Intro to Spark

By Aaron Merlob (edited and adapted to python by Ryan Henning)

Agenda

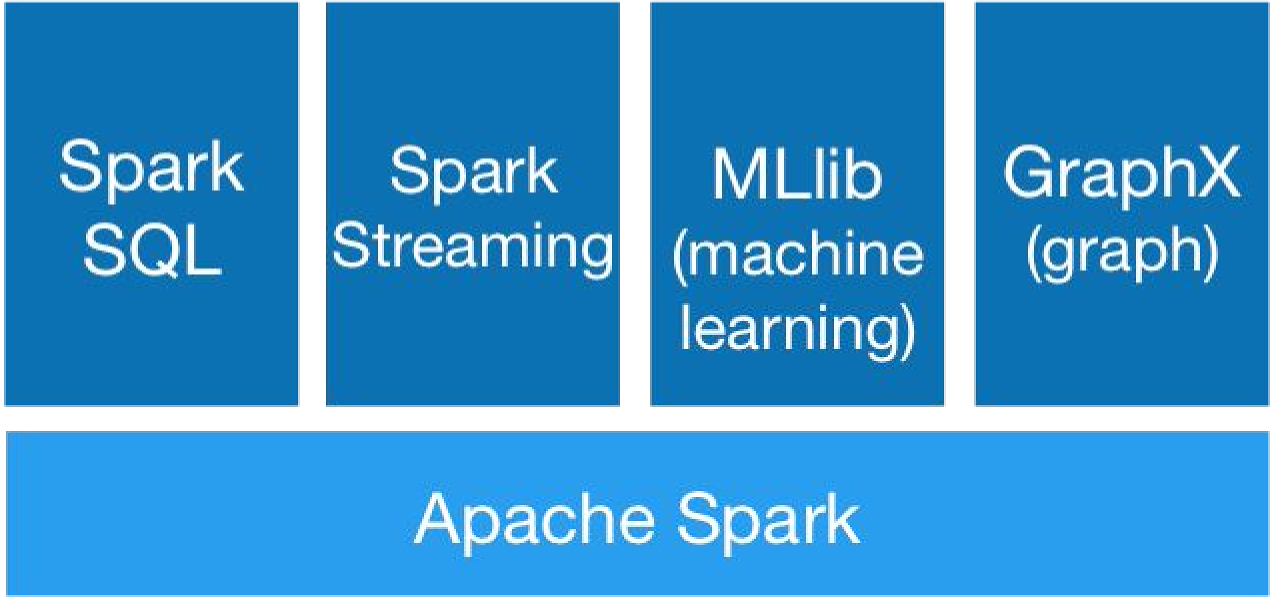
- About Apache Spark
- Resilient Distributed Dataset (RDD):
  - Features & Implementation
- Sample Functions:
  - Transformations and Actions
- Pair RDDs & More Sample Functions
- Code Samples & Mini-Quizzes

Apache Spark

- Open-source cluster computing framework
- “Successor” to Hadoop MapReduce
- Supports Scala, Java, and Python and R!

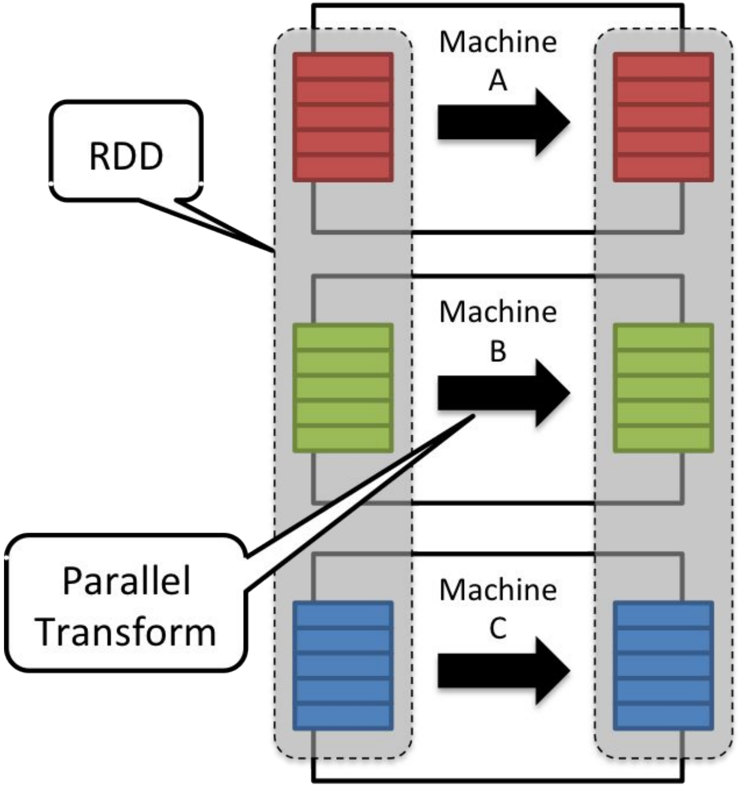
https://en.wikipedia.org/wiki/Apache_Spark

