Hadoop, a distributed framework for Big Data

By: Zahra Rezaei
z.rezaei2010@gmail.com

Outline

- Hadoop - Basics
- HDFS
- Yarn
- MapReduce
- Spark
- Related Apache sub-projects (Pig, HBase,Hive)

Apache Hadoop

- A framework for storing & processing Petabyte of data using commodity hardware and storage

- Apache project
- Implemented in Java
- Community of contributors is growing
  - Yahoo: HDFS and MapReduce
  - Powerset: HBase
  - Facebook: Hive and FairShare scheduler
  - IBM: Eclipse plugins

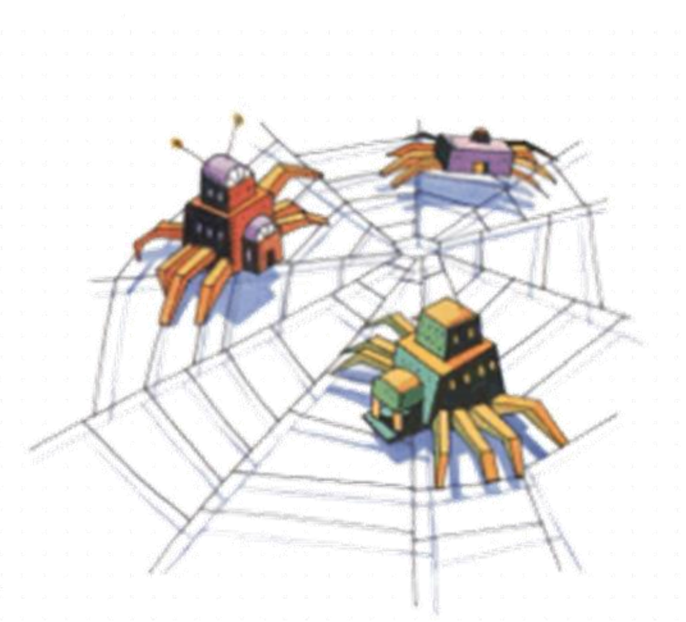
What is Hadoope?

- Software platform that lets one easily write and run applications that process vast amounts of data. It includes:
- – MapReduce – offline computing engine
- – HDFS – Hadoop distributed file system
- – HBase (pre-alpha) – online data access
- Yahoo! is the biggest contributor

- Here's what makes it especially useful:
- Scalable: process petabytes.
- Economical: processing across clusters (in thousands).
- Efficient: By distributing the data, it can process it in parallel on the nodes where the data is located.
- Reliable: It automatically maintains multiple copies of data and automatically redeploys computing tasks based on failures.

- Apache top level project, open-source implementation of frameworks for reliable, scalable, distributed computing and data storage.
- It is a flexible and highly-available architecture for large scale computation and data processing on a network of commodity hardware.

- **Designed to answer the question: "How to process big data with reasonable cost and time?"**



History of Hadoop

- Started as a sub-project of Apache Nutch
- Nutch's job is to index the web and expose it for searching
- Started by Doug Cutting
- In 2004 Google publishes Google File System (GFS) and MapReduce framework papers
- The Google File System - 2003

History of Hadoop

2003

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

→ hadoop HDFS

2004

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

→ hadoop Map Reduce

2006

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com

Google, Inc.

Abstract

gtable is a distributed storage system for managing tured data that is designed to scale to a very large petabytes of data across thousands of commodity rs. Many projects at Google store data in Bigtable, ding web indexing, Google Earth, and Google Fi- e. These applications place very different demands igtable, both in terms of data size (from URLs to

achieved scalability and high performance, but Big provides a different interface than such systems. Big does not support a full relational data model; inste provides clients with a simple data model that sup dynamic control over data layout and format, an lows clients to reason about the locality properties o data represented in the underlying storage. Data i dexed using row and column names that can be arbi strings. Bigtable also treats data as uninterpreted str

→ APACHE HBASE

History of Hadoop

- **2008 - Hadoop Wins Terabyte Sort Benchmark (**sorted 1 terabyte of data in 209 seconds, compared to previous record of 297 seconds)

- 2009 - Avro and Chukwa became new members of Hadoop Framework family

- 2010 - Hadoop's Hbase, Hive and Pig subprojects completed, adding more computational power to Hadoop framework

- **2011 - ZooKeeper Completed**

- **2013 - Hadoop 1.1.2 and Hadoop 2.0.3 alpha.**

  - Ambari, Cassandra, Mahout have been added

And Now

- **Hadoop:**
  - an open-source software framework that supports data-intensive distributed applications, licensed under the Apache v2 license.
- **Goals / Requirements:**
  - Abstract and facilitate the storage and processing of large and/or rapidly growing data sets
    - Structured and non-structured data
    - Simple programming models
  - High scalability and availability
  - Use commodity (cheap!) hardware with little redundancy
  - Fault-tolerance
  - Move computation rather than data

Organization used hadoop

AOL
ebay
IBM
twitter
amazon.com
Adobe
facebook
hulu
last·fm
the social music revolution
LinkedIn
YAHOO!

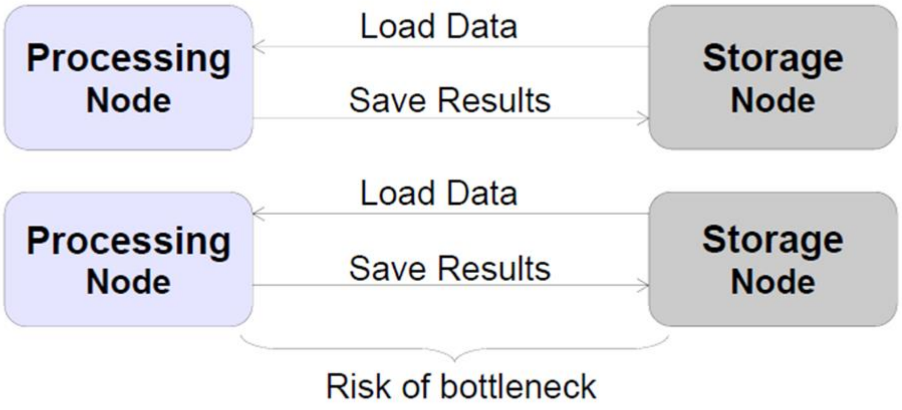
Source: http://wiki.apache.org/hadoop/PoweredBy

Hadoop system principles

- Scale-Out rather than Scale-Up
- Bring code to data rather than data to code
- Deal with failures – they are common
- Abstract complexity of distributed and concurrent applications

Code to Data

- Traditional data processing architecture
- Nodes are broken up into separate processing and storage nodes connected by high-capacity link
- Many data-intensive applications are not CPU demanding causing bottlenecks in network

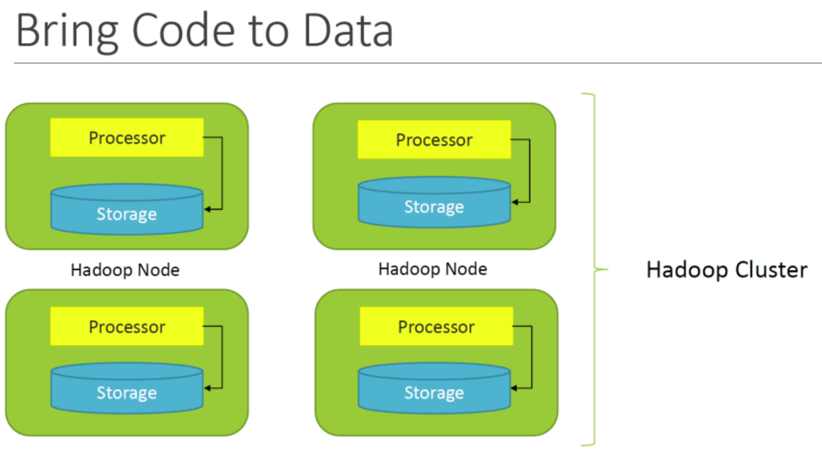


BRING CODE TO DATA

Hadoop co-locates processors and storage

Code is moved to data (size is tiny, usually in KBs)

Processors execute code and access underlying local storage



Failures are Common

- Given a large number machines, failures are common
- Large warehouses may see machine failures weekly or even daily

- Hadoop is designed to cope with node failures
  - Data is replicated
  - Tasks are retried

