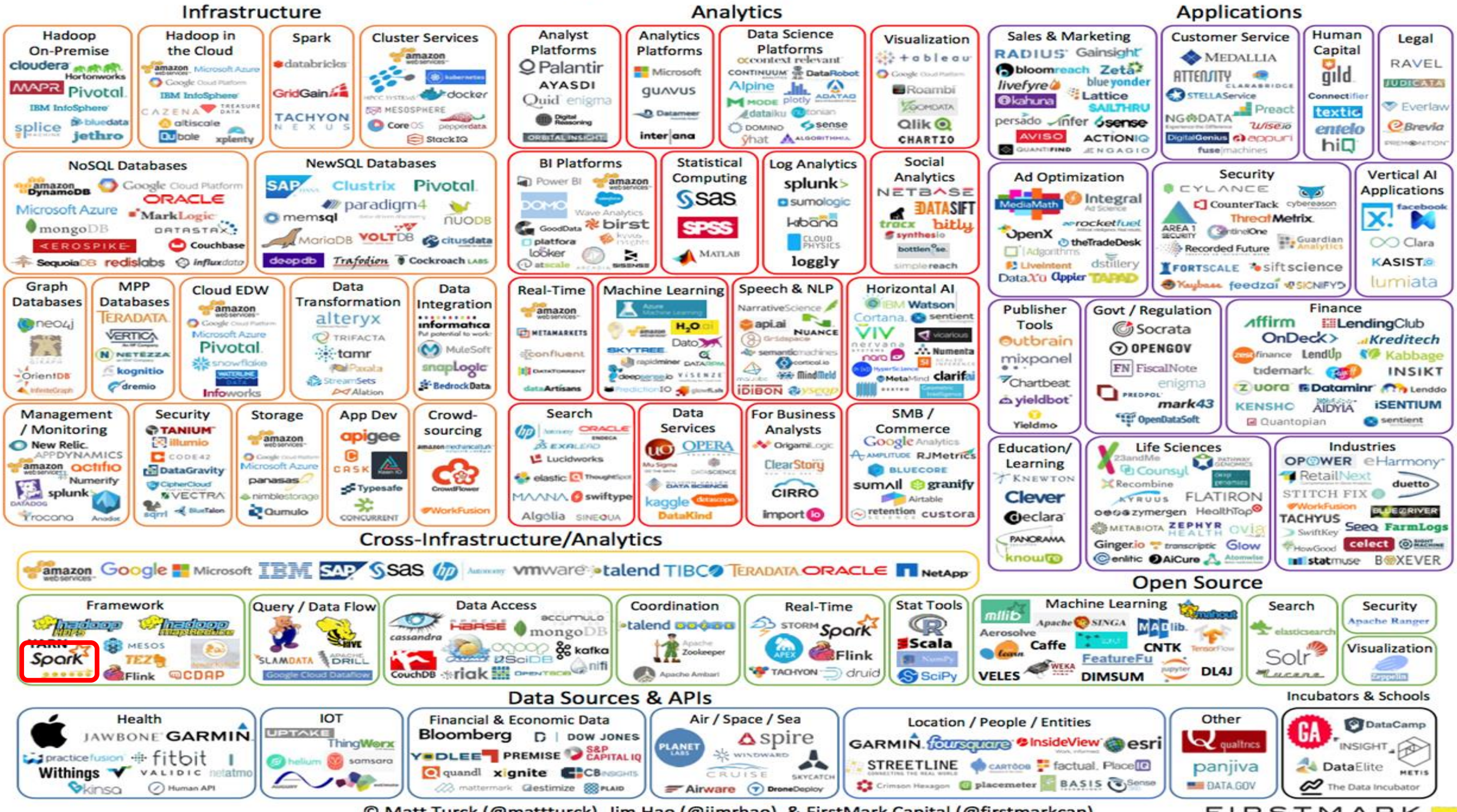
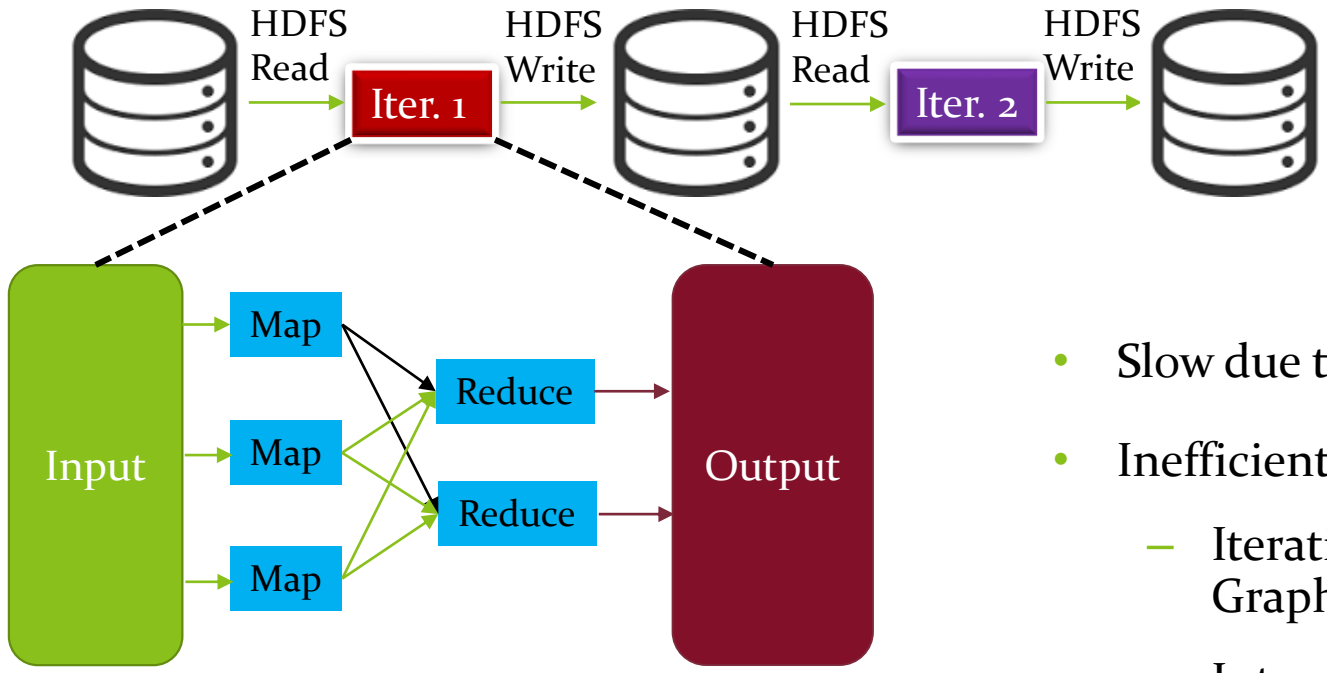
Department of Computer Science and Engineering
UNIVERSITY OF COLORADO
DENVER | ANSCHUTZ MEDICAL CAMPUS

