



WQD7007 Big Data Management

Big Data Pipeline Using Hadoop

Agenda

- What is Hadoop?
- Big Data Pipeline using Hadoop

What is Hadoop?



- Apache Hadoop is an **open source framework** for **distributed storage** and **processing** of **large sets** of data on **commodity** hardware.
 - Hadoop enables businesses to quickly gain insight from **massive amounts of structured and unstructured data**.
 - Numerous Apache Software Foundation projects make up the services required by an enterprise to deploy, integrate and work with Hadoop.

What is Hadoop?



- The base Apache Hadoop framework are composed of following modules:
 - **Hadoop Common:** The common utilities that support the other Hadoop modules.
 - **Hadoop Distributed File System (HDFS):** A distributed file system that provides high-throughput access to application data.
 - **Hadoop YARN:** A framework for job scheduling and cluster resource management.
 - **Hadoop MapReduce:** A YARN-based system for parallel processing of large data sets.

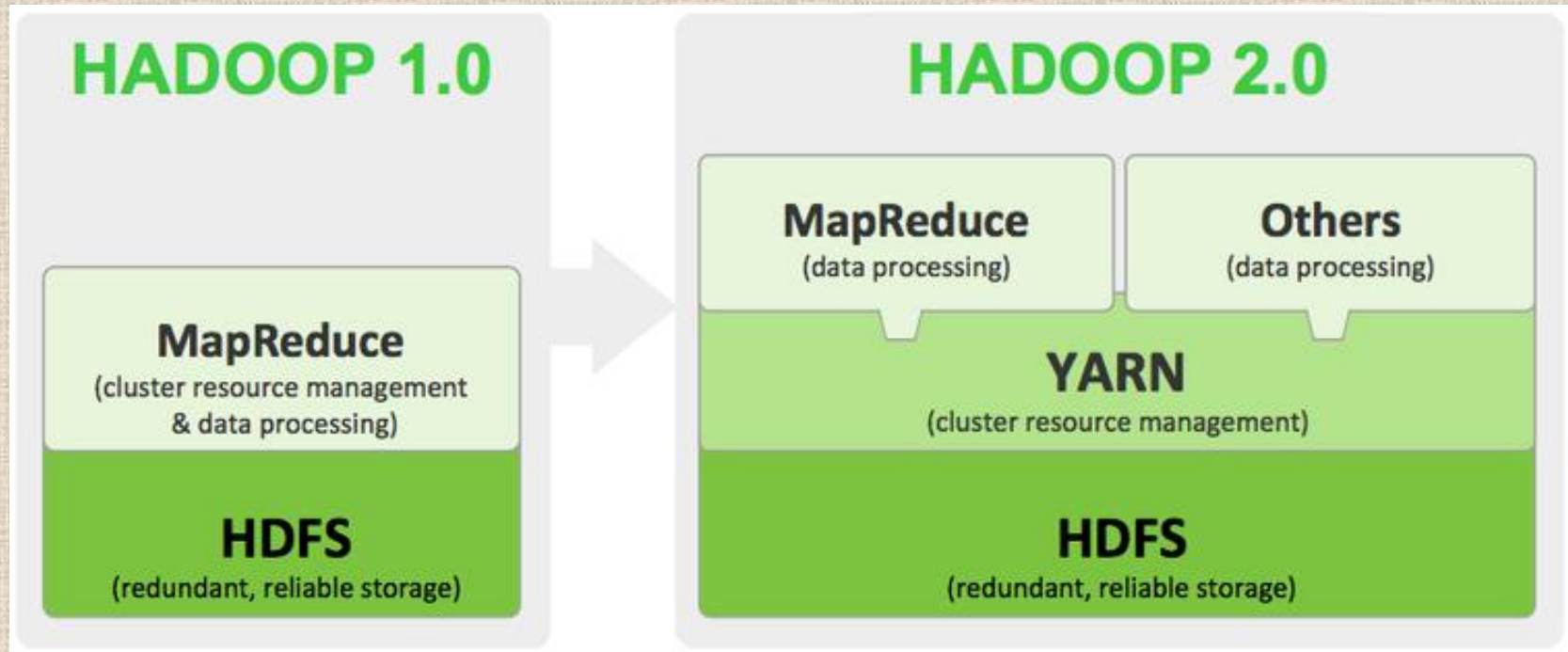
Hadoop 1.0

- Hadoop 1.0 is based on the Hadoop .20.205 branch (it went 0.18 -> 0.19 -> 0.20 -> 0.20.2 -> 0.20.205 -> 1.0).
- Not hard for an open source developer, but obscure for an enterprise product – so everyone agreed to call 0.20.205 ‘1.0’, the project having matured to that point.

Hadoop 2.0

- Hadoop 2.0 is from the Hadoop 0.23 branch, with **major components re-written to enable support for features like High Availability, and MapReduce 2.0 (YARN)**, and to enable Hadoop to scale out past 4,000 machines per cluster.
- Specifically, Hadoop 2.0 adds:
 - HDFS Federation – multiple, redundant namenodes acting in congress
 - MapReduce NextGen aka YARN aka MRv2 – which transforms Hadoop into a full blown platform as a service.