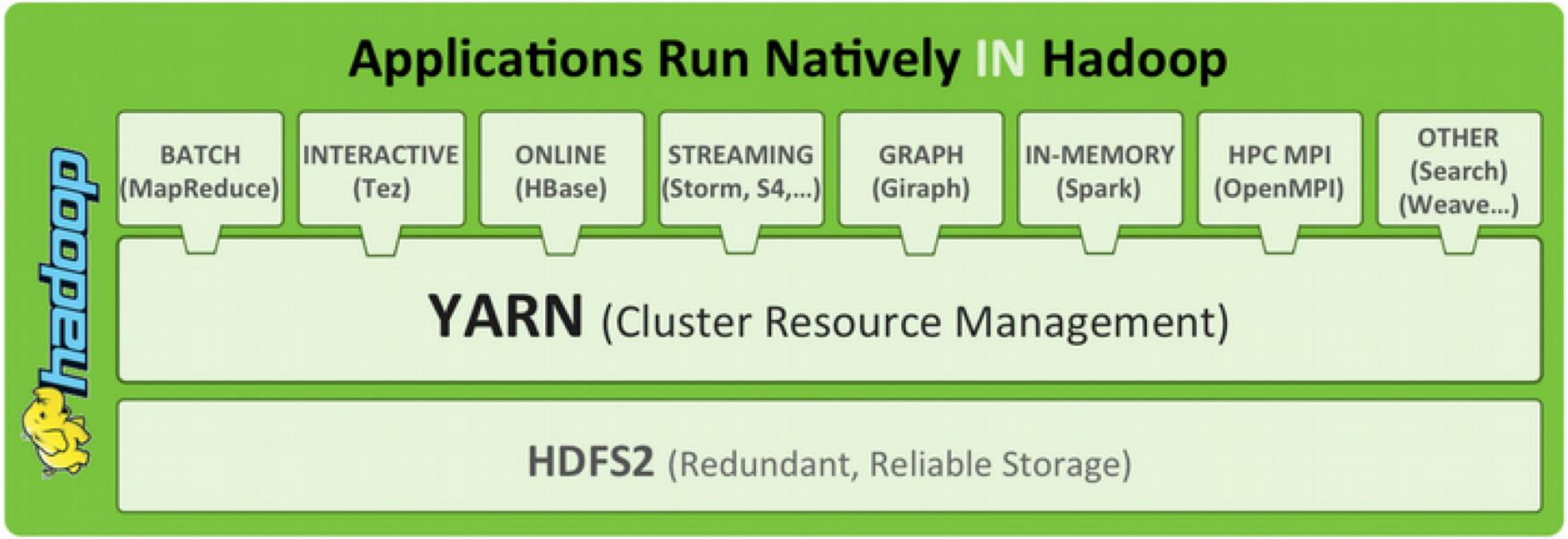
Abstract Complexity

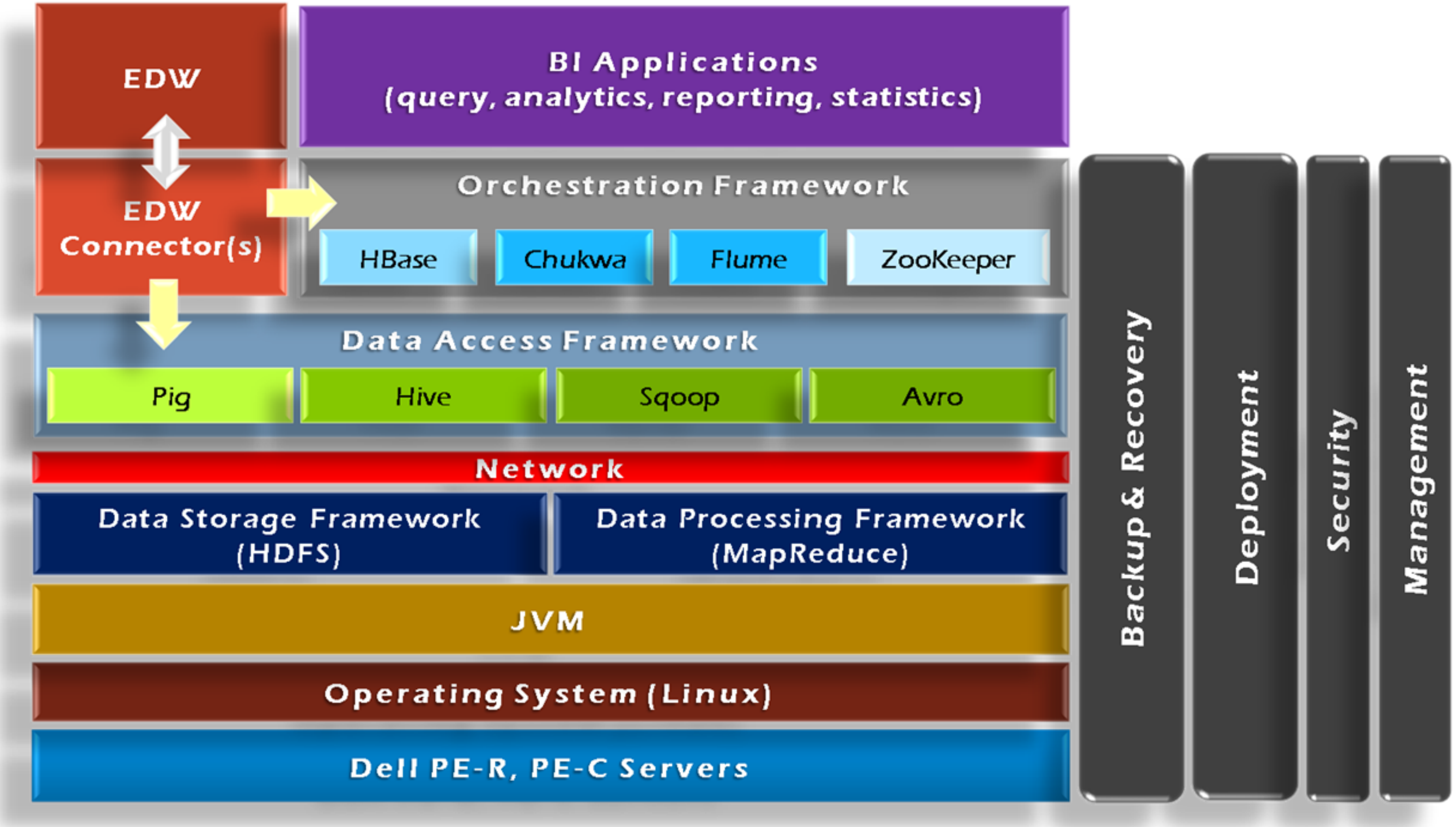
- Frees developer from worrying about systemlevel challenges
  - processing pipelines, data partitioning, code distribution

- Allows developers to focus on application development and business logic

Distribution Vendors

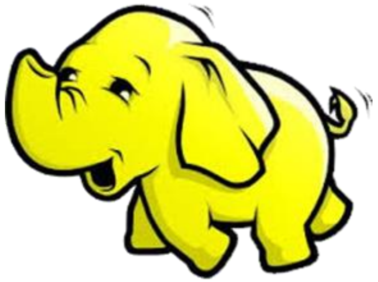
- Cloudera Distribution for Hadoop (CDH)
- MapR Distribution
- Hortonworks Data Platform (HDP)
- Apache BigTop Distribution


Applications Run Natively IN Hadoop
BATCH (MapReduce)
INTERACTIVE (Tez)
ONLINE (HBase)
STREAMING (Storm, S4,...)
GRAPH (Giraph)
IN-MEMORY (Spark)
HPC MPI (OpenMPI)
OTHER (Search) (Weave...)
YARN (Cluster Resource Management)
HDFS2 (Redundant, Reliable Storage)
hadoop


EDW
BI Applications
(query, analytics, reporting, statistics)
EDW
Connector(s)
Orchestration Framework
HBase
Chukwa
Flume
ZooKeeper
Data Access Framework
Pig
Hive
Sqoop
Avro
Network
Data Storage Framework
(HDFS)
Data Processing Framework
(MapReduce)
JVM
Operating System (Linux)
Dell PE-R, PE-C Servers
Backup & Recovery
Deployment
Security
Management

Hadoop projects

- HDFS : A distributed filesystem that runs on large clusters of commodity machines
- MapReduce : A distributed data processing model
- Hbase : A distributed, column-oriented database.
- Hive : A distributed data warehouse. Hive manages data stored in HDFS and provides a query language based on SQL

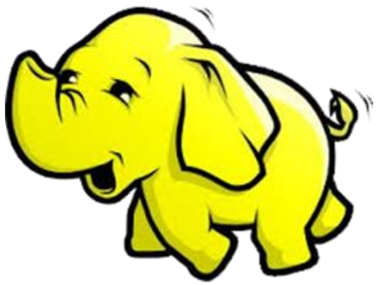


- **Pig : A data flow language and execution environment for exploring very large datasets**

- **ZooKeeper : A distributed, highly available coordination service.**

Hadoop Distributed File System

- Appears as a single disk
- Runs on top of a native filesystem
  - Ext3,Ext4,…
- Based on Google's Filesystem GFS
- Fault Tolerant
  - Can handle disk crashes, machine crashes, etc...
- portable Java implementation



HDFS is Good for...

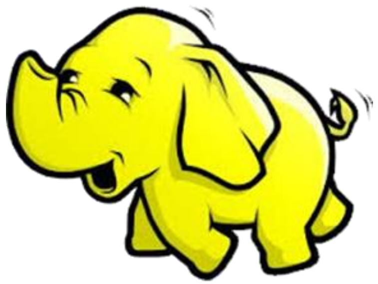
- Storing large files
  - Terabytes, Petabytes, etc...
  - Millions rather than billions of files
  - 100MB or more per file
- Streaming data
  - Write once and read-many times patterns
  - Optimized for streaming reads rather than random reads
- “Cheap” Commodity Hardware
  - No need for super-computers, use less reliable commodity hardware

HDFS is not so good for …

- Low-latency reads
  - High-throughput rather than low latency for small chunks of data
    - HBase addresses this issue

- Large amount of small files
  - Better for millions of large files instead of billions of small files
    - For example each file can be 100MB or more

- Multiple Writers
  - Single writer per file



HDFS Architecture

- Master-Slave Architecture

- ## HDFS Master "Namenode"
  - Manages all filesystem metadata
  - File name to list blocks + location mapping
  - File metadata (i.e. "inode")
  - Collect block reports from Datanodes on block locations
  - Controls read/write access to files
  - Manages block replication

- ## HDFS Slaves "Datanodes"
  - Notifies NameNode about block-IDs it has
  - Serve read/write requests from clients
  - Perform replication tasks upon instruction by namenode
  - Rack-aware

NameNode:

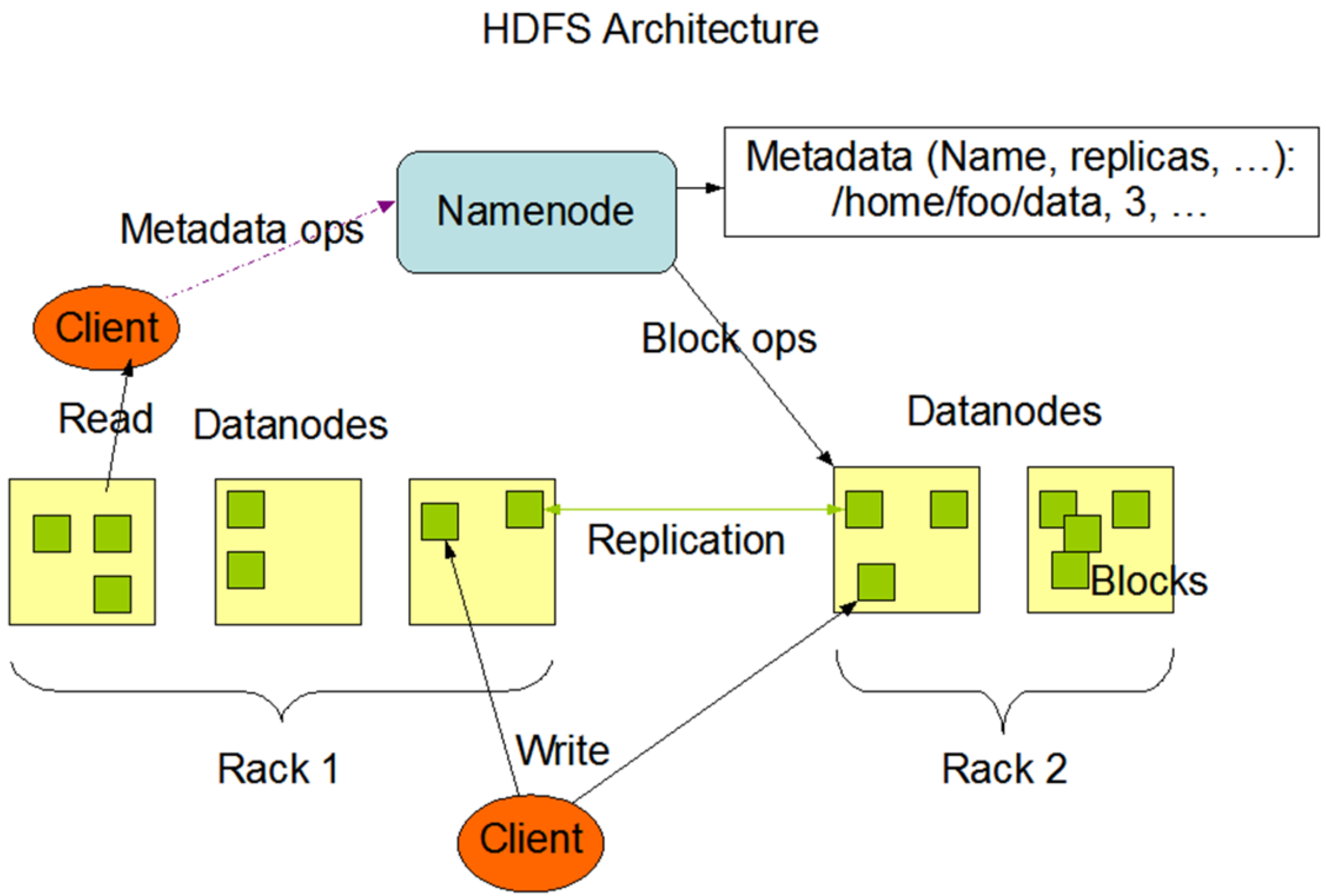
- Stores metadata for the files, like the directory structure of a typical FS.
- The server holding the NameNode instance is quite crucial, as there is only one.
- Transaction log for file deletes/adds, etc. Does not use transactions for whole blocks or file-streams, only metadata.
- Handles creation of more replica blocks when necessary after a DataNode failure

DataNode:

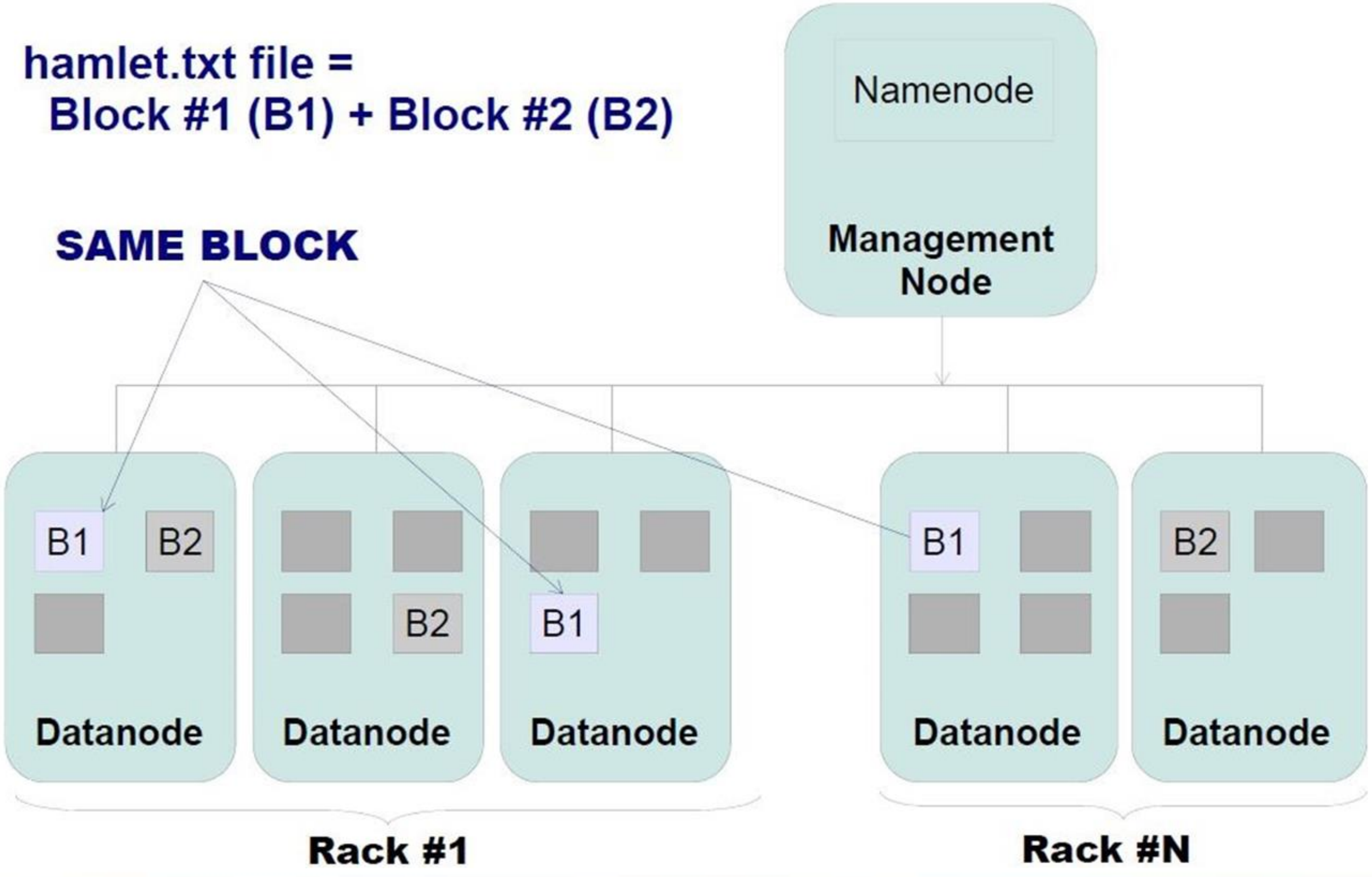
- Stores the actual data in HDFS
- Can run on any underlying filesystem (ext3/4, NTFS, etc)
- Notifies NameNode of what blocks it has
- NameNode replicates blocks 2x in local rack, 1x elsewhere

