An Introduction to Spark and to its Programming Model

Amy Krause, Andreas Vroutsis

EPCC, The University of Edinburgh

Introduction to Apache Spark

- Fast, expressive cluster computing system compatible with Apache Hadoop
- It is much faster and much easier than Hadoop MapReduce to use due its rich APIs
- Large community
- Goes far beyond batch applications to support a variety of workloads:
  - including interactive queries, streaming, machine learning, and graph processing

Figure: *Real Time Processing In Spark*

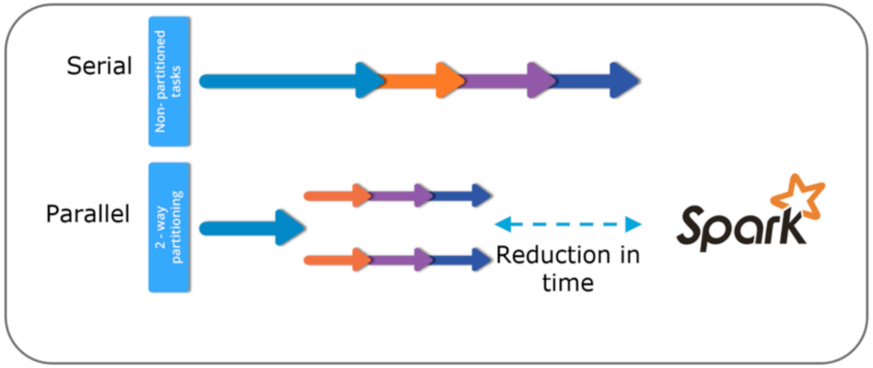
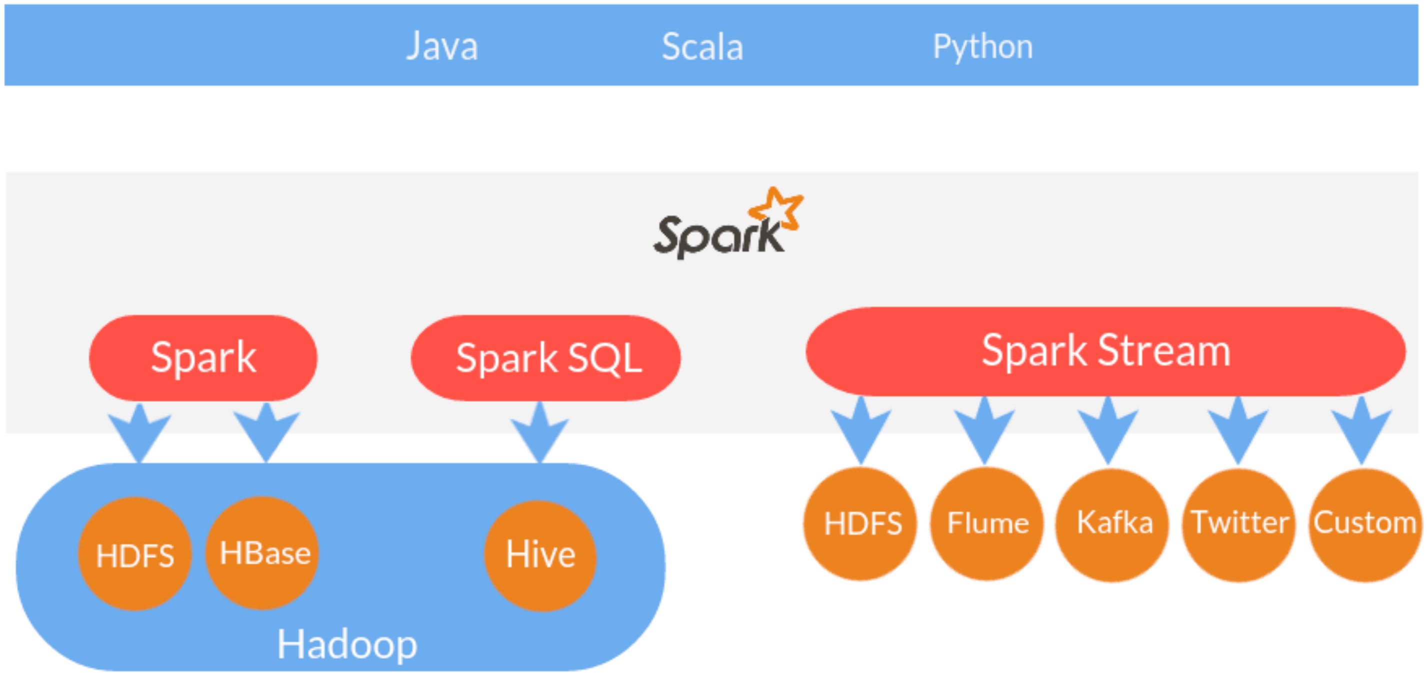


