Summary of Big Data Frameworks Course

2015

Professor Sasu Tarkoma

Course Schedule

Tuesday 10.3. Introduction and the Big Data Challenge

Tuesday 17.3. MapReduce and Spark: Overview

Tuesday 24.3. MapReduce Optimizations and Algorithms. Spark Internals.

Tuesday 31.3. Distributed algorithms for Big Data: Elastic Data Processing and Developing Spark Algorithms.

Tuesday 14.4. MLBase, MLLib, and GraphX. Streaming Spark.

Tuesday 21.4. Industry views to Big Data

Tuesday 28.4. Summary

Four exercise problem sheets

Big Data

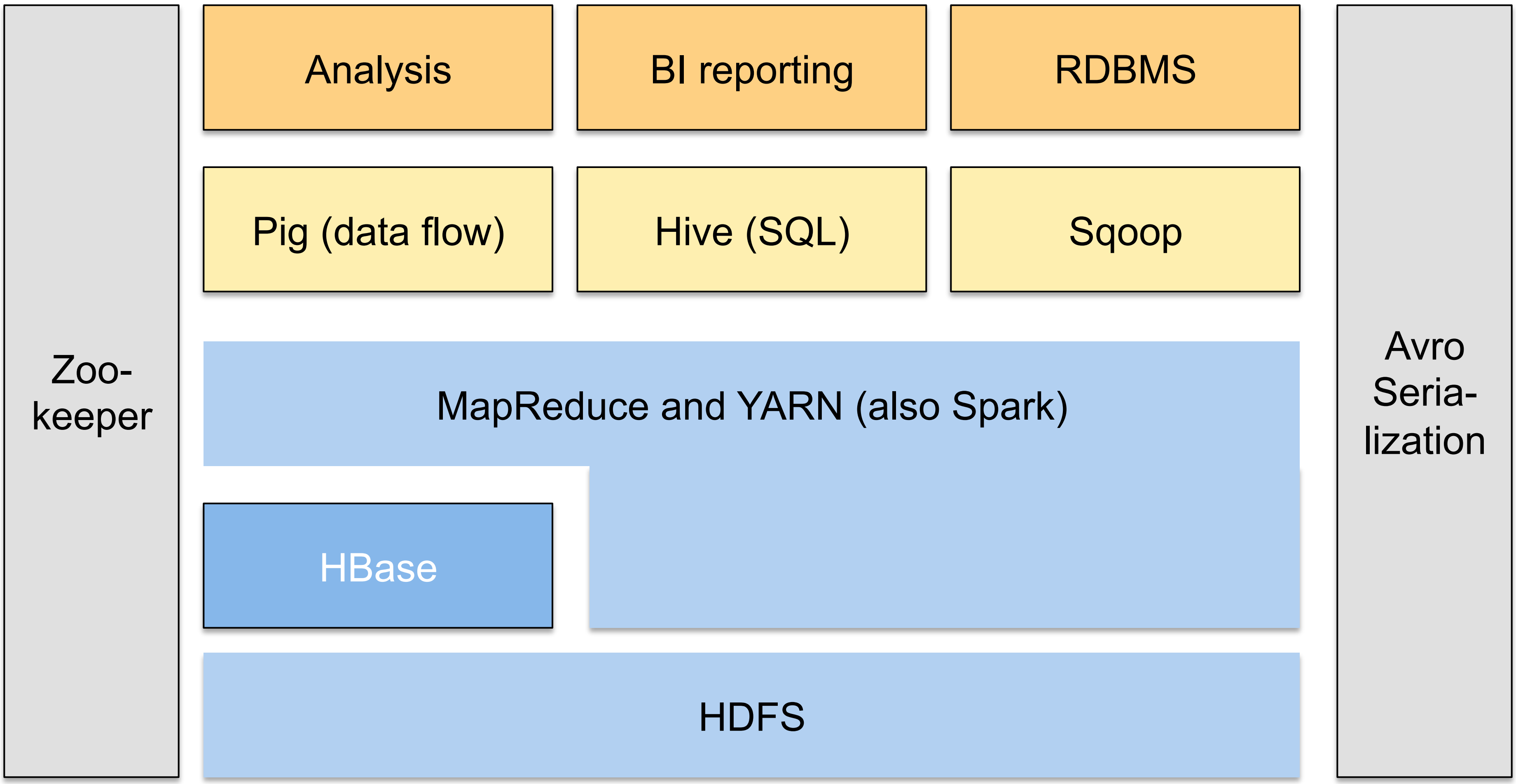
- A **massive** volume of structured and unstructured data
- **Cannot** be processed with traditional database and software solutions
- **Traditional** data analysis algorithms run **too slow** over the data
- ***High-volume, high-velocity, high-variety***
- **Big Data Pipeline:**
  - Data acquisition, storage, analysis, post-processing, results

Big Data Process

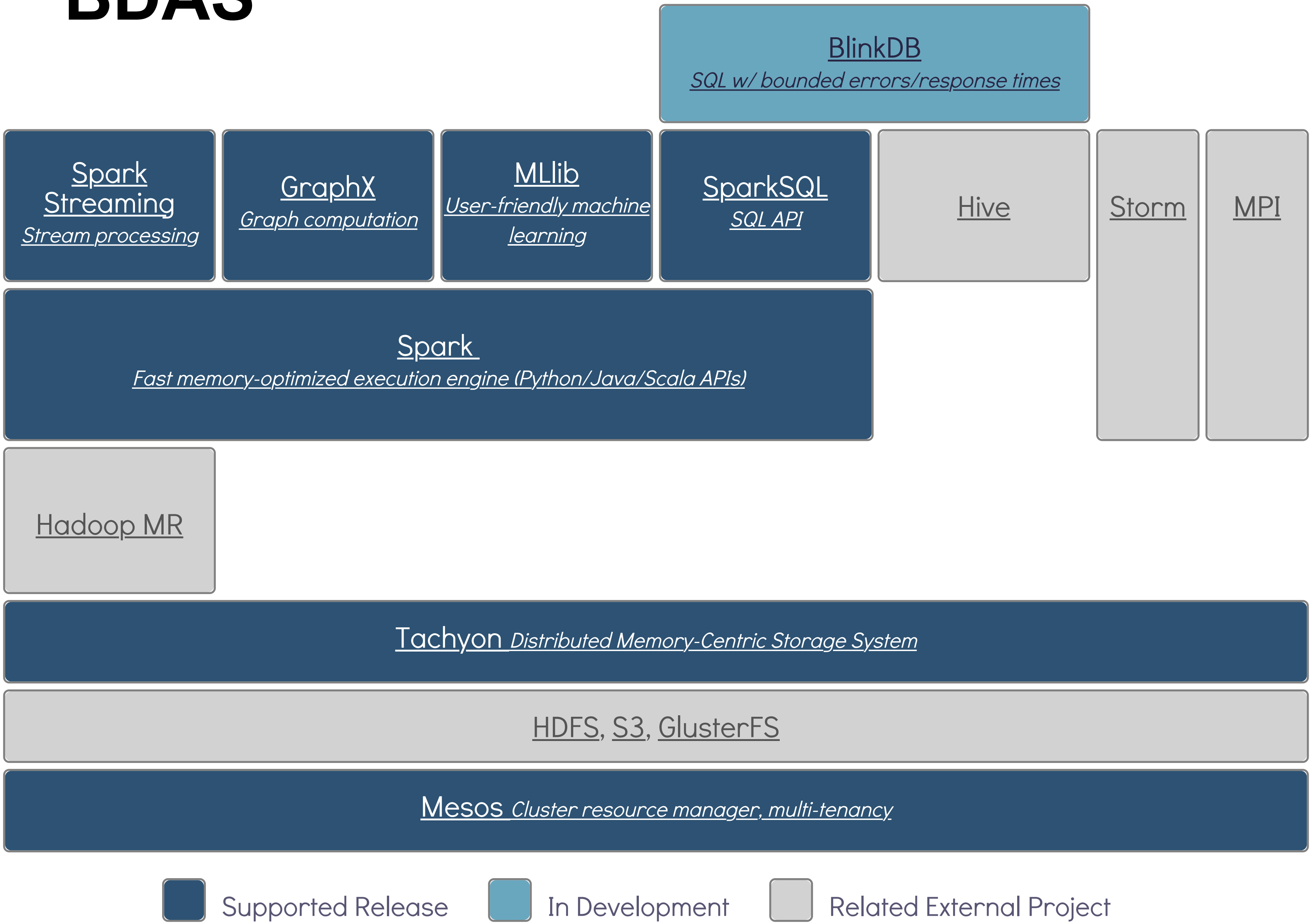
1. Acquisition
2. Extraction
3. Integration
4. Analysis
5. Interpretation
6. Decision
7. Understanding decision and starting from 1.

