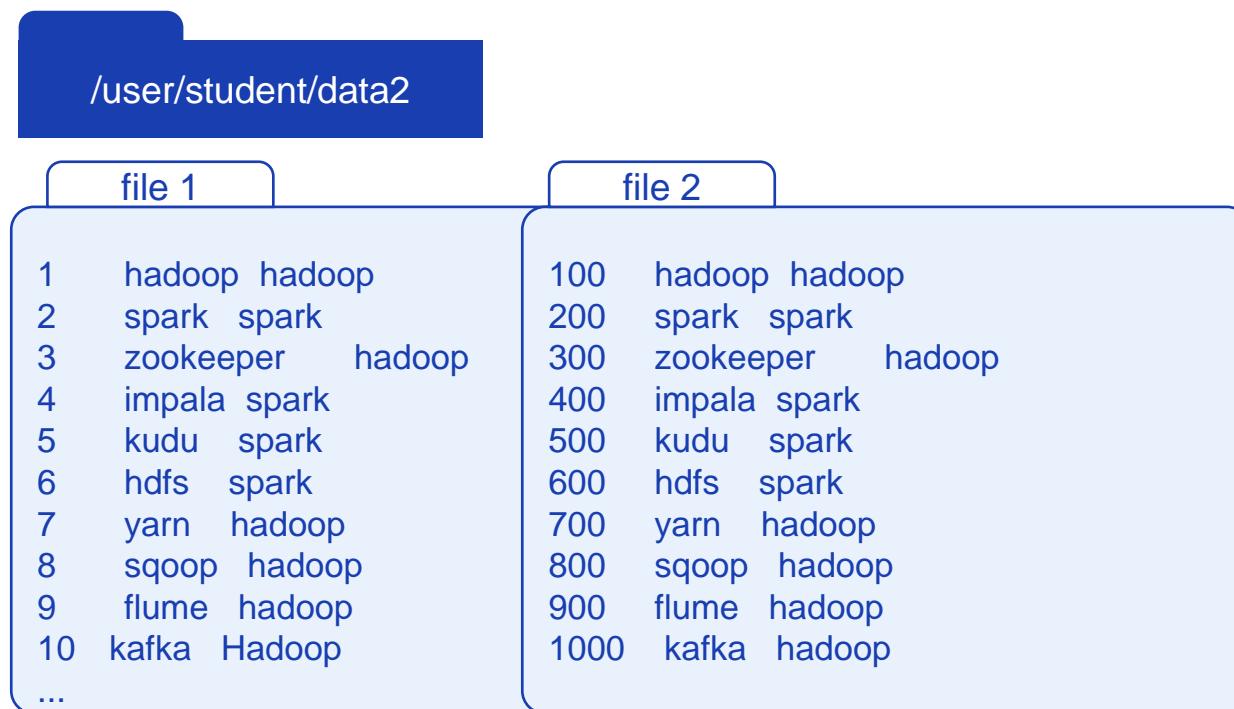


Table Partitioning (4/4)

- Partitioned tables store data in subdirectories

```
CREATE TABLE product_by_category (
    id integer,
    name string
)
PARTITIONED BY (category string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LOCATION '/user/student/product_by_category';
```

```
hive> desc product_by_category;
OK
id      int
name    string
category string
```

```
# Partition Information
# col_name          data_type        comment
category          string
```

```
Time taken: 0.446 seconds, Fetched: 7 row(s)
```

Partition type (1/4)

- There are two types of partitions: dynamic partition and static partition
 - Static partition : Data input using a static partition is entered by passing partition information as a fixed value to the INSERT statement
 - Dynamic partition : Data input using dynamic partitions is entered by passing a column for querying partition information to the INSERT statement
- Static Partition
 - When loading data, you specify which partition to store it

```
ALTER TABLE product_by_category
ADD PARTITION (category='hadoop');

INSERT OVERWRITE table product_by_category
PARTITION(category='hadoop')
SELECT id, name from product_category
WHERE category = 'hadoop';
```

Partition type (2/4)

- If the previous slide command is executed, the following folder is created in hdfs and data is entered

- HDFS directory: /user/student/product_by_category/category=hadoop/

```
[student@localhost ~] hdfs dfs -cat product_by_category/category=Hadoop/000000_0
```

```
1 Hadoop
3 zookeeper
7 yarn
8 sqoop
9 flume
10 kafka
11 nifi
12 hive
100 Hadoop
300 zookeeper
700 yarn
800 sqoop
900 flume
10000 kafka
[student@localhost ~]
```

product_by_category.id	product_by_category.name	product_by_category.category
1	hadoop	hadoop
3	zookeeper	hadoop
7	yarn	hadoop
8	sqoop	hadoop
9	flume	hadoop
10	kafka	hadoop
11	nifi	hadoop
12	hive	hadoop
100	Hadoop	hadoop
300	zookeeper	hadoop
700	yarn	hadoop
800	sqoop	hadoop
900	flume	hadoop
10000	kafka	hadoop

Partition type (3/4)

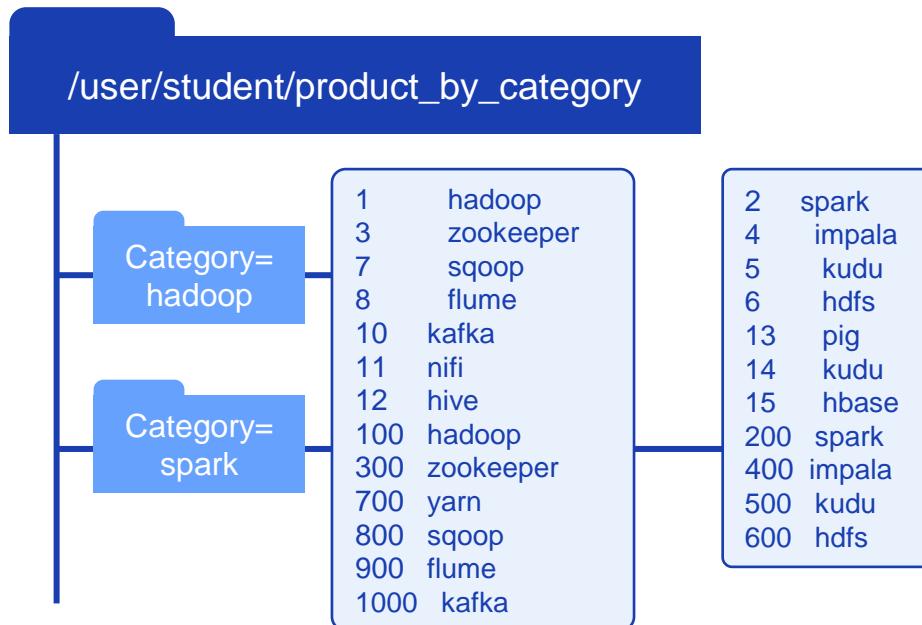
- Dynamic partition
 - Use an INSERT statement to load data
 - The partition column must be specified last in the SELECT columns
 - Hive doesn't recommend using only dynamic partitions by default
 - Set `hive.exec.dynamic.partition.mode=nonstrict`

```
SET hive.exec.dynamic.partition.mode=nonstrict;
```

```
INSERT OVERWRITE TABLE product_by_category
PARTITION(category)
SELECT name, id, category from product_category;
```

Partition type (4/4)

- If the previous slide command is executed, the following folder is created in hdfs and data is entered
 - HDFS directory: /user/student/product_by_category/category=hadoop/
 - HDFS directory: /user/student/product_by_category/category=spark/



0: jdbc:hive2://> select * from product_by_category;

OK

product_by_category.id	product_by_category.name	product_by_category.category
1	hadoop	hadoop
3	zookeeper	hadoop
7	yarn	hadoop
8	sqoop	hadoop
9	flume	hadoop
10	kafka	hadoop
11	nifi	hadoop
12	hive	hadoop
100	hadoop	hadoop
300	zookeeper	hadoop
700	yarn	hadoop
800	sqoop	hadoop
900	flume	hadoop
1000	kafka	hadoop
2	spark	spark
4	impala	spark
5	kudu	spark
6	hdfs	spark
13	pig	spark
14	kudu	spark
15	hbase	spark
200	spark	spark
400	impala	spark
500	kudu	spark
600	hdfs	spark

Nested Partitions

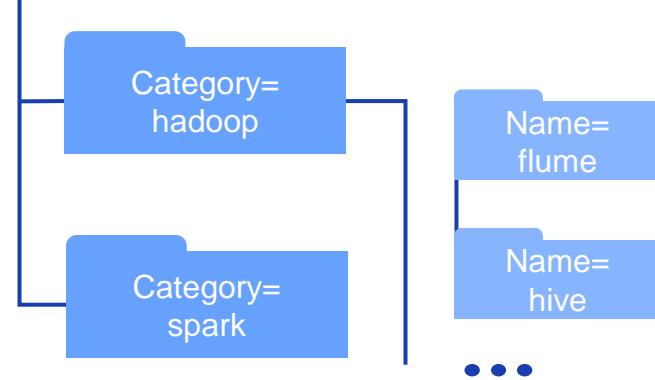
- When the table is partitioned using multiple columns, then Hive creates nested sub-directories based on the order of the partition columns

```
CREATE TABLE product_by_category (id int)
PARTITIONED BY (category string, name string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LOCATION '/user/student/product_by_category';
```

```
0: jdbc:hive2://> desc product_by_category;
OK
```

col_name	data_type	comment
id	int	
category	string	
name	string	
# Partition Information	NULL	NULL
# col_name	NULL	comment
category	data_type	
name	string	

/user/student/product_by_category



Hive Properties in Partitions

- There is a limit to the number of creations of dynamic partitions because dynamic partitions are slow and can create a huge number of partitions
- When creating more partitions than the default setting, the following settings should be made
 - set `hive.exec.dynamic.partition.mode=strict`;
 - number of dynamic partitions

```
SET hive.exec.dynamic.partition.mode=strict;
SET hive.exec.max.dynamic.partitions=1000;
SET hive.exec.max.dynamic.partitions.pernode=100;
```

Managing the Partitions (1/2)

- To show the current partitions in table

```
SHOW PARTITIONS product_by_category;
```

```
0: jdbc:hive2://> show partitions product_by_category;
OK
```

partition
category=hadoop
category=nosql
category=spark

```
category=hadoop
category=nosql
category=spark
```

```
3 rows selected (1.591 seconds)
```

- Modify/Delete Partition
 - To modify or delete a partition, use the ALTER statement

```
ALTER TABLE product_by_category ADD PARTITION (category='nosql');
```

Managing the Partitions (2/2)

- Modify the partition's LOCATION

```
ALTER TABLE product_by_category PARTITION  
(category='nosql') SET LOCATION '/user/student/data3';
```

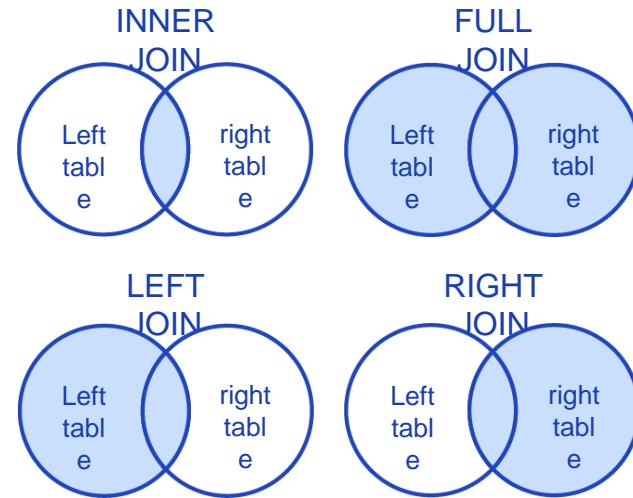
- Delete partition

```
ALTER TABLE product_by_category DROP PARTITION (category='nosql');
```

- ```
ALTER TABLE product_by_category DROP PARTITION (category < 'nosql');
```

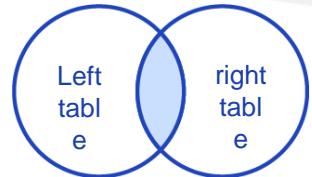
# Joins

- To combine records from two or more tables in the database we use JOIN clause
- Hive and Impala support several types of Joins operations
  - Basically, there are 4 types of Join  
INNER JOIN  
LEFT OUTER JOIN  
RIGHT OUTER JOIN  
FULL OUTER JOIN



- `table_reference JOIN table_factor [join_condition]`
  - | `table_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table_reference join_condition`
  - | `table_reference LEFT SEMI JOIN table_reference join_condition`
  - | `table_reference CROSS JOIN table_reference [join_condition]`

# Inner Join



- INNER JOIN includes records with matching key values between Table A and Table B

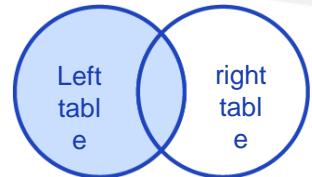
```
SELECT c.name, c.capital, c.population, c1 cname FROM country c
INNER JOIN continent_info c1 ON c.continent_id = c1.id;
```

| Country   |               |            |       |
|-----------|---------------|------------|-------|
| name      | capital       | population | Co_id |
| B_country | Africa        | 2083850    | 5     |
| E_country | Asia          | 963089     | 1     |
| F_country | Europe        | 671580     | 3     |
| G_country | Oceania       | 823494     | 3     |
| K_country | North America | 1266720    | 2     |
| D_country | South America | 5256570    | 4     |

| Continent_info |               |
|----------------|---------------|
| T_id           | cname         |
| 1              | Africa        |
| 2              | Asia          |
| 3              | Europe        |
| 4              | Oceania       |
| 5              | North America |
| 6              | South America |

| <Result   |           |              |               |
|-----------|-----------|--------------|---------------|
| c.name    | c.capital | c.population | c1 cname      |
| B_Country | B_City    | 2083850      | North America |
| E_Country | E_City    | 963089       | Africa        |
| F_Country | F_City    | 671580       | Europe        |
| G_Country | G_City    | 823494       | Europe        |
| K_Country | K_City    | 1266720      | Asia          |
| D_Country | D_City    | 5256570      | Oceania       |
| A_Country | A_City    | 3253651      | South America |

# Left Outer Join



- A LEFT JOIN returns all the values from the left table, plus the matched values from the right table, or NULL in case of no matching JOIN predicate

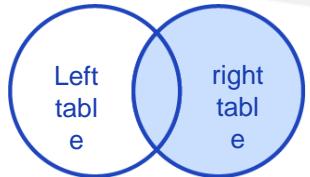
```
SELECT * FROM a LEFT OUTER JOIN b ON a.sid = b.cid;
```

| A table |       |        | B     |     |         |
|---------|-------|--------|-------|-----|---------|
| sid     | count | p_name | table | cid | service |
| 1       | 10    | hadoop | 1     | 1   | hdfs    |
| 3       | 20    | spark  | 2     | 1   | yarn    |
| 5       | 30    | nosql  | 3     | 3   | kudu    |
| 7       | 100   | spark  | 4     | 3   | NULL    |
| 9       | 3     | NULL   | 5     | 5   | NULL    |
|         |       |        | 7     | 7   | impala  |

**<Result >**

| a.sid | a.node_count | a.platform_name | b.id | b.cid | b.service |
|-------|--------------|-----------------|------|-------|-----------|
| 1     | 10           | hadoop          | 1    | 1     | hdfs      |
| 1     | 10           | hadoop          | 2    | 1     | yarn      |
| 3     | 20           | spark           | 3    | 3     | kudu      |
| 3     | 20           | spark           | 4    | 3     | NULL      |
| 5     | 50           | nosql           | 5    | 5     | NULL      |
| 7     | 100          | spark           | 7    | 7     | impala    |
| 9     | 3            | NULL            | NUL  | NUL   | NULL      |
|       |              |                 | L    | L     |           |

# Right Outer Join



- A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table, or NULL in case of no matching join predicate

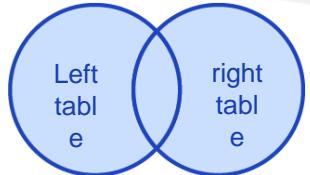
```
SELECT * FROM a RIGHT OUTER JOIN b ON a.sid = b.cid;
```

| A table |       |        | B     |    |        |
|---------|-------|--------|-------|----|--------|
| sid     | count | p_name | table | id | cid    |
| 1       | 10    | hadoop | 1     | 1  | hdfs   |
| 3       | 20    | spark  | 2     | 1  | yarn   |
| 5       | 30    | nosql  | 3     | 3  | kudu   |
| 7       | 100   | spark  | 4     | 3  | NULL   |
| 9       | 3     | NULL   | 5     | 5  | NULL   |
|         |       |        | 7     | 7  | impala |

**<Result >**

| a.sid | a.node_count | a.platform_name | b.id | b.cid | b.service |
|-------|--------------|-----------------|------|-------|-----------|
| 1     | 10           | hadoop          | 1    | 1     | hdfs      |
| 1     | 10           | hadoop          | 2    | 1     | yarn      |
| 3     | 20           | spark           | 3    | 3     | kudu      |
| 3     | 20           | spark           | 4    | 3     | NULL      |
| 5     | 50           | nosql           | 5    | 5     | NULL      |
| 7     | 100          | spark           | 7    | 7     | impala    |

# Full Outer Join



- The HiveQL FULL OUTER JOIN combines the records of both the left and the right outer tables that fulfil the JOIN condition

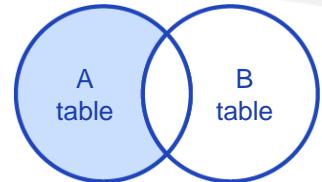
```
SELECT * FROM a FULL OUTER JOIN b ON a.sid = b.cid;
```

| A table |       |        |
|---------|-------|--------|
| sid     | count | p_name |
| 1       | 10    | hadoop |
| 3       | 20    | spark  |
| 5       | 30    | nosql  |
| 7       | 100   | spark  |
| 9       | 3     | NULL   |

| B  |     |         |
|----|-----|---------|
| id | cid | service |
| 1  | 1   | hdfs    |
| 2  | 1   | yarn    |
| 3  | 3   | kudu    |
| 4  | 3   | NULL    |
| 5  | 5   | NULL    |
| 7  | 7   | impala  |

| a.sid | a.node_count | a.platform_name | b.id | b.cid | b.service |
|-------|--------------|-----------------|------|-------|-----------|
| 1     | 10           | hadoop          | 1    | 1     | hdfs      |
| 1     | 10           | hadoop          | 2    | 1     | yarn      |
| 3     | 20           | spark           | 3    | 3     | kudu      |
| 3     | 20           | spark           | 4    | 3     |           |
| 5     | 50           | nosql           | 5    | 5     |           |
| 7     | 100          | spark           | 7    | 7     | impala    |
| 9     | 3            |                 | NUL  | NUL   | NULL      |

# Left Semi Join



- A left semi joins only return records from the table on the left

```
SELECT * FROM a LEFT SEMI JOIN b ON a.sid = b.cid;
```

| A table |       |        | B      |    |     |
|---------|-------|--------|--------|----|-----|
| sid     | count | p_name | table  | id | cid |
| 1       | 10    | hadoop |        | 1  | 1   |
| 3       | 20    | spark  |        | 2  | 1   |
| 5       | 30    | nosql  |        | 3  | 3   |
| 7       | 100   | spark  |        | 4  | 3   |
| 9       | 3     | NULL   |        | 5  | 5   |
|         |       |        |        | 7  | 7   |
|         |       |        | hdfs   |    |     |
|         |       |        | yarn   |    |     |
|         |       |        | kudu   |    |     |
|         |       |        | NULL   |    |     |
|         |       |        | NULL   |    |     |
|         |       |        | impala |    |     |

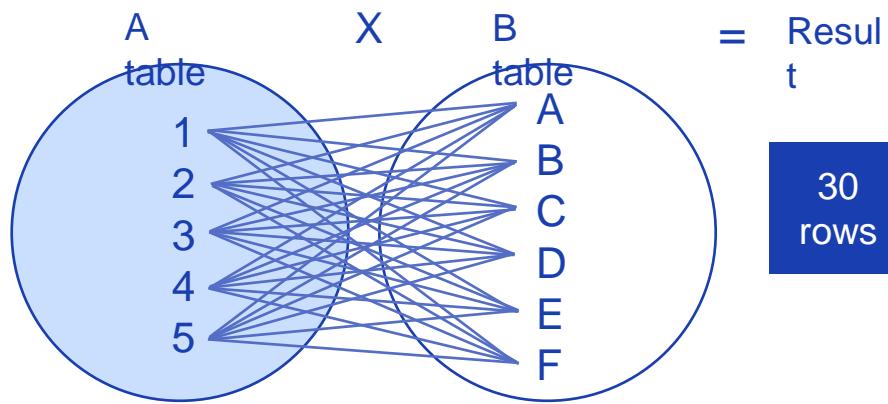
<Result>

| a.sid | a.node_count | a.platform_name |
|-------|--------------|-----------------|
| 1     | 10           | hadoop          |
| 3     | 20           | spark           |
| 5     | 50           | nosql           |
| 7     | 100          | spark           |

# Cross Join

- CROSS JOIN is a way of joining multiple tables in which all the rows or tuples from one table are paired with the rows and tuples from another table

```
SELECT * FROM a CROSS JOIN b;
```



| a.s_id | a.node_count | a.platform_name | b.id | b.cid | b.service |
|--------|--------------|-----------------|------|-------|-----------|
| 1      | 10           | hadoop          | 1    | 1     | hdfs      |
| 1      | 10           | hadoop          | 2    | 1     | yarn      |
| 1      | 10           | hadoop          | 3    | 3     | kudu      |
| 1      | 10           | hadoop          | 4    | 3     |           |
| 1      | 10           | hadoop          | 5    | 5     |           |
| 1      | 10           | hadoop          | 7    | 7     |           |
| 3      | 20           | spark           | 1    | 1     |           |
| 3      | 20           | spark           | 2    | 1     |           |
| 3      | 20           | spark           | 3    | 3     |           |
| 3      | 20           | spark           | 4    | 3     |           |
| 3      | 20           | spark           | 5    | 5     |           |
| 3      | 20           | spark           | 7    | 7     |           |
| 5      | 50           | nosql           | 1    | 1     |           |
| 5      | 50           | nosql           | 2    | 1     |           |
| 5      | 50           | nosql           | 3    | 3     |           |
| 5      | 50           | nosql           | 4    | 3     |           |
| 5      | 50           | nosql           | 5    | 5     |           |
| 5      | 50           | nosql           | 7    | 7     |           |
| 7      | 100          | spark           | 1    | 1     |           |
| 7      | 100          | spark           | 2    | 1     |           |
| 7      | 100          | spark           | 3    | 3     |           |
| 7      | 100          | spark           | 4    | 3     |           |
| 7      | 100          | spark           | 5    | 5     |           |
| 7      | 100          | spark           | 7    | 7     |           |
| 9      | 3            |                 | 1    | 1     |           |
| 9      | 3            |                 | 2    | 1     |           |
| 9      | 3            |                 | 3    | 3     |           |
| 9      | 3            |                 | 4    | 3     |           |
| 9      | 3            |                 | 5    | 5     |           |
| 9      | 3            |                 | 7    | 7     |           |

## Built-In Functions (1/7)

- Hive and Impala provide functions similar to the built-in functions provided by SQL.
  - Most are SQL-like, but some are specific to impala and HiveQL
  - Function names are not case-sensitive

- A list of functions and information about function function

```
hive> SHOW functions;
.....
--display the function list
hive> DESC function abs
```

```
0: jdbc:hive2://> desc function abs;
OK
```

| tab_name                                    |
|---------------------------------------------|
| abs(x) – returns the absolute value<br>of x |

1 row selected (0.038 seconds)

## Built-In Functions (2/7)

- Mathematical functions
  - Perform numeric calculations – round(a,b), ceil(a), abs(a), rand(), sqrt(a), floor(a)
- Date functions
  - Date-related functions are mainly used to change the output format of a date – current\_timestamp(), unix\_timestamp(a), to\_date(a)
- String functions
  - There are many different types of functions that manipulate strings - concat(a, b), concat\_ws(a, array,<string>), substring(a, b, c) , upper(a), lower(a), trim(a)

```
0: jdbc:hive2://> select concat_ws('@', 'student', 'samsung.com') as e_mail;
```

```
OK
```

|        |
|--------|
| e_mail |
|--------|

|                     |
|---------------------|
| student@samsung.com |
|---------------------|

```
1 row selected (0.143 seconds)
```

```
0: jdbc:hive2://> select upper('samsung') as upper;
```

```
OK
```

|       |
|-------|
| upper |
|-------|

|         |
|---------|
| SAMSUNG |
|---------|

```
1 row selected (0.167 seconds)
```

```
0: jdbc:hive2://> select trim(' s t u d e n t ') as trim;
```

```
OK
```

|      |
|------|
| trim |
|------|

|               |
|---------------|
| s t u d e n t |
|---------------|

```
1 row selected (0.141 seconds)
```

## Built-In Functions (3/7)

- Cast function
  - Use cast to change the type of a given value - `cast(expr as <type>)`
- Condition function
  - Conditional functions can be used to change the output of a value based on a condition – `isnull()`, `coalesce()`, `if()`

```
0: jdbc:hive2://> select if(10 > 1, 'a',
'b');
OK

_c0
a

1 row selected (0.139 seconds)
0: jdbc:hive2://> select if(10 < 1, 'a',
'b');
OK

_c0
b

1 row selected (0.143 seconds)
```

True: if the conditional statement is true  
False: if condition is false

Return the first non-null value in the order,

```
0: jdbc:hive2://> select coalesce(null,
'2');
OK

_c0
1

1 row selected (0.53 seconds)
0: jdbc:hive2://> select coalesce(null, null,
'2');
OK

_c0
2


```

```
1 row selected (0.206 seconds)
0: jdbc:hive2://> select coalesce(10, '1',
'2');

_c0
10


```

## Built-In Operators (4/7)

- Relational Operators

| Operators | Operands            | Operators     | Operands            |
|-----------|---------------------|---------------|---------------------|
| A = B     | all primitive types | A IS NULL     | all types           |
| A != B    | all primitive types | A IS NOT NULL | all types           |
| A < B     | all primitive types | A <= B        | all primitive types |
| A > B     | all primitive types | A >= B        | all primitive types |

## Built-In Operators (5/7)

- Arithmetic Operators

| Operators | Operands          | Descriptions                                       |
|-----------|-------------------|----------------------------------------------------|
| A+B       | All numeric types | Gives the result of adding A and B                 |
| A-B       | All numeric types | Gives the result of subtracting B from A           |
| A*B       | All numeric types | Gives the result of multiplying A and B            |
| A/B       | All numeric types | Gives the result of dividing B from A              |
| A%B       | All numeric types | Gives the remainder resulting of dividing B from A |
| A&B       | All numeric types | Gives the result of bitwise AND of A and B         |
| A   B     | All numeric types | Gives the result of bitwise OR of A and B          |
| ~A        | All numeric types | Gives the result of bitwise NOT of A               |

## Built-In Operators (6/7)

- Logical Operators

| Operators | Operands | Descriptions                                            |
|-----------|----------|---------------------------------------------------------|
| A AND B   | Boolean  | TRUE if both A and B are TRUE, otherwise FALSE          |
| A && B    | Boolean  | Same as A AND B                                         |
| A OR B    | Boolean  | TRUE if either A or B or both are TRUE, otherwise FALSE |
| A    B    | Boolean  | Same as A OR B                                          |

## Built-In Operators (7/7)

- Complex type Operators

| Operators | Operands                            | Descriptions                                                                                                                                                                               |
|-----------|-------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| A[n]      | A is an Array and n is an int       | returns the nth element in the array A. The first element has index 0, for example, if A is an array comprising of ['spark', 'hadoop'] then A[0] returns 'spark' and A[1] returns 'hadoop' |
| M[key]    | M is a Map<K, V> and key has type K | returns the value corresponding to the key in the map for example, if M is a map comprising of {'h' -> 'hadoop', 's' -> 'spark', 'n' -> 'nosql'} then M['n'] returns 'nosql'               |
| S.x       | S is a struct                       | returns the x field of S, for example, for struct hadoop {int har, string doo} hadoop.har returns the integer stored in the har field of the struct                                        |

## Using MAP Columns (1/2)

- MAP keys must all be one data type, and values must all be one data type
  - `MAP<KEY-TYPE.VALUE-TYPE>`

```
CREATE TABLE product_map (
 id int,
 platform string,
 detail_info map<string, string>
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
COLLECTION ITEMS TERMINATED BY '|'
MAP KEYS TERMINATED BY ':';
DESC product_map;
```

| col_name    | data_type           | comment |
|-------------|---------------------|---------|
| id          | int                 |         |
| platform    | string              |         |
| detail_info | map<string, string> |         |

## Using MAP Columns (2/2)

- MAPs are similar to ARRAYS, except that instead of using numbers to access the values, we use labels

```
SELECT id, platform, detail_info['node'] AS number_of_node, detail_info['location'] AS location
FROM product_map;
```

| id | platform | number_of_node | location |
|----|----------|----------------|----------|
| 1  | hadoop   | 10             | Seoul    |
| 5  | spark    | 20             | Busan    |
| 7  | nosql    | 30             | Jeju     |

Result

select ALL \*

| product_map.id | product_map.platform | product_map.detail_info                              |
|----------------|----------------------|------------------------------------------------------|
| 1              | hadoop               | {"node": "10", "name": "Hdfs ", "location": "Seoul"} |
| 5              | spark                | {"node": "20", "name": "Kafka", "location": "Busan"} |
| 7              | nosql                | {"node": "30", "name": "Hbase", "location": "Jeju"}  |

# Using STRUCT Columns (1/2)

- A STRUCT stores a fixed number of named fields
  - Each field can have a different data type

```
CREATE TABLE product_complex (
 id integer,
 name string
 detail STRUCT<node:int, name:string, location:string>
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
COLLECTION ITEMS TERMINATED BY '::';
DESC product_complex;
```

| col_name    | data_type                                    | comment |
|-------------|----------------------------------------------|---------|
| id          | int                                          |         |
| platform    | string                                       |         |
| detail_info | struct<node:int,name:string,location:string> |         |

## Using STRUCT Columns (2/2)

- The Struct type have names and types

```
SELECT id, platform, detail_info.name, detail_info.node FROM product_complex;
```

| id | platform | name  | node |
|----|----------|-------|------|
| 1  | hadoop   | Hdfs  | 10   |
| 5  | spark    | Kafka | 20   |
| 7  | nosql    | HBase | 30   |

Result

select \*

| product_complex.id | product_complex.platform | product_complex.detail_info                          |
|--------------------|--------------------------|------------------------------------------------------|
| 1                  | hadoop                   | {"node": "10", "name": "Hdfs ", "location": "Seoul"} |
| 5                  | spark                    | {"node": "20", "name": "Kafka", "location": "Busan"} |
| 7                  | nosql                    | {"node": "30", "name": "Hbase", "location": "Jeju"}  |

# Aggregation and Windowing

- Aggregate and Windowing(analytics functions) both use values over multiple rows
- Aggregation function
  - Aggregate Functions in Hive can be used with or without GROUP BY functions
  - Hive Aggregate Functions are the most used built-in functions
- Windowing
  - Windowing allows features to create a window on the set of data in order to operate aggregation like COUNT, AVG, MIN, MAX
  - Other analytical functions such as LEAD, LAG, FIRST\_VALUE, and LAST\_VALUE

# Built-in Aggregation Functions

- Function list
  - COUNT (\*): Counts all rows
  - COUNT(column): Counts all rows where field is not NULL
  - COUNT(distinct column): Counts all rows where field is unique and not NULL
  - MAX(): Returns the largest value
  - MIN(): Returns the smallest value
  - SUM(): Adds all supplied values and returns result
  - AVG(): Returns the average of all supplied values

# Windows

- A window defines a set of rows in a table
- OVER (window-specification) specifies the windows over which to apply an aggregation or windowing function

PATRION BY  
service



| id | name      | service   | cost | rank |
|----|-----------|-----------|------|------|
| 10 | HUE       | HADOOP    | 3200 | 1    |
| 9  | IMPALA    | HADOOP    | 5400 | 1    |
| 8  | KUDU      | HADOOP    | 5000 | 1    |
| 2  | YARN      | HADOOP    | 1500 | 1    |
| 1  | HDFS      | HADOOP    | 1800 | 1    |
| 11 | MONGODB   | NoSQL     | 5000 | 1    |
| 4  | Cassandra | NoSQL     | 1300 | 1    |
| 3  | HBASE     | NoSQL     | 2000 | 1    |
| 7  | AWS       | PUB-CLOUD | 8000 | 1    |
| 6  | AZURE     | PUB-CLOUD | 5000 | 1    |
| 12 | KAFKA     | SPARK     | 8000 | 1    |
| 5  | SPARKSQL  | SPARK     | 1500 | 1    |

The diagram illustrates the concept of windows in a relational database. A vertical dashed line on the right side of the table separates the data into four distinct windows, each labeled "Window W". The windows are defined by the "service" column: HADOOP (rows 1-5), NoSQL (rows 6-9), PUB-CLOUD (rows 10-11), and SPARK (rows 12-13). Each window contains all the columns of the table.

# Windowing Functions

- The syntax of the query with windows:

```
SELECT <columns_name>, <aggregate>(column_name) OVER (<>windowing specification>)
FROM <table_name>;
```

- column\_name – column name of the table
- Aggregate – Any aggregate function(s) like COUNT, AVG, MIN, MAX
- Windowing specification – It includes following:
  - PARTITION BY – Takes a column(s) of the table as a reference
  - ORDER BY – Specified the Order of column(s) either Ascending or Descending
  - Frame – Specified the boundary of the frame by start and end value. The boundary either be a type of RANGE or ROW followed by PRECEDING, FOLLOWING and any value

# RANK Functions (1/3)

- ROW\_NUMBER
  - One of the analytics function, assign the unique number for each row based on the column value that used in the OVER
- RANK
  - A built in analytic function which is used to rank a record within a group of rows
- DENSE\_RANK
  - Rank of current value within the window (with consecutive rankings)

## RANK Functions (2/3)

- RANK

```
SELECT name, service, cost,
RANK() OVER(ORDER BY cost DESC) AS costRank
FROM Projects;
```

| name      | company   | cost | costrank |
|-----------|-----------|------|----------|
| KAFKA     | SPARK     | 8000 | 1        |
| AWS       | PUB-CLOUD | 8000 | 1        |
| IMPALA    | HADOOP    | 5400 | 3        |
| MONGODB   | NoSQL     | 5000 | 4        |
| KUDU      | HADOOP    | 5000 | 4        |
| AZURE     | PUB-CLOUD | 5000 | 4        |
| HUE       | HADOOP    | 3200 | 7        |
| HBASE     | NoSQL     | 2000 | 8        |
| HDFS      | HADOOP    | 1800 | 9        |
| SPARKSQL  | SPARK     | 1500 | 10       |
| YARN      | HADOOP    | 1500 | 10       |
| Cassandra | NoSQL     | 1300 | 12       |

## RANK Functions (3/3)

- ROW\_NUMBER vs DENSE\_RANK vs RANK

```
SELECT name, service, cost, RANK() OVER(ORDER BY cost DESC) AS rank,
DENSE_RANK() OVER(ORDER BY cost DESC) AS dense_rank,
ROW_NUMBER() OVER(ORDER BY cost DESC) AS row_number FROM projects;
```

| name      | company   | cost | rank | dense_rank | row_number |
|-----------|-----------|------|------|------------|------------|
| KAFKA     | SPARK     | 8000 | 1    | 1          | 1          |
| AWS       | PUB-CLOUD | 8000 | 1    | 1          | 2          |
| IMPALA    | HADOOP    | 5400 | 3    | 2          | 3          |
| MONGODB   | NoSQL     | 5000 | 4    | 3          | 4          |
| KUDU      | HADOOP    | 5000 | 4    | 3          | 5          |
| AZURE     | PUB-CLOUD | 5000 | 4    | 3          | 6          |
| HUE       | HADOOP    | 3200 | 7    | 4          | 7          |
| HBASE     | NoSQL     | 2000 | 8    | 5          | 8          |
| HDFS      | HADOOP    | 1800 | 9    | 6          | 9          |
| SPARKSQL  | SPARK     | 1500 | 10   | 7          | 10         |
| YARN      | HADOOP    | 1500 | 10   | 7          | 11         |
| Cassandra | NoSQL     | 1300 | 12   | 8          | 12         |

# RANK and ROW\_NUMBER Functions

- What is the ranking of each projects by cost, within each service?

```
SELECT name, service, cost, rank() OVER(PARTITION BY service ORDER BY cost) AS Rank,
row_number() OVER(PARTITION BY service ORDER BY cost) AS row_number
FROM projects;
```

| name      | company   | cost | rank | row_number |
|-----------|-----------|------|------|------------|
| YARN      | HADOOP    | 1500 | 1    | 1          |
| HUE       | HADOOP    | 3200 | 2    | 2          |
| HDFS      | HADOOP    | 3200 | 2    | 3          |
| KUDU      | HADOOP    | 5000 | 4    | 4          |
| IMPALA    | HADOOP    | 5400 | 5    | 5          |
| Cassandra | NoSQL     | 1300 | 1    | 1          |
| HBASE     | NoSQL     | 2000 | 2    | 2          |
| MONGODB   | NoSQL     | 5000 | 3    | 3          |
| AWS       | PUB-CLOUD | 5000 | 1    | 1          |
| AZURE     | PUB-CLOUD | 5000 | 1    | 2          |
| KAFKA     | SPARK     | 8000 | 1    | 1          |
| SPARKSQL  | SPARK     | 8000 | 1    | 2          |

# Other Windowing Functions

- LEAD
  - The number of rows to lead can optionally be specified. If the number of rows to lead is not specified, the lead is one row
  - Returns null when the lead for the current row extends beyond the end of the window
- LAG
  - The number of rows to lag can optionally be specified. If the number of rows to lag is not specified, the lag is one row
  - Returns null when the lag for the current row extends before the beginning of the window
- FIRST\_VALUE
  - This takes at most two parameters. The first parameter is the column for which you want the first value, the second (optional) parameter must be a boolean which is false by default. If set to true it skips null values
- LAST\_VALUE
  - This takes at most two parameters. The first parameter is the column for which you want the last value, the second (optional) parameter must be a boolean which is false by default. If set to true it skips null values

# Compare with Apache Hive and Impala

- Comparing Hive and Impala to Relational DB

[Discussion]

| Features       | Hive      | Impala     | RDBMS     |
|----------------|-----------|------------|-----------|
| Query language | Hive QL   | Impala SQL | SQL(full) |
| Update         | No*       | No*        | Yes       |
| Delete         | No*       | No*        | Yes       |
| Transactions   | No*       | No*        | Yes       |
| Index support  | Limited   | Limited    | Extensive |
| Latency        | High      | Low        | Very low  |
| Data size      | Petabytes | Petabytes  | Terabytes |
| Storage cost   | Very Low  | Low        | Very High |

# Review Questions

[Quiz]

Quiz 1

- What are some reasons you would not replace a typical relational database with Hive?

Quiz 2

- What are two benefits that Hive and Hadoop have over typical data warehousing systems?

Quiz 3

- When you query a table in Hive, where does the data that is being queried reside?

Quiz 4

- What is the difference between an external and managed table?

# Regular Expression

[Discussion]

A regular expression(regex) matches a pattern in text.

One thing that is notable for people who know regex from a language other than Java(e.g. perl or awk) is that the backslash is a special character in Java and must be escaped by another backslash in a regex.

## Case

- look
- \d
- \d{5}
- \d\s\w+
- \w{5,9}
- .?\.
- .\*?\.

## [Lab7]

### Handling partitioned table for performance



## [Lab8] Working with complex Data type



# [Lab9]

## Complex Queries



Chapter 6

# **Big Data Processing with Apache Spark**

Big Data Course

# Chapter Description

---

## Objectives:

- ✓ Apache Spark is a general purpose data processing framework that incorporates different modules for processing different types of data. It is currently one of the most active open source projects with many enterprises adopting Spark for their use cases.
  - We will learn how to use the Spark Core API layer to process, cleanse and query unstructured data and semi-structured data. We will also learn how to convert such data to structured data.
  - Spark's Dataframe API is a higher-level API build on top of the Core API and is optimized to process structured data. We shall explore processing data using the API functionality as well as through passing SQL syntax queries
  - The Structured Streaming API and DStream API is used to process structured streaming data and unstructured streaming data, respectively. We will learn how to use the API with emphasis on connecting to Kafka topics for the streaming data source
  - Finally, we will explore Spark's architecture and how it affects performance tuning

## Contents of Chapter:

1. Unstructured Data Processing
2. Structured Data Processing
3. Streaming Data Processing

Unit 1

# Unstructured Data Processing

Big Data Processing with Apache Spark

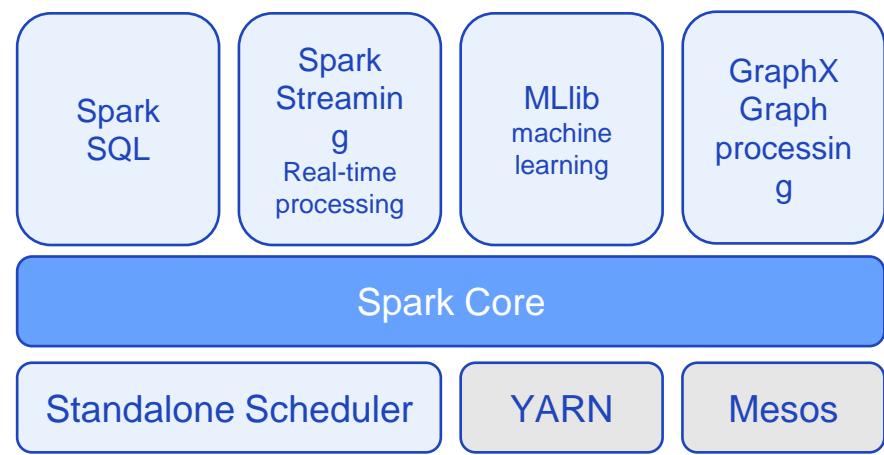
Unit 1.

# Unstructured Data Processing

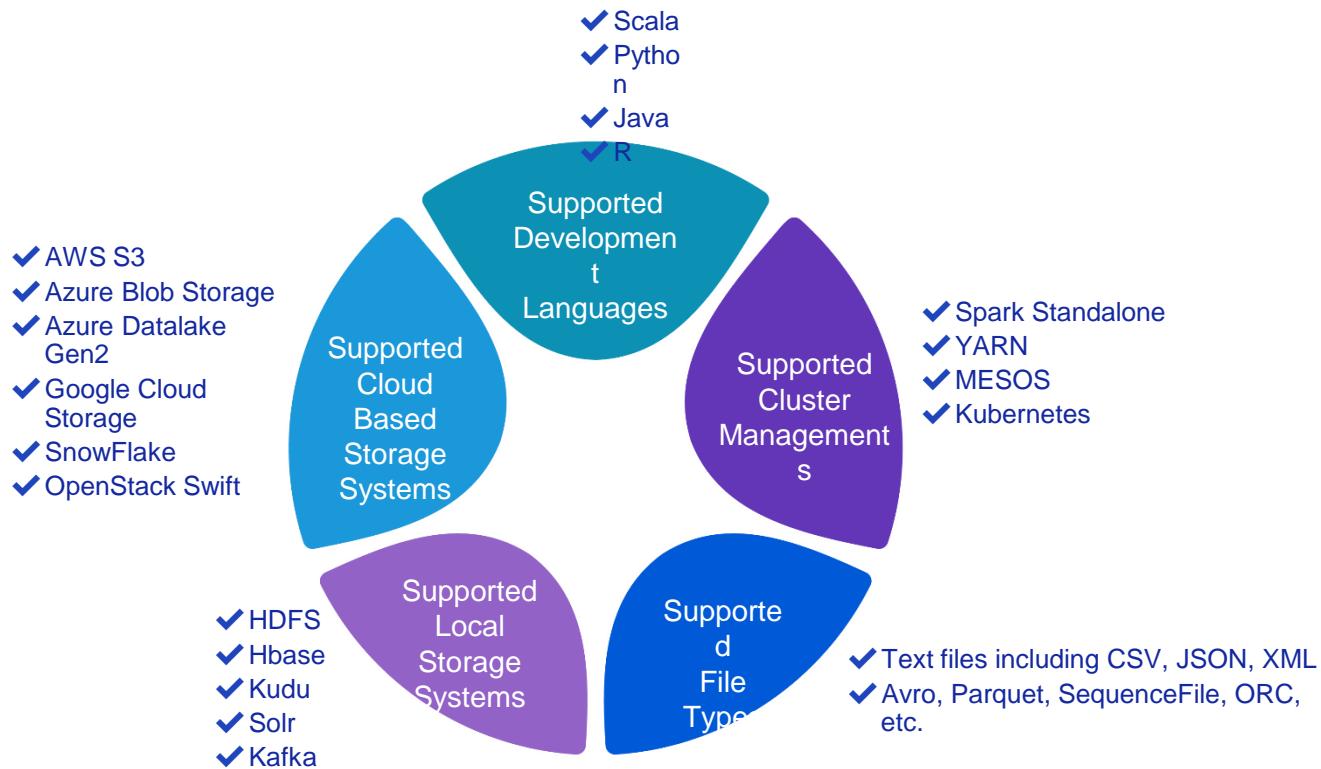
- | 1.1. Introduction to Apache Spark
- | 1.2. Python Basics
- | 1.3. Data Transformation with Core API
- | 1.4. Working with Pair RDDs

# What is Apache Spark?

- Apache Spark is a unified analytics engine for large-scale data processing
  - A set of high-level APIs in Scala, Python, Java and R
- General-purpose data processing engine that provides
  - Interactive processing
  - Extremely fast batch processing
  - Real-time stream processing
  - Distributed machine learning modeling and inference engine
  - Graph processing
- In-memory distributed computation engine
- Originally developed by AMP Lab at the University of California, Berkeley

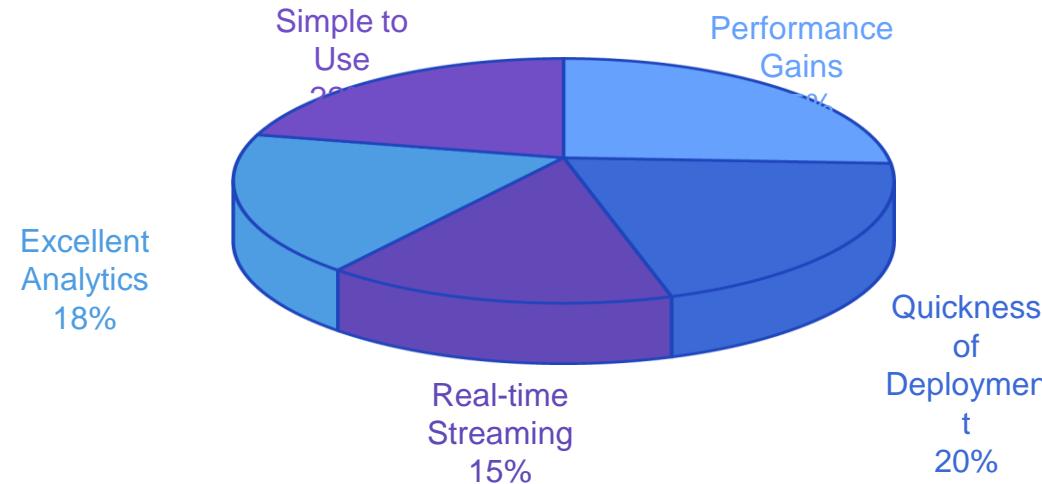


# Flexible Architecture



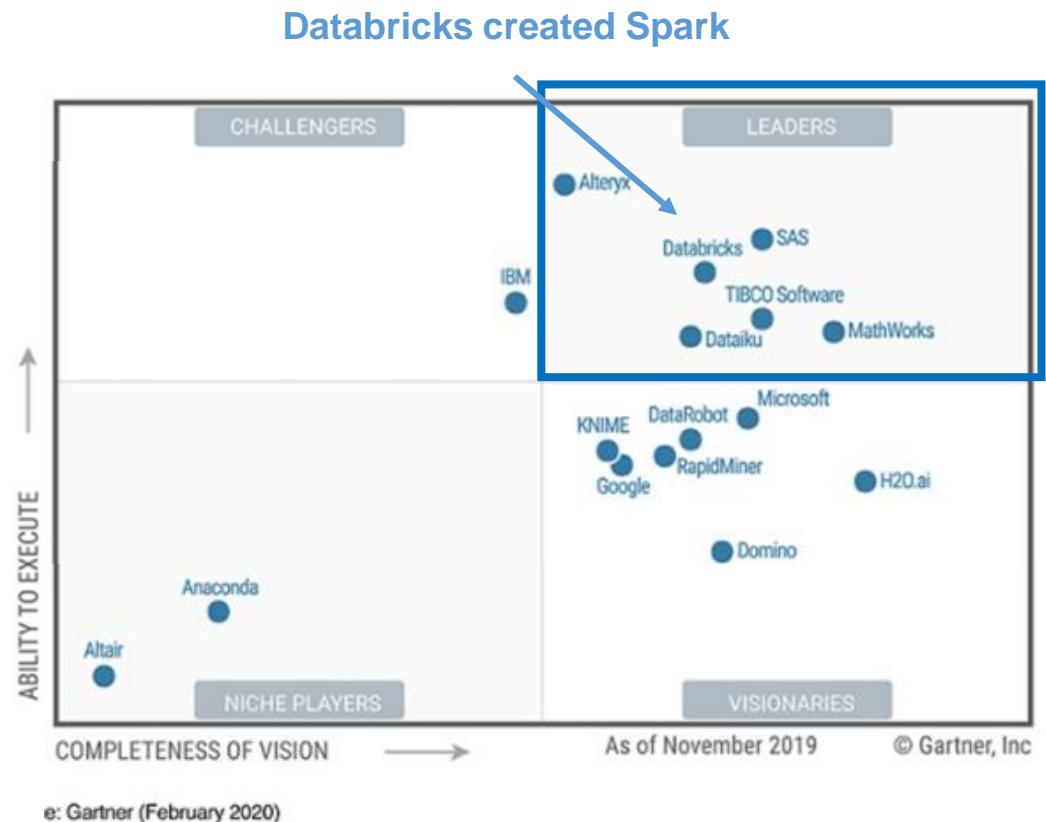
# Why is Apache Spark Popular?

- Apache Spark is the next generation processing engine within the Hadoop Ecosystem
- It is widely used for many different tasks:
  - ETL and Data Cleansing
  - SQL batch jobs over large data sets
  - Processing streaming data from sensors, IoT, financial systems, etc.
  - Machine learning and inference



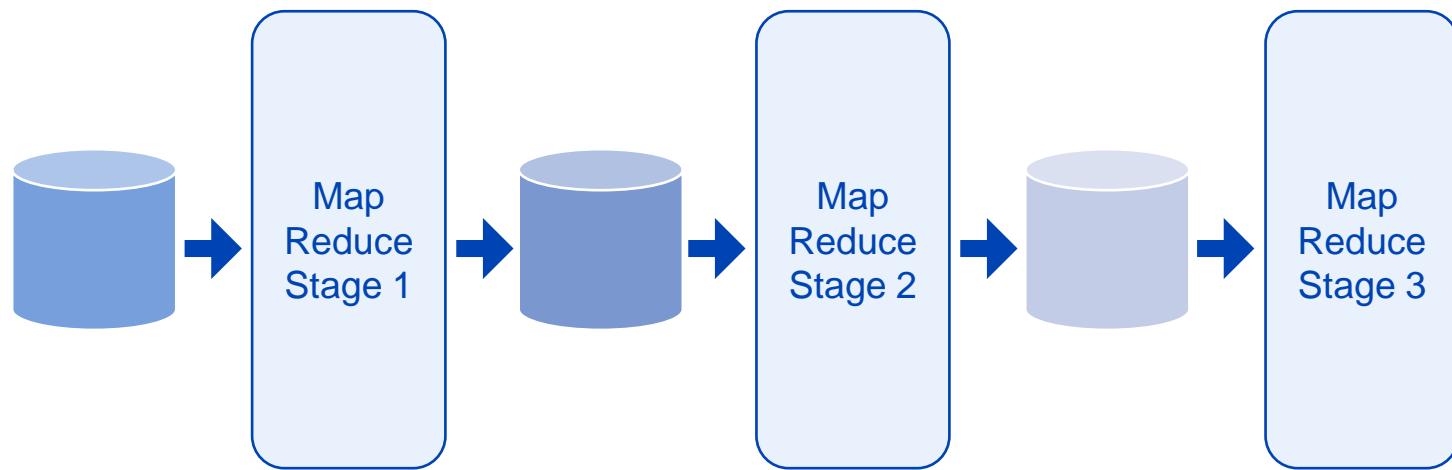
# Gartner Magic Quadrant

- Gartner Magic Quadrant is a popular source for reviewing technology trends
  - They review products based on completeness of vision and ability to execute
  - Products in the top right quadrant are the most complete and well executed products
- Apache Spark was given high marks for its ability to unify data and machine learning work loads all within a single platform
  - End-to-end integration of data exploration, data pre-processing, machine learning to machine learning inference



# The Problem with MapReduce

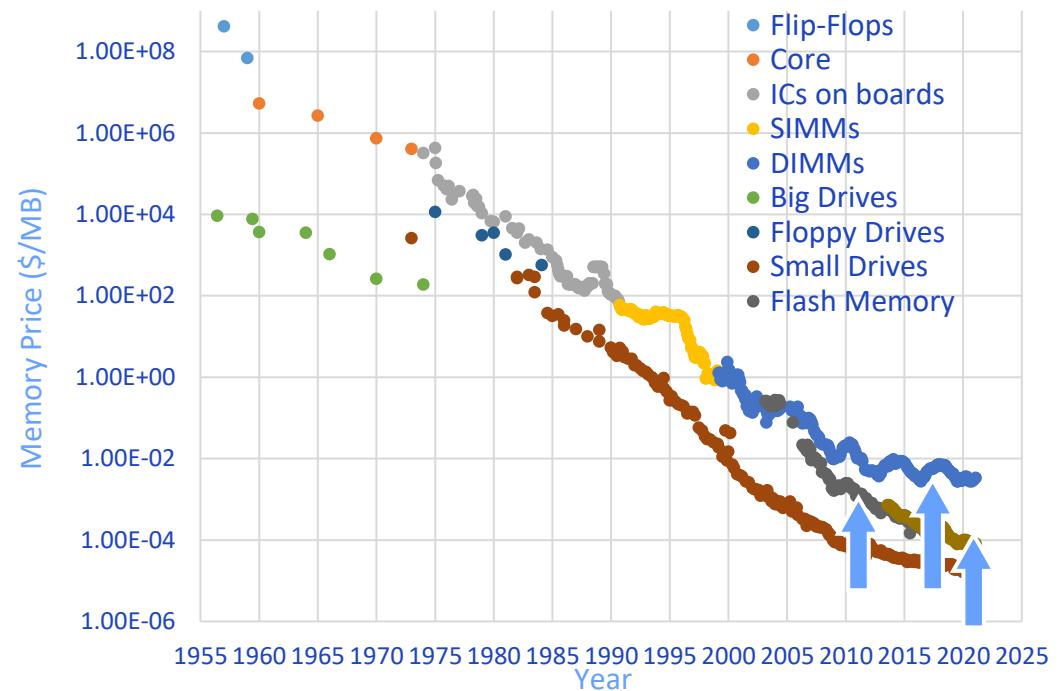
- A Map Reduce job typically consists of multiple Map and Reduce cycles
- Each Map cycle requires a read from HDFS and a write to local disk
- Each Reduce cycle requires a read from local disk and write to HDFS
- Disk I/O is very EXPENSIVE



# Evolving Hardware: Cost and Performance

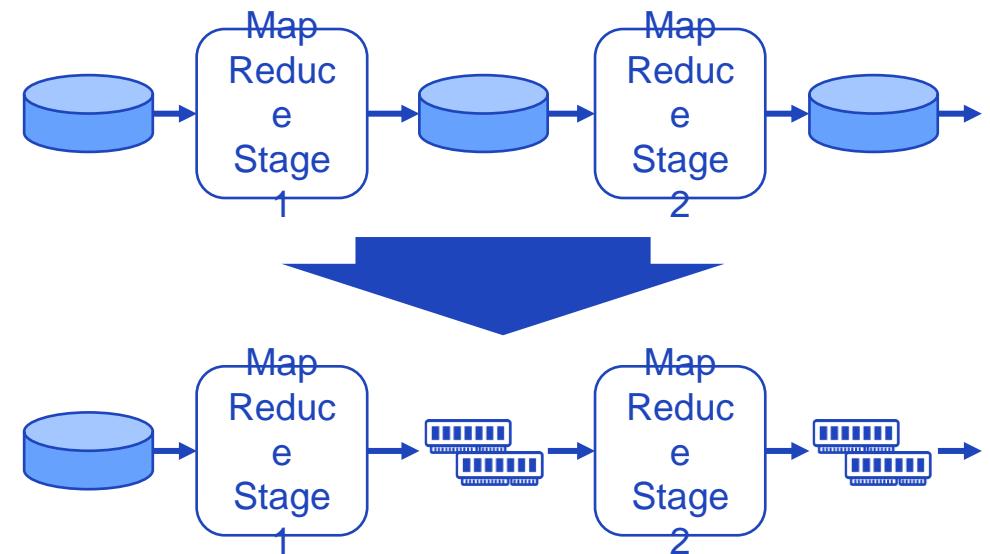
- Hadoop 0.14.1 introduced in 2007
  - ▶ Both HDFS and MapReduce, heavily dependent on disk based storage

Historical Cost: Computer Memory and Storage



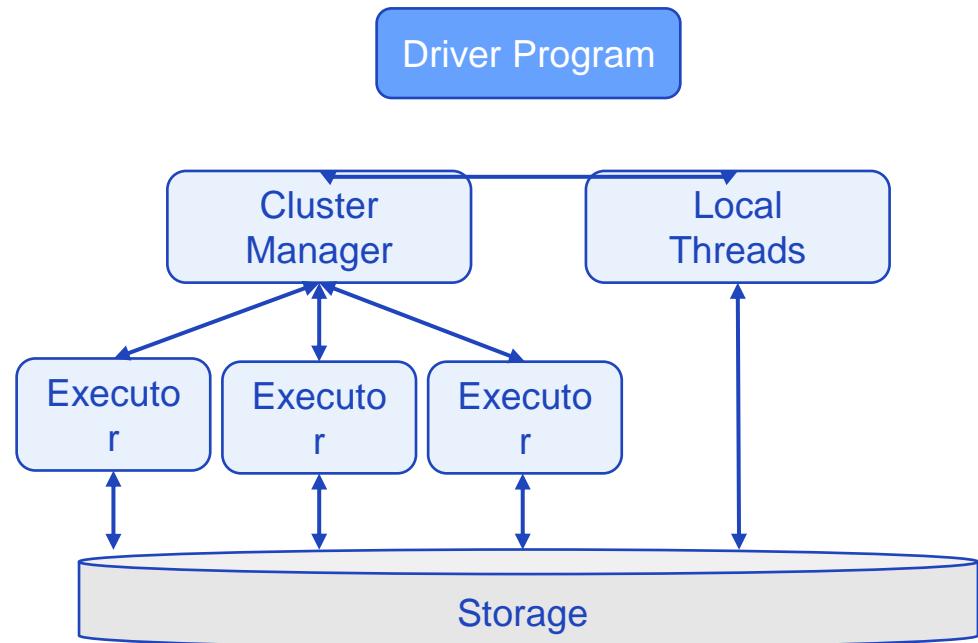
# In-Memory Distributed Computation

- Incur disk I/O cost only at the beginning of the computation to load the data into memory
- Use memory based storage during computation cycle
- Typically **100+ times** faster compared to MapReduce



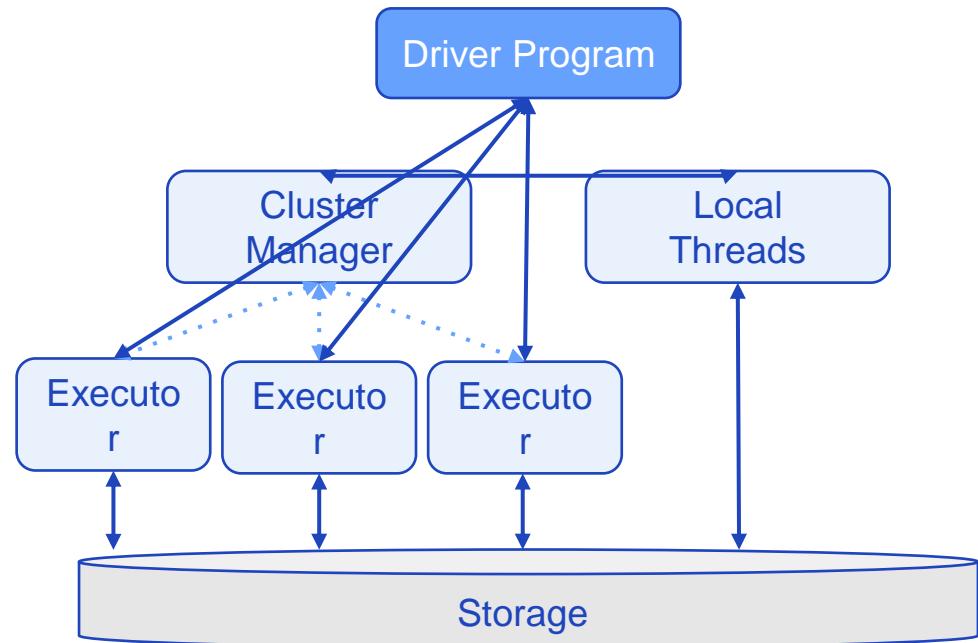
# Spark Driver and Executors (1/2)

- Apache Spark follows a master-slave type of architecture
- The Driver program acts as the master and coordinates creating or removing Executors
- The Driver partitions the data and passes each partition to an Executor for parallel distributed processing
- Each Executor performs the actual computation on its partition of the data
- Executors communicate back results to the Driver



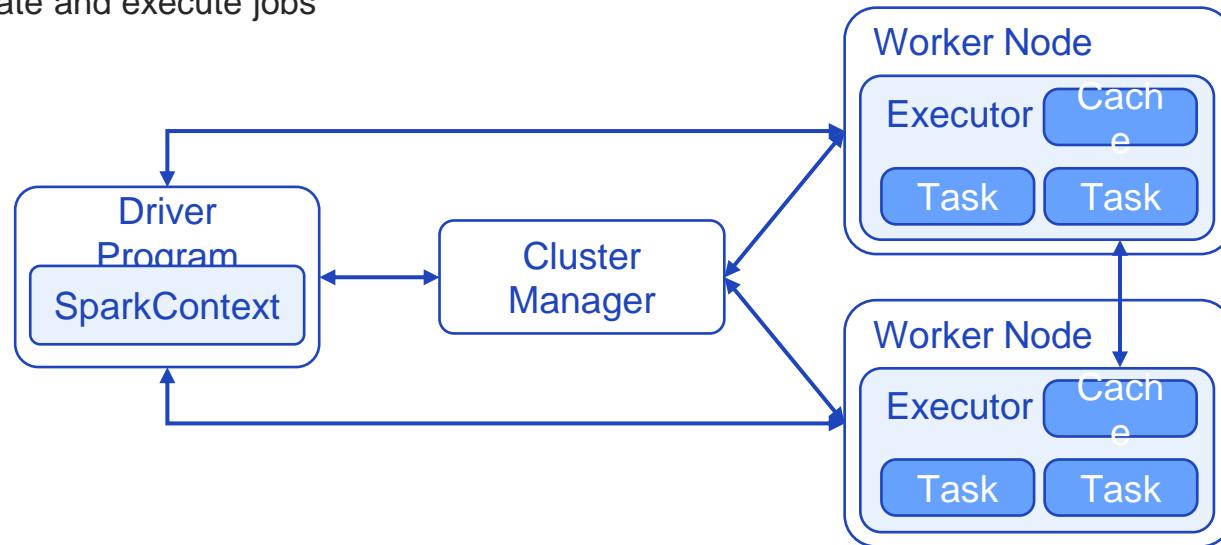
## Spark Driver and Executors (2/2)

- Apache Spark follows a master-slave type of architecture
- The Driver program acts as the master and coordinates creating or removing Executors
- The Driver partitions the data and passes each partition to an Executor for parallel distributed processing
- Each Executor performs the actual computation on its partition of the data
- Executors communicate back results to the Driver



# SparkContext Entry Point Object

- A `SparkContext` object is the main entry point for Spark functionality.
  - It represents the connection to a Spark cluster
  - The Driver program uses `SparkContext` to establish communication with the cluster and resource managers in order to coordinate and execute jobs



# Other Entry Point Objects

- For Spark versions prior to Spark 2.x, there are several additional entry point objects
- SQLContext
  - Entry point to Spark SQL and provides basic SQL functionality
  - Perform SQL like operations over Datasets and Dataframes
- HiveContext
  - In addition to basic SQL functionality, allows communication with Apache Hive
  - Use in lieu of SQLContext when Hive functionality is necessary
- Spark Streaming Context
  - The main entry point for all streaming functionality
  - Uses by Spark Dstream API
- Post Spark 2.x, SparkSession provides a unified entry point object
  - Use SparkSession to create SparkContext
  - Use SparkSession directly for SQL/Hive integration

# Apache Spark Shell

- Apache Spark provides a shell environment for easy interactive development
- The shell provides a REPL (Read Evaluate Print Loop) environment where developers can get results immediately returned without having to compile their code
- Automatically creates the core entry objects such as Spark Context and Spark Session
- Only available for Scala and Python
  - not available for Java and R
- Call with spark-shell for Scala
- Call with pyspark for Python

```
Welcome to
 /_/_/_/_/_/_/_/_

 / \ . \ _ , / / / \ \ \ \
 /_/
version 3.1.2

Using Python version 3.8.11 (default, Aug 10 2021 21:35:17)
Spark context Web UI available at http://10.20.70.210:4040
Spark context available as 'sc' (master = local[*], app id = local-1629266540018).
SparkSession available as 'spark'.
>>> sc
<SparkContext master=local[*] appName=PySparkShell>
>>> spark
<pyspark.sql.session.SparkSession object at 0x7fd9e46bc250>
>>>
```

# Spark Shell on Jupyter Notebook

- It Jupiter possible to integrate Jupyter Notebook with Pyspark
- After installing Jupyter Notebook, add the following to your Linux shell startup

For bash this will be bashrc

```
export PYSPARK_DRIVER_PYTHON="jupyter"
export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
```

- After setting the two environmental variables, calling pyspark will open in Jupyter instead

Unit 1.