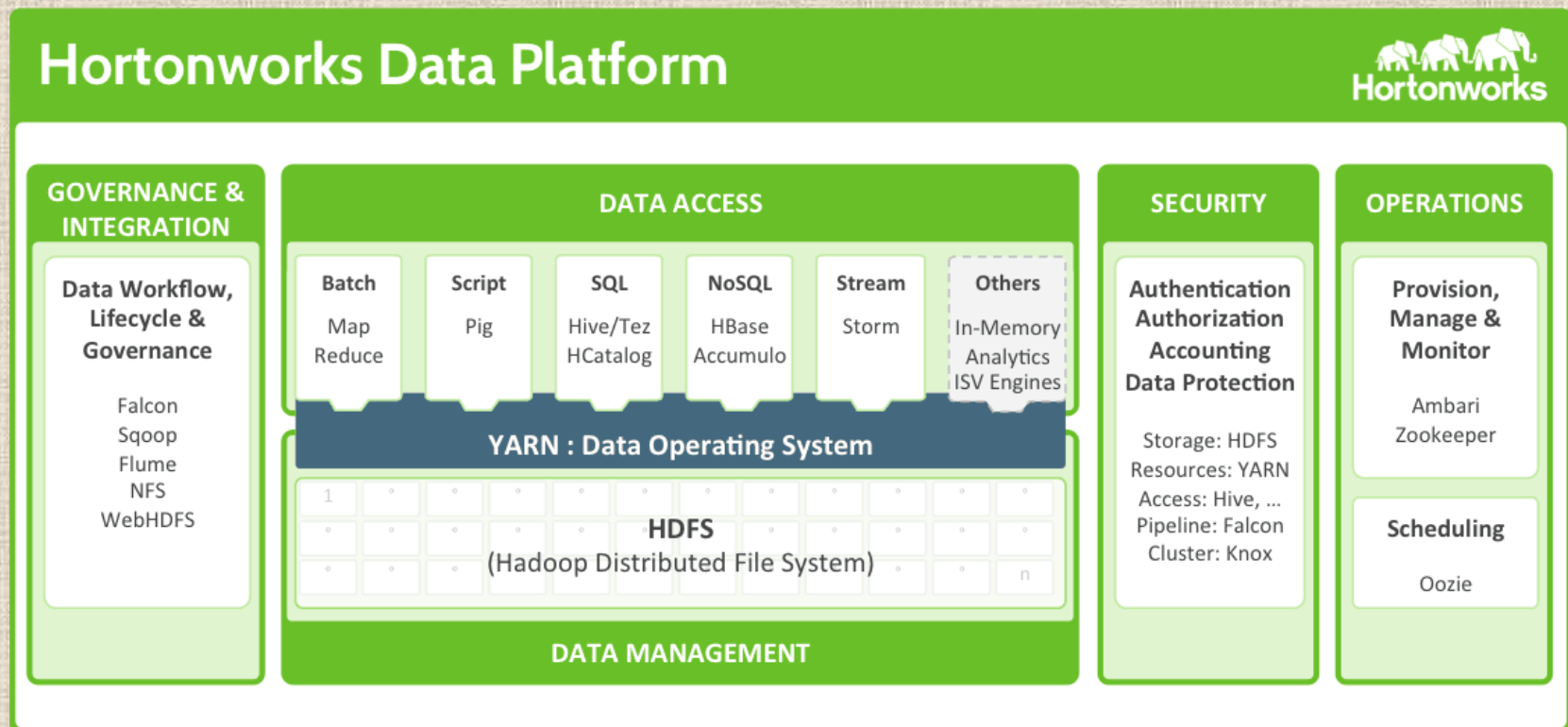
Hadoop 1.0 to Hadoop 2.0



Five pillars to Hadoop

- Data Management
- Data Access
- Data Governance and Integration
- Security
- Operations

