2014.9.30

Bio Bigdata

BICube min-kyung Kim

강남 토즈 타워점 401호

Outline

- **Bigdata**
- Hadoop
- Spark
- 3rd pary Biobigdata
- Machine Learning
- Streamng
- Future

- Bigdata

What is Bigdata?

- Data is massive, unstructured, and dirty
- Data sets so large and complex
- Difficult to process using traditional data processing applications
- Massively parallel software running on tens, hundreds, or even thousands of servers

- What is Bigdata?

Why Biobigdata?

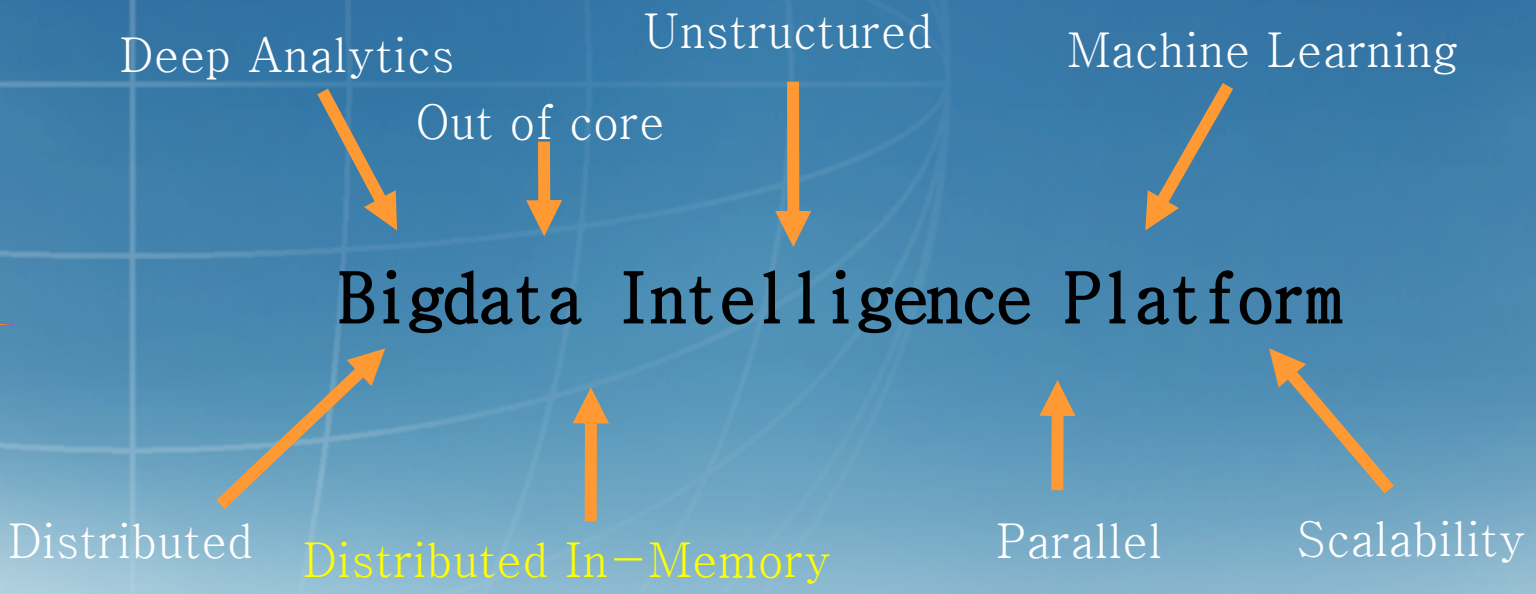
- Asian Genome:3.3Billion 35bp 104GB
- Afrian Genome:4.0Billion 35bp 144GB
- 1000Genomes=5,994,000 process

=23,976,000 hours

=2737 years

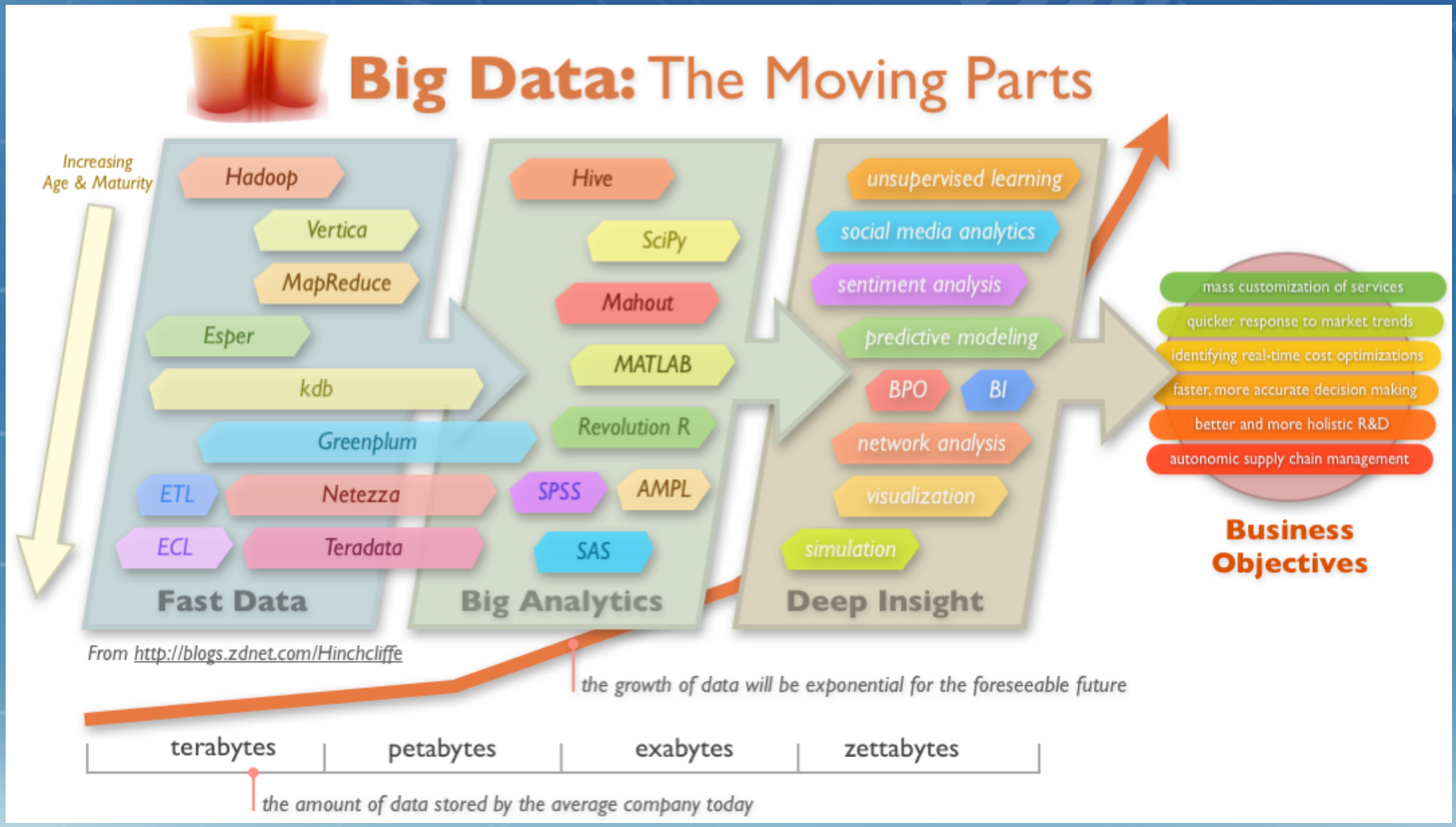
by Deepak Singh

- Bigdata
  - Rethink algorithms
  - Rethink Computing
  - Rethink Data Management
  - Rethink Data Sharing



Bigdata

More Accurate Decision Making



- Bigdata

Why Bigdata?

Central limit theorem

the arithmetic mean of a sufficiently large number of iterates of independent random variables

Overfitting, Six sigma

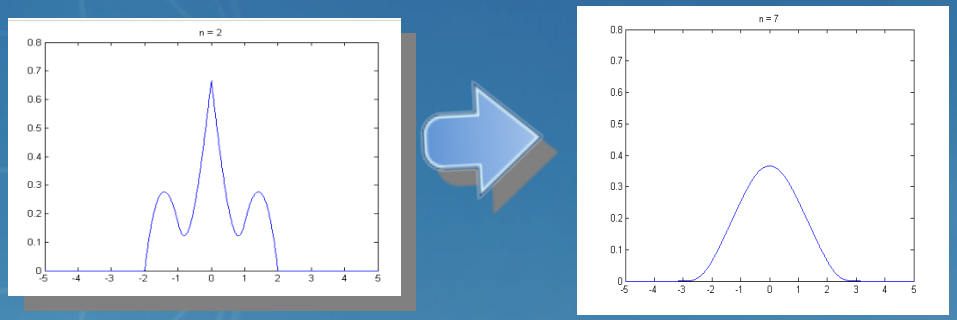
So bigdata is not necessary !

Large data is enouph

But ….

Why Markov Assumtions?

Why Independent Assumptions?



Outline

- Bigdata
- **Hadoop**
- Spark
- Future
- Future
- Future
- Future

Hadoop

What is Hadoop?

- A open-source software for **reliable, scalable, distributed** computing.

- A framework that allows for the distributed **processing of large data sets** across clusters of computers using simple programming models

- scale up from single servers to **thousands** of machines, each offering local computation and **storage**.

- Hadoop

Included modules(Hadoop 2)

- Hadoop Common: Utilities that support the other Hadoop modules.
- Hadoop Distributed File System (HDFS™): A distributed file system
- Hadoop YARN: A framework for job scheduling and cluster resource management.
- Hadoop MapReduce: A YARN-based system for parallel processing of large data sets.

- Hadoop

Hadoop Distributed Filesystem(HDFS) Overviewing

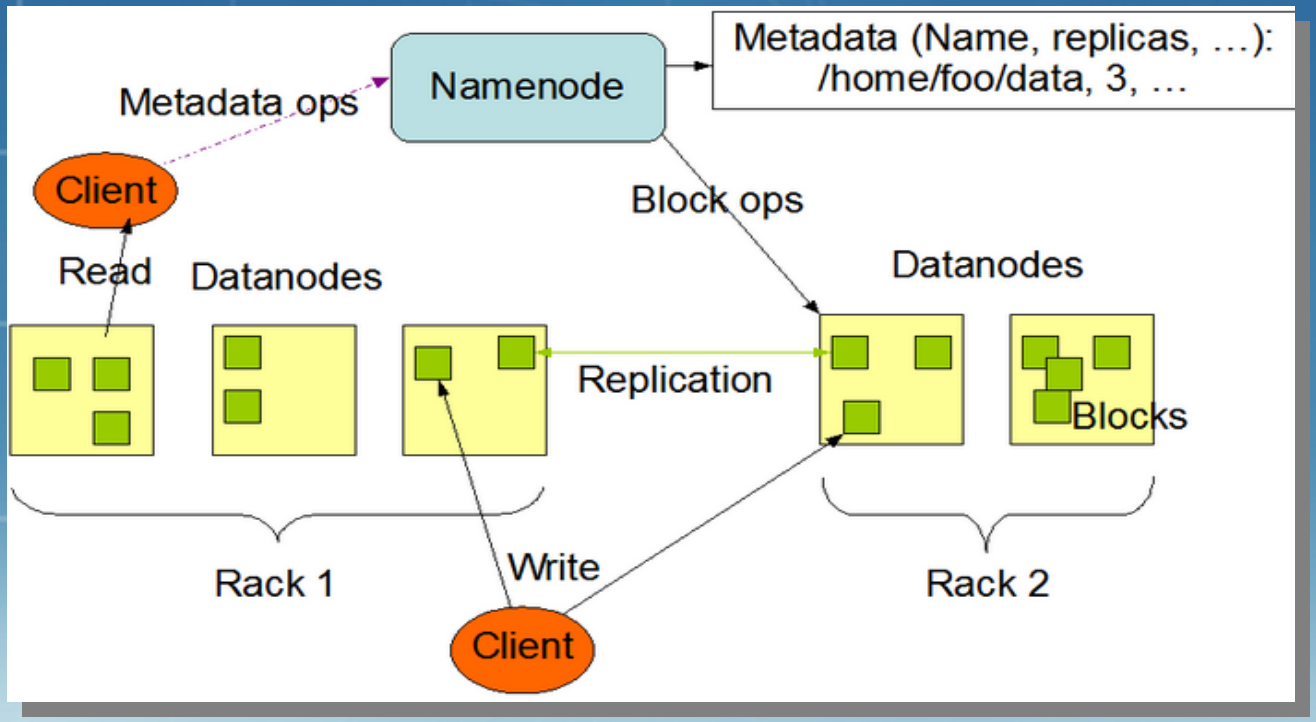
- Run on Commodity Server
- The Idea of "Write onece, Read many times"
- Large Streaming Reads
- High Throughtput is more important than low latency

more important than

- Hadoop

HDFS Architecture

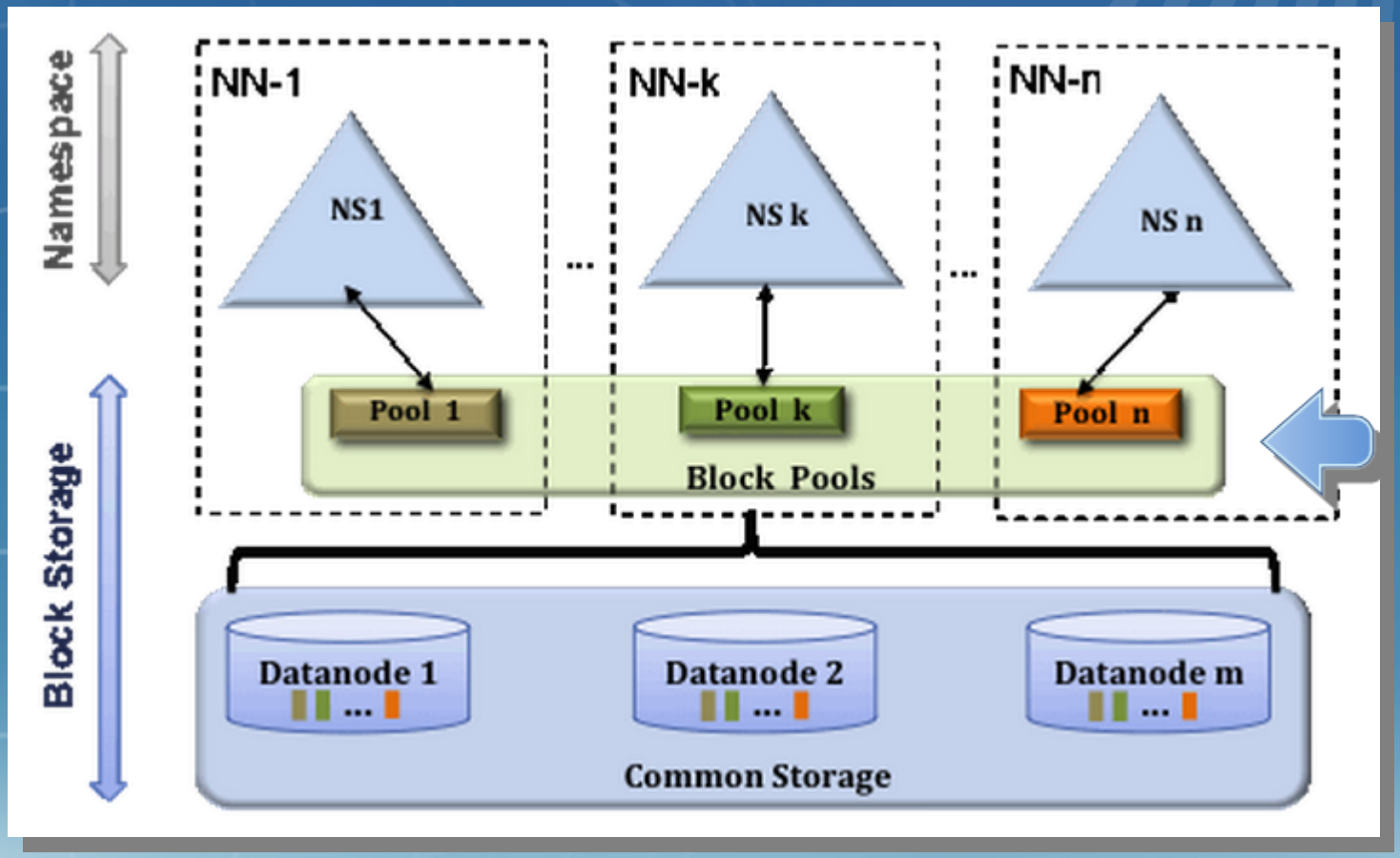
- Stored as Block
- Provide Reliability through Replication
- NameNode stores Metadata and Manage Access
- No data caching(Large dataset)



- Hadoop

HDFS Federation

- Block management



- Hadoop

File Storage

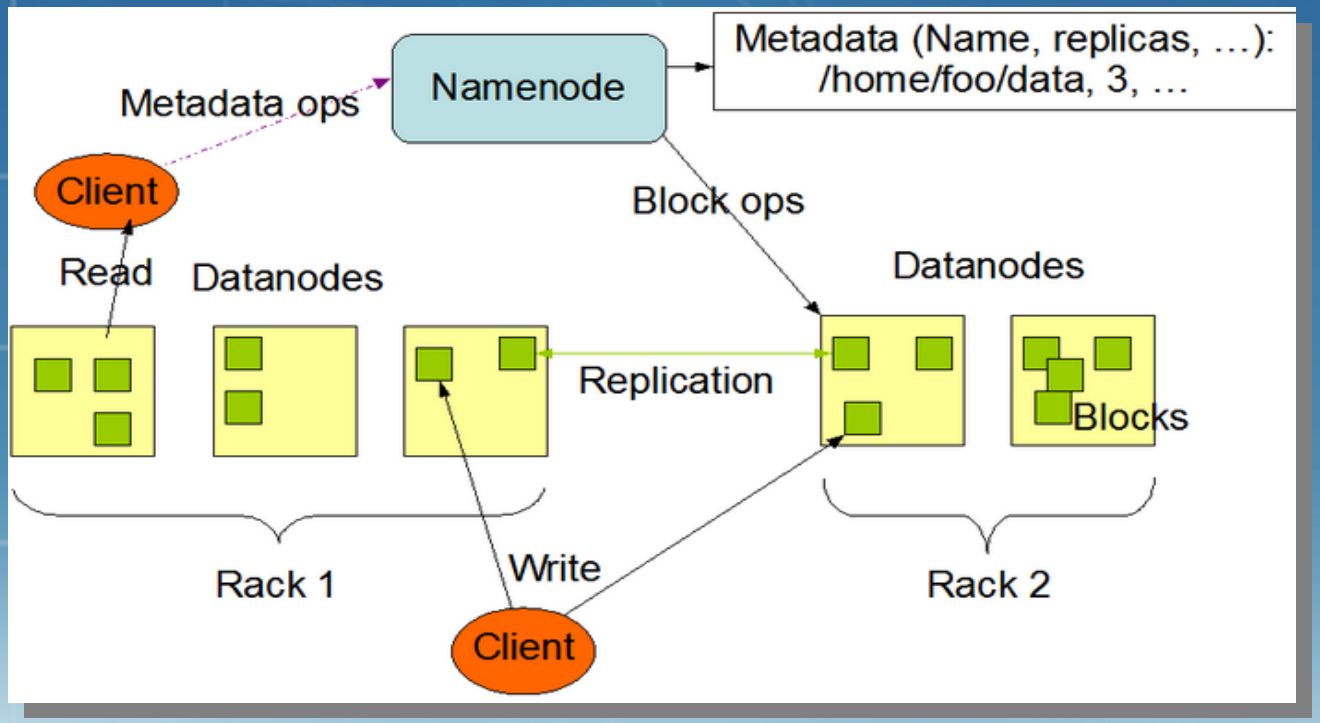
- NameNode
  - Metadata: **filename**, Location of **block**, file **attribute**
  - keep metadata in RAM
  - Metadata Size is limited to the amount of the Name Node RAM size
- DataNode
  - Store file contents as block
  - Same file is stored on Different DataNode
  - Same Block is replicated across Several DataNode
  - Periodically sends a report to the NameNode

exchange heartbeats with each nodes
Client always read from nearest node

- Hadoop

Internals of file read

- Client is guided by NameNode
- Client directly contacts DataNode
- Scale to large number of clients



