CARLSON SCHOOL OF MANAGEMENT
UNIVERSITY OF MINNESOTA

Introduction to Hadoop and Ecosystems

MSBA 6320 Prof Liu

Slides credits: Cloudera Academic Partners Program

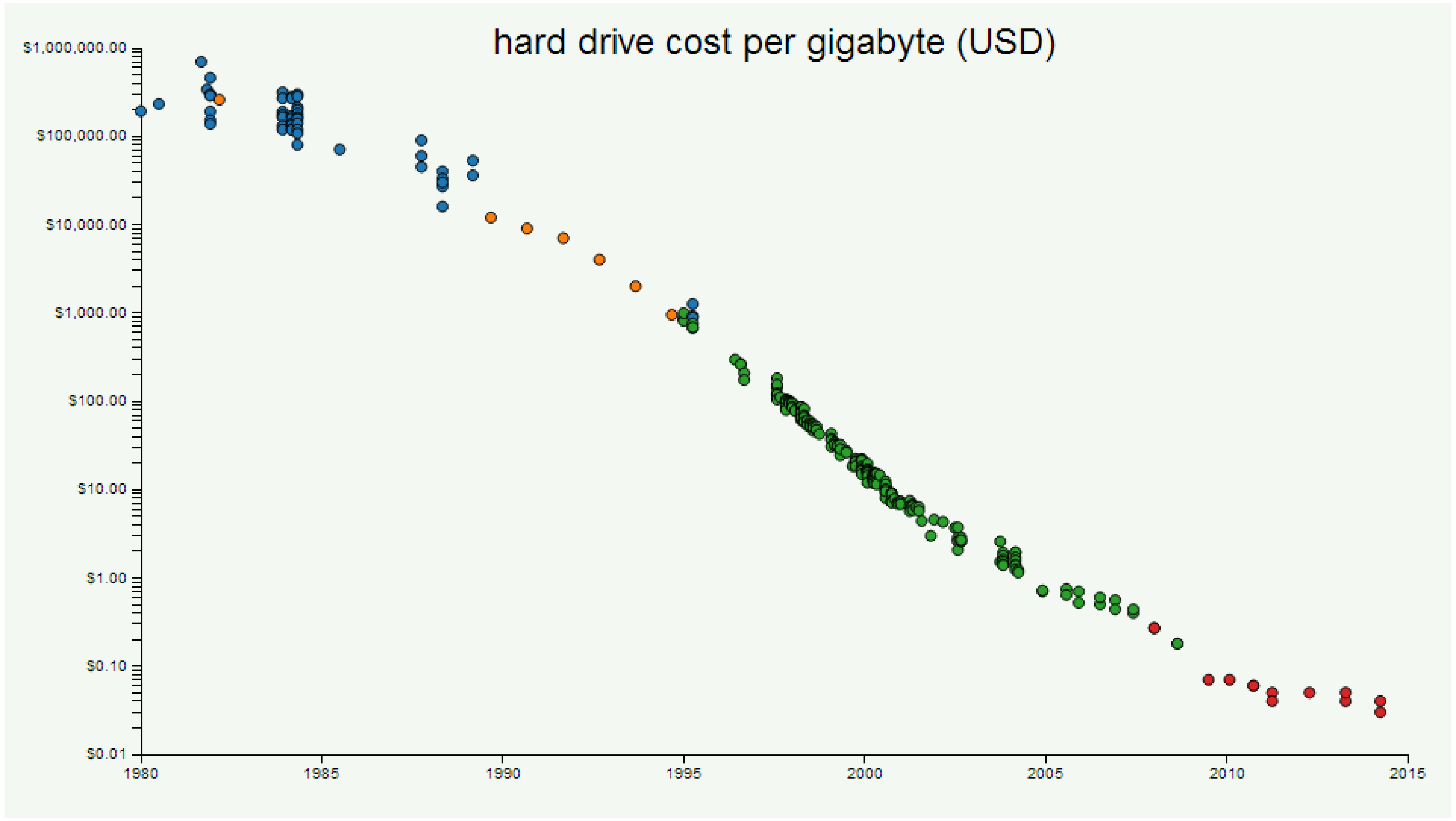
Goals

- The goal of this section is to give you a basic understanding of Hadoop's core components (HDFS and MapReduce) and ecosystem
- In this section, you will learn
  - Which factors led to the era of Big Data
  - What Hadoop is and what significant features it provides
  - How does it offer reliable storage for massive amounts of data with HDFS
  - How does it support large scale data processing through MapReduce
  - What are the new capabilities introduced by YARN?
  - How can Hadoop Ecosystem tools boost an analyst's productivity

Topics

- The Motivation For Hadoop
- Hadoop Overview
- HDFS
- MapReduce
- The Hadoop Ecosystem

Cost of Data Storage



Traditional storing big data is prohibitively expensive

- How has industry typically dealt with these problem?
  - Perform an ETL (Extract, Transform, Load) on the data to summarize the data, before archiving the result in a data warehouse
    - Discarding the details
  - Run queries against the summary data
- Unfortunately, this process often resulted in lost detail
  - But there could be real nuggets in the lost detail
  - More data == Deeper understanding
    - “There’s no data like more data” (Moore 2001)
    - “It’s not who has the best algorithms that wins. It’s who has the most data”

Now we can store data cheaply

- But we're having too much data to process with traditional tools

- Two key problems to address
  - How can we reliably store large amounts of data at a reasonable cost?
    - Hardware failures
  - How can we analyze all the data we have stored?
    - Time needed to read the data into memory for analysis

- Hadoop provides reliable distributed storage, and a general framework for parallel processing at low cost.

Topics

- The Motivation For Hadoop
- Hadoop Overview
- HDFS
- MapReduce
- The Hadoop Ecosystem

What Is Apache Hadoop?

- A system for providing scalable, economical data storage and processing.
- ‘Core’ Hadoop consists of two main components
  - Storage: The Hadoop Distributed File System (HDFS)
    - A framework for distributing data across a cluster in ways that are scalable, reliable, available, fast, and economical
  - Processing: MapReduce
    - A framework for processing data in ways that are scalable, reliable, available, and fast
  - Plus the infrastructure needed to make them work, including
    - Filesystem and administration utilities
    - Job scheduling and monitoring

Where Did Hadoop Come From?

- Started with research that originated at Google back in 2003
  - Google's objective was to index the entire World Wide Web
  - Google had reached the limits of scalability of RDBMS technology
  - This research led to a new approach to store and analyze large quantities of data
    - Google File Systems (Ghemawat, et al 2003)
    - MapReduce: Simplified Data Processing on Large Clusters (Dean and Ghemawat 2004)
- A developer by the name of Doug Cutting was wrestling with many of the same problems in the implementation of his own open-source search engine, Lucene
  - He started an open-source project based on Google’s research and created Hadoop in 2005.
  - Hadoop was named after his son’s toy elephant.

More @ White 2015: A brief history of Apache Hadoop, pp12-14

Hadoop Cluster and Scalability

- Hadoop is a distributed system
  - A collection of servers running Hadoop software is called a **cluster**
- Individual servers within a cluster are called **nodes**
  - Typically standard rack-mount servers running Linux
  - Each node both stores and processes data
    - Called "data locality"
- Add more nodes to the cluster to increase scalability
  - Facebook and Yahoo are each running clusters in excess of 4400 nodes
  - Scalability is linear
  - Horizontal scaling simplifies capacity planning as well
    - **Horizontal scaling**: scale by adding more machines to the pool of resources
    - **Vertical scaling**: scale by adding more power CPU/RAM to an existing machine

Powerful Hadoop clusters can be built from cheap commodity hardware, resulting in lowered cost!