HDFS Architecture

- Secondary Namenode
  - Performs house keeping work so Namenode doesn't have to
  - Requires similar hardware as Namenode machine
  - Not used for high-availability – not a backup for Namenode

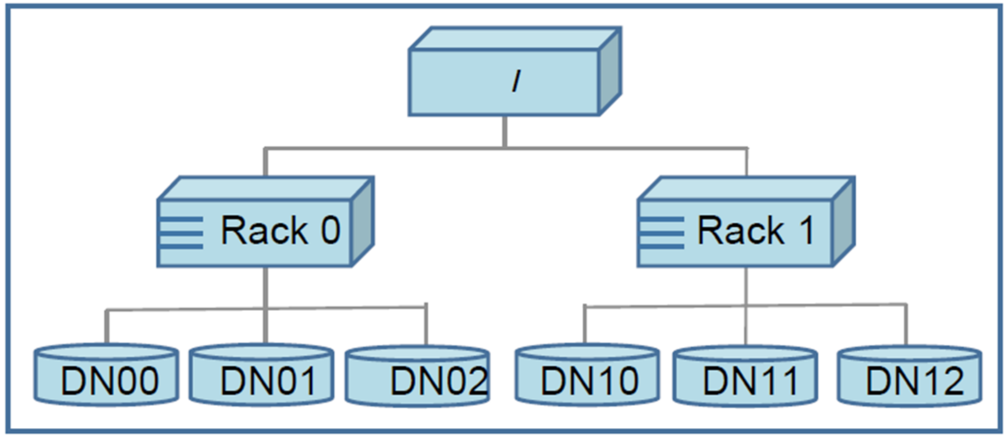

HDFS Architecture
Metadata ops
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

Files and Blocks

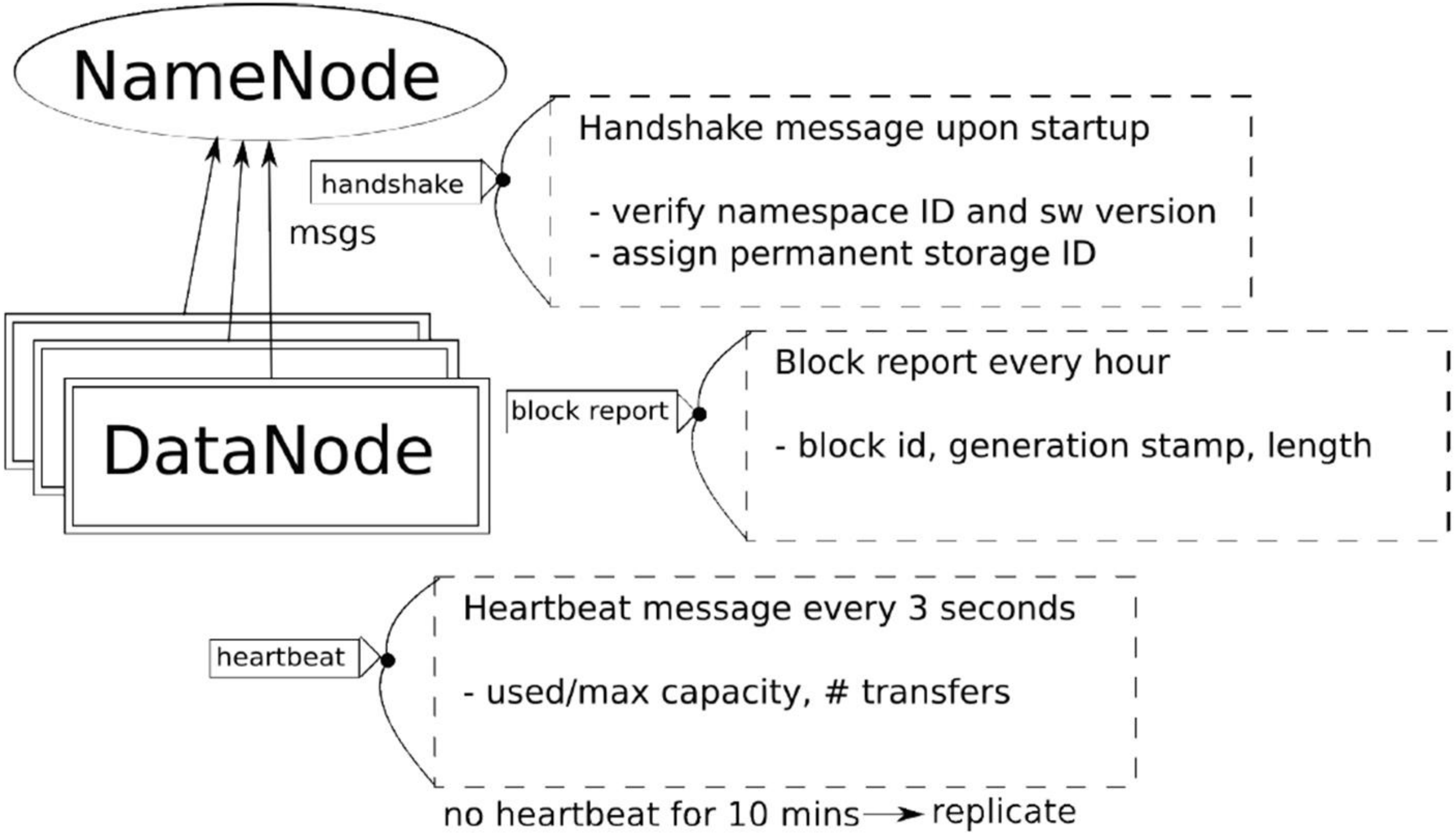


REPLICA MANGEMENT

- A common practice is to spread the nodes across multiple racks
- improve data reliability, availability, and network bandwidth utilization
- Namenode determines replica placement



HDFS Component Communication



handshake

- During startup each DataNode connects to the NameNode and performs a handshake

- The purpose is to verify the namespace ID and the software version

- After the handshake the DataNode registers with the NameNode

block report

- A block report contains the *block id*, the length for each block replica
- The first is sent immediately after the DataNode registration
- Subsequent block reports are sent every hour.

heartbeats

- During normal operation DataNodes send *heartbeats* to the NameNode to confirm that the DataNode is operating and the block replicas it hosts are available.

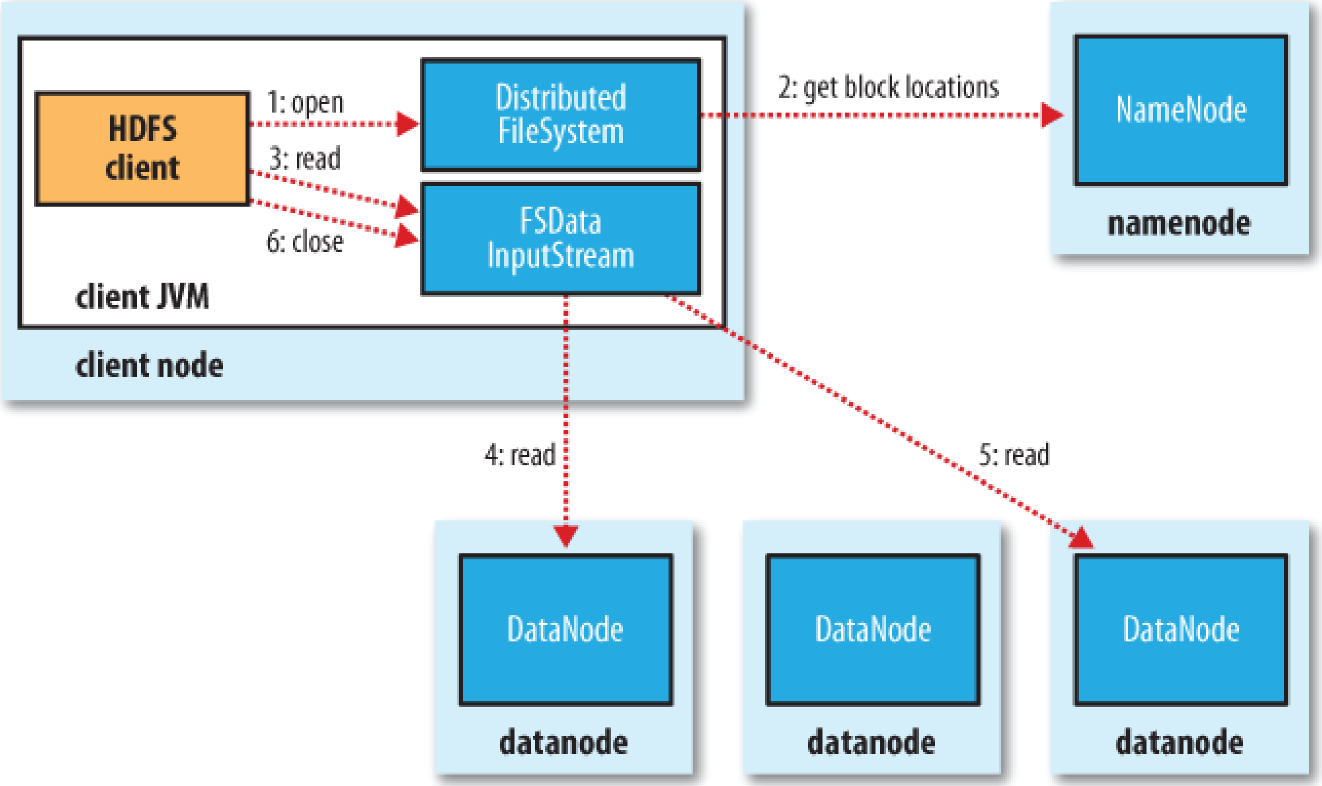
- Heartbeats from a DataNode also carry information about:
  - Total storage capacity
  - Fraction of storage in use

