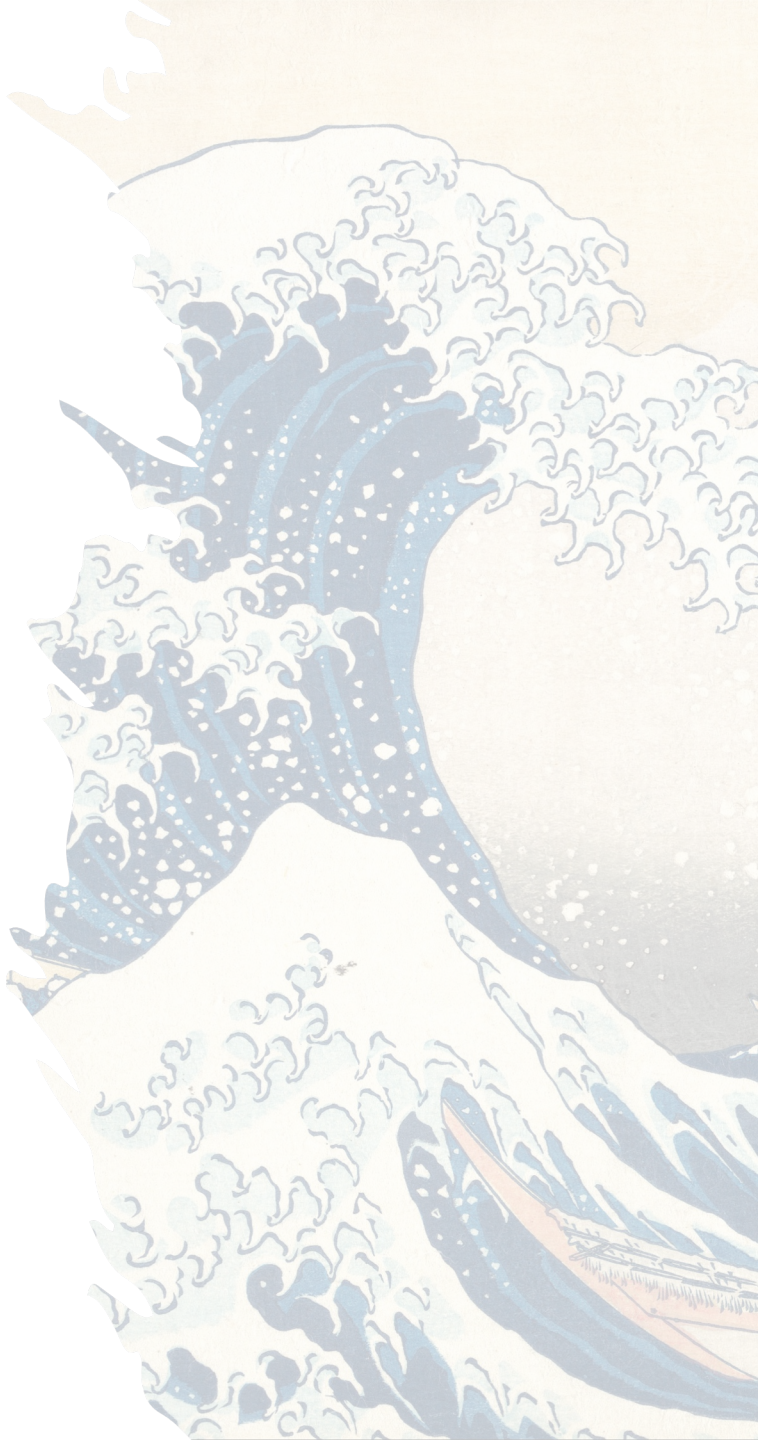
Stelios Sotiriadis

8. Distributed data processing for Big Data

Agenda

- What is distributed data processing?
- Hadoop MapReduce ecosystem
- Deploy Hadoop on a Docker container in the Google Cloud Platform
  - Run a simple Hadoop job.



Think about the following…

1. What is the ”big data” problem?

   Disk read speed!

2. What is an example of an application with the “biggest data” you can imagine?

   Search engine…

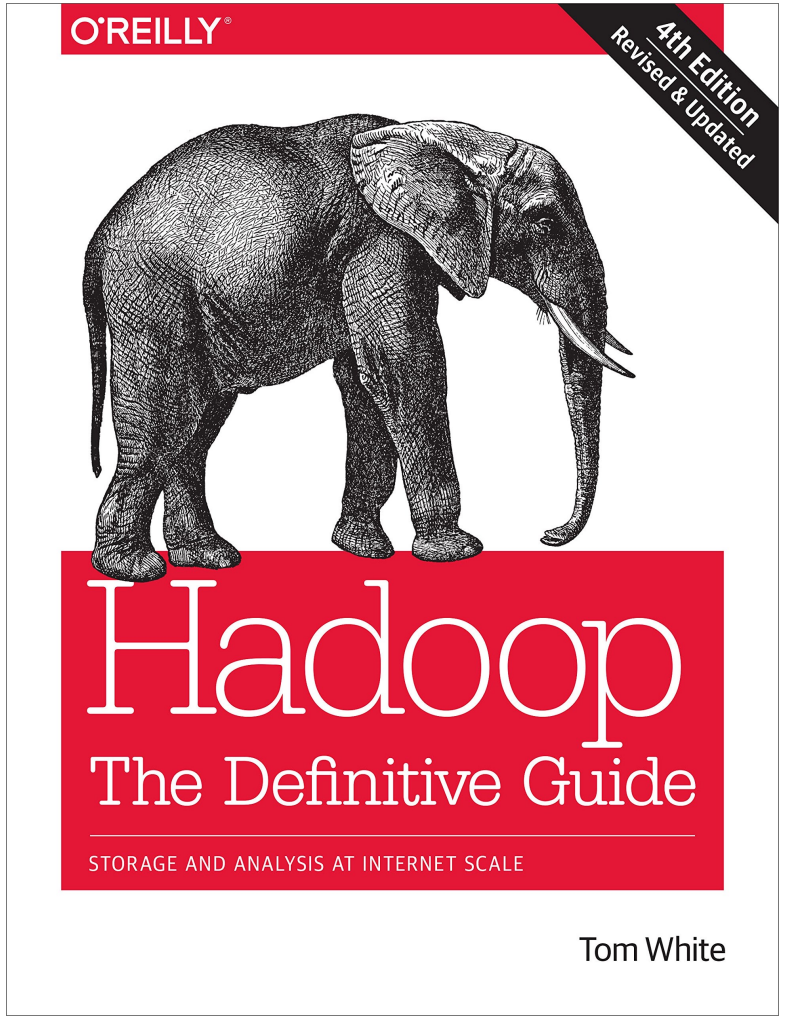
3. Is “big data” about “starting data”?

   No! It’s also about processing!

Big Data systems

Today's class is on Hadoop

- Today we will talk about:
  - Big data systems
  - The Hadoop MapReduce framework
- Today's lecture is based on:
  - Hadoop, The Definitive Guide, 4th edition by Tom White



Why do we need big data systems? (1)

- We live in the data age! Data footprint grows in huge numbers.
  - The digital universe size is **44 zettabytes** (estimation for 2022).
    - 1 zettabyte = 10^{21} bytes
    - **More than one disk drive per person in the world**.
- Google processes over 20 petabytes of data daily through an average of 100,000 MapReduce jobs spread across its massive computing clusters.

Why do we need big data systems? (2)

- There are 63,000 search queries every second (today)
  - Google used MapReduce until 2019 (Sept), then they moved to Google DataFlow
  - Dataflow brings in a whole new programming model, but is still using the traditional MapReduce system to do the actual computation
    - It is a MapReduce optimization model

What are the technical problem?