Fault Tolerance

- Paradox: adding nodes increases chances that any one of them will fail
  - Solution: build redundancy into the system and handle it automatically.
- Files loaded into HDFS are replicated across nodes in the cluster
  - If a node fails, its data is re-replicated using one of the other copies
- Data processing jobs are broken into individual tasks
  - Each task takes a small amount of data as input
  - Thousands of tasks (or more) often run in parallel
  - If a node fails during processing, its tasks are rescheduled elsewhere
- Routine failures are handled automatically without any loss of data
  - Developers/Users do not need to worry about hardware failures

Review Question

- Advantages of a Hadoop system

Topics

- The Motivation For Hadoop
- Hadoop Overview
- HDFS
- MapReduce
- The Hadoop Ecosystem

HDFS: Hadoop Distributed File System

- Provides inexpensive and reliable storage for massive amounts of data
  - Optimized for a relatively small number of large files
    - Each file likely to exceed 100 MB, multi-gigabyte files are common
  - Store file in hierarchical directory structure
    - e.g. , `/sales/reports/asia.txt`
  - Cannot modify files once written
    - Need to make changes? remove and recreate
  - Use Hadoop specific utilities to access HDFS

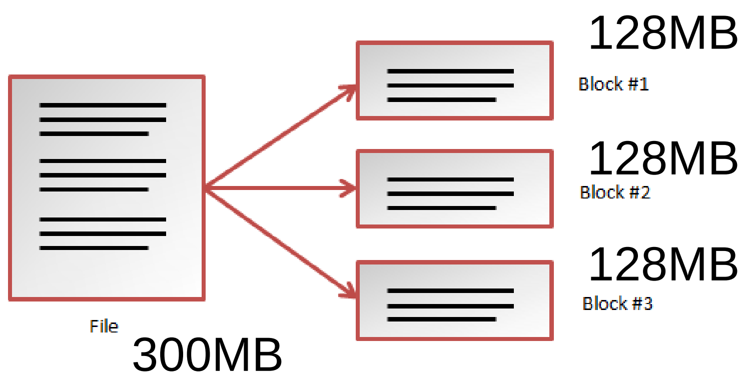
HDFS and Unix File System*

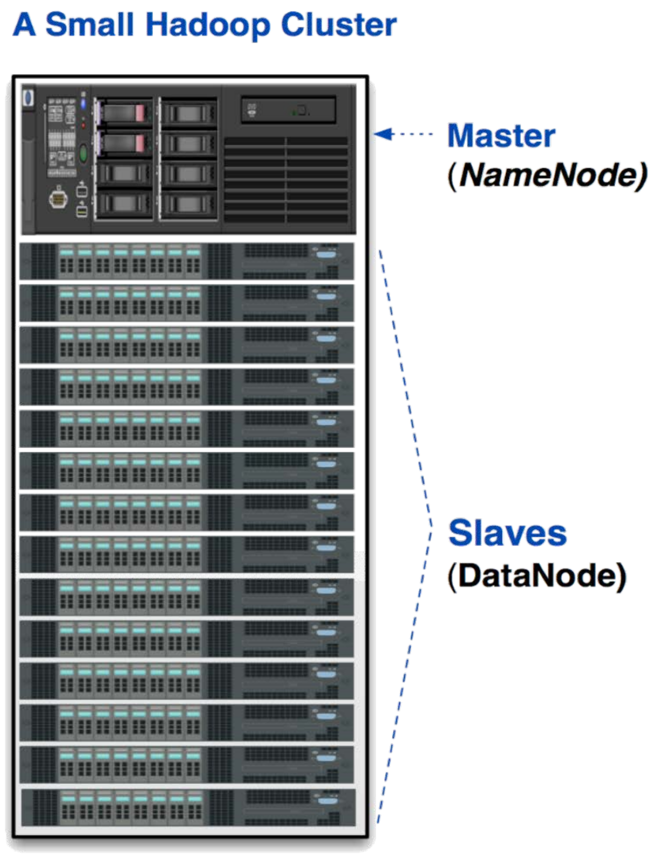
- In some ways, HDFS is similar to a UNIX filesystem
  - Hierarchical, with UNIX/style paths (e.g. `/sales/reports/asia.txt`)
  - UNIX/style file ownership and permissions
- There are also some major deviations from UNIX
  - No concept of a current directory
  - Cannot modify files once written
    - You can delete them and recreate them, but you can't modify them
  - Must use Hadoop specific utilities or custom code to access HDFS

HDFS Architecture (1 Of 3)

- **Blocks**: HDFS files are broken into blocks, a smallest unit for read and write. HDFS uses128 MB blocks by default (Windows default block size is 4 KB).
  - Each block is replicated multiple times and stored in different nodes of the cluster

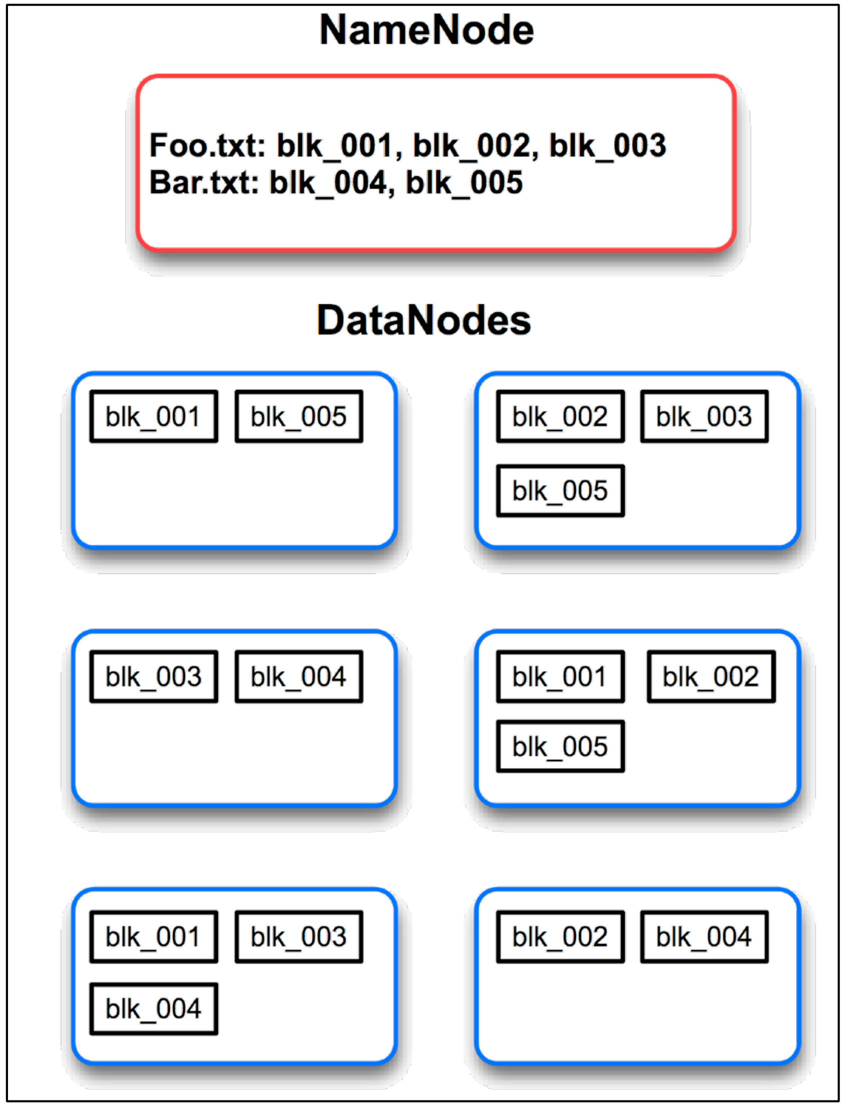
- An HDFS cluster has a master/slave architecture
- HDFS NameNode (one or two)
  - Manages namespace (file to block mappings) and metadata (block to machine mappings).
  - Monitors dataNodes
- HDFS DataNodes (many)
  - Reads and writes the actual data





HDFS Architecture (2 Of 3)

- Example:
  - The NameNode holds metadata for the two files
    - Foo.txt (300MB) and Bar.txt (200MB)
    - Assume HDFS is configured for 128MB blocks
  - The DataNodes hold the actual blocks
    - Each block is 128MB in size
    - Each block is replicated three times on the cluster
    - Block reports are periodically sent to the NameNode