- The default heartbeat interval is three seconds

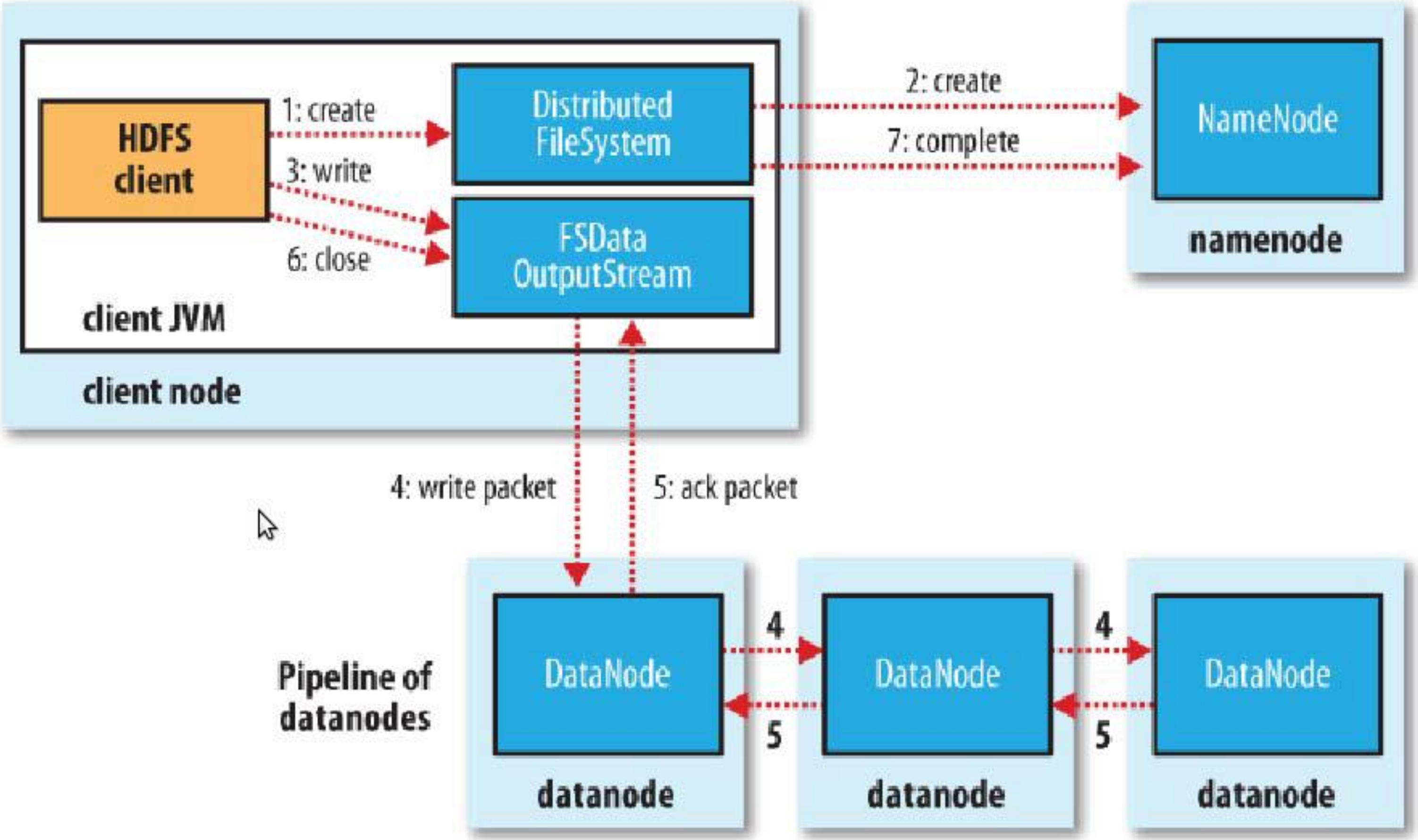
Resd,Write, Append,Delete

- Using the FileSystem API:
- an open() method to get the input stream for a file
- The create() methods for writing
- append to an existing file using the append() method
- Use the delete() method on FileSystem to permanently remove files or directories

Reading Data From HDFS



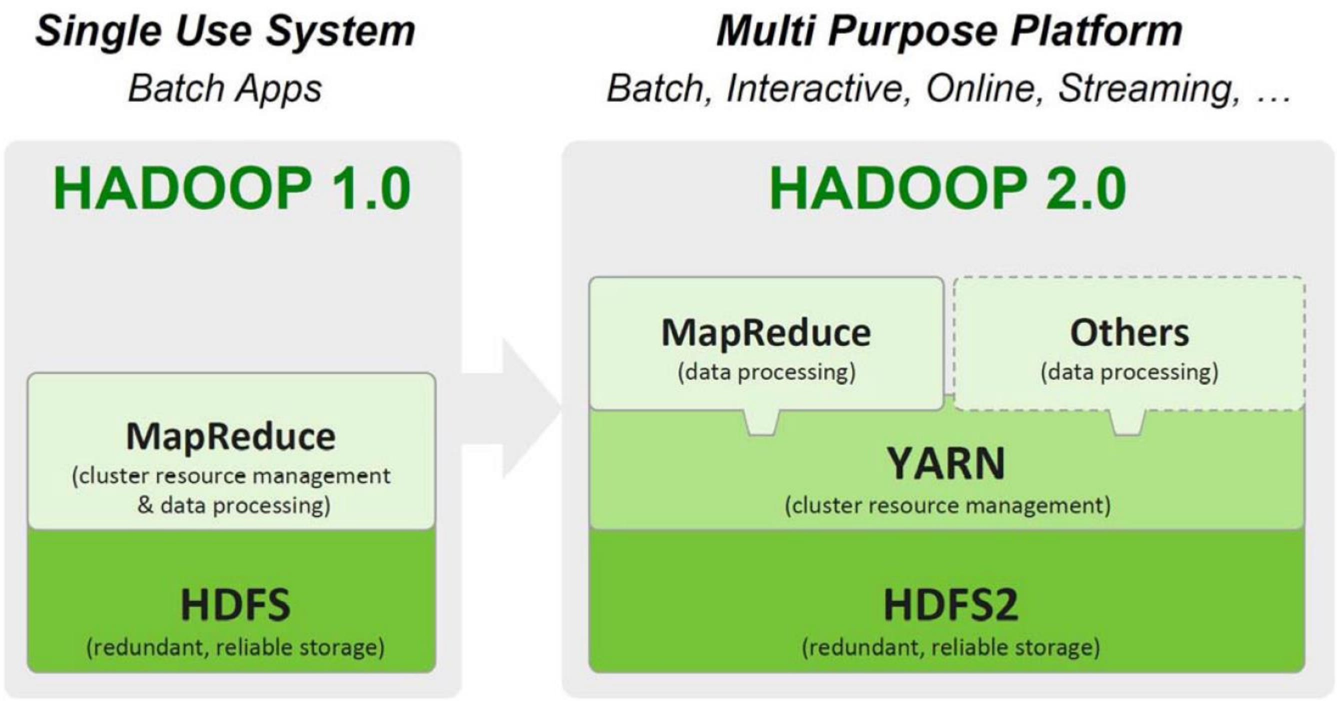
Writing Data To HDFS



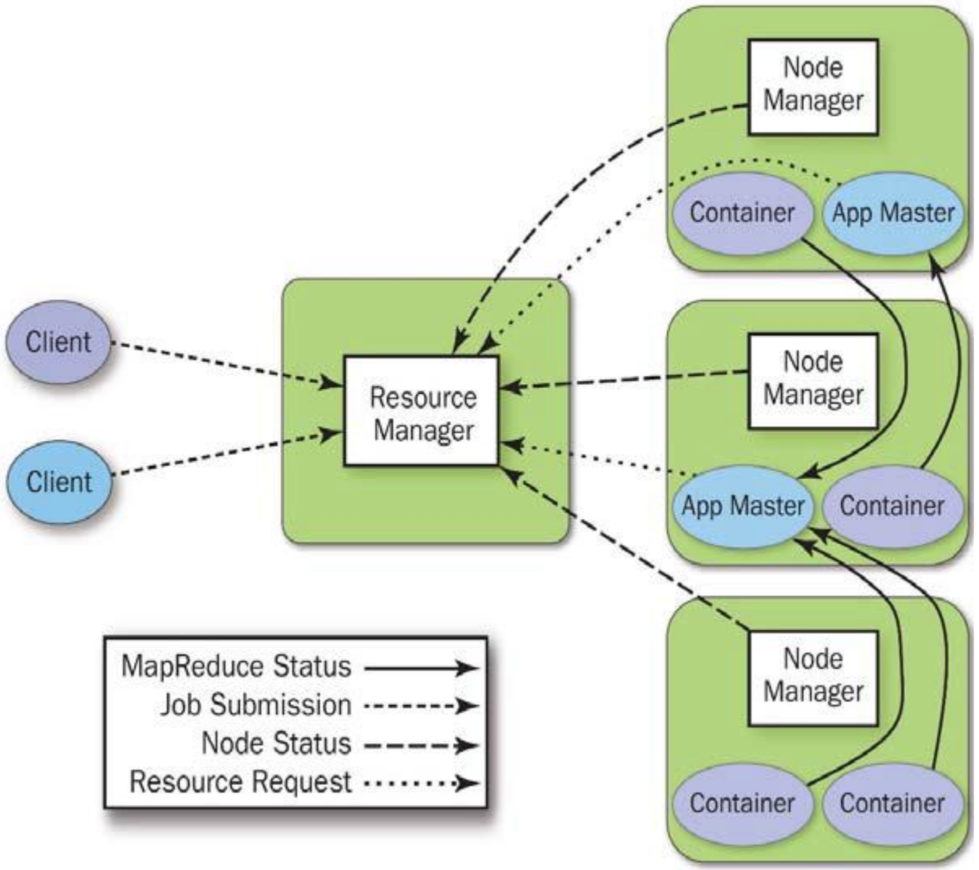
YARN

- **Hadoop MapReduce Classic**
- MapReduce Classic Limitations:
- Scalability
- Maximum Cluster size – 4,000 nodes
- Maximum concurrent tasks – 40,000
- Coarse synchronization in JobTracker
- •Availability
- • Failure kills all queued and running jobs
- •Hard partition of resources into map and reduce slots
- • Low resource utilization
- Lacks support for alternate paradigms and services
- Iterative applications implemented using MapReduce are 10x slower

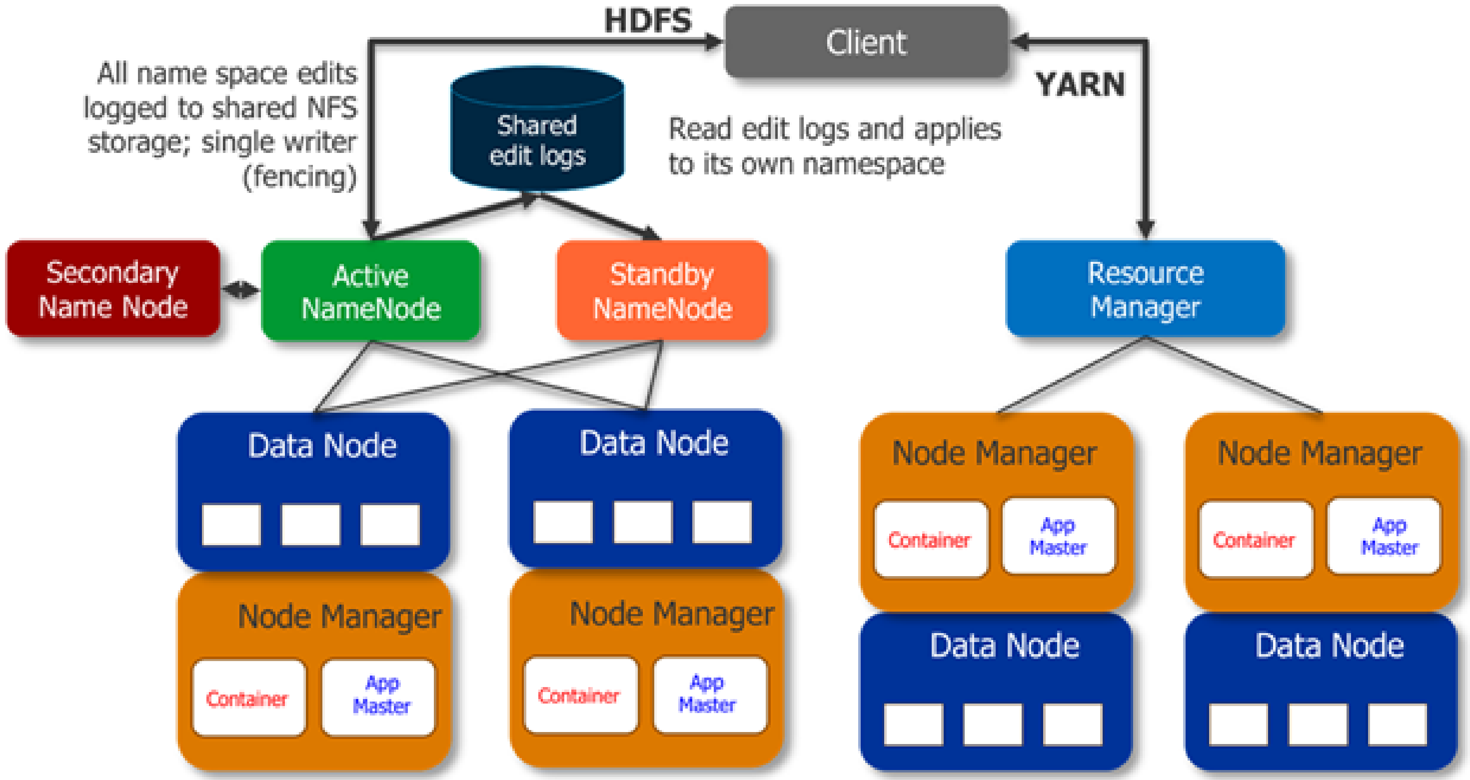
Hadoop as Next-Gen Platform



- YARN Architecture and Concepts
- Application
- Application is a job submitted to the framework
- Example – Map Reduce Job
- Container
- Basic unit of allocation
- Fine-grained resource allocation across multiple resource types (memory, cpu, disk, network,
- gpu etc.)
- • container_0 = 2GB, 1CPU
- • container_1 = 1GB, 6 CPU
- • Replaces the fixed map/reduce slots


Node
Manager
Container
App Master
Client
Resource
Manager
Node
Manager
Client
App Master
Container
Node
Manager
Container
Container
MapReduce Status
Job Submission
Node Status
Resource Request

