OCTOBER 13-16, 2016 • AUSTIN, TX

Spark Search

Taka Shinagawa
Developer, SparkSearch.org

Spark Search

Taka Shinagawa — sparksearch.org

October 16, 2015

Speaker's Background

SF-based Analytics Engineer developing "Spark Search" as a personal project

- Data Mining, Analytics, NLP, Machine Learning for finance, online advertising, e-commerce
- Web/Mobile/Social App Development (Ruby on Rails, JS)
- OS Kernel, Network Security, Applied Crypto, OpenSSL
- Cross-Language Information Retrieval– Research Project for TREC

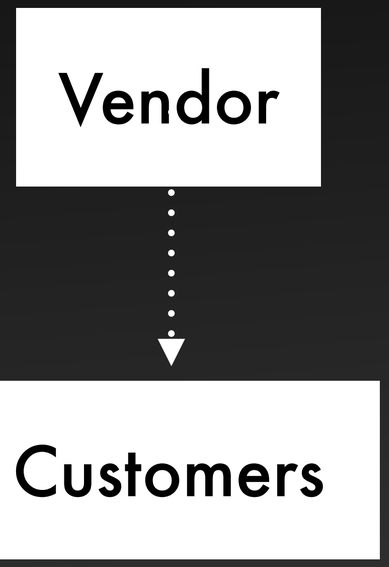
At Fortune 500 corporations and multiple early & late-stage startups as well as academic research

Audience Background?

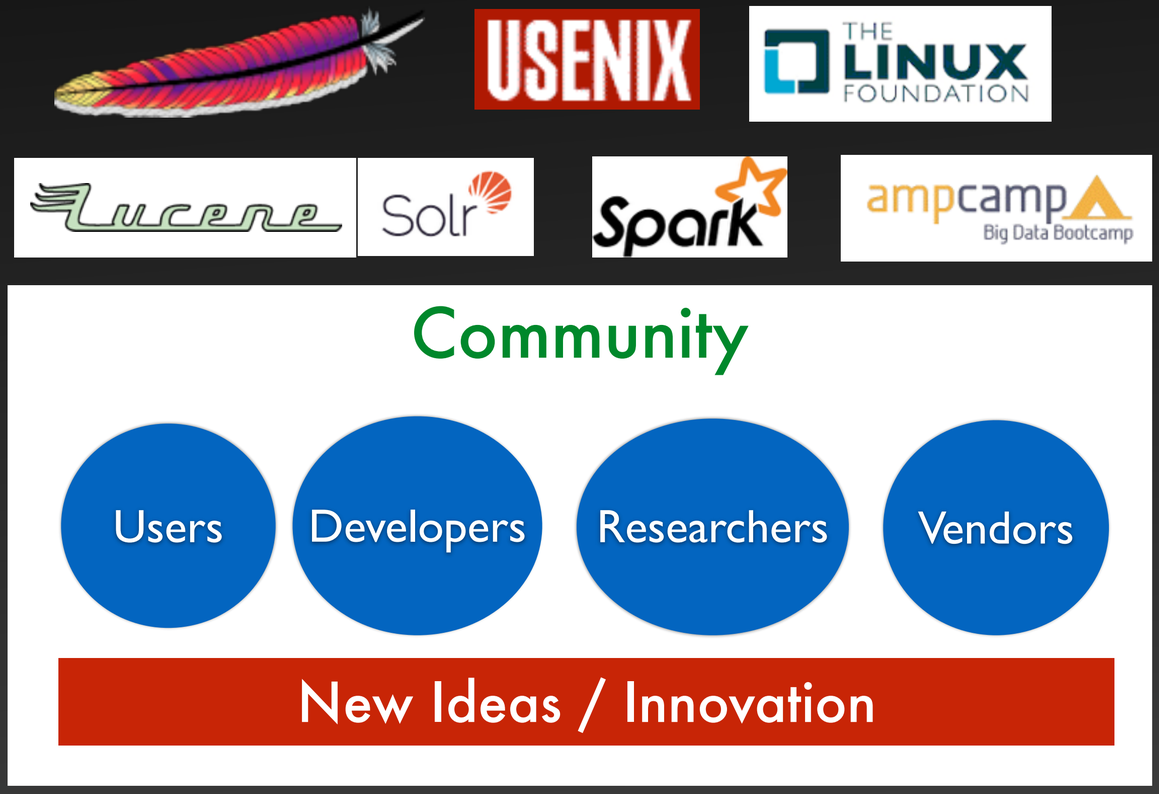
- Big Data – Spark, Hadoop or ?
- Search – Lucene, Solr, Elasticsearch or ?
- Language – Java, Scala, Closure, Python, Ruby, C/C++ or ?

Community Matters!

1961 - 2002 – DECUS (Digital Equipment Computer Users' Society)
1992 - 2010 – Microsoft PDC



1947, 1947 – IEEE Computer Society, ACM
1975 – USENIX
1999 – Apache Software Foundation
2007 – Linux Foundation

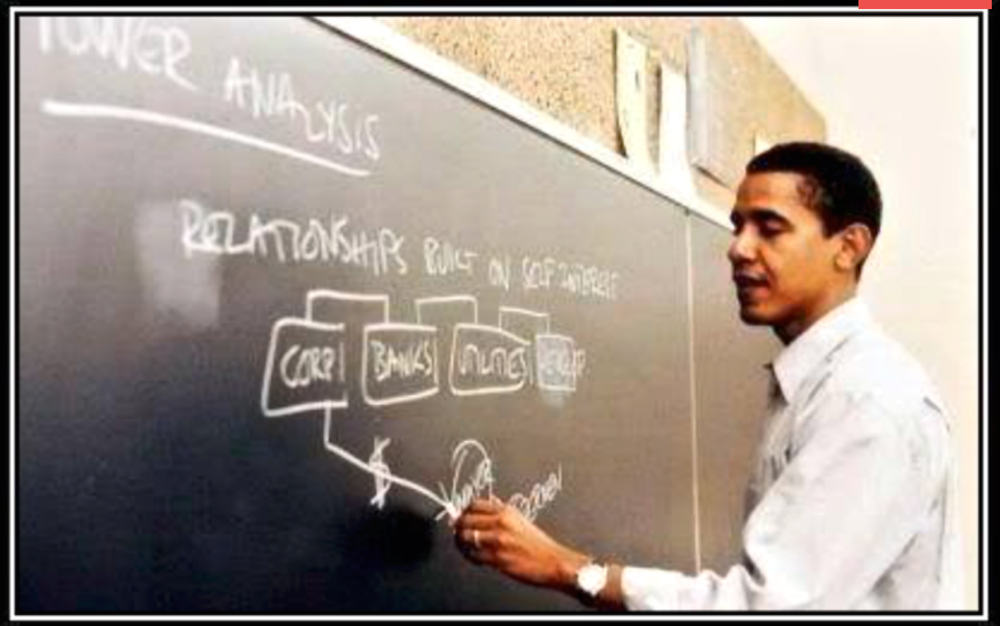


Calling for Your Participation

DEVELOPERS

Developers, developers, developers, developers!
Developers, developers, developers, developers!
Developers, developers, developers! developers!

http://www.intomobile.com/wp-content/uploads/2011/04/developers-ballmer.jpg



COMMUNITY ORGANIZER

~~A fake job generally used to fill resumes that would normally be empty~~

http://tomfernandez28.com/2015/05/12/community-organizer/

Today's Topic –  Spark Search

Spark Search is:

- Spark + Lucene integration in development in Scala ("Spark Search" = Lucene+Spark in this presentation)
- My personal project
- For "offline transactions" such as interactive search, analytics and machine learning (feature engineering/extraction)
- NOT for real-time search and "online transactions" such as web service & application backend
- index = Lucene index = an index = indexes = indices (for my convenience)

Outline

1. Motivation & Background
   - Big Data Search Challenge
   - About Spark
2. Spark Search
   - Architecture
   - Operations
   - Performance

Q&A

1. Motivation & Background

1.1. Big Data Search Challenge

Lucene is Very Useful for Data Scientists, too

- For small data, sequential text search or string pattern matching is fast enough. This doesn't work for big data
- With Lucene index, we can perform full-text search on big data much more quickly
- Lucene can provides ranking scores, text similarities, faceting as well as NLP-related stuffs (e.g. language analyzer, n-gram) for analytics & ML
- Lucene index is basically a matrix of documents (vectorization is already done)

Challenges for Data Scientists

- However, many data scientists and analytics experts are NOT Lucene, Solr, Elasticsearch experts
- Learning curve for non-search specialists (Solr 5 has become easier :-)
- For big data, many clients for Solr and ES are not very scalable. Problems with millions of documents
- Data scientists wouldn't set up a large computer cluster just for search

Challenges with Indexing Big Data

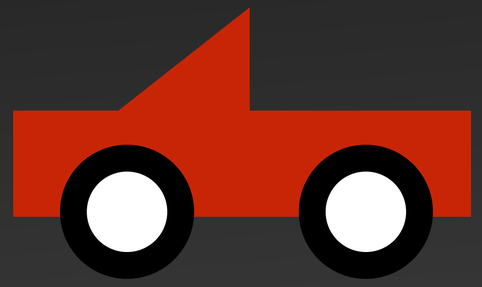
- Lucene is just a library without built-in distributed computing tools and front-end. It doesn't scale and hands-on tools are not available. That's why Solr and ES exist
- Solr/ES are not as easy as we might expect for typical data scientists to index and search big data
- Solr/ES connectors with Spark are helpful, but
  - the network connection can be the bottleneck easily
  - the best data locality requires two clusters on the same nodes