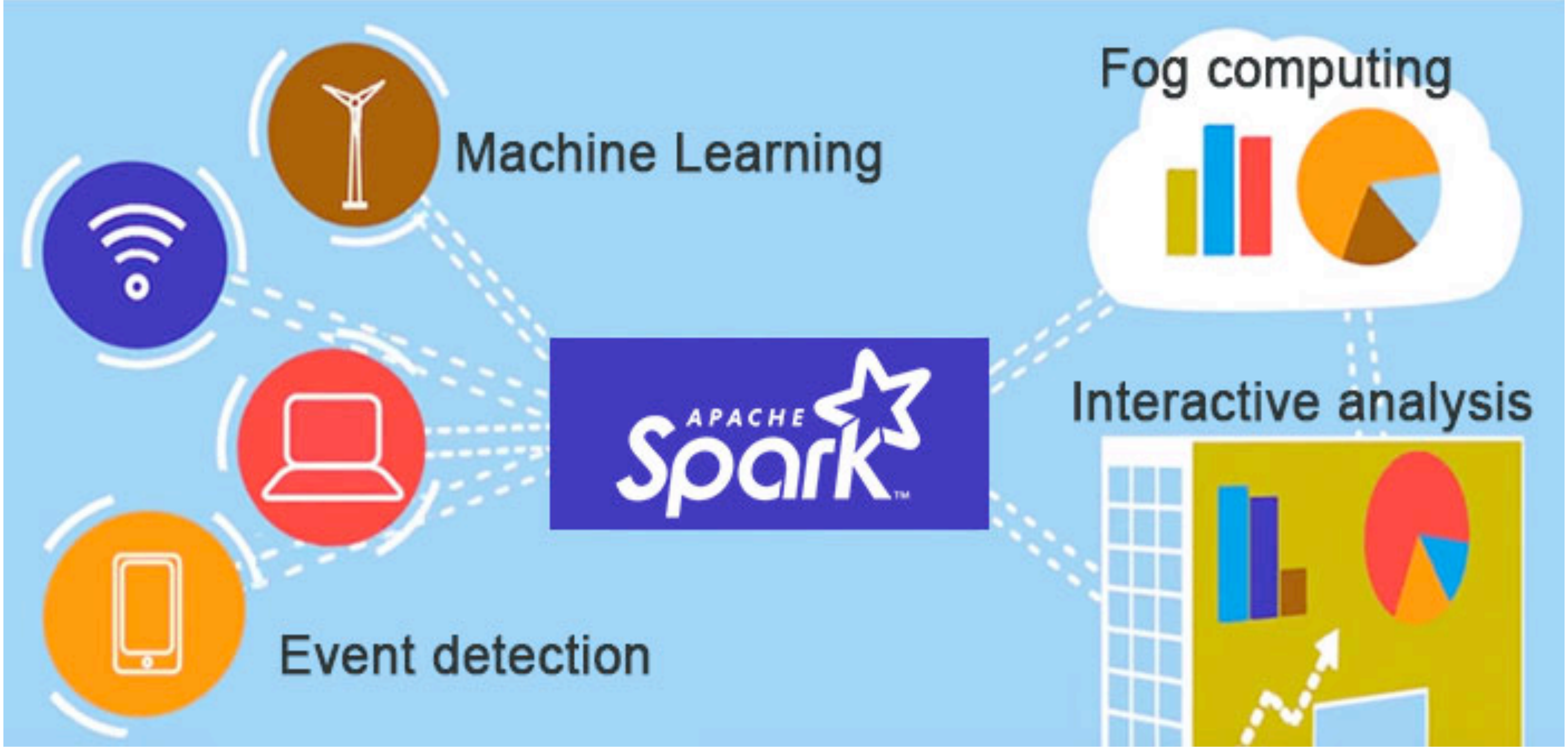
Figure: *Data Parallelism In Spark*

Introduction to Apache Spark

- General-purpose cluster in-memory computing system
- Provides high-level APIs in Java, Scala, python



Apache Spark Applications



NETFLIX

Uses Spark Streaming to provide the best-in-class movie streaming and recommendation tool to its users.

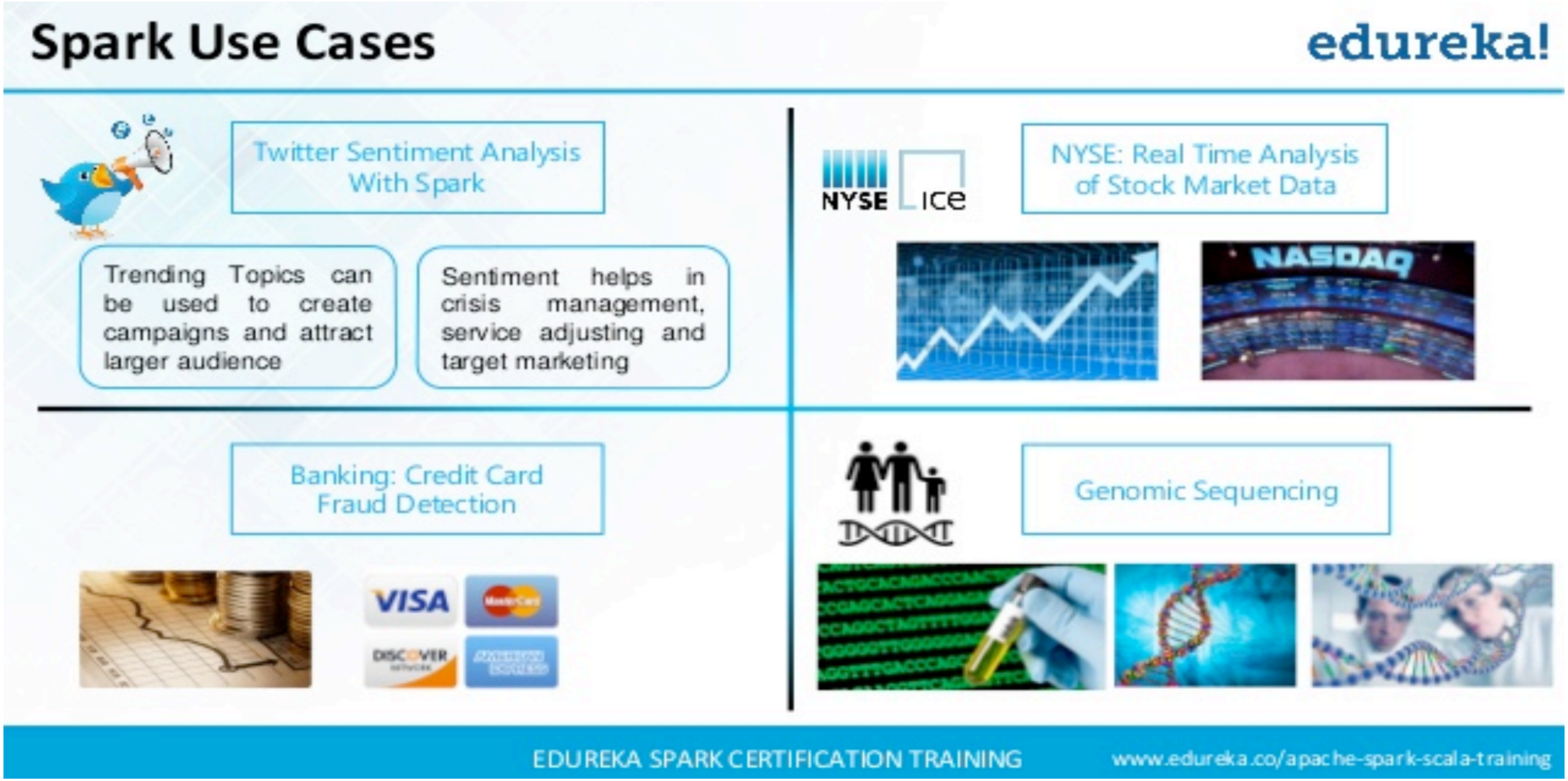
UBER

Uses Spark to collect TBs of raw and unstructured data every day from its users to convert it into structured data. This makes it ready for further complex analytics.

