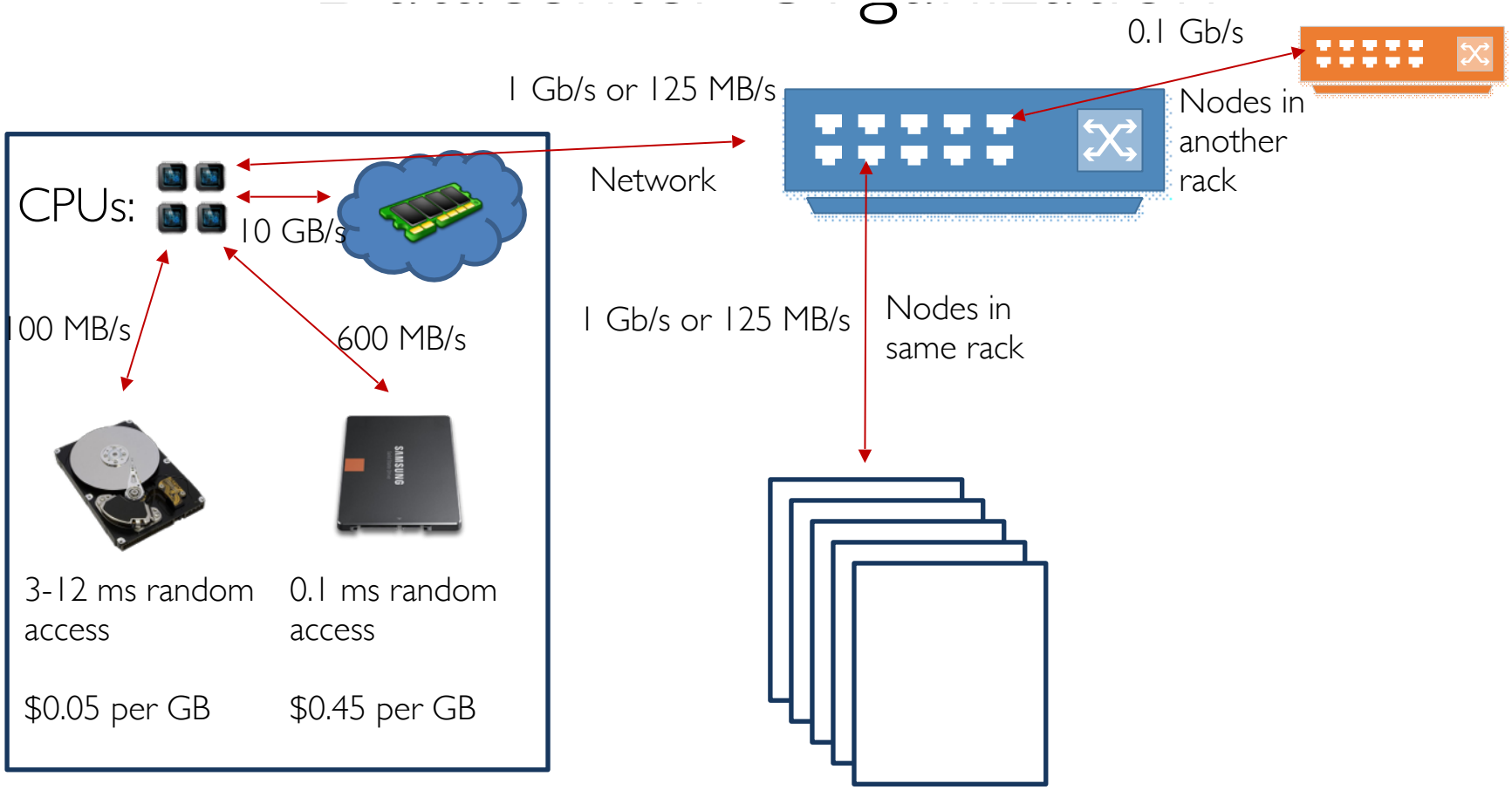
In-Memory Processing with Apache Spark

Vincent Leroy

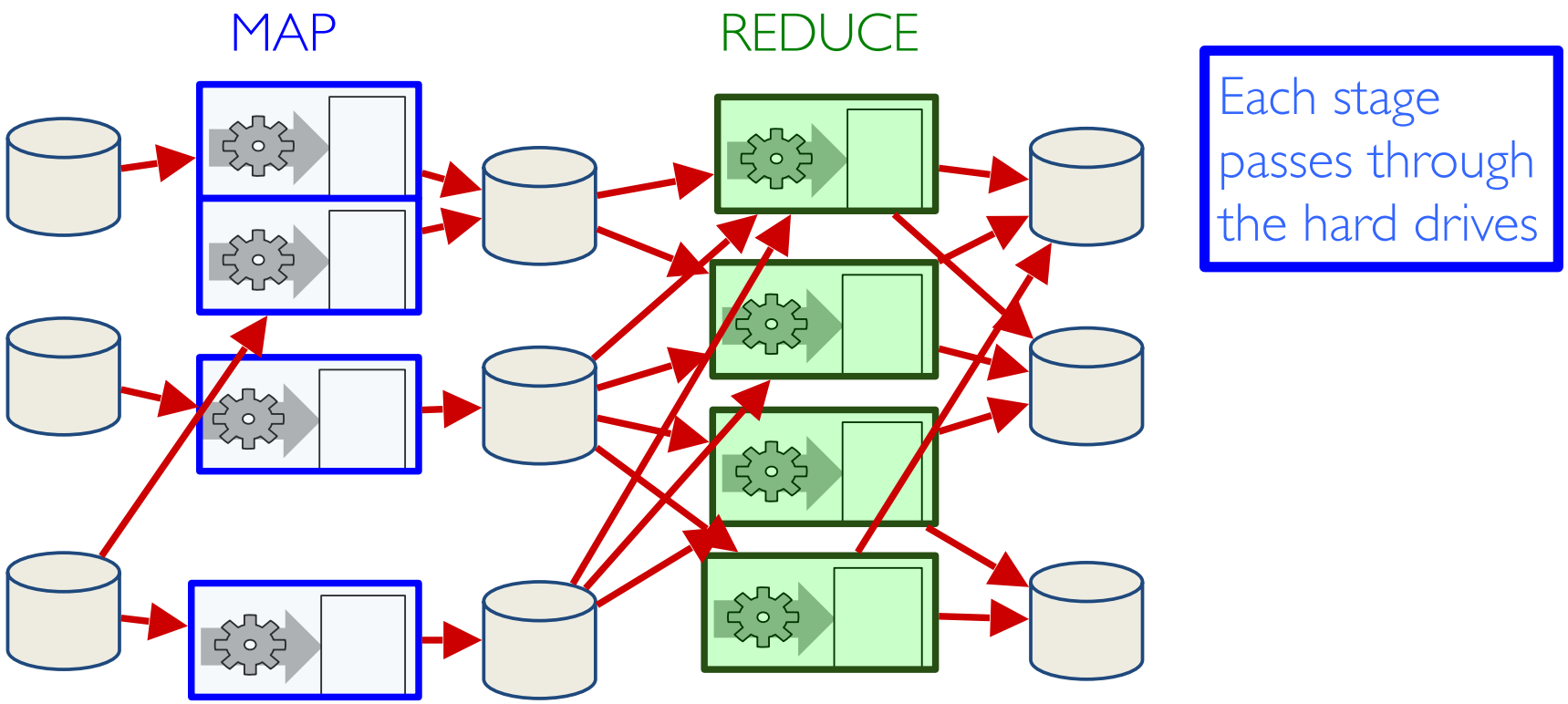
Sources

- Resilient Distributed Datasets, *Henggang Cui*
- Coursera Introduction to Apache Spark, *University of California, Databricks*

