UNIVERSITY OF WATERLOO

Data-Intensive Distributed Computing

CS 451/651 431/631 (Winter 2018)

Part 2: From MapReduce to Spark (2/2)

January 23, 2018

Jimmy Lin
David R. Cheriton School of Computer Science
University of Waterloo

These slides are available at http://lintool.github.io/bigdata-2018w/

Source: Wikipedia (The Scream)

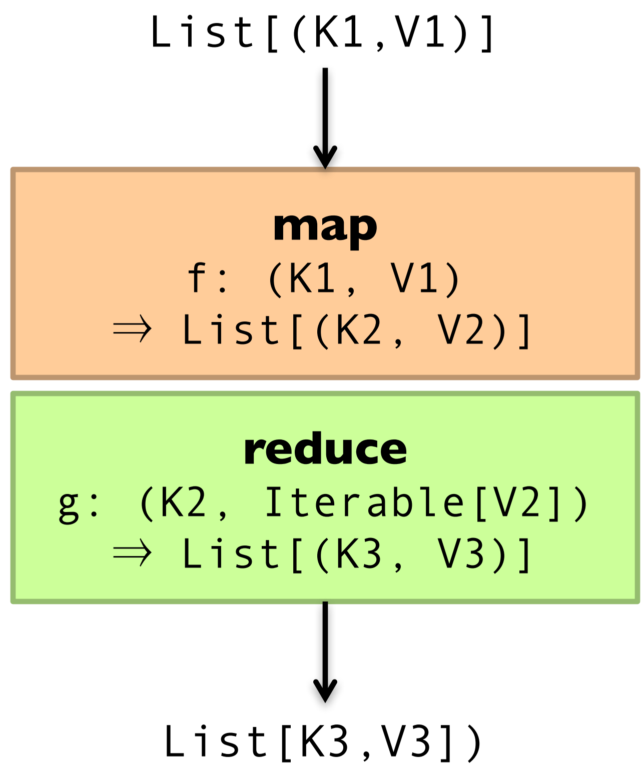
An Apt Quote

All problems in computer science can be solved by another level of indirection... Except for the problem of too many layers of indirection.

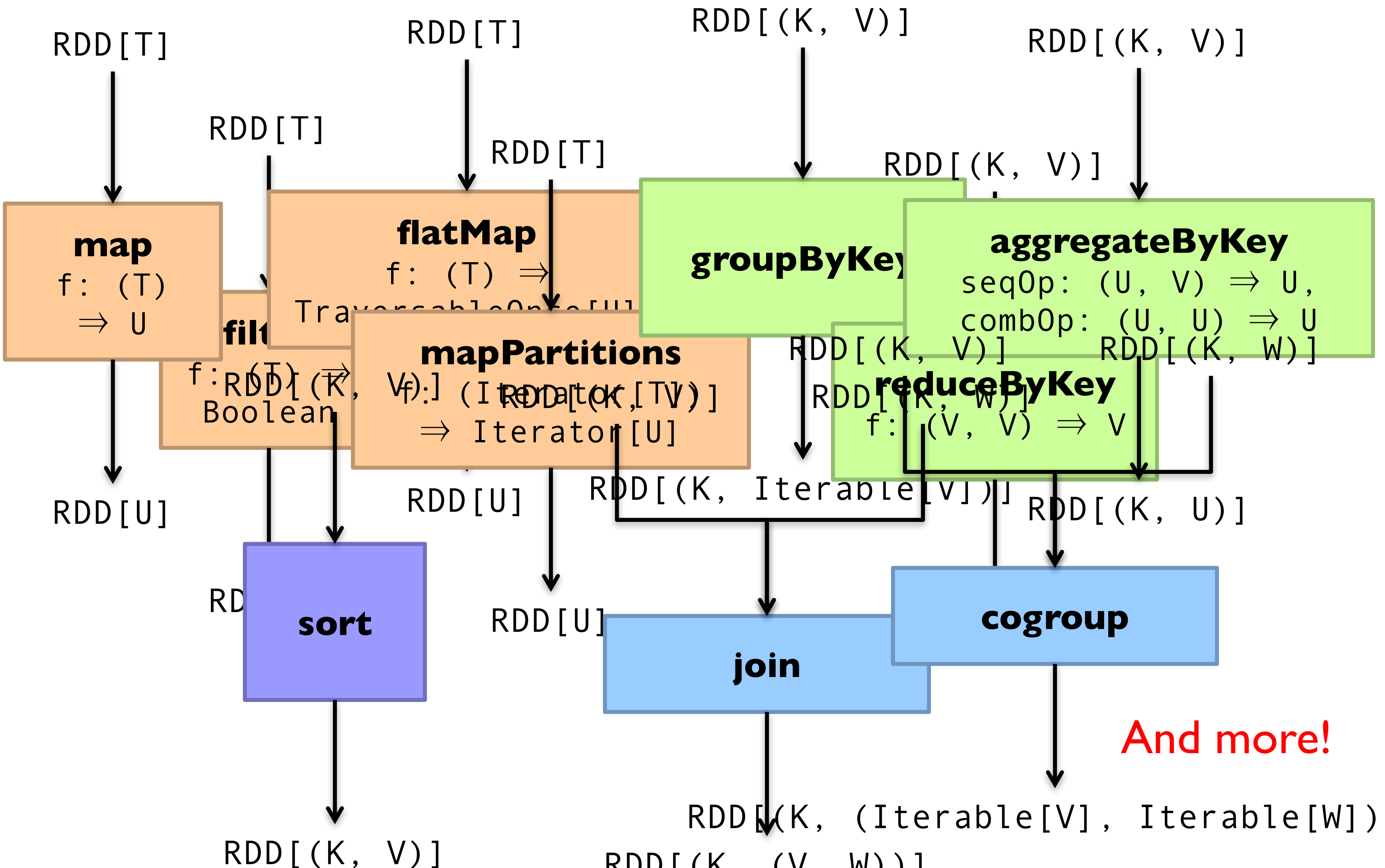
- David Wheeler


The datacenter *is* the computer!
What's the instruction set?
What are the abstractions?
Source: Google

MapReduce


List[(K1,V1)]
map
f: (K1, V1)
⇒ List[(K2, V2)]
reduce
g: (K2, Iterable[V2])
⇒ List[(K3, V3)]
List[K3,V3])

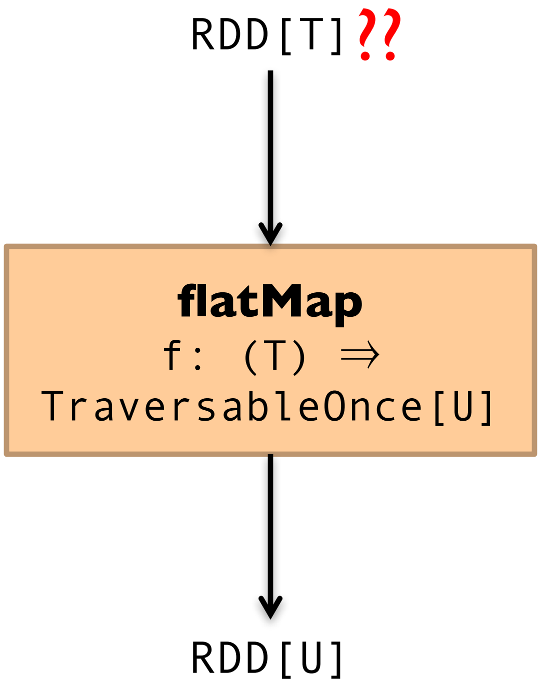
Spark



And more!

Spark Word Count

```
val textFile = sc.textFile(args.input())

textFile
  .flatMap(line => tokenize(line))
  .map(word => (word, 1))
  .reduceByKey((x, y) => x + y)
  .saveAsTextFile(args.output())
```



What’s an RDD?

Resilient Distributed Dataset (RDD)

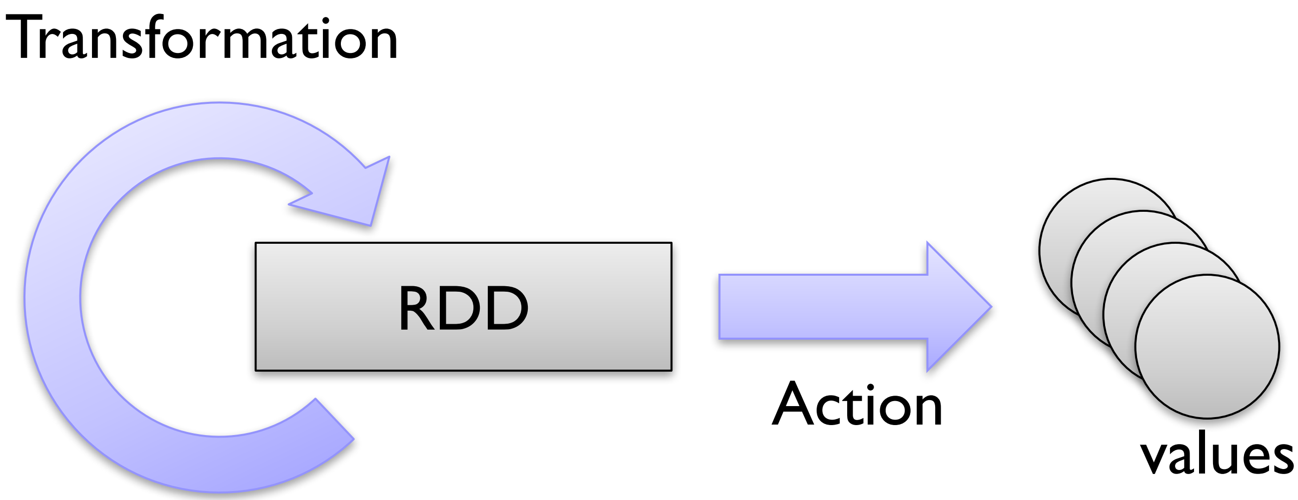
= immutable = partitioned

Wait, so how do you actually do anything?

Developers define *transformations* on RDDs

Framework keeps track of lineage

RDD Lifecycle



Transformations are lazy:
Framework keeps track of lineage

Actions trigger actual execution

Spark Word Count

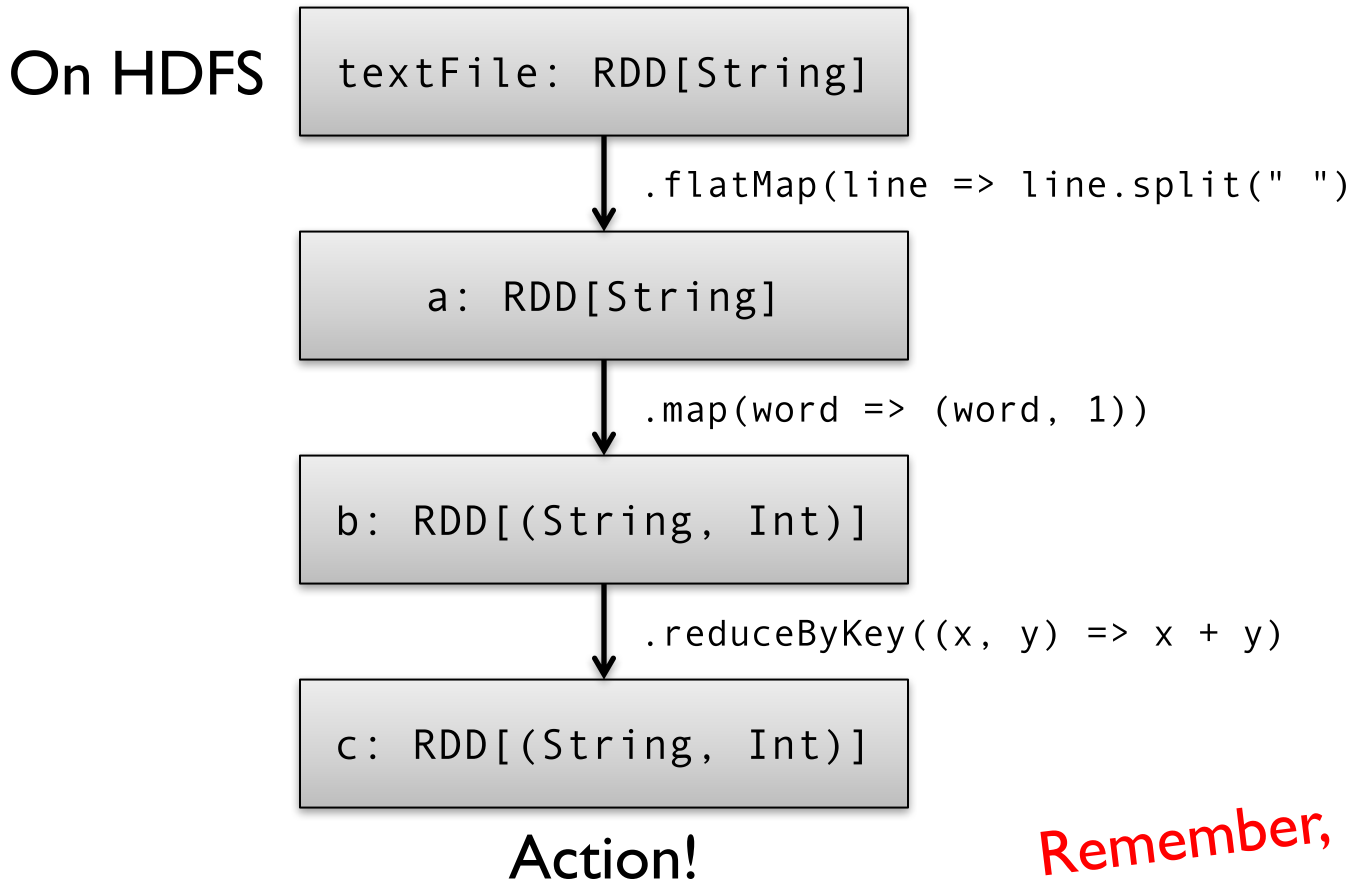
RDDs

```
val textFile = sc.textFile(args.input())

val a = textFile.flatMap(line => line.split(" "))
val b = a.map(word => (word, 1))
val c = b.reduceByKey((x, y) => x + y)

c.saveAsTextFile(args.output())
```