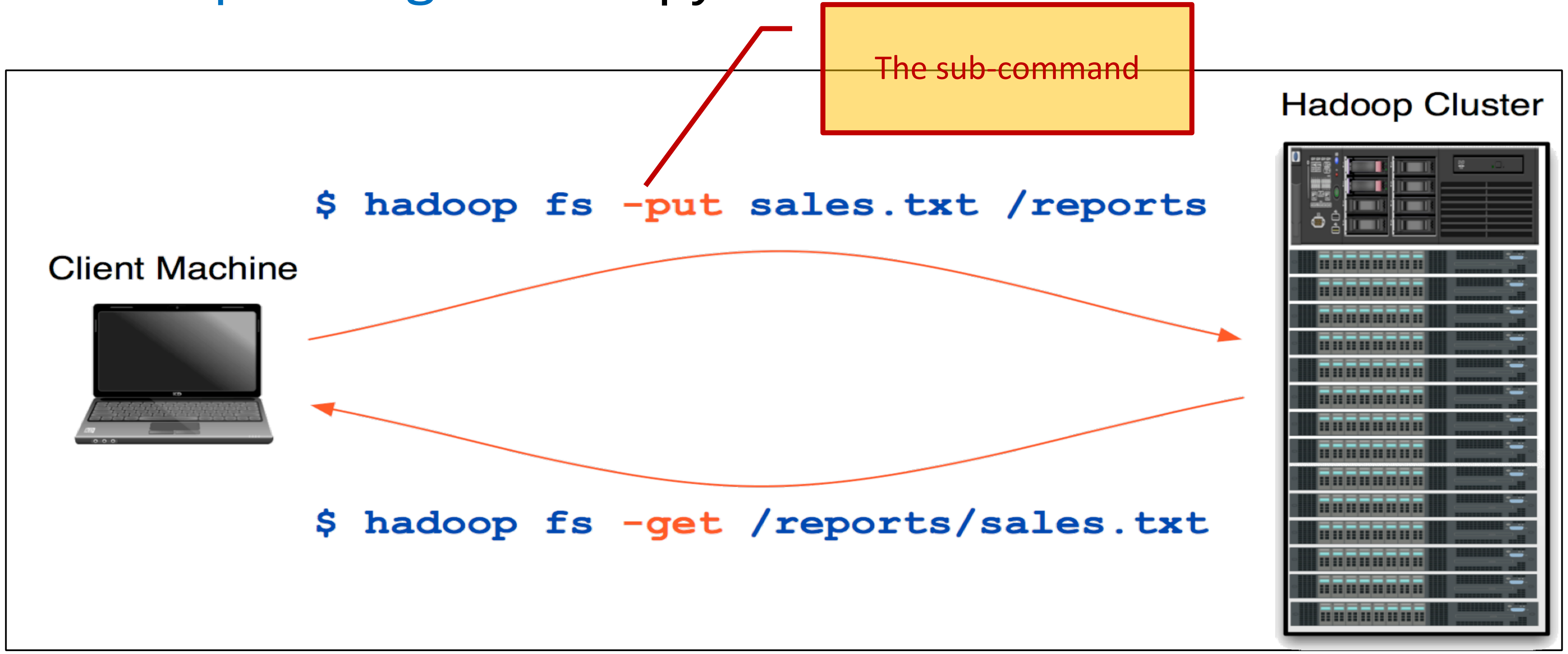


HDFS Architecture (3 Of 3)

- Optimal for handling millions of large files, rather than billions of small files, because:
  - In pursuit of responsiveness, the NameNode stores all of its file/block information in memory
    - A rule of thumb is each block/directory/file takes about 150 bytes
    - A name node can handle millions of objects, but not billions
    - Too many files will cause the NameNode to run out of memory
    - Too many blocks (if the blocks are small) will also cause the NameNode to run out of memory
  - A Java Virtual Machine (JVM) is required to process a block; if you have too many blocks, you will need many JVMs at the same time, and you will begin to see the limits of HDFS scalability.

Copying Local Data To And From HDFS

- Remember that HDFS is separated from your local filesystem
  - Use `hadoop fs -put` to copy local files to HDFS
  - Use `hadoop fs -get` to copy HDFS files to local files



```
$ hadoop fs -put sales.txt /reports
```

```
$ hadoop fs -get /reports/sales.txt
```

Review Question

- HDFS Design

Topics

- The Motivation For Hadoop
- Hadoop Overview
- HDFS
- MapReduce
- The Hadoop Ecosystem

Introducing MapReduce

- Now that we have described how Hadoop stores data, lets turn our attention to how it processes data
- We typically process data in Hadoop using MapReduce
- MapReduce is not a language, it's a programming model
- MapReduce consists of two functions: map and reduce
- Before MapReduce, coordinating the processes in a large-scale distributed computation is a challenge.
  - MapReduce spare the programmer from having to think about failure, since the implementation detects failed tasks and reschedules replacements on machines that are healthy.
  - MapReduce follows **shared-nothing** architecture, meaning that tasks **have no dependence** on one other. MapReduce programmers need not worry about order in which tasks run. By contrast, other distributed computing models often require programmers to explicitly manage their own checkpointing and recovery.

Map function

- The map function always runs first
  - Each mapper takes <key, value> pairs as input, one pair at a time
  - Many mappers run in parallel to each other, each assigned to a chunk of the input data.
  - For each input pair, a mapper emits a list of output pairs.
    - `(key, value) → list(key, value)`
    - `E.g. "fox jumps over" --> ("fox",1),("jumps",1),("over",1)`
    - `"fox jumps over" --> ("FOX JUMPS OVER")`
  - Typically used to “break down”
    - Filter, transform, or parse data,
    - e.g. Parse the stock symbol, price and time from a data feed
    - Break a sentence into words
  - The output from the map function (eventually) becomes the input to the reduce function

Reduce function

- The reduce function takes a list of values for every key, and transforms the data on the (aggregation) logic provided in the reduce function
  - It takes a <key, value-list> pair as input, and emits a list of <key, value> pairs as output

    ```
    (Key, List of values for the key) --> List(Key, Value)
    ```
  - `E.g. ("jumps",(1,1,2,1,1)) --> ("jumps",6)`
  - Each job may have multiple reducers, each responsible for a specific key range.
  - Typically used to aggregate data from the map function
    - e.g. Compute the average hourly price of the stock
  - A reduce step is optional
    - You can run something called a “map-only” job

- Between map and reduce, there is typically a hidden phase known as the “Shuffle and Sort” which organizes map outputs for delivery to the reducer.

A MapReduce Example

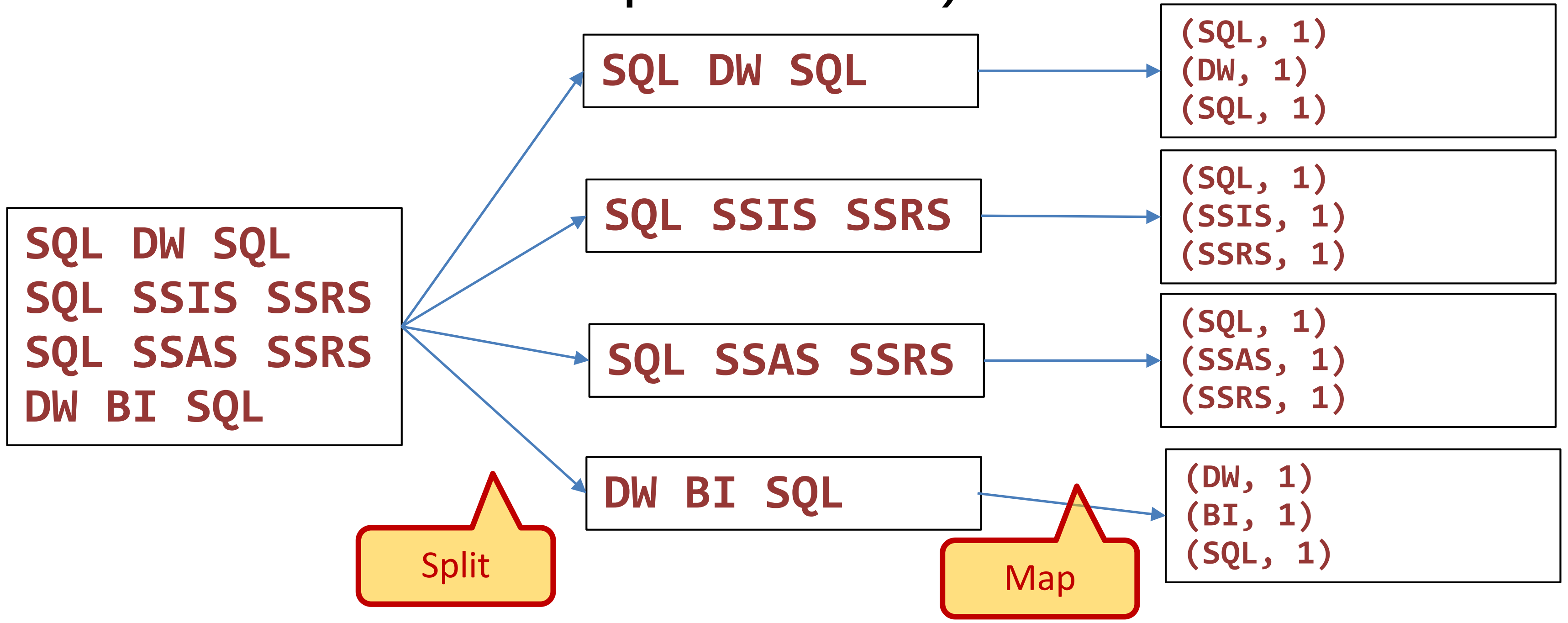
- The following slides will explain an entire MapReduce job
  - Input: a text document (a list of sentences)
  - Output: word counts

```
SQL DW SQL
SQL SSIS SSRS
SQL SSAS SSRS
DW BI SQL
```

```
BI, 1
DW, 2
SQL, 5
SSAS, 1
SSIS, 1
SSRS, 2
```

