Big Data Frameworks: Internals

Mohammad A. Hoque
mohammad.a.hoque@helsinki.fi

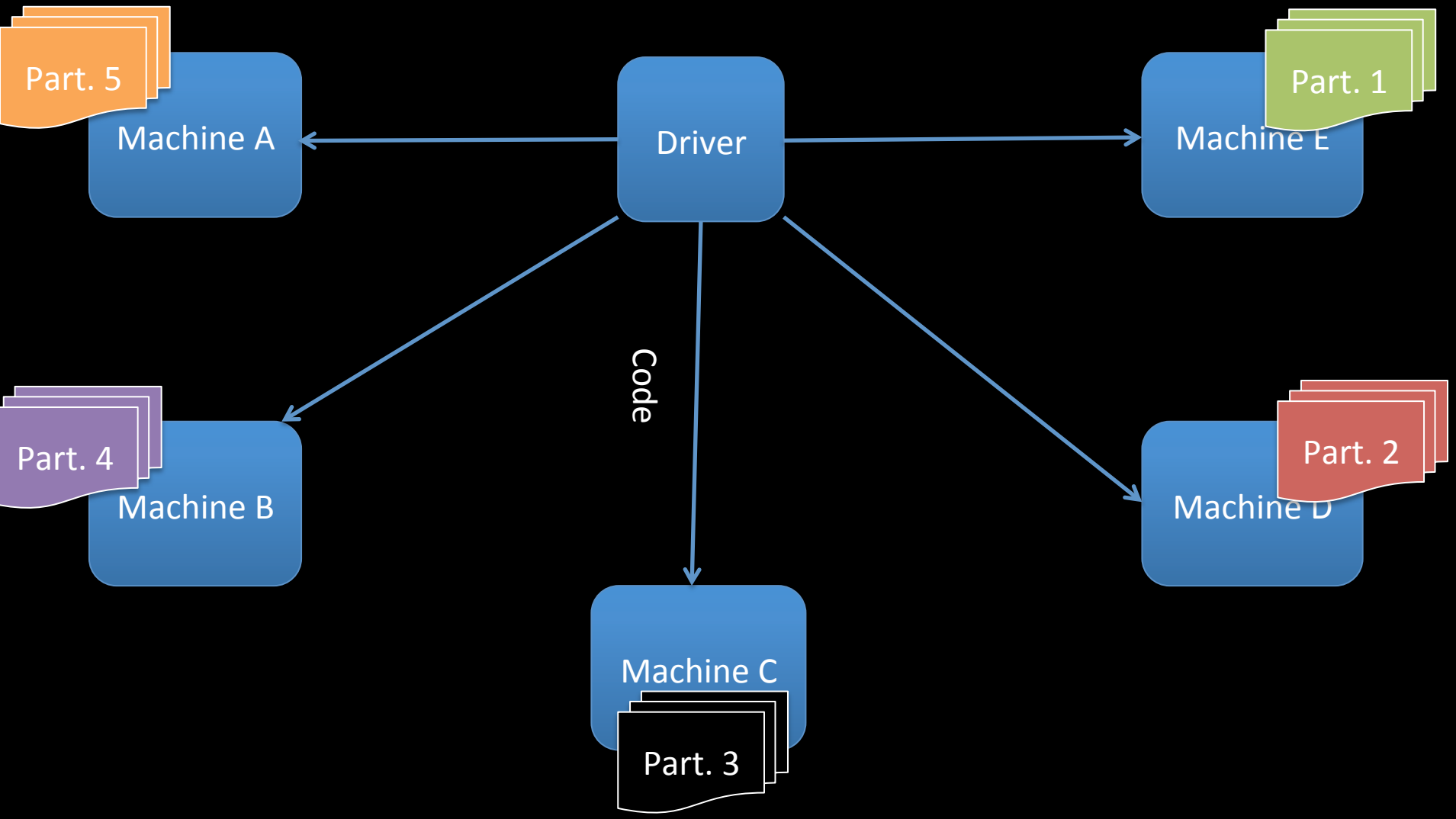
Spark Framework Application

```
object WordCount {
  def main (args: Array[String]){
    val driver = "spark://192.168.0.3:7077"
    val sc = new SparkContext(driver, "SparkWordCount")
    val numPartitions = 10
    val lines = sc.textFile("here"+"sometext.txt", numPartitions)
    val words = lines.flatMap (_.split(" "))
    val groupWords = words.groupBy { x => x}
    val wordCount = groupWords.map( x => (x._1,x._2.size))
    val result = wordCount.saveAsTextFile("there"+"wordcount")
  }
}
```

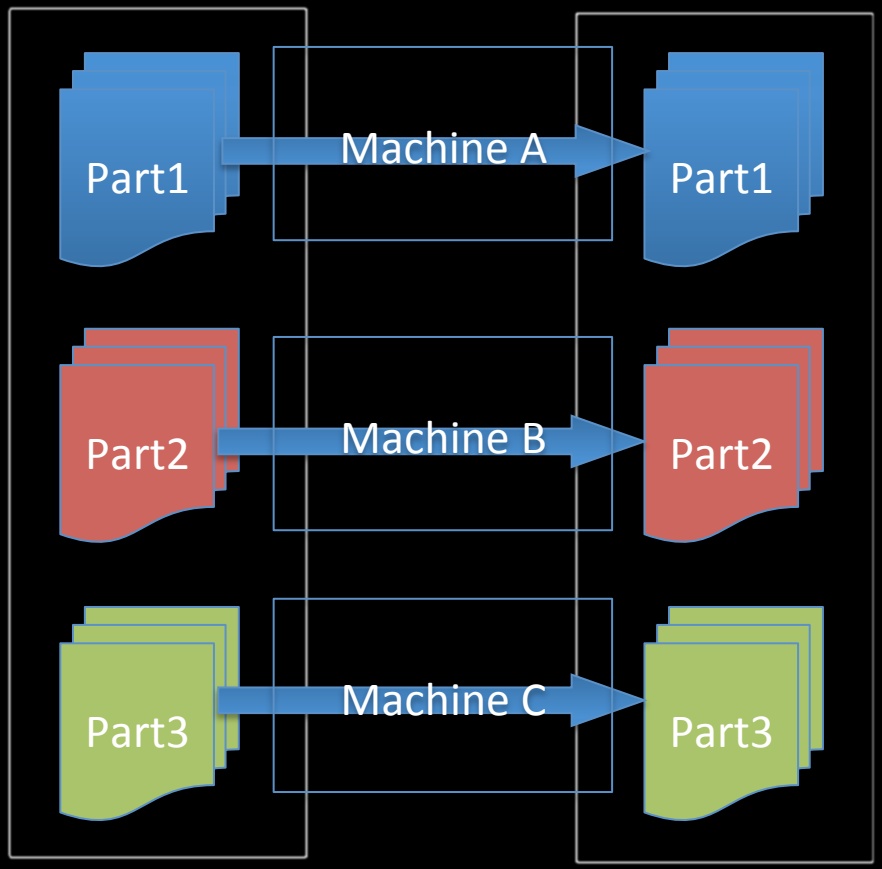
Spark Application Framework

- SparkContext initializes the application driver and gives the application execution to the driver.
- RDD is generated from the external data sources; such as HDFS.
- RDD goes through a number of **Transformations**; such a Map, flatMap, sortByKey, etc.
- Finally, the count/collect/save/take **Action** is performed, which converts the final RDD into an output for storing to an external source.