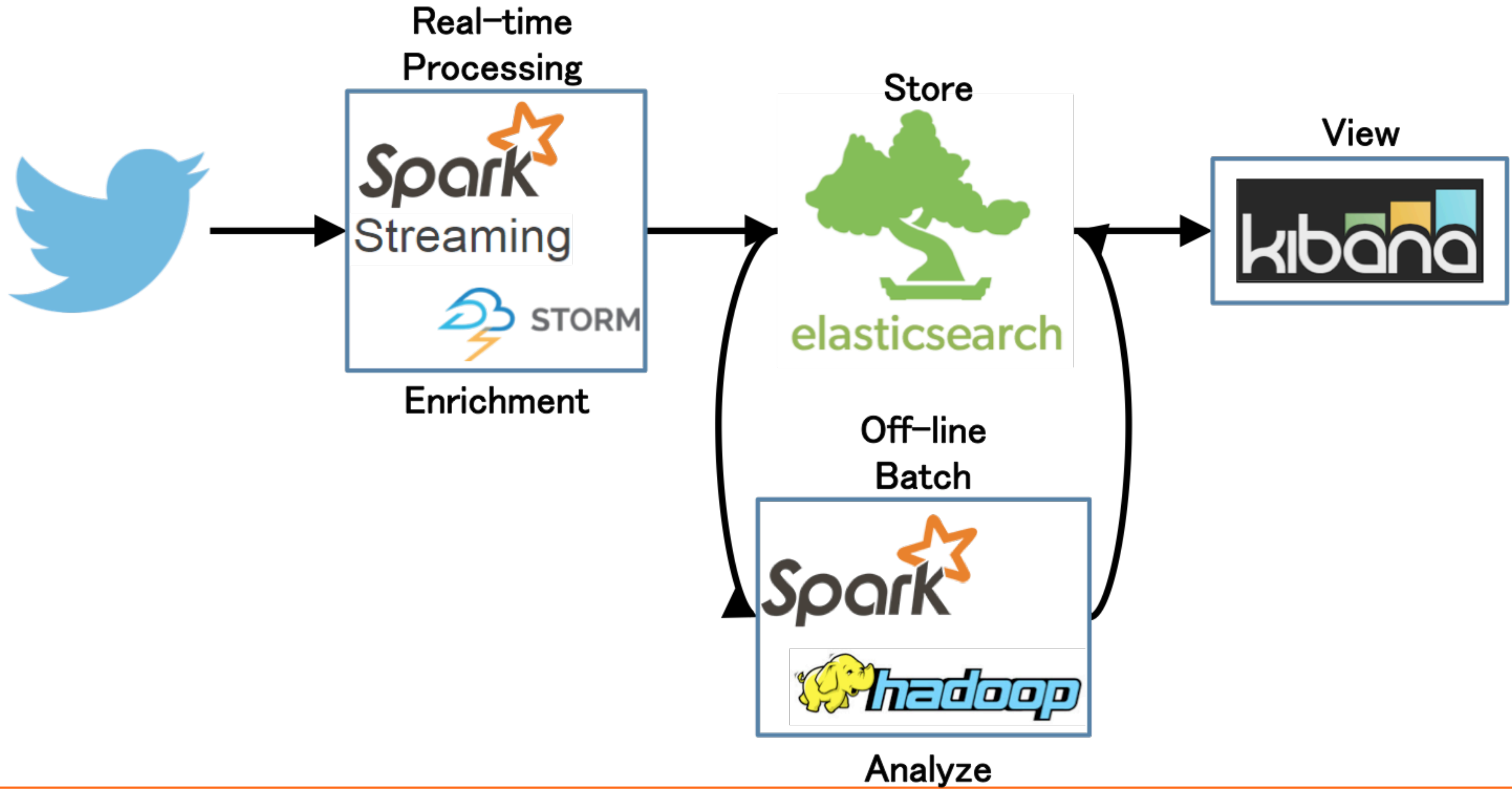
Pinterest

Feeds real-time data into Spark via Spark Streaming to get instant insights on how users are engaging with Pins globally. This makes Pinterest's recommendations (i.e. to show Pins) to be accurate.

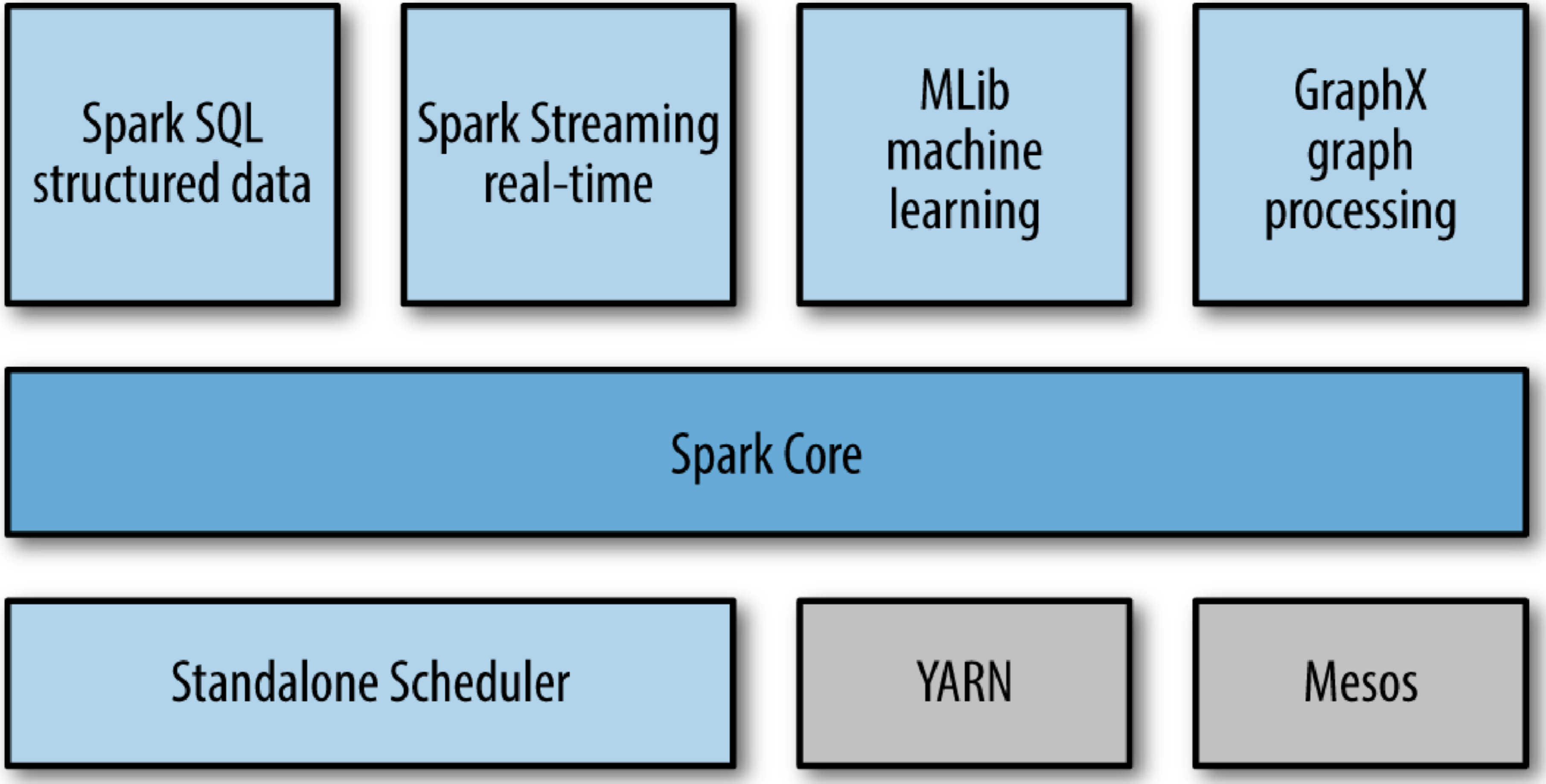
Use cases


Spark Use Cases
edureka!
Twitter Sentiment Analysis With Spark
Trending Topics can be used to create campaigns and attract larger audience
Sentiment helps in crisis management, service adjusting and target marketing
NYSE
ICE
NYSE: Real Time Analysis of Stock Market Data
NASDAQ
Banking: Credit Card Fraud Detection
VISA
DISCOVER
Genomic Sequencing
EDUREKA SPARK CERTIFICATION TRAINING
www.edureka.co/apache-spark-scala-training