- Storage capacities of hard drives have increased massively over the years
- However, access speeds have yet to keep up!
  - The rate at which data can be read from drives
- **Let us see an example:**
  - A drive from 1990 could store 1.3 GB of data with a transfer speed of 4.4 MB/s
  - **You could read all the data from a full drive in around five minutes**
- A drive today can store 1 TB (the norm), with a transfer speed of around 100 MB/s
  - **You need two and a half hours to read all the data off the disk!**

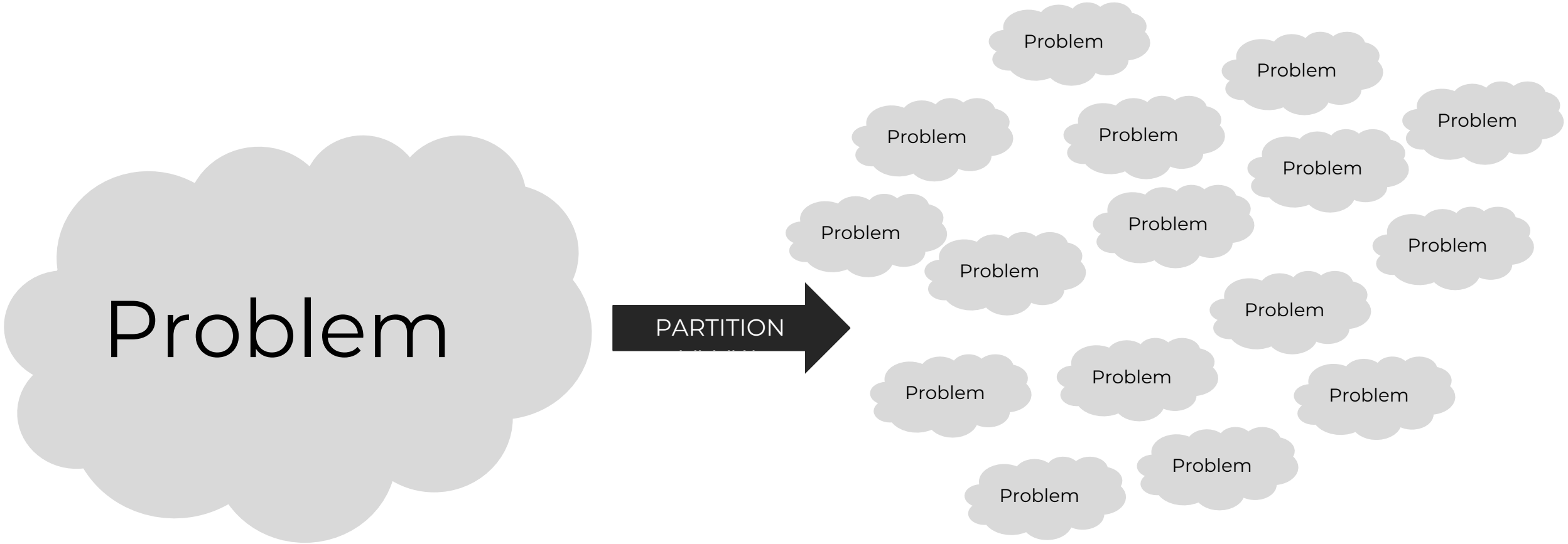
How to solve it?

- Problem:
  - Long times to read all data on a single drive and writing is even slower!
- The obvious way to reduce the time:
  - Read from multiple disks at once
- Let assume that we have 100 drives, each holding one hundredth of the data.
- Working in parallel: **We could read the data in under two minutes!**

Still, we have problems…

- Reading and writing data in parallel to or from multiple disks is challenging
- **1st Problem to solve:**
  - Hardware failure:
    - As soon as you start using many pieces of hardware, the chance that one will fail is fairly high.
- **2nd Problem to solve:**
  - Most analysis tasks need to be able to combine the data in some way from many disks
    - Data read from one disk may need to be combined with the data from another disk
- This is Hadoop about, a reliable shared storage and analysis system for **Big Data**

Big Data on Hadoop is about divide and conquer!



To solve a big problem decompose it to small partitions…

- Domain decomposition is about **DATA**:
  - Dividing the data into lots of small and (if possible) equal-sized partitions
    - Weather data from all over the world since 1900: Split into small files
- Functional decomposition is about **TASKS**:
  - Dividing the computational work into tasks that perform different portions of the overall work.
    - Find max temperature per year:
      - Process each file in parallel, and finally aggregate results…

How to decompose data?

- To answer this question, we need to understand the concept of “**amount of structure in a dataset**”.
- Structured data:
  - Data that is organised into entities that have a defined format (RDBMS)
- Semi-structured data:
  - Data is looser, and though there may be a schema and it is often ignored (may be used only as a guide)
  - Similarly JSON data
- Unstructured data:
  - Does not have any particular internal structure: plain text, logs, emails or image data.
- **Hadoop MapReduce works well on unstructured or semi-structured data.**

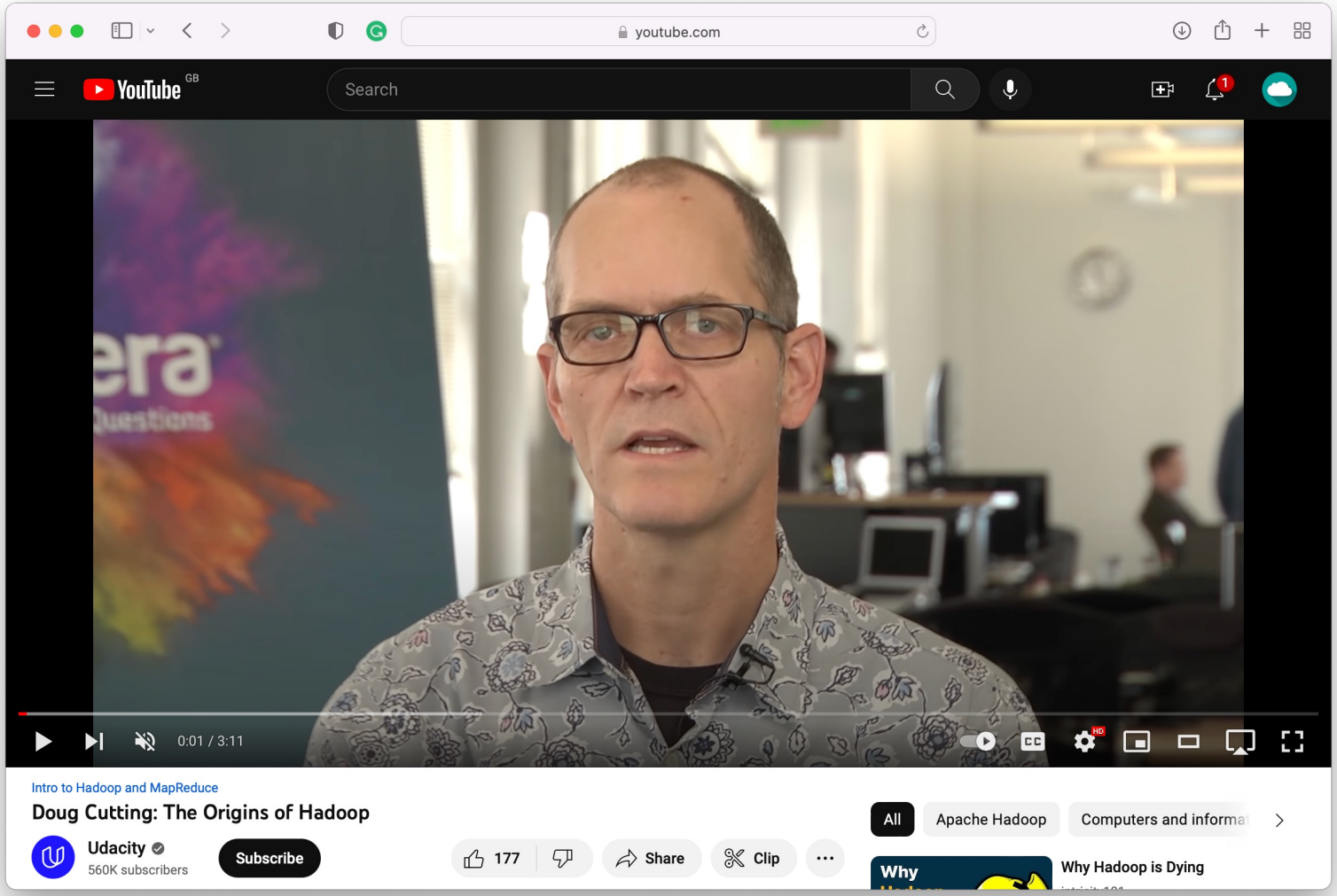
Batch processing vs Online (interactive) processing

- The ability to run an ad hoc queries against your **whole dataset** and get the results in a **reasonable time**.
- Hadoop MapReduce is fundamentally a batch processing system, and is not suitable for online (interactive) analysis.
  - You can’t run a query and expect results back in a few seconds or less.
- Queries typically take minutes or more, **so it’s best for offline use**, where there isn’t a human sitting in the processing loop waiting for results.
- Today we focus on Batch processing and Hadoop MapReduce.