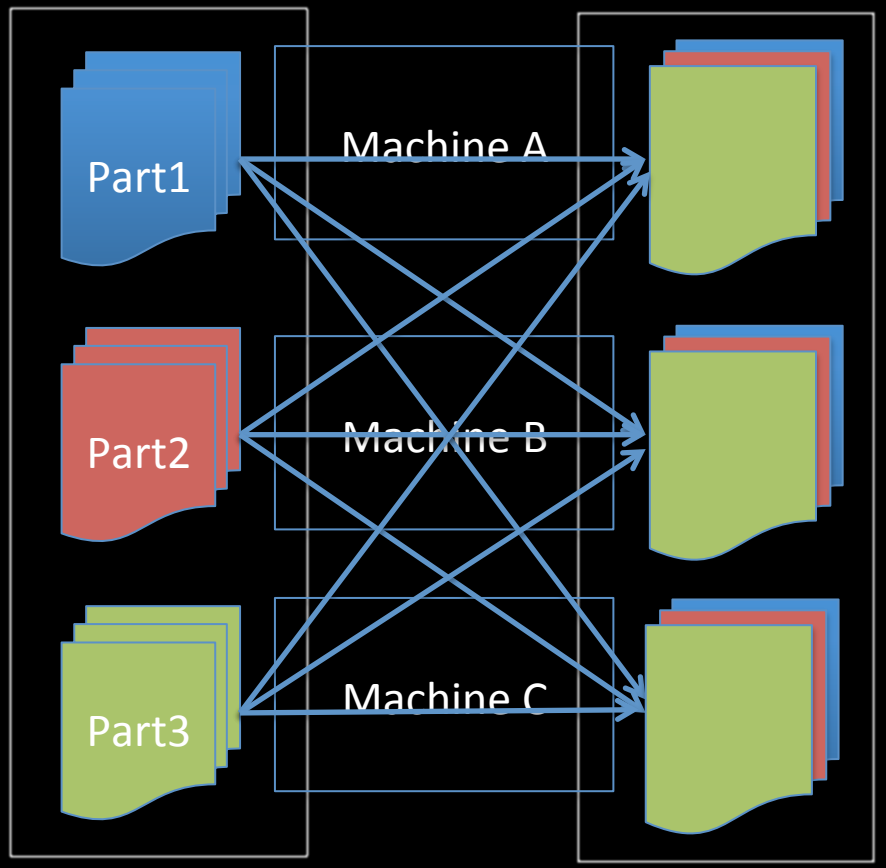
Spark Application Framework



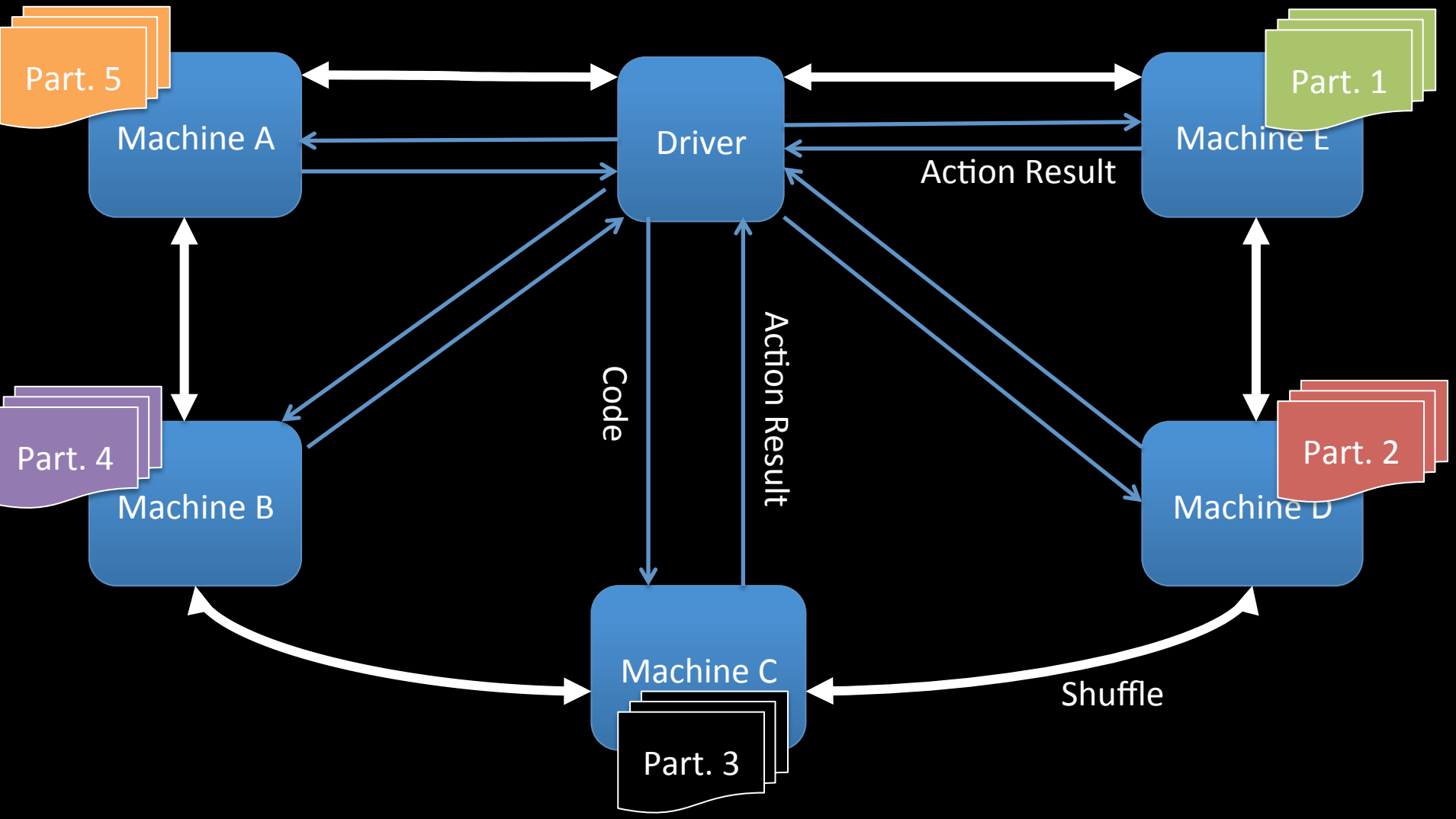
Transformation



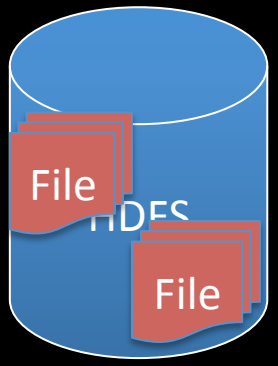
Transformation with Shuffling



Spark Application Framework

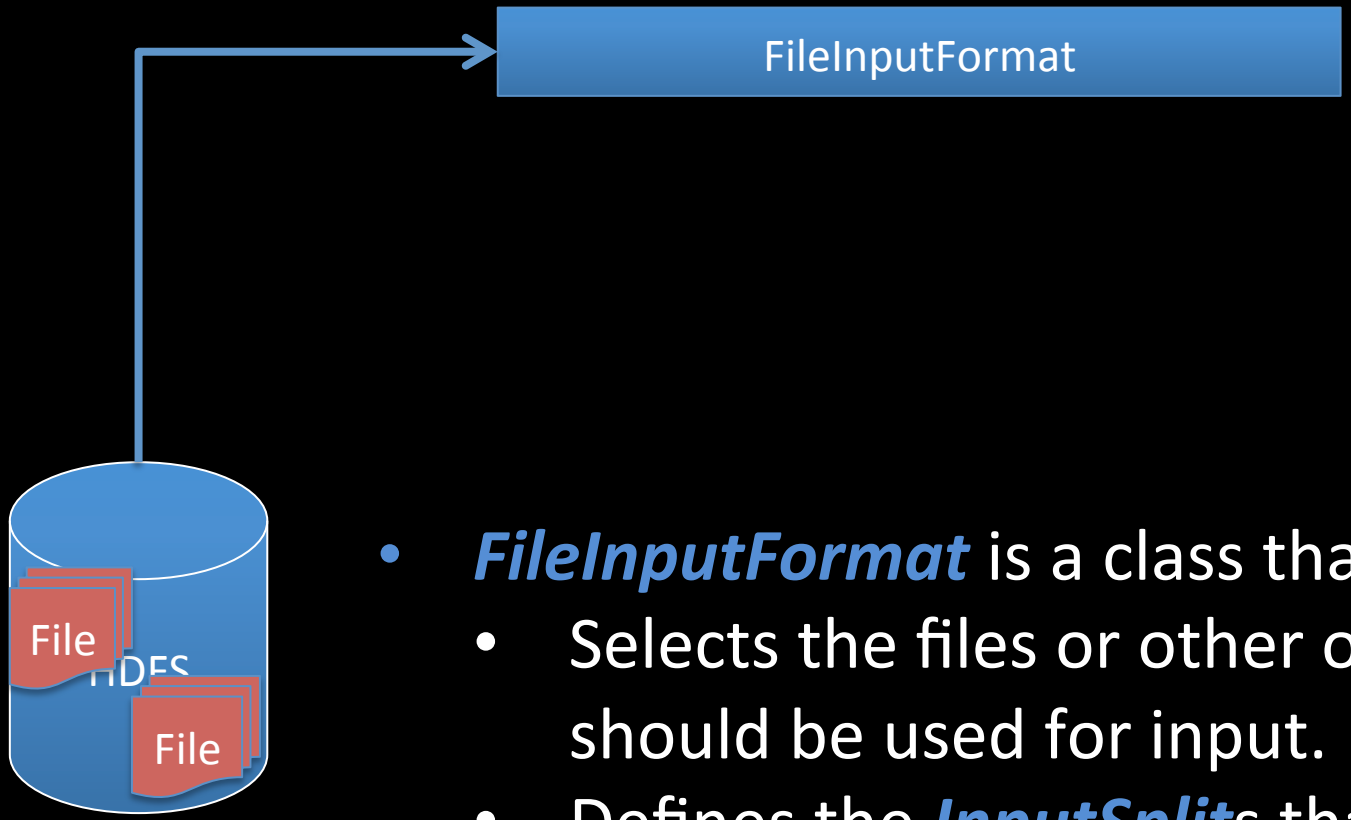


RDD is generated from the input file stored in HDFS

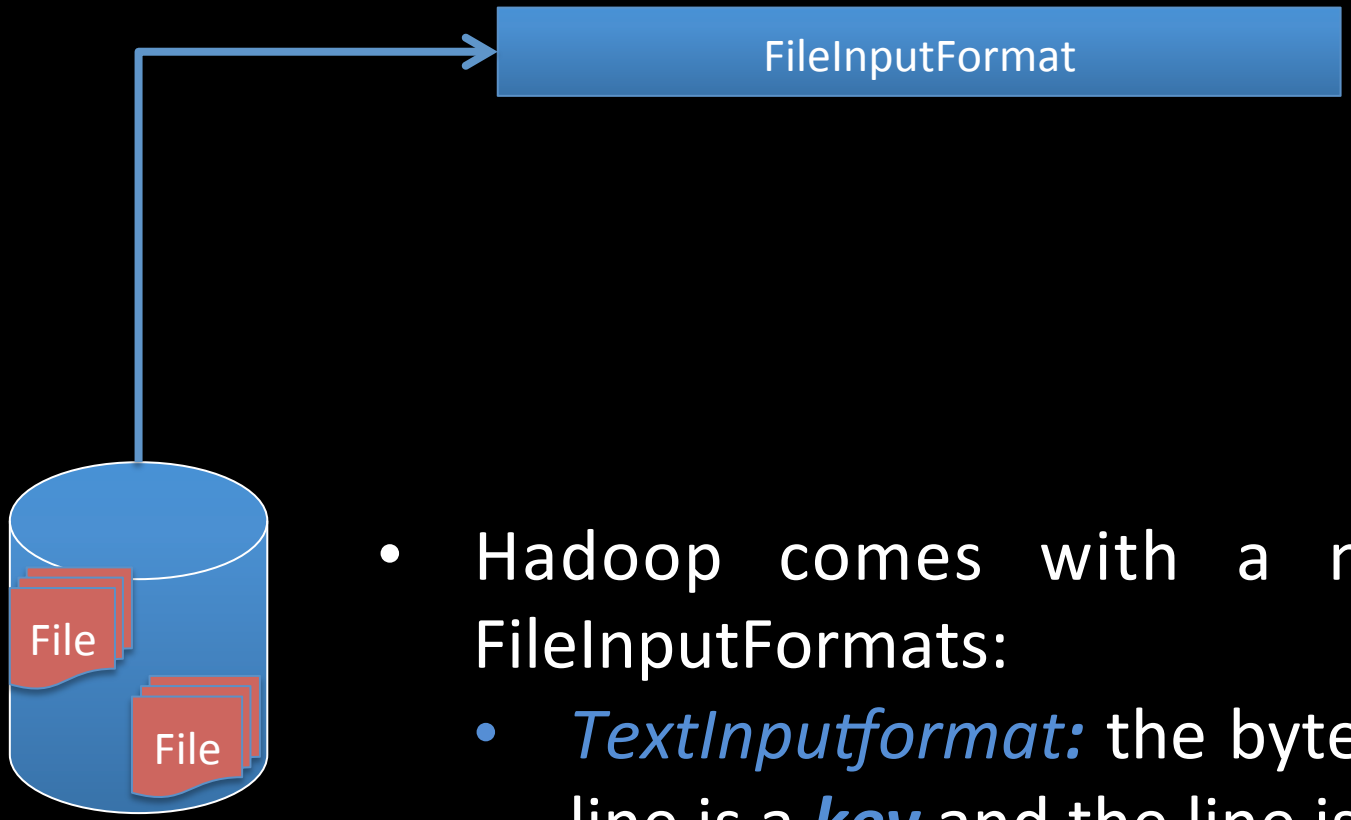


- This is where the data for a MapReduce task is initially stored and the files typically reside in HDFS.

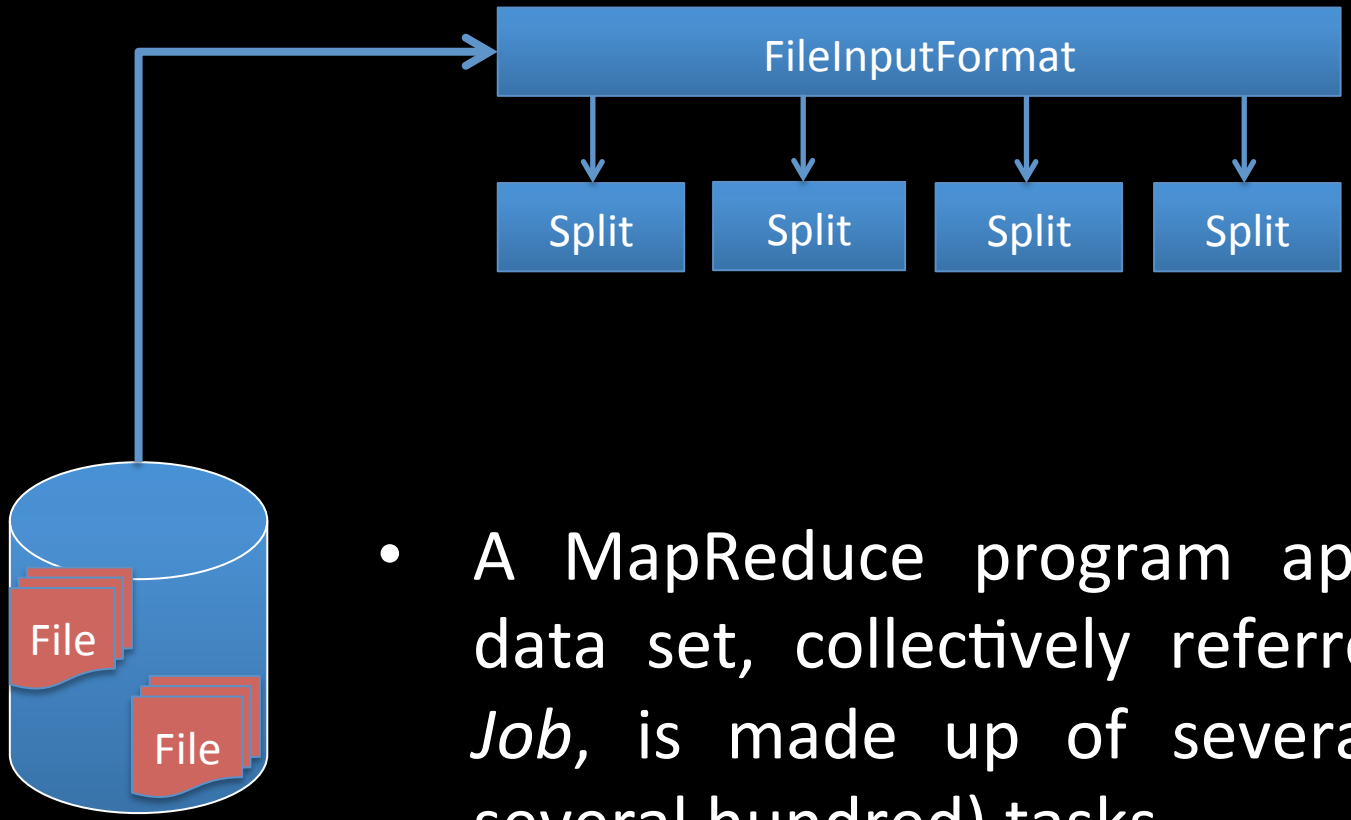
- The format of these files; text and binary
  - Text – Single Line (JSON)
  - Multi-Line (XML)
  - Binary file, with fixed size objects



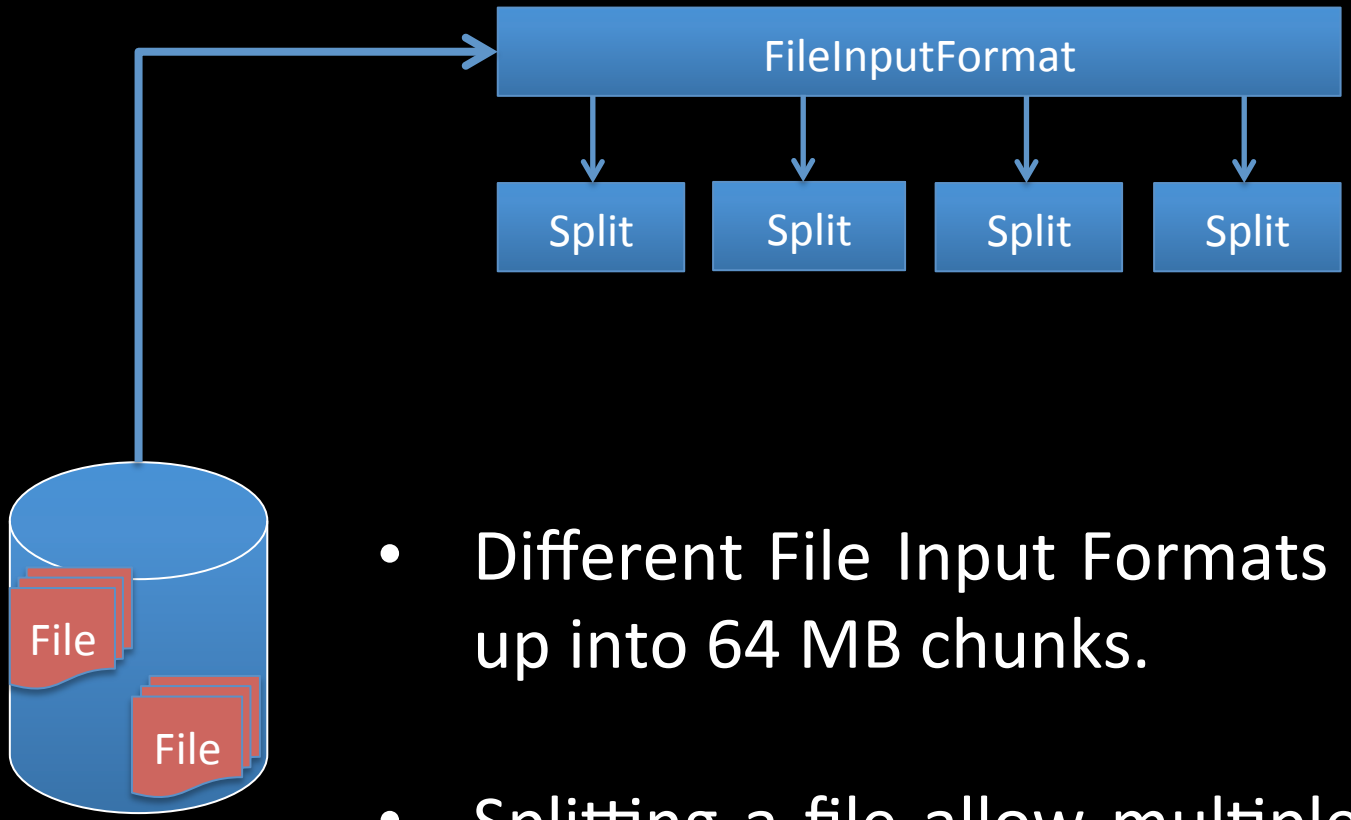
- ***FileInputFormat*** is a class that
  - Selects the files or other objects that should be used for input.
  - Defines the ***InputSplit***s that break a file into tasks.
  - Provides a factory for ***RecordReader*** objects that read the file.



- Hadoop comes with a number of FileInputFormats:
  - *TextInputformat:* the byte offset of a line is a ***key*** and the line is the ***value***.
  - *KeyValueInputFormat:* the text until the first tab is the key and the remaining is the value.
  - *SequenceFileInputFormat:* Object files. Key and values are defined by the user.

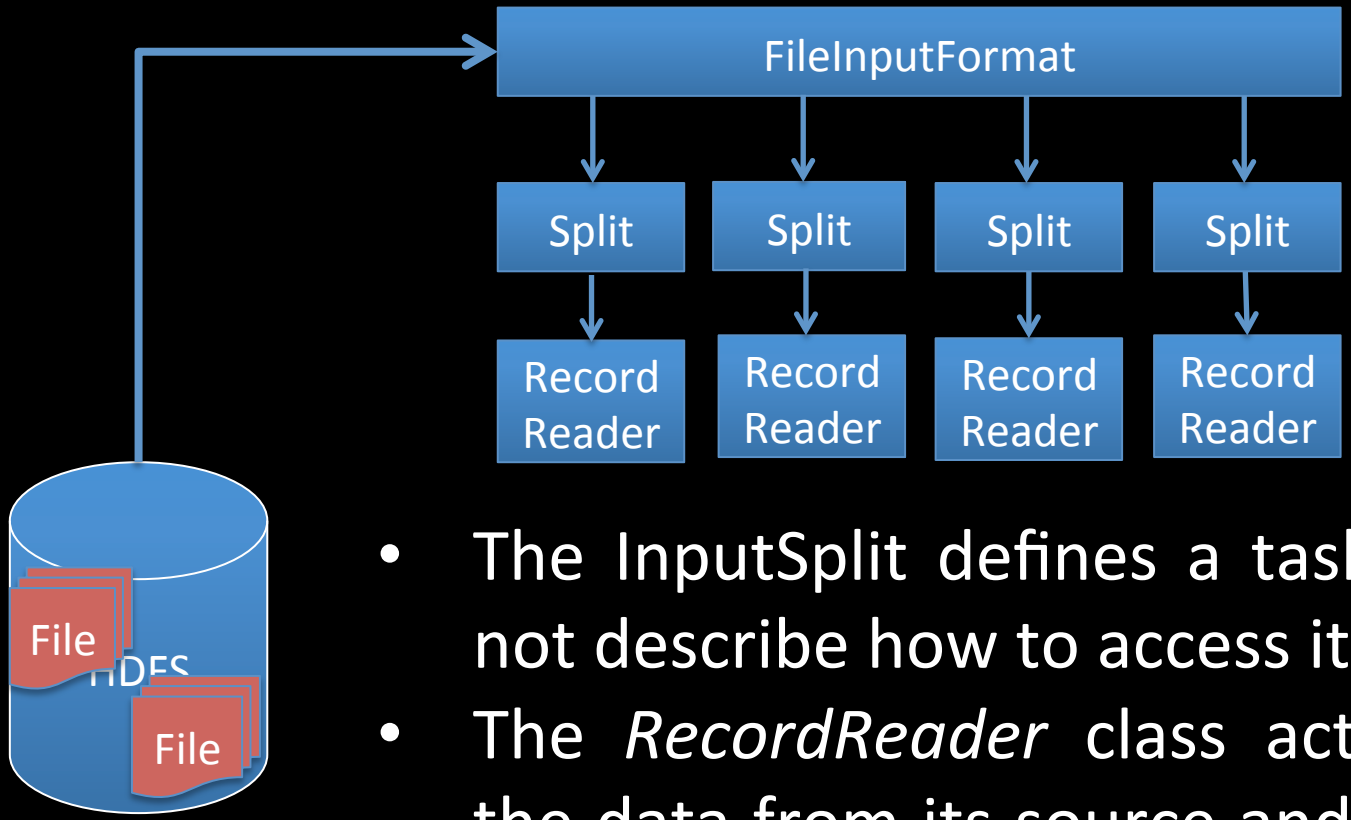


- A MapReduce program applied to a data set, collectively referred to as a *Job*, is made up of several (possibly several hundred) tasks.
- An InputSplit describes a unit of work that comprises a single *map task* in a MapReduce program.
- A Map tasks may involve reading a whole file; they often involve reading only part of a file.