Action

Transformations

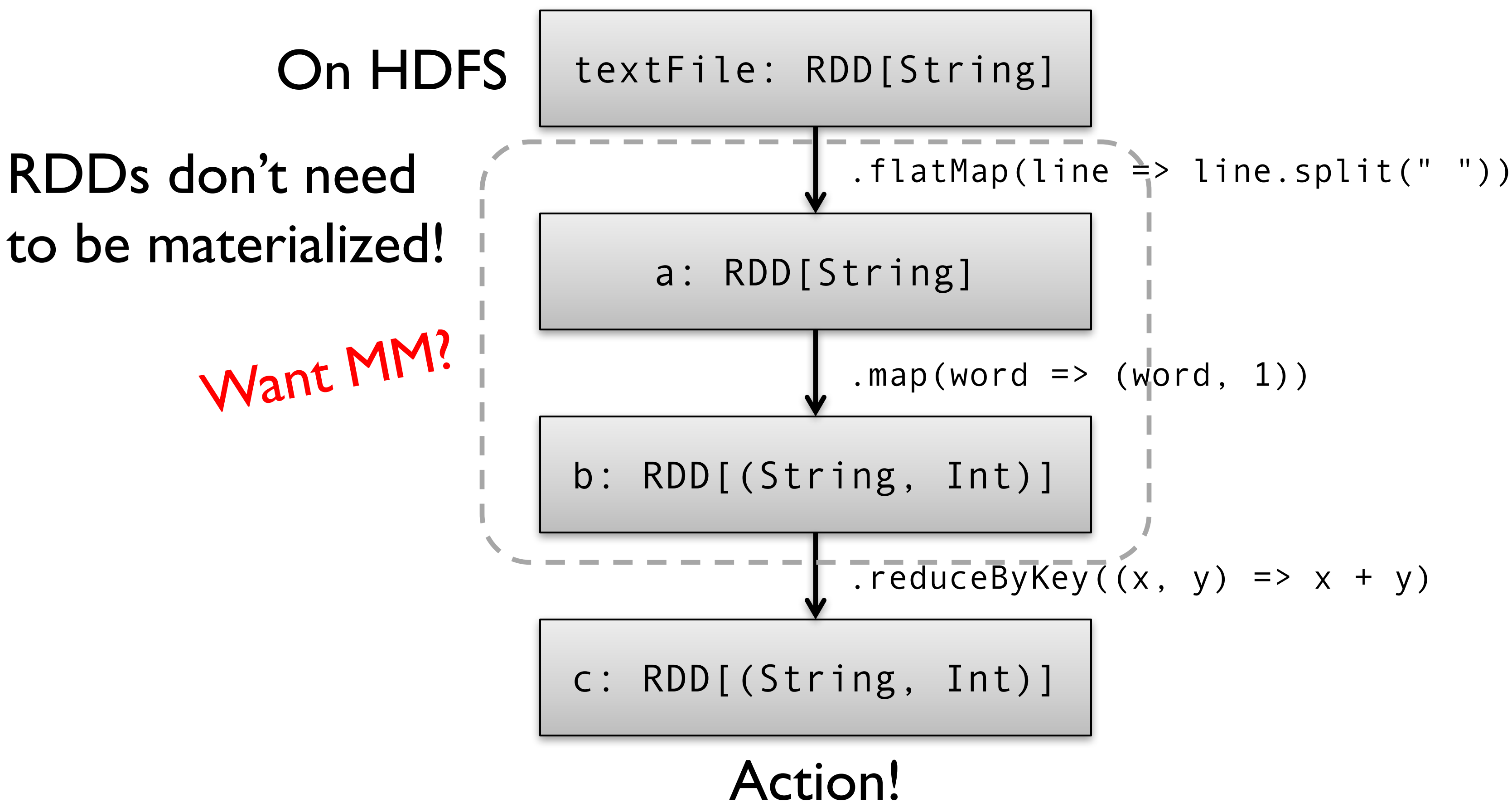
RDDs and Lineage



Remember,
transformations are lazy!

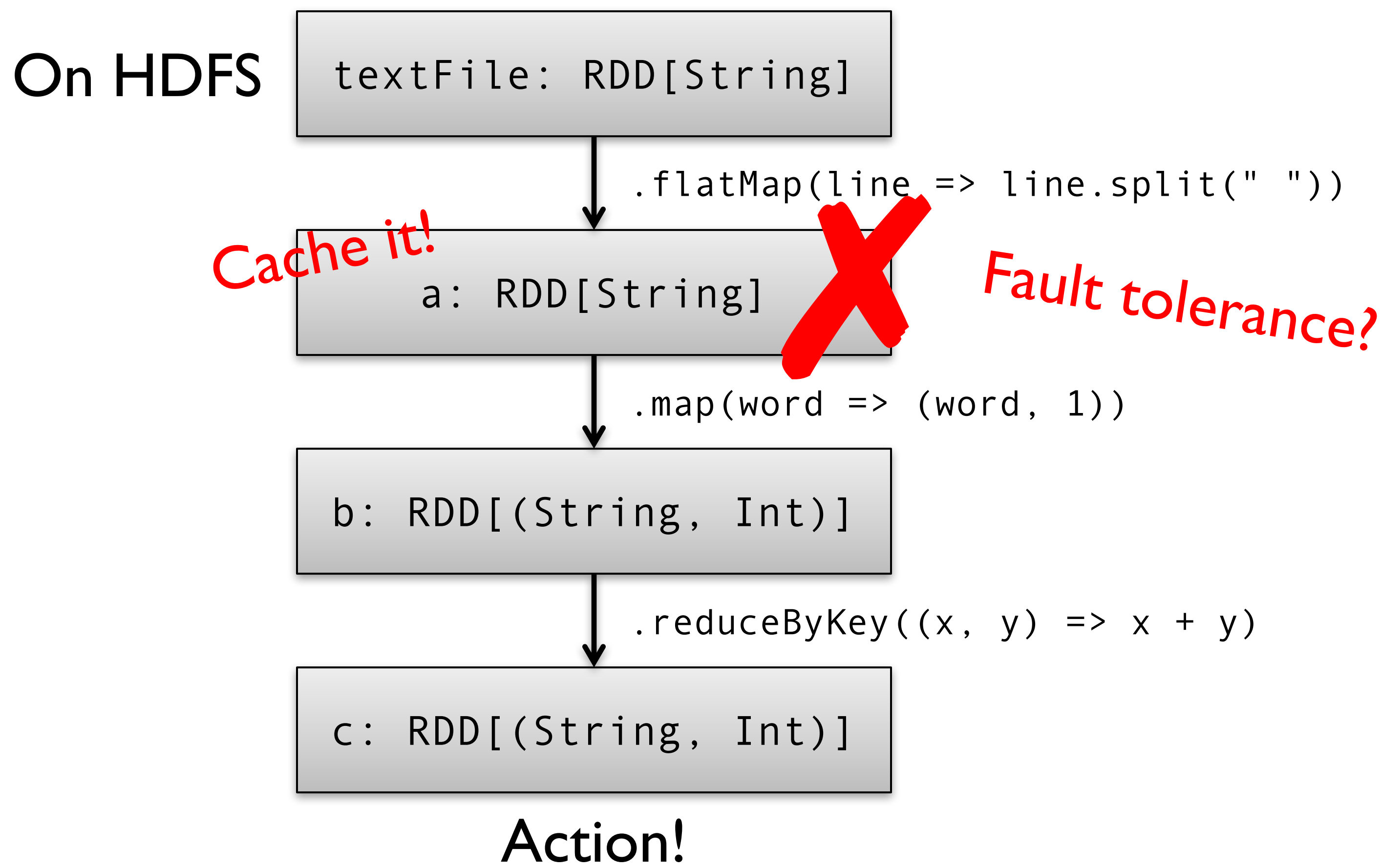
RDDs and Optimizations

Lazy evaluation creates optimization opportunities



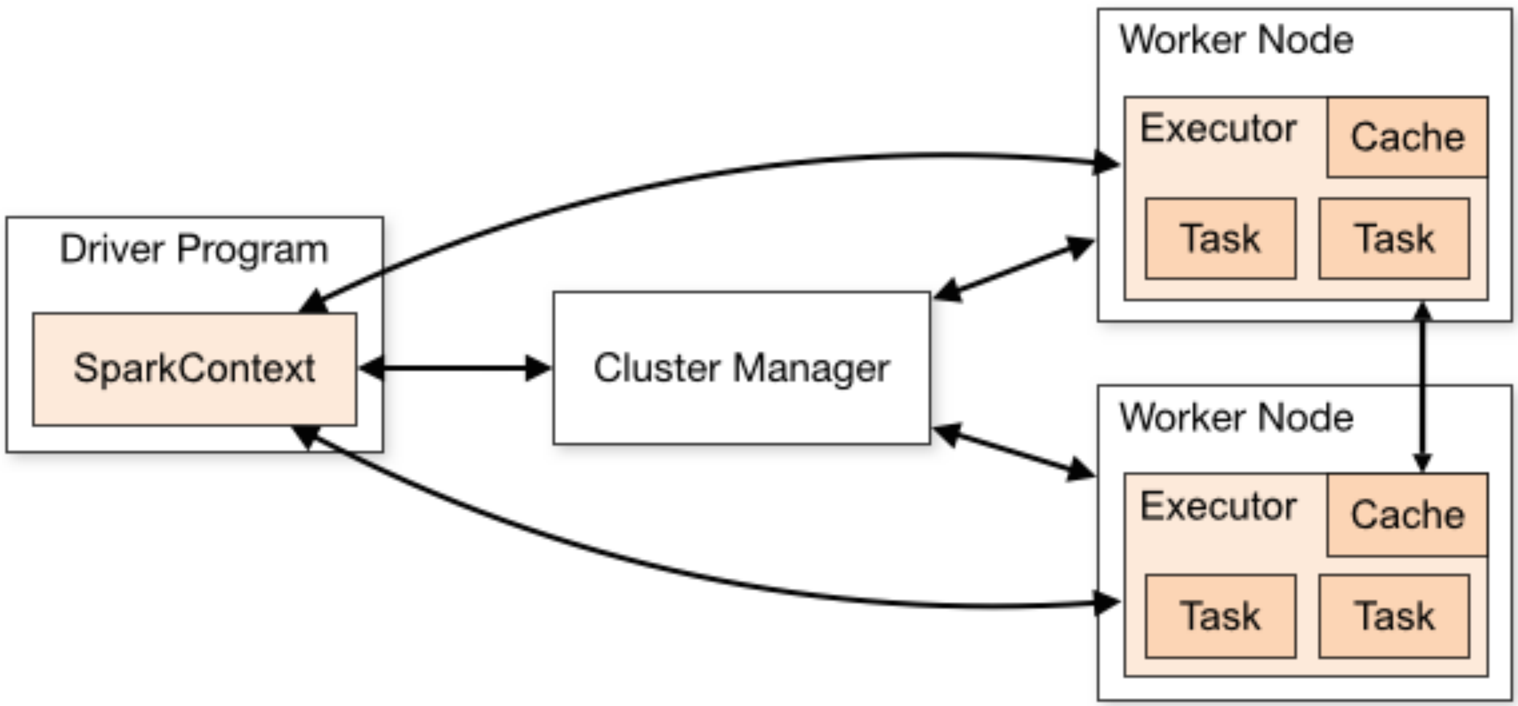
RDDs and Caching

RDDs can be materialized in memory (and on disk)!