Hadoop MapReduce

A batch processing ecosystem...

Watch Doug Cutting: The Origins of Hadoop



What is Apache Hadoop?

- Hadoop is a collection of software systems that was created by Doug Cutting and Mike Cafarella in 2005.
- **Doug**, was working at **Yahoo**! (at that time) is now Chief Architect of Cloudera, and named Hadoop project after his son's toy elephant.
- Hadoop facilitates and simplifies the processing of vast amounts of data in parallel on large clusters of commodity hardware in a reliable, fault-tolerant manner:
  - Petabytes of data
  - Thousands of nodes
- Today is used by Google, Yahoo!, Facebook, IBM, Twitter etc.

Doug Cutting with Hadoop, his son's stuffed elephant

ILLUSTRATION: CLOUDERA

Why to learn Hadoop?

- Hadoop MapReduce is one of the earliest attempts to process parallel computations.
- Hadoop is the essence of big data analysis.
- Today there are many alternative frameworks (Spark, Storm, Flink etc.)
- **Understanding Hadoop pros and cons is key to understand where each framework fits into the Big Data map.**

Hadoop is about divide and conquer…

Partition computations → Distributed computation ← Computation tier is a general-purpose scheduler and a distributed processing framework (in Hadoop called MapReduce)

Partition data → Distributed storage ← Distributed Storage is provided via the Hadoop Distributed Filesystem (HDFS)

Commodity hardware →

← Hadoop runs on commodity hardware

Hadoop is a collection of modules

- **Hadoop Common**: The common utilities that support the other Hadoop modules.
- **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
- **Hadoop MapReduce**: A system for parallel processing of large data sets.
- **YARN**: A framework for job scheduling and cluster resource management.

HDFS

Hadoop Distributed File System

Idea of running an analysis in parallel (1)

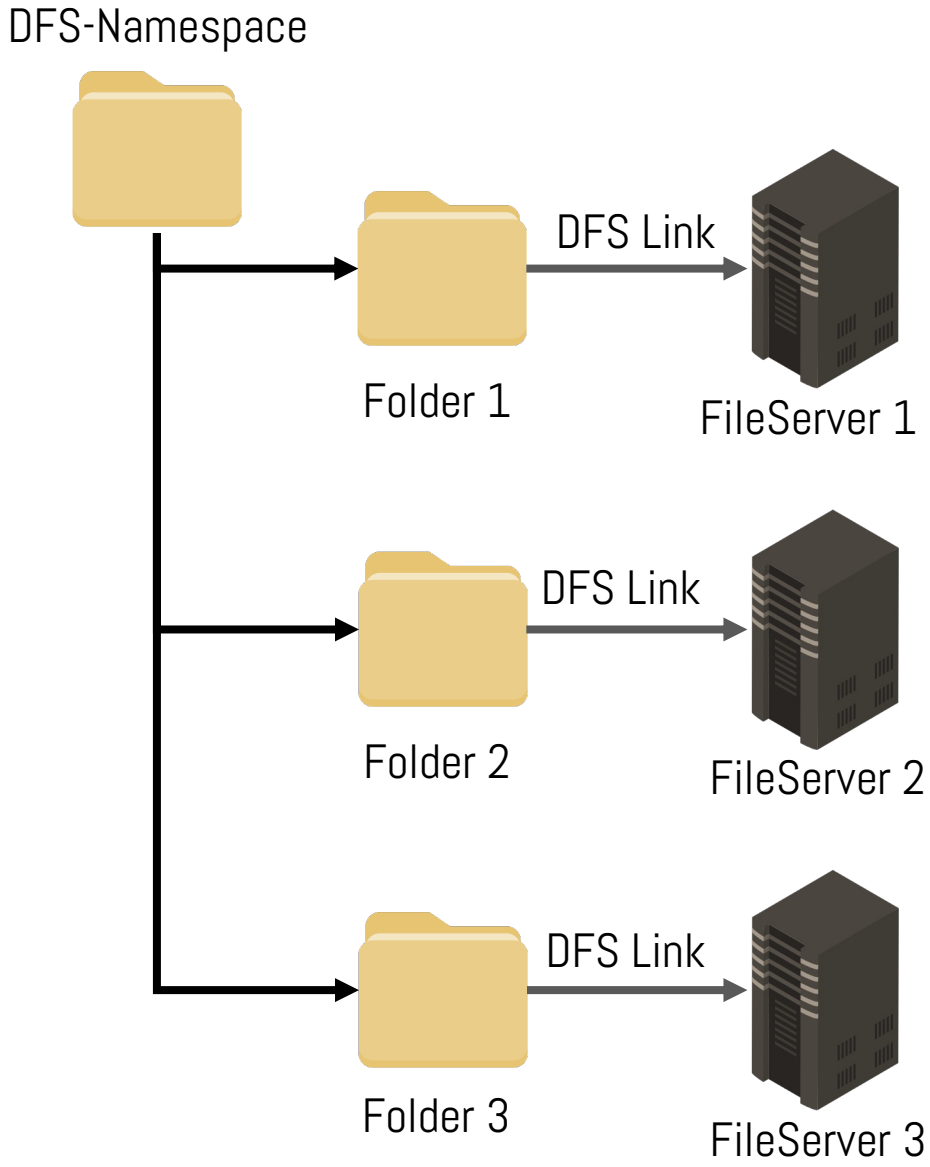
- You have a big json file (10 TB)
  - The file has a field called **sales_per_day**
  - For example: it is about Amazon sales in the past 20 years
- You want to analyse it!
  - What is the maximum sale ever happened?

Idea of running an analysis in parallel (2)

- Idea of divide and conquer!
  - Let us split the file in smaller chunks and then analyse each chunk in lots of nodes in parallel.
    - Each node will output 1 value, that will be the maximum number of sales
  - Then get the results and aggregate lots of results to find the max sale.
- But, how can I split the file in lots of pieces?
  - **Use Hadoop distributed file system!**

Distributed File System

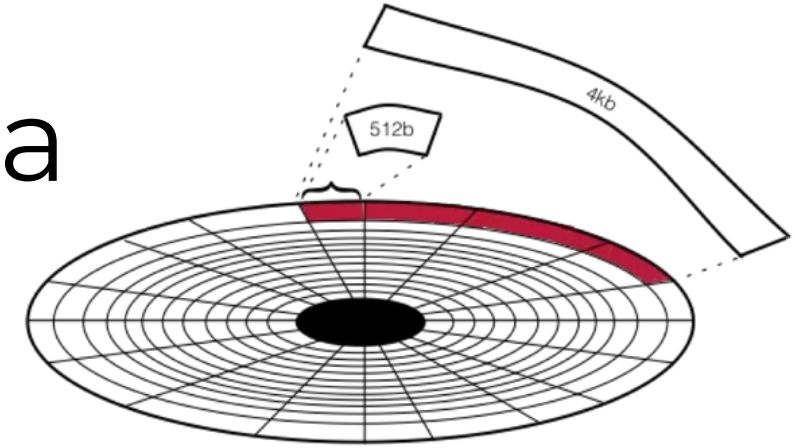
- A Distributed File System (DFS) manages files and folders across multiple computers.
- We can have multiple disks from multiple computers integrated into one distributed file system over the network.
- We have a unique namespace for the whole cluster.
  - Note: **file systems** are **namespaces** that assign names to **files**



HDFS at a glance…

- Definition of HDFS:
  - “HDFS is a filesystem designed for storing **very large files** with **streaming data access patterns**, running on clusters on **commodity hardware”**
- **Very large files:**
  - Files of terabytes in size, there are Hadoop clusters running today that store petabytes of data.
- **Streaming data access:**
  - HDFS is built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern
  - **Streaming data access** implies that instead of reading **data** as packets or chunks, **data** is read continuously with a constant bitrate.

