Programming for Data Science at Scale

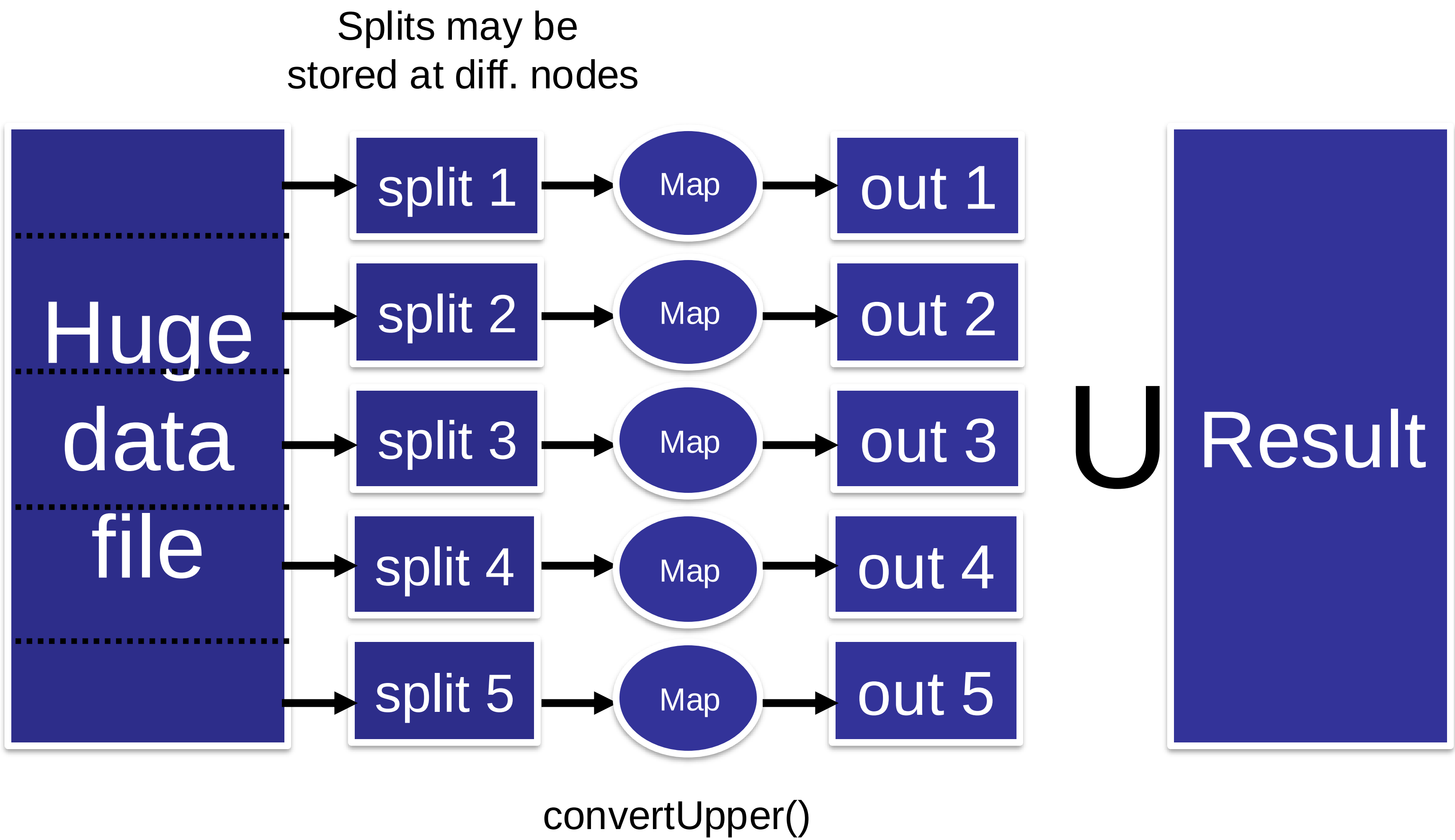
Data-Parallel Programming Model

THE UNIVERSITY of EDINBURGH

Amir Shaikhha, Fall 2024

The vision…

Sample function: convert all text to upper case

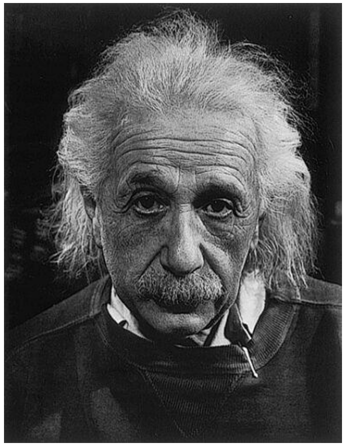


The vision (2)

More complicated: the word-count problem

- Huge file → extract frequencies of words
- Example

Logic will get you from A to B.
Imagination will take you everywhere.