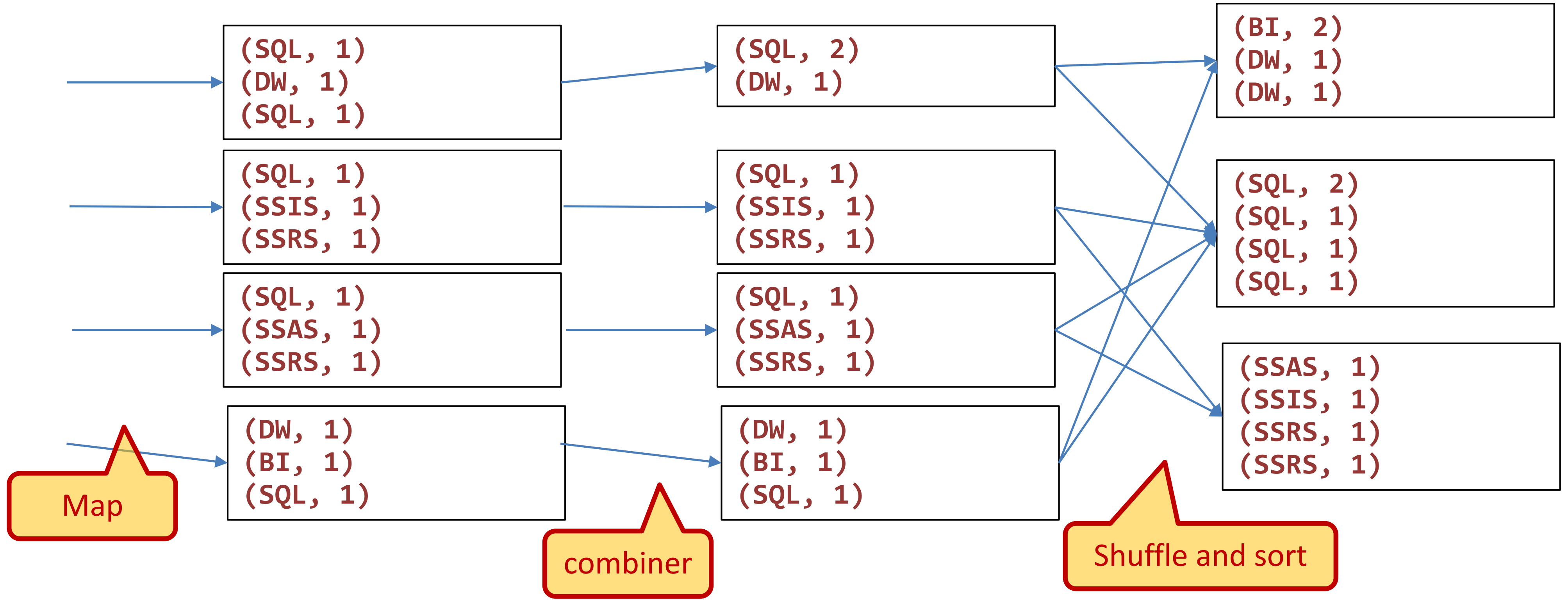
A MapReduce Example – Split and Map

- Hadoop splits a job into many individual map tasks
  - The number of map tasks is determined by the amount of input data
  - Each map task receives a portion of the overall job input to process in parallel
  - In this case, there are 4 mappers, each processing one chunk (in practice, each chunk includes multiple records)



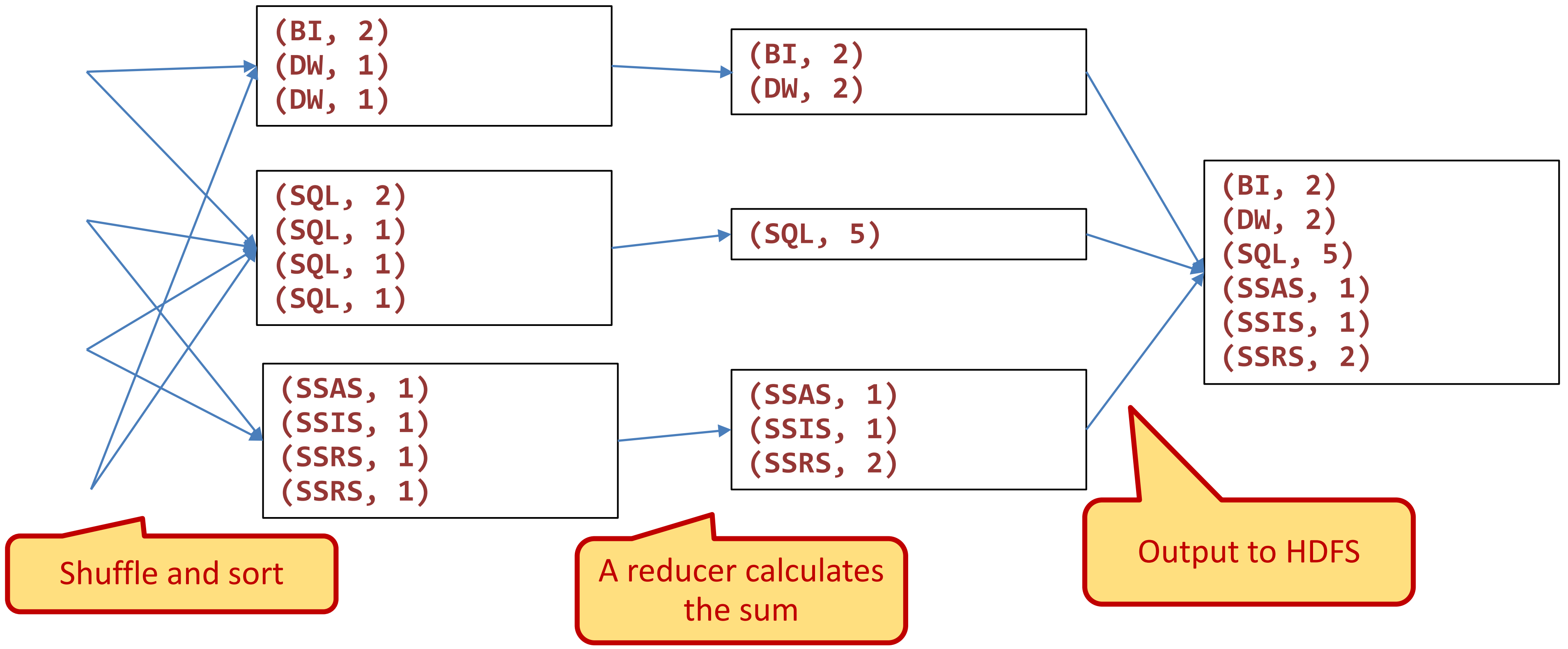
A MapReduce Example – Shuffle & Sort

- Map outputs are shuffled and sorted by key
  - Sometimes a combiner is run to before shuffle and sort to reduce the amount of data transferred across the network
    - There may not be a combiner for certain aggregate operations
    - E.g. there are combiners for sum and max. but there is no combiner for average.
  - Shuffle and sort is implicit and carried out by the MapReduce framework.
  - It results are feed into reducers (we have three in this case)