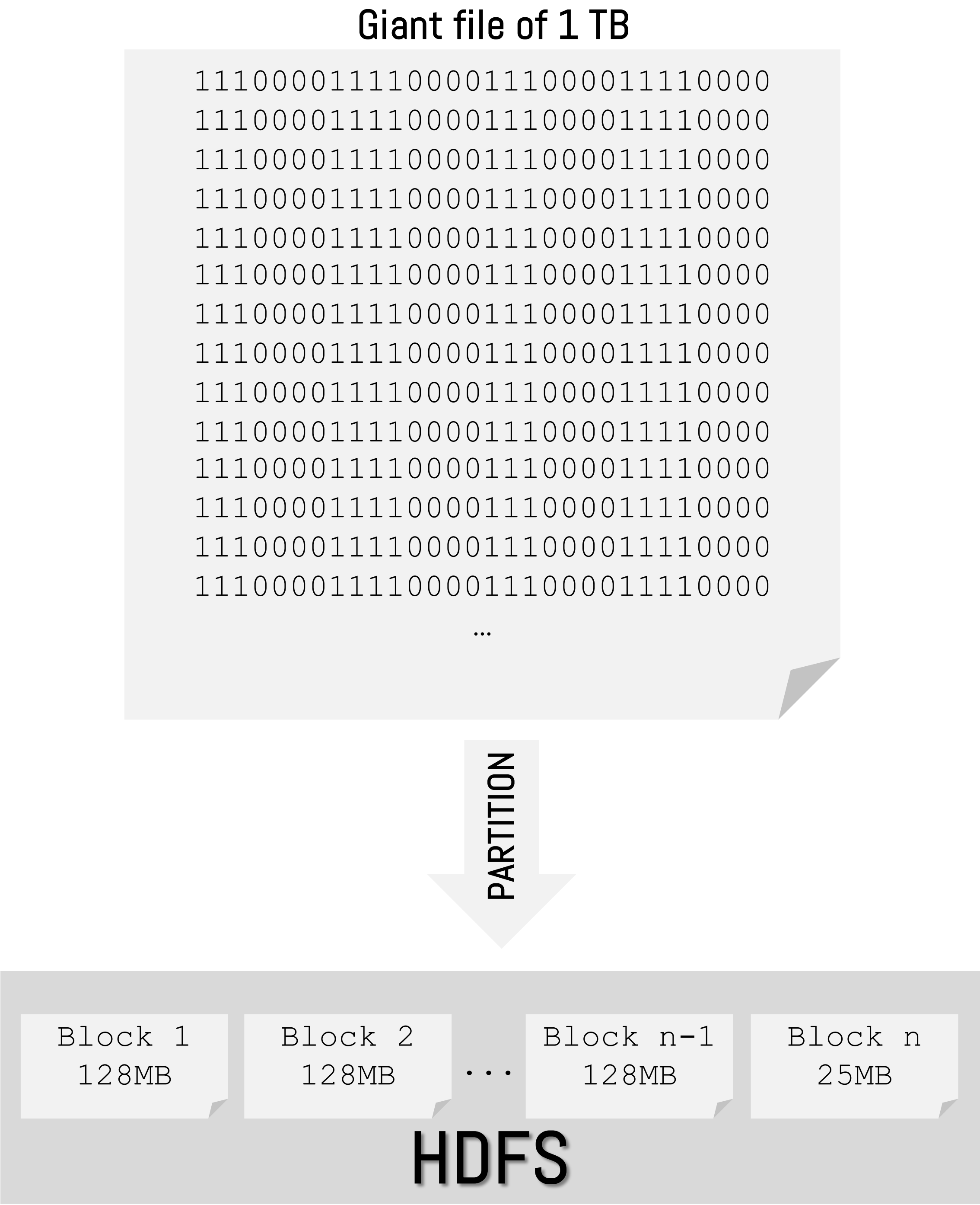
HDFS is about storing data



- HDFS integrates file systems of different nodes into one unique namespace
- Each filesystem has a block size
  - A disk (e.g. in your laptop) has a block size, which is the minimum amount of data that it can read or write.
- Each time you save a file you use disk blocks to save it
  - Newest drives has disk block size from 512 bytes to 4096 bytes

Disc blocks

- HDFS has the concept of a block, but it is a much larger unit:
  - **128 MB** by default in Hadoop 2, or **64 MB** in Hadoop 1
- Files in HDFS are broken into block-sized chunks, which are stored as independent units
- Why it is so big?
  - The reason is to minimize the cost of seeks
  - A block will be stored as a contiguous piece of information on the disk
  - That means that the total time to read it completely is the time to locate it (seek time) + the time to read its content without doing any more seeks



