Aalto University
School of Science

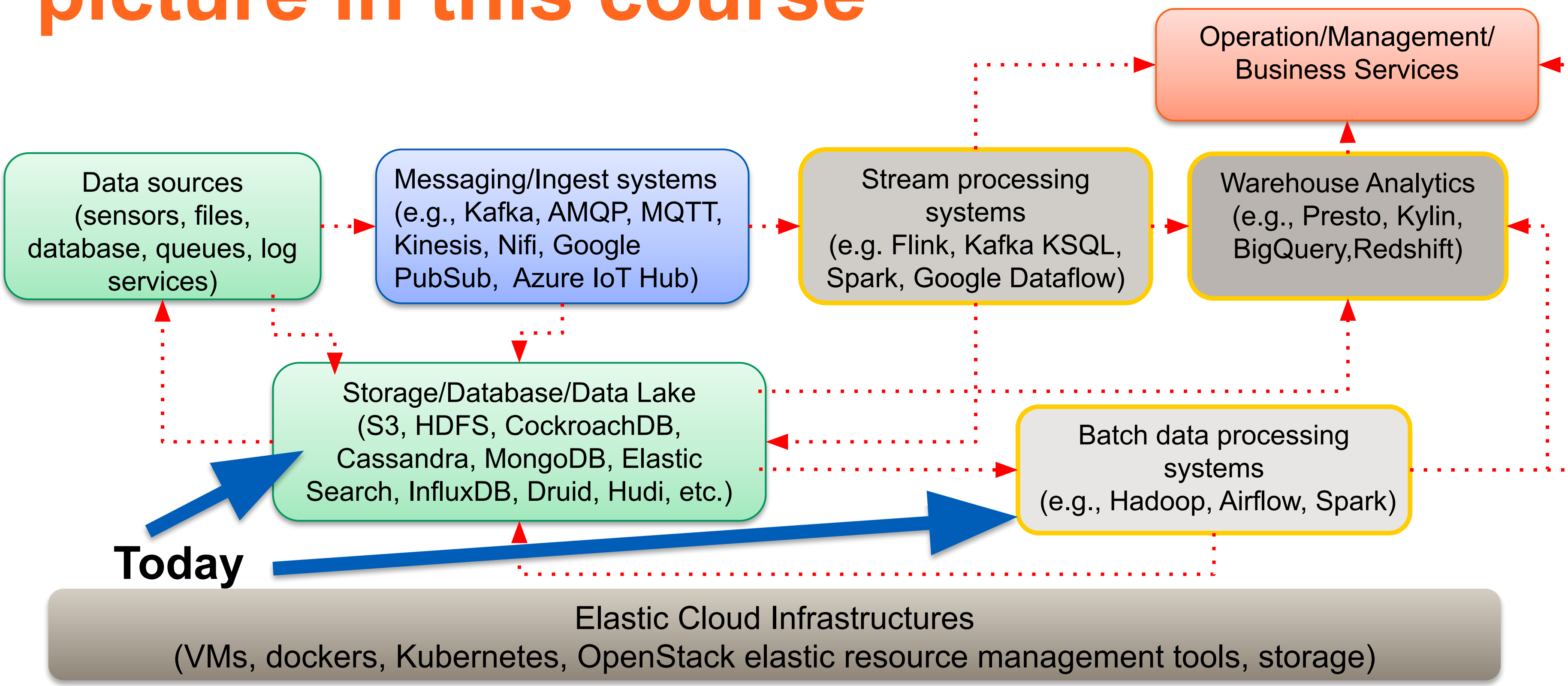
Hadoop and its Big Data Ecosystems

Hong-Linh Truong
Department of Computer Science
linh.truong@aalto.fi, https://rdsea.github.io

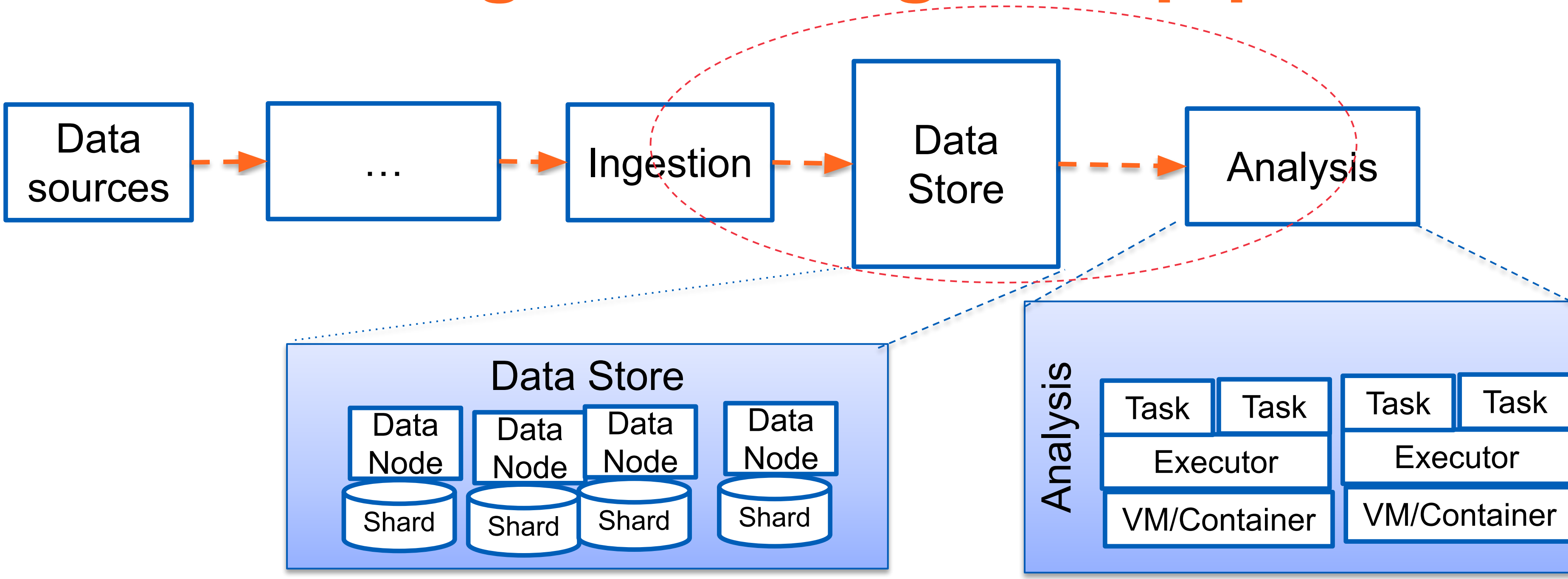
Learning objectives

- Understand massive scale data management and processing with Hadoop
- Understand and able to use Hadoop components for big data platform designs
- Able to integrate Hadoop with other frameworks for data ingestion and analytics systems

Big data at large-scale: the big picture in this course

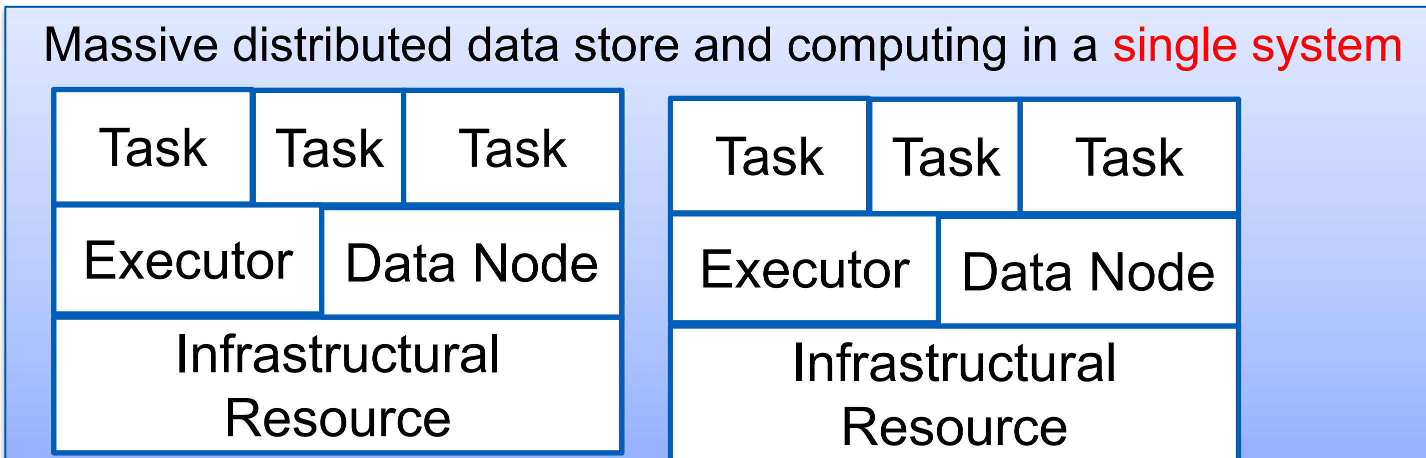


Consider again our big data pipelines



Combining data storage and analysis

Data locality, massive parallel and distributed data processing with large-scale data store