Lightning-Fast Cluster Computing

Big Data Landscape 2016

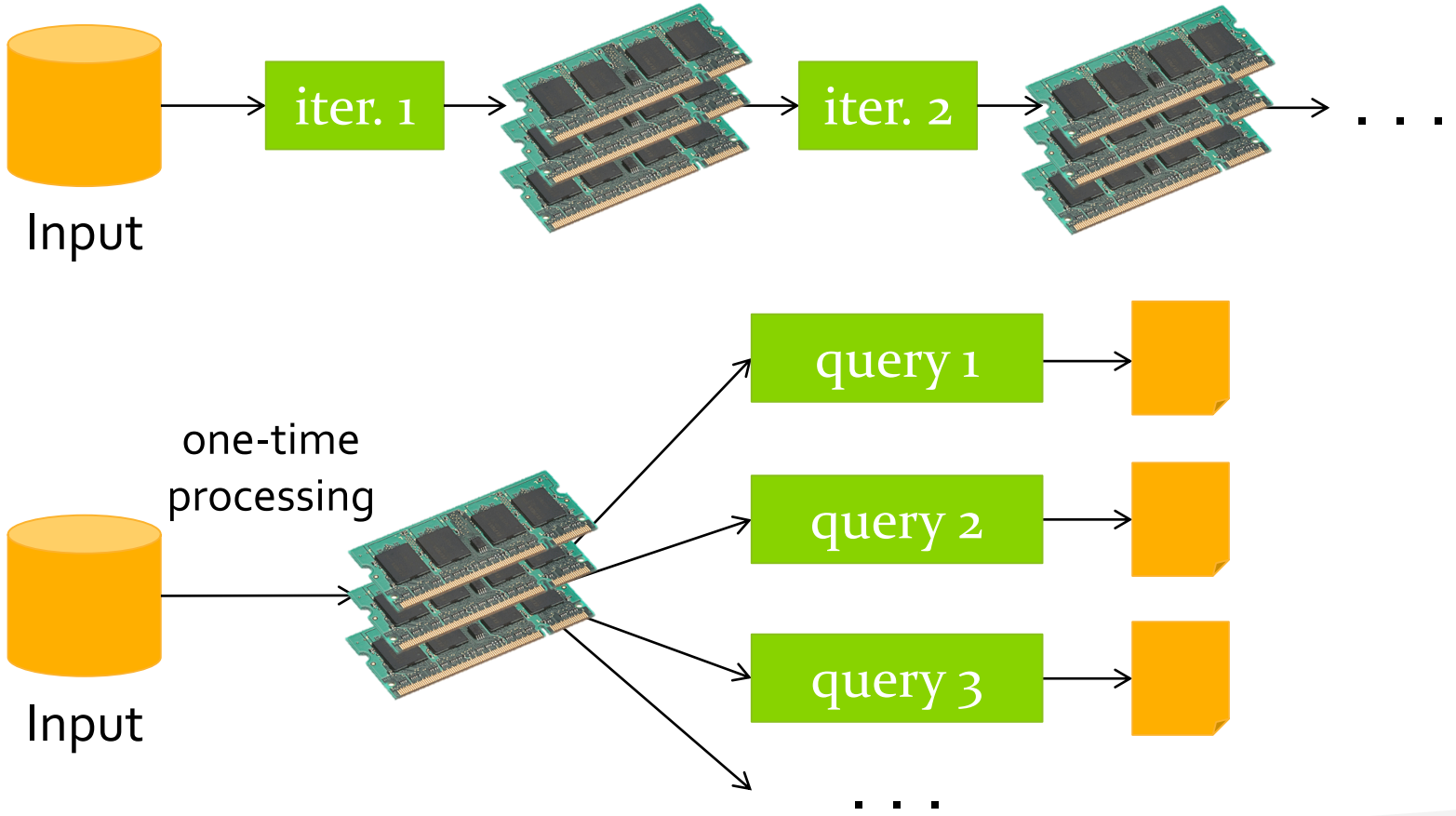


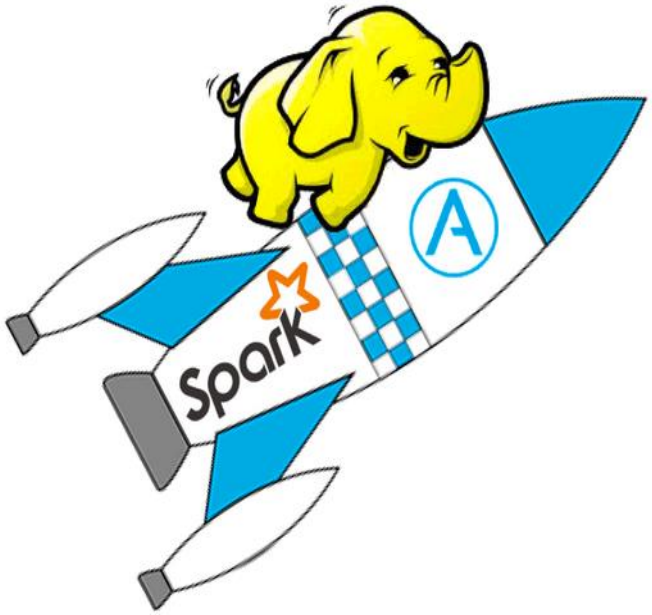
Limitation of Hadoop Map Reduce



- Slow due to replication
- Inefficient for:
  - Iterative algorithms (Machine Learning, Graphs & network analysis)
  - Interactive Data Mining (R)

Why Spark as Solution: In-Memory Data Sharing

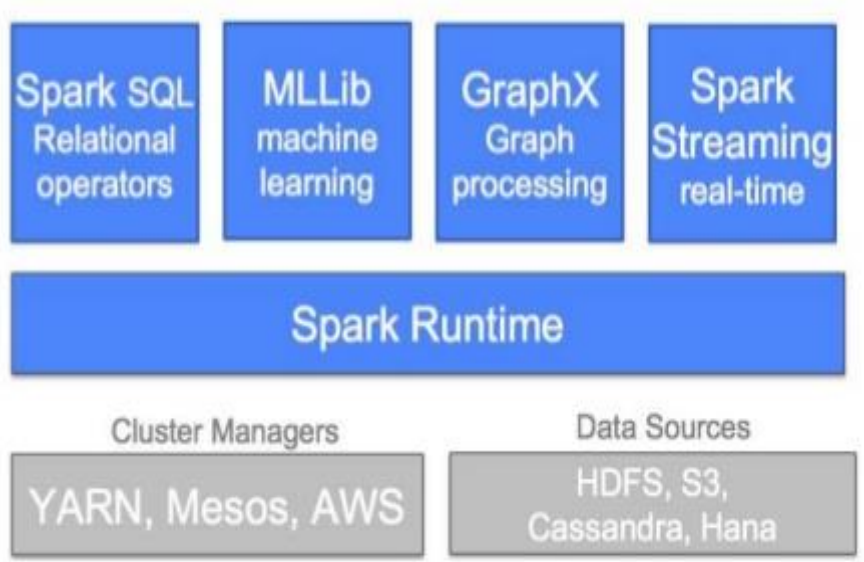


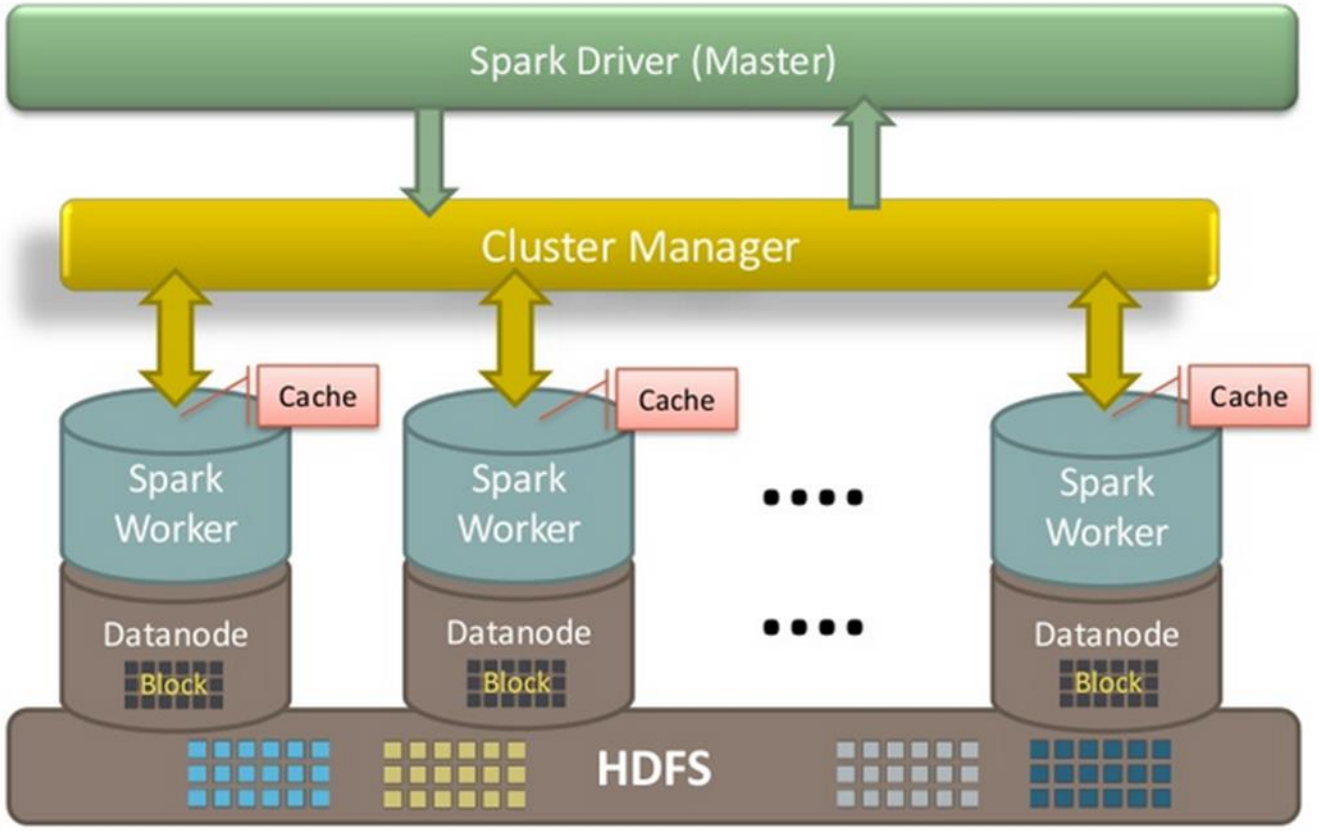


What is Spark

- A Big Data analytics cluster-computing framework written in scala
- **Open sourced** originally developed at AMP Lab @ UC Berkely
- Provides **In-Memory** analytics which is faster than Hadoop/Hive (Up to 100x)
- Designed for **iterative** algorithms and interactive analytics
- Highly compatible with Hadoop's storage APIs.
  - Can run on existing Hadoop Cluster Setup
- Developers can write driver programs in using multiple languages