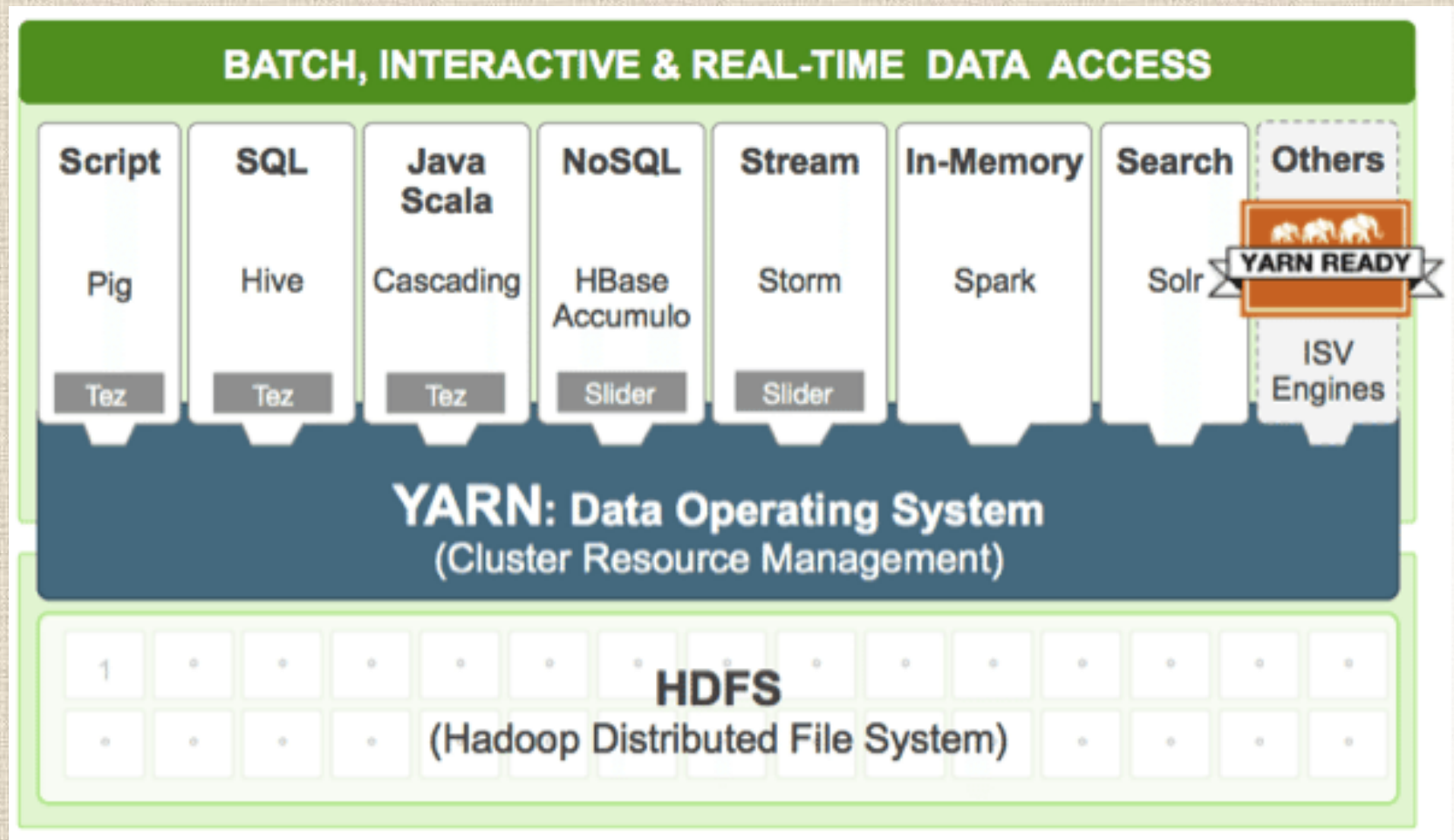
Five pillars to Hadoop



Data Management - YARN

- **Apache Hadoop YARN (Yet Another Resource Negotiator)** – Part of the core Hadoop project, YARN is a next-generation framework for Hadoop data processing extending MapReduce capabilities by **supporting non-MapReduce workloads associated with other programming models**.
 - allows **multiple data processing engines** such as interactive SQL, real-time streaming, data science and batch processing **to handle data stored in a single platform**, unlocking an entirely new approach to analytics.
 - **is the pre-requisite for Enterprise Hadoop** as it provides the **resource management** and pluggable architecture for enabling a wide variety of data access methods to operate on data stored in Hadoop with predictable performance and service levels.

YARN - The Architectural Center of Enterprise Hadoop



MapReduce and YARN: background

- A computer cluster is a set of loosely or tightly connected computers that work together so that, in many respects, they can be viewed as a single system. Therefore, cluster computing can be viewed as the techniques used to connect multiple computers together as a single system.
- Cluster computing faces several challenges:
 - how to store data persistently and keep it available if nodes fail
 - how to deal with node failures during a long running computation.
 - there is network bottleneck which delays the time of processing data.
- MapReduce offers a solution by bring computation close to data → minimizing data movement.
- It is a simple programming model designed to process large volumes of data in parallel by dividing the job into a set of independent tasks.

MapReduce and YARN: background

- The biggest limitation with MapReduce programming is that map and reduce jobs are **not stateless**.
 - This means that **Reduce jobs have to wait for Map jobs to be completed first.**
 - This limits maximum parallelism
- Therefore YARN was born as a **generic resource management and distributed application framework.**

MapReduce

- A MapReduce job splits a large data set into **independent chunks** and organizes them into **key, value pairs** for parallel processing.
 - This parallel processing **improves the speed and reliability** of the cluster.
- The **Map** function divides the input into **ranges** by the **InputFormat** and creates a **map task** for each range in the input. The **JobTracker** distributes those tasks to the **worker nodes**. The **output** of each map task is partitioned into a group of **key-value pairs for each reduce**.
 - `map(key1,value) → list<key2,value2>`