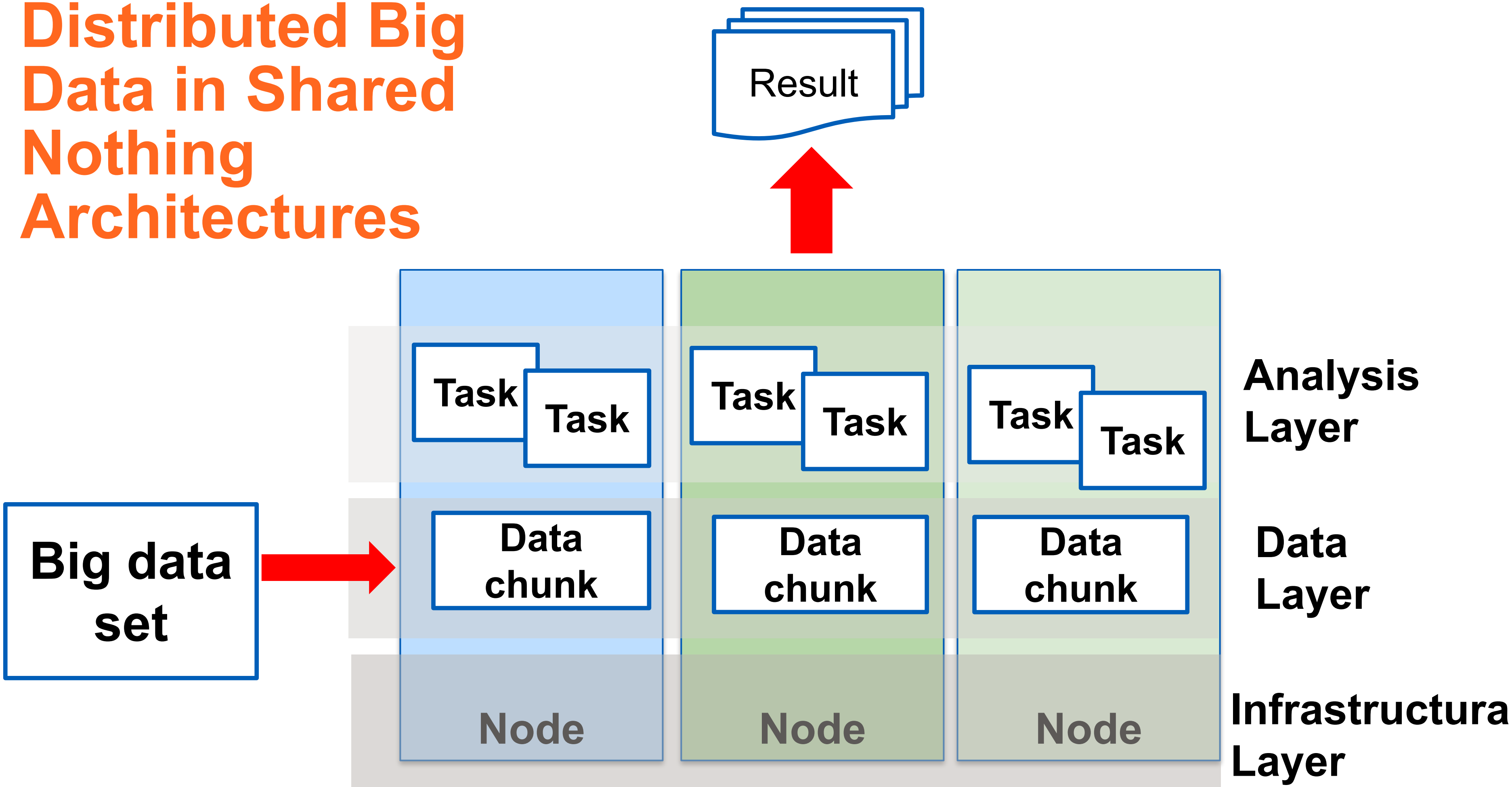


Single system: many machines connected into a type of "cluster"

Shared nothing architectures

- **Machine as a node:**
  - completely independent from other nodes
  - no shared physical resources (CPUS, memory, hard disk, ...)
- **A distributed system of machines with shared nothing architectures**
  - allow to scale and extend physical resources easily
  - enable high fault tolerance designs
  - quite common and easy-to-be-setup for enterprises
    - *think about commodity clusters*

Distributed Big Data in Shared Nothing Architectures



Benefit

- Reduce cost and increase performance when moving data is much more expensive than moving computation
- Consolidate and integrate various types of data
- Save shared infrastructural resources
- Provide suitable solutions for different types of customers/clients and different infrastructures

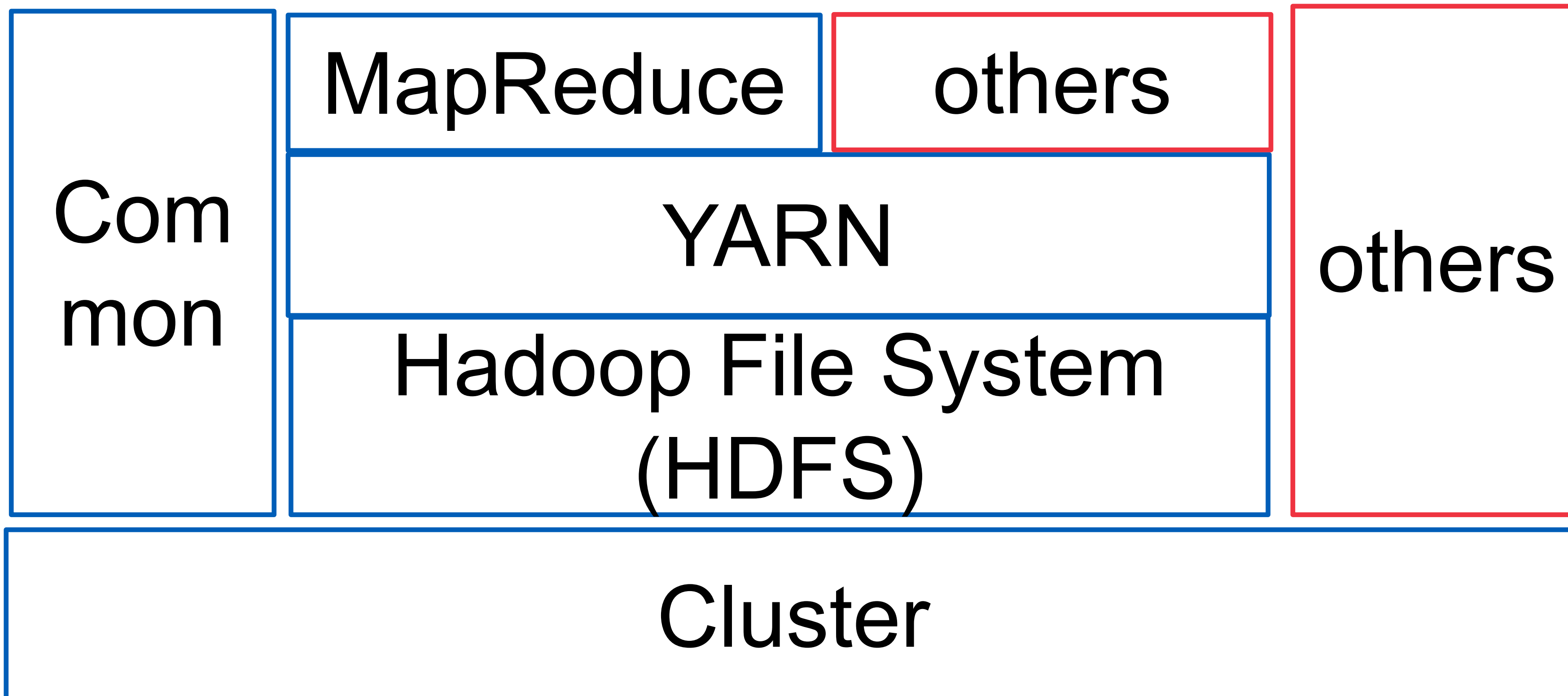
Benefit - Variety

- **Multiple data formats**
- **In big data we have a lot of files**
  - should we always transform them into a single format?
  - must always be in a database form?
- **Processing files directly is a benefit!**
- **The datalake concept:**
  - multiple types of big data analytics with high concurrent/parallel data writes/reads
  - dealing with different data access frequencies: **hot, warm and cold data**

Hadoop

- **http://hadoop.apache.org/, original from Yahoo**
- **The goal is to combine storage and processing in the same cluster system**
- **Designed for *massive scale of data and computing with shared nothing architecture***
  - commodity hardware, highly scalability, fault tolerance, easy to extend
- **Suitable for both on-premise and clouds**
- **There are very rich software ecosystems centered around Hadoop**

Hadoop: layers



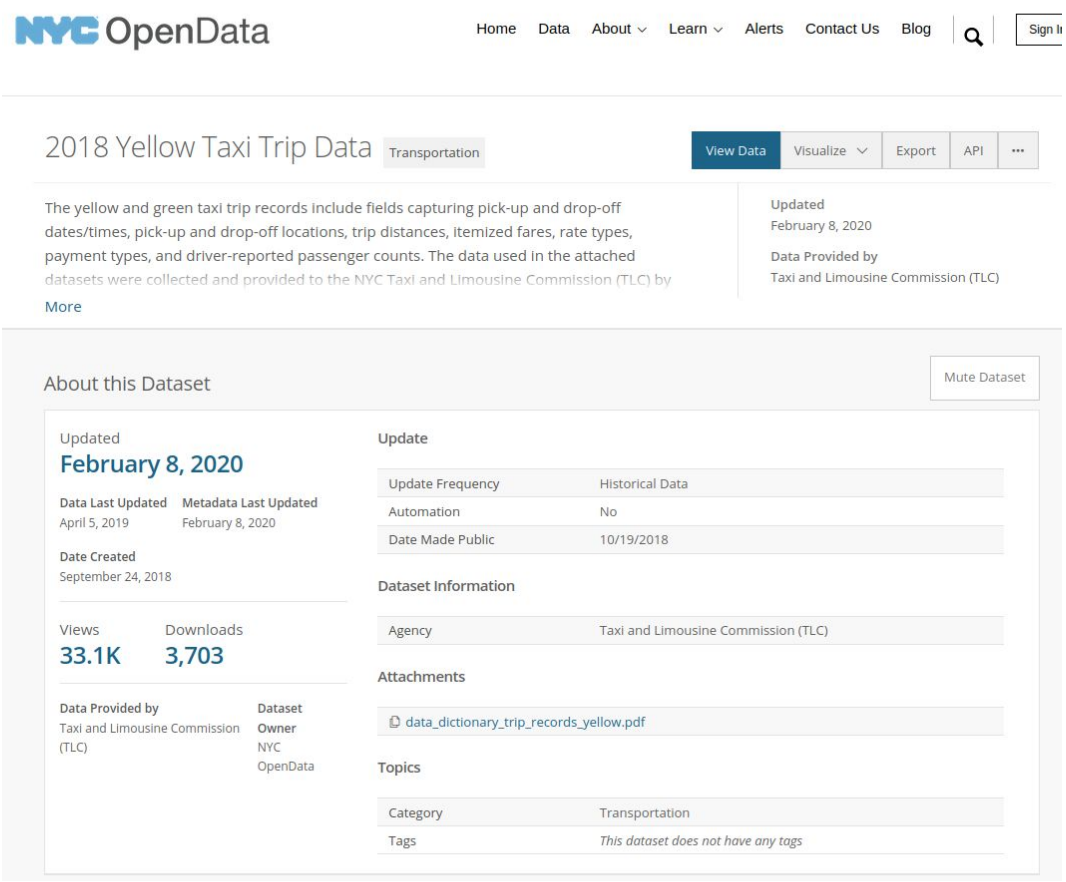
Hadoop key components

- **HDFS as a distributed file system**
  - for managing data
- **YARN as a resource management system**
  - for executing and managing tasks
- **MapReduce as one programming model**
  - for MapReduce applications
- **Coordination (ZooKeeper)**
  - for fault tolerance and metadata

