Apache Spark

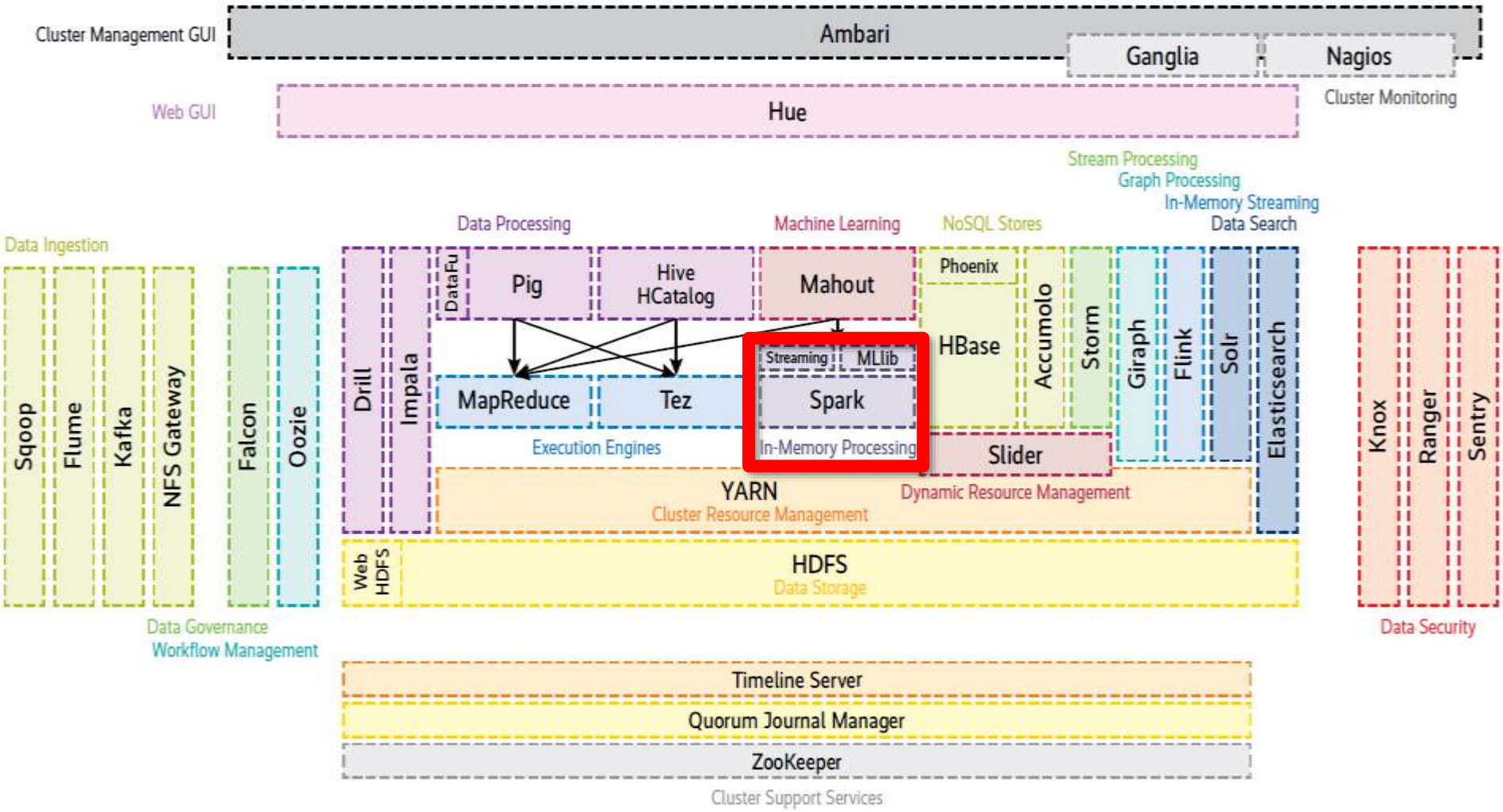
http://spark.apache.org/

Prof. Dr. Stephan Trahasch

Outline

- Spark
- RDD, DataFrame and DataSet
- Using Spark

Spark in the Hadoop Ecosystem



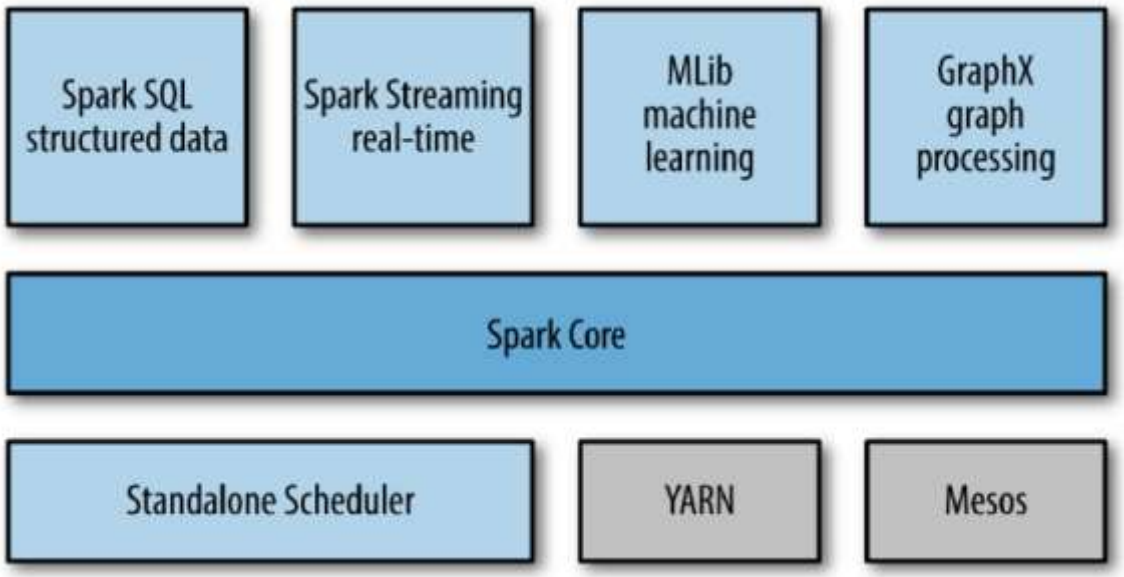
Apache Spark

- Apache Spark is a fast, general and distributed in-memory engine for large-scale data processing
- Originally developed at UC Berkeley AMPLab in 2009, https://amplab.cs.berkeley.edu/ Now Databricks https://databricks.com/
- Core concept RDD – Resilent Distributed Datasets
- Application programming interface for Java, Python, Scala, and R
- Papers http://spark.apache.org/research.html
  - An Architecture for Fast and General Data Processing on Large Clusters. Matei Zaharia, 2013 (PhD Disseration).
  - 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. NSDI 2012. April 2012. Best Paper Award.
  - Spark: Cluster Computing with Working Sets. Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica. HotCloud 2010. June 2010.

Apache Spark – current version 2.x

Research project 2009 (AMPLab University Berkeley)

- 2010: Open source
- 2013: Top-Level Project Apache SF
- 2014: Version 1.0
- Matei Zaharia: ACM Award `14 for his Phd about RDDs



Platform for distributed data processing

Large-Scale Usage

Largest cluster
8000 Nodes (Tencent)

Largest single job
1 PB (Alibaba, Databricks)

Top Streaming Intake
1 TB/hour (HHMI Janelia Farm)

2014 On-Disk Sort Record
Fastest Open Source Engine for sorting a PB

databricks

Companies that presented at Spark Summit 2015 in San Francisco

Source: Slide 5 of Spark Community Update

databricks

Companies using Spark for Machine Learning Tasks

Source: [Why you should use Spark for Machine Learning]

databricks

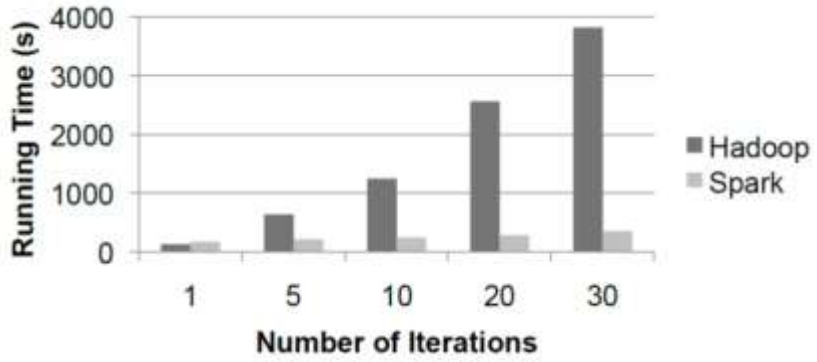
Example TOYOTA

- Original batch job: 160 hours
- Same Job re-written using Apache Spark: 4 hours
- ML task
  - Prioritize incoming social media in real-time using Spark MLlib (differentiate campaign, feedback, product feedback, and noise)
  - ML life cycle: Extract features and train:
  - V1: 56% Accuracy -> V9: 82% Accuracy
  - Remove False Positives and Semantic Analysis (similarity between concepts)

