IUT Béziers

Samuel Aubertin

04 June 2015

HELP College of Arts and Technology
college of achievers

Spark?

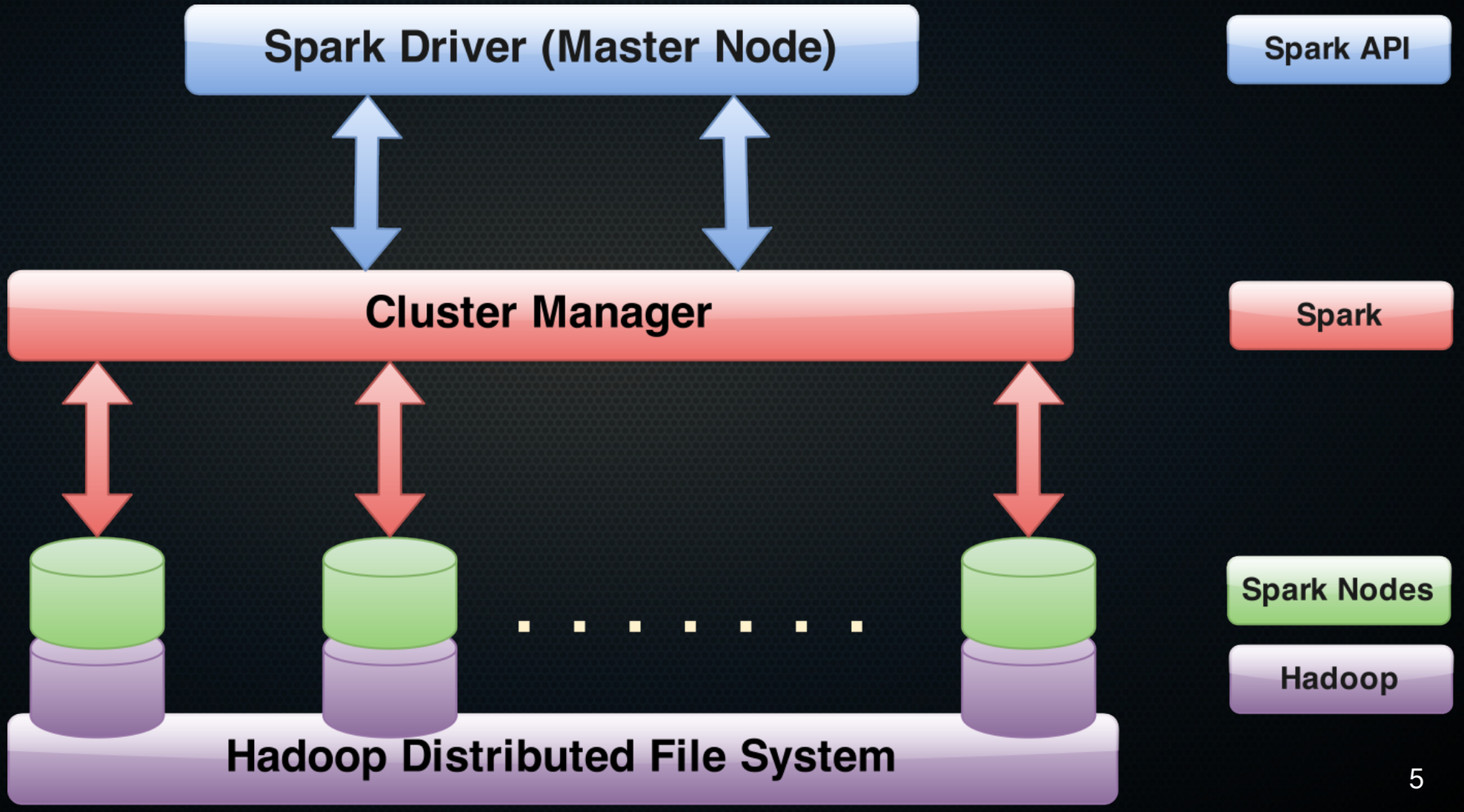
- A big data cluster-computing framework written in Scala
- Open Sourced originally in AMPLab at UC Berkeley
- Highly compatible with Hadoop Storage API
  - → Can run on top of an Hadoop cluster
- Developers can write programs using multiple programming languages

Summary

- Design overview
- Resilient Distributed Datasets
- Spark MapReduce
- Spark SQL
- Examples


Spark
hadoop

Spark – Architecture

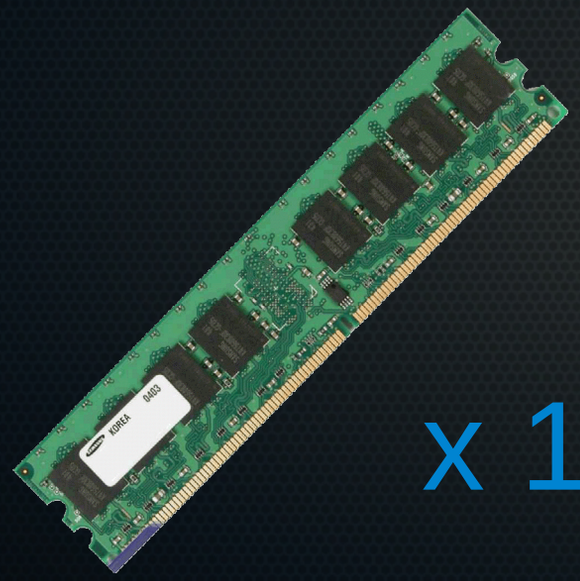


Spark – Motivations

- Hadoop limitations
  - MapReduce is slow due to replication, serialization and disk I/O
  - Inefficient for iterative algorithms and interactive data mining
- We want more *complex* and *fast* apps!
- Efficient primitives are needed for data sharing: **RAM**

Spark – RAM VS HDD

RAM I/O = nanoseconds (10^{-9} s)



<

x 1 000 000

HDD I/O = milliseconds (10^{-3} s)

Nanoseconds ?