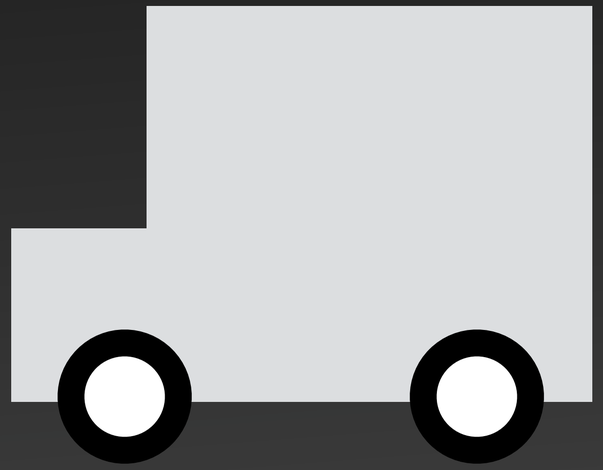
Hypothesis from My Past Experience

Lucene is faster on a single system without parallelism

- My impression: Solr & ES have lots of overhead

Analogy: small lighter car vs large heavier car with the same engine





Experiments

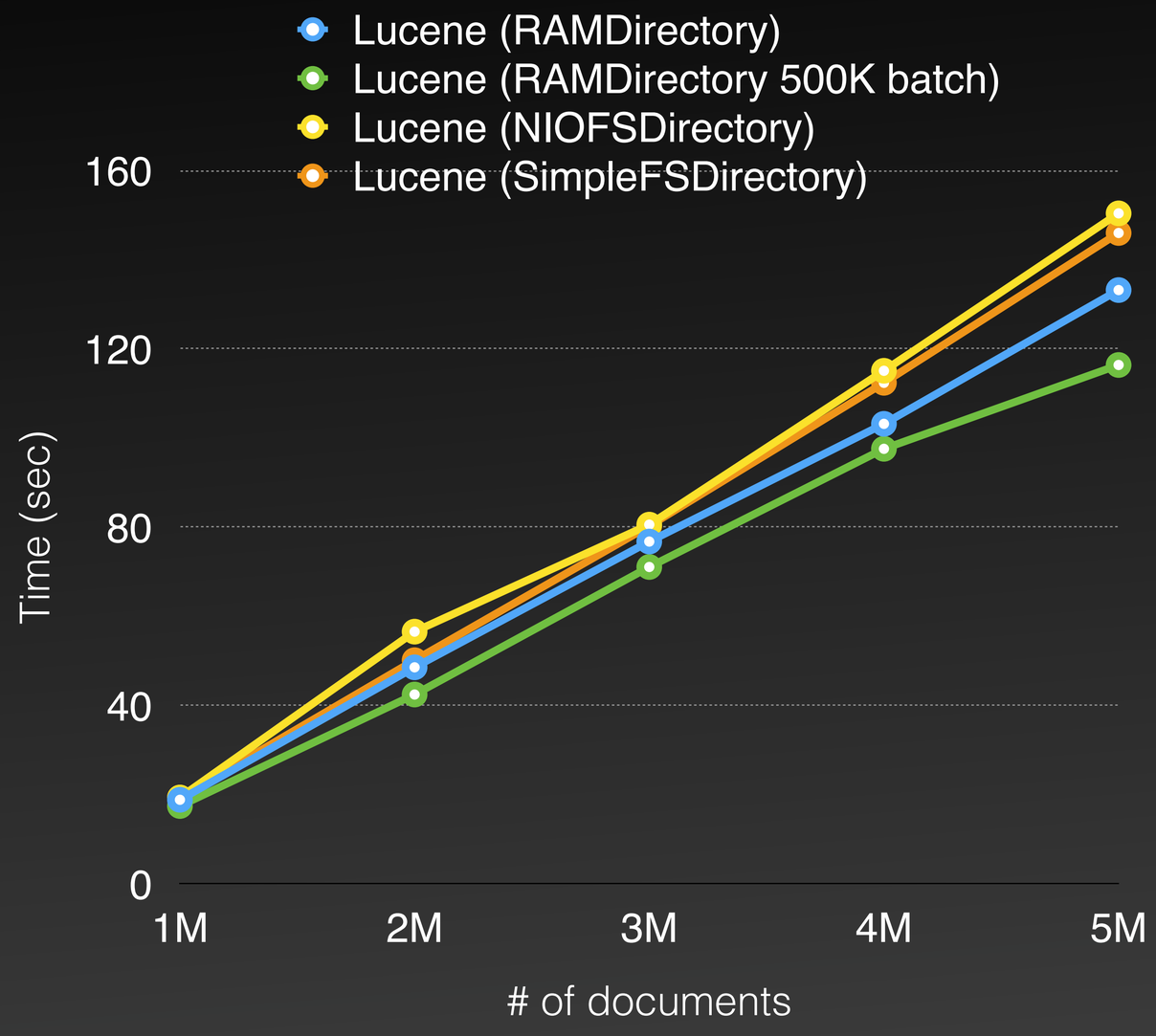
- Goal is to get some rough idea on the performance overhead per single node with Lucene vs Solr vs ES (minimum parallelism on Solr/ES)
  - NOT to measure absolute performance numbers with the most optimized settings
- Indexing millions of short documents (each line of a Wikipedia dump file as a document) with Lucene, Solr and Elasticsearch (no XML parser used)
- The simplest setup – 1 shard, no replication, 1 node (on my laptop), no parallelism at Solr/ES level

Lucene (Setup)

- Lucene 5.3.1, Java8
- 1 Shard, No Replica
- Test with RAMDirectory (memory), SimpleFSDirectory (magnetic disk), NIOFSDirectory (magnetic disk)

Lucene (Result)

- Linearly Scalable on a single node
- Performance between RAMDirectory and FSDirectory is NOT significant– the gap is smaller than I expected
- Batch Indexing doesn't improve performance much (unline Solr and ES)

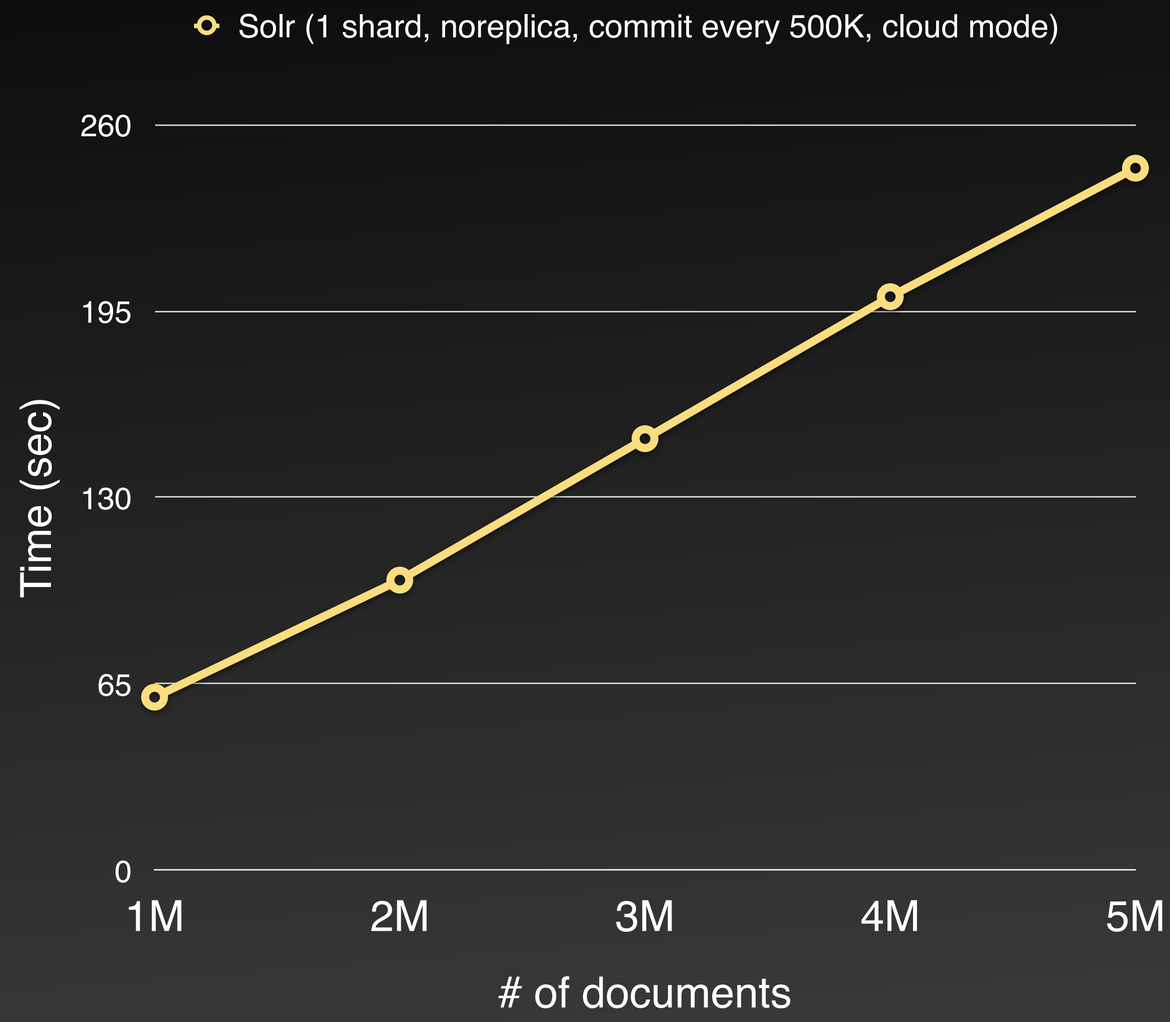


Solr (Setup)

- Solr 5.3.1, Java8
- SolrJ library
- 1 shard, no replication, 1 node, cloud mode (aka SolrCloud)

Solr (Result)

- Indexing without batching is a bad idea for millions of docs (needed to spend some time to figure this out)
- Indexing with 500K docs batches (need clear documentation about batching and batch sizes)

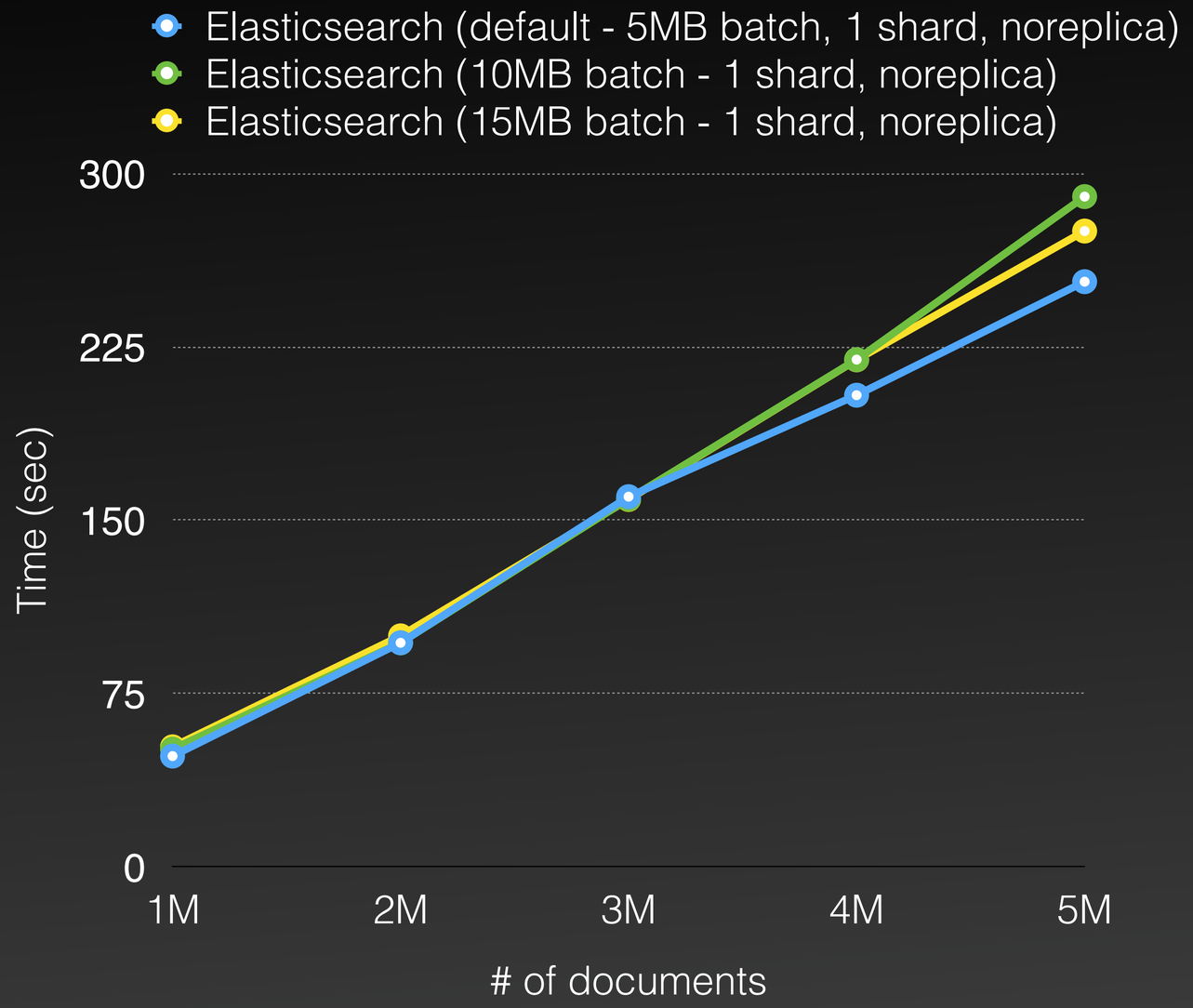


Elasticsearch (Setup)

