Distributed Data Processing

UA.DETI.CBD
José Luis Oliveira / Carlos Costa

Introduction

- Large-Scale Data Processing
  - aim to use 1000s of CPUs, but with simplified managing
- MapReduce
- Apache Hadoop

- To start:
  - https://www.youtube.com/watch?v=9s-vSeWej1U

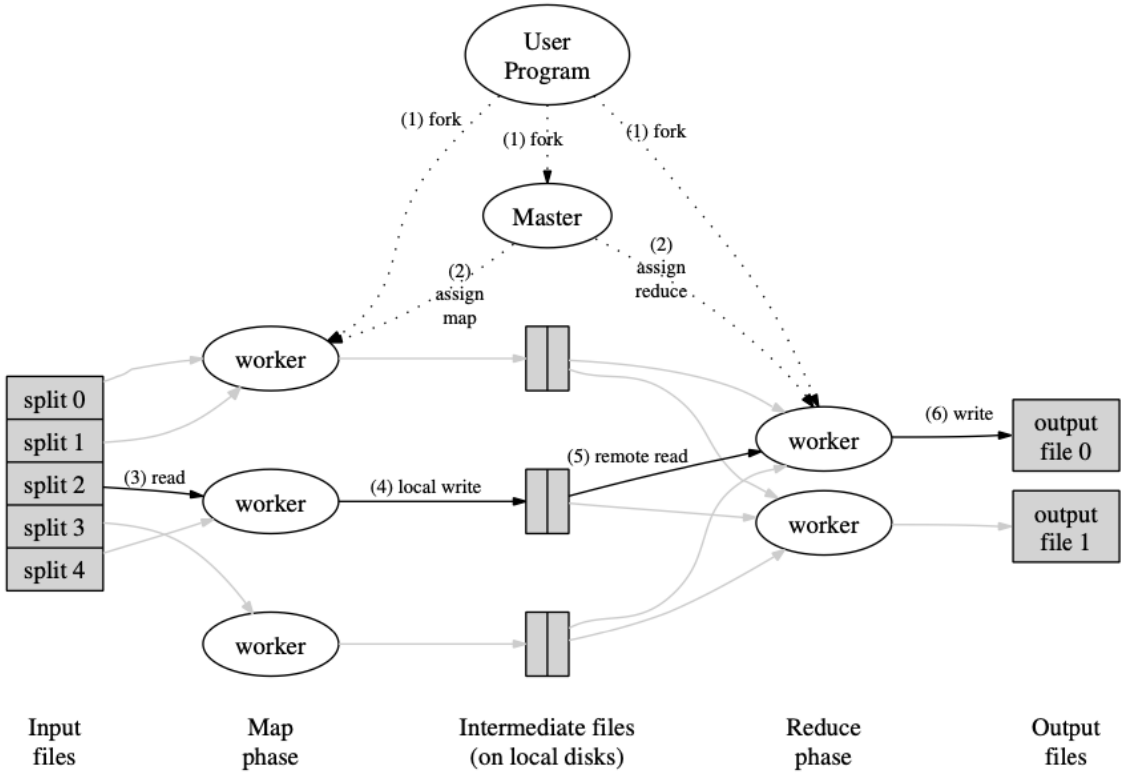
Context

- Big Data
- Distributed
  - Not possible to store it in a single place
- Transmission
  - High Latency

"Moving Computation is Cheaper than Moving Data"

What is MapReduce?

- "MapReduce is a programming model and an associated implementation for processing and generating large data sets"
  - Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: Simplified data processing on large clusters." (2004). [Google]



What is MapReduce?

- Terms are borrowed from Functional Language (e.g., Lisp)
- Example: Sum of squares
  - **map** square '(1 2 3 4))

    Output: (1 4 9 16)
    - *processes each record sequentially and independently*
  - **reduce** + '(1 4 9 16))

    (+ 16 (+ 9 (+ 4 1)))

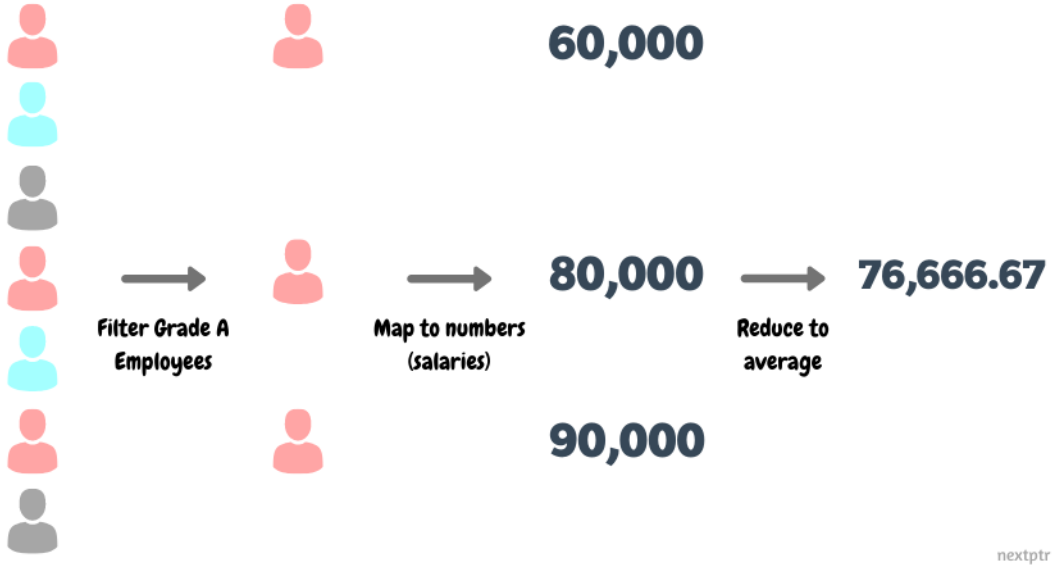
    Output: 30
    - *processes set of all records in batches*

- This concept has been reused in several programming languages and computational tools

Examples: Java stream

```
List<String> values =  Arrays.asList("1","2","3","4","5","6","7");
int soma = values.stream
                .map(num->Integer.parseInt(num))
                .reduce(0, Integer::sum);
```

```
OptionalDouble avgSalary = employees.stream()
                .filter((e) -> e.grade == 'A')      // filter 'A' grade employees
                .mapToInt((e) -> e.salary)          // get IntStream of salaries
                .average();                         // get average
```



Examples: MongoDB

Database:

```
db.orders.insertMany([
   { _id: 1, cust_id: "Ant O. Knee", ord_date: new Date("2020-03-01"), price: 25,
   items: [ { sku: "oranges", qty: 5, price: 2.5 }, { sku: "apples", qty: 5, price: 2.5 }
], status: "A" },
   { _id: 2, cust_id: "Ant O. Knee", ord_date: new Date("2020-03-08"), price: 70, items: [
{ sku: "oranges", qty: 8, price: 2.5 }, { sku: "chocolates", qty: 5, price: 10 } ], status:
"A" },
   ...
])
```