We have a feedback loop and the process is iterative.

Apache Hadoop Ecosystem

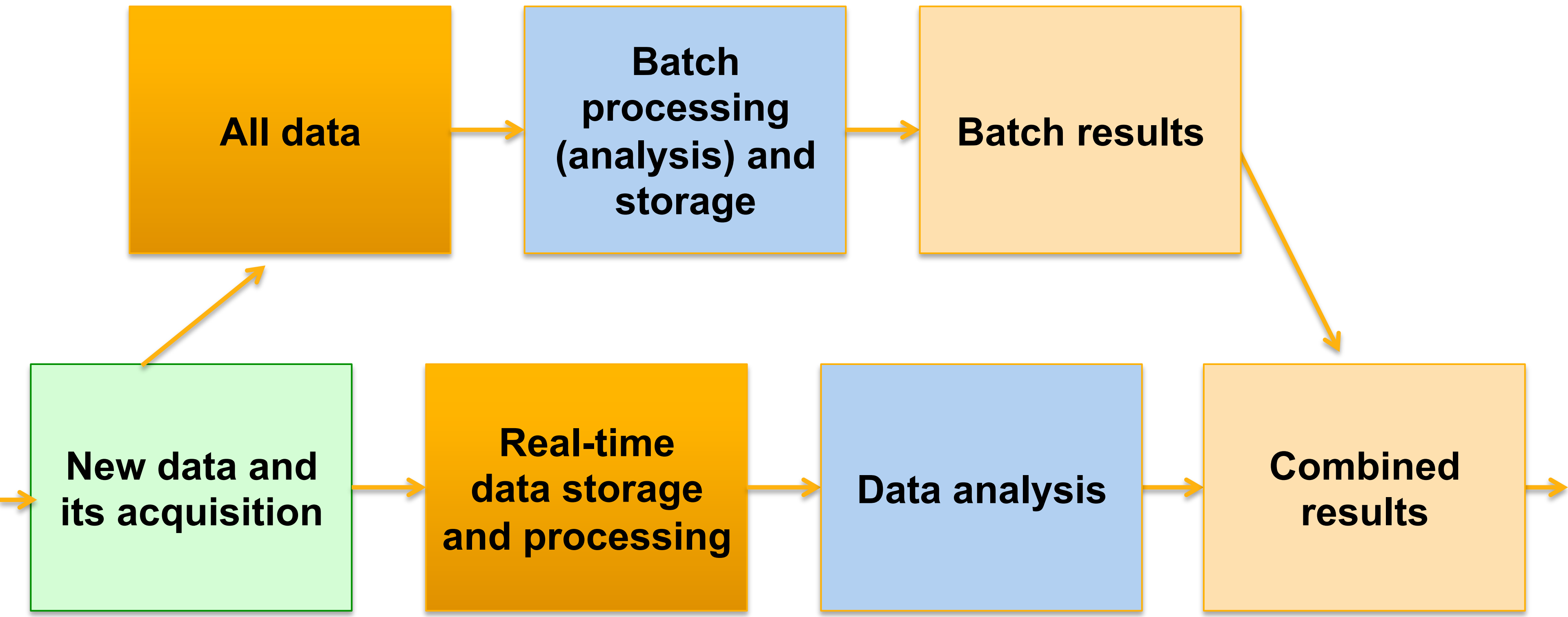


BDAS



https://amplab.cs.berkeley.edu/software/

Lambda Architecture



This is integrated in Spark

MapReduce Model

Google MapReduce introduced in 2004

Jeffrey Dean et al. MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004.

Apache Hadoop since 2005

http://hadoop.apache.org/

Apache Hadoop 2.0 introduced in 2012

Vinod Kumar Vavilapalli et al. Apache Hadoop YARN: Yet Another Resource Negotiator, SOCC 2013.

New cluster resource management layer (YARN)