Real Time Data Architecture for analyzing tweets
- Twitter Sentiment Analysis



Spark Ecosystem



PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE

Spark Core

- Contains the basic functionality for
  - task scheduling,
  - memory management,
  - fault recovery,
  - interacting with storage systems,
  - and more.
- Defines the Resilient Distributed Data sets (RDDs)
  - main Spark programming abstraction.

Spark SQL and Data Frames

- For working with structured data
- View datasets as relational tables
- Define a schema of columns for a dataset
- Perform SQL queries
- Supports many sources of data
  - Hive tables, Parquet and JSON
- We will be looking at this in the DataFrames practical

Spark Streaming

- Data analysis of streaming data
  - e.g. log files generated by production web servers
- Aimed at high-throughput and fault-tolerant stream processing
- Dstream: stream of datasets that contain data from certain intervals
- Usually applied to time series data; intervals are time windows (1 second; 1 minute etc)
- Joins on unlimited streams are impossible so Spark Streaming joins within a time window, or a static dataset with a stream

Spark MLlib

- MLlib is a library that contains common Machine Learning (ML) functionality:
  - Basic statistics
  - Classification (Naïve Bayes, decision tress, LR)
  - Clustering (k-means, Gaussian mixture, …)
  - And many others!
- All the methods are designed to scale out across a cluster.

Spark GraphX

- Graph Processing Library
- Defines a graph abstraction
  - Directed multi-graph
  - Properties attached to each edge and vertex
  - RDDs for edges and vertices
- Provides various operators for manipulating graphs (e.g. subgraph and mapVertices)

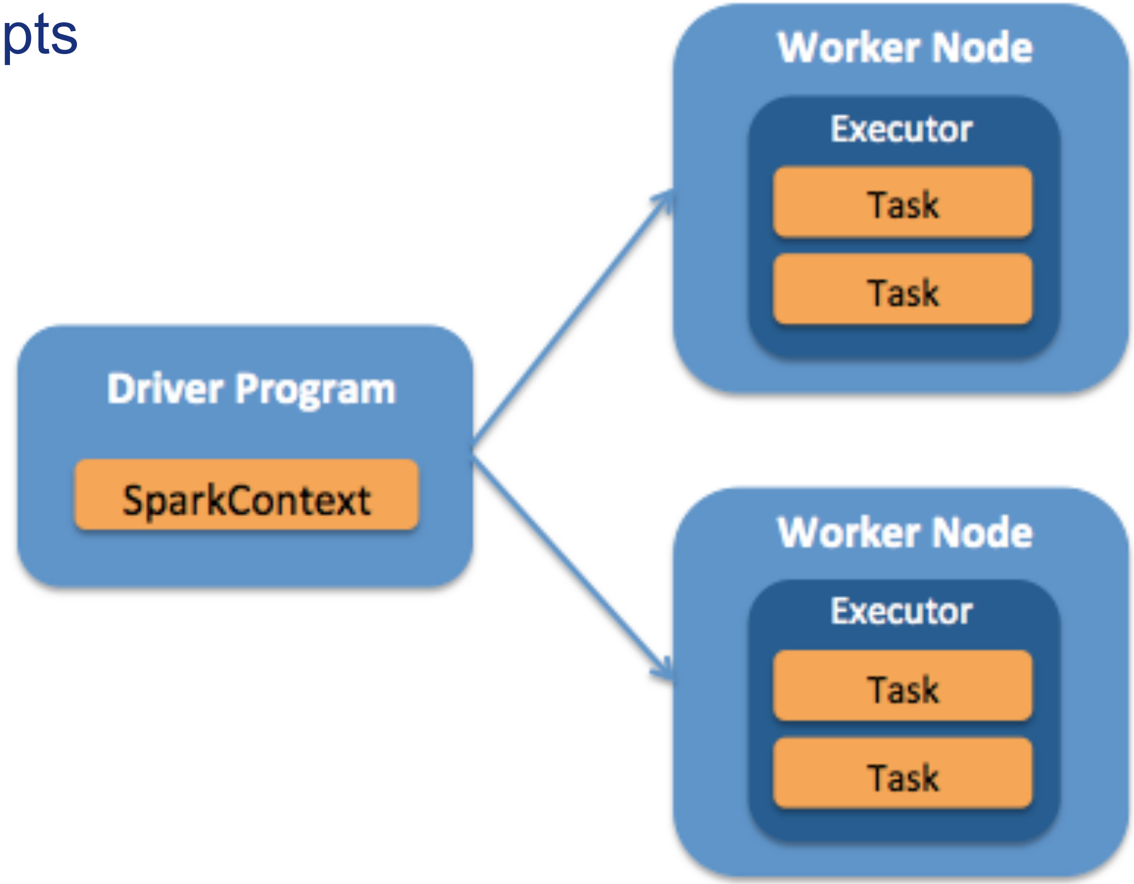
Programming Spark in Python (pySpark)

- We will use Python's interface to Spark called **pySpark**
- **A driver program** accesses the Spark environment through a **SparkContext** objet
- They **key concept** in Spark are datasets called **RDD**s (**R**esilient **D**istributed Dateset)
- Basic idea: We load our data into RDDs and perform some **operations**