Spark Core + Libraries



https://spark.apache.org

Resilient ***Distributed*** Dataset



http://horicky.blogspot.com/2013/12/spark-low-latency-massively-parallel.html

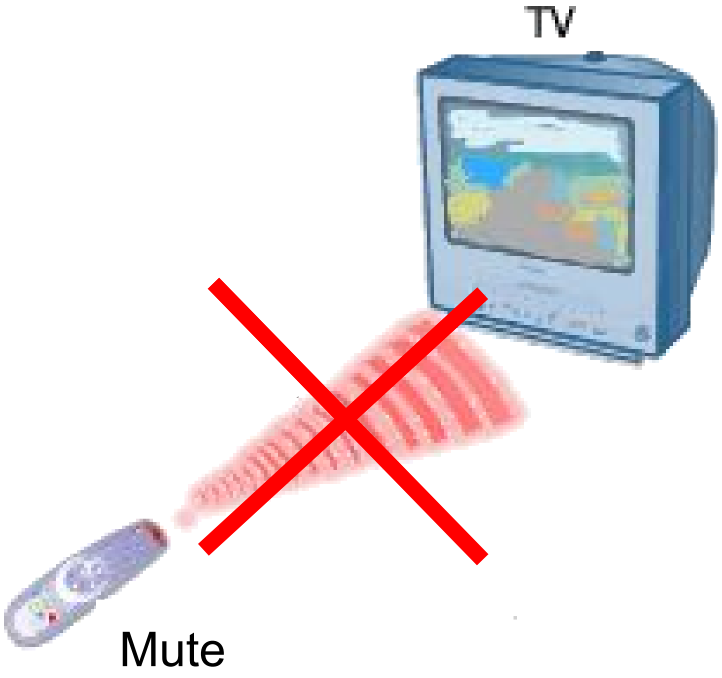
RDD - Main Abstraction

Resilient Distributed Dataset (RDD)

- Distributed Collection
- Fault-tolerant
- Parallel operation - Partitioned
- Many data sources

… warning: bad puns coming (blame Aaron)

RDDs are Immutable



Immutable
Lazily Evaluated
Cachable

RDDs are Lazily Evaluated

How Good Is Aaron's Presentation?

Immutable
Lazily Evaluated
Cachable

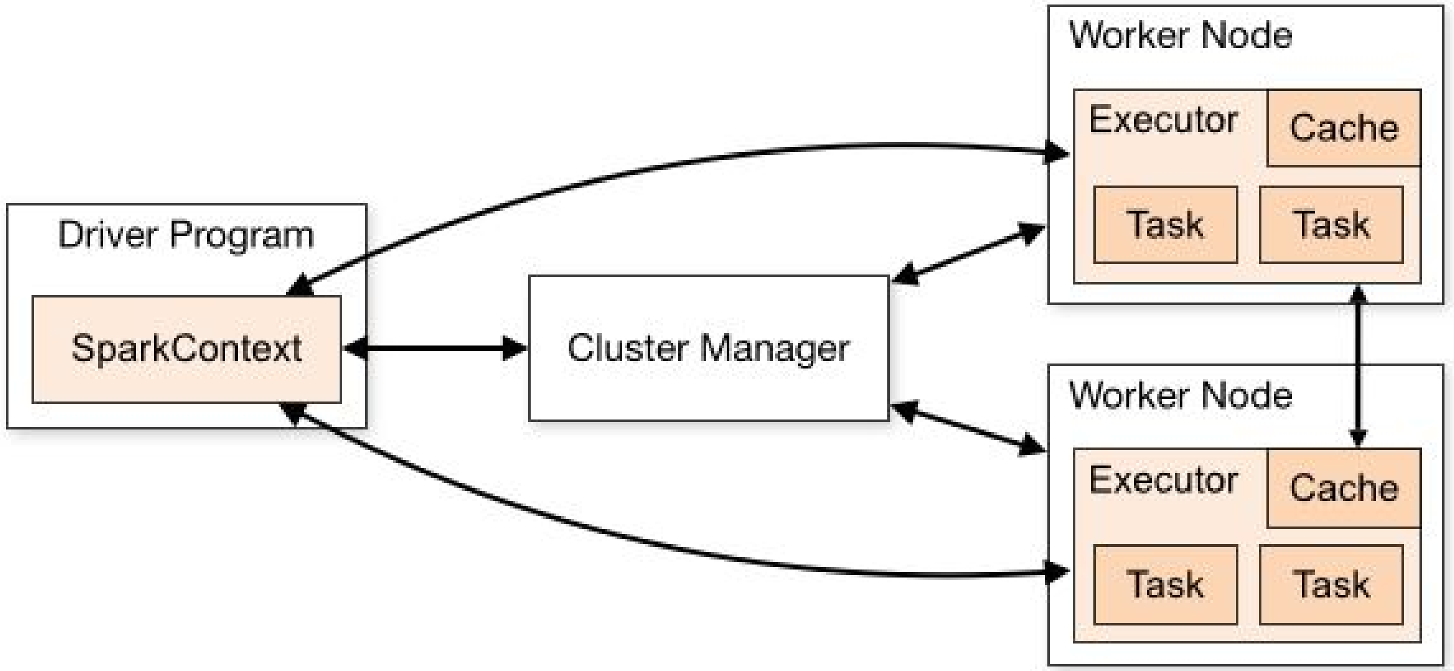
RDDs are Cachable

Immutable
Lazily Evaluated
Cachable

Mechanical Sympathy

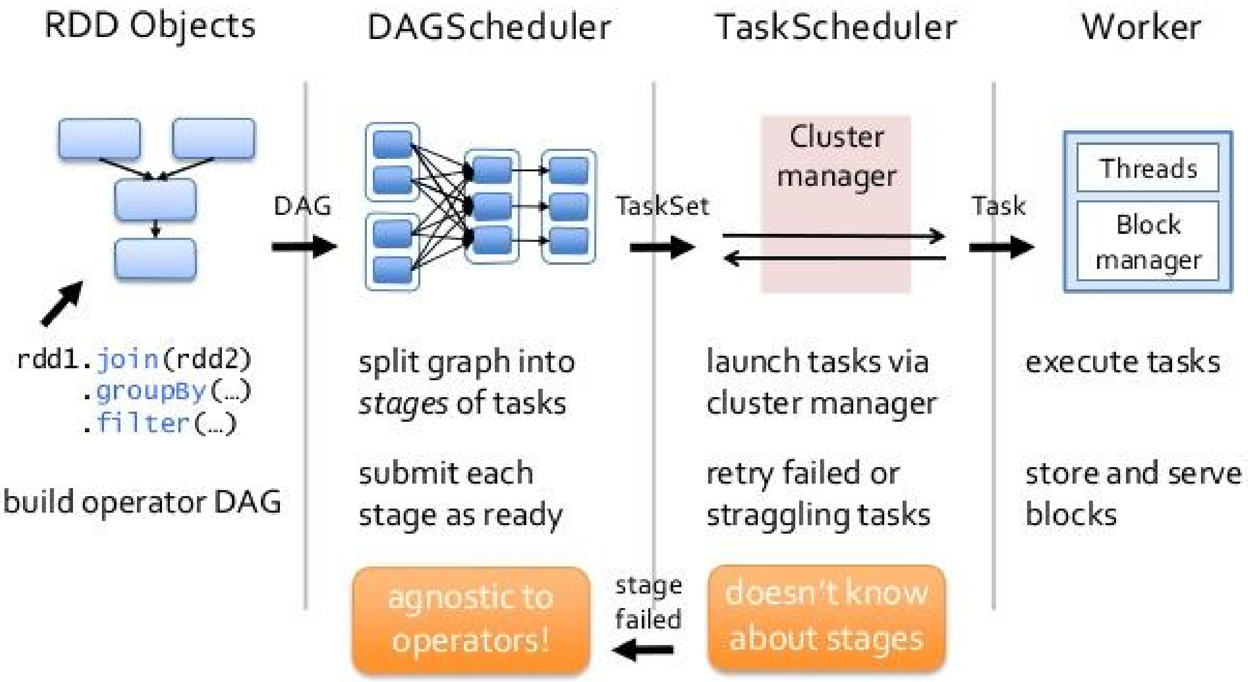
"You don't have to be an engineer to be be a racing driver, but you do have to have Mechanical Sympathy." ***– Jackie Stewart, racing driver***

Distributed Components



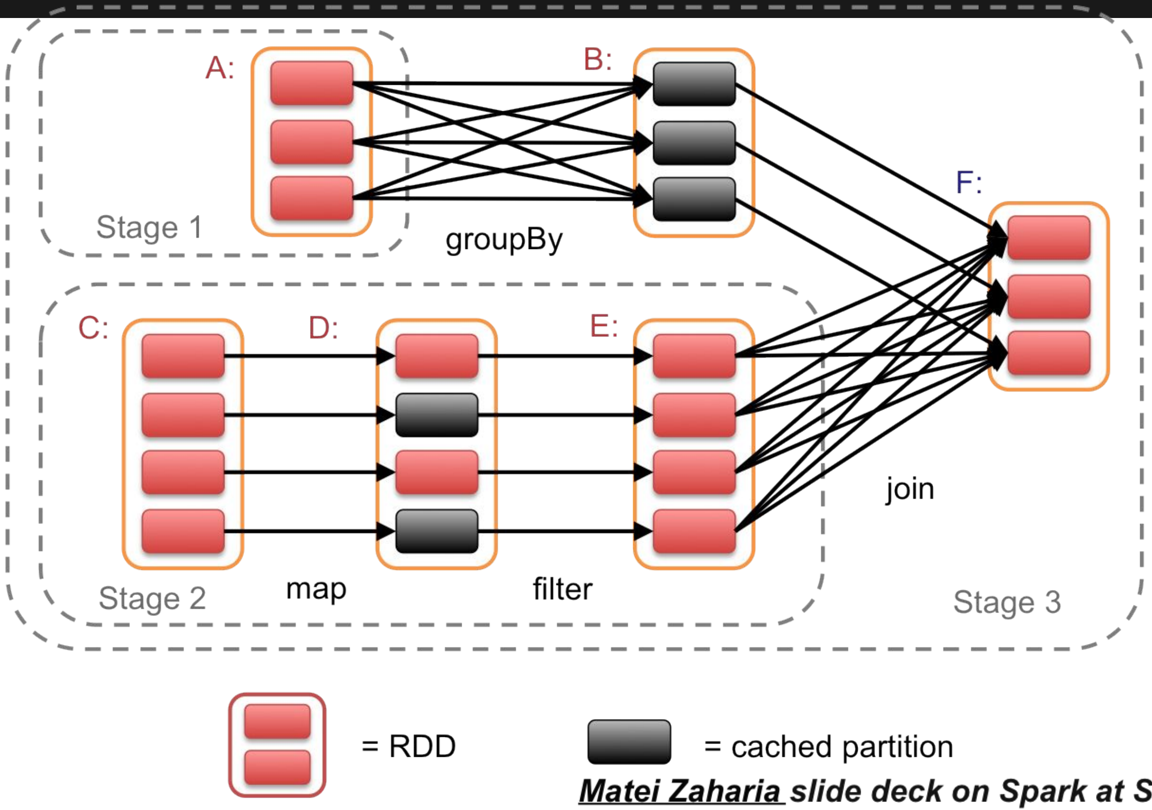
http://spark.apache.org/docs/latest/cluster-overview.html

Scheduler Process



https://cwiki.apache.org

DAG Example



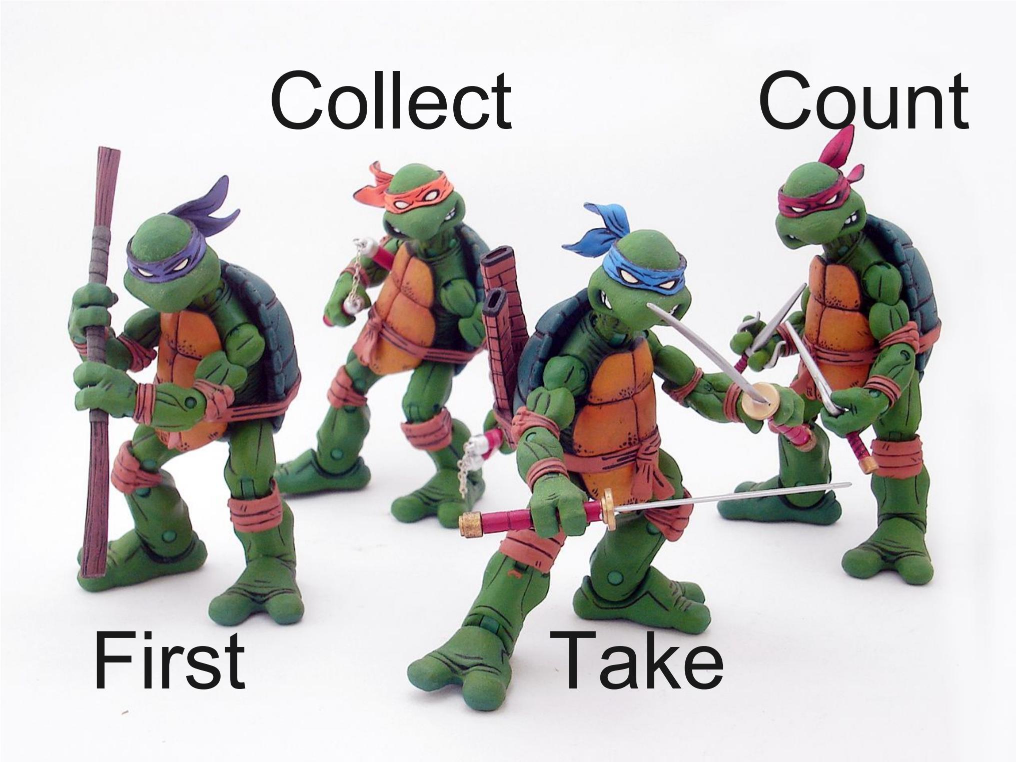
***Matei Zaharia** slide deck on Spark at Strata conference Feb 2013*

Two types of RDD Operations:

Transformations

Actions

Four Actions



Collect & Count

Collect - Return all the elements of the RDD as an array at the driver program.

Count - Return the number of elements in the RDD

http://spark.apache.org/docs/latest/programming-guide.html#resilient-distributed-datasets-rdds

First & Take

First - Return the first element in the RDD

Take - Return an array with the first *n* elements of the RDD

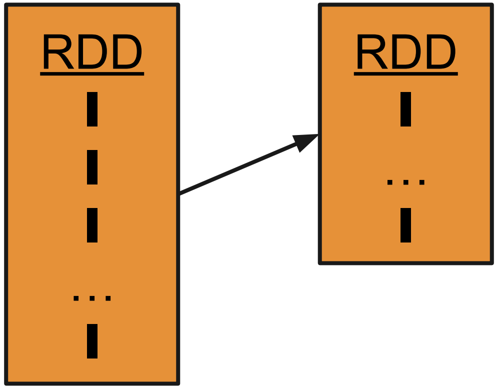
3 Transformations

- Filter
- Map
- FlatMap

RDD.filter

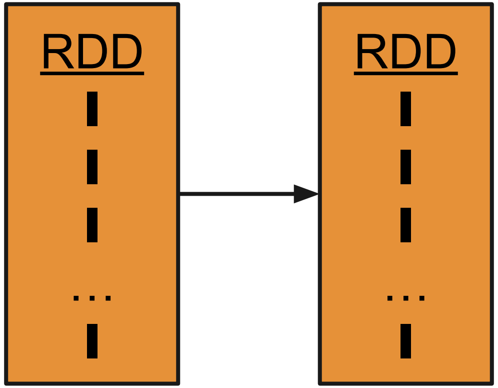
Applies a function to each element

Returns elements that evaluate to true



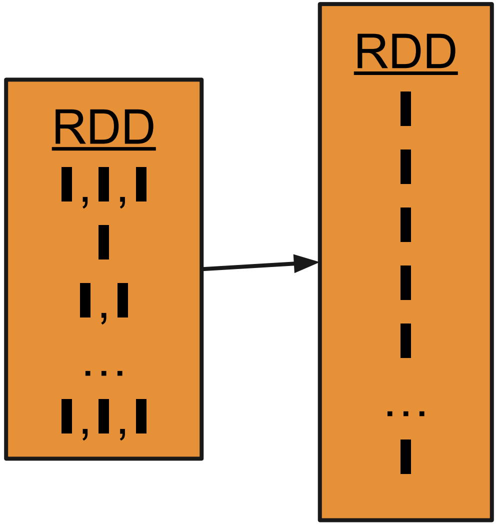
RDD.map

- Transforms each element
- Preserves # of elements



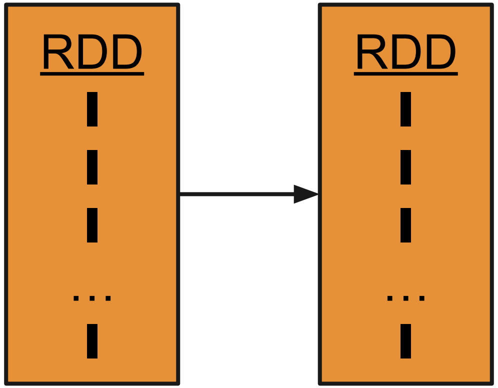
RDD.flatMap

- Transforms each element into 0-N elements
- Changes # of elements



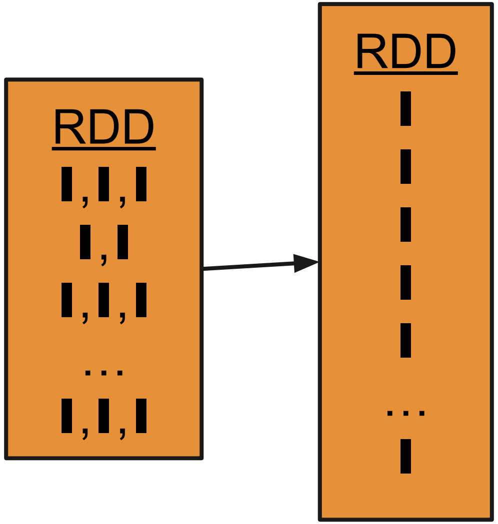
Map vs FlatMap - Examples

Alphabet letters (A,B,C) to
NATO Phonetic Letters (Alfa, Bravo, Charlie)



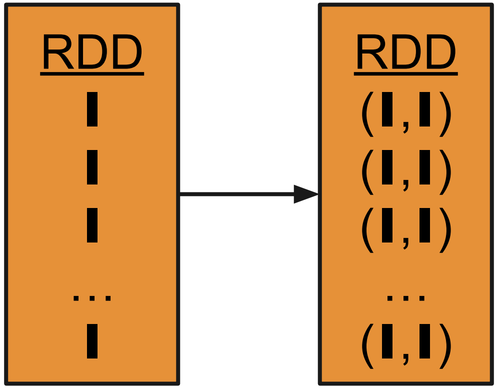
Map vs FlatMap - Examples

Paragraph of words ("Mary had a little lambda") to individual words



Map vs FlatMap - Examples

Numbers x, transform to the tuple (x, x^2)



Pair RDDs

Operations on tuples (key, value)

Offers better partitioning

Exposes new functionality

Reduce & ReduceByKey

- **Reduce** (Action)

Aggregate RDD elements using a function

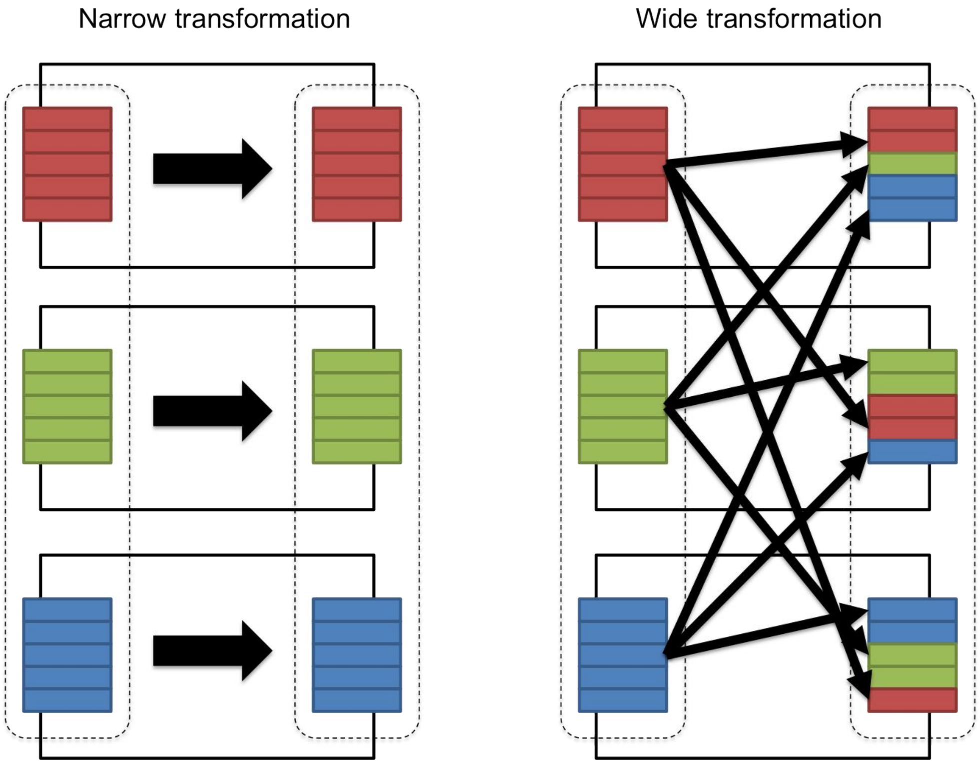
Returns single element

- **ReduceByKey** (Transformation)

Aggregate Pair RDD elements using a function

Returns Pair RDD

Transformation Complexity



http://horicky.blogspot.com/2013/12/spark-low-latency-massively-parallel.html

Code Examples (python)

Map, Collect:

```
data = sc.parallelize([1, 2, 3])
mapped_data = data.map(lambda x: x**2)
x = mapped_data.collect()
```

What is x's value and type?

Code Examples (python)

FlatMap, Take:

```
data = sc.parallelize([1, 2, 3])
flat_data = data.flatMap(lambda x: range(0, x))
x = flat_data.take(4)
```

What is x’s value and type?

Code Examples (python)

Reduce, Count:

```
data = sc.parallelize([1, 2, 3])
flat_data = data.flatMap(lambda x: range(0, x))
c = flat_data.count()
r = flat_data.reduce(lambda a, b: a + b)
```

What is c’s value and type?
What is r’s value and type?

Pop Quiz

What is the ‘sc’ in sc.parallelize()?

SparkContext.

- Given to you when you launch Spark shell
- Your way to get data into/out of RDDs

Transformation vs. Action?

```
data = sc.parallelize( \
    ["Aaron Merlob", "Ryan Henning", "Aaron Ryan", ""])
words = data.flatMap(lambda s: s.split(" "))
result = words.map(lambda w: (w, 1)) \
            .reduceByKey(lambda a, b: a + b)
result.filter(lambda p: 'a' in p[0]).count()
result.filter(lambda p: p[1] > 1).count()
```

We do redundant work right here!

Transformation vs. Action?

```
data = sc.parallelize( \
    ["Aaron Merlob", "Ryan Henning", "Aaron Ryan", ""])
words = data.flatMap(lambda s: s.split(" "))
result = words.map(lambda w: (w, 1))  \
              .reduceByKey(lambda a, b: a + b).cache()
result.filter(lambda p: 'a' in p[0]).count()
result.filter(lambda p: p[1] > 1).count()
```

This prevents the redundant work.

What did we learn?

- Spark coordinates multiple computers.
- RDDs are immutable and lazily evaluated.
- Transformations build a plan of attack (DAG).
- Actions force an evaluation (produce answers).
- Developers designate what they want to cache!
- The Spark shell gives you a SparkContext (‘sc’).

Live Demo

With `pyspark` shell. It creates the SparkContext (sc) for you!

With iPython notebook. You'll have to create the SparkContext yourself.
(Unless you do fancy things with iPython notebook…)

tmux

Live demo...

APPENDIX

Type Inferred - Comparison

Python shell:

```
numbers = sc.parallelize([1,2,3])
letters = sc.parallelize([“a”,“b”,“c”])
numbers.map(lambda x: x * x).collect()
letters.map(lambda x: x * x).collect()
```

Fails at runtime

Type Inferred - Comparison

Scala shell:

```
val numbers = sc.parallelize(Seq(1,2,3))
val letters = sc.parallelize(Seq(“a”,“b”,“c”))
numbers.map(x => x * x).collect()
letters.map(x => x * x).collect()
```

Fails at compile-time