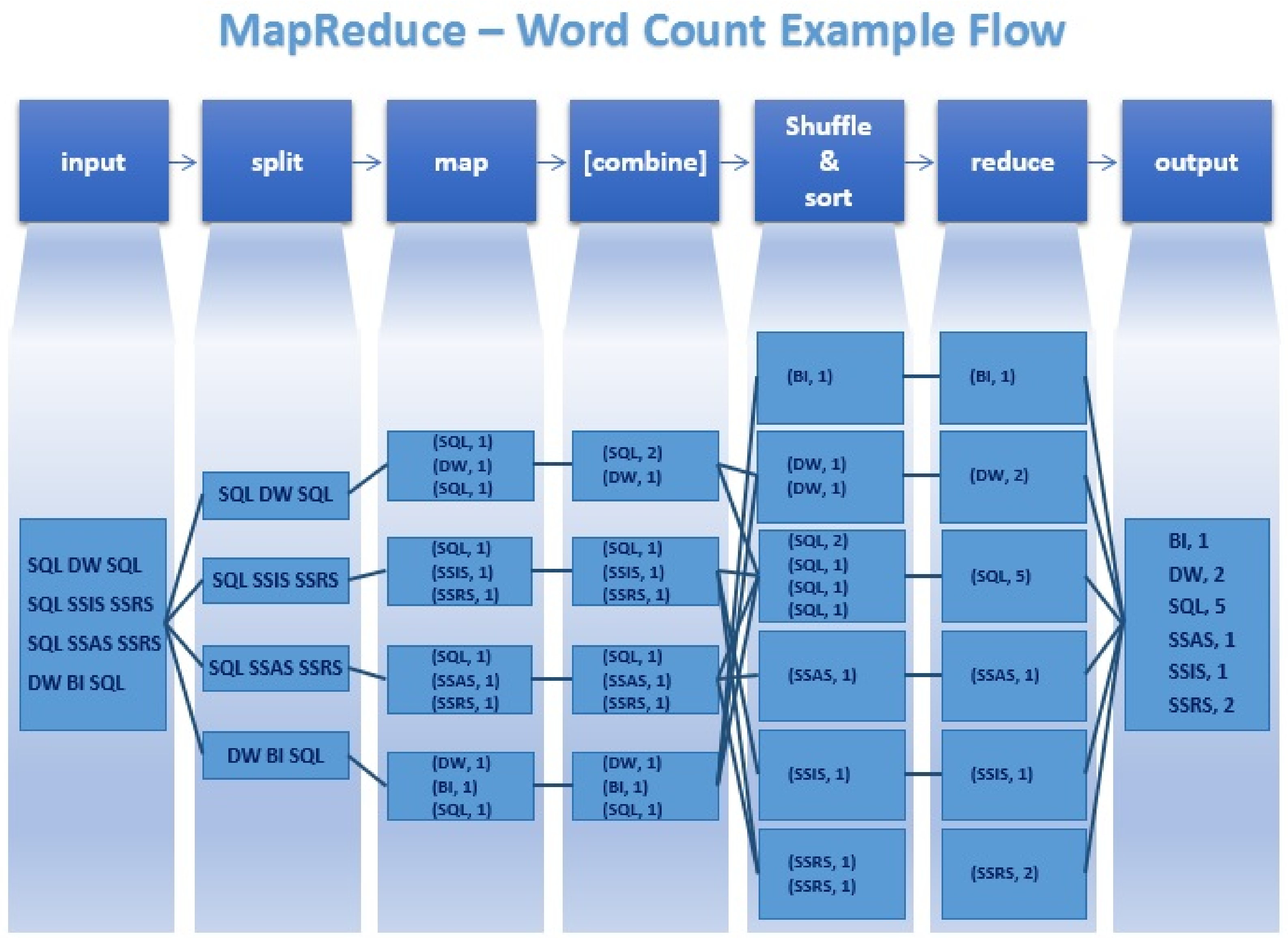


A MapReduce Example – Reduce

- Reducer input comes from the shuffle and sort process
  - All value for the same key is collected and feed into a reducer
  - The reducers aggregates the values and emit a key value pair
  - Each reducer output file is written to a job specific directory in HDFS



Putting It All Together



Note: we used 6 reducers here, but 3 in the step-wise example

Group Exercise: MapReduce

- In groups of 3-4 (preferably from different organizations)
  - Work out the following example using pence & paper.
- **Input**: a file consists of order ID, employee name, and sales amount, separated by spaces

```
0 Alice 5
1 Bob 4
2 Alice 3
3 Alice 9
4 Diana 1
5 Bob 9
```

Hands On

Paper & pencil approach to MapReduce

- **Desired output**: sum of sales by employees.
  - Following the word count example to draw the steps of a MapReduce work flow (assuming two mappers/reducers)

Benefits of MapReduce

- Simplicity (via fault tolerance)
  - Particularly when compared with other distributed programming models
- Flexibility
  - Offers more analytic capabilities and works with more data types than platforms like SQL (key, value pairs can accommodate different kinds of structured and unstructured data)
- Scalability
  - Because it works with
    - Small quantities of data at a time
    - Running in parallel across a cluster
    - Sharing nothing among the participating nodes

Review Question

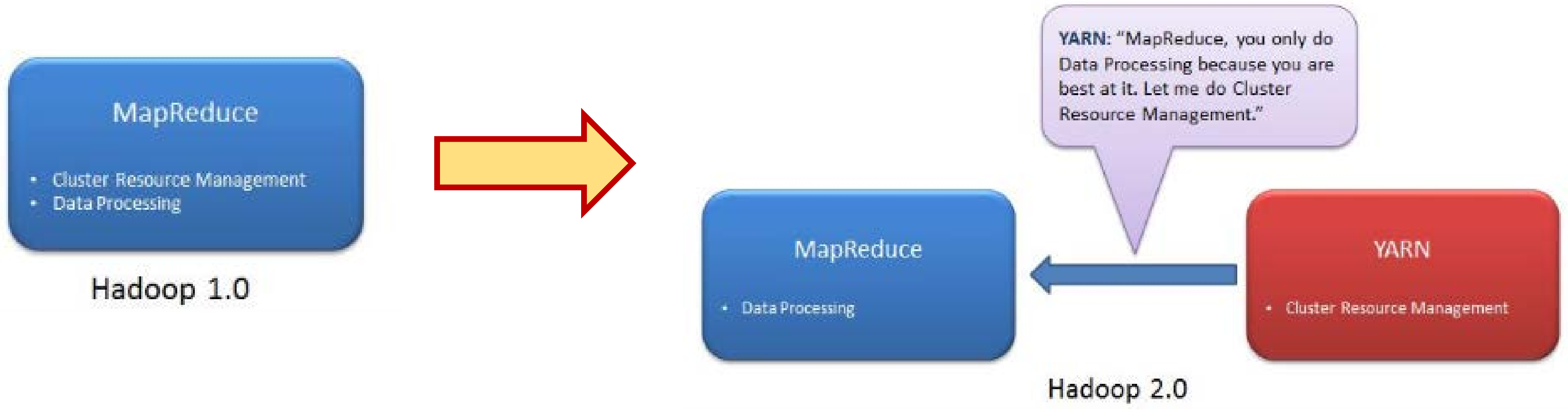
- The role of MapReduce ★
- The Benefits of MapReduce ★

Hadoop 2.0

- So far we have focused original ideas of Hadoop
  - It has batch processing in mind
  - It supports only MapReduce applications, which has too high a latency for interactive and streaming applications.
- Since then, many improvements have been made to the Hadoop, including
  - YARN (**Y**et **A**nother **R**esource **N**egotiator)
  - The ability to run non MapReduce applications (e.g. support Spark)
  - Beyond batch processing: the ability to support interactive and streaming applications.

YARN

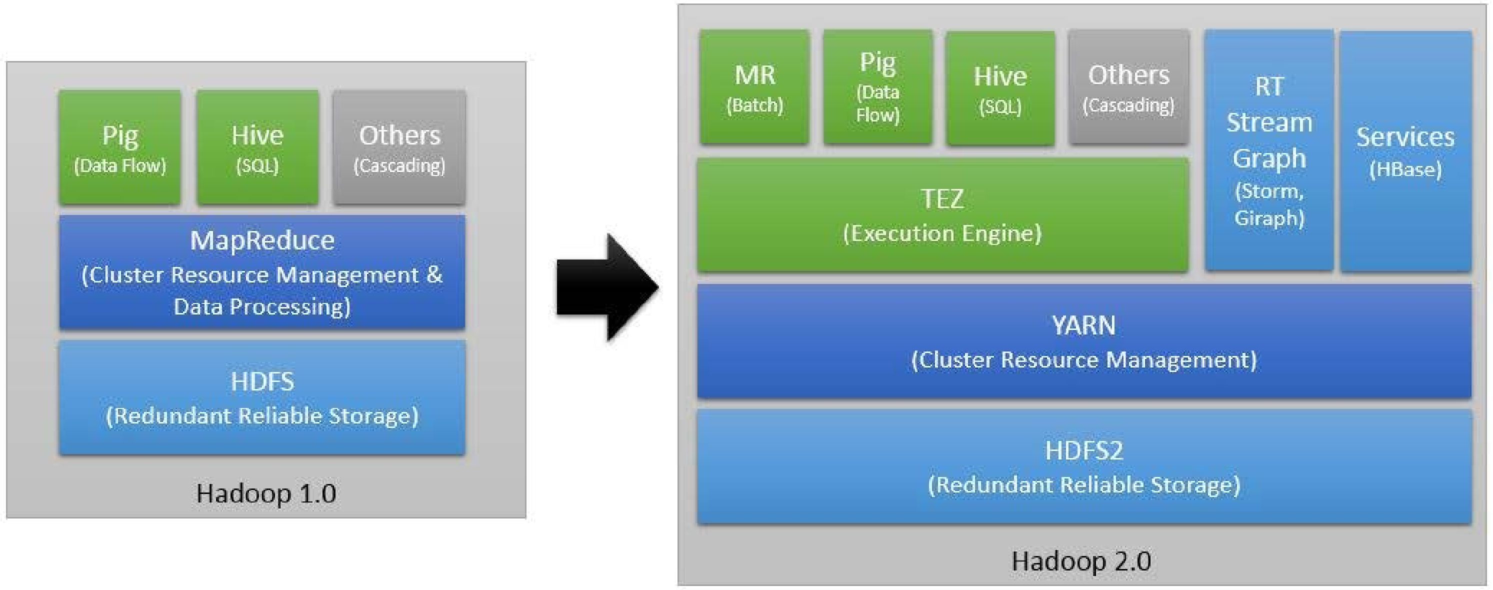
- YARN (**Y**et **A**nother **R**esource **N**egotiator) is a new component added in Hadoop 2.0
  - In Hadoop 1.0, MapReduce does both distributed computing and resource management (via jobtracker)
    - Limits availability, scalability, resource utilization, and running non-MapReduce applications. If the job tracker fails, all jobs must restart.
  - YARN takes over the resource management (CPU, memory etc) and job scheduling, allows multiple kinds of processing to run on a single Hadoop cluster (e.g. batch processing, interactive application, streaming)


MapReduce
• Cluster Resource Management
• Data Processing
Hadoop 1.0
YARN: "MapReduce, you only do Data Processing because you are best at it. Let me do Cluster Resource Management."
MapReduce
• Data Processing
YARN
• Cluster Resource Management
Hadoop 2.0

Read more: http://saphanatutorial.com/how-yarn-overcomes-mapreduce-limitations-in-hadoop-2-0/

Hadoop 2.0 Architecture

- Separate MapReduce into a YARN layer and MR (for batch processing)
- Support non-MapReduce applications, e.g. streaming and interactive applications (e.g. Impala, Spark, Storm, HBase, Giraph etc)
- Improved nameNode availability



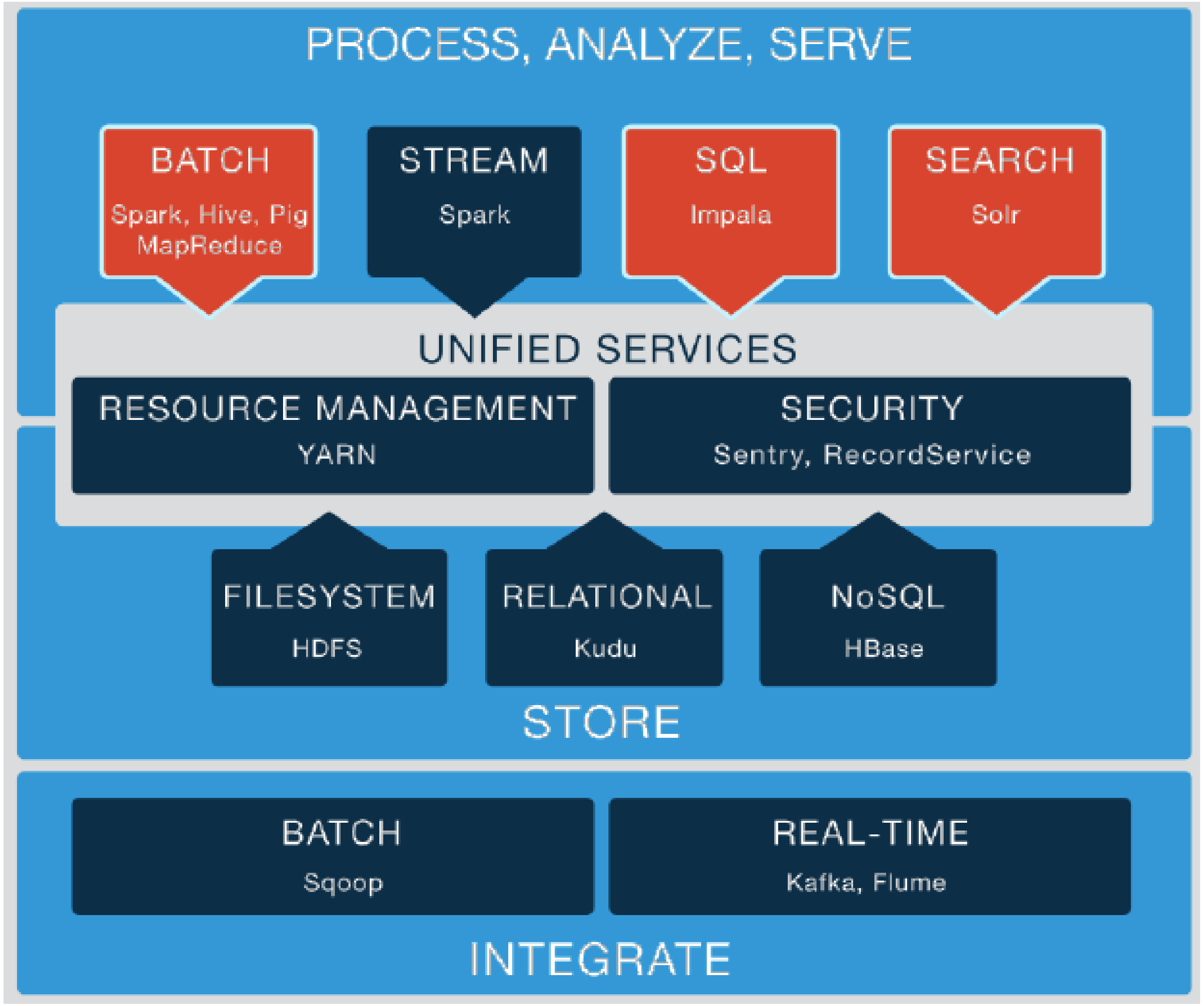
Read more: http://saphanatutorial.com/hadoop-1-0-vs-hadoop-2-0/

Review Qustion

- Hadoop 2.0 improvements

Topics

- The Motivation For Hadoop
- Hadoop Overview
- HDFS
- MapReduce
- The Hadoop Ecosystem


PROCESS, ANALYZE, SERVE
BATCH
Spark, Hive, Pig MapReduce
STREAM
Spark
SQL
Impala
SEARCH
Solr
UNIFIED SERVICES
RESOURCE MANAGEMENT
YARN
SECURITY
Sentry, RecordService
FILESYSTEM
HDFS
RELATIONAL
Kudu
NoSQL
HBase
STORE
BATCH
Sqoop
REAL-TIME
Kafka, Flume
INTEGRATE

Data Storage

- **Hadoop Distributed File System (HDFS)**
  - HDFS is the storage layer for Hadoop
  - Provides inexpensive reliable storage for massive amounts of data on commodity hardware
  - Data is distributed when stored
- **Apache HBase**
  - A NoSQL distributed database built on HDFS
  - Scales to support very large amounts of data
  - High throughput
  - A table can have many thousands of columns & billions of rows
  - No high-level query language, API access only.