Hadoop Architecture



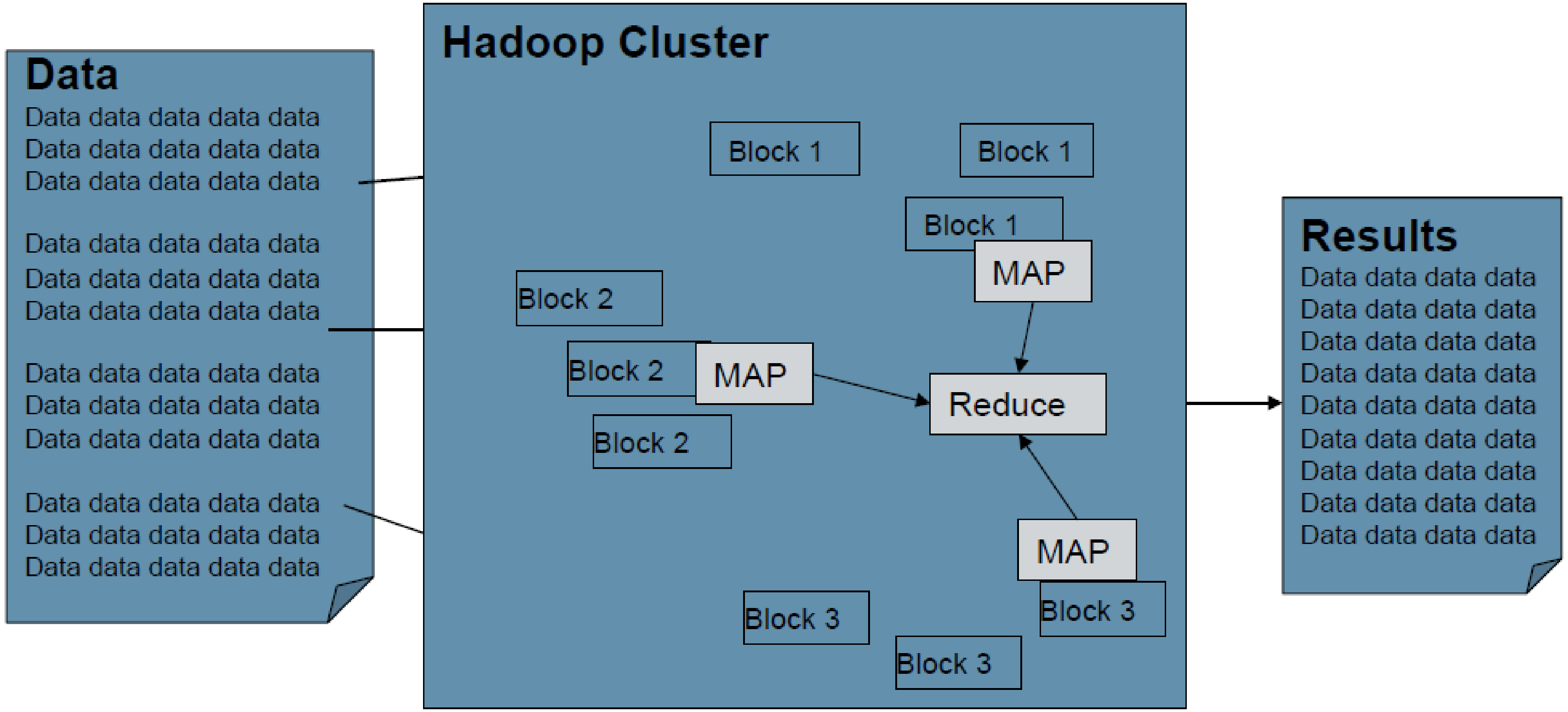
What is MapReduce?

- Parallel programming model meant for large clusters
  - User implements Map() and Reduce()
- Parallel computing framework
- Libraries take care of EVERYTHING else
- Parallelization
- Fault Tolerance
- Data Distribution
- Load Balancing
- Useful model for many practical tasks (large data)

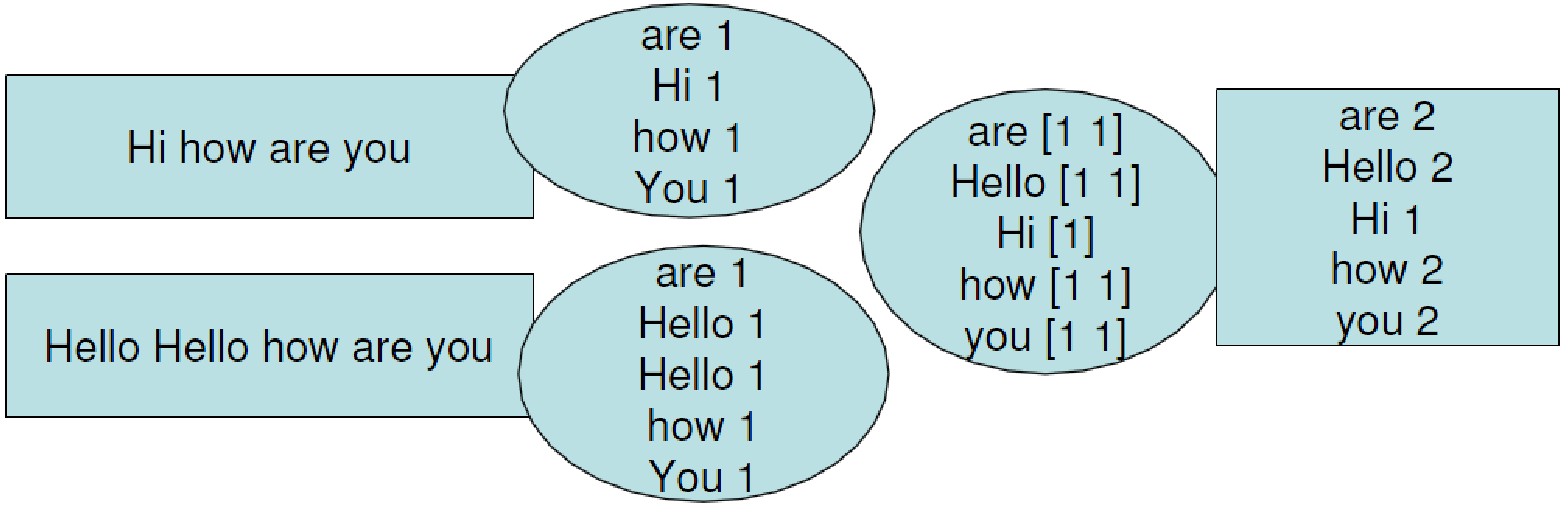
Functional Abstractions Hide Parallelism

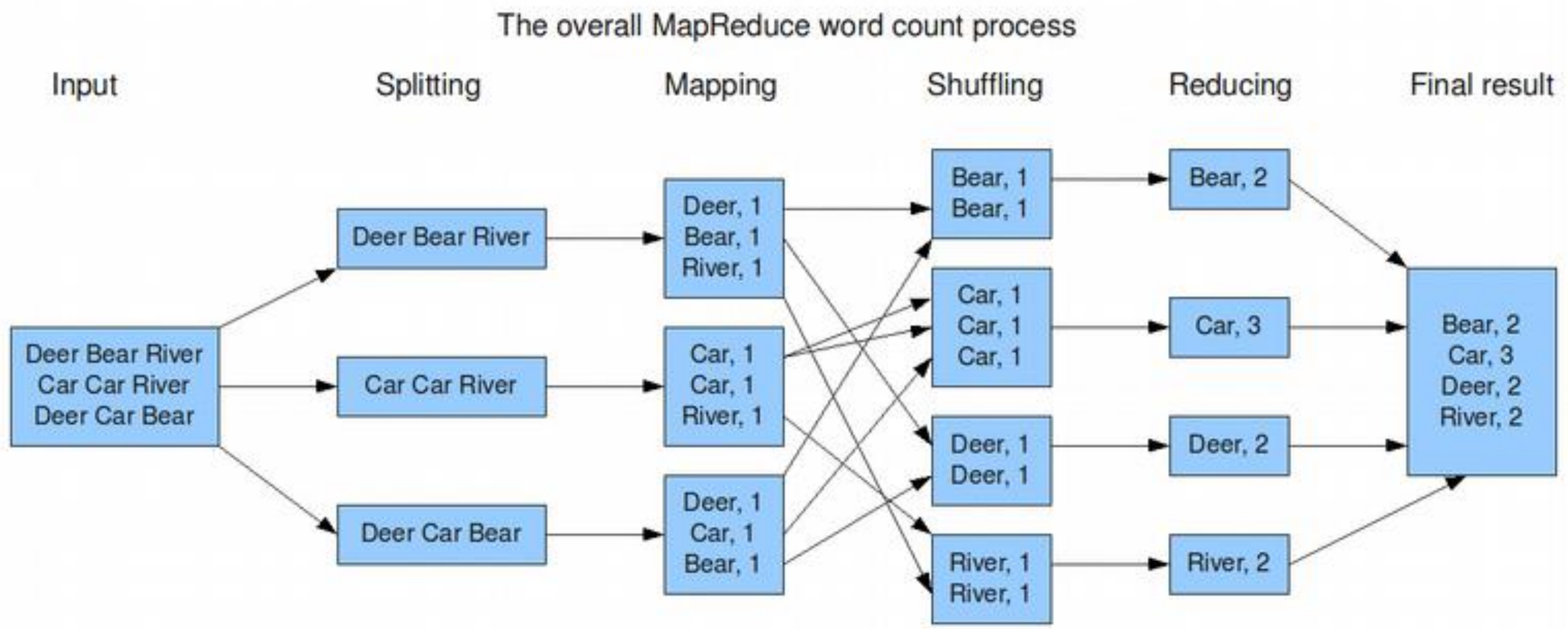
- Map and Reduce
- Functions borrowed from functional programming languages (eg. Lisp)
- Map()
- Process a key/value pair to generate intermediate key/value pairs•
- Reduce()
- Merge all intermediate values associated with the same key

Map Reduce


Data
Data data data data data
Data data data data data
Data data data data data
Data data data data data
Data data data data data
Data data data data data
Data data data data data
Data data data data data
Data data data data data
Data data data data data
Data data data data data
Data data data data data
Hadoop Cluster
Block 1
Block 1
Block 1
MAP
Block 2
Block 2
MAP
Reduce
Block 2
MAP
Block 3
Block 3
Block 3
Results
Data data data data
Data data data data
Data data data data
Data data data data
Data data data data
Data data data data
Data data data data
Data data data data
Data data data data

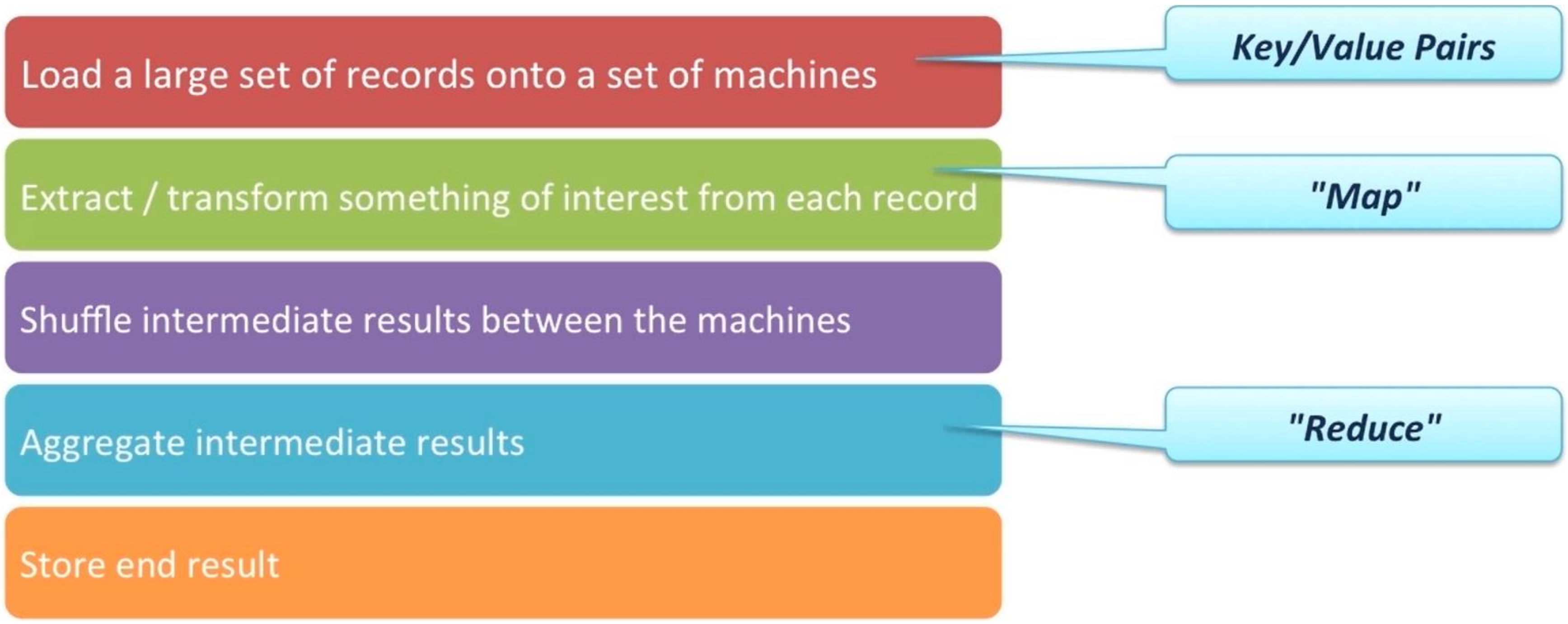
Distributed Processing




The overall MapReduce word count process
Input
Splitting
Mapping
Shuffling
Reducing
Final result
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer, 1
River, 1
River, 1
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

