Parallel Programming with Spark

Dr. Rajiv Misra
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss:
  - Overview of Spark
  - Fundamentals of Scala & functional programming
  - Spark concepts
  - Spark operations
  - Job execution

Introduction to Spark

What is Spark?

- Fast, expressive cluster computing system compatible with Apache Hadoop
  - Works with any Hadoop-supported storage system (HDFS, S3,SequenceFile, Avro, …)
- Improves **efficiency** through:
  - In-memory computing primitives
  - General computation graphs → **Up to 100× faster**
- Improves **usability** through:
  - Rich APIs in Java, Scala, Python
  - Interactive shell → **Often 2-10× less code**

How to Run It

- Local multicore: just a library in your program
- EC2: scripts for launching a Spark cluster
- Private cluster: Mesos, YARN, Standalone Mode

Scala vs Java APIs

- Spark originally written in Scala, which allows concise function syntax and interactive use
- APIs in Java, Scala and Python
- Interactive shells in Scala and Python

Introduction to Scala & functional programming

About Scala

- **High-level language for the Java VM**
  - Object-oriented + functional programming
- **Statically typed**
  - Comparable in speed to Java
  - But often no need to write types due to type inference
- **Interoperates with Java**
  - Can use any Java class, inherit from it, etc; can also call Scala code from Java

Best Way to Learn Scala

- Interactive shell: just type scala

- Supports importing libraries, tab completion and all constructs in the language.

Quick Tour

Declaring variables:

```
var x: Int = 7
var x = 7  // type inferred
val y = "hi"  // read-only
```

Java equivalent:

```
int x = 7;

final String y = "hi";
```

Functions:

```
def square(x: Int): Int = x*x

def square(x: Int): Int = {
  x*x
}

def announce(text: String) {
  println(text)
}
```

Last expression in block returned

Java equivalent:

```
int square(int x) {
  return x*x;
}

void announce(String text) {
  System.out.println(text);
}
```

Quick Tour

Generic types:

```
var arr = new Array[Int](8)

var lst = List(1, 2, 3)
// type of lst is List[Int]
```

Factory method

Java equivalent:

```
int[] arr = new int[8];
List<Integer> lst =
  new ArrayList<Integer>();
lst.add(...)
```

Can't hold primitive types

Indexing:

```
arr(5) = 7

println(lst(5))
```

Java equivalent:

```
arr[5] = 7;

System.out.println(lst.get(5));
```

Quick Tour

Processing collections with functional programming:

```
val list = List(1, 2, 3)
list.foreach(x => println(x))  // prints 1, 2, 3
list.foreach(println)          // same
list.map(x => x + 2)  // => List(3, 4, 5)
list.map(_ + 2)       // same, with placeholder notation
list.filter(x => x % 2 == 1)  // => List(1, 3)
list.filter(_ % 2 == 1)       // => List(1, 3)
list.reduce((x, y) => x + y)  // => 6
list.reduce(_ + _)            // => 6
```

Function expression (closure)

All of these leave the list unchanged (List is immutable)

Scala Closure Syntax

```
(x: Int) => x + 2   // full version
x => x + 2          // type inferred
_ + 2               // when each argument is used exactly once
x => {              // when body is a block of code
  val numberToAdd = 2
  x + numberToAdd
}

// If closure is too long, can always pass a function
def addTwo(x: Int): Int = x + 2

list.map(addTwo)
```

Scala allows defining a "local function" inside another function

Other Collection Methods

- Scala collections provide many other functional methods; for example, Google for "Scala Seq"

Method on Seq[T]	Explanation
`map(f: T => U): Seq[U]`	Pass each element through f
`flatMap(f: T => Seq[U]): Seq[U]`	One-to-many map
`filter(f: T => Boolean): Seq[T]`	Keep elements passing f
`exists(f: T => Boolean): Boolean`	True if one element passes
`forall(f: T => Boolean): Boolean`	True if all elements pass
`reduce(f: (T, T) => T): T`	Merge elements using f
`groupBy(f: T => K): Map[K,List[T]]`	Group elements by f(element)
`sortBy(f: T => K): Seq[T]`	Sort elements by f(element)
`. . .`	

Spark Concepts

Spark Overview

- **Goal: Work with distributed collections as you would with local ones**

- Concept: resilient distributed datasets (RDDs)
  - Immutable 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)

Main Primitives

Resilient distributed datasets (RDDs)

- Immutable, partitioned collections of objects

- Transformations (e.g. map, filter, groupBy, join)
  - Lazy operations to build RDDs from other RDDs

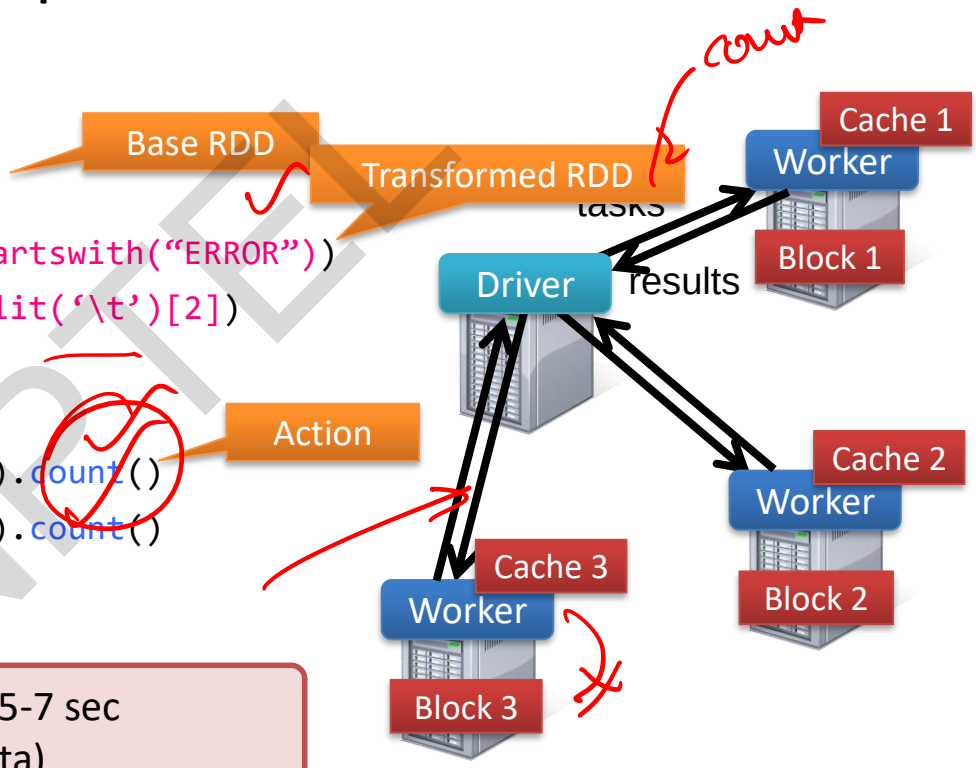
Actions (e.g. count, collect, save)

- Return a result or write it to storage

Example: Mining Console Logs

- Load error messages from a log into memory, then interactively search for 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: “foo” in s).count()
messages.filter(lambda s: “bar” in s).count()
. . .
```



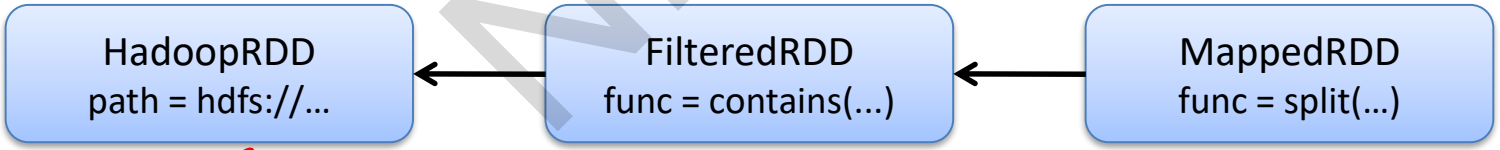
Result: scaled to 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)

RDD Fault Tolerance

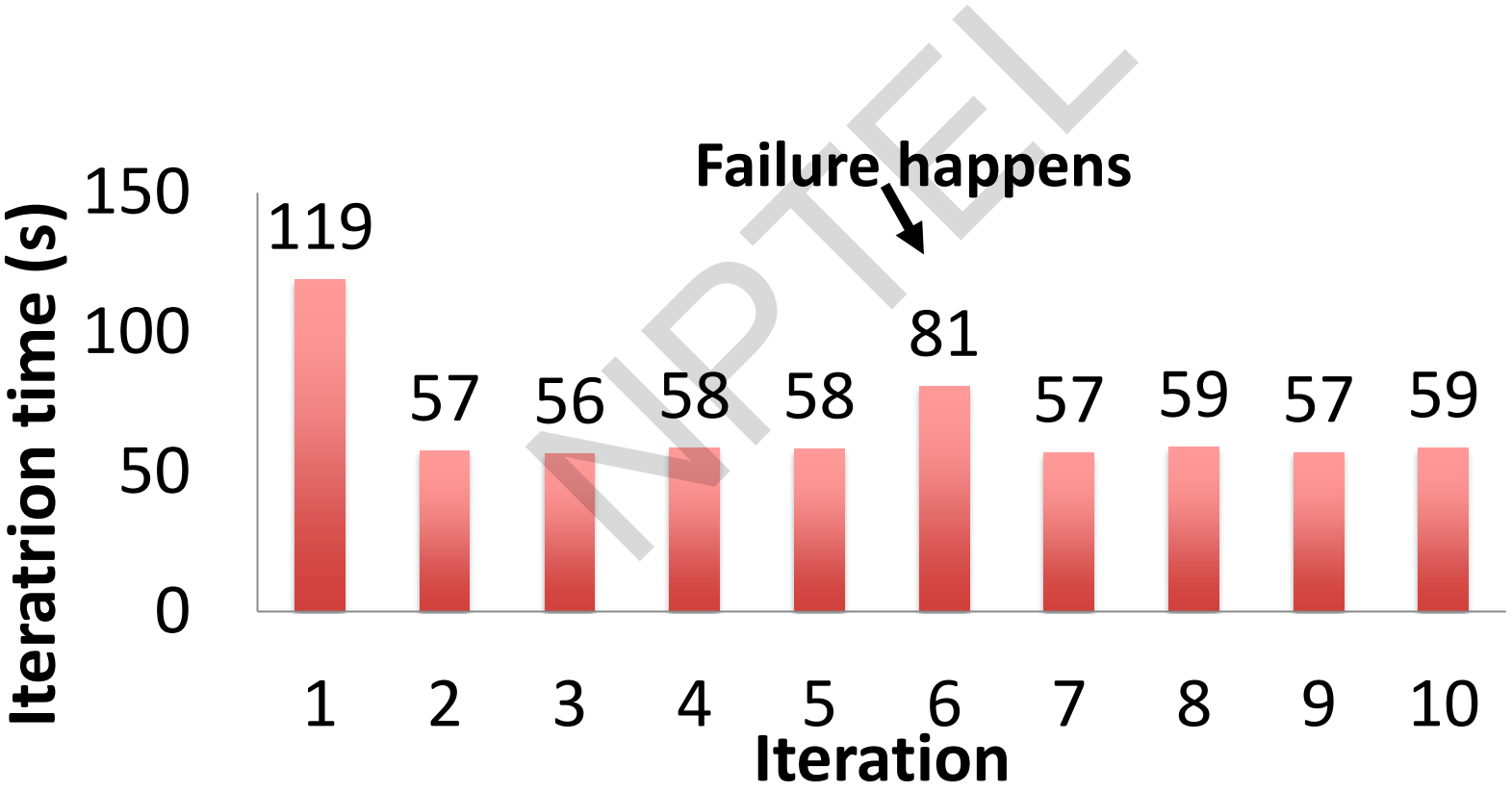
RDDs track the transformations used to build them (their *lineage*) to recompute lost data

E.g:

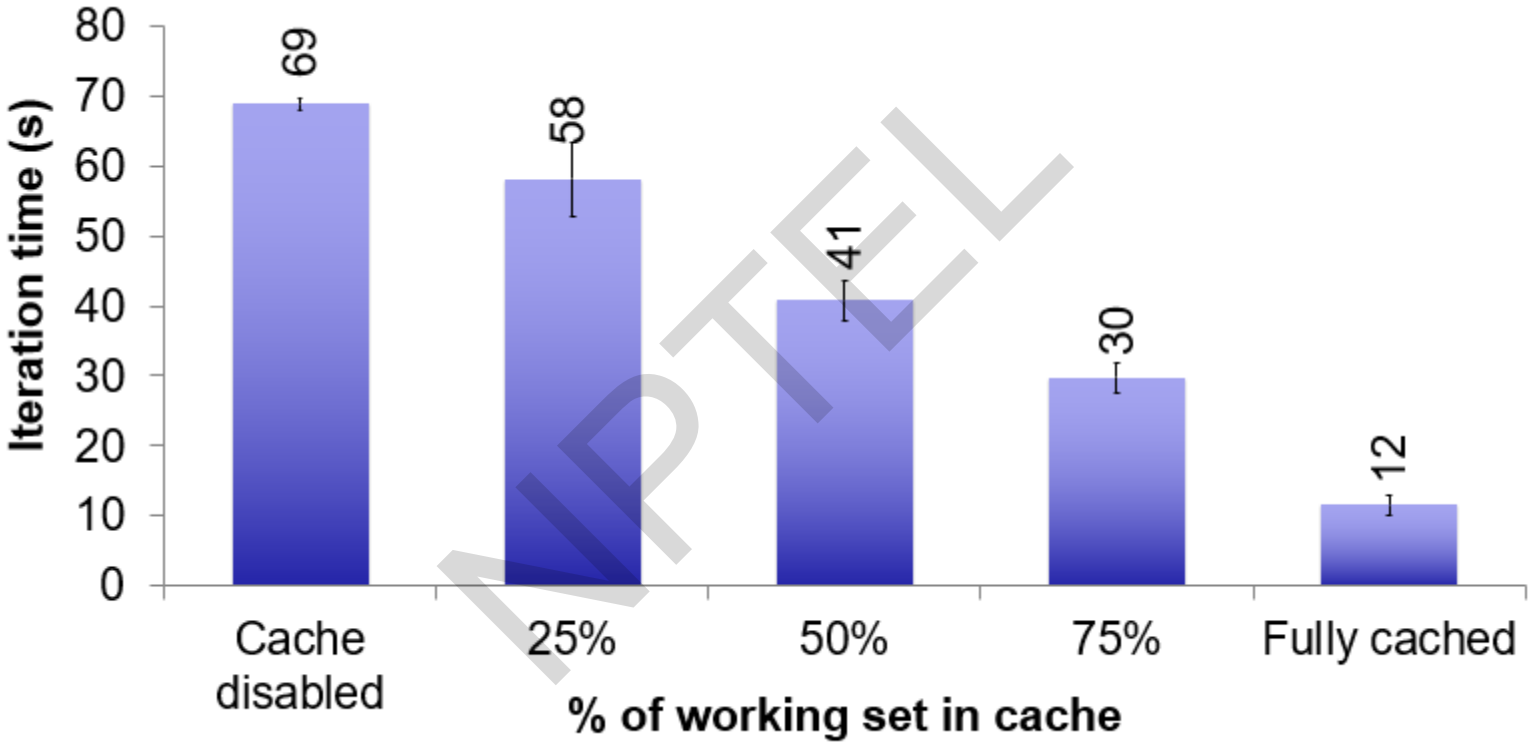
```
messages = textFile(...).filter(lambda s: s.contains("ERROR"))
                        .map(lambda s: s.split('\t')[2])
```



Fault Recovery Test



Behavior with Less RAM



Which Language Should I Use?

- Standalone programs can be written in any, but console is only Python & Scala
- **Python developers:** can stay with Python for both
- **Java developers:** consider using Scala for console (to learn the API)

- Performance: Java / Scala will be faster (statically typed), but Python can do well for numerical work with NumPy

Tour of Spark operations

Learning Spark

- Easiest way: Spark interpreter (`spark-shell` or `pyspark`)
  - Special Scala and Python consoles for cluster use
- Runs in local mode on 1 thread by default, but can control with `MASTER` environment var:

```
MASTER=local      ./spark-shell           # local, 1 thread
MASTER=local[2]   ./spark-shell           # local, 2 threads
MASTER=spark://host:port ./spark-shell    # Spark standalone
cluster
```

First Stop: SparkContext

- Main entry point to Spark functionality
- Created for you in Spark shells as variable sc
- In standalone programs, you'd make your own (see later for details)

Creating RDDs

```
# Turn a local collection into an RDD
sc.parallelize([1, 2, 3])

# Load text file from local FS, HDFS, or S3
sc.textFile("file.txt")
sc.textFile("directory/*.txt")
sc.textFile("hdfs://namenode:9000/path/file")

# Use any existing Hadoop InputFormat
sc.hadoopFile(keyClass, valClass, inputFmt,
conf)
```

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(0, 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")
```

Working with Key-Value Pairs

- Spark's "distributed reduce" transformations act on RDDs of *key-value pairs*
- Python:

```
pair = (a, b)
pair[0] # => a
pair[1] # => b
```

- Scala:

```
val pair = (a, b)
pair._1 // => a
pair._2 // => b
```

- Java:

```
Tuple2 pair = new Tuple2(a, b);
// class scala.Tuple2
pair._1 // => a
pair._2 // => b
```

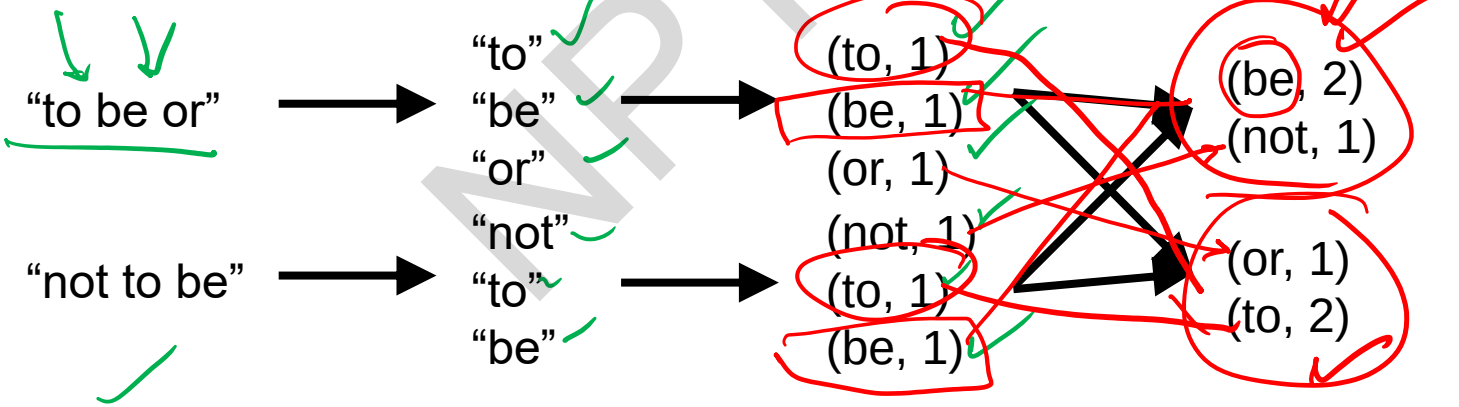
Some Key-Value Operations

```
pets = sc.parallelize([(“cat”, 1), (“dog”, 1), (“cat”, 2)])
pets.reduceByKey(lambda x, y: x + y)
# => {(cat, 3), (dog, 1)}
pets.groupByKey()
# => {(cat, Seq(1, 2)), (dog, Seq(1)}
pets.sortByKey()
# => {(cat, 1), (cat, 2), (dog, 1)}
```


reduceByKey also automatically implements combiners on the map side

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)
```



Other Key-Value Operations

- ```
  val visits = sc.parallelize(List(
    (“index.html”, “1.2.3.4”),
    (“about.html”, “3.4.5.6”),
    (“index.html”, “1.3.3.1”)))
  ```
- ```
  val pageNames = sc.parallelize(List(
    (“index.html”, “Home”), (“about.html”, “About”)))
  ```
- ```
  visits.join(pageNames)
  // (“index.html”, (“1.2.3.4”, “Home”))
  // (“index.html”, (“1.3.3.1”, “Home”))
  // (“about.html”, (“3.4.5.6”, “About”))
  ```
- ```
  visits.cogroup(pageNames)
  // (“index.html”, (Seq(“1.2.3.4”, “1.3.3.1”),
  Seq(“Home”)))
  // (“about.html”, (Seq(“3.4.5.6”), Seq(“About”)))
  ```

Multiple Datasets

```
visits = sc.parallelize([("index.html", "1.2.3.4"),
                         ("about.html", "3.4.5.6"),
                         ("index.html", "1.3.3.1")])

pageNames = sc.parallelize([("index.html", "Home"),
("about.html", "About")])

visits.join(pageNames)
# ("index.html", ("1.2.3.4", "Home"))
# ("index.html", ("1.3.3.1", "Home"))
# ("about.html", ("3.4.5.6", "About"))

visits.cogroup(pageNames)
# ("index.html", (Seq("1.2.3.4", "1.3.3.1"), Seq("Home")))
# ("about.html", (Seq("3.4.5.6"), Seq("About")))
```

Controlling 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)
```

Can also set spark.default.parallelism property

Using Local Variables

- External variables you use in a closure will automatically be shipped to the cluster:

```
query = raw_input("Enter a query:")
pages.filter(lambda x:
x.startswith(query)).count()
```

- Some caveats:
  - Each task gets a new copy (updates aren't sent back)
  - Variable must be Serializable (Java/Scala) or Pickle-able (Python)
  - Don't use fields of an outer object (ships all of it!)