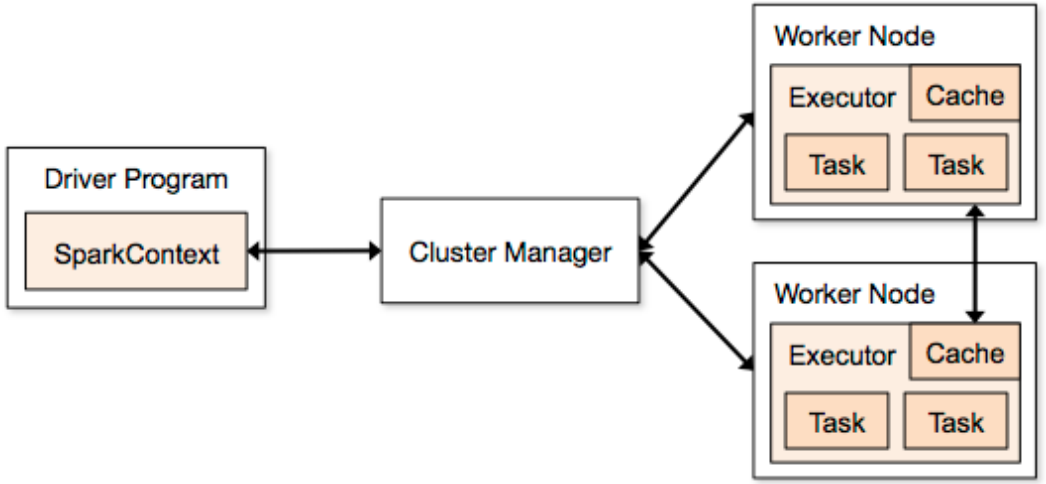
The Spark Stack & Architecture



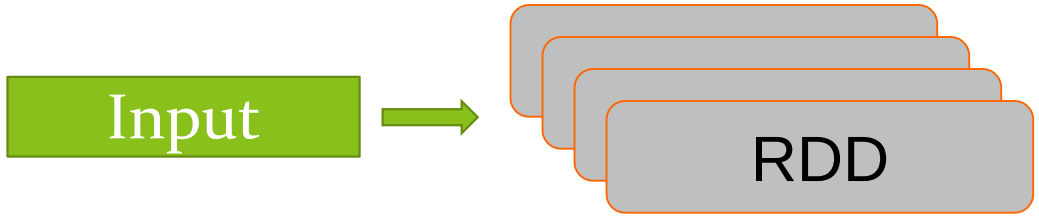


Core Spark Concepts

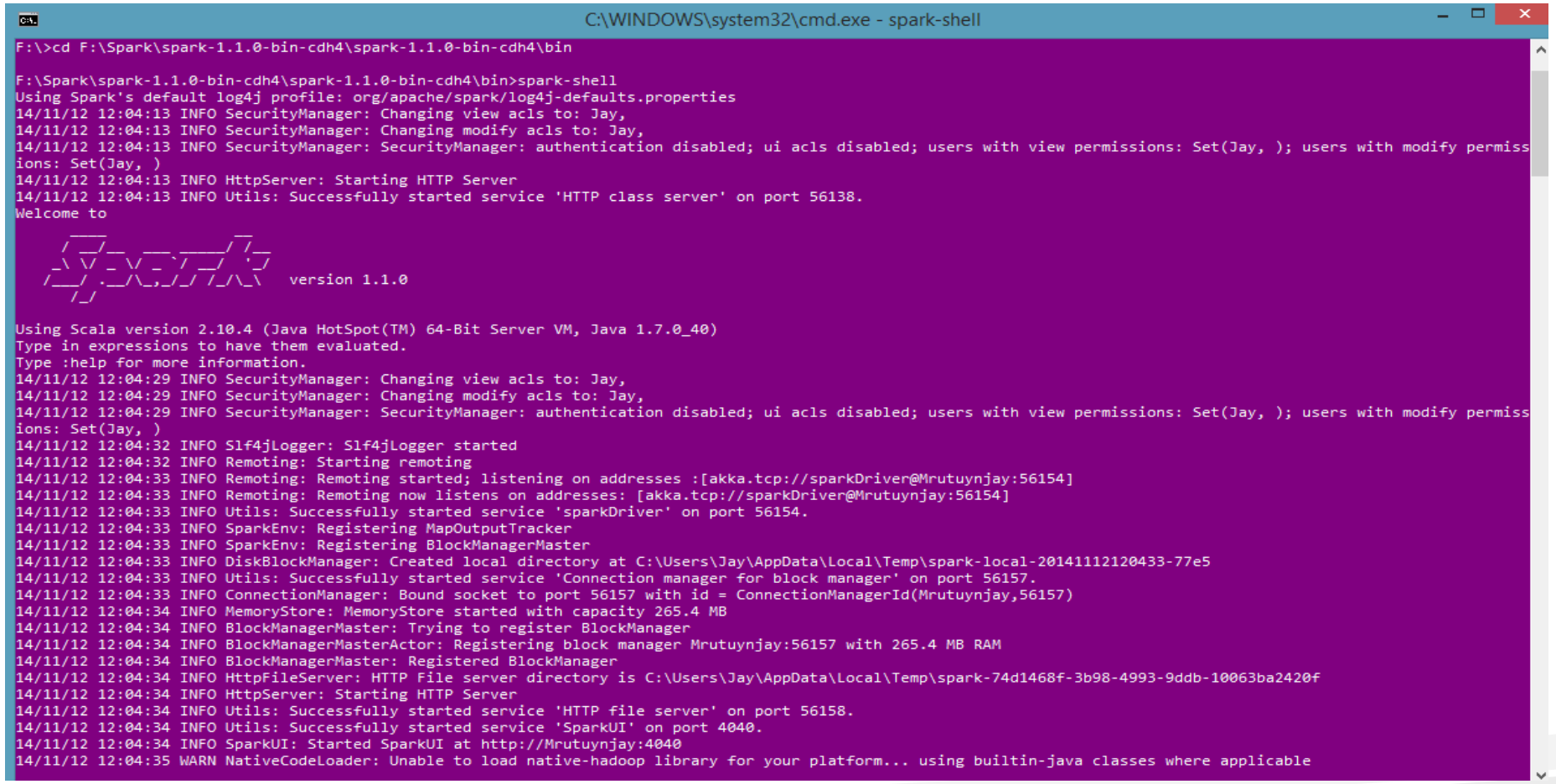
- **Spark Context: (Spark Context Object)**
  - Driver programs access point.
  - Represents a connection to a computing cluster.
  - Used to build *resilient distributed datasets or RDDs*



- **RDD: Resilient Distributed Datasets**
  - Immutable data structure
  - Fault Tolerant
  - Parallel Data Structure
  - In-Memory (Explicitly)



Spark-Shell


C:\WINDOWS\system32\cmd.exe - spark-shell
F:\>cd F:\Spark\spark-1.1.0-bin-cdh4\spark-1.1.0-bin-cdh4\bin
F:\Spark\spark-1.1.0-bin-cdh4\spark-1.1.0-bin-cdh4\bin>spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
14/11/12 12:04:13 INFO SecurityManager: Changing view acls to: Jay,
14/11/12 12:04:13 INFO SecurityManager: Changing modify acls to: Jay,
14/11/12 12:04:13 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(Jay,); users with modify permissions: Set(Jay,)
14/11/12 12:04:13 INFO HttpServer: Starting HTTP Server
14/11/12 12:04:13 INFO Utils: Successfully started service 'HTTP class server' on port 56138.
Welcome to
version 1.1.0
Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_40)
Type in expressions to have them evaluated.
Type :help for more information.
14/11/12 12:04:29 INFO SecurityManager: Changing view acls to: Jay,
14/11/12 12:04:29 INFO SecurityManager: Changing modify acls to: Jay,
14/11/12 12:04:29 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(Jay,); users with modify permissions: Set(Jay,)
14/11/12 12:04:32 INFO Slf4jLogger: Slf4jLogger started
14/11/12 12:04:32 INFO Remoting: Starting remoting
14/11/12 12:04:33 INFO Remoting: Remoting started; listening on addresses :[akka.tcp://sparkDriver@Mrutuynjay:56154]
14/11/12 12:04:33 INFO Remoting: Remoting now listens on addresses: [akka.tcp://sparkDriver@Mrutuynjay:56154]
14/11/12 12:04:33 INFO Utils: Successfully started service 'sparkDriver' on port 56154.
14/11/12 12:04:33 INFO SparkEnv: Registering MapOutputTracker
14/11/12 12:04:33 INFO SparkEnv: Registering BlockManagerMaster
14/11/12 12:04:33 INFO DiskBlockManager: Created local directory at C:\Users\Jay\AppData\Local\Temp\spark-local-20141112120433-77e5
14/11/12 12:04:33 INFO Utils: Successfully started service 'Connection manager for block manager' on port 56157.
14/11/12 12:04:33 INFO ConnectionManager: Bound socket to port 56157 with id = ConnectionManagerId(Mrutuynjay,56157)
14/11/12 12:04:34 INFO MemoryStore: MemoryStore started with capacity 265.4 MB
14/11/12 12:04:34 INFO BlockManagerMaster: Trying to register BlockManager
14/11/12 12:04:34 INFO BlockManagerMasterActor: Registering block manager Mrutuynjay:56157 with 265.4 MB RAM
14/11/12 12:04:34 INFO BlockManagerMaster: Registered BlockManager
14/11/12 12:04:34 INFO HttpFileServer: HTTP File server directory is C:\Users\Jay\AppData\Local\Temp\spark-74d1468f-3b98-4993-9ddb-10063ba2420f
14/11/12 12:04:34 INFO HttpServer: Starting HTTP Server
14/11/12 12:04:34 INFO Utils: Successfully started service 'HTTP file server' on port 56158.
14/11/12 12:04:34 INFO Utils: Successfully started service 'SparkUI' on port 4040.
14/11/12 12:04:34 INFO SparkUI: Started SparkUI at http://Mrutuynjay:4040
14/11/12 12:04:35 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

Hello World!! : Text Search

Spark Context

```
scala> val lines = sc.textFile("README.md")
14/11/12 12:06:39 INFO MemoryStore: ensureFreeSpace(78179) called with curMem=0, maxMem=278302556
14/11/12 12:06:39 INFO MemoryStore: Block broadcast_0 stored as values in memory (estimated size 76.3 KB, free 265.3 MB)
lines: org.apache.spark.rdd.RDD[String] = README.md MappedRDD[1] at textFile at <console>:12
```

RDD is created

RDD Operation: ***Transformation***

```
scala> val pythonLines = lines.filter(line => line.contains("Python"))
pythonLines: org.apache.spark.rdd.RDD[String] = FilteredRDD[11] at filter at <console>:14
```

```
scala> pythonLines.count()
14/11/12 13:21:33 INFO SparkContext: Starting job: count at <console>:17
14/11/12 13:21:33 INFO DAGScheduler: Got job 5 (count at <console>:17) with 2 output partitions (allowLocal=false)
res12: Long = 3
```

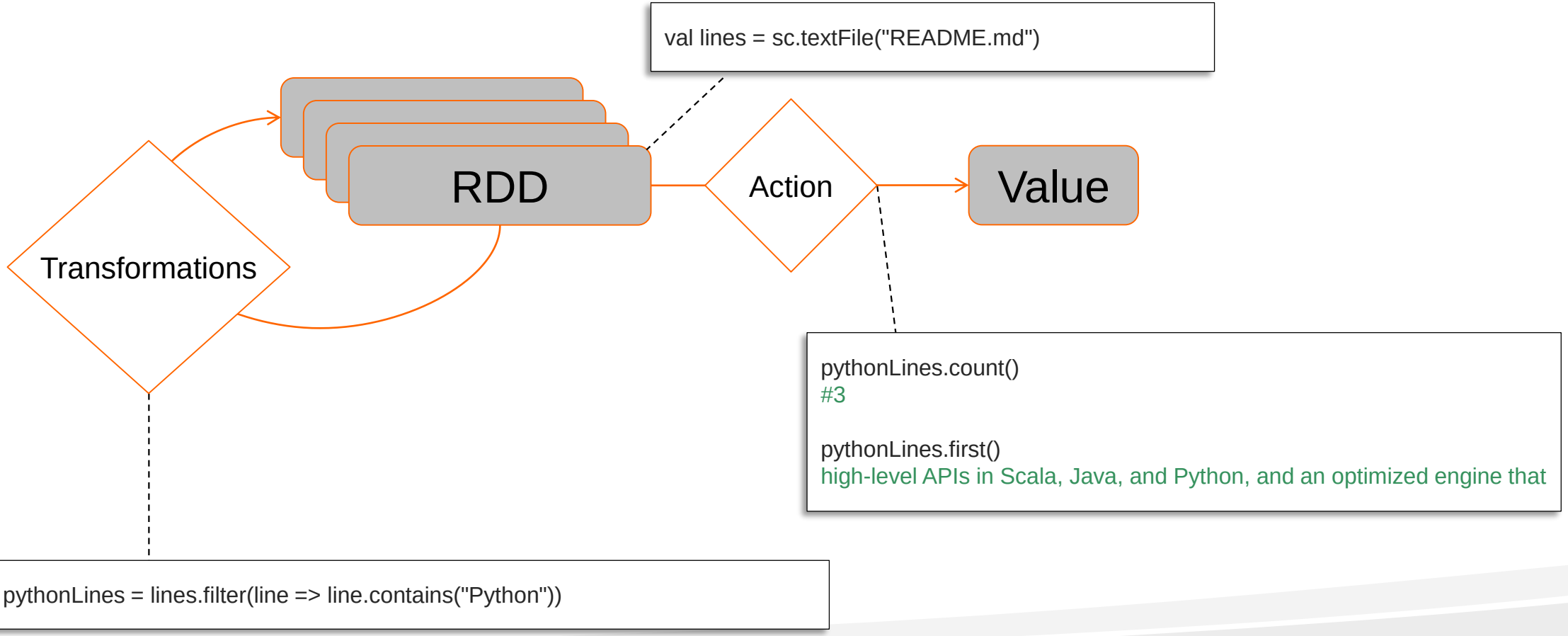
RDD Operation: ***Action***

```
scala> pythonLines.first()
14/11/12 13:21:39 INFO SparkContext: Starting job: first at <console>:17
14/11/12 13:21:39 INFO DAGScheduler: Got job 6 (first at <console>:17) with 1 output partitions (allowLocal=true)
res13: String = high-level APIs in Scala, Java, and Python, and an optimized engine that
```

RDD: Resilient Distributed Dataset

- An RDD in Spark is simply a distributed collection of objects.
- Each RDD is split into multiple *partitions*, which may be computed on different nodes of the cluster.
- RDDs can contain any type of Python, Java or Scala objects, including user-defined classes
- RDD Created in two ways
  - 1. Loading the external data
  - 2. By distributing a collection of objects in from driver program
- Operation on RDD:
  - ***Transformation****:* construct a new RDD from a previous one
  - ***Actions****:* compute a result based on an RDD, and either return it to the driver program or save it to an external storage system
- Spark computes RDDs in a ***lazy fashion***

Resilient Distributed Dataset Contd...



RDD Operations

Transformations

- Create new dataset from and existing one
- Lazy in nature. They are executed only when some action is performed
- Example:
  - Map()
  - Filter()
  - Distinct()

Actions

- Returns to the driver program a value or exports data to a storage system after performing a computation
- Example:
  - Count()
  - Reduce()
  - Collect()
  - Take()

Persistence

- For caching data in memory for future operations.
- Options to store on disk or RAM or mixed (Storage Level)
- Example:
  - Persist()
  - Cache()

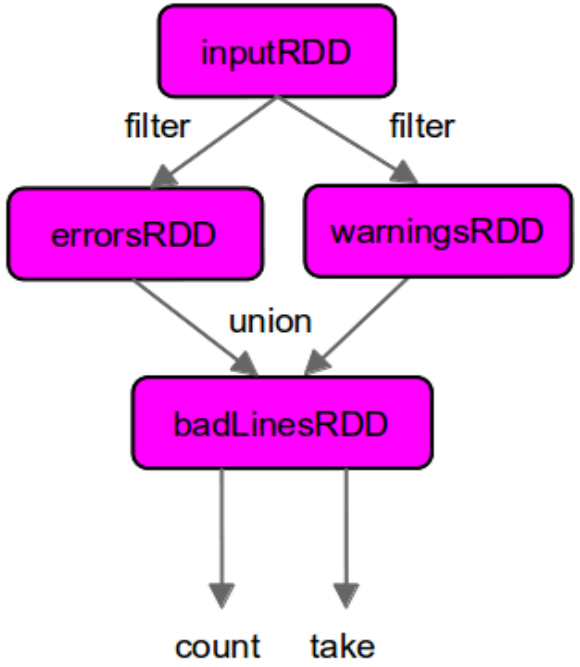
RDD: Lazy Fashion & persist/Caching

- **Lazy Fashion:**
  - RDDs are computed actually created the first time they are used in an action.
- **Persist:**
  - Spark's RDDs are by default recomputed each time you run an action on them.
  - Want to use RDD in multiple actions, ask Spark to store it in memory by using *persist. i.e. RDD.persist()*
  - RDD.unpersist(): To remove from the cache
  - Persisting RDDs on disk instead of memory is also possible

Level	Space Used	CPU time	In memory	On Disk	Nodes with data	Comments
MEMORY_ONLY	Low	Low	Y		1	
MEMORY_ONLY_2	Low	Low	Y		2	
MEMORY_AND_DISK	High	Medium	Some	Some	1	Spills to disk if there is too much data to fit in memory.
DISK_ONLY	Low	High	N	Y	1	
DISK_ONLY_2	Low	High	N	Y	2	

RDD Fault Tolerance

- Spark keeps track of the set of dependencies between different RDDs, called the *lineage graph*.
- It uses this information to compute each RDD on demand and to recover lost data if part of a persistent RDD is lost.

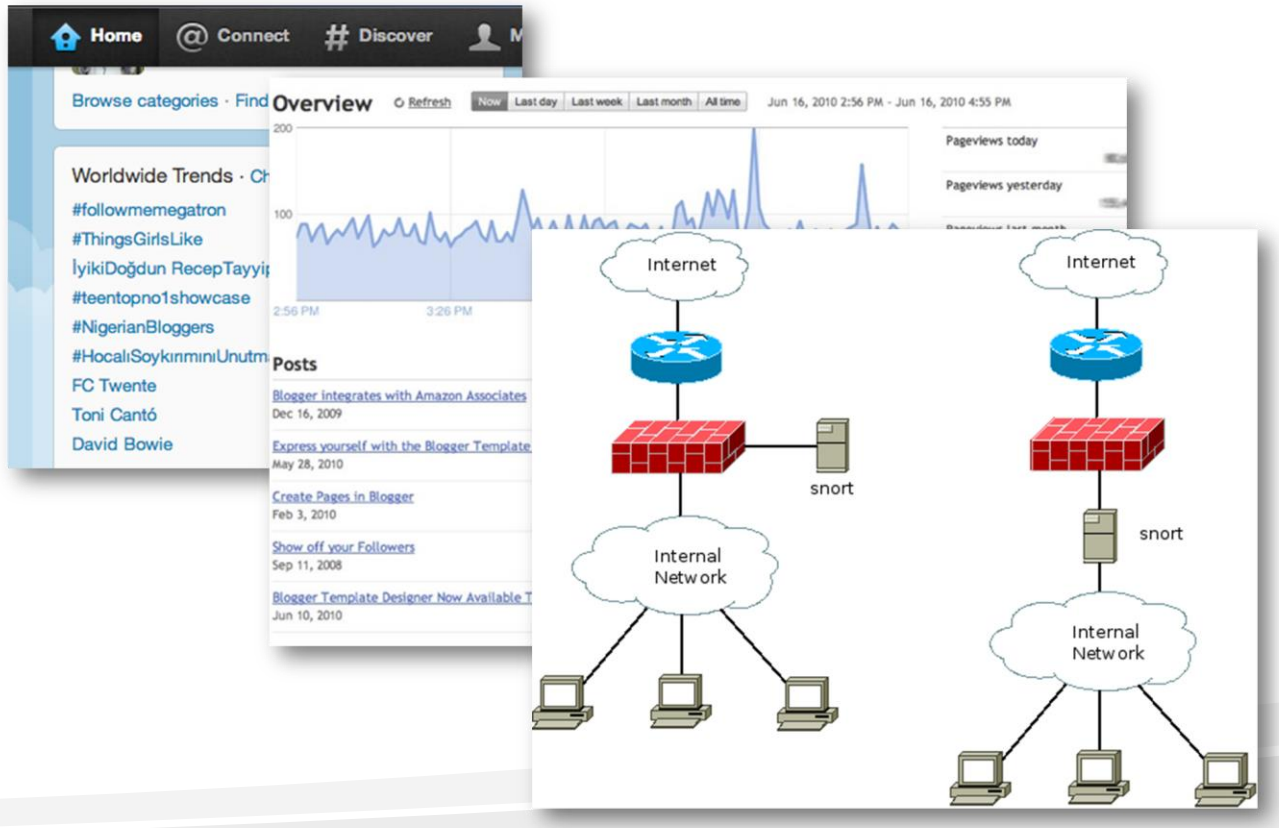


Spark Streaming

- Framework for large scale stream processing
  - Scales to 100s of nodes
  - Can achieve second scale latencies
  - Integrates with Spark's batch and interactive processing
  - Provides a simple batch-like API for implementing complex algorithm
  - Can absorb live data streams from Kafka, Flume, ZeroMQ, etc.

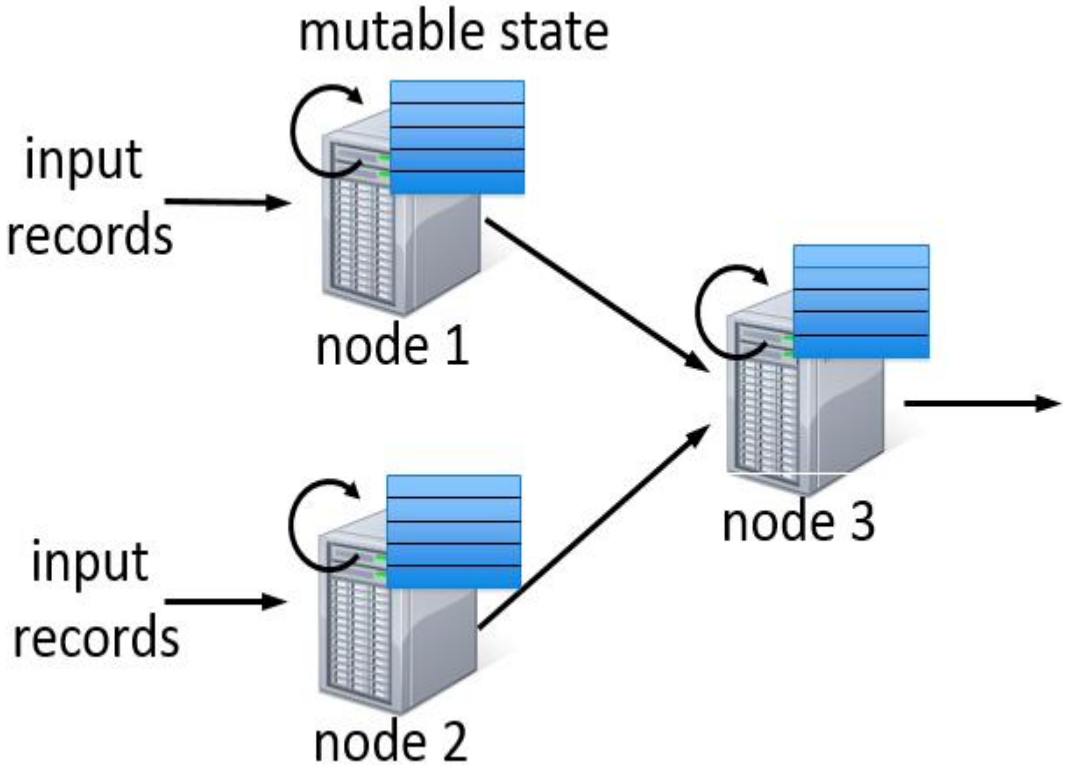
Motivation

- Many important applications must process large streams of live data and provide results in near-real-time
  - Social network trends
  - Website statistics
  - Intrustion detection systems
  - etc.
- Require large clusters to handle workloads
- Require latencies of few seconds



Stateful Stream Processing

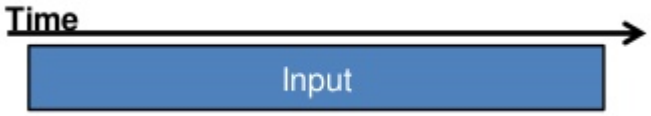
- Traditional streaming systems have a event-driven **record-at-a-time** processing model
  - Each node has mutable state
  - For each record, update state & send new records
- State is lost if node dies!
- Making stateful stream processing be fault-tolerant is challenging

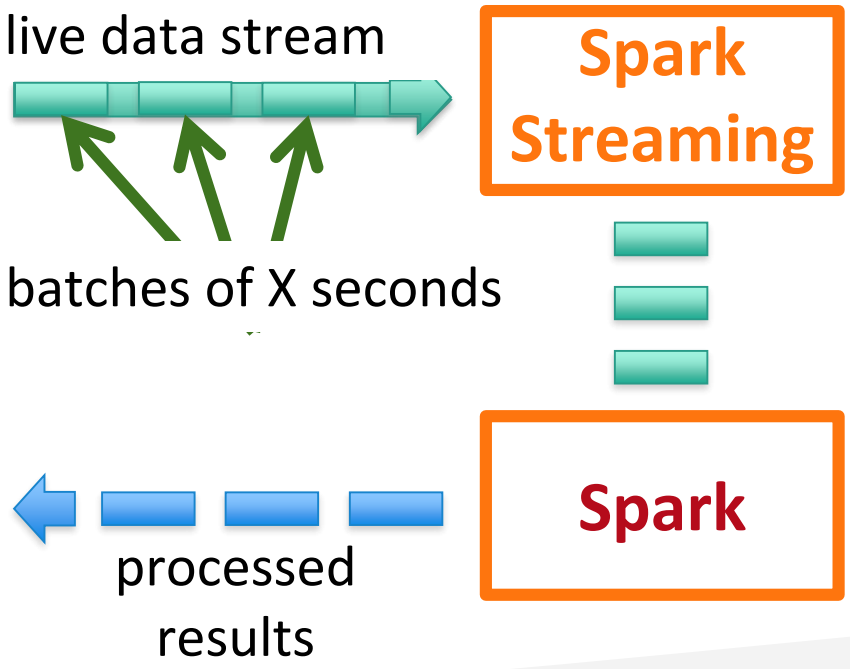


Discretized Stream Processing

Run a streaming computation as a **series of very small, deterministic batch jobs**

- Chop up the live stream into batches of X seconds
- Spark treats each batch of data as RDDs and processes them using RDD operations
- Finally, the processed results of the RDD operations are returned in batches

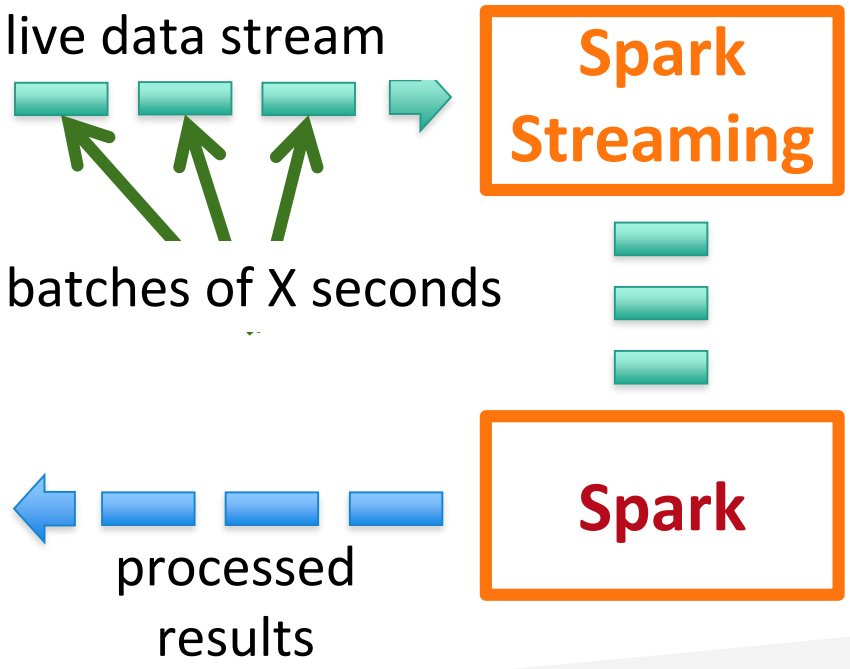




Discretized Stream Processing

Run a streaming computation as a **series of very small, deterministic batch jobs**

- Batch sizes as low as ½ second, latency ~ 1 second
- Potential for combining batch processing and streaming processing in the same system



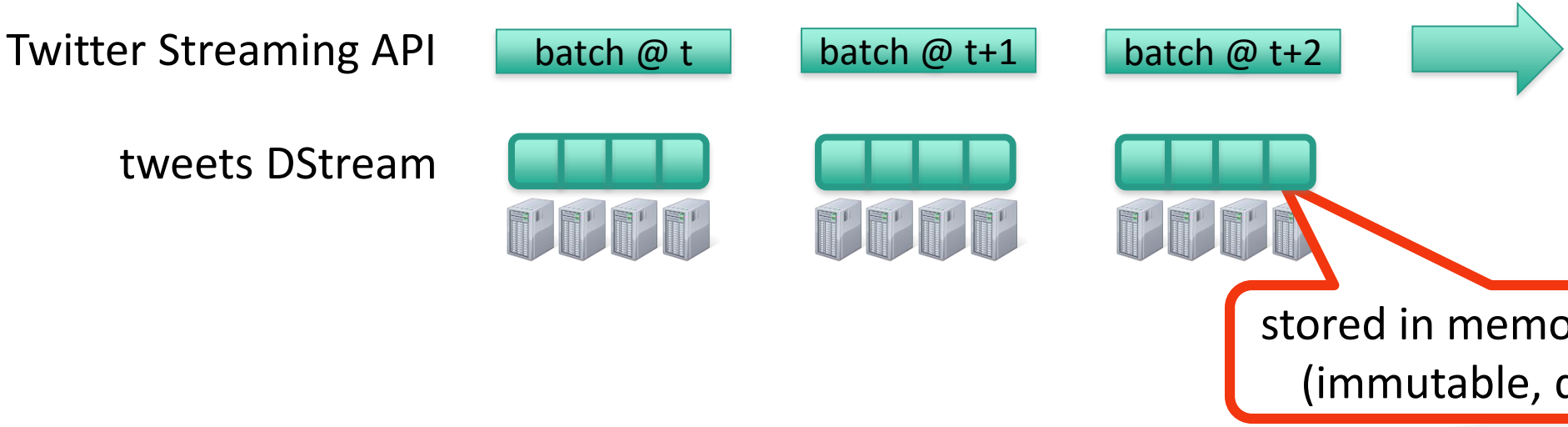
Key concepts

- **DStream** – sequence of RDDs representing a stream of data
  - Twitter, HDFS, Kafka, Flume, ZeroMQ, Akka Actor, TCP sockets
- **Transformations** – modify data from on DStream to another
  - Standard RDD operations – map, countByValue, reduce, join, …
  - Stateful operations – window, countByValueAndWindow, …
- **Output Operations – send data to external entity**
  - saveAsHadoopFiles – saves to HDFS
  - foreach – do anything with each batch of results

Example 1 – Get hashtags from Twitter

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
```

DStream: a sequence of RDD representing a stream of data

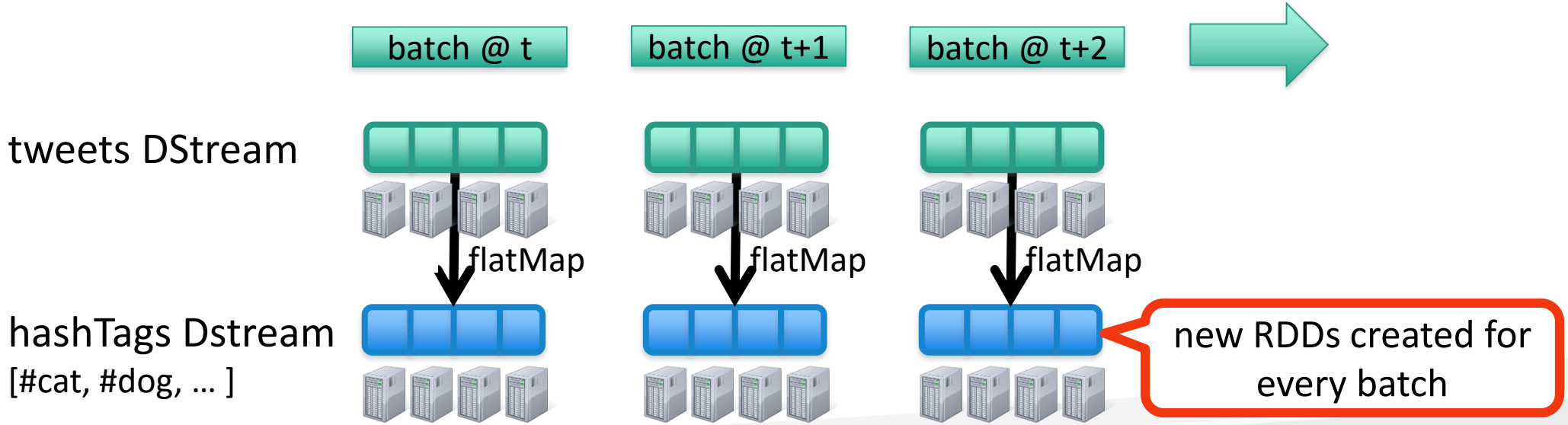


Example 1 – Get hashtags from Twitter

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)

val hashTags = tweets.flatMap (status => getTags(status))
```

new DStream

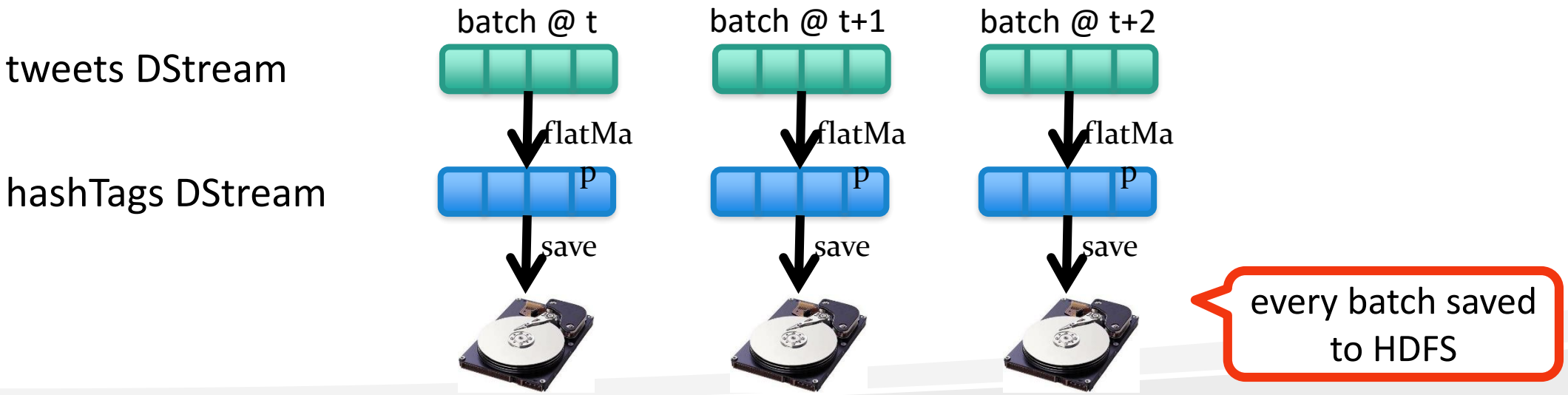
transformation: modify data in one Dstream to create another DStream



Example 1 – Get hashtags from Twitter

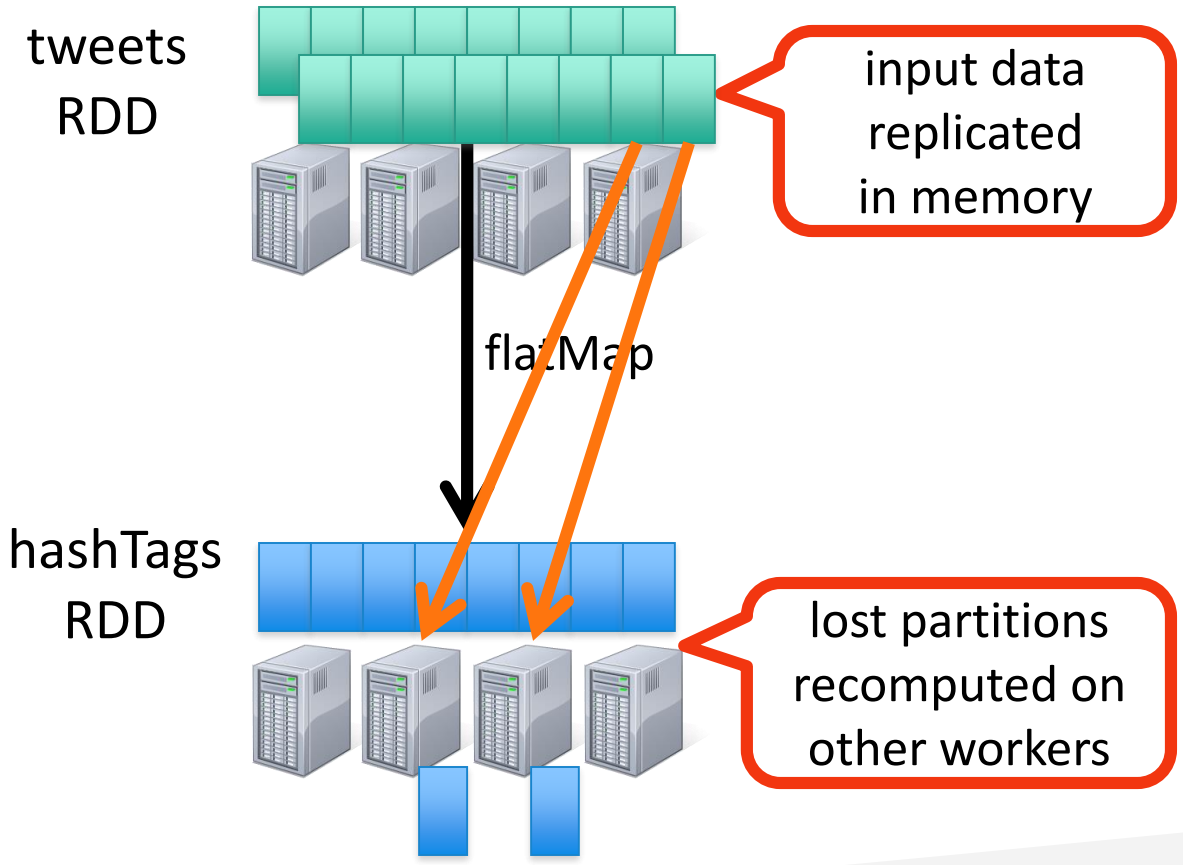
```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)

val hashTags = tweets.flatMap (status => getTags(status))

hashTags.saveAsHadoopFiles("hdfs://...")
```

output operation: to push data to external storage



Fault-tolerance

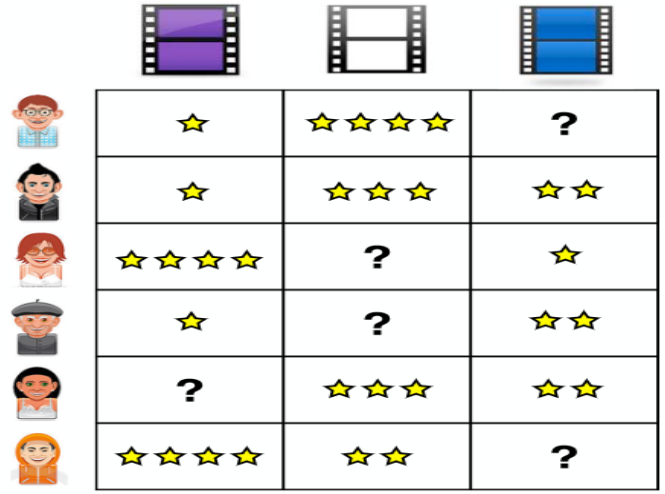
- RDDs are remember the sequence of operations that created it from the original fault-tolerant input data
- Batches of input data are replicated in memory of multiple worker nodes, therefore fault-tolerant
- Data lost due to worker failure, can be recomputed from input data

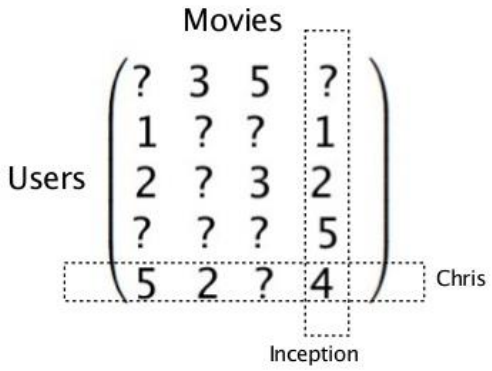


Spark: Machine Learning (MLlib)

- MLlib is a standard component of Spark providing machine learning primitives on top of Spark
- Algorithms supported by Mllib
  - **Classification:** SVM
  - **Regression:** Linear Regression, and random forests
  - **Collaborative Filtering**: Alternating Least Squares (ALS)
  - **Clustering:** K-means
  - **Dimensionality Reduction:** Singular Value Decomposition (SVD)
  - **Basic Statistics:** Summary Statistics, correlation, hypothesis testing
  - **Feature Extraction and Sampling**:

Collaborative Filtering





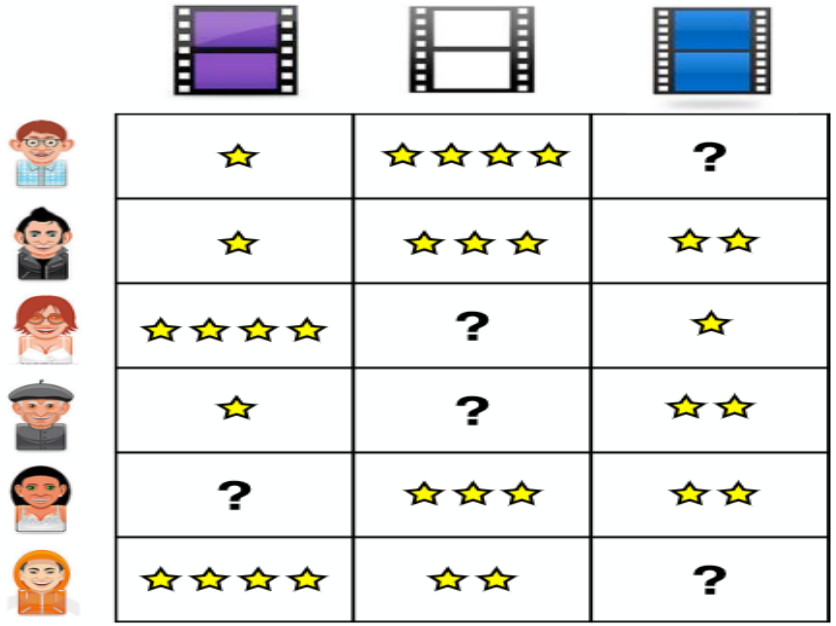
- Recover a rating matrix from a subset of its entries.

- ALS - wall-clock time

System	Wall-clock time (seconds)
MATLAB	15443
Mahout	4206
GraphLab	291
MLlib	481

Collaborative Filtering

```
// Load and parse the data
val data = sc.textFile("mllib/data/als/test.data")
val ratings = data.map(_.split(',') match {
    case Array(user, item, rate) =>
      Rating(user.toInt, item.toInt, rate.toDouble)
})

// Build the recommendation model using ALS
val numIterations = 20
val model = ALS.train(ratings, 1, 20, 0.01)

// Evaluate the model on rating data
val usersProducts = ratings.map { case Rating(user, product, rate) =>
  (user, product)
}
val predictions = model.predict(usersProducts)
```



Spark SQL

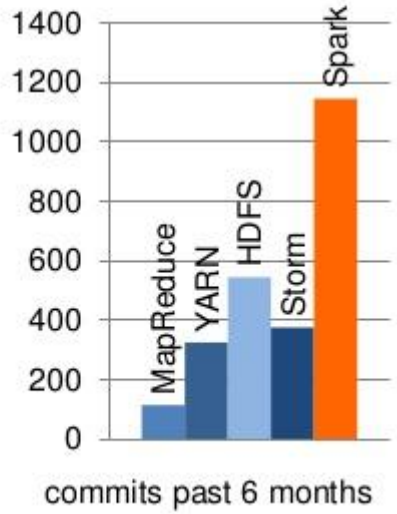
- **Spark SQL** unifies access to structured data.
- Load and query data from a variety of sources
- Run unmodified Hive queries on existing warehouses
- Connect through JDBC or ODBC.
- Spark SQL includes a server mode with industry standard JDBC and ODBC connectivity.

The Spark Community

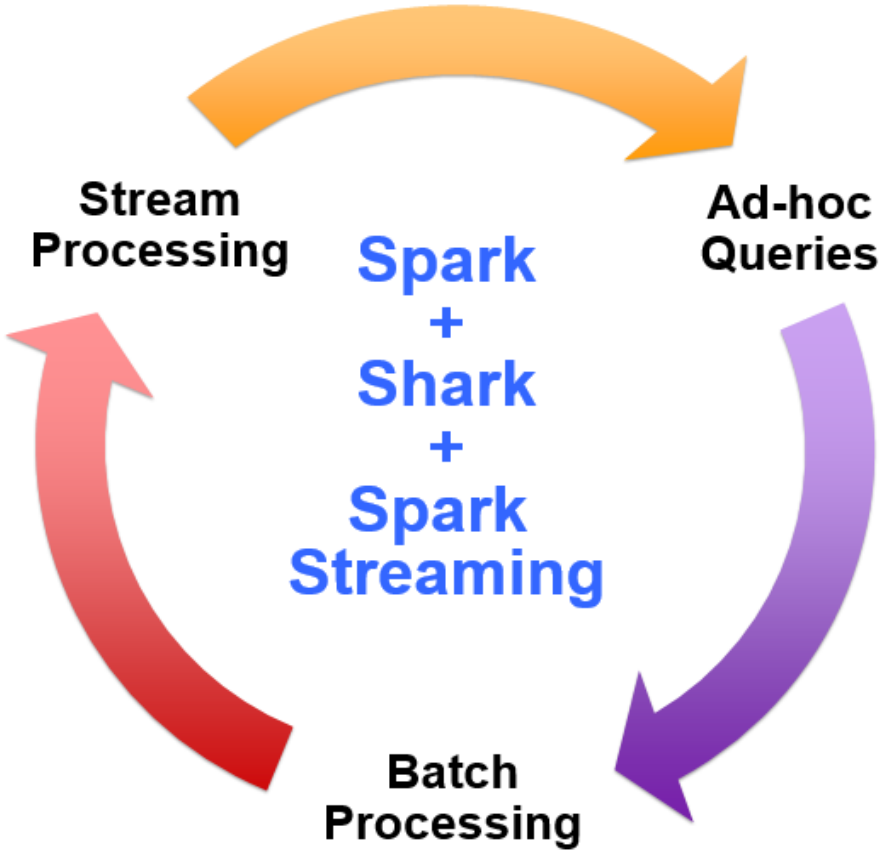
Spark Community

250+ developers, **50+** companies contributing

Most active open source project in big data



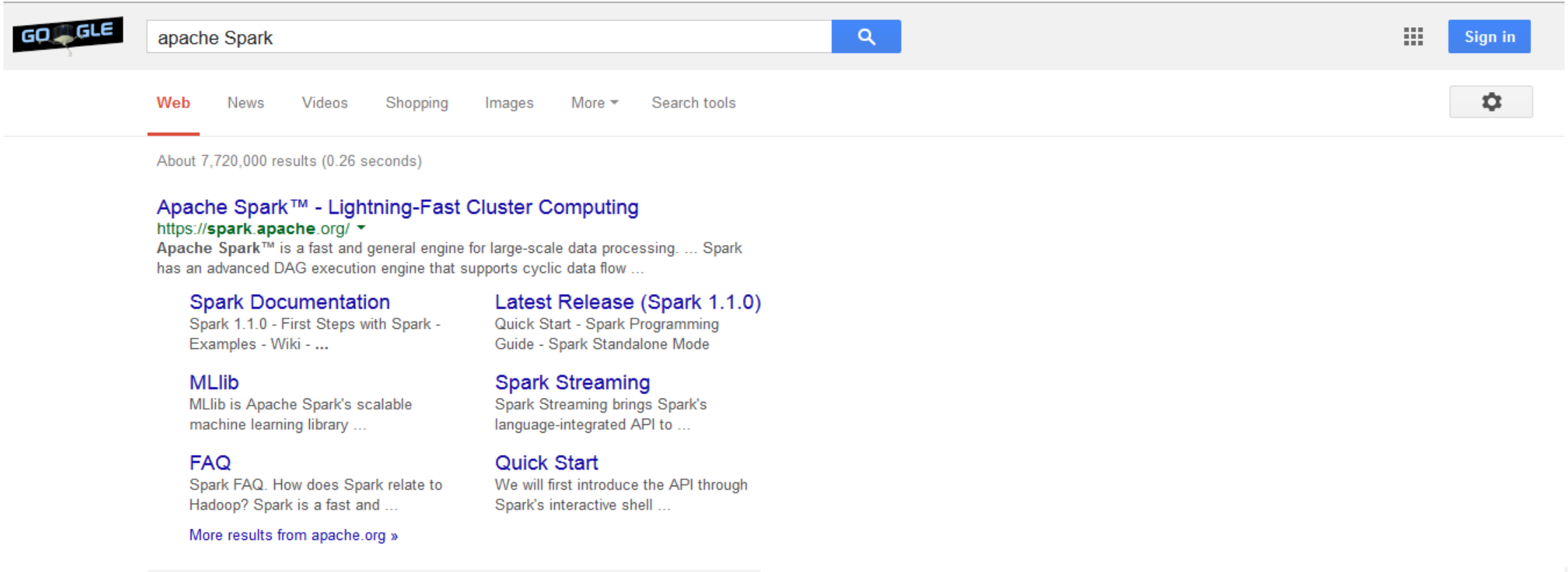
Vision - *one stack to rule them all*



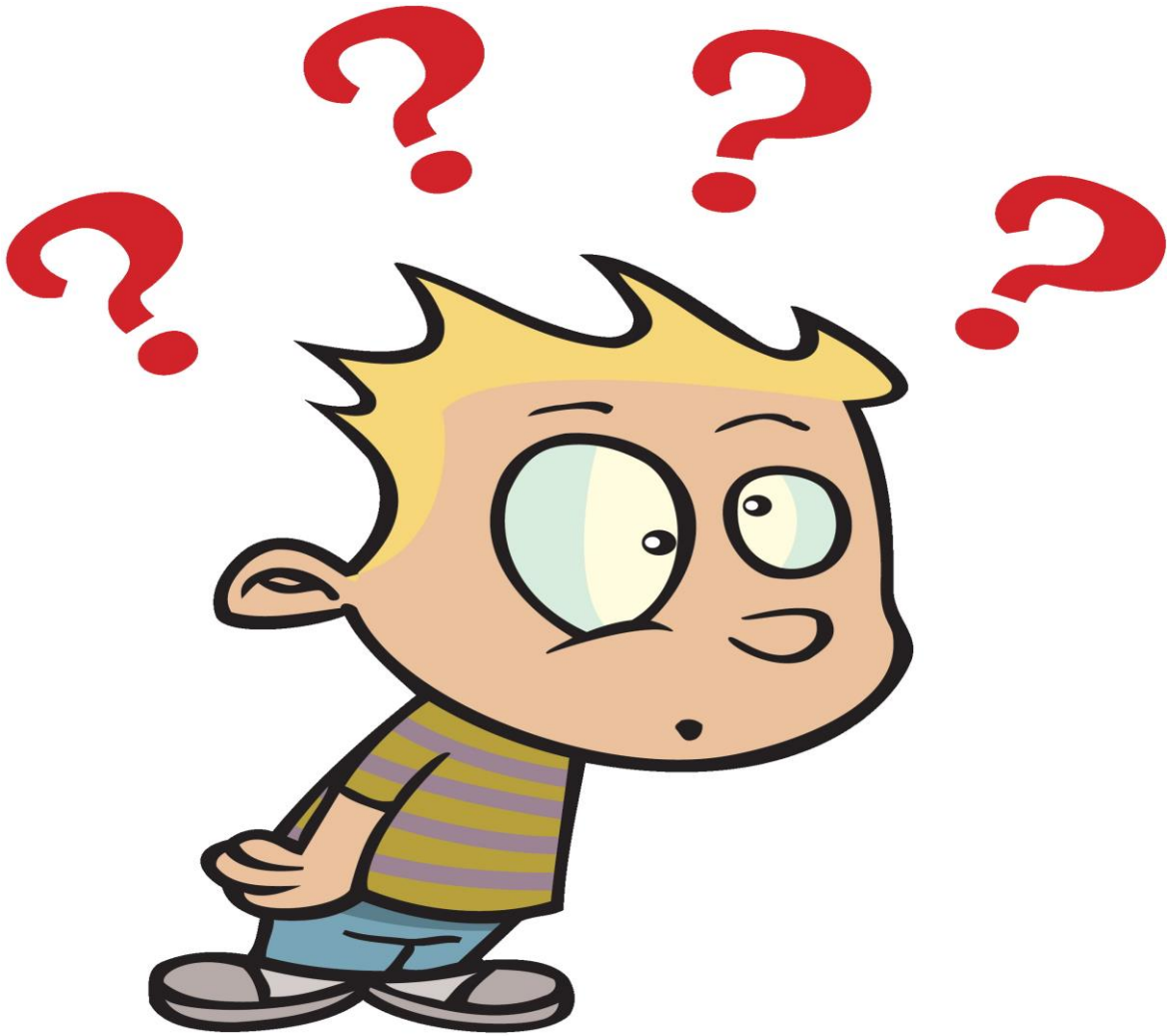
A New Feature Addition to MLlib

- ELM: Extreme Learning Machine.
  - A latest and fast learning method.
  - Based Single Layer Neural Network model.
  - Works 100x faster than the Backpropgation algorithm
  - More training accuracy compared SVM*
  - It used Singular Value Decomposition (SVD) for computation which is already supported by Spark MLlib
  - Very recent research publications (2014) prove parallel and distributed model of ELM

References (google Apache Spark)



Question?



Thank You!!!!

Video About Google:

https://www.youtube.com/watch?v=poysH2Glw5w