Programming environment - Spark concepts

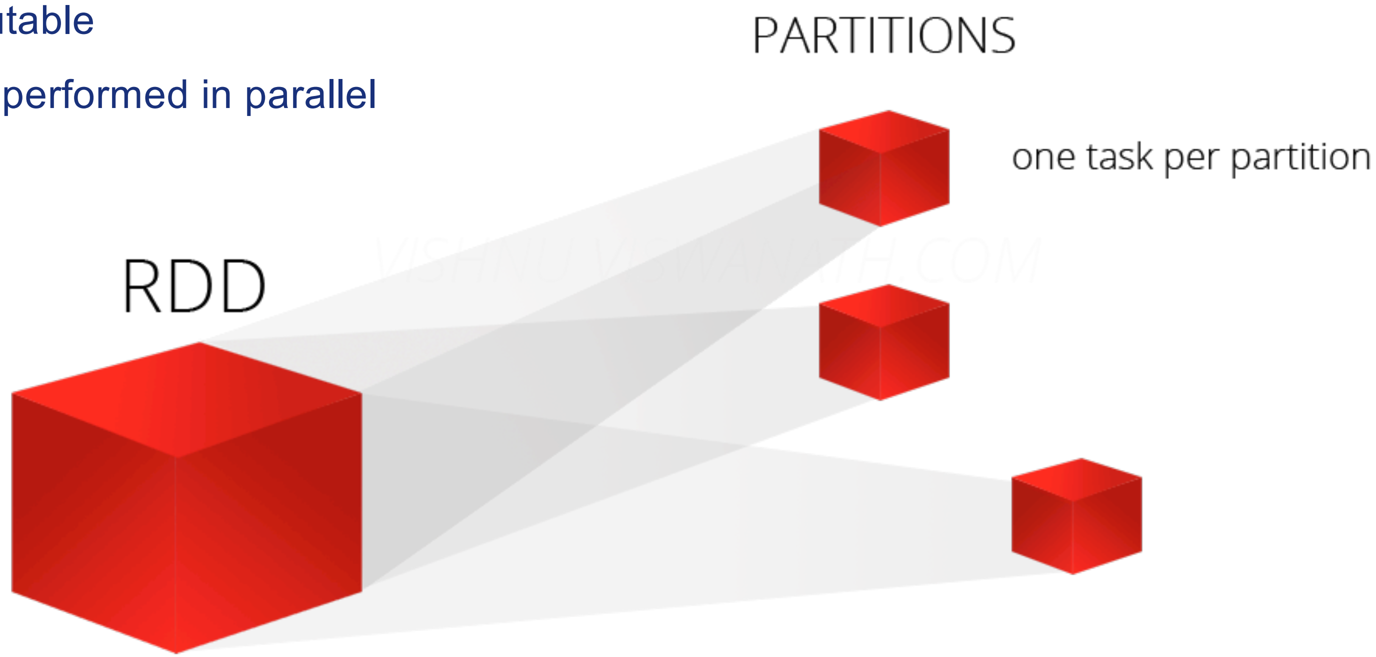
- Driver programs access Spark through a **SparkContext** object which represents a connection to the computing cluster.
- In a *shell* the **SparkContext** is created for you and available as the variable **sc**.
- You can use it to build **Resilient Distributed Data (RDD)** objects.
- Driver programs manage a number of *worker nodes* called *executors.*

Programming environment: Spark concepts



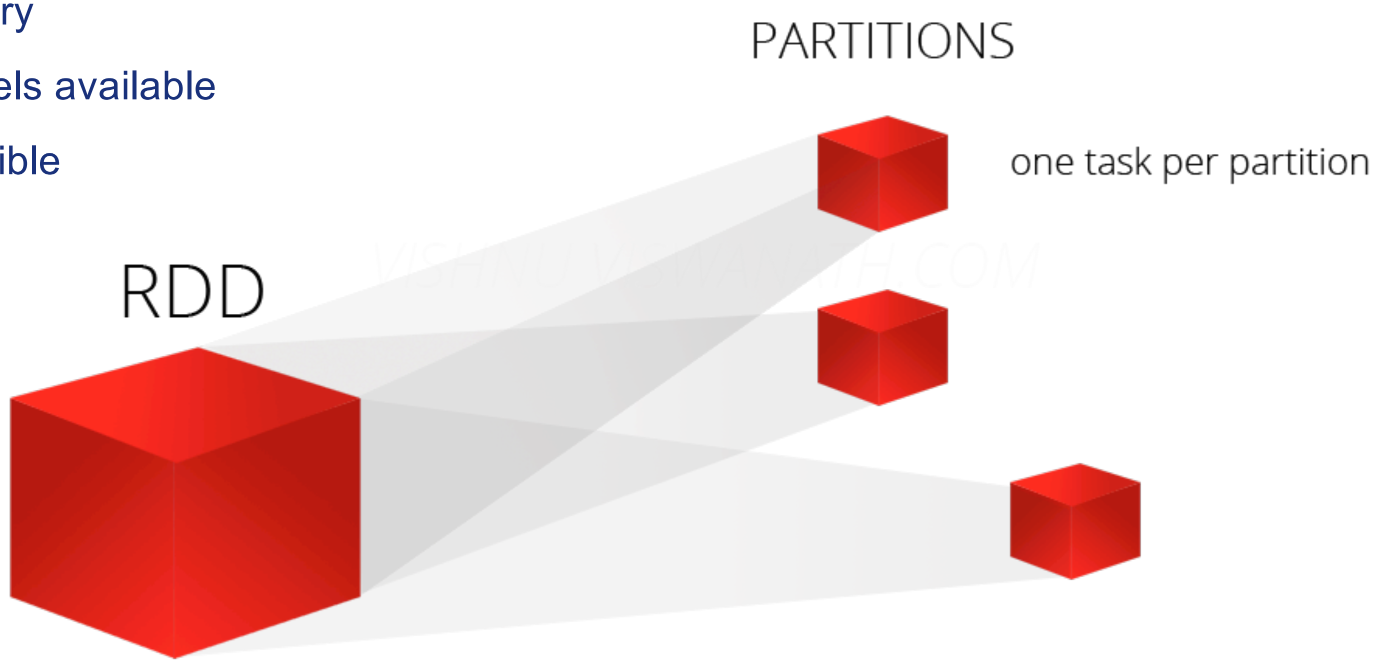
RDD abstraction

- Represents data or transformations on data
- A distributed collection of items - **partitions**
- Read-only: they are immutable
- Enables operations to be performed in parallel



