Data Engineering

MG-GY 8441

Big Data

Agenda

- Distributed Storage
- Parallel Computing
- Dependency Graphs
- Spark

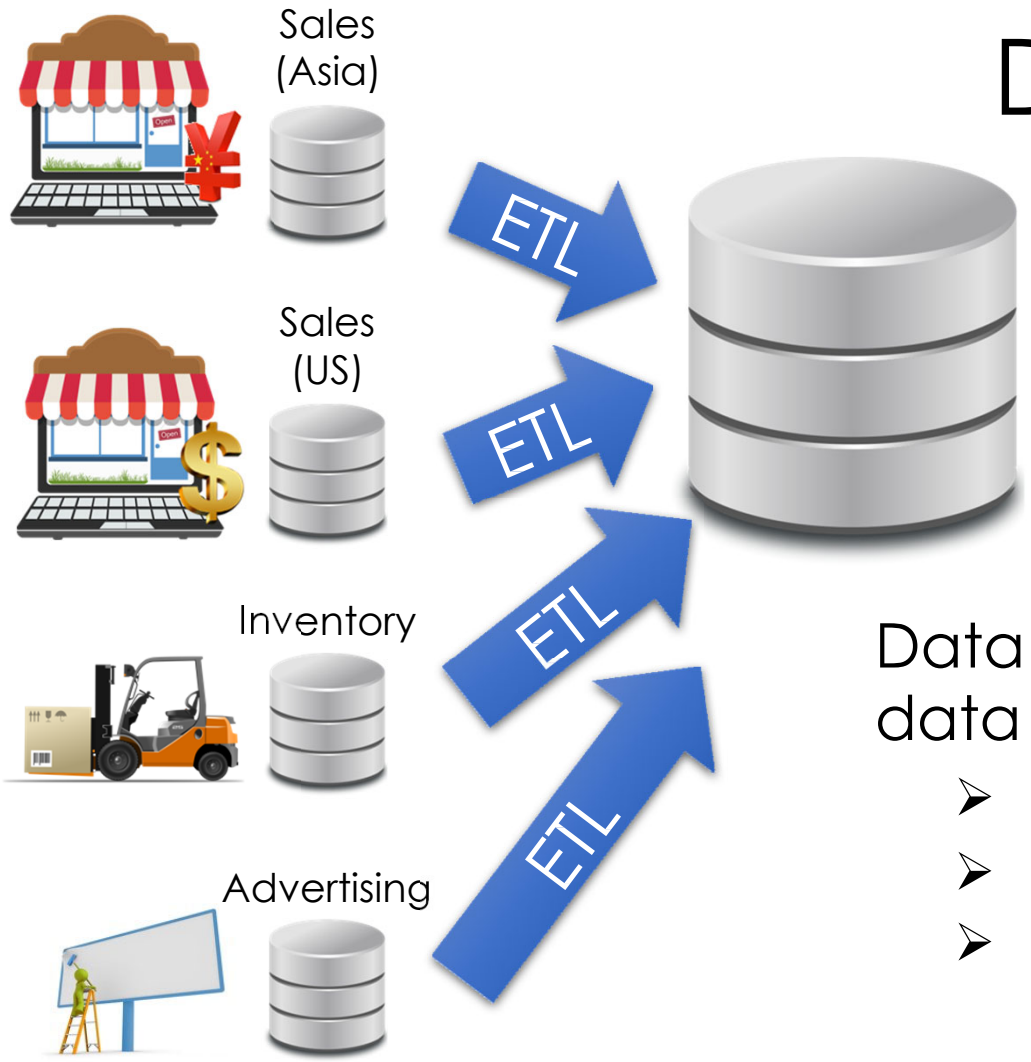
References

- Leskovec, Rajaraman, Ullman, Mining of Massive Datasets (Chapter 2.1 - 2.3)
- (Optional) Dean, Ghemawat, MapReduce: Simplified Data Processing on Large Clusters

Examples

- (2019) Facebook generates 4 Petabytes
- (2019) YouTube collects 300 hours of HD video every minute
- (2017) Twitter stores >500PB of data
- (2017) CERN data center: > 200PB

Data Warehouse

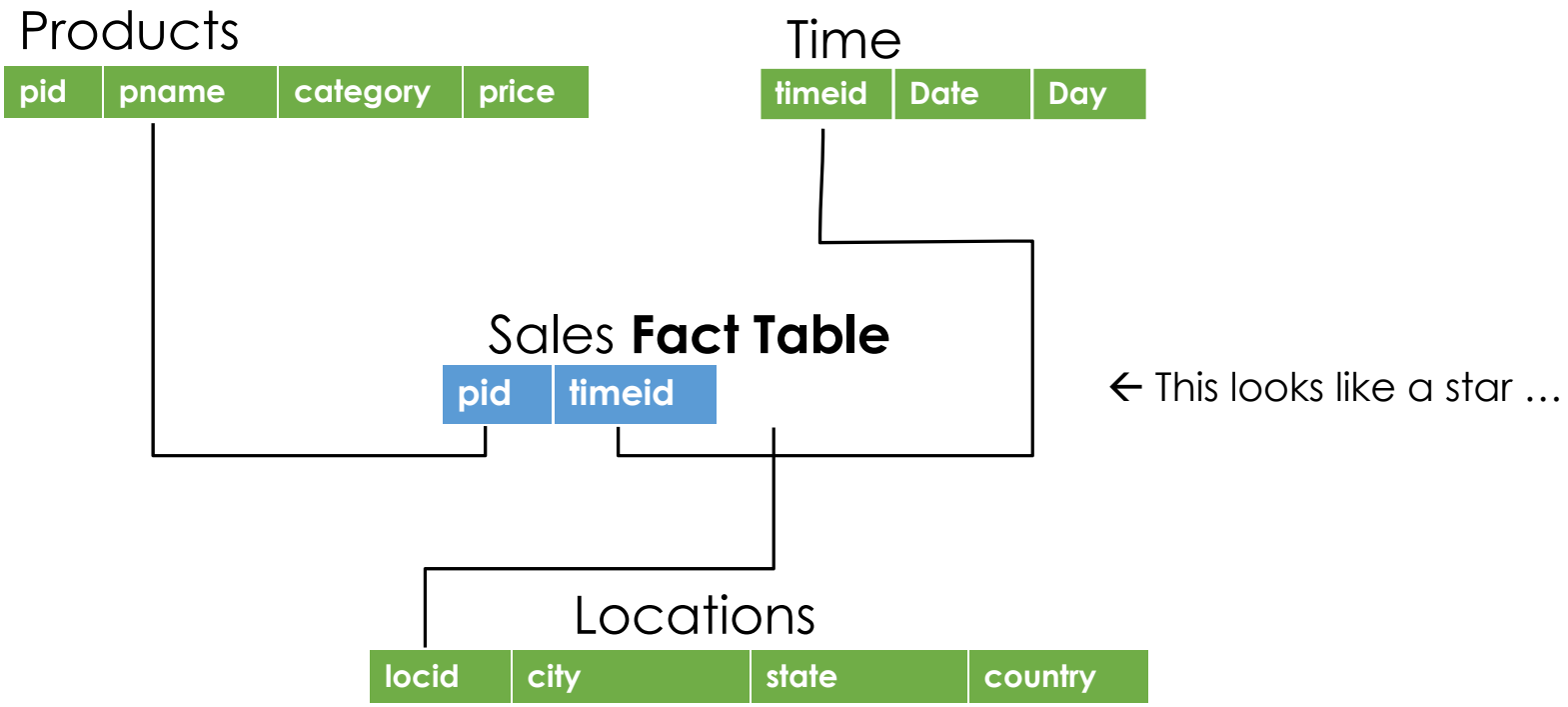


Collects and organizes historical data from multiple sources

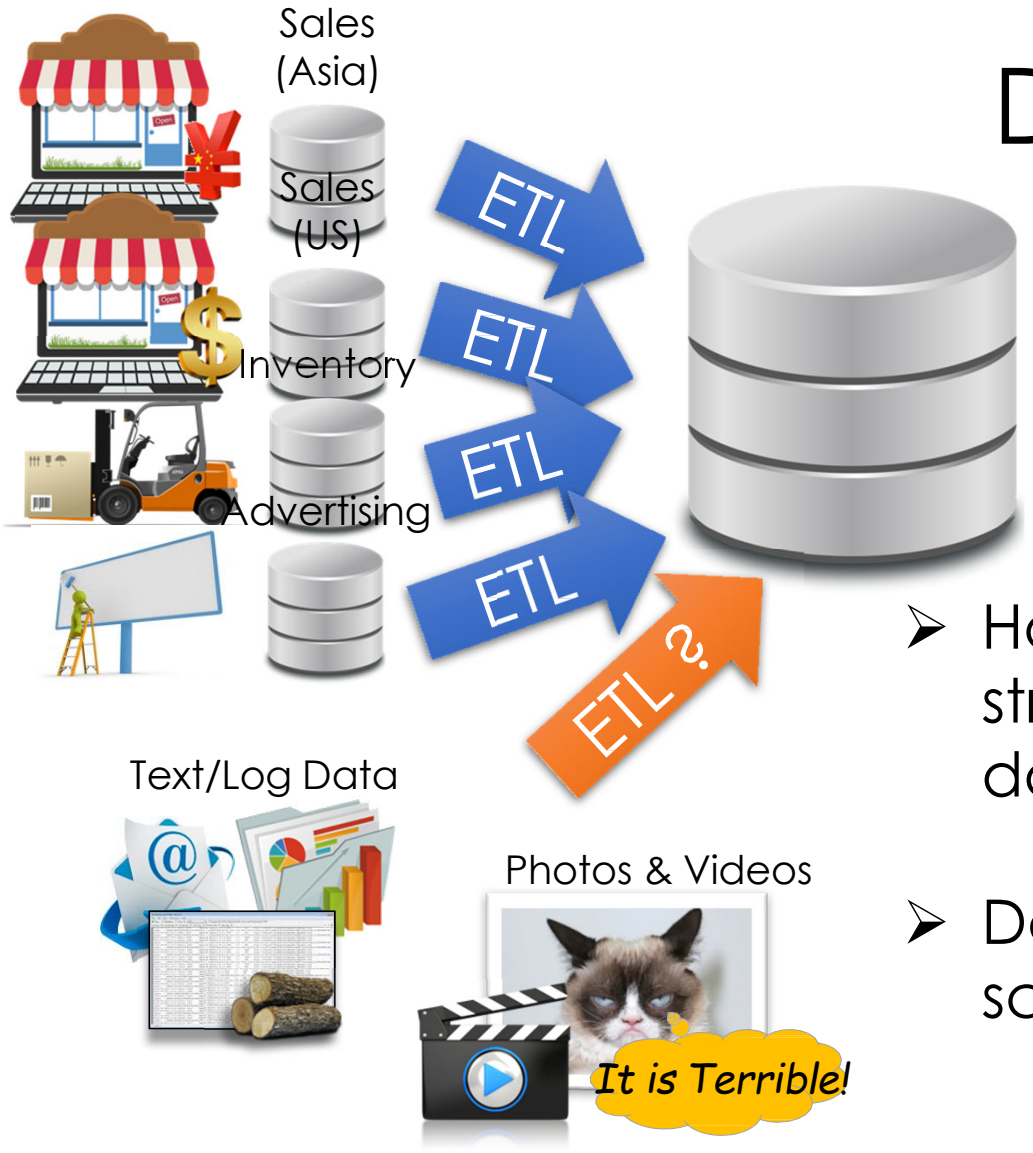
Data is *periodically* **ETL**ed into the data warehouse:

- **Extracted** from remote sources
- **Transformed** to standard schemas
- **Loaded** into the (typically) relational (SQL) data system

Schema


Products
pid
pname
category
price
Time
timeid
Date
Day
Sales **Fact Table**
pid
timeid
← This looks like a star …
Locations
locid
city
state
country

Data Warehouse

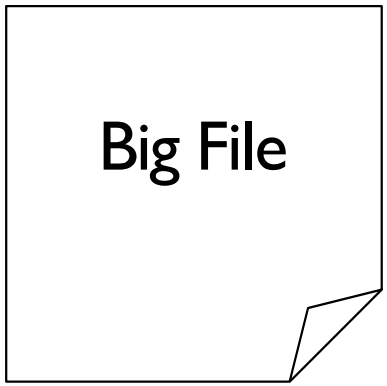


Collects and organizes historical data from multiple sources

- How do we deal with semi-structured and unstructured data?
- Do we really want to force a schema on load?

Distributed File Systems

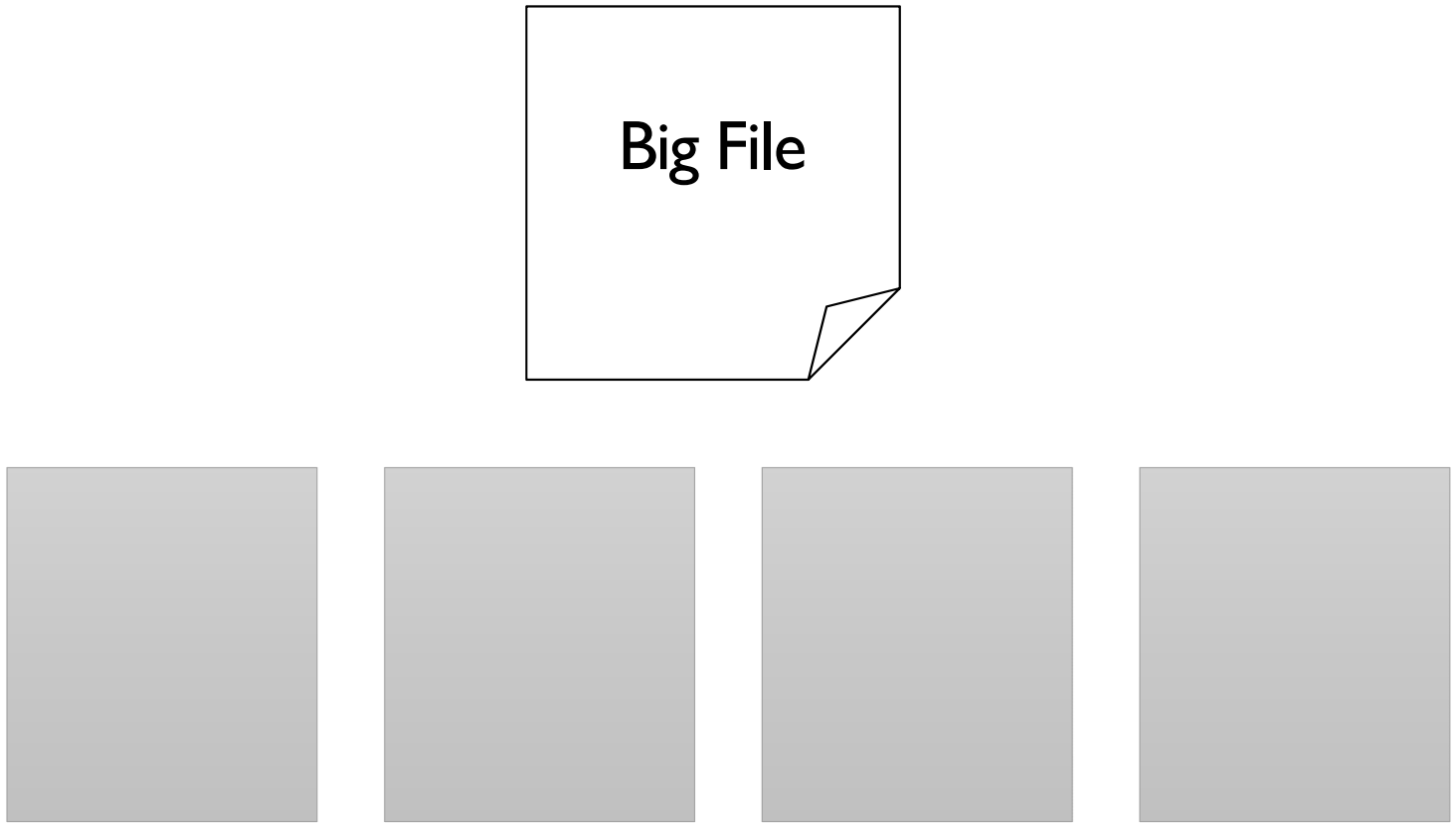
Fault Tolerant Distributed File Systems (e.g., HDFS)



How do we **store** and **access** very **large files** across **cheap** commodity devices ?

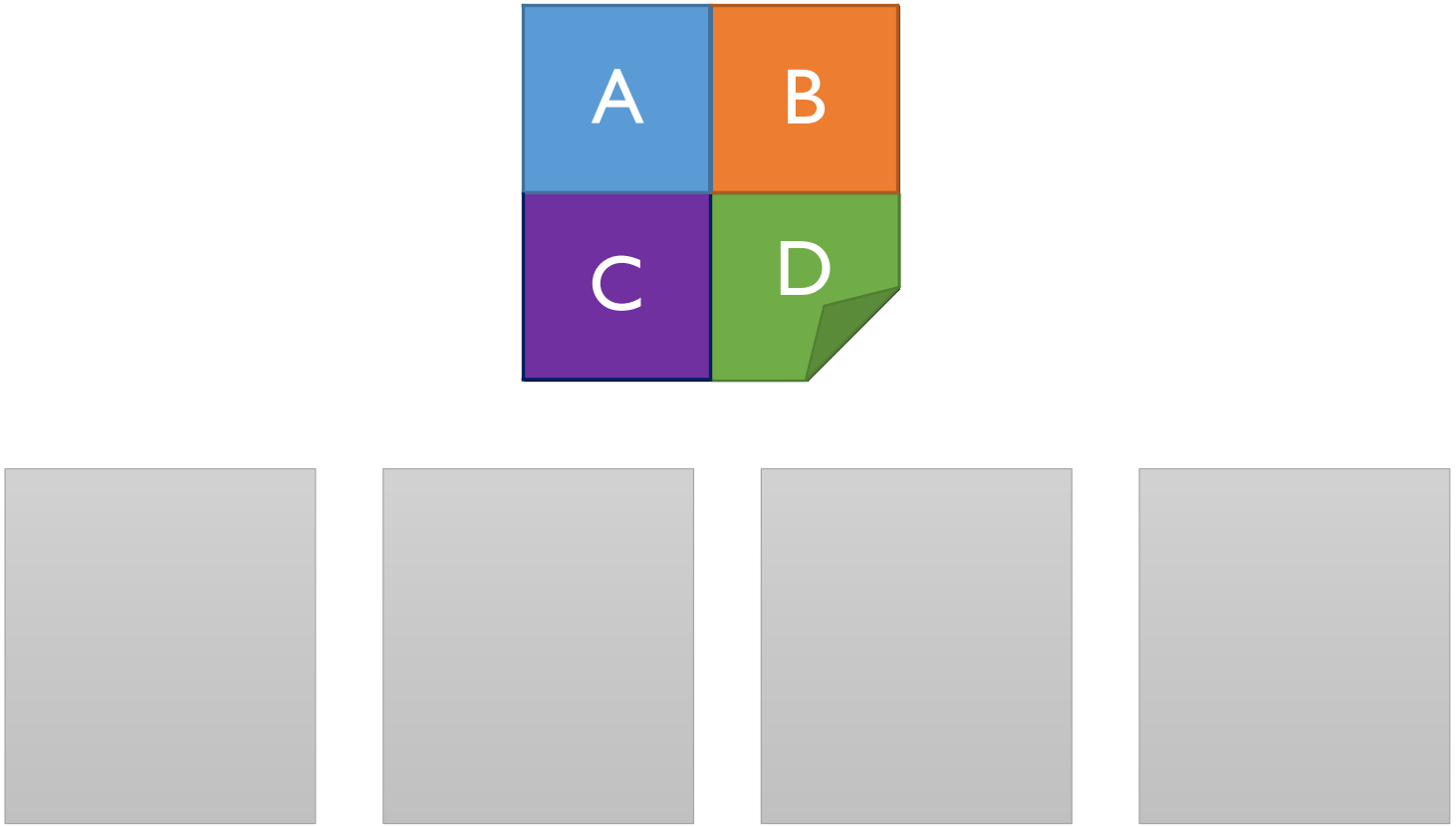
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems (e.g., HDFS)



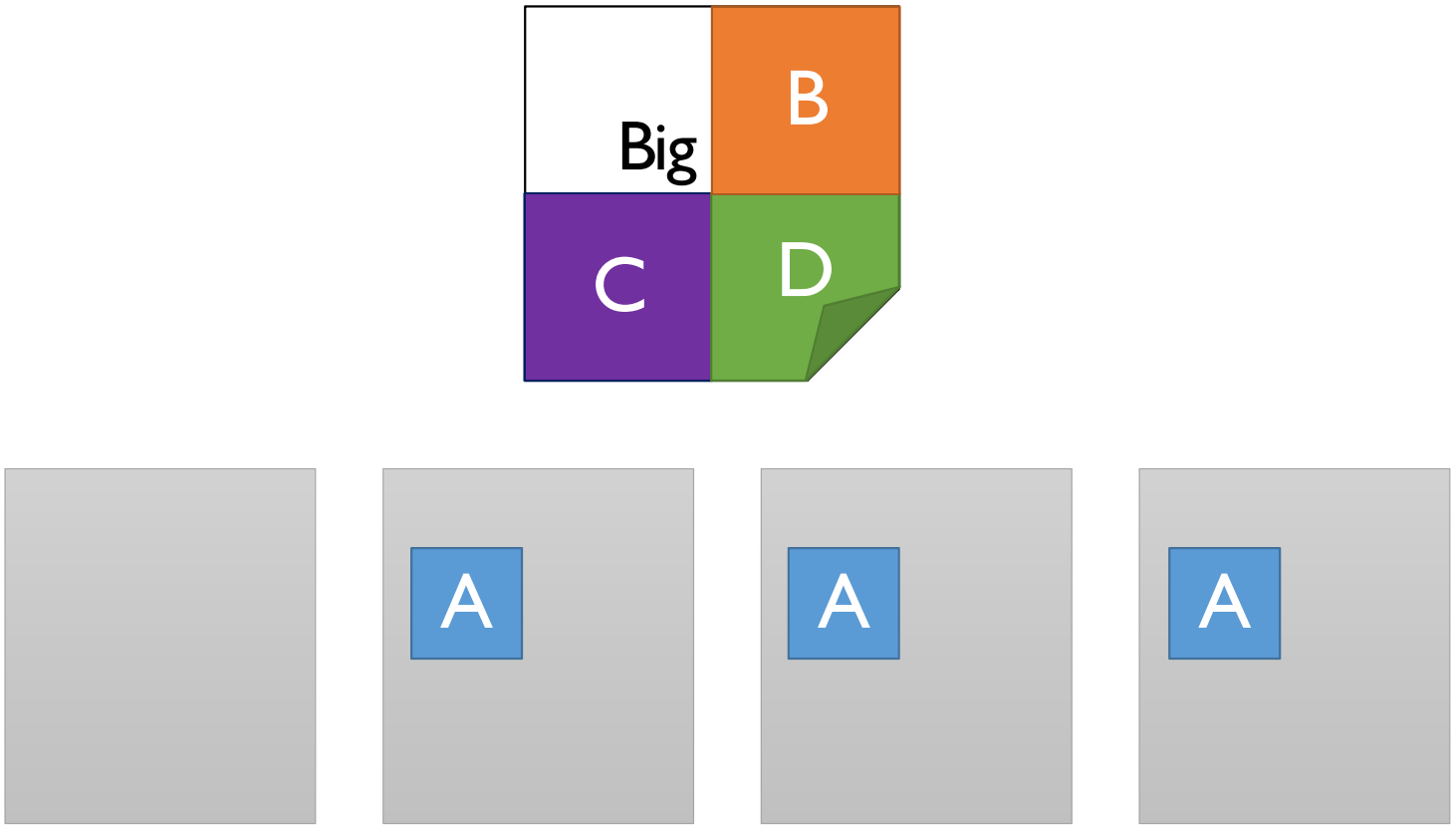
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems (e.g., HDFS)



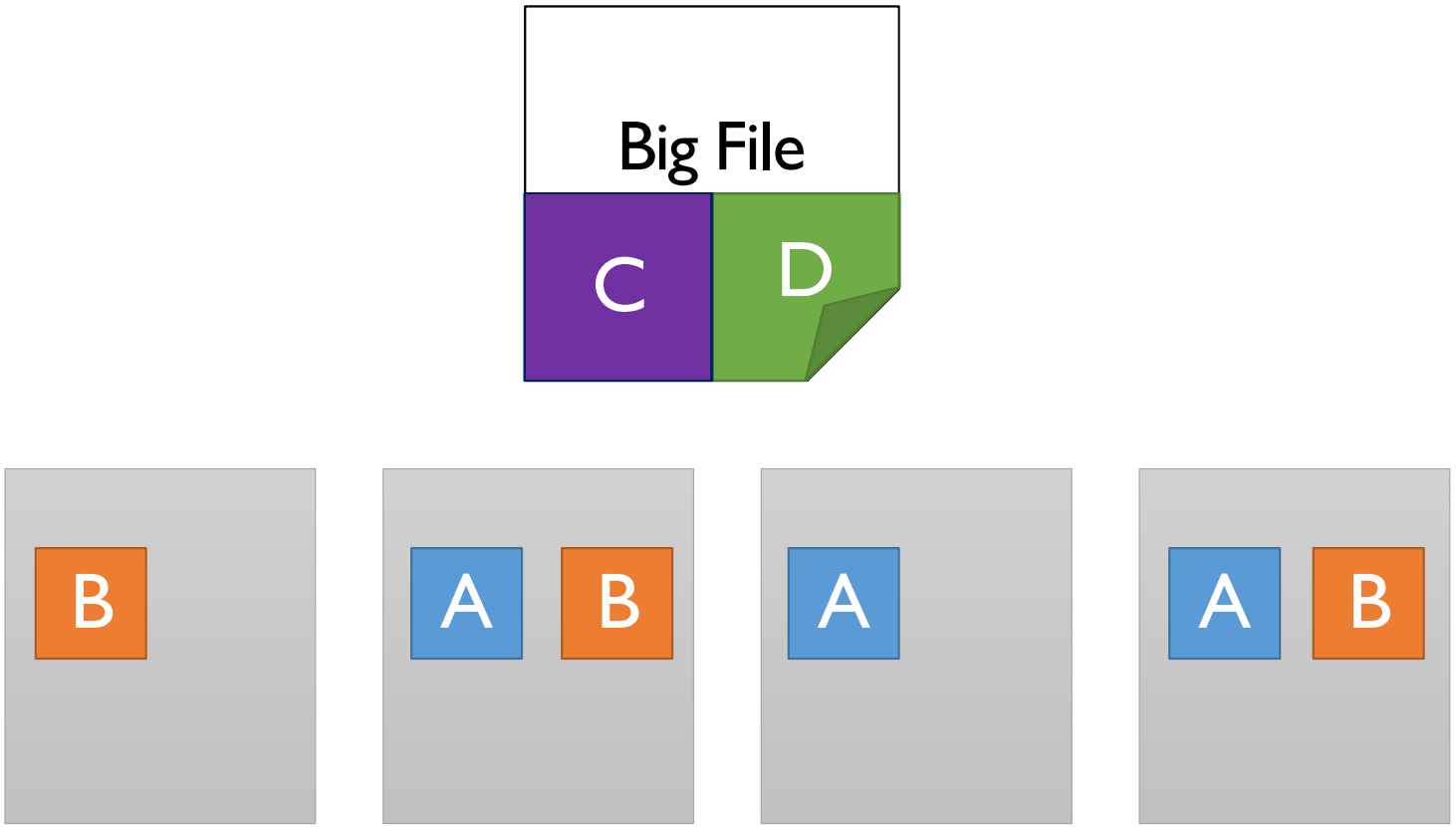
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems (e.g., HDFS)