Hadoop File System (HDFS)

- **For handling very big data files**
  - GBs of data within a single file
- **Assume model of data**
  - write-once-read-many
  - not suitable for random-access update
- **Deal with hardware failures, support data locality, reliability**

Example



e.g., 112M rows

Snapshot from: https://data.cityofnewyork.us/Transportation/2018-Yellow-Taxi-Trip-Data/t29m-gskq

HDFS - data blocks

- **Files are stored in many nodes**
  - but we access them just like “typical file systems”
- **A file includes many blocks**
- **File blocks are replicated and distributed across nodes**
- **Conventional way of access data**
  - naming resolving: hdfs://
  - common operations: list, put, get, …

HDFS Architecture

Crucial components

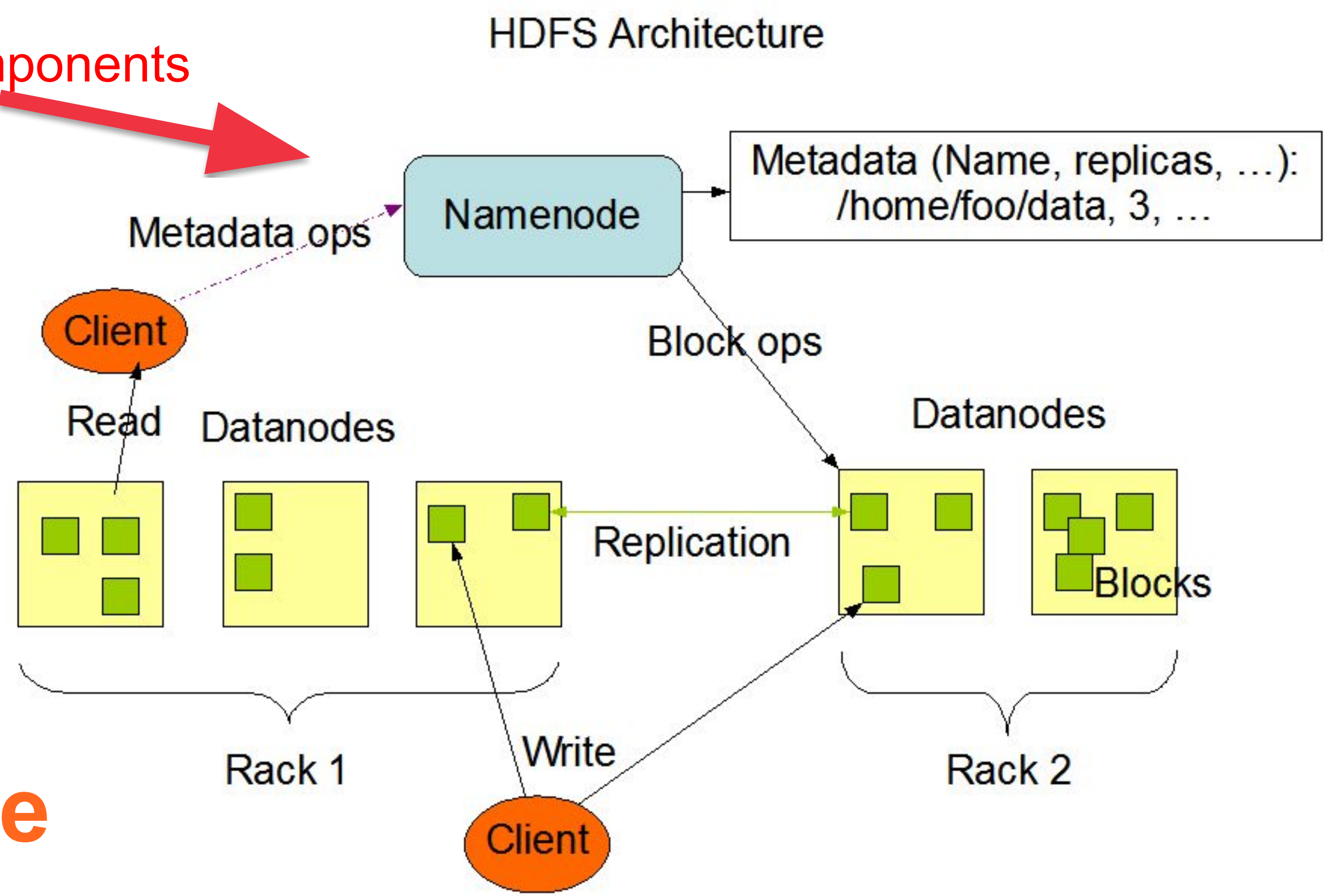


Figure source: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

File blocks, metadata and data replication

- **Block size is 128MB (default)**
  - can be configurable but should not be small size
  - all blocks of the same file are the same, except the last one
- **Data is replicated across the cluster**
  - usually, replication factor is 3
- **NameNode manages file system metadata**

HDFS fault tolerance

- **Data blocks**
  - file blocks are replicated and distributed across nodes
- **Rack awareness**
  - avoid communication problems between nodes in different racks
- **Monitoring**
  - DataNode reports to NameNode
- **Read and write**
  - using NameNode for metadata and for information of DataNodes
  - NameNode has replication (master-worker)

Compatible file systems with HDFS

- For integration and analysis purpose: many file systems are compatible with HDFS
- Amazon S3
- Azure Blob Storage
- Azure Data Lake Storage
- OpenStack Swift

Is that Hadoop dead/unattractive?

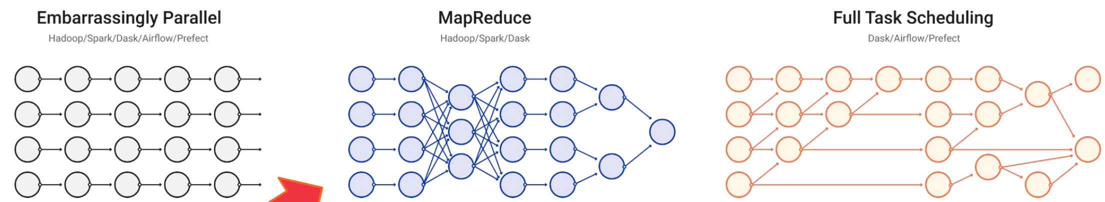
- **Hadoop is for shared nothing architecture**
- **Shared nothing architectures are not “morden”?**
  - new, advanced developments w.r.t. memory, networks, ...
  - in-memory processing, move data to computing nodes
- **Many tools seem not powerful for data science/ML?**
  - Java vs Python development for data science and ML
  - services in ecosystem
  - near realtime analytics use cases
- **So why do we still study it?**
  - foundation issues & still many Hadoop services are important (data warehouse, deep storage)
  - well-supported by many cloud providers for big data workloads

Why do we still need to learn Hadoop?

- **Hadoop file systems as a storage for many big data services**
  - databases: Accumulo, Apache Druid
  - data warehouses/Datalakes: Hive, Hudi
- **Support different models of access/analytics**
  - via SQL styles atop big data platforms: You can do extract/transform/load (ETL), reporting, and data analysis using SQL styles
  - large-scale parallel programming (e.g., Spark)
- **Still many needs given shared nothing architectures**

Why do we still need to learn Hadoop?