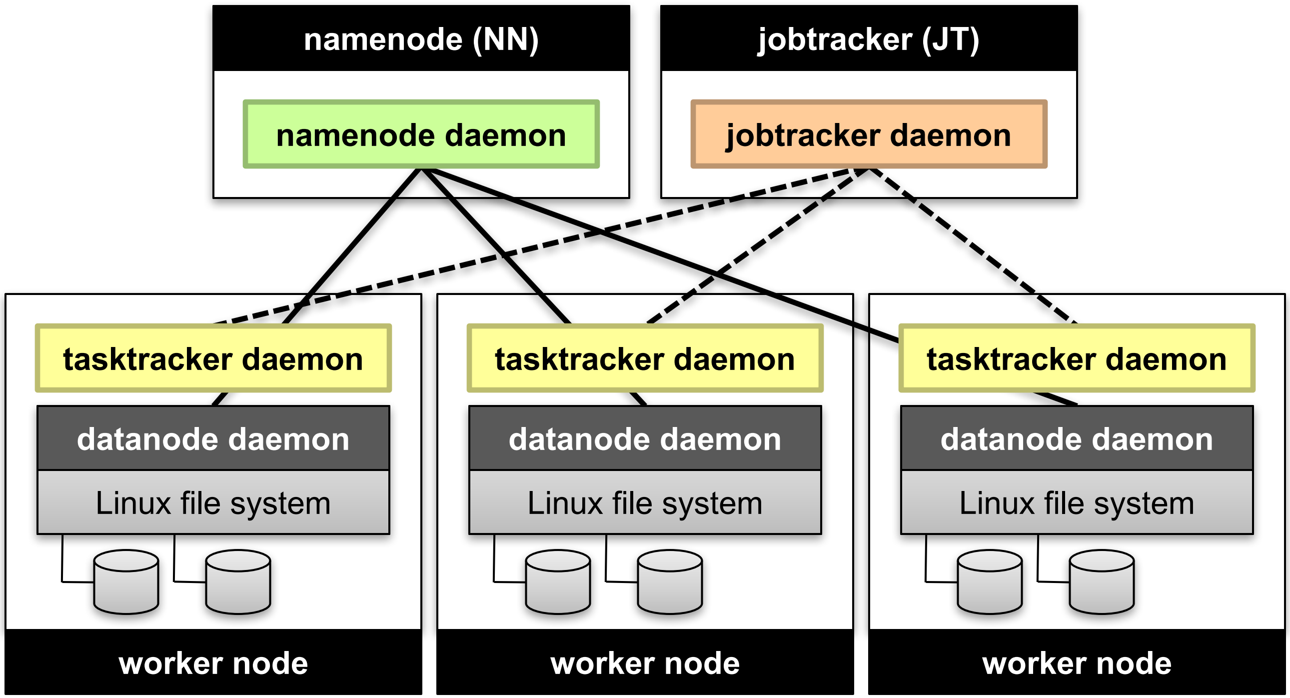


Spark works even if the RDDs are *partially* cached!

Spark Architecture



Hadoop MapReduce Architecture



An Apt Quote

All problems in computer science can be solved by another level of indirection... Except for the problem of too many layers of indirection.

- David Wheeler

YARN

Hadoop's (original) limitations:

Can only run MapReduce

What if we want to run other distributed frameworks?

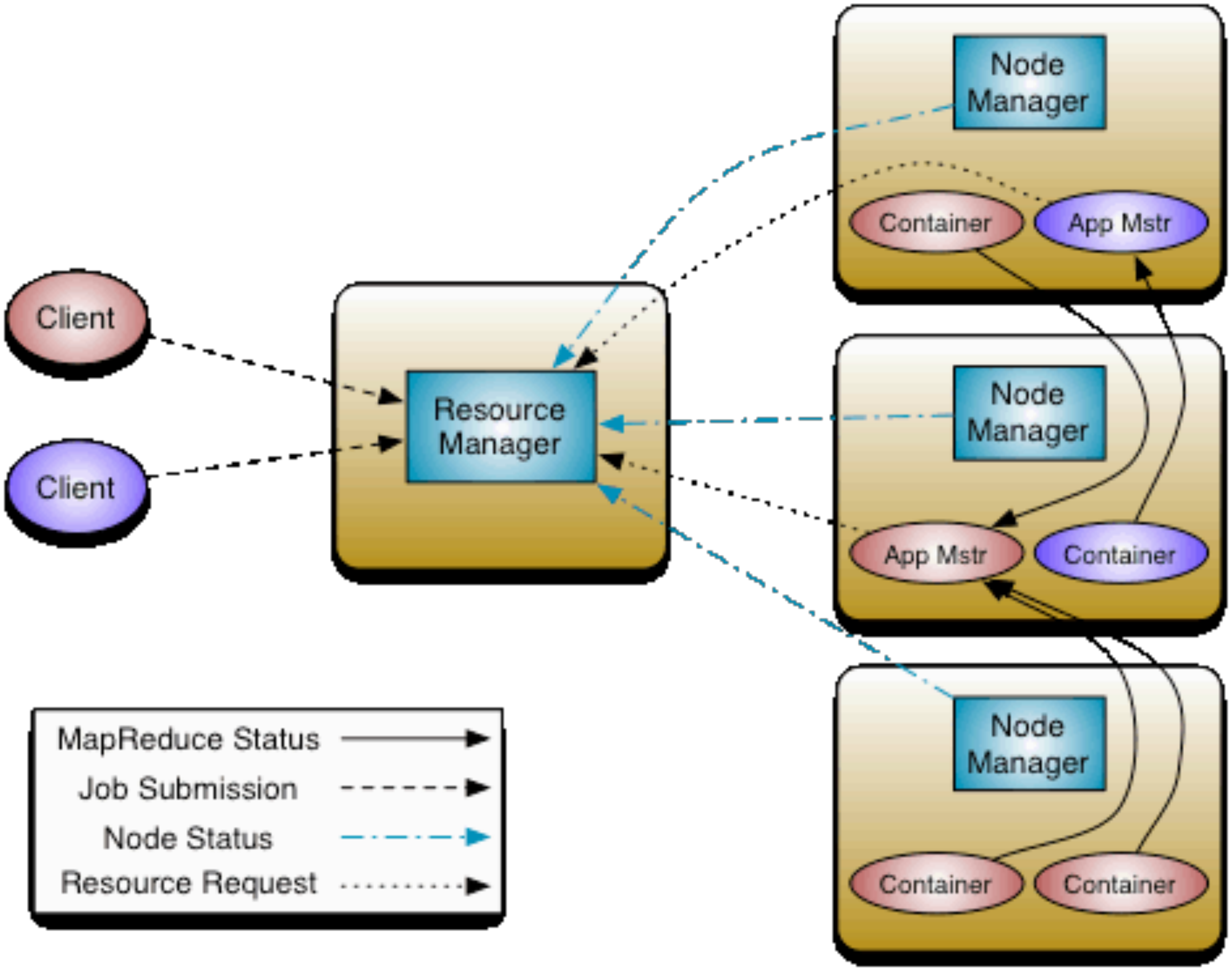
YARN = Yet-Another-Resource-Negotiator

Provides API to develop any generic distributed application

Handles scheduling and resource request

MapReduce (MR2) is one such application in YARN

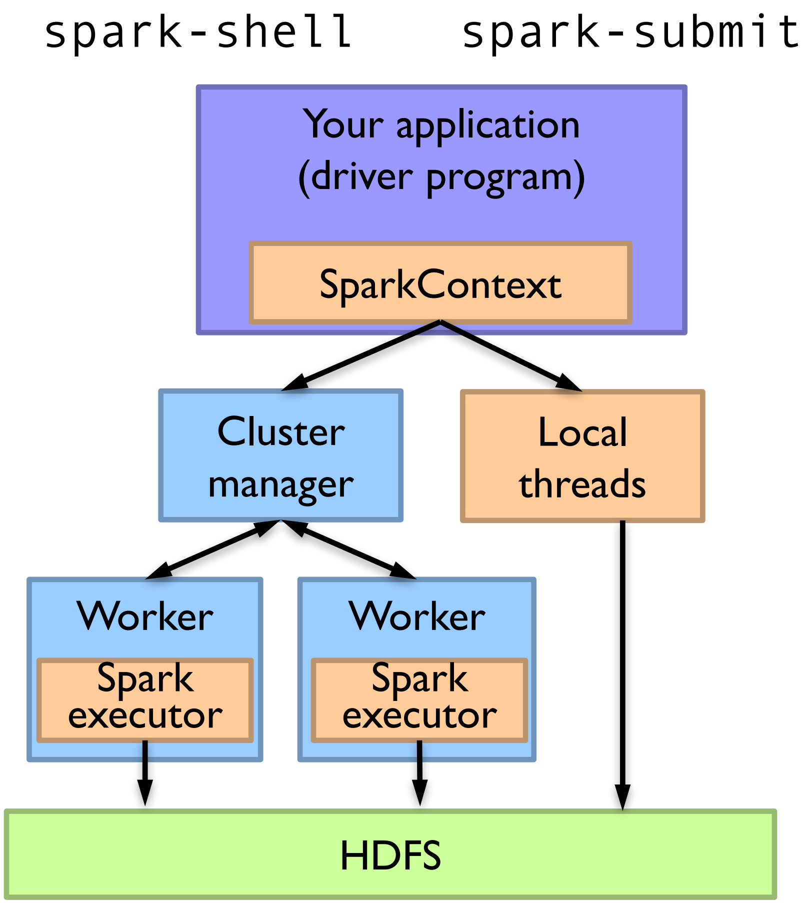
YARN



Client
Client
Resource Manager
Node Manager
Container
App Mstr
Node Manager
App Mstr
Container
Node Manager
Container
Container
MapReduce Status
Job Submission
Node Status
Resource Request