Closure Mishap Example

```
class MyCoolRddApp {
  val param = 3.14
  val log = new Log(...)
  ...

  def work(rdd: RDD[Int]) {
    rdd.map(x => x + param)
       .reduce(...)
  }
}
```

NotSerializableException: MyCoolRddApp (or Log)

How to get around it:

```
class MyCoolRddApp {
  ...

  def work(rdd: RDD[Int]) {
    val param_ = param
    rdd.map(x => x + param_)
       .reduce(...)
  }
}
```

References only local variable instead of `this.param`

Other RDD Operations

sample(): deterministically sample a subset

union(): merge two RDDs

cartesian(): cross product

pipe(): pass through external program

See Programming Guide for more:
www.spark-project.org/documentation.html

More Details

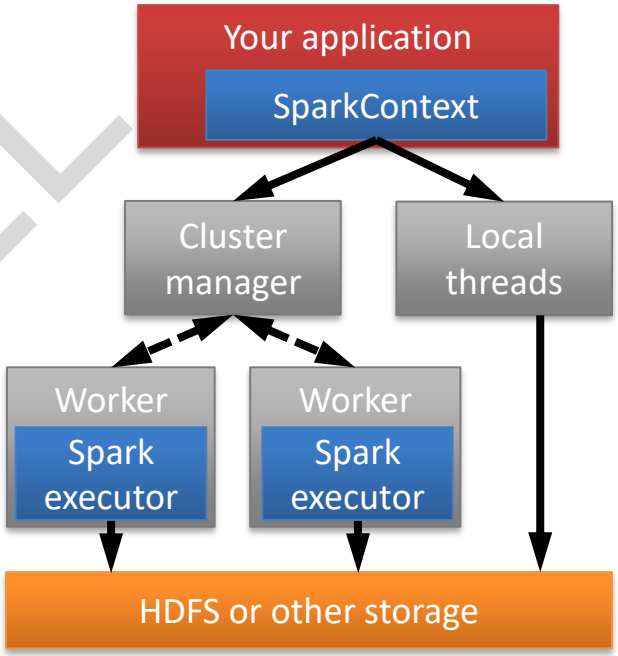
- Spark supports lots of other operations!

- Full programming guide: spark-project.org/documentation

Job execution

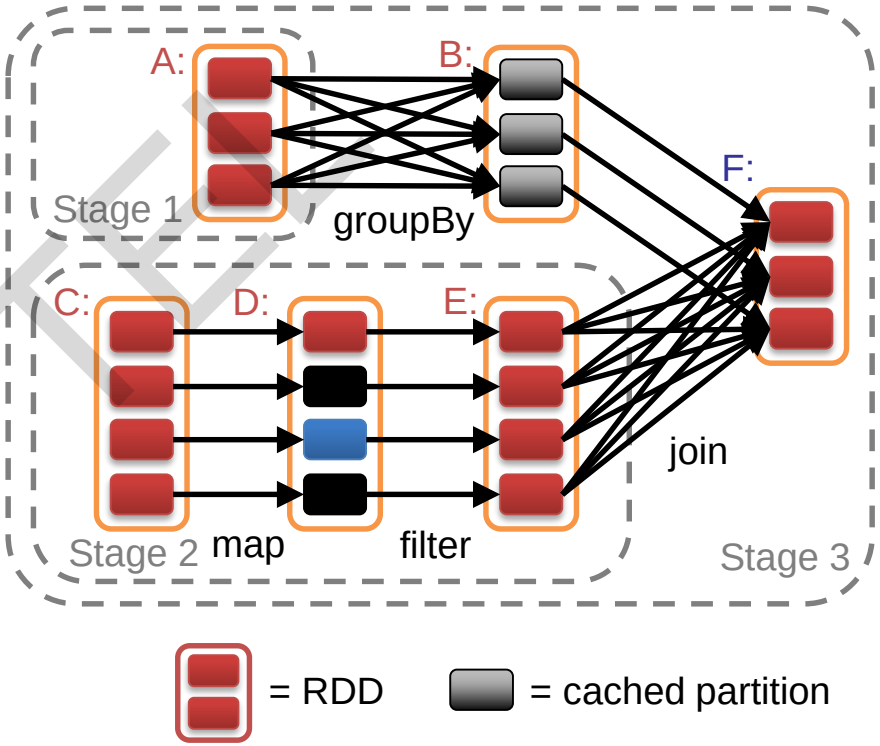
Software Components

- Spark runs as a library in your program
  (one instance per app)
- Runs tasks locally or on a cluster
  - Standalone deploy cluster, Mesos or YARN
- Accesses storage via Hadoop InputFormat API
  - Can use HBase, HDFS, S3, …



Task Scheduler

- Supports general task graphs
- Pipelines functions where possible
- Cache-aware data reuse & locality
- Partitioning-aware to avoid shuffles



More Information

- Scala resources:
  - www.artima.com/scalazine/articles/steps.html (First Steps to Scala)
  - www.artima.com/pins1ed (free book)
- Spark documentation: www.spark-project.org/documentation.html

Hadoop Compatibility

- Spark can read/write to any storage system / format that has a plugin for Hadoop!
  - Examples: HDFS, S3, HBase, Cassandra, Avro, SequenceFile
  - Reuses Hadoop's InputFormat and OutputFormat APIs

- APIs like SparkContext.textFile support filesystems, while SparkContext.hadoopRDD allows passing any Hadoop JobConf to configure an input source

Create a SparkContext

Scala

```
import spark.SparkContext
import spark.SparkContext._

val sc = new SparkContext("masterUrl", "name", "sparkHome", Seq("app.jar"))
```

Java

```
import spark.api.j...                                    ...xt

JavaSparkContext sc = new JavaSparkContext(
    "masterUrl", "name", "sparkHome", new String[] {"app.jar"}));
```

Python

```
from pyspark import SparkContext

sc = SparkContext("masterUrl", "name", "sparkHome", ["library.py"]))
```

Cluster URL, or local / local[N]

App name

Spark install path on cluster

List of JARs with app code (to ship)

Complete App: Scala

```
import spark.SparkContext
import spark.SparkContext._

object WordCount {
  def main(args: Array[String]) {
    val sc = new SparkContext("local", "WordCount", args(0), Seq(args(1)))
    val lines = sc.textFile(args(2))
    lines.flatMap(_.split(" "))
         .map(word => (word, 1))
         .reduceByKey(_ + _)
         .saveAsTextFile(args(3))
  }
}
```

Complete App: Python

```
import sys
from pyspark import SparkContext

if __name__ == "__main__":
    sc = SparkContext( “local”, “WordCount”, sys.argv[0], None)
    lines = sc.textFile(sys.argv[1])

    lines.flatMap(lambda s: s.split(“ ”)) \
         .map(lambda word: (word, 1)) \
         .reduceByKey(lambda x, y: x + y) \
         .saveAsTextFile(sys.argv[2])
```

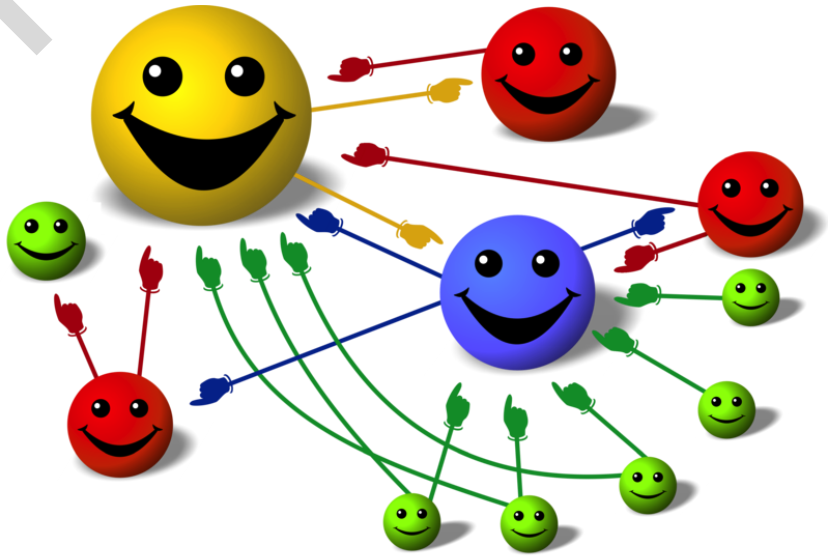
Example: PageRank

Why PageRank?

- Good example of a more complex algorithm
  - Multiple stages of map & reduce
- Benefits from Spark's in-memory caching
  - Multiple iterations over the same data

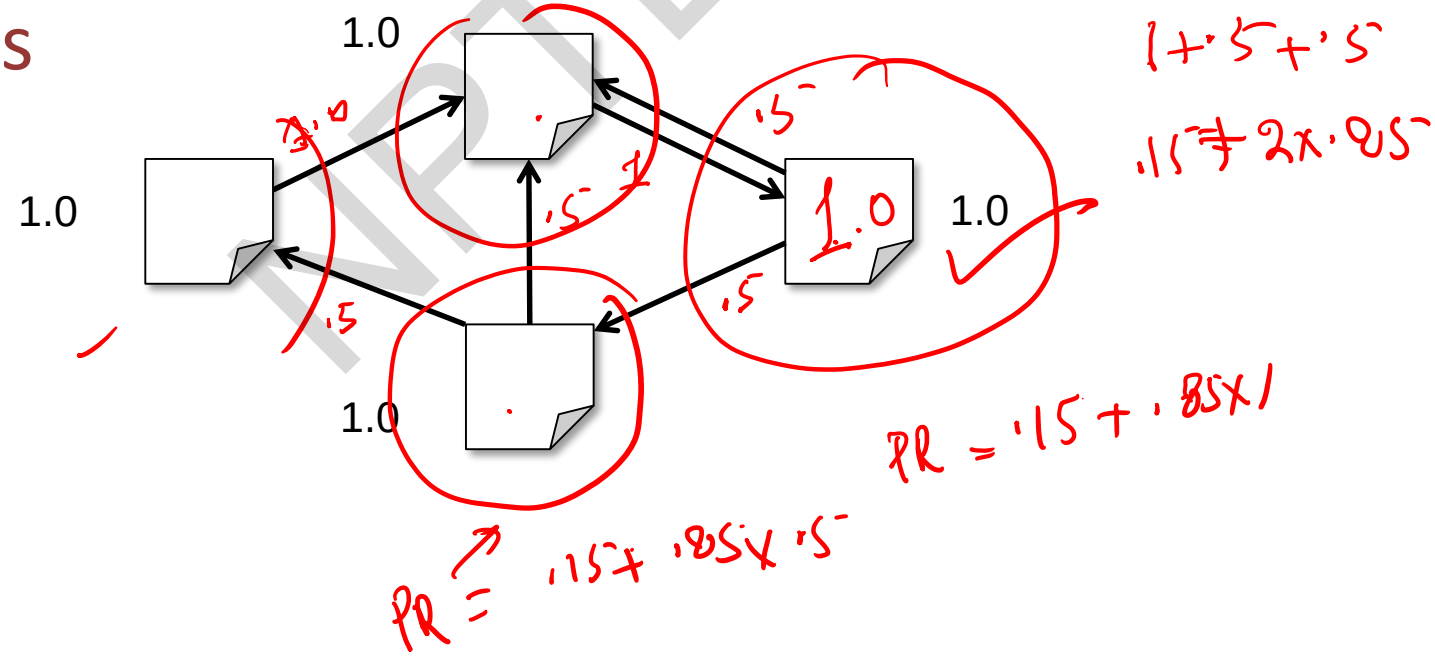
Basic Idea

- Give pages ranks (scores) based on links to them
  - Links from many pages → high rank
  - Link from a high-rank page → high rank



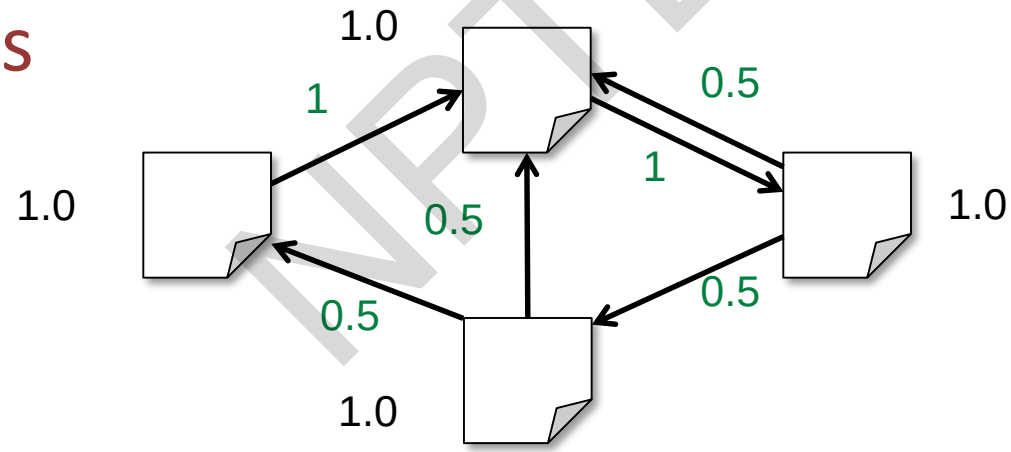
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



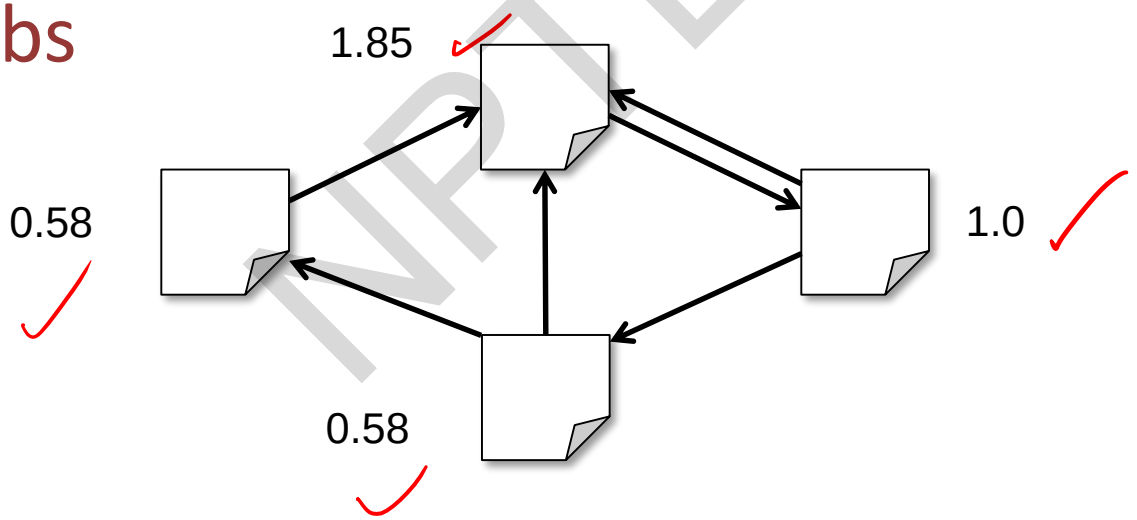
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



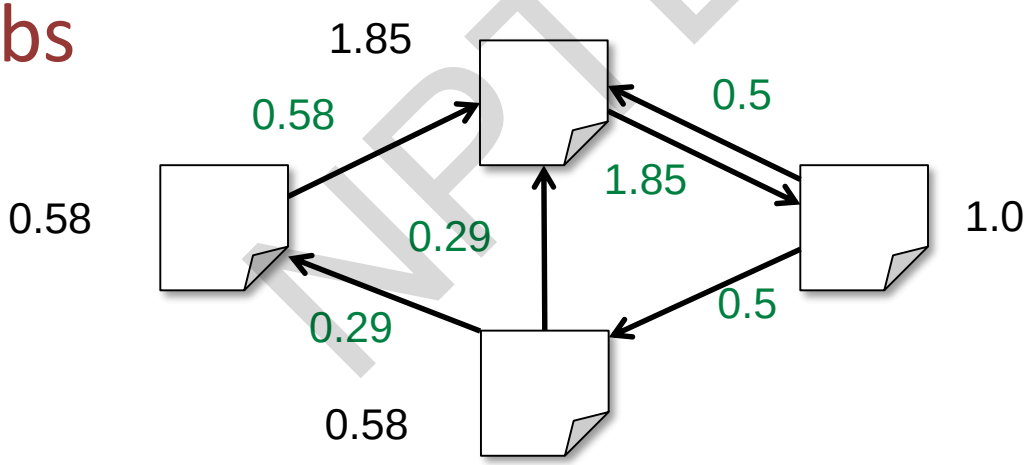
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



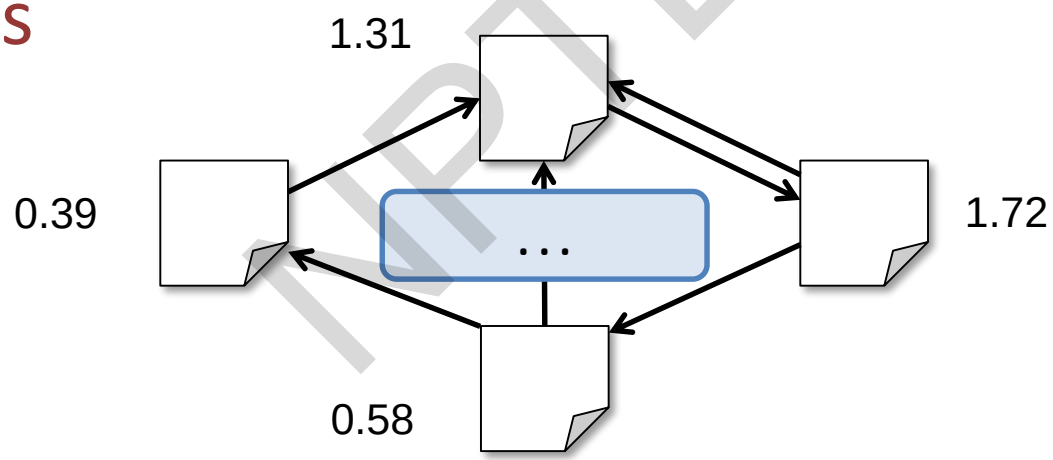
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



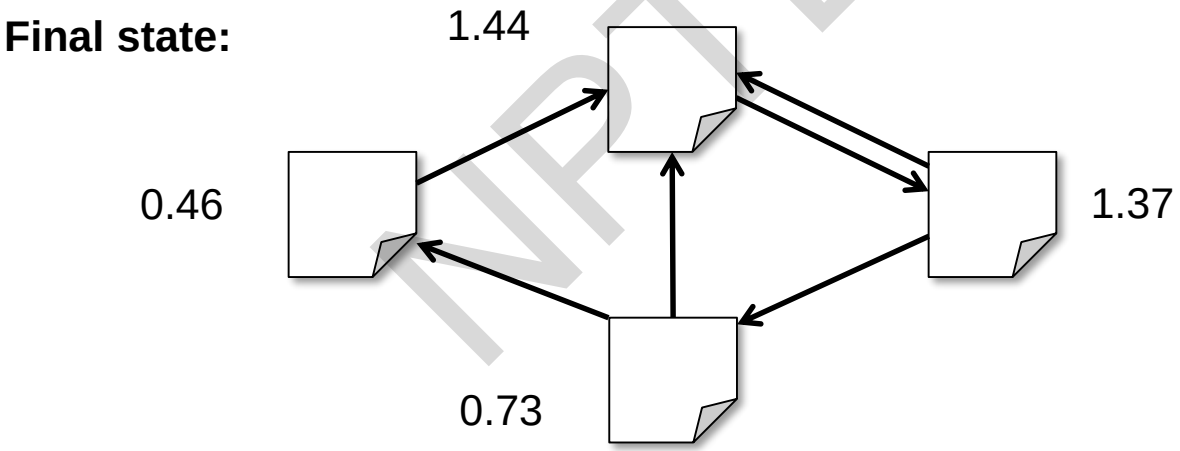
Algorithm

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



Algorithm

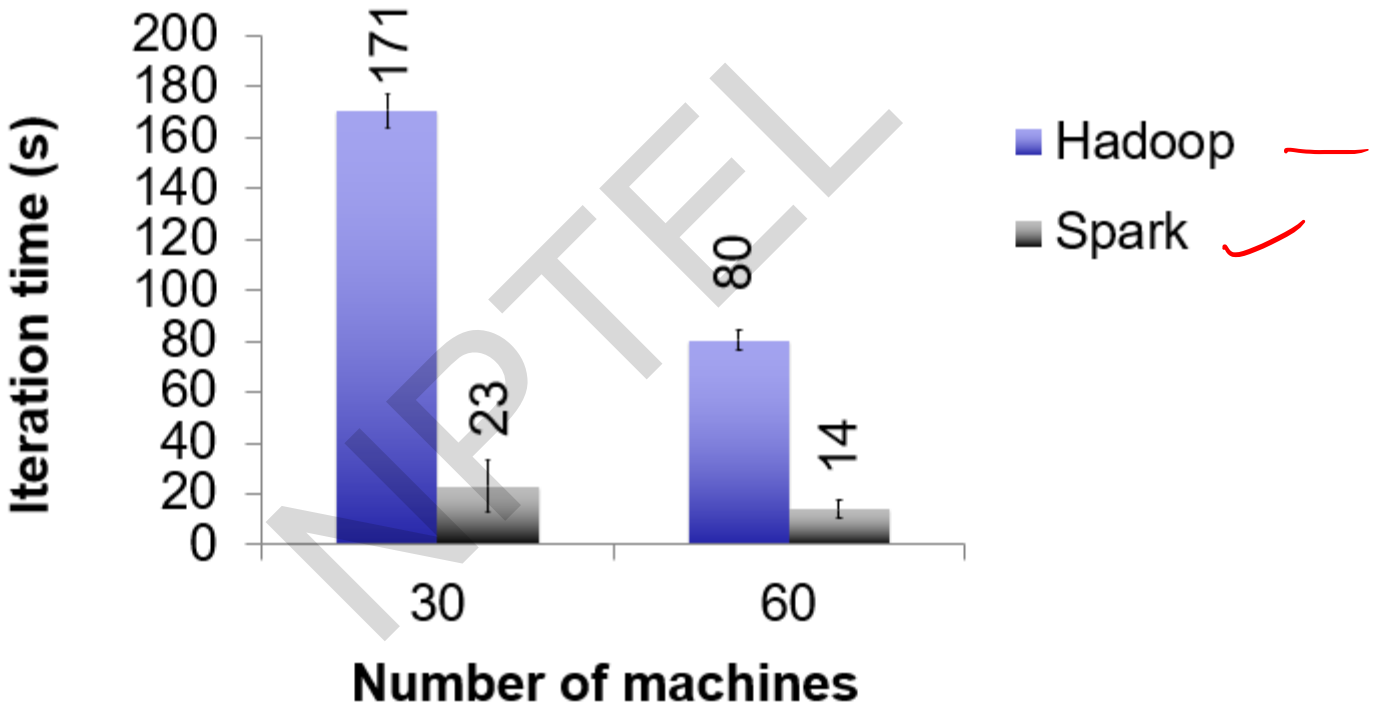
1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|neighbors_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



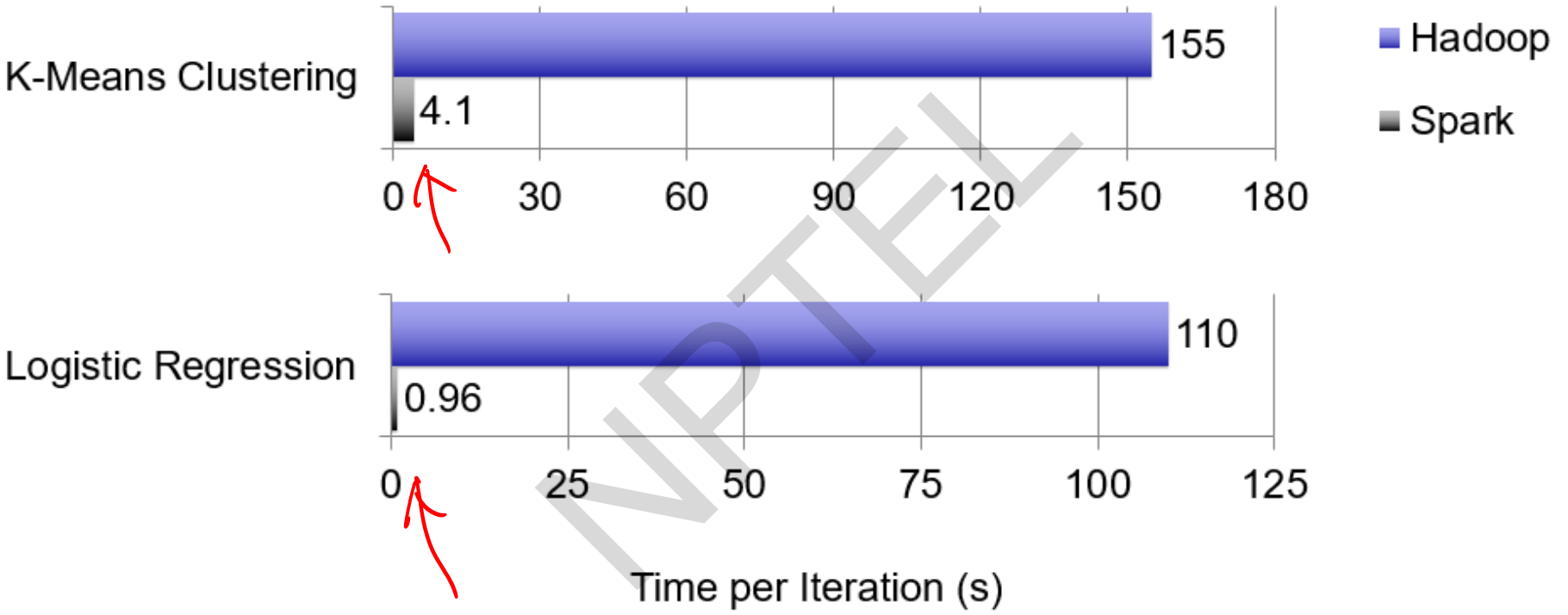
Scala Implementation

```
val links = // RDD of (url, neighbors) pairs
var ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  val contribs = links.join(ranks).flatMap {
    case (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }
  ranks = contribs.reduceByKey(_ + _)
                  .mapValues(0.15 + 0.85 * _)
}

ranks.saveAsTextFile(...)
```

PageRank Performance



Other Iterative Algorithms



References

- **Parallel Programming With Spark**
  Matei Zaharia, UC Berkeley

- **Parallel Programming with Spark**
  Qin Liu, The Chinese University of Hong Kong

- **http://spark.apache.org/**

- **https://amplab.cs.berkeley.edu/**

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 14 companies contributing
- Details, tutorials, videos: www.spark-project.org

Introduction to Spark

Dr. Rajiv Misra
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss the '**framework of spark**', Resilient Distributed Datasets (RDDs) and also discuss Spark execution.

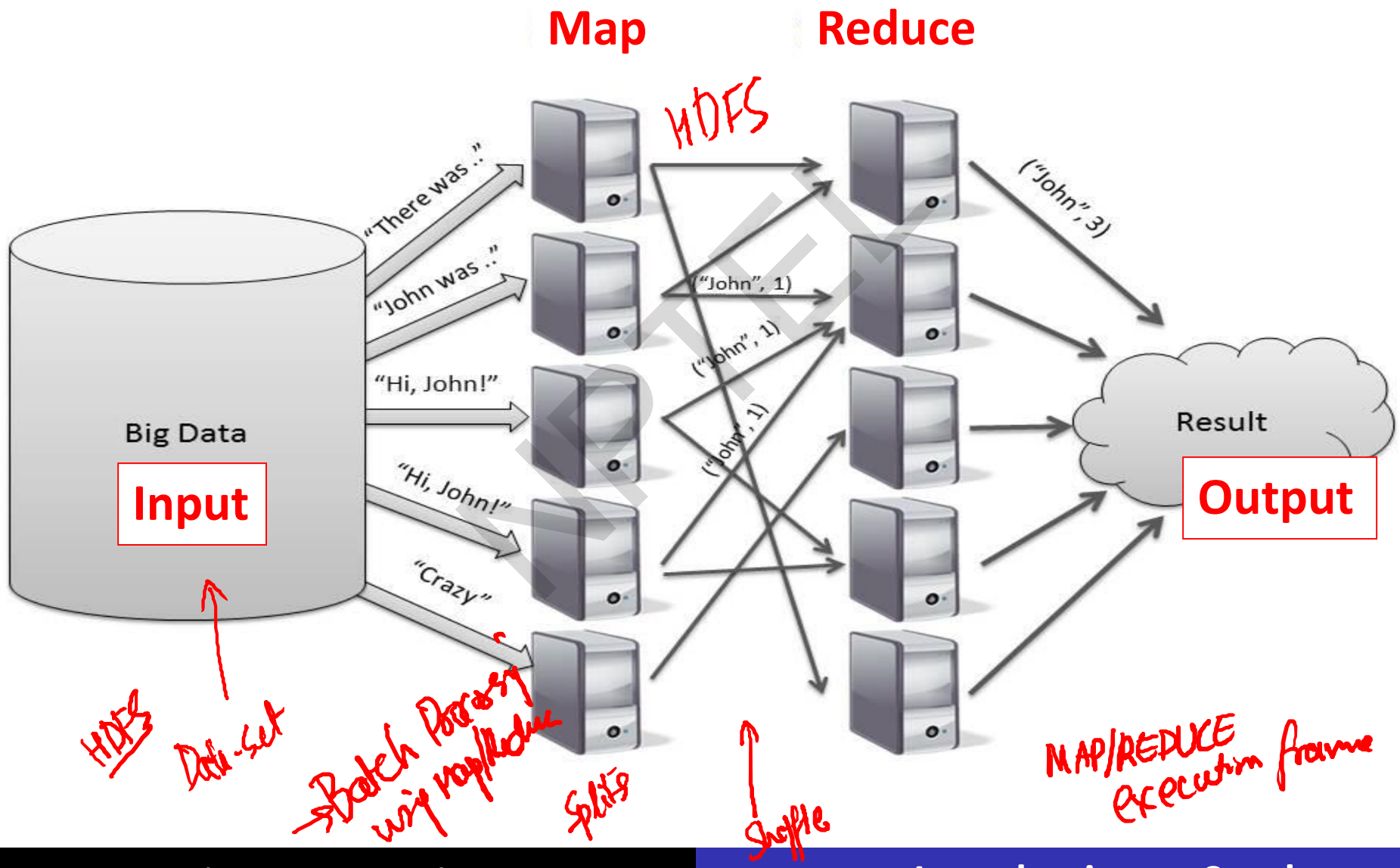
Need of Spark

- **Apache Spark** is a big data analytics framework that was originally developed at the University of California, Berkeley's AMPLab, in 2012. Since then, it has gained a lot of attraction both in academia and in industry.

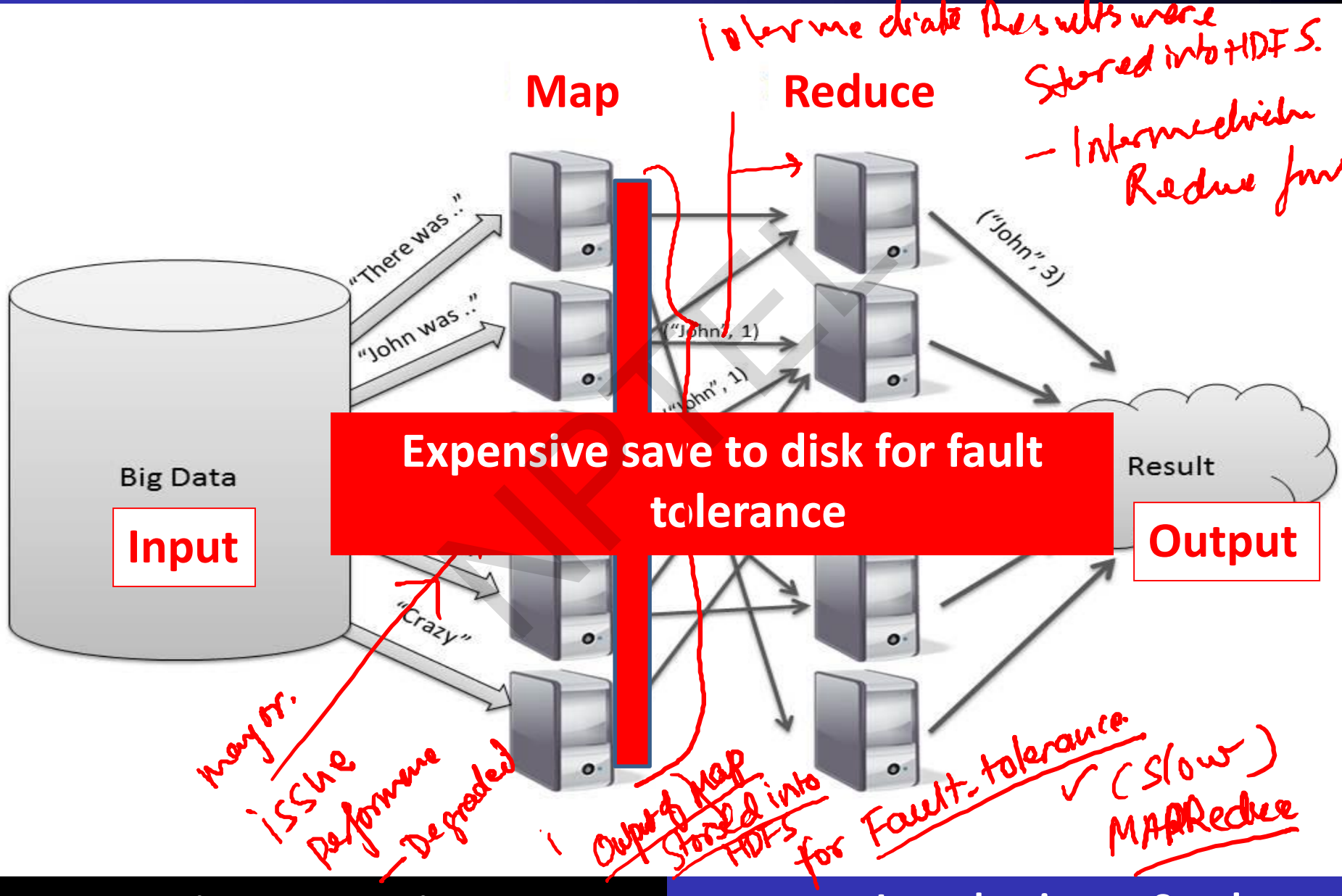
- It is an another system for big data analytics

- **Isn't MapReduce good enough?**
  - **Simplifies batch processing on large commodity clusters**

Need of Spark

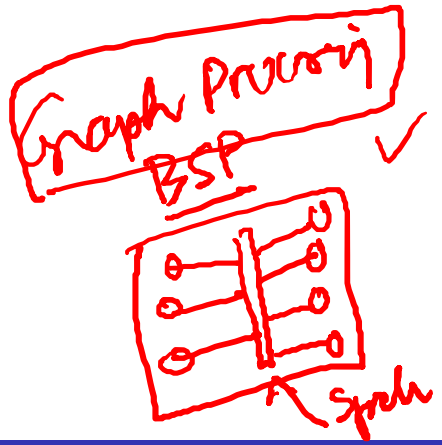


Need of Spark



Need of Spark

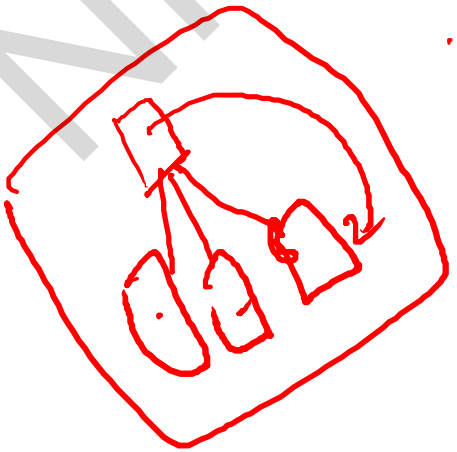
- MapReduce can be expensive for some applications e.g.,
  - **Iterative**
  - **Interactive**
- Lacks efficient data sharing
- Specialized frameworks did evolve for different programming models
  - **Bulk Synchronous Processing (Pregel)**
  - **Iterative MapReduce (Hadoop) ….**



