



BIG DATA SESSION 5

NICK KADOCHNIKOV

APACHE SPARK

WHAT IS SPARK?

- Fast and expressive cluster computing system compatible with Apache Hadoop
- Improves efficiency through:
 - General execution graphs
 - In-memory storage

→ Up to 10× faster on disk,
100× in memory
- Improves usability through:
 - Rich APIs in Java, Scala, Python, R
 - Interactive shell

→ 2-5× less code

HISTORY OF SPARK

- Spark was initially started by Matei Zaharia at UC Berkeley's AMPLab in 2009, and open sourced in 2010 under a BSD license.
- In 2013, the project was donated to the Apache Software Foundation and switched its license to Apache 2.0.
- Michael Franklin, who co-founded and directed AMPLab when Spark was created, is now professor and chair at the Computer Science Department of the University of Chicago.



SPARK IS STILL VERY YOUNG!

Version	Original release date	Latest version	Release date
0.5	2012-06-12	0.5.1	2012-10-07
0.6	2012-10-14	0.6.2	2013-02-07 ^[33]
0.7	2013-02-27	0.7.3	2013-07-16
0.8	2013-09-25	0.8.1	2013-12-19
0.9	2014-02-02	0.9.2	2014-07-23
1.0	2014-05-26	1.0.2	2014-08-05
1.1	2014-09-11	1.1.1	2014-11-26
1.2	2014-12-18	1.2.2	2015-04-17
1.3	2015-03-13	1.3.1	2015-04-17
1.4	2015-06-11	1.4.1	2015-07-15
1.5	2015-09-09	1.5.2	2015-11-09
1.6	2016-01-04	1.6.3	2016-11-07
2.0	2016-07-26	2.0.2	2016-11-14
2.1	2016-12-28	2.1.2	2017-10-09
2.2	2017-07-11	2.2.1	2017-12-01
2.3	2018-02-28	2.3.0	2018-02-28
Legend: Old version Older version, still supported Latest version Latest preview version			

BIG DATA AND SPARK

- Apache Spark is a computing platform designed to be *fast* and *general-purpose*, and *easy to use*
 - Speed
 - In-memory computations
 - Faster than MapReduce for complex applications on disk
 - Generality
 - Covers a wide range of workloads on one system
 - Batch applications (e.g. MapReduce)
 - Iterative algorithms
 - Interactive queries and streaming
 - Ease of use
 - APIs for Scala, Python, Java, R
 - Libraries for SQL, machine learning, streaming, and graph processing
 - Runs on Hadoop clusters or as a standalone
 - including the popular MapReduce model

BRIEF OVERVIEW OF SCALA

- Everything is an Object:
 - Primitive types such as numbers or boolean
 - Functions
- Numbers are objects
 - $1 + 2 * 3 / 4 \rightarrow (1).+(((2).*(3))./(x)))$
 - Where the $+$, $*$, $/$ are valid identifiers in Scala
- Functions are objects
 - Pass functions as arguments
 - Store them in variables
 - Return them from other functions
- Function declaration
 - `def functionName ([list of parameters]) : [return type]`

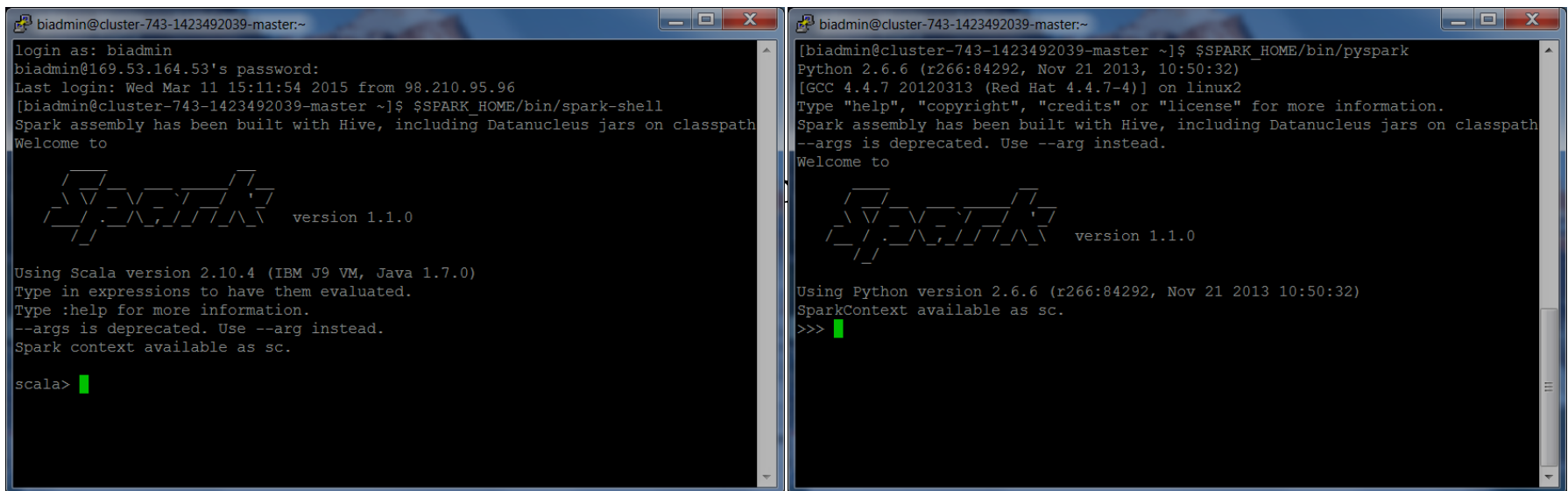
PYSPARK



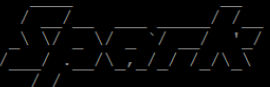

- Python API, pyspark, can be used
 - in batch mode
 - interactively
 - in a python command prompt in Jupyter notebook
 - in Jupyter notebook

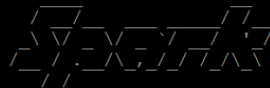

SPARK JOBS AND SHELL

- Spark jobs can be written in Scala, Python, Java or Spark R
- Spark shells for Scala and Python
- APIs are available for all four.
- Must adhere to the appropriate versions for each Spark release.



The image shows two terminal windows side-by-side, both titled 'biadmin@cluster-743-1423492039-master:~'. The left window shows the Spark Scala shell. It starts with a login prompt for 'biadmin' and a password prompt. After logging in, the user runs '\$SPARK_HOME/bin/spark-shell'. The terminal displays the Spark logo, 'version 1.1.0', and information about the Scala version (2.10.4) and Java VM (IBM J9 VM, Java 1.7.0). It also shows the Spark context available as 'sc'. The prompt is 'scala>'. The right window shows the Spark Python shell. It starts with the same login prompt. After logging in, the user runs '\$SPARK_HOME/bin/pyspark'. The terminal displays the Spark logo, 'version 1.1.0', and information about the Python version (2.6.6) and GCC version (4.4.7). It also shows the Spark context available as 'sc'. The prompt is '>>>'.

```
biadmin@cluster-743-1423492039-master:~  
login as: biadmin  
biadmin@169.53.164.53's password:  
Last login: Wed Mar 11 15:11:54 2015 from 98.210.95.96  
[biadmin@cluster-743-1423492039-master ~]$ $SPARK_HOME/bin/spark-shell  
Spark assembly has been built with Hive, including Datanucleus jars on classpath  
Welcome to  
 version 1.1.0  
Using Scala version 2.10.4 (IBM J9 VM, Java 1.7.0)  
Type in expressions to have them evaluated.  
Type :help for more information.  
--args is deprecated. Use --arg instead.  
Spark context available as sc.  
scala> 
```

```
biadmin@cluster-743-1423492039-master:~  
[biadmin@cluster-743-1423492039-master ~]$ $SPARK_HOME/bin/pyspark  
Python 2.6.6 (r266:84292, Nov 21 2013, 10:50:32)  
[GCC 4.4.7 20120313 (Red Hat 4.4.7-4)] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
Spark assembly has been built with Hive, including Datanucleus jars on classpath  
--args is deprecated. Use --arg instead.  
Welcome to  
 version 1.1.0  
Using Python version 2.6.6 (r266:84292, Nov 21 2013 10:50:32)  
SparkContext available as sc.  
>>> 
```

