

Top 5 Mistakes when writing Spark applications

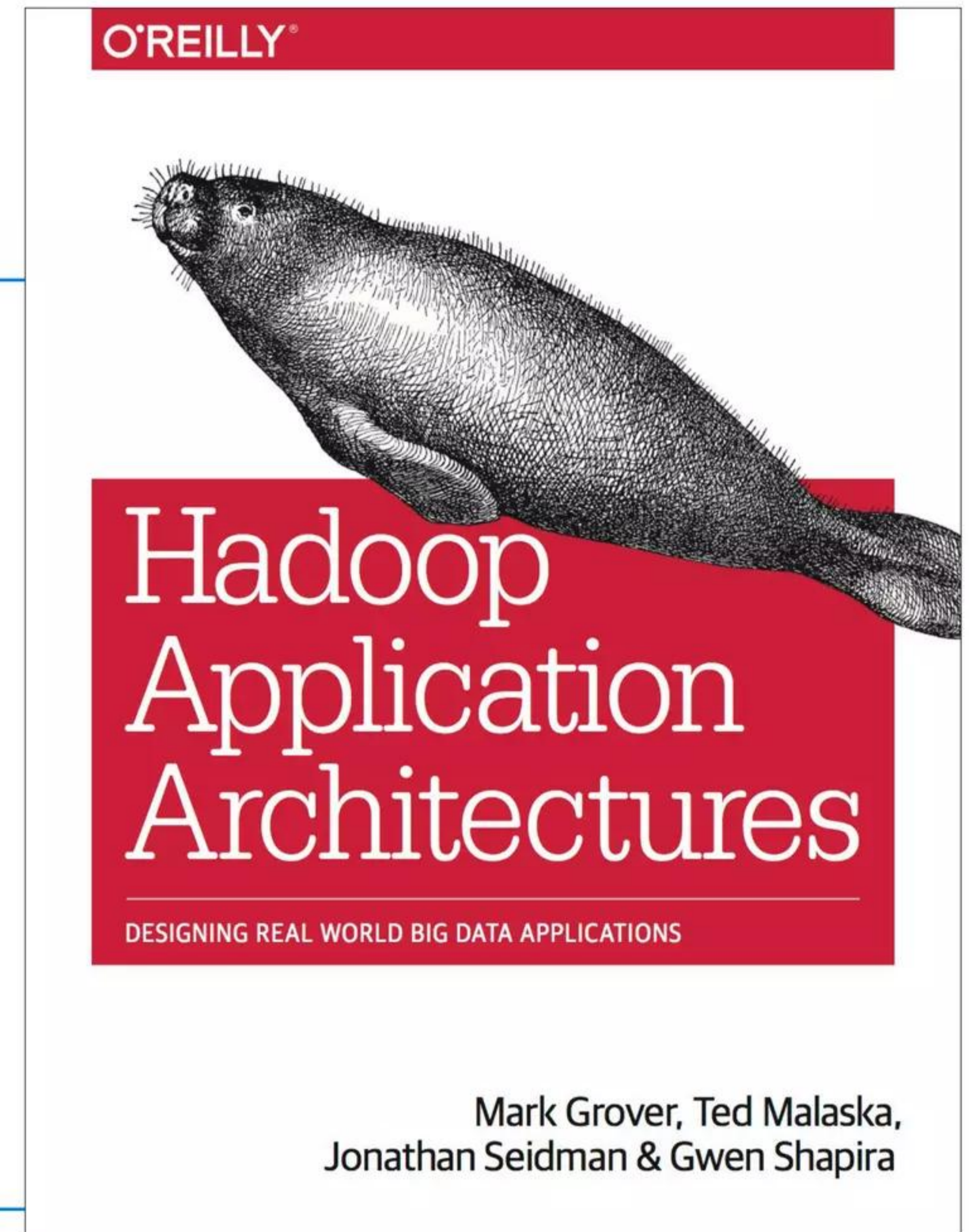
Mark Grover | @mark_grover | Software Engineer
Ted Malaska | @TedMalaska | Principal Solutions Architect
tiny.cloudera.com/spark-mistakes



SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY

About the book

- [@hadooparchbook](#)
- [hadooparchitecturebook.com](#)
- [github.com/hadooparchitecturebook](#)
- [slideshare.com/hadooparchbook](#)



Mistakes people make

when using Spark



SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY

Mistakes ~~people~~ we made

when using Spark



SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY

Mistake # 1



SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY

Executors, cores, memory !?!



- 6 Nodes
- 16 cores each
- 64 GB of RAM each



Decisions, decisions, decisions

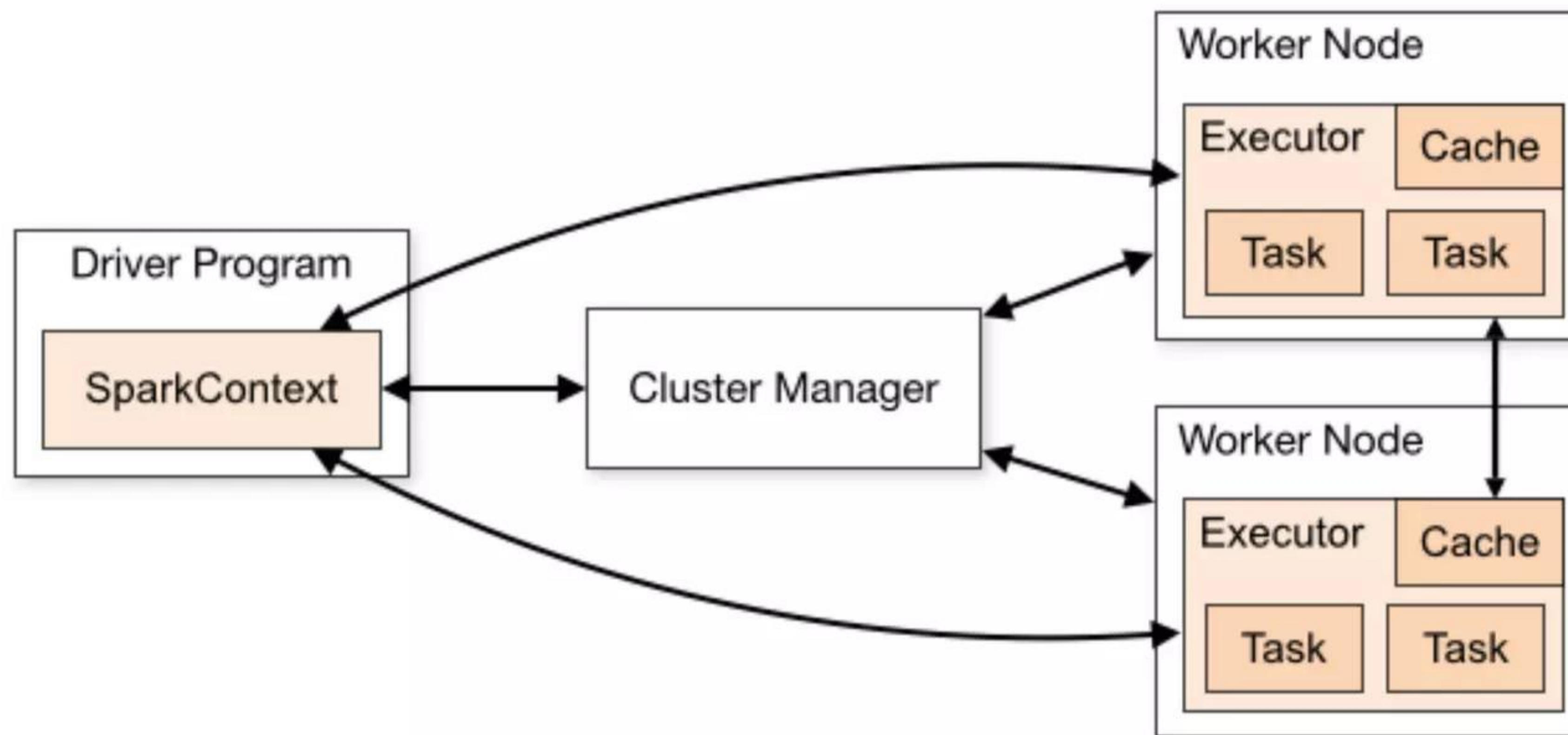
- 6 nodes
- 16 cores each
- 64 GB of RAM