- Hadoop

MapReduce

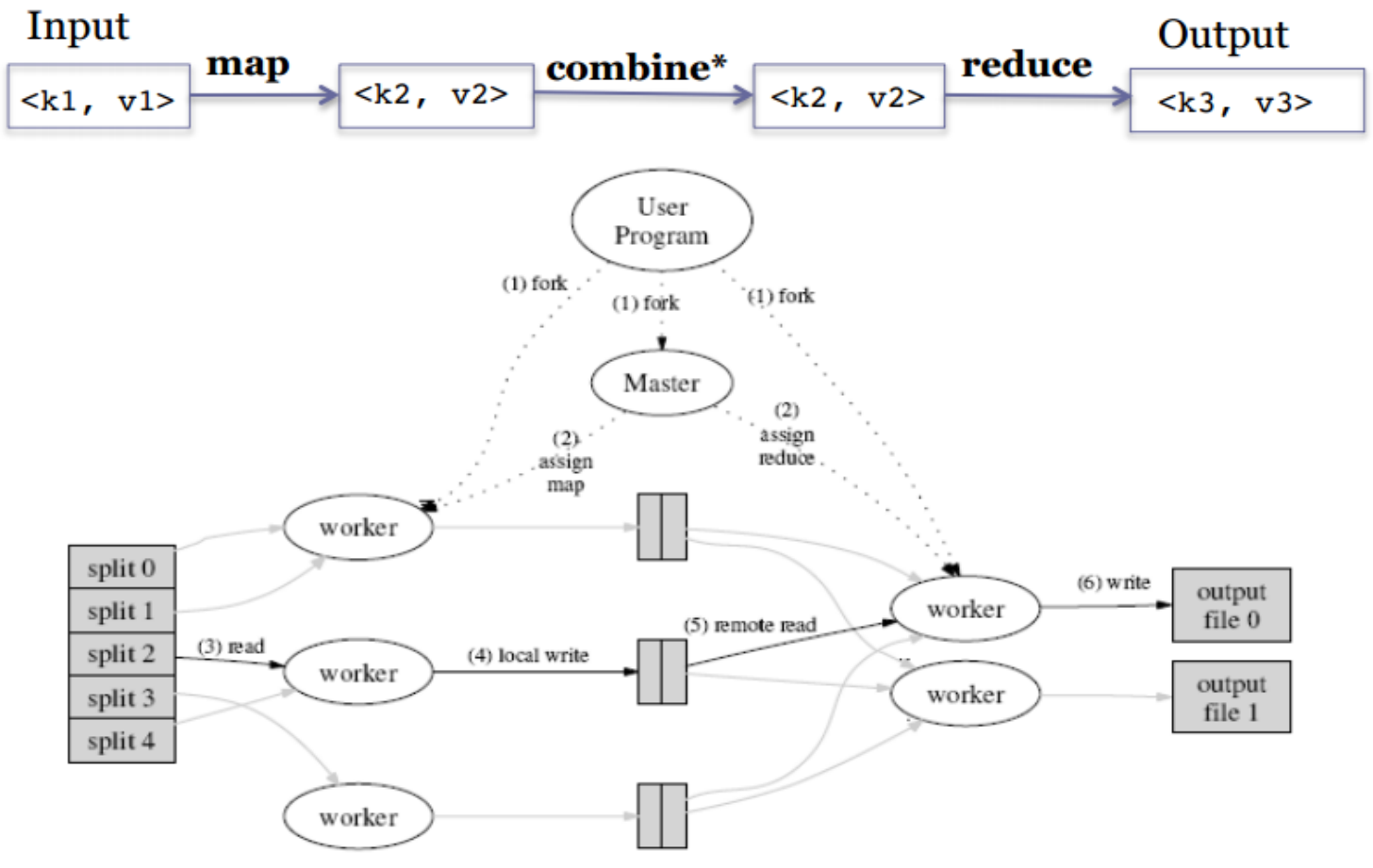
- Method for distributing a task
- Each node processes data on the self node
- Two phases : map, reduce

  Map : (K1,V1) → (K2,V2)

  Reduce : (K2, list(V2)) → list(K3,V3)
- Automatic parallelization
- Fault-tolerance
- Usually writen in java

- Hadoop

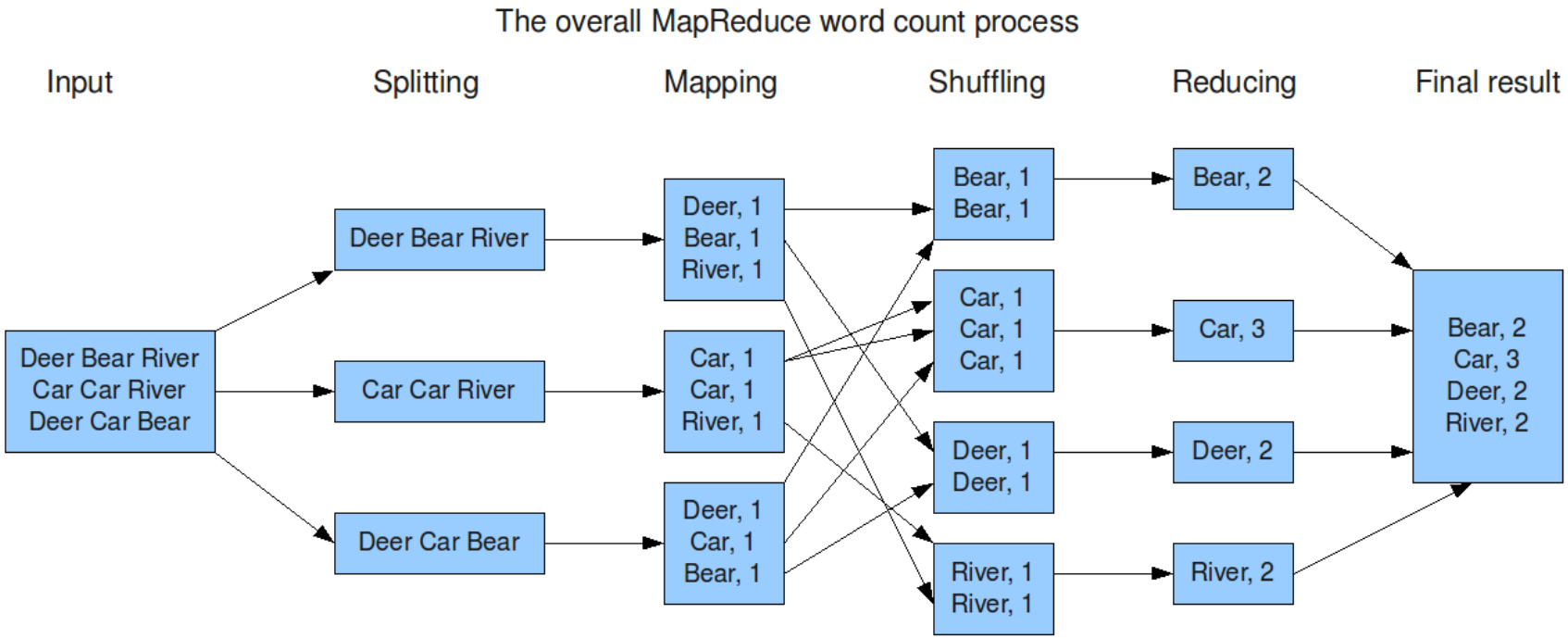
MapReduce: Google



From http://code.google.com/edu/parallel/mapreduce-tutorial.html

- Hadoop

MapReduce



shuffling : mix recuder input, sorting

- Hadoop

MapReduce

```
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {
      public static void main(String[] args){

      }
}
```

- Hadoop

MapReduce

```
public class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) throws
        IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
}
```

- Hadoop

MapReduce

```
public  class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

MapReduce

```
public class WordCount {
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = new Job(conf, "wordcount");

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
  }
}
```

- Hadoop

FS Shell

Options

cat, chgrp, chmod, chown, copyFromLocal, copyToLocal, count, cp, du, dus, expunge, get, getmerge, ls, lsr, mkdir, moveFromLocal, moveToLocal, mv, put, rm, rmr, setrep, stat, tail, test, text, touchz

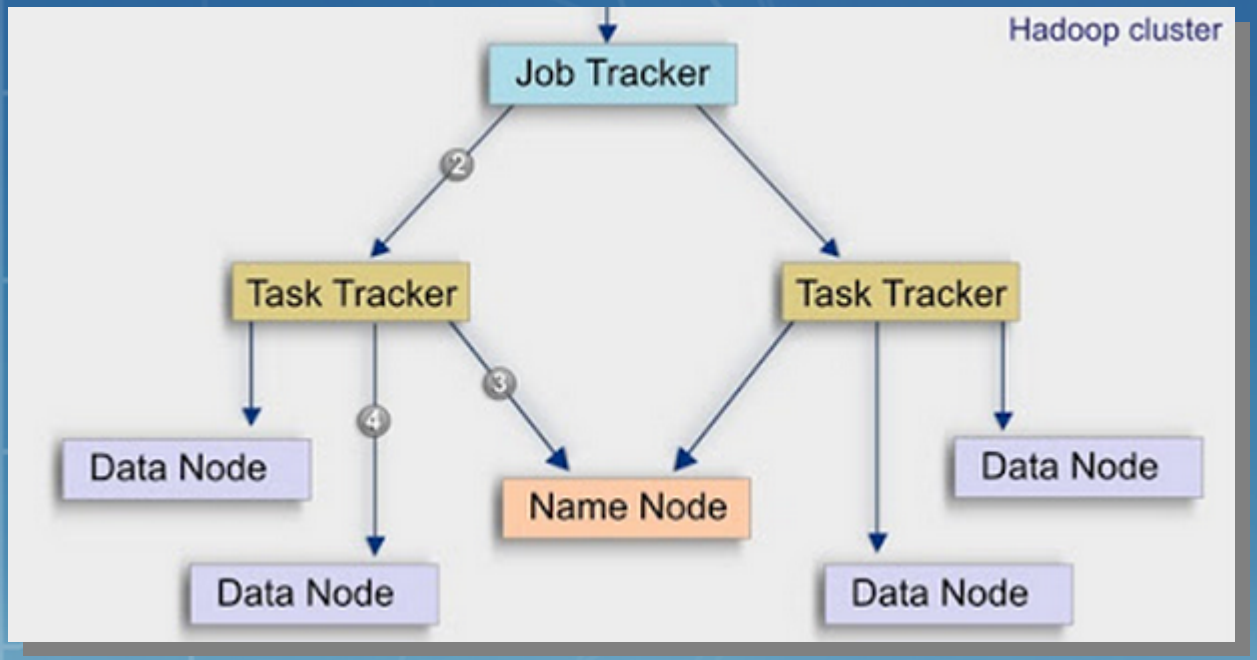
hadoop fs -cat /user/hadoop/file

hadoop fs -put localfile /user/hadoop/hadoopfile

- Hadoop

MapReduce(Hadoop 1)

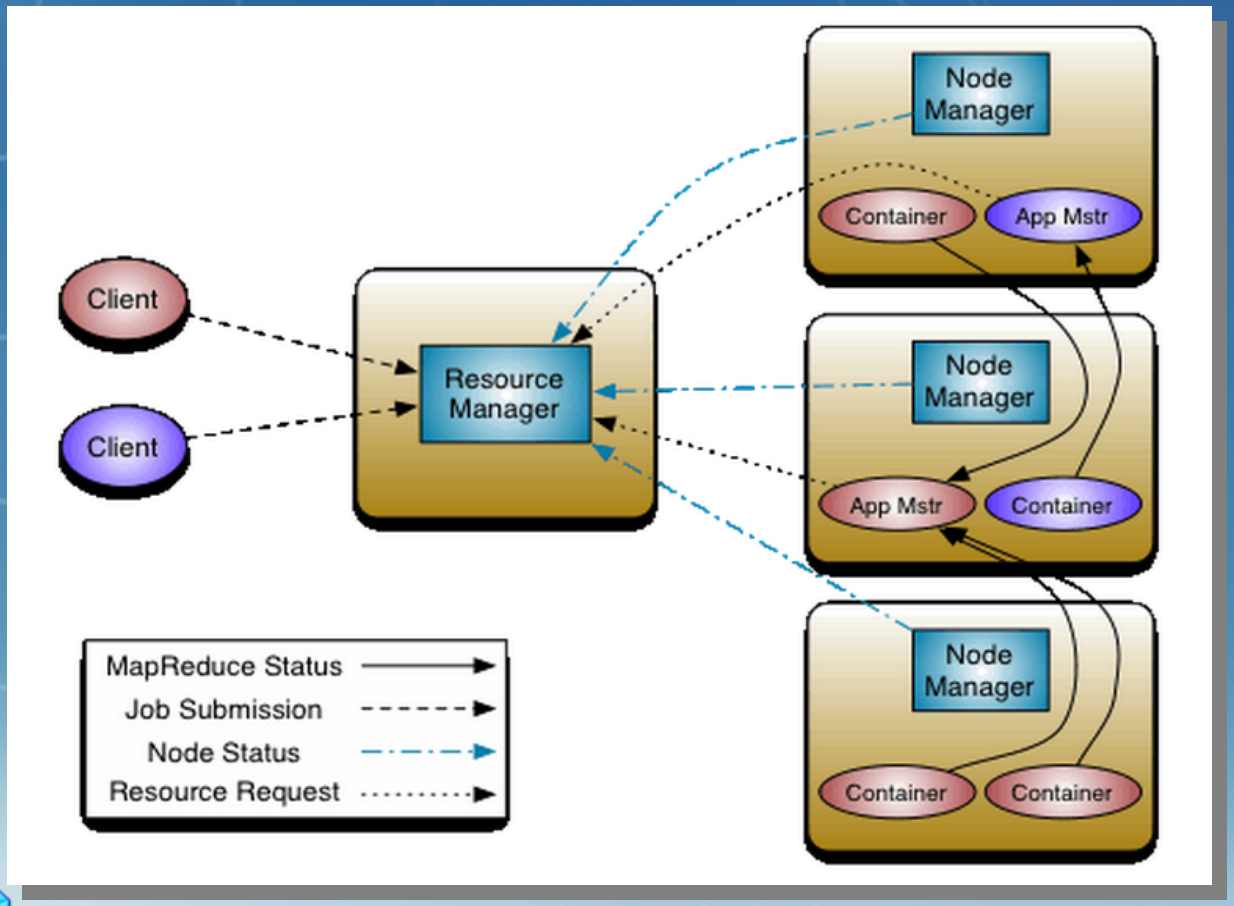
- Jobtracker
- TaskTracker

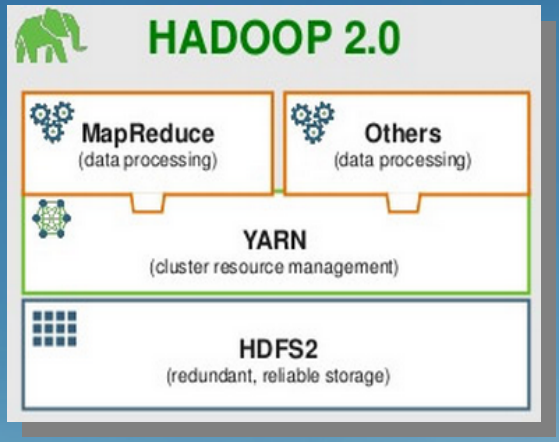


Hadoop

NextGen MapReduce (YARN:Hadoop 2)

- Jobtracker → ResourceManager
- TaskTracker → NodeManager





- Hadoop
  - Hadoop Streamming
    - Utility
    - None java programming
    - Use sdtin, stdout
    - **Only Text**

- Hadoop

Python MapReduce using Hadoop Streaming
use sys.sdtin to read data and sys.stdout to print data

$cat data | mapper.py | sort | reducer.py

```
cat mobydick.txt | ./mapper.py | sort | ./reducer.py >
out/mobydick_cooccur_local.txt

head | out/mobydick_cooccur_local.tx

hadoop jar $HADOOP_HOME/hadoop-streaming.jar \
$PARAMS \
-mapper mapper.py \
-reducer reducer.py \
-input cooccur_example/mobydick.txt \
-output cooccur_example/mobydick_out \
-file mapper.py -file reducer.py
```

Python Streaming Mapper

```
/**************************************************************
**********************mapper .py*****************************
**************************************************************/
#!/usr/bin/env python
import sys, re

for line in sys.stdin:
    # remove non alpha-numeric
    line = re.sub("[^a-zA-Z0-9]", " ", line)

    # remove leading and trailing whitespaces
    line = line.strip()

    # split the line into words
    words = line.split()

    # increase counters
    for word in words:
        # tab delimited; this word happend one time
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

Python Streaming Reducer

```
/**********************************************************
**********reducer.py**************************/
#!/usr/bin/env python
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # parse the mapper input
    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count wasn't a numbe, so we
            # (silently) discard the line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
                # write result to STDOUT
                print '%s\t%s' % (current_word, current_count)
            current_count = count
            current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

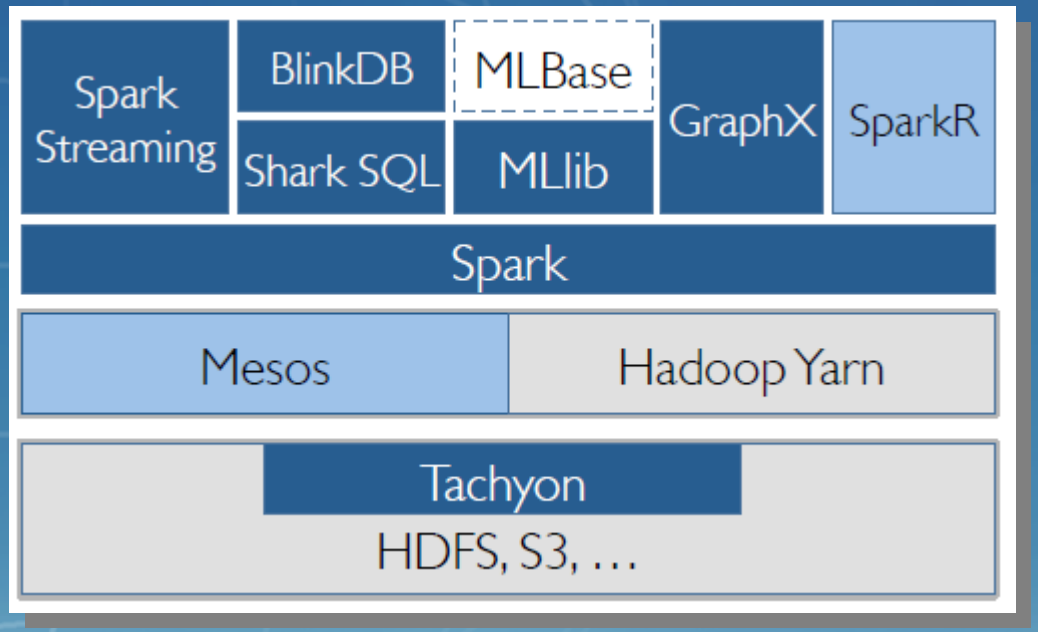
- Outline

- Bigdata
- Hadoop
- **Spark**
- 3rd party
- Machine Learning
- Streaming
- Future

- Spark

What is Spark

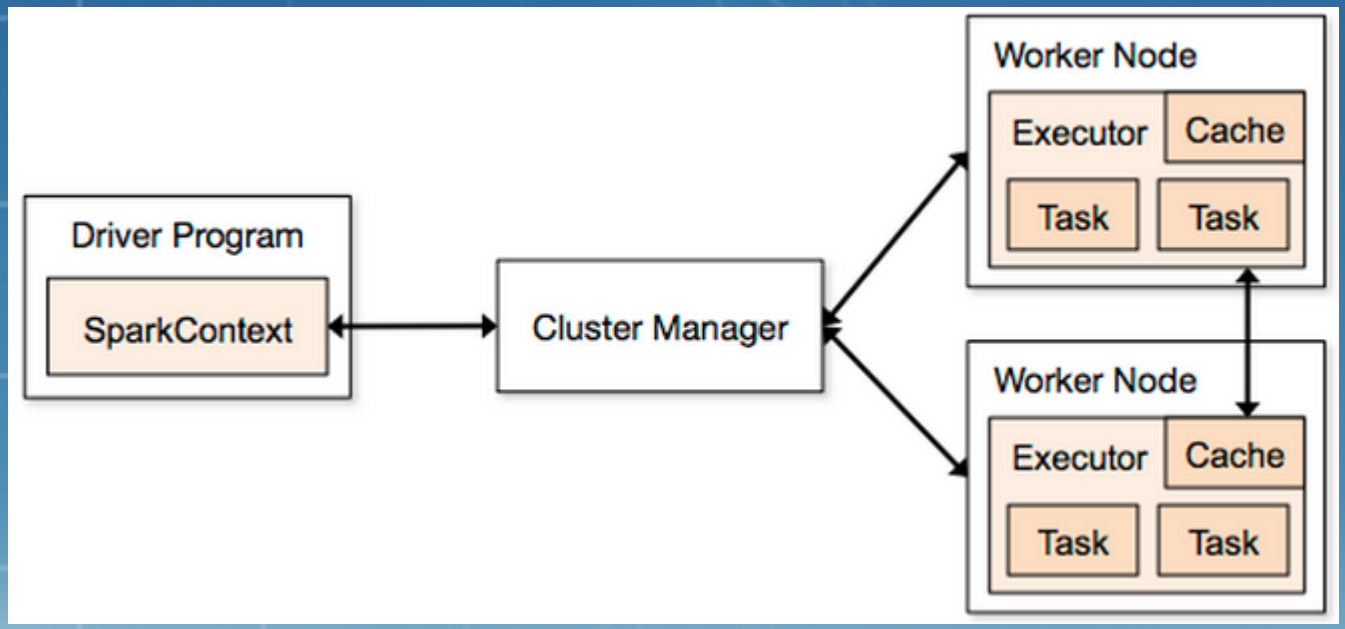
- A fast and general engine for large-scale data processing
- Spark 1.1.0 (latest release)



Spark

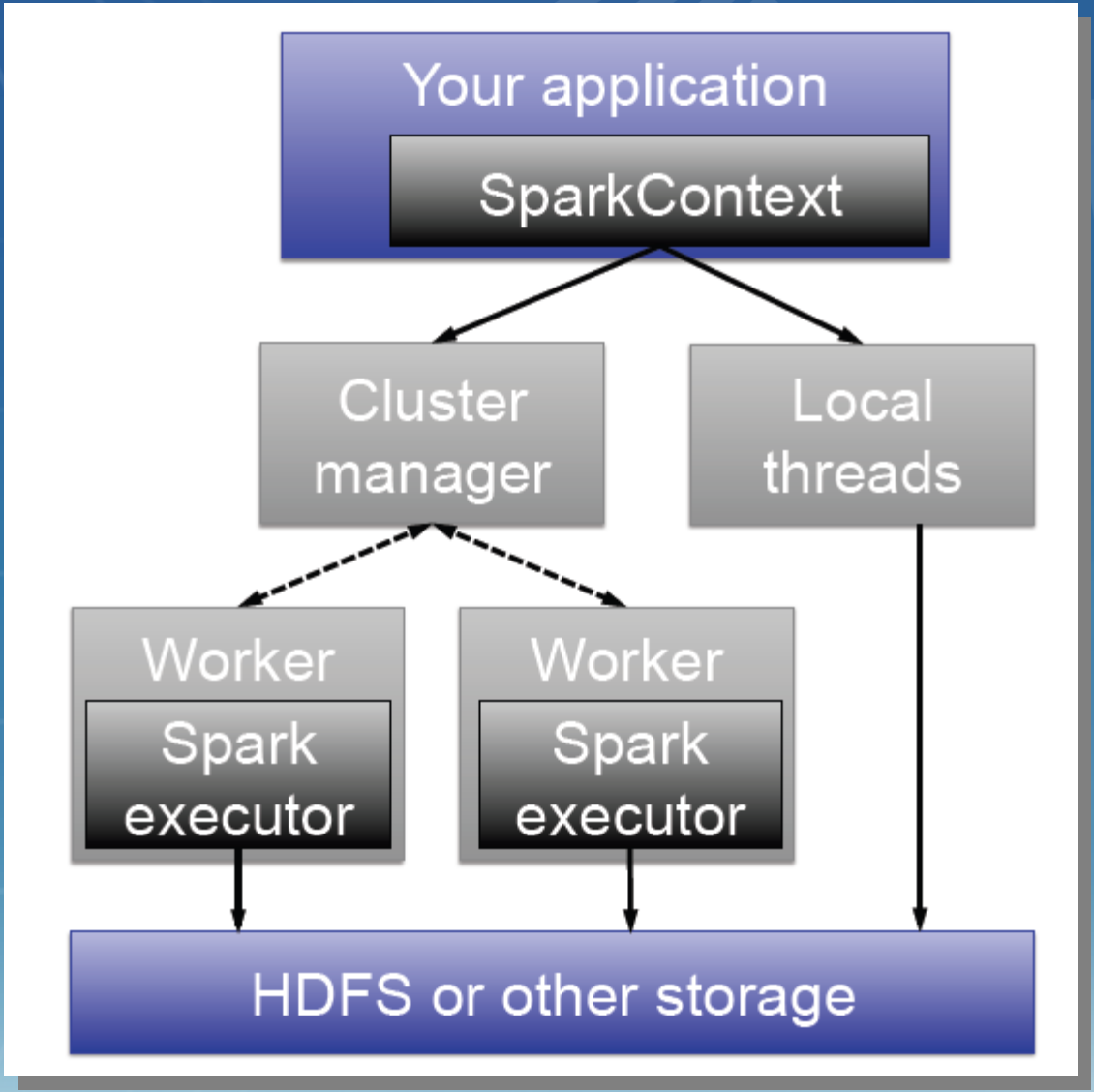
Process