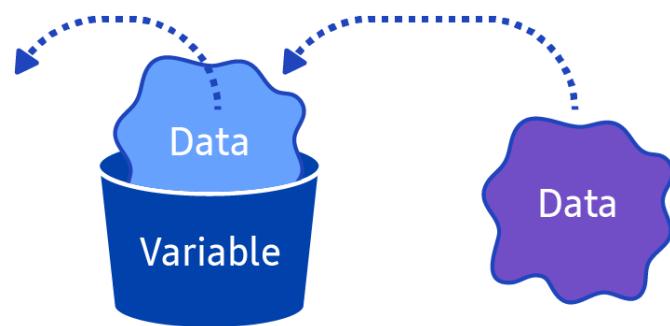
# Unstructured Data Processing

- | 1.1. Introduction to Apache Spark
- | 1.2. Python Basics
- | 1.3. Data Transformation with Core API
- | 1.4. Working with Pair RDDs

# Creating Variables

- Variables are used to hold references to some object
  - We can just simply think of it as holding a value for us
- Python variables do not have to be declared
- Variable types are discovered implicitly
- ~~Create a variable by simply assigning a value to it~~  

```
x = 1
```



## Naming Rules (1/2)

- Names are case sensitive

```
Student pylon
```

- They can contain:

- Letters
- Numbers

```
MyVariable myvariable_1
myvariable_student
```

## Naming Rules (2/2)

- Can not start with a number

```
2myvariable # ERROR
```

- Reserved words

- del from not while as elif global or with assert else if pass yield break except import print class exec in raise continue finally is return def for lambda try

# Integer Data Type in Python

- Integers
  - Called “int” in Python
  - In Python 3, memory is the only limit as to how large an integer variable can be
  - You can change the base to

| Prefix                           | Interpretation | Base |
|----------------------------------|----------------|------|
| 0b (zero + lowercase letter 'b') | Binary         | 2    |
| 0B (zero + uppercase letter 'B') |                |      |
| 0o (zero + lowercase letter 'o') | Octal          | 8    |
| 0O (zero + uppercase letter 'O') |                |      |
| 0o (zero + lowercase letter 'o') | Hexadecimal    | 16   |
| 0O (zero + uppercase letter 'O') |                |      |

# Floating-point Data Type in Python

- Floating-point numbers
  - Called “float” in Python
  - Specify with a decimal point

```
my_pi = 3.14
my_float = .3
```

- Optionally, use character e or E followed by a positive or negative integer to specify scientific notation

```
my_var = 3.7e8
my_float = 2.17E-3
```

# Basic Data Types in Python

- String Data Type
  - Called "str" in Python
  - Sequence of character data
  - All characters between opening delimiter and matching closing delimiter are part of the string
  - Delimiters can be double quote " or single quote '
  - The length of the string is only restricted by

```
print("I am a string")
print('I am a string')
print("I'm a string")
print('I'm not a valid string')
```

# Boolean Data Type in Python

- Boolean Data Type
  - Called "bool" in Python
  - Boolean types may be either True or False

```
is_male = False
is_student = True
```

# Collection Data Types in Python (1/3)

- Collection type functionality description

| Data types | Descriptions                                                    | Examples                                |
|------------|-----------------------------------------------------------------|-----------------------------------------|
| List       | Sequence of different data type objects which can be updated    | [1, 'abc', [1,2]]                       |
| Tuple      | Sequence of different data type objects which cannot be updated | ('abc', 142)                            |
| Set        | Unordered collection of distinct objects                        | {1,5,2}                                 |
| Dictionary | Consists of multiple Key-Value pairs with unique keys           | {'fname' : 'Student', 'lname' : 'Good'} |

# Collection Data Types in Python (2/3)

- The basic collection types are distinguished by the following characteristics
  - Ordered – Is the list sortable?
  - Mutable – Can you modify the values?
  - Allow Duplicates – Can there be duplicated values?

| Name       | Ordered | Mutable | Allow Duplicate |
|------------|---------|---------|-----------------|
| List       | Yes     | Yes     | Yes             |
| Tuple      | Yes     | No      | Yes             |
| Set        | No      | Both    | No              |
| Dictionary | No      | Yes     | No              |

# Collection Data Types in Python (3/3)

- Create collections with comma separated items with relevant operator:
  - List uses square brackets [ ]
  - Tuples use parenthesis ( )
  - Sets use curly braces { }

```
my_list = [1.2, 5, "hello"]
my_tuple = (1.2, 5, "hello")
my_set = {1, 2, 3, 4}
my_dict = {1: "one", 2: "two", 3: "three", 4: "four"}
print(my_list)
print(my_tuple)
print(my_set)
print(my_dict)
```

Input

```
[1.2, 5, 'hello']
(1.2, 5, 'hello')
{1, 2, 3, 4}
{1: 'one', 2: 'two', 3:
'three', 4: 'four'}
```

Output

# Creating Strings

- Create Strings with either "quote" or 'single quote'

```
string1 = "I am a string"
string2 = 'I am a string also'
```

- If you need to put a single quote in the sentence,  
~~use a quote to create the sentence and vice versa~~

```
string3 = "I don't like this"
string4 = 'I am the "best" string'
```

# Escaping Characters

- Certain characters have special meaning
  - For example, if we started a string with a quote, a matching quote would end the string
  - What if we don't want that to happen?

```
string5 = "You can use a \" or \' to create strings in Python"
```

- Sometimes you want to give special meaning

```
string6 = "I am \t special \n really special"
```

# Slicing Strings

- You can access just a "slice" or part of a string
- Strings are considered an array or list of characters and have indexes

| S  | T  | R  | I  | N  | G  |
|----|----|----|----|----|----|
| 0  | 1  | 2  | 3  | 4  | 5  |
| -6 | -5 | -4 | -3 | -2 | -1 |

- How to use start\_index and end\_index (not inclusive) to slice the string

```
my_string = "my_python"
print(my_string[4])
print(my_string[0:6])
print(my_string[1:-1])
print(my_string[:-2])
print(my_string[2:])
print(my_string[:])
```

Input

Output

```
y
my_pyt
y_pytho
my_pyth
_python
my_python
```

# Concatenate and Multiply Strings

- Use + to concatenate strings

```
"Hello Student " + "Welcome to the Apache Spark Lessons"
```

- Multiply Strings with \*

```
border_unit = "*-*"
print(border_unit * 5)
```

- ```
student_name = "Good Student"  
"Hello " + student_name + " Welcome to the Apache Spark Class"
```

Useful String Methods (1/2)

- **lower()**, **upper()**
 - returns lowercase or uppercase of the string
- **len()**
 - return length of string
- **index(letter or "string")** or **find()**
 - return index of first occurrence of letter or string
- **replace("this", "that")**
 - replace this with that
- **strip()**
 - remove any white space in front and back
- **split("delimiter")**
 - parse paragraph by delimiter

Useful String Methods (2/2)

- String Methods Usage

```
txt = " Students learn pyspark "

print(txt.lower())
print(len(txt))
print(txt.index("e"))
print(txt.replace("learn", "teach"))
print(txt.strip())
print(txt.split(" "))
```

Input

```
students learn pyspark
26
6
Students teach pyspark
Students learn pyspark
[', ', 'Students', 'learn', 'pyspark', ', ']
```

Output

Basic Arithmetic Operators

- These are the basic mathematical operators in Python

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
%	Modulus	$x \% y$
//	Floor Division	$x // y$
**	Exponentiation	$x ** y$

Comparison Operators

- Use comparison operators on a wide variety of datatypes, including complex data types such as collections

Operator	Name	Example
<code>==</code>	Equal	<code>x == y</code>
<code>!=</code>	Not equal	<code>x != y</code>
<code>></code>	Greater than	<code>x > y</code>
<code><</code>	Less than	<code>x < y</code>
<code>>=</code>	Greater than or equal to	<code>x >= y</code>

Logical Operators

- Logical operators are used to combine basic conditions into complex multi conditions

Operator	Description	Example
and	Returns True if both statements are true	$x \geq 4 \text{ and } x \leq 8$
or	Returns True if one of the statements is true	$x > 100 \text{ or } x < 50$
not	Return the opposite of the evaluated result. Return True if false and False if true	<code>not (x > 100 or x < 50)</code>

Identity Operators

- Identity operators are different from equality comparisons
 - Equality comparison simply compare to equal "values"
 - Identity operators compare the actual object - equivalent and located in same memory location

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
Is not	Returns True if the variables are not the same object	x is not y

Membership Operator

- Used to test for membership in collection data types

Operator	Description	Example
in	Returns True if sequence specified is in the object compared	x in y
not in	Returns True if sequence specified is NOT in the object compared	x not in y

Augmented Assignment

- Augmented assignments are shortcut operators when the operator is applied to a variable and assigned to that variable, over-writing it

Operator	Example	Equivalent
<code>+=</code>	<code>x += 5</code>	<code>x = x + 5</code>
<code>-=</code>	<code>x -= 2</code>	<code>x = x - 2</code>
<code>*=</code>	<code>x *= 4</code>	<code>x = x * 4</code>
<code>/=</code>	<code>x /= 2</code>	<code>x = x / 2</code>
<code>%=</code>	<code>x %= 3</code>	<code>x = x % 3</code>
<code>//=</code>	<code>x //= 6</code>	<code>x = x // 6</code>
<code>**=</code>	<code>x **= 2</code>	<code>x = x ** 2</code>

Operator Precedence in Python

- Operators have precedence dictating the order in which the operators are applied

Operator	Meaning
()	Parenthesis
**	Exponent
+x, -x	Unary plus, Unary minus
*, /, //, %	Multiplication, Division, Floor Division, Modulus
+, -	Addition, Subtraction
==, !=, >, >=, <=, <, is, is not, in, not in	Comparisons, Identity, Membership
not	Logical NOT
and	Logical AND
or	Logical OR

Decision Making with If Statement

- In Python, we use the if statement to make decisions and execute one or the other block of Python code
- As in all other Python code blocks, a colon starts the block and the entire block must be indented

```
if <condition> :  
    <true-block>  
    ...  
    Instructions that are executed  
    if the condition evaluates to True  
    ...  
else :  
    <false-block>  
    ...  
    Instructions that are executed  
    if the condition evaluates to False  
    ...
```

Complex Decision Making

- Use **if/elif** statement for multiple conditions
- Combine **if/else** with **if/elif** for more complex decision making
- Both **if/elif** and **if/else** can be nested
- When comparing through **if/elif** statement, as soon as a condition is satisfied, further blocks are not evaluated

```
if <condition1> :  
    <true-block>  
    # satisfying conditon1  
elif <condition2> :  
    <true_block>  
    # not satisfying conditon1 but  
    condition2  
elif <condition3> :  
    <true_block>  
...  
else :  
    <false-block>  
    # not satisfying any condition above
```

Repeating Code with While Statement

- As you program, there will be times when you often want to repeat through a section of code repeatedly while just changing a small portion of code
 - Add from 1 to 100
 - Repeat while certain condition(s) is met or unmet
- Most programming languages provide a "**while**" like control for this

```
while <condition> :  
    <block>  
    ...  
    repeat the statements in <block>  
    while the <condition> is True  
    If not, escape the while loop.  
    ...
```

Iteration with For Loop Statement

- A for loop is used for iterating over a sequence
- A sequence it a collection type which is iterable
 - Strings and Lists
 - Others include tuple, dictionaries, sets, etc
- Unlike a while loop where a condition is used to terminate the loop, a for loop iterator has an explicit begin and end

```
for <item> in <iterable> :  
    <block>  
    ...  
    repeat the statements in <block>  
    for each element of the <iterable>, in  
    order  
    ...
```

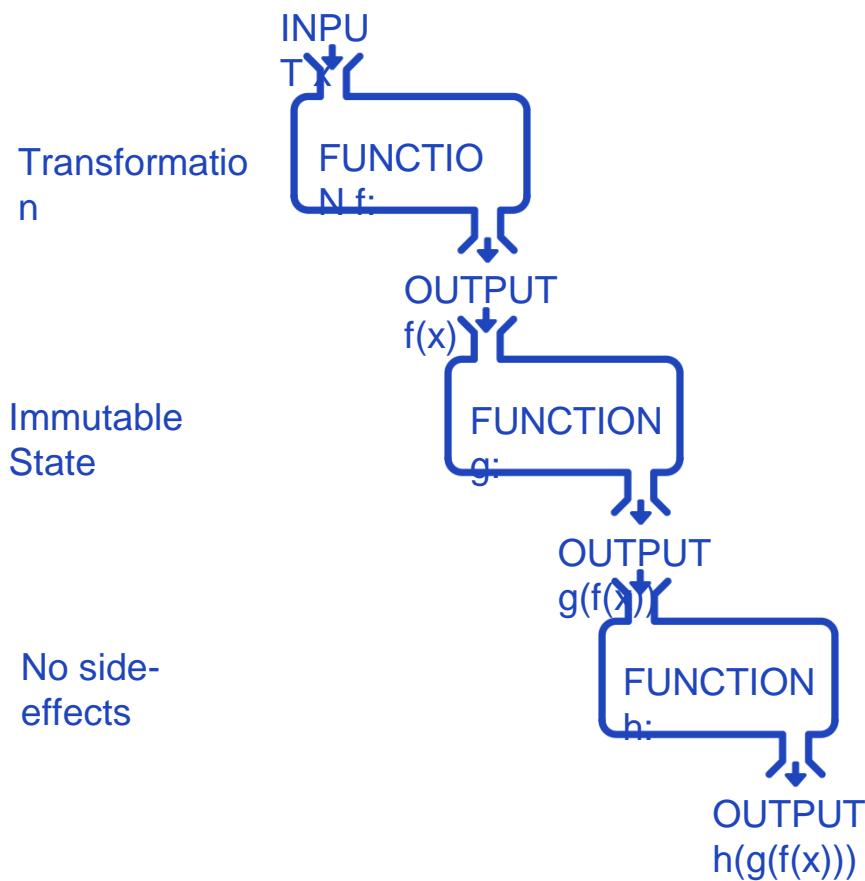
Creating Python Functions

- Create a Function with the keyword **def** followed by the function name
- Next any input parameter are placed inside parenthesis ()
- Next, a colon starts the code block which must be indented
- The first statement of a function can be an optional documentation string called Docstring
- The last statement of the function is **return** which is used to exit and return the results of the function

```
def my_function (parameter1, parameter2, . . .) :  
    """ Documentation about this function """  
    code  
    code  
    . . .  
    return some_result
```

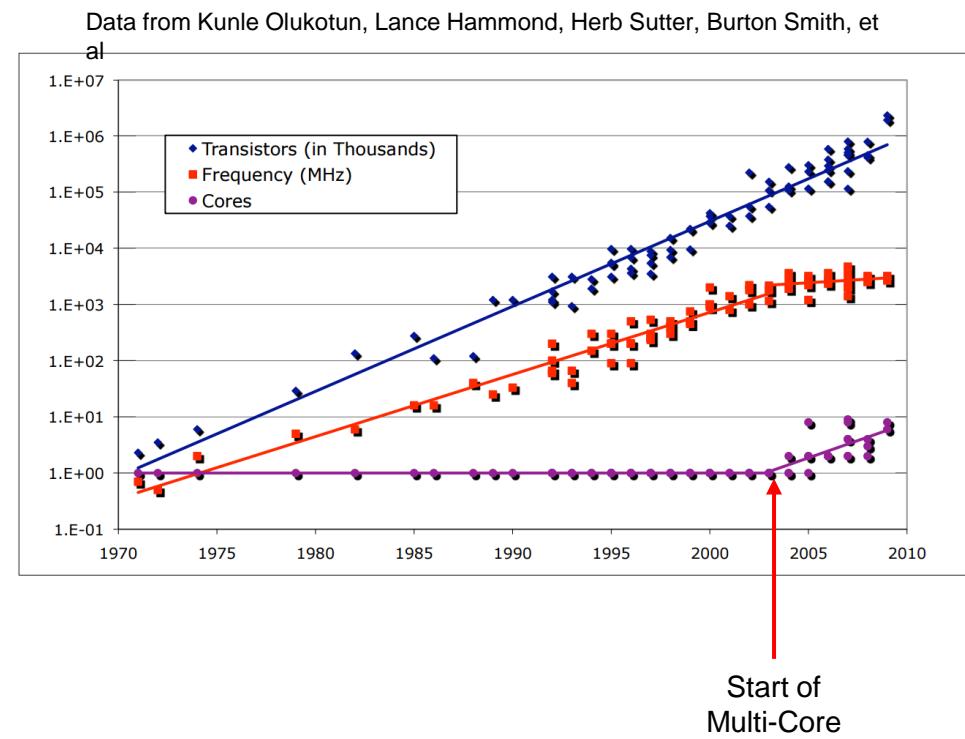
Functional Programming

- Functional programming is a computation style in which programs consist entirely of evaluating pure functions
- A pure function is a function whose output values result entirely from its input values without causing any side effects
- A side effect is when the function causes changes in the calling environment in any way
 - Example: Modify the value of an argument within a function



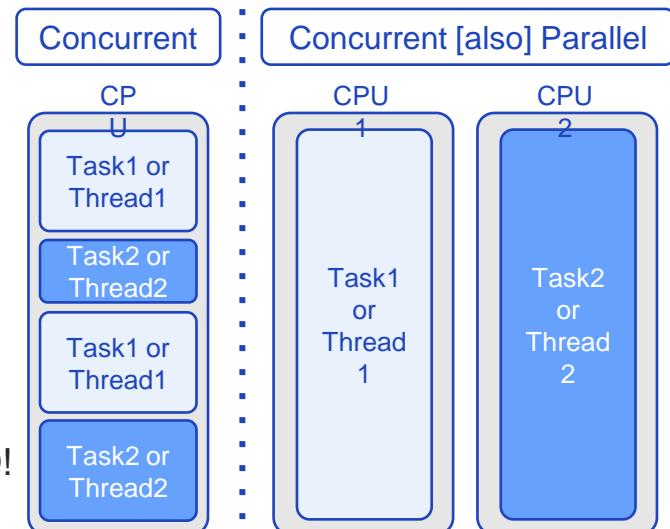
Why Functional Programming?

- We have all been using imperative programming style targeted towards von Neumann architecture machines
 - Imperative programming paradigm uses statements that change a program's state
 - Procedural programming and Object-oriented programming are all imperative programming styles
- Since a little after year 2000, there has been a fundamental change in computer hardware
 - No more improvement in clock speed
 - Multi Cores
- Imperative programming and multi-cores do not play well together
 - Why? Hint: Imperative programs work by changing a program's state



Concurrent and Parallel Programming

- Parallel programming
 - Execute code simultaneously on multiple parallel hardware
 - Hadoop and Spark both work on clusters with multiple computers
- Concurrent programming
 - Execute code on multiple cores in a single machine
 - Each core makes progress at same time
 - Hadoop and Spark uses multiple cores in executing tasks
- Both Parallel programming and Concurrent programming are HARD!
- Non-determinism caused by concurrent threads accessing shared mutual states and modifying the state
 - The combination of parallel/concurrent processing and mutable states is causing the problem
 - Avoid mutable state, make states immutable



Functional Programming in Python

- To support Functional Programming, functions should be first-class citizens
- What does it mean to be a first-class citizen
 - Have equal characteristics as other values such as numbers, strings or collections
 - Be able to save functions to variables
 - Pass functions as arguments to a function
 - Receive function as result of a function

```
def some_func():
    print("some function")

# save to variable
my_variable = some_func

#pass it to another function
print(1, "mystring", some_func)

#return from another function
def another_func(func_parameter):
    return func_parameter()

another_func(some_func)
```

Lambda Function in Python

- When we work with other objects in Python, we don't always save it to a variable before using it

```
my_float = 2.3  
a_function(my_float)      OR      a_function(2.3)
```

- Similarly, we should not have to always define a function in order to use it
- Anonymous Functions in Python allows us to define a function on the fly without defining it

- Use lambda notation**

```
def my_func(str):  
    return str.lower()  
  
a_function(my_func)      OR      a_function(lambda str: str.lower())
```

Unit 1.

Unstructured Data Processing

- | 1.1. Introduction to Apache Spark
- | 1.2. Python Basics
- | 1.3. Data Transformation with Core API
- | 1.4. Working with Pair RDDs

Getting Started with Core API (1/2)

- The easiest way to get started with Spark Development is from the Spark Shell
- The Shell can be started with several options
- The location where the Driver program is created can be determined using the **--master** and **--deploy-mode** options
 - local[N] - Use local mode with N threads
 - local[*] - Use local mode with max available threads
 - client - Use cluster in client mode
 - cluster - Use cluster in cluster mode
- Any additional Java jar with **--jars**
- Add additional Python files with **--py-files**
- Add additional packages with **--packages**

Getting Started with Core API (2/2)

- Various way to start Spark

- Run spark-shell locally

```
$ spark-shell --master local[4]
```

```
$ pyspark --master yarn --deploy-mode client
```

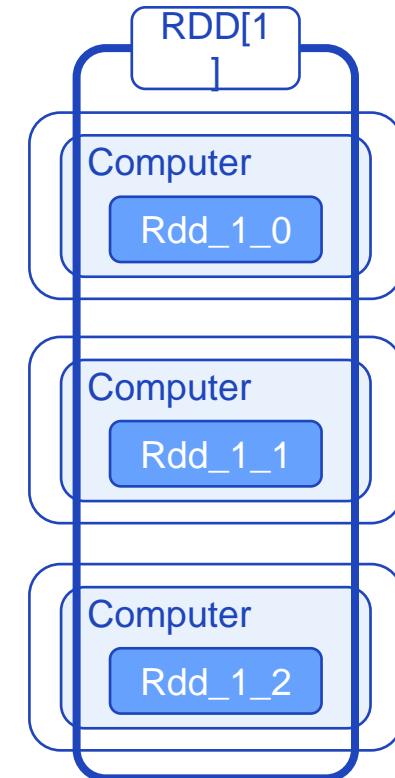
```
$ spark-shell --jars jarfile1, jarfile2
```

```
$ py-spark --master local[*] --py-files py-file1, py-file2
```

```
$ py-spark --master local[*] --packages org.apache.hadoop:hadoop-aws:3.1.2
```

Resilient Distributed Datasets

- Resilient Distributed Datasets (RDD) is a fundamental data structure in Spark
- Primary data abstraction in Apache Spark and the Spark Core API
- RDDs are fault-tolerant immutable distributed collection of objects that can be operated in parallel
 - Spark partitions the data over the Executors
 - Each Executor operates over its portion of the data
 - All Executors operate in parallel
 - If any data is lost, Spark will automatically recreate it



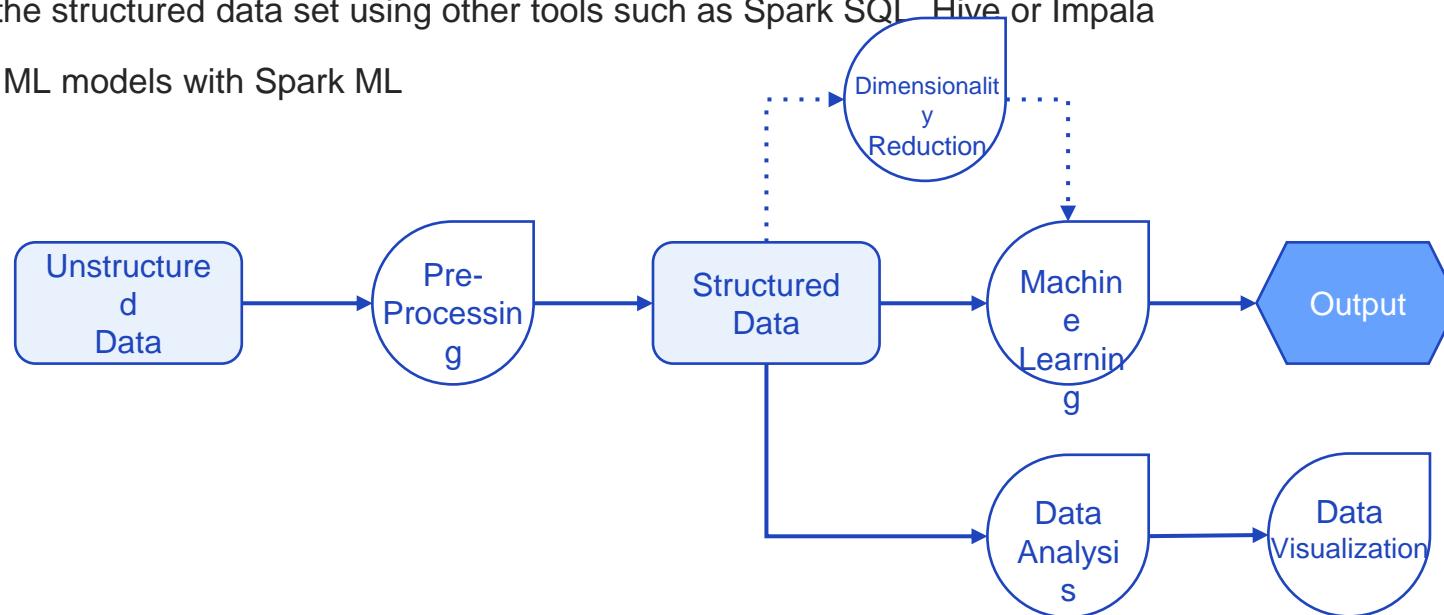
RDD Content Data Type

- What kind of data can be stored in RDDs?
 - Primitive data types such as integer, float, double, string, characters, Boolean, etc.
 - Complex data types such as lists, arrays, sequences, tuples, dictionaries
 - Nested complex types, including mixed types
 - Any serializable Scala or Java objects



RDD: Most Common Use Case

- In most cases, RDDs will consist of unstructured or semi-structured data
- A very common use case is to pre-process the unstructured/semi-structured data and convert to a structured data set
 - Query the structured data set using other tools such as Spark SQL, Hive or Impala
 - Create ML models with Spark ML



Creating Resilient Distributed Datasets

- The easiest and the simplest method for creating RDDs is to create them using existing iterables or collections in the driver program
- Use SparkContext's **parallelize** method
- The operation creates a parallelized collection where the collection is partitioned and distributed over participating Executors
 - Control number of partitions with `parallelize(my_collection, number_of_partitions)`
- ~~Mostly used during development for quick testing~~

```
my_collection = [1, 2, 3, 4]
parallel_collection = sc.parallelize(my_collection)
parallel_collection.take(2)
```

Input

[1, 2]

Output

Reading External Datasets

- Spark can create distributed datasets from any storage source supported by Hadoop
 - Local file system or HDFS file system
 - NoSQL such as HBase and Cassandra
 - Cloud sources such as Amazon S3
- Unstructured or semi-structured datasets should be read using the Core API
 - Unstructured text files such as natural language files or semi-structured such as log files
- Structured datasets should be read using Dataframe API
 - Binary files that include schema information such as Parquet, ORC, etc
 - Data from tables such as Hive, HBase or Cassandra
- Some files may be read using Core API or Dataframe API, depending on use case
 - Text files that are semi-structured such as JSON, XML, or CSV Files

Create RDD from Text Files

- Create text file RDDs using SparkContext's **textFile** method
 - sc.textFile(<path to dataset>)
 - Path to dataset can be directories or specific files
 - Use absolute path or relative path
 - Default relative path is user's HDFS home directory

```
hdfs_abs_wild = sc.textFile("/labs/*.txt")
print("Dataset using wild card: ", hdfs_abs_wild.count())
hdfs_abs_list = sc.textFile("/labs/data1.txt,/labs/data2.txt")
print("Dataset from a list:      ", hdfs_abs_list.count())
hdfs_directory = sc.textFile("/labs")
print("Dataset from directory:  ", hdfs_directory.count())
```

Input

```
Dataset using wild card:      202
Dataset from a list:         202
Dataset from directory:      202
```

Output

Specify Local or HDFS Files

- Specify specific URI to access HDFS or local files
 - `hdfs://<namenode host>/<path to dataset>`
 - `file:/<path to local dataset>`
- Some considerations for local file system
 - If accessing a local file, Spark requires that the local file exist on all nodes in the cluster at same location

```
hdfs_uri_file = sc.textFile("hdfs://localhost:9000/user/student/alice.txt") Input
print("Alice from HDFS URI path: ", hdfs_uri_file.count())
hdfs_relative_path = sc.textFile("alice.txt")
print("Alice from relative path file: ", hdfs_relative_path.count())
local_file = sc.textFile("file:/home/student/Data/alice_in_wonderland.txt")
print("Alice from local home directory: ", local_file.count())
```

Alice from HDFS URI path :	3761	Output
Alice from relative path file:	3761	
Alice from local home directory:	3761	

Create RDD from s3 Buckets

- Specify specific URI to access S3 files
 - s3a://<bucket and path to dataset>
- Start the shell with the following package
 - pyspark --packages=com.amazonaws:aws-java-sdk-bundle:1.11.271, org.apache.hadoop:hadoop-aws:3.1.2

```
access_key = "xxxxxxxxxxxxxxxxxxxxxx"
secret_key = "xxxxxxxxxxxxxxxxxxxxxx"
hadoop_conf=sc._jsc.hadoopConfiguration()
hadoop_conf.set("fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem")
hadoop_conf.set("fs.s3a.access.key", access_key)
hadoop_conf.set("fs.s3a.secret.key", secret_key)
s3_file = sc.textFile("s3a://samsung-student-lab/alice_in_wonderland.txt")

print("Alice from s3 directory: ", s3_file.count())
```

Input

Alice from s3 directory: 3761

Output

How Does Spark Read Text?

- When Spark reads text files, each newline is used to separate elements of an RDD
- We see that Spark displays the RDD as a List of Strings
 - Python uses square brackets [] for lists
- Any whitespace, tabs, commas , quotes, etc. are part of the element



"How doth the little crocodile Improve his shining tail, And pour the waters of the Nile On every golden scale!"
"How cheerfully he seems to grin, How neatly spread his claws, And welcome little fishes in With gently smiling jaws!"

```
data_src = "file:/home/student/Data/except.txt"
textRDD = sc.textFile(data_src)
textRDD.take(4)
```

Input

```
[ 'How doth the little crocodile',
  'Improve his shining tail, ',
  'And pour the waters of the Nile',
  'On every golden scale!' ]
```

Output

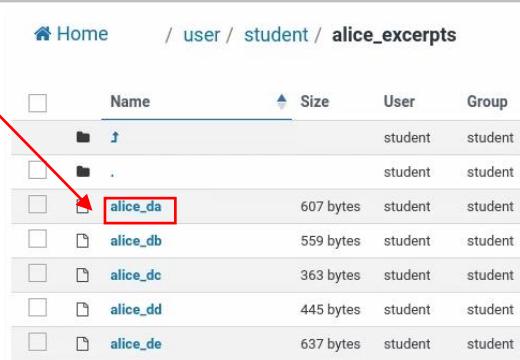
Reading Whole Text Files

- Sometimes we do not want the newline (\n) to delimit each element of an RDD
 - Records in JSON or XML format span over multiple lines
- Use SparkContext's **wholeTextFiles** method with a source directory
 - Each file in the source directory becomes a single element in the RDD

```
Data_src = "alice_excerpts"
wholeRDD = sc.wholeTextFiles(data_src)
wholeRDD.take(1)
```

[("hdfs://localhost:9000/user/student/alice_excerpts/alice_da",
'"Mary Ann! Mary Ann!" said the voice. "Fetch me my gloves th
is moment!"\nThen came a little patterning of feet on the stair
s. Alice knew it was\nthe Rabbit coming to look for her, and sh
e trembled till she shook the\nhouse, quite forgetting that she
was now about a thousand times as\nlarge as the Rabbit, and had
no reason to be afraid of it.\n\nPresently the Rabbit came up t
o the door, and tried to open it; but, as\nthe door opened inwa
rds, and Alice's elbow was pressed hard against it,\nthat attem
pt proved a failure. Alice heard it say to itself "Then I'll\nng o round and get in at the window."\n')]

Input



Name	Size	User	Group
..		student	student
.		student	student
alice_da	607 bytes	student	student
alice_db	559 bytes	student	student
alice_dc	363 bytes	student	student
alice_dd	445 bytes	student	student
alice_de	637 bytes	student	student

Reading Other File Formats

- Apache Spark can read any file format supported by Hadoop
 - Internally, Spark uses the Hadoop JAR file libraries for reading data sources
 - Use SparkContext's hadoopFile (Hadoop 1.x) or newAPIhadoopFile (Hadoop 2.x)
- SparkContext's textFile is in fact, a shortcut

```
data_src = "alice.txt"
apiFile = sc. \
newAPIHadoopFile(
    data_src,
    "org.apache.hadoop.mapreduce.lib.input.TextInputFormat", # inputFormatClass
    "org.apache.hadoop.io.Text", # keyClass
    "org.apache.hadoop.io.LongWritable", # valueClass
)
print("Alice from API input format: ", apiFile.count())
```

Input

Alice from API input format: 3761

Output

RDD Operations

- RDDs support two types of operations
 - Transformations create a new dataset from existing one
 - Action return a value to the Driver program after running a computation on the dataset
- Operation Examples:
 - **map** is a transformation that passes each dataset element through a function and returns a new RDD representing the result
 - **reduce** is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program
- Lazy Transformation
 - All Spark transformation are lazy transformation
 - They do not get executed until an action is called requiring a return value
- Immutable Transformations
 - Spark transformations are immutable - the transformation creates a new RDD

RDD Actions

- Actions that do not require a function as parameter

Action	Meaning
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data
count()	Return the number of elements in the dataset
first()	Return the first element of the dataset (similar to take(1))
take(n)	Return an array with the first n elements of the dataset
saveAsTextFile(path)	Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call toString on each element to convert it to a line of text in the file

RDD Actions Output

- **count()** and **first()** both return values
- **take(n)** and **collect()** both return collections
- Recall that actions return values to the Driver program from all of the Executors
 - Do NOT use **collect()** in production code

```
myRDD = sc.parallelize([1, 2, 3, 4, 5, 6])
print("This RDD has", myRDD.count(), "elements")
print("The first element is", myRDD.first())
print("The first 2 elements are", myRDD.take(2))
print("The entire collection is", myRDD.collect())
```

Input

```
This RDD has 6 elements
The first element is 1
The first 2 elements are [1, 2]
The entire collection is [1, 2, 3, 4, 5, 6]
```

Output

Saving a RDD as Text File

- Use **saveAsTextFile(path)** to save an RDD as a text file on the path provided as parameter
 - path is a directory that Spark will create
 - if path already exists, Spark will give ERROR
 - If relative path is used, Spark will default to the HDFS home directory of user
- Notice that there are several files saved in the directory
 - Each partition will be saved as a separate file

	Name	Size	User	Group
	..		student	student
	.		student	student
	part-00005	2 bytes	student	student
	part-00004	2 bytes	student	student
	part-00003	2 bytes	student	student
	part-00002	2 bytes	student	student
	part-00001	2 bytes	student	student
	part-00000	2 bytes	student	student
	_SUCCESS	0 bytes	student	student

Input

```
myRDD = sc.parallelize([1, 2, 3, 4, 5, 6])
myRDD.saveAsTextFile("myFirstRDD")
```

RDD Actions

- Actions that require a function to complete

Action	Meaning
reduce(func)	Aggregate the elements of the dataset using a function func (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel
foreach(func)	Run a function func on each element of the dataset. This is usually done for side effects such as updating an Accumulator or interacting with external storage systems
foreachPartition(func)	Run a function func on each partition of the dataset
getNumPartitions()	Return the number of partitions

The **reduce** RDD Action

- The **reduce(func)** action returns the result of applying func to the elements of the RDD
- func is function that takes two arguments
- func must be associative
 - $x \text{ operator } y$ is equal to $y \text{ operator } x$
- func must be commutative
 - $(x \text{ operator } y) \text{ and then operator } z$ is equal to $(y \text{ operator } z) \text{ and then operator } x$

```
myRDD = sc.parallelize([1, 2, 3, 4, 5, 6])
myRDDsum = myRDD.reduce(lambda l, r: l+r)
print("The sum of my RDD is", myRDDsum)
```

Input

```
The sum of my RDD is 21
```

Output

RDD foreach Action

- The **foreach** action applies a function to each element in the RDD
- However, the action itself returns None as can be seen in the output
- The supplied function is usually used to affect some side effect such as pretty printing each element as shown in the example

Input

```
def wow_print(e):
    print("-"*(len(e)+4))
    print("*", e, "*")
    print("-"*(len(e)+4))

my_fruits = ["apples", "oranges", "pear",
             "banana"]
dataRDD = sc.parallelize(my_fruits)
print("The output of foreach is: ",
      dataRDD.foreach(wow_print))
```

Output

```
The output of foreach is : None
-----
* apples *
-----
-----
* oranges *
-----
-----
* pear *
-----
-----
* banana *
-----
```

RDD Partition Actions

- These actions apply at the partition level
- The function passed to **foreachPartition** is expected to have an iterator parameter
 - The iterator can be used in a for loop to access each element in the partition

```
def show_part(it):
    for i in it:
        print(i)
    print("*****")
partRDD = sc.parallelize([1, 2, 3, 4, 5, 6], 2)
print("This RDD has", partRDD.getNumPartitions(), "partitions")
partRDD.foreachPartition(lambda iterator:
    show_part(iterator))
```

Input

Output

```
This RDD has 2 partitions
1
2
3
*****
4
5
6
*****
```

Spark Transformations

- Spark transformations can be categorized by number of dependent parent RDDs
 - Narrow Dependency - Only a single parent RDD is required to create the new RDD
 - Wide Dependency - Several RDDs are required to create the new RDD
- Another category is whether a function parameter is required or not
 - Some transformations do not require a separate function and has internal logic on how to transform
 - An example for such a transformation would be `distinct()` that removed duplicate elements
- Operation at element level or partition level
 - Some transformations affect each element while others works at the partition level
- Certain transformations work on set operations
 - Requires to dataset
 - This is different from Wide dependency

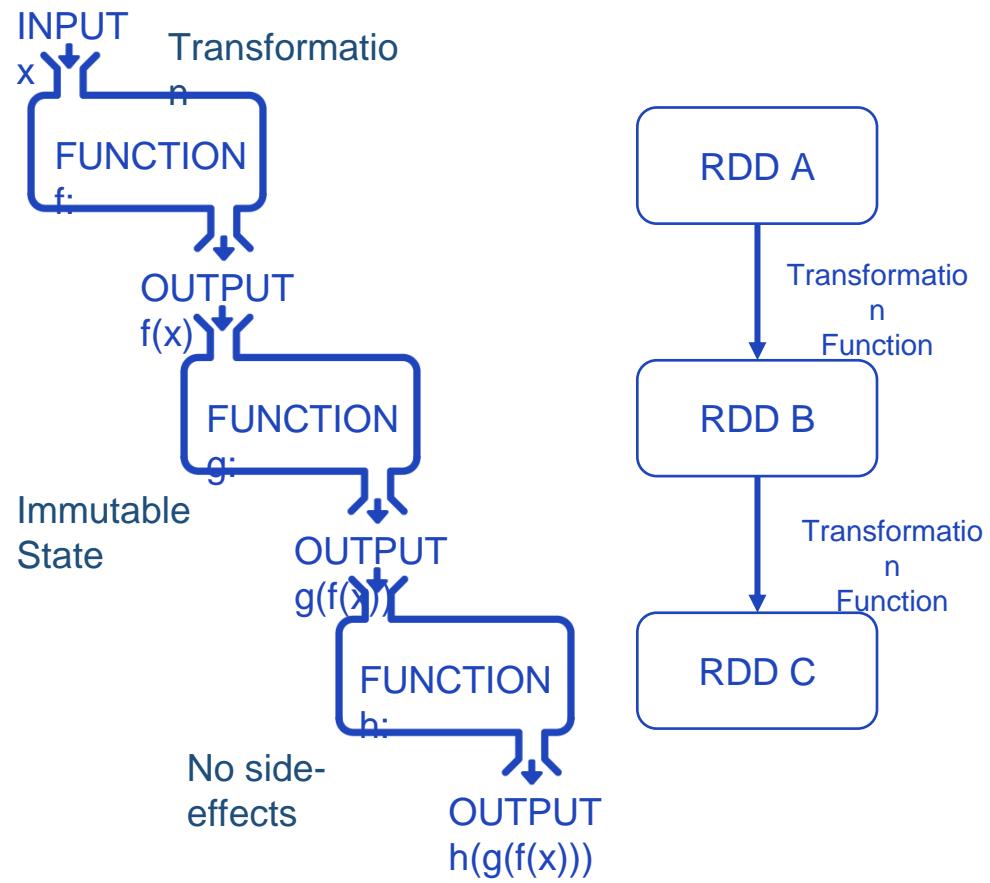
Basic RDD Transformations

- Transformations of a single dataset

Transformation	Meaning
map(func)	Return a new distributed dataset formed by passing each element of the source through a function func
filter(func)	Return a new dataset formed by selecting those elements of the source on which func returns true.
distinct()	Return a new dataset that contains the distinct elements of the source dataset
flatMap(func)	Similar to map, but each input item can be mapped to 0 or more output items (so func should return a Seq rather than a single item)

Passing Functions to Transformations (1/2)

- Recall Functional Programming from the Python lessons
 - Pass functions to transform objects
 - All objects are immutable and transformation does not remove or modify the original object
 - This prevents side-effects where functions cause changes to the state and calling environment
- Apache Spark utilizes Functional Programming
 - Transformation methods immutably create new RDDs where the function has been applied to every element of the source RDD
 - Most transformations require a function as a parameter



Passing Functions to Transformations (2/2)

- In most cases, anonymous functions are used to pass functions as parameters to a transformation
 - In Python - lambda notation

```
transformation( lambda param 1, param 2, . . . :  
    transformation code using parameters)
```

- In Scala - arrow notation

```
transformation( param 1:type , param 2:type , . . . =>  
    transformation code using parameters)
```

- ```
def myFunction(param)
transformation(myFunction)
```

# Chaining Transformations

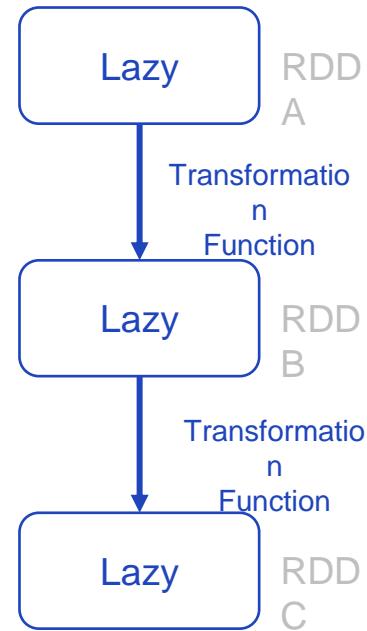
- We could save each transformation into a separate RDD

```
rdd1 = sc.textFile(path)
rdd2 = rdd1.transformation(lambda param 1, : transformation)
rdd3 = rdd2.transformation(lambda param 1, : transformation)
rdd4 = rdd3.transformation(lambda param 1, : transformation)
```

- 
- ```
rdd1  = sc.textFile( path) \
        .transformation( lambda param 1, : transformation) \
        .transformation( lambda param 1, : transformation) \
        .transformation( lambda param 1, : transformation) \
```

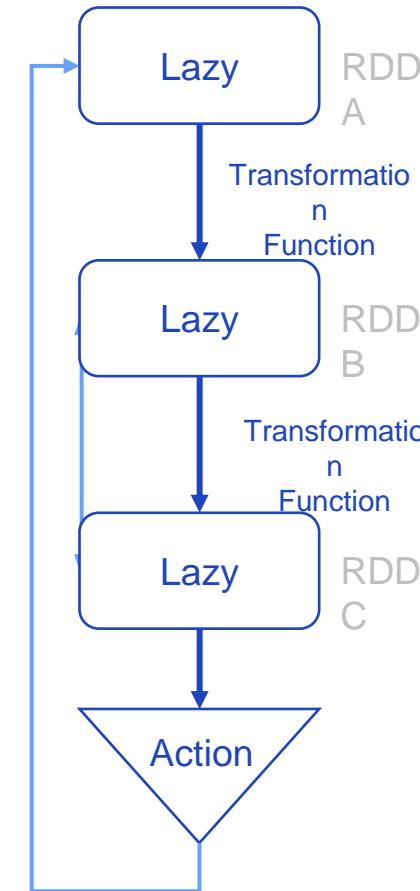
Lazy Execution (1/3)

- Spark transformation are lazy transformation
 - Until an action is called, requiring a value, all the transformations are on hold



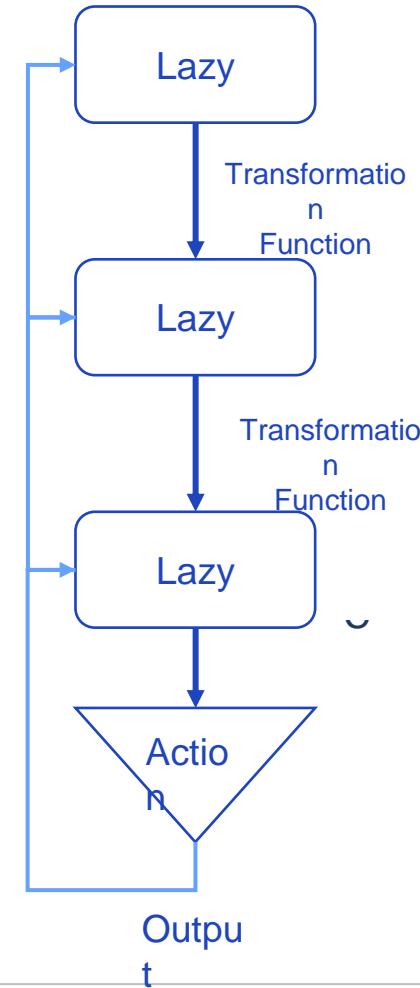
Lazy Execution (2/3)

- Spark transformation are lazy transformation
 - Until an action is called, requiring a value, all the transformations are on hold
- When an action is called, Spark evaluates the dependency for that action
 - This is a Directed Acyclic Graph (DAG) where each node represents a RDD and each edge represents a transformation operation



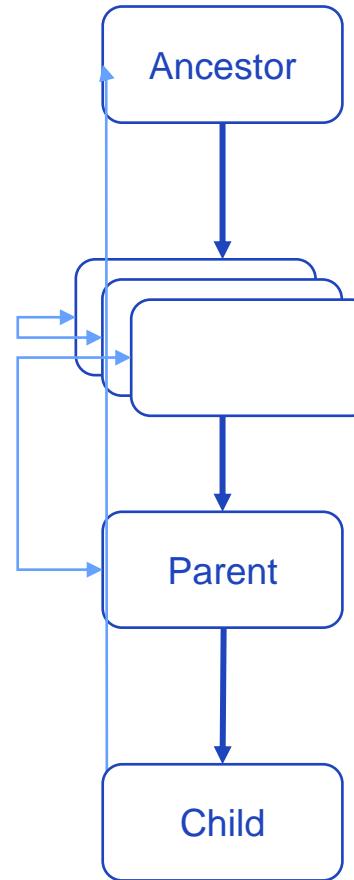
Lazy Execution (3/3)

- Spark transformation are lazy transformation
 - Until an action is called, requiring a value, all the transformations are on hold
- When an action is called, Spark evaluates the dependency for that action
 - This is a Directed Acyclic Graph (DAG) where each node represents a RDD and each edge represents a transformation operation
- When Spark reaches the top unevaluated RDD, it begins to perform the actual transformations
 - Each transformation triggers the next RDD to be created until the final action output is produced



Spark Lineage

- Transformations create new RDDs based on one or more existing RDDs
- We described this as a dependency graph where the child RDD depends on one or more parent RDDs
 - Parent RDDs in turn depend on RDDs further up the dependency graph
 - This dependency graph can be described as a Directed Acyclic Graph (DAG)
- Each RDD can be said to have a lineage with a sequence of ancestors that it depends on



Map and Filter Transformation Example

- Use **map** to transform all characters to lower case, and then filter for lines that contains "alice"

Input	Output
<pre>data_src = "alice_excerpts" aliceRDD = sc.textFile(data_src) \ .map(lambda line: line.lower()) \ .filter(lambda line: "alice" in line) for line in aliceRDD.take(5): print(line) print("There were", aliceRDD.count(), "lines with Alice in it")</pre>	<p>then came a little patterning of feet on the stairs. alice knew it was the door opened inwards, and alice's elbow was pressed hard against it, that attempt proved a failure. alice heard it say to itself "then i'll "<u>_that_</u> you won't!" thought alice, and, after waiting till she fancied there was a long silence after this, and alice could only hear whispers There were 61 lines with Alice in it</p> <p>[Stage 12 : =====> (48+6) There were 61 lines with Alice in it</p>

Spark `toDebugString()`

- Spark maintains the lineage information for each RDD
- Using the `toDebugString()` method, it is possible to see each RDDs lineage information

Input

```
data_src = "alice_excerpts"
aliceRDD = sc.textFile(data_src) \
    .map(lambda line: line.lower()) \
    .filter(lambda line: "alice" in line)
print(aliceRDD.toDebugString().decode("utf-8"))
```

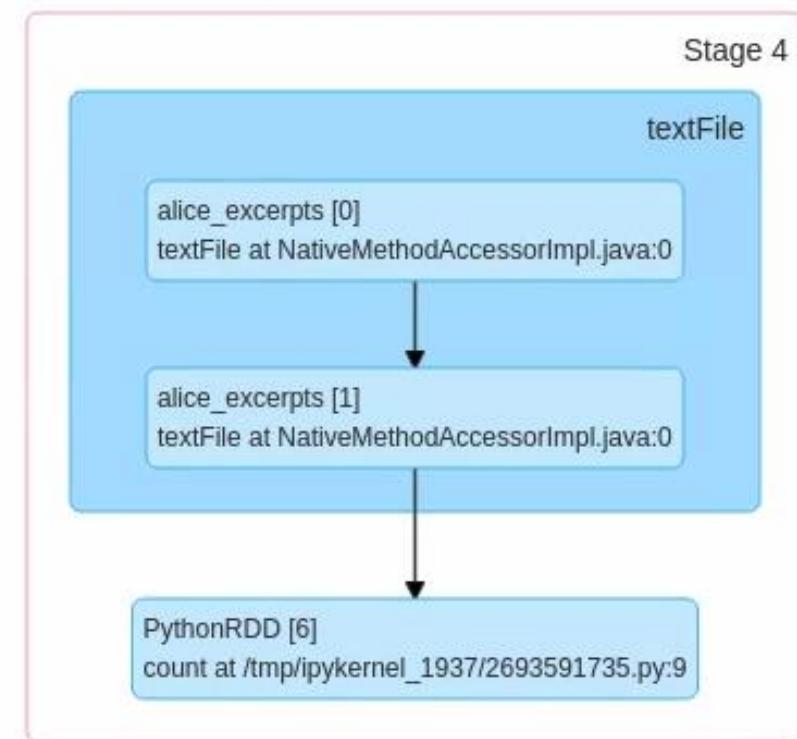
Output

```
(65) PythonRDD[10] at RDD at PythonRDD.scala:53 [ ]
|   alice_excerpts MapPartitionsRDD[9] at textFile at NativeMethodAccessorImpl.java:0 [ ]
|   alice_excerpts HadoopRDD[8] at textFile at NativeMethodAccessorImpl.java:0 [ ]
```

View the DAG on Spark Web UI

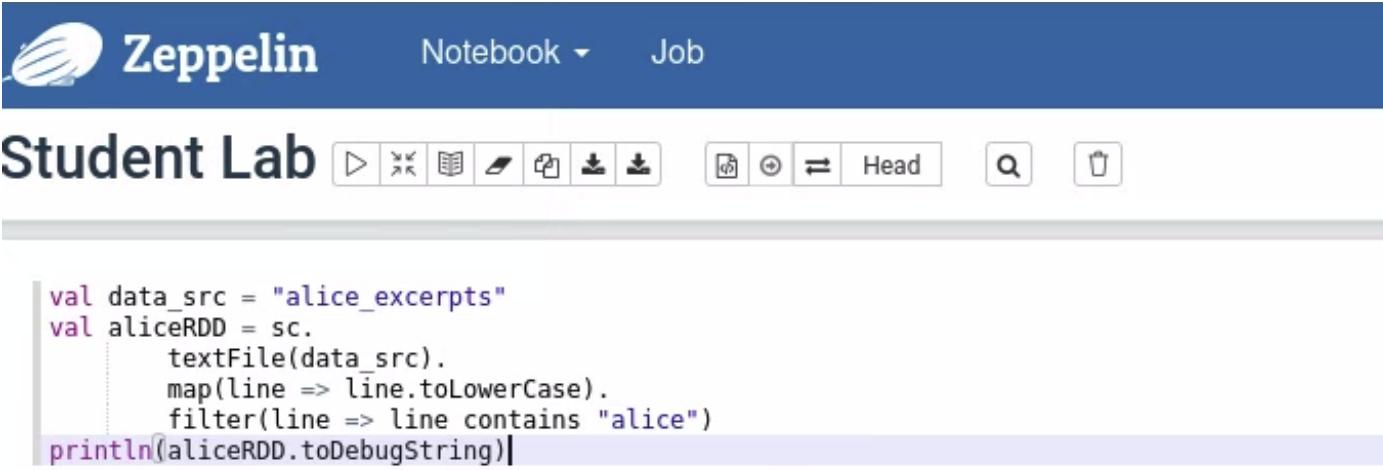
- Apache Spark has a browser based Spark Web UI
- When running in local mode, it can be viewed on the localhost at port 4040
 - The port can be configured to another number
- Each SparkContext generates its own Spark Web UI interface
 - If multiple sparkcontexts exist, the port is incremented by 1 as each instance is created
 - From 4040 to 4041, for example
- When running in cluster mode, the Spark Web UI can be accessed from the YARN Web UI

▼ DAG Visualization



Spark toDebugString - Scala

- Scala's `toDebugString` is a little more informative



The screenshot shows the Zeppelin interface with a blue header bar containing the Zeppelin logo and navigation links for "Notebook" and "Job". Below the header is a toolbar with various icons. The main area is titled "Student Lab". In the code editor, there is a block of Scala code:

```
val data_src = "alice_excerpts"
val aliceRDD = sc.
    textFile(data_src).
    map(line => line.toLowerCase).
    filter(line => line contains "alice")
println(aliceRDD.toDebugString)
```

The output below the code shows the resulting `toDebugString` representation:

```
(65) MapPartitionsRDD[7] at filter at <console>:32 []
|  MapPartitionsRDD[6] at map at <console>:31 []
|  alice_excerpts MapPartitionsRDD[5] at textFile at <console>:29 []
|  alice_excerpts HadoopRDD[4] at textFile at <console>:29 []
data_src: String = alice_excerpts
aliceRDD: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[7] at filter at <console>:32
```

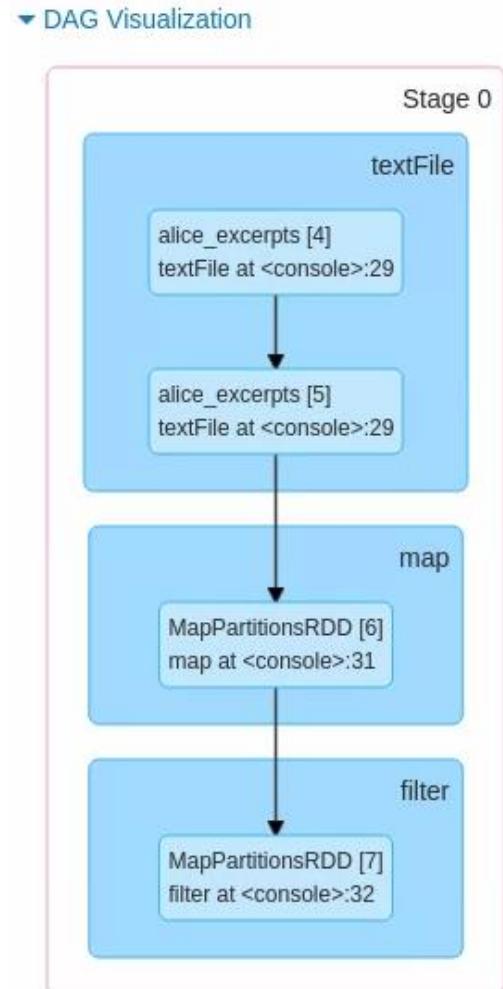
Spark Web UI - Scala

- Each completed job is displayed and can be selected
- In the description, you will see the name of the "Action" that triggered the job to execute
 - In our case, it was count

▼ Completed Jobs (1)

Page: 1

Job Id (Job Group) ▾	Description
0 (zeppelin anonymous 2GESRDZRZ paragraph_1629880937329_1360145546)	Started by: anonymous count at <console>:26



FlatMap Transformation (1/3)

- The transformation function in **flatMap** should return a collection or sequence
- Similar to map, however, **flatMap** takes each item in the collection or sequence and outputs as a separate element
- Looking at the code:
 - dataRDD is single element RDD consisting of the fruits string
 - In both cases, the split method takes the string and returns a collection of fruits
 - **map** returns that collection
 - **flatMap** take each item in the collection and outputs it as a separate element

FlatMap Transformation (2/3)

```
my_fruits = ["apples oranges pear banana"]
dataRDD = sc.parallelize(my_fruits)
print("Number of elements is this RDD:", dataRDD.count())
print("It contains the following string:", dataRDD.collect())
```

Input

```
Number of elements is this RDD: 1
It contains the following string: ['apples oranges pear banana']
```

Output

```
mapRDD = dataRDD.map(lambda line: line.split(' '))
for line in mapRDD.collect():
    print(line)
```

Input

```
['apples', 'oranges', 'pear', 'banana']
```

Output

FlatMap Transformation (3/3)

Input

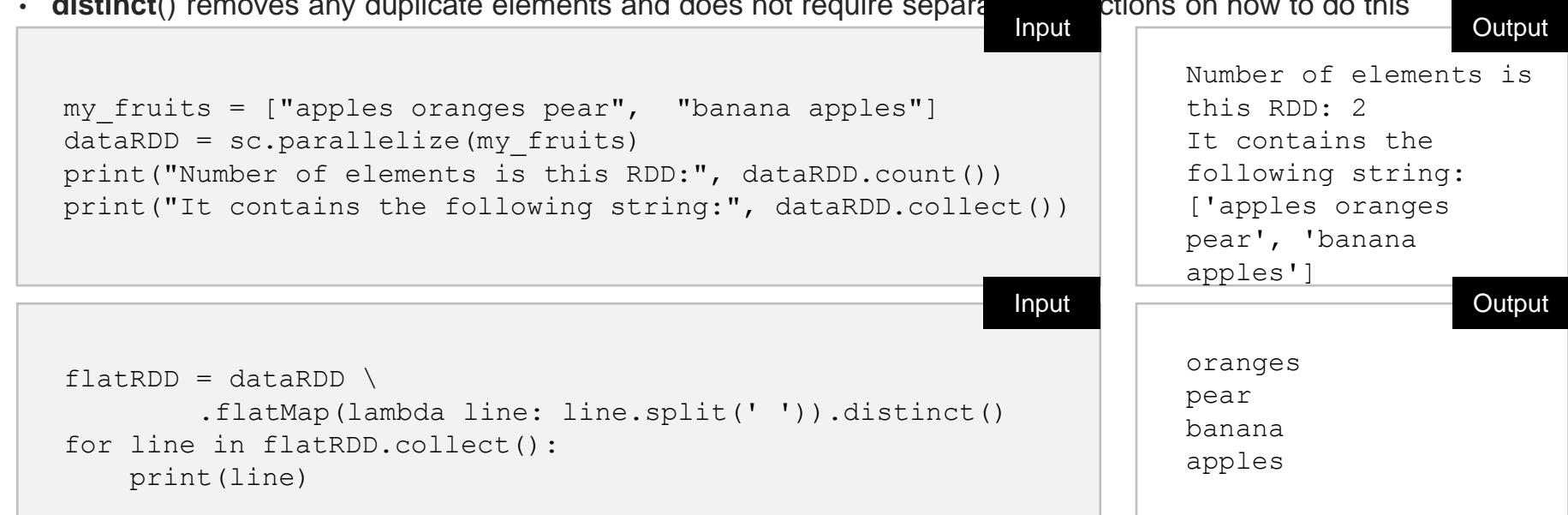
```
flatRDD = dataRDD.flatMap(lambda line: line.split(' '))
for line in flatRDD.collect():
    print(line)
```

Output

```
apples
oranges
pear
banana
```

Distinct() Transformation

- So far, we have seen that all the transformation require a function to be passed as a parameter
 - The function determines how the transformation will take place
- Some transformation methods, however, do not require a separate function to be passed
 - **distinct()** removes any duplicate elements and does not require separate instructions on how to do this



Set Operator RDD Transformation

- Transformation of multiple datasets

Transformation	Meaning
union(otherDataset)	Return a new dataset that contains the union of the elements in the source dataset and the argument
intersection(otherDataset)	Return a new RDD that contains the intersection of elements in the source dataset and the argument
cartesian(otherDataset)	When called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements)
subtract(otherDataset)	Return a new dataset that contains all elements in the source dataset that is not contained in the other dataset

Set Operator Transformations (1/2)

- These transformations take two datasets and perform set operations on them

```
fruits1 = ["apples", "oranges", "pear"]
fruits2 = ["banana", "apples"]
fruit1RDD = sc.parallelize(fruits1)
fruit2RDD = sc.parallelize(fruits2)

fruit1RDD.union(fruit2RDD).collect()
```

Input

```
['apples', 'oranges', 'pear', 'banana', 'apples']
```

Output

```
fruit1RDD.intersection(fruit2RDD).collect()
```

Input

```
['apples']
```

Output

Set Operator Transformations (2/2)

- These transformations take two datasets and perform set operations on them

```
fruit1RDD.subtract(fruit2RDD).collect()
```

Input

```
['oranges', 'pear']
```

Output

```
fruit1RDD.cartesian(fruit2RDD).collect()
```

Input

```
[('apples', 'banana'),  
 ('apples', 'apples'),  
 ('oranges', 'banana'),  
 ('oranges', 'apples'),  
 ('pear', 'banana'),  
 ('pear', 'apples')]
```

Output

Partition Based RDD Transformation

- Transformations that operate on the partition level, instead of the row level

Transformation	Meaning
mapPartitions(func)	Similar to map, but runs separately on each partition (block) of the RDD, so func must be of type <code>Iterator<T> => Iterator<U></code> when running on an RDD of type T
mapPartitionsWithIndex (func)	Similar to mapPartitions, but also provides func with an integer value representing the index of the partition, so func must be of type <code>(Int, Iterator<T>) => Iterator<U></code> when running on an RDD of type T

RDD Partition Transformation

- The **mapPartition(func)** transformation operates at the partition level
- As part of the API protocol, **mapPartition** provides an iterator to be used by func
- func is expected to have an iterator parameter that can be used to navigate the elements of the partition
- func is expected to return an iterator of map Partition the results

Python yield is used here to provide this functionality.

```
def show_part(it):
    for i in it:
        print(i)
    print("*****")
partRDD = sc.parallelize([1, 2, 3, 4, 5, 6], 2)
print("This RDD has",partRDD.getNumPartitions(),"partitions")
partRDD.foreachPartition(lambda iterator: show_part(iterator))
```

Input

Output

```
6
*****
15
*****
```

RDD Partition with Index Transformation

- The **mapPartitionWithIndex(func)** transformation works exactly like the **mapPartitions(func)** transformation but with the added functionality of keeping track of the partition index number
 - Partition index numbers start at 0
- The transformation will provide both partition index value and an iterator to navigate through the elements of the partition
- func is expected to return a iterator as was the case for **mapPartitions**
 - Yield statement provides this functionality

Input

```
def show_part(iterator):
    for i in iterator:
        print(i)
    print("*****")
def add_I(index, it):
    yield (index, sum(it))

partRDD = sc.parallelize([1, 2, 3, 4, 5, 6], 2)
partRDD \
    .mapPartitionsWithIndex(lambda ix, it: add_I(ix, it)) \
    .foreachPartition(lambda iterator: show_part(iterator))
```

Output

```
(0, 6)
*****
(1, 15)
*****
```

Data Redistribution RDD Transformation

- Transformations that change number of partitions

Transformation	Meaning
coalesce(numPartitions)	Decrease the number of partitions in the RDD to numPartitions. Useful for running operations more efficiently after filtering down a large dataset.
repartition(numPartitions)	Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network.
repartitionAndSortWithinPartitions (partitioner)	Repartition the RDD according to the given partitioner and, within each resulting partition, sort records by their keys. This is more efficient than calling repartition and then sorting within each partition because it can push the sorting down into the shuffle machinery.

Coalesce Transformation

- The **coalesce** transformation changes the number of partitions without shuffling the data
- Shuffling the data is a very expensive operation
- Typically used to reduce number of partitions after a large dataset has been pruned

Home / user / student / coalesceRDD

	Name	Size	User
□	↳		student
□	.		student
□	_SUCCESS	0 bytes	student
□	part-00000	12 bytes	student

```
myRDD = sc.parallelize([1, 2, 3, 4, 5, 6], 3)
print("Original # of partitions:", myRDD.getNumPartitions())
coalesceRDD = myRDD.coalesce(1)
print("New # of partitions:", coalesceRDD.getNumPartitions())
coalesceRDD.saveAsTextFile("coalesceRDD")
```

Input

```
Original # of partitions: 3
New # of partitions: 1
```

Output

Repartition RDD Transformation (1/2)

- Like the **coalesce** transformation, **repartition** may be used to change the number of partitions
 - Unlike, **coalesce**, **repartition** reshuffles the data
 - This can be useful to fix the skewing problem
 - Skewing occurs when one or few partitions contain the majority of the dataset
 - In a distributed parallel system, the time to completion is dictated by the slowest partition
 - Therefore, it is important to try to make each partition's workload be evenly distributed
- Shuffling is an expensive operation, and care should be taken when used

Repartition RDD Transformation (2/2)

```
myRDD = sc.parallelize([1, 2, 3, 4, 5, 6, 7, 8, 9], 3)
print("Original RDD:", myRDD.getNumPartitions())
myRDD.foreachPartition(lambda iterator: show_part(iterator))

shuffledRDD = myRDD.repartition(3)
print("Reshuffled RDD:", shuffledRDD.getNumPartitions())
shuffledRDD.foreachPartition(lambda iterator: show_part(iterator))
```

Input

Original RDD: 3

```
1
2
3
*****
4
5
6
*****
7
8
9
*****
```

Output

Reshuffled RDD: 3

```
4
5
6
*****
1
2
3
7
8
9
*****
```

Output

Miscellaneous RDD Transformation

- Miscellaneous transformations

Transformation	Meaning
sample(withReplacement, fraction, seed)	Sample a fraction with Replacement of the data, with or without replacement, using a given random number generator seed.
pipe(command, [envVars])	Pipe each partition of the RDD through a shell command, e.g. a Perl or bash script. RDD elements are written to the process's stdin and lines output to its stdout are returned as an RDD of strings.

Sampling Transformation

- Use **sample(withReplacement, fraction, seed)** to generate a new RDD with sampled data
 - Use fraction to control the percentage of sampled data
 - Use the same seed to ensure you get the same sampled data when repeating experiments
- withReplacement controls whether you can have repeated samples
 - Set to True to allow repeated values

```
myRDD = sc.parallelize(range(100))
print(myRDD.sample(False, 0.3, 123).collect())
```

Input

```
[1, 4, 5, 7, 9, 13, 14, 17, 22, 24, 26, 38, 43, 46, 54,
55, 56, 59, 60, 63, 67, 68, 78, 83, 93, 96, 99]
```

Output

```
myRDD = sc.parallelize(range(100))
print(myRDD.sample(True, 0.3, 123).collect())
```

Input

```
[0, 11, 13, 14, 20, 22, 23, 27, 27, 28, 29, 34, 34, 35,
37, 43, 51, 60, 62, 64, 70, 70, 71, 77, 77, 87, 89]
```

Output

RDD pipe Transformation (1/2)

- The pipe transformation allows developers to execute script files from the shell over the RDD dataset
 - Process's stdin is used to receive the RDD dataset
 - Process's stdout is used to send the results back
- Allows developers to use languages other than Scala, Python, or Java to effect a transformation

RDD pipe Transformation (2/2)

```
[student@localhost Data] $ cat repeat.sh
#!/bin/bash
While read LINE; do
    echo ${LINE}
done
```

Cmd

```
data_src = ["Having", "fun", "learning", "Spark"]
script_path = "/home/student/Data/repeat.sh"
myRDD = sc.parallelize(data_src)
pipeRDD = myRDD.pipe(script_path)
print(pipeRDD.collect())
```

Input

```
['Having', 'fun', 'learning', 'Spark']
```

Output

Unit 1.

Unstructured Data Processing

- | 1.1. Introduction to Apache Spark
- | 1.2. Python Basics
- | 1.3. Data Transformation with Core API
- | 1.4. Working with Pair RDDs

What is a Pair RDD

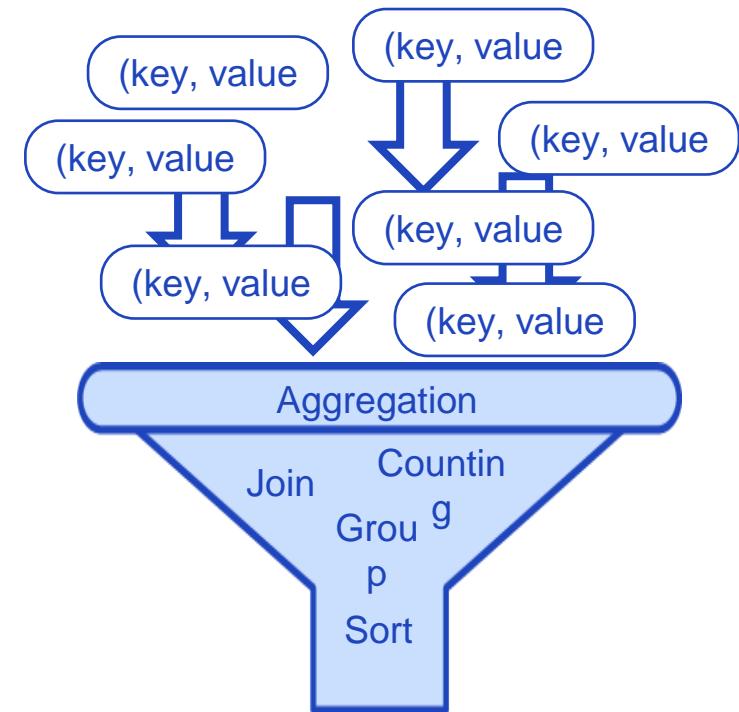
- Spark defines `PairRDDFunctions` class that are specialized to work with Pair RDDs
- A pair RDD is an RDD where all the elements are in a key-value pair tuple
 - Tuples are created with parenthesis in Python
 - A Key-Value pair tuple is a tuple with 2 elements
 - The left side is the Key and the right side is the Value
 - The elements of a tuple can be any primitive, or complex data type
 - Tuples are immutable types
- Key-Value pair is a very common data type
 - The data type used in Map Reduce algorithms

```
key = "Name"  
value = "Jonathan"  
key_pair_tuple = (key, value)
```

Input

What can be done with Pair RDD?

- The Key-Value data structure exposes many operations
 - Counting number of keys
 - Aggregating the value of items with the same key
 - Sorting and grouping by key
 - Grouping and joining two RDDs based on the same key



Creating Pair RDD

- As with other RDD operations the first step required is to create a Pair RDD
 - `map()`, `flatMap()`, `keyBy()`, `flatMapValues()` are useful transformations for creating Pair RDD
 - We are already familiar with `map()` and `flatMap()`

Input

```
data_src = "alice_excerpts"
pairRDD = sc.textFile(data_src) \
    .flatMap(lambda line: line.split(' ')) \
    .map(lambda word: (word, 1))
for i in pairRDD.take(5):
    print(i)
```

Output

```
('Mary', 1)
('Ann!', 1)
('Mary', 1)
('Ann!"', 1)
('said', 1)
```

Creating Pair RDD with keyBy(func)

- keyBy(func) expects a function parameter func to describe how to find the key
- The value will be the original contents of the parent RDD

Input

```
my_fruits = ["apples", "oranges", "pear", "banana"]
pairRDD = sc.parallelize(my_fruits) \
    .keyBy(lambda fruit: len(fruit))

for i in pairRDD.collect():
    print(i)
```

Output

```
(6, 'apples')
(7, 'oranges')
(4, 'pear')
(6, 'banana')
```

Using map() instead of keyBy()

- One of the issues with **keyBy()** is that the transformation is not transparent
 - Developers have to understand how the **keyBy()** protocol works
 - This implementation using **map()** on the other hand is very transparent and preferred by developers

Input

```
my_fruits = ["apples", "oranges", "pear", "banana"]
pairRDD = sc.parallelize(my_fruits) \
    .map(lambda fruit: (len(fruit), fruit))

for i in pairRDD.collect():
    print(i)
```

Output

```
(6, 'apples')
(7, 'oranges')
(4, 'pear')
(6, 'banana')
```

Using flatMapValues(func)

- How this transformation works can be deciphered from the name of the method
 - There is a flatMap so we expect func to produce a collection that can be "flattened"
 - The Values says, the input element is expected to be in (key,value) format and func is to be applied to the value portion of the pair tuple
 - The original value has been "flattened" into several elements.

Input

```
fav_fruits = [("Henry", "apples grapes banana"),
              ("Shaun", "watermelon strawberry"),
              ("Sharon", "pear apples kiwi")]

pairRDD = sc.parallelize(fav_fruits) \
    .flatMapValues(lambda fruits: fruits.split(" "))
for i in pairRDD.collect():
    print(i)
```

Output

```
('Henry', 'apples')
('Henry', 'grapes')
('Henry', 'banana')
('Shaun', 'watermelon')
('Shaun', 'strawberry')
('Sharon', 'pear')
('Sharon', 'apples')
('Sharon', 'kiwi')
```

Let's see that is slow motion

- The input will be an key-value pair

```
("Henry", "apples grapes banana")
```

- Apply func on the value portion of the key-value pair

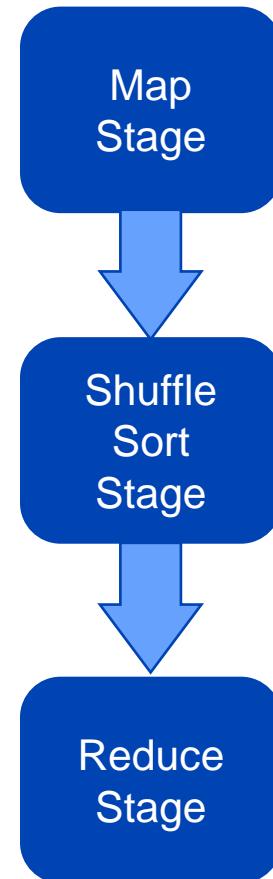
```
("Henry", ['apples', 'grapes', 'banana'])
```

- ```
("Henry", 'apples', 'grapes', 'banana')
```

- ```
("Henry", 'apples')
("Henry", 'grapes')
("Henry", 'banana')
```

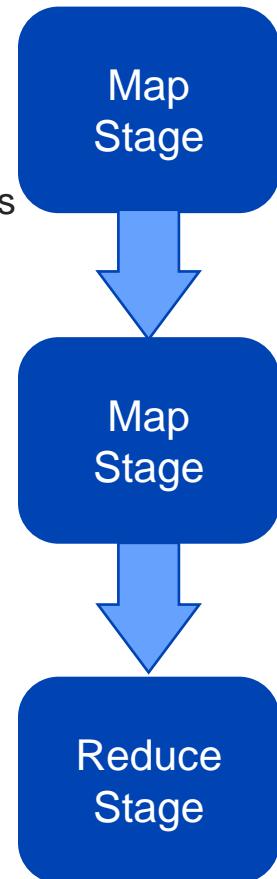
MapReduce Review

- Map Reduce is the original Hadoop distributed parallel processing paradigm
- Data is formatted into Key-Value pairs and partitioned across multiple mappers
- During the Map stage, the Key-Value pair is transformed.
 - Each mapper can work on its portion of the data independently of other mappers
- After the Map stage, the data is sorted and shuffled to the next stage Reducers
- During the Reduce stage, each reducer typically aggregates the value portion of the data
 - Each reducer can work on its portion of the data independently of other reducers
- The entire Map-Shuffle Sort-Reduce cycle is repeated in sequence multiple times to produce desired output



MapReduce on Spark

- We can also utilize the MapReduce distributed parallel processing paradigm
- Apache Spark allows us to be much more efficient with 100x or more performance gains
- We have already seen the performance gain due to in-memory processing
- In addition, in Spark, we no longer need to follow the map-shuffle sort-reduce cycle
 - Any combination of map and reduce cycles
 - Shuffle sort occurs when we transition from map to reduce cycles only
- Spark transformations that provide mapper capability
 - **map()**, **flatMap()**, **filter()**, etc.
- Spark transformations that provide reducer capability
 - **groupByKey**, **sortByKey**, **reduceByKey**, **aggrgateByKey**, **join**, **countByKey**, etc.