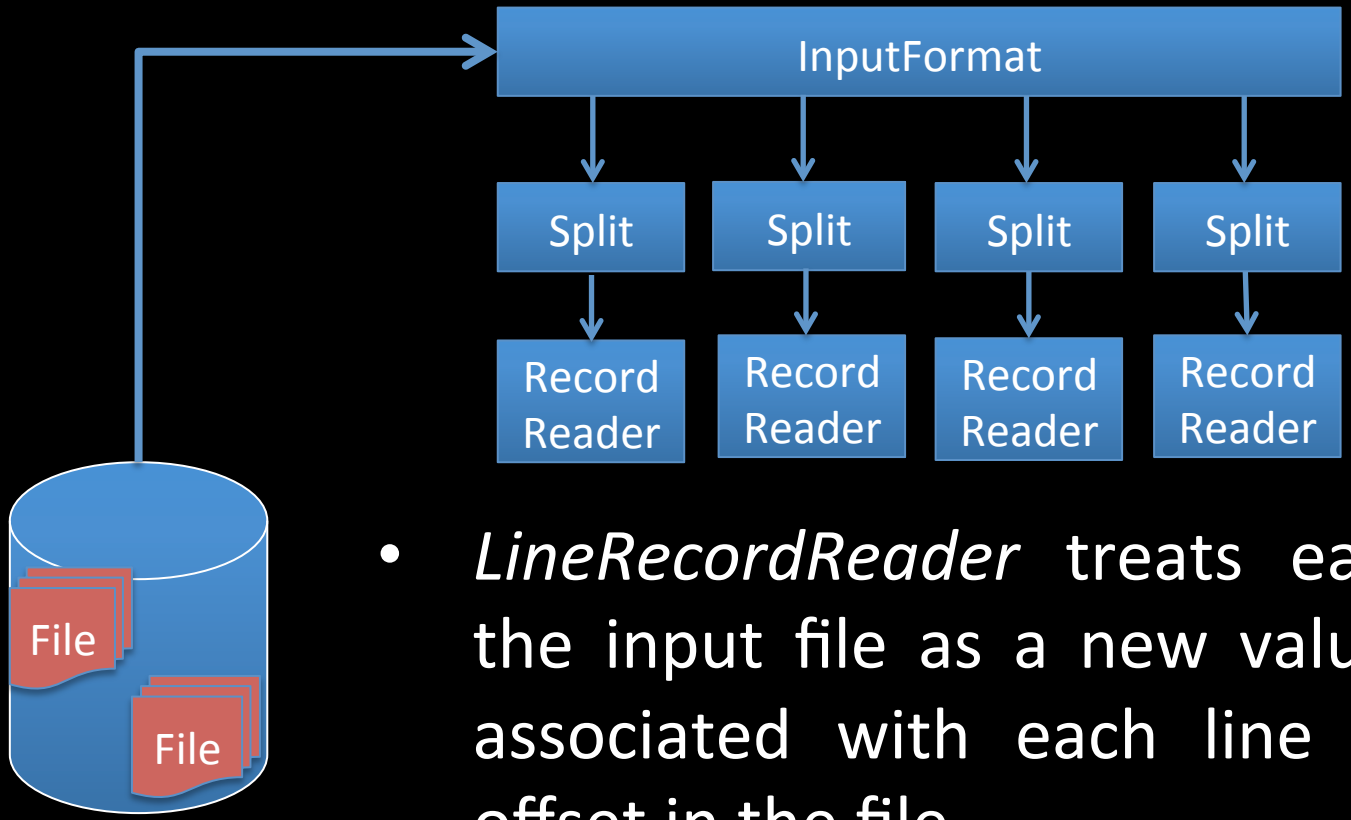


- Different File Input Formats break a file up into 64 MB chunks.
- Splitting a file allow multiple map tasks over a single file in parallel.
- If the file is very large, the performance can be improved significantly through such parallelism.

```
for (FileStatus file: files) {
    Path path = file.getPath();
    FileSystem fs = path.getFileSystem(job.getConfiguration());
    long length = file.getLen();
    BlockLocation[] blkLocations = fs.getFileBlockLocations(file, 0, length);
    if ((length != 0) && isSplitable(job, path)) {
      long blockSize = file.getBlockSize();
      long splitSize = computeSplitSize(blockSize, minSize, maxSize);
      long bytesRemaining = length;
      while (((double) bytesRemaining)/splitSize > SPLIT_SLOP) {
        int blkIndex = getBlockIndex(blkLocations, length-bytesRemaining);
        splits.add(new FileSplit(path, length-bytesRemaining, splitSize,
                                 blkLocations[blkIndex].getHosts()));
        bytesRemaining -= splitSize;
      }
      if (bytesRemaining != 0) {
        splits.add(new FileSplit(path, length-bytesRemaining, bytesRemaining,
                   blkLocations[blkLocations.length-1].getHosts()));
      }
    } else if (length != 0) {
      splits.add(new FileSplit(path, 0, length, blkLocations[0].getHosts()));
    } else {
       splits.add(new FileSplit(path, 0, length, new String[0]));
    }
  }
```



- The InputSplit defines a task, but does not describe how to access it.
- The *RecordReader* class actually loads the data from its source and converts it into (key, value) pairs suitable for reading by the Mapper.



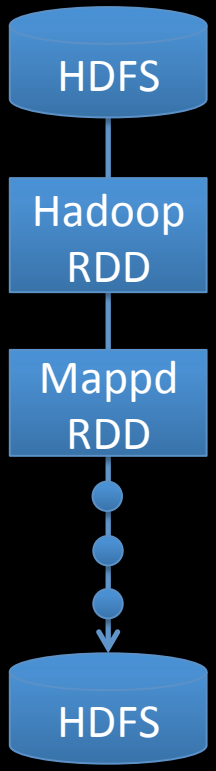
- *LineRecordReader* treats each line of the input file as a new value. The key associated with each line is its byte offset in the file.
- The RecordReader is invoke repeatedly until the entire InputSplit has been consumed.
- Each invocation of the RecordReader leads to another call to the map() method of the Mapper.

Mahout XMLRecordReader

```
public XmlRecordReader(FileSplit split,
JobConf jobConf) throws IOException
{
        startTag = jobConf.get(”<article>").
                getBytes("utf-8");
        endTag = jobConf.get(”<article>”).
                getBytes("utf-8");
        start = split.getStart();
        end = start + split.getLength();
        Path file = split.getPath();
        FileSystem fs =
        file.getFileSystem(jobConf);
        fsin = fs.open(split.getPath());
        fsin.seek(start);
}
```

```
public boolean next(LongWritable key, Text
value) throws IOException {
if (fsin.getPos() < end) {
     if (readUntilMatch(startTag, false)) {
       try {
                buffer.write(startTag);
                if (readUntilMatch(endTag, true)) {
                    key.set(fsin.getPos());
                    value.set(buffer.getData(), 0,
buffer.getLength());
                          return true;
                        }

              } finally {buffer.reset();}
       }
    }
     return false;
}
```


HDFS
Hadoop RDD
Mappd RDD
HDFS

RDD goes through a number of Transformations

Spark Application

```
1.  object WordCount {
2.    def main (args: Array[String]){
3.      val driver = "spark://192.168.0.3:7077"
4.      val sc = new SparkContext(driver, "SparkWordCount")
5.      val numPartitions = 10
6.      val lines = sc.textFile("here"+"sometext.txt", numPartitions)
7.      val words = lines.flatMap (_.split(" "))
8.      val groupWords = words.groupBy { x => x}
9.      val wordCount = groupWords.map( x => (x._1,x._2.size))
10.     val result = wordCount.saveAsTextFile("there"+"wordcount")
11.   }
12. }
```

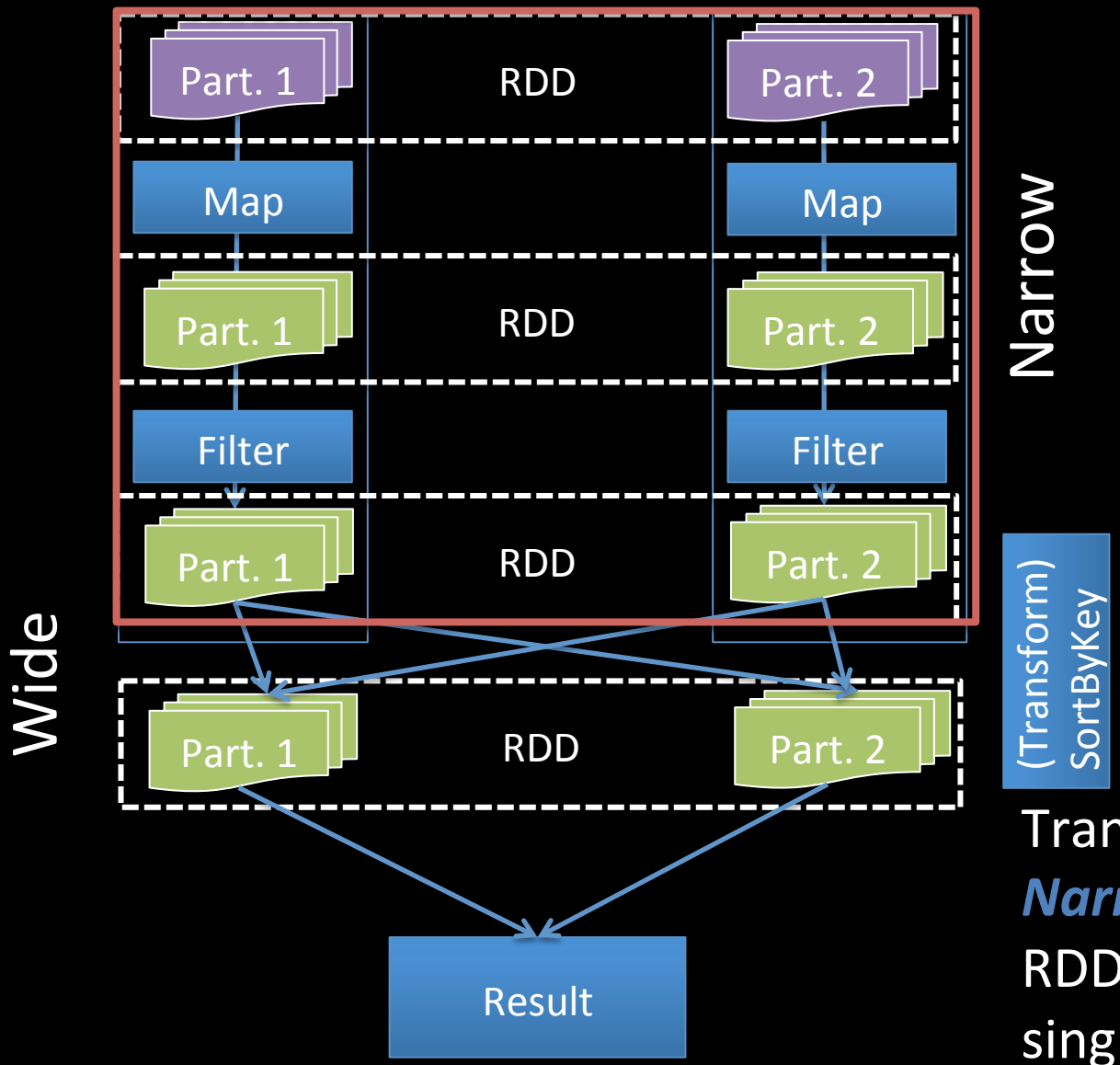
Spark Application Execution

- How does the spark submit the job to the worker?
  - (1) Map, (2) flatMap, (2) groupBy, (3) Map, Or
  - (1) Map ->flatMap, (2) groupBy, (3) Map
- How Many tasks per submission?
- Before submitting tasks of a job, how does the driver know about the resource information of the workers?

How does the spark submit the job to the workers?

DAG Scheduler

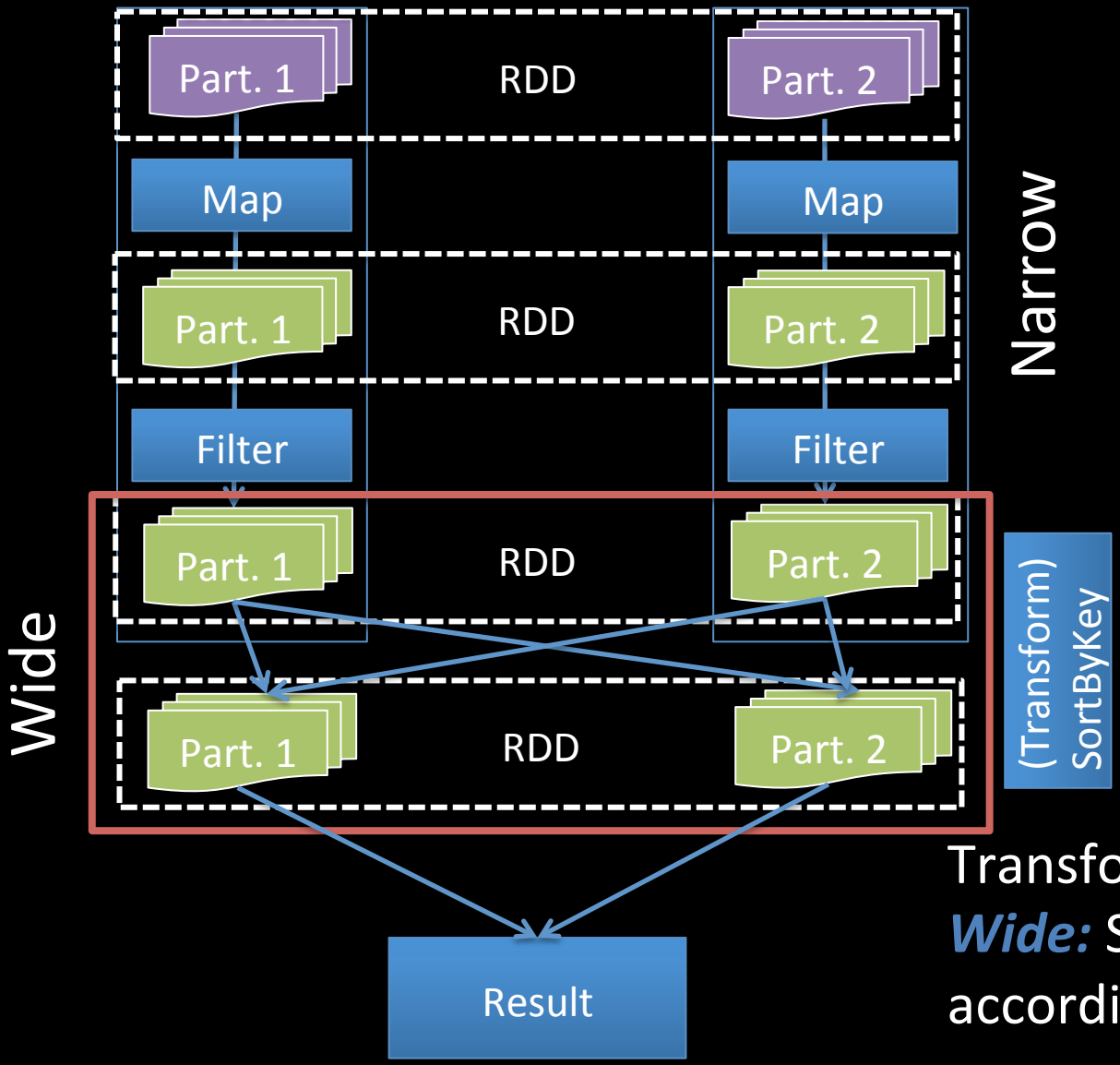
DAG Scheduler



Transformation

Narrow : All partitions of an RDD will be consumed by a single child RDD , no shuffling.

