University of Salford MANCHESTER

SCHOOL OF SCIENCE, ENGINEERING & ENVIRONMENT

Program: MSc of Data Science
Module: Big Data Tools and Techniques

Week 2

Introduction to Tools & Techniques for Big Data

2025

JISC Code

709429

Ground Rules

1. Choose a quiet place to attend the class and please concentrate during the lecture
2. Put your questions in Padlet (not Teams' chat box) and I will review them in the due time (Padlet link is in the Bb, week 2)
3. Handout has been shared on BB
4. Turn off your mic during the lecture
5. We will have 5 mins break after the first hour of the lecture (please remind me)
6. Jisc code will be shared during the break time

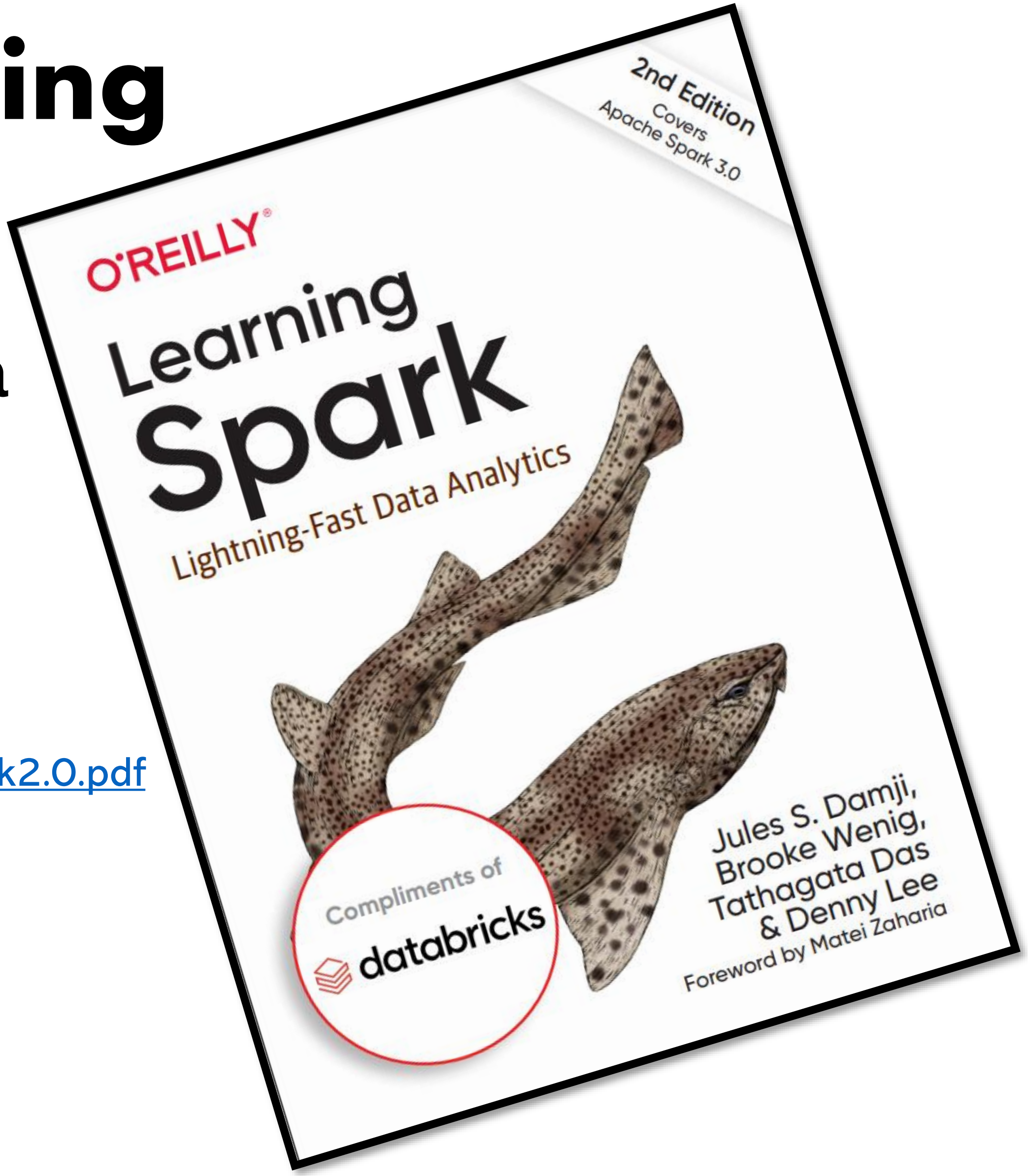
Learning Outcomes

1. To introduce Apache Spark and Databricks
2. To give you an understanding of data storage options and distributed filesystems
3. To explain why we use distributed systems for processing big data and the challenges this entails
4. To provide an overview of the map-reduce programming model and its implementation in Hadoop MapReduce
5. To give you an understanding of why Apache Spark can often improve on Hadoop MapReduce

Supporting your Learning

Main module text:

"Learning Spark: Lightning-Fast Data Analytics"



Free e-book from Databricks

https://pages.databricks.com/rs/094-YMS-629/images/LearningSpark2.0.pdf

Complete the weekly quizzes!

Module Plan

Week	Lecture / Workshop
1	Introduction to Big Data
2	Introduction to tools for Big Data
3	Spark RDDs
4	Spark DataFrames
5	Spark SQL
6	Spark Streaming
7	Machine Learning in Spark (MLlib)
8	Recommender Systems (MLlib)
9	NoSQL (MongoDB)
10	Applications of Big Data Tools & Techniques

Refresher: What is Big Data?

- The DIKW (Data, Information, Knowledge, Wisdom) Pyramid
- Descriptive, Diagnostic, Predictive & Prescriptive Analytics
- 5V's of Big Data:
  - Volume
  - Velocity
  - Variety
  - Veracity
  - Value
- The Big Data Analytics Lifecycle
- Cloud Computing – Advantages & Disadvantages
- Cloud Service Models (SaaS, PaaS, IaaS)

Refresher: What is Big Data?

- The DIKW (Data, Information, Knowledge, Wisdom) Pyramid
- Descriptive, Diagnostic, Predictive & Prescriptive Analytics
- 5V's of Big Data:
  - Volume
  - Velocity
  - Variety
  - Veracity
  - Value
- The Big Data Analytics Lifecycle
- Cloud Computing – Advantages & Disadvantages
- Cloud Service Models (SaaS, PaaS, IaaS)

Working with Big Data

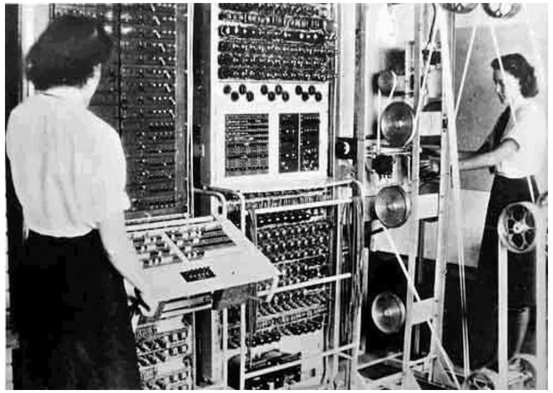
We need to consider:

- Data storage
- Data processing

Processing Big Data

What about when we are **processing** big data?

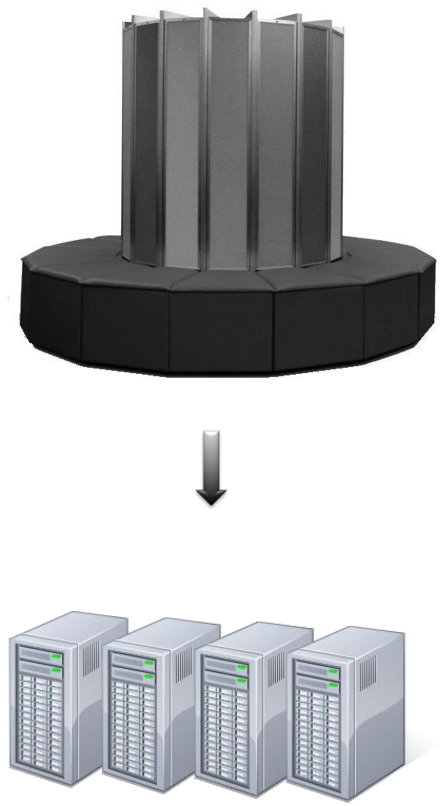
- Traditionally, computation has been processor-bound
  - Relatively small amounts of data
  - Lots of complex processing



- The early solution: bigger computers
  - Faster processor, more memory
  - But even this couldn't keep up

Data Processing

- The better solution: more computers
  - Distributed systems – use multiple machines for a single job



"In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, we didn't try to grow a larger ox. We shouldn't be trying for bigger computers, but for more systems of computers."

