Data Analytics with HPC

Apache Spark

Reusing this material

archer

epcc | THE UNIVERSITY OF EDINBURGH

Outline

- What is Apache Spark
- How to use it
- Example scenario
- Available libraries
- Demo

What is Apache Spark

- Open-source distributed data analytics platform
- Runs on a standalone cluster or on Hadoop (and others)
- Large community
- Many libraries that are actively being developed
  - MLlib: machine learning
  - DataFrames, Datasets, and SQL
  - Structured Streaming
  - GraphX
  - SparkR
- Many third-party libraries

How to use it

- Interactive mode for testing and development
  - On local machine using shared memory and one or more cores
  - Or interacting with cluster
- Job submission to a cluster manager
  - Spark Standalone cluster
  - Hadoop YARN
  - Apache Mesos
  - Amazon EC2

More details

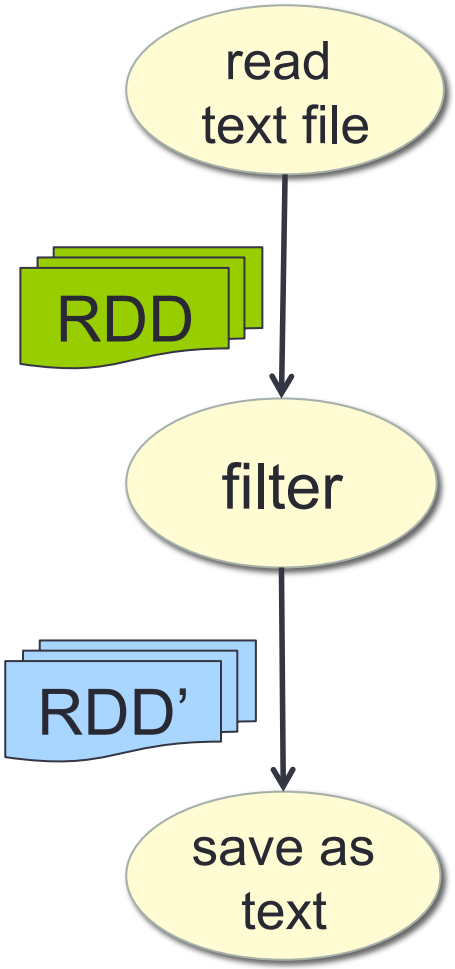
- Provides access to many data sources
  - HDFS
  - HBase
  - S3
  - …
- Distributes parallel computations across a cluster
- Data is cached reliably
  - Can be faster than Hadoop
  - Improves the performance of iterative algorithms
- Runs on Java VM

Java, Scala, Python, R

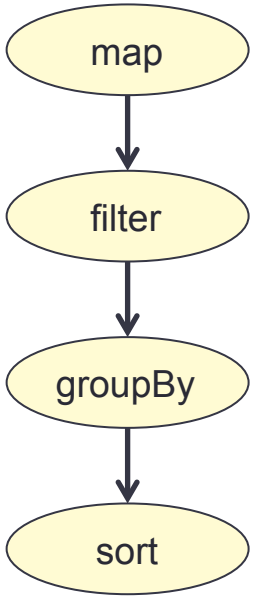
- Spark is written in Scala
  - http://www.scala-lang.org/
  - Compiled to Java byte code
  - Runs on the JVM, i.e. supported on any platforms that run Java
- Client libraries in various languages
  - Scala, Java, Python and R
  - May support only a subset depending on language
    - Not all APIs available in Python yet

Basic functionality

- Resilient Distributed Dataset (RDD)
  - Distributed collection of data items
    - For example, lines from a text file, or
    - Sensor data with timestamp and values
- Apply a chain of:
  - Transformations, e.g.
    - map, filter, group-by, join
  - Actions, e.g.
    - reduce, count, save-as

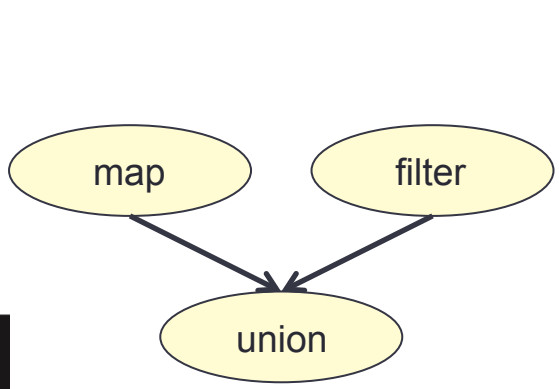


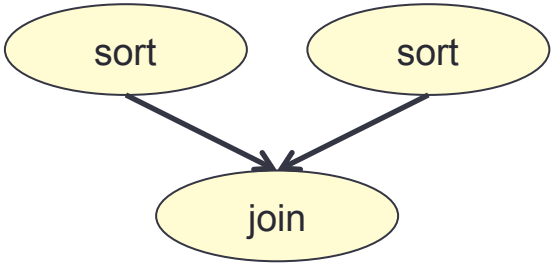
Transformations



Transformation: Apply to create new RDDs.