Spark Programs

Scala, Java, Python, R

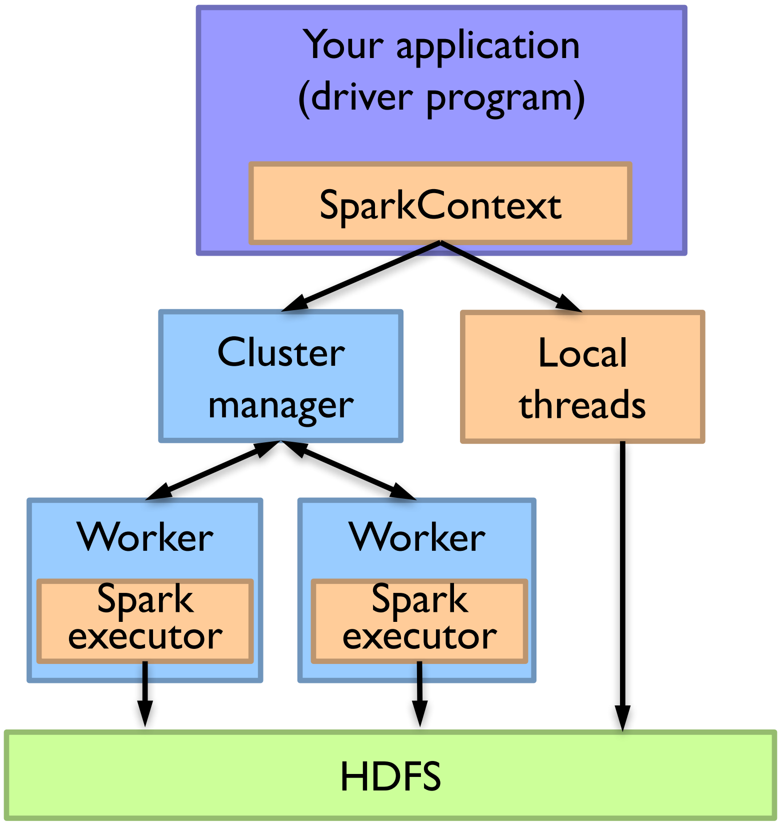


Spark context: tells the framework where to find the cluster

Use the Spark context to create RDDs

Spark Driver

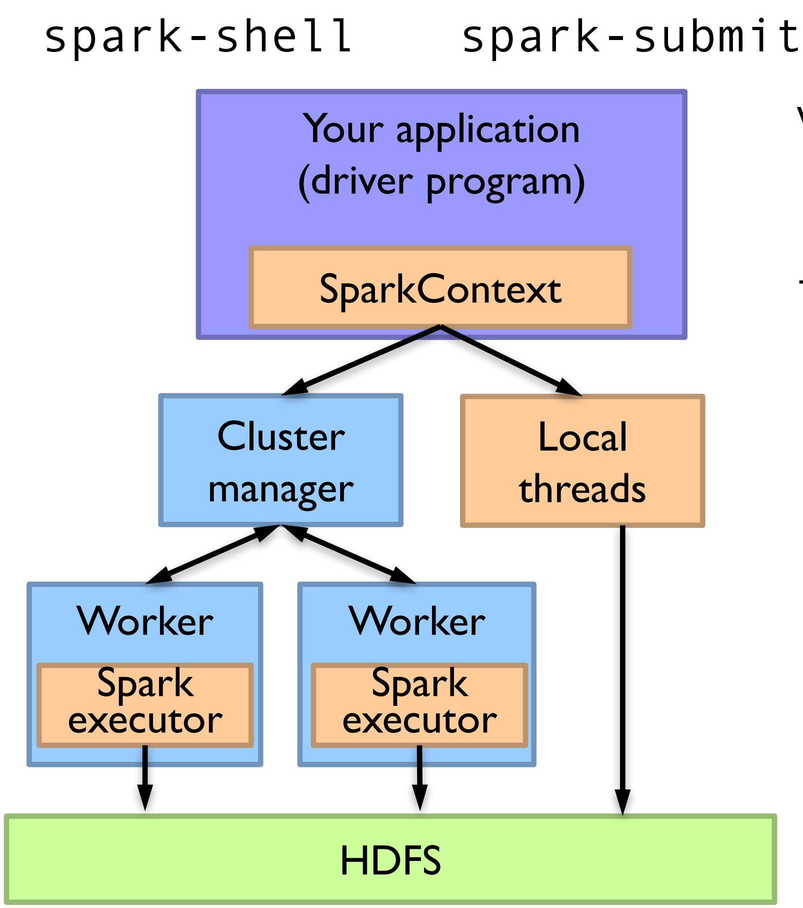
`spark-shell` `spark-submit`




```
val textFile =
  sc.textFile(args.input())

textFile
  .flatMap(line => tokenize(line))
  .map(word => (word, 1))
  .reduceByKey((x, y) => x + y)
  .saveAsTextFile(args.output())
```

What's happening
to the functions?

Spark Driver

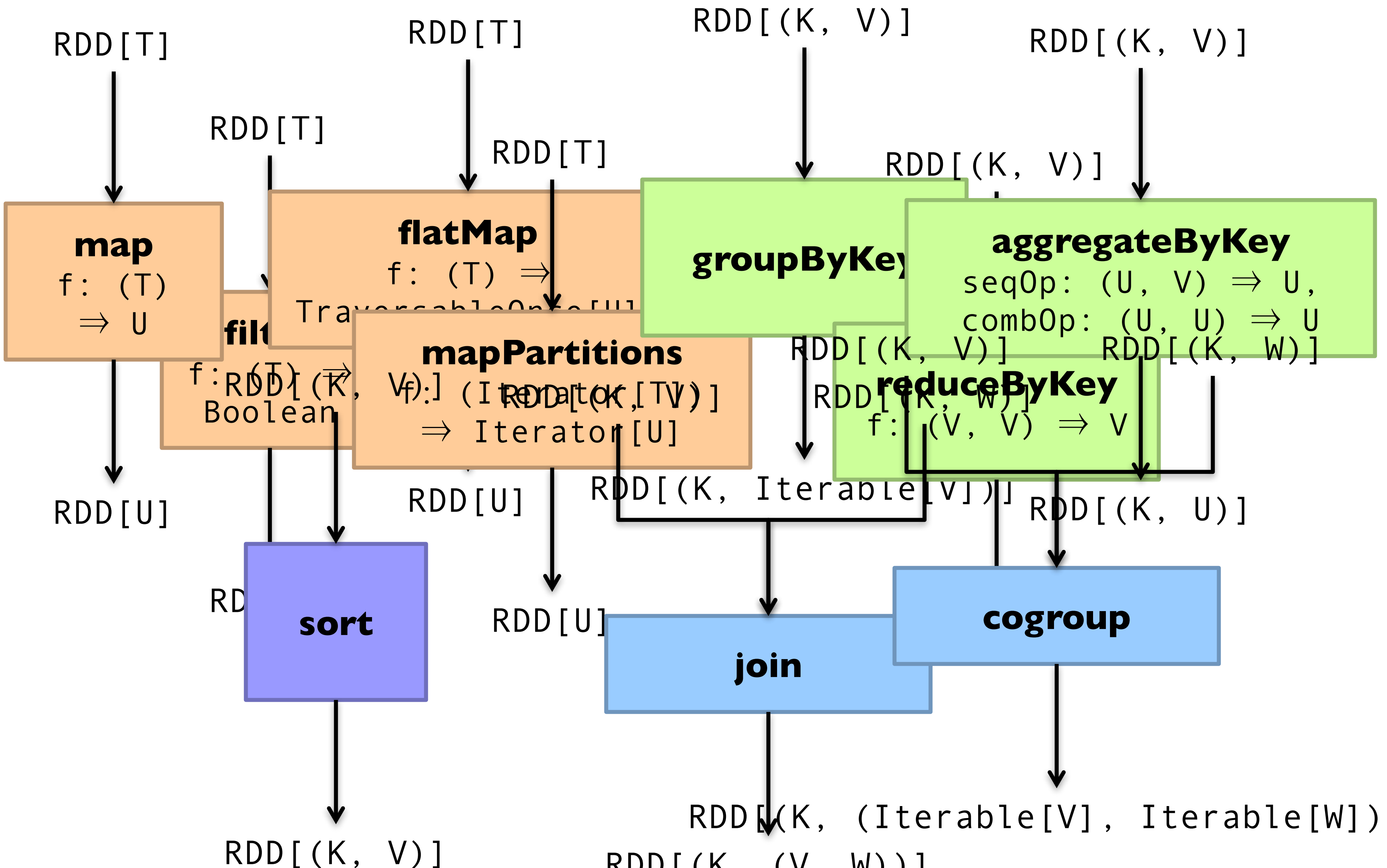



```
val textFile =
  sc.textFile(args.input())

textFile
  .flatMap(line => tokenize(line))
  .map(word => (word, 1))
  .reduceByKey((x, y) => x + y)
  .saveAsTextFile(args.output())
```

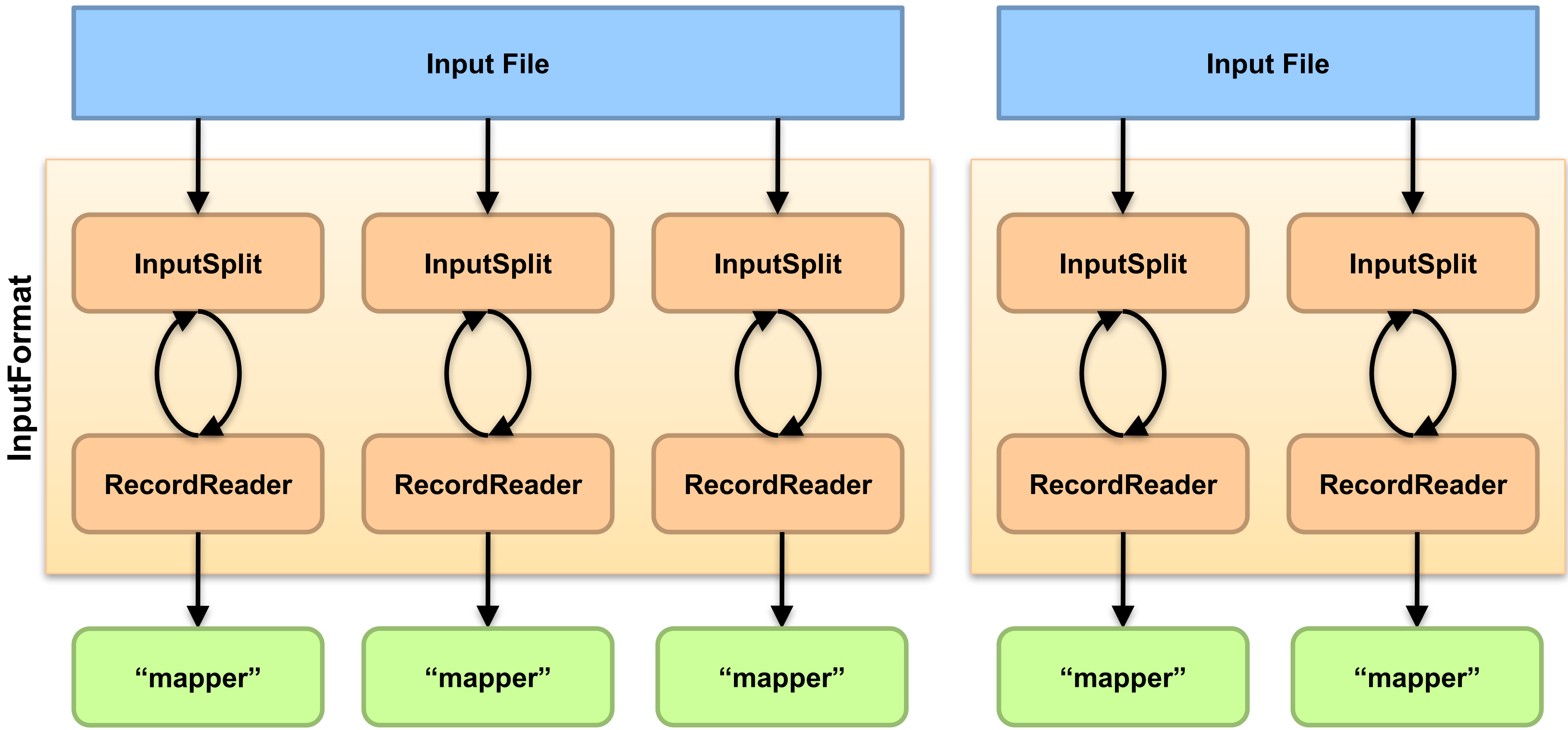
Note: you can run code "locally", integrate cluster-computed values!

Beware of the collect action!