SPARK RDDs AND DATAFRAMES

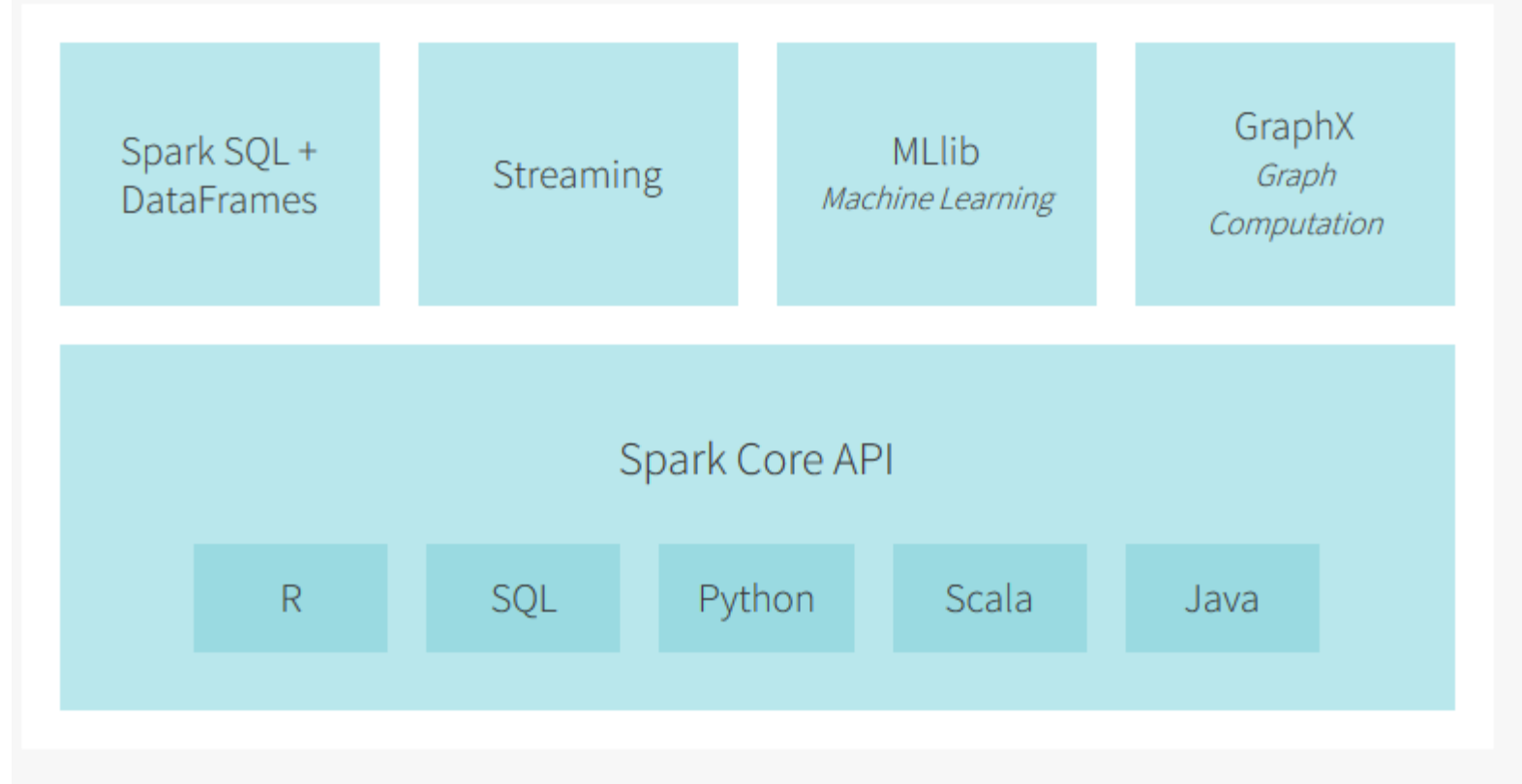
- Before Spark 2.0, the main abstraction of Spark was the Resilient Distributed Dataset – (RDD)
- Spark 2.0 added Spark DataFrame
 - We will learn both
- Advantages of Spark Dataframe:
 - SQL interface to data - more convenient to program (for SQL users)
 - Better query optimization (on SQL-like procedures)
- Advantages of RDDs:
 - More extensible operations
 - SparkML operations have only been partially extended to RDDs

SPARK STREAMING AND SPARKML

- There are two streaming libraries:
 - Structured Streaming - based on DataFrames,
 - DStreams - based on RDDs
- There are two APIs to machine learning library:
 - The old one based on RDD is in a maintenance state - only bug fixes are applied to it but no new features are introduced;
 - the new machine learning library based on DataFrames is still catching up with the functionality of the old library



Apache Spark Ecosystem



<https://databricks.com/spark/about>

WHICH LANGUAGE SHOULD I USE?

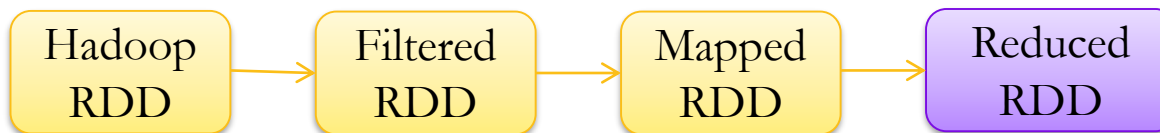
- Standalone programs can be written in any, but interactive shell is only Python & Scala
- **Python users:** can do Python for both
- **Java users:** consider learning Scala for shell
- Performance: Java & Scala are faster due to static typing, but Python is often fine
- R users should consider... switching to Python

SPARK APPROACH

- **Write programs in terms of transformations on distributed datasets**
- Concept: resilient distributed datasets (RDDs)
 - Collections of objects spread across a cluster
 - Built through parallel transformations (map, filter, etc.)
 - Automatically rebuilt on failure
 - Controllable persistence (e.g. caching in RAM)

RESILIENT DISTRIBUTED DATASETS (RDD)

- Spark's primary abstraction
- Distributed collection of elements
- Parallelized across the cluster
- Two types of RDD operations
 - Transformations
 - Creates a DAG
 - Lazy evaluations
 - No values returned
 - Actions
 - Performs the transformations and the action that follows
 - Returns a value
- Fault tolerance
- Caching
- Example of RDD flow.

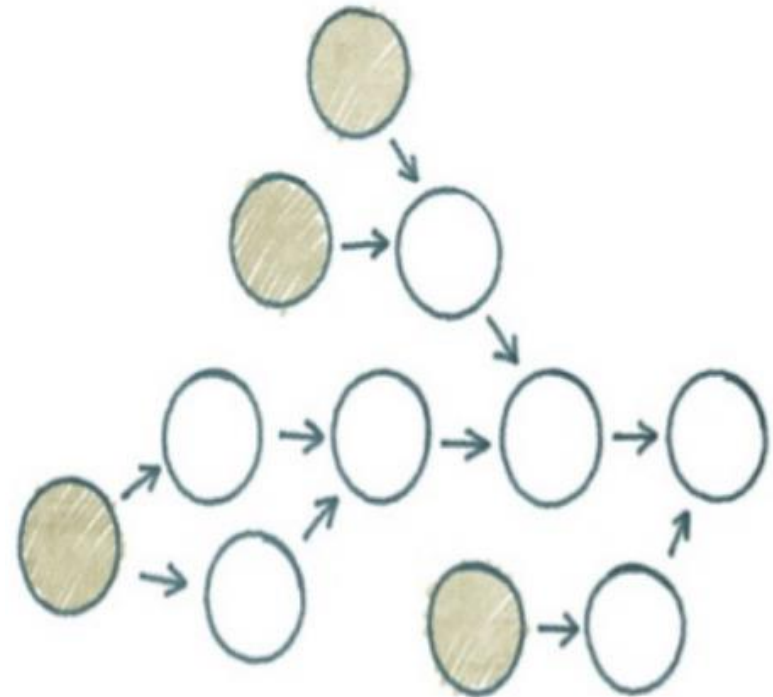


RESILIENT DISTRIBUTED DATASET (RDD)

