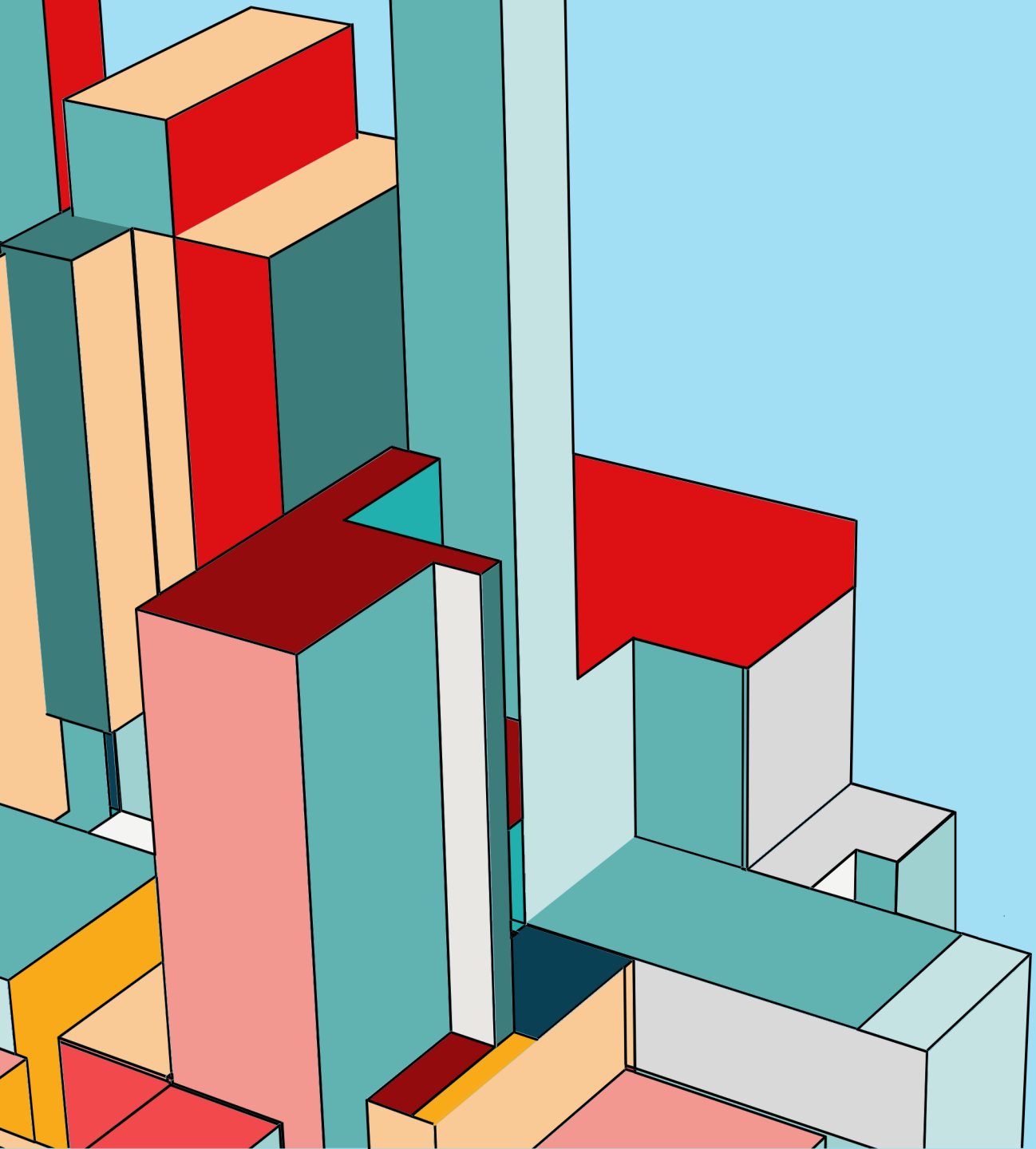
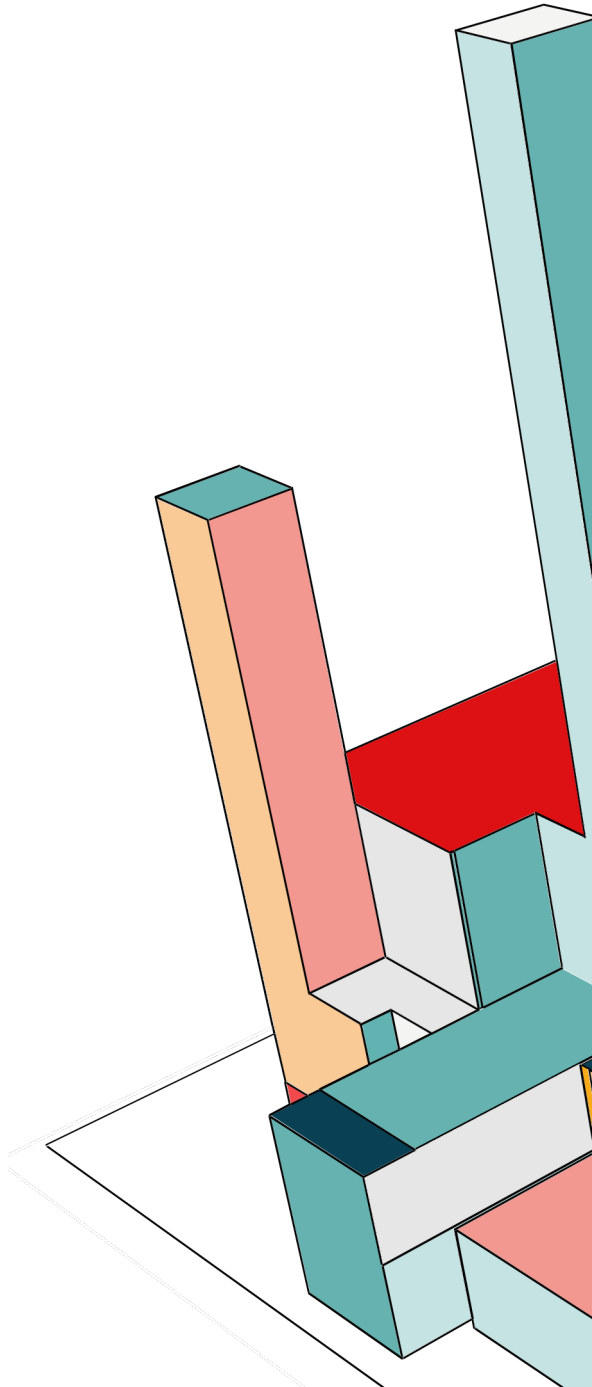


DAY ONE

AGENDA

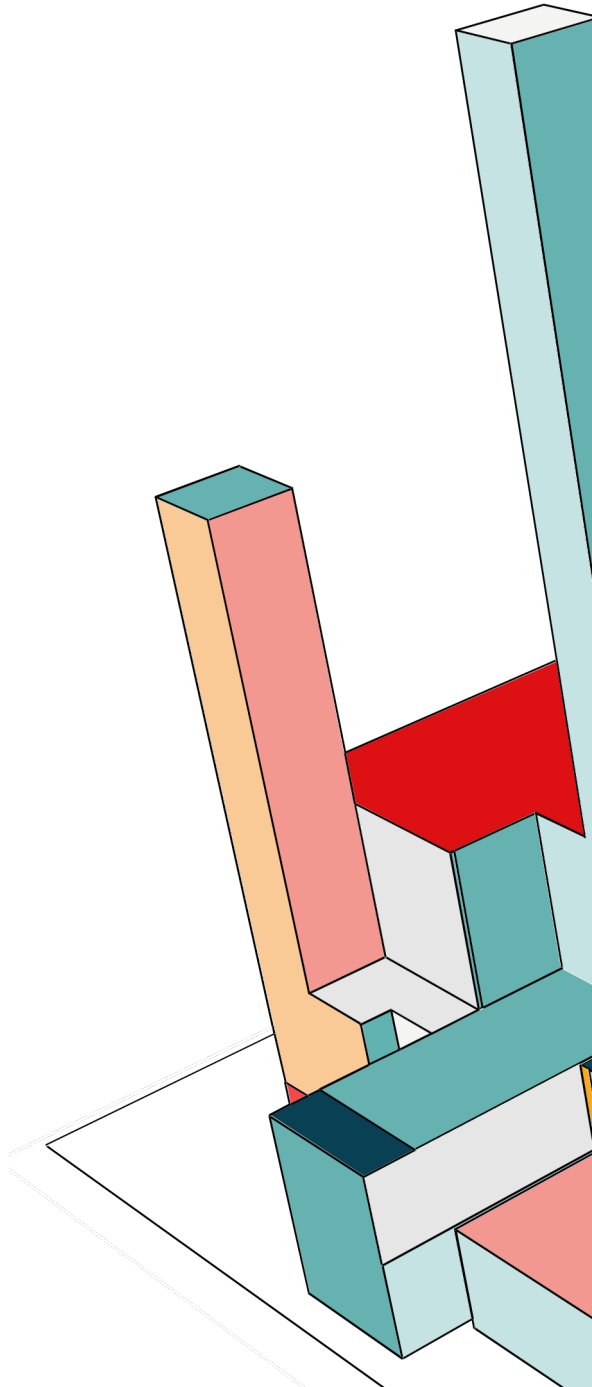
- Course Introduction
- Learning outcomes
- Assignments and expectations

- Intro to Big Data Management Fabric
- Hadoop
- Spark



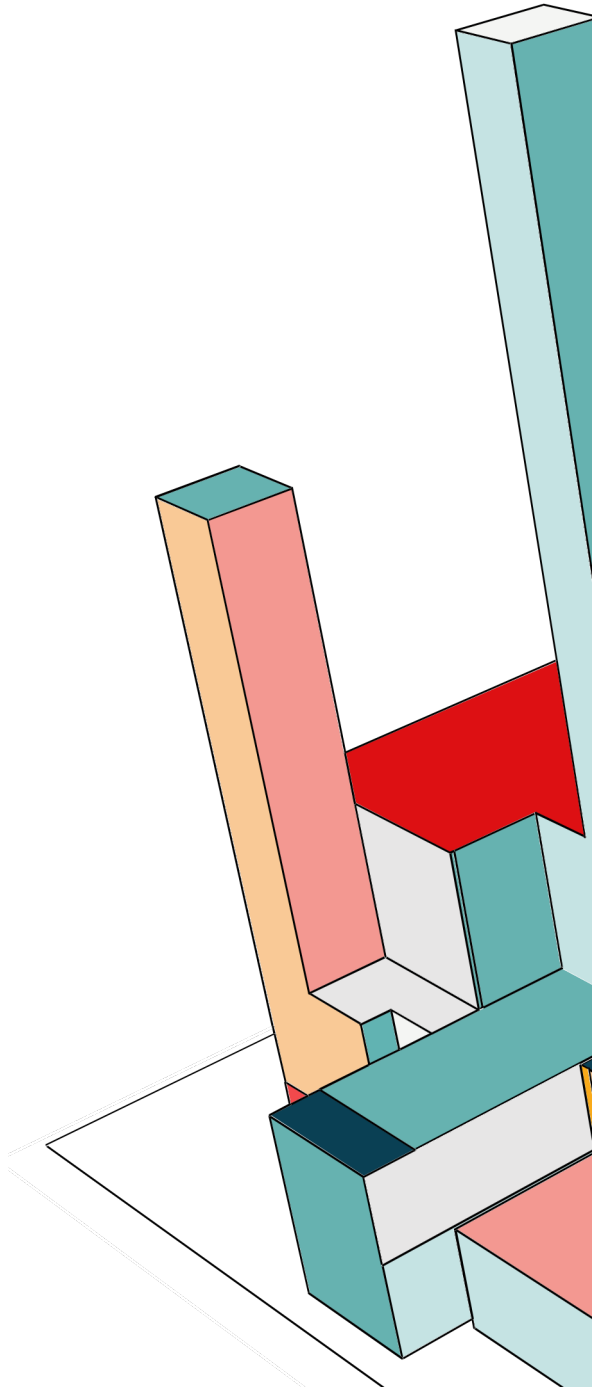
COURSE INTRO:
SYLLABUS

- Day 1 : Key concepts
- Day 2: Hadoop
    - Hadoop Distributed File System (HDFS)
    - MapReduce
    - HDFS commands
- Day 3: YARN (Thur. on Zoom 8-10 am, OH:1-2 pm)
- Day 4: Spark



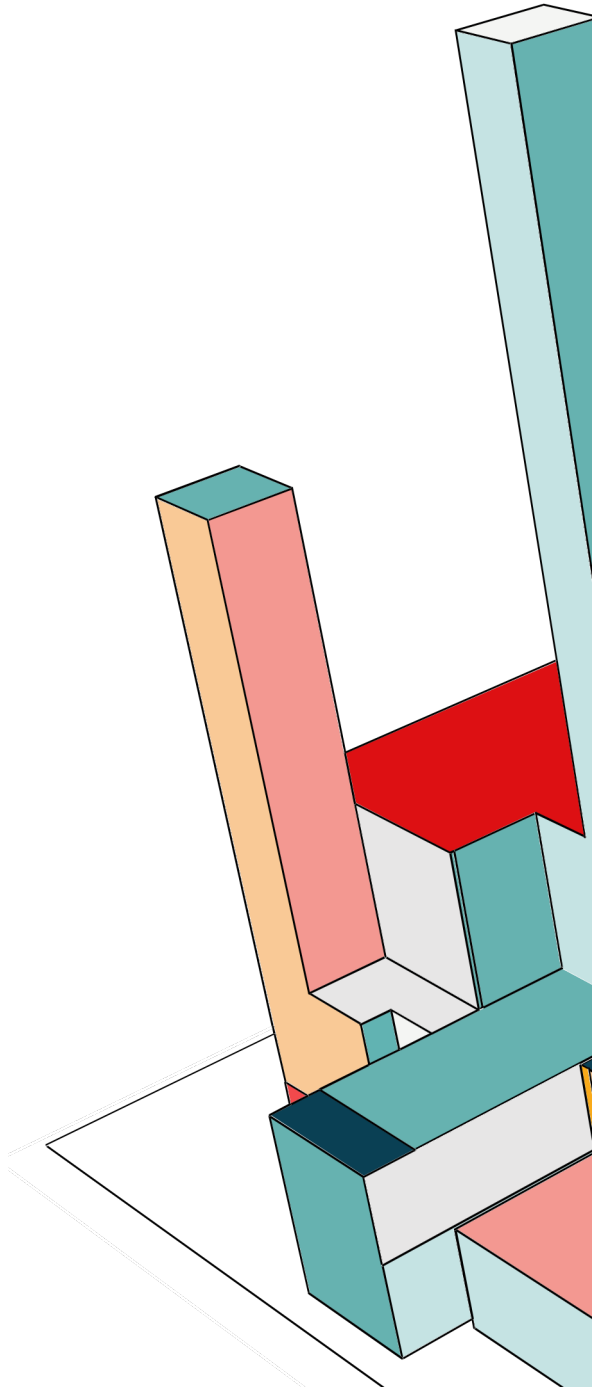
SYLLABUS

- Day 5: Apache Spark
    - Resilient Distributed Dataset (RDD)
- Day 6: Spark SQL
- Day 7: Spark MLlib
- Day 8: Spark Streaming



LEARNING OUTCOME:

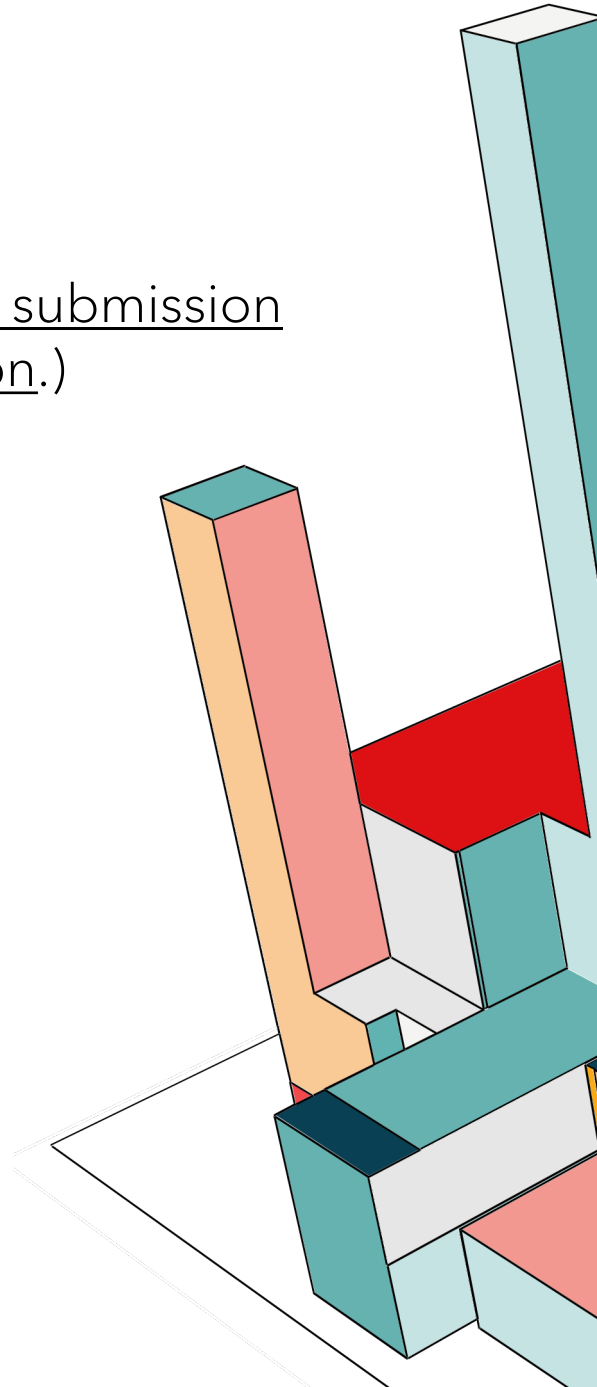
- To gain understanding of the characteristic of big data.
- To know techniques for working on big data platforms.
- To have some experiences of performing the techniques.



ASSIGNMENTS:
GRADING

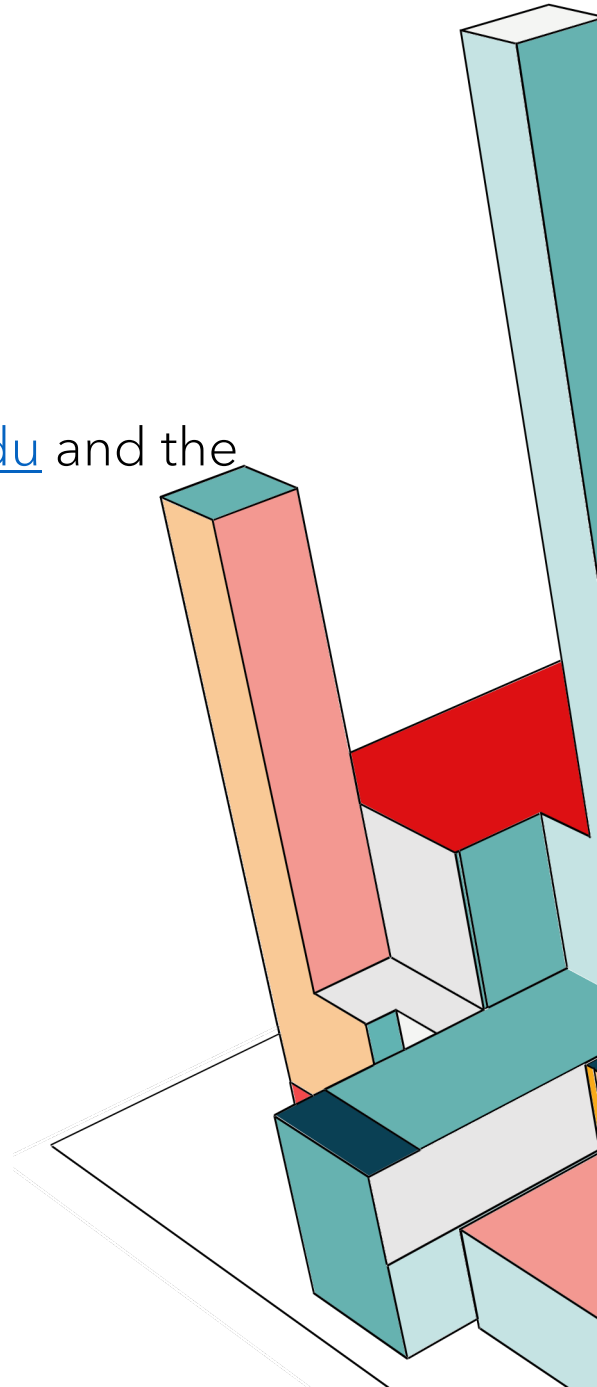
- ALL assignments are due **May 31, 2024, 11:59 PM, PST.** (Note: Any late submission after the due day could lead to a 10% deduction on each late submission.)