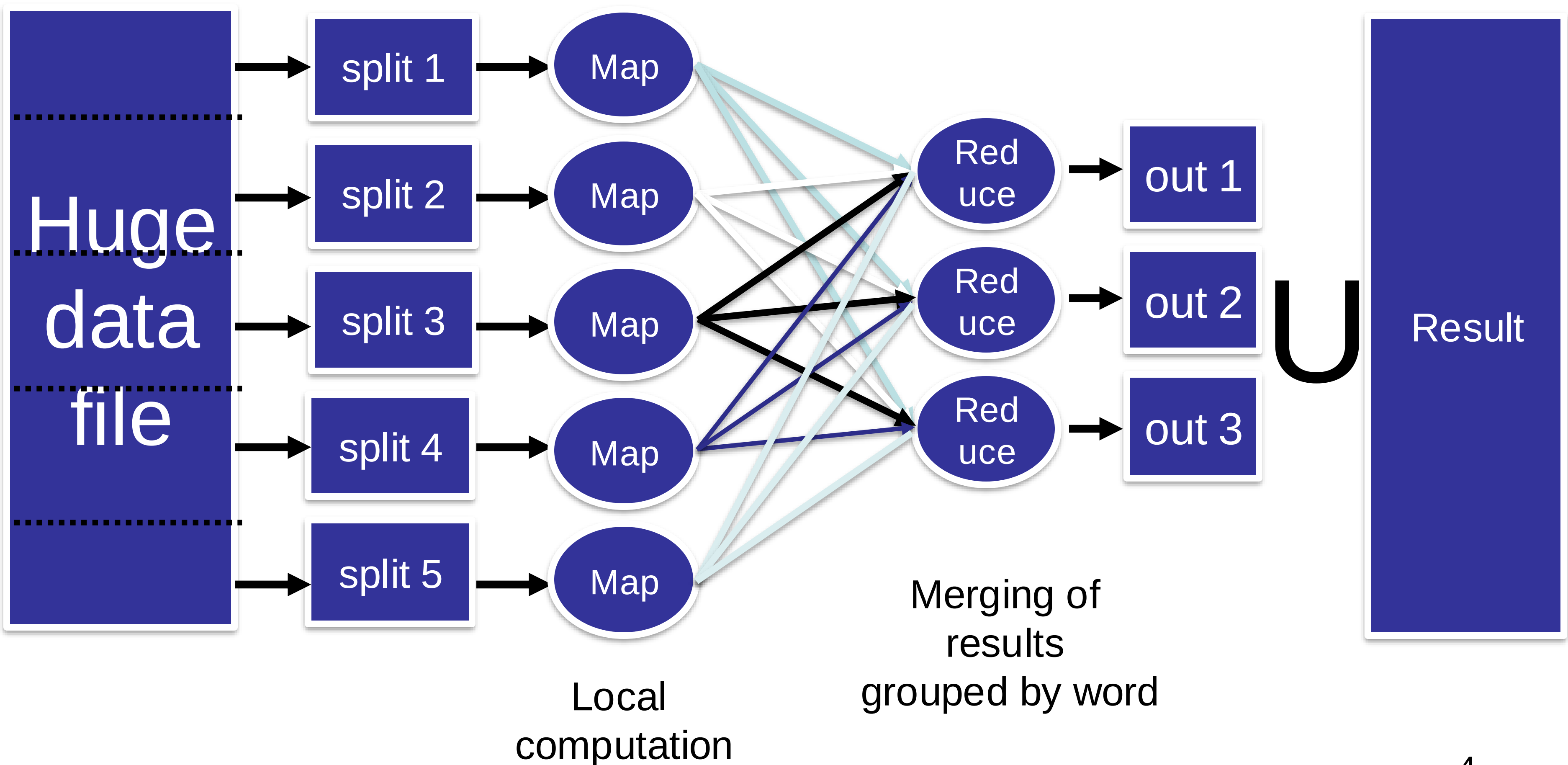
Einstein once said…

Extracted frequencies:

- <Logic,1>, <will,2>, <get,1>, <you,2>, …

The vision (3)

Sample application: the word-count example



MapReduce programming model

- Data model: everything is a <key,value> pair
- Programming model - two core functions
  - Map(key,value): Invoked for every split of the input data. Value corresponds to the split.
  - Reduce(key,list(values)): Invoked for every unique key emitted by Map. List(values) corresponds to all values emitted from ALL mappers for this key.