- What a billion looks like?
- Let's use the speed of light ≈ 3 x 10^8 m/s
- 1 millisecond (10^{-3} s) ≈ 300 km
- 1 nanosecond (10^{-9} s) ≈ ?

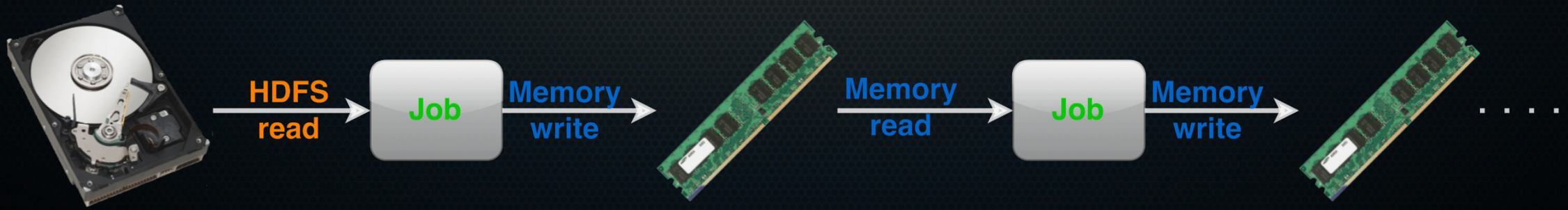
30 cm !

- So you better *get* your nanoseconds !