Grace Hopper

Distributed System Challenges

- Programming complexity
  - Keeping data and processes in sync
- Finite bandwidth
  - We want to minimise the number of times we are transferring data between nodes in the cluster
- Partial failures
  - The probability of any one specific node (machine) failing is low, but the probability of node failures when using hundreds / thousands of nodes is high

Big Data Processing

- Distributed
  - Thousands of nodes (machines) all working together
- Scalable
  - Adding nodes adds capacity proportionally
- Fault-tolerance
  - Node failure is inevitable

Counting Words

Imagine you have been tasked with counting the frequency of different words in a chunk of text with a group of friends – as in the example below:

Input Data

```
the cat sat on the mat
the aardvark sat on the sofa
```

Result

aardvark	1
cat	1
mat	1
on	2
sat	2
sofa	1
the	4

Counting Words

In this scenario:

- You are spread out in a noisy room
- You want to count the words as quickly as possible!

This is analogous to data processing on cluster (with each of you representing a single machine, or node)

Counting Words

Several challenges arise which mirror those of a distributed computing environment:

- No Shared Memory Access
- Bandwidth Limitations
- Fault Tolerance

Suggested Solution

- Divide the text equally between yourselves
- Each person counts the frequency of the words in their portion of the text (perhaps organising each out on a separate sheet of paper)
- At the end of stage 1, each member of your group will have a sub-total for their portion of the text.
- To get the overall total we can sum all the sub-totals for each word. But we want to do this efficiently – so we want you all to be working on this second stage
- To do this, you could 'shuffle and sort' your sub-totals – e.g., one of you takes the counts of words beginning A, etc. You can then all work on the second step to provide the final totals for all words.

Suggested Solution

This analogy was intended to give you an intuition for the challenges of processing data on a cluster. We will be introducing the MapReduce model later which is loosely analogous to this.

We will come back to this example later!

Working with Big Data

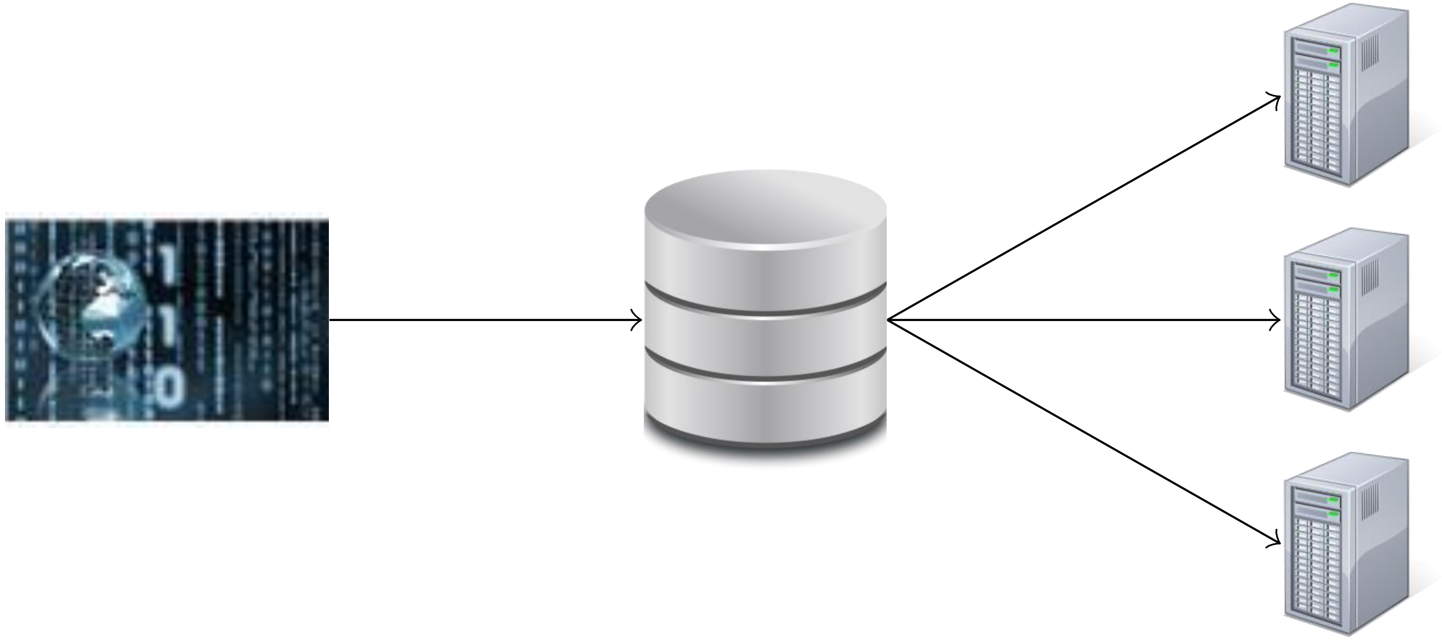
We need to consider:

- Data storage
- Data processing

What about data storage?

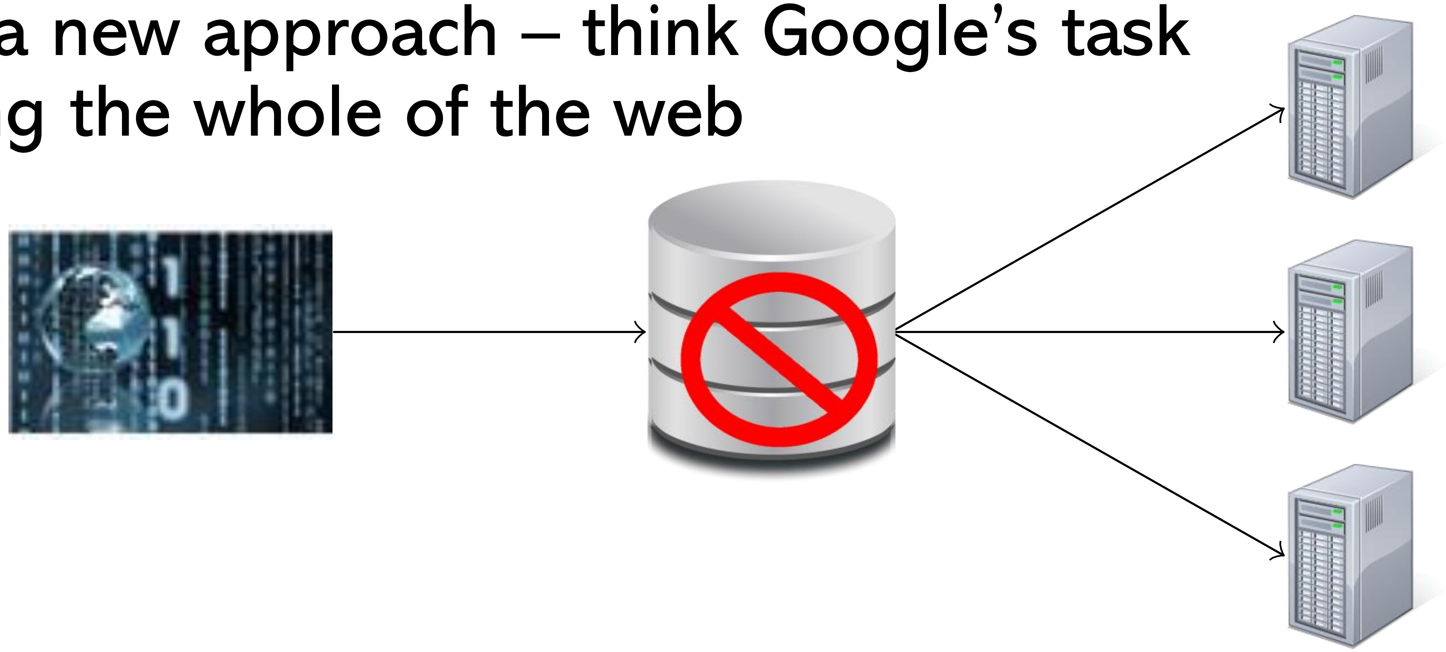
The Data Bottleneck

- Traditionally, data is stored in a central location
  - Data is copied to processors at runtime
  - Fine for limited amounts of data