- These are second-order functions
  - Map(key,value, MapperClassName)
  - Reduce(key,list(values), ReducerClassName)

→ parallelism and deployment handled by the system

MapReduce programming model (2)

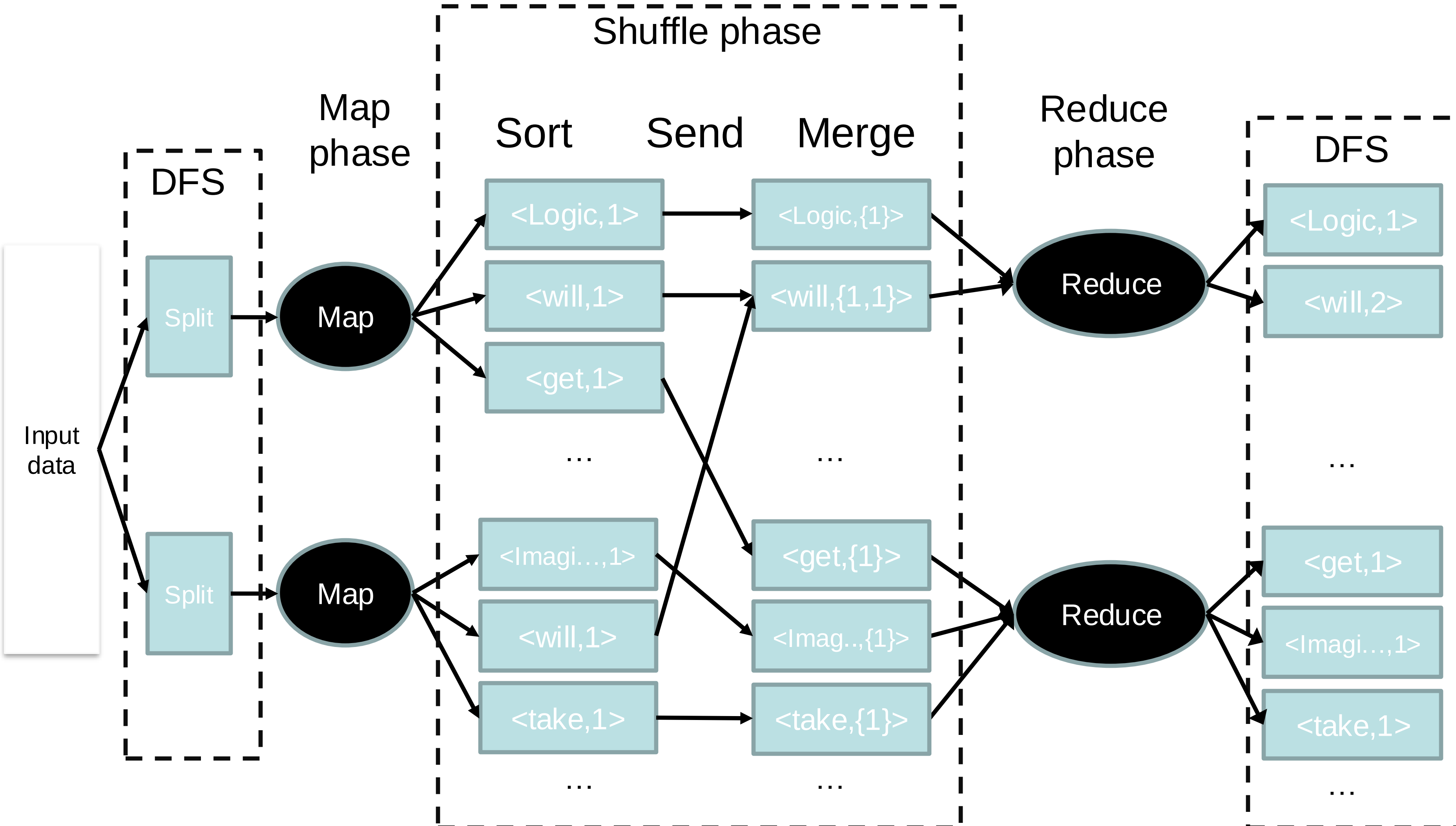
- The word-count problem
  - Input: Text file, broken in splits
  - Output: Frequency of each word observed in the file
  - Map(key,value): value: a split of the text file

```
for each word in value
   emit pair <word,+1>
```