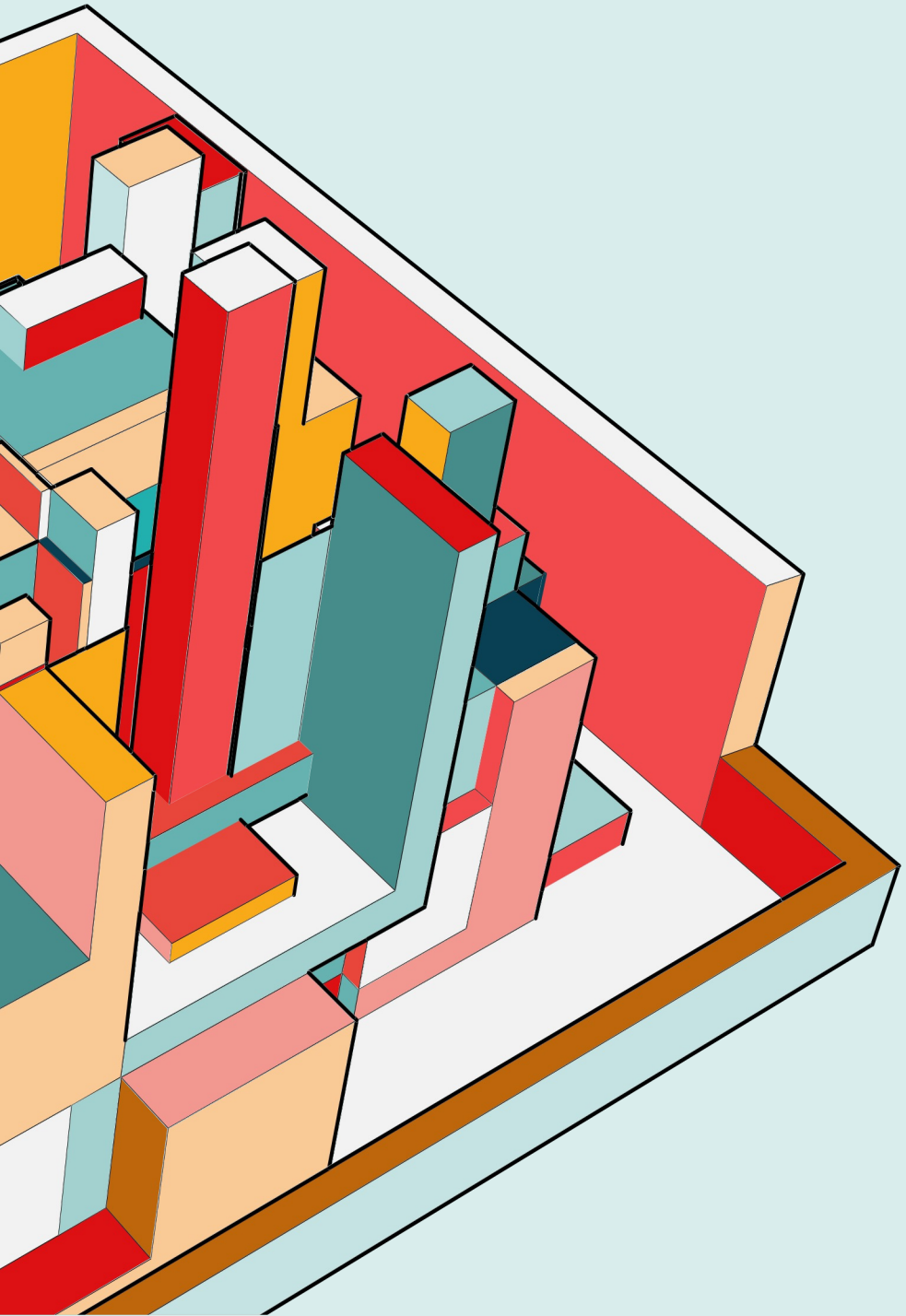
Assignments	% of Grade
Discussions	30%
Assignments	65%
Quizzes	5%
Total	100%



ATTENDANCE:

- I update the attendance sheet periodically. Email me at kkowarsc@uci.edu and the department for sick absences or emergency events.





BIG DATA

HOW LARGE CAN OUR DATASET BE?

- Enron SPAM Data (100k articles)
- Google ngram corpus (2.2 GB)
- Wikipedia graph (2 GB)
- $100 Billion problem (Click-through rate or CTR) prediction (20-30 GB)
- Airlines and Weather - predicting delays (20-40GB)

HOW LARGE CAN OUR DATA BE?

When it is O(n): about 33 minutes

Assumption:
Your personal laptop has the following features"

- Storage: 1TB
- RAM: 16GB
- Inteli7

- Your task is working on a dataset of 20GB

Complexity	Runtime
O(n)	33 mins
O(nlogn)	46 mins
O(n^2)	111 days
O(2^n)	340 years

HOW LARGE CAN OUR DATA BE?

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        # Last i elements are already in place
        for j in range(0, n-i-1):
            # Swap if the element found is greater than the next
element
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

arr = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(arr)
print("Sorted array:", arr)
```

Time complexity: O(n^2)

Inefficient algorithms are usually Naïve Modeling with nested loops.

HOW LARGE CAN OUR DATA BE?

An algorithm with an exponential runtime complexity of O(2^n) is generating all subset of a set.

Recursive algorithm

```
def generate_subsets(s):
    if len(s) == 0:
        return [[]]

    subsets_without_first = generate_subsets(s[1:])
    subsets_with_first = [[s[0]] + subset for subset in
subsets_without_first]

    return subsets_without_first + subsets_with_first

my_set = [1, 2, 3]
subsets = generate_subsets(my_set)
print(subsets)
```

HOW LARGE CAN OUR DATA BE?

Fibonacci sequence using naïve modeling approach.

Recursive algorithm

```
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)

result = fibonacci(5)
print(result)
```

Time complexity: O(2^n)

HOW LARGE CAN OUR DATA BE?

Fibonacci Sequence

```
def fibonacci_generator(n):
    a, b = 0, 1
    count = 0
    while count < n:
        yield a
        a, b = b, a + b
        count += 1