Figure source: https://docs.dask.org/en/stable/graphs.html



SQL-style → **needs for internal processing** → Data parallelism

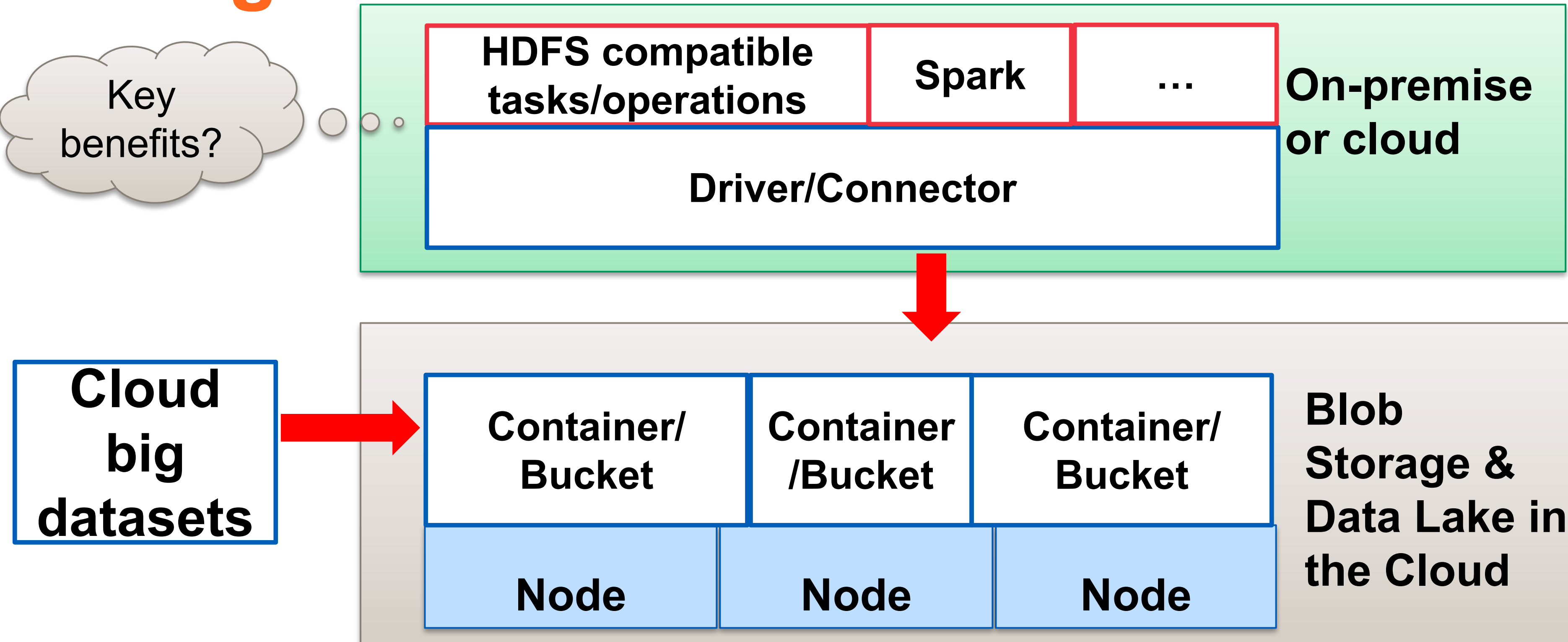
Data parallelism → **enables the implementation** → SQL-on-Hadoop

SQL-on-Hadoop → **offers SQL features for analytics** → SQL-style

Enabling analysis big data using SQL style

- **Provide command line tools & JDBC and server for integration**
- **“SQL-on-Hadoop”**
- **Examples**
  - Apache Hive, http://hive.apache.org/, on top of Hadoop
    - *Data warehouse, access data in HDFS or Hbase*
  - Apache Druid (using HDFS as a deep storage)
  - Spark SQL for various types of files

Integration models with other cloud storage services



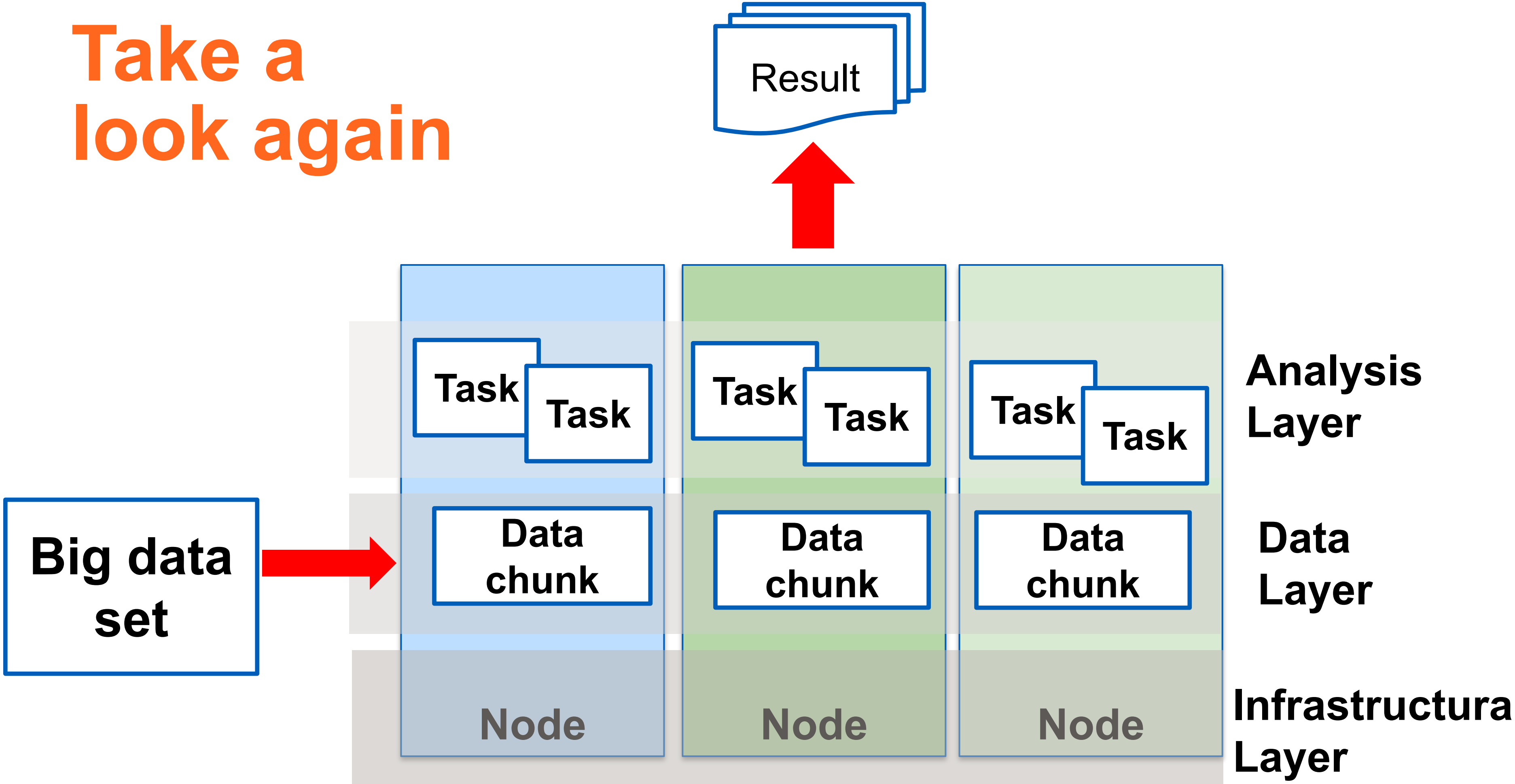
YARN (Yet Another Resource Negotiator)

If HDFS can be used to store different files, what would be the good way to enable “data processing” atop HDFS?

Key requirements:

scalability/elasticity, high utilization of infrastructural resources, serviceability for multiple users/application types

Take a look again



How to leverage the same infrastructure and data management layers to perform data analysis?

How to enable different (distributed) programming models for data analytics?

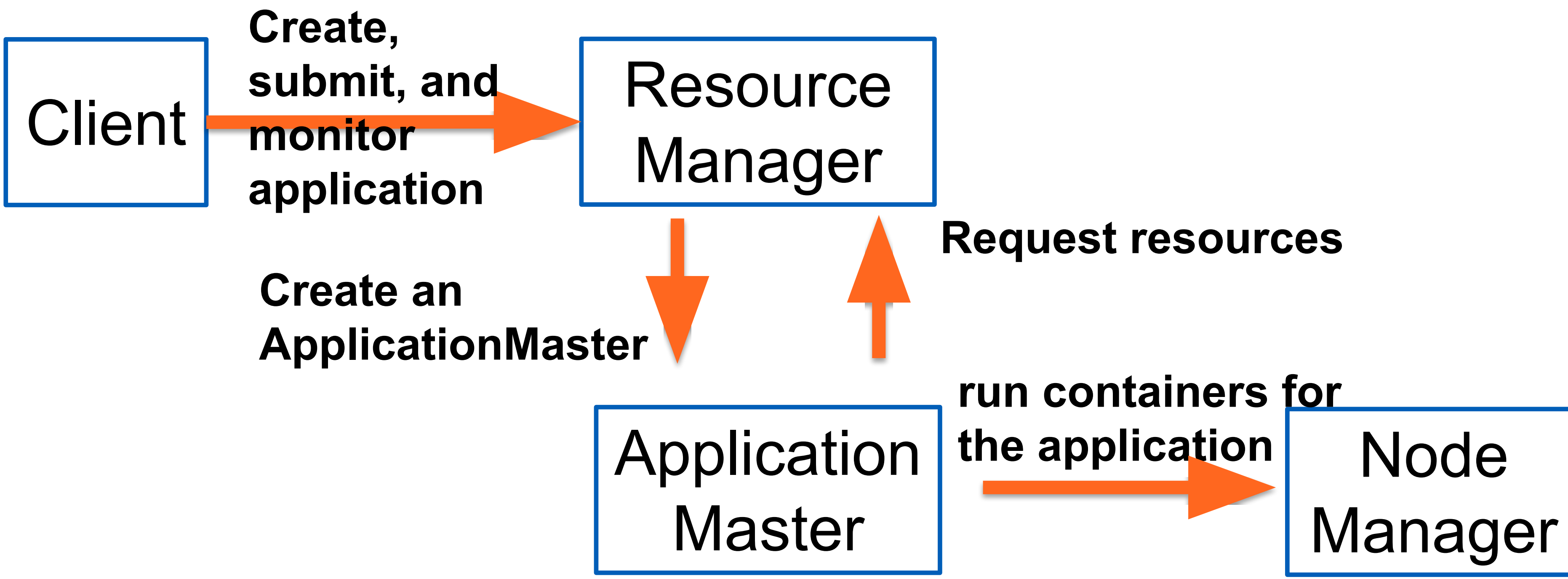
YARN (Yet Another Resource Negotiator)

- **Manage resources for processing tasks**
  - each node in the cluster provides resources for executing tasks
- **Resource types:**
  - CPU, Memory and Disks
  - also support GPU and FPGA Node
- **Resources are abstracted into "Containers"**
  - don't be confused: it is not a (Docker) container!
- **Multi-tenancy support**

YARN Components

- **Resource Manager**
  - scheduler: how to schedule tasks atop resources
- **NodeManager**
  - for managing resources of execution tasks in a node
- **ApplicationMaster**
  - application-specific manager for each application
  - work with Resource Manager and NodeManager provisioning resources and manage application execution
  - handle application-specific tasks