Spark – Data Motion

- Hadoop MapReduce

- Spark



Spark – RAM Computing

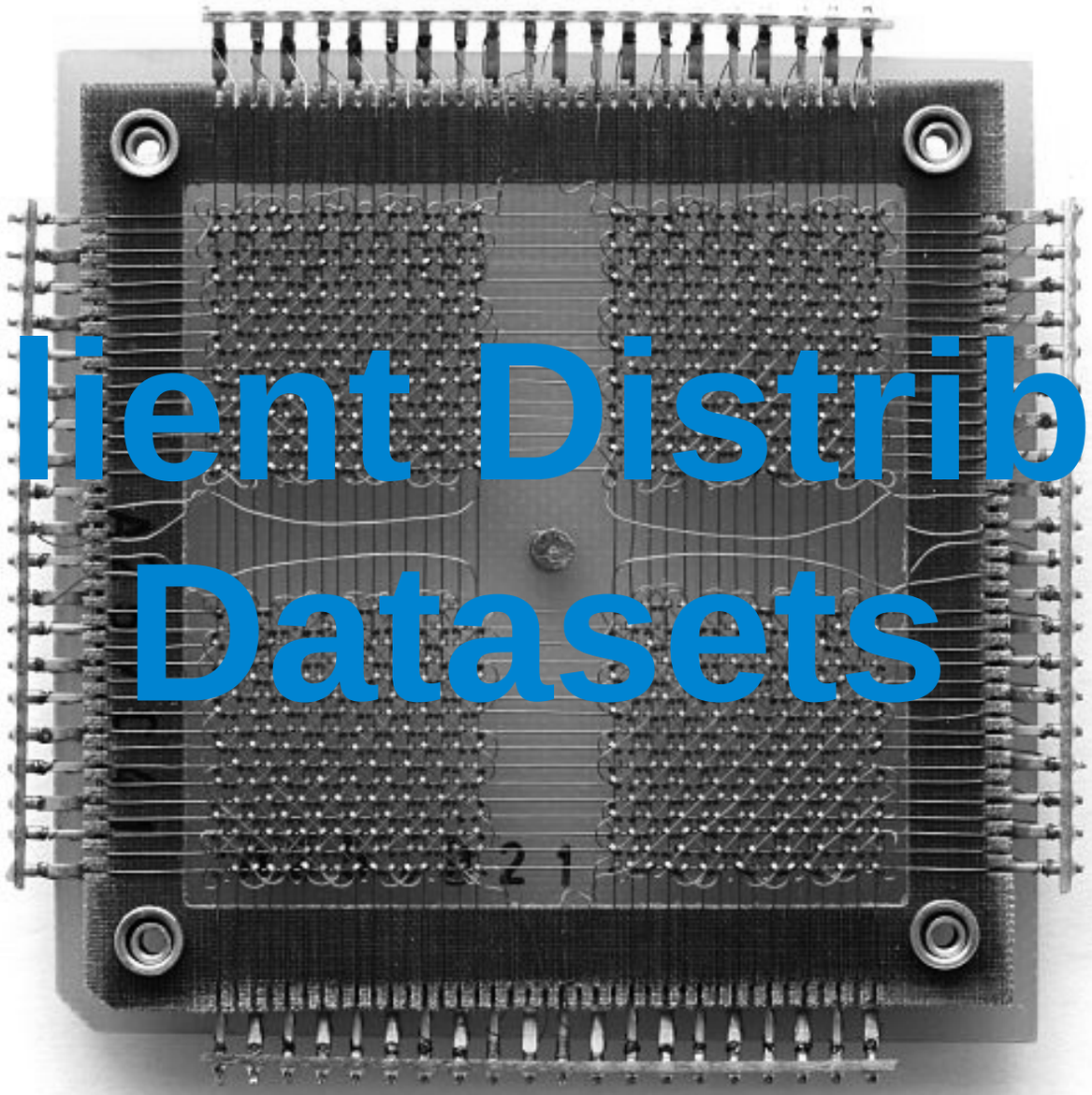
10-100x FASTER

than disk

Spark – Cool… but *how*?

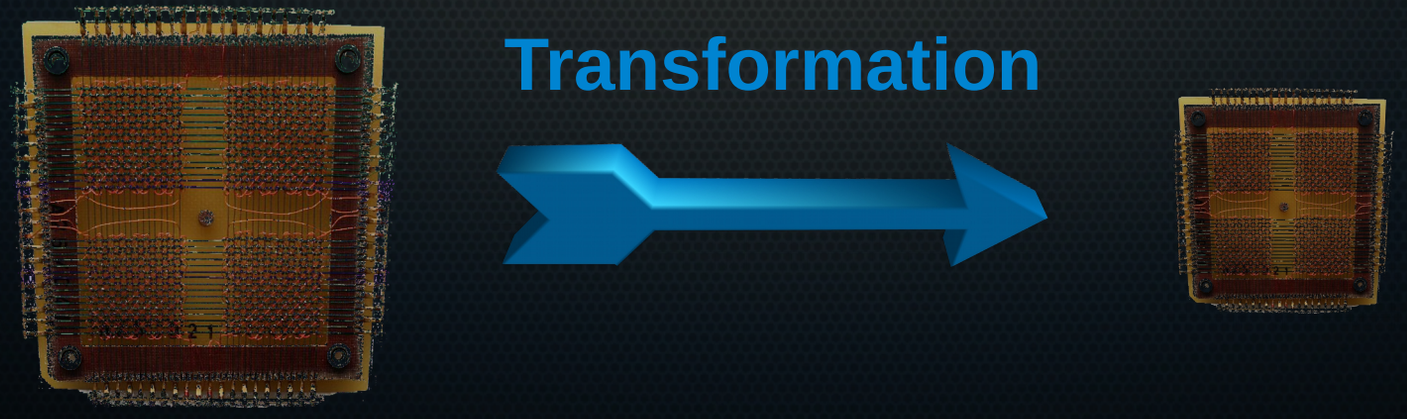
- Challenge
  - How to design a distributed memory abstraction that is both *fault-tolerant* and *efficient*?
- Solution
  - Resilient Distributed Datasets (aka RDDs)

Resilient Distributed Datasets



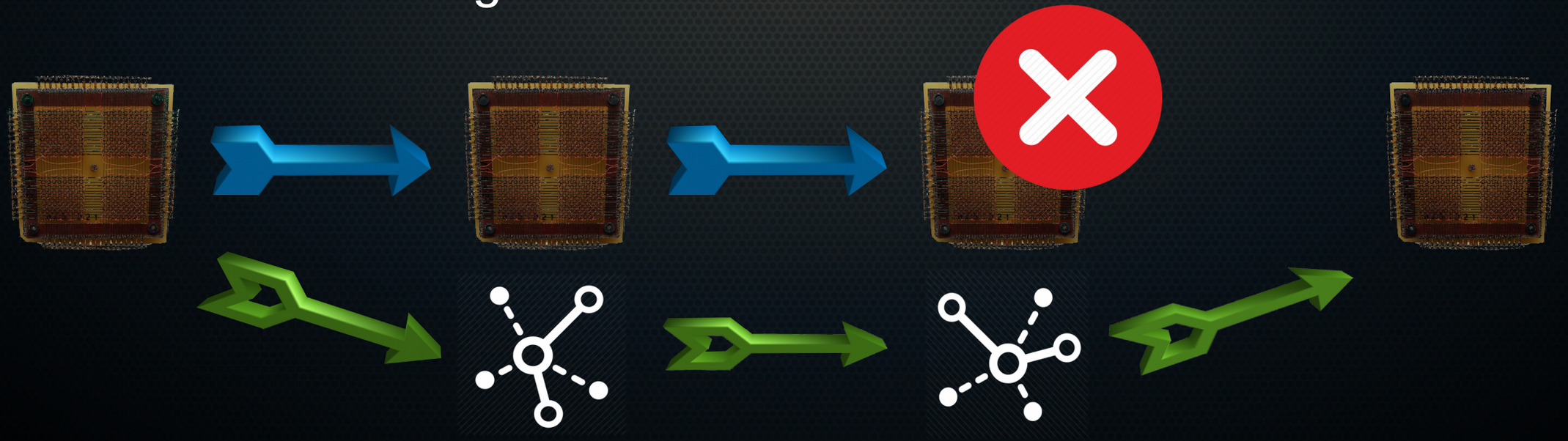
Spark – RDDs Definition

- Restricted form of distributed shared memory
  - Immutable, partitioned collection of records
  - Can only be built through *coarse-grained* deterministic transformations (map, filter, join, ...)