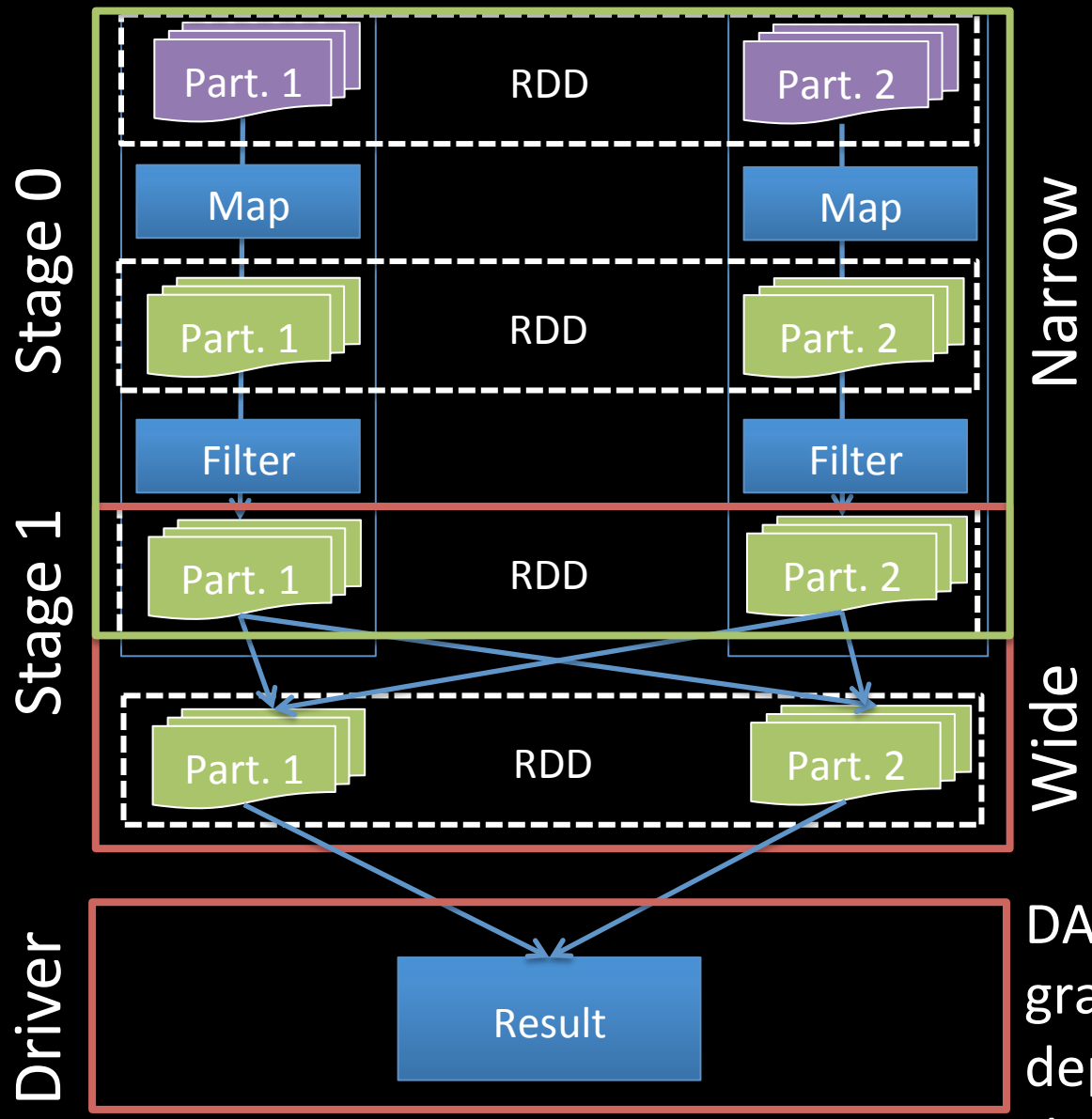
DAG Scheduler



Transformation

Wide: Shuffling takes place according to their key value.

DAG Scheduler

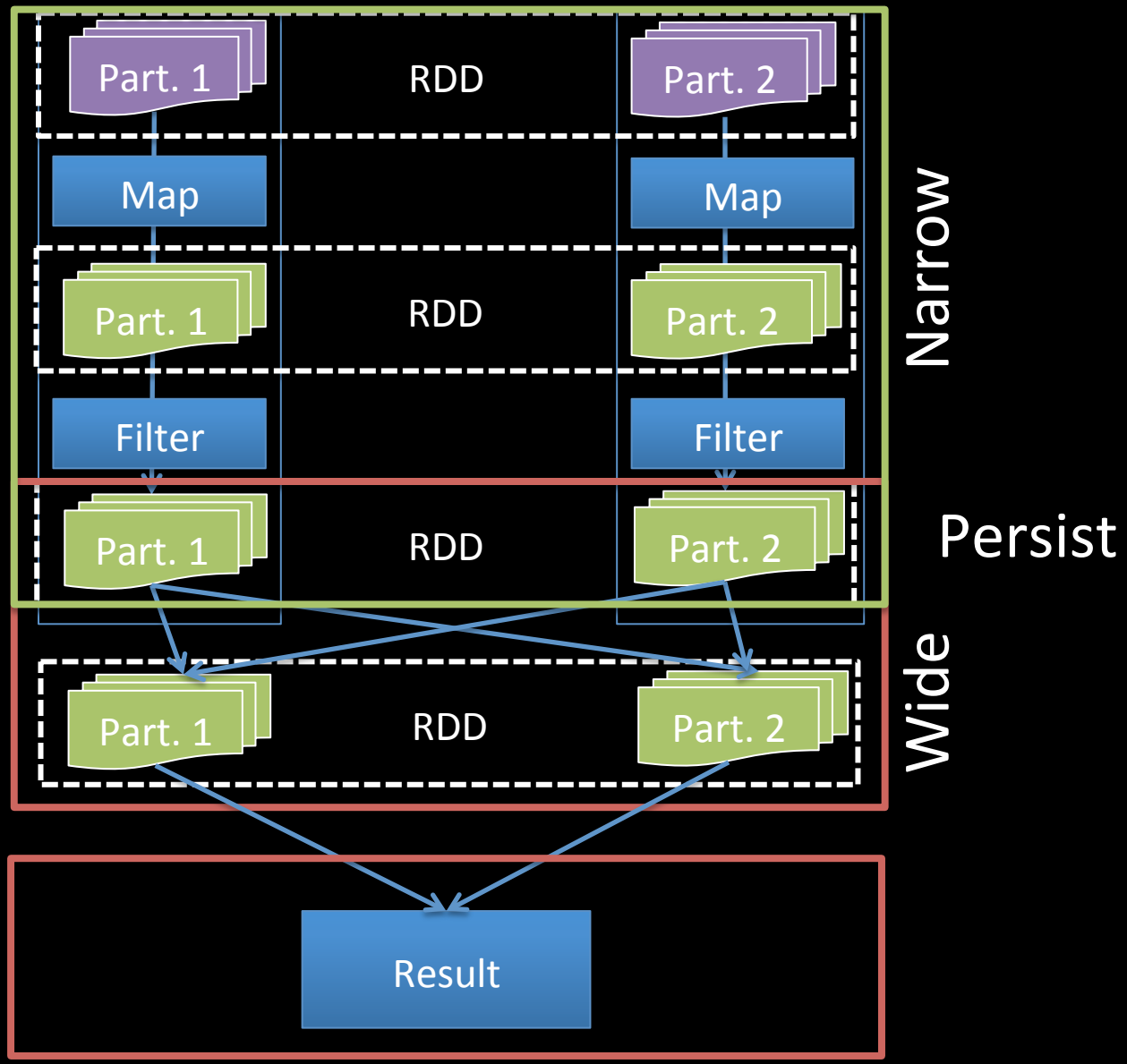


DAG Scheduler Splits the graph according to the dependency and submits to the lower layer Scheduler.

DAG Scheduler

DAG serves Fault tolerance

- If a partition is lost while executing a stage consisting of transformations with Narrow dependencies
  - It traces back and re-computes only the lost partition of the parent RDD, and
  - The responsible machine will take care.
- In the case of Wide dependencies, the lost partition can affect a lot of others
  - Spark mitigates this my persisting the last computed partitions before the shuffling takes place.
- There is also checkpoint API which enables to persist RDD on desired transformation.


Part. 1
RDD
Part. 2
Map
Map
Part. 1
RDD
Part. 2
Filter
Filter
Part. 1
RDD
Part. 2
Part. 1
RDD
Part. 2
Result
Narrow
Persist
Wide

DAG Scheduler

INFO [Executor task launch worker-1] (Logging.scala:59) - Input split: file:/data/mmbrain/2015/inputJson/group5/part-dp:939524096+33554432
INFO [task-result-getter-3] (Logging.scala:59) - Finished task 27.0 in **stage 0.0** (TID 27) in 2455 ms on localhost (28/43)
INFO [Executor task launch worker-1] (Logging.scala:59) - Finished task 28.0 in stage 0.0 (TID 28). 4565 bytes result sent to driver
INFO [sparkDriver-akka.actor.default-dispatcher-2] (Logging.scala:59) - Starting task 29.0 in stage 0.0 (TID 29, localhost, PROCESS_LOCAL, 1650 byte
INFO [Executor task launch worker-1] (Logging.scala:59) - Running task 29.0 in stage 0.0 (TID 29)

INFO [Executor task launch worker-1] (Logging.scala:59) - Input split: file:/data/mmbrain/2015/inputJson/group5/part-dp:973078528+33554432
INFO [task-result-getter-0] (Logging.scala:59) - Finished task 28.0 in stage 0.0 (TID 28) in 2084 ms on localhost (29/43)
INFO [Executor task launch worker-1] (Logging.scala:59) - Finished task 29.0 in stage 0.0 (TID 29). 4728 bytes result sent to driver
INFO [sparkDriver-akka.actor.default-dispatcher-2] (Logging.scala:59) - Starting task 30.0 in stage 0.0 (TID 30, localhost, PROCESS_LOCAL, 1650 byte

INFO [Executor task launch worker-1] (Logging.scala:59) - Running task 9.0 in **stage 1.0** (TID 52)
INFO [Executor task launch worker-1] (Logging.scala:59) - Partition rdd_9_9 not found, computing it
INFO [Executor task launch worker-1] (Logging.scala:59) - Input split: file:/data/mmbrain/2015/inputJson/group5/part-dp:301989888+33554432
INFO [Executor task launch worker-1] (Logging.scala:59) - ensureFreeSpace(16880) called with curMem=471734, maxMem=4123294433
INFO [Executor task launch worker-1] (Logging.scala:59) - Block rdd_9_9 stored as values in memory (estimated size 16.5 KB, free 3.8 GB)
INFO [sparkDriver-akka.actor.default-dispatcher-14] (Logging.scala:59) - Added rdd_9_9 in memory on localhost:43449 (size: 16.5 KB, free: 3.8 GB)
INFO [Executor task launch worker-1] (Logging.scala:59) - Updated info of block rdd_9_9

How Many tasks per Stage?

Number of Tasks

- Number of InputSplit defines the number of tasks.
  - Hadoop FileInputFormat defines
  - Immediate Narrow transformations will follow the parent
- If GroupBy is used
  - The number of keys define the number of tasks

Control Number of Tasks

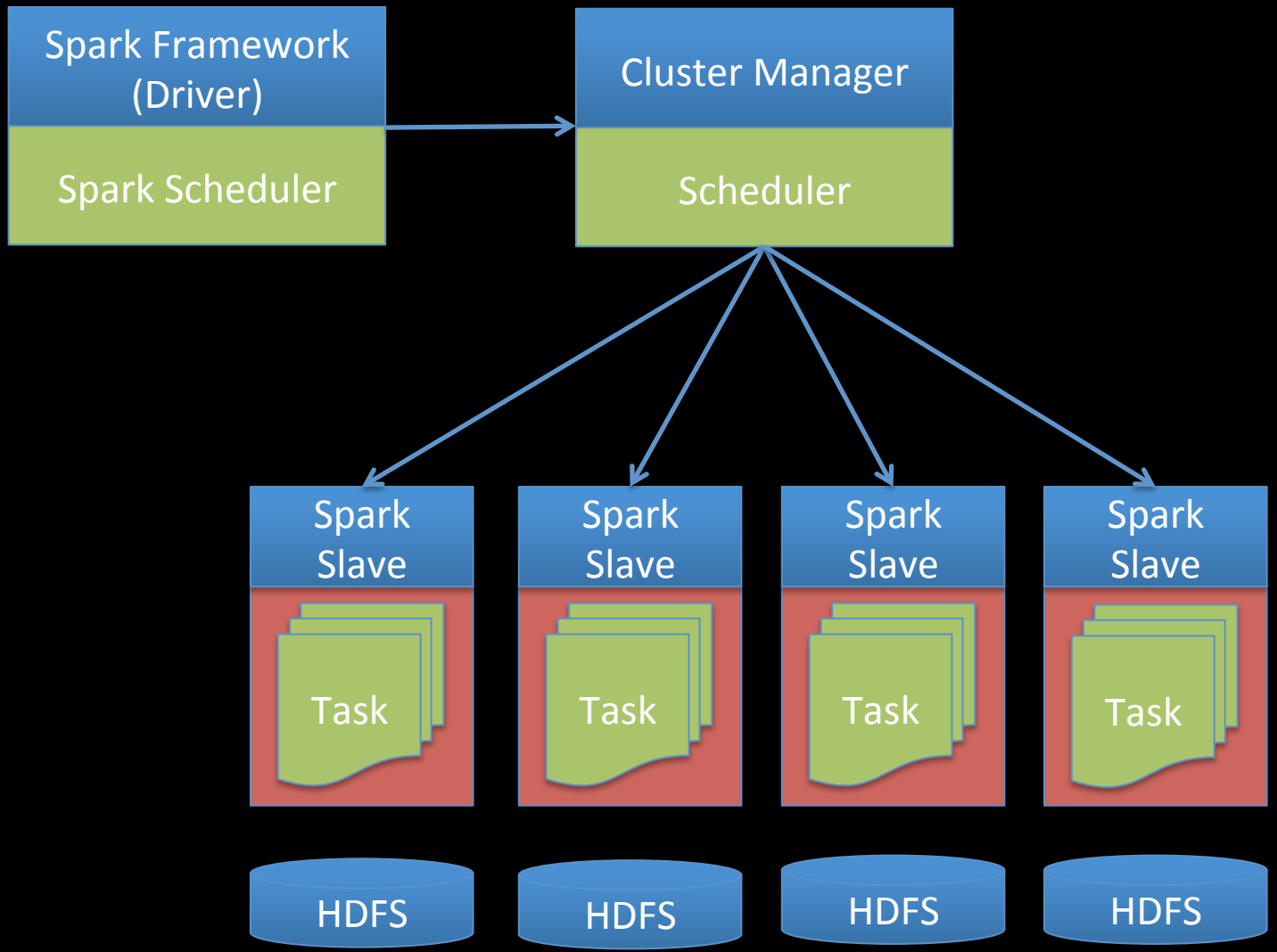
```
1.  object WordCount {
2.    def main (args: Array[String]){
3.      val driver = "spark://192.168.0.3:7077"
4.      val sc = new SparkContext(driver, "SparkWordCount")
5.      val numPartitions = 10
6.      val lines = sc.textFile("here"+"sometext.txt", numPartitions)
7.      val words = lines.flatMap (_.split(" "))
8.      val groupWords = words.groupBy { x => x}
9.      val wordCount = groupWords.map( x => (x._1,x._2.size))
10.     val result = wordCount.saveAsTextFile("there"+"wordcount")
11.   }
12. }
```

Control Number of Tasks

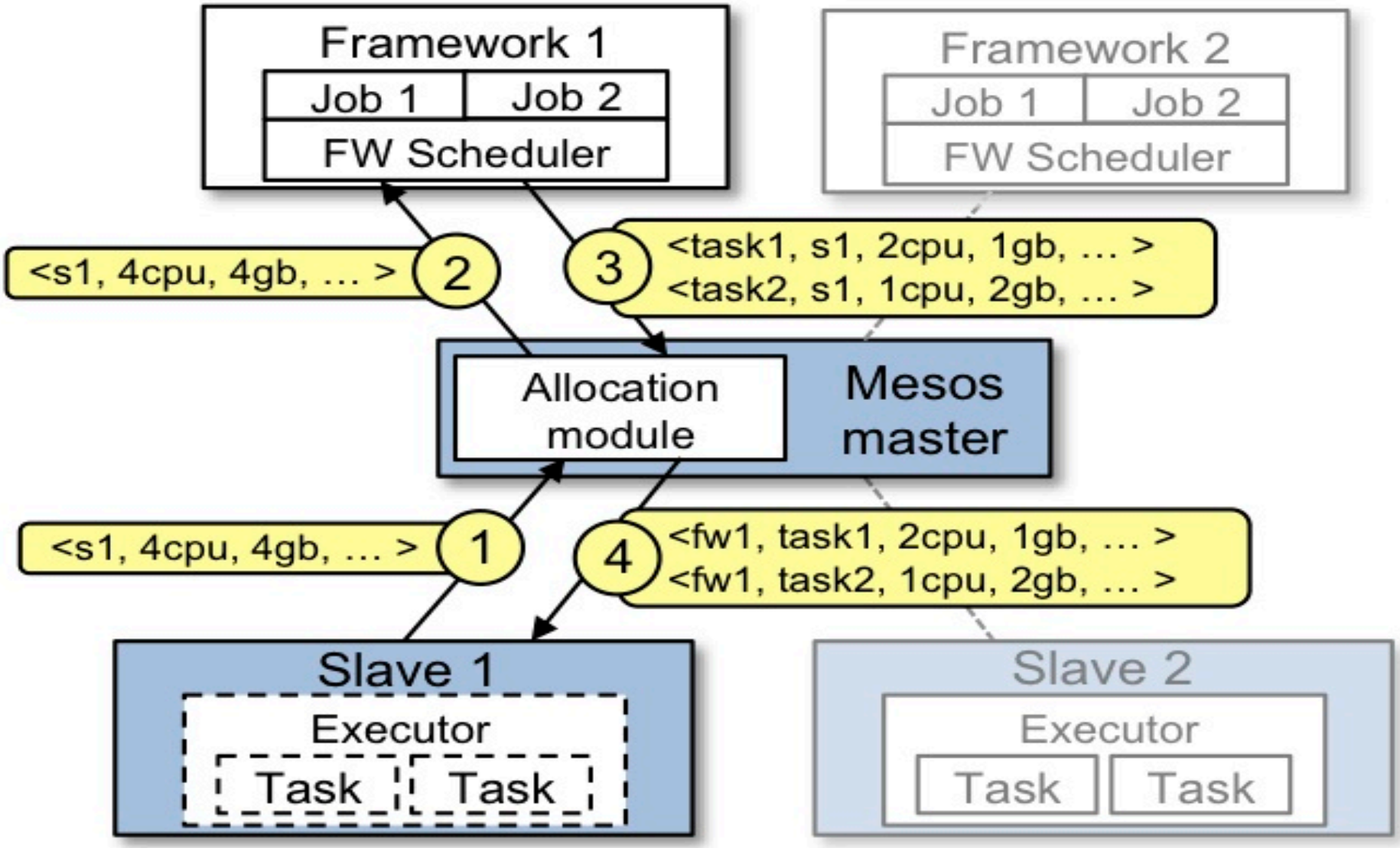
```
1.  object WordCount {
2.    def main (args: Array[String]){
3.      val driver = "spark://192.168.0.3:7077"
4.      val sc = new SparkContext(driver, "SparkWordCount")
5.      val numPartitions = 10
6.      val lines = sc.textFile("here"+"sometext.txt", numPartitions).coalesce(numPartitions)
7.      val words = lines.flatMap (_.split(" "))
8.      val groupWords = words.groupBy { x => x}
9.      val wordCount = groupWords.map( x => (x._1,x._2.size))
10.     val result = wordCount.saveAsTextFile("there"+"wordcount")
11.   }
12. }
```

How does the driver know about the resource information of the workers?