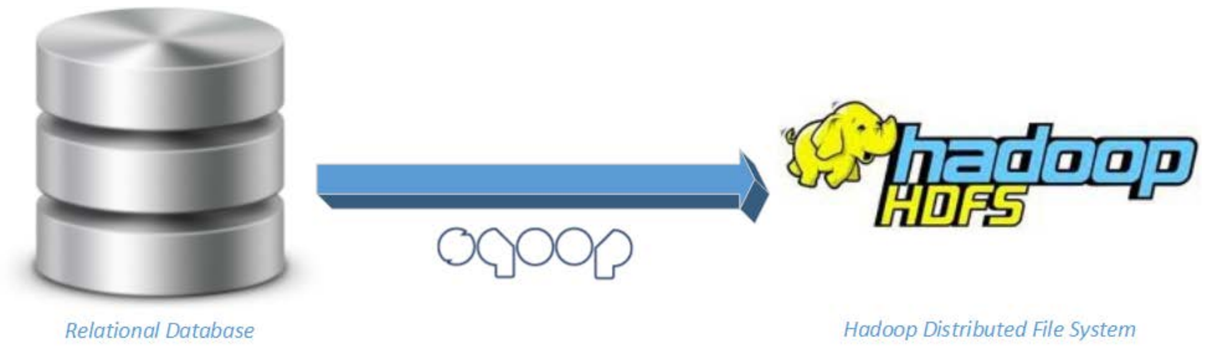
Apache Kudu

- Distributed columnar (key-value) storage for structured data
- Allows random access and updating data (unlike HDFS)
- Supports **SQL**-based analytics
- Works directly on native file system; is not built on HDFS
- Integrates with Spark, MapReduce, and Apache Impala
- Created at Cloudera, donated to Apache Software Foundation

Data Integration Tools (1)

- **HDFS**
  - Direct file transfer

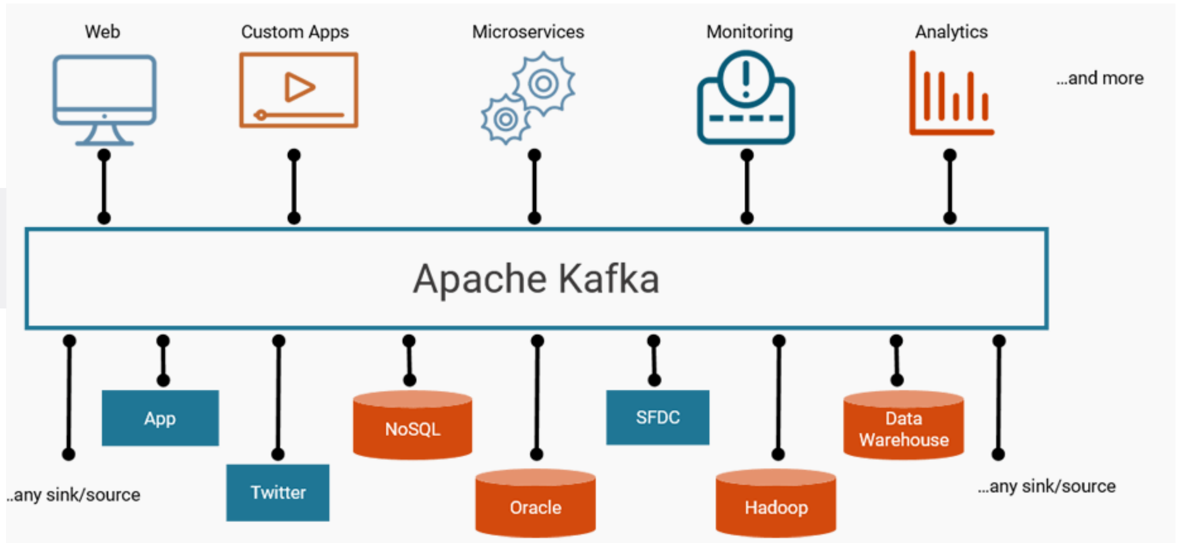
- **Apache Sqoop**
  - High speed import for HDFS from RDBMS (and vice versa)
  - Supports many RDBMS:
    - E.g. Oracle, Teradata, MySQL, MongoDB, Netezza

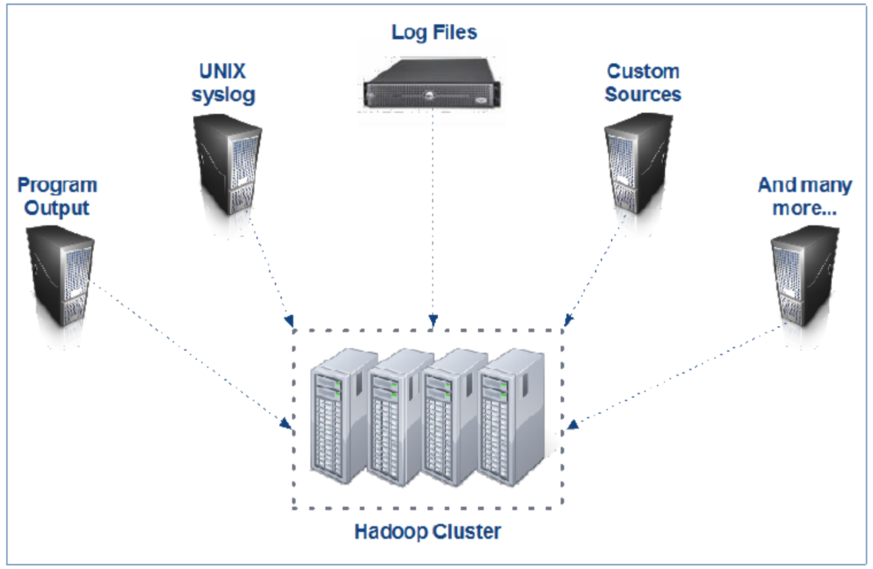


Data Integration Tools (2)

- **Apache Kafka**
  - A high throughput, scalable **messaging** system
  - Distributed, reliable publish-subscribe system

https://kafka.apache.org/powered-by





- **Apache Flume**
  - Distributed service for ingesting streaming data
  - Ideally suited for **event data** from multiple systems

Batch Processing Tools (1 of 4)

- **Hadoop MapReduce** is the original Hadoop framework
  - Primarily Java based
  - Was the core Hadoop processing engine
  - Has mature fault tolerance built into the framework
  - But quickly losing ground to Spark and other engines

```
map(String name, String contents):
    for each word w in contents:
        if (IsCapitalized(w)):
            uppercase->Increment();
        EmitIntermediate(w, "1");
```

Batch Processing Tools (2 of 4)

- **Apache Pig** builds on Hadoop to offer high-level data processing
  – An alternative to write low-level MapReduce code
  – Useful for ETL (extract, transformation, and loading)
  – Used by developers and analysts
- Pig Interpreter
  – Turns Pig Latin scripts into MapReduce (or Spark) Jobs
  – Submits those jobs to a Hadoop cluster

```
people = LOAD '/user/training/customers' AS (cust_id, name);
orders = LOAD '/user/training/orders' AS (ord_id, cust_id, cost);
groups = GROUP orders BY cust_id;
totals = FOREACH groups GENERATE group, SUM(orders.cost) AS t;
result = JOIN totals BY group, people BY cust_id;
DUMP result;
```

Batch Processing Tools (3 of 4)

- **Apache Hive** is the data warehouse application for Hadoop.
  - Uses a SQL-like language called HiveQL
  - Useful for managing and querying large tables in Hadoop
  - Translate SQL to MapReduce jobs

```
 SELECT  zipcode, SUM(cost) AS total
    FROM  customers
    JOIN  orders
      ON  customers.cust_id = orders.cust_id
   WHERE  zipcode LIKE '63'
GROUP BY  zipcode
ORDER BY  total DESC;
```

Batch Processing Tools (4 of 4)

- **Spark:** Spark is a fast large-scale in-memory processing engine
  - Interface with YARN clusters and HDFS, and many other storage options
  - Highly popular, is overtaking Hadoop
  - Has core spark and four components
    - SQL, Streaming, Mllib, GraphX

- **Spark SQL** – for SQL operations; not a SQL engine but flexible for SQL based data munging.