# Using the Fibonacci generator
fib_gen = fibonacci_generator(5)
for num in fib_gen:
    print(num)
```

Time complexity: O(n)

HOW TO DEAL WITH BIG DATA?

Recall the traveling salesman problem: find out the shortest route that visits each city exactly once and returns to the original city. The naive brute-force approach is O(n!) or O(2^n)

HOW TO DEAL WITH BIG DATA?

Recall the traveling salesman problem: find out the shortest route that visits each city exactly once and returns to the original city. The naive brute-force approach is O(n!) or O(2^n)

PARALLELISM

Divide and Conquer

DIVIDE AND CONQUER

Goal

1. Scalability to large data volumes
    - Scan 1,000 TB (1PB) on one node at 50 MB/s = 120 days
    - Scan on 10,000-node cluster = 16 minutes!
2. Cost efficiency:
    - Commodity nodes (cheap but unreliable)
    - Commodity network
    - Automatic fault tolerance (fewer admins)
    - Easy to use (fewer programmers)

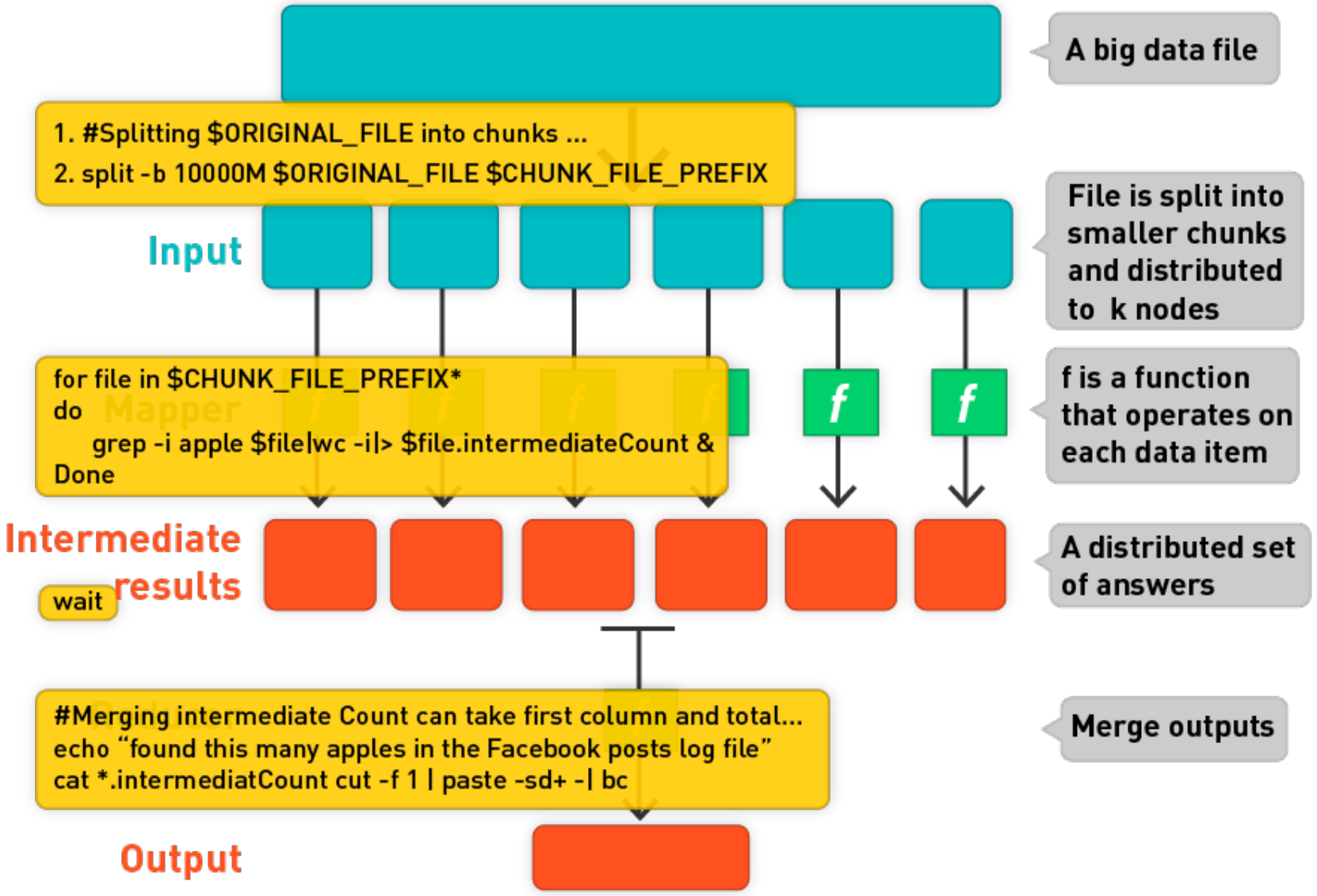
Challenges of Group Work

1. Division of work
2. Coordination costs (partition problem, communicate)
    - Coordination costs represent time and energy that group work consumes that individual work does not
    - They include the time it takes to coordinate schedules, arrange meetings, meet, correspond, make decisions collectively
    - Integrate the contributions of group members, etc.
    - The time spent on each of these tasks may be small, but together they are significant
3. Coordination costs can't be eliminated
4. Synergy, motivation are also costs
5. Similar challenges exist in distributed computing

PARALLEL COMPUTING

- Breaking the problem into independent components so that each processing element can execute its part of the algorithm simultaneously with the others.
- The processing elements can be diverse and include resources such as
  - a single computer with multiple processors,
  - several networked computers, specialized hardware, or
  - a combination of the above.
- A type of synchronization through devices

Schematic of Parallel Processing



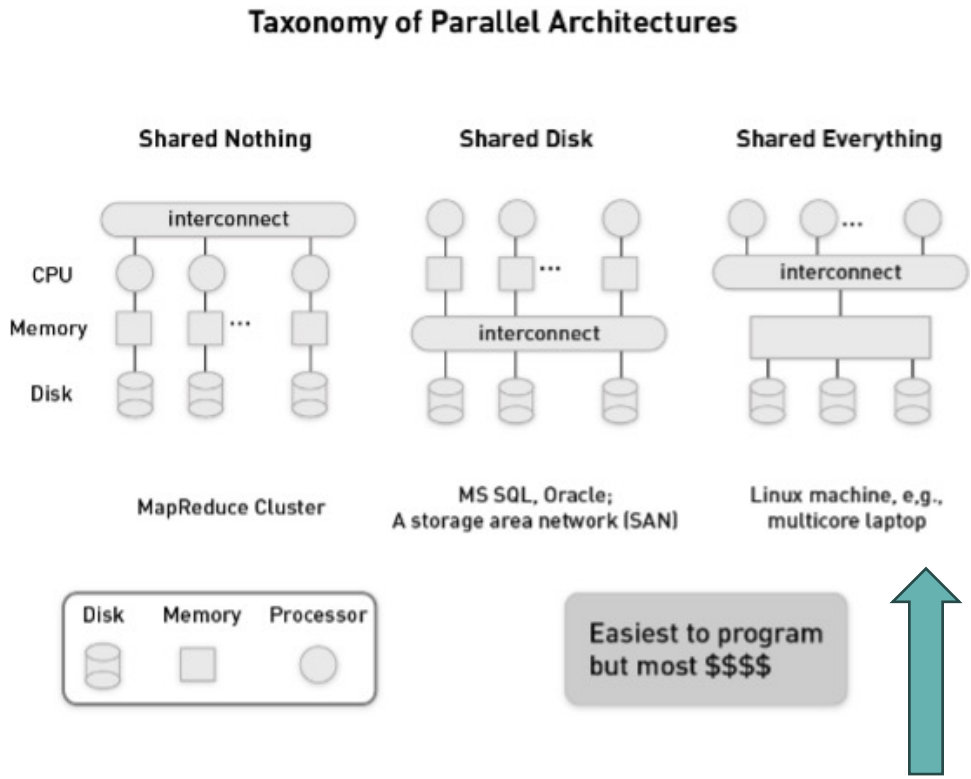
PARALLEL COMPUTERS

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism.

1. Single computer
    - With multicore and multi-processor computers having multiple processing elements within a single machine
2. Clusters of computers
    - While clusters, massively parallel processors (MPPs), and grids use multiple computers to work on the same task
3. Specialized parallel computer architectures are sometimes used alongside traditional processors for accelerating specific tasks, e.g., GPUs for graphics processing

PARALLEL SYSTEMS

- Resources
  - CPU
  - Memory
  - Disk
  - Network
- There are many parallel architectures that are commonly used in practice:
  - Shared nothing
  - Shared disc
  - Shared all



Resources, like memory and I/O devices, are accessible by all processors.

PARALLEL COMPUTERS

Hardware Overview:

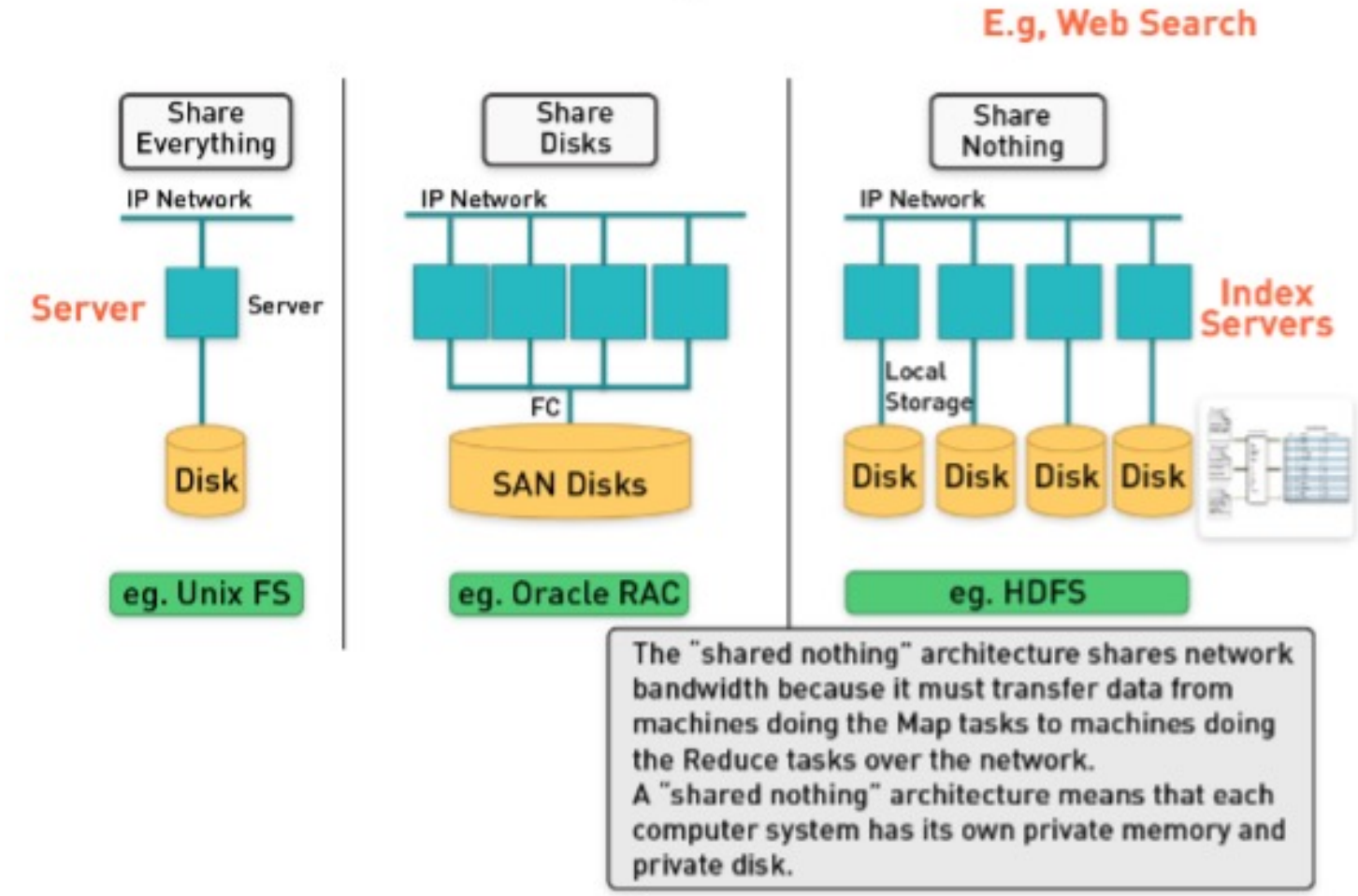
Windows users

Using Task Manager

1. **Right-click on the Taskbar**: This is the bar at the bottom of your screen.
2. **Select "Task Manager"**: This opens the Task Manager window.
3. **Click on "Performance" Tab**: In Task Manager, go to the "Performance" tab.
4. **Click on "CPU"**: This will display your processor information, including the number of cores.

Model Name: MacBook Pro
Model Identifier: MacBookPro15,1
Processor Name: 6-Core Intel Core i7
Processor Speed: 2.6 GHz
Number of Processors: 1
Total Number of Cores: 6
L2 Cache (per Core): 256 KB
L3 Cache: 9 MB
Hyper-Threading Technology: Enabled
Memory: 16 GB
System Firmware Version: 2022.100.22.0.0 (iBridge: 21.16.4222.0.0,0)
OS Loader Version: 580~1678
Serial Number (system): C02YG2C0JG5J
Hardware UUID: 418D713A-2808-54D3-B7A2-8B6F0C7D8251
Provisioning UDID: 418D713A-2808-54D3-B7A2-8B6F0C7D8251
Activation Lock Status: Enabled

SHARE ARCHITECTURE



DATA LOCALITY AND ISOLATION

Moving data to computation nodes is hard

Network bandwidth is limited

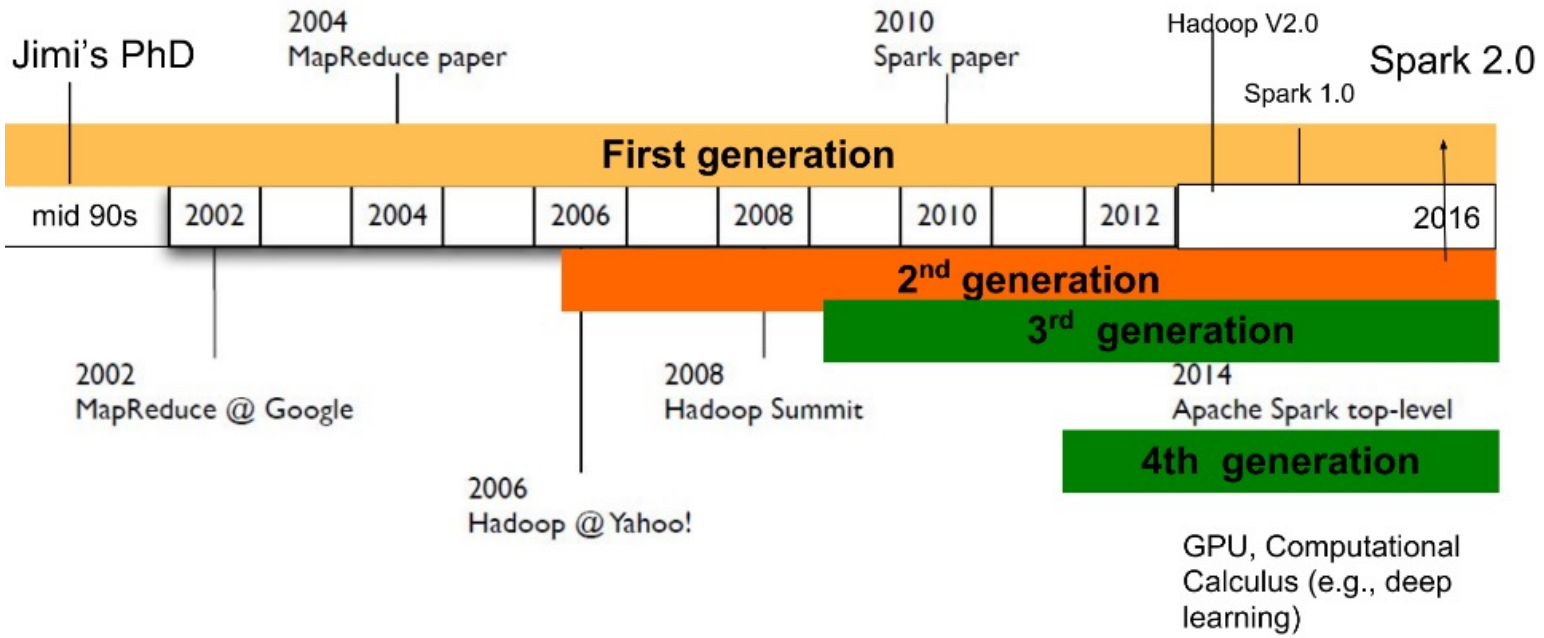
FRAMEWORK FOR PARALLEL COMPUTATION

- Cluster and grid computing
  - Provide logical abstractions
  - Libraries implementing the message passing interface (MPI) for same local network
  - MapReduce is a framework processing parallelizable problems across huge data sets, using large number of pcs or nodes, cluster or grid (nodes are shared across geographically or heterogeneous hardware)

MAP AND REDUCE

- Map phase dominates, the complexity is O(n*m), where n:input size; m: number of mappers (like log file analysis)
- If the reducer phase dominates, the overall complexity is O(n*r*m), where r is number of reduce tasks (example?)
- What if the number of tasks are processed by many computers?

EVOLUTION OF MAPREDUCE FRAMEWORK



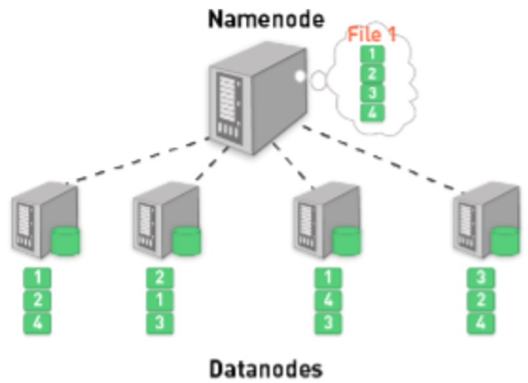
MAPREDUCE + SCALABILITY + FAULT TOLERANCE

- Inspired by functional programming
  - The model is inspired by the map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as in their original forms
- Scalability and fault tolerance
  - The key contributions of the MapReduce framework are not the actual map and reduce functions but the scalability and fault tolerance achieved for a variety of applications by optimizing the execution engine once
- Optimizing the communication
  - Optimizing the communication cost is essential to a good MapReduce algorithm
- The use of this model is beneficial only when the optimized distributed shuffle operation (which reduces network communication cost) and fault-tolerance features of the MapReduce framework come into play

MapReduce Frameworks (cont.)

- The name MapReduce originally referred to the proprietary Google technology but has since become generic.
- A popular open-source implementation that has support for distributed shuffles is part of Apache Hadoop.
- MapReduce libraries have been written in many programming languages, with different levels of optimization.

Hadoop Distributed File System



- Single **namenode**(master node) stores metadata (file names, block locations, etc.)
  - May be replicated also
- Datanodes provide redundant store for the data blocksn
- Client library provides file access
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data
    - Master node is not a bottleneck
    - Computation is done at chunk node (close to data)

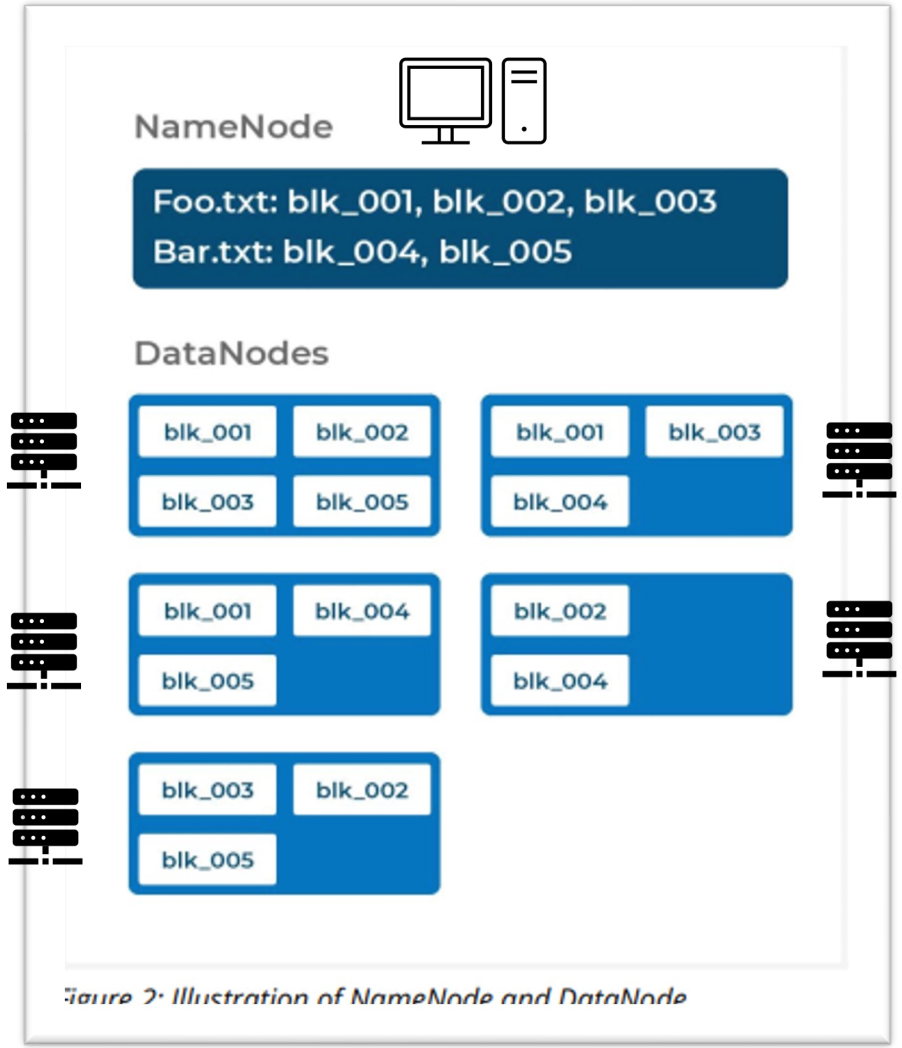


Figure 2: Illustration of NameNode and DataNode

Hadoop Distributed File System

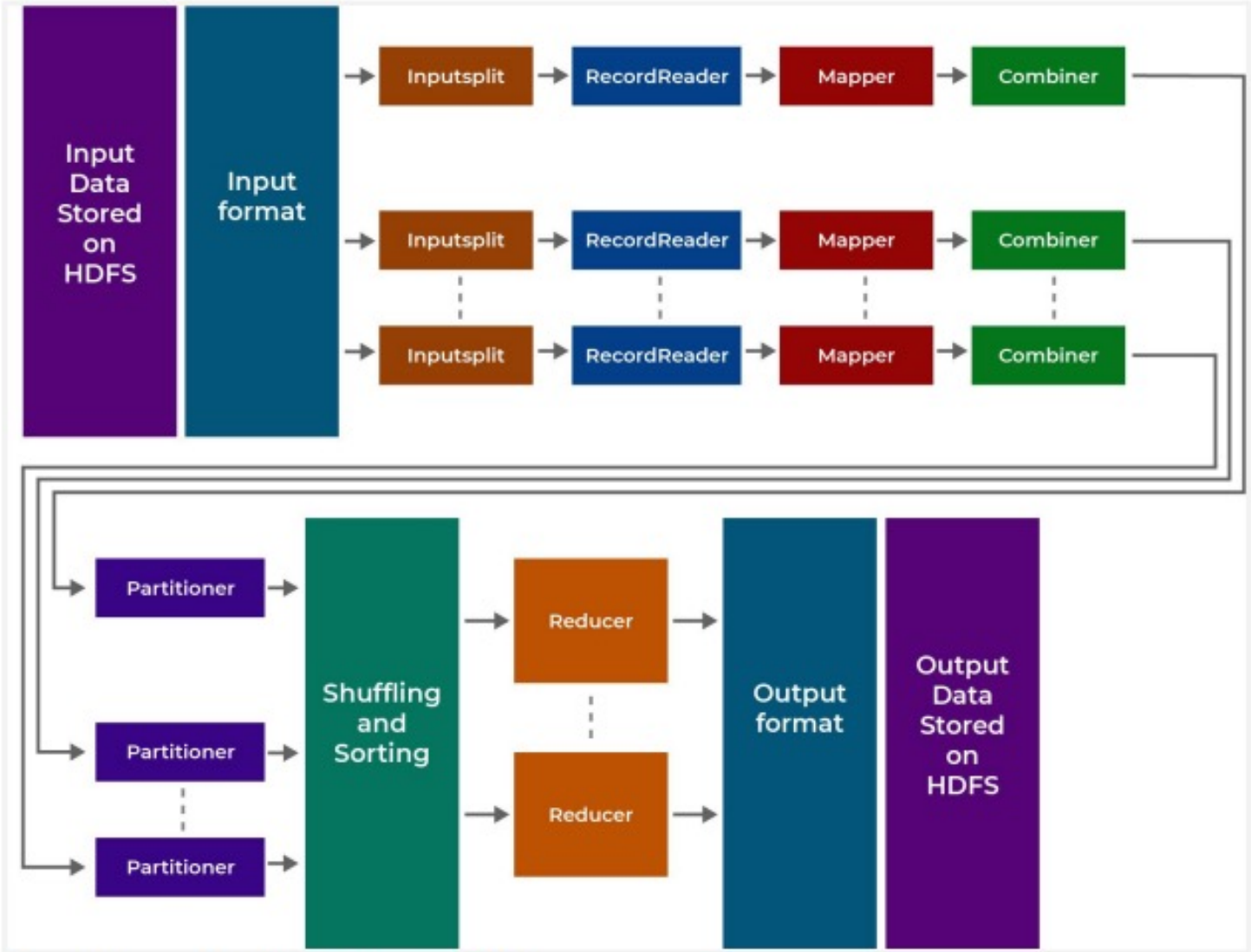
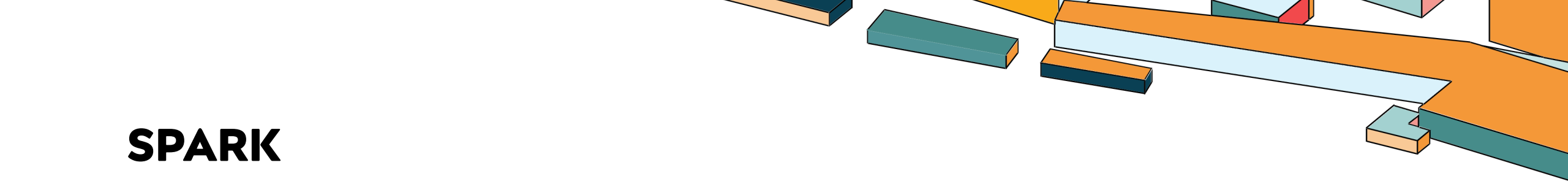


Image 3: Phases of the MapReduce model TBD

SPARK



SPARK

- An open-source, distributed computing framework designed for **speed** and ease of use.
- **In-memory** processing capabilities, allowing it to perform computations much faster than traditional batch processing systems like Hadoop MapReduce.
- Supports various programming languages (Scala, Java, Python, R) and offers **high-level APIs** like Spark SQL, Spark Streaming, MLlib (machine learning), and GraphX (graph processing).
- Resilient Distributed Datasets (**RDD**s) as the primary data abstraction, which are distributed, immutable, and fault-tolerant data structures.
- Offers interactive querying, real-time processing, iterative algorithms, and complex analytics workloads.

SPARK

An executor can do many tasks

Driver: orchestrates the job flow, schedule tasks, and is in working mode when the application is running

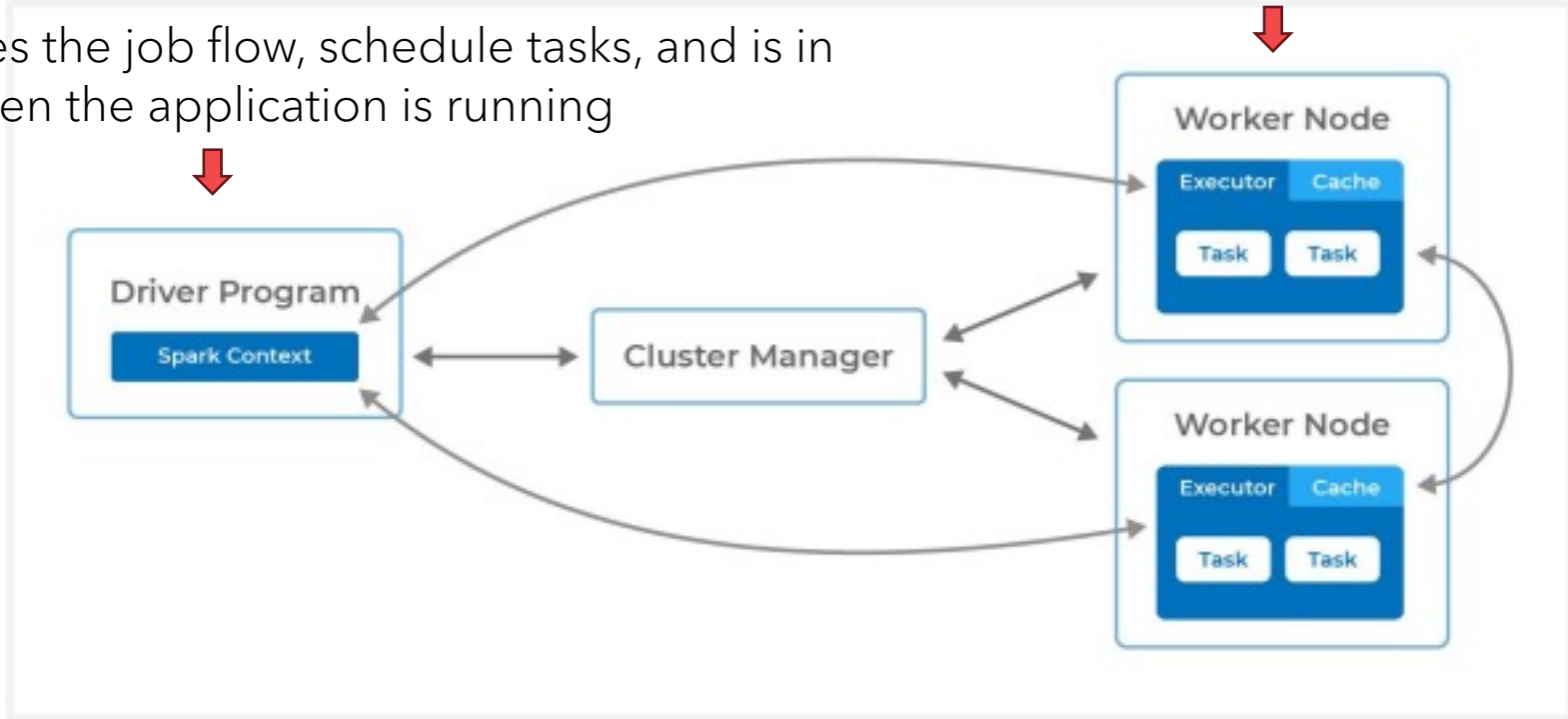


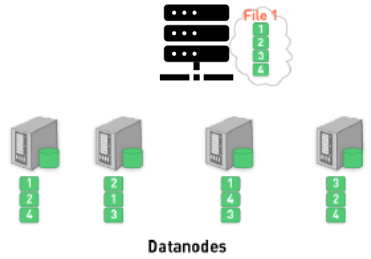
Figure 4: Spark flow

A Spark application maps to a single driver process and series executors.

DIFFERENCES

	Hadoop	Spark
Processing Model	Disk-based storage	In-memory processing