Spark Application Framework



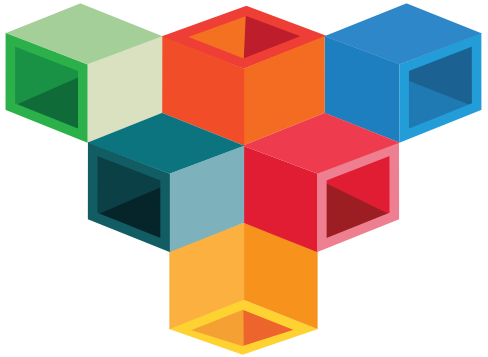
Resource Allocation Example



Resource Allocation Example

- For example, how can a framework achieve data locality without MESOS knowing which nodes store the data required by the framework?
  - MESOS answers these questions by simply giving frameworks the ability to **reject** offers. A framework will reject the offers that do not satisfy its constraints and accept the ones that do.

What is Tachyon?



TACHYON

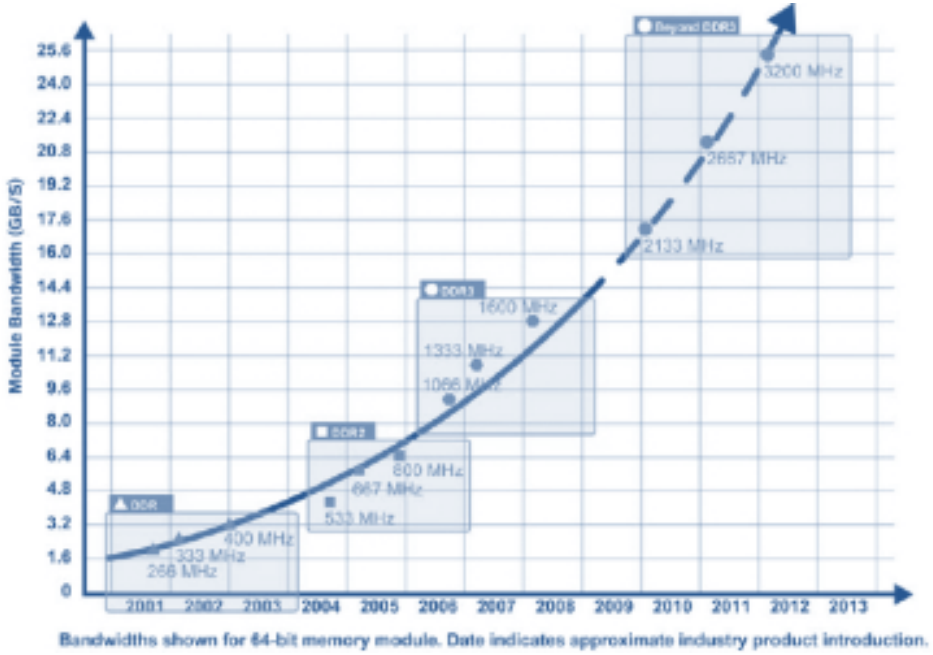
Open Source Memory-Centric Distributed Storage System

TACHYON
NEXUS

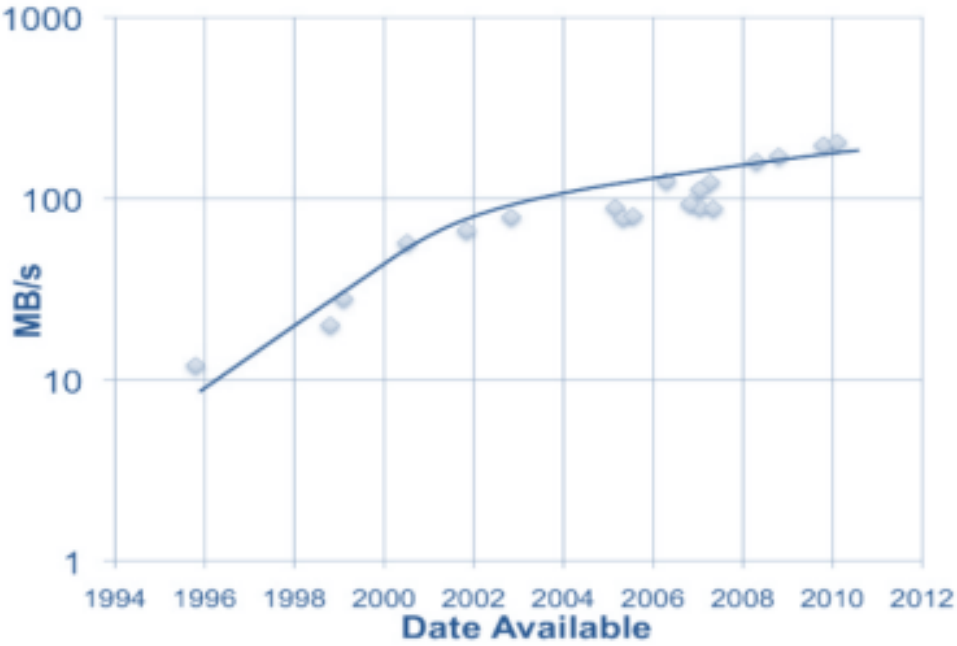
Why Use Tachyon?

Performance Trend: Memory is Fast

- RAM throughput increasing **exponentially**



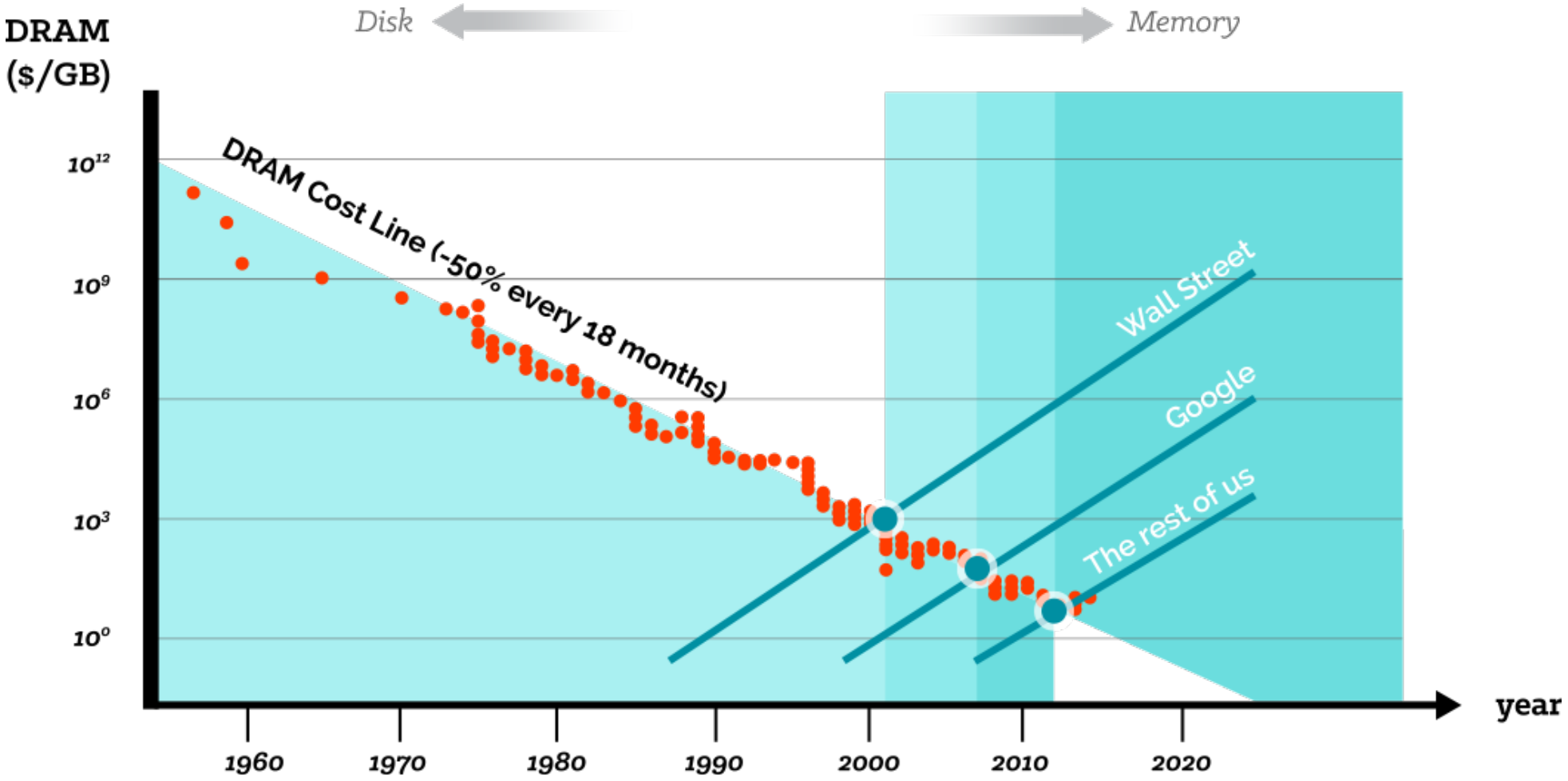
- Disk throughput increasing **slowly**



Memory-locality is important!

TACHYON NEXUS

Price Trend: Memory is Cheaper



source: jcmit.com

TACHYON
NEXUS

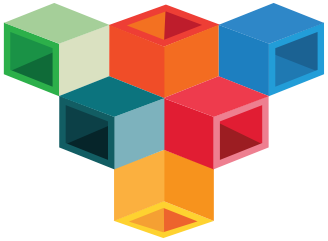
These Memory Trends are Realized By Many…

TACHYON
NEXUS

Is the Problem Solved?

Missing a Solution for the Storage Layer

TACHYON
NEXUS



TACHYON

enables reliable data sharing at memory-speed within and across computation frameworks/jobs

TACHYON
NEXUS

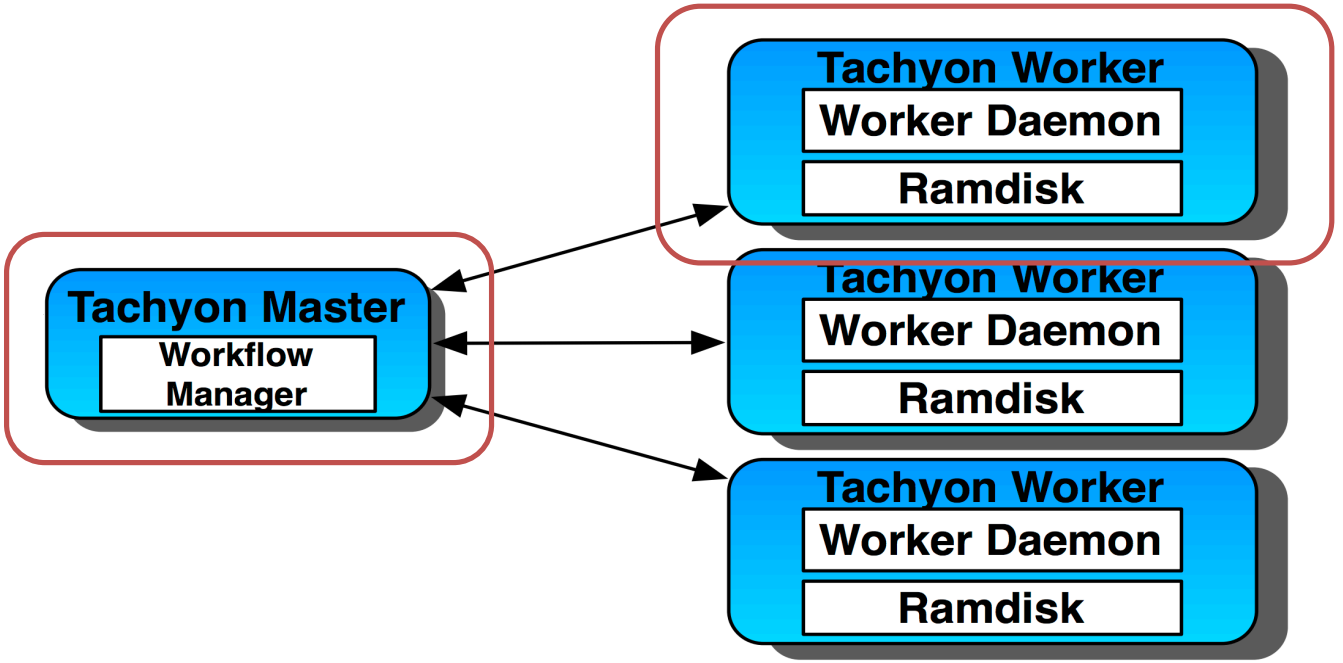
How Does Tachyon Work?