Source: Toyota Customer 360 Insights on Apache Spark and MLlib Performance

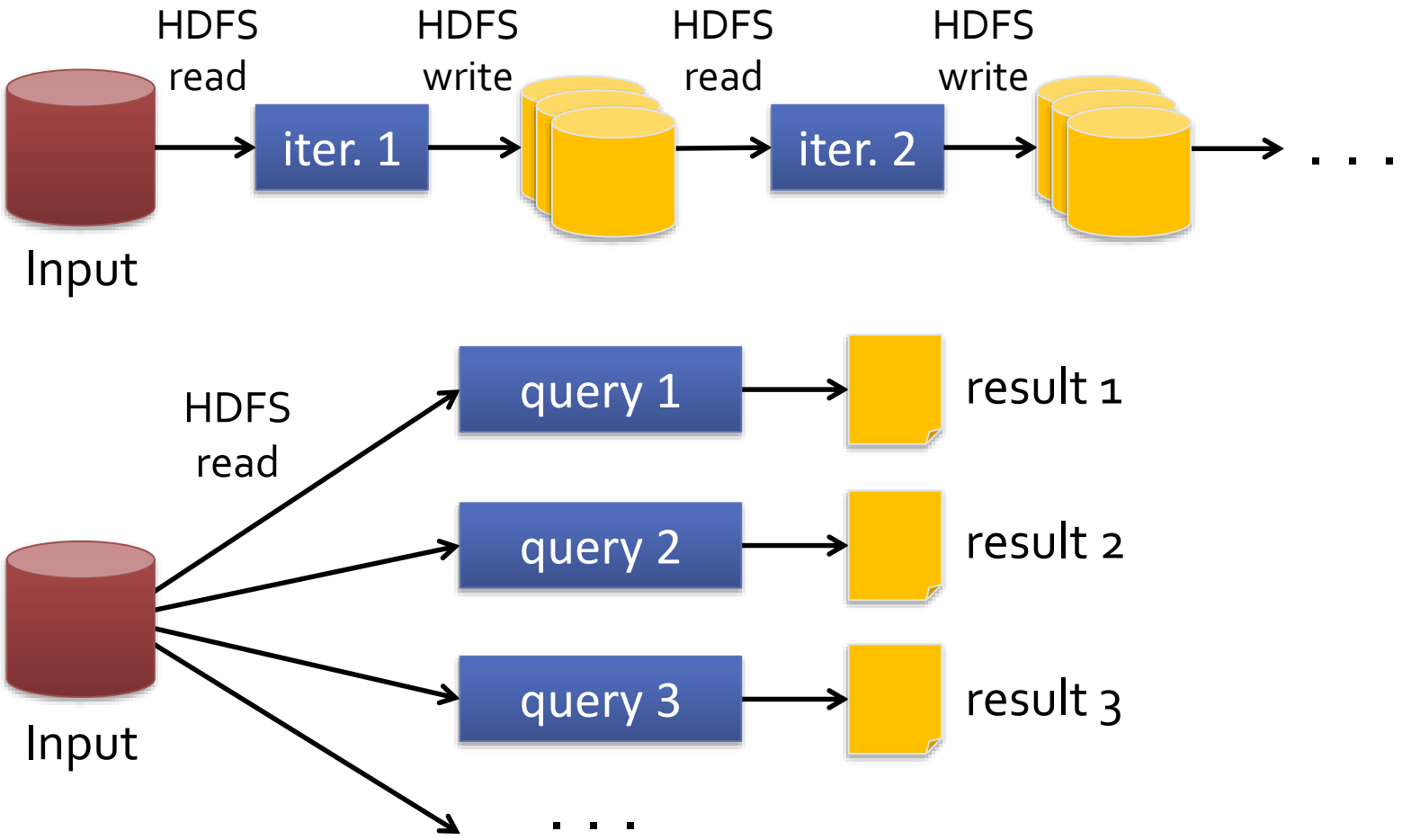
From MapReduce to Spark

- Dynamic ad-hoc queries in data warehouse systems
  - Not possible with MapReduce
- Increased performance through new hardware
  - Main memory, processor power



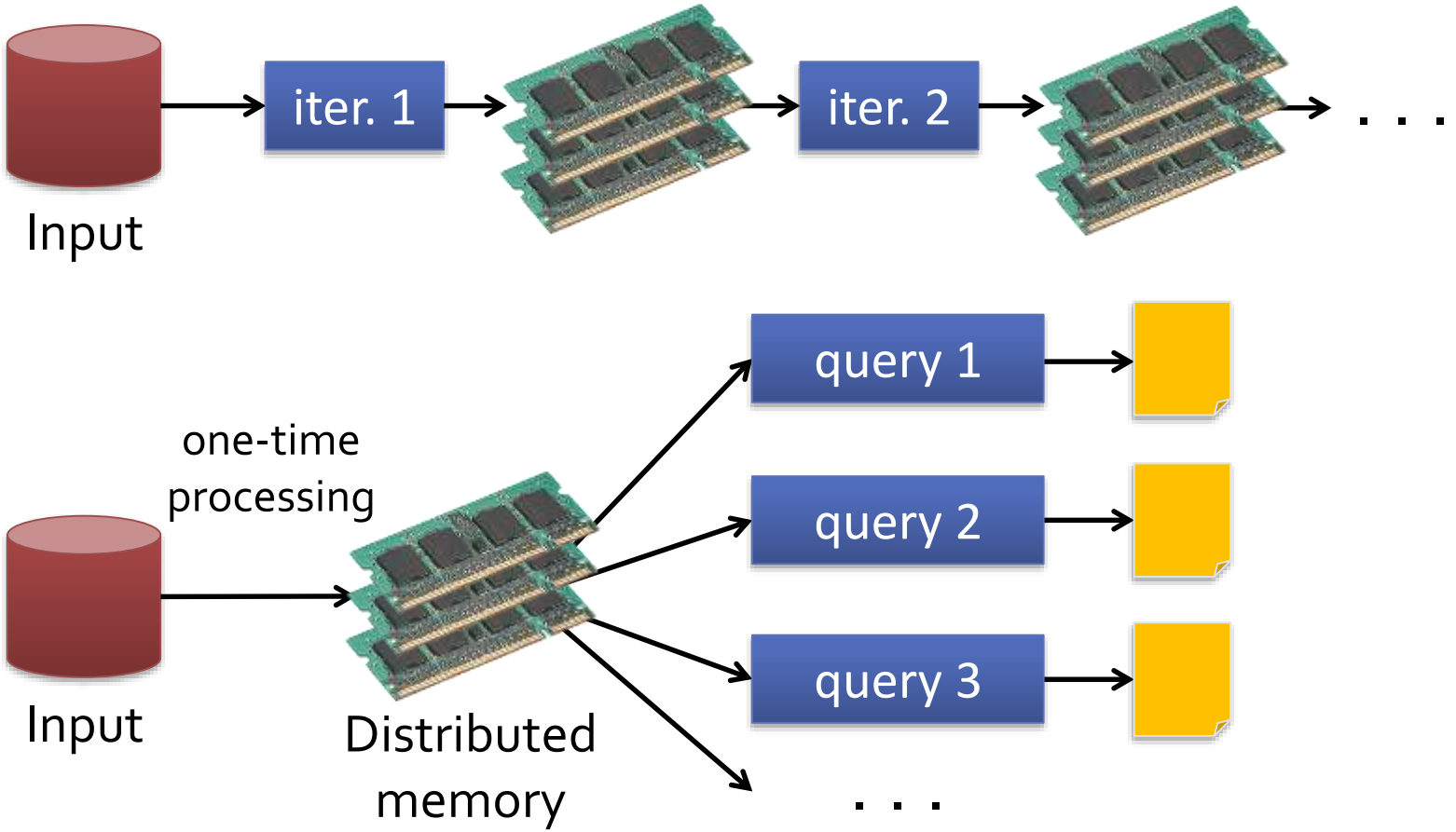
- Further development of MapReduce
  - In-Memory Computing (vs. Disk)
  - Distributed calculation via RDDs

Data Sharing in MapReduce



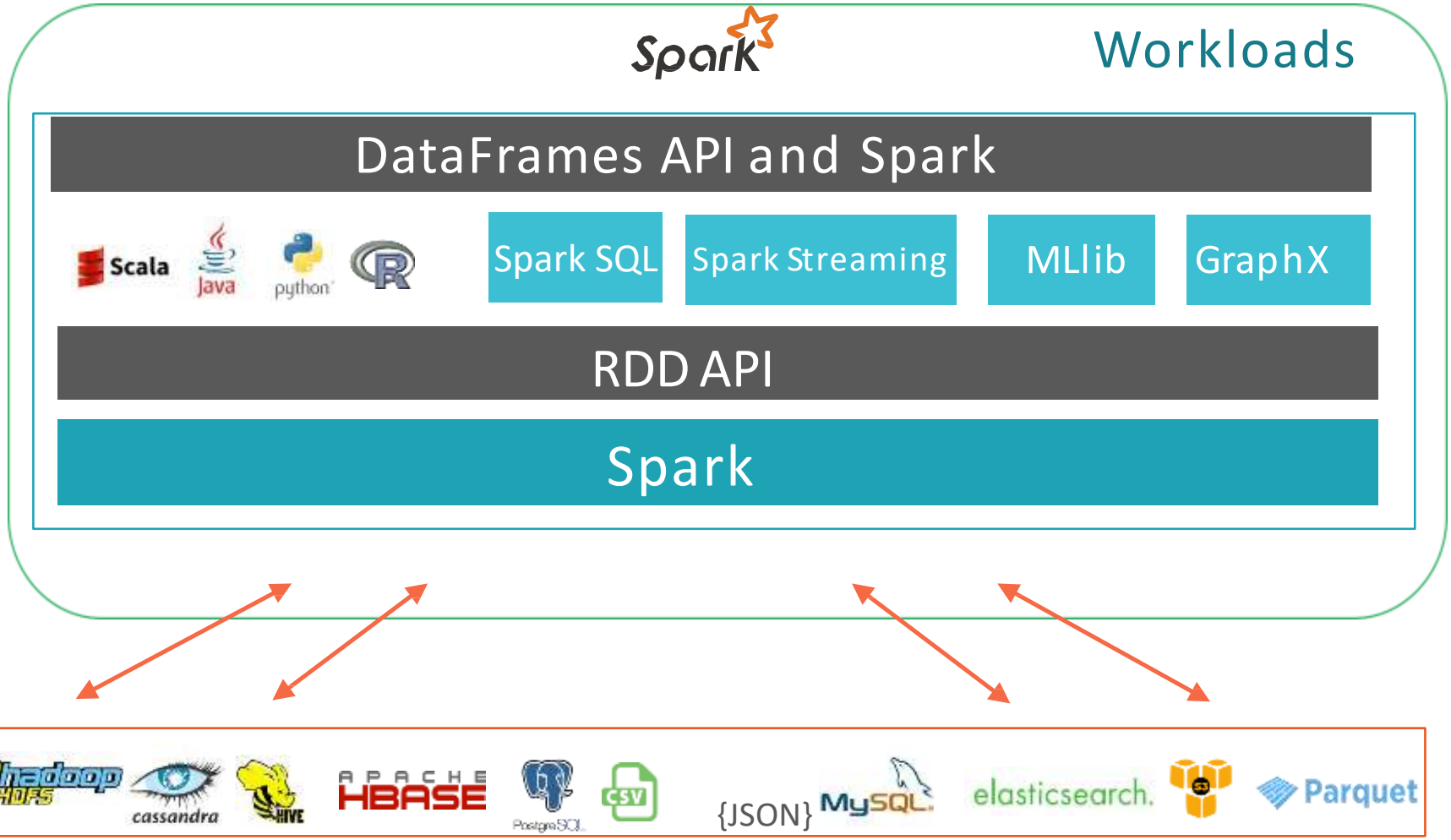
Slow due to replication, serialization, and disk IO