- Fault-tolerant collection of elements that can be operated on in parallel.
- Immutable
- Three methods for creating RDD
 - Parallelizing an existing collection
 - Referencing a dataset
 - Transformation from an existing RDD
- Two types of RDD operations
 - Transformations
 - Actions
- Dataset from any storage supported by Hadoop
 - HDFS
 - Cassandra
 - HBase
 - Amazon S3, etc.
- Types of files supported:
 - Text files
 - SequenceFiles
 - Hadoop InputFormat

DIRECT ACYCLIC GRAPH (DAG)

- Directed
 - Only one direction
- Acyclic
 - No looping
- This supports fault tolerance

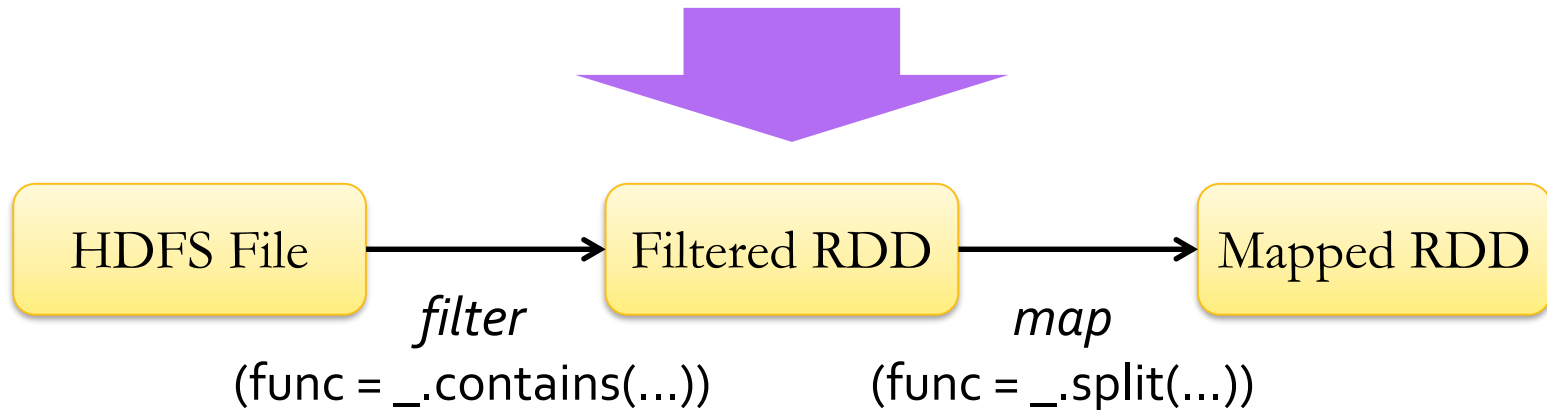


FAULT RECOVERY

RDDs track *lineage* information that can be used to efficiently recompute lost data

Ex:

```
msgs = textFile.filter(lambda s: s.startswith("ERROR"))  
               .map(lambda s: s.split("\t")[2])
```

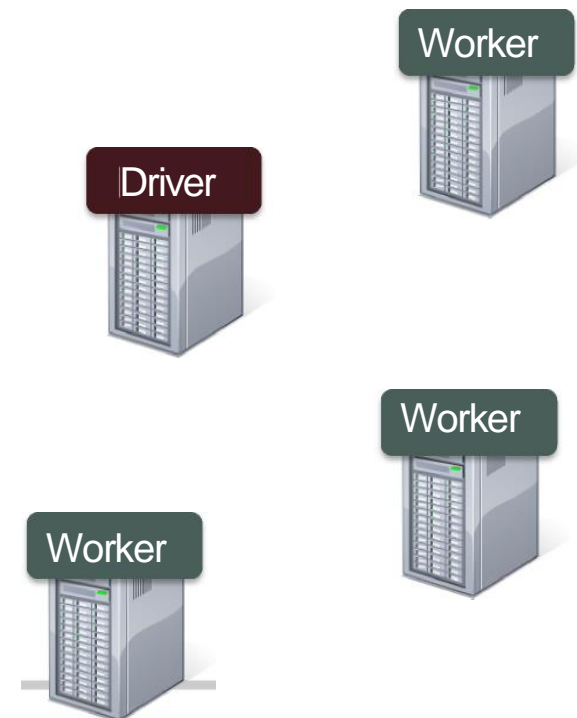


EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

EXAMPLE: LOG MINING

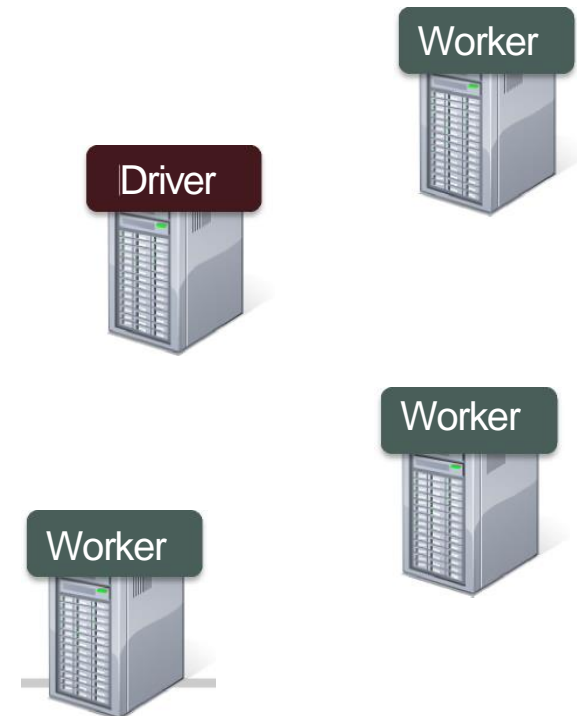
- Load error messages from a log into memory, then interactively search for various patterns



EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
```

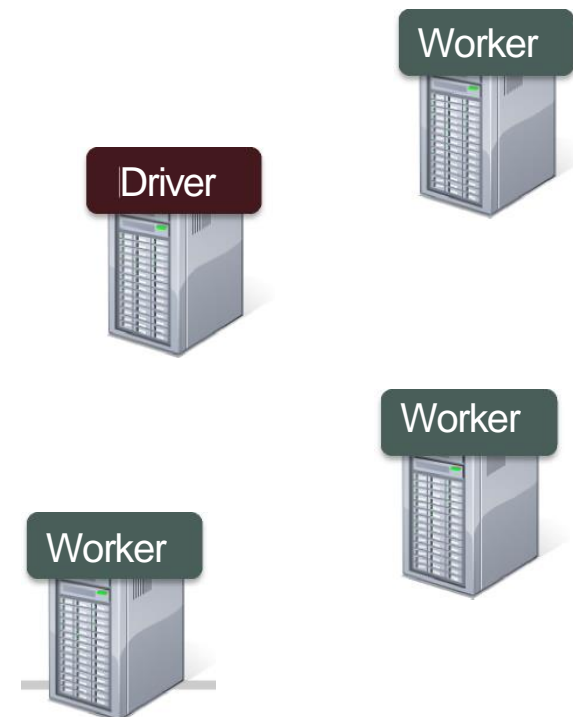


EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

Base RDD

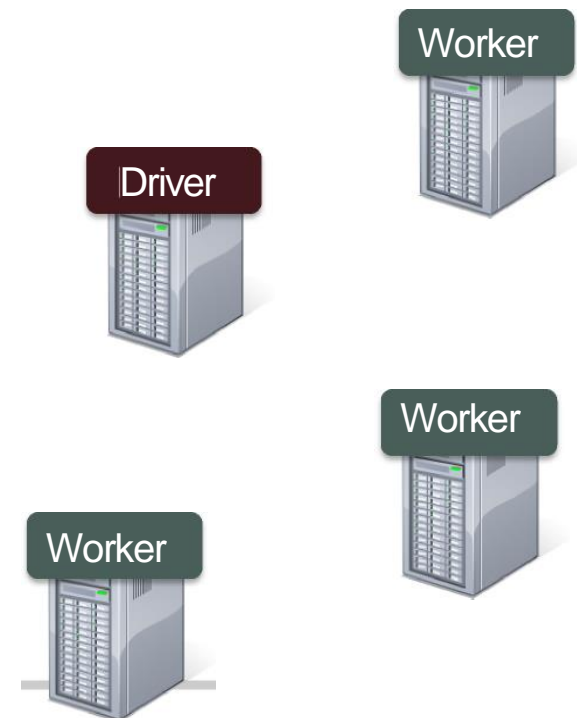
```
lines = spark.textFile("hdfs://...")
```



EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))
```



