

Introduction to Big Data, Map Reduce, Spark, and Spark RDDs

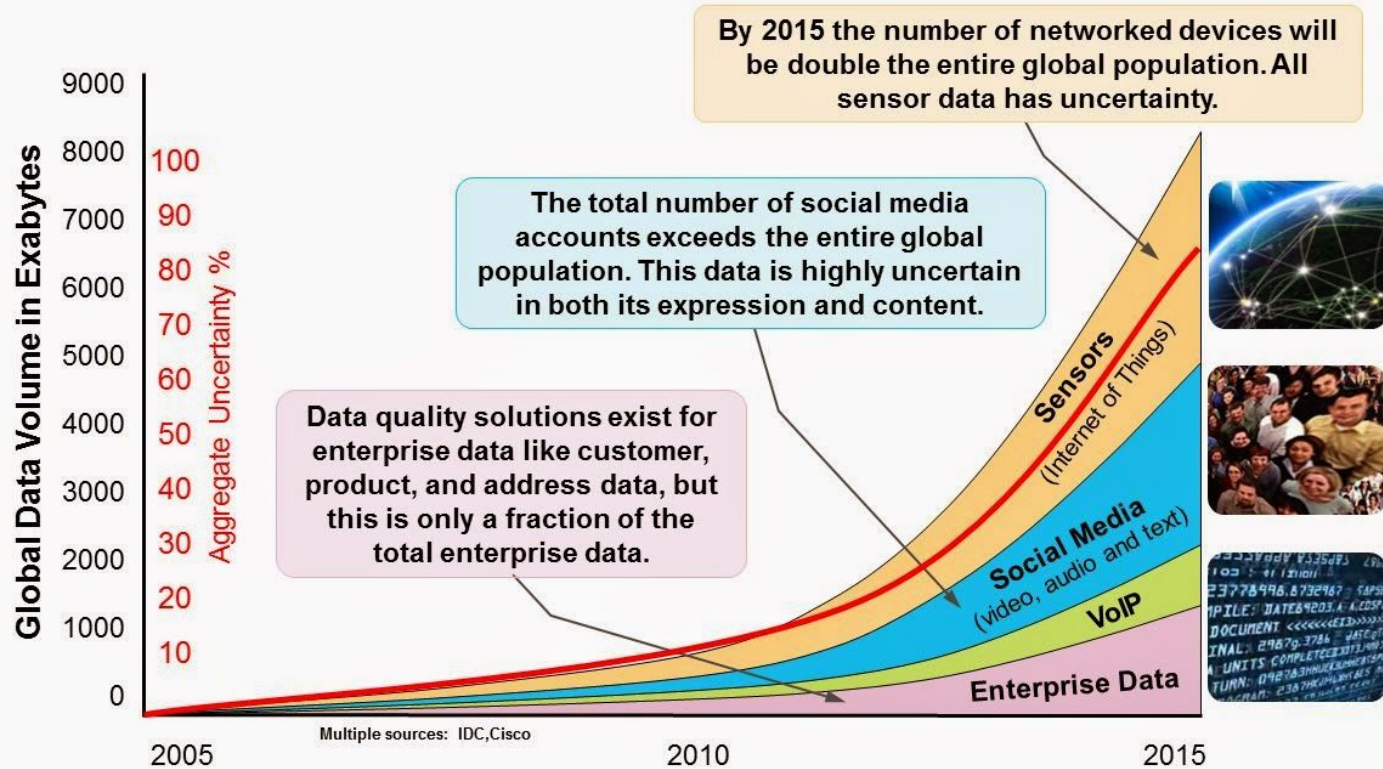
Taryn Heilman
Ryan Henning
Frank Burkholder

galvanize



- Define Big Data
- Define distributed computing
- Name two major distributed computing paradigms, and the fundamental difference between them
- ELI5: Map Reduce
- Describe how Spark does distributed computing
- Define what a Spark RDD is, by its properties and operations
- Explain the difference between transformations and actions on an RDD
- Explain RDD persisting/caching and situations where this is useful

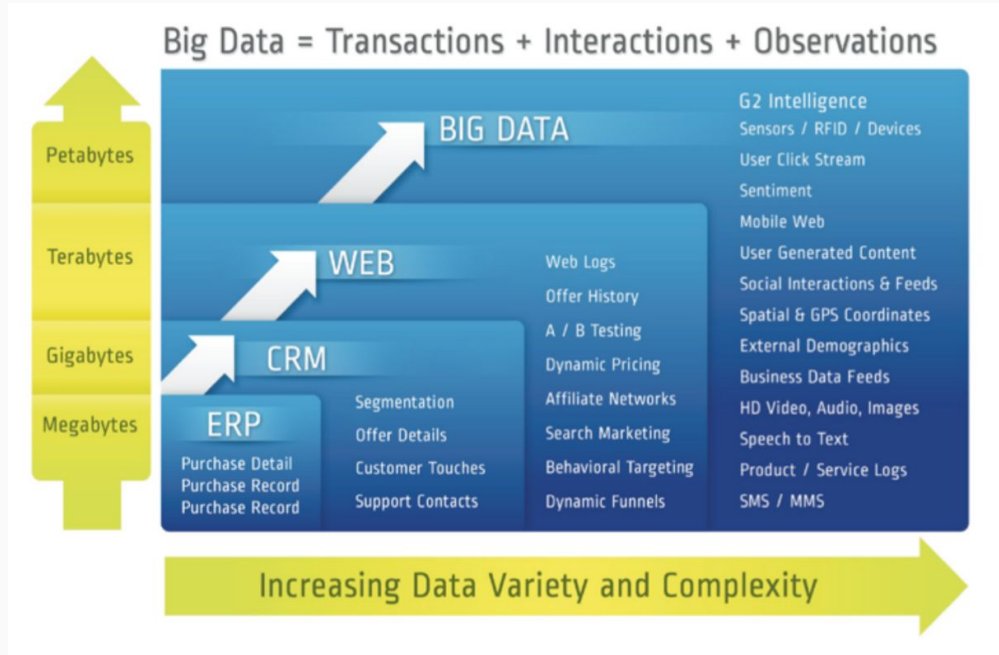
Growth of data



Source: IBM Global Data Growth

Big Data

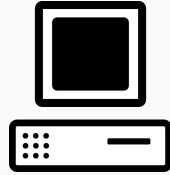
- Data so large that it cannot be stored on one machine.
- Can be
 - Structured: highly organized, searchable, fits into relational tables
 - Unstructured: no predefined format, multiple formats
- Often described as 3 Vs: (high volume, velocity, and variety)
- Two possible solutions to Big Data:
 - Make bigger computers (scale up)
 - Distribute data and computation onto multiple computers (scale out)



ERP: Enterprise resource planning
CRM: Customer relationship management

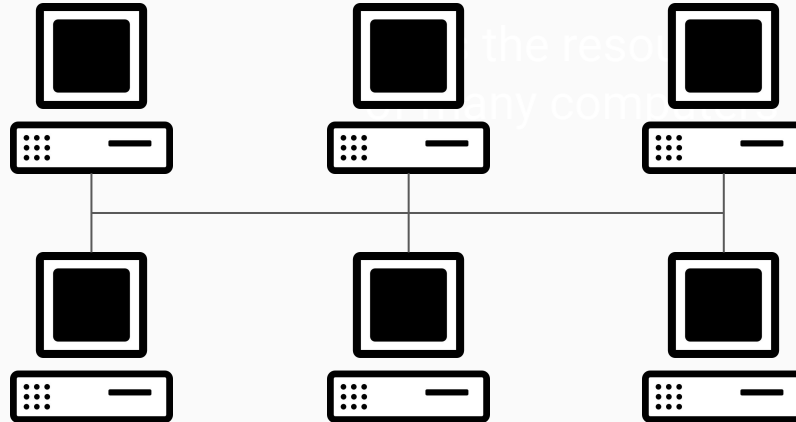
Local

Uses the resources
of 1 computer



Distributed

Uses the resources
of many,
interconnected
computers



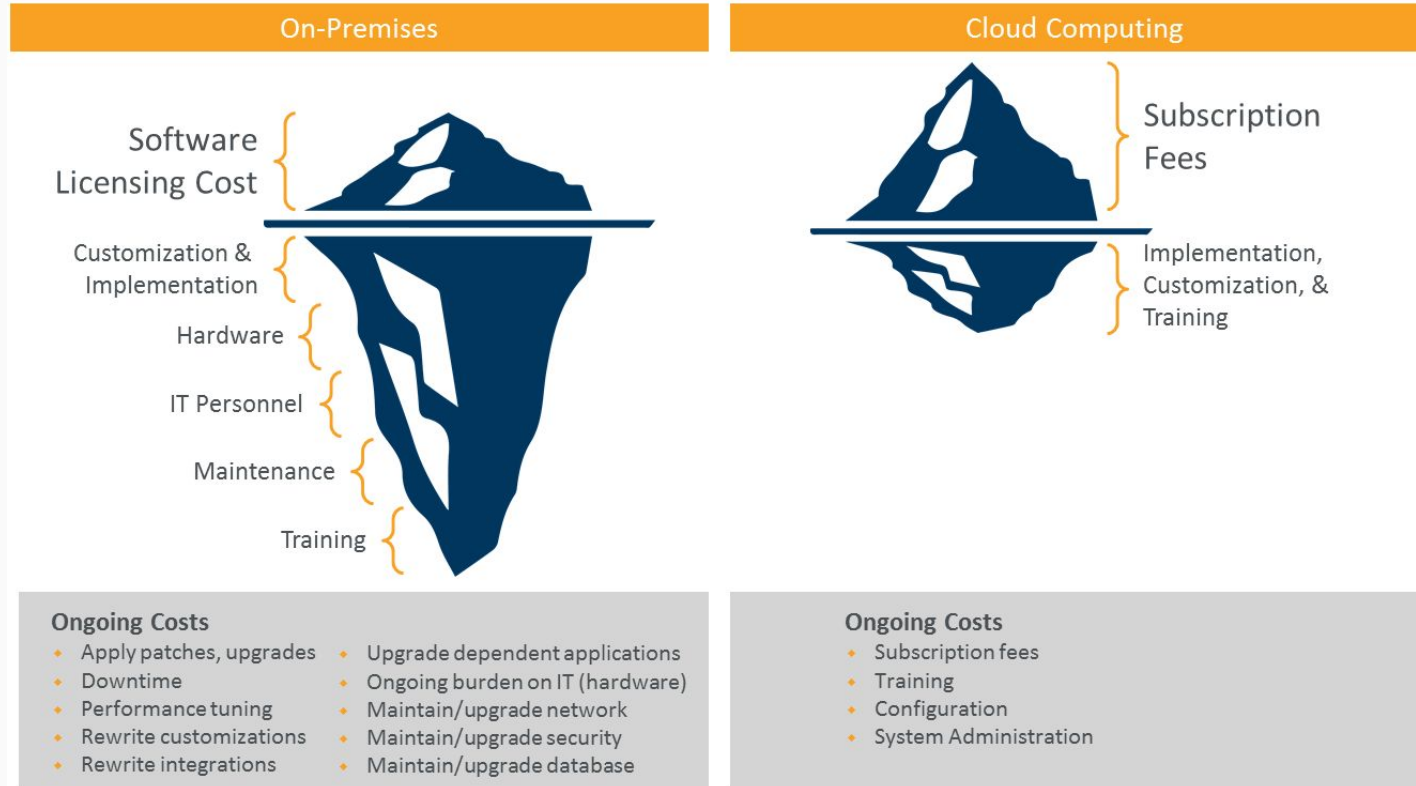
On-premise or the cloud?

On-premise:

Software and/or hardware that are installed on the premises of the company that uses them.

Cloud:

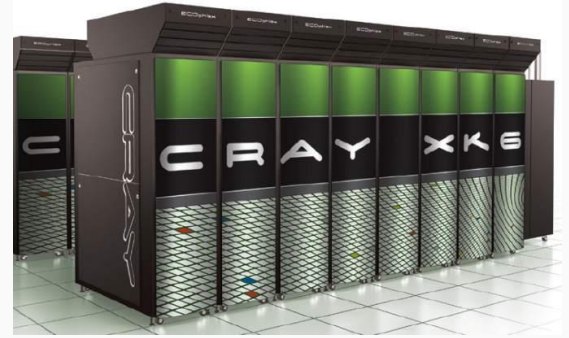
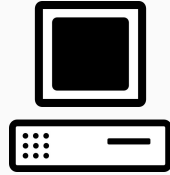
Software and/or hardware installed at a remote facility and provided to companies for a fee.



Storage and computation paradigms

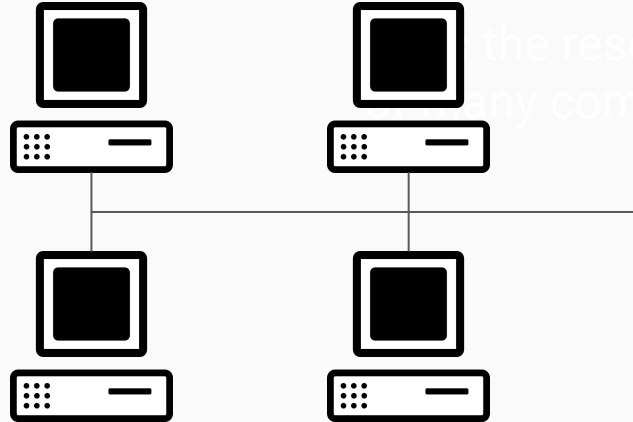
Local

Uses the resources
of 1 computer



Distributed

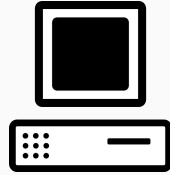
Uses the resources
of many,
interconnected
computers



Storage and computation paradigms

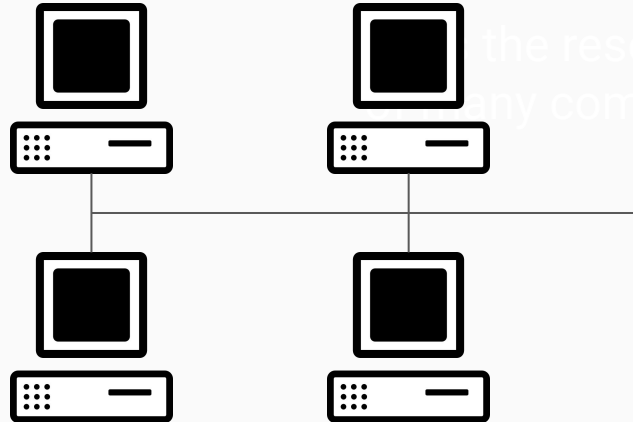
Local

Uses the resources
of 1 computer



Distributed

Uses the resources
of many,
interconnected
computers



When to use distributed computing

Size of data	Analysis tools	Data storage	Examples
< 10 GB	R/Python	Local: can fit in one machine's RAM	Thousands of sales figures
10 GB - 1 TB	R/Python with indexed files (key, record)	Local: fits on one machine's hard drive	Millions of web pages
> 1 TB	Hadoop, Spark, Distributed Databases	Distributed: Stored across multiple machines	Billions of web clicks, about 1 month of tweets





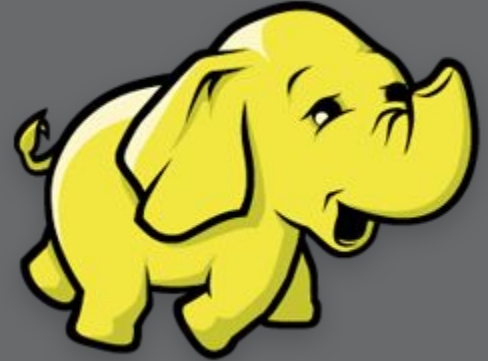
- **Hadoop** is an open-source, **distributed computing framework** that was created (2011) to make it easier to work with big data.
- It provides a method to access and **process data that are distributed among multiple clustered computers**.
- Hadoop manages **distributed data storage (HDFS)** and **distributed computation (Map Reduce)**
- **In computation (Map Reduce), writes to the hard drive.** The MapReduce computation engine has fallen out of favor (faster, better approaches like Spark) but HDFS continues as a common data storage solution.



- **Data science friendly** distributed computing. Released in 2014.
- Highly efficient distributed operations
- More use cases than Hadoop's MapReduce
- Relatively easy integration into existing paradigms (works with HDFS)
- Advanced machine learning library APIs for Scala, Java, Python, and R (we'll be using PySpark)
- **Computation and most results of computation, stay in RAM.**

- Define Big Data
- Define distributed computing
- Name two major distributed computing paradigms, and the fundamental difference between them
- ELI5: Map Reduce
- Describe how Spark does distributed computing
- Define what a Spark RDD is, by its properties and operations
- Explain the difference between transformations and actions on an RDD
- Compare/contrast Spark RDDs, Dataframes, and Datasets
- Explain RDD persisting/caching and situations where this is useful

Map Reduce

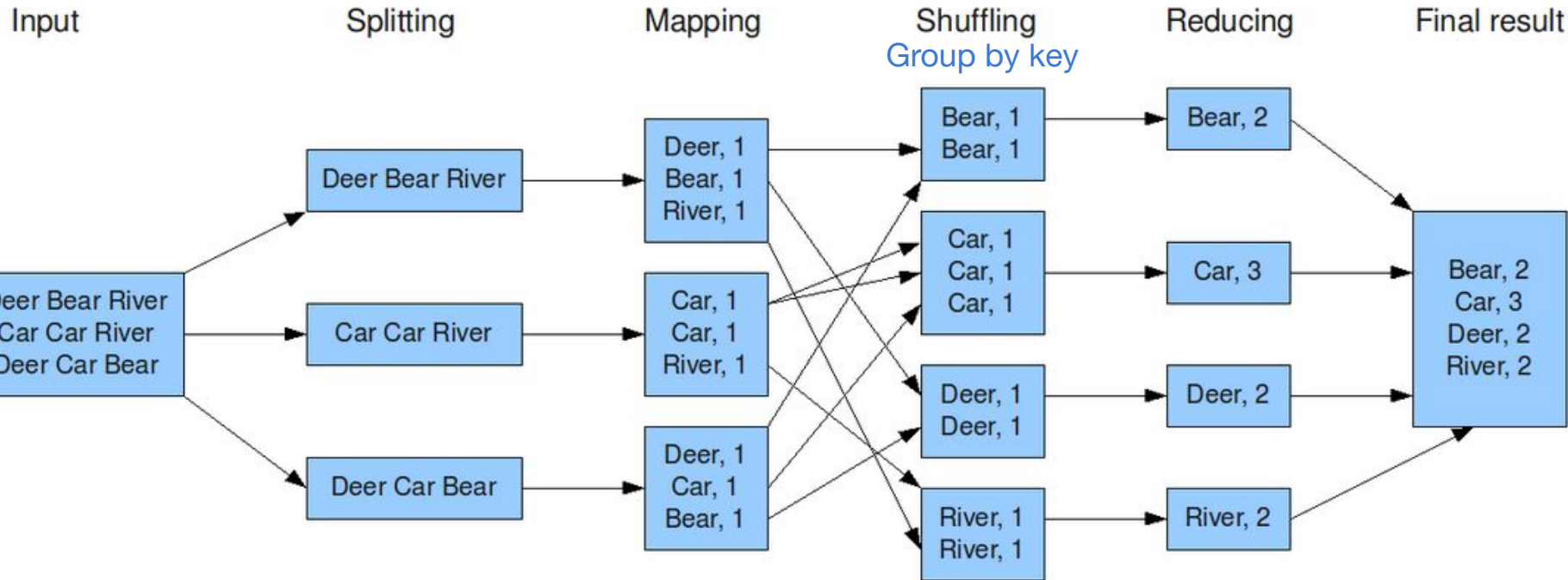


galvanize

- **Send the computation to the data rather than trying to bring the data to the computation.**
- Computation and communication are handled as (key, value) pairs.
- In the “map” step, the **mapper** maps a function on the data that transforms it into (key, value) pairs. A **local combiner** may be run after the map to aggregate the results in (key, local aggregated values).
- After the mapping phase is complete, the (key, value) or (key, local aggregated value) results need to be brought together, sorted by key. This is called the “shuffle and sort” step.
- Results with the same key are all sent to the same MapReduce TaskTracker for aggregation in the **reducer**. This is the “reduce” step.
- The final reduced results are communicated to the Client.

MapReduce Illustration: Word Count

The overall MapReduce word count process



MapReduce Exercise: Counting Cards by Suit



Input

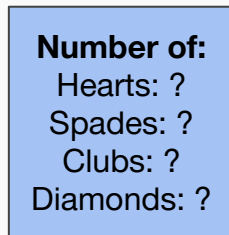
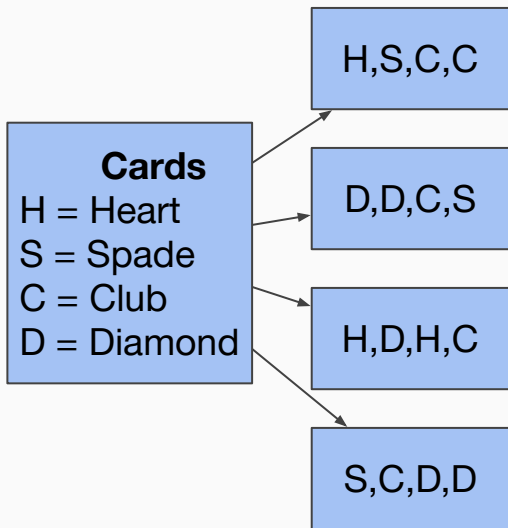
Splitting

Mapping

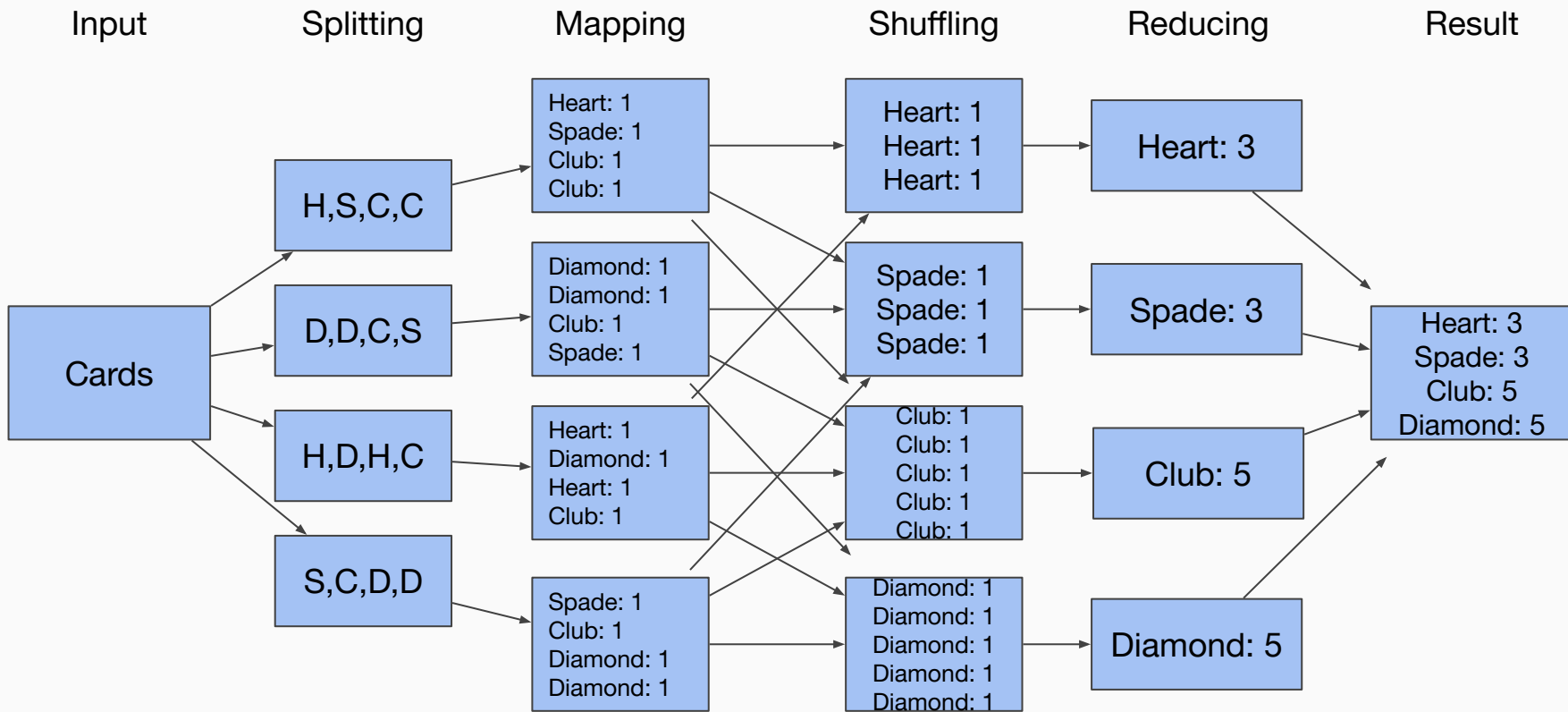
Shuffling

Reducing

Result

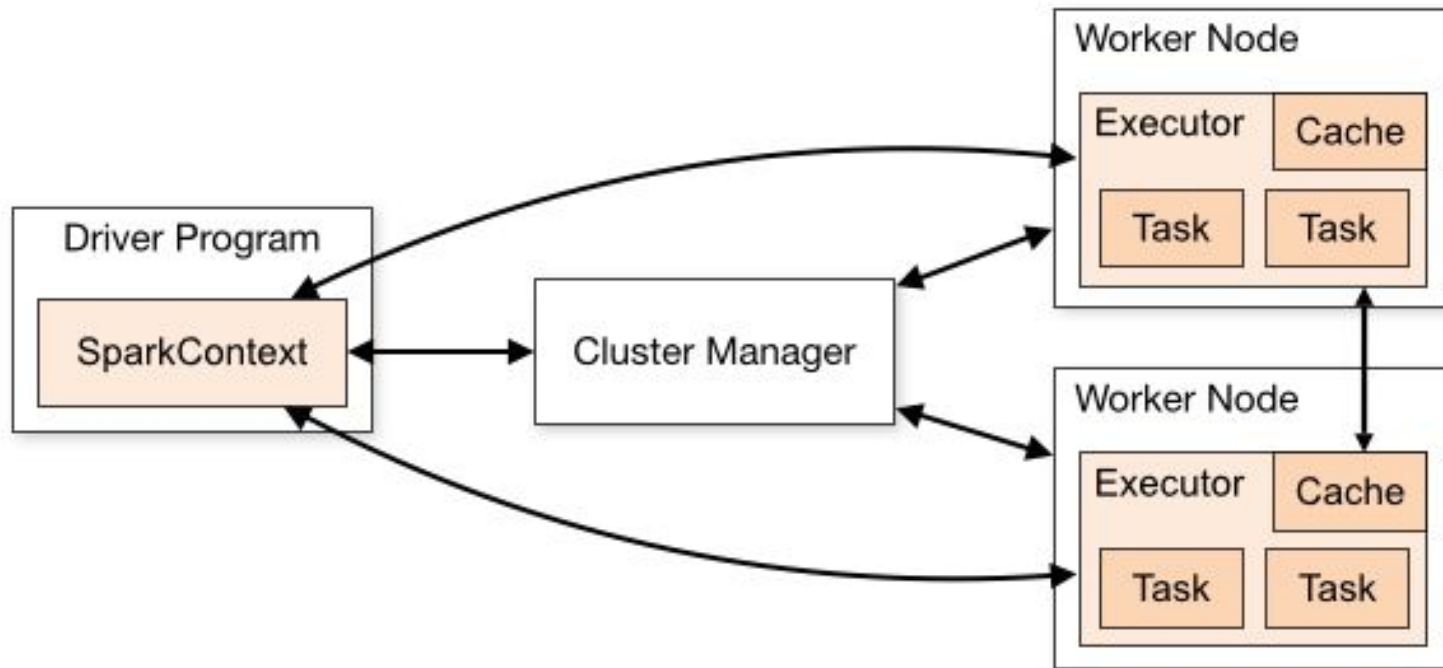


MapReduce Exercise: Counting Cards by Suit



- Hadoop - open-source framework made to handle big data through distributed computing.
- HDFS - data management component of Hadoop
- MapReduce - computation component of Hadoop, but also a general computation paradigm:
 - A local mapper maps a function on data, perhaps using local combiner, then sends the results somewhere to be reduced
 - Data is handled as (key, value) pairs
 - All computations written to hard disk (for redundancy, but slow)
- Many other [components in Hadoop Ecosystem](#)

- Define Big Data
- Define distributed computing
- Name two major distributed computing paradigms, and the fundamental difference between them
- **ELI5: Map Reduce**
- Describe how Spark does distributed computing
- Define what a Spark RDD is, by its properties and operations
- Explain the difference between transformations and actions on an RDD
- Explain RDD persisting/caching and situations where this is useful

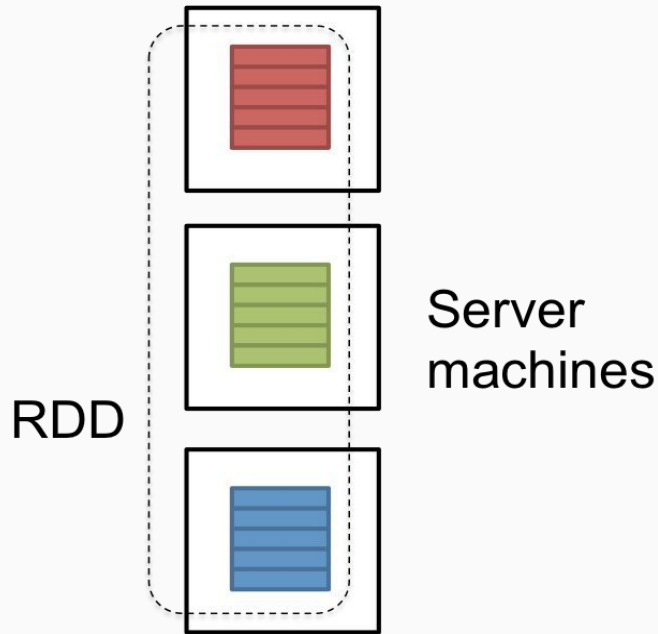


Spark RDDs

galvanize

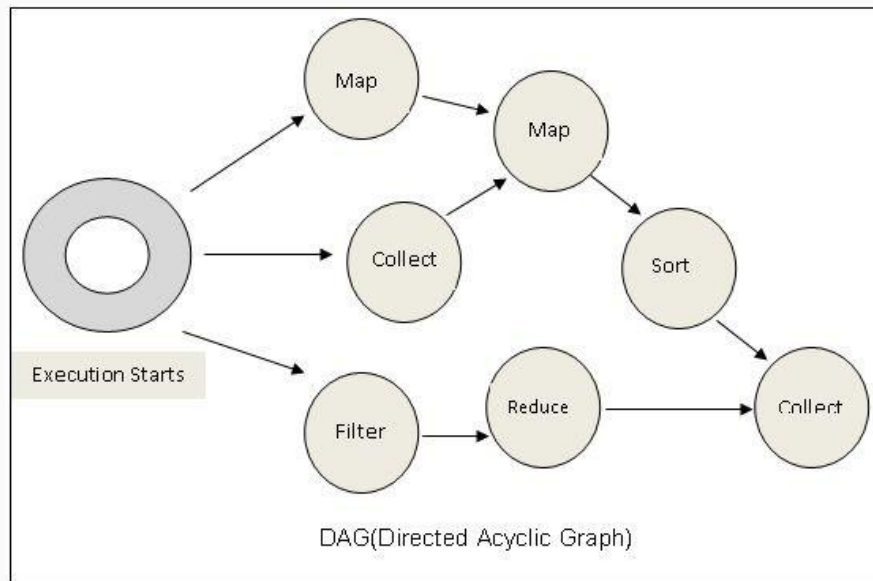
Resilient Distributed Datasets

- fundamental data structure in Spark
- created from HDFS, S3, HBase, JSON, text, local
- distributed across the cluster as partitions (atomic chunks of data)
- each partition is in memory
- can recover from errors (DAG - more in a bit)
- traceability of each partition, can re-run the processing (DAG again)
- **immutable** : you *cannot* modify an RDD in place
- **Lazily Evaluated**
- Cachable



A “Functional” Programming paradigm

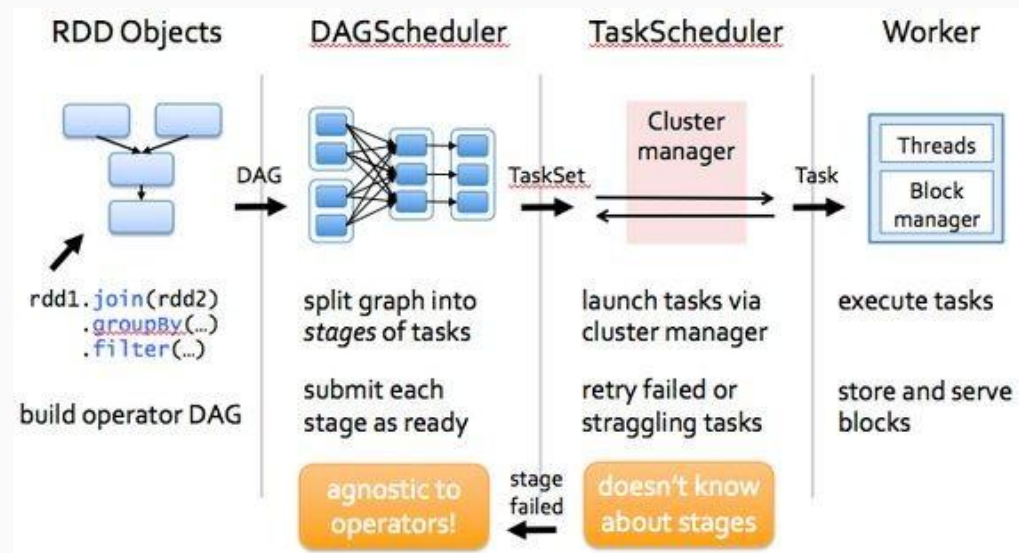
- RDDs are immutable ! You can **only transform** an existing RDD into another one.
- Spark provides many **transformation functions**.
- Programming = construct a **Directed Acyclic Graph (DAG)**.
- **DAG passed from the client to the manager**, who then distributes them to workers, who apply them across their partitions of the RDD.



[\[Image Source\]](#)

Directed-Acyclic-Graph

- You construct your sequence of transformations in Python.
- Spark functional programming interface builds up a DAG. (But the DAG isn't executed yet - **lazy evaluation**).
- This DAG is sent by the driver for execution to the cluster manager when an action is called (now it's evaluated)



Interacting with RDDs: Transformations and Actions

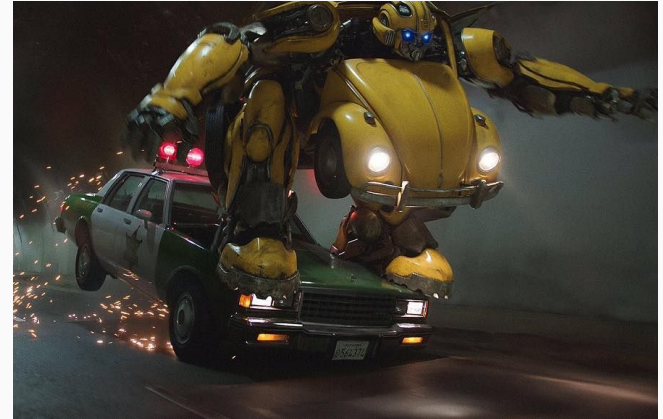


Two types of RDD Operations:

Transformations



Actions



Examples of RDD transformations (go into DAG)



Method	Type	Category	Description
<code>.map(func)</code>	transformation	mapping	Return a new RDD by applying a function to each element of this RDD.
<code>.flatMap(func)</code>	transformation	mapping	Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.
<code>.filter(func)</code>	transformation	reduction	Return a new RDD containing only the elements that satisfy a predicate.
<code>.sample()</code>	transformation	reduction	Return a sampled subset of this RDD.
<code>.distinct()</code>	transformation	reduction	Return a new RDD containing the distinct elements in this RDD.
<code>.keys()</code>	transformation	<k, v>	Return an RDD with the keys of each tuple.
<code>.values()</code>	transformation	<k, v>	Return an RDD with the values of each tuple.
<code>.join(rddB)</code>	transformation	<k, v>	Return an RDD containing all pairs of elements with matching keys in self and other. Each pair of elements will be returned as a (k, (v1, v2)) tuple, where (k, v1) is in self and (k, v2) is in other.
<code>.reduceByKey()</code>	transformation	<k, v>	Merge the values for each key using an associative and commutative reduce function.
<code>.groupByKey()</code>	transformation	<k, v>	Merge the values for each key using non-associative operation, like mean.
<code>.sortBy(keyfunc)</code>	transformation	sorting	Sorts this RDD by the given keyfunc.
<code>.sortByKey()</code>	transformation	sorting/<k, v>	Sorts this RDD, which is assumed to consist of (key, value) pairs.

When you evaluate the DAG, you should get results!

.collect() - Return all the elements of the RDD as an array at the driver program.

.count() - Return the number of elements in the RDD

.reduce() - Reduces RDD using given function

.take() - Return an array with the first n elements of the RDD

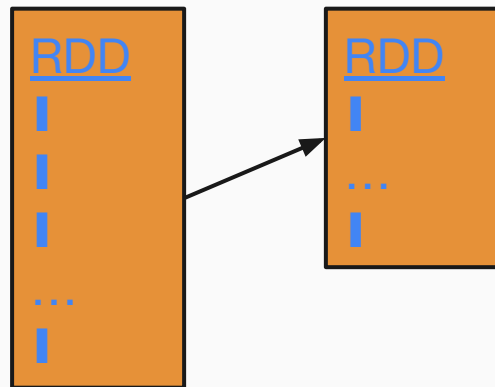
.first() - Return the first element in the RDD

.saveAsTextFile() - save RDD as text file

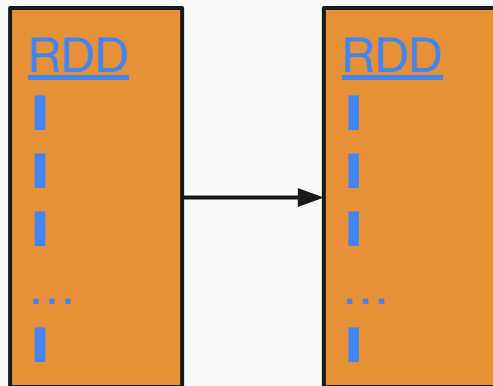
Applies a function to each element

Returns elements that evaluate to true

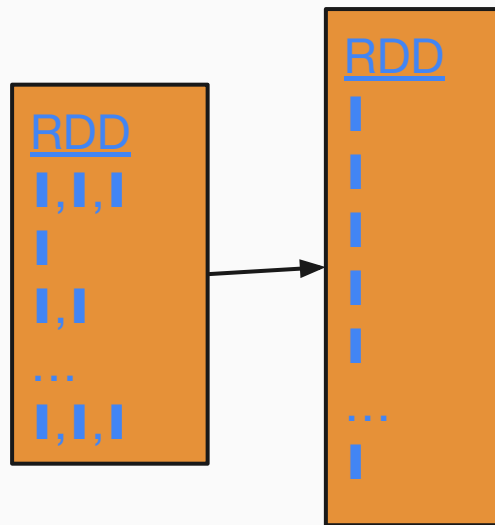
Example: keep only even numbers



- Transforms each element
- Preserves # of elements
- Example: Alphabet letters (A,B,C) to NATO Phonetic Letters (Alfa, Bravo, Charlie)




- Transforms each element into 0-N elements
- Changes # of elements
- Example: Paragraph of words (“Mary had a little lambda”) to individual words



Map, Collect:

sc = Spark Context, API in PySpark to use Spark



```
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?

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?

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?

Persisting/Caching



Explicitly keep an RDD in memory

Used if you have an RDD that is or will be used for different operations many times

```
rdd.persist()
```

```
# OR
```

```
rdd.cache()
```

`.cache()` uses the default storage level `MEMORY_ONLY`, while `.persist()` gives you the option to specify the storage level

This is useful when you will be doing a lot of repetitive calculations on an RDD/df...like machine learning!

Explicitly keeps an RDD in memory

Caching will only take place **when an action is performed**

```
from pyspark.storagelevel import StorageLevel  
  
rdd.persist(StorageLevel.MEMORY_ONLY)
```

Can also do `rdd.unpersist()` to free memory

Storage Level	Meaning
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER	Store RDD as <i>serialized</i> Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer , but more CPU-intensive to read.
MEMORY_AND_DISK_SER	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.

- 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').

- Define Big Data
- Define distributed computing
- Name two major distributed computing paradigms, and the fundamental difference between them
- ELI5: Map Reduce
- Describe how Spark does distributed computing
- Define what a Spark RDD is, by its properties and operations
- Explain the difference between transformations and actions on an RDD
- Explain RDD persisting/caching and situations where this is useful

Before doing the assignment:

`spark_rdd_selfdirected_walkthrough.ipynb`

You'll need to start your docker container and then navigate to the file (it's in this lecture folder).

Operate on tuples (key, value)

Offers better partitioning

Exposes new functionality:

- `reduceByKey`: Aggregates pair RDD elements using a function, returns pair RDD
- `groupByKey`: group elements by key, need to aggregate afterwards with `reduce` or `map`, returns pair RDD
- `sortByKey`: sorts the RDD by key, returns pair RDD