- Number of executors (`--num-executors`)
- Cores for each executor (`--executor-cores`)
- Memory for each executor (`--executor-memory`)



Spark Architecture recap



Answer #1 – Most granular

- Have smallest sized executors as possible
- 1 core each
- Total of $16 \times 6 = 96$ cores
- 96 executors
- $64/16 = 4$ GB per executor (per node)



Answer #1 – Most granular

- Have smallest sized executors as possible
- 1 core each
- Total of $16 \times 6 = 96$ cores
- 96 executors
- $64/16 = 4$ GB per executor (per node)



Why?

- Not using benefits of running multiple tasks in same JVM



Answer #2 – Least granular

- 6 executors
- 64 GB memory each
- 16 cores each



Answer #2 – Least granular

- 6 executors
- 64 GB memory each
- 16 cores each



Why?

- Need to leave some memory overhead for OS/Hadoop daemons



Answer #3 – with overhead

- 6 executors
- 63 GB memory each
- 15 cores each

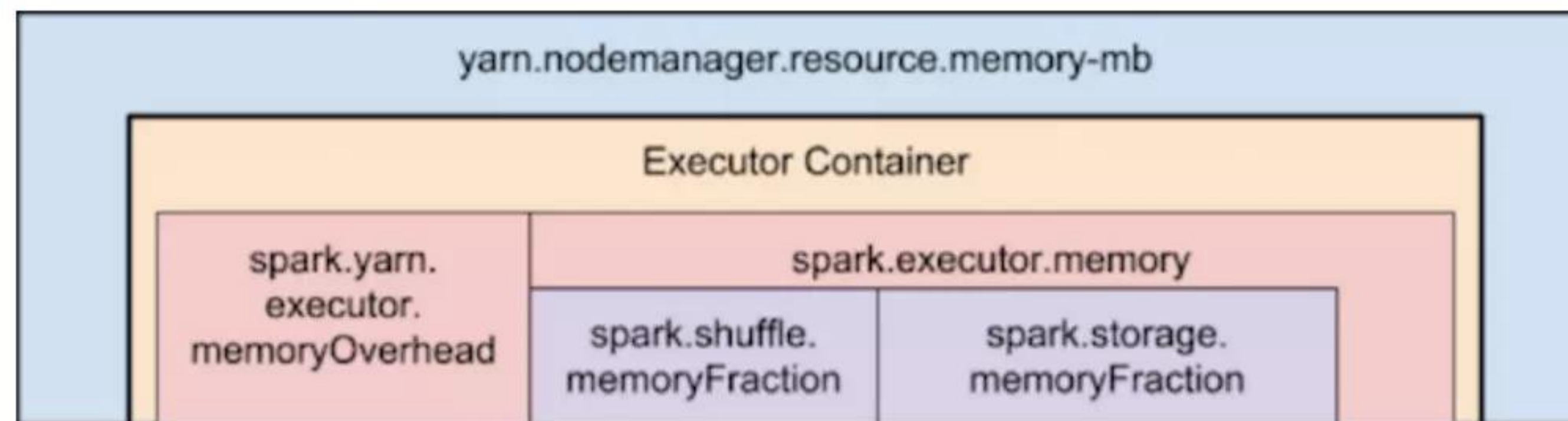


Answer #3 – with overhead

- 6 executors
- 63 GB memory each
- 15 cores each



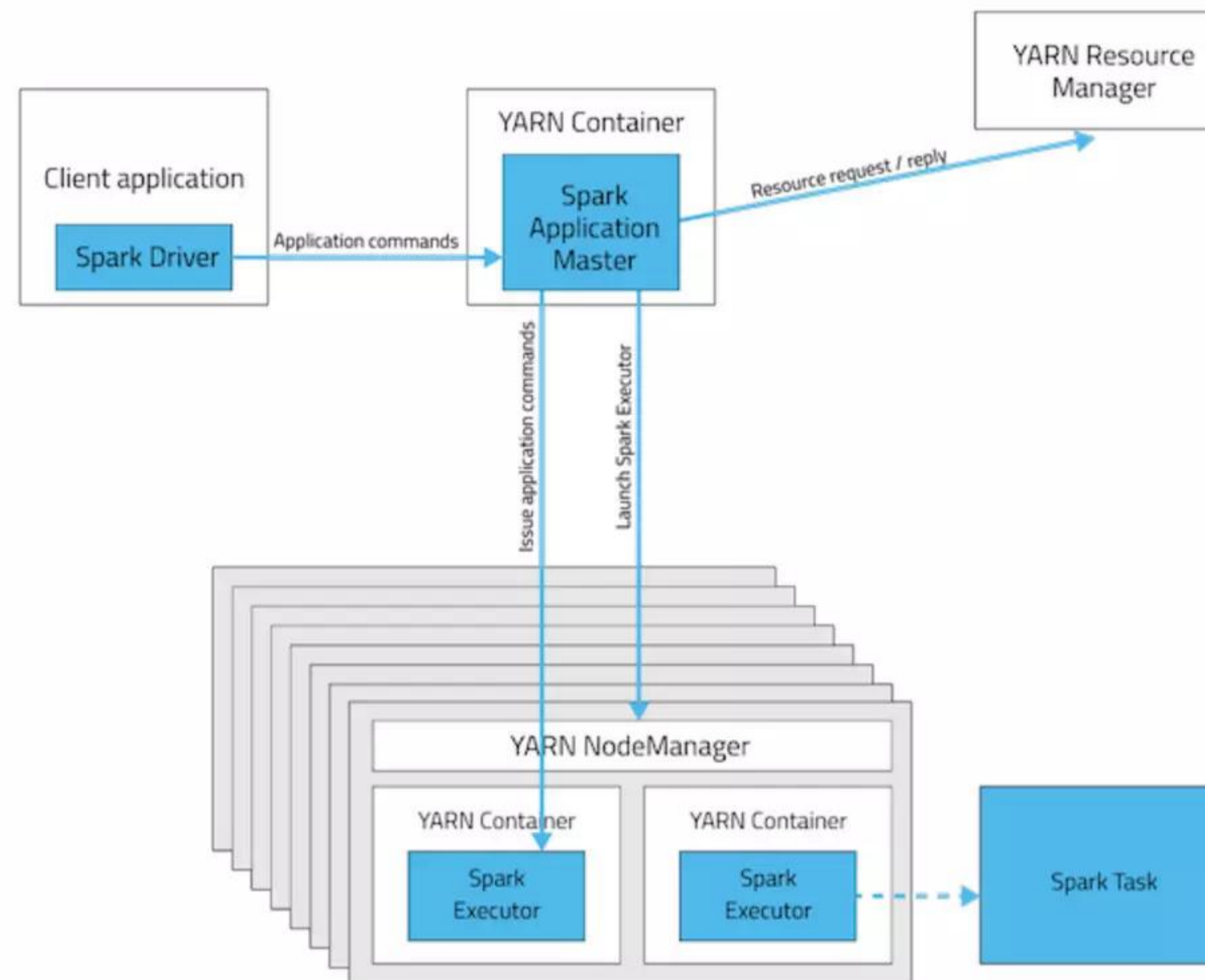
Spark on YARN – Memory usage



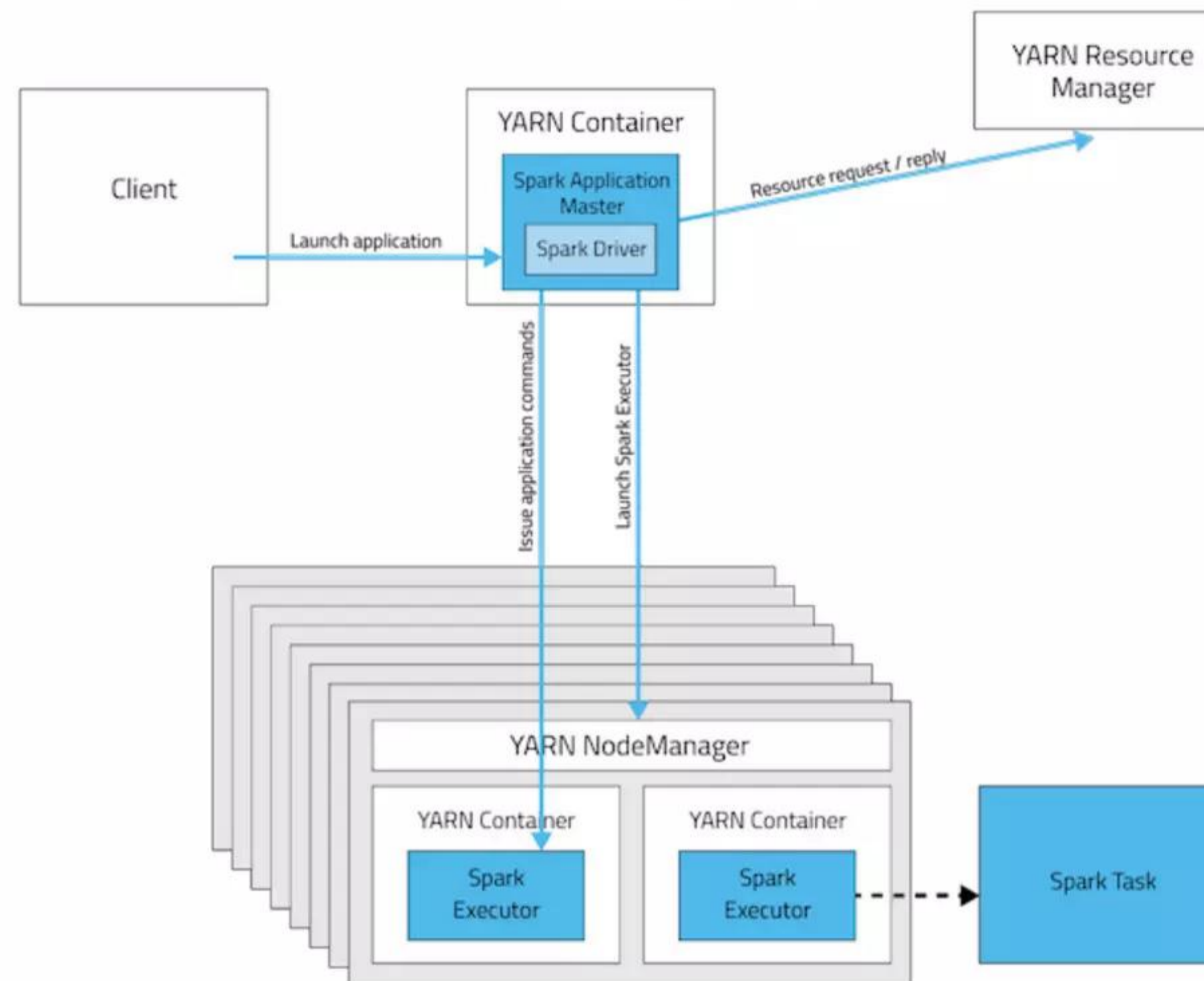
- --executor-memory controls the heap size
- Need some overhead (controlled by spark.yarn.executor.memory.overhead) for off heap memory
 - Default is $\max(384\text{MB}, .07 * \text{spark.executor.memory})$



YARN AM needs a core: Client mode



YARN AM needs a core: Cluster mode



HDFS Throughput

- 15 cores per executor can lead to bad HDFS I/O throughput.