Return the Total Price Per Customer:

```
var mapFunction1 = function() {
   emit(this.cust_id, this.price);
};
var reduceFunction1 = function(keyCustId, valuesPrices) {
   return Array.sum(valuesPrices);
};
db.orders.mapReduce(
   mapFunction1,
   reduceFunction1,
   { out: "map_reduce_example" }
)
db.map_reduce_example.find().sort( { _id: 1 } )
```

```
{ "_id" : "Ant O. Knee", "value" : 95 }
{ "_id" : "Busby Bee", "value" : 125 }
{ "_id" : "Cam Elot", "value" : 60 }
...
```

https://docs.mongodb.com/manual/tutorial/map-reduce-examples/

MapReduce frameworks

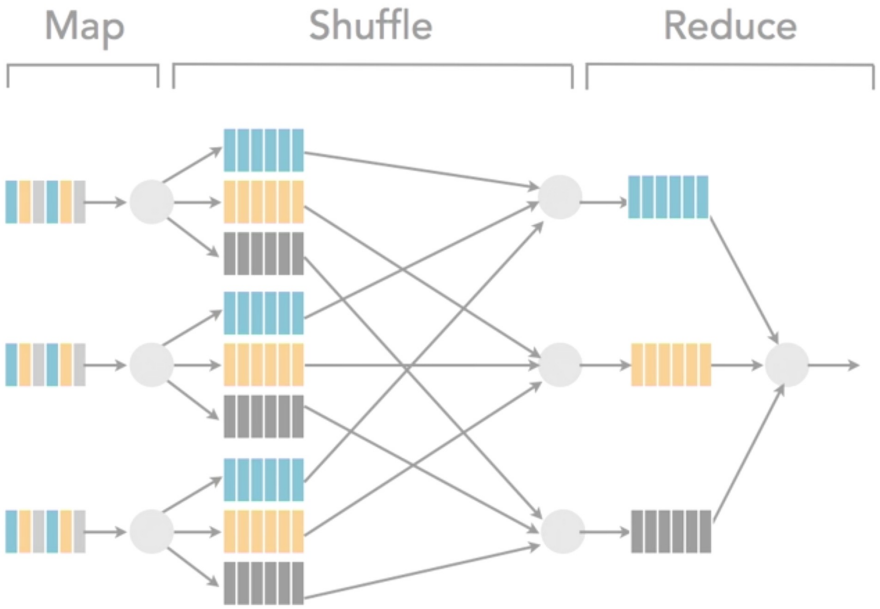
- Library/Tools that allow easily writing applications that process large amounts of data in parallel
  - distributed across several nodes
- Good retry/failure semantics
- Solution Pattern
  - many problems can be modelled in this way
- Implementations
  - **Hadoop**: the mapper and reducer are each a Java class that implements a particular interface.
  - **Spark**: newer and faster, it processes data in RAM using a concept known as an RDD, Resilient Distributed Dataset.

MapReduce frameworks

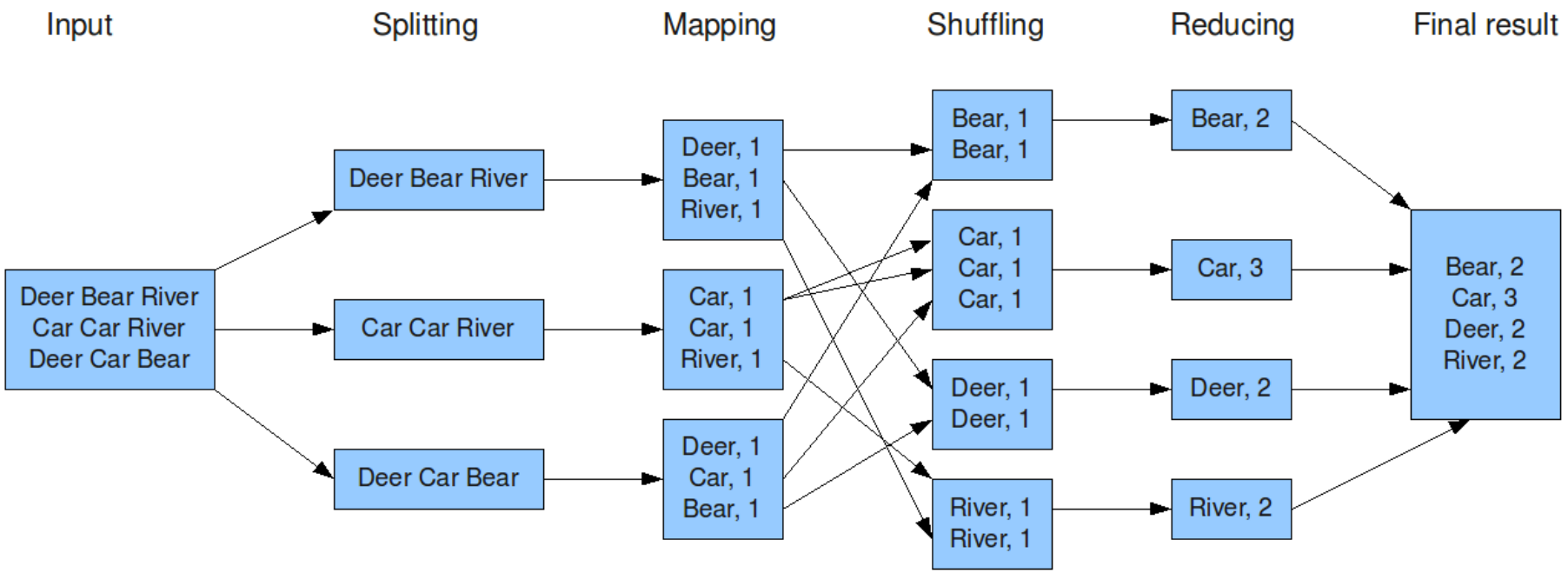
- **Framework**…
  - Automatic parallelization & distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring & status updates
- Two main tasks:
  - Mappers & Reducers
- Pipeline
  - Splits the input data-set into independent chunks
  - Mappers process of chunks in a parallel manner
  - aggregates the outputs of the maps
  - Reducers process the mappers output

