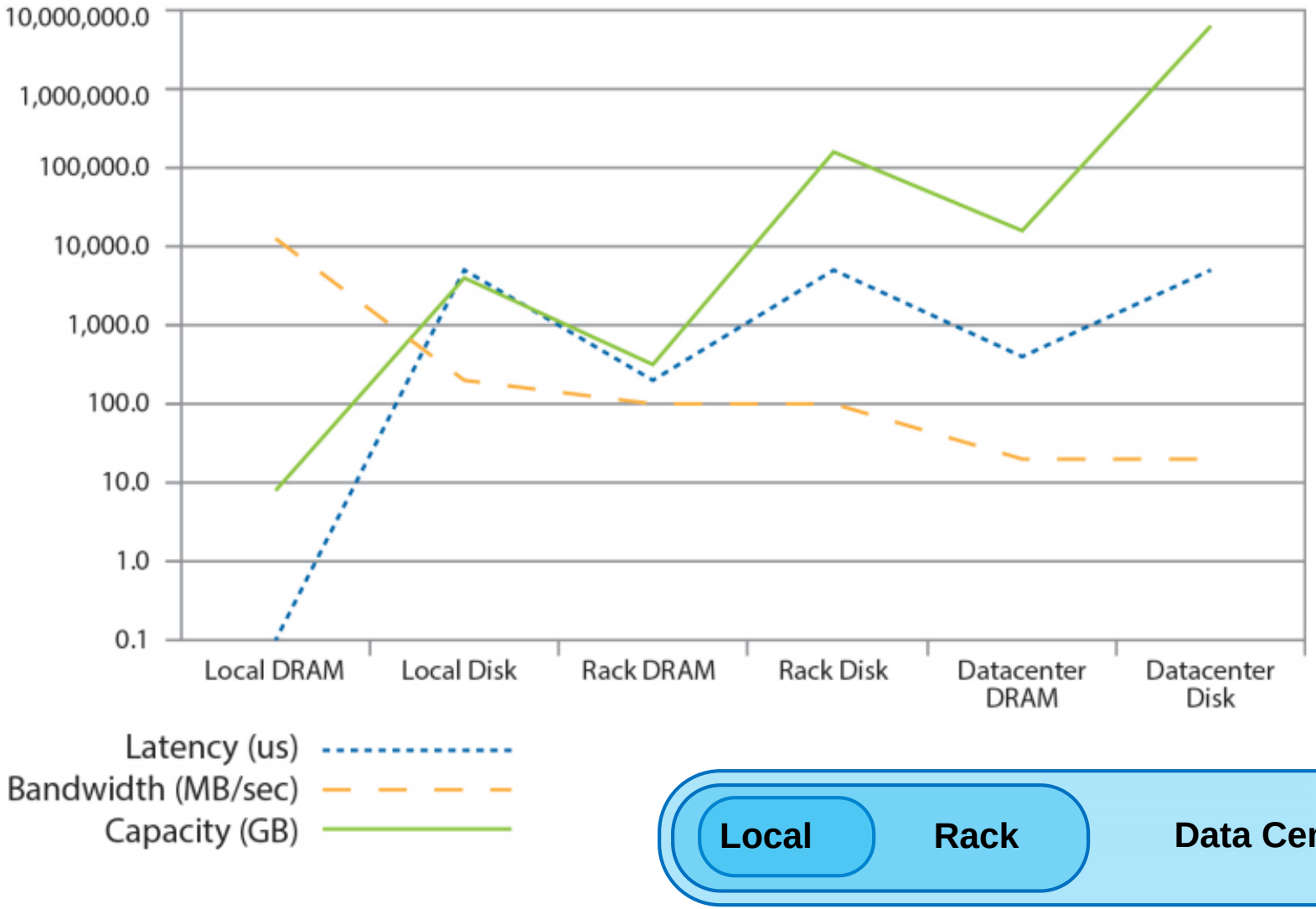
Consultation Hours

- We will have consultation hours in the following time slots:
  - Slots: 3:30-4:30pm Monday Jan 30, Feb 6, Feb 13.
  - Venue: my office (COM3-02-12)
- Alternatively, email me if you have any question.

Recap: The Cost of Data Accesses in Data Center

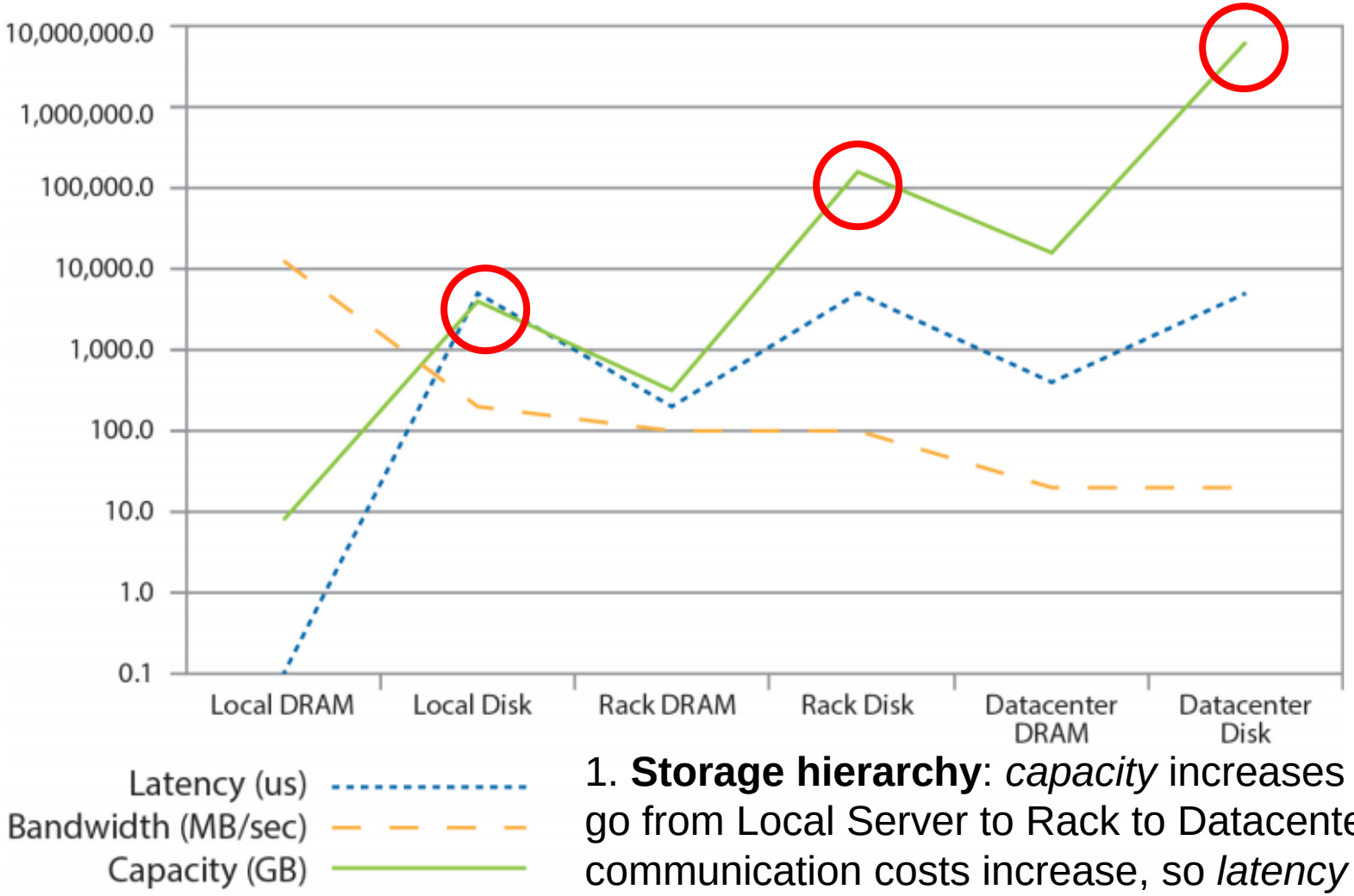
- A data access may cross several components including hard disk, DRAM, rack switch and cluster switch etc.
- We will do some back-of-envelop calculations on the cost
  - Like algorithmic complexity analysis, we approximate for the scale or the order of magnitude, rather than the exact number.
  - Many details are simplified: a) an one-way communication (data -> program), b) no failures etc.
- Latency
  - = The sum of the latency caused in all components in the data access.
- Bandwidth
  - = The minimum of the bandwidths of all components in the data access.
  - Assume the hardware peak bandwidth
  - The actual achieved bandwidth depends on the algorithm.

Recap: The Cost of Moving Data Around Data Center



Source: Barroso and Urs Hölzle (2013)

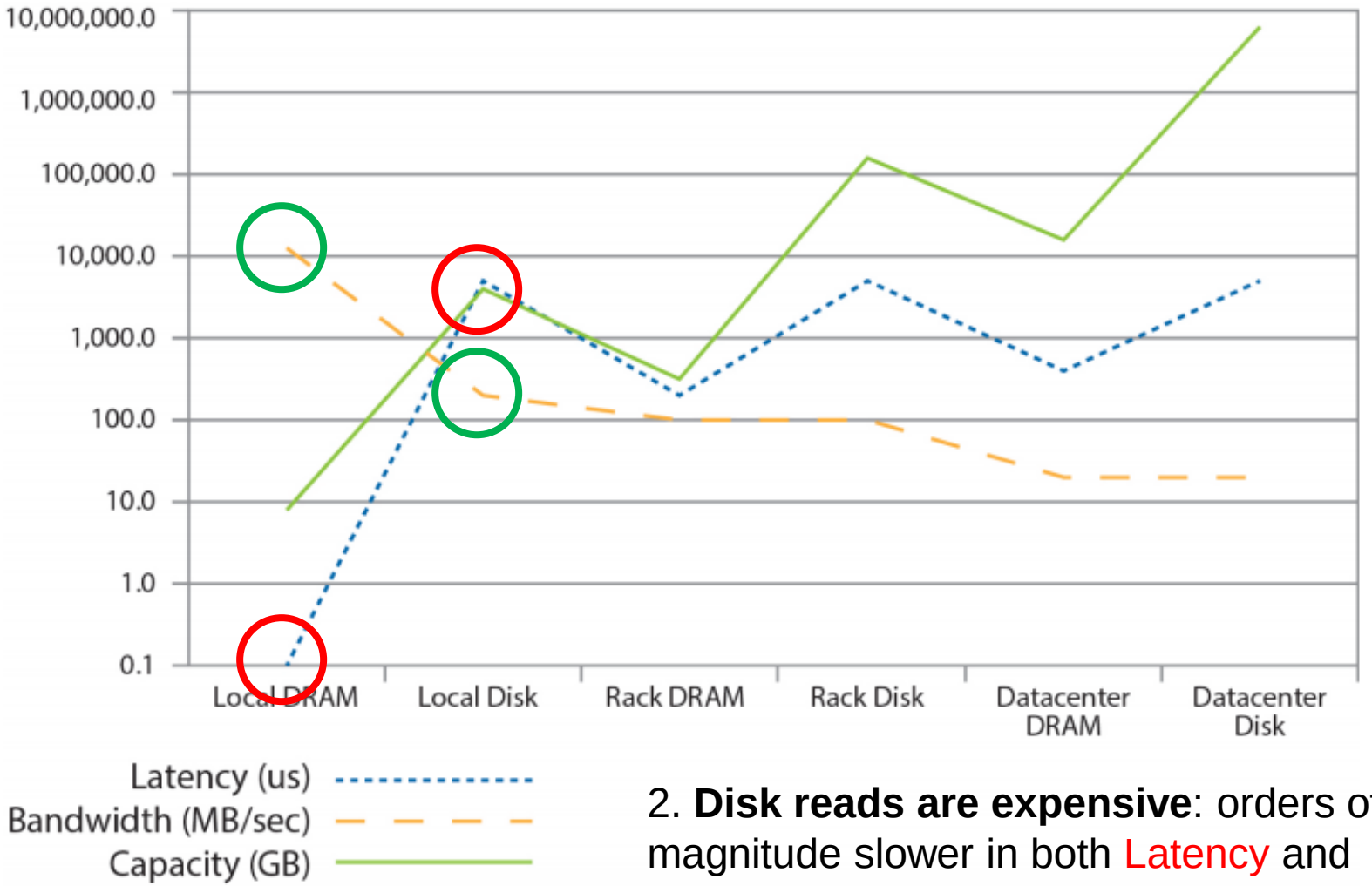
Recap: The Cost of Moving Data Around Data Center



1. **Storage hierarchy**: *capacity* increases as we go from Local Server to Rack to Datacenter. But, communication costs increase, so *latency* increases and *bandwidth* decreases. Takeaway: sending data further on a network is costly!

Source: Barroso and Urs Hölzle (2013)

Recap: The Cost of Moving Data Around Data Center



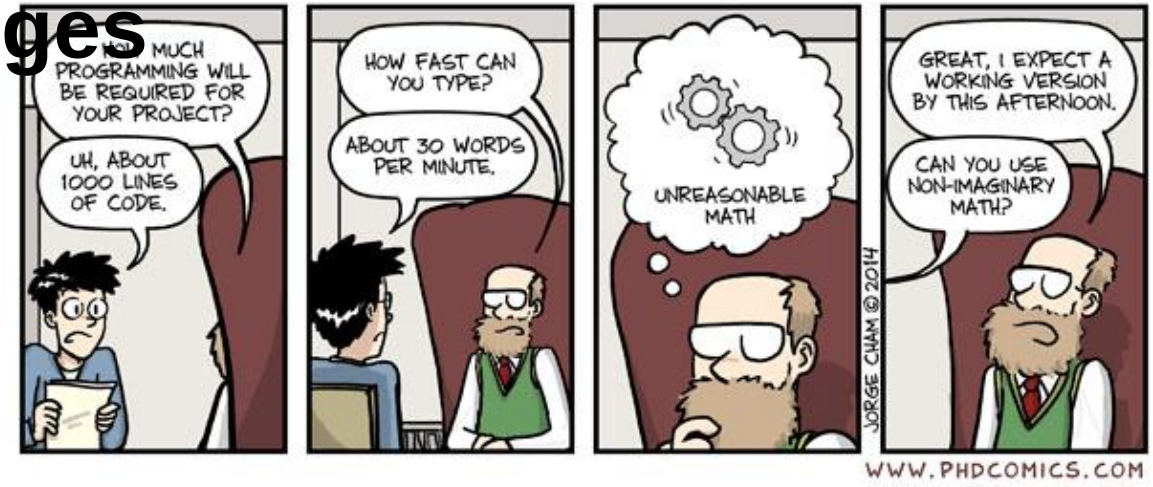
2. **Disk reads are expensive**: orders of magnitude slower in both Latency and Bandwidth. Latency of disk reads is even more than from Rack / Datacenter DRAM.