- Best is to keep under 5 cores per executor



Calculations

- 5 cores per executor
 - For max HDFS throughput
- Cluster has $6 * 15 = 90$ cores in total (after taking out Hadoop/Yarn daemon cores)
- $90 \text{ cores} / 5 \text{ cores/executor} = 18$ executors
- 1 executor for AM => 17 executors
- Each node has 3 executors
- $63 \text{ GB} / 3 = 21 \text{ GB}$, $21 \times (1 - 0.07) \sim 19 \text{ GB}$ (counting off heap overhead)



Correct answer

- 17 executors
- 19 GB memory each
- 5 cores each

** Not etched in stone*



Read more

- From a great blog post on this topic by Sandy Ryza:

<http://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-2/>



Mistake # 2



SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY

Application failure

```
15/04/16 14:13:03 WARN scheduler.TaskSetManager: Lost task 19.0 in
stage 6.0 (TID 120, 10.215.149.47):
java.lang.IllegalArgumentException: Size exceeds Integer.MAX_VALUE
at sun.nio.ch.FileChannelImpl.map(FileChannelImpl.java:828) at
org.apache.spark.storage.DiskStore.getBytes(DiskStore.scala:123) at
org.apache.spark.storage.DiskStore.getBytes(DiskStore.scala:132) at
org.apache.spark.storage.BlockManager.doGetLocal(BlockManager.scala:51
7) at
org.apache.spark.storage.BlockManager.getLocal(BlockManager.scala:432)
at org.apache.spark.storage.BlockManager.get(BlockManager.scala:618)
at
org.apache.spark.CacheManager.putInBlockManager(CacheManager.scala:146
) at org.apache.spark.CacheManager.getOrCompute(CacheManager.scala:70)
```



Why?