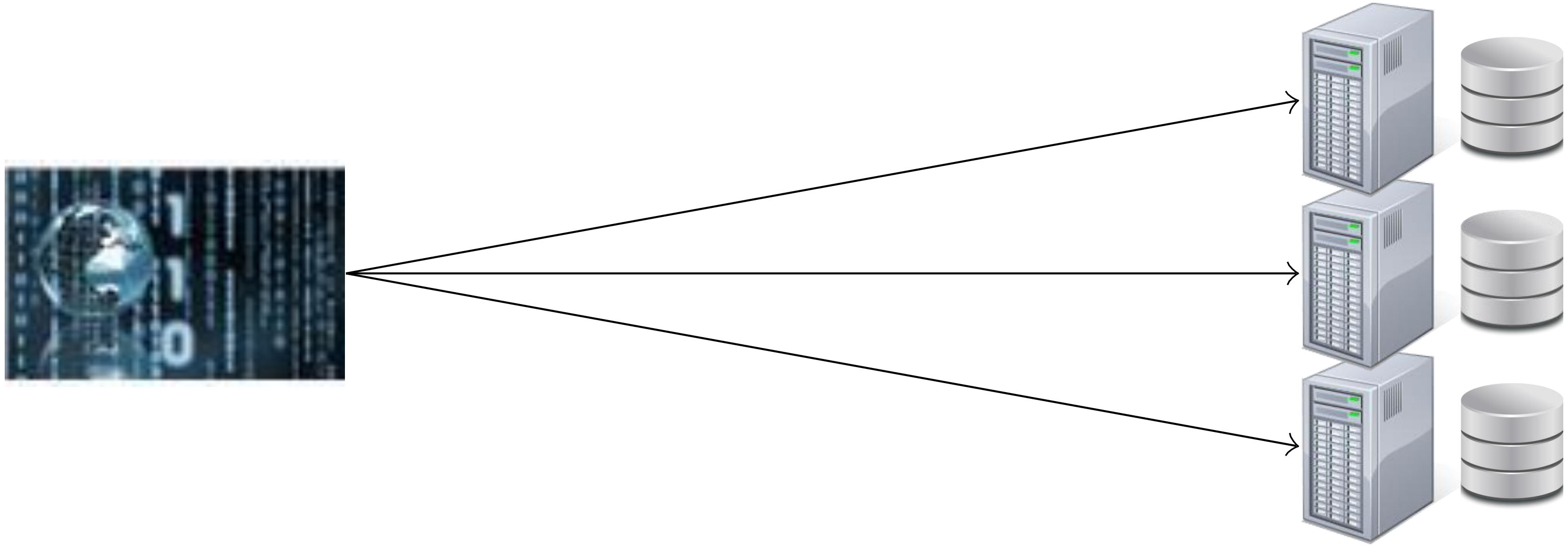
The Data Bottleneck

- Modern systems have much more data
  - terabytes+ a day
  - petabytes+ total
- We need a new approach – think Google's task of indexing the whole of the web



The Data Bottleneck

- We need a new approach – think Google's task of indexing the whole of the web
- Google's solution: Google File System (2003)



Apache Hadoop

- Released in 2006
- Provides an open-source framework for Big Data processing
- Hadoop File System (HDFS) – distributed file system
- Hadoop MapReduce – for data processing using the MapReduce paradigm

Apache Hadoop

- Platform that handles large datasets in a distributed fashion.
- Uses MapReduce to split the data into blocks & assign the chunks to nodes across a cluster.
- MapReduce processes the data in parallel on each node.
  - Every machine in a cluster stores and processes data.
  - Hadoop stores the data to disks (e.g. using HDFS).

Hadoop File System (HDFS)

- A filesystem which can store any type of data
- Provides redundant storage for massive amounts of data
- HDFS is a filesystem written in Java
- Sits on top of a native filesystem
- Scalable
- Fault tolerance
- Supports efficient processing with Spark, or MapReduce, or the other Hadoop frameworks

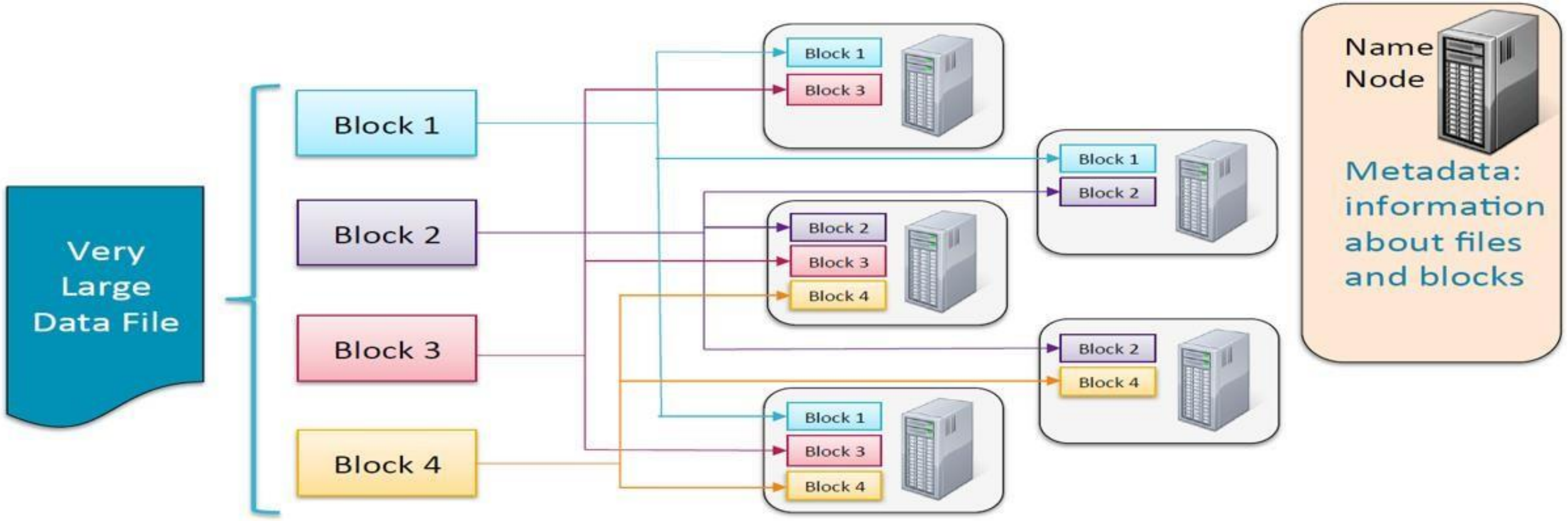
Hadoop File System (HDFS)

Some design requirements for HDFS:

- HDFS performs best with a modest number of large files
  - Millions, rather than billions, or less
  - Each file typically 100MB or more
- Files in HDFS are ‘write once’
  - Appends are permitted
  - No random writes to files are allowed

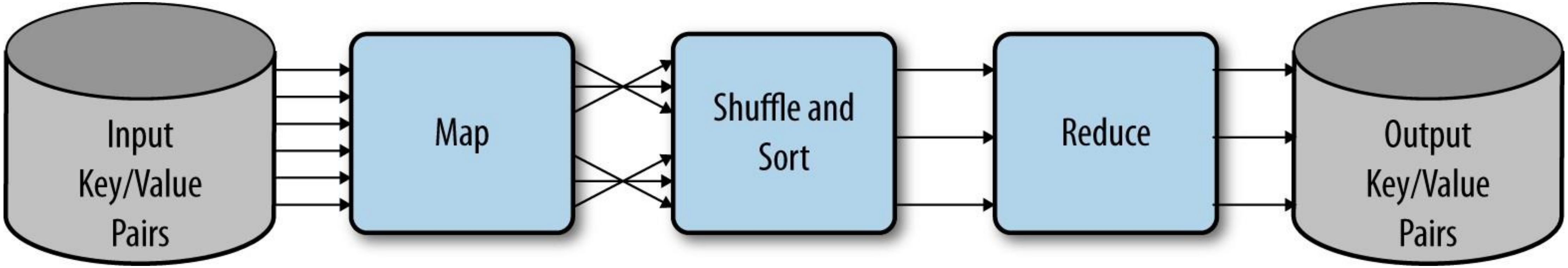
Hadoop File System (HDFS)

- Data files are split into blocks which are distributed to Data Nodes
- Each block is replicated on multiple data nodes (default 3x)
- NameNode stores metadata



MapReduce

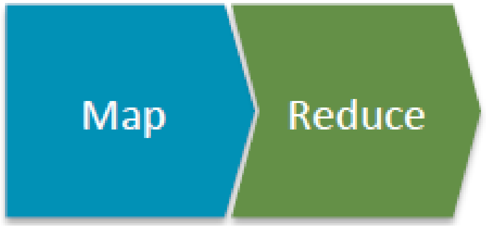
- MapReduce refers to both a programming model and its associated implementation in Hadoop.
- It facilitates concurrent processing on a cluster, by splitting a computation into a map phase and a reduce phase