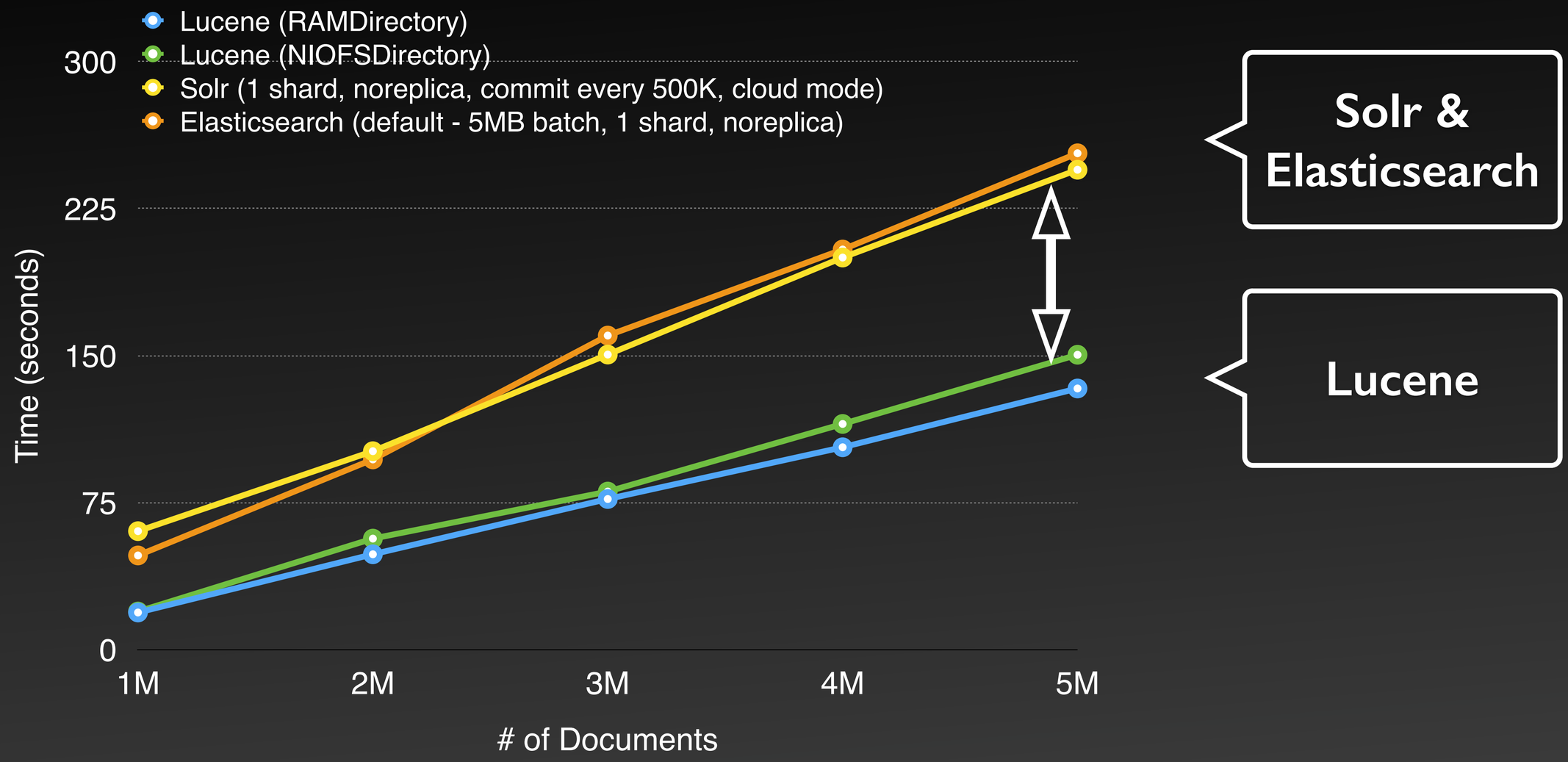
- Elasticsearch 1.7.2 (uses Lucene 4.10.4)
- 1 shard, no replication, 1 node, Transport Client
- Bulk API (https://www.elastic.co/guide/en/elasticsearch/client/java-api/current/bulk.html)

Elasticsearch (Result)

- Bulk Request doesn't scale
- Must use BulkProcessor
- Indexing Performance Tips (https://www.elastic.co/guide/en/elasticsearch/guide/current/indexing-performance.html)
- Various Errors and JVM Exceptions
- Occasional Data Loss (after JVM exception or indexer crash)

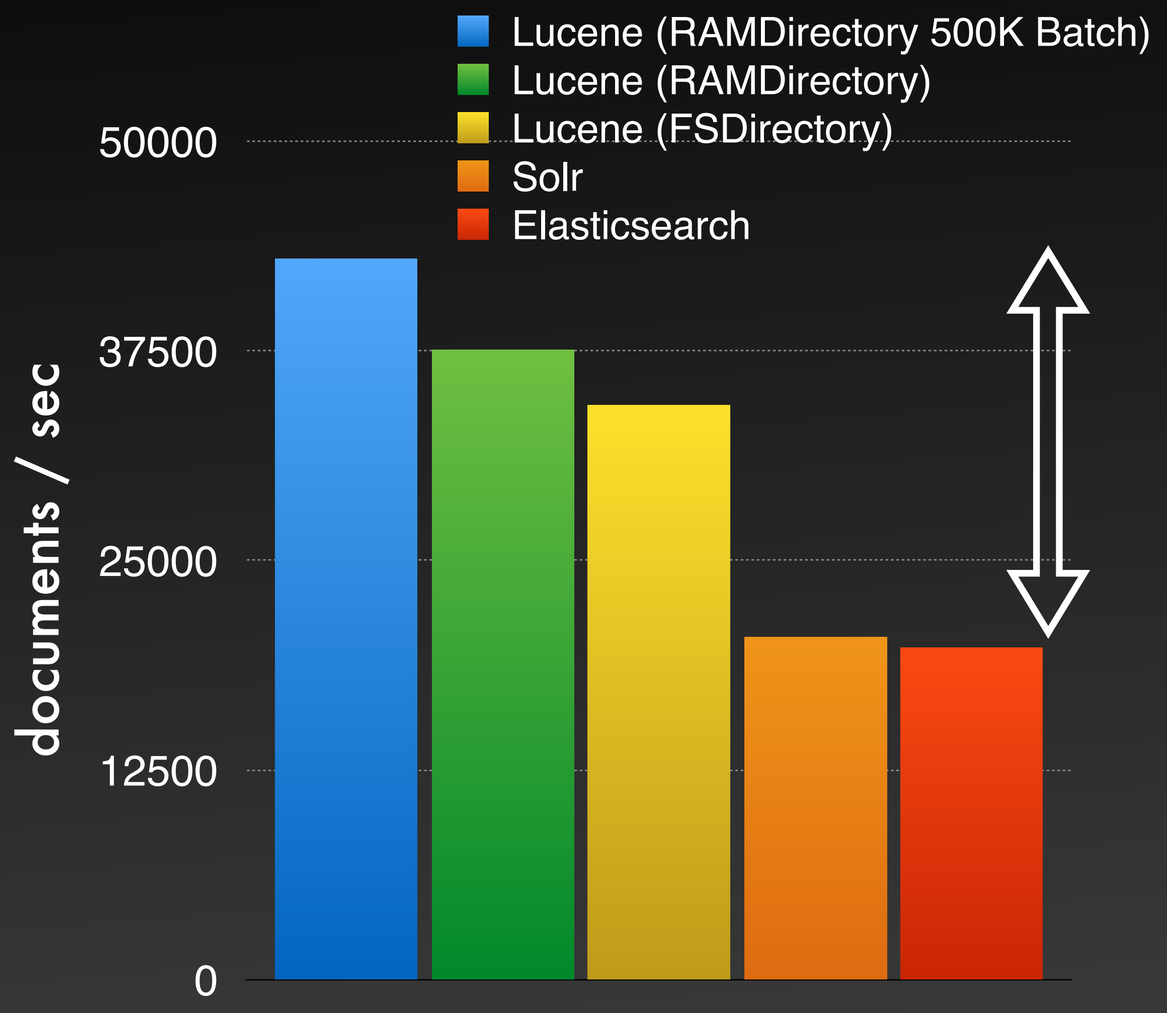


Experiment Results (Indexing locally, minimum parallelism with Solr/ES)



Findings from the Experiment

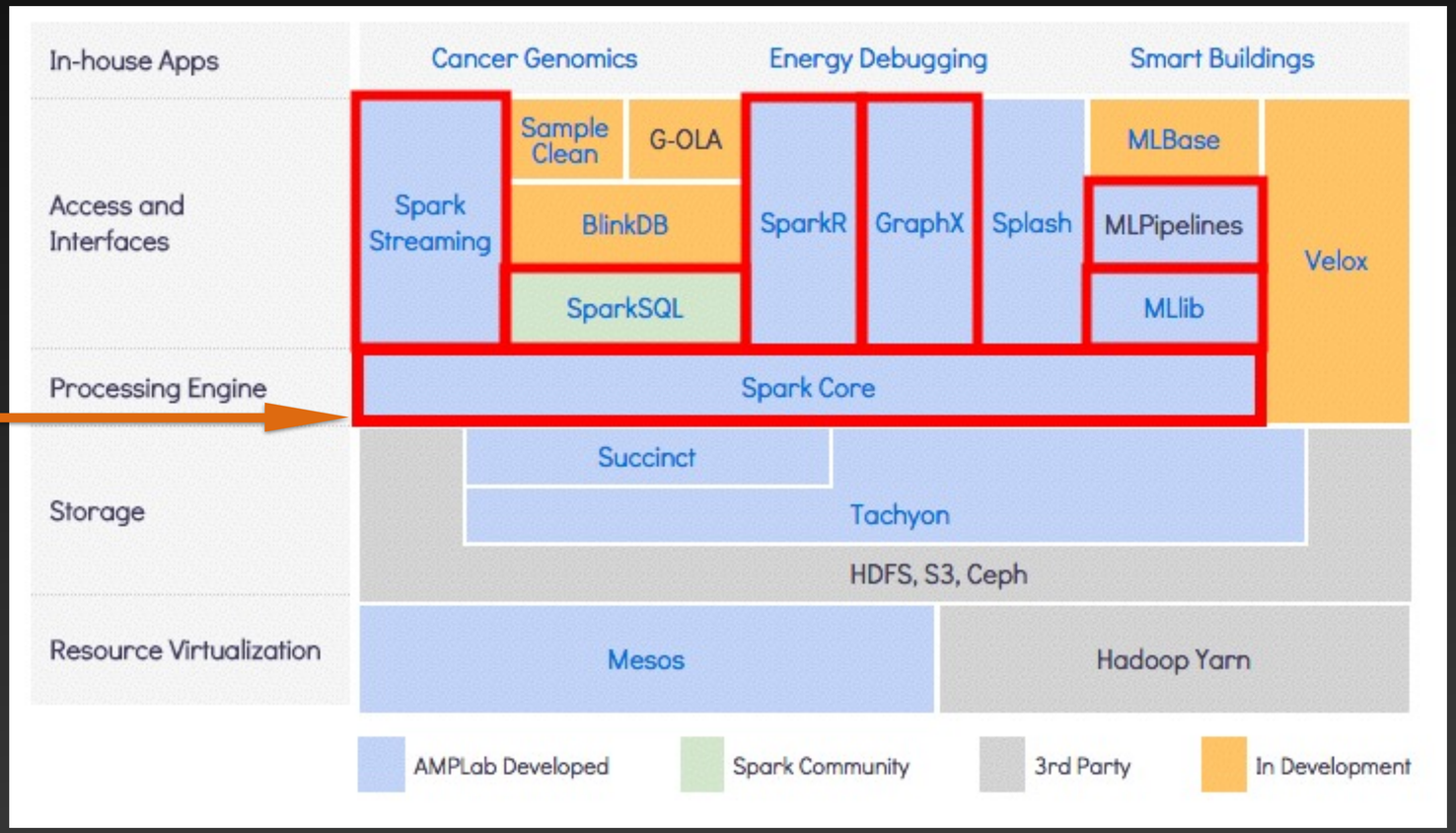
1. Solr and ES are slower than Lucene for indexing on a single machine with minimum parallelism (lots of overhead)
2. Solr and ES require expertise and fine tuning for big data indexing
3. Indexing big data is not as easy as we wish