Source: Barroso and Urs Hölzle (2013)

Recap: “Big Ideas” of Massive Data Processing in Data Centers

- Scale “out”, not “up”
  - scale ‘out’ = combining many cheaper machines; scale ‘up’= increasing the power of each individual machine
  - Also called ’horizontal’ vs ‘vertical’ scaling
- Seamless scalability
  - E.g. if processing a certain dataset takes 100 machine hours, ideal scalability is to use a cluster of 10 machines to do it in about 10 hours.
- Move processing to the data
  - Clusters have limited bandwidth: we should move the task to the machine where the data is stored
- Process data sequentially, avoid random access
  - Seeks are expensive, disk throughput is reasonable

Recap: Challenges



- How do we assign work units to workers?
- What if we have more work units than workers?
- What if workers need to share partial results?
- How do we aggregate partial results?
- How do we know all the workers have finished?
- What if workers die/fail?

Recap: The datacenter *is* the computer

- It's all about the right level of abstraction
  - Moving beyond the single machine architecture
  - What's the "instruction set" (or "API") of the datacenter computer?
- Hide system-level details from the developers
  - No more race conditions, lock contention, etc.
  - No need to explicitly worry about reliability, fault tolerance, etc.
- Separating the *what* from the *how*
  - Developer specifies the computation that needs to be performed
  - Execution framework ("runtime") handles actual execution

CS4225/CS5425 Big Data Systems for Data Science

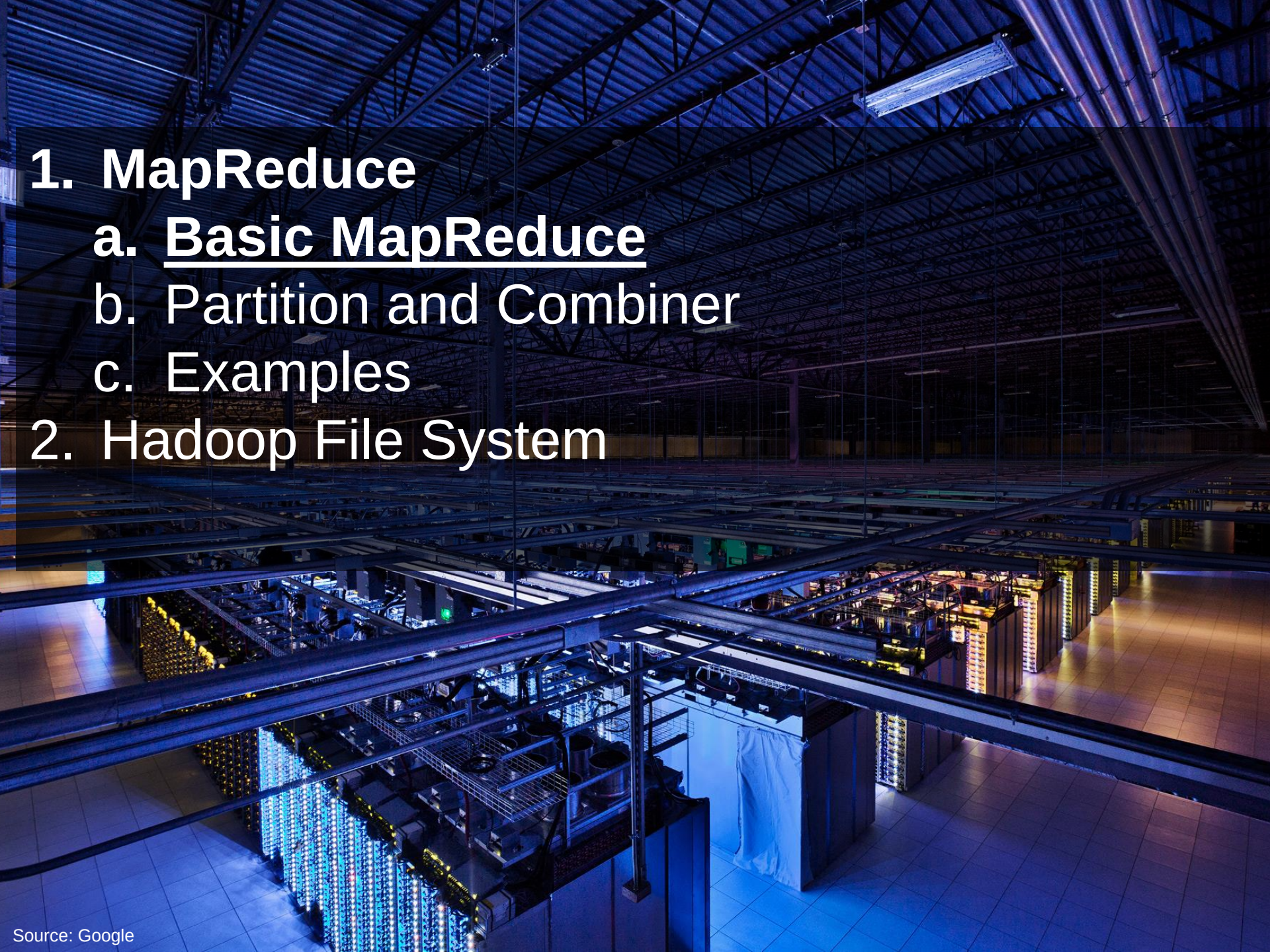
Introduction of MapReduce

Bingsheng He
School of Computing
National University of Singapore
hebs@comp.nus.edu.sg

School of Computing

Learning Outcomes

- Abstractions for MapReduce
- Programming with MapReduce
- Hadoop File System



1. **MapReduce**
   a. **Basic MapReduce**
   b. Partition and Combiner
   c. Examples
2. Hadoop File System

Source: Google

Typical Big Data Problem

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

Map

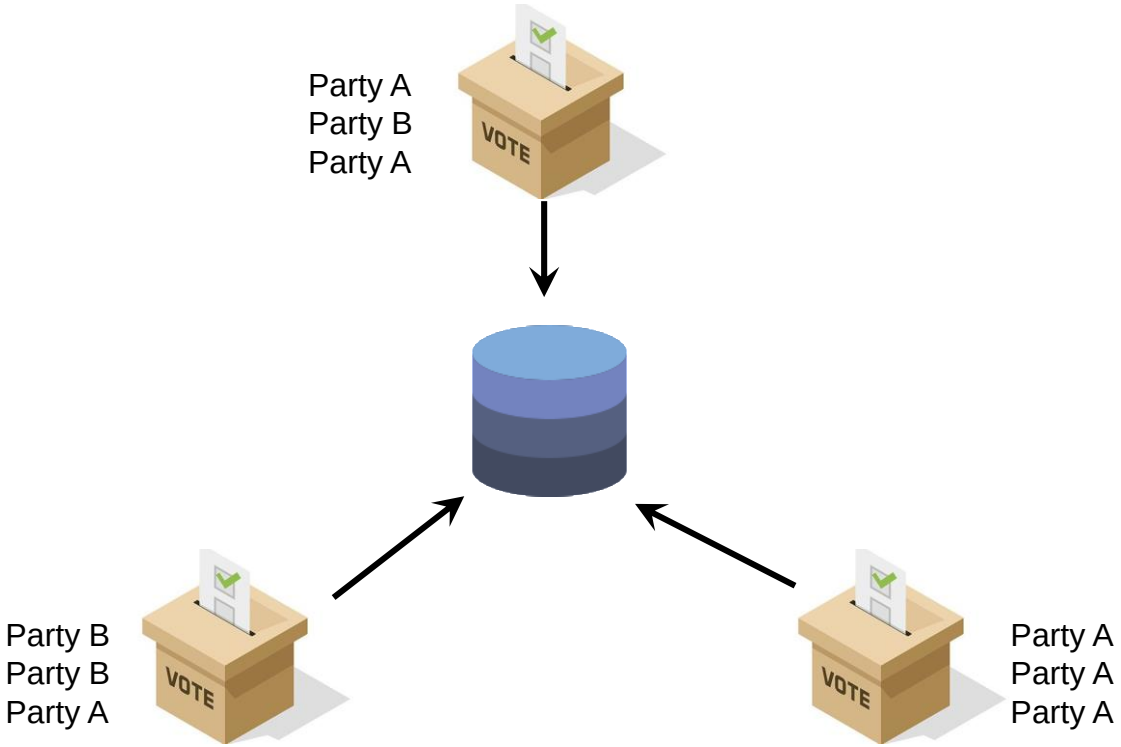
Reduce

Key idea: provide a functional abstraction for these two operations

(Dean and Ghemawat, OSDI 2004)

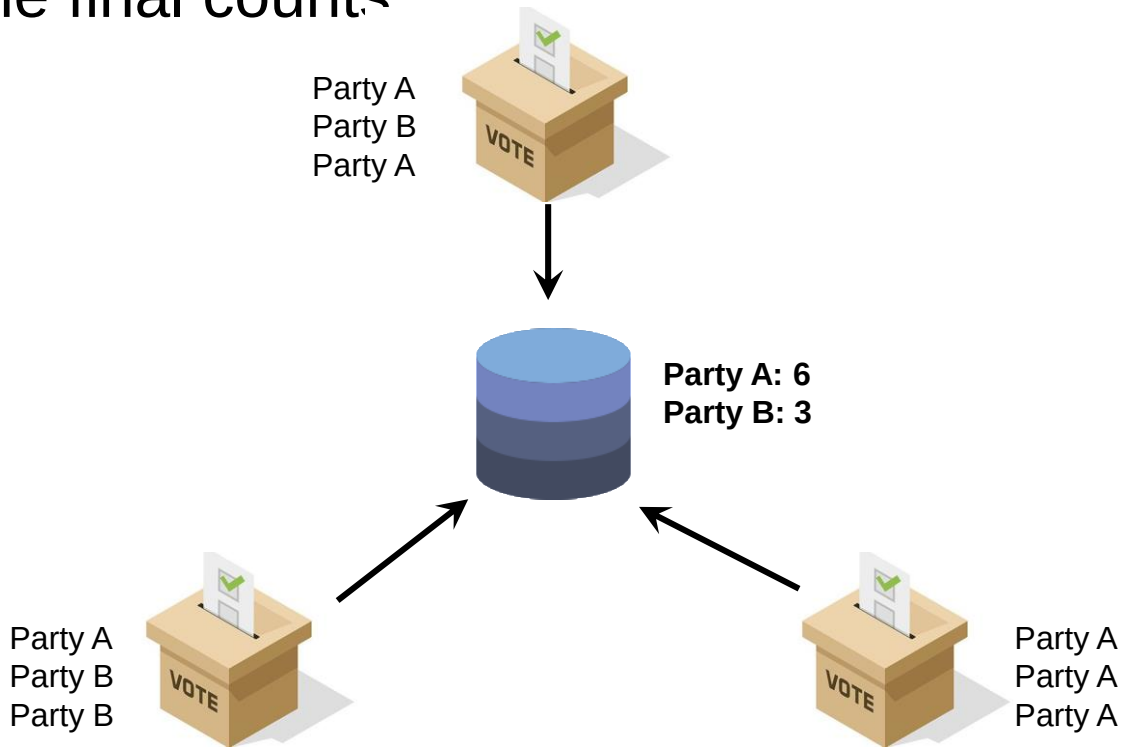
Basic Example: Tabulating Election Results from Multiple Polling Sites

- Imagine you are hired to develop the software for Singapore's election counting system.
- You need to **aggregate** vote counts from multiple stations into the final counts.



Basic Example: Tabulating Election Results from Multiple Polling Sites

- Imagine you are hired to develop the software for Singapore's election counting system.
- You need to **aggregate** vote counts from multiple stations into the final counts



Tabulating Election Results: MapReduce

Map

Shuffle

Reduce

Party A
Party B
Party A

Party A
Party B
Party B

Party A
Party A
Party A

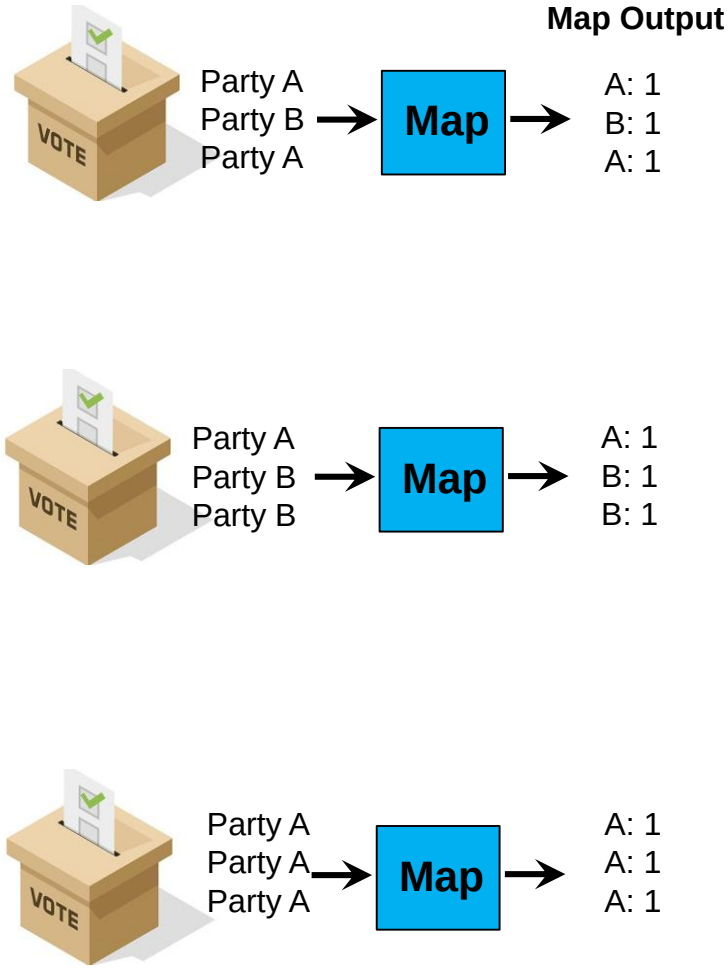
Party A: 6
Party B: 3

Tabulating Election Results: MapReduce

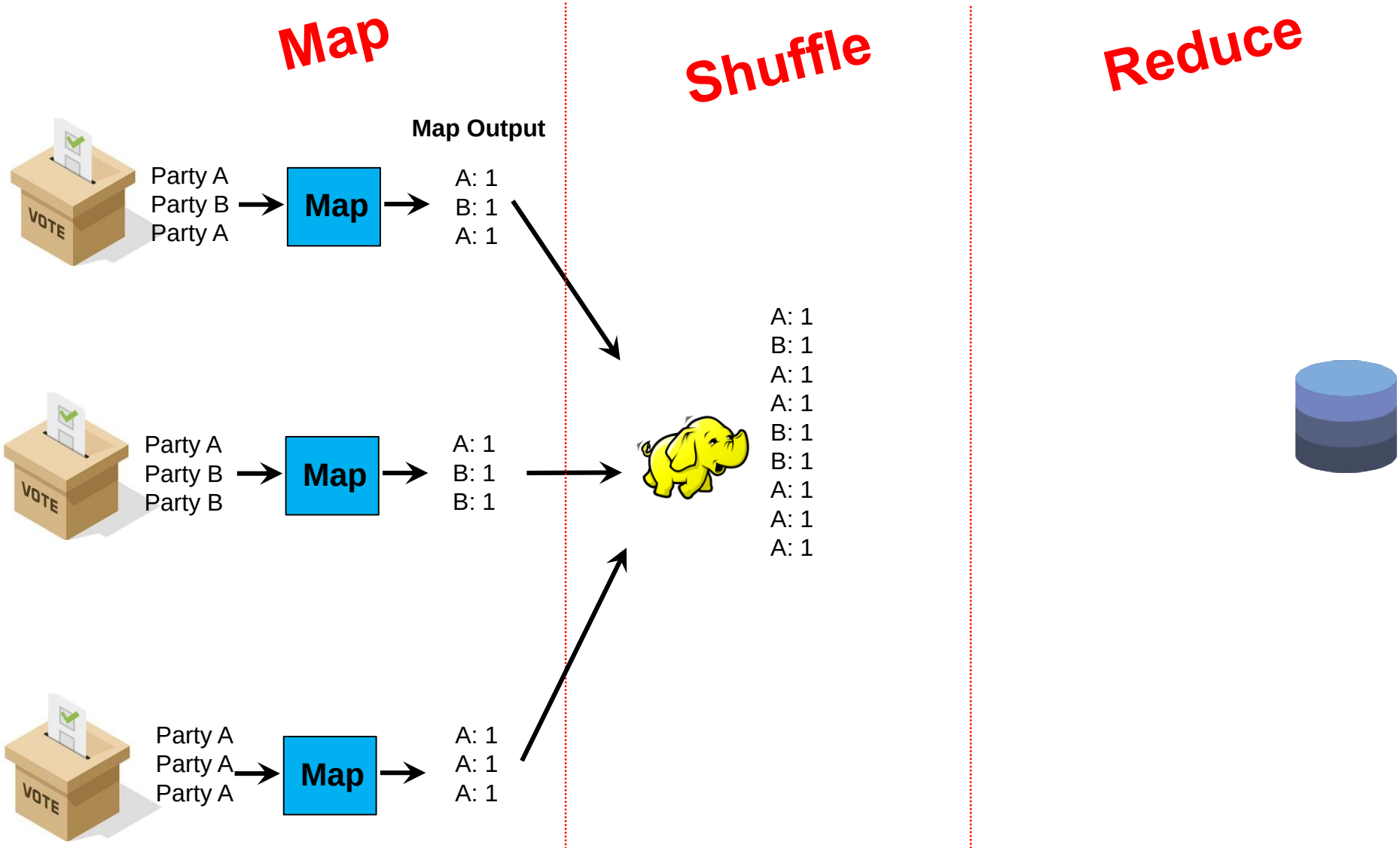
Map

Shuffle

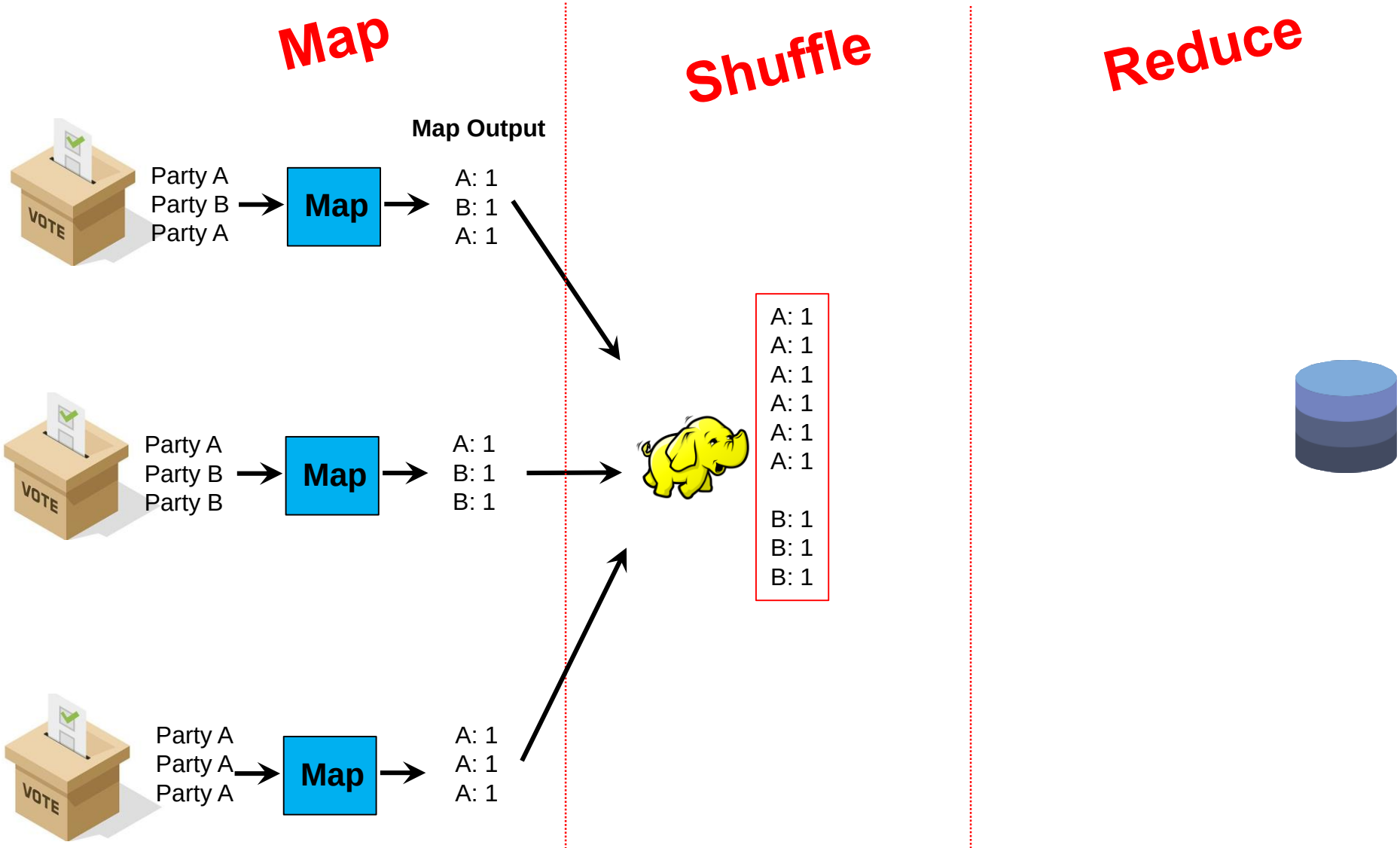
Reduce



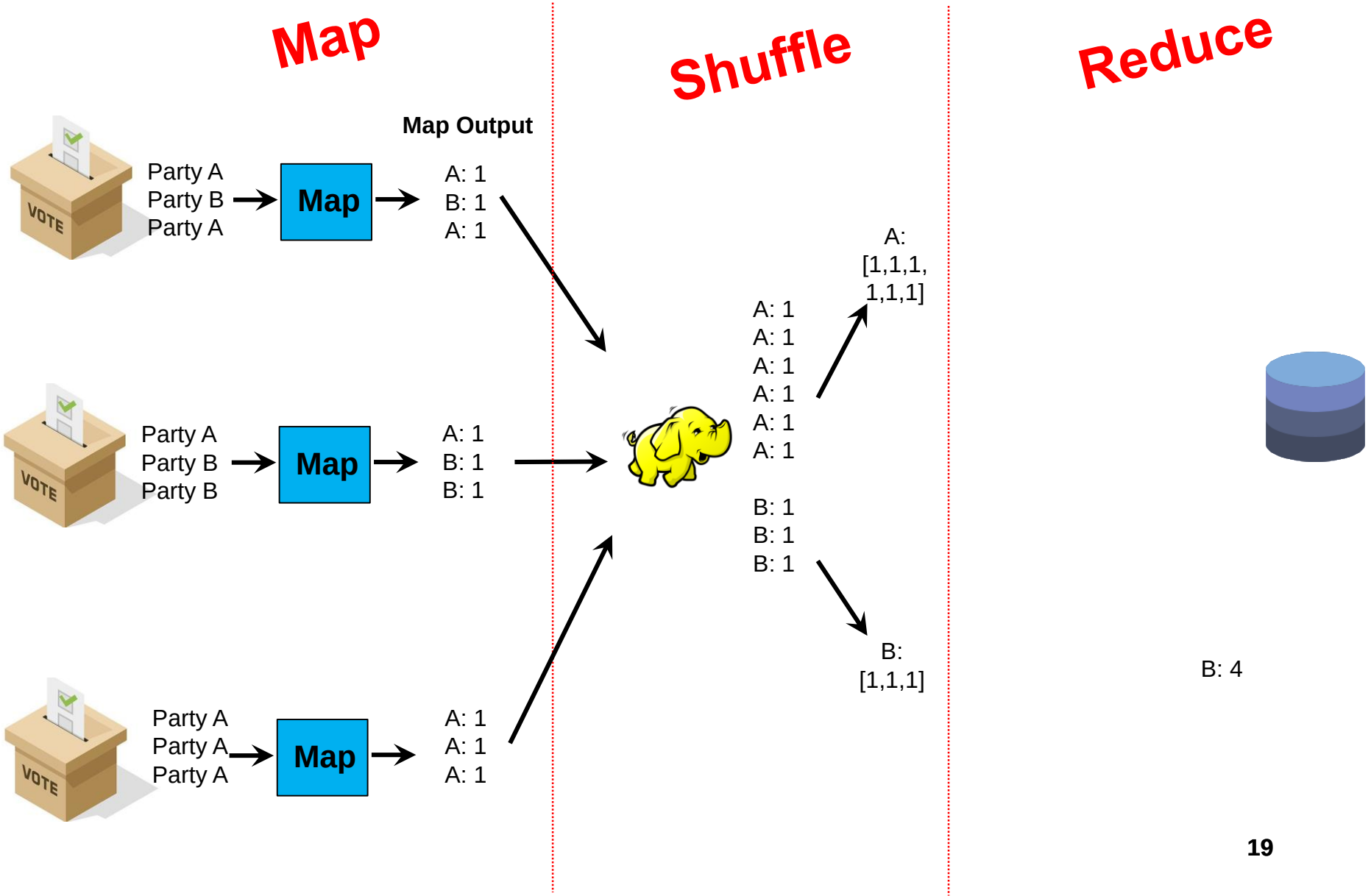
Tabulating Election Results: MapReduce



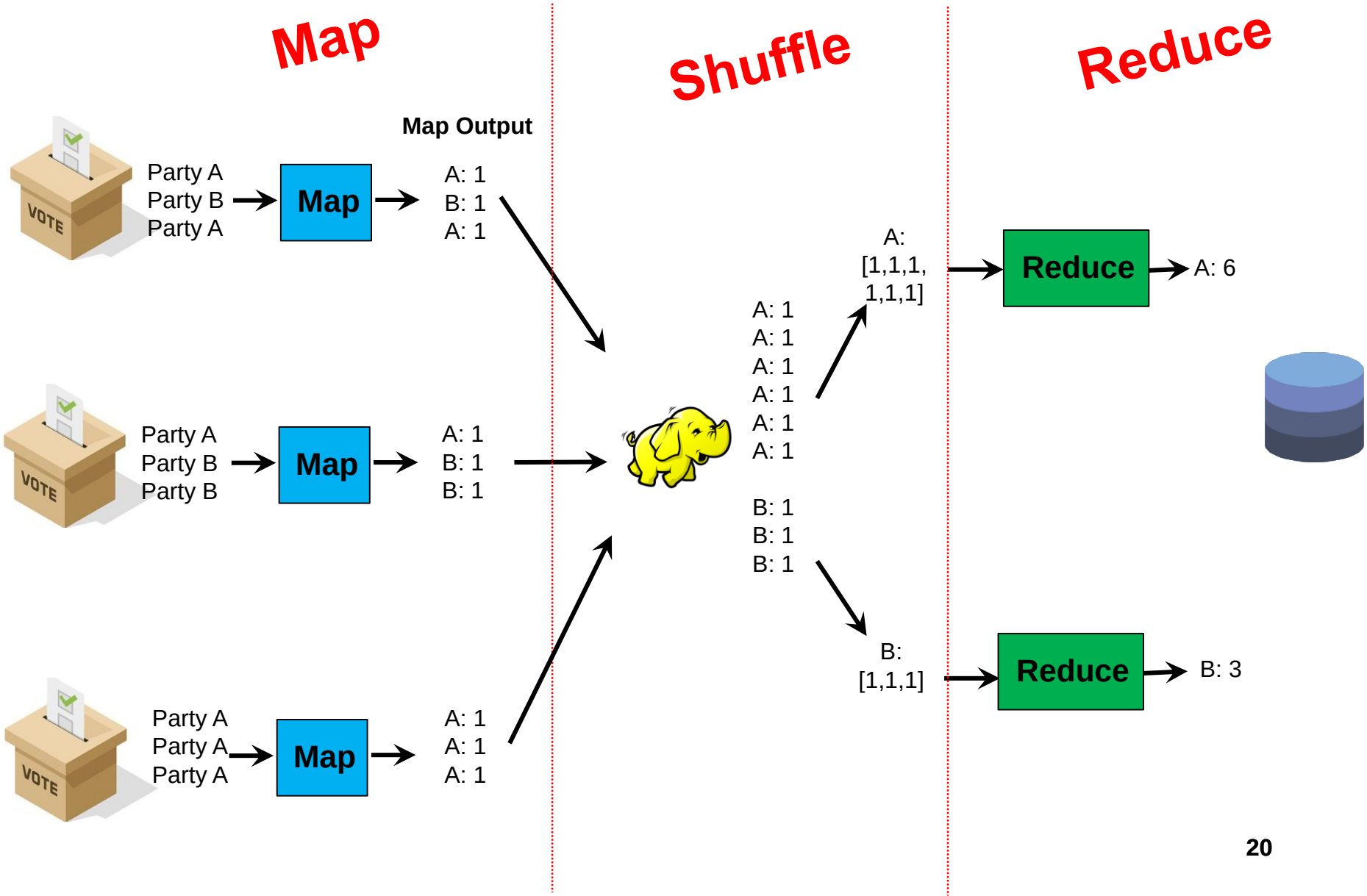
Tabulating Election Results: MapReduce



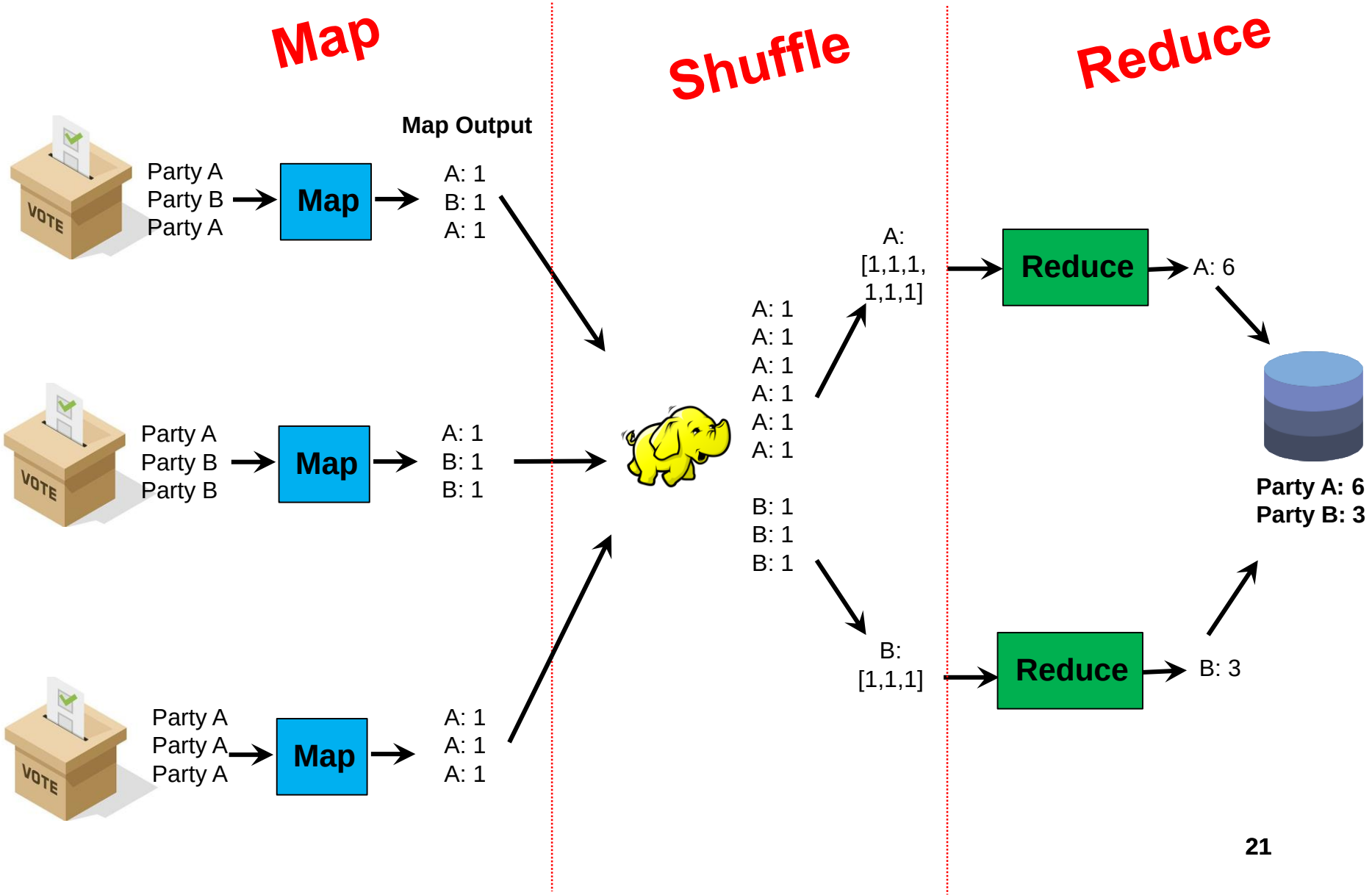
Tabulating Election Results: MapReduce



Tabulating Election Results: MapReduce



Tabulating Election Results: MapReduce



Writing MapReduce Programs

- Programmers specify two functions:

  map $(k_1, v_1) \rightarrow \text{List}(k_2, v_2)$

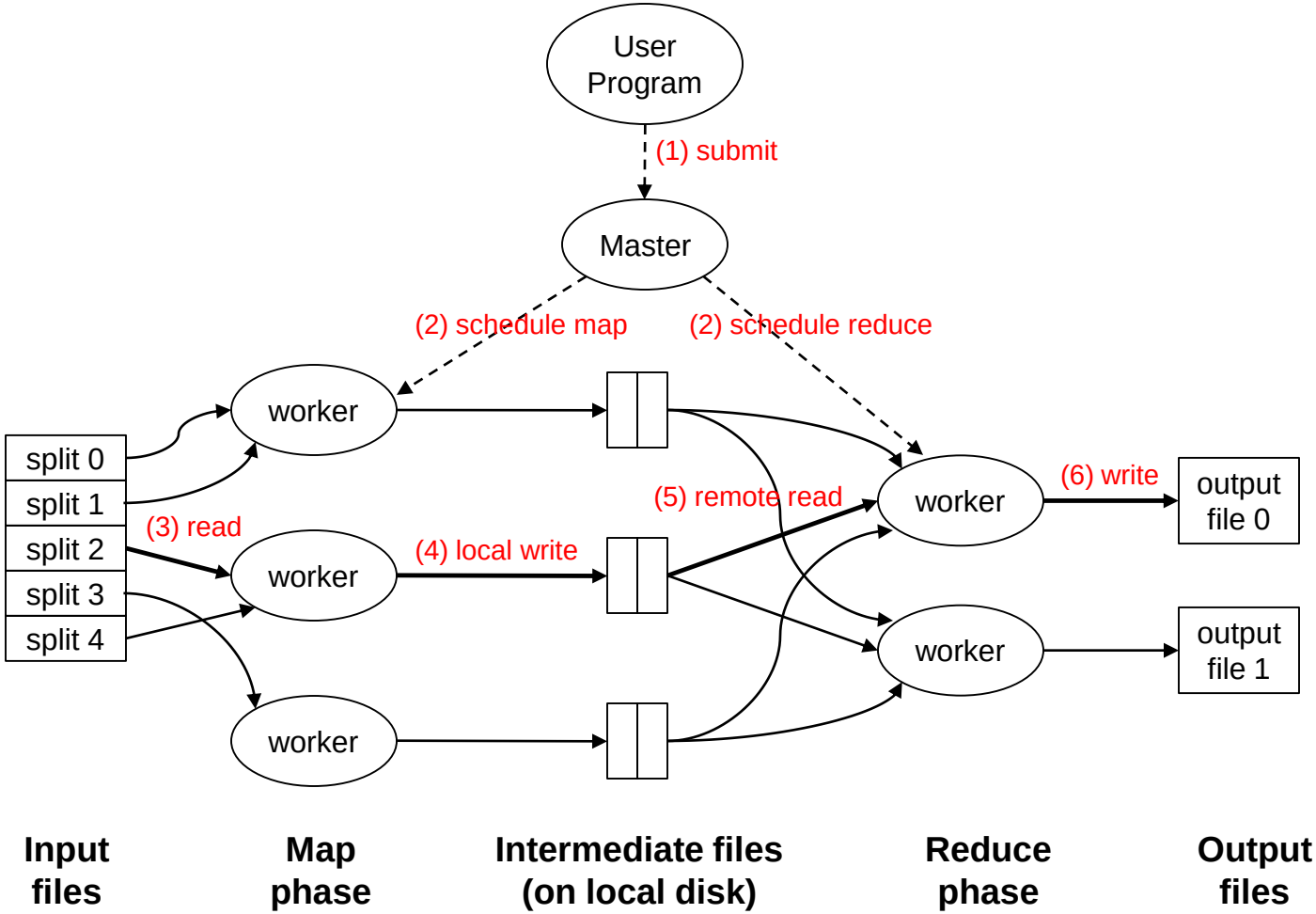
  reduce $(k_2, \text{List}(v_2)) \rightarrow \text{List}(k_3, v_3)$

  - All values with the same key are sent to the same reducer
- The execution framework handles the challenging issues…
  - How do we assign work units to workers?
  - What if we have more work units than workers?
  - What if workers need to share partial results?
  - How do we aggregate partial results?
  - How do we know all the workers have finished?
  - What if workers die/fail?

MapReduce “Runtime”

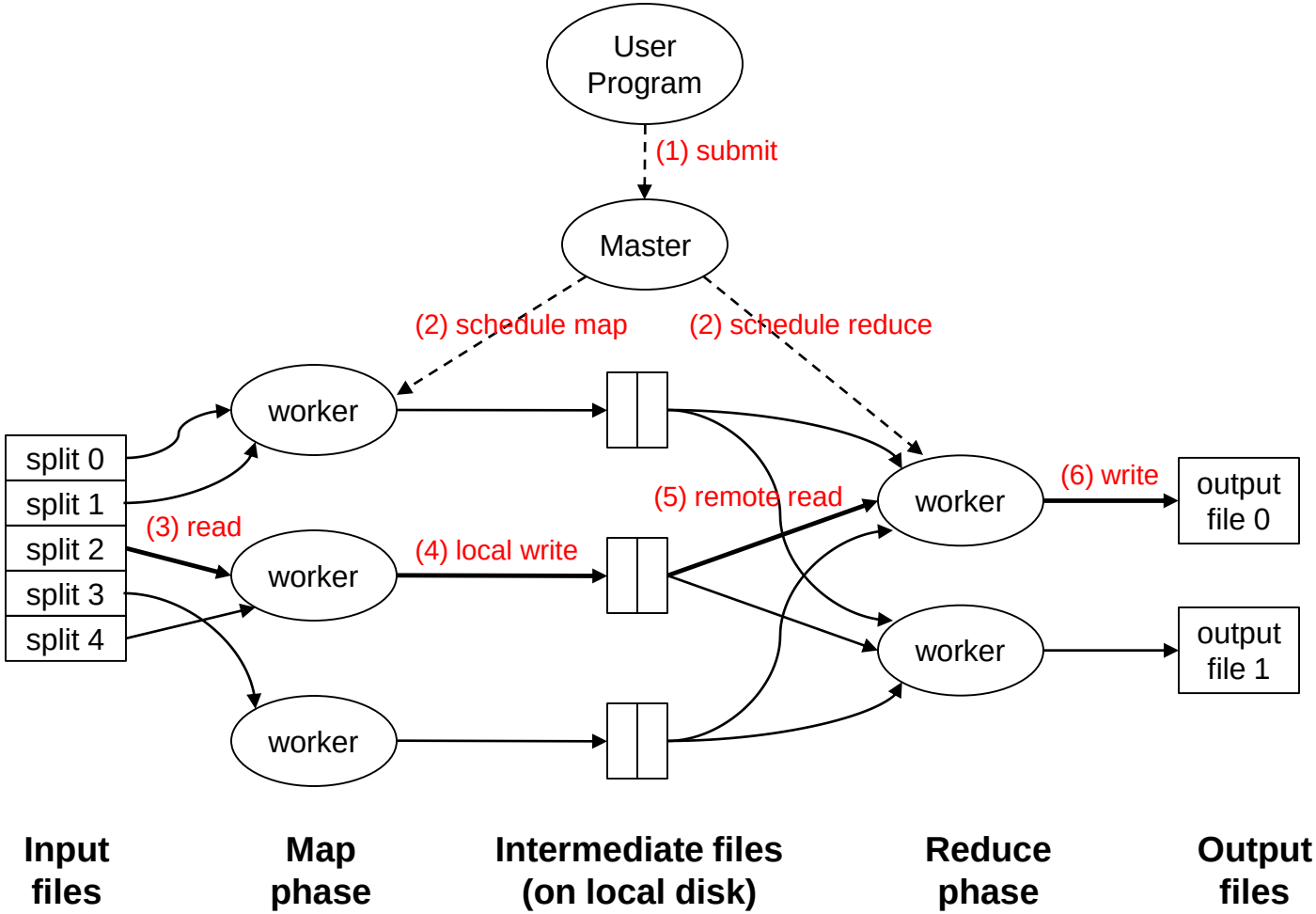
- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles “data distribution”
  - Moves processes to data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed file system (later)

MapReduce Implementation



Adapted from (Dean and Ghemawat, OSDI 2004)

MapReduce Implementation

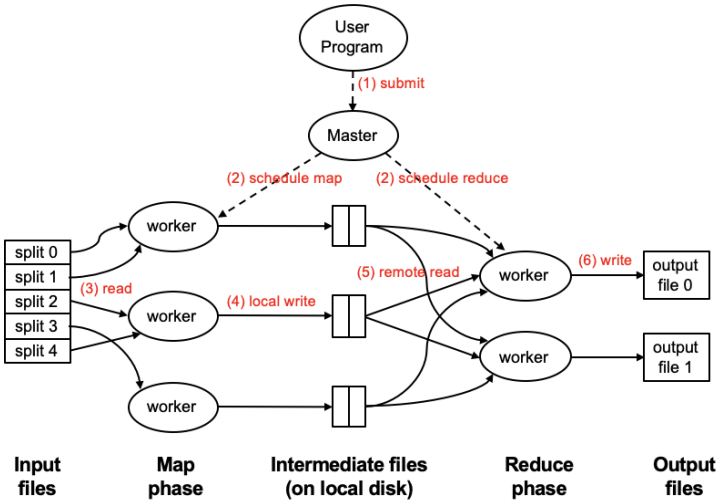


Q: What disadvantages are there if the size of each split (or chunk) is too big or small?

Adapted from (Dean and Ghemawat, OSDI 2004)

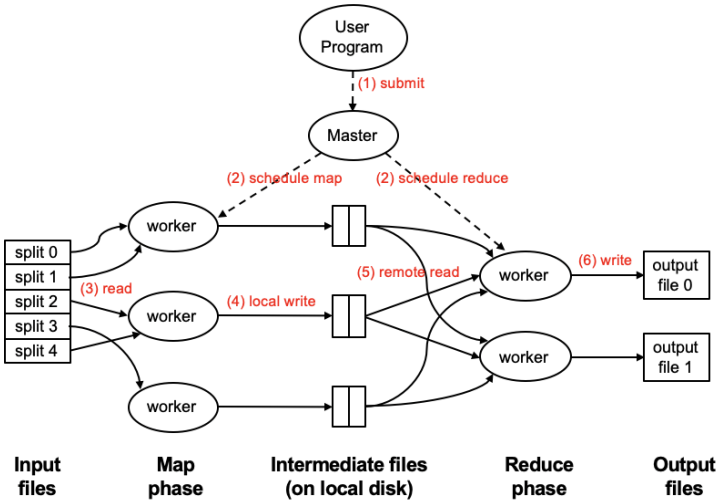
Important Clarifications: Map Task, Mapper, Map Function?

- These terms can get confusing, please be clear on them
- **Map Task** is a basic unit of work; it is typically 128MB. At the beginning the input is broken into splits of 128MB. A map task is a **job** requiring to process one split; not a physical machine.
- A single physical machine (or **worker**) can handle multiple map tasks. Typically, when a machine completes a map task (e.g. split 0), it is re-assigned to another task (e.g. split 3)



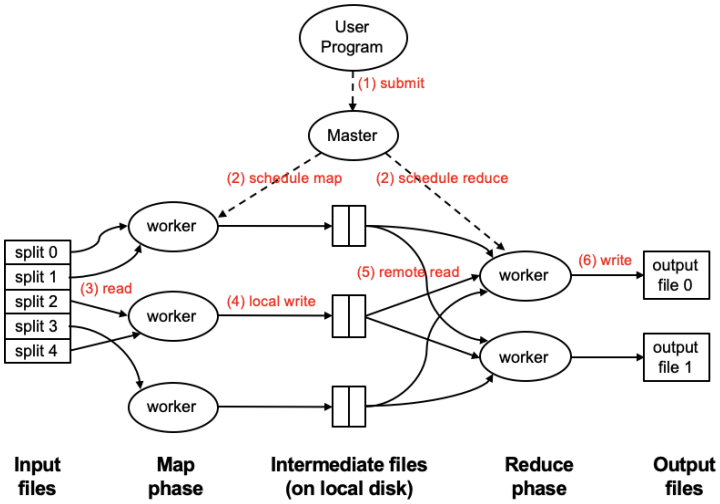
Important Clarifications: Map Task, Mapper, Map Function?

- **A "mapper" or "reducer"** will generally refer to the process executing a map or reduce task, not to physical machines.
- E.g. in this diagram there are 5 map tasks, and thus 5 mappers, but only 3 physical machines.



Important Clarifications: Map Task, Mapper, Map Function?

- **"Map Function"** is a single call to the user-defined **map** $(k_1, v_1) \rightarrow \text{List}(k_2, v_2)$ function.
- Note that a single map task can involve many calls to such a map function: e.g. within a 128MB split, there will often be many (key, value) pairs, each of which will produce one call to a map function.



Two more details…

- Barrier between map and reduce phases
  - But we can begin copying intermediate data earlier
- Keys arrive at each reducer in sorted order
  - No enforced ordering *across* reducers

MapReduce Implementations

- Google has a proprietary implementation in C++
  - Bindings in Java, Python
- Hadoop is an open-source implementation in Java
  - Development led by Yahoo, now an Apache project
  - Used in production at Yahoo, Facebook, Twitter, LinkedIn, Netflix, …
  - The *de facto* big data processing platform
  - Large and expanding software ecosystem
- Lots of custom research implementations
  - For GPUs, cell processors, etc.

1. **MapReduce**
   a. Basic MapReduce
   b. Partition and Combiner
   c. Examples
2. Hadoop File System

Source: Google

Writing MapReduce Programs

- Programmers specify two functions:

  map $(k_1, v_1) \rightarrow \text{List}(k_2, v_2)$

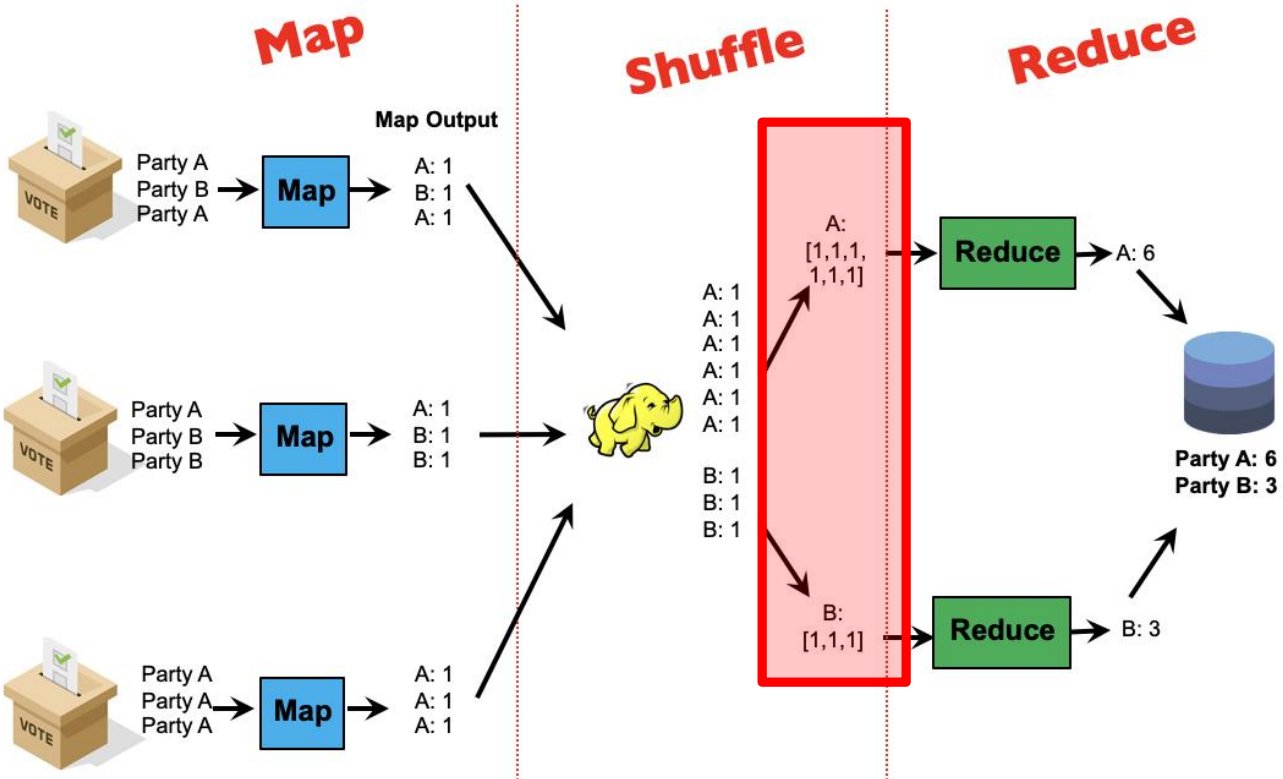
  reduce $(k_2, \text{List}(v_2)) \rightarrow \text{List}(k_3, v_3)$

  - All values with the same key are reduced together
- The execution framework handles everything else…
- Not quite…usually, programmers **optionally** also specify

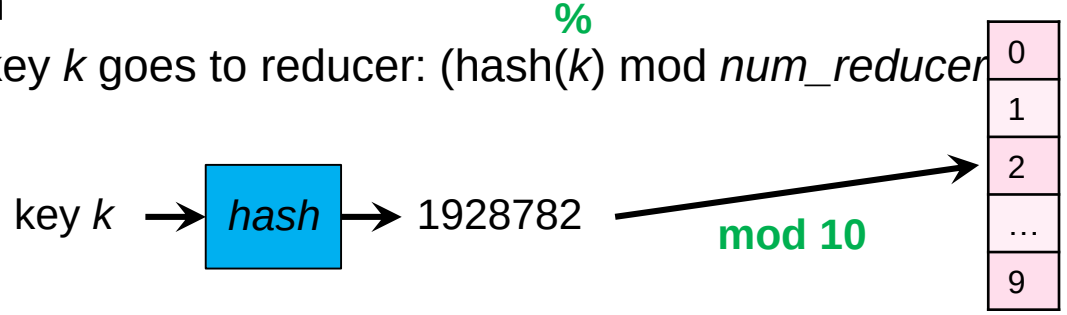
  partition, and **combine** functions

  - These are an optional optimization to reduce network traffic

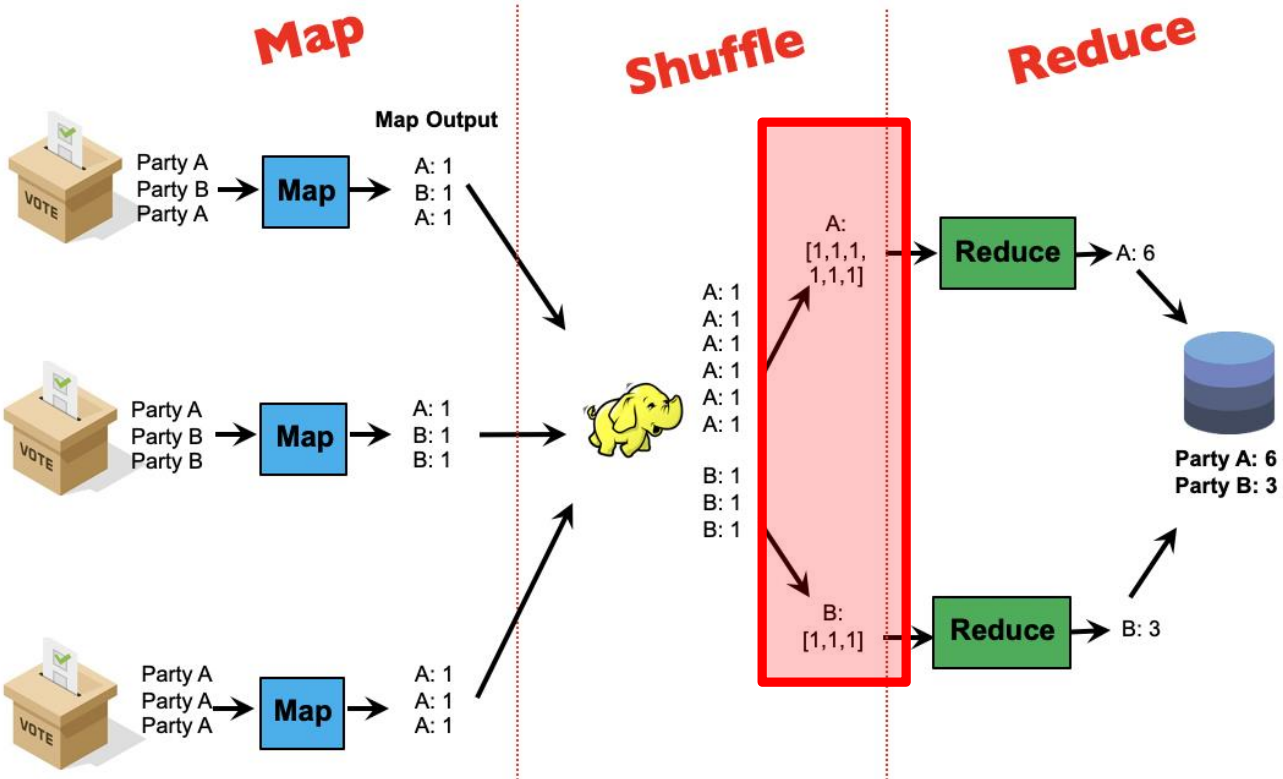
Partition Step



- Note that key A went to reducer 1, and key B went to reducer 2
- By default, the assignment of keys to reducers is determined by a **hash function**
  - e.g., key *k* goes to reducer: (hash(*k*) mod *num_reducer*)
-

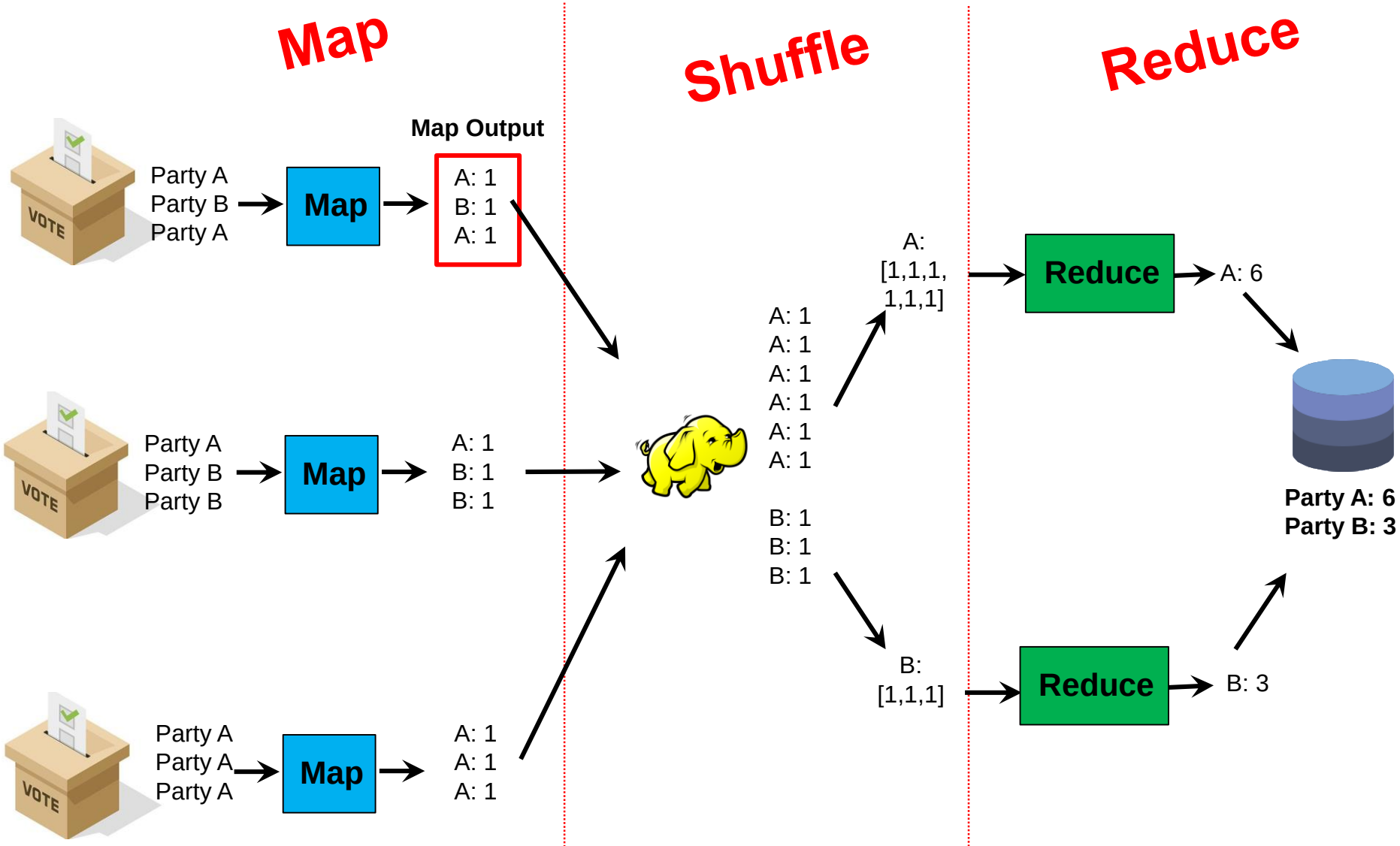


Partition Step



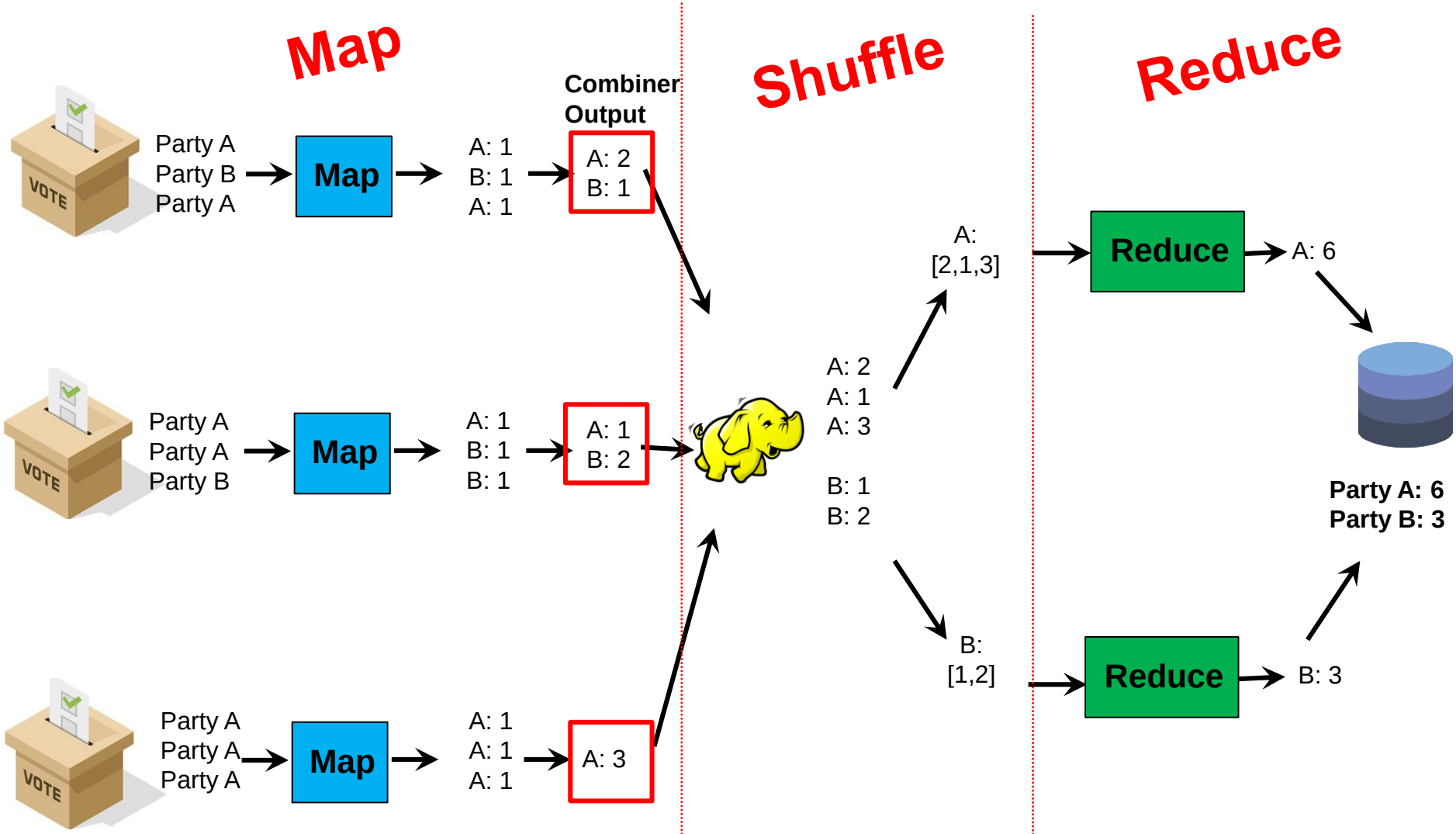
- Note that key A went to reducer 1, and key B went to reducer 2
- By default, the assignment of keys to reducers is determined by a **hash function**
  - e.g., key *k* goes to reducer: (hash(*k*) % mod *num_reducers)*
- User can optionally implement a custom partition, e.g. to better spread out the load among reducers (if some keys have much more values than others)

Combiner Step



- Writing map output to disk is expensive! Can we reduce disk writes?

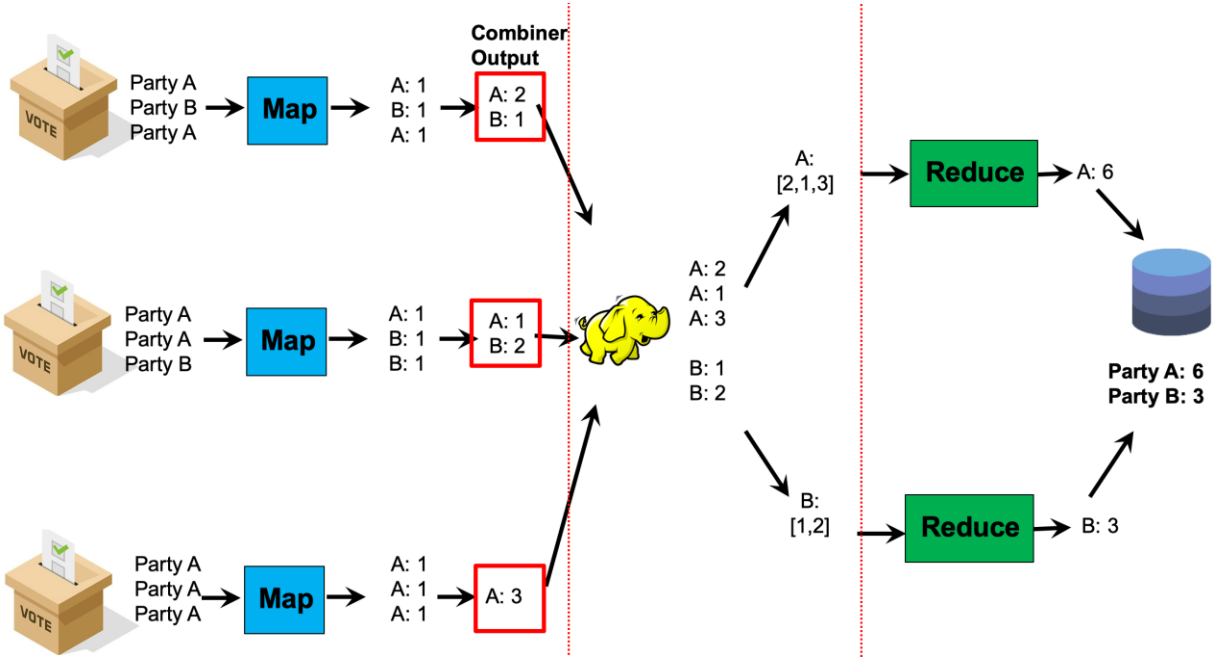
Combiner Step



- Combiners **locally aggregate** output from mappers.
- Combiners are 'mini-reducers': in this example, combiners and reducers are the same function!

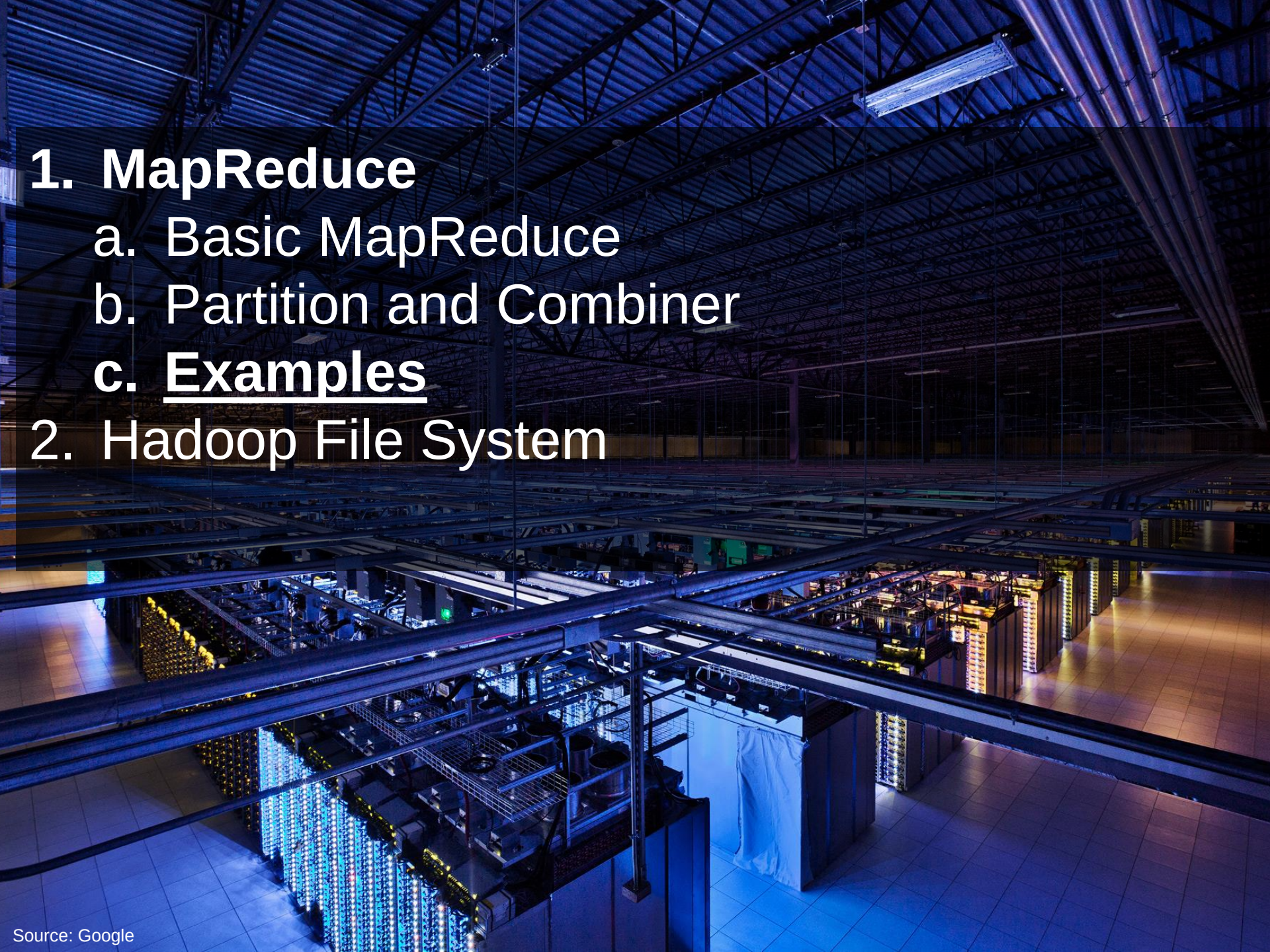
Correctness of Combiner

- It is the user's responsibility to ensure that the combiner does not affect the correctness of the final output, whether the combiner runs 0, 1, or multiple times
  - Example: in election example, the combiner and reducer are a "sum" over values with the same key. Summing can be done in any order without affecting correctness:
    - e.g. sum(sum(1, 1), 1, sum(1, 1, 1)) = sum(1, 1, 1, 1, 1, 1) = 6
  - The same holds for "max" and "min"
  - How about "mean" or "minus"?



Correctness of Combiner

- It is the user's responsibility to ensure that the combiner does not affect the correctness of the final output, whether the combiner runs 0, 1, or multiple times
  - Example: in election example, the combiner and reducer are a "sum" over values with the same key. Summing can be done in any order without affecting correctness:
    - e.g. sum(sum(1, 1), 1, sum(1, 1, 1)) = sum(1, 1, 1, 1, 1, 1) = 6
  - The same holds for "max" and "min"
  - How about "mean" or "minus"?
    - Answer: No! E.g. mean(mean(1, 1), 2) ≠ mean(1, 1, 2).
  - (Optional) In general, it is correct to use reducers as combiners if the reduction involves a binary operation (e.g. +) that is both
    - **Associative:** a + (b + c) = (a + b) + c
    - **Commutative:** a + b = b + a

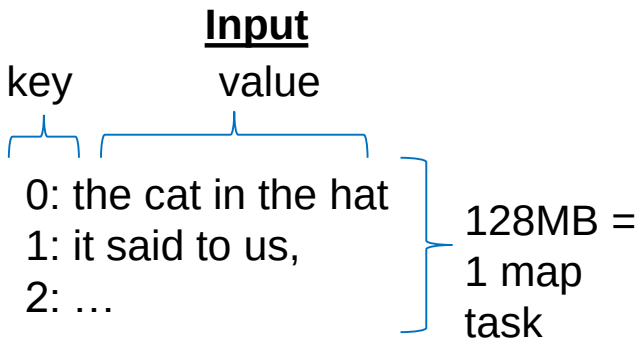


1. **MapReduce**
   a. Basic MapReduce
   b. Partition and Combiner
   c. Examples
2. Hadoop File System

Source: Google

Word Count in Pseudo Code

```
1   class Mapper {
2     def map(key: Long, value: Text) = {
3       for (word <- tokenize(value)) {
4         emit(word, 1)
5     }
6   }
7
8   class Reducer {
9     def reduce(key: Text, values: Iterable[Int]) = {
10      for (value <- values) {
11        sum += value
12      }
13      emit(key, sum)
14    }
15  }
```



This mapper processes each word one by one, and emits a “1”, to be summed by the reducers.

Word Count in Hadoop: Skeleton

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
    …
    1. Map task function
    2. Reduce task function
    3. Driver function
    …

}
```

Necessary imports

Word Count in Hadoop: Map

```
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);
    }
  }
}
```

Word Count in Hadoop: Reduce

```
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);
  }
}
```

Word Count in Hadoop: Driver

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(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);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Hadoop/MapReduce Examples and Resources

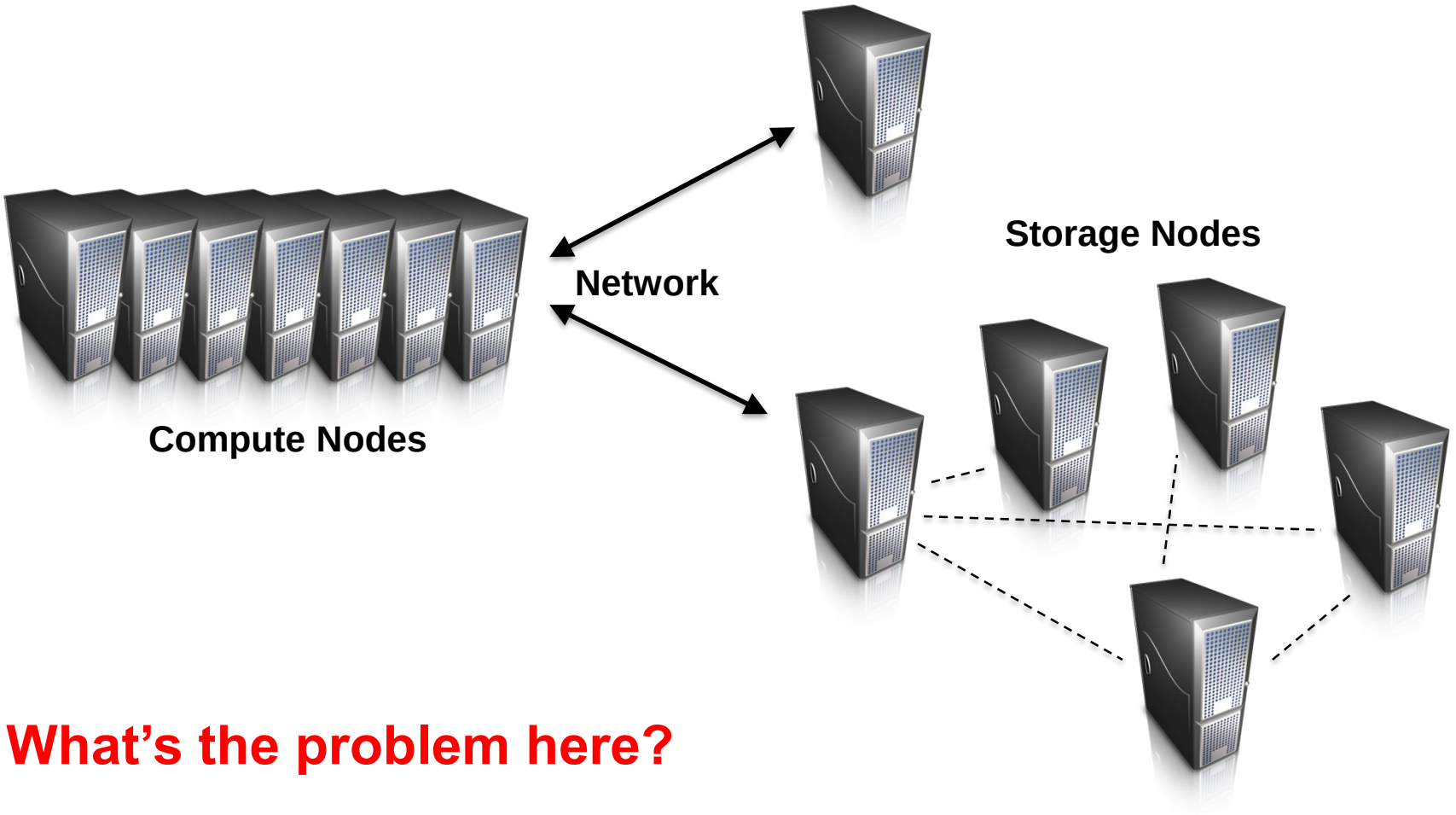
- Hadoop movielens dataset analysis: https://github.com/braineering/moviedoop
- Hadoop wikipedia pageranking:
  - https://github.com/abij/hadoop-wiki-pageranking
- Adam genomic data analyzer:
  - https://github.com/bigdatagenomics/adam
- Other resources (Libs/Frameworks/Books/Tutorials)
  - See awesome-hadoop:

https://github.com/youngwookim/awesome-hadoop

1. **MapReduce**
   a. Basic MapReduce
   b. Partition and Combiner
   c. Examples
2. **Hadoop File System**

Source: Google

How do we get data to the workers?



What's the problem here?

Distributed File System

- Don’t move data to workers… move workers to the data!
  - Store data on the local disks of nodes in the cluster
  - Start up the workers on the node that has the data local
- A distributed file system is the answer
  - GFS (Google File System) for Google’s MapReduce
  - HDFS (Hadoop Distributed File System) for Hadoop

GFS / HDFS: Assumptions

- Commodity hardware instead of “exotic” hardware
  - Scale “out”, not “up”
- High component failure rates
  - Inexpensive commodity components fail all the time
- “Modest” number of huge files
- Files are write-once, mostly appended to
- Large streaming reads instead of random access
  - High sustained throughput over low latency

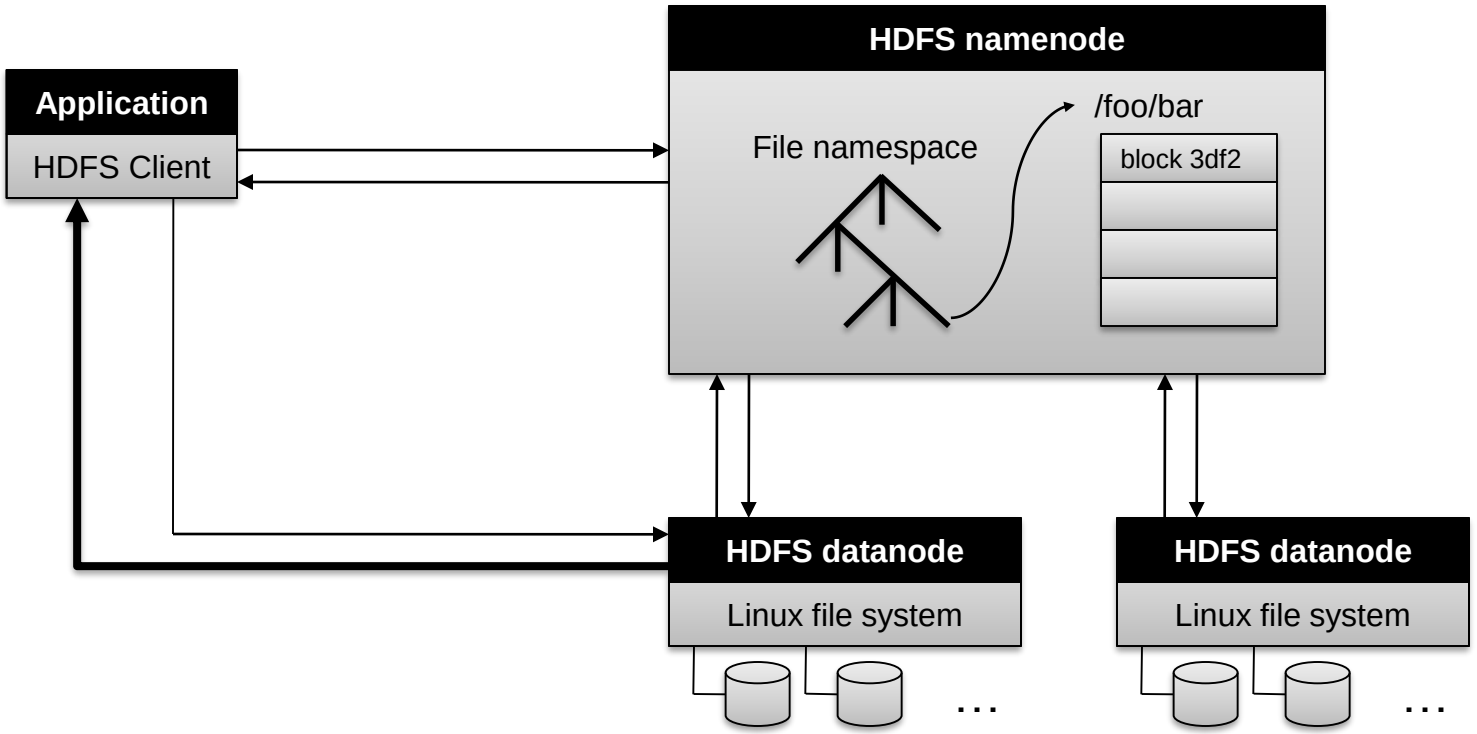
GFS slides adapted from material by (Ghemawat et al., SOSP 2003)

Design Decisions

- Files stored as chunks
  - Fixed size (64MB for GFS, 128MB for HDFS)
- Reliability through replication
  - Each chunk replicated across 3+ chunkservers
- Single master to coordinate access, keep metadata
  - Simple centralized management

HDFS = GFS clone (same basic ideas)

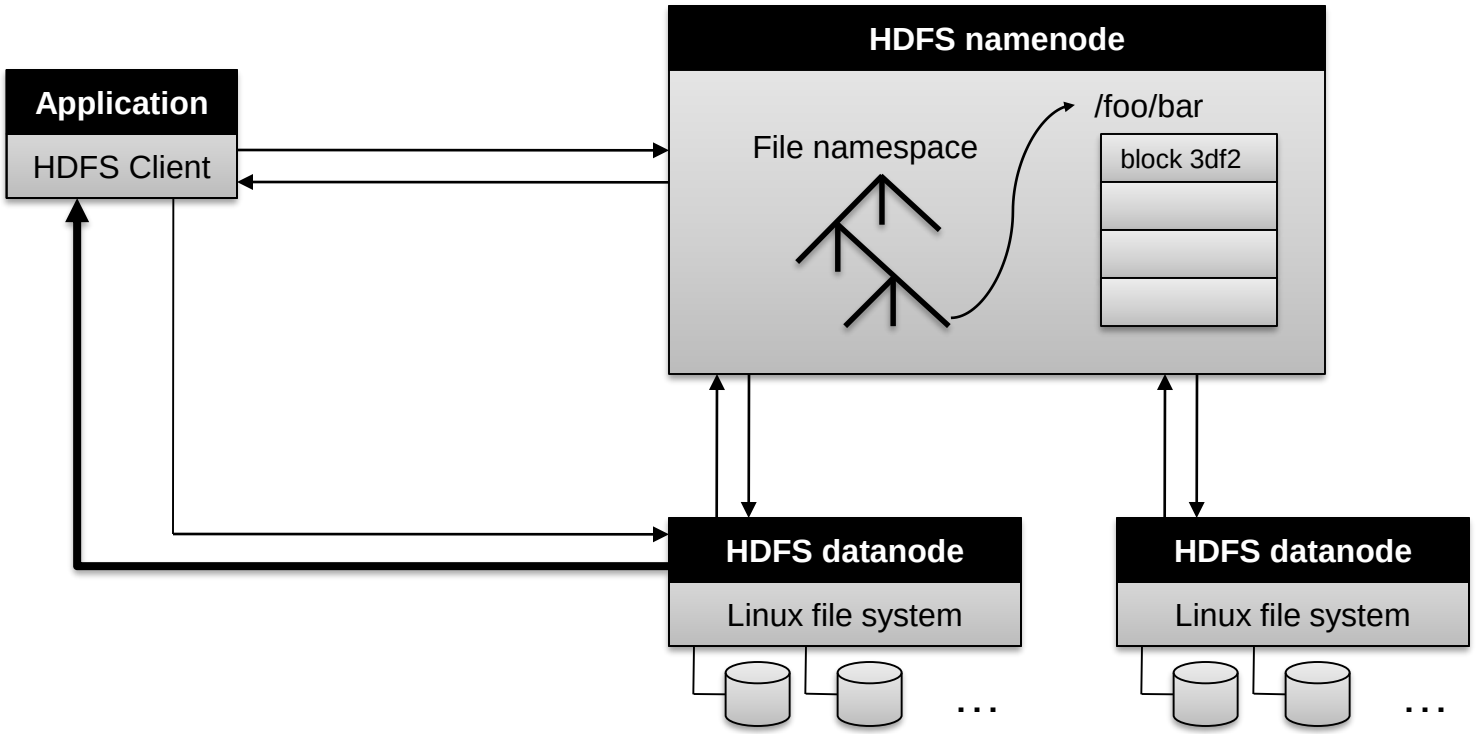
HDFS Architecture



Q: How to perform replication when writing data?

Adapted from (Ghemawat et al., SOSP 2003)

HDFS Architecture



Q: How to perform replication when writing data?

A: Namenode decides which datanodes are to be used as replicas. The 1st datanode forwards data blocks to the 1st replica, which forwards them to the 2nd replica, and so on.

Adapted from (Ghemawat et al., SOSP 2003)

Namenode Responsibilities

- Managing the file system namespace:
  - Holds file/directory structure, metadata, file-to-block mapping, access permissions, etc. Coordinating file operations:
  - Directs clients to datanodes for reads and writes
  - No data is moved through the namenode
- Maintaining overall health:
  - Periodic communication with the datanodes
  - Block re-replication and rebalancing
  - Garbage collection

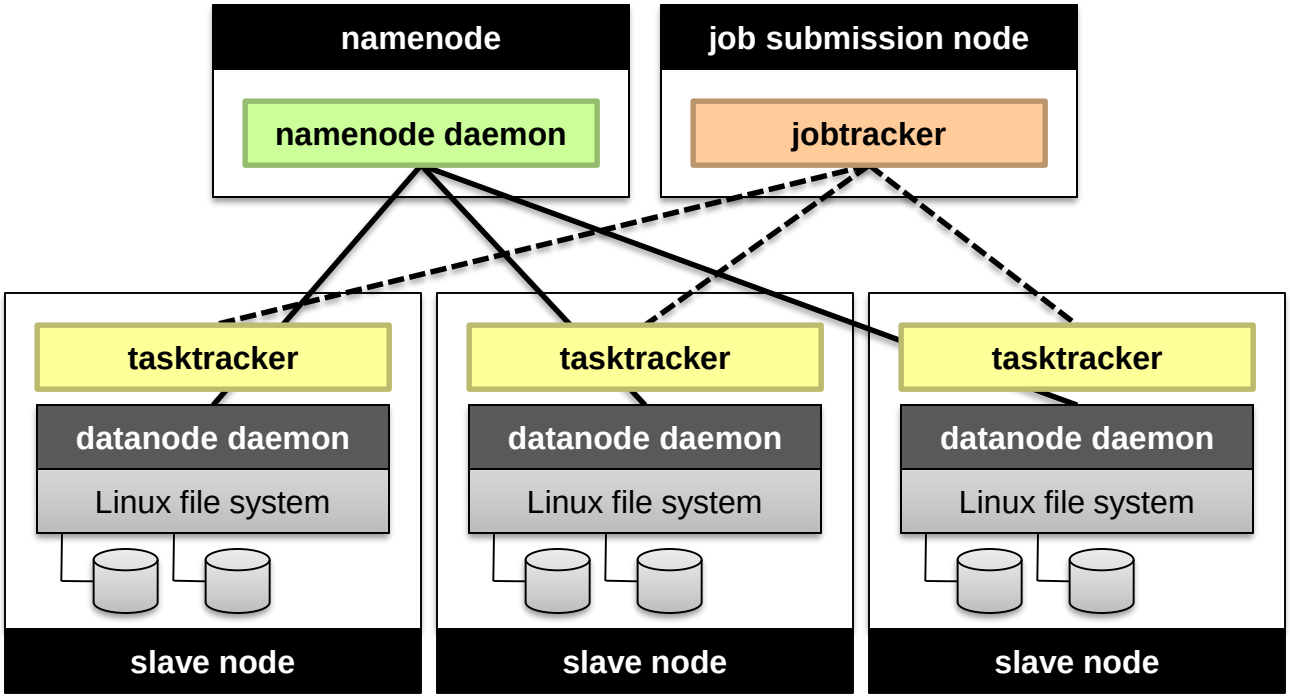
- **Q**: What if the namenode's data is lost?

Namenode Responsibilities

- Managing the file system namespace:
  - Holds file/directory structure, metadata, file-to-block mapping, access permissions, etc. Coordinating file operations:
  - Directs clients to datanodes for reads and writes
  - No data is moved through the namenode
- Maintaining overall health:
  - Periodic communication with the datanodes
  - Block re-replication and rebalancing
  - Garbage collection

- **Q**: What if the namenode's data is lost?
- **A**: All files on the filesystem cannot be retrieved since there is no way to reconstruct them from the raw block data. Fortunately, Hadoop provides 2 ways of improving resilience, through backups and secondary namenodes (out of syllabus, but you can refer to Resources for details)

Putting everything together…



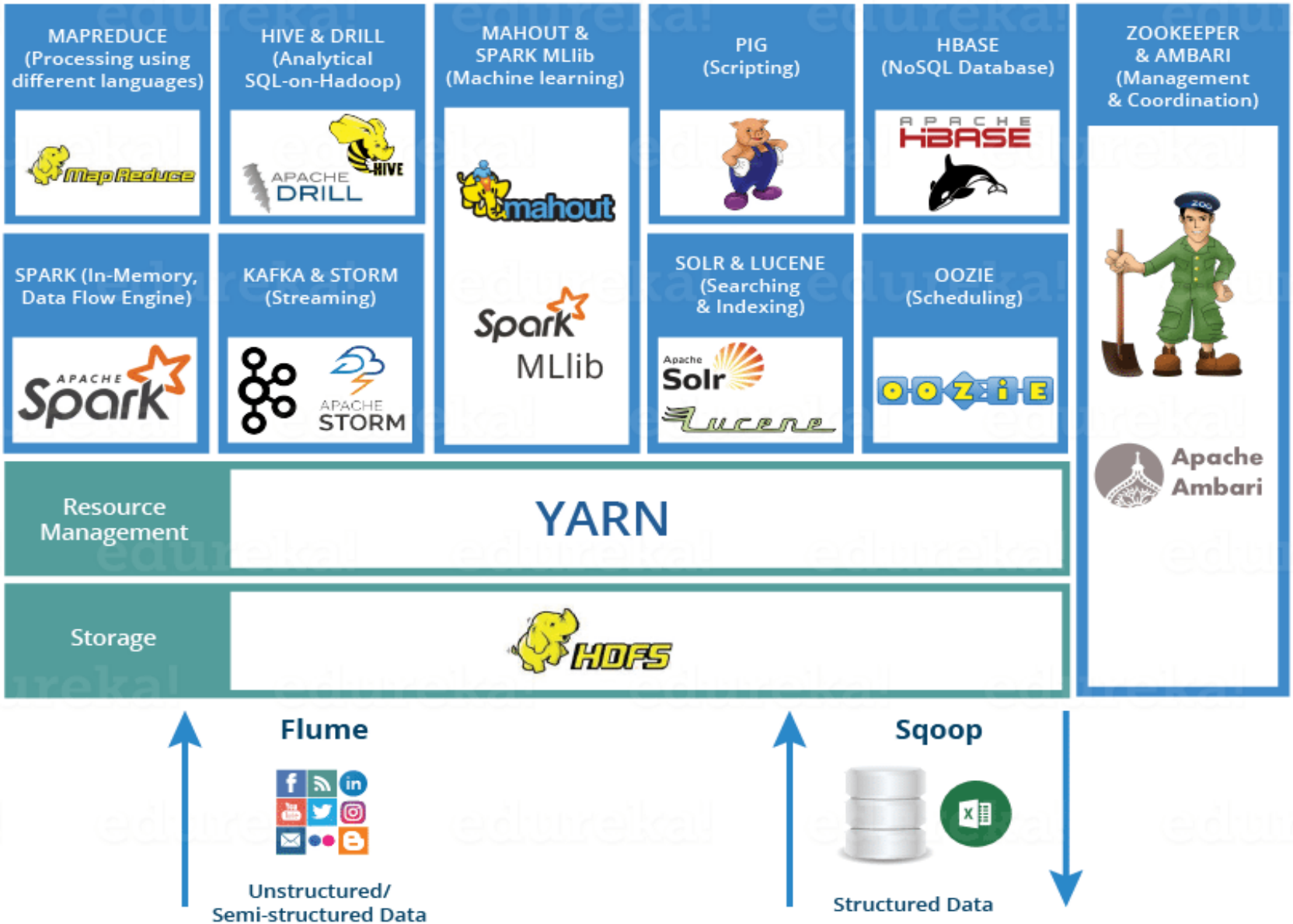
Further Reading

- “Mining Massive Datasets”, Chapter 2.1-2.5
- Jeffrey Dean and Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters
  - http://labs.google.com/papers/mapreduce.html
- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung: The Google File System
  - http://labs.google.com/papers/gfs.html

Take-away

- Big data needs new programing abstractions and runtime systems, rather than conventional approaches in parallelization.
- With the popularity of Hadoop, MapReduce programming framework has become quite common for Big Data.
- As we will see, MapReduce can be used to efficiently and effectively develop various applications.
  - Coming lectures: large relational database and data mining

Hadoop Ecosystem



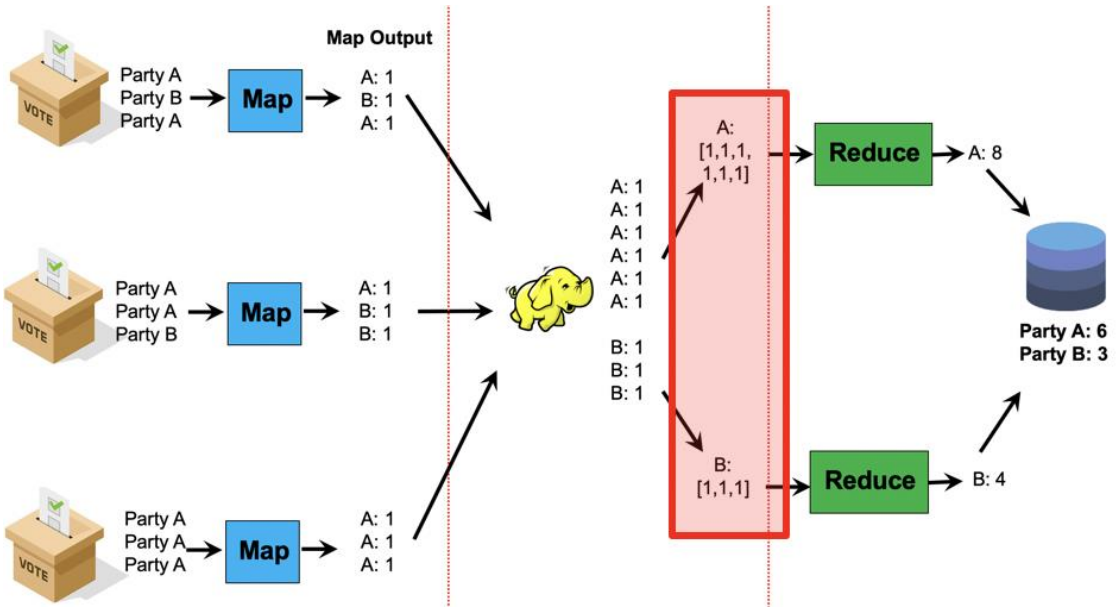
Q: Which statement(s) about Hadooop's partitioner is true?

1. The partitioner determines which keys are processed on which mappers.
2. The partitioner can improve performance when some keys are much more common than others.
3. The partitioner is run in between the map and reduce phases.

Q: Which statement(s) about Hadooop's partitioner is true?

1. The partitioner determines which keys are processed on which mappers.
2. The partitioner can improve performance when some keys are much more common than others.
3. The partitioner is run in between the map and reduce phases.

- **A: 2 and 3**



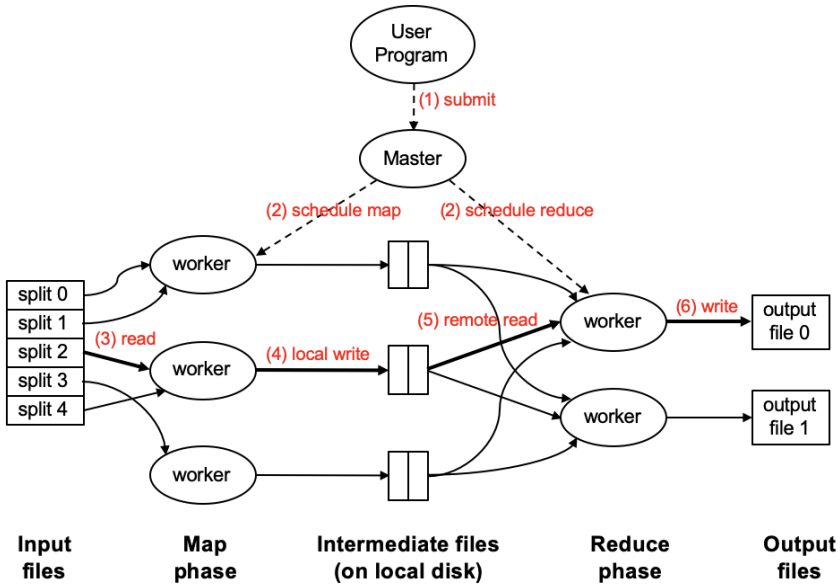
Q: True or false: the Shuffle stage of MapReduce is run within the Master node.

1. True
2. False

Q: True or false: the Shuffle stage of MapReduce is run within the Master node.

1. True
2. False

A: False (it runs in the worker nodes)



Resources

- Hadoop: The Definitive Guide (by Tom White)
- Hadoop Wiki
  - Introduction
    - http://wiki.apache.org/lucene-hadoop/
  - Getting Started
    - http://wiki.apache.org/lucene-hadoop/GettingStartedWithHadoop
  - Map/Reduce Overview
    - http://wiki.apache.org/lucene-hadoop/HadoopMapReduce
    - http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses
  - Eclipse Environment
    - http://wiki.apache.org/lucene-hadoop/EclipseEnvironment
- Javadoc
  - http://lucene.apache.org/hadoop/docs/api/
- YARN
  - https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

Resources

- Releases from Apache download mirrors
  - http://www.apache.org/dyn/closer.cgi/lucene/hadoop/
- Nightly builds of source
  - http://people.apache.org/dist/lucene/hadoop/nightly/
- Source code from subversion
  - http://lucene.apache.org/hadoop/version_control.html

Acknowledgement

- Much of course slides are adopted/revised from
  - Jimmy Lin, http://lintool.github.io/UMD-courses/bigdata-2015-Spring/
  - Bryan Hooi
- Some slides are also adopted/revised from
  - Claudia Hauff, TU Delft: https://chauff.github.io/
  - Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2014. Mining of Massive Datasets (2nd ed.). Cambridge University Press. http://www.mmds.org/