Data Abstraction	Stored in HDFS and using MapReduce	RDD
Use	Write MR jobs in Java	Higher-level APIs (Shell, PySpark) and libraries
Real-time Processing	Batch processing	Streaming

Hadoop Distributed File System



- Big files
  - 100s of GBs, TBs, even PBs
- Typical usage patterns
  - Append only
    - Data are rarely updated in place
  - Reads common
- Optimized for large files, sequential reads
- Files split into blocks (chunks): 64–128 MB, today 1GB–2GB
  - Blocks are replicated (usually three times) across several **datanodes** (called chunk or slave nodes)
  - Chunk nodes are compute nodes, too

What Is MapReduce Used For?

At Google:

-Index building for Google Search

-Article clustering for Google News

-Statistical machine translation

At Yahoo!:

-Index building for Yahoo! Search

-Spam detection for Yahoo! Mail

At Facebook:

-Data mining

-Ad optimization

-Spam detection

NativeX (Mobile Ad Tech):

-Ad optimization

-Building predictive models with 10s of GBs of data

-Process 10s of TBs of data

In research:

-Analyzing Wikipedia conflicts (PARC)

-Natural language processing (CMU)

-Bioinformatics (Maryland)

-Astronomical image analysis (Washington)

-Ocean climate simulation (Washington)

-Graph OLAP (NUS)

-& hellip;

- [Your application]

And for machine learning.

BUSINESS CASE

- Big Data Analytics in Marketing
  - Businesses use Hadoop or Spark for tasks such as customer segmentation, market basket analysis, sentiment analysis, and predictive modeling.

- Data Warehousing
  - Big data technique serves as a cost-effective solution for storing and processing vast amounts of data in a distributed manner.
  - Integrate and analyze diverse data sources, build data lakes.

BUSINESS CASE

- Log File Processing and Analysis
  - Processing and analyzing log data generated by applications, servers, and network devices in memory.
- Recommendation Systems
- Fraud Detection and Risk Management
- Internet of Things or IoT Data Processing
  - Massive volumes of real-time data from IoT devices

ANALYTICS AND BIG DATA

Type of Analysis	Description of Analysis
Basic Analysis	Slicing and dicing of data, reporting, and visualizations
Advanced Analysis	Analysis based on statistical/predictive modeling techniques and text analytics
Operational Analysis	Analysis of business processes

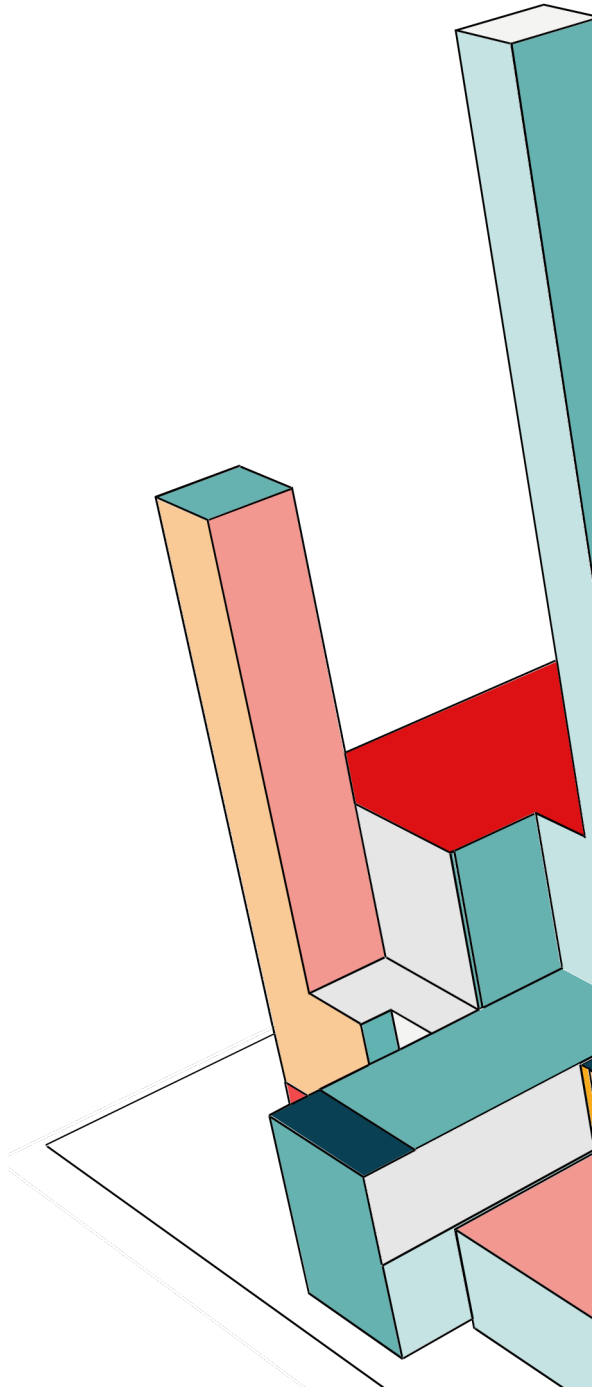
Table 1: Types of analysis

WHAT DID YOU LEARN?

- Quiz
- Discussion

DAY 2 INTRO TO HADOOP

- Hadoop Cluster
- Building blocks of Hadoop
    - Using Hadoop distributed file system
    - What is MapReduce?
        - Basics of a MR Flow



HADOOP

- Hadoop is an open-source distributed computing **framework** that allows for the distributed storage and processing of large datasets across clusters of computers.

- It consists of two main components: *Hadoop **Distributed** File System* (HDFS) for distributed storage and *MapReduce for distributed **processing***.

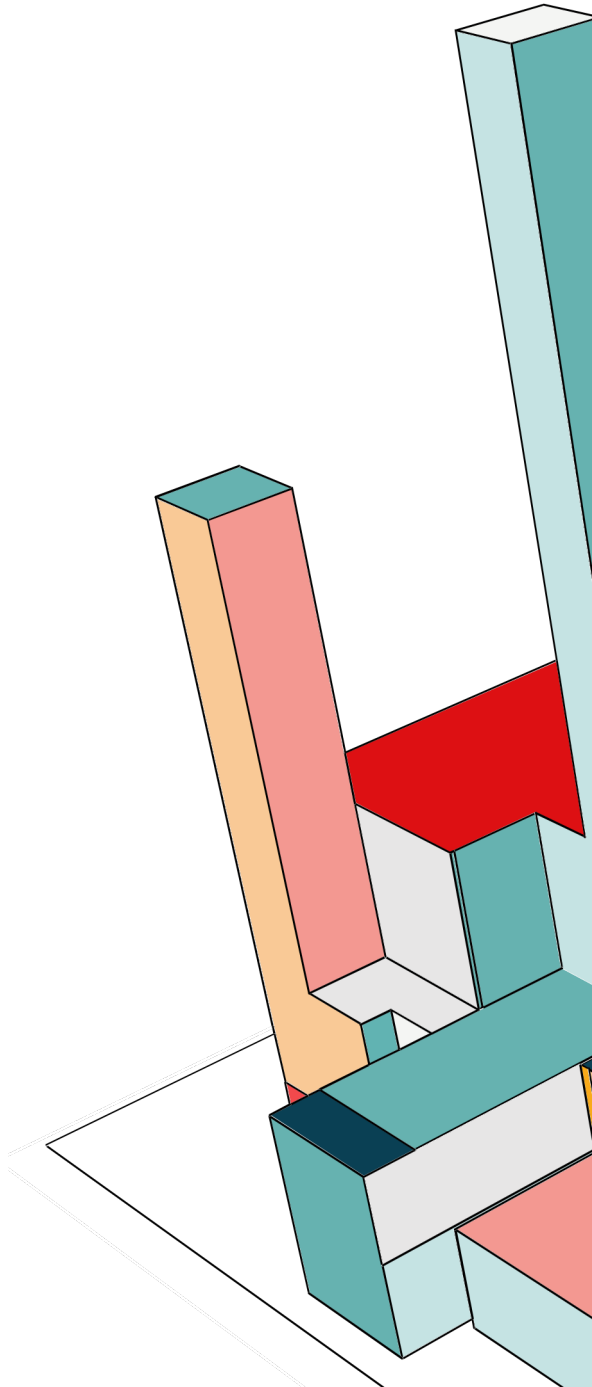
  - Single namespace for entire cluster

  - Replicates data 3X for fault tolerance

- Hadoop MapReduce (Google:20 PB /day)

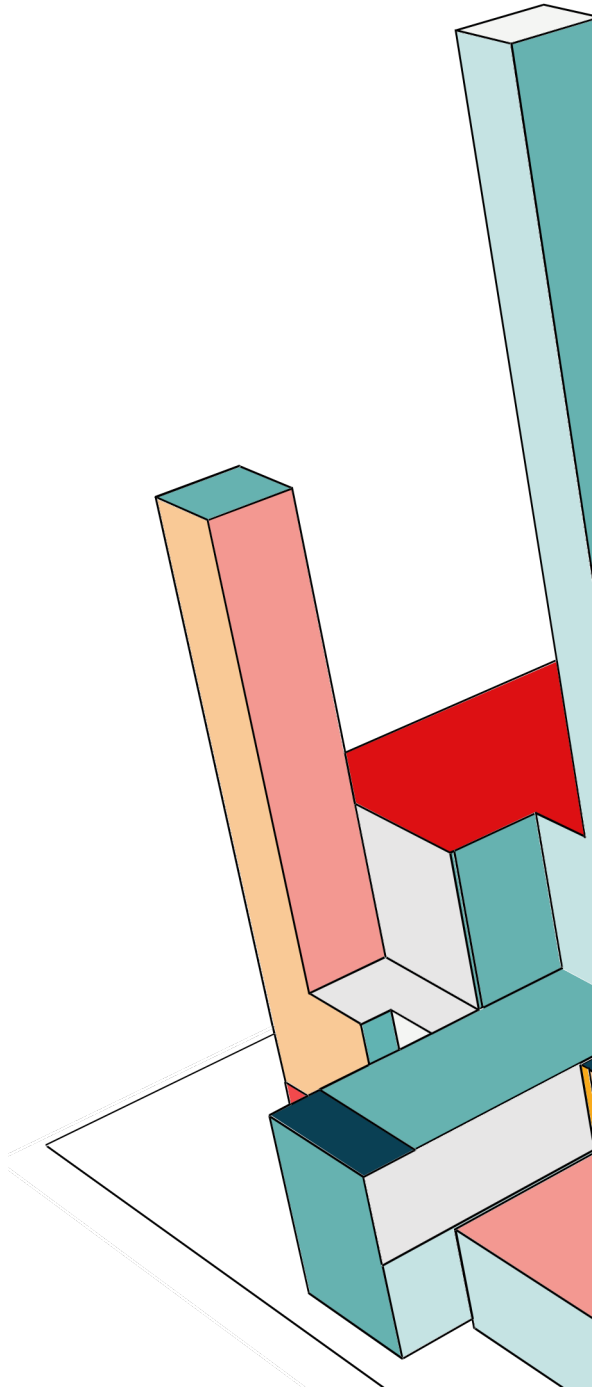
  - a programming model for execute the map and reduce functions

  - manages works distribution and fault tolerance

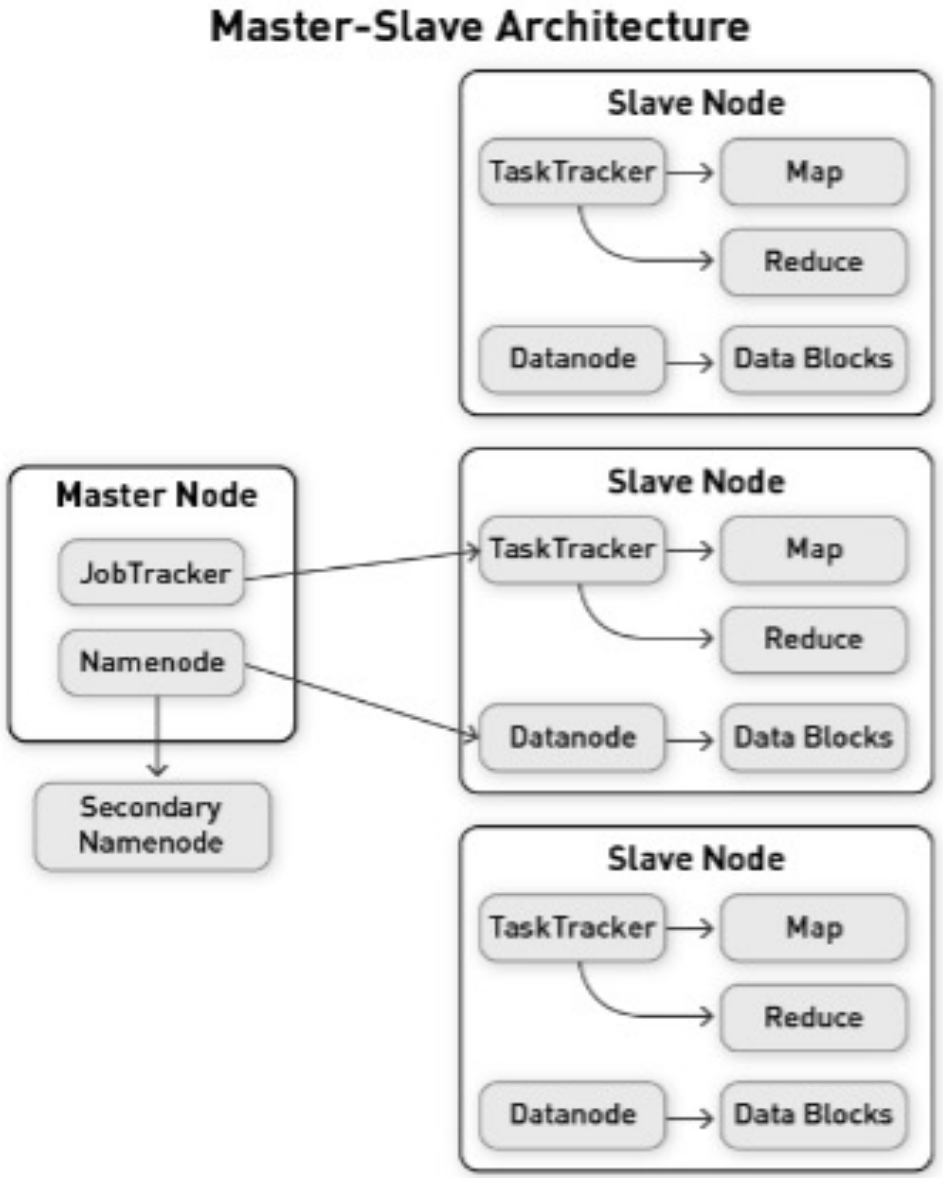


HADOOP

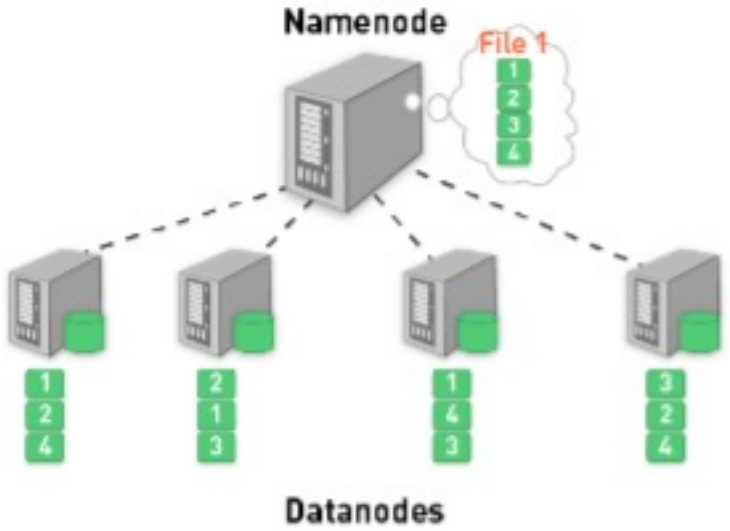
- Hadoop common utilities
- Avro, a data serialization system with scripting languages
- HDFS
- Hive for ad hoc query
- MapReduce, distributed processing on computer



HADOOP

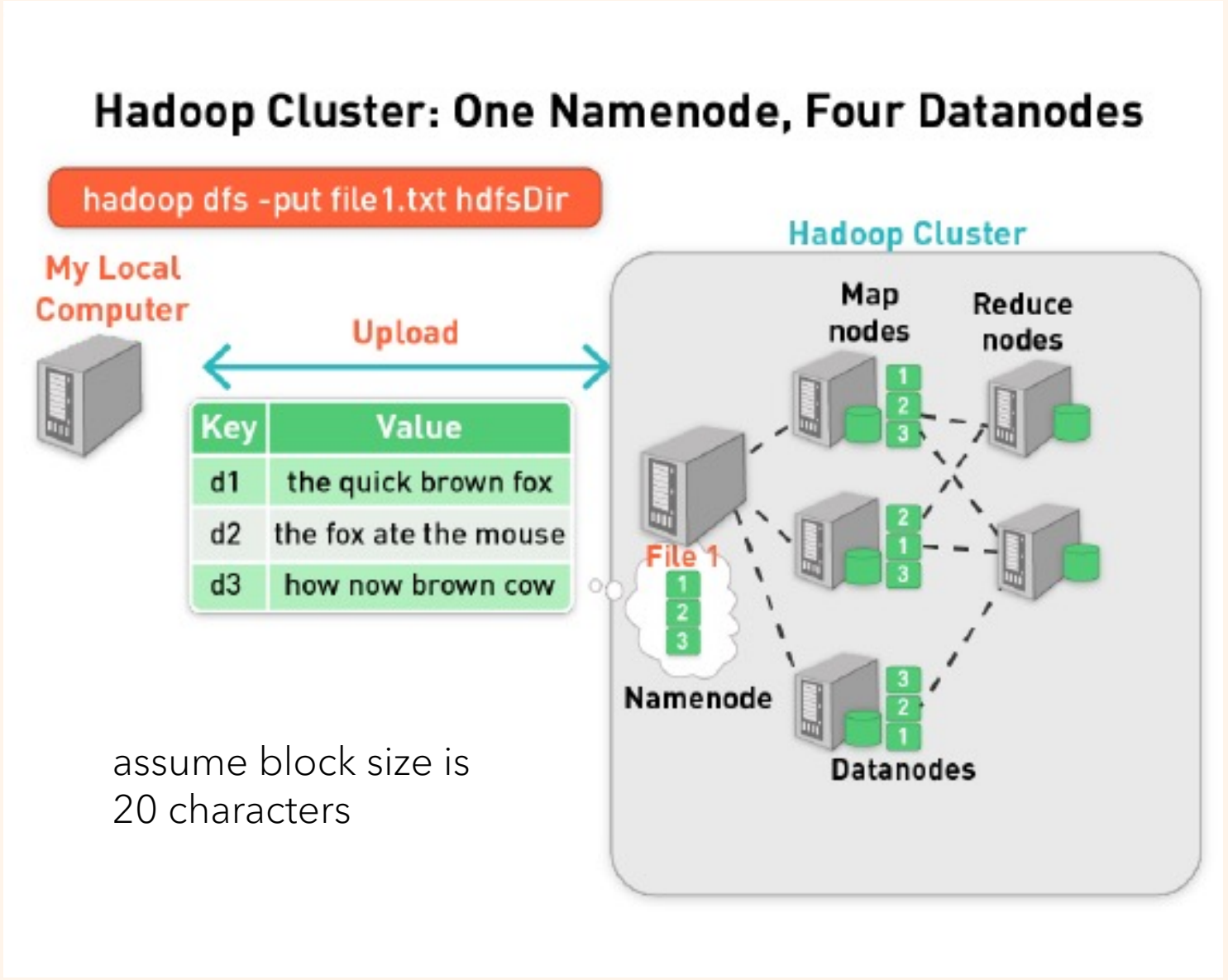

Master-Slave Architecture
Slave Node
TaskTracker
Map
Reduce
Datanode
Data Blocks
Master Node
JobTracker
Namenode
Secondary Namenode
Slave Node
TaskTracker
Map
Reduce
Datanode
Data Blocks
Slave Node
TaskTracker
Map
Reduce
Datanode
Data Blocks

HADOOP



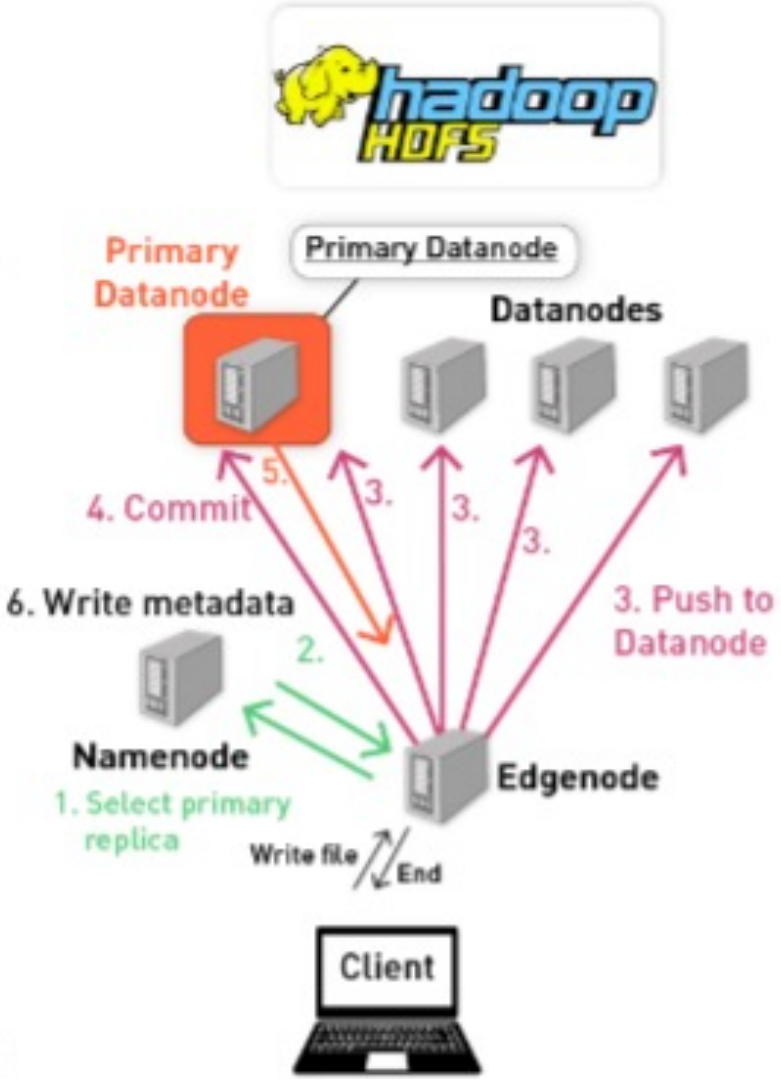
- Single **namenode**(master node) stores metadata (file names, block locations, etc.)
  - May be replicated also
- Datanodes provide redundant store for the data blocksn
- Client library provides file access
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data
    - Master node is not a bottleneck
    - Computation is done at chunk node (close to data)

HADOOP CLUSTER



Write File in HDFS

1. Client contacts the namenode, who designates one of the replicas as the primary.
2. The response of the namenode contains who is the primary and who are the secondary replicas.
3. The client pushes its changes to all datanodes in any order, but this change is stored in a buffer of each datanode.
4. The client sends a "commit" request to the primary, which determines an order to update and then pushes this order to all other secondaries.
5. After all secondaries complete the commit, the primary response is sent to the client about the success.
6. All changes of block distribution and metadata changes are written to an operation log file at the namenode.

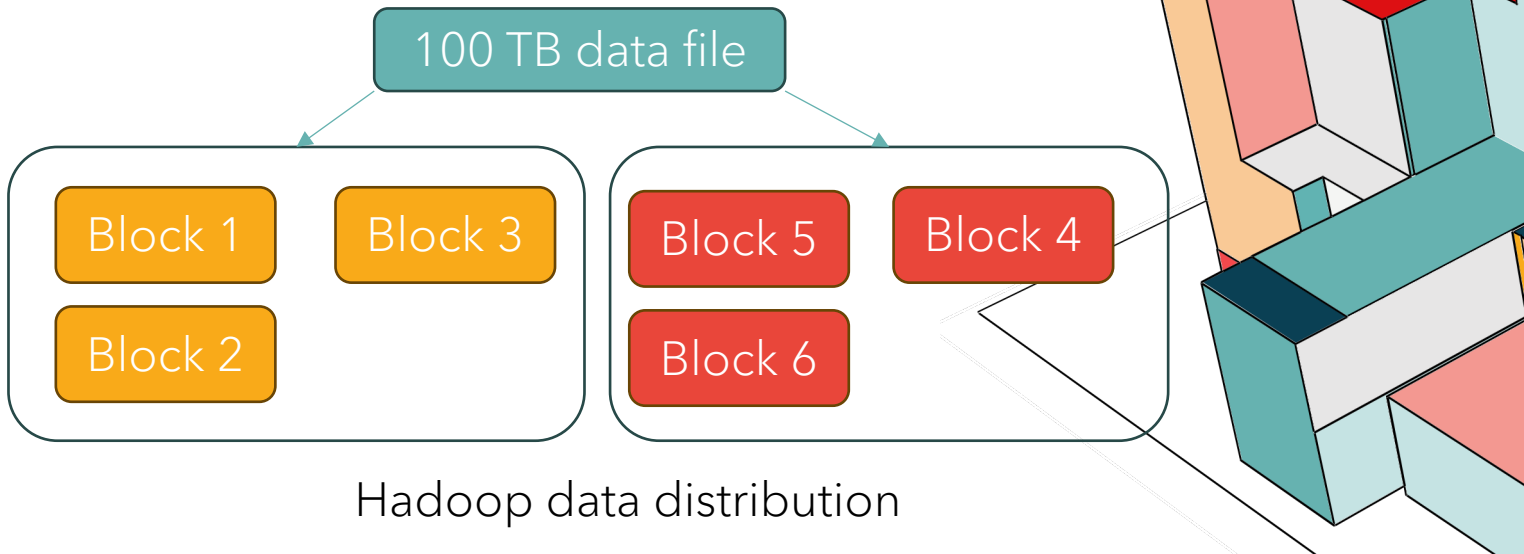


HDFS Summary

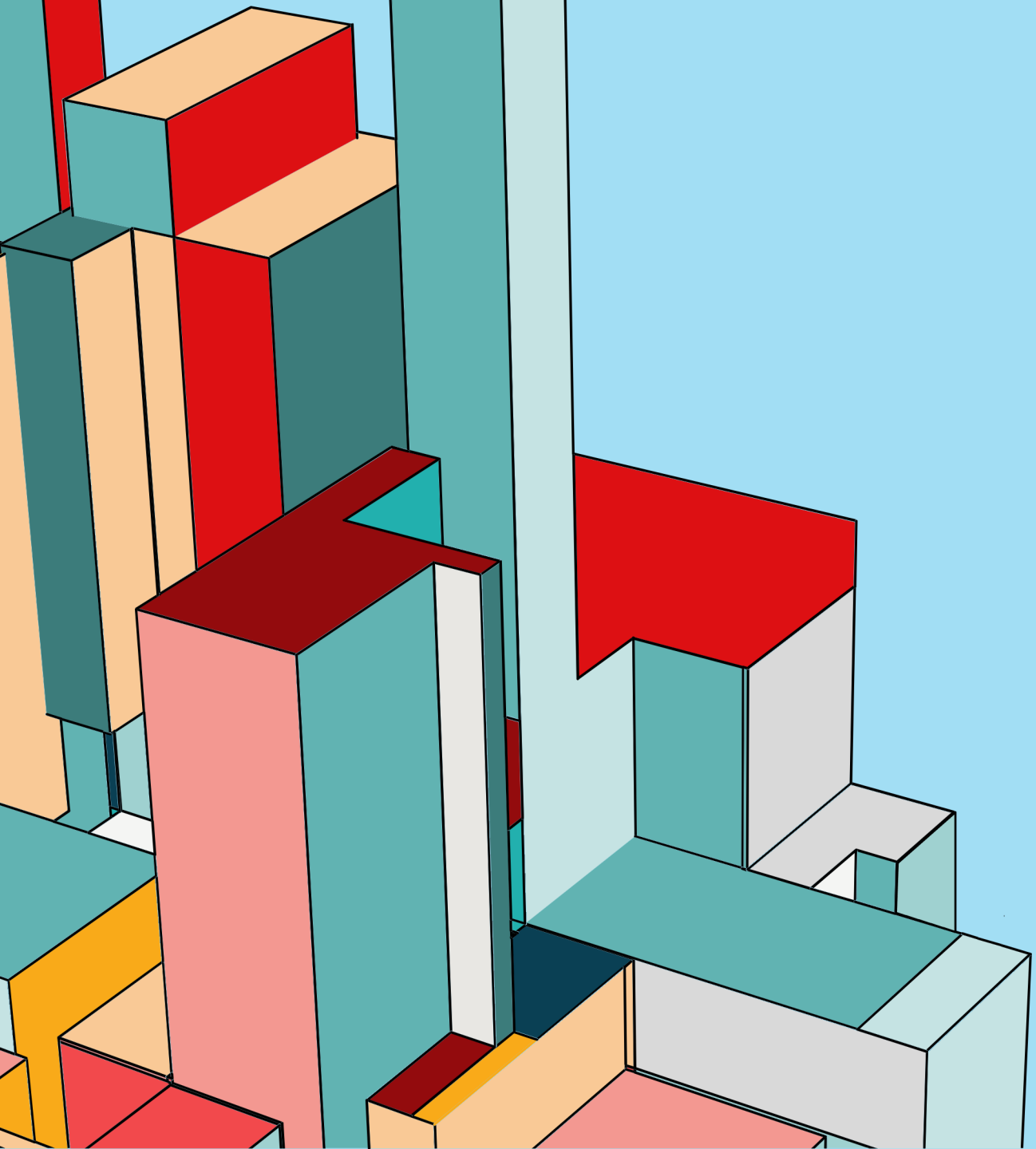
- Hadoop file system
  - Master-slave architecture
  - Organized data
  - How to get data in
  - How to get data out
- HDFS goals
  - Store large data sets
  - Deal with hardware failures
  - Emphasize streaming data access

BUILDING BLOCKS OF HADOOP

- Hadoop Distributed File System: data repo that stores all data which is processed by the Hadoop application
- MapReduce: processing program to deal with the data in HDFS
- HDFS and MapReduce provide a framework that supports for distributed storage and distributed computation within a Hadoop cluster.

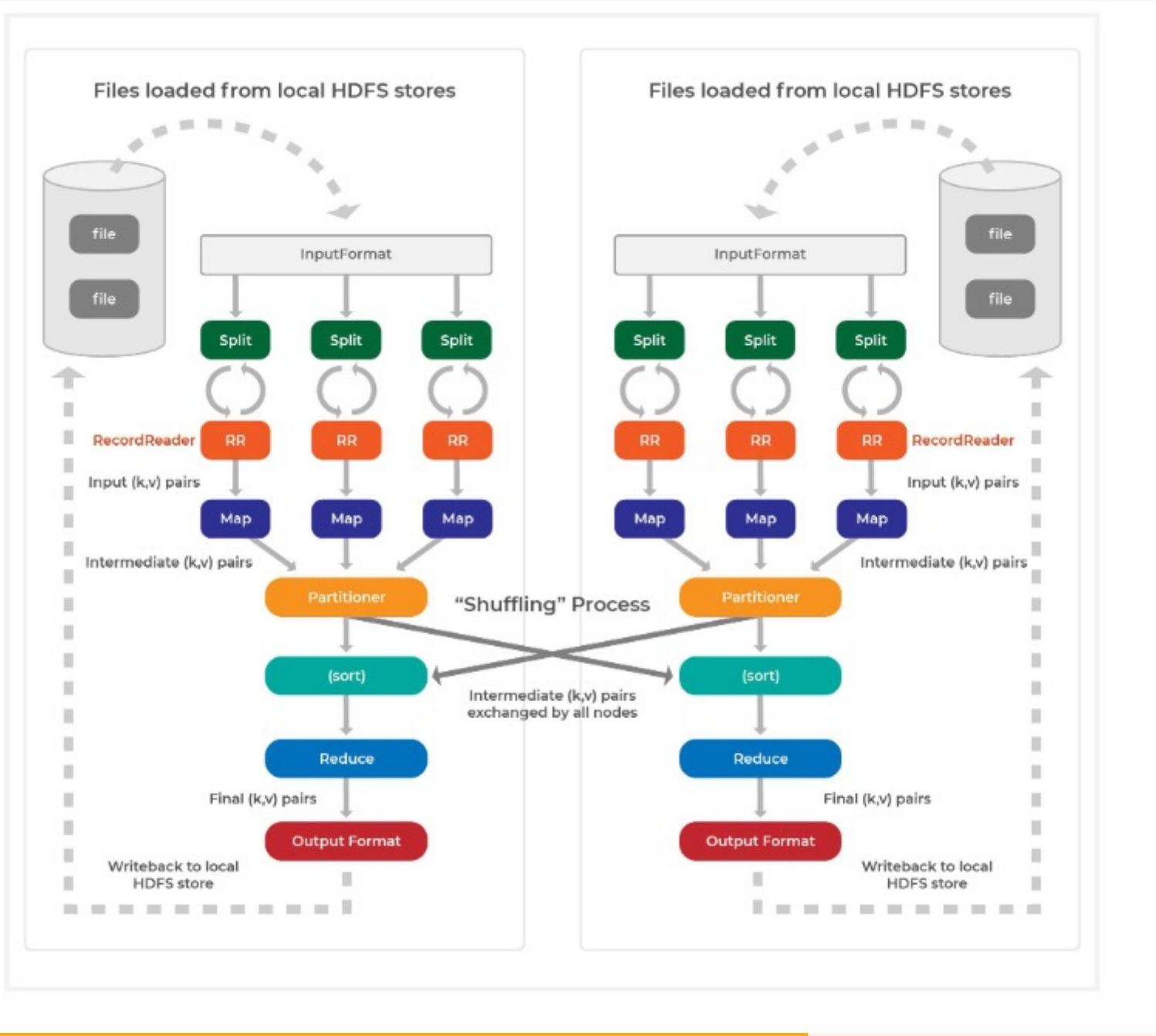


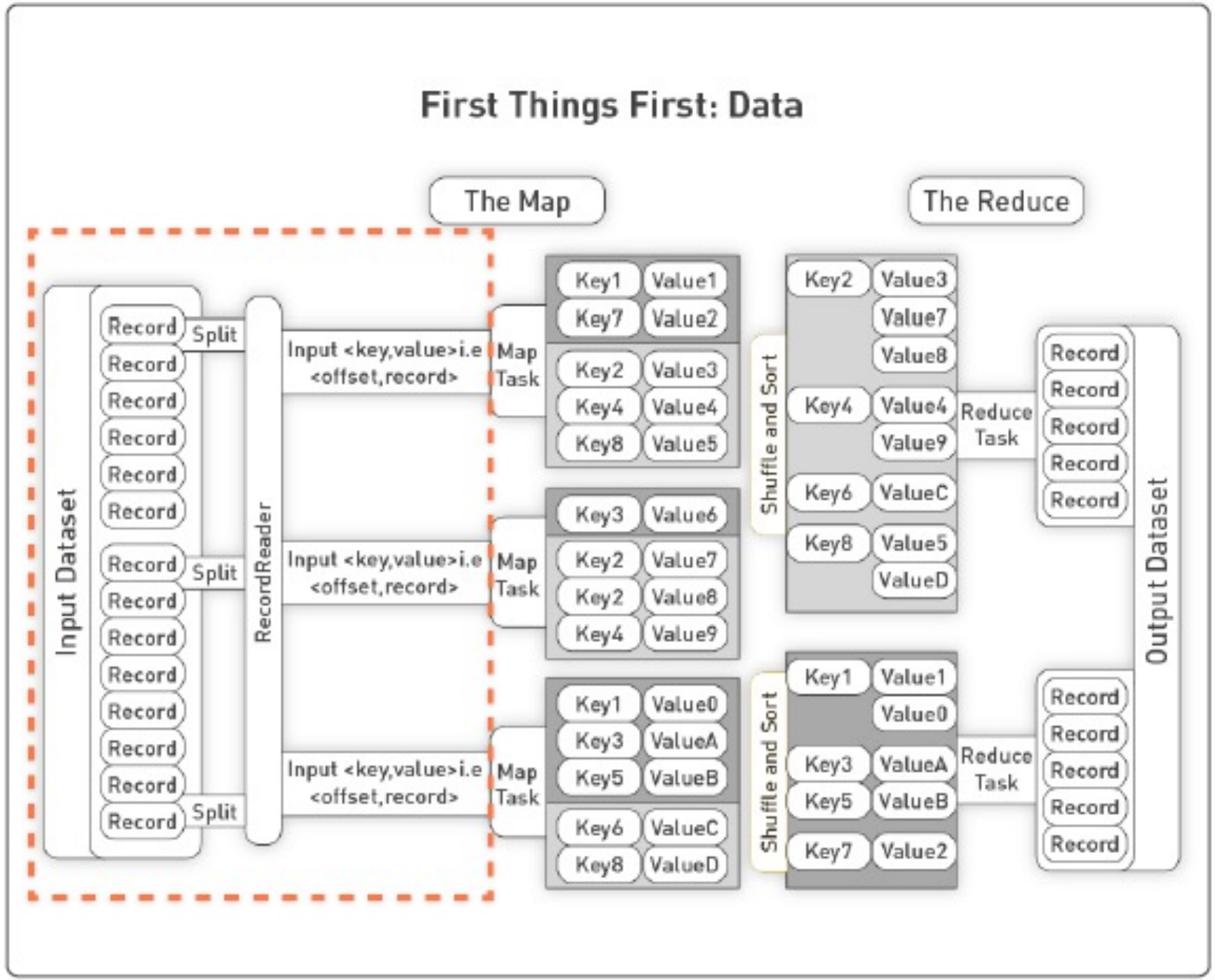
Hadoop data distribution

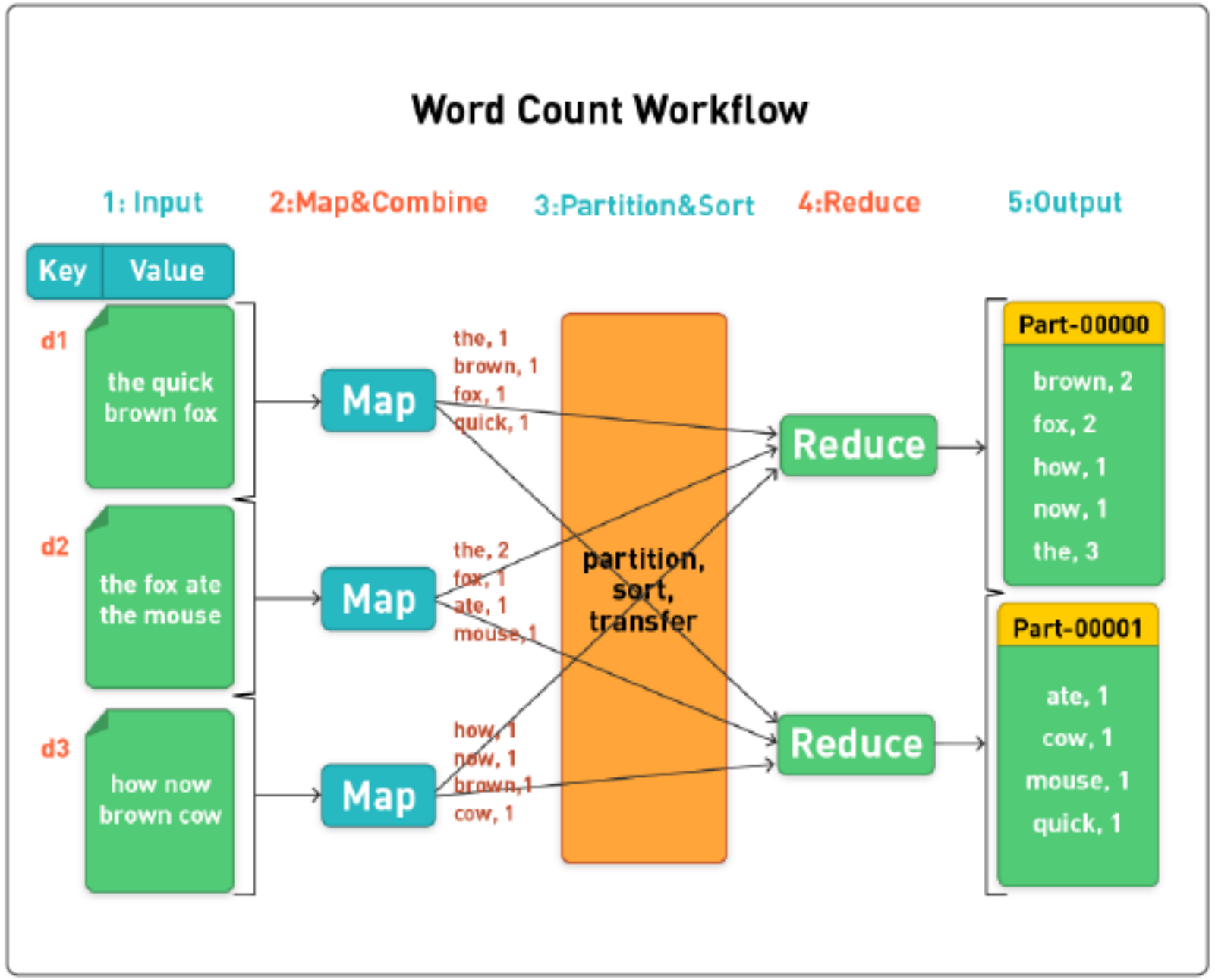


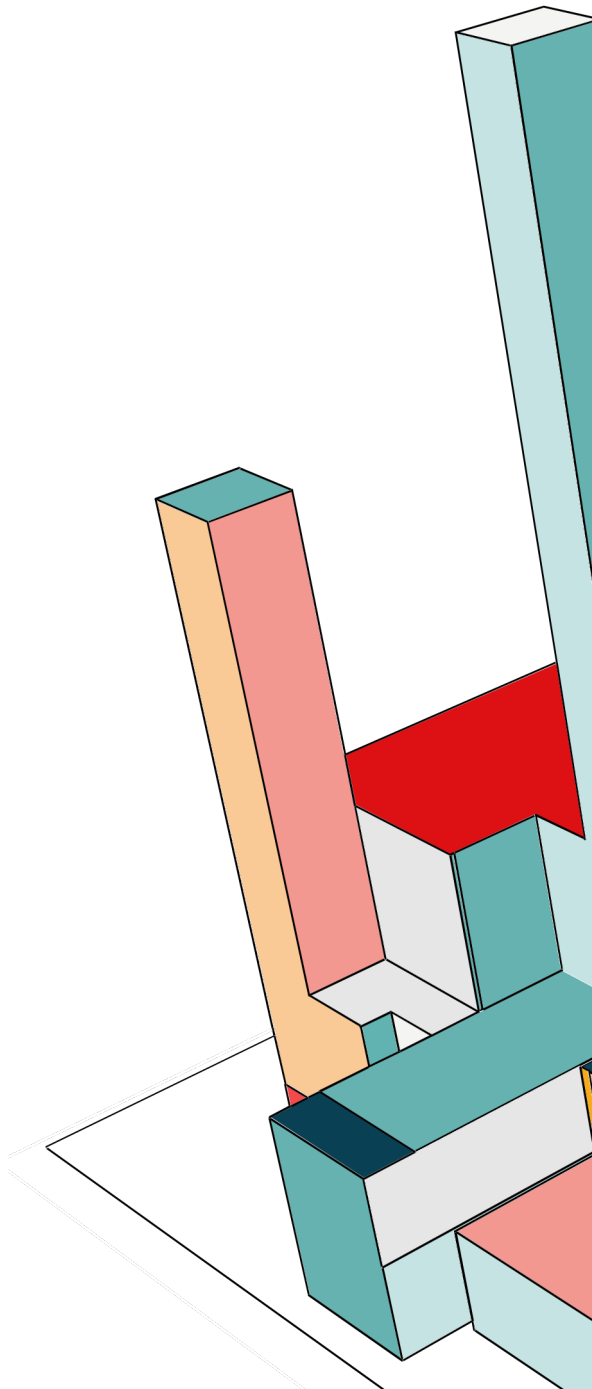
MAPREDUCE

MapReduce Framework

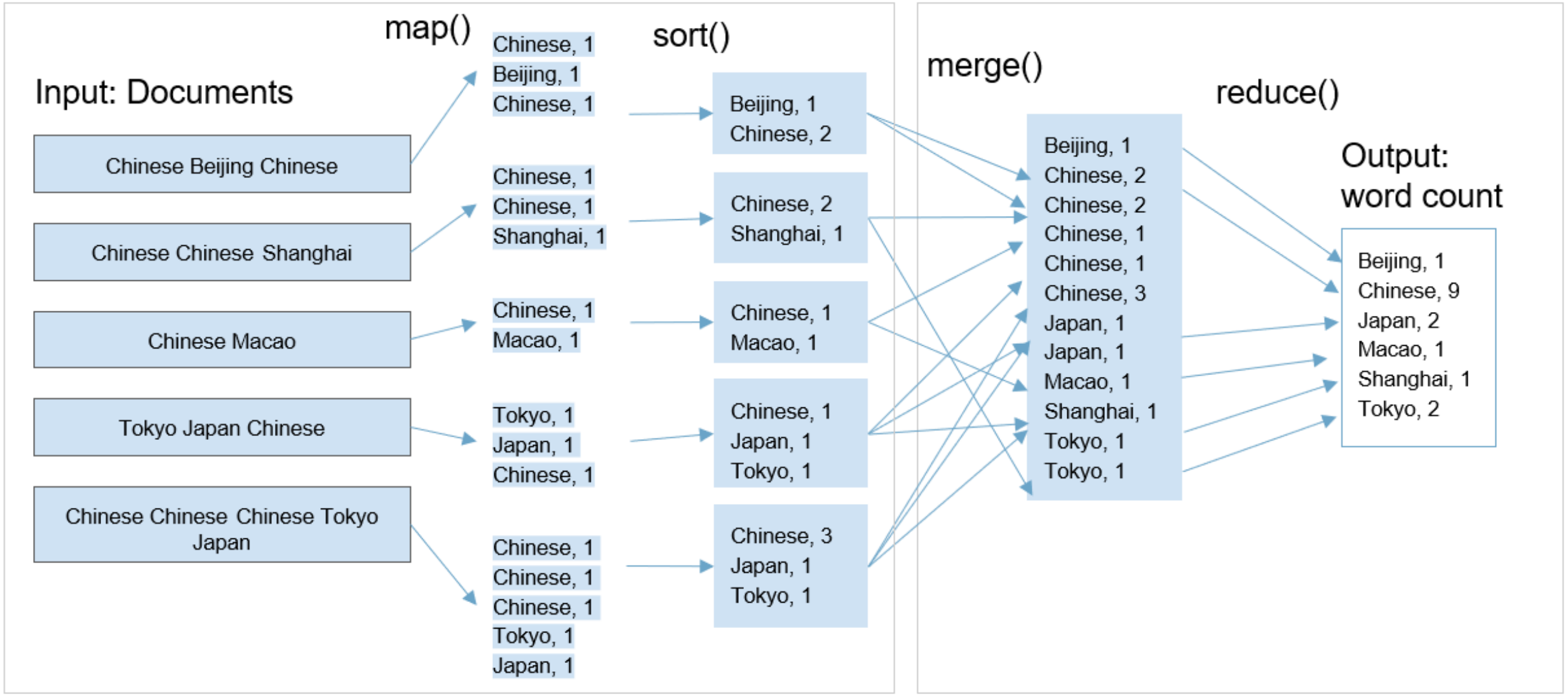



First Things First: Data
The Map
The Reduce
Input Dataset
Record
Split
RecordReader
Input <key,value>i.e <offset,record>
Map Task
Key1 Value1
Key7 Value2
Key2 Value3
Key4 Value4
Key8 Value5
Key3 Value6
Key2 Value7
Key2 Value8
Key4 Value9
Key1 Value0
Key3 ValueA
Key5 ValueB
Key6 ValueC
Key8 ValueD
Shuffle and Sort
Key2 Value3
Value7
Value8
Key4 Value4
Value9
Key6 ValueC
Key8 Value5
ValueD
Key1 Value1
Value0
Key3 ValueA
Key5 ValueB
Key7 Value2
Reduce Task
Output Dataset


Word Count Workflow
1: Input
2:Map&Combine
3:Partition&Sort
4:Reduce
5:Output
Key
Value
d1
the quick brown fox
d2
the fox ate the mouse
d3
how now brown cow
Map
Map
Map
the, 1
brown, 1
fox, 1
quick, 1
the, 2
fox, 1
ate, 1
mouse, 1
how, 1
now, 1
brown, 1
cow, 1
partition, sort, transfer
Reduce
Reduce
Part-00000
brown, 2
fox, 2
how, 1
now, 1
the, 3
Part-00001
ate, 1
cow, 1
mouse, 1
quick, 1



Single Reducer example



MAPREDUCE - KEY CONCEPTS

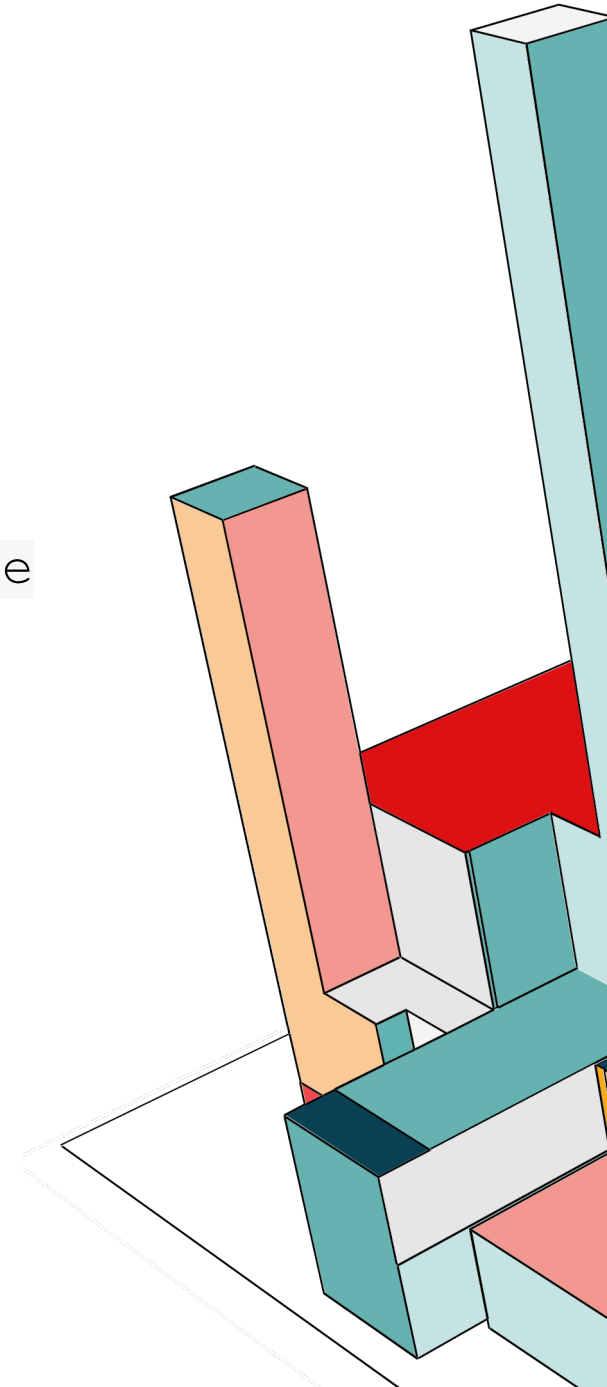
Phases of MapReduce

- Map phase
  - Each mapper takes a line as input and breaks it into words. It then emits a key/value pair of the word and 1.
- Shuffle/sort phase (*partition and sort the mapper output*)
  - The output from the mappers is sorted by key and shuffled into the reducers such that key/value pairs with the same key end up in the same reducer.
  - Sort is a simple and very useful command found in Unix systems. It rearranges lines of text numerically and/or alphabetically. (*Hadoop Streaming KeyBasedComparator is modeled after Unix sort, and utilizes command line options which are the same as Unix sort command line options*).
  - http://www.theunixschool.com/2012/08/linux-sort-command-examples.html
- Reduce phase
  - Each reducer sums the counts for each word and emits a single key/value with the word and sum.

HADOOD INSTALLATION IN COLAB

1. COPY DATA TO HDFS