Data Sharing in Spark



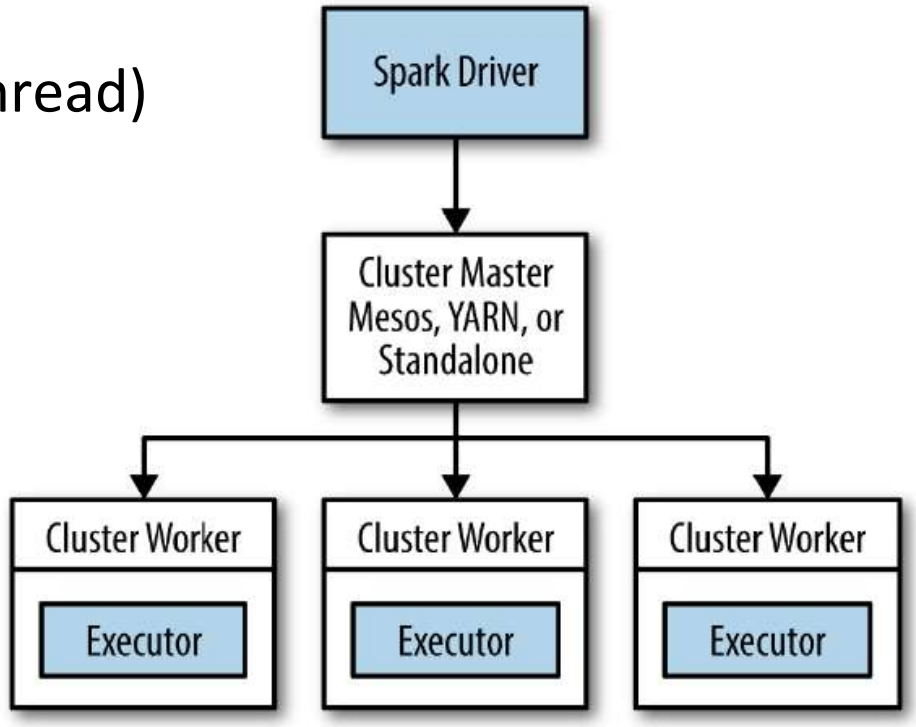
10-100× faster than network and disk

Apache Spark Engine

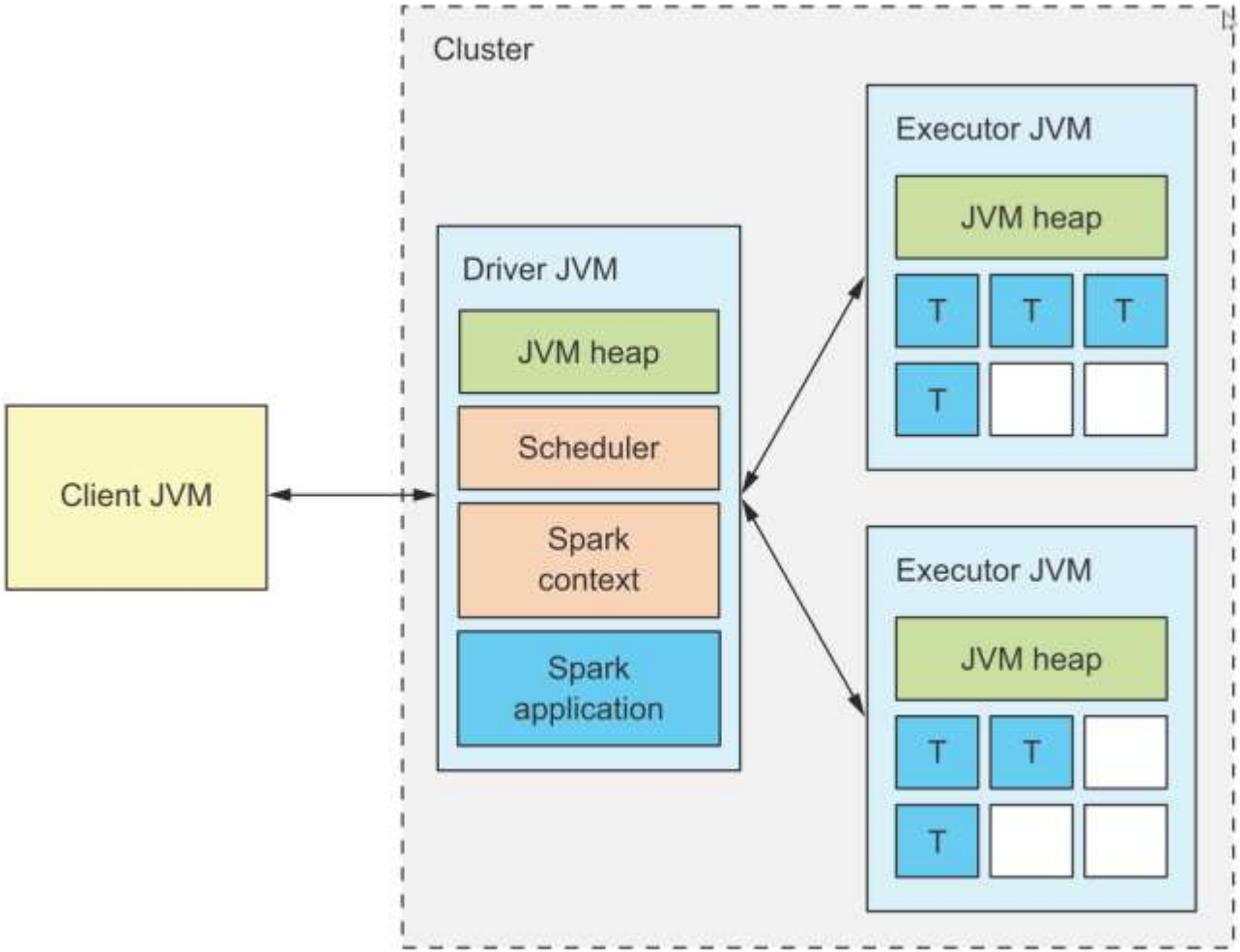


Components of a Spark cluster

- Master / Slave architecture
- Master: coordination (Driver Thread)
  - Executes programs
  - *SparkSession / SparkContext* as entrypoint
  - Definition of RDDs and actions
- Executor
- Ressource coordination with cluster manager



Components running in a cluster: client, driver, executors



Source: Spark in Action

Spark Cluster Types

- Standalone Cluster
  - Provides faster job startup than those jobs running on YARN.
- YARN Cluster
  - YARN lets you run different types of Java applications, so you can mix legacy Hadoop and Spark applications.
  - YARN provides methods for isolating and prioritizing applications among users and organizations.
  - It's the only cluster type that supports Kerberos-secured HDFS.
- MESOS Cluster
  - Mesos is a scalable and fault-tolerant distributed systems kernel
  - Mesos is a "scheduler of scheduler frameworks"
  - provides scheduling of other types of resources (CPU, disk space ..)
- Spark Local Mode
  - Special case of a Spark standalone cluster running on a single machine

Outline

- Spark
- RDD, DataFrame and Data Set
- Using Spark

Core Concept: Resilient Distributed Dataset – RDD

Resilient
if data in memory is lost, it can be recreated

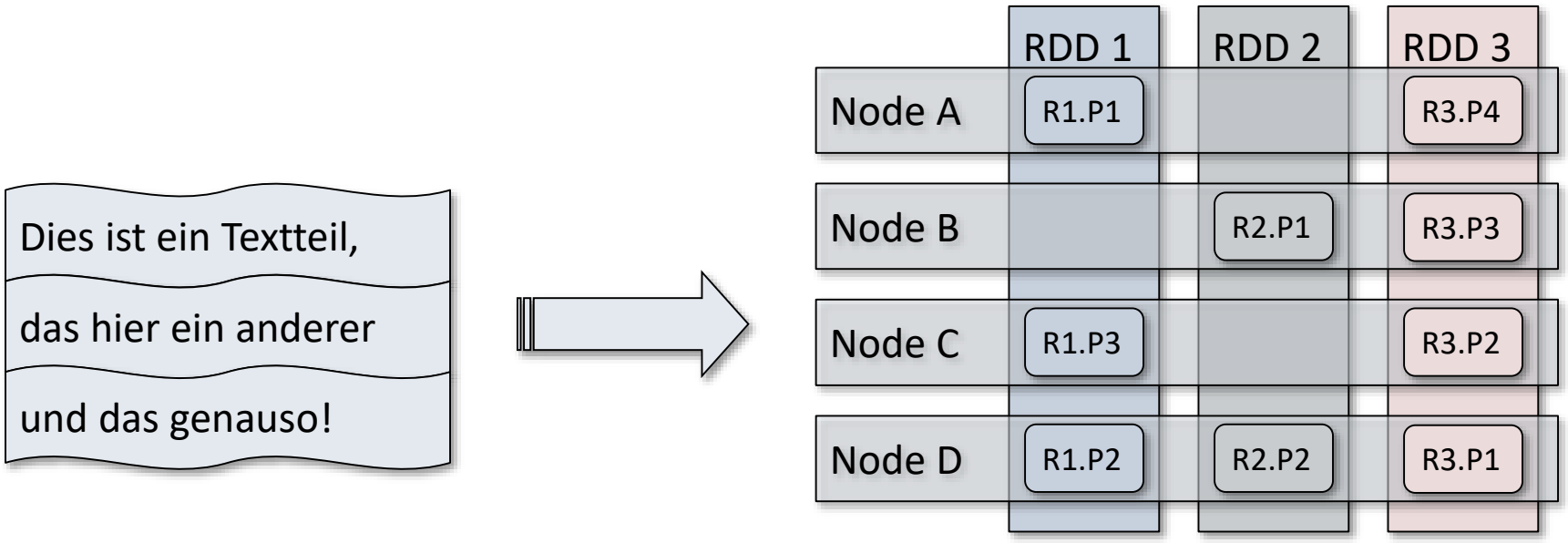
Distributed
processed across the cluster