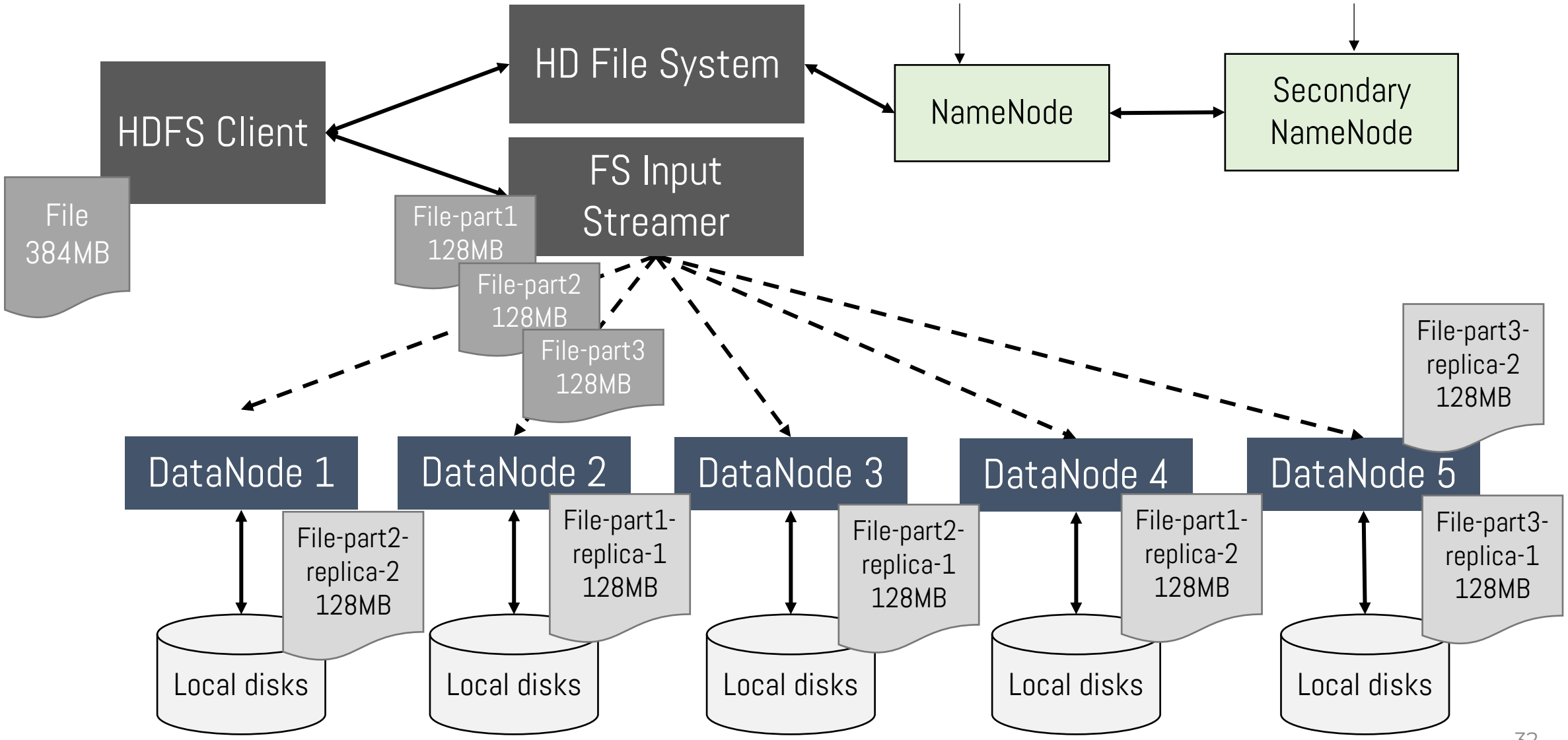
HDFS Data Organisation

- Each file written into HDFS is split into data blocks (128 MBs by default)
- Each block is stored in one or more nodes
- Each block **is then replicated** (by default 3 times) for fault tolerance
  - Each copy of the block is called replica
- We have a block placement policy:
  - First replica is placed on the local node
  - Second replica is placed in a different rack
  - Third replica is placed in the same rack as the second replica

HDFS managers

- There are two types of machines in a HDFS cluster
- **NameNode**:
  - It is the heart of an HDFS filesystem, it maintains and manages the file system metadata.
  - Keeps information about:
    - What blocks make up a file, and where (the datanodes...) those blocks are stored.
- **DataNode:**
  - It is where HDFS stores the actual data
  - There are usually quite a few of these.

HDFS


NameNode:
• File-part1 in DataNode1
• File-part2 in DataNode2
• File-part3 in DataNode4
• ... replicas ...
Secondary NameNode (copy):
• File-part1 in DataNode1
• File-part2 in DataNode2
• File-part3 in DataNode4
• ... replicas..
HD File System
HDFS Client
NameNode
Secondary NameNode
FS Input Streamer
File 384MB
File-part1 128MB
File-part2 128MB
File-part3 128MB
File-part3-replica-2 128MB
DataNode 1
DataNode 2
DataNode 3
DataNode 4
DataNode 5
File-part2-replica-2 128MB
File-part1-replica-1 128MB
File-part2-replica-1 128MB
File-part1-replica-2 128MB
File-part3-replica-1 128MB
Local disks
Local disks
Local disks
Local disks
Local disks

Namenode

- The namenode manages the filesystem namespace
  - It maintains the filesystem tree and the metadata for all the files and directories in the tree
  - The namenode **also knows the datanodes on which all the blocks for a given file are located**
  - A client accesses the filesystem on behalf of the user by communicating with the namenode and datanodes

Datanodes

- Datanodes are the work horses of the filesystem.
- **They store and retrieve blocks** when they are told to (clients or namenode).
- They report back to the namenode periodically with lists of blocks they are storing.
- **Without the namenode, the filesystem cannot be used.**
  - If the machine running the namenode were obliterated, all the files on the filesystem would be lost.
  - For this reason, it is important to make the namenode resilient to failures, and Hadoop provides a mechanism for this (a **secondary namenode**).
    - It keeps a copy of the namespace, which can be used in the event of the **namenode** failing

Summarize about HDFS

- **Is about how to prepare a file for parallel processing…**
- Hadoop Distributed File System:
  - Each file enter in the HDFS is split in different pieces called blocks
    - The bigger the pieces, the less we spend on seeks!
  - Allow computation to move closer to the data
  - HDFS creates multiple replicas of data blocks and distributed them on compute nodes throughout the cluster to ensure reliability and rapid computations

TRUE or FALSE

1. Divide and conquer in big data is about dividing data. FALSE (Data & Computations)
2. When a file is written in HDFS, it is split in size of KBs. FALSE (64 or 128 MBs)
3. In HDFS, a namenode is responsible for storing data. FALSE
4. In HDFS, a file is not replicated (by default). FALSE (3 replicas)
5. In HDFS, a secondary namenode stores a copy of the namenode filesystem for fault tolerance. TRUE

MapReduce

MapReduce is about Map and Reduce...

How all started...

- Once upon a time...
  - Doug Cutting and Mike Cafarella decided to build an open source web search engine...
- Building a web search engine from scratch was an ambitious goal:
  - Software to crawl and index websites is complex to develop!
  - Challenging task to run **without** a dedicated operations team
- It's expensive too:
  - They estimated a system supporting a one-billion-page index would cost around $500K in hardware, with a monthly running cost of $30K.
  - Yet, they believed it was a worthy goal!

The first effort: Nutch

- Nutch was started in 2002: A working crawler and search system
  - Nutch was not able to scale to the billions of pages on the Web.
- But they were lucky…
  - In 2003 Google published a paper about the Google's distributed filesystem (GFS)
  - This would solve their storage needs for the very large files generated from the web crawling/indexing process.
  - In 2004, Nutch developers produced the Nutch Distributed Filesystem (NDFS).
- In 2004, Google published the paper that introduced MapReduce to the world!
- In 2005, Nutch developers had a working MapReduce implementation in Nutch
- In 2006, they moved out of Nutch to form an independent project called Hadoop
  - Doug Cutting joined Yahoo! and focus to turn Hadoop into a system that ran at web scale
- February 2008 Yahoo! announced that its production search index was being generated by a 10,000-core Hadoop cluster

Hadoop is on!

- In April 2008, Hadoop **broke a world record to become the fastest system to sort an entire terabyte of data**.
  - Running on a 910-node cluster, **Hadoop sorted 1 terabyte in 209 seconds** (just under 3.5 minutes)
- In November 2008, Google reported that its MapReduce implementation sorted 1 terabyte in **68** seconds.
- In April 2009, it was announced that a team at Yahoo! had used Hadoop to sort 1 terabyte in **62** seconds.
- The race goes on…
- In the 2014 competition, a team from **Databricks** were joint winners of the Gray Sort benchmark (competition for fast to sort humongous data).
  - They used a 207-node Apache Spark cluster to sort **100 terabytes** of data in 1406 seconds:
    - A rate of **4.27 terabytes per minute**!

What is MapReduce?

- MapReduce is a programming model for data processing
- Hadoop is the framework to support distributed storage and processing
- Hadoop can run MapReduce programs written in various languages such as Java, Ruby, and Python.
- MapReduce programs are inherently parallel!