Pair RDD Transformations

Transformation	Meaning
reduceByKey(func)	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function func, which must be of type (V,V) => V.
aggregateByKey(zeroValue) (seqOp, combOp)	When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations.
groupByKey()	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Effective groups all the values of the same key
sortByKey([ascending])	When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument.
join(otherDataset)	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key. Outer joins are supported through leftOuterJoin, rightOuterJoin, and fullOuterJoin.

Word Count in Spark (1/4)

- Read the source file

Input

```
data_src = "alice_clean"
wcRDD = sc.textFile(data_src)

for i in wcRDD.take(5):
    print(i)
```

Output

Mary Ann Mary Ann said the voice Fetch me my gloves this moment
Then came a little patterning of feet on the stairs Alice knew it was
the Rabbit coming to look for her and she trembled till she shook the
house quite forgetting that she was now about a thousand times as
large as the Rabbit and had no reason to be afraid of it

Word Count in Spark (2/4)

- Vectorize each word from the data source

Input

```
data_src = "alice_clean"
wcRDD = sc.textFile(data_src) \
    .flatMap(lambda line: line.split(" "))
for i in wcRDD.take(5):
    print(i)
```

Output

```
Mary
Ann
Mary
Ann
said
```

Word Count in Spark (3/4)

- Create the Key-Pair RDD where the Key is each word and the value is the constant 1

Input

```
data_src = "alice_clean"
wcRDD = sc.textFile(data_src) \
    .flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1))

for i in wcRDD.take(5):
    print(i)
```

Output

```
('Mary', 1)
('Ann', 1)
('Mary', 1)
('Ann', 1)
('said', 1)
```

Word Count in Spark (4/4)

- Aggregate all values with the same key by adding them
 - Use **reduceByKey**

Input

```
data_src = "alice_clean"
wcRDD = sc.textFile(data_src) \
    .flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda l, r: l + r)
for i in wcRDD.take(5):
    print(i)
```

Output

```
('said', 63)
(' ', 324)
('before', 3)
('The', 14)
('two', 7)
```

AggregateByKey Transformation (1/4)

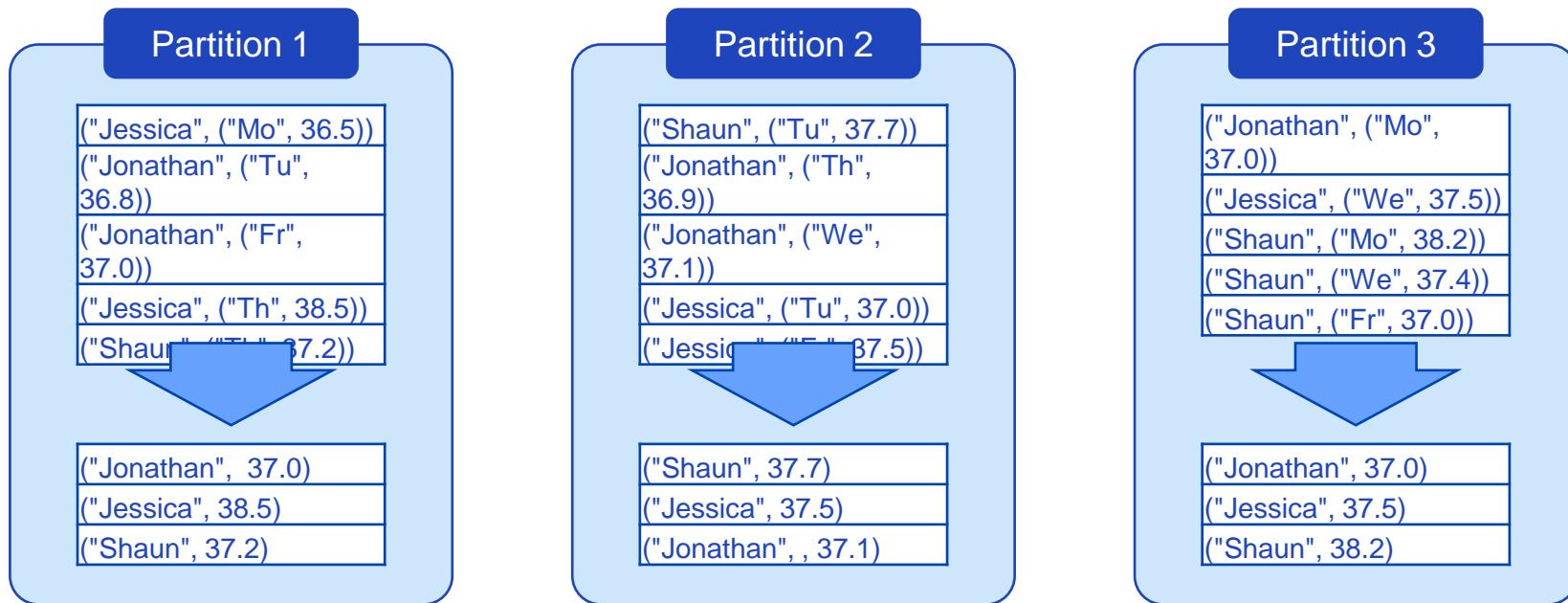
- **aggregateByKey** takes 3 arguments - zeroValue, seqOp, and combOp
- Data input and output format
 - Data input to the transformation is of form (Key of type [K], Value of type [V])
 - Data returned after transformation is of form (Key of type [K], Value of type [U])
 - Example: Input ("Jessica", ("Monday", 36.5)) returns Output ("Jessica", 36.5)
- Data is aggregated over 2 steps, once at each partition, and once over all the data set
- seqOp uses an accumulator to aggregate the value at the Partition level
 - zeroValue is the starting value of this accumulator
- combOp takes the aggregated values in each partition and aggregates them over the entire data set
 - This requires that all the partitions share each others data
 - Since seqOp has already aggregated the Value portion from [V] to [U], combOp should be a function that takes input of form [U] and aggregates output of form [U]

AggregateByKey Transformation (2/4)

- zeroValue - This is the initial value of the accumulator
- Initial value or Zero value
 - It can be 0 if aggregation is sum type
 - Double.MaxValue or known minimum value if aggregation objective is to find minimum
 - Double.MinValue or known maximum value if aggregation objective is to find maximum
 - Set to an empty List, if we just want a collection as output

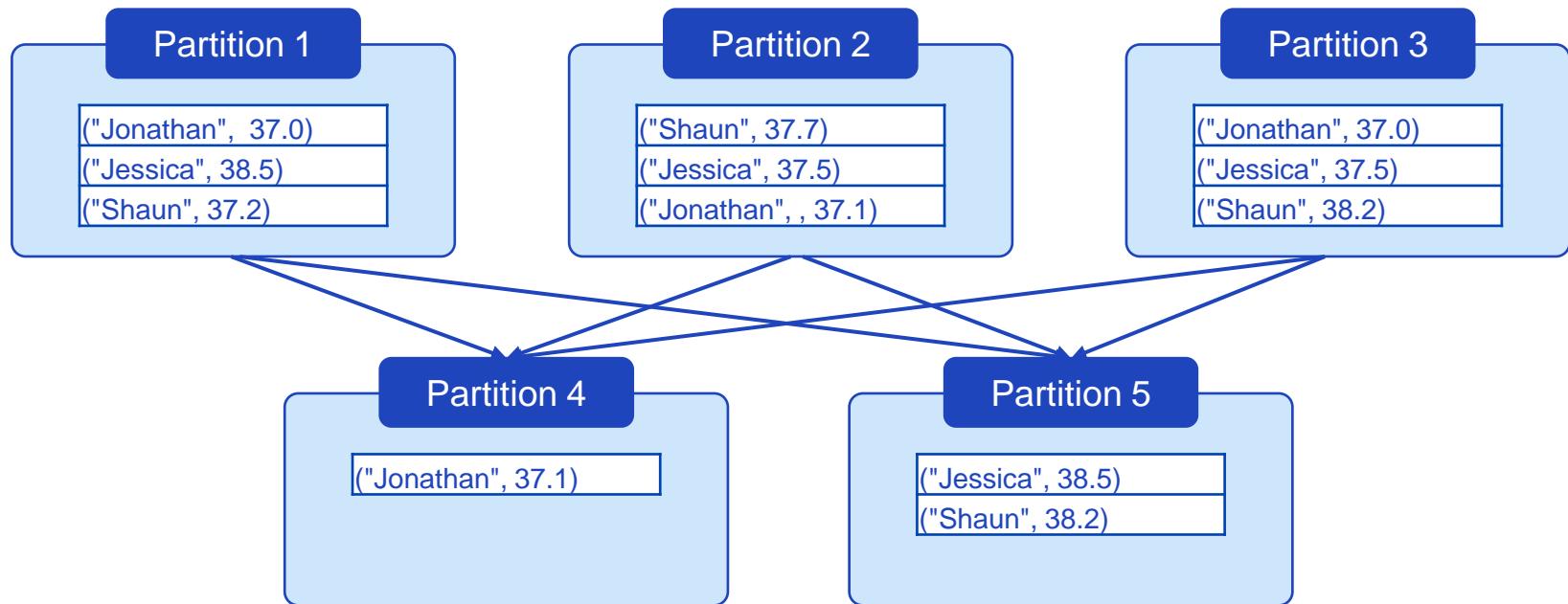
AggregateByKey Transformation (3/4)

- seqOp - Function that aggregates and converts value type [V] to type [U]
- Performs aggregation on the Partition level



AggregateByKey Transformation (4/4)

- combOp - Function that aggregates or merges multiple type [U] elements to a single type [U] element
 - Aggregates all the [U] value for each key into a single



AggregateByKey Example

- The data source and parameter settings

Input

```
data_src = [("Jessica", "Mo", 36.5), ("Shaun", "Tu", 37.7), ("Jonathan", "Mo", 37.0),
            ("Jonathan", "Tu", 36.8), ("Jonathan", "Th", 36.9), ("Jessica", "We", 37.5),
            ("Jonathan", "Fr", 37.0), ("Jonathan", "We", 37.1), ("Shaun", "Mo", 38.2),
            ("Jessica", "Th", 38.5), ("Jessica", "Tu", 37.0), ("Shaun", "We", 37.4),
            ("Shaun", "Th", 37.2), ("Jessica", "Fr", 37.5), ("Shaun", "Fr", 37.0)]
zeroValue = 0
def seqOp(accumulator, element):
    if(accumulator > element[1]):
        return accumulator
    else:
        return element[1]
def combOp(accumulator1, accumulator2):
    if(accumulator1 > accumulator2):
        return accumulator1
    else:
        return accumulator2
```

AggregateByKey Example

- Executing **aggregateByKey** and output results

```
aggRDD = sc.parallelize(data_src) \
    .map(lambda tup: (tup[0], (tup[1], tup[2]))) \
    .aggregateByKey(zeroValue, seqOp, combOp)

for i in aggRDD.collect():
    print(i)
```

Input

```
('Jessica', 38.5)
('Jonathan', 37.1)
('Shaun', 38.2)
```

Output

What result will this code produce?

- The zeroValue starting value has been changed at an tuple of (string, float)
- seqOp and combOp have been modified slightly

```
zeroValue = ("", 0.0)
def seqOp(accumulator, element):
    if(accumulator[1] > element[1]):
        return accumulator
    else:
        return element

def combOp(accumulator1, accumulator2):
    if(accumulator1[1] > accumulator2[1]):
        return accumulator1
    else:
        return accumulator2
```

Input

Did you guess correctly?

- Instead of returning the highest temperature, it returns the highest temperature and the day that it occurred

```
aggRDD = sc.parallelize(data_src) \
    .map(lambda tup: (tup[0], (tup[1], tup[2]))) \
    .aggregateByKey(zeroValue, seqOp, combOp)

for i in aggRDD.collect():
    print(i)
```

Input

```
('Jessica', ('Th', 38.5))
('Jonathan', ('We', 37.1))
('Shaun', ('Mo', 38.2))
```

Output

GroupByKey Transformation

- **groupByKey** aggregates all the values with the same key, and produces a collection of values
 - Input ☐ (Key, Value)
 - Output ☐ (Key, [Value1, Value2, Value3, . . .])
- Technically, the Iterable datatype is produced for the Value collection

Input	Output																		
<pre>ate_fruit = [('Henry', 'apples'), ('Shaun', 'strawberry'), ('Sharon', 'apples'), ('Shaun', 'watermelon'), ('Sharon', 'pear'), ('Henry', 'banana'), ('Henry', 'grapes'), ('Sharon', 'kiwi'), ("Shaun", 'peach')] fav_fruits = sc.parallelize(ate_fruit) \ .groupByKey() for (name, fruits) in fav_fruits.collect(): print(name) for fruit in fruits: print(' ', fruit)</pre>	<table><tbody><tr><td>Sharon</td><td>apples</td></tr><tr><td></td><td>pear</td></tr><tr><td></td><td>kiwi</td></tr><tr><td>Henry</td><td>apples</td></tr><tr><td></td><td>banana</td></tr><tr><td></td><td>grapes</td></tr><tr><td>Shaun</td><td>strawberry</td></tr><tr><td></td><td>watermelon</td></tr><tr><td></td><td>peach</td></tr></tbody></table>	Sharon	apples		pear		kiwi	Henry	apples		banana		grapes	Shaun	strawberry		watermelon		peach
Sharon	apples																		
	pear																		
	kiwi																		
Henry	apples																		
	banana																		
	grapes																		
Shaun	strawberry																		
	watermelon																		
	peach																		

Join Transformation

- The **join** transformation aggregates all the values with the same Key and create a tuple of Values
 - Input \square (Key, Value)

```
Output  $\square$  (Key, (Value1, Value2))
```

```
gen = [('Henry', 'M'), ('Henry', 'F'), ('Jessica', 'F'), ('Jonathan', 'M'), ('Shaun', 'M')]  
age = [('Henry', 38), ('Henry', 16), ('Jessica', 22), ('Jonathan', 18), ('Shaun', 52)]  
  
gen_rdd = sc.parallelize(gen)  
age_rdd = sc.parallelize(age)  
person_rdd = gen_rdd.join(age_rdd)  
for i in person_rdd.collect():  
    print(i)
```

Input

```
Output
```

```
('Jessica', ('F', 22))  
('Jonathan', ('M', 18))  
('Henry', ('M', 38))  
('Henry', ('M', 16))  
('Henry', ('F', 38))  
('Henry', ('F', 16))  
('Shaun', ('M', 52))
```

[Lab1]

Working with the PySpark Shell



[Lab2] Basic Python



[Lab3]

Working with Core API Transformations



[Lab4]

Working with Pair RDDs



[Lab5]

Putting it all together



Unit 2

Structured Data Processing

Big Data Processing with Apache Spark

Unit 2.

Structured Data Processing

- | 2.1. Introduction to Spark SQL
- | 2.2. Spark SQL Operations
- | 2.3. Interoperating RDDs and DataFrames

Apache Spark SQL

- Spark SQL is a Spark module for structured data processing
- Unlike basic Spark RDD API, the interfaces provide more information about the structure of the data and computation being performed
- Internally, Spark SQL uses this extra information to perform optimizations through the Catalyst Optimizer
 - The Catalyst Optimizer analyzes the DataFrame transformations and creates optimized code
- There are several ways of interacting with Spark SQL
 - Using ISO standard SQL queries
 - Through the use of the DataFrame and Datasets API
 - Both methods use the same execution engine, behind the scenes, independent of which API or language was used
 - Allows developers to create applications in the environment and platform that works best for them

Spark Session

- The entry point for Apache Spark DataFrame API is the `SparkSession` object
- In versions prior to 2.x, there are two separate objects used as the entrypoint for DataFrames API
 - `SQLContext` provided basic SQL functionality
 - `HiveContext` added interacting with Apache Hive
- Two different versions with different functionality created a lot of confusion in the Spark community
- Starting with 2.x Apache Spark has unified this to the single `SparkSession` object
 - This new object combines the functionality of both `SQLContext` and `HiveContext`
 - Includes built-in support for writing Hive queries (`HiveQL`)
 - Access to Hive user-defined functions (`Hive UDF`)
 - Ability to read data from Hive tables

Spark DataFrames and Datasets

- DataFrames and Datasets are used to represent structured data in Apache Spark
 - Datasets is only available in Scala and Java
- The easiest way to think of DataFrames and Datasets is as data that has column and rows
 - Data has schema
 - Columns have column names and column types
 - Each row of data represents an item of information
 - Similar to tables in relational databases

	Col1 ^e	Col2	Col3	...
Row1				
Row2				
Row3				
.				

Apache Spark Datasets

- A Dataset is a distributed collection of data
 - Available as of Apache Spark 1.6
 - Provides the benefits of RDD such as strong typing
 - Ability to use powerful lambda functions
 - Adds the benefits of Spark SQL's optimized execution engine (Catalyst Optimizer)
- A Dataset can be constructed from JVM objects (Scala or Java)
 - Each JVM object row in a Dataset is strongly typed and enforced at compile time
 - Scala case classes are an easy way to create JVM objects
 - Case classes provide a quick and easy method for creating objects with schema
- Datasets can be manipulated using functional transformations
 - map(), flatMap(), filter(), etc.
- Dataset API is available only for Scala and Java

Apache Spark DataFrames

- Difference between DataFrames and Datasets is how data is represented
 - Each row in a DataFrame is a loose collection of dynamically typed **Row** object
 - DataFrames can only contain Row objects
 - Each row in a Dataset is strongly typed JVM object
- Each **Row** object contains an ordered collection of values
 - Basic primitive types such as integers, floats, Boolean
 - Collections such as Strings, Lists, Arrays, etc.
- Conceptually equivalent to a table in a relational database
 - Similar to an R or Python dataframe
- DataFrames benefit from the use of the Apache Spaks Catalyst Optimizer
 - Produces highly optimized code

Spark Session

- The `SparkSession` class provides functions and attributes to access all Spark functionality
- Most Spark functionality is accessed as a method of the `SparkSession`
 - `sql`: execute a Spark SQL query
 - `catalog`: entry point for the Catalog API for managing tables
 - `read`: function to read data from a file or other data source
 - `conf`: object to manage Spark configuration settings
 - `sparkContext`: entry point for core Spark API
- The Spark Shell provides this object under the variable name "**spark**"

Creating Spark DataFrame

- With a SparkSession, applications can create DataFrames from:
 - From code and data created in memory
 - Existing RDD through transformations
 - From Hive Table
 - Spark data sources
- Spark SQL supports a wide range of data source type and formats
 - Text files such as CSV, TSV, JSON, XML, or plain text
 - Binary format files
 - Apache Parquet - Default format of Spark
 - Apache ORC - Default format of Hive
 - Apache Avro - Extensively used for data serialization/deserialization
 - Public Cloud Sources
 - Amazon S3
 - Microsoft Azure Data Lake Storage Gen2

Hadoop Binary File Formats

- Apache Parquet
 - Optimized, binary, columnar storage of structured data
- Apache Avro Data Format
 - Apache Avro is a data serialization system
 - Stores data in a compact, structured, binary, row-oriented format
- Apache ORC
 - Optimized for both column-based and record-based access, specific to the Hadoop ecosystem
- SequenceFile Format
 - The original Hadoop binary storage format

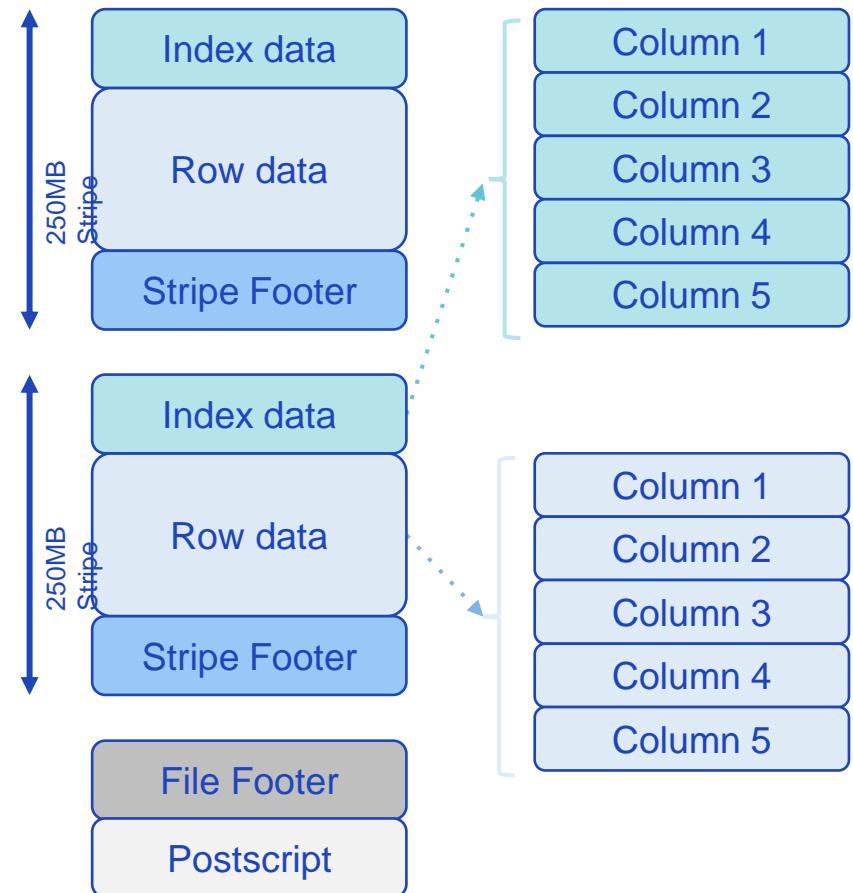
Apache Parquet

- Parquet is a very common storage format
 - Supported by Sqoop, Spark, Hive, Impala, and other tools
- Key Features
 - Optimized binary storage of structured data
 - Schema metadata is embedded in the file
 - Efficient performance and size for large amounts of data
 - Parquet works well with Impala
- Use parquet-tools to view Parquet file schema and data
 - Use show to display the records

```
$ parquet-tools show /path/to/parquet/file  
$ parquet-tools inspect /path/to/parquet/file
```

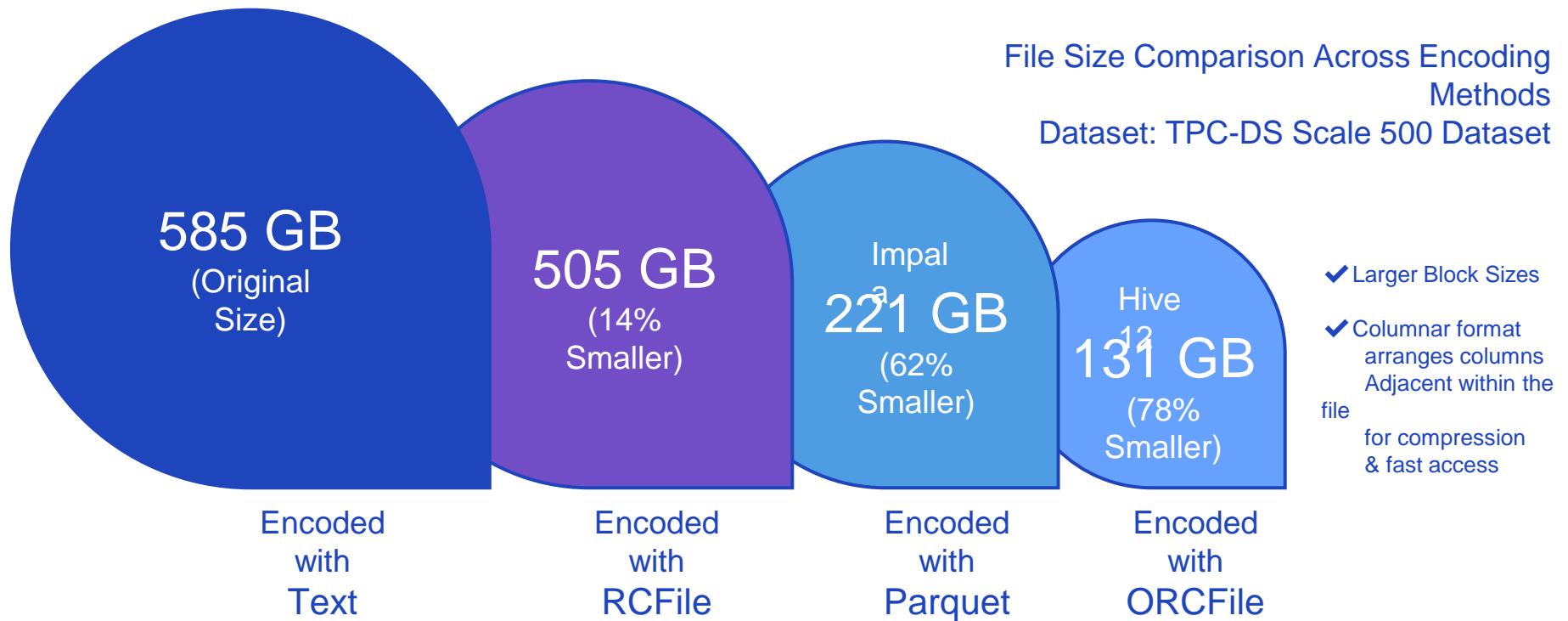
ORC File Format

- ORC stands for Optimized Record Columnar
- ORC is a row columnar data format
- Highly optimized for reading, writing, and processing data in Hive
- ORC files are made from stripes of data
- Each stripe contains index, row data, and footer
- Key statistics (count, max, min, sum) are cached



File Size Comparison

- While ORC files are smaller, Parquet files are more efficient for data processing



Generic Load and Save Functions

- Loading and saving a Dataframe using the default Parquet format

```
df = spark.read.load("file:/home/student/Data/resources/users.parquet")
df.select("name", "favorite_color").write.save("namesAndFavColors.parquet")
```

Generic Load and Save Functions

- By default, Apache Spark compresses Parquet format files using Snappy compression
 - This can be seen from Hue, where the file name is suffixed with `***.snappy.parquet`

The screenshot shows a file browser interface with the following details:

Home / user / student / **namesAndFavColors.parquet**

	Name	Size	User
<input type="checkbox"/>	..		student
<input type="checkbox"/>	.		student
<input type="checkbox"/>	_SUCCESS	0 bytes	student
<input type="checkbox"/>	part-00000-3e3438ba-a233-44fc-bd82-b10c02d0e3eb-c000.snappy.parquet	653 bytes	student

Generic Load and Save Functions

- The content of Parquet formatted files can not be viewed directly from Hue, unlike text files.
- Use parquet-tools from the command line
 - Use hdfs dfs -get <name of parquet file>
 - parquet-tools show <name of parquet file> to view content

```
[student@localhost Data]$ hdfs dfs -get namesAndFavColors.parquet/part*
[student@localhost Data]$ ls
alice_in_wonderland.txt
anonymous-msweb.data
part-00000-3e3438ba-a233-44fc-bd82-b10c02d0e3eb-c000.snappy.parquet
pig_data1.txt
pig_data2.txt
resources
scripts
[student@localhost Data]$ parquet-tools show part-00000-3e3438ba-a233-44fc-bd82-
b10c02d0e3eb-c000.snappy.parquet
+-----+
| name    | favorite_color |
+-----+
| Alyssa  |
| Ben     | red           |
+-----+
```

Manually Specifying Format

- In previous example, we did not specify a format because default parquet format was used
- We can specifically specify the format of the source file and the format to save it to
 - Use SparkSession's read-load combination to read source file
 - Use SparkSession's write-load combination to save a file
 - Specify format as a parameter

```
SparkSession.read.format(<format>).load(<path>)
SparkSession.write.format(<format>).save(<path>)
```

```
peopleDF =
spark.read.format("json").load("file:/home/student/Data/resources/people.json")
peopleDF.select("name",
"age").write.format("parquet").save("namesAndAges.parquet")
```

Result of Previous Load and Save

```
[student@localhost Data]$ cat resources/people.json
```

```
{“name” : “Michael”}
{“name” : “Andy”, “age”:30}
{“name” : “Justin”, “age”:19}
```

Input

Home / user / student / namesAndAges.parquet

	Name	Size	User
□			
□			student
□			student
□	_SUCCESS	0 bytes	student
□	part-00000-4fd79ba4-ab63-4f08-91e2-de77139ff9e9-c000.snappy.parquet	687 bytes	student

```
[student@localhost Data]$ parquet-tools showde77139ff9e9-c000.snappy.parquet
```

name	age
Michael	nan
Andy	30
Justin	19

Using Options with Read-Load

- For each type of data source, there are application options that can be specified
- Here, for a CSV file format:
 - Specify the delimiter with "sep"
 - Specify whether Spark should infer the schema
 - Specify if the first row contains a header row

```
peopleDFcsv = spark.read.format("csv") \
    .option("sep", ";") \
    .option("inferSchema", "true") \
    .option("header", "true") \
    .load("file:/home/student/Data/resources/people.csv")
```

Displaying DataFrame Schema

- Once the DataFrame has been created, we can view the schema of the DataFrame
 - Use the `printSchema()` action

```
peopleDFcsv = spark.read.format("csv") \
    .option("sep", ",") \
    .option("inferSchema", "true") \
    .option("header", "true") \
    .load("file:/home/student/Data/resources/people.csv")
```

Input

```
peopleDFcsv.printSchema()
```

Input

```
root
 |-- name: string (nullable = true)
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
```

Output

Displaying Contents of DataFrame

- We can also view the contents of the DataFrame
 - Use `.show(n, <truncate>)` action

```
peopleDFcsv = spark.read.format("csv") \
    .option("sep", ",") \
    .option("inferSchema", "true") \
    .option("header", "true") \
    .load("file:/home/student/Data/resources/people.csv")
```

Input

```
peopleDFcsv.show()
```

Input

```
+----+----+-----+
| name | age |      job |
+----+----+-----+
| Jorge | 30 | Developer |
| Bob   | 32 | Developer |
+----+----+-----+
```

Output

Using Options with Write-Save

- Just as with read-load, we can specify options for write-save
- Save the file in Orc format with:
 - Set the bloom filters

```
root
| -- name: string (nullable = true)
| -- favorite_color: string (nullable = true)
| -- favorite_numbers: array (nullable = true)
|   | -- element: integer (containsNull = true)
```

Input

```
usersDF = spark.read.orc("file:/home/student/Data/resources/users.orc")
usersDF.write.format("orc") \
    .option("orc.bloom.filter.columns", "favorite_color") \
    .option("orc.dictionary.key.threshold", "1.0") \
    .option("orc.column.encoding.direct", "name") \
    .save("users_with_options.orc")
```

Output

DataFrame Save Modes

- Save operations can optionally take a mode that specifies how to handle existing data if present

mode	Meaning
error	Default mode. When saving a DataFrame to a data source, if data already exists, an exception is expected to be thrown.
append	When saving a DataFrame to a data source, if data/table already exists, contents of the DataFrame are expected to be appended to existing data.
overwrite	Overwrite mode means that when saving a DataFrame to a data source, if data/table already exists, existing data is expected to be overwritten by the contents of the DataFrame.
ignore	Ignore mode means that when saving a DataFrame to a data source, if data already exists, the save operation is expected not to save the contents of the DataFrame and not to change the existing data. This is similar to a CREATE TABLE IF NOT EXISTS in SQL.

Spark Data Compression Options

- Compression allows us to reduce the size of the stored data
- Trade-off between reducing I/O bandwidth and CPU time to decompress data
 - Often reducing the size aggressively means more compute time and less aggressive is larger data size
- Many Hadoop jobs are I/O bound and can improve performance greatly
 - As we move more towards AI/ML, access pattern is becoming iterative, and compute bound
- GZip is more aggressive but higher compression
 - Good for cold storage
- Snappy is the best choice for frequently accessed data
- BZip2 can achieve even greater compression than Gzip for certain types of files

Setting the Compression Option

- DataFrames may be saved with compression options
- The syntax is slightly different for the version of Apache Spark

- Spark 2.2+

```
df.write.option("compression", "<compression>").save(<path>)
```

- ```
df.write..save(<path>, compression=<compression>)
```
- ```
df.write..codec(<compression>).save(<path>)
```

Using File Format Shortcuts

- Apache Spark provides file format shortcuts for both the read-load and write-save combinations
- Supported formats:
 - JSON
 - CSV
 - ORC

```
peopleDF = spark.read.<file format>(<path>)
spark.write.option(<opt>). <file format>(<path>)

# CSV Example
peopleDF = spark.read.csv(<path>)
spark.write.option(<opt>).csv(<path>)
```

Reading Hive Tables

- Apache Hive provides database-like access to data stored in HDFS
 - The schema and other metadata information is stored in the Hive metastore
 - The Hive metastore is typically a RDBMS such as MySQL / MariaDB
 - The actual content is stored in HDFS
- Spark SQL supports reading and writing data stored in Apache Hive
 - Spark must have the configuration settings for Hive and HDFS in its configuration directory
 - Spark Sessions must be initialized with Hive support enabled
 - Hive warehouse location, the default location where Hive tables are stored in HDFS

```
tableDF = spark.read.table(<table_name>)
```

Saving to Hive Tables

- Spark DataFrames can be saved to Hive using the saveAsTable command
- Table will be saved as a Managed table and data will be stored in the Hive Warehouse directory

```
au.write.saveAsTable("authors_managed")
spark.sql("desc formatted authors_managed").show(30, False)
```

Col_name	Data_type	comment	Input	Output
Id	Int	Null		
First_name	String	Null		
# Detailed Table Information				
Database	Default			
Table	Authors_managed			
Owner	Student			
Created time	Mon Aug 30 01:45:25 KST 2021			
Last Access	UNKNOWN			
Created by	Spark 3.1.2			
Type	MANAGED			
Provider	Parquet			
Location	hdfs://localhost:9000/user/hive/warehouse/authors_managed			
Serde Library	org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe			
InputFormat	org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat			
OutputFormat	org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat			

Saving to External Hive Tables

- To Store as an External table, give **path** option
 - Dropping external tables only removes the metadata

InputOutput

```
au.write.option("path","/user/student/authors_external").saveAsTable("authors_external")
spark.sql("desc formatted authors_external").show(30, False)
```

Col_name	Data_type	comment
Id	Int	Null
First_name	String	Null
# Detailed Table Information		
Database	Default	
Table	Authors_external	
Owner	Student	
Created time	Mon Aug 30 02:19:57 KST 2021	
Last Access	UNKNOWN	
Created by	Spark 3.1.2	
Type	EXTERNAL	
Provider	Parquet	
Statistics	889 bytes	
Location	hdfs://localhost:9000/user/student/authors_external	
Serde Library	org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe	
InputFormat	org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat	
OutputFormat	org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat	

Creating DataFrame from a Collection

- Use `createDataFrame(<collection of row elements>, <collection of column names>)`
- Each element in the List becomes a row in the DataFrame
 - Use Nested Lists or Tuple for each column item
 - Provide names for the columns as comma delimited collection

Input

```
my_list = [(1, "Henry", 23.00),  
           (2, "Shaun", 15.25),  
           (3, "Sharon", 45.50),  
           (4, "Jessica", 20.00)]  
my_schema = ("id", "name", "amt")  
myDF = spark.createDataFrame(my_list,  
                           my_schema)  
myDF.printSchema()  
myDF.show()
```

root

```
-- id: long (nullable = true)  
-- name: string (nullable =  
true)  
-- amt: double (nullable = true)  
+---+---+---+  
| id| name| amt|  
+---+---+---+  
| 1| Henry| 23.0|  
| 2| Shaun| 15.25|  
| 3| Sharon| 45.5|  
| 4| Jessica| 20.0|  
+---+---+---+
```

Output

Provide Explicit Schema Information

- Use `createDataFrame(<collection of elements., <comma delimited schema string>)`
- Each element in the List becomes a row in the DataFrame
 - Use Nested Lists or Tuple for each column item
- Provide the schema information

```
my_list = [(1, "Henry", 23.00),  
           (2, "Shaun", 15.25),  
           (3, "Sharon", 45.50),  
           (4, "Jessica", 20.00)]  
  
my_schema = "id string, name string, amt  
double"  
  
myDF = spark.createDataFrame(my_list,  
                           my_schema)  
myDF.printSchema()  
myDF.show()
```

Input

```
root  
|-- id: string (nullable = true)  
|-- name: string (nullable =  
true)  
|-- amt: double (nullable = true)  
  
+---+---+---+  
| id| name| amt|  
+---+---+---+  
| 1| Henry| 23.0|  
| 2| Shaun| 15.25|  
| 3| Sharon| 45.5|  
| 4| Jessica| 20.0|  
+---+---+---+
```

Output

Unit 2.

Structured Data Processing

- | 2.1. Introduction to Spark SQL
- | 2.2. Spark SQL Operations
- | 2.3. Interoperating RDDs and DataFrames

Spark SQL Operations

- Similar to the Core API, Spark SQL operations can be categorized
- Transformations:
 - Applies transformations and creates a new Dataset/DataFrame from existing one
- Actions:
 - Returns the result of the Action to the Driver program as output
- Immutable Transformations
 - As in the Core API, DataFrame / Dataset transformations are immutable
 - Each transformation leaves the original intact, and create a new DataFrame / Dataset

Spark SQL Actions

- We have already seen a couple of Spark SQL Actions
 - **printSchema()** - return the schema information of the DataFrame / Dataset
 - **show(n)** - print the first n rows of the DataFrame / Dataset in a nice formatted output
 - **write** - Save the DataFrame / Dataset in a specified location
- Some other commonly used Actions are:
 - **take(n)** - return the first n row elements of the DataFrame / Dataset
 - The **take(n)** action returns the actual elements.
 - In the case of DataFrames, this would be Row objects
 - In the case of DataSets, this would typically be the case-class objects
 - **count()** - return the number of rows in the DataFrame / Dataset
 - **first()** - return the first element of the DataFrame / Dataset
 - **collect** - return all the elements of the DataFrame / Dataset

Spark SQL take(n) Action

- Let's compare the **show(n)** action with the **take(n)** action

```
peopleDFcsv = spark.read.format("csv") \
    .option("sep", ",") \
    .option("inferSchema", "true") \
    .option("header", "true") \
    .load("file:/home/student/Data/resources/people.csv")
```

Input

peopleDFcsv.take(2)

Input

```
[Row(name='Jorge', age=30,
      job='Developer'),
 Row(name='Bob', age=32,
      job='Developer')]
```

Output

peopleDFcsv.show(2)

Input

Name	Age	Job
Jorge	30	Developer
Bob	32	Developer

Output

Spark SQL take(n) and first()

- The **take(n)** action returns a sequence or collection of size n
 - **take(1)** returns a sequence or collection of size 1
- **first()** action returns the first element

```
peopleDFcsv = spark.read.format("csv") \
    .option("sep", ",") \
    .option("inferSchema", "true") \
    .option("header", "true") \
    .load("file:/home/student/Data/resources/people.csv")
peopleDFcsv.take(1)
```

Input

```
[Row(name='Jorge', age=30, job='Developer')]
```

Output

```
peopleDFcsv.first()
```

Input

```
Row(name='Jorge', age=30, job='Developer')
```

Output

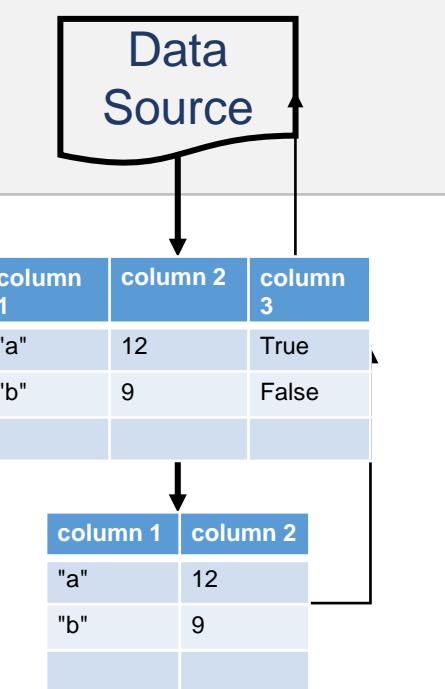
Lazy and Eager Execution

- Lazy Transformation:
 - As in Core API, transformations are not applied immediately
 - An action triggers all dependent Datasets/Dataframes to be actualized
 - As in RDDs, Spark tracks the lineage for each DataFrame / Dataset and follows the DAG dependency graph to actualize the dependent DataFrames / Datasets
- However, schema is eagerly executed.
 - You may notice that Spark takes some time after the read transformation
 - Spark scans the data source in order to determine the schema
 - Does not actually read the entire content

Lazy and Eager Pipeline Execution

- Eager schema operation by lazy content operation

```
tableDF = spark.read.table("table_name")
filterDF = tableDF.select("column1", "column2")
filterDF.show(2)
```



select() Transformation

- Use **select()** to transform and create a new DF/DS consisting of the selected columns

```
Input  
authorDF = spark.read.table("authors")  
authorDF.printSchema()  
  
Output  
root  
|-- id: integer (nullable = true)  
|-- first_name: string (nullable = true)  
|-- last_name: string (nullable = true)  
|-- email: string (nullable = true)  
|-- birthdate: string (nullable = true)  
|-- added: string (nullable = true)
```

```
Input  
emailDF = authorDF.select("id", "email")  
emailDF.printSchema()  
emailDF.show(5)
```

```
Output  
root  
|-- id: integer (nullable = true)  
|-- email: string (nullable = true)  
+---+-----+  
| id|          email |  
+---+-----+  
| 1 |      barmstrong@ex... |  
| 2 |      hand.stella@ex... |  
| 3 |      darren.blanda@ex... |  
| 4 |      shanahan.aliyah@e |  
| 5 |      bednar.robin@ex... |  
+---+-----+  
only showing top 5 rows
```

where() Transformation

- Filter the rows using the condition string
 - `filter()` and `where()` are equivalent DataFrame / Dataset transformations

Input	Output
<pre>emailDF = authorDF.select("id", "email")) emailDF.printSchema() emailDF.show(5)</pre>	<pre>root -- id: integer (nullable = true) -- email: string (nullable = true) +---+ id email +---+ 1 barmstrong@example... 2 hand.stella@example... 3 darren.blanda@example... 4 shanahan.aliyah@email... 5 bednar.robin@example... +---+ only showing top 5 rows</pre>

Input	Output
<pre>bEmailDF = emailDF.where("id > 10") bEmailDF.show(5)</pre>	<pre>root -- id: integer (nullable = true) -- email: string (nullable = true) +---+ id email +---+ 1 virgil45@example... 2 dgleason@example... 3 Onie.wehner@example... 4 izulauf@example... 5 Marcellle.breitenb@example... +---+ only showing top 5 rows</pre>

limit(n) Spark SQL Transformation

- Limit the number of rows to the first n rows in the DataFrame / Dataset

Input

```
emailDF = authorDF.select("id", "email")
)
emailDF.printSchema()
emailDF.show(5)
```

Output

```
root
 |-- id: integer (nullable = true)
 |-- email: string (nullable = true)
+---+
| id|      email |
+---+
| 1|  barmstrong@ex...
| 2| hand.stella@ex...
| 3| darren.blanda@ex...
| 4| shanahan.aliyah@e...
| 5| bednar.robin@ex...
+---+
only showing top 5 rows
```



Input

```
top2DF = emailDF.limit(2)
top2DF.show(5)
```

Output

```
root
 |-- id: integer (nullable = true)
 |-- email: string (nullable = true)
+---+
| id|      email |
+---+
| 1|  barmstrong@ex...
| 2| hand.stella@ex...
+---+
only showing top 5 rows
```

orderBy() Spark SQL Transformation

- Is this operation seeming expensive?
- `orderBy(<order_by_column>)` sorts the DataFrame / Dataset by the given column

[Discussion]

```
authorDF.orderBy("last_name").show(5)
```

Input

+-----+ <th>id</th> <th>first_name</th> <th>last_name</th> <th>email</th> <th> birthdate </th> <th>added</th> <th> </th>	id	first_name	last_name	email	birthdate	added	
+-----+					+-----+	+-----+	+-----+
2923 Tiara		Abbott	ischaefer@example...	1994-02-24	1983-06-13	21:19:...	
2953 Arnoldo		Abbott	taryn86@example.com	1974-05-28	1975-04-26	11:17:...	
2991 Bettie		Abbott	joyce.torp@example...	1985-05-23	1983-01-10	07:04:...	
3310 Mario		Abbott	monty79@example.net	1986-07-24	1980-06-27	20:14:...	
3574 Vanessa		Abbott	effie.marquardt@e...	2000-08-15	2005-12-27	06:59:...	
+-----+				+-----+	+-----+	+-----+	+-----+

only showing top 5 rows

Output

Using Columns in Transformation

- Most transformation require specifying columns as part of its parameters
- So far, we have used strings to indicate the columns in the transformations
 - `select("column1", "column2")`
 - `where("some string condition")`
 - `orderBy("column")`
- More complex transformations require that method be passed Column objects
- Transformation using Column objects are referred to as column expressions
- Observe the overloaded `select()` and `where()` method

```
def select(col: String, cols: String*) : DataFrame
    Selects a set of columns.
def select(cols: Column*) : DataFrame
    Selects a set of column based expressions.
```

```
def where(conditionExpr: String) : Dataset[T]
    Filters rows using the given SQL
    expression.
def where(condition: Column) : Dataset[T]
    Filters rows using the given condition.
```

PySpark Column Object Reference

- Python has two different notations to refer to Column objects

```
peopleDF = spark.read \
    .json("file:/home/student/Data/resources/people.json")
peopleDF.printSchema()
```

Input

```
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
```

Output

```
peopleDF.select(peopleDF.age, peopleDF['name']).show()
```

Input

```
+----+-----+
| age | name |
+----+-----+
| null | Michael |
| 30 | Andy |
| 19 | Justin |
+----+-----+
```

Output

Scala Column Object Reference

- Scala has three different notations to refer to Column objects

```
val peopleDF = spark.read.  
    json("file:/home/student/Data/resources/people.json")  
peopleDF.printSchema  
peopleDF.select ($"age", peopleDF("name"), 'age').show
```

Input

```
+----+-----+  
| age|    name|age |  
+----+-----+  
|null|Michael|null|  
|  30|    Andy|30  |  
|  19| Justin|19  |  
+----+-----+
```

Output

col() Function

- Both Scala and Python provides the col("column_name") function that returns the Column object of the column passed as a string parameter

Scala

```
peopleDF.  
select(col("age")).  
show
```

Input

```
+---+  
| age|  
+---+  
|null|  
| 30|  
| 19|  
+---+
```

Output

Python

```
peopleDF. \  
select(col("age")). \  
show()
```

Input

```
+---+  
| age|  
+---+  
|null|  
| 30|  
| 19|  
+---+
```

Output

Why Column Expressions?

- Using column expressions instead of simple strings, allows developers to access functions
- Built-in Functions:
 - Scalar Functions - Functions that return a single value per row
 - Aggregate Functions - Functions that return a single value on a group of rows
- User-Defined Function:
 - User-Defined Scalar Functions
 - User-Defined Aggregate Functions
 - Only in Scala and Java

Input

```
peopleDF. \
    select(col("age"), col("age") + 10). \
    show()
```

Output

age	(age + 10)
null	null
30	40
19	29

Commonly Used Scalar Functions/Operators

- Arithmetic Operators
 - +, -, *, /, %
- Comparative Operators
 - == (=== in Scala), >, >=, <, <=, !=, <>,
- Logical Operators
 - &, |, ~ (Python)
 - &&, ||, !, and, or , not (Scala)
- String Operators
 - **substring, replace, contains, like, upper, lower**
- Testing data
 - **isNull, isNotNull, NaN** (not a member)

```
my_list = [[1, "henry park", 23.00],  
           [2, "shaun smith", 15.25],  
           [3, "sharon ramirez", 45.50],  
           [4, "jessica bolt", 20.00]]  
my_schema = ("id", "name", "amt")  
myDF = spark.createDataFrame(my_list, my_schema)  
myDF \  
    .select(upper(myDF.name), "amt") \  
    .where( (col("amt") > 18) & (myDF["amt"] < 40)  
) \  
    .show()
```

Input

```
+-----+----+  
| upper(name) | amt |  
+-----+----+  
| HENRY PARK|23.0 |  
| JESSICA BOLT|20.0 |  
+-----+----+
```

Output

Date and Timestamp Functions

- There are several standards for saving date and time
 - Unix timestamps
 - Date Type
 - SQL Timestamp Type
- Being able to convert between formats as well as specify or modify the format of date and time is an important function

Epoch Unix Timestamp
163050308
8
SECONDS SINCE JAN 01 1970.
(UTC)
10:31:30 PM

Format Date Functions

- There date functions return some portion of a date in requested format

PySpark Date Function	Description
current_date	Returns the current date at the start of query evaluation.
current_timestamp	Returns the current timestamp at the start of query evaluation.
second(timestamp)	Returns the second component of the string/timestamp.
minute(timestamp)	Returns the minute component of the string/timestamp.
hour(timestamp)	Returns the hour component of the string/timestamp.
weekofyear(date)	Returns the week of the year of the given date. A week is considered to start on a Monday and week 1 is the first week with >3 days.
month(date)	Returns the month component of the date/timestamp.
quarter(date)	Returns the quarter of the year for date, in the range 1 to 4.
year(date)	Returns the year component of the date/timestamp.
date_format(timestamp, fmt)	Converts timestamp to a value of string in the format specified by the date format fmt.

Sample Use: Date Format Functions

```
from pyspark.sql.functions import *
authorDF = spark.read.table("authors")
authBdayDF = authorDF.select("first_name", "last_name", \
                             weekofyear(authorDF.birthdate), \
                             date_format(authorDF.added, "y"))
authBdayDF.printSchema()
authBdayDF.show(5)
```

Input

```
root
|-- first_name: string (nullable = true)
|-- last_name: string (nullable = true)
|-- weekofyear(birthdate): integer (nullable = true)
|-- date_format(added, y): string (nullable = true)
+---+---+---+
|first_name| last_name|weekofyear(birthdate)|date_format(added, y)|
+---+---+---+
| Walton| Adams| 9| 1997|
| Marietta| Walsh| 22| 2010|
| Lily| Wintheiser| 34| 1973|
| Estevan| Gleason| 29| 1995|
| Thaddeus| Rowe| 9| 2017|
+---+---+---+
only showing top 5 rows
```

Output

alias Column Function

- Column.alias(alias)
 - Returns the column aliased with a new name
- In the last transformation , we generated some column names that literally reflected the column expression calculation that we performed on it
- Let's use alias to rename those columns

Input	Output
<pre>from pyspark.sql.functions import * authorDF = spark.read.table("authors") authBdayDF = authorDF \ .select("first_name", "last_name", weekofyear(authorDF.birthdate) .alias("Week Born"), date_format(authorDF.added, "y") .alias("Member Since")) authBdayDF.show(5)</pre>	<pre>+-----+-----+-----+-----+ first_name last_name Week Born Member Since +-----+-----+-----+-----+ Walton Adams 9 1997 Marietta Walsh 22 2010 Lily Wintheiser 34 1973 Estevan Gleason 29 1995 Thaddeus Rowe 9 2017 +-----+-----+-----+-----+ only showing top 5 rows</pre>

Date Calculation Functions

- These functions perform some calculation on a given date

PySpark Date Function	Description
add_months(start_date, num_months)	Returns the date that is num_months after start_date.
date_add(start_date, num_days)	Returns the date that is num_days after start_date.
date_sub(start_date, num_days)	Returns the date that is num_days before start_date.
datediff(endDate, startDate)	Returns the number of days from startDate to endDate.
last_day(date)	Returns the last day of the month which the date belongs to.
months_between(date1, date2)	Returns number of months between two dates
next_day(start_date, day_of_week)	Returns the first date which is later than start_date
to_date(date_str[, fmt])	Convert string type containing date value to date format

withColumn() Transformation

- DataFrame.withColumn(colName, colExpression)
 - Returns a new DataFrame by adding a column or replacing the existing column that has the same name
 - colName: string name of the new column or exiting column
 - colExpression: a column expression of the new column or replacement column

- Use datediff as the column expression to calculate number of days between current date and added date.

```
from pyspark.sql.functions import *
authorDF = spark.read.table("authors")
authBdayDF = authorDF \
    .select("added") \
    .withColumn("On Record",
               datediff(current_date(),
                         to_date(authorDF.added)))
authBdayDF.printSchema()
authBdayDF.show(5, False)
```

Input

root	Output
-- added: string (nullable = true)	-- On Record: integer (nullable = true)
+-----+-----+	+-----+-----+
added	On Record
+-----+-----+	+-----+-----+
1997-01-02 04:18:41.0 9009	
2010-08-26 18:20:14.0 4025	
1973-06-11 07:28:12.0 17615	
1995-01-29 16:08:31.0 9713	
2017-01-05 04:13:48.0 1701	
+-----+-----+	+-----+-----+
only showing top 5 rows	

root

Output

when Function

- **when** function evaluates the given conditions and returns values accordingly
- It is possible to chain additional when clauses
 - Acts like **if / elif / else** clause in Python
- Use **otherwise** to catch when all the when fails

Acts like `else` clause in Python

Input	Output
<pre>dataDF = spark.createDataFrame([(1, "a", "4"), (2, "b", "0"), (3, "b", "4"), (4, "d", "4")], ("id", "code", "amt")) dataDF.withColumn("when_column", when((col("code") == "a") (col("code") == "d"), "A") .when((col("code") == "b") & (col("amt") == "4"), "B") .otherwise("A1")).show()</pre>	<pre>+---+---+---+-----+ id code amt when_column +---+---+---+-----+ 1 a 4 A 2 b 0 A1 3 b 4 B 4 d 4 A +---+---+---+-----+</pre>

User-Defined Scalar Function (1/2)

- This data contains a list as one of the columns.
 - Create a UDF that counts the number of favorite numbers

```
src = "file:/home/student/Data/resources/users.parquet"
usersDF = spark.read \
    .parquet(src)
usersDF.show()
```

Input

```
+-----+-----+-----+
|  name|favorite_color|favorite_numbers|
+-----+-----+-----+
|Alyssa|          null|[3, 9, 15, 20]|
|   Ben|            red|          [] |
+-----+-----+-----+
```

Output

User-Defined Scalar Function (2/2)

- Create a UDF using the **udf()** function
 - Need to import udf
 - Use lambda notation in the udf function to pass parameters to the UDF
- The udf function requires entering the output datatype

```
from pyspark.sql.functions import udf  
from pyspark.sql.types import IntegerType  
  
# Example of user-defined Scalar function  
def cnt_fav(arr):  
    return len(arr)  
  
cntFavUDF = udf(lambda a: cnt_fav(a), IntegerType())  
  
usersDF.select(  
    col("name"),  
    cntFavUDF(col("favorite_numbers"))  
    .alias("Fav_Cnt")) \  
    .show()
```

Input

name	Fav_Cnt
Alyssa	4
Ben	0

Output

groupBy Transformation

- Groups the DataFrame using the specified columns, so we can run aggregation on them
 - Returns pyspark.sql.GroupedData object
- The GroupedData object has many methods that aggregate data on the rows grouped on the specified column
 - count()**, **max()**, **min()**, **sum()**, **mean()** perform the expected aggregation function

Input	func	Output
<pre>my_list = [["henry park", 'M', 23.00], ["shaun smith", 'M', 15.25], ["sharon ramirez", 'F', 45.50], ["jessica bolt", 'F', 20.00]] my_schema = "name string, gender string, amt double" myDF = spark.createDataFrame(my_list, my_schema) myDF.groupBy("gender").sum("amt").show()</pre>		<pre>+-----+-----+ gender sum(amt) +-----+-----+ F 65.5 M 38.25 +-----+-----+</pre>

Aggregating Multiple Columns (1/5)

- Using the **agg()** wrapper, it is possible to calculate multiple aggregations on multiple columns
 - Data source used for the aggregation analysis

Input

```
from pyspark.sql.functions import col, sum, avg, max

my_company = [
    ("Henry", "Engineering", "TX", 105000, 20000),
    ("Jessica", "Sales", "TX", 75000, 15000),
    ("Shaun", "Finance", "CA", 70000, 14000),
    ("Sharon", "Marketing", "TX", 90000, 18000),
    ("Jonathan", "HR", "CA", 95000, 19000),
    ("Jason", "Engineering", "TX", 110000, 21000),
    ("Scott", "Finance", "CA", 85000, 17000),
    ("Timothy", "Marketing", "CA", 95000, 19000) ]

schema = ["employee_name", "department", "state", "salary", "bonus"]
df = spark.createDataFrame(my_company, schema)
```

Aggregating Multiple Columns (2/5)

- Using the **agg()** wrapper, it is possible to calculate multiple aggregations on multiple columns
 - Multiple columns can be grouped by
 - Aggregate functions that support it can have multiple column parameters

```
df.groupBy("state", "department") \
    .sum("salary", "bonus") \
    .show(truncate=False)
```

Input

state	department	sum(salary)	sum(bonus)
CA	Marketing	95000	19000
TX	Engineering	215000	41000
CA	Finance	155000	31000
TX	Sales	75000	15000
TX	Marketing	90000	18000
CA	HR	95000	19000

Output

Aggregating Multiple Columns (3/5)

- Using the **agg()** wrapper, it is possible to calculate multiple aggregations on multiple columns
 - Using **agg()**, we can produce multiple output columns of aggregated data over the same column

```
df.groupBy("department") \
  .agg(sum("salary").alias("Sum of Salaries"), \
       mean("salary").alias("Mean Salary"), \
       max("bonus").alias("Max Bonus")) \
  .show(truncate=False)
```

Input

department	Sum of Salaries	Mean Salary	Max Bonus
Sales	75000	75000.0	15000
Engineering	215000	107500.0	21000
HR	95000	95000.0	19000
Finance	155000	77500.0	17000
Marketing	185000	92500.0	19000

Output

Aggregating Multiple Columns (4/5)

- Using the `agg()` wrapper, it is possible to calculate multiple aggregations on multiple columns
 - It is possible to filter the output of the aggregated output using a `where()` clause

```
df.groupBy("department") \
  .agg(sum("salary").alias("Sum of Salaries"), \
       mean("salary").alias("Mean Salary"), \
       max("bonus").alias("Max Bonus")) \
  .where(col("Max Bonus") >= 20000) \
  .show(truncate=False)
```

Input

department	Sum of Salaries	Mean Salary	Max Bonus
Engineering	215000	107500.0	21000

Output

Aggregating Multiple Columns (5/5)

- Finally re-order the aggregated columns for a more intuitive output.

```
df.groupBy("state", "department") \
  .sum("salary", "bonus") \
  .orderBy("state", "department") \
  .show(truncate=False)
```

Input

state	department	sum(salary)	sum(bonus)
CA	Finance	155000	31000
CA	HR	95000	19000
CA	Marketing	95000	19000
TX	Engineering	215000	41000
TX	Marketing	90000	18000
TX	Sales	75000	15000

Output

Using join() Transformation (1/4)

- Joins with another DataFrame, using the lineage given join expression
- `dataframe1.join(datafram2, <string, list of string, or column expression to join on>, <join type>)`
 - If the join on is a string or list of strings indicating column to join on, column name(s) must exist on both dataframes

```
my_staff = [
    (1,"Henry","Engineering","TX"),
    (2,"Jessica","Sales","TX"),
    (3,"Shaun","Finance","CA"),
    (4,"Sharon","Marketing","TX"),
    (5,"Jonathan","HR","CA"),
    (6,"Jason","Engineering","TX"),
    (7,"Scott","Finance","CA"),
    (8,"Timothy","Marketing","CA")]

schema = ["emp_id", "employee_name",
          "department", "state"]
staffDF = spark \
    .createDataFrame(my_staff, schema)
```

```
on c
income = [
    (1,105000,20000),
    (2,75000,15000),
    (3,70000,14000),
    (4,90000,18000),
    (5,95000,19000),
    (6,110000,21000)]

schema =
["emp_id","salary","bonus"]
incomeDF = spark \
    .createDataFrame(income,schema)
```

Using join() Transformation (2/4)

- When the join column name is the same, we can use a single string
 - The output produces a single column for the join on column

```
staffDF.join(incomeDF, "emp_id").show(truncate=False)
```

Input

emp_id	employee_name	department	state	salary	bonus
6	Jason	Engineering	TX	110000	21000
5	Jonathan	HR	CA	95000	19000
1	Henry	Engineering	TX	105000	20000
3	Shaun	Finance	CA	70000	14000
2	Jessica	Sales	TX	75000	15000
4	Sharon	Marketing	TX	90000	18000

Output

Using join() Transformation (3/4)

- If the join on column names are different use column expression with column objects
 - The output produces both columns of the join on columns

```
income = [(1,105000,20000),(2,75000,15000),(3,70000,14000),  
          (4,90000,18000),(5,95000,19000),(6,110000,21000)]
```

Input

```
schema = ["id","salary","bonus"]  
incomeDF = spark \  
    .createDataFrame(income,schema)  
staffDF \  
    .join(incomeDF, staffDF.emp_id == incomeDF.id) \  
    .show(truncate=False)
```

emp_id	employee_name	department	state	id	salary	bonus
6	Jason	Engineering	TX	6	110000	21000
5	Jonathan	HR	CA	5	95000	19000
1	Henry	Engineering	TX	1	105000	20000
3	Shaun	Finance	CA	3	70000	14000
2	Jessica	Sales	TX	2	75000	15000
4	Sharon	Marketing	TX	4	90000	18000

Output

Using join() Transformation (4/4)

- Join provide many join options
 - Default is inner join

- left, leftouter, right, rightouter etc.

```
staffDF \
    .join(incomeDF,
          staffDF.emp_id == incomeDF.id,
          "left_outer") \
    .show(truncate=False)
```

Input

emp_id	employee_name	department	state	id	salary	bonus
7	Scott	Finance	CA	null	null	null
6	Jason	Engineering	TX	6	110000	21000
5	Jonathan	HR	CA	5	95000	19000
1	Henry	Engineering	TX	1	105000	20000
3	Shaun	Finance	CA	3	70000	14000
8	Timothy	Marketing	CA	null	null	null
2	Jessica	Sales	TX	2	75000	15000
4	Sharon	Marketing	TX	4	90000	18000

Output

Typed and Untyped Operation

- Dataset operations can be classified as typed and untyped transformations
- Typed Operations
 - Transformations that maintain the strongly typed Dataset data type
 - JVM Object of type [U] JVM Object of type [U]
- Untyped Operations
 - Transformation that destroy the original Dataset data type
 - Because untyped transformations produce DataFrames, they are sometimes called DataFrame operations
 - JVM Object of type [U] Row Object type

Examples of Untyped Transformations

- Untyped Transformation

```
#spark, df are from the previous example  
#Print the schema in a tree format  
df.printSchema()
```

Input

```
root  
| -- age: long (nullable = true)  
| -- name: string (nullable = true)
```

Output

```
#Select only the "name" column  
df.select("name").show()
```

Input

```
+-----+  
|    name|  
+-----+  
|Michael|  
|   Andy|  
| Justin|  
+-----+
```

Output

Examples of Untyped Transformations

- Untyped Transformation

```
# Select everybody, but increment  
# the age by 1  
  
df.select(df['name'], df['age'] + 1).show()
```

Input

```
+-----+  
| name | (age + 1) |  
+-----+-----+  
| Michael | null |  
| Andy | 31 |  
| Justin | 20 |  
+-----+-----+
```

Output

Using SQL Queries

- You can query tables and dataframes using standard SQL queries
 - Use `SparkSession.sql (<ISO standard query>)` method
- Useful for developers who are more familiar with SQL
 - Substitute for `DataFrame API` methods and `Column expressions`
- Allows ad-hoc queries

\$ spark-sql						Cmd
Spark-sql> select * from authors limit 5;						Input
1	Walton	Adams	barmstrong@example...	1989-03-01	1997-01-02	Output
04:18:....						
2	Marietta	Walsh	hand.stella@example...	2018-05-30	2010-08-26	
18:20:....						
3	Lily	Wintheiser	darren.blanda@example...	1981-08-21	1973-06-11	
07:28:....						
4	Estevan	Gleason	shanahan.aliyah@example...	2013-07-17	1995-01-29	
16:08:....						
5	Thaddeus	Rowe	bednar.robin@example...	2019-02-26	2017-01-05	
04:13:....						

Using SparkSession.sql with Hive

- Use SparkSession.sql to use standard SQL queries

```
authorsDF = spark.sql("SELECT * FROM authors LIMIT 5")
authorsDF.show()
```

Input

	id	first_name	last_name	email	birthdate	added
1	1	Walton	Adams	barmstrong@example.com	1989-03-01	1997-01-02 04:18:...
2	2	Marietta	Walsh	hand.stella@example.com	2018-05-30	2010-08-26 18:20:...
3	3	Lily	Wintheiser	darren.blanda@example.com	1981-08-21	1973-06-11 07:28:...
4	4	Estevan	Gleason	shanahan.aliyah@example.com	2013-07-17	1995-01-29 16:08:...
5	5	Thaddeus	Rowe	bednar.robin@example.com	2019-02-26	2017-01-05 04:13:...