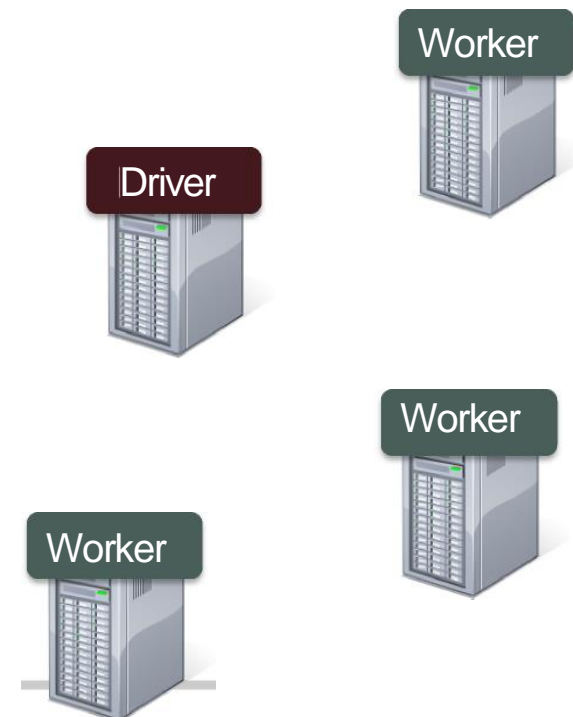
EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
```

Base RDD

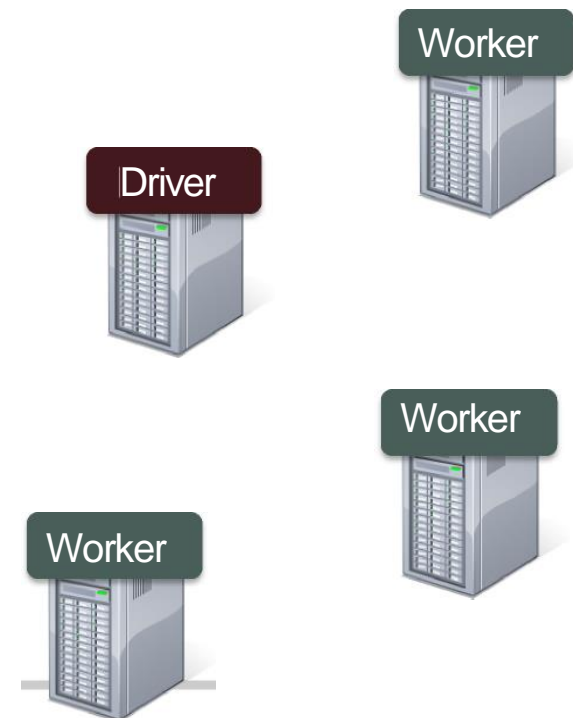
Transformed RDD



EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
```

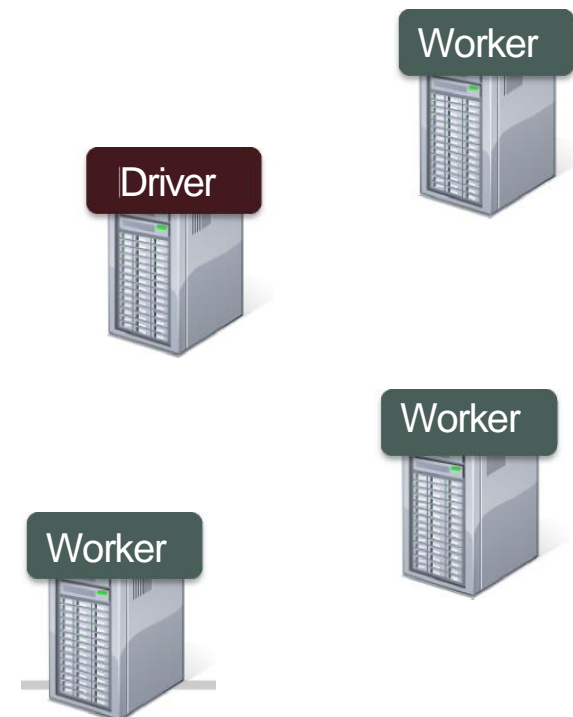


EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
```

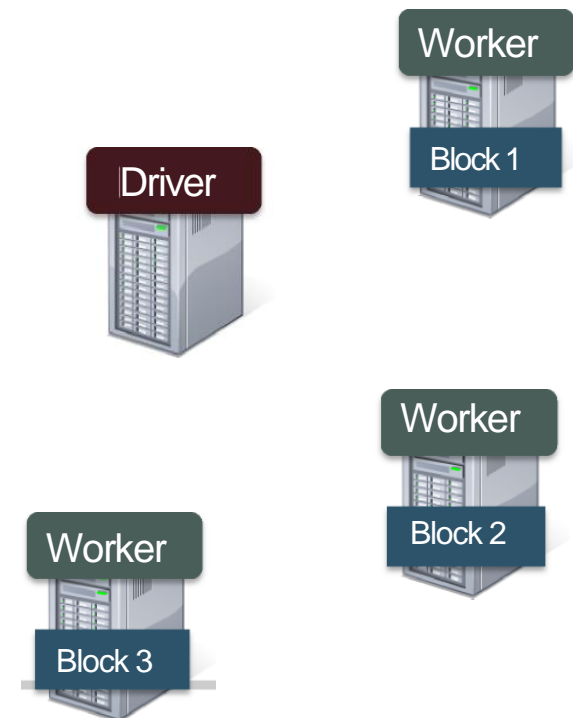
Action



EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

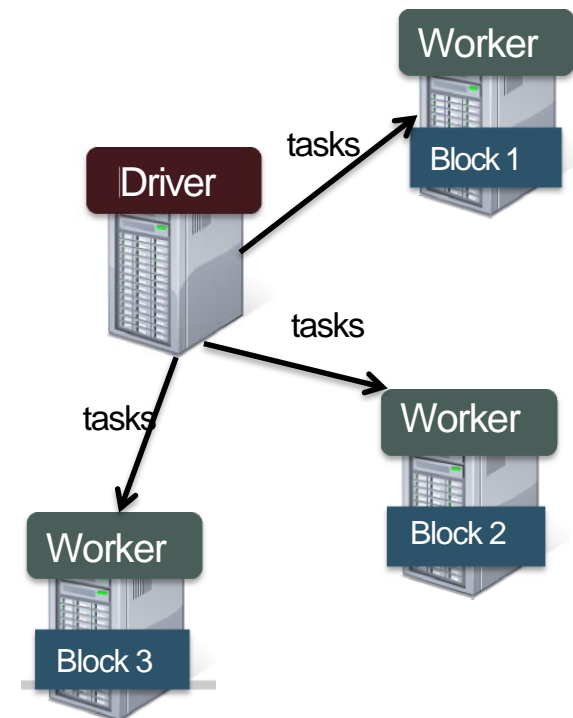
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
```



EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

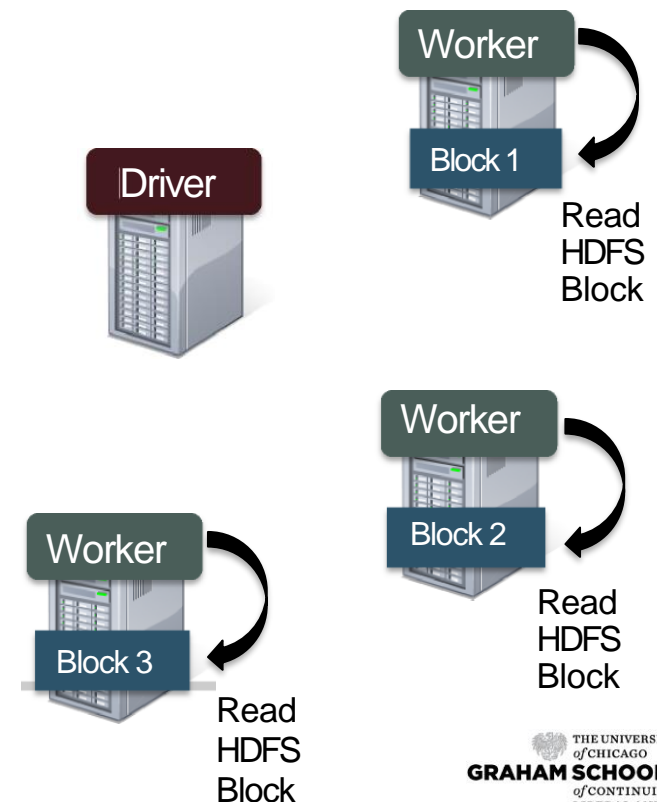
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
```



EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

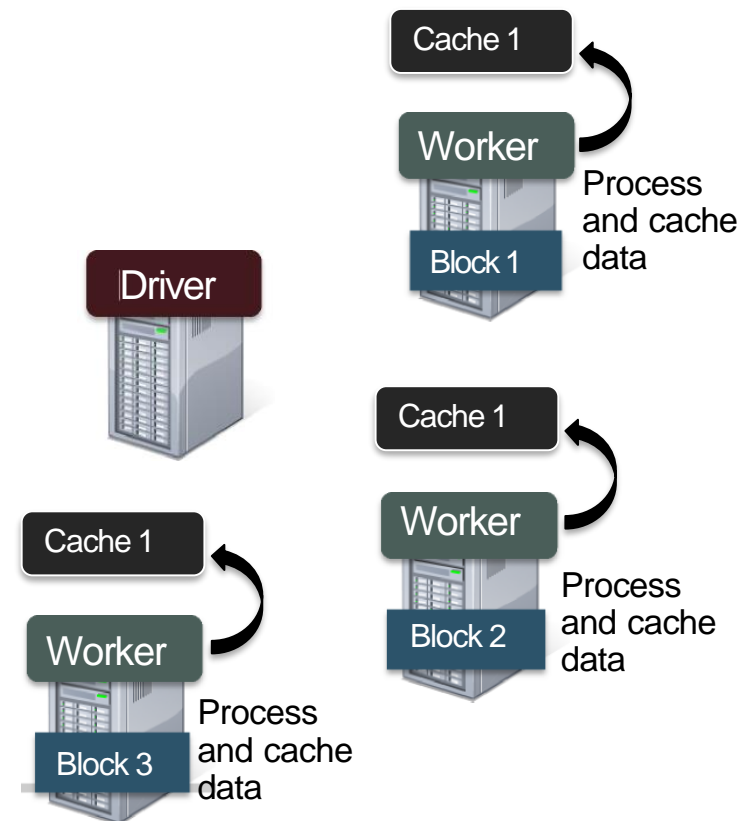
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
```



EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

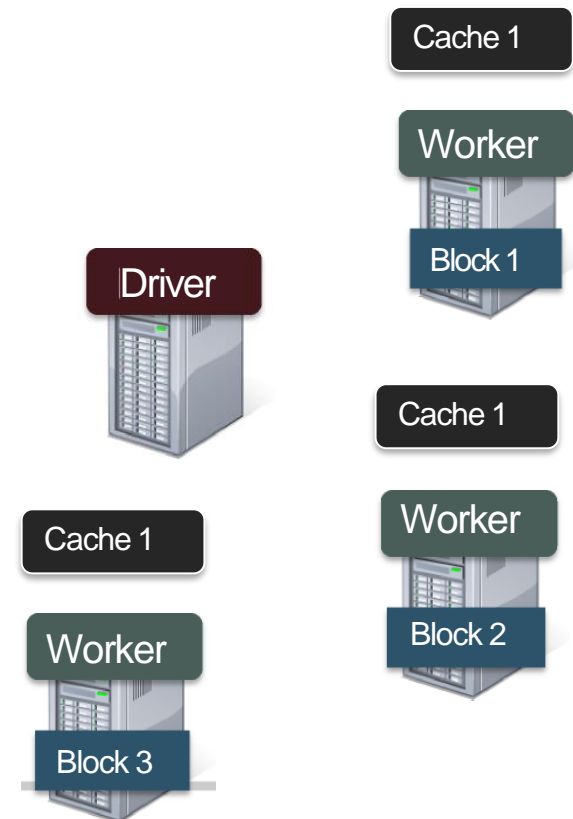
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
```



EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

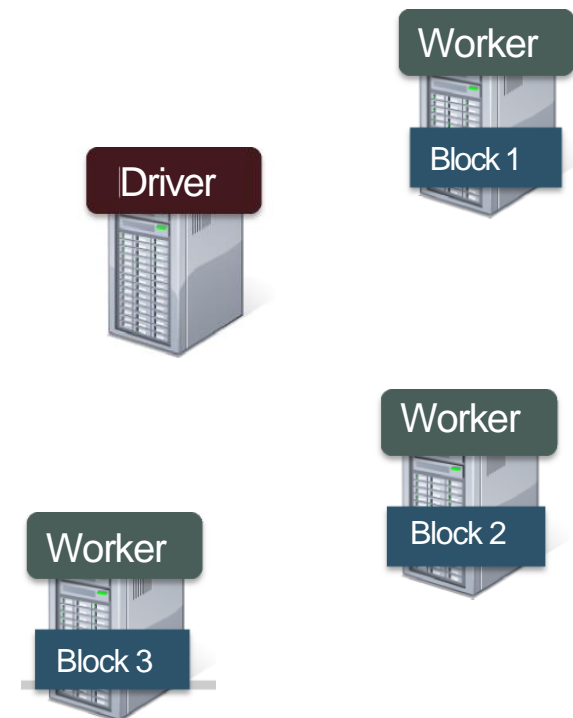
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
```



EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

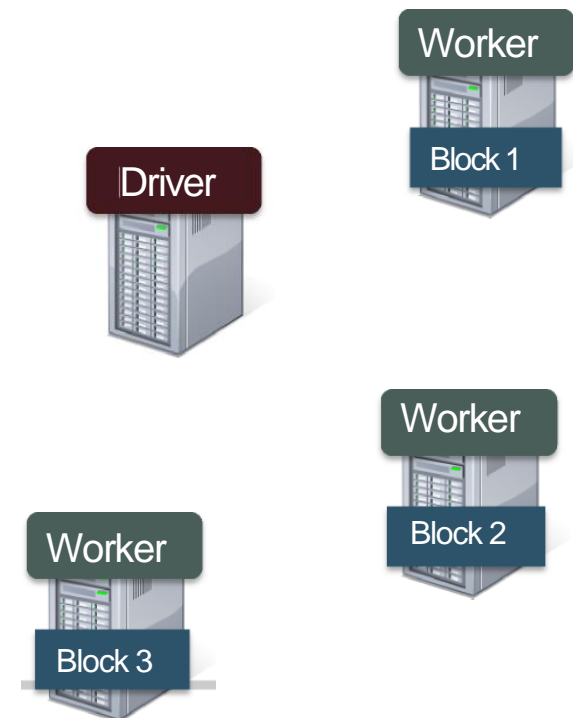
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```



EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

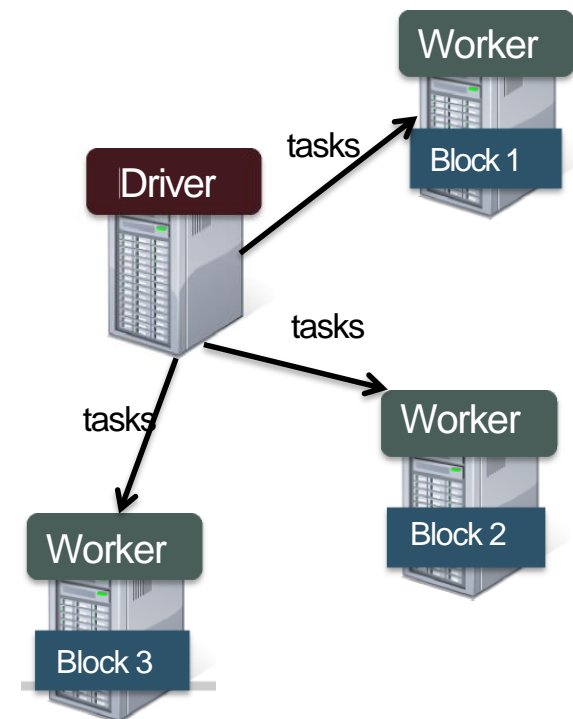
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```



EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

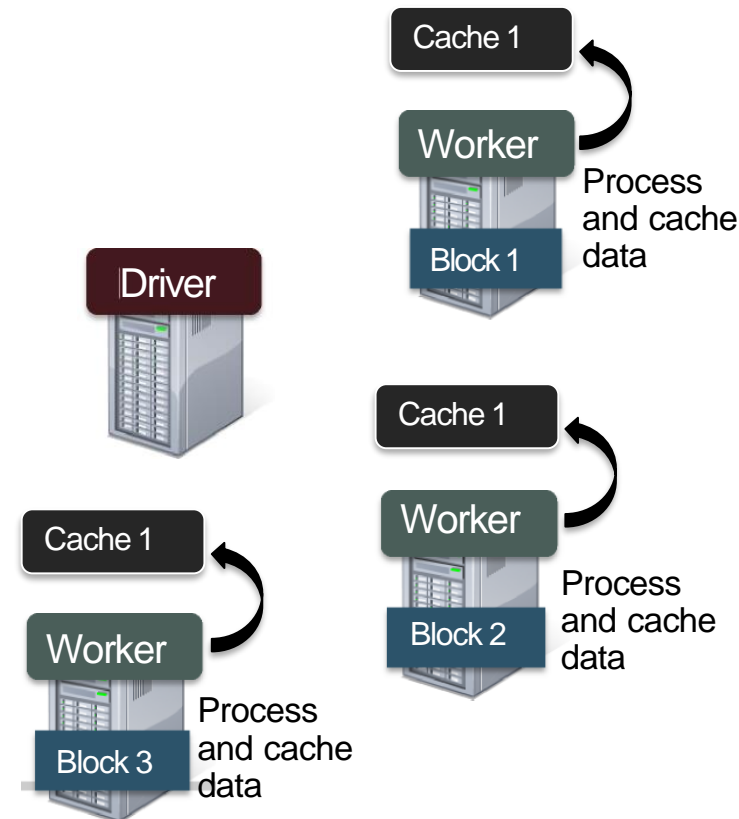
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```



EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

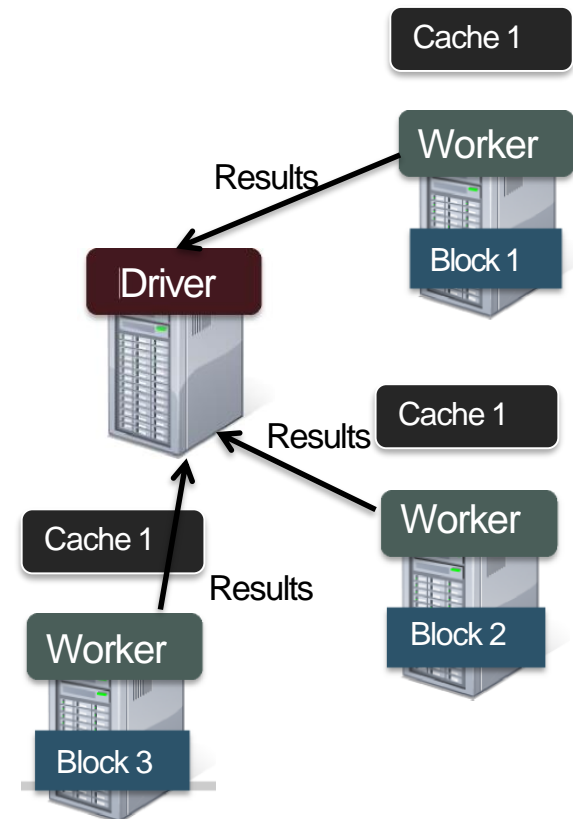
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```



EXAMPLE: LOG MINING

- Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```



EXAMPLE: LOG MINING

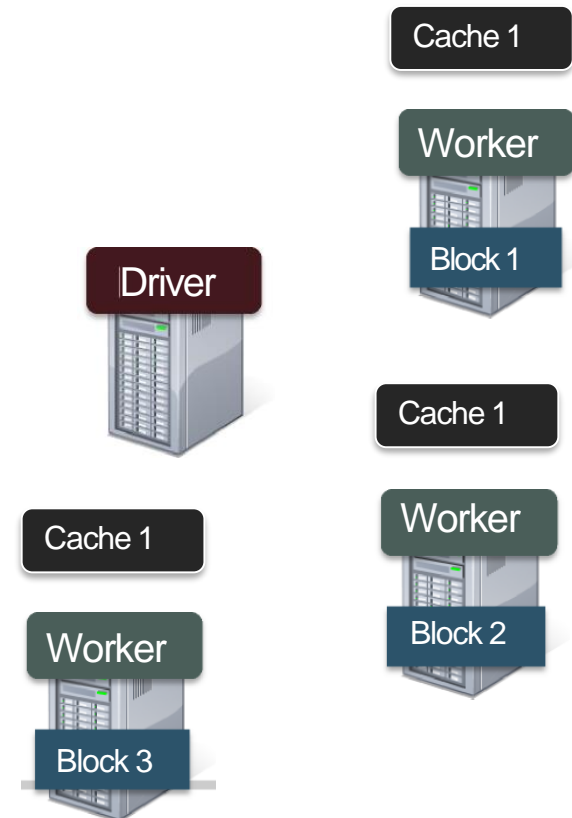
- Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