MapReduce: Wordcount Example

Input Data

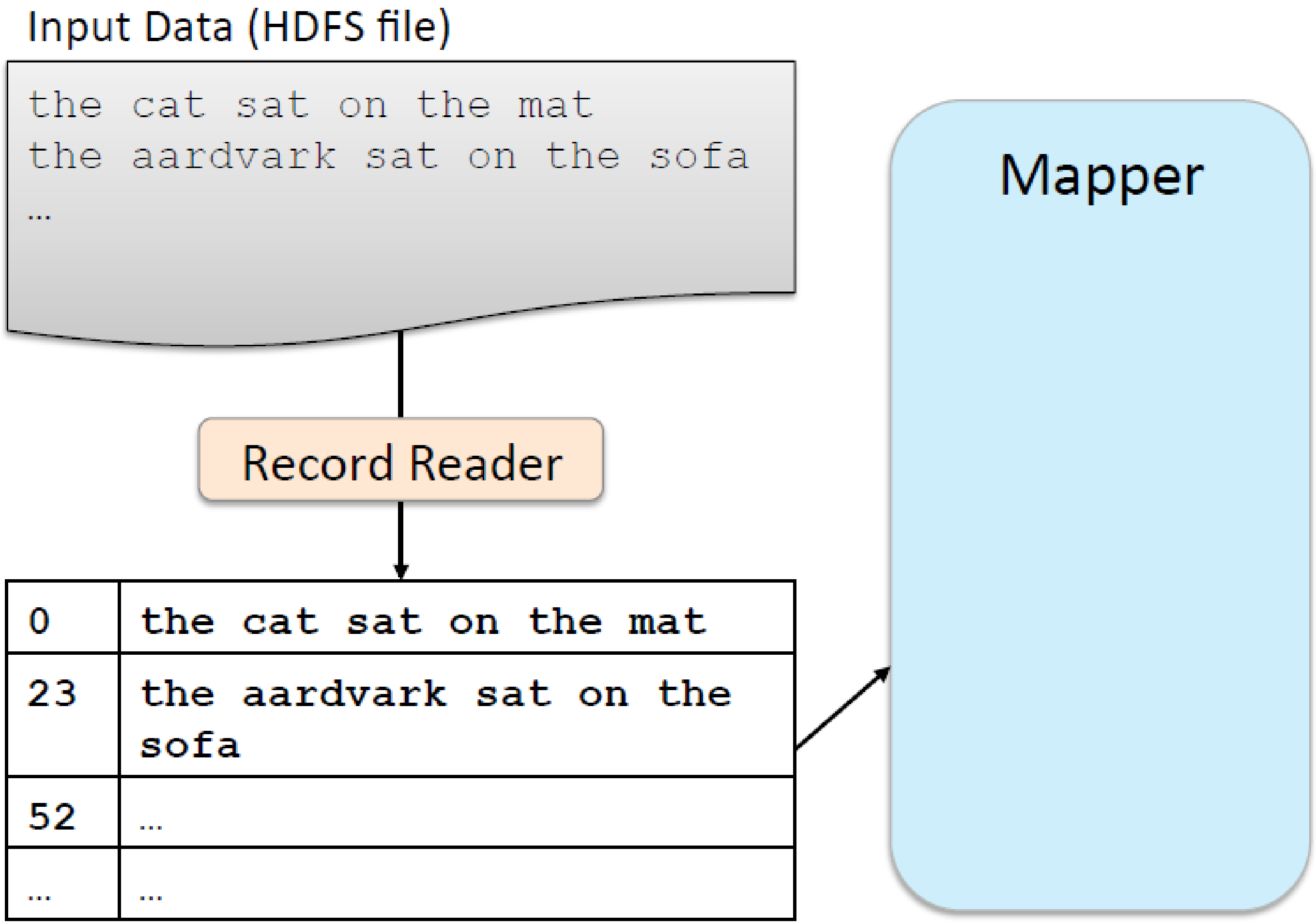
```
the cat sat on the mat
the aardvark sat on the sofa
```