MapReduce

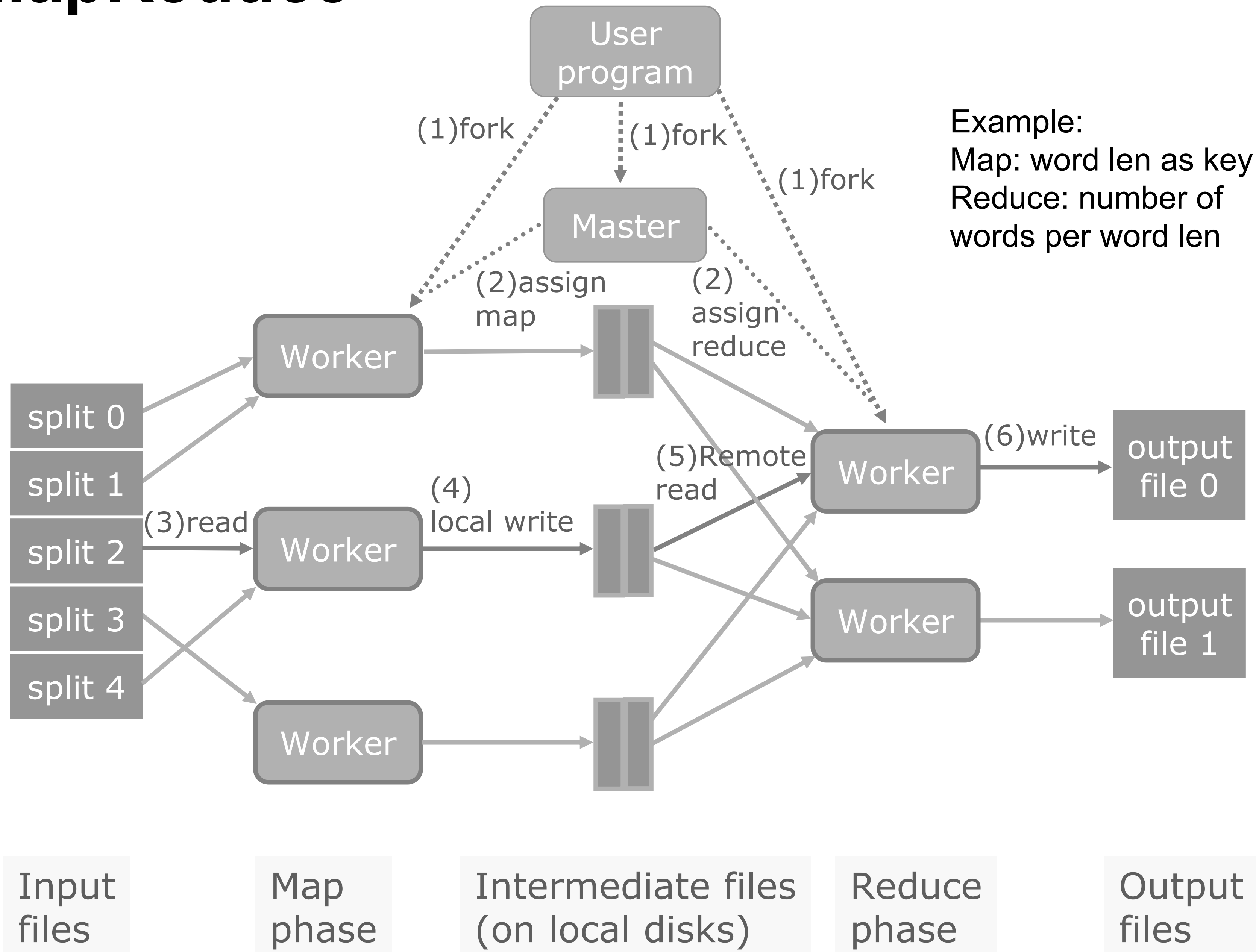
Automatic distribution and parallelization

Fault-tolerance

Cluster management tools

Abstraction for programmers

MapReduce



Example:
Map: word len as key
Reduce: number of words per word len

MapReduce Summary

Two key functions that need to be implemented:

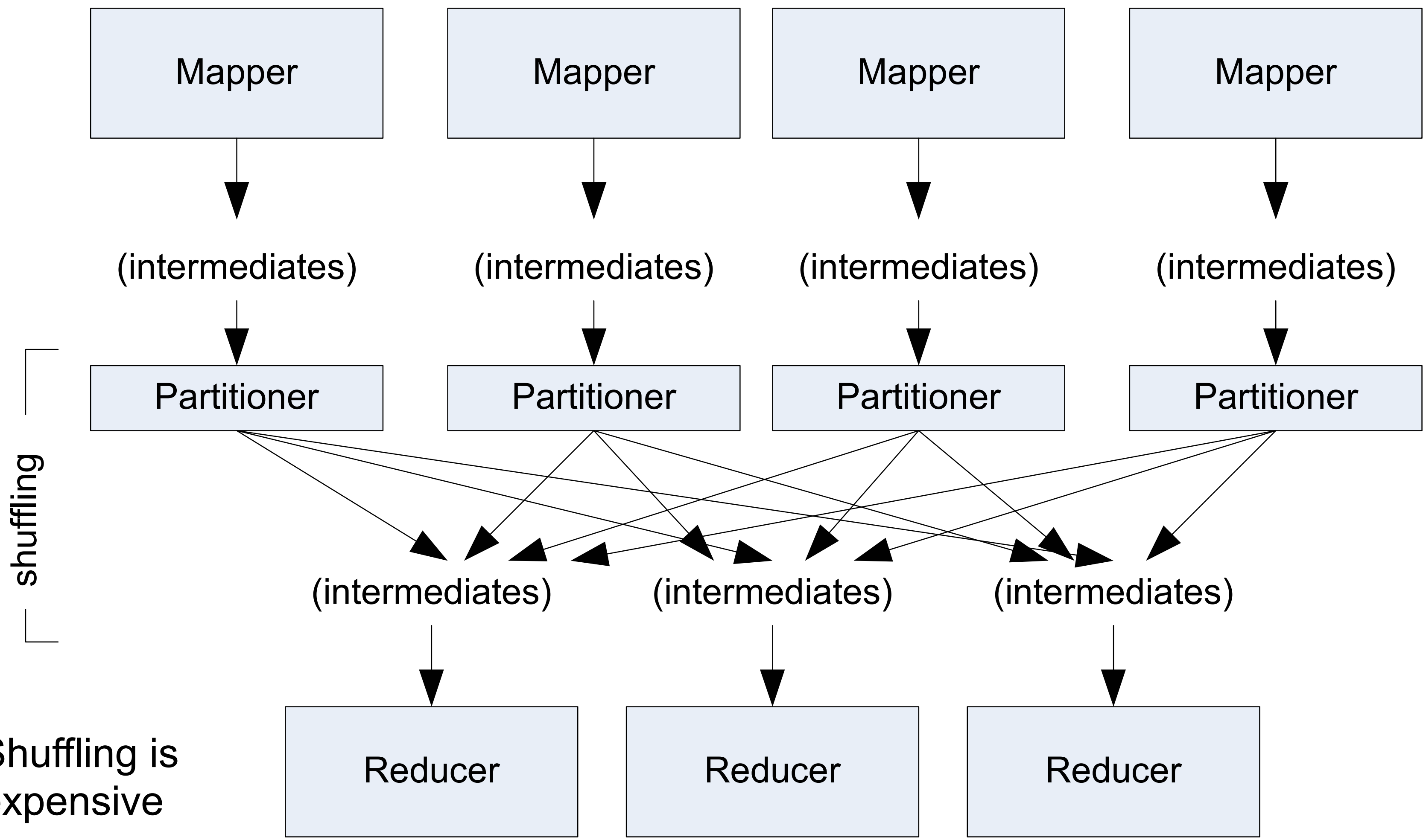
- **map** (in_key, in_value) → (out_key, intermediate_value) list
- **reduce** (out_key, intermediate_value list) → out_value list

With two optimizations:

- **combine** (key, intermediate_value list) → intermediate_out_value list
- **partition** (key, number of partitions) → partition for key

Partition and Shuffle

Data loading is
expensive



Shuffling is
expensive

Hadoop

Hadoop is an Apache open source framework that implements the MapReduce paradigm

Originally created by Yahoo!