Dataset
initial data can come from a Data Source or be created programmatically

- RDDs are the fundamental unit of data in Spark
- RDDs are immutable
  - Data in an RDD is never changed
  - Transform in sequence to modify the data as needed
- Spark programming consists of performing operations on RDDs

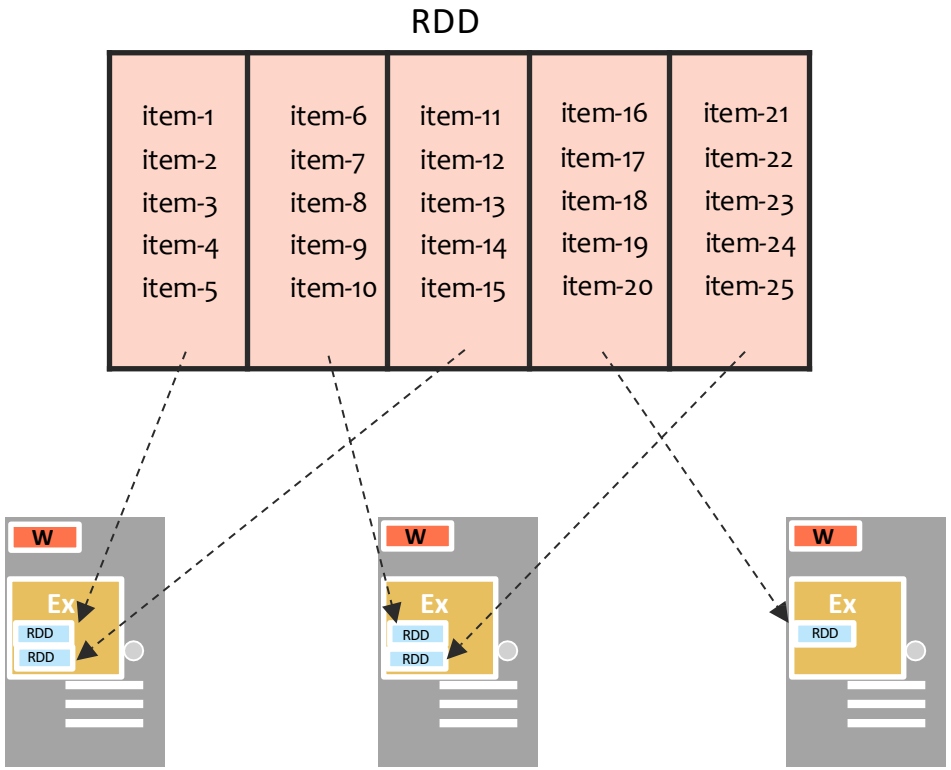
Resilient Distributed Datasets

- Fault-tolerant collection of elements partitioned across the nodes of the cluster that can be operated on in parallel.
- Immutable



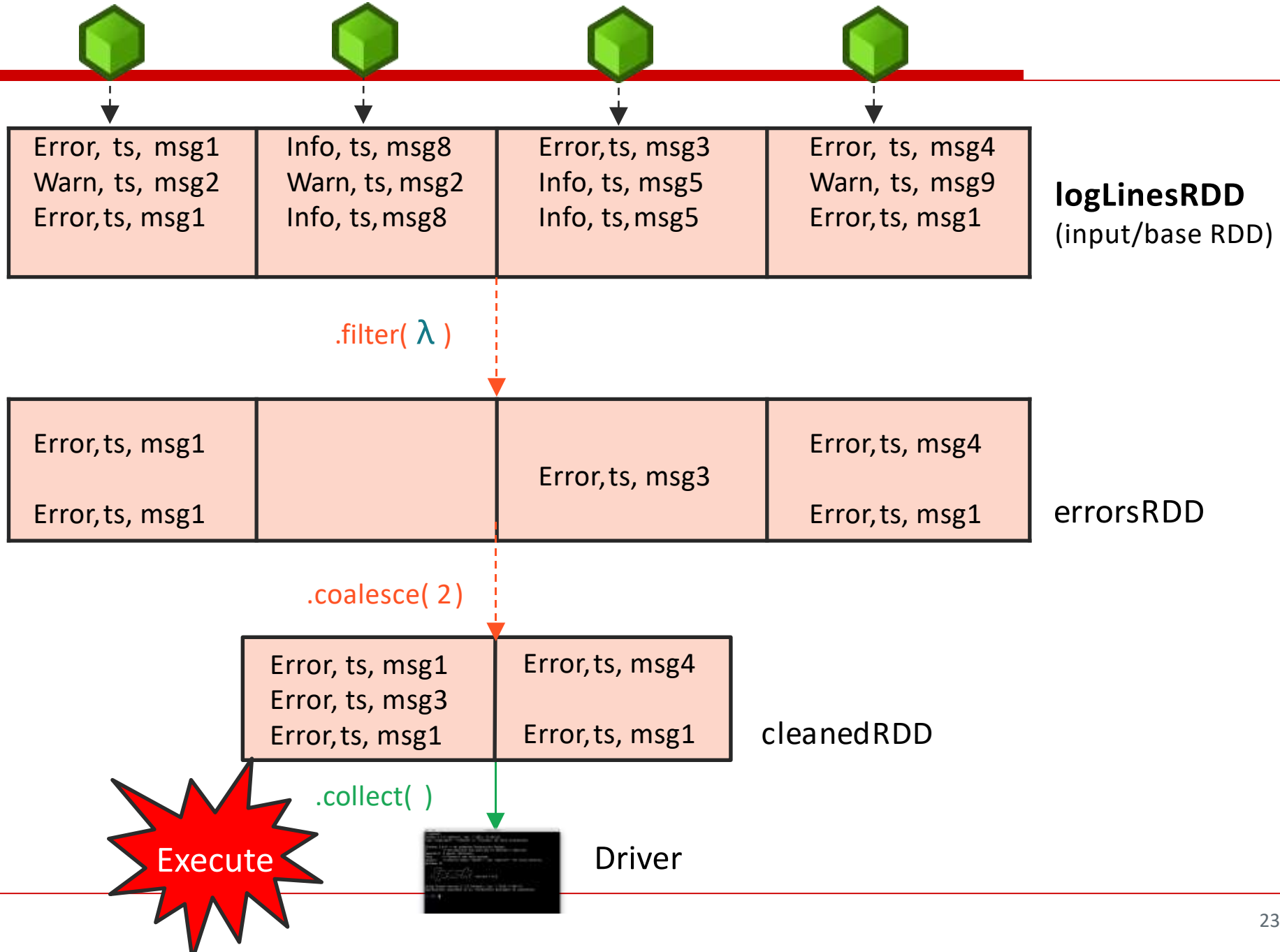
More Partitions = More Parallelism

- Write programs in terms of operations on distributed datasets
- Partitioned collections spread across cluster, in memory or disk



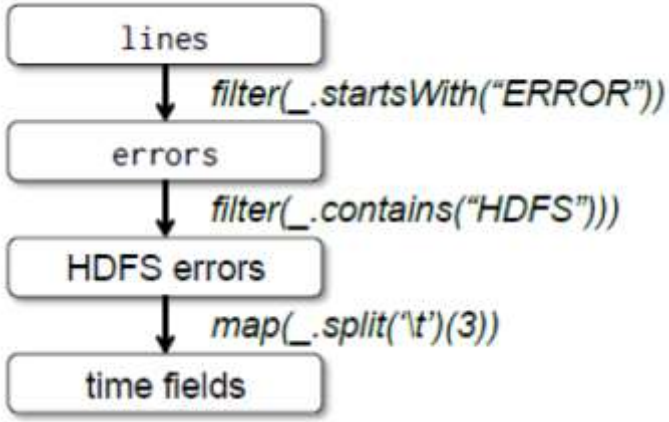
Two types of operations: transformations and actions

- RDDs support two types of operations:
  transformations (map, filter, join) , which create a new dataset from an existing one, and
  actions (count, collect, save), which return a value to the driver program.
- Transformations are lazy (not computed immediately)
- Transformations are executed when an action is run
- Persist (cache) distributed data in memory or disk
- RDDs automatically rebuilt on machine failure


Error, ts, msg1
Warn, ts, msg2
Error, ts, msg1
Info, ts, msg8
Warn, ts, msg2
Info, ts, msg8
Error, ts, msg3
Info, ts, msg5
Info, ts, msg5
Error, ts, msg4
Warn, ts, msg9
Error, ts, msg1
logLinesRDD
(input/base RDD)
.filter(λ)
Error, ts, msg1
Error, ts, msg1
Error, ts, msg3
Error, ts, msg4
Error, ts, msg1
errorsRDD
.coalesce(2)
Error, ts, msg1
Error, ts, msg3
Error, ts, msg1
Error, ts, msg4
Error, ts, msg1
cleanedRDD
.collect( )
Execute
Driver

Transformationen and Actions

- Transformations are lazy
  - Work with RDD dataset
    - Filter, KV pairs, Joins, …
  - RDD generation
    - Input: 1 (Filterung) – n (Join) RDDs
    - Output: 1 RDD

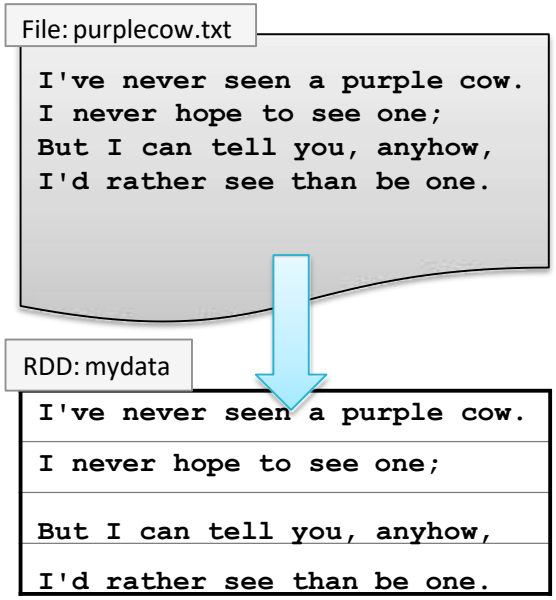


- Actions
  - Calculation on RDDs
  - Result for the driver
  - Starting point for transformation and action

- Representation of commands as *Directed Acyclic Graphs = DAGs*

Example: A File-based RDD

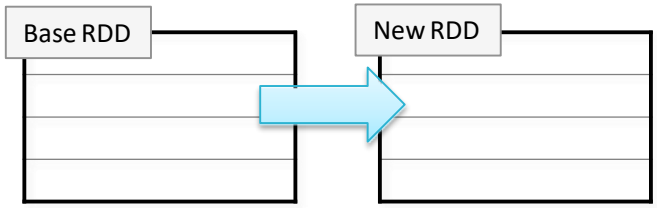
```
> val mydata = sc.textFile("purplecow.txt")
…
15/01/29 06:20:37 INFO storage.MemoryStore:
   Block broadcast_0 stored as values to
   memory (estimated size 151.4 KB, free 296.8
   MB)

> mydata.count()

…
15/01/29 06:27:37 INFO spark.SparkContext: Job
   finished: take at <stdin>:1, took
   0.160482078 s
4
```



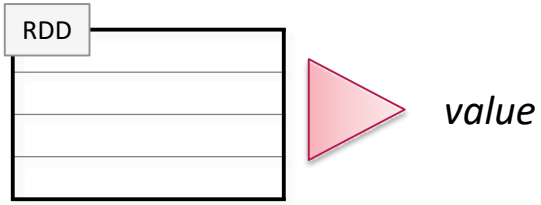
RDD Operations

Two types of RDD operations

1. Transformations
   define a new RDD
   based on the current one(s)
   are lazy (not computed immediately)
   are executed when an action is run



2. Actions
   return values

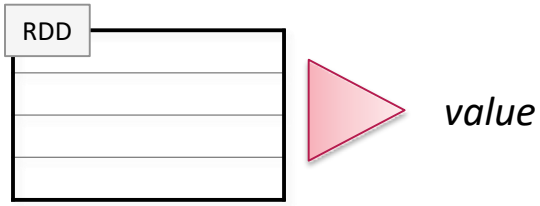


Persist (cache) distributed data in memory or disk

RDD Operations: Actions

Some common actions

- **`count()`** – return the number of elements
- **`take(`***`n`***`)`** – return an array of the first *n* elements
- **`collect()`** – return an array of all elements
- **`saveAsTextFile(`***`file`***`)`** – save to text file(s)



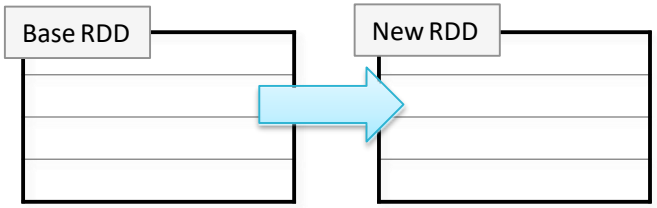

```
> mydata =
  sc.textFile("purplecow.txt")

> mydata.count()
4

> for line in mydata.take(2):
    print line
I've never seen a purple cow.
I never hope to see one;
```

```
> val mydata =
  sc.textFile("purplecow.txt")

> mydata.count()
4

> for (line <- mydata.take(2))
    println(line)
I've never seen a purple cow.
I never hope to see one;
```

RDD Operations: Transformations

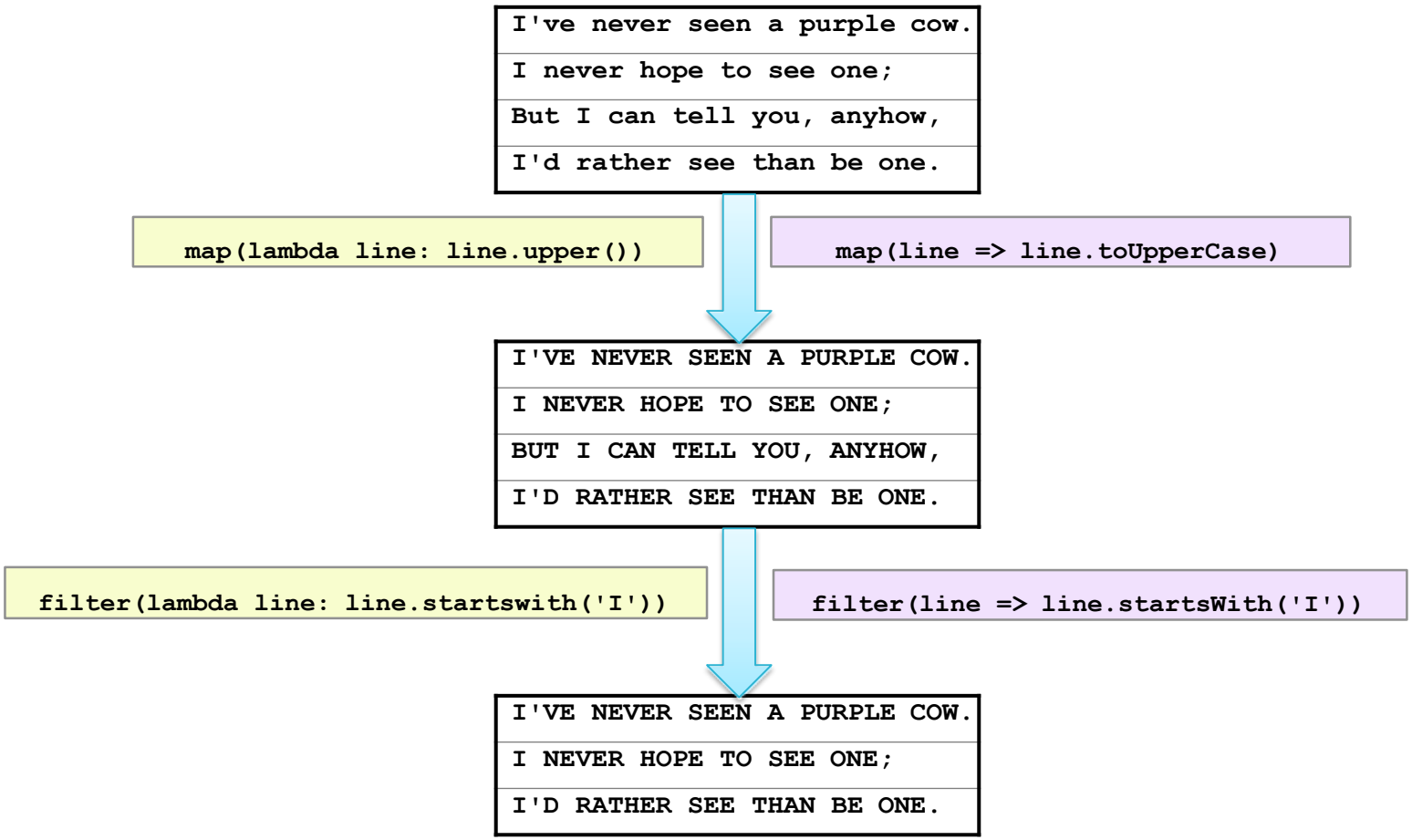
- Transformations create a new RDD from an existing one

- RDDs are immutable
  - Data in an RDD is never changed
  - Transform in sequence to modify the data as needed



- Some common transformations
  - **`map(`***`function`***`)`**
    creates a new RDD by performing a function on each record in the base RDD
  - **`filter(`***`function`***`)`**
    creates a new RDD by including or excluding each record in the base RDD according to a boolean function

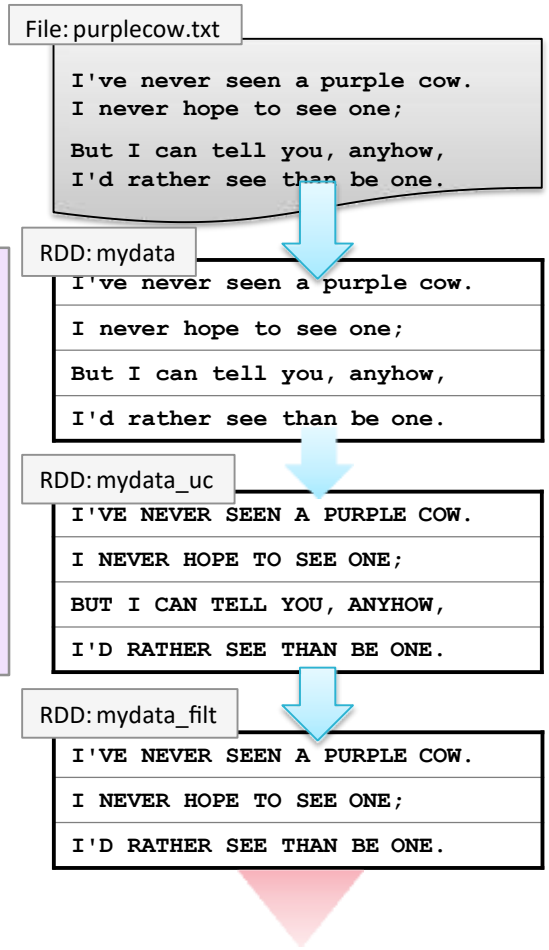
Example: `map` and `filter` Transformations



Lazy Execution

Data in RDDs is not processed until an action is performed

```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line =>
  line.toUpperCase())
> val mydata_filt = mydata_uc.filter(line
  => line.startsWith("I"))
> mydata_filt.count()
3
```



Chaining Transformations (Scala)

Transformations may be chained together

```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line => line.toUpperCase())
> val mydata_filt = mydata_uc.filter(line => line.startsWith("I"))
> mydata_filt.count()
3
```

is exactly equivalent to

```
> sc.textFile("purplecow.txt").map(line => line.toUpperCase()).
   filter(line => line.startsWith("I")).count()
3
```

Chaining Transformations (Python)

Same example in Python

```
> mydata = sc.textFile("purplecow.txt")
> mydata_uc = mydata.map(lambda s: s.upper())
> mydata_filt = mydata_uc.filter(lambda s: s.startswith('I'))
> mydata_filt.count()
3
```

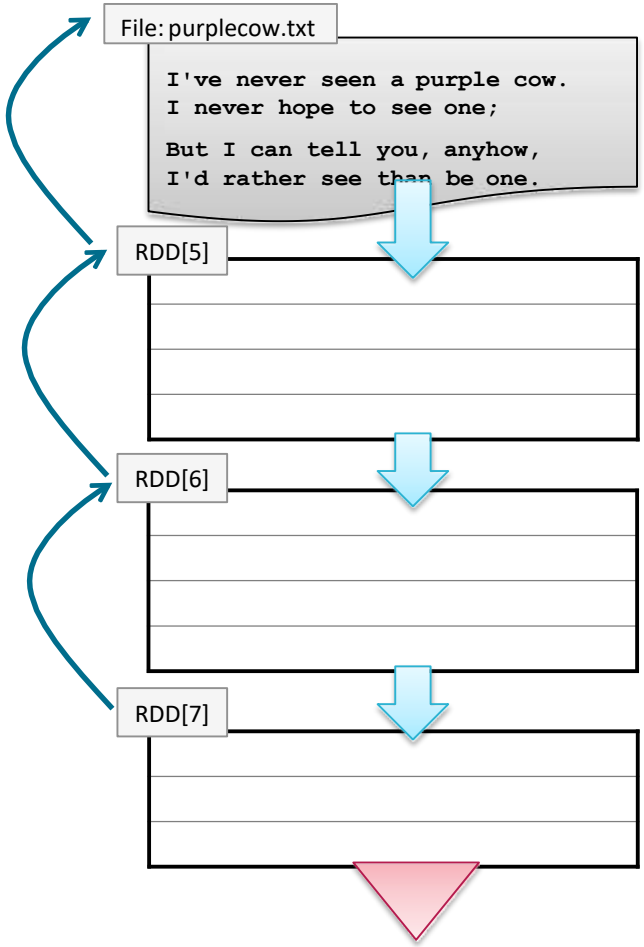
is exactly equivalent to

```
> sc.textFile("purplecow.txt").map(lambda line: line.upper()) \
   .filter(lambda line: line.startswith('I')).count()
3
```

RDD Lineage and toDebugString (Scala)

- Spark maintains each RDD's lineage
  - the previous RDDs on which it depends
- Use toDebugString to view the lineage of an RDD

```
> val mydata_filt =
  sc.textFile("purplecow.txt").
  map(line => line.toUpperCase()).
  filter(line =>
  line.startsWith("I"))

> mydata_filt.toDebugString
(2) FilteredRDD[7] at filter …
 |  MappedRDD[6] at map …
 |  purplecow.txt MappedRDD[5] …
 |  purplecow.txt HadoopRDD[4] …
```



RDD Lineage and `toDebugString` (Python)

toDebugString output is not displayed as nicely in Python

```
> mydata_filt.toDebugString()
(1) PythonRDD[8] at RDD at …\n |  purplecow.txt MappedRDD[7] at textFile
at …[]\n |  purplecow.txt HadoopRDD[6] at textFile at …[]
```

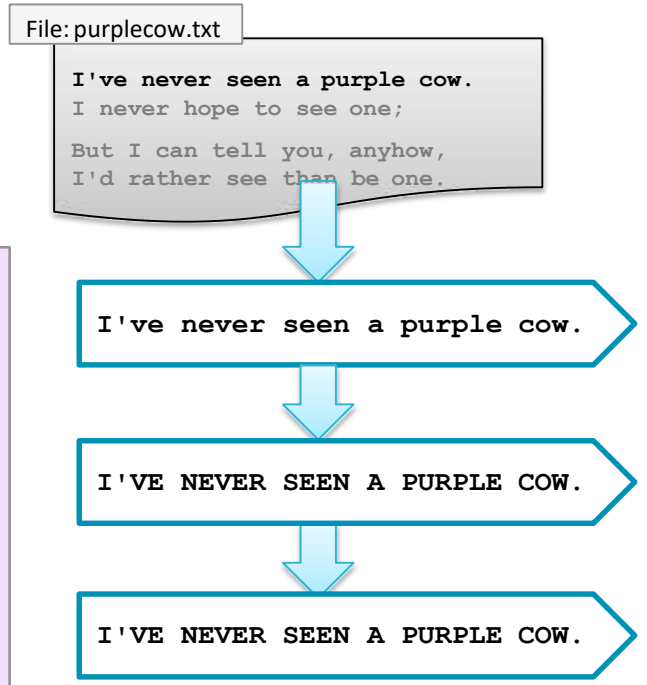
Use `print` for `pre_er` output

```
> print mydata_filt.toDebugString()
(1) PythonRDD[8] at RDD at …
 |  purplecow.txt MappedRDD[7] at textFile at …
 |  purplecow.txt HadoopRDD[6] at textFile at …
```

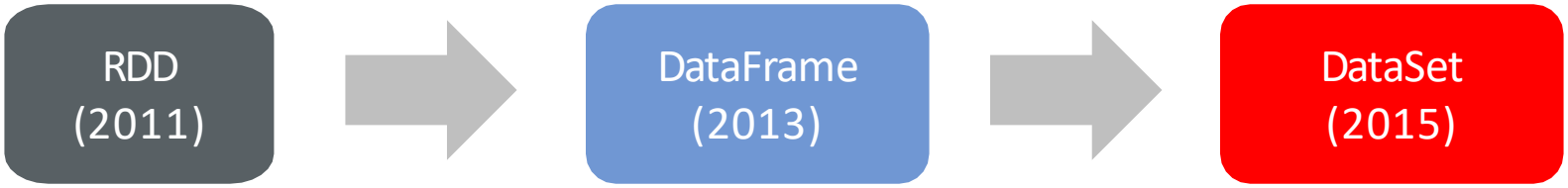
Pipelining

When possible, Spark will perform sequences of Transformations by row so no data is stored

```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line =>
  line.toUpperCase())
> val mydata_filt = mydata_uc.filter(line
  => line.startsWith("I"))
> mydata_filt.take(2)
I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
```



History of Spark APIs



- Distribute collection of JVM objects
- Functional Operators (map, filter, etc.)

- Distribute collection of Row objects
- Expression-based operations and UDFs
- Logical plans and optimizer
- Fast/efficient internal representations

- Internally rows, externally JVM objects
- "Best of both worlds" type safe + fast

DataSets and DataFrames

- RDD will remain the low-level API in Spark
- In 2015 DataFrames & DataSets as structured data APIs
  - DataFrames are collections of rows with a schema
  - Datasets add static types, e.g. Dataset[Person]
  - Both run on Tungsten

- Spark 2.0 merges these APIs: DataFrame = DataSet[Row]
- Datasets & DataFrames give richer semantics and optimizations
  New libraries will increasingly use these as interchange format
  Examples: Structured Streaming, MLlib, GraphFrames

RDD and DataFrames

dept	age	name
Bio	48	H Smith
CS	54	A Turing
Bio	43	B Jones
Chem	61	M Kennedy

Data grouped into named columns

RDD API

```
pdata.map(lambda x: (x.dept, [x.age, 1])) \
  .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
  .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
  .collect()
```

DataFrame API

```
data.groupBy("dept").avg("age")
```

Write Less Code: Compute an Average

```
private IntWritable one = new IntWritable(1);
private IntWritable output = new IntWritable();
protected void map(LongWritable key,
                   Text value,
                   Context context) {
    String[] fields = value.split("\t");
    output.set(Integer.parseInt(fields[1]));
    context.write(one, output);
}

-----------------------------------------------

IntWritable one = new IntWritable(1);
DoubleWritable average = new DoubleWritable();

protected void reduce(IntWritable key,
                      Iterable<IntWritable> values,
                      Context context) {
    int sum = 0;
    int count = 0;
    for (IntWritable value: values) {
        sum += value.get();
        count++;
    }
    average.set(sum / (double) count);
    context.write(key, average);
}
```

```
rdd = sc.textFile(...).map(_.split(" "))
rdd.map { x => (x(0), (x(1).toFloat, 1)) }.
    reduceByKey { case ((num1, count1), (num2, count2)) =>
     (num1 + num2, count1 + count2)
    }.
    map { case (key, (num, count)) => (key, num / count)
    }.
    collect()
```

```
rdd = sc.textFile(...).map(lambda s: s.split())
rdd.map(lambda x: (x[0], (float(x[1]), 1))).\
    reduceByKey(lambda t1, t2: (t1[0] + t2[0], t1[1] +
    t2[1])).\
    map(lambda t: (t[0], t[1][0] / t[1][1])).\
    collect()
```

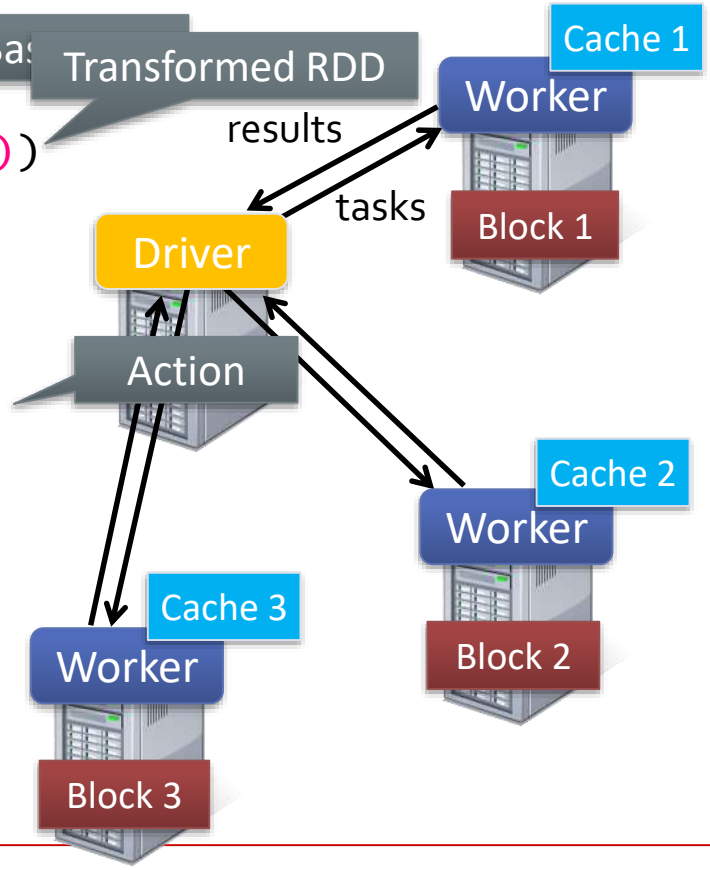
Using DataFrames

```
import org.apache.spark.sql.functions._
val df = rdd.map(a => (a(0), a(1))).toDF("key", "value")
df.groupBy("key")
  .agg(avg("value"))
  .collect()
```

Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```



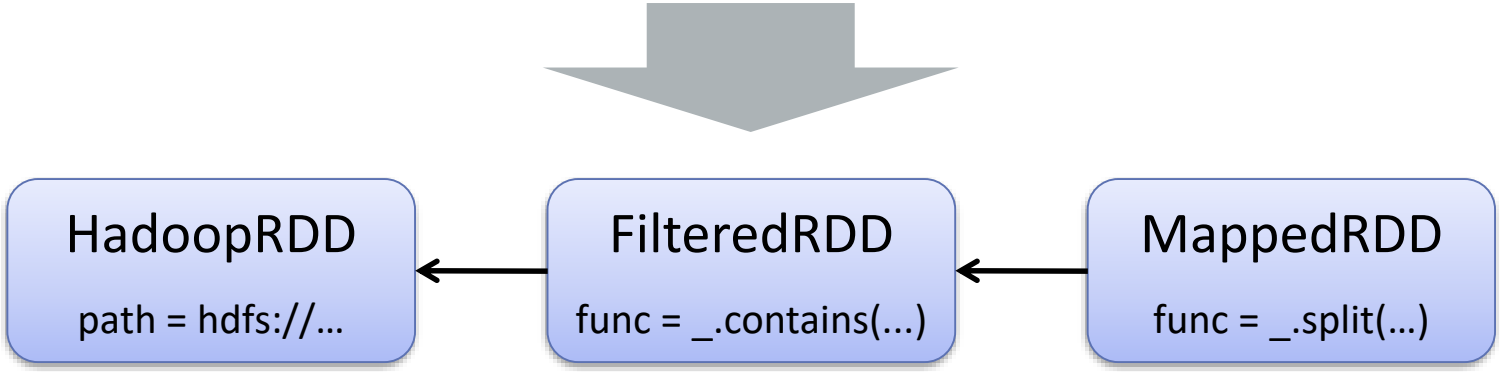
Result: scaled to 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)

Fault Tolerance

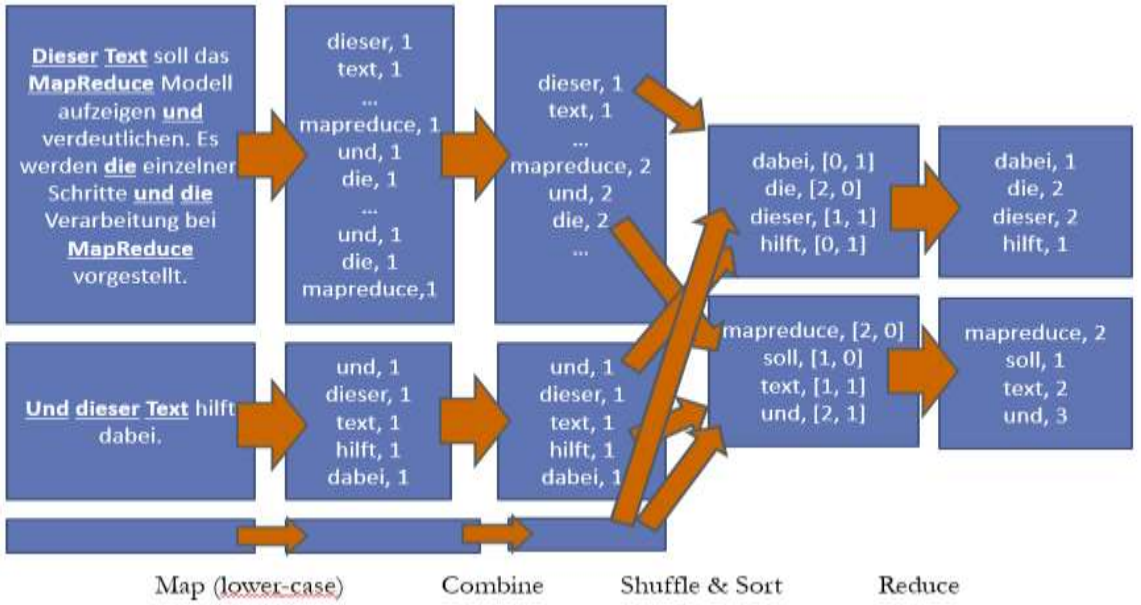
RDDs track the series of transformations used to build them (their *lineage*) to recompute lost data

E.g:

```
messages = textFile(...).filter(_.contains("error"))
                        .map(_.split('\t')(2))
```

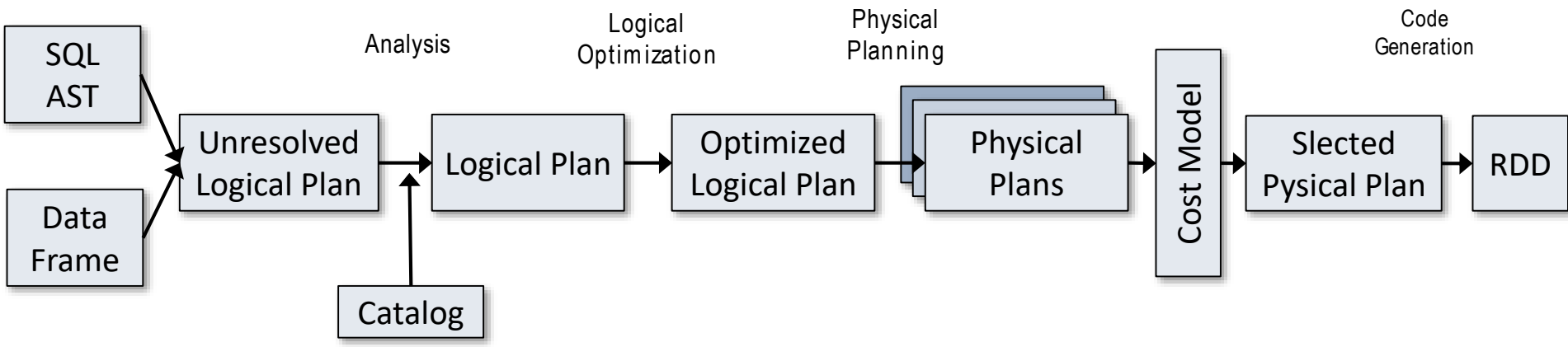


Example: Word Count




```
JavaRDD<String> rdd = sc.textFile("s3://...")
JavaRDD<String> words = rdd.flatMap(new FlatMapFunction<String, String>() {
  public Iterable<String> call(String x) { return Arrays.asList(x.split(" ")); }
});
JavaPairRDD<String, Integer> result = words.mapToPair(
  new PairFunction<String, String, Integer>() {
    public Tuple2<String, Integer> call(String x) { return new Tuple2(x, 1); }
}).reduceByKey(
  new Function2<Integer, Integer, Integer>() {
    public Integer call(Integer a, Integer b) { return a + b; }
});
```

Plan Optimization & Execution



DataFrames and SQL share the same optimization/execution pipeline.

Outline

- Spark
- RDD, DataFrame and DataSet
- Using Spark

Using Spark

Interactive

- Scala or Python Shell `./bin/spark-shell` or `./bin/pyspark`
- Zeppelin Notebook

Application

- Submit Java / Scala Application to Cluster
  `spark-summit`

Spark UI

During Spark's execution of the previous code block, users can monitor the progress of their job through the Spark UI. If you are running in local mode this will just be the http://localhost:4040.

Spark Shell

```
[strahasch@bda-job bin]$ pwd
/usr/hdp/current/spark2-client/bin
[strahasch@bda-job bin]$ ll
total 40
-rwxr-xr-x. 1 root root 1089 Aug 26 11:47 beeline
-rwxr-xr-x. 1 root root 1933 Aug 26 11:47 find-spark-home
-rwxr-xr-x. 1 root root 2493 Aug 26 11:47 load-spark-env.sh
-rwxr-xr-x. 1 root root 2989 Aug 26 11:47 pyspark
-rwxr-xr-x. 1 root root 1030 Aug 26 11:47 run-example
-rwxr-xr-x. 1 root root 3806 Aug 26 11:47 spark-class
-rwxr-xr-x. 1 root root 1039 Aug 26 11:47 sparkR
-rwxr-xr-x. 1 root root 3017 Aug 26 11:47 spark-shell
-rwxr-xr-x. 1 root root 1065 Aug 26 11:47 spark-sql
-rwxr-xr-x. 1 root root 1040 Aug 26 11:47 spark-submit
[strahasch@bda-job bin]$ ./spark-shell
```

- The Spark Shell provides interactive data exploration (REPL)
- Writing Spark applications without the shell will be covered later

```
Spark context Web UI available at http://141.79.64.91:4041
Spark context available as 'sc' (master = local[*], app id = local-1511799975823).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.1.2.6.2.0-205
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_112)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.1.2.6.2.0-205
      /_/

Using Python version 2.7.5 (default, Aug  4 2017 00:39:18)
SparkSession available as 'spark'.
>>>
```

Spark Context

- Every Spark application requires a Spark Context
- The main entry point to the Spark API
- Spark Shell provides a preconfigured Spark Context called sc

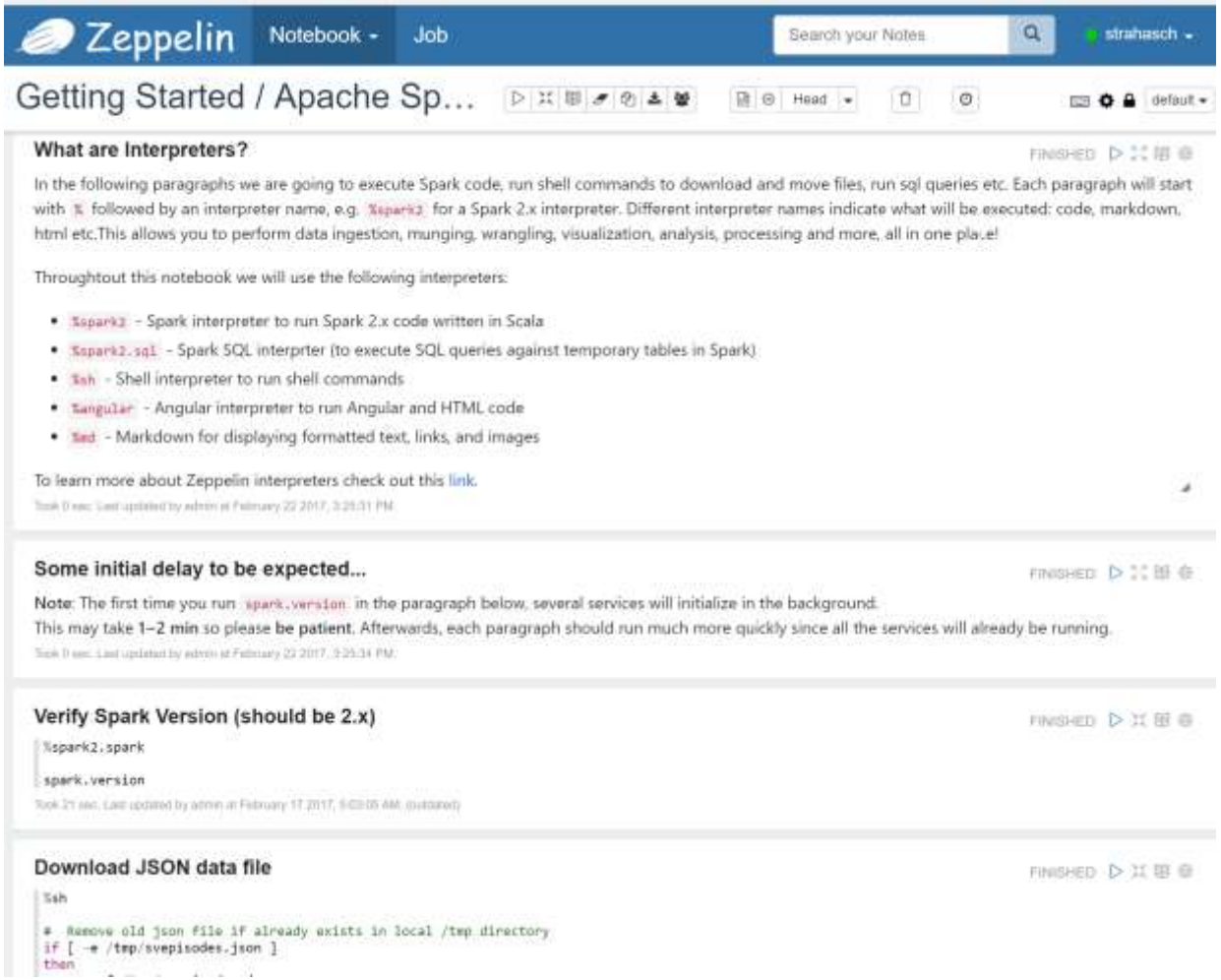
Python

```
Using Python version 2.7.8 (default, Aug 27 2015 05:23:36)
SparkContext available as sc, HiveContext available as sqlCtx.

>>> sc.appName
u'PySparkShell'
```

Scala

```
...
Spark context available as sc.
SQL context available as sqlContext.

scala> sc.appName
res0: String = Spark shell
```

Zeppelin Notebooks



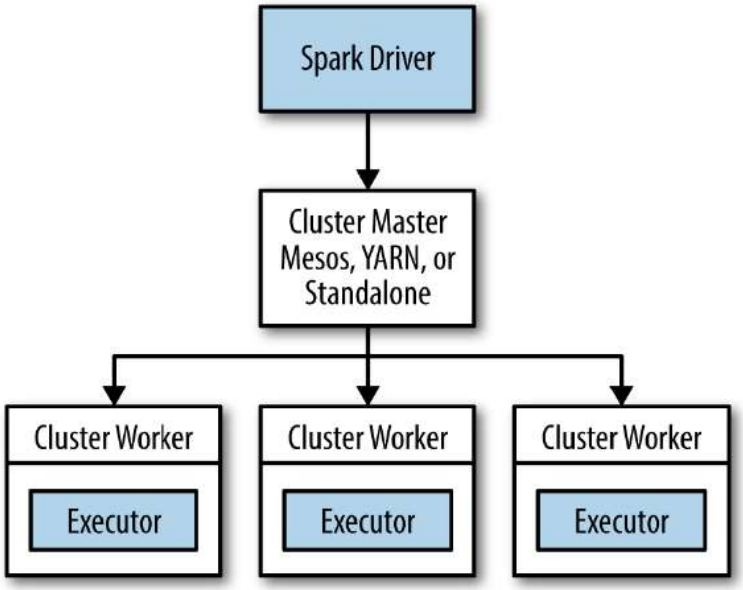
- WebUI for reproducible data analysis
- You can mix code, results, visualization and documentation.

Spark Application

Spark Applications consist of
- A driver process and
- a set of executor processes.

Driver process runs on a node in the cluster and is responsible for

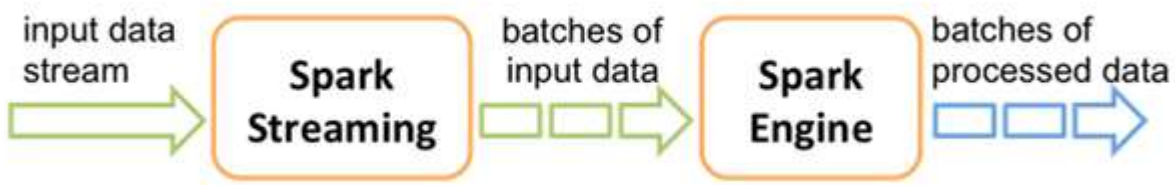
1. maintaining information about the application
2. responding to a user's program
3. analyzing, distributing, and scheduling work across the executors.



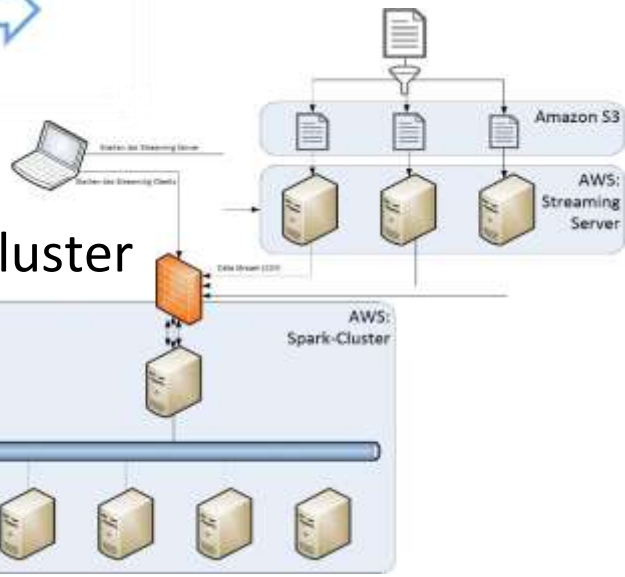
Spark, in addition to its cluster mode, also has a local mode. The driver and executors are simply processes, this means that they can live on a single machine or multiple machines.
In local mode, these run (as threads) on your individual computer instead of a cluster.

Streaming Data and Machine Learning

- Von Batch zum Streaming
  - Micro Batching



- Senden von Daten per Socket-Stream
  - Entgegennehmen und Verarbeiten im Spark-Cluster
    - Online-Analysen
    - Visualisierung

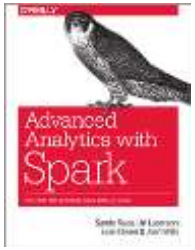


- DEBS Challenge Daten analysieren (MLib)
  - 100 Mio. Energieverbrauchs-Datensätze von Haushalten
  - Ausreißer erkennen / Clustering von Verbrauchern

Literature and Links

- https://databricks.com/

- Learning Spark. Holden Karau.



- Advanced Analytics with Spark. Sandy Ryza.
- Spark Online Documentation
  - https://spark.apache.org/docs/latest/

- Spark Download
  - https://spark.apache.org/downloads.html