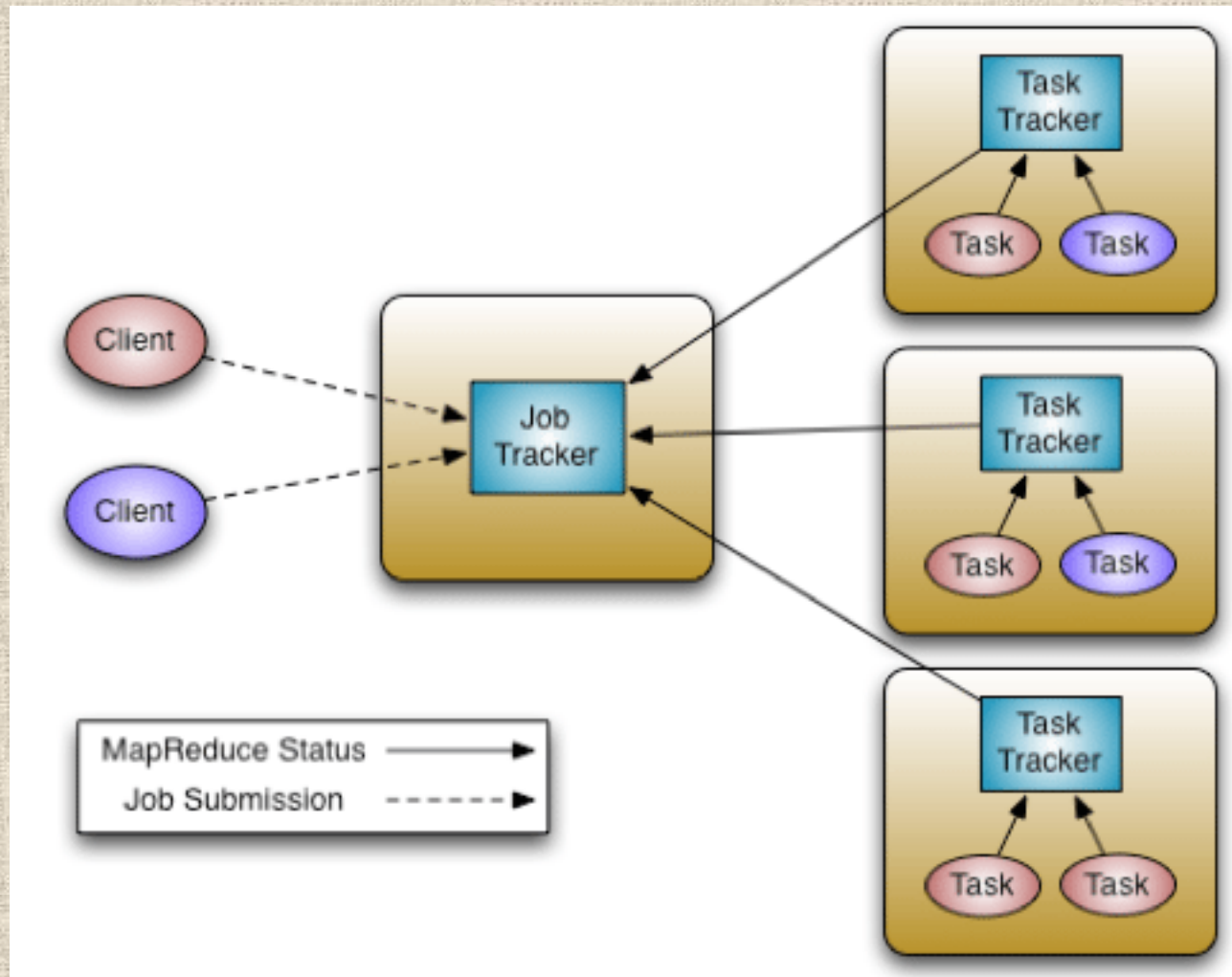
MapReduce

- The **Reduce** function then **collects the various results** and **combines** them to answer the **larger problem** that the master node needs to solve.
- Each reduce pulls the **relevant partition** from the machines where the maps executed, then writes its output back into HDFS.
 - Thus, the reduce is able to **collect the data from all of the maps** for the keys and **combine** them to solve the problem.
 - `reduce(key2, list<value2>) -> list<value3>`
- Example?


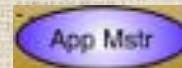
MapReduce in Hadoop

- The current Apache Hadoop MapReduce System is composed of the **JobTracker (master)**, and **TaskTrackers (per-node slaves)**.
- The **JobTracker** is responsible for
 - resource management (managing the worker nodes i.e. TaskTrackers),
 - tracking resource consumption/availability
 - job life-cycle management (scheduling individual tasks of the job, tracking progress, providing fault-tolerance for tasks etc).
- The **TaskTracker** is responsible for
 - launch/teardown tasks on orders from the JobTracker
 - provide task-status information to the JobTracker periodically.

MapReduce in Hadoop

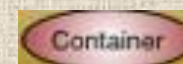


YARN – Yet Another Resource Negotiator

- To **split up** the functionalities of resource management and job scheduling/monitoring (initially by JobTracker) into separate daemons.
 - to have a global **ResourceManager** (*RM*), and 
 - per-application **ApplicationMaster** (*AM*). 
- An **application** is either a single job or a DAG (directed acyclic graph) of jobs.
- The ResourceManager and the NodeManager form the **data-computation framework**.

YARN

- The **ResourceManager** is the **ultimate authority** that arbitrates resources among all the applications in the system.
 - has a pluggable **Scheduler**, which is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc.
 - The Scheduler performs its scheduling function based on the *resource requirements* of the applications; it does so based on the abstract notion of a **Resource Container** which incorporates resource elements such as memory, CPU, disk, network etc.

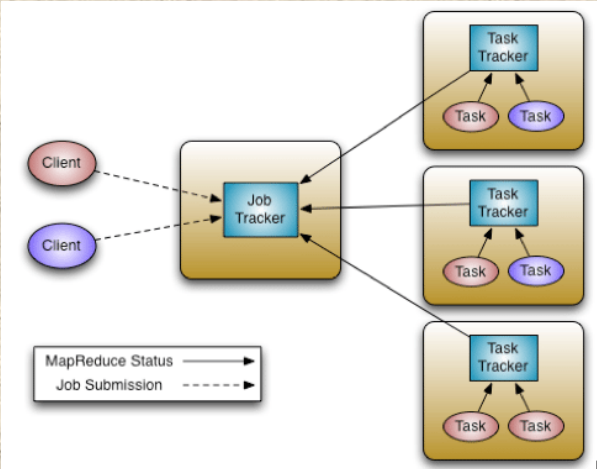
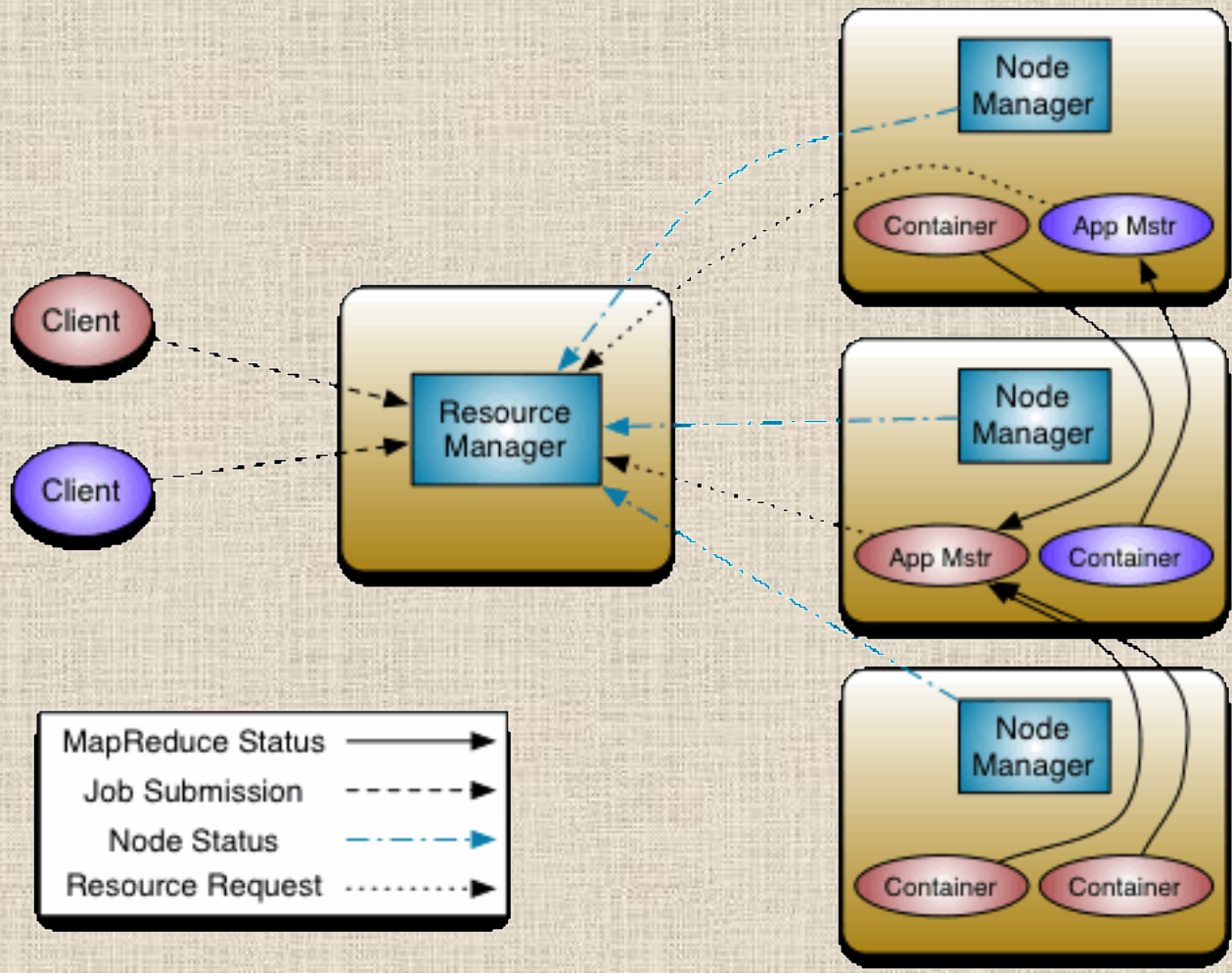


YARN

- The **NodeManager** is the **per-machine framework agent** who is responsible for **containers**, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager/Scheduler.
- The per-application **ApplicationMaster (AppMstr)** is, in effect, a **framework specific library** and is tasked with **negotiating resources** from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.
 - From the system perspective, the ApplicationMaster itself runs as a normal *container*.



YARN



MapReduce

Data Management -HDFS

- Store and process **vast quantities of data** in a storage layer that scales linearly.
- **Hadoop Distributed File System (HDFS)** is the core technology for the **efficient scale out storage layer**, and is designed to run across low-cost commodity hardware.
 - is a Java-based file system that provides scalable and reliable data storage that is designed to span large clusters of commodity servers.

HDFS – Hadoop Distributed File System

- HDFS:
 - is a scalable, fault-tolerant, distributed storage system that works closely with a wide variety of concurrent data access applications, **coordinated by YARN.**
 - will “just work” under a variety of physical and systemic circumstances.
- By distributing storage and computation **across many servers**, the combined storage resource **can grow linearly with demand** while **remaining economical** at every amount of storage.

HDFS

- An HDFS cluster is **comprised of** a **NameNode**, which manages the cluster metadata, and **DataNodes** that store the data.
- Files and directories are represented on the NameNode by **inodes**. Inodes record attributes like permissions, modification and access times, or namespace and disk space quotas.
- **The file content is split into large blocks (typically 128 megabytes), and each block of the file is independently replicated at multiple DataNodes.**
- The blocks are stored on the local file system on the DataNodes.