Spark Transformations

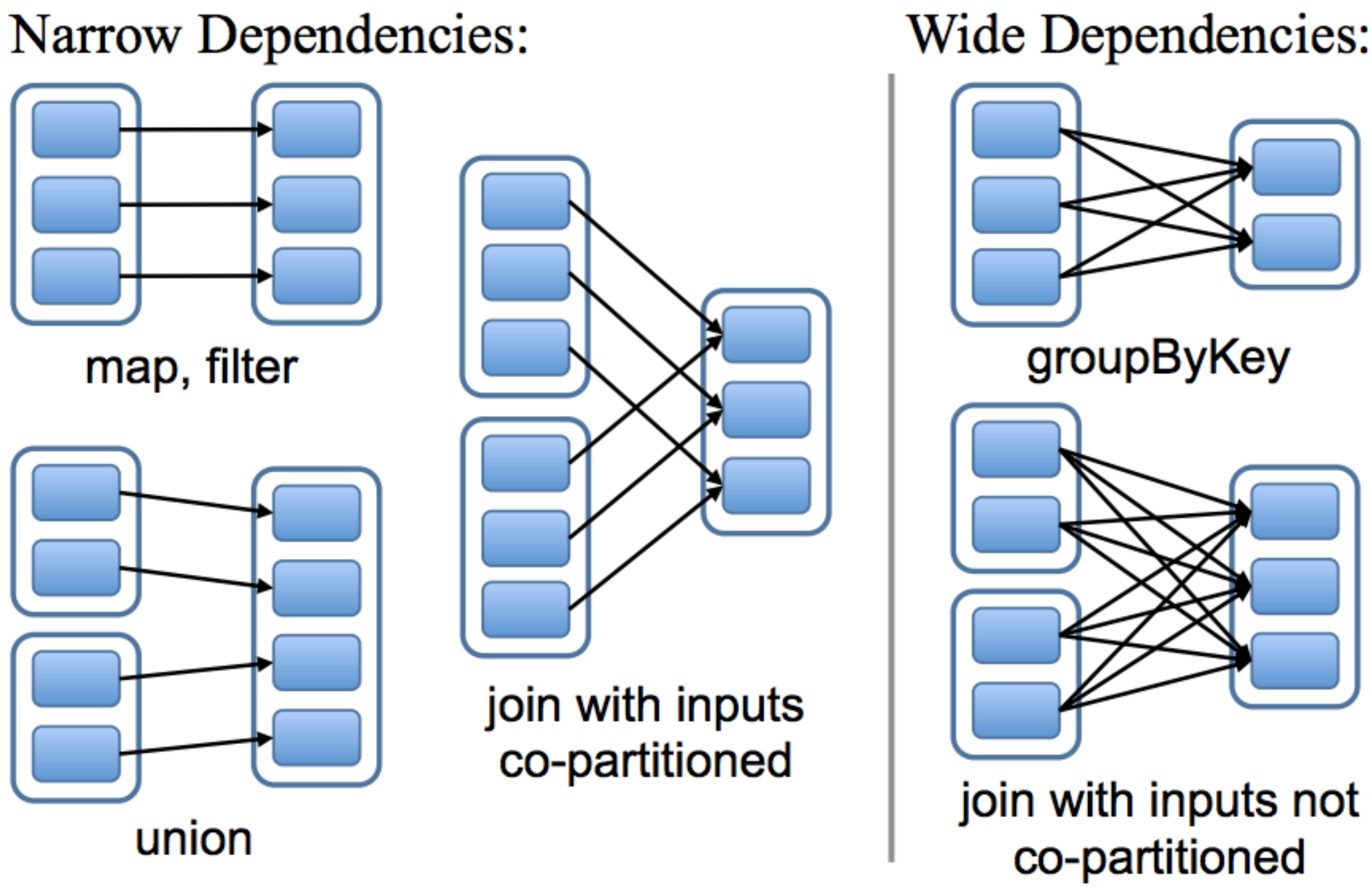


Starting Points

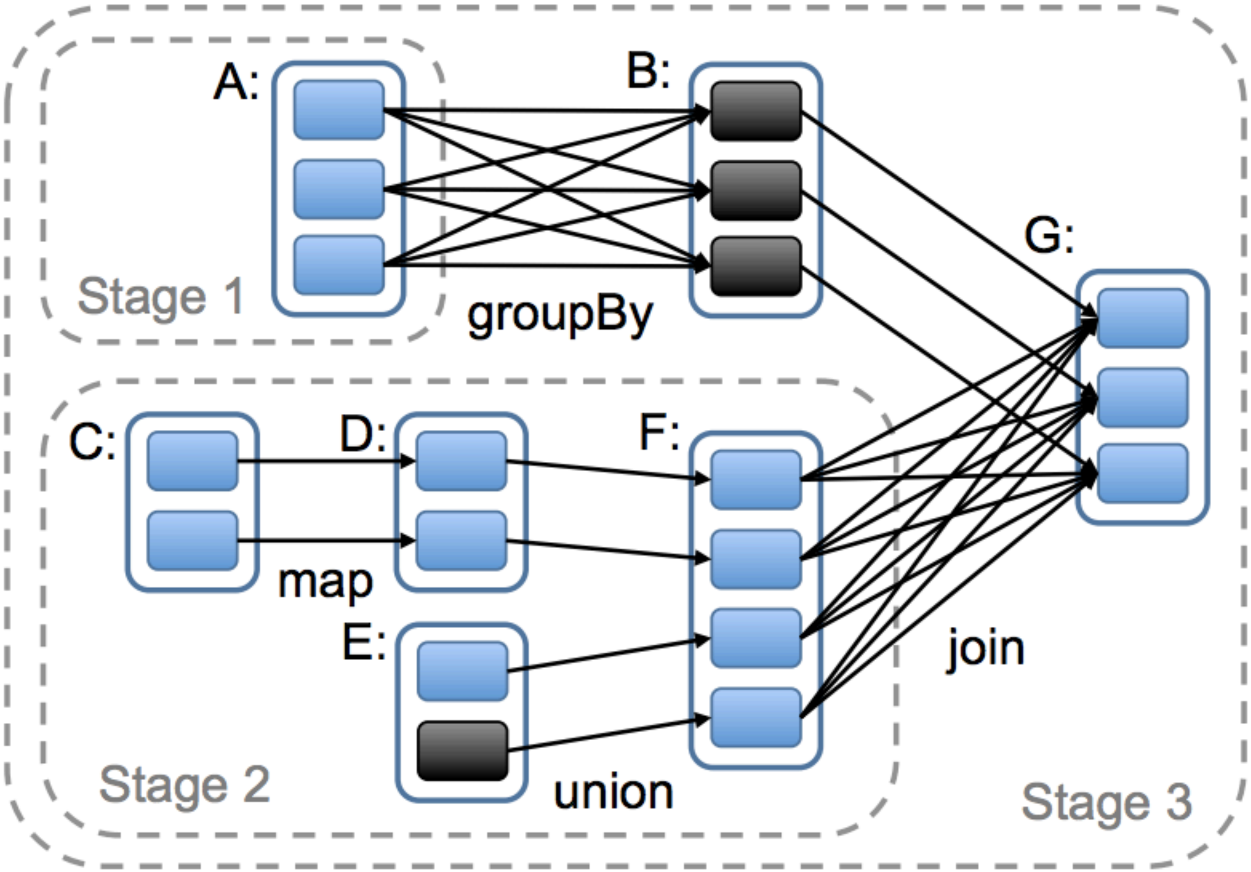


Source: redrawn from a slide by Cloduera, cc-licensed

Physical Operators



Execution Plan



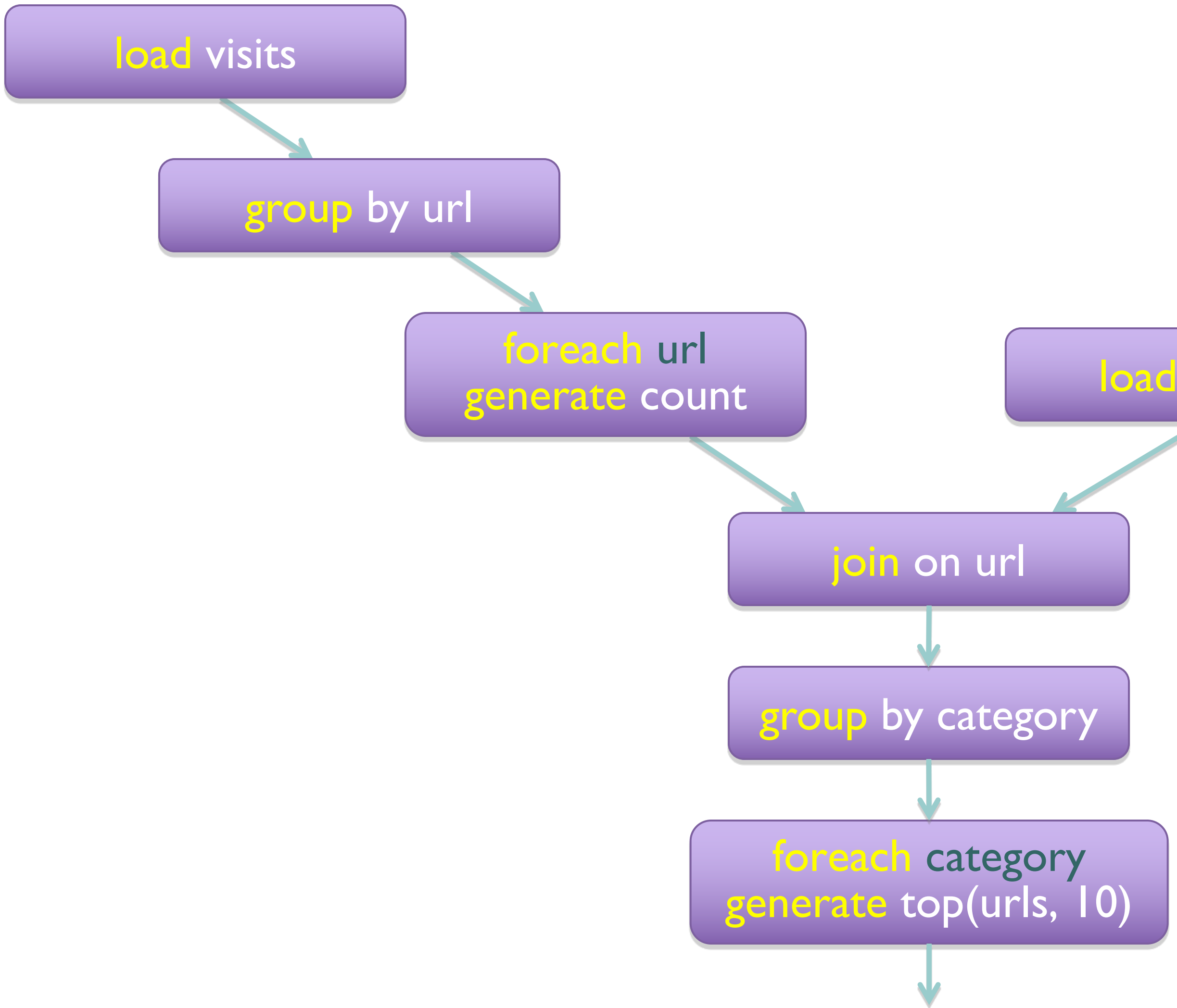
Wait, where have we seen this before?

Pig: Example Script

```
visits = load '/data/visits' as (user, url, time);
gVisits = group visits by url;
visitCounts = foreach gVisits generate url, count(visits);
urlInfo = load '/data/urlInfo' as (url, category, pRank);
visitCounts = join visitCounts by url, urlInfo by url;
gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);

store topUrls into '/data/topUrls';
```

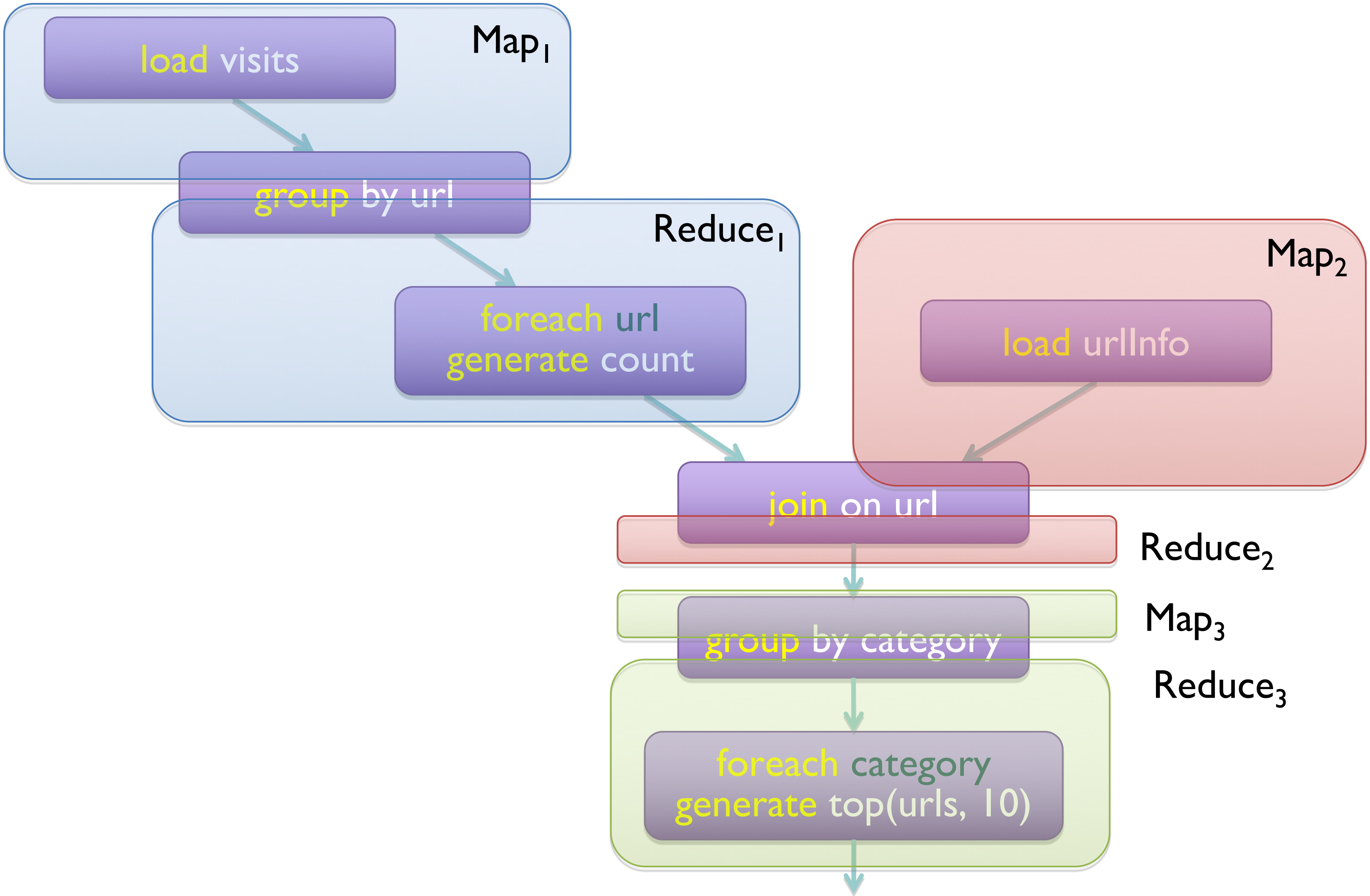
Pig Slides adapted from Olston et al. (SIGMOD 2008)

Pig Query Plan



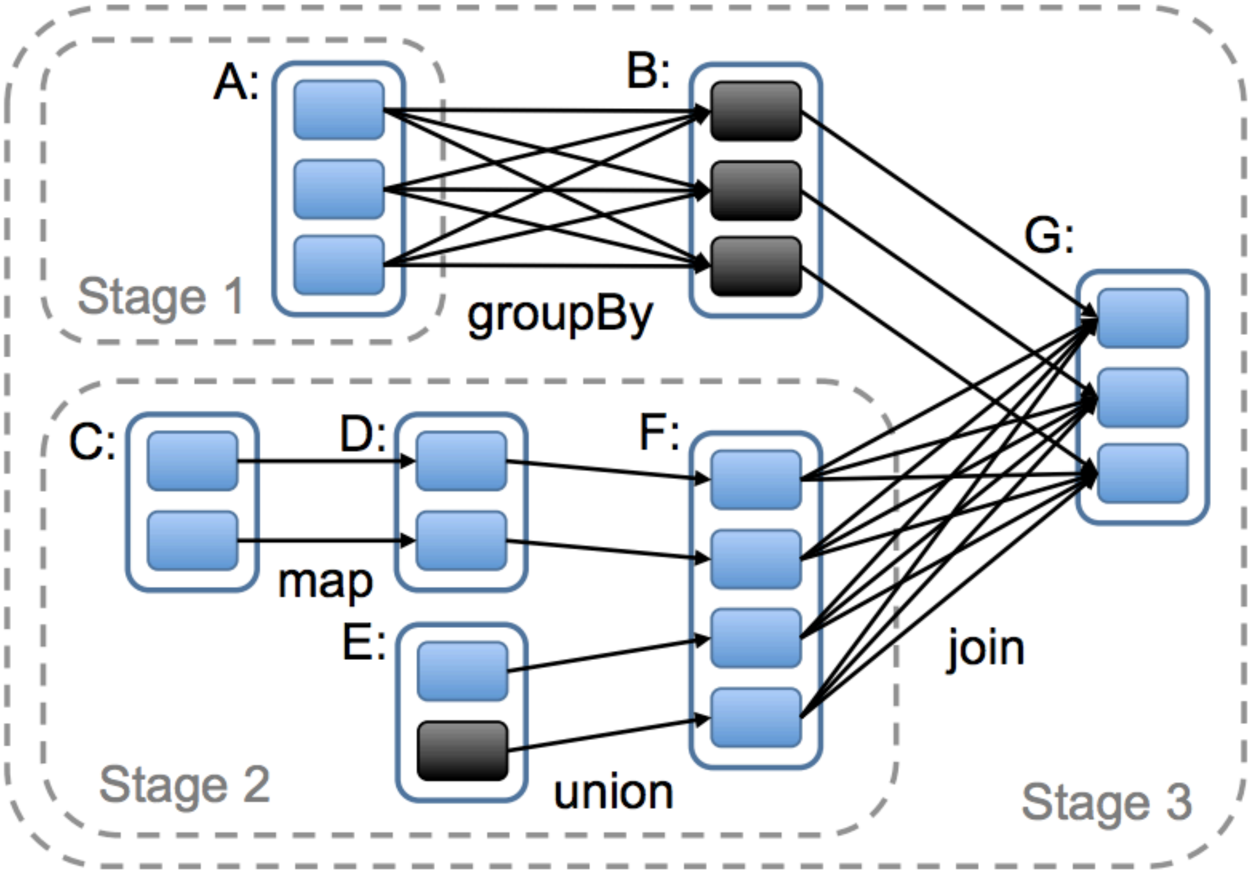
Pig Slides adapted from Olston et al. (SIGMOD 2008)

Pig: MapReduce Execution



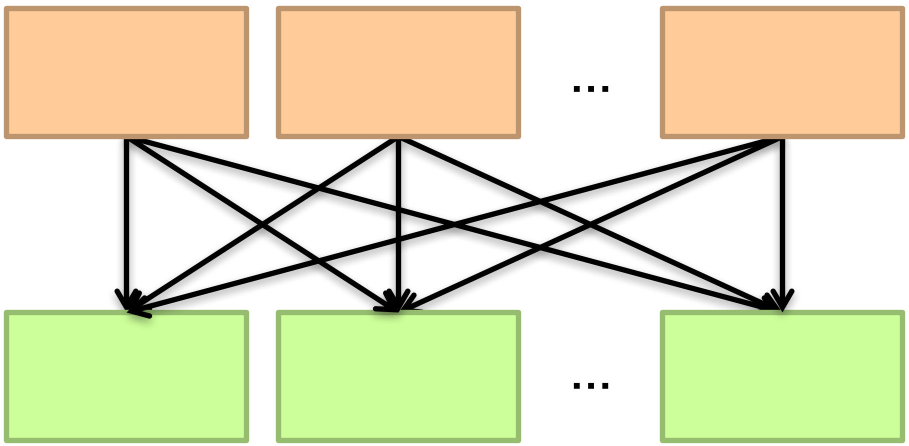
Pig Slides adapted from Olston et al. (SIGMOD 2008)

Execution Plan



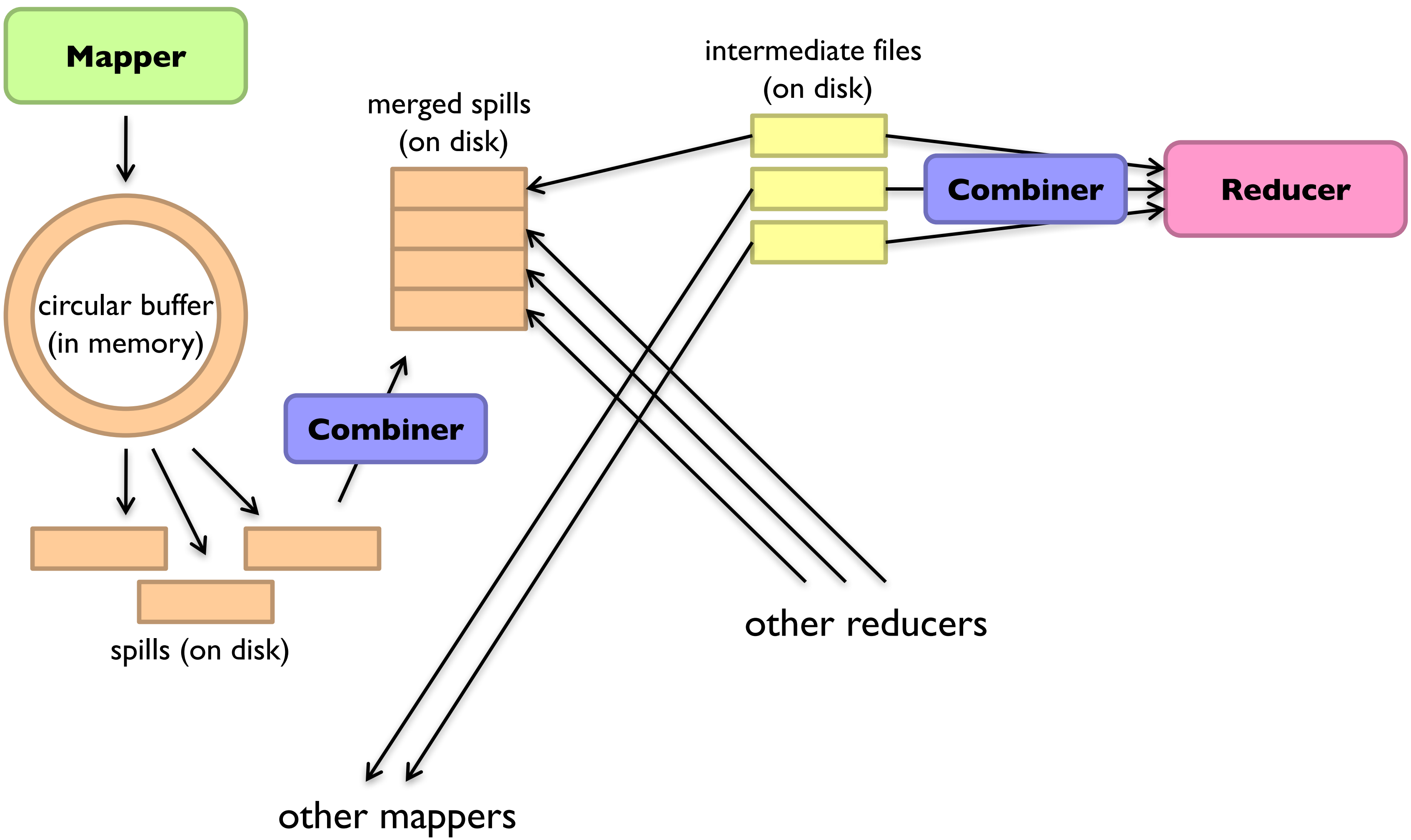
Kinda like a sequence of MapRedue jobs?

Can't avoid this!