Datacenter Organization



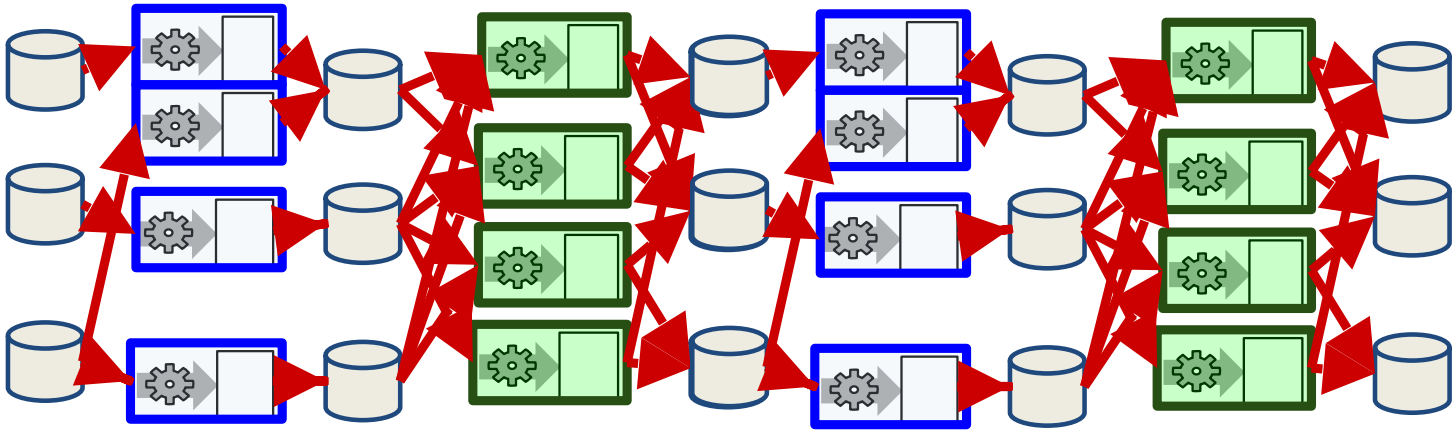
MapReduce Execution

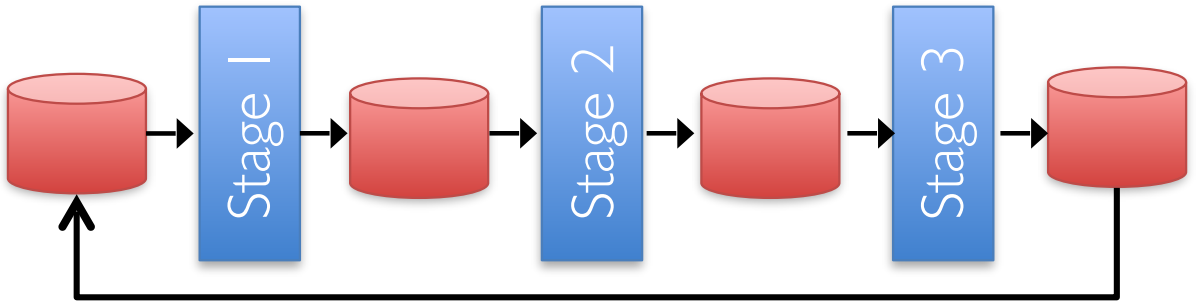


Iterative Jobs

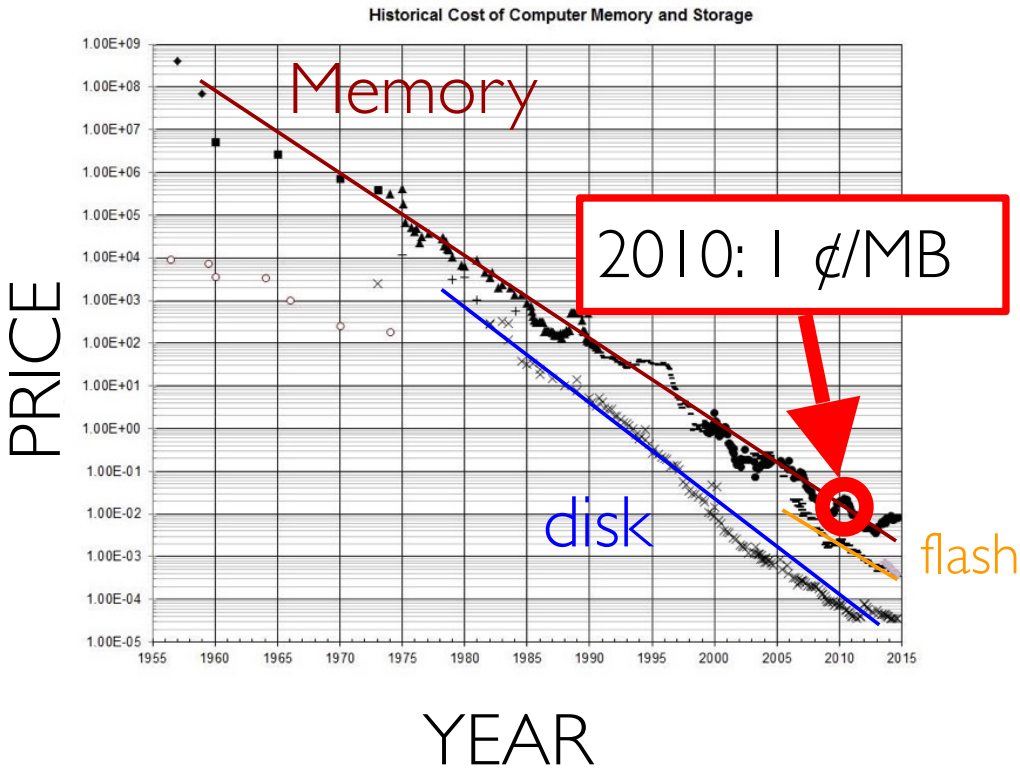