```
val conf = new SparkConf().setAppName("Simple Application")
val sc = new SparkContext(conf)
val textFile = sc.textFile("README.md")
```



- Spark

Components:



RDD

Resilient Distributed Datasets

- Collections of objects spread across a cluster, stored in RAM or on Disk
- Built through parallel transformations
- Automatically rebuilt on failure
- Standard RDD operations–map,countByValue,reduce,join
- Stateful operations–window,countByValueAndWindow,…

- Spark

Language Support

Python

```
lines = sc.textFile(...)
lines.filter(lambda s: “ERROR” in s).count()
```

Scala

```
val lines = sc.textFile(...)
lines.filter(x => x.contains(“ERROR”)).count()
```

Java

```
JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
  Boolean call(String s) {
    return s.contains(“error”);
  }
}).count();
```

Standalone Programs

- Python, Scala, & Java

Interactive Shells

- Python & Scala

Performance

- Java & Scala are faster due to static typing
- …but Python is often fine

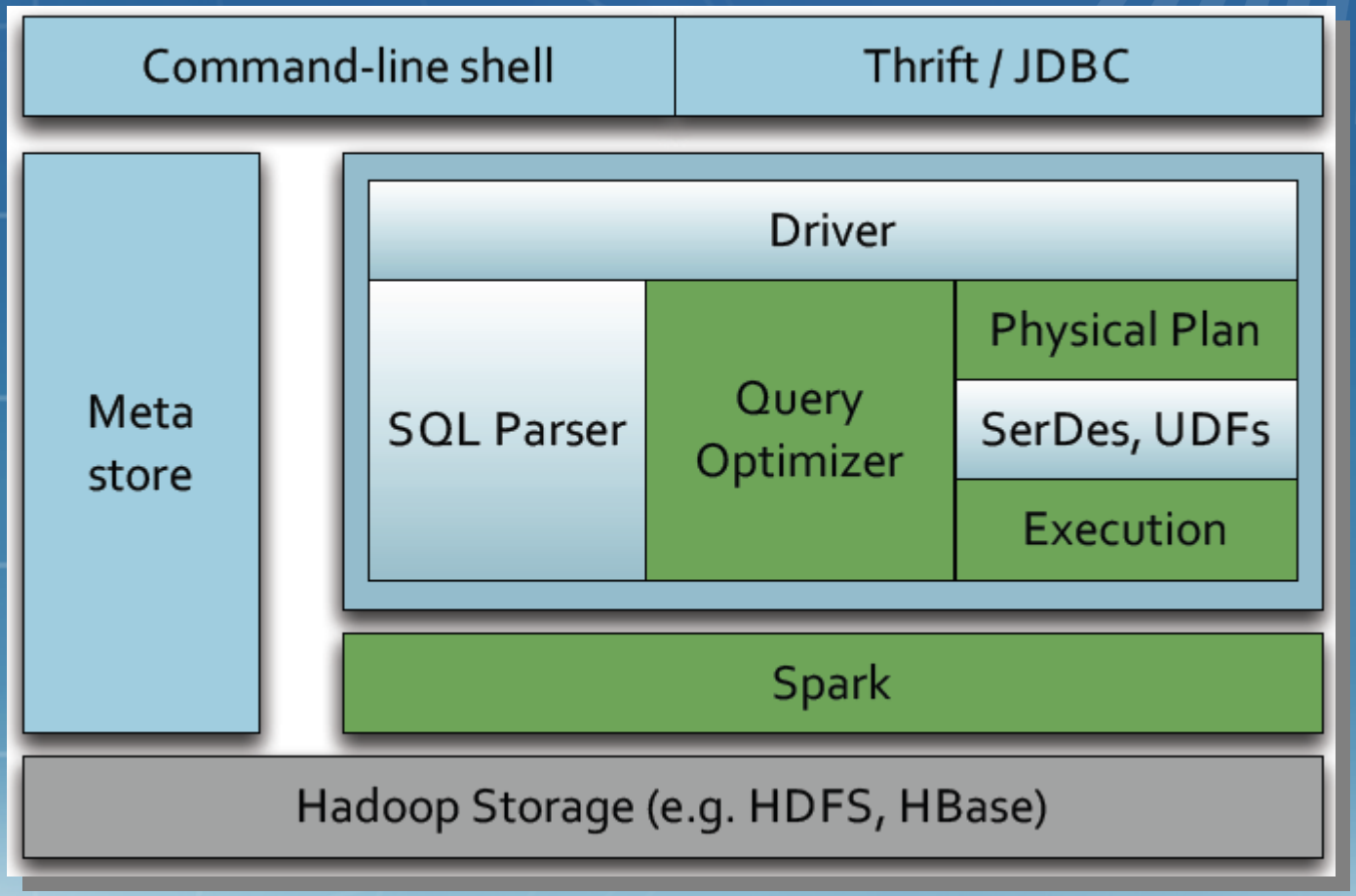
- Spark

Spark SQL

- **Unifies access** to structured data
- integrated APIs in Python, Scala and Java

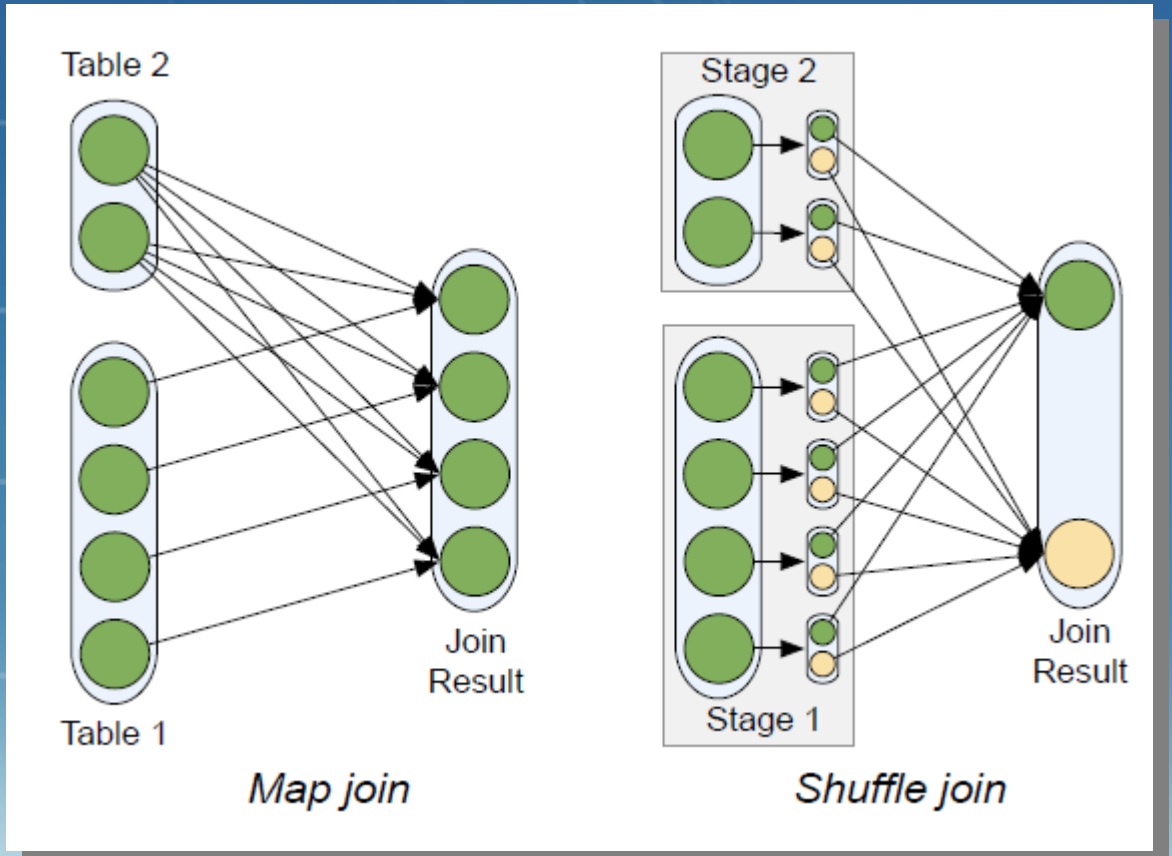
```
sqlCtx = new HiveContext(sc)
results = sqlCtx.sql(
      "SELECT * FROM people")
names = results.map(lambda p: p.name)
```

Spark SQL
Shark Architecture



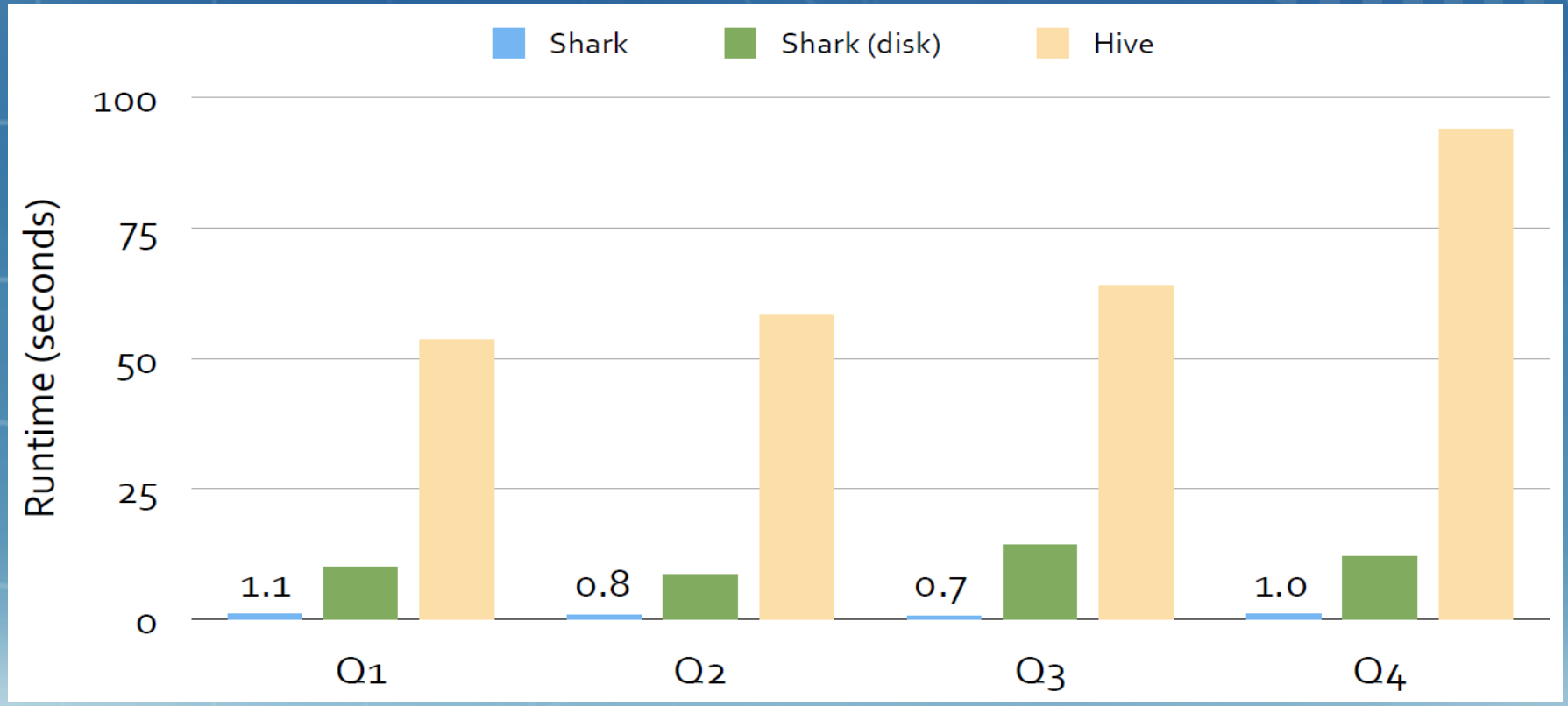
- Spark

Spark SQL
Shark
Partial DAG Execution (PDE)



- Spark

Spark SQL
Shark
Compare to hive



- Spark

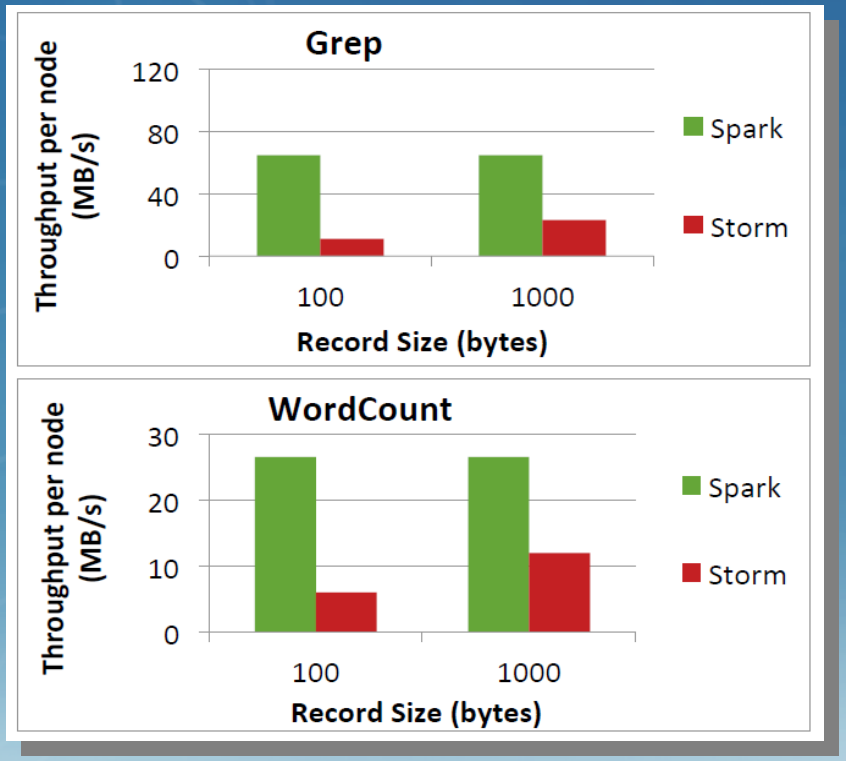
Spark SQL
Shark
Compare to Impala

	Shark	Impala
Focus	integrate SQL with complex analytics	data warehouse / OLAP
Execution	Spark (MapReduce like)	Parallel Databases
In-memory	in-memory tables	no (buffer cache)
Fault-tolerance	tolerate slave failures	no
Large (out-of-core) joins	yes	no
UDF	yes	no

- Spark

Spark Streaming

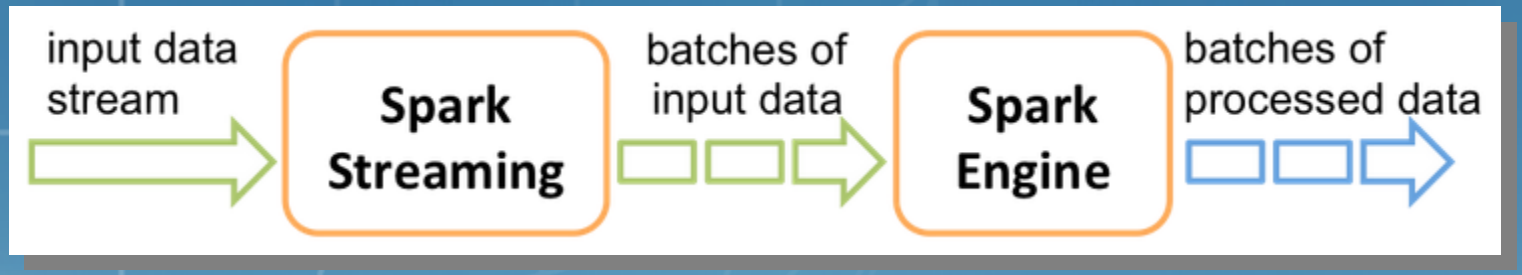
- makes it easy to build **scalable fault-tolerant streaming** applications.
- powerful **interactive** applications
- provide results in **near-real-time**



Spark Streaming