```
! {hadoop_path} fs -put -f $local_txt_file $hdfs_txt_file
```



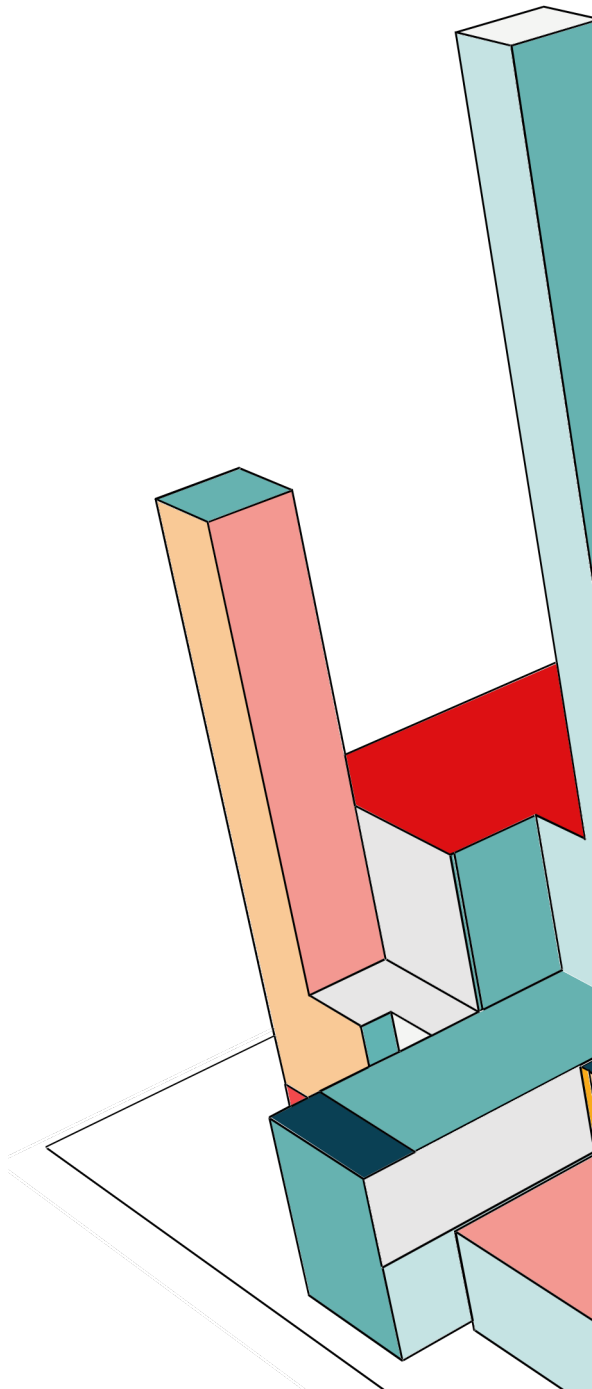
2. MAP FUNCTION

- Splitting lines to words

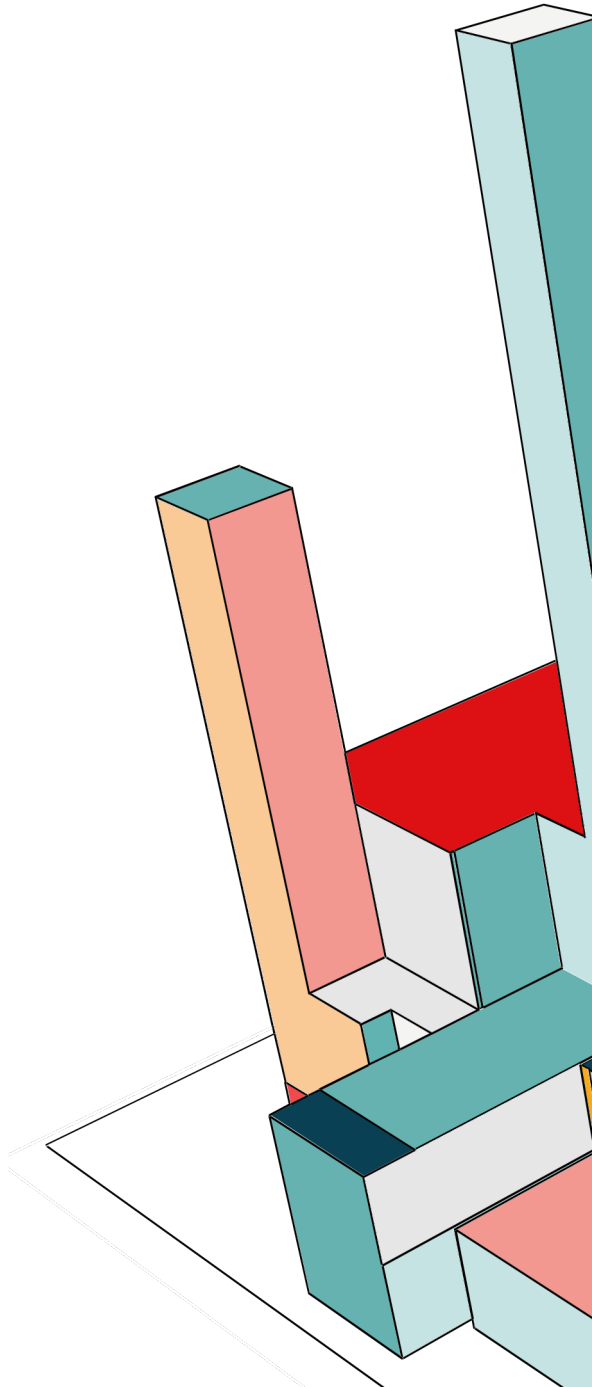
```
df.withColumn("words", f.split("value", " "))
```

- Transforming each word into a key-value pair (voc, 1)

```
df.withColumn("word", f.explode("words"))
```



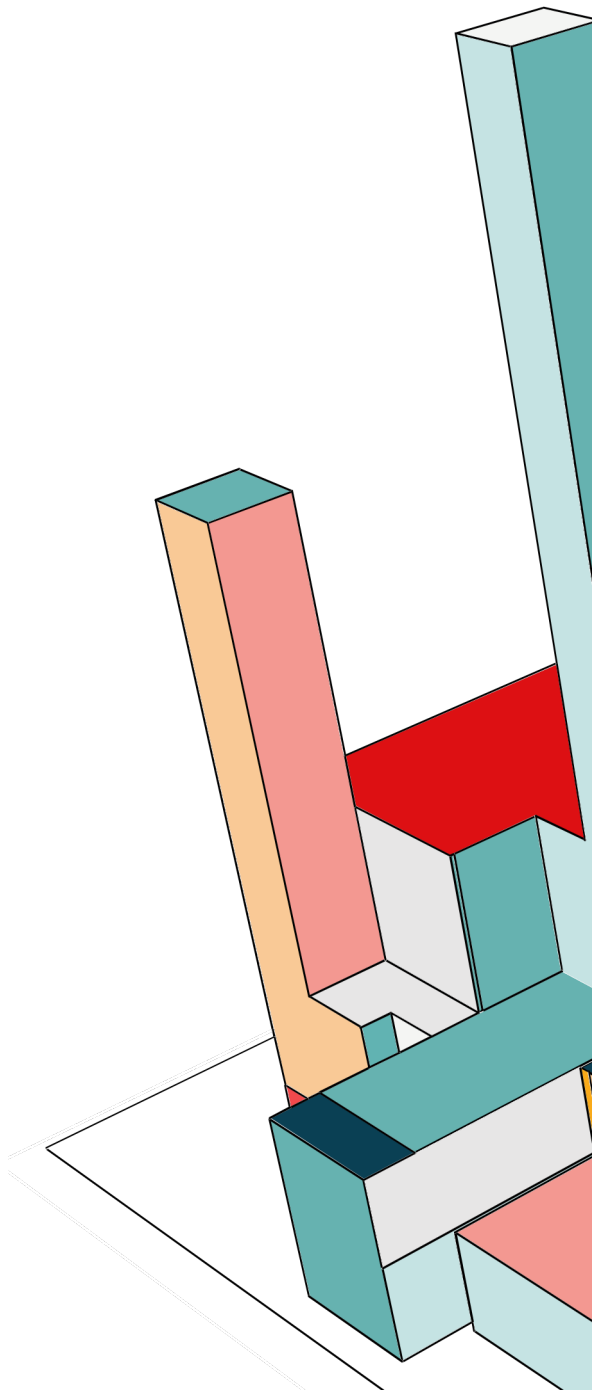
3. PARTITION AND SORT



4. SHUFFLE/AGGREGATE /REDUCE