- No Spark shuffle block can be greater than 2 GB

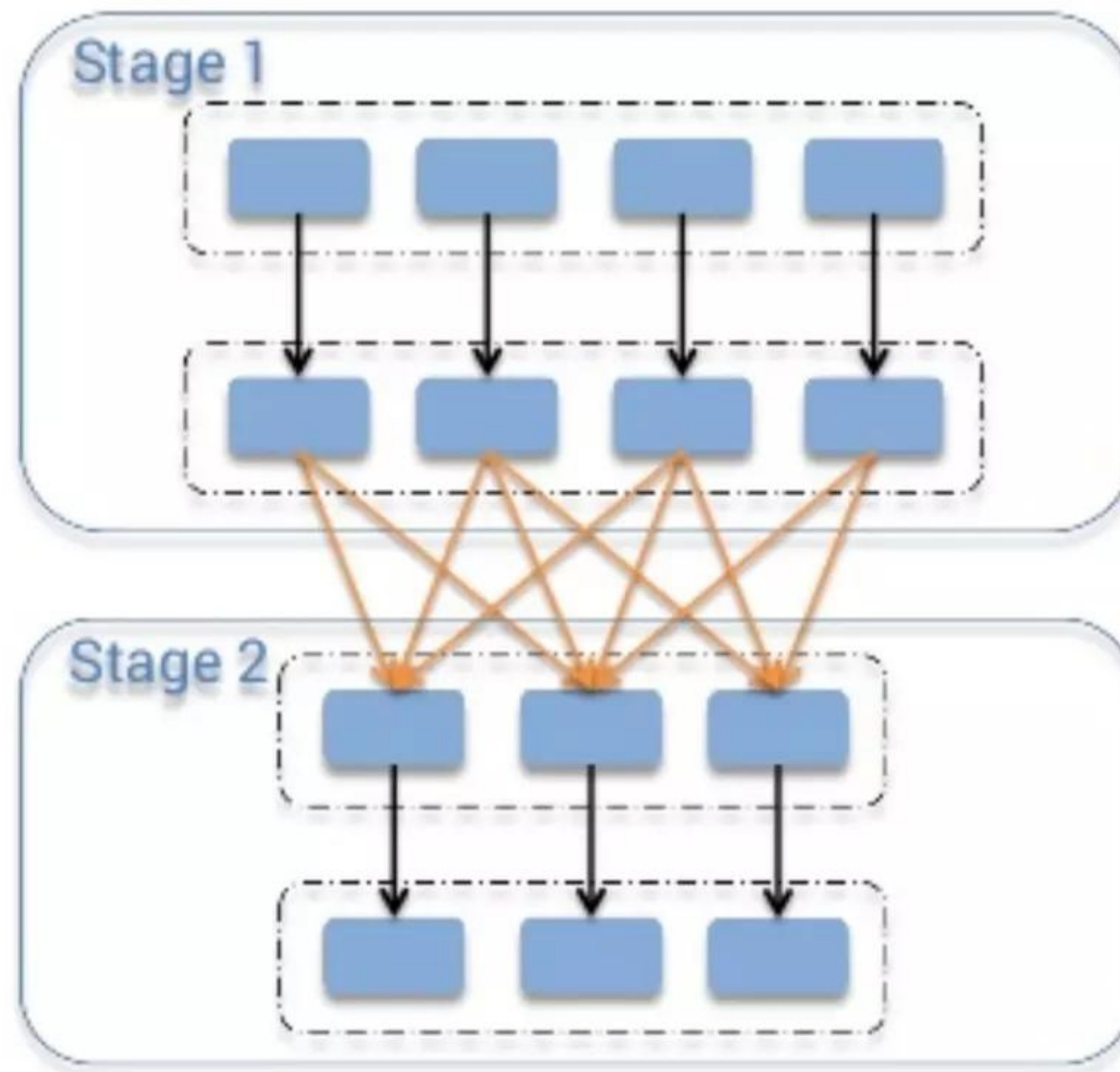


Ok, what's a shuffle block again?

- In MapReduce terminology, a Mapper-Reducer pair – the file from local disk that the reducers read from local disk in MapReduce.



In other words



Each yellow arrow in this diagram represents a shuffle block.



Wait! What!?! This is Big Data stuff, no?

- Yeah! Nope!
- Spark uses `ByteBuffer` as abstraction for storing blocks

```
val buf = ByteBuffer.allocate(length.toInt)
```

- `ByteBuffer` is limited by `Integer.MAX_SIZE` (2 GB)!



Once again

- No Spark shuffle block can be greater than 2 GB



Spark SQL

- Especially problematic for Spark SQL
- Default number of partitions to use when doing shuffles is 200
 - This low number of partitions leads to high shuffle block size



Umm, ok, so what can I do?

1. Increase the number of partitions
 - Thereby, reducing the average partition size
2. Get rid of skew in your data
 - More on that later



Umm, how exactly?

- In Spark SQL, increase the value of `spark.sql.shuffle.partitions`
- In regular Spark applications, use `rdd.repartition()` or `rdd.coalesce()`



But, how many partitions should I have?

- Rule of thumb is around 128 MB per partition



But!

- Spark uses a different data structure for bookkeeping during shuffles, when the number of partitions is less than 2000, vs. more than 2000.



Don't believe me?

- In `MapStatus.scala`

```
def apply(loc: BlockManagerId, uncompressedSizes:
Array[Long]): MapStatus = {
  if (uncompressedSizes.length > 2000) {
    HighlyCompressedMapStatus(loc,
uncompressedSizes)
  } else {
    new CompressedMapStatus(loc, uncompressedSizes)
  }
}
```



Ok, so what are you saying?

- If your number of partitions is less than 2000, but close enough to it, bump that number up to be slightly higher than 2000.



Can you summarize, please?

- Don't have too big partitions
 - Your job will fail due to 2 GB limit
- Don't have too few partitions
 - Your job will be slow, not making use of parallelism
- Rule of thumb: ~128 MB per partition
- If #partitions < 2000, but close, bump to just > 2000



Mistake # 3



SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY

Slow jobs on Join/Shuffle

- Your dataset takes 20 seconds to run over with a map job, but take 4 hours when joined or shuffled. What wrong?