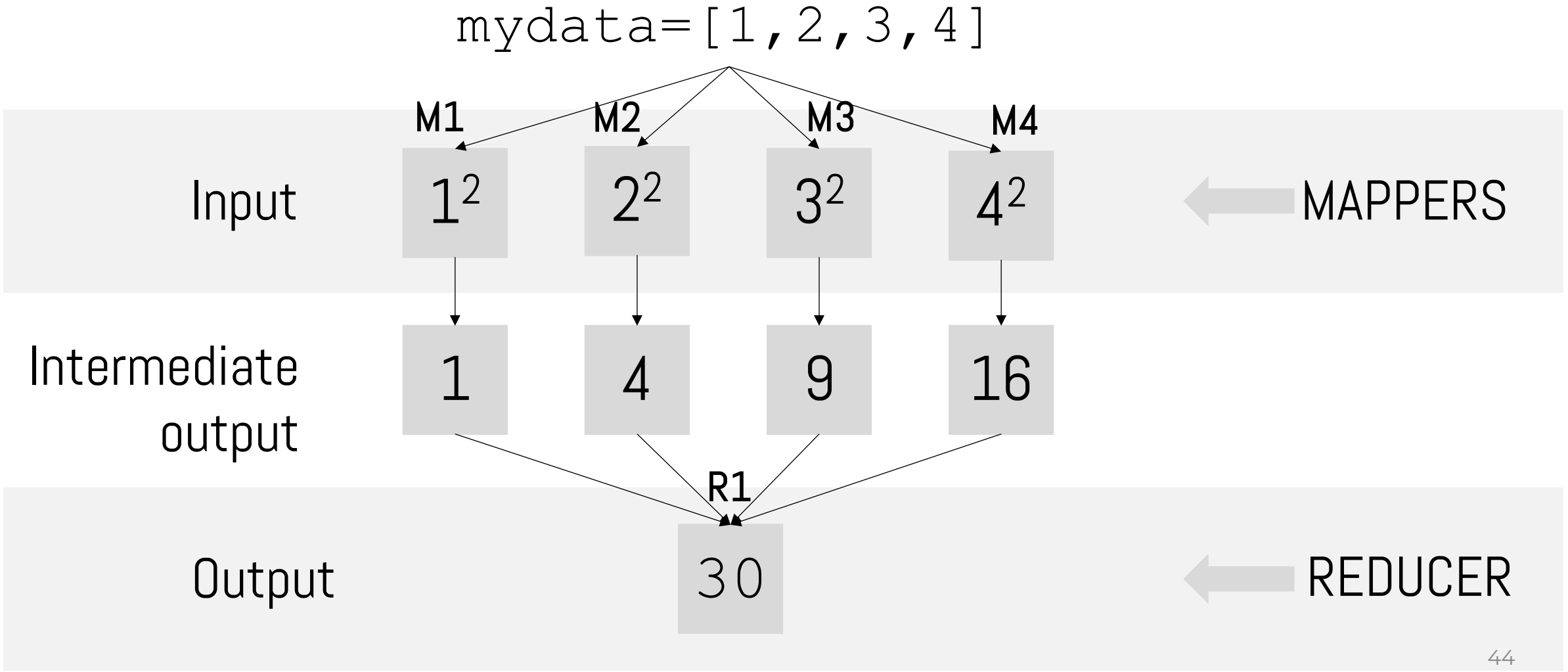
MapReduce

- **Hadoop “takes computation to the data”**
- MapReduce works by breaking the processing into two phases:
  - The **map** phase
  - The **reduce** phase
- Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer.

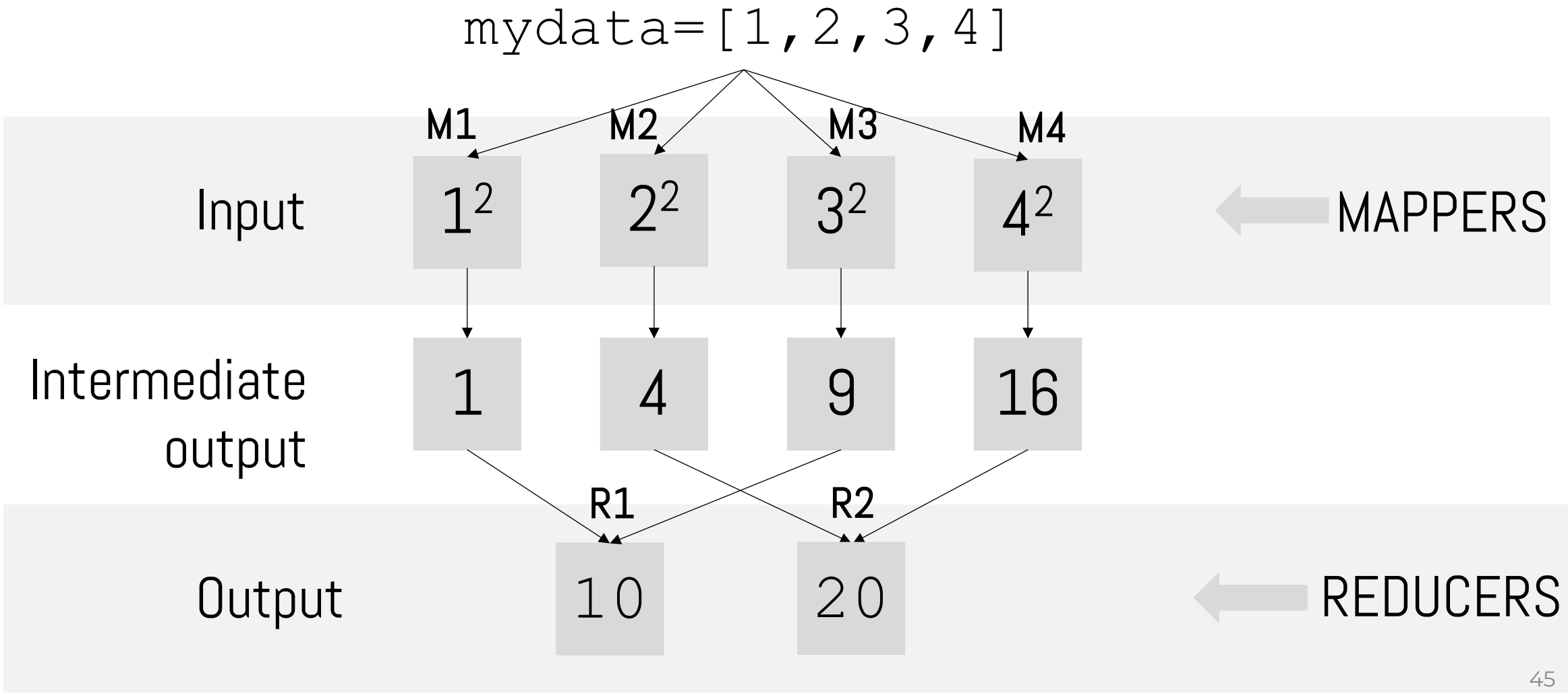
Functional Programming: Map and Reduce

- **Map**: Returns a list constructed by applying a function (first argument) to all items of a list (passed as second argument).
  - map (f, [a,b,c]) = [f(a), f(b), f(c)]
  - Example:
    - map ($f(x)=x^2$, [1,2,3]) = [f(1), f(2),f(3)] = [1,4,9]
- **Reduce**: Returns a list constructed by applying a function (first argument) on the list passed as the second argument.
  - reduce f [a,b,c,] = f(a,b,c)
  - Example:
    - reduce (sum[1,4,9]) = sum(1, sum (4, sum (9))) = **14**