  - Reduce(key,list(values)): Key: word, values: list of (+1's)

```
count=0
for each value in list(values)
   count+=value
emit pair<key,count>
```

MapReduce – under the hood

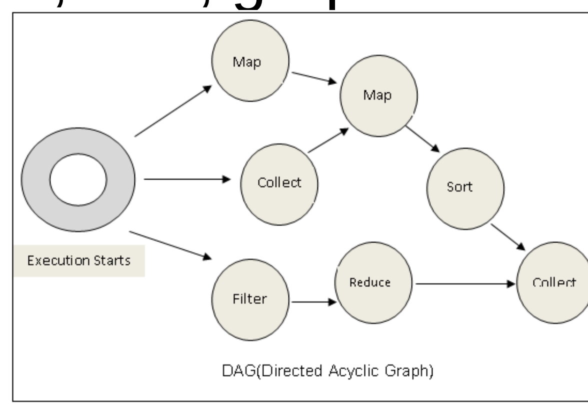



```
for each word in value
    emit pair <word,+1>
```

```
for each value in list(values)
    count+=value
emit pair<word,count>
```

Dataflow programming model

- MapReduce simple but weak for some reqs.
  - Cannot define complex processes
  - Everything file-based, no distributed memory
  - Procedural → difficult to optimize
- Dataflow
  - Processing expressed as a DAG, tree, graph with cycles, …
  - Vertices: processing tasks
  - Edges: Communication
    - DAG: Spark, Dryad
    - Tree: Dremel
    - Directed graph with cycles: Pregel



Spark DAG example

Dataflow programming model (2)

- Describing the processing tasks
  - Declarative languages, e.g., Dremel

```
SELECT DocId AS Id, COUNT(Name.Language.Code)
WITHIN Name AS Cnt FROM t
WHERE REGEXP(Name.Url, '^http');
```

  - Functional programming, e.g., Spark

```
val wordCounts = textFile.flatMap(line => line.split(" ")).
        map(word => (word, 1)).
        reduceByKey((a, b) => a + b)
wordCounts.collect()
```

  - Domain-specific languages, e.g., Pregel

for graph processing

```
class PageRankVertex
  : public Vertex<double, void, double> {
      public: virtual void Compute(MessageIterator* msgs) {
          const int64 n = GetOutEdgeIterator().size();
          SendMessageToAllNeighbors(GetValue() / n);
      }
  };
```