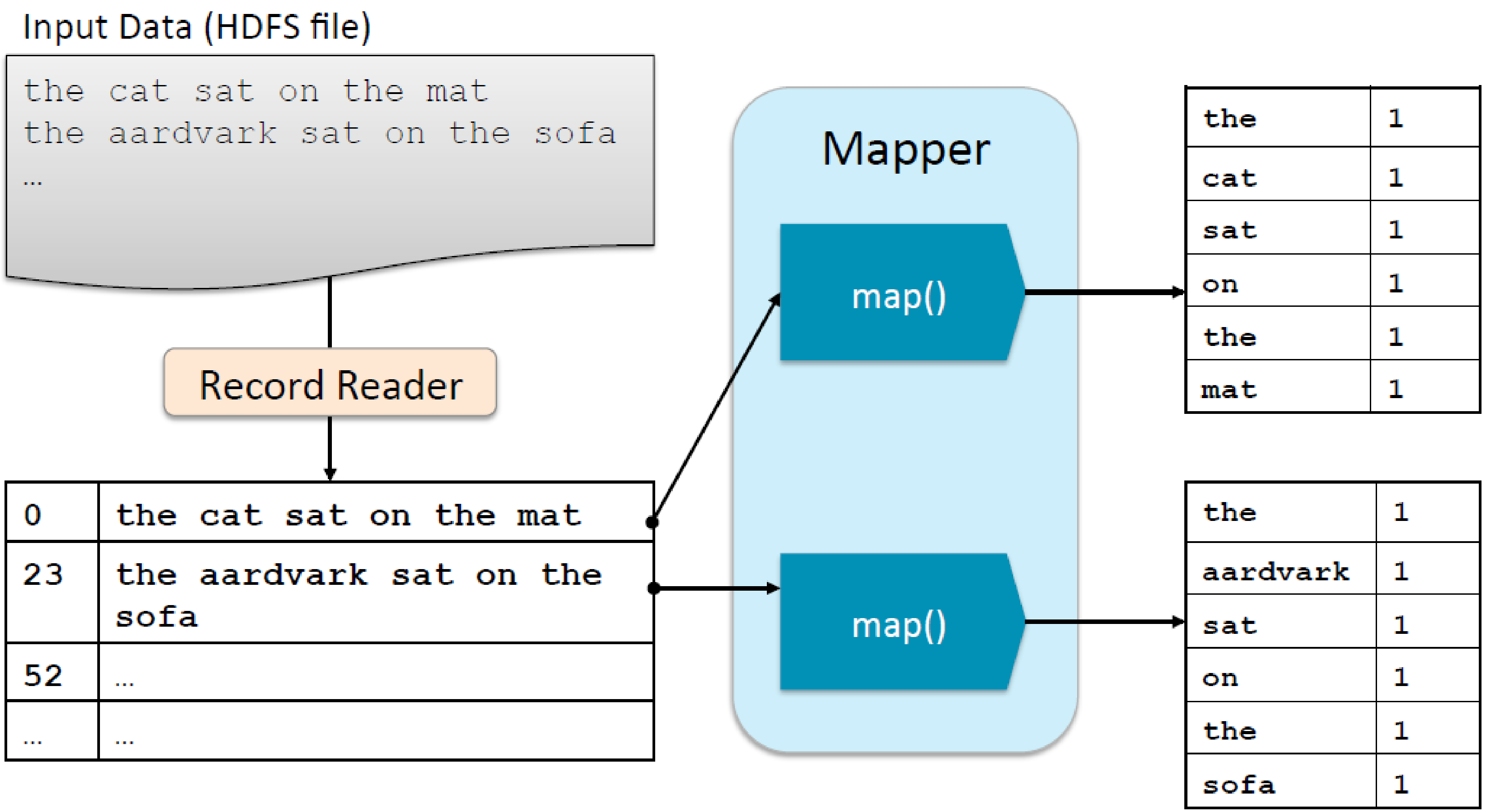
Result

aardvark	1
cat	1
mat	1
on	2
sat	2
sofa	1
the	4

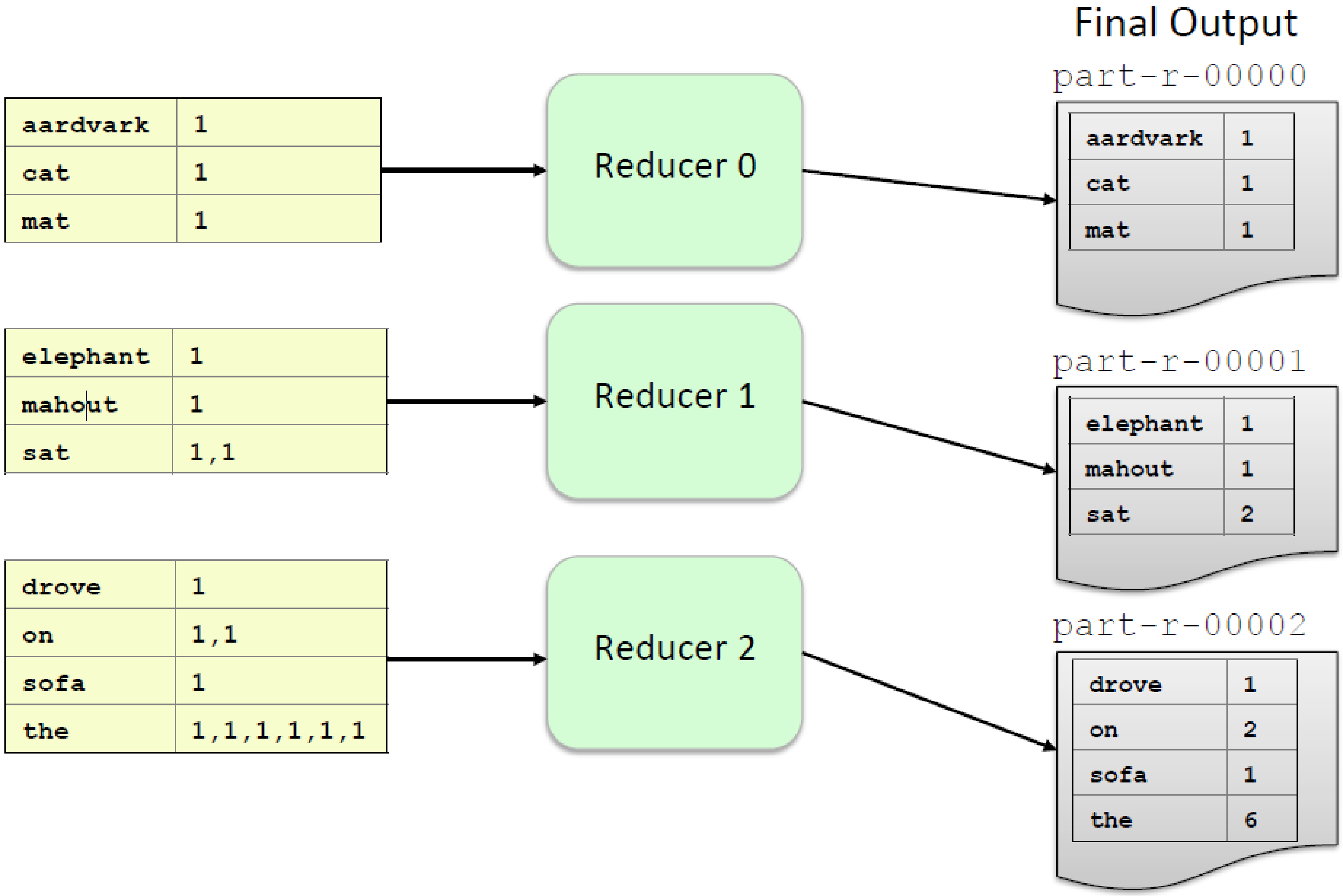
MapReduce: Map



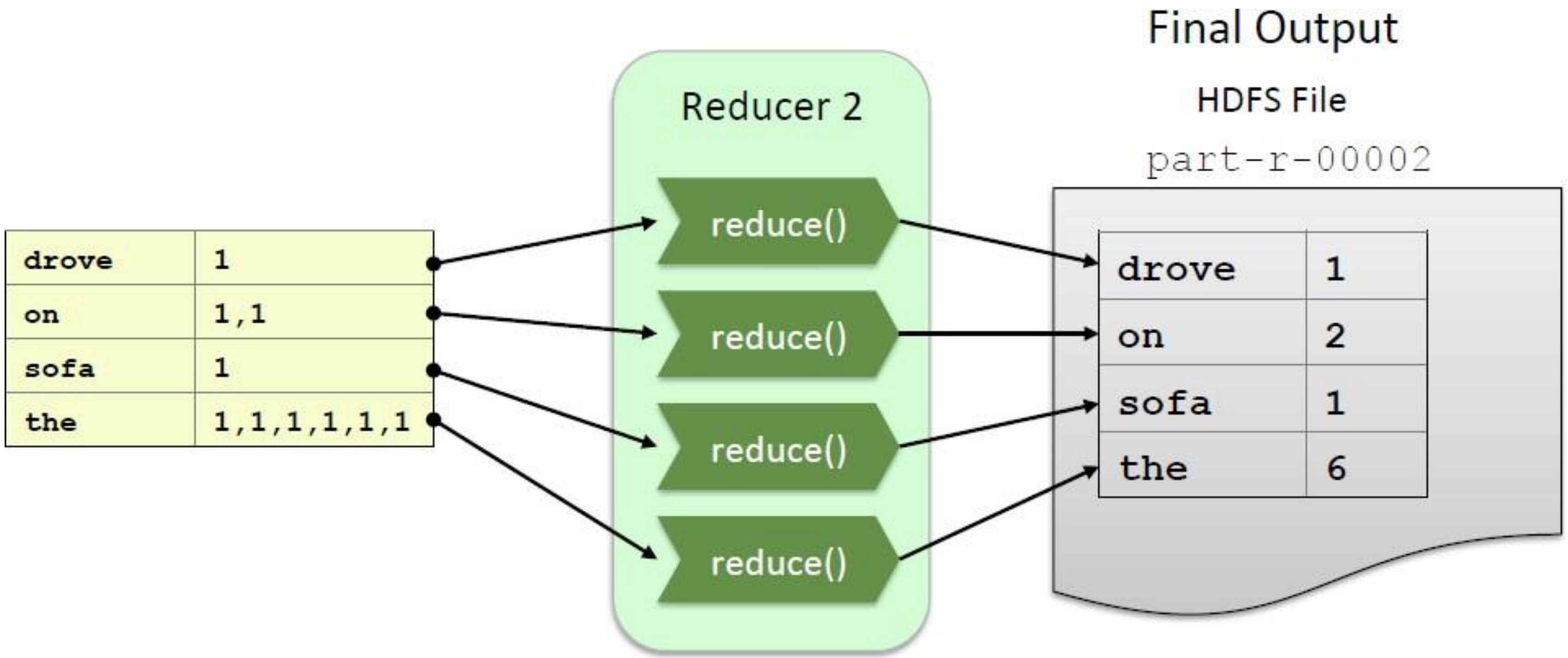
MapReduce: Map



MapReduce: Reduce



MapReduce: Reduce



Why do we care about counting words?

- Word count is challenging over massive amounts of data
  - Using a single compute node would be too time-consuming
  - Number of unique words could exceed available memory
- Statistics are often simple aggregate functions
  - Distributive in nature
  - e.g., max, min, sum, count
- Map-reduce breaks complex tasks down into smaller elements which can be executed in parallel
- Many common tasks are very similar to word count
  - e.g. log file analysis

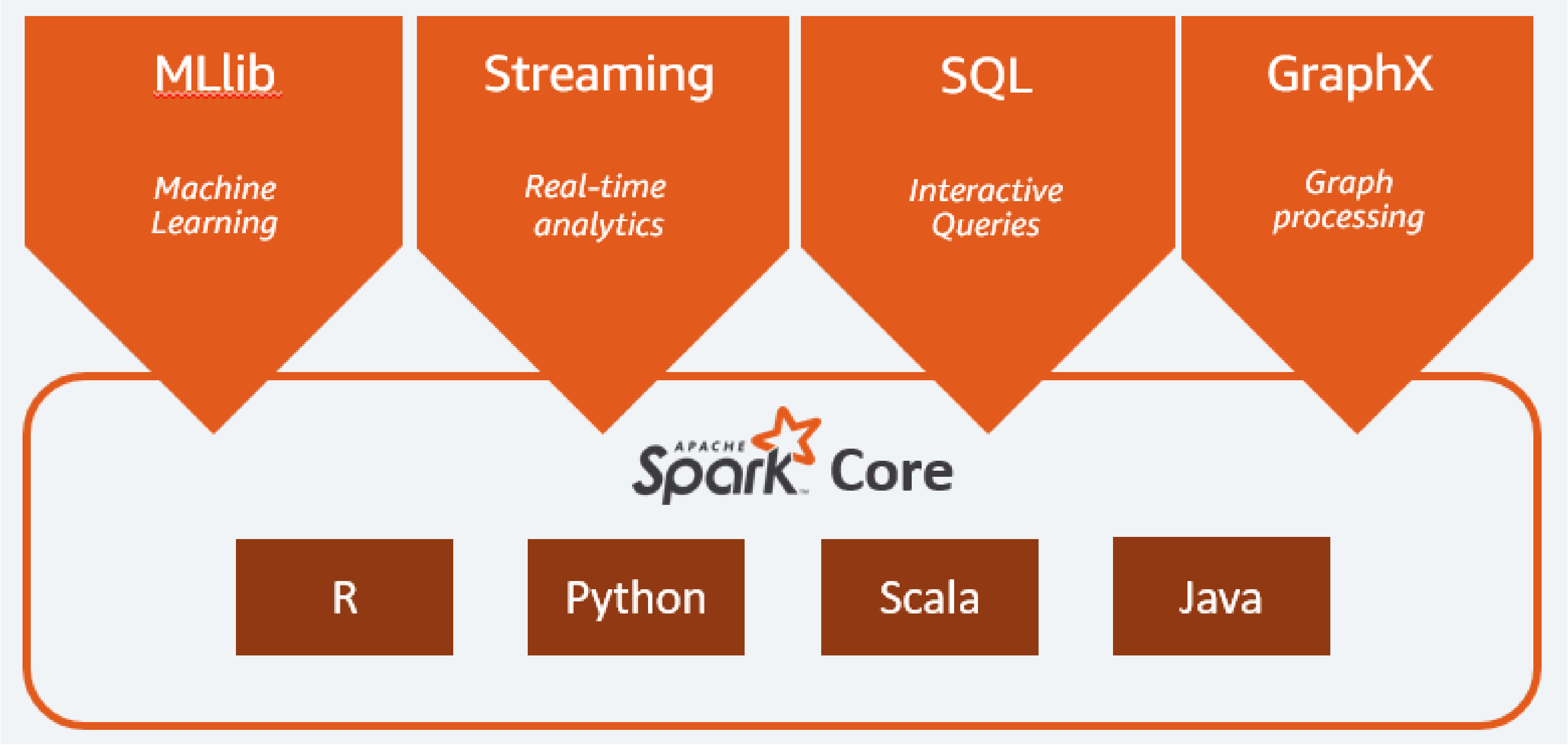
Problems with Hadoop MapReduce

- Hadoop MapReduce reads and writes from disk between computations, thus slowing down the processing speed if the data needs to be reused for multiple computations
- This means iterative algorithms in particular (including many machine learning algorithms) can be slow.
- Not all computations can be readily split into map and reduce phases, or it can be difficult to do so.
- Hadoop is better for batch processing, and is less suited to stream processing

Apache Spark

- Apache Spark is a fast and general engine for large-scale data processing
- Written in Scala
- Like Hadoop, Spark splits up large tasks across different nodes.
- However, it often tends to perform faster than Hadoop.
- In this module, we will be using Spark (not Hadoop). However, it’s worth being aware of Hadoop and some of the key differences between the two.

Apache Spark



https://aws.amazon.com/what-is/apache-spark/

Why Apache Spark?