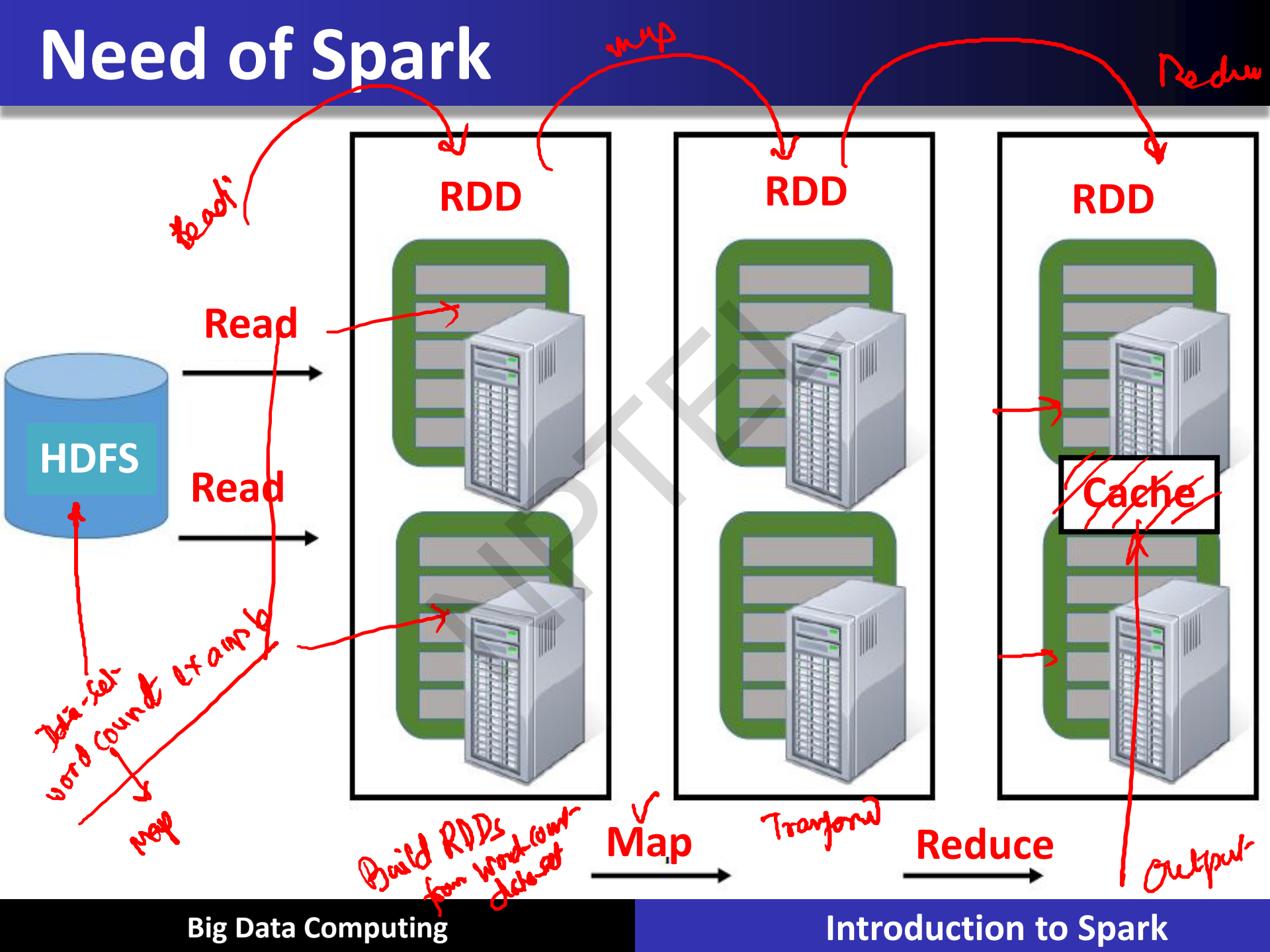
Solution: Resilient Distributed Datasets (RDDs)

Resilient Distributed Datasets (RDDs)

- Immutable, partitioned collection of records
- Built through coarse grained transformations (map, join …)
- Can be cached for efficient reuse



Need of Spark



Solution: Resilient Distributed Datasets (RDDs)

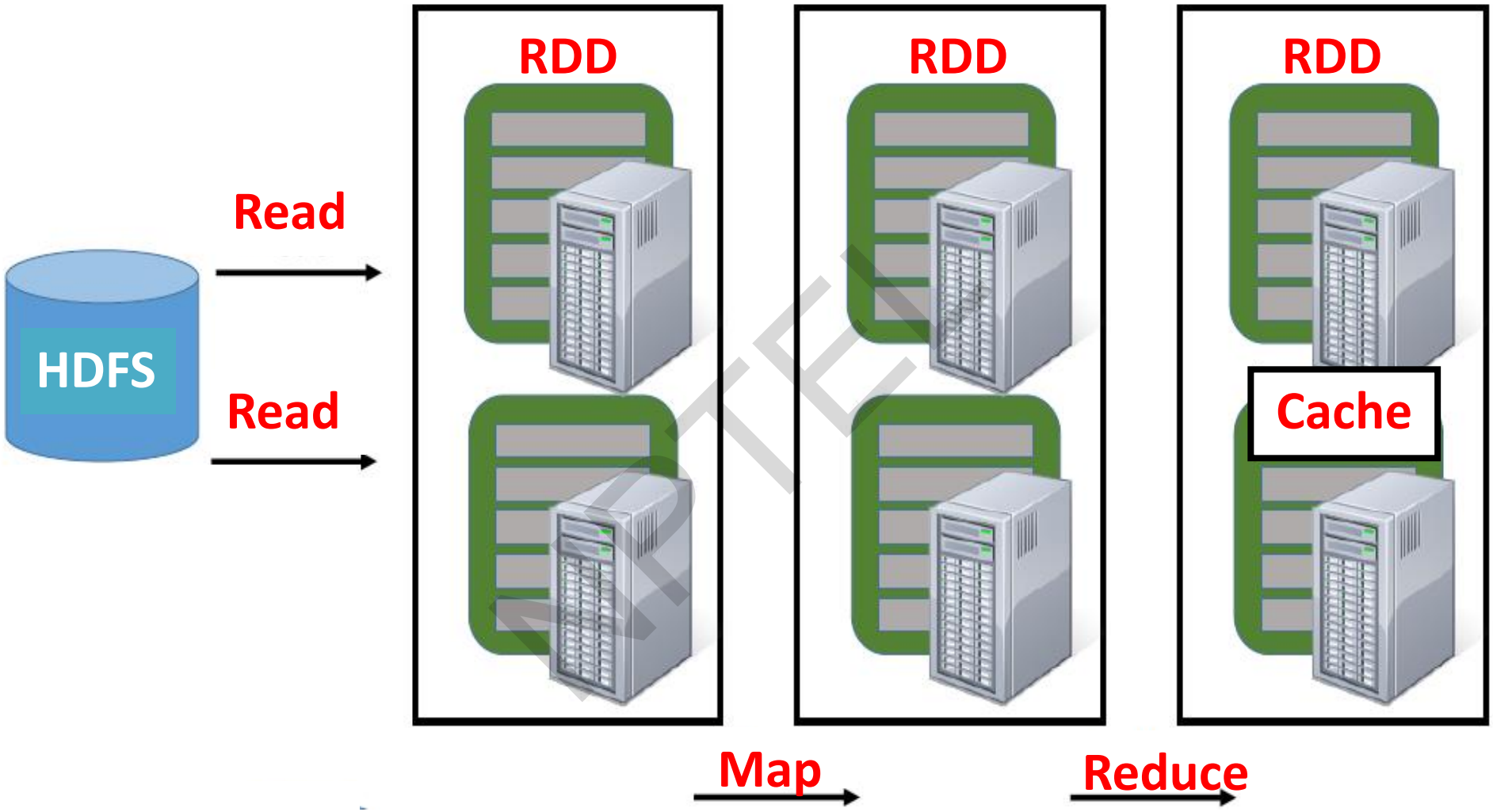
Resilient Distributed Datasets (RDDs)

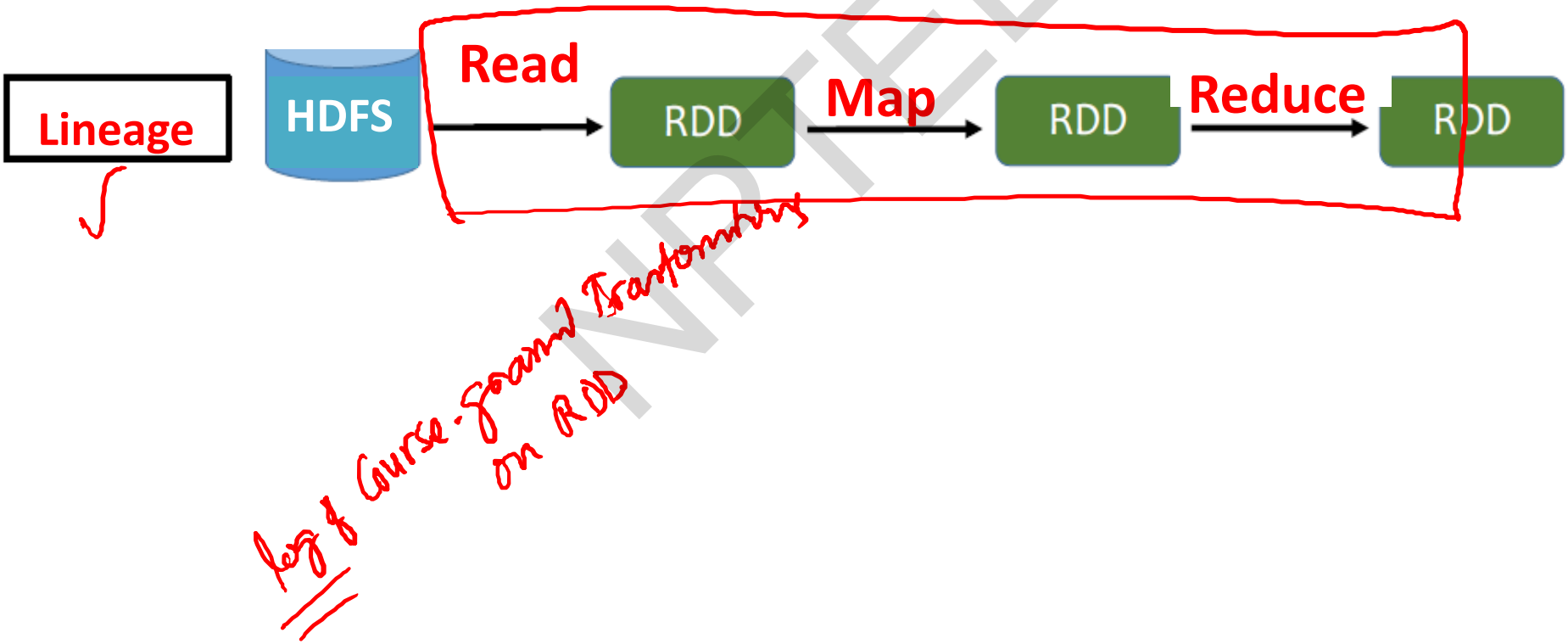
- Immutable, partitioned collection of records
- Built through coarse grained transformations (map, join …)

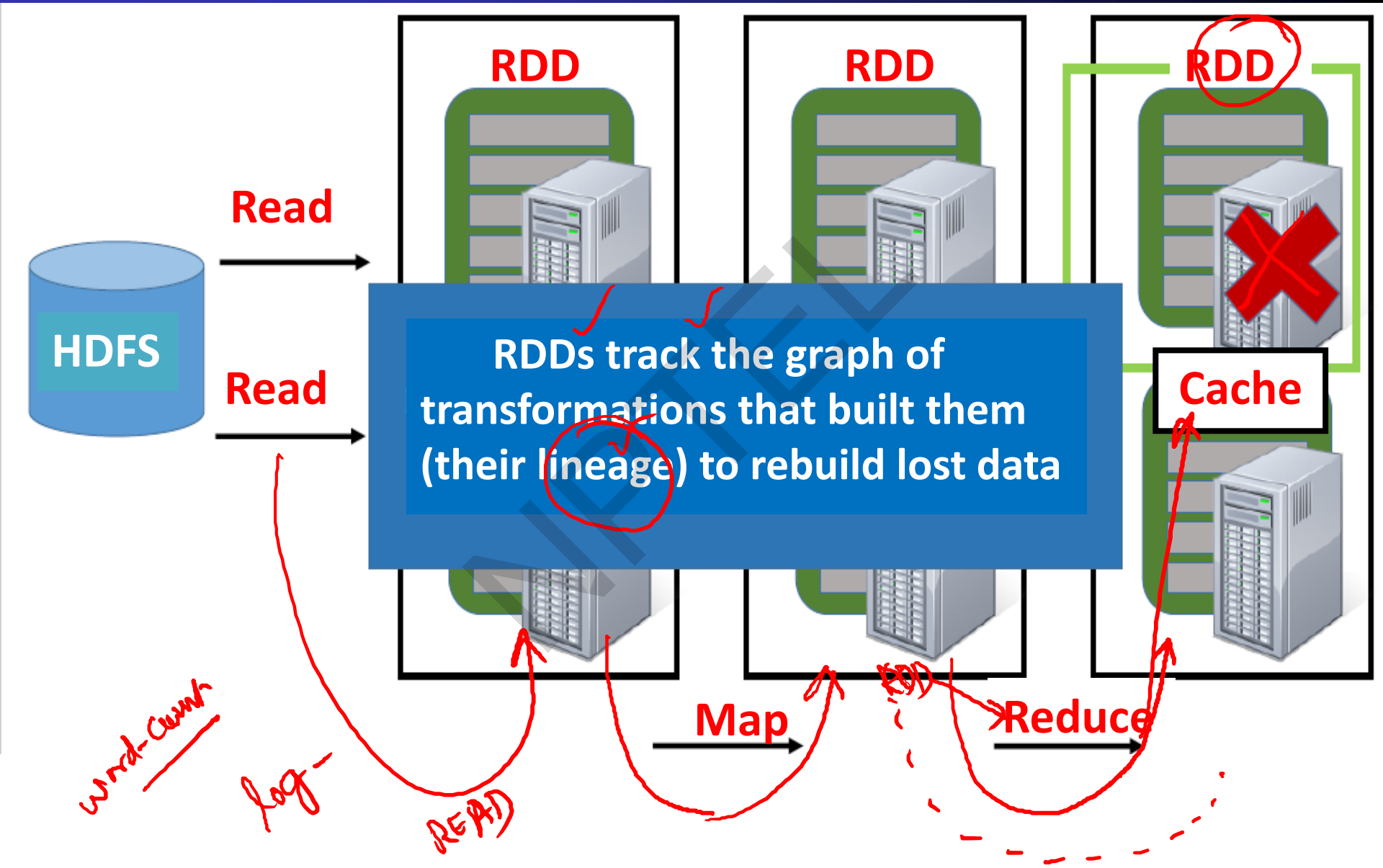
Fault Recovery?

- Lineage!
  - Log the coarse grained operation applied to a partitioned dataset
  - Simply recompute the lost partition if failure occurs!
  - No cost if no failure

Concept used by Spark for fault-tolerance is called Lineage!


RDD
RDD
RDD
Read
Read
HDFS
Cache
Map
Reduce


Lineage
HDFS
Read
RDD
Map
RDD
Reduce
RDD
log of Course-Grained Transformations on RDD


RDD
RDD
RDD
Read
HDFS
Read
RDDs track the graph of transformations that built them (their lineage) to rebuild lost data
Cache
Map
Reduce

What can you do with Spark?

- **RDD operations**
  - Transformations e.g., filter, join, map, group-by …
  - Actions e.g., count, print …
- **Control**
  - **Partitioning:** Spark also gives you control over how you can partition your RDDs.
  - **Persistence:** Allows you to choose whether you want to persist RDD onto disk or not.

Spark Applications

i. Twitter spam classification

ii. EM algorithm for traffic prediction

iii. K-means clustering

iv. Alternating Least Squares matrix factorization

v. In-memory OLAP aggregation on Hive data

vi. SQL on Spark

Reading Material

- Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica

"Spark: Cluster Computing with Working Sets"

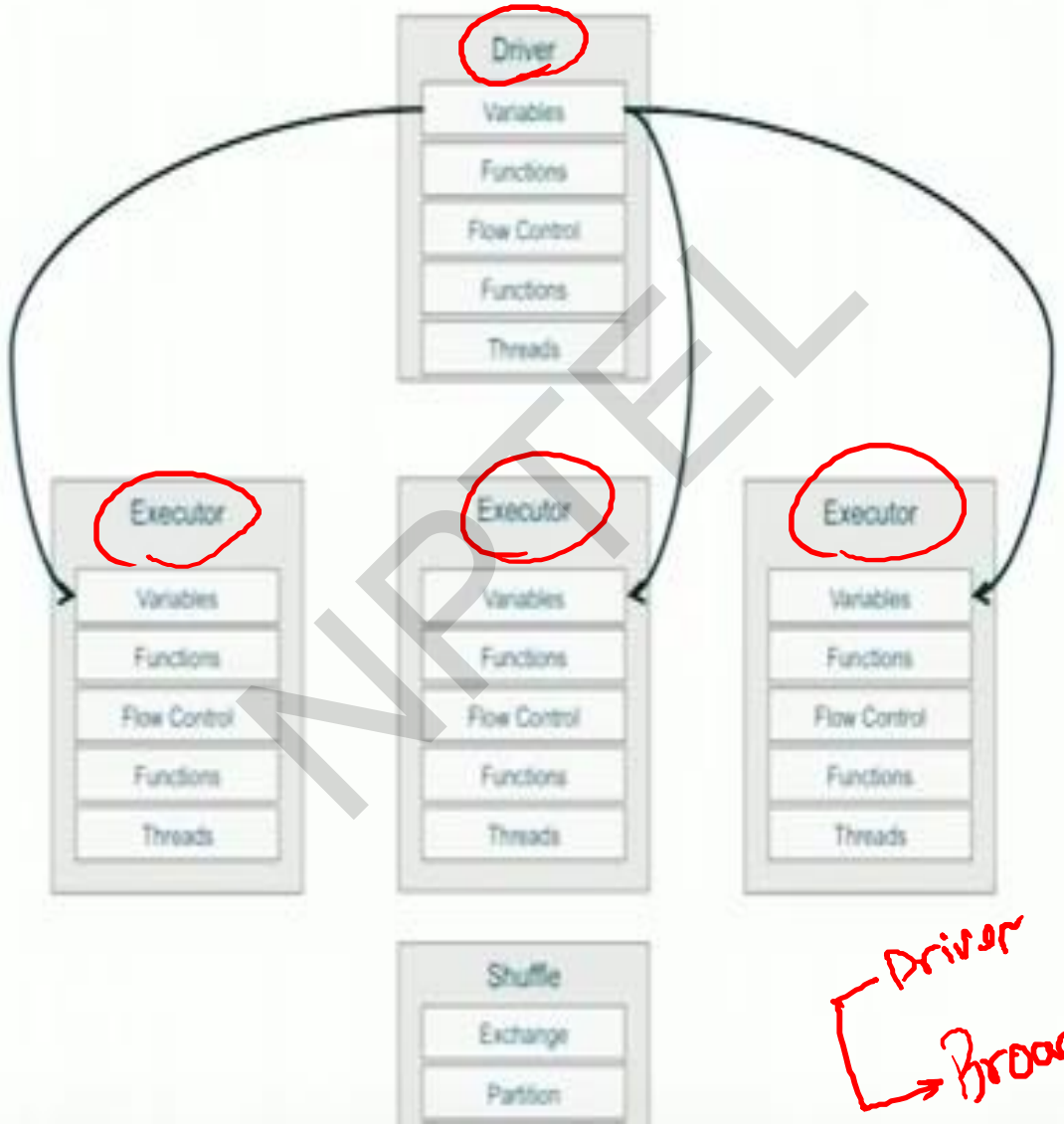
- Matei Zaharia, Mosharaf Chowdhury et al.

"Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing"

https://spark.apache.org/

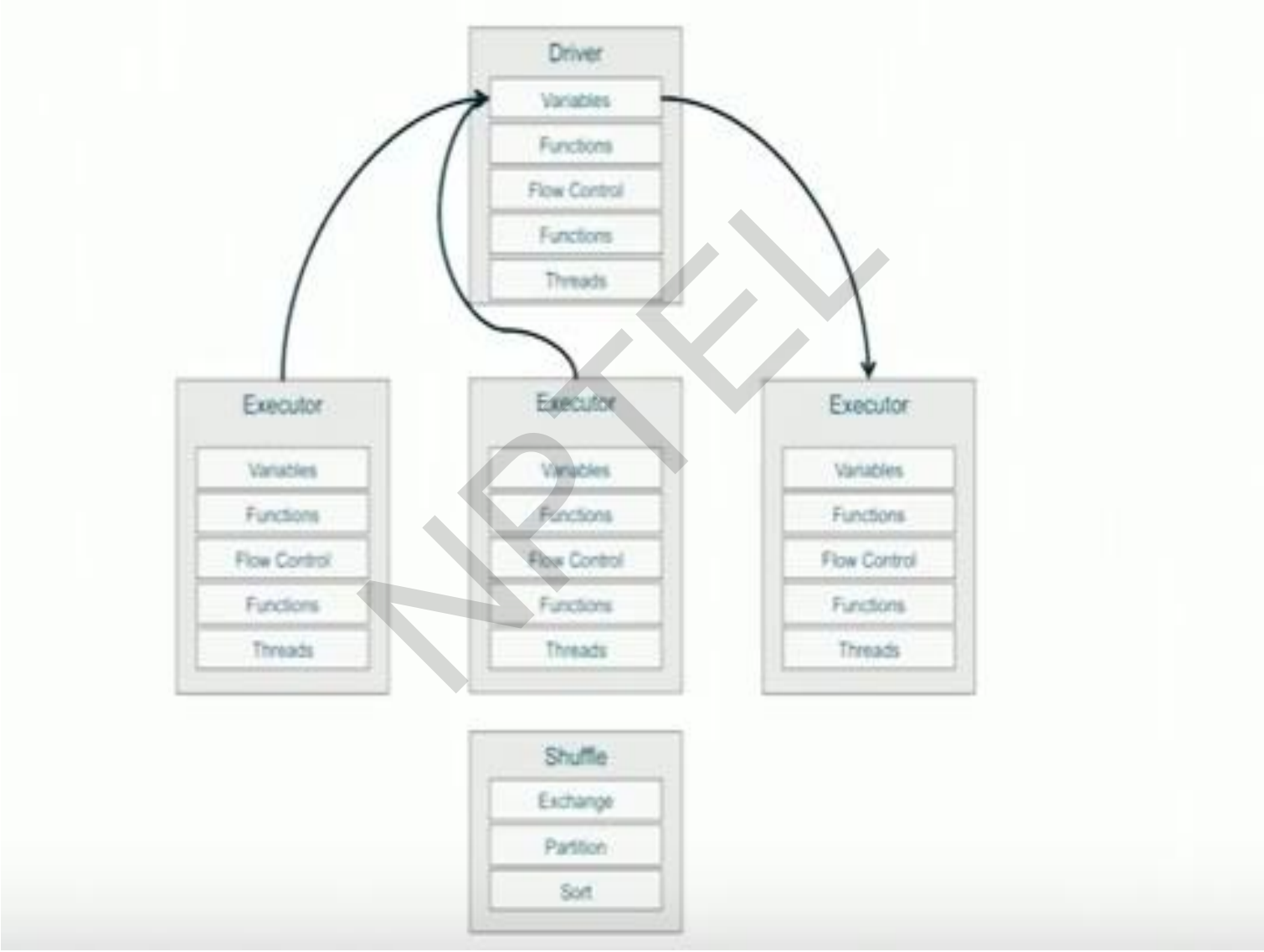
Spark Execution

Distributed Programming (Broadcast)

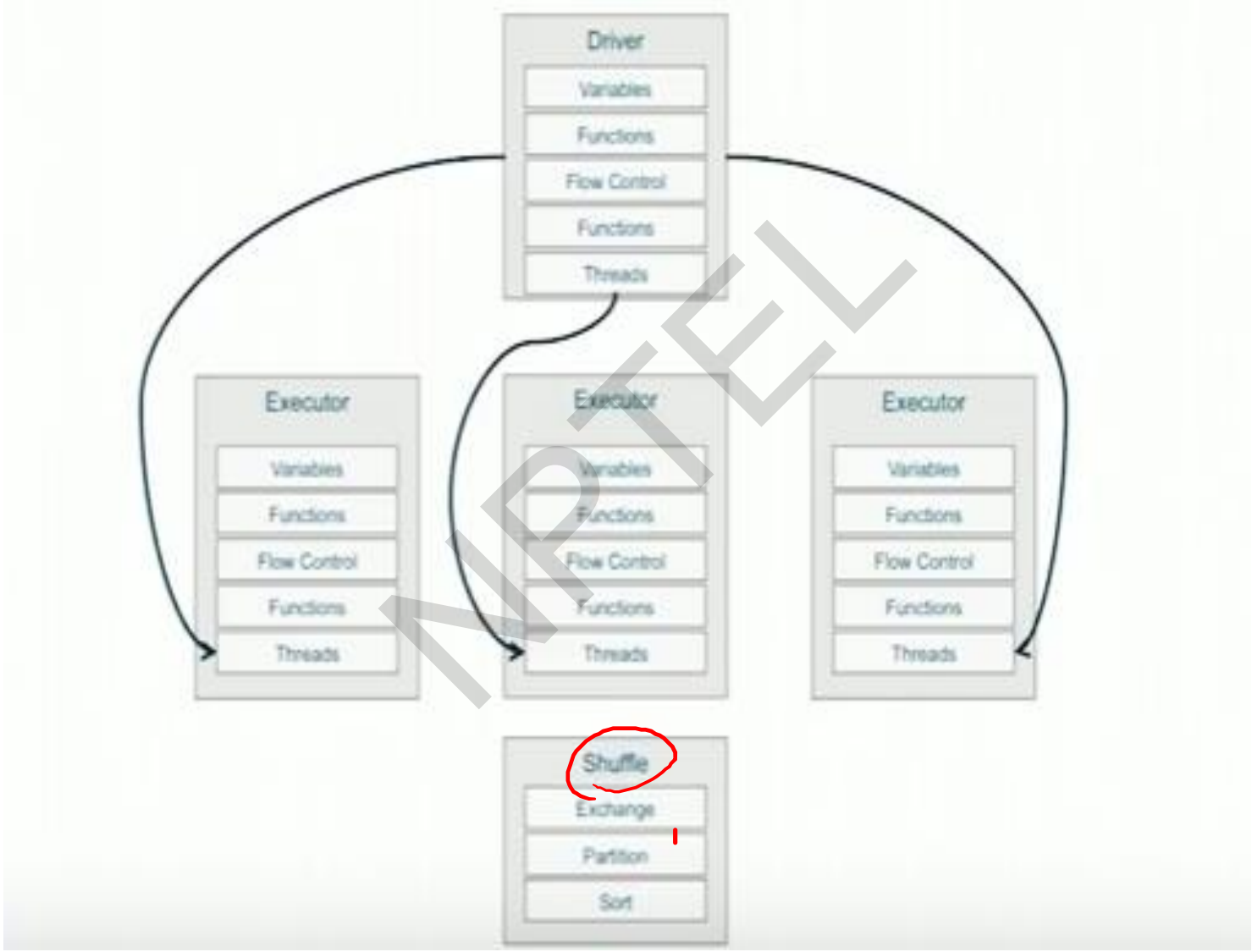


Driver → Broadcast to Executors

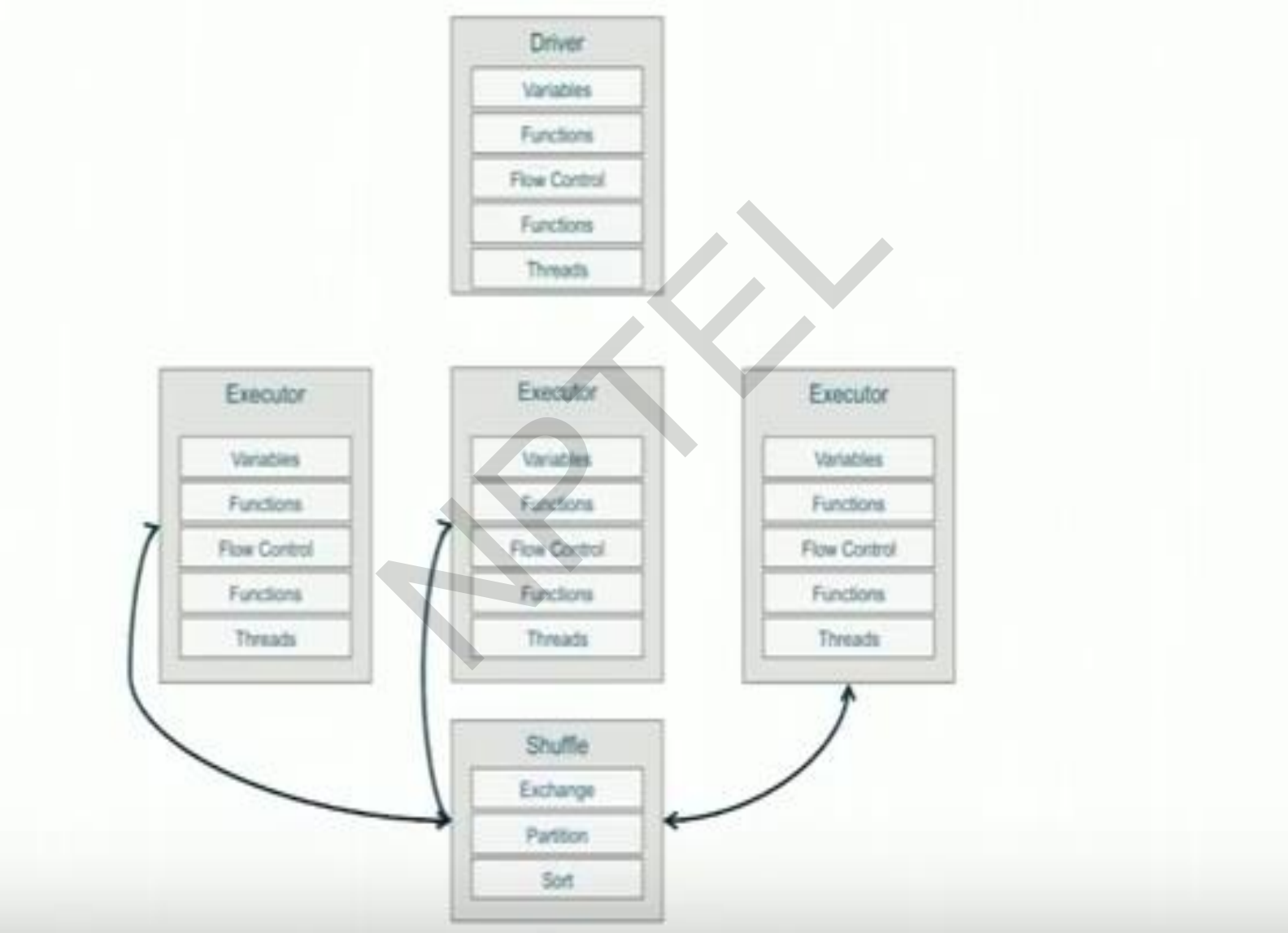
Distributed Programming (Take)



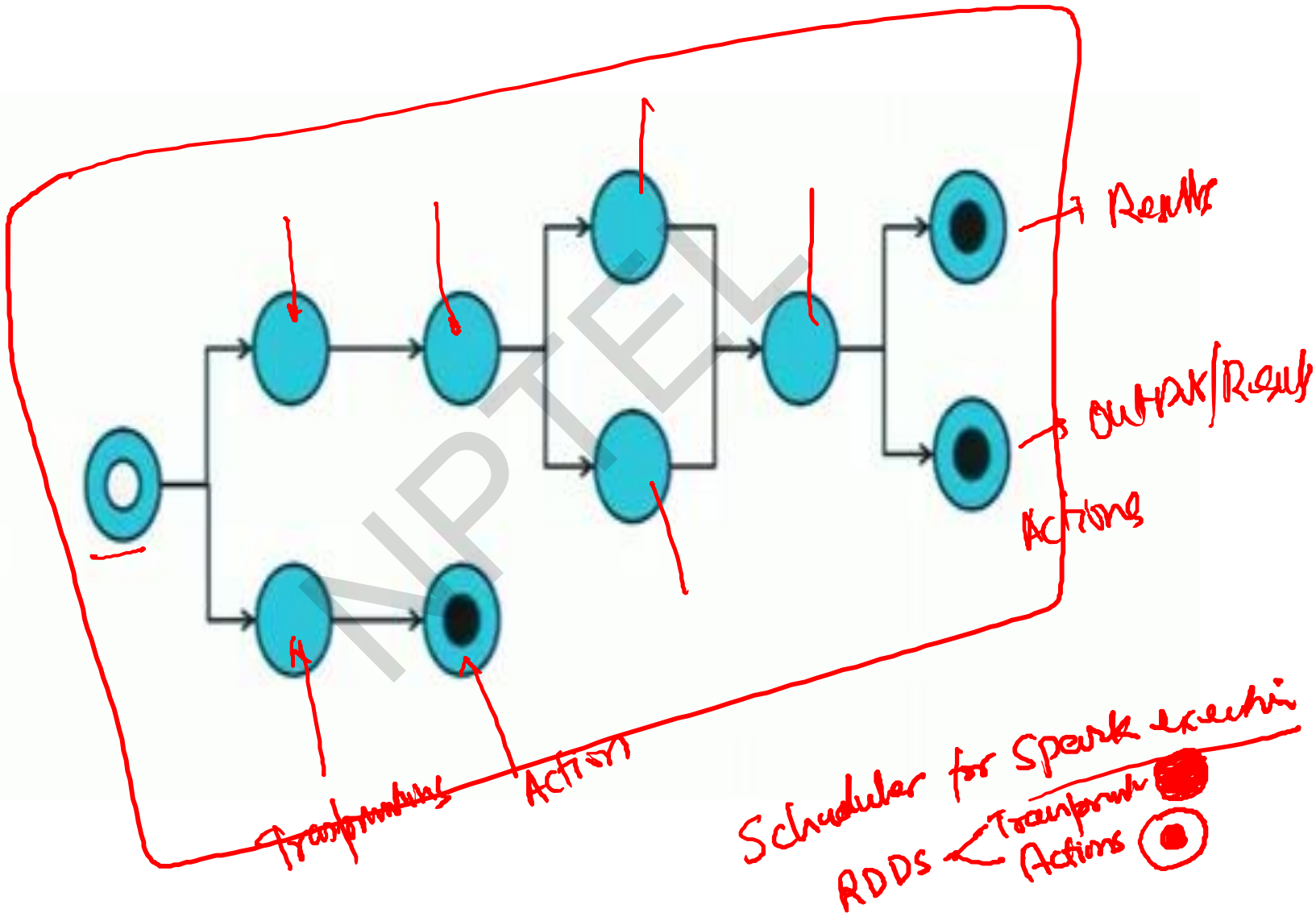
Distributed Programming (DAG Action)



Distributed Programming (Shuffle)



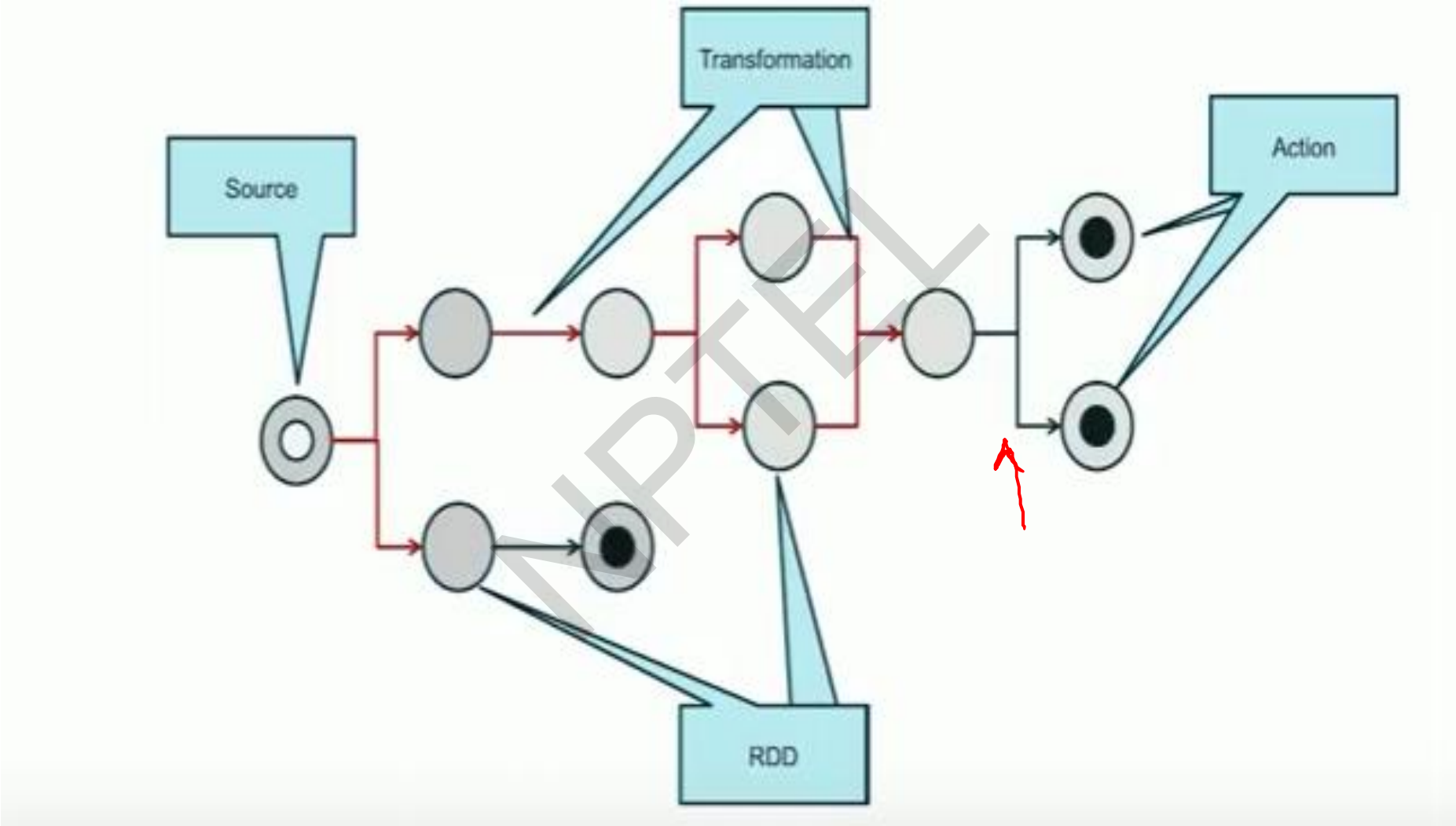
DAG (Directed Acyclic Graph)



DAG (Directed Acyclic Graph)

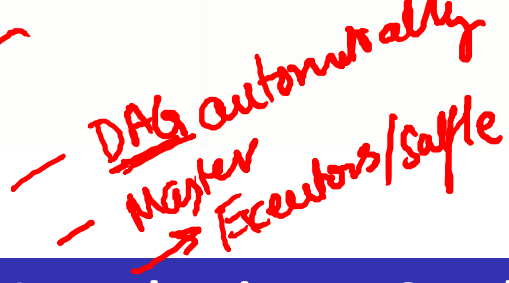
- **Action**
  - Count
  - Take
  - Foreach
- **Transformation**
  - Map
  - ReduceByKey
  - GroupByKey
  - JoinByKey

DAG (Directed Acyclic Graph)



Flume Java

```
1. val conf = new SparkConf().setMaster("local[2]")
2. val sc = new SparkContext(conf)
3. val lines = sc.textFile(path, 2)
4. val words = lines.flatMap(_.split(" "))
5. val pairs = words.map(word => (word, 1))
6. val wordCounts = pairs.reduceByKey(_ + _)
7. val localValues = wordCounts.take(100)
8. localValues.foreach(r => println(r))
```

Spark Implementation

Spark ideas

- Expressive computing system, not limited to map-reduce model
- Facilitate system memory
  - avoid saving intermediate results to disk
  - cache data for repetitive queries (e.g. for machine learning)
- Compatible with Hadoop