1.2. Apache Spark

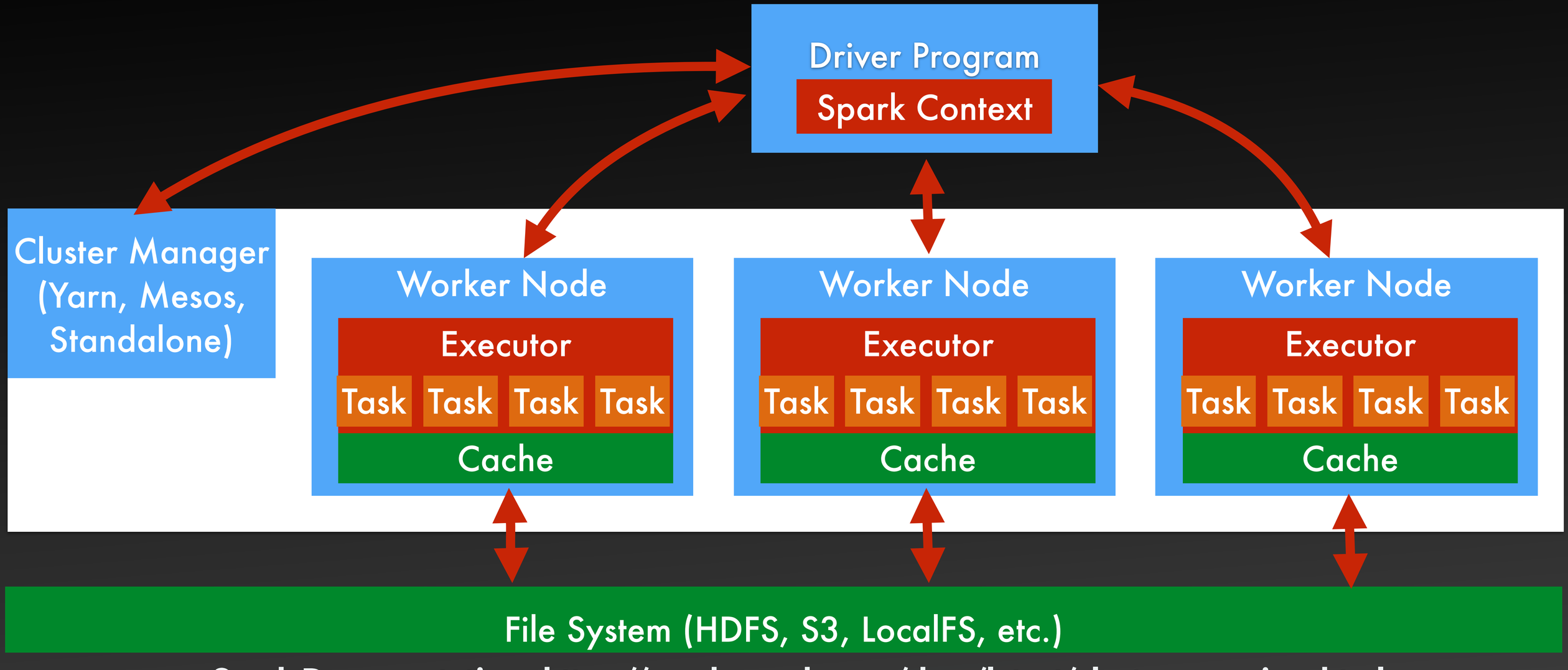
What's Spark – History and BDAS

- "In-memory", distributed general big data processing framework by Matei Zaharia (then at UC Berkeley AMP Lab, now Databricks/MIT). Top-level Apache project since 2014
- The Core Processing Engine in the BDAS (Berkeley Data Analytics Stack) at AMP Lab



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

What's Spark – Distributed Computing Architecture



Spark Documentation https://spark.apache.org/docs/latest/cluster-overview.html

Spark Key Ideas

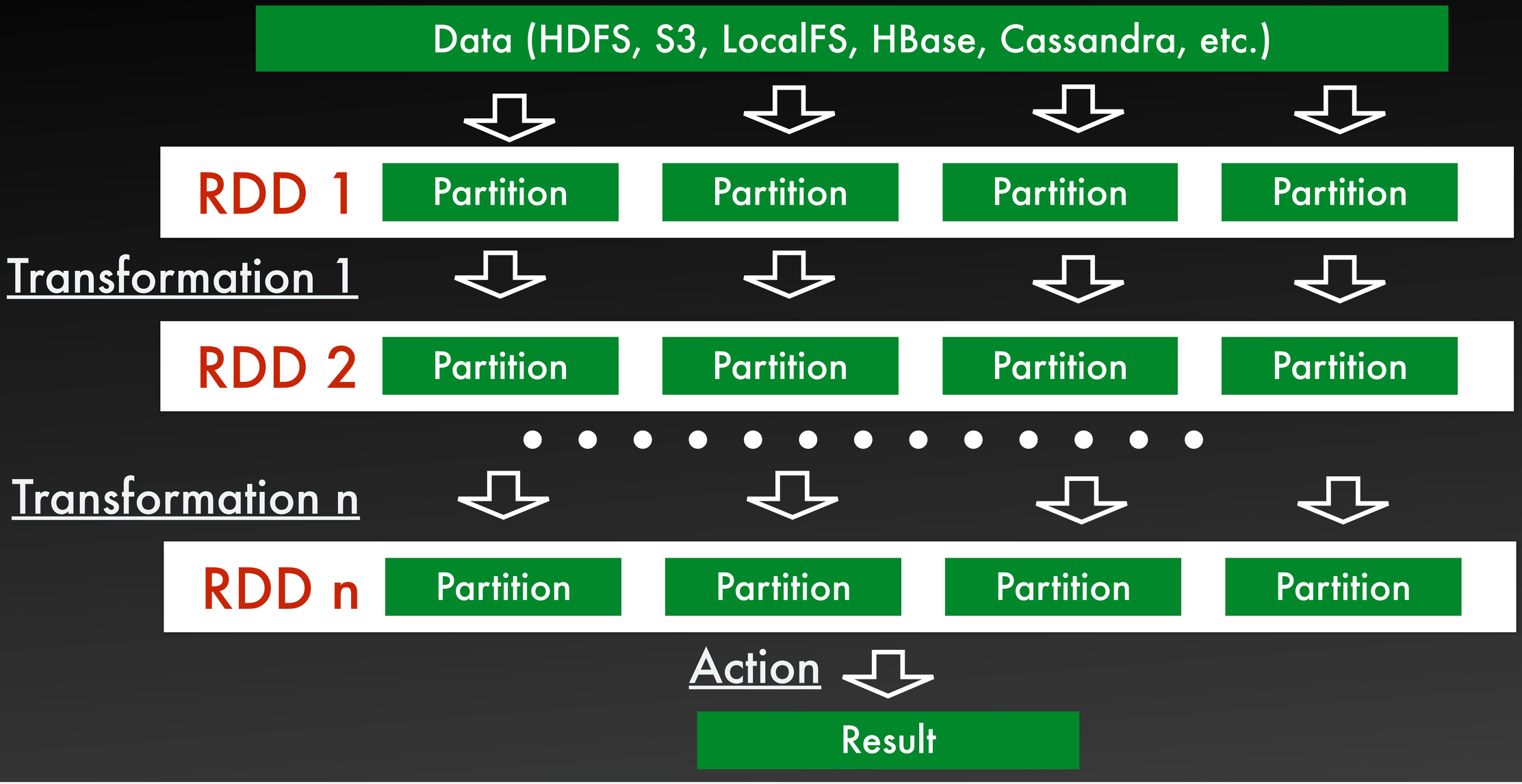
- RDD – Parallel dataset with partitions
- Transformations – Lazy operations to build RDDs from other RDDs
- Action – Return a result or write it to storage

- Tasks – Fundamental unit of work
- Stage – Set of tasks that run in parallel
- DAG – Logical graph of RDD operations

- Spark Context – Main entry point to access Spark functionality

http://ampcamp.berkeley.edu/wp-content/uploads/2012/06/matei-zaharia-part-1-amp-camp-2012-spark-intro.pdf (by Matei Zaharia at AMP Camp 1, 2012)
https://spark-summit.org/2013/talk/wendell-understanding-the-performance-of-spark-applications/ (by Patrick Wendell at Spark Summit 2013)

What's Spark – Data Computation Flow



Spark vs Hadoop MapReduce

- Spark is much faster than Hadoop's MapReduce, especially for iterative computations and interactive analytics
- Hadoop MapReduce uses disks to store data— even intermediate data
- Spark APIs are easier to program than MapReduce APIs

Spark Dataframe

- Distributed collection of data with schema/columns (RDD + Schema) – used to be called schemaRDD (Spark 1.0 - 1.2) and renamed to Dataframes since Spark 1.3
- Similar to Dataframes with R and Python Pandas
- Integrated with Spark SQL
- Dataframe can read data from external sources (e.g. JSON, Parquet, CSV, Solr, Cassandra) through Data Sources API

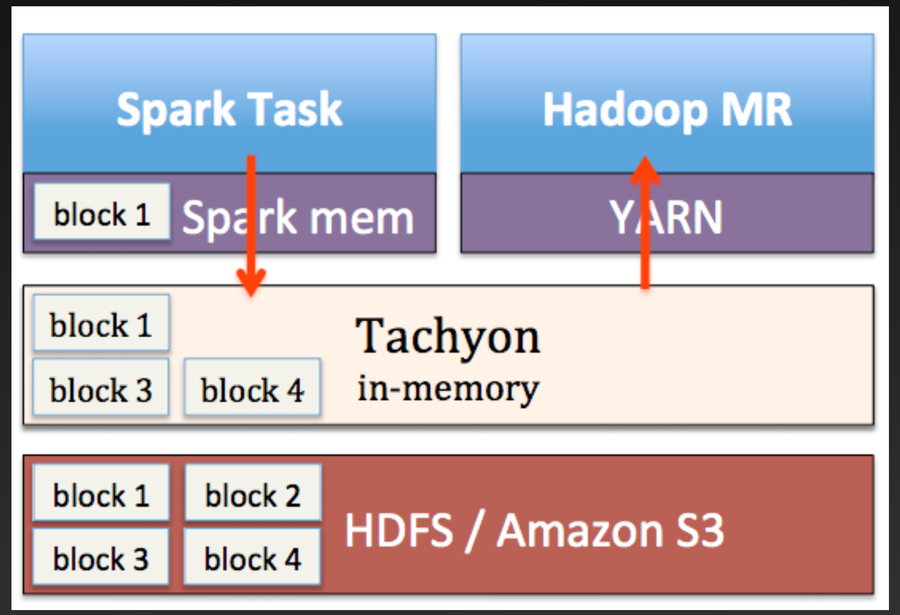
  http://spark.apache.org/docs/latest/sql-programming-guide.html

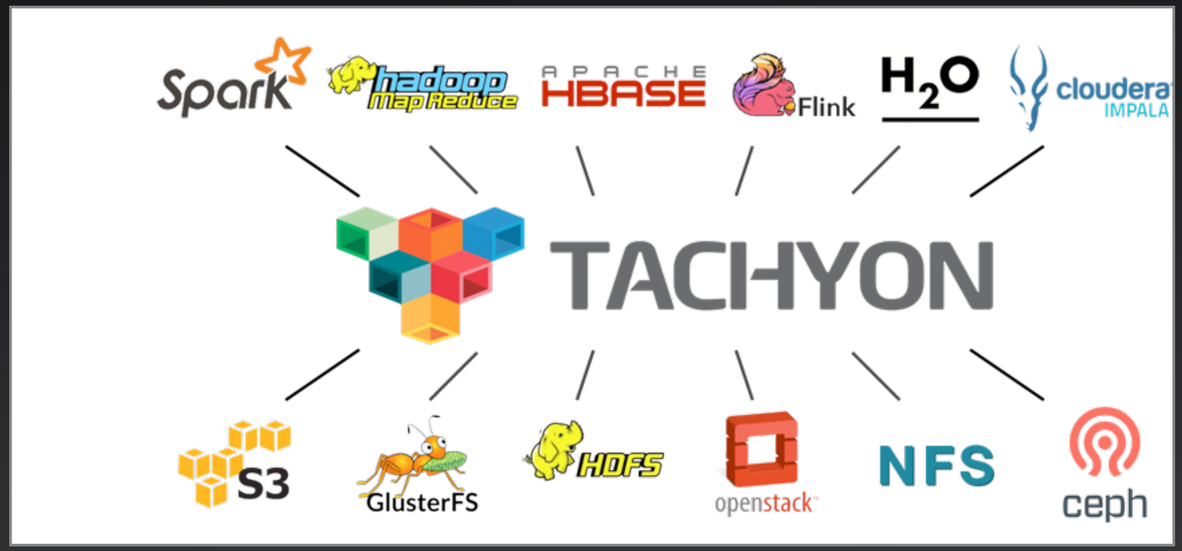
  https://databricks.com/blog/2015/02/17/introducing-dataframes-in-spark-for-large-scale-data-science.html