Hadoop is based on the HDFS file system

Hadoop has its own RPC protocol

The Hadoop framework includes

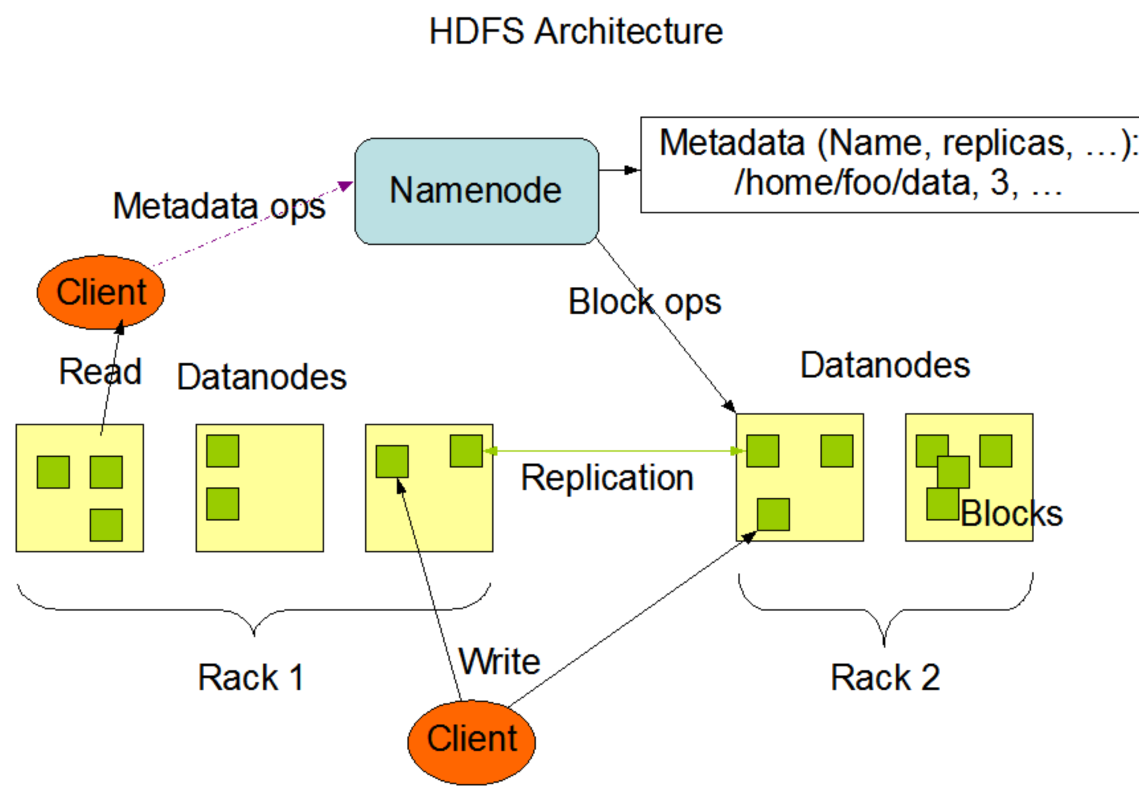
- Apache Pig, Apache Hbase, Apache Hive, Apache Spark, …

Used in production systems by Facebook, Google, Yahoo! And many other companies

http://hadoop.apache.org/

HDFS Architecture

- HDFS has a master/slave architecture
- NameNode is the master server for metadata
- DataNodes manage storage
- A file is stored as a sequence of blocks
- The blocks are replicated for fault-tolerance
- Common replication scheme: factor of 3, one replica local, two in a remote rack
- **Rack-aware replica placement**



Namenode provides information for retrieving blocks
Nearest replica is used to retrieve a block

http://hadoop.apache.org/docs/r1.2.1/images/hdfsarchitecture.gif

Key algorithms for MapReduce

Inverted Index

Statistics

Sorting

Searching

K-Means

Transitive closure

PageRank

Advanced algorithms

K-Means for MapReduce

Map phase

- Each map reads the K centroids and a block from the input dataset
- Each point is assigned to the closest centroid
- **Output: <centroid, point>**

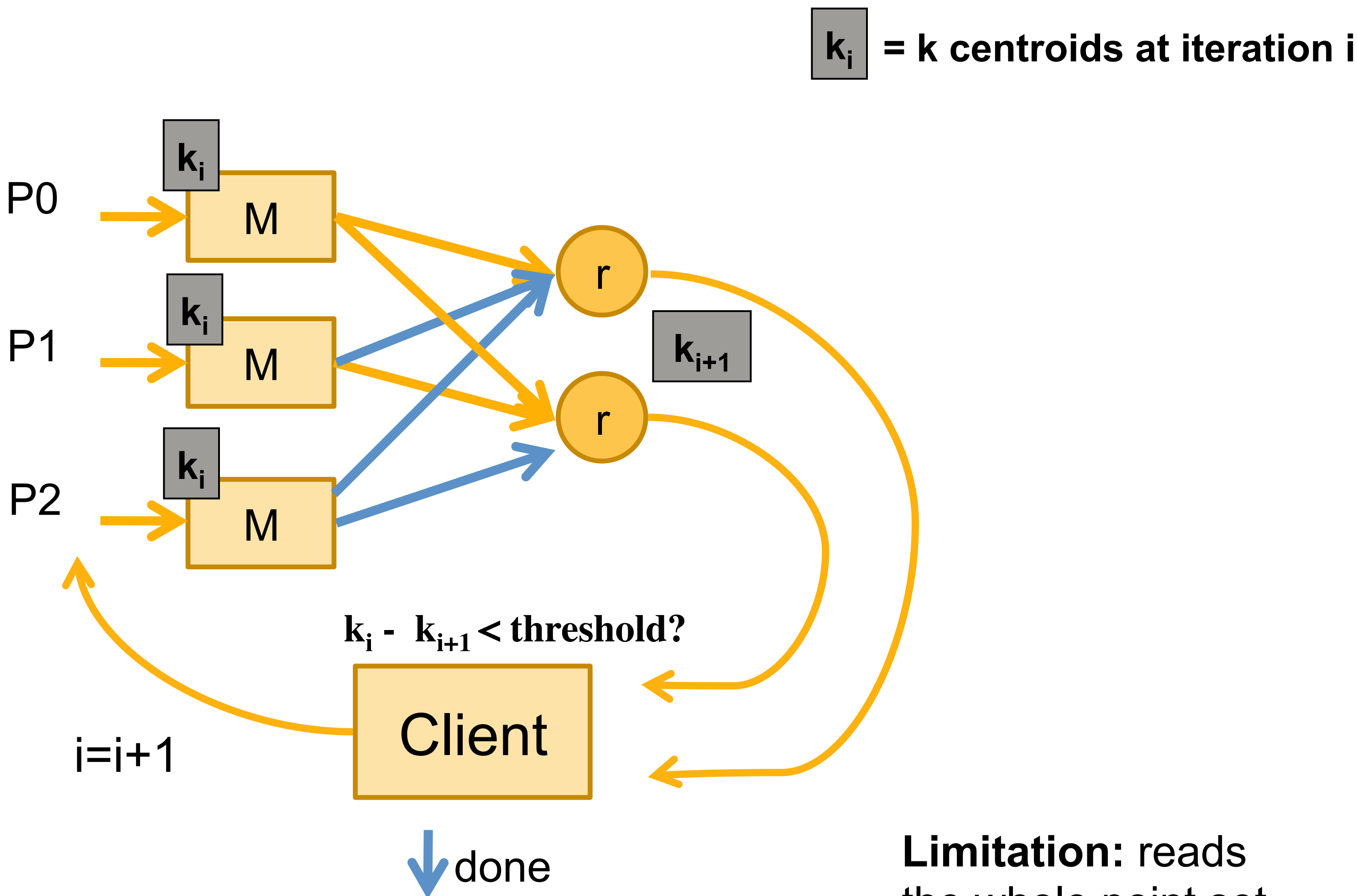
Reduce phase

- Obtain all points for a given centroid
- Recompute the new centroid
- **Output: <new centroid>**

Iteration:

- Compare the old and new set of K centroids
- If they are similar then Stop
- Else Start another iteration unless maximum of iterations has been reached.