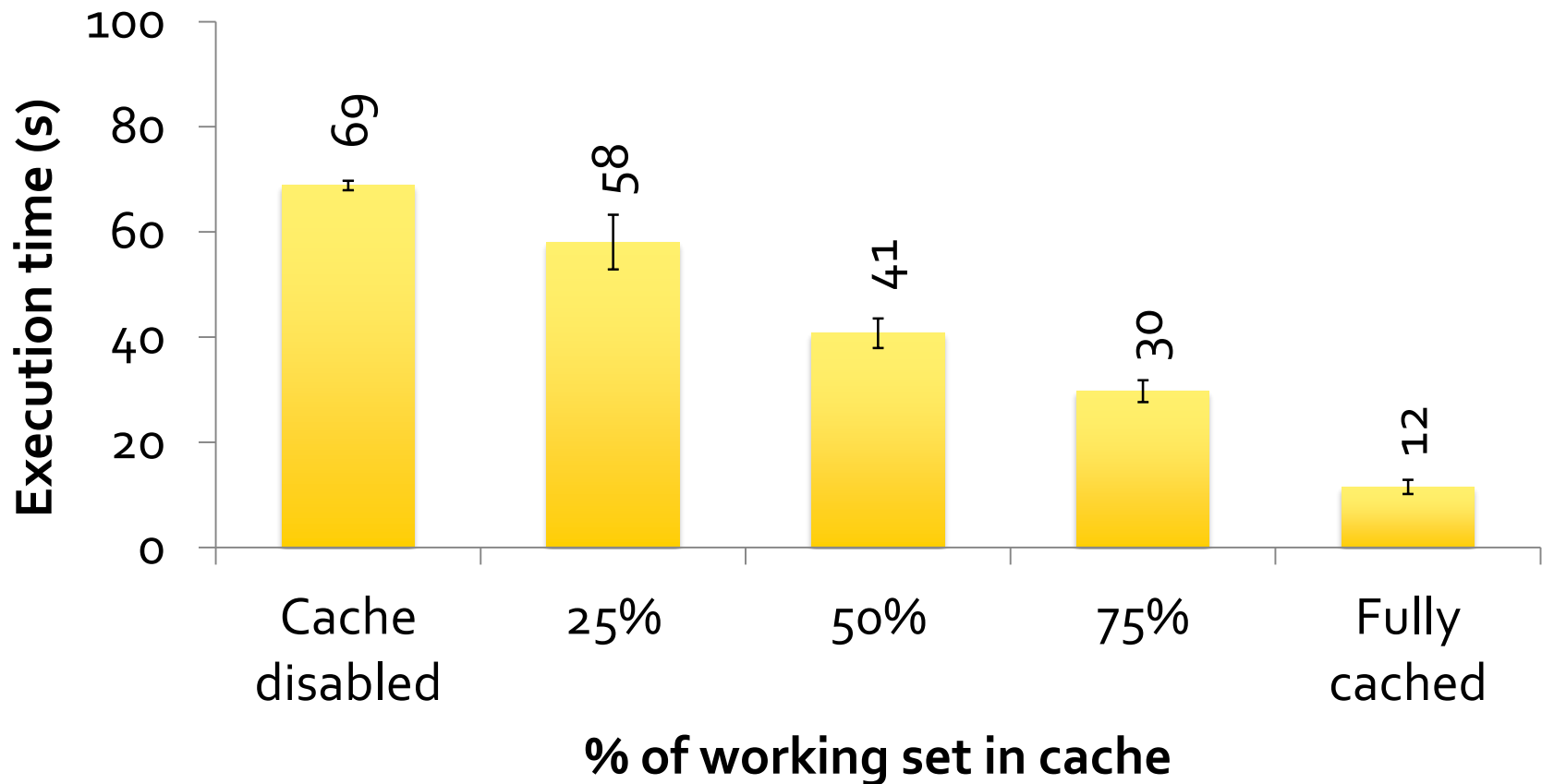
Cache your data. Faster Results

Full-text search of Wikipedia

- 60GB on 20 EC2 machines
- 0.5 sec from cache vs. 20s for on-disk



BEHAVIOR WITH LESS RAM



RDD OPERATIONS - TRANSFORMATIONS

- A subset of the transformations available. Full set can be found on Spark's website.
- Transformations are lazy evaluations
- Returns a pointer to the transformed RDD

Transformation	Meaning
map(func)	Return a new dataset formed by passing each element of the source through a function <i>func</i> .
filter(func)	Return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true.
flatMap(func)	Similar to map, but each input item can be mapped to 0 or more output items. So func should return a Seq rather than a single item
join(<i>otherDataset</i> , [<i>numTasks</i>])	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.
reduceByKey(func)	When called on a dataset of (K, V) pairs, returns a dataset of (K,V) pairs where the values for each key are aggregated using the given reduce function <i>func</i>
sortByKey([ascending],[numTasks])	When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K,V) pairs sorted by keys in ascending or descending order.

RDD OPERATIONS - ACTIONS

- Actions returns values

Action	Meaning
collect()	Return all the elements of the dataset as an array of the driver program. This is usually useful after a filter or another operation that returns a sufficiently small subset of data.
count()	Return the number of elements in a dataset.
first()	Return the first element of the dataset
take(n)	Return an array with the first n elements of the dataset.
foreach(func)	Run a function func on each element of the dataset.

RDD PERSISTENCE

- Each node stores any partitions of the cache that it computes in memory
- Reuses them in other actions on that dataset (or datasets derived from it)
 - Future actions are much faster (often by more than 10x)
- Two methods for RDD persistence
 - `persist()`
 - `cache()` → essentially just `persist` with `MEMORY_ONLY` storage

Storage Level	Meaning
MEMORY_ONLY	Store as deserialized Java objects in the JVM. If the RDD does not fit in memory, part of it will be cached. The other will be recomputed as needed. This is the default. The <code>cache()</code> method uses this.
MEMORY_AND_DISK	Same except also store on disk if it doesn't fit in memory. Read from memory and disk when needed.
MEMORY_ONLY_SER	Store as serialized Java objects (one byte array per partition). Space efficient, but more CPU intensive to read.
MEMORY_AND_DISK_SER	Similar to <code>MEMORY_AND_DISK</code> but stored as serialized objects.
DISK_ONLY	Store only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Same as above, but replicate each partition on two cluster nodes
OFF_HEAP (experimental)	Store RDD in serialized format in Tachyon.

SHARED VARIABLES AND KEY-VALUE PAIRS

- When a function is passed from the driver to a worker, normally a separate copy of the variables are used.
- Two types of variables:
 - Broadcast variables
 - Read-only copy on each machine
 - Distribute broadcast variables using efficient broadcast algorithms
 - Accumulators
 - Variables added through an associative operation
 - Implement counters and sums
 - Only the driver can read the accumulators value
 - Numeric types accumulators. Extend for new types.

Scala: key-value pairs

```
val pair = ('a', 'b')  
pair._1 // will return 'a'  
pair._2 // will return 'b'
```

Python: key-value pairs

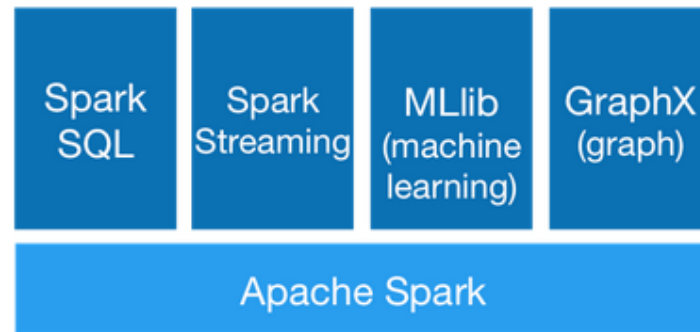
```
pair = ('a', 'b')  
pair[0] # will return 'a'  
pair[1] # will return 'b'
```

Java: key-value pairs

```
Tuple2 pair = new Tuple2('a', 'b');  
pair._1 // will return 'a'  
pair._2 // will return 'b'
```

SPARK LIBRARIES

- Extension of the core Spark API.
- Improvements made to the core are passed to these libraries.
- Little overhead to use with the Spark core



spark.apache.org

SPARK SQL AND SPARK DATA FRAME

- Allows relational queries expressed in
 - SQL
 - HiveQL
 - Scala
- SchemaRDD
 - Row objects
 - Schema
 - Created from:
 - Existing RDD
 - Parquet file
 - JSON dataset
 - HiveQL against Apache Hive
- Supports Scala, Java, and Python



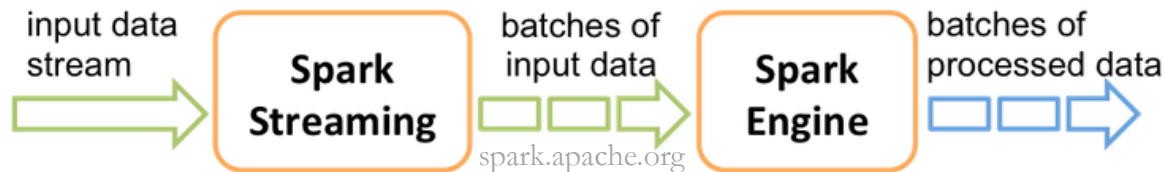
SPARK STREAMING

- Scalable, high-throughput, fault-tolerant stream processing of live data streams
 - Receives live input data and divides into small batches which are processed and returned as batches
 - DStream – sequence of RDD
 - Currently supports Scala and Java
 - Basic Python support available in Spark 1.2.
- Receives data from:
 - Kafka
 - Flume
 - HDFS / S3
 - Kinesis
 - Twitter
 - Pushes data out to:
 - HDFS
 - Databases
 - Dashboard

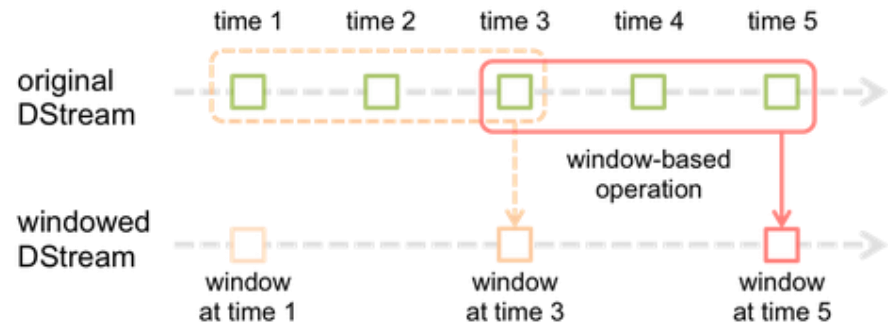


SPARK STREAMING - INTERNALS

- The input stream (DStream) goes into Spark Streaming
- Breaks up into batches
- Feeds into the Spark engine for processing
- Generate the final results in streams of batches