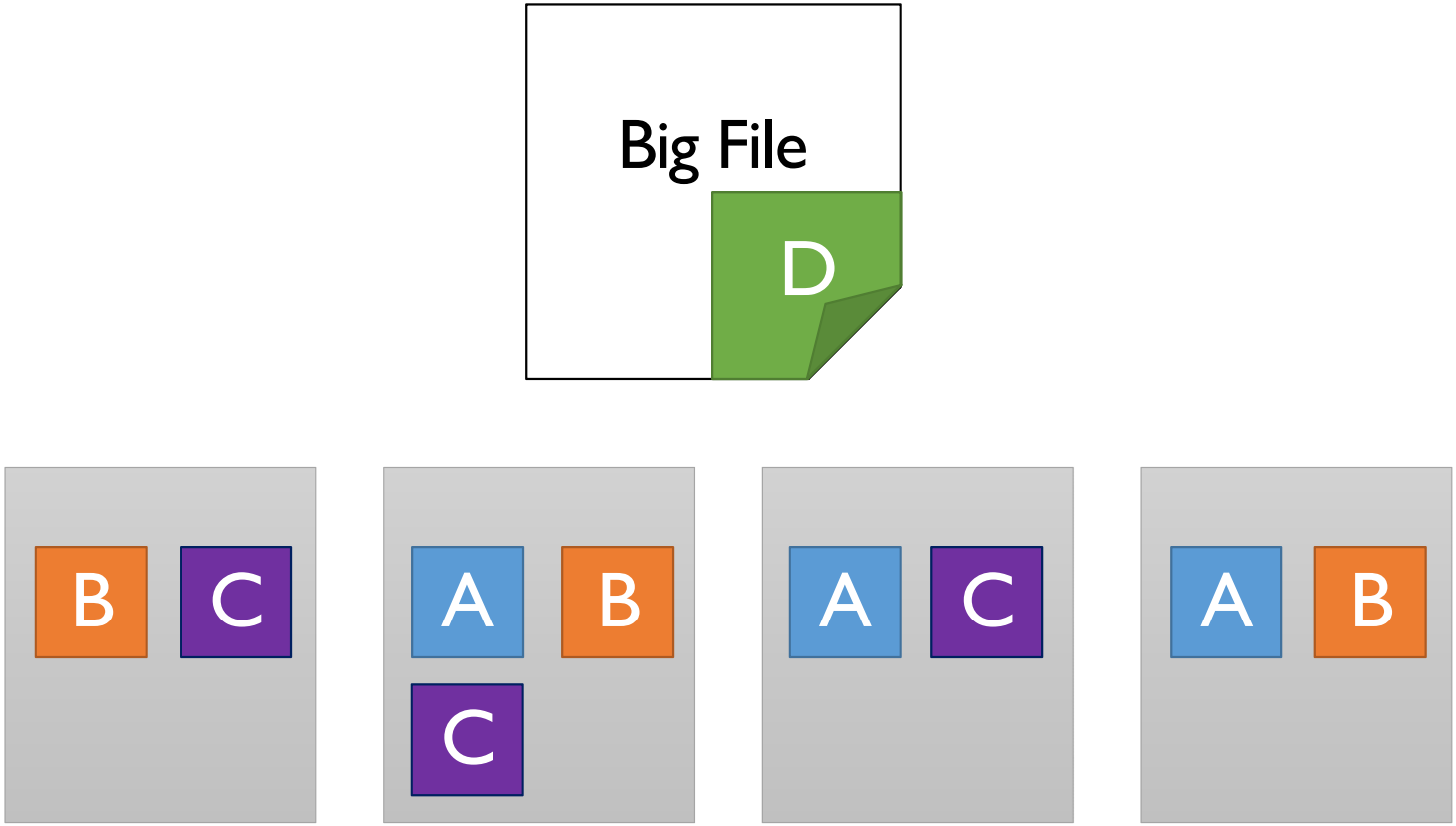
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems (e.g., HDFS)



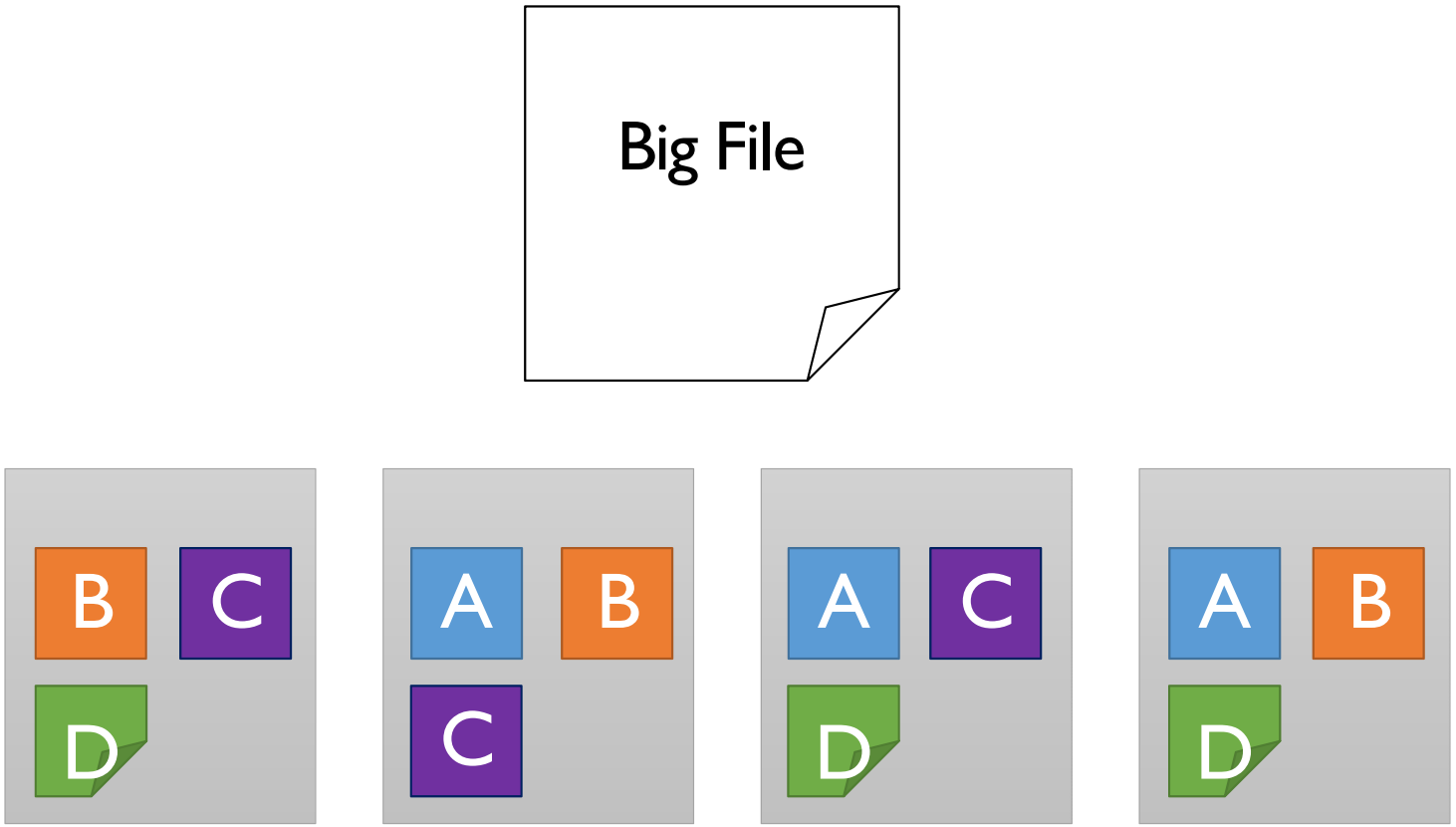
[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems (e.g., HDFS)



[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems (e.g., HDFS)

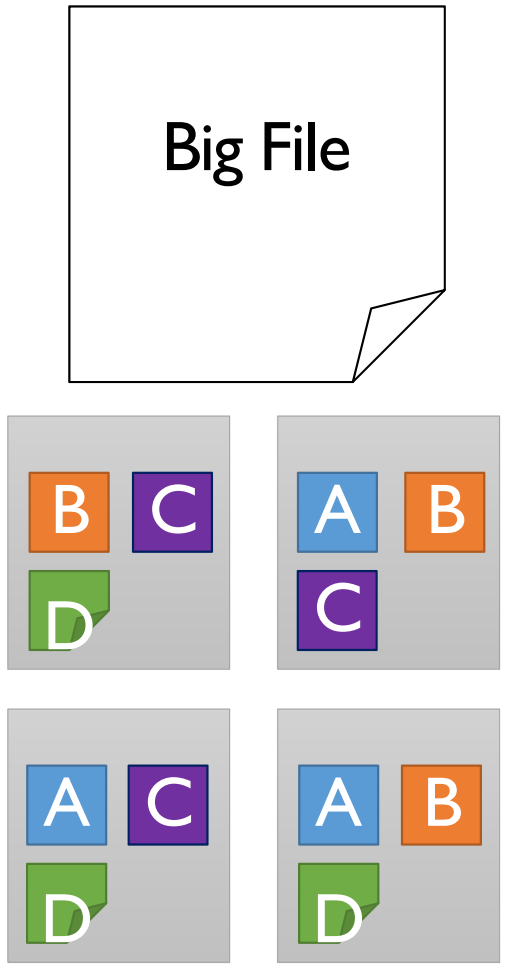


[Ghemawat et al., SOSP'03]

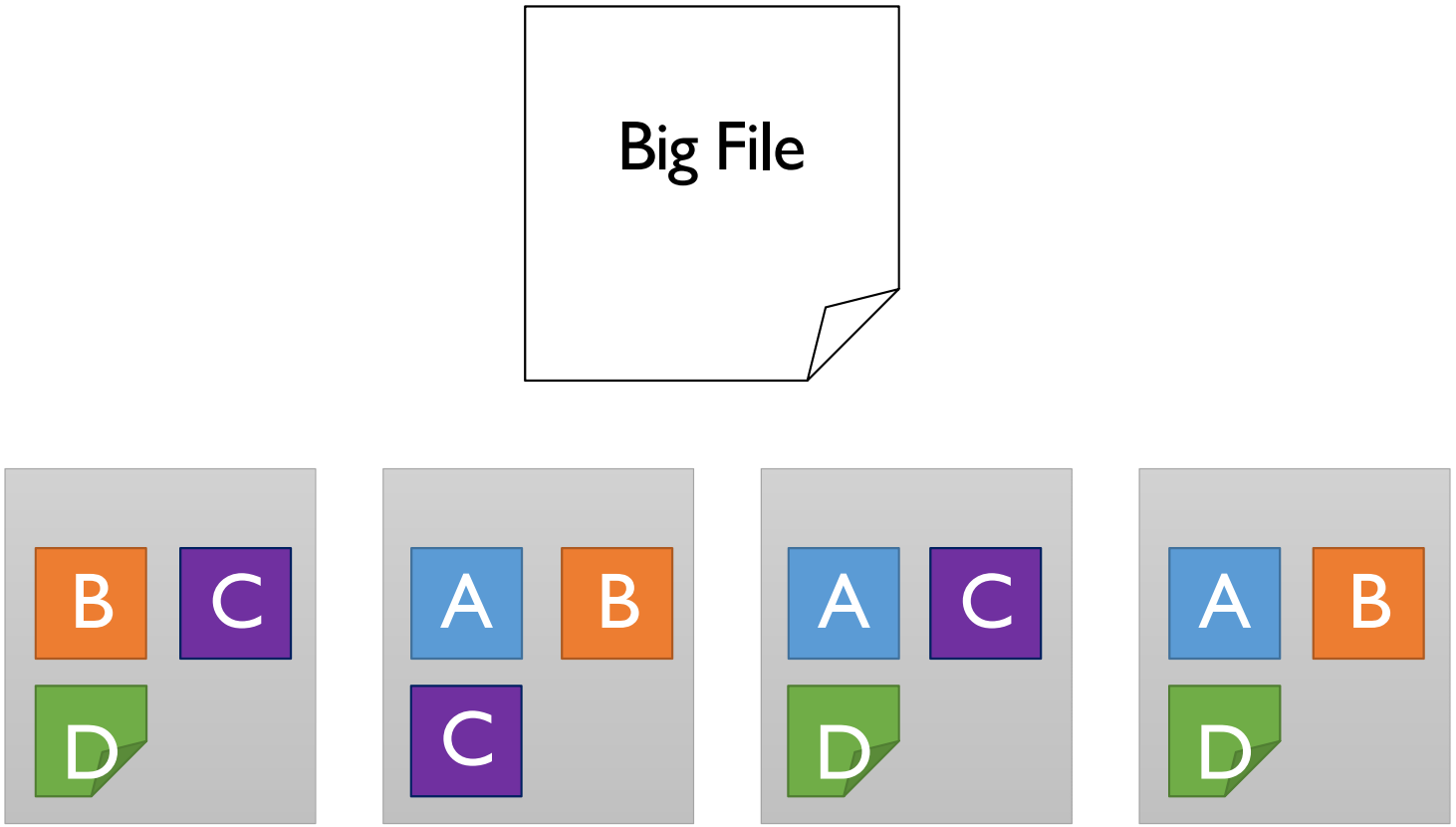
Fault Tolerant Distributed File Systems (e.g., HDFS)

- Split large files over multiple machines
  - Easily support very massive files spanning
- Read parts of file in parallel
  - Fast reads of large files
- Often built using cheap commodity storage devices

Cheap commodity storage devices will fail!

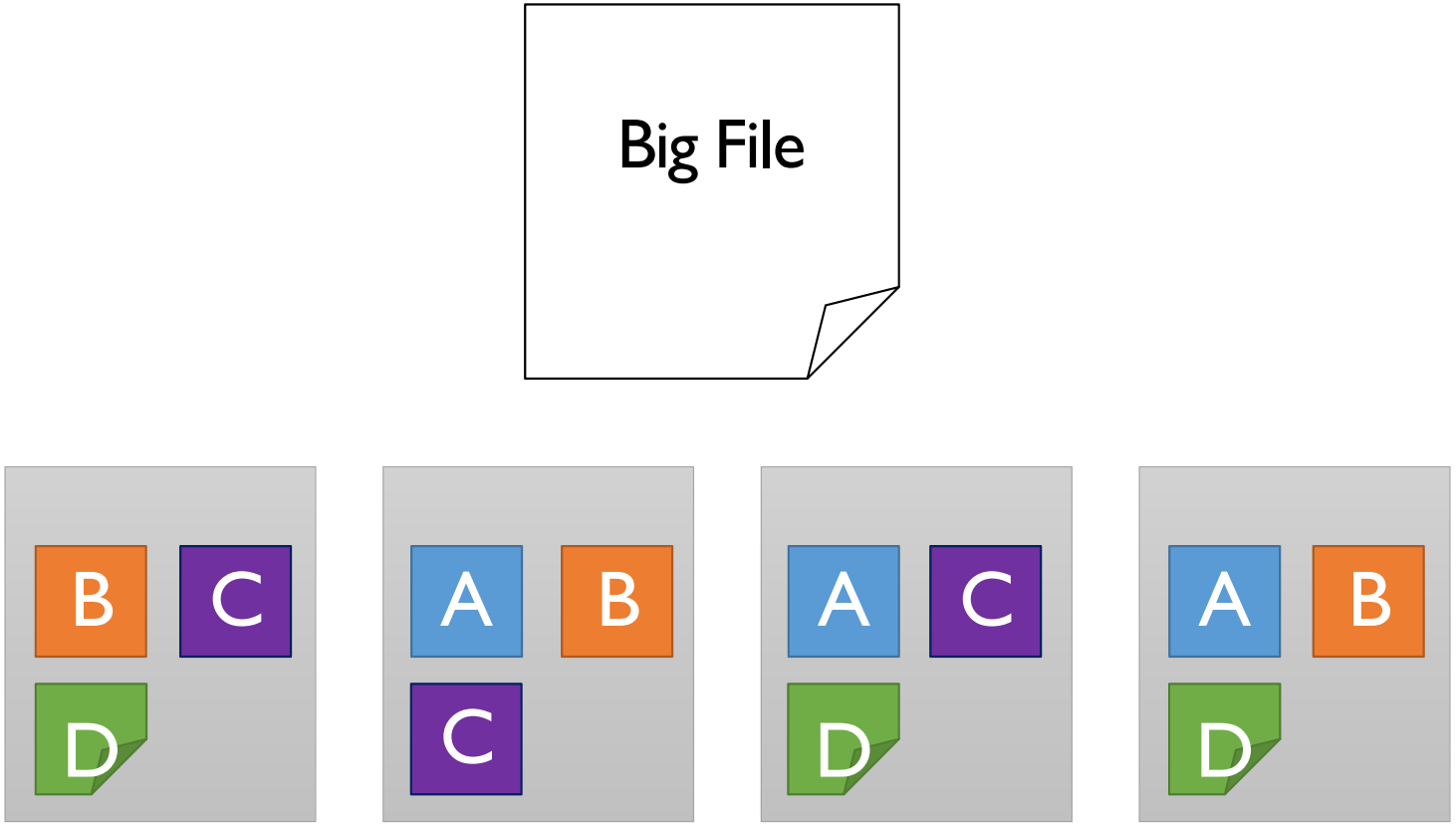


Fault Tolerant Distributed File Systems (e.g., HDFS)



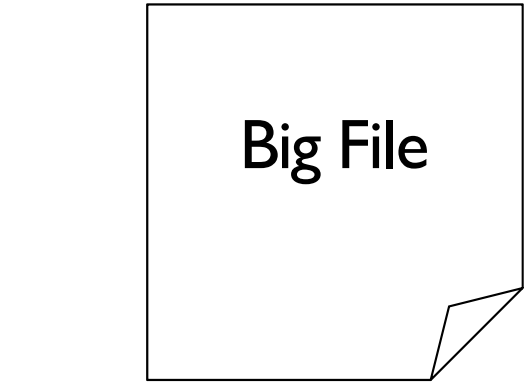
[Ghemawat et al., SOSP'03]

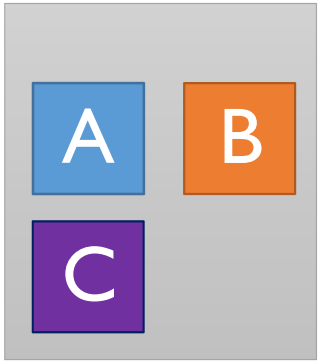
Fault Tolerant Distributed File Systems
Failure Event

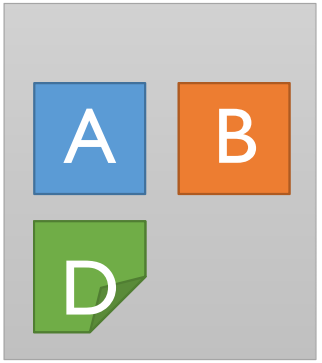


[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems
Failure Event

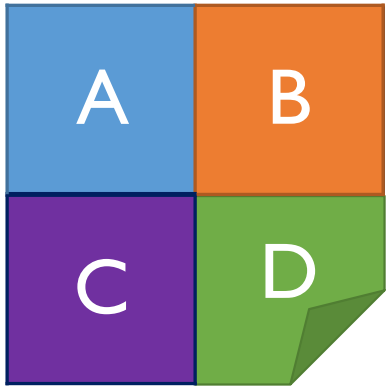


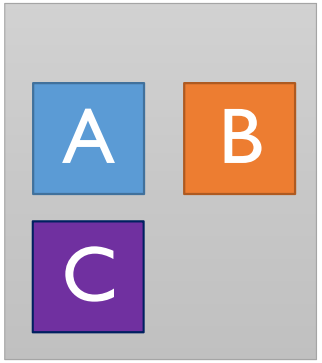


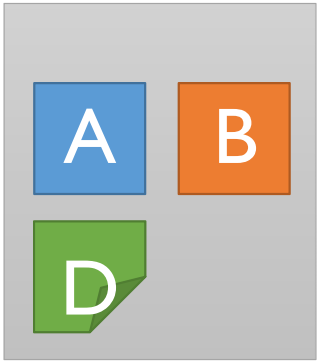


[Ghemawat et al., SOSP'03]

Fault Tolerant Distributed File Systems
Failure Event



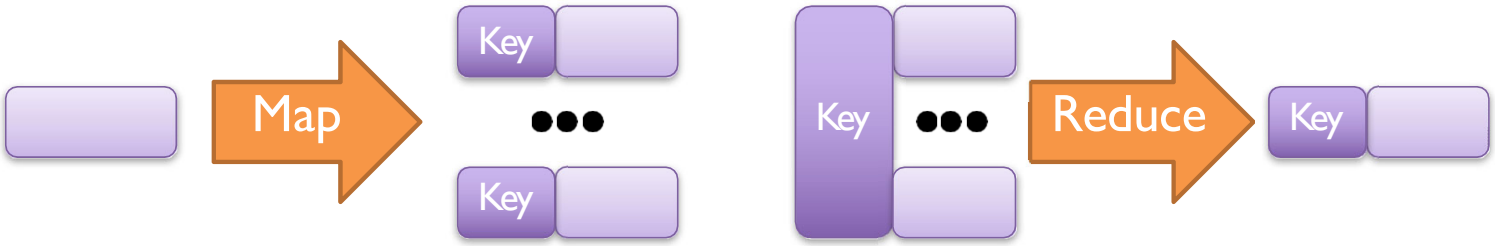




[Ghemawat et al., SOSP'03]

Map-Reduce

The Map Reduce Abstraction

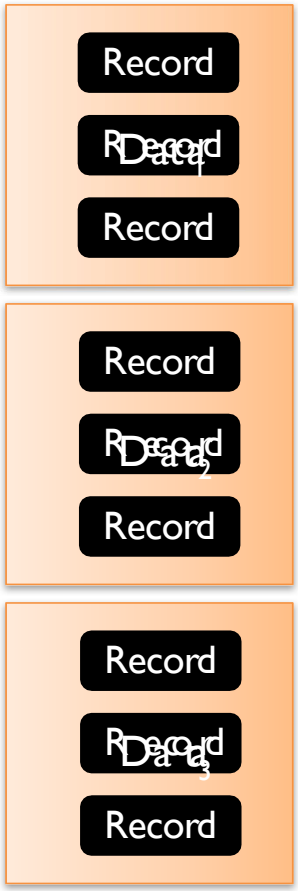


Example: *Word-Count*


```
Reduce(word, counts) {
  emit (word, SUM(counts))
}
```

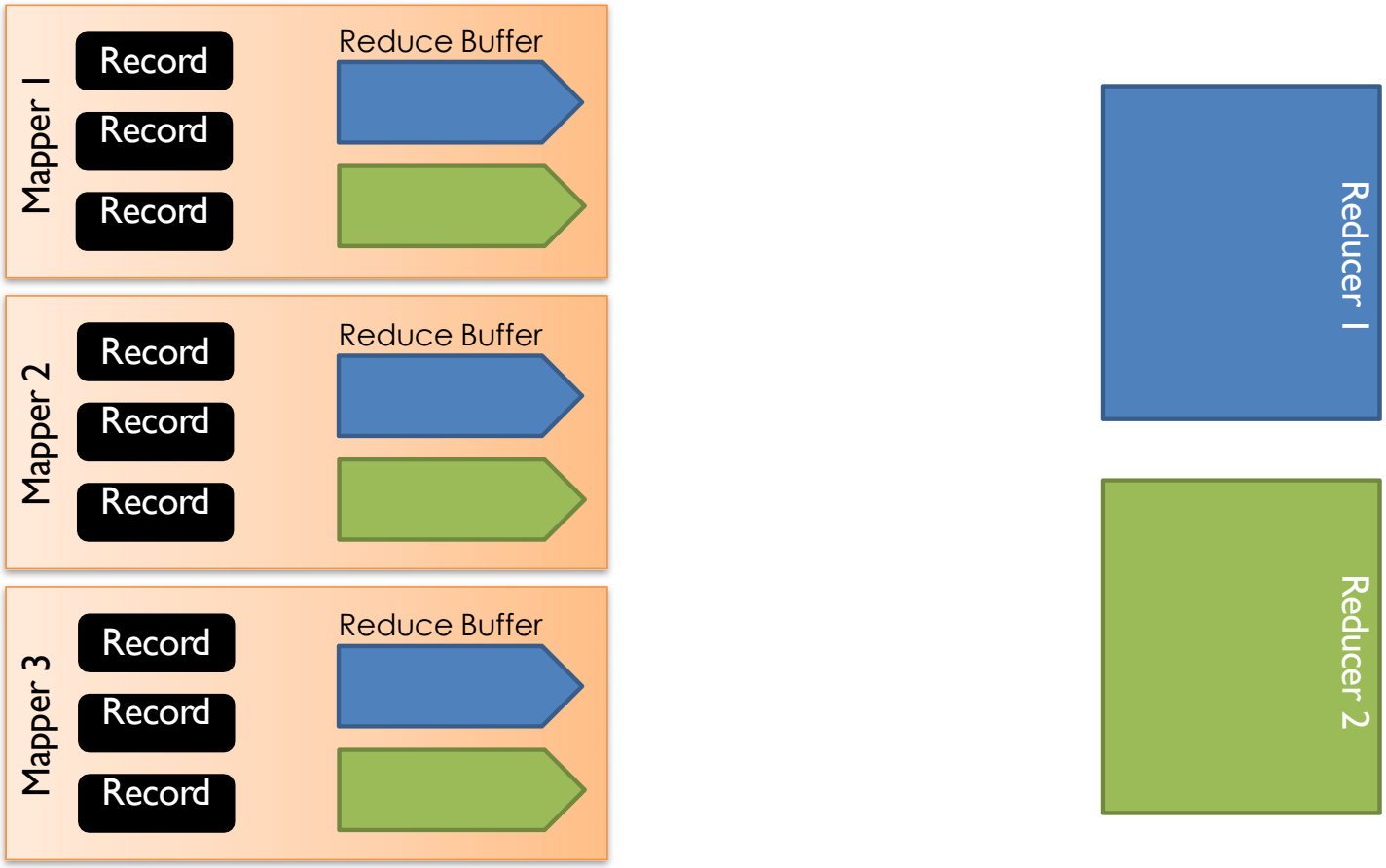
[Dean & Ghemawat, OSDI'04]

The Map Reduce System



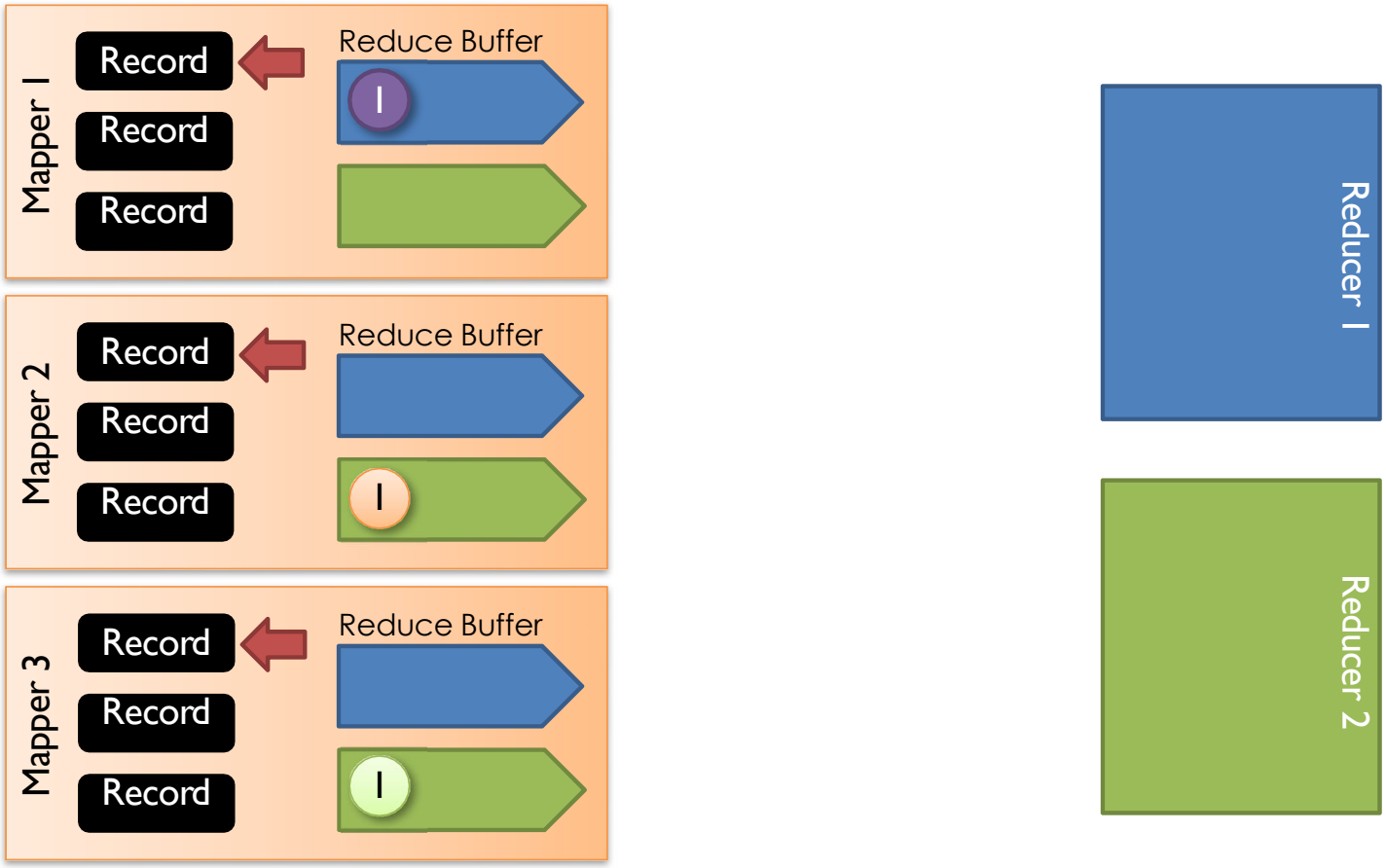
[Dean & Ghemawat, OSDI'04]

The Map Reduce System



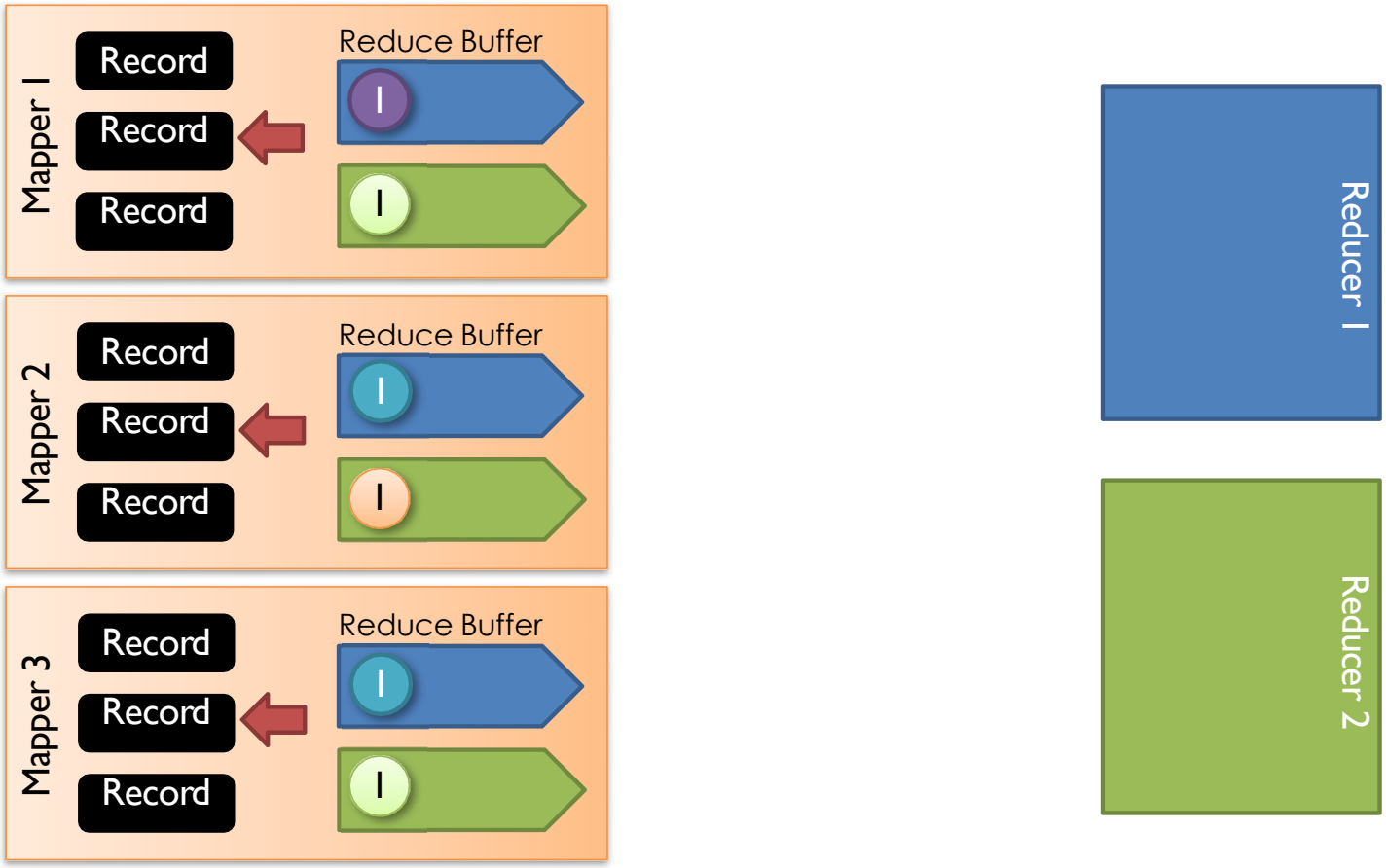
[Dean & Ghemawat, OSDI'04]

The Map Reduce System



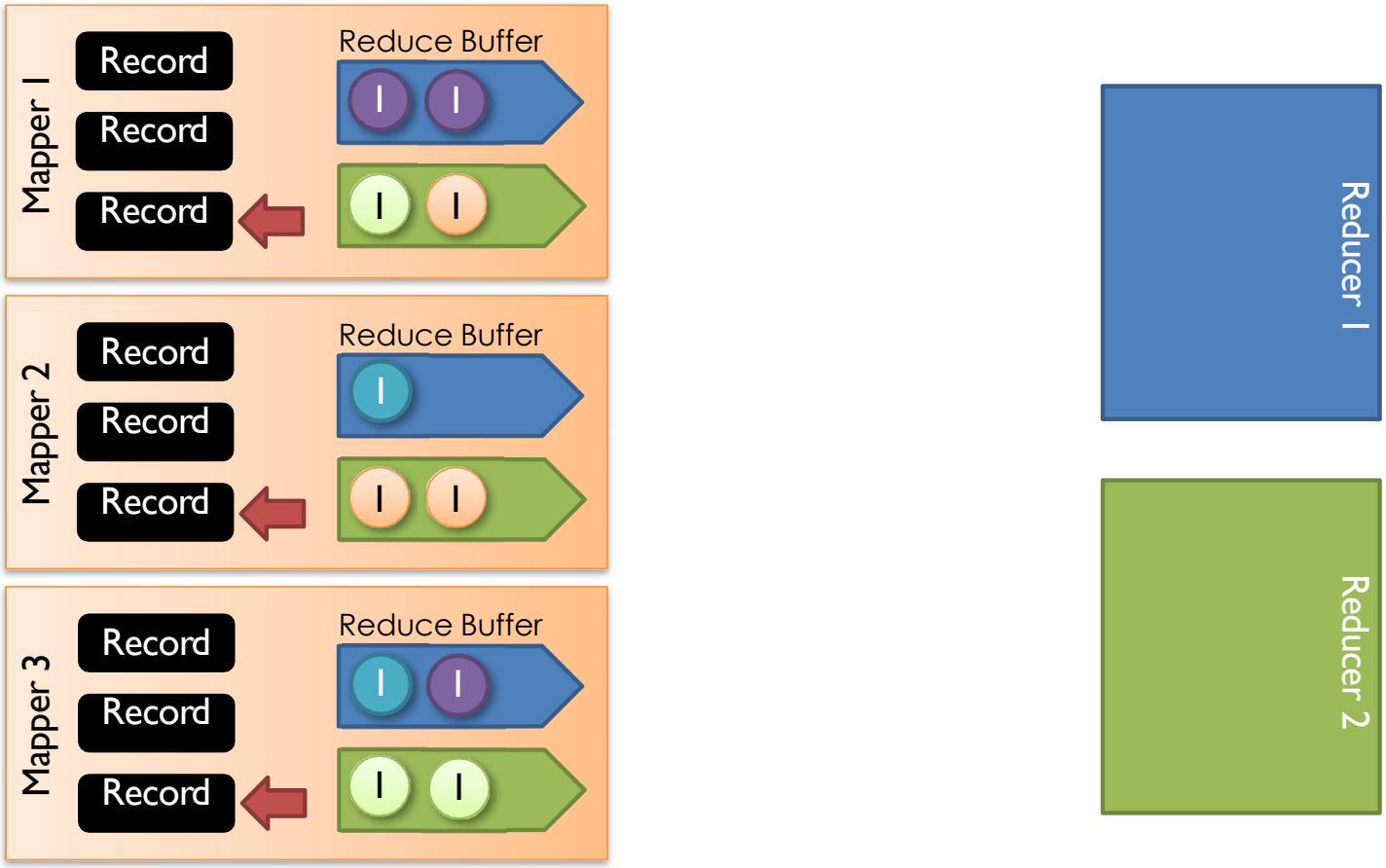
[Dean & Ghemawat, OSDI'04]

The Map Reduce System



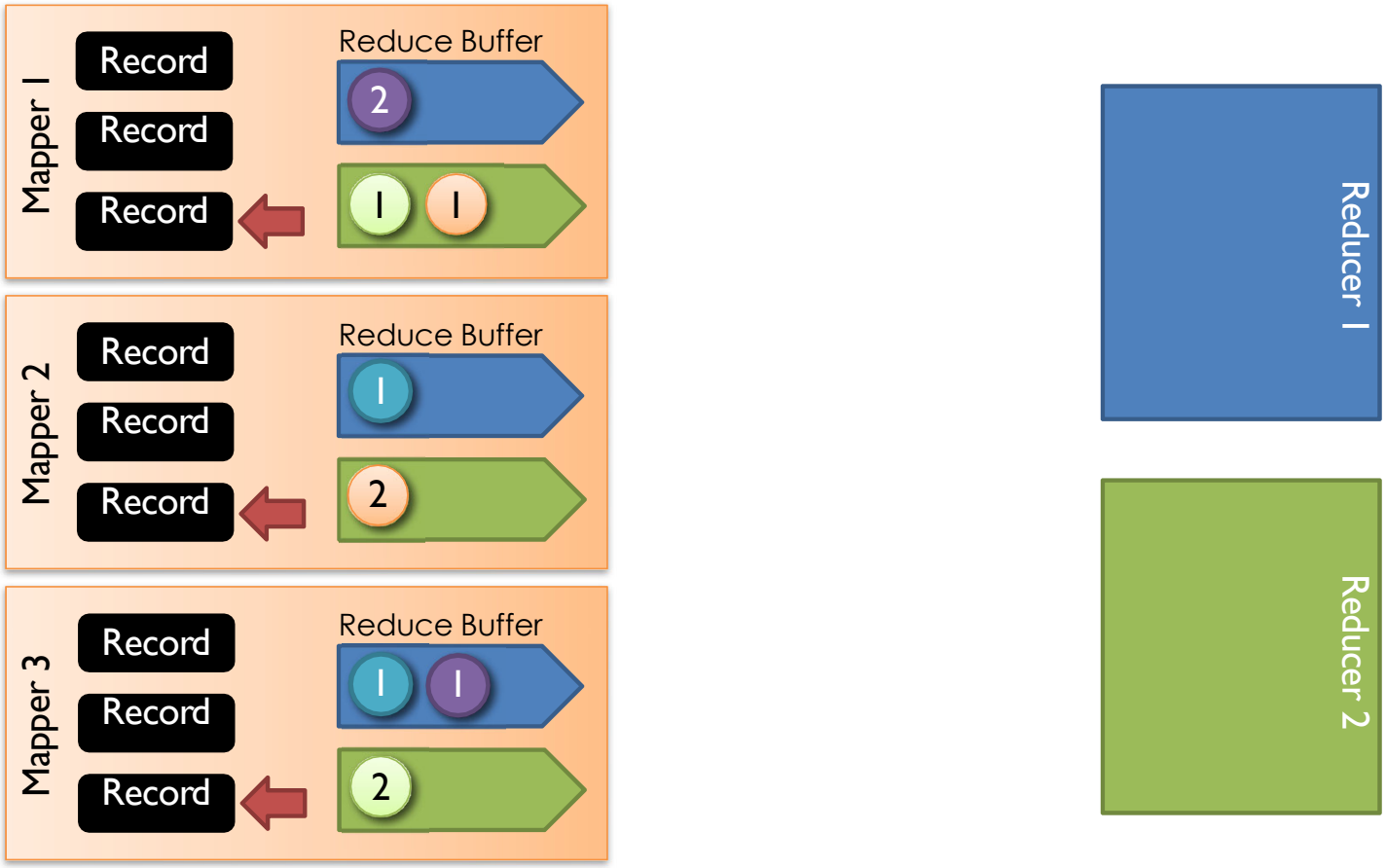
[Dean & Ghemawat, OSDI'04]

The Map Reduce System



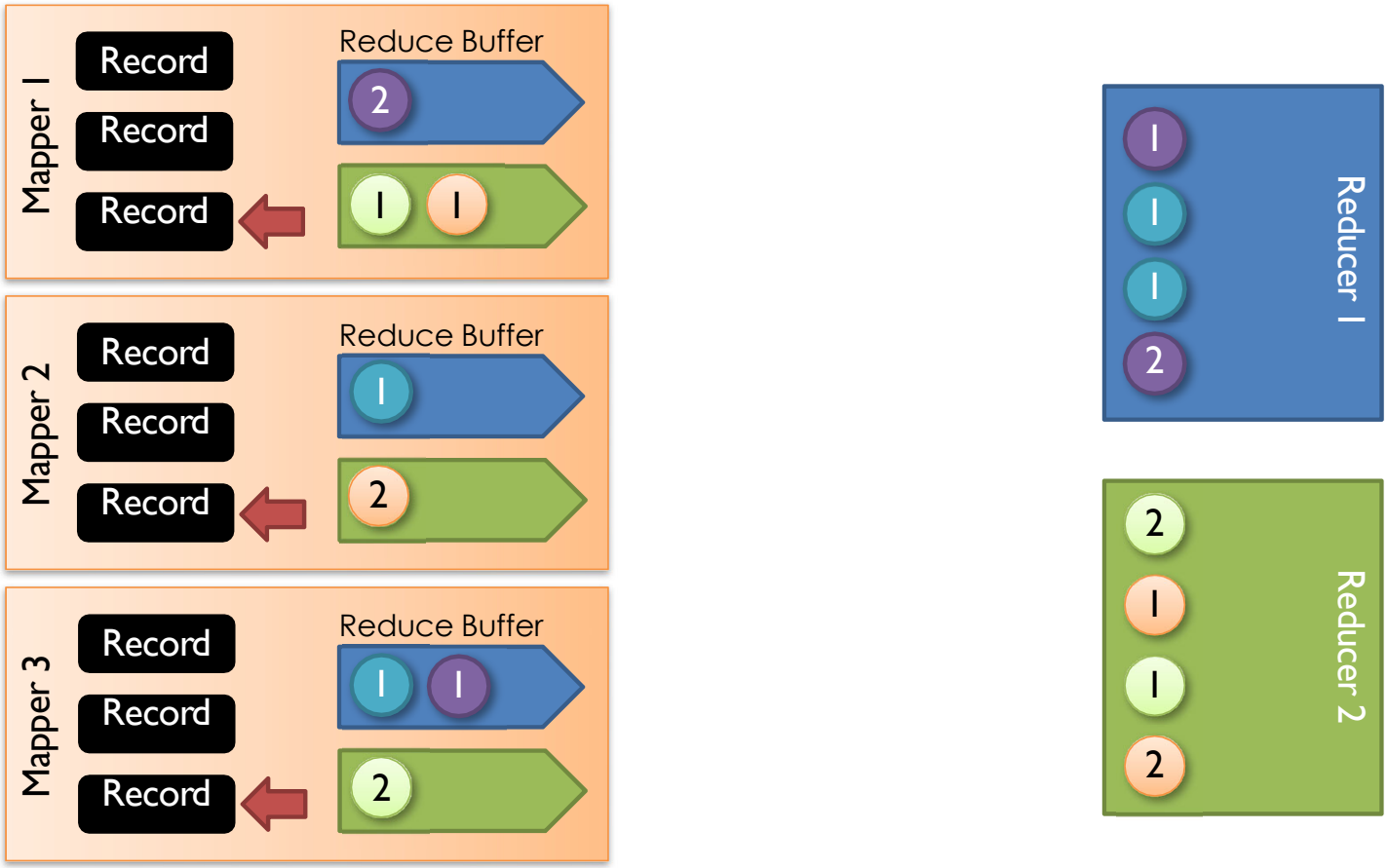
[Dean & Ghemawat, OSDI'04]

The Map Reduce System



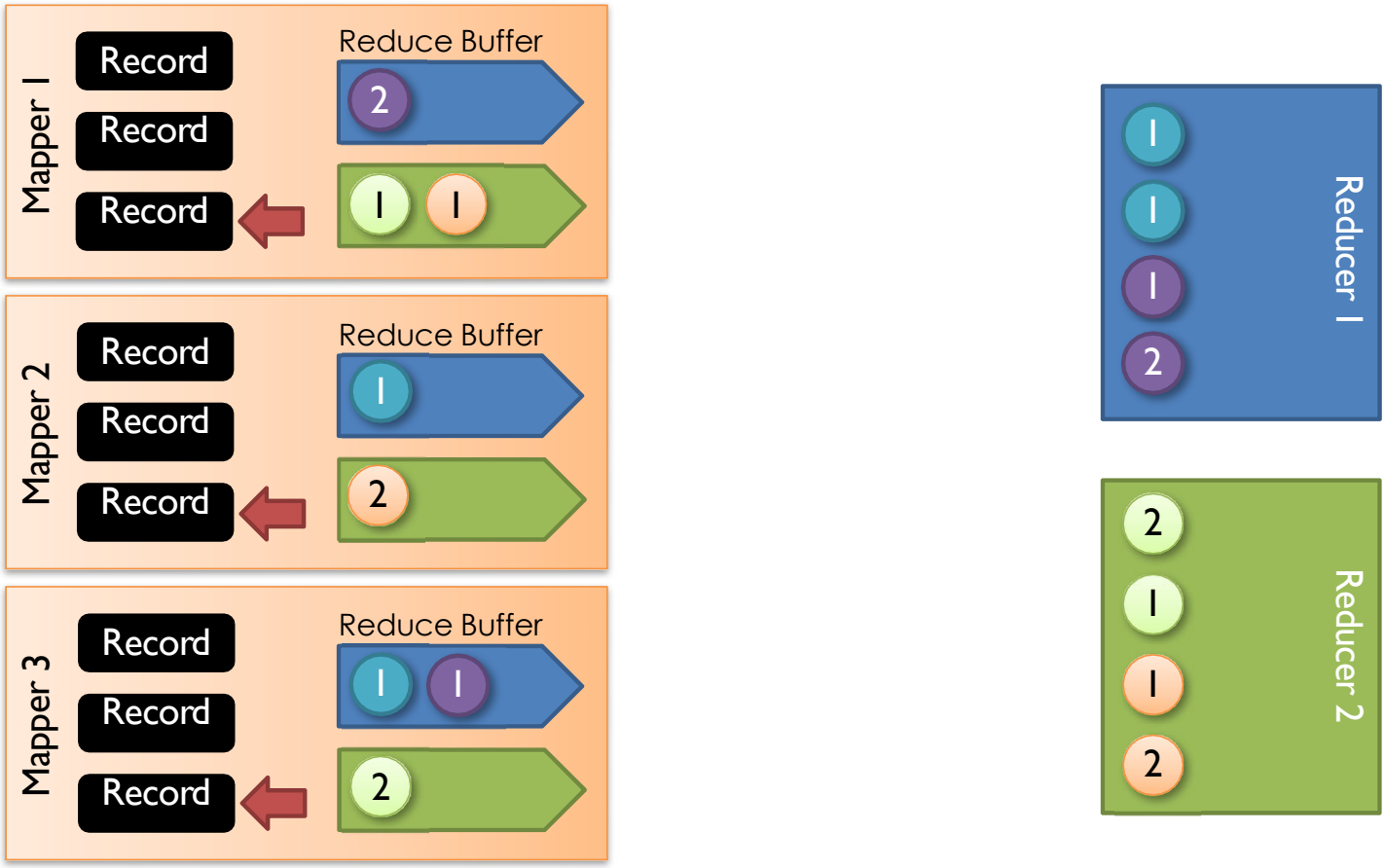
[Dean & Ghemawat, OSDI'04]

The Map Reduce System



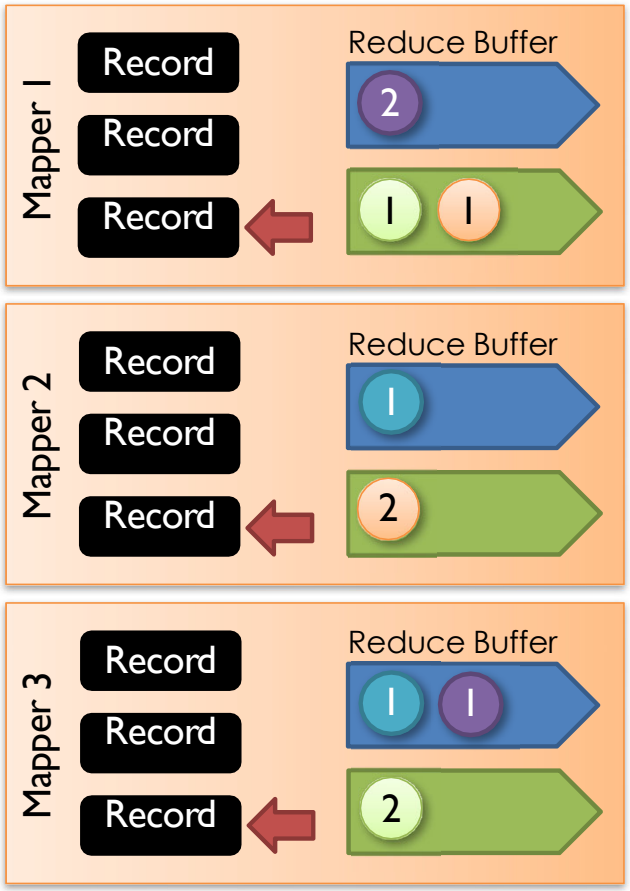
[Dean & Ghemawat, OSDI'04]

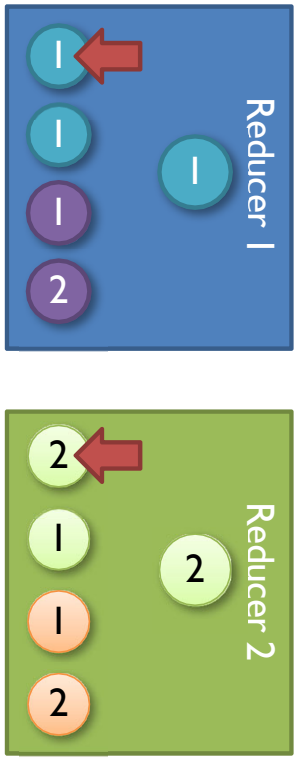
The Map Reduce System



[Dean & Ghemawat, OSDI'04]

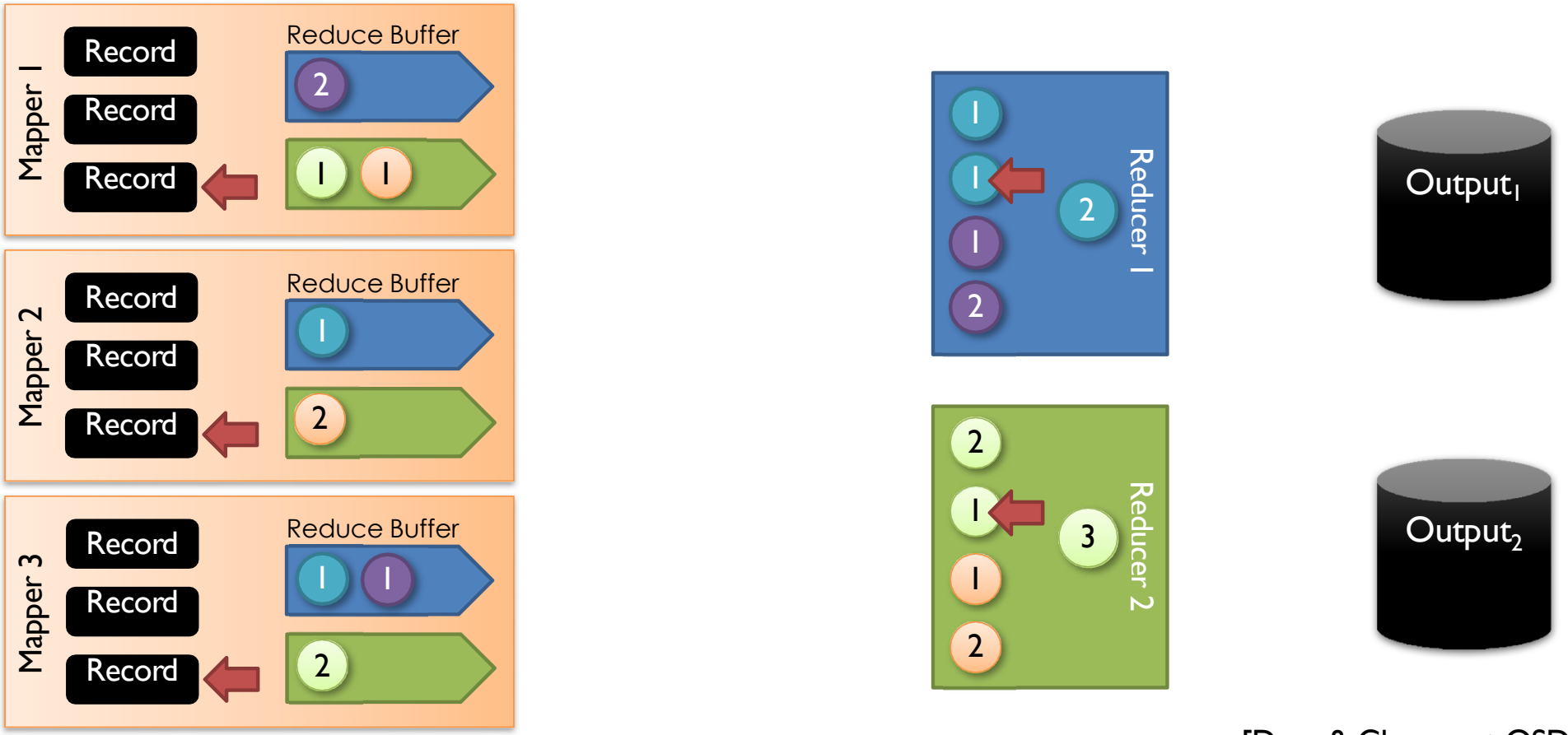
The Map Reduce System





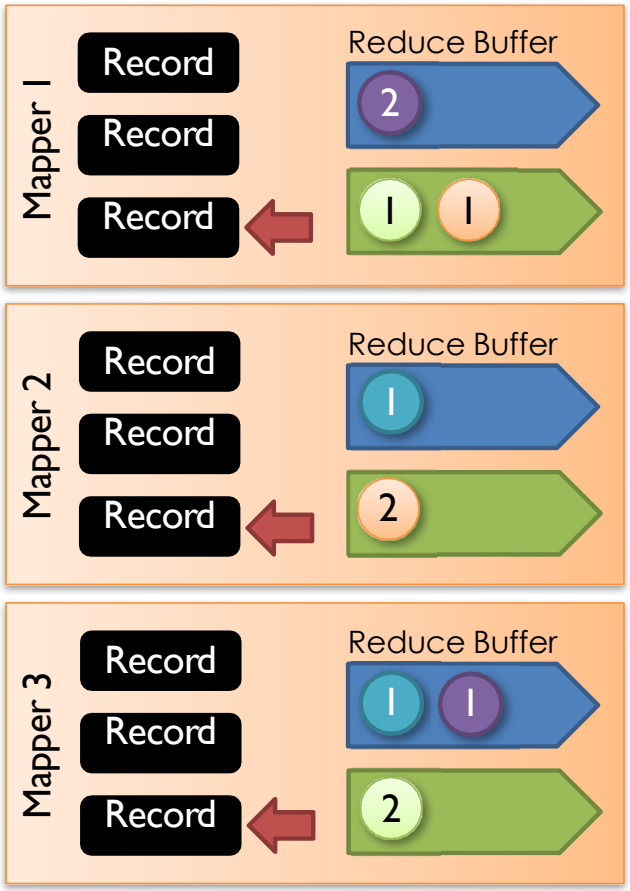
[Dean & Ghemawat, OSDI'04]

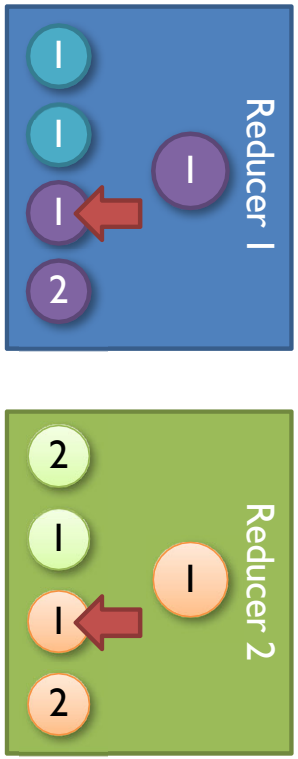
The Map Reduce System

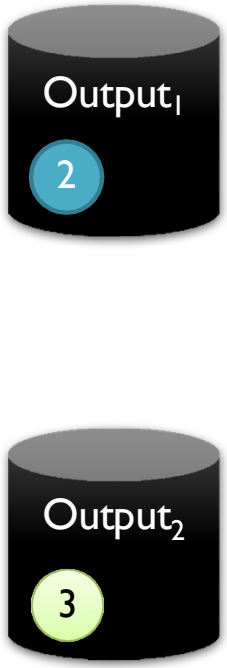


[Dean & Ghemawat, OSDI'04]

The Map Reduce System

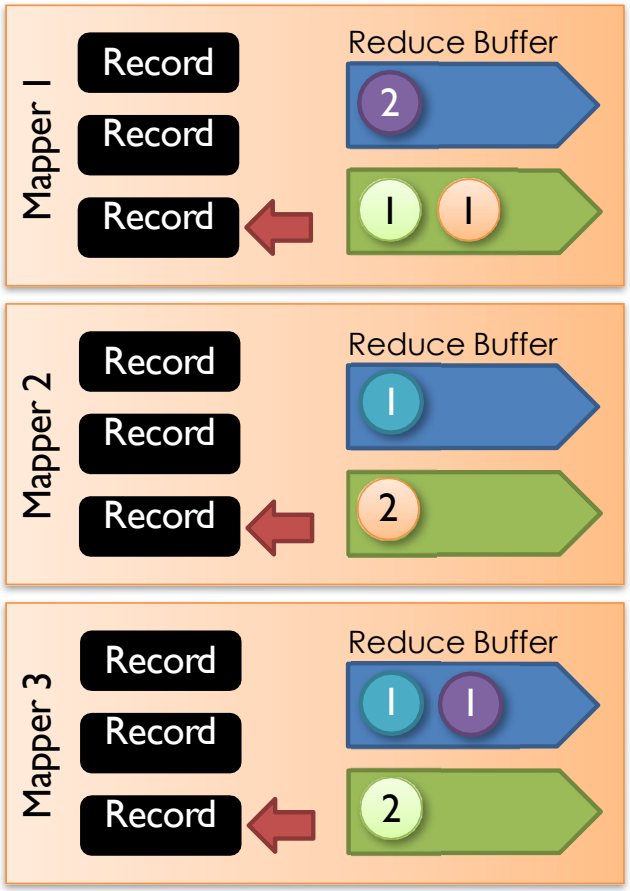


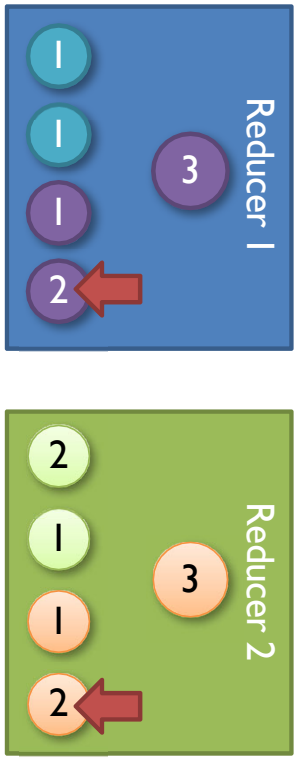


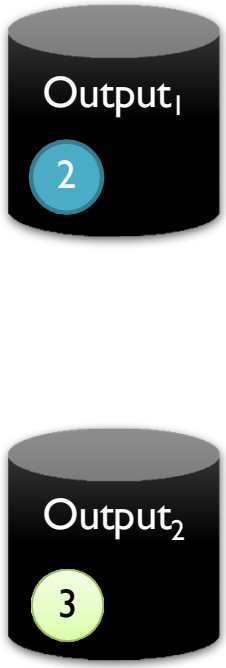


[Dean & Ghemawat, OSDI'04]

The Map Reduce System

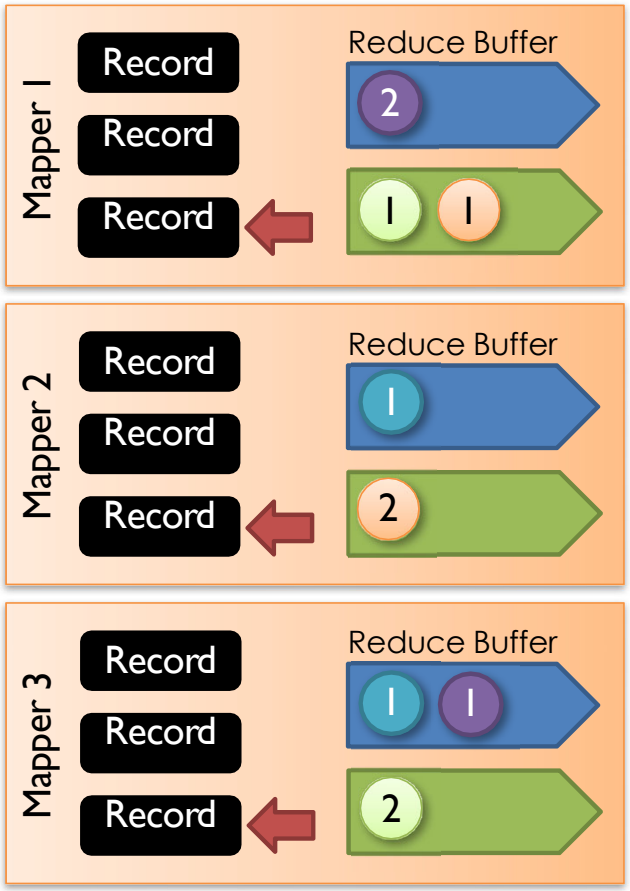


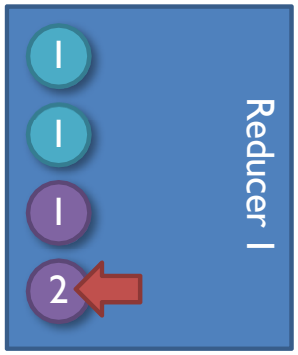


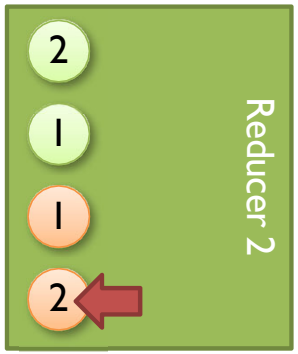


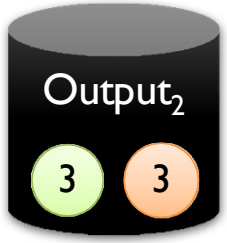
[Dean & Ghemawat, OSDI'04]

The Map Reduce System



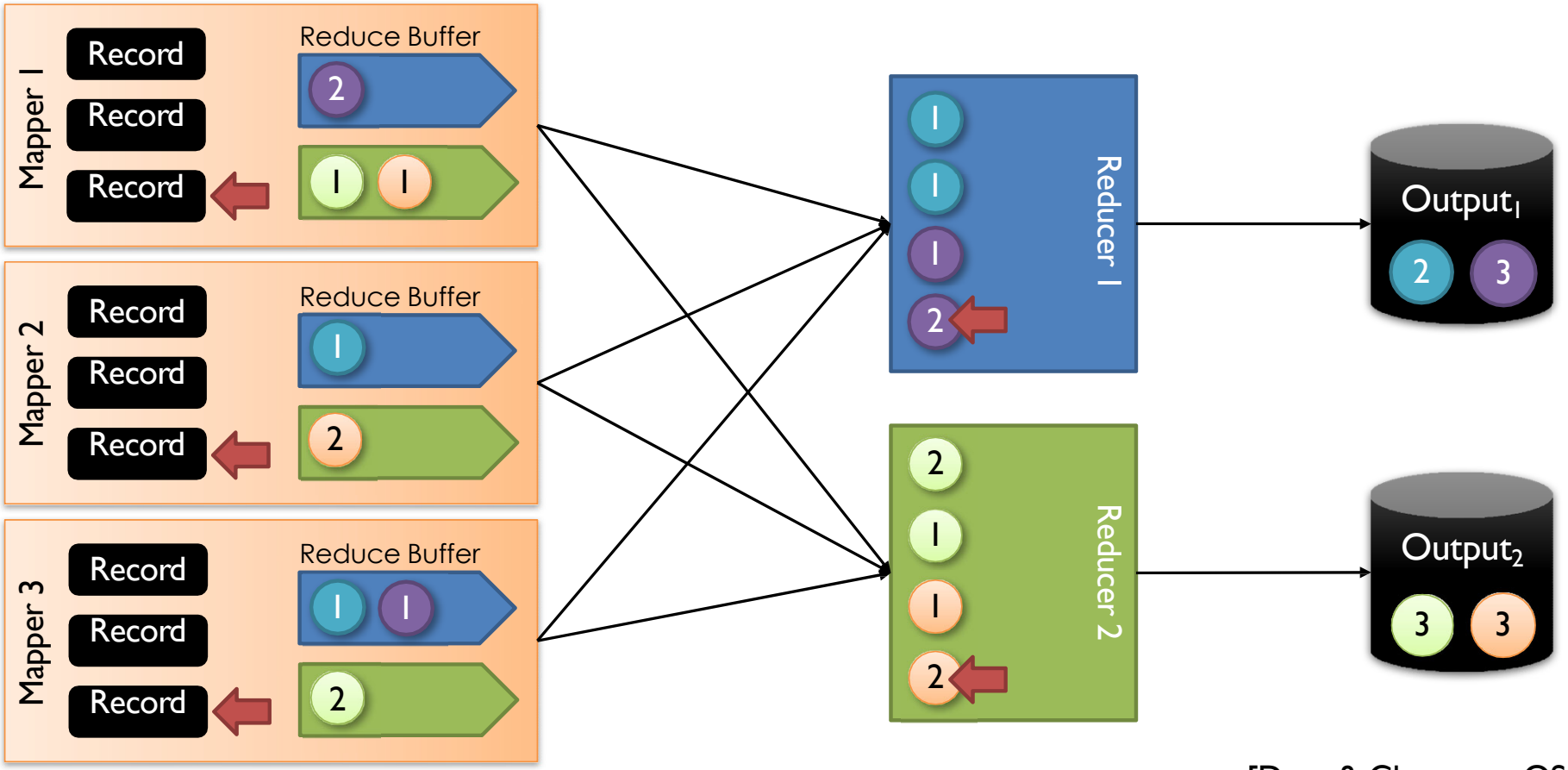






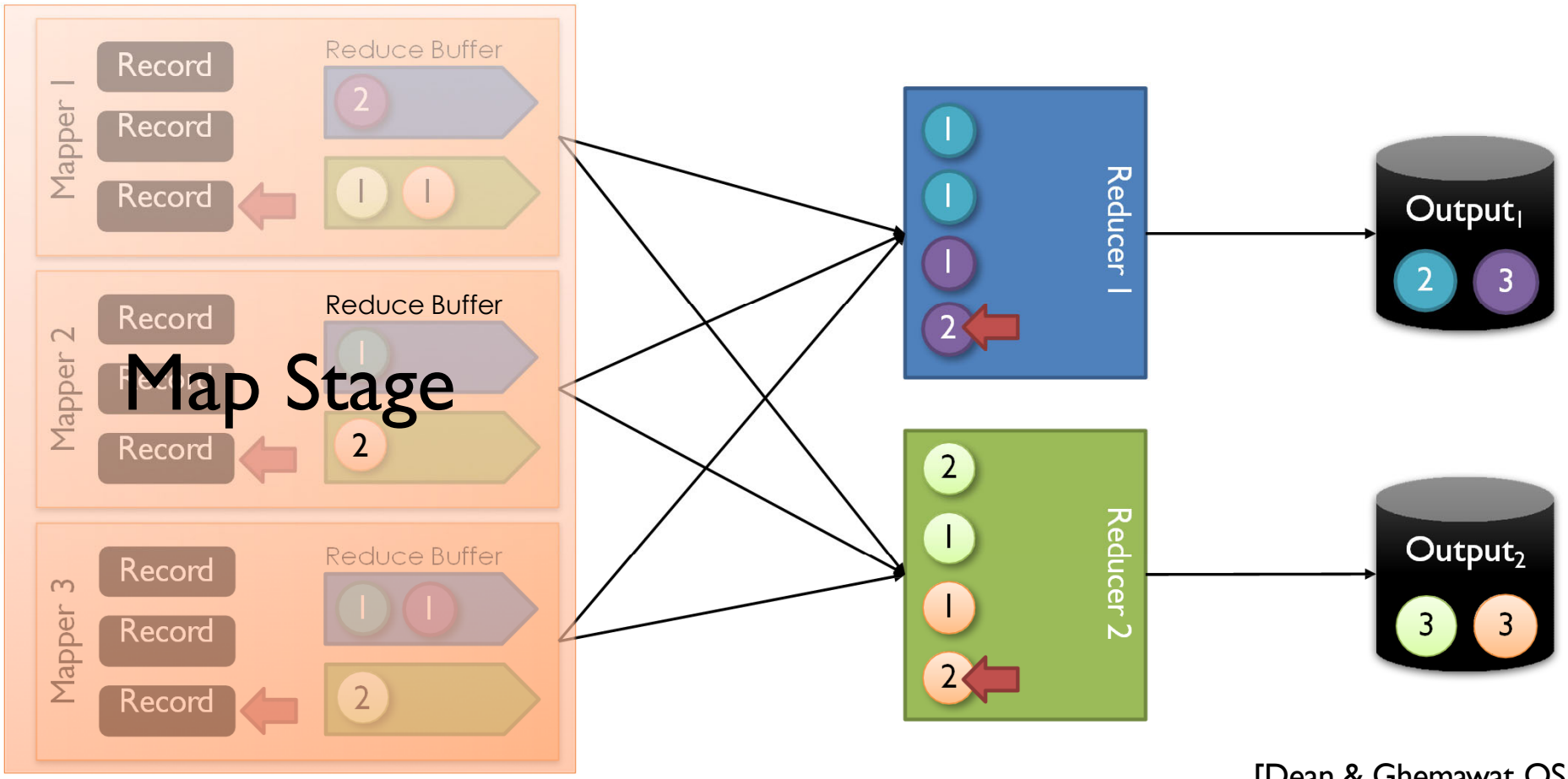
[Dean & Ghemawat, OSDI'04]

The Map Reduce System



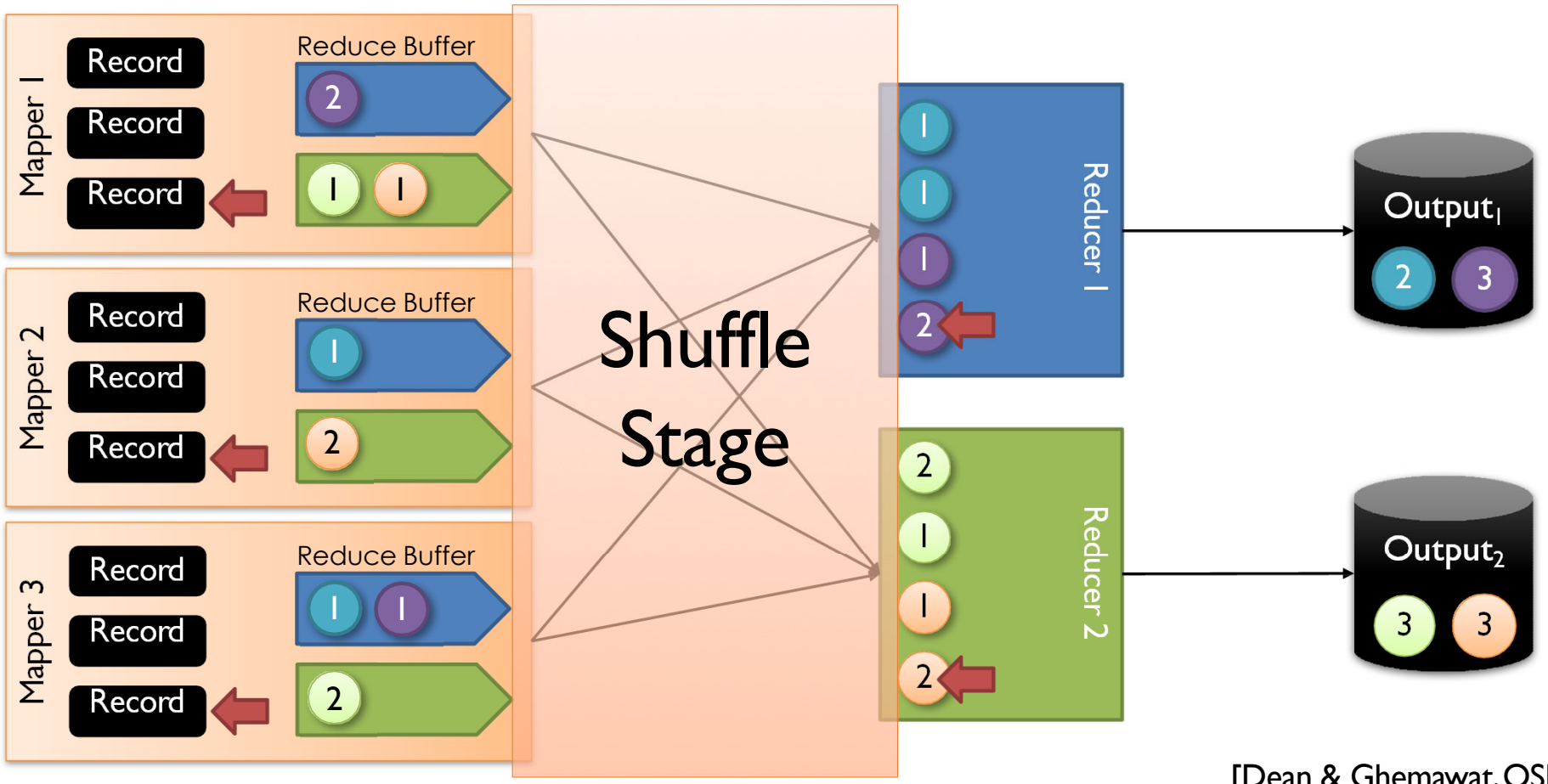
[Dean & Ghemawat, OSDI'04]

The Map Reduce System



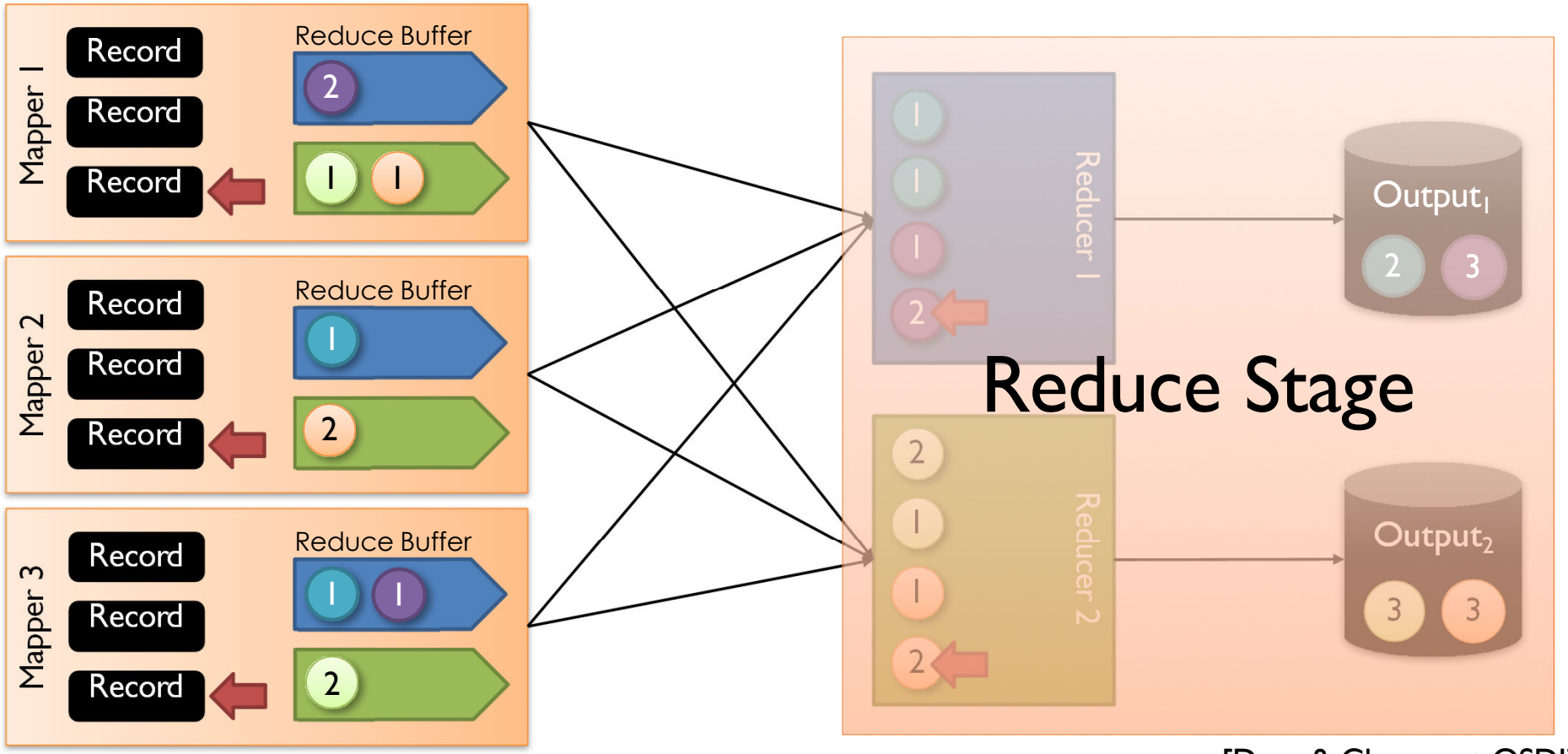
[Dean & Ghemawat, OSDI'04]

The Map Reduce System



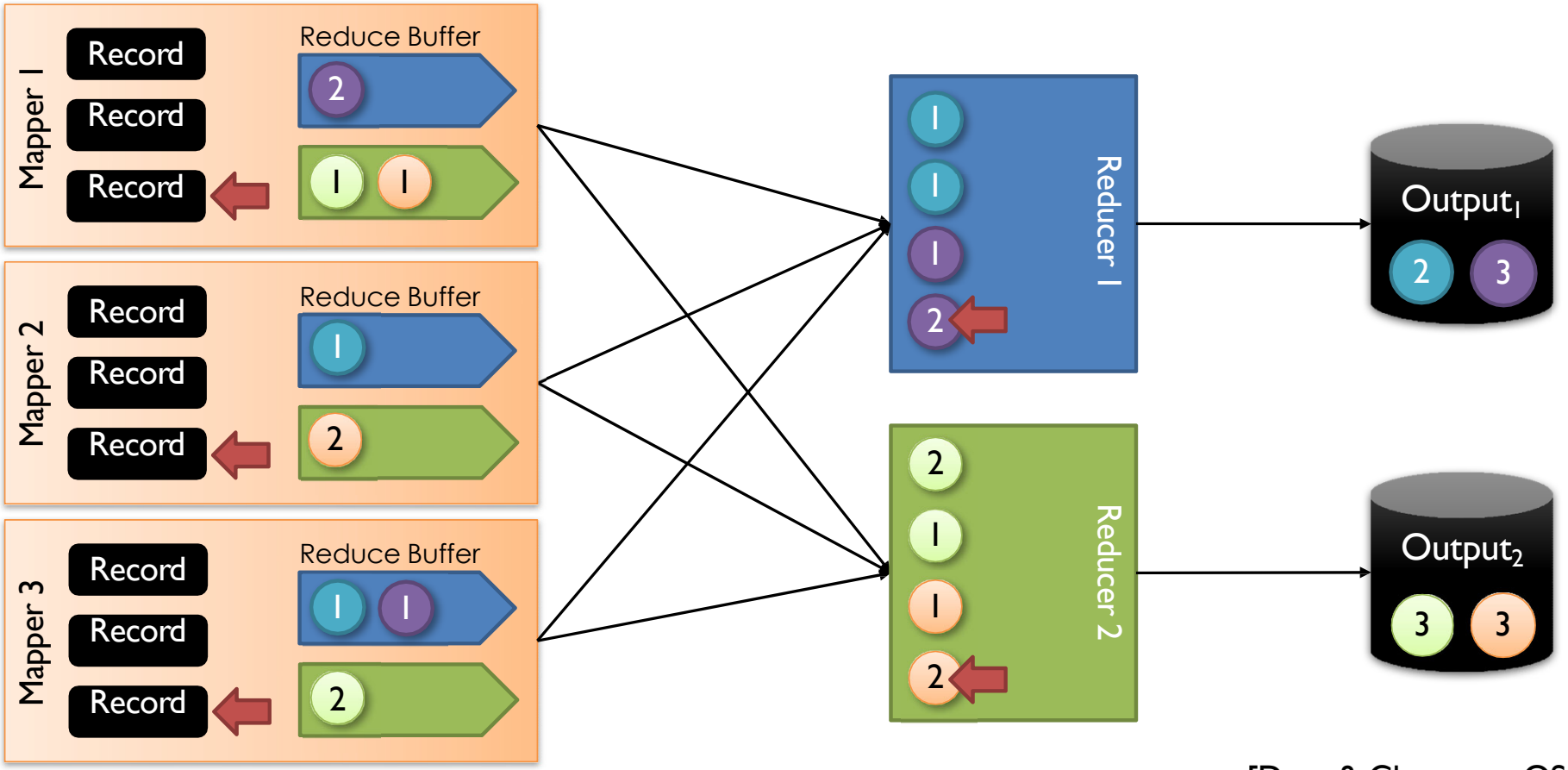
[Dean & Ghemawat, OSDI'04]

The Map Reduce System



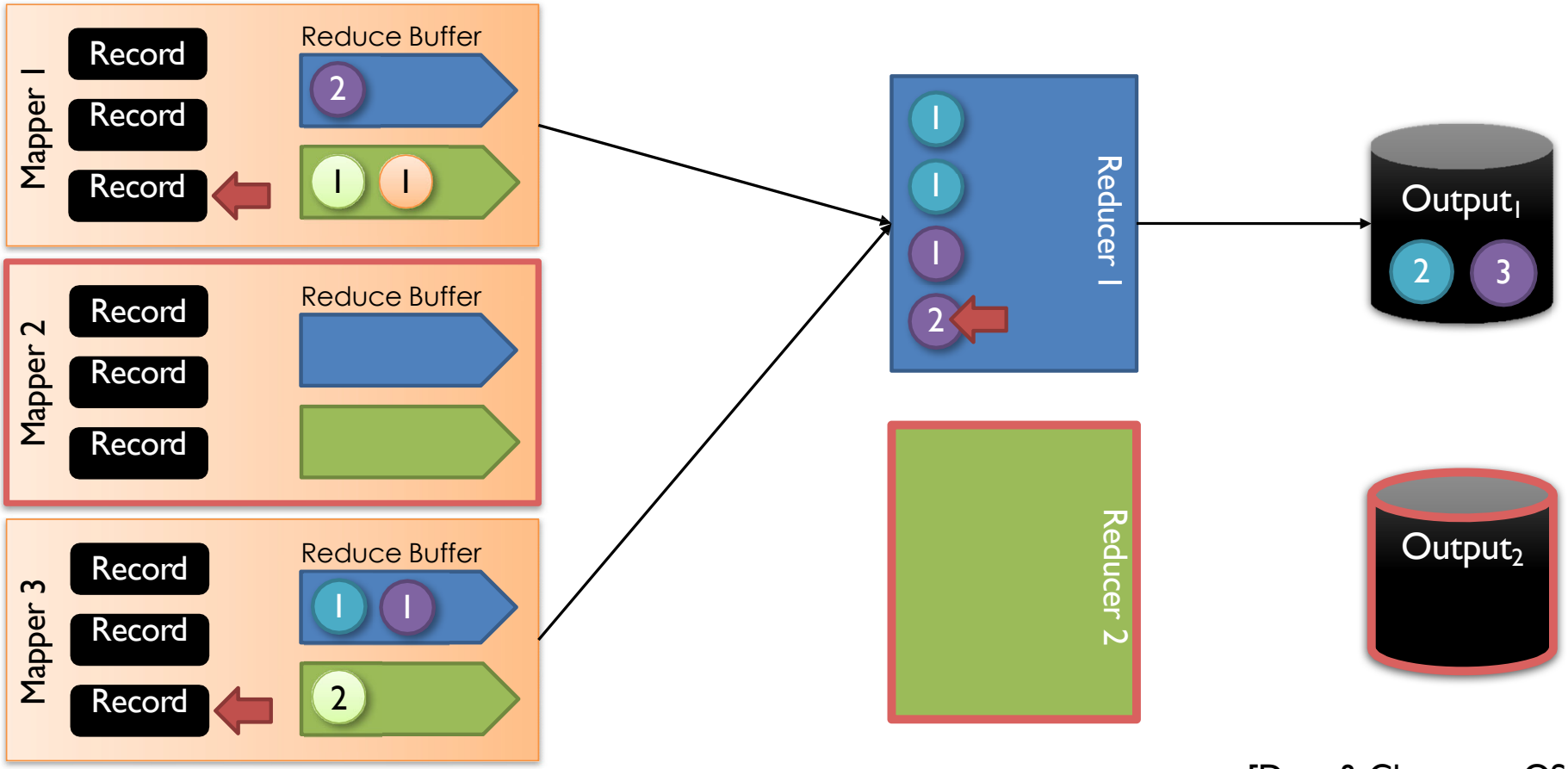
[Dean & Ghemawat, OSDI'04]

The Map Reduce: **Fault Tolerance**



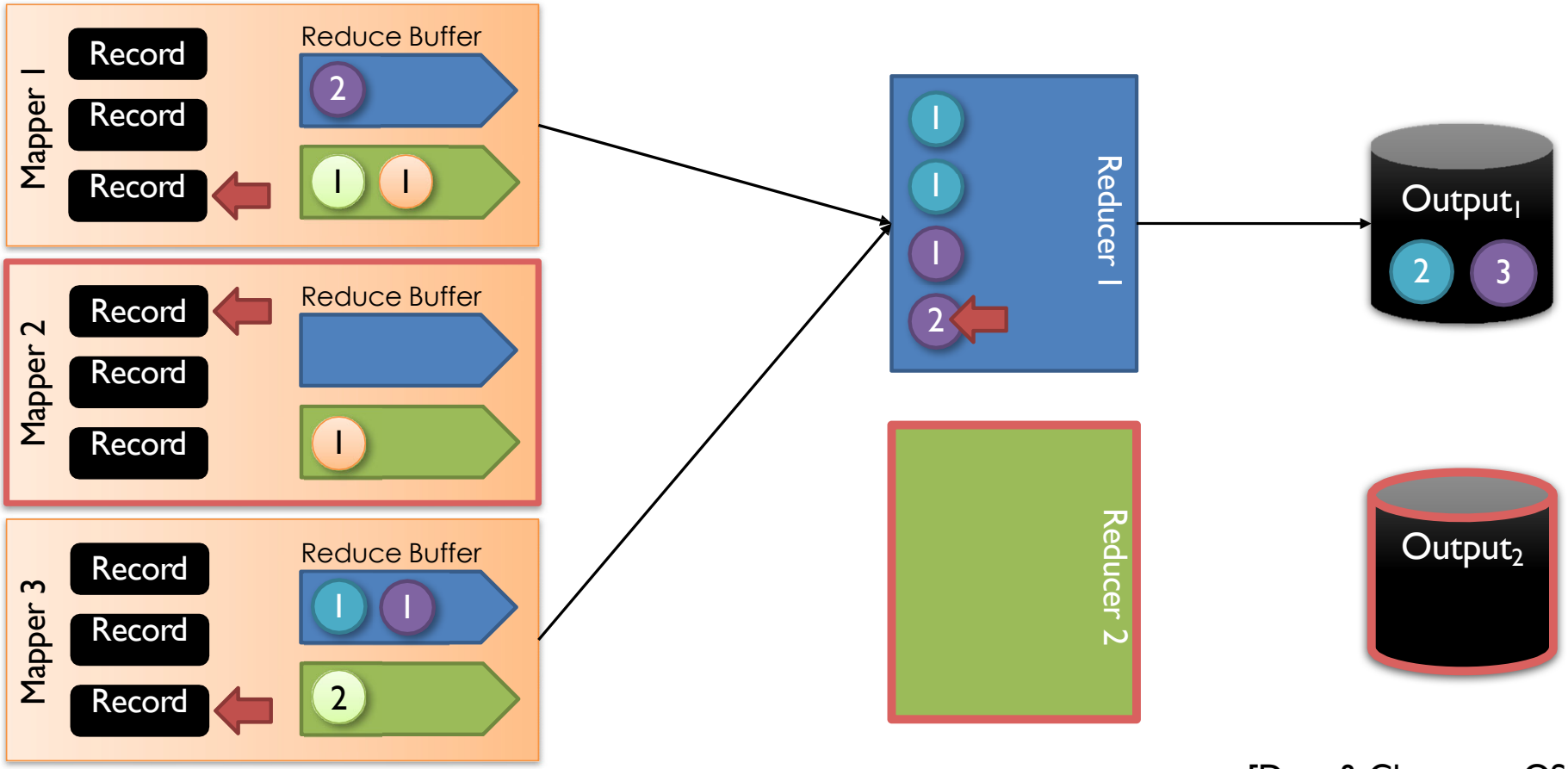
[Dean & Ghemawat, OSDI'04]

The Map Reduce: **Fault Tolerance**



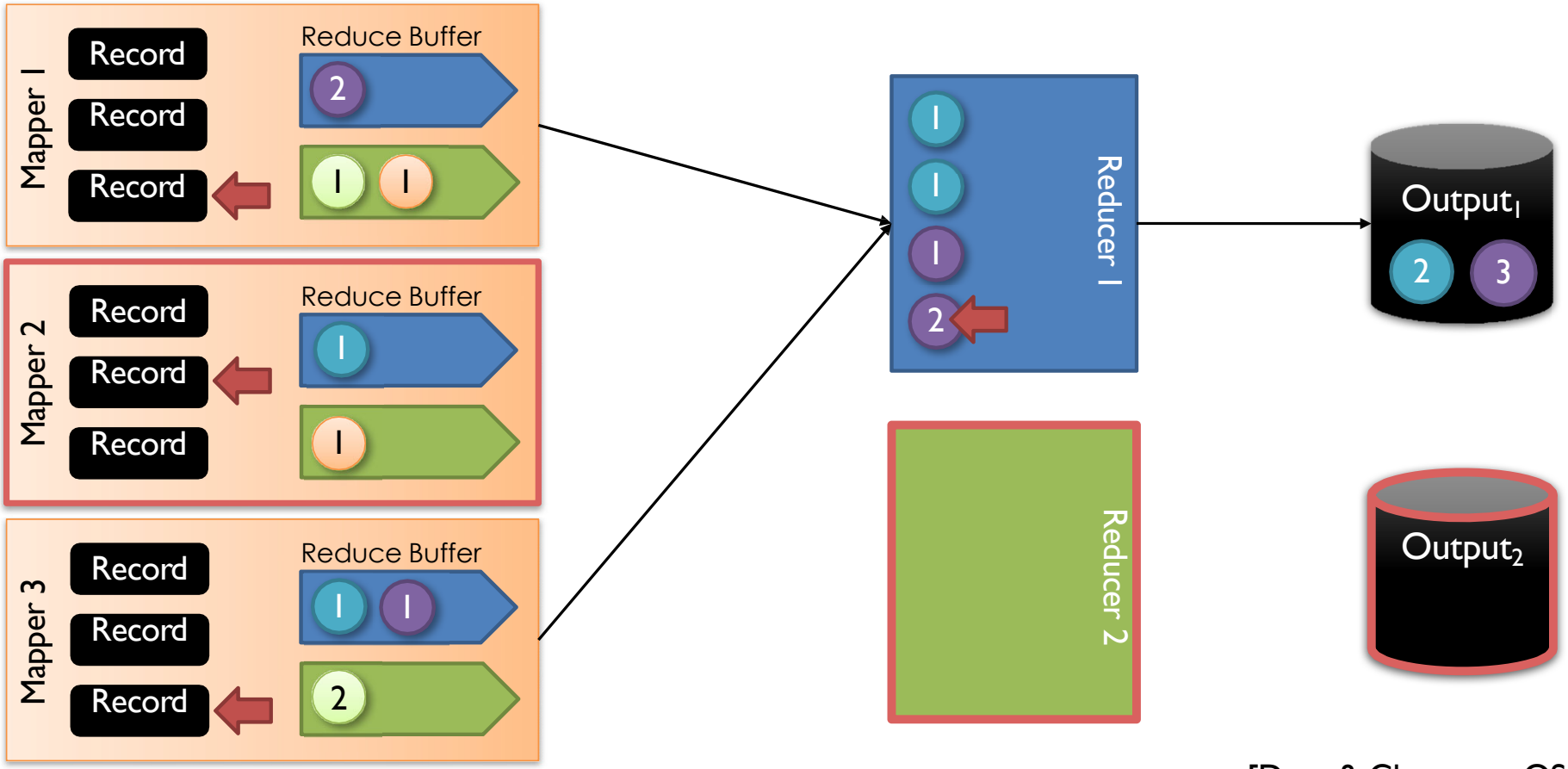
[Dean & Ghemawat, OSDI'04]

The Map Reduce: **Fault Tolerance**



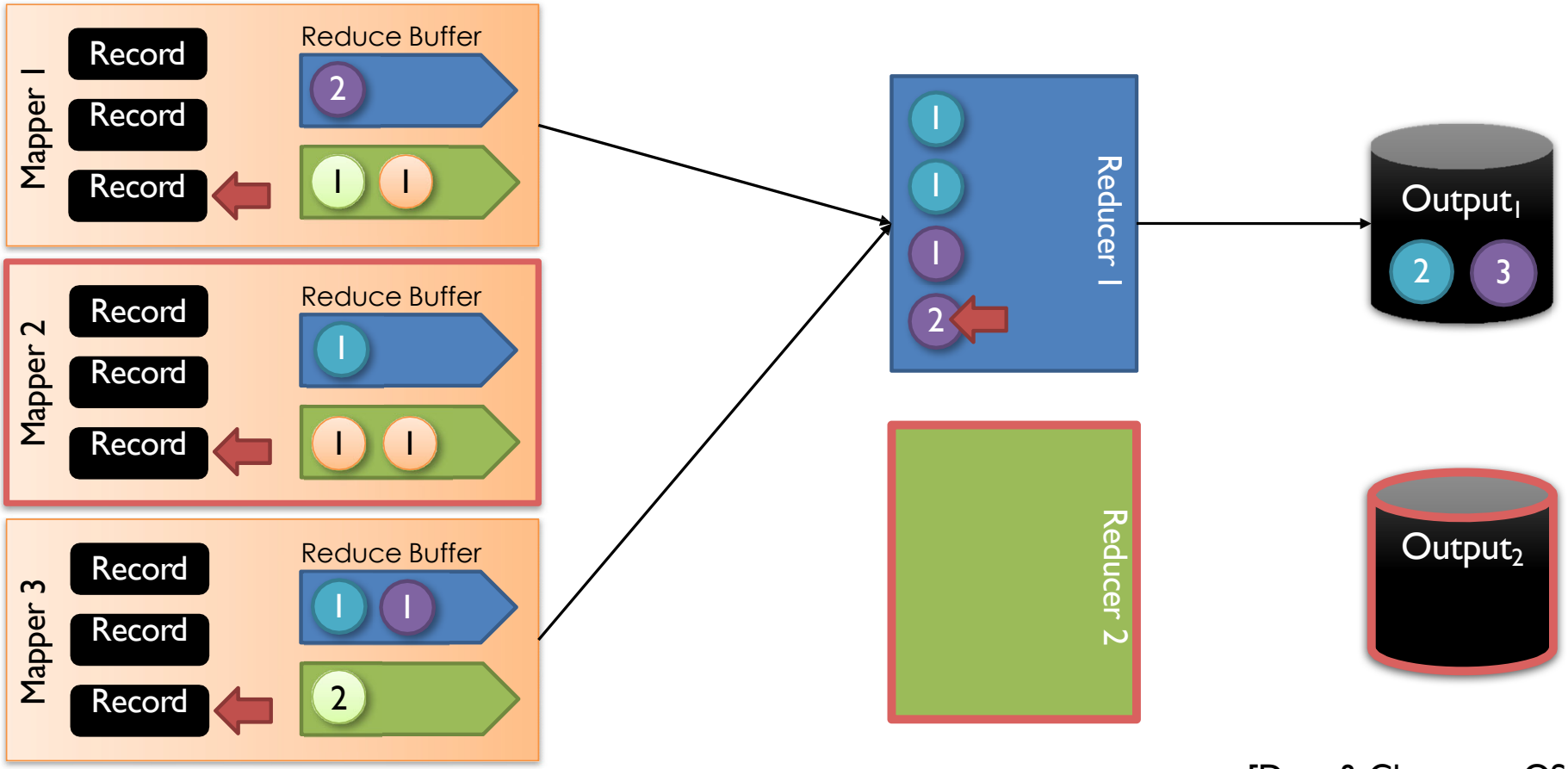
[Dean & Ghemawat, OSDI'04]

The Map Reduce: **Fault Tolerance**



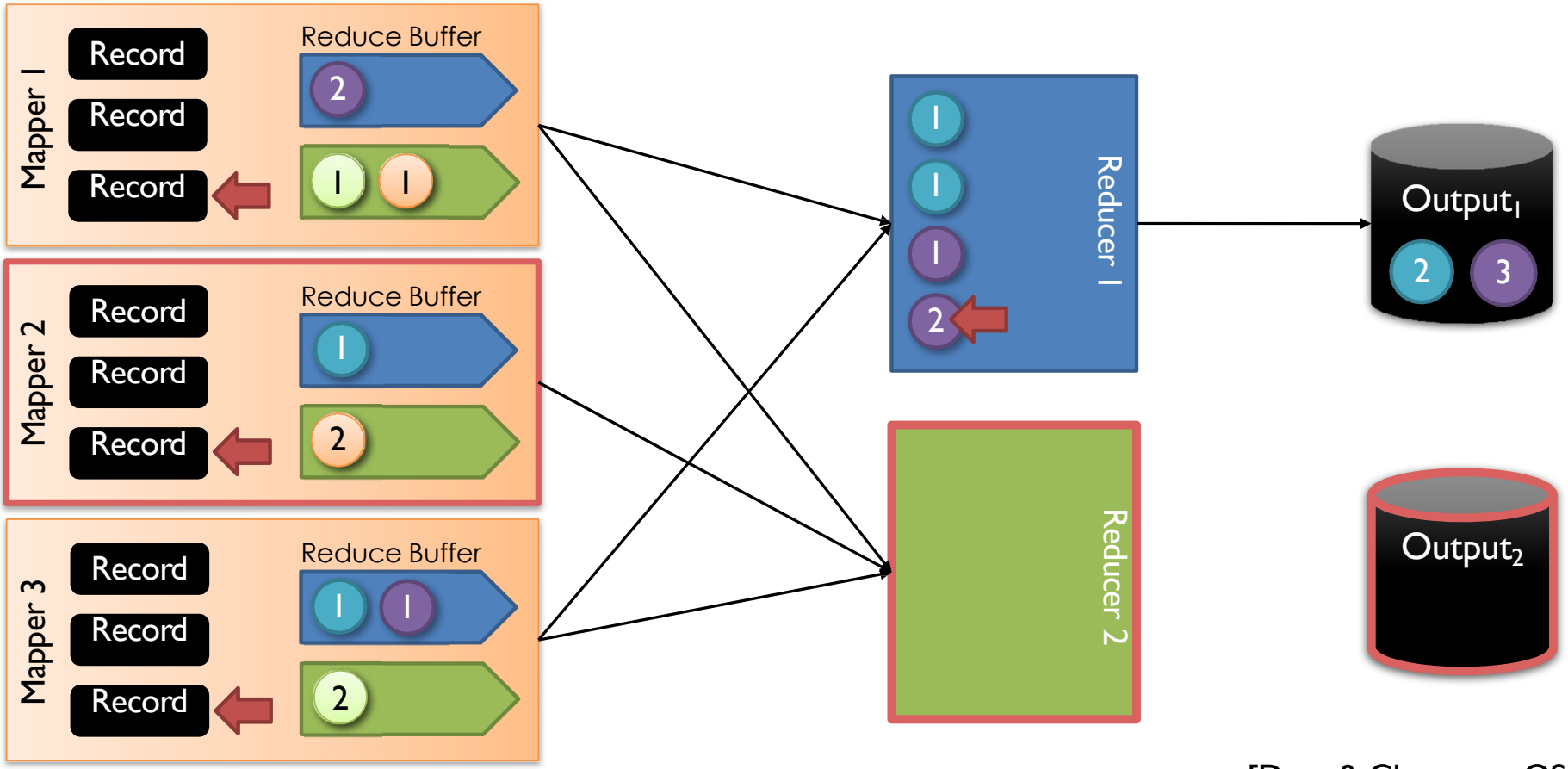
[Dean & Ghemawat, OSDI'04]

The Map Reduce: **Fault Tolerance**



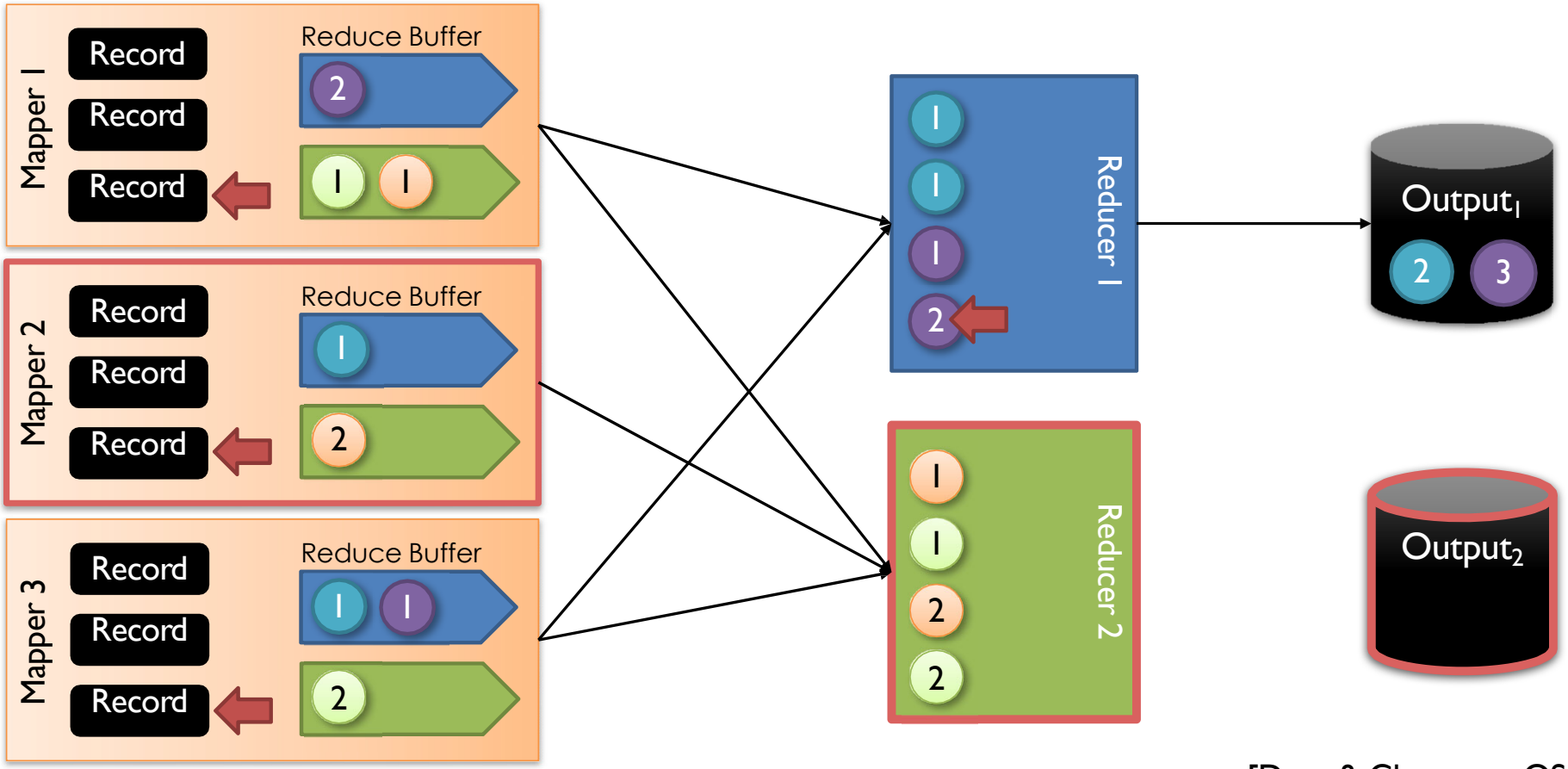
[Dean & Ghemawat, OSDI'04]

The Map Reduce: **Fault Tolerance**



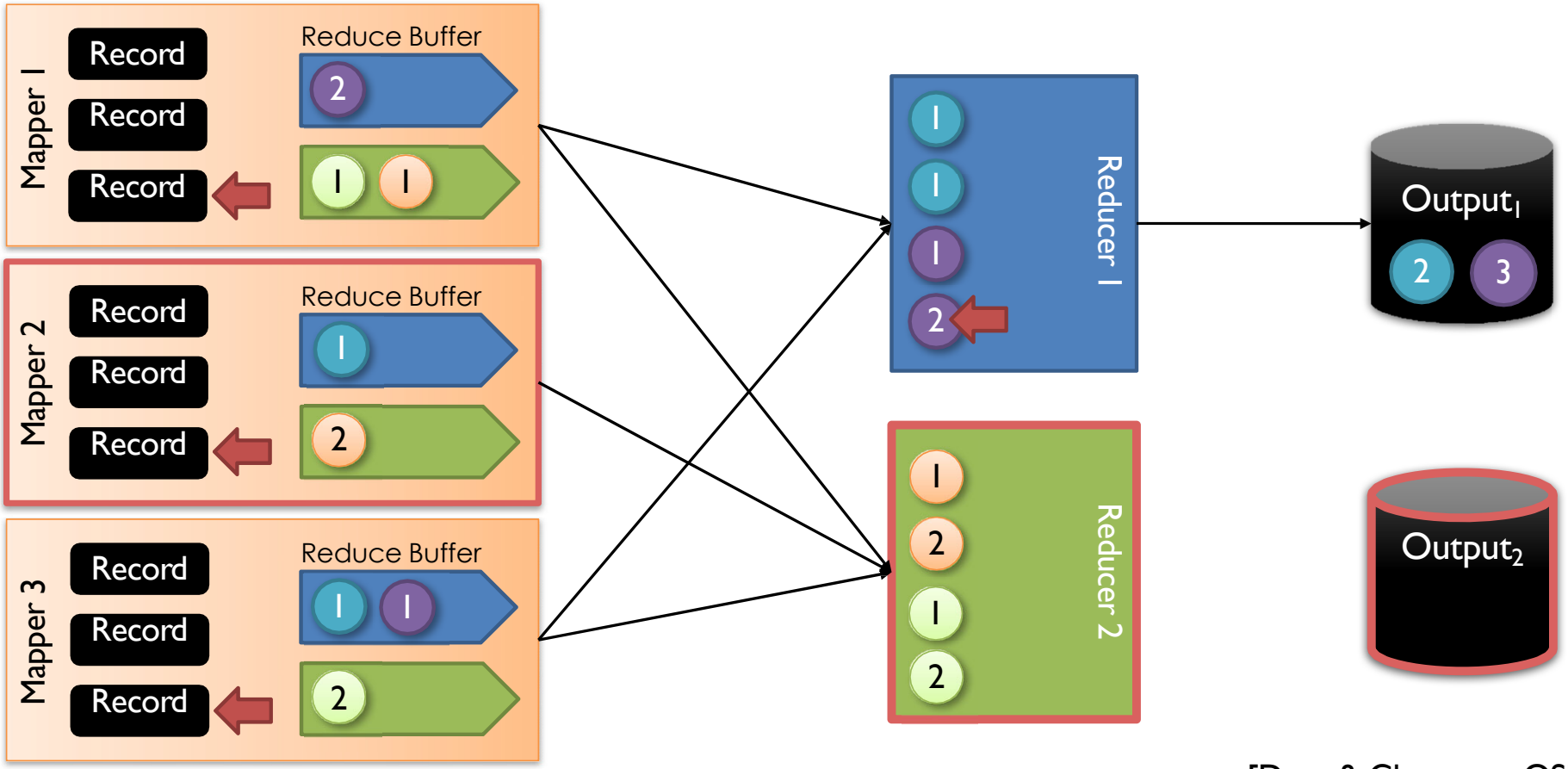
[Dean & Ghemawat, OSDI'04]

The Map Reduce: **Fault Tolerance**



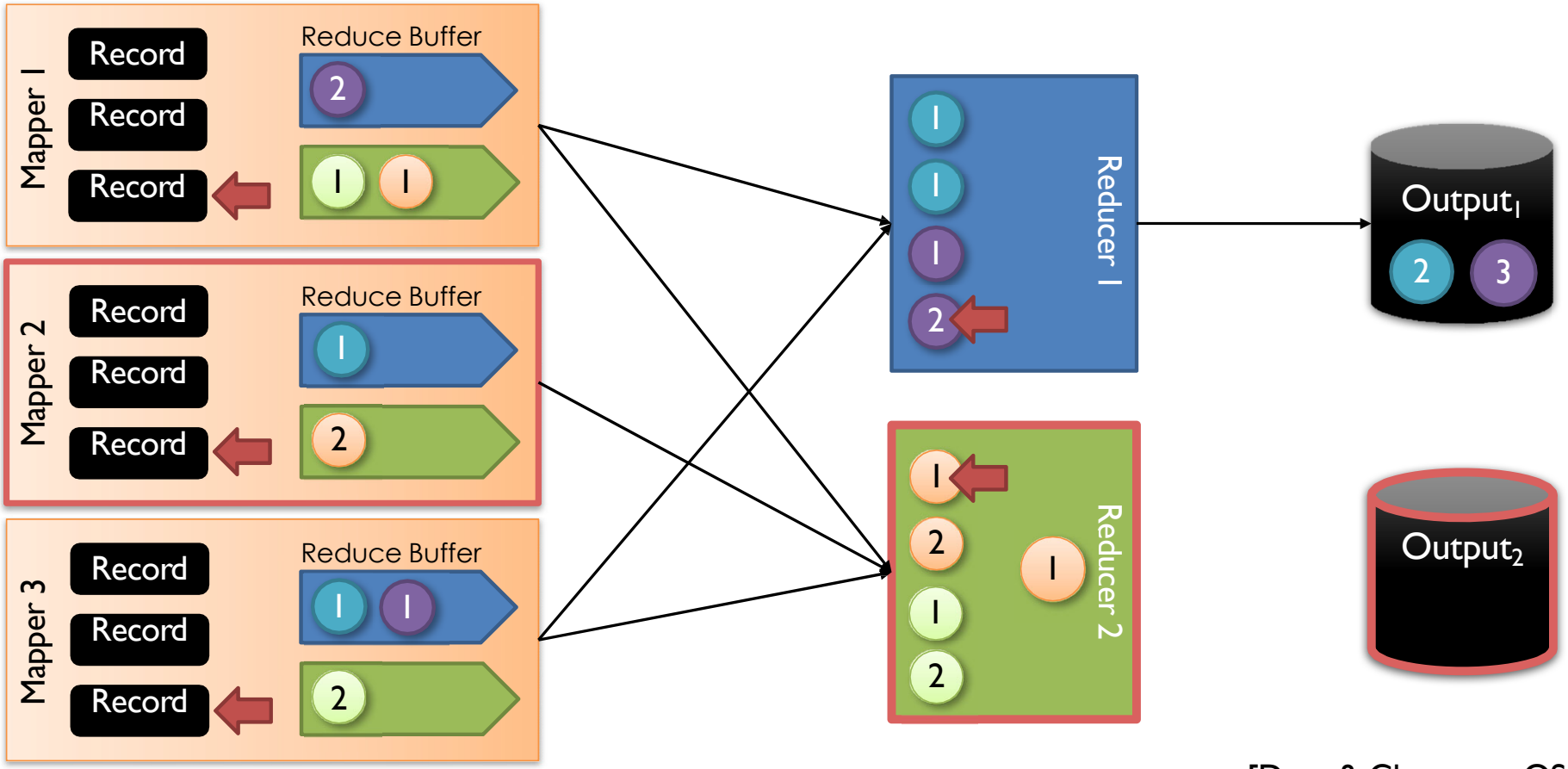
[Dean & Ghemawat, OSDI'04]

The Map Reduce: **Fault Tolerance**



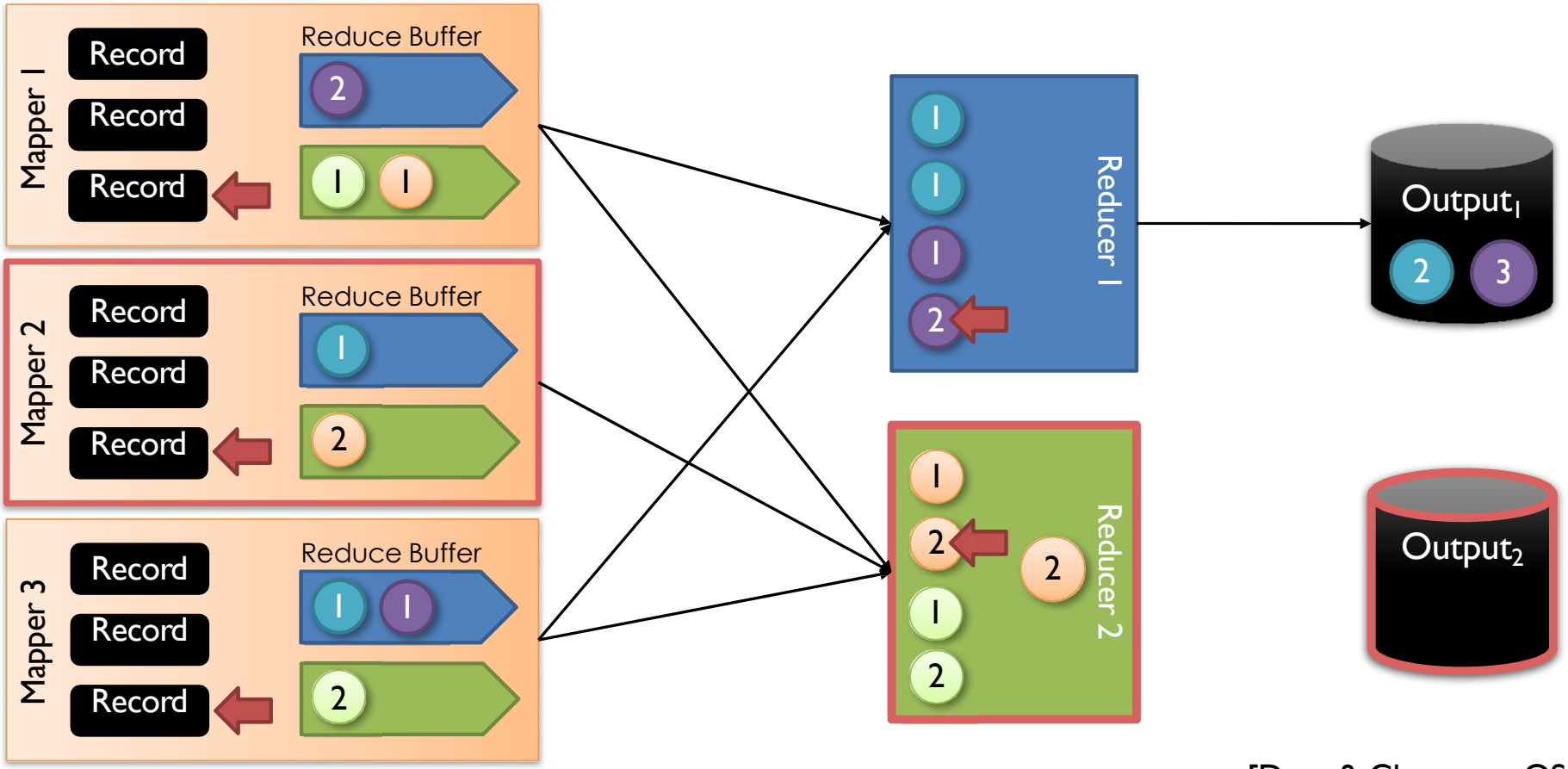
[Dean & Ghemawat, OSDI'04]

The Map Reduce: **Fault Tolerance**



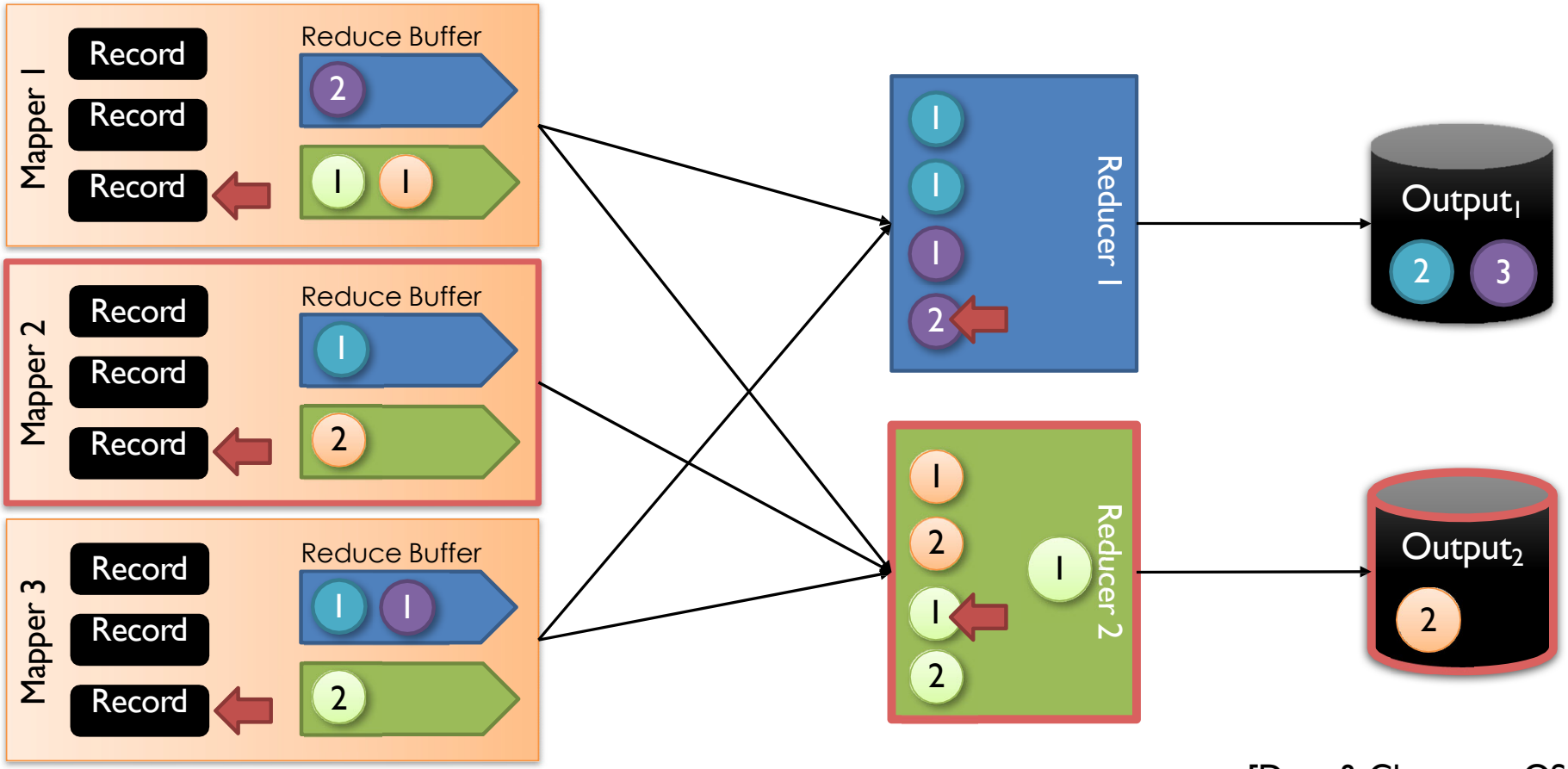
[Dean & Ghemawat, OSDI'04]

The Map Reduce: **Fault Tolerance**



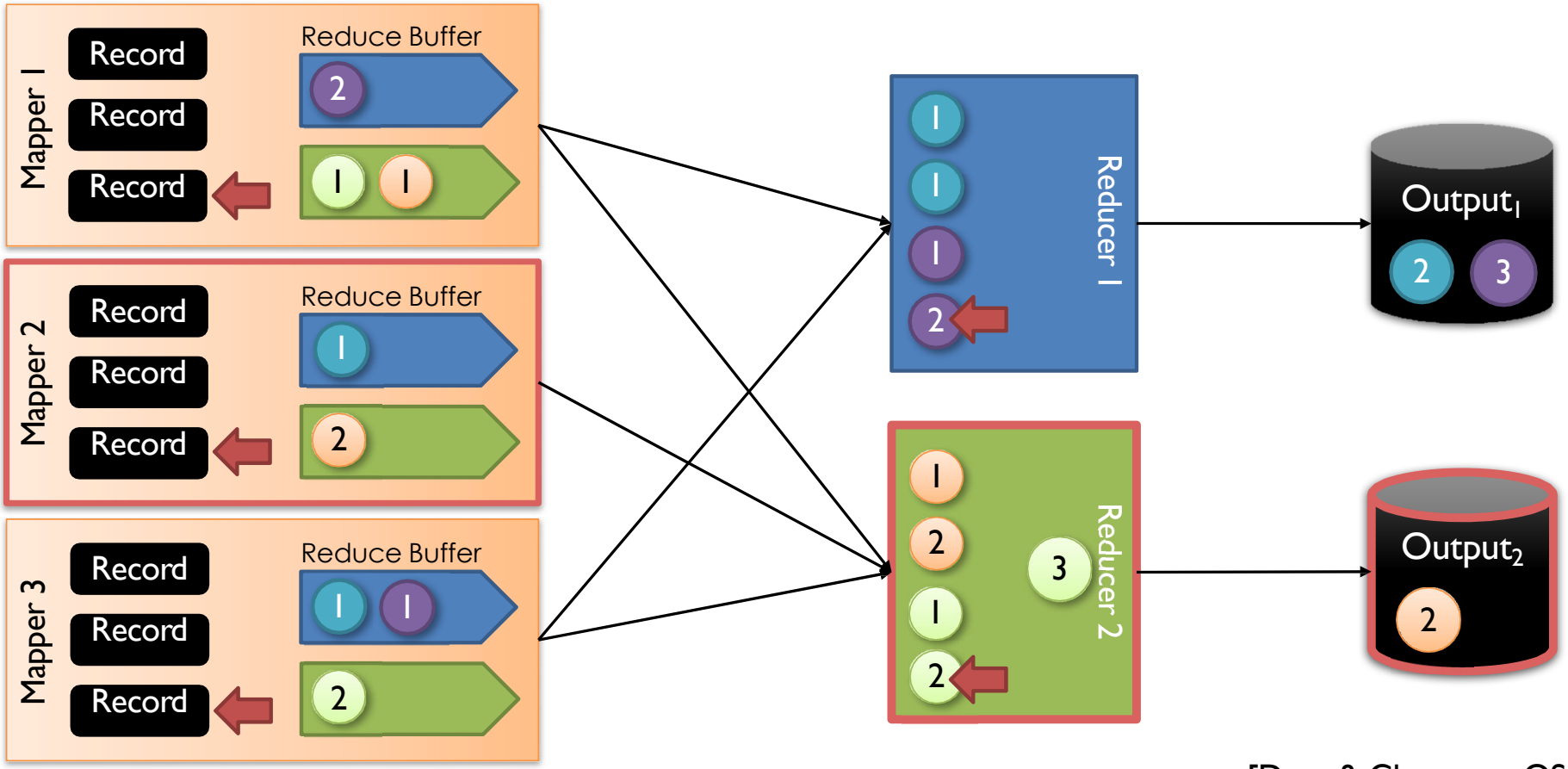
[Dean & Ghemawat, OSDI'04]

The Map Reduce: **Fault Tolerance**



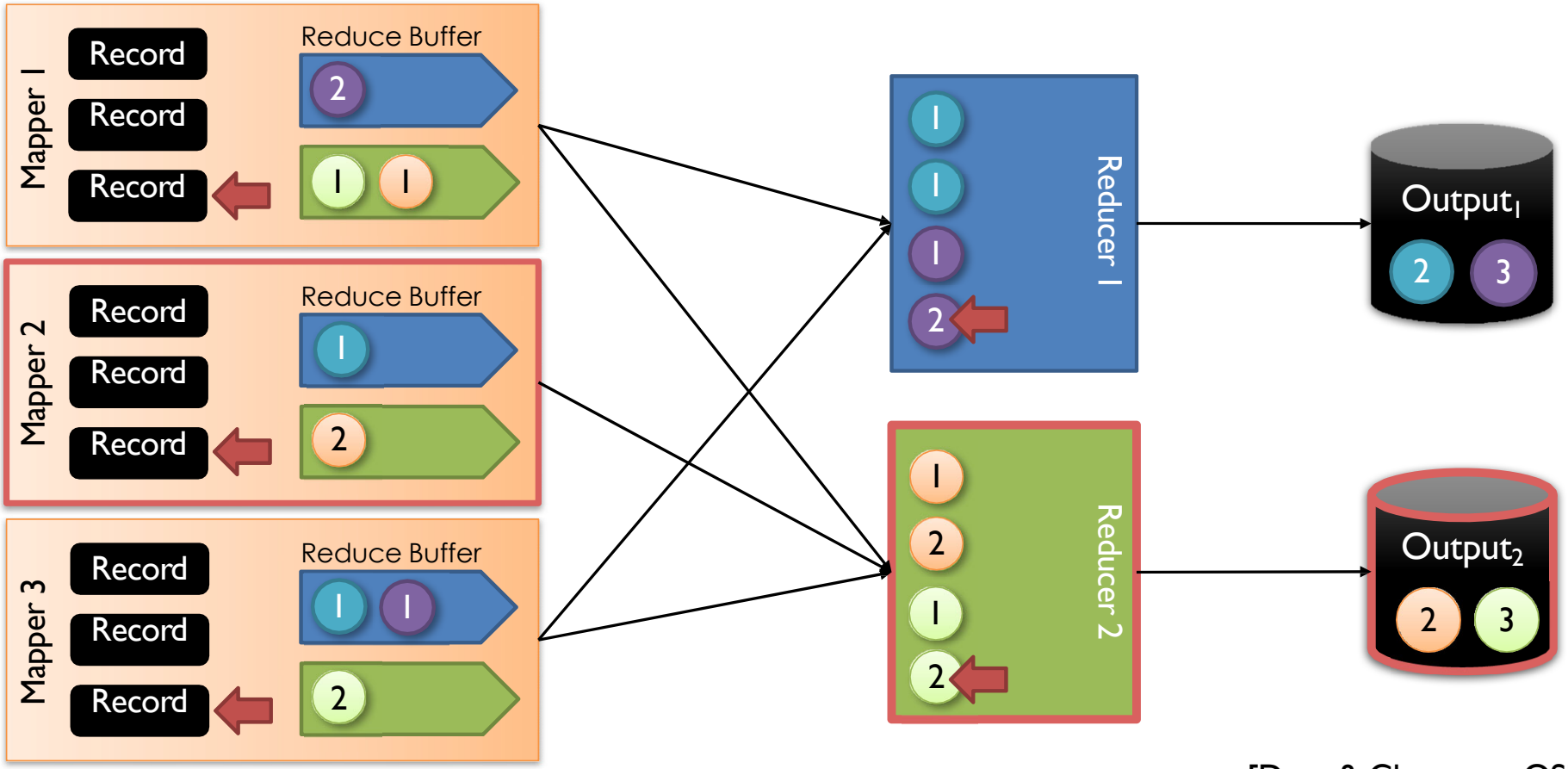
[Dean & Ghemawat, OSDI'04]

The Map Reduce: **Fault Tolerance**



[Dean & Ghemawat, OSDI'04]

The Map Reduce: **Fault Tolerance**

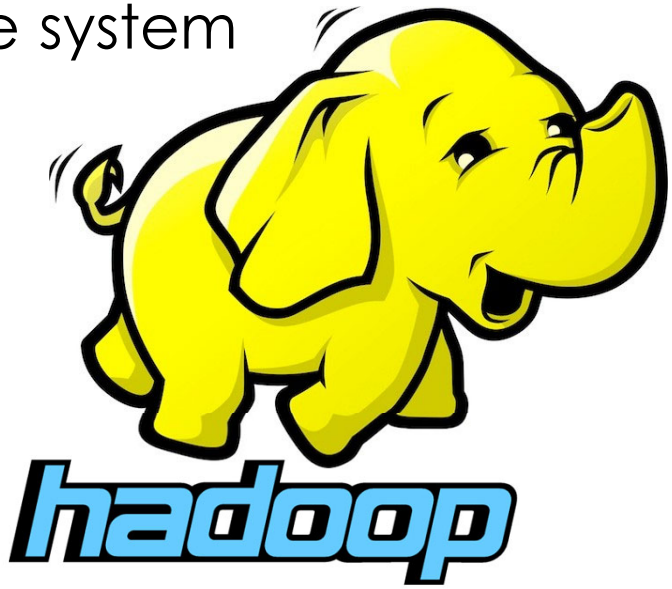


[Dean & Ghemawat, OSDI'04]

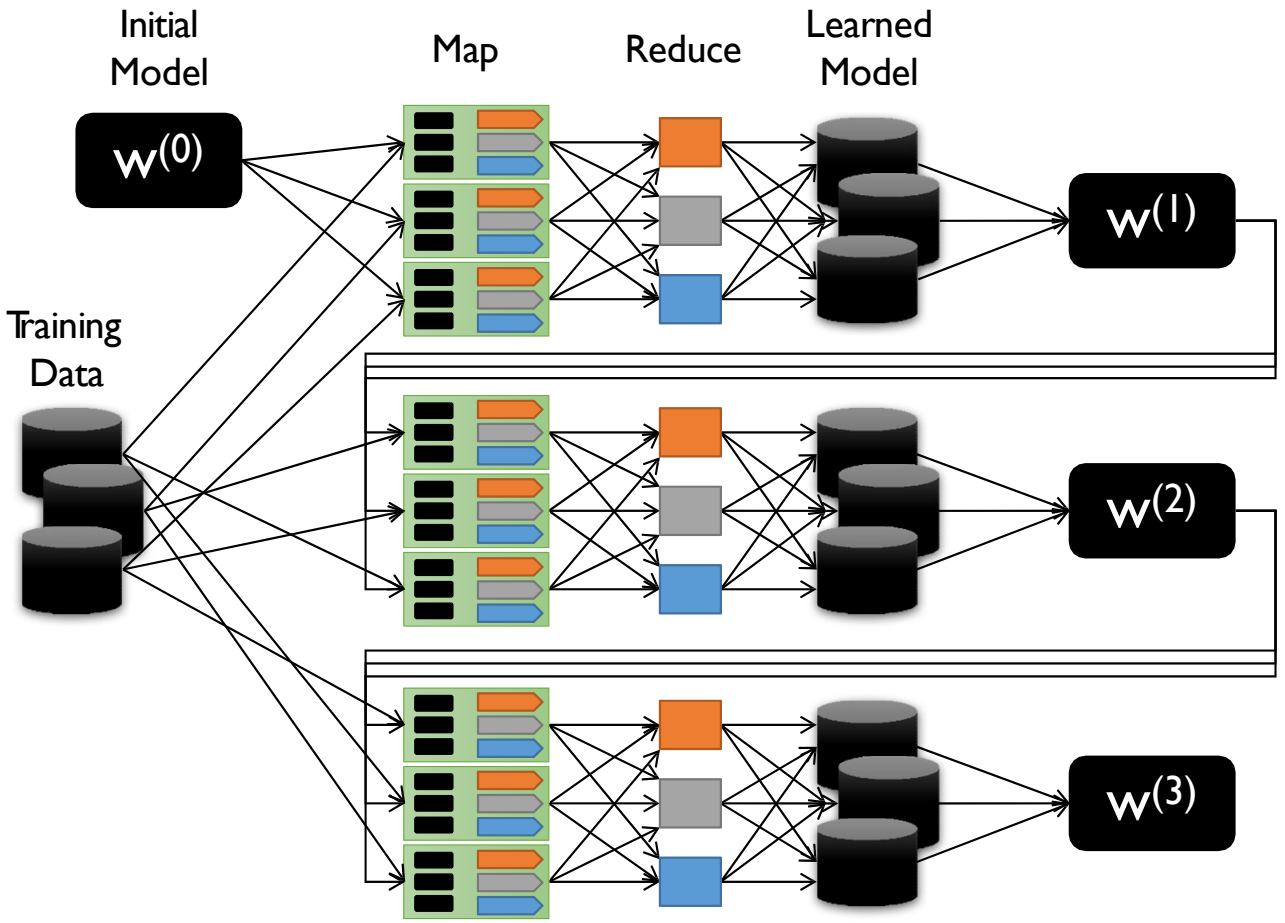
Map Reduce Technologies

Hadoop

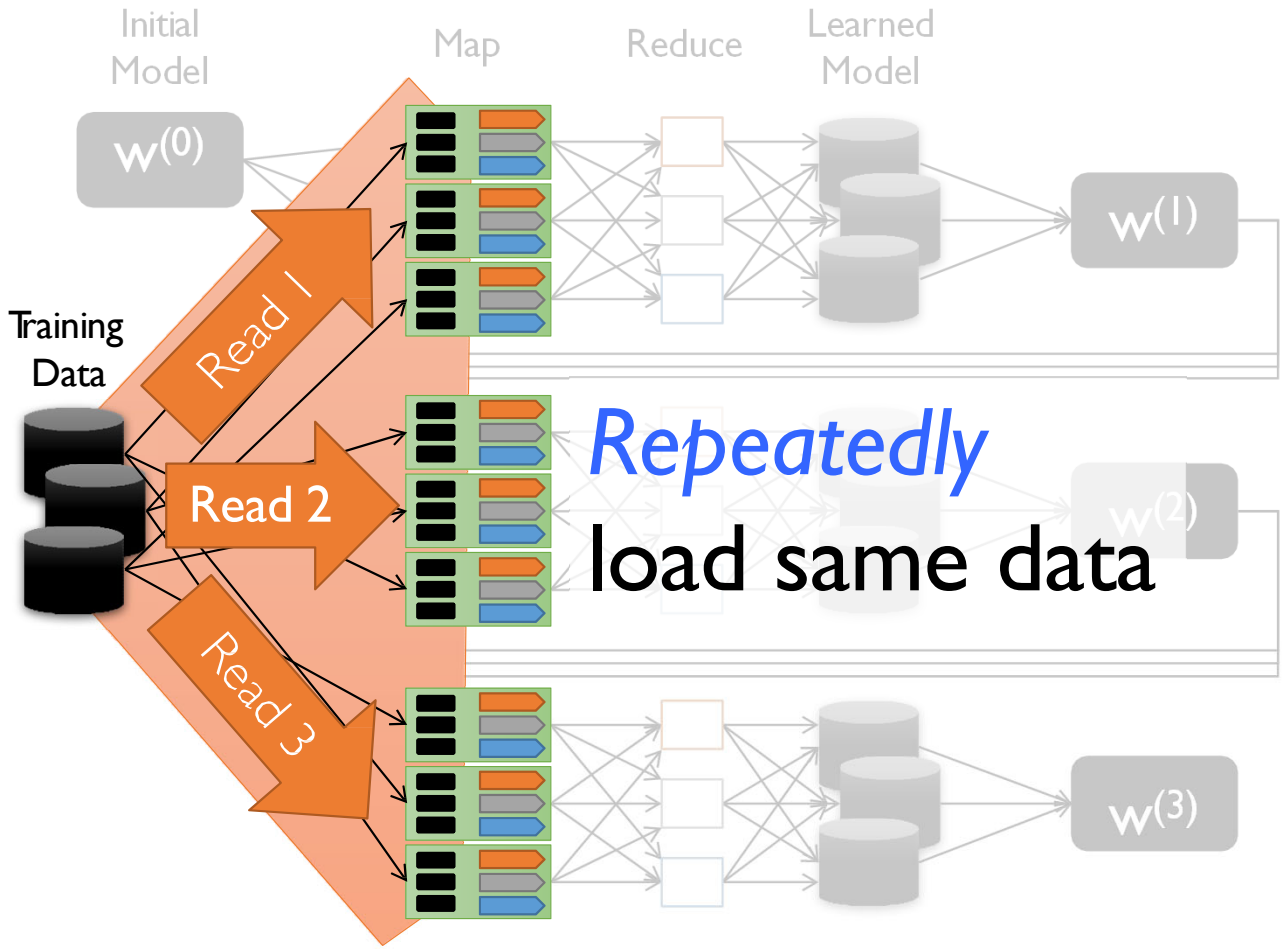
- First open-source map-reducesoftware system
  - Managed by Apache foundation
- Based on Google's
  - Map-Reduce
  - Google File System
- Several key technologies
  - **HDFS:** Hadoop File System
  - **Yarn:** Yet another resource negotiator
  - **MapReduce:** map-reduce compute framework



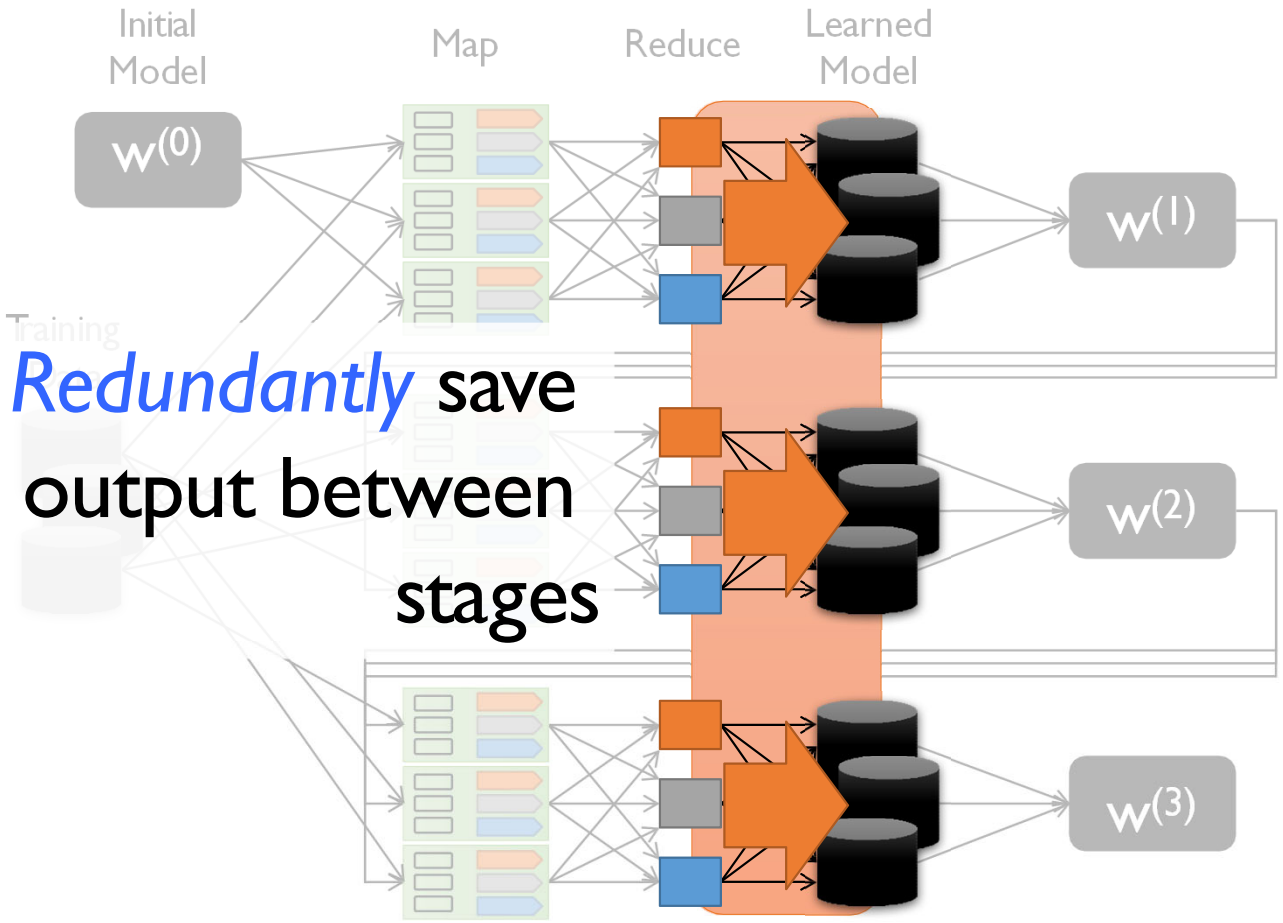
Iteration in Map-Reduce



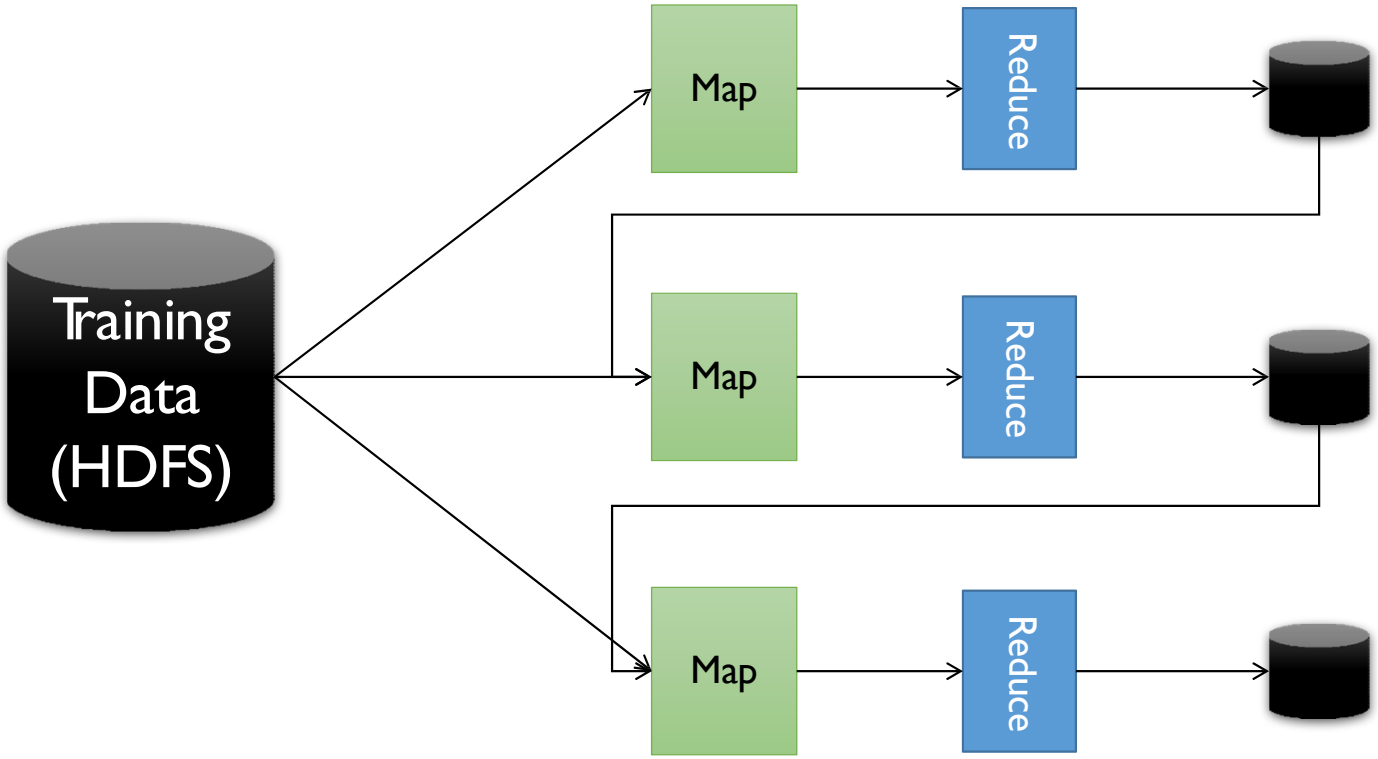
Cost of Iteration in Map-Reduce



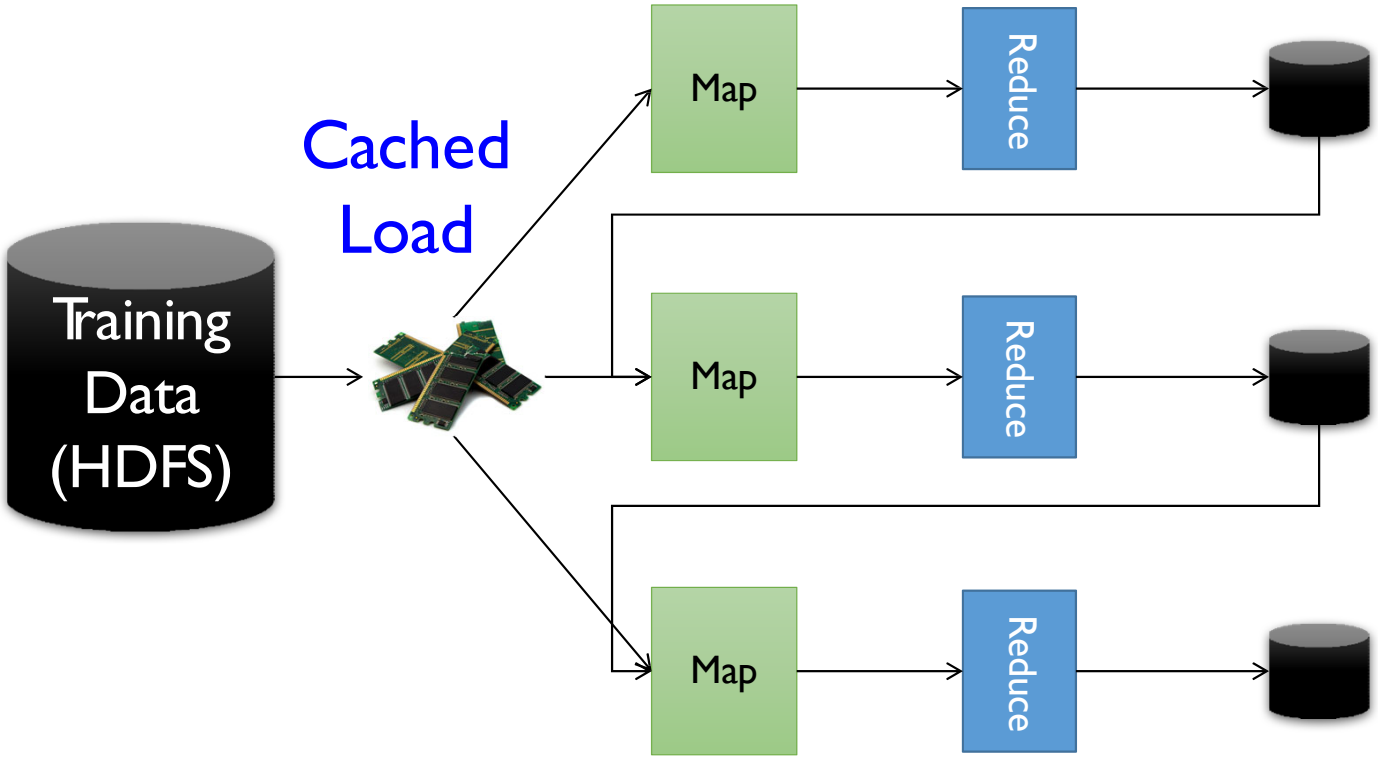
Cost of Iteration in Map-Reduce



Dataflow View

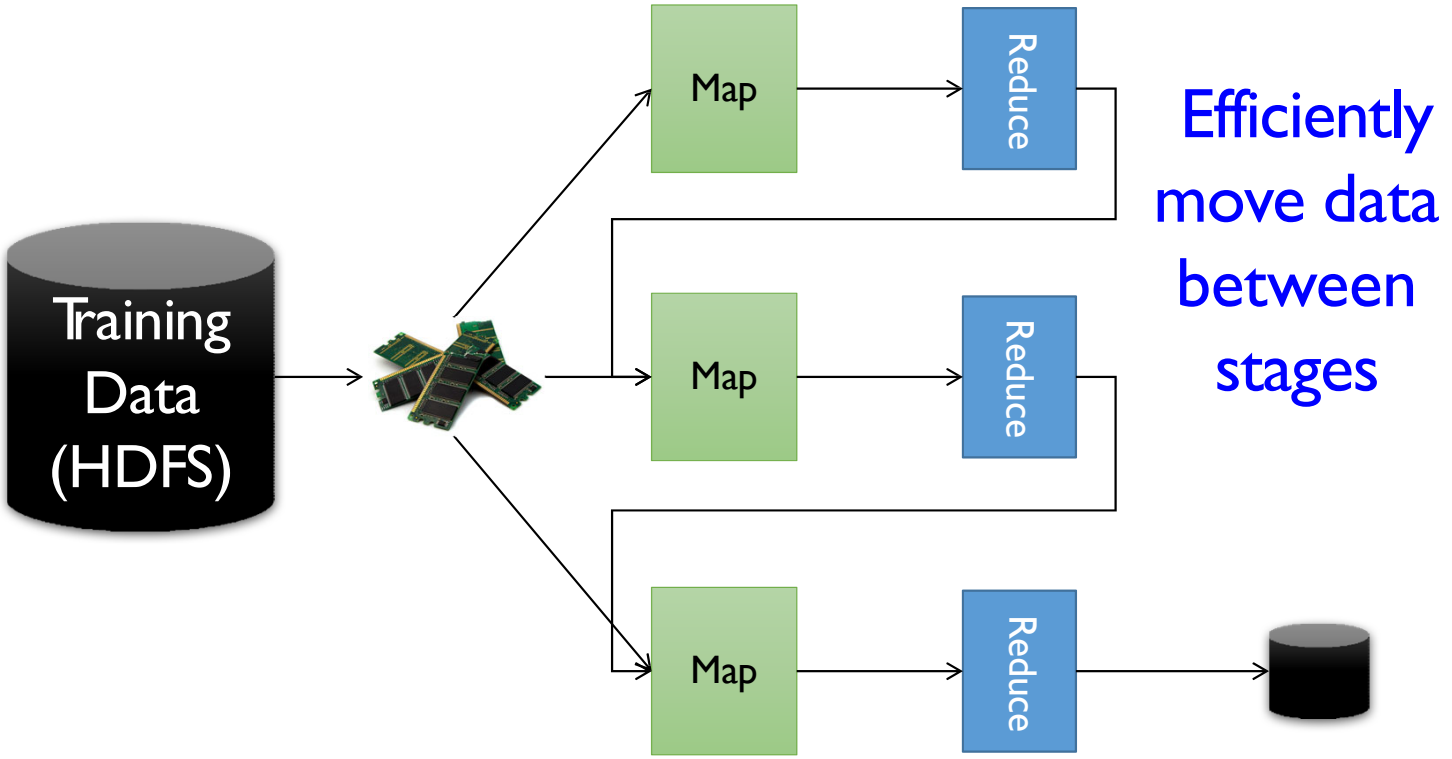


Memory Opt. Dataflow



10-100× faster than network and disk

Memory Opt. Dataflow View



In-Memory Dataflow System

Developed at the UC Berkeley AMP Lab

M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. *Spark: cluster computing with working sets*. HotCloud'10

M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica. *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*, NSDI 2012

What Is 

- Parallel execution engine for big data processing
- **General**: efficient support for multiple workloads
- **Easy** to use: 2-5x less code than Hadoop MR
  - High level API's in Python, Java, and Scala
- **Fast**: up to 100x faster than Hadoop MR
  - Can exploit in-memory when available
  - Low overhead scheduling, optimized engine

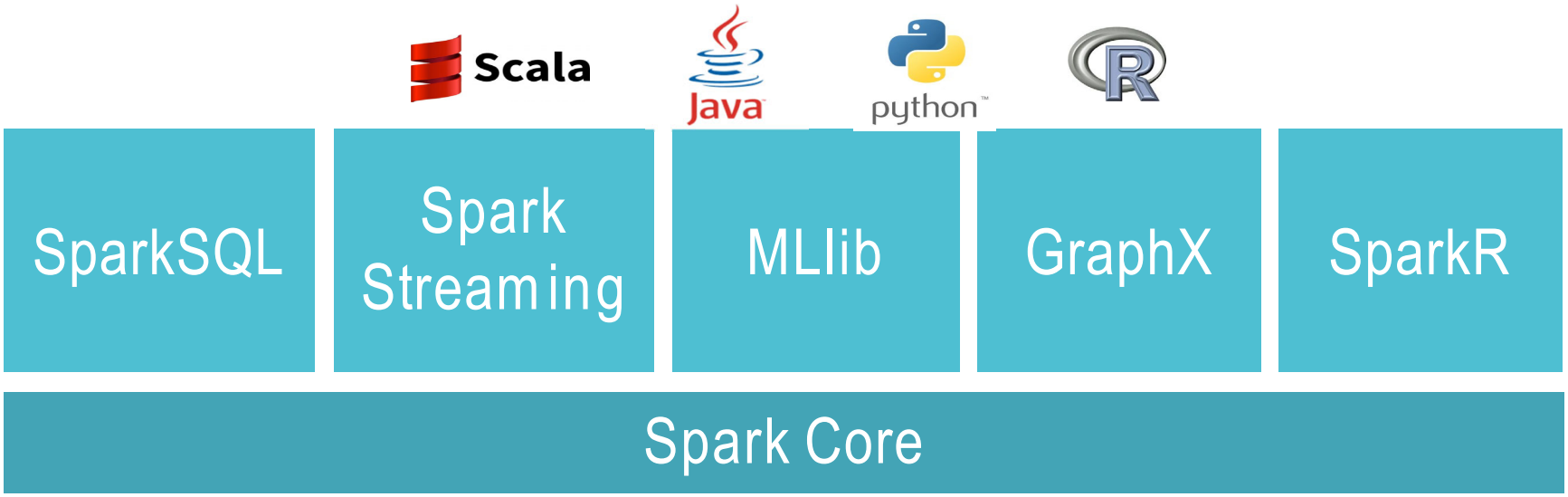
Spark Programming Abstraction

- *Write programs in terms of transformations on distributed datasets*
- Resilient Distributed Datasets (RDDs)
  - Distributed collections of objects that can stored in memory or on disk
  - Built via parallel transformations (map, filter, …)
  - Automatically rebuilt on failure

Slide provided by M. Zaharia

General

- Unifies ***batch, interactive, streaming*** workloads
- Easy to build sophisticated applications
  - Support iterative, graph-parallel algorithms
  - Powerful APIs in Scala, Python, Java, R



Easy to Write Code

```
public class WordCount {
  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length < 2) {
      System.err.println("Usage: wordcount <in> [<in>...] <out>");
      System.exit(2);
    }
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    for (int i = 0; i < otherArgs.length - 1; ++i) {
      FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
    }
    FileOutputFormat.setOutputPath(job,
      new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

WordCount in 50+ lines of Java MR

```
val f = sc.textFile(inputPath)
val w = f.flatMap(l => l.split(" ")).map(word => (word, 1)).cache()
w.reduceByKey(_ + _).saveAsText(outputPath)
```

WordCount in 3 lines of Spark

Fast: Time to sort 100TB

2013 Record: Hadoop

2100 machines

72 minutes

2014 Record: Spark

207 machines

23 minutes

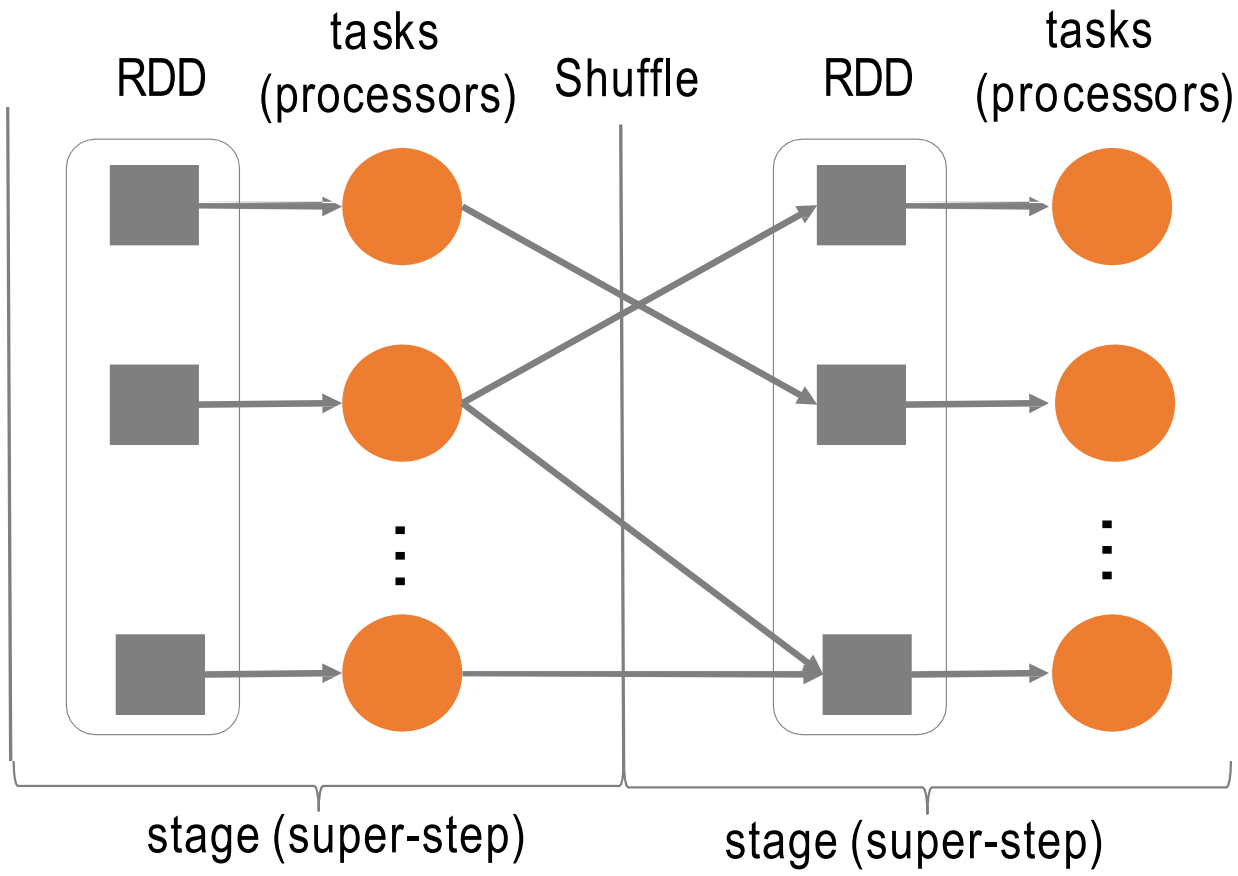
Also sorted 1PB in 4 hours (bottlenecked by network)

Source: Daytona GraySort benchmark, sortbenchmark.org

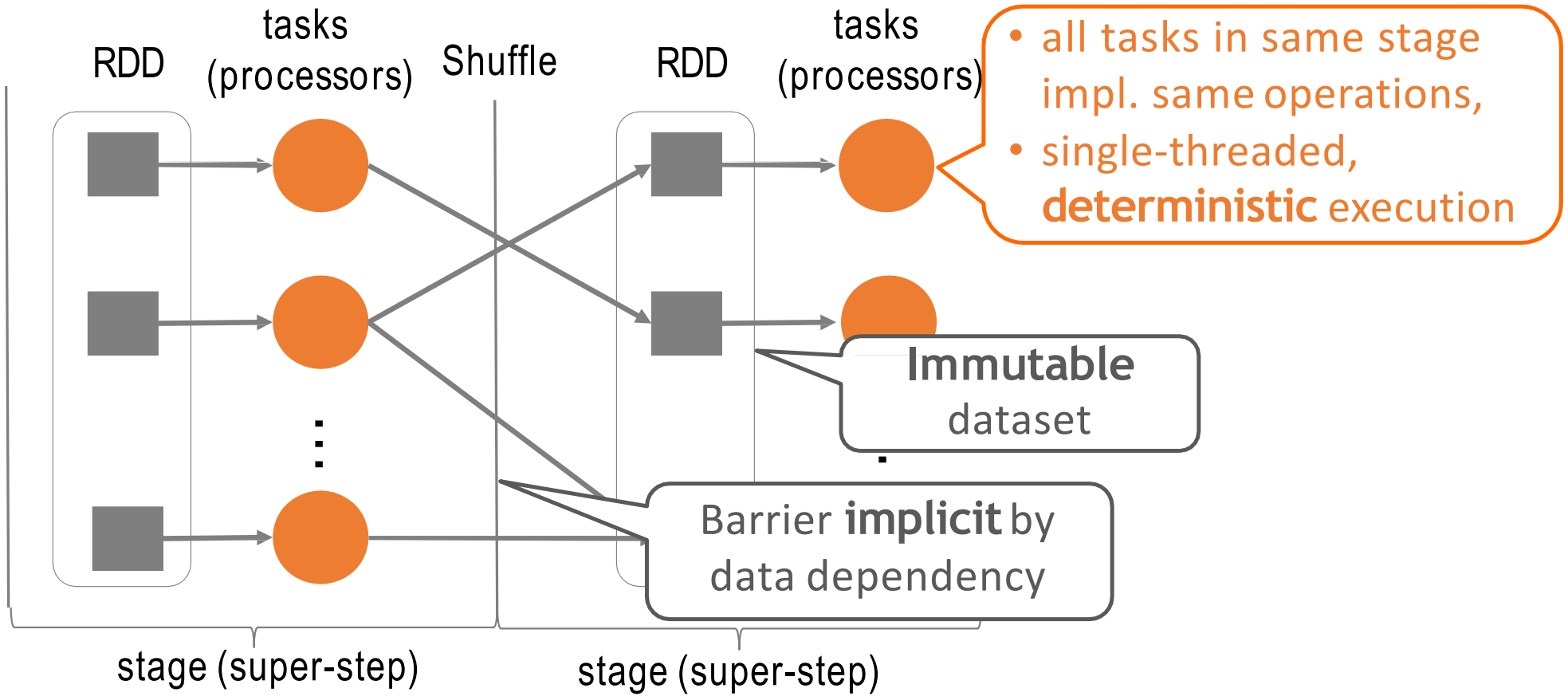
RDD: Resilient Distributed Datasets

- Collections of objects partitioned & distributed across a cluster
  - Stored in RAM or on Disk
  - Resilient to failures
- Operations
  - Transformations
  - Actions

Spark


RDD
tasks
(processors)
Shuffle
RDD
tasks
(processors)
stage (super-step)
stage (super-step)

Spark


RDD
tasks (processors)
Shuffle
RDD
tasks (processors)
• all tasks in same stage impl. same operations,
• single-threaded, **deterministic** execution
Immutable dataset
Barrier **implicit** by data dependency
stage (super-step)
stage (super-step)

Operations on RDDs

- Transformations f(RDD) => RDD
  - Lazy (not computed immediately)
  - E.g., “map”, “filter”, “groupBy”

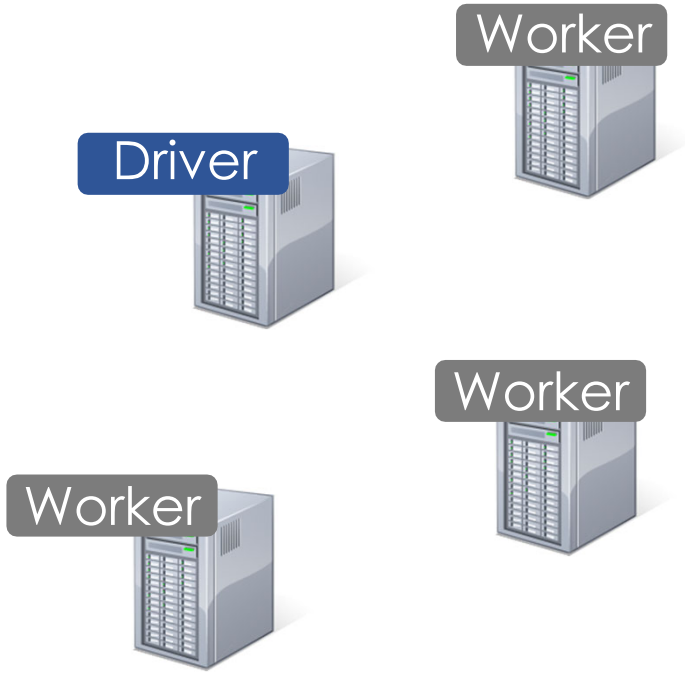
- Actions:
  - Triggers computation
  - E.g. “count”, “collect”, “saveAsTextFile”

Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

Example: Log Mining

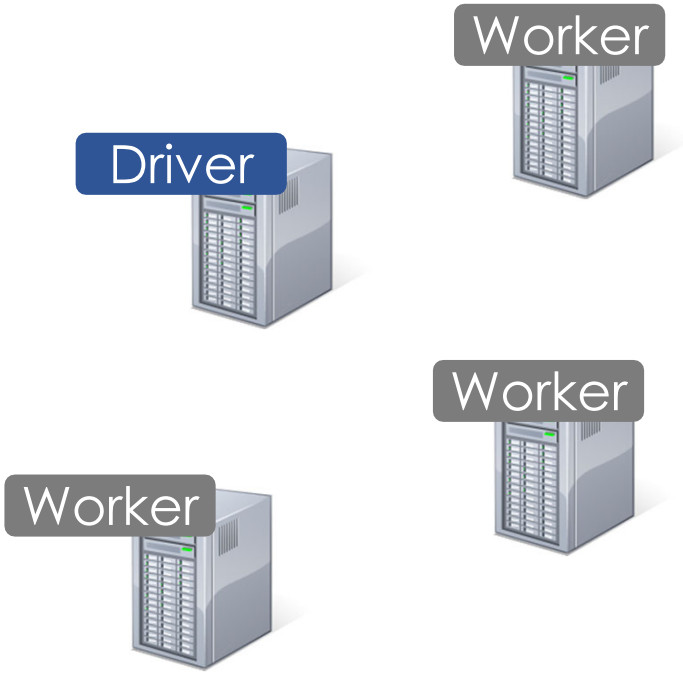
Load error messages from a log into memory, then interactively search for various patterns



Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
```