- Disk I/O for each repetition

  → Slow when executing many small iterations





Memory Cost



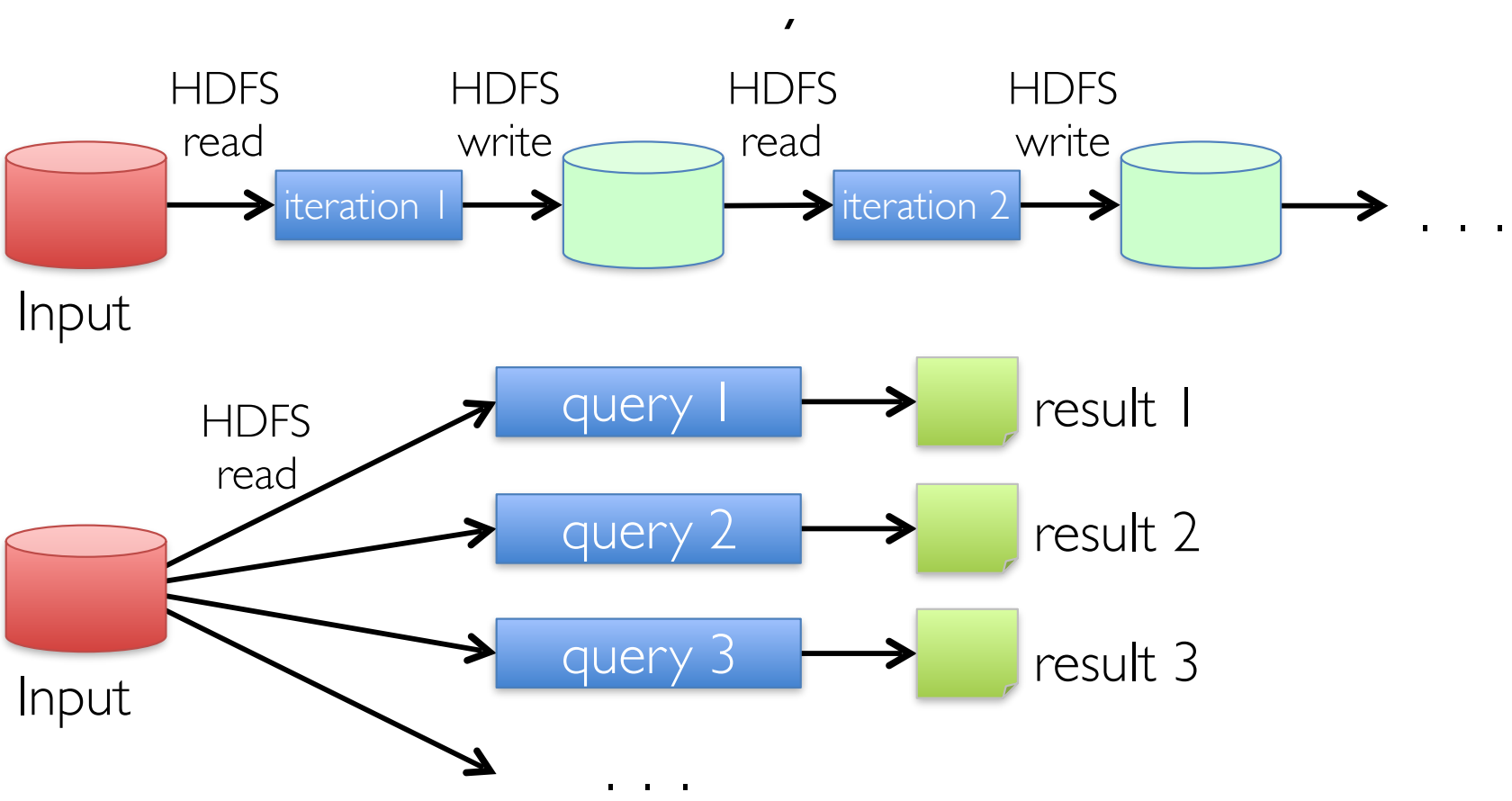
Lower cost means can put more memory in each server

In-Memory Processing

- Many datasets fit in memory (of a cluster)
- Memory is fast and avoid disk I/O

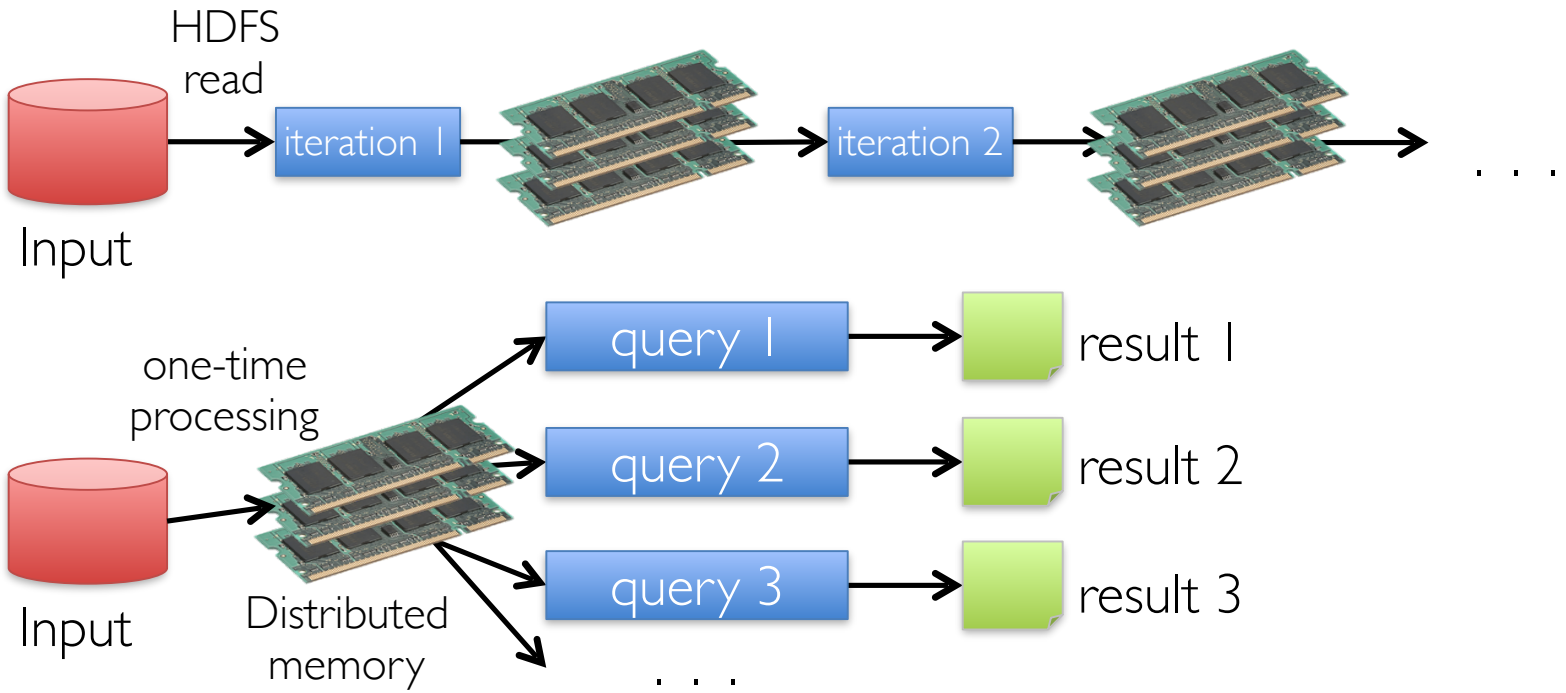
→ Spark distributed execution engine

Replace Disk with Memory



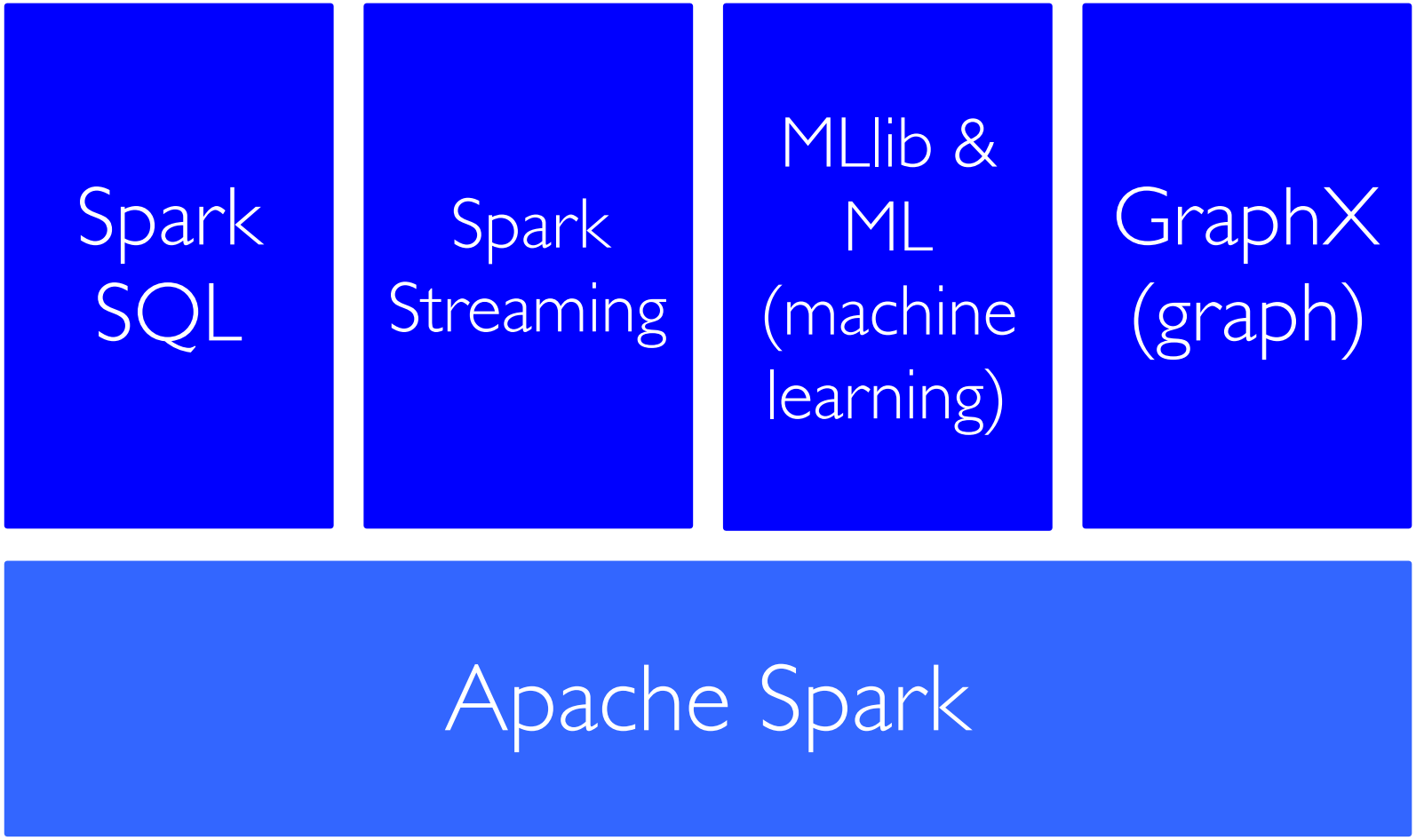
Replace Disk with Memory

In-Memory Data Sharing



10-100x faster than network and disk

Spark Architecture



Resilient Distributed Datasets (RDDs)