Memory-Centric Storage Architecture

Lineage in Storage Layer

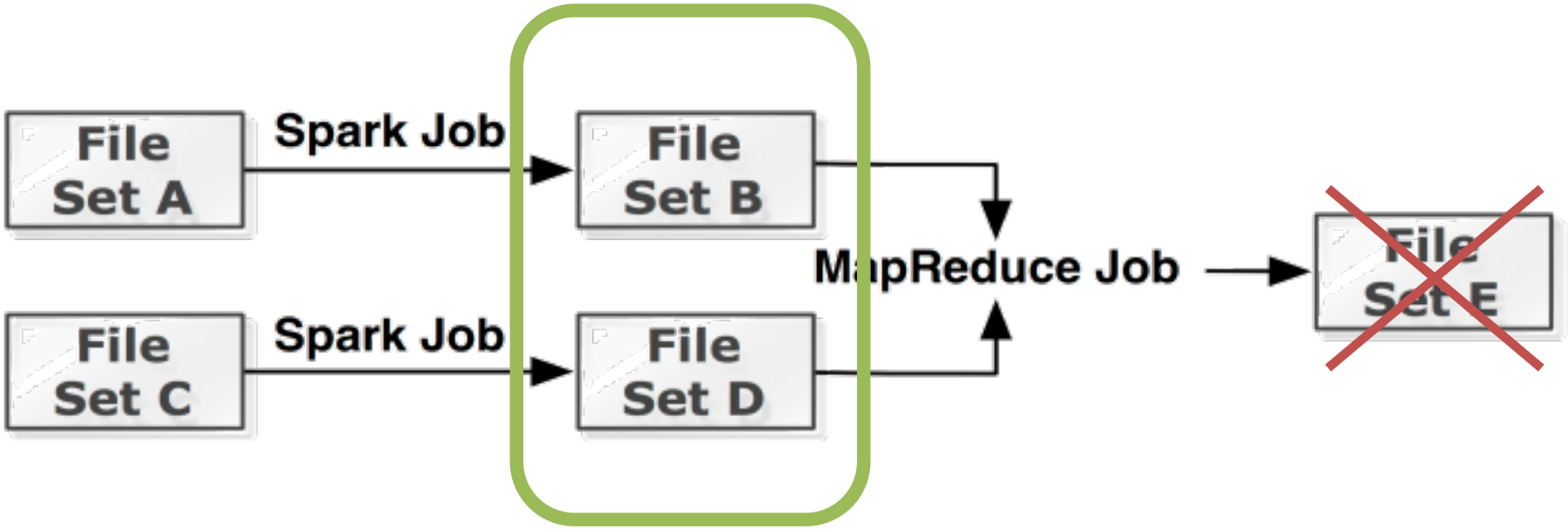
TACHYON
NEXUS

Tachyon Memory-Centric Architecture



TACHYON
NEXUS

Lineage in Tachyon



Fast and general engine for large-scale data processing

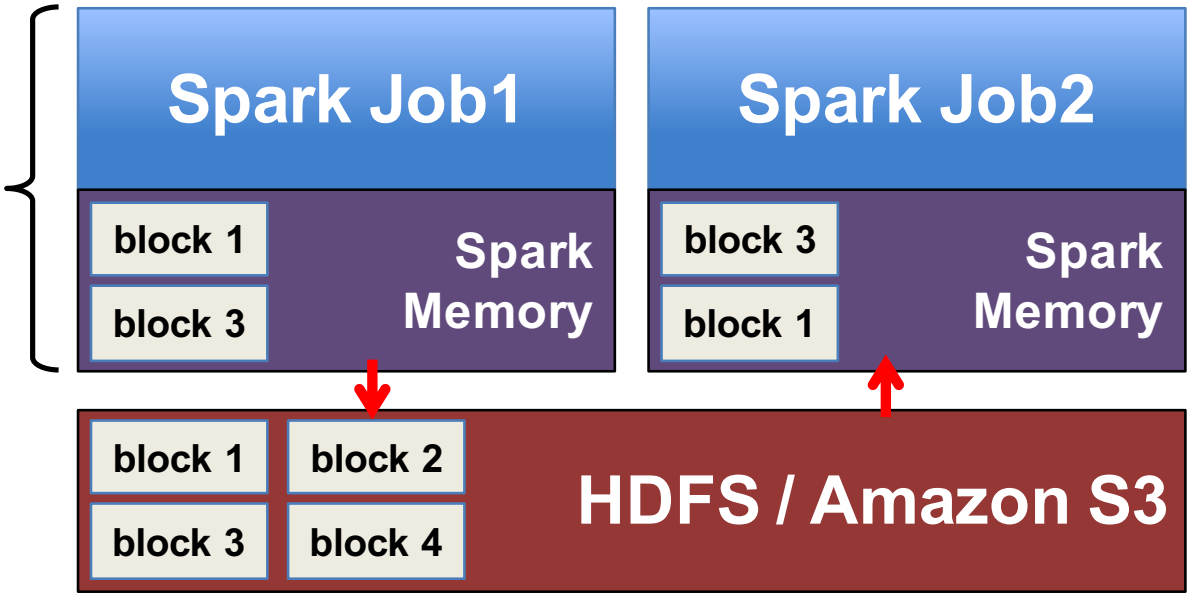
What are some potential issues?

TACHYON
NEXUS

Issue 1

Data Sharing bottleneck in analytics pipeline: Slow writes to disk

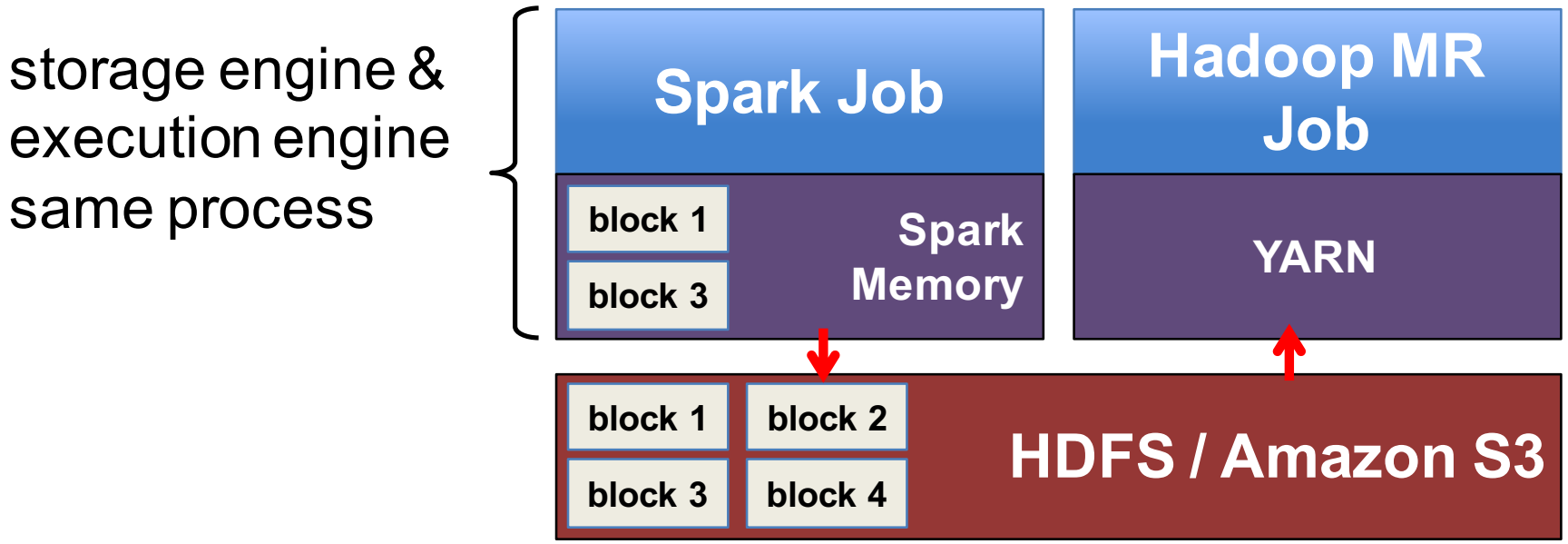
storage engine &
execution engine
same process



TACHYON
NEXUS

Issue 1

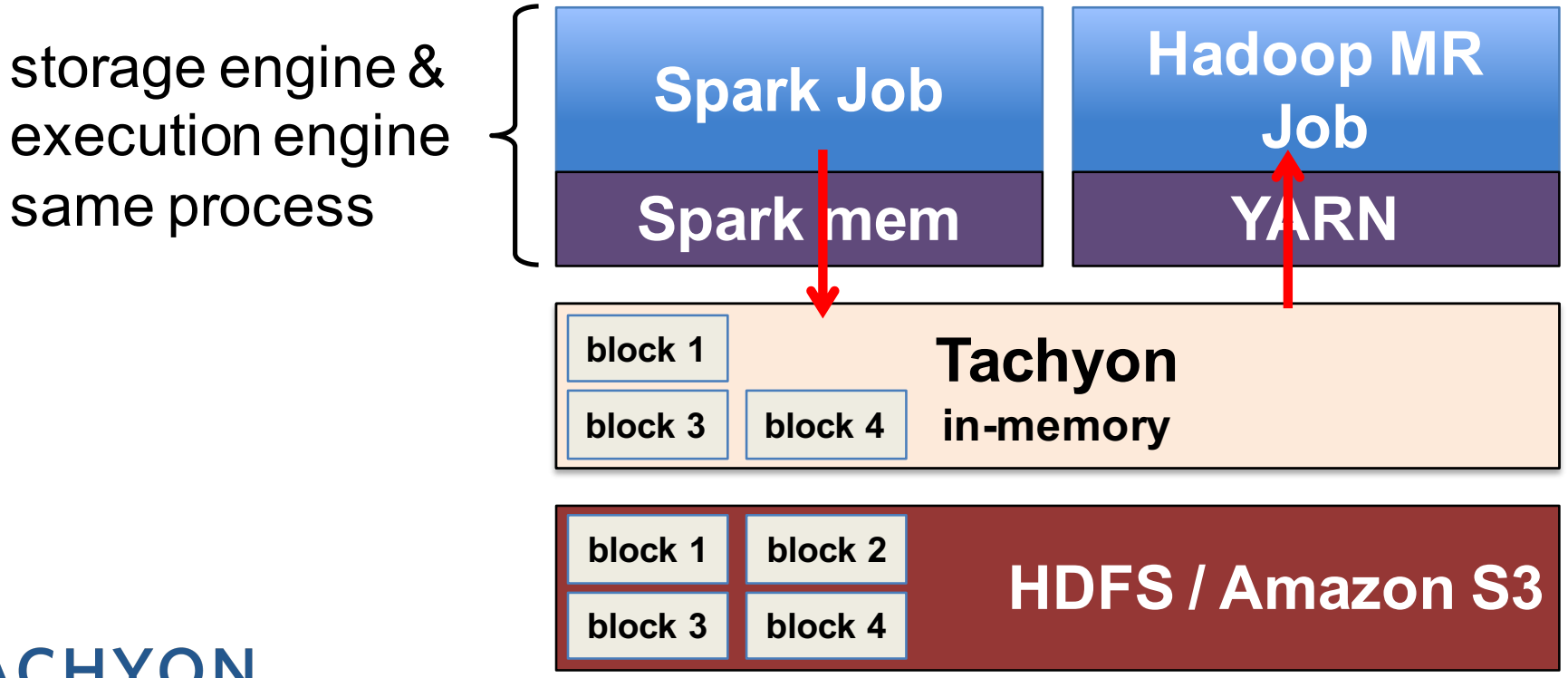
Data Sharing bottleneck in analytics pipeline: Slow writes to disk



TACHYON NEXUS

Issue 1 resolved with Tachyon

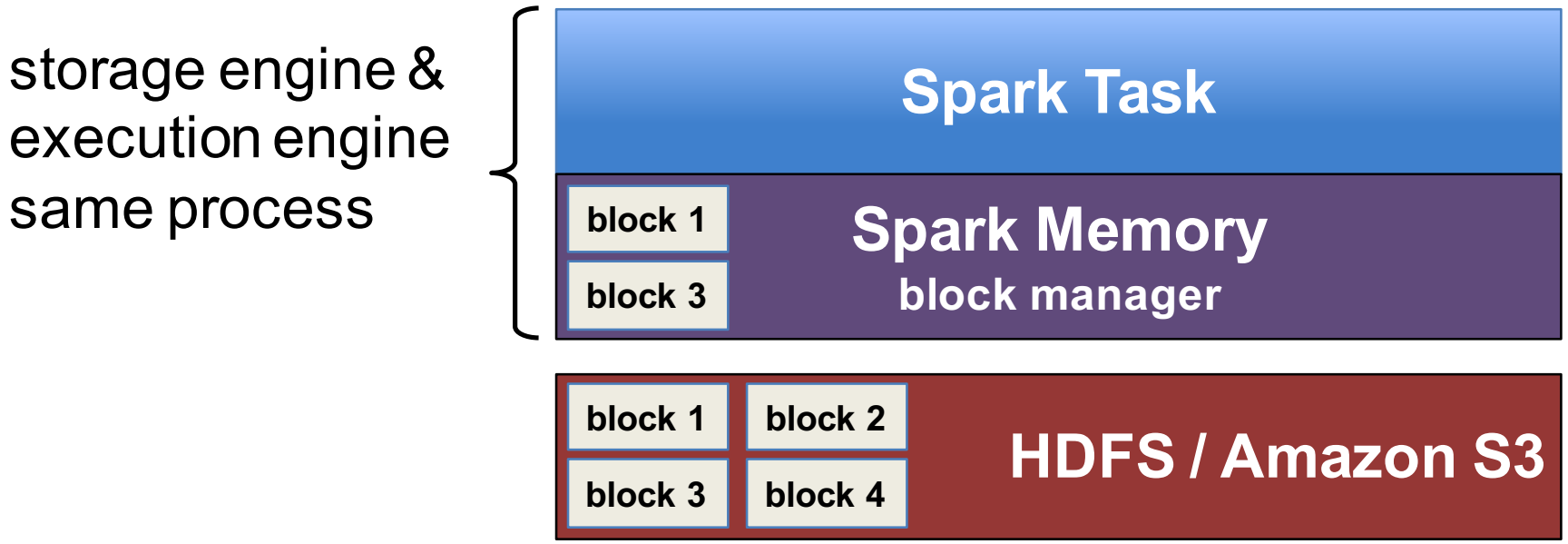
Memory-speed data sharing among different jobs and different frameworks



TACHYON
NEXUS

Issue 2

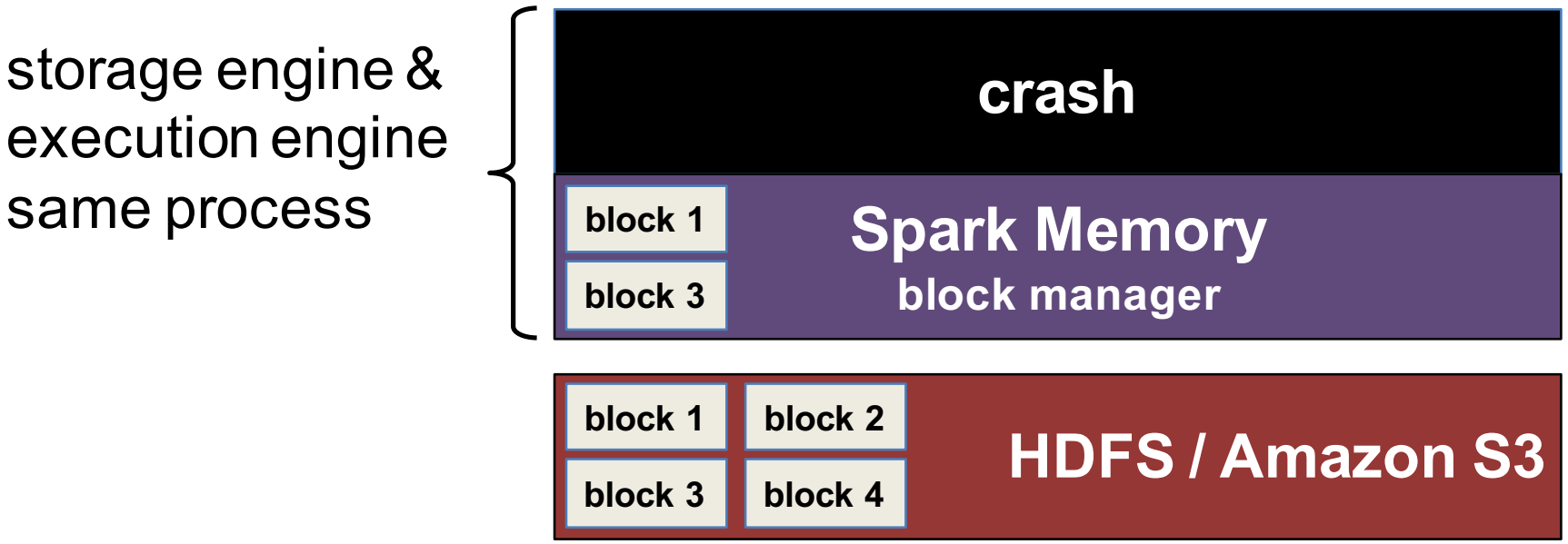
In-Memory data loss when computation crashes