Output

SQL Queries can be Complex

- Quotes usage in SQL

```
authorsDF = spark \  
    .sql("""SELECT first_name, last_name, Birth_Week, Member_Since FROM  
        (SELECT first_name, last_name, weekofyear(birthdate) as Birth_Week,  
            date_format(added, "y") as Member_Since FROM authors)  
    WHERE Member_Since >= "2000"  
    LIMIT 5 """)  
authorsDF.show()
```

Input

first_name	last_name	Birth_Week	Member_Since
Marietta	Walsh	22	2010
Thaddeus	Rowe	9	2017
Cortez	Russel	24	2007
Caterina	Cartwright	40	2001
Skye	Powlowski	34	2000

Output

Using `SparkSession.sql` with `DataFrames`

- In order to use dataframes as tables in the SQL query, temporary views must be created for them
- Temporary views are session-scoped and will disappear if the session that created it terminates
 - `createTempView(<name of view>)`
 - `createOrReplaceTempView (<name of view>)`
- If the temporary view needs to be shared among all sessions and kept alive until the Spark application terminates, use global temporary views
 - `createGlobalTempView (<name of view>)`
 - `createOrReplaceGlobalTempView (<name of view>)`
 - Global temporary views are saved in a Spark system preserved database
 - Use `global_view.<name of view>` to access the global view

Example Using Temporary view

- Create a temporary view for the staffDF from previous example
- Use the temporary view name in the SQL query

```
staffDF.createOrReplaceTempView("staff")
spark.sql(""" SELECT * FROM staff """).show()
```

Input

emp_id	employee_name	department	state
1	Henry	Engineering	TX
2	Jessica	Sales	TX
3	Shaun	Finance	CA
4	Sharon	Marketing	TX
5	Jonathan	HR	CA
6	Jason	Engineering	TX
7	Scott	Finance	CA
8	Timothy	Marketing	CA

Output

Query without Temporary View

- Sometimes we may want to do a one-off query and not bother with creating a temporary view
- Use `data format.`path to file`` syntax in lieu of a table name

```
df = spark.read.parquet("users.parquet")
df.createOrReplaceTempView("users")
spark.sql(""" SELECT * FROM users """).show()
```

Input

```
+-----+-----+
| name|favorite_color|favorite_numbers|
+-----+-----+
| Alyssa|        null| [3, 9, 15, 20]|
|   Ben|         red|      [] |
+-----+-----+
```

Output

```
spark \
    .sql(""" SELECT * FROM parquet.`users.parquet`""").show()
```

Input

```
+-----+-----+
| name|favorite_color|favorite_numbers|
+-----+-----+
| Alyssa|        null| [3, 9, 15, 20]|
|   Ben|         red|      [] |
+-----+-----+
```

Output

Performance Using SQL Queries

- Spark SQL converts both direct SQL queries and SparkSession methods to equivalent code using the Catalyst optimizer - there is no difference in performance

```
spark.sql(""" SELECT * from authors WHERE first_name LIKE 'A%' LIMIT 3 """).show()
```

Input

```
+----+-----+-----+-----+-----+
| id|first_name|last_name|      email| birthdate|      added|
+----+-----+-----+-----+-----+
| 29|    America|Marquardt|ulockman@example.org|2018-11-21|2010-10-03 14:12:...
| 31|      Alvis|     Crist|kennith25@example...|1973-05-02|2003-07-11 12:52:...
| 32|     Adele|   Schultz| elias04@example.com|1970-04-05|1973-12-06 15:09:...
+----+-----+-----+-----+-----+
```

Output

```
df = spark.read.table("authors")
df.where(df.first_name.startswith('A')).limit(3).show()
```

Input

```
+----+-----+-----+-----+-----+
| id|first_name|last_name|      email| birthdate|      added|
+----+-----+-----+-----+-----+
| 29|    America|Marquardt|ulockman@example.org|2018-11-21|2010-10-03 14:12:...
| 31|      Alvis|     Crist|kennith25@example...|1973-05-02|2003-07-11 12:52:...
| 32|     Adele|   Schultz| elias04@example.com|1970-04-05|1973-12-06 15:09:...
+----+-----+-----+-----+-----+
```

Output

Using SparkSession.sql for DDL and DML

- `SparkSession.sql` can accept DDL and DML queries, in addition to SQL queries
- Notice that in addition to the Hive tables, Spark lists the temporary views that we created in previous slides

```
spark.sql("show tables").show()
```

Input

database	tableName	isTemporary
default	authors	false
default	authors_external	false
default	authors_managed	false
default	posts	false
	staff	true
	users	true

Output

Create Hive Table with SparkSession.sql

- Create Hive table using DDL statement from SparkSession.sql

```
spark.sql("""" CREATE EXTERNAL TABLE IF NOT EXISTS test
            (id int, name string)
            LOCATION '/user/student/test' """")
spark.sql("show tables").show()
```

Input

database	tableName	isTemporary
default	authors	false
default	authors_external	false
default	authors_managed	false
default	posts	false
	staff	true
	users	true

Output

Modify Hive Table with SparkSession.sql

- Modify Hive table using DML statement from SparkSession.sql

Input

```
spark.sql(""" ALTER TABLE test  
          ADD COLUMNS (age int) """)  
spark.sql(""" DESCRIBE test """).show()
```

Output

col_name	data_type	comment
id	int	null
name	string	null
age	int	null

[Lab8]

Spark SQL Transformations



[Lab9]

Working with Spark SQL



Unit 3

Processing Streaming Data

Big Data Processing with Apache Spark

Unit 3

Processing Streaming Data

- | 3.1. Introduction to Spark Streaming
- | 3.2. Working with Unstructured Streaming Data
- | 3.3. Working with Structured Streaming Data

What is Spark Streaming

- Apache Spark Streaming is a scalable, fault-tolerant and high-throughput stream processing engine
- Apache Spark Streaming consists of two separate APIs
 - Spark Streaming API is an extension of the Core API and used to process unstructured streaming data
 - The Structured Streaming API is an extension of the DataFrame API and is used to process structured data
- Catalyst Optimizer
 - Structured Streaming is built on top of the higher-level APIs and takes advantage of Catalyst Optimizer
 - Spark Streaming API is built directly on top of the Core API engine and unable to use the Catalyst Optimizer



Spark Streaming vs Structured Streaming

- **Spark Streaming API**

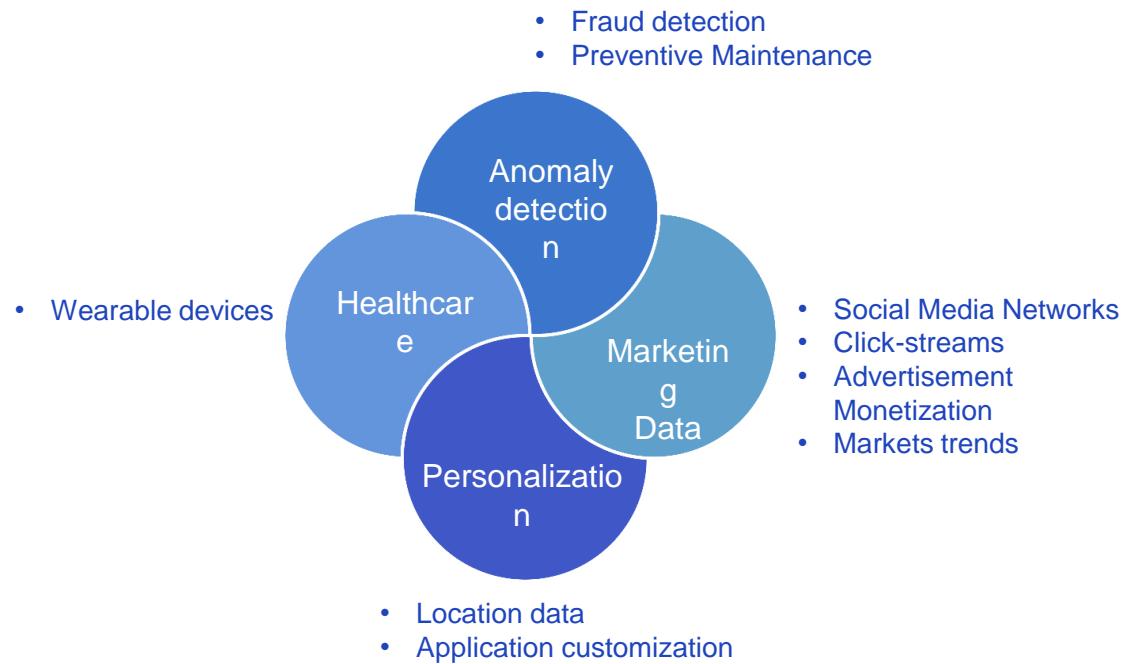
- Primarily for processing unstructured or semi-structured data
- Based on RDD Core API
 - MapReduce based semantics
- Core level lower API
- Limited optimization
- Once and only once processing

- **Structured Streaming API**

- Primarily for processing structured data
- Based on DataFrame API
 - SQL based semantics
- Higher level API
- Uses Catalyst Optimizer for highly efficient optimization
- Guaranteed consistency between streaming and static queries

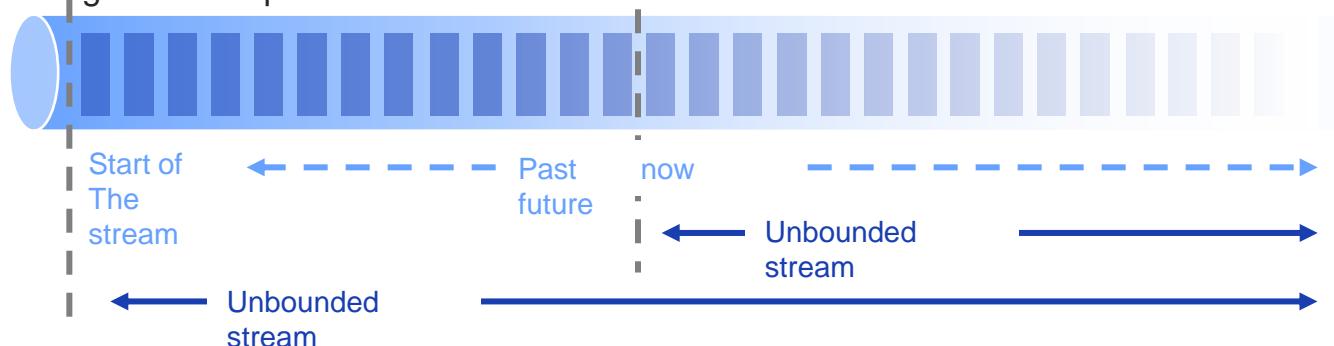
Spark Streaming Use-Case

- There are many sources of streaming data that are generated and must be processed in real-time



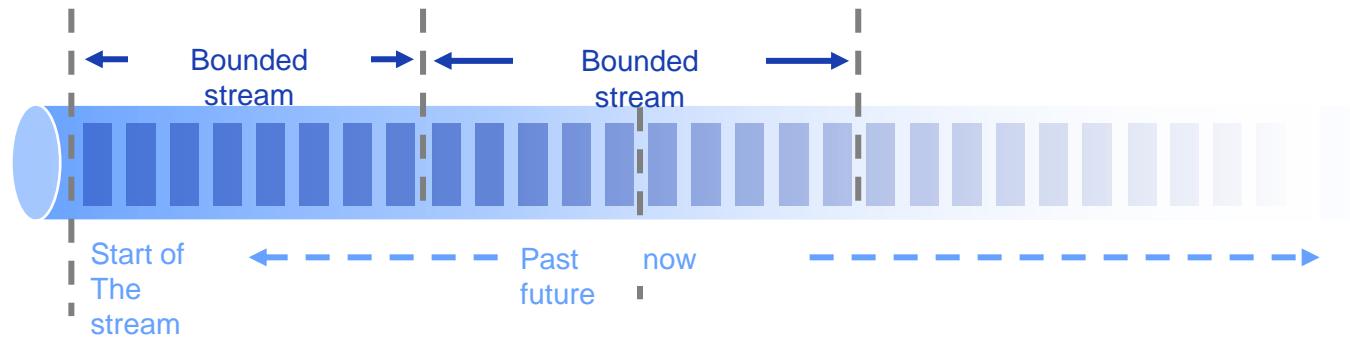
Unbounded Data Streams - Review

- Unbounded data stream has some starting point
 - May be very far in the past and not of any consequence in processing current incoming data
- Incoming data stream has no defined end
 - The incoming data stream is not expected to terminate at any time
- Unbounded streams must be ingested in real-time and in specific order
 - It is not possible to wait until the entire stream is read - it may never end
 - Events must be ingested and processed at the time the event occurred



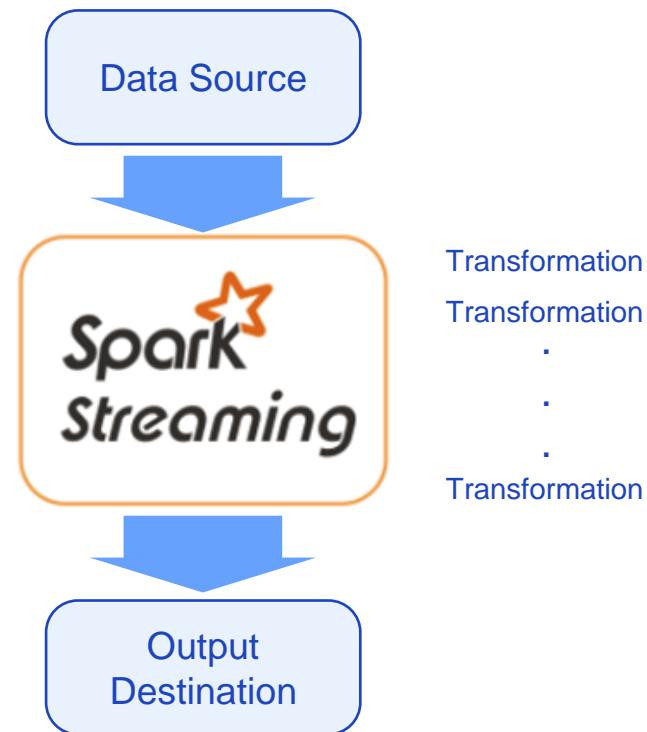
Bounded Data Streams - Review

- Bounded data stream has a starting point and an ending point
- Possible to ingest and store the entire dataset before processing
- Ingesting and processing events in order is not as strict
 - Data may be re-sorted to match the desired query since it has been saved



Developing Streaming Application

- Most streaming applications follow the same execution pattern
 - Step 1: Create DataFrame or DStream from incoming data source
 - Step 2: Define queries and transformations on the DataFrame or DStream
 - Step 3: Save or display the results



Spark Streaming Operations

- As in the other APIs, there are two types of operations in Spark Streaming
 - Transformation applies operations on the data and creates a new object with the changes
 - All Spark transformation execute immutably
 - When an Action is called, all Executors return their results to the Driver program
 - All Spark transformations execute lazily
 - An action triggers Spark to follow the lineage and DAG dependency graph to formulate a final physical plan

Getting Started with Streaming

- We can use the Spark shell for development of streaming applications
- Streaming application require a minimum of two threads or executed on YARN
 - One thread is required to ingest the incoming streaming data
 - Another thread processes the streaming data

```
% pyspark --master 'local[2]'          # Specify minimum of 2 threads
% pyspark --master 'local[*]'         # Use as many threads as available. Must be at least 2
% pyspark --master yarn              # Run pyspark shell on YARN
```

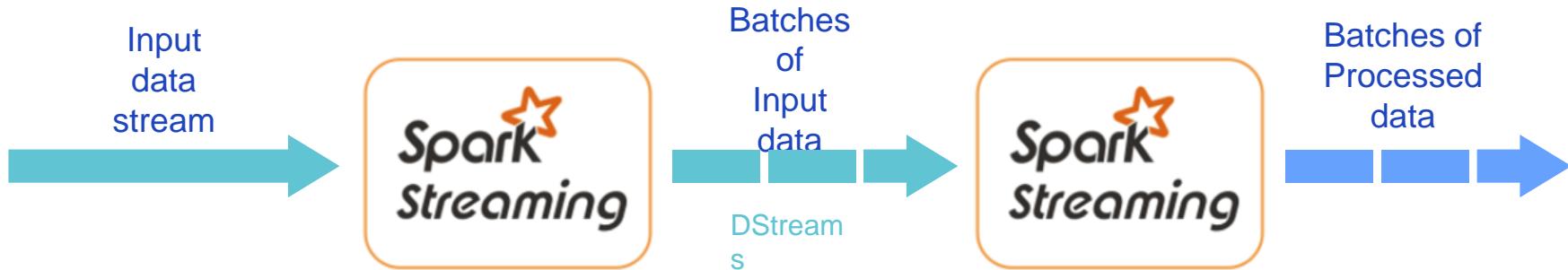
Unit 3

Processing Streaming Data

- | 3.1. Introduction to Spark Streaming.
- | 3.2. Working with Unstructured Streaming Data
- | 3.3. Working with Structured Streaming Data

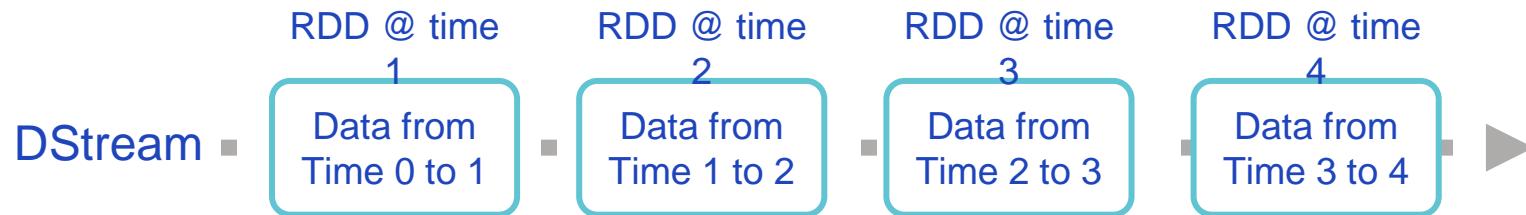
Micro-batches

- Spark Streaming ingests streaming data and divides the data into smaller batches
 - The Spark engine processes the data at each batch level
 - Recall micro-batch processing and event-driven processing from our previous lessons
 - Spark is a micro-batch based processing system



DStreams

- Spark Streaming uses a high-level abstraction called DStream
 - DStream stands for discretized stream
 - The process of creating micro-batches from the continuous stream of data creates DStreams
 - Internally, a DStream is represented by a sequence of RDDs



Spark StreamingContext

- Spark Context and SparkSession are the main objects for Core API and DataFrame API, respectively
- Stream API's equivalent main point of entry is the StreamingContext object
 - Provides all the functionality of Spark Streaming API
- Unlike the Spark Context and SparkSession, the StreamingContext must be instantiated in the shell
 - Import StreamingContext
 - Pass the current Spark Context as parameter

```
from pyspark.streaming import StreamingContext

# Set the log level to ERROR
sc.setLogLevel("ERROR")

# Create and configure a new Streaming Context
# with a 1 second batch duration
ssc = StreamingContext(sc,1)
```

Input

Starting the Streaming Engine

- The next step is to specify the logic of the streaming application
- Finally, we have to start the Streaming Spark Engine to begin processing the streaming data
 - Use the **start()** method of the StreamingContext
- Once the StreamingContext is started, it should wait for the termination signal to stop
 - Use **awaitTermination()** method

Input

```
ssc = StreamingContext(sc,1)

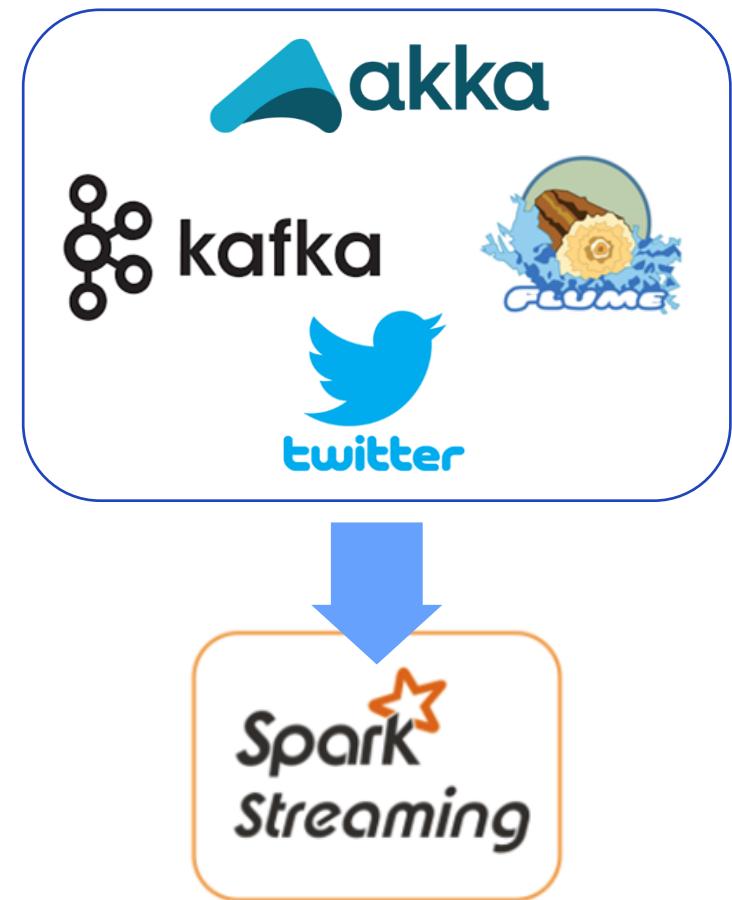
# Place streaming application logic here

ssc.start()
ssc.awaitTermination()

# Any code placed after the Spark Streaming engine has started will be ignored
```

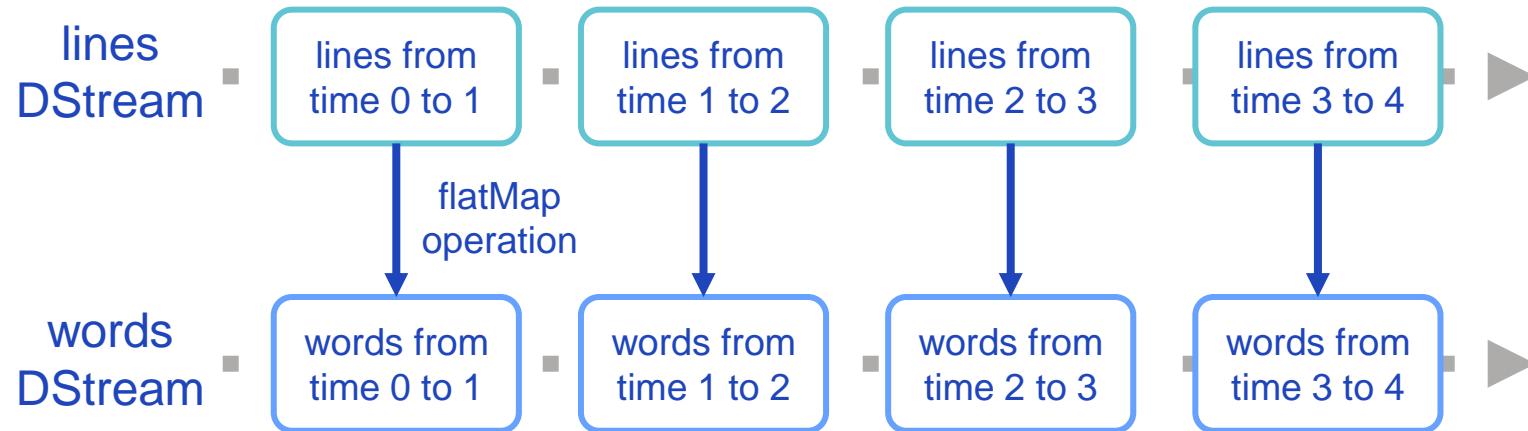
Spark Streaming Data Sources

- Spark Streaming provides several built-in data sources out of the box
- Basic sources are available directly from Spark Streaming API
 - Socket connections
 - File systems - Monitors a HDFS directory for new files
- Advanced sources require dependency libraries to be imported
 - Kafka
 - AWS Kinesis
 - Flume
 - Akka Actors
 - Twitter



Transforming DStreams

- DStreams are also created through transformation of parent Dstreams
- Each transformation is affected on the underlying sequence of RDDs
- Example:
 - Read some streaming text
 - Vectorize the words using **flatMap**



Transformations on Spark Streaming

- Underlying DStreams is a sequence of RDDs
 - All Core API transformation are available to DStream API
- Many commonly used transformations have a Spark Streaming direct shortcut
 - map(func), flatMap(func), filter(func), repartition(), union()
- Many of the Pair RDD transformations also have direct shortcuts
 - reduce(func), reduceByKey(), groupByKey(), join()
- Some transformations unique to Spark Streaming
 - Stateful transformation including Windows based transformations
- ```
The transform() passes a pointer to the current DStream to process
my_dstream.transform(lambda rdd: rdd.distinct())
```

Input

## Closer Look at foreachRDD (1/3)

- Recall the **foreachPartition** transformation in the Core API
  - Generally used for avoiding a very expensive operation such as connecting to a RDBMS for each row
- The **foreachRDD** has a similar use, and developers must be careful when using this transformation
- Example: For each DStream, we are trying to connect to a RDBMS, send all the records in the DStream, and close the connection
  - Unfortunately, the connection is created on the Driver

```
def sendRecords2DB(rdd):
 # this connection is created on the Driver
 cnn = createConnection("connection string")
 rdd.foreach(lambda record: cnn.send(record))
 cnn.close()

my_dstream.foreachRDD(lambda rdd: sendRecord2DB(rdd))
```

Input

## Closer Look at foreachRDD (2/3)

- In the previous version, we tried to send the DB connection to each of the Executors which will fail
- This time, we will make sure that each Executor creates its own DB connection
  - From my\_stream, we call **sendRecord2DB** "foreach" element of my\_stream
  - **sendRecord2DB**, and subsequently the **createConnection**, is now executed by all the Executors since it is called on each row element of my\_stream

- ```
def sendRecord2DB(record):
    cnn = createConnection("connection string")
    # this connection is created on the Executor
    cnn.send(record)
    cnn.close()

my_dstream.foreachRDD(lambda rdd: rdd.foreach(sendRecord2DB))
```

Input

Closer Look at foreachRDD (3/3)

- In engineering, this type of problem is referred to as a Goldilocks problem
 - Refers to problems where extremes must be avoided, and a middle-ground solution must be found
- Desired middle-ground solution:
 - Each Executor responsible for creating a single connection to the database
 - All the records in a Partition utilizes that single connection to update row items
 - Combine **foreachRDD** and **foreachPartition** to accomplish this

```
def sendPartition2DB(iterator):
    # this connection is created on the Executor
    cnn = createConnection("connection string")
    for record in iterator:
        cnn.send(record)
    cnn.close()

my_dstream.foreachRDD(lambda rdd: rdd.foreachPartition(sendPartition2DB))
```

Input

Output Operations (1/3)

- Actions are required to trigger the Spark Streaming transformations
 - Very often, actions in the underlying RDD operations will trigger it
- These commonly used Output operations also act as Actions and trigger transformations

Output Operation	Meaning
pprint()	Prints the first ten elements of every batch of data in a DStream on the driver node running the streaming application. This is useful for development and debugging.
saveAsTextFiles(prefix, [suffix])	Save this DStream's contents as text files. The file name at each batch interval is generated based on prefix and suffix: "prefix-TIME_IN_MS[.suffix]".
foreachRDD(func)	The most generic output operator that applies a function, func, to each RDD generated from the stream. This function should push the data in each RDD to an external system, such as saving the RDD to files, or writing it over the network to a database. Note that the function func is executed in the driver process running the streaming application, and will usually have RDD actions in it that will force the computation of the streaming RDDs.

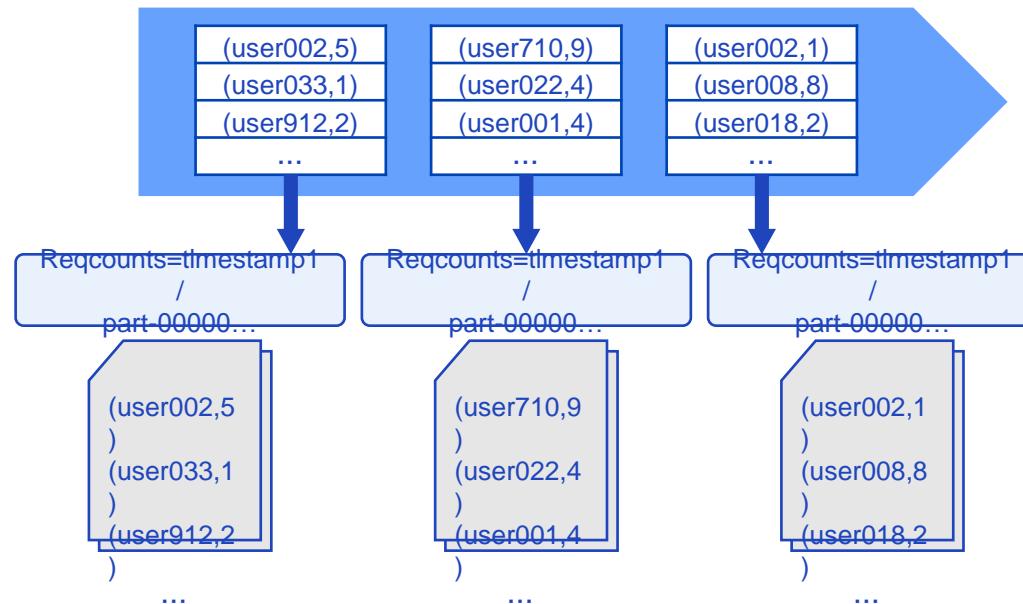
Output Operations (2/3)

- Use Output operations to save the processed data to file systems or RDBMS
- These Output operations allow saving the output in other formats
 - These operations are not available in the Python version of the API

Output Operation	Meaning
saveAsObjectFiles(prefix, [suffix])	Save this DStream's contents as SequenceFiles of serialized Java objects. The file name at each batch interval is generated based on prefix and suffix: "prefix-TIME_IN_MS[.suffix]".
saveAsHadoopFiles(prefix, [suffix])	Save this DStream's contents as Hadoop files. The file name at each batch interval is generated based on prefix and suffix: "prefix-TIME_IN_MS[.suffix]".

Output Operations (3/3)

- The Output operations generates a filename based on the prefix, timestamp, and a optional suffix
- Since Spark is a distributed parallel processing engine, each filename becomes a directory in HDFS
 - Each Executor saves its' partition of the data separately



Stateful Transformation

- Spark Streaming provides stateful transformations that preserve information across DStreams
 - **UpdateStateByKey** preserves state during the lifetime of the StreamingContext
 - Window transformation also preserve state across DStreams, but only over a defined window
- Because **UpdateStateByKey** has to preserve state information over the lifetime of the Streaming Context, its performance can suffer tremendously
 - Caution should be used
- Window transformations is alternative where state is preserved but for a shorter duration
 - Very rarely does our use-case demand that state be preserved forever
 - Windows allows us to define a window of time over which we will keep track of state
- Syntax
 - `my_dstream.updateStateByKey(function)`
 - `my_dstream.reduceByKeyAndWindow(function, <window information>)`

updateStateByKey Transformation (1/3)

- updateStateByKey works with DStreams with Pair RDDs in the underlying sequential RDD
- In order to use **updateStateByKey** to keep track of state continuously, two things are required
 - A state of arbitrary data type to keep track of current state
 - A defined function to update the state

Input

```
def updateFunction(newValues, runningCount):  
    if runningCount is None: return sum(newValues)  
    else: return sum(newValues) + runningCount
```

updateStateByKey Transformation (2/3)

- Spark Streaming will call the update function for all existing keys
 - The function is called even if the current DStream does not have new data for that key
 - This means that as the number of keys increase over the lifetime of the streaming application, each subsequent batch or DStream will have to iterate over a ever increasing number of keys
- For fault-tolerance, Spark needs to save the state to durable storage periodically
 - Use StreamingContext.checkpoint(<checkpoint directory>)

```
from pyspark.streaming import StreamingContext
ssc = StreamingContext(sc, 1)

ssc.checkpoint("checkpoint")

# read streaming data from socket
hostname = "localhost"
port = 44444
lines = ssc.socketTextStream(hostname, port)
```

Input

updateStateByKey Transformation (3/3)

- Similar to other Pair RDD aggregation transformations, **updateStateByKey** first groups the underlying RDDs by the same key
- The values of matching keys are collected and passed to the updateFunction
- The key is used to retrieve any existing previous state value

```
lines = ssc.socketTextStream(hostname, port)
running_wc = lines \
    .flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .updateStateByKey(updateFunction)

running_wc.pprint()

ssc.start()
ssc.awaitTermination()
```

Input

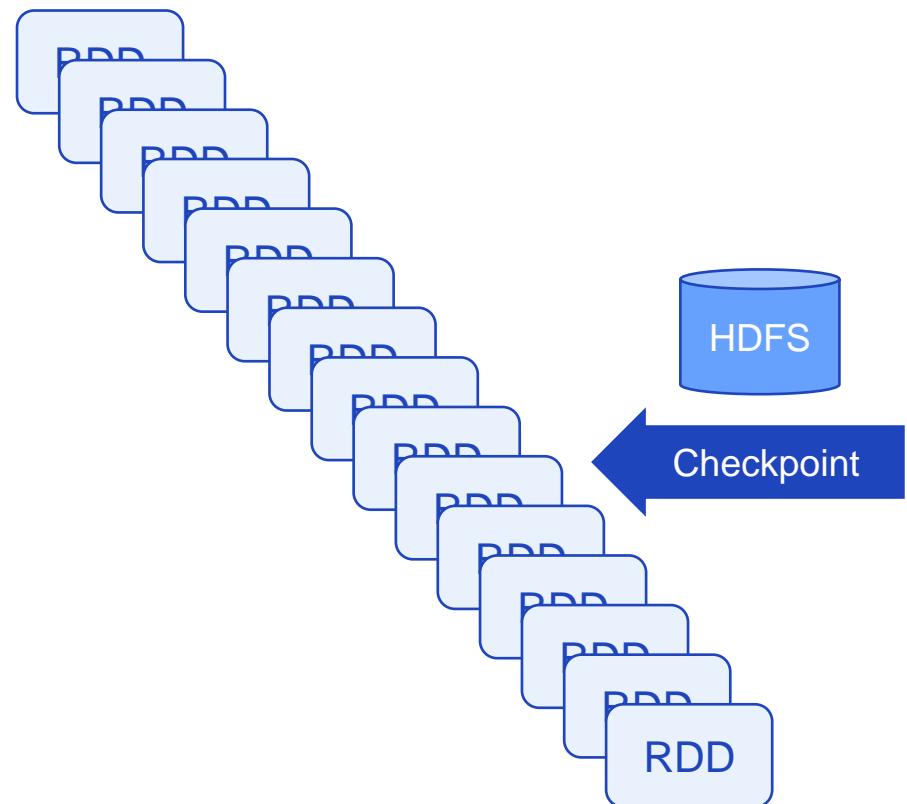
Closer Look at Checkpointing

- Spark keeps a directed acyclic graph of the dependency relationships between all immutable RDDs
 - This is the lineage information for each RDD
- Spark keeps and stores lineage information for all RDDs until it is no longer required
 - It is no longer required if there is no more dependency on that RDD
- Normally in Spark Streaming, each batch of DStream is processed and forgotten
 - We can remove lineage for all RDDs in already processed DStreams

What happens when we
call `updateStateByKey` ?

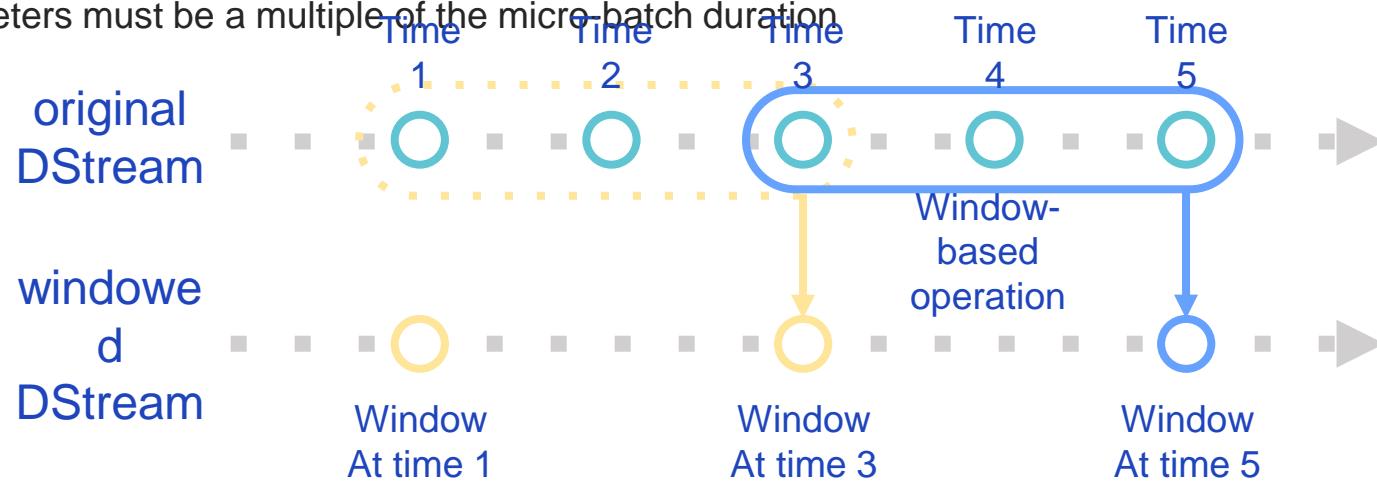
Checkpoint to Avoid Infinite Lineage

- Lineage is kept as long as there is a dependency
- The **updateStateByKey** creates and never-ending dependency all the way back to the very first RDD in the very first DStream processed by the Streaming application
 - This is referred to as infinite lineage
- Checkpointing periodically saves the running state durable to HDFS
 - This allows any lineage kept prior to the checkpoint to be deleted



Window Operations

- Window operation allow transformations to be applied over a sliding window
 - All DStream batches that fall inside the window will be processed
- Length of window controls number of batches that fall inside the window
- Sliding interval controls how often a new window operation is performed
- Both parameters must be a multiple of the micro-batch duration



Window Transformations (1/2)

- These Window transformations process DStreams that contain Pair RDDs
- Produces aggregated results over grouped rows with equal keys

Transformation	Meaning
reduceByKeyAndWindow (func, windowLength, slideInterval, [numTasks])	Call on DStream composed of Key:Value tuples. Similar to the generic reduceByKey, however the reduction is performed over a window of windowLength duration that is created every slideInterval.
countByValueAndWindow (windowLength, slideInterval, [numTasks])	Call on DStream composed of Key:Value tuples. Returns a DStream of Key:Value tuple, where Value is the frequency count of Key over a window of windowLength duration that is created every slideInterval.

Window Transformations (2/2)

- These Window transformations process DStreams that contain regular RDDs
- Produces results over all rows in the DStream

Transformation	Meaning
window (windowLength, slideInterval)	Return a new DStream which is computed based on windowed batches of the source DStream.
countByWindow (windowLength, slideInterval)	Return a sliding window count of elements in the stream.
reduceByWindow(func, windowLength, slideInterval)	Return a new single-element stream, created by aggregating elements in the stream over a sliding interval using func. The function should be associative and commutative so that it can be computed correctly in parallel.

reduceByKeyAndWindow() Transformation

- This transformation behave in essence the same way as the normal **reduceByKey(func)**
- The key difference is the data over which it aggregate the provided function
- **reduceByKeyAndWindow(<func>,windowLength=<integer in seconds>, slideInterval=<integer in seconds>)**
 - Apply func to aggregate Pair RDDs with same key over windowLength seconds,

```
# Code to instantiate Streaming context and create checkpoint directory

window_wc = \
    ssc.socketTextStream(hostname, port) \
    .flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKeyAndWindow(lambda v1, v2: v1+v2, 5, 2)

pprint(window_wc)

# Code to start streaming context and await termination
```

Input

countByWindow Transformation

- Count number of rows within Window
- The output screen shows a new calculation every 2 seconds

Input

```
# Create streaming context with batch duration of
# 1 second

# filter lines with Alice and count in
# Window size of 5 sec, every 2 sec

alice_window = lines \
    .map(lambda line: line.upper()) \
    .filter(lambda line: "ALICE" in line) \
    .countByWindow(5, 2)

alice_window.pprint()

ssc.start()
ssc.awaitTermination()
```

Output

```
Time : 2021-09-04 14:27:58
12

Time : 2021-09-04 14:28:00
41

Time : 2021-09-04 14:28:02
```

Unit 3

Processing Streaming Data

- | 3.1. Introduction to Spark Streaming
- | 3.2. Working with Unstructured Streaming Data
- | 3.3. Working with Structured Streaming Data

Spark Structured Streaming

- Spark's Structured Streaming employs the same Spark SQL engine
 - Allows developers to develop and create streaming queries and applications as if on static data
 - Structured Streaming takes care of incrementally re-running the logic on new data as they arrive in order to update the final results
 - Because the underlying engine is the same Spark SQL engine, developers can use the same DataFrame API
 - Uses the same SparkSession object and its methods and operators
- In addition to the basic DataFrame API, Structured Streaming provides additional functionality geared toward processing streaming data
 - Event windows and watermarks to handle late arriving data
 - Joining static and streaming data - effectively built-in Lambda Architecture

Let's Get Familiar with an Example (1/4)

- Reading the streaming data with **readStream()** method
 - Similar to SparkSession's **read()** method
 - Use **format()** to specify source type
 - Use **option()** to set options for the selected source type

Input

```
# Create DataFrame representing the stream of input lines from
# Socket connection to localhost at port 44444

lines = spark \
    .readStream \
    .format("socket") \
    .option("host", "localhost") \
    .option("port", 44444) \
    .load()
```

Let's Get Familiar with an Example (2/4)

- Create the application logic using transformations and queries
- Set action to trigger the transformation logic

Input

```
# import the spark.sql functions to create column expression
from pyspark.sql.functions import *

# Split the lines into words
words = lines.select(
    explode(
        split(lines.value, " ")
    ).alias("word")
)

# Generate running word count
wordCounts = words.groupBy("word").count()
```

Let's Get Familiar with an Example (3/4)

- **writeStream()** is the Structured Streaming counter part to **readstream()** method
 - In SparkSession we had spark.**read()** and spark.write
 - We set the output mode to produce an output for the complete stream of data so far
 - Choose and output format - here we are sending to the console

```
# Start running the query that prints the running counts to the console
query = wordCounts \
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .start()

query.awaitTermination()
```

Input

Let's Get Familiar with an Example (4/4)

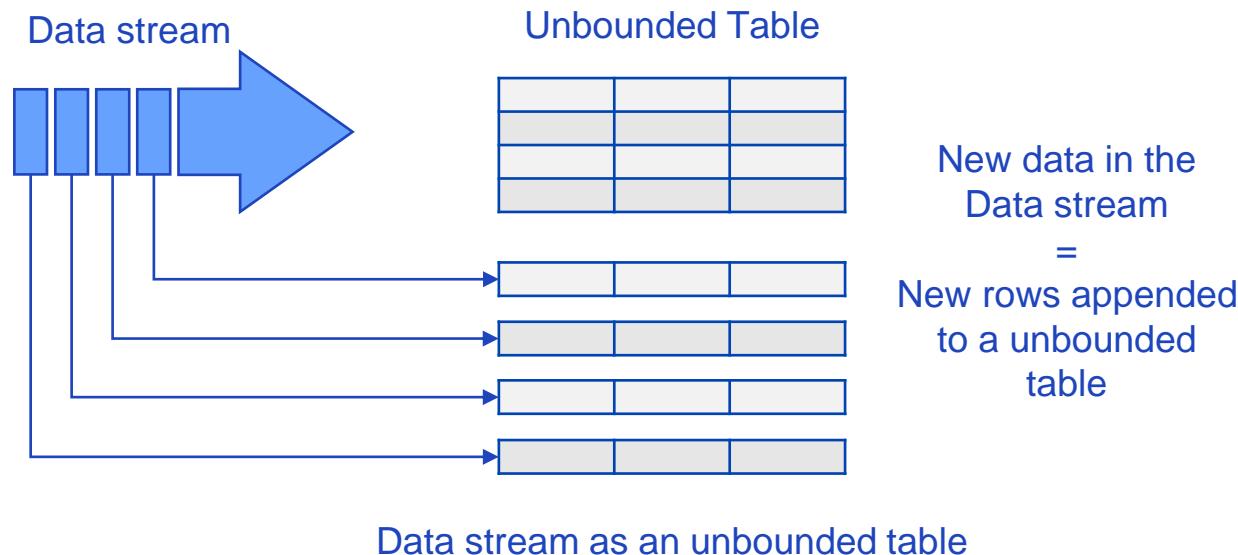
- Observing the output gives us insight to how Structured Streaming micro-batches work
- As in Spark Streaming, Structured Streaming also processes data in micro-batches
 - Spark Streaming sets a time interval for each batch
 - Setting a time interval is possible using `trigger()` method
 - In the default methodology, a new batch consists of all the rows that have arrived and accumulated since processing the last batch
- In Batch 0, there is usually no output
 - There hasn't been any accumulated data since we just started
- In Batch 1, we see output results
 - This is based on the new rows received and accumulated since "finishing" Batch 0

Batch: 0	
word	count

Batch: 1	
word	count
enough;	2
corners;	1
spoke;	1
waters	1
high:	3
knot!"	1
pack,	1
youth,	3
sisters,"	1
still	12
1.F.3.,	3

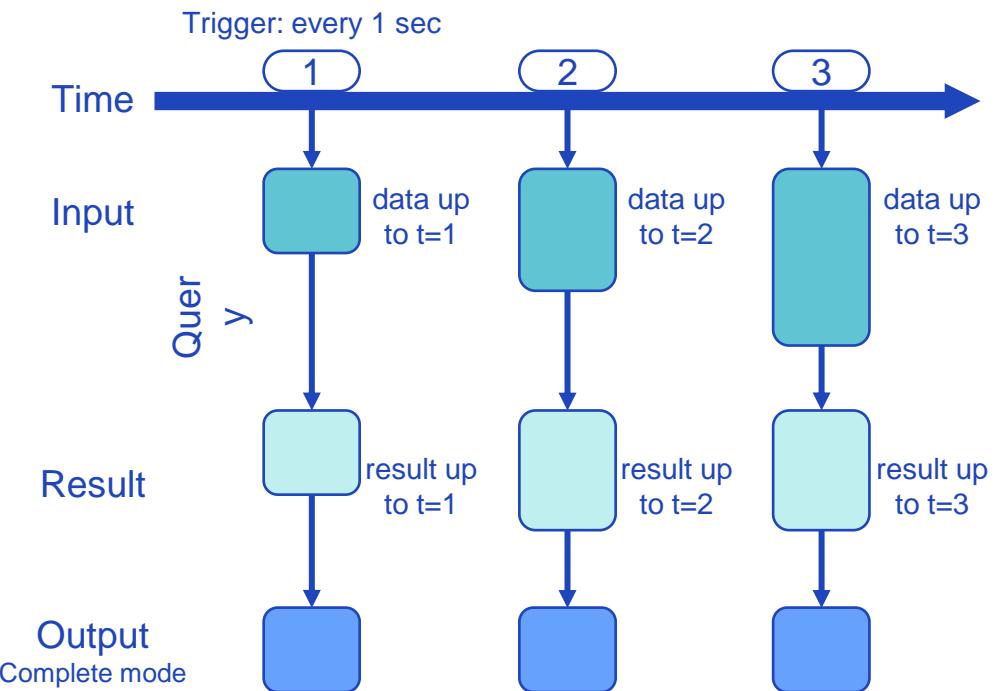
Structured Streaming Data Organization

- Structured Streaming organizes data as an unbounded table
 - Data continuously accumulates and is appended to the table
- In the DataFrame API, each dataframe is viewed as a bounded table



Programming Model

- As in Spark Streaming, Structured Streaming also processes data in micro-batches
 - As we saw in the example code, the default batch mode is to process all the accumulated rows since starting last batch
 - Spark Streaming also allows time intervals to be set for each batch
 - Setting a time interval is possible using the **trigger()** method



Output Modes

- Output is defined as what is stored to the external storage
 - Default output mode is Append mode

Output Mode	Meaning
Complete	The entire result is stored to the external storage
Append	Only new rows appended to the result since the last micro-batch is stored to external storage. This mode can only be used when existing rows are not allowed to change
Update	Only rows that have been either updated or appended since the last micro-batch is stored to external storage.

Illustration of Programming Model (1/2)

- In our example code, we generate the lines DataFrame by reading the input table, transform it into words, and produce the results table in wordCounts through the **count()** action
- These sets of operations is the same as if performed on a static table
 - In fact we can view the data in our current micro-batch as a static table

Input

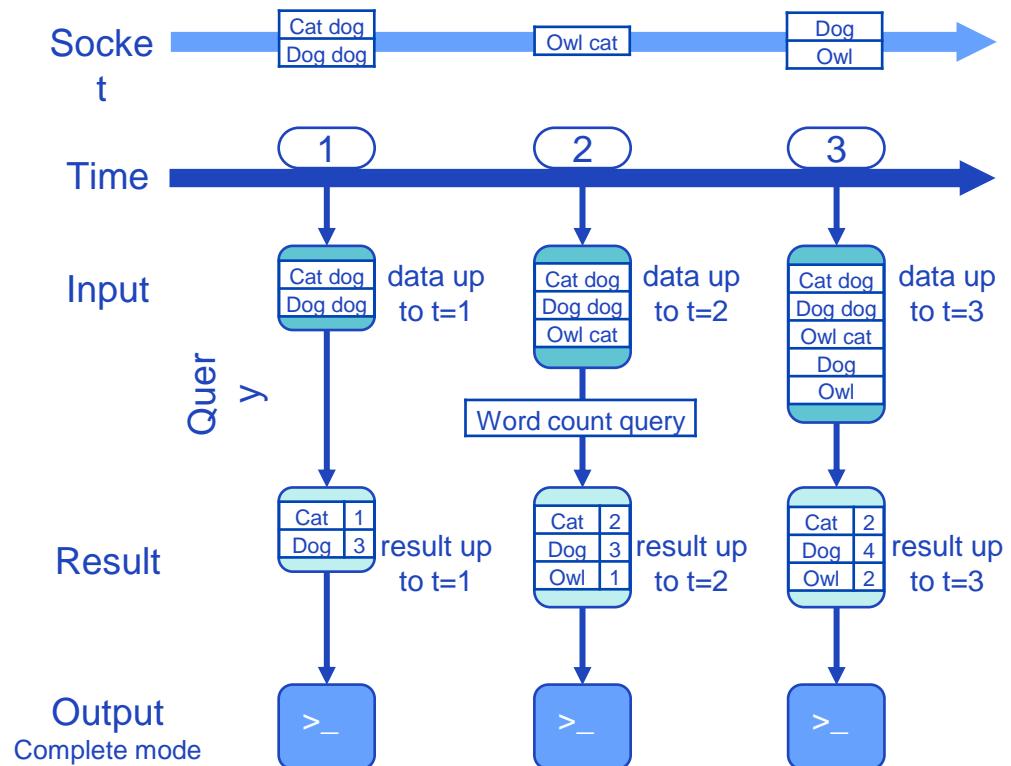
```
# Read the input table
lines = spark.readStream . . . . . .

# Split the lines into words
words = . . . . . .

# Generate running word count
wordCounts = words.groupBy("word").count()
```

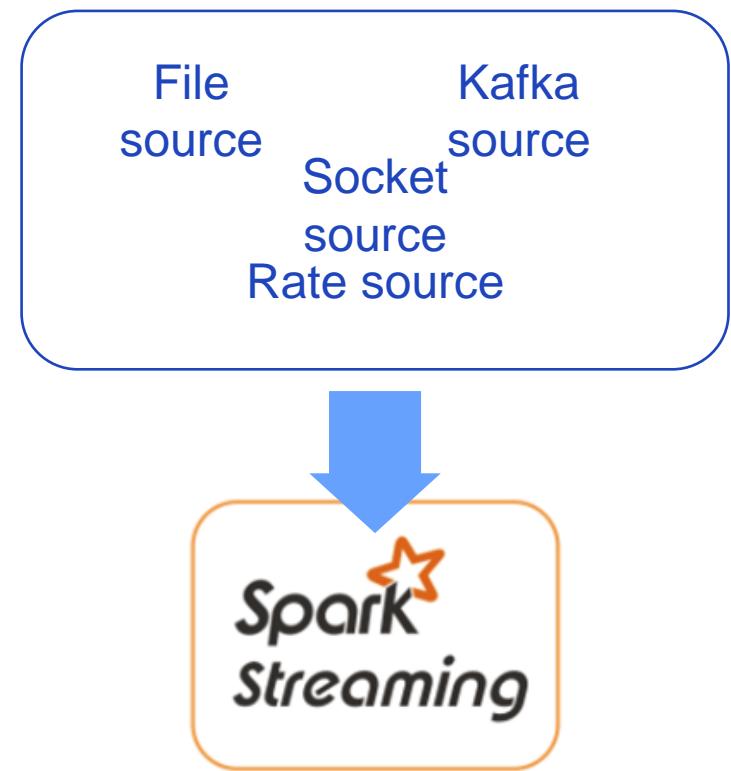
Illustration of Programming Model (2/2)

- Where Structured Streaming differs is that it continues to check for new rows in the input table AFTER the query has begun
- Structure Streaming runs an incremental query and combines with the previous results table
- Spark does not keep in memory the entire input table
 - It processes each batch incrementally and deletes any unnecessary data
 - Only keeps as much state as necessary
- Developer, however, does not need to keep track of any of this



Structured Streaming Data Sources

- Built-in data sources
- File source
 - Read files from a file system in a specified path
 - Supported formats are plain text, CSV, JSON, ORC and Parquet
- Kafka source
 - Get messages from Kafka topic
 - Only supported in Scala and Java in Spark 3.x
- Socket source - primarily for testing
 - Reads UTF-8 encoded text from a Linux socket at designated Port
- Rate source - primarily for testing
 - Generates data at a specified rows per second rate
 - Each row contains a timestamp and value of long datatype



Schema for Streamed Data (1/2)

- In the DataFrame API, making sure that we had the correct schema for the DataFrame was an important part of the development
 - In certain situations, schema could be inferred
- In Structured Streaming, schema must be provide for all file based data sources
 - Schema must be specified for data sources that have structure such as ORC, Parquet, CSV and JSON

```
from pyspark.sql.types import *
from pyspark.sql.functions import *

inputPath = "src-path/my.json"

# Define the schema
mySchema = StructType([ StructField("time", TimestampType(), True),
                       StructField("action", StringType(), True)
                     ])

streamingDF = spark.readStream.schema(mySchema).json(inputPath)
```

Input

Schema for Streamed Data (2/2)

- Kafka has a fixed schema
 - Key / binary - Where some custom producers insert additional information
 - Value / binary - The actual content of the Kafka message
 - topic / string - The Kafka topic
 - partition / integer - The Kafka partition number when a topic has partitions

Input

```
# Create a Kafka streaming dataframe
kafkaDF = spark.readStream.format("kafka") . \
    option("kafka.bootstrap.servers", "localhost:9092").option("subscribe",
"mytopic").load()

# Kafka datasources have a fixed schema that includes key and value columns as binary
# Use selectExpr() to execute SQL expressions without creating a temporary view
# Cast the key and value column to String type
kafkaDF.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
```

Operations on Streaming DataFrames

- Spark makes available most of the transformations from the DataFrames API to Structured Streaming
 - Untyped SQL operations such as **select**, **where** (filter), **groupBy**, **orderBy**
 - RDD like operations such as map, filter, flatMap
 - Spark actually considers both to be DataFrames and distinguishes them with the `isStreaming()` value
- ~~Register streaming dataframes and use SQL commands~~

Input

```
# Create streaming DF
streamingDF = spark.readStream . . . . .
print(streamingDF.isStreaming()) # Streaming is True

# Use DF transformations
countryDF = streamingDF.select("Country").where("UnitsSold >= 1000")

# Create temporary view and run SQL queries
streamingDF.createOrReplaceTempView("sales")
# This creates another streaming dataframe
goodDF = spark.sql(""" SELECT Country FROM sales WHERE UnitsSold >= 1000 """)
```

Example Reading CSV from Socket

- Here, we present reading a CSV string from a Linux Socket
 - Since the Socket returns a single column, we have to transform it to create schema

Input

```
# Create streaming DF from Socket source
csvDF = spark.readStream.format("socket") \
    .option("host", "localhost").option("port", "44444").load()

# Text is in CSV format. Split string and use withColumn to create new columns
from pyspark.sql.functions import *
salesDF = csvDF
    .withColumn("Country", split(csvDF.value, ",") [0]) \
    .withColumn("UnitsSold", split(csvDF.value, ",") [1].cast("integer")) \
    .withColumn("UnitPrice", split(csvDF.value, ",") [2].cast("integer"))

# Run queries or transformations
bestDF = salesDF.select("Country").where("UnitsSold >= 1000")
```

Aggregation on Streaming DataFrames

- In DataFrames, **groupBy()** creates GroupedData objects which can then be aggregated with many available transformations such as count, max, min, and the agg(aggregation function) wrapper
 - A column is defined on which to group the data
- All non-streaming dataframe aggregations work with streaming dataframes
 - By default, results are bases on all records received, up to the current micro-batch

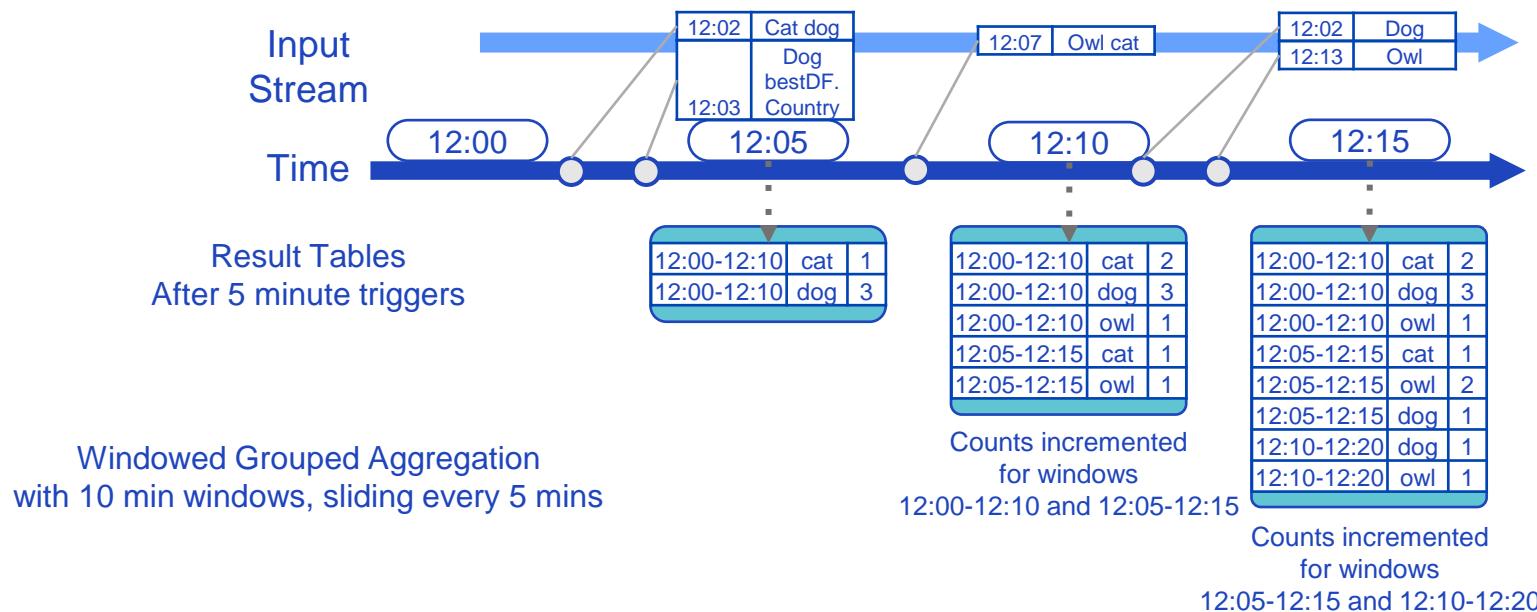
Input

```
# Create streaming DF from Socket source
csvDF = . . . .
# Transform and assert schema on text
salesDF = csvDF. . . .
# Select countries that have transaction of 1000 units sold or more
bestDF = salesDF.select("Country").where("UnitsSold >= 1000")

# Group by Country and count how may times such transactions occurred
bestDF.groupBy(bestDF.Country).count()
```

Aggregation with Windows

- Structured Streaming can also aggregate over sliding Windows based on some event time
 - From the wordcount example, imagine that the streaming data has an addition timestamp column
 - Use `window(<timestamp column>, <window length>, <slide interval>)` transformation



Code for Window Aggregation

- The window() method generates a string with the start of window and end of window timestamps
 - "[2021-09-05 01:35:00, 2021-09-05 01:45:00]"
- The groupBy method is given two columns to group the records by
 - Equal word column
 - Equal "window time" string column

Input

```
# Create streaming dataframe from (word, timestamp) source
schema = "word string, time timestamp"
wordAt = spark.readStream . . . . .

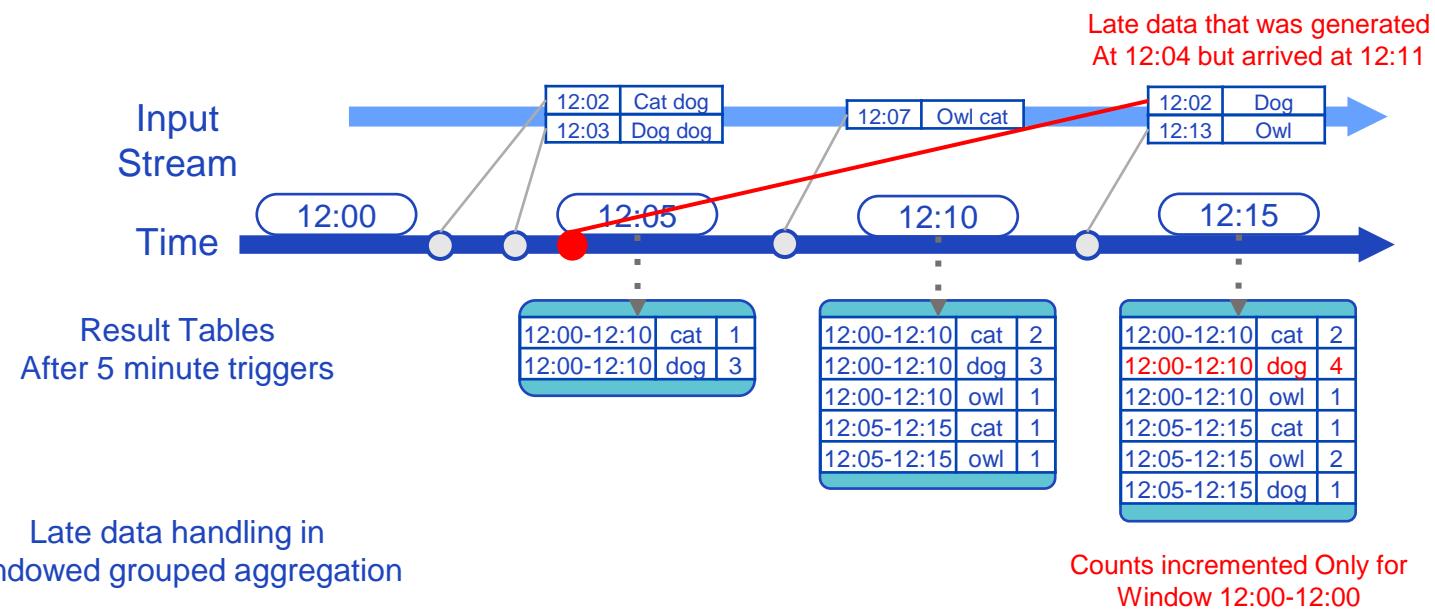
# Group by word ad timestamp over a window of 10 minutes, every 5 minutes
windowWC = wordAt.groupBy(
    window(col("time", "10 minutes", "5 minutes"),
    col("word")) \
    .count()
```

Dealing with Late Arrivals

- Unfortunately, data does not always arrive on time
 - Various causes including temporary server down, busy network traffic, etc. can cause delays
- What happens in Window aggregation when a record with a timestamp before the current window?
 - In other words, it should have been counted on a previous window
- The "window" of opportunity has "almost" passed and we can't change the past or can we?
- Spark allows developers to define Watermarks
 - A Watermark defines how late a record can arrive before it is disregarded
 - For records, that arrive within the watermark, old records are fixed

Keeping State For Watermarks

- Spark will keep in memory, old states whose watermark is still valid
 - Still allowed to update and fix past data



Window Aggregation with Watermarks

- Before grouping, set watermark to let Spark know how long states must be maintained
 - Use `withWatermark(<timestamp column on which we are grouping>, <duration of watermark>)`

Input

```
# Create streaming dataframe from (word, timestamp) source
schema = "word string, time timestamp"
wordAt = spark.readStream . . . . .

# Group by word ad timestamp over a window of 10 minutes, every 5 minutes
# Set watermark to 10 minutes for the time timestamp column
windowWC = wordAt.withWatermark("time", "10 minutes") \
    groupBy(window(col("time", "10 minutes", "5 minutes"),
                  col("word")) \
    .count()
```

[Lab11] Working with the DStream API



Chapter 7.

Big Data Modeling & AI

Big Data Course

Chapter Description

Objectives:

- ✓ Machine learning is a core field of artificial intelligence and is being used in various fields.
- ✓ We will learn a basic knowledge of machine learning and related algorithms and public cloud machine learning platform. In particular, it performs data preprocessing using Spark learned in the previous chapter.
 - Machine learning workflow with dataset
 - Machine learning for algorithm for classification, regression, clustering
 - SageMaker in AWS and Azure machine Learning in Azure
- ✓ Learn how to create Apache Spark ML/MLlib Transformers, Estimators and Pipelines and utilize standardized machine learning algorithms provided within the API to create data models.
- ✓ Finally, we use regression-based algorithms leveraging Spark ML to gain insight into how classification is performed, understand several underlying algorithms and use them to create models.

Contents of Chapter:

1. Machine Learning & Model
2. Apache Spark Machine Learning Library

Unit 1.

Machine Learning & Model

Big Data Modeling and AI

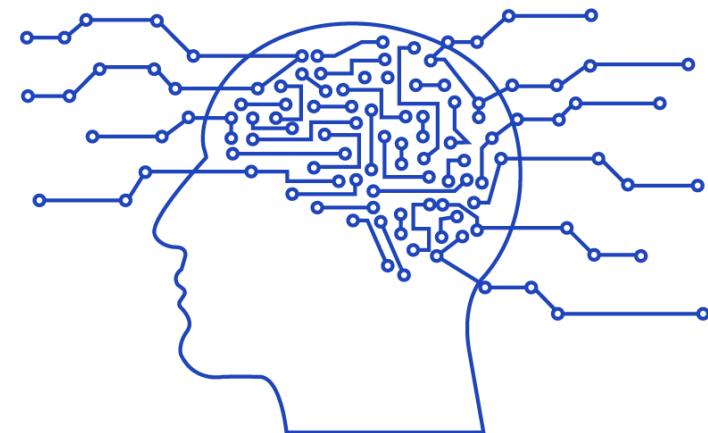
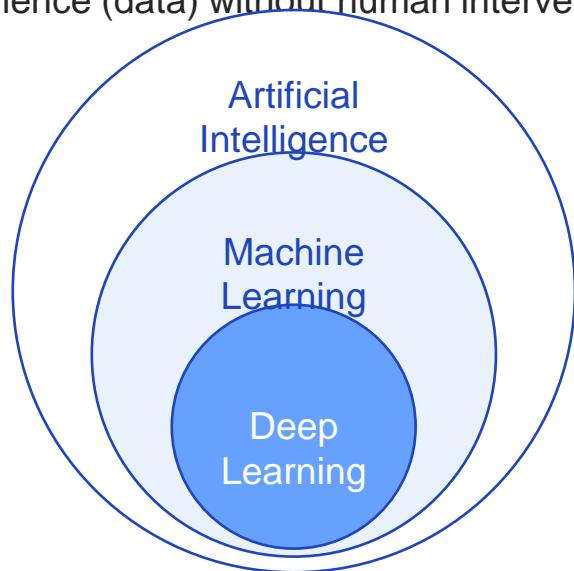
Unit 1

Machine Learning & Model

- | 1.1. Machine Learning Basic
- | 1.2. Machine Learning in Public Cloud
- | 1.3. Apache Spark ML

What is Machine Learning? (1/2)

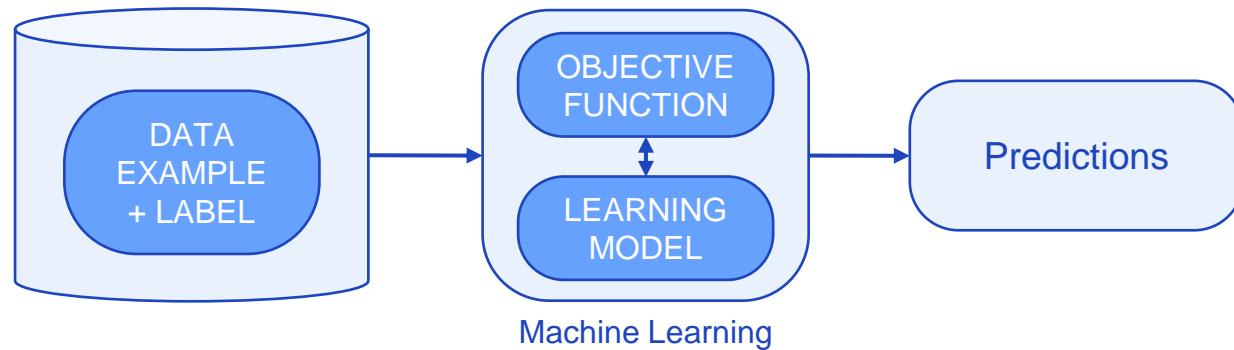
- Algorithms and technologies that enable computers to learn as a branch of artificial intelligence
- AI field based on data learning
- Algorithms and technologies that allow computers to automatically learn specific tasks through accumulated experience (data) without human intervention



Machine
Learning

What is Machine Learning? (2/2)

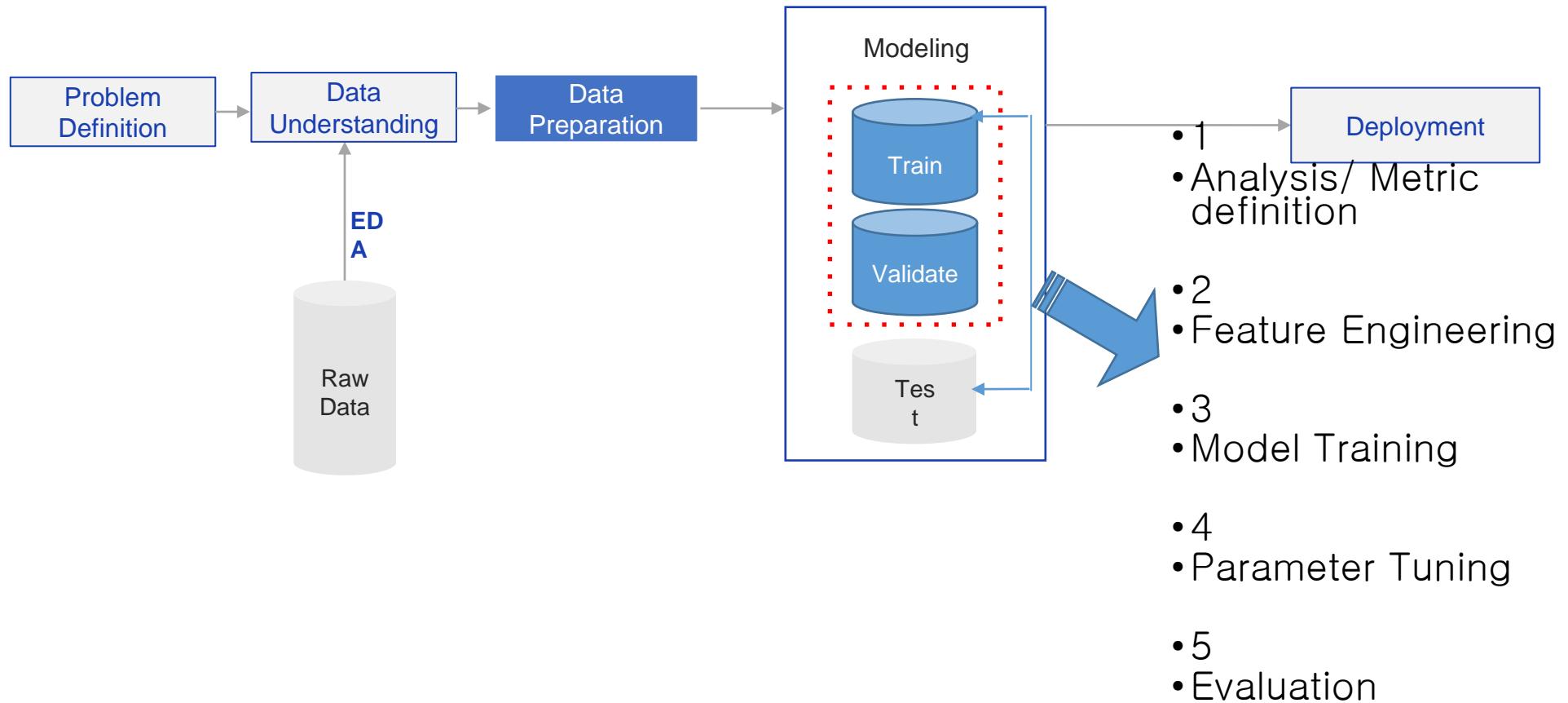
- Learning
 - The task of creating a model through a specific algorithm that best describes the relationship between input and output in data



- Example: Data entering the training model as input
- Label: the correct answer for the EXAMPLE
- Learning model: describe the correlation between input and output
- Objective function: algorithms for training the model

Machine Learning workflow (1/2)

- Problem Definition -> Data Understanding -> Data preparation -> Modeling -> Evaluation -> Deployment



Machine Learning workflow (2/2)

- The process of creating a model
 - 1. Determine the model and formulate it
 - 2. Train the model using the data. And fit the model to a pattern in the data
 - 3. Evaluate how well the trained model represents the data pattern (test)

$$Y = f(x)$$

- Y(Label) is the output variable or dependent variable
- X(Feature) is the input variable or independent variable
- f is the model

Terminology and definitions (1/3)

- Model
 - A model means a program
 - A function that formulates a pattern in the data
 - A simplified representation of the real world to achieve a purpose
- Learning
 - Learning refers to the technology in which computers find rules in data by themselves
- Training
 - Training is literally the process of implementing learning
- Input
 - An input is what the model (program) should look for rules
- target
 - The target means the correct answer that the model (program) must match

Terminology and definitions (2/3)

- Parameters:

- Learned from data by training and not manually set by the practitioner
- Contain the data pattern
- Ex) Coefficients of linear regression
- Ex) Weights of neural network

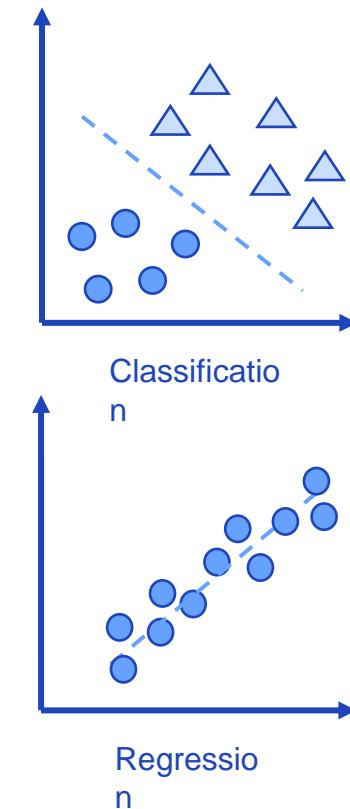
- Hyperparameters:

- Can be set manually by the practitioner
- Can be tuned to optimize the machine learning performance
- Ex) k in KNN algorithm
- Ex) Learning rate in neural network
- Ex) Maximum depth in Tree algorithm



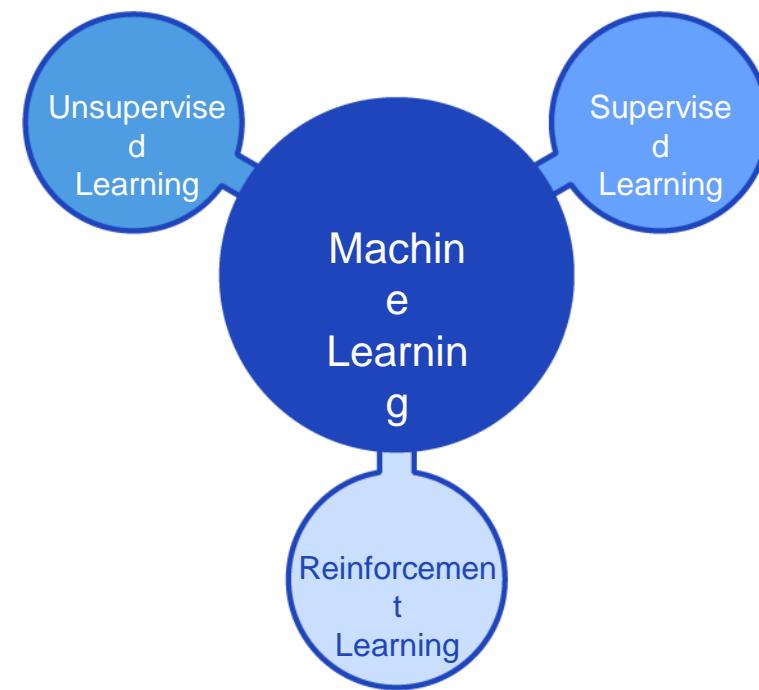
Terminology and definitions (3/3)

- Classification: A predictive model that approximates a mapping function from input variables to identify discrete output variables
 - Decision tree classification
 - Random forest classification
 - K-nearest neighbor
- Regression: A predictive model that estimate a mapping function based on the input and output variables.
 - Simple linear regression
 - Multiple linear regression
 - Polynomial regression
- The most significant difference between regression vs classification is that while regression helps predict a continuous quantity, classification predicts discrete class labels.