Tachyon – RDD Data Caching & Sharing

- Tachyon is the default distributed off-heap caching store for Spark
- Tachyon is faster than HDFS

www.tachyon-project.org

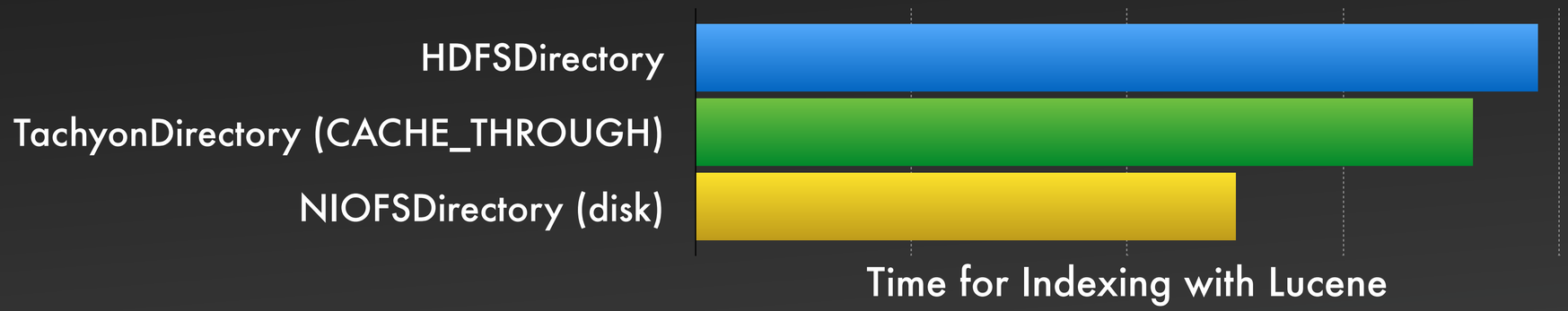




http://www.cs.berkeley.edu/~haoyuan/talks/Tachyon_2014-10-16-Strata.pdf

TachyonDirectory – Tachyon for Lucene

- I have implemented TachyonDirectory as part of Spark Search and have been optimizing
- Tachyon Write Type matters (MUST_CACHE, CACHE_THROUGH, THROUGH, ASYNC_THROUGH)
- Lucene Indexing (IndexWriter) operations are different from simple caching
- Still fine-tuning TachyonDirectory implementation (buffer size, etc.)



Many Reasons to Use Spark

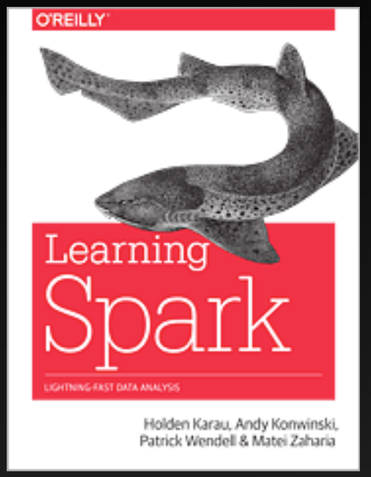
- Fast and Lightweight
- Expressive, Clean APIs
- Easy to Use
- Command Line Interface with Scala & Python
- Unified Framework with MLlib, Streaming, Dataframe/SQL, GraphX for the entire data pipeline
- Thriving User & Developer Community
- Ecosystem (integration with many technologies)
- Good Documentation
- and more

Why Spark? – Personal View

Spark is the best platform for scalable machine learning and analytics

- R / RStudio (< 2GB)
- Python / Pandas / Scikit-Learn (< 10GB)
- Mahout [MapReduce] (slower)
- Spark (any size & fast, Scala/Java/Python/R support)

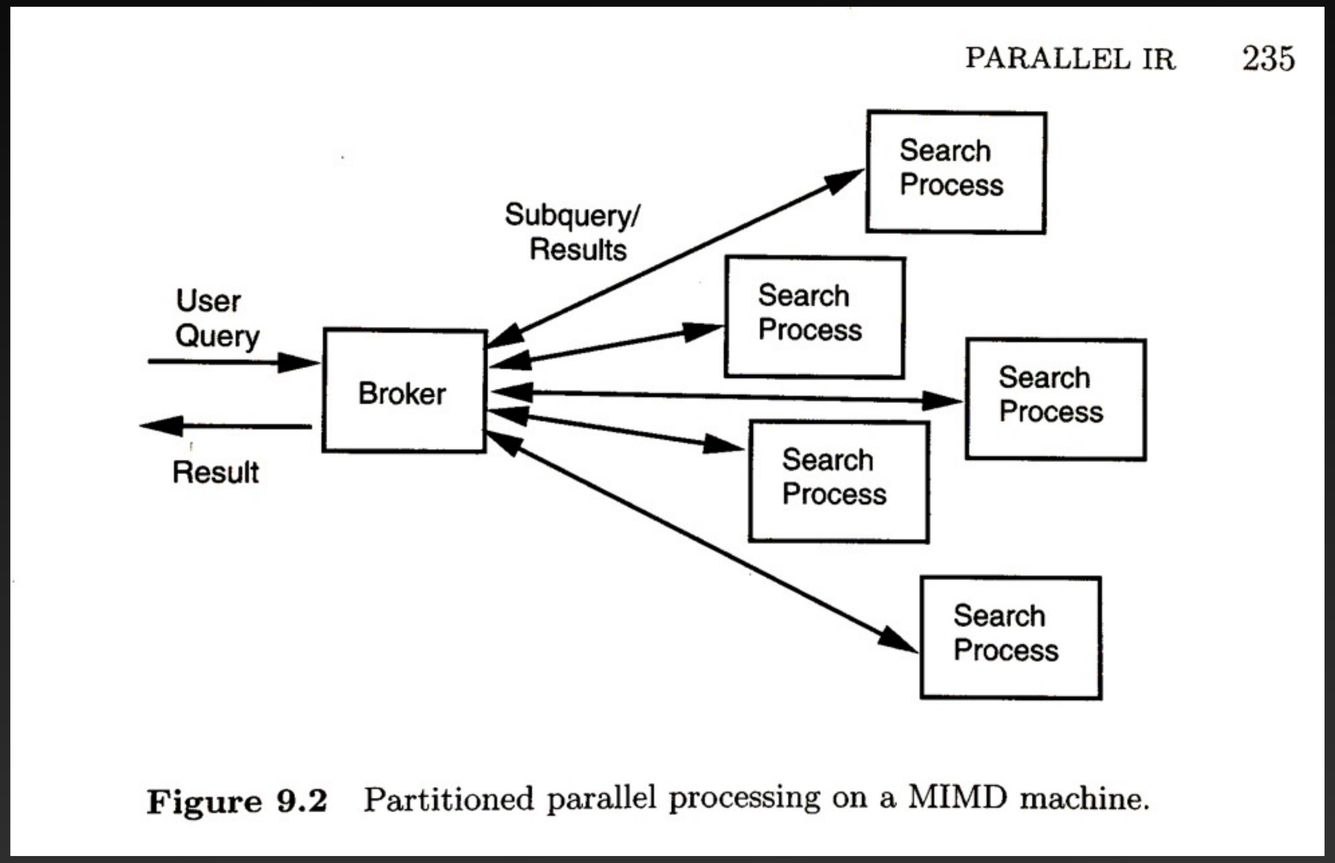
More on Spark



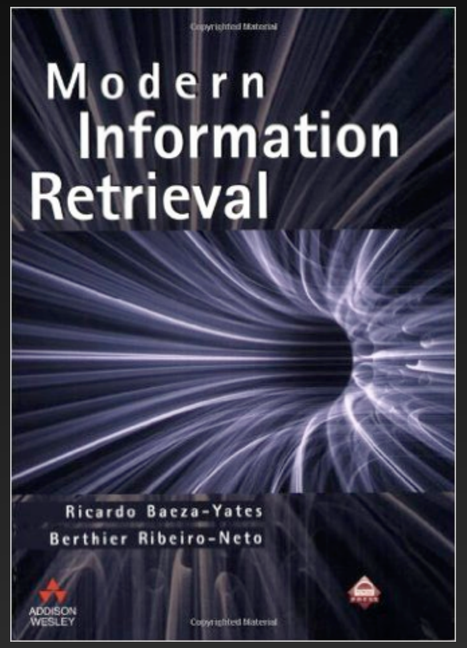
- Spark Documentation (http://spark.apache.org/docs/latest)
- UC Berkeley AMP Camp (http://ampcamp.berkeley.edu)
- Spark Summit (https://spark-summit.org)
- Spark Developer Resource at Databricks (https://databricks.com/spark/developer-resources)
- "Learning Spark" (http://shop.oreilly.com/product/0636920028512.do)
- Spark Summit Europe 2015 (Oct 27-29, Amsterdam)

2. Spark Search

Parallel IR Principle

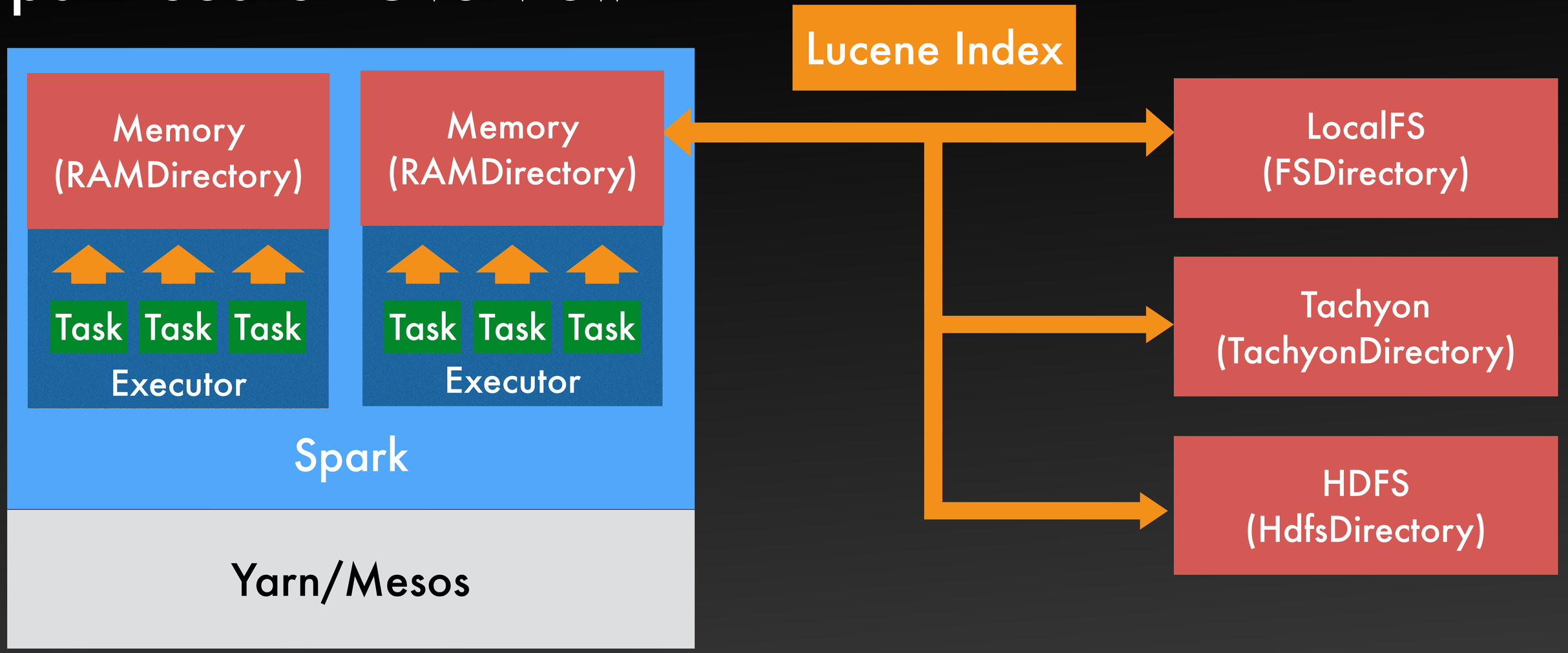


Chapter 9 "Parallel and Distributed IR"
by Dr. Eric Brown at IBM Research

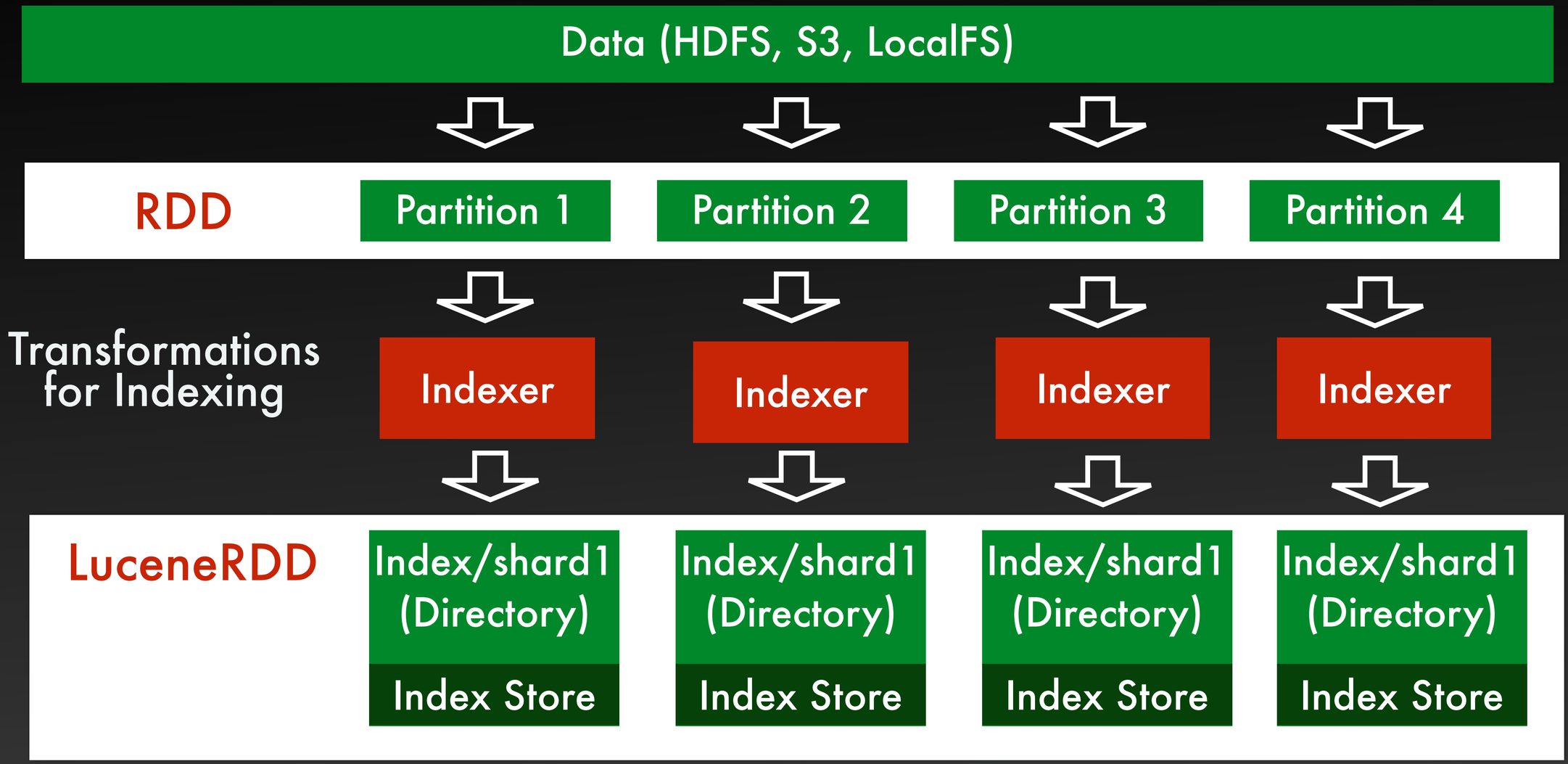


"Modern Information Retrieval"
(ISBN:020139829X)
published in 1999

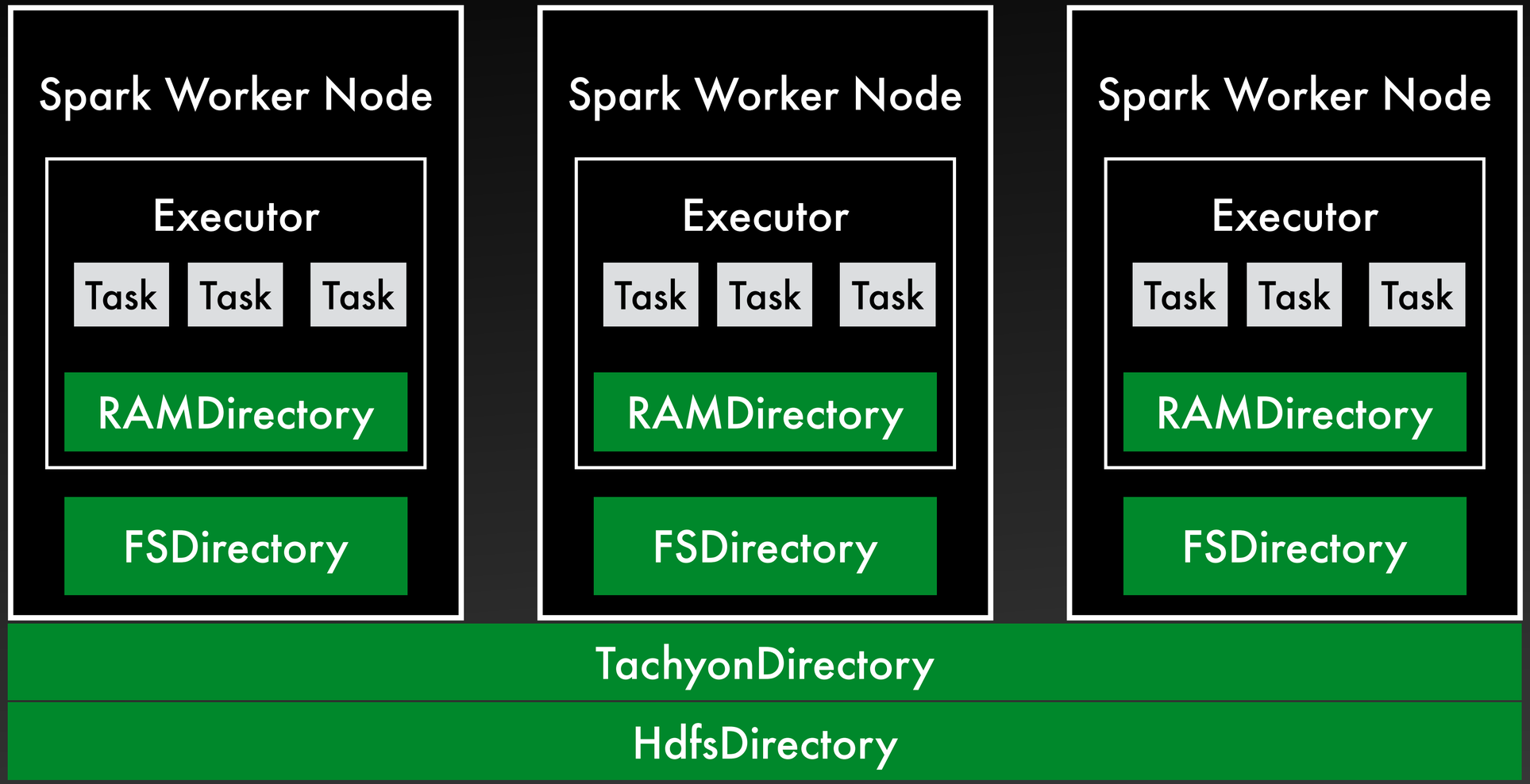
Spark Search Overview



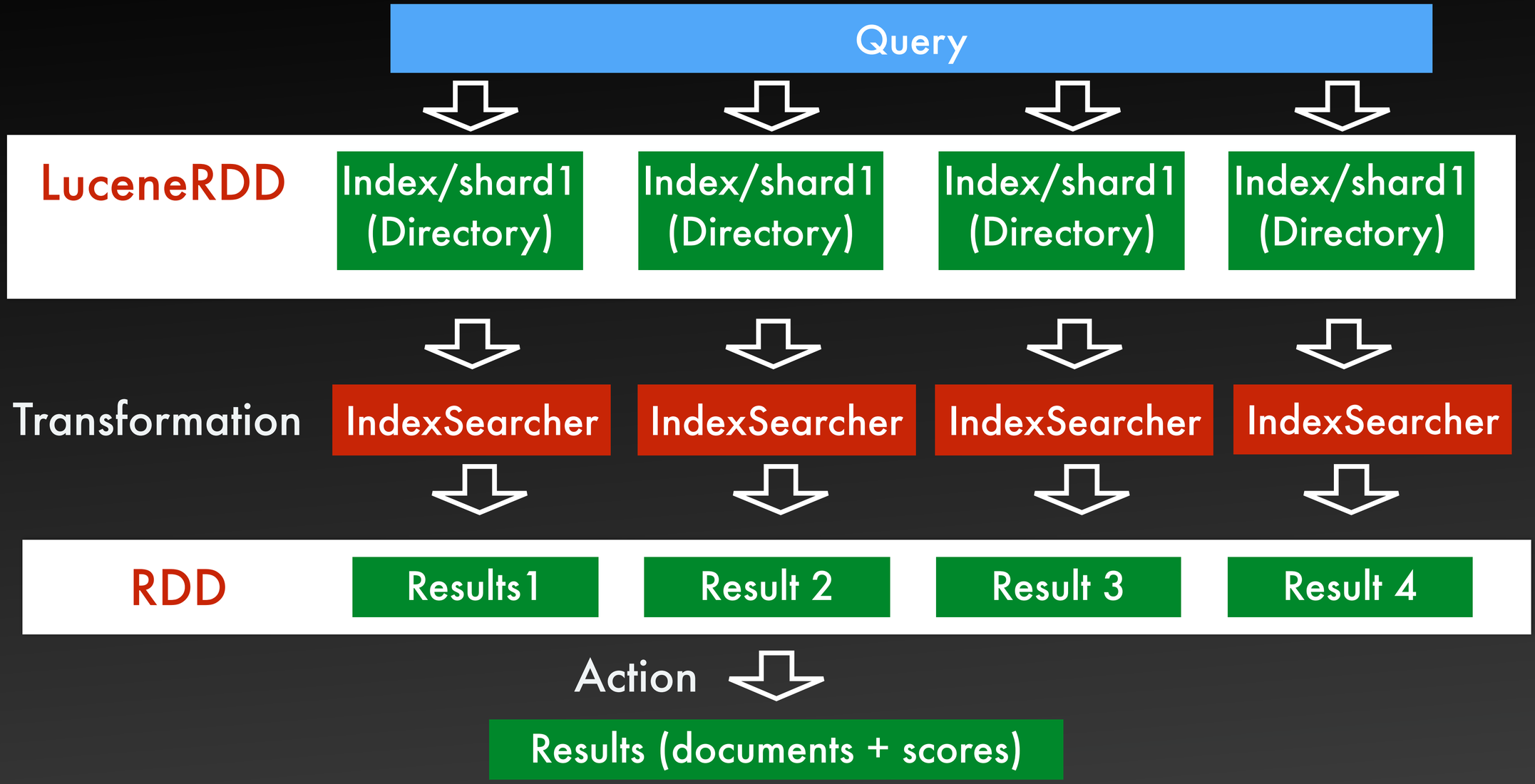
Spark Search – Indexing



Spark Search – Index Store Types



Spark Search – Search



Indexing and Querying – RDD (API Preview)

Indexing

```
val rdd = sc.textFile("/tmp/inputData.txt")
val luceneRDD = LuceneRDD(rdd).index
```

Querying

```
val count = luceneRdd.count("*:*")
val results = luceneRdd.search("body:lucene")
val docs = results.take(100)  // get top 100 docs
val docsAll = results.collect()  // get all docs
```