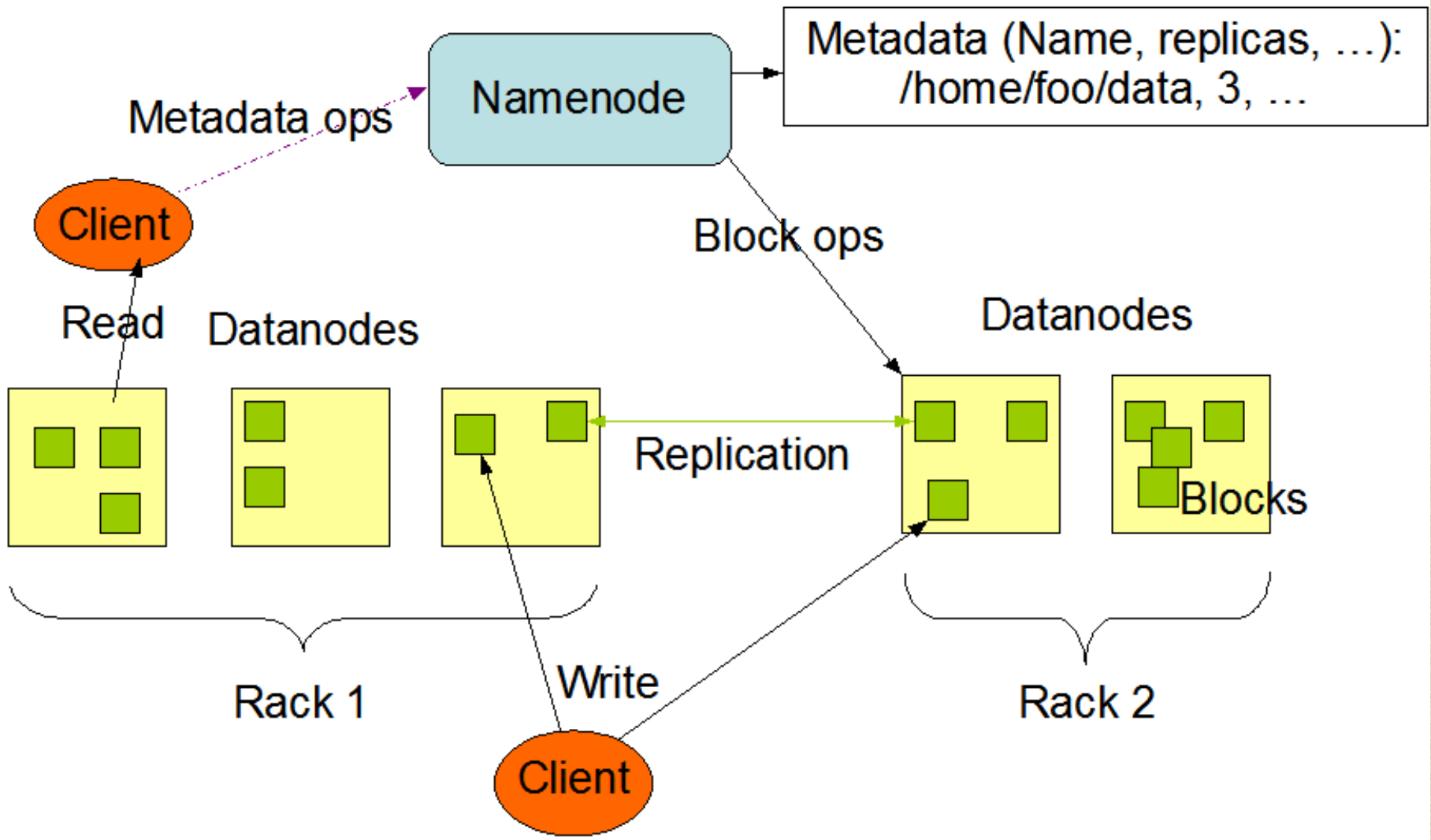
HDFS

- The **Namenode** actively monitors the number of replicas of a block.
 - When a replica of a block is lost due to a DataNode failure or disk failure, the NameNode creates another replica of the block.
 - The NameNode maintains the namespace tree and the mapping of blocks to DataNodes, holding the entire namespace image in RAM.
 - The NameNode **does not directly send requests to DataNodes**. It sends instructions to the DataNodes by **replying to heartbeats** sent by those DataNodes. The instructions include commands to:
 - replicate blocks to other nodes
 - remove local block replicas
 - re-register and send an immediate block report, or
 - shut down the node

HDFS cartoon

- <https://wiki.scc.kit.edu/gridkaschool/upload/1/18/Hdfs-cartoon.pdf>
- What can you summarize from the cartoon?
 - Writing data in HDFS cluster
 - Reading data in HDFS cluster
 - Fault tolerance in HDFS

HDFS Architecture



HDFS architecture

- The file content is **split into large blocks** (typically 128 megabytes), and each block of the file is **independently replicated at multiple DataNodes**. The blocks are stored on the local file system on the DataNodes.
- The Namenode **actively monitors the number of replicas of a block**. When a **replica** of a block is **lost** due to a DataNode failure or disk failure, **the NameNode creates another replica of the block**. The NameNode **maintains the namespace tree and the mapping of blocks to DataNodes**, holding the entire namespace image in RAM.
- The NameNode does not directly send requests to DataNodes. It **sends instructions to the DataNodes by replying to heartbeats** sent by those DataNodes. The instructions include commands to:
 - **replicate blocks to other nodes,**
 - **remove local block replicas,**
 - re-register and send an immediate block report, or
 - shut down the node.

Data Access



- Interact with your data in a wide variety of ways – **from batch to real-time.**
 - **Apache Hive** – Built on the MapReduce framework, Hive is a data warehouse that **enables easy data summarization and ad-hoc queries** via an SQL-like interface for large datasets stored in HDFS.
 - **Apache Pig** – A platform for **processing and analyzing large data sets.** Pig consists of a high-level language (Pig Latin) for expressing data analysis programs paired with the MapReduce framework for processing these programs.

Data Access

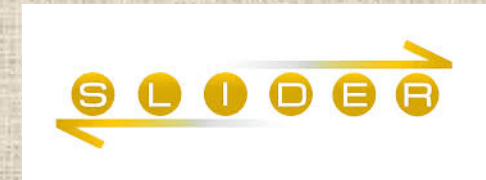


- Interact with your data in a wide variety of ways – from **batch to real-time**.
 - **Apache Spark** – Spark is ideal for **in-memory data processing**. It allows data scientists to implement fast, iterative algorithms for advanced analytics such as clustering and classification of datasets.
 - **Apache HBase** – A **column-oriented NoSQL** data storage system that provides **random** real-time read/write access to big data for user applications.

Data Access

- **Others:**

- Apache Storm
- Apache Kafka
- Apache HCatalog
- Apache Slider
- Apache Tez
- Apache Solr
- Apache Mahout
- Apache Accumulo



Data Governance and Integration



- Quickly and easily load data, and manage according to policy.
 - Workflow Management
 - Apache Flume
 - Apache Sqoop

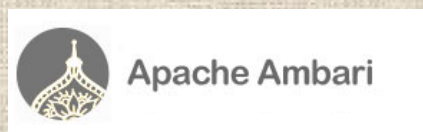
Security



Apache Ranger

- **Address requirements of Authentication, Authorization, Accounting and Data Protection.**
 - Apache Knox
 - Apache Ranger
- Security is provided at every layer of the Hadoop stack from HDFS and YARN to Hive and the other Data Access components on up through the entire perimeter of the cluster via Apache Knox.

Operations



- Provision, manage, monitor and operate Hadoop clusters at scale.
 - Apache Ambari
 - Apache Oozie
 - Apache ZooKeeper

Applications

- Apache Hadoop can be useful across a range of use cases spanning virtually every industry.
- It is becoming popular anywhere that you **need to store, process, and analyze large volumes of data**. Examples:
 - digital marketing **automation**,
 - fraud detection and prevention,
 - social network and relationship analysis,
 - predictive modeling for new drugs,
 - retail in-store behavior analysis, and
 - mobile device location-based marketing.