YARN basic model



YARN Architecture

User applications running within “Containers”:

This model decouples from specific types of applications

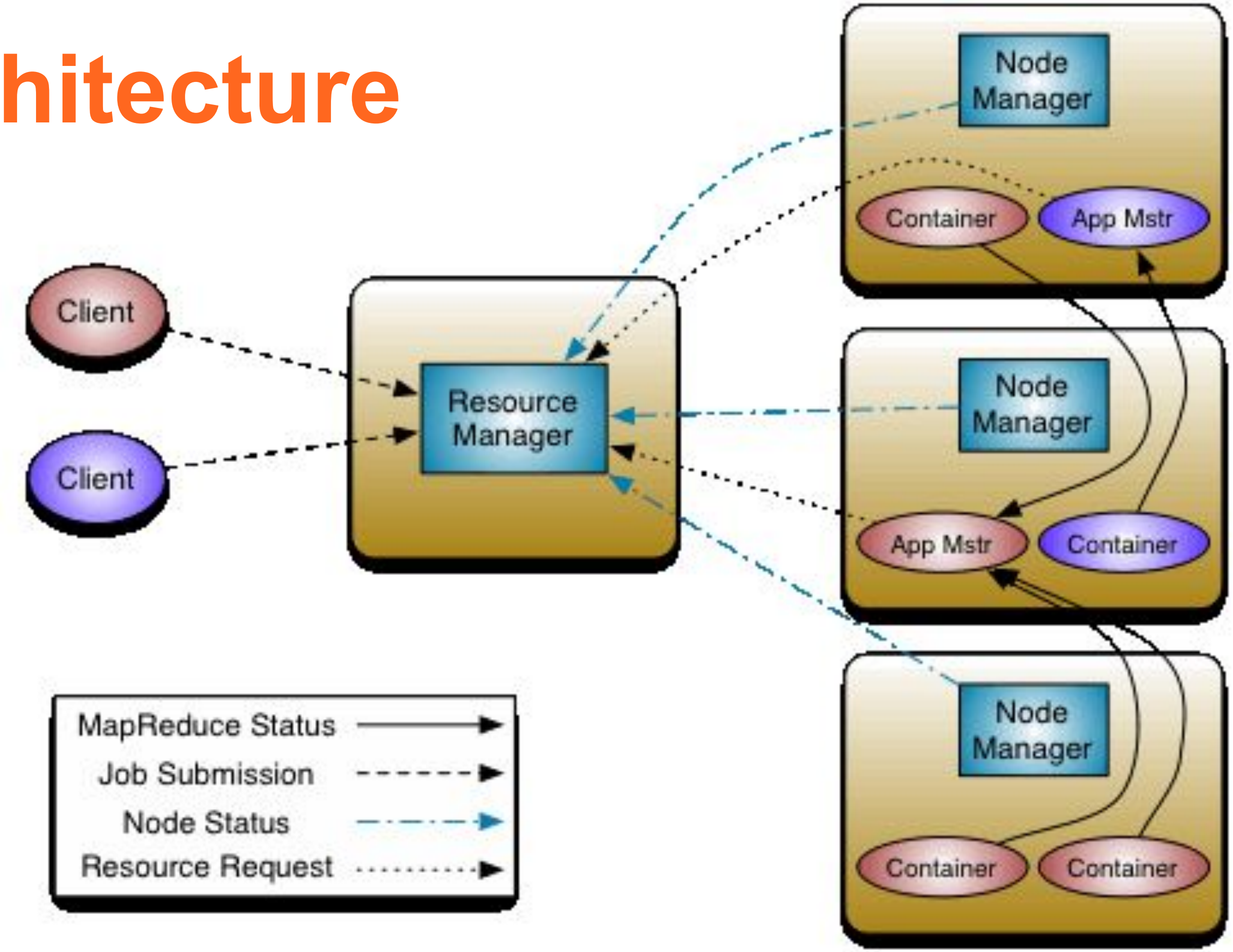


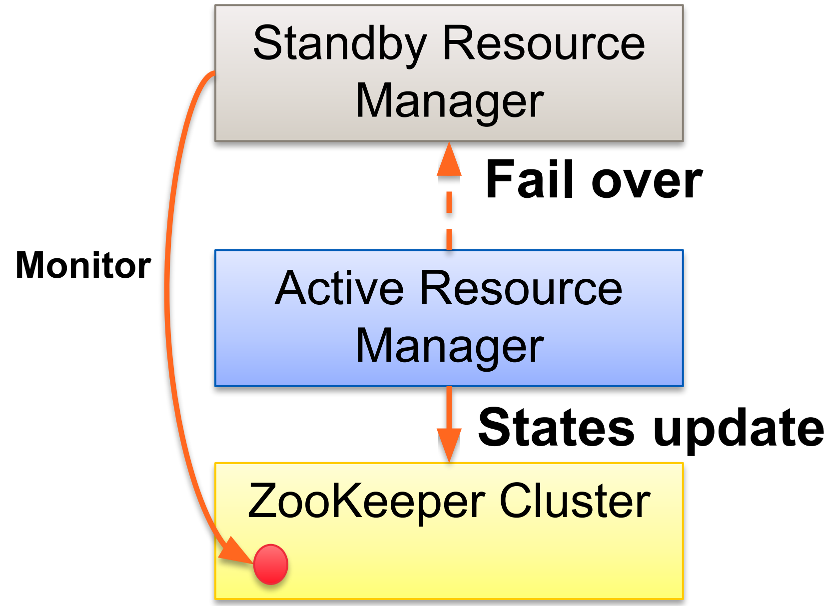
Figure source: https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html

Scheduling of tasks

- **Scheduling**
  - FIFO
  - CapacityScheduler
    - *Use multiple queues, each with a limit of resources*
  - FairScheduler
    - *All apps will get an average share of resources over time*
- **You can research and add new types of scheduling algorithms**

Fault tolerance

- **Resource Manager is a critical component**
  - active-passive Resource Manager
  - Zookeeper quorum failover
- **ApplicatonMaster**
  - ApplicationMaster is application-specific
  - Resource Manager restarts AM
- **Node**
  - remove out of the cluster



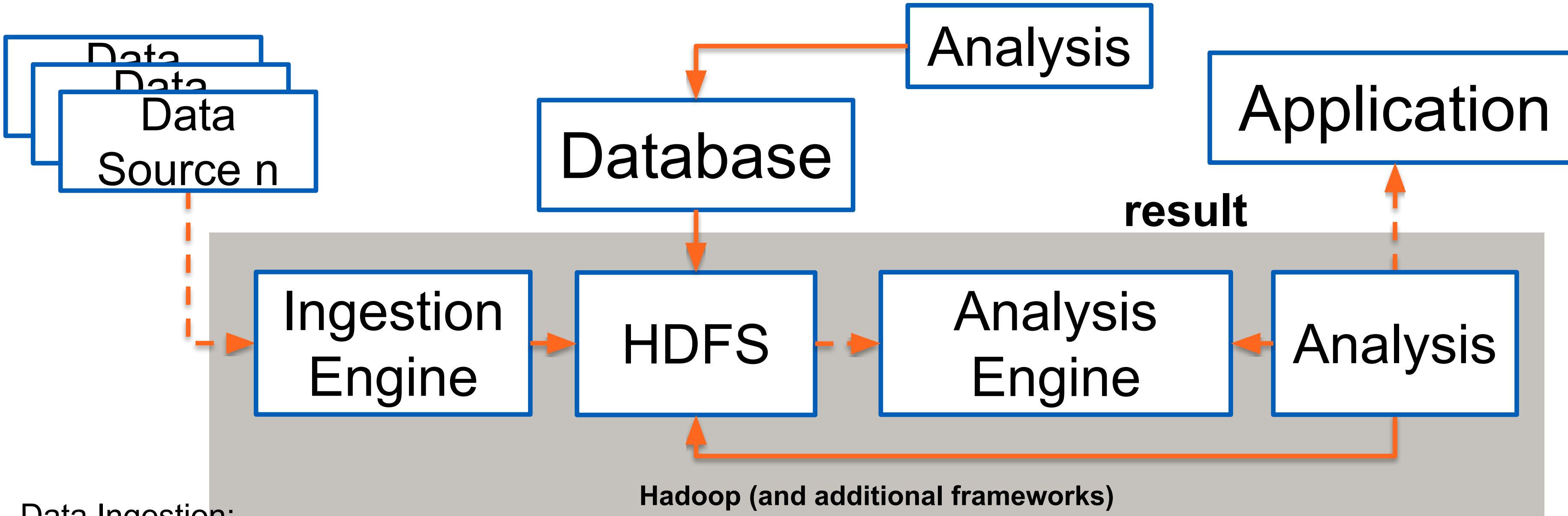
Programming models

- **YARN allows different programming models for applications**
  - MapReduce
  - Apache Spark
  - Workflows
    - *E.g., Apache Tez*

Integration models

- **Using Hadoop for developing large-scale data analysis**
  - Apache Spark, HBase, Hive, Apache Tez
- **Using Hadoop HDFS as components in a big data system**
  - Hadoop HDFS can become data store
  - emerging datalake models, combined batch and stream ingestions for incremental data processing
    - *e.g., Apache Hudi*

ETL and Analytics with Hadoop/HDFS



- Data Ingestion:
  - HDFS Client/Hadoop Streaming
  - Spark Streaming
  - Kafka Connect
  - Apache Nifi
- HDFS as storage for databases
  - Accumulo, Druid, etc.
- Computing/Data Processing Framework
  - Apache Spark
  - Hadoop MapReduce
  - Apache Tez

Hadoop-native big database/data warehouse systems

HBase

- **NoSQL database atop Hadoop**
  - use HDFS for storing data
  - use YARN for running jobs on data
- **Follow a master-based architecture**

Reading – Why HBase?
https://engineering.fb.com/2010/11/15/core-data/the-underlying-technology-of-messages/
https://engineering.fb.com/2014/06/05/core-data/hydrabase-the-evolution-of-hbase-facebook/
https://engineering.fb.com/2018/06/26/core-data/migrating-messenger-storage-to-optimize-performance/