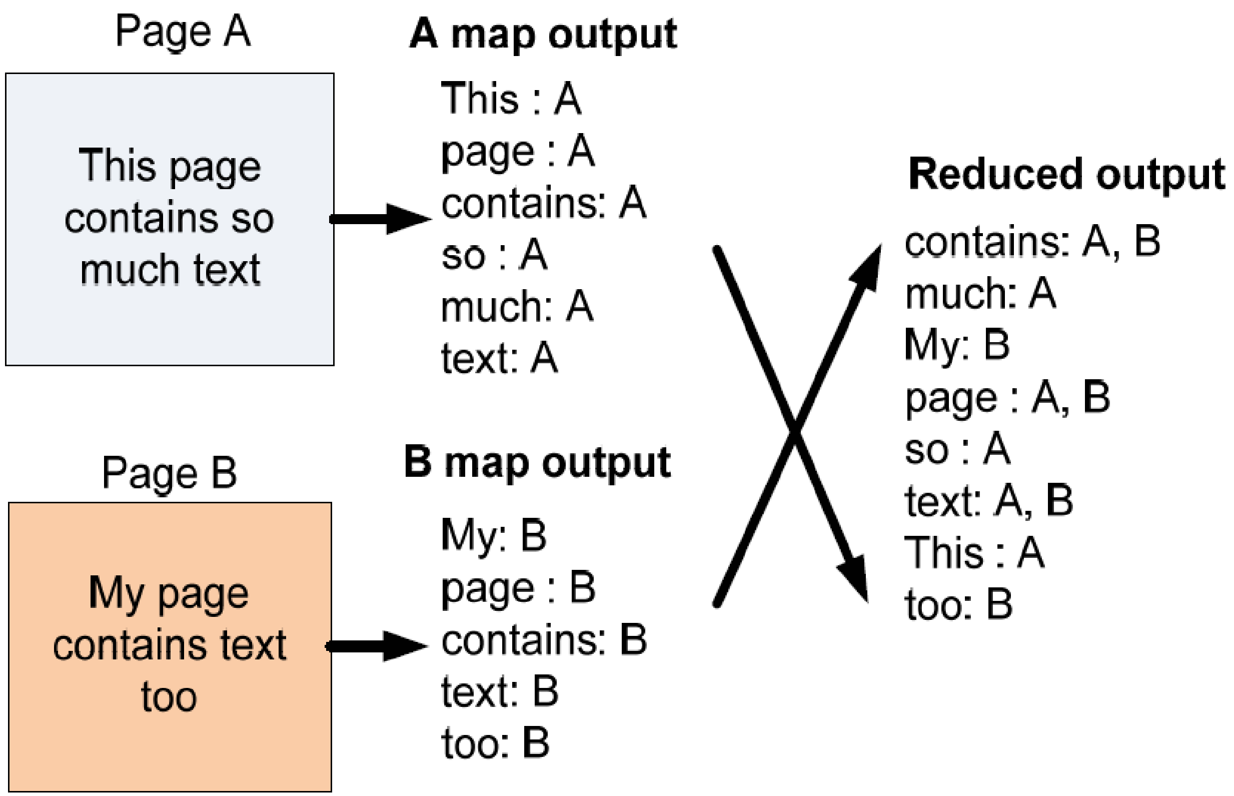
Map-Reduce on Large Clusters

- **Motivation and Demand:**
  - Tend to be very short, code-wise
  - Represent a data flow

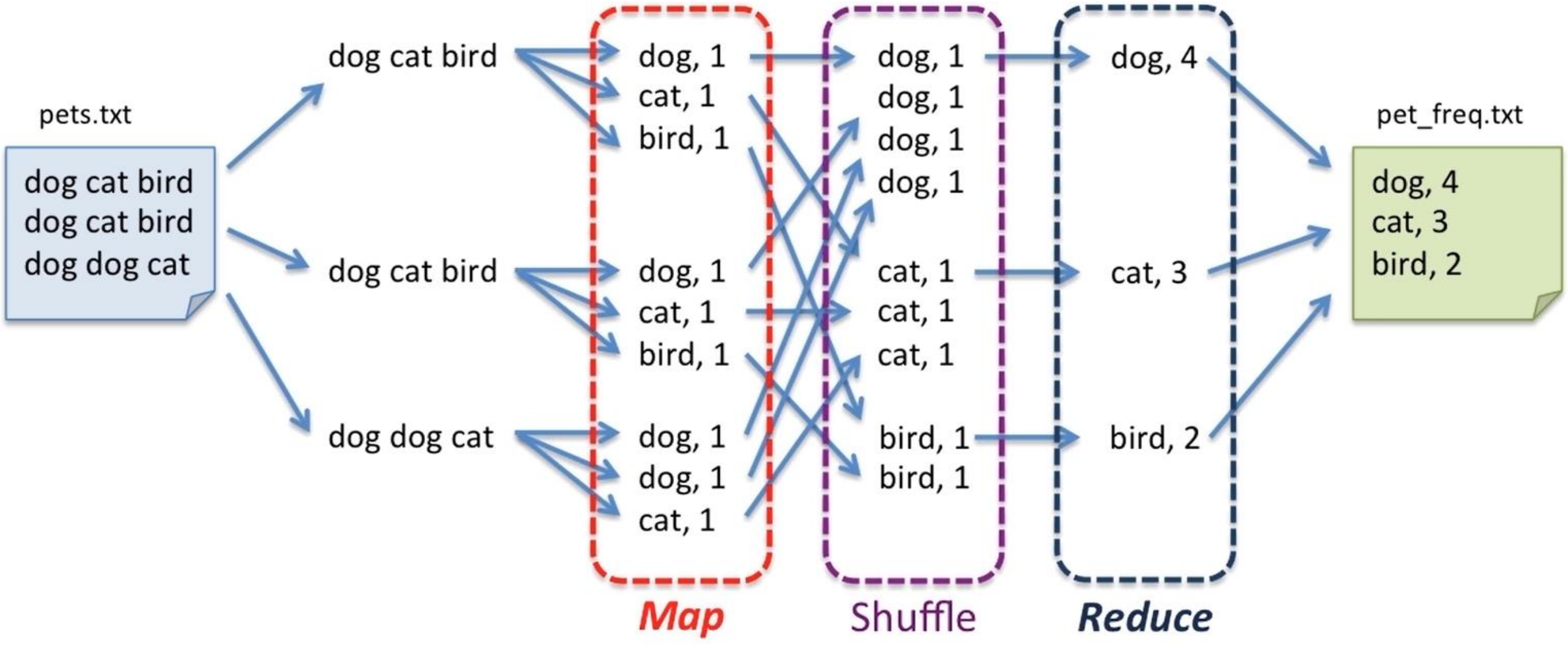


Map-Reduce (Cont.)

Index: Data Flow

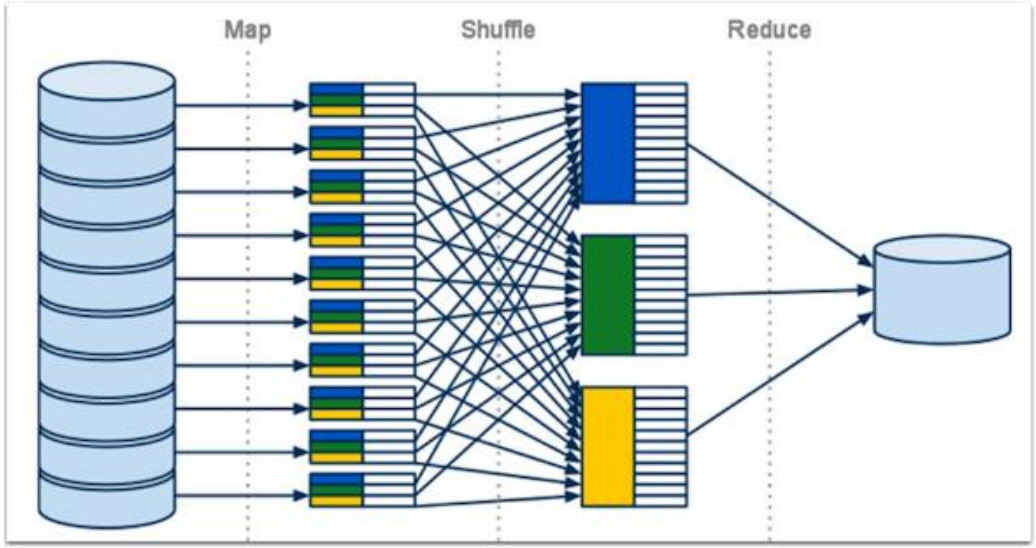


Map-Reduce (Cont.)



Map-Reduce (Cont.)

- **Each step has one Map phase and one Reduce phase**
  - Convert any into MapReduce pattern
- **Great solution for one-pass computations**
  - Not very efficient for Multi-pass computations and algorithms



Map/Reduce in Python

- import sys
- for line in sys.stdin:
- line = line.strip()
- words = line.split()
- for word in words:
- print '%s\t%s' % (word,1)

Word count

- from operator import itemgetter
- import sys
- current_word = None
- current_count = 0
- word = None
- for line in sys.stdin:
- line = line.strip()
- word, count = line.split('\t', 1)

- try:
- count = int(count)
- except ValueError:
- continue
- if current_word == word:
- current_count += count
- else:
- if current_word:
- print '%s\t%s' % (current_word, current_count)
- current_count = count
- current_word = word
- if current_word == word:
- print '%s\t%s' % (current_word, current_count)

Hadoop Framework

Features :

- Open Source Framework for Processing Large Data
- Work on Cheap and Unreliable Clusters
- Known in Companies who deal with Big Data Applications
- Compatible with Java, Python and Scala

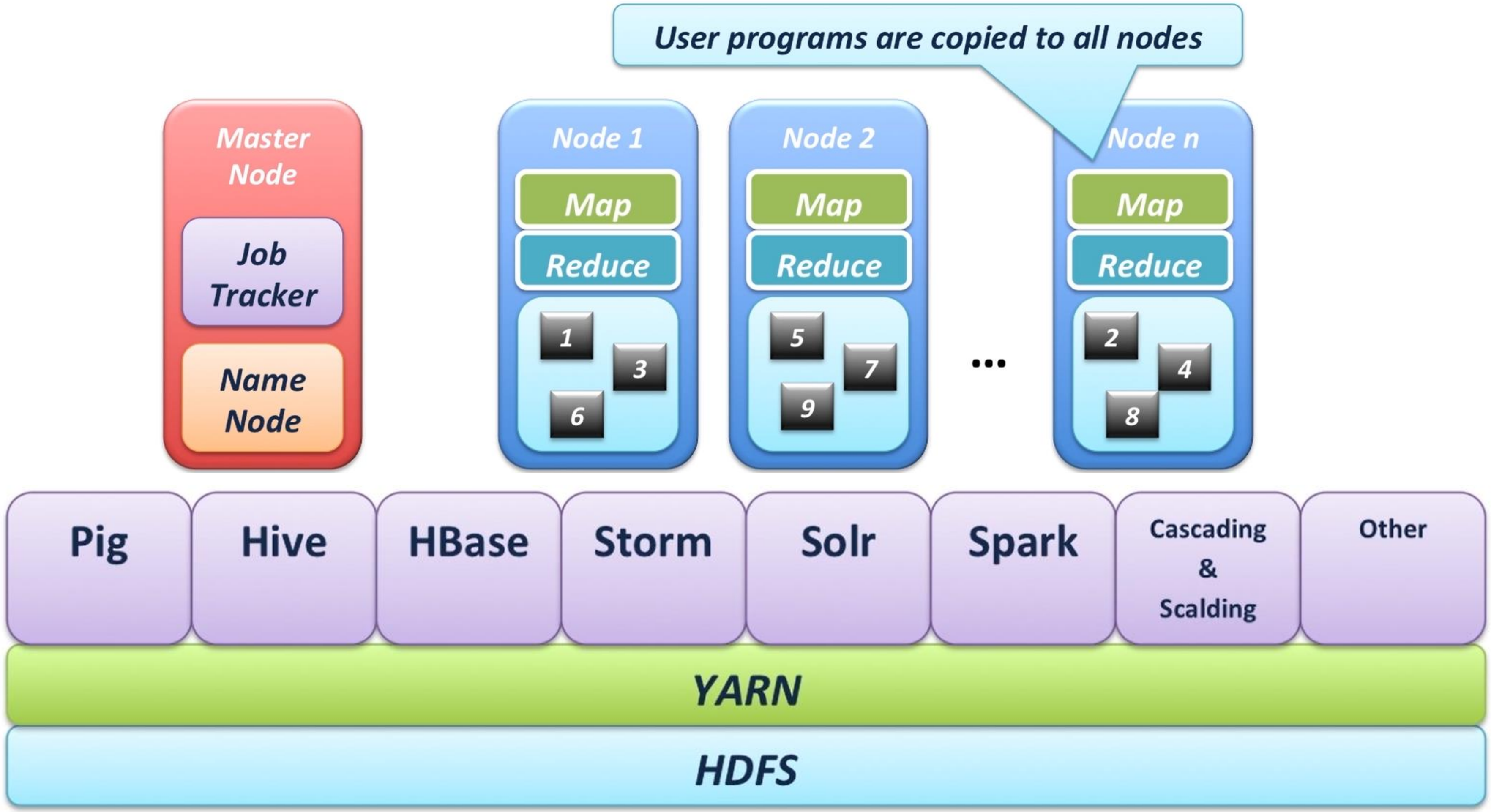
Programmers	Framework
• Map • Reduce	• Deals with fault tolerance • Assign workers to map and reduce tasks • Moves processes to data • Shuffles and sorts intermediate data • Deals with errors

Hadoop Framework (Cont.)