TACHYON
NEXUS

Issue 2

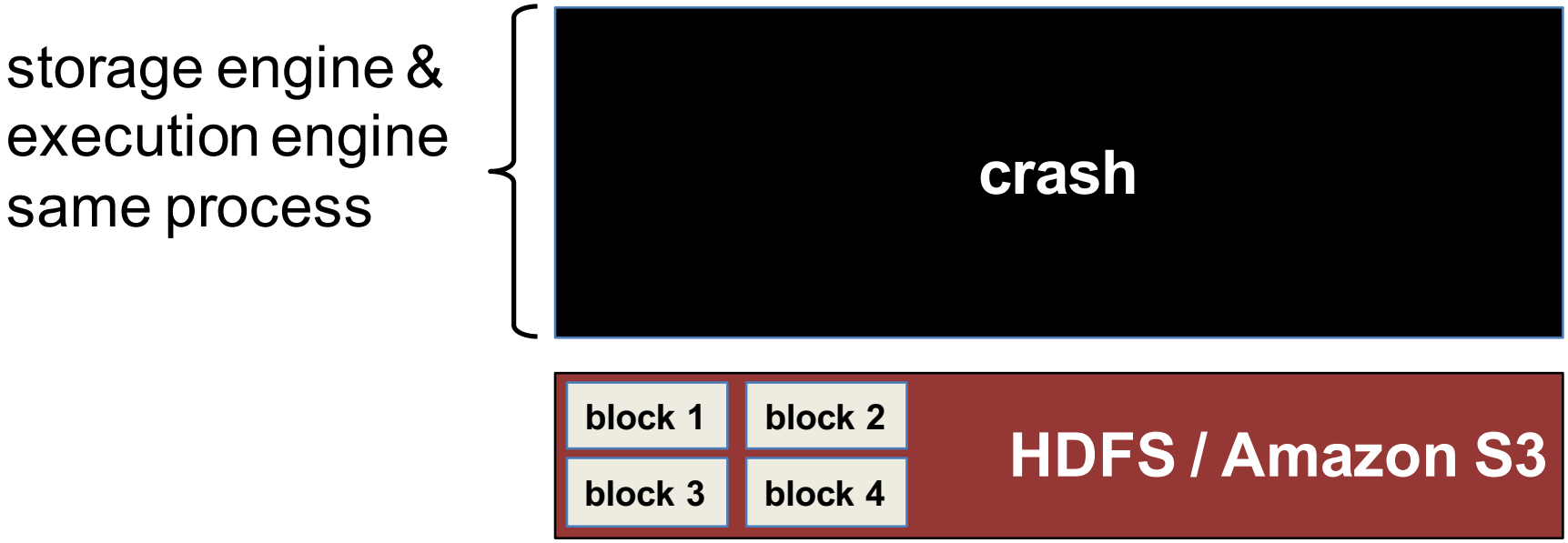
In-Memory data loss when computation crashes



TACHYON
NEXUS

Issue 2

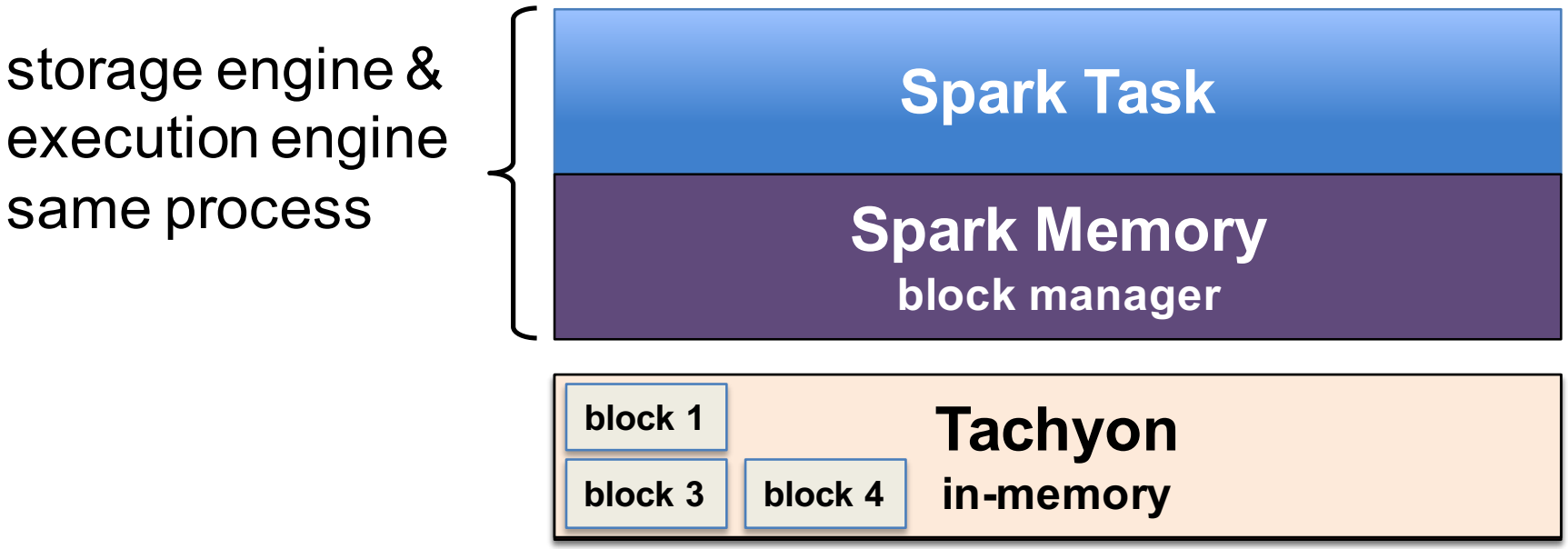
In-Memory data loss when computation crashes



TACHYON
NEXUS

Issue 2 resolved with Tachyon

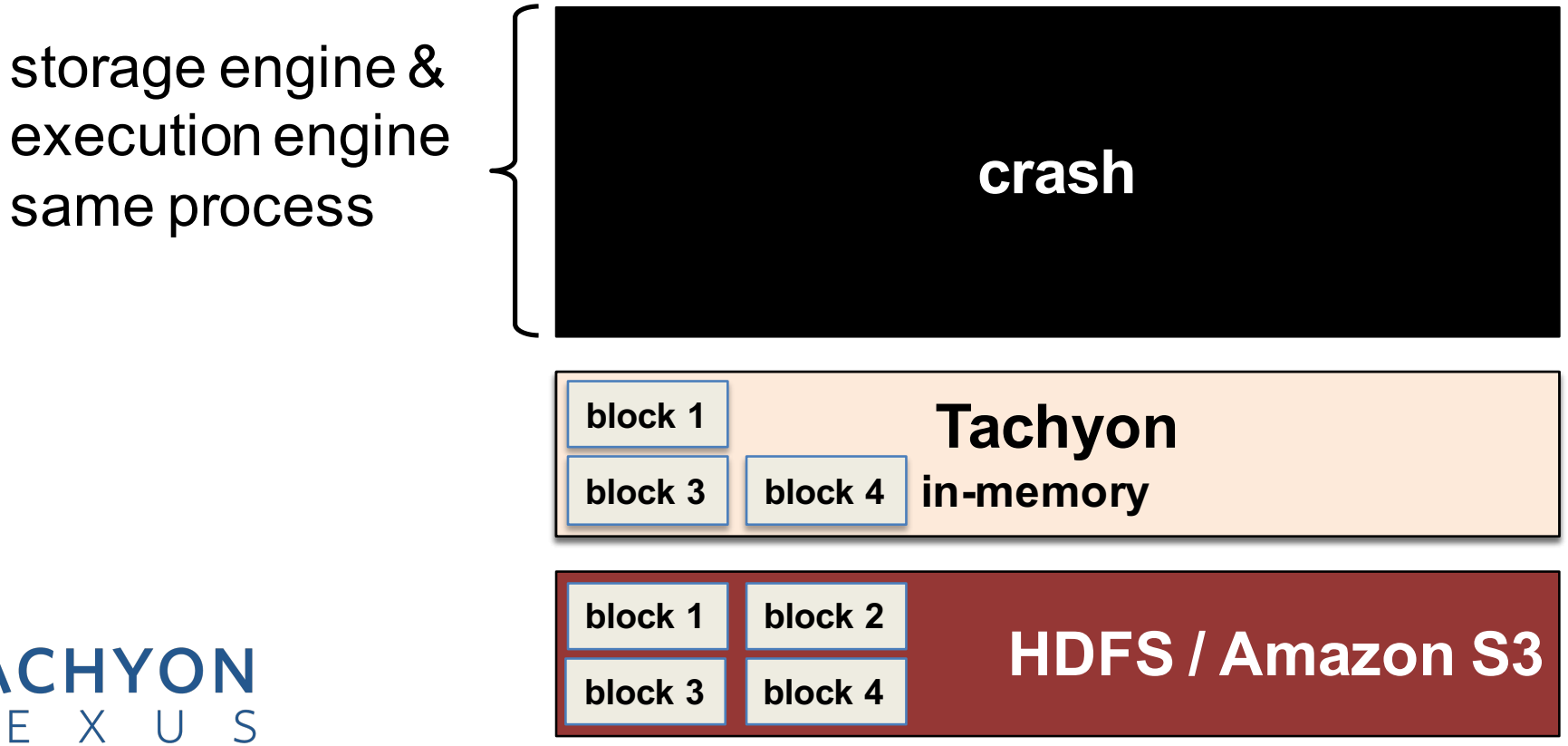
Keep in-memory data safe, even when computation crashes



TACHYON
NEXUS

Issue 2 resolved with Tachyon

Keep in-memory data safe, even when computation crashes

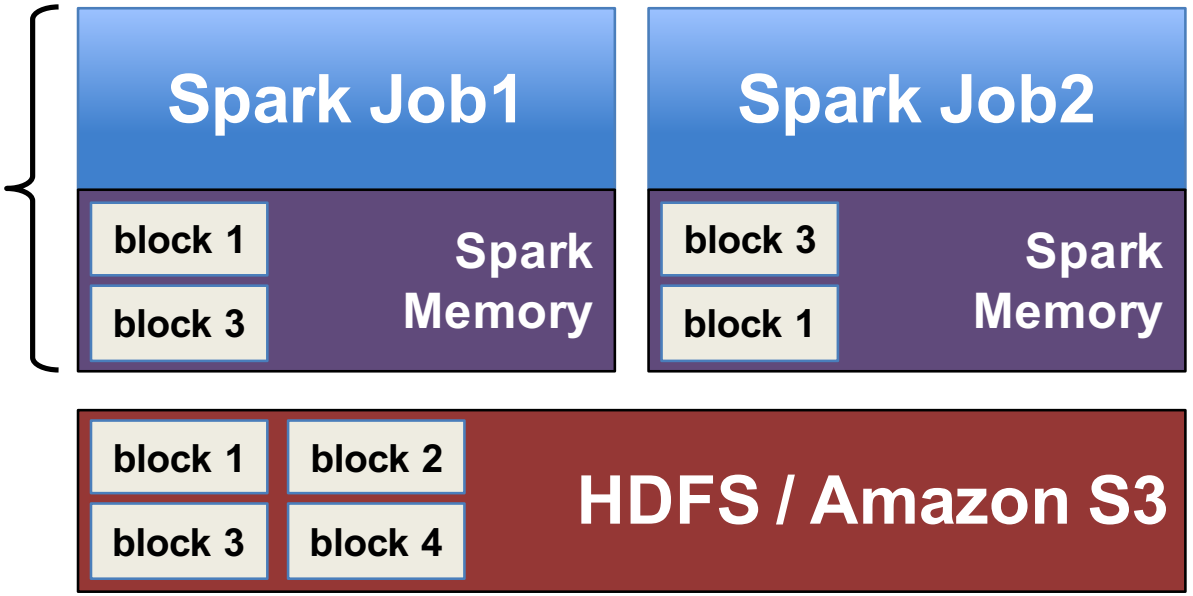


TACHYON
NEXUS

Issue 3

In-memory Data Duplication & Java Garbage Collection

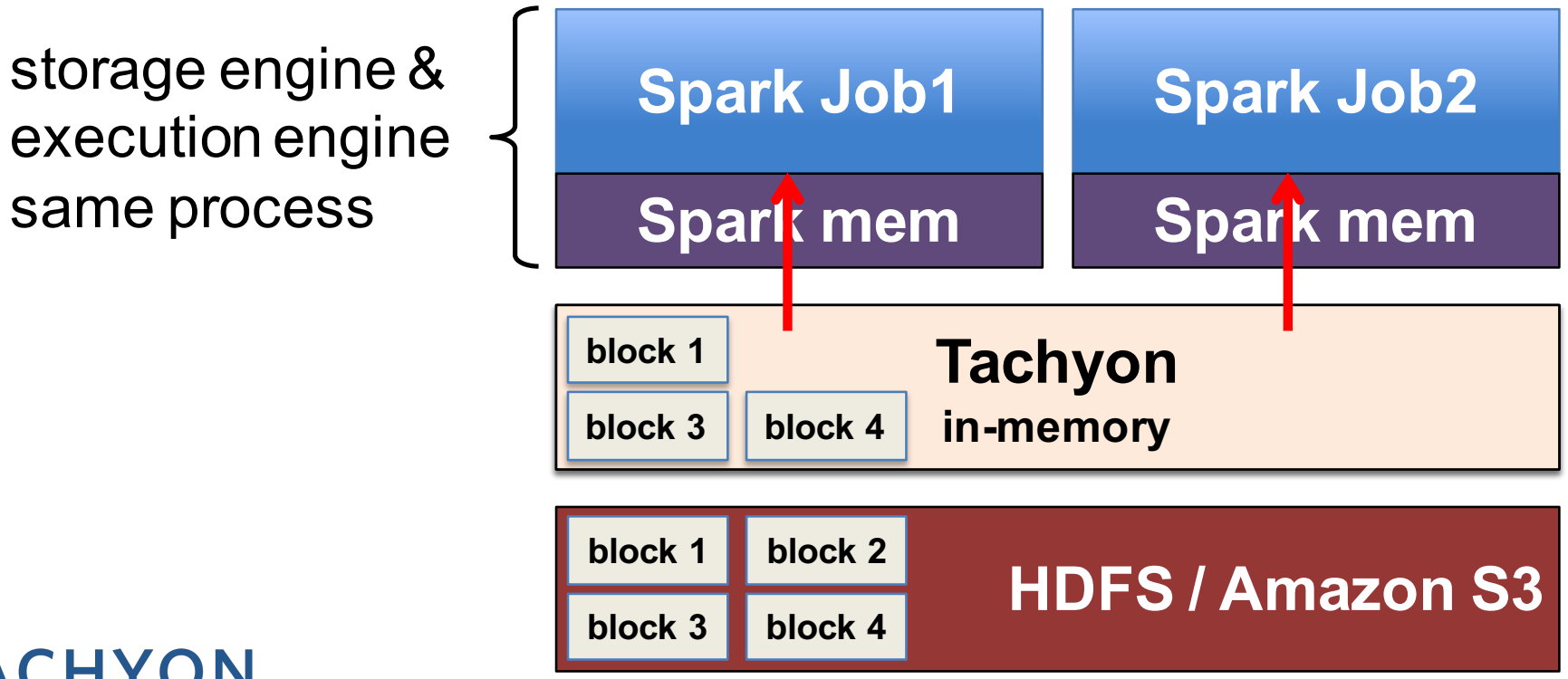
storage engine &
execution engine
same process



TACHYON
NEXUS

Issue 3 resolved with Tachyon

No in-memory data duplication, much less GC



TACHYON
NEXUS

Tachyon Use Case: Baidu

- Framework: **SparkSQL**
- Under Storage: **Baidu's File System**
- Tachyon Storage Media: **MEM + HDD**
- **100+** Tachyon nodes
- **1PB+** Tachyon managed storage
- **30x** Performance Improvement

Tachyon Use Case: An Oil Company

- Framework: **Spark**
- Under Storage: **GlusterFS**
- Tachyon Storage Media: **MEM** only
- Analyzing data in traditional storage