- Data Collection
  - Distributed
  - Read-only
  - In-memory
  - Built from stable storage or other RDDs

RDD Creation

```
# Parallelize in Python
wordsRDD = sc.parallelize(["fish", "cats", "dogs"])
```

Parallelize

Take an existing in-memory collection and pass it to SparkContext's parallelize method

```
# Read a local txt file in Python
linesRDD = sc.textFile("/path/to/README.md")
```

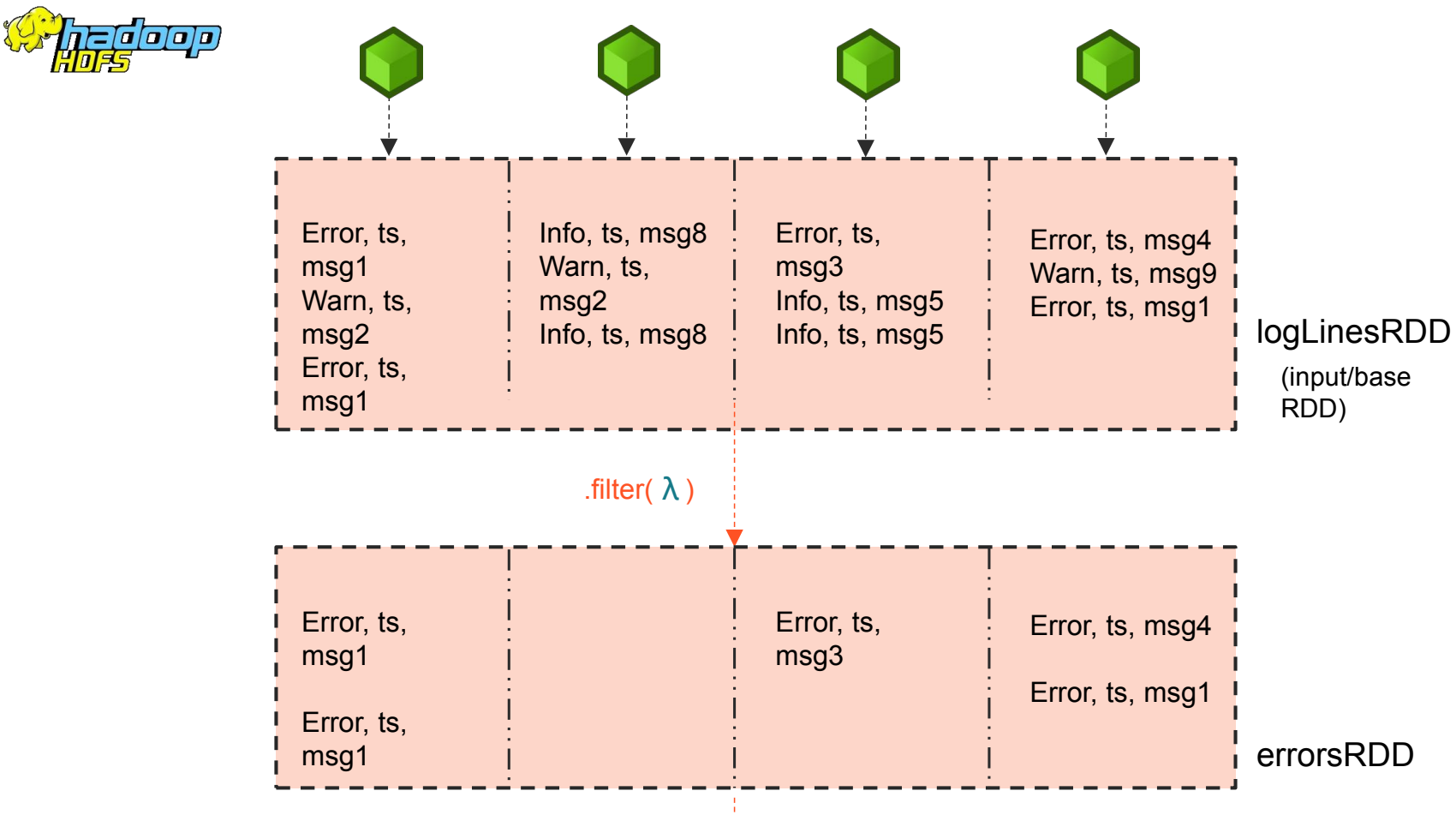
Read from Text File

There are other methods to read data from HDFS, C*, S3, HBase, etc.

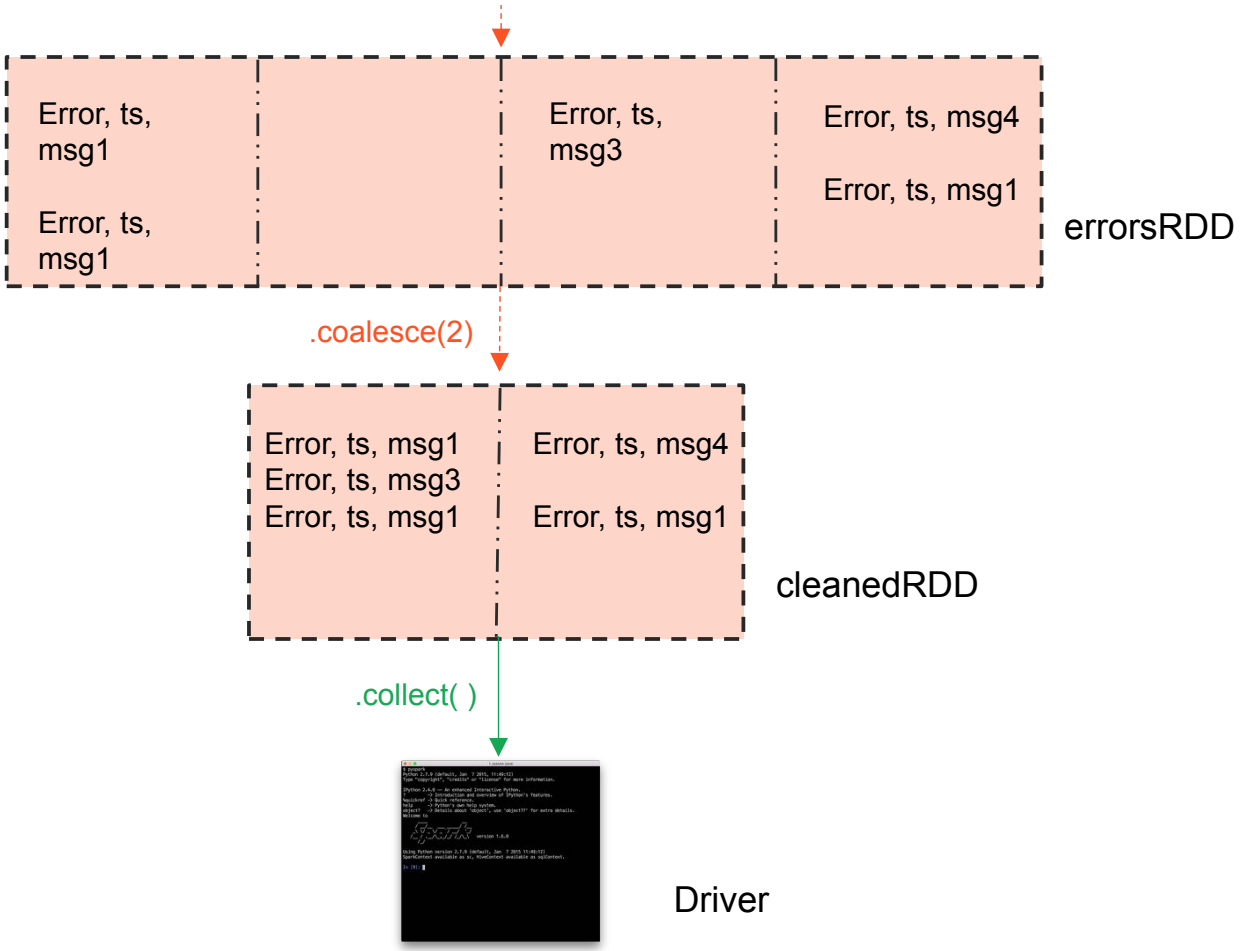
Operations on RDDs

- Transformations: lazy execution
  - Map, filter, intersection, groupByKey, zipWithIndex …
- Actions: trigger execution of transformations
  - Collect, count, reduce, saveAsTextFile …
- Caching
  - Avoid multiple executions of transformations when re-using RDD

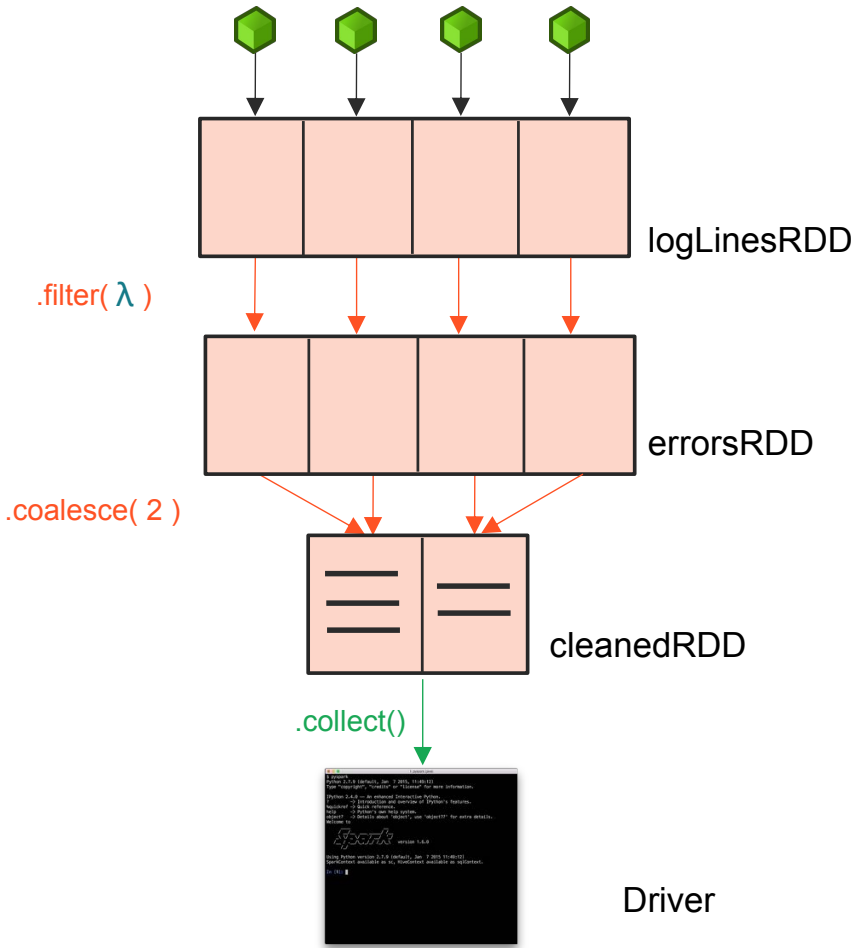
RDD Example


hadoop
HDFS
Error, ts, msg1
Warn, ts, msg2
Error, ts, msg1
Info, ts, msg8
Warn, ts, msg2
Info, ts, msg8
Error, ts, msg3
Info, ts, msg5
Info, ts, msg5
Error, ts, msg4
Warn, ts, msg9
Error, ts, msg1
logLinesRDD
(input/base RDD)
.filter(λ)
Error, ts, msg1
Error, ts, msg1
Error, ts, msg3
Error, ts, msg4
Error, ts, msg1
errorsRDD

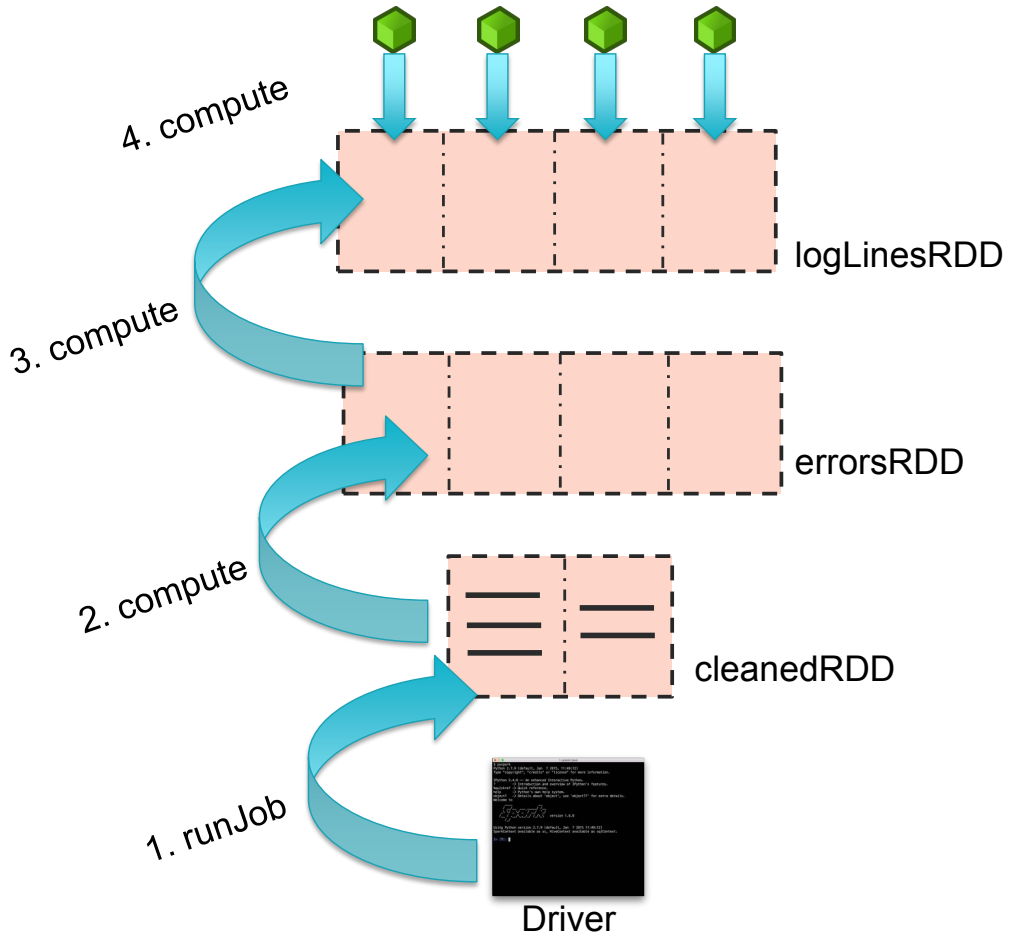
RDD Example


Error, ts, msg1
Error, ts, msg1
Error, ts, msg3
Error, ts, msg4
Error, ts, msg1
errorsRDD
.coalesce(2)
Error, ts, msg1
Error, ts, msg3
Error, ts, msg1
Error, ts, msg4
Error, ts, msg1
cleanedRDD
.collect( )
Driver

RDD Lineage


logLinesRDD
.filter(λ)
errorsRDD
.coalesce(2)
cleanedRDD
.collect()
Driver

Execution


4. compute
logLinesRDD
3. compute
errorsRDD
2. compute
cleanedRDD
1. runJob
Driver

RDD Fault Tolerance

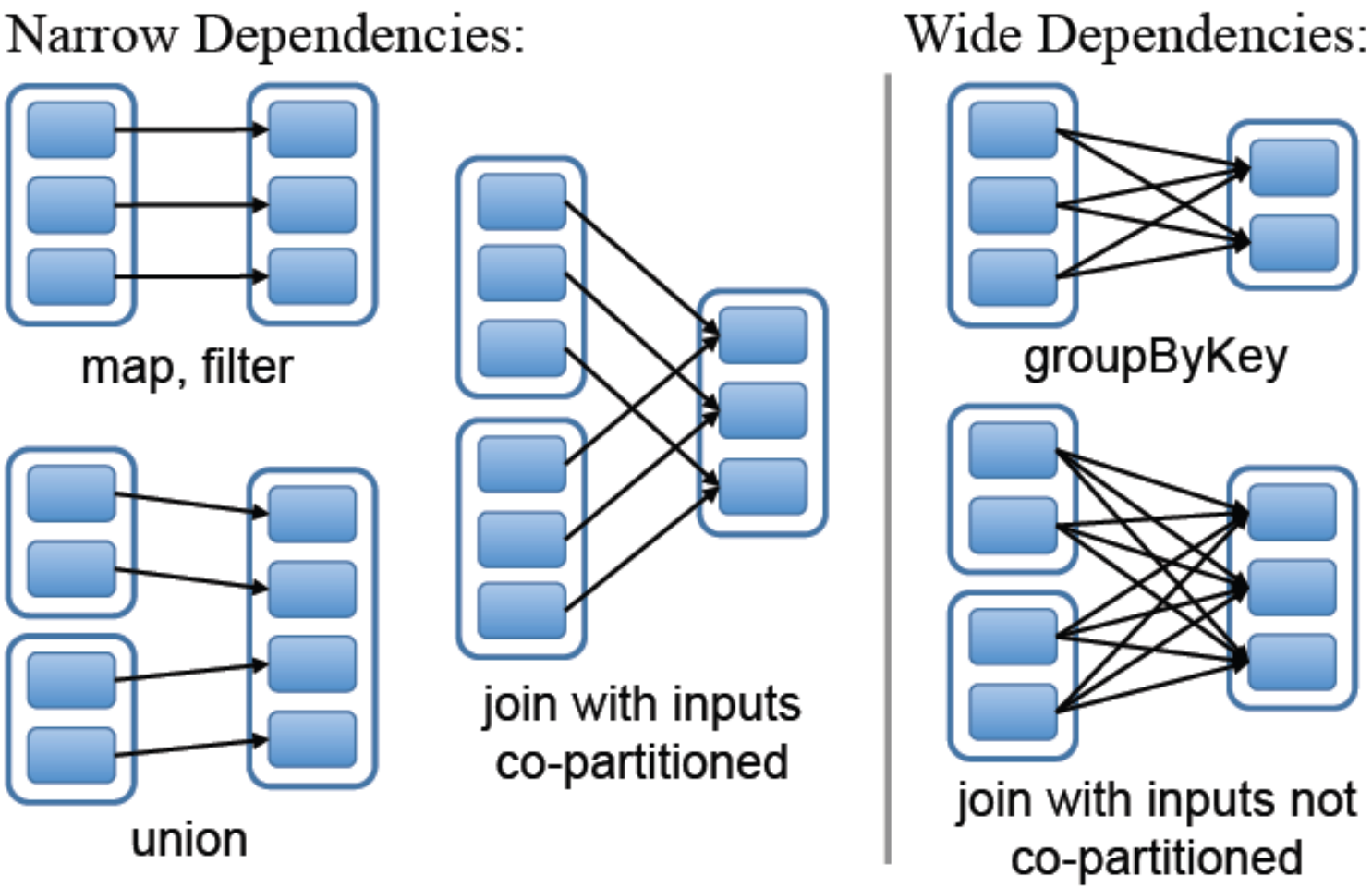
- Hadoop conservative/pessimistic approach