Recall: Big data: column-family data model

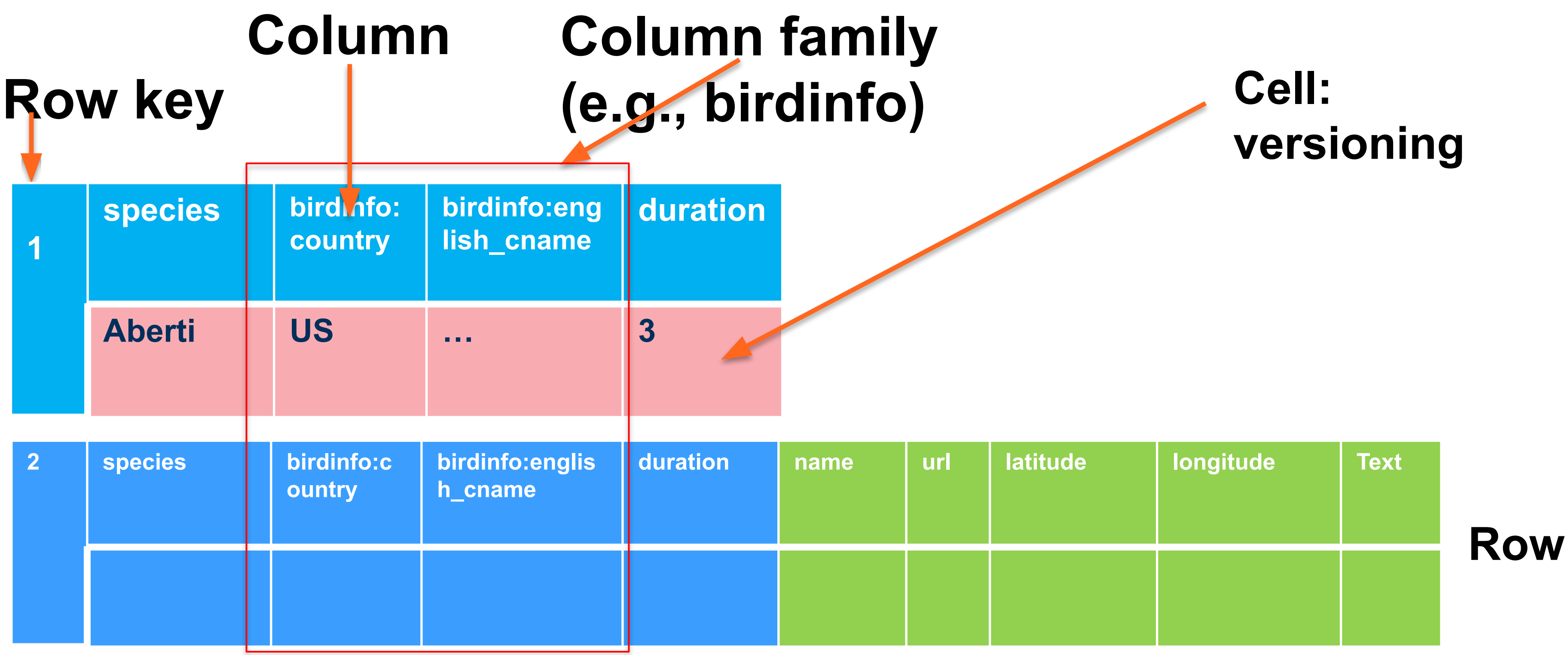
Many situations we aggregate and scan few columns of million rows of data ⇒ store big data in columns enable fast scan/retrieval/aggregation

Column Family = (Column, Column, …): for similar type of data

Column Key =Family: qualifier

Data = (Key, Value) where Key =(Row Key, Column Key, Timestamp)

Example of a data model in HBase



Example of a data model

- **Example with families: birdinfo, songinfo, location**

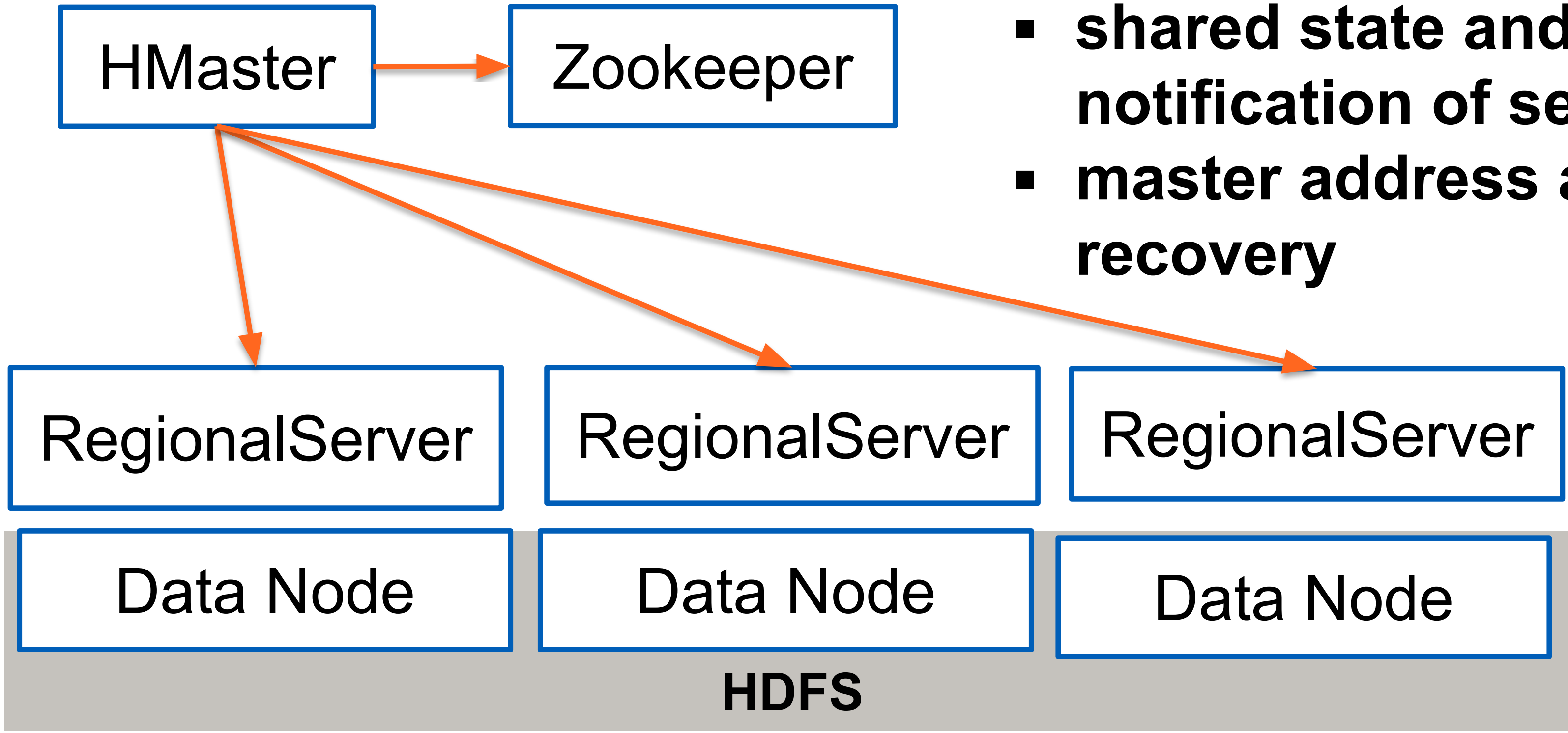
```
hbase:038:0> scan 'hbird0'
ROW                              COLUMN+CELL
 17804                           column=birdinfo:english_cname, timestamp=2022-02-05T11:56:09.114, value=Aberts Towhee
 17804                           column=songinfo:duration, timestamp=2022-02-05T11:56:09.154, value=3
 17804                           column=songinfo:file_id, timestamp=2022-02-05T11:56:09.178, value=17804
 17804                           column=songinfo:file_name, timestamp=2022-02-05T11:56:09.205, value=XC17804.mp3
 71852                           column=birdinfo:country, timestamp=2022-02-05T11:56:09.415, value=Mexico
 71852                           column=birdinfo:english_cname, timestamp=2022-02-05T11:56:09.445, value=Ash-throated Flycatc
                                 her
 71852                           column=birdinfo:species, timestamp=2022-02-05T11:56:09.488, value=cinerascens
 71852                           column=location:latitude, timestamp=2022-02-05T11:56:09.504, value=32.156
 71852                           column=location:longitude, timestamp=2022-02-05T11:56:12.253, value=-115.79299999999999
 71852                           column=songinfo:duration, timestamp=2022-02-05T11:56:09.430, value=28
 71852                           column=songinfo:file_id, timestamp=2022-02-05T11:56:09.459, value=71852
2 row(s)
Took 0.0206 seconds
```

Enable analytics based on column families (as well as data management)

Data model – sharding and storage

- **Table includes multiple Regions**
  - a Region keeps related row data of a Table (partitioning)
- **Auto-sharding**
  - Regions are spitted based on policies
- **Region has multiple column families**
  - Different column families will be stored in different files
  - HFiles are used to store real data (also include index data)