- **MapReduce Framework**
  - Assign work for different nodes
- **Hadoop Distributed File System (HDFS)**
  - Primary storage system used by Hadoop applications.
  - Copies each piece of data and distributes to individual nodes
    - Name Node (Meta Data) and Data Nodes (File Blocks)
    - Redundant information (Three times by default)
    - Machines in a given cluster are cheap and unreliable
    - Decreases the risk of catastrophic failure
      - » Even in the event that numerous nodes fail
  - Links together the file systems on different nodes to make an integrated big file system (Parallel Processing)

Hadoop Framework (Cont.)

- **Hadoop V.2 :** Hadoop NextGen MapReduce **(YARN)**



Hadoop Framework (Cont.)

- **Hadoop Programming**
  - Java
    - Full control of MapReduce , Cascading (Open Java Library)
  - Python , Scala, Ruby
- **Data Retrieval / Query Language**
  - Hive
    - SQL- Like Language
  - Pig
    - Data Flow Language (Simple and Out of Small Steps)
  - Scalding
    - Library built on top of Scala (Elegant Model)

Hadoop Framework (Cont.)

...atforms & tools for big data analytics in healthcare

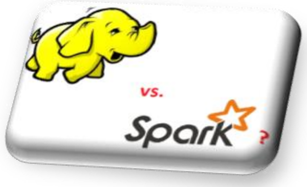
Platform/Tool	Description
The Hadoop Distributed File System (HDFS)	HDFS enables the underlying storage for the Hadoop cluster. It divides the data into smaller parts and distributes it across the various servers/nodes.
MapReduce	MapReduce provides the interface for the distribution of sub-tasks and the gathering of outputs. When tasks are executed, MapReduce tracks the processing of each server/node.
PIG and PIG Latin (Pig and PigLatin)	Pig programming language is configured to assimilate all types of data (structured/unstructured, etc.). It is comprised of two key modules: the language itself, called PigLatin, and the runtime version in which the PigLatin code is executed.
Hive	Hive is a runtime Hadoop support architecture that leverages Structure Query Language (SQL) with the Hadoop platform. It permits SQL programmers to develop Hive Query Language (HQL) statements akin to typical SQL statements.
Jaql	Jaql is a functional, declarative query language designed to process large data sets. To facilitate parallel processing, Jaql converts "'high-level' queries into 'low-level' queries" consisting of MapReduce tasks.
Zookeeper	Zookeeper allows a centralized infrastructure with various services, providing synchronization across a cluster of servers. Big data analytics applications utilize these services to coordinate parallel processing across big clusters.
HBase	HBase is a column-oriented database management system that sits on top of HDFS. It uses a non-SQL approach.
Cassandra	Cassandra is also a distributed database system. It is designated as a top-level project modeled to handle big data distributed across many utility servers. It also provides reliable service with no particular point of failure (http://en.wikipedia.org/wiki/Apache_Cassandra) and it is a NoSQL system.
Oozie	Oozie, an open source project, streamlines the workflow and coordination among the tasks.
Lucene	The Lucene project is used widely for text analytics/searches and has been incorporated into several open source projects. Its scope includes full text indexing and library search for use within a Java application.
Avro	Avro facilitates data serialization services. Versioning and version control are additional useful features.
Mahout	Mahout is yet another Apache project whose goal is to generate free applications of distributed and scalable machine learning algorithms that support big data analytics on the Hadoop platform.

Big Data Programming

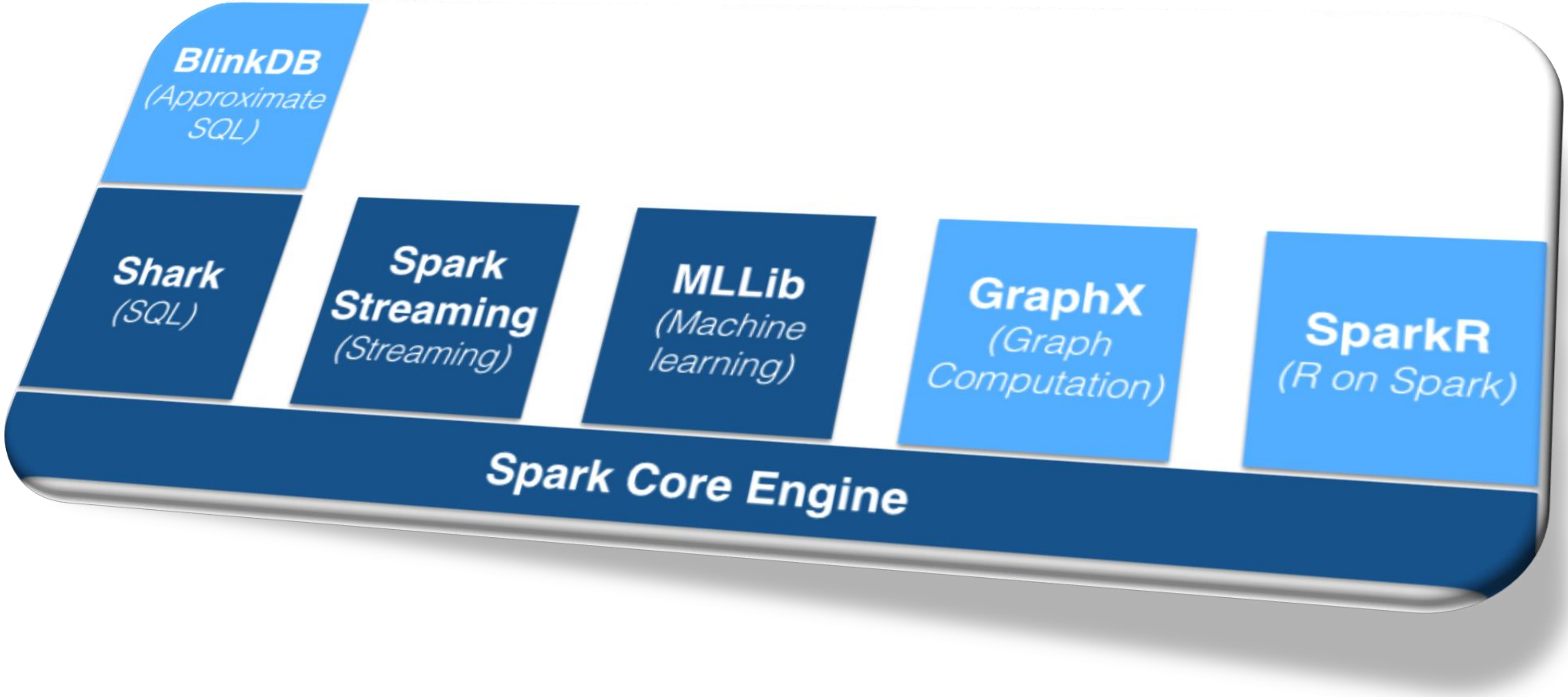
- R – Java- Python and Scala (Commonly Used)
- Three References : (Recommended to Read)
  - https://www.linkit.nl/knowledge-base/177/4_most_used_languages_in_big_data_projects_Java
  - https://www.linkit.nl/knowledge-base/226/4_most_used_languages_in_big_data_projects_R
  - https://www.linkit.nl/eng/knowledge-base/196/4_most_used_languages_in_big_data_projects_Python

Apache Spark Framework

- **Spark Features (More than Distributed Processing)**
  - Ease of use, and sophisticated analytics
  - In-memory data storage and near real-time processing
  - Holds intermediate results in memory
  - Store as much as data in memory and then goes to disk
- **Spark vs Hadoop**
  - On top of existing HDFS
  - Data sets that are diverse in nature (Text, Videos, …)
  - Variety in source of data (Batch v. real-time streaming data).
  - 100 times faster in memory, 10 times faster when running on disk.



Apache Spark Framework (Cont.)



Apache Spark Framework (Cont.)

- **Compatible with Java, Scala and Python**
- **Perform Data Analytics and Machine Learning**
  - SQL Queries, Streaming Data
  - Machine Learning and Graph Data Processing
    - Spark MLlib, Spark's Machine Learning library
- **Spark and data stored in a Cassandra database**
  - **(Case Study)**

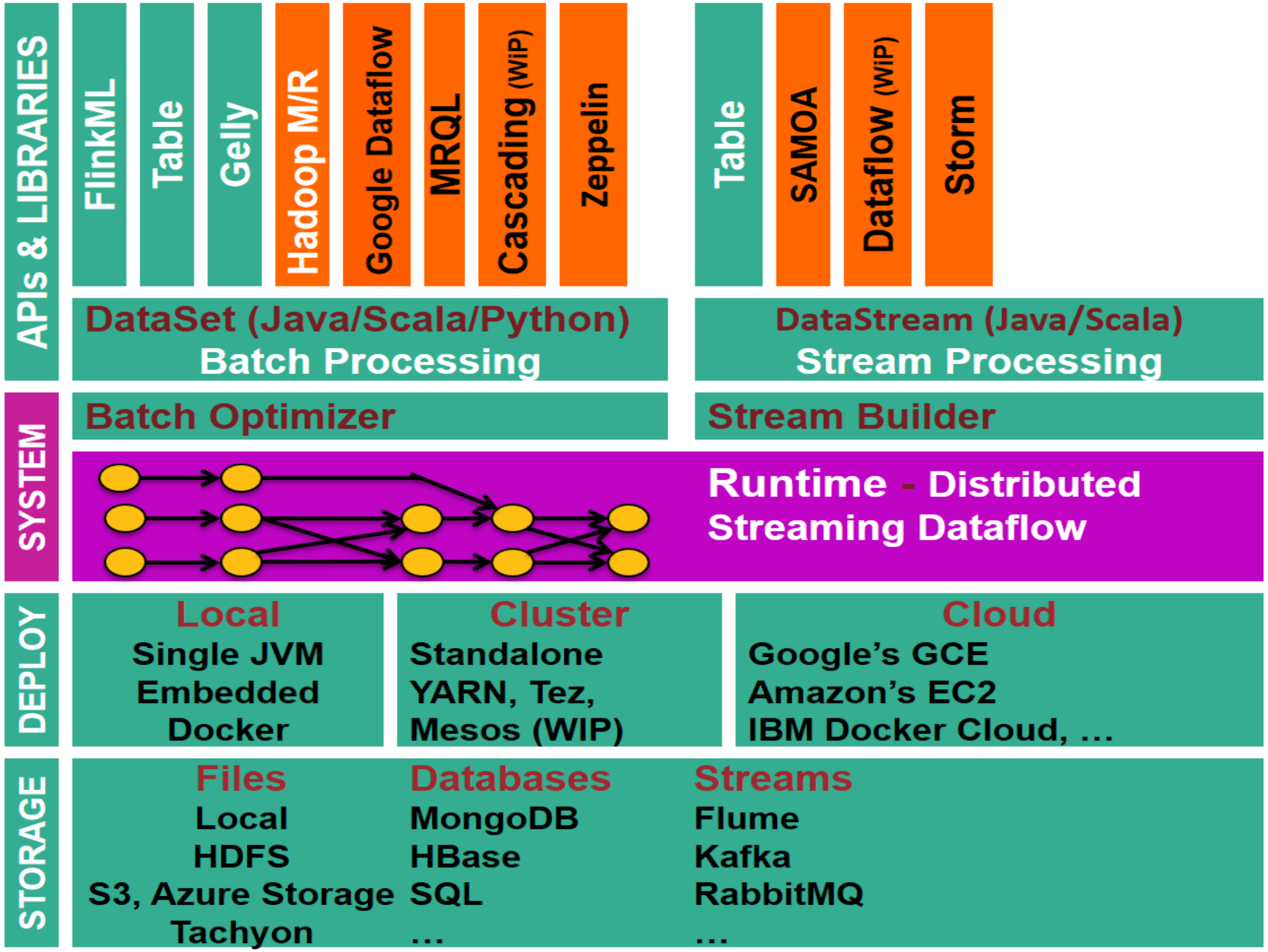
Apache Flink

- Apache Flink is an open source platform
  - Distributed stream and batch data processing.”
    - https://flink.apache.org/
- The definition in wikipedia:
  - https://en.wikipedia.org/wiki/Apache_Flink

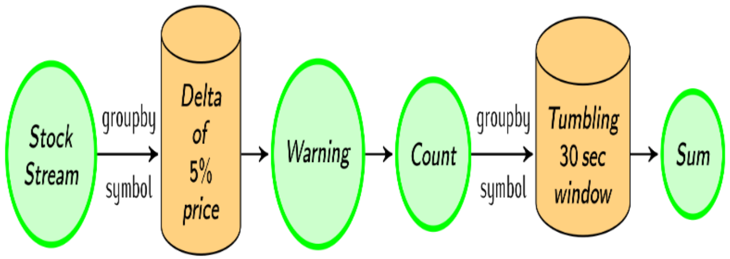
Apache Flink (Cont.)