- Speed
  - Run workloads up to 100x faster.
- Ease of Use
  - Write applications quickly in Java, Scala, Python, R, and SQL.
- Generality
  - Combine SQL, streaming, and complex analytics.
- Runs Everywhere
  - Spark runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud. It can access diverse data sources.

Hadoop MapReduce vs Spark

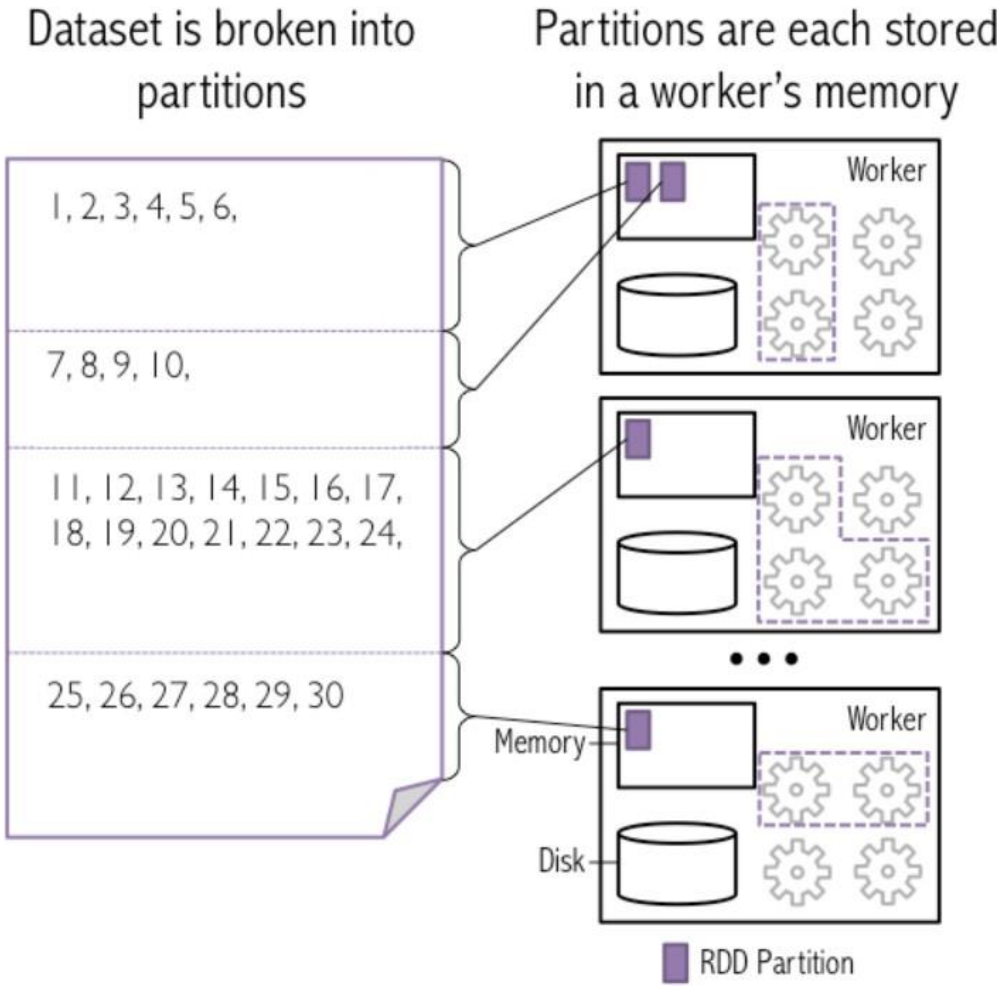
- Spark implements map-reduce with much greater flexibility
  - Map and reduce functions can be interspersed
  - Results can be stored in memory
  - Operations can easily be chained

RDD (Resilient Distributed Dataset)

- RDD (Resilient Distributed Dataset)
  - Resilient – if data in memory is lost, it can be recreated
  - Distributed – processed across the cluster
  - Dataset – initial data can come from a file or be created programmatically
- RDDs are the fundamental unit of data in Spark
- Most Spark programming consists of performing operations on RDDs
- We will be talking about RDDs in a lot more depth next week!

RDDs on a cluster

- Resilient Distributed Datasets
  - Data is partitioned across worker nodes
- Partitioning is done automatically by Spark
  - Optionally, you can control how many partitions are created
- Next lecture, we will provide a more thorough introduction to RDDs



Lazy Execution

- In Spark we analyse data by applying operations on RDDs
- These are either transformations or actions
  - **Transformations** – map, filter, reduceByKey etc. These define steps to process data but do not execute immediately
  - **Actions** - show, collect. These trigger the execution and returns the results
- Nothing is processed until an action is performed—Spark waits until it knows exactly what needs to be done.

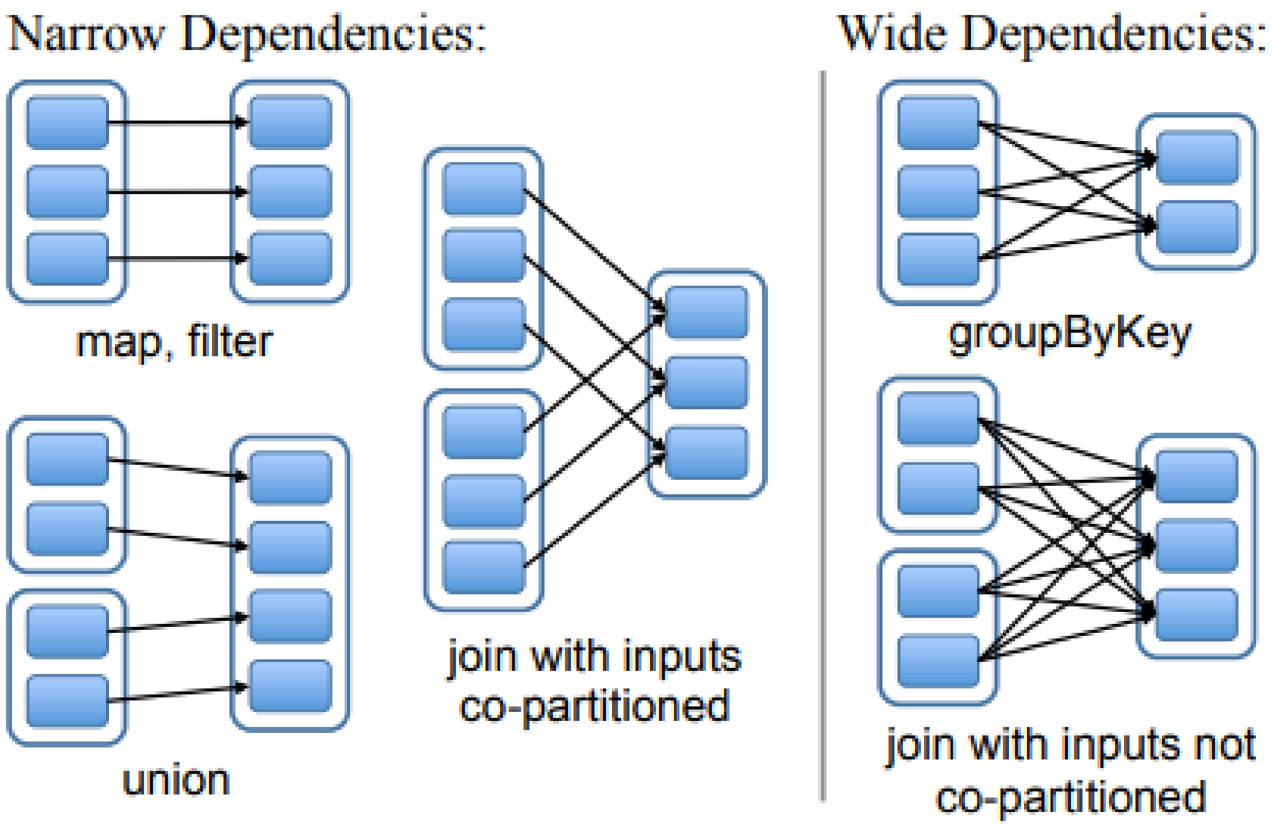
Directed Acyclic Graph

- Spark constructs a Directed Acyclic Graph (DAG) where the vertices are RDDs and the edges are operations to be applied to the RDD.

- **Directed** - edge points from one node to another, showing the order of operations.
- **Acyclic -** there are no cycles, meaning that no task can have a dependency on a task that follows it.