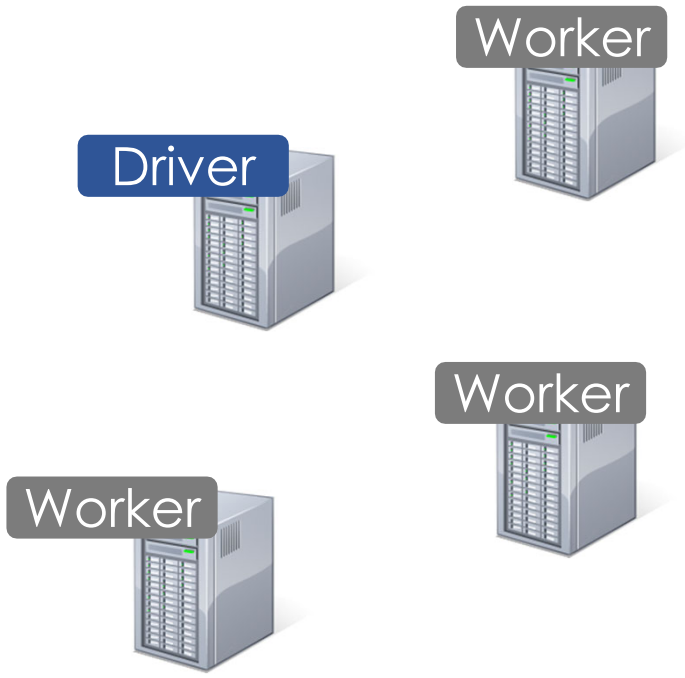


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

Base RDD

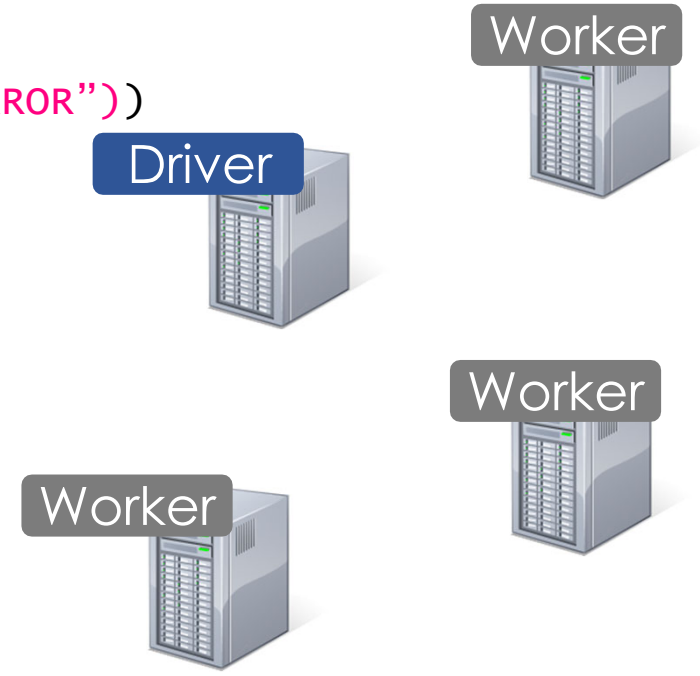
```
lines = spark.textFile("hdfs://...")
```



Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
```

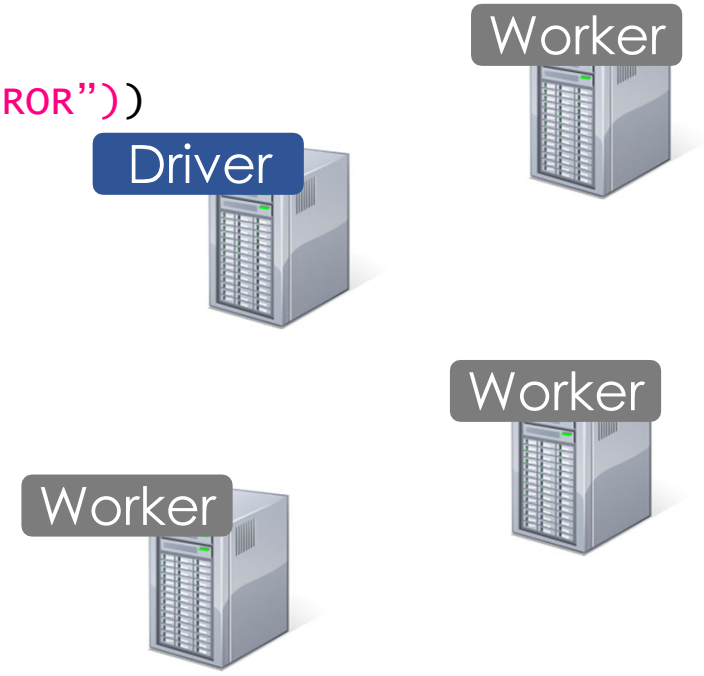


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

Transformed RDD

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
```

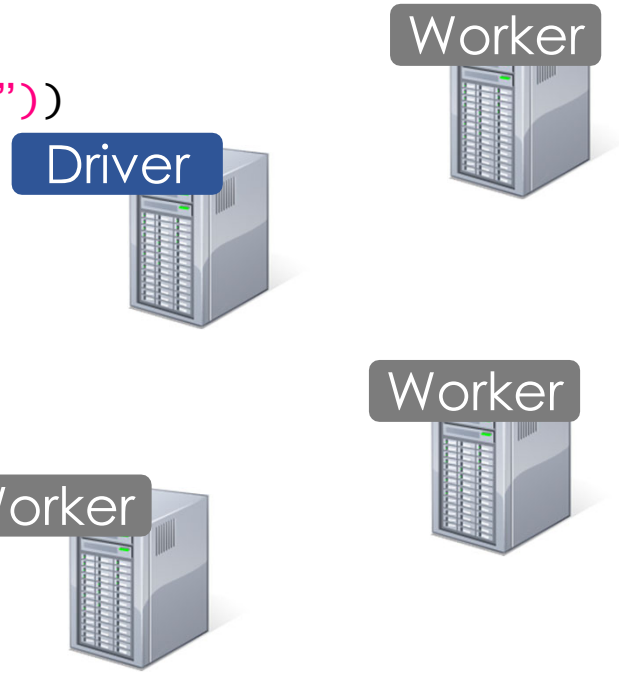


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()

messages.filter(lambda s: “mysql” in s).count()
```

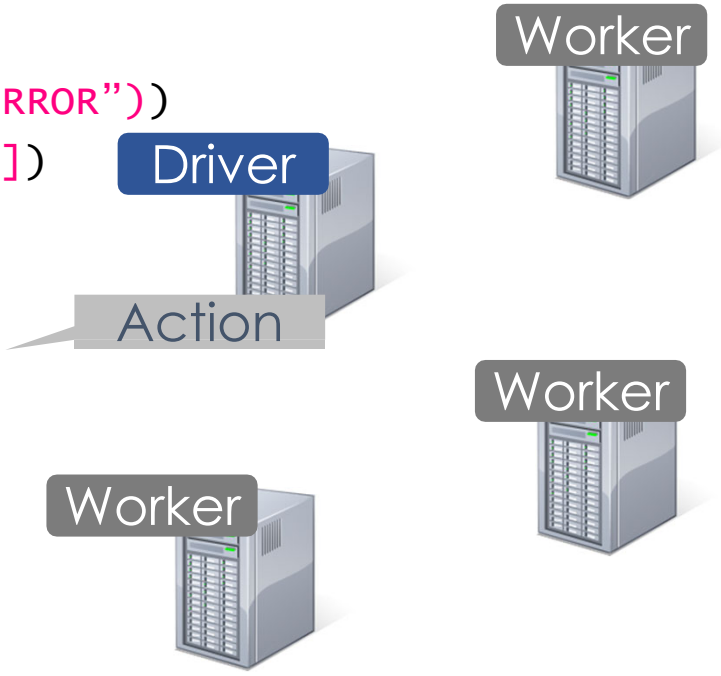


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```

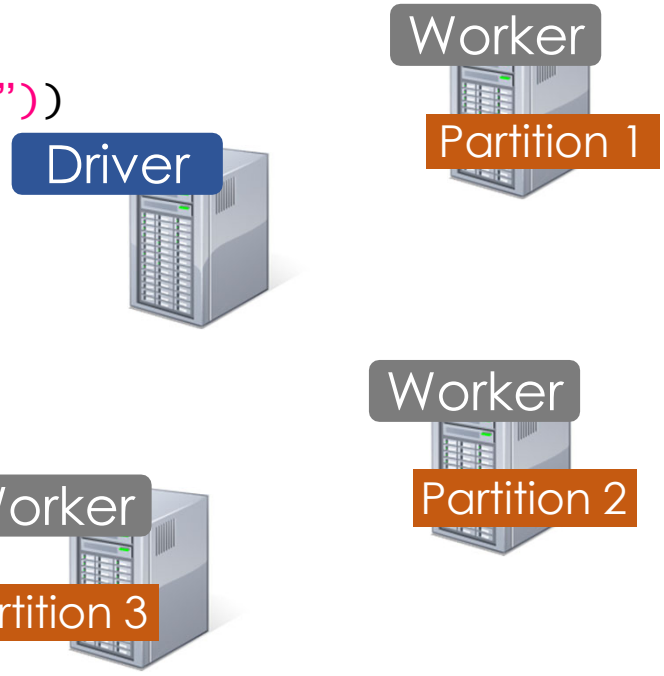


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()

messages.filter(lambda s: “mysql” in s).count()
```

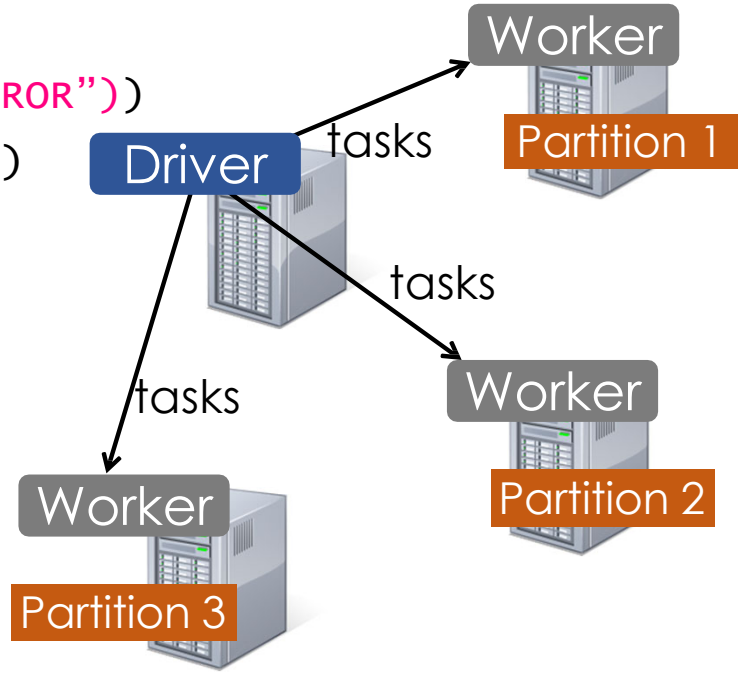


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```

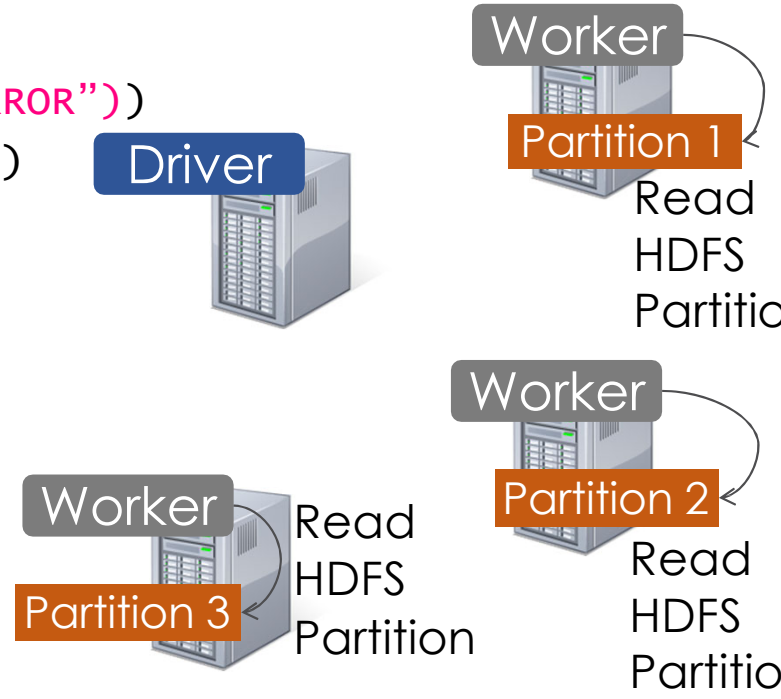


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()

messages.filter(lambda s: “mysql” in s).count()
```

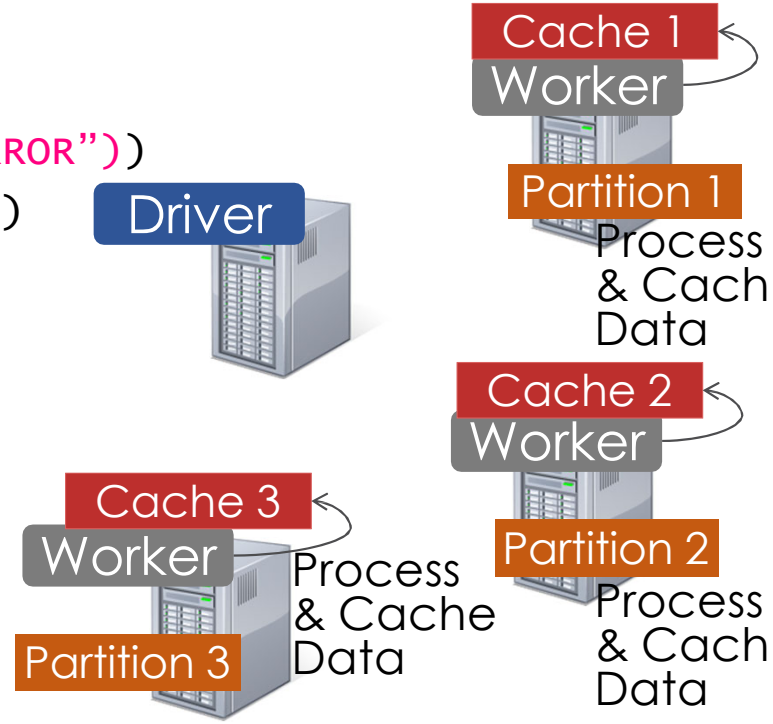


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```

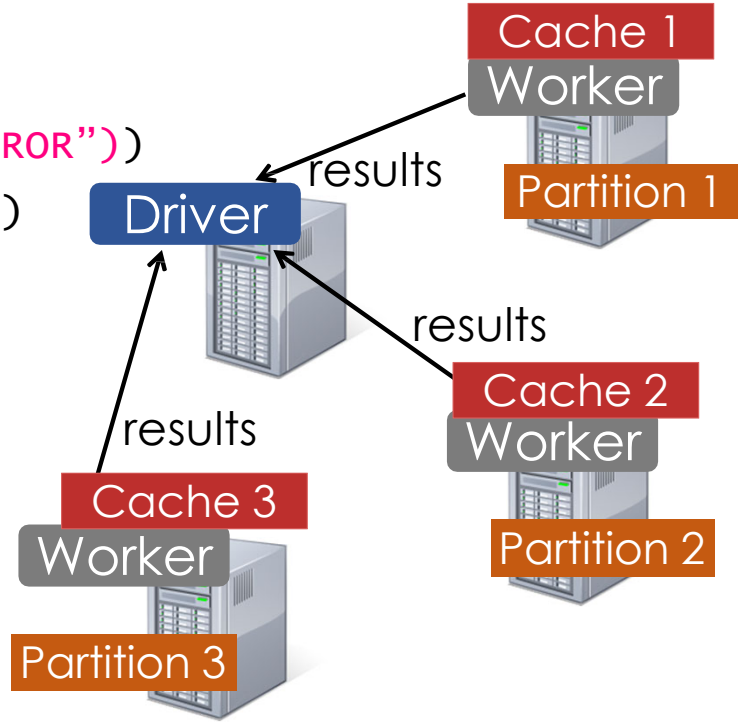


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```

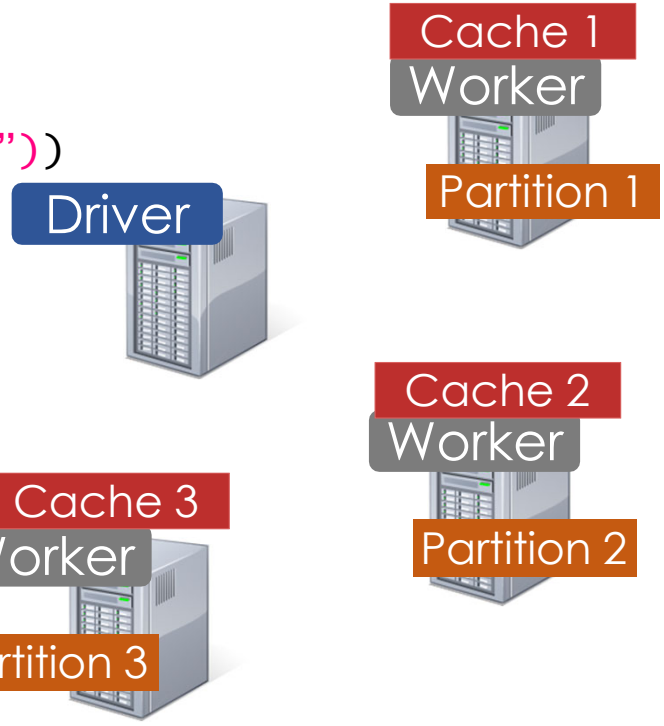


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

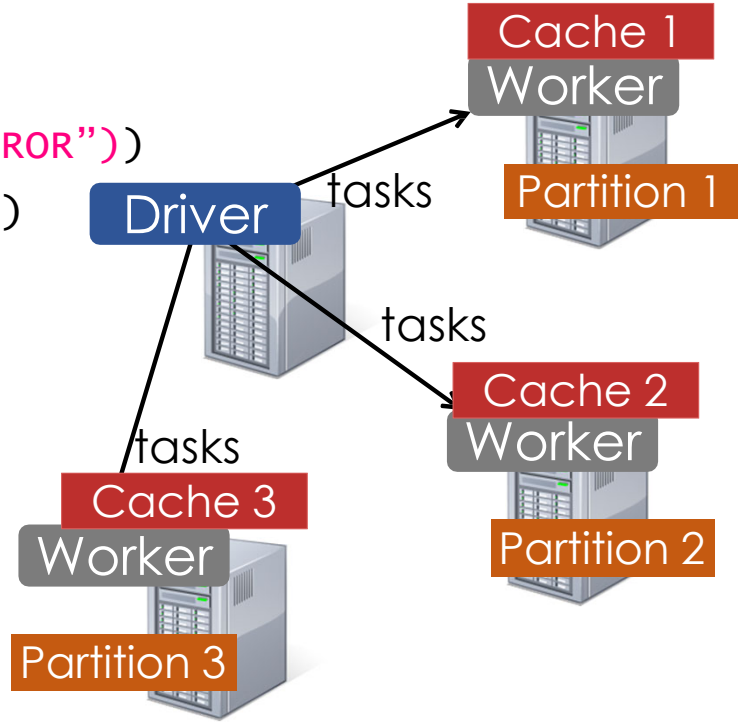


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()

messages.filter(lambda s: “mysql” in s).count()
messages.filter(lambda s: “php” in s).count()
```

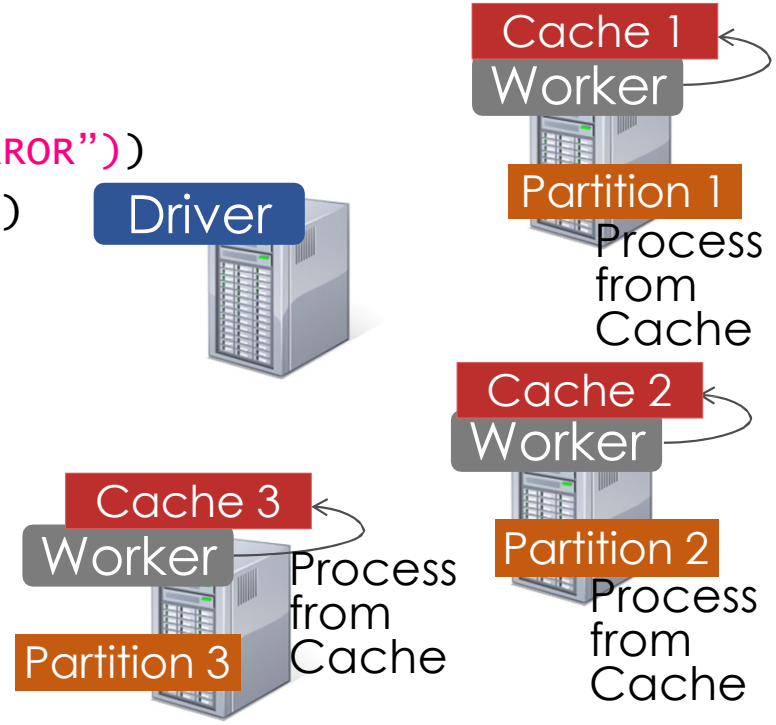


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

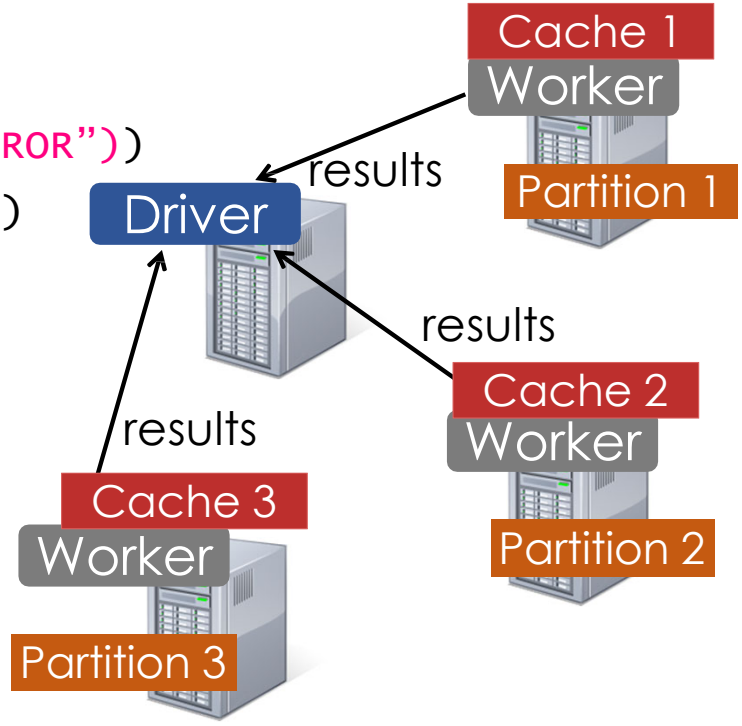


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

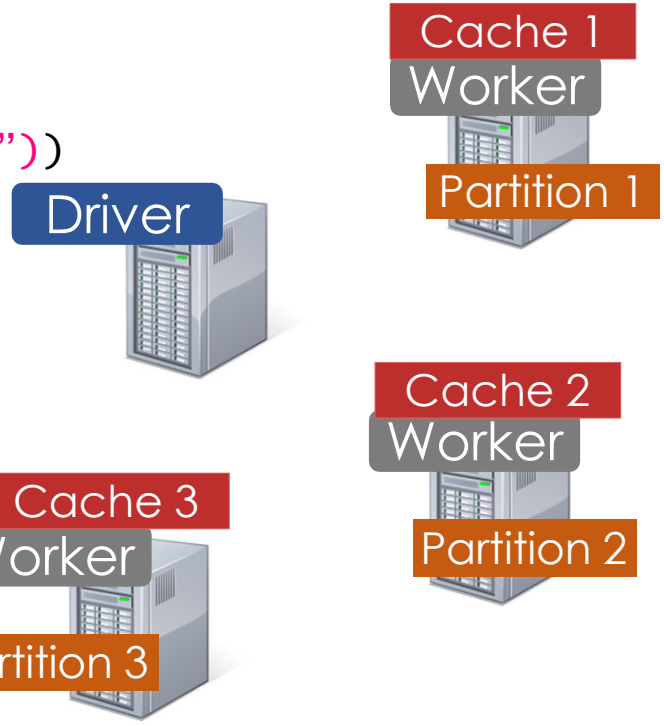


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```



Cache your data ➔ Faster Results

Full-text search of Wikipedia

- 60GB on 20 EC2 machines
- 0.5 sec from mem vs. 20s for on-disk