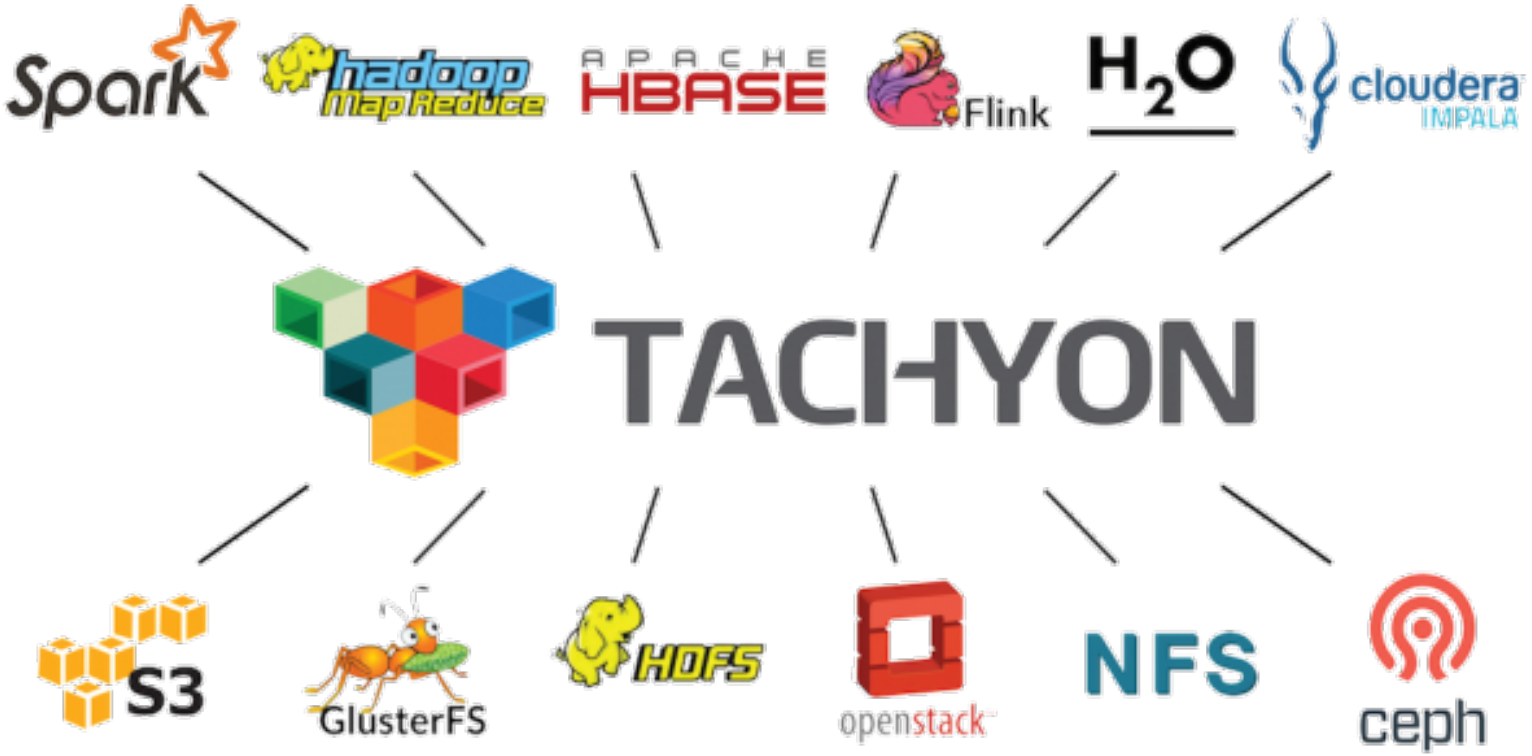
TACHYON
NEXUS

Tachyon Use Case: A SAAS Company

- Framework: **Spark**
- Under Storage: **S3**
- Tachyon Storage Media: **SSD** only
- Elastic Tachyon deployment

TACHYON
NEXUS

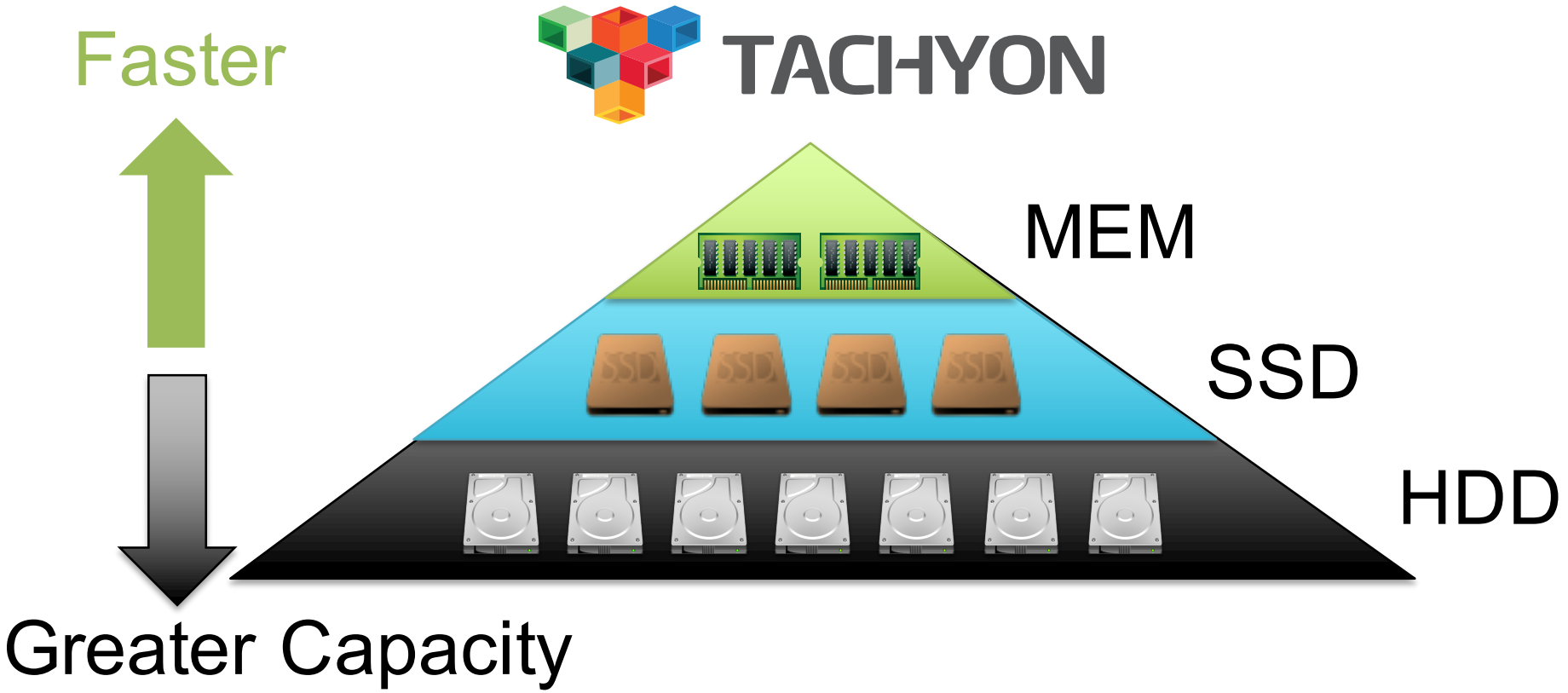
1. Growing Ecosystem



Use different frameworks to enable workloads on different storage

TACHYON NEXUS

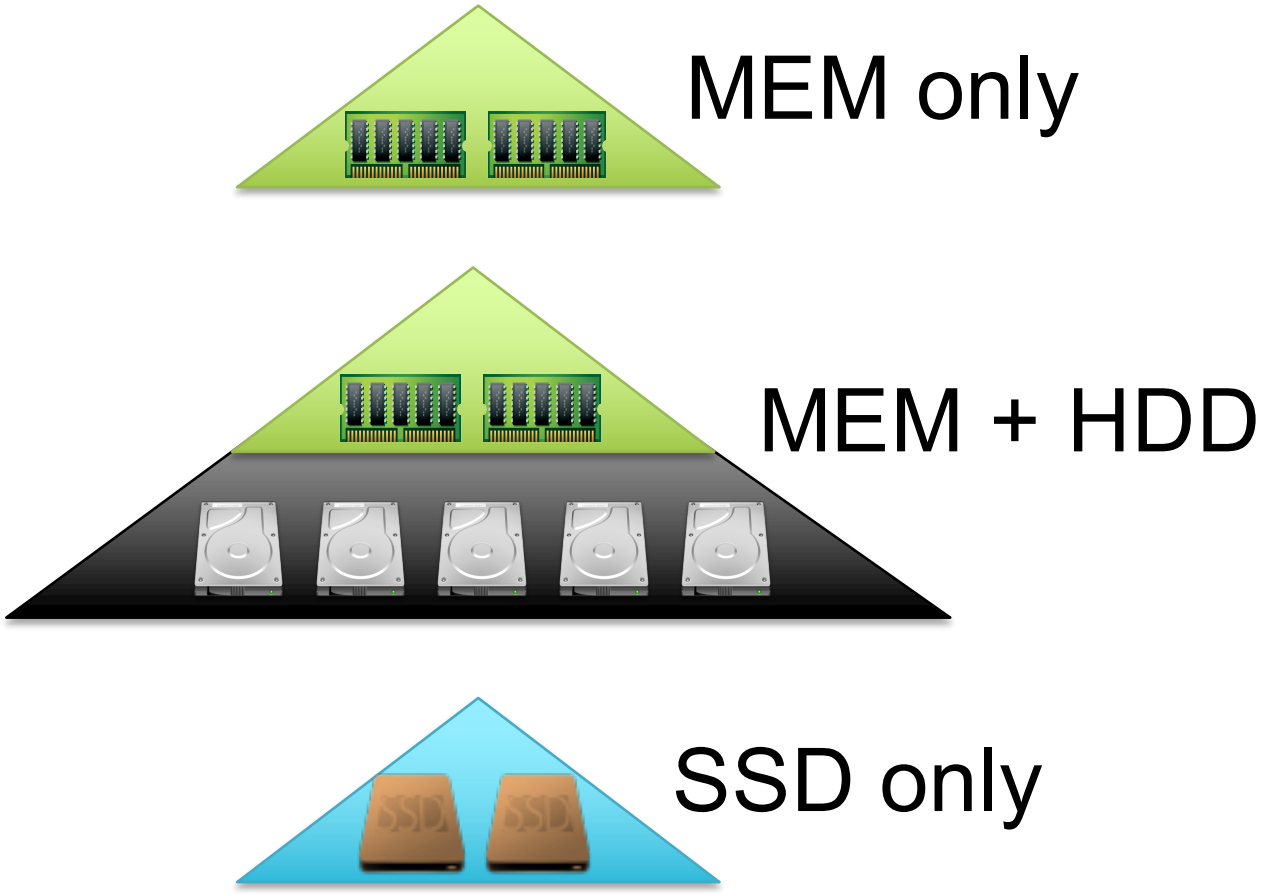
2. Tiered Storage



Tachyon manages more than DRAM

TACHYON
NEXUS

2. Tiered Storage



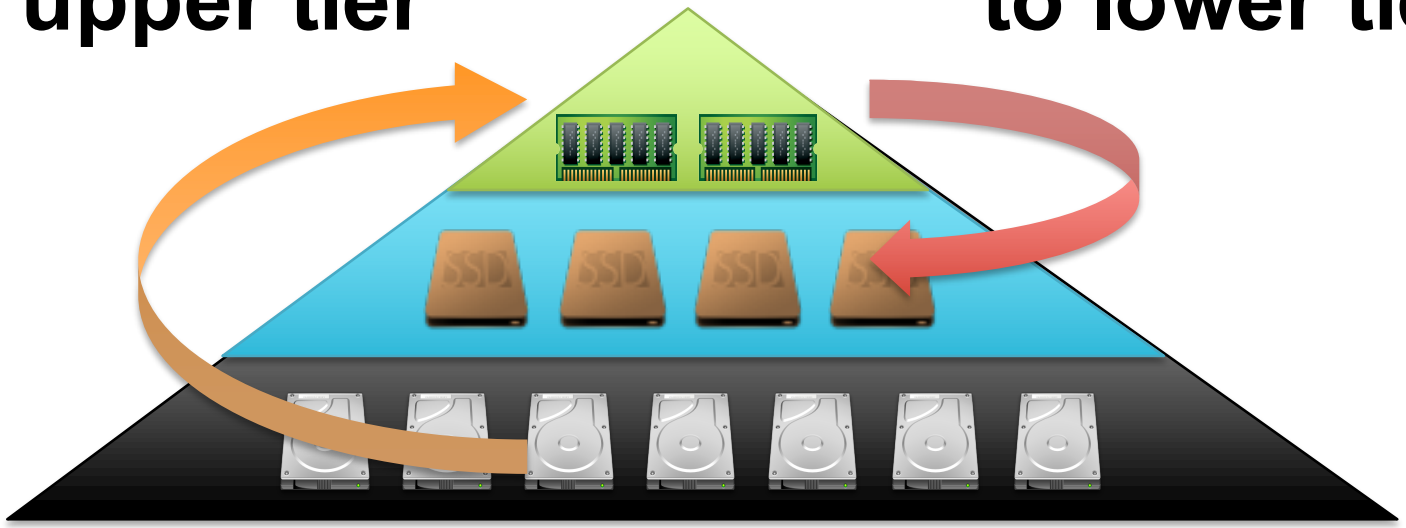
Configurable storage tiers

TACHYON
NEXUS

3. Pluggable Data Management Policy

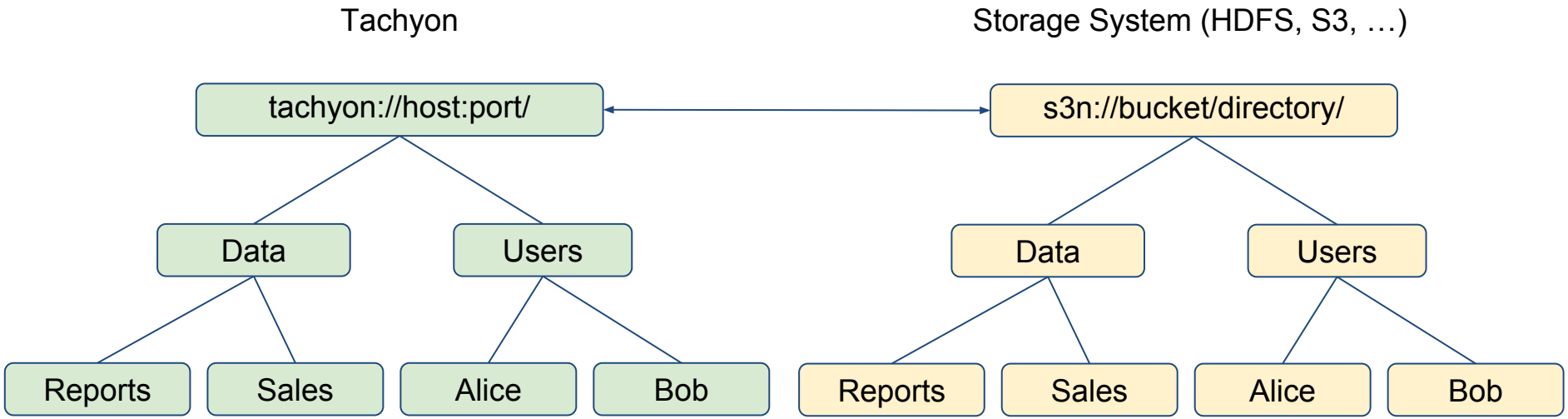
Promote hot data to upper tier

Evict stale data to lower tier



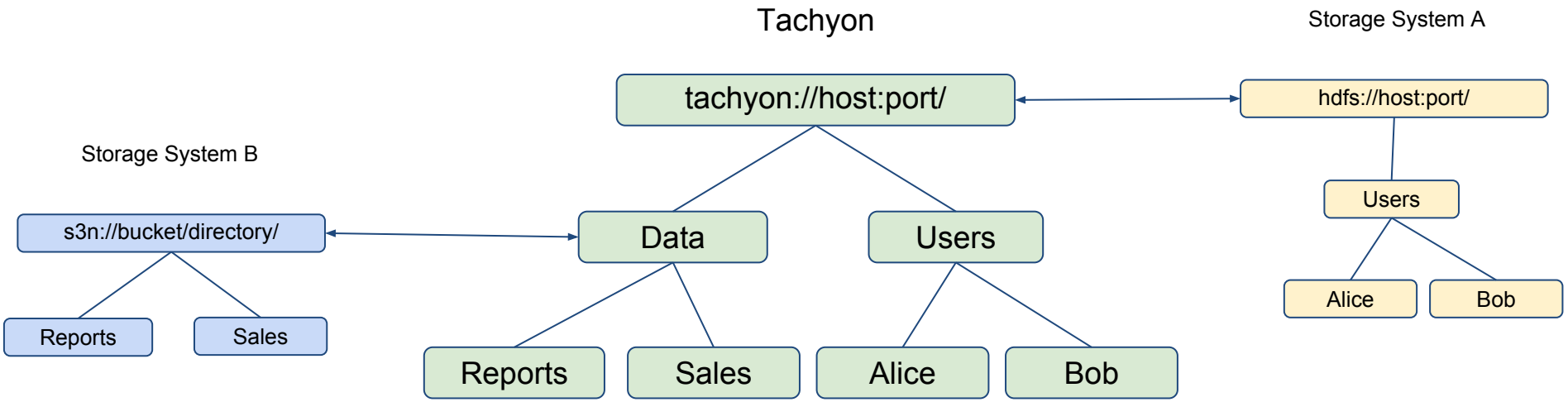
TACHYON
NEXUS

4. Transparent Naming



- Persisted Tachyon files are mapped to under storage
- Tachyon paths are preserved in under storage

5. Unified Namespace



- Unified namespace for multiple storage systems
- Share data across storage systems
- On-the-fly mounting/unmounting