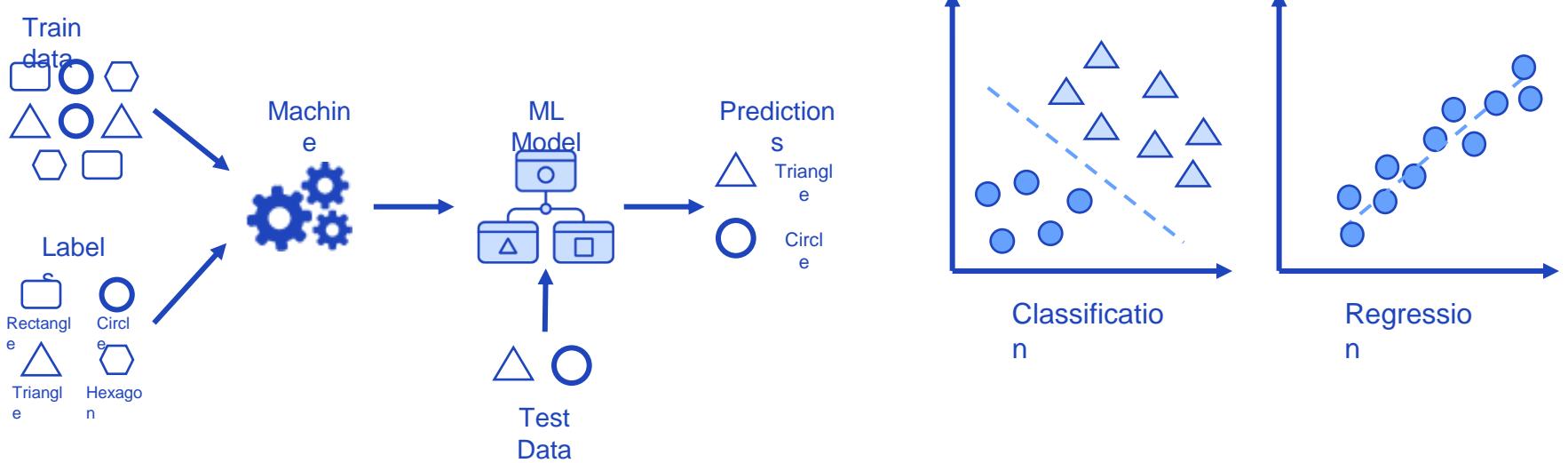
Types of Machine Learning

- Supervised
 - How to train a model using input data and corresponding correct answer data (label)
- Unsupervised
 - Learning only with input data without correct answer data
- Reinforcement
 - Enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences



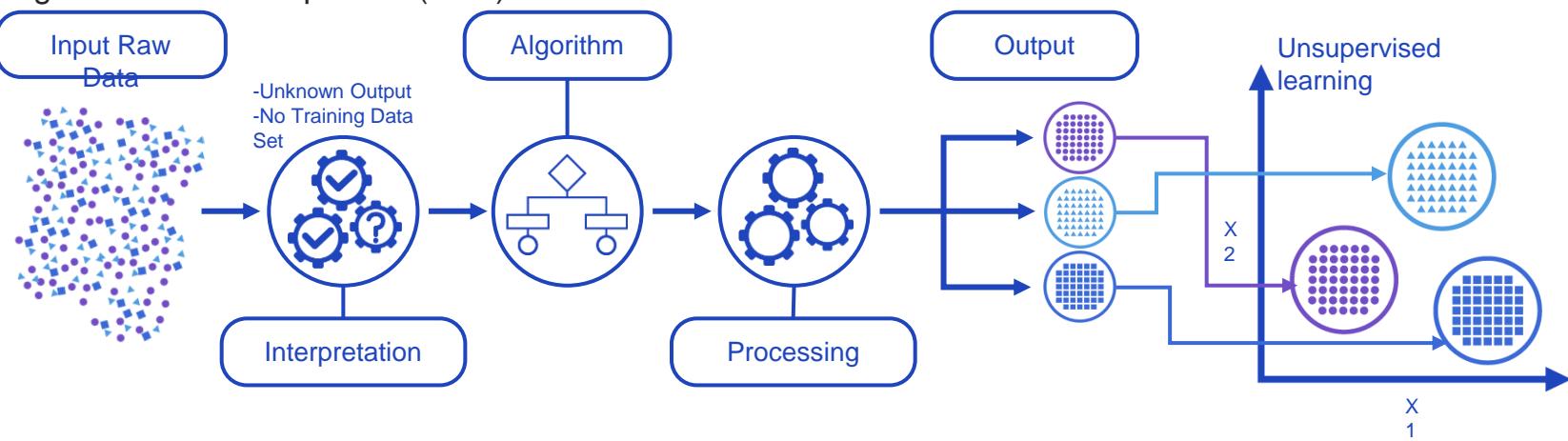
Supervised Learning

- The main purpose of supervised learning is to train a model on labeled training data to make predictions on unseen future data. Supervised here means a series of samples with the desired output signal (label)
 - Classification
 - Regression



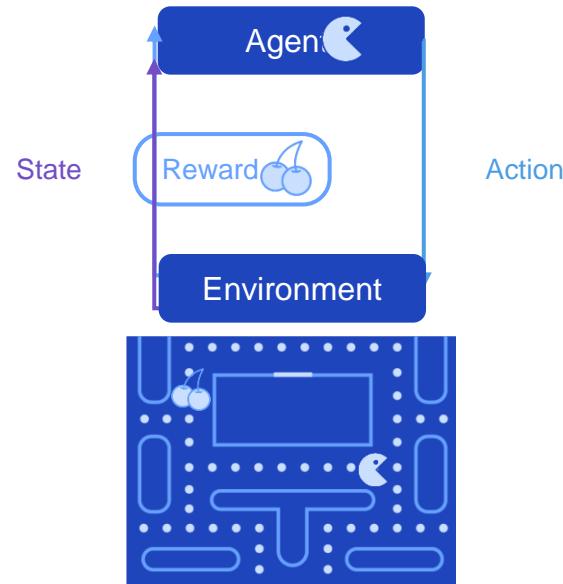
Unsupervised learning

- A method of training a model using training data without a target
- It deals with unlabeled or unstructured data. Create an algorithmic model of specific patterns and rules with no labeled data
 - Clustering
 - Principal component analysis(PCA)
 - Singular value decomposition(SVD)



Reinforcement learning

- Reinforcement learning aims to improve system (agent) performance by interacting with the environment
- Agents interact with their environment to learn a set of behaviors that maximize their reward through reinforcement learning
 - Exploratory trial and error method
 - Apply to game



Supervised vs Unsupervised learning

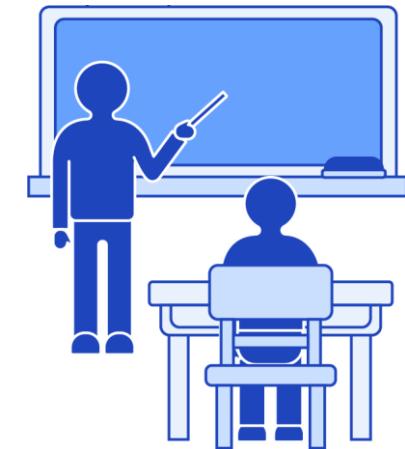
	Supervised	Unsupervised
Goals	The goal is to predict outcomes for new data. You know up front the type of results to expect.	The goal is to get insights from large volumes of new data. The machine learning itself determines what is different or interesting from the dataset.
Input Data	Algorithms are trained using labeled data.	Algorithms are used against data which is not labeled.
Complexity	Supervised learning is a simpler method.	Unsupervised learning is computationally complex.
Real Time Learning	Highly accurate and trustworthy method.	Less accurate and trustworthy method.
Accuracy of Results	Highly accurate and trustworthy method.	Less accurate and trustworthy method.
Drawbacks	Supervised learning models can be time-consuming to train, and the labels for input and output variables require expertise.	Unsupervised learning methods can have wildly inaccurate results unless you have human intervention to validate the output variables.

Which is best for you?

- Choosing the right approach for your situation depends on how your data scientists assess the structure and volume of your data, as well as the use case
 - Evaluate your input data
 - Define your goals
 - Review your options for algorithms
- The Classifying big data can be a real challenge in supervised learning, but the results are highly accurate and trustworthy
- The unsupervised learning can handle large volumes of data in real time. But, there's a lack of transparency into how data is clustered and a higher risk of inaccurate results
- Semi-supervised learning is a mixed medium, where you use a training dataset with both labeled and unlabeled data
 - It's particularly useful when it's difficult to extract relevant features from data — and when you have a high volume of data

Why use a Supervised learning?

- Supervised machine learning helps you to solve various types of real-world computation problems
- Supervised learning is typically done in the context of classification
 - when we want to map input to output labels, or regression
 - when we want to map input to a continuous output
- Supervised learning allows you to collect data or produce a data output from the previous experience
- Types of Supervised learning algorithms
 - Logistic regression
 - Naive Bayes
 - Support vector machine(SVM)
 - Random Forest
 - K-Nearest Neighbor(KNN)



Why use an Unsupervised learning?

- It is similar to how a human learns. It involves thinking by experiences, which moves it closer to real AI
- It works on unlabeled data, which makes unsupervised learning further critical as real-world data is mostly unlabeled
- Unsupervised machine learning finds all kind of unknown patterns in data
- It helps look for useful insights from the data
- Types of Unsupervised learning algorithms
 - K-means clustering
 - Hierarchical Clustering
 - Anomaly Detection
 - PCA
 - Apriori Algorithm



Which algorithm to choose? (1/2)

[Quiz]

Select an appropriate algorithm according to the characteristics and circumstances of the collected data.

- If you want to answer a YES|NO question, it is classification
- If you want to predict a numerical value, it is regression
- If you want to group data into similar observations, it is clustering

Case 1

- Customer segmentation: divide a customer base into groups of individuals that are similar in specific ways relevant to marketing, such as age, gender, interests, spending habits, etc..
- Market segmentation

Which algorithm to choose? (2/2)

[Quiz]

Case 2

- Sales / Stock forecasts
- Premiums on insurance based on different factors
- Equipment failure prediction

Case 3

- Spam Mail
- Customer ration analysis
- Pattern recognition: words, people, images, movements, etc.
- Fraudulent Transaction Detection

Types of Machine Learning

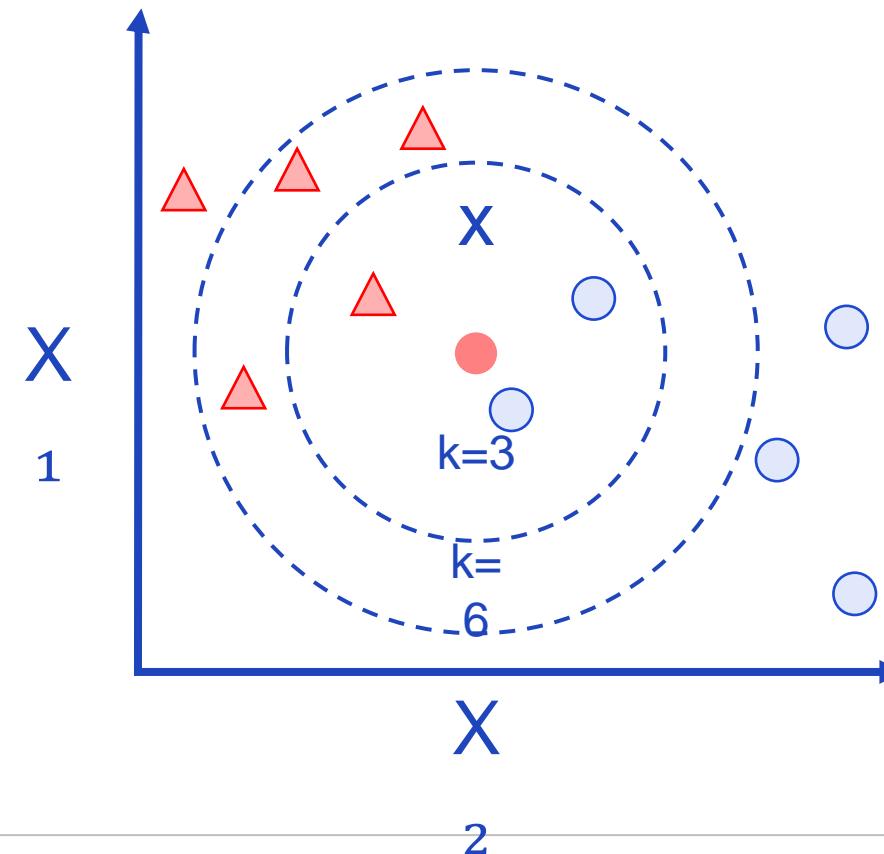
Type	Algorithm/Method
Unsupervised learning	Clustering
	MDS, t-SNE
	PCA, NMF
	Association analysis
Supervised learning	Linear regression
	Logistic regression
	Tree, Random Forest, Ada Boost, XGBoost.
	Naïve Bayes
	KNN
	Support vector machine (SVM)
	Neural Network

KNN Algorithm (1/4)

- KNN (K Nearest Neighbors)
 - The K-NN algorithm is an algorithm that classifies the labels of the data with the highest frequency among the K pieces of data by referring to the labels (attributes) of K pieces of other data that are close to each other
 - Prediction is based on the k nearest neighboring points
- Pros:
 - Simple and intuitive
 - No model parameters to calculate. So, there is no training step
- Cons:
 - Since there is no “model”, little insight can be extracted
 - No model parameters that store the learned pattern. The training dataset is required for prediction
- Prediction is not efficient “Lazy algorithm”

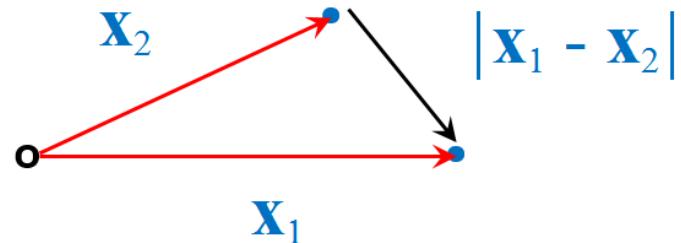
KNN Algorithm (2/4)

- K=3
 - X is classified as 
- K=6
 - X is classified as 



KNN Algorithm (3/4)

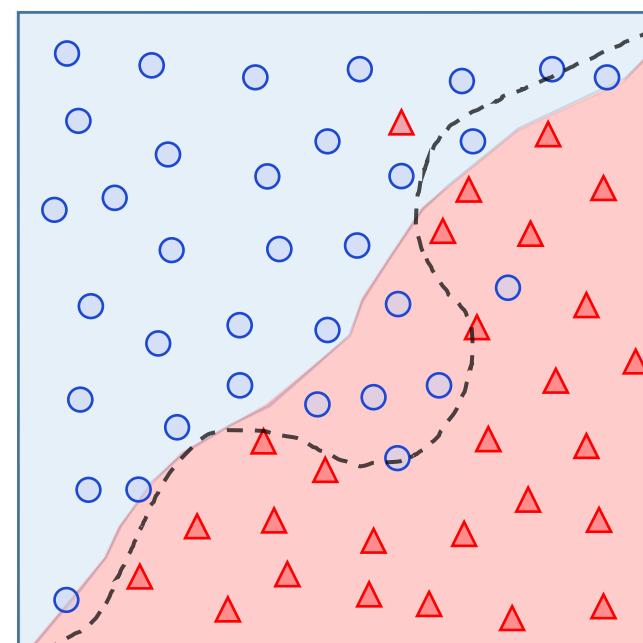
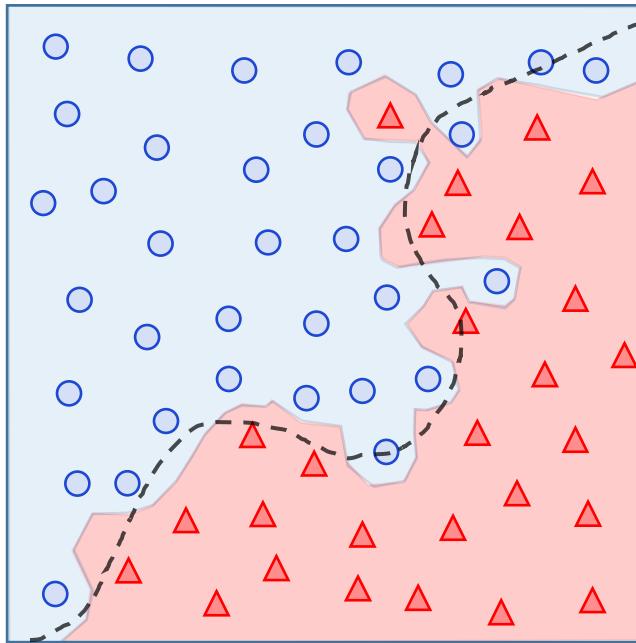
- Distance metric:
 - Euclidean distance: it is the modulus of the difference between two position vectors



$$|\mathbf{X}_1 - \mathbf{X}_2| = \sqrt{(x_{11} - x_{21})^2 + (x_{12} - x_{22})^2 + \dots + (x_{1d} - x_{2d})^2}$$

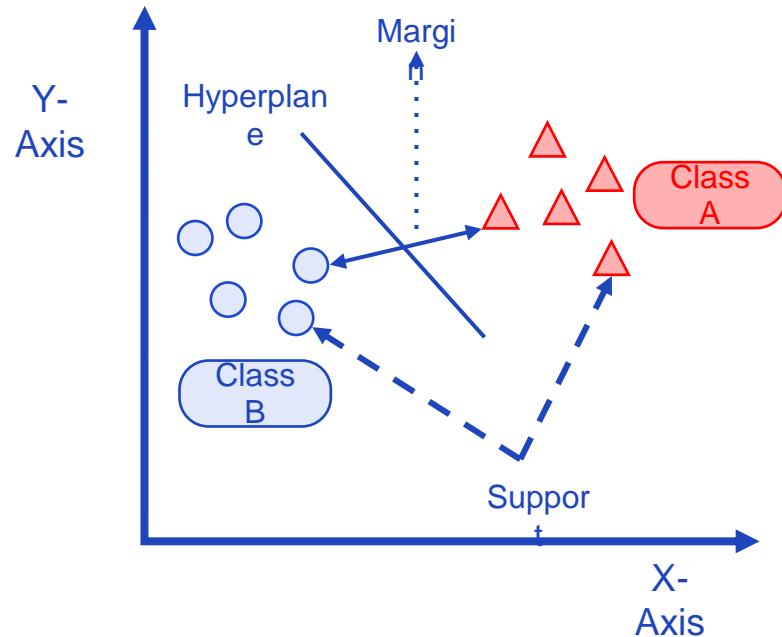
KNN Algorithm (4/4)

- When k is too small, overfitting may happen
- When k is too big, underfitting may happen



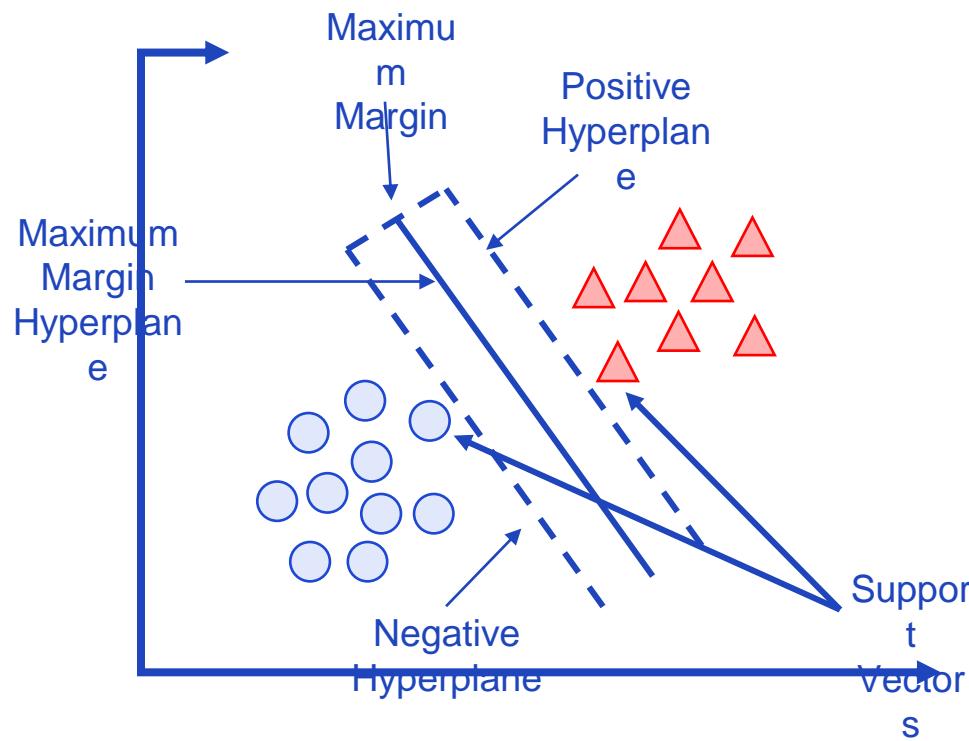
SVM Algorithm (1/5)

- SVM(Support vector machines) are a set of supervised learning methods used for classification, regression, and outliers detection
- To create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category
 - This best decision boundary is called a hyperplane
- Main element
 - Support Vectors
 - Hyperplane
 - Margin



SVM Algorithm (2/5)

- SVM chooses the extreme points/vectors that help in creating the hyperplane



SVM Algorithm (3/5)

Hyperplane

- ▶ For a k dimensional configurational space, the hyperplane has the dimension $k - 1$.
- ▶ For example,

For a two dimensional space, a hyperplane is a bisecting line that can be parametrized as

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

The two dimensional space is subdivided into two:

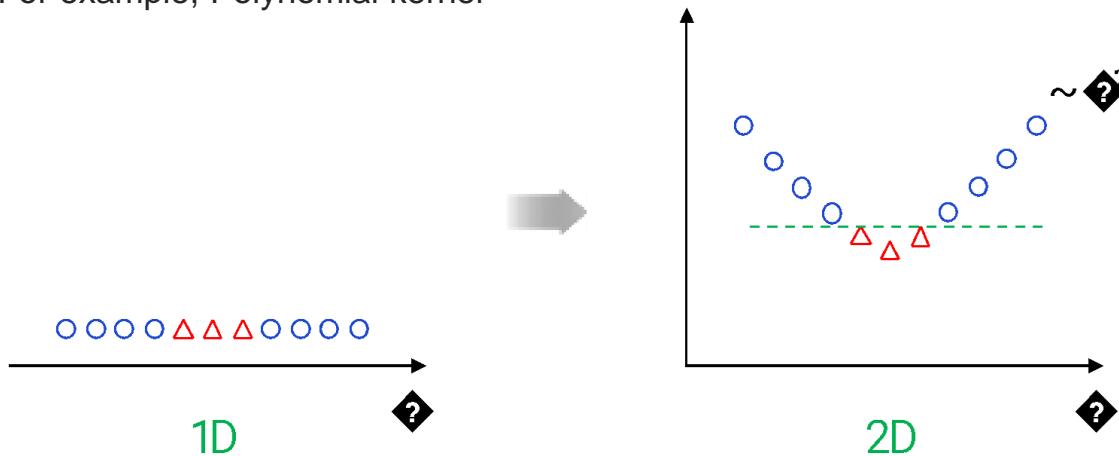
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$$

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 < 0$$

An observation would belong to either one of them \Rightarrow binary classification!

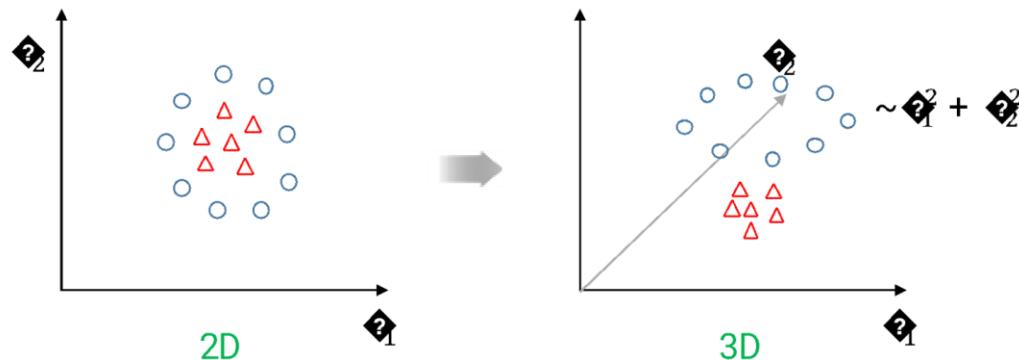
SVM Algorithm (4/5)

- Kernel
 - Mapping to a higher dimension using the “kernel” functions
 - Kernel functions introduce an effective non-linear classification boundary
 - For example, Polynomial kernel



SVM Algorithm (5/5)

- Kernel



- Pros:

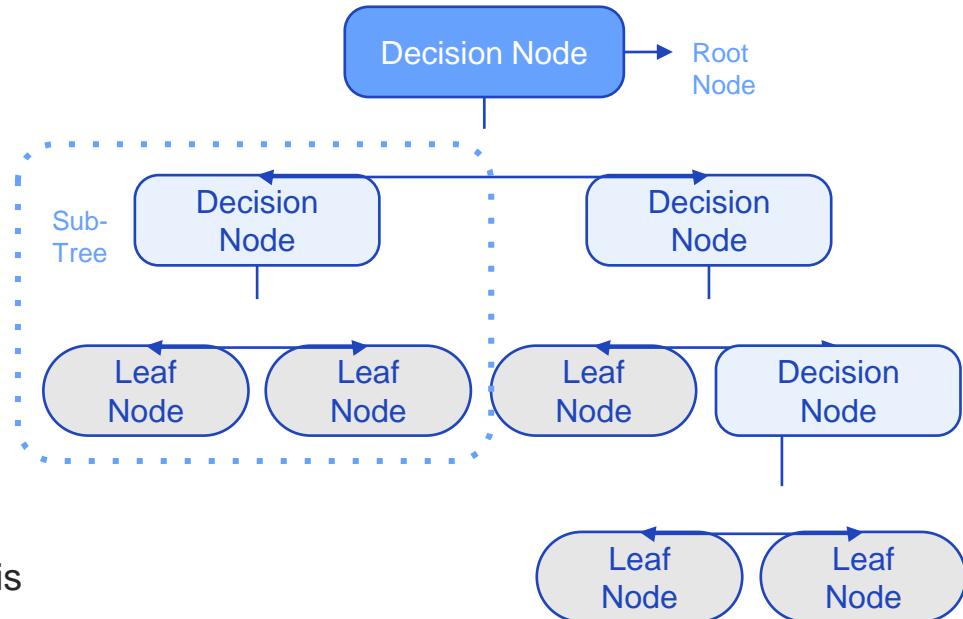
- Not very sensitive to the outliers
- Performance is good

- Cons:

- Training is relatively slow. Performs poorly for large data
- The kernel and the hyperparameter set should be carefully optimized

Decision Tree Algorithm (1/2)

- Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems
- There are two nodes
 - Decision nodes are used to make any decision and have multiple branches
 - Leaf nodes are the output of those decisions and do not contain any further branches
- The decisions or the test are performed on the basis of features of the given dataset
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees



Decision Tree Algorithm (2/2)

- Pros:

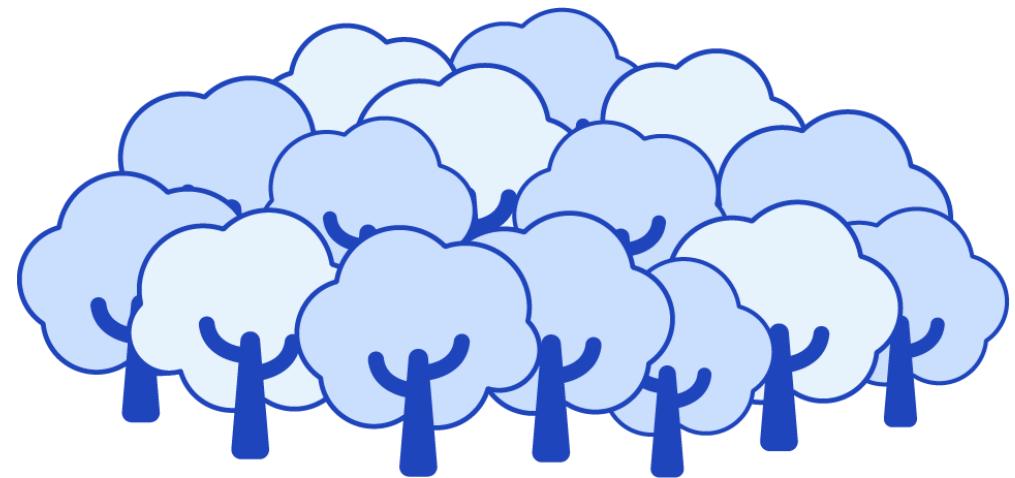
- Intuitive and easy to understand
 - No assumptions about the variables
 - No need to scale or normalize data
 - Not that sensitive to the outliers

- Cons:

- Not that powerful in the most basic form
 - Prone to overfitting. Thus, “pruning” is often required

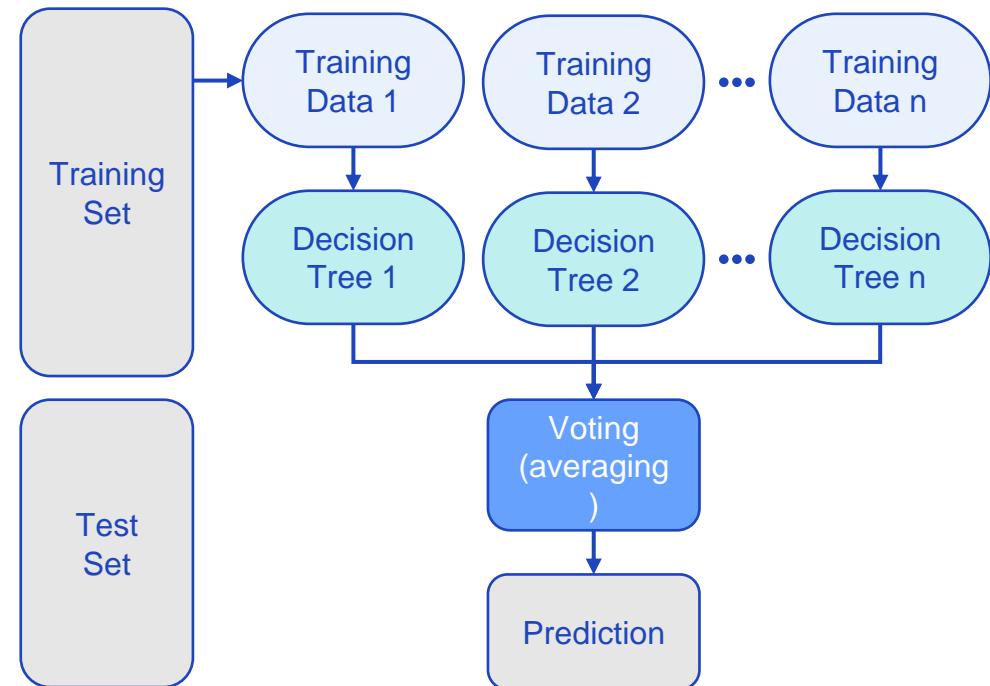
Random Forest Algorithm (1/3)

- Random Forest
 - Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique
 - An ensemble algorithm based on trees
 - Can be applied to classification and regression
 - Randomly chosen trees form a forest
 - Prediction is decided by the majority vote



Random Forest Algorithm (2/3)

- How does Random Forest algorithm work
 1. Make trees with the randomly selected variables and observations
 2. Keep only those that have the lowest Gini impurity (or entropy)
 3. Repeat from the step 1) for a given number of times
 4. Using the trees gathered during the training step, we can make predictions by majority vote



Random Forest Algorithm (3/3)

- Use cases
 - Banking: Banking sector mostly uses this algorithm for the identification of loan risk
 - Medicine: With the help of this algorithm, disease trends and risks of the disease can be identified
 - Land Use: We can identify the areas of similar land use by this algorithm
 - Marketing: Marketing trends can be identified using this algorithm
- Pros:
 - Powerful
 - Few assumptions
 - Little or no concern about the overfitting problem
- Cons:
 - Training is time consuming

Naive Bayes Algorithm (1/3)

- Naïve Bayes algorithm
 - Simple Multiclass Classification Algorithm
 - Assumes independence between every pair of features
 - In a single pass, it computes the conditional probability distribution of each feature given label
 - It then applies Bayes' theorem to compute the conditional probability distribution of label given observation and use it for prediction
 - It is a conditional probability-based classification method that calculates the feature probability that data belongs to each class.
 - It is a straightforward application of the Bayes' theorem

Naive Bayes Algorithm (2/3)

- Bayes' theorem
 - $P(A)$ is the probability of hypothesis A being true. This is known as the prior probability
 - $P(B)$ is the probability of the evidence(regardless of the hypothesis)
 - $P(B|A)$ is the probability of the evidence given that hypothesis is true
 - $P(A|B)$ is the probability of the hypothesis given that the evidence is there

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

- Naive Bayes Classifier
 - Naive Bayes is a kind of classifier which uses the Bayes Theorem. It predicts membership probabilities for each class such as the probability that given record or data point belongs to a particular class

Naive Bayes Algorithm (3/3)

- Types of Naive Bayes Algorithm
 - Multinomial Naive Bayes: It is used when the characteristics of the data are expressed as the number of appearances
 - Bernoulli Naive Bayes: A model that is suitable when the features of the data are expressed as 0 or 1
 - Gaussian Naive Bayes: It is suitable for calculating conditional probabilities under the assumption that the values of features are normally distributed, and for classifying data with continuous properties
- Pros:
 - Intuitive, simple, and Fast
 - Not that sensitive to the noise and outliers
- Cons:
 - Assumes that the features are independent which may not be strictly true
 - Not among the best performing algorithms

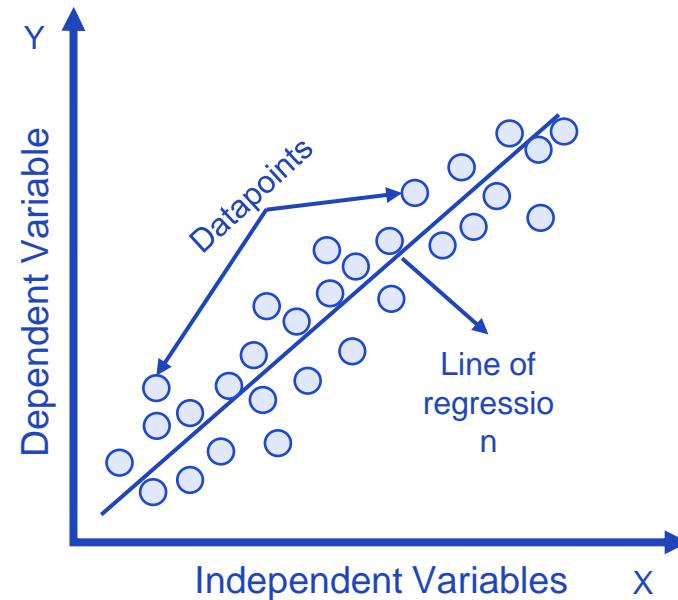
Linear Regression (1/5)

- I Linear regression is one of the easiest and most popular Machine Learning algorithms
- I Predicting the value of unobserved data by finding a function based on the observed data
 - ▶ There is one or more explanatory variables: X_1, X_2, \dots, X_k
 - ▶ There is one response variable: Y
 - ▶ The variables X_i and Y are connected by a linear relation:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_k X_k + \varepsilon$$

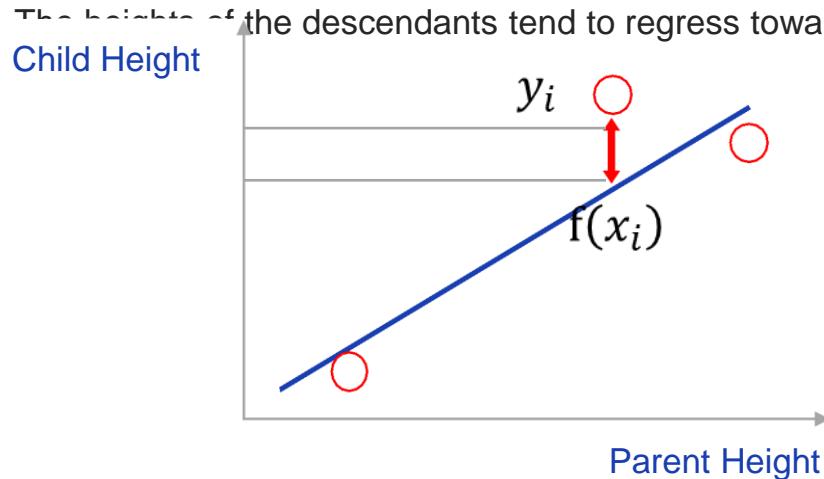
↑ ↑ ↑
Explanatory variables

Response variable



Linear Regression (2/5)

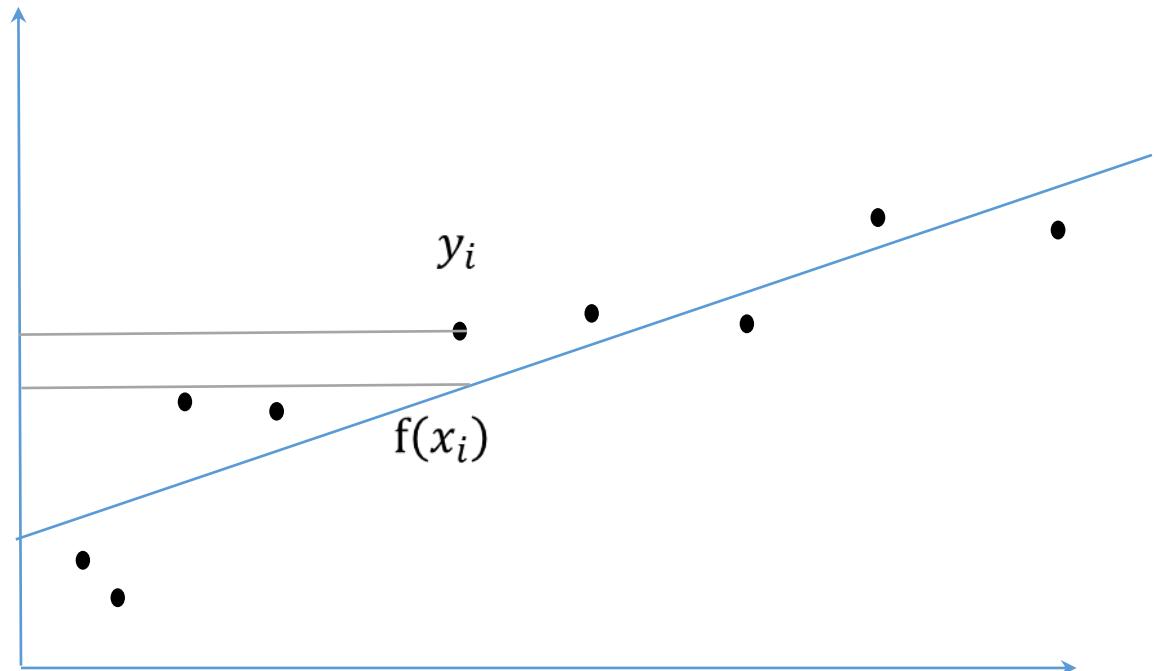
- Types of Linear Regression
 - Simple linear regression: Predict one outcome variable with one predictor variable
 - Multiple regression: Predict one outcome variable with multiple variables
- Historical background
 - Term “regression” coined by Francis Galton, 19th century biologist
 - ~~The heights of~~ the descendants tend to regress towards the mean



Linear Regression (3/5)

- | What is goal of function $f(x)$?
- | to minimize the difference between the actual label and the result obtained from our model
- | Want model to be as close to the data as possible

- ▶ minimize $|y_i - f(x_i)|$
- ▶ minimize $(y_i - f(x_i))^2$



Linear Regression (4/5)

Simple Linear Regression Optimization

- ▶ Choose β_0, β_1 such that Sum of Squared Error(SSE) is minimized
- ▶ Simple Linear Regression is an optimization problem where β_1, β_2 are chosen to minimize the SSE

$$f(x_i) = \beta_0 + \beta_1 * x_i$$



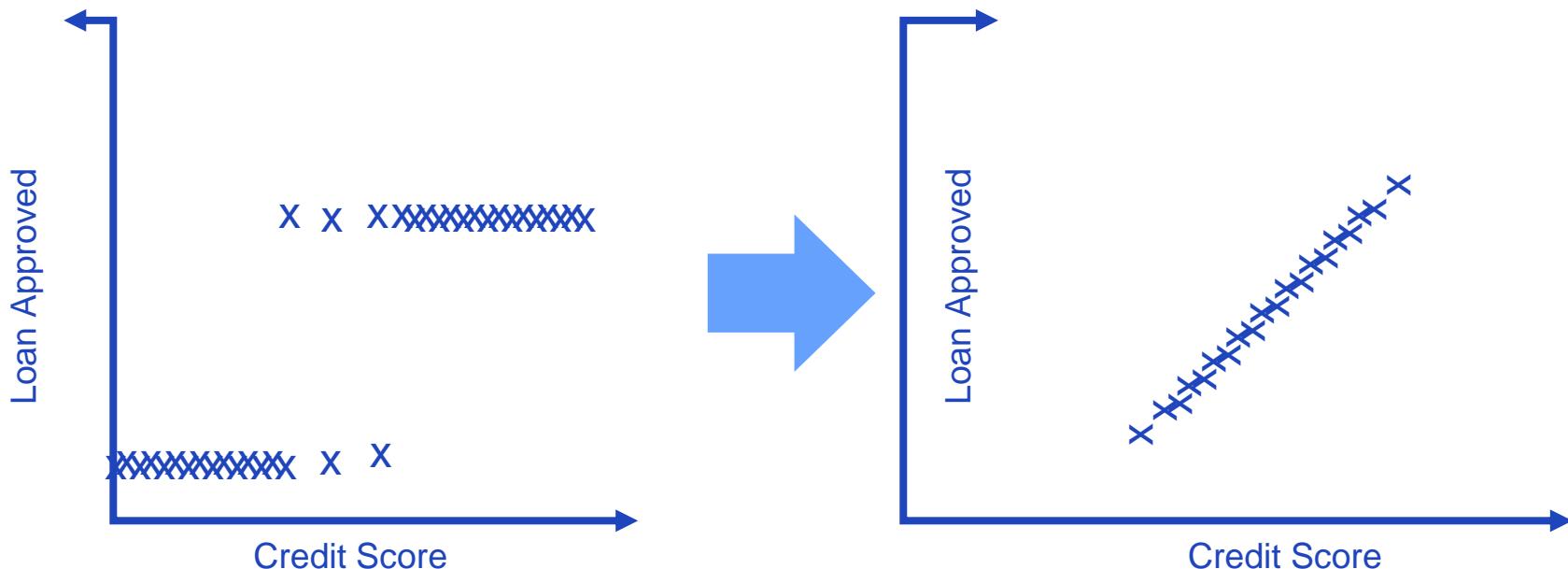
$$\text{Minimize } \text{SSE}(f) = \sum_{i=1}^n (y_i - f(x_i))^2$$

Linear Regression (5/5)

- Maps data into one straight line
 - There can be multiple straight lines that describe the data
- How to choose one straight line that best describes it:
- Least Square Method - Find a way to minimize the error
- Pros:
 - Solid statistical and mathematical background.
 - Source of insights
 - Fast training
- Cons:
 - Many assumptions: linearity, normality, independence of the explanatory variables, etc.
 - Sensitive to outliers
 - Prone to multi-collinearity

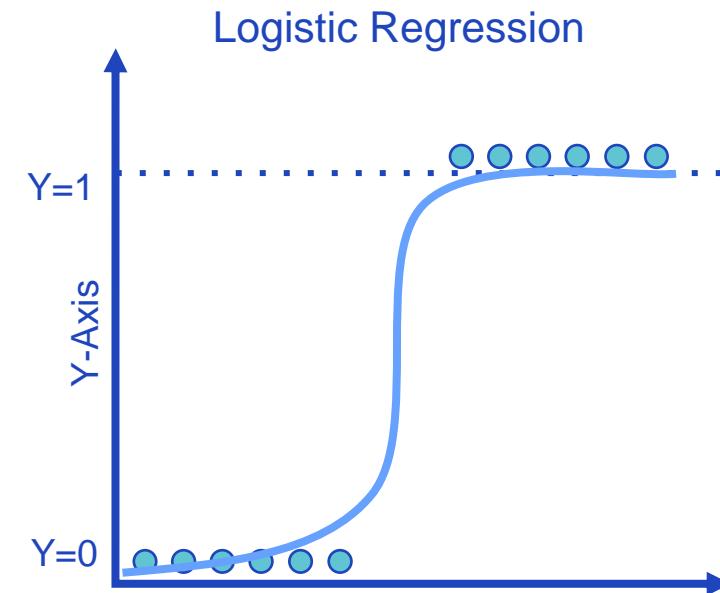
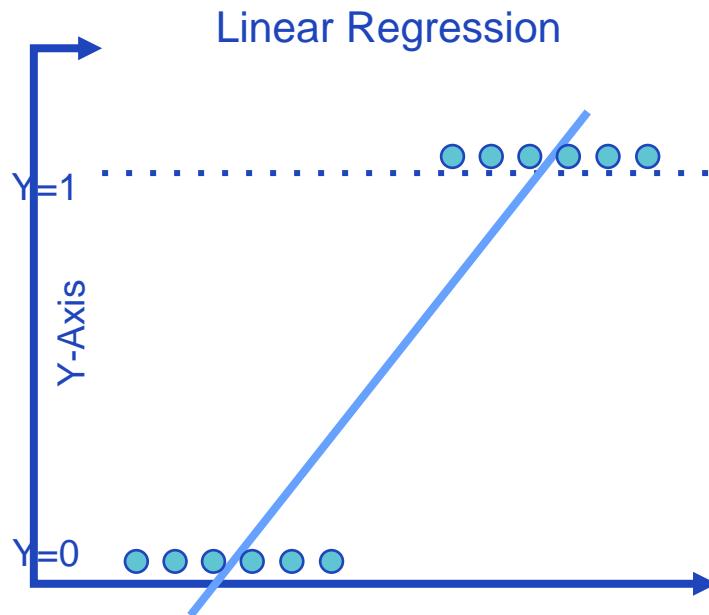
Motivation for Logistic Regression

- Attempts to convert a yes/no scatter to a linear one



Logistic Regression (1/6)

- Why logistic regression?
- Linear Regression vs Logistic Regression



Logistic Regression (2/6)

- Logit = Log Odds function
- The odds ratio is the probability of success/probability of failure

- $\log[p/(1-p)]$
- p = the probability of an event happening

- $1 - p$ = the probability of an event not happening

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = \log(p) - \log(1-p) = -\log\left(\frac{1}{p} - 1\right).$$

- By taking the logarithm of 0 to ∞ , it can be made to $-\infty$ to ∞
- The linear combinations of variables X_i is the “Logit” denoted here as S

Logistic Regression (3/6)

- Probability and Odds

$$\text{Probability} = \frac{\text{Number of Outcomes}}{\text{Total Possible Outcome}}$$

$$\text{Odds} = \frac{\text{Prob(occurring)}}{\text{Prob(not occurring)}}$$

	Probability	Odds
Roll a 6 on the dice		
Flip head on coin		
Get an Ace in card		

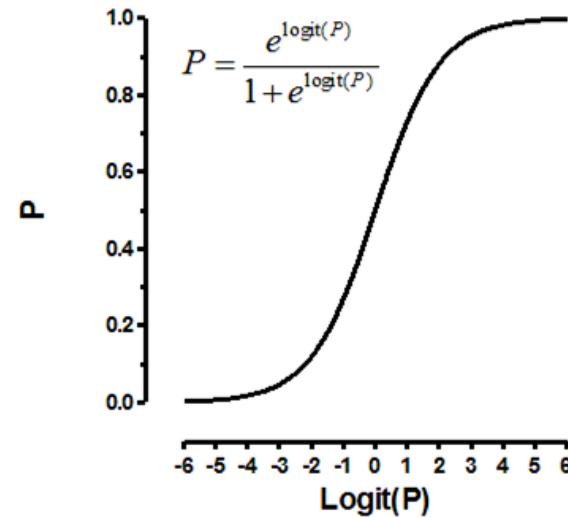
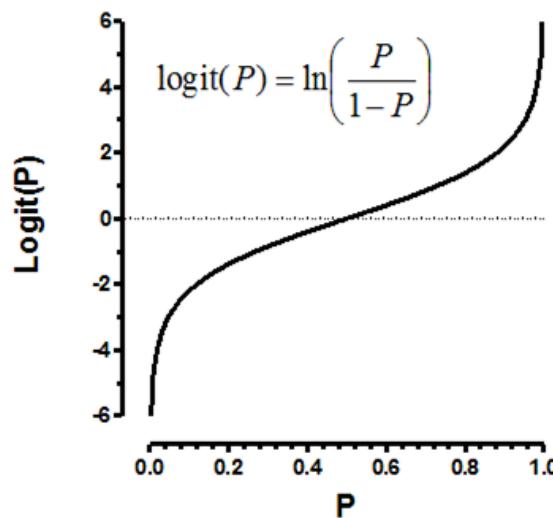
Logistic Regression (4/6)

- ↳ Logistic Regression to Bernoulli distribution
 - ▶ Estimating unknown p for any given linear combination of independent variables x_i .
 - ▶ The label y_i in logistic regression follows the Bernoulli distribution having an unknown probability, p
 - ▶ Bernoulli distribution is just a special case of the Binomial distribution when $n=1$
 - ▶ Success is 1 and failure is 0
 - ▶ Probability of success is p and probability of failure is $1-p$
 - ▶ We need a way to link the x_i features to Bernoulli distribution

Logistic Regression (5/6)

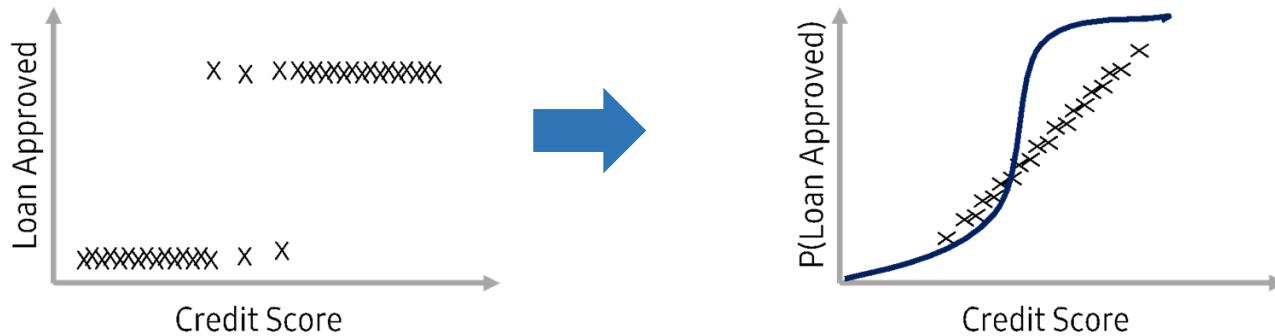
• The Logit Function

- ▶ Natural log of the odds ratio is the logit function
- ▶ Logit Function is $\ln(\text{odds})$ or $\ln\left(\frac{p}{1-p}\right)$ or $\ln(p) - \ln(1-p)$
- ▶ And Inverse shows the probability on the Y axis



Logistic Regression (6/6)

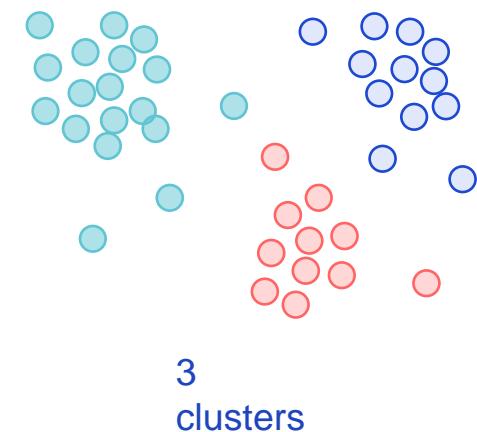
- Logistic Regression using Inverse Logit
 - Inverse Logit function allows us to convert to probability of y label



- Pros:
 - Simple and relatively easy to implement
 - Fast training
- Cons:
 - Not among the most accurate classification algorithms

K-Means Clustering (1/7)

- K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters
 - cluster refers to a collection of data points aggregated together because of certain similarities
 - Here K defines the number of pre-defined clusters that need to be created in the process
- K-means clustering
 - There is one ore more explanatory variables X_1, X_2, \dots, X_d
 - There is no response variable. Hence, this is a unsupervised learning algorithm
 - It arranges the unlabeled dataset into several clusters
- Purpose of the k-means clustering
 - Cluster the observations in k clusters
 - Find the centroids (cluster means) that characterize the clusters

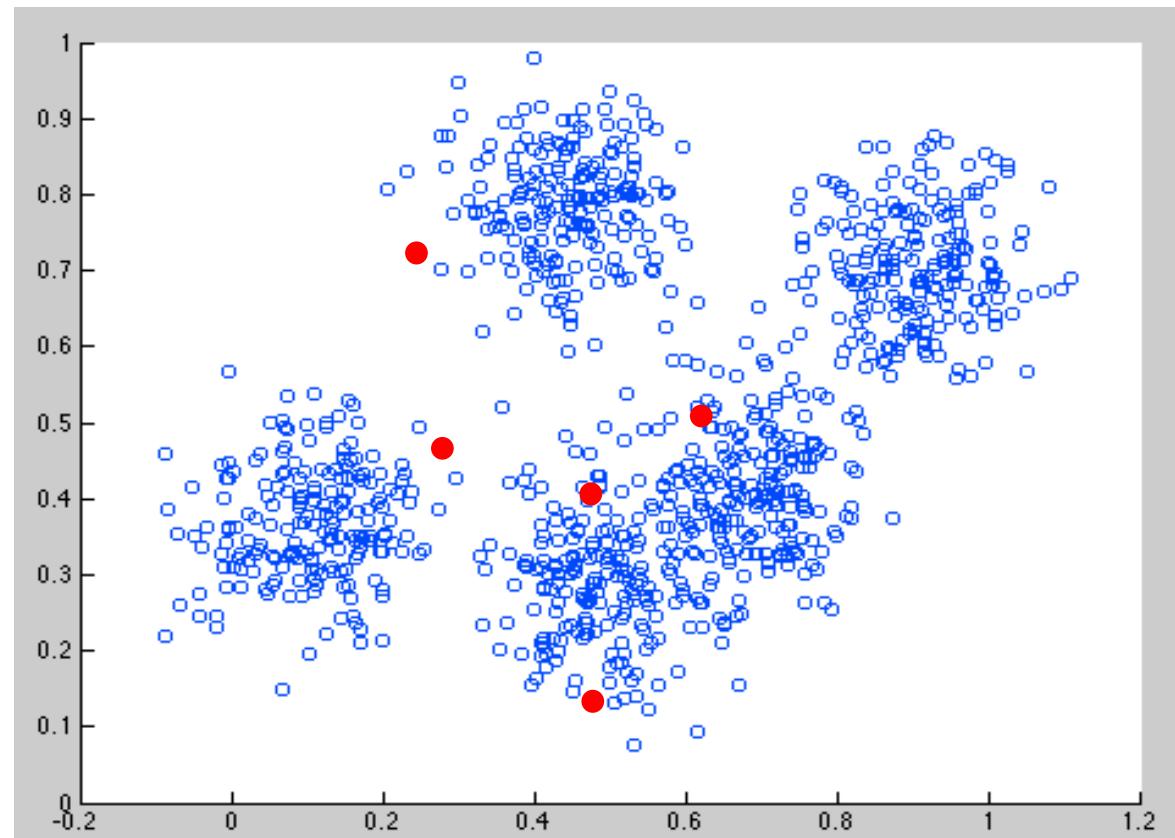


K-Means Clustering (2/7)

- How does the K-means algorithm work
 1. Choose the number K to determine the number of clusters
 2. Select arbitrary K points or centroids. (It can be different from the input dataset)
 3. Assign all data points to their nearest centroid. It will create the predetermined K clusters
 4. Calculate the variance and put a new centroid of each cluster
 5. Repeat the third step. Keep reassigning each data point to the latest cluster's closest centroid
 6. If any reassignment happens, then move to step-4; else, end
 7. Finally, model is ready

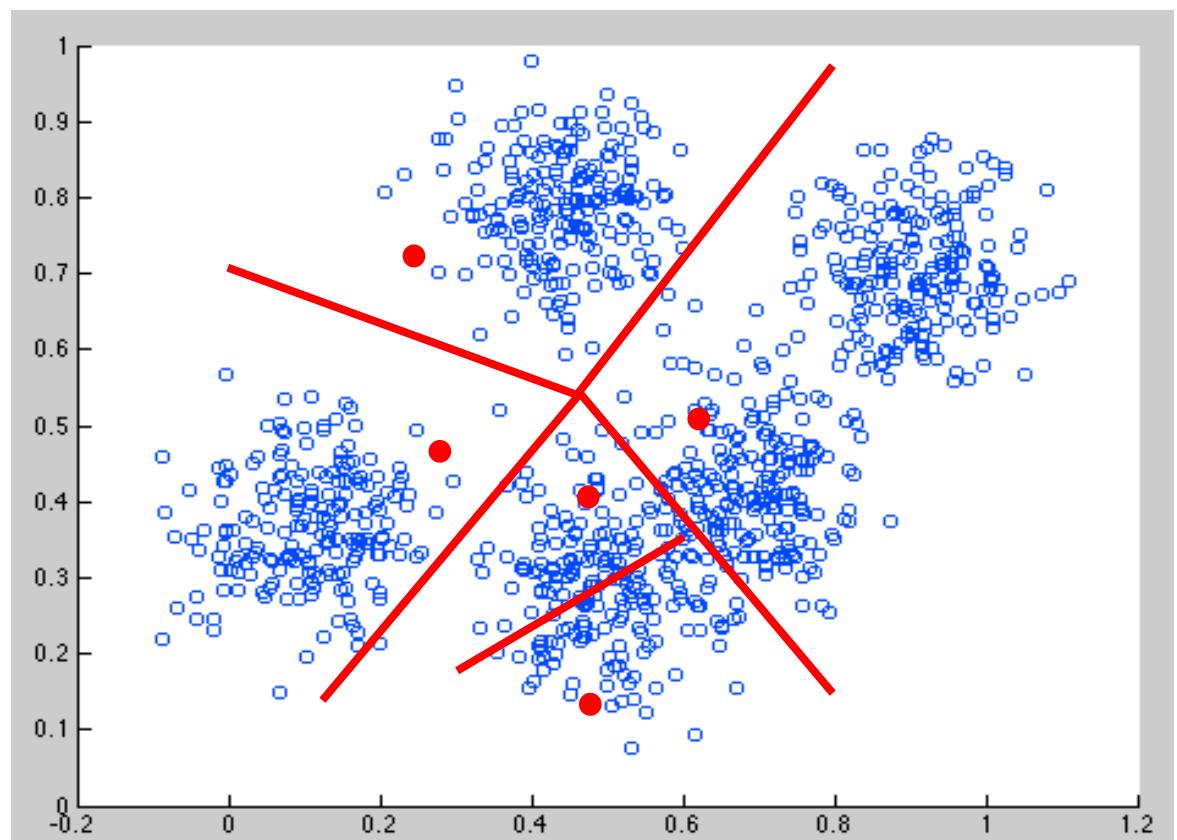
K-Means Clustering (3/7)

- Initial Centers
 - Input number of clusters K
 - Assign K centers randomly



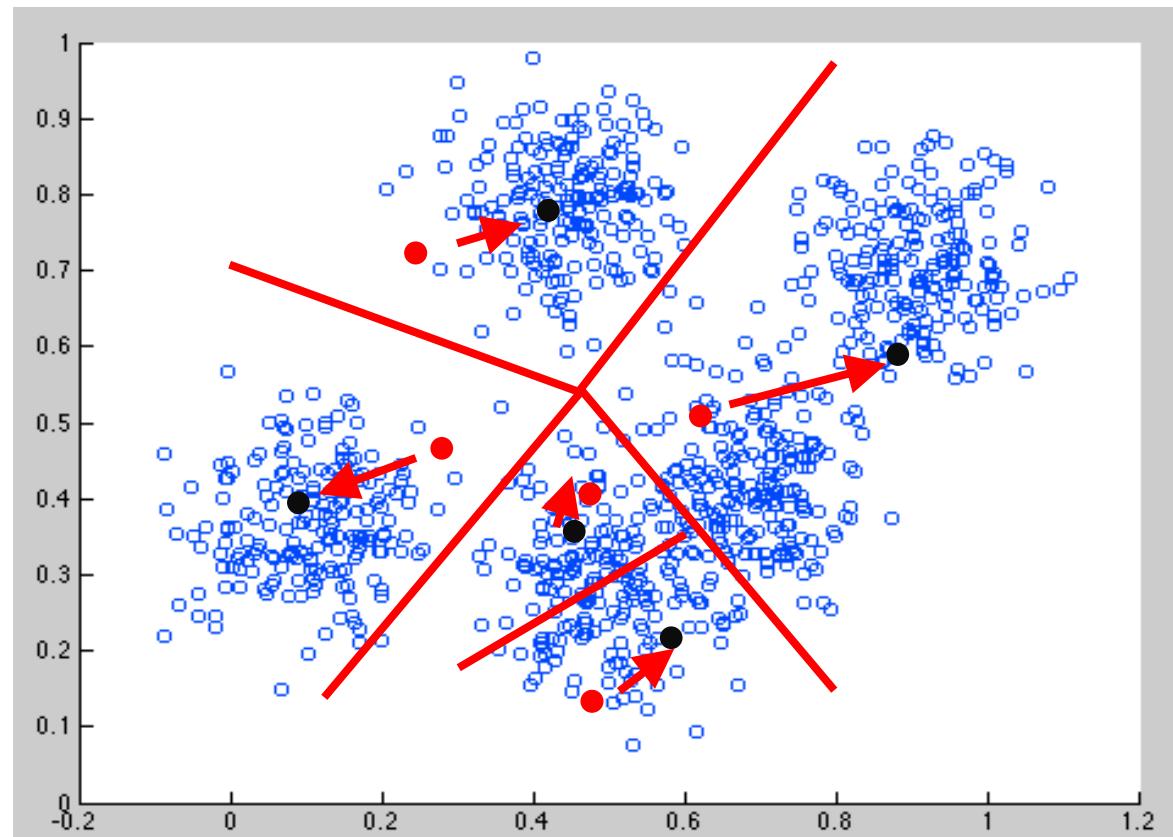
K-Means Clustering (4/7)

- Assign Centers
 - Assign all points to the closest center



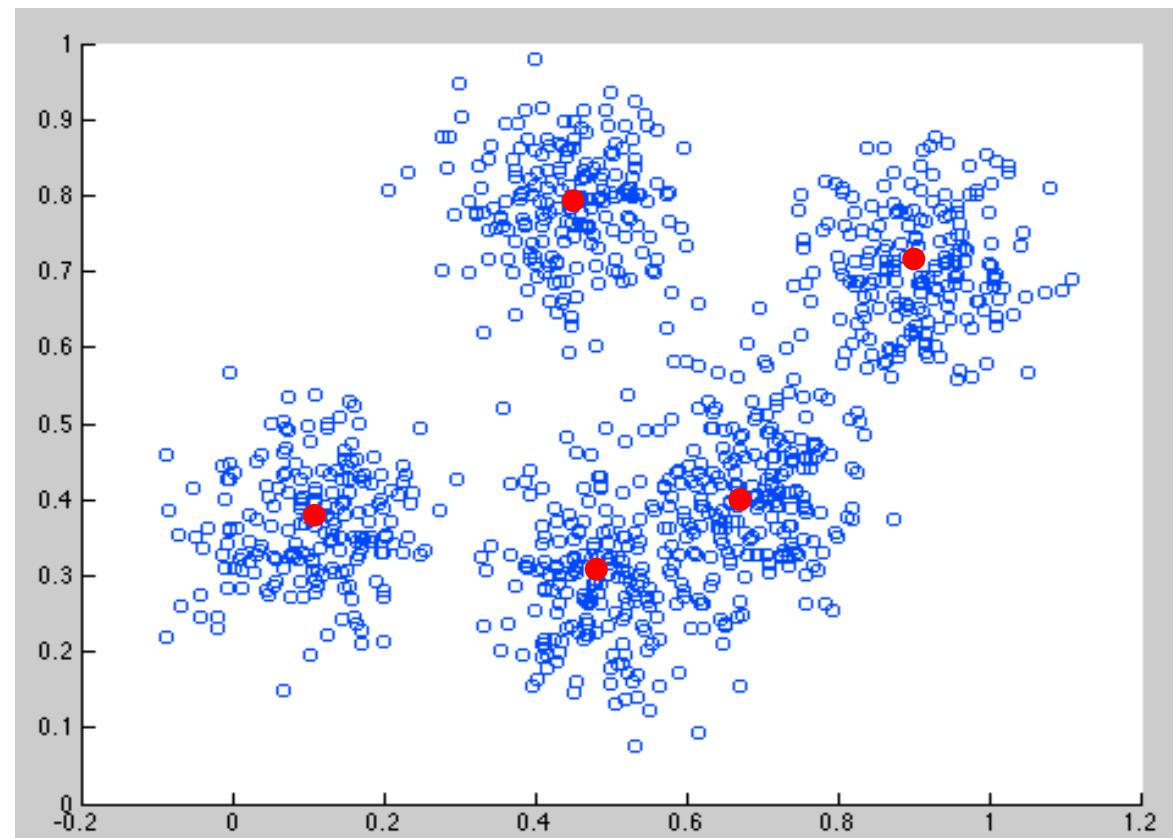
K-Means Clustering (5/7)

- Move center to middle
 - Change cluster centers to be in the middle of its points



K-Means Clustering (6/7)

- Initial Centers
 - Repeat until convergence



K-Means Clustering (7/7)

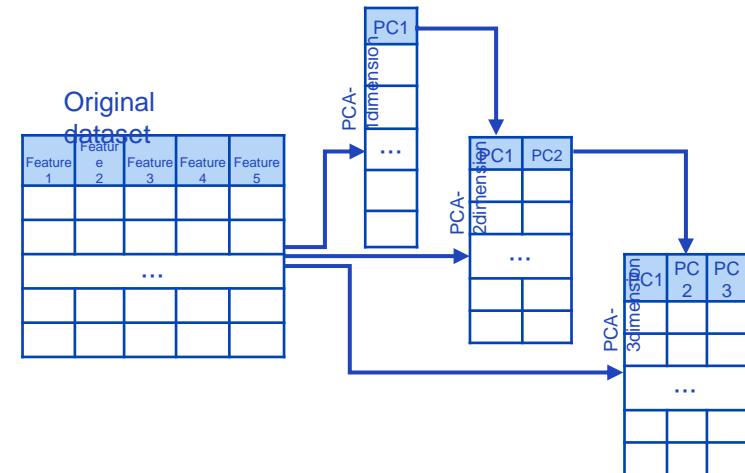
- Group items based on similar characteristics
 - Input the number of clusters (K) and randomly initialize the centers
 - Assign every point in the data to closest one of these centers
 - For each of the cluster centers, move the center to the middle of that cluster (mean)
 - Reassign each point to the new closest center
 - Repeat until convergence
- Pros:
 - Intuitive interpretation of the results
 - Quick and easy
- Cons:
 - Sensitive to the noise and outliers.
 - Cluster boundaries can only be linear

Principal Component Analysis (1/6)

- Purpose of the principal component analysis (PCA)
 - PCA is an algorithm that reduces the dimensions of high-dimensional data to low-dimensional data
 - Transform a set of correlated variables into another set of uncorrelated variables
 - Get a new set of orthogonal vectors (principal components or PCs)
 - Order the new variables from the largest to the smallest variance
- Pros:
 - Easy to compute
 - Speeds up other machine learning algorithms
 - Counteracts the issues of high-dimensional data
- Cons:

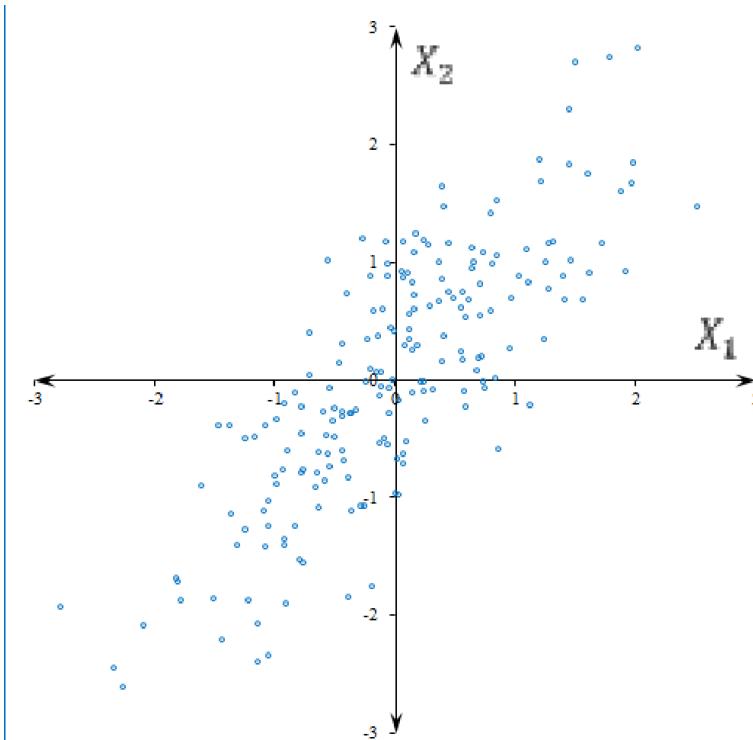
Principal Component Analysis (2/6)

- PCA is used across a variety of use cases
 - Visualize multidimensional data
 - Compress information
 - Simplify complex business decisions
 - Clarify convoluted scientific processes
- Conception
 - PCA creates a low-dimensional representation of a data set
 - It searches for planes closest to the data, then the points are projected onto it
 - The components are independent of each other and maximize the variance
 - Each successive component contains less variance than the previous component



Principal Component Analysis (3/6)