RDD abstraction

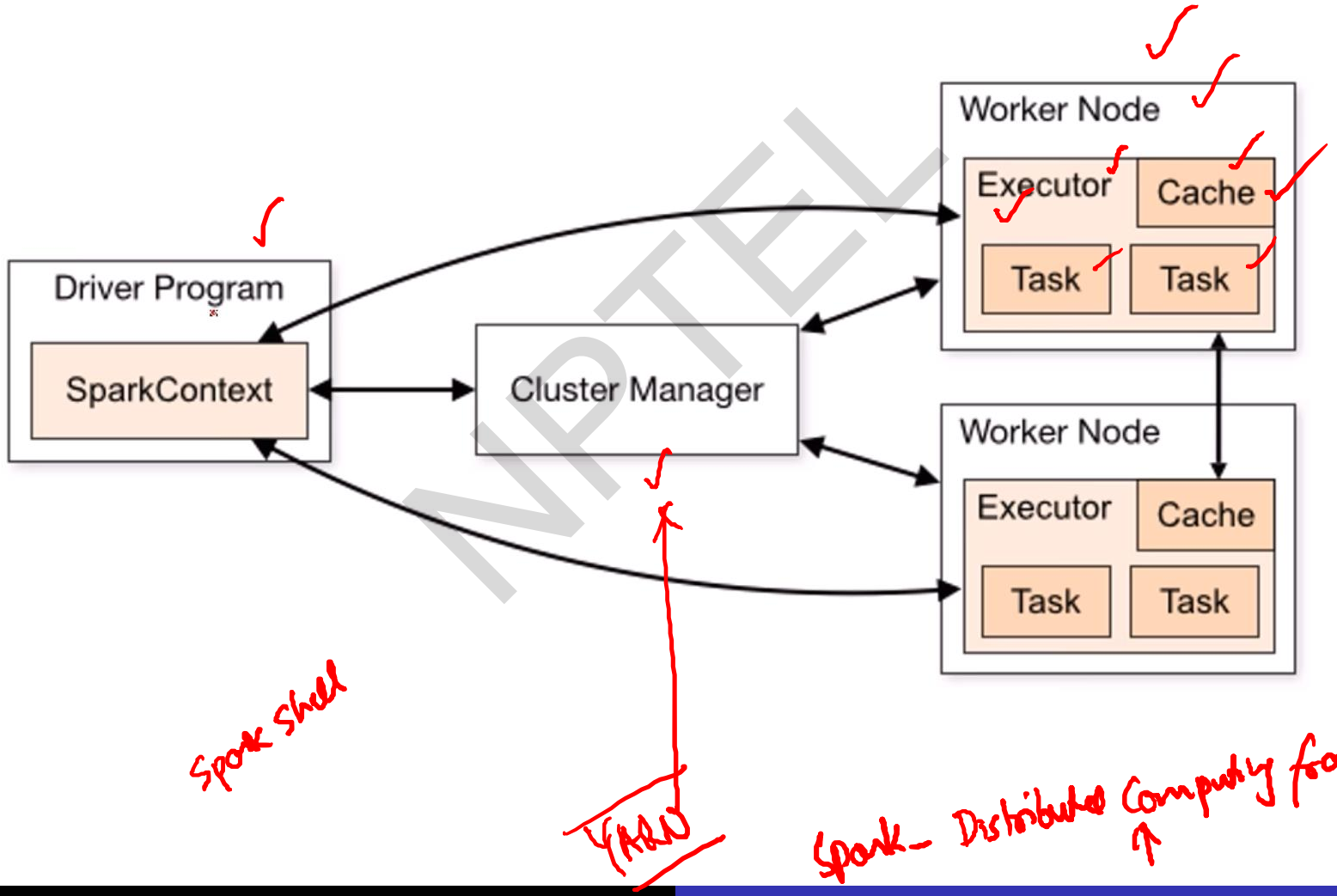
- Resilient Distributed Datasets
- Partitioned collection of records
- Spread across the cluster
- Read-only
- Caching dataset in memory
  - different storage levels available
  - fallback to disk possible

RDD operations

- *Transformations* to build RDDs through deterministic operations on other RDDs
  - transformations include *map*, *filter*, *join*
  - lazy operation

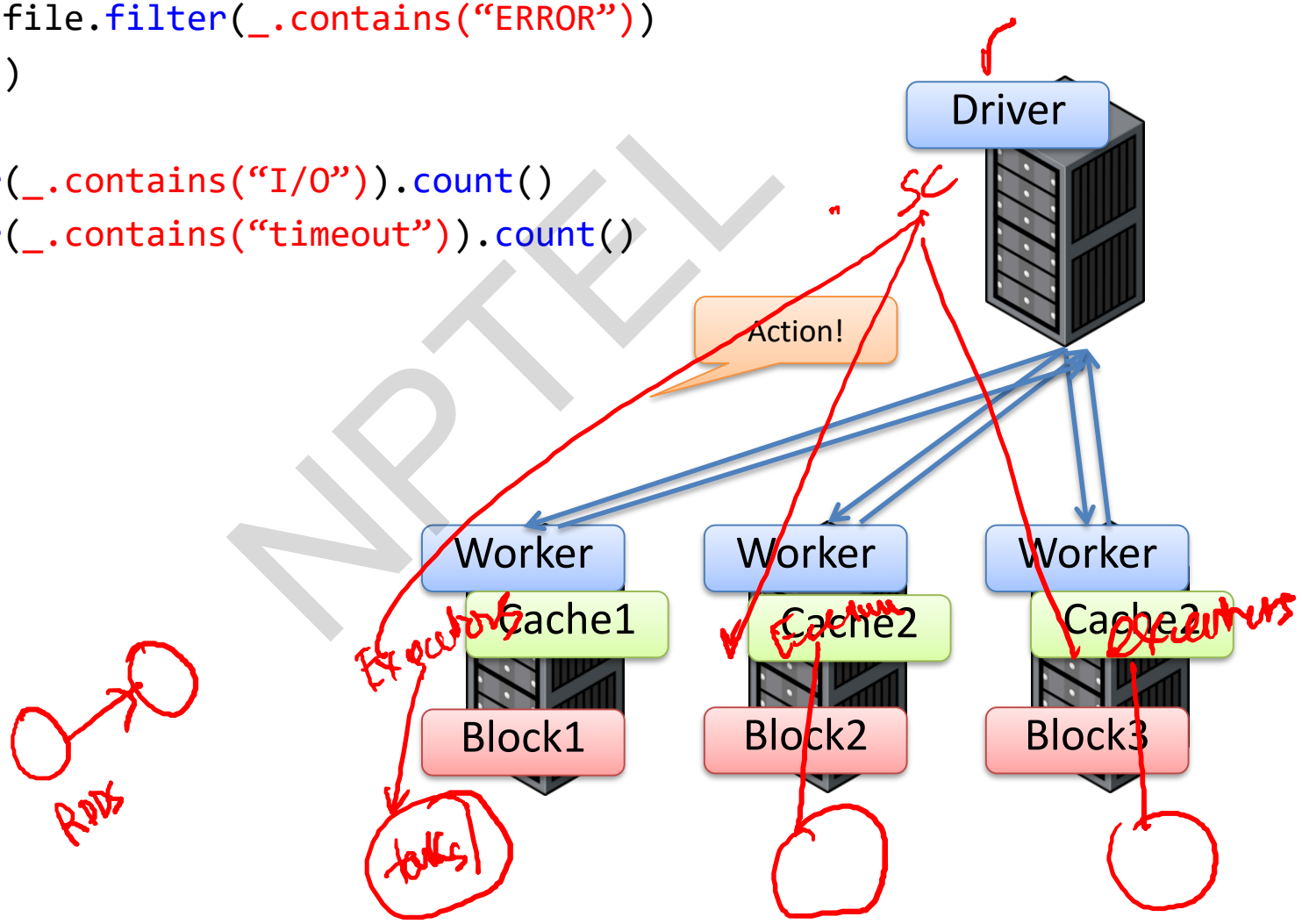
- *Actions* to return value or export data
  - actions include *count*, *collect*, *save*
  - triggers execution

Spark Components



Job example

```
val log = sc.textFile("hdfs://...")
val errors = file.filter(_.contains("ERROR"))
errors.cache()

errors.filter(_.contains("I/O")).count()
errors.filter(_.contains("timeout")).count()
```

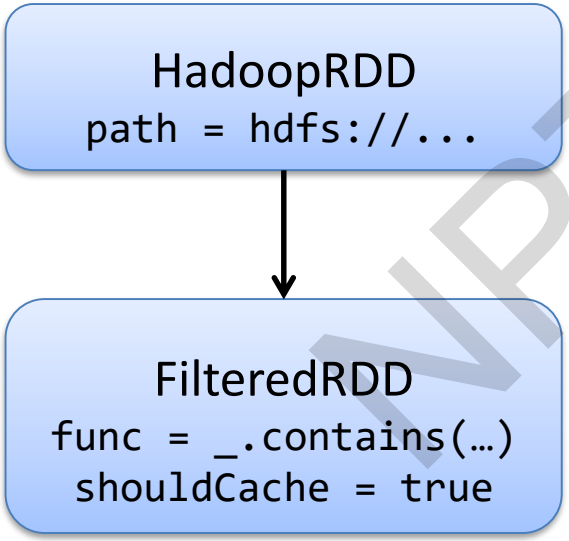


RDD partition-level view

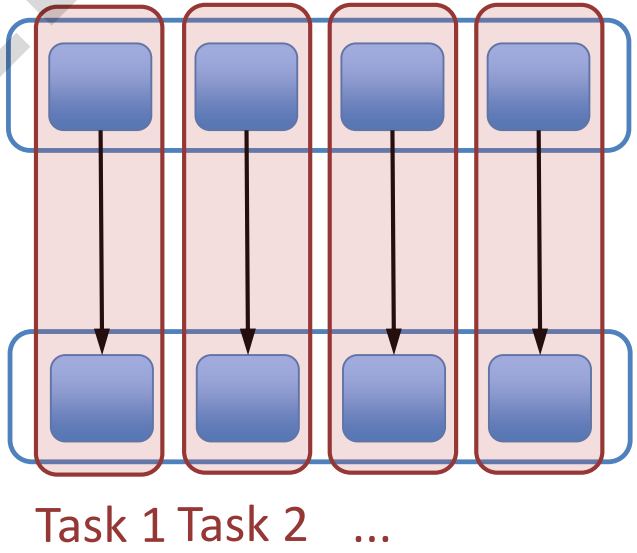
Dataset-level view:

log:

errors:

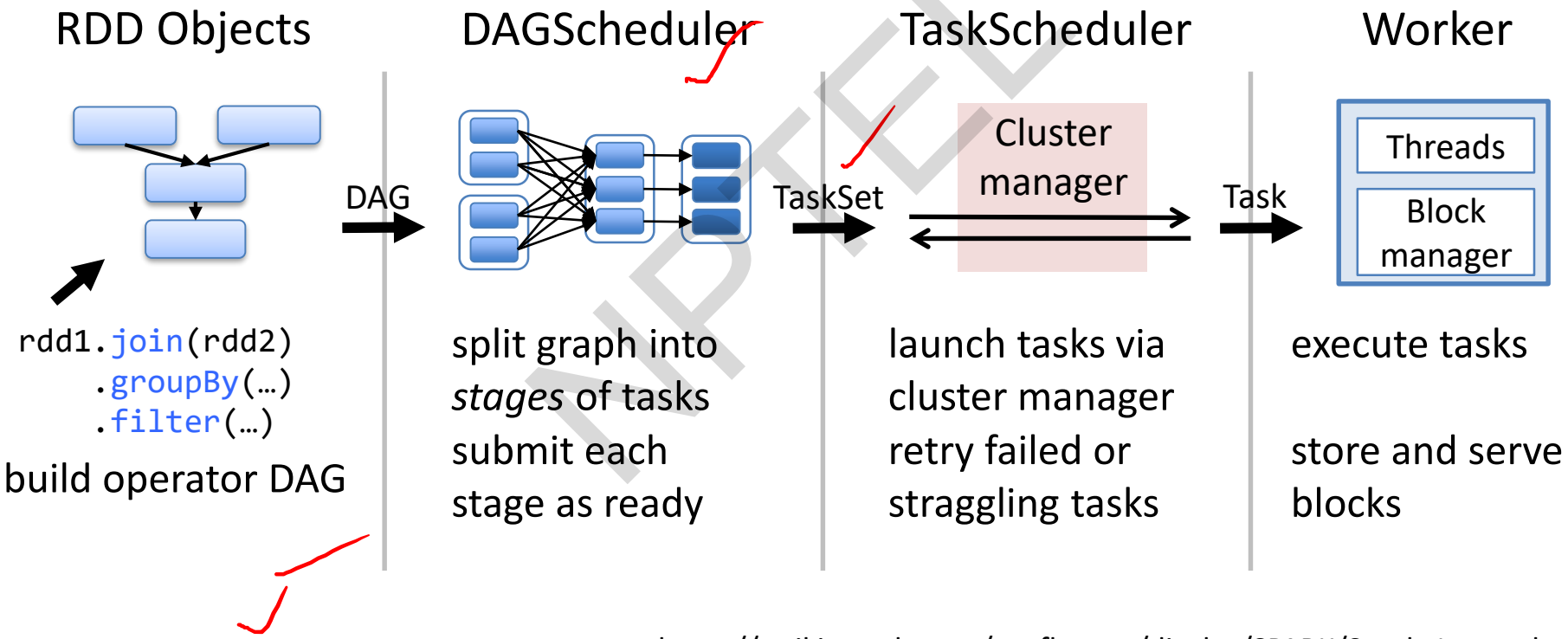


Partition-level view:



source: https://cwiki.apache.org/confluence/display/SPARK/Spark+Internals

Job scheduling



source: https://cwiki.apache.org/confluence/display/SPARK/Spark+Internals

Available APIs

- You can write in Java, Scala or Python
- Interactive interpreter: Scala & Python only
- Standalone applications: any
- Performance: Java & Scala are faster thanks to static typing

Hand on - interpreter

- script

```
http://cern.ch/kacper/spark.txt
```

- run scala spark interpreter

```
$ spark-shell
```

- or python interpreter

```
$ pyspark
```

Hand on – build and submission

- download and unpack source code

```
wget http://cern.ch/kacper/GvaWeather.tar.gz; tar -xzf GvaWeather.tar.gz
```

- build definition in

```
GvaWeather/gvaweather.sbt
```

- source code

```
GvaWeather/src/main/scala/GvaWeather.scala
```

- building

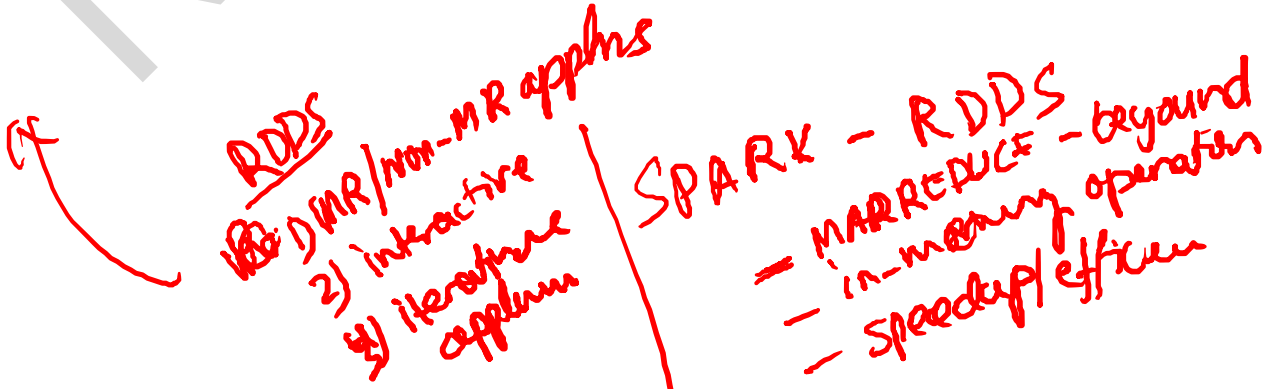
```
cd GvaWeather
sbt package
```

- job submission

```
spark-submit --master local --class GvaWeather \
        target/scala-2.10/gva-weather_2.10-1.0.jar
```

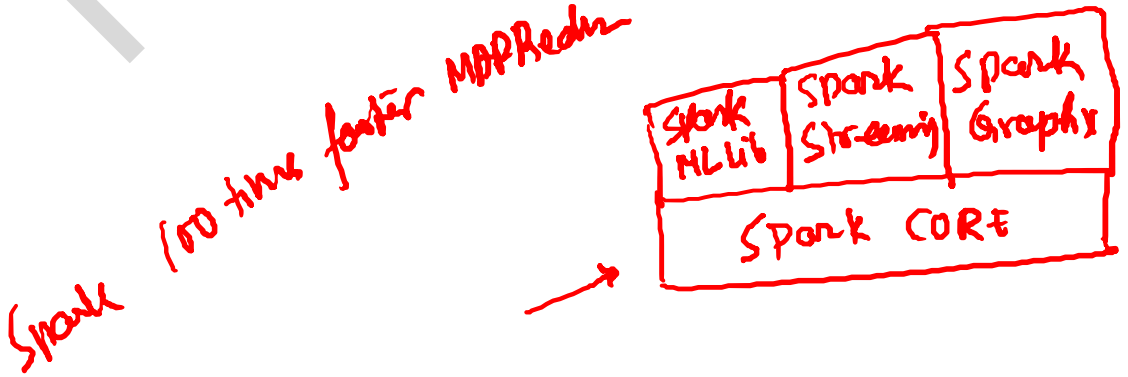
Summary

- Concept not limited to single pass map-reduce
- Avoid sorting intermediate results on disk or HDFS
- Speedup computations when reusing datasets



Conclusion

- RDDs (Resilient Distributed Datasets (RDDs) provide a simple and efficient programming model
- Generalized to a broad set of applications
- Leverages coarse-grained nature of parallel algorithms for failure recovery



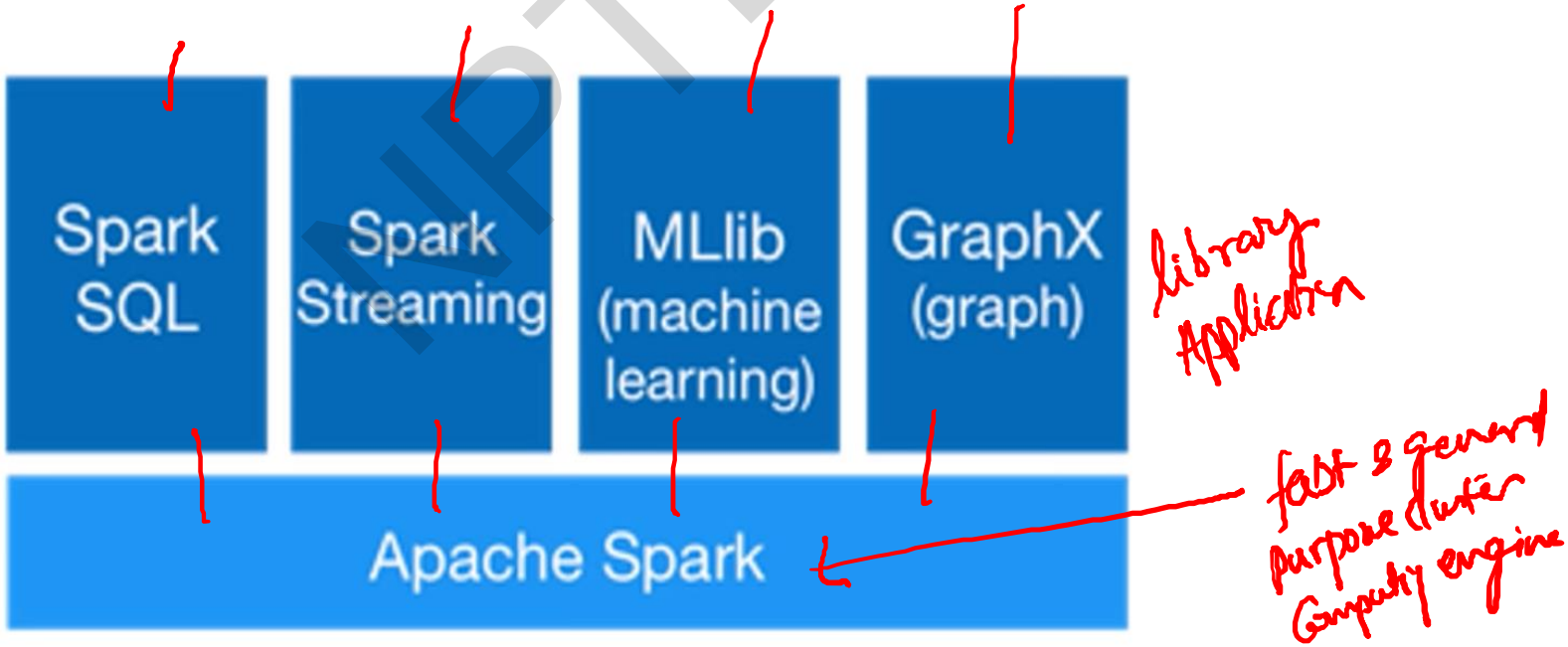
Spark Built-in Libraries

Dr. Rajiv Misra
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Introduction