- written in Java and Scala, consists of:
  - Big data processing engine:
  - Distributed and scalable streaming dataflow engine
- Several APIs in Java / Scala / Python:
  - DataSet API – Batch processing
  - DataStream API – Real-Time streaming analytics
- Domain-Specific Libraries:
  - FlinkML: Machine Learning Library for Flink
  - Gelly: Graph Library for Flink
  - Table: Relational Queries
  - FlinkCEP: Complex Event Processing for Flink

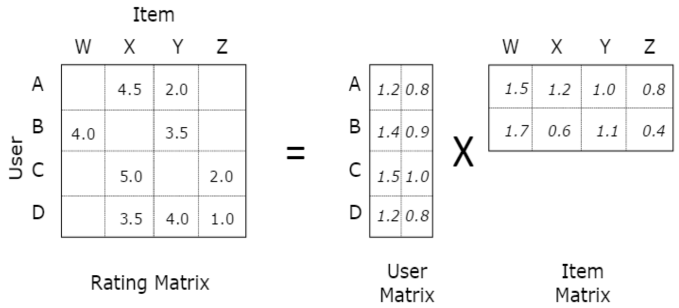
Apache Flink (Cont.)



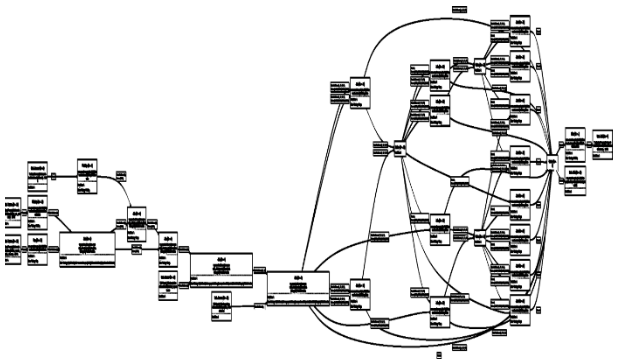
Apache Flink (Cont.)



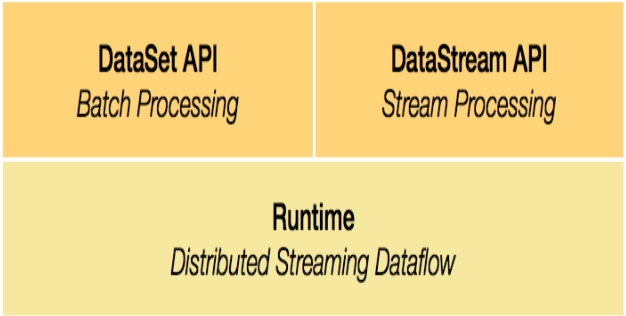
Real-Time stream processing

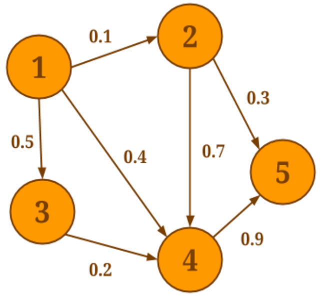


Machine Learning



Batch Processing





Graph Analysis

Big Data and Cloud

- **Cloud Computing Platform & Services**
  - (Cloudera, Hortonworks, MapR, Azure)

Apache Spark

- Processing engine; instead of just “map” and “reduce”, defines a large set of operations (transformations & actions)
- Operations can be arbitrarily combined in any order
- Open source software
- Supports Java, Scala and Python
- Key construct: Resilient Distributed Dataset (RDD)

RDD Operations

Transformations	Actions
map (func)	reduce(func)
flatMap(func)	collect()
filter(func)	count()
groupByKey()	first()
reduceByKey(func)	take(n)
mapValues(func)	saveAsTextFile(path)
sample(...)	countByKey()
union(other)	foreach(func)
distinct()	...
sortByKey()	
...	

Sample Spark transformations

- 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

- union(otherDataset): Return a new dataset that contains the union of the elements in the source dataset and the argument.

Sample Spark Actions

- intersection(otherDataset): Return a new RDD that contains the intersection of elements in the source dataset and the argument.

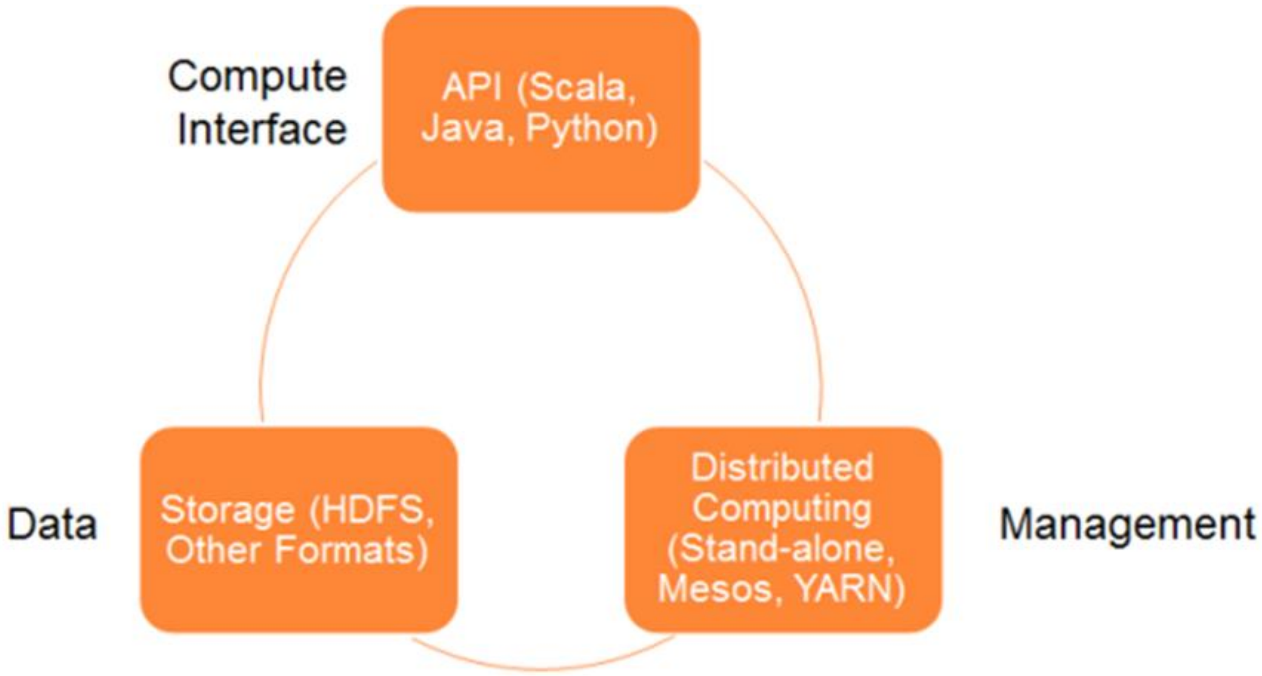
- distinct([numTasks])): Return a new dataset that contains the distinct elements of the source dataset

- join(otherDataset, [numTasks]): 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.

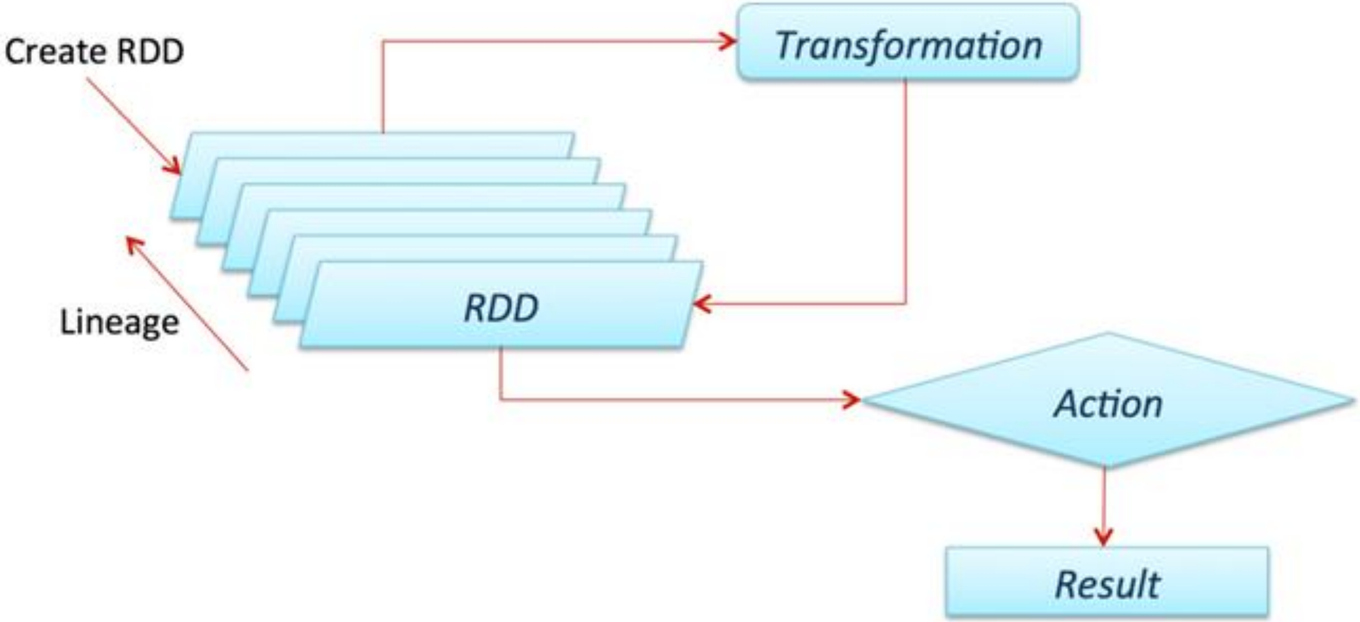
Sample Spark Actions

- 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.
- 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.

Spark Architecture



Fault tolerance because an RDD know how to recreate and re-compute the datasets.
RDDs are immutable.



Hadoop and Spark

Hadoop	Spark
Map & Reduce -> suitable for on-pass computations	multi-step data pipelines using directed acyclic graph (DAG) pattern.
Clusters are hard to set up and manage	supports in-memory data sharing across DAGs.
need to integrate with Mahout (Machine Learning) and Storm (Streaming data processing)	Spark as an alternative to Hadoop MapReduce

Gray sort competition: Winner Spark-based (previously MR)

Sort benchmark, Daytona Gray: sort of 100 TB of data (1 trillion records)

	Hadoop MR Record	Spark Record (2014)
Data Size	102.5 TB	100 TB
Elapsed Time	72 mins	23 mins
# Nodes	2100	206
# Cores	50400 physical	6592 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**

Spark-based System 3x faster with 1/10 # of nodes

http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

Spark vs. Hadoop MapReduce

- Performance: Spark normally faster but with caveats
- Spark can process data in-memory; Hadoop MapReduce persists back to the disk after a map or reduce action
- Spark generally outperforms MapReduce, but it often needs lots of memory to do well; if there are other resource-demanding services or can't fit in memory, Spark degrades
- MapReduce easily runs alongside other services with minor performance differences, & works well with the 1-pass jobs it was designed for
- Ease of use: Spark is easier to program
- Data processing: Spark more general
- Maturity: Spark maturing, Hadoop MapReduce mature

Contact Info :

- Nima Farajian
  - Telephone : +98 912 297 3630
  - Telegram : @nima555ir
  - Email : Nimaff2000@yahoo.com
- Mehdi Habibzadeh
  - Telephone : +98 912 326 7046
  - Telegram : +1 514 632 2838
  - Email : Nimahm@gmail.com

Zahra Rezaei

  - Telephone:+989133615108
  - Telegram :@Z_Rezaei2010
  - Email : z.rezaei2010@gmail.com