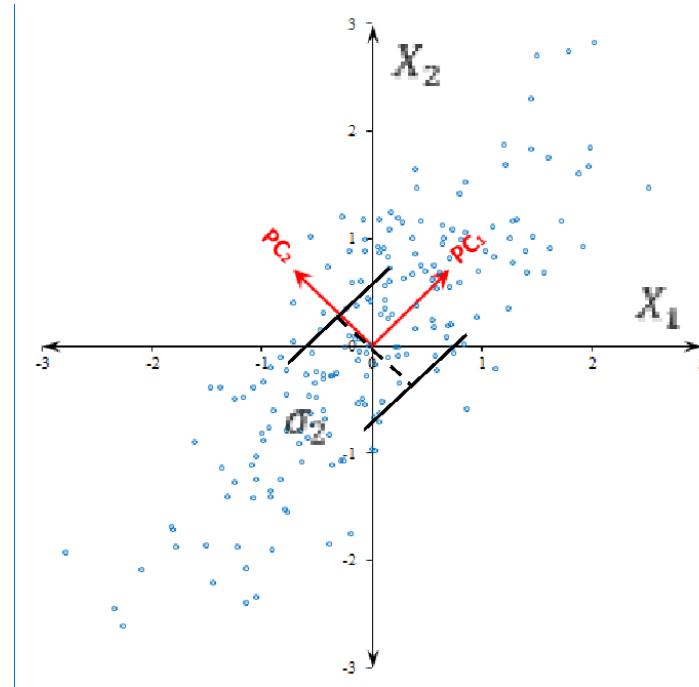
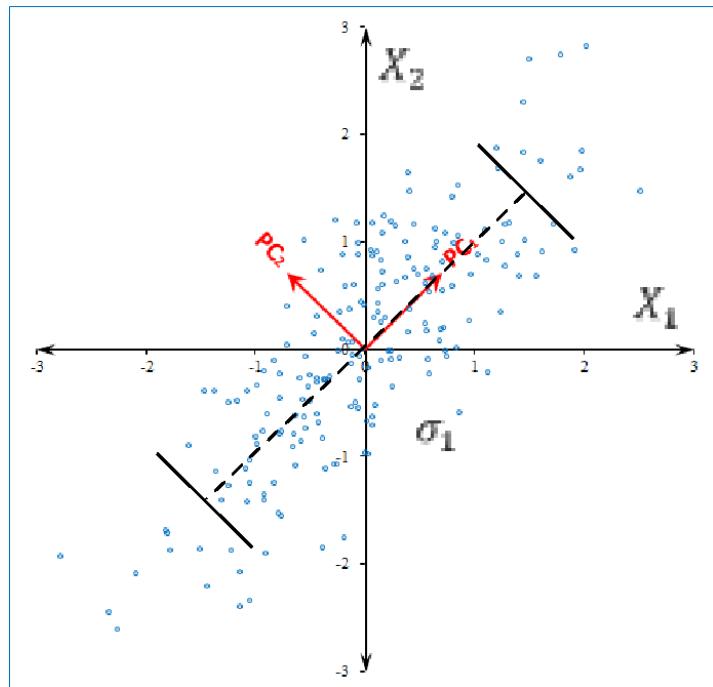
- Loading
 - Normalized principal component (PC)
 - There are as many loading vectors as the number of variables
- Variance σ^2 (or standard deviation σ)
 - The principal components have associated variances
- Transformed scores
 - Raw scores (original observations) represented using the PCs as new coordinate axes



A dataset with two variables (X_1 and X_2)

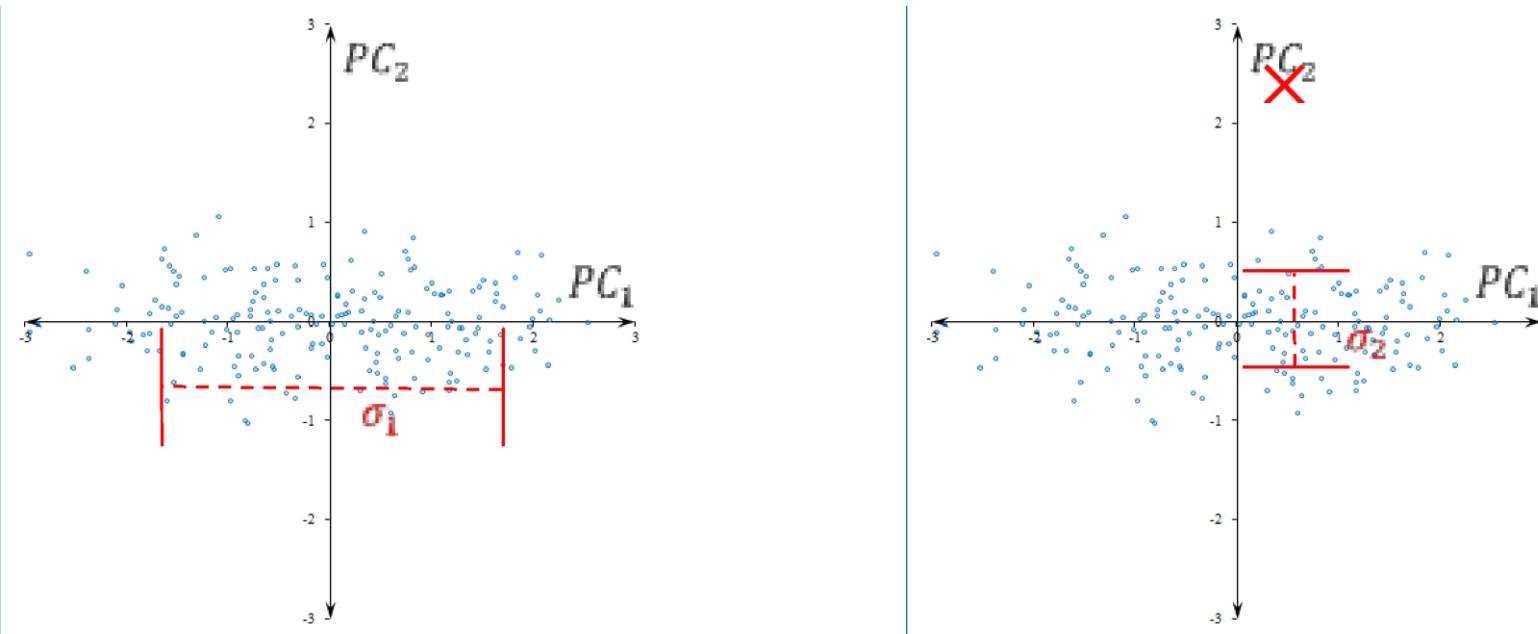
Principal Component Analysis (4/6)

- PC_1 is the direction of largest variance, while PC_2 is the direction of the next largest variance



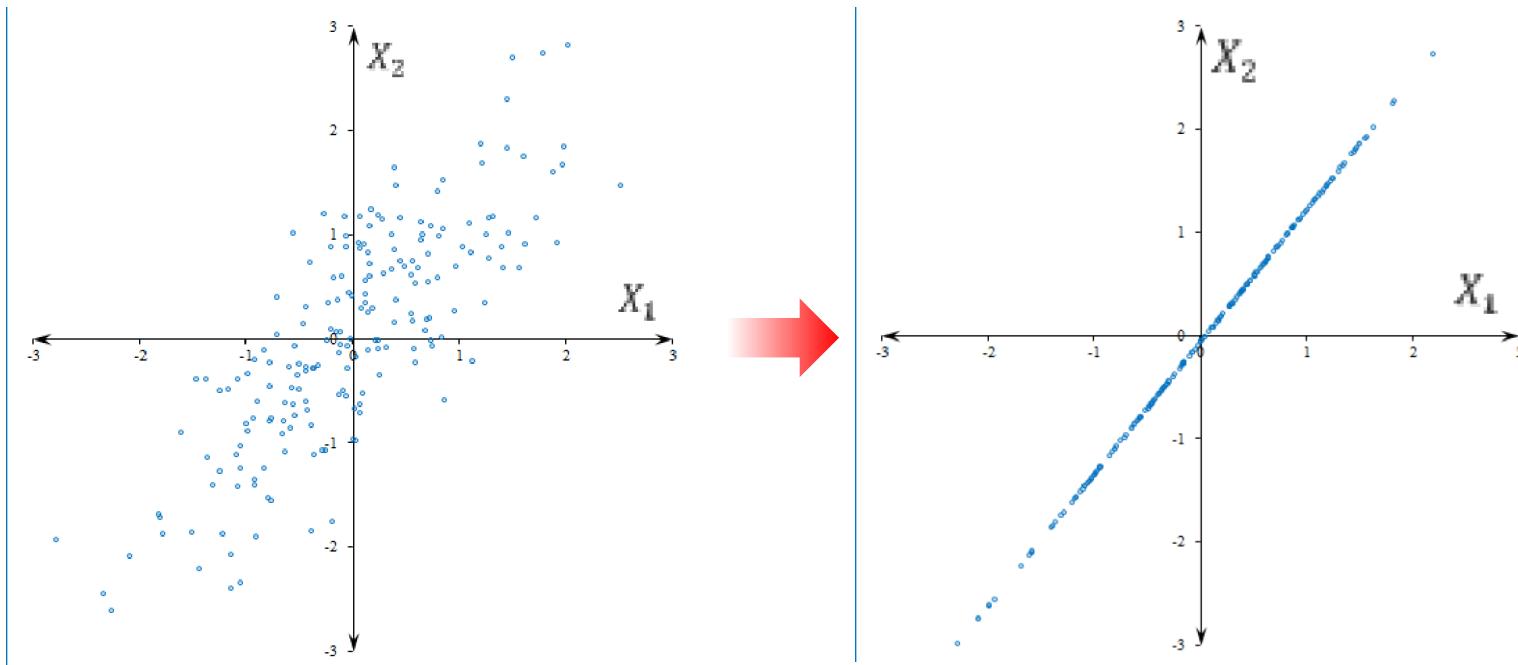
Principal Component Analysis (5/6)

- Transformed scores
 - The observations can be represented using the PC_1 and PC_2 as new coordinate axes



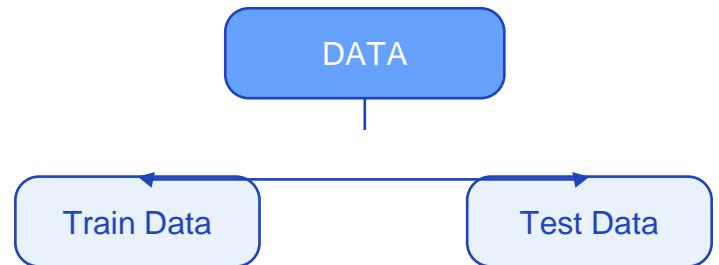
Principal Component Analysis (6/6)

- Dimensional reduction
 - Now, we can go back to the original coordinate system and show the “reduced dimensional input”



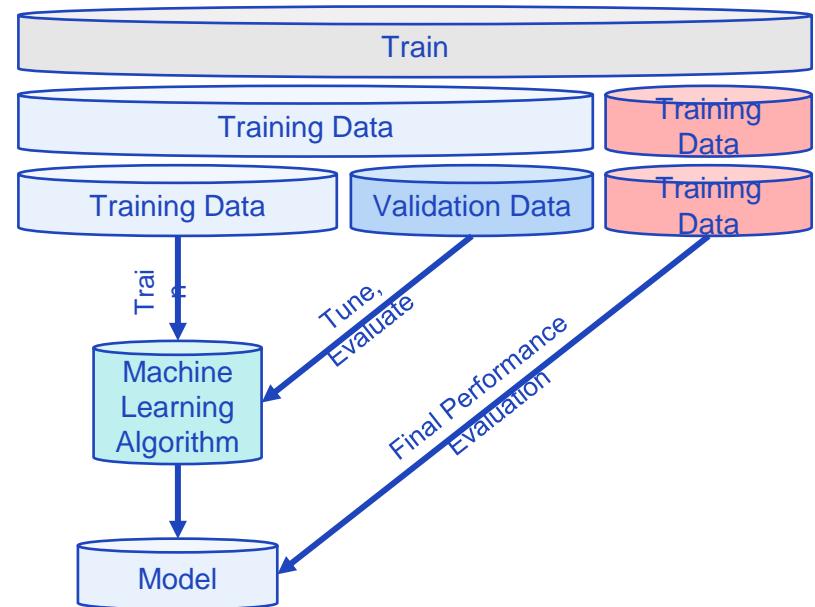
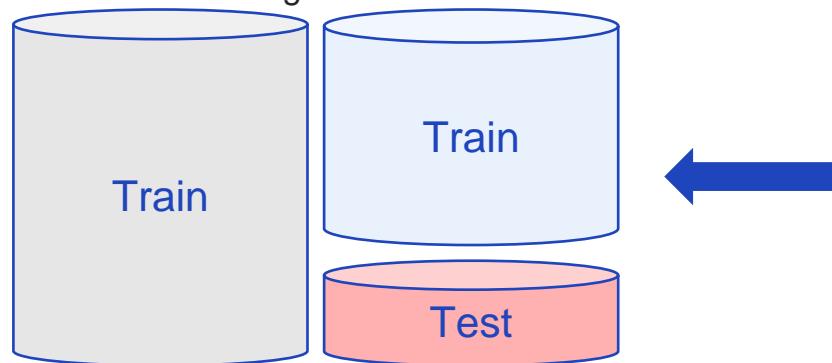
Generalization and Overfitting

- Generalization
 - A property that allows a model created by learning a training set, which is a subset, to be applied to other untrained subsets
- Data Set
 - The set should be large enough to yield statistically significant results
 - The test set must be selected to have the same characteristics as the training set
- Overfitting model
 - Overfitting by creating a model that is more complex than necessary
 - The loss during training is small, but it does not make good predictions on new data



Data Set

- Train set, Validation set and Test set
 - The data set is divided into a training set, a validation set, and a test set in a way to reduce the possibility of overfitting by dividing the data set into sets
- Run the following process
 - Train a model with the training set
 - Evaluate the model with the validation set
 - Adjust the model according to the results of the set validation



Cross-Validation

- Cross-Validation is the most popular way to evaluate models
- Cross-Validation Process
 - Divide the data set into N approximately equal sized “folds”
 - Train the algorithm on N-1 folds and use the last fold to compute the evaluation measure
 - Repeat N times, assigning each 1 of the N folds as the test fold
 - Report the mean and standard deviation of the evaluation measure over the N folds
 - Typically N is somewhere between 5 and 10

Test Data	Train Data									
-----------	------------	------------	------------	------------	------------	------------	------------	------------	------------	------------

Nested Cross-Validation (1/2)

- Nested Cross-Validation Process
 1. Divide the data set into N approximately equal sized “folds”
 2. Reserve 1 fold for test fold and N-1 for training folds
 3. Reserve 1 of the training folds for validation
 4. For $\lambda=\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \dots$ train the algorithm on the remaining N-2 training folds
 5. Repeat from step 3, N-1 times, rotating the validation fold within the N-1 training folds
 6. Choose λ that minimizes average training error over the training folds
 7. Use that λ to evaluate on the test set
 8. Repeat from step 2, N-1 times, by rotating the test fold
 9. Report the mean and standard deviation of the evaluation measure over the N folds

Nested Cross-Validation (2/2)

- For 10 folds and 5 different λ values tested
 - 10 (for each test fold) \times 9 (for each validation fold) \times 5 (for each λ value)
 - 450 test runs
- If there are multiple parameters to test out, 450 for each needed
- Nested cross-validation can become very expensive computationally



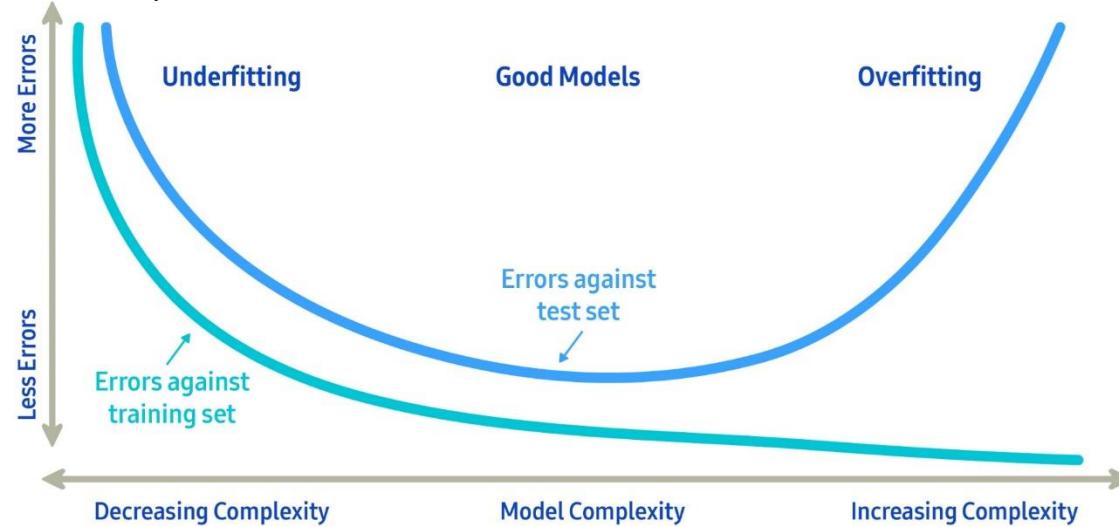
Overfitting & Underfitting

- Overfitting
 - Overfitting occurs when the model generated tries to fit the training data too closely
 - The model is only good for the training data and will likely produce very poor results with any new test data
 - Basically useless
- Underfitting
 - Underfitting occurs when the model generated has too much deviation from a good predictive model
 - While this model may not be completely useless, the margin of error is too large



Occam's Razor

- Among competing hypothesis, the one with the fewest assumptions should be selected
- William of Ockham (c. 1287–1347), was an English Franciscan friar, scholastic philosopher and theologian
- In our context, we can re-state this as:
 - The best machine learning models are simple models that fit the data well
- Selecting the Best Model



Unit 1

Machine Learning & Model

- | 1.1. Machine Learning Basic
- | 1.2. Machine Learning in Public Cloud
- | 1.3. Apache Spark ML Pipelines

AWS Machine Learning Stack

- Extensive machine learning capabilities

What is SageMaker?

- Amazon SageMaker is a fully managed machine learning service
- Easily build and train machine learning models, and then directly deploy them into a production-ready hosted environment
- All components used for ML provided in a single toolset
 - Labeling
 - Data preparation
 - Feature engineering
 - Statistical bias detection
 - Auto-ML
 - Training
 - Tuning
 - Hosting
 - Monitoring and workflow

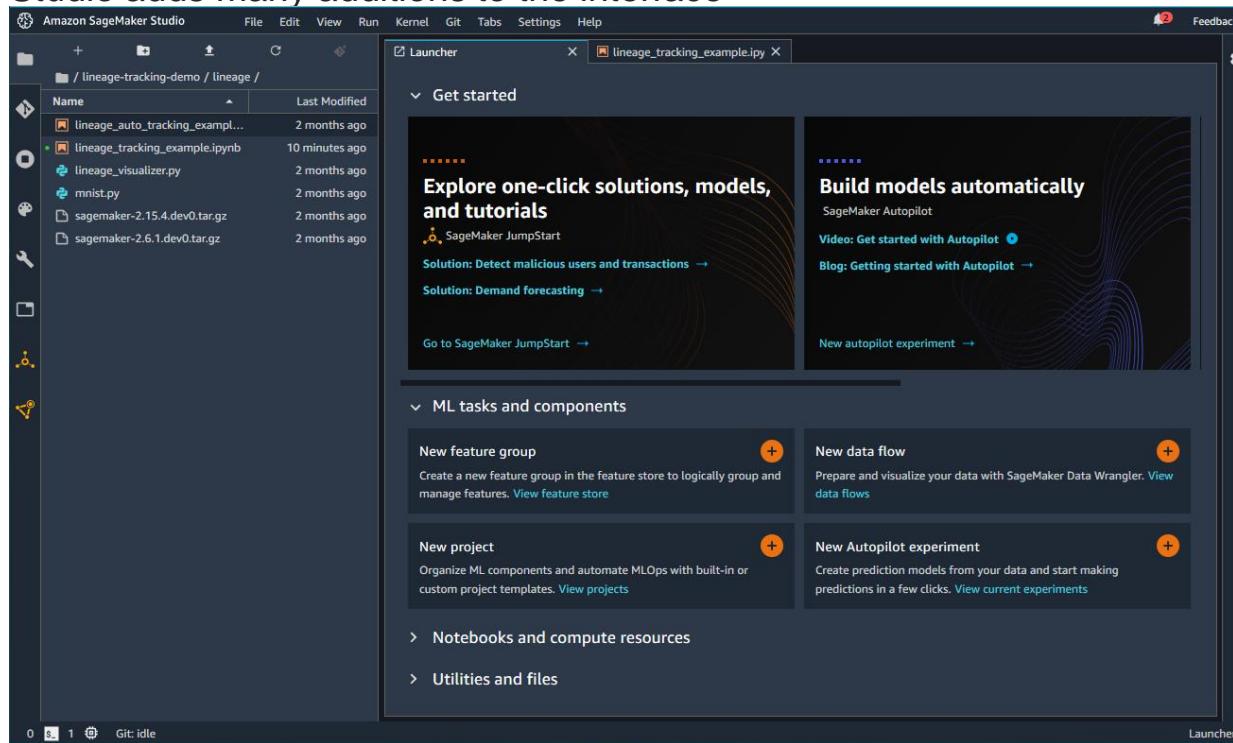


SageMaker Studio (1/3)

- Amazon SageMaker Studio is a web-based, integrated development environment (IDE) for machine learning that lets user build, train, debug, deploy, and monitor machine learning models
- SageMaker Studio provides all the tools needed to take models from experimentation to production while boosting productivity
- In a single unified visual interface:
 - Write and execute code in Jupyter notebooks
 - Build and train machine learning models
 - Deploy the models and monitor the performance of their predictions
 - Track and debug the machine learning experiments

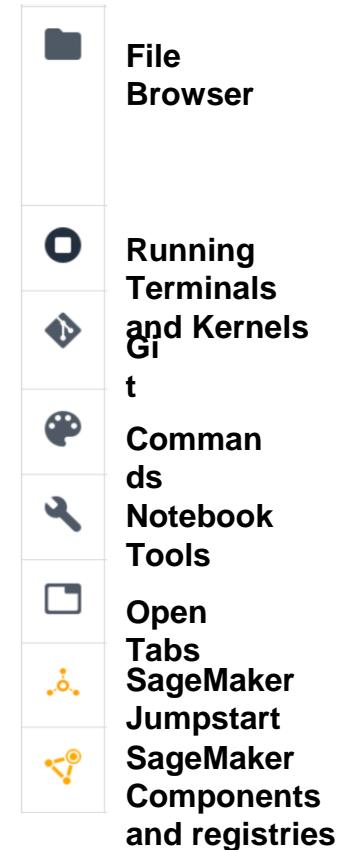
SageMaker Studio (2/3)

- Amazon SageMaker Studio extends the JupyterLab interface
- Studio adds many additions to the interface



SageMaker Studio (3/3)

- Left side bar
 - When you hover over an icon, a tooltip displays the icon name. When you choose an icon, the file and resource browser displays the described functionality
- File and resource browser
 - The file and resource browser displays lists of your notebooks, experiments, trials, trial components, and endpoints
- Main work area
 - The main work area consists of multiple tabs that contain your open notebooks and terminals, and detailed information about your experiments and endpoints
- Settings
 - The settings pane allows to adjust table and chart properties

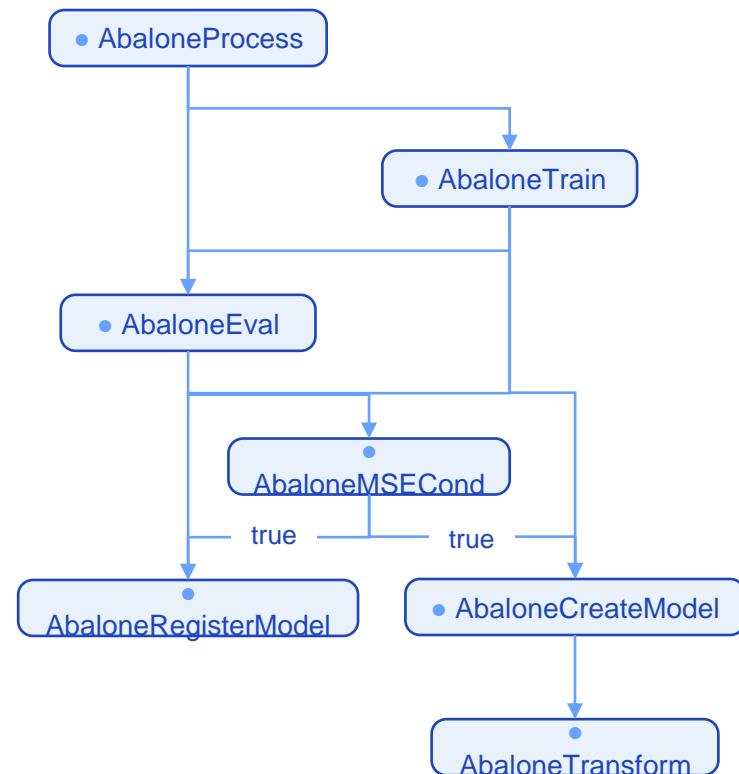


Automate MLOps with SageMaker

- Create end-to-end ML solutions with CI/CD by using SageMaker projects
- Use SageMaker projects to create an ML ops solution to orchestrate and manage:
 - Data preparation and feature engineering
 - Training models
 - Evaluating models
 - Deploying models
 - Monitor and update models

SageMaker Pipeline

- A series of interconnected steps that is defined by a JSON pipeline definition
- This pipeline definition encodes a pipeline using a directed acyclic graph (DAG)
- DAG gives information on the requirements for and relationships between each step of pipeline
- The structure of a pipeline's DAG is determined by the data dependencies between steps

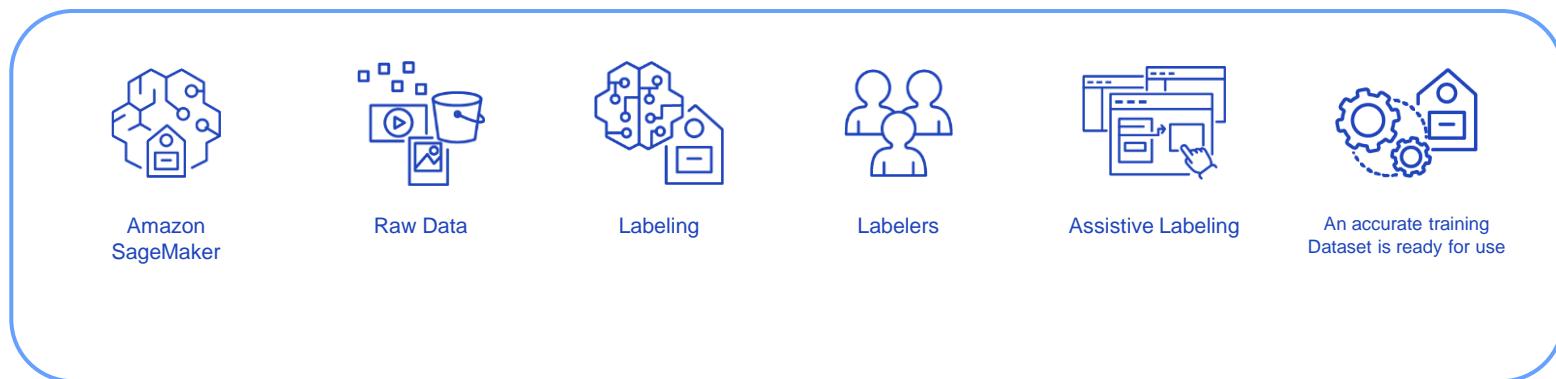


Data Wrangler

- A feature of SageMaker Studio that provides an end-to-end solution to import, prepare, transform, featurize, and analyze data
 - Import – Connect to and import data from Amazon Simple Storage Service (Amazon S3), Amazon Athena (Athena), and Amazon Redshift
 - Data Flow – Create a data flow to define a series of ML data prep steps
 - Transform – Clean and transform dataset using standard transforms like string, vector, and numeric data formatting tools
 - Analyze – Analyze features in dataset at any point in the flow
 - Export – Data Wrangler offers export options to other SageMaker services, including Data Wrangler jobs, Feature Store, and pipelines, making it easy to integrate the data prep flow into the ML workflow

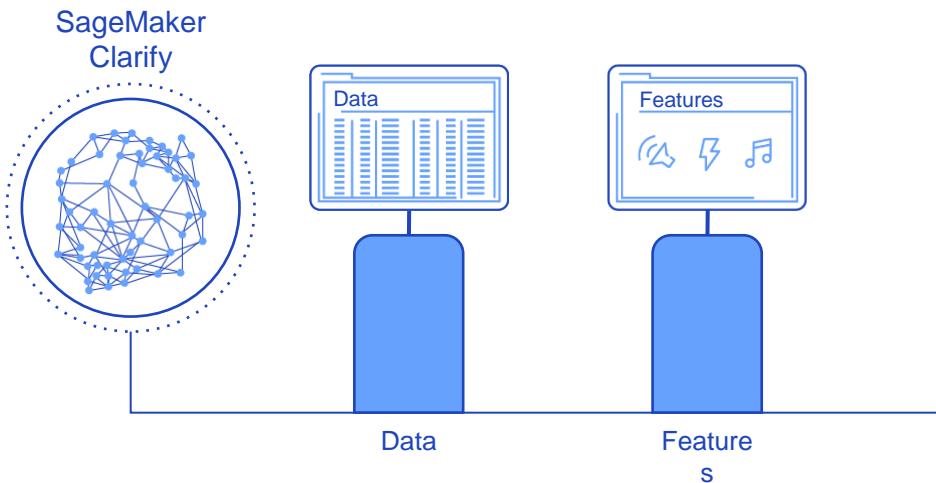
Ground Truth to Label Data

- Amazon SageMaker Ground Truth is a fully managed data labeling service that makes it easy to build highly accurate training datasets for machine learning
- Leverage custom or built-in data labeling workflows through the SageMaker Ground Truth console to label data
- Helps building high-quality training datasets for machine learning models
- How it works



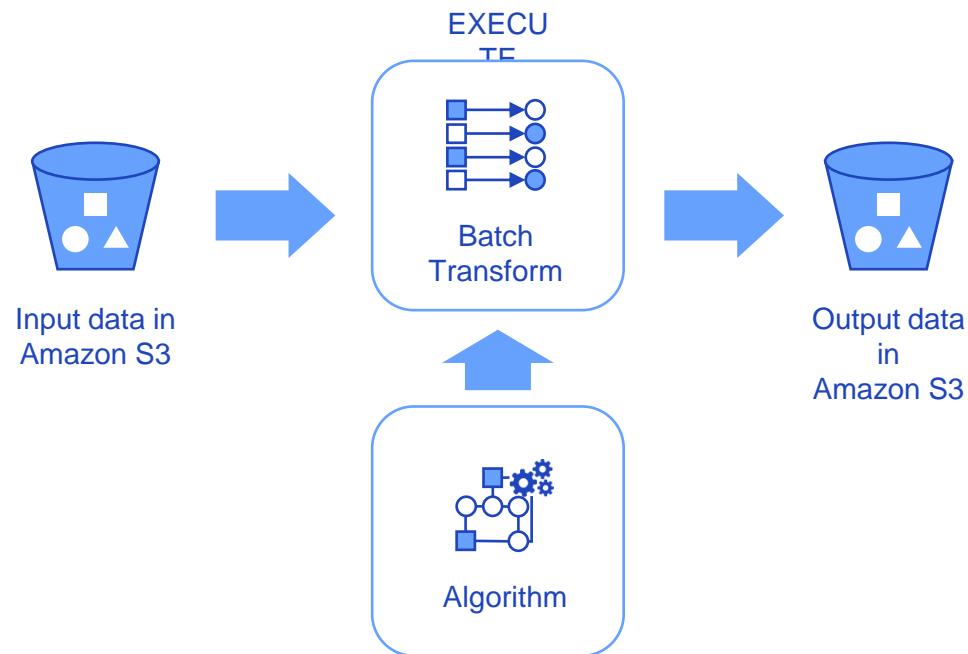
ML Lineage Tracking & Clarify

- ML lineage tracking
 - Track the lineage of machine learning workflows
- Clarify
 - Improve machine learning models by detecting potential bias and help explain the predictions that models make



SageMaker Batch Transform

- Preprocess datasets, run inference when we don't need a persistent endpoint, and associate input records with inferences to assist the interpretation of results



Amazon SageMaker Built-in Algorithms

- Built-in Algorithms

Classification

- Linear Learner
- XGBoost
- KNN

Computer Vision

- Image Classification<>
- Object Detection<>
- Semantic Segmentation

Topic Modeling

- LDA
- NTM

Forecasting

- DeepAR

Working with Text

- BlazingText
- Supervised
- Unsupervised

Recommendation

- Factorization Machines

Clustering

- Kmeans

Sequence Translation

- Seq2Seq

Anomaly Detection

- Random Cut Forests
- IP Insights

Feature Reduction

- PCA
- Object2Vec

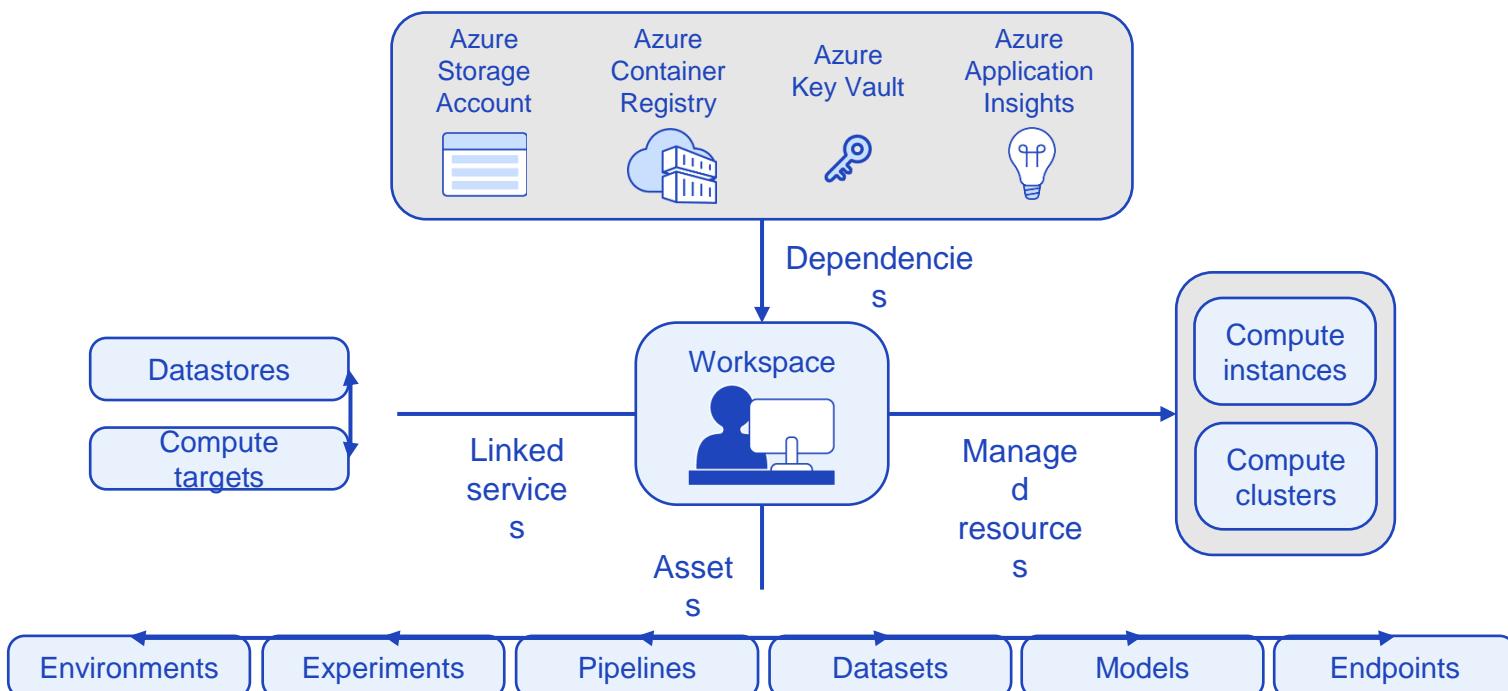
Regression

- Linear
- Learner

- XGBoost
- KNN

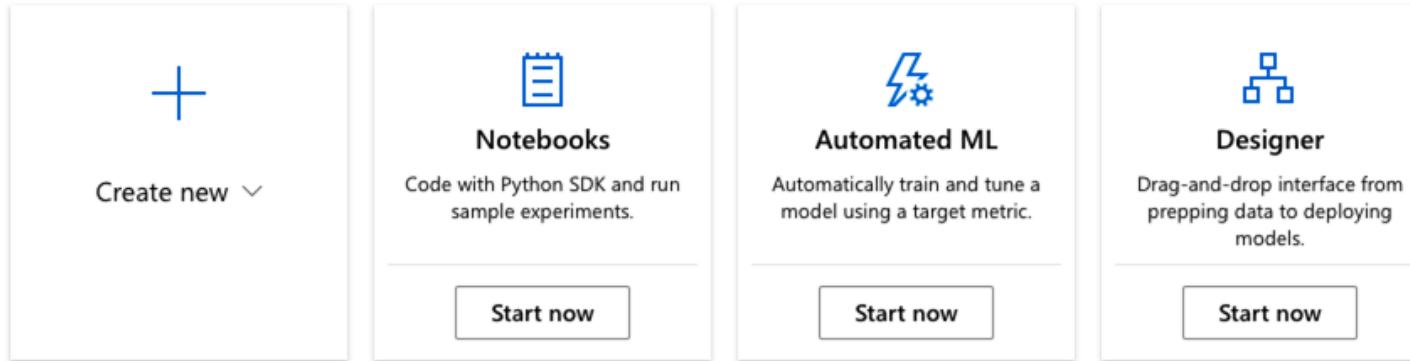
Azure Machine Learning

- Azure Machine Learning is a cloud-based service that helps data scientists save time by simplifying tasks such as preparing data, training models, and deploying prediction services



Azure Machine Learning Studio

- Studio is a web portal for Azure Machine Learning



- Provides a codeless environment and a code environment for data science platforms
- Notebooks – Create Jupyter notebooks and run scripts
- Automated ML – Step through ML without generating code
- Designer – provide functionality by drag-in-drop using pre-built modules

How to use Azure Machine Learning

- Create workspace resource

Machine learning

Create a machine learning workspace

Basics Networking Advanced Tags Review + create

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ 종량제1

Resource group * ⓘ SIC

Workspace name * ⓘ azure-ml-Lecture

Region * ⓘ East US

Storage account * ⓘ (new) azurermlecture0964733062

Key vault * ⓘ (new) azurermlecture2244523976

Application insights * ⓘ (new) azurermlecture2195067607

Container registry * ⓘ None

Review + create < Previous Next : Networking

Microsoft.MachineLearningServices | Overview

Deployment

Search (Cmd+/)

Delete Cancel Redeploy Refresh

We'd love your feedback!

Your deployment is complete

Deployment name: Microsoft.MachineLearningServices
Subscription: 종량제1
Resource group: SIC

Home > Microsoft.MachineLearningServices >

azure-ml-Lecture

Machine learning

Search (Cmd+/)

Download config.json Delete

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Essentials

Resource group : SIC
Location : East US
Subscription : 종량제1
Subscription ID : 4e4d7de4-4f0d-4a4b-b389-b569fe38696f

How to use Azure Machine Learning

- Create compute instance
 - Used for code creation and execution of Python scripts and Jupyter notebooks
- Create compute clusters
 - Deploy training and batch inference experiments using GPU/CPU

Create compute cluster (CPU/GPU)

Virtual Machine

Advanced Settings

Configure Settings
Configure compute cluster settings for your selected virtual machine size.

Name	Category	Cores	Available quota	RAM	Storage	Cost/Node
Standard_DS2_v2	General purpose	2	10 cores	7 GB	14 GB	\$0.16/hr

Compute name * (1)

Minimum number of nodes * (1)

Maximum number of nodes * (1)

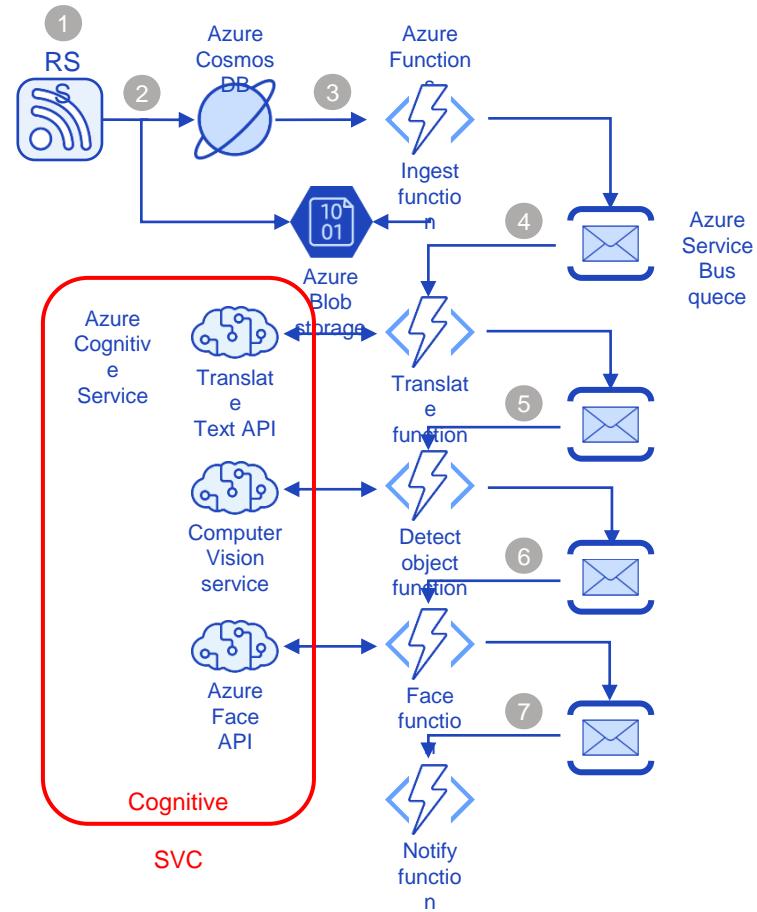
Idle seconds before scale down * (1)

Enable SSH access (1)

[Advanced settings](#)

Cognitive Services

- Azure Cognitive Services is a cloud-based service with REST APIs and client library SDKs to help you build cognitive intelligence into your applications
- Service category
 - Visual API
 - Voice API
 - Language API
 - Decision API
 - Search API



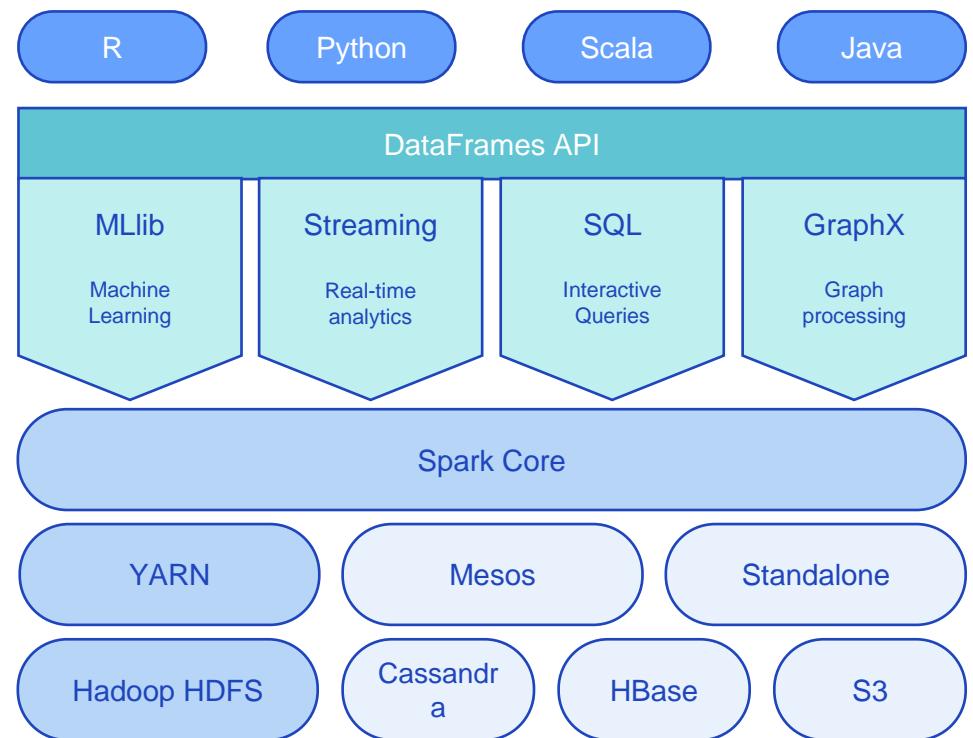
Unit 1

Machine Learning & Model

- | 1.1. Machine Learning Basic
- | 1.2. Machine Learning in Public Cloud
- | 1.3. Apache Spark ML Pipelines

What is Apache Spark?

- Apache Spark is a unified analytics engine for large-scale data processing
 - A set of high-level APIs in Scala, Python, Java and R
- General-purpose data processing engine that provides
 - Interactive processing
 - Extremely fast batch processing
 - Real-time stream processing
 - Distributed machine learning modeling and inference engine
 - Graph processing
- In-memory distributed computation engine



What is Spark MLlib?

- Spark MLlib Is a library of ML algorithms and utilities designed to run in parallel on Spark cluster
 - Introduces a few new data types including vector(dense and sparse), labeled point, rating, etc.
 - Allows to invoke various algorithms on distributed datasets(RDD/Dataset)
- Spark ML can help to build nice ML pipelines
- Existing libraries and frameworks reduce a lot of tedious work
- MLlib is Spark's library of machine learning (ML) functions designed to run in parallel on clusters.
 - MLlib contains a variety of learning algorithms
- MLlib invokes various algorithms on RDDs
- Some classic ML algorithms are not included with Spark MLlib because they were not designed for parallel

Spark MLlib Overview

- MLlib is Spark's machine learning library
 - Goal is to make practical machine learning scalable and easy
- Consists of common learning algorithms and utilities
 - Classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as lower-level optimization primitives and higher-level pipeline APIs
- Divided into two packages
 - `Spark.mllib`
 - `Spark.ml`

Spark DataFrames

- Conceptually equivalent to a table in a relational database
 - Similar to an R or Python dataframe
- Rich optimizations under the hood
- DataFrames can be constructed from a wide array of sources
 - Structured data files
 - Tables in Hive
 - External databases
 - Existing RDDs

Data Types - MLlib

- Supports local vectors and matrices stored on a single machine
- Distributed matrices backed by one or more RDDs
- Local vectors and local matrices are simple data models that serve as public interface
- A training example used in supervised learning is called a labeled point

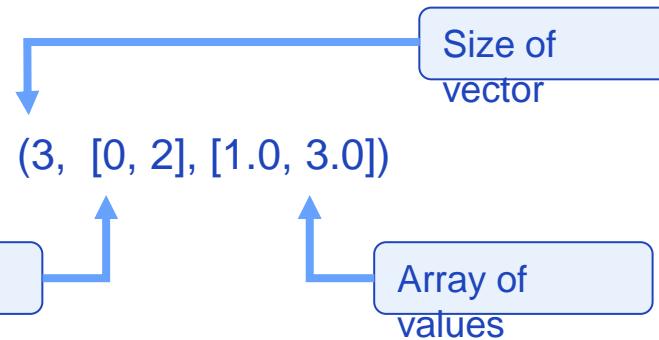
Data Type – Local Vector

- Local vectors are collections of integer and double datatype values, store on a single machine.
 - Their collection indexes start with 0
 - Two kinds are available - dense and sparse
- Dense vectors store the values in an array of type double: Array[Double]
- Sparse vectors store the data over parallel arrays. One array stores the value, while the other stores the index

Dense vector
index Array of double-typed values

[1.0, 0.0, 3.0]

Sparse vector
two parallel arrays
one for indices
one for values



Labeled Point

- Labeled Points stores the labels or ground truth used in supervised algorithms
- A dense or sparse local vector is used to store the labels
- Binary Classification : Label values are 0 (negative response) or 1 (positive response)
- Multiclass Classification: Labels are 0, 1, 2, etc for each classification
- Values are stored as double instead of integers
 - Allows same Labeled Point datatype to be used in regression and classification algorithms

Data Type – Dense Local Matrix

- A local matrix contains 3 values to describe the matrix
 - Integer types are used for the row and column indices
 - The actual values are stored in an Array of type double: Array[Double]
 - There are Dense and Sparse local matrices
- Dense matrix
 - Values are stored in column-major order, in the Array[Double]

$$\begin{pmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \\ 5.0 & 6.0 \end{pmatrix}$$

```
# Create a dense matrix ((1.0, 2.0), (3.0, 4.0), (5.0, 6.0))
# Format is (<num rows>, <num cols>, <Array[Double]> of values in column major
order

# Using factory class and method implementations of Matrices

dense_m = Matrices.dense(3, 2, [1.0, 3.0, 5.0, 2.0, 4.0, 6.0])
```

Data Type – Sparse Local Matrix

- Sparse matrix
 - Stored as Compressed Sparse Column in column-major order
- Compressed Sparse Column Format
 - NNZ = Number of Non-Zero elements = 3
 - values = Array of all non-zero values in column order
 - rowIndices = Array of row index number of matching value

$$\begin{bmatrix} 9.0 & 6.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 8.0 \end{bmatrix}$$

□ [9.0, 6.0, 8.0]
□ [0, 0, 0,
3]

2,

```
# create a sparse matrix using factory method
# Matrices.sparse numRows, numCols, colPtrs, rowIndices, values)

sparse_mat = Matrices.sparse(3, 3, [0, 1, 2, 3], [0, 0, 2], [9, 6, 8])
```

Data Type – Distributed Matrix

- Unlike the Local Matrix datatype, values for the Distributed Matrix datatype are distributed
 - Stored in one or more RDDs - remember RDDs are distributed themselves
 - The row and column indices are stored as type long
- Caution must be taken when choosing this datatype
 - Changing a Distributed Matrix to another datatype will involve all the Executors to shuffle their data
 - Shuffling is a very expensive and involves I/O over the network
- Four types of distributed matrices implemented:
 - RowMatrix
 - IndexedRowMatrix
 - CoordinateMatrix
 - BlockMatrix

Data Type – Row Matrix

- A RowMatrix is a distributed matrix, where each row is stored as a local vector
- Row indices are not important and therefore the local vectors can be organized into RDDs

```
from pyspark.mllib.linalg.distributed import RowMatrix
# Create an RDD of vectors.
rows = sc.parallelize([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
# Create a RowMatrix from an RDD of vectors.
mat = RowMatrix(rows)
# Get its size.
m = mat.numRows() # 4
n = mat.numCols() # 3
# Get the rows as an RDD of vectors again.
rowsRDD = mat.rows
```

Data Type – IndexedRowMatrix

- An IndexedRowMatrix is similar to a RowMatrix but now, the row indices are important
- Stored as an RDD of indexed rows

- Indexed Row objects store row index value

```
from pyspark.mllib.linalg.distributed import IndexedRow, IndexedRowMatrix

# Create an RDD of indexed rows using IndexedRow class:
indexedRows = sc.parallelize([IndexedRow(0, [1, 2, 3]),
                             IndexedRow(1, [4, 5, 6]),
                             IndexedRow(2, [7, 8, 9]),
                             IndexedRow(3, [10, 11, 12]))]

# Or by using a tuple of (<index long>, <value vector>)
indexedRows = sc.parallelize([(0, [1, 2, 3]),
                             (1, [4, 5, 6]),
                             (2, [7, 8, 9]),
                             (3, [10, 11, 12]))]

# Create an IndexedRowMatrix from an RDD of IndexedRows.
mat = IndexedRowMatrix(indexedRows)
```

Data Type – CoordinateMatrix

- A CoordinateMatrix is used when both row and column indices are important
- Stored as an RDD of tuples or MatrixEntry class
 - Each tuple □ (<row index> of type long, <column index of type long>, <value of type Double>)
 - Alternatively, instead of a tuple, the MatrixEntry class may be used

```
from pyspark.mllib.linalg.distributed import CoordinateMatrix, MatrixEntry

# Create an RDD of coordinate entries using MatrixEntry class
entries = sc.parallelize([MatrixEntry(0, 0, 1.2),
                         MatrixEntry(1, 0, 2.1),
                         MatrixEntry(6, 1, 3.7)])

# Or using (long, long, float) tuples:
entries = sc.parallelize([(0, 0, 1.2), (1, 0, 2.1), (2, 1, 3.7)])

# Create an CoordinateMatrix from an RDD of MatrixEntries.
mat = CoordinateMatrix(entries)
```

Data Type – BlockMatrix

- A BlockMatrix is a distributed matrix stored as RDD of MatrixBlocks
- A MatrixBlock can be thought of as a sub-matrix of the full distributed matrix
 - Represented by a tuple of (<sub-matrix row index as type Int>, <sub-matrix col index as type Int>, Matrix)
 - The Matrix is the actual sub-matrix
 - The size of the sub-matrix can be configured as rowsPerBlock x colsPerBlock
 - Default size is 1024 x 1024
- Easiest to create a BlockMatrix by converting IndexedRowMatrix or CoordinateMatrix
 - Use the toBlockMatrix method
- BlockMatrix supports methods such as add and multiply with other BlockMatrix
- There is also validate() function that verifies the proper creation of a BlockMatrix if the toBlockMatrix is not used and created manually through code

Data Type – BlockMatrix

- A BlockMatrix is a distributed matrix backed by an RDD of MatrixBlocks

```
from pyspark.mllib.linalg import Matrices
from pyspark.mllib.linalg.distributed import BlockMatrix
# Create an RDD of sub-matrix blocks.
blocks = sc.parallelize([(0, 0), Matrices.dense(3, 3, [1, 2, 3, 4, 5,
6,7,8,9])), ((1, 1), Matrices.dense(3, 3, [10, 11, 12, 13, 14, 15, 16, 17,
18,])),((2,0), Matrices.dense(3,3, [1, 1, 1, 1, 1, 1, 1, 1]))])
# Create a BlockMatrix from an RDD of sub-matrix blocks.
mat = BlockMatrix(blocks, 3, 3)
# Get its size.
m = mat.numRows() # 3
n = mat.numCols() # 3
# Get the blocks as an RDD of sub-matrix blocks.
blocksRDD = mat.blocks
# Convert to a LocalMatrix.
localMat = mat.toLocalMatrix()
# Convert to an IndexedRowMatrix.
indexedRowMat = mat.toIndexedRowMatrix()
# Convert to a CoordinateMatrix.
coordinateMat = mat.toCoordinateMatrix()
```

[Lab1]

Working with Azure Machine Learning



[Lab2] Working with SageMaker in AWS



Unit 2.

Spark Machine Learning Library

Big Data Modeling and AI

Unit 2

Spark Machine Learning Library

- | 2.1. Creating Spark MLlib Pipelines
- | 2.2. Classification and Regression
- | 2.3. Clustering & Collaborative Filtering

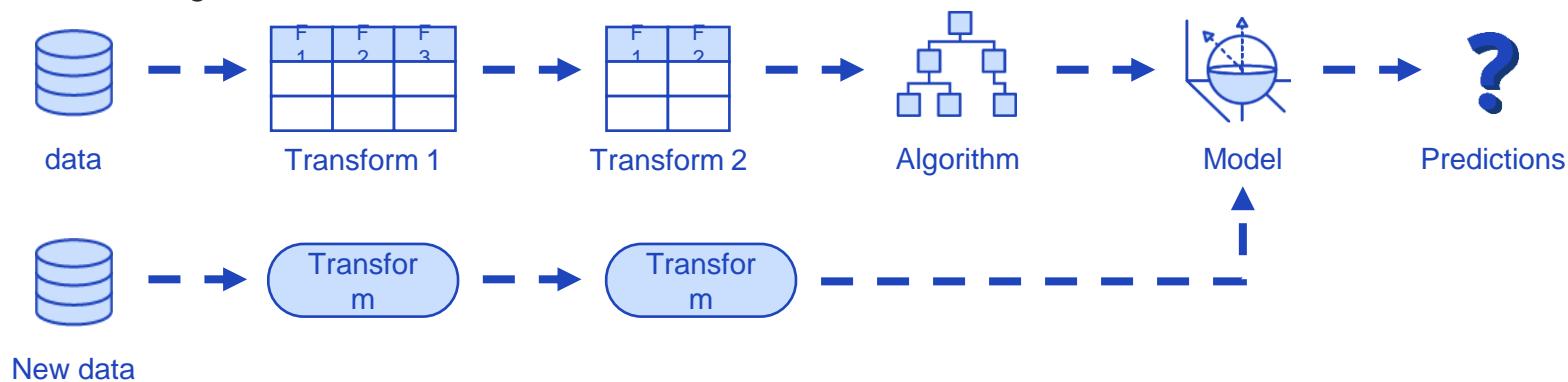
Spark MLlib Pipelines (1/3)

- A Machine Learning(ML) pipeline is a complete workflow combining machine learning algorithms.
 - It is similar to data pipeline of Big data
- Create an Apache Spark machine learning pipeline
 - Spark MLlib and ML support a variety of machine learning algorithms
 - There can be many steps required to process and learn from data, requiring a sequence of algorithms



Spark MLlib Pipeline (2/3)

- Machine learning task consist of a complex series of steps
 - Data transformations, algorithm training, and model prediction
 - We can think of these steps as a pipeline through which data travels
- Specific pipeline steps can be divided into two types
 - Transformer: Convert one dataset to another set
 - Estimator: An algorithm that can be fitted to a dataset to create a model

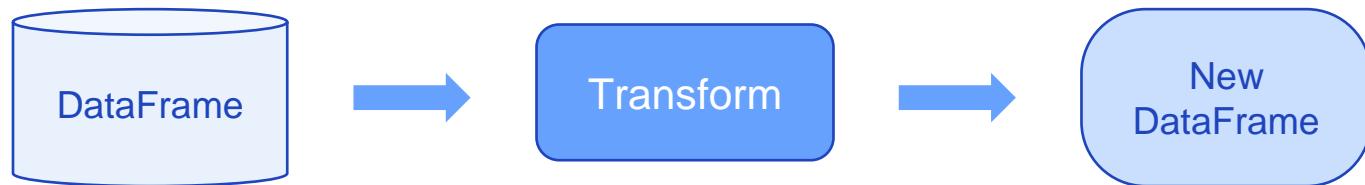


Spark MLlib Pipeline (3/3)

- Main concepts in Pipelines
 - DataFrame: SparkML dataset type
 - Transformer: Transforms one DataFrame to another DataFrame
 - Estimator: Executes an algorithm on a dataset and generates a model
 - Pipeline: Chains multiple steps to define an ML workflow
 - Parameters: Estimators and Transformers use a unified API to specify parameters
- Why use DataFrame for machine learning?
 - Efficient operation is possible due to the columnar structure
 - Single input type of various algorithms
 - DataFrames are a convenient data type for making the steps in a machine learning operations

Transformers (1/2)

- The transformers takes a DataFrame as input and returns a new DataFrame
 - Generalizes the concept of transformations to a broader definition
 - By this definition, the trained machine learning model is a transformer
 - Read a column (e.g., text), map it into a new column (e.g., feature vectors), and output a new DataFrame with the mapped column appended

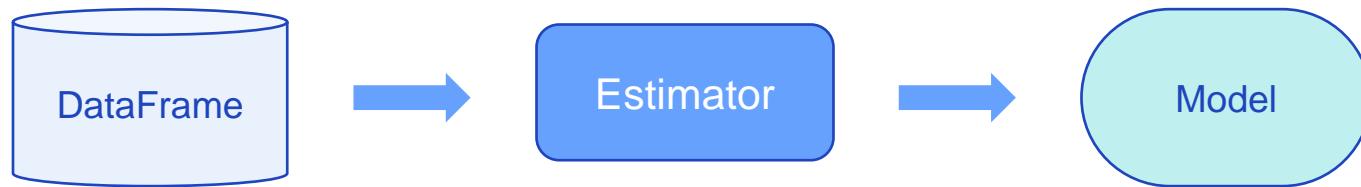


Transformers (2/2)

- All transformers implement transformation methods and **fit()** methods if the transformation depends on the data
- A Transformer transforms an input dataframe into another dataframe
 - The **inputCol()** parameter specifies the name of the column in the DataFrame to convert
 - The **outputCol()** parameter specifies the column that stores the converted data in the output DataFrame.
- If there are two types of transformer, transform is the main function
 - Feature Transformer
 - Learning Model
- Commonly used transformers for feature transformer
 - **StringIndexer()**
 - **PolynomialExpansion()**
 - **VectorAssembler()**

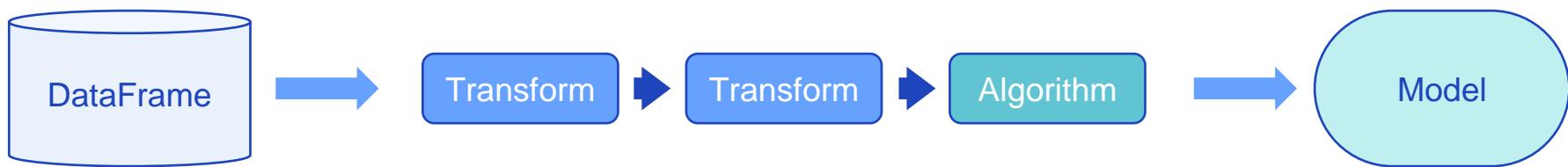
Estimators

- The estimator represents the algorithm performed to determine the model parameters
 - Take a DataFrame as input and create a model (Transformer)
 - The estimator is initialized with a specific set of parameters used in the execution of the algorithm.
 - All estimators implement the **fit()** method, which is called to run the algorithm on the provided data.
 - The result is an instance of a specific algorithm model



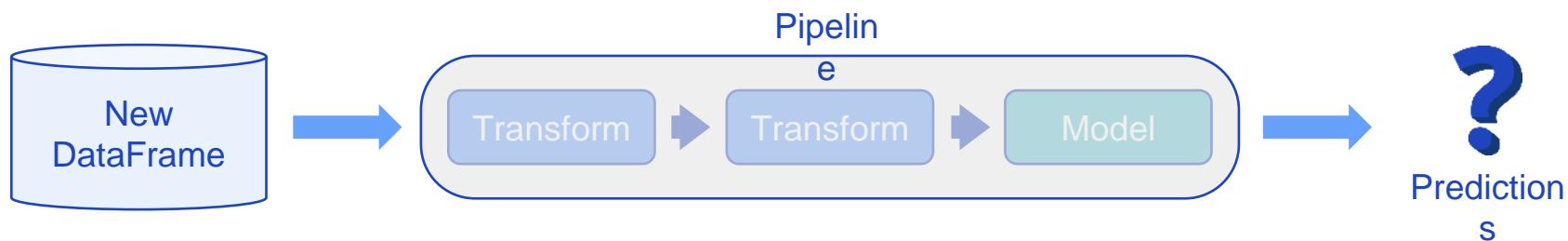
Pipelines (1/3)

- The pipeline represents a series of steps in a machine learning workflow
 - The pipeline itself is an estimator
- Each pipeline stage is either a transformer or an estimator
- Takes a DataFrame as input and creates a PipelineModel as output



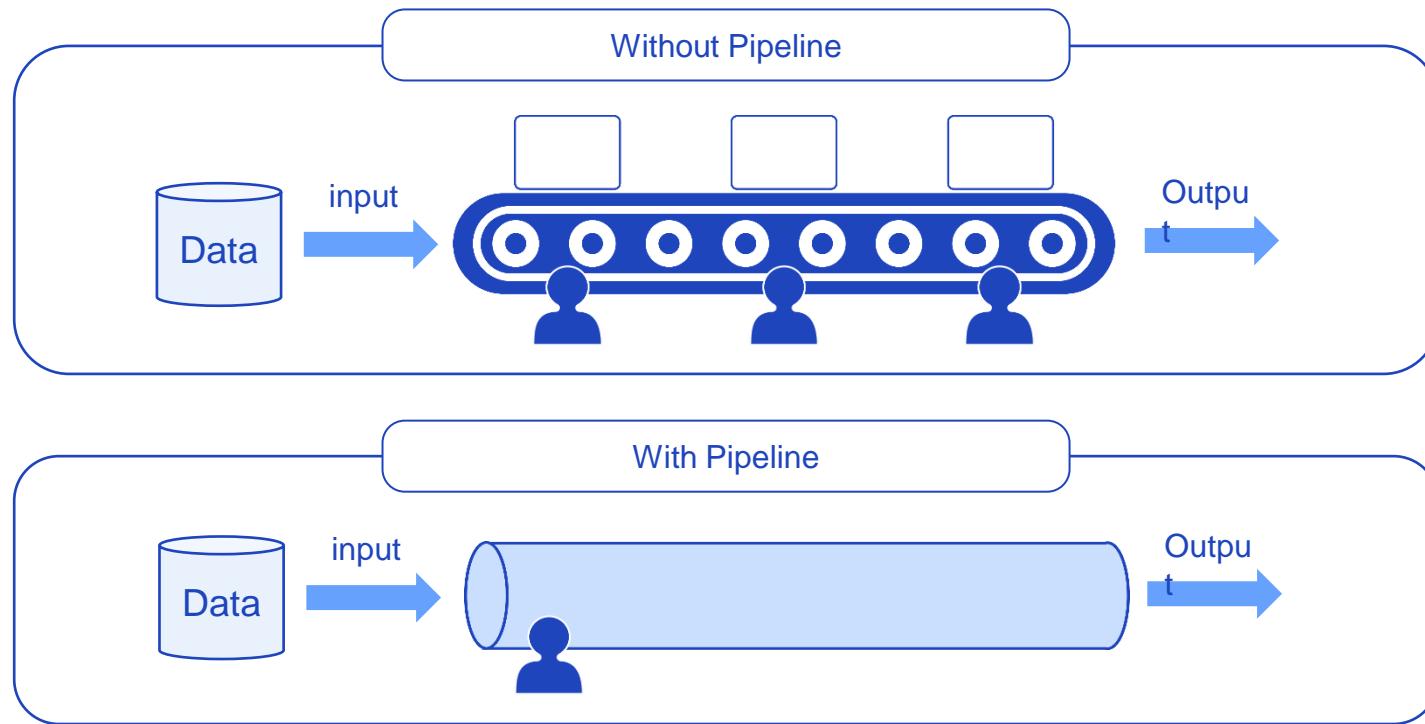
Pipelines (2/3)

- A PipelineModel contains the model outputs from the transformations and algorithms connected to the pipeline
 - PipelineModel implements transform methods and acts as a Transformer
 - Apply transformations in the pipeline and make predictions using the model generated by the algorithm
 - Takes a DataFrame as input and returns a DataFrame containing model predictions



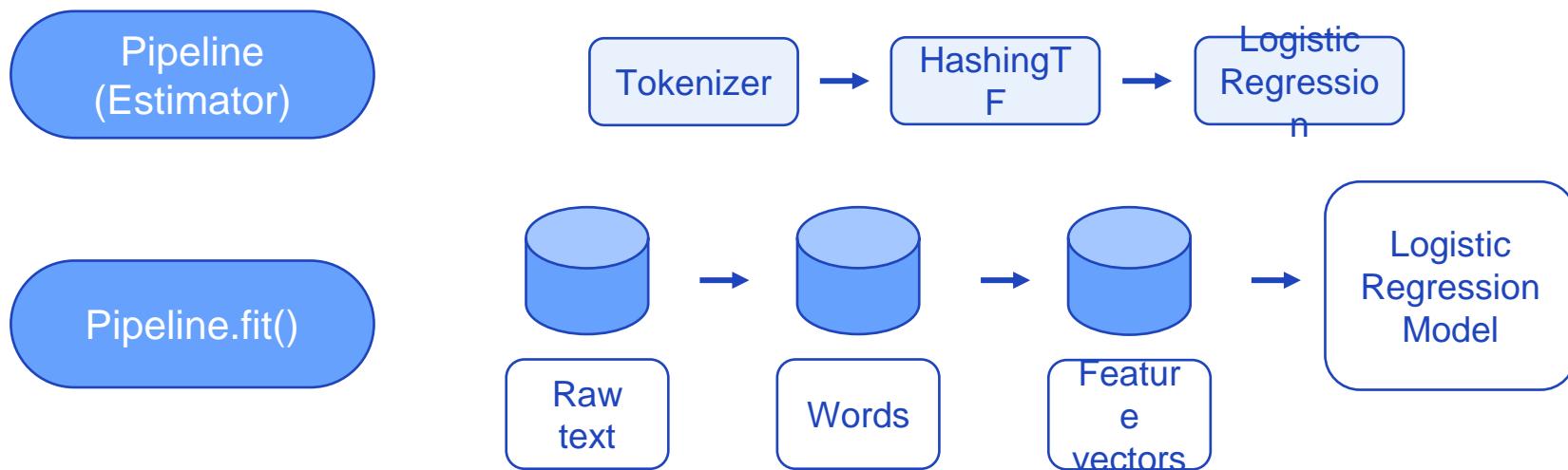
Pipelines (3/3)

- Comparison with and without Pipeline



ML workflow without Pipeline

- Now the Pipeline that incorporates all the transformers and estimators can be used directly against new data



Create a Spark ML without pipeline (1/2)

- Example in Python

```
# Prepare training data from a list of (label, features) tuples.
training = spark.createDataFrame([
    (1.0, Vectors.dense([0.0, 1.1, 0.1])),
    (0.0, Vectors.dense([2.0, 1.0, -1.0])),
    (0.0, Vectors.dense([2.0, 1.3, 1.0])),
    (1.0, Vectors.dense([0.0, 1.2, -0.5]))], ["label", "features"])

# Create a LogisticRegression instance. This instance is an Estimator.
lr = LogisticRegression(maxIter=10, regParam=0.01)

# Learn a LogisticRegression model. This uses the parameters stored in lr.
model1 = lr.fit(training)
# We may alternatively specify parameters using a Python dictionary as a paramMap
paramMap = {lr.maxIter: 20}
paramMap[lr.maxIter] = 30 # Specify 1 Param, overwriting the original maxIter.
# Specify multiple Params.
paramMap.update({lr.regParam: 0.1, lr.threshold: 0.55}) # type: ignore
# You can combine paramMaps, which are python dictionaries.
# Change output column name
paramMap2 = {lr.probabilityCol: "myProbability"} # type: ignore
paramMapCombined = paramMap.copy()
paramMapCombined.update(paramMap2) # type: ignore
```

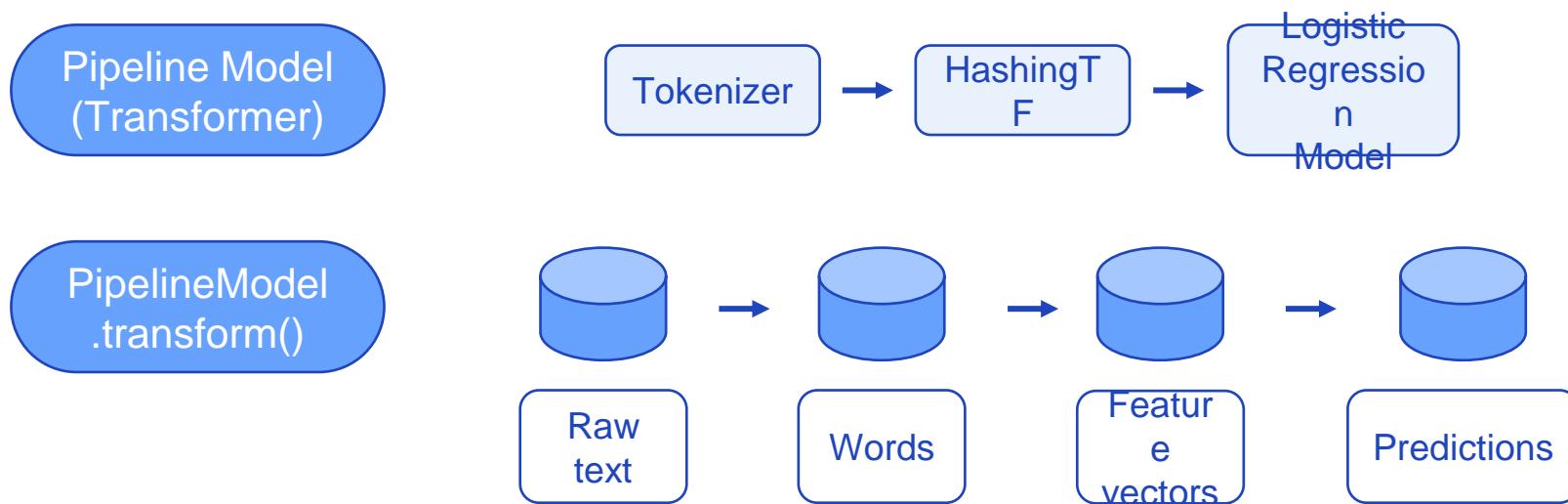
Create a Spark ML without pipeline (2/2)