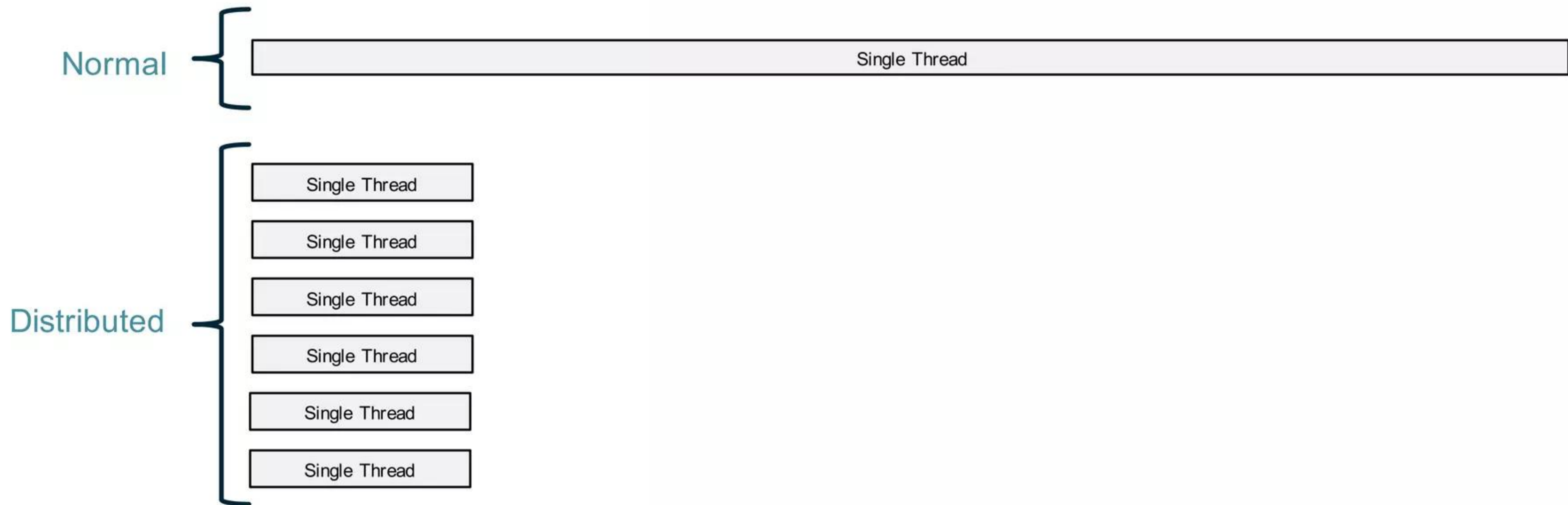
Skew and Cartesian



SPARK SUMMIT EAST
2016

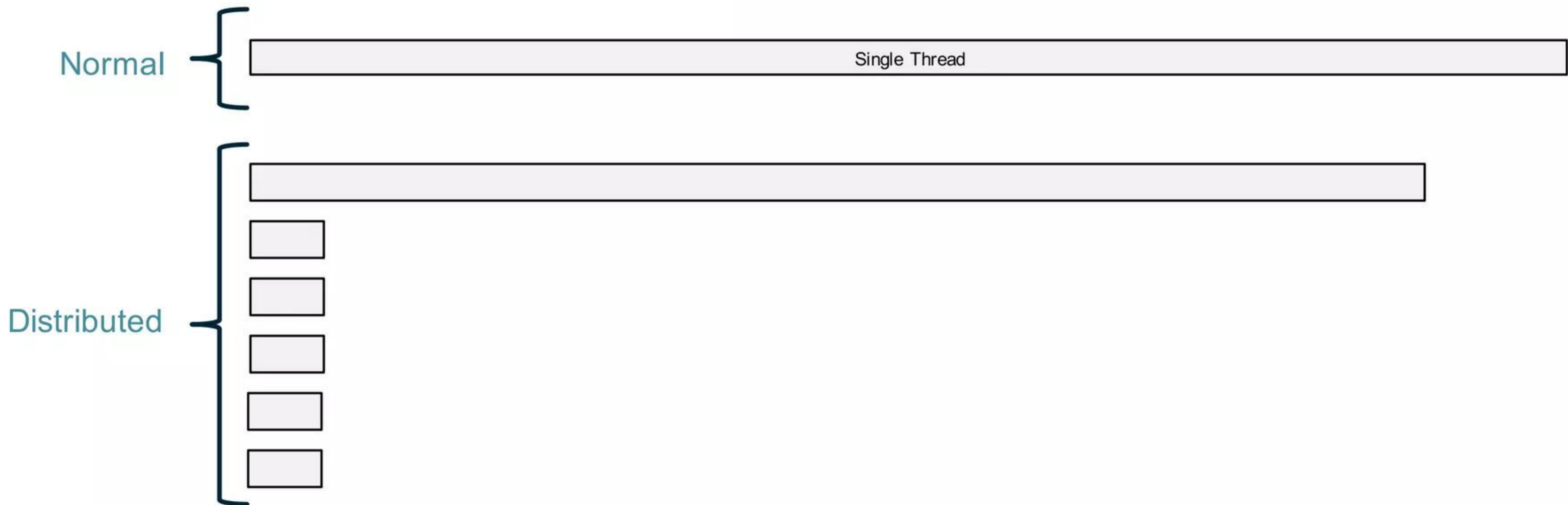
Mistake - Skew

The Holy Grail of Distributed Systems



Mistake - Skew

What about Skew, because that is a thing



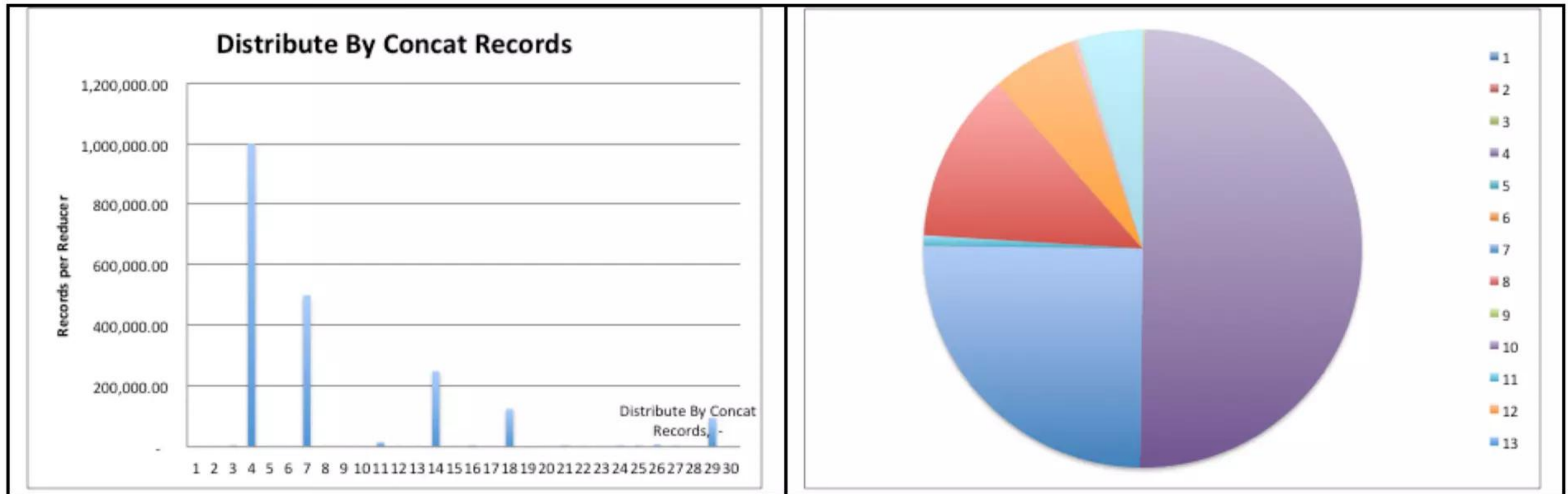
Mistake – Skew : Answers

- Salting
- Isolation Salting
- Isolation Map Joins

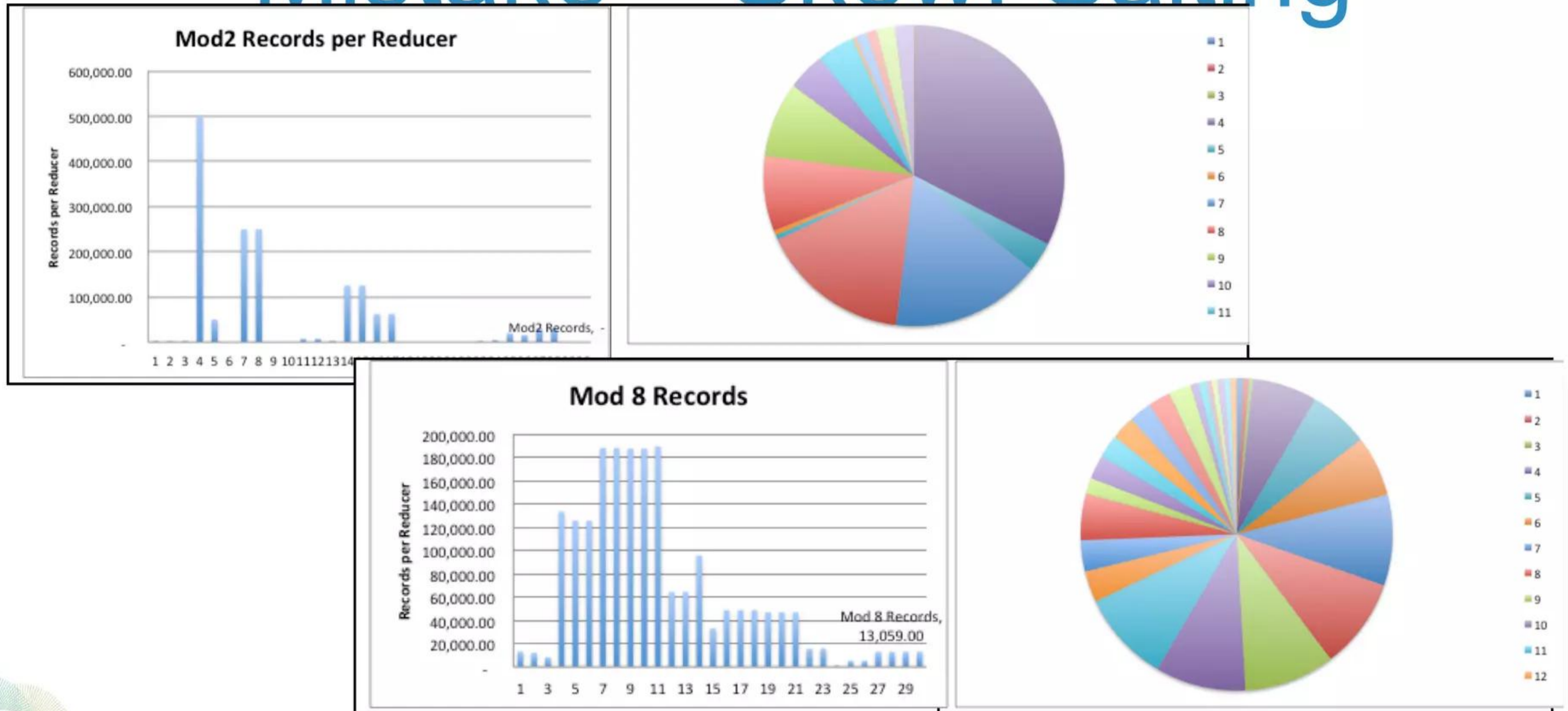
Mistake – Skew : Salting

- Normal Key: “Foo”
- Salted Key: “Foo” +
random.nextInt(saltFactor)