Why Spark? (1)

Scala **Collections** ⟷ APACHE Spark™

Similar API ☺

Which programming language is this?

```
Integer totalAgeReduce =
roster.stream()
    .map(Person::getAge)
    .reduce( 0, (a, b) -> a + b);


Map<String, List<String>> a = words
     .stream().collect(
         Collectors.groupingBy(w ->
sortChars(w)));
```

PLs that have a functional collection interface like Scala

C++, C#, F#, Clojure, Haskell, Java8, JavaScript, Perl, PHP, Python, Ruby, Scheme, Smalltalk, Standard ML, OCAML, …

See

https://en.wikipedia.org/wiki/Map_(higher-order_function)

Fault Tolerance

- Essential for scaling out
- The main reason behind the success of MapReduce in Google
- Requires writing intermediate data to disk

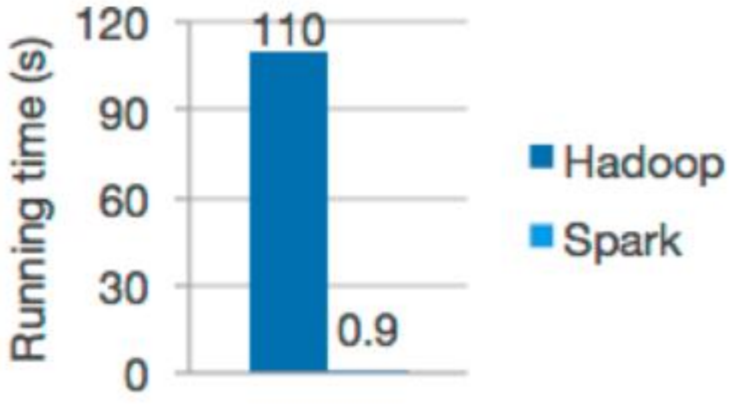
Fault Tolerance in Spark

- Data
  - Immutable
  - In-memory
- Operations = Functional transformations
- Fault tolerance = Replay operations

Why Spark? (2)

- Compared to Hadoop MapReduce, improves efficiency through:
  - General execution graphs
  - In-memory storage