Interactive Query Tools (1 of 2)

- **Apache Impala** is a high-performance, interactive SQL engine
  - Very low latency – measured in milliseconds
  - Run on Hadoop clusters
  - Supports a dialect of SQL (Impala SQL)

```
 SELECT  zipcode, SUM(cost) AS total
   FROM  customers
   JOIN  orders
     ON  customers.cust_id = orders.cust_id
  WHERE  zipcode LIKE '63'
GROUP BY  zipcode
ORDER BY  total DESC;
```

Interactive Query Tools (2 of 2)

- **Apache Drill** is a high performance SQL engine that can query a variety of structured, semi-structured data formats
  - Uses Standard ANSI SQL
  - Query semi-structured data formats and sources, e.g. HDFS, HBase, JSON, MongoDB, Amazon S3.
  - High performance, scalable.

```
SELECT * FROM dfs.root.`/web/logs`;

SELECT country, count(*)
  FROM mongodb.web.users
  GROUP BY country;

SELECT timestamp
  FROM s3.root.`clicks.json`
  WHERE user_id = 'jdoe';
```

Machine Learning

- **Apache Mahout** is a machine learning (ML) software for Hadoop
  - It provides a host of pre-made ML algorithms
  - Recently undergone big changes to provide a R-like programming environment
- **Apache Spark MLlib**: a Machine Learning component of spark
- **H2O**: in-memory, distributed, fast, and scalable machine learning platform; can be used from R, Python, Scala.
- **H2O/Sparkling Water**: Combine H2O with Spark

Streaming

- **Apache Storm** is a distributed and fault-tolerance real-time computation platform
  - Horizontal scalability
  - Guaranteed data processing
  - Fault-tolerance
  - High-performance
- **Spark Streaming**: the streaming component of Apache Spark

Other Hadoop Ecosystem Tools

- **Apache Sentry**: Provides fine-grained access control for varous Hadoop ecosystem components.

- **Apache Zookeeper**: a distributed hierarchical key-value store for providing a distributed configuration service, synchronization service, and naming registry for large distributed systems.

Zookeeper

- **Apache Ooozie**: A workflow engine that ensures jobs are submitted in the correct sequence.

The Hadoop Ecosystem

Data ingestion/integration:

Storage:

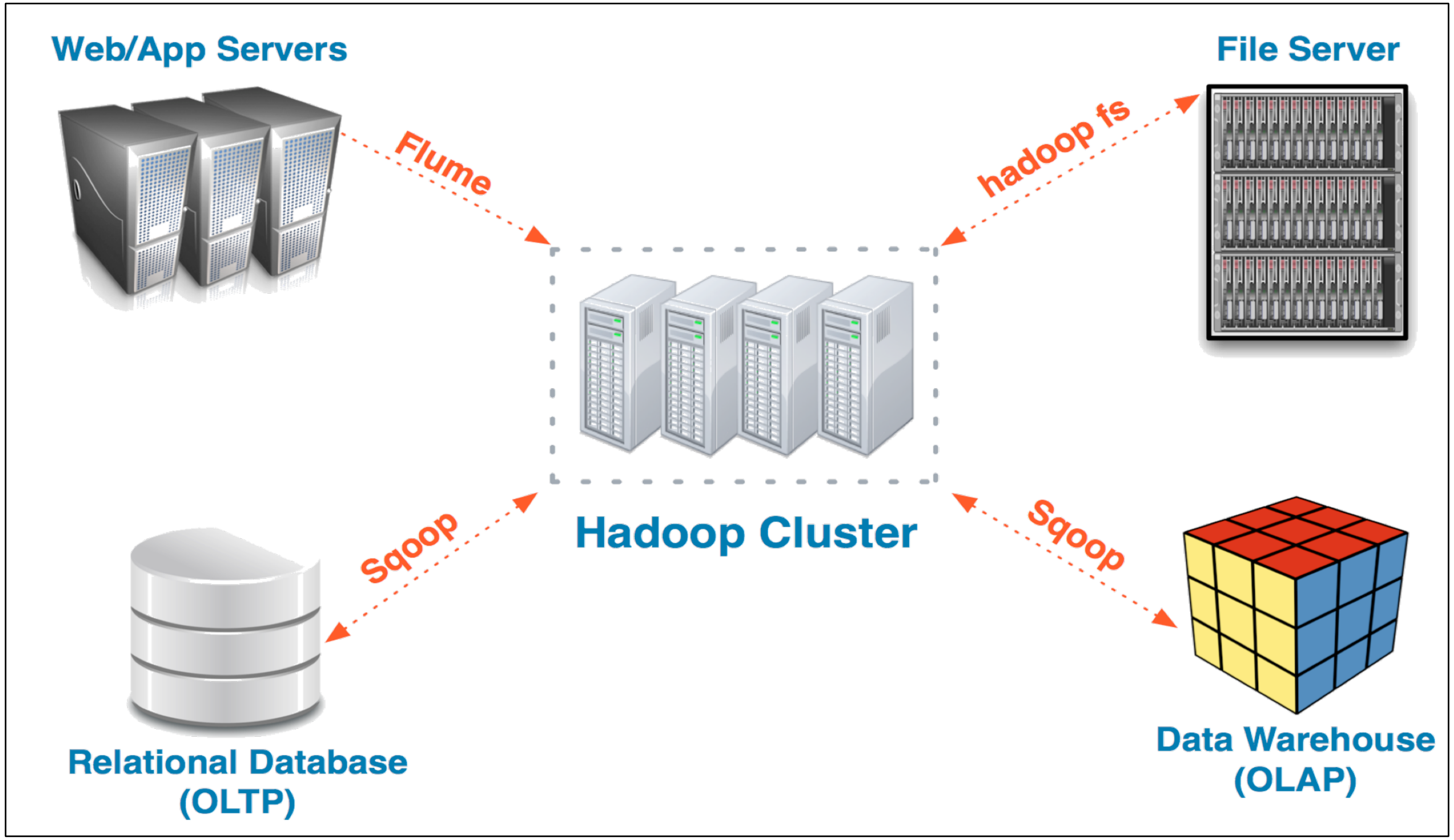
Batch processing:

Interactive Processing: Impala, Drill

Machine Learning:

Streaming processing:

Data Center Integration



Hadoop & Data Lake

- A **Data Lake** approach
  - The traditional **Data Mart /Data Warehouse** is a "*store of bottled water*"
  - The data lake is a "large body of water in a more nature state" (James Dixon, CTO of Pentaho)
  - Contents of the data lake stream in from a source to fill the lake, and various uses can examine, dive in, or take samples.
- **Hadoop** is an ideal platform for building a data lake
  - It provides for storing a mixture of structured and unstructured data, side by side.
  - It could be used to store far more detail about the transaction than you would in an ordinary RDBMS (due to costs limitations)

Source: data science central
Read more: Hester, L. 2016. "Maximizing Data Value with a Data Lake," *Data Science Central Blog*.

Review Question

- Hadoop ecosystem tools ☆

Bibliography

- The following offer more information on topics discussed in this section
  - Hadoop: The Definitive Guide, 4th Edition (**U of M Students have free access**)
    - https://goo.gl/4kQVSy
  - Guide to HDFS Commands
    - http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/FileSystemShell.html
  - Cloudera Version and Packaging Information (installed on our VM)
    - https://www.cloudera.com/documentation/enterprise/release-notes/topics/cdh_vd_cdh_package_tarball_510.html#tarball_510x

Bibliography

- Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. 2006. "Bigtable: A Distributed Storage System for Structured Data," *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*, pp. 205–218.
- Davenport, T. H., and Dyché, J. 2013. "Big Data in Big Companies," *International Institute for Analytics* (May), pp. 1–31.
- Dean, J., and Ghemawat, S. 2004. "MapReduce: Simplified Data Processing on Large Clusters," *Proceedings of 6th Symposium on Operating Systems Design and Implementation*, pp. 137–149.
- Ghemawat, S., Gobioff, H., and Leung, S.-T. 2003. "The Google File System," *ACM SIGOPS Operating Systems Review* (37:5), p. 29.
- Hester, L. 2016. "Maximizing Data Value with a Data Lake," *Data Science Central Blog*.
- Michael Schroeck, Shockley, R., Smart, J., Romero-Morales, D., and Tufano, P. 2012. "Analytics : The Real-World Use of Big Data," *IBM Report*.
- NewVantage. 2016. "Big Data Executive Survey 2016," *NewVantage Partner Report*.
- Tableau. 2016. "Top Ten Big Data Trends for 2017," *Tableau Report*.
- White, T. 2012. "A Brief History of Apache Hadoop," in *Hadoop: The Definitive Guide*, pp. 12–14.