Narrow vs Wide Dependencies

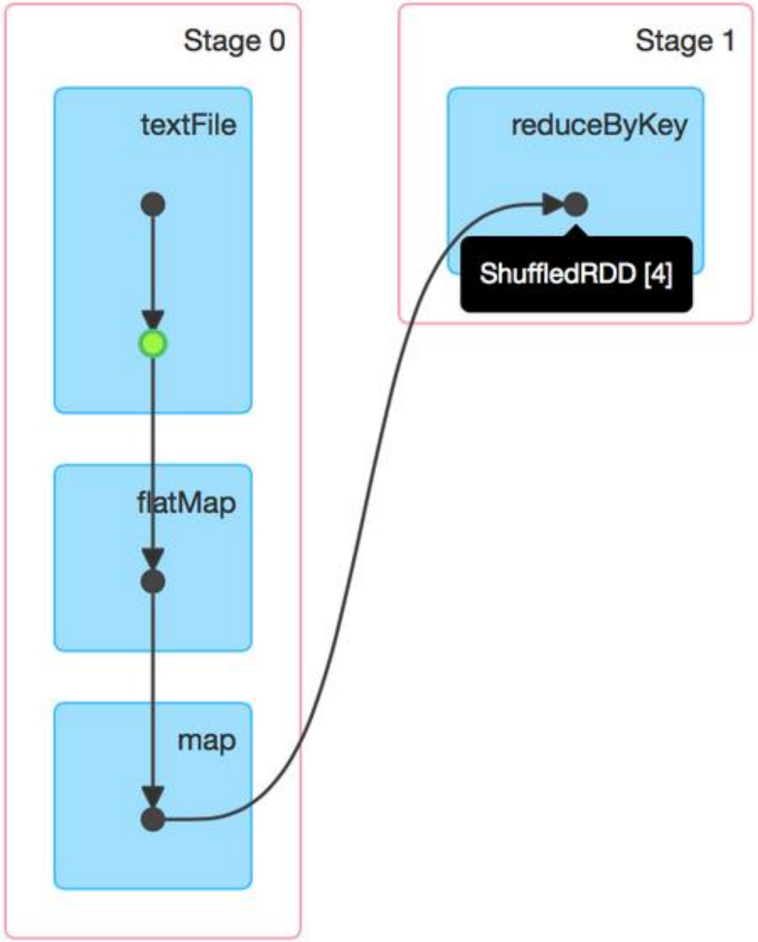
For narrow dependencies, each output partition can be computed from a single input partition



Tasks and Stages

This is used to split a job into *tasks* and *stages* – all the tasks in one stage can be executed in parallel, and data shuffles are required between stages.

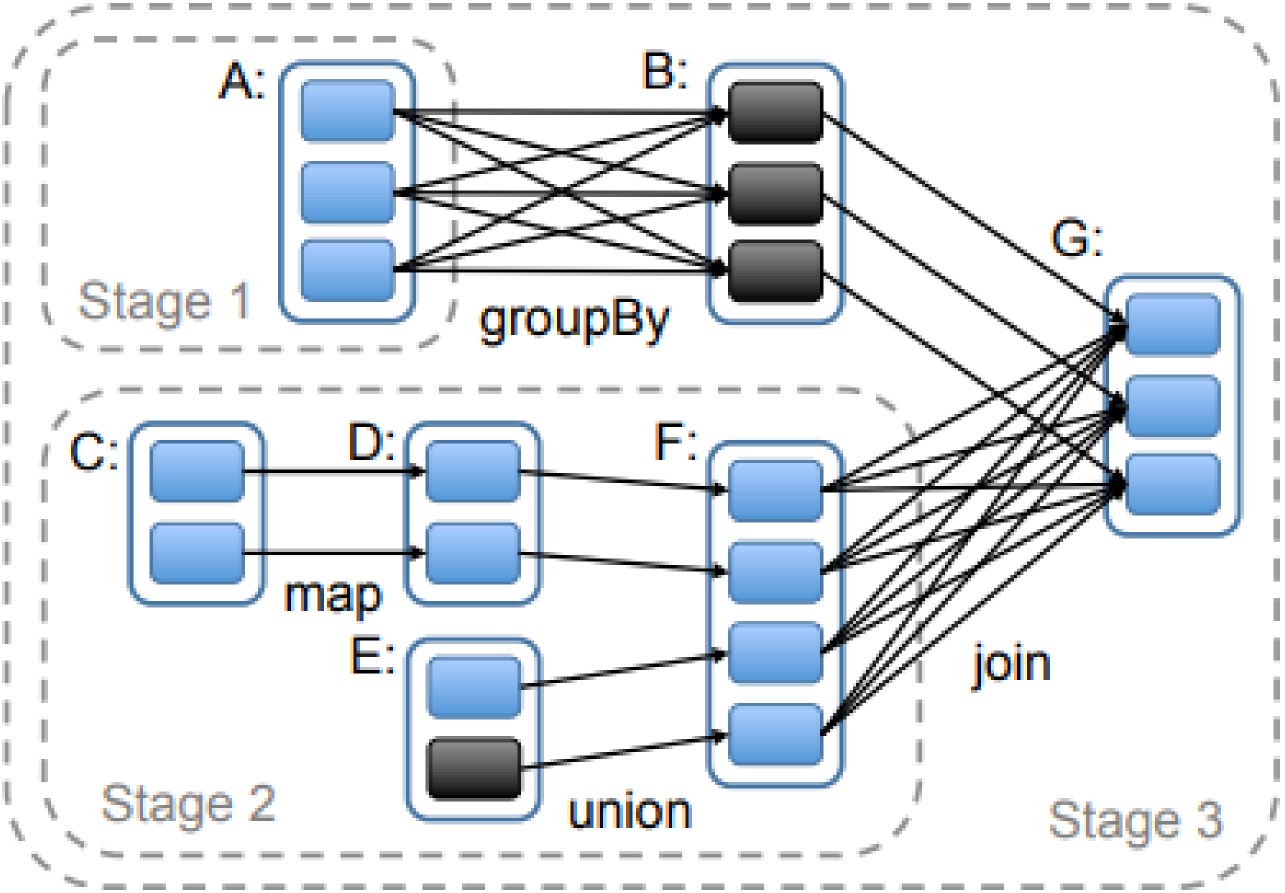
Back to counting words…



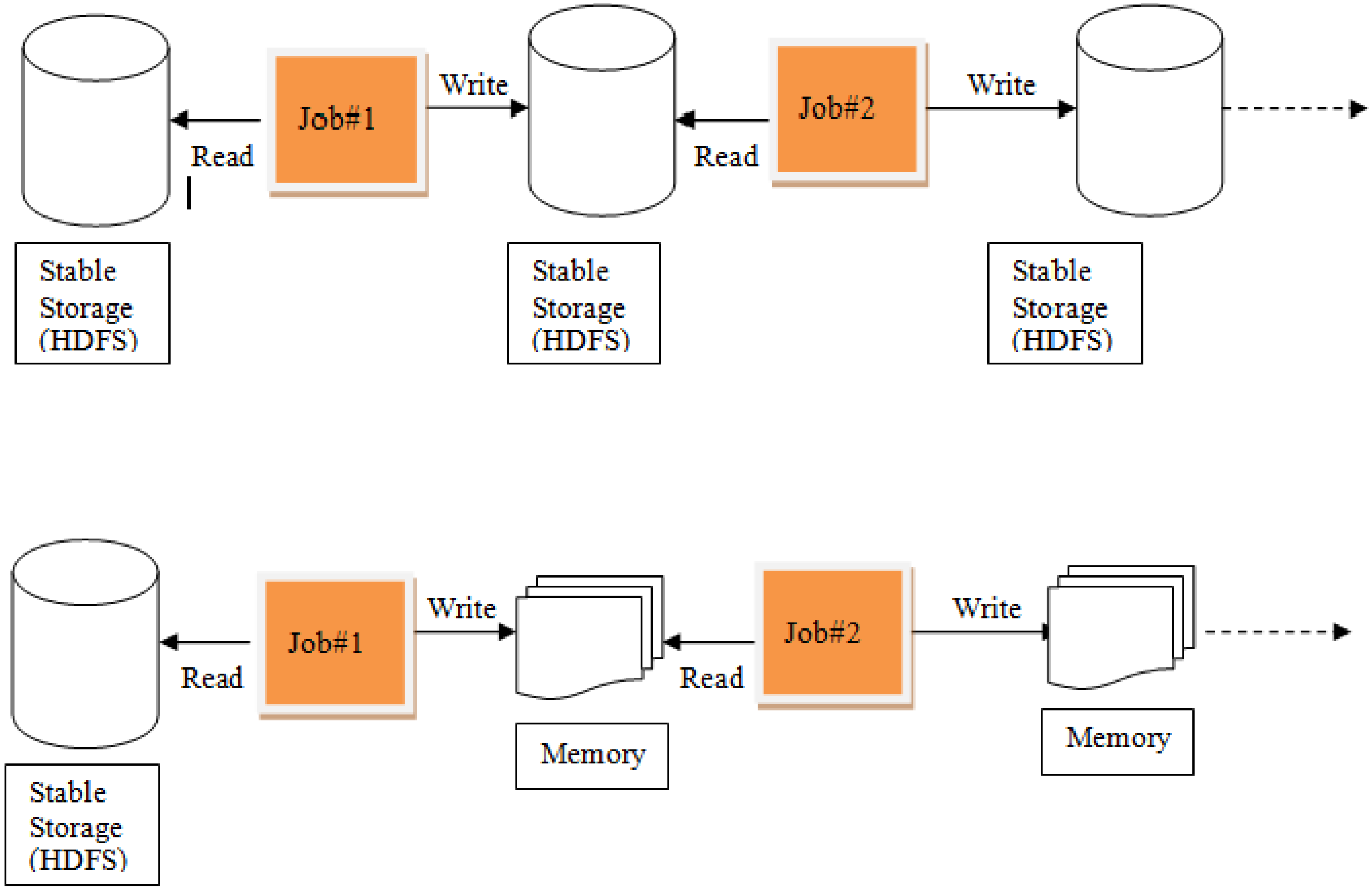
https://www.databricks.com/blog/2015/06/22/understanding-your-spark-application-through-visualization.html

Spark – Tasks vs Stages

To run an action on RDD G, we build stages at wide dependencies and pipeline narrow transformations inside each stage



Hadoop MapReduce vs Spark



Fault Tolerance

RDD Lineage (or Dependency Graph): Spark maintains a record of all transformations that were applied to create an RDD. This allows it to reconstruct lost data by reapplying the transformations to the original data source or intermediate RDDs.