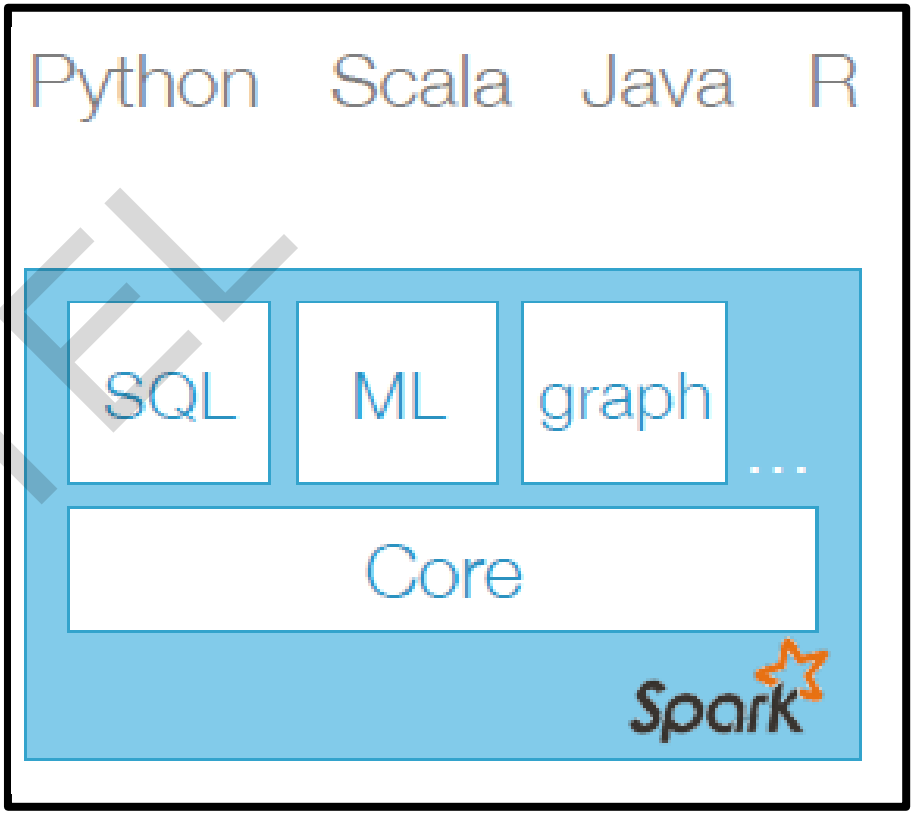
- Apache Spark is a fast and general-purpose cluster computing system for large scale data processing

- High-level APIs in Java, Scala, Python and R

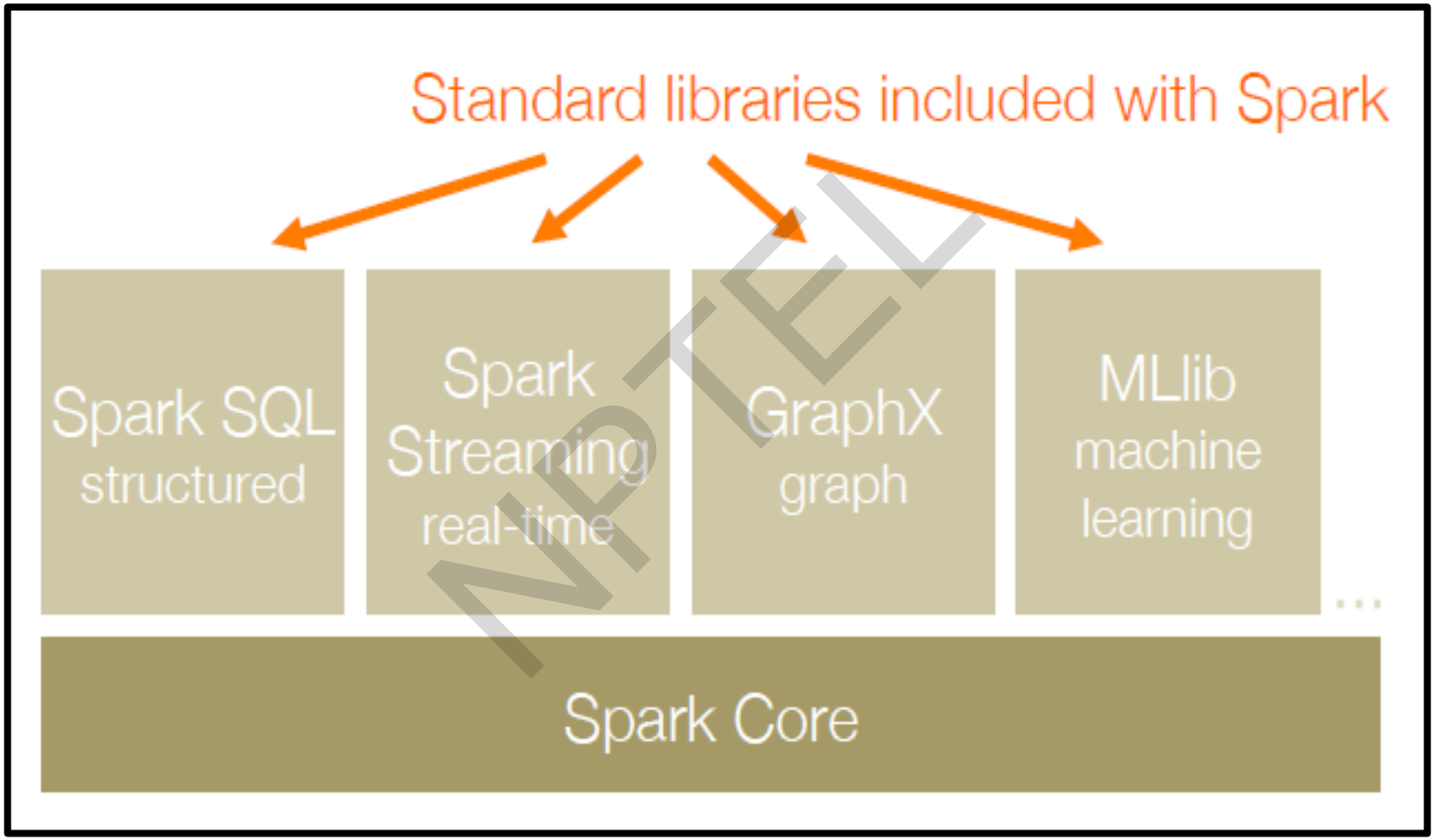


Standard Library for Big Data

- Big data apps lack libraries“ of common algorithms
- Spark’s generality + support“ for multiple languages make it“ suitable to offer this
- Much of future activity will be in these libraries



A General Platform



Machine Learning Library (MLlib)

MLlib algorithms:

(i) Classification: logistic regression, linear SVM," naïve Bayes, classification tree

(ii) Regression: generalized linear models (GLMs), regression tree

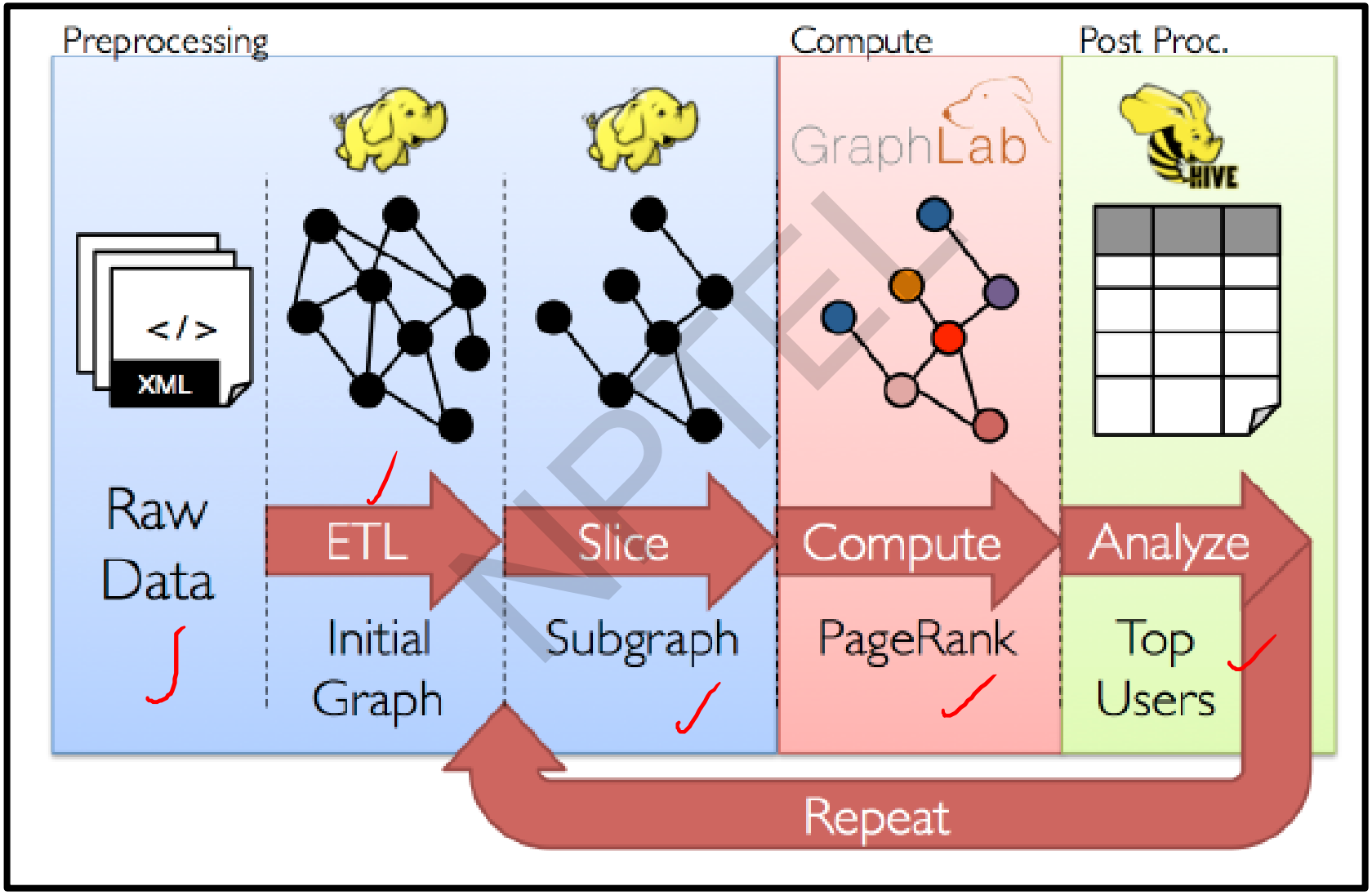
(iii) Collaborative filtering: alternating least squares (ALS), non-negative matrix factorization (NMF)

(iv) Clustering: k-means

(v) Decomposition: SVD, PCA

(vi) Optimization: stochastic gradient descent, L-BFGS

GraphX


Preprocessing
Compute
Post Proc.
GraphLab
HIVE
XML
Raw Data
ETL
Slice
Compute
Analyze
Initial Graph
Subgraph
PageRank
Top Users
Repeat

GraphX

- General graph processing library
- Build graph using RDDs of nodes and edges
- Large library of graph algorithms with composable steps

GraphX Algorithms

(i) Collaborative Filtering
Alternating Least Squares
Stochastic Gradient Descent
Tensor Factorization

(ii) Structured Prediction
Loopy Belief Propagation
Max-Product Linear Programs
Gibbs Sampling

(iii) Semi-supervised ML
Graph SSL
CoEM

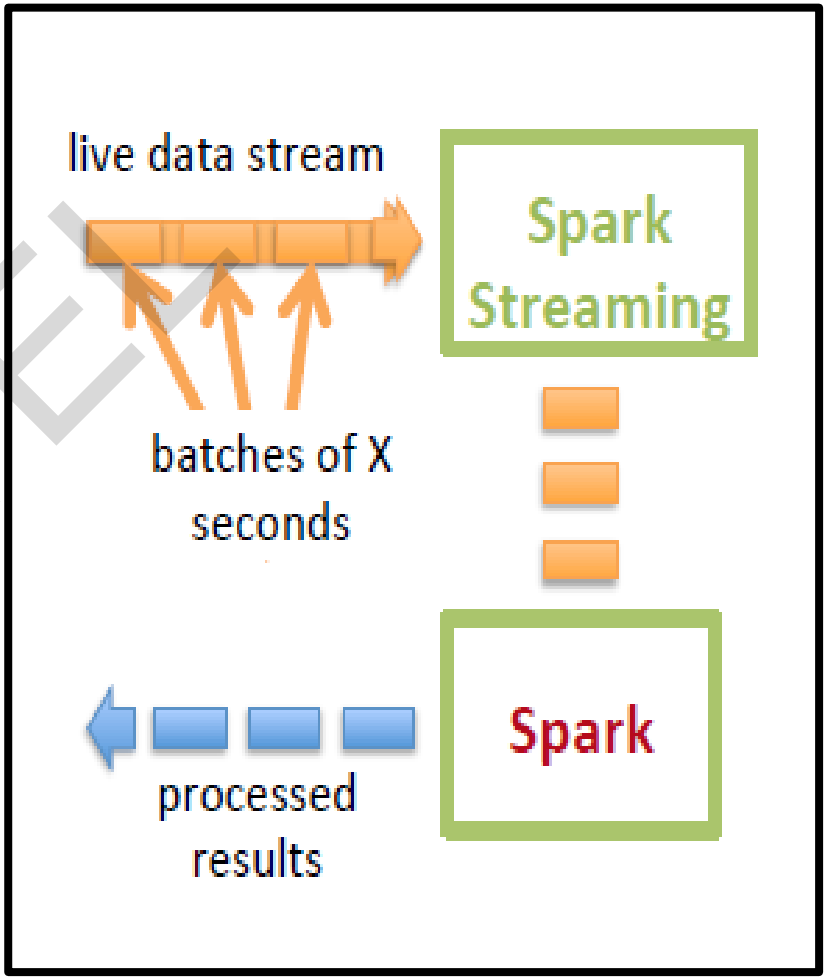
(iv) Community Detection
Triangle-Counting
K-core Decomposition
K-Truss

(v) Graph Analytics
PageRank
Personalized PageRank
Shortest Path
Graph Coloring

(vi) Classification
Neural Networks

Spark Streaming

- Large scale streaming computation
- Ensure exactly one semantics
- Integrated with Spark → unifies batch, interactive, and streaming computations!



Spark SQL

Enables loading & querying structured data in Spark

From Hive:

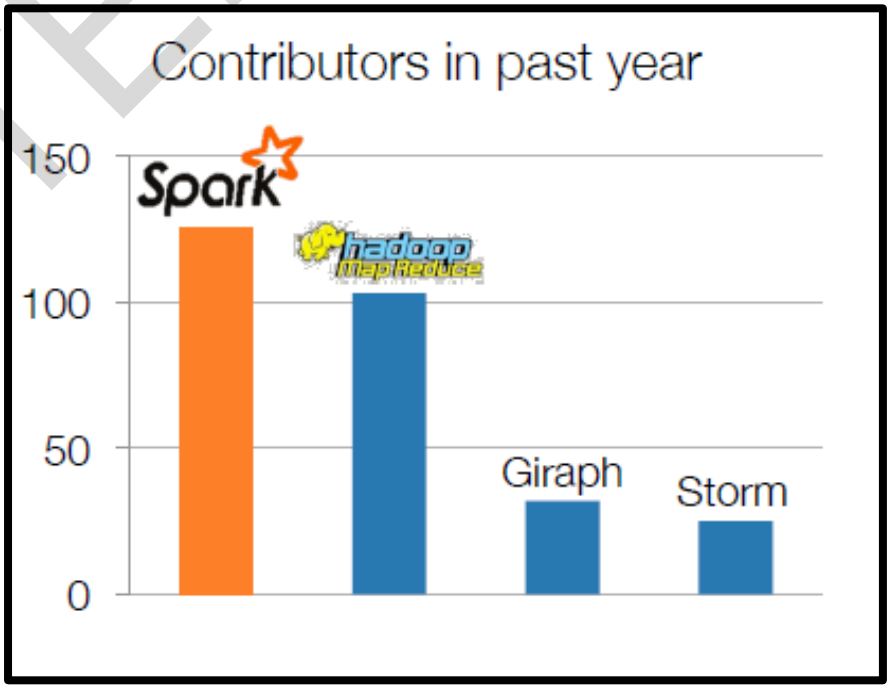
```
c = HiveContext(sc)
rows = c.sql("select text, year from hivetable")
rows.filter(lambda r: r.year > 2013).collect()
```

From JSON:

```
c.jsonFile("tweets.json").registerAsTable("tweets")
c.sql("select text, user.name from tweets")
```

Spark Community

- Most active open source community in big data
- 200+ developers, 50+ companies contributing



Design of Key-Value Stores

Dr. Rajiv Misra
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss the design and insight of **Key-value/NoSQL stores** for today's cloud storage systems.

- We will also discuss one of the most popular cloud storage system i.e. **Apache Cassandra** and different consistency solutions.

The Key-value Abstraction

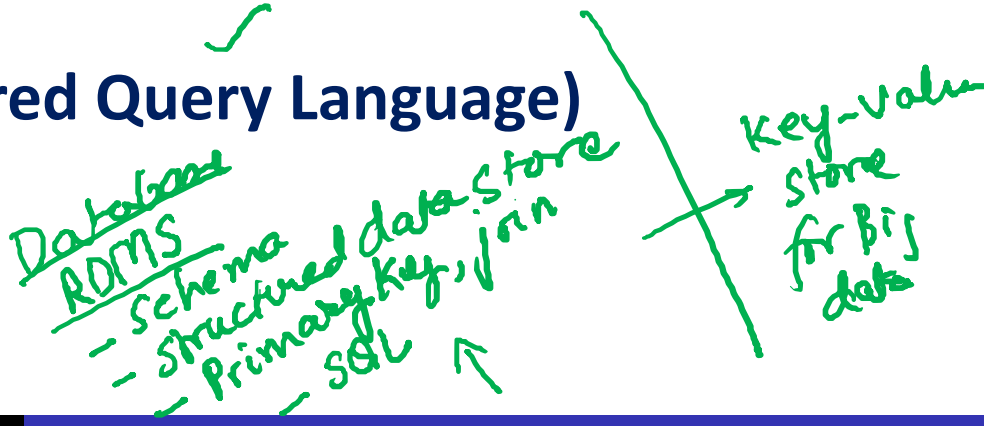
- (Business) Key → Value
- (flipkart.com) item number → information about it
- (easemytrip.com) Flight number → information about flight, e.g., availability
- (twitter.com) tweet id → information about tweet
- (mybank.com) Account number → information about it

The Key-value Abstraction (2)

- **It's a dictionary datastructure.**
  - Insert, lookup, and delete by key
  - Example: hash table, binary tree
- But distributed.
- Seems familiar? Remember **Distributed Hash tables (DHT) in P2P systems?**
- Key-value stores reuse many techniques from DHTs.

Is it a kind of database ?