Managing Parallelism



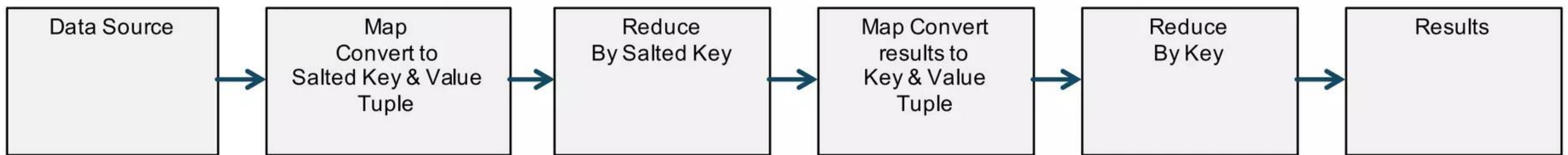
Mistake – Skew: Salting



Add Example Slide

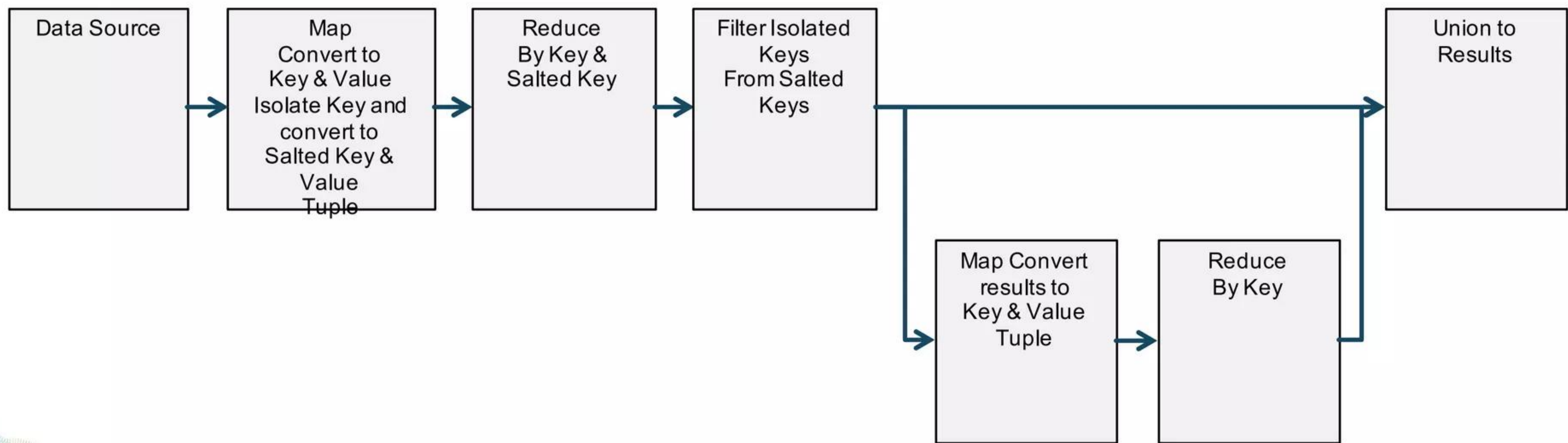
Mistake – Skew : Salting

- Two Stage Aggregation
 - Stage one to do operations on the salted keys
 - Stage two to do operation access unsalted key results