```
.groupBy("word")
.count()
```

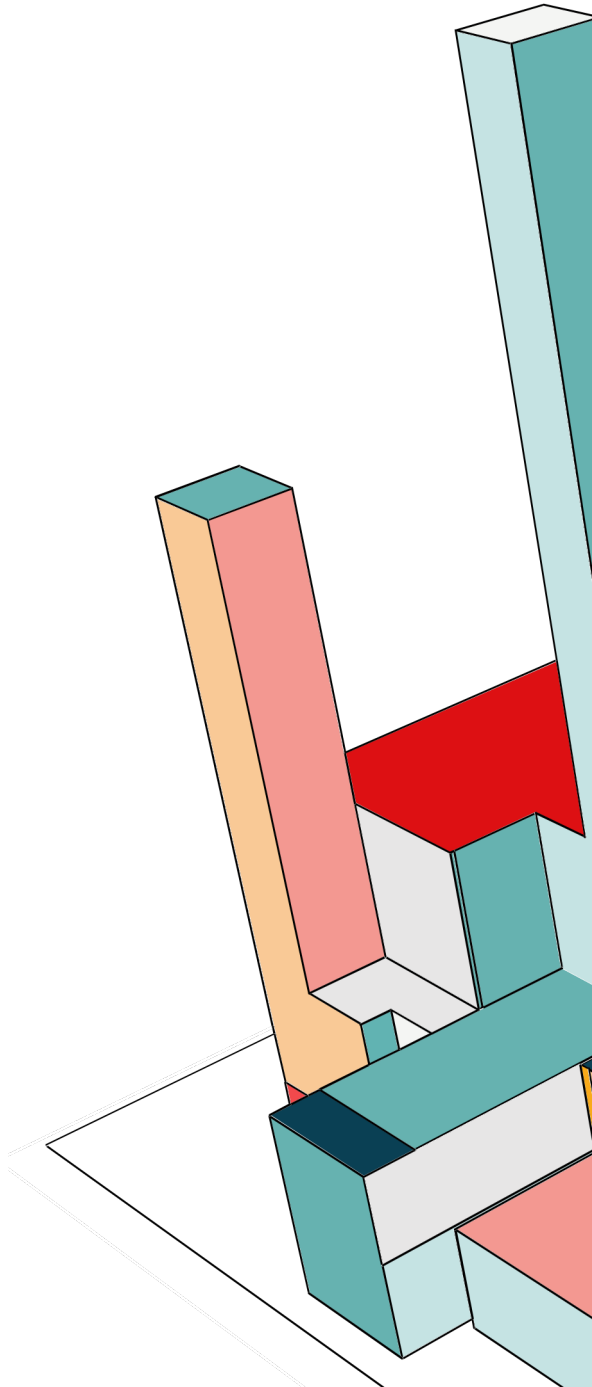
Note that groupBy() typically cause a shuffle in order to aggregate the counts for each unique word



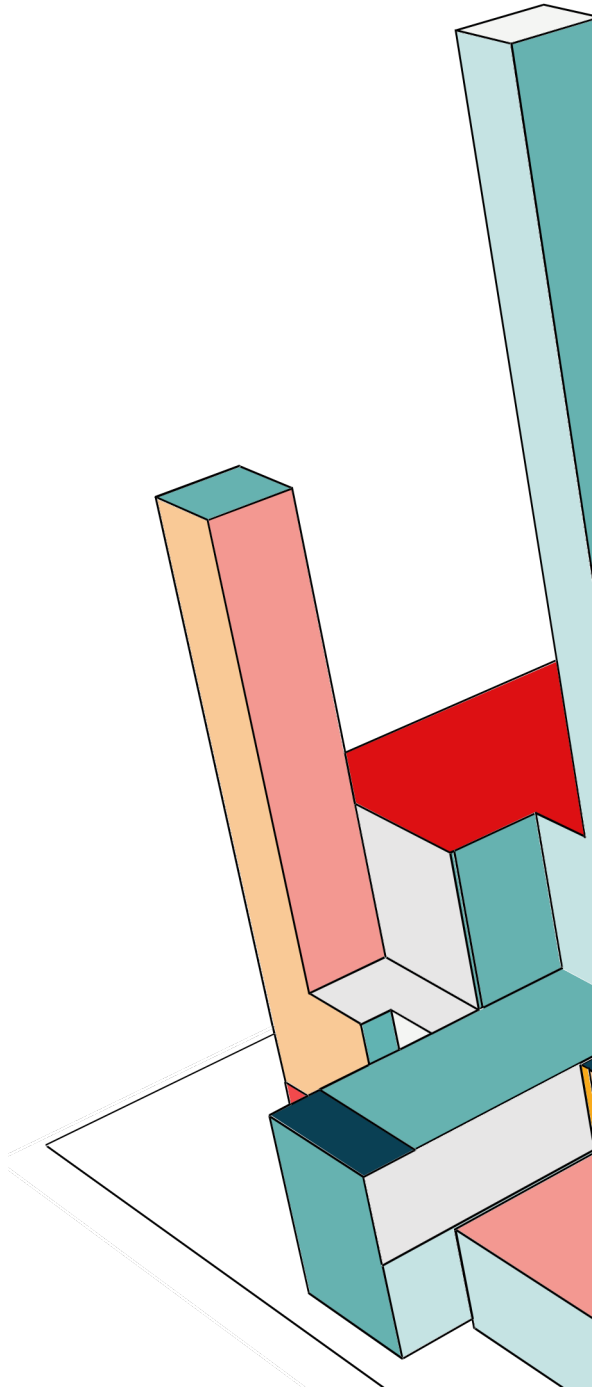
5. OUTPUT

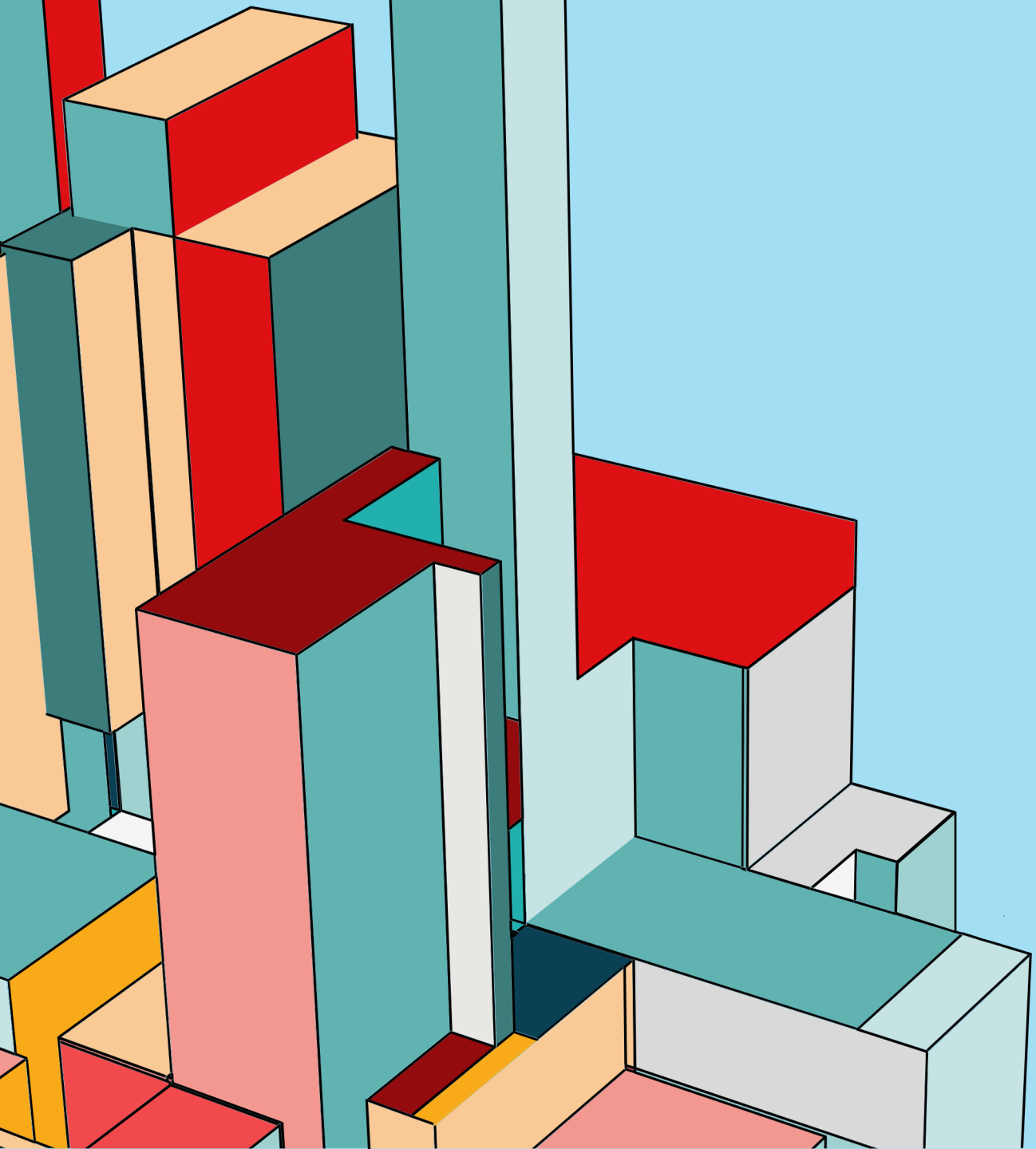
```
from pyspark.sql import SparkSession
from pyspark.sql import functions as f

# Create a SparkSession
spark = SparkSession.builder.getOrCreate()

# Clear the Spark session
spark.stop()
spark = SparkSession.builder.getOrCreate()

# Read the text file from HDFS
hdfs_txt_file = '/mydata/HW6_Kowarsch.txt'
df = spark.read.text(hdfs_txt_file)

# Perform word count
word_count = (df
        .select("value")
        .withColumn("words", f.split("value", " "))
        .select("words")
        .withColumn("word", f.explode("words"))
        .groupBy("word")
        .count()
        .orderBy("count", ascending=False))

# Display the results
word_count.show(n=20, truncate=False)
```

```
+-------+-----+
|word   |count|
+-------+-----+
|the    |76   |
|       |71   |
|of     |47   |
|to     |30   |
|is     |26   |
|a      |26   |
|in     |20   |
|The    |16   |
|that   |16   |
|be     |13   |
|and    |12   |
|privacy|12   |
|query  |10   |
|,      |9    |
|data   |9    |
|patient|9    |
|for    |8    |
|on     |8    |
|or     |8    |
|set    |7    |
+-------+-----+
```



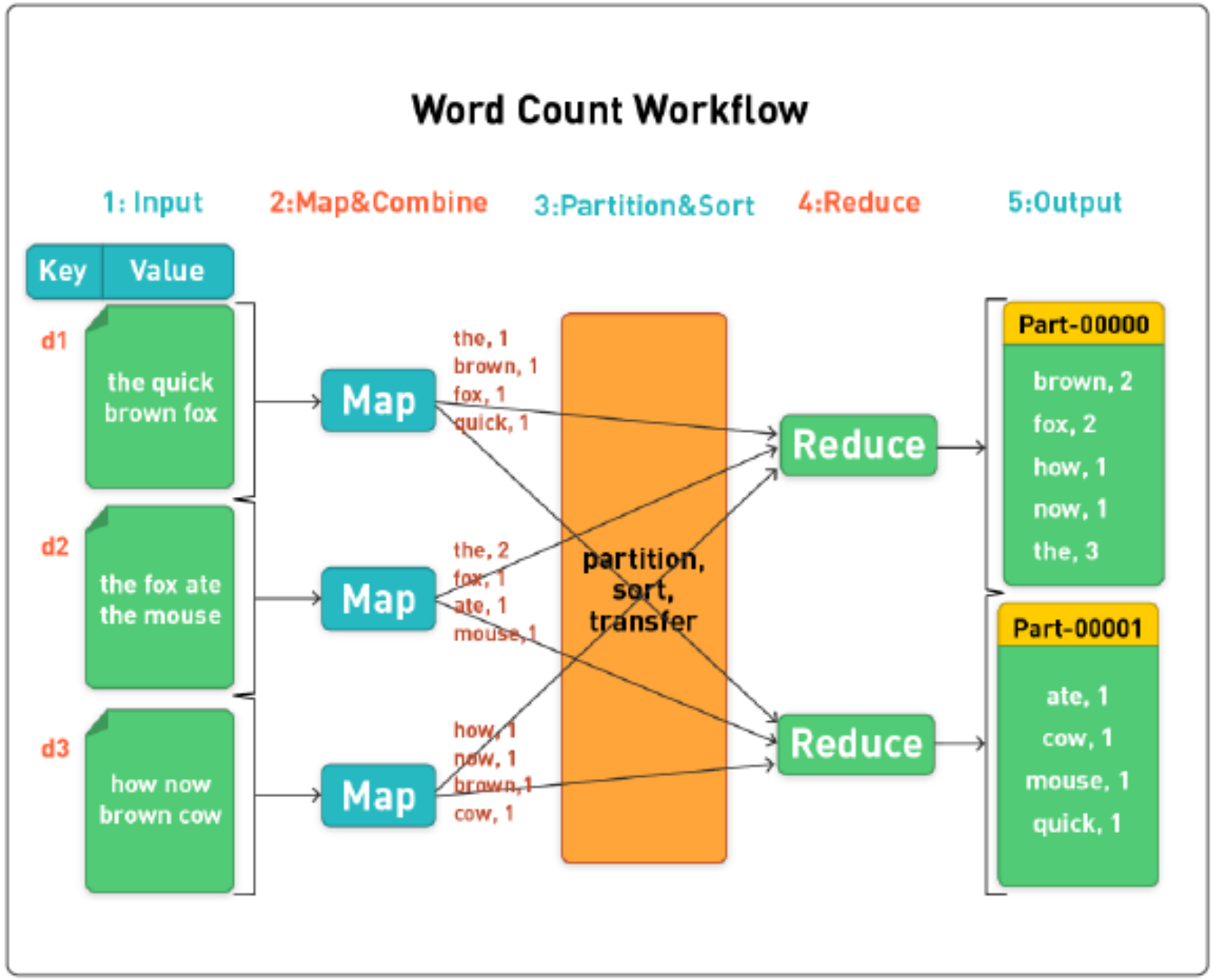
WHAT DID YOU LEARN ?

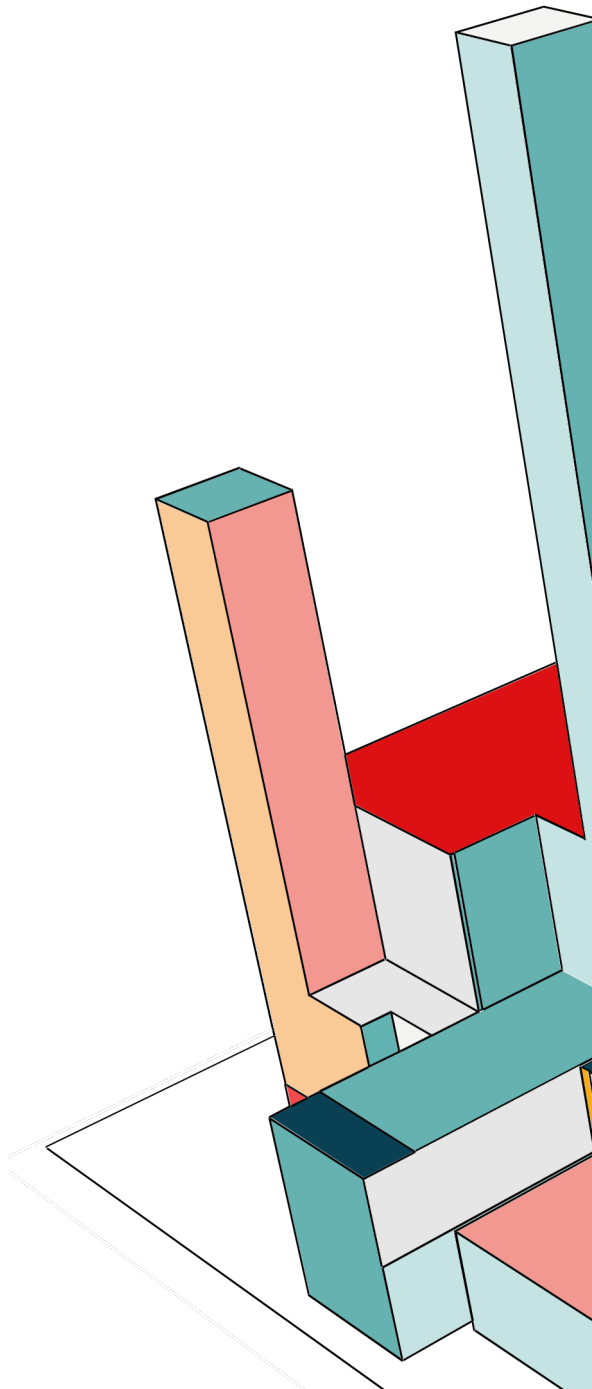




DAY 3

Advanced MR Concepts and Hadoop 2.0

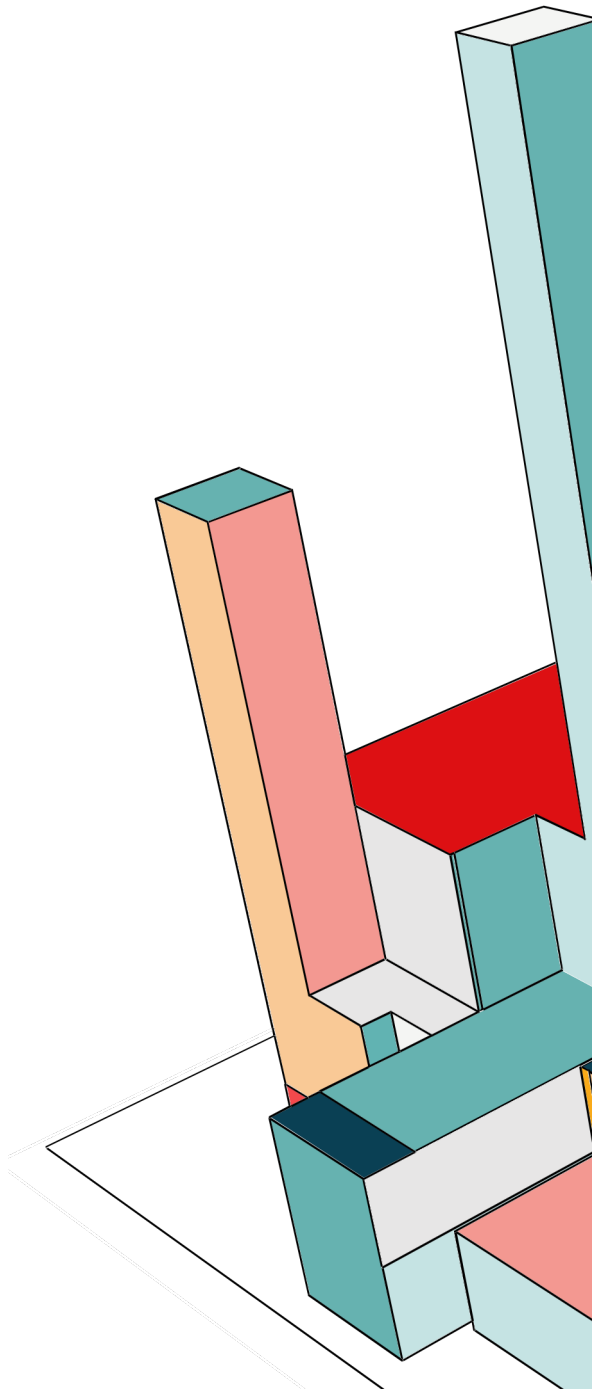

Word Count Workflow
1: Input
2:Map&Combine
3:Partition&Sort
4:Reduce
5:Output
Key
Value
d1
the quick brown fox
d2
the fox ate the mouse
d3
how now brown cow
Map
Map
Map
the, 1
brown, 1
fox, 1
quick, 1
the, 2
fox, 1
ate, 1
mouse, 1
how, 1
now, 1
brown, 1
cow, 1
partition, sort, transfer
Reduce
Reduce
Part-00000
brown, 2
fox, 2
how, 1
now, 1
the, 3
Part-00001
ate, 1
cow, 1
mouse, 1
quick, 1



- Map function is similar as what in SQL?

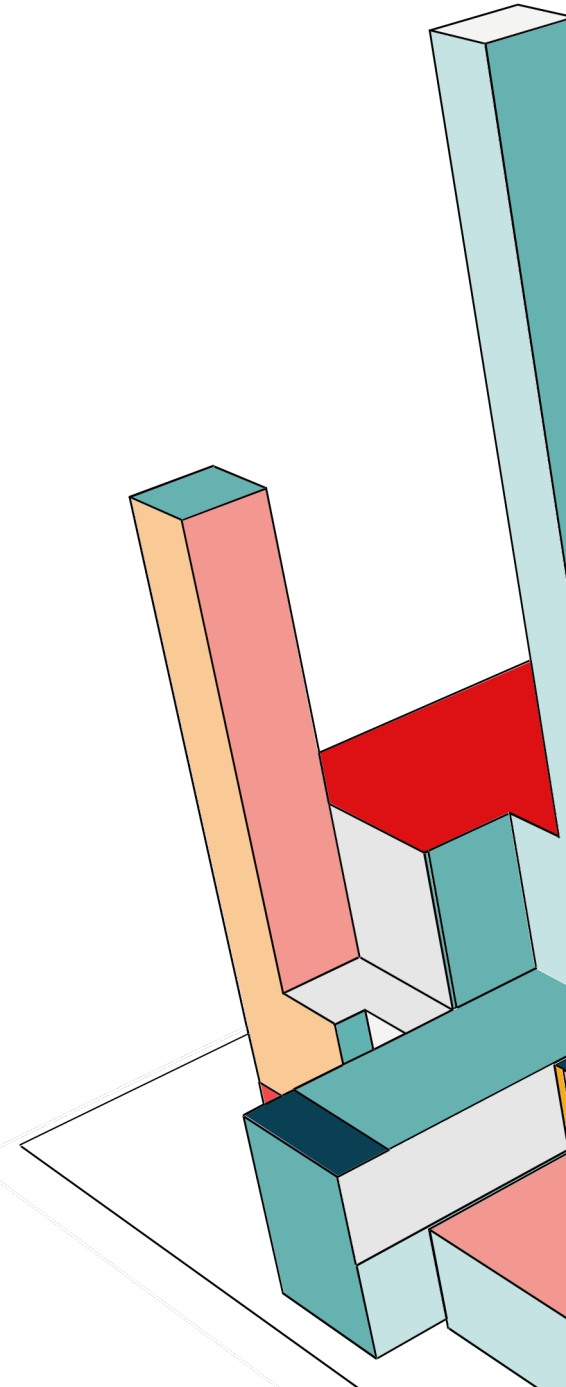
Select CASE WHEN condition then 1

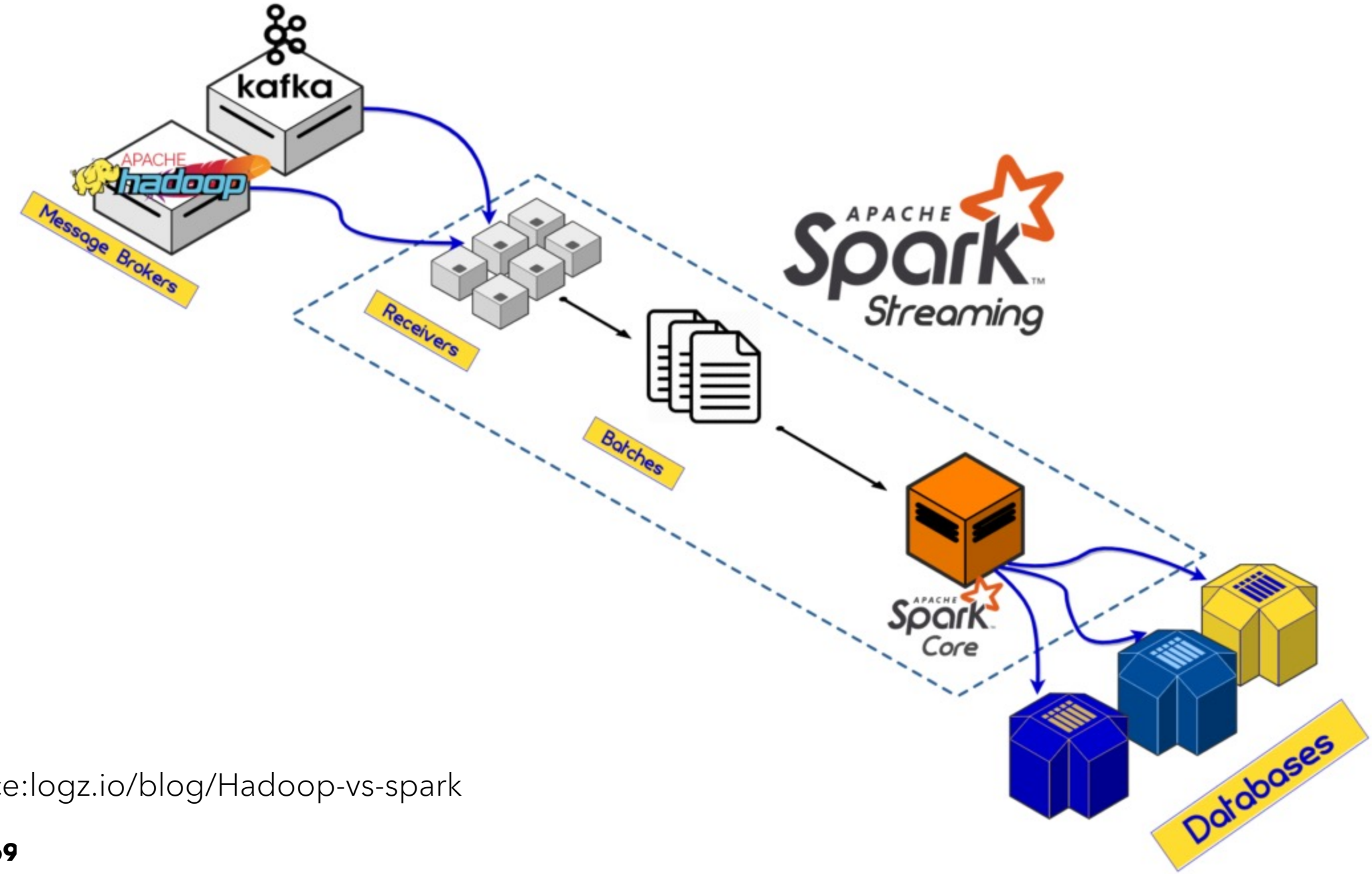
ELSE 0 AS Count_as_one



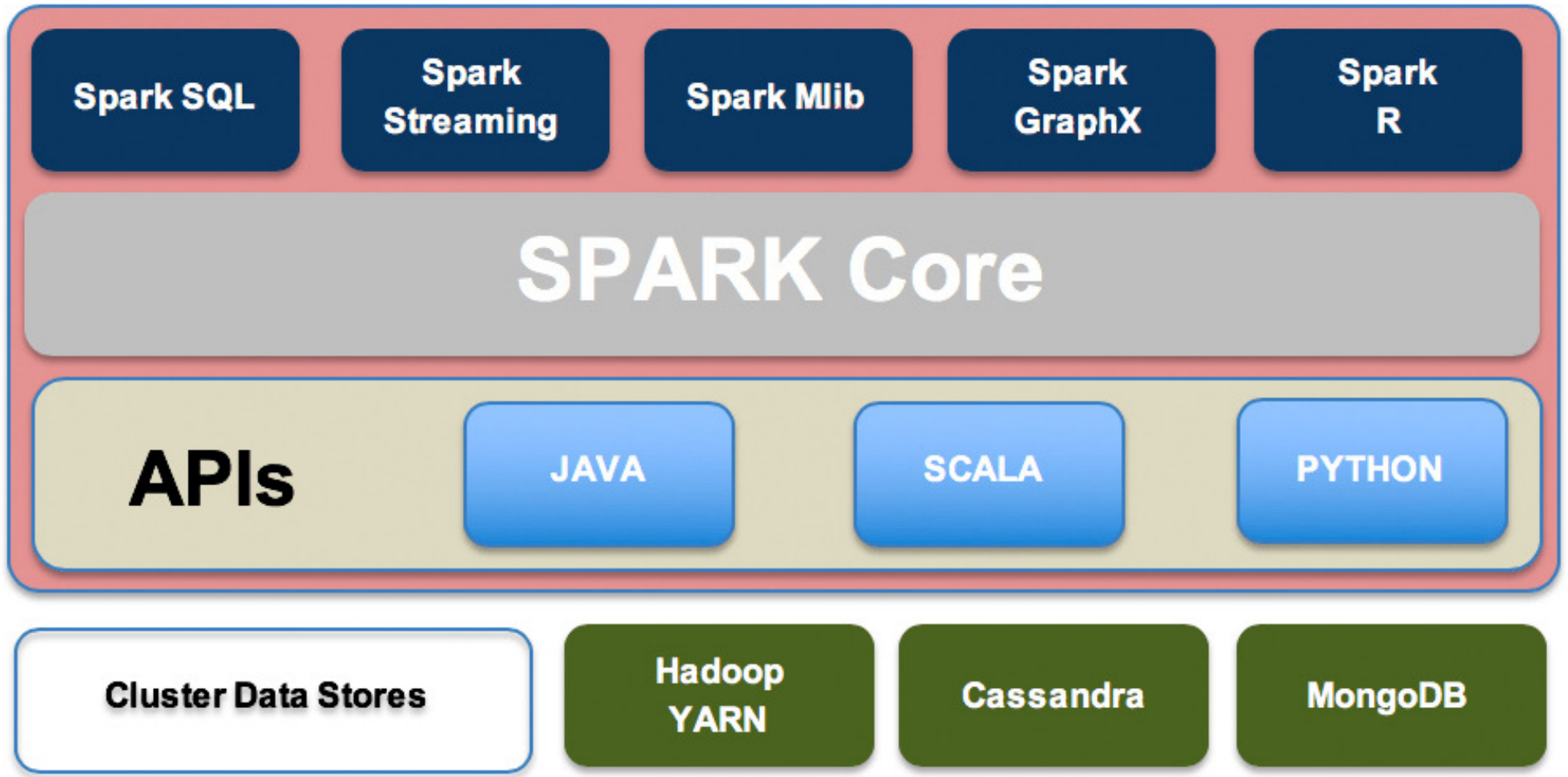
HADOOP AND SPARK

Hadoop	Spark
HDFS: a distributed file system that stores data across multiple machines.	Unified Analytics Engine: for fast computation, batch, real-time, and iterative algorithms
MapReduce: A programming model for processing large datasets with a distributed algorithm on a cluster	In-Memory Processing
Hadoop Ecosystem: Hive (SQL queries), Pig (data flow), Hbase, etc	Ease of Use APIs for Java, Scala, Python, and R

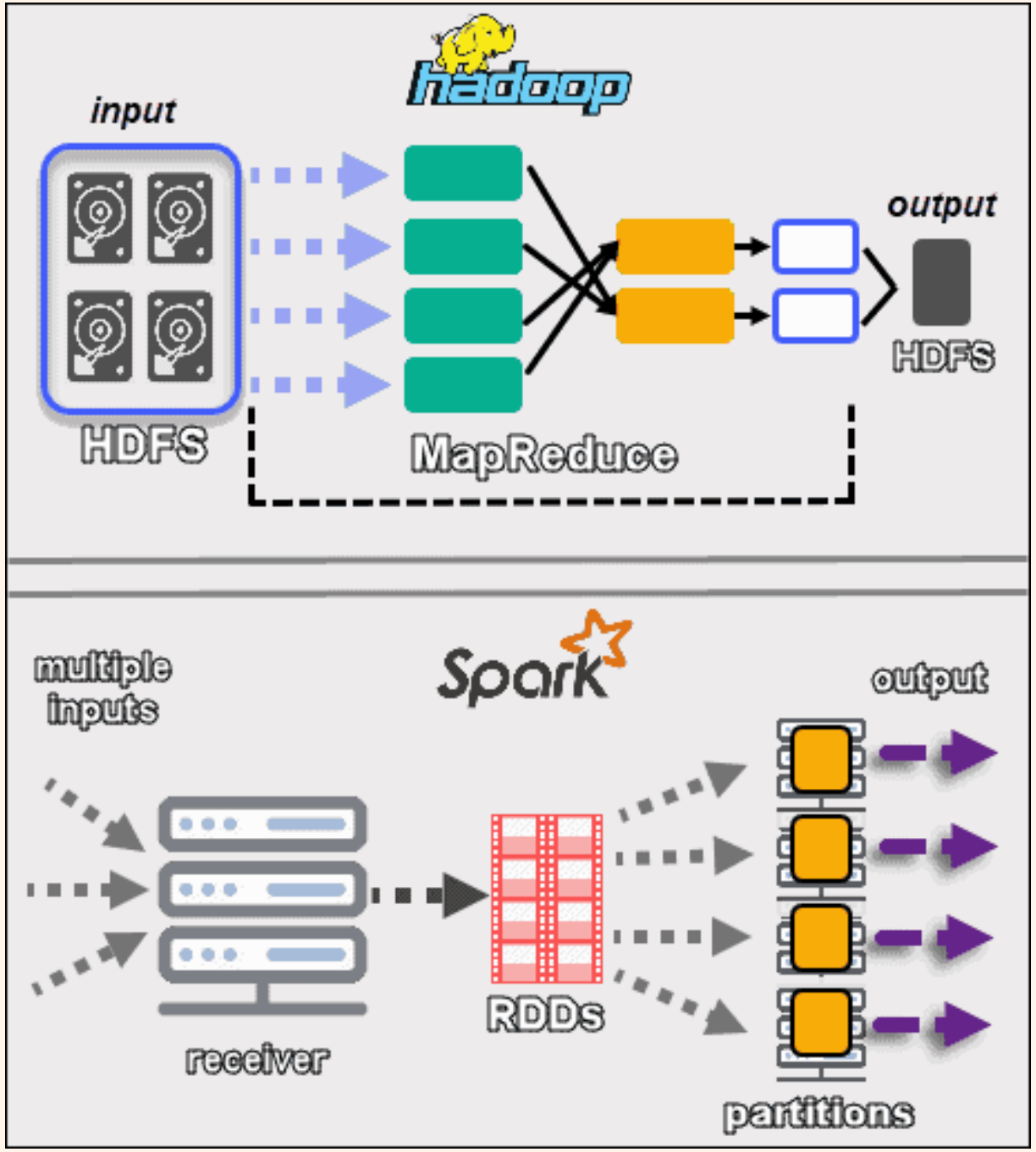




Source:logz.io/blog/Hadoop-vs-spark

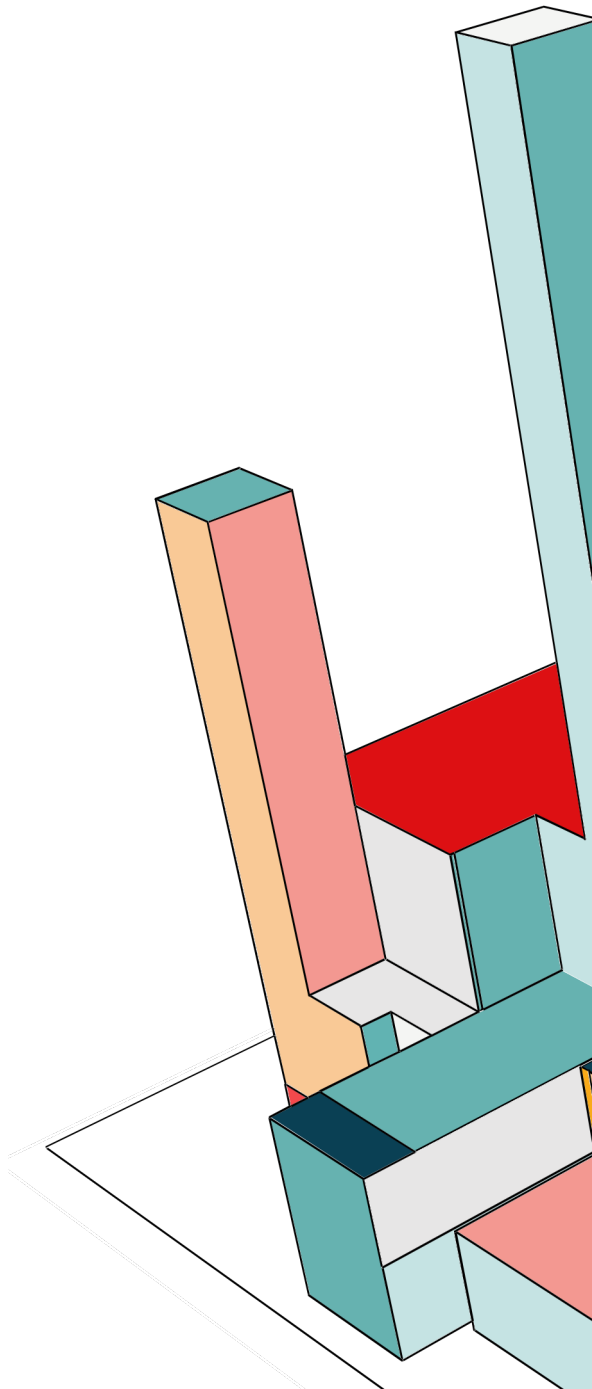


Source:logz.io/blog/Hadoop-vs-spark



[Hadoop vs Spark: Detailed Comparison of Big Data Frameworks (phoenixnap.com)]

This is crucial for ensuring that all values associated with the same key are sent to the same reducer.
By default, a hash value for each key is computed and assigns the key-value pair to a reducer
A combiner function is similar to a reducer but operates on the output of the map function before it is shuffled and sent to the reducers.



WHAT DID YOU LEARN ?