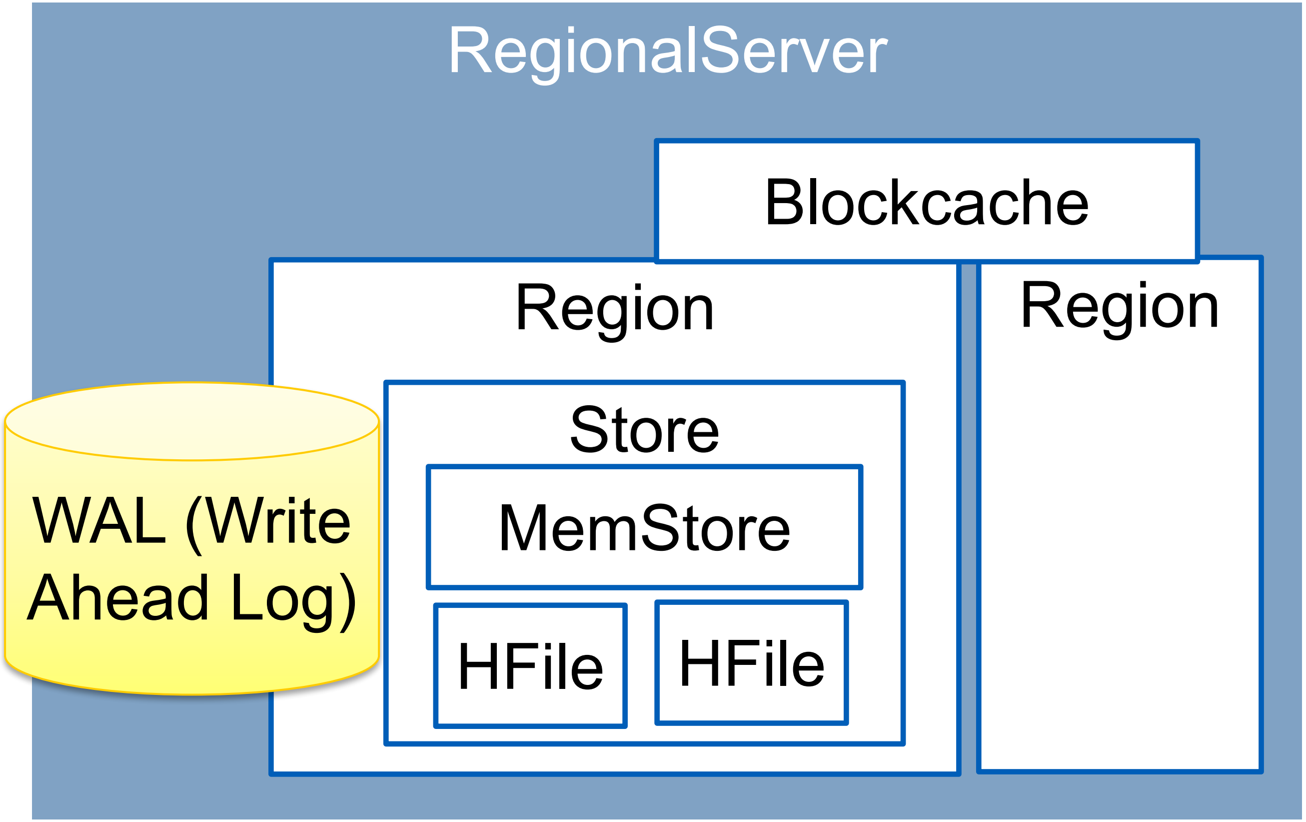
HBase architecture



Zookeeper:

- **shared state and failure notification of servers**
- **master address and recovery**

HBase architecture



MemStore: write cache for data in memory before written into files

BlockCache: for read cache

WAL is for durability

ACID

- **Atomic within a row**
- **Consistency**
    - can be programmed: e.g.,
        - read: STRONG (read performed by the primary region) and TIMELINE (primary region first, if not, then the secondary region)
- **Durability**
    - can be programmed
    - WAL (write ahead log)

Example of using HBase

Open Scalable Open Time Series Database (OpenTSDB)

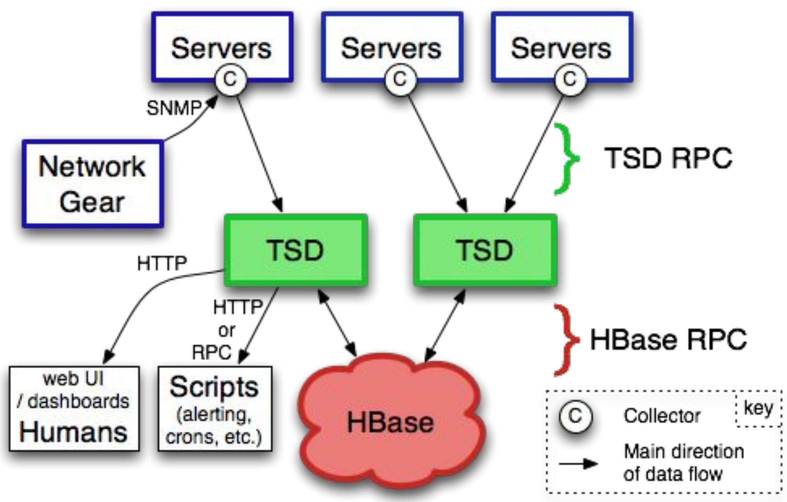


Figure source: http://opentsdb.net/overview.html

Salesforce

- **Salesforce Shield:** for supporting data security/compliance
- **Field Audit Trail:** for track changes
- **Event Monitoring:** log/event
- **Argus:** time series data and alerts

Source: https://engineering.salesforce.com/investing-in-big-data-apache-hbase-420edfba2d30/

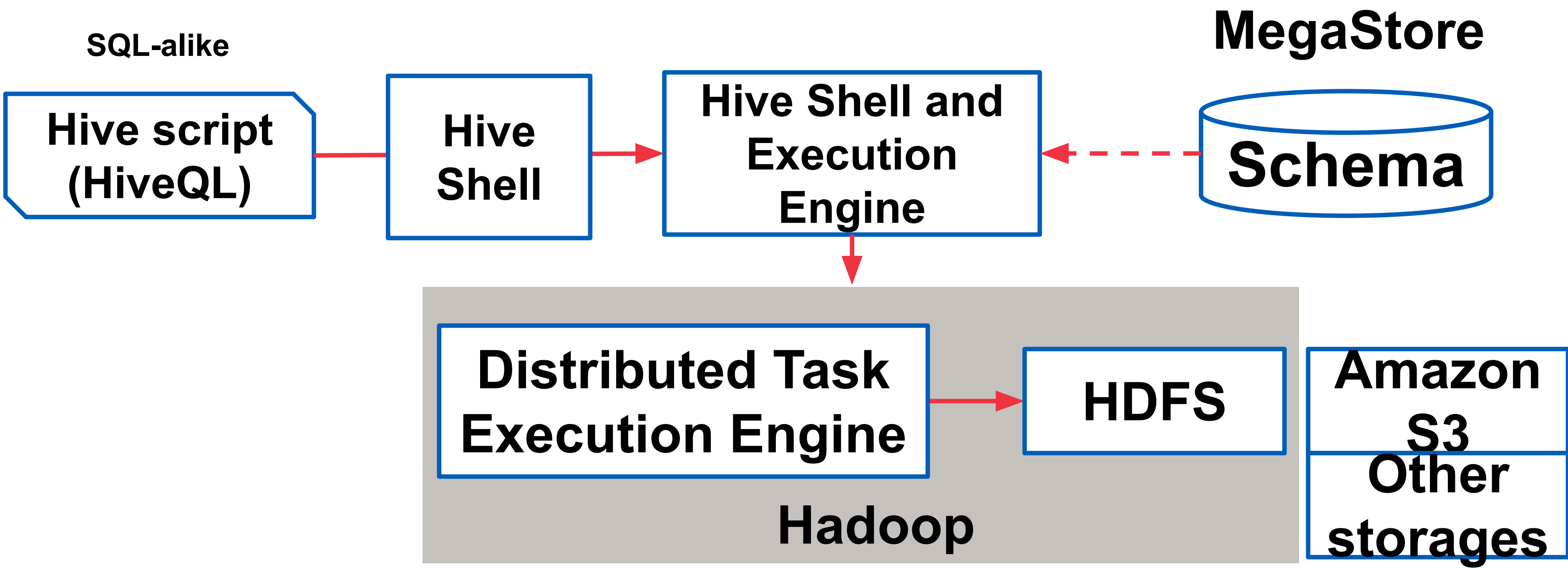
Apache Hive

Apache Hive

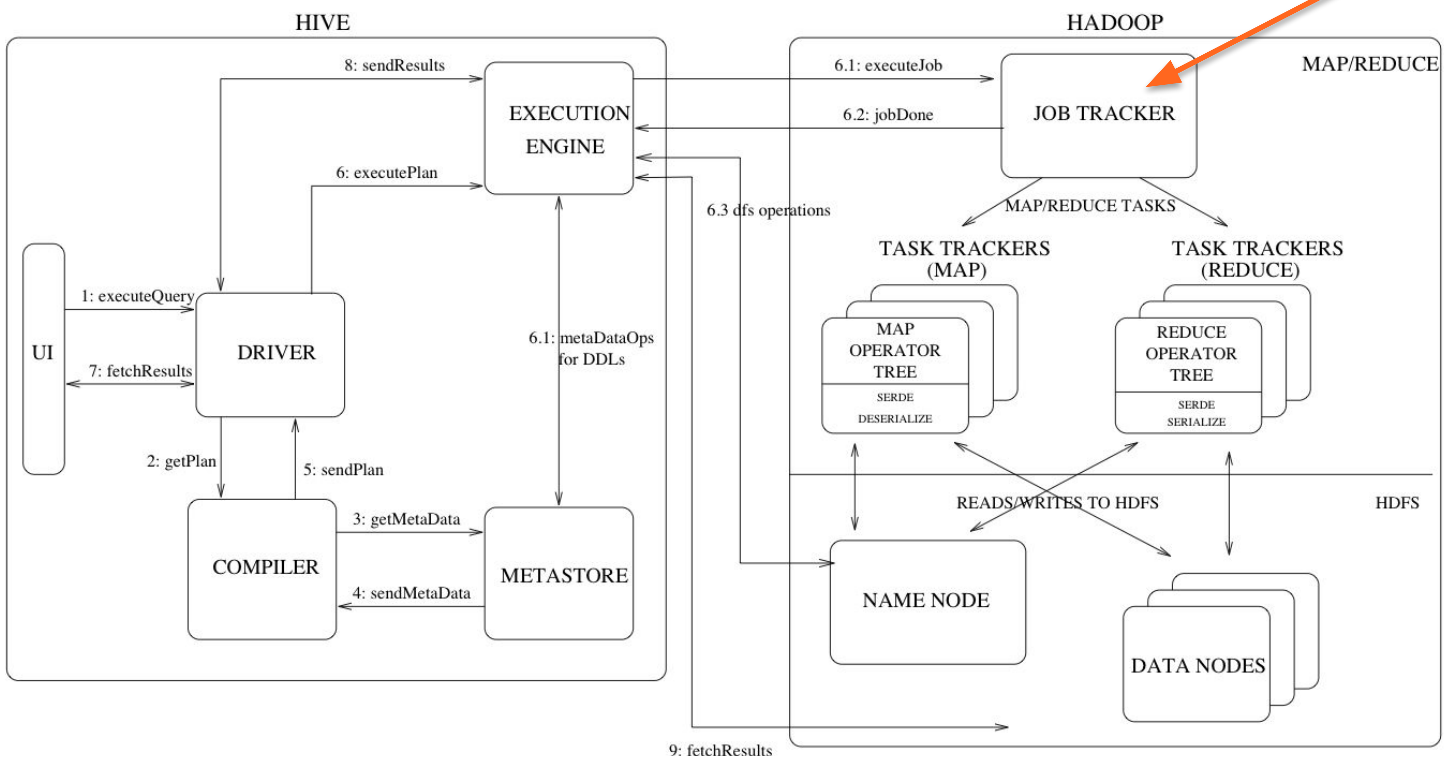
- **http://hive.apache.org/, on top of Hadoop**
  - data warehouse at a very large-scale
  - access data in large-scale storage like HDFS or S3
- **Support access to data via SQL styles**
  - extract/transform/load (ETL), reporting, and data analysis using SQL styles
- **Provide command line tools & JDBC and server for integration**