MapReduce K-Means



Limitation: reads the whole point set P at each iteration

Source: HaLoop presentation, Yyingyi Bu et al. VLDB 2010

Optimizing K-Means for MapReduce

Combiners can be used to optimize the distributed algorithm

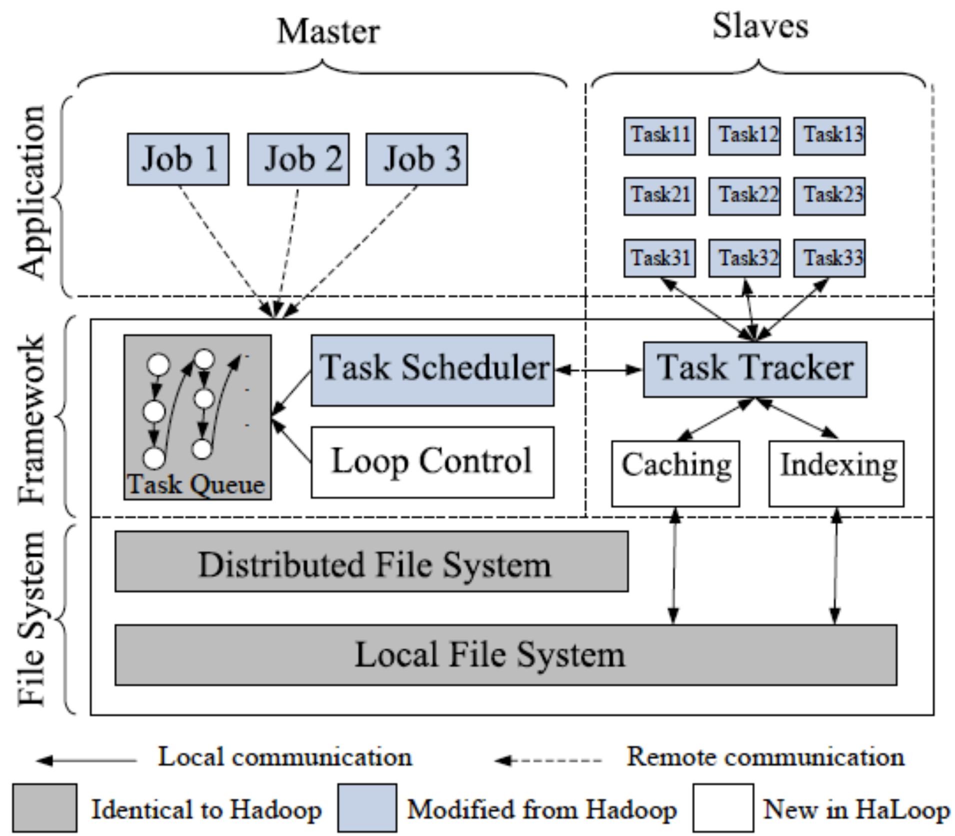
- Compute for each centroid the local sums of the points
- Send to the reducer: <centroid, partial sums>

Use of a single **reducer**

- Data to reducers is very small
- Single reducer can tell immediately if the computation has converged
- Creation of a single output file

HaLoop for iterative MapReduce

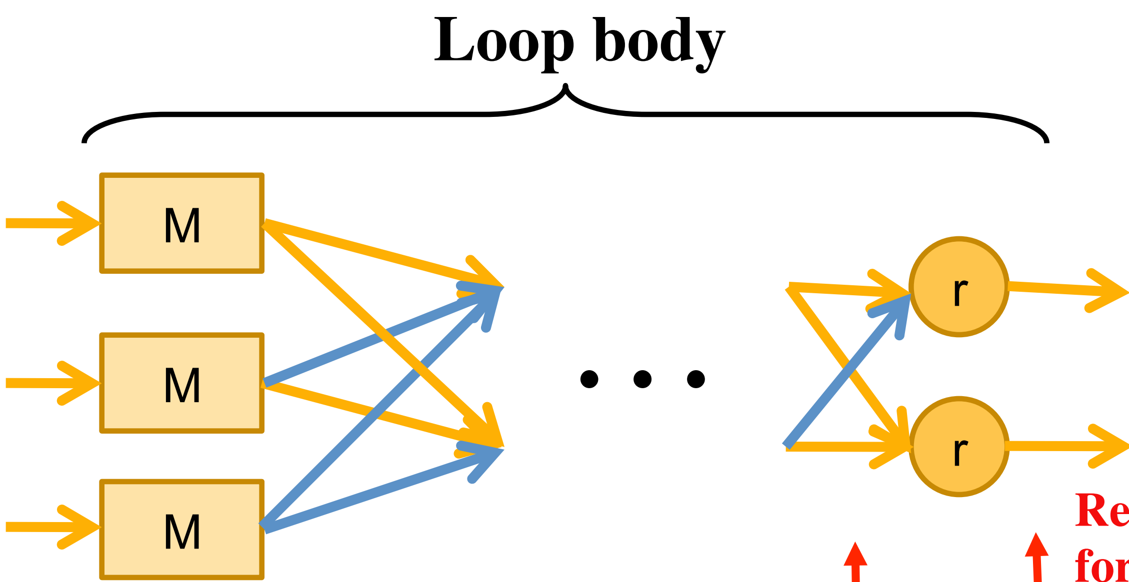
- MapReduce cannot express iteration or recursion
- HaLooP modifies Hadoop for supporting fixpoint operations, loop-aware task scheduling, and cache management
- Map – Reduce – Fixpoint model for recursive languages



For example: the vector of PageRank values of web pages is the fixed point of a linear transformation derived from the link structure

Source: HaLoop presentation, Yyingyi Bu et al. VLDB 2010

HaLoop: Inter-iteration caching



Reducer output cache (RO) for access to output of previous iterations, for fixpoint evaluation

Reducer input cache (RI) for loop invariant data without map/shuffle

Mapper output cache (MO)