MapReduce dataflow

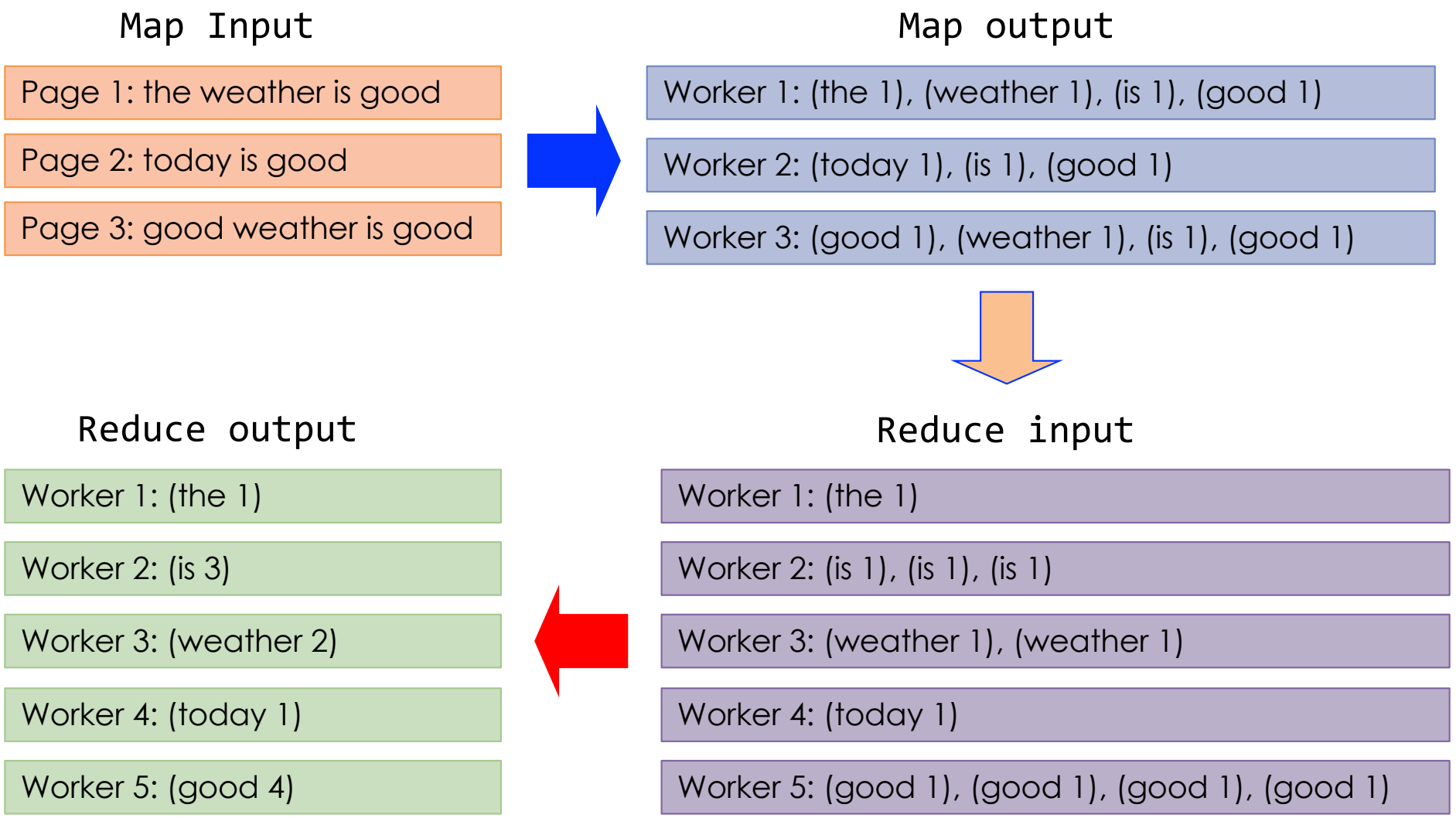
- The **map** function
  - is called once for every input record to extract (one or more) key-value from the input record
- MapReduce framework
  - collects all the key-value pairs with the same key – **shuffle**
- The **reduce** function
  - iterates over each collection of values with the same key and can produce output records
    - e.g., the number of occurrences of the key



Example: Words count dataflow



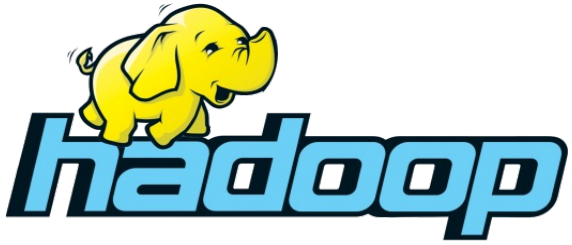
Words count – tasks execution



MapReduce applications

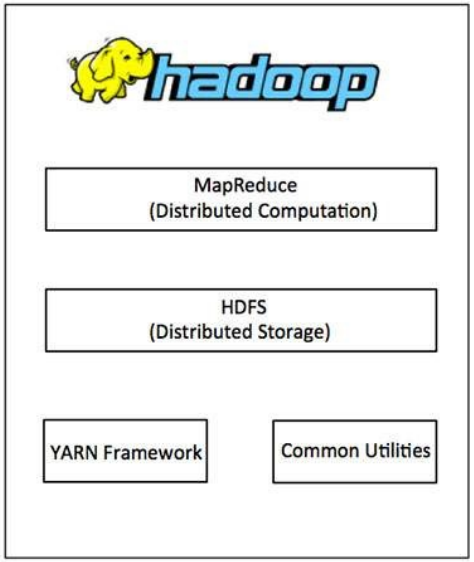
- words count / histogram
- distributed search
- distributed sort
- web link-graph reversal
- term-vector per host
- web access log stats
- inverted index construction
- document clustering
- machine learning
- statistical machine translation
- …