```
import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._


val conf = new SparkConf().setMaster("local[2]").setAppName("NetworkWordCount")
val ssc = new StreamingContext(conf, Seconds(1))
val lines = ssc.socketTextStream("localhost", 9999)
```



Batch to memory : Mini batch

Spark Streaming

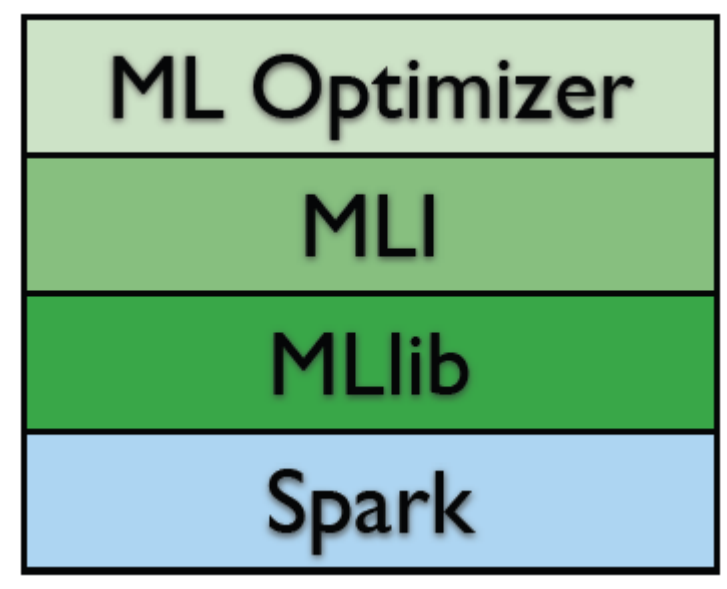
- Kafka spark-streaming-kafka_2.10
- Flume spark-streaming-flume_2.10
- Kinesis spark-streaming-kinesis-asl_2.10 [Apache Software License]
- Twitter spark-streaming-twitter_2.10
- ZeroMQ spark-streaming-zeromq_2.10
- MQTT spark-streaming-mqtt_2.10

- Spark

MLlib

scalable machine learning library Usable in Java, Scala and Python,100x faster than MapReduce.

- linear models (SVMs, logistic regression, linear regression)
- decision trees
- naive Bayes
- alternating least squares (ALS)
- k-means
- singular value decomposition (SVD)
- principal component analysis (PCA)
- Feature extraction and transformatio
- stochastic gradient descent
- limited-memory BFGS (L-BFGS)



- Spark

GraphX

API for graphs and graph-parallel computation.

- Graph-parallel primitives on Spark.
- Currently slower than GraphLab, but

No need for specialized systems

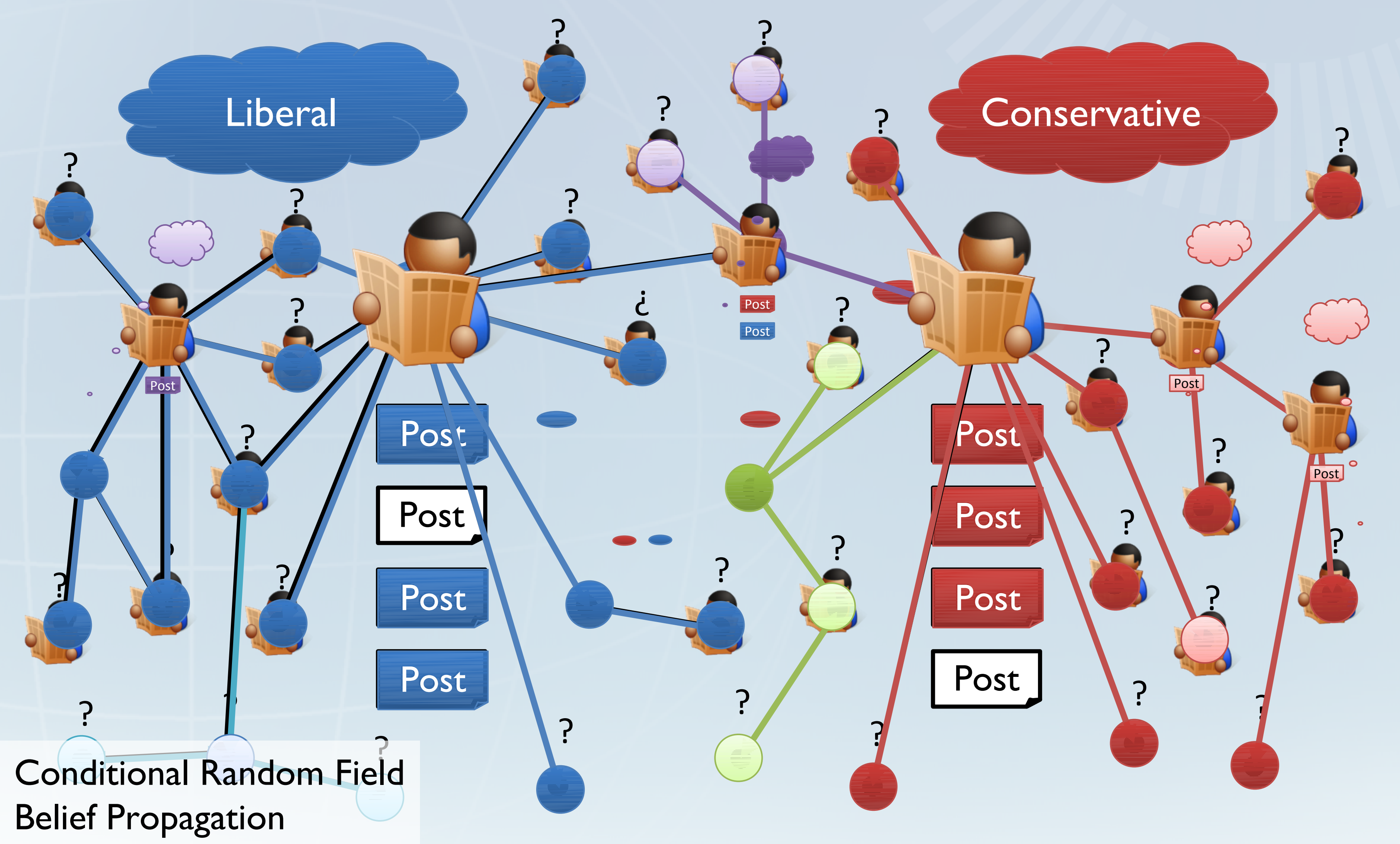
- Easier ETL, and easier consumption of output

Interactive graph data mining

- Future work will bring performance closer to specialized engines.

Spark

GraphX



Conditional Random Field
Belief Propagation

- Spark

GraphX

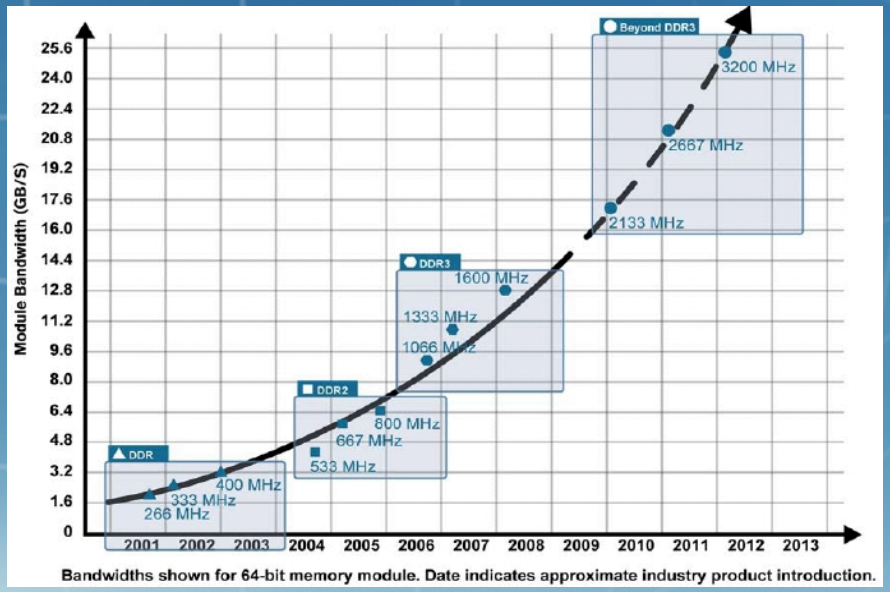
More Graph algorithms

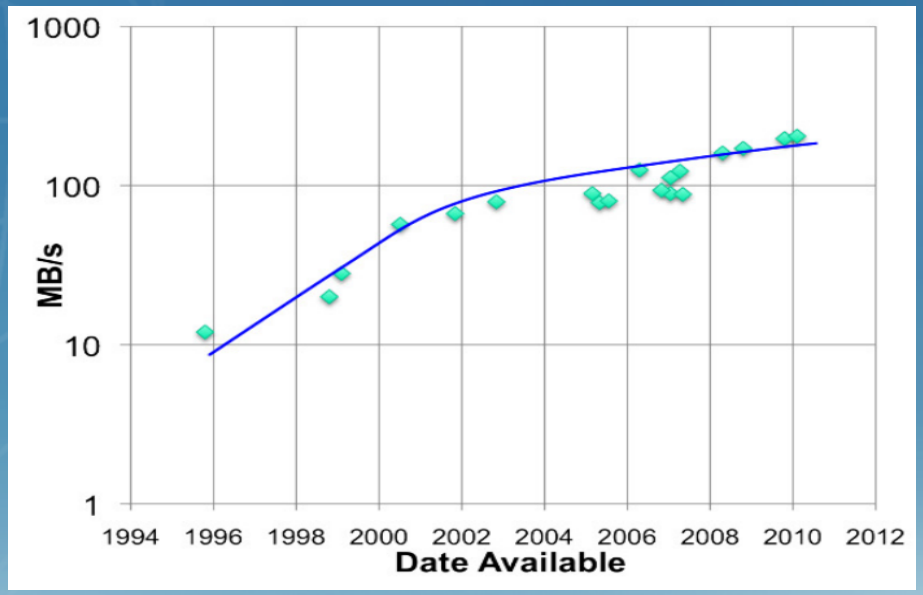
- Collaborative Filtering
  - Alternating Least Squares
  - Stochastic Gradient Descent
  - Tensor Factorization
- SVD
  - Structured Prediction
  - Loopy Belief Propagation
  - Max-Product Linear Programs
  - Gibbs Sampling
- Semi-supervised ML
  - Graph SSL
  - CoEM
- Graph Analytics
  - PageRank
  - Single Source Shortest Path
  - Triangle-Counting
  - Graph Coloring
  - K-core Decomposition
  - Personalized PageRank
- Classification
  - Neural Networks
  - Lasso

- Spark

Tachyon

- High-throughput, fault-tolerant in-memory storage
- Interface compatible to HDFS
- Further improve performance for Spark, and Hadoop
- Growing community with 10+ organizations contributing





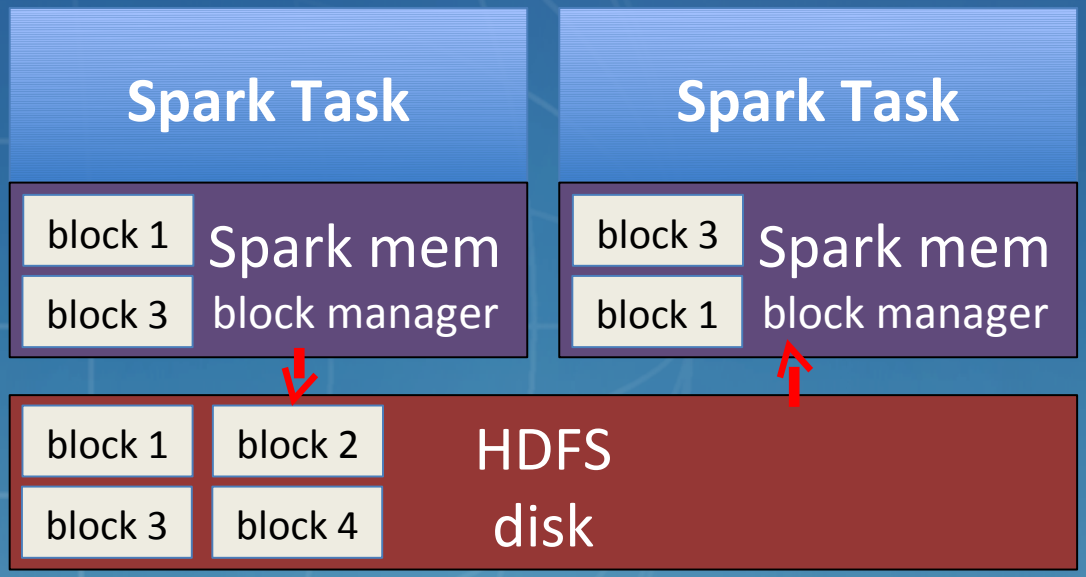
RAM throughput increasing exponentially, Disk slowly

- Spark

Tachyon

Motive

Different jobs share



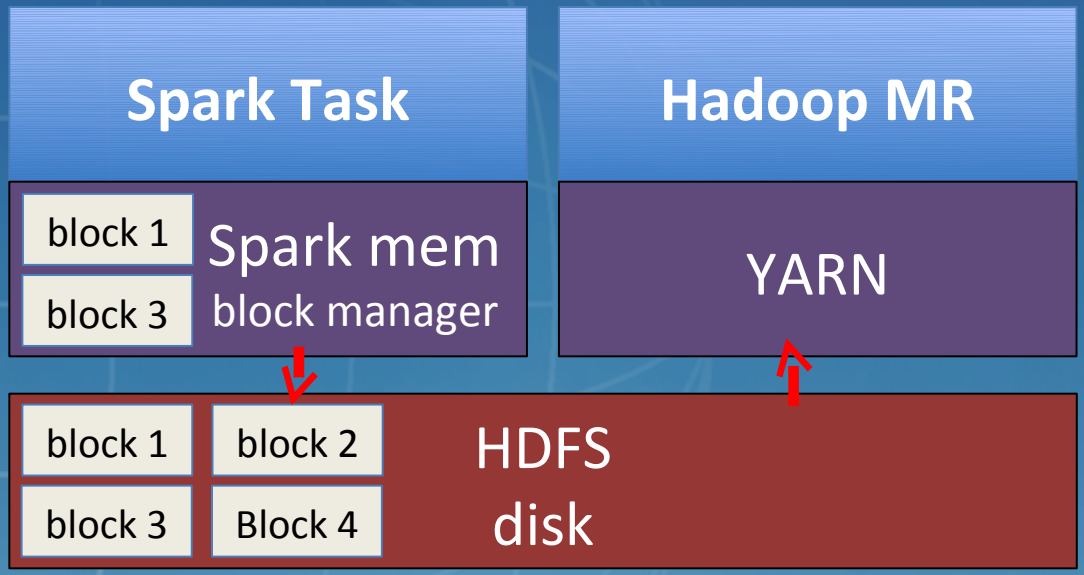
→ Slow writes to disk

- Spark

Tachyon

Motive

Different framework share



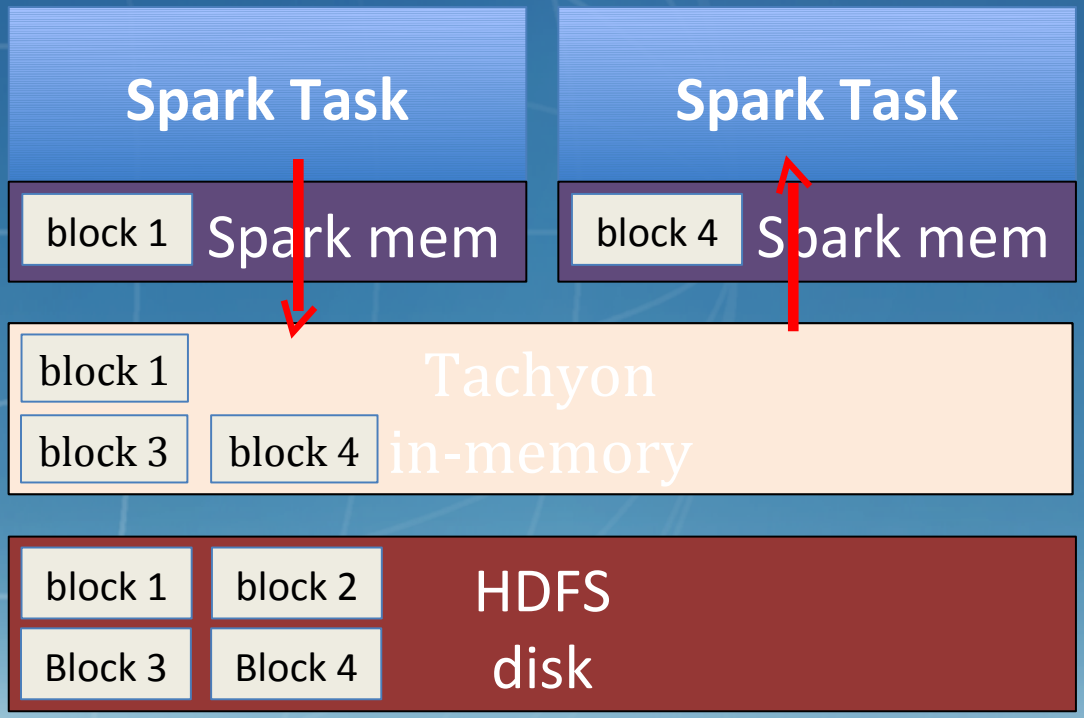
→ Slow writes to disk

- Spark

Tachyon

Solution

execution engine & storage engine same JVM process (no GC & duplication)



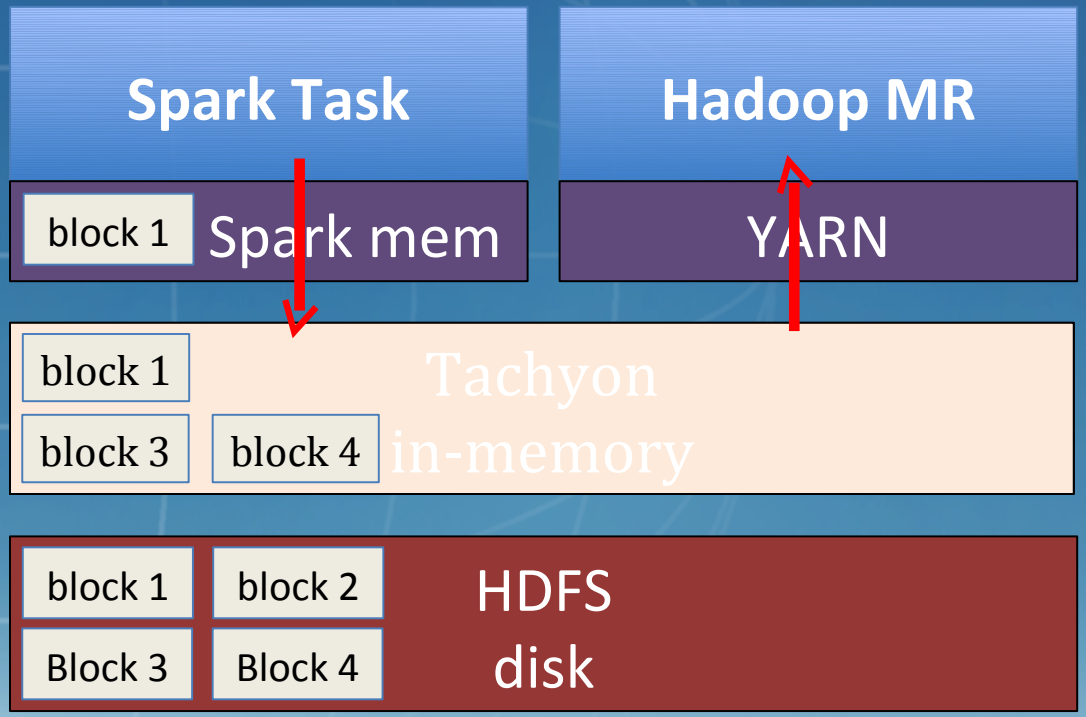
Different Job Task avoiding GC overhead

- Spark

Tachyon

Solution

execution engine & storage engine same JVM process (no GC & duplication)

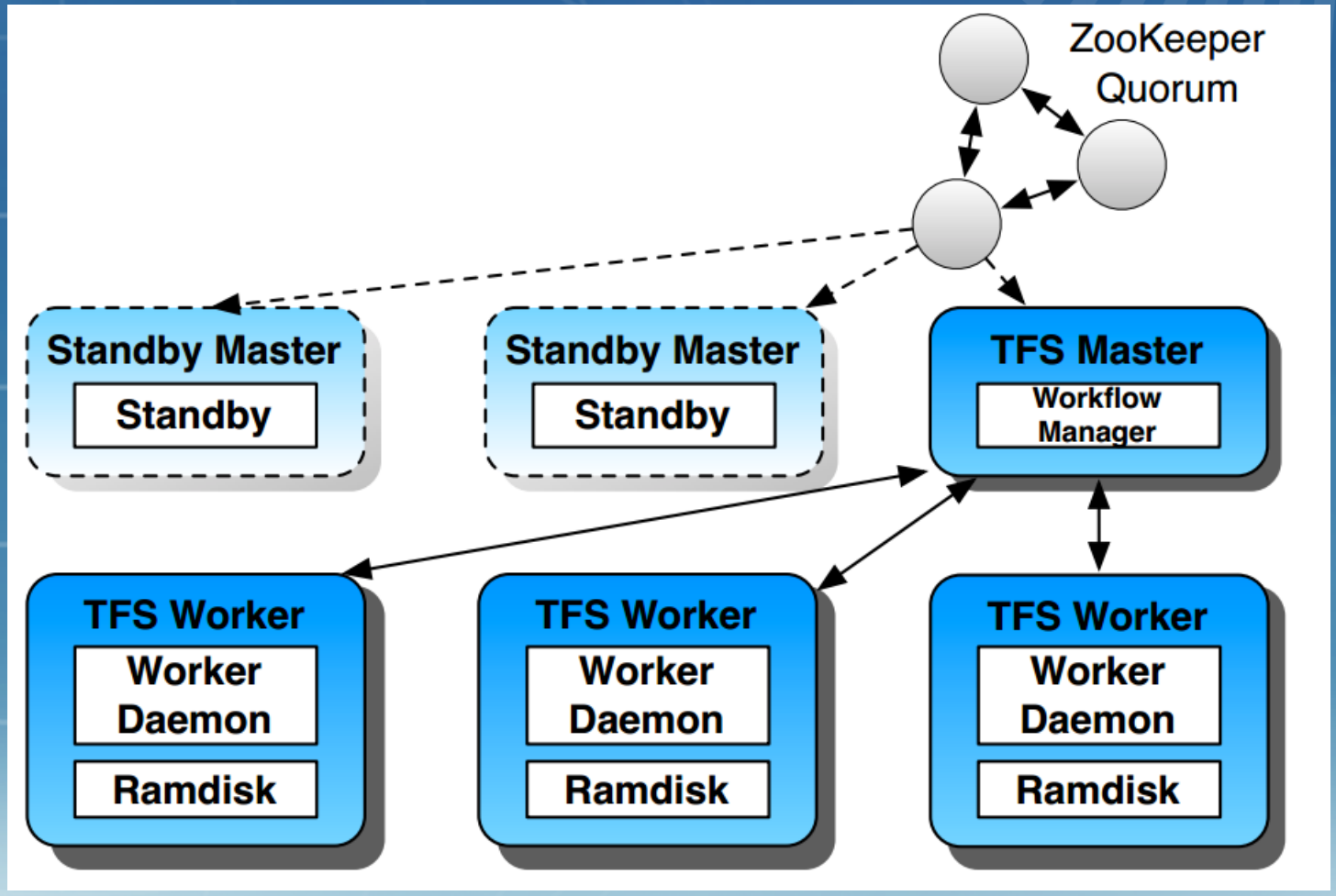


Different Freamewok avoiding GC overhead

- Spark

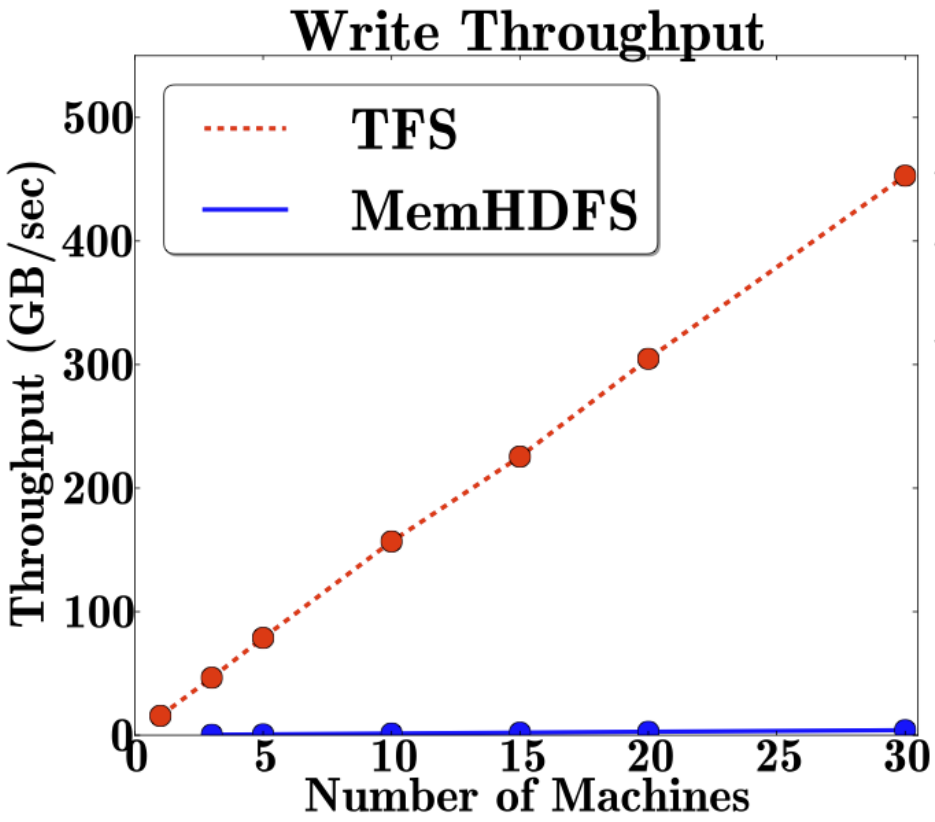
Tachyon

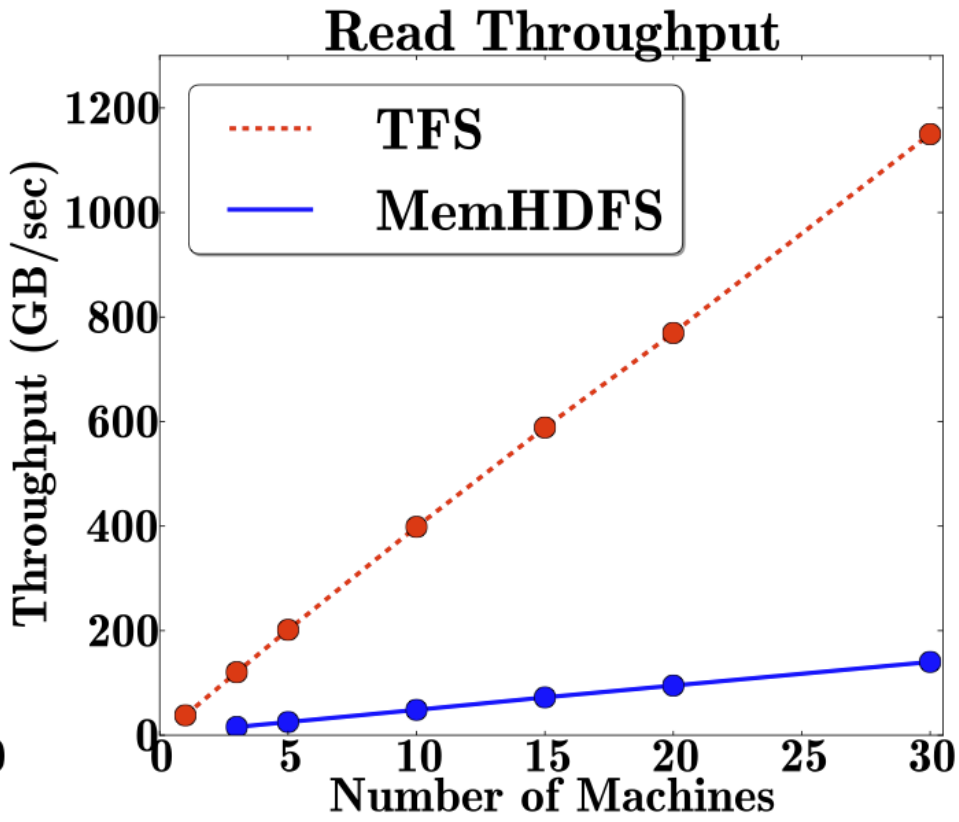
Architecture



Tachyon

Comparison with in Memory HDFS





- Spark

Tachyon

Open Source Status

- V0.4.0 (Feb 2014)
- 20 Developers (7 from Berkeley, 13 from outside)
  - 11 Companies
- Writes go synchronously to under filesystem
(No lineage information in Developer Preview release)
- MapReduce and Spark can run without any code change
(ser/de becomes the new bottleneck)

- Spark

Tachyon

Using : hdfs → tachyon

- Spark