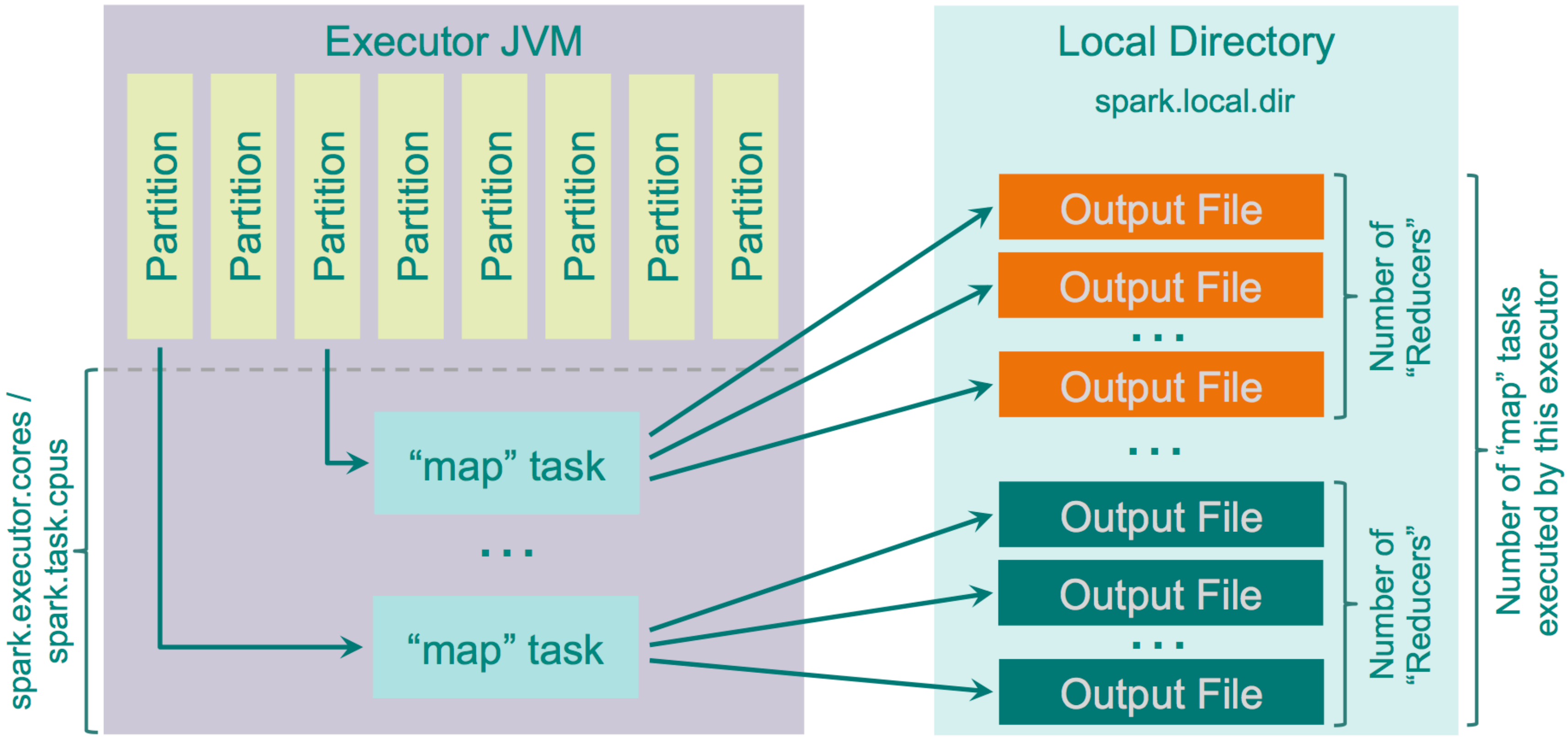
But, what's the major difference?

Remember this?



Spark Shuffle Implementations

Hash shuffle

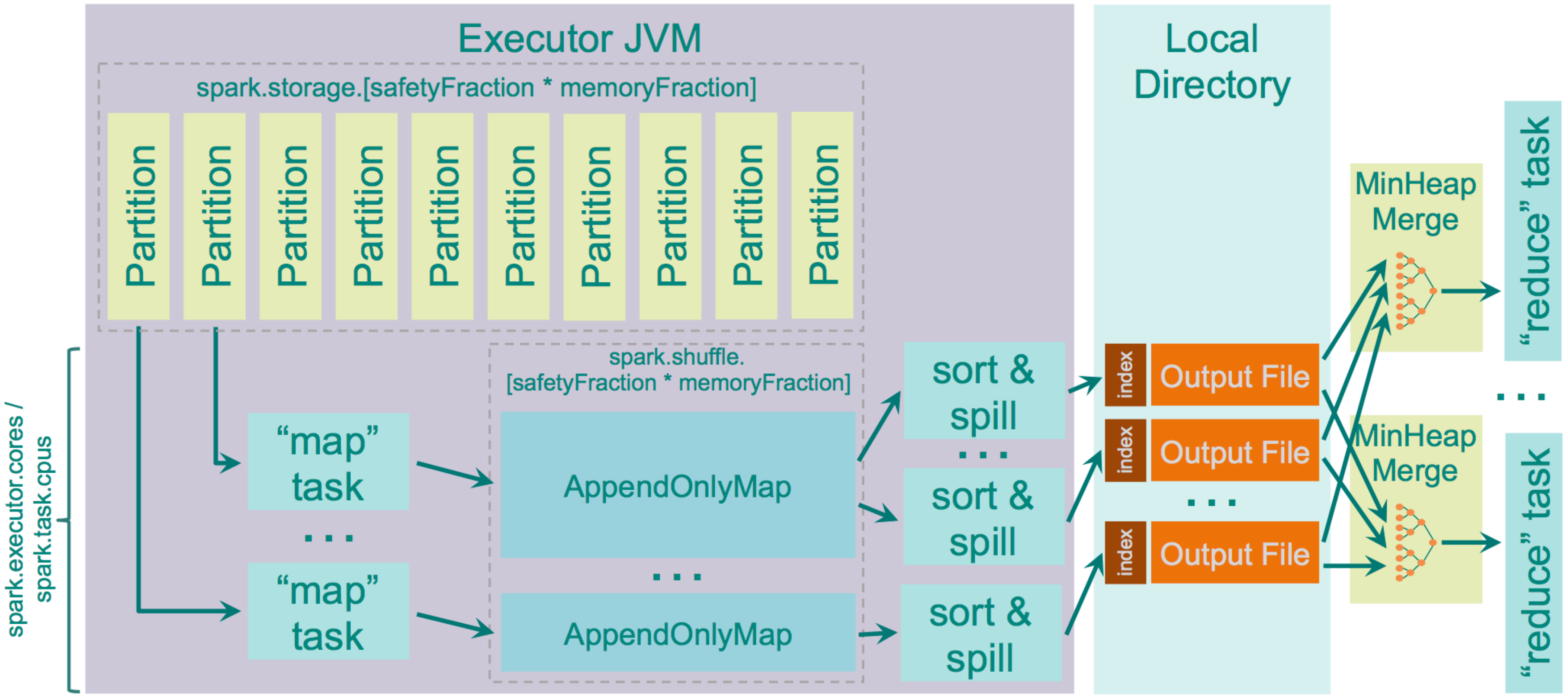


What happened to sorting?

Source: http://0x0fff.com/spark-architecture-shuffle/

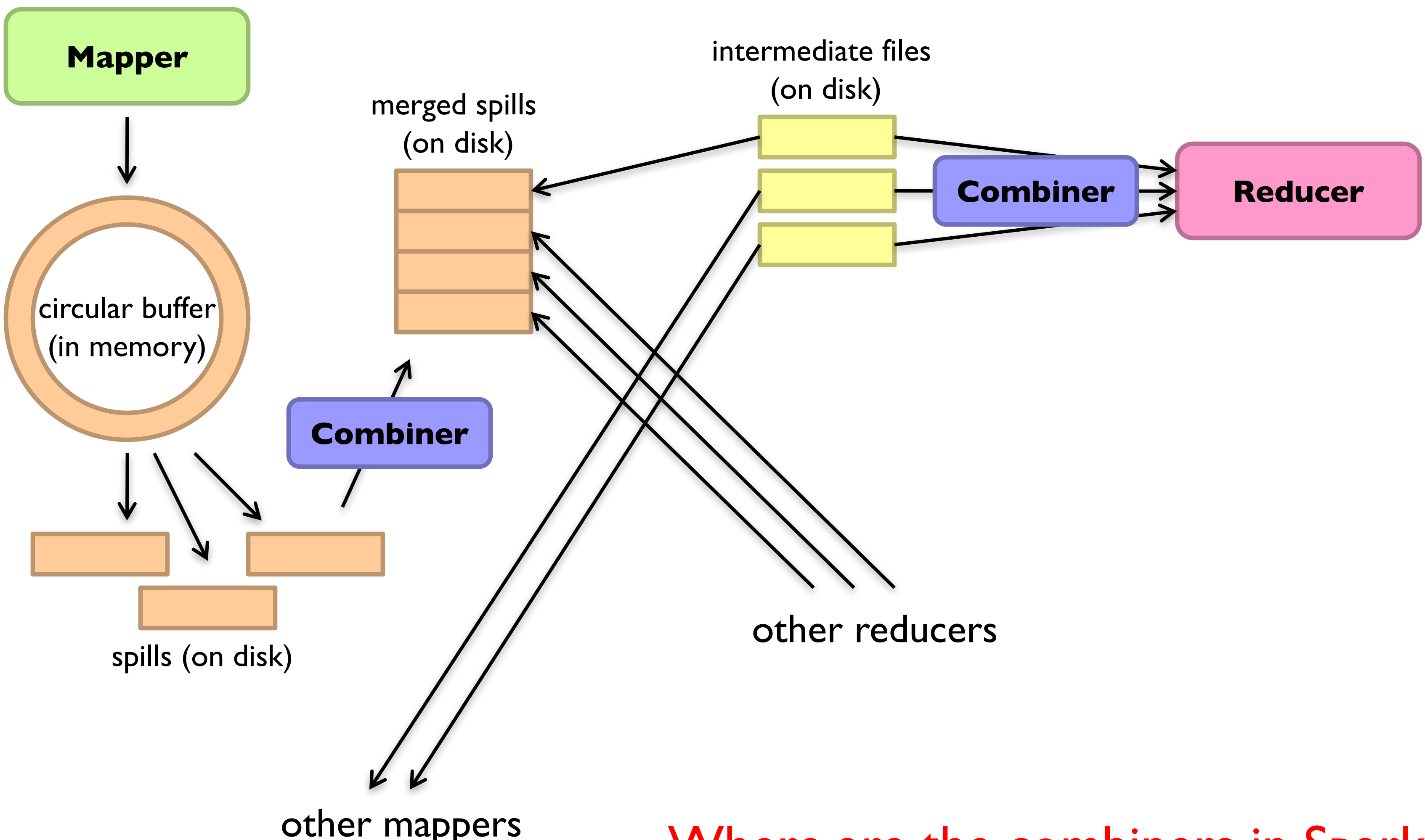
Spark Shuffle Implementations

Sort shuffle



Source: http://0x0fff.com/spark-architecture-shuffle/

Remember this?



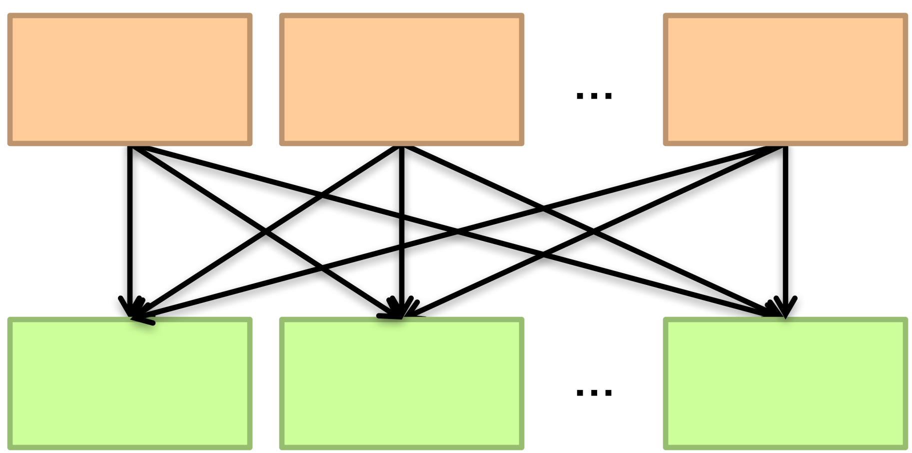
Where are the combiners in Spark?

Reduce-like Operations

`RDD[(K, V)]` ↓ **reduceByKey** `f: (V, V) ⇒ V` ↓ `RDD[(K, V)]`

`RDD[(K, V)]` ↓ **aggregateByKey** `seqOp: (U, V) ⇒ U, combOp: (U, U) ⇒ U` ↓ `RDD[(K, U)]`

What happened to combiners?



Spark #wins

Richer operators

RDD abstraction supports
optimizations (pipelining, caching, etc.)