Example Scenario: If a node processing a partition of an RDD crashes, Spark will identify the lost partition and rerun the necessary transformations to regenerate only that lost data from its dependencies.

Databricks

In this module we will be using **Databricks:**

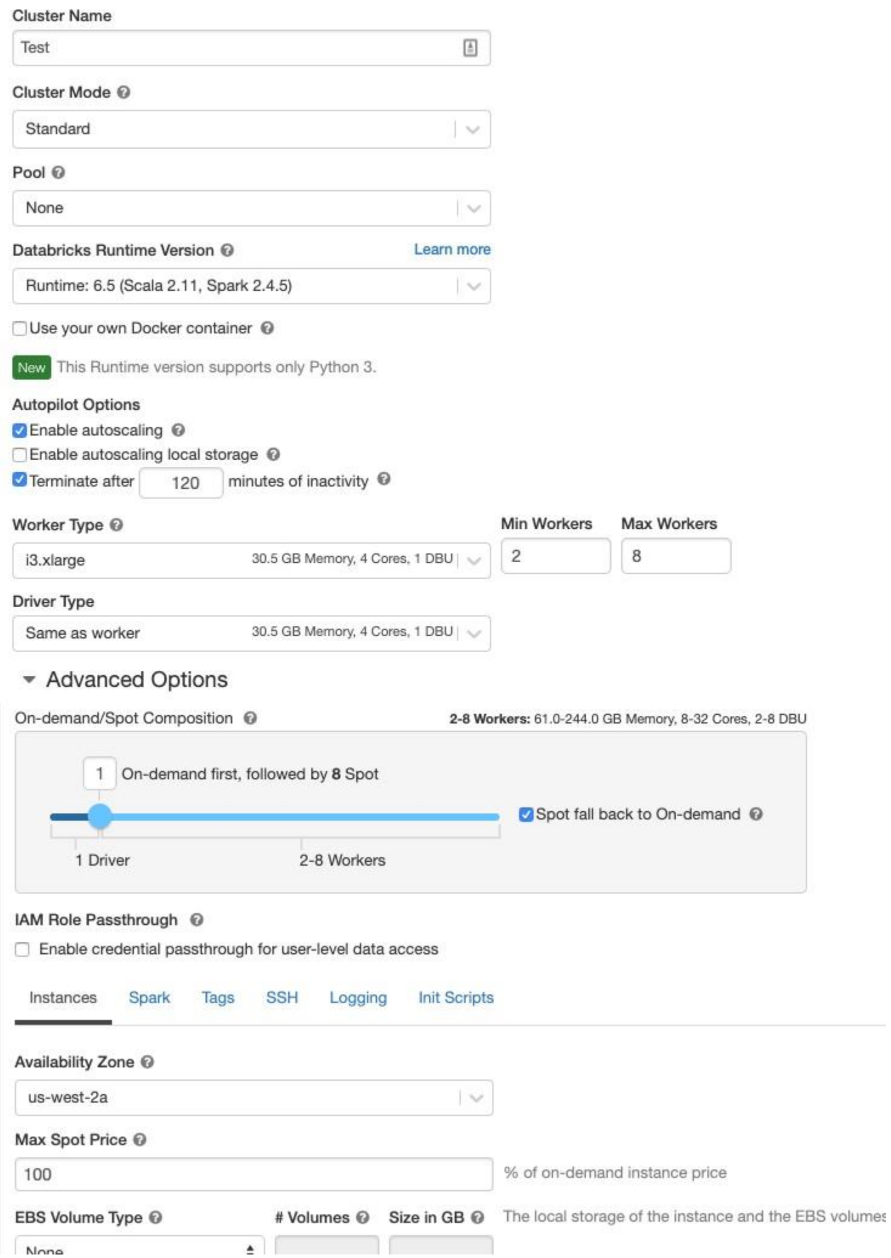
- The Databricks platform is a cloud platform which makes it easy to set up and manage a cluster
- Databricks is built around Apache Spark
- In Databricks, notebooks allow you to develop code and present results. They are the primary tool we use for creating data science workflows.
- Databricks notebooks provide real-time co-authoring in multiple languages, automatic versioning, and built-in data visualizations

Databricks File System (DBFS)

- Distributed file system mounted into Databricks workspace
- Persisted object storage – files stored in the DBFS are persisted when a cluster is terminated

Databricks cluster options

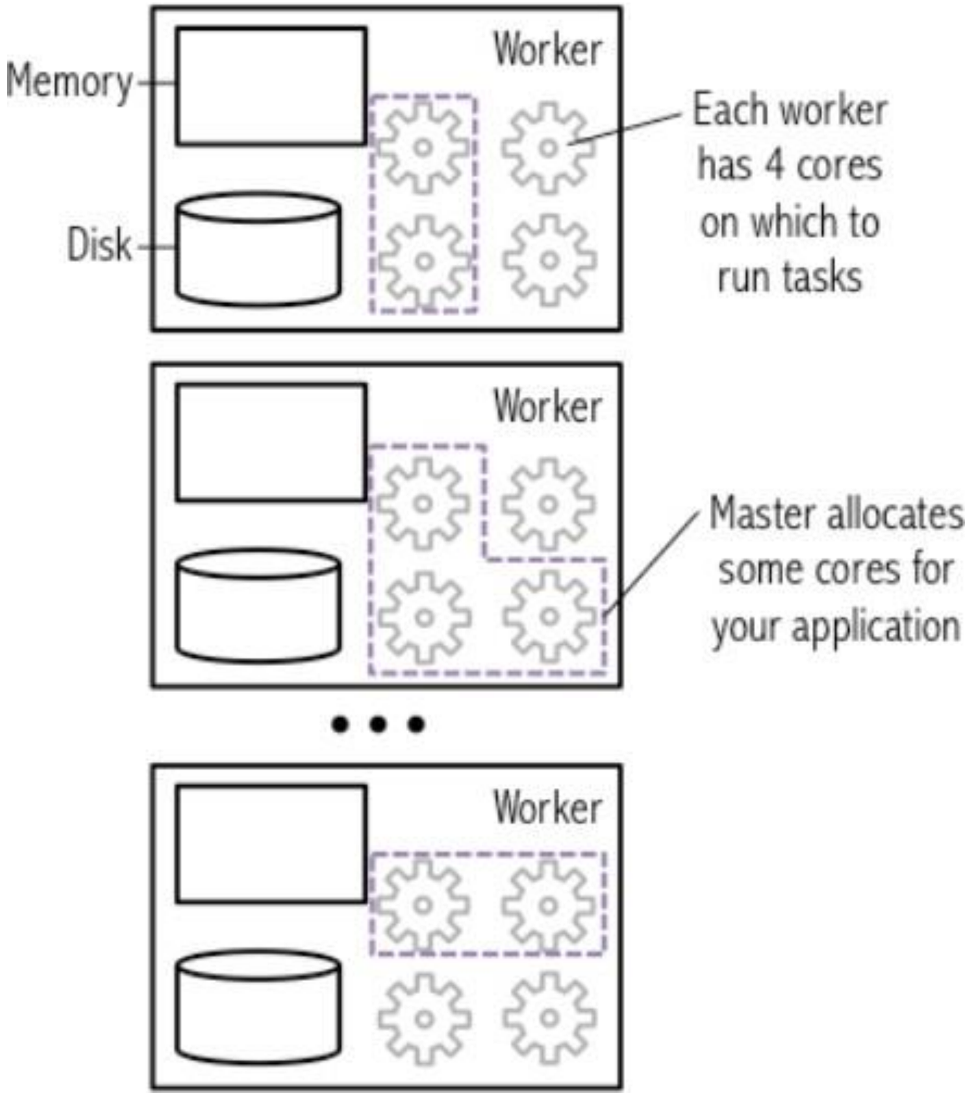
- Driver node:
  - State info of all notebooks attached to cluster.
  - Runs the master which coordinates with executors.
- Worker node:
  - Run the executors.
  - All distributed processing happens on workers.
  - On Databricks, one executor per worker node.



Spark cluster

- Driver and executors
- Driver jobs → tasks → executors.
  - Results from executors → driver

Note: In Databricks Community Edition there is no Worker, and the Master executes the entire code



Learning Outcomes: Recap

1. To introduce Apache Spark and Databricks
2. To give you an understanding of data storage options and distributed filesystems
3. To explain why we use distributed systems for processing big data and the challenges this entails
4. To provide an overview of the map-reduce programming model and its implementation in Hadoop MapReduce
5. To give you an understanding of why Apache Spark can often improve on Hadoop MapReduce