```
val file = sc.textFile("tachyon://ip:port/path")
```

- Shark(Spark SQL)

```
CREATE TABLE orders_tachyon AS SELECT * FROM
orders;
```

- Hadoop MapReduce

```
hadoop jar examples.jar wordcount
tachyon://localhost/input
tachyon://localhost/output
```

- Spark

Tachyon

Tachyon
is in Fedora 20

Thanks to Redhat!

- Spark

BlinkDB

- A data analysis (warehouse) system
- is compatible with Apache Hive and Facebook's Presto (storage, serdes, UDFs, types, metadata)

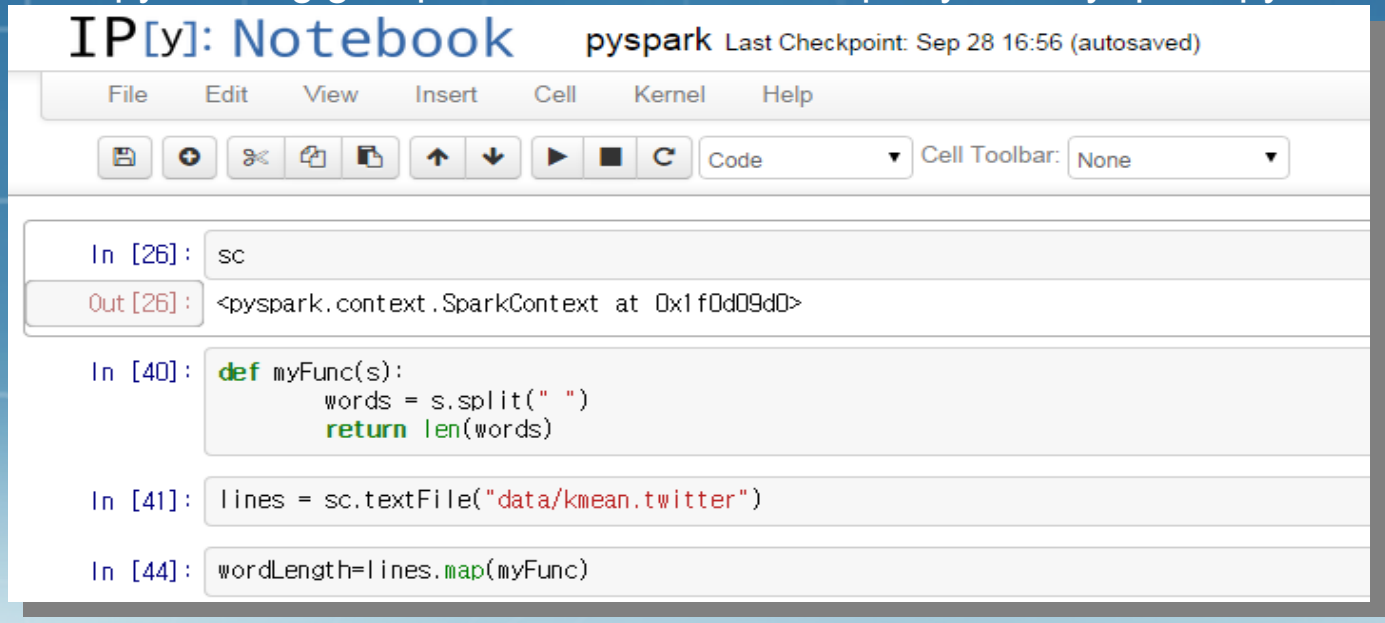
Support **interactive** SQL-like aggregate queries over **massive sets of data**

- Spark

PySpark

- The Spark Python API
- All of PySpark's library dependencies, including **Py4J**, are bundled with PySpark and automatically imported.
- Can run with Ipython notebook

PySpark and Ipython nodebook setting manaual link:
http://nbviewer.ipython.org/gist/fperez/6384491/00-Setup-IPython-PySpark.ipynb



- Spark

SparkR

```
install.packages("rJava")
library(devtools)
install_github("amplab-extras/SparkR-pkg", subdir="pkg")
library(SparkR)
sc <- sparkR.init(master="local")

./sparkR
```

http://amplab-extras.github.io/SparkR-pkg/

http://amplab-extras.github.io/SparkR-pkg/

- Outline

- Bigdata
- Hadoop
- Spark
- **3rd Party Biobigdata**
- Machine Learning
- Streaming
- Future

- 3rd party

Dumbo

- Python module that makes writing and running Hadoop Streaming programs very easy

http://klbostee.github.io/dumbo/
https://github.com/klbostee/dumbo/wiki

- 3rd party

Happy
  Hadoop + Python = Happy
  A framework for writing map-reduce programs for Hadoop using **Jython**

https://code.google.com/p/happy/

3rd party

Happy

```
import sys, happy, happy.log
happy.log.setLevel("debug")
log = happy.log.getLogger("wordcount")
class WordCount(happy.HappyJob):
    def __init__(self, inputpath, outputpath):
        happy.HappyJob.__init__(self)
        self.inputpaths = inputpath
        self.outputpath = outputpath
        self.inputformat = "text"
    def map(self, records, task):
        for _, value in records:
            for word in value.split():
                task.collect(word, "1")
    def reduce(self, key, values, task):
        count = 0;
        for _ in values: count += 1
        task.collect(key, str(count))
        log.debug(key + ":" + str(count))
        happy.results["words"] = happy.results.setdefault("words", 0) + count
        happy.results["unique"] = happy.results.setdefault("unique", 0) + 1
if __name__ == "__main__":
    if len(sys.argv) < 3:
        print "Usage: <inputpath> <outputpath>"
        sys.exit(-1)
    wc = WordCount(sys.argv[1], sys.argv[2])
    results = wc.run()
    print str(sum(results["words"])) + " total words"
    print str(sum(results["unique"])) + " unique words"
```

- 3rd party

Octopy : Google?

```
### triangular.py
# server
source = dict(zip(range(100), range(100)))

def final(key, value):
    print key, value

# client
def mapfn(key, value):
    for i in range(value + 1):
        yield key, i

def reducefn(key, value):
    return sum(value)
```

https://code.google.com/p/octopy/

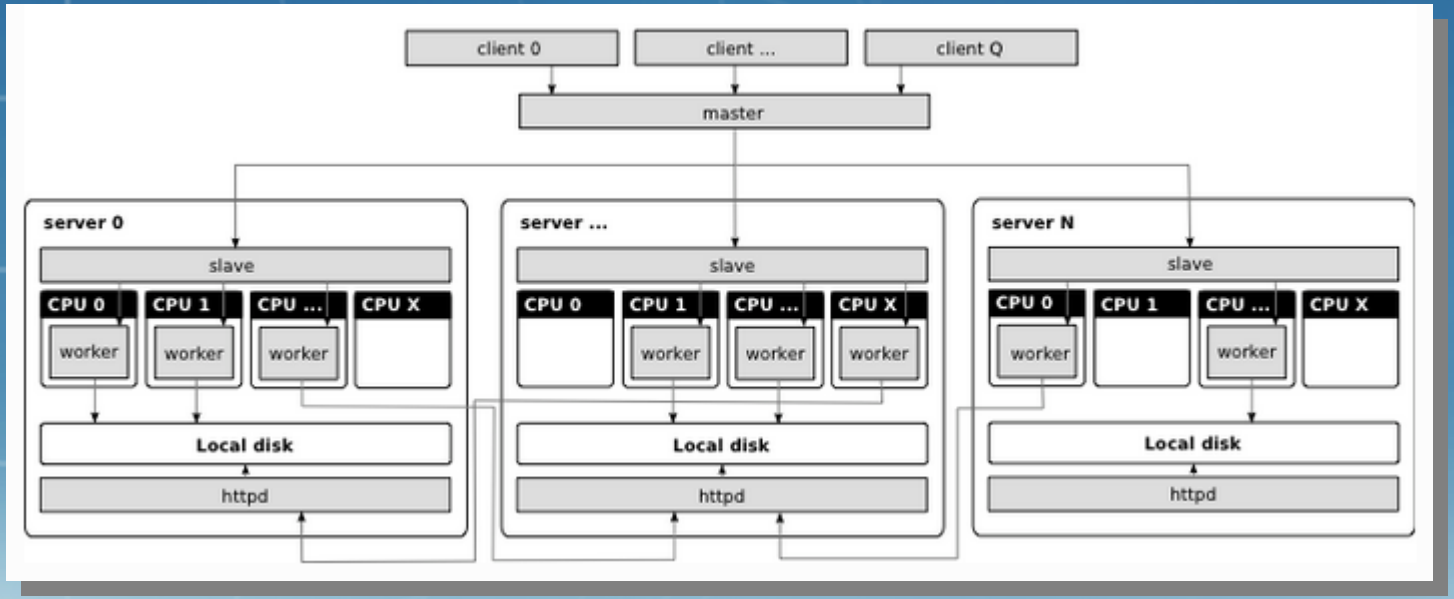
- 3rd party

Disco

open-source framework for distributed computing based on the MapReduce paradigm

Disco Distributed Filesystem

http://discoproject.org/



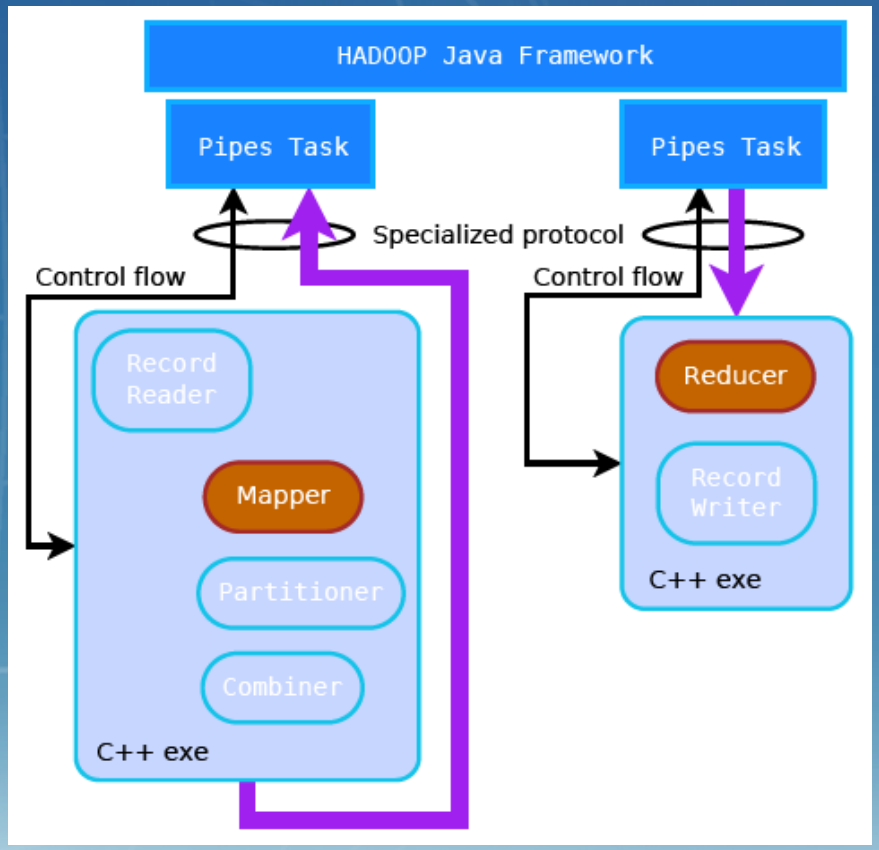
Pydoop

- Python API for Hadoop
- Access to most MR components, including RecordReader,RecordWriter and Partitioner
- Get configuration, set counters and report status via context objects

http://pydoop.sourceforge.net/docs/index.html

Pydoop

Using C/C++ APIs for both MR and HDFS are supported by Hadoop Pipes



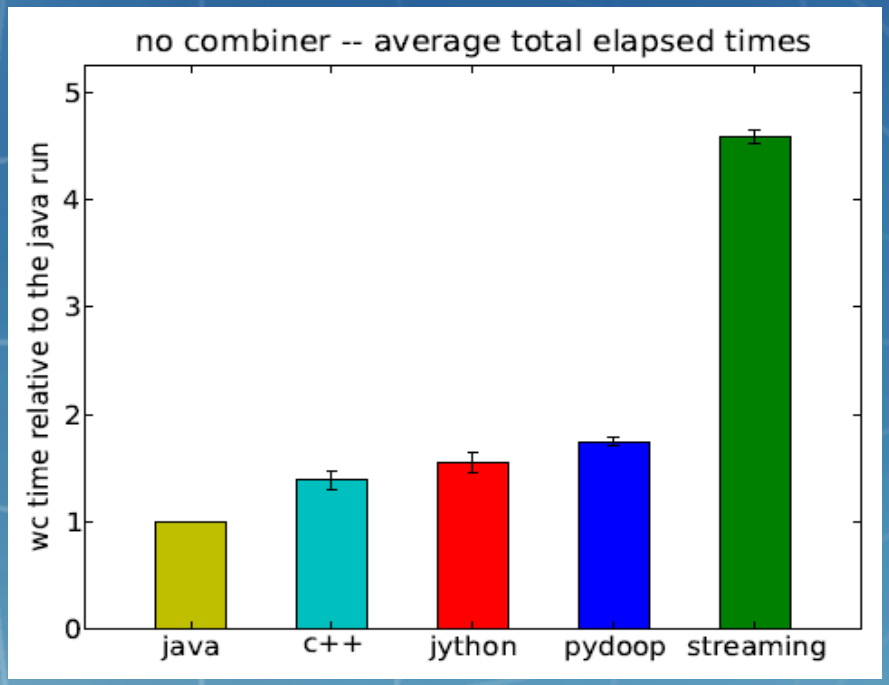
Pydoop

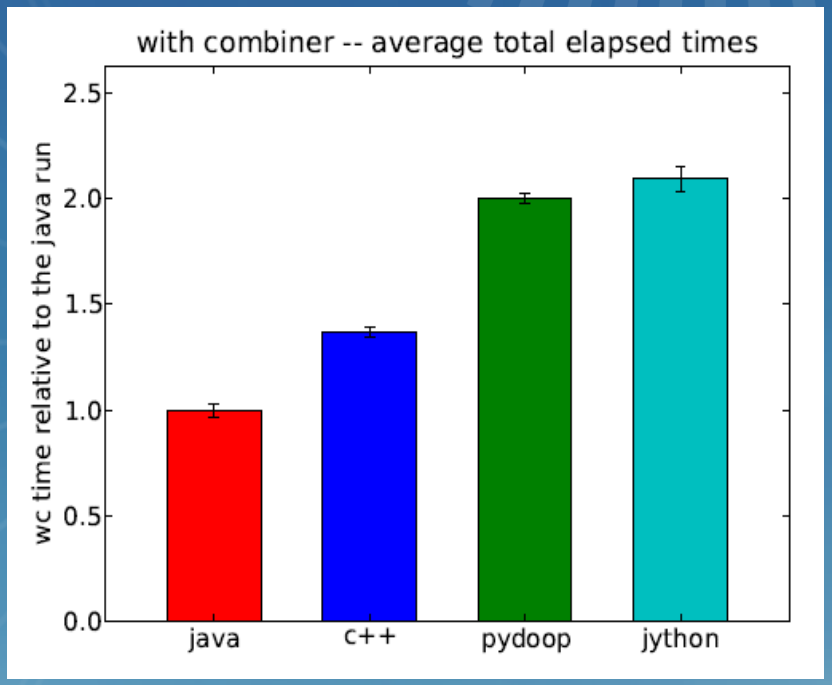
```
from pydoop.pipes import Mapper, Reducer, Factory, runTask
class WordCountMapper(Mapper):
  def map(self, context):
      words = context.getInputValue().split()
      for w in words:
          context.emit(w, "1")