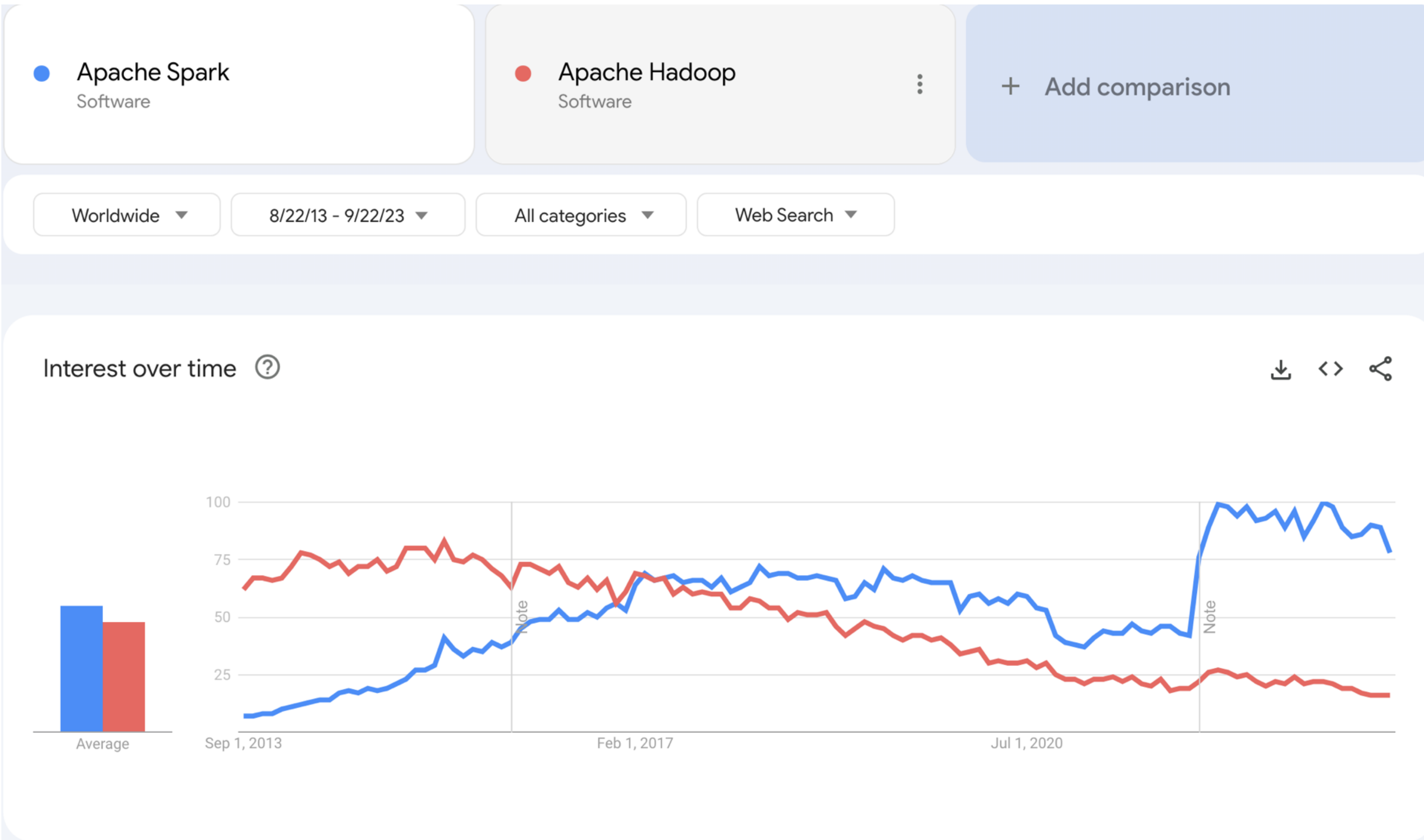
Up to 10 × faster on disk, 100 × in memory



Logistic regression in Hadoop and Spark

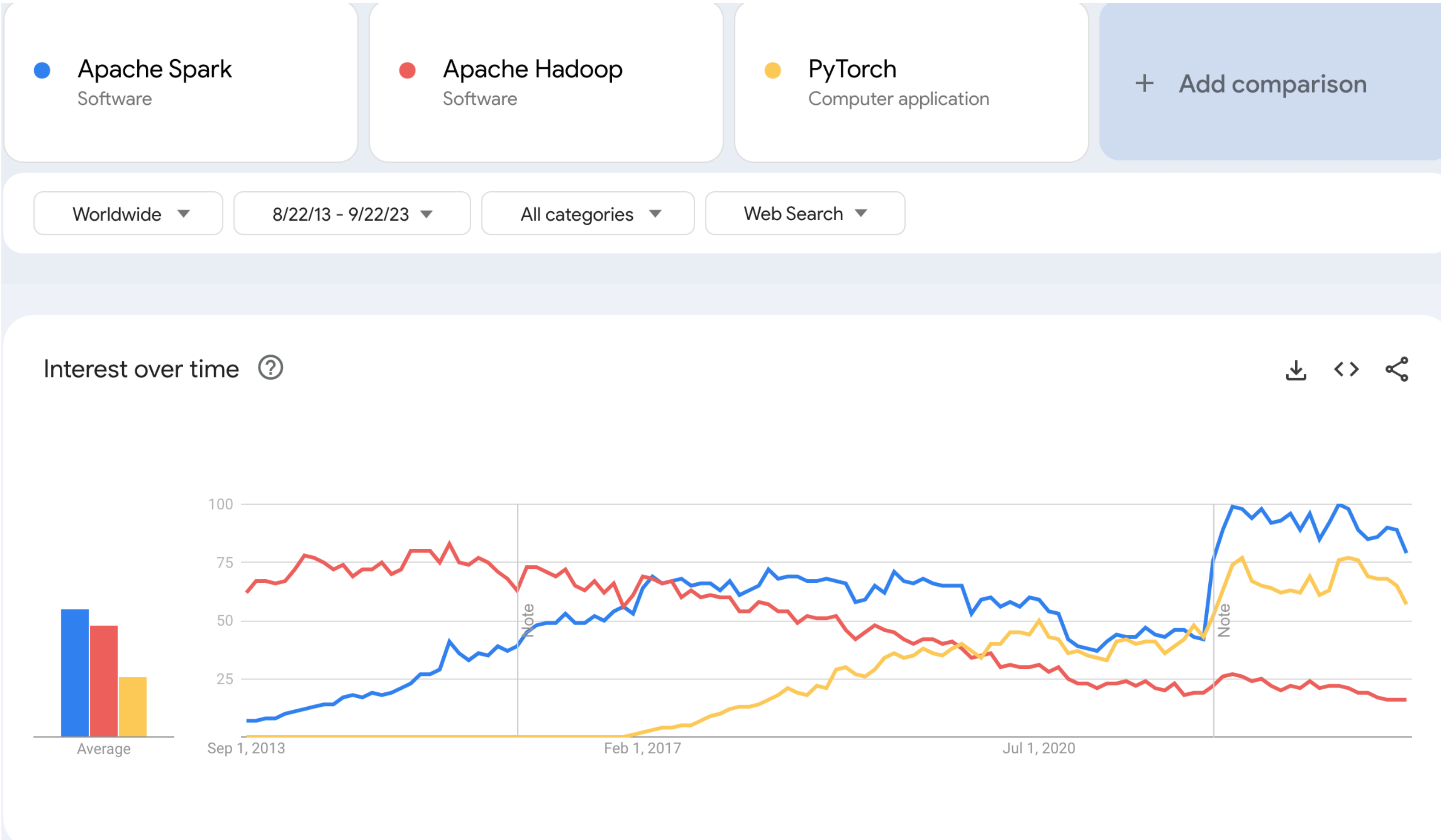
http://spark.apache.org/

Why Spark? (3)



Apache Spark
Software
Apache Hadoop
Software
+ Add comparison
Worldwide
8/22/13 - 9/22/23
All categories
Web Search
Interest over time
100
75
50
25
Note
Note
Average
Sep 1, 2013
Feb 1, 2017
Jul 1, 2020

Why Spark? (4)



Apache Spark
Software
Apache Hadoop
Software
PyTorch
Computer application
+ Add comparison
Worldwide
8/22/13 - 9/22/23
All categories
Web Search
Interest over time
100
75
50
25
Average
Sep 1, 2013
Feb 1, 2017
Jul 1, 2020
Note
Note

Learn Scala

Scala School!

From ∅ to Distributed Service

Other Languages:

- 한국어
- Русский
- 简体中文

About

Scala school started as a series of lectures at Twitter to prepare experienced engineers to be productive Scala programmers. Scala is a relatively new language, but draws on many familiar concepts. Thus, these lectures assumed the audience knew the concepts and showed how to use them in Scala. We found this an effective way of getting new engineers up to speed quickly. This is the written material that accompanied those lectures. We have found that these are useful in their own right.

Lessons

Basics
Values, functions, classes, methods, inheritance, try-catch-finally. Expression-oriented programming

Basics continued
Case classes, objects, packages, apply, update, Functions are Objects (uniform access principle), pattern matching.

Collections
Lists, Maps, functional combinators (map, foreach, filter, zip, folds)

Pattern matching & functional composition
More functions! PartialFunctions, more Pattern Matching

Type & polymorphism basics
Basic Types and type polymorphism, type inference

https://twitter.github.io/scala_school/

QUESTIONS?