- Example in Python

```
# Now learn a new model using the paramMapCombined parameters.  
# paramMapCombined overrides all parameters set earlier via lr.set* methods.  
model2 = lr.fit(training, paramMapCombined)  
  
# Prepare test data  
test = spark.createDataFrame([  
    (1.0, Vectors.dense([-1.0, 1.5, 1.3])),  
    (0.0, Vectors.dense([3.0, 2.0, -0.1])),  
    (1.0, Vectors.dense([0.0, 2.2, -1.5]))], ["label", "features"])  
  
# Make predictions on test data using the Transformer.transform() method.  
# LogisticRegression.transform will only use the 'features' column.  
# Note that model2.transform() outputs a "myProbability" column instead of the usual  
# 'probability' column since we renamed the lr.probabilityCol parameter previously.  
prediction = model2.transform(test)  
result = prediction.select("features", "label", "myProbability", "prediction") \  
    .collect()  
  
for row in result:  
    print("features=%s, label=%s -> prob=%s, prediction=%s"  
        % (row.features, row.label, row.myProbability, row.prediction))
```

ML workflow with Pipeline

- Now the Pipeline that incorporates all the transformers and estimators can be used directly against new data



Create a Spark ML with pipeline (1/2)

- Example in Python

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer

# Prepare training documents from a list of (id, text, label) tuples.
training = spark.createDataFrame([
    (0, "a b c d e spark", 1.0),
    (1, "b d", 0.0),
    (2, "spark f g h", 1.0),
    (3, "hadoop mapreduce", 0.0)
], ["id", "text", "label"])

# Configure an ML pipeline, which consists of three stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# Fit the pipeline to training documents.
model = pipeline.fit(training)
```

Input

Create a Spark ML with pipeline (2/2)

- Example in Python

```
# Prepare test documents, which are unlabeled (id, text) tuples.  
test = spark.createDataFrame([  
    (4, "spark i j k"),  
    (5, "l m n"),  
    (6, "spark hadoop spark"),  
    (7, "apache hadoop")  
, ["id", "text"])]  
# Make predictions on test documents and print columns of interest.  
prediction = model.transform(test)  
selected = prediction.select("id", "text", "probability", "prediction")  
for row in selected.collect():  
    rid, text, prob, prediction = row # type: ignore  
    print(  
        "(%d, %s) --> prob=%s, prediction=%f" % (  
            rid, text, str(prob), prediction # type: ignore  
        )  
    )  
  
(4, spark i j k) --> prob=[0.1596407738787412, 0.8403592261212588], prediction=1.000000  
(5, l m n) --> prob=[0.8378325685476614, 0.16216743145233858], prediction=0.000000  
(6, spark hadoop spark) --> prob=[0.06926633132976266, 0.9307336686702373], prediction=1.000000  
(7, apache hadoop) --> prob=[0.9821575333444208, 0.017842466655579203], prediction=0.000000
```

Unit 2

Spark Machine Learning Library

- | 2.1. Creating Spark MLlib Pipelines
- | 2.2. Classification and Regression
- | 2.3. Clustering & Collaborative Filtering

Machine Learning Recap

- Machine learning algorithms try to predict or make decisions based on training data
 - There are multiple types of learning problems, including classification, regression, or clustering. All of which have different objectives
- MLlib supported Supervised algorithm methods

Binary Classification Problems

linear SVMs, logistic regression, decision trees, random forests, gradient-boosted trees, naive bayes

Multiclass Classification Problems

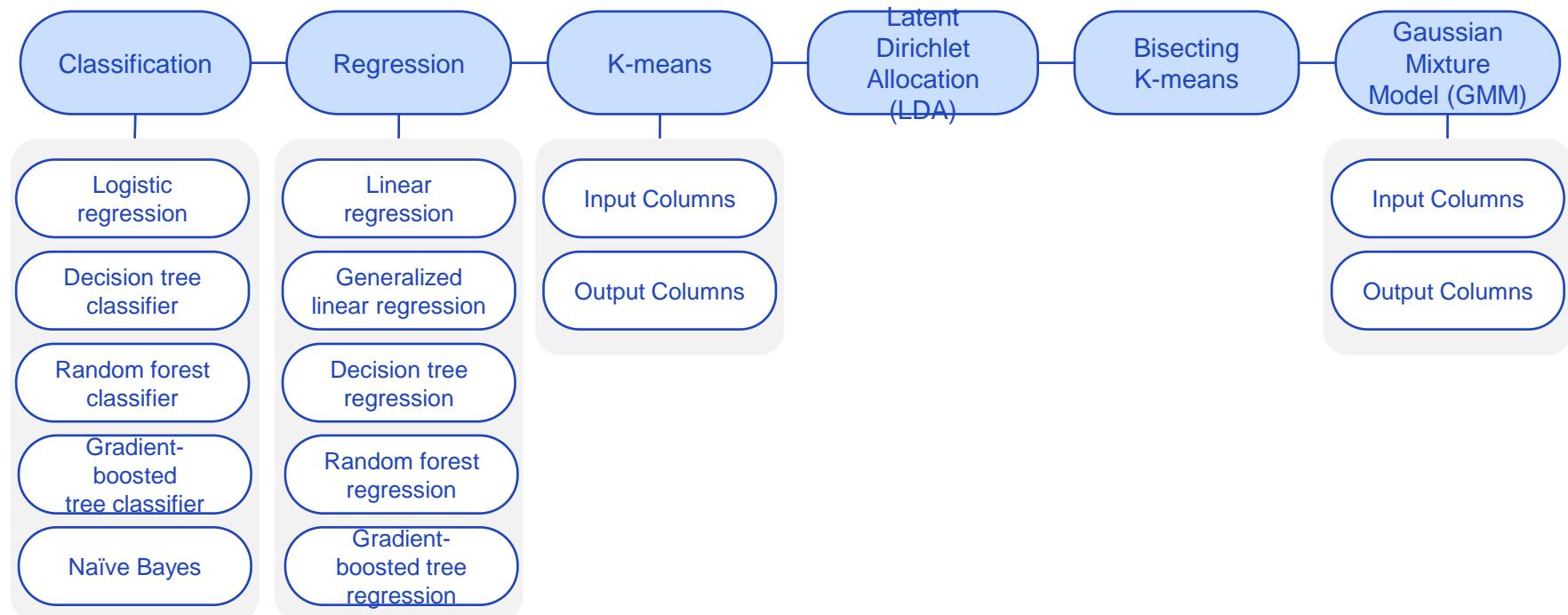
logistic regression, decision trees, random forests, naive Bayes

Regression Problems

linear least squares, Lasso, ridge regression, decision trees, random forests, gradient-boosted trees, isotonic regression

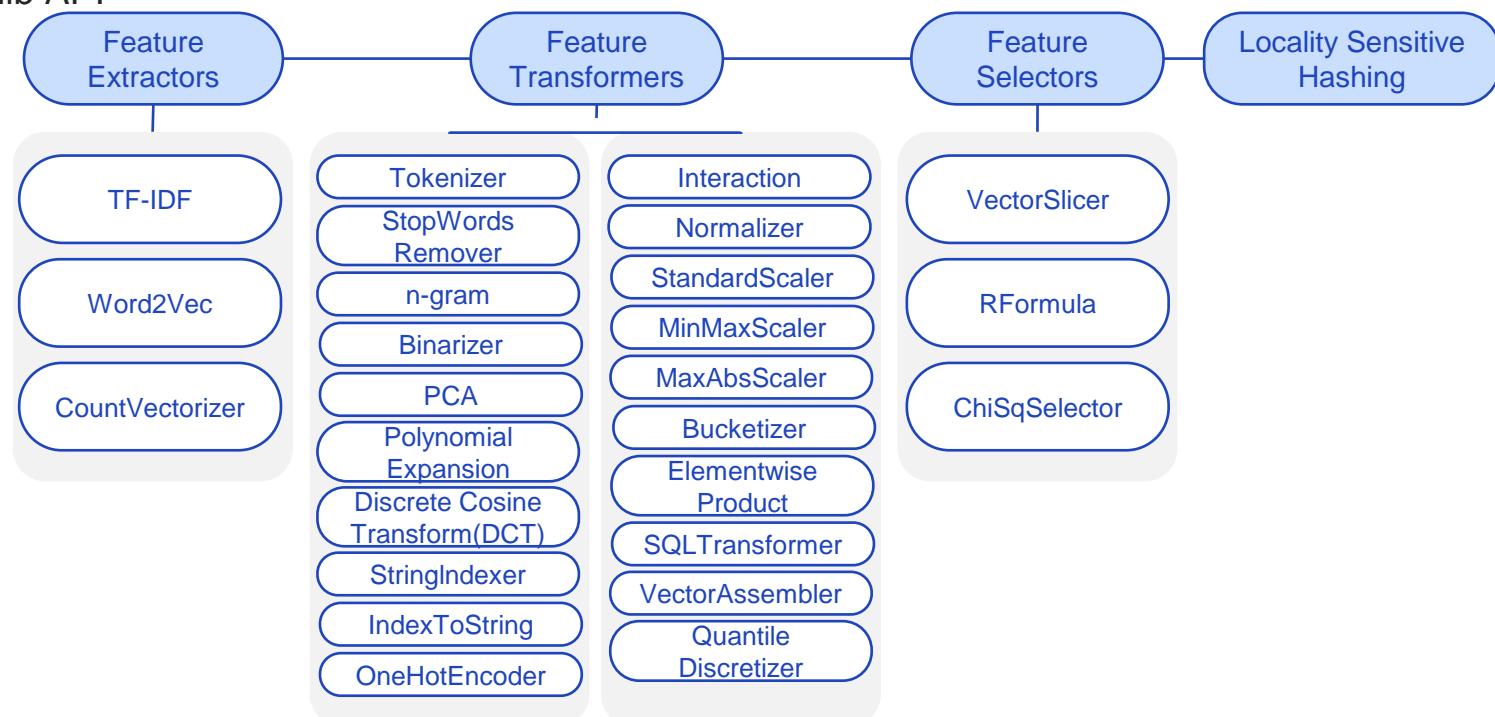
Algorithms by Spark MLlib

- Machine learning algorithms try to predict or make decisions based on training data
 - There are multiple types of learning problems, including classification, regression, or clustering



Pre-Processing by Spark MLlib

- Spark Programming + Spark SQL + Spark Streaming
- Spark MLlib API



Pre-Processing example

- Tokenizer -> StopWordsRemover -> Word2Vec

id	raw	filtered
0	[I, saw, the, red, balloon]	[saw, red, balloon]
1	[Mary, had, a, little, lamb]	[Mary, little, lamb]

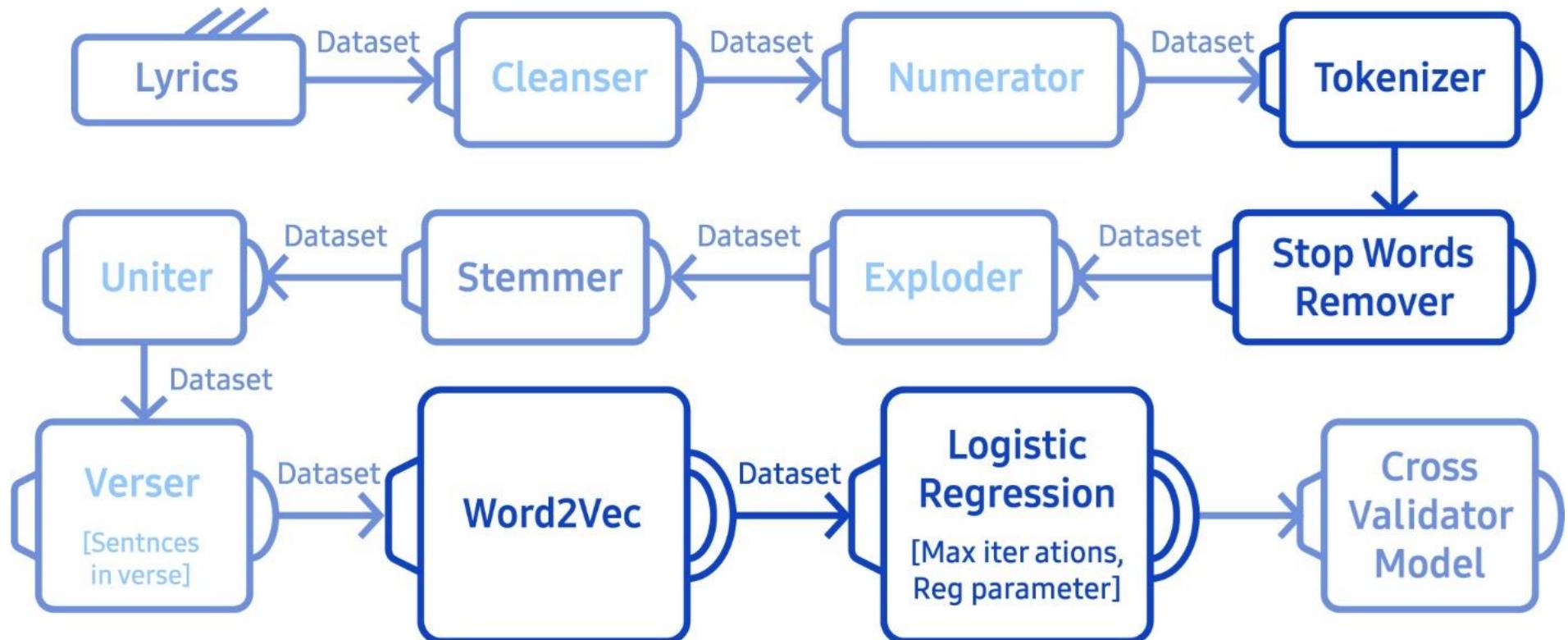
Input

```
import org.apache.spark.ml.feature.StopWordsRemover

val remover = new StopWordsRemover()
  .setInputCol("raw")
  .setOutputCol("filtered")
val dataSet = spark.createDataFrame(Seq(
  (0, Seq("I", "saw", "the", "red", "balloon")),
  (1, Seq("Mary", "had", "a", "little", "lamb")))
)).toDF("id", "raw")

remover.transform(dataSet).show(false)
```

Spark ML pipeline example



What is Classification?

- A supervised learning technique in which a Boolean label value y is predicted from a feature vector
- Learns from a set of training data that has been labeled with ground truth observations
- Determines a function (f) that when applied to x generates a positive (true) or negative (false) value that correlates to y

Binary Classification Example

- Predict if graduating student will get hired within 6 months based on data sets collected for each student
- Each observation is represented by a true/false
 - Each student is represented as a set of numbers

Feature Vectors

X_i

```
[3.8    2200    3.5     8    100000    100000    0.9     1     1]  
[3.3    2000    3.7     8     50000    100000    0.3     0     1]  
[3.0    2100    3.3     9     90000     30000    0.6     0     1]  
[3.0    2300    3.8     7     50000    150000    0.9     1     1]  
[2.5    1800    2.9    10     50000      0     0.7     1     0]  
[2.6    1900    3.0     8    100000     60000    0.4     0     1]
```

Labels *Y_i*

```
1  
-1  
1  
-1  
1  
1
```

Classification – Training Set

- Let's limit to Graduating GPA and Total Student Loan

Feature Vectors
 X_i

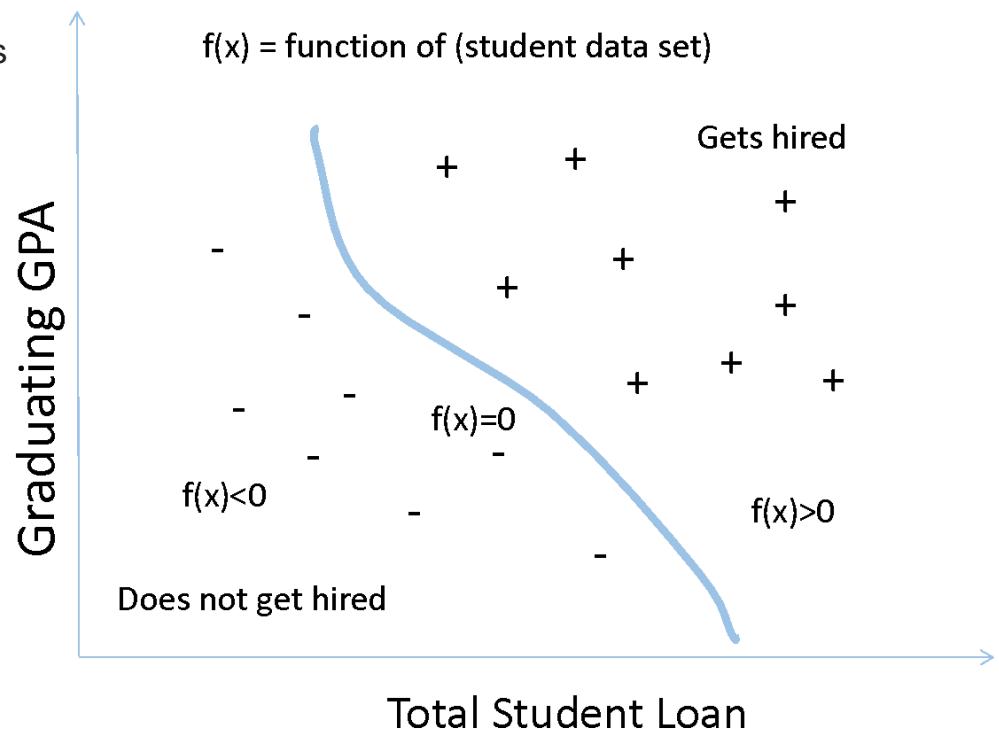
```
[3.5  
100000]  
[3.7  
50000]  
[3.3  
90000]  
[3.8  
50000]  
[2.9  
50000]  
[3.0  
100000]
```

Labels Y_i (Hired within 6 months)

```
1  
-1  
1  
-1  
1  
1
```

Formal Definition of Classification

- Given a training set (x_i, y_i) for $i=1..n$, where
 - x_i is a feature vector describing some characteristics
 - y_i is the label
- We want to create a classification model f that can predict label y for a new x



What is Regression Machine Learning?

- A supervised learning technique which is used to predict continuous values
- Divided into two packages:
 - spark.mllib contains the original API built on top of RDDs
 - spark.ml provides higher-level API built on top of DataFrames
- Using spark.ml is recommended because with DataFrames the API is more versatile and flexible. Plan is to keep supporting spark.mllib along with the development of spark.ml

Regression Example

- Predict the future income of a student
 - Predict future income level of graduating student based on data sets collected for each student
- Training Set - each observation is represented by a set of numbers
 - Each student is represented as a set of numbers

Feature Vectors

X_i

```
[3.8    2200    3.5     8    100000    100000    0.9     1     1]  
[3.3    2000    3.7     8     50000    100000    0.3     0     1]  
[3.0    2100    3.3     9     90000     30000    0.6     0     1]  
[3.0    2300    3.8     7     50000    150000    0.9     1     1]  
[2.5    1800    2.9    10     50000      0     0.7     1     0]  
[2.6    1900    3.0     8    100000     60000    0.4     0     1]
```

Labels Y_i (Future Income)

```
120  
100  
90  
150  
100  
95
```

Simple Linear Regression – data set

- Only a single feature is included from the observation
 - Each student is represented with a single feature X_i , the Outgoing GPA

Feature Vectors
 X_i

[3.5]
[3.7]
[3.3]
[3.8]
[2.9]
[3.0]

Labels Y_i (Future Income)

120
100
90
150
100
95

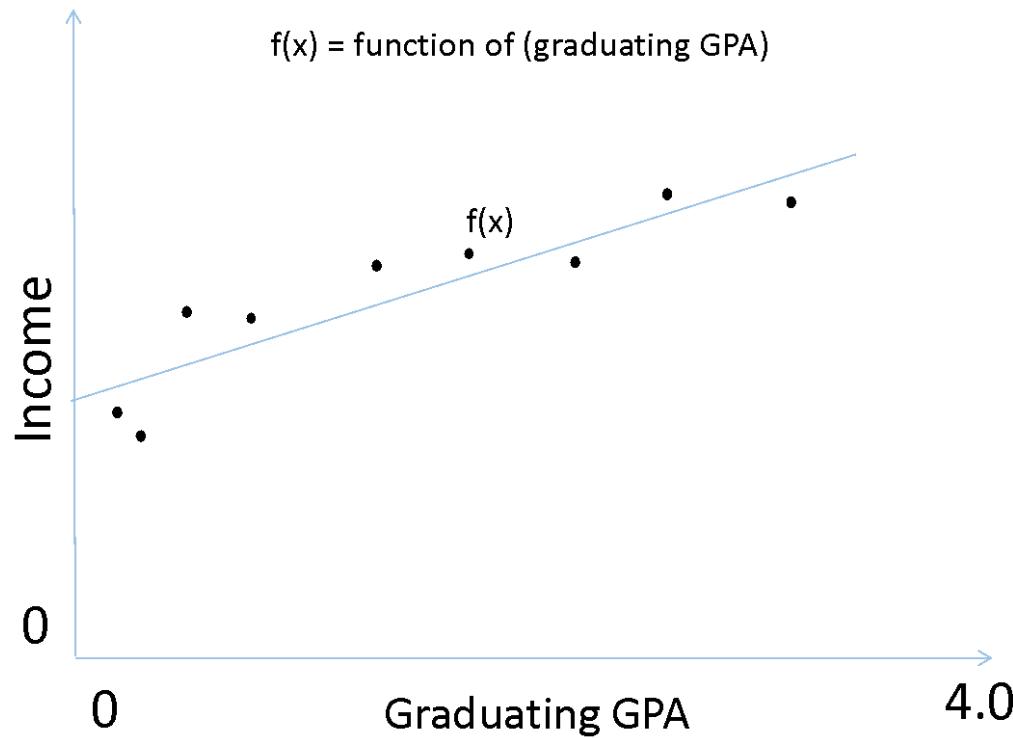
Simple Linear Regression - Function

- | A simple linear equation

- |
 - ▶ $f(x_i) = \beta_0 + \beta_1 * x_i$
 - ▶ $f(x_i) = 75 + 20 * x_i$

- | function(graduation GPA)

- |
 - ▶ $=75K + 20K * \text{GPA}$



Sample Logistic Regression

- Using PySpark

```
from pyspark.mllib.classification import LogisticRegressionWithLBFGS, LogisticRegressionModel
from pyspark.mllib.regression import LabeledPoint

# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split(' ')]
    return LabeledPoint(values[0], values[1:])

data = sc.textFile("data/mllib/sample_svm_data.txt")
parsedData = data.map(parsePoint)

# Build the model
model = LogisticRegressionWithLBFGS.train(parsedData)

# Evaluating the model on training data
labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
print("Training Error = " + str(trainErr))

# Save and load model
model.save(sc, "myModelPath")
sameModel = LogisticRegressionModel.load(sc, "myModelPath")
```

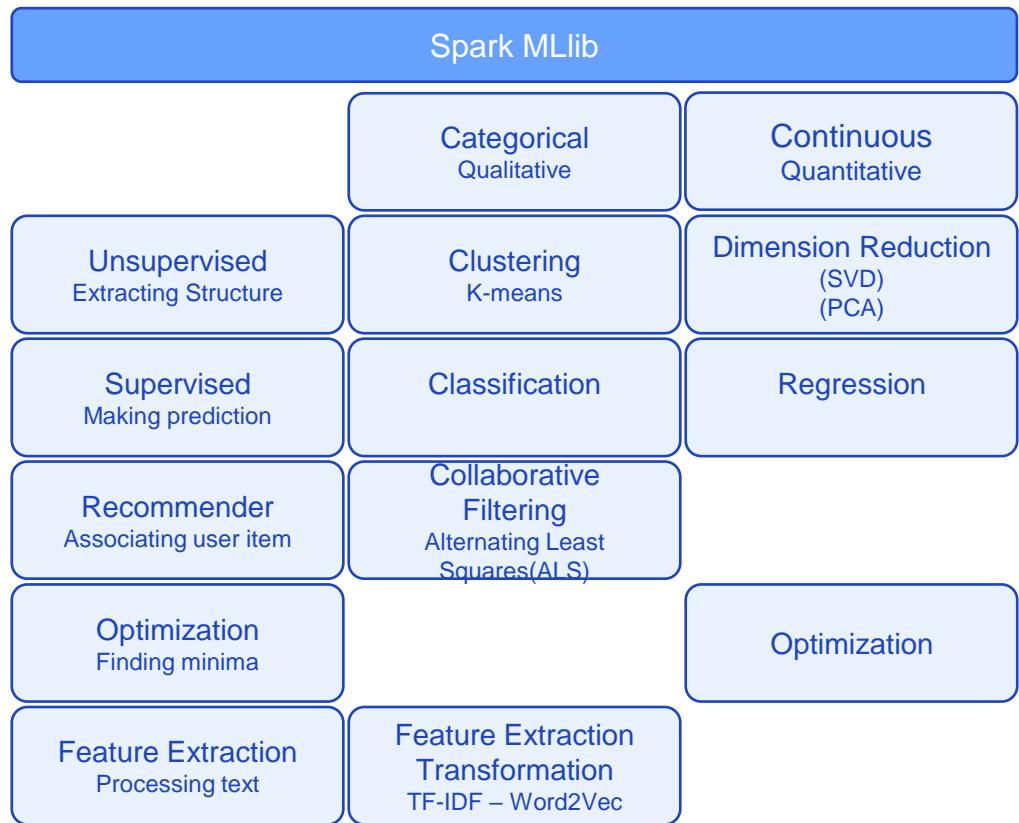
Unit 2

Spark Machine Learning Library

- | 2.1. Creating Spark MLlib Pipelines
- | 2.2. Classification and Regression
- | 2.3. Clustering & Collaborative Filtering

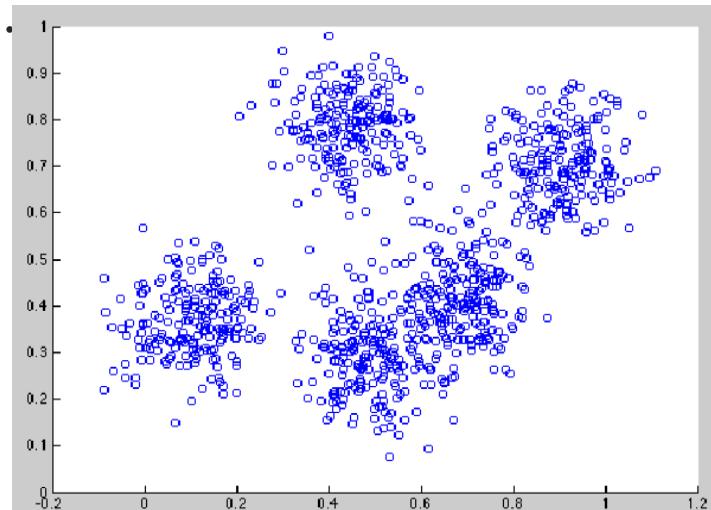
Spark MLlib Recap

- MLlib features
 - Utilities: linear algebra, statistics, etc.
 - Features extractor, Feature Transformer, Feature Selector
 - Regression
 - Binary Classification, Multiclass Classification
 - Clustering
 - Collaborative filtering
 - Dimensionality reduction

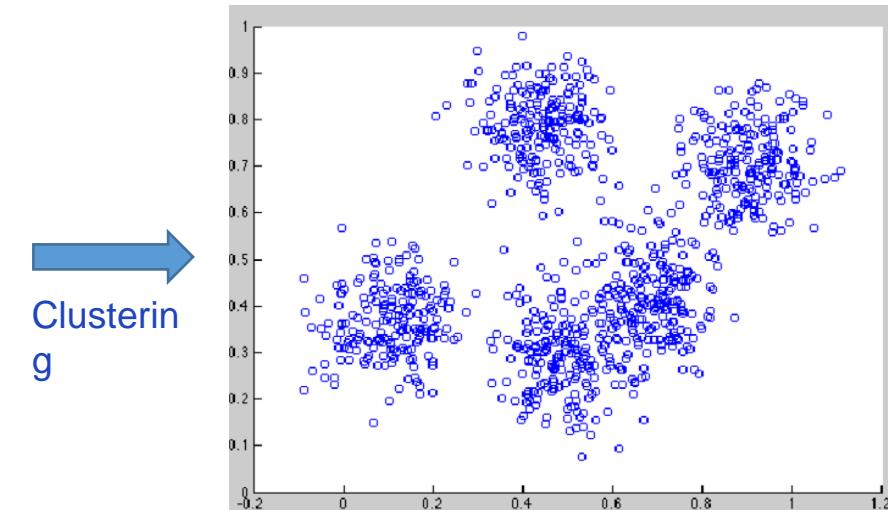


What does Clustering Do?

- Group items based on similar characteristics
 - Grouping customers with similar purchase behaviors
 - Automatically grouping documents, web pages, or other text
 - Grouping news items based on topic
 - Derive plant and animal taxonomies



Clustering



K-Means Clustering

- Group items based on similar characteristics
 - Input the number of clusters (K) and randomly initialize the centers
 - Assign every point in the data to closest one of these centers
 - For each of the cluster centers, move the center to the middle of that cluster (mean)
 - Reassign each point to the new closest center
 - Repeat from step 2 until convergence
- Train method key parameters
 - Rdd -> dataset name
 - k -> number of clusters
 - initializationMode -> how to initialize centers

Sample K-Means clustering

```
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt

# Load and parse the data
data = sc.textFile("data/mllib/kmeans_data.txt")
parsedData = data.map(lambda line: array([Float(x) for x in line.split('')]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations=10,
    runs=10, initializationMode="random")

# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y: x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))

# Save and load Model
clusters.save(sc, "myModelPath")
sameModel = KMeansModel.load(sc, "myModelPath")
```

Collaborative filtering

- Collaborative filtering is commonly used for recommender systems
- spark.mllib currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries
- spark.mllib uses the alternating least squares (ALS) algorithm to learn these latent factors
- Collaborative Filtering - These techniques aim to fill in the missing entries of a user-item association matrix
- MLLib currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries
- MLLib uses the alternating least squares (ALS)

The diagram illustrates a user-item association matrix. On the left, there are five user icons, each labeled with a blue dollar sign (\$) symbol. To the right of the matrix, there are five item icons: a mountain, an open book, a play button, and a video game controller. Above the matrix, there are two blue brackets: one spanning the user icons labeled 'N user' and another spanning the item icons labeled 'M item'. The matrix itself is a 5x5 grid of cells. Filled cells contain numerical ratings: (User 1, Item 1) has a rating of 5, (User 1, Item 2) has 1, (User 1, Item 3) has 3, (User 1, Item 4) has 5, (User 1, Item 5) has 2, and (User 2, Item 5) has 5. All other cells are empty.

	Mountain	Open Book	Play Button	Game Controller
User \$ 1	5		1	
User \$ 2			3	
User \$ 3		5		
User \$ 4			2	
User \$ 5				5

[Lab3]

Build a pipeline Machine Learning application



[Lab4]

Build Logistic Regression Model



[Lab5]

Build a Clustering Application



[Lab6]

Build Linear Regression Model



Chapter 8

Data Visualization

Big Data Course

Chapter Description

Objectives:

- ✓ In this chapter, we will learn about how to use visualization for our data. There are many open source tools to choose from but since we are on the Hadoop platform, we shall introduce HUE (Hadoop User Experience) and the visualization capabilities it provides
- ✓ We shall then move on to a much more general discussion of Visualization using commercial products. Compared to open sources, commercial products have much richer features.
 - While there are many good commercial products available, we will focus on MS Power BI in this chapter. MS Power BI is highest ranked visualization tool in the 2021 Gartner report and there is a free version on which we can easily get hands-on experience.

Contents of Chapter:

1. Open Source Visualization Tools
2. Popular Commercial Visualization Tool

Unit 1

Open Source Visualization Tools

Data Visualization

Unit 1

Open Source Visualization Tools

- | 1.1. Data Visualization Basics
- | 1.2. Introduction to Apache Hue and Jupyter

Why Visualization?

- Data has a story to tell
 - Good visualization tells that story
- Data Visualization is a form of art that helps grab our attention and focus on the correct message
- Good visualization helps us quickly see trends and understand the underlying data
- Visualization removes any technical or mathematical barriers to understand the data
 - A big spreadsheet with interrelated functions and formulas will not convey anything to the non-technical



Benefits of Good Visualization

- Audience is able to view and absorb information much more easily
- Identify emerging patterns and trends
 - Help link an organizations operational strategy with business outcome
 - Predict sales and revenue opportunities
- Help organizations identify issues and shortcomings and make adjustments quickly that help their bottom line



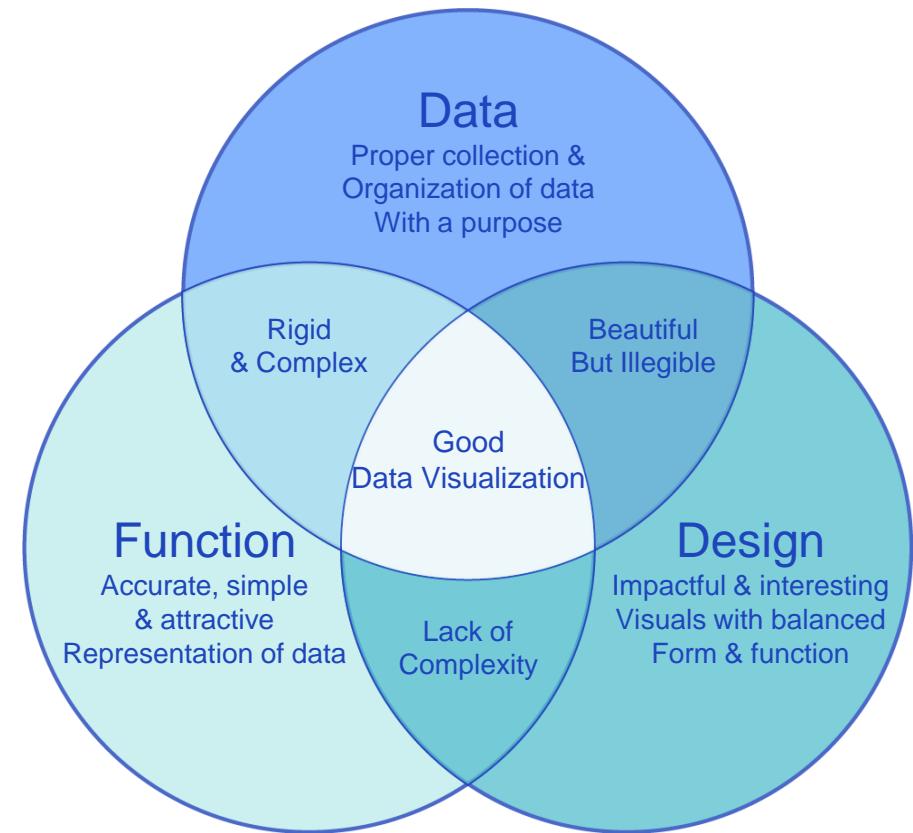
The Right Visualizations

- A presentation and the visualization within it has a specific target audience
 - Visualization must be designed with the audience and their needs in mind
 - Marketing and Sales
 - Product Planning
 - Finance and Accounting
- Choose the right visualization techniques for the type of underlying data
 - Tables for records
 - Line charts for tracking changes
 - Bar charts to compare quantities



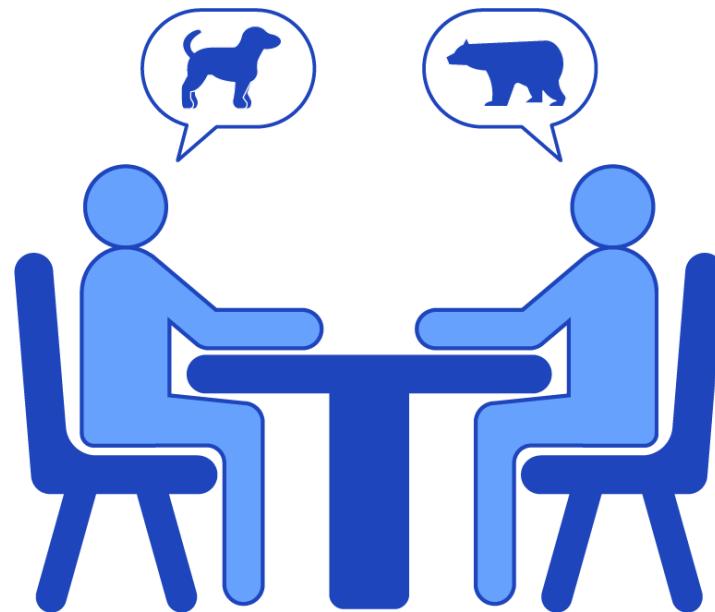
Ingredients to a Good Visualization

- Good Data
 - Garbage-in □ Garbage out
 - Data from many sources, well integrated, is the starting ingredients for a good visualization
- Good Functions
 - Functional representation of the data that makes it easier to see and understand
 - Mixing and matching, slicing and dicing - a good chef knows how to take the ingredients and turn it into a special dish
- Good Design
 - Interesting visuals that keep the audience interested in the data
 - Art and form - the dish is not complete without the finishing presentation touches



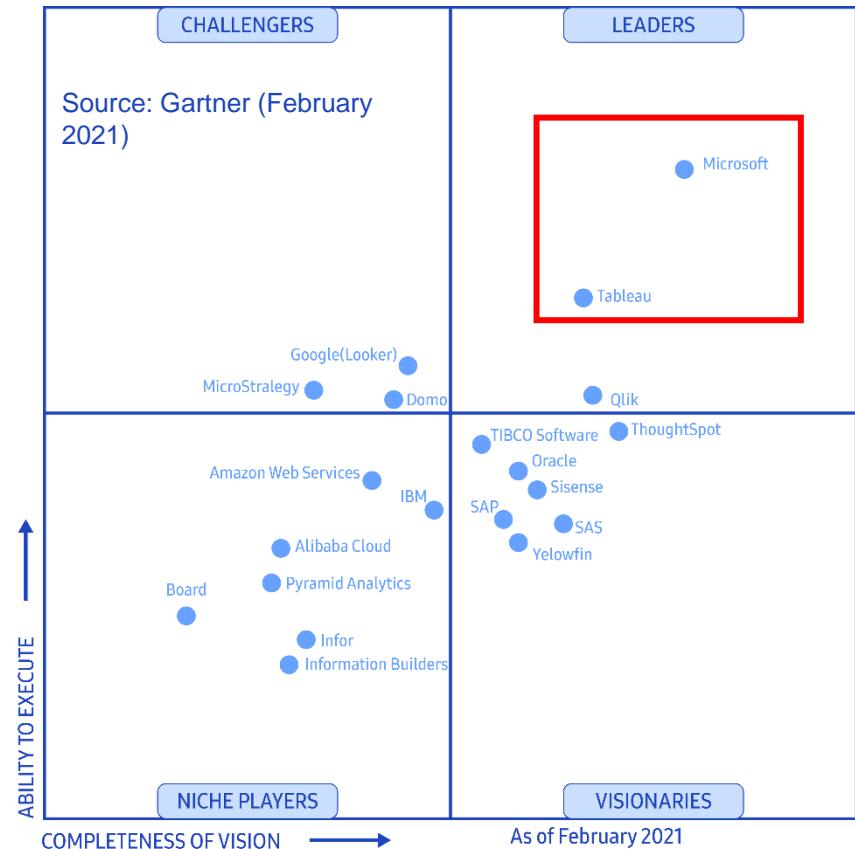
Finally But Not Least - Actually, FIRST!!!

- Data, design, function are all key components to a good presentation
- However, a good visualization starts before a single chart is ever created
- It starts with the client interview
 - What is the question that the client is trying to answer?
 - Where and how can the data be collected and wrangled to answer the question
 - Make sure that you and the client fully understand the question and the expected results



Gartner Magic Quadrant

- Gartner's Magic Quadrant is an industry standard for IT professionals to compare and gain insight to various competing products currently available in the market place
- The X-Axis is completeness of vision
 - This axis shows which products have basic features as well as vision of product that make them stand out from the crowd
- The Y-Axis is ability to execute
 - This axis show how well each product has been able to follow through with their vision with actual delivery to an easy to use and well thought out product
- The top right quadrant represents products with full vision and also able to deliver on that vision



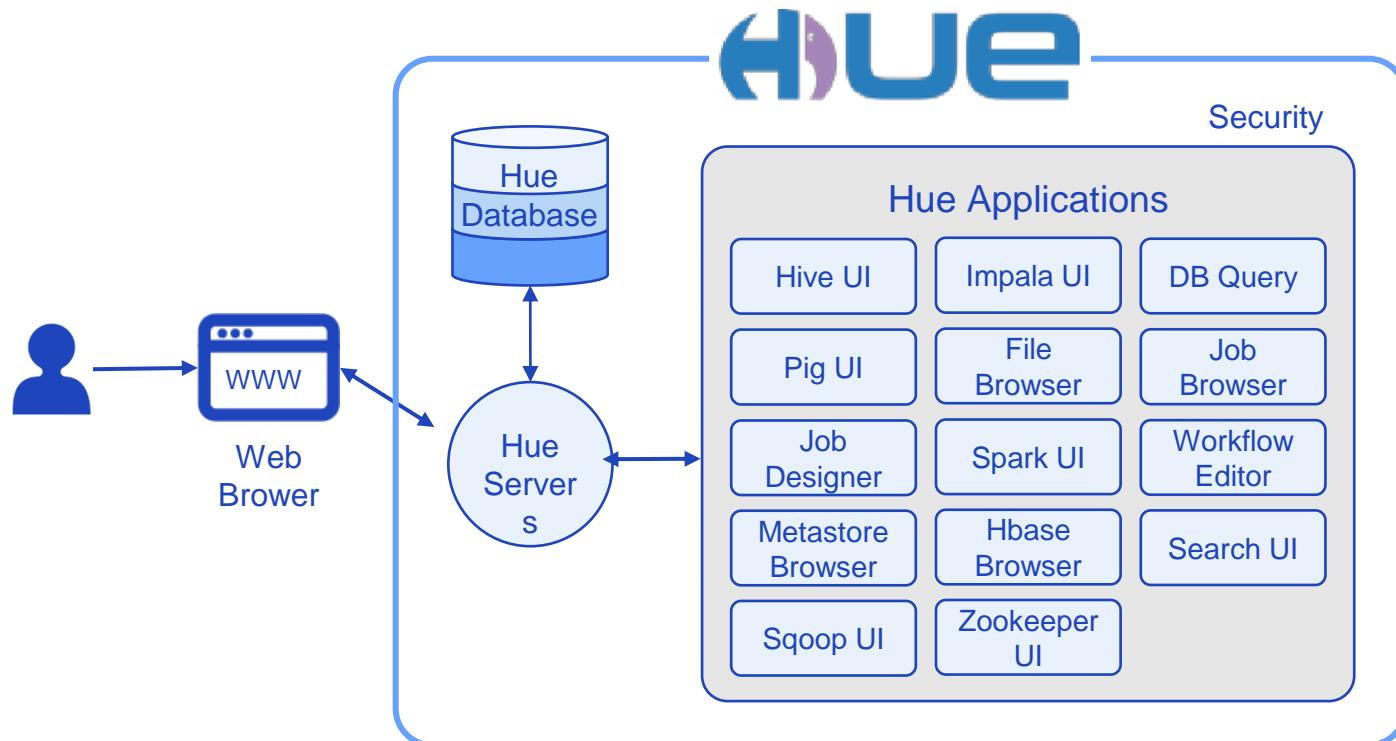
Unit 1

Open Source Visualization Tools

- | 1.1. Data Visualization Basics
- | 1.2. Introduction to Apache Hue and Jupyter

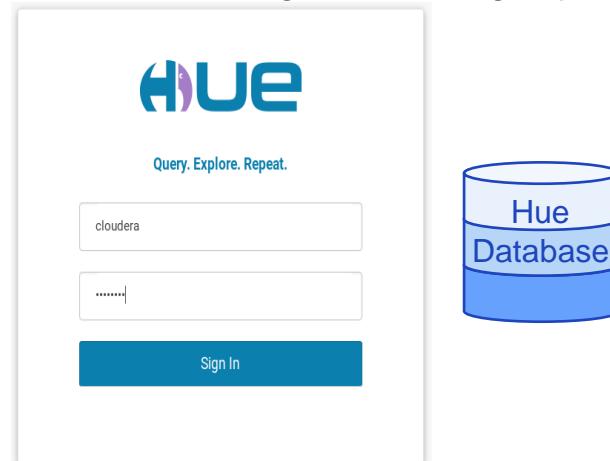
What is HUE?

- Provides a web interface for interacting with a Hadoop cluster



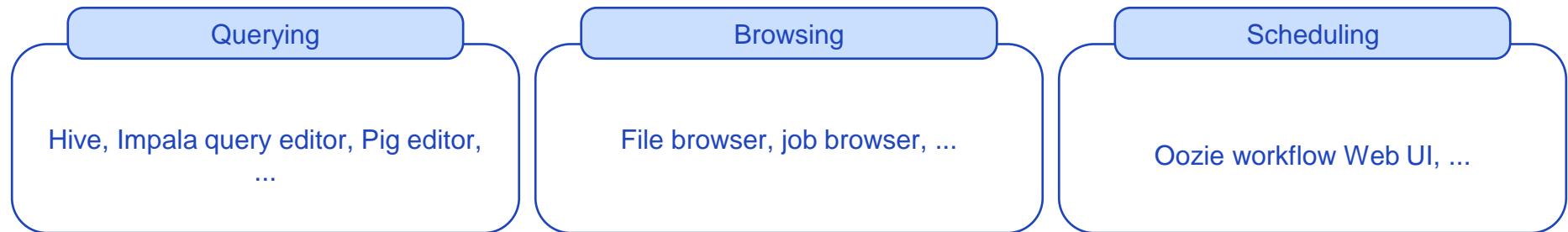
Accessing HUE

- SuperUser (First User Login)
- Access Hue from browser
 - http://<hue_server>:8888 (Non-secure , w/o Hue balancer)
- The user who logs in to Hue for the first time has Superuser privileges
 - Superuser has the authority to create and manage users and groups, and to designate superusers



Why HUE?

- Main Functionalities of Hue



Hue : Query Editor

- Provides Querying Web UI for various Hadoop services

The screenshot shows the Apache Hue Query Editor web interface. At the top, there's a navigation bar with the Hue logo, a 'Query' dropdown menu, and a search bar labeled 'Search data and saved documents...'. Below the search bar is a 'Jobs' section with links for 'Assistant' and 'Functions'. On the left, there's a sidebar titled 'Tables' showing three tables: 'products', 'test', and 'test2'. The main area has a 'Query' dropdown menu open, showing options like 'Editor', 'Dashboard', 'Hive', 'Pig', 'Java', 'Spark', 'MapReduce', 'Shell', 'Sqoop 1', and 'Distcp', along with a 'Add more...' button. To the right of the dropdown is a large text input field with placeholder text 'Add a description...', a file browser icon, and a help icon. Below the input field are buttons for 'default', 'text', and a gear icon. A tooltip 'filename, or press CTRL + space' is shown over the input field. To the right of the input field is a sidebar titled 'Impala' with a search bar and a list of functions: Aggregate, Analytic, Bit, Conditional, Date, Mathematical, Misc, String, and Type Conversion.

Hue : Querying (1/6)

- Main Areas

The screenshot shows the Apache Hue web interface with three main panels highlighted by red boxes:

- Quick Browse**: On the left, a sidebar showing the "default" database with tables "products", "test", and "test2".
- App**: The central panel for querying. It includes a "Top search" bar, a query input field ("Add a name... Add a description..."), a preview area with the placeholder "Example: SELECT * FROM tablename, or press CTRL + space", and a "Query History" section listing recent queries.
- Assistant**: A sidebar on the right containing a list of functions categorized under "Impala": Aggregate, Analytic, Bit, Conditional, Date, Mathematical, Misc, String, and Type Conversion.

Hue : Querying (2/6)

- Query Editor for Hive / Impala

The screenshot shows the Apache Hue Query Editor interface. A red box highlights a specific block of code:

```
30 room_type,
31 price as `price($)`
32 from ab_nyc_2019
33 where price < 150.0 and neighbourhood_group = 'Manhattan'
34 order by `price($)` ,location desc;
35
36 select location, room_type, count(*) as avail_rooms
37 from (
38 select name as room_descriptions,
39 host_name,
40 concat(neighbourhood,' ,neighbourhood_group) as location,
41 room_type,
42 price as `price($)`
43 from ab_nyc_2019
44 where price < 150.0 and neighbourhood_group = 'Manhattan'
45 ) t1
46 group by location, room_type
47 order by avail_rooms desc;
```

Annotations with blue arrows and text:

- Provides Query Save and Load functions
- Space to write Query
- DB in use
- Executable only for the specified block

Hue : Querying (3/6)

- Query Editor for Hive / Impala

DB / Table information

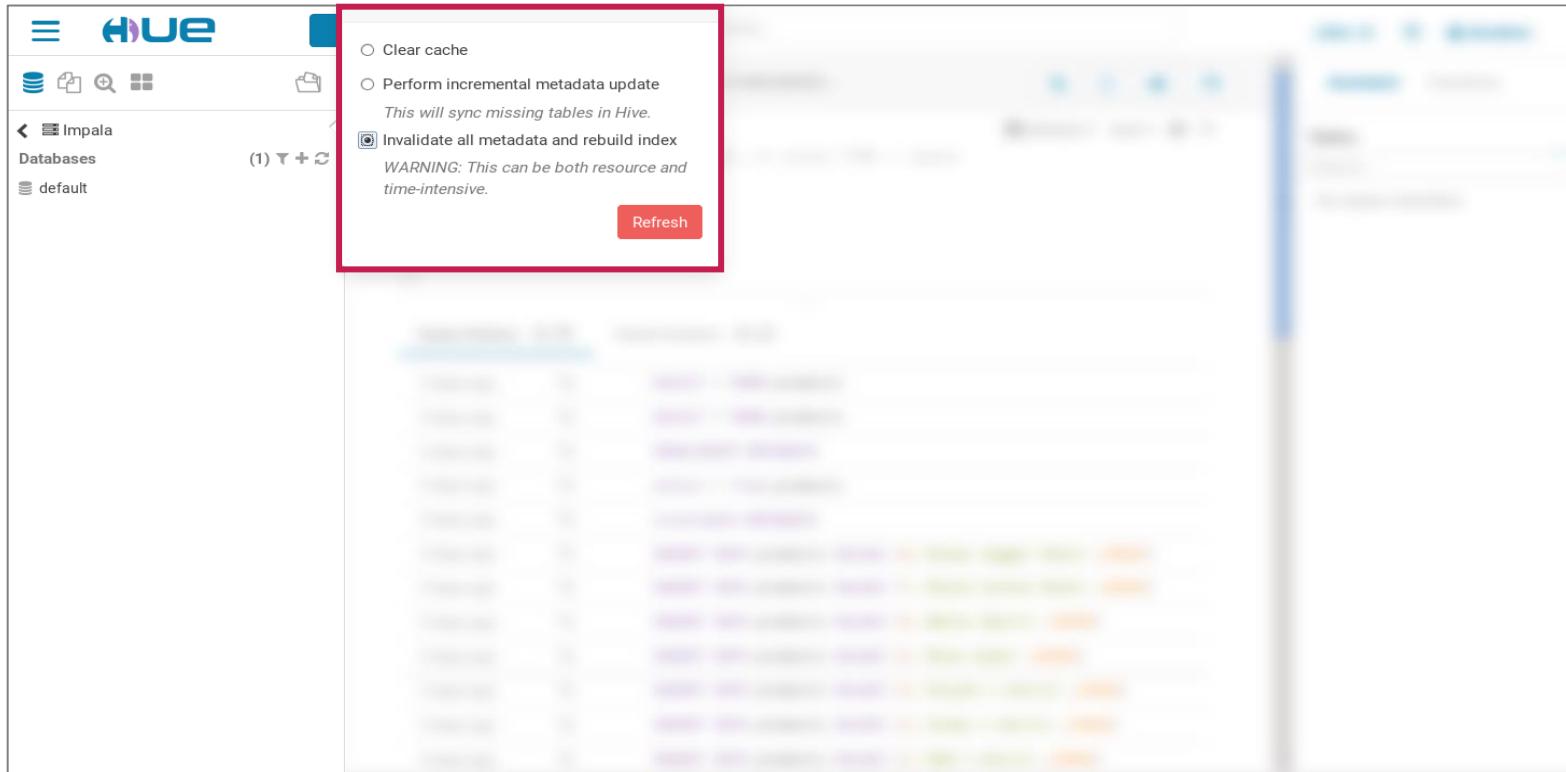
The screenshot shows the Apache Hue interface. On the left, there's a sidebar titled 'airbnb' with a 'Tables' section containing a list of tables and their column details. A red box highlights this sidebar. On the right, a larger window displays the 'ab_nyc_2019' table with columns 'ab_nyc_2019.id' and 'ab_nyc_2019.name'. The 'Sample' tab is selected, showing 8 rows of data. A blue box highlights this window. Below the table, the text 'Provides detailed information about the clicked table' is displayed.

ab_nyc_2019.id	ab_nyc_2019.name
1	Clean & quiet apt home by the park
2	Skylit Midtown Castle
3	THE VILLAGE OF HARLEM...NEW YORK !
4	Cozy Entire Floor of Brownstone
5	Entire Apt: Spacious Studio/Loft by central parl
6	Large Cozy 1 BR Apartment In Midtown East
7	BlissArtsSpace!
8	Large Furnished Room Near B'way

• Provides detailed information about the clicked table

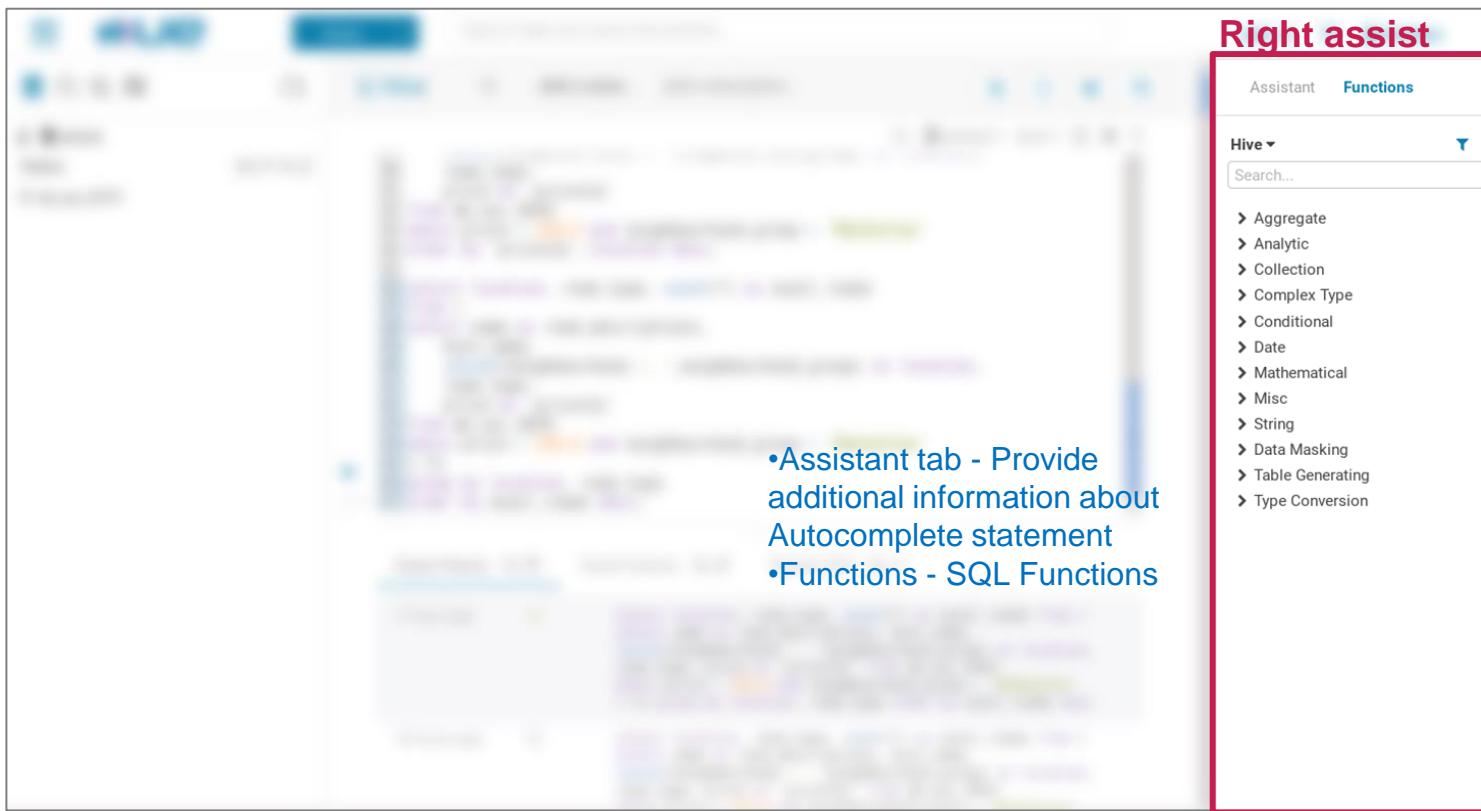
Hue : Querying (4/6)

- Impala - Invalidate metadata



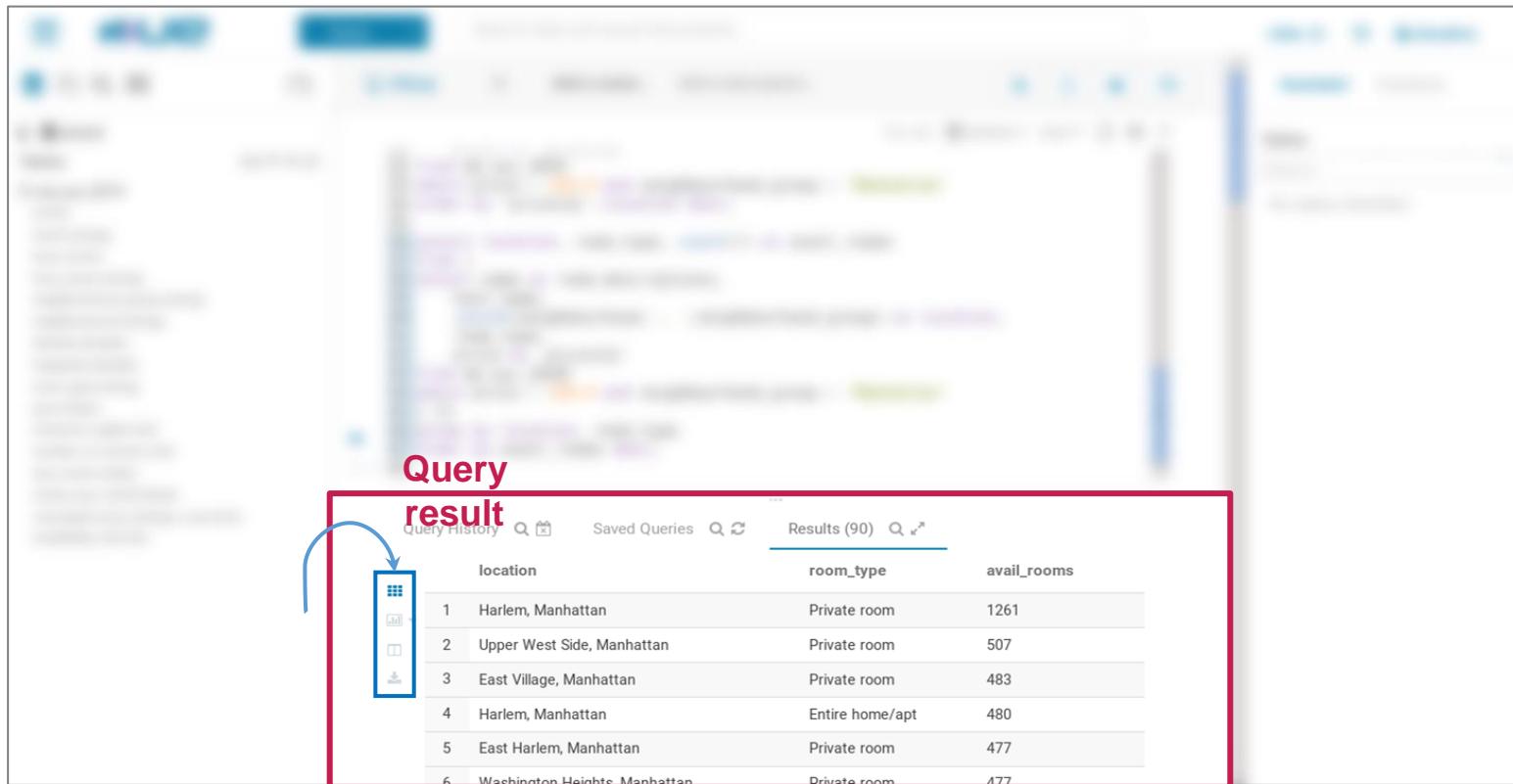
Hue : Querying (5/6)

- Right assist



Hue : Querying (6/6)

- Query result



The screenshot shows the Apache Hue interface with a blurred background. In the foreground, a red box highlights a table titled "Query result". The table has three columns: "location", "room_type", and "avail_rooms". The data is as follows:

	location	room_type	avail_rooms
1	Harlem, Manhattan	Private room	1261
2	Upper West Side, Manhattan	Private room	507
3	East Village, Manhattan	Private room	483
4	Harlem, Manhattan	Entire home/apt	480
5	East Harlem, Manhattan	Private room	477
6	Washington Heights, Manhattan	Private room	477

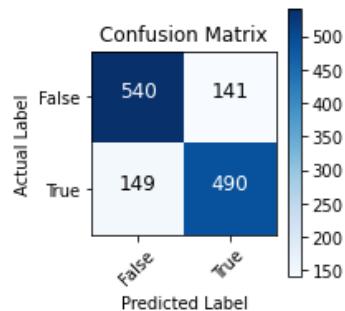
About Jupyter Notebook

- Jupyter notebook evolved from IPython notebook
- Jupyter notebook is an open-source interactive computing notebook for several different programming languages including Python
 - Actually, Jupyter notebook also supports Julia, R, Haskell, Ruby, etc.
- It is a web-based application; runs on a browser
- A big advantage of Jupyter is its ability to combine formatted text and code

XGBoost Prediction Result

The confusion matrix shows number of predictions for each of the Correct (Positive and Negative) prediction and Incorrect (Positive and Negative) predictions

```
In [46]: print_confusion_matrix(xgb, x_test, y_test, "xgboost")
```



	precision	recall	f1-score	support
0.0	0.78	0.79	0.79	681
1.0	0.78	0.77	0.77	639
accuracy			0.78	1320
macro avg	0.78	0.78	0.78	1320
weighted avg	0.78	0.78	0.78	1320

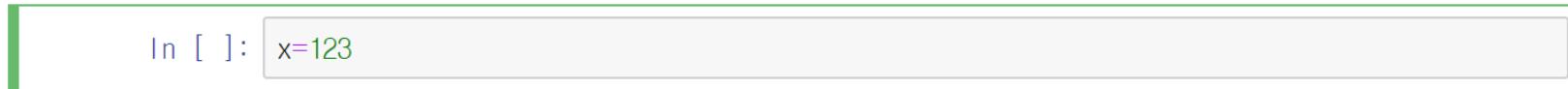
Features of Jupyter Notebook

- Some of note-worthy features of Jupyter notebook are:
 - Function auto-complete
 - Codes are organized in cell units
 - Supports Markdown, LaTex, etc. for formatting text
- To run the notebook, run the following command at the terminal

```
$ jupyter notebook
```

Jupyter Notebook Introduction (1/5)

- When you click on a cell, its border turns green. This is the **edit mode**



- Some useful shortcuts while in the edit mode are:

Key Stroke	Action
CTRL + a	Select the whole line.
CTRL + d	Delete the whole line.
CTRL + z	Undo the last change.
CTRL + s	Save and checkpoint.

- From the command mode you can switch to the edit mode by pressing the [ENTER] key.
- In both the edit and the command modes, press [SHIFT] + [ENTER] to run or compile the selected cell

Jupyter Notebook Introduction (2/5)

- You can enter into the **command mode** by selecting a cell and then pressing the [ESC] key



- Some useful shortcuts while in the **command mode** are:

Key Stroke	Action
a	Insert a cell above.
b	Insert a cell below.
d + d (twice)	Delete the current cell.
m	Change the cell type to Markdown .
y	Change the cell type to code .

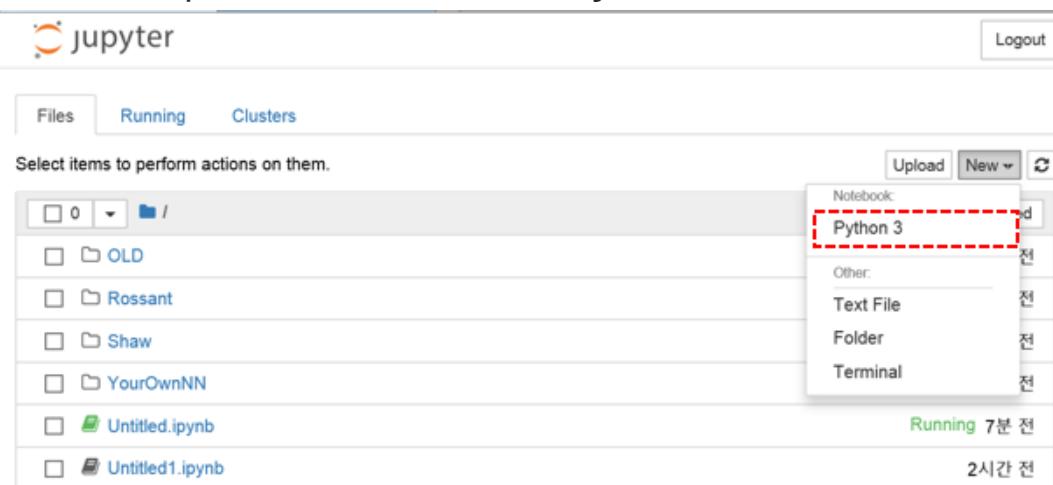
Jupyter Notebook Introduction (3/5)

- Some of the useful Markdown tags:

Tag	Action
#	Title largest.
##	Title next largest.
- , *, or number	List items.
>	Quotes.
[text](URL)	Hyperlink.
* <i>Italic</i> *	Italic font.
** Bold **	Bold font.
\$ ~ \$, \$\$ ~ \$\$	LaTex expression.
-----	Horizontal line.
 	Line break.

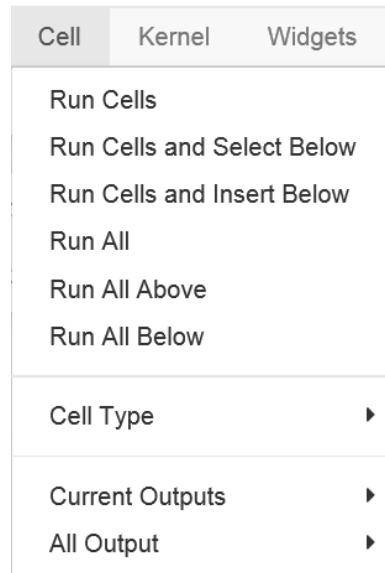
Jupyter Notebook Introduction (4/5)

- From the **Home** page click on the **Running** tab to view all the running notebooks
 - You can click on the Shutdown button to terminate a notebook
- Press the **New** dropdown and then select **Python 3** to start a new notebook



Jupyter Notebook Introduction (5/5)

- Enter into a notebook, then select **Cell** → **Run All** to execute all the cells from top to bottom in a row