class WordCountReducer(Reducer):
  def reduce(self, context):
  s = 0
  while context.nextValue():
      s += int(context.getInputValue())
  context.emit(context.getInputKey(), str(s))
  runTask(Factory(WordCountMapper, WordCountReducer))
```

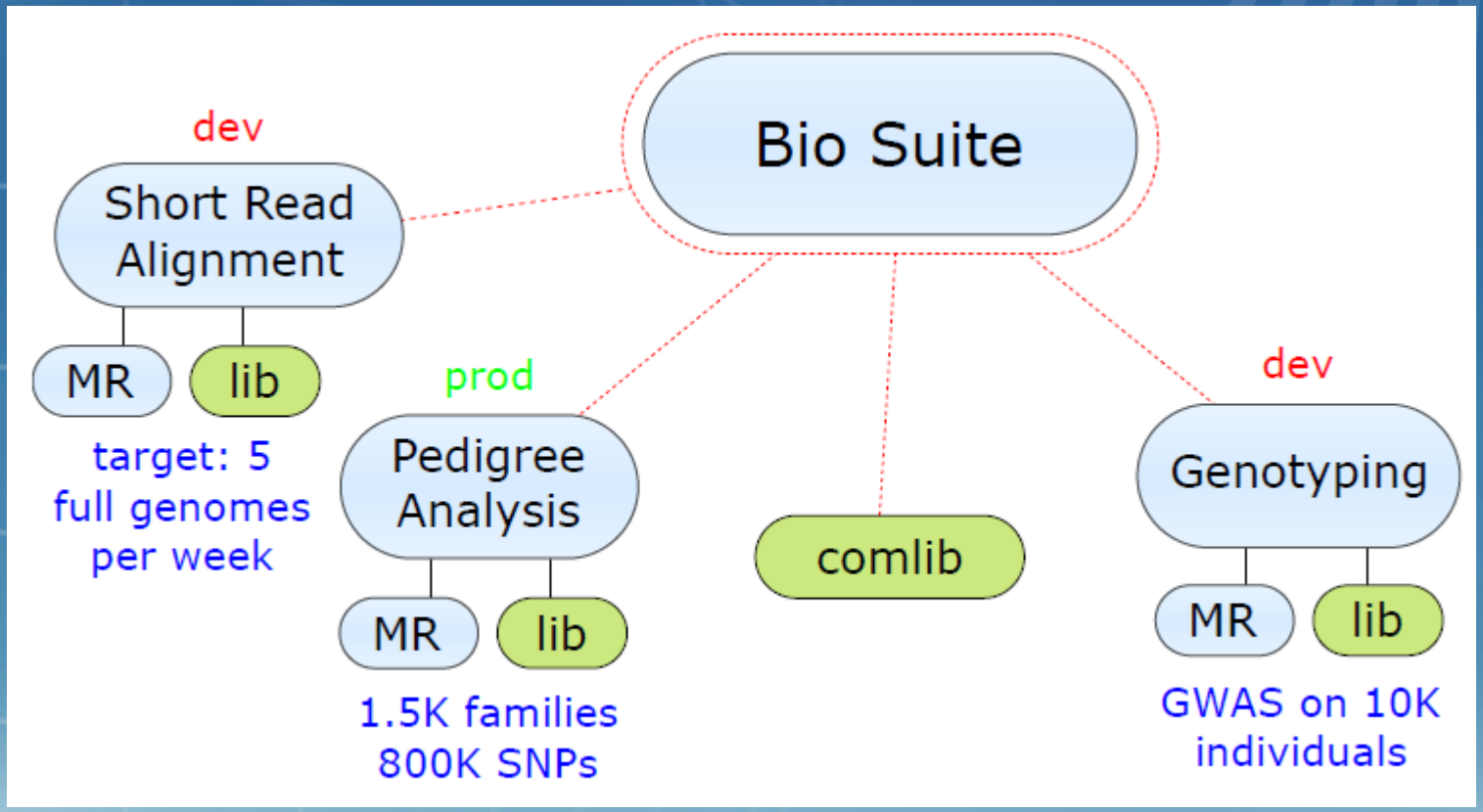
Pydoop

- Performance:





Pydoop Status



Hadoop-BAM

- Java library for the **manipulation** of files using the Hadoop MapReduce frameworkwith the Picard SAM JDK
- BAM, SAM, FASTQ, FASTA, QSEQ, BCF, and VCF
- ADAM, SeqPig using
- Develop processing pipeline that enables efficient, scalable use of cluster/cloud
- Provide data format that has efficient parallel/distributed access across platforms

http://sourceforge.net/projects/hadoop-bam/

ADAM

A genomics processing engine and specialized file format built using Apache Avro, Apache Spark and **Parquet**.

In-Memory RDD	▸Transform records using ***Apache Spark*** ▸Query with SQL using *Shark* ▸Graph processing with *GraphX* ▸Machine learning using *MLBase*
Record/Split	▸Schema-driven records w/ ***Apache Avro*** ▸Store and retrieve records using ***Parquet*** ▸Read BAM Files using ***Hadoop-BAM***
File/Block	▸***Hadoop*** Distributed Filesystem ▸Local Filesystem
Physical	▸Commodity Hardware ▸Cloud Systems - Amazon, GCE, Azure

ADAM

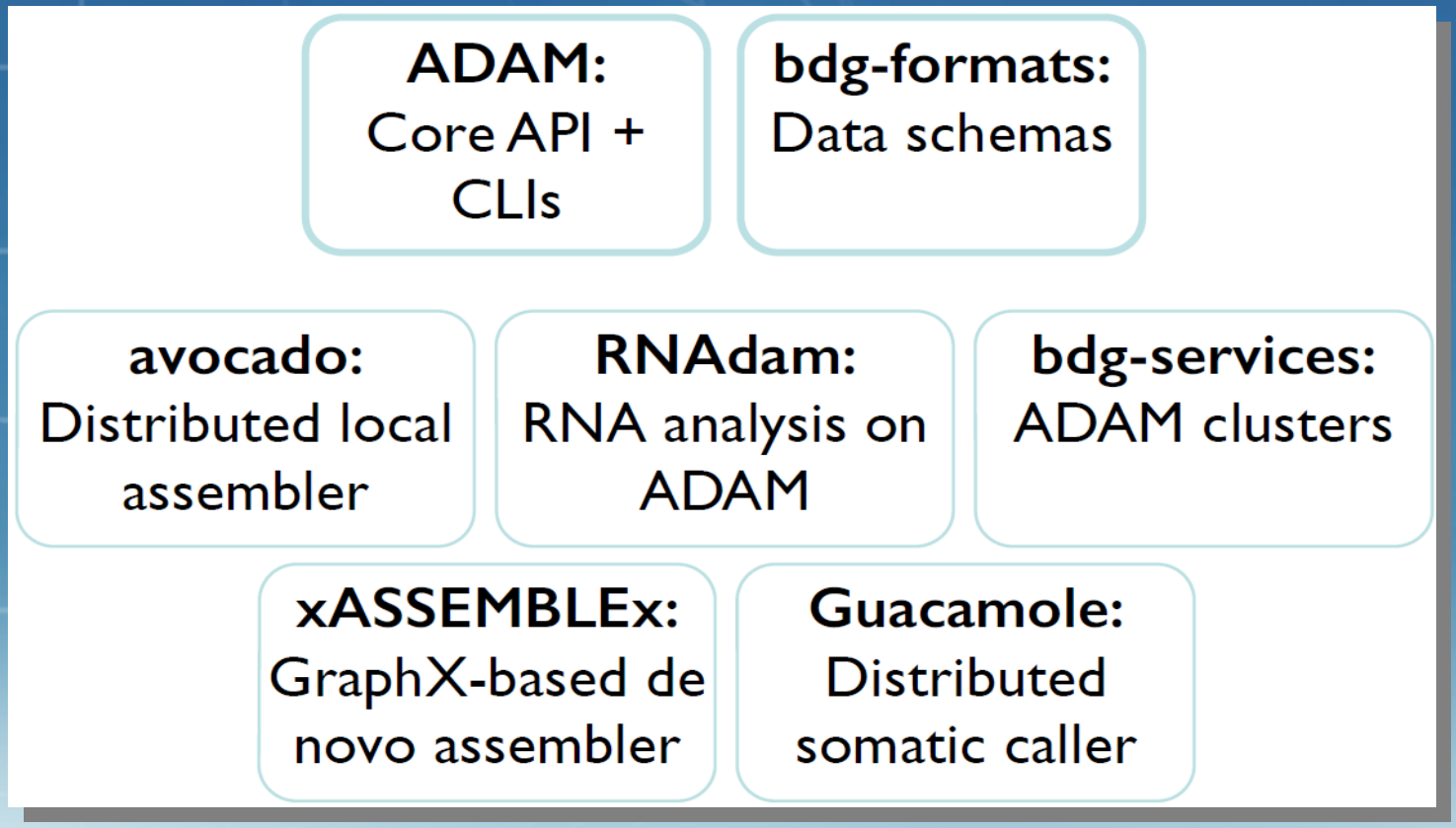
Parquet

- Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language.
- OSS Created by Twitter and Cloudera, based on Google Dremel, just entered Apache Incubator
- Columnar File Format:
  - Limits I/O to only data that is needed
  - Compresses very well - ADAM files are 5-25% smaller than BAM files without loss of data
  - Fast scans - load only columns you need, e.g. scan a read flag on a whole genome, highcoverage file in less than a minute

ADAM

Avocado

- A distributed pipeline for calling variants, and is built on top of Apache Spark and the ADAM API



SeqPig

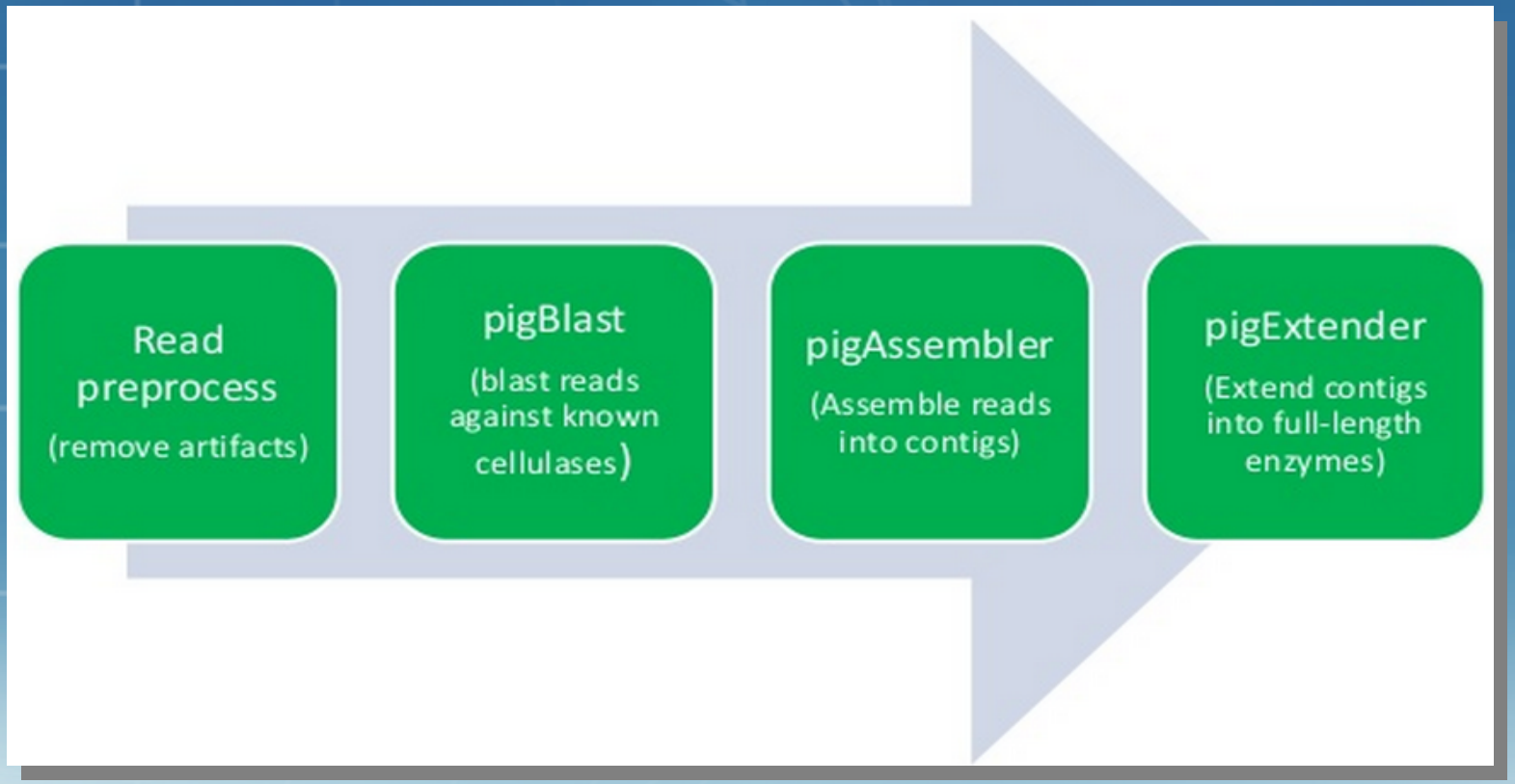
A library for Apache Pig for the distributed analysis user-defined-functions (UDF's)
supports BAM/SAM, FastQ and Qseq input and output

http://sourceforge.net/projects/seqpig/

- 3rd party

BioPig

- https://github.com/JGI-Bioinformatics/biopig
- http://www.slideshare.net/ZhongWang3/biopig-for-large-sequencing-data



Hadoop Seal

a suite of distributed applications for **aligning** short DNA reads, and **manipulating** and **analyzing** short read alignments.

Bcl2Qseq
Extract reads in qseq format from an Illumina run directory.

Demux
Separate sample data in the qseq file format produced by a multiplexed Illumina run.

PairReadsQSeq
convert files in the Illumina qseq file format to **prq format** to be processed by the alignment program, Seqal.

Seqal
Distributed short read mapping and duplicate removal tool.

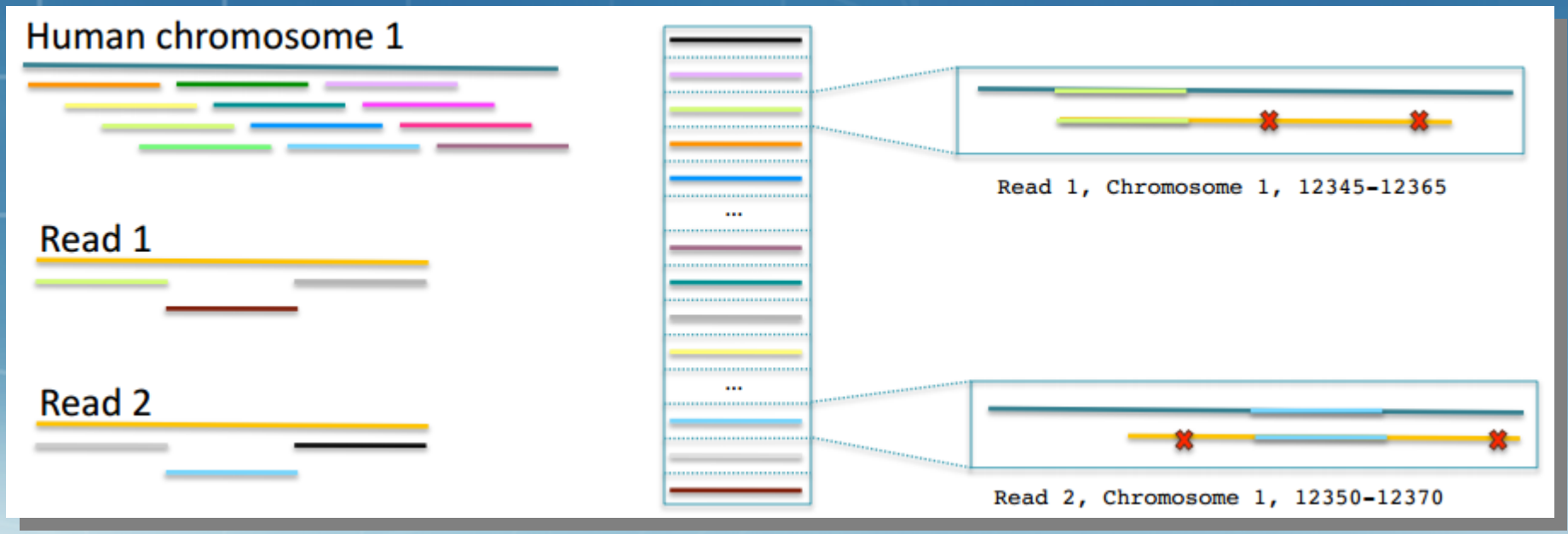
ReadSort
Distributed **sorting** of read mappings.

RecabTable
distributed **calculation** of covariates table to **estimate** empirical base qualities.

CloudBurst

- Highly Sensitive Short Read Mapping with MapReduce
- SNP discovery, genotyping, and personal genomics

July 8, 2010 - CloudBurst 1.1.0 released with support for Hadoop 0.20.X



Crossbow

- a scalable, portable, and automatic Cloud Computing tool for finding SNPs from short read data
- Genotyping from short reads using cloud computing
- combines Bowtie and SoapSNP
- Crossbow scripts require Perl 5.6 or later.
- Johns Hopkins Univ.

- Outline

- Bigdata
- Hadoop
- Spark
- 3rd party
- **Machine Learning**
- Streaming
- Future

Apache Mahout

- Scalable machine learning library.
- Goodbye MapReduce !
- Mahout's Spark Shell Binding
- Algorithms
  - User and Item based recommenders
  - Matrix factorization based recommenders
  - K-Means, Fuzzy K-Means clustering
  - Latent Dirichlet Allocation
  - Singular Value Decomposition
  - Logistic regression classifier
  - (Complementary) Naive Bayes classifier
  - Random forest classifier
  - High performance java collections