!!! These APIs are pre-release versions. Subject to change

Indexing and Querying – Dataframe (API Preview)

Indexing

```
val df = sqlContext.read.json(“/tmp/inputData.json")
val luceneTableRDD = LuceneTableRDD(df, "/tmp/luceneIndex", "simplefs").index
```

Querying

```
val counts = luceneTableRdd.count(“body:lucene”)
val results = luceneTableRdd.search(“body:lucene”)
val docs = results.take(100)  // get top 100 docs
val docsAll = results.collect()  // get all docs
```

!!! These APIs are pre-release versions. Subject to change

Indexing and Querying – Data Sources API

Indexing

```
val df = sqlContext.read.json(“/tmp/inputData.json")
df.write.format("sparksearch").option("storeType", “niofs”).save(“/tmp/luceneIndex")
```

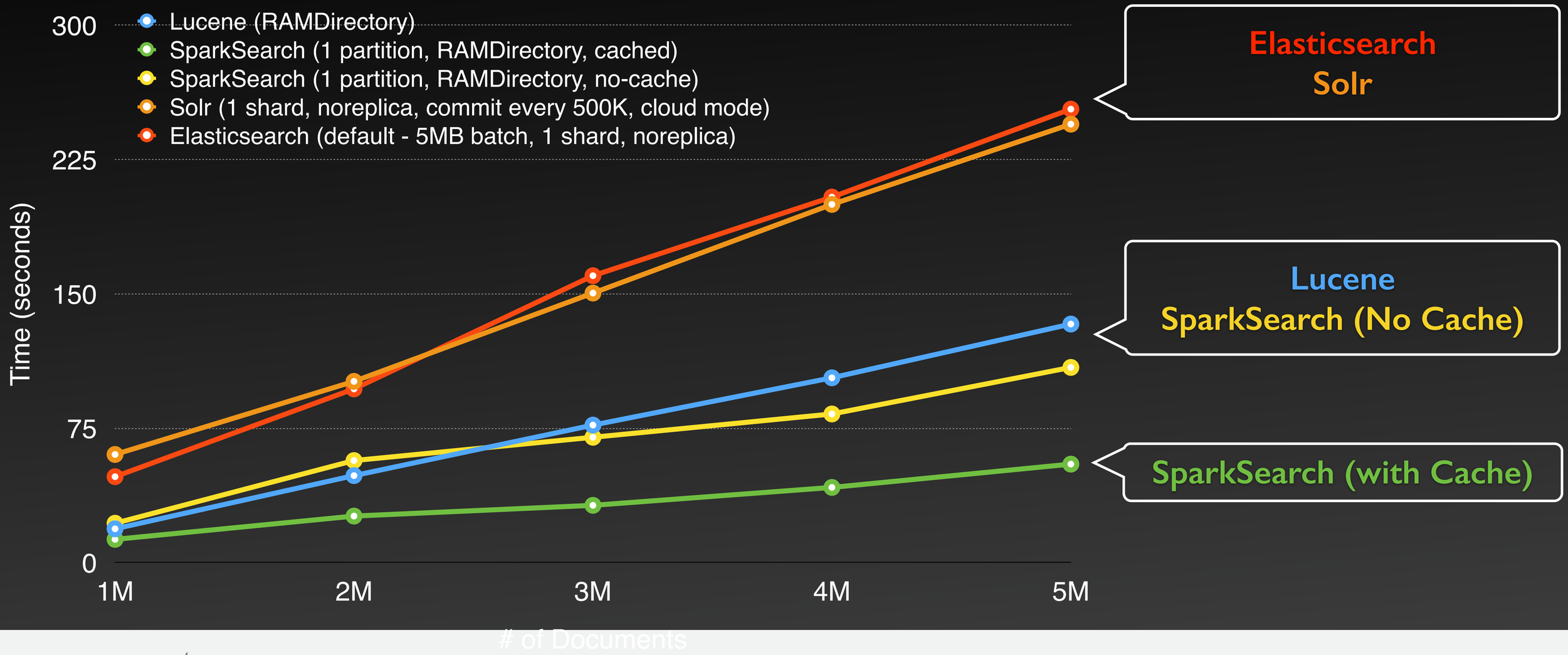
Querying

```
val dfResults = sqlContext.read.format(“sparksearch").option("query",
                "body:lucene").option("storeType", "niofs").load("/tmp/luceneIndex")
```

!!! The API parameters are pre-release versions. Subject to change

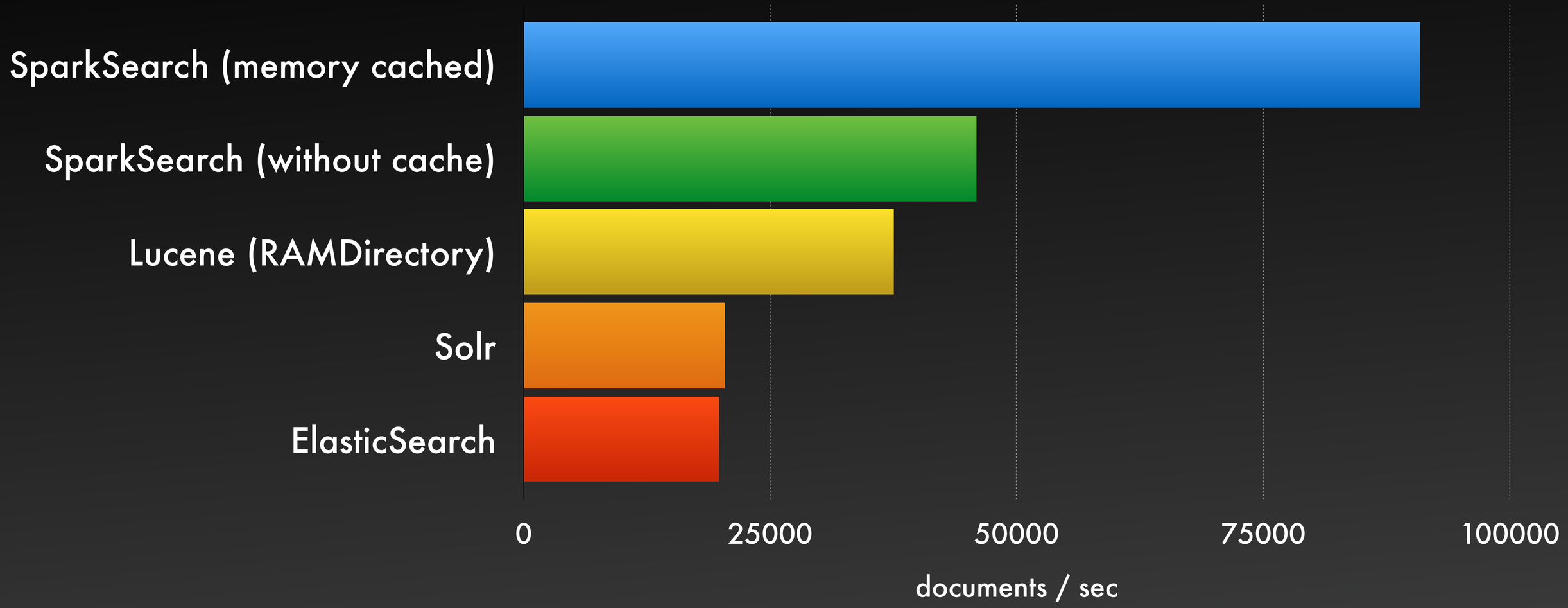
Spark Search vs others – Indexing

(small 5M docs, single machine, minimum parallelism on Solr/ES, 1 partition with SparkSearch)



SparkSearch Indexing Speed

(small 5M docs, single machine, minimum parallelism on Solr/ES, 1 partition with SparkSearch)



SparkSearch Query Performance

- Query time on shards can be predicted with their Lucene index sizes
- Better to have even/similar shard sizes
- One big shard's query time affects overall query performance

3. Scalability Test

Spark Search for 100GB Data on AWS

How long does it take to index 100GB?

Computing Resources (AWS EC2)

- 8 instances (r3.4xlarge), No VPC
- CPUs: 16 vCPU x 8 = 128 vCPUs
- Memory: 122GB x 8 = 976GB
- SSD Storage: 320GB x 8

Dataset – Public Reddit Comments Corpus

- The Entire Dataset – 1TB- 1.65B comments on Reddit in JSON format
- One JSON record per line
- The Dataset used in this example – 100GB 177.5M (177,544,909) Reddit comments (10% of the entire dataset). Combined multiple files into a single 100GB file

https://archive.org/details/2015_reddit_comments_corpus

Dataset Schema

12 fields

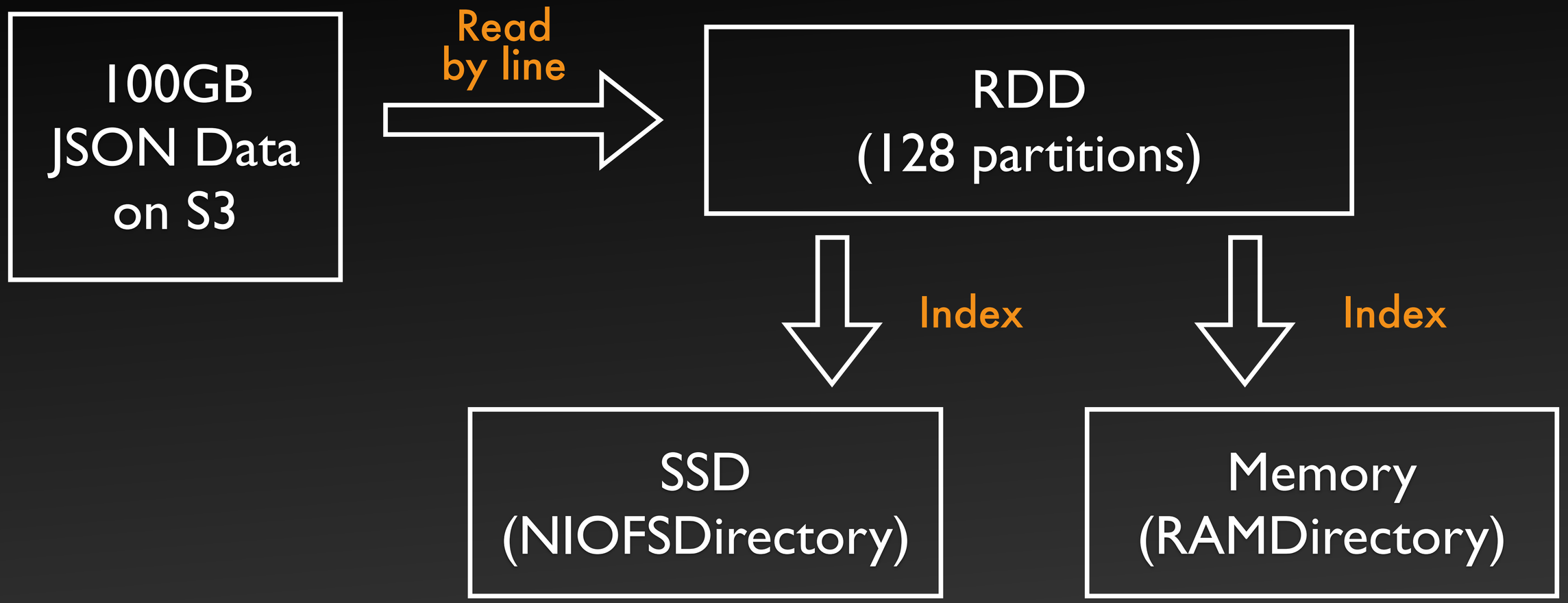
the “body” field stores Reddit comments

```
scala> reddit.printSchema
root
 |-- archived: boolean (nullable = true)
 |-- author: string (nullable = true)
 |-- author_flair_css_class: string (nullable = true)
 |-- author_flair_text: string (nullable = true)
 |-- body: string (nullable = true)
 |-- controversiality: long (nullable = true)
 |-- created_utc: string (nullable = true)
 |-- distinguished: string (nullable = true)
 |-- downs: long (nullable = true)
 |-- edited: string (nullable = true)
 |-- gilded: long (nullable = true)
 |-- id: string (nullable = true)
 |-- link_id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- parent_id: string (nullable = true)
 |-- removal_reason: string (nullable = true)
 |-- retrieved_on: long (nullable = true)
 |-- score: long (nullable = true)
 |-- score_hidden: boolean (nullable = true)
 |-- subreddit: string (nullable = true)
 |-- subreddit_id: string (nullable = true)
 |-- ups: long (nullable = true)
```

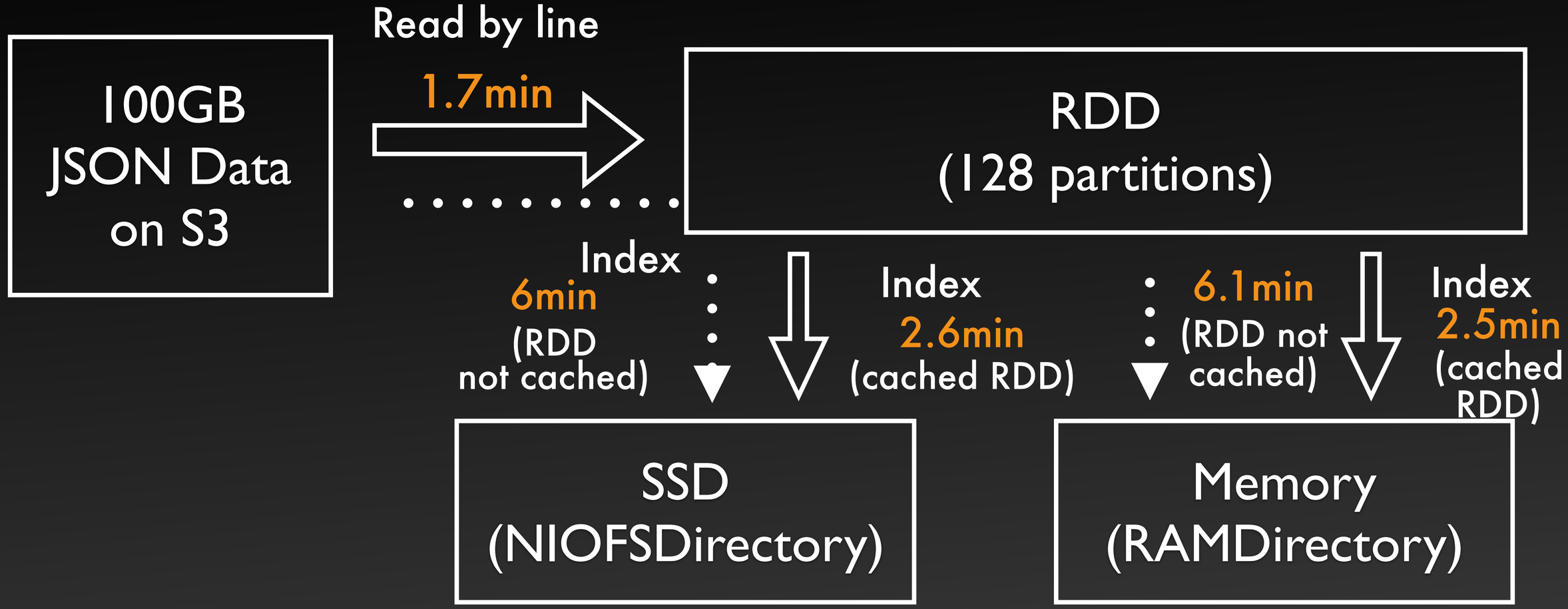
A Sample JSON Record from the Dataset

{“ups”:3,"subreddit":"pics","id":"c0ifz1b","author_flair_text":null,"distinguished":null,
“created_utc":"1262984678","author":"DrK","link_id":"t3_an7yh","author_flair_css_cl
ass":null,“subreddit_id":"t5_2qh0u","controversiality":0,
"body":"Learn to use [Lucene syntax](http://lucene.apache.org/java/2_3_2/
queryparsersyntax.html) or use http://www.searchreddit.com/“,
“score_hidden”:false,“edited":false,"score":3,"retrieved_on"1426178579,
"archived":true,"parent_id":"t1_c0iftql","gilded":0,"downs":0,"name":"t1_c0ifz1b"}

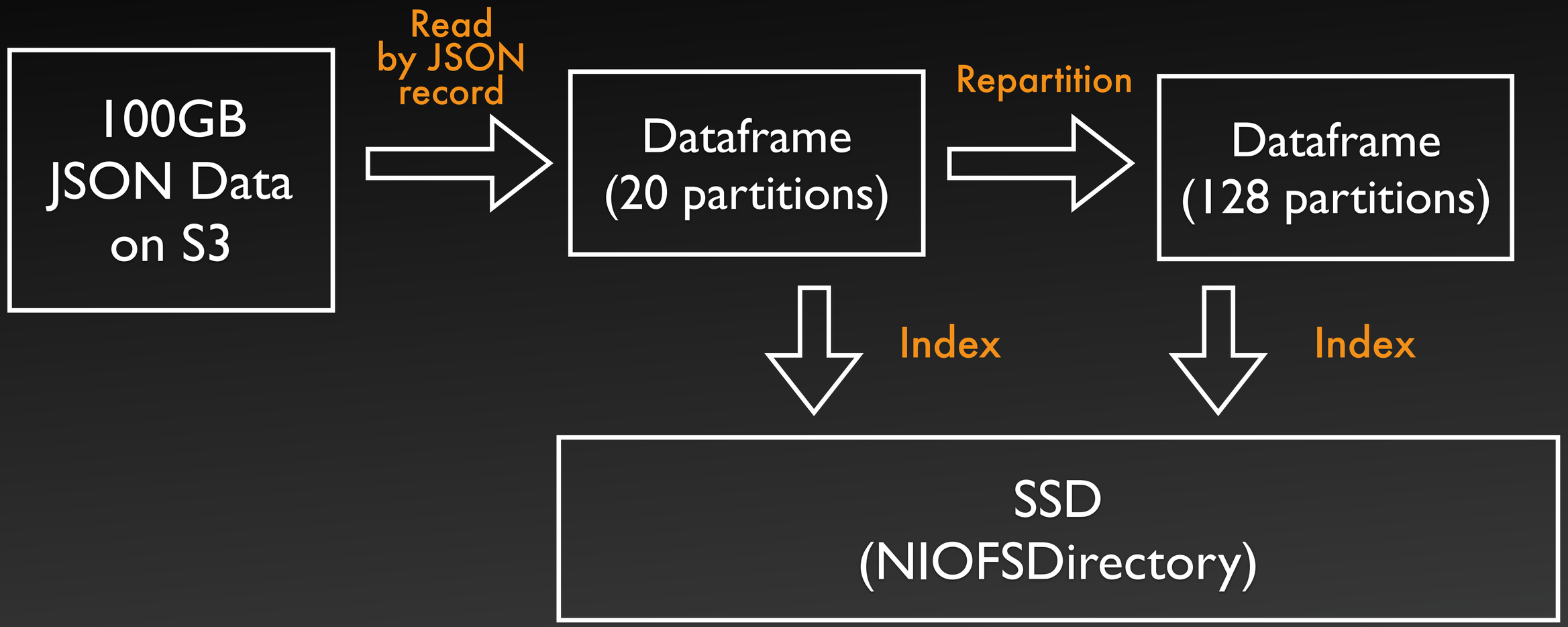
Steps – Indexing 100GB with RDD



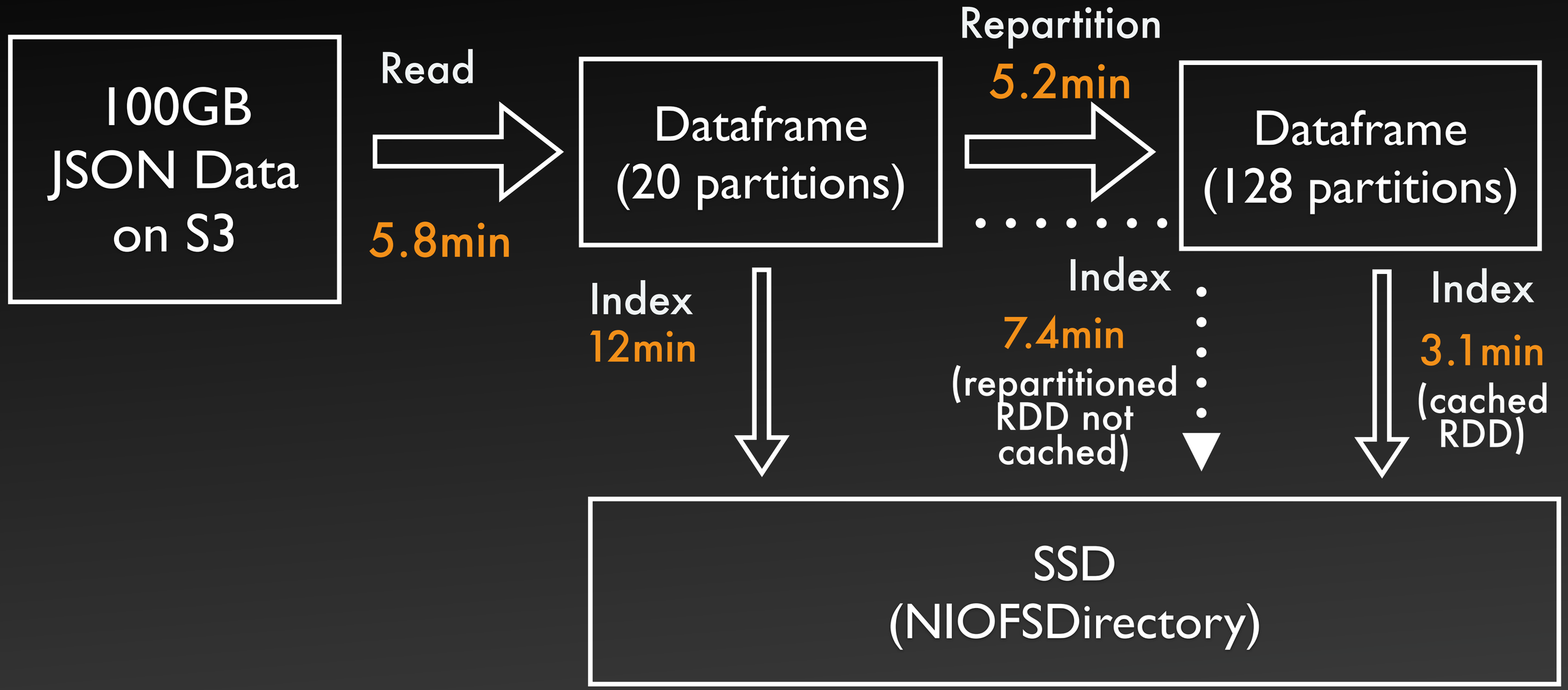
Results – Indexing 100GB with RDD



Steps – Indexing 100GB (177.5M JSON records) with Dataframe



Results – Indexing 100GB (177.5M JSON records) with Dataframe



Findings from the 100GB Scalability Test

- Spark Search is fast and scalable – Spark is a great distributed computing engine for Lucene
- Reliability and performance depends on the store (type of Lucene Directory)
- FSDirectory with SSD is the most reliable store with performance
- RAMDirectory can be improved for better reliability and performance
- Both indexing and query times can be estimated with each partition's input data and Lucene index size

Summary

- SparkSearch (Lucene+Spark) is very fast, efficient and scalable – Spark adds little overhead
- Simple APIs – Just a single line for indexing
- All indexing and search operations can be performed with Spark command line or application.

What's Next

- Finalize the APIs (feedback from users)
- Create examples for analytics and ML as well as for SparkStreaming
- Testing in production size clusters and bigger data (Multi TB datasets)
  - More EC2 instances with VPC
- Any Suggestion? I need your feedback

Next Event

- IBM Datapalooza (Nov 10 -12, 2015 in SF)
  http://www.spark.tc/datapalooza

Email me for discount!

Questions & Feedback

taka@sparksearch.org

Love to hear your feedback, help and ideas – contact me!

Update

@sparksearch

www.SparkSearch.org