Sum of square example



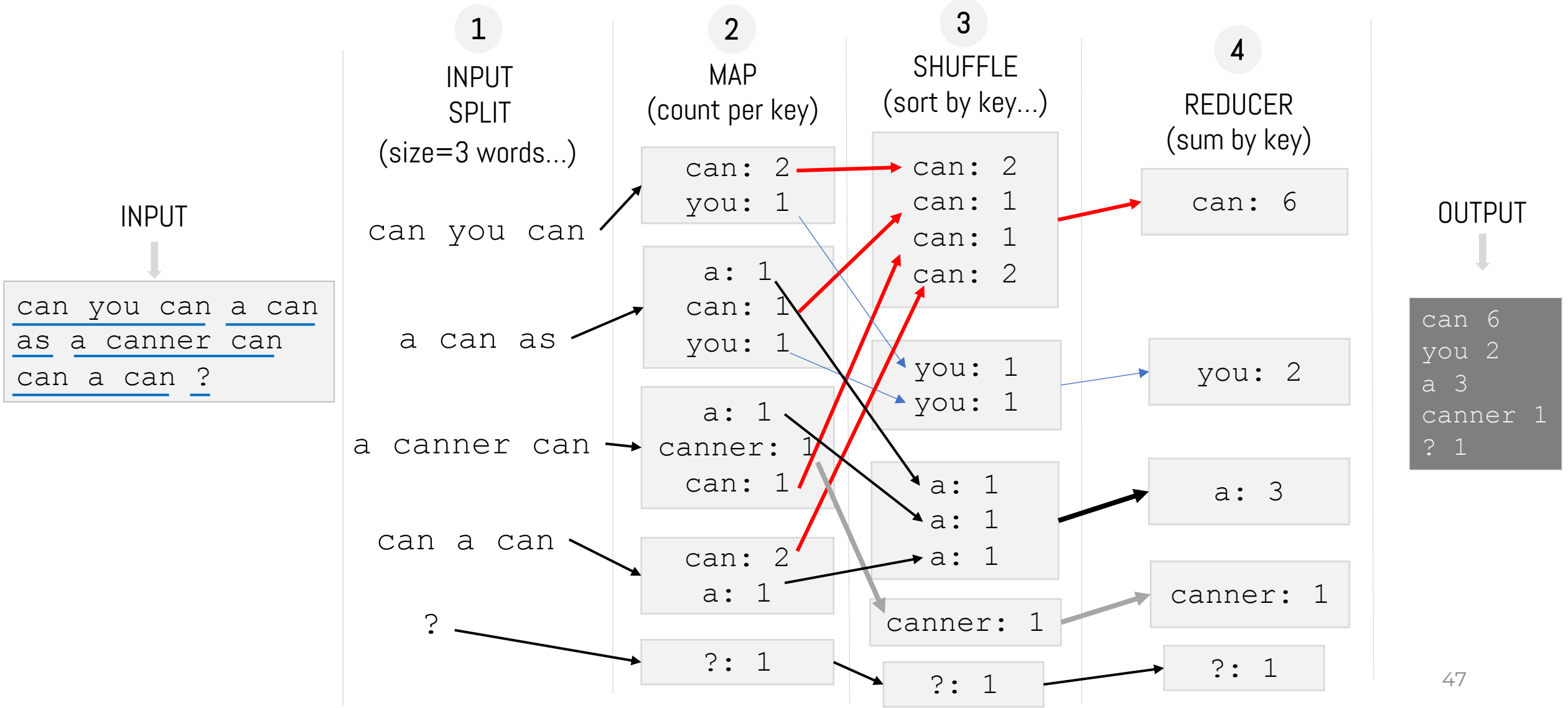
Sum of square example, for **even and odd** numbers



MapReduce wordcount example of a job

- Input: myInputFile.txt
- Output: myOutputFile.txt
- Why to run this job in parallel?
  - Partition the file…Partition the computation…
  - Run partitioned computation to partitioned file…
- How to perform word count in parallel?
  - Use Hadoop MapReduce

How MapReduce counts words?



Let us use a real world example

- Example from **Hadoop: The Definitive Guide**
- Input to our map phase is the raw National Climatic Data Center (NCDC) data.
- We want to process all the data, and the data is semi-structured and record-oriented.
- Query: What is the maximum air temperature per year?

```
0057
332130   # USAF weather station identifier
99999    # WBAN weather station identifier
19500101 # observation date
0300     # observation time
4
+51317   # latitude (degrees x 1000)
+028783  # longitude (degrees x 1000)
FM-12
+0171    # elevation (meters)
99999
V020
320      # wind direction (degrees)
1        # quality code
N
0072
1
00450    # sky ceiling height (meters)
1        # quality code
C
N
010000   # visibility distance (meters)
1        # quality code
N
9
-0128    # air temperature (degrees Celsius x 10)
1        # quality code
-0139    # dew point temperature (degrees Celsius x 10)
1        # quality code
10268    # atmospheric pressure (hectopascals x 10)
1        # quality code
```

Analysing the Data with Hadoop…

- To take advantage of the parallel processing that Hadoop provides, we need to express our query as a MapReduce job.
- We need to break it into two phases:
  - the map phase and
  - the reduce phase
- Each phase has **key-value pairs as input and output**, the types of which may be chosen by the programmer.
  - In our case key is the year, value is the air temperature

Example (1)

- Datafiles are organized by date and weather station.
  - There is a directory for each year from 1901 to 2001, each containing a **gzipped** (Linux zip) file for each weather station with its readings for that year.
- For example, here are the first entries for 1990:

```
% ls raw/1990 | head
010010-99999-1990.gz
010014-99999-1990.gz
010015-99999-1990.gz
010016-99999-1990.gz
...
```

- What's the highest recorded global temperature for each year in the dataset?

Example (2)

- Our map function is simple: We pull out the year and the air temperature, because these are the only fields we are interested in.
  - The map function is also a good place to drop bad records
  - These lines are presented to the map function as the key-value pairs:

    (0, 0067011990999991950051507004...9999999N9+00001+99999999999...)

    (106, 0043011990999991950051512004...9999999N9+00221+99999999999...)

    ...

- The map function extracts the year and the air temperature from the data.
  - (1950, 0)
  - (1950, 22)
  - ...
  - To extract we use the offset from the beginning of the file...

Example (3)

- The output from the map function is processed by the MapReduce framework before being sent to the reduce function.
- This processing sorts and groups the key-value pairs by key, for example:

  (1949, [111, 78])

  (1950, [0, 22, −11])

  ...

- Each year appears with a list of all its air temperature readings.
- All the reduce function has to do now is iterate through the list and pick up the maximum reading, for example:

  (1949, 111)

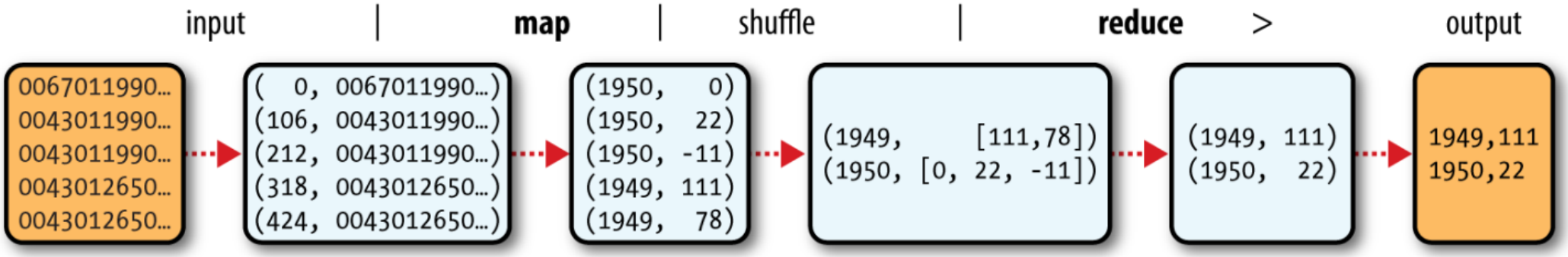
  (1950, 22)

  ...

- This is the final output: the maximum global temperature recorded in each year.

MapReduce logical data flow

- Input: The raw file
- Map: extract year and temperature (key: value)
- Shuffle: Sort by keys and aggregate results by keys in a list
- Reduce: Apply reduce function (maximum of temperature per year)
- Output: Write data to a file



Phases

- Input Splits:
  - An input to a MapReduce job is divided into fixed-size pieces called input splits.
  - Input split is a chunk of the input that is consumed by a single map
- Mapping
  - This is the very first phase in the execution of map-reduce program.
  - In this phase data in each split is passed to a mapping function to produce output values.
- Shuffling
  - This phase consumes the output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output
- Reducing
  - In this phase, output values from the Shuffling phase are aggregated.
  - This phase combines values from Shuffling phase and returns a single output value.

When to use MapReduce?

YES!

- You need to analyse the **whole dataset**, in a batch fashion
  - For example for ad hoc analysis.
    - Count the words of a dataset once.
- MapReduce suits applications where the **data is written once**, and **read many times**.