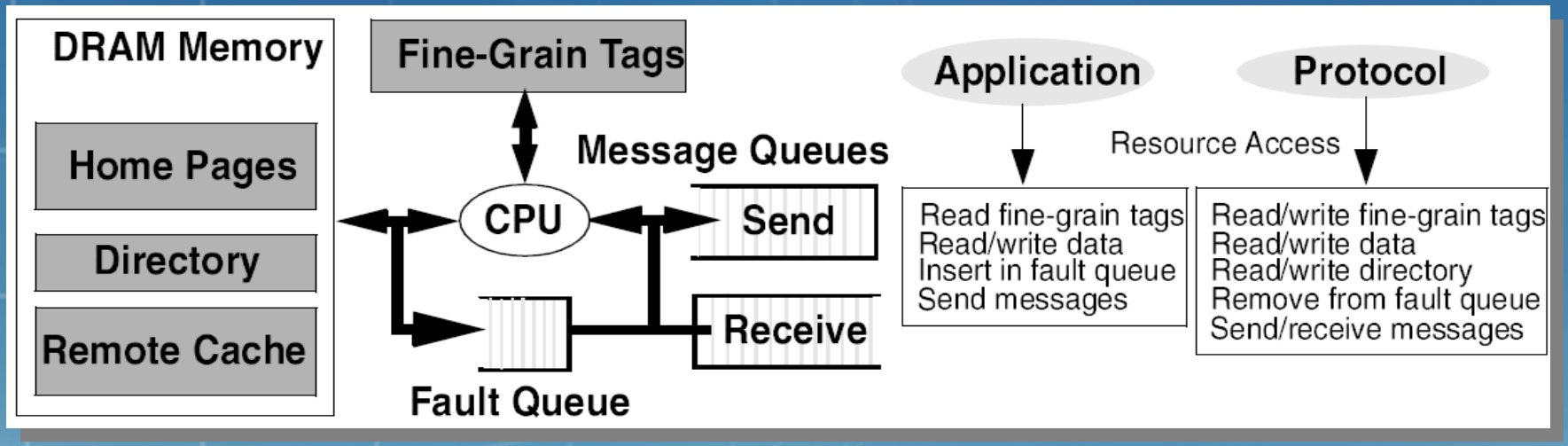
H2O

- The world's fastest in-memory platform for machine learning and predictive analytics on big data
- Fine-Grain Distributed Processing
- Support Deep Learning Neural Network using R
- Support for Java, Scala, and Python,R.
  The solution's interface is driven by JSON APIs

H2O

Fine-Grain Distributed Processing

combines the responsiveness of in-memory processing with the ability to run fast **serialization between nodes and clusters**



Vowpal Wabbit

- a fast out-of-core learning system.

- Two ways to have a fast learning algorithm
  start with a slow algorithm and speed it up
  build an intrinsically fast learning algorithm

- by Microsoft Research and (previously) Yahoo! Research.

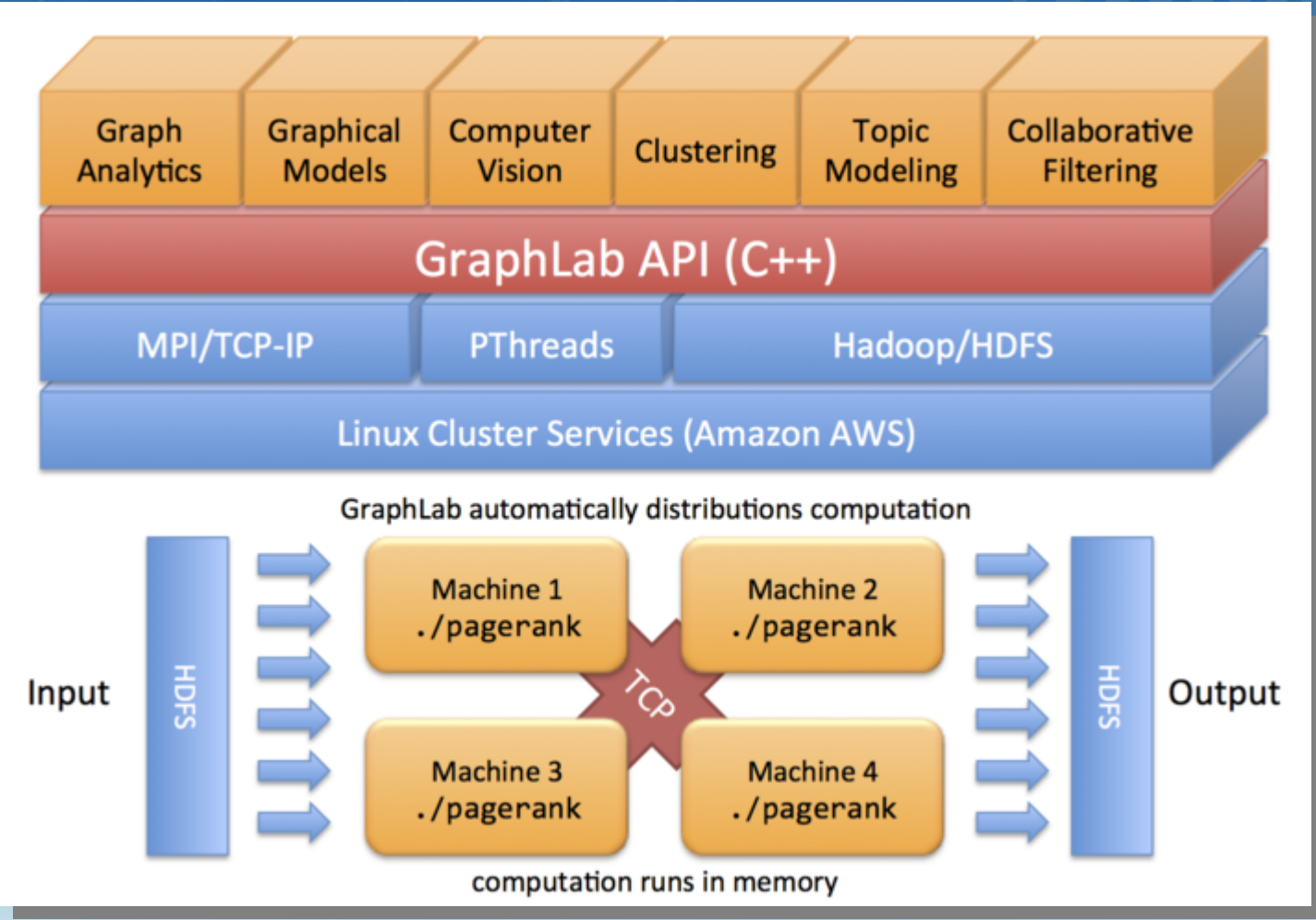
https://github.com/JohnLangford/vowpal_wabbit/wiki

GraphLab

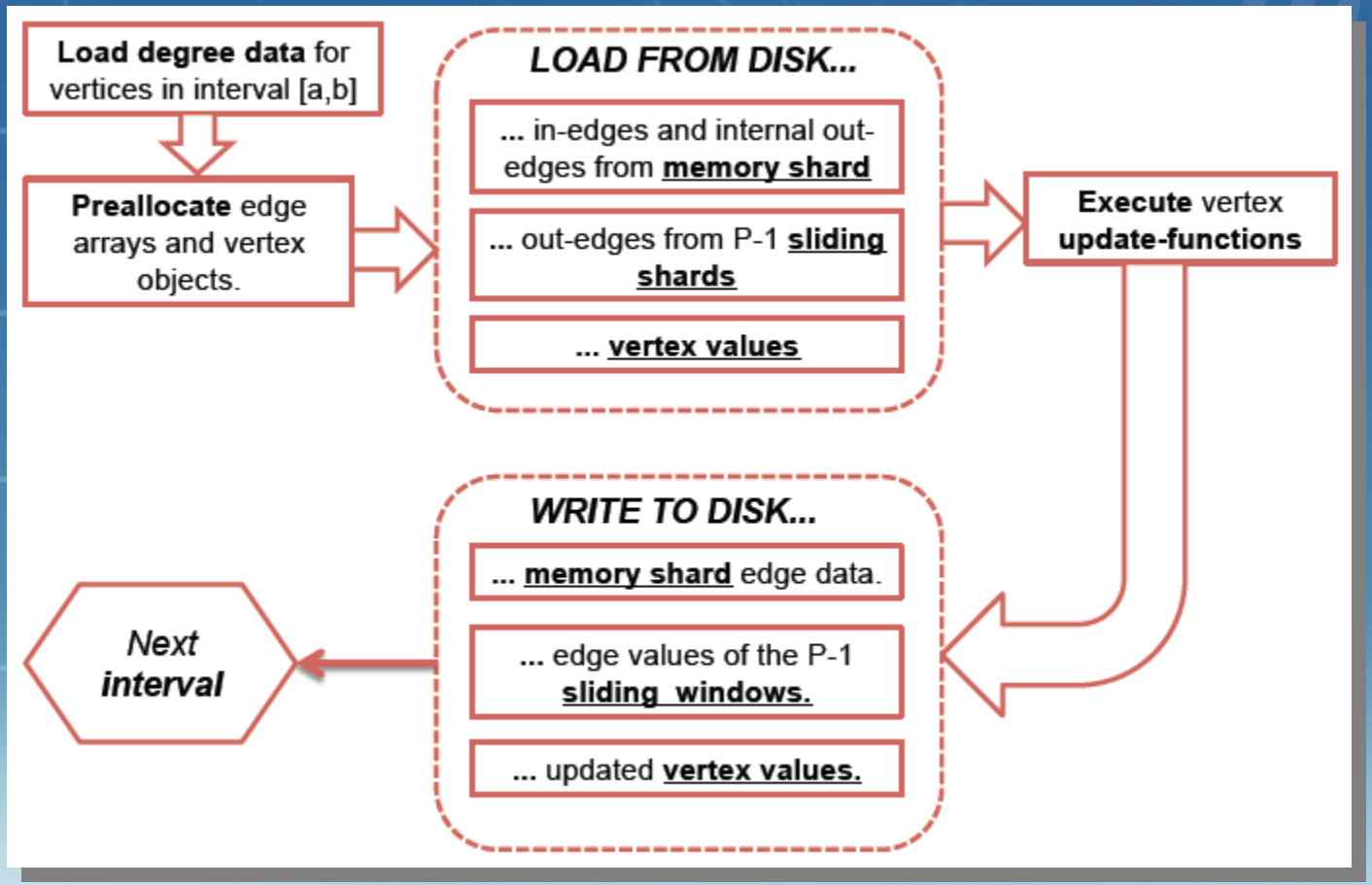
- Carnegie Mellon University in 2009 to develop a new parallel computation abstraction tailored to machine learning
- Toolkits
  - Topic Modeling
  - graph_algorithms
  - Graph Analytics
  - Clustering
  - Collaborative Filtering
  - Graphical Models
  - Factor Graph Toolkit
  - Linear iterative solver
  - Computer Vision

GraphLab

- Automatically distributions computation on HDFS



GraphLab
GraphChi
Disk-based large-scale graph computation



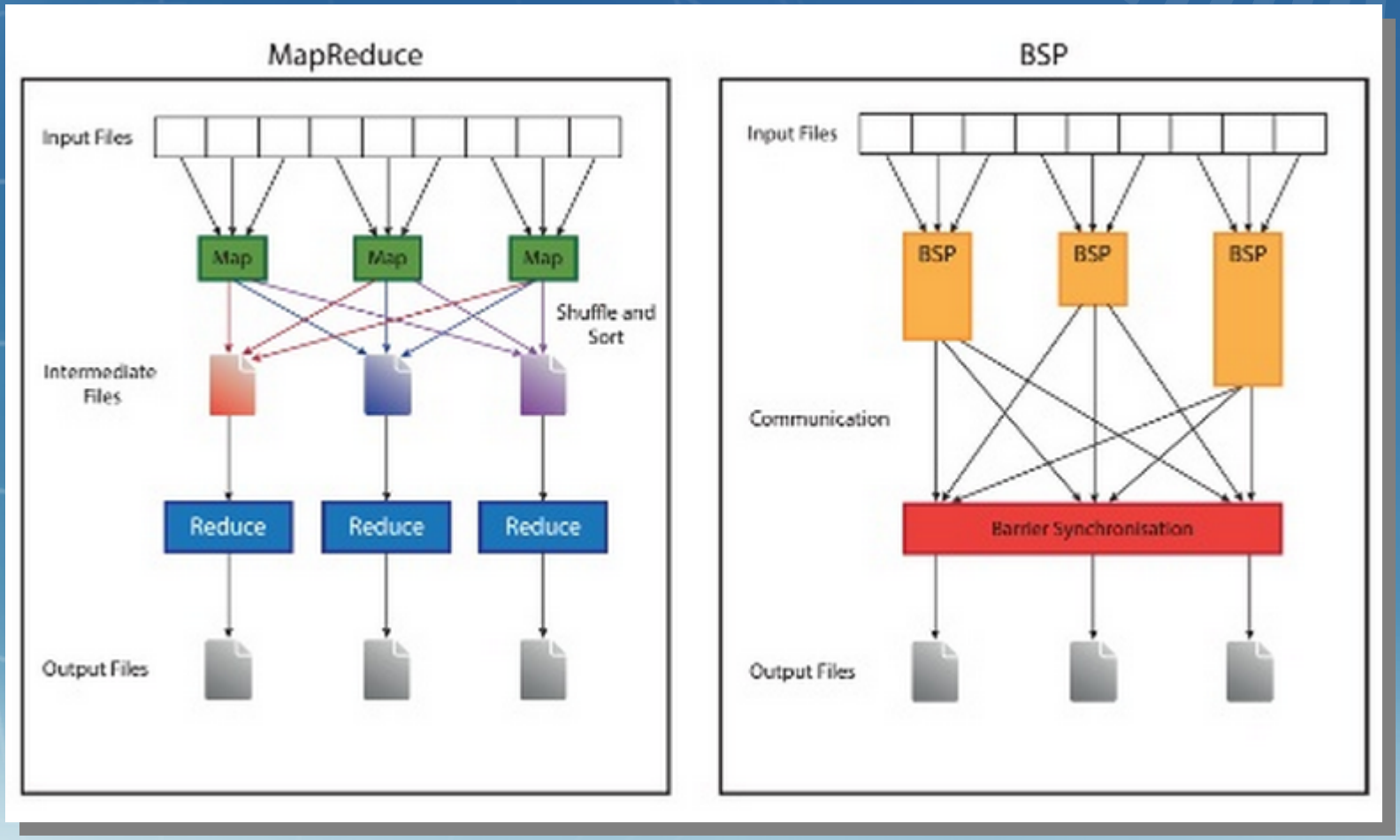
Apache Giraph

An iterative graph processing system built for high scalability

Giraph originated as the open-source counterpart to Pregel

are inspired by the Bulk Synchronous Parallel model of distributed computation introduced by Leslie Valiant.

Apache Hama
A general BSP framework on top of Hadoop



Apache Hama
http://www.slideshare.net/udanax/apache-hama-at-samsung-open-source-conference

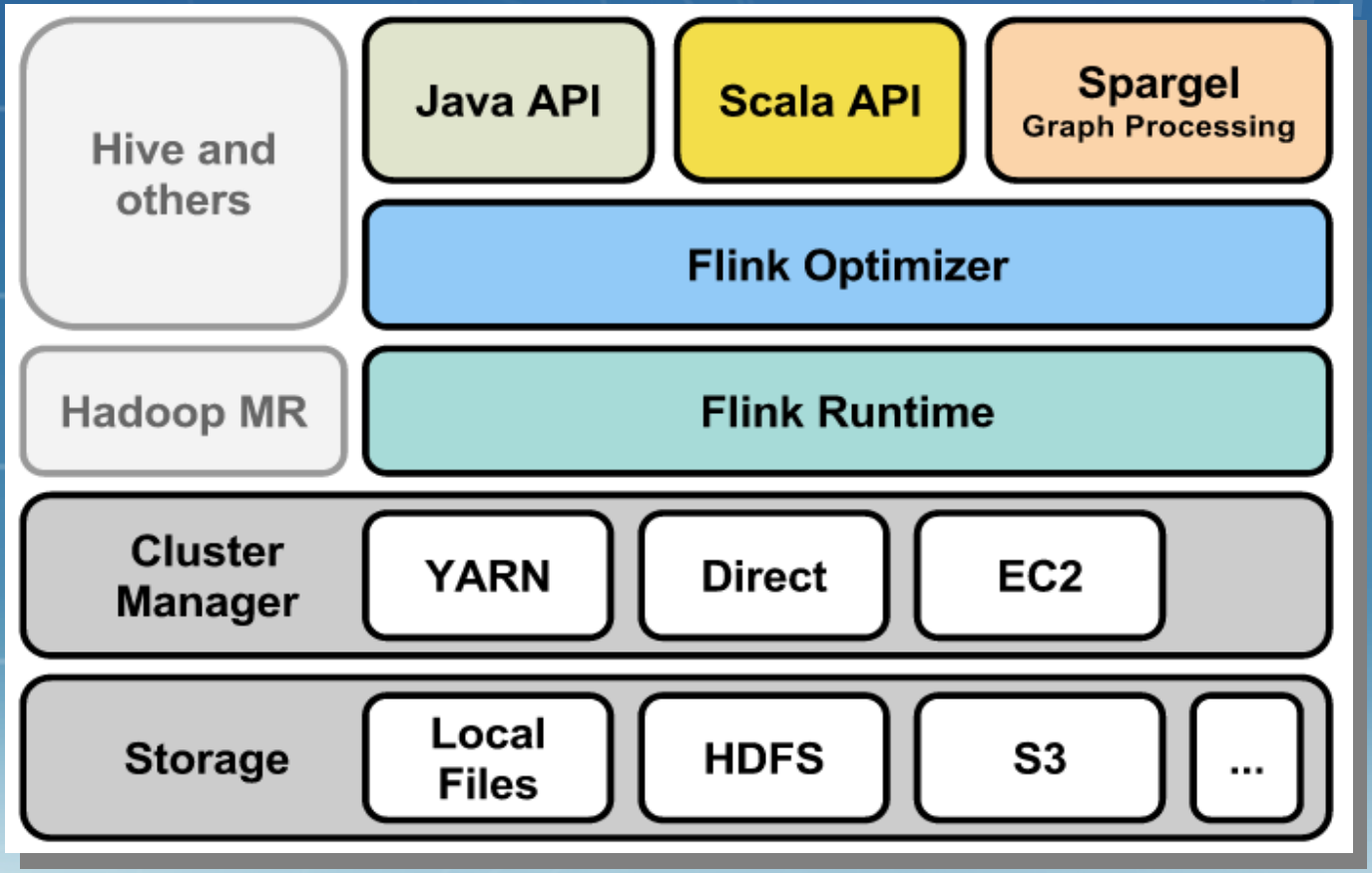
Apache Hama's **Advanced Analytics** Examples:

- Sparse Matrix-Vector Multiplication
- Semi-Clustering
- **K-means Clustering**
- Neural Networks
- **Gradient Descent**
- **PageRank**
- Single Source Shortest Path
- Bipartite Matching

Apache Flink(Stratosphere)

A platform for efficient, distributed, general-purpose data processingProgramming APIs for different languages (Java, Scala) and paradigms (record-oriented, graph-oriented).

Incubating..



Tajo

A big data warehouse system on Hadoop

Compatible

- ANSI/ISO SQL standard compliance
- Hive MetaStore access support
- JDBC driver support
- Various file formats support, such as CSV, RCFile, RowFile, SequenceFile and Parque

- Outline

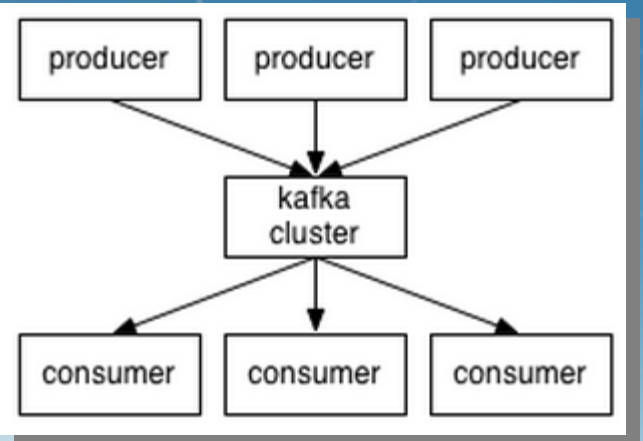
- Bigdata
- Hadoop
- Spark
- 3rd party
- Machine Learning
- **Streaming**
- Future

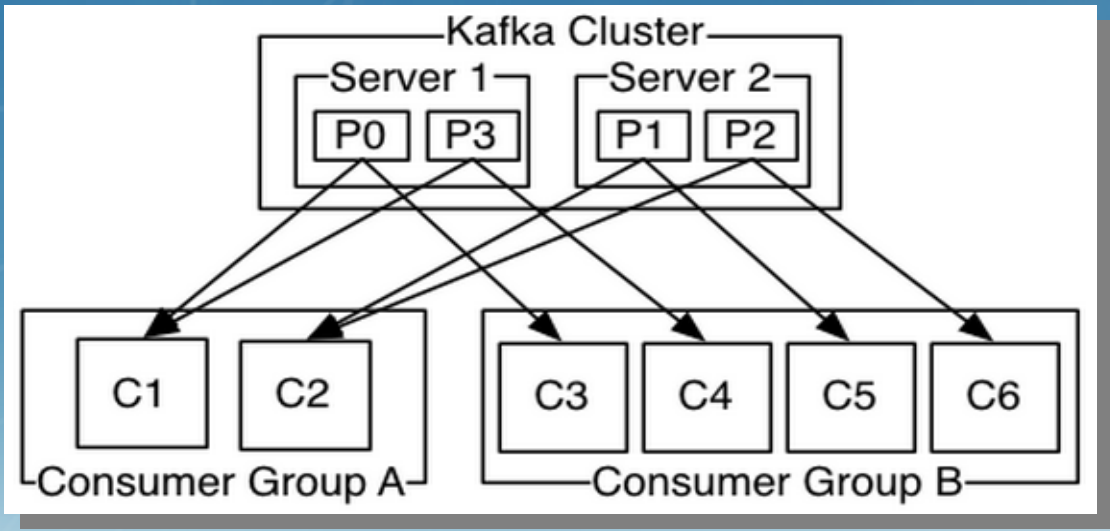
- Streaming

Kafka

- A high-throughput distributed messaging system.

- Producers
  publish data to the topics of their choice.
- Consumers
  Messaging traditionally has two models: queuing and publish-subscribe.





- Streaming

Kafka Ecosystem:

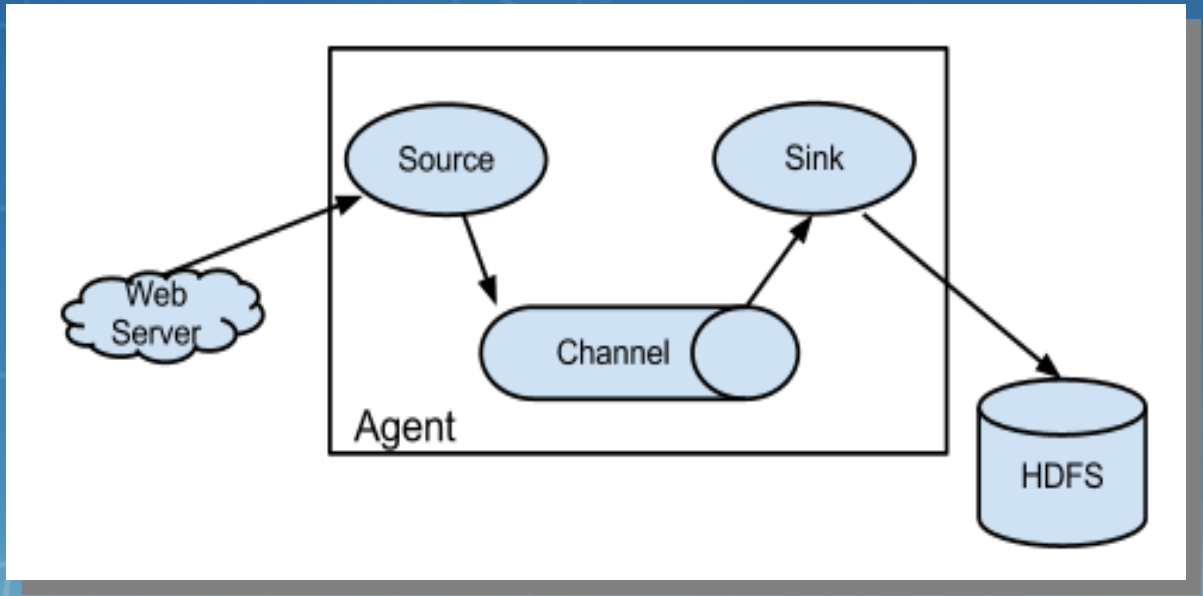
- Storm - A stream-processing framework.
- Samza - A YARN-based stream processing framework.
- Storm Spout - Consume messages from Kafka and emit as Storm tuples
- Kafka-Storm - Kafka 0.8, Storm 0.9, Avro integration
- SparkStreaming - Kafka reciever supports Kafka 0.8 and above
- Camus - LinkedIn's Kafka=>HDFS pipeline. This one is used for all data at LinkedIn, and works great.
- Kafka Hadoop Loader A different take on Hadoop loading functionality from what is included in the main distribution.
- Flume - Contains Kafka Source (consumer) and Sink (producer)

- Streaming

Flume

A distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data.

Avro
Thrift
Syslog
Netcat



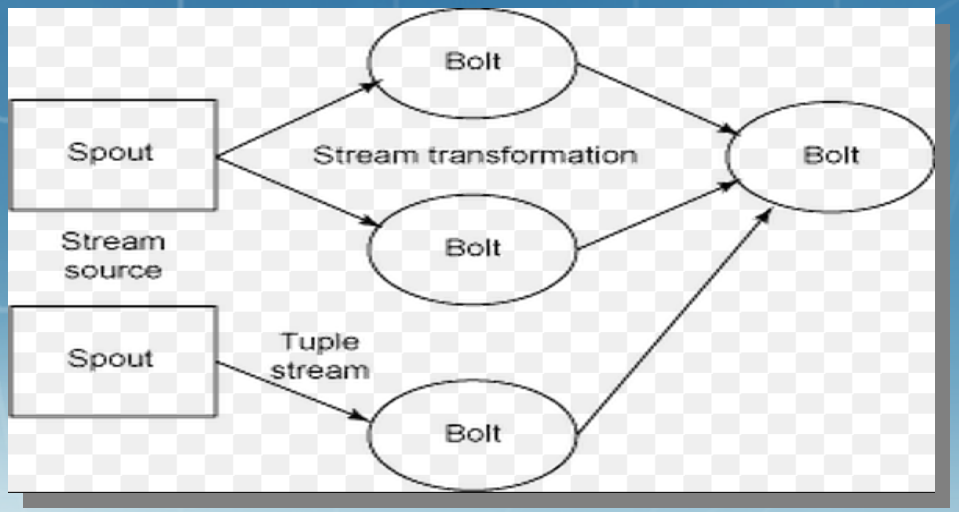
- Streaming

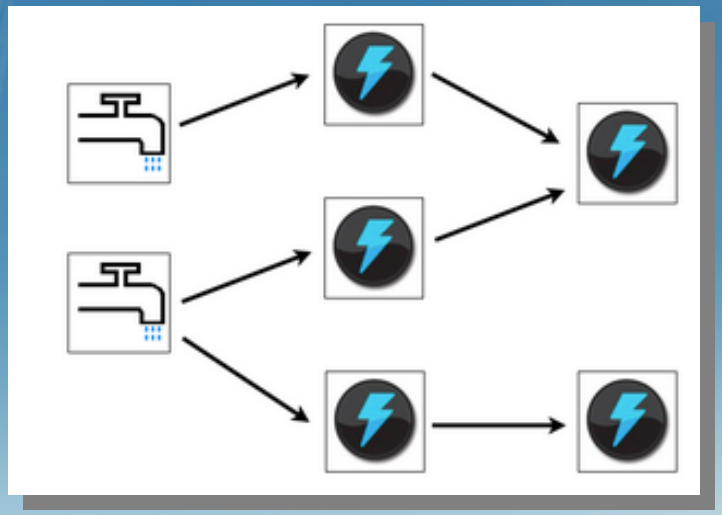
Storm

a free and open source distributed realtime computation system

Topologies

To do realtime computation on Storm, you create what are called “topologies”. A topology is a graph of computation.





- Streaming

