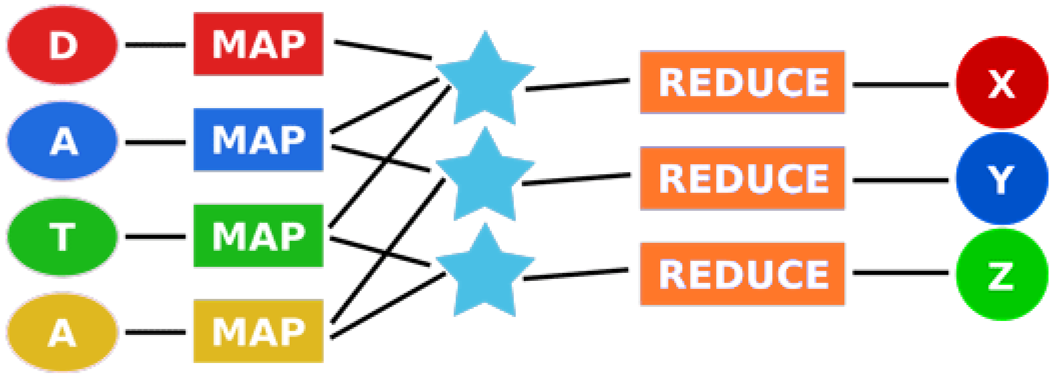
# COMP810 Data Warehousing and Big Data

## Map-Reduce and Hadoop

Dr Weihua Li

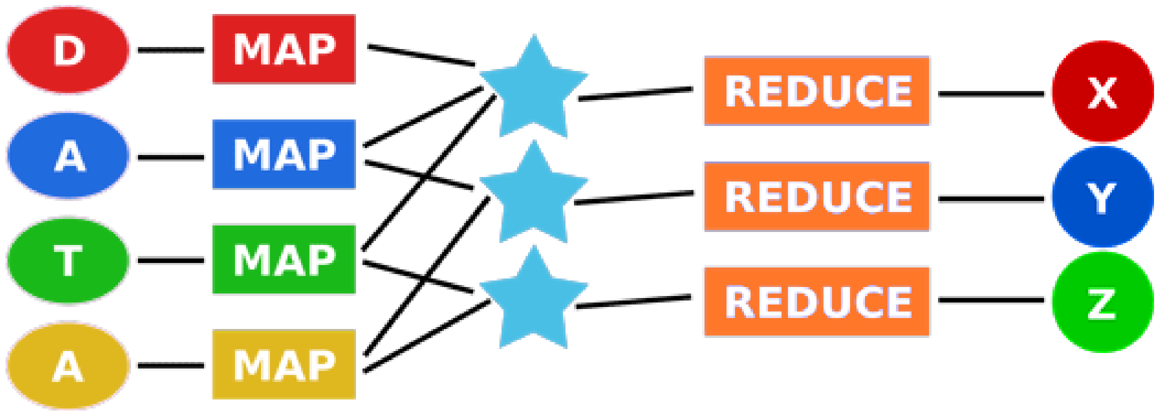
# Outline

- MapReduce

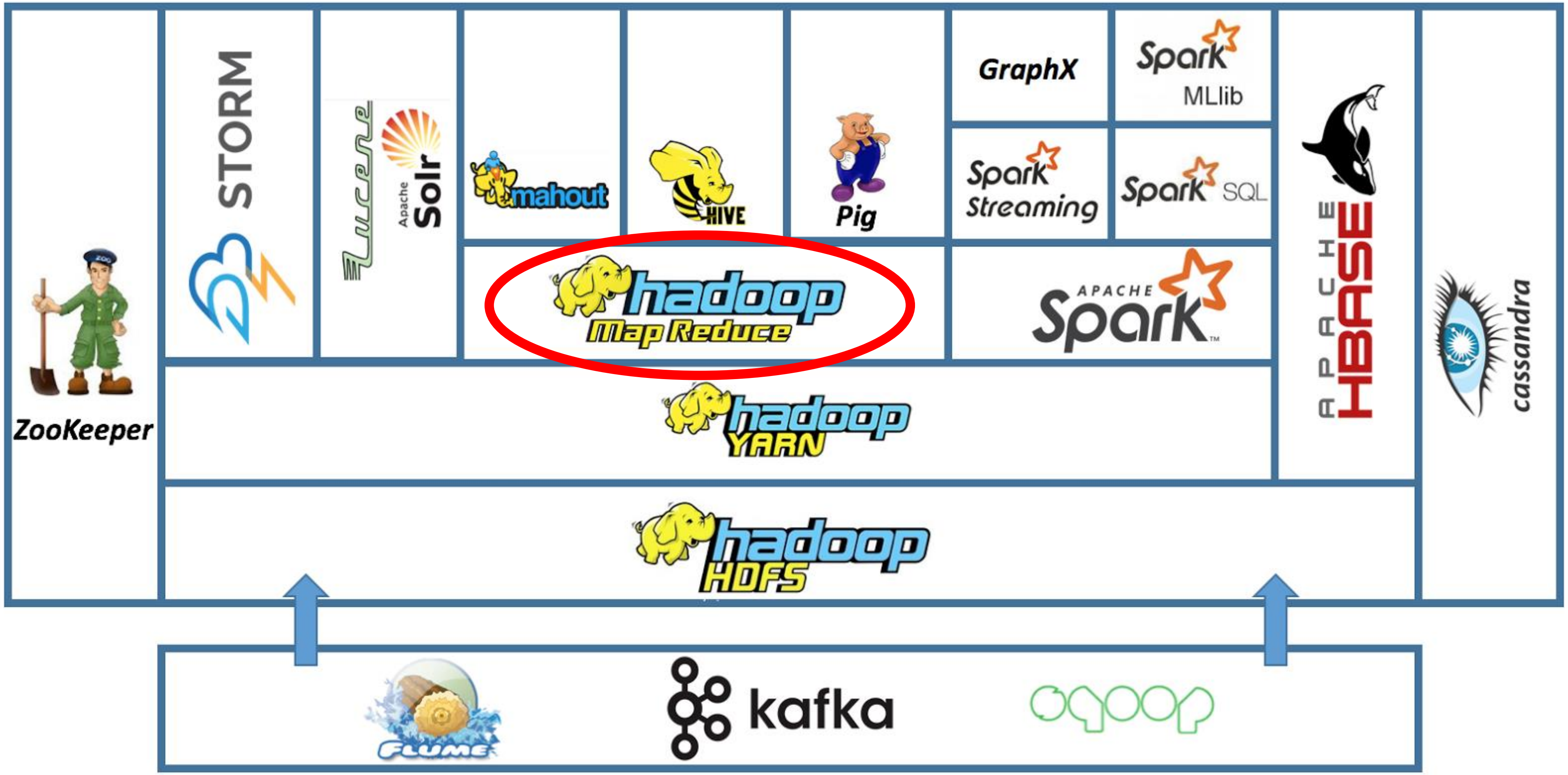



- Apache Hadoop

# MapReduce



# Apache Hadoop Ecosystem



# Motivation: Google Example

20+ billion web pages x 20 KB = 400+ TB

1 computer reads 3.5 GB/sec from disk


- 32 hours (1.3 days) to read the web

Takes even longer to do something useful with the data!

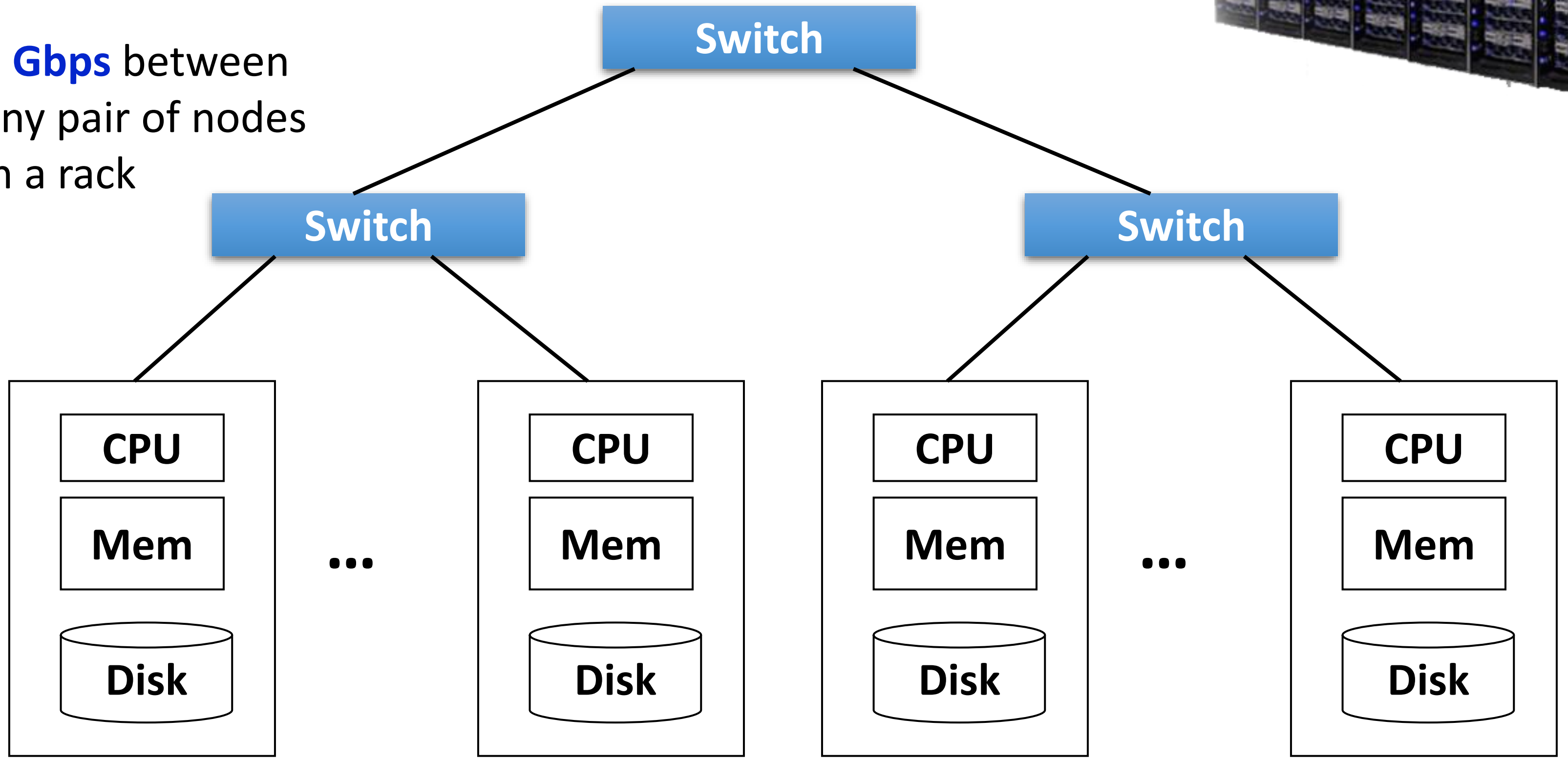
Today, a standard architecture for such problems is:

- Cluster of commodity Linux nodes
- Commodity network (ethernet) to connect them

# Cluster Architecture

**2-10 Gbps** backbone between racks

**1 Gbps** between
any pair of nodes
in a rack




Each rack contains 16-64 nodes.

**In 2011 it was estimated that Google had 1M machines.**

# Cluster Computing Challenges

- Node Failures
  - One server may stay up 3 years (1,000 days)
  - If you have 1,000 servers, expect to have 1 failure/day
  - 1M servers in a cluster, expect to have 1,000 failures every day!


- Network Bottleneck
  - Network bandwidth = 1Gbps
  - Moving 10TB takes 1 day


- Distributed Programming is hard
  - Need a simple model that hides most of the complexity

# MapReduce

- Map-reduce addresses all of these three challenges
  - **Node Failures:** Store data redundantly on multiple nodes for persistence and availability.
  - **Network Bottleneck**: Move computation close to data to minimize data movement.
  - **Uneasy Distributed Programming**: Simple programming model to hide the complexity

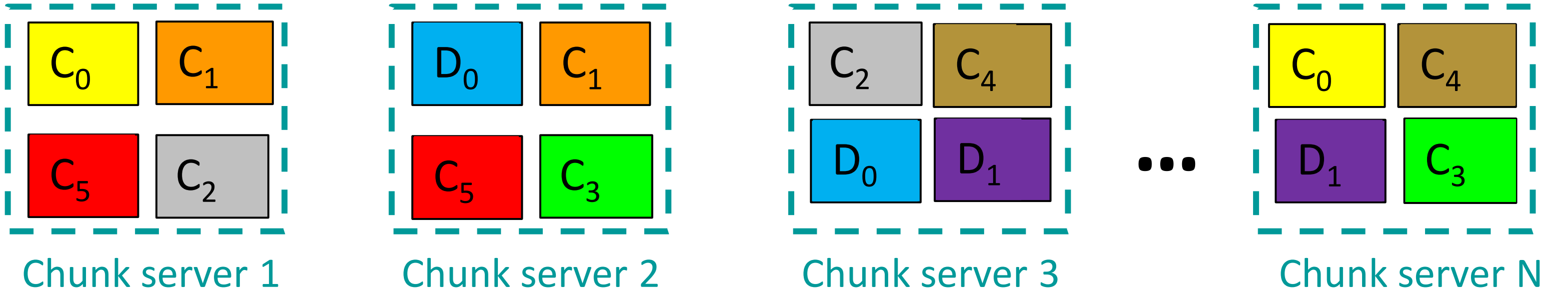
# Redundant Storage Infrastructure

- **Distributed File System**
  - Store data across a cluster, and each piece of data multiple times.
  - Provide global file namespace, redundancy and availability.
  - E.g., Google File System (GFS), Hadoop File System (HDFS)

- **Typical Usage Pattern**
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place, update through appends
  - Reads and appends are common

# Distributed File System

- Reliable distributed file system
- Data kept in “chunks” spread across multiple machines
- Each chunk replicated on different machines
  - Seamless recovery from disk or machine failure

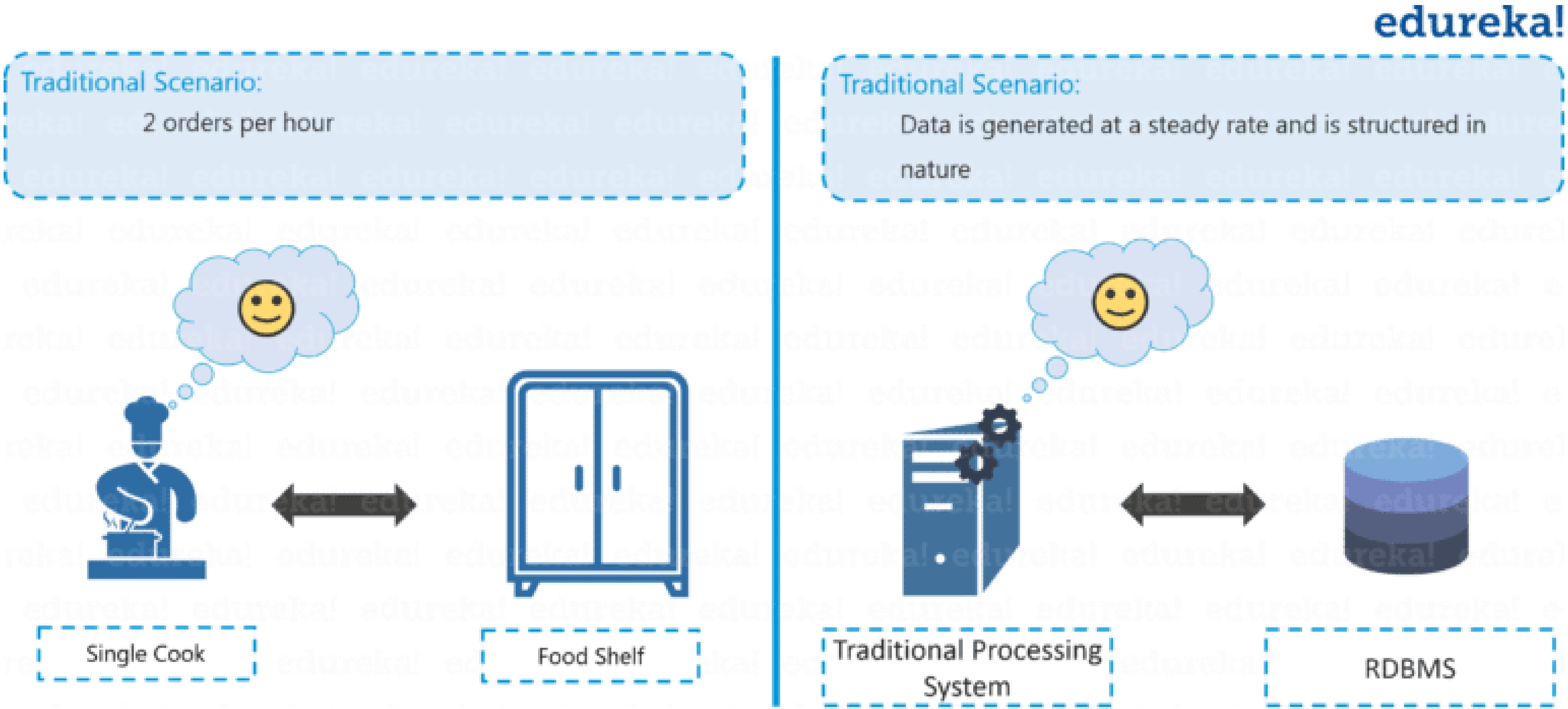



Chunk servers also serve as compute servers

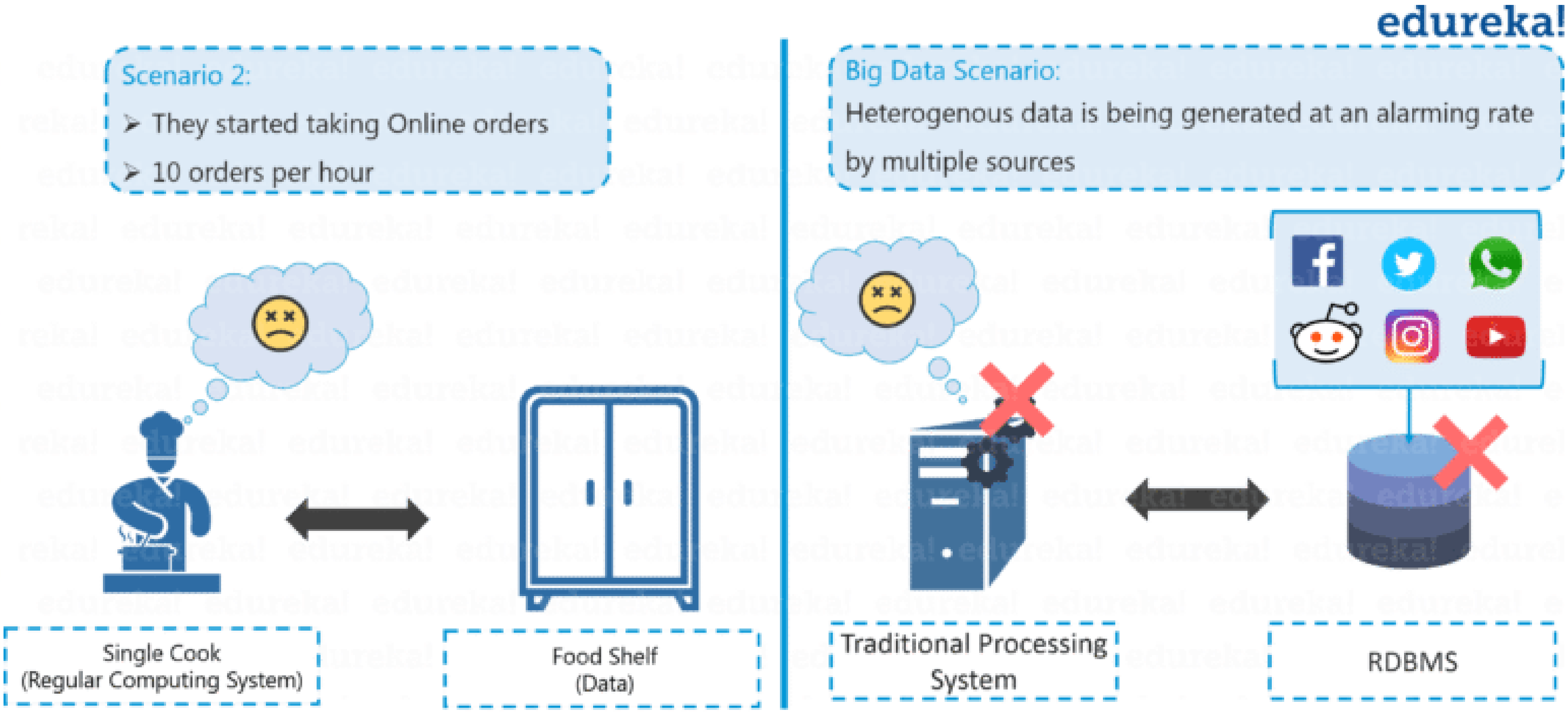
Bring computation directly to the data!

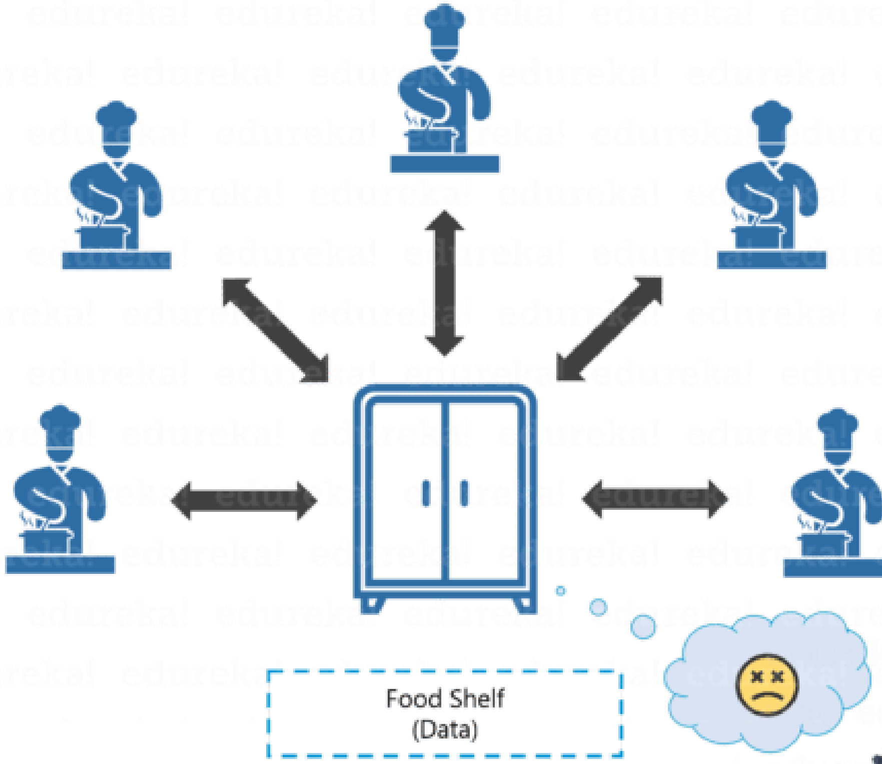
# Map-Reduce – Restaurant Analogy

# Traditional Scenario



# Big Data Scenario





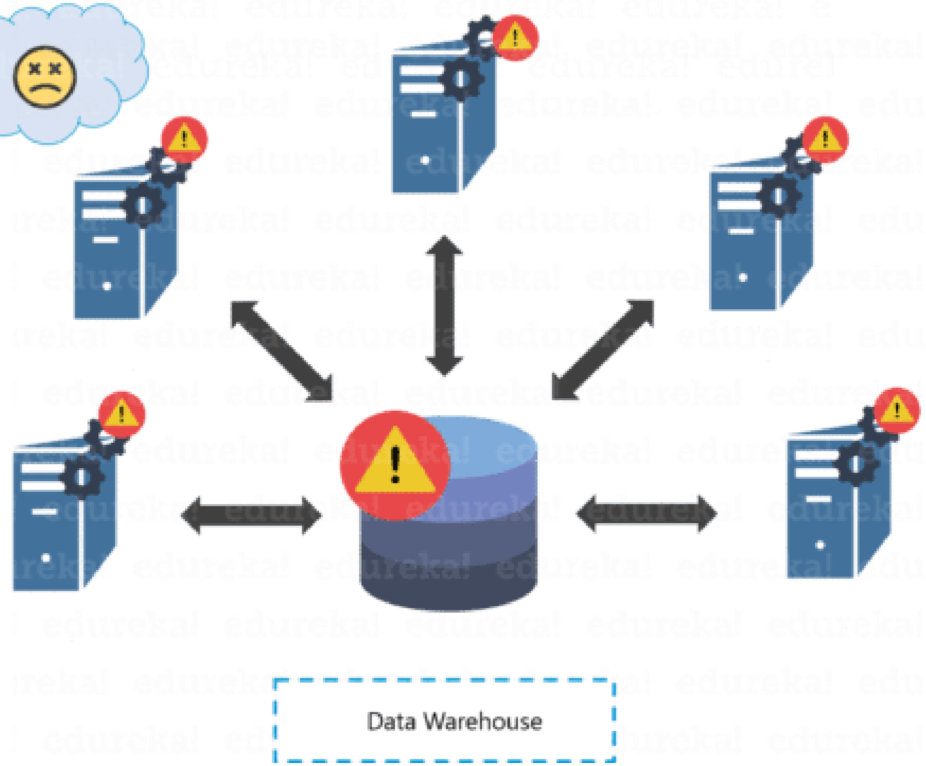

Scenario:

Multiple Cook cooking food

Issue:

Food Shelf becomes the BOTTLENECK

edureka!



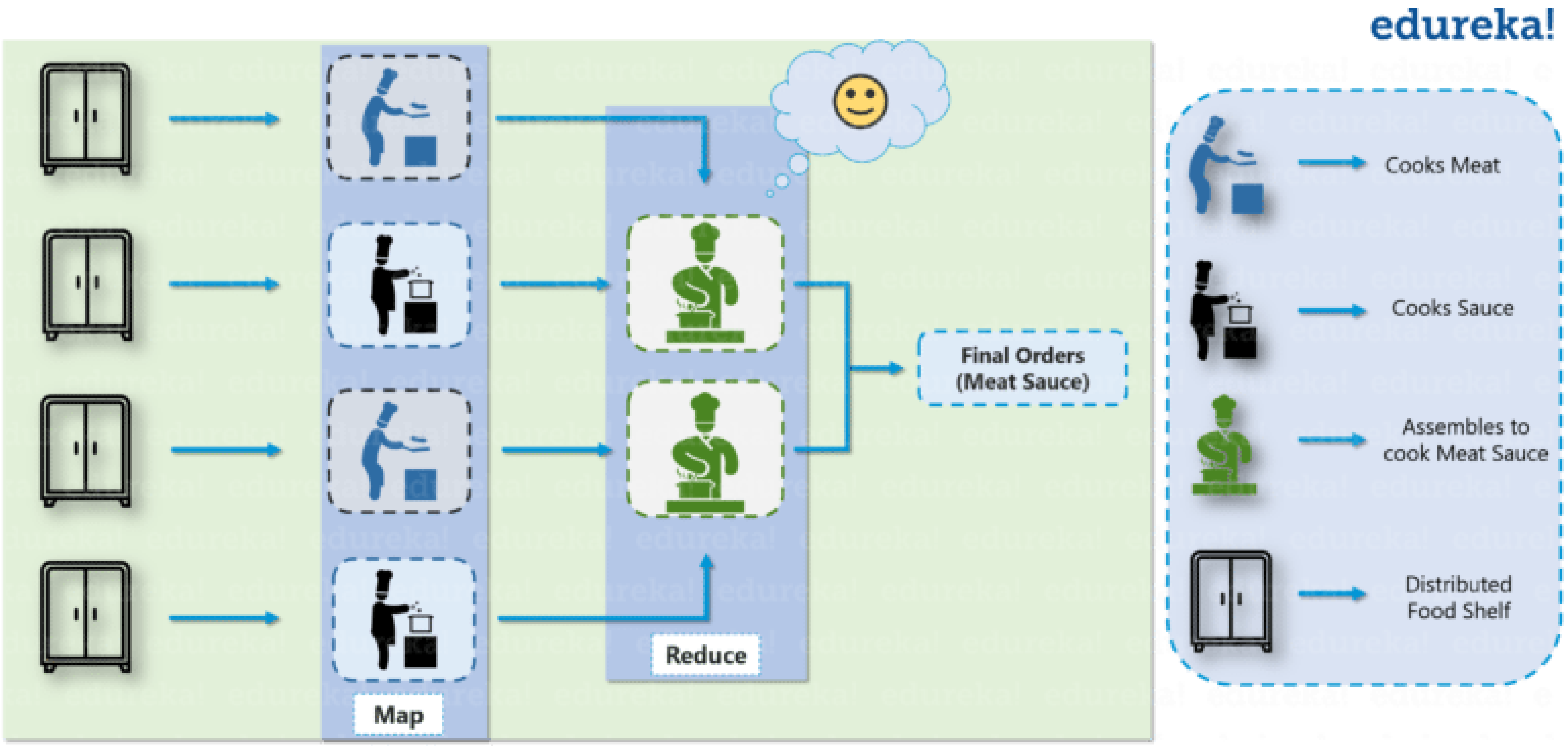

Scenario:

Multiple Processing Unit for data processing

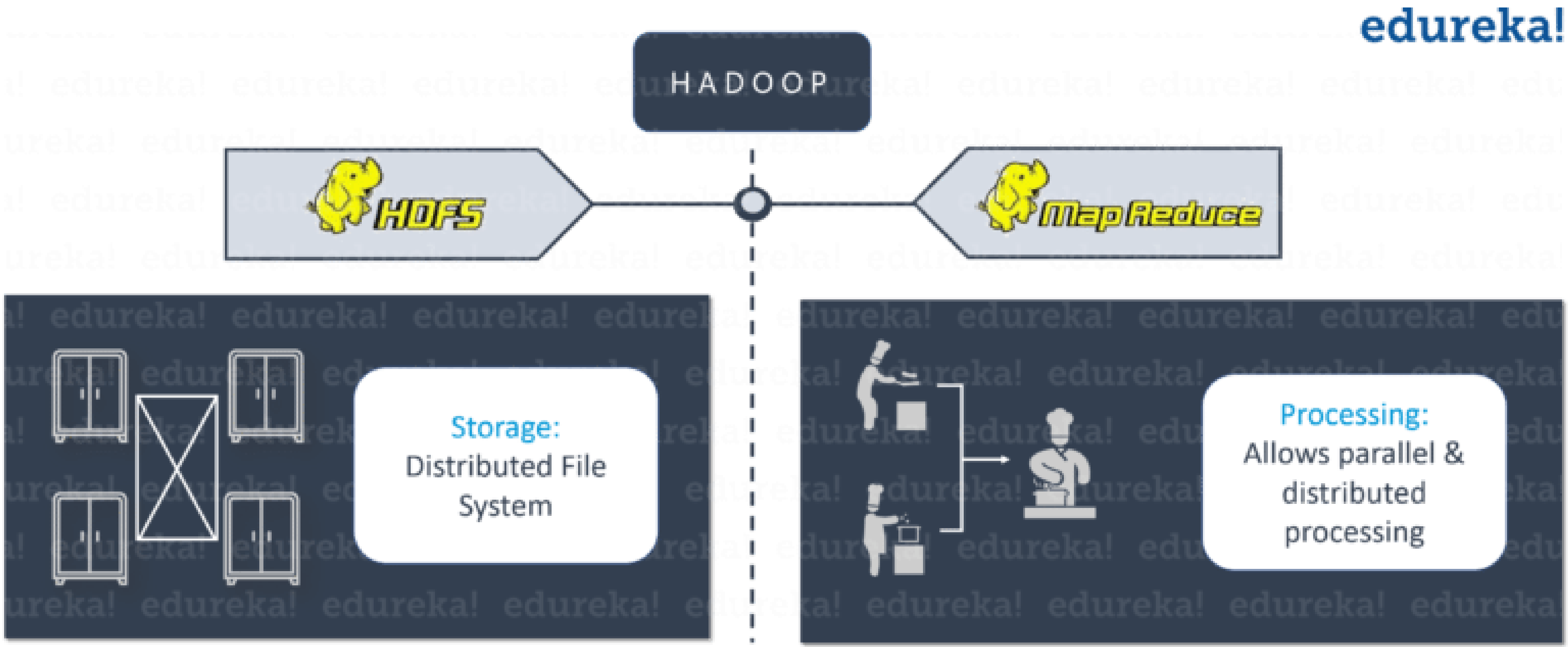
Issue:

Bringing data to processing generated lot of Network overhead

# Apply Map-Reduce Concept



# MapReduce and HDFS



# Map-Reduce Computational Model
## Classic Example
## Count word occurrences in a set of documents

# Programming Model: MapReduce

- Warm-up task:
  - We have a huge text document (e.g., 10 TB)
  - Count the number of times each distinct word appears in the file
- Sample application:
  - Analyse web server logs to find popular URLs
  - Term statistics for search engine

# Task: Word Count

**Scenario 1:**

- File is too large for RAM, but all <word, count> pairs fit in memory

**Scenario 2:**

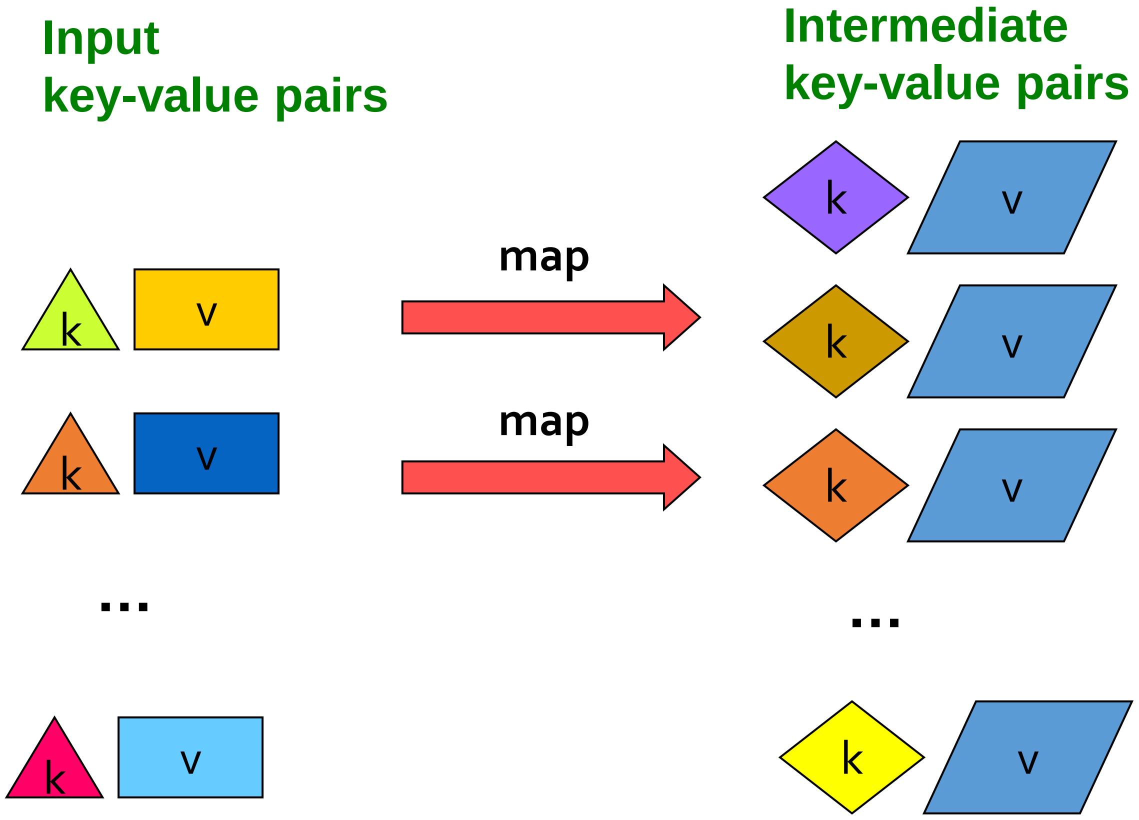
- So many distinct words, and <word, count> pairs do not fit in memory
- Divide a large file to small pieces
- Compute in parallel and generate multiple <word, count> pairs, and merge them
- Scenario 2 captures the essence of MapReduce

# MapReduce: Overview

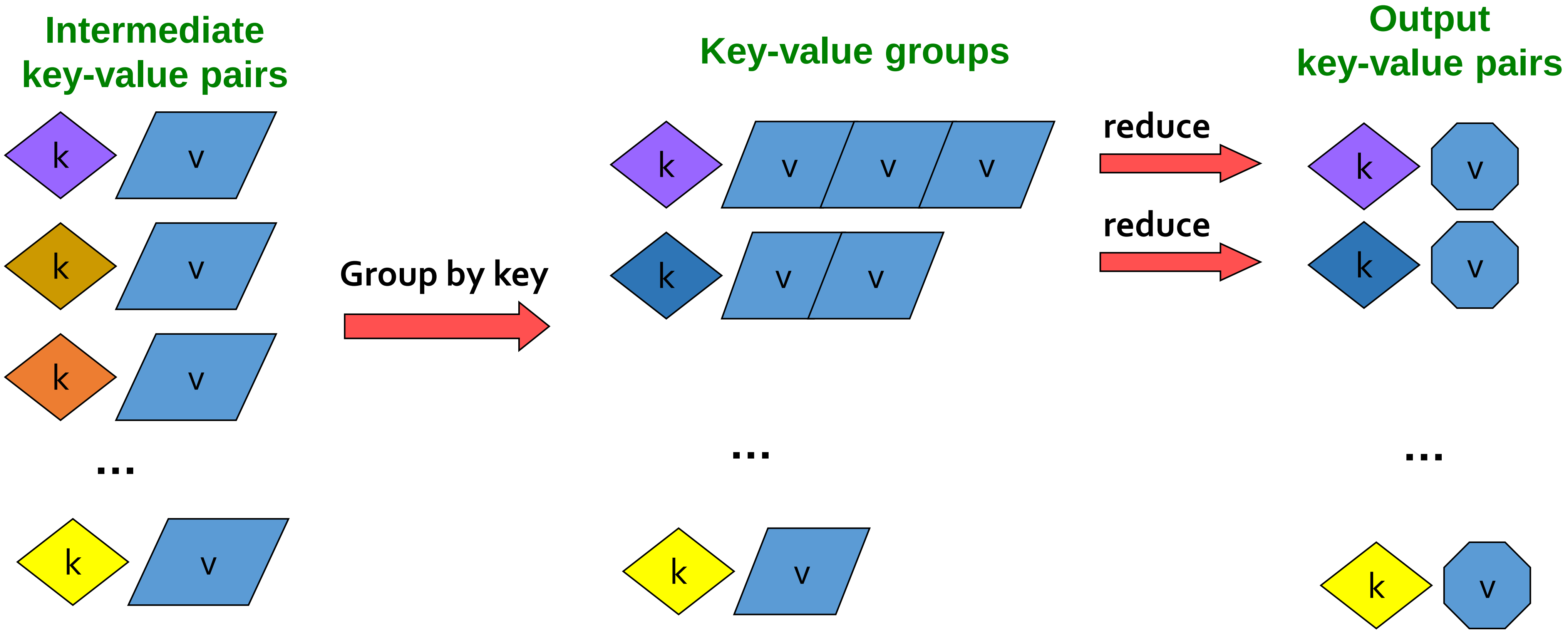
- **Map:**
  - Scan input file, e.g., doc.txt
  - Extract something that you care about
- **Group by key:**
  - Sort and Shuffle
- **Reduce:**
  - Aggregate, summarize, filter or transform
  - Write the result

Change Map and Reduce to fit the problem

# MapReduce: The Map Step



# MapReduce: The Reduce Step



# MapReduce: Word Counting

**Provided by the programmer**

**MAP:**
Read input and produces a set of key-value pairs

**Group by key:**
Collect all pairs with same key

**Provided by the programmer**

**Reduce:**
Collect all values belonging to the key and output

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ……………………..

**Big document**

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
….

**(key, value)**

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
…

**(key, value)**

(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
…

**(key, value)**

**Sequential read disk**

# Java Implementation of Word Count MapReduce

```
public class MapClass extends Mapper<LongWritable,
Text, Text, IntWritable> {

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

   @Override
   protected void map(LongWritable key, Text value,
Context context) throws IOException,
InterruptedException {

      String line = value.toString();
      StringTokenizer st = new StringTokenizer(line, " ");
      while (st.hasMoreTokens()) {
         word.set(st.nextToken());
         context.write(word, one);
      }
   }
}
```

```
public class ReduceClass extends Reducer<Text,
IntWritable, Text, IntWritable> {

@Override
 protected void reduce(Text key, Iterable<IntWritable>
values, Context context)  throws IOException,
InterruptedException {

      int sum = 0;
      Iterator<IntWritable> valuesIt = values.iterator();
      while (valuesIt.hasNext()) {
         sum = sum + valuesIt.next().get();
      }
      context.write(key, new IntWritable(sum));
   }
}
```

# MapReduce Example: Host Size

- Suppose we have a large web corpus with a metadata file formatted as follows:
  - Each record of the form: (URL, size, date, …)
- For each host, find out the total number of bytes

- Map
  - For each record, output (hostname (URL), size)
- Reduce
  - Sum the sizes of each host

# MapReduce Example: Language Model

- Count number of times each 5-word sequence occurs in a large corpus of documents

- Map
  - Extract (5-word sequence, count) from document

- Reduce
  - Combine the counts

# Hadoop

# What is Hadoop

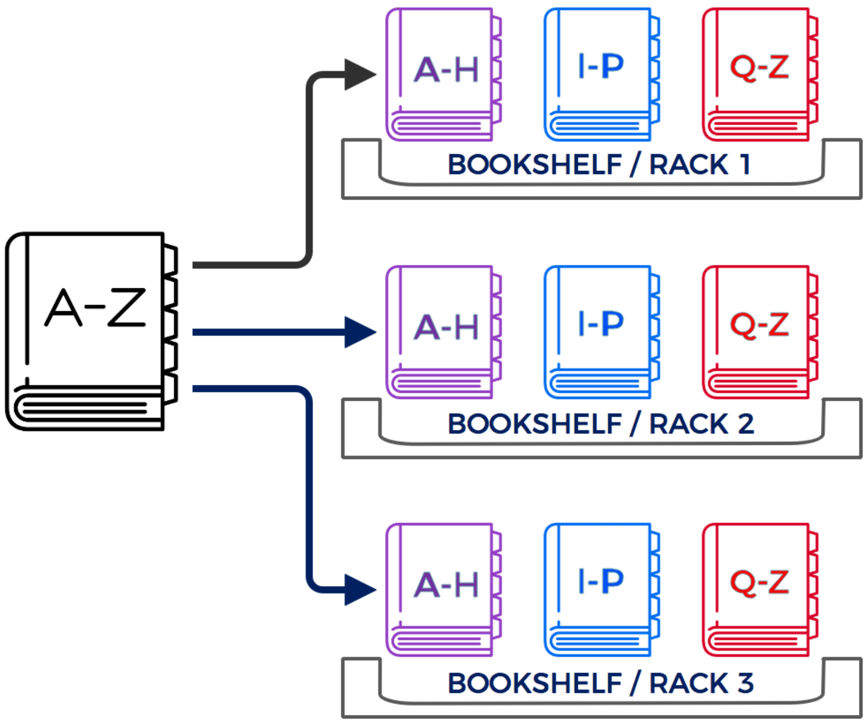
- The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.
  - Hadoop was developed, based on the paper written by Google on the MapReduce system and it applies concepts of functional programming.
  - Hadoop is written in the Java programming language and ranks among the highest-level Apache projects.
- The Hadoop Framework
  - Well-known in big data spaces.
  - Consist of multiple projects of Apache Software Foundation.
  - Support various types of datasets, e.g., structured and unstructured.

# Four Primary Components

- Hadoop Common
  - common utilities that support other modules.
- Hadoop Distributed File System (HDFS)
  - a distributed filesystem that provides high-throughput access to application data.
- Hadoop YARN
  - a framework for job scheduling and cluster resource management.
- Hadoop MapReduce
  - a programming model for parallel processing of large datasets.

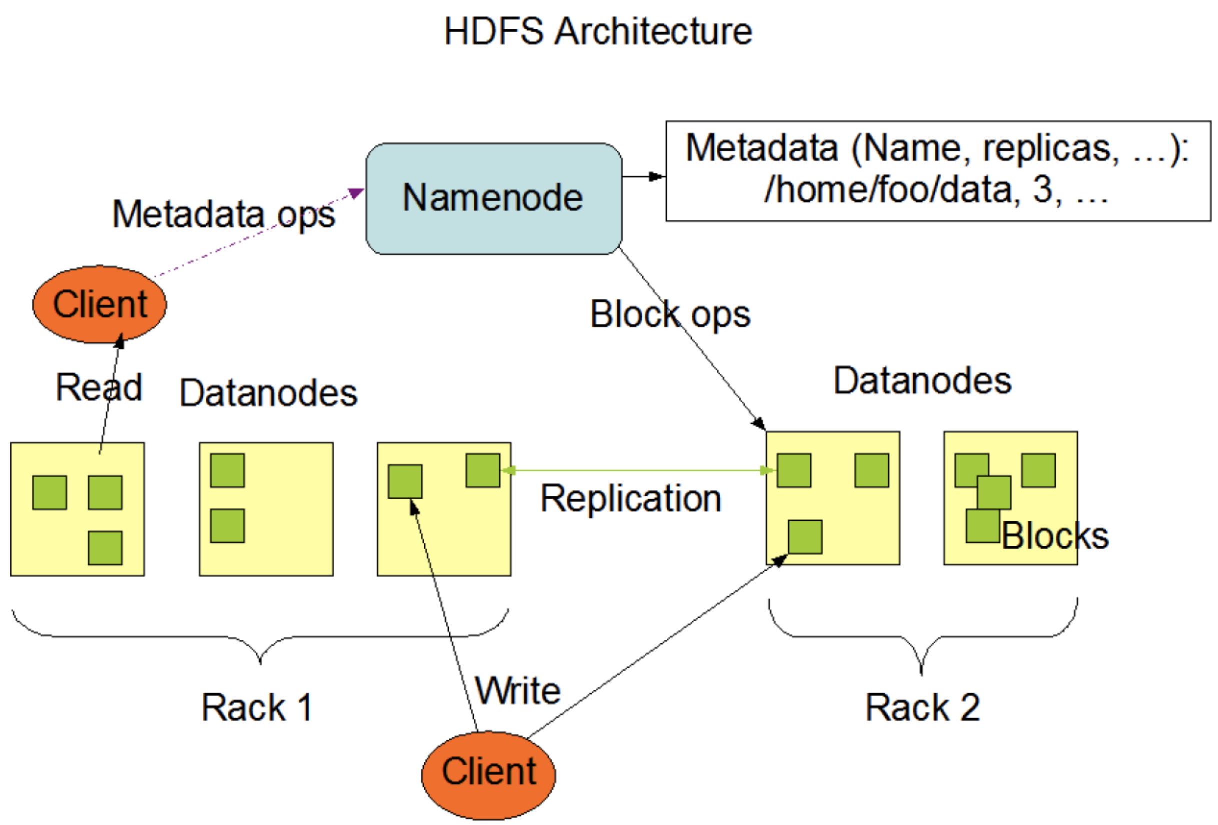
# Hadoop Distributed File System (HDFS)

- HDFS performs two main functions
  - Namespaces: Provides namespaces that hold cluster metadata, that is, the location of data in the Hadoop cluster
  - Data storage: Acts as storage for data used  in Hadoop cluster
- Bookshelf Analogy
  - Consider a large book that consists of Chapter A-Z. in HDFS, books have been split into smaller chunks.



# HDFS Backend

- NameNode
  - Consider as master node.
  - Contain cluster metadata and the location of data
  - Store the entire namespace in RAM
- DataNode
  - Individual servers, responsible for storing chunks of data
  - Perform compute operations



# Reference


- Rajaraman, Anand, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.
- Dasgupta, Nataraj. *Practical big data analytics: Hands-on techniques to implement enterprise analytics and machine learning using Hadoop, Spark, NoSQL and R*. Packt Publishing Ltd, 2018.
- Shubham Sinha, *Hadoop Tutorial - A Comprehensive Guide To Hadoop*, Oct 9 2014
- *edureka!*, https://www.edureka.co/
- Leskovec, Jure, Anand Rajaraman, and Jeff Ullman. "Big Data Management and Analytics." (2015).