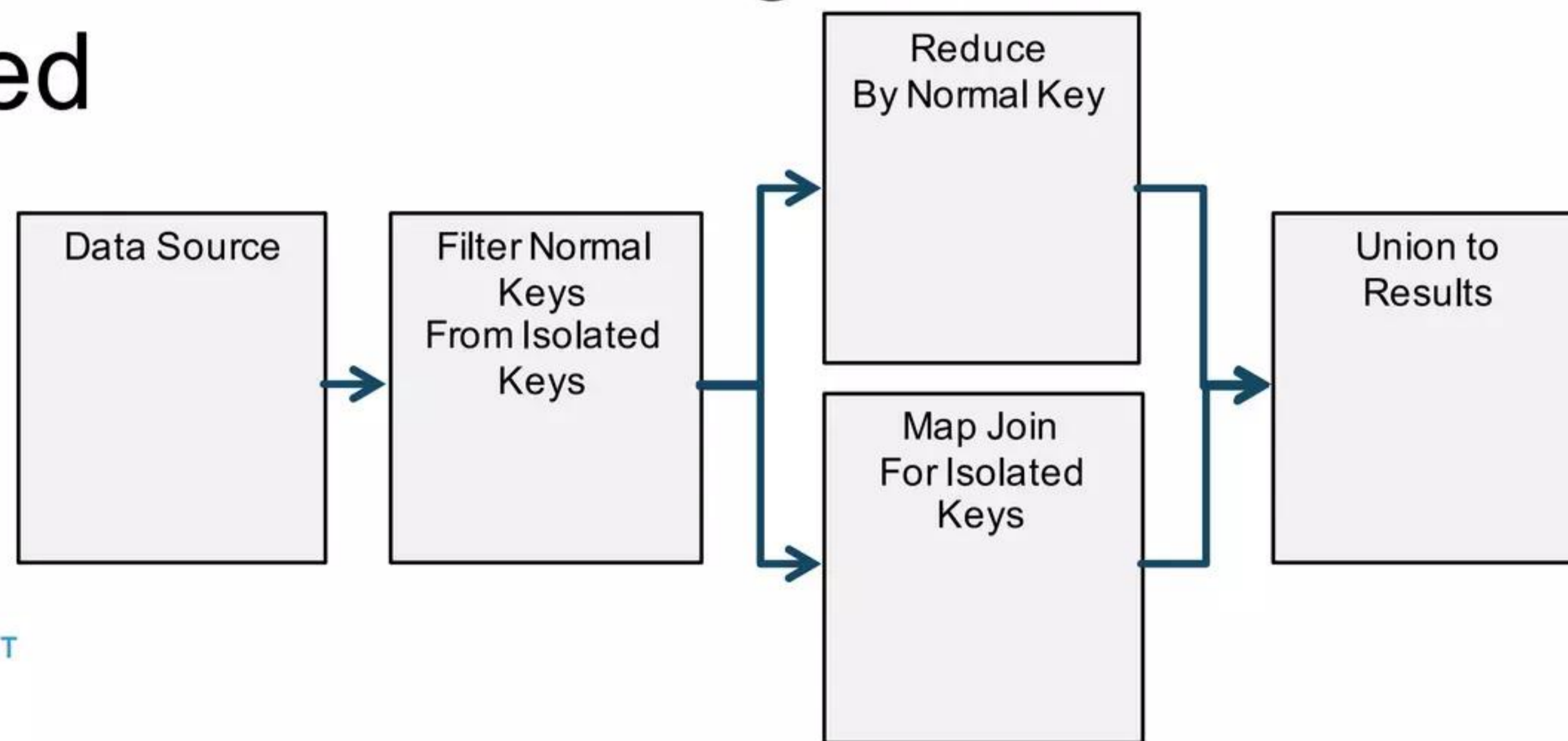
Mistake – Skew : Isolated Salting

- Second Stage only required for Isolated Keys



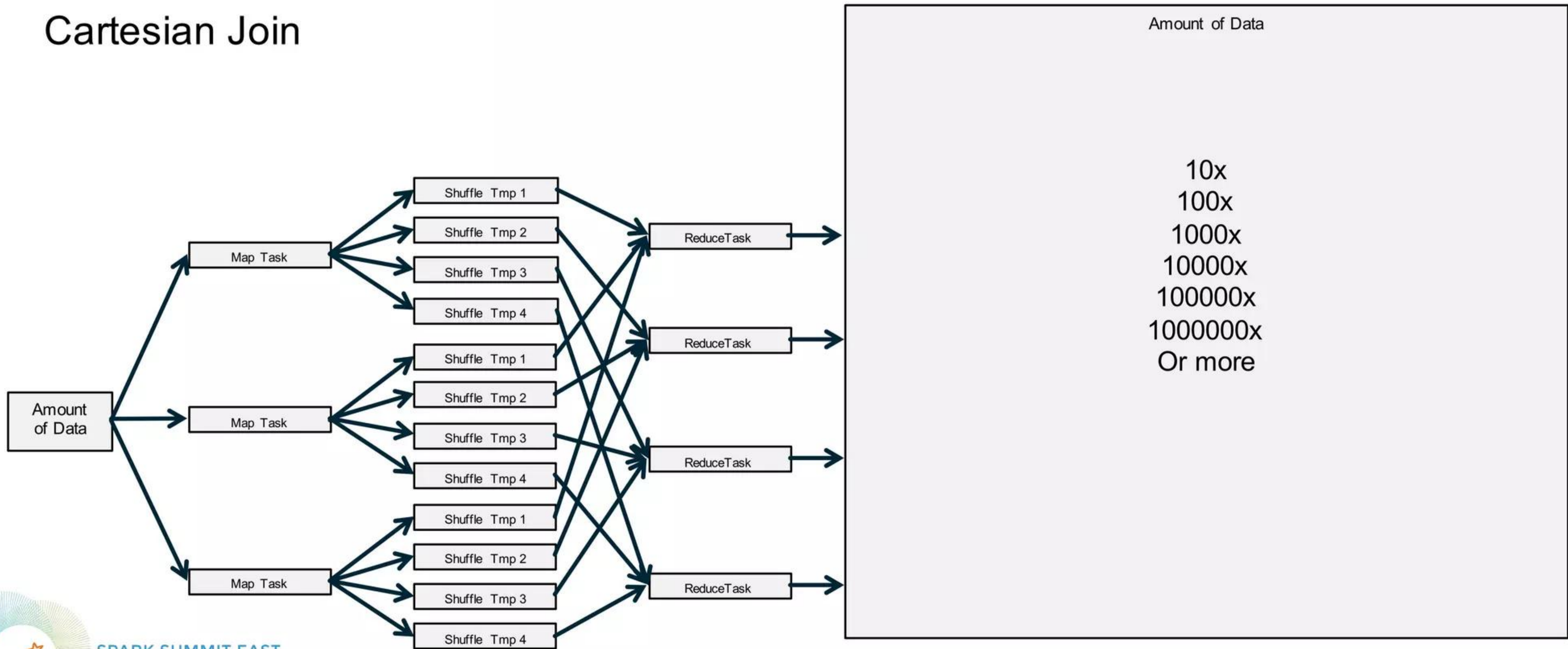
Mistake – Skew : Isolated Map Join

- Filter Out Isolated Keys and use Map Join/Aggregate on those
- And normal reduce on the rest of the data
- This can remove a large amount of data being shuffled



Managing Parallelism

Cartesian Join



Managing Parallelism

- To fight Cartesian Join
 - Nested Structures
 - Windowing
 - Skip Steps



Mistake # 4



SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY

Out of luck?

- Do you every run out of memory?
- Do you every have more then 20 stages?
- Is your driver doing a lot of work?

Mistake – DAG Management

- Shuffles are to be avoided
- ReduceByKey over GroupByKey
- TreeReduce over Reduce
- Use Complex Types



Mistake – DAG Management: Shuffles

- Map Side Reducing if possible
- Think about partitioning/bucketing ahead of time
- Do as much as possible with a single Shuffle
- Only send what you have to send
- Avoid Skew and Cartesians