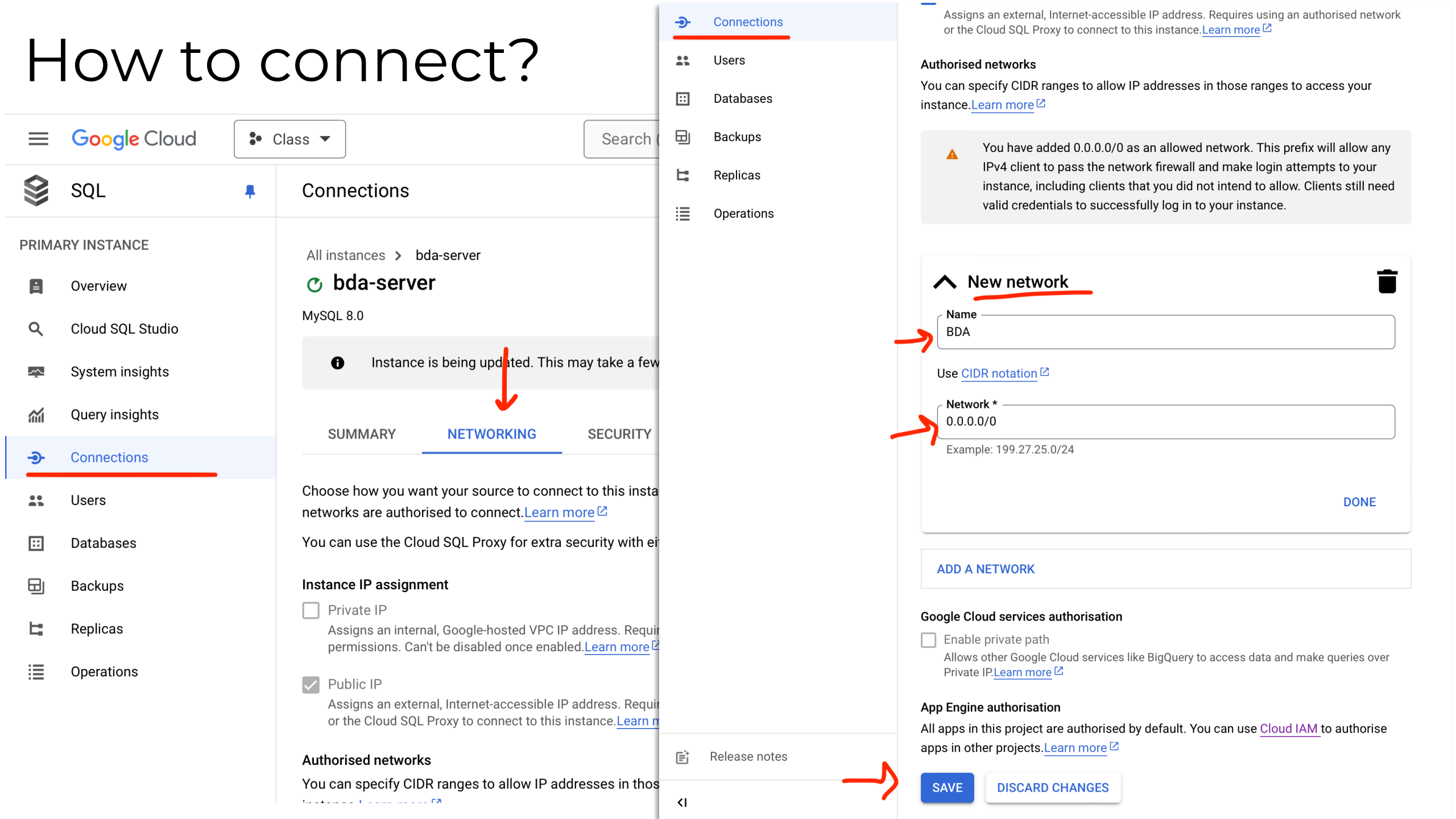
NO!

- When we need to run queries or updates on a dataset that has been indexed to deliver low-latency retrieval and update times of a relatively small amount of data.
- Continually updated dataset:
  - Use a relational database

Lab 8 material

Supporting material on MySQL and Python

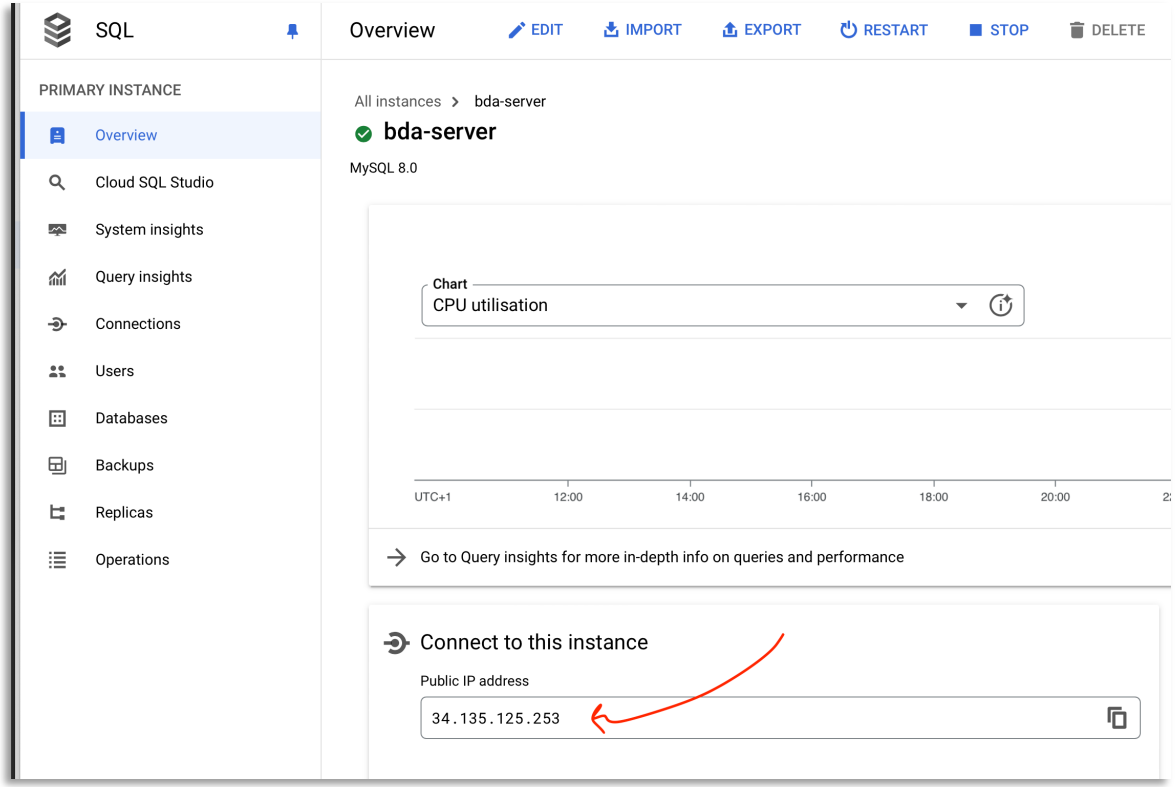
How to connect?



How to connect?

- Install the MySQL connector using `pip`
  - `!pip install mysql.connector`
- Load the MySQL connector library
  - `import mysql.connector`
- Then create a connection:

```
connection = mysql.connector.connect(
            host= "MYSQL_IP_ADDRESS",
            user= "MYSQL_USERNAME",
            passwd= "MYSQL_PASSWORD"
                database = "MYSQL_DATABASE"
        )
```



Connection is needed to create a `cursor`

- Create a new cursor using the connection
  - `cursor = connection.cursor()`
- Execute a query
  - `cursor.execute("CREATE DATABASE music_db;")`

Example of MySQL query using Python

- Let's extract a list of our customers:

```
query = """
SELECT *
FROM customers;
"""
cursor.execute(query)
results = cursor.fetchall()
for result in results:
  print(result)
```