Spark – RDDs Error Handling

- Efficient fault recovery using *lineage*
  - Track the graph of transformations that build themselves
  - Recompute lost partition on failure
  - No cost if nothing fails



Spark – Programming Interface

- Programmer can perform 3 types of operations
  - Transformations *Returns an RDD*
    - `Map(), Filter(), Distinct()`
  - Actions *Returns a result*
    - `Count(), Reduce(), Collect(), Take()`
  - Persistence *Accelerates access time*
    - `Persist(), Cache()`

Spark – RDD Laziness

- Laziness
  - Since RDDs track every *transformations* applied to themselves
  - No need to compute anything until an *action* wants a result!
- So we just compute <u>what we need</u>

Spark – RDD Persistence

- Caching is a *key tool* for iterative algorithms and fast interactive computing
- You can mark an RDD to be persisted using the persist() or cache() methods on it
- The first time it is computed in an action, it will be kept in memory on the nodes.

  → Fault-tolerant !

Spark – RDD Persistence

- Each persisted RDD can be stored using a different storage level
  - On disk
  - In memory
  - Replicated or not across nodes
- Spark also automatically persists some intermediate data in shuffle operations (e.g. reduceByKey), even without users calling persist()

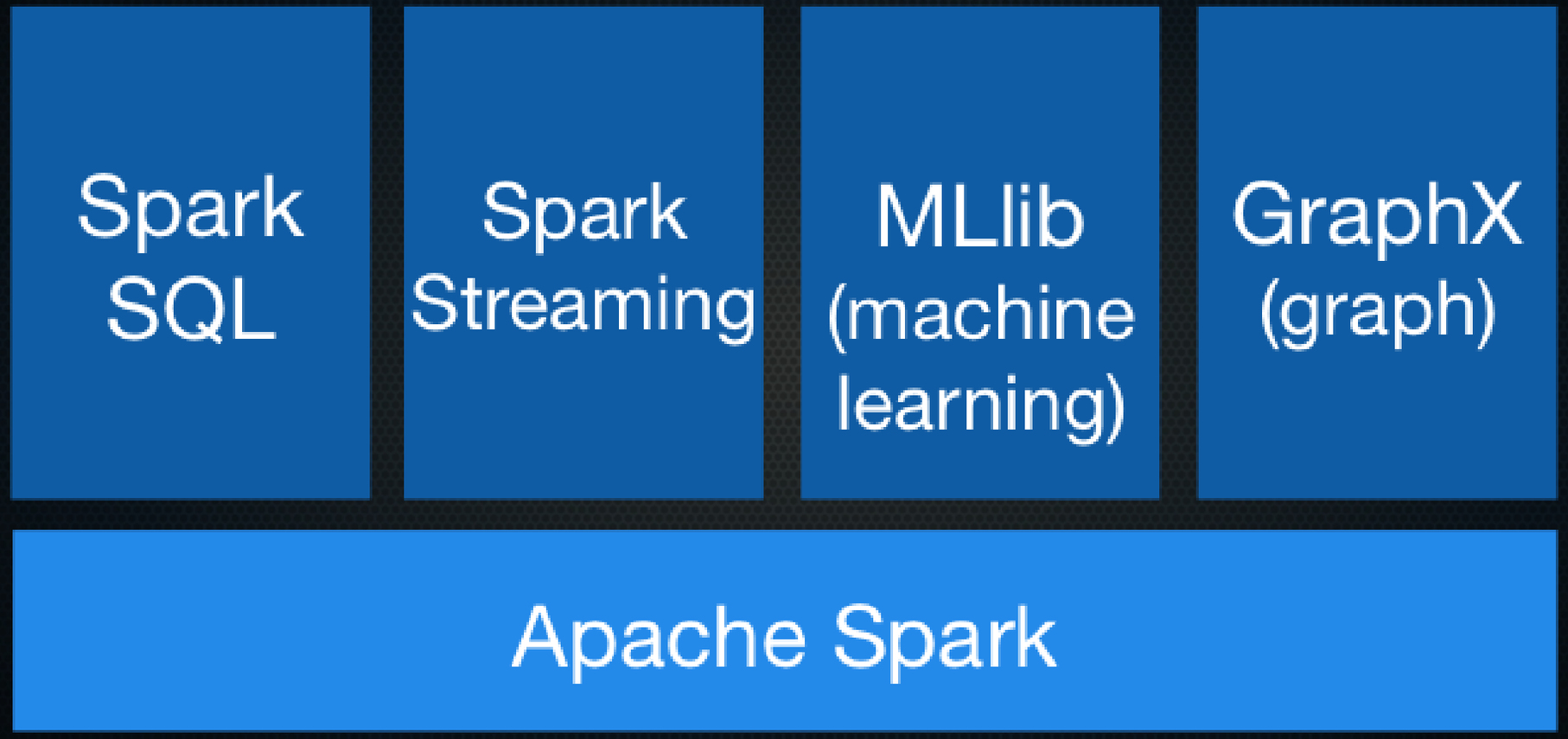
Spark – RDDs Methods

Transformations (define a new RDD)	**map** -filter -sample groupByKey - **reduceByKey** sortByKey -flatMap - **union** **join** -cogroup -cross mapValues
Actions (return a result)	collect - **reduce** count - save -lookUpKey

Spark – RDDs abstraction

- RDDs express many parallel algorithms
  - → Simply applies the same operation to many items
- Unify many programming models
  - Data Flow models: MapReduce, SQL, …
  - Specialized models for iterative apps: Graph processing, iterative MapReduce, bulk incremental, …
- Support *new apps* that these models don't!

Spark – New applications



MapReduce with Spark

Spark – MapReduce Example

```
sc.textFile(8000beans" )
```

RDD[8000beans]

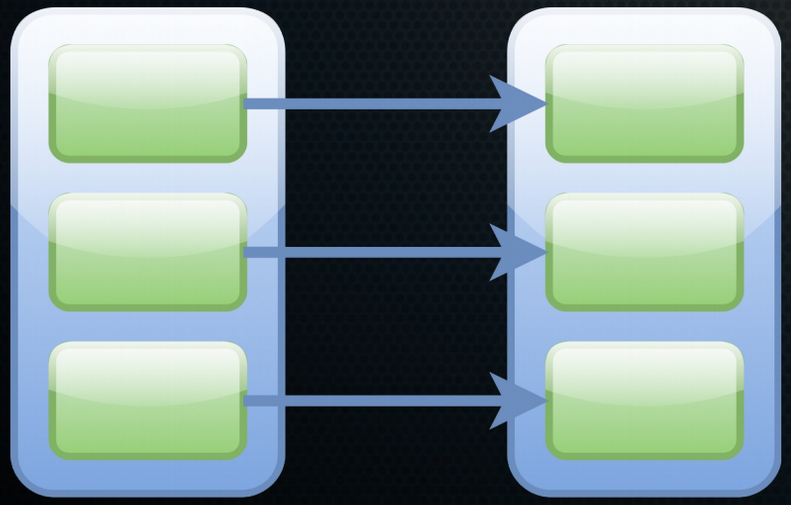
textFile

Spark – MapReduce Example

```
sc.textFile(8000beans" )
  .map( line => line.split("" ) )
```

RDD[8000beans]

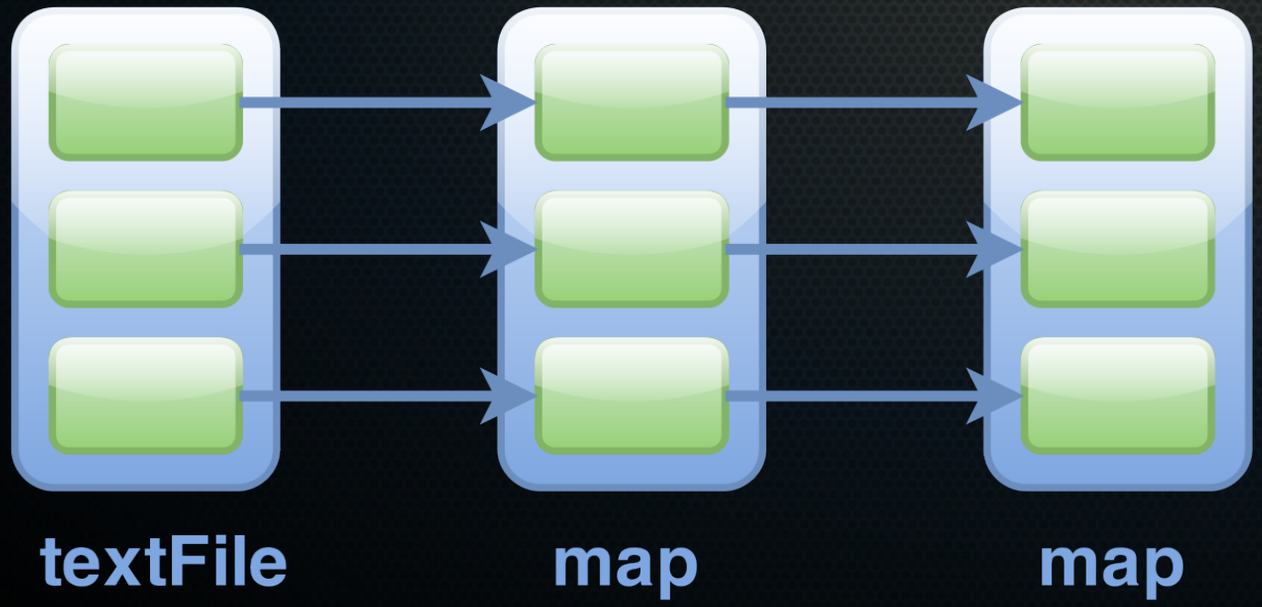
RDD[List[bean]]



Spark – MapReduce Example

```
sc.textFile(8000beans" )
  .map( line => line.split("" ) )
  .map( color => color[], 1 )
```

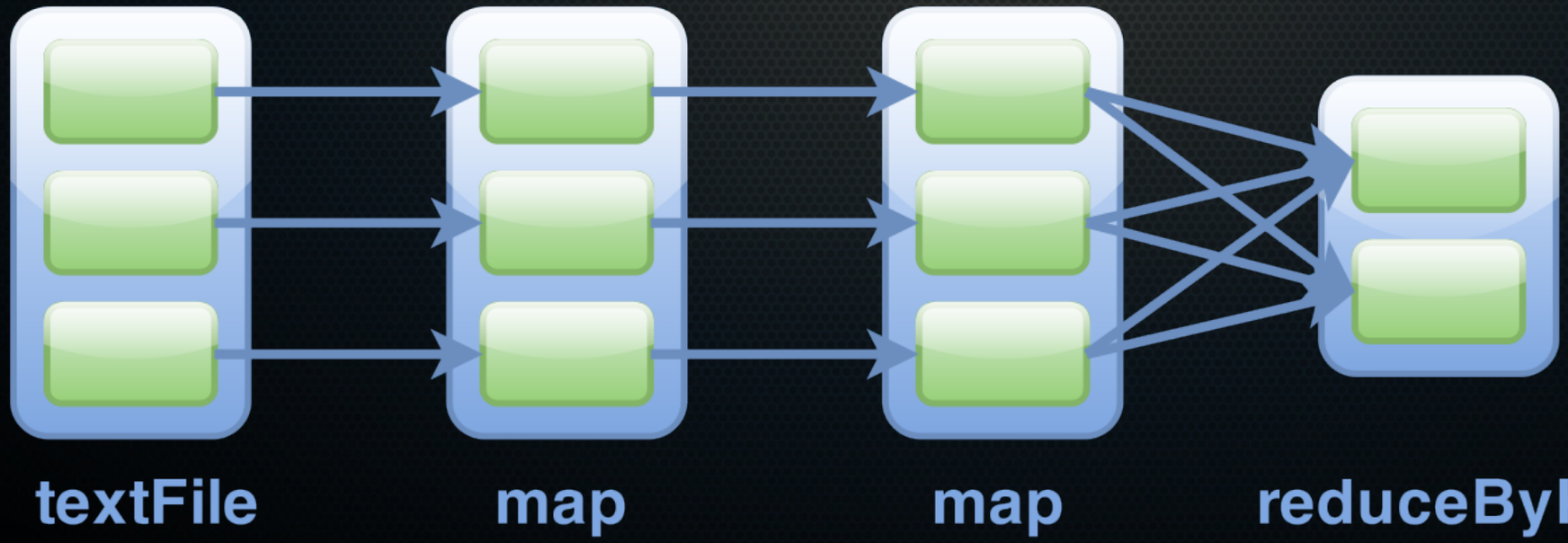
RDD[8000beans]
RDD[List[bean]]
RDD[beancolor, Int]



Spark – MapReduce Example

```
sc.textFile(8000beans" )
  .map( line => line.split("" ) )
  .map( color => color[], 1 )
  .reduceByKey( _+_ )
```

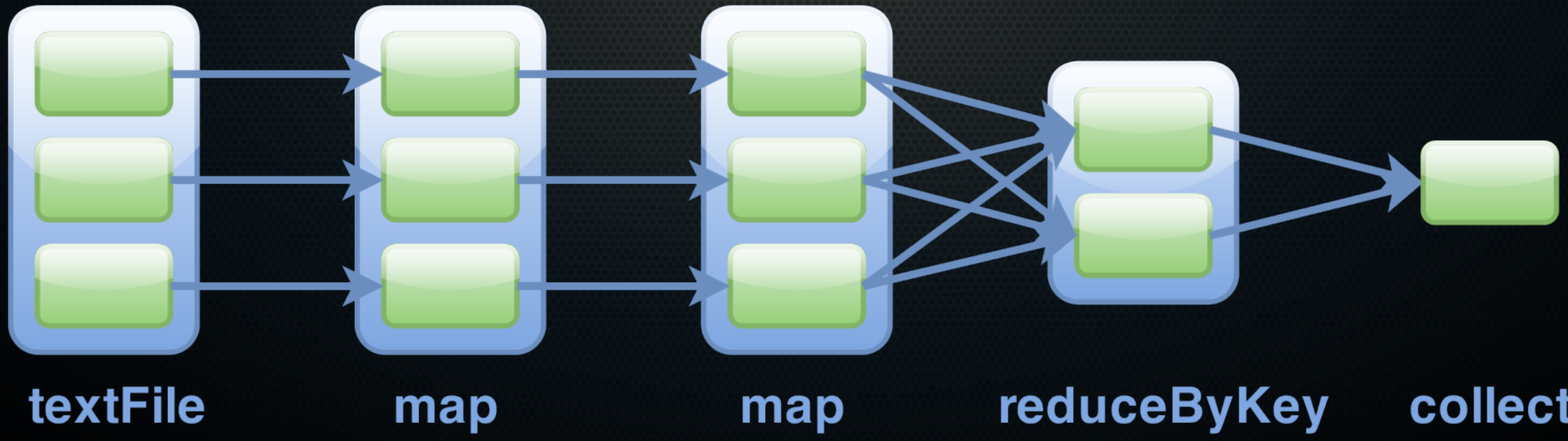
RDD[8000beans]
RDD[List[bean]]
RDD[beancolor, Int]
RDD[beancolor, Int]



Spark – MapReduce Example

```
sc.textFile(8000beans" )
  .map( line => line.split("" ) )
  .map( color => color[], 1 )
  .reduceByKey( _+_ )
  .collect()
```

RDD[8000beans]
RDD[List[bean]]
RDD[beancolor, Int]
RDD[beancolor, Int]
Array[(beancolor, Int)]

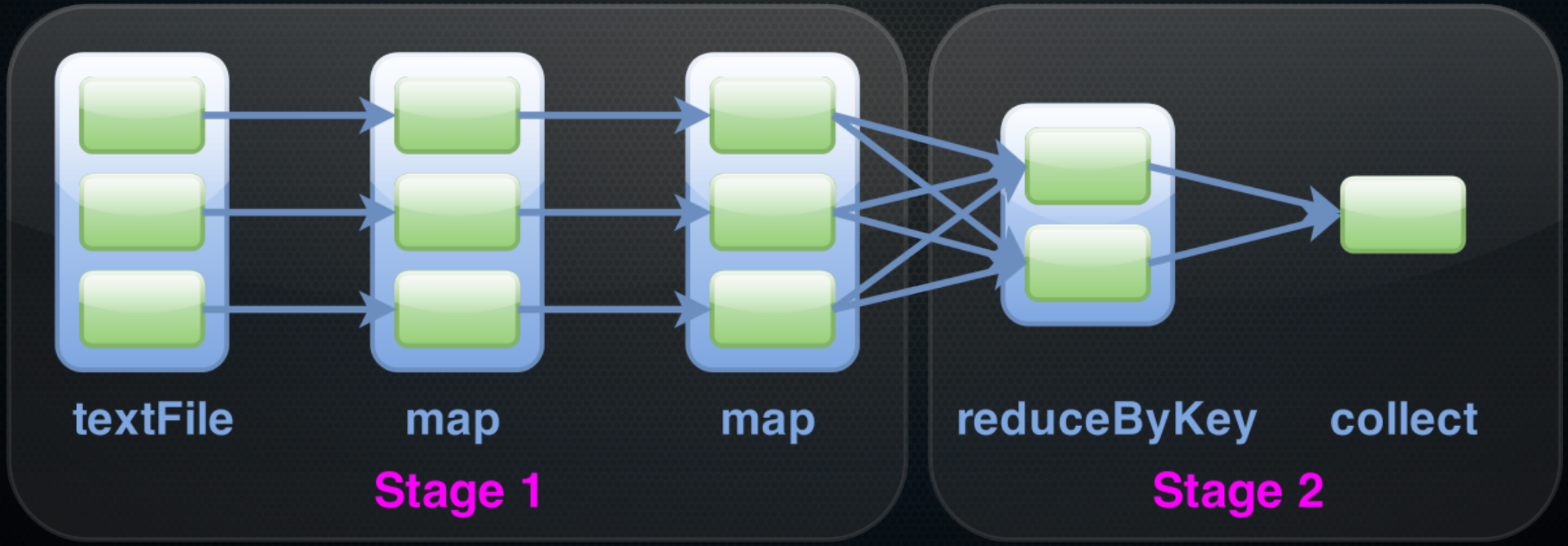


Spark – MapReduce Execution

- User create RDDs, Transform them, and run Actions on them
- This results in a Directed Acyclic Graph (DAG) of operators
- DAG is compiled into Stages
- Each Stage is executed as a series of Task
  (one Task for each RDD Partition)

Spark – MapReduce Execution

- Stages are sequences of RDDs, that *don't have* a MapReduce Shuffle in between !



Spark - Summary of Components

- Task
  - The fundamental unit of execution in Spark
- Stage
  - Set of Tasks that run parallel
- DAG
  - Logical Graph of RDD operations
- RDD
  - Parallel dataset with partitions

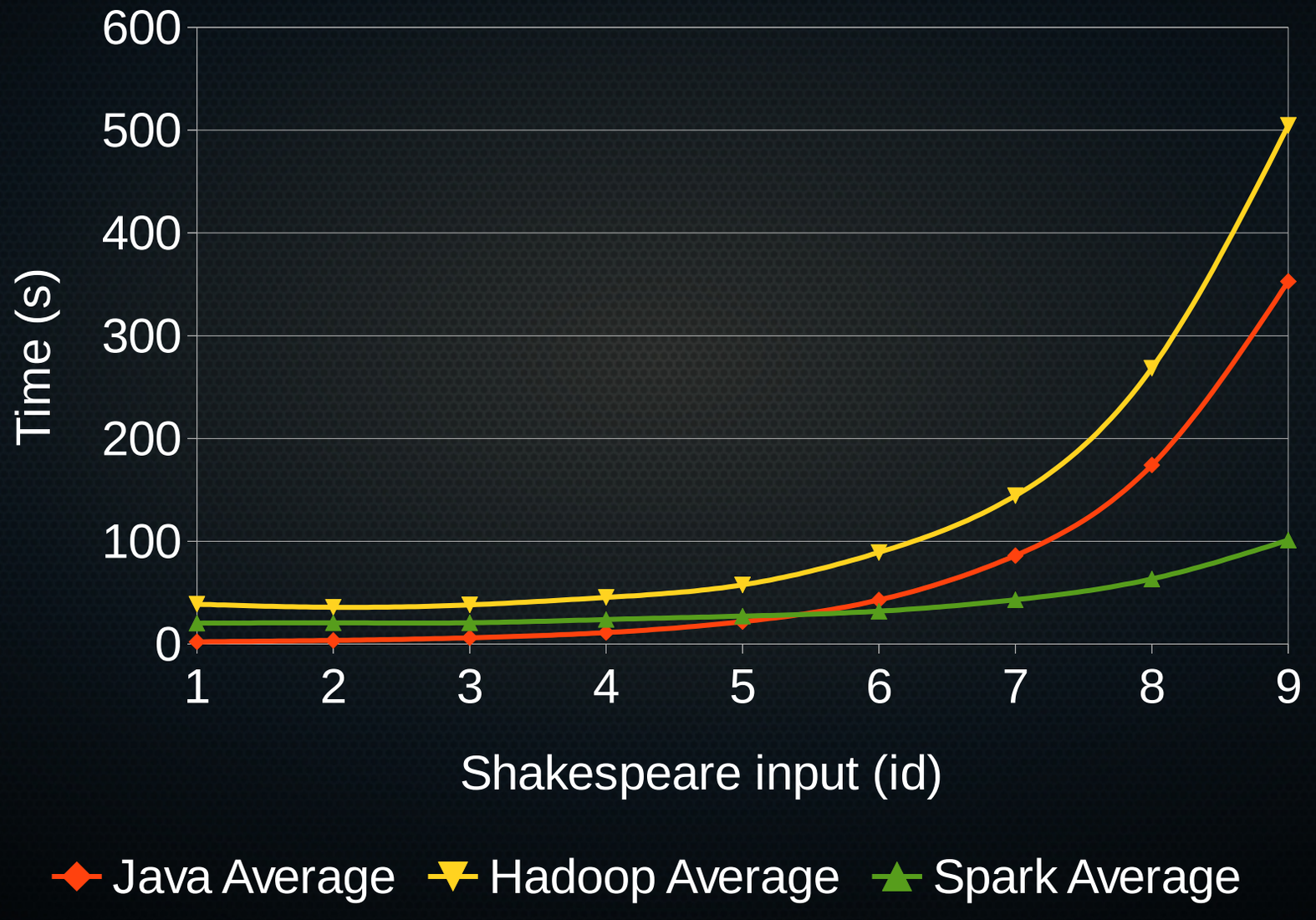
Spark – Tests Methodology

- Run “WordCount“ in Java, Hadoop and Spark on the same dataset
  - File size varying from 5.4 MB to 1.3 GB
- Collect
  - From app submitting to prompt return duration: appTime
- Compare
  - appTime between Java, Hadoop and Spark

Spark – MapReduce Tests

- Setup
  - 3 x (1CPU x64 @ 2.5GHz, 2GB RAM) VMs with CentOS 7
  - Hadoop 2.6.0, Spark 1.3.1, OpenJDK 1.8
- Datasets
  - shakespeare.txt
    - 5.MB, 124 787 lines, 904 061 words
  - shakespeare_*N*.txt
    - shakespeare.txt * *N*

Java VS Hadoop VS Spark



Java VS Hadoop VS Spark

- Hadoop is always *slower* than Java !
- Hadoop and Spark overhead on small files is comprehensible
  - Clustering induces delay on small jobs
- For files > 100MB, Spark is always the fastest


Spark SQL

Spark - DataFrames

- A DataFrame is a distributed collection of data organized into named columns
- Conceptually *equivalent* to a table in a relational database but with richer optimizations under the hood
- DataFrames can be constructed from a wide array of sources such as:
  - Structured data files, Hive tables, external databases or existing RDDs and DataFrames

Spark - DataFrames

- Ability to scale from *KB* of data on a single laptop to *PB* on a large cluster
- State-of-the-art optimization and code generation through the Spark SQL Catalyst optimizer
- Seamless integration with all big data tooling and infrastructure via Spark API

Spark – MySQL in spark-shell

- Using the MySQL JDBC driver is easy :

```
[hadoop@hspoc-m ~]$ ./spark-shell-mysql.sh
```

→ Interactive Spark shell with MySQL integration !


Live Examples

Thanks !
Any Questions ?

- sk4nz@sk4nz.ninja
- https://github.com/samuelaubertin/hspoc
- https://twitter.com/sk4nz

Glossary

- Master
- Worker
- RDD
- Transformation
- Action
- Persistence

Webography

- Apache Spark documentation
  https://spark.apache.org/docs/latest/
- Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing
  Matei Zaharia - Mosharaf Chowdhury - Tathagata Das
  Ankur Dave - Justin Ma - Murphy McCauley
  Michael J. Franklin - Scott Shenker - Ion Stoica
  https://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf