Introduction to Big Data Analytics and Engineering

CAMBRIDGE SPARK

@cambridgespark

Schedule

- What is “Big Data”
- The Hadoop ecosystem
- Working with Spark RDDs
- Competition

Download

Slides: http://goo.gl/oukHdC

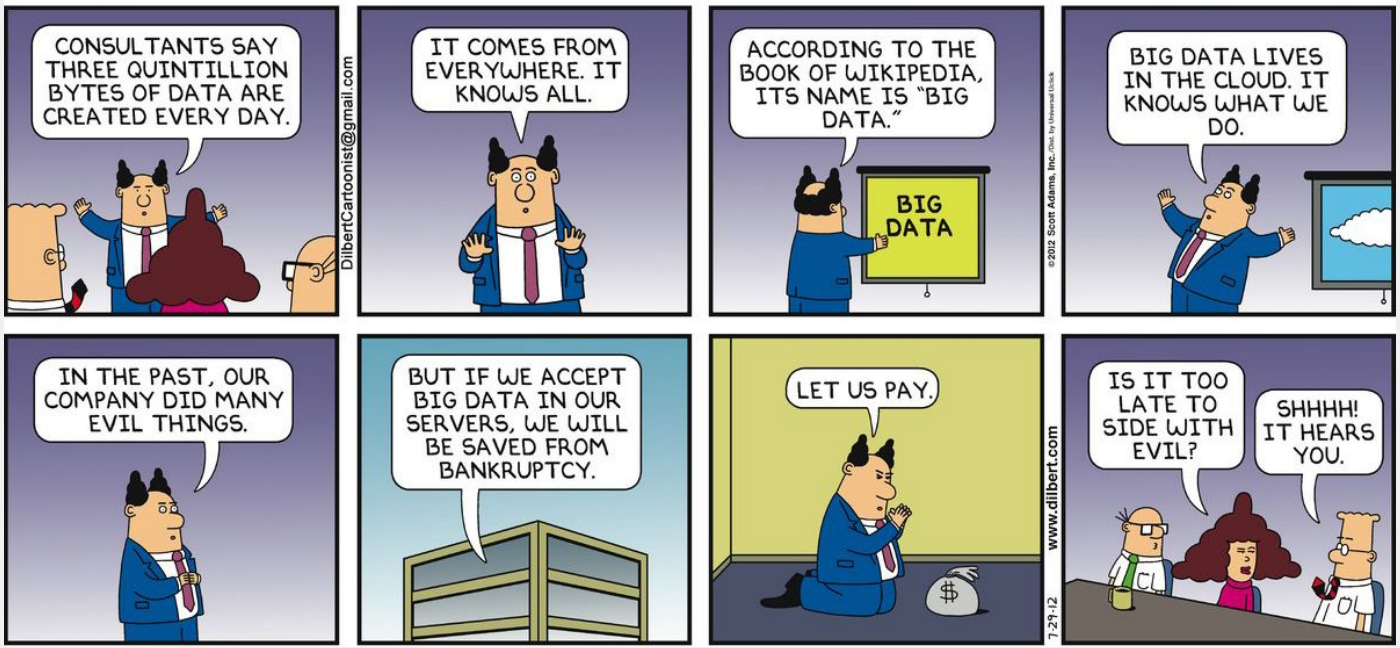
Files: http://gitlab.cambridgespark.com/pub/bigdata-spark

Setup Instructions (see SETUP.MD)

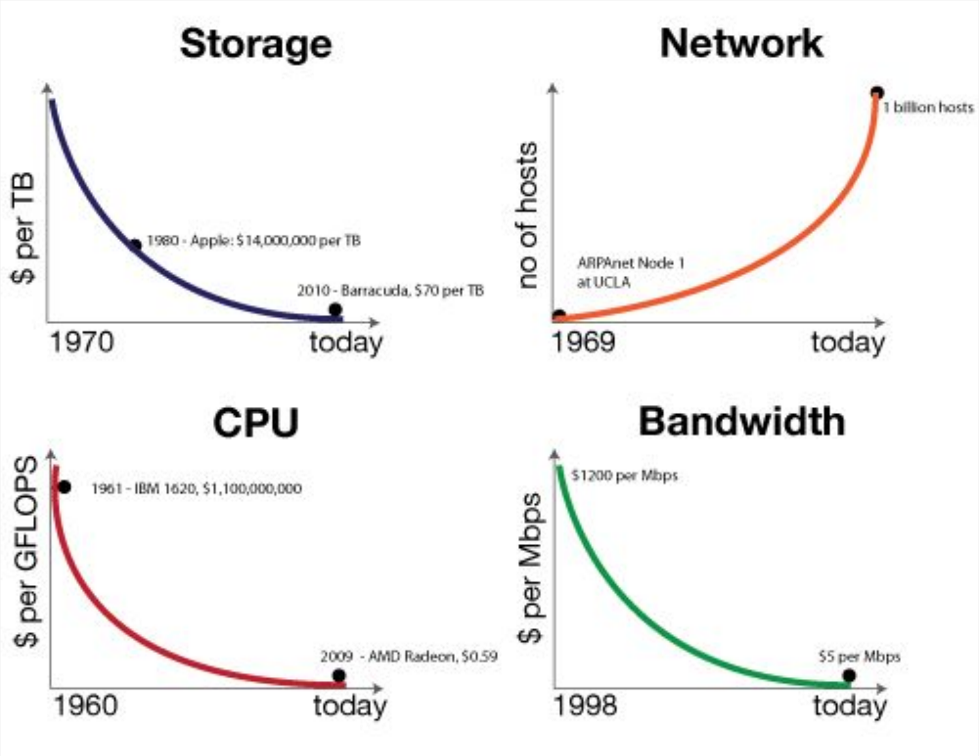
- Python 3.6 / Anaconda: https://www.anaconda.com/download
- JDK 8: https://java.com/en/download/help/index_installing.xml
- `pip install pyspark`
- `jupyter notebook Spark.ipynb`

What is “Big Data”

Big Data knows what we do

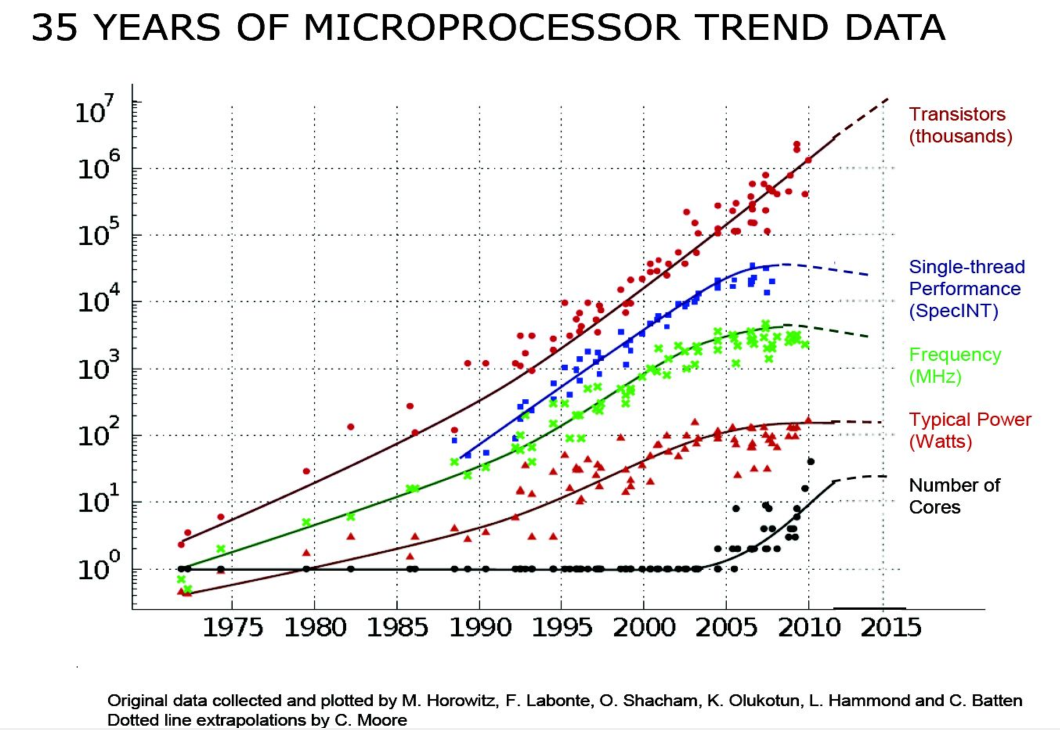


Context



http://radar.oreilly.com/2011/08/building-data-startups.html

Context



https://www.karlrupp.net/wp-content/uploads/2015/06/35years.png

Context

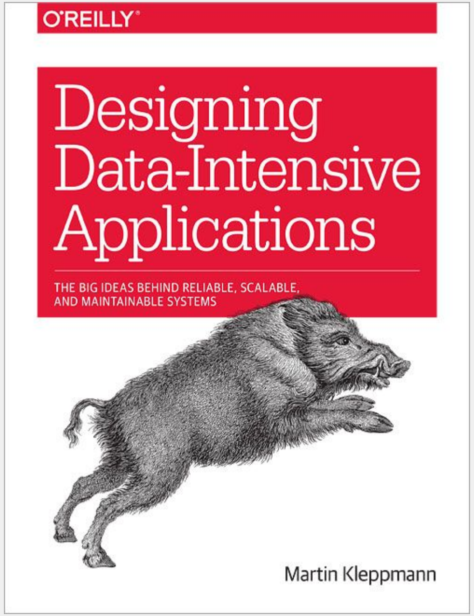
1. More Data
2. Cheaper hardware
3. Going parallel and distributed

Defining Big Data

1. Volume
2. Velocity
3. Variety

Defining Big Data

Many of the technologies described in this book fall within the realm of the *Big Data* buzzword. However, the term "Big Data" is so overused and underdefined that it is not useful in a serious engineering discussion. This book uses less ambiguous terms, such as single-node versus distributed systems, or online/interactive versus offline/batch processing systems.



NoSQL… Big Data… Scalability… CAP Theorem… Eventual Consistency… Sharding…

Nice buzzwords, but how does the stuff actually work?

LA >>> What volume do you deal with?

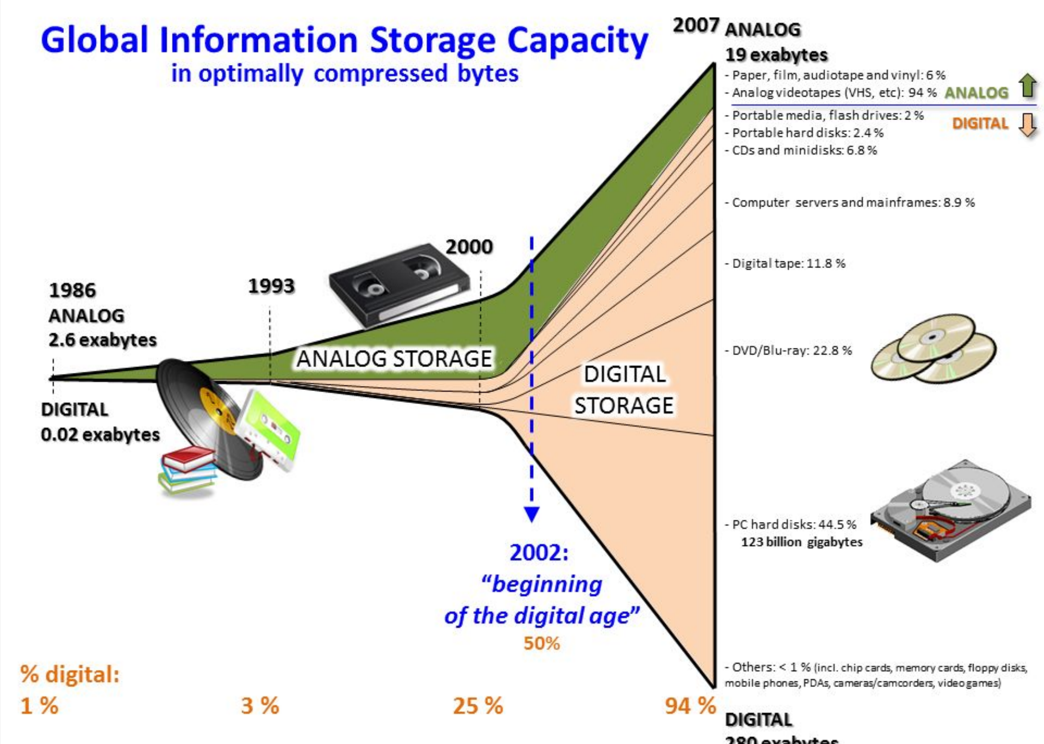
What sort of volume are you dealing with at your workspace?

- In terms of disk space?
- In terms of number of files?
- In terms of number of transactions?

Volume example

- Facebook average daily contribution of one user takes 1MB of space
- Upwards of 1,000 million users.
- This translates into an average of **1,000TB** or **1PB** per day.

Volume trend



Source: Hilbert, M., & López, P. (2011). The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(6025), 60 –65. http://www.martinhilbert.net/WorldInfoCapacity.html

Exabyte

Digital content production in 1999

1,5 exabyte

1,500 petabyte

1,500,00 terabyte

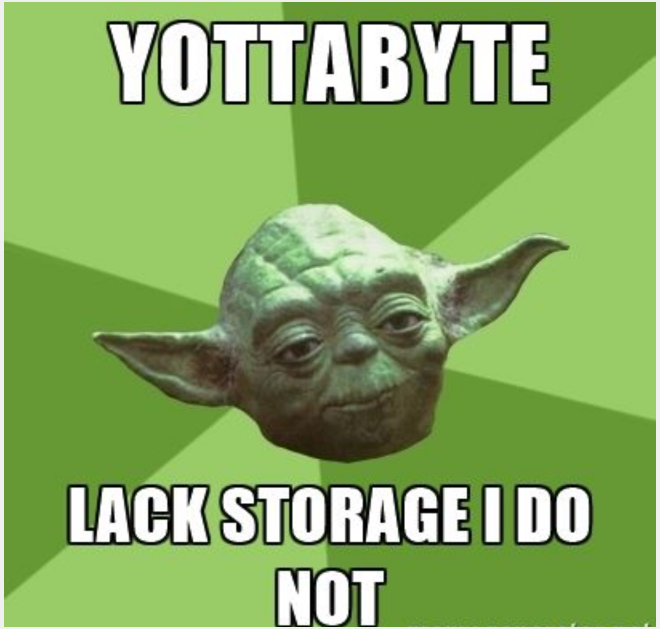
Equivalent to **750,000,000,000** HD images

Zetabyte

Cisco reported global IP traffic in 2016 exceeded **1 Zetabyte** (1000EB)

This is roughly as many bits as there are stars in the universe!

Yottabyte = 1,000ZB



- $30 to buy a standard terabyte hard-drive.
- $ 30 trillion to store one yottabyte.

Volume challenges

- **Single machine not enough**
- How to process data across multiple machines?
- How to provide fault tolerance?

Velocity

- CERN tracks ~600 million particle collisions per second resulting in 10 GB of data transferred from its servers every second.
- Twitter recently indicated that they handle **3,000 images every second**.
- WhatsApp is reported to process 60 billion messages a day.

Variety

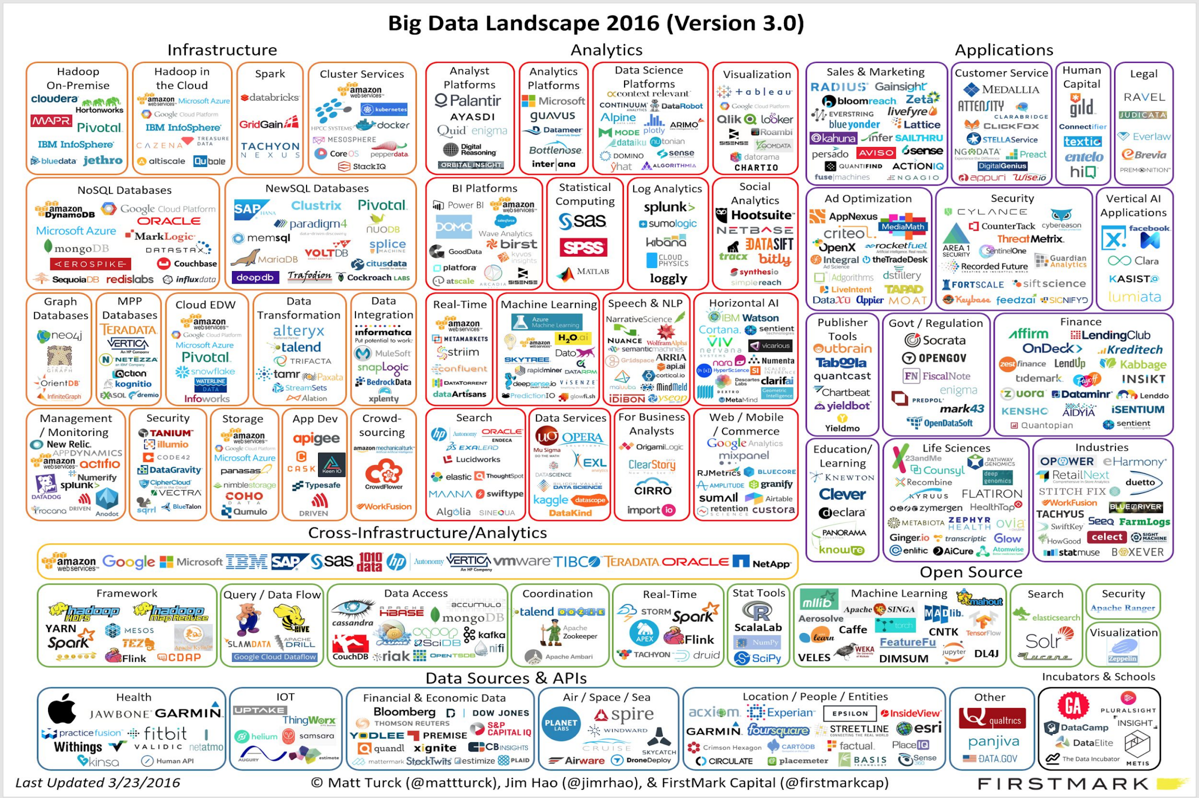
- Popular columns and rows model
- Relational databases
- Conceptually simple to related to

Variety challenges

- Unified database schema impractical
- Graph structures
- Some data is *semi-structured* (JSON)
- Some data is *unstructured* (e.g. images and videos)

See *previous database module* to understand data model trade-offs

Big Data Systems



See https://hadoopecosystemtable.github.io

Batch processing: examples

- Calculate a monthly payroll summary.
- Train a machine learning model.
- Analyse a large set of statements to build regulatory compliance reports.

Batch processing

- Data at rest
- High latency
- Concerned with volume and variety

Stream processing: examples

- Classify an incoming email as spam or not.
- Detect whether a bank transaction is fraudulent.

Stream processing

- Data in motion
- Low latency
- Concerned with velocity

LA >>> Quiz

Which of the following business problems are suited for batch or stream processing?

1. Generate a balance sheet from all the invoices in a month.
2. Provide real-time alerts to stock market changes.
3. Build page-rank metrics for the web.
4. Detect anomalies in network traffic.

Big Data: recap

- Big Data can be characterised by *volume*, *velocity* and *variety*.
- *Batch processing* is used to process all the data in one go.
- *Stream processing* is used to process data in real-time, as it is received.

Hadoop ecosystem

CAMBRIDGE SPARK

Dealing with big data

Too slow

Dealing with big data

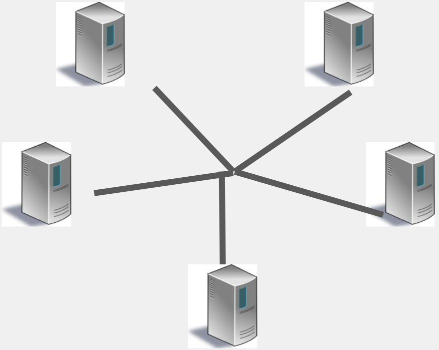
Too slow

Scaling-up

Dealing with big data

Too slow

Scaling-up

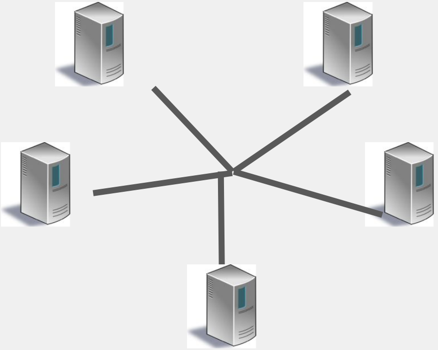


Scaling-out

LA >>> Quiz: What are the pros and cons of each approach?

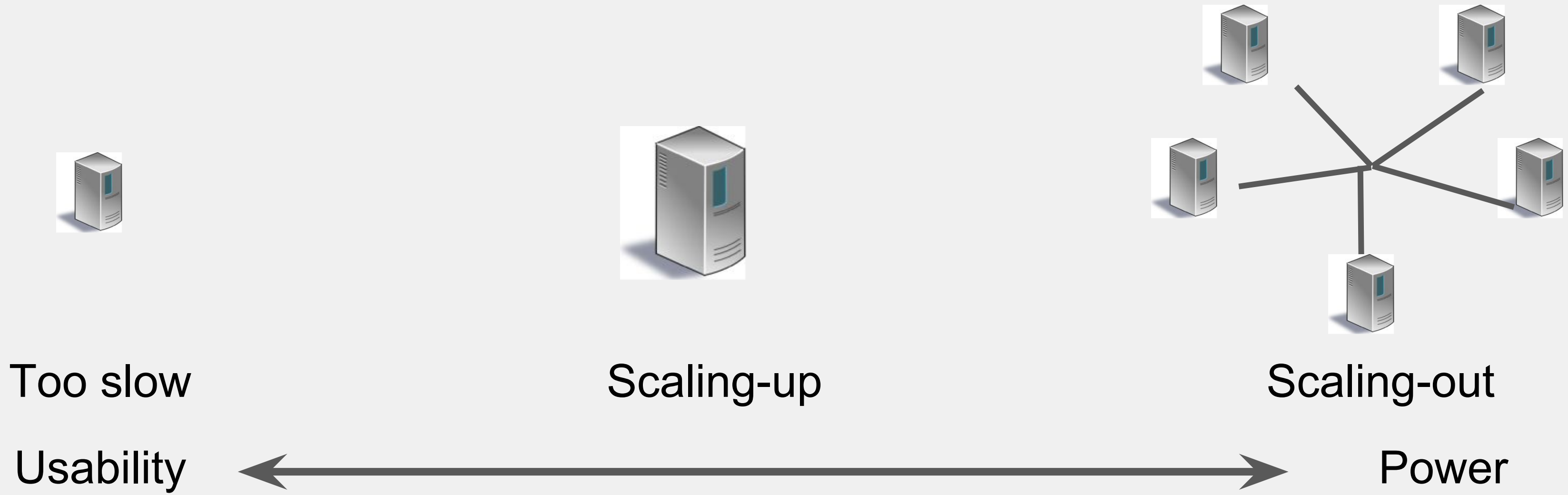
Too slow

Scaling-up



Scaling-out

Dealing with big data

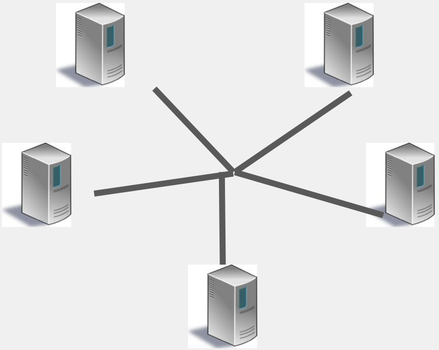


Dealing with big data

Too slow

Scaling-up

- ✓ Same programming model
- ✗ High upfront cost
- ✗ Flexibility
- ✗ Single point of failure
- ✗ Technological limitations

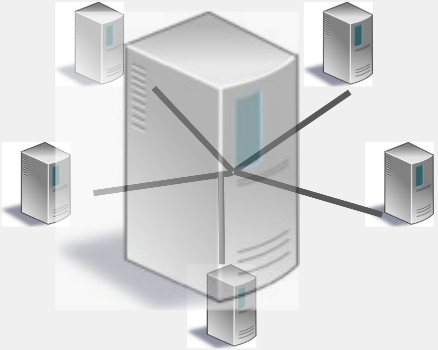


Scaling-out

- ✗ Difficult to use
- ✓ Cheap
- ✓ Flexible
- ✓ Fault tolerant

Dealing with big data

Hadoop ecosystem



Too slow

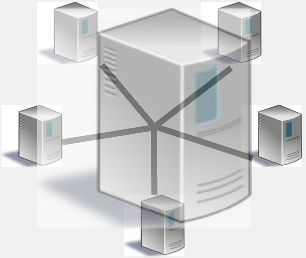
Scaling-up

Scaling-out

Usability ⟷ Power

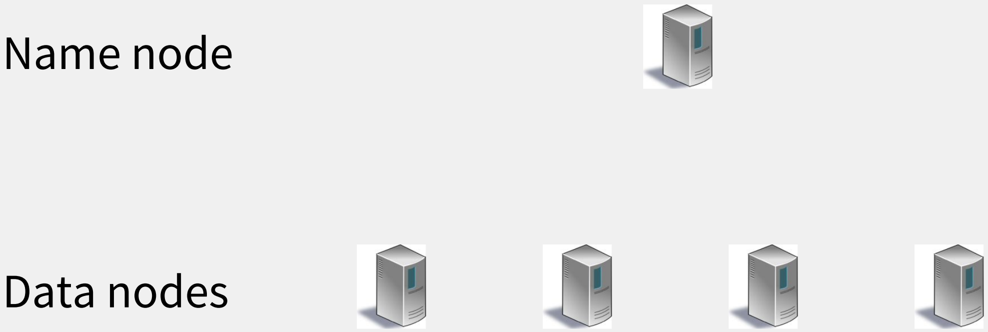
Hadoop ecosystem

Data processing:	Hadoop MapReduce	Apache Spark
Resource management:	Hadoop YARN	
File system:	Hadoop HDFS	



- Scalability
- Fault tolerance

Hadoop HDFS



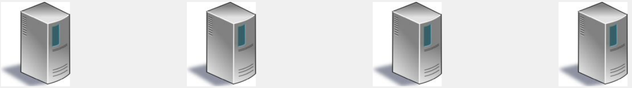
Hadoop HDFS

- Files are broken down into *blocks* and spread across machine
  - ➢ High bandwidth access to the data
  - ➢ Fault tolerance

File /user/alice/data1.txt => 1, 2, 3

Blocks 1, 3 2, 3 1,2,3 1,2

Data nodes

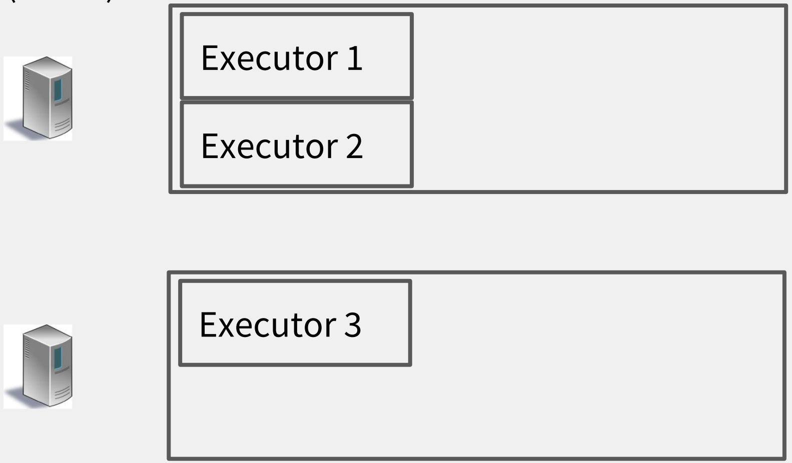


Interacting with HDFS

- File system shell
  - `/bin/hdfs dfs -ls`
  - `/bin/hdfs dfs -mkdir`
  - `/bin/hdfs dfs -copyFromLocal`
  - `/bin/hdfs dfs -copyToLocal`
- Hadoop MapReduce / Apache Spark

Cluster managers

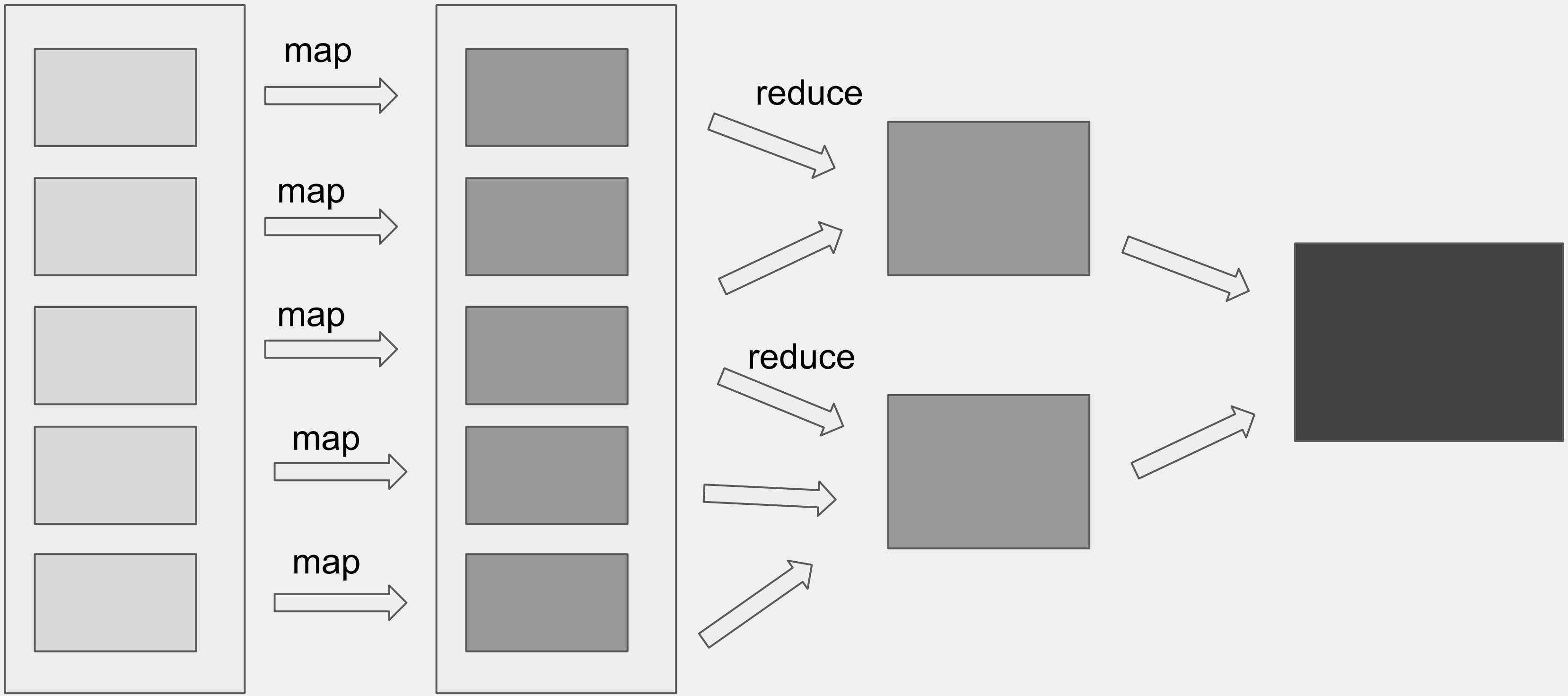
- Examples Hadoop YARN, Apache Mesos, Spark Standalone
- Data processing application requests executors
  - Memory (MB)
  - CPU (Cores)



Hadoop MapReduce vs Apache Spark

	MapReduce	Spark
Initial release	2011	2014
# Operations	2	~20
Iterative computation	No	Yes
Interactive shell	No	Yes
Sorting 100TB	72 mins with 2100 machines	23 mins with 206 machines

Map - Reduce Pattern

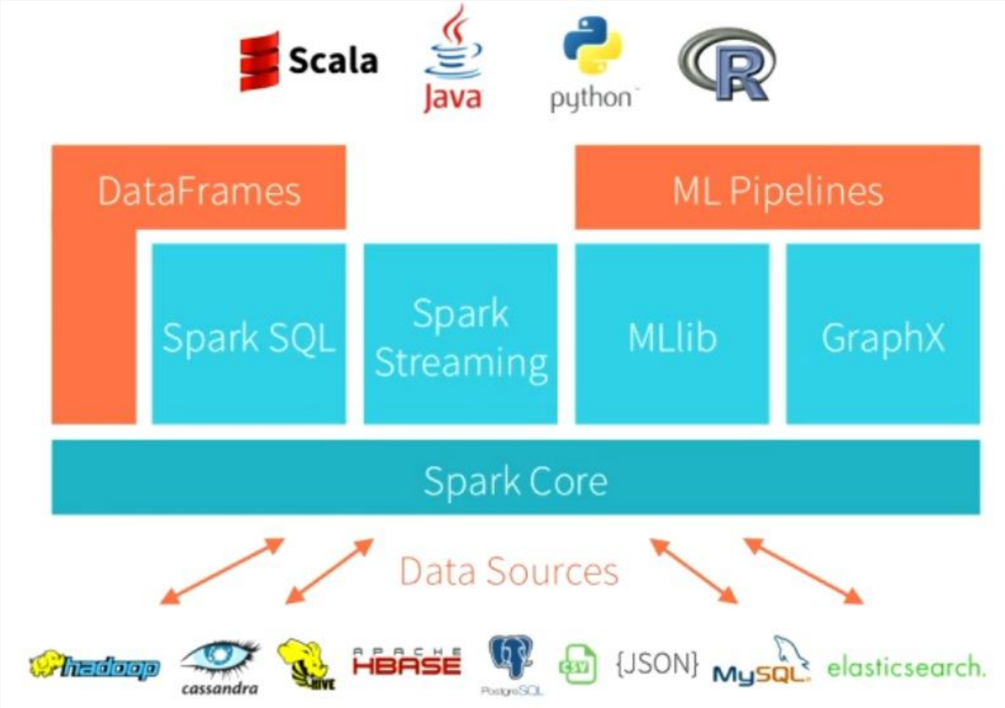


Hadoop: recap

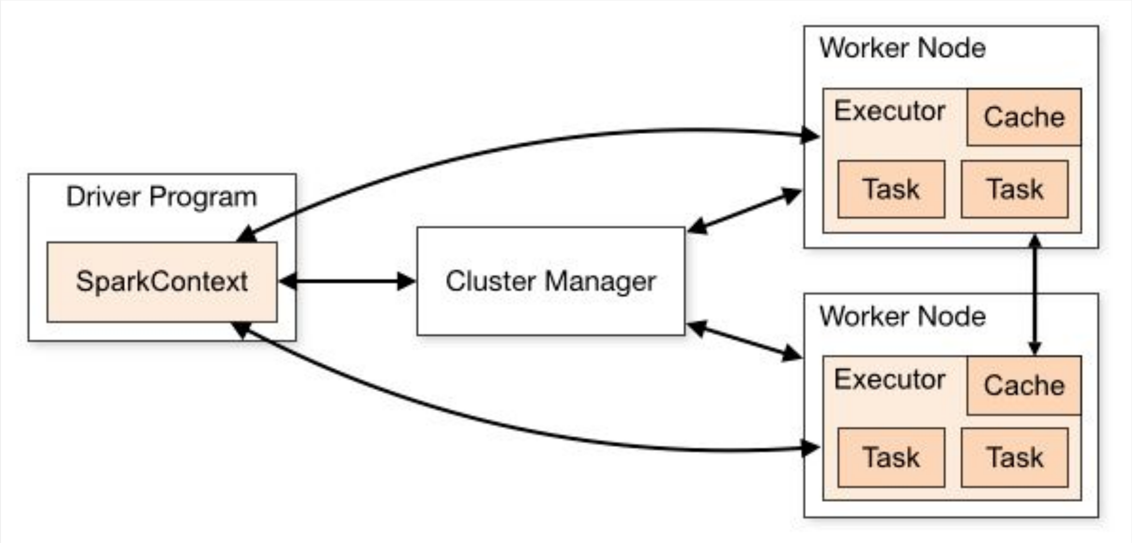
- Hadoop ecosystem makes distributed systems easy to use
- You will mainly interact with two components:
  - HDFS to store large files in a distributed way
  - Apache Spark to do data processing
- One single machine is always easier to work with than a distributed system

Spark RDD

Apache Spark



Apache Spark Architecture



Apache Spark: Resilient Distributed Datasets (RDDs)

- Most basic abstraction
- Collection of elements (e.g. data points)
- Divided across the cluster

RDD

Creating an RDD

Python List

`l = [1, 2, 3, 4]`

`sc.parallelize(l)`

RDD

`1, 2, 3, 4`

File (local or on HDFS)

`f = “data.txt”`

`sc.textFile(f)`

RDD

`line1, line2, ...`

Basic RDD manipulation

- `rdd.count()`: returns the number of elements in your RDD
- `rdd.take(n)`: returns the first n elements in your RDD

Transformations and Actions

- Transformations: operations that return a new RDD
- Actions: operations that return a value to your Python programs

Quiz:

- Is `count()` an action or a transformation?
- Is `take()` an action or a transformation?

Common transformations: `filter()`

```
Amelia
Jack
Jessica
Olivia
Harry
Lily
```

filter(lambda name: name[0] == ‘J’) →

```
Jack
Jessica
```

Common transformations: map()

```
Amelia
Jack
Jessica
Olivia
Harry
Lily
```

```
map(lambda name: name.upper())
```

```
AMELIA
JACK
JESSICA
OLIVIA
HARRY
LILY
```

Common transformations: `flatMap()`

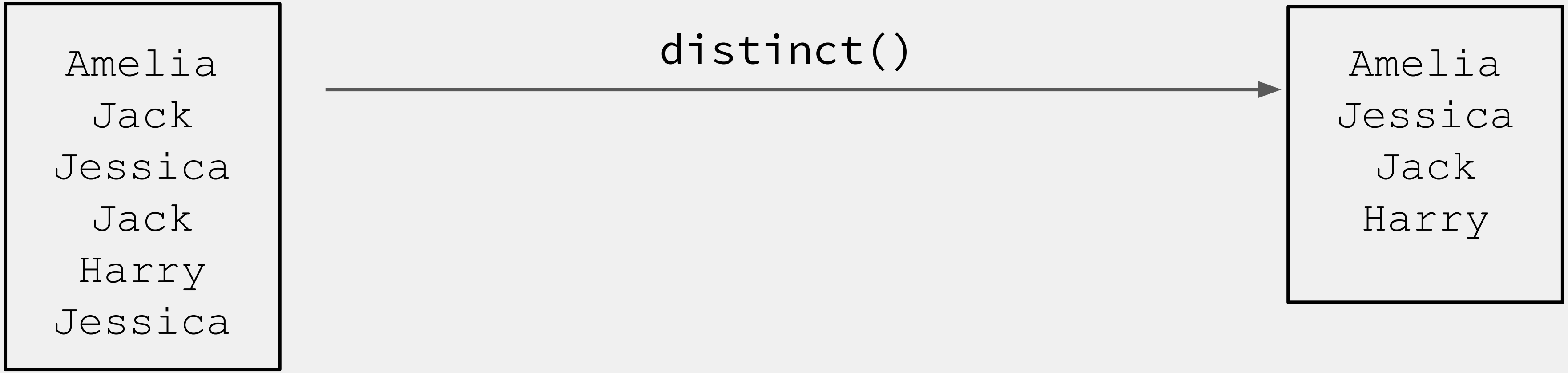
Amelia
Jack
Jessica
Olivia
Harry
Lily

`flatMap(lambda name: name.split('i'))` →

Amel
a
Jack
Jess
ca
Ol
v
a
Harry
L
ly

Note: the function **`split`** returns a list

Common transformations: `distinct()`

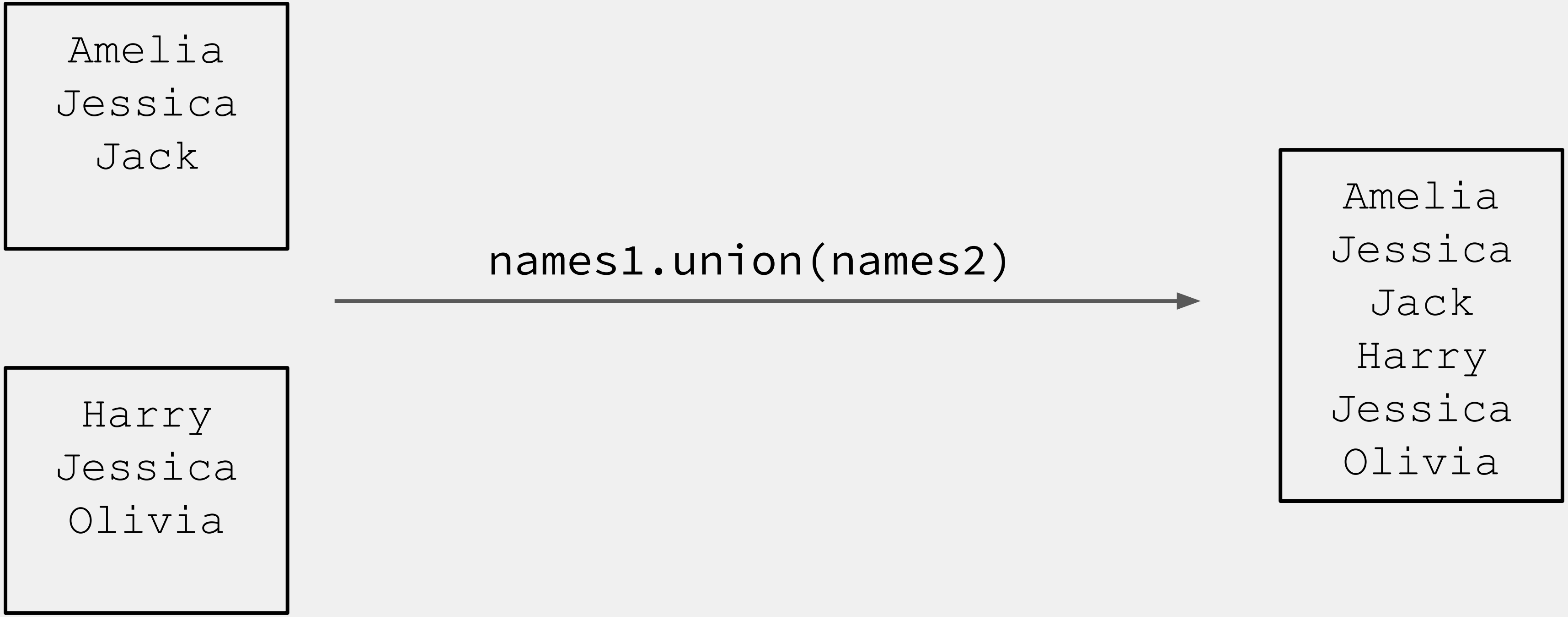


jupyter

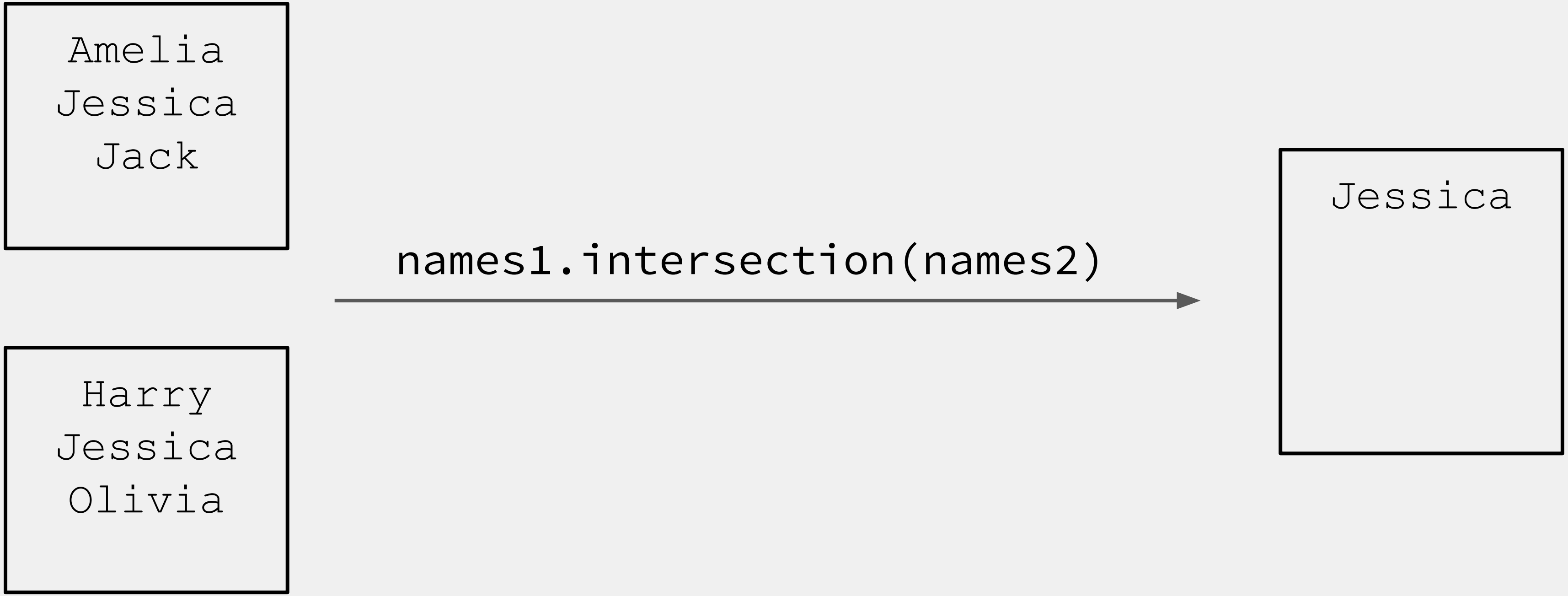
Hands-on session

>>> Create RDDs, `filter()`, `map()` and `distinct()`

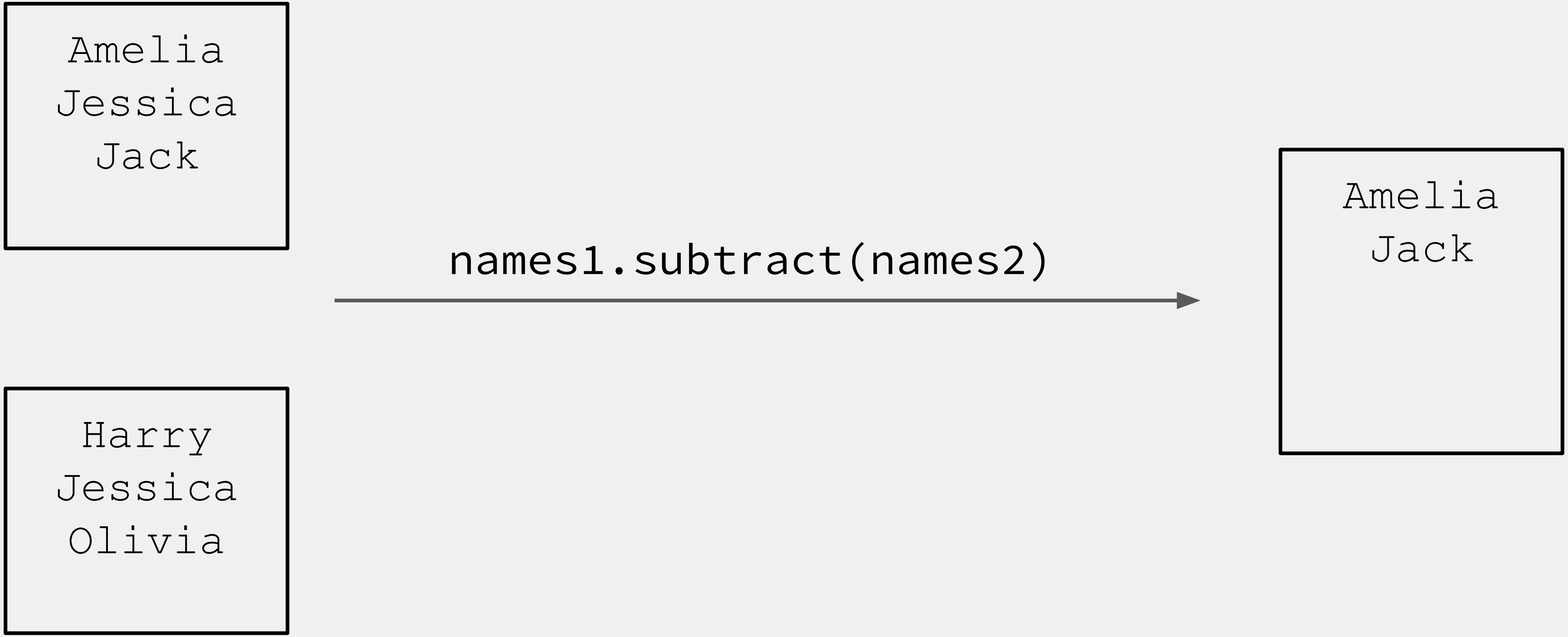
Set-like transformations: `union()`



Set-like transformations: `intersection()`

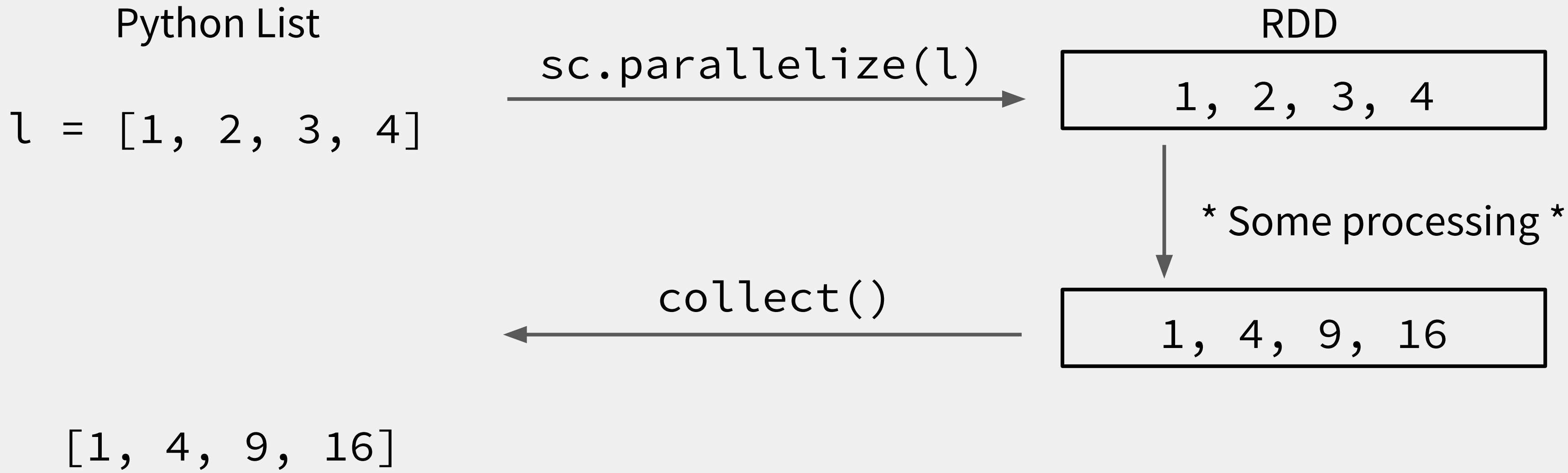


Set-like transformations: `subtract()`



Common actions

- `count()`
- `take()`
- `collect()`: opposite of `parallelize()`, turns an RDD into a Python list



Common actions: `reduce()`

Takes as input a function that processes two elements and returns one

$\boxed{\texttt{1, 2, 3, 4}} \xrightarrow{\texttt{reduce(lambda x, y: x + y)}} \texttt{10}$

Hands-on session

>>> Set like transformations, `reduce()` & bonus

RDD Basics: recap

- RDDs are Sparks main abstraction to hold collections
- They have a rich API to process them
  - Transformations (`filter()`, `map()`, `union()`...) return processed RDDs
  - Actions (`count()`, `collect()`, `reduce()`...) return values to the Python program

Lazy evaluation

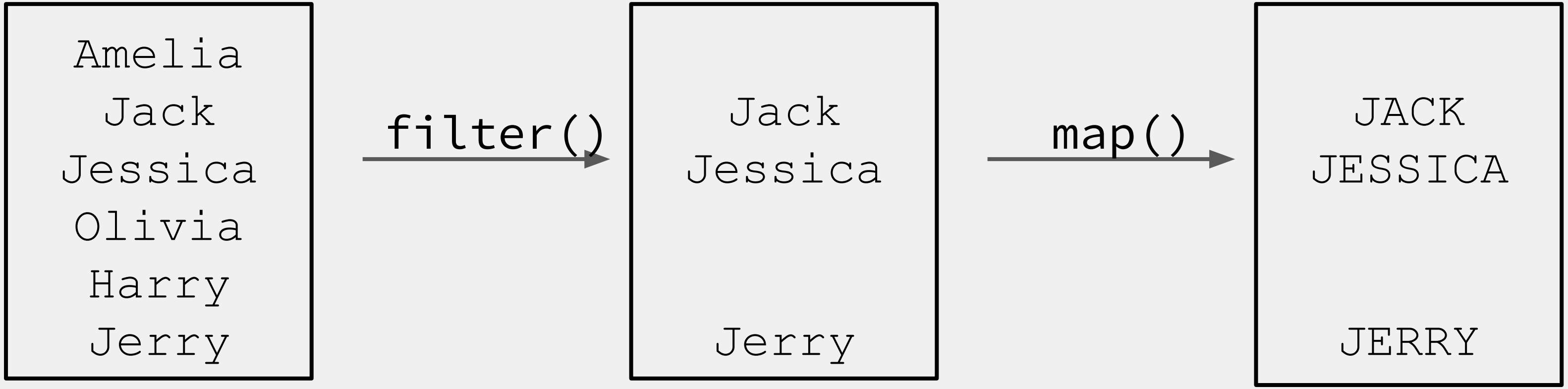
All transformations are *lazy*, they are only executed once an action gets called.

Similar in concept to a recipe:

- Transformations (e.g. `filter()`, `map()`) add instructions onto the recipe
- Actions (e.g. `count()`, `collect()`) process the entire recipe to return a result

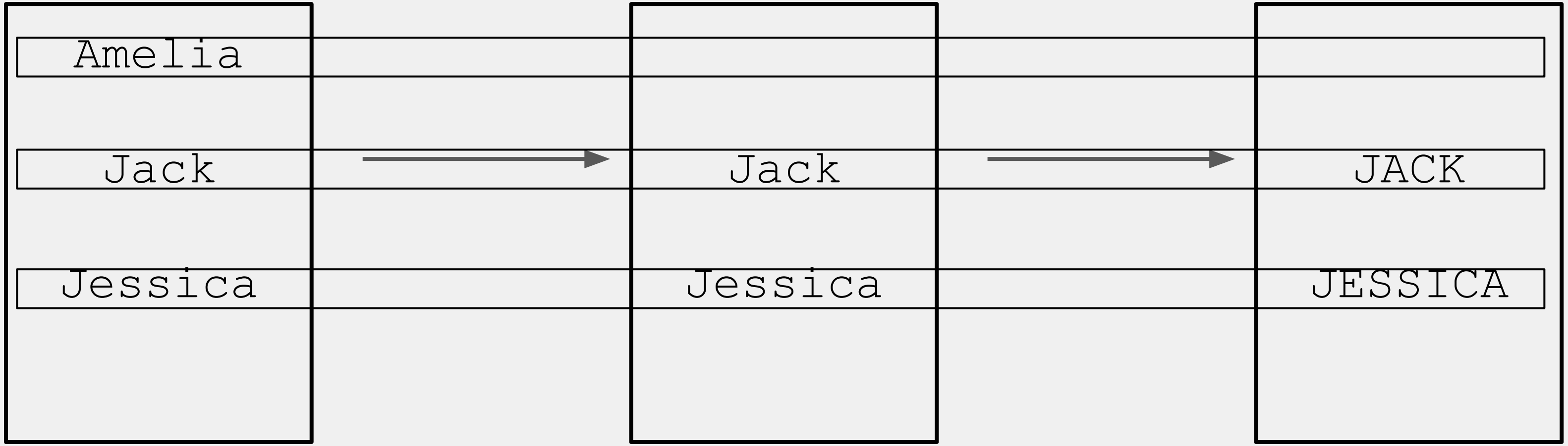
Lazy evaluation

Allows Spark to batch together transformations:



Lazy evaluation

Allows Spark to batch together transformations:



Pair RDDs

Spark's way of storing key/value pairs

Normal RDD where elements are tuples

```
(key, value)
```

Note that in Python, the key and value of a tuple can be accessed as:

```
key = key_value[0]

value = key_value[1]
```

Transformations on pair RDDs: `groupByKey()`

```
Alice: skiing
Bob:     cats
Bob:   coffee
Greg:   pasta
Alice:   cars
Alice:   dogs
```

`groupByKey()` →

```
Alice:[skiing, dogs, cars]
Bob:        [cats, coffee]
Greg:              [pasta]
```

Transformations on pair RDDs: `reduceByKey()`

```
Alice:   4
Bob:     2
Bob:     1
Greg:    4
Alice:   3
Alice:   3
```

```
reduceByKey(lambda x, y: x+y)
```

```
Alice: 10
Bob:    3
Greg:   4
```

Some useful pair RDD transformations

- `keys()`: an RDD of the keys
- `values()`: an RDD of the values
- `mapValues(func)` and `flatMapValues(func)`: Apply `func` onto the values

Hands-on session

>>> Word count in Spark, average of each key

Quiz >>> Implement lookup as a transformation

- `lookup(key)`: returns the values associated with a key

RDD Further: recap

- Transformations are lazy
- Pair RDDs store keys and values
- They have a few specific transformations
  - `groupByKey(), reduceByKey()...`
- Joins are useful to merge pair RDDs together

Spark dataset challenge:
Analysing the HackerNews dataset

CAMBRIDGE SPARK