- Yes, kind of
- **Relational Database Management Systems (RDBMSs)** have been around for ages
- **MySQL** is the most popular among them
- Data stored in tables
- Schema-based, i.e., structured tables
- Each row (data item) in a table has a primary key that is unique within that table
- Queried using **SQL (Structured Query Language)**
- Supports joins



Relational Database Example

users table

user_id	name	zipcode	blog_url	blog_id
110	Smith	98765	smith.com	11
331	Antony	54321	antony.in	12
767	John	75676	john.net	13

Primary keys

Foreign keys

blog table

Id	url	last_updated	num_posts
11	smith.com	9/7/17	991
13	john.net	4/2/18	57
12	antony.in	15/6/16	1090

Example SQL queries

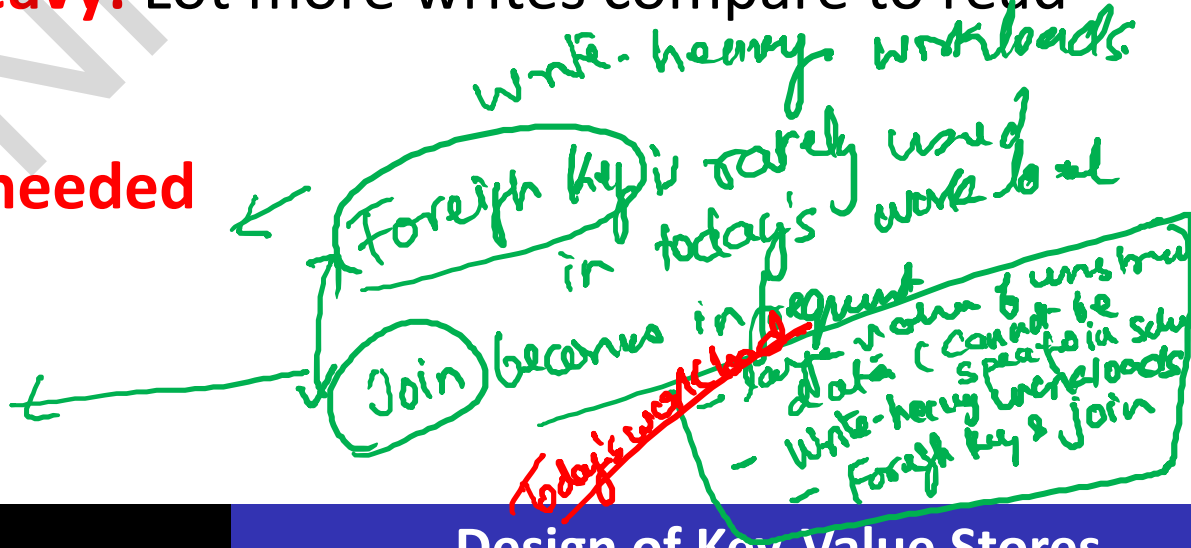
1. SELECT zipcode
   FROM users
   WHERE name = "John"

2. SELECT url
   FROM blog
   WHERE id = 11

3. SELECT users.zipcode, blog.num_posts
   FROM users JOIN blog
   ON users.blog_url = blog.url

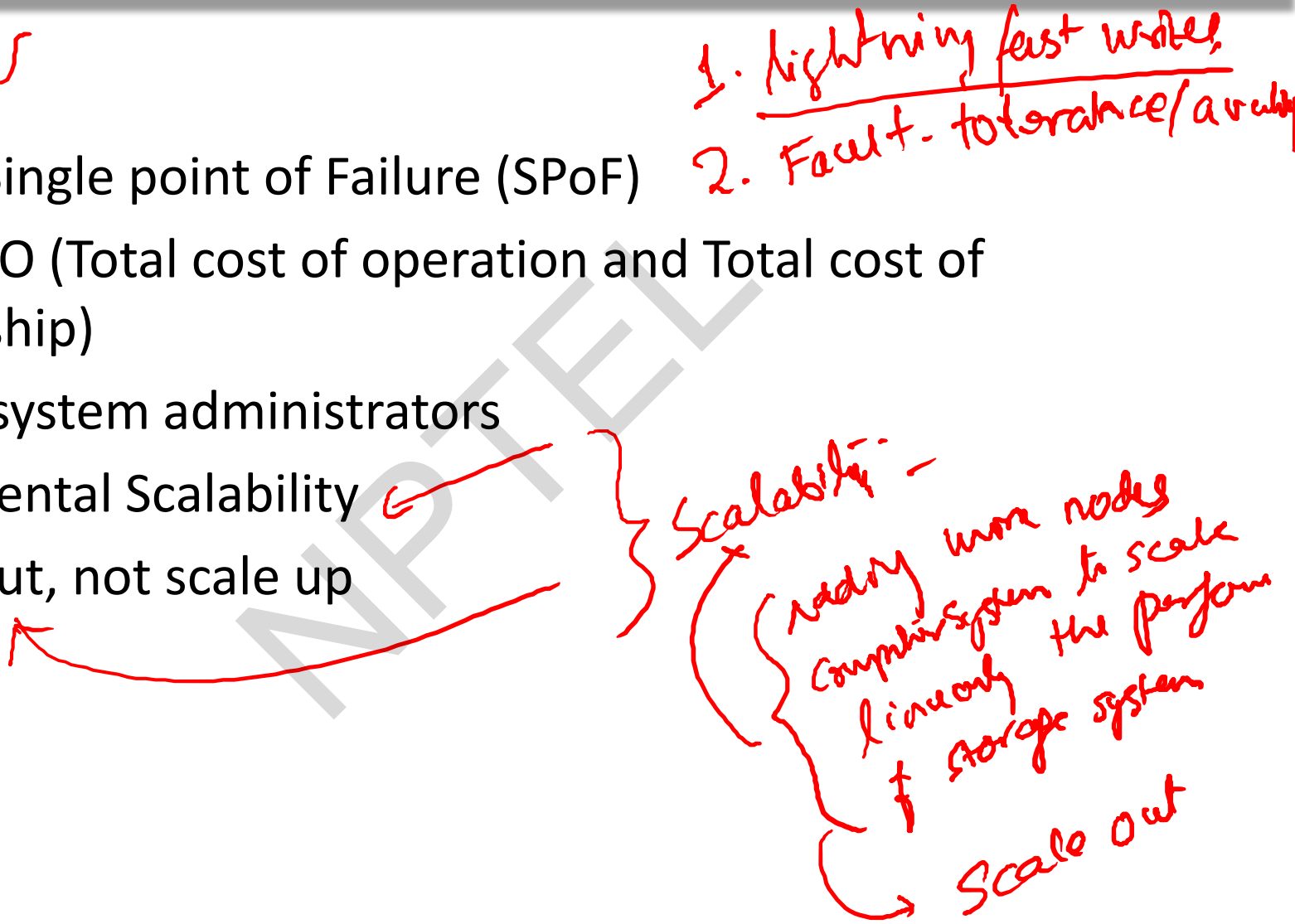
Mismatch with today's workloads

- **Data: Large and unstructured:** Difficult to come out with schemas where the data can fit
- **Lots of random reads and writes:** Coming from millions of clients.
- **Sometimes write-heavy:** Lot more writes compare to read
- **Foreign keys rarely needed**
- **Joins infrequent**



Needs of Today's Workloads

- Speed
- Avoid Single point of Failure (SPoF)
- Low TCO (Total cost of operation and Total cost of ownership)
- Fewer system administrators
- Incremental Scalability
- Scale out, not scale up



Scale out, not Scale up

- **Scale up = grow your cluster capacity by replacing with more powerful machines**
  - Traditional approach
  - Not cost-effective, as you're buying above the sweet spot on the price curve
  - And you need to replace machines often

- **Scale out = incrementally grow your cluster capacity by adding more COTS machines (Components Off the Shelf)**
  - Cheaper
  - Over a long duration, phase in a few newer (faster) machines as you phase out a few older machines
  - Used by most companies who run datacenters and clouds today

Key-value/NoSQL Data Model

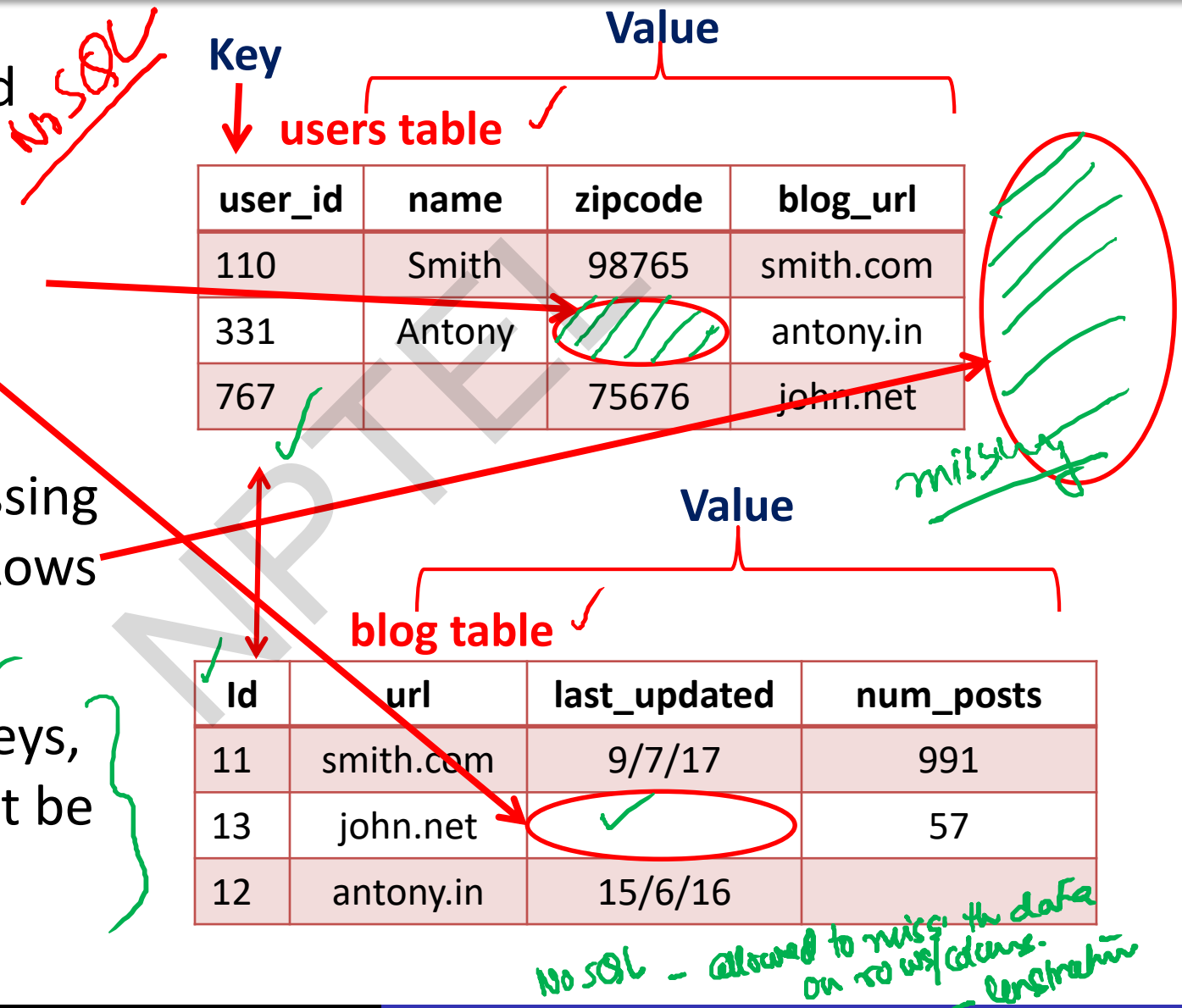
- NoSQL = "Not Only SQL"
- Necessary API operations: **get(key) and put(key, value)**
  - And some extended operations, e.g., "CQL" in Cassandra key-value store

- **Tables**
  - "Column families" in Cassandra, "Table" in HBase, "Collection" in MongoDB
  - Like RDBMS tables, but …
  - May be unstructured: May not have schemas
    - Some columns may be missing from some rows
  - Don't always support joins or have foreign keys
  - Can have index tables, just like RDBMSs

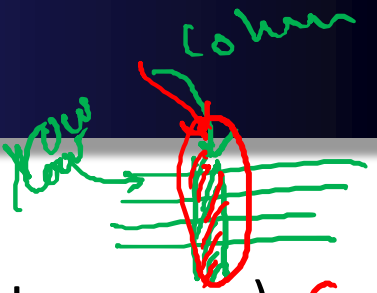
not have schema

Key-value/NoSQL Data Model

- Unstructured
- No schema imposed
- Columns missing from some Rows
- No foreign keys, joins may not be supported



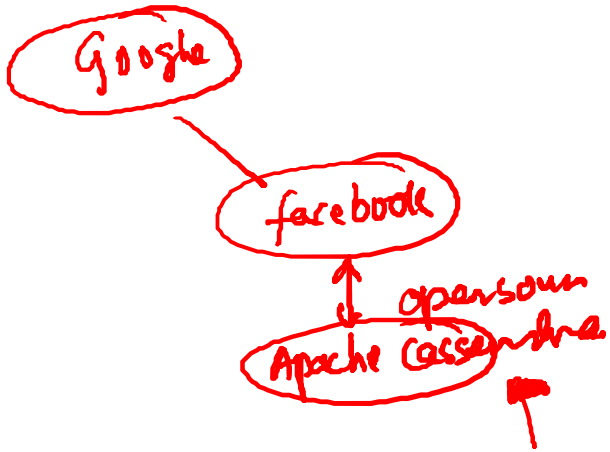
Column-Oriented Storage



NoSQL systems often use column-oriented storage

- RDBMSs store an entire row together (on disk or at a server)
- NoSQL systems typically store a column together (or a group of columns).
  - Entries within a column are indexed and easy to locate, given a key (and vice-versa)
- **Why useful?**
  - Range searches within a column are fast since you don't need to fetch the entire database
  - E.g., Get me all the blog_ids from the blog table that were updated within the past month
    - Search in the the last_updated column, fetch corresponding blog_id column
    - Don't need to fetch the other columns

Design of Apache Cassandra

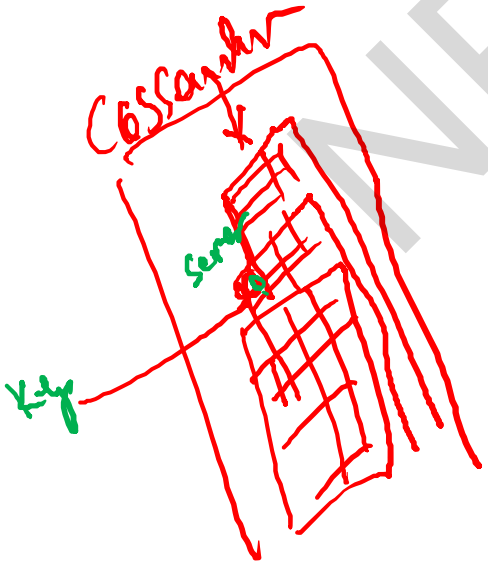


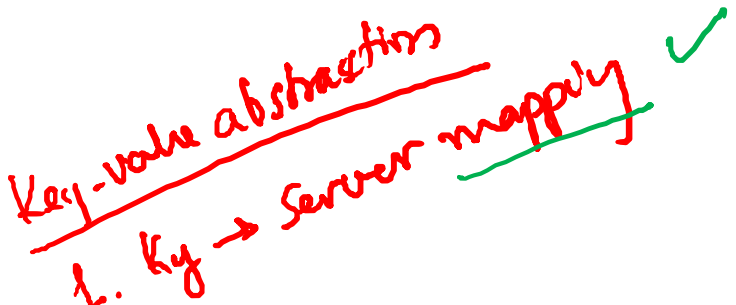
Cassandra

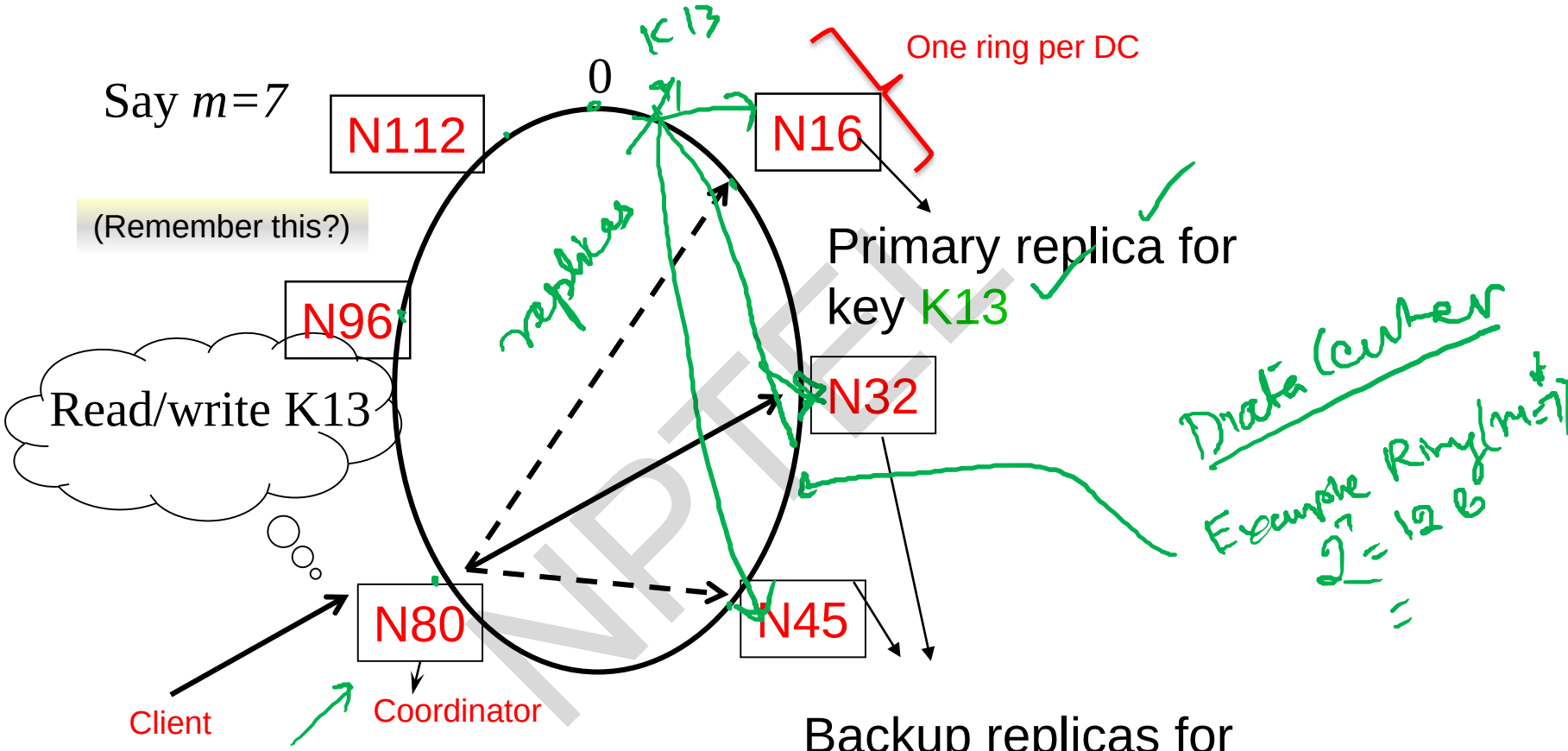
- **A distributed key-value store**
- Intended to run in a datacenter (and also across DCs)
- Originally designed at Facebook
- Open-sourced later, today an Apache project
- Some of the companies that use Cassandra in their production clusters
  - Blue chip companies: IBM, Adobe, HP, eBay, Ericsson
  - Newer companies: Twitter
  - Nonprofit companies: PBS Kids
  - Netflix: uses Cassandra to keep track of positions in the video.

Inside Cassandra: Key -> Server Mapping

- How do you decide which server(s) a key-value resides on?







Cassandra uses a Ring-based DHT but without finger tables or routing

Key → server mapping is the "Partitioner"