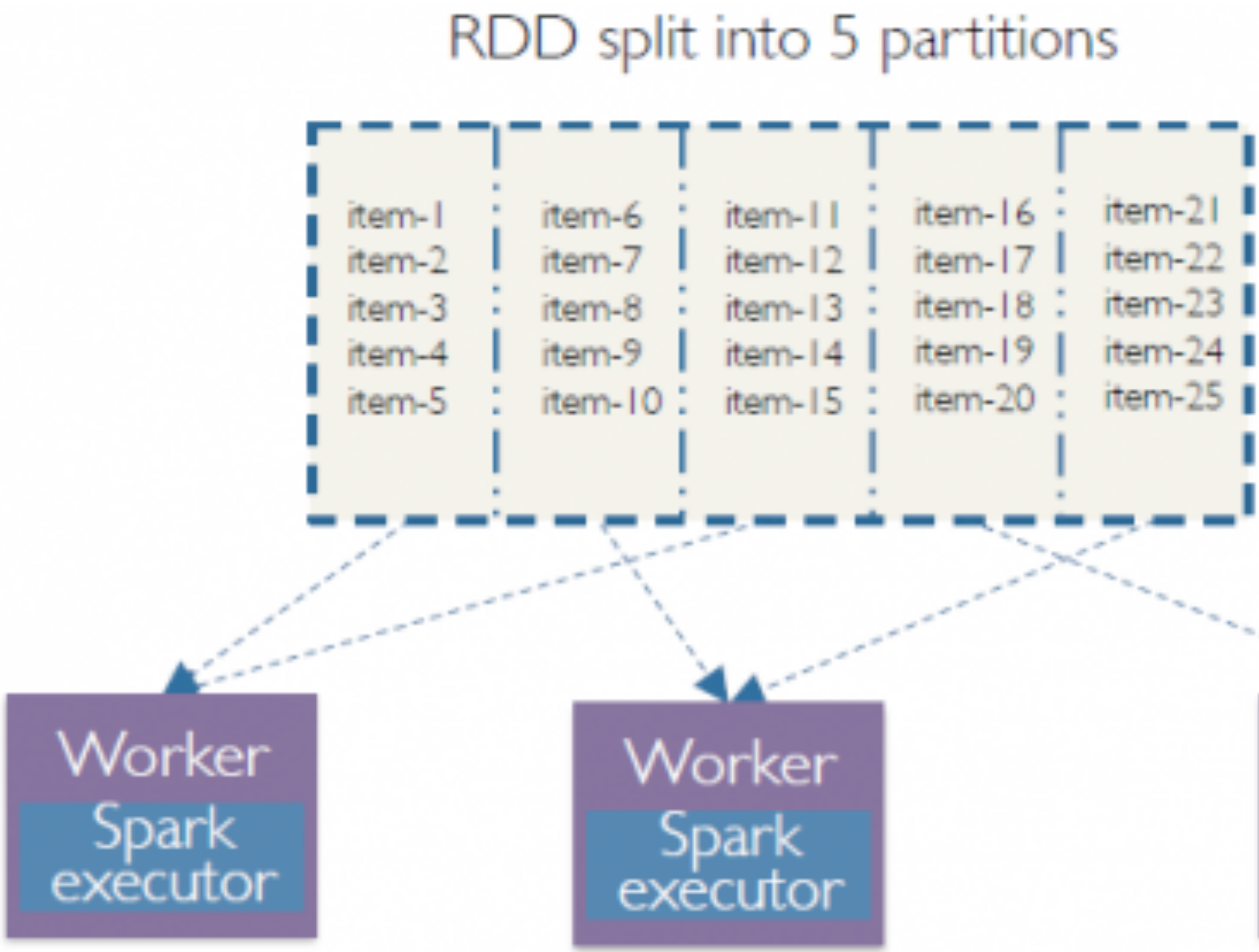
RDD abstraction

- Fault tolerant:
  - Lineage of data is preserved, so data can be re-created on a new node at any time
- Caching dataset in memory
  - different storage levels available
  - fallback to disk possible



Programming with RDDs

- All work is expressed by either:
  - creating new RDDs
  - transforming existing RDDs
  - calling operations on RDDs to compute a result.
- Distributes the data contained in RDDs across the nodes (executors) in the cluster and parallelizes the operations.
- Each RDD is split into multiple partitions, which can be computed on different nodes of the cluster.


RDD split into 5 partitions
item-1
item-2
item-3
item-4
item-5
item-6
item-7
item-8
item-9
item-10
item-11
item-12
item-13
item-14
item-15
item-16
item-17
item-18
item-19
item-20
item-21
item-22
item-23
item-24
item-25
Worker
Spark executor
Worker
Spark executor
Worker
Spark executor

3 workers

An RDD can be created in 2 ways

- Parallelize a collection

```
# Parallelize in Python
wordsRDD = sc.parallelize([“fish", “cats“, “dogs”])
```

- Take an existing in-memory collection and pass it to SparkContext’s parallelize method
- Not generally used outside of prototyping and testing since it requires entire dataset in memory on one machine

- Read from file

```
# Read a local txt file in Python
linesRDD = sc.textFile("/path/to/README.md")
```

- There are other methods to read data from HDFS, C*, S3, HBase, etc.

First Program!

Context → RDD-1 → RDD-2 →

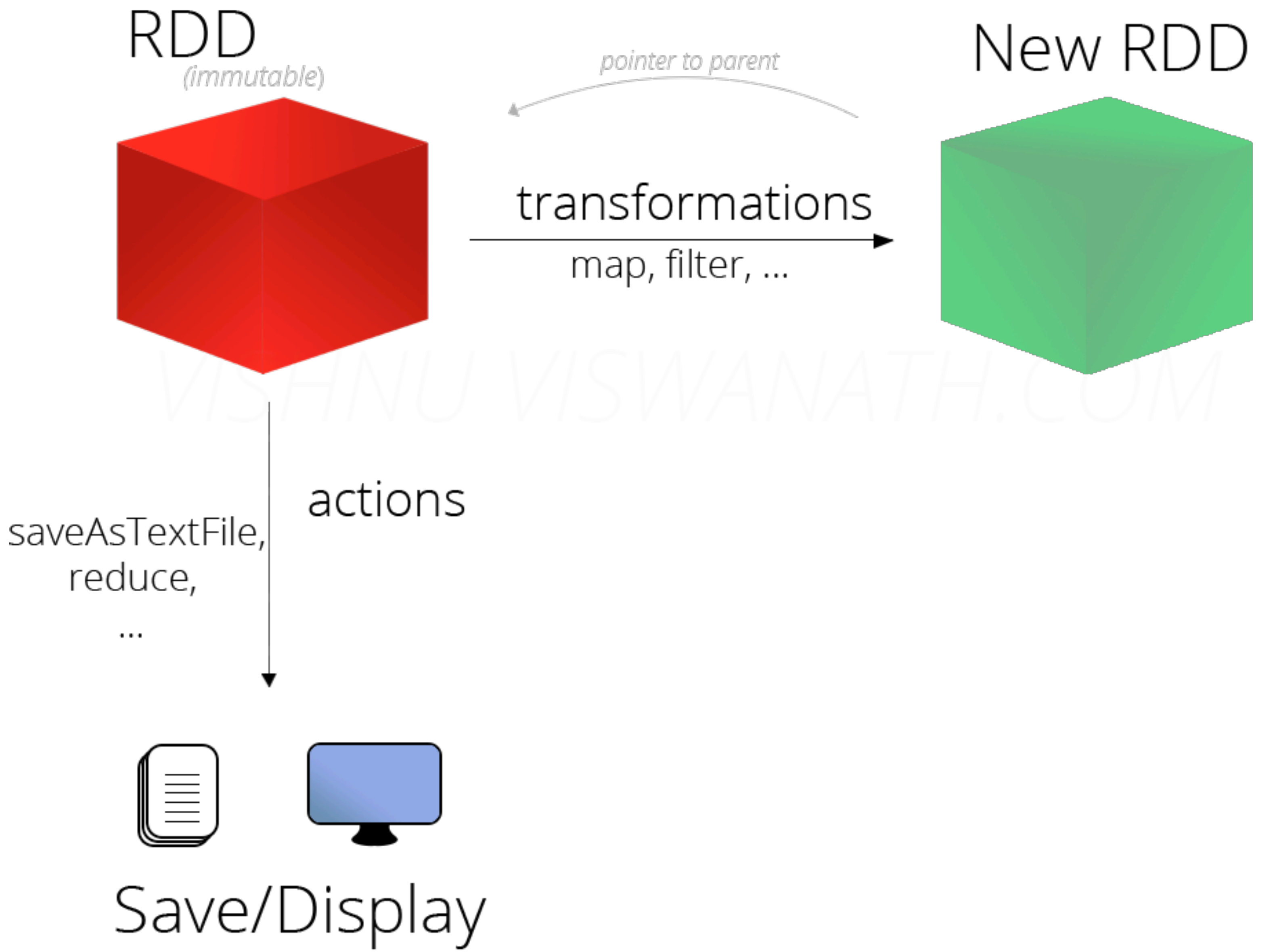
```
sc = SparkContext(master="local[*]")

lines = sc.textFile("README.md", 4)

lines.count()

pythonLines = lines.filter(lambda line : "Python" in line)

pythonLines.first()
```