```
In [27]: # Creating a Logistic Regression Model by fitting the data
lr = LogisticRegression(lr=0.01, num_iter=1000, verbose=True)
lr_log = lr.fit(x_train_lr, y_train_lr)

epoch: 0      loss: 0.6931471803599453
epoch: 10     loss: 0.6904899637675816
epoch: 20     loss: 0.6889239094938908
epoch: 30     loss: 0.6877765367187595
epoch: 40     loss: 0.6868688414532079
epoch: 50     loss: 0.6861264028512316
epoch: 60     loss: 0.685506647126465
epoch: 70     loss: 0.6849811804514382
epoch: 80     loss: 0.6845297410991716
epoch: 90     loss: 0.6841373605861225
epoch: 100    loss: 0.6837927364357145
epoch: 110    loss: 0.6834871959245932
epoch: 120    loss: 0.683213997239752
epoch: 130    loss: 0.682967841058459
epoch: 140    loss: 0.6827445205184289
epoch: 150    loss: 0.6825406655388402
epoch: 160    loss: 0.6823535532062365
epoch: 170    loss: 0.6821809653950383
epoch: 180    loss: 0.6820210807339536
epoch: 190    loss: 0.6818723918967042
```

[Lab1] Working with Query editor in Hue



[Lab2] Using Maps function in HUE



Unit 2

Popular Commercial Visualization Tool

| Data Visualization

Unit 2

Popular Commercial Visualization Tool

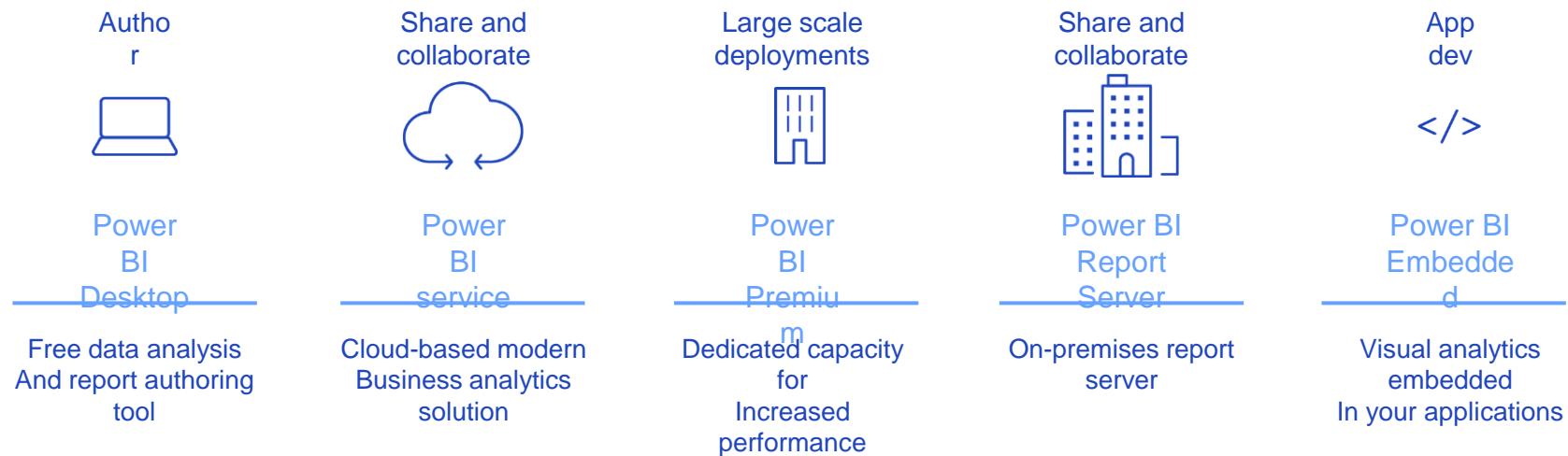
| 2.1. Introduction to Power BI

What is Power BI



Power BI Tool Services

- What can we do with Power BI?

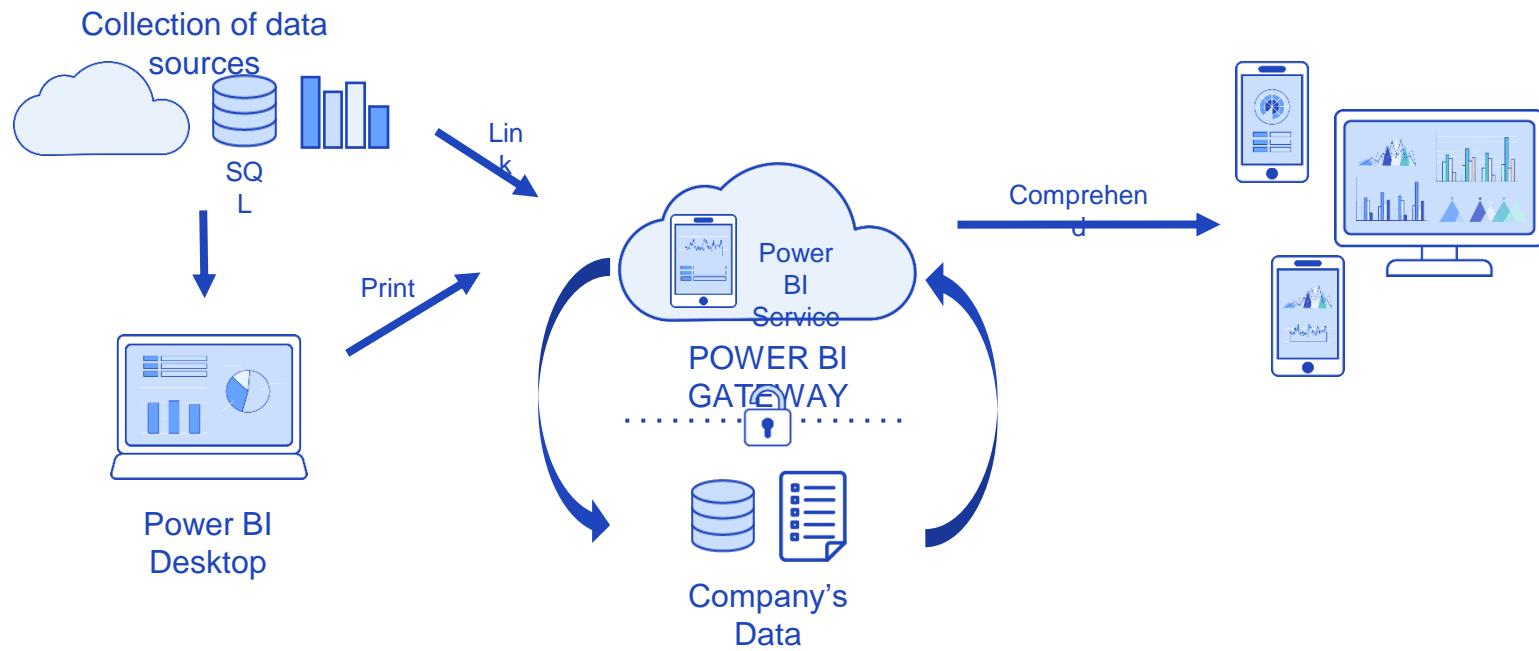


Power BI Components

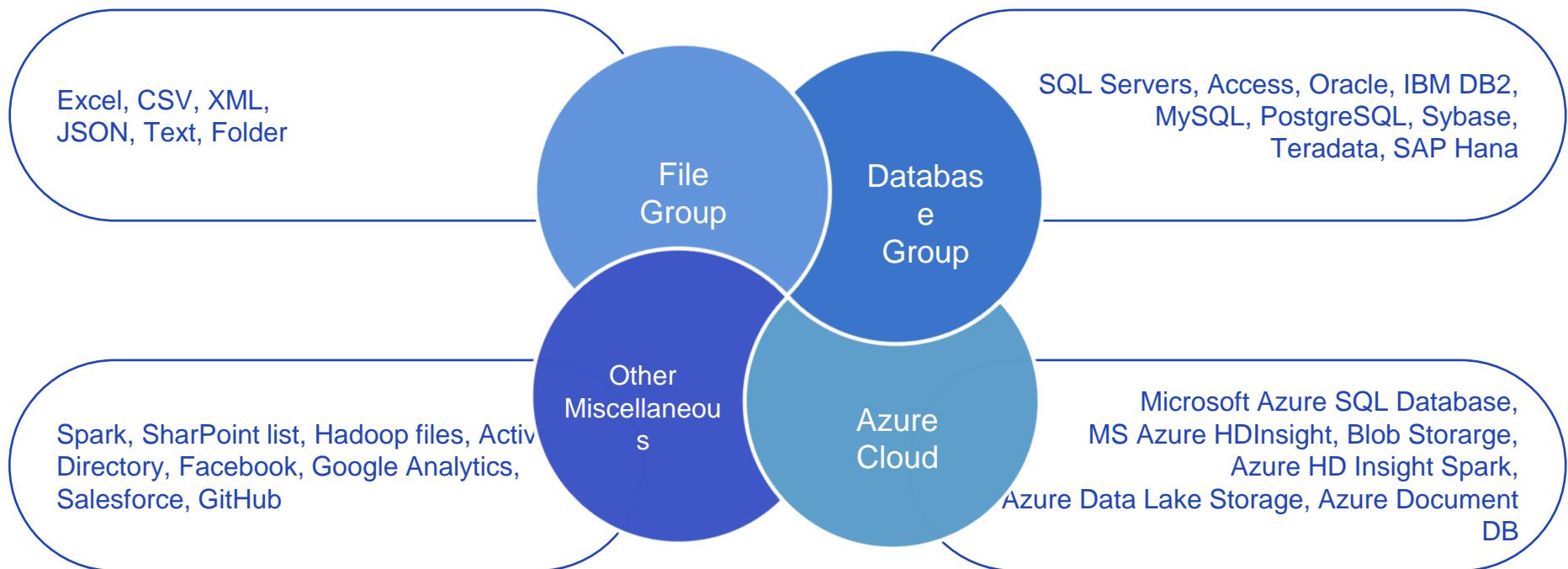
- Power Query** Service to access, search and transform data from various data sources.
Data mash-up supported
- Power Pivot** Provides tools for in-memory to model data
- Power View** Tools to graphically represent data using visuals and use them for analysis
- Power Map** Tools and capabilities to visualize geo-spatial data on 3D models in a map
- Power Q&A** Search or discover insights from your data by entering natural query languages

Data and Process Work Flow

- Basic architecture and work flow of Power BI



Data Sources Supported by Power BI



Support for DAX formulas and R Scripts

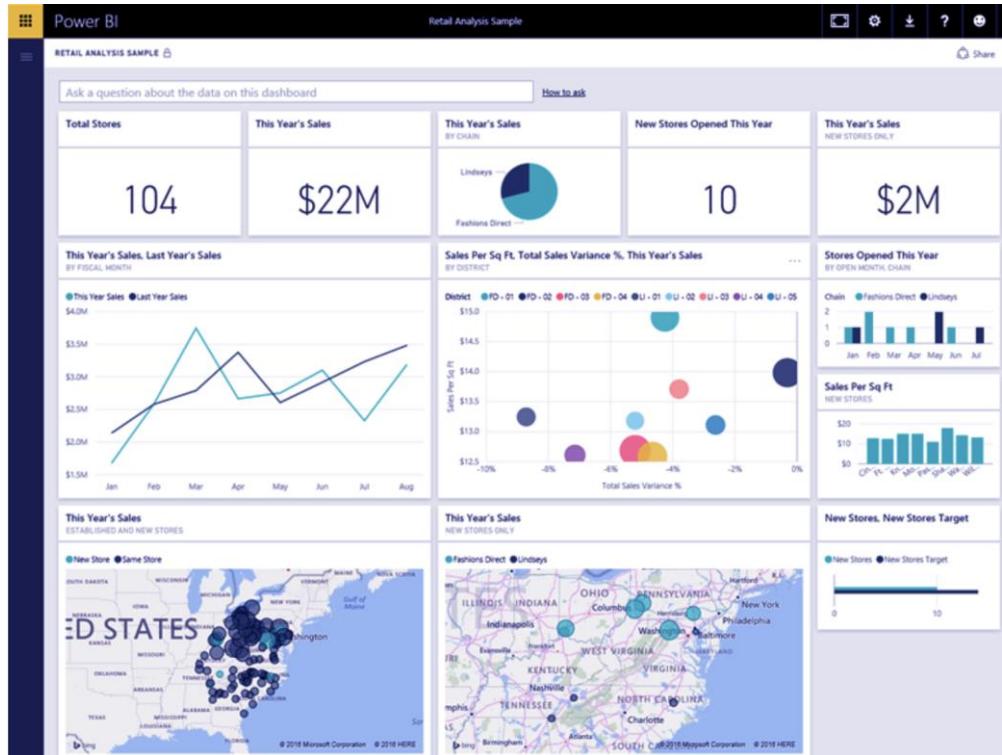
- DAX is Data Analysis Expression
 - A functional language
- A formula language used in analysis services
 - Includes functions, operators and values to perform advanced calculations and queries on data in related tables and columns in tabular data models

```
ProjectedSales2021 = SUM(Sales[TotalSales2020])*1.21
```

- ```
library(mice)
tempData <- mice(dataset, m=1, maxit=50, meth='pmm', seed=100)
completedData <- complete(tempData, 1)
output <- dataset
output$completedValues <- completedData$"SMI missing values"
```

# Building Blocks - Visualizations

- Power BI provides many diagrams, charts, shadings, and customizations
- The various visualizations have many customizable options that can be used to further highlight the core information that needs to be conveyed



# Building Blocks - Datasets

- Datasets are the data from the various supported data sources that ingested, cleansed and made ready for analysis.
- We have already seen the numerous types of data sources from which Power BI is able to create Datasets
- Shown here is an Excel based dataset

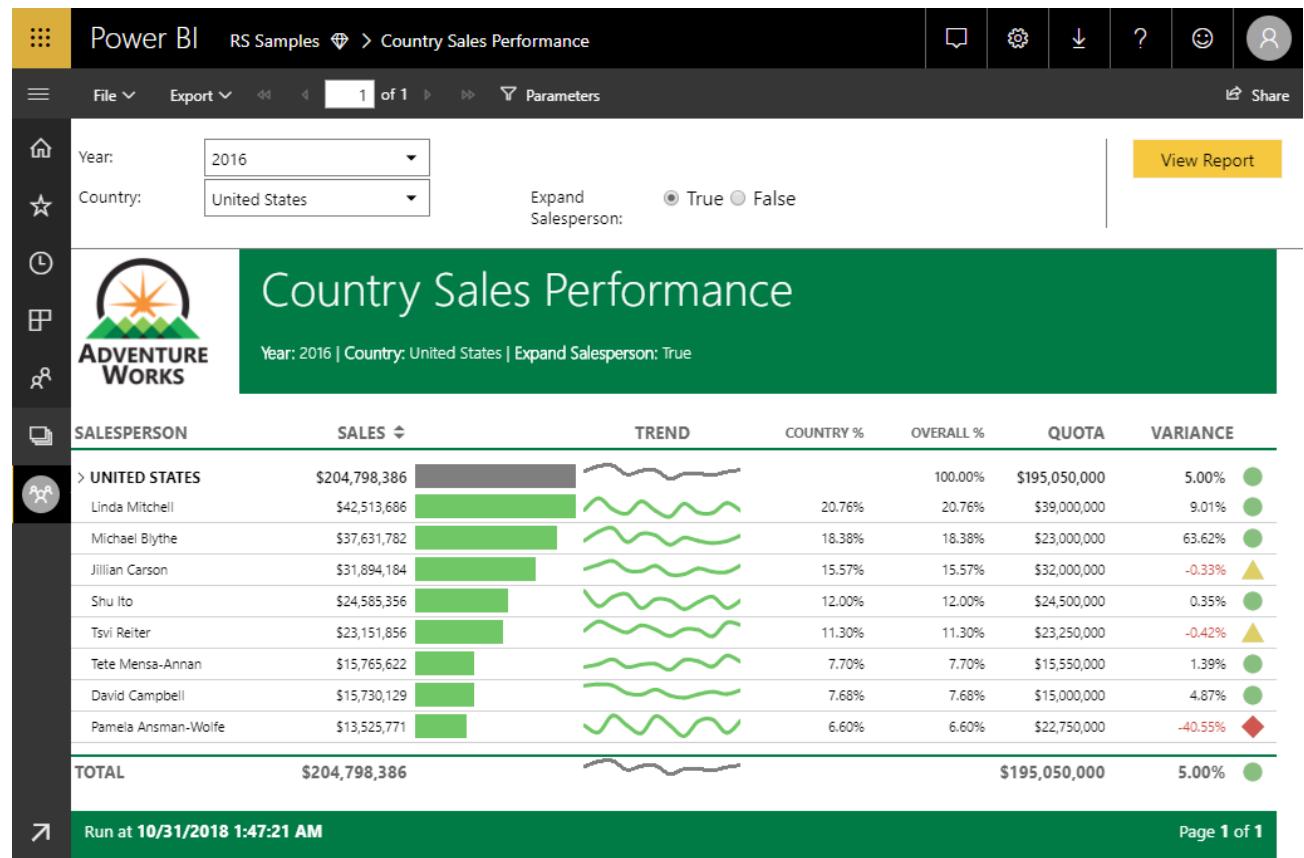


The screenshot shows an Excel spreadsheet titled 'C2132' with a filter icon and a cell reference '2'. The data is organized into columns labeled B through H. Column B is 'Year', column C is 'Month', column D is 'Month Name', column E is 'Calendar Month', column F is 'Births', column G is 'Births Per Day', and column H is 'Births (Normalized)'. The data spans from row 2119 to 2139. The 'Month' column contains numerical values (1 through 12) which are dropdown menus. The 'Month Name' column lists the months from January to September. The 'Calendar Month' column lists the dates from 1/1/2004 to 9/1/2005. The 'Births' column shows the raw birth counts, while 'Births Per Day' and 'Births (Normalized)' show the average daily births and normalized values respectively. A cursor is visible over the cell containing '2/February'.

| C2132 | B    | C     | D          | E              | F      | G              | H                   |
|-------|------|-------|------------|----------------|--------|----------------|---------------------|
| 1     | Year | Month | Month Name | Calendar Month | Births | Births Per Day | Births (Normalized) |
| 2119  | 2004 | 1     | January    | 1/1/2004       | 2,937  | 94.7           | 2842                |
| 2120  | 2004 | 2     | February   | 2/1/2004       | 2,824  | 97.4           | 2921                |
| 2121  | 2004 | 3     | March      | 3/1/2004       | 3,128  | 100.9          | 3027                |
| 2122  | 2004 | 4     | April      | 4/1/2004       | 2,896  | 96.5           | 2896                |
| 2123  | 2004 | 5     | May        | 5/1/2004       | 3,008  | 97.0           | 2911                |
| 2124  | 2004 | 6     | June       | 6/1/2004       | 3,047  | 101.6          | 3047                |
| 2125  | 2004 | 7     | July       | 7/1/2004       | 2,981  | 96.2           | 2885                |
| 2126  | 2004 | 8     | August     | 8/1/2004       | 3,079  | 99.3           | 2980                |
| 2127  | 2004 | 9     | September  | 9/1/2004       | 3,219  | 107.3          | 3219                |
| 2128  | 2004 | 10    | October    | 10/1/2004      | 3,547  | 114.4          | 3433                |
| 2129  | 2004 | 11    | November   | 11/1/2004      | 3,365  | 112.2          | 3365                |
| 2130  | 2004 | 12    | December   | 12/1/2004      | 3,143  | 101.4          | 3042                |
| 2131  | 2005 | 1     | January    | 1/1/2005       | 2,921  | 94.2           | 2827                |
| 2132  | 2005 | 2     | February   | 2/1/2005       | 2,699  | 96.4           | 2892                |
| 2133  | 2005 | 3     | March      | 3/1/2005       | 3,024  | 97.5           | 2926                |
| 2134  | 2005 | 4     | April      | 4/1/2005       | 3,037  | 101.2          | 3037                |
| 2135  | 2005 | 5     | May        | 5/1/2005       | 3,231  | 104.2          | 3127                |
| 2136  | 2005 | 6     | June       | 6/1/2005       | 3,163  | 105.4          | 3163                |
| 2137  | 2005 | 7     | July       | 7/1/2005       | 3,119  | 100.6          | 3018                |
| 2138  | 2005 | 8     | August     | 8/1/2005       | 3,156  | 101.8          | 3054                |
| 2139  | 2005 | 9     | September  | 9/1/2005       | 3,439  | 114.6          | 3439                |

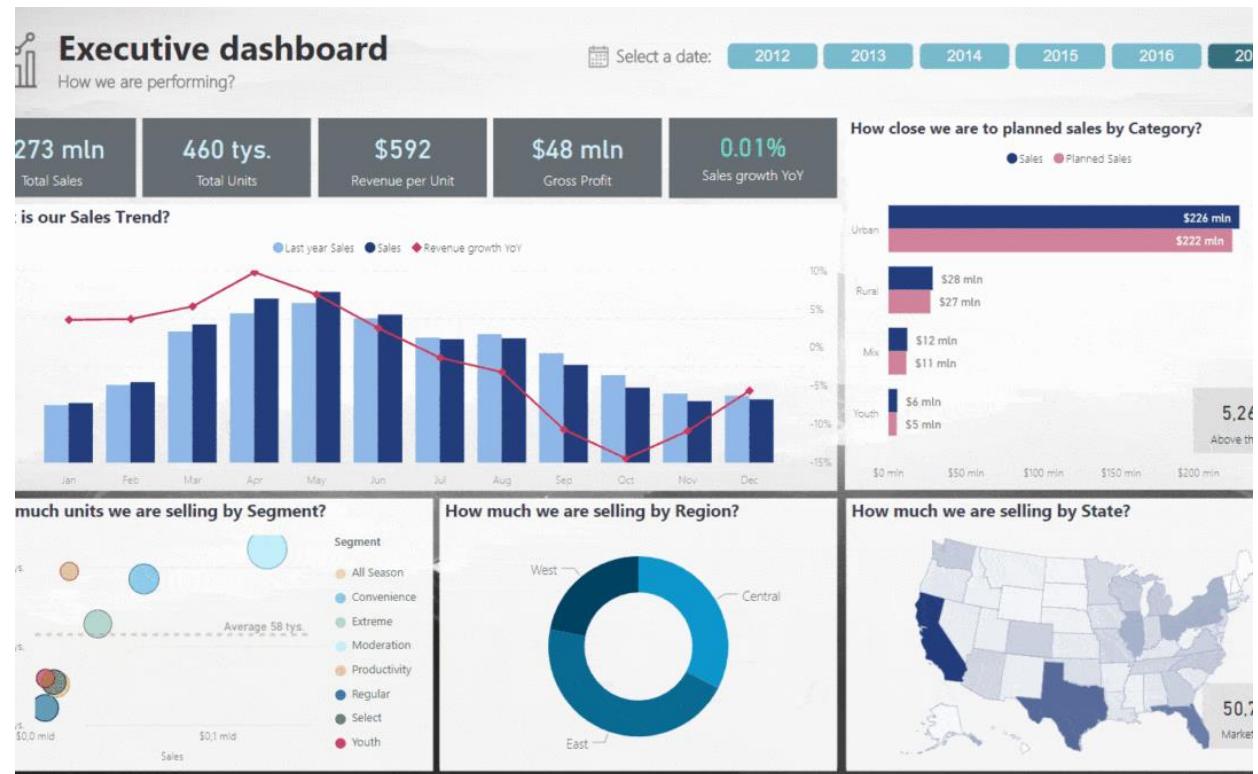
# Building Blocks - Reports

- Reports allow multiple visualizations to be organized together into logical reports
  - The reports can span multiple pages which can then be accessed using tabs
- Reports can be interactive where views can set parameters for the report



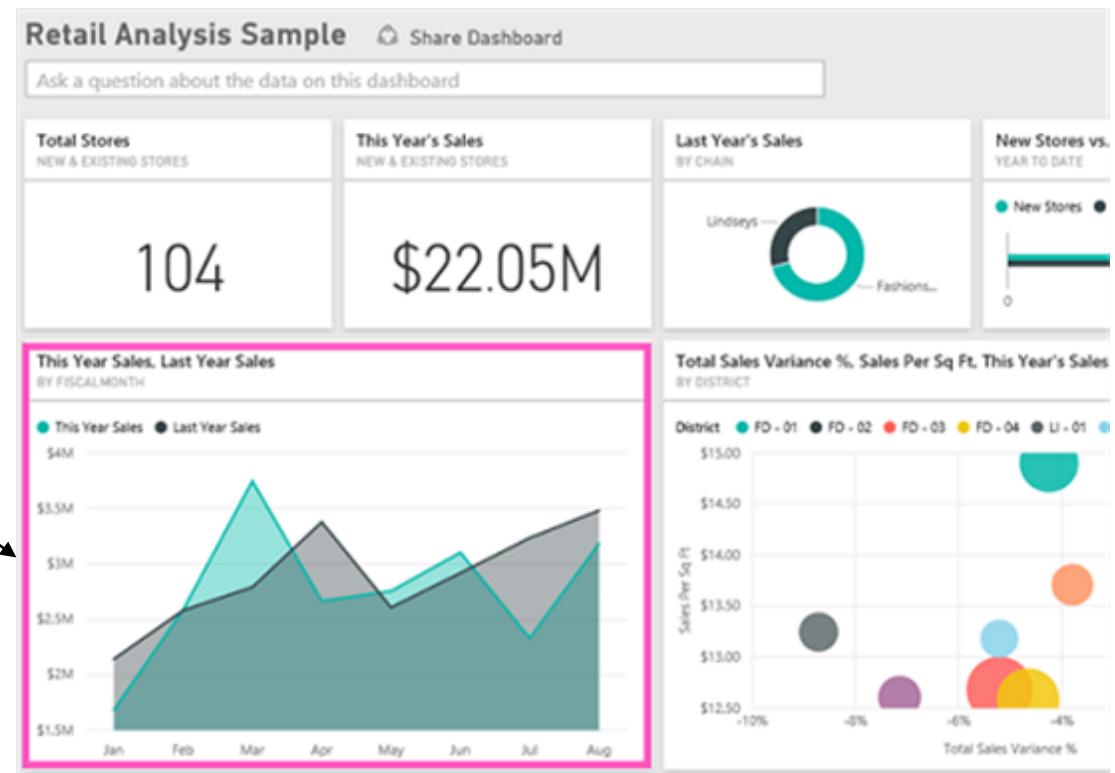
# Building Blocks - Dashboards

- Dashboards can be thought of as high-level pages from where clients can drill further down into various reports
- In general, Reports provide more detailed information while Dashboards are used to provide the key pieces of information



# Building Blocks - Tiles

- Tiles are the charts and graphs that provide a single piece of information
- Reports and Dashboards generally consist of putting together multiple Tiles to gather in a logical manner to produce a final detailed Report on a particular subject



# Power BI and Tableau Comparison

- What is difference between Power BI and Tableau

| Power BI                                                                                                                                                      | Tableau                                                                                               |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| Power BI works best with limited size data. Working with bigger sized data will require steps to reorganize and summarize the data into more manageable sizes | Tableau can handle huge volumes of data with great performance                                        |
| Power BI is very easy to get started with                                                                                                                     | Tableau has a steeper learning curve                                                                  |
| Power BI is used by experienced and novice users alike, often for quick and easy insights                                                                     | Tableau is used by analysts and experienced users for the purpose of visualizing their analytics work |
| Power BI offers numerous data point for visualization                                                                                                         | Tableau offers many data visualization functionalities                                                |
| Power BI has lesser connectors to data sources                                                                                                                | Tableau has a numerous connectors to many different types of data sources                             |
| Power BI is best for medium to smaller companies. Although the Power BI Premium is well suited for larger organization with greater needs.                    | Tableau is best suited for large and medium enterprises                                               |
| Limited customer support                                                                                                                                      | Extensive customer support                                                                            |

## [Lab3] Working with Power BI



Chapter 9.

# **Security and Access Control**

Big Data Course

# Chapter Description

---

## Objectives:

- ✓ Security is an important area for IT. Because Hadoop cluster also stores and processes a lot of data, important data must be kept safely. The main concepts and methods for Hadoop security are explained.
  - We will first learn Hadoop cluster security concept and why security is important in Hadoop.
  - Understand the concepts of HDFS permission, ACL, ranger, etc. that control illegal access to data, and understand the protocol for data storage encryption and transmission security.
  - We'll learn the basics of security including the concepts of authentication, authorization, and encryption.
  - Kerberos supports secure usage by integrating with multiple ecosystems in Hadoop with strong authentication methods. Understand the basic components of Kerberos and Major Terminology
  - Finally, how Kerberos works and how the client and Hadoop service are authenticated will be explained.

## Contents of Chapter:

1. Secure Access to Cluster Service
2. Secure Access to Cluster Data

Unit 1.

# Secure Access to Cluster Services

Security and Access Control

Unit 1.

# Secure Access to Cluster Services

# Hadoop Cluster Security

- Hadoop cluster security is mostly concerned with limiting who can access protected data, by limiting who can submit jobs on the cluster
- Three main factors



Authenticatio  
n



Authorizatio  
n



Encryptio  
n

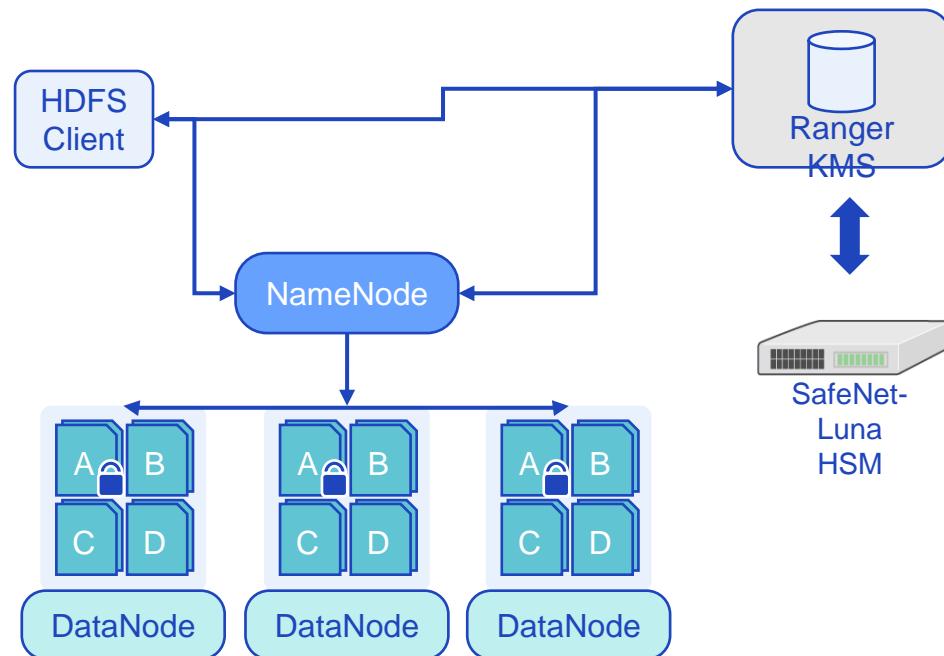
- Kerberos is used for authentication, and security services such as ranger are used for authorization.

# Quick Basic overview

- Perimeter
  - Strong User authentication
  - Network isolation, edge nodes
  - Firewalls
- Access
  - Authorization controls
  - Granular access to HDFS files, Hive/Impala objects with Ranger
- Data
  - Encryption-at-rest
  - Encrypted HTTP traffic (Transport Layer Security -TLS)

# Transparent Data Encryption in HDFS

- Selective encryption of relevant files/folders
- Prevent rogue admin access to sensitive data
- Fine grained access controls
- Transparent to end application w/o changes
- Ranger KMS integrated to external HSM



# Types of Hadoop Authentication

- These Items require strong authentication in Hadoop
  - Hadoop user
  - Hadoop service
  - HTTP-based console
  - Data protection on the go
- Kerberos enhances security
- Optional-Provides strong authentication for both client and server
- Wrap Hadoop API calls in a SASL handshake
- Application can be executed with the submitter's own account

# HDFS permissions

- HDFS permissions are primarily POSIX
  - Remember that hdfs is the HDFS superuser, not the root
  - The directory execution bit is the sticky bit
- POSIX style ACLs are supported
  - However, it is disabled by default (`dfs.namenode.acls.enabled`)
  - You can add users, groups, and other permissions and apply default masks
  - ACLs are best used to adjust file permissions
- Fine-grained Authorization
  - HDFS permissions & ACLs
  - File permissions for user-group-others may be too simple

# Strong Authentication

- Hadoop can use Kerberos to provide stronger authentication for security
  - Hadoop daemons can use this to authenticate all RPCs
- Kerberos
  - Linux supports MIT Kerberos natively
- Kerberos is the only authentication model Hadoop supports
  - In-transit encryption service hooks are available
  - Browser authentication supported by HTTP SPNEGO
- LDAP/Active Directory integration is supported
  - Applying existing user databases to Hadoop cluster is a common ask

# Kerberos Terminology

- **Realm**

- A network that uses Kerberos, composed of one or more servers called KDCs and a potentially large number of clients.

- **Ticket**

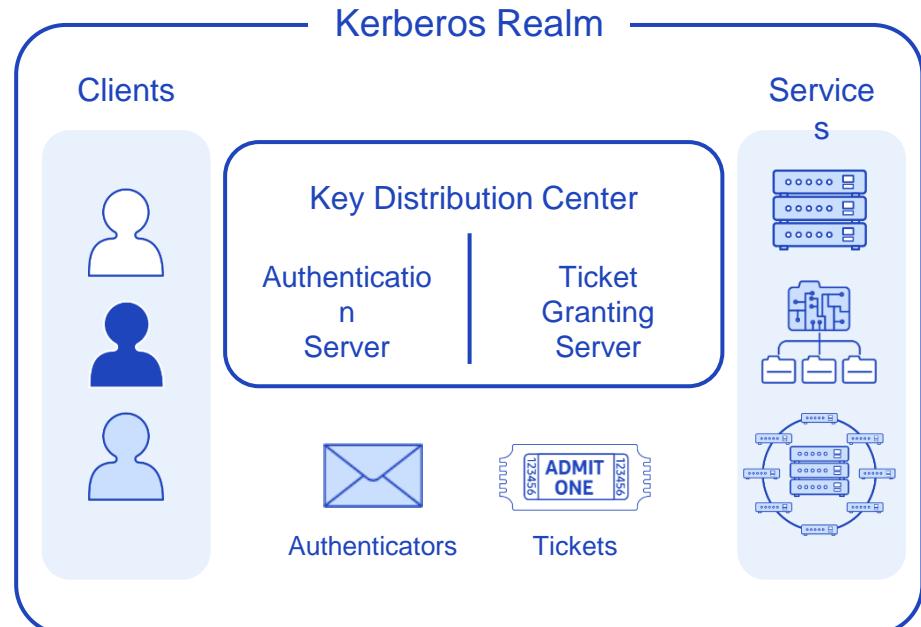
- A temporary set of electronic credentials that verify the identity of a client for a particular service. Also called credentials

- **Principal (or principal name)**

- The principal is the unique name of a user or service allowed to authenticate using Kerberos.

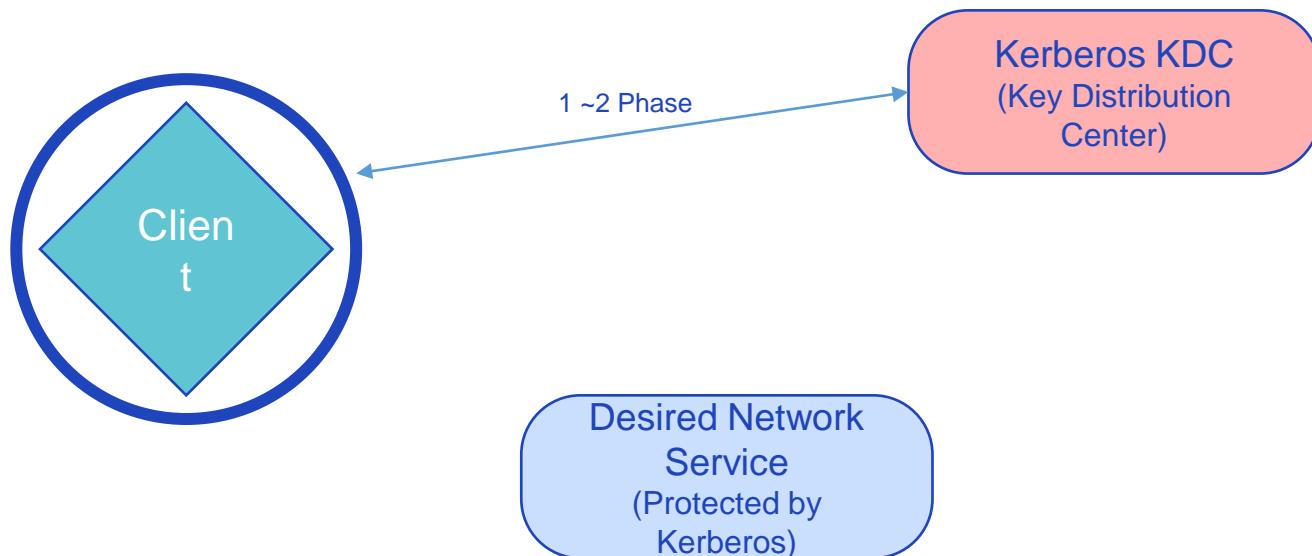
- **keytab (or key table)**

- A file that includes an unencrypted list of principals and their keys



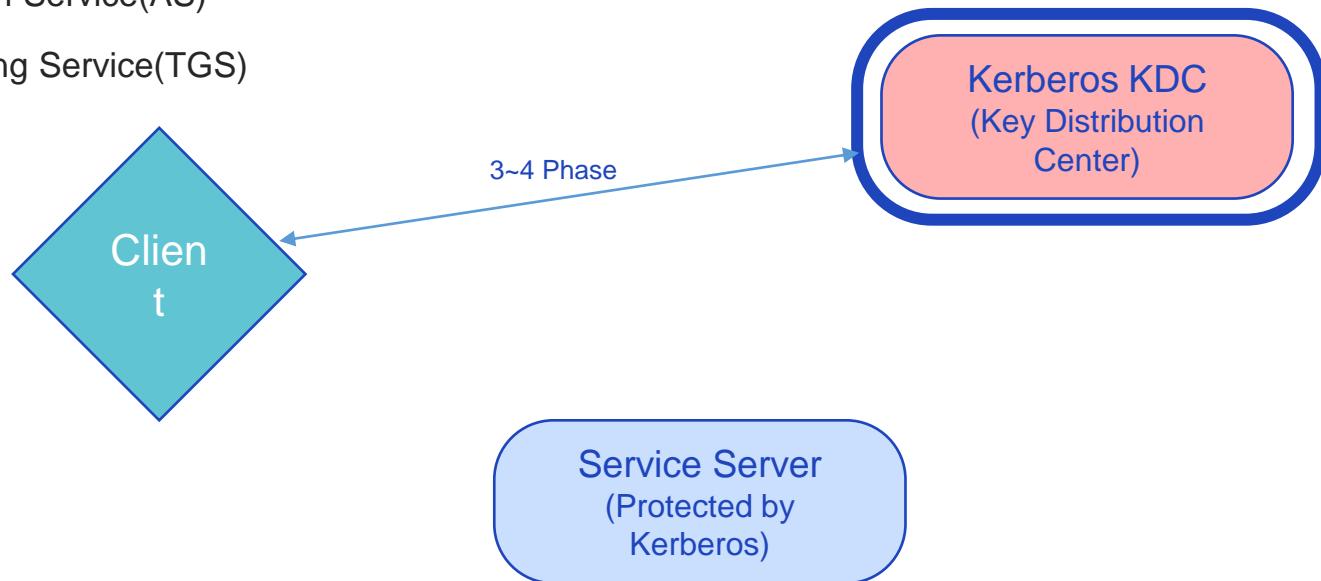
# Key components in Kerberos (1/3)

- Kerberos has three parts: a client, server, and trusted third party (KDC) to mediate between them
- The client is software that desires access to a Hadoop service
  - The beeline command is one example of a client



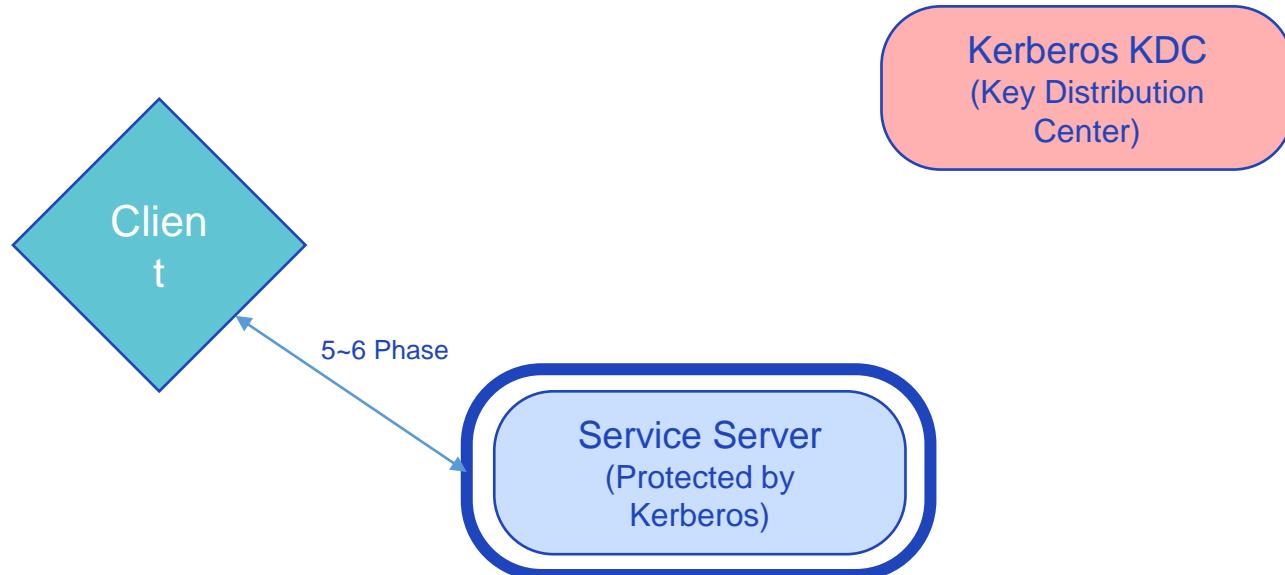
## Key components in Kerberos (2/3)

- The Kerberos server (KDC) authenticates and authorizes a client
- KDC consists of two services
  - Authentication Service(AS)
  - Ticket Granting Service(TGS)

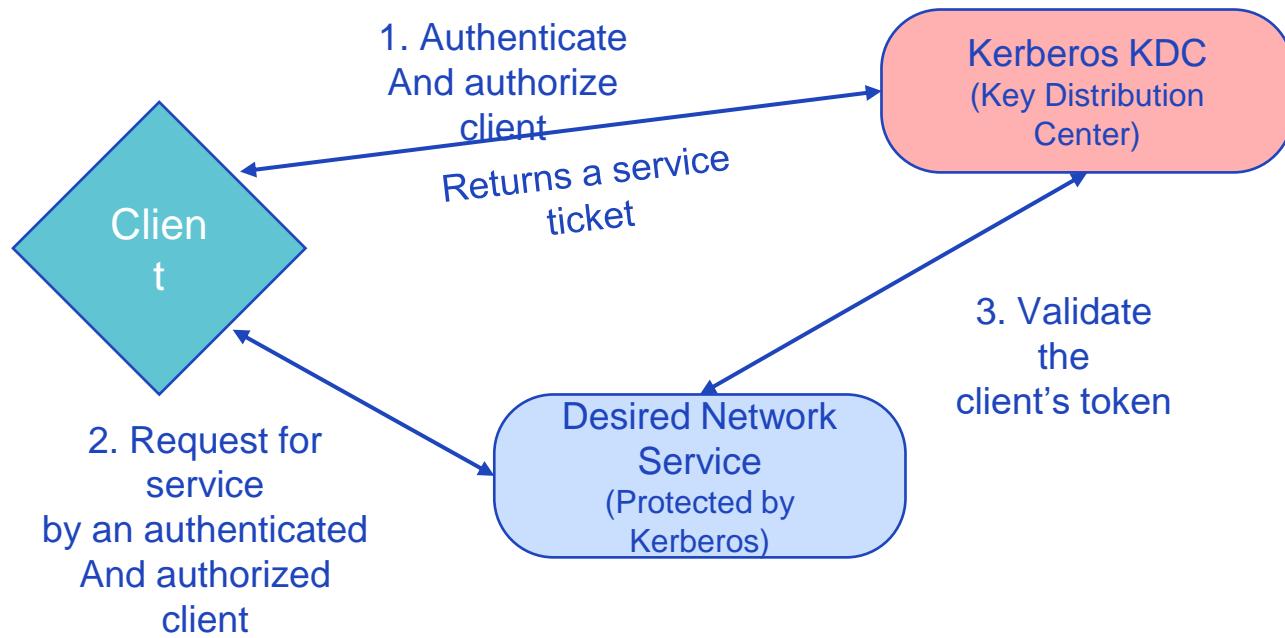


## Key components in Kerberos (3/3)

- This is the Hadoop service that the client wants to access
  - This is service daemon such as Hiveserver 2

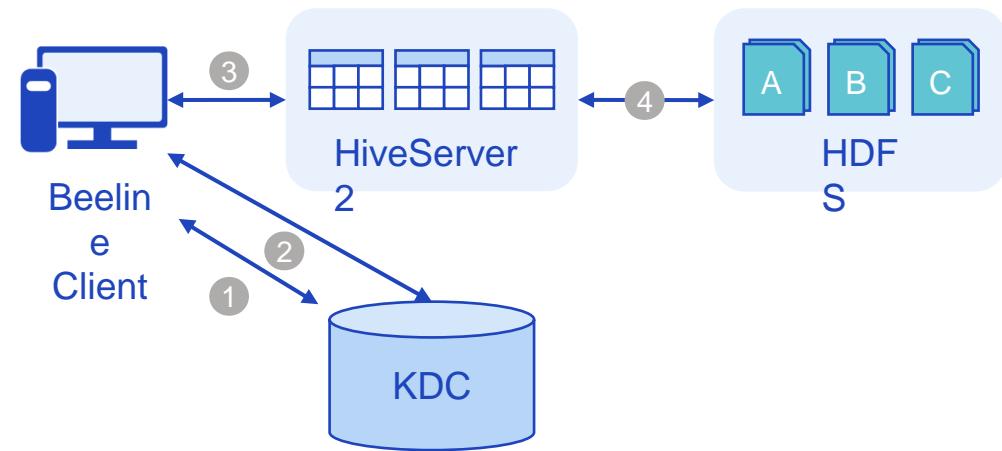


# How Kerberos works



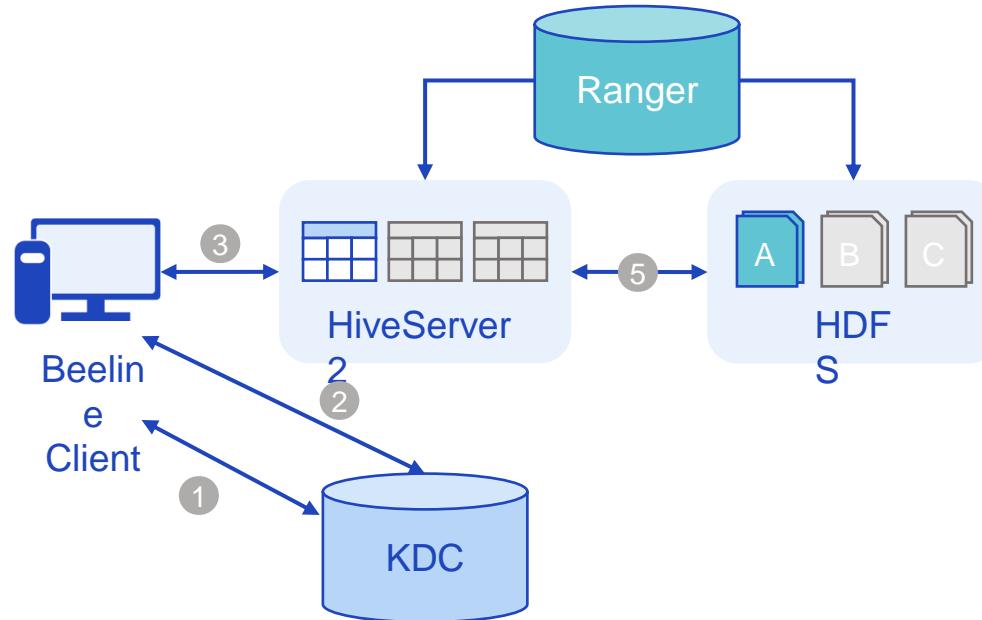
# Hive use through Beeline Kerberos

- Client
  - Requests a TGT(Ticket-granting tickets)
  - Receives TGT
  - Client decrypts it with the password hash
  - Send the TGT and receives a service Ticket
  - Submit a query
- KDC
  - Issues service tickets for Client based on Ticket-granting tickets issued by authentication server
  - Sends a Service Ticket
- HiveServer2
  - Hive gets NN service ticket and create MapReduce using NN service Ticket



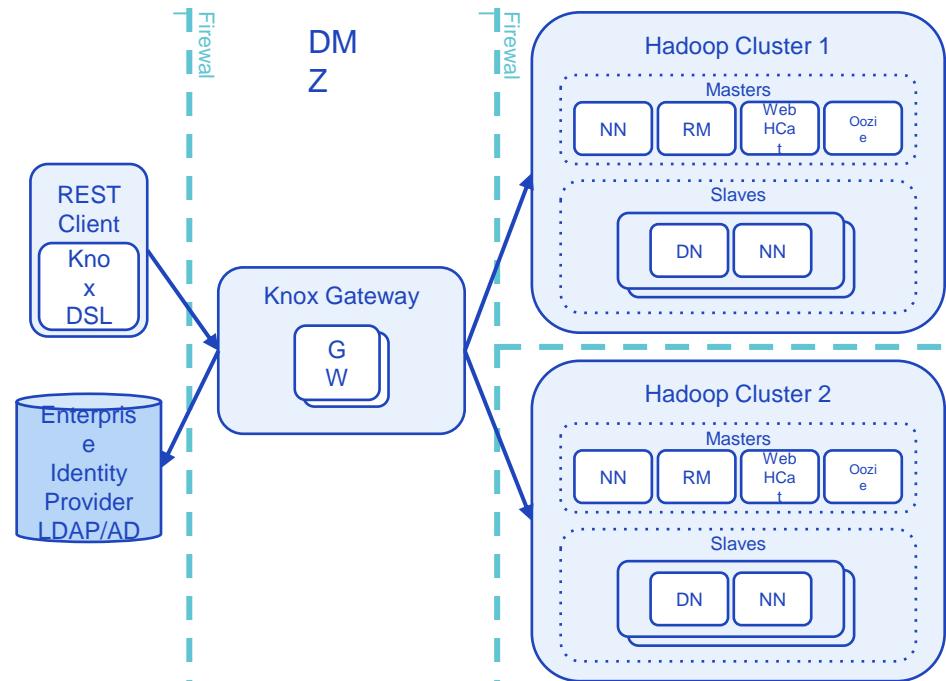
# Add authorization with Ranger

- Apache Ranger to provide centralized security administration and management
- To create and update policies in a policy database



# Security Features

- Authentication
  - Kerberos support
  - Kerberos is an industry standard used to authenticate users and resources in Hadoop clusters
  - Perimeter security by Apache Knox
- Authorizations
  - Fine grained access control - HDFS, HBase, and Hive
  - Role base access control
  - Column level support
  - Permission support – create, drop , index, user
- Encryption
  - Ensuring only verified users can access data
  - OS filesystem, HDFS, network-level encryption support



Unit 2

# Secure Access to Cluster Data

# Apache Ranger (1/2)

- Apache Ranger is an open source application to define, administer, and manage security policies
- Central Security Administration
  - Providing a single interface to security managers
  - Centralize security policy management
  - Guarantees consistent coverage across the Hadoop stack
- Pluggable and can be easily extended to any data source using a service-based definition

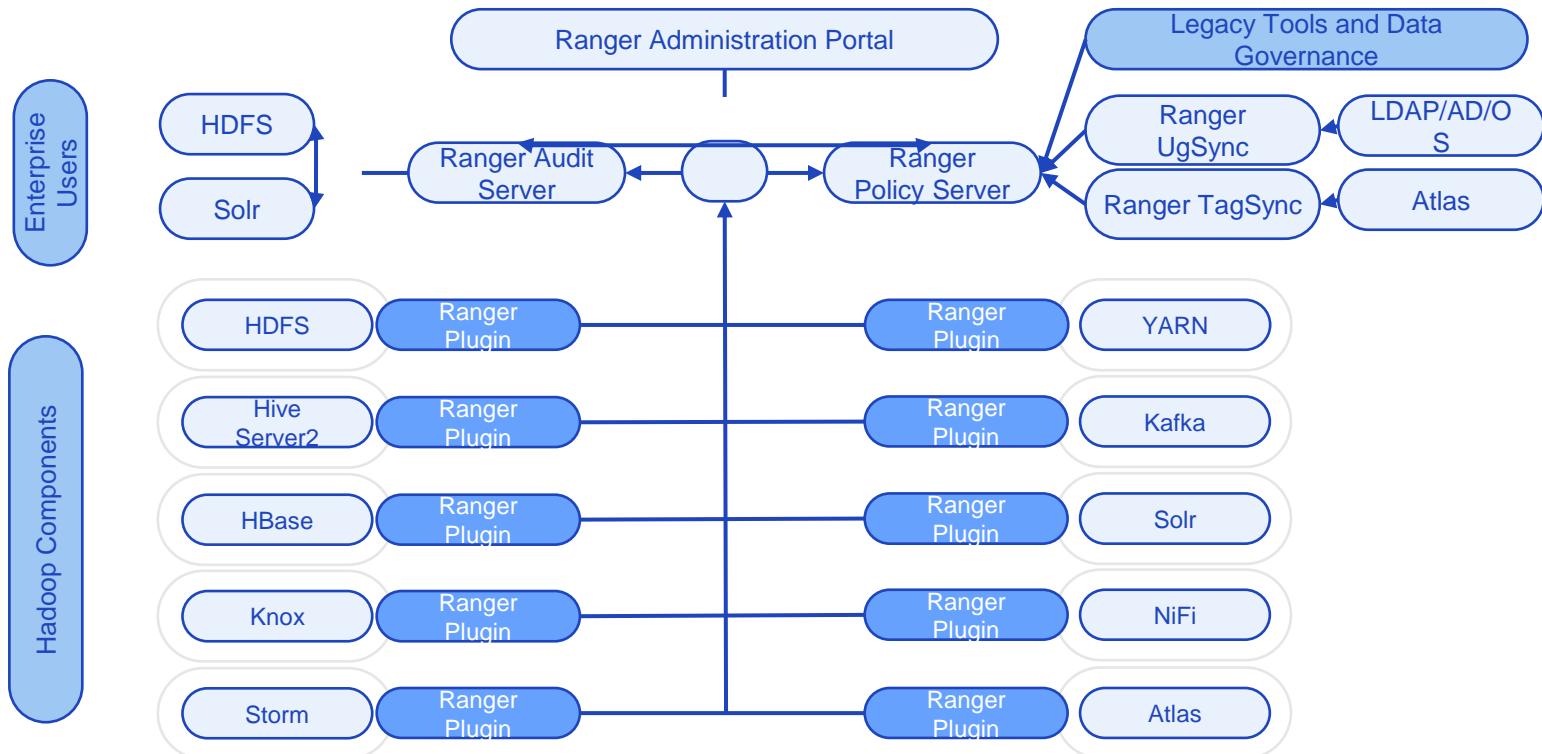


## Apache Ranger (2/2)

| Authorization                                                                                                                                                                                                                                                                                                                                                                                  | KMS                                                                                                                                                                                         | Auditing                                                                                                                                                                                                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Centralized platform to define, administer and manage security policies consistently across Hadoop components</p> <ul style="list-style-type: none"><li>• HDFS, Hive, HBase, YARN, Kafka, Solr, Storm, Knox, NiFi, Atlas</li><li>• Extensible Architecture</li><li>• Custom policy conditions, user context enrichers</li><li>• Easy to add new component types for authorization</li></ul> | <ul style="list-style-type: none"><li>• Store and manage encryption keys</li><li>• Support HDFS Transparent Data Encryption</li><li>• Integration with HSM</li><li>• Safenet LUNA</li></ul> | <ul style="list-style-type: none"><li>• Central audit location for all access requests</li><li>• Support multiple destination sources (HDFS, Solr, etc.)</li><li>• Real-time visual query interface</li></ul> |

# Apache Ranger Architecture

- Authorization and Auditing with Ranger

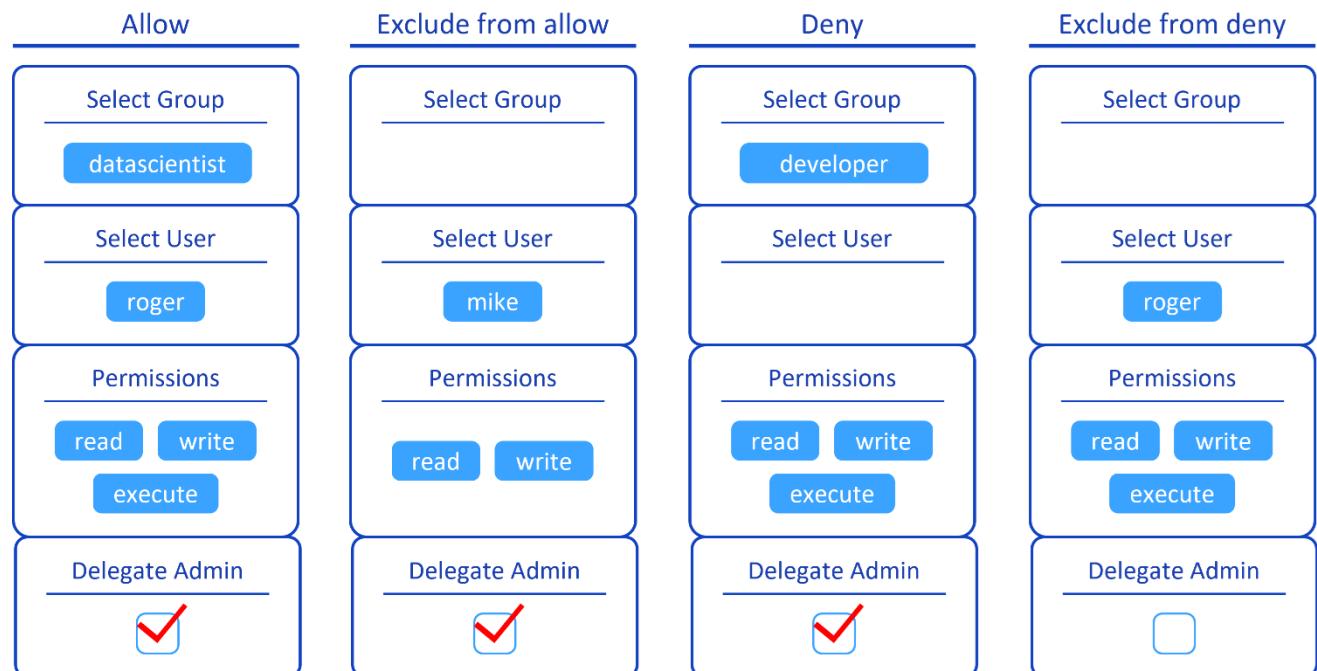


# Ranger Component

- Ranger Policy Portal and Server
  - Central interface for security management
  - Create and update policies stored in the policy database
  - Plugins within each component poll these policies on a regular basis
- Ranger plugin
  - Lightweight Java program
  - Embed in the process of each cluster component
  - The plugin pulls the policy from the central server and saves it locally in a file
- User group synchronization
  - User synchronization utility for pulling users and groups from Unix, LDAPAD
  - Saved in the Ranger portal and used for policy definition

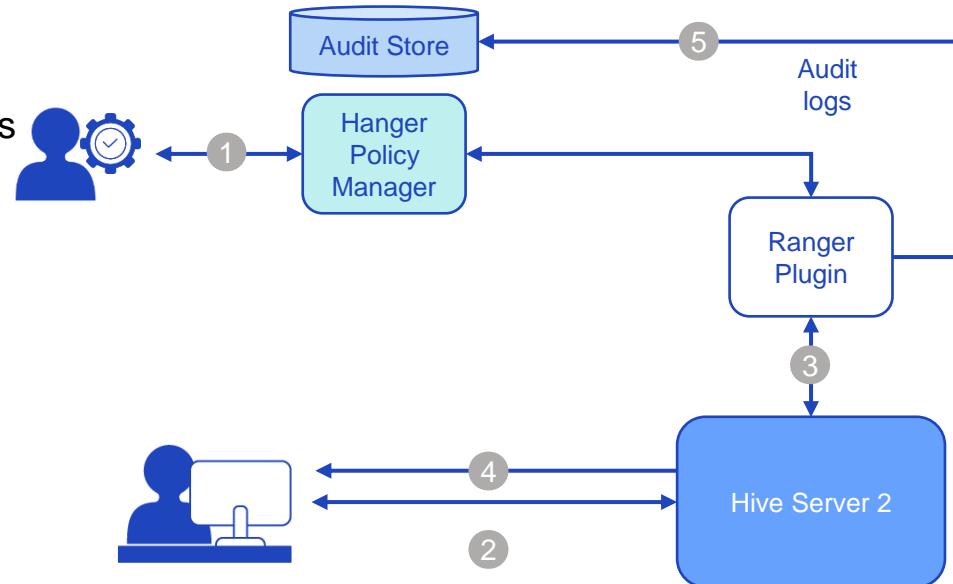
# Apache Ranger Policies

- The policy provides rules for allowing or denying access to users
  - All policies can be applied to roles, groups, or specific users rules to allow or deny
- Resource-based policies
  - Identify who can use the resource
  - Create or edit using the service plugin
- Tag-based policy
  - Restrict access using classifications and other attributes



# Ranger workflow

- Step1: The admin sets policies for Hive DB, Tables, Columns, views
- Step2: Users access Hive on beeline command tool
- Step3: Hive authorizes with Ranger plugin
- Step4: HiveServer2 provides data access to users
- Step5: Leave audit logs



# Ranger Admin Portal

- Central interface for security administration
- Admin users can
  - Define repositories
  - Create and update policies
  - Manage Ranger users/groups
  - Define audit policies
  - View audit activities

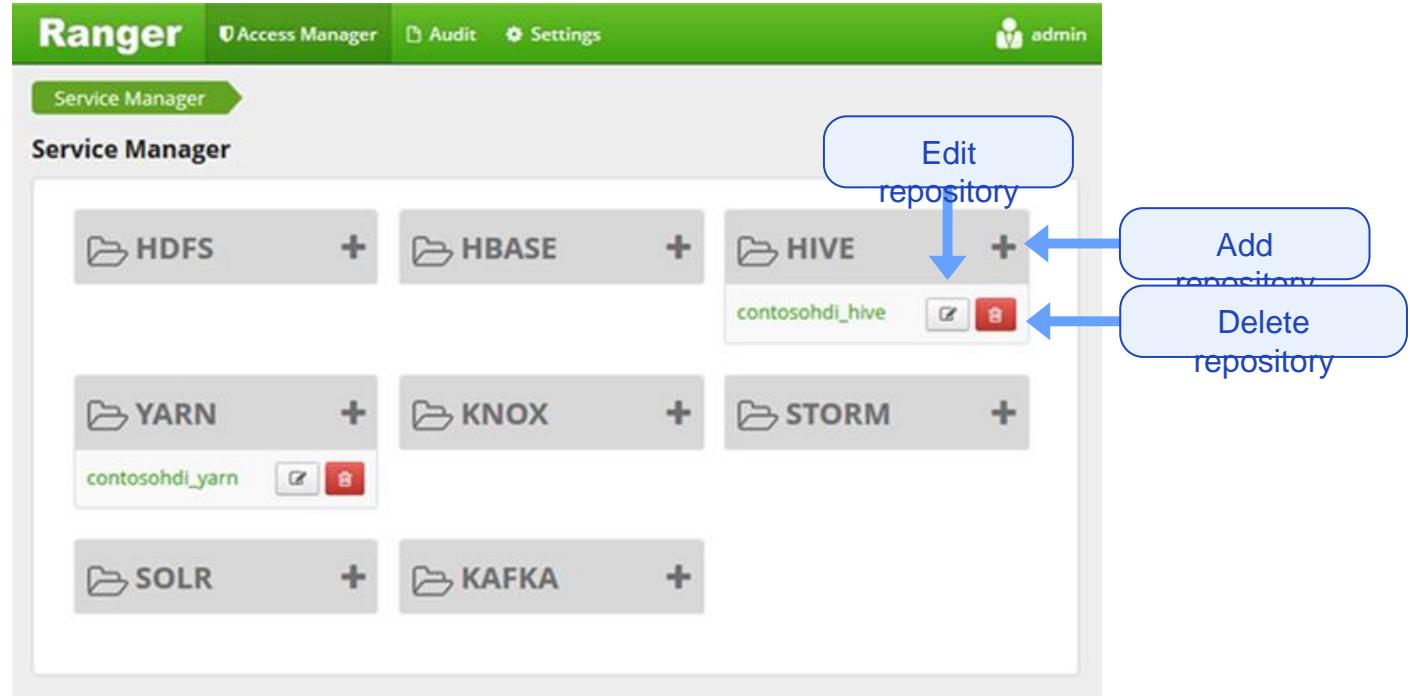
The screenshot shows the Ranger Admin Portal interface. At the top, there is a navigation bar with links for Ranger, Access Manager, Audit, Security Zone, Settings, and a user icon labeled 'admin'. Below the navigation bar, the page title is 'Service Manager > Hadoop SQL Policies > Edit Policy'. The main content area is titled 'Policy Details :'. It shows a policy named 'access: ww\_customers' with a Policy ID of 78. The policy is set to 'enabled' and 'normal'. The 'Policy Type' is 'Access'. Under 'Policy Conditions', it says 'No Conditions'. The policy details include sections for 'database' (set to 'include'), 'table' (set to 'include'), and 'column' (set to 'include'). There is also a 'Description' field and an 'Audit Logging' toggle switch set to 'YES'. At the bottom, there is a section titled 'Allow Conditions :'. The browser status bar at the bottom left indicates 'Ranger - Mozilla Firefox'.

# Apache Ranger Plugins

- Ranger plugins
  - HDFS
  - HIVE
  - STORM
  - HBase
- Steps to enable plugins
  - Start the policy manager
  - Create the plugin repository in the Policy manager
  - Install the plugin
  - Restart the plugin service



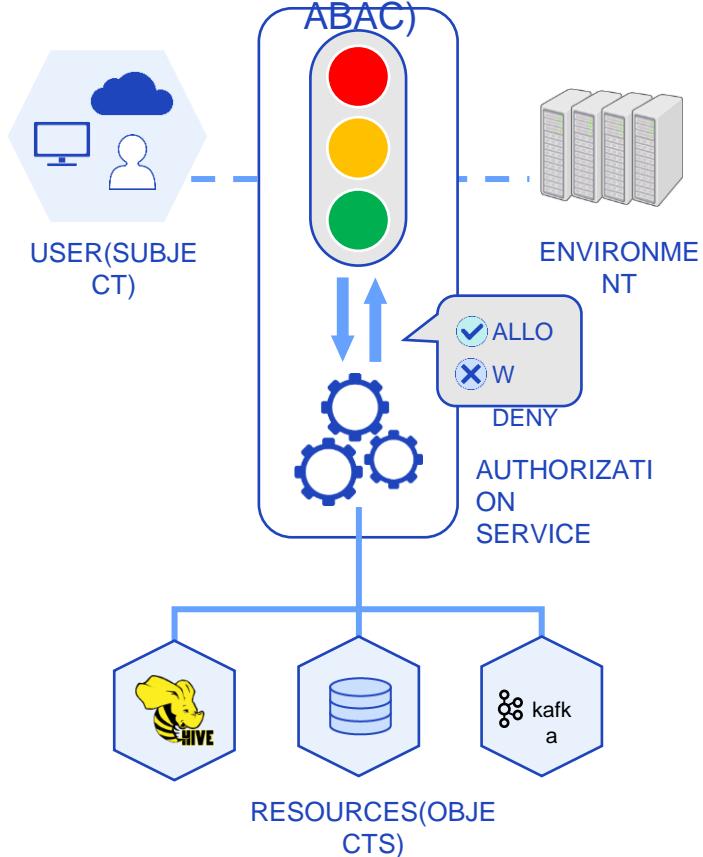
# Repository Manager



# Ranger – ABAC Model

- Combination of the subject, action, resource, and environment
- Uses descriptive attributes: AD group, Apache Atlas-based tags or classifications, geo-location, etc.
- Ranger approach is consistent with NIST 800-162
- Avoid role proliferation and manageability issues

ATTRIBUTE BASED ACCESS CONTROL(ABAC)



# What is Apache Atlas?

- Apache Atlas is a platform for managing data standards and lineage.
- Data Governance and Metadata framework for Hadoop
- Dynamic Tag-based Security Policies
- Capabilities
  - Metadata Catalog & Search
  - Lineage & Chain of Custody
  - Managed/searched by attaching a Tag for table/columns
  - Metadata Audits & Security



Apache **Atlas**

# Why use Apache Atlas? (1/2)

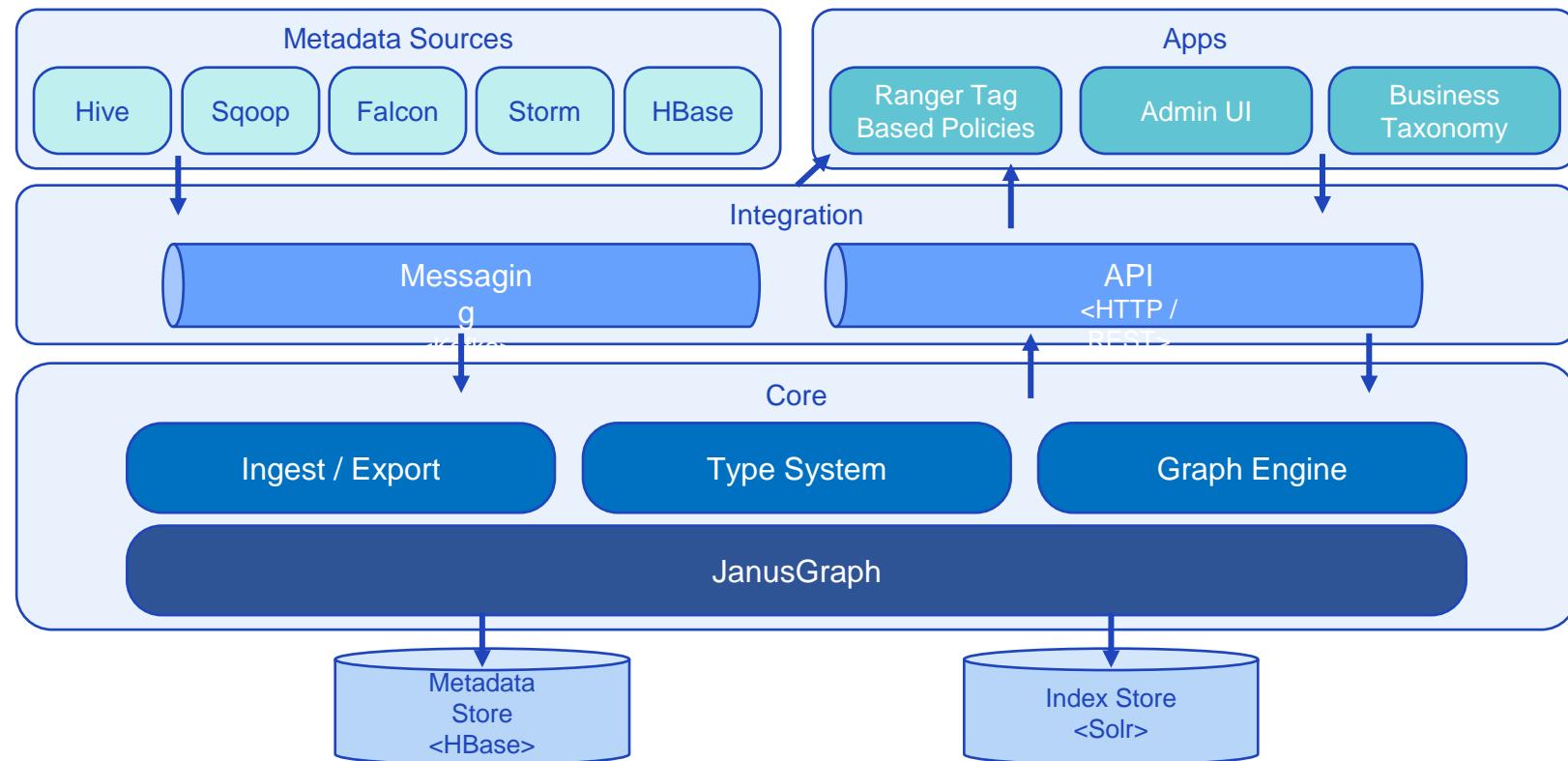
- Metadata types and interfaces
  - Various Hadoop and Non-Hadoop metadata available as predefined types.
  - New types of metadata can be defined and managed.
  - A type can have simple properties, complex properties, or reference objects.
  - Instances of types called entities collect the details and associations of metadata objects.
- Classification
  - Ability to dynamically create classifications - personal information (PII), expiration information (EXPIRES\_ON), data quality (DATA\_QUALITY), sensitive information (SENSITIVE)
  - Classification includes properties - the expiry period (expiry date) attribute of the expiry information (EXPIRES\_ON) classification
  - Entities are related to various classifications, and it is easy to search and enhance security.
- Lineage
  - Intuitive UI allows you to see how data is moved and processed as a genealogy
  - Access and update genealogy with REST APIs

# Why use Apache Atlas? (2/2)

- Search and Discovery
  - Intuitive UI allows you to search for objects by type, classification, attribute value or free text
  - Rich REST APIs enable even complex searches
  - Searching for objects with something like SQL-like Query language - Domain Specific Language (DSL)
- Security and Data Masking
  - Fine-grained security of metadata access, enabling access and control of object instances, and classification operations such as addition/modification/removal
  - In combination with Apache Ranger, authorization processing/data masking is provided based on data access and classification is linked with objects within Apache Atlas.
  - For example, users who can access data classified as personal information (PII) and sensitive information (SENSITIVE)
- Meta Service supported by Atlas
  - Hive, HBase, Ranger, Sqoop, Storm/Kafka, etc.

# Apache Atlas Architecture

- Overall Workflow in Apache Atlas



# Atlas Component (1/2)

- **Core**

- Type system: Atlas allows users to define models for managed metadata objects
- Graph Engine: Provides great flexibility and allows you to efficiently handle rich relationships between metadata objects
- Import / Export: The ingest component allows you to add metadata to Atlas. The export component also exposes the metadata changes detected by Atlas and raises them as an event

- **Integration**

- API: All functionality of Atlas is exposed to end users via a REST API that allows types and entities to be created, updated and deleted
- Messaging: users can choose to integrate with Atlas using a messaging interface that is based on Kafka

# Atlas Component (2/2)

- Metadata sources
  - Atlas supports integration with many sources of metadata out of the box
  - Atlas supports ingesting and managing metadata from the following sources: Hbase, Hive, Sqoop, Storm, Kafka
- Applications
  - Atlas Admin UI: a web based application that allows data stewards and scientists to discover and annotate metadata.
  - Tag Based Policies: an advanced security management solution for the Hadoop ecosystem having wide integration with a variety of Hadoop components