Data Pipeline

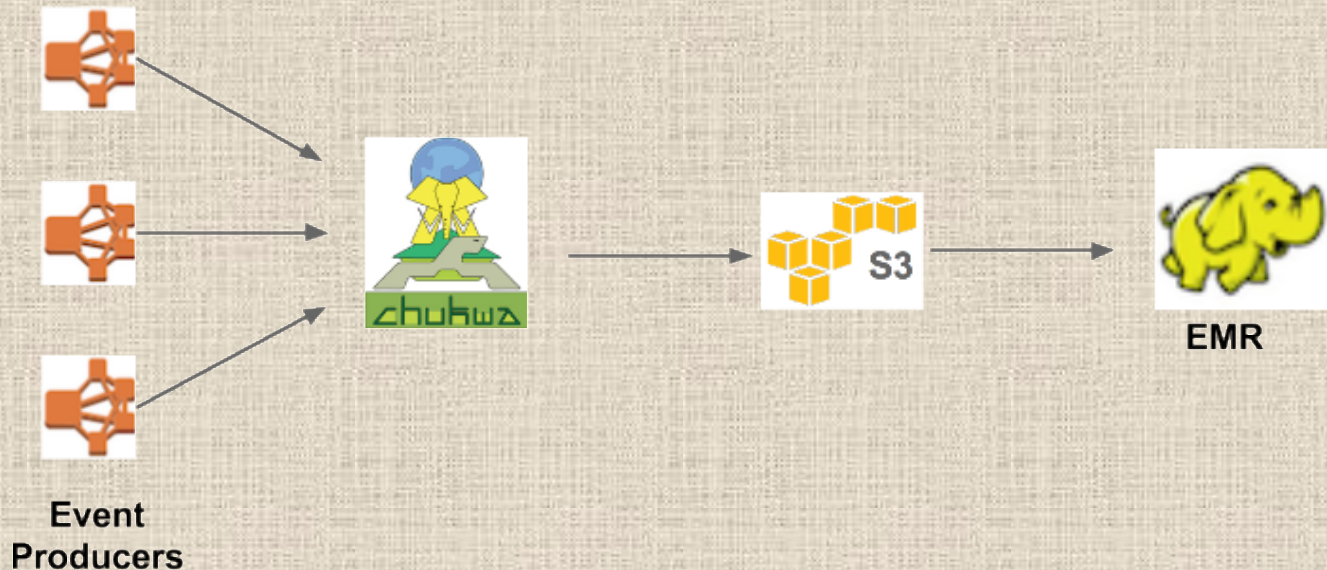
- Is the process of structuring, processing, and transforming data in stages regardless of what the source data form may be.
- to collect, aggregate, process and move data at cloud scale.

Example: Netflix Data Pipeline

- There are several hundred **event streams flowing** through the pipeline. For example:
 - Video viewing activities
 - UI activities
 - Error logs
 - Performance events
 - Troubleshooting & diagnostic events

Example: Netflix – V1.0 Chukwa pipeline

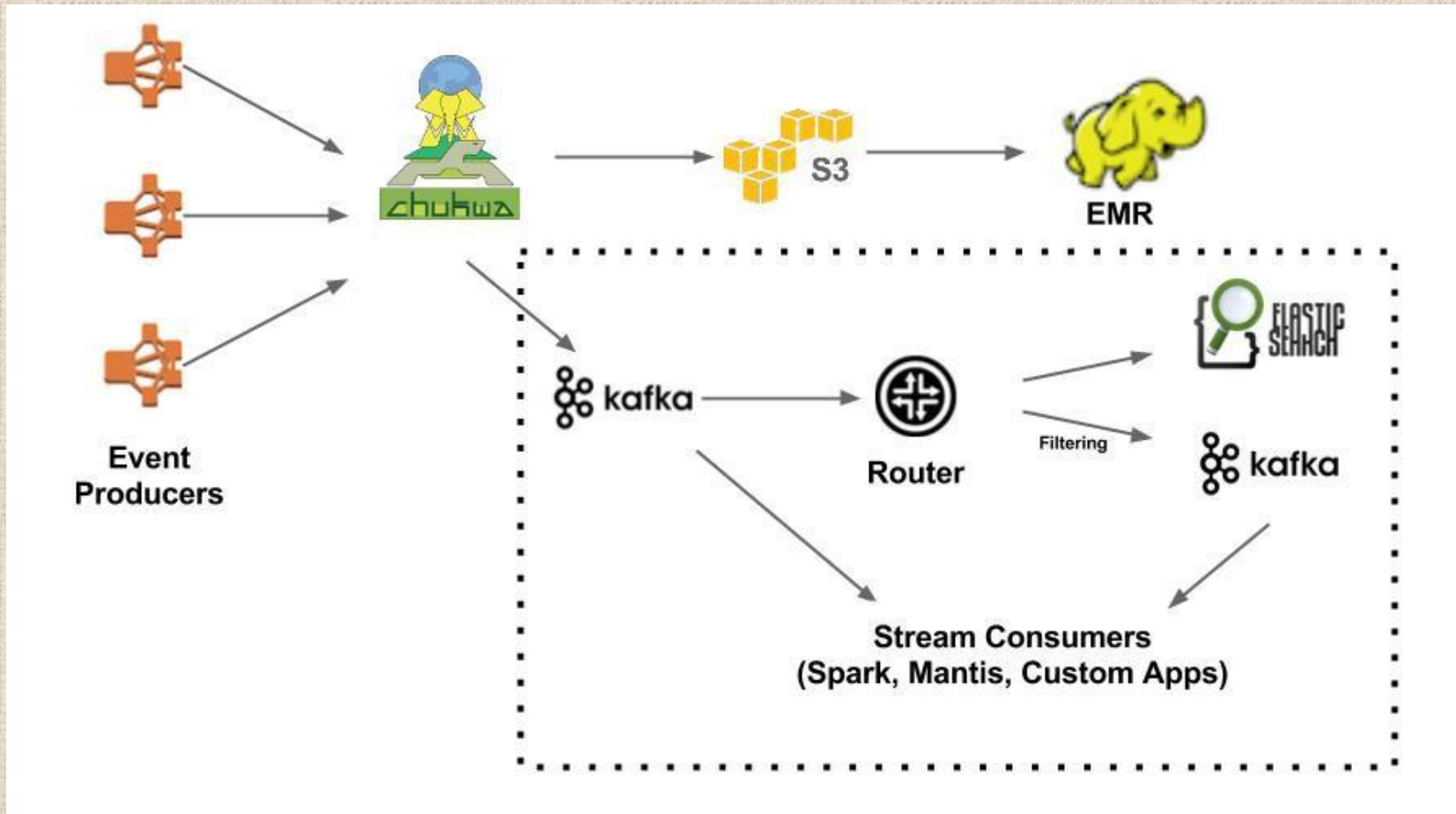
- To aggregate and upload events to **Hadoop/Hive** for **batch processing**.
- **Chukwa** collects events and writes them to S3 in Hadoop sequence file format.
- The Big Data Platform team further processes those S3 files and writes to Hive in Parquet format.
- End-to-end latency is up to 10 minutes. That is sufficient for batch jobs which usually scan data at daily or hourly frequency.