- Sliding window operations
 - Windowed computations
 - Window length
 - Sliding interval
 - `reduceByKeyAndWindow`



spark.apache.org

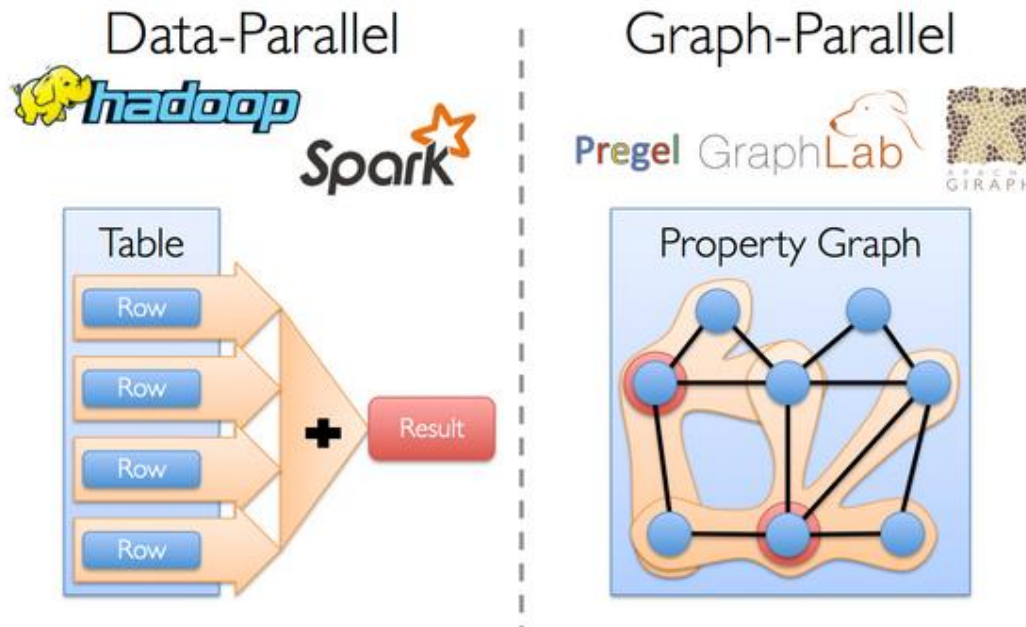
MLLIB



- ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering
- Featurization: feature extraction, transformation, dimensionality reduction, and selection
- Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
- Persistence: saving and load algorithms, models, and Pipelines
- Utilities: linear algebra, statistics, data handling, etc.

GRAPHX

- GraphX for graph processing
 - Graphs and graph parallel computation
 - Social networks and language modeling



<https://spark.apache.org/docs/latest/graphx-programming-guide.html#overview>

WORKING WITH SPARK

BASIC TRANSFORMATIONS

- `nums = sc.parallelize([1, 2, 3])`
- `# Pass each element through a function`
- `squares = nums.map(lambda x: x*x) // {1, 4, 9}`
- `# Keep elements passing a predicate`
- `even = squares.filter(lambda x: x % 2 == 0) // {4}`
- `# Map each element to zero or more others`
- `nums.flatMap(lambda x: => range(x))`
`# => {0, 0, 1, 0, 1, 2}`

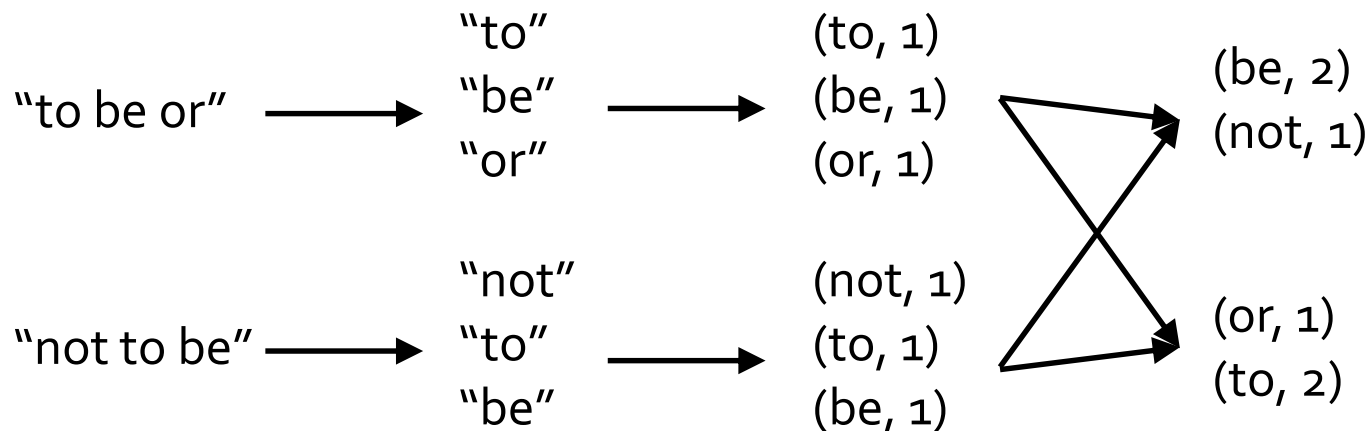
Range object (sequence of numbers 0, 1, ..., x-1)

BASIC ACTIONS

- `nums = sc.parallelize([1, 2, 3])`
- `# Retrieve RDD contents as a local collection`
- `nums.collect() # => [1, 2, 3]`
- `# Return first K elements`
- `nums.take(2) # => [1, 2]`
- `# Count number of elements`
- `nums.count() # => 3`
- `# Merge elements with an associative function`
`nums.reduce(lambda x, y: x + y) # => 6`
- `# Write elements to a text file`
`nums.saveAsTextFile("hdfs://file.txt")`

EXAMPLE: WORD COUNT

- `lines = sc.textFile("hamlet.txt")`
- `counts = lines.flatMap(lambda line: line.split(" "))`
 `.map(lambda word => (word, 1))`
 `.reduceByKey(lambda x, y: x + y)`



SETTING THE LEVEL OF PARALLELISM

- All the pair RDD operations take an optional second parameter for number of tasks
- `words.reduceByKey(lambda x, y: x + y, 5)`
- `words.groupByKey(5)`
- `visits.join(pageViews, 5)`

OTHER RDD OPERATORS

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin
- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip
- sample
- take
- first
- partitionBy
- mapWith
- pipe
- save
- ...

More details: spark-project.org/docs/latest/

CONCLUSION

- Spark offers a rich API to make data analytics fast: both fast to write and fast to run
- Achieves 100x speedups in real applications
- Growing community with 20+ companies contributing



www.spark-project.org

RUNNING PYSPARK FROM RCC

From JupyterHub:

- <https://hadoop.rcc.uchicago.edu/hub/login>

From shell:

- `pyspark2 --master yarn`

SPARK-SUBMIT

From command line, you need to set:

- `export PATH=/software/Anaconda3-5.1.0-hadoop/bin:$PATH`
`export PYSPARK_PYTHON=/software/Anaconda3-5.1.0-hadoop/bin/python`
`export PYSPARK_DRIVER_PYTHON=/software/Anaconda3-5.1.0-hadoop/bin/python`

Or to export them into environment variable, say `env.sh`, and then source it:

- `source env.sh`
 - Before executing “`spark2-submit`”

To submit in batch:

- `spark2-submit --master yarn spark_code_1.py`
- `nohup spark2-submit --master yarn spark_code_1.py 1>output.out 2>error.log&`

To see the list of running jobs:

- `yarn application -list`

SPARK-SUBMIT

pyspark makes some housekeeping for you: in particular you get sc automatically. When using batch, you need to add a few lines to your code to set up context:

```
=====
```

```
from pyspark import SparkConf, SparkContext  
from pyspark.sql import SQLContext
```

```
sc = SparkContext("yarn-client","my_code")  
sqlCtx = SQLContext(sc)
```

```
=====
```

THANK YOU!