MQTT(MQ Telemetry Transport)

A machine-to-machine (M2M)/"Internet of Things" connectivity protocol.

- publish/subscribe messaging transport
- used in sensors communicating to a broker via satellite link, over occasional dial-up connections with healthcare providers

Three qualities of service for message delivery:

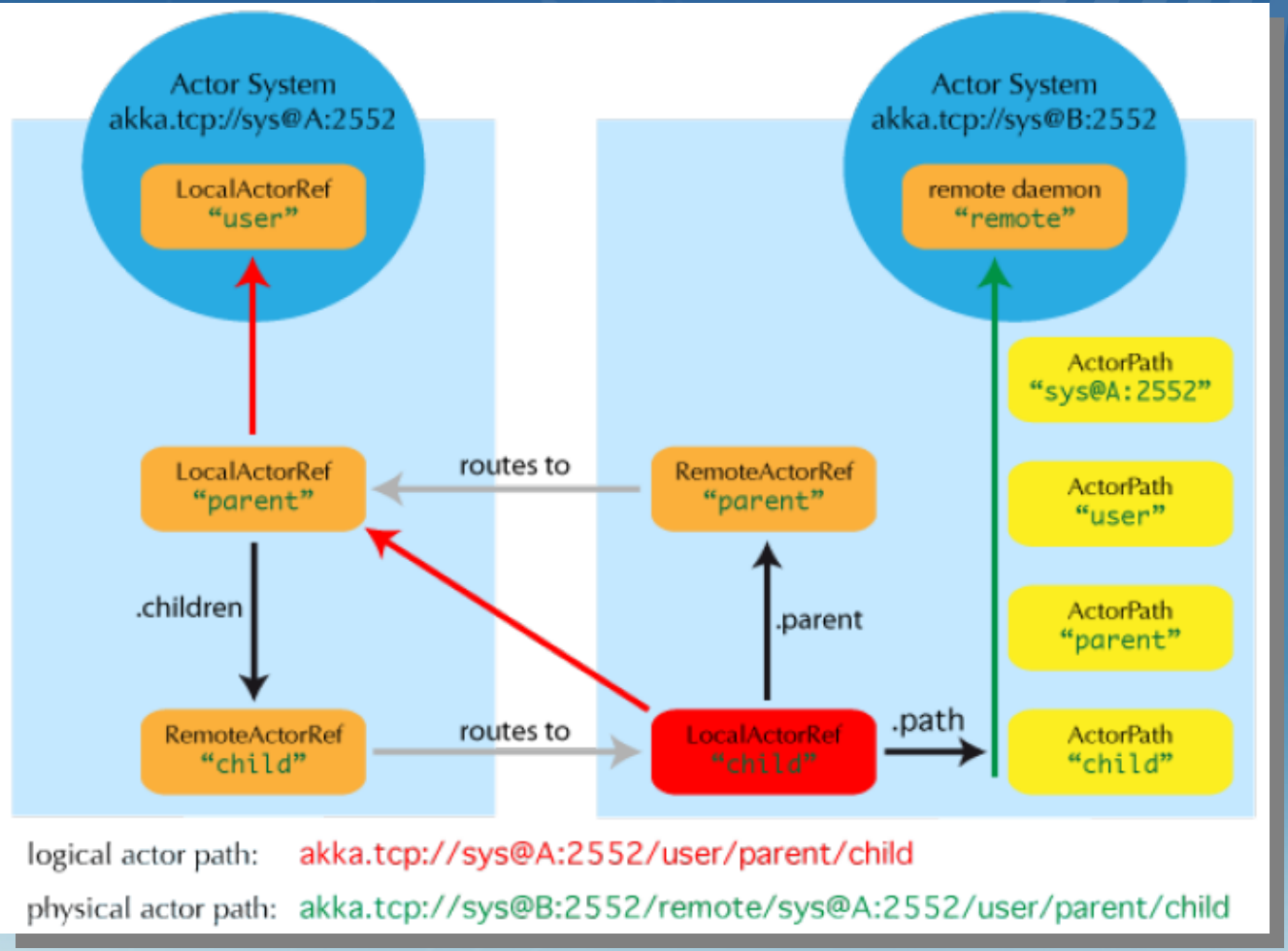
- "At most once", where messages are delivered according to the best efforts of the underlying TCP/IP network. Message loss or duplication can occur. This level could be used, for example, with ambient sensor data where it does not matter if an individual reading is lost as the next one will be published soon after.
- "At least once", where messages are assured to arrive but duplicates may occur.
- "Exactly once", where message are assured to arrive exactly once. This level could be used, for example, with billing systems where duplicate or lost messages could lead to incorrect charges being applied.

- Streaming

Akka

- Scalable real-time transaction processing
- an unified runtime and programming model
  - Scale up (Concurrency)
  - Scale out (Remoting)
  - Fault tolerance
- Actor
  - Simple and high-level abstractions for concurrency and parallelism.
  - Asynchronous, non-blocking and highly performant event-driven programming model.
  - Very lightweight event-driven processes (several million actors per GB of heap memory).

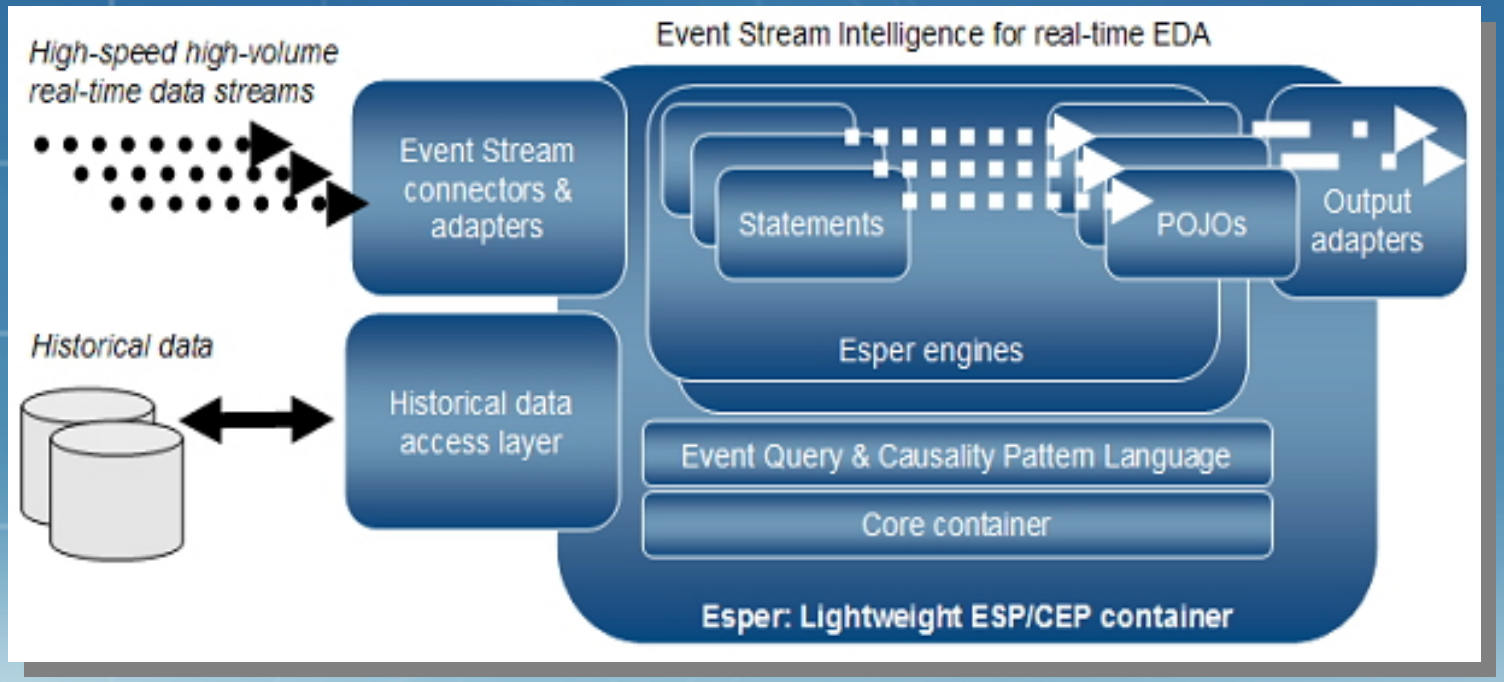
Akka
Remote Deployment



Streaming

Esper

- a component for complex event processing (CEP) and event series analysis
- Event Processing Language (EPL)
  highly scalable, memory-efficient, in-memory computing, SQL-standard, minimal latency, real-time streaming-capable
- offers a Domain Specific Language (DSL) for processing events.



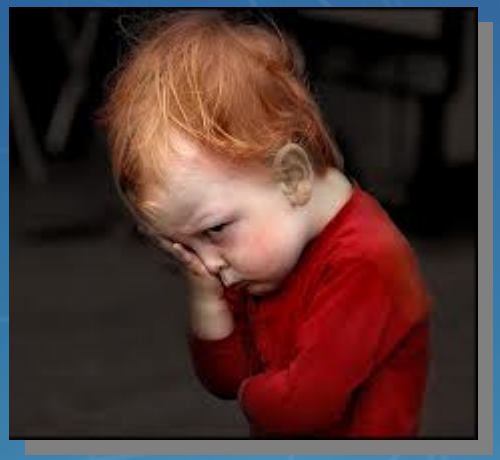
- Outline

- Bigdata
- Hadoop
- Spark
- 3rd party
- Machine Learning
- Streaming
- **Future**

Future

Bigdata and Machine Learning

Very Difficult !



- Future

Solution

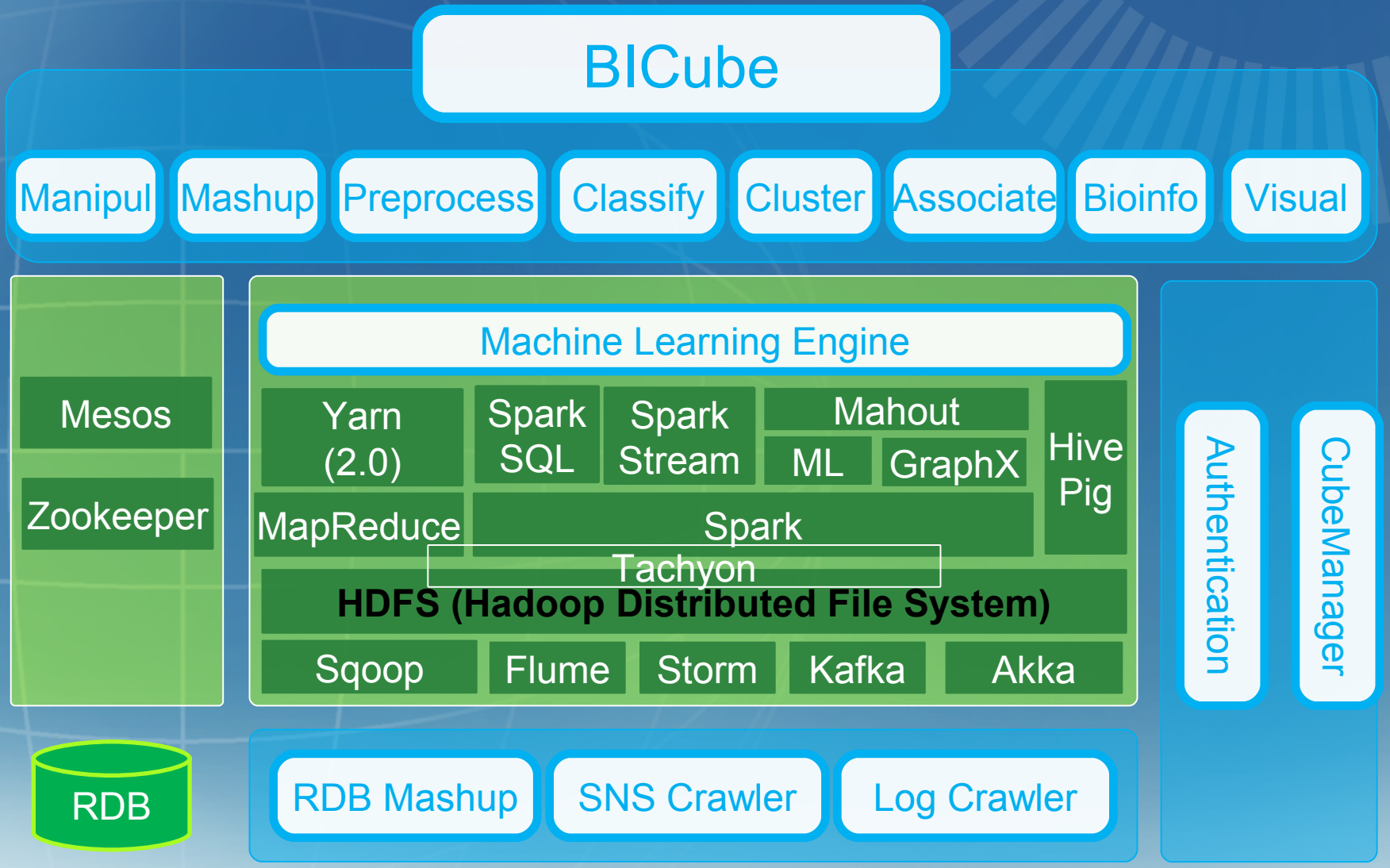
- For End User
- For ML Developer
- Interactive Advanced Machine Learning Development Platform
- DSL style iteractive ML modeling(like matlab)
- Domain specific libraries and tool

- Future

Conclusion

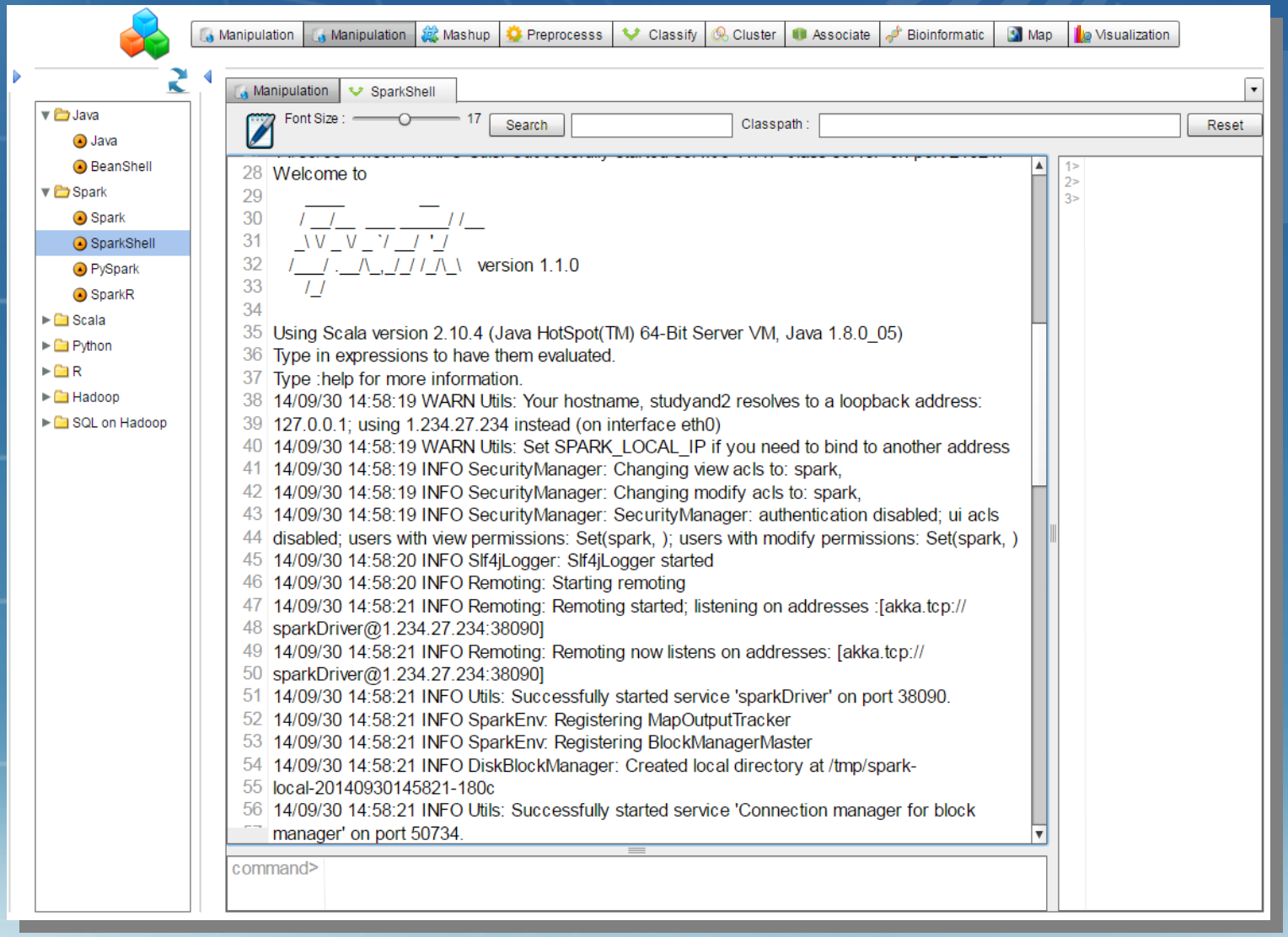
- Bigdata is very important
- ML needs Computation power (Graphical model)
- Real time streaming and computation
- High throughput Streaming
- Data size will improve the ML
- Biobigdata needs scalable ML
- High throughput sequence analysis
- De novo Genome Assembly is base step

Demo


BICube
Manipul
Mashup
Preprocess
Classify
Cluster
Associate
Bioinfo
Visual
Mesos
Zookeeper
Machine Learning Engine
Yarn (2.0)
Spark SQL
Spark Stream
Mahout
ML
GraphX
Hive Pig
MapReduce
Spark
Tachyon
HDFS (Hadoop Distributed File System)
Sqoop
Flume
Storm
Kafka
Akka
Authentication
CubeManager
RDB
RDB Mashup
SNS Crawler
Log Crawler

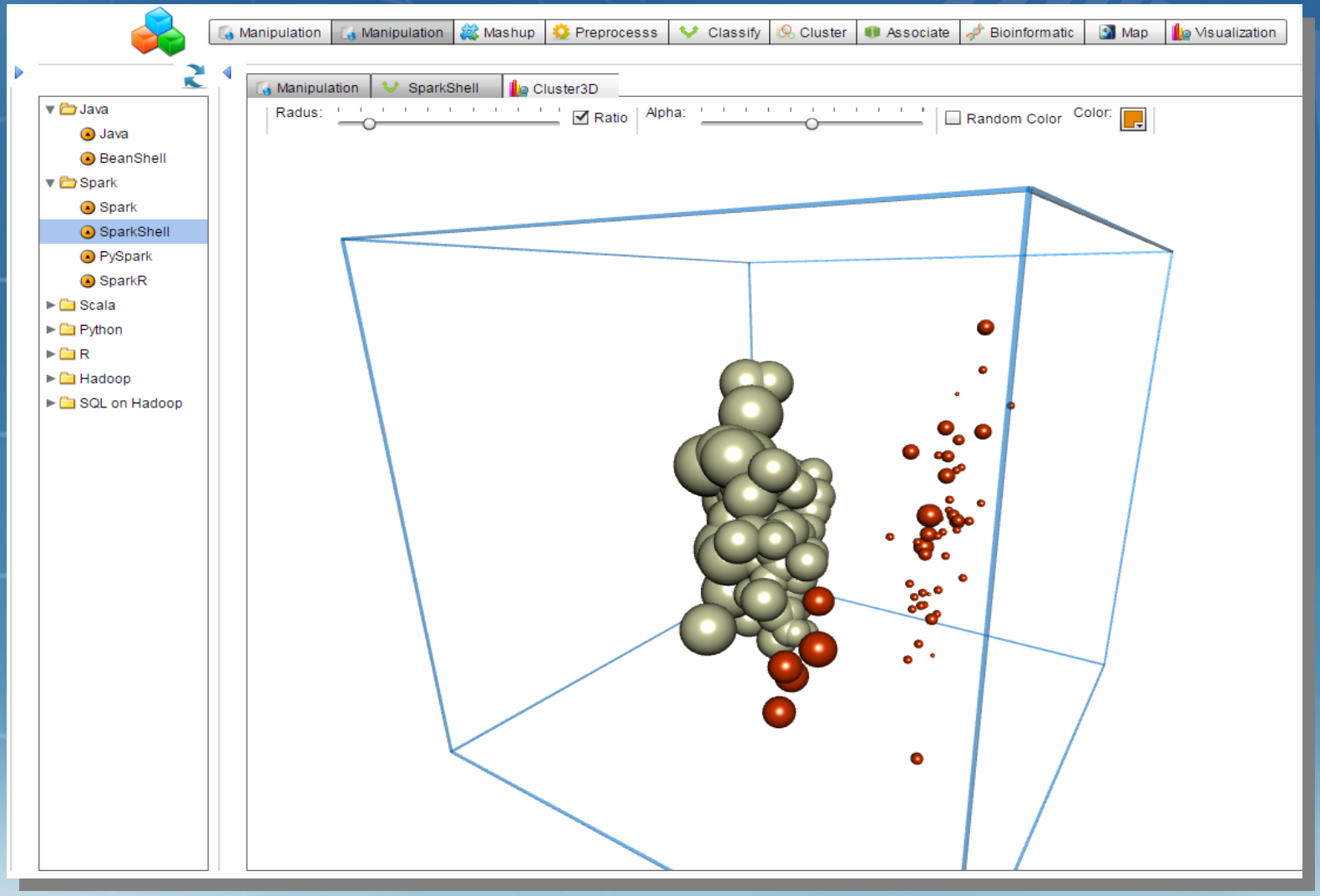
Demo

Spark Interactive Shell



Demo

View GPU Cluster3D using Spark Interactive Shell



Thanks !
Question ?

and Demo...

```
import com.chimpler.example.kmeans._
import com.bicube.akka.visualization.VisualDeploy

val args2 = Array(
     "/user/root/spark/data/twitter.location"
     ,"/user/root/spark/data/kmean.image3.png"
     )

KMeansAppHadoop.run(args2, sc)


import com.bicube.akka.visualization.VisualDeploy
val  v=new VisualDeploy()
v.deploy("{" +
"\"name\":\"ImageViewer\"" +
", \"module\":\"image\"" +
", \"path\":\"spark/data/kmean.image3.png\"" +
"}");
```

- Spark Shell

Demo

```
import com.bicube.akka.visualization._
val  v=new VisualDeploy()


v.deploy("{" +
"\"name\":\"PDBViewer\"" +
", \"module\":\"pdb3d\"" +
", \"path\":\"hdfs:/user/root/spark/data/1CRN.pdb.gz\"" +
"}");

import com.bicube.akka.visualization._
val  v=new VisualDeploy()
v.deploy("{" +
"\"name\":\"Cluster3D\"" +
", \"module\":\"cluster3d\"" +
", \"path\":\"hdfs:/user/root/cluster/kmeans-iris/output-iris\"" +
"}");
```