  →Go to disk / stable storage (HDFS)

- Spark optimistic

  →Don't "waste" time writing to disk, re-compute in case of crash using lineage

RDD Dependencies



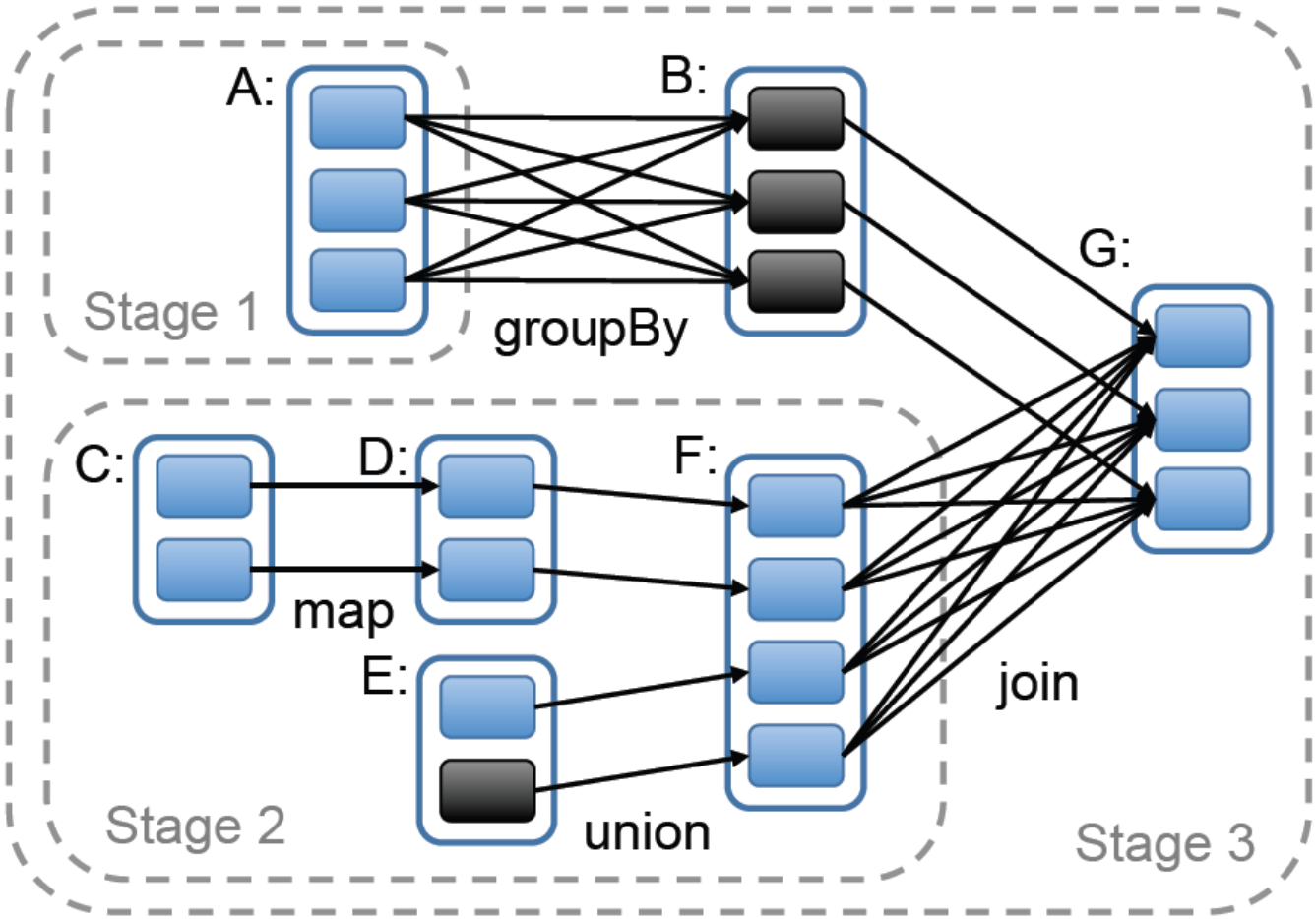
RDD Dependencies

- Narrow dependencies
  - allow for pipelined execution on one cluster node
  - easy fault recovery
- Wide dependencies
  - require data from all parent partitions to be available and to be shuffled across the nodes
  - a single failed node might cause a complete re-execution.

Job Scheduling

- To execute an action on an RDD
  - scheduler decide the stages from the RDD's lineage graph
  - each stage contains as many pipelined transformations with narrow dependencies as possible

Job Scheduling



Spark Interactive Shell

```
scala> val wc = lesMiserables.flatMap(_.split(" ")).map((_,1)).reduceByKey(_+_)
wc: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[5] at reduceByKey at <console>:14
scala> wc.take(5).foreach(println)
(créanciers;,1)
(abondent.,1)
(plaisir,,5)
(déplaçaient,1)
(sociale,,7)
scala> val cw = wc.map(p => (p._2, p._1))
cw: org.apache.spark.rdd.RDD[(Int, String)] = MappedRDD[5] at map at <console>:16
scala> val sortedCW =  cw.sortByKey(false)
sortedCW: org.apache.spark.rdd.RDD[(Int, String)] = ShuffledRDD[11] at sortByKey at <console>:18
scala> sortedCW.take(5).foreach(println)
(16757,de)
(14683,)
(11025,la)
(9794,et)
(8471,le)
scala> sortedCW.filter(x => "Cosette".equals(x._2)).collect.foreach(println)
(353,Cosette)
```

Wordcount