Apache Hadoop

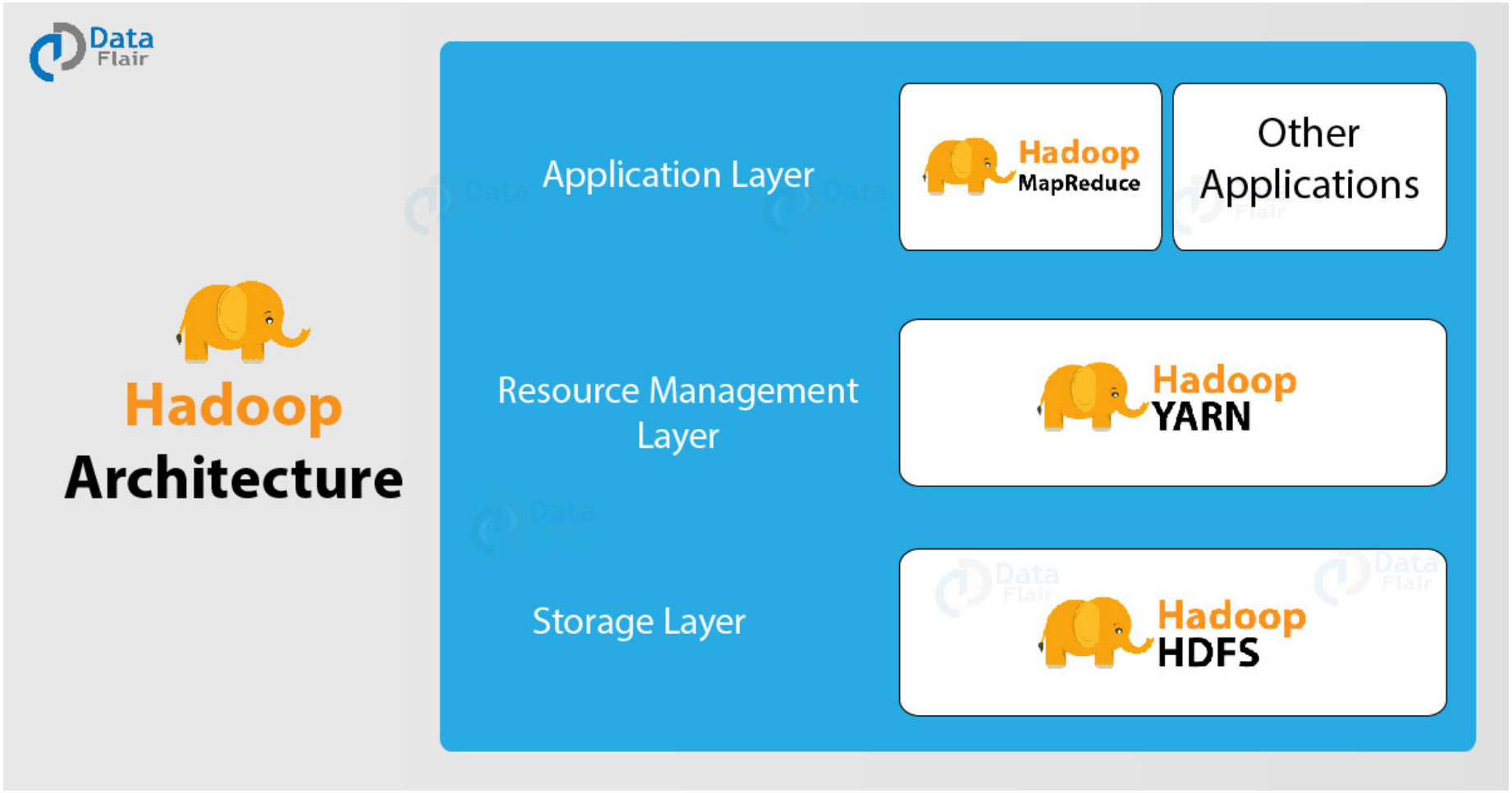


Apache Hadoop Framework

- A framework that allows distributed processing of **large data sets across clusters of computers** using simple programming models.
- Open-source framework
  - Java
  - https://hadoop.apache.org
- **Main components**
  - Hadoop Distributed File System (HDFS)
    - Distributed, scalable, and portable file system
  - Hadoop Yet Another Resource Negotiator (YARN)
  - Hadoop Common Utilities
  - Hadoop MapReduce
    - Implementation of the MapReduce programming model



Apache Hadoop Framework



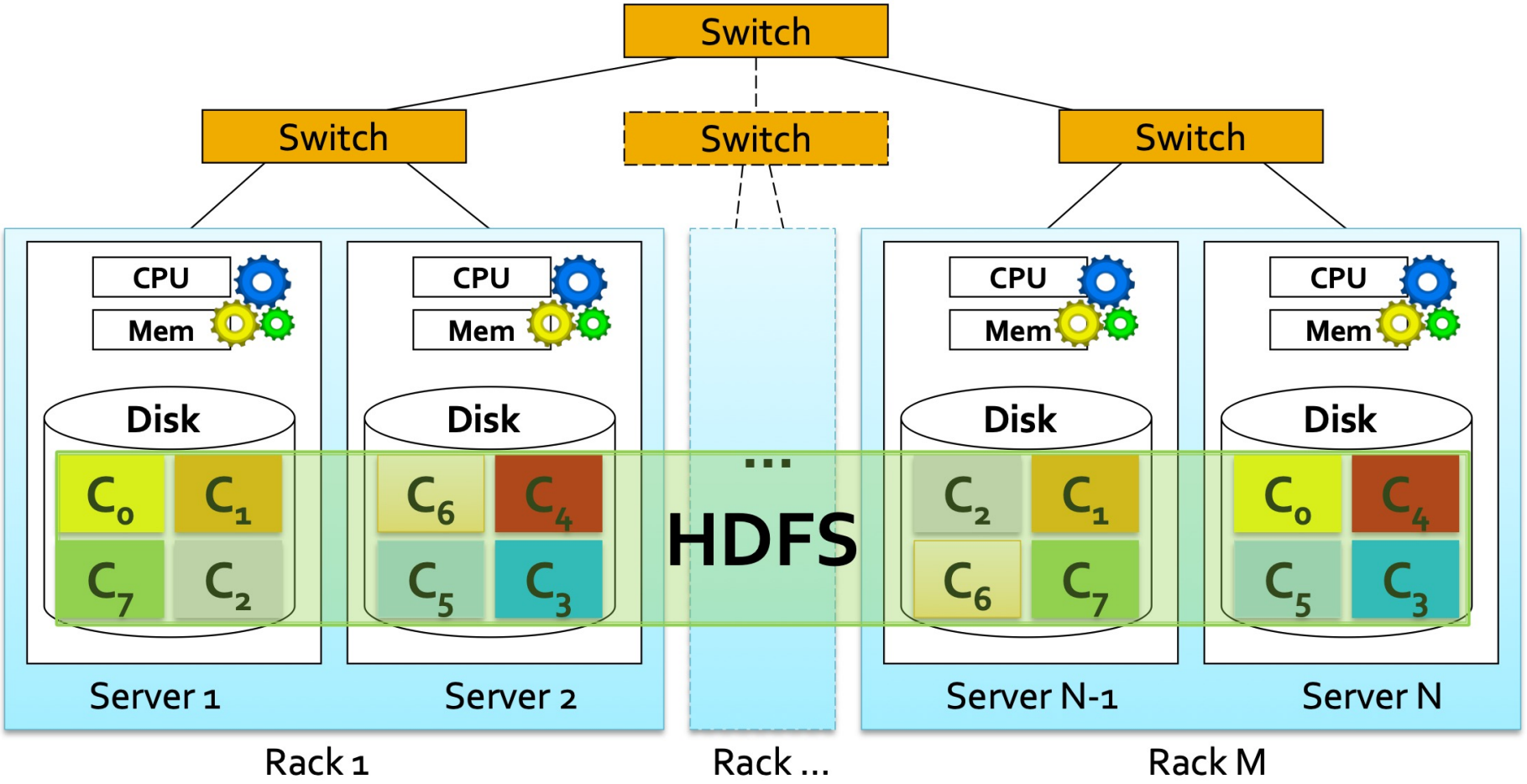
Hadoop Distributed File System (HDFS)

- Distributed file system designed to run on commodity hardware.
- Stores data redundantly on multiple nodes – Fault-tolerant file system
  - 3+ replicas for each block
  - Default Block Size : 128MB
- Master-Slave architecture
  - NameNode: Master
  - DataNode: Slave
- Typical usage pattern
  - Huge files (GB to TB)
  - Data is rarely updated
  - Reads and appends are common
    - Usually, random read/write operations are not performed

HDFS – Assumptions and Goals

- **Hardware Failure**
  - An HDFS instance may consist of +100/+1000 of server machines.
- **Streaming Data Access**
  - HDFS is designed more for batch processing rather than interactive use. The emphasis is on high throughput of data access rather than low latency of data access.
- **Large Data Sets**
  - Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size.
- **Simple Coherency Model**
  - HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed except for appends and truncates.
- **"Moving Computation is Cheaper than Moving Data"**
  - Especially true when the size of the data set is huge. Minimizes network congestion and increases the throughput of the system.
- **Portability Across Heterogeneous Platforms**

HDFS

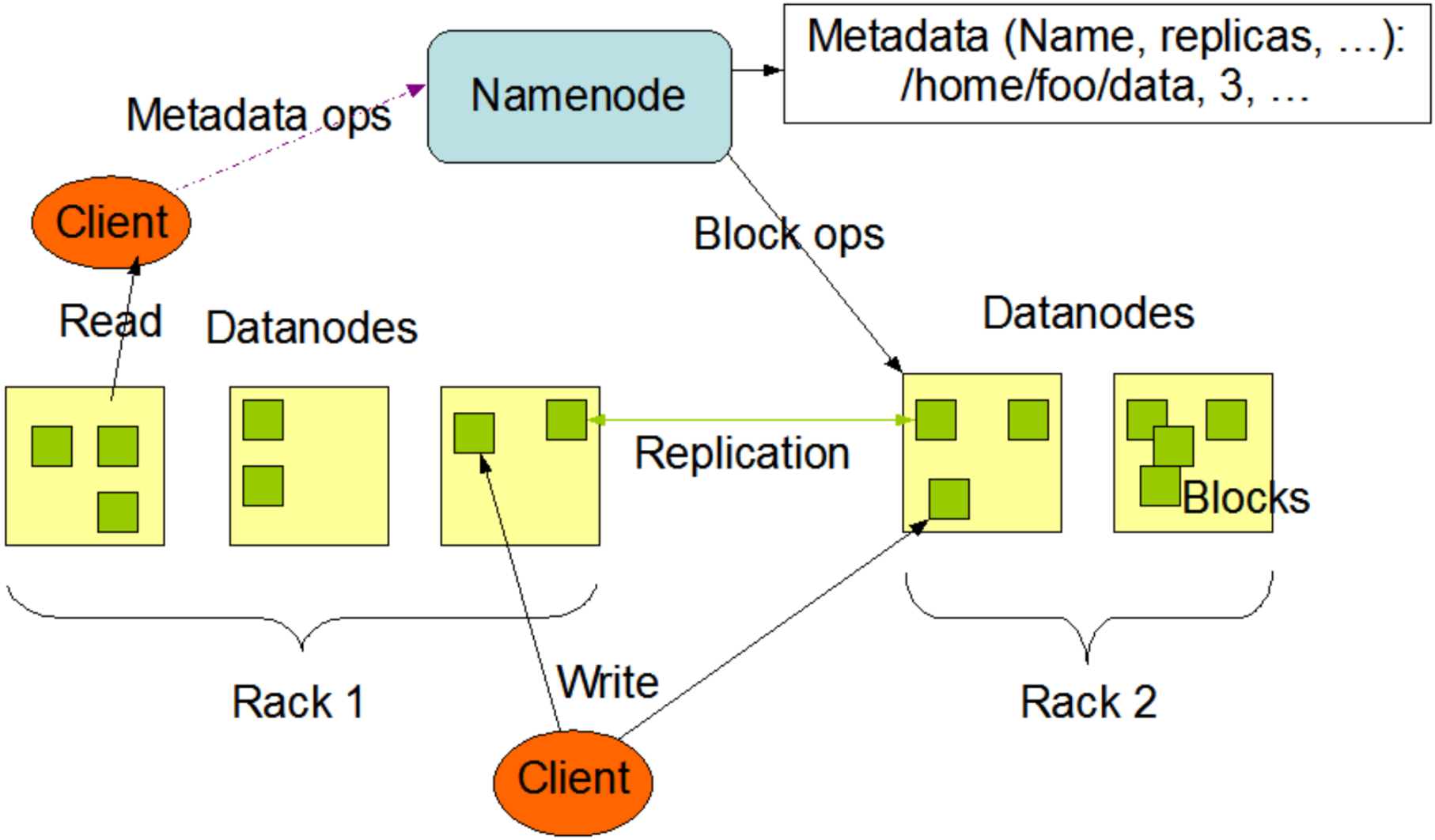


Example with number of replicas per chunk = 2

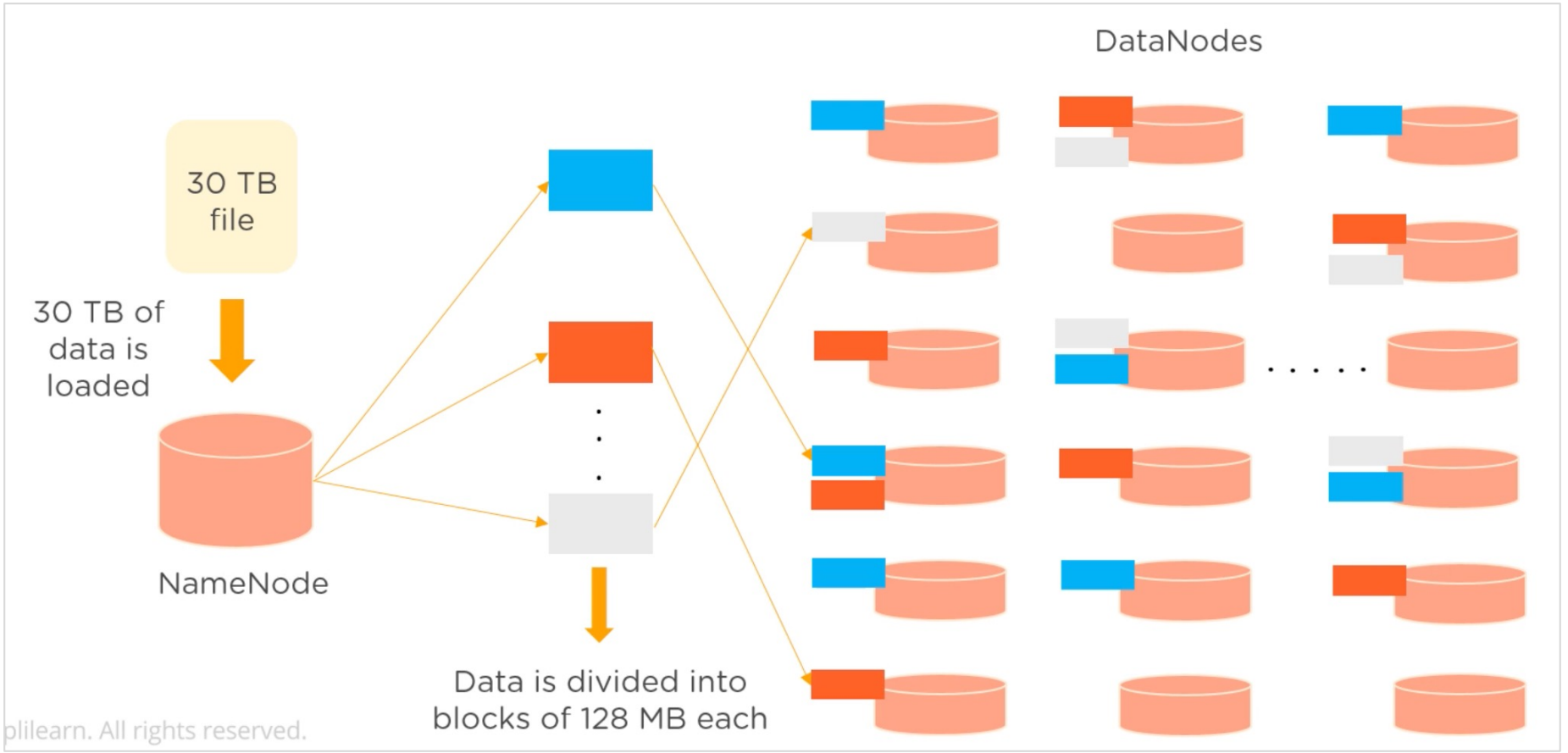
HDFS

- Master-Slave architecture
  - Master: NameNode
  - Slave: DataNode
- The Master node is a special node/server that
  - Stores HDFS metadata
    - e.g., the mapping between the name of a file and the location of its chunks
  - Might be replicated
- Client applications: file access through HDFS APIs
  - Talk to the master node to find data/chuck servers associated with the file of interest
  - Connect to the selected chunk servers to access data

Interaction between HDFS components



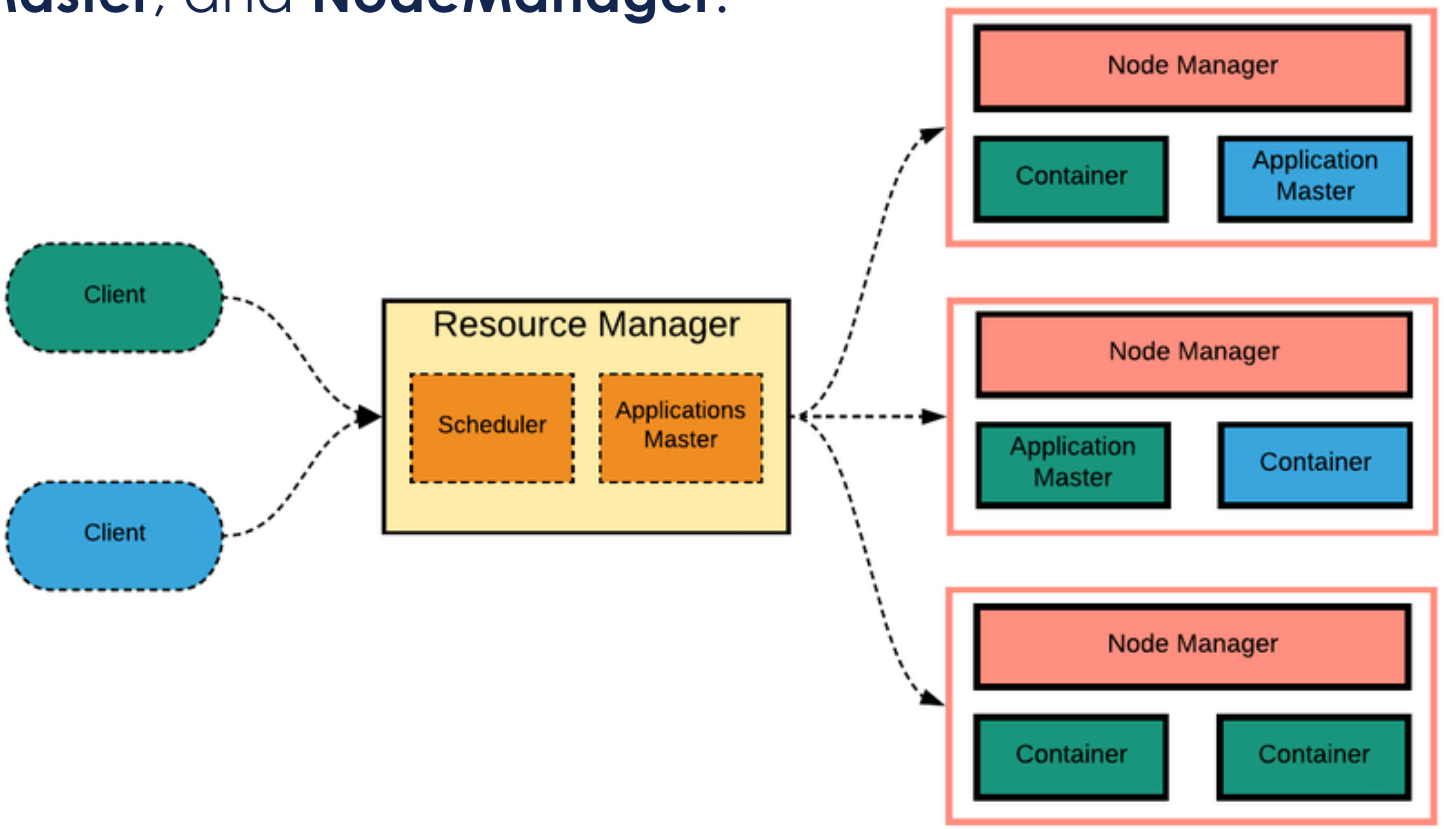
Storage example



https://www.youtube.com/watch?v=iANBytZ26MI

Hadoop YARN

- Resource management and job scheduling.
  - Global **ResourceManager**, per-application **Application Master**, and **NodeManager**.



Hadoop MapReduce

- MapReduce is the data processing layer of Hadoop.
- MapReduce job comprises a number of map tasks and reduces tasks.
  - Each task works on a part of data. This distributes the load across the cluster.
  - **Map task** load, parse, transform and filter data.
  - Each **reduce task** works on the sub-set of output from the map tasks (e.g., by applying grouping and aggregation).

Apache Hadoop

- Install & config
- Run
  ~ **sbin** ./start-dfs.sh
- Basic help for all the commands
  ~ **hadoop**
- Distributed file system commands
  ~ **hadoop fs**
- Execution of MapReduce jobs
  ~ **hadoop jar**

HDFS commands

~ **hadoop** fs [generic options]
[-appendToFile <localsrc> ... <dst>]
[-cat [-ignoreCrc] <src> ...]
[-checksum [-v] <src> ...]
[-chgrp [-R] GROUP PATH...]
[-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
[-chown [-R] [OWNER][:[GROUP]] PATH...]
[-copyFromLocal [-f] [-p] [-l] [-d] [-t <thread count>] <localsrc> ... <dst>]
[-copyToLocal [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
[-count [-q] [-h] [-v] [-t [<storage type>]] [-u] [-x] [-e] <path> ...]
[-cp [-f] [-p | -p[topax]] [-d] <src> ... <dst>]
[-createSnapshot <snapshotDir> [<snapshotName>]]
[-deleteSnapshot <snapshotDir> <snapshotName>]
[-df [-h] [<path> ...]]
[-du [-s] [-h] [-v] [-x] <path> ...]
[-expunge [-immediate] [-fs <path>]]
[-find <path> ... <expression> ...]
[-get [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
…

Hadoop – HDFS Commands

- Help
  ~ hadoop fs -help
- Creating a directory
  ~ hadoop fs –mkdir /myhdfsdir
- Listing a directory
  ~ hadoop fs –ls
  ~ hadoop fs –ls /myhdfsdir
- Copy a file/dir from local file system to HDFS
  ^ hadoop fs –put /myhdfsdir/ /user/xpto/
- Get a file/dir from HDFS to local file system
  ~ hadoop fs –get /user/xpto/ /myhdfsdir/
  - … and many more

  https://www.tutorialspoint.com/hadoop/hadoop_command_reference.htm

Execute a MapReduce job

- Run the job

  ~ **hadoop jar** WordCount.jar /myhdfsdir/myapp/input1 /myhdfsdir/myapp/output1

- Retrieve and explore the job result

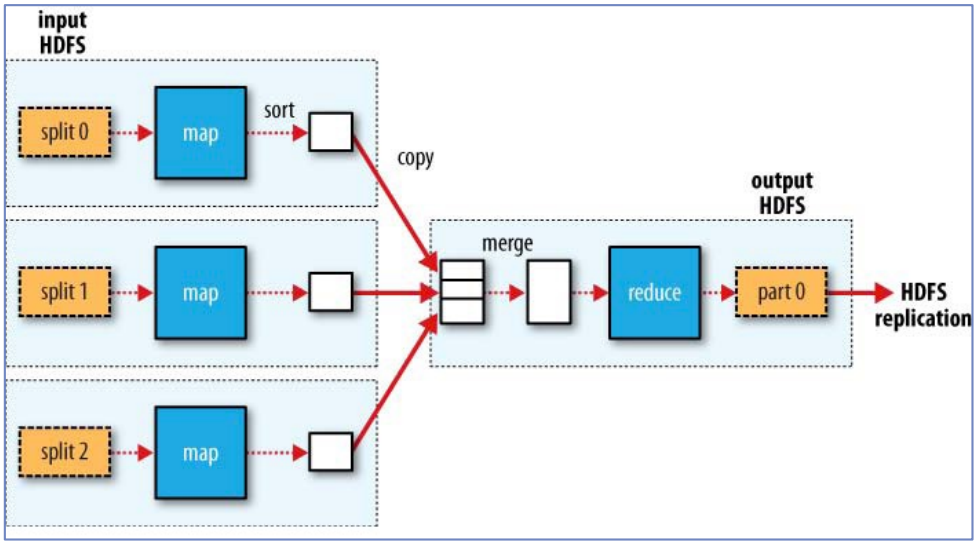
  ~ **hadoop fs –copyToLocal** /myhdfsdir/myapp/output1/part-r-00000 result.txt
  ~ cat result.txt

- Clean the output HDFS directory

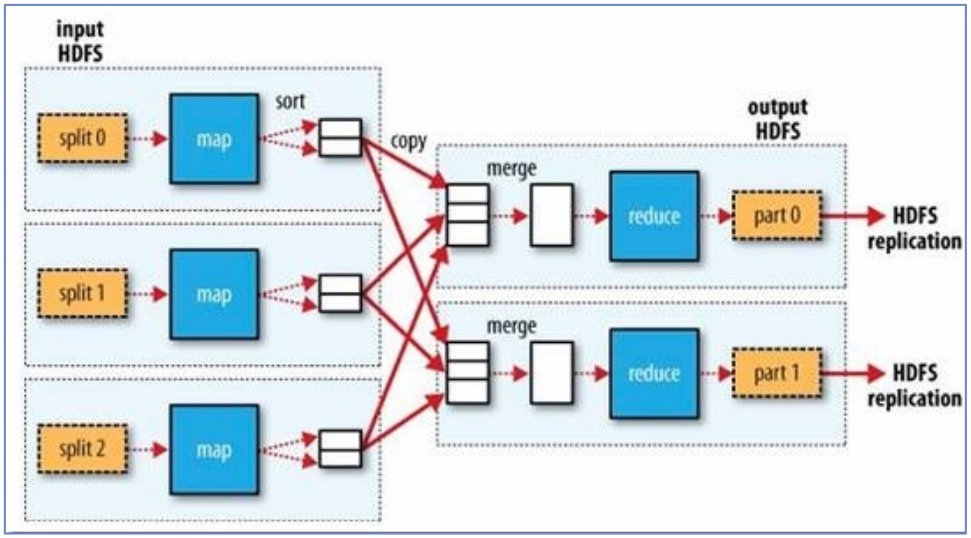
  ~ **hadoop fs -rm** /myhdfsdir/myapp/output1/

Task execution - distinct configurations

single reduce task



multiple reduce task



Java Interface

- **Mapper** class
  - Implementation of the **map function**
  - Template parameters
    `KEYIN, VALUEIN` – types of input key-value pairs
    `KEYOUT, VALUEOUT` – types of intermediate key-value pairs
  - Intermediate pairs are emitted via **`context.write(k, v)`**

```
class MyMapper extends Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
  @Override
  public void map(KEYIN key, VALUEIN value, Context context)
    throws IOException, InterruptedException
  {
    // Implementation
  }
}
```

Java Interface

- **Reducer** class
  - Implementation of the **reduce function**
  - Template parameters
    KEYIN, VALUEIN – types of intermediate key-value pairs
    KEYOUT, VALUEOUT – types of output key-value pairs
  - Output pairs are emitted via **context.write(k, v)**

```
class MyReducer extends Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
  @Override
  public void reduce(KEYIN key, Iterable<VALUEIN> values, Context context)
    throws IOException, InterruptedException
  {
    // Implementation
  }
}
```

MapReduce – WordCount example (1)

```
public class WordCount extends Configured implements Tool {
    private static IntWritable ONE = new IntWritable(1);
    @Override
    public int run(String[] args) throws Exception {
        FileSystem fs = FileSystem.get(getConf());
        Job job = Job.getInstance(getConf());
        job.setJarByClass(WordCount.class);
        job.setJobName("WordCount");
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        if (fs.exists(new Path(args[1])))
            fs.delete(new Path(args[1]), true);
        boolean success = job.waitForCompletion(true);
        return success ? 0 : 1;
    }
```

Configurations

Mapper class

Reducer class

MapReduce – WordCount example (2)

```
...
    public static void main(String[] args) throws Exception {
        int ret = ToolRunner.run(new WordCount(), args);
        System.exit(ret);
    }

    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
        @Override
        public void map(LongWritable key, Text value, Context context) throws
                                                    IOException, InterruptedException {
            String line = value.toString();
            String[] words = line.split("[\t\n.,:; ?!-/()\\[\\]\"\']+");
            for (String word : words) {
                if (word.trim().length() > 0) {
                    Text text = new Text();
                    text.set(word);
                    context.write(text, ONE);
                }
            }
        }
    }   // end class Map
```

Mapper

MapReduce – WordCount example (3)

Reducer

```
...

    public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>
{
        public void reduce(Text key, Iterable<IntWritable> values, Context context)
                                                throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values)
                sum += val.get();
            context.write(key, new IntWritable(sum));
        }
    }   // end class Reduce

}   // end class WordCount
```

MapReduce – Run the example

Compile

```
> $ javac -classpath $CLASSPATH -d WordCountDir WordCount.java
```

Create JAR

```
> $ jar -cvf WordCount.jar -C WordCountDir/ .
```

Run WordCount App

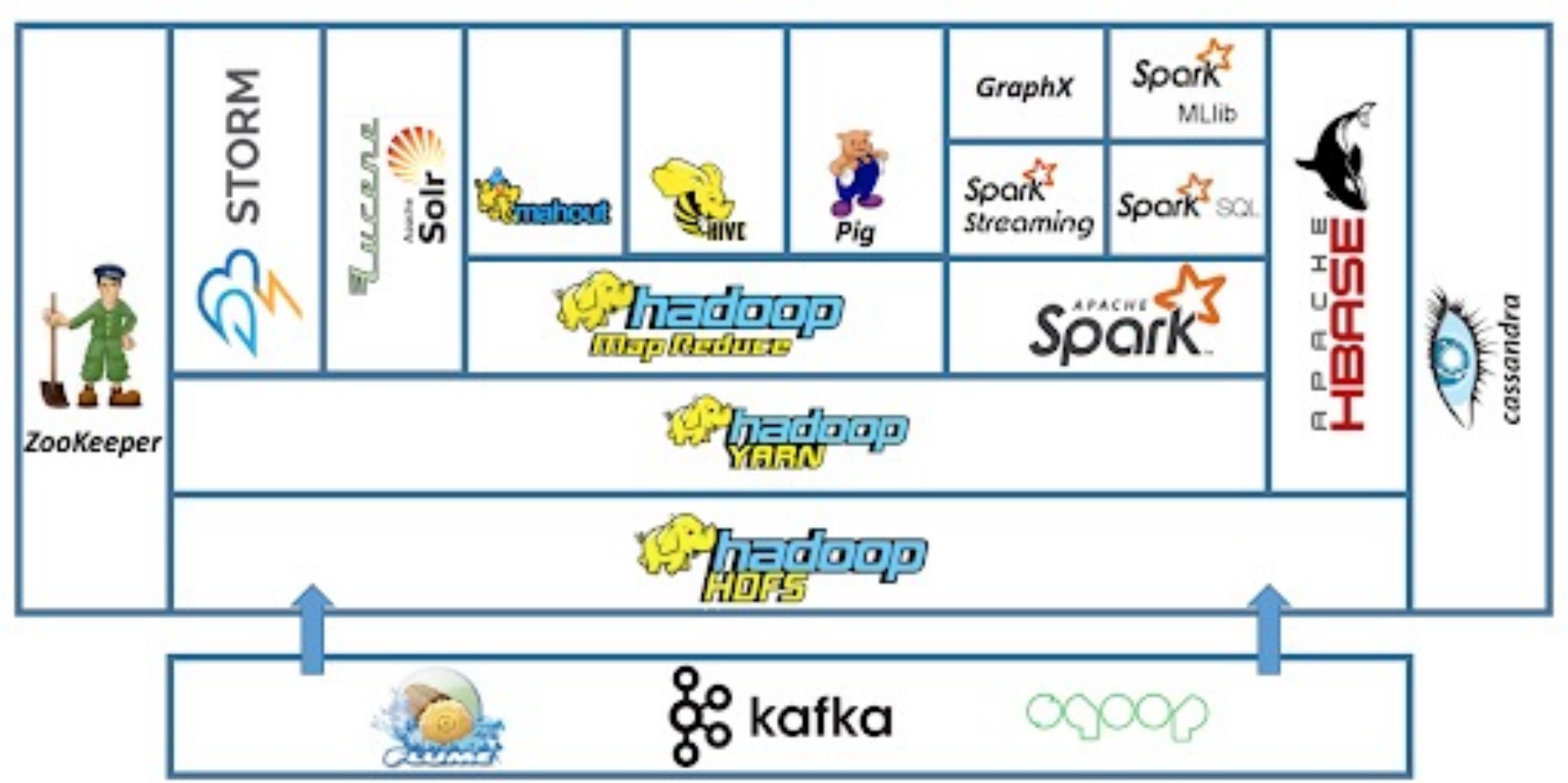
```
> $ hadoop jar WordCount.jar /in/test.txt /out/wc
```

See the Result

```
> $ hadoop fs -cat /out/wc/part-r-00000

ABOUT 1
ACCOUNT 2
ACTUAL 1
ADDITIONAL 1
ADVANCING 2
...
```

Hadoop ecosystem



Summary

- Concept
- MapReduce framework
  - Programming Model
  - Execution Plan
  - Dataflow
- MapReduce Applications
- Apache Hadoop
  - HDFS
  - MapReduce Programming