RDD operations

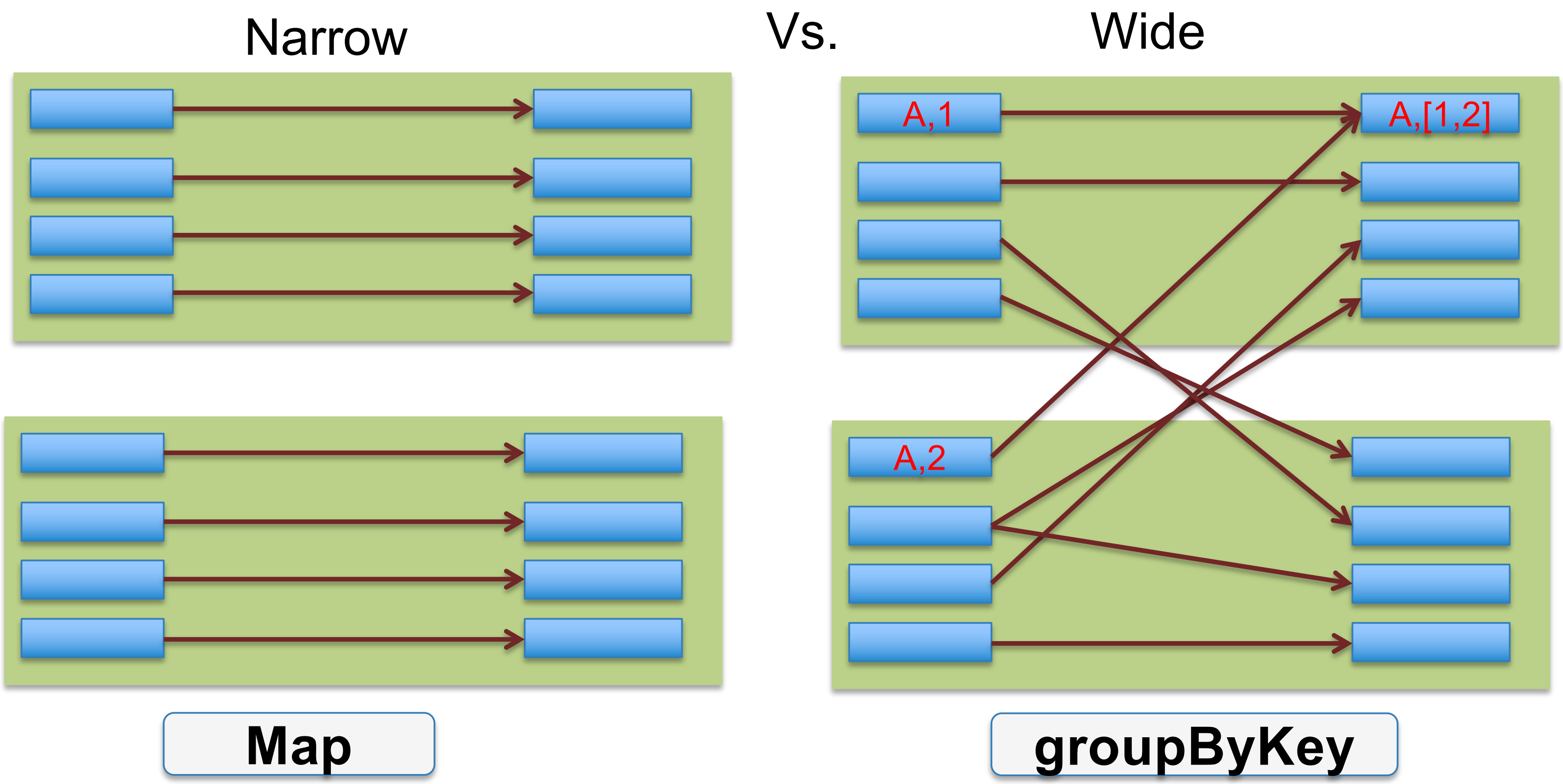
- Once created, RDDs offer two types of operations:
  - **transformations**
    - transformations include map, filter, join
    - lazy operation to build RDDs from other RDDs
  - **actions**
    - actions include count, collect, save
    - return a result or write it to storage

Transformation vs Actions

Transformations	Actions
map (func)	reduce(func)
flatMap(func)	collect()
filter(func)	count()
groupByKey()	first()
reduceByKey(func)	take(n)
mapValues(func)	saveAsTextFile(path)
sample(…)	countByKey()
union(other)	foreach(func)
distinct()	…
sortByKey()	
…	


RDD
(immutable)
pointer to parent
New RDD
transformations
map, filter, ...
actions
saveAsTextFile,
reduce,
...
Save/Display

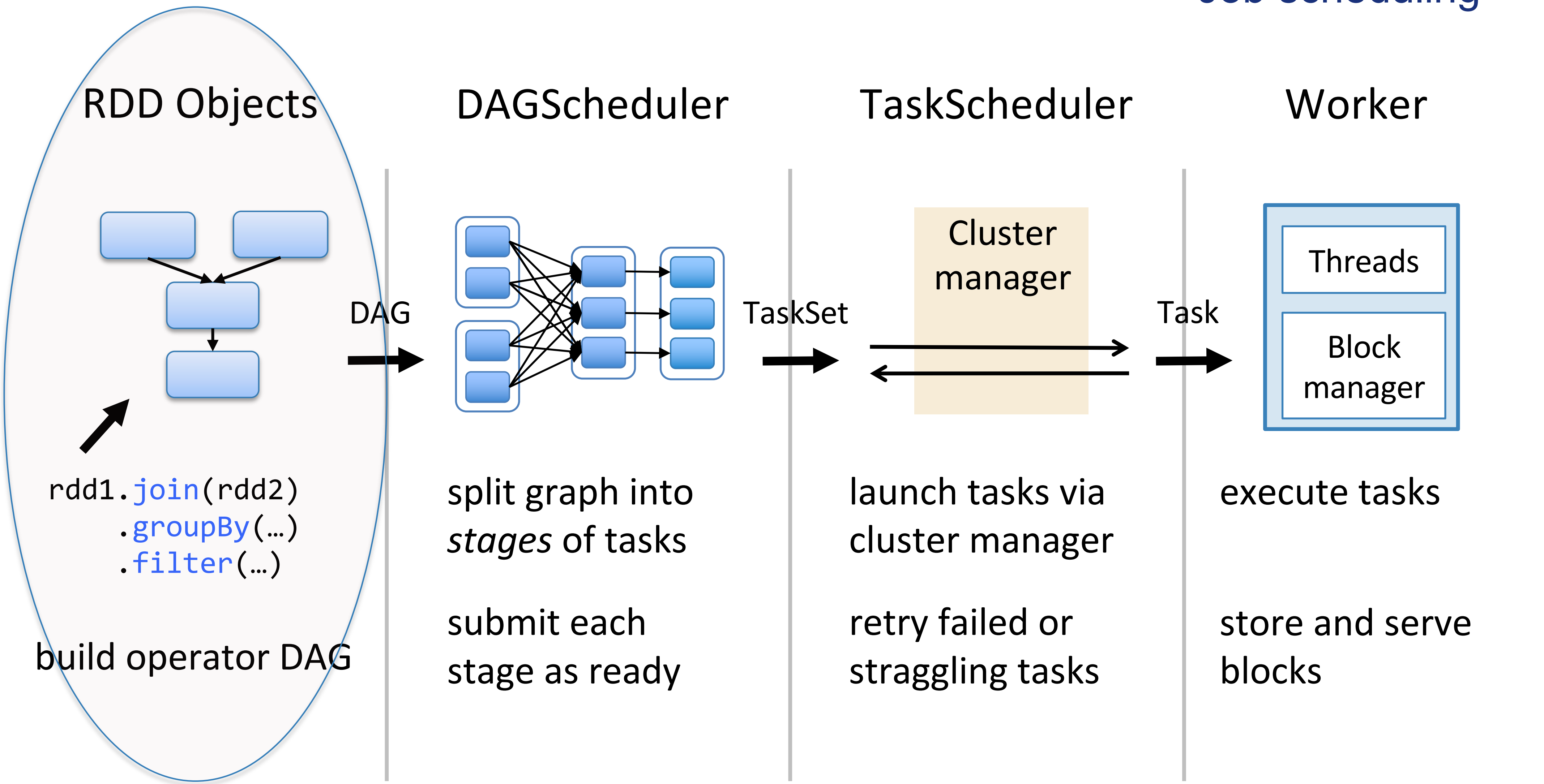
Narrow Vs. Wide transformation



Life cycle of a Spark program

1. Create some **input RDDs** from external data or parallelize a collection in your driver program.
2. Lazily **transform** them to define new RDDs using transformations like filter() or map()
3. Optional: Ask Spark to cache() any intermediate RDDs that will need to be reused.
4. Launch **actions** such as count() and collect() to kick off a parallel computation, which is then optimized and executed by Spark.

Job scheduling



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

Example: Mining Console Logs

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

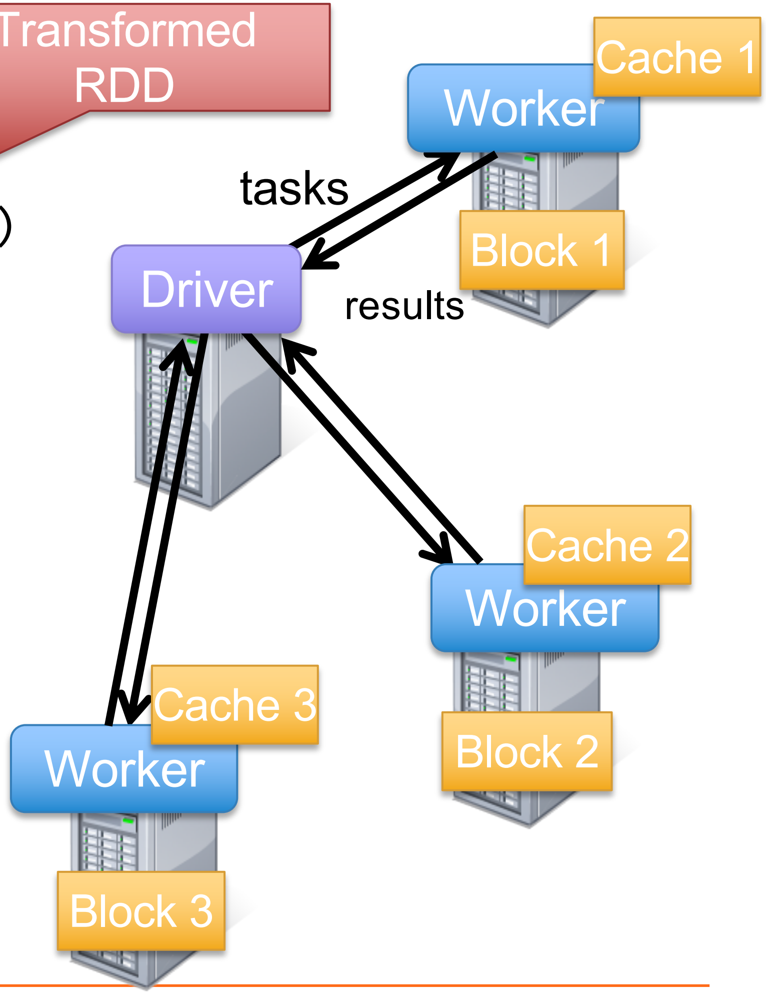
Base RDD

Transformed RDD

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split('\t')[2])
messages.cache()
```

Action

```
messages.filter(lambda s: "foo" in s).count()
messages.filter(lambda s: "bar" in s).count()
. . .
```



Some Apache Spark tutorials

- https://www.cloudera.com/documentation/enterprise/5-7-x/PDF/cloudera-spark.pdf
- https://stanford.edu/~rezab/sparkclass/slides/itas_workshop.pdf
- https://www.coursera.org/learn/big-data-essentials
- https://www.cloudera.com/documentation/enterprise/5-6-x/PDF/cloudera-spark.pdf

Spark: when not to use

- Even though Spark is versatile, that doesn't mean Spark's in-memory capabilities are the best fit for all use cases:
  - For many simple use cases Apache MapReduce and Hive might be a more appropriate choice
  - Spark was not designed as a multi-user environment
  - Spark users are required to know that memory they have is sufficient for a dataset
  - Adding more users adds complications, since the users will have to coordinate memory usage to run code

THANK YOU FOR YOUR ATTENTION

www.prace-ri.eu