Mapper input cache (MI) for access to non-local mapper input on later iterations

Largest gain by caching loop invariant data

Source: HaLoop presentation, Yyingyi Bu et al. VLDB 2010

Apache Spark

- Spark is a general-purpose computing framework for iterative tasks

- API is provided for Java, Scala and Python

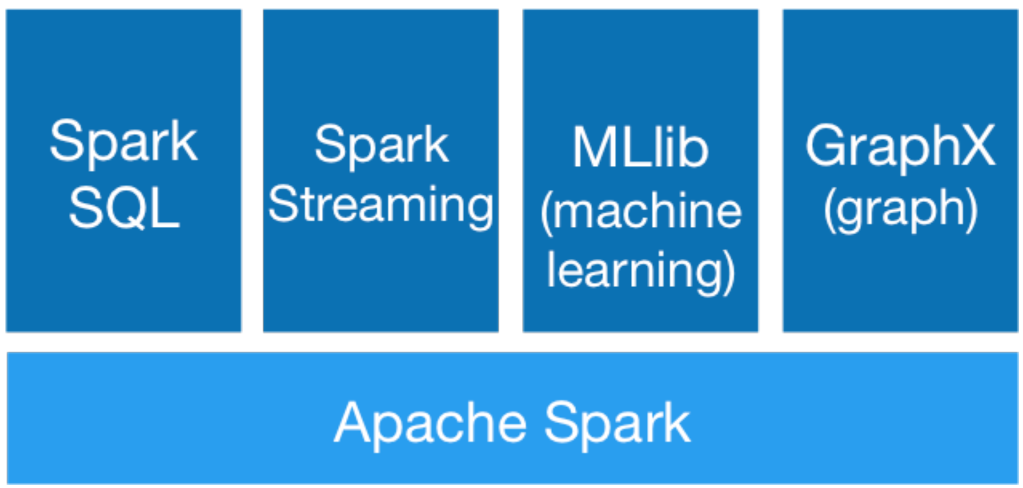
- The model is based on MapReduce enhanced with new operations and an engine that supports execution graphs

- Tools include Spark SQL, MLLlib for machine learning, GraphX for graph processing and Spark Streaming

Spark Aim

Unifies batch, streaming, interactive computing

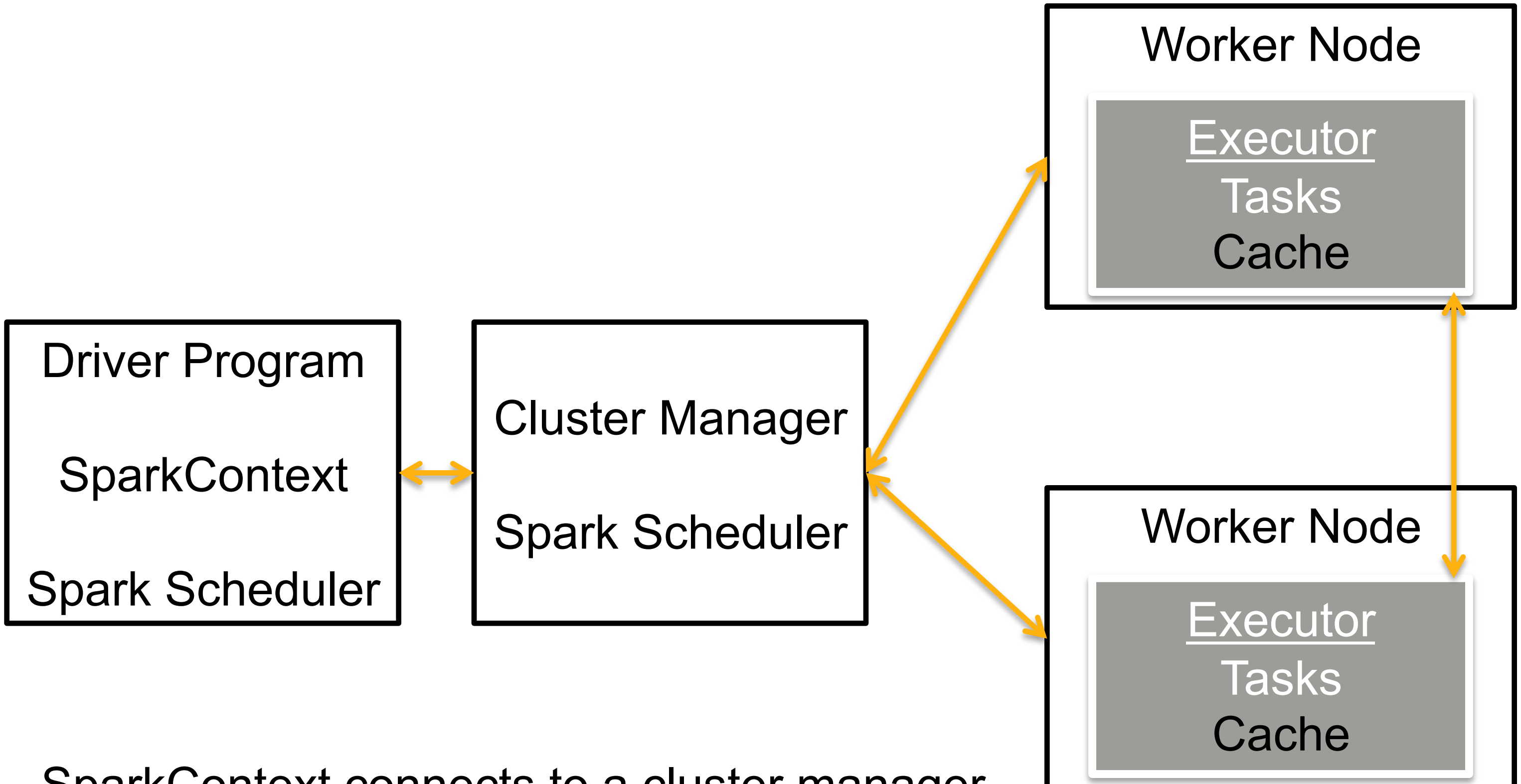
Making it easy to build sophisticated applications



Key idea in Spark

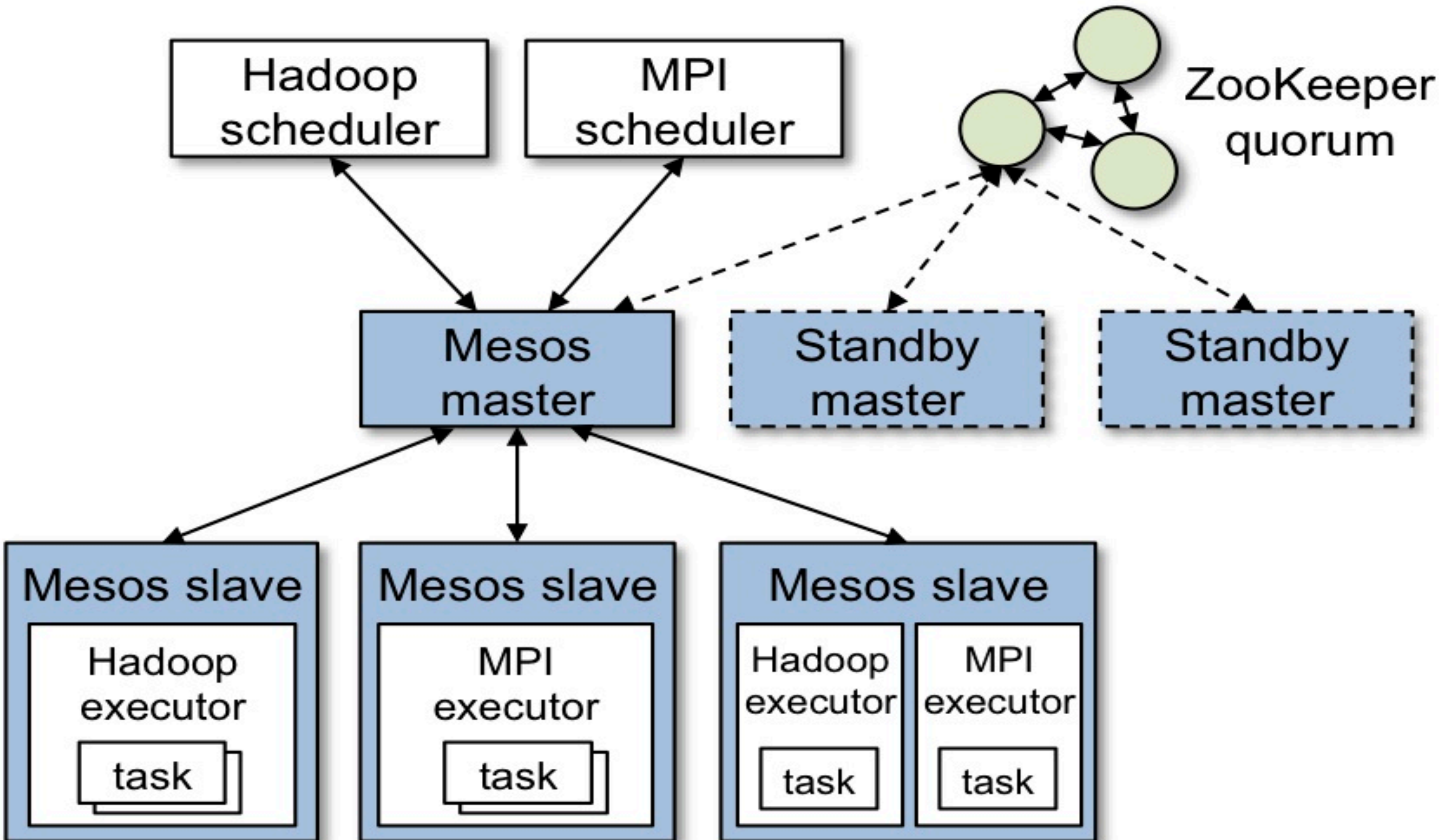
- Resilient distributed datasets (RDDs)
  - Immutable collections of objects across a cluster
  - Built with parallel transformations (map, filter, …)
  - Automatically rebuilt when failure is detected
  - Allow persistence to be controlled (in-memory operation)
- Transformations on RDDs
  - Lazy operations to build RDDs from other RDDs
  - Always creates a new RDD
- Actions on RDDs
  - Count, collect, save

Spark overview



SparkContext connects to a cluster manager
Obtains executors on cluster nodes
Sends app code to them
Sends task to the executors

MESOS Architecture



Task Scheduler

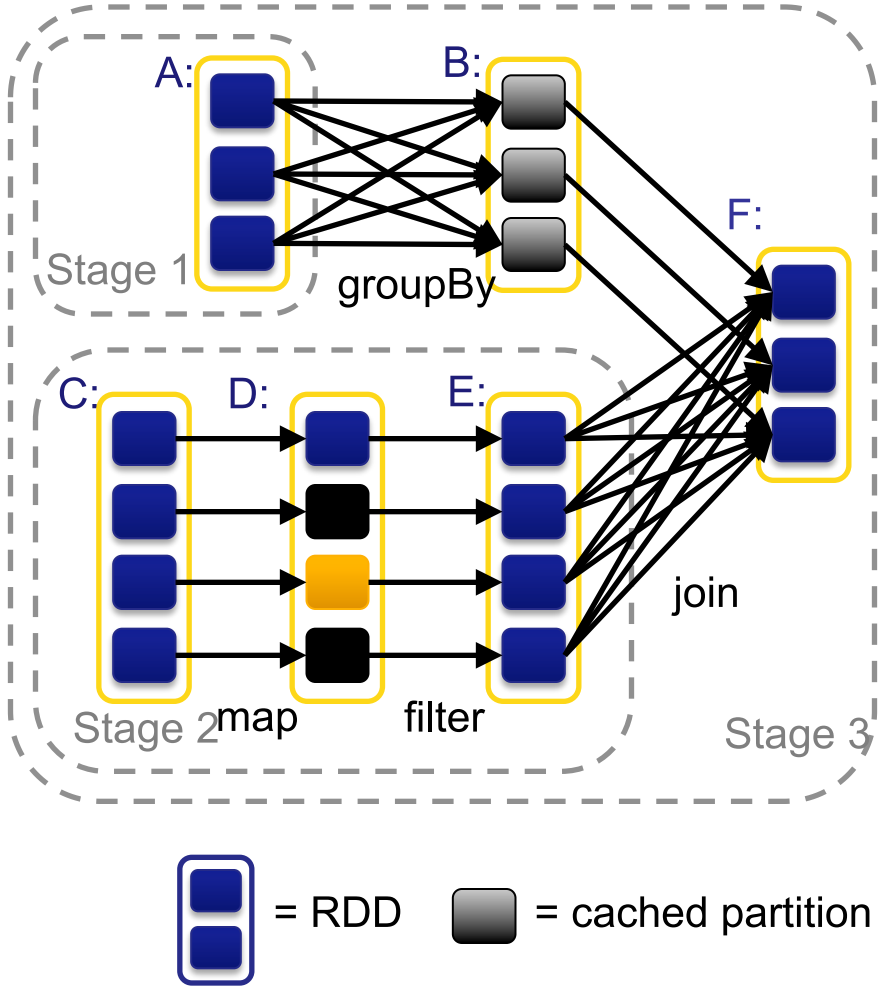
Supports general task graphs

Pipelines functions where possible

Cache-aware data reuse & locality

Partitioning-aware to avoid shuffles

MESOS provides resource allocation (offer resources to framework, accept/reject by framework scheduler)



M.Zaharia. Parallel programming with Spark. O'Reilly Strata Conference. February 2013.

Implementing Spark Algorithms

Broadcast everything

- Master broadcasts data and initial models
- At each iteration updated models are broadcast by master (driver program)
- Does not scale well due to communication overhead

Data parallel

- Worker loads data
- Master broadcasts initial models
- At each iteration updated models are broadcast by master
- Works for large datasets, because data is available to workers

Fully parallel

- Workers load data and they instantiate the models
- At each iteration, models are shared via join between workers
- **Much better scalability**

Machine Learning and Spark

Spark RDDs support efficient data sharing

In-memory caching increases performance

- Reported to have performance of up to 100 times faster than Hadoop in memory or 10 times faster on disk

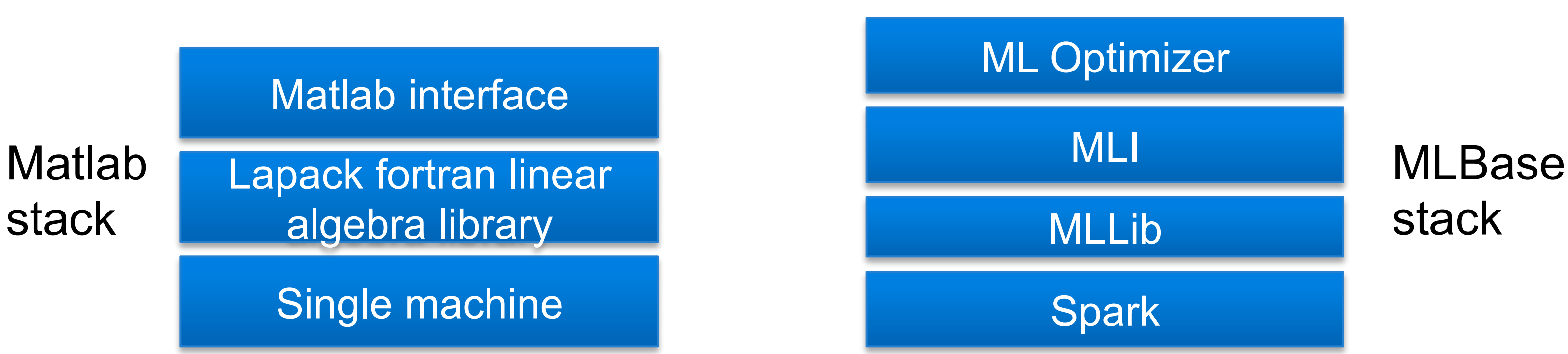
High-level programming interface for complex algorithms

MLBase and MLlib

MLBase has been designed for simplifying the development of machine learning pipelines:

- MLlib is a machine learning library
- MLI (ML Developer API) is an API for machine learning development that aims to abstract low-level details from the developers
- MLOpt is a declarative layer that aims to automate the machine learning pipeline
  - The idea is that the system searches feature extractors and models best fit for the ML task

Source: Towards an Optimizer for MLbase, Ameet Talwalkar, Databricks, 2014.



Graph-Parallel Systems

Graph-based computation depends only on the neighbors of a particular vertex

“Think like a Vertex.” – Pregel (SIGMOD 2010)

Systems with specialized APIs to simplify graph processing

Pregel from Google

Push abstraction: Vertex programs interact by sending messages

Receive msgs, process, send msgs

GraphLab

Pull abstraction: Vertex programs access adjacent vertices and edges

Foreach (j in neighbours) calculate pagerank total for j

GraphX

Separation of system support for each view (table, graph) involves expensive data movement and duplication

GraphX makes tables and graphs views of the same physical data

The views have their own optimized semantics

- Table operators inherited from Spark
- Graph operators form relational algebra

Performance Gains for PageRank Revisited

Spark is reported to be 4x faster than Hadoop

Graphlab is 16x faster than Spark

GraphX is roughly 3x slower than Graphlab

GraphX is reported to compare favourably to Graphlab with pipelines (raw -> hyperlink -> pagerank -> top 20)

Graph structure can be exploited for significant performance gains

Spark Streaming

Spark extension of accepting and processing of streaming high-throughput live data streams

Data is accepted from various sources

Kafka, Flume, TCP sockets, Twitter, …

Machine learning algorithms and graph processing algorithms can be applied for the streams

Similar systems

Twitter (Storm), Google (MillWheel), Yahoo! (S4)

Discretized Streams: A Fault-Tolerant Model for Scalable Stream Processing. Matei Zaharia,Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, Ion Stoica. Berkeley EECS (2012-12-14)

Stream Processing: Discretized

Streaming computation is run as a series of very small deterministic batch jobs

Live stream is divided into **batches of x seconds**

Each batch of data is an RDD and RDD operations can be used

Results are also returned in batches

Batch size as low as 0.5 seconds, results in approx. one second latency

Can combine streaming and batch processing

Discussion

A distributed data operating system is emerging

- Supported by YARN and MESOS

Various data services on top of this (Hadoop and Spark)

Some services are being integrated (for example in Spark) for better coherence and performance

Important points

- Data format (row/column, block size)
- Network topology and data/code placement
- Algorithm structure and coordination
- Scheduling and resource management

Outlook

Big Data Frameworks are evolving

Spark represents unification of streaming, machine learning and graphs

Big Data pipeline management is at an early stage

How to achieve better mapping between cluster resources, scheduling, and pipelines?

Exam material (in addition to slides and exercises)

Articles (part of the exam material):

1. Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. Originally OSDI 2004. CACM Volume 51 Issue 1, January 2008. http://dl.acm.org/citation.cfm?id=1327492
2. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. Matei Zaharia et al. NSDI (2012) usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf
3. HaLoop: Efficient Iterative Data Processing on Large Clusters by Yingyi Bu et al. In VLDB'10: The 36th International Conference on Very Large Data Bases, Singapore, 24-30 September, 2010.
4. MLbase: A Distributed Machine-learning System. Tim Kraska et al. CIDR 2013. http://www.cs.ucla.edu/~ameet/mlbase.pdf

Additional material (not part of the exam):

http://spark.apache.org

http://spark.apache.org/docs/latest/programming-guide.html

www.databricks.com

Grading

Course grading will be based on the final exam and the assignments/exercises.

Exam 60% and exercises 40% of the grade.

- Exam
  - Friday 8.5. 9:00 at B123
  - Exam will have essay questions
  - 4 questions, answer 3

Main theme	Prerequisites	Approaches learning goals	Meets learning goals	Deepens learning goals
Big Data Frameworks: definitions and systems	Basics of data communications and distributed systems (Introduction to Data Communications, Distributed Systems)	Knowledge of how to define the concepts of MapReduce and variants and state their central features Ability to describe at least one system in detail	Ability of being able to compare different Big Data frameworks in a qualitative manner Ability to assess the suitability of different systems to different use cases	Ability to give one's own definition of the central concepts and discuss the key design and deployment issues
Internal operation and implementation of a Big Data framework	Basics of data communications and distributed systems (Introduction to Data Communications, Distributed Systems) Big-O-notation and basics of algorithmic complexity Basics of reliability in distributed systems	Knowledge of the design and implementation level concepts of Big Data frameworks, specifically Hadoop and Spark. Knowledge of how distributed state is maintained and synchronized. Understanding of the communication and computational costs in Big Data processing. Ability to describe at least one algorithm in detail	Ability of being able to compare different Big Data frameworks based on their design and implementation. Ability of designing distributed Big Data systems building on existing frameworks for batch and streaming processing. Knowledge of key performance issues and the ability to analyze these systems Knowledge of the most important factors pertaining to reliability	The knowledge of designing a Big Data platform for a given problem Familiarity with the state of the art
Distributed algorithms for Big Data frameworks	Basics of algorithm design and machine learning	Knowledge of the basic design of a distributed algorithm for MapReduce and Spark. Ability to use graph processing and machine learning in a distributed cluster environment	Ability to design and implement a solution that uses distributed algorithms for a large dataset Ability to create both batch and streaming solutions	Design and implementation of a new machine learning algorithm for Big Data Familiarity with the state of the art
Data Science applications	-	Knowledge of the basic Data Science use cases based on Big Data frameworks	Knowledge of at least two Data Science use cases and how they use the Big Data framework Knowledge of Data Science pipelines	Familiarity with the state of the art Automation of Data Science pipelines