Example: Netflix – V1.5 Chukwa pipeline with real-time branch

- With the emergence of **Kafka and Elasticsearch** over the last couple of years, there has been a **growing demand for real-time analytics** in Netflix.
- In addition to uploading events to **S3/EMR**, Chukwa can also **tee traffic to Kafka** (the front gate of real-time branch). In V1.5, approximately 30% of the events are branched to the real-time pipeline. The centerpiece of the real-time branch is the router. It is responsible for **routing data from Kafka to the various sinks: Elasticsearch or secondary Kafka**.
 - When Chukwa tees traffic to Kafka, it can deliver full or filtered streams. Sometimes, further filtering is applied on the Kafka streams written from Chukwa. That is why the router is needed to consume from **one Kafka topic** and produce to a different Kafka topic.
- Once data is delivered to Kafka, it **empowers** users with real-time stream processing: **Mantis, Spark**, or custom applications.

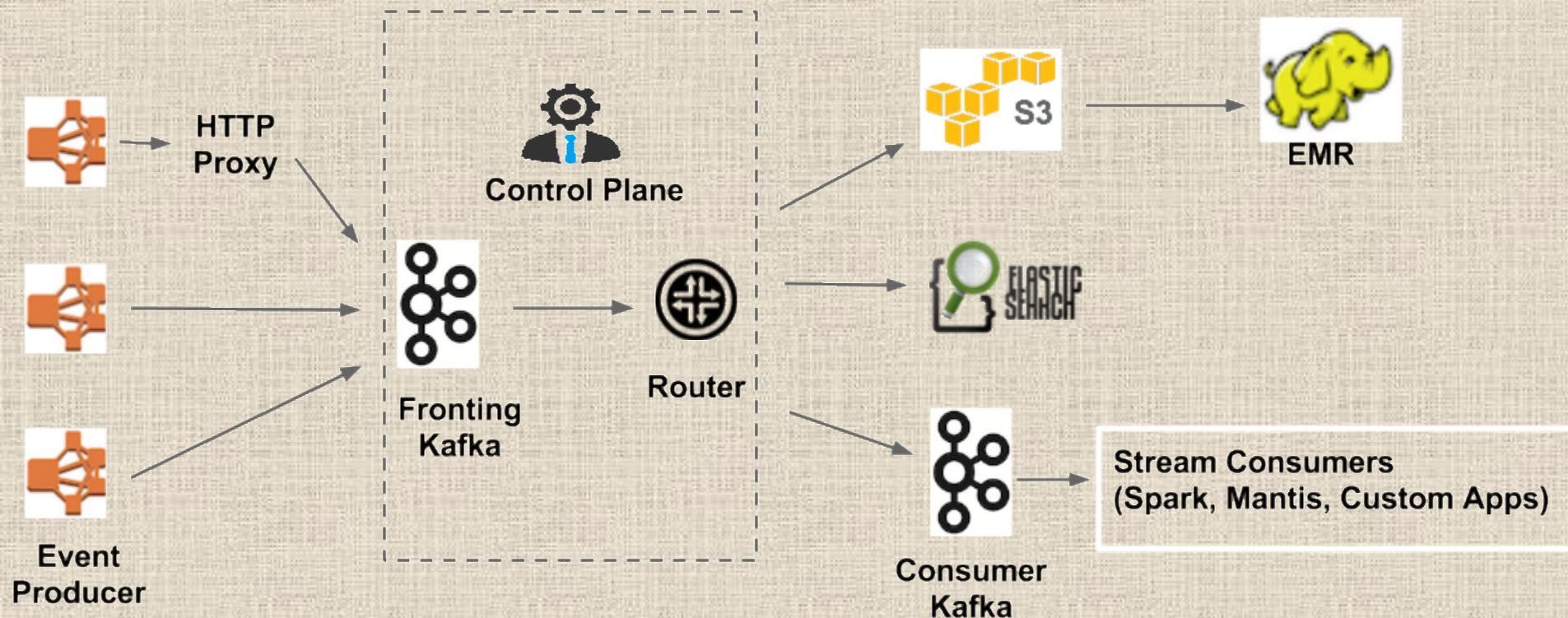
Example: Netflix – V1.5 Chukwa pipeline with real-time branch



Example: Netflix – V2.0 Keystone pipeline (Kafka fronted)

- In addition to the issues related to routing service, there are other motivations to revamp the data pipeline:
 - **Simplify the architecture.**
 - Kafka implements replication that **improves durability**, while Chukwa doesn't support replication.
 - Kafka has a vibrant community with strong momentum.
- There are three major components:
 - **Data Ingestion** — There are two ways for applications to ingest data: use our Java library and write to Kafka directly. send to an HTTP proxy which then writes to Kafka.
 - **Data Buffering** — Kafka serves as the replicated persistent message queue. It also helps absorb temporary outages from downstream sinks.
 - **Data Routing** — The routing service is responsible for moving data from fronting Kafka to various sinks: S3, Elasticsearch, and secondary Kafka.

Example: Netflix – V2.0 Keystone pipeline (Kafka fronted)



Another example: Customer Churn Analysis