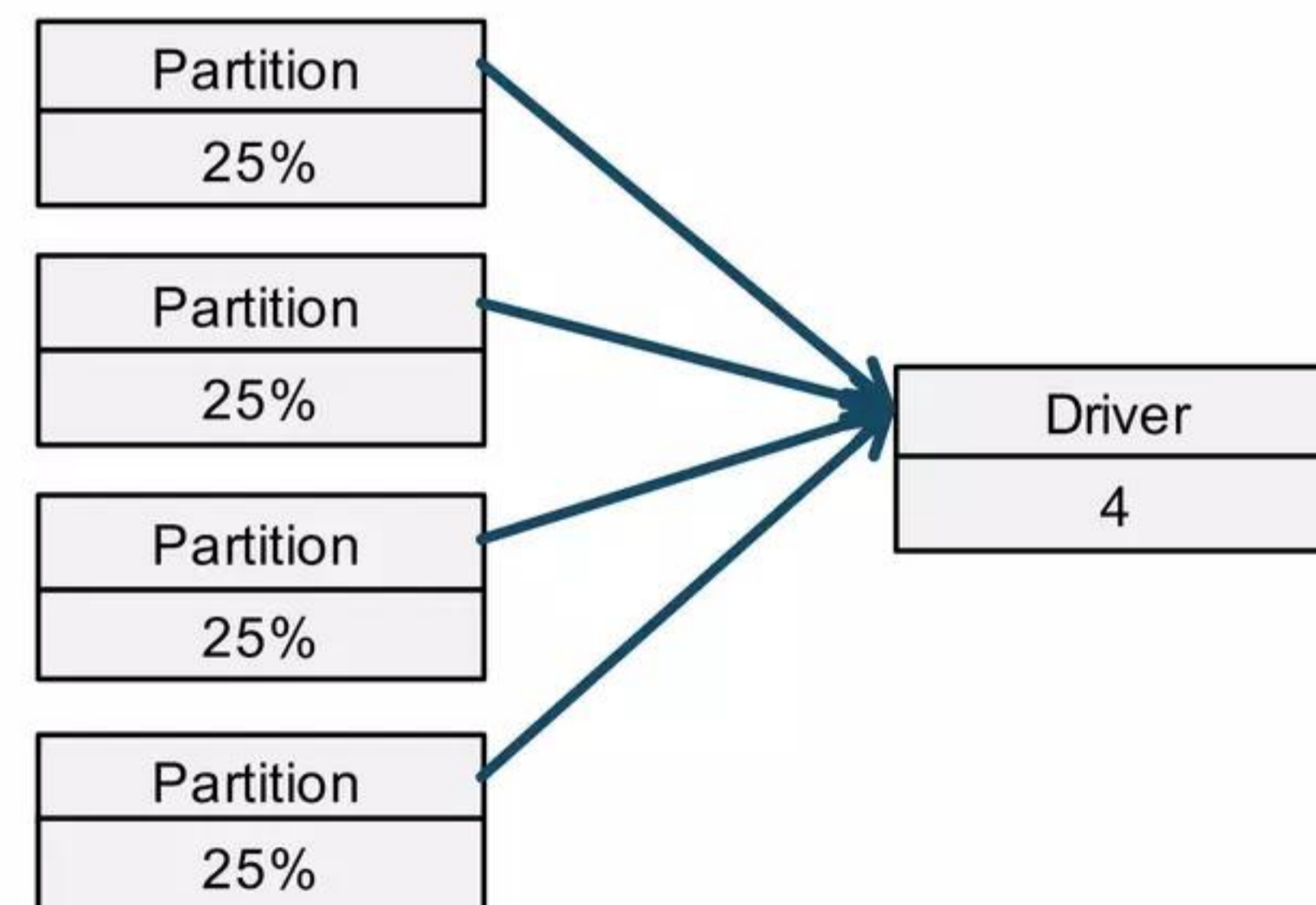
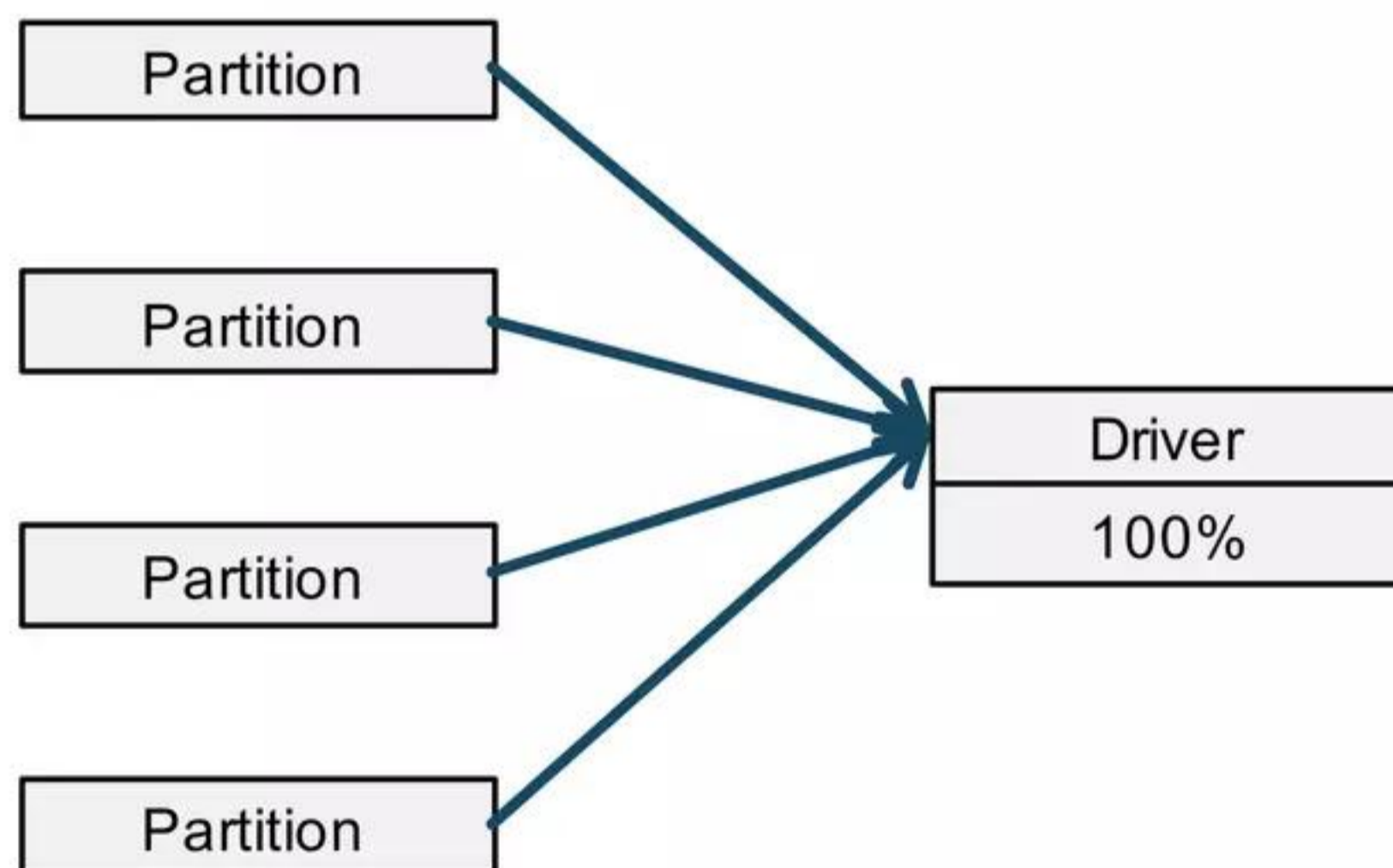


ReduceByKey over GroupByKey

- ReduceByKey can do almost anything that GroupByKey can do
 - Aggregations
 - Windowing
 - Use memory
 - But you have more control
- ReduceByKey has a fixed limit of Memory requirements
- GroupByKey is unbound and dependent of the data

TreeReduce over Reduce

- TreeReduce & Reduce returns a result to the driver
- TreeReduce does more work on the executors
- Where Reduce bring everything back to the driver



Complex Types

- Top N List
- Multiple types of Aggregations
- Windowing operations

- All in one pass

Complex Types

- Think outside of the box use objects to reduce by
- (Make something simple)

How-to: Do Data Quality Checks using Apache Spark DataFrames

July 9, 2015 | By Ted Malaska | 3 Comments

Categories: [How-to](#) [Spark](#)

Apache Spark's ability to support data quality checks via DataFrames is progressing rapidly. This post explains the state of the art and future possibilities.

Apache Hadoop and [Apache Spark](#) make Big Data accessible and usable so we can easily find value, but that data has to be correct, first. This post will focus on this problem and



Mistake # 5



SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY

Ever seen this?

```
Exception in thread "main" java.lang.NoSuchMethodError:
com.google.common.hash.HashFunction.hashInt(I)Lcom/google/common/hash/HashCode;
  at org.apache.spark.util.collection.OpenHashSet.org
$apache$spark$util$collection$OpenHashSet$$hashcode(OpenHashSet.scala:261)
  at
org.apache.spark.util.collection.OpenHashSet$mcl$sp.getPos$mcl$sp(OpenHashSet.scala:165)
  at
org.apache.spark.util.collection.OpenHashSet$mcl$sp.contains$mcl$sp(OpenHashSet.scala:102)
  at
org.apache.spark.util.SizeEstimator$$anonfun$visitArray$2.apply$mcVI$sp(SizeEstimator.scala:214)
  at scala.collection.immutable.Range.foreach$mVc$sp(Range.scala:141)
  at
org.apache.spark.util.SizeEstimator$.visitArray(SizeEstimator.scala:210)
  at.....
```



But!

- I already included guava in my app's maven dependencies?



Ah!

- My guava version doesn't match with Spark's guava version!

Shading

```
<plugin>  
  <groupId>org.apache.maven.plugins</groupId>  
  <artifactId>maven-shade-plugin</artifactId>  
  <version>2.2</version>  
...  
  <relocations>  
    <relocation>  
      <pattern>com.google.protobuf</pattern>  
      <shadedPattern>com.company.my.protobuf</shadedPattern>  
    </relocation>  
  </relocations>
```



Summary



SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY

5 Mistakes

- Size up your executors right
- 2 GB limit on Spark shuffle blocks
- Evil thing about skew and cartesian
- Learn to manage your DAG, yo!
- Do shady stuff, don't let classpath leaks mess you up



THANK YOU.

tiny.cloudera.com/spark-mistakes

Mark Grover | @mark_grover

Ted Malaska | @TedMalaska



SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY