DISTRIBUTED AND CLOUD COMPUTING

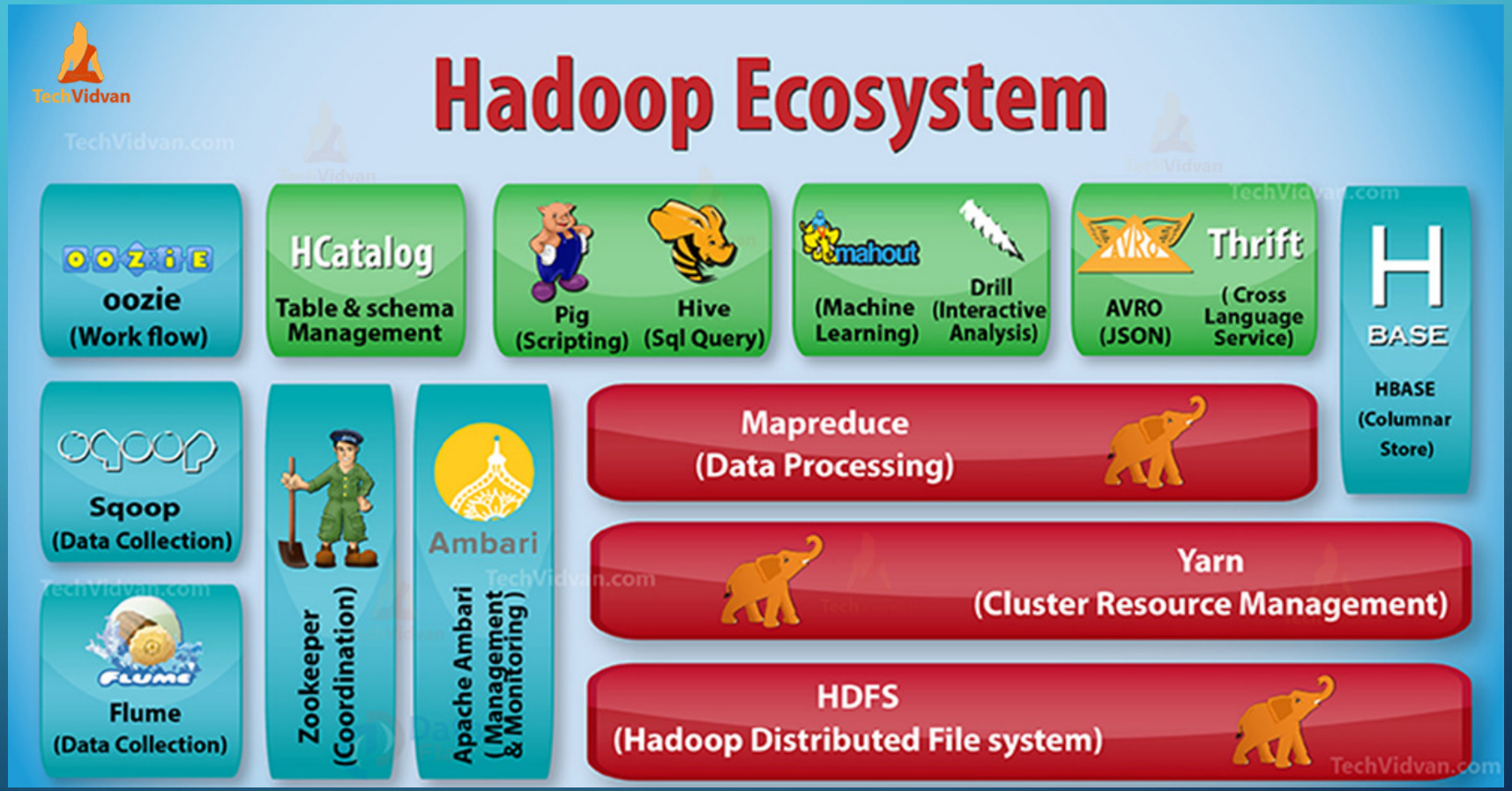
LAB7 HADOOP AND MAPREDUCE

APACHE HADOOP

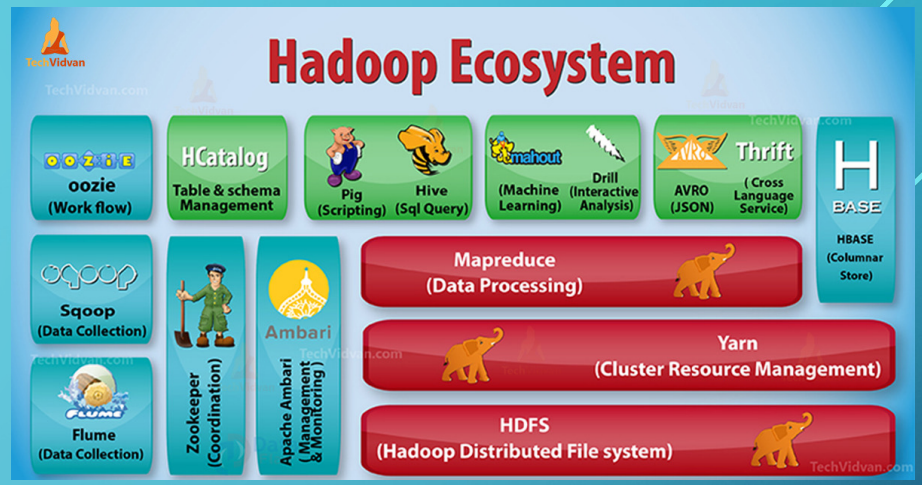
- The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.
- Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.
- It provides a distributed filesystem (HDFS) that can store data across thousands of servers, and a means of running work (Map/Reduce jobs) across those machines, running the work near the data.

https://cwiki.apache.org/confluence/display/HADOOP2/ProjectDescription

HADOOP ECOSYSTEM



HADOOP ECOSYSTEM



- MapReduce
  - Data processing using MapReduce paradigm
- HDFS (Hadoop Distributed File System)
  - High-throughput and fault-tolerant
  - Initially designed to be deployed on commodity hardware
- YARN
  - Resource management

AN EXAMPLE

- Data processing of a weather dataset
- The dataset records the temperature recorded between 1901-2001
- How to get the highest temperature each year?
  - Loop each line of the data?
  - Use Hadoop for parallel processing?

Dataset:

```
raw
|- 1901
|  |- 010010-99999-1901.gz
|  |- 010014-99999-1901.gz
|  |- 010015-99999-1901.gz
|  |- 010016-99999-1901.gz
|  |- 010017-99999-1901.gz
|  |- 010030-99999-1901.gz
|  |- 010040-99999-1901.gz
|  |- 010080-99999-1901.gz
|  |- 010100-99999-1901.gz
|- 1902
|- 1903
|- 1904
|- 1905
|- 1906
|- 1907
|- 1908
|- 1909
|- 1910
.
.
.
|- 2001
```

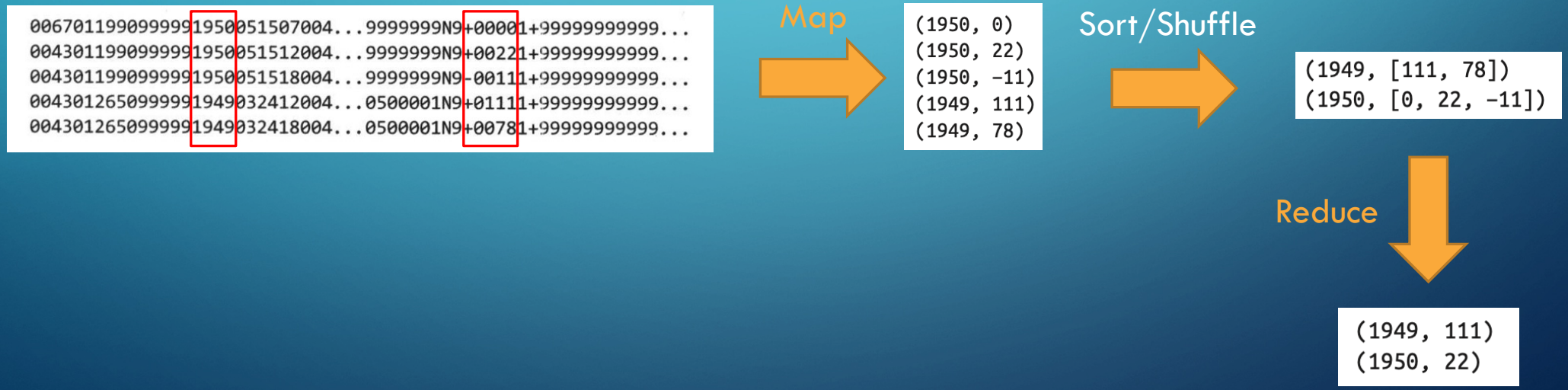
```
% ls raw/1990 | head
010010-99999-1990.gz
010014-99999-1990.gz
010015-99999-1990.gz
010016-99999-1990.gz
010017-99999-1990.gz
010030-99999-1990.gz
010040-99999-1990.gz
010080-99999-1990.gz
010100-99999-1990.gz
```

Sample lines of data (unused columns are indicated by ellipses):

```
0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
0043012650999991949032418004...0500001N9+00781+99999999999...
```

AN EXAMPLE

- The Map function: extract the year and the air temperature
- The Reduce function pick up the maximum reading of each year

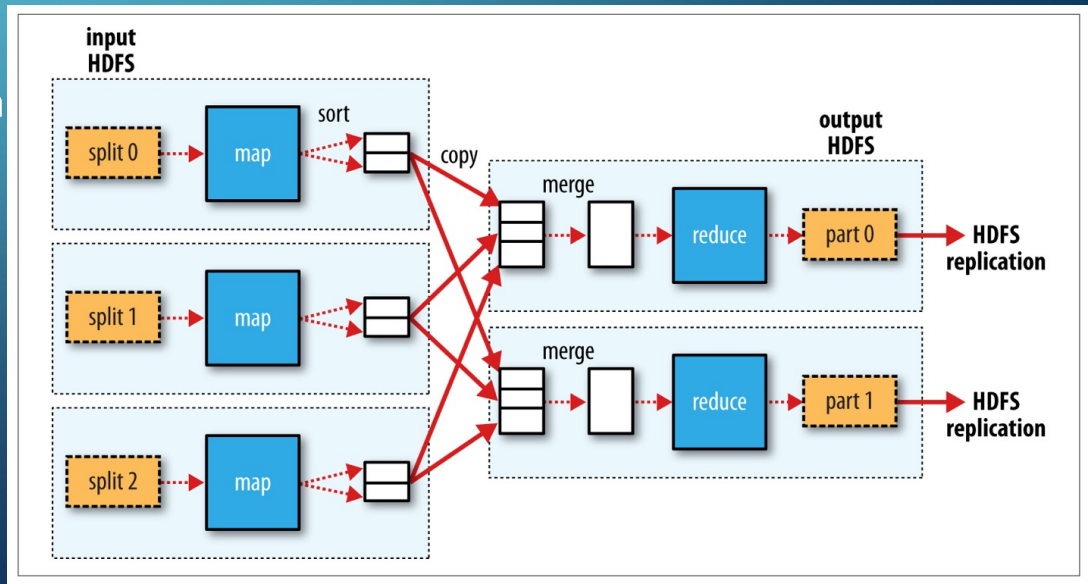


MAPREDUCE AS A PROGRAMMING PARADIGM

- The philosophy:

  "Moving Computation is Cheaper than **Moving Data"**

- The MapReduce paradigm only contains two steps: Map and Reduce
  - Data structure: Key-Value pair
  - Map: Split input to smaller blocks, apply operation
  - Reduce: Group all results from Map by key

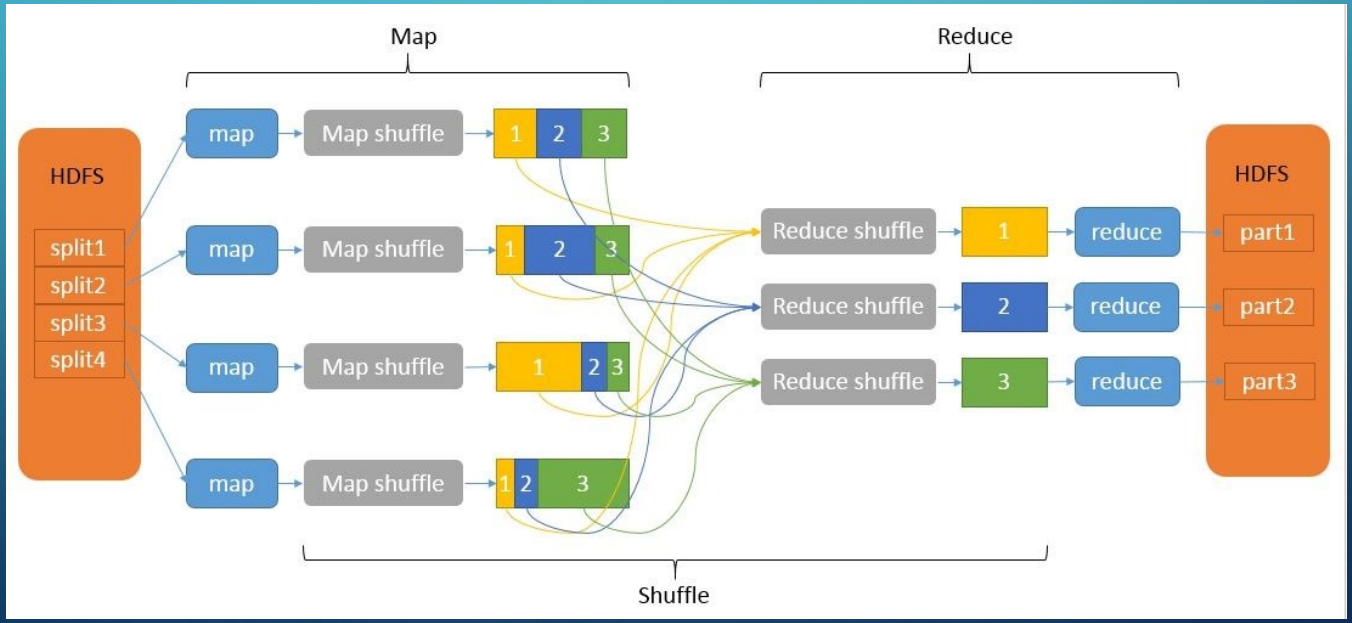


MAPREDUCE IN HADOOP

- In Hadoop, MR is capable to process vast amount of data (at least TBs) in parallel on large clusters (1000+ nodes)
- Reliable and fault-tolerant
- Computation in Hadoop MR is organized as *jobs*, framework will schedule/monitor/restart each individual tasks in the job.

MAPREDUCE IN HADOOP

- In the implementation, we have one more stage: shuffle/sort
- Why? Because Map and Reduce are iterative operations, so sorted data could have better performance



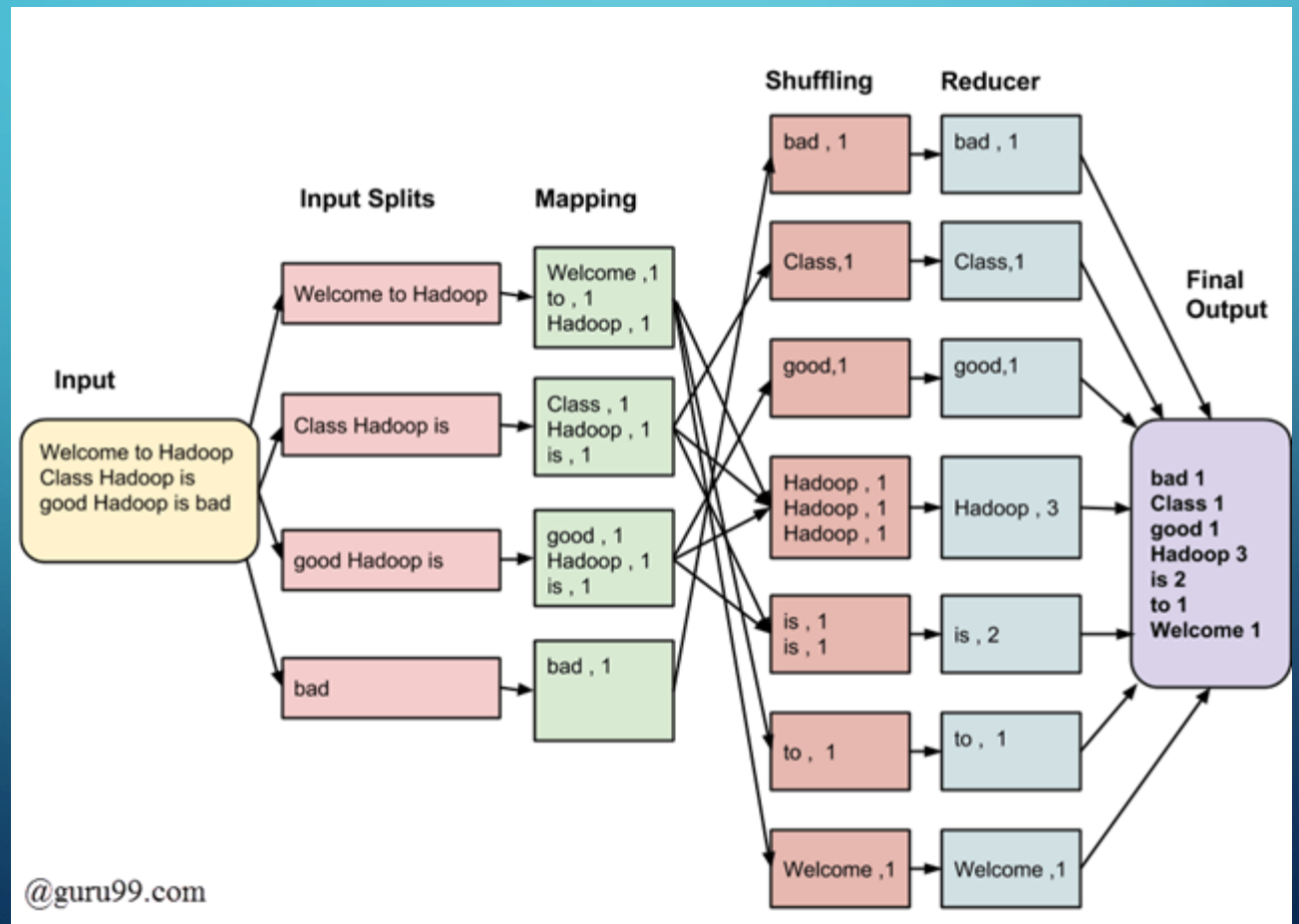
MAPREDUCE IN HADOOP

- API for Hadoop MapReduce
  - Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
  - Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT>

- Basically, you need to extend these template classes and override the methods

MAPREDUCE IN HADOOP

A word count example



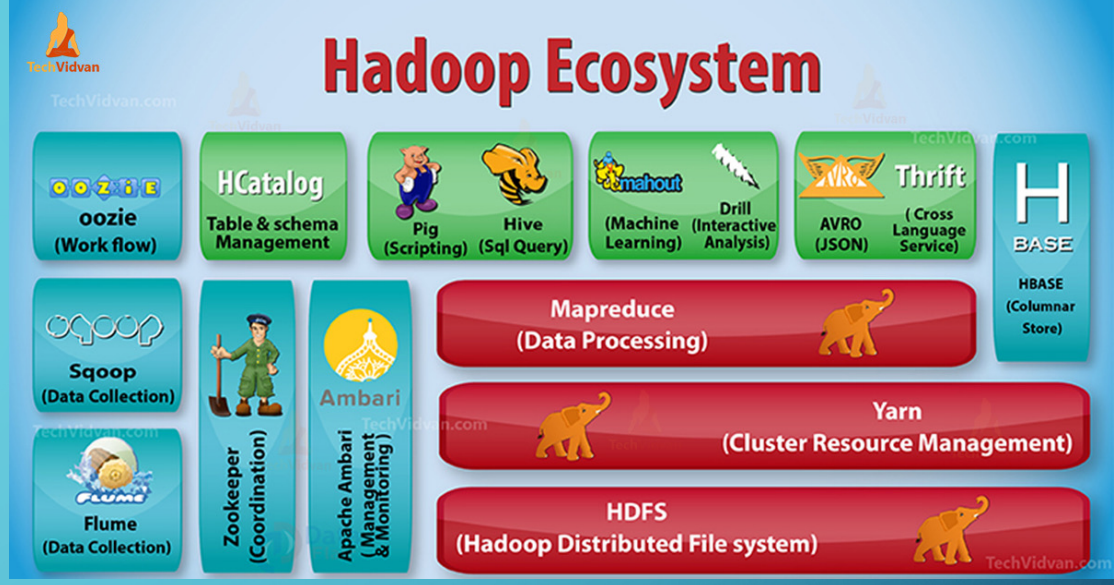
MAPREDUCE IN HADOOP

```
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);
  }
}
```

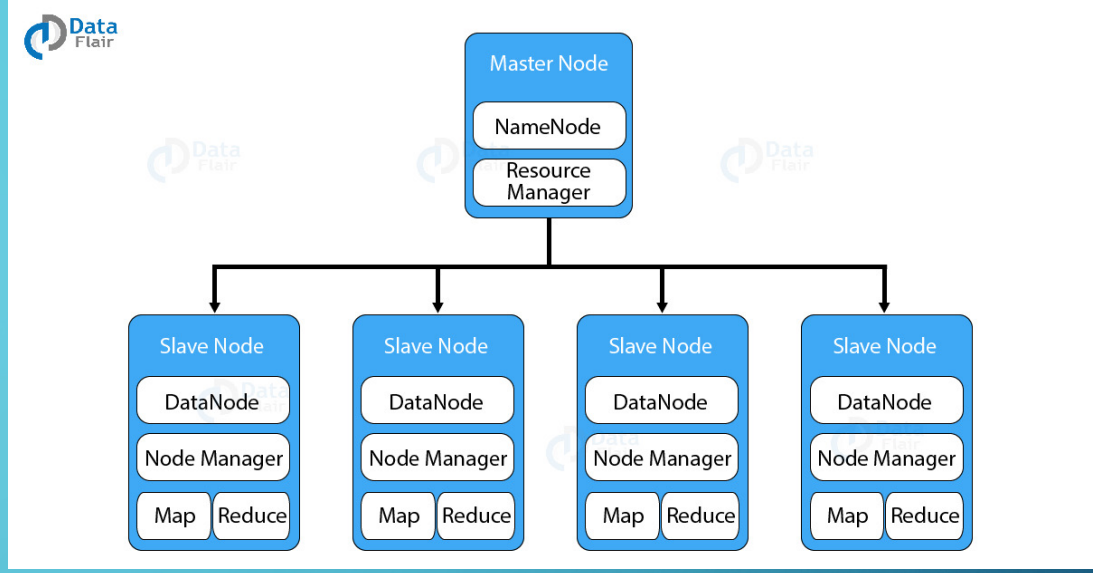
```
< Hello, 1>
< World, 1>
< Bye, 1>
< World, 1>
```

HADOOP ECOSYSTEM



- MapReduce
  - Data processing using MapReduce paradigm
- HDFS (Hadoop Distributed File System)
  - High-throughput and fault-tolerant
  - Initially designed to be deployed on commodity hardware
- YARN
  - Resource management

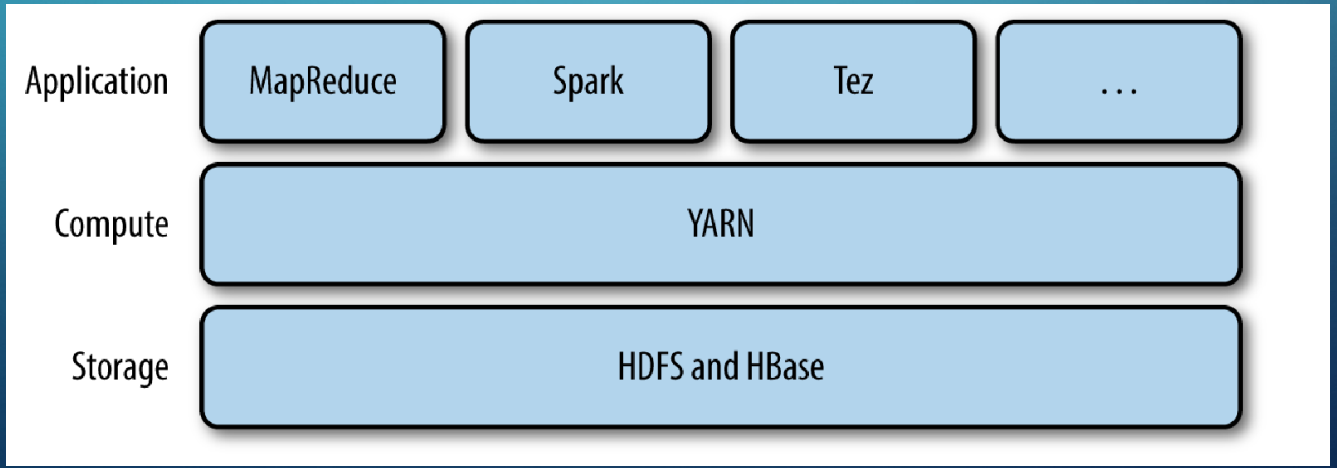
HDFS ARCHITECTURE



- Store each file as a sequence of blocks
- Blocks belonging to a file are replicated for fault tolerance
- Master-Slave architecture
  - One NameNode (software): a master server.
    - Manages the filesystem namespace
    - Regulates access to files by clients
    - Executes operations like opening, closing, and renaming files and directories
    - Determines the mapping of blocks to DataNodes
  - A number of DataNodes (software): one per node in the cluster.
    - Serve read and write requests from the client
    - Perform block creation, deletion, and replication upon instruction from the NameNode
- A file is split into one or more blocks, and these blocks are stored in a set of DataNodes

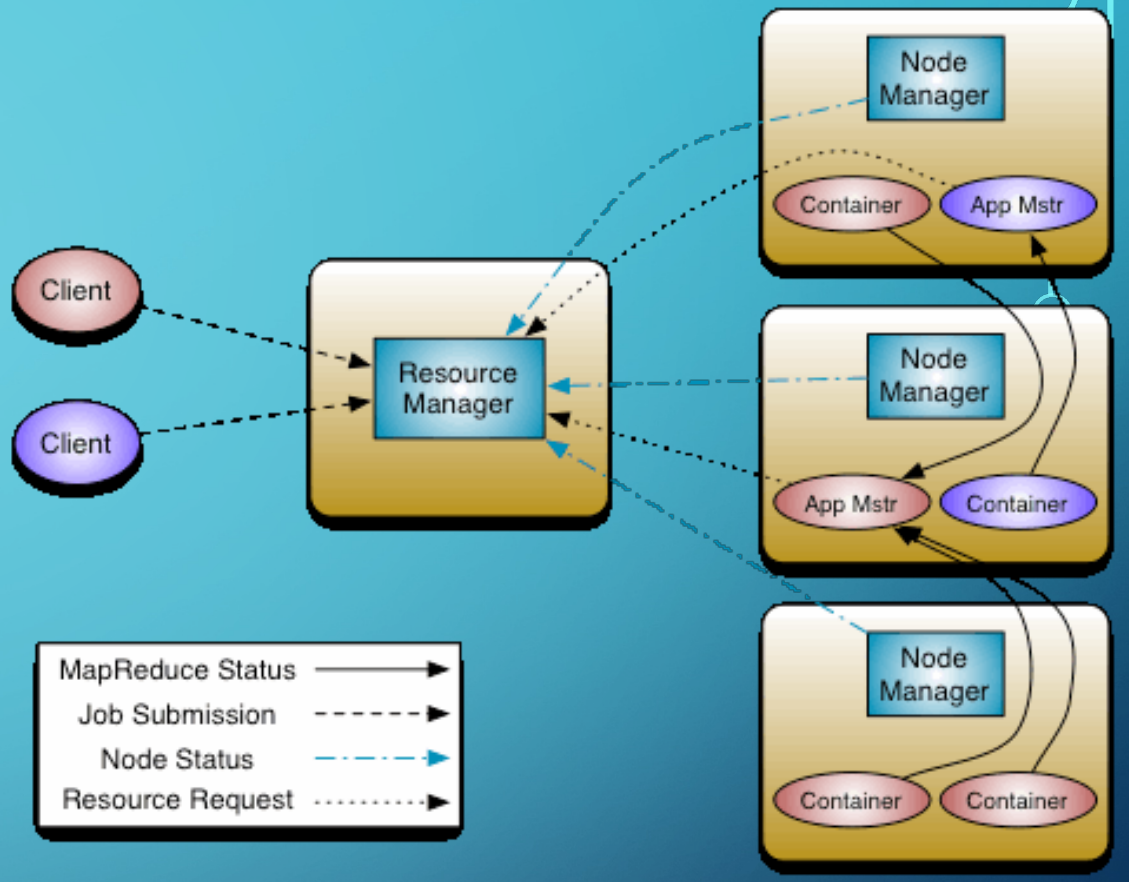
YARN ARCHITECTURE

- The role of YARN (Yet Another Resource Negotiator): Resource management and job scheduling/monitoring
- Users write to higher-level APIs provided by distributed computing frameworks, which themselves are built on YARN and hide the resource management details from the user.
- As shown in the figure, some distributed computing frameworks (MapReduce, Spark, and so on) run as *YARN applications* on the cluster compute layer (YARN) and the cluster storage layer (HDFS and HBase).



YARN ARCHITECTURE

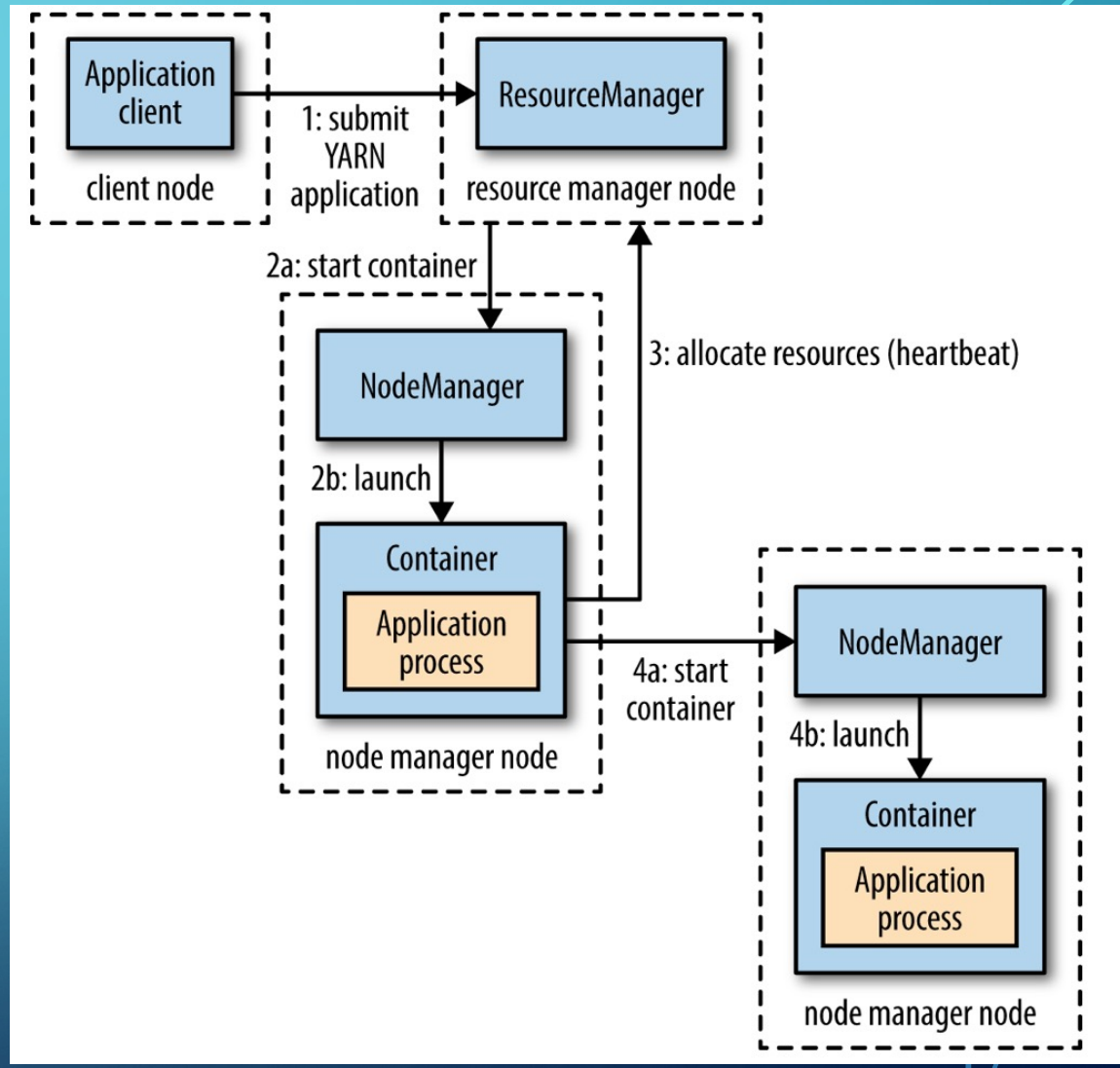
- Components
  - ResourceManager: One per cluster
    - The central controller, allocate resources, submitting & managing the jobs
  - NodeManager: the per-machine framework agent
    - Monitors containers
      - A container executes an application-specific process with a constrained set of resources (CPU, memory, disk, network).
      - A container may be a Unix process or a Linux cgroup
    - Reports the resource usage to the ResourceManager/Scheduler.



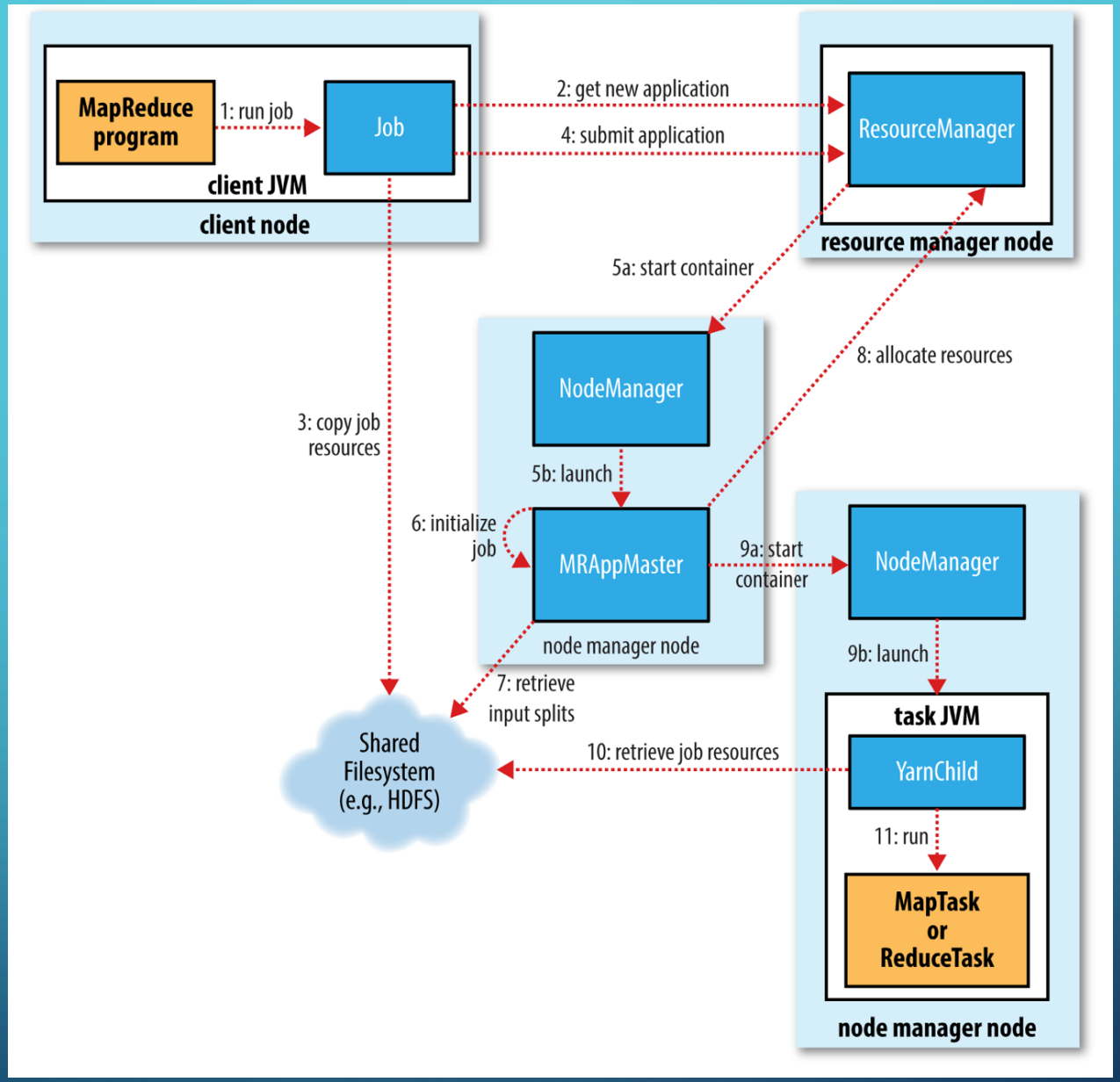
https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html

YARN ARCHITECTURE

- 1. A client asks the *ResourceManager* to run an *application master* process
- 2. *ResourceManager* finds a *NodeManager* that can launch the *application master* in a *container*
- 3. The *application master* *may request more containers from the ResourceManager*
- 4. The *application master* uses the newly requested containers to run a distributed computation



How YARN runs an application



How Hadoop runs a MapReduce job
(from *Hadoop: The Definitive Guide. 4th edition*)

DEPLOYING HADOOP

- Standalone (Local) mode: For debugging
- Pseudo-Distributed mode: All nodes run on the same machine
- Fully distributed mode: Runs on a cluster

DOWNLOAD A RELEASE

- Use version 3.2.4
- Link: https://hadoop.apache.org/releases.html

- Also, don't forget to install JDK
- Extract the archive to /opt/hadoop-3.2.4

Deploying Hadoop on WSL 2.0

- Remember to set `JAVA_HOME` in `hadoop-env.sh`
- When closing firewall - if not `iptables`, disable `ufw`
- When `ssh`, if connection refused, check with:

```
service ssh status
sudo service ssh start
```

- Adds as suggested here: https://segmentfault.com/a/1190000038473734

```
HDFS_DATANODE_USER=root
HDFS_DATANODE_SECURE_USER=hdfs
HDFS_NAMENODE_USER=root
HDFS_SECONDARYNAMENODE_USER=root
```

if `starting-dfs.sh` fails.

PRACTICE

Try deploying Hadoop in pseudo-distributed mode on your local VM

- Tutorial: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html
- Other references: https://segmentfault.com/a/1190000038473734

Complete the WordCount example

- Official tutorial: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

FURTHER READING

- Hadoop: The Definitive Guide. 4th edition. (*Available on Blackboard*)
- https://research.google/pubs/pub36249/ MapReduce: The Programming Model and Practice (*Available on Blackboard*)
- https://data-flair.training/blogs/hadoop-mapreduce-tutorial/