Scala, Java, Python, R, bindings

Spark #wins

Quick Start Using Scala - Data ×

Jimmy

Secure https://cdn2.hubspot.net/hubfs/438089/notebooks/Quick_Start/Quick_Start_Using_Scala.html

databricks — Quick Start Using Scala (Scala) — Import Notebook

```
// Take a look at the file system
display(dbutils.fs.ls("/databricks-datasets/samples/docs/"))
```

path	name	size
dbfs:/databricks-datasets/samples/docs/README.md	README.md	3137

```
// Setup the textFile RDD to read the README.md file
//   Note this is lazy
val textFile = sc.textFile("/databricks-datasets/samples/docs/README.md")
```

```
textFile: org.apache.spark.rdd.RDD[String] = /databricks-datasets/samples/docs/README.md MapPartitionsRDD[1873222] at textFile at <console>:34
```

RDDs have ***actions***, which return values, and ***transformations***, which return pointers to new RDDs.

```
// When performing an action (like a count) this is when the textFile is read and aggregate calculated
//    Click on [View] to see the stages and executors
textFile.count()
```

```
res2: Long = 65
```

Scala Count (Jobs)

Jobs Stages Storage Environment Executors SQL JDBC/ODBC Server

Spark #lose

Java serialization (w/ Kryo optmizations)
Scala: poor support for primitives

Algorithm design, redux

Source: Wikipedia (Mahout)

Two superpowers:

Associativity
Commutativity
(sorting)

What follows… very basic category theory…

The Power of Associativity

You can put parentheses where ever you want!

$$(v_1 \oplus v_2 \oplus v_3) \oplus (v_4 \oplus v_5 \oplus v_6 \oplus v_7) \oplus (v_8 \oplus v_9)$$

$$(v_1 \oplus v_2) \oplus (v_3 \oplus v_4 \oplus v_5) \oplus (v_6 \oplus v_7 \oplus v_8 \oplus v_9)$$

$$(v_1 \oplus v_2 \oplus (v_3 \oplus v_4 \oplus v_5)) \oplus (v_6 \oplus v_7 \oplus v_8 \oplus v_9)$$

Credit to Oscar Boykin for the idea behind these slides

The Power of Commutativity

You can swap order of operands however you want!

$$\left(v_1 \oplus v_2 \oplus v_3\right) \oplus \left(v_4 \oplus v_5 \oplus v_6 \oplus v_7\right) \oplus \left(v_8 \oplus v_9\right)$$

$$\left(v_4 \oplus v_5 \oplus v_6 \oplus v_7\right) \oplus \left(v_1 \oplus v_2 \oplus v_3\right) \oplus \left(v_8 \oplus v_9\right)$$

$$\left(v_8 \oplus v_9\right) \oplus \left(v_4 \oplus v_5 \oplus v_6 \oplus v_7\right) \oplus \left(v_1 \oplus v_2 \oplus v_3\right)$$

Implications for distributed processing?

You don’t know when the tasks begin
You don’t know when the tasks end
You don’t know when the tasks interrupt each other
You don’t know when intermediate data arrive
…

It’s okay!

Word Count: Baseline

```
class Mapper {
  def map(key: Long, value: String) = {
    for (word <- tokenize(value)) {
      emit(word, 1)
    }
  }
}

class Reducer {
  def reduce(key: String, values: Iterable[Int]) = {
    for (value <- values) {
      sum += value
    }
    emit(key, sum)
  }
}
```

Fancy Labels for Simple Concepts…

Semigroup = (M , $\oplus$)

$$\oplus : M \times M \rightarrow M, \text{ s.t., } \forall m_1, m_2, m_3 \ni M$$

$$(m_1 \oplus m_2) \oplus m_3 = m_1 \oplus (m_2 \oplus m_3)$$

Monoid = Semigroup + identity

$$\varepsilon \text{ s.t., } \varepsilon \oplus m = m \oplus \varepsilon = m, \ \forall m \ni M$$

Commutative Monoid = Monoid + commutativity

$$\forall m_1, m_2 \ni M, m_1 \oplus m_2 = m_2 \oplus m_1$$

A few examples?
(hint, previous slide!)

Back to these…

RDD[(K, V)]

↓

reduceByKey

f: (V, V) ⇒ V

↓

RDD[(K, V)]

RDD[(K, V)]

↓

aggregateByKey

seqOp: (U, V) ⇒ U,

combOp: (U, U) ⇒ U

↓

RDD[(K, U)]

Wait, I've seen this before?

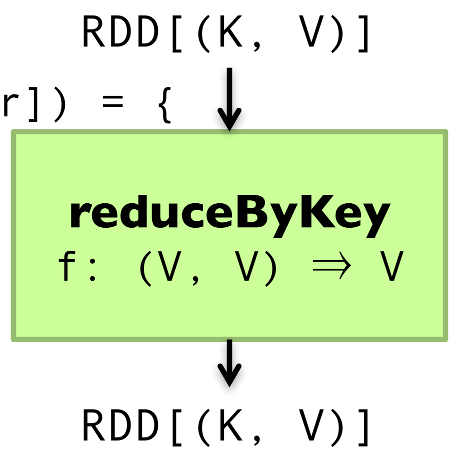
Computing the Mean: Version 1

```
class Mapper {
  def map(key: String, value: Int) = {
    emit(key, value)
  }
}

class Reducer {
  def reduce(key: String, values: Iterable[Int]) {
    for (value <- values) {
      sum += value
      cnt += 1
    }
    emit(key, sum/cnt)
  }
}
```

Computing the Mean: Version 3

```
class Mapper {
  def map(key: String, value: Int) =
    context.write(key, (value, 1))
}
class Combiner {
  def reduce(key: String, values: Iterable[Pair]) = {
    for ((s, c) <- values) {
      sum += s
      cnt += c
    }
    emit(key, (sum, cnt))
  }
}
class Reducer {
  def reduce(key: String, values: Iterable[Pair]) = {
    for ((s, c) <- values) {
      sum += s
      cnt += c
    }
    emit(key, sum/cnt)
  }
}
```

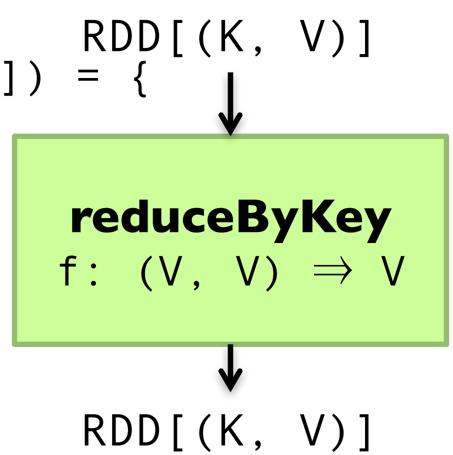
Wait, I've seen this before?



Co-occurrence Matrix: Stripes

```
class Mapper {
  def map(key: Long, value: String) = {
    for (u <- tokenize(value)) {
      val map = new Map()
      for (v <- neighbors(u)) {
        map(v) += 1
      }
      emit(u, map)
    }
  }
}

class Reducer {
  def reduce(key: String, values: Iterable[Map]) = {
    val map = new Map()
    for (value <- values) {
      map += value
    }
    emit(key, map)
  }
}
```

Wait, I've seen this before?



Computing the Mean: Version 2

```
class Mapper {
  def map(key: String, value: Int) =
    context.write(key, value)
}
class Combiner {
  def reduce(key: String, values: Iterable[Int]) = {
    for (value <- values) {
      sum += value
      cnt += 1
    }
    emit(key, (sum, cnt))
  }
}
class Reducer {
  def reduce(key: String, values: Iterable[Pair]) = {
    for ((s, c) <- values) {
      sum += s
      cnt += c
    }
    emit(key, sum/cnt)
  }
}
```

RDD[(K, V)]

aggregateByKey
seqOp: (U, V) ⇒ U,
combOp: (U, U) ⇒ U

RDD[(K, U)]

Synchronization: Pairs vs. Stripes

Approach 1: turn synchronization into an ordering problem

Sort keys into correct order of computation

Partition key space so each reducer receives appropriate set of partial results

Hold state in reducer across multiple key-value pairs to perform computation

Illustrated by the "pairs" approach

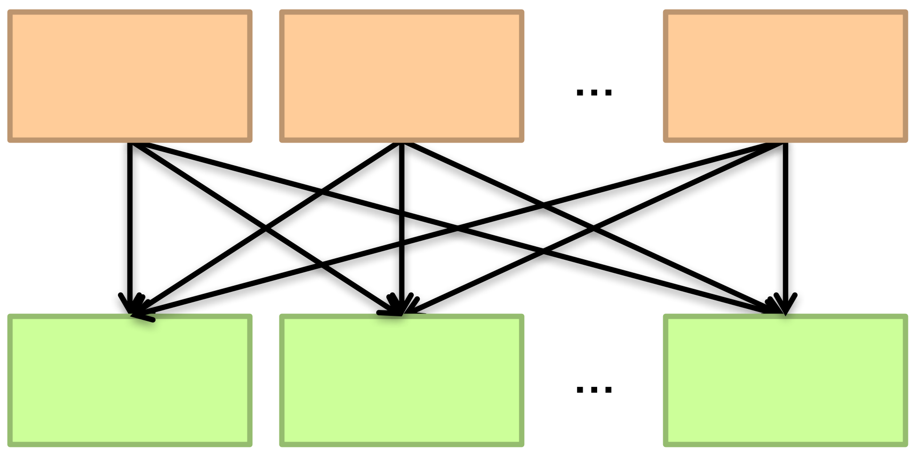
Approach 2: data structures that bring partial results together

Each reducer receives all the data it needs to complete the computation

Illustrated by the "stripes" approach

Commutative monoids!

Because you can't avoid this…



But commutative monoids help

Synchronization: Pairs vs. Stripes

Approach 1: turn synchronization into an ordering problem

Sort keys into correct order of computation

Partition key space so each reducer receives appropriate set of partial results

Hold state in reducer across multiple key-value pairs to perform computation

Illustrated by the "pairs" approach

What about this?

Approach 2: data structures that bring partial results together

Each reducer receives all the data it needs to complete the computation

Illustrated by the "stripes" approach

Commutative monoids!

f(B|A): "Pairs"

(a, *) → 32 Reducer holds this value in memory

(a, b_1) → 3	⇒	(a, b_1) → 3 / 32
(a, b_2) → 12		(a, b_2) → 12 / 32
(a, b_3) → 7		(a, b_3) → 7 / 32
(a, b_4) → 1		(a, b_4) → 1 / 32
…		…

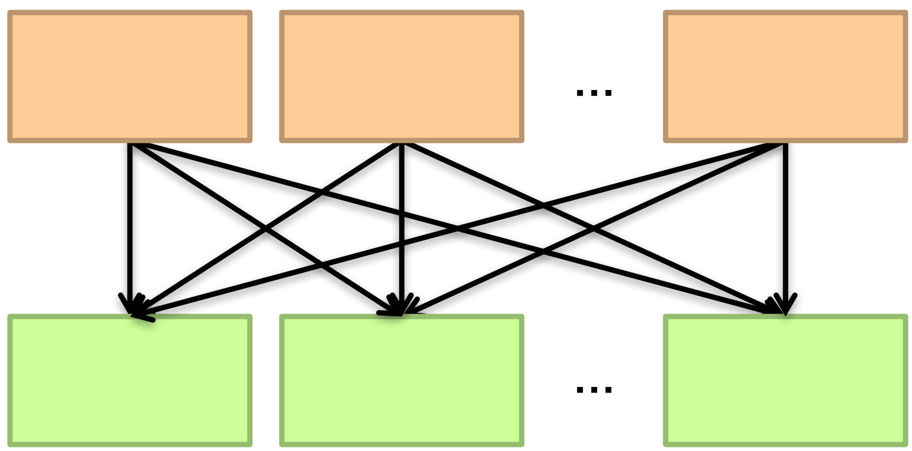
For this to work:

- Emit extra (a, *) for every b_n in mapper
- Make sure all a's get sent to same reducer (use partitioner)
- Make sure (a, *) comes first (define sort order)
- Hold state in reducer across different key-value pairs

Two superpowers:

Associativity
Commutativity
(sorting)

When you can't "monoidify"



Sequence your computations by sorting

An Apt Quote

All problems in computer science can be solved by another level of indirection... Except for the problem of too many layers of indirection.

- David Wheeler


The datacenter *is* the computer!
What's the instruction set?
What are the abstractions?
Source: Google

Algorithm design in a nutshell…

Exploit associativity and commutativity via commutative monoids (if you can)

Exploit framework-based sorting to sequence computations (if you can't)

Source: Wikipedia (Walnut)

Source: Wikipedia (Japanese rock garden)