Reading – Hive in Facebook
https://engineering.fb.com/2009/06/10/web/hive-a-petabyte-scale-data-warehouse-using-hadoop/

High-level data flow language & programs



Hive building blocks



Distributed tasks with MapReduce, Tez (Workflow) or Spark

Figure source: https://cwiki.apache.org/confluence/display/Hive/Design

Hive data organization

- **Databases**
- **Table**
  - Managed table versus External tables
  - External table: data is referenced so delete only table metadata but not the data)
  - Table is mapped to a directory in HDFS
- **Low-level files: CSV/Parquet,ORC, …**

Example

Large tables lead to performance issues

```
0: jdbc:hive2://localhost:10000> describe taxiinfo;
+------------------------+------------+----------+
|        col_name        | data_type  | comment  |
+------------------------+------------+----------+
| vendorid               | int        |          |
| tpep_pickup_datetime   | timestamp  |          |
| tpep_dropoff_datetime  | timestamp  |          |
| passenger_count        | int        |          |
| trip_distance          | float      |          |
| ratecodeid             | int        |          |
| store_and_fwd_flag     | int        |          |
| pulocationid           | string     |          |
| dolocationid           | int        |          |
| payment_type           | int        |          |
| fare_amount            | float      |          |
| extra                  | float      |          |
| mta_tax                | float      |          |
| tip_amount             | float      |          |
| tolls_amount           | float      |          |
| improvement_surcharge  | float      |          |
| total_amount           | float      |          |
+------------------------+------------+----------+
```

Hive data organization for performance optimization

- **Partitioning: using value of a column as a partitioning key**
  - partition keys determine how data in Table will be divided
    - *E.g. date or countries*
  - each partition is stored as a subdirectory
    - *Avoid many sub directories!*
- **Buckets: using a hash function of a column for grouping records into the same bucket**
  - avoid large number of small partitions
  - each bucket is stored in a file
  - bucket columns should be optimized for join/filter operations

Example of partitions

```
CREATE TABLE taxiinfo1 ( ….)
 PARTITIONED BY  (year int, month int)
… ;
```

Indicate partition info

Define partition names

```
LOAD DATA LOCAL INPATH …… INTO TABLE taxiinfo1
PARTITION (year=2019, month=11);
```

```
truong@aaltosea:/opt/hadoop$ bin/hdfs dfs -ls /user/hive/warehouse/taxiinfo1
Found 4 items
drwxr-xr-x   - truong supergroup          0 2021-03-02 22:37 /user/hive/warehouse/taxiinfo1/year=2017
drwxr-xr-x   - truong supergroup          0 2021-03-02 22:37 /user/hive/warehouse/taxiinfo1/year=2018
drwxr-xr-x   - truong supergroup          0 2021-03-02 22:36 /user/hive/warehouse/taxiinfo1/year=2019
drwxr-xr-x   - truong supergroup          0 2021-03-02 22:33 /user/hive/warehouse/taxiinfo1/year=__HIVE_DEFAULT_PA
truong@aaltosea:/opt/hadoop$ bin/hdfs dfs -ls /user/hive/warehouse/taxiinfo1/year=2019
Found 2 items
drwxr-xr-x   - truong supergroup          0 2021-03-02 22:36 /user/hive/warehouse/taxiinfo1/year=2019/month=11
drwxr-xr-x   - truong supergroup          0 2021-03-02 22:36 /user/hive/warehouse/taxiinfo1/year=2019/month=12
```

Example of buckets

```
CREATE TABLE taxiinfo2 (VendorID int, ….)
 CLUSTERED BY (VendorID) INTO 2 BUCKETS
……;
```

Identify bucket column

```
truong@aaltosea:/opt/hadoop$ bin/hdfs dfs -ls /user/hive/warehouse/taxiinfo2
Found 2 items
-rw-r--r--   1 truong supergroup      66784 2021-03-02 22:54 /user/hive/warehouse/taxiinfo2/000000_0
-rw-r--r--   1 truong supergroup      31339 2021-03-02 22:54 /user/hive/warehouse/taxiinfo2/000001_0
```

Combine partitions with buckets

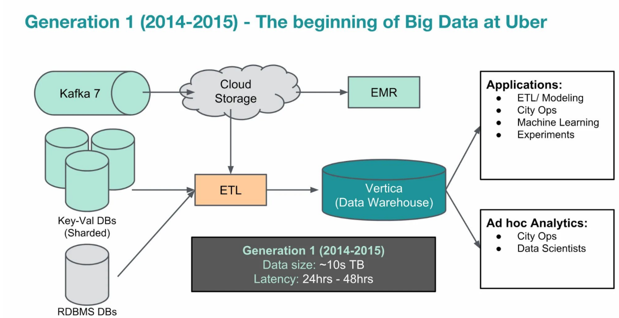
```
truong@aaltosea:/opt/hadoop$ bin/hdfs dfs -ls /user/hive/warehouse/taxiinfo3/year=2019/month=11
Found 2 items
-rw-r--r--   1 truong supergroup      66784 2021-03-02 23:04 /user/hive/warehouse/taxiinfo3/year=2019/month=11/000000_0
-rw-r--r--   1 truong supergroup      31339 2021-03-02 23:04 /user/hive/warehouse/taxiinfo3/year=2019/month=11/000001_0
```

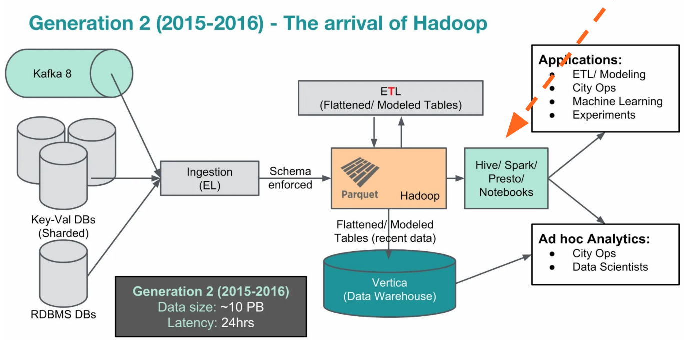
ACID

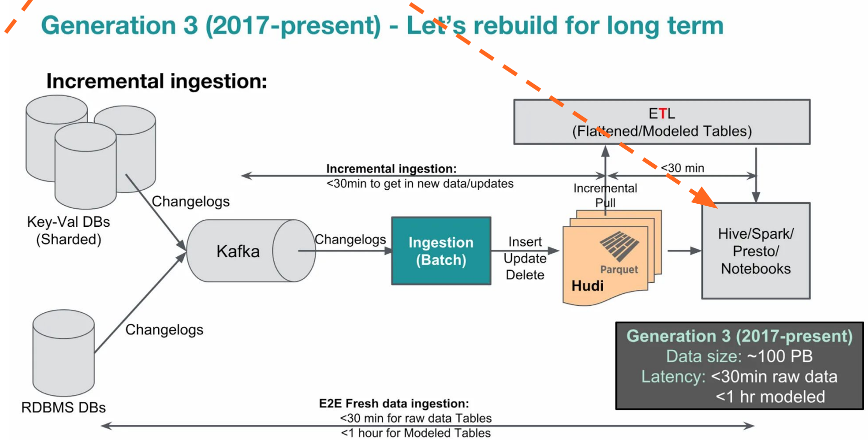
- **Full ACID support**
  - row-level transaction
- **Locks are used for data isolation**
  - shared lock: for concurrent read of tables/partitions
  - exclusive lock: for modifying table/partition

Uber example

Hive for analytics: “extremely large queries”, “ “20,000 Hive queries per day” (see the source)







Figures source:
https://www.uber.com/en-FI/blog/uber-big-data-platform/

Summary

- **Hadoop software ecosystem is very powerful**
  - many applications and use cases have been developed
  - Hadoop File System (HDFS) is a crucial subject
- **Managed Hadoop ecosystem services by cloud providers**
  - try to look at Azure HDInsight, Google Dataproc, and Amazon EMR
- **High-level distributed query engineering using Hadoop components (HDFS, Hive)**
- **Understand the combination of data management with data processing techniques in the same system with Hadoop that simplify your big data tasks**

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io