- For example:
  - *map* (e.g. convert value into another)
  - *filter* (e.g. remove entries outside a valid range)
  - *join* two datasets (match by key)
  - *union*, *intersection*, *distinct*
  - *groupByKey*, *reduceByKey*, *aggregateByKey*
  - *sortByKey*





archer

epcc

THE UNIVERSITY OF EDINBURGH

Examples: Transformations

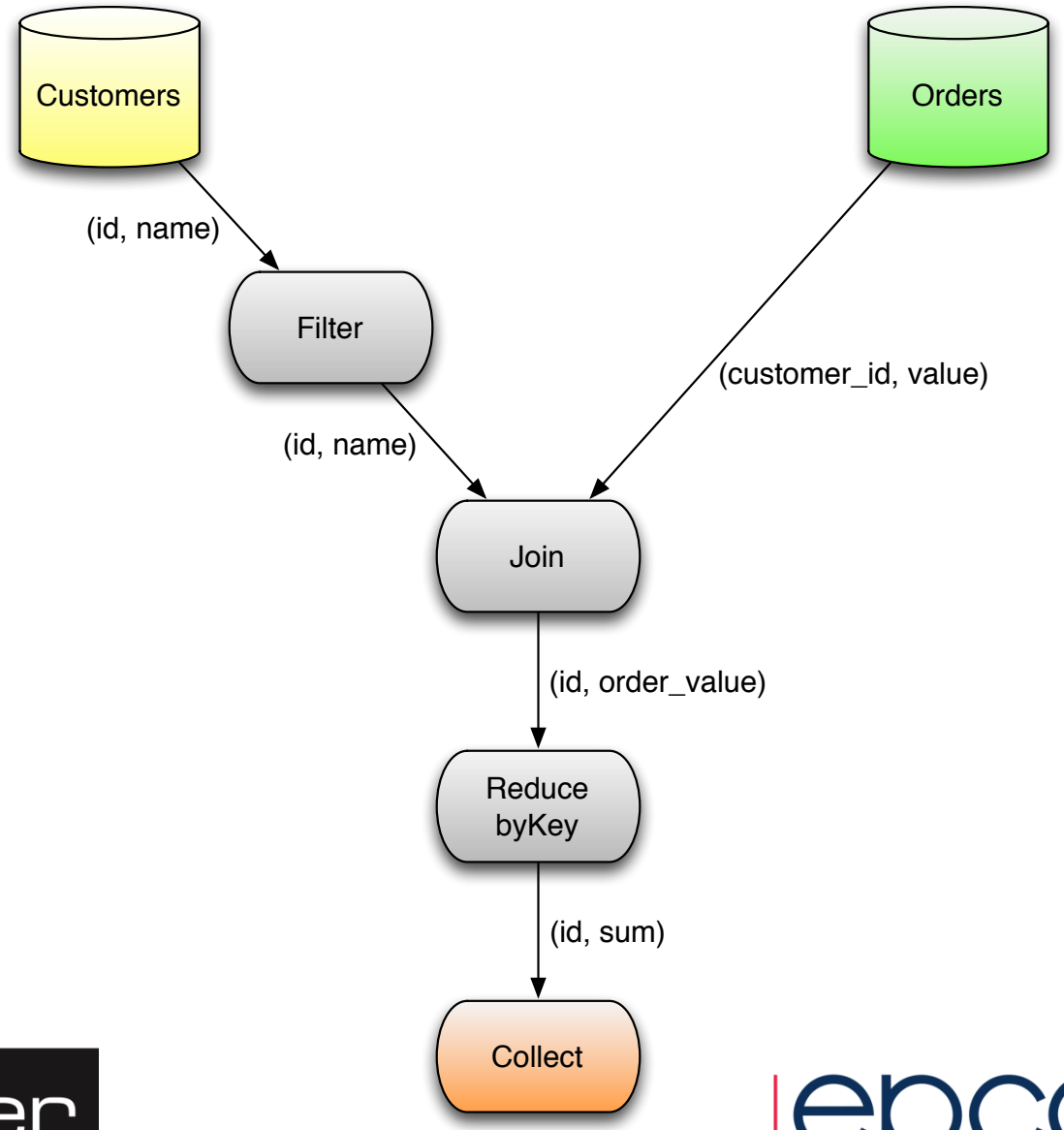
- Map: e.g. x => 2*x
  - 1, 2, 3, 4, 5, 6, 7 => 2, 4, 6, 8, 10, 12, 14
- Filter: e.g. accept if x is between 0 and 100
  - 1, -12, 3, 234, 1, 65, 721 => 1, 3, 1, 65
- Group by key:
  - (A, 1), (A, 2), (B, 5), (B, 5), (C, 17) => (A, [1,2]), (B, [5, 5]), (C, [17])
- Reduce by key: e.g. add values for each key
  - (A, 1), (A, 2), (B, 5), (B, 5), (C, 17) => (A, 3), (B, 10), (C, 17)

Actions

Action: Apply to *materialise* an RDD and create an output dataset. May have side effects.

- For example:
  - *reduce*: take two arguments and return one
  - *count, countByKey*
  - *take, takeSample, takeOrdered, first*
  - *saveAsTextFile, saveAsSequenceFile*: save results to a file or database
  - *foreach*: apply a function to each element

Example



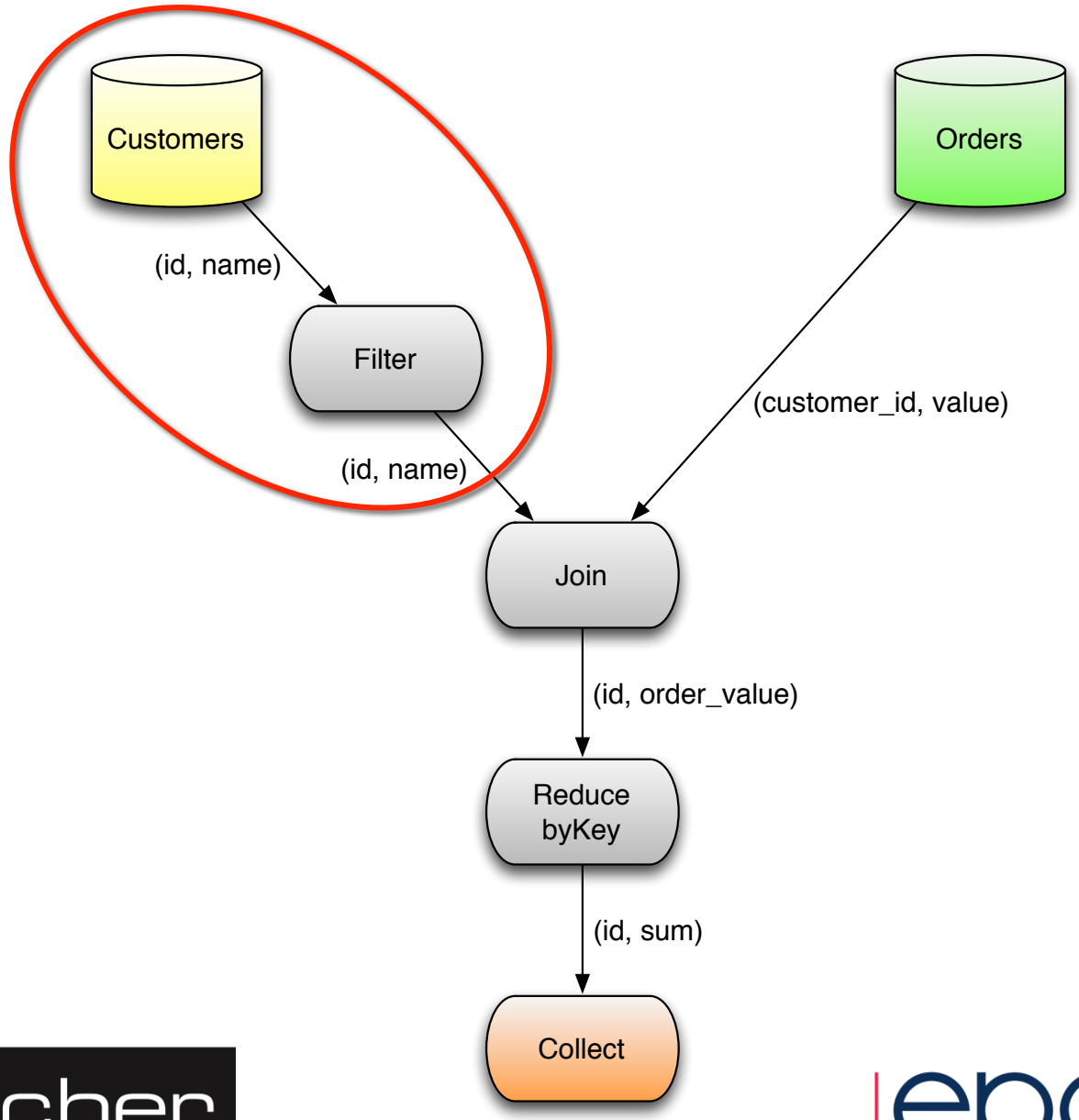
Customers and Orders

Table 1: Customers

ID	Name
1	Alice
2	Bob
3	Charlie

Table 2: Orders

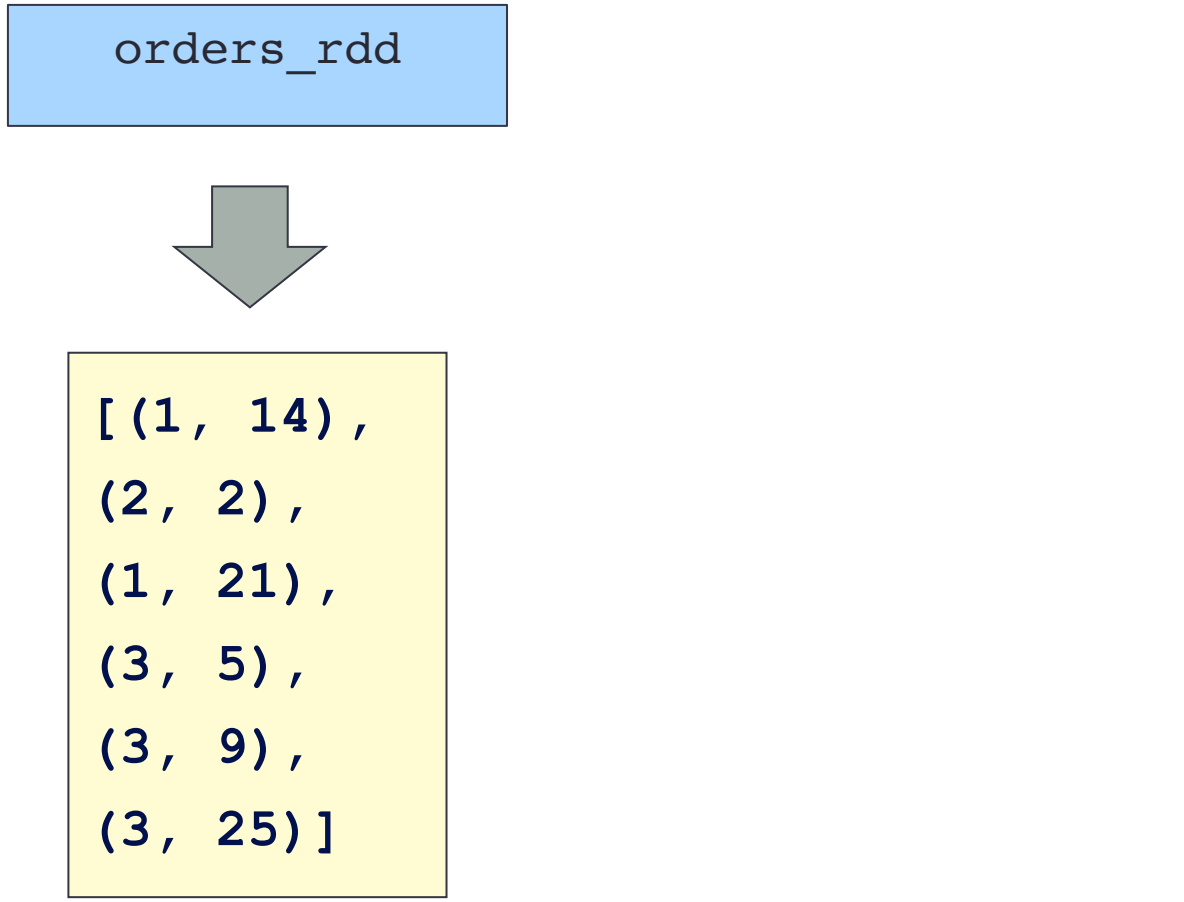
ID	Customer	Order Value
1	1	14
2	2	2
3	1	21
4	3	5
5	3	9
6	3	25

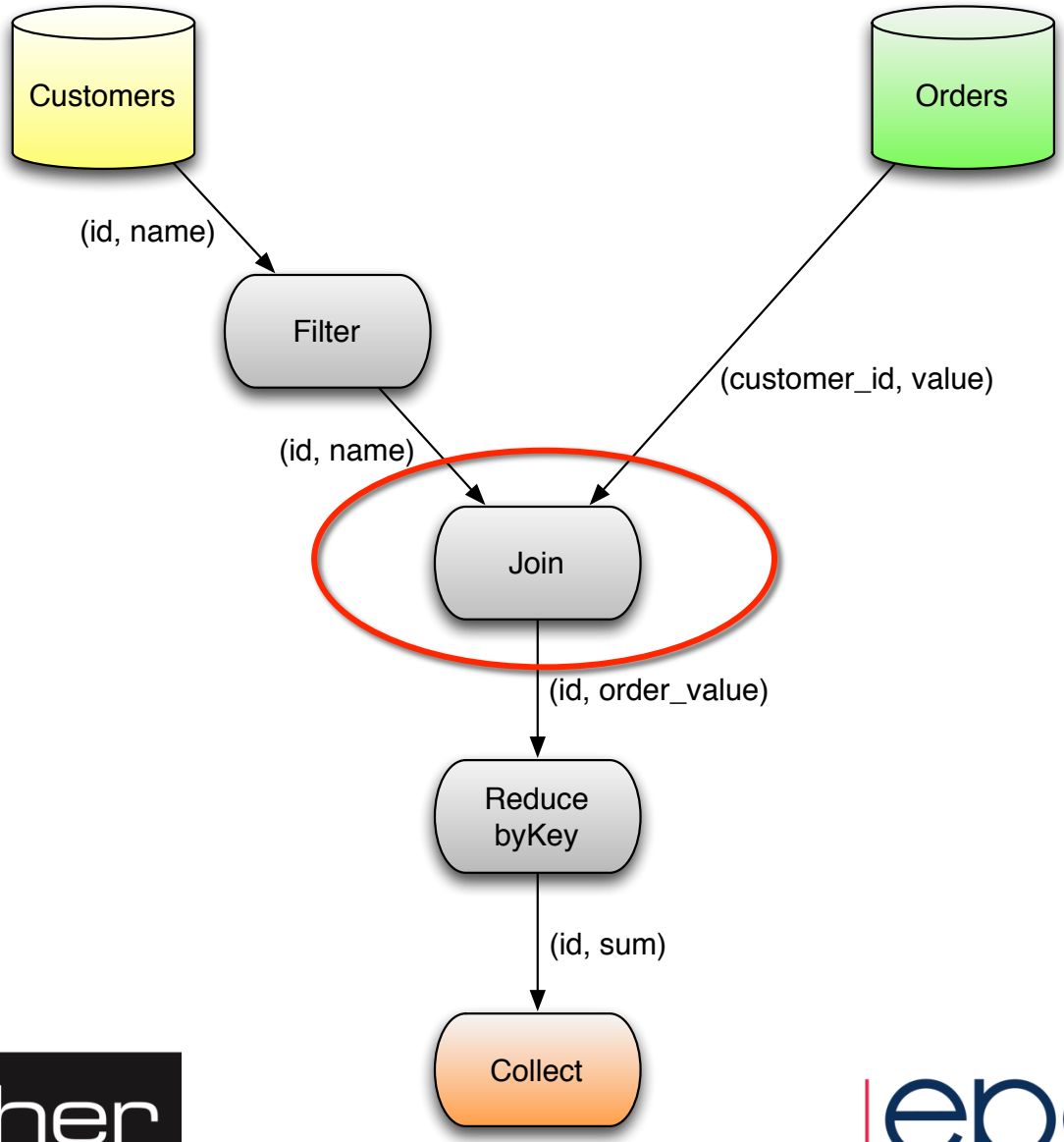

Customers
Orders
(id, name)
Filter
(customer_id, value)
(id, name)
Join
(id, order_value)
Reduce
byKey
(id, sum)
Collect

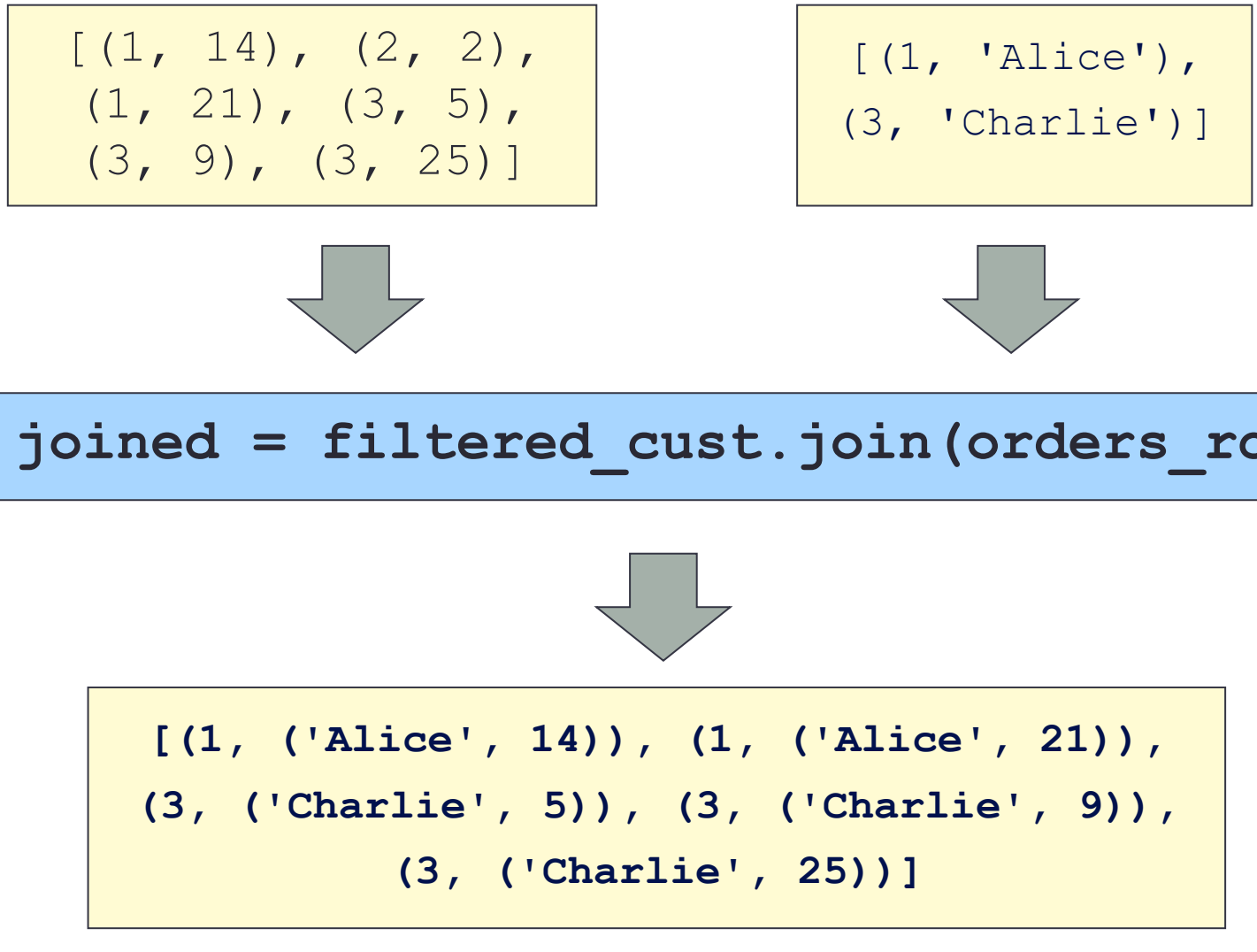
```
[(1, 'Alice'), (2, 'Bob'), (3, 'Charlie')]
```

```
filtered_cust = cust_rdd.filter(
      lambda (id,name): name in ['Alice', 'Charlie'])
```

```
[(1, 'Alice'), (3, 'Charlie')]
```


orders_rdd
[(1, 14),
(2, 2),
(1, 21),
(3, 5),
(3, 9),
(3, 25)]

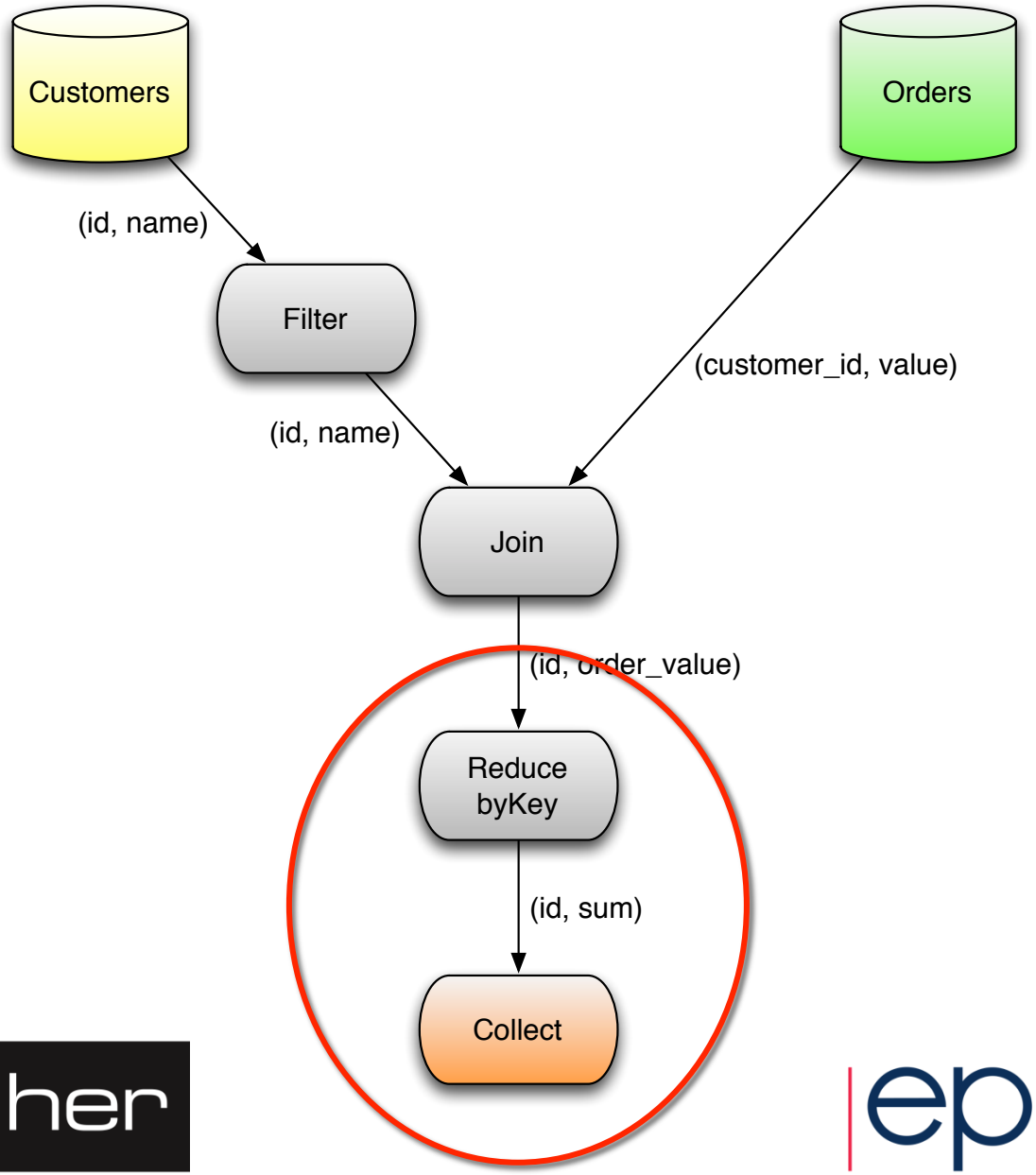

Customers
Orders
(id, name)
Filter
(customer_id, value)
(id, name)
Join
(id, order_value)
Reduce
byKey
(id, sum)
Collect



archer

epcc

THE UNIVERSITY OF EDINBURGH



archer

epcc

THE UNIVERSITY OF EDINBURGH

```
[(1, ('Alice', 14)), (1, ('Alice', 21)),
(3, ('Charlie', 5)), (3, ('Charlie', 9)),
(3, ('Charlie', 25))]
```

```
mapped = joined.map(lambda (k, (v1,v2)): (k, v2))
```

```
[(1, 14), (1, 21), (3, 5), (3, 9), (3, 25)]
```

```
[(1, 14), (1, 21),
(3, 5), (3, 9), (3, 25)]
```

```
sums = mapped.reduceByKey(lambda a, b: a+b)
sums.collect()
```

```
[(1, 35), (3, 39)]
```

Execution

- Transformations are *lazy*: No results are computed until an action is performed
- Computations are broken into tasks and distributed to worker nodes
- Intermediate results are spilled to disk automatically if necessary
- You can explicitly *cache* datasets for reuse

Job submission to a cluster

- Submit job to the master
  - Master listening on *host:port*
- Master distributes tasks to the worker nodes
- Monitor progress in the web UI
  - Task distribution
  - Memory use

Spark Standalone cluster

To run standalone cluster:

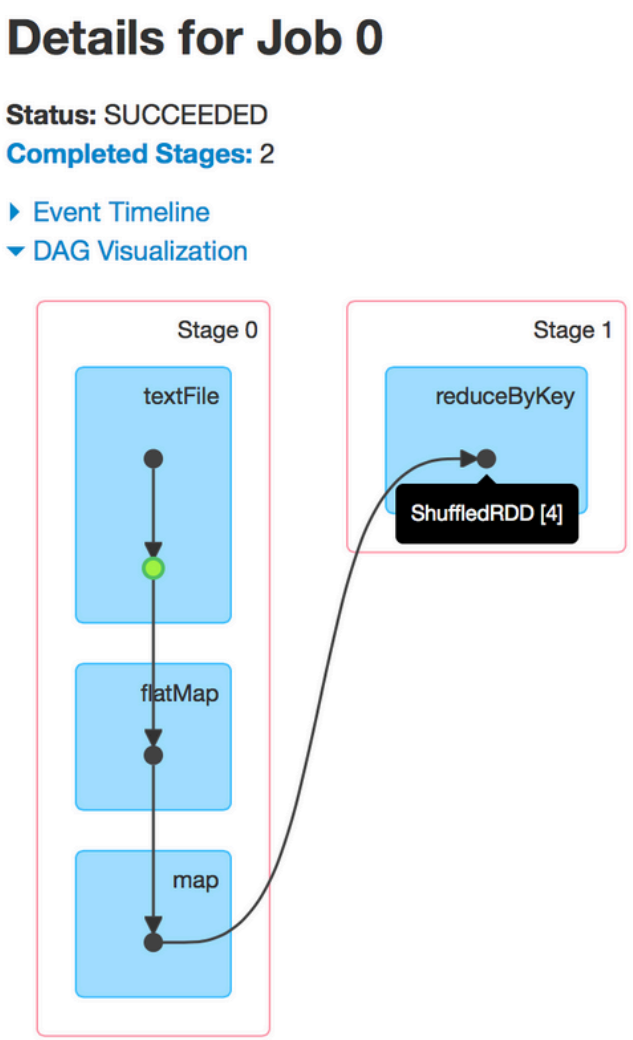
- Start the master node
- Start the worker nodes
  - Workers automatically register with the master (given the URL)
- Master node receives job submissions and distributes tasks to worker nodes

Running on Hadoop YARN

- Requires a Hadoop YARN cluster
- Takes advantage of the functionalities provided by a Hadoop cluster
  - Node management and configuration
  - Distributed file system
  - Data replication
  - Fault recovery

Example application on YARN

"Word Count"



Spark libraries

- MLlib
- Spark Streaming
- GraphX
- SparkSQL and DataFrames

MLlib

- Machine learning library
- Functionality:
  - Basic statistics
  - Classification (Naïve Bayes, decision trees, …)
  - Clustering (k-means, Gaussian mixture, …)
  - And many others!
- Frequent updates with new features

MLlib examples

Basic statistics:

```
summary = Statistics.colStats(data)
print(summary.mean())
print(summary.variance())
```

Correlation:

```
Statistics.corr(data, method="pearson")
```

Classification:

```
clusters = KMeans.train(data, 2,
maxIterations=10, runs=10,
initializationMode="random")
```

SparkSQL and DataFrames

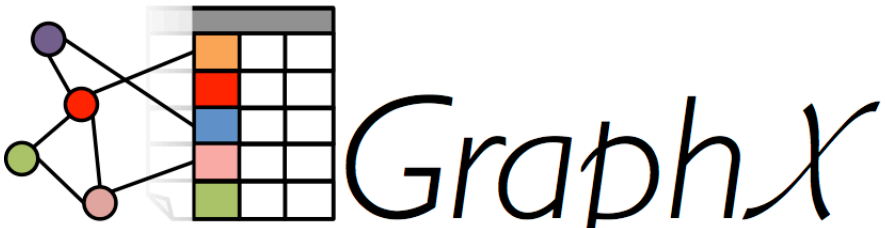
- View datasets as relational tables
- Define a schema of columns for a dataset
- Perform SQL queries
- DataFrame functionality is very popular in R

Spark Streaming

- Data analysis of streaming data
- Aimed at high-throughput and fault-tolerant stream processing
- Based on *discretized streams (Dstream)* containing batches of input data
  - Stream of datasets that contain data from a certain interval (or "window")
- Some APIs currently not available in Python

GraphX

- Graph Processing Library
- Defines a graph abstraction
  - Directed multi-graph
  - Properties attached to each edge and vertex
  - RDDs for edges and vertices
- Graph operations
  - numEdges, numVertices, …
  - triangleCount, connectedComponents
  - collectNeighbors
  - joinVertices
  - …



archer

Summary

- Apache Spark is a framework for data analysis
- Easy to learn
- Widely used
- Active user community
- Comes with a set of machine learning libraries
  - Actively being developed and extended

Spark Demo: k-means clustering

- Interactive PySpark on your local machine
- Interactive PySpark running on a Hadoop cluster
- Job submission to a Hadoop cluster

https://github.com/akrause2014/DataScienceCourse/

```
$ PYSPARK_DRIVER_PYTHON=jupyter
PYSPARK_DRIVER_PYTHON_OPTS="notebook"  bin/pyspark

[I 12:46:02.365 NotebookApp] Serving notebooks from local directory:
[I 12:46:02.365 NotebookApp] 0 active kernels
[I 12:46:02.365 NotebookApp] The Jupyter Notebook is running at: http://
localhost:8888/?
token=c5192c759583f0a499eab119e7d5ed4fbdb6fe5bd56df971
[I 12:46:02.365 NotebookApp] Use Control-C to stop this server and shut
down all kernels (twice to skip confirmation).
[C 12:46:02.366 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first
time, to login with a token:
        http://localhost:8888/?
token=c5192c759583f0